ETH zürich

spcl.inf.ethz.ch
@spcl_eth

D INFK

Nikoli Dryden (ndryden@ethz.ch)

Parallelism in Training Deep Neural Networks

DPHPC Guest Lecture

With contributions from Tal Ben-Nun, Torsten Hoefler, Dan Alistarh, and others at SPCL, LLNL, UIUC, IST Austria, and Tokyo Tech

SPCL

Overview

- What is deep learning?
- Some deep neural networks
- Parallelizing and distributing training
- Communication for training
- Applications

Some General References

- **Russell & Norvig, *Artificial Intelligence: A Modern Approach***
- **Goodfellow, Bengio, & Courville, *Deep Learning***
  - Freely available online: http://www.deeplearningbook.org/
- **Ben-Nun & Hoefler, *Demystifying Parallel and Distributed Deep Learning***
  - https://arxiv.org/abs/1802.09941
- **Many slides adapted from Tal Ben-Nun, Torsten Hoefler, Svetlana Lazebnik, and prior talks**

What is Deep Learning good for?

What is Deep Learning good for?

Digit Recognition

1989

2012

What is Deep Learning good for?

Digit Recognition

1989

2012

2017

What is Deep Learning good for?

Digit Recognition

1989

2012

2017

What is Deep Learning good for?

Digit Recognition

Object Classification

1989

2012

2017

What is Deep Learning good for?

Digit Recognition

Object Classification

Image Captioning
GANs

1989

2012

2017

What is Deep Learning good for?

Digit Recognition

Object Classification

Segmentation

Image Captioning

GANs

1989

2012

2017

What is Deep Learning good for?

Digit Recognition

Object Classification

Segmentation

Image Captioning

GANs

1989

2012

2017

What is Deep Learning good for?

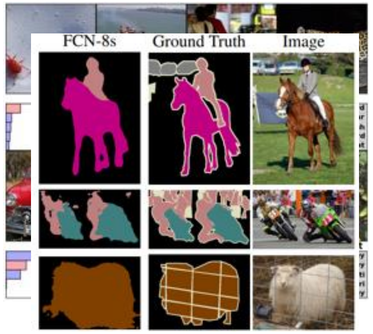
Digit Recognition

Object Classification

Segmentation

Image Captioning

GANs



1989

2012

2017

What is Deep Learning good for?

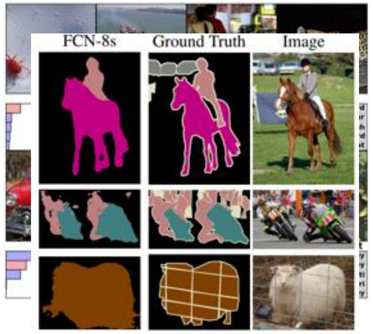
Digit Recognition

Object Classification

Segmentation

Image Captioning

GANs



1989

2012 2013

2017

What is Deep Learning good for?

Digit Recognition

Object Classification

Segmentation

Image Captioning

GANs

1989

2012 2013

2017

What is Deep Learning good for?

Digit Recognition

Object Classification

Segmentation

Image Captioning

GANs

1989

2012 2013

2017

What is Deep Learning good for?

Digit Recognition

Object Classification

Segmentation

Image Captioning

GANs

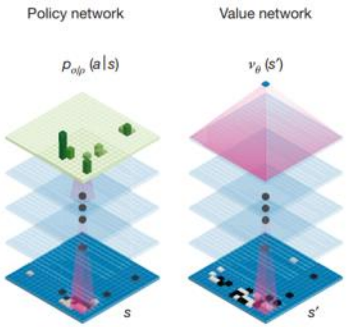
1989 2012 2013 2014 2017

What is Deep Learning good for?

Digit Recognition

Object Classification
Segmentation
Image Captioning
GANs

Gameplay AI
Translation



1989

2012 2013 2014

2017

What is Deep Learning good for?

Digit Recognition

Object Classification

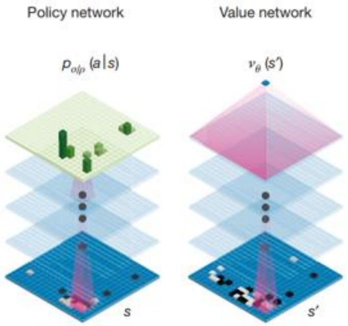
Segmentation

Image Captioning

GANs

Gameplay AI

Translation



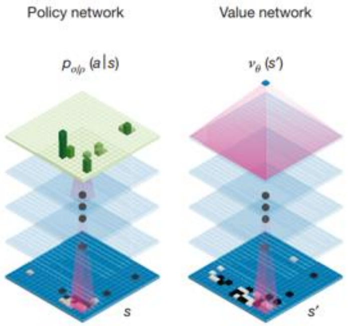
1989 2012 2013 2014 2016 2017

What is Deep Learning good for?

Digit Recognition

Object Classification
Segmentation
Image Captioning
GANs

Gameplay AI
Translation



1989 2012 2013 2014 2016 2017

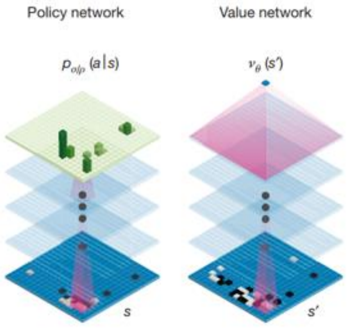
What is Deep Learning good for?

Digit Recognition

Object Classification
Segmentation
Image Captioning
GANs

Gameplay AI
Translation

Neural Computers



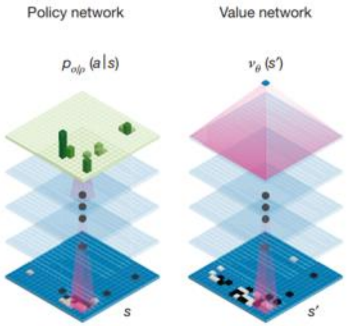
1989 2012 2013 2014 2016 2017

What is Deep Learning good for?

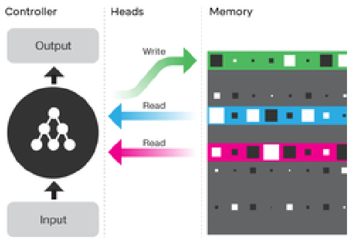
Digit Recognition

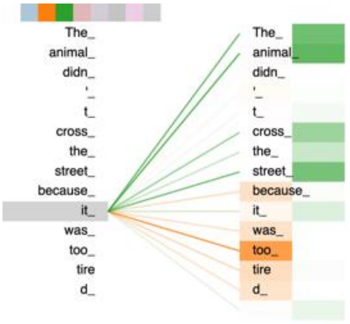
Object Classification
Segmentation
Image Captioning
GANs

Gameplay AI
Translation



Neural Computers





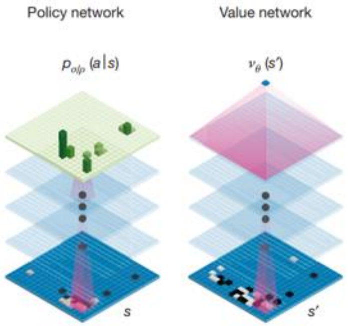
1989 2012 2013 2014 2016 2017

What is Deep Learning good for?

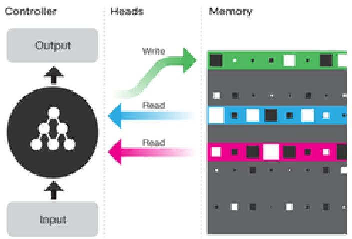
Digit Recognition

Object Classification
Segmentation
Image Captioning
GANs

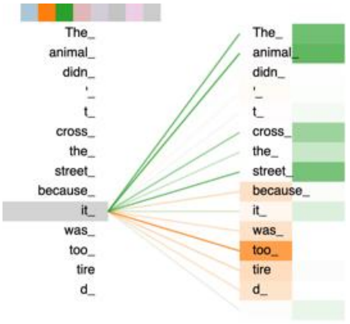
Gameplay AI
Translation



Neural Computers



Language Models



1989
2012 2013 2014
2016
2017

What is Deep Learning good for?

Digit Recognition

Object Classification

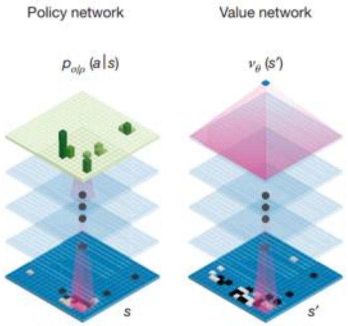
Segmentation

Image Captioning

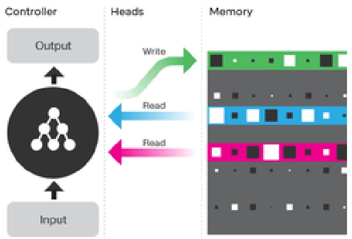
GANs

Gameplay AI

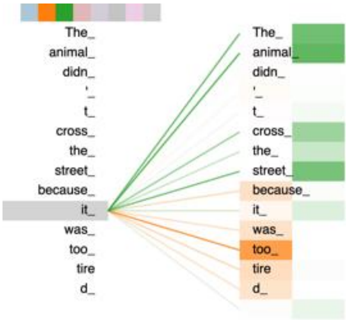
Translation



Neural Computers



Language Models



1989

2012 2013 2014

2016

2017

What is Deep Learning good for?

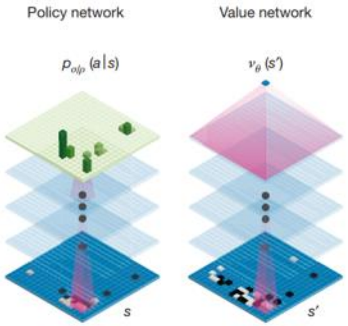
Digit Recognition

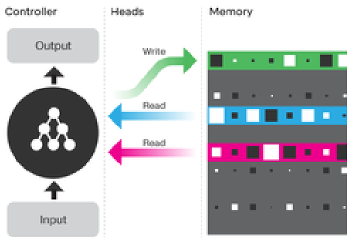
Object Classification
Segmentation
Image Captioning
GANs

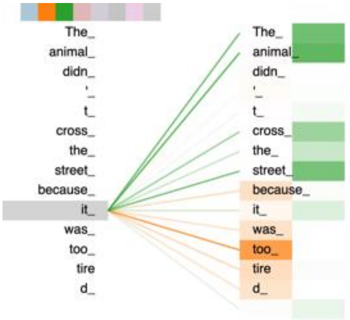
Gameplay AI
Translation

Neural Computers

Language Models







1989 2012 2013 2014 2016 2017 2019

What is Deep Learning good for?

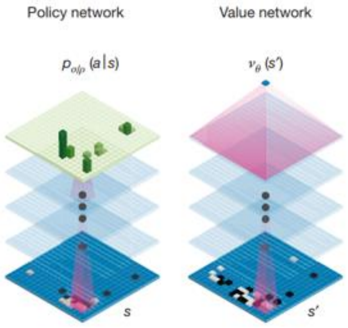
Digit Recognition

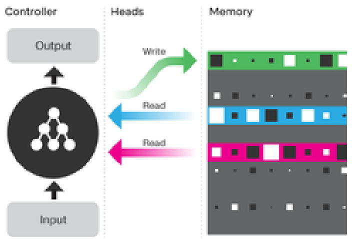
Object Classification
Segmentation
Image Captioning
GANs

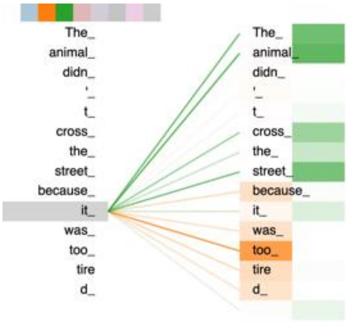
Gameplay AI
Translation

Neural Computers

Language Models







1989 2012 2013 2014 2016 2017 2018 2019

What is Deep Learning good for?

Digit Recognition

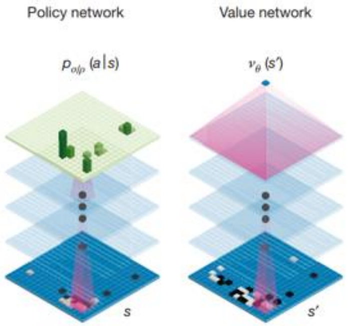
Object Classification
Segmentation
Image Captioning
GANs

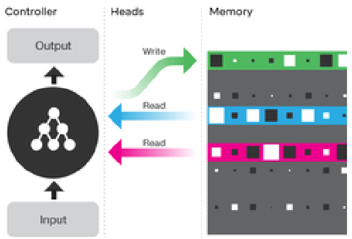
Gameplay AI
Translation

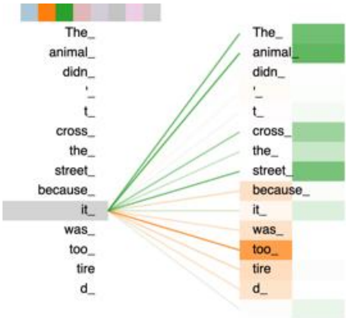
Neural Computers

Language Models

Towards
Real Physics
RTS







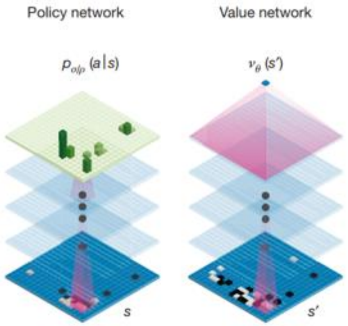
1989 2012 2013 2014 2016 2017 2018 2019

What is Deep Learning good for?

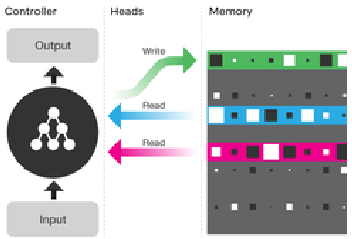
Digit Recognition

Object Classification
Segmentation
Image Captioning
GANs

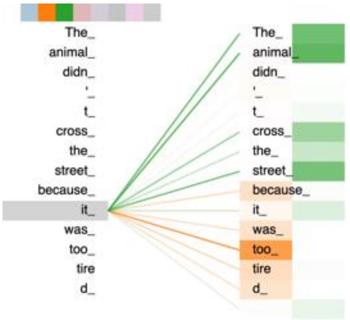
Gameplay AI
Translation



Neural Computers



Language Models



Towards
Real Physics
RTS

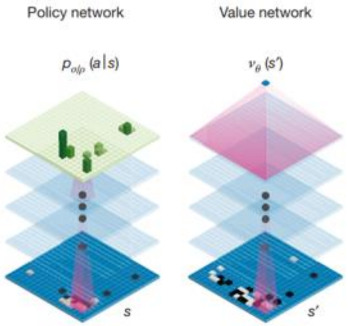
1989 2012 2013 2014 2016 2017 2018 2019

What is Deep Learning good for?

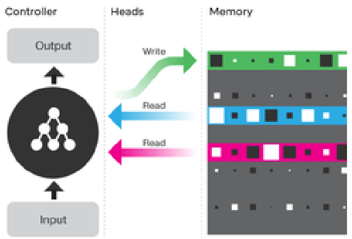
Digit Recognition

Object Classification
Segmentation
Image Captioning
GANs

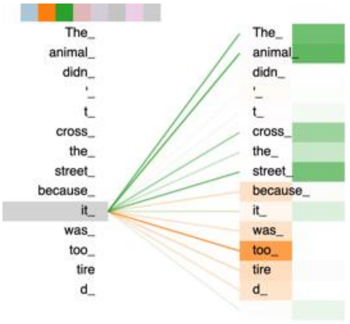
Gameplay AI
Translation



Neural Computers



Language Models



Towards
Real Physics
RTS

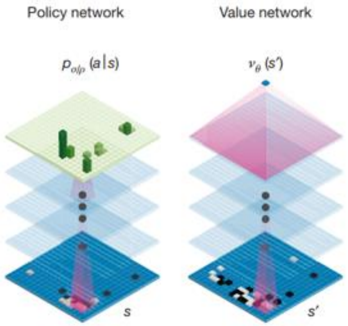
1989 2012 2013 2014 2016 2017 2018 2019

What is Deep Learning good for?

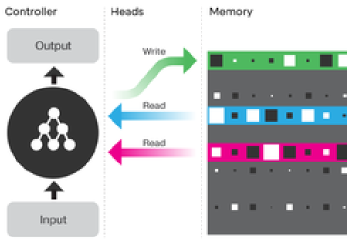
Digit Recognition

Object Classification
Segmentation
Image Captioning
GANs

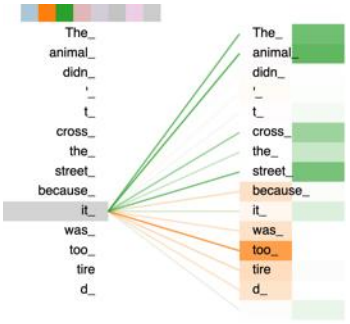
Gameplay AI
Translation



Neural Computers



Language Models

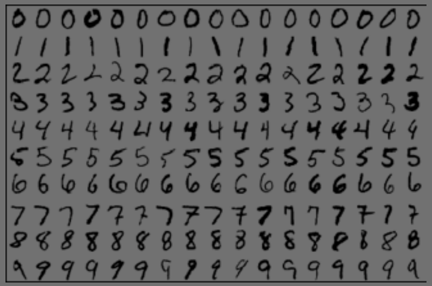


Towards
Real Physics
RTS

1989 2012 2013 2014 2016 2017 2018 2019

What is Deep Learning good for?

Digit Recognition

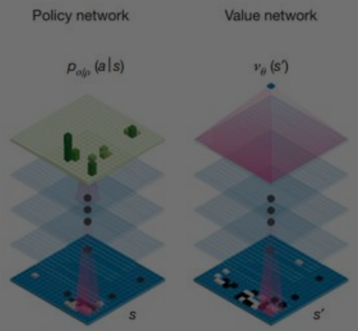


1989

Object Classification
Segmentation
Image Captioning
GANs

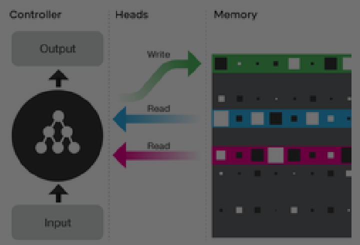
2012 2013 2014

Gameplay AI
Translation



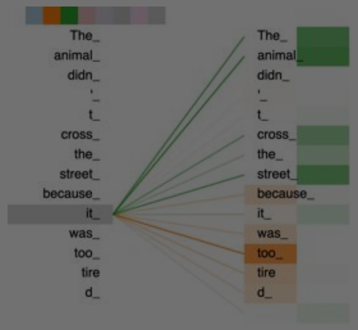
2016

Neural Computers



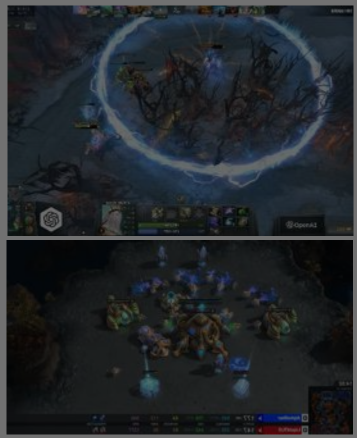
2017

Language Models



2018

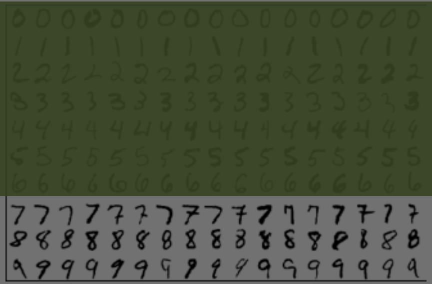
Towards
Real Physics
RTS



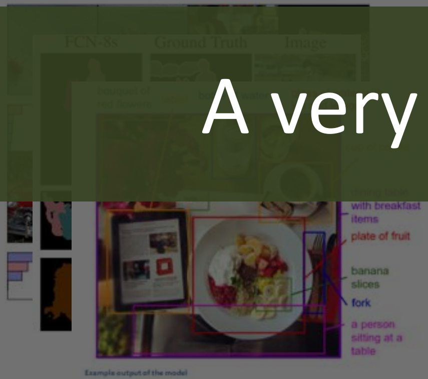
2019

What is Deep Learning good for?

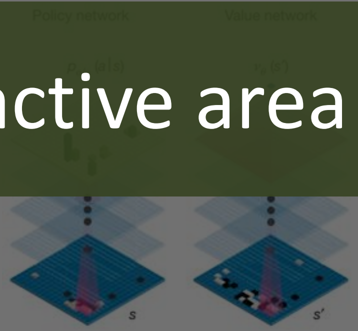
Digit Recognition



Object Classification
Segmentation
Image Captioning
GANs

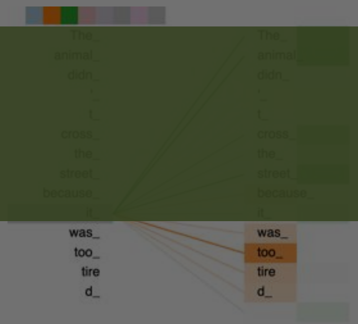


Gameplay AI
Translation



Neural Computers

Language Models



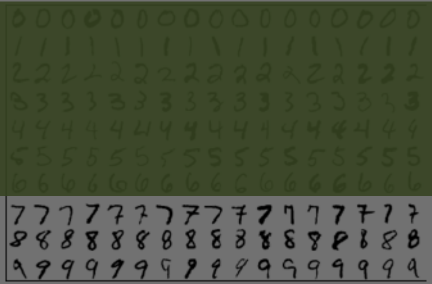
Towards
Real Physics
RTS

A very active area of research!

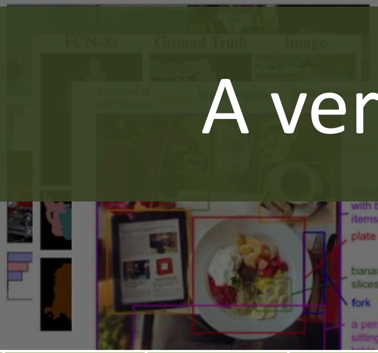
1989 | 2012 2013 2014 | 2016 | 2017 | 2018 | 2019

What is Deep Learning good for?

Digit Recognition



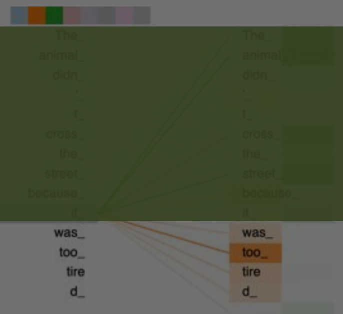
Object Classification
Segmentation
Image Captioning
GANs



Gameplay AI
Translation

Neural Computers

Language Models



Towards
Real Physics
RTS

A very active area of research!

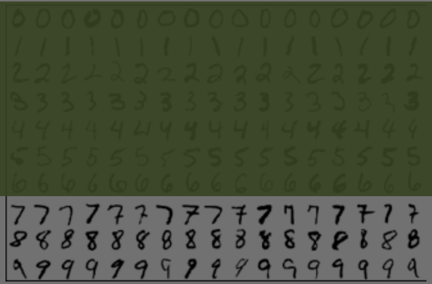
1989

2019

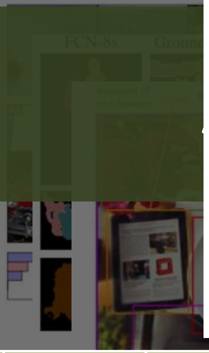
Subject	2009	2010	2011	2012	2013	2014	2015	2016	2017	2018	2019
cs.AI	380	479	789	1082	1768	1028	1106	1938	2820	4263	4371
cs.CV	148	286	385	577	852	1349	2262	3631	5704	8599	10353
cs.LG	231	333	469	1222	1418	1742	2485	3564	5225	10472	17267
stat.ML	164	256	439	1131	1203	1360	1827	2628	4021	8376	11551
Total	923	1354	2082	4012	5241	5479	7680	11761	17770	31710	43542

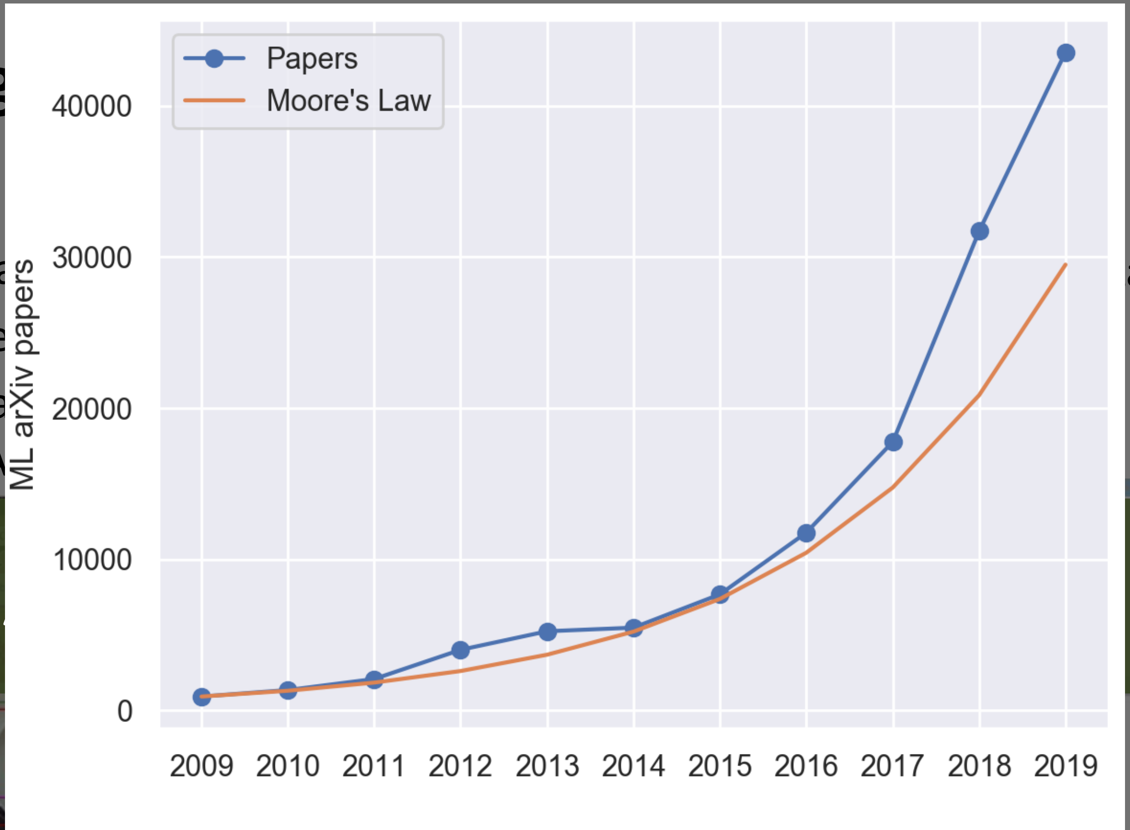
What is Deep Learning

Digit Recognition

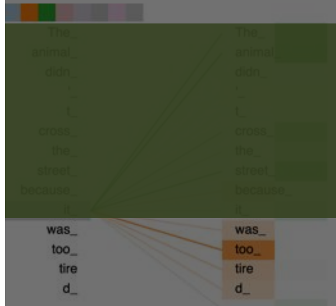


Object Cla
Segme
Image Ca
GA





anguage Models



Towards
Real Physics
RTS

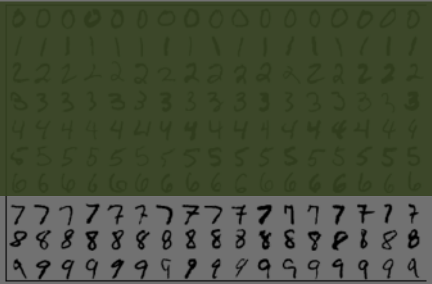
Subject	2009	2010	2011	2012	2013	2014	2015	2016	2017	2018	2019
cs.AI	380	479	789	1082	1768	1028	1106	1938	2820	4263	4371
cs.CV	148	286	385	577	852	1349	2262	3631	5704	8599	10353
cs.LG	231	333	469	1222	1418	1742	2485	3564	5225	10472	17267
stat.ML	164	256	439	1131	1203	1360	1827	2628	4021	8376	11551
Total	923	1354	2082	4012	5241	5479	7680	11761	17770	31710	43542

1989

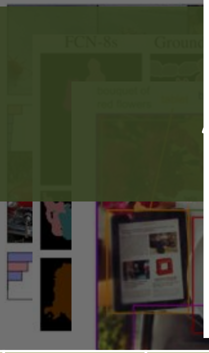
2019

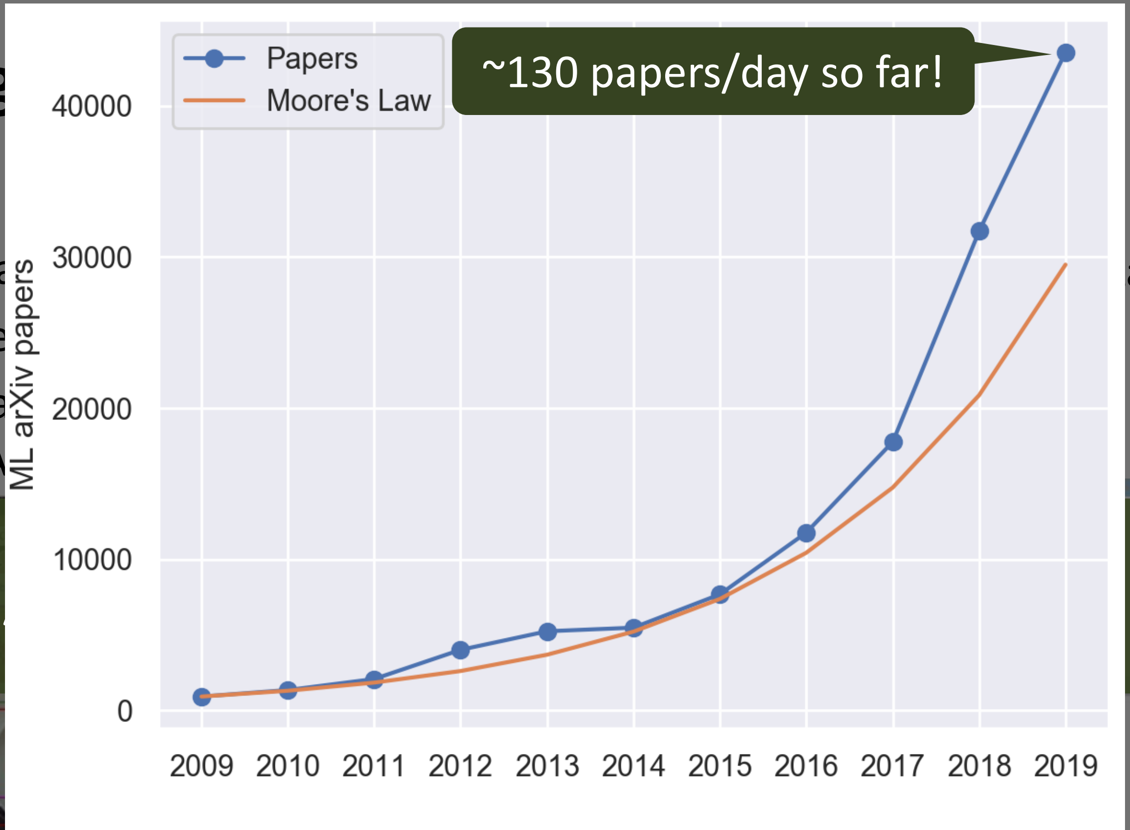
What is Deep Learning

Digit Recognition

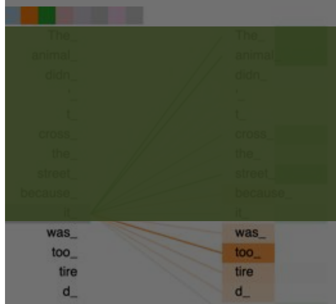


Object Cla
Segme
Image Ca
GA





anguage Models



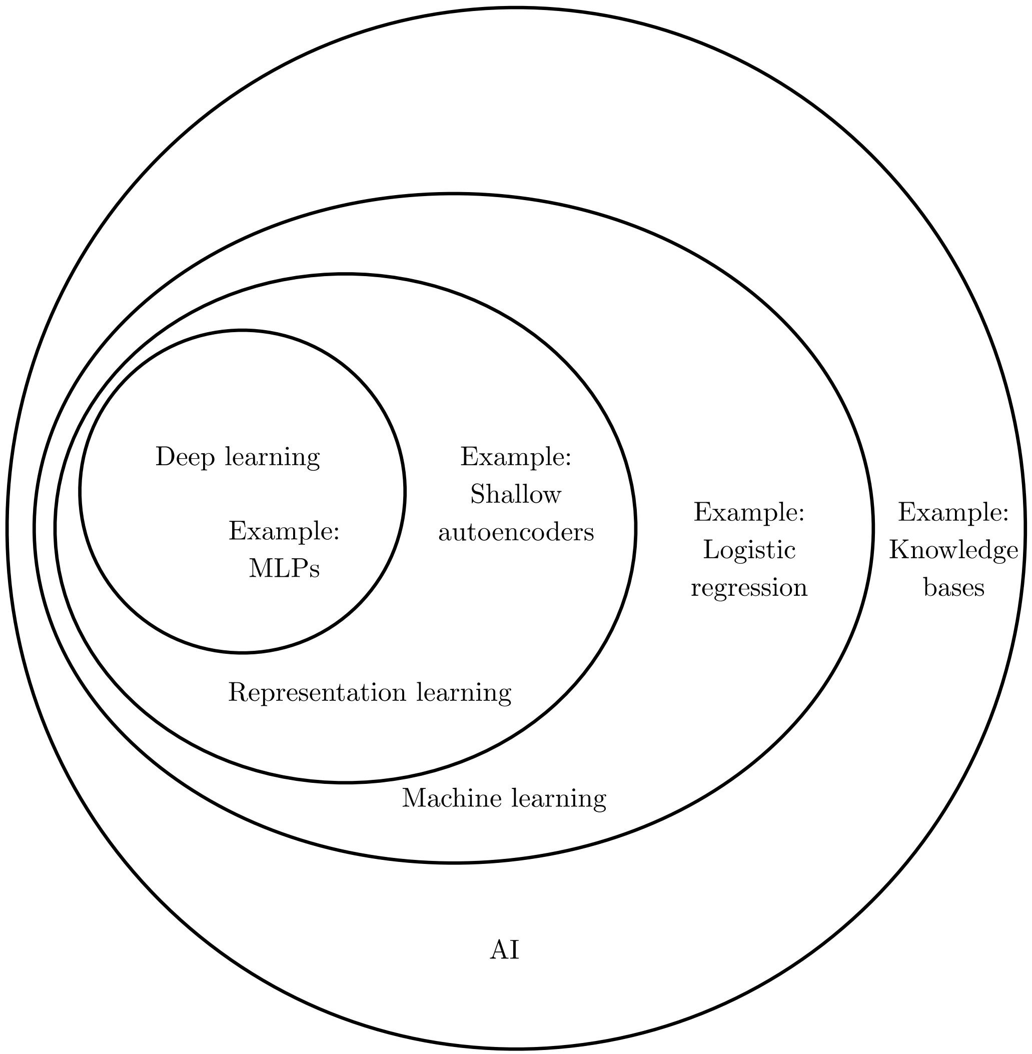
Towards
Real Physics
RTS

Subject	2009	2010	2011	2012	2013	2014	2015	2016	2017	2018	2019
cs.AI	380	479	789	1082	1768	1028	1106	1938	2820	4263	4371
cs.CV	148	286	385	577	852	1349	2262	3631	5704	8599	10353
cs.LG	231	333	469	1222	1418	1742	2485	3564	5225	10472	17267
stat.ML	164	256	439	1131	1203	1360	1827	2628	4021	8376	11551
Total	923	1354	2082	4012	5241	5479	7680	11761	17770	31710	43542

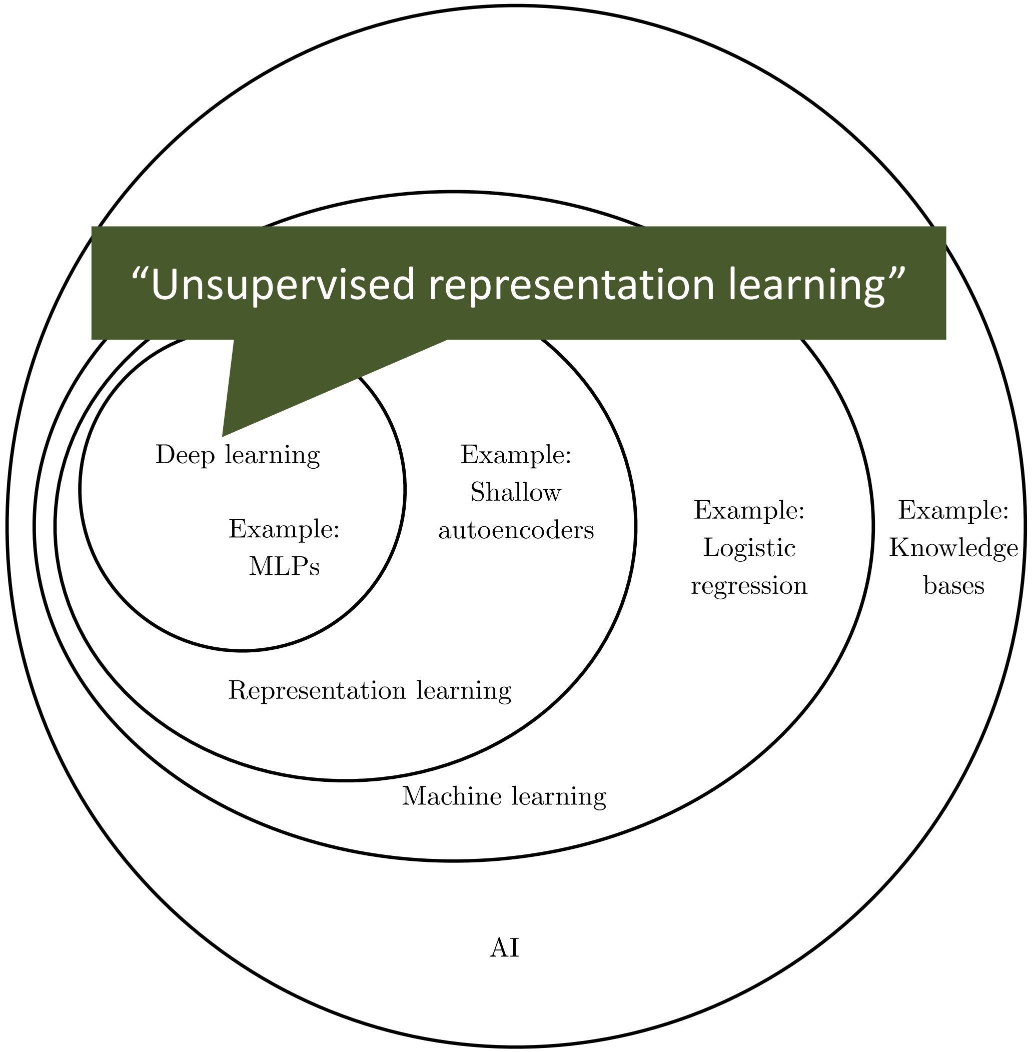
1989

2019

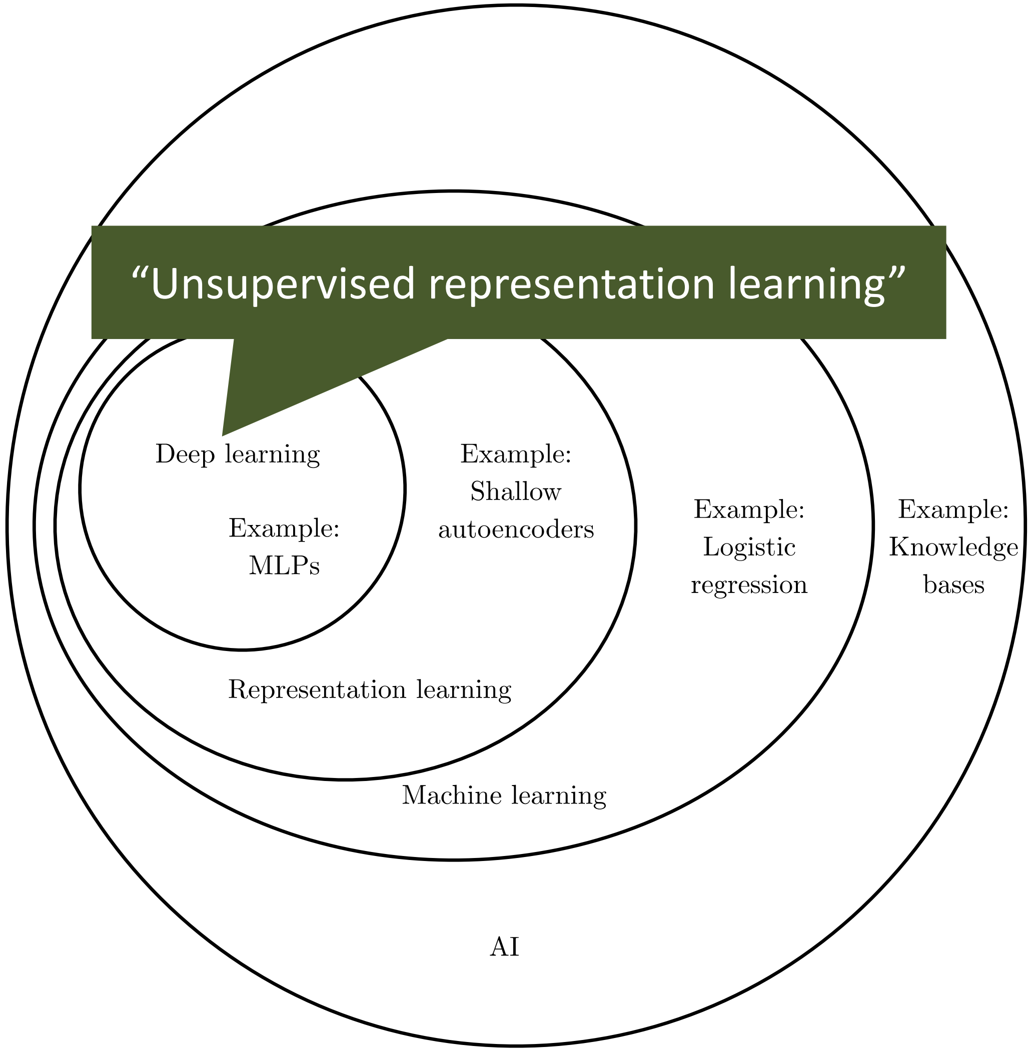
Classes of AI Problems



Classes of AI Problems

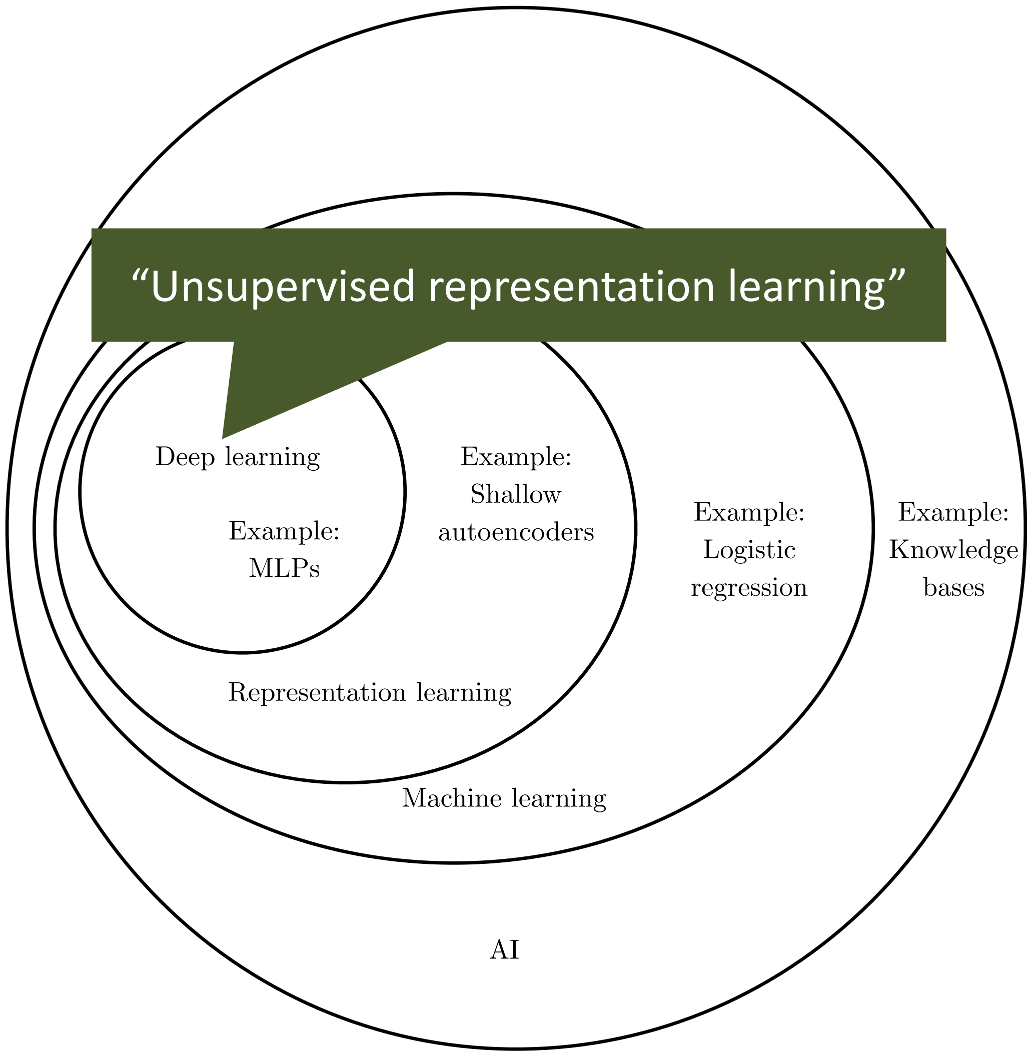


Classes of AI Problems



- **Supervised learning**
  - Learn mapping from labeled inputs
    $$\mathrm{argmin}_{f\in\mathcal{H}}\ \mathbb{E}_{x,y\sim\mathcal{D}}[\ell(f(x),y)]$$
- **Unsupervised learning**
  - Learn patterns in inputs
    $$\mathrm{argmin}_{f\in\mathcal{H}}\ \mathbb{E}_{x\sim\mathcal{D}}\left[\ell\left(f(x)\right)\right]$$
- **Reinforcement learning**
  - Learn policy to maximize reward
    $$\mathrm{argmax}_{\pi\in\mathcal{H}}\ \mathbb{E}_{O\sim\Omega}[R(\pi,O)]$$
- **Many others…**

Classes of AI Problems



- **Supervised learning**
  - Learn mapping from labeled inputs

$$\text{argmin}_{f\in\mathcal{H}}\ \mathbb{E}_{x,y\sim\mathcal{D}}[\ell(f(x),y)]$$

- **Unsupervised learning**
  - Learn patterns in inputs

$$\text{argmin}_{f\in\mathcal{H}}\ \mathbb{E}_{x\sim\mathcal{D}}[\ell(f(x))]$$

- **Reinforcement learning**
  - Learn policy to maximize reward

$$\text{argmax}_{\pi\in\mathcal{H}}\ \mathbb{E}_{O\sim\Omega}[R(\pi,O)]$$

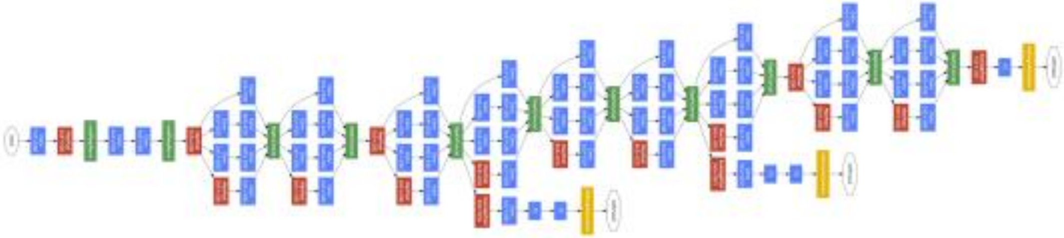
- **Many others…**

Deep

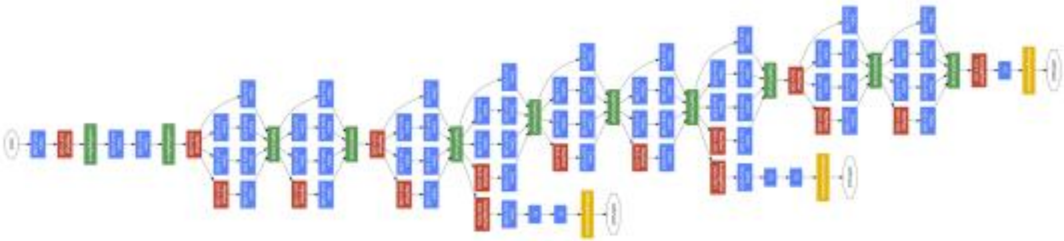
A brief theory of supervised deep learning (mini-batch SGD)

A brief theory of supervised deep learning (mini-batch SGD)

A brief theory of supervised deep learning (mini-batch SGD)

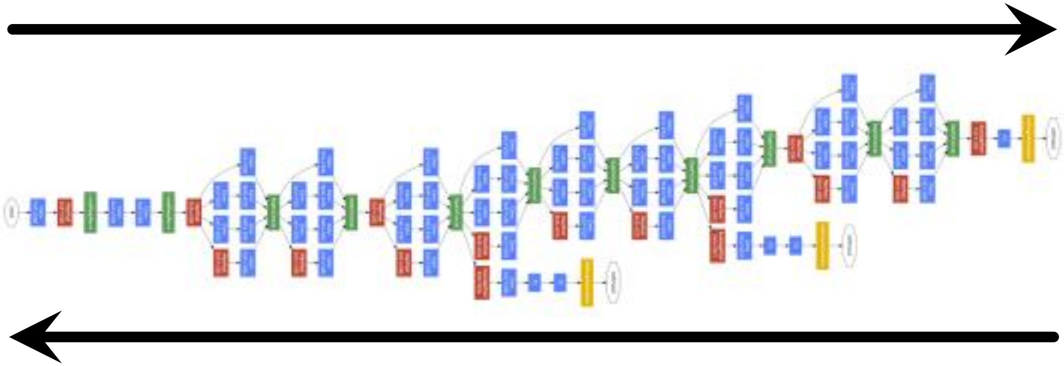


A brief theory of supervised deep learning (mini-batch SGD)



labeled samples $x \in X \subset \mathcal{D}$

A brief theory of supervised deep learning (mini-batch SGD)

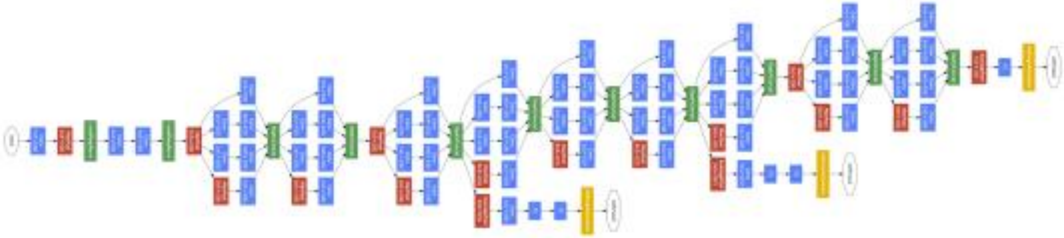


Cat	1.00
Dog	0.00
Airplane	0.00
Horse	0.00
Banana	0.00
Truck	0.00

labeled samples $x \in X \subset \mathcal{D}$

A brief theory of supervised deep learning (mini-batch SGD)

$f(x)$



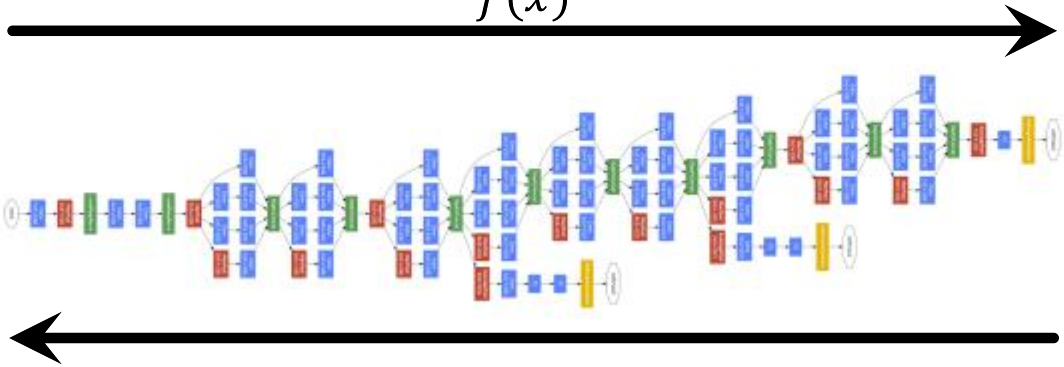
Cat	0.54	Cat	1.00
Dog	0.28	Dog	0.00
Airplane	0.07	Airplane	0.00
Horse	0.33	Horse	0.00
Banana	0.02	Banana	0.00
Truck	0.02	Truck	0.00

labeled samples $x \in X \subset \mathcal{D}$

label domain Y

A brief theory of supervised deep learning (mini-batch SGD)

$f(x)$

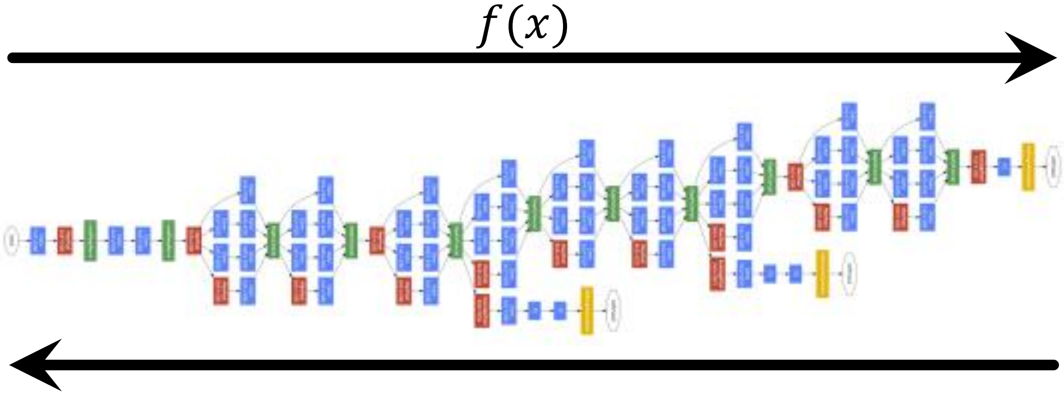


Cat	0.54	Cat	1.00
Dog	0.28	Dog	0.00
Airplane	0.07	Airplane	0.00
Horse	0.33	Horse	0.00
Banana	0.02	Banana	0.00
Truck	0.02	Truck	0.00

labeled samples $x \in X \subset \mathcal{D}$

label domain Y

A brief theory of supervised deep learning (mini-batch SGD)



Cat	0.54
Dog	0.28
Airplane	0.07
Horse	0.33
Banana	0.02
Truck	0.02

Cat	1.00
Dog	0.00
Airplane	0.00
Horse	0.00
Banana	0.00
Truck	0.00

labeled samples $x \in X \subset \mathcal{D}$

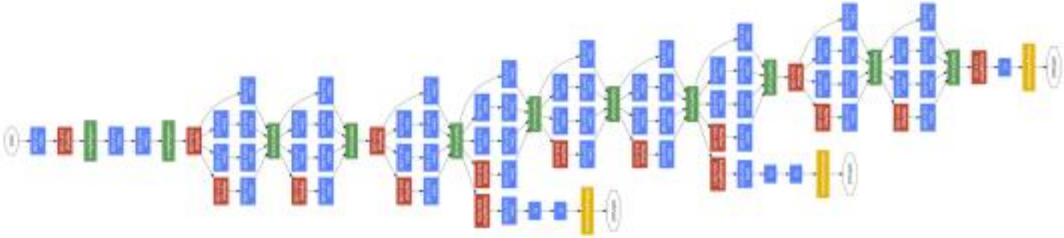
label domain Y

true label $l(x)$

$$f(x): X \rightarrow Y$$

A brief theory of supervised deep learning (mini-batch SGD)

$f(x)$



Cat	0.54
Dog	0.28
Airplane	0.07
Horse	0.33
Banana	0.02
Truck	0.02

Cat	1.00
Dog	0.00
Airplane	0.00
Horse	0.00
Banana	0.00
Truck	0.00

labeled samples $x \in X \subset \mathcal{D}$

label domain Y

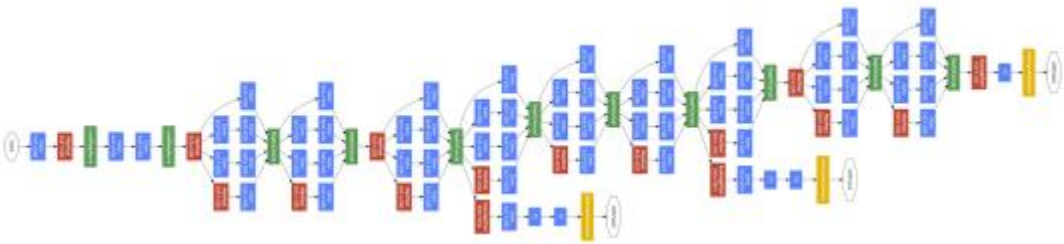
true label $l(x)$

$f(x): X \rightarrow Y$

A brief theory of supervised deep learning (mini-batch SGD)

labeled samples $x \in X \subset \mathcal{D}$

$f(x)$



Cat	0.54
Dog	0.28
Airplane	0.07
Horse	0.33
Banana	0.02
Truck	0.02

label domain Y

Cat	1.00
Dog	0.00
Airplane	0.00
Horse	0.00
Banana	0.00
Truck	0.00

true label $l(x)$

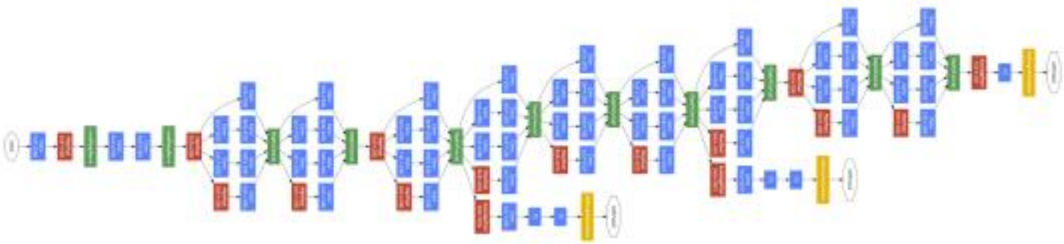
$f(x): X \rightarrow Y$

network structure
(fixed)

A brief theory of supervised deep learning (mini-batch SGD)

labeled samples $x \in X \subset \mathcal{D}$

$f(x)$



layer-wise parameter update
(backpropagation)

	label domain Y	true label $l(x)$
Cat	0.54	1.00
Dog	0.28	0.00
Airplane	0.07	0.00
Horse	0.33	0.00
Banana	0.02	0.00
Truck	0.02	0.00

$f(x): X \rightarrow Y$

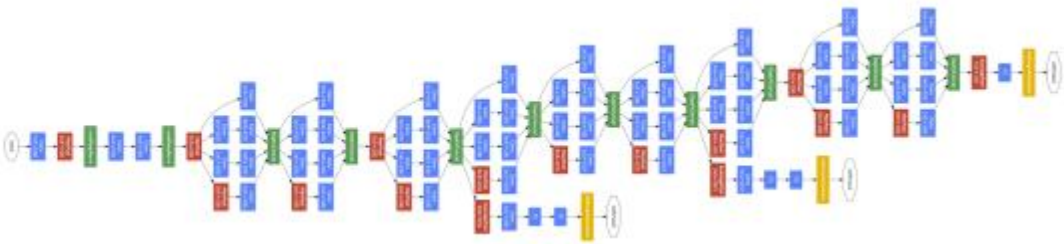
network structure (fixed)

weights w (learned)

A brief theory of supervised deep learning (mini-batch SGD)

labeled samples $x \in X \subset \mathcal{D}$

$f(x)$



layer-wise parameter update
(backpropagation)

label domain Y		true label $l(x)$
Cat	0.54	1.00
Dog	0.28	0.00
Airplane	0.07	0.00
Horse	0.33	0.00
Banana	0.02	0.00
Truck	0.02	0.00

$f(x): X \rightarrow Y$

network structure (fixed)

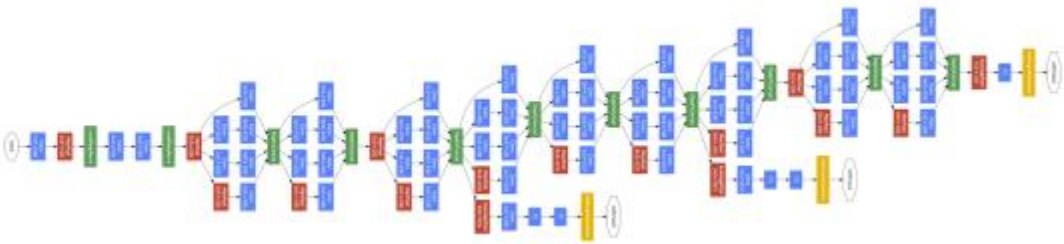
weights w (learned)

$$w^* = \text{argmin}_{w \in \mathbb{R}^d} \mathbb{E}_{x \sim \mathcal{D}}[\ell(w, x)]$$

A brief theory of supervised deep learning (mini-batch SGD)

labeled samples $x \in X \subset \mathcal{D}$

$f(x)$



layer-wise parameter update
(backpropagation)

Cat	0.54
Dog	0.28
Airplane	0.07
Horse	0.33
Banana	0.02
Truck	0.02

label domain Y

Cat	1.00
Dog	0.00
Airplane	0.00
Horse	0.00
Banana	0.00
Truck	0.00

true label $l(x)$

$f(x): X \rightarrow Y$

network structure
(fixed)

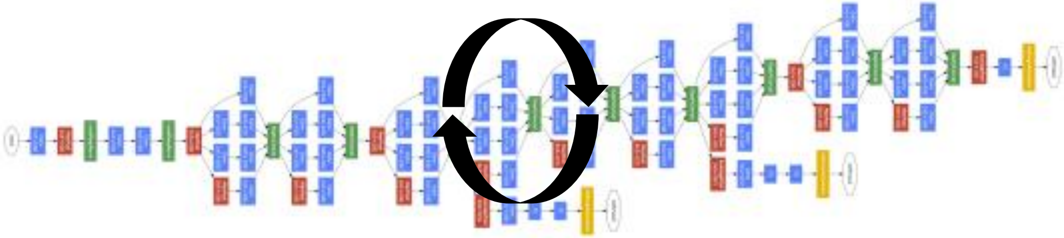
weights w
(learned)

$$w^* = \text{argmin}_{w \in \mathbb{R}^d} \mathbb{E}_{x \sim \mathcal{D}}[\ell(w, x)]$$

A brief theory of supervised deep learning (mini-batch SGD)

labeled samples $x \in X \subset \mathcal{D}$

$f(x)$



layer-wise parameter update
(backpropagation)

Cat	0.54
Dog	0.28
Airplane	0.07
Horse	0.33
Banana	0.02
Truck	0.02

label domain Y

Cat	1.00
Dog	0.00
Airplane	0.00
Horse	0.00
Banana	0.00
Truck	0.00

true label $l(x)$

$$f(x): X \to Y$$

network structure
(fixed)

weights w
(learned)

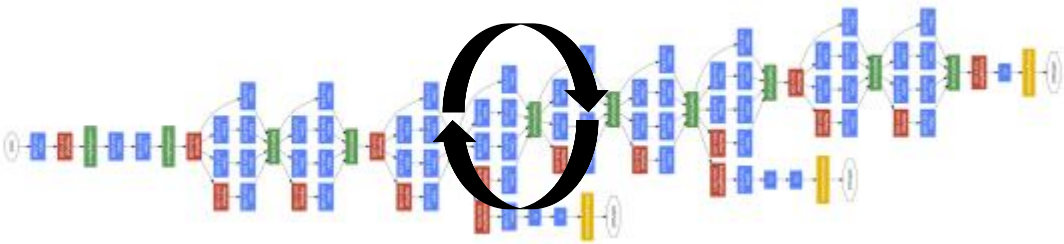
$$\ell_{0-1}(w,x) = \begin{cases} 0 & f(x) = l(x) \\ 1 & f(x) \neq l(x) \end{cases}$$

$$w^* = \operatorname{argmin}_{w \in \mathbb{R}^d} \mathbb{E}_{x \sim \mathcal{D}}[\ell(w,x)]$$

A brief theory of supervised deep learning (mini-batch SGD)

labeled samples $x \in X \subset \mathcal{D}$

$f(x)$



layer-wise parameter update
(backpropagation)

Cat	0.54
Dog	0.28
Airplane	0.07
Horse	0.33
Banana	0.02
Truck	0.02

label domain Y

Cat	1.00
Dog	0.00
Airplane	0.00
Horse	0.00
Banana	0.00
Truck	0.00

true label $l(x)$

$f(x): X \rightarrow Y$

network structure
(fixed)

weights w
(learned)

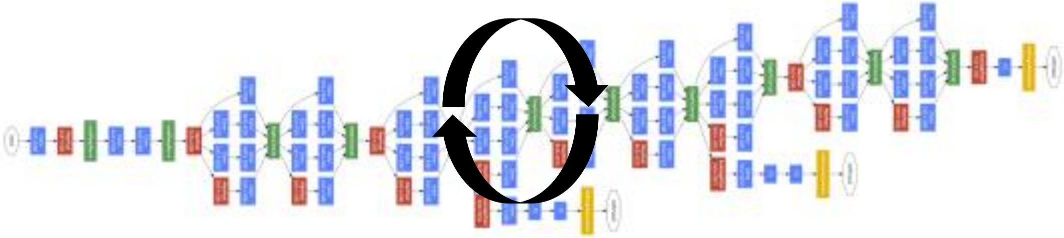
$$\ell_{0-1}(w, x) = \begin{cases} 0 & f(x) = l(x) \\ 1 & f(x) \neq l(x) \end{cases}$$

$$w^* = \operatorname{argmin}_{w \in \mathbb{R}^d} \mathbb{E}_{x \sim \mathcal{D}}[\ell(w, x)]$$

A brief theory of supervised deep learning (mini-batch SGD)

labeled samples $x \in X \subset \mathcal{D}$

$f(x)$



layer-wise parameter update
(backpropagation)

Cat	0.54
Dog	0.28
Airplane	0.07
Horse	0.33
Banana	0.02
Truck	0.02

label domain Y

Cat	1.00
Dog	0.00
Airplane	0.00
Horse	0.00
Banana	0.00
Truck	0.00

true label $l(x)$

$$f(x): X \rightarrow Y$$

network structure
(fixed)

weights w
(learned)

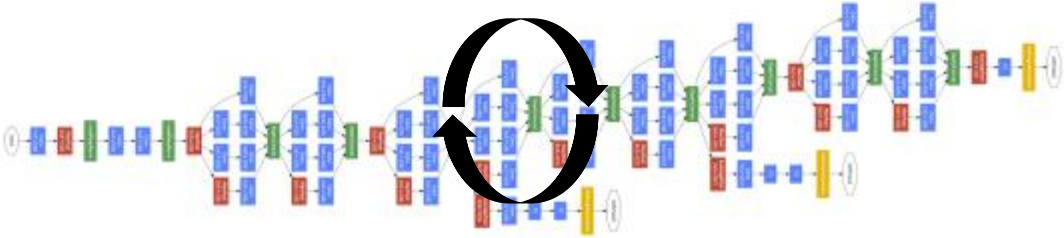
$$\ell_{0-1}(w, x) = \begin{cases} 0 & f(x) = l(x) \\ 1 & f(x) \neq l(x) \end{cases}$$

$$w^* = \operatorname{argmin}_{w \in \mathbb{R}^d} \mathbb{E}_{x \sim \mathcal{D}}[\ell(w, x)]$$

A brief theory of supervised deep learning (mini-batch SGD)

labeled samples $x \in X \subset \mathcal{D}$

$f(x)$



layer-wise parameter update
(backpropagation)

Cat	0.54	Cat	1.00
Dog	0.28	Dog	0.00
Airplane	0.07	Airplane	0.00
Horse	0.33	Horse	0.00
Banana	0.02	Banana	0.00
Truck	0.02	Truck	0.00

label domain Y — true label $l(x)$

$f(x): X \to Y$

network structure (fixed) — weights w (learned)

$$\ell_{sq}(w,x) = \left(f(x) - l(x)\right)^2$$

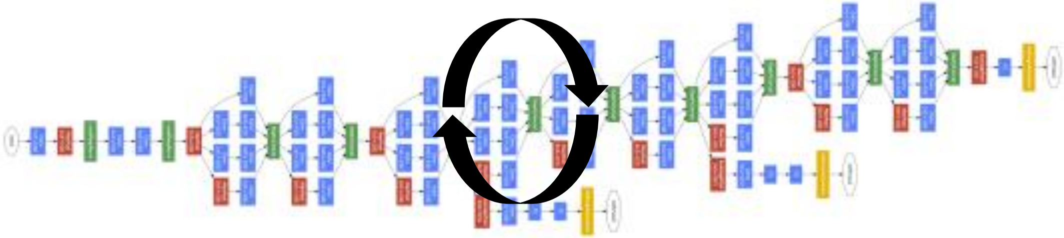
$$\ell_{0-1}(w,x) = \begin{cases} 0 & f(x) = l(x) \\ 1 & f(x) \neq l(x) \end{cases}$$

$$w^* = \operatorname{argmin}_{w \in \mathbb{R}^d} \mathbb{E}_{x \sim \mathcal{D}}[\ell(w,x)]$$

A brief theory of supervised deep learning (mini-batch SGD)

labeled samples $x \in X \subset \mathcal{D}$

$f(x)$



layer-wise parameter update
(backpropagation)

Cat	0.54
Dog	0.28
Airplane	0.07
Horse	0.33
Banana	0.02
Truck	0.02

label domain Y

Cat	1.00
Dog	0.00
Airplane	0.00
Horse	0.00
Banana	0.00
Truck	0.00

true label $l(x)$

$$f(x): X \to Y$$

network structure (fixed)

weights w (learned)

$$w^* = \mathrm{argmin}_{w \in \mathbb{R}^d}\, \mathbb{E}_{x \sim \mathcal{D}}[\ell(w,x)]$$

$$f(x) = f_n\left(f_{n-1}\left(f_{n-2}(\ldots f_1(x)\ldots)\right)\right)$$

convolution 1

$$\ell_{sq}(w,x) = \left(f(x) - l(x)\right)^2$$

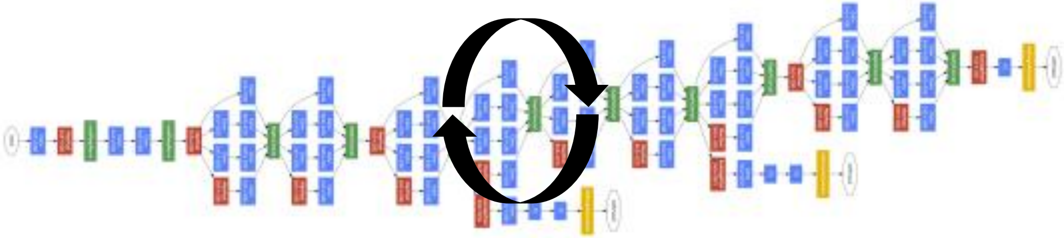
$$\ell_{0-1}(w,x) = \begin{cases} 0 & f(x) = l(x) \\ 1 & f(x) \neq l(x) \end{cases}$$

$$\ell_{ce}(w,x) = -\sum_i l(x)_i \cdot \log \frac{e^{f(x)_i}}{\sum_k e^{f(x)_k}}$$

A brief theory of supervised deep learning (mini-batch SGD)

labeled samples $x \in X \subset \mathcal{D}$

$f(x)$



layer-wise parameter update
(backpropagation)

Cat	0.54
Dog	0.28
Airplane	0.07
Horse	0.33
Banana	0.02
Truck	0.02

label domain Y

Cat	1.00
Dog	0.00
Airplane	0.00
Horse	0.00
Banana	0.00
Truck	0.00

true label $l(x)$

$$f(x): X \to Y$$

network structure (fixed)

weights w (learned)

$$w^* = \mathrm{argmin}_{w \in \mathbb{R}^d} \mathbb{E}_{x \sim \mathcal{D}}[\ell(w,x)]$$

$$f(x) = f_n\left(f_{n-1}\left(f_{n-2}(\ldots f_1(x) \ldots)\right)\right)$$

convolution 1

convolution 2

$$\ell_{sq}(w,x) = \left(f(x) - l(x)\right)^2$$

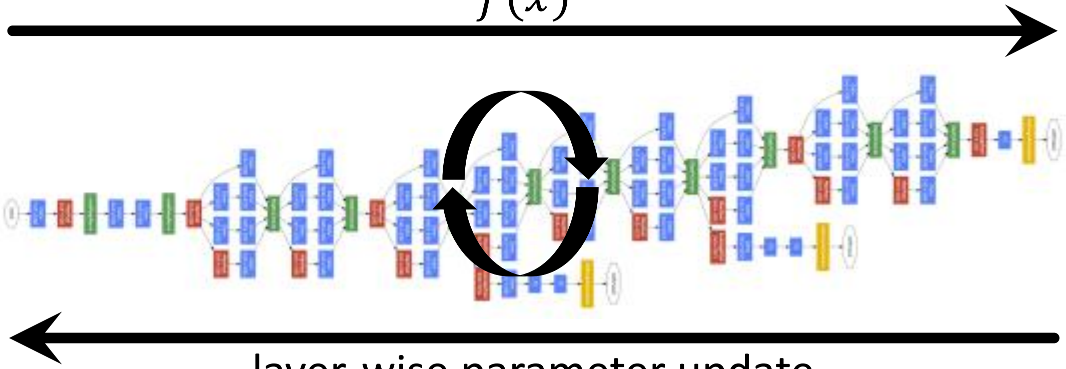
$$\ell_{0-1}(w,x) = \begin{cases} 0 & f(x) = l(x) \\ 1 & f(x) \neq l(x) \end{cases}$$

$$\ell_{ce}(w,x) = -\sum_i l(x)_i \cdot \log \frac{e^{f(x)_i}}{\sum_k e^{f(x)_k}}$$

A brief theory of supervised deep learning (mini-batch SGD)

labeled samples $x \in X \subset \mathcal{D}$

$f(x)$



layer-wise parameter update
(backpropagation)

Cat	0.54
Dog	0.28
Airplane	0.07
Horse	0.33
Banana	0.02
Truck	0.02

label domain Y

Cat	1.00
Dog	0.00
Airplane	0.00
Horse	0.00
Banana	0.00
Truck	0.00

true label $l(x)$

$$f(x): X \to Y$$

network structure
(fixed)

weights w
(learned)

$$w^* = \text{argmin}_{w \in \mathbb{R}^d} \mathbb{E}_{x \sim \mathcal{D}}[\ell(w, x)]$$

$$f(x) = f_n\left(f_{n-1}\left(f_{n-2}\left(\ldots f_1(x) \ldots\right)\right)\right)$$

convolution 1
convolution 2
pooling
convolution 3

$$\ell_{sq}(w, x) = \left(f(x) - l(x)\right)^2$$

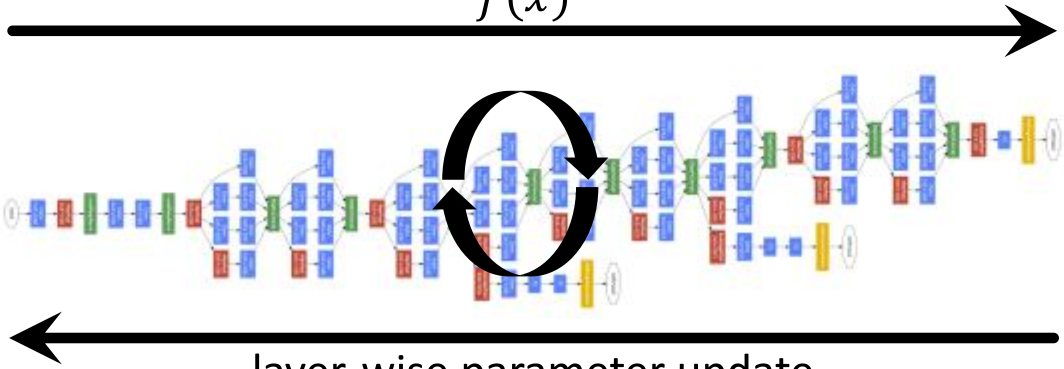
$$\ell_{0-1}(w, x) = \begin{cases} 0 & f(x) = l(x) \\ 1 & f(x) \neq l(x) \end{cases}$$

$$\ell_{ce}(w, x) = -\sum_i l(x)_i \cdot \log \frac{e^{f(x)_i}}{\sum_k e^{f(x)_k}}$$

A brief theory of supervised deep learning (mini-batch SGD)

labeled samples $x \in X \subset \mathcal{D}$

$f(x)$



layer-wise parameter update
(backpropagation)

Cat	0.54	Cat	1.00
Dog	0.28	Dog	0.00
Airplane	0.07	Airplane	0.00
Horse	0.33	Horse	0.00
Banana	0.02	Banana	0.00
Truck	0.02	Truck	0.00

label domain Y — true label $l(x)$

$$f(x): X \to Y$$

network structure (fixed) — weights w (learned)

$$w^* = \mathrm{argmin}_{w \in \mathbb{R}^d} \mathbb{E}_{x \sim \mathcal{D}}[\ell(w,x)]$$

$$f(x) = f_n\left(f_{n-1}\left(f_{n-2}\left(\ldots f_1(x) \ldots\right)\right)\right)$$

convolution 1 | convolution 2 | pooling | convolution 3

$$\ell_{sq}(w,x) = \left(f(x) - l(x)\right)^2$$

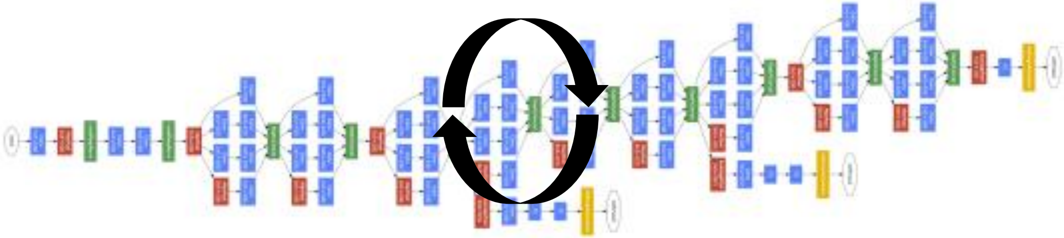
$$\ell_{0-1}(w,x) = \begin{cases} 0 & f(x) = l(x) \\ 1 & f(x) \neq l(x) \end{cases}$$

$$\ell_{ce}(w,x) = -\sum_i l(x)_i \cdot \log \frac{e^{f(x)_i}}{\sum_k e^{f(x)_k}}$$

A brief theory of supervised deep learning (mini-batch SGD)

labeled samples $x \in X \subset \mathcal{D}$

$f(x)$



layer-wise parameter update
(backpropagation)

Cat	0.54
Dog	0.28
Airplane	0.07
Horse	0.33
Banana	0.02
Truck	0.02

label domain Y

Cat	1.00
Dog	0.00
Airplane	0.00
Horse	0.00
Banana	0.00
Truck	0.00

true label $l(x)$

$f(x): X \to Y$

network structure
(fixed)

weights w
(learned)

$$w^* = \text{argmin}_{w \in \mathbb{R}^d} \mathbb{E}_{x \sim \mathcal{D}}[\ell(w,x)]$$

$$f(x) = f_n\left(f_{n-1}\left(f_{n-2}\left(\ldots f_1(x) \ldots\right)\right)\right)$$

convolution 1

convolution 2

pooling

convolution 3

fully connected

$$\ell_{sq}(w,x) = \left(f(x) - l(x)\right)^2$$

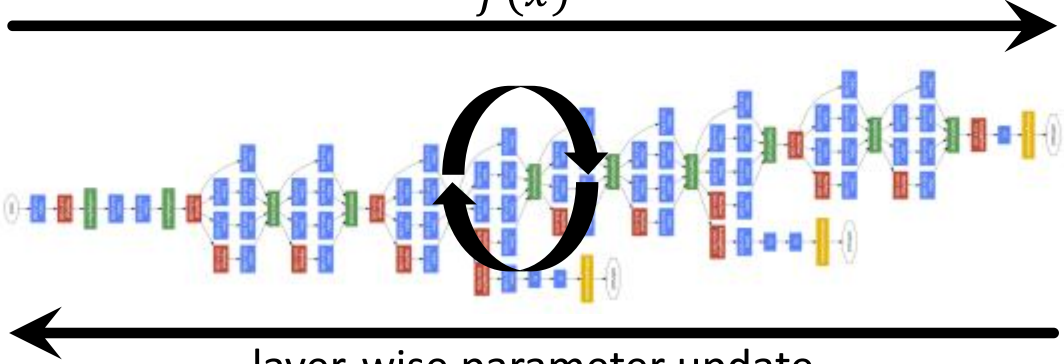
$$\ell_{0-1}(w,x) = \begin{cases} 0 & f(x) = l(x) \\ 1 & f(x) \neq l(x) \end{cases}$$

$$\ell_{ce}(w,x) = -\sum_i l(x)_i \cdot \log \frac{e^{f(x)_i}}{\sum_k e^{f(x)_k}}$$

A brief theory of supervised deep learning (mini-batch SGD)

labeled samples $x \in X \subset \mathcal{D}$

$f(x)$



layer-wise parameter update
(backpropagation)

label domain Y		true label $l(x)$
Cat	0.54	1.00
Dog	0.28	0.00
Airplane	0.07	0.00
Horse	0.33	0.00
Banana	0.02	0.00
Truck	0.02	0.00

$f(x): X \to Y$

network structure (fixed)

weights w (learned)

$$w^* = \operatorname{argmin}_{w \in \mathbb{R}^d} \mathbb{E}_{x \sim \mathcal{D}}[\ell(w,x)]$$

$$f(x) = f_n\left(f_{n-1}\left(f_{n-2}\left(\dots f_1(x) \dots\right)\right)\right)$$

$$\ell_{sq}(w,x) = \left(f(x) - l(x)\right)^2$$

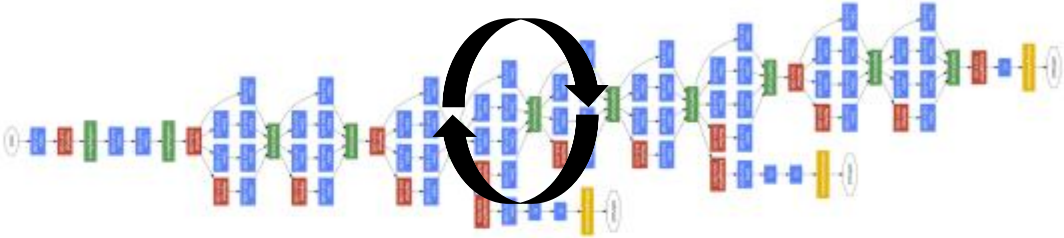
$$\ell_{0-1}(w,x) = \begin{cases} 0 & f(x) = l(x) \\ 1 & f(x) \neq l(x) \end{cases}$$

$$\ell_{ce}(w,x) = -\sum_i l(x)_i \cdot \log \frac{e^{f(x)_i}}{\sum_k e^{f(x)_k}}$$

A brief theory of supervised deep learning (mini-batch SGD)

labeled samples $x \in X \subset \mathcal{D}$

$f(x)$



layer-wise parameter update
(backpropagation)

Cat	0.54
Dog	0.28
Airplane	0.07
Horse	0.33
Banana	0.02
Truck	0.02

label domain Y

Cat	1.00
Dog	0.00
Airplane	0.00
Horse	0.00
Banana	0.00
Truck	0.00

true label $l(x)$

$$f(x): X \to Y$$

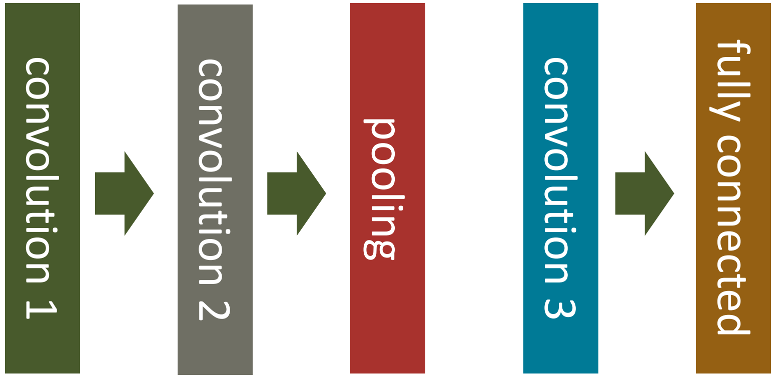
network structure
(fixed)

weights w
(learned)

$$w^* = \text{argmin}_{w \in \mathbb{R}^d} \mathbb{E}_{x \sim \mathcal{D}}[\ell(w,x)]$$

$$f(x) = f_n\left(f_{n-1}\left(f_{n-2}\left(\ldots f_1(x) \ldots\right)\right)\right)$$

$f_1(x)$



$$\ell_{sq}(w,x) = \left(f(x) - l(x)\right)^2$$

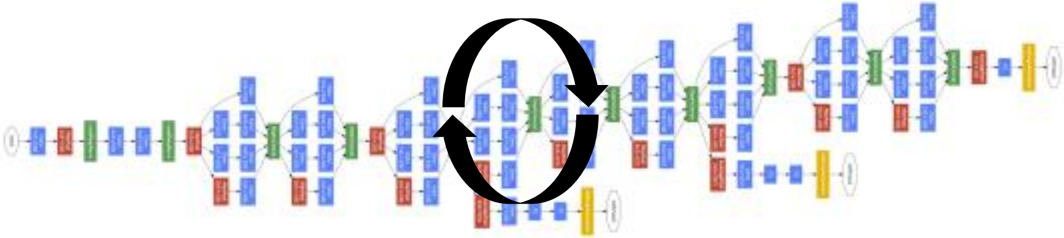
$$\ell_{0-1}(w,x) = \begin{cases} 0 & f(x) = l(x) \\ 1 & f(x) \neq l(x) \end{cases}$$

$$\ell_{ce}(w,x) = -\sum_i l(x)_i \cdot \log \frac{e^{f(x)_i}}{\sum_k e^{f(x)_k}}$$

A brief theory of supervised deep learning (mini-batch SGD)

labeled samples $x \in X \subset \mathcal{D}$

$f(x)$



layer-wise parameter update
(backpropagation)

Cat	0.54
Dog	0.28
Airplane	0.07
Horse	0.33
Banana	0.02
Truck	0.02

label domain Y

Cat	1.00
Dog	0.00
Airplane	0.00
Horse	0.00
Banana	0.00
Truck	0.00

true label $l(x)$

$f(x): X \to Y$

network structure (fixed) | weights w (learned)

$$w^* = \text{argmin}_{w \in \mathbb{R}^d} \mathbb{E}_{x \sim \mathcal{D}}[\ell(w, x)]$$

$$f(x) = f_n\left(f_{n-1}\left(f_{n-2}\left(\dots f_1(x) \dots\right)\right)\right)$$

$f_1(x)$ $f_2\left(f_1(x)\right)$

convolution 1 → convolution 2 → pooling → convolution 3 → fully connected

$$\ell_{sq}(w, x) = \left(f(x) - l(x)\right)^2$$

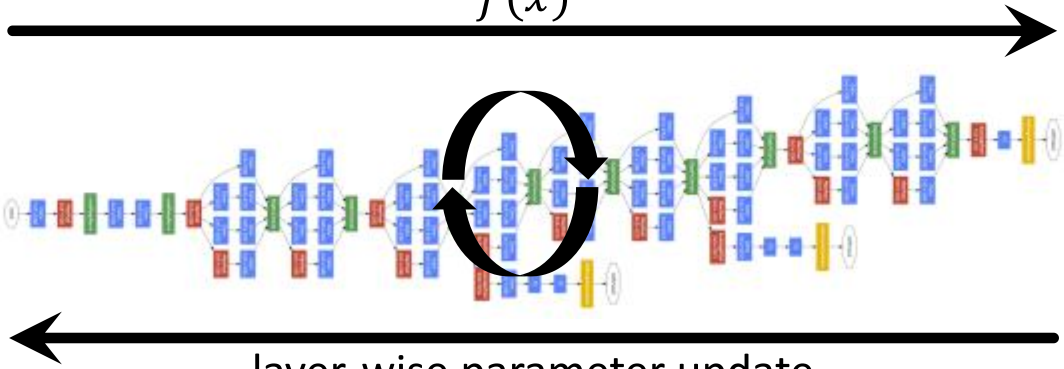
$$\ell_{0-1}(w, x) = \begin{cases} 0 & f(x) = l(x) \\ 1 & f(x) \neq l(x) \end{cases}$$

$$\ell_{ce}(w, x) = -\sum_i l(x)_i \cdot \log \frac{e^{f(x)_i}}{\sum_k e^{f(x)_k}}$$

A brief theory of supervised deep learning (mini-batch SGD)

labeled samples $x \in X \subset \mathcal{D}$

$f(x)$



layer-wise parameter update
(backpropagation)

Label		Label	
Cat	0.54	Cat	1.00
Dog	0.28	Dog	0.00
Airplane	0.07	Airplane	0.00
Horse	0.33	Horse	0.00
Banana	0.02	Banana	0.00
Truck	0.02	Truck	0.00

label domain Y true label $l(x)$

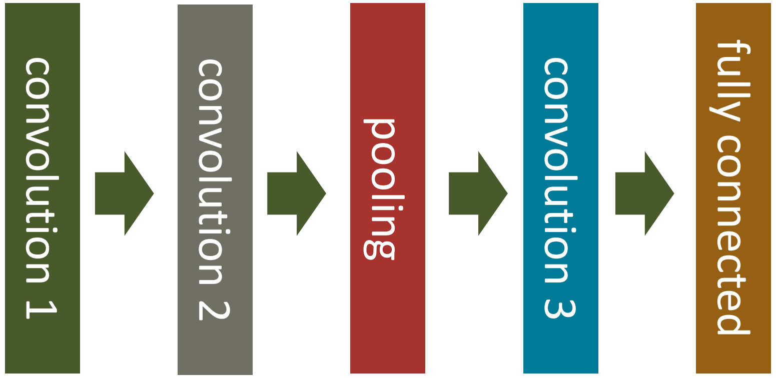
$f(x): X \to Y$

network structure (fixed) weights w (learned)

$$w^* = \mathrm{argmin}_{w \in \mathbb{R}^d} \mathbb{E}_{x \sim \mathcal{D}}[\ell(w, x)]$$

$$f(x) = f_n\left(f_{n-1}\left(f_{n-2}\left(\dots f_1(x) \dots\right)\right)\right)$$

$f_1(x)$ $f_2(f_1(x))$ … $f(x)$



$$\ell_{sq}(w, x) = \left(f(x) - l(x)\right)^2$$

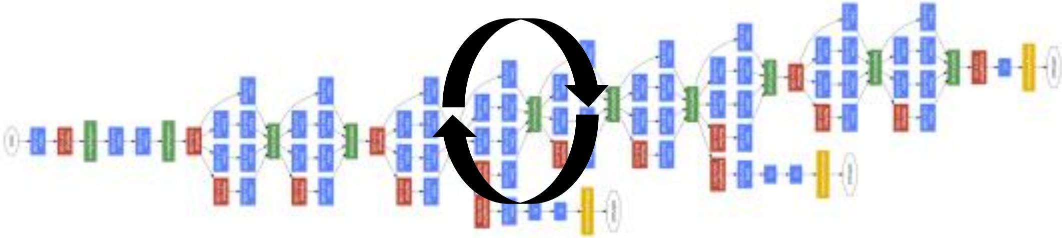
$$\ell_{0-1}(w, x) = \begin{cases} 0 & f(x) = l(x) \\ 1 & f(x) \neq l(x) \end{cases}$$

$$\ell_{ce}(w, x) = -\sum_i l(x)_i \cdot \log \frac{e^{f(x)_i}}{\sum_k e^{f(x)_k}}$$

A brief theory of supervised deep learning (mini-batch SGD)

labeled samples $x \in X \subset \mathcal{D}$

$f(x)$



layer-wise parameter update
(backpropagation)

label domain Y	
Cat	0.54
Dog	0.28
Airplane	0.07
Horse	0.33
Banana	0.02
Truck	0.02

true label $l(x)$	
Cat	1.00
Dog	0.00
Airplane	0.00
Horse	0.00
Banana	0.00
Truck	0.00

$f(x): X \to Y$

network structure (fixed)

weights w (learned)

$$w^* = \text{argmin}_{w \in \mathbb{R}^d} \mathbb{E}_{x\sim\mathcal{D}}[\ell(w,x)]$$

$$f(x) = f_n\left(f_{n-1}\left(f_{n-2}\left(\ldots f_1(x) \ldots\right)\right)\right)$$

$f_1(x)$ $f_2(f_1(x))$ … $f(x)$

convolution 1 → convolution 2 → pooling → convolution 3 → fully connected

$$\ell_{sq}(w,x) = \left(f(x) - l(x)\right)^2$$

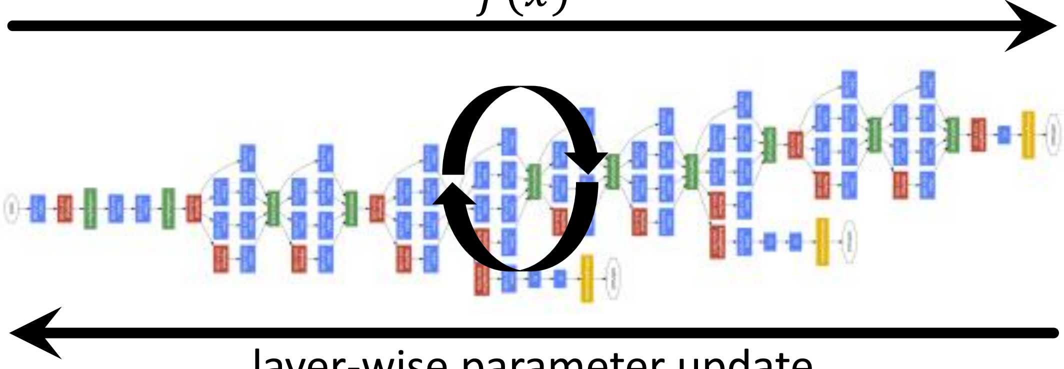
$$\ell_{0-1}(w,x) = \begin{cases} 0 & f(x) = l(x) \\ 1 & f(x) \neq l(x) \end{cases}$$

$$\ell_{ce}(w,x) = -\sum_i l(x)_i \cdot \log \frac{e^{f(x)_i}}{\sum_k e^{f(x)_k}}$$

A brief theory of supervised deep learning (mini-batch SGD)

labeled samples $x \in X \subset \mathcal{D}$

$f(x)$



layer-wise parameter update (backpropagation)

label domain Y		true label $l(x)$	
Cat	0.54	Cat	1.00
Dog	0.28	Dog	0.00
Airplane	0.07	Airplane	0.00
Horse	0.33	Horse	0.00
Banana	0.02	Banana	0.00
Truck	0.02	Truck	0.00

$f(x): X \rightarrow Y$

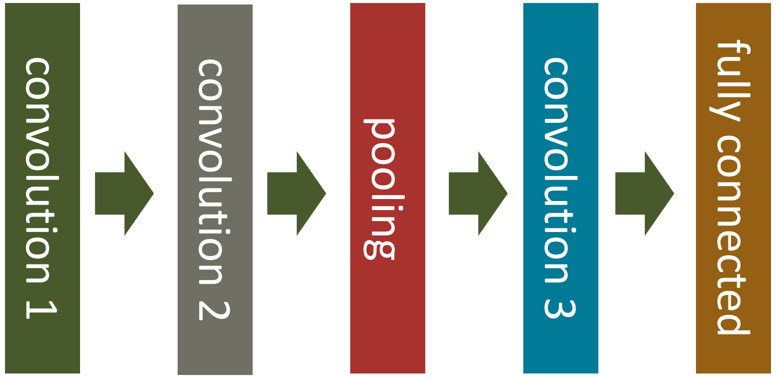
network structure (fixed)

weights w (learned)

$$w^* = \mathrm{argmin}_{w \in \mathbb{R}^d}\, \mathbb{E}_{x \sim \mathcal{D}}[\ell(w,x)]$$

$$f(x) = f_n\left(f_{n-1}\left(f_{n-2}(\ldots f_1(x)\ldots)\right)\right)$$

$f_1(x)$ $\quad f_2(f_1(x))$ $\quad \ldots \quad$ $f(x)$



$$\ell_{sq}(w,x) = \left(f(x) - l(x)\right)^2$$

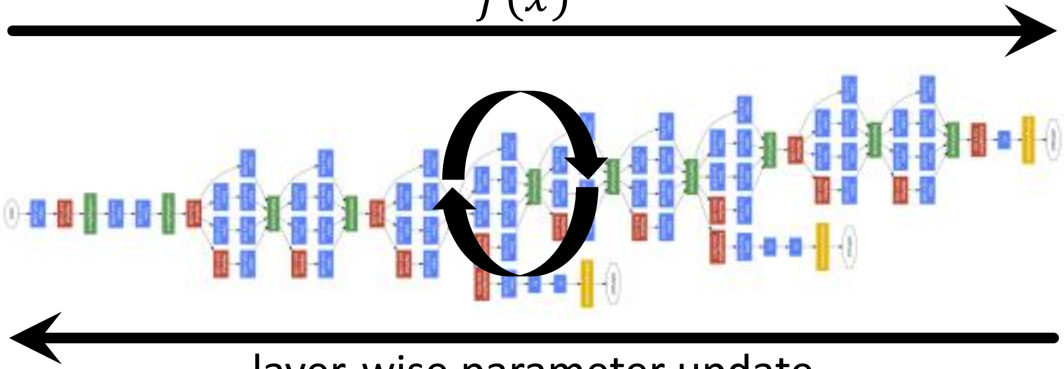
$$\ell_{0-1}(w,x) = \begin{cases} 0 & f(x) = l(x) \\ 1 & f(x) \neq l(x) \end{cases}$$

$$\ell_{ce}(w,x) = -\sum_i l(x)_i \cdot \log \frac{e^{f(x)_i}}{\sum_k e^{f(x)_k}}$$

A brief theory of supervised deep learning (mini-batch SGD)

labeled samples $x \in X \subset \mathcal{D}$

$f(x)$



layer-wise parameter update
(backpropagation)

	label domain Y	true label $l(x)$
Cat	0.54	1.00
Dog	0.28	0.00
Airplane	0.07	0.00
Horse	0.33	0.00
Banana	0.02	0.00
Truck	0.02	0.00

$f(x): X \to Y$

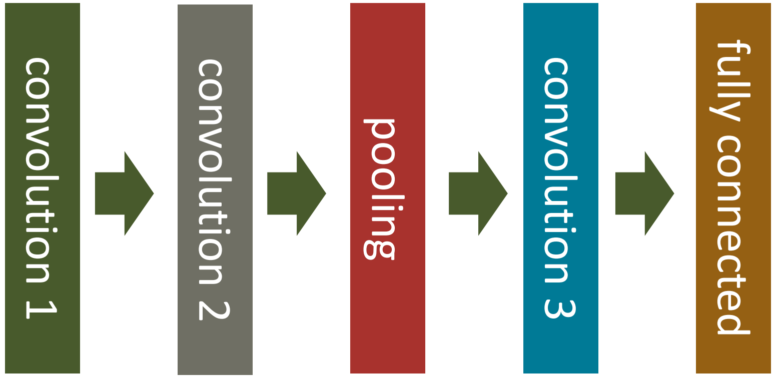
network structure (fixed)

weights w (learned)

$$w^* = \mathrm{argmin}_{w \in \mathbb{R}^d} \mathbb{E}_{x \sim \mathcal{D}}[\ell(w,x)]$$

$$f(x) = f_n\left(f_{n-1}\left(f_{n-2}\left(\dots f_1(x) \dots\right)\right)\right)$$

$f_1(x)$ $\quad f_2(f_1(x))$ $\quad \dots$ $\quad f(x)$



$$\ell_{sq}(w,x) = \left(f(x) - l(x)\right)^2$$

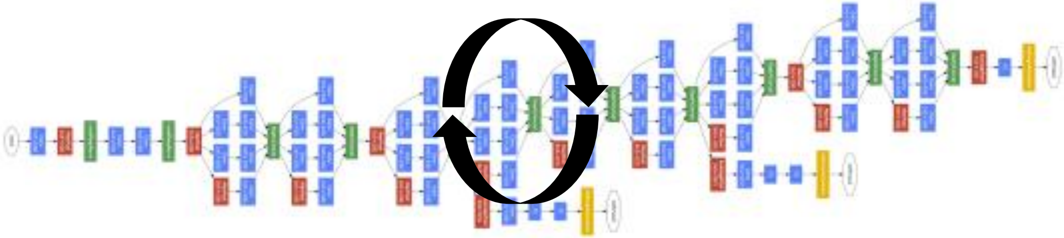
$$\ell_{0-1}(w,x) = \begin{cases} 0 & f(x) = l(x) \\ 1 & f(x) \neq l(x) \end{cases}$$

$$\ell_{ce}(w,x) = -\sum_i l(x)_i \cdot \log \frac{e^{f(x)_i}}{\sum_k e^{f(x)_k}}$$

A brief theory of supervised deep learning (mini-batch SGD)

labeled samples $x \in X \subset \mathcal{D}$

$f(x)$



layer-wise parameter update
(backpropagation)

label domain Y		true label $l(x)$
Cat	0.54	1.00
Dog	0.28	0.00
Airplane	0.07	0.00
Horse	0.33	0.00
Banana	0.02	0.00
Truck	0.02	0.00

$f(x): X \rightarrow Y$

network structure (fixed)

weights w (learned)

$$w^* = \operatorname{argmin}_{w \in \mathbb{R}^d} \mathbb{E}_{x \sim \mathcal{D}}[\ell(w,x)]$$

$$f(x) = f_n\left(f_{n-1}\left(f_{n-2}\left(\ldots f_1(x) \ldots\right)\right)\right)$$

$f_1(x)$ $f_2(f_1(x))$ … $f(x)$

convolution 1 → convolution 2 → pooling → convolution 3 → fully connected

$$\ell_{sq}(w,x) = \left(f(x) - l(x)\right)^2$$

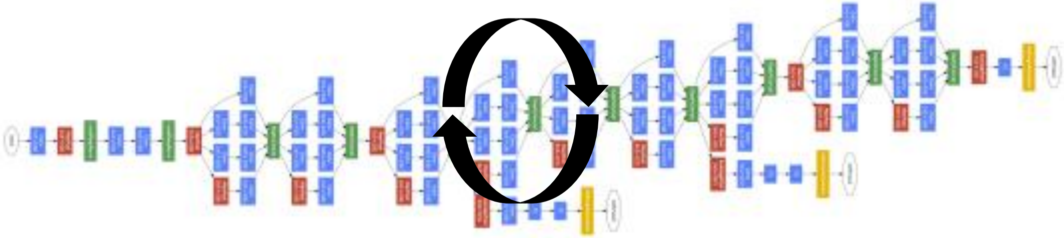
$$\ell_{0-1}(w,x) = \begin{cases} 0 & f(x) = l(x) \\ 1 & f(x) \neq l(x) \end{cases}$$

$$\ell_{ce}(w,x) = -\sum_i l(x)_i \cdot \log \frac{e^{f(x)_i}}{\sum_k e^{f(x)_k}}$$

A brief theory of supervised deep learning (mini-batch SGD)

labeled samples $x \in X \subset \mathcal{D}$

$f(x)$



layer-wise parameter update (backpropagation)

Cat	0.54
Dog	0.28
Airplane	0.07
Horse	0.33
Banana	0.02
Truck	0.02

label domain Y

Cat	1.00
Dog	0.00
Airplane	0.00
Horse	0.00
Banana	0.00
Truck	0.00

true label $l(x)$

$f(x): X \rightarrow Y$

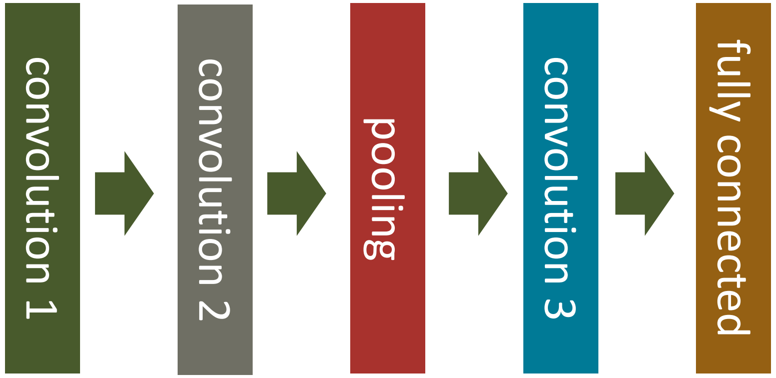
network structure (fixed)

weights w (learned)

$$w^* = \text{argmin}_{w \in \mathbb{R}^d} \mathbb{E}_{x\sim\mathcal{D}}[\ell(w,x)]$$

$$f(x) = f_n\left(f_{n-1}\left(f_{n-2}\left(\ldots f_1(x) \ldots\right)\right)\right)$$

$f_1(x)$ $f_2(f_1(x))$ … $f(x)$



$$\ell_{sq}(w,x) = \left(f(x) - l(x)\right)^2$$

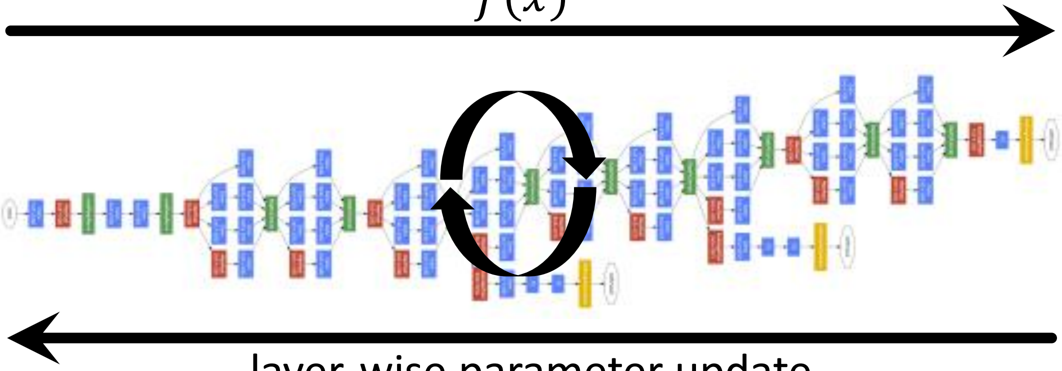
$$\ell_{0-1}(w,x) = \begin{cases} 0 & f(x) = l(x) \\ 1 & f(x) \neq l(x) \end{cases}$$

$$\ell_{ce}(w,x) = -\sum_i l(x)_i \cdot \log \frac{e^{f(x)_i}}{\sum_k e^{f(x)_k}}$$

A brief theory of supervised deep learning (mini-batch SGD)

labeled samples $x \in X \subset \mathcal{D}$

$f(x)$



layer-wise parameter update
(backpropagation)

label domain Y		true label $l(x)$	
Cat	0.54	Cat	1.00
Dog	0.28	Dog	0.00
Airplane	0.07	Airplane	0.00
Horse	0.33	Horse	0.00
Banana	0.02	Banana	0.00
Truck	0.02	Truck	0.00

$f(x): X \rightarrow Y$

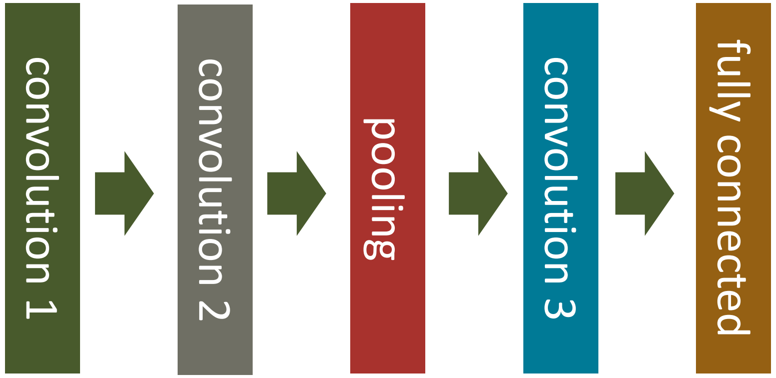
network structure (fixed)

weights w (learned)

$$w^* = \operatorname{argmin}_{w \in \mathbb{R}^d} \mathbb{E}_{x \sim \mathcal{D}}[\ell(w, x)]$$

$$f(x) = f_n\left(f_{n-1}\left(f_{n-2}\left(\dots f_1(x) \dots\right)\right)\right)$$

$f_1(x)$ $f_2(f_1(x))$ … $f(x)$



$$\ell_{sq}(w, x) = \left(f(x) - l(x)\right)^2$$

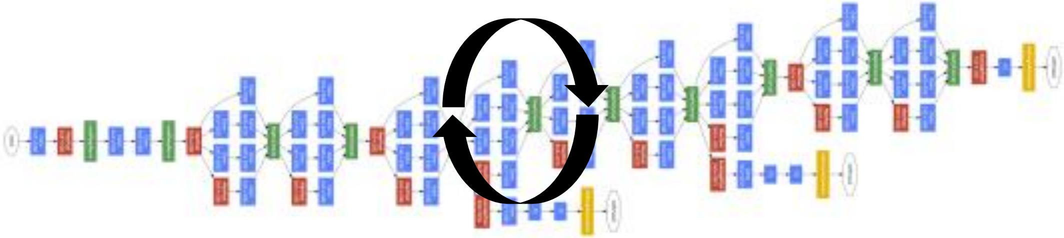
$$\ell_{0-1}(w, x) = \begin{cases} 0 & f(x) = l(x) \\ 1 & f(x) \neq l(x) \end{cases}$$

$$\ell_{ce}(w, x) = -\sum_i l(x)_i \cdot \log \frac{e^{f(x)_i}}{\sum_k e^{f(x)_k}}$$

A brief theory of supervised deep learning (mini-batch SGD)

labeled samples $x \in X \subset \mathcal{D}$

$f(x)$



layer-wise parameter update (backpropagation)

label domain Y		true label $l(x)$	
Cat	0.54	Cat	1.00
Dog	0.28	Dog	0.00
Airplane	0.07	Airplane	0.00
Horse	0.33	Horse	0.00
Banana	0.02	Banana	0.00
Truck	0.02	Truck	0.00

$f(x): X \to Y$

network structure (fixed) | weights w (learned)

$$w^* = \text{argmin}_{w \in \mathbb{R}^d} \mathbb{E}_{x \sim \mathcal{D}}[\ell(w,x)]$$

$$f(x) = f_n\left(f_{n-1}\left(f_{n-2}\left(\dots f_1(x) \dots\right)\right)\right)$$

$f_1(x)$ | $f_2(f_1(x))$ | … | $f(x)$

convolution 1 → convolution 2 → pooling → convolution 3 → fully connected

$$\ell_{sq}(w,x) = \left(f(x) - l(x)\right)^2$$

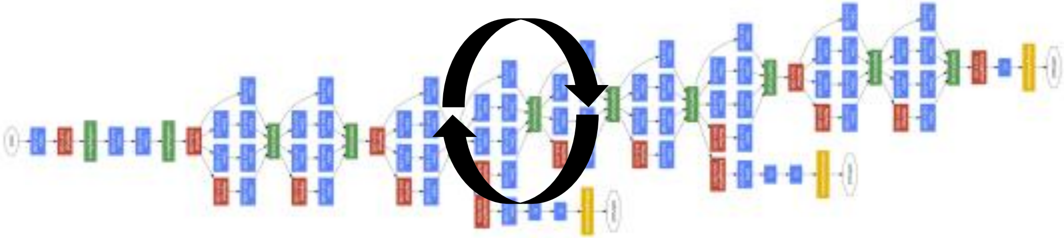
$$\ell_{0-1}(w,x) = \begin{cases} 0 & f(x) = l(x) \\ 1 & f(x) \neq l(x) \end{cases}$$

$$\ell_{ce}(w,x) = -\sum_i l(x)_i \cdot \log \frac{e^{f(x)_i}}{\sum_k e^{f(x)_k}}$$

A brief theory of supervised deep learning (mini-batch SGD)

labeled samples $x \in X \subset \mathcal{D}$

$f(x)$



layer-wise parameter update (backpropagation)

	label domain Y		true label $l(x)$
Cat	0.54	Cat	1.00
Dog	0.28	Dog	0.00
Airplane	0.07	Airplane	0.00
Horse	0.33	Horse	0.00
Banana	0.02	Banana	0.00
Truck	0.02	Truck	0.00

$f(x): X \rightarrow Y$

network structure (fixed)

weights w (learned)

$$w^* = \text{argmin}_{w \in \mathbb{R}^d} \mathbb{E}_{x \sim \mathcal{D}}[\ell(w, x)]$$

$$f(x) = f_n\left(f_{n-1}\left(f_{n-2}\left(\dots f_1(x) \dots\right)\right)\right)$$

$f_1(x)$ $f_2(f_1(x))$ … $f(x)$

convolution 1 → convolution 2 → pooling → convolution 3 → fully connected

$$\ell_{sq}(w, x) = \left(f(x) - l(x)\right)^2$$

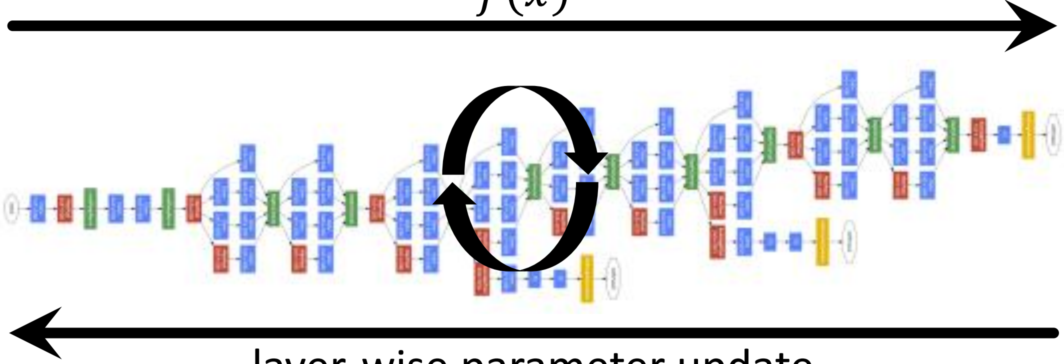
$$\ell_{0-1}(w, x) = \begin{cases} 0 & f(x) = l(x) \\ 1 & f(x) \neq l(x) \end{cases}$$

$$\ell_{ce}(w, x) = -\sum_i l(x)_i \cdot \log \frac{e^{f(x)_i}}{\sum_k e^{f(x)_k}}$$

A brief theory of supervised deep learning (mini-batch SGD)

labeled samples $x \in X \subset \mathcal{D}$

$f(x)$



layer-wise parameter update (backpropagation)

Cat	0.54
Dog	0.28
Airplane	0.07
Horse	0.33
Banana	0.02
Truck	0.02

label domain Y

Cat	1.00
Dog	0.00
Airplane	0.00
Horse	0.00
Banana	0.00
Truck	0.00

true label $l(x)$

$f(x): X \rightarrow Y$

network structure (fixed)

weights w (learned)

$$w^* = \text{argmin}_{w \in \mathbb{R}^d} \mathbb{E}_{x \sim \mathcal{D}}[\ell(w, x)]$$

$$f(x) = f_n\left(f_{n-1}\left(f_{n-2}(\ldots f_1(x) \ldots)\right)\right)$$

$f_1(x)$ $f_2(f_1(x))$ … $f(x)$

convolution 1 → convolution 2 → pooling → convolution 3 → fully connected

$$\ell_{sq}(w, x) = \left(f(x) - l(x)\right)^2$$

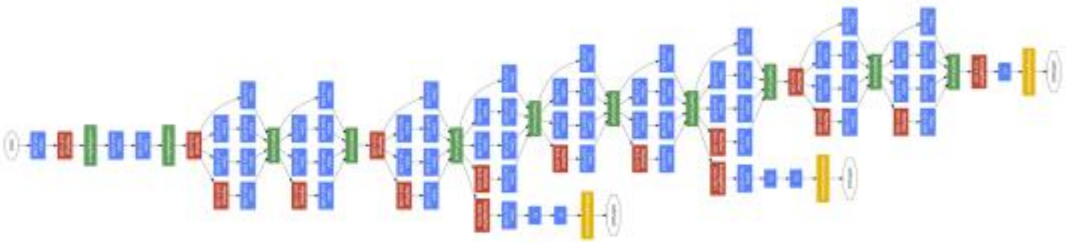
$$\ell_{0-1}(w, x) = \begin{cases} 0 & f(x) = l(x) \\ 1 & f(x) \neq l(x) \end{cases}$$

$$\ell_{ce}(w, x) = -\sum_i l(x)_i \cdot \log \frac{e^{f(x)_i}}{\sum_k e^{f(x)_k}}$$

A brief digression on backpropagation

- "Backward propagation of errors" (Rumelhart, Hinton, & Williams 1986)

$f(x)$

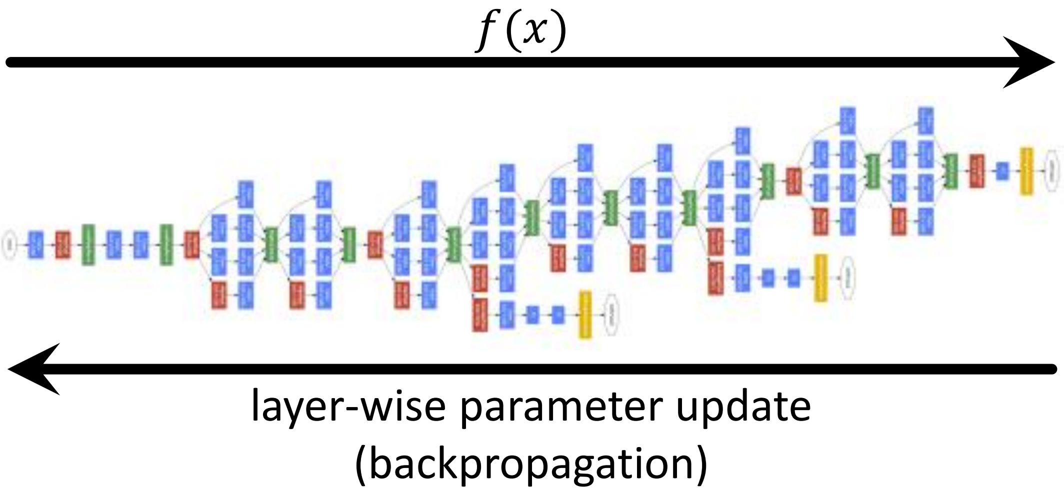


Cat	0.54	Cat	1.00
Dog	0.28	Dog	0.00
Airplane	0.07	Airplane	0.00
Horse	0.33	Horse	0.00
Banana	0.02	Banana	0.00
Truck	0.02	Truck	0.00

$$\ell_{ce}(w,x) = -\sum_i l(x)_i \cdot \log \frac{e^{f(x)_i}}{\sum_k e^{f(x)_k}}$$

A brief digression on backpropagation

- "Backward propagation of errors" (Rumelhart, Hinton, & Williams 1986)

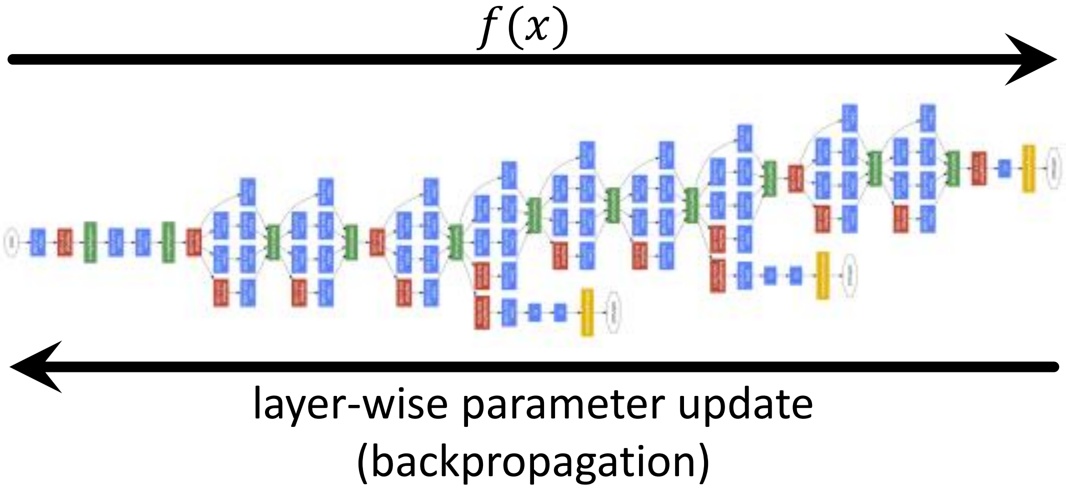


Cat	0.54	Cat	1.00
Dog	0.28	Dog	0.00
Airplane	0.07	Airplane	0.00
Horse	0.33	Horse	0.00
Banana	0.02	Banana	0.00
Truck	0.02	Truck	0.00

$$\ell_{ce}(w,x) = -\sum_i l(x)_i \cdot \log \frac{e^{f(x)_i}}{\sum_k e^{f(x)_k}}$$

A brief digression on backpropagation

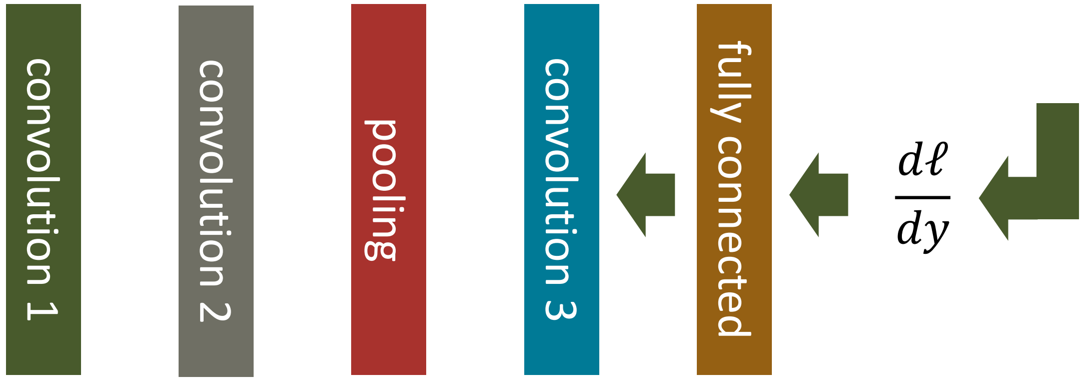
- ”Backward propagation of errors” (Rumelhart, Hinton, & Williams 1986)



Cat	0.54
Dog	0.28
Airplane	0.07
Horse	0.33
Banana	0.02
Truck	0.02

Cat	1.00
Dog	0.00
Airplane	0.00
Horse	0.00
Banana	0.00
Truck	0.00

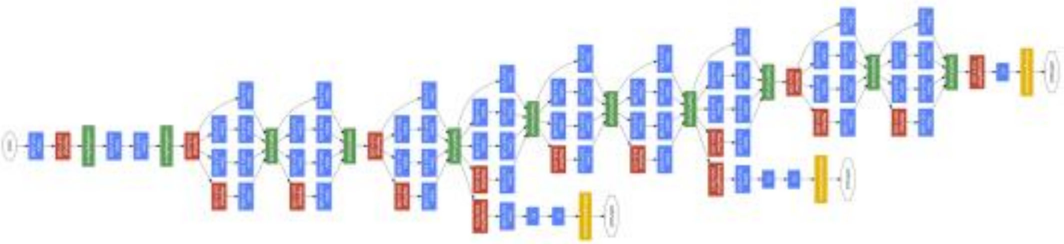
$$\ell_{ce}(w,x) = -\sum_i l(x)_i \cdot \log \frac{e^{f(x)_i}}{\sum_k e^{f(x)_k}}$$



A brief digression on backpropagation

- ”Backward propagation of errors” (Rumelhart, Hinton, & Williams 1986)

$f(x)$

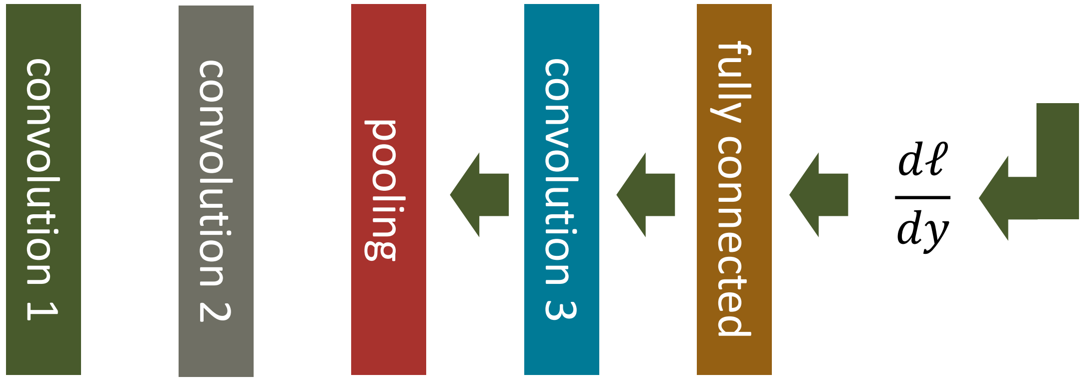


layer-wise parameter update
(backpropagation)

Cat	0.54
Dog	0.28
Airplane	0.07
Horse	0.33
Banana	0.02
Truck	0.02

Cat	1.00
Dog	0.00
Airplane	0.00
Horse	0.00
Banana	0.00
Truck	0.00

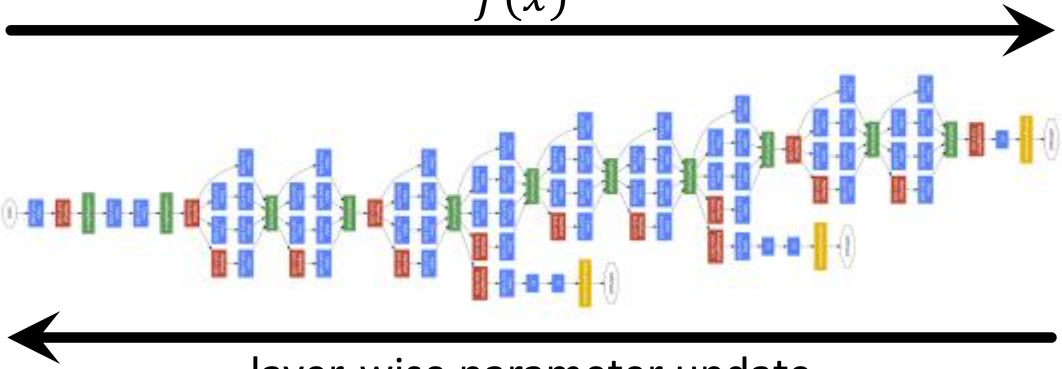
$$\ell_{ce}(w,x) = -\sum_i l(x)_i \cdot \log \frac{e^{f(x)_i}}{\sum_k e^{f(x)_k}}$$



A brief digression on backpropagation

- "Backward propagation of errors" (Rumelhart, Hinton, & Williams 1986)

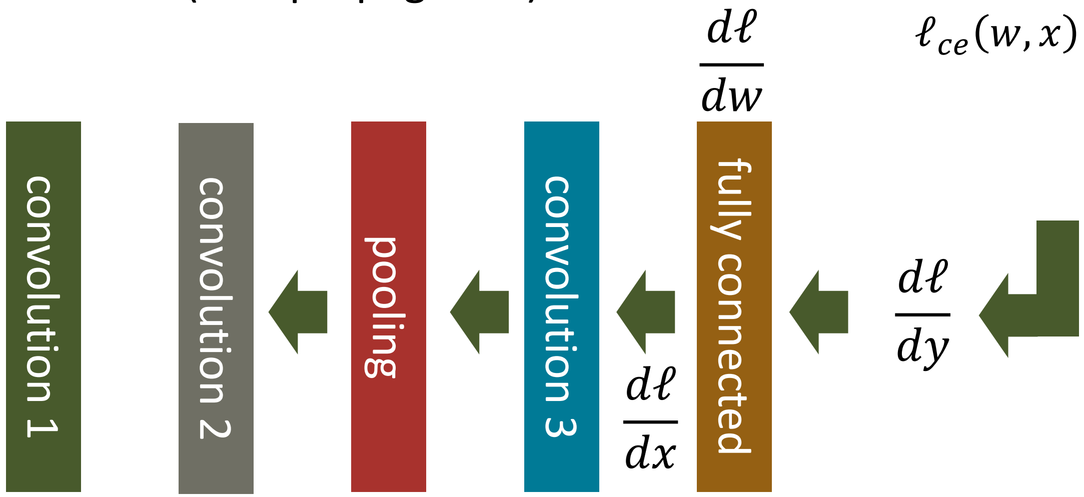
$f(x)$



layer-wise parameter update
(backpropagation)

Cat	0.54	Cat	1.00
Dog	0.28	Dog	0.00
Airplane	0.07	Airplane	0.00
Horse	0.33	Horse	0.00
Banana	0.02	Banana	0.00
Truck	0.02	Truck	0.00

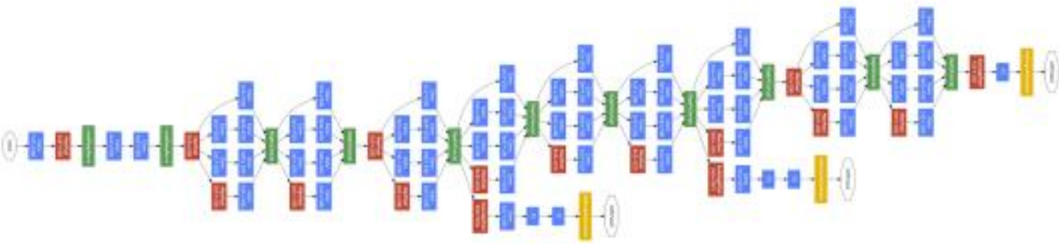
$$\ell_{ce}(w,x) = -\sum_i l(x)_i \cdot \log \frac{e^{f(x)_i}}{\sum_k e^{f(x)_k}}$$



A brief digression on backpropagation

- "Backward propagation of errors" (Rumelhart, Hinton, & Williams 1986)

$f(x)$

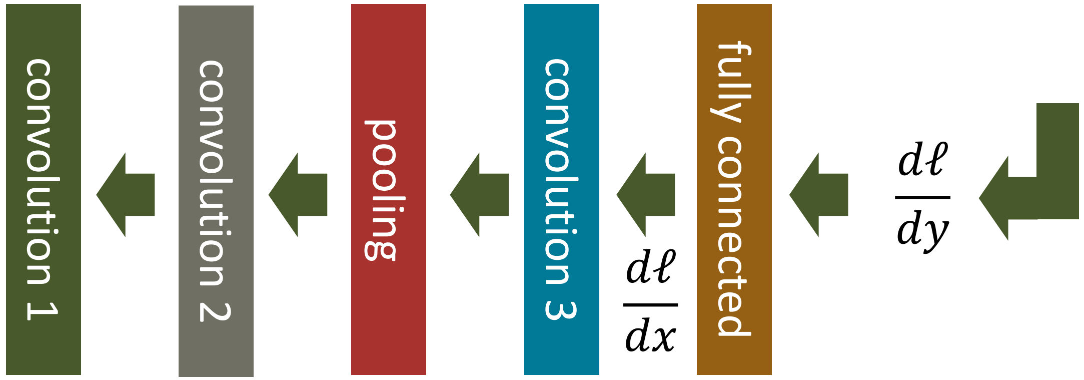


layer-wise parameter update
(backpropagation)

Cat	0.54	Cat	1.00
Dog	0.28	Dog	0.00
Airplane	0.07	Airplane	0.00
Horse	0.33	Horse	0.00
Banana	0.02	Banana	0.00
Truck	0.02	Truck	0.00

$$\ell_{ce}(w,x) = -\sum_i l(x)_i \cdot \log \frac{e^{f(x)_i}}{\sum_k e^{f(x)_k}}$$

$\frac{d\ell}{dw}$

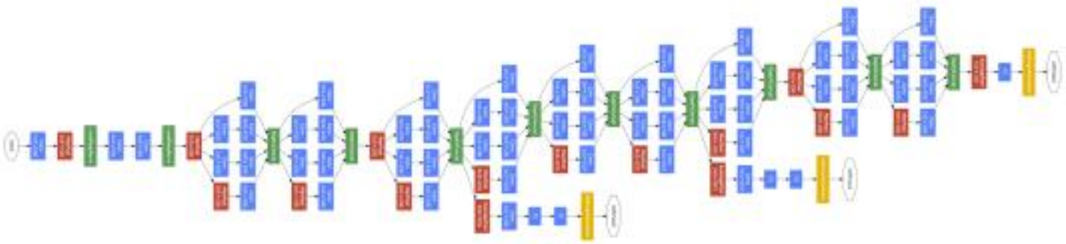


$\frac{d\ell}{dx}$ $\frac{d\ell}{dy}$

A brief digression on backpropagation

- "Backward propagation of errors" (Rumelhart, Hinton, & Williams 1986)

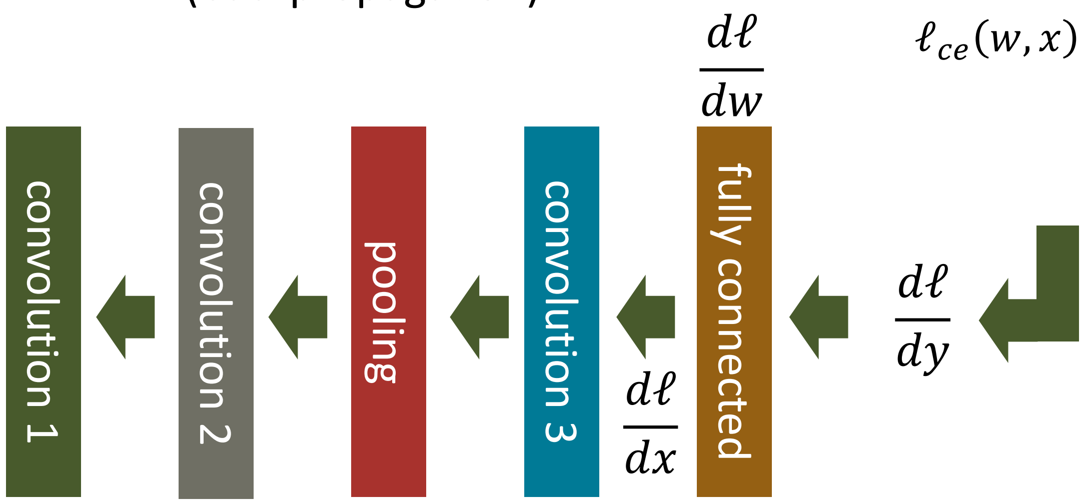
$f(x)$



layer-wise parameter update
(backpropagation)

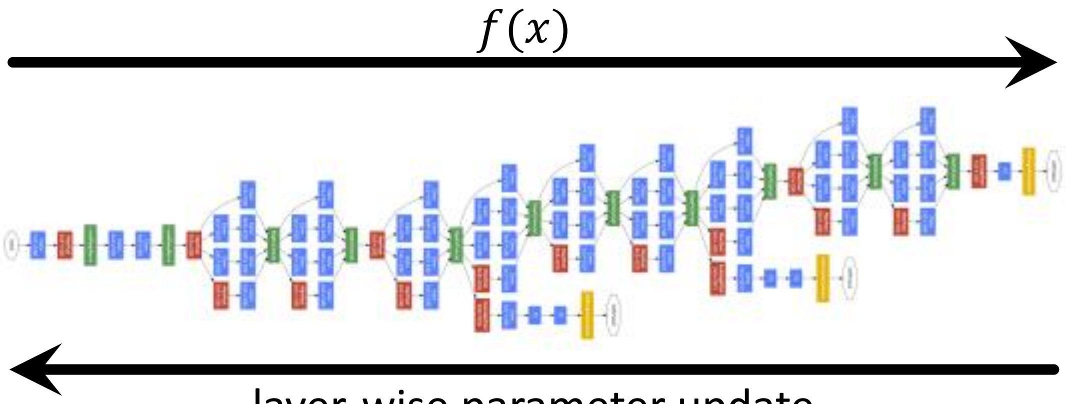
Cat	0.54	Cat	1.00
Dog	0.28	Dog	0.00
Airplane	0.07	Airplane	0.00
Horse	0.33	Horse	0.00
Banana	0.02	Banana	0.00
Truck	0.02	Truck	0.00

$$\ell_{ce}(w,x) = -\sum_i l(x)_i \cdot \log \frac{e^{f(x)_i}}{\sum_k e^{f(x)_k}}$$



A brief digression on backpropagation

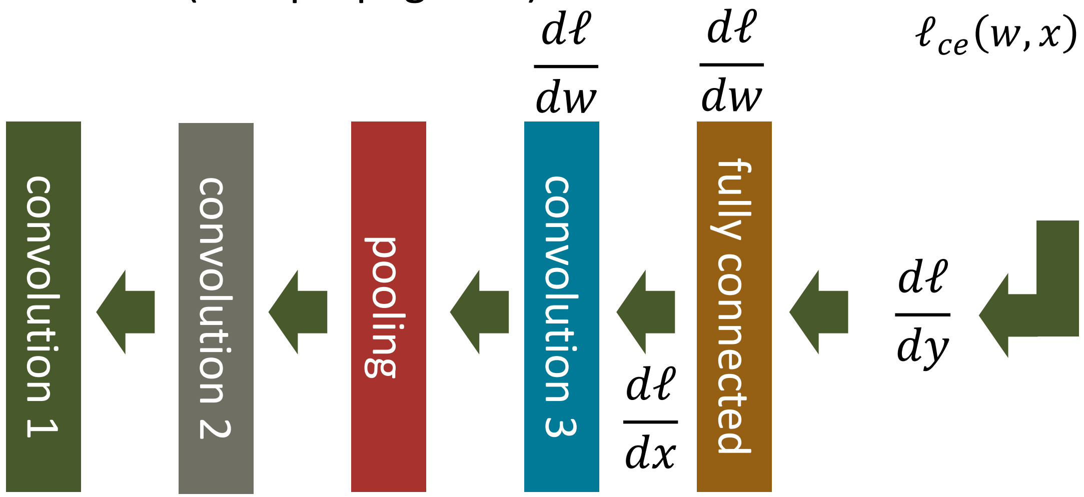
- ”Backward propagation of errors” (Rumelhart, Hinton, & Williams 1986)



layer-wise parameter update
(backpropagation)

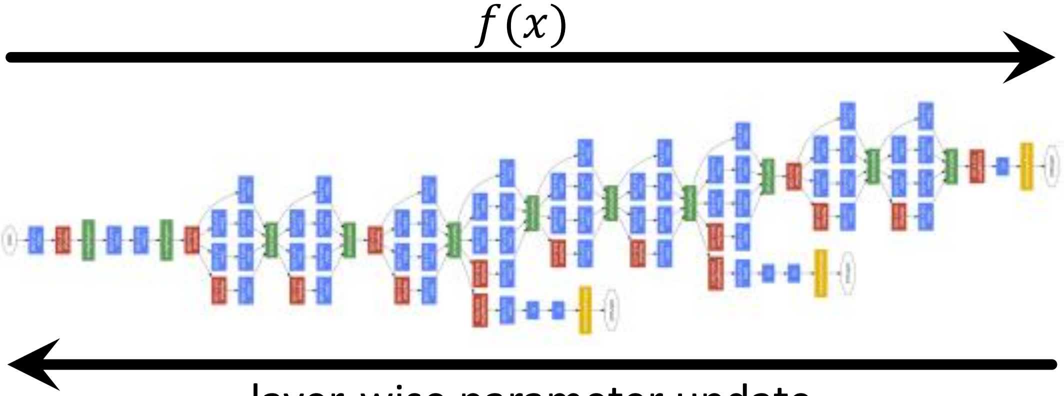
Cat	0.54	Cat	1.00
Dog	0.28	Dog	0.00
Airplane	0.07	Airplane	0.00
Horse	0.33	Horse	0.00
Banana	0.02	Banana	0.00
Truck	0.02	Truck	0.00

$$\ell_{ce}(w,x) = -\sum_i l(x)_i \cdot \log \frac{e^{f(x)_i}}{\sum_k e^{f(x)_k}}$$



A brief digression on backpropagation

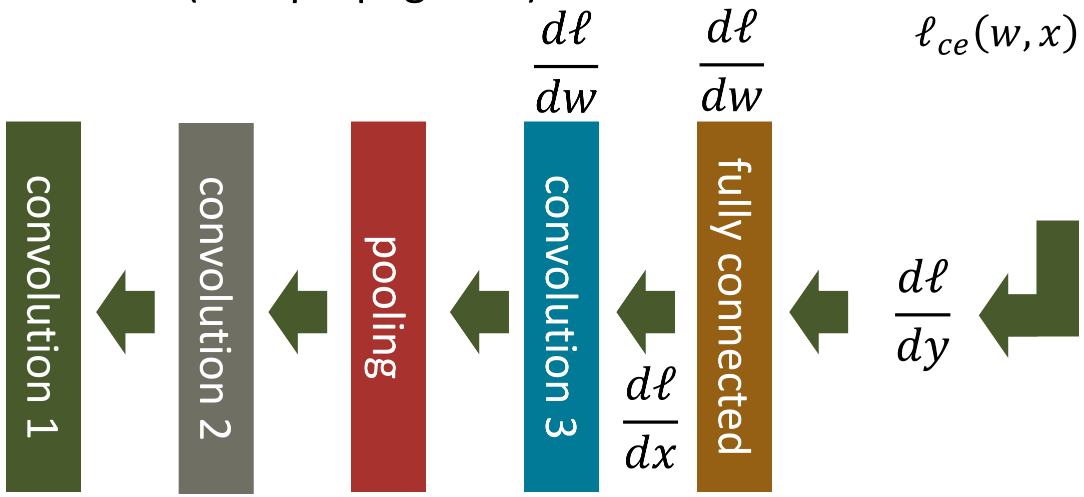
- ”Backward propagation of errors” (Rumelhart, Hinton, & Williams 1986)



layer-wise parameter update
(backpropagation)

Cat	0.54	Cat	1.00
Dog	0.28	Dog	0.00
Airplane	0.07	Airplane	0.00
Horse	0.33	Horse	0.00
Banana	0.02	Banana	0.00
Truck	0.02	Truck	0.00

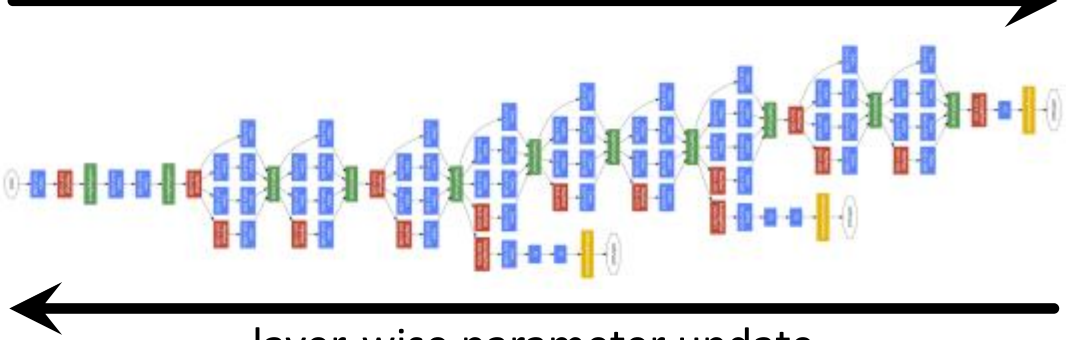
$$\ell_{ce}(w,x) = -\sum_i l(x)_i \cdot \log \frac{e^{f(x)_i}}{\sum_k e^{f(x)_k}}$$



A brief digression on backpropagation

- "Backward propagation of errors" (Rumelhart, Hinton, & Williams 1986)

$f(x)$

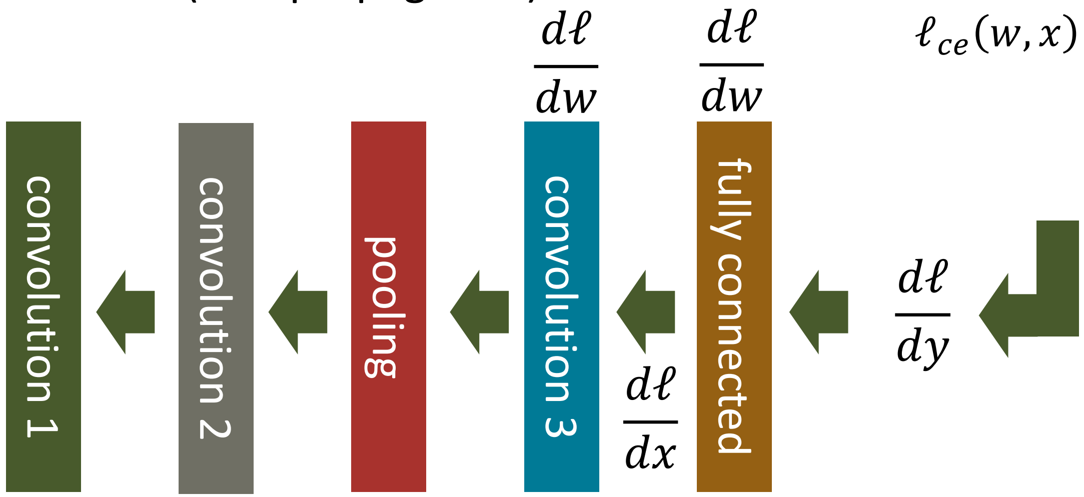


layer-wise parameter update
(backpropagation)

Cat	0.54
Dog	0.28
Airplane	0.07
Horse	0.33
Banana	0.02
Truck	0.02

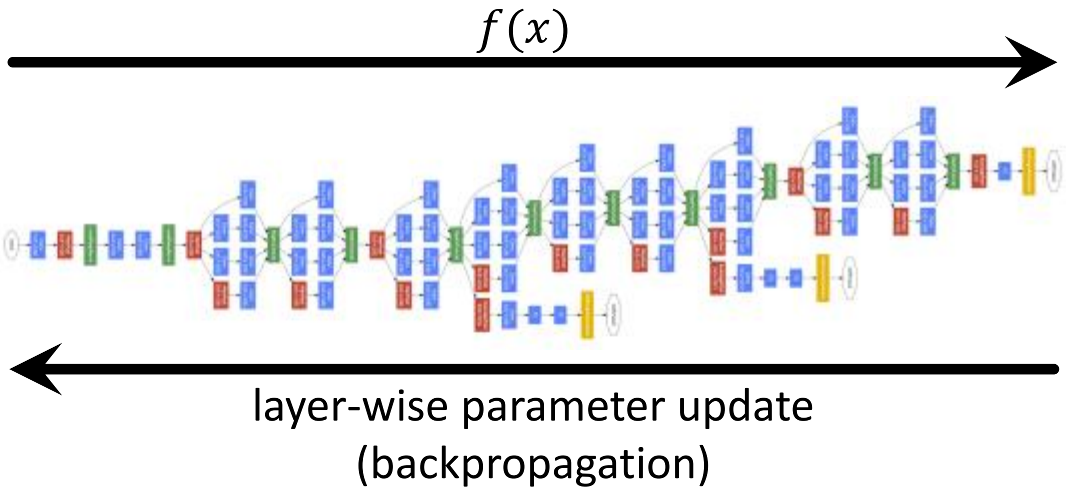
Cat	1.00
Dog	0.00
Airplane	0.00
Horse	0.00
Banana	0.00
Truck	0.00

$$\ell_{ce}(w,x) = -\sum_i l(x)_i \cdot \log \frac{e^{f(x)_i}}{\sum_k e^{f(x)_k}}$$



A brief digression on backpropagation

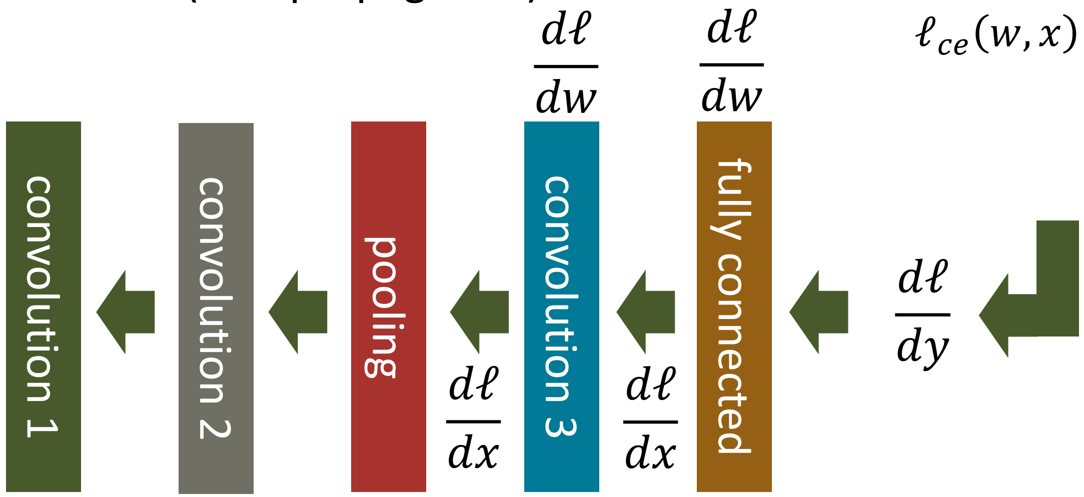
- "Backward propagation of errors" (Rumelhart, Hinton, & Williams 1986)



Cat	0.54
Dog	0.28
Airplane	0.07
Horse	0.33
Banana	0.02
Truck	0.02

Cat	1.00
Dog	0.00
Airplane	0.00
Horse	0.00
Banana	0.00
Truck	0.00

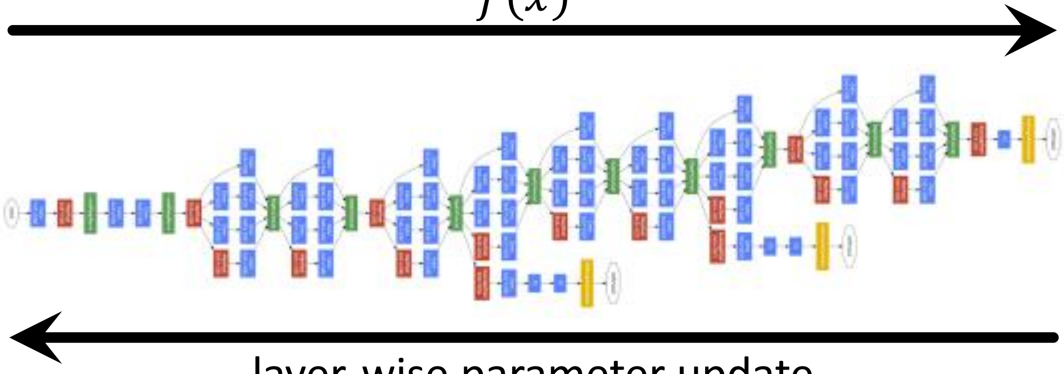
$$\ell_{ce}(w,x) = -\sum_i l(x)_i \cdot \log \frac{e^{f(x)_i}}{\sum_k e^{f(x)_k}}$$



A brief digression on backpropagation

- ”Backward propagation of errors” (Rumelhart, Hinton, & Williams 1986)

$f(x)$

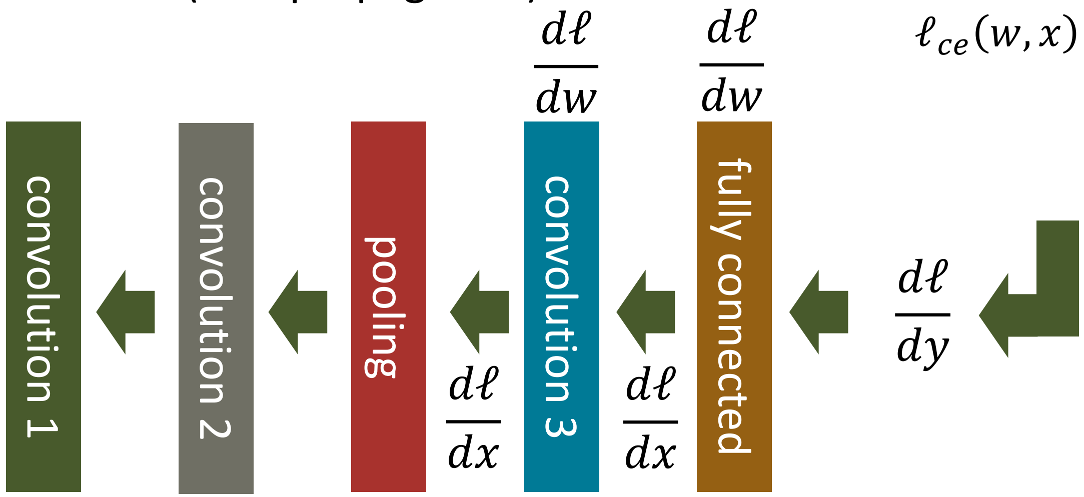


layer-wise parameter update
(backpropagation)

Cat	0.54
Dog	0.28
Airplane	0.07
Horse	0.33
Banana	0.02
Truck	0.02

Cat	1.00
Dog	0.00
Airplane	0.00
Horse	0.00
Banana	0.00
Truck	0.00

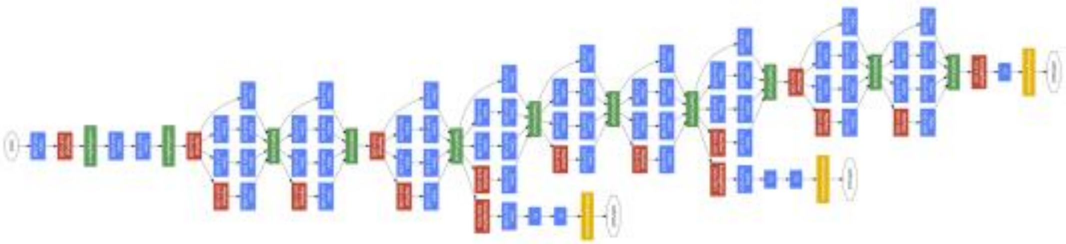
$$\ell_{ce}(w,x) = -\sum_i l(x)_i \cdot \log \frac{e^{f(x)_i}}{\sum_k e^{f(x)_k}}$$



A brief digression on backpropagation

- ”Backward propagation of errors” (Rumelhart, Hinton, & Williams 1986)

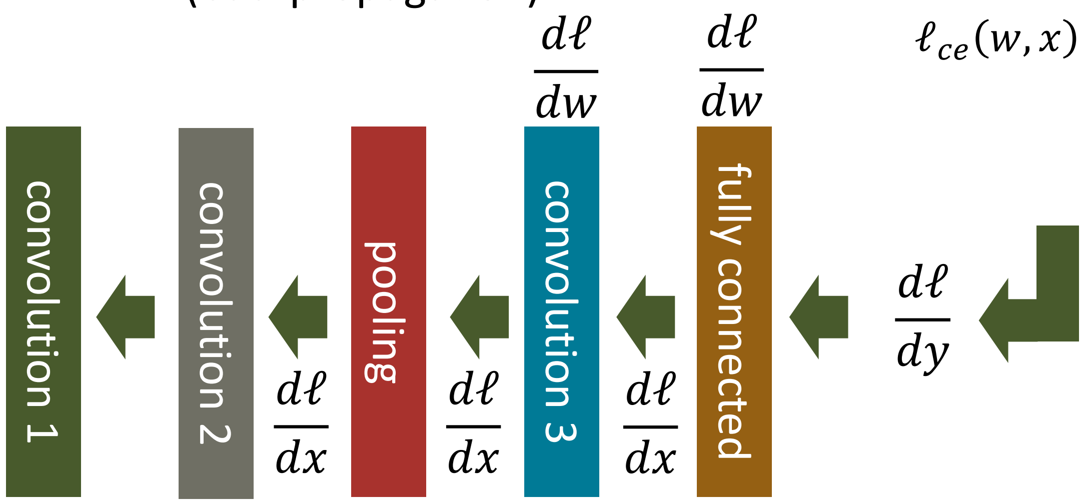
$f(x)$



layer-wise parameter update
(backpropagation)

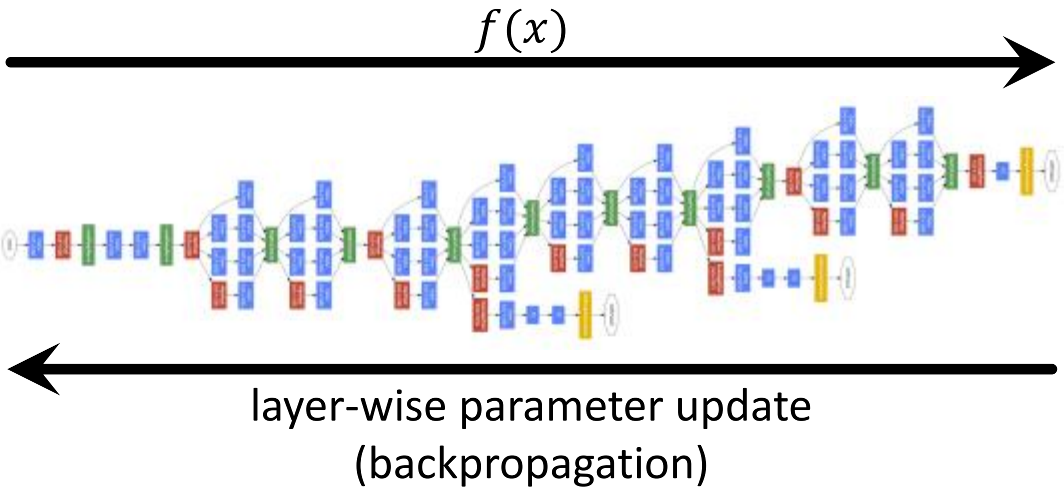
Cat	0.54	Cat	1.00
Dog	0.28	Dog	0.00
Airplane	0.07	Airplane	0.00
Horse	0.33	Horse	0.00
Banana	0.02	Banana	0.00
Truck	0.02	Truck	0.00

$$\ell_{ce}(w,x) = -\sum_i l(x)_i \cdot \log \frac{e^{f(x)_i}}{\sum_k e^{f(x)_k}}$$



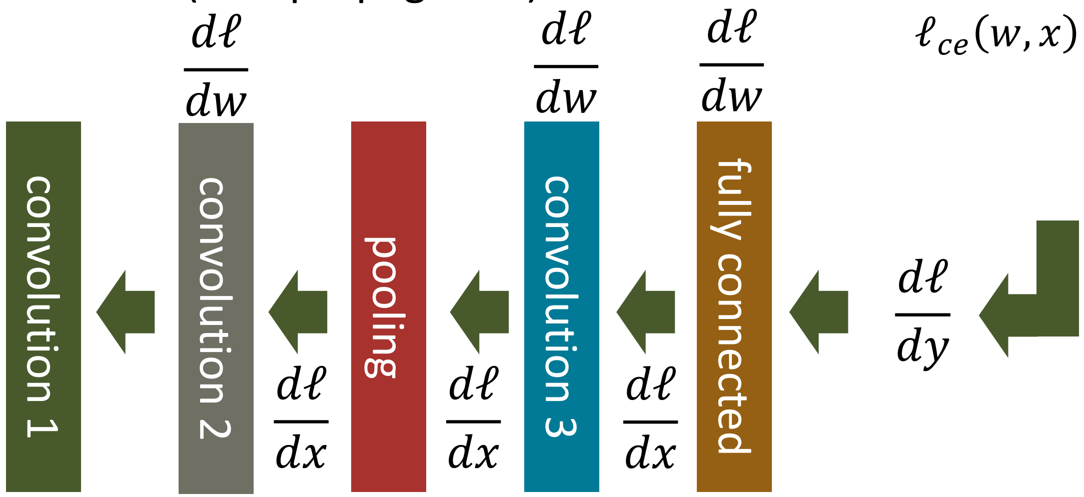
A brief digression on backpropagation

- ”Backward propagation of errors” (Rumelhart, Hinton, & Williams 1986)



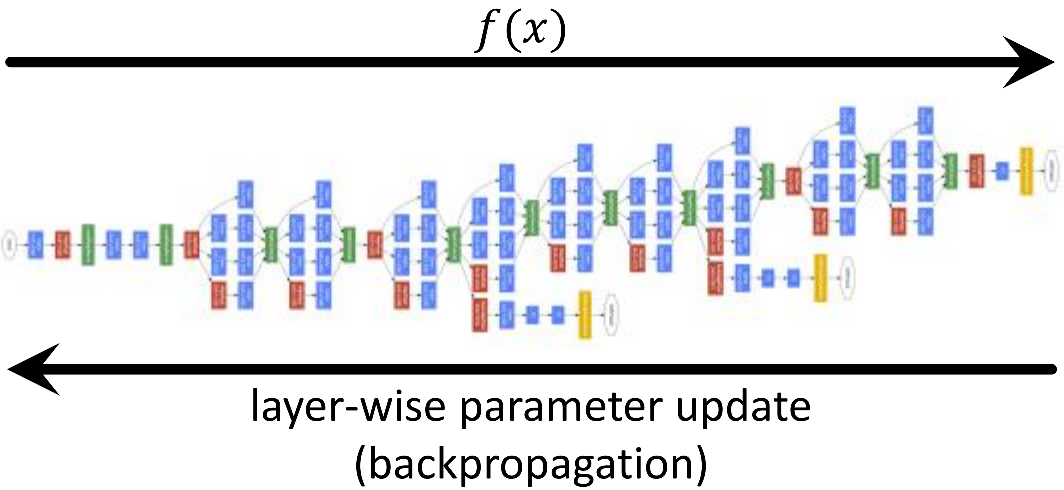
Cat	0.54	Cat	1.00
Dog	0.28	Dog	0.00
Airplane	0.07	Airplane	0.00
Horse	0.33	Horse	0.00
Banana	0.02	Banana	0.00
Truck	0.02	Truck	0.00

$$\ell_{ce}(w,x) = -\sum_i l(x)_i \cdot \log \frac{e^{f(x)_i}}{\sum_k e^{f(x)_k}}$$



A brief digression on backpropagation

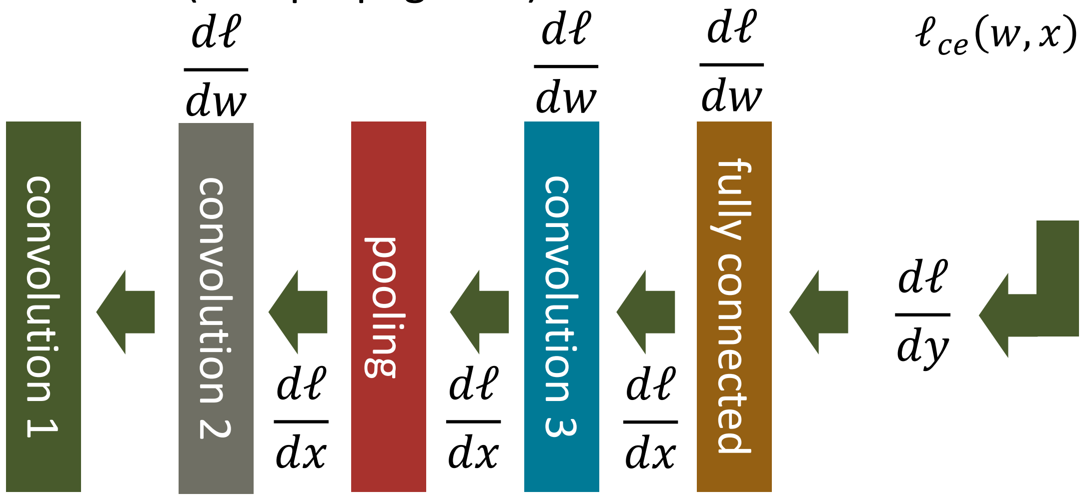
- ”Backward propagation of errors” (Rumelhart, Hinton, & Williams 1986)



Cat	0.54
Dog	0.28
Airplane	0.07
Horse	0.33
Banana	0.02
Truck	0.02

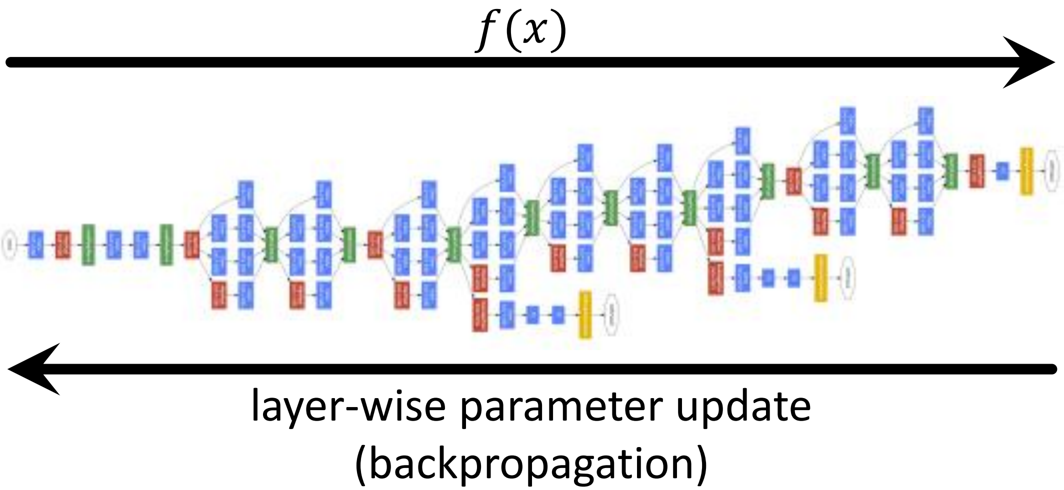
Cat	1.00
Dog	0.00
Airplane	0.00
Horse	0.00
Banana	0.00
Truck	0.00

$$\ell_{ce}(w,x) = -\sum_i l(x)_i \cdot \log \frac{e^{f(x)_i}}{\sum_k e^{f(x)_k}}$$



A brief digression on backpropagation

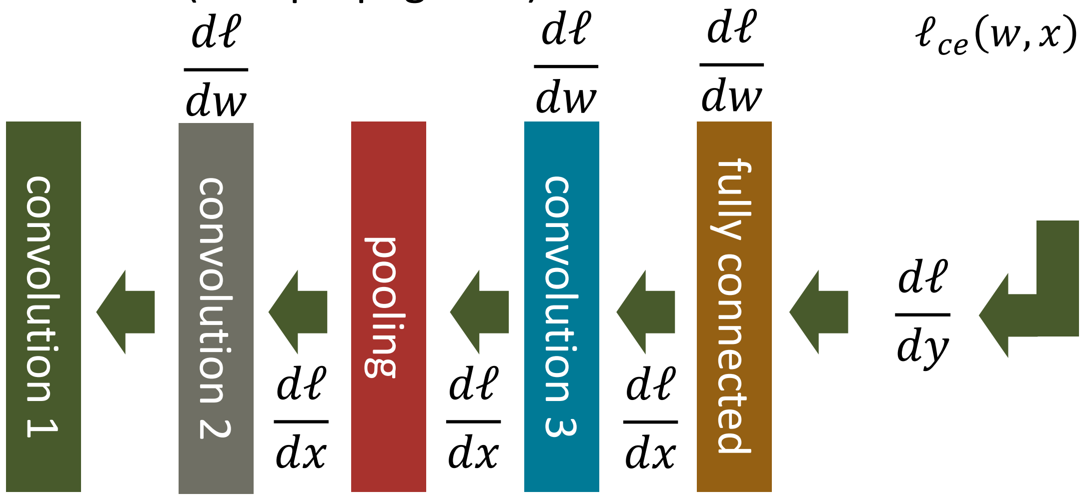
- ”Backward propagation of errors” (Rumelhart, Hinton, & Williams 1986)



Cat	0.54
Dog	0.28
Airplane	0.07
Horse	0.33
Banana	0.02
Truck	0.02

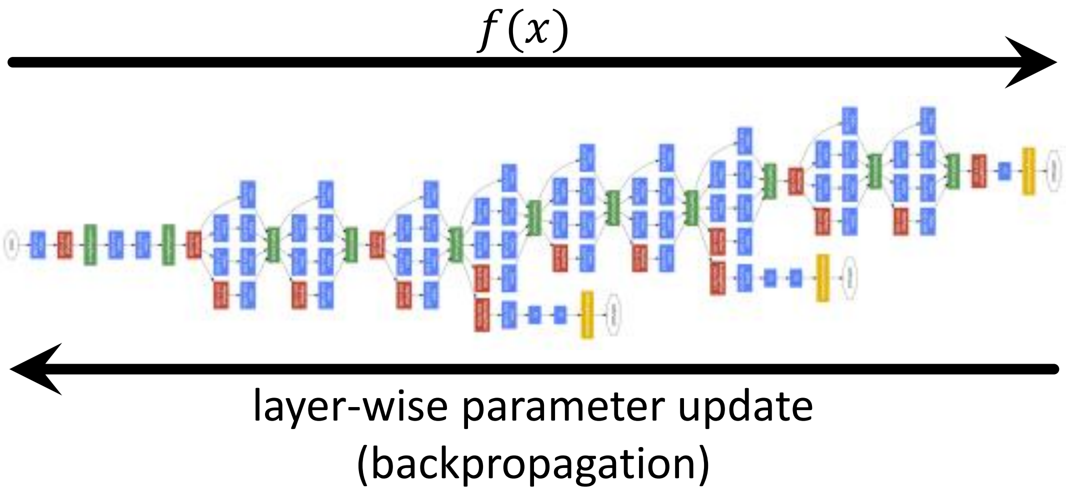
Cat	1.00
Dog	0.00
Airplane	0.00
Horse	0.00
Banana	0.00
Truck	0.00

$$\ell_{ce}(w,x) = -\sum_i l(x)_i \cdot \log \frac{e^{f(x)_i}}{\sum_k e^{f(x)_k}}$$



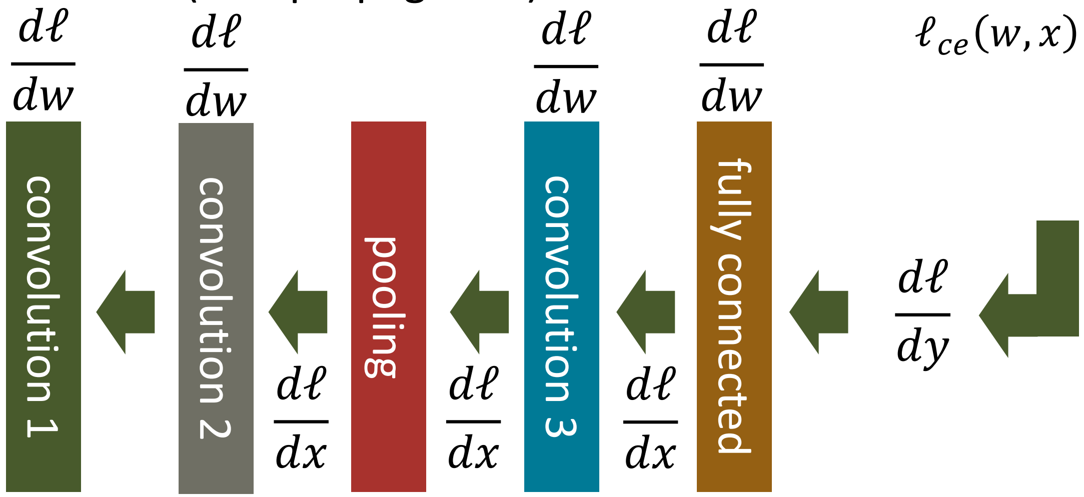
A brief digression on backpropagation

- "Backward propagation of errors" (Rumelhart, Hinton, & Williams 1986)



Cat	0.54	Cat	1.00
Dog	0.28	Dog	0.00
Airplane	0.07	Airplane	0.00
Horse	0.33	Horse	0.00
Banana	0.02	Banana	0.00
Truck	0.02	Truck	0.00

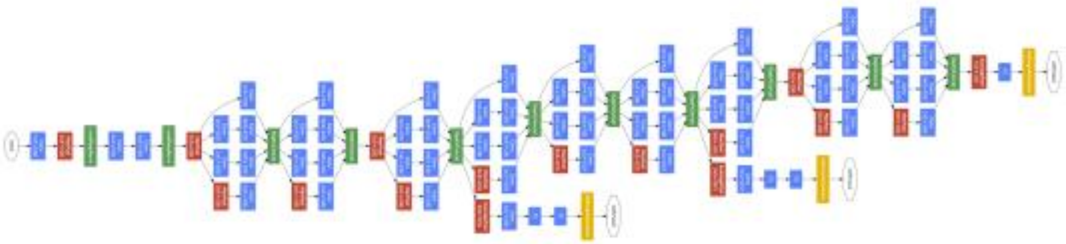
$$\ell_{ce}(w,x) = -\sum_i l(x)_i \cdot \log \frac{e^{f(x)_i}}{\sum_k e^{f(x)_k}}$$



A brief digression on backpropagation

- "Backward propagation of errors" (Rumelhart, Hinton, & Williams 1986)

$f(x)$

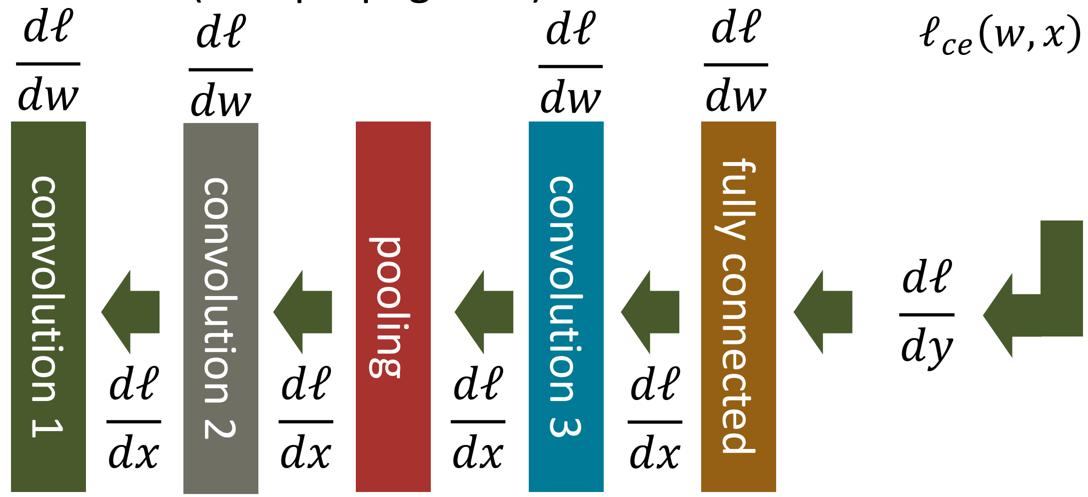


layer-wise parameter update
(backpropagation)

Cat	0.54
Dog	0.28
Airplane	0.07
Horse	0.33
Banana	0.02
Truck	0.02

Cat	1.00
Dog	0.00
Airplane	0.00
Horse	0.00
Banana	0.00
Truck	0.00

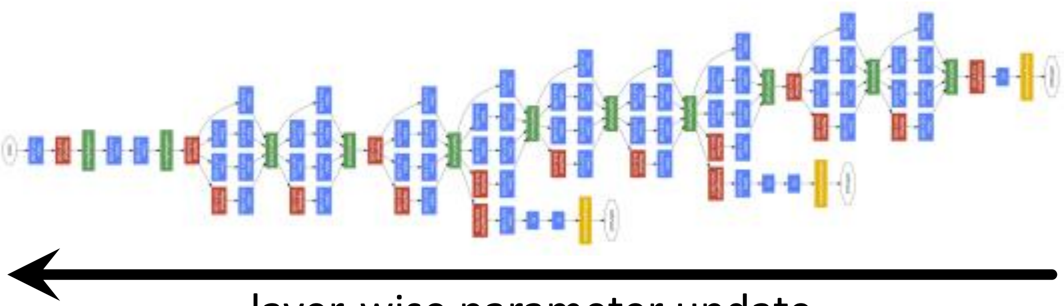
$$\ell_{ce}(w,x) = -\sum_i l(x)_i \cdot \log \frac{e^{f(x)_i}}{\sum_k e^{f(x)_k}}$$



A brief digression on backpropagation

- ”Backward propagation of errors” (Rumelhart, Hinton, & Williams 1986)

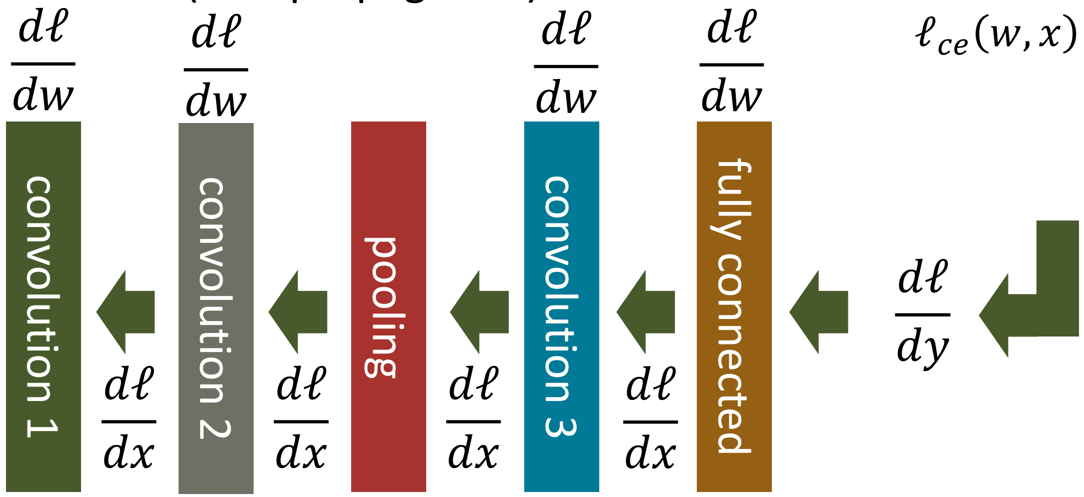
$f(x)$



layer-wise parameter update (backpropagation)

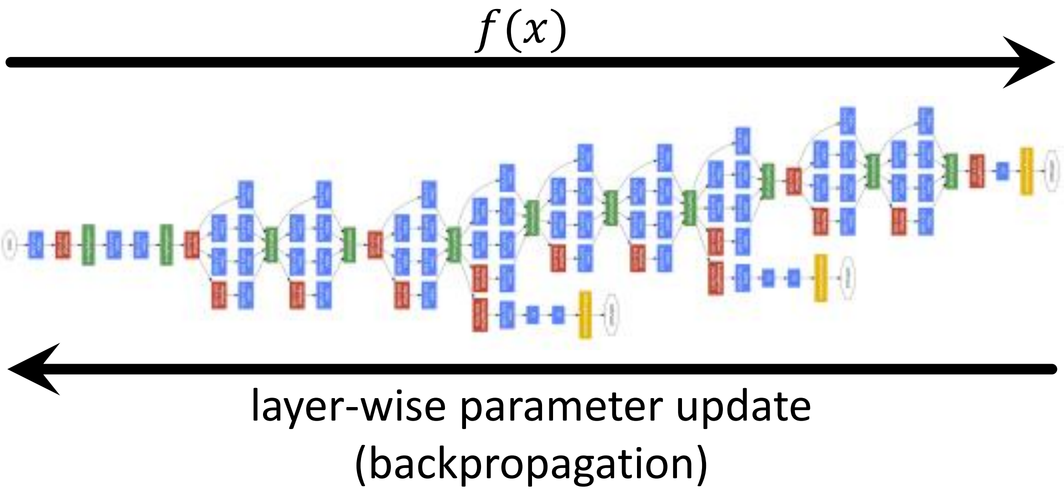
Cat	0.54	Cat	1.00
Dog	0.28	Dog	0.00
Airplane	0.07	Airplane	0.00
Horse	0.33	Horse	0.00
Banana	0.02	Banana	0.00
Truck	0.02	Truck	0.00

$$\ell_{ce}(w,x) = -\sum_i l(x)_i \cdot \log \frac{e^{f(x)_i}}{\sum_k e^{f(x)_k}}$$



A brief digression on backpropagation

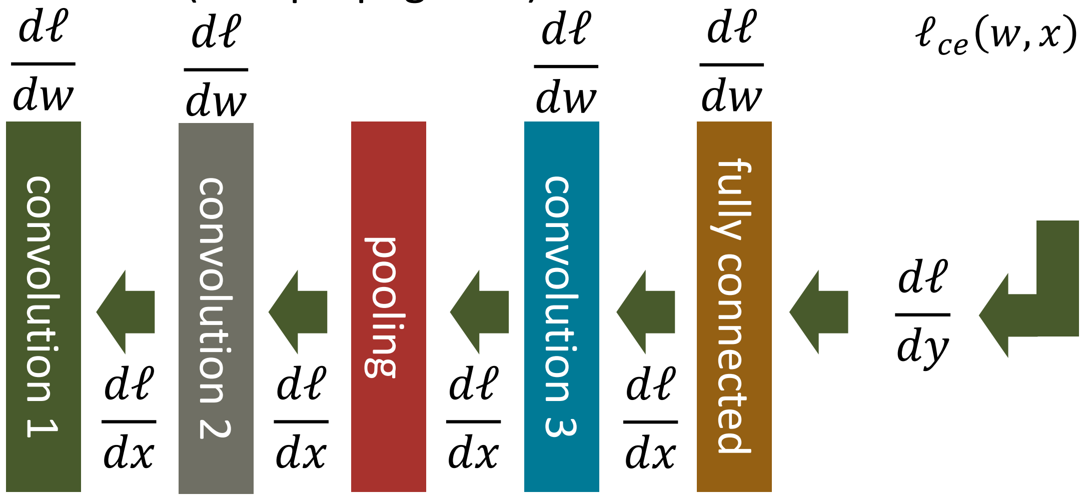
- ”Backward propagation of errors” (Rumelhart, Hinton, & Williams 1986)



Cat	0.54
Dog	0.28
Airplane	0.07
Horse	0.33
Banana	0.02
Truck	0.02

Cat	1.00
Dog	0.00
Airplane	0.00
Horse	0.00
Banana	0.00
Truck	0.00

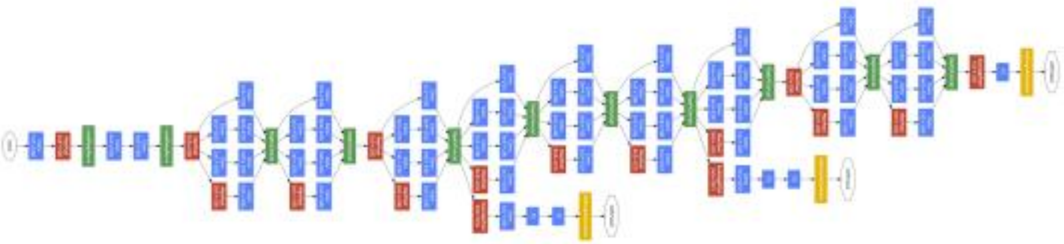
$$\ell_{ce}(w,x) = -\sum_i l(x)_i \cdot \log \frac{e^{f(x)_i}}{\sum_k e^{f(x)_k}}$$



A brief digression on backpropagation

- "Backward propagation of errors" (Rumelhart, Hinton, & Williams 1986)

$f(x)$

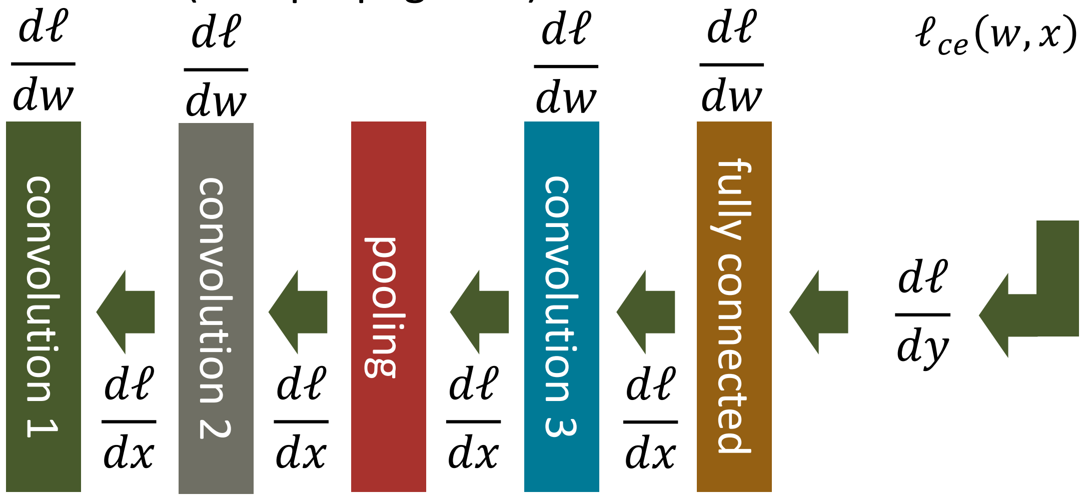


layer-wise parameter update
(backpropagation)

Cat	0.54
Dog	0.28
Airplane	0.07
Horse	0.33
Banana	0.02
Truck	0.02

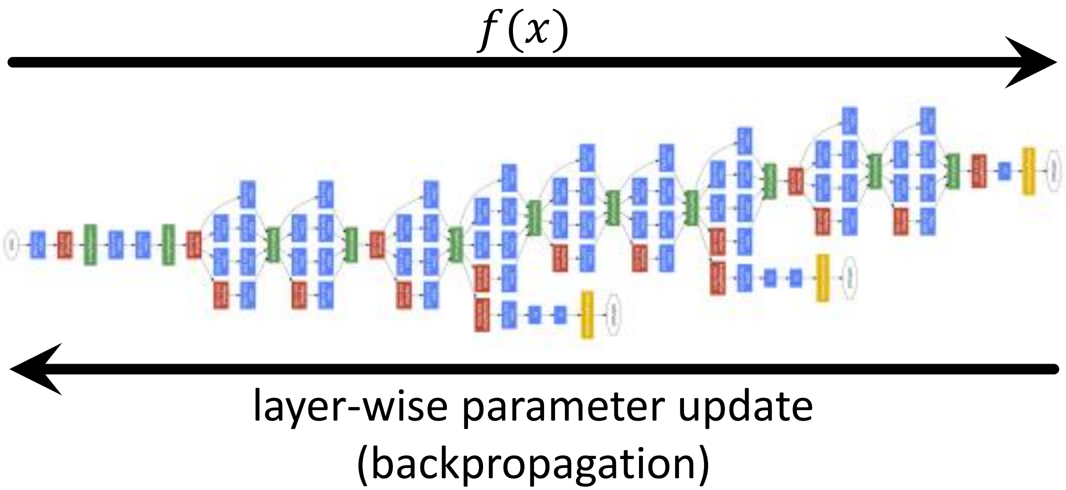
Cat	1.00
Dog	0.00
Airplane	0.00
Horse	0.00
Banana	0.00
Truck	0.00

$$\ell_{ce}(w,x) = -\sum_i l(x)_i \cdot \log \frac{e^{f(x)_i}}{\sum_k e^{f(x)_k}}$$



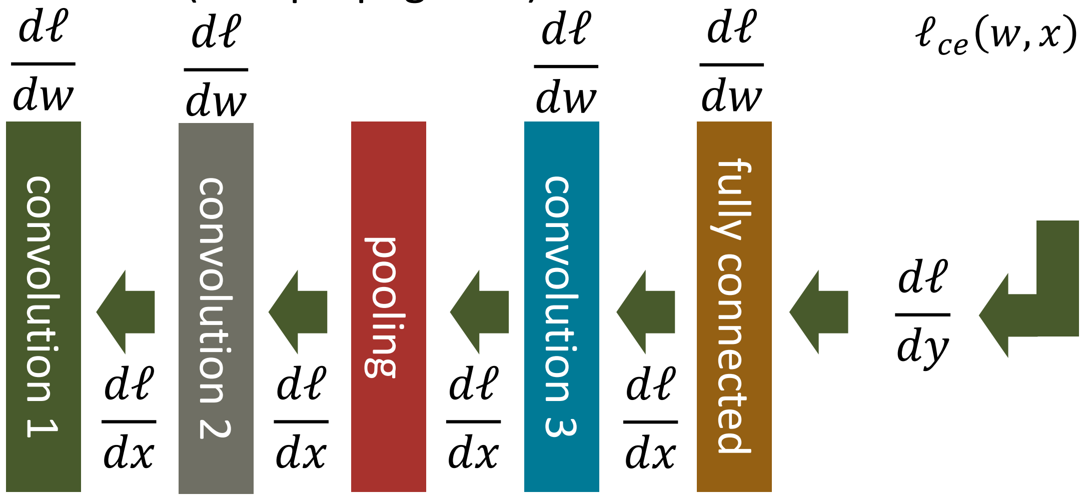
A brief digression on backpropagation

- "Backward propagation of errors" (Rumelhart, Hinton, & Williams 1986)



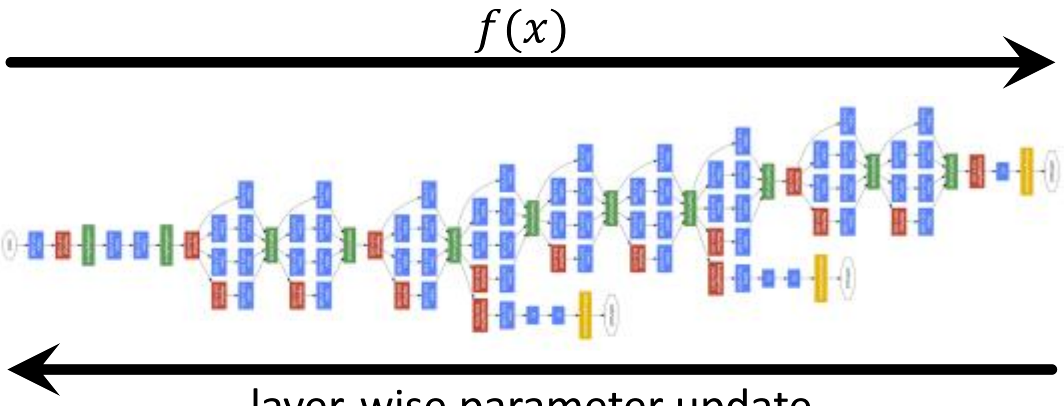
Cat	0.54	Cat	1.00
Dog	0.28	Dog	0.00
Airplane	0.07	Airplane	0.00
Horse	0.33	Horse	0.00
Banana	0.02	Banana	0.00
Truck	0.02	Truck	0.00

$$\ell_{ce}(w,x) = -\sum_i l(x)_i \cdot \log \frac{e^{f(x)_i}}{\sum_k e^{f(x)_k}}$$



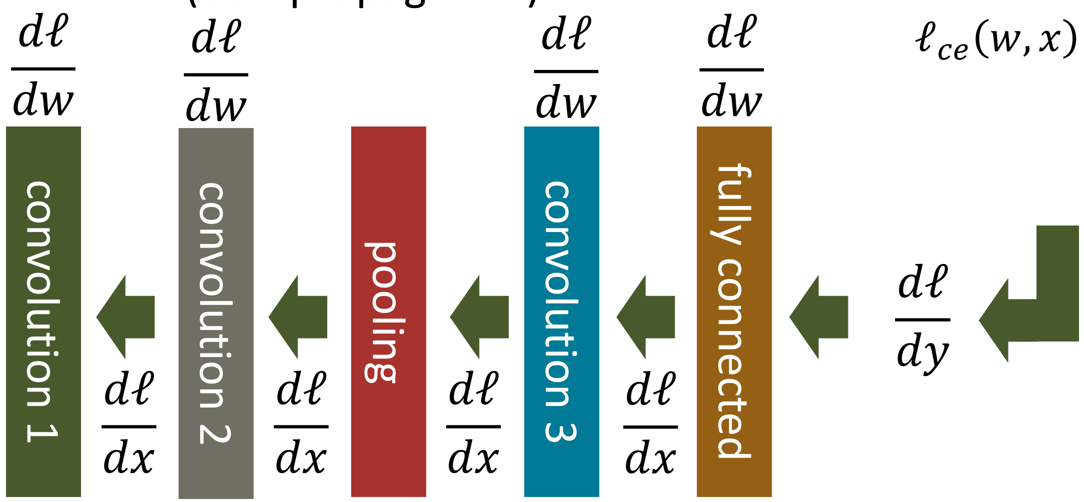
A brief digression on backpropagation

- "Backward propagation of errors" (Rumelhart, Hinton, & Williams 1986)



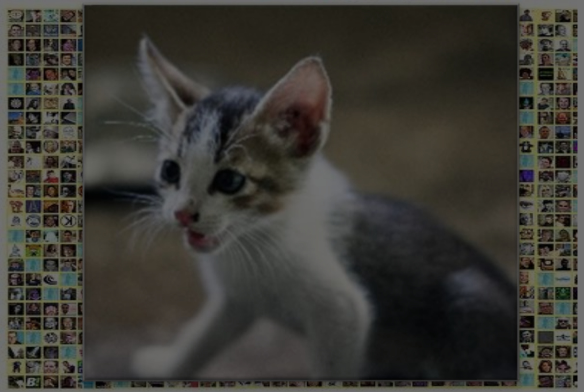
Cat	0.54	Cat	1.00
Dog	0.28	Dog	0.00
Airplane	0.07	Airplane	0.00
Horse	0.33	Horse	0.00
Banana	0.02	Banana	0.00
Truck	0.02	Truck	0.00

$$\ell_{ce}(w,x) = -\sum_i l(x)_i \cdot \log \frac{e^{f(x)_i}}{\sum_k e^{f(x)_k}}$$

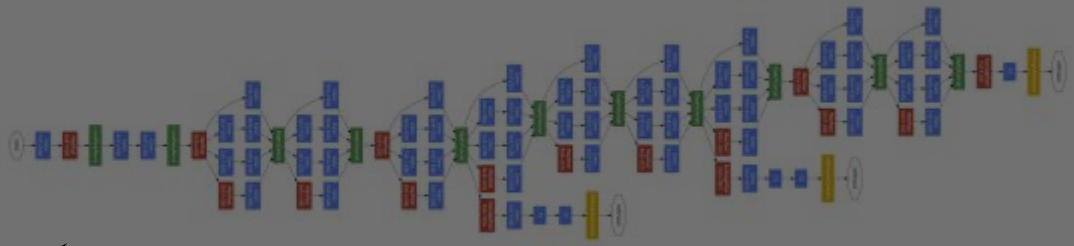


A brief digression on backpropagation

- ”Backward propagation of errors” (Rumelhart, Hinton, & Williams 1986)



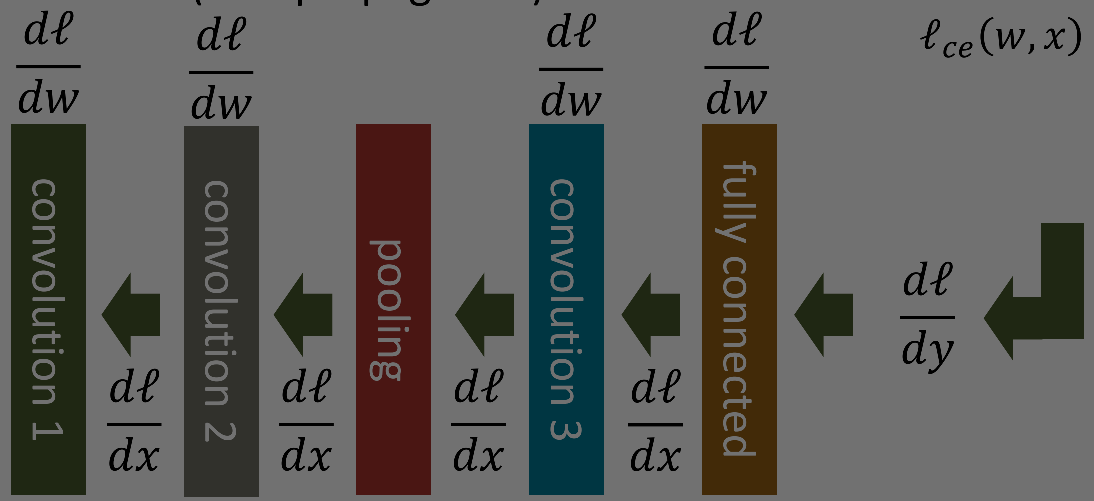
$f(x)$



Cat	0.54
Dog	0.28
Airplane	0.07
Horse	0.33
Banana	0.02
Truck	0.02

Cat	1.00
Dog	0.00
Airplane	0.00
Horse	0.00
Banana	0.00
Truck	0.00

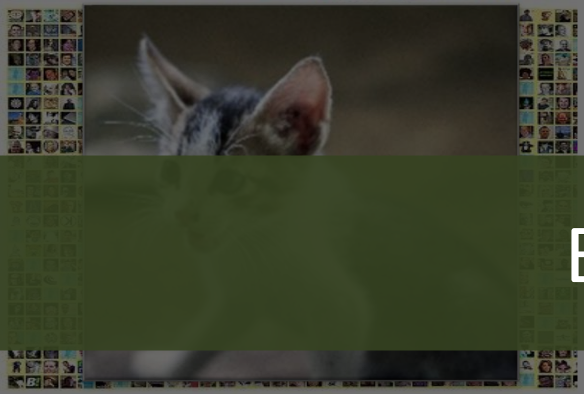
layer-wise parameter update
(backpropagation)



$$\ell_{ce}(w,x) = -\sum_i l(x)_i \cdot \log \frac{e^{f(x)_i}}{\sum_k e^{f(x)_k}}$$

A brief digression on backpropagation

- "Backward propagation of errors" (Rumelhart, Hinton, & Williams 1986)

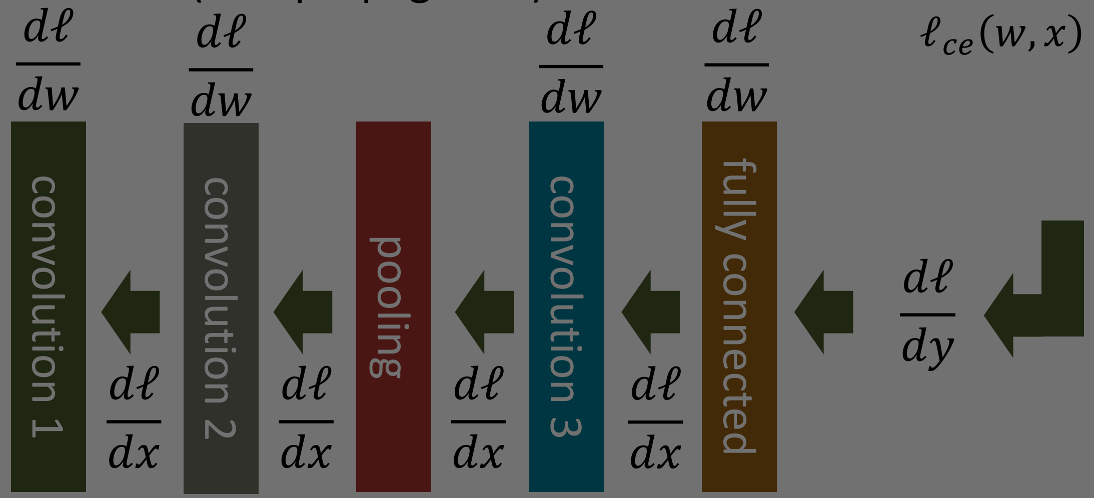


$f(x)$

Cat	0.54	Cat	1.00
Dog	0.28	Dog	0.00
Airplane	0.07	Airplane	0.00
Horse	0.33	Horse	0.00
Banana	0.02	Banana	0.00
Truck	0.02	Truck	0.00

Backpropagation is just the chain rule!

layer-wise parameter update (backpropagation)



$$\ell_{ce}(w,x) = -\sum_i l(x)_i \cdot \log \frac{e^{f(x)_i}}{\sum_k e^{f(x)_k}}$$

Stochastic Gradient Descent

$$w^* = \operatorname{argmin}_{w \in \mathbb{R}^d} \mathbb{E}_{x \sim \mathcal{D}}[\ell(w, x)]$$

for $t = 0$ **to** T **do** ▷ Stopping condition
  $x \leftarrow$ Random element from X ▷ Sample dataset
  $o_1 \leftarrow f_1(x)$
  for $i = 2$ **to** *layers* **do** ▷ Forward evaluation of ℓ
    $o_i \leftarrow f_i(o_{i-1})$
  end for
  for $i =$ *layers*-1 **to** 1 **do** ▷ Compute gradient of ℓ via backpropagation
    $\nabla o_i \leftarrow \frac{\partial \ell}{\partial o_i}(o_{i-1}, o_i, \nabla o_{i+1})$ ▷ Gradient w.r.t. data
    $\nabla w_i^{(t)} \leftarrow \frac{\partial \ell}{\partial w_i}(o_{i-1}, o_i, \nabla o_{i+1})$ ▷ Gradient w.r.t. layer parameters
  end for
  $w^{(t+1)} \leftarrow w^{(t)} + u(\nabla w^{(t)}, w^{(0,\ldots,t)}, t)$ ▷ Update weights with function u
end for

$f_1(x)$ convolution 1

$f_2(f_1(x))$ convolution 2

pooling

…

convolution 3

$f(x)$ fully connected

T. Ben-Nun, T. Hoefler: Demystifying Parallel and Distributed Deep Learning: An In-Depth Concurrency Analysis, CSUR 2019

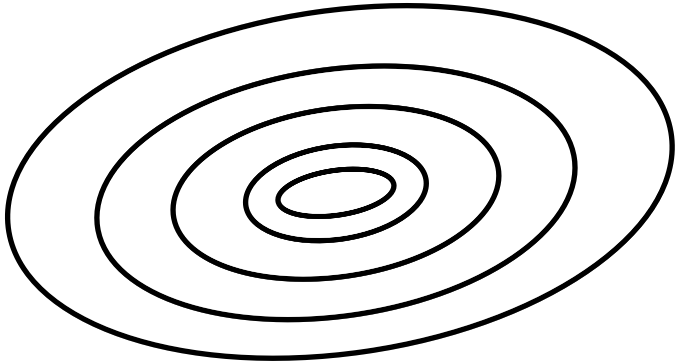
Stochastic Gradient Descent

$$w^* = \operatorname{argmin}_{w \in \mathbb{R}^d} \mathbb{E}_{x \sim \mathcal{D}}[\ell(w, x)]$$

```
for t = 0 to T do                                          ▷ Stopping condition
    x ← Random element from X                              ▷ Sample dataset
    o_1 ← f_1(x)
    for i = 2 to layers do                                 ▷ Forward evaluation of ℓ
        o_i ← f_i(o_{i-1})
    end for
    for i = layers−1 to 1 do                               ▷ Compute gradient of ℓ via backpropagation
        ∇o_i ← ∂ℓ/∂o_i (o_{i-1}, o_i, ∇o_{i+1})            ▷ Gradient w.r.t. data
        ∇w_i^(t) ← ∂ℓ/∂w_i (o_{i-1}, o_i, ∇o_{i+1})        ▷ Gradient w.r.t. layer parameters
    end for
    w^(t+1) ← w^(t) + u(∇w^(t), w^(0,...,t), t)            ▷ Update weights with function u
end for
```

$f_1(x)$	convolution 1
$f_2(f_1(x))$	convolution 2
	pooling
…	
	convolution 3
$f(x)$	fully connected

- Layer storage = $|w_l| + |f_l(o_{l-1})| + |\nabla w_l| + |\nabla o_l|$



Stochastic Gradient Descent

$$w^* = \text{argmin}_{w \in \mathbb{R}^d} \mathbb{E}_{x \sim \mathcal{D}}[\ell(w,x)]$$

for $t = 0$ **to** T **do** ▹ Stopping condition
 $x \leftarrow$ Random element from X ▹ Sample dataset
 $o_1 \leftarrow f_1(x)$
 for $i = 2$ **to** *layers* **do** ▹ Forward evaluation of ℓ
  $o_i \leftarrow f_i(o_{i-1})$
 end for
 for $i =$ *layers*-1 **to** 1 **do** ▹ Compute gradient of ℓ via backpropagation
  $\nabla o_i \leftarrow \frac{\partial \ell}{\partial o_i}(o_{i-1}, o_i, \nabla o_{i+1})$ ▹ Gradient w.r.t. data
  $\nabla w_i^{(t)} \leftarrow \frac{\partial \ell}{\partial w_i}(o_{i-1}, o_i, \nabla o_{i+1})$ ▹ Gradient w.r.t. layer parameters
 end for
 $w^{(t+1)} \leftarrow w^{(t)} + u(\nabla w^{(t)}, w^{(0,\ldots,t)}, t)$ ▹ Update weights with function u
end for

$f_1(x)$ convolution 1

$f_2(f_1(x))$ convolution 2

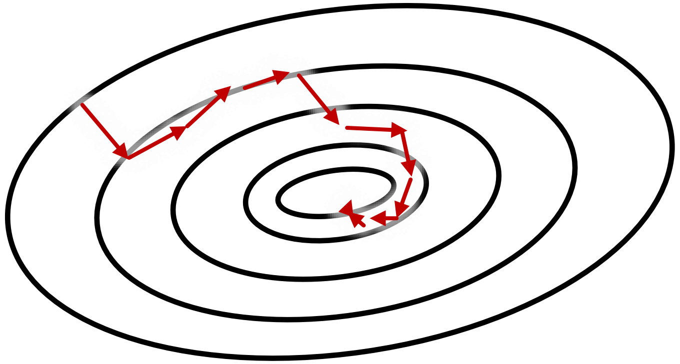
pooling

...

convolution 3

$f(x)$ fully connected

- Layer storage = $|w_l| + |f_l(o_{l-1})| + |\nabla w_l| + |\nabla o_l|$



T. Ben-Nun, T. Hoefler: Demystifying Parallel and Distributed Deep Learning: An In-Depth Concurrency Analysis, CSUR 2019

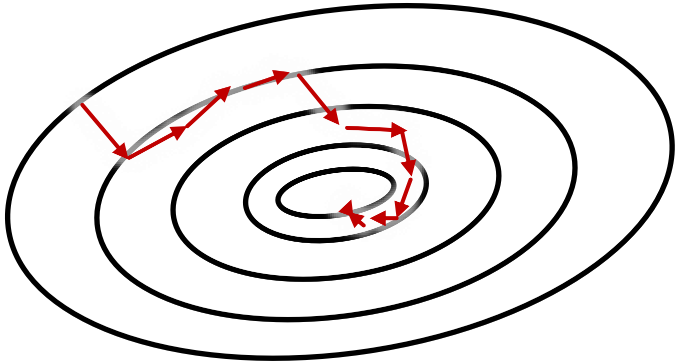
Stochastic Gradient Descent

$$w^* = \operatorname{argmin}_{w \in \mathbb{R}^d} \mathbb{E}_{x \sim \mathcal{D}}[\ell(w, x)]$$

```
for t = 0 to T do                                        ▷ Stopping condition
    x ← Random element from X                            ▷ Sample dataset
    o_1 ← f_1(x)
    for i = 2 to layers do                               ▷ Forward evaluation of ℓ
        o_i ← f_i(o_{i−1})
    end for
    for i = layers−1 to 1 do                             ▷ Compute gradient of ℓ via backpropagation
        ∇o_i ← ∂ℓ/∂o_i (o_{i−1}, o_i, ∇o_{i+1})          ▷ Gradient w.r.t. data
        ∇w_i^(t) ← ∂ℓ/∂w_i (o_{i−1}, o_i, ∇o_{i+1})      ▷ Gradient w.r.t. layer parameters
    end for
    w^(t+1) ← w^(t) + u(∇w^(t), w^(0,...,t), t)          ▷ Update weights with function u
end for
```

- $f_1(x)$ — convolution 1
- $f_2(f_1(x))$ — convolution 2
- pooling
- …
- convolution 3
- $f(x)$ — fully connected

- Layer storage = $|w_l| + |f_l(o_{l-1})| + |\nabla w_l| + |\nabla o_l|$



Learning Rate	$w^{(t+1)} = w^{(t)} - \eta \cdot \nabla\ell(w^{(t)}, z) \quad = w^{(t)} - \eta \cdot \nabla w^{(t)}$
Adaptive Learning Rate	$w^{(t+1)} = w^{(t)} - \eta_t \cdot \nabla w^{(t)}$
Momentum [Qian 1999]	$w^{(t+1)} = w^{(t)} + \mu \cdot (w^{(t)} - w^{(t-1)}) - \eta \cdot \nabla w^{(t)}$
Nesterov Momentum [Nesterov 1983]	$w^{(t+1)} = w^{(t)} + v_t; \quad v_{t+1} = \mu \cdot v_t - \eta \cdot \nabla\ell(w^{(t)} - \mu \cdot v_t, z)$
AdaGrad [Duchi et al. 2011]	$w_i^{(t+1)} = w_i^{(t)} - \frac{\eta \cdot \nabla w_i^{(t)}}{\sqrt{A_{i,t} + \varepsilon}}; \quad A_{i,t} = \sum_{\tau=0}^{t} \left(\nabla w_i^{(t)}\right)^2$
RMSProp [Hinton 2012]	$w_i^{(t+1)} = w_i^{(t)} - \frac{\eta \cdot \nabla w_i^{(t)}}{\sqrt{A'_{i,t} + \varepsilon}}; \quad A'_{i,t} = \beta \cdot A'_{t-1} + (1-\beta)\left(\nabla w_i^{(t)}\right)^2$
Adam [Kingma and Ba 2015]	$w_i^{(t+1)} = w_i^{(t)} - \frac{\eta \cdot M_{i,t}^{(1)}}{\sqrt{M_{i,t}^{(2)} + \varepsilon}}; \quad M_{i,t}^{(m)} = \frac{\beta_m \cdot M_{i,t-1}^{(m)} + (1-\beta_m)\left(\nabla w_i^{(t)}\right)^m}{1-\beta_m^t}$

T. Ben-Nun, T. Hoefler: Demystifying Parallel and Distributed Deep Learning: An In-Depth Concurrency Analysis, CSUR 2019

Stochastic Gradient Descent

$$w^* = \mathrm{argmin}_{w \in \mathbb{R}^d} \mathbb{E}_{x \sim \mathcal{D}}[\ell(w, x)]$$

for $t = 0$ **to** T **do** ▷ Stopping condition
  $x \leftarrow$ Random element from X ▷ Sample dataset
  $o_1 \leftarrow f_1(x)$
  for $i = 2$ **to** *layers* **do** ▷ Forward evaluation of ℓ
    $o_i \leftarrow f_i(o_{i-1})$
  end for
  for $i =$ *layers*-1 **to** 1 **do** ▷ Compute gradient of ℓ via backpropagation
    $\nabla o_i \leftarrow \frac{\partial \ell}{\partial o_i}(o_{i-1}, o_i, \nabla o_{i+1})$ ▷ Gradient w.r.t. data
    $\nabla w_i^{(t)} \leftarrow \frac{\partial \ell}{\partial w_i}(o_{i-1}, o_i, \nabla o_{i+1})$ ▷ Gradient w.r.t. layer parameters
  end for
  $w^{(t+1)} \leftarrow w^{(t)} + u(\nabla w^{(t)}, w^{(0,\ldots,t)}, t)$ ▷ Update weights with function u
end for

$f_1(x)$ convolution 1

$f_2(f_1(x))$ convolution 2

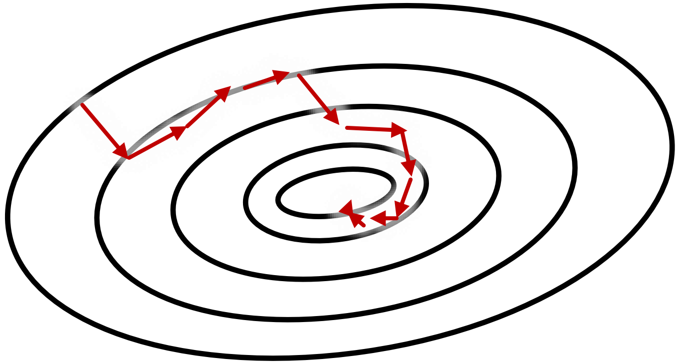
pooling

…

convolution 3

$f(x)$ fully connected

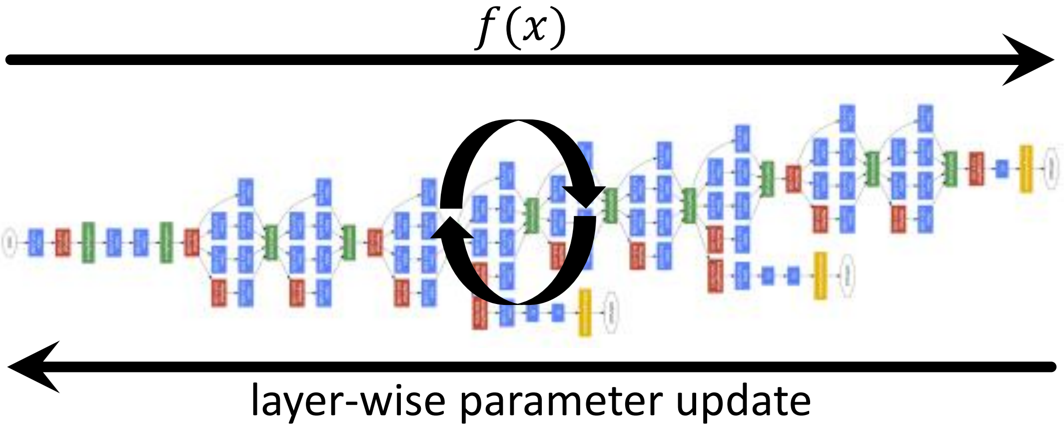
- Layer storage = $|w_l| + |f_l(o_{l-1})| + |\nabla w_l| + |\nabla o_l|$



Learning Rate	$w^{(t+1)} = w^{(t)} - \eta \cdot \nabla \ell(w^{(t)}, z) \quad = w^{(t)} - \eta \cdot \nabla w^{(t)}$
Adaptive Learning Rate	$w^{(t+1)} = w^{(t)} - \eta_t \cdot \nabla w^{(t)}$
Momentum [Qian 1999]	$w^{(t+1)} = w^{(t)} + \mu \cdot (w^{(t)} - w^{(t-1)}) - \eta \cdot \nabla w^{(t)}$
Nesterov Momentum [Nesterov 1983]	$w^{(t+1)} = w^{(t)} + v_t; \quad v_{t+1} = \mu \cdot v_t - \eta \cdot \nabla \ell(w^{(t)} - \mu \cdot v_t, z)$
AdaGrad [Duchi et al. 2011]	$w_i^{(t+1)} = w_i^{(t)} - \frac{\eta \cdot \nabla w_i^{(t)}}{\sqrt{A_{i,t} + \varepsilon}}; \quad A_{i,t} = \sum_{\tau=0}^{t} \left(\nabla w_i^{(t)}\right)^2$
RMSProp [Hinton 2012]	$w_i^{(t+1)} = w_i^{(t)} - \frac{\eta \cdot \nabla w_i^{(t)}}{\sqrt{A'_{i,t} + \varepsilon}}; \quad A'_{i,t} = \beta \cdot A'_{t-1} + (1-\beta)\left(\nabla w_i^{(t)}\right)^2$
Adam [Kingma and Ba 2015]	$w_i^{(t+1)} = w_i^{(t)} - \frac{\eta \cdot M_{i,t}^{(1)}}{\sqrt{M_{i,t}^{(2)} + \varepsilon}}; \quad M_{i,t}^{(m)} = \frac{\beta_m \cdot M_{i,t-1}^{(m)} + (1-\beta_m)\left(\nabla w_i^{(t)}\right)^m}{1 - \beta_m^t}$

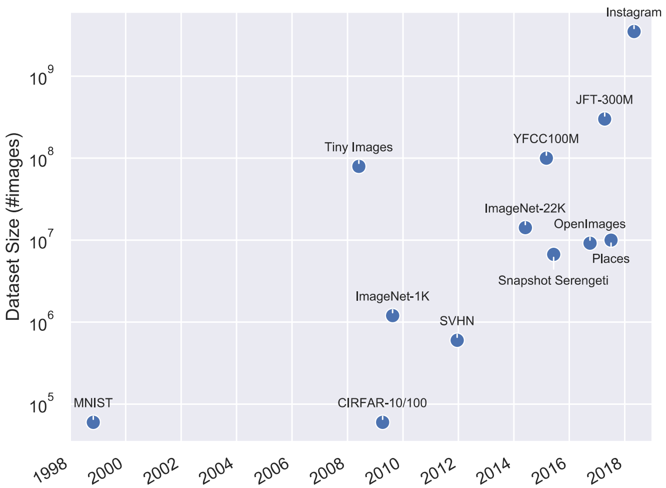
T. Ben-Nun, T. Hoefler: Demystifying Parallel and Distributed Deep Learning: An In-Depth Concurrency Analysis, CSUR 2019

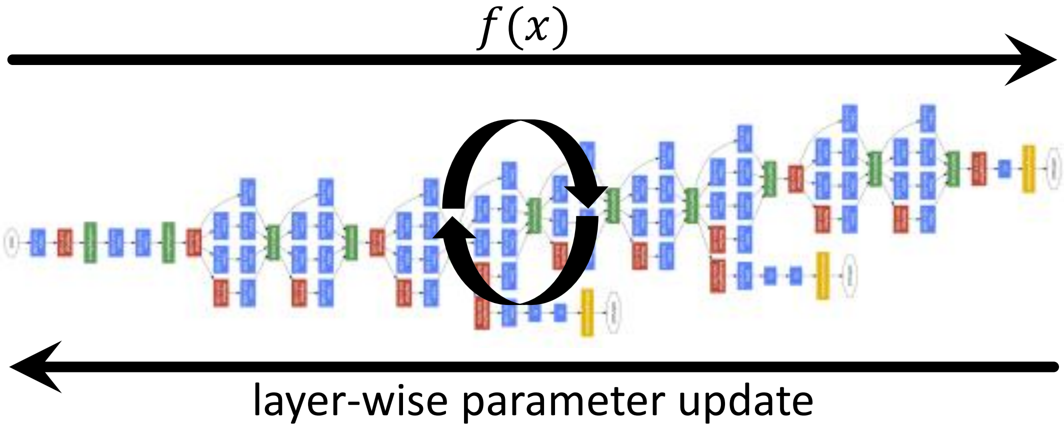
The scale of deep learning



Cat	0.54	Cat	1.00
Dog	0.28	Dog	0.00
Airplane	0.07	Airplane	0.00
Horse	0.33	Horse	0.00
Banana	0.02	Banana	0.00
Truck	0.02	Truck	0.00

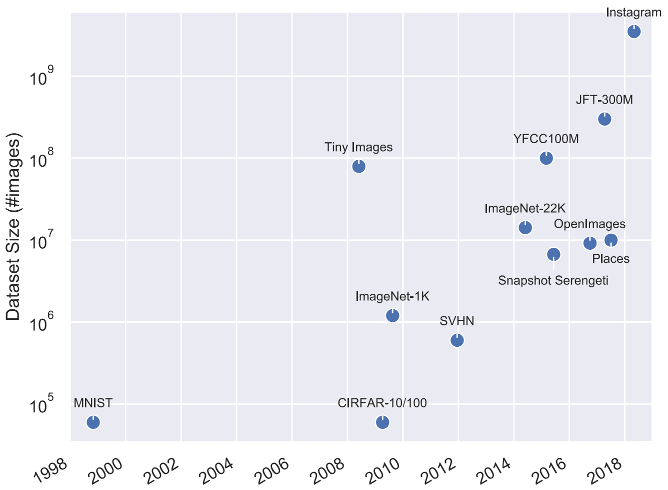
The scale of deep learning

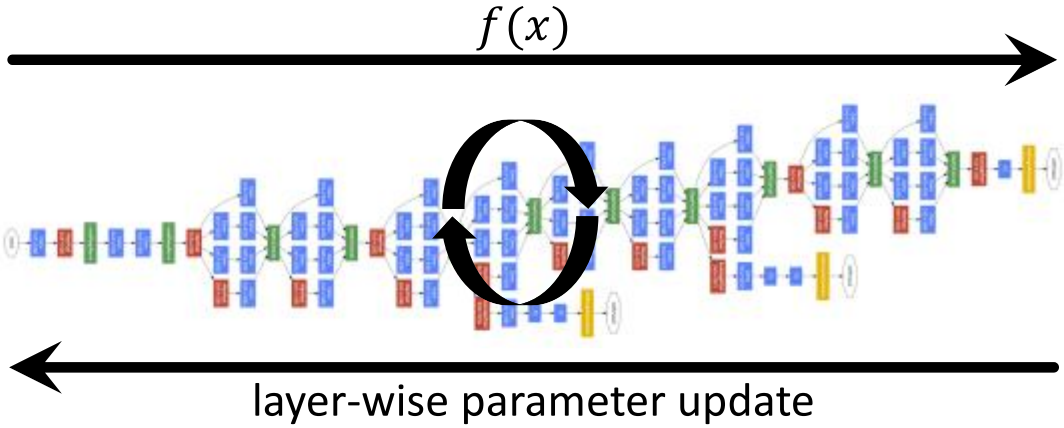




Cat	0.54	Cat	1.00
Dog	0.28	Dog	0.00
Airplane	0.07	Airplane	0.00
Horse	0.33	Horse	0.00
Banana	0.02	Banana	0.00
Truck	0.02	Truck	0.00

The scale of deep learning

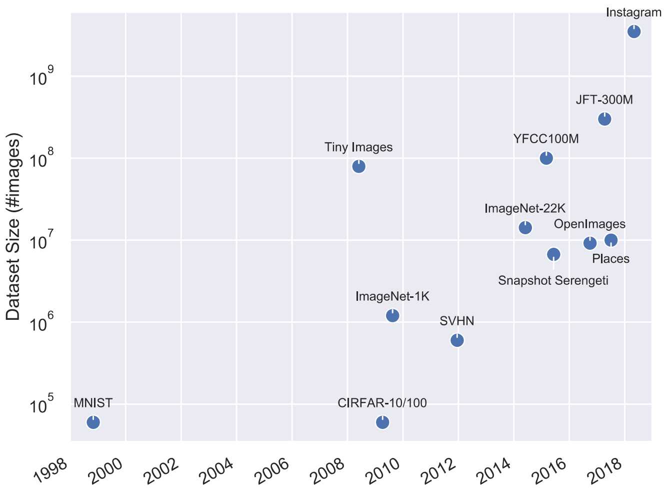


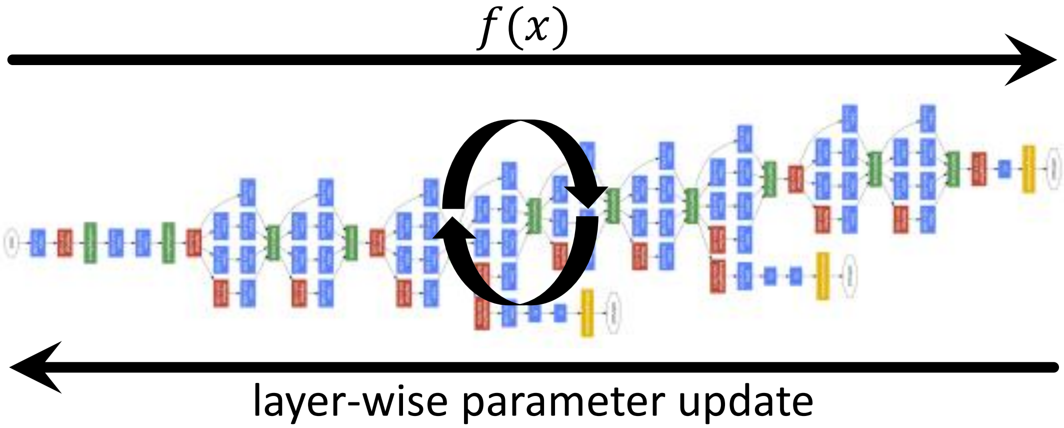


Cat	0.54	1.00
Dog	0.28	0.00
Airplane	0.07	0.00
Horse	0.33	0.00
Banana	0.02	0.00
Truck	0.02	0.00

- ImageNet 1k: 150 GB
- ImageNet 22k: ~2 TB
- Industry: Much larger

The scale of deep learning





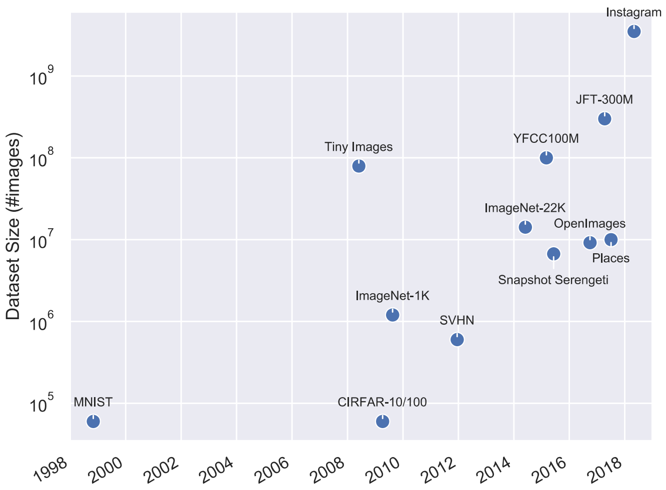
Cat	0.54
Dog	0.28
Airplane	0.07
Horse	0.33
Banana	0.02
Truck	0.02

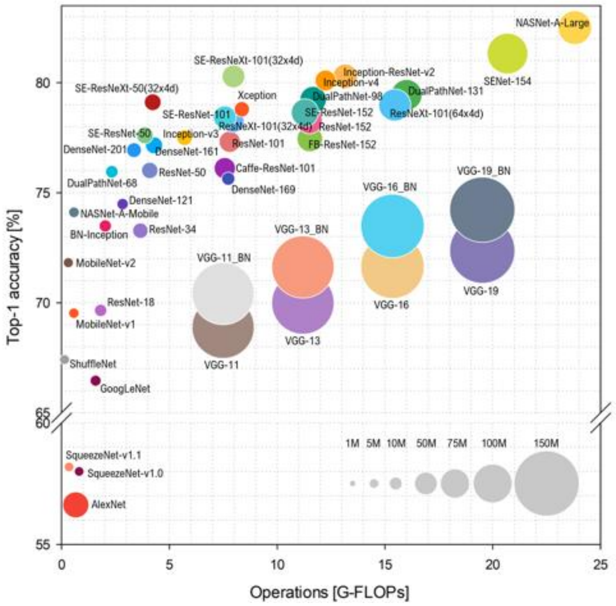
Cat	1.00
Dog	0.00
Airplane	0.00
Horse	0.00
Banana	0.00
Truck	0.00

- ImageNet 1k: 150 GB
- ImageNet 22k: ~2 TB
- Industry: Much larger

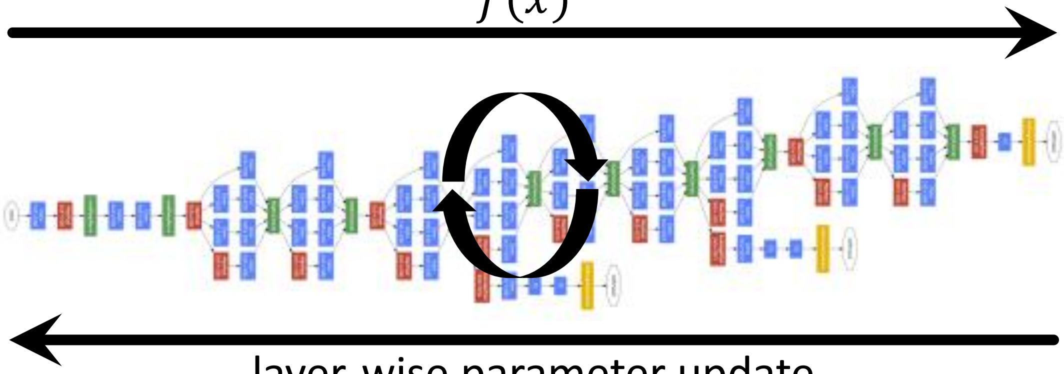
- >100 layers deep
- ~25M – >10B parameters
- 0.1 – 40 GiB storage

The scale of deep learning





$f(x)$



layer-wise parameter update

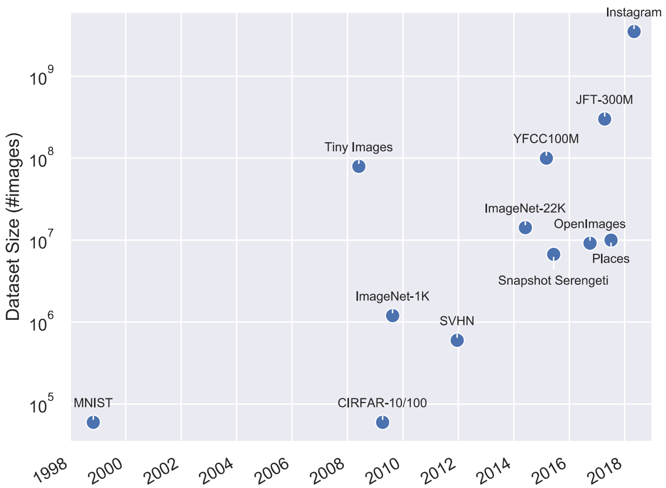
Cat	0.54
Dog	0.28
Airplane	0.07
Horse	0.33
Banana	0.02
Truck	0.02

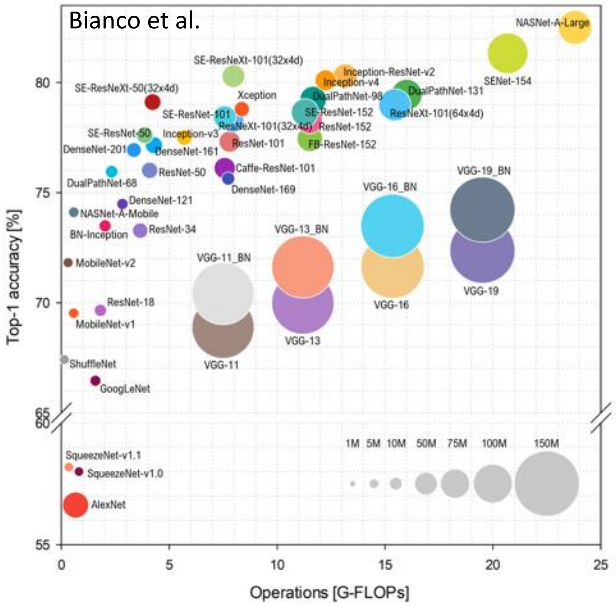
Cat	1.00
Dog	0.00
Airplane	0.00
Horse	0.00
Banana	0.00
Truck	0.00

- ImageNet 1k: 150 GB
- ImageNet 22k: ~2 TB
- Industry: Much larger

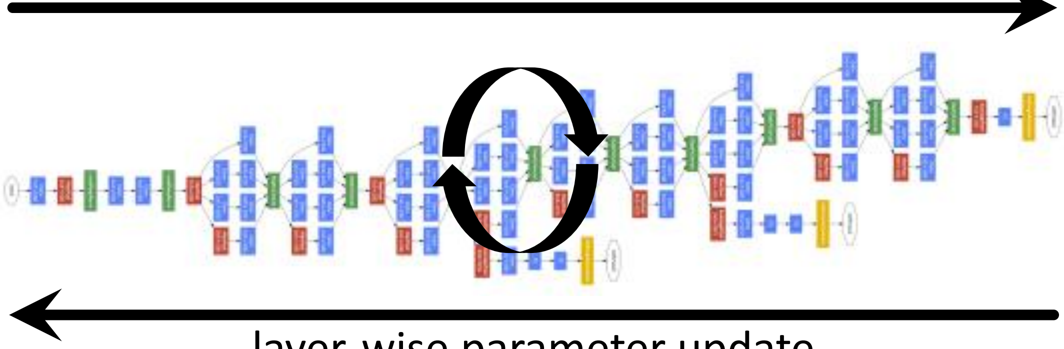
- >100 layers deep
- ~25M – >10B parameters
- 0.1 – 40 GiB storage

The scale of deep learning





$f(x)$



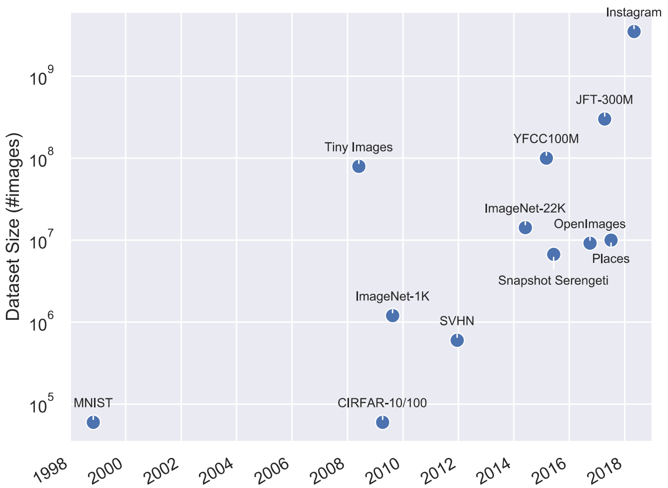
layer-wise parameter update

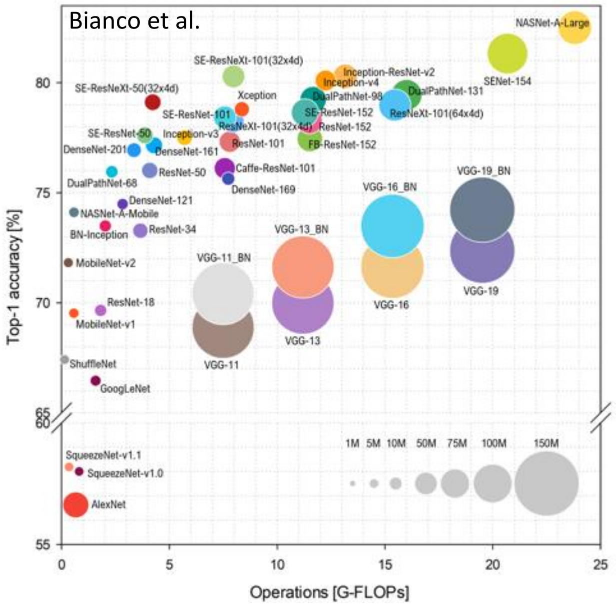
Cat	0.54	Cat	1.00
Dog	0.28	Dog	0.00
Airplane	0.07	Airplane	0.00
Horse	0.33	Horse	0.00
Banana	0.02	Banana	0.00
Truck	0.02	Truck	0.00

- ImageNet 1k: 150 GB
- ImageNet 22k: ~2 TB
- Industry: Much larger

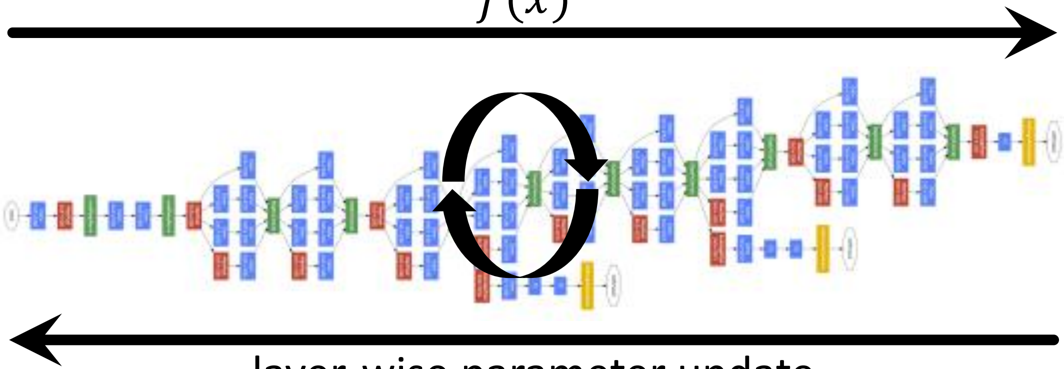
- >100 layers deep
- ~25M – >10B parameters
- 0.1 – 40 GiB storage

The scale of deep learning





$f(x)$



layer-wise parameter update

Cat	0.54
Dog	0.28
Airplane	0.07
Horse	0.33
Banana	0.02
Truck	0.02

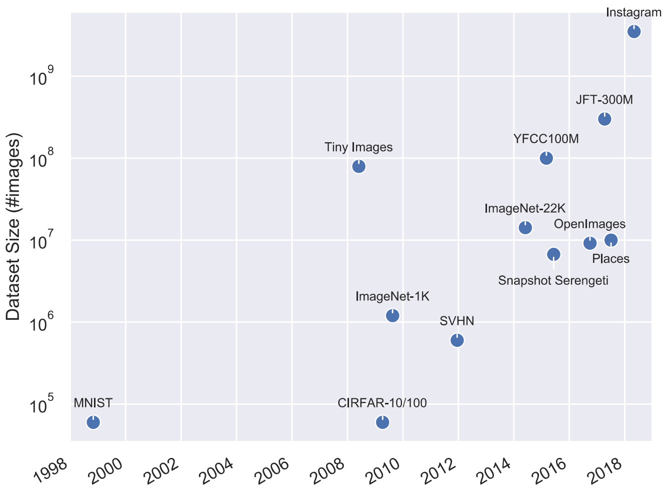
Cat	1.00
Dog	0.00
Airplane	0.00
Horse	0.00
Banana	0.00
Truck	0.00

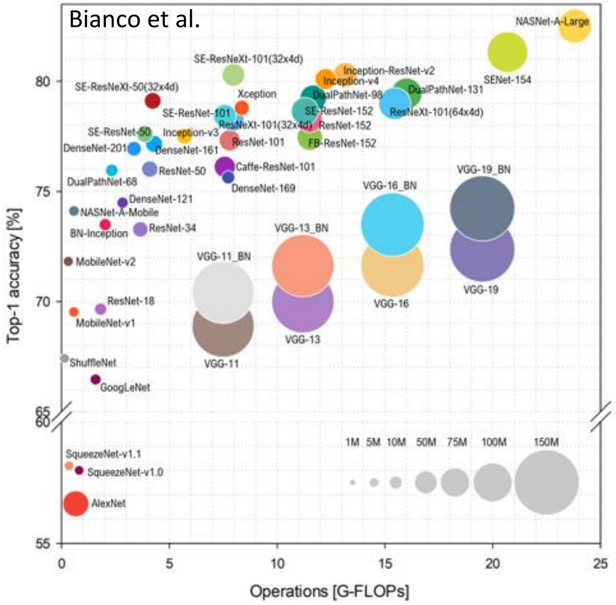
- ImageNet 1k: 150 GB
- ImageNet 22k: ~2 TB
- Industry: Much larger

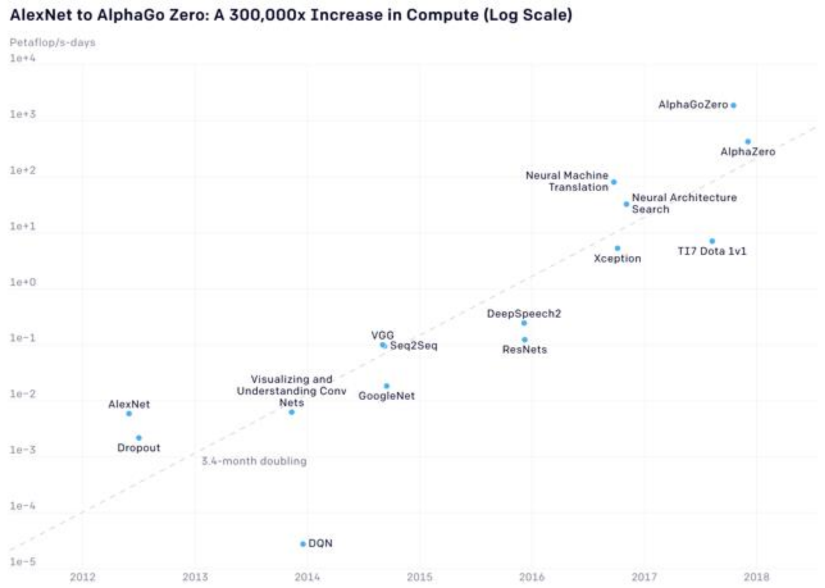
- >100 layers deep
- ~25M – >10B parameters
- 0.1 – 40 GiB storage

- 10-22k labels
- Growing
- Weeks to train

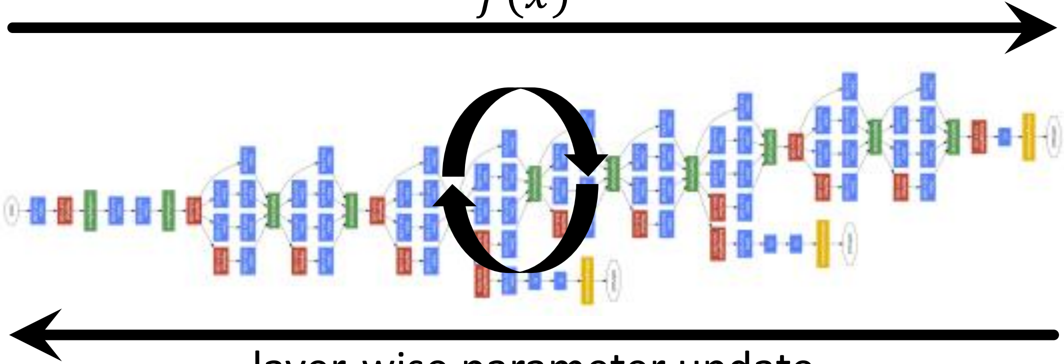
The scale of deep learning







$f(x)$



layer-wise parameter update

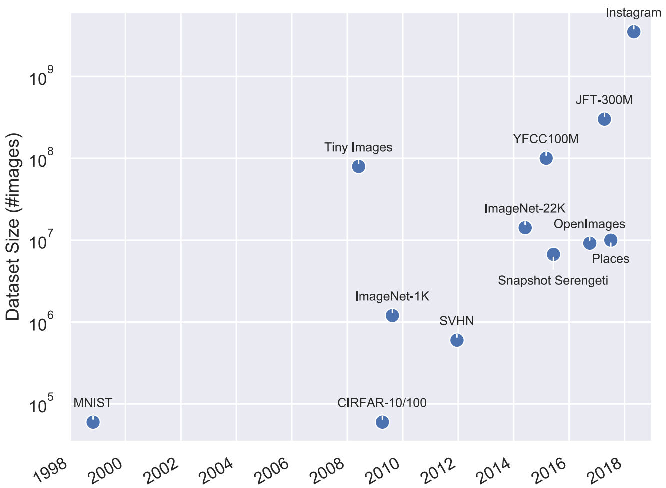
Cat	0.54	Cat	1.00
Dog	0.28	Dog	0.00
Airplane	0.07	Airplane	0.00
Horse	0.33	Horse	0.00
Banana	0.02	Banana	0.00
Truck	0.02	Truck	0.00

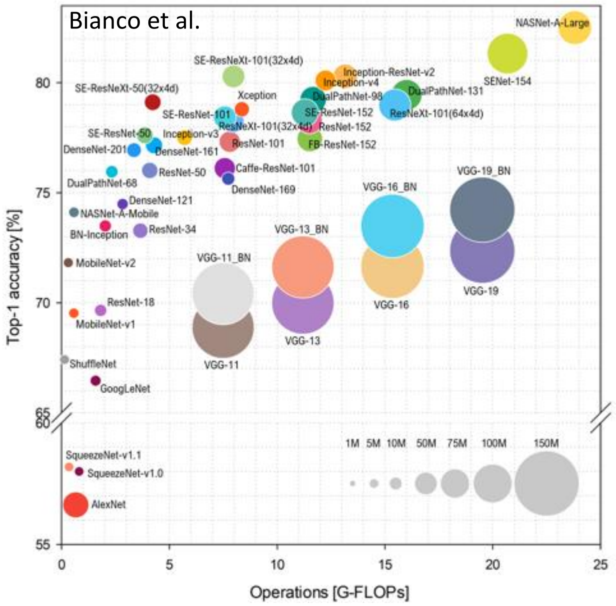
- ImageNet 1k: 150 GB
- ImageNet 22k: ~2 TB
- Industry: Much larger

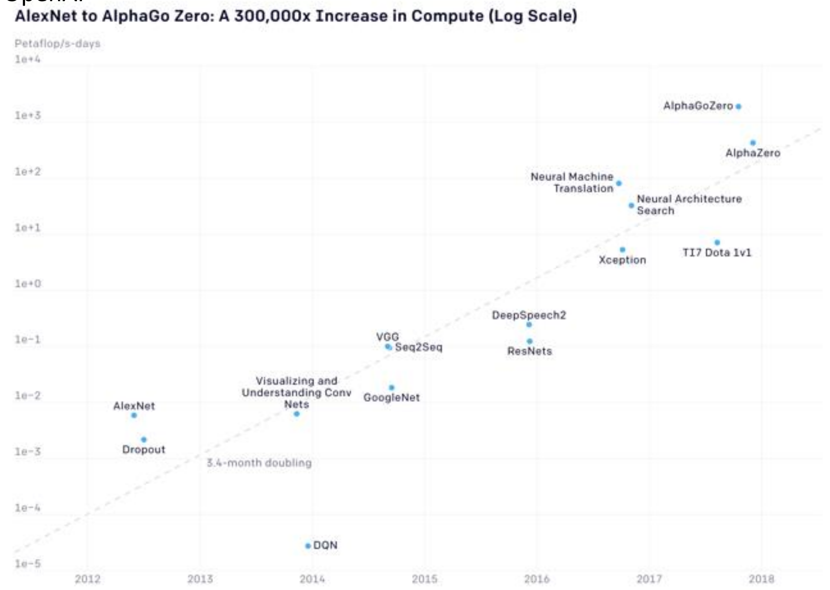
- >100 layers deep
- ~25M – >10B parameters
- 0.1 – 40 GiB storage

- 10-22k labels
- Growing
- Weeks to train

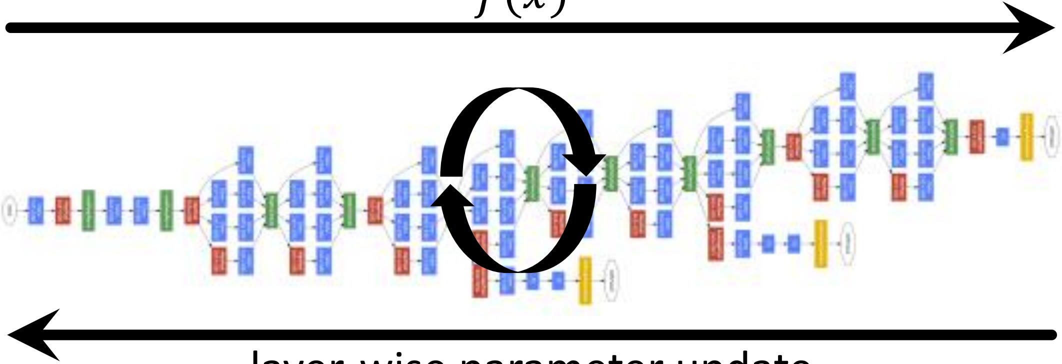
The scale of deep learning







$f(x)$



layer-wise parameter update

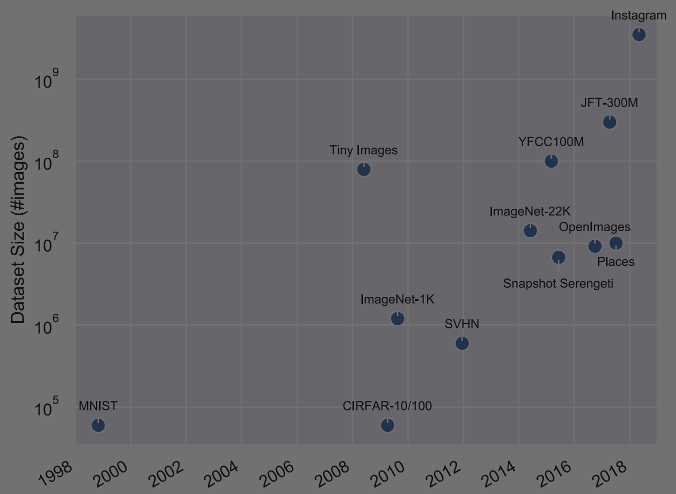
Cat	0.54	Cat	1.00
Dog	0.28	Dog	0.00
Airplane	0.07	Airplane	0.00
Horse	0.33	Horse	0.00
Banana	0.02	Banana	0.00
Truck	0.02	Truck	0.00

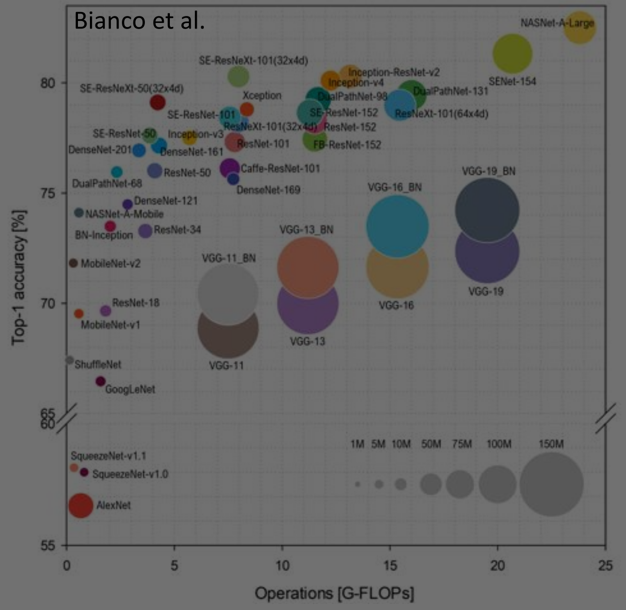
- ImageNet 1k: 150 GB
- ImageNet 22k: ~2 TB
- Industry: Much larger

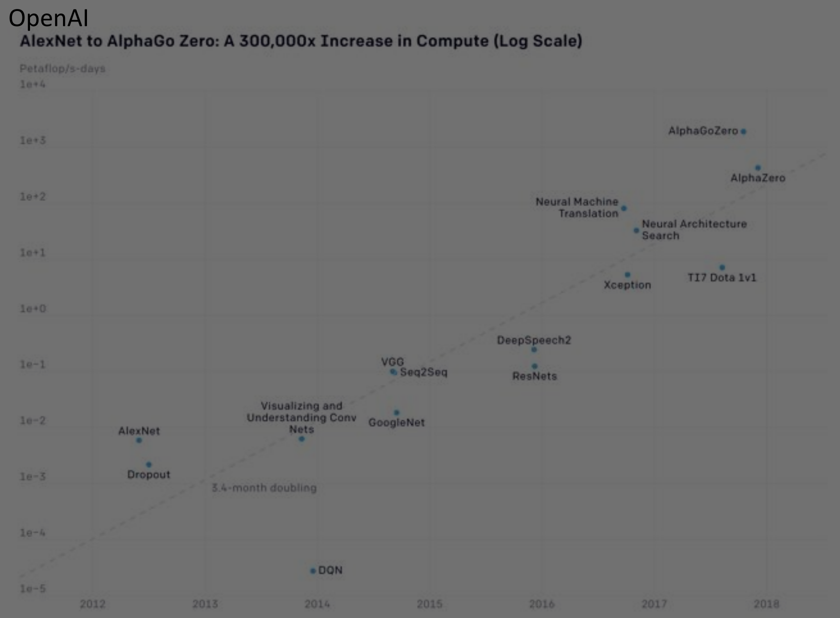
- >100 layers deep
- ~25M – >10B parameters
- 0.1 – 40 GiB storage

- 10-22k labels
- Growing
- Weeks to train

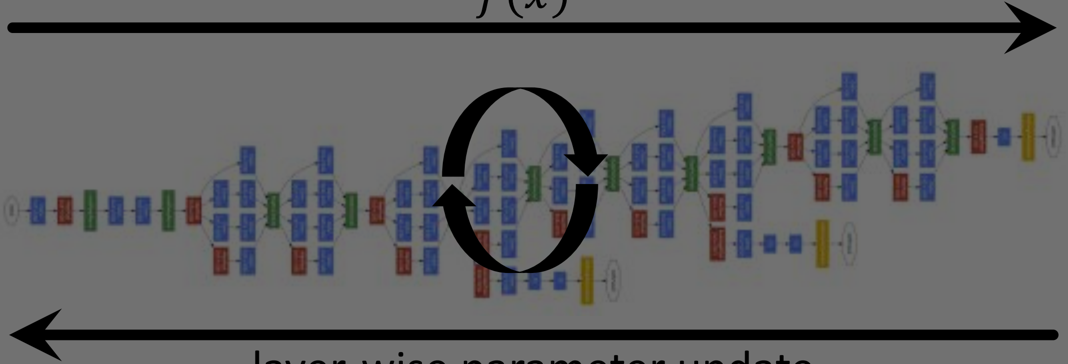
The scale of deep learning







$f(x)$



layer-wise parameter update

Cat	0.54
Dog	0.28
Airplane	0.07
Horse	0.33
Banana	0.02
Truck	0.02

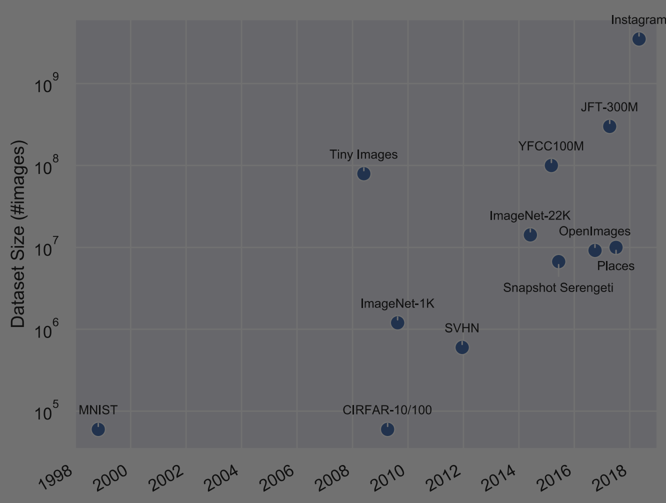
Cat	1.00
Dog	0.00
Airplane	0.00
Horse	0.00
Banana	0.00
Truck	0.00

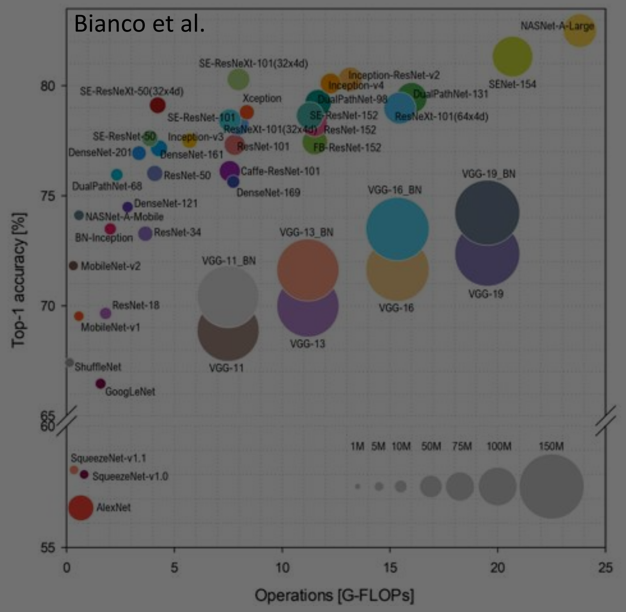
- ImageNet 1k: 150 GB
- ImageNet 22k: ~2 TB
- Industry: Much larger

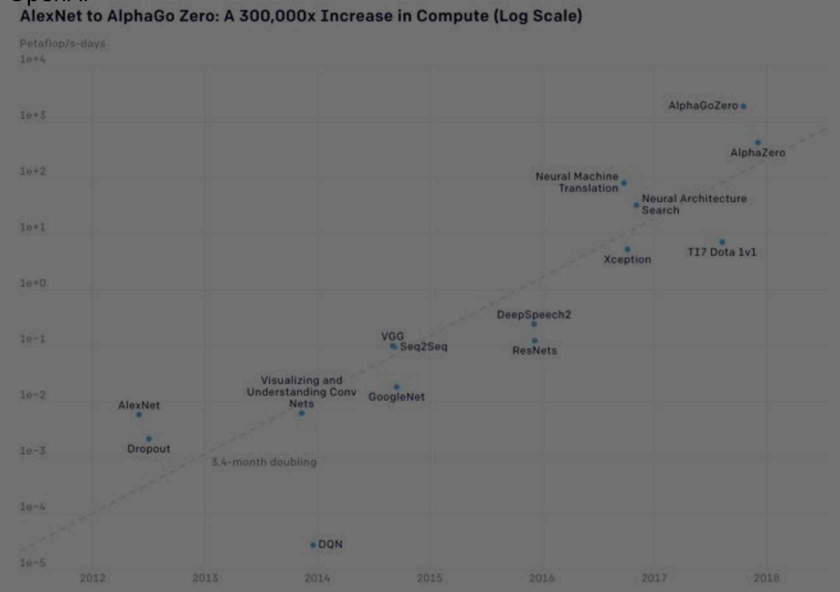
- >100 layers deep
- ~25M – >10B parameters
- 0.1 – 40 GiB storage

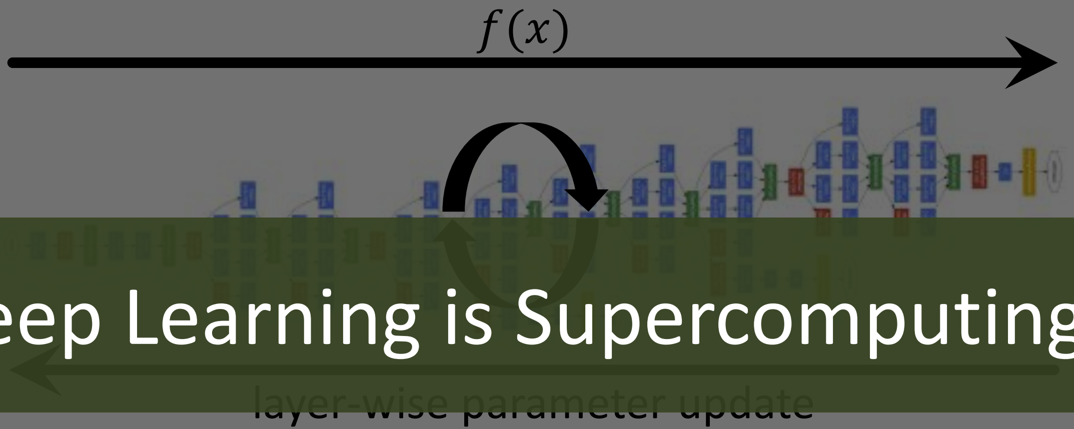
- 10-22k labels
- Growing
- Weeks to train

The scale of deep learning









Cat	0.54	Cat	1.00
Dog	0.28	Dog	0.00
Airplane	0.07	Airplane	0.00
Horse	0.33	Horse	0.00
Banana	0.02	Banana	0.00
Truck	0.02	Truck	0.00

Deep Learning is Supercomputing!

- ImageNet 1k: 150 GB
- ImageNet 22k: ~2 TB
- Industry: Much larger

- >100 layers deep
- ~25M – >10B parameters
- 0.1 – 40 GiB storage

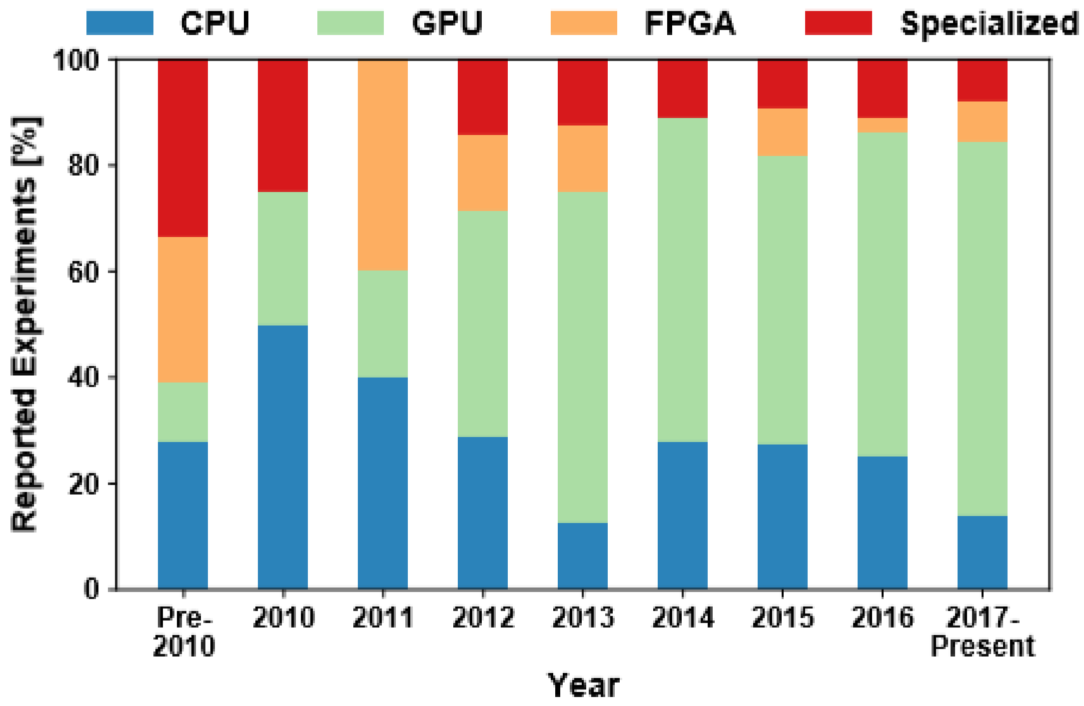
- 10-22k labels
- Growing
- Weeks to train

Trends in deep learning: hardware and multi-node

T. Ben-Nun, T. Hoefler: Demystifying Parallel and Distributed Deep Learning: An In-Depth Concurrency Analysis, CSUR 2019

Trends in deep learning: hardware and multi-node

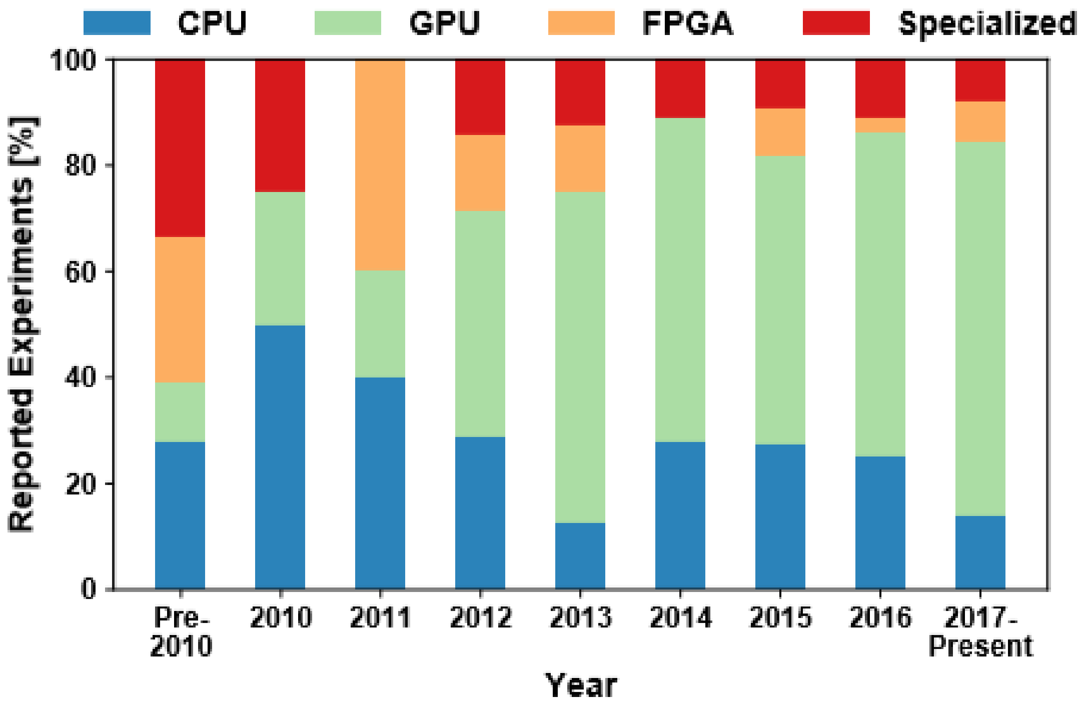
The field is moving fast – trying everything imaginable – survey results from 252 papers in the area of parallel deep learning



T. Ben-Nun, T. Hoefler: Demystifying Parallel and Distributed Deep Learning: An In-Depth Concurrency Analysis, CSUR 2019

Trends in deep learning: hardware and multi-node

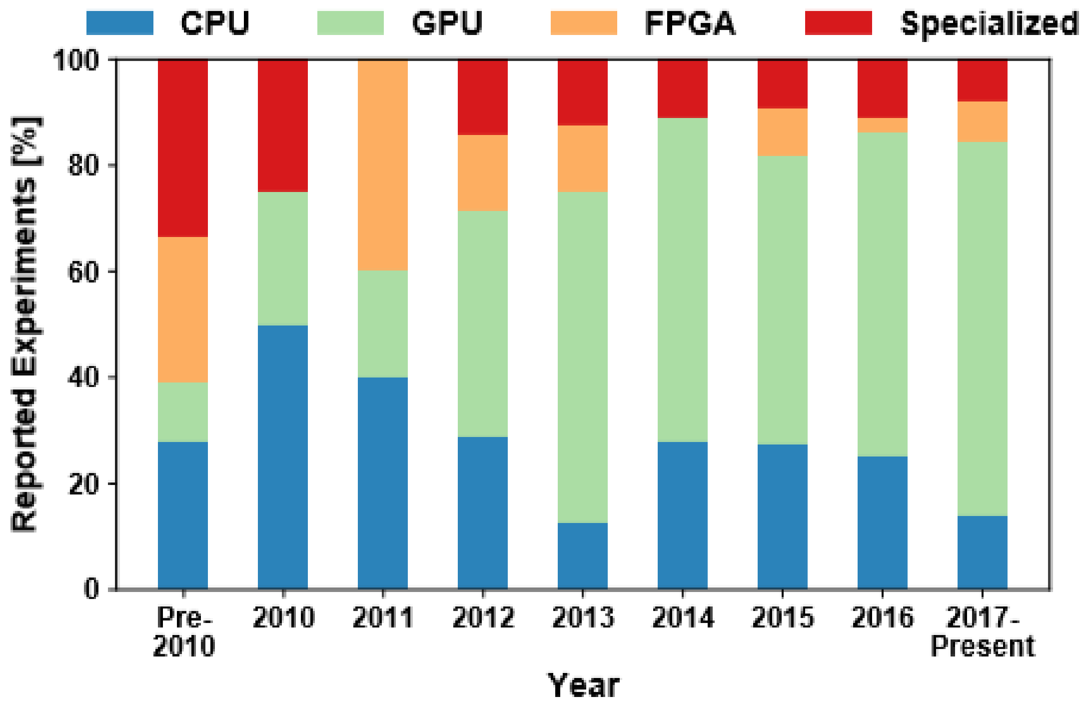
The field is moving fast – trying everything imaginable – survey results from 252 papers in the area of parallel deep learning



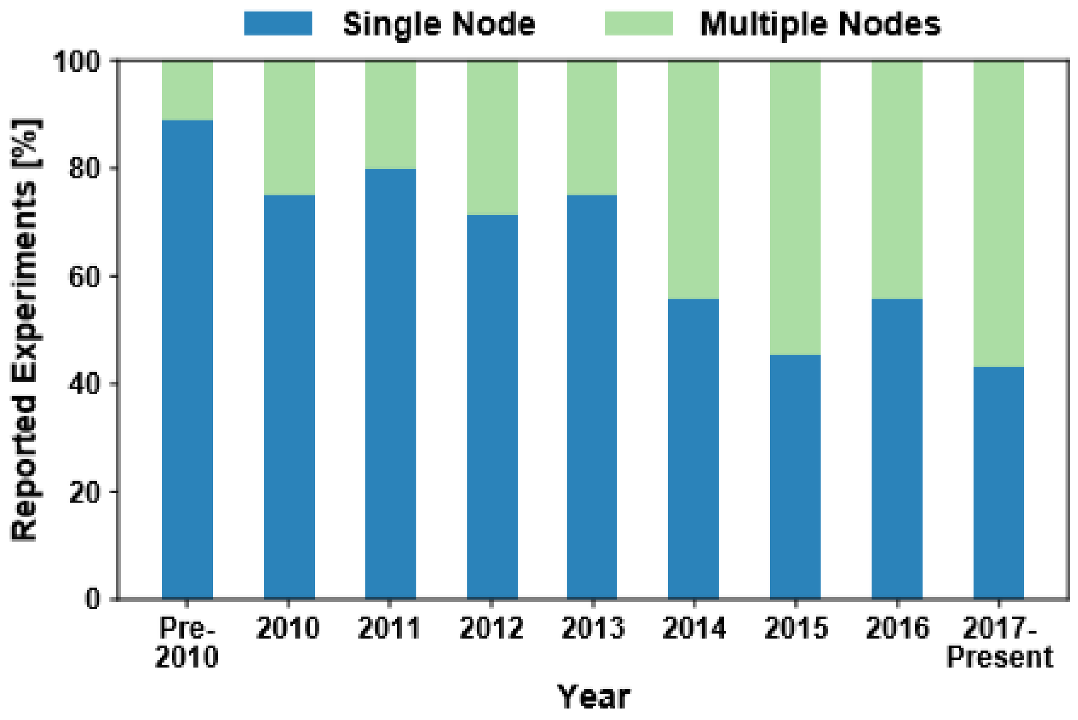
T. Ben-Nun, T. Hoefler: Demystifying Parallel and Distributed Deep Learning: An In-Depth Concurrency Analysis, CSUR 2019

Trends in deep learning: hardware and multi-node

The field is moving fast – trying everything imaginable – survey results from 252 papers in the area of parallel deep learning



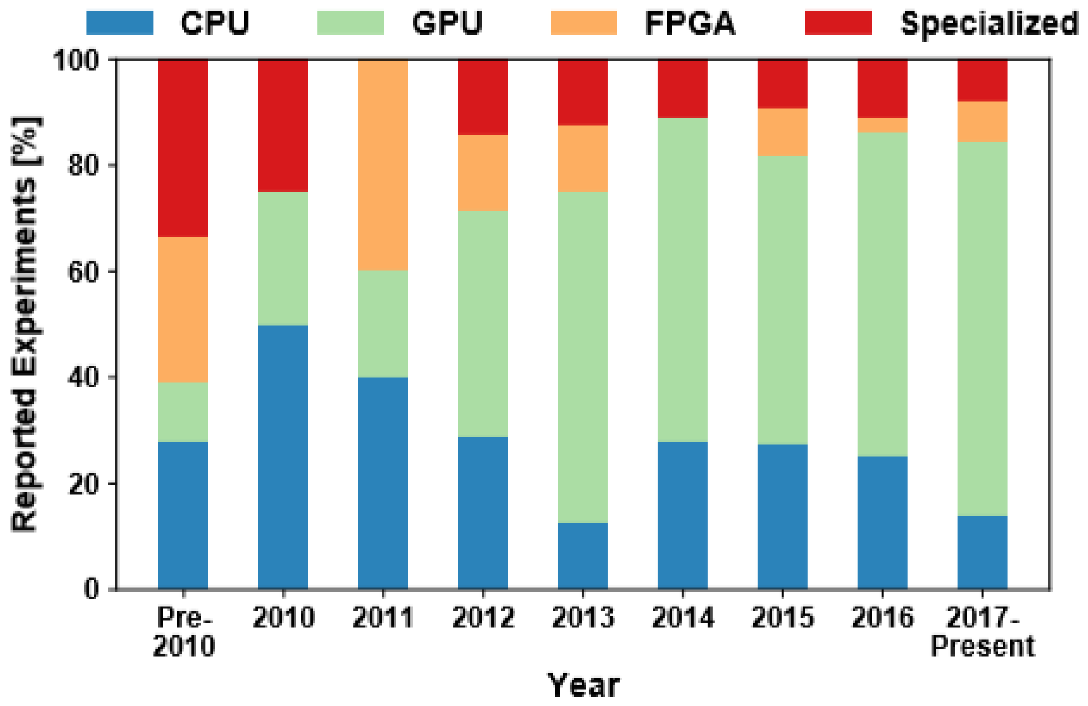
Hardware used



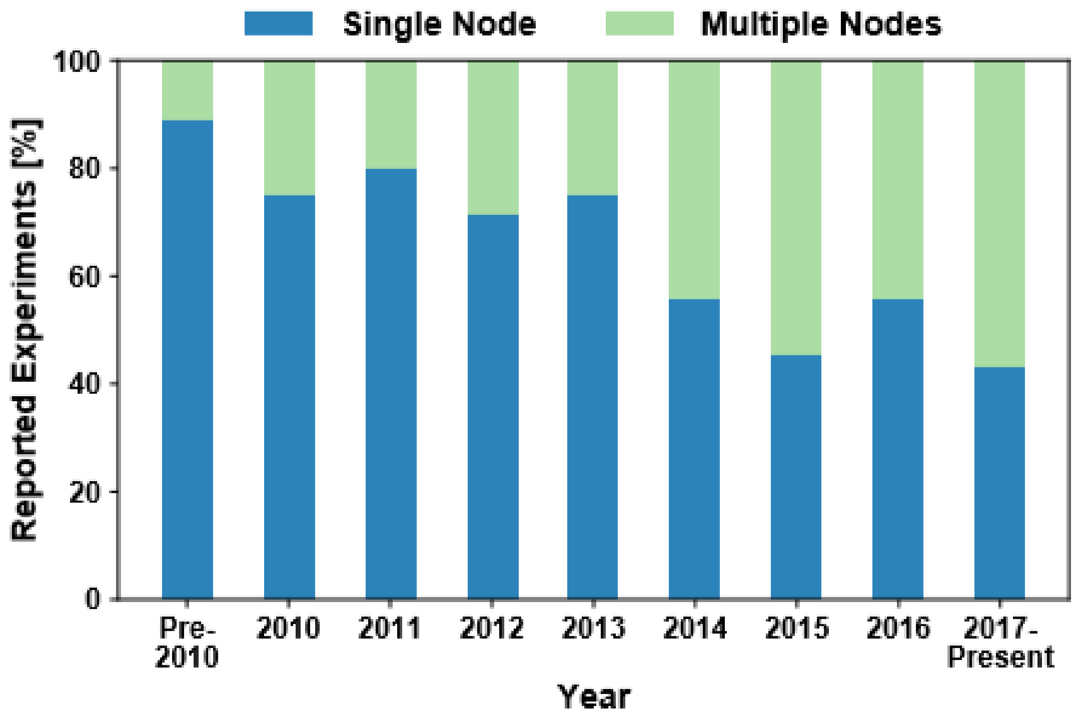
T. Ben-Nun, T. Hoefler: Demystifying Parallel and Distributed Deep Learning: An In-Depth Concurrency Analysis, CSUR 2019

Trends in deep learning: hardware and multi-node

The field is moving fast – trying everything imaginable – survey results from 252 papers in the area of parallel deep learning



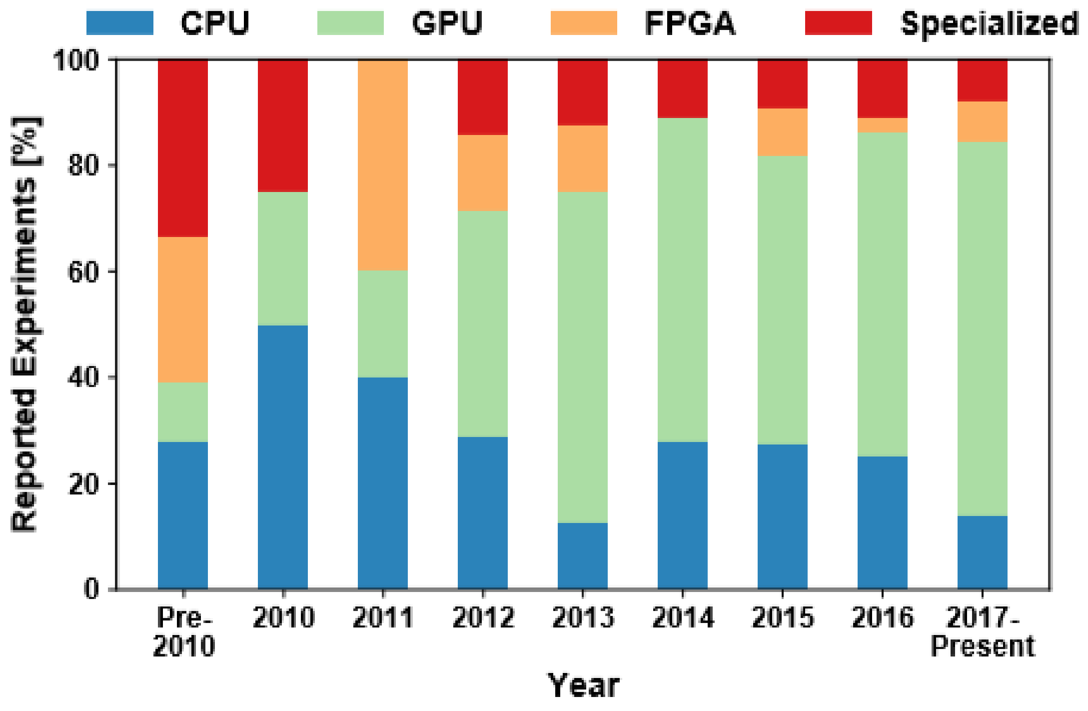
Hardware used



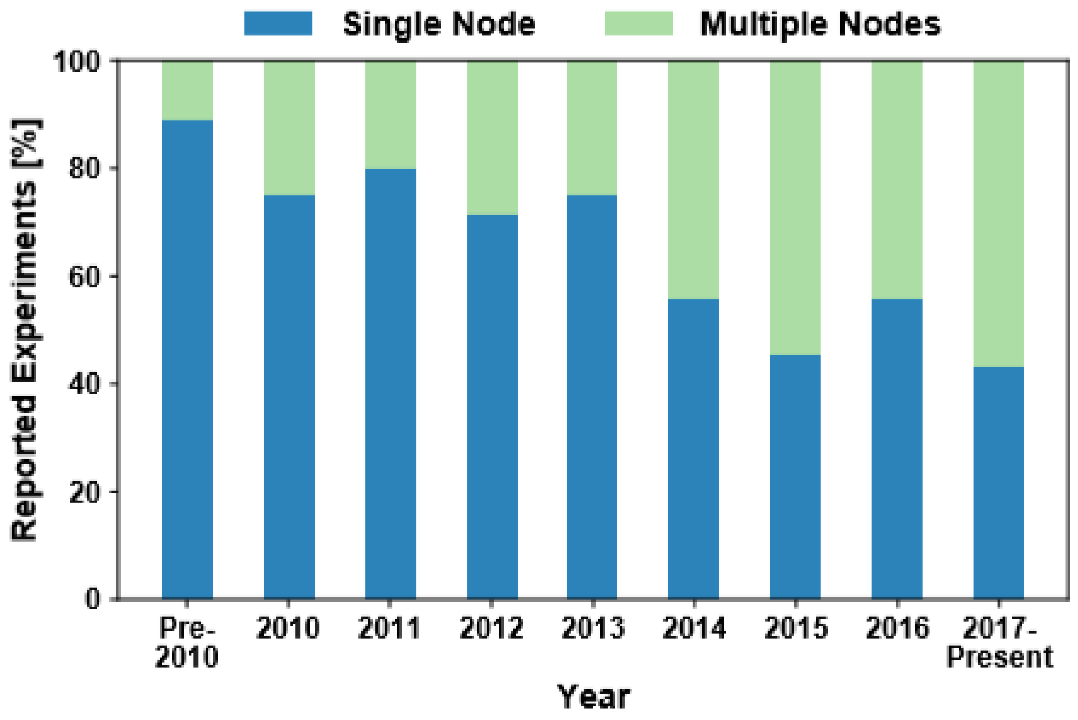
T. Ben-Nun, T. Hoefler: Demystifying Parallel and Distributed Deep Learning: An In-Depth Concurrency Analysis, CSUR 2019

Trends in deep learning: hardware and multi-node

The field is moving fast – trying everything imaginable – survey results from 252 papers in the area of parallel deep learning



Hardware used

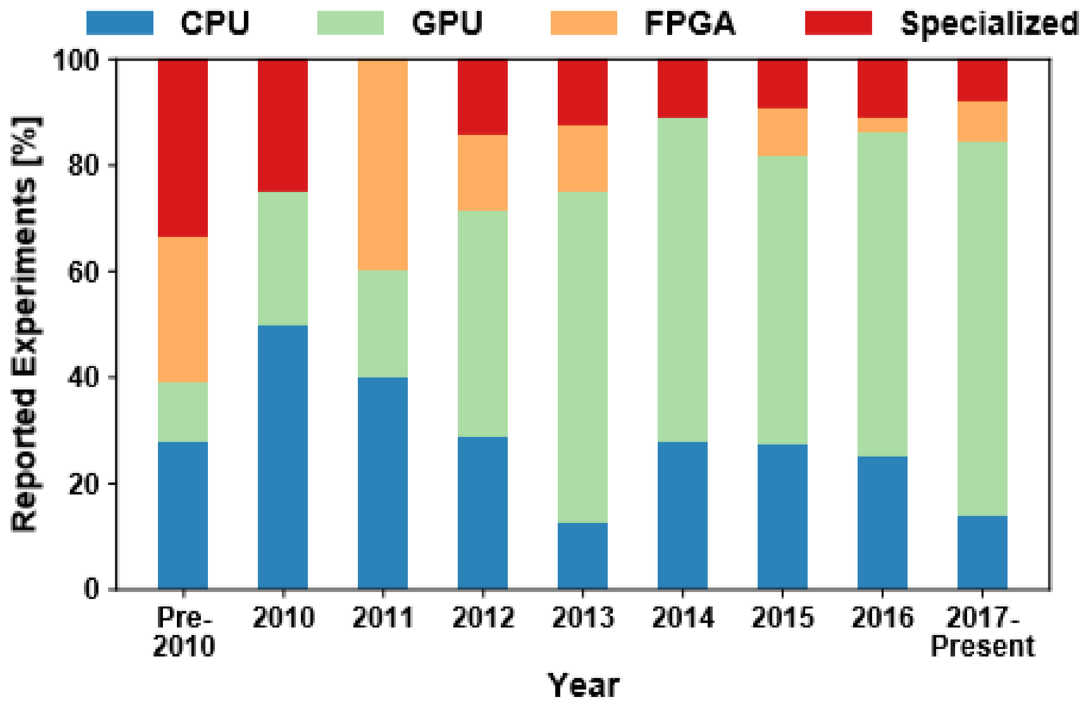


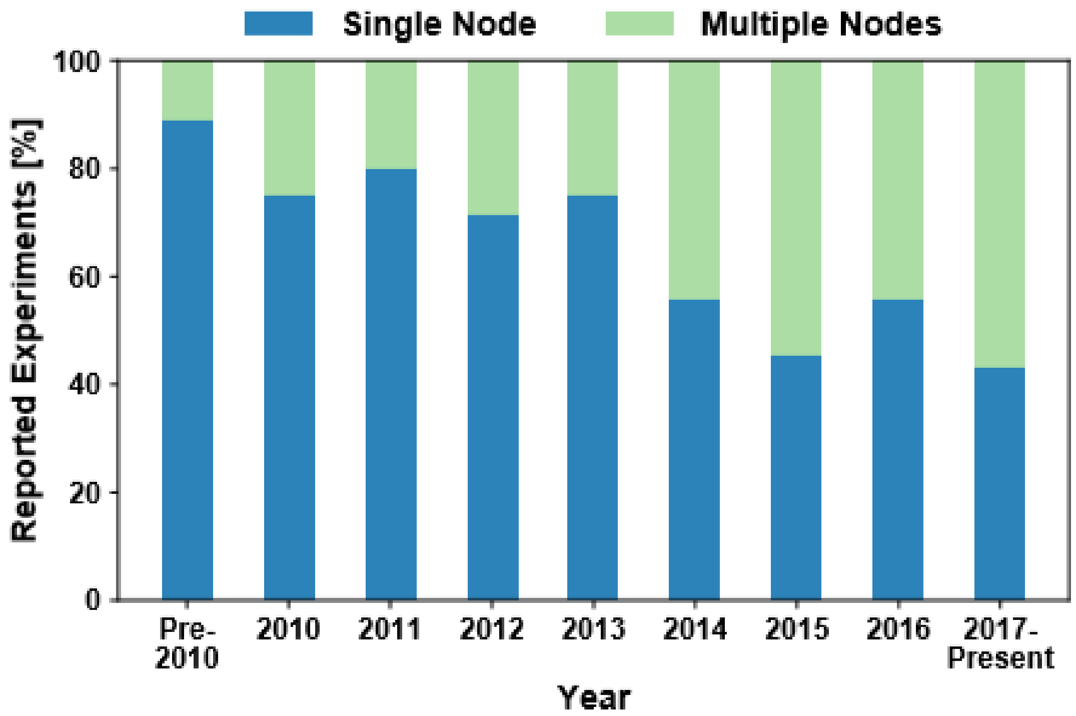
Shared vs. distributed memory

T. Ben-Nun, T. Hoefler: Demystifying Parallel and Distributed Deep Learning: An In-Depth Concurrency Analysis, CSUR 2019

Trends in deep learning: hardware and multi-node

The field is moving fast – trying everything imaginable – survey results from 252 papers in the area of parallel deep learning





Deep Learning is largely on distributed memory today!

T. Ben-Nun, T. Hoefler: Demystifying Parallel and Distributed Deep Learning: An In-Depth Concurrency Analysis, CSUR 2019

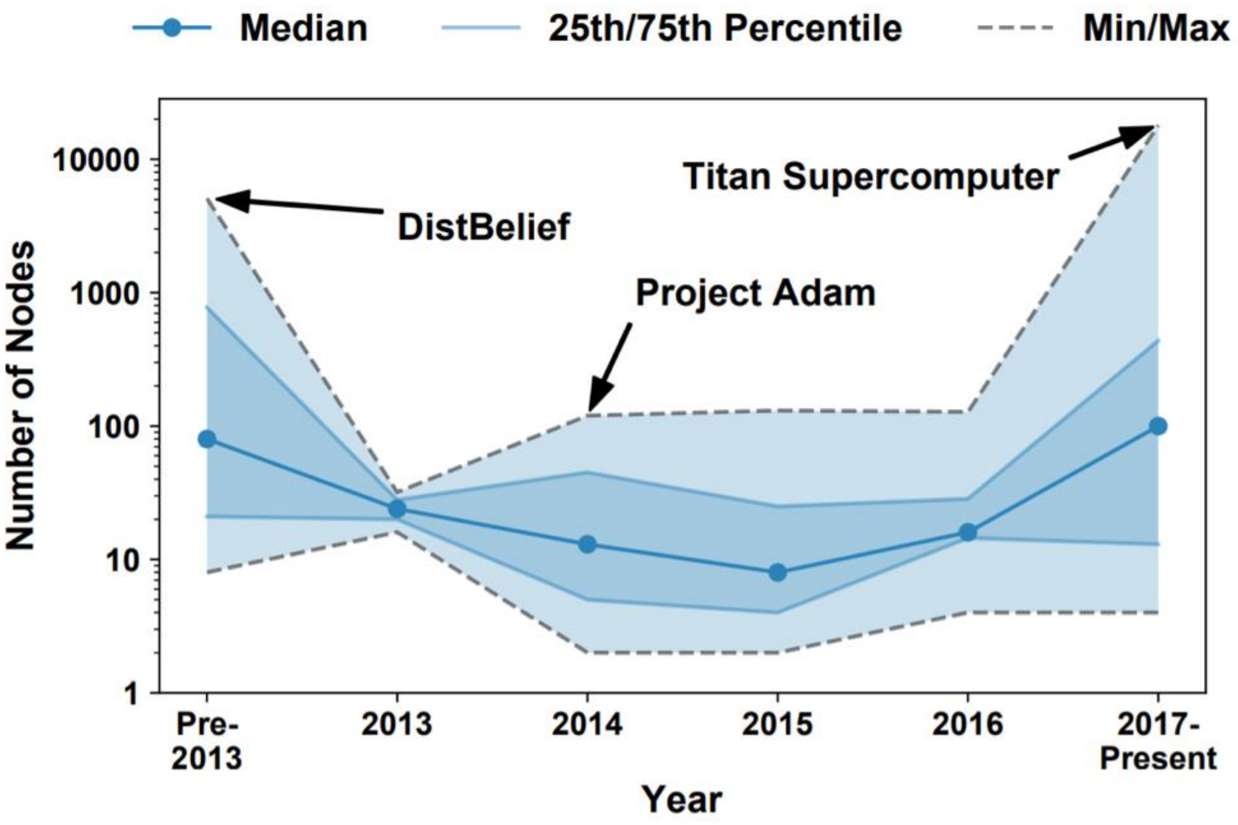
Trends in distributed deep learning: node count and communication

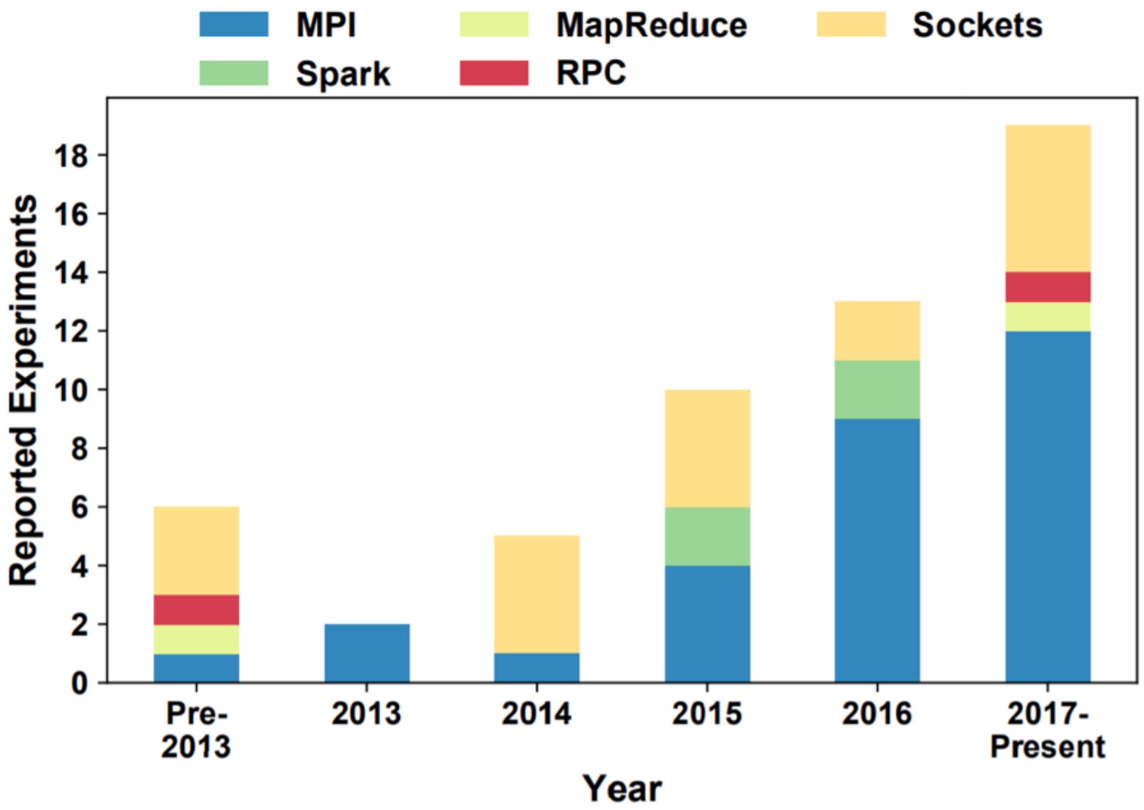
The field is moving fast – trying everything imaginable – survey results from 252 papers in the area of parallel deep learning

T. Ben-Nun, T. Hoefler: Demystifying Parallel and Distributed Deep Learning: An In-Depth Concurrency Analysis, CSUR 2019

Trends in distributed deep learning: node count and communication

The field is moving fast – trying everything imaginable – survey results from 252 papers in the area of parallel deep learning

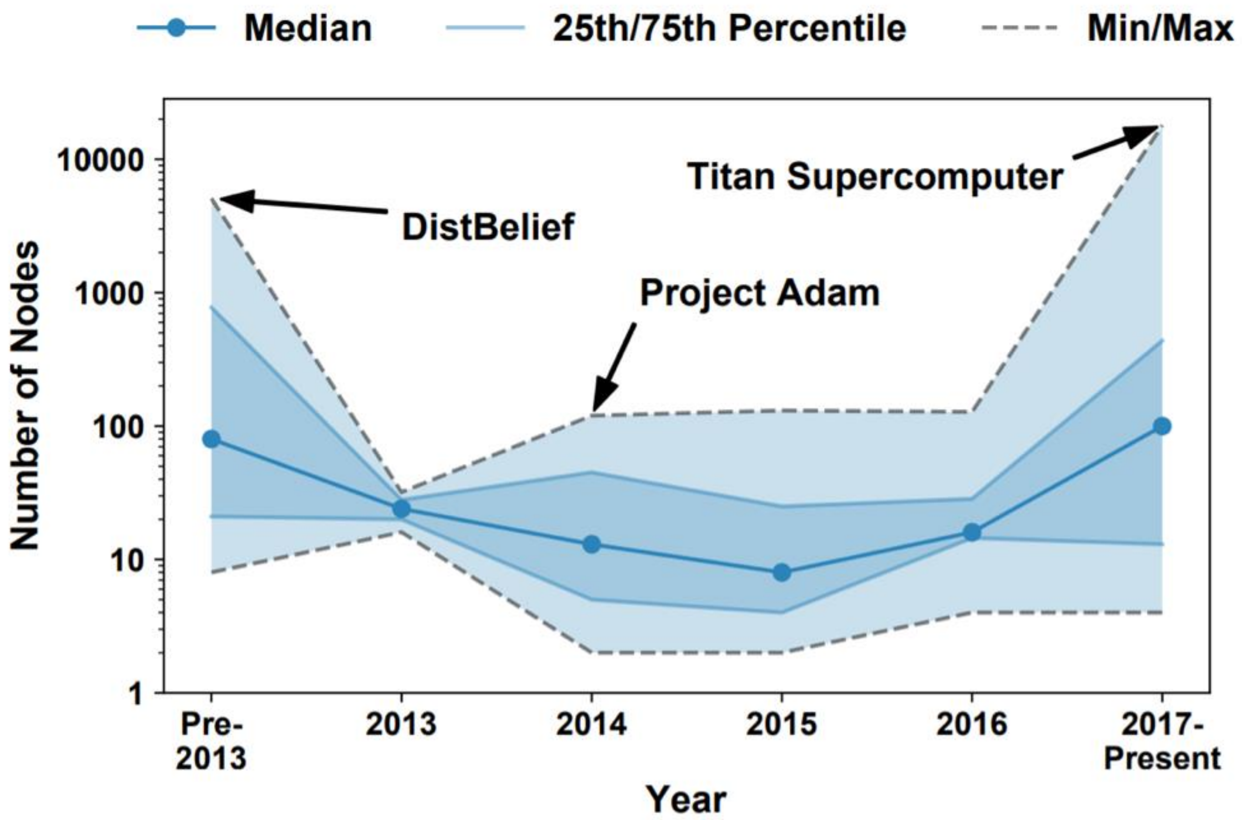


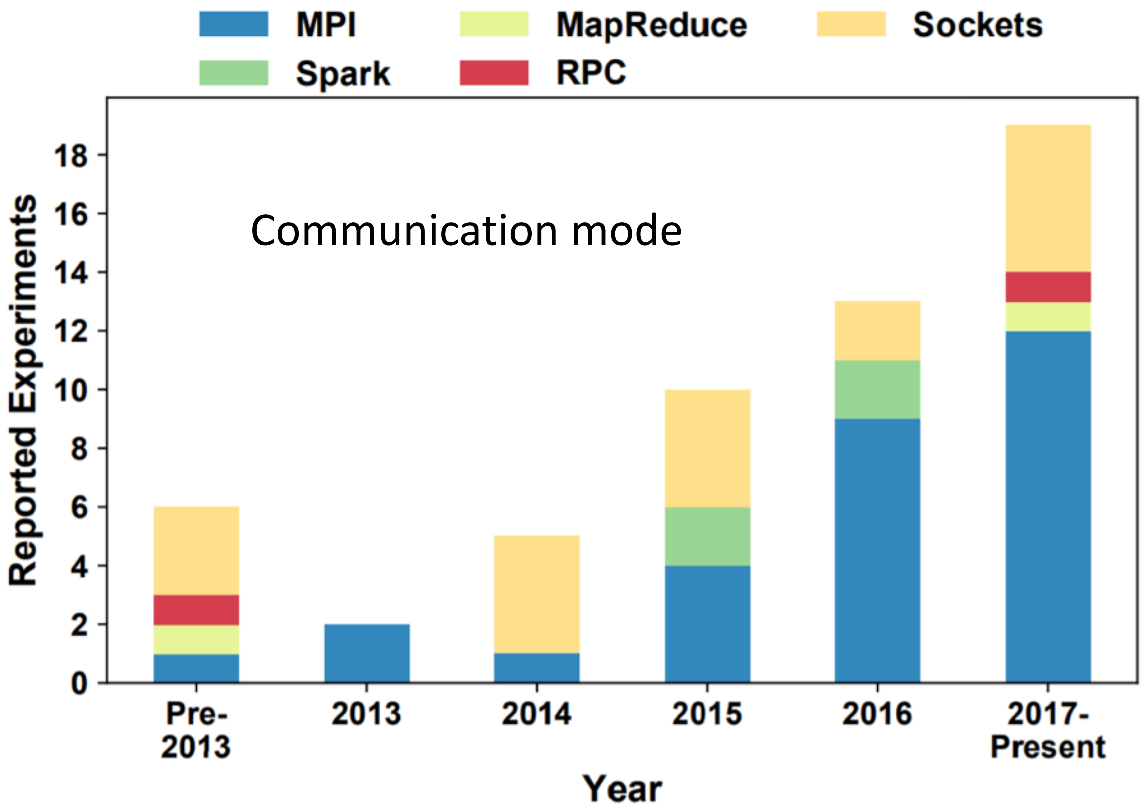


T. Ben-Nun, T. Hoefler: Demystifying Parallel and Distributed Deep Learning: An In-Depth Concurrency Analysis, CSUR 2019

Trends in distributed deep learning: node count and communication

The field is moving fast – trying everything imaginable – survey results from 252 papers in the area of parallel deep learning



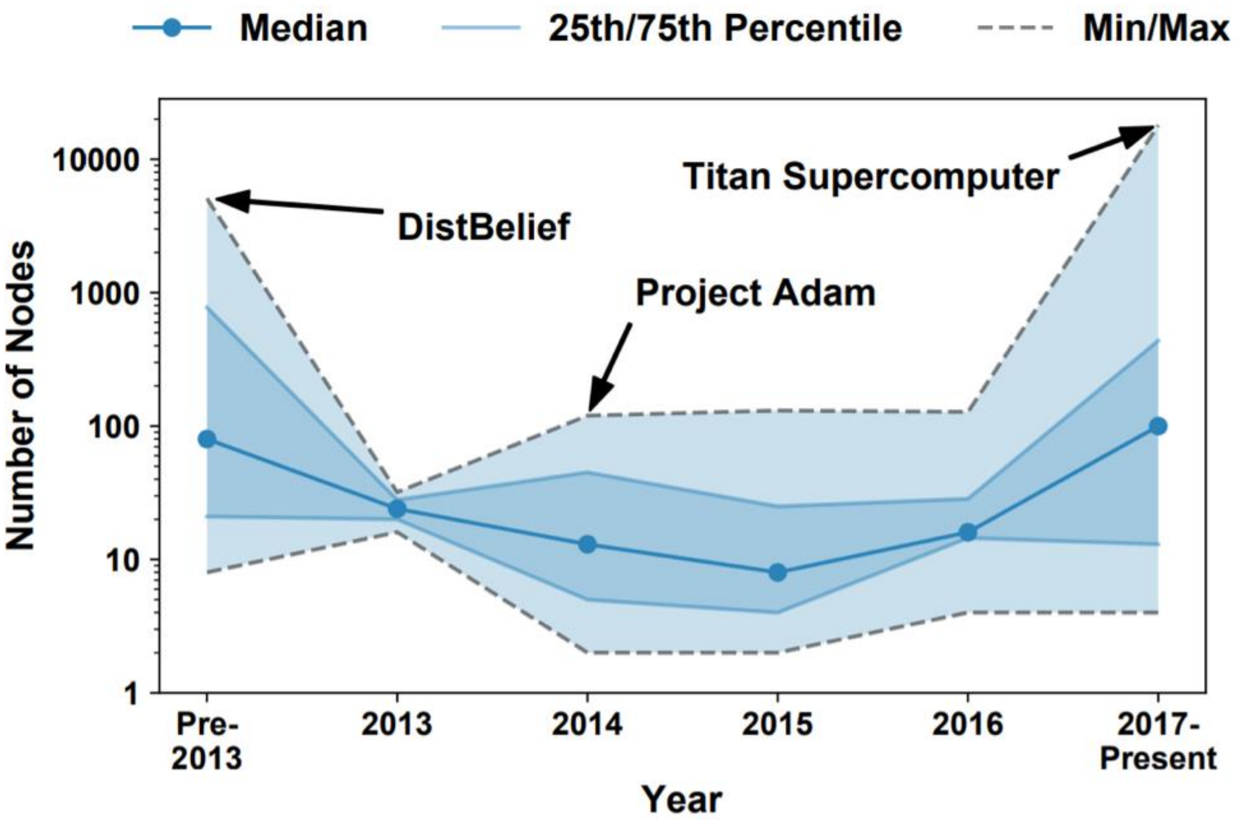


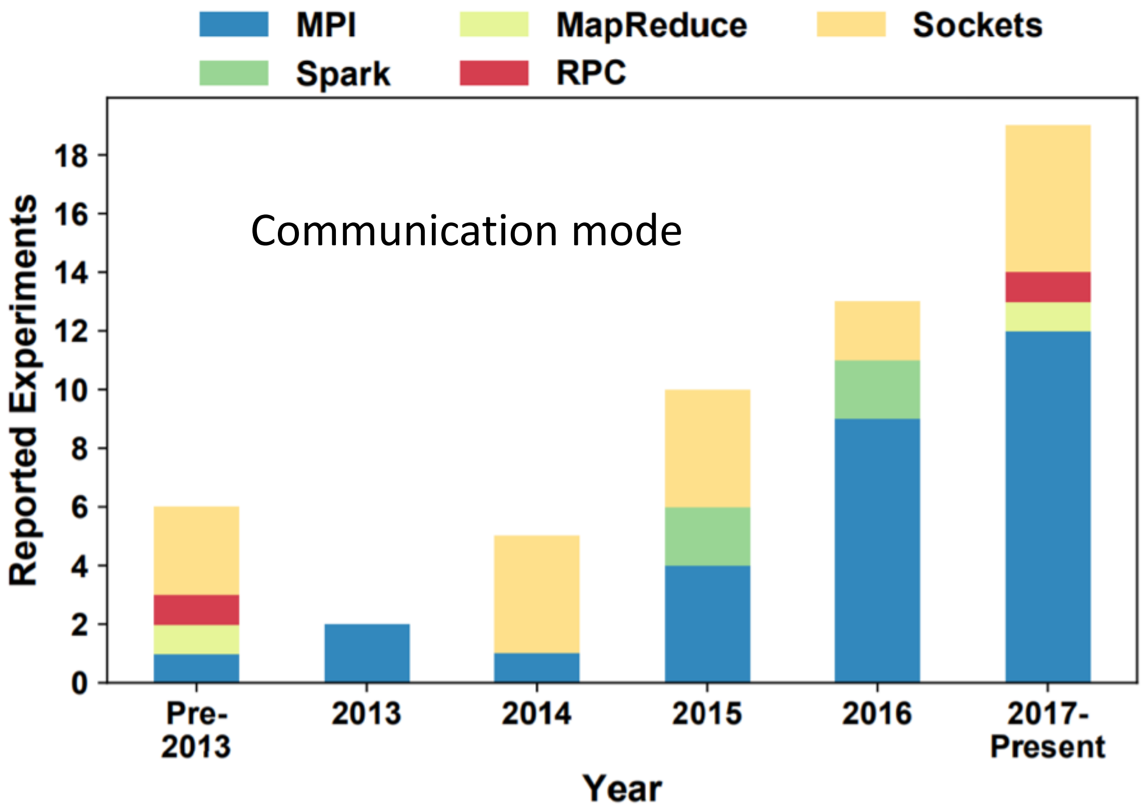
Deep Learning research is converging to MPI!

T. Ben-Nun, T. Hoefler: Demystifying Parallel and Distributed Deep Learning: An In-Depth Concurrency Analysis, CSUR 2019

Trends in distributed deep learning: node count and communication

The field is moving fast – trying everything imaginable – survey results from 252 papers in the area of parallel deep learning



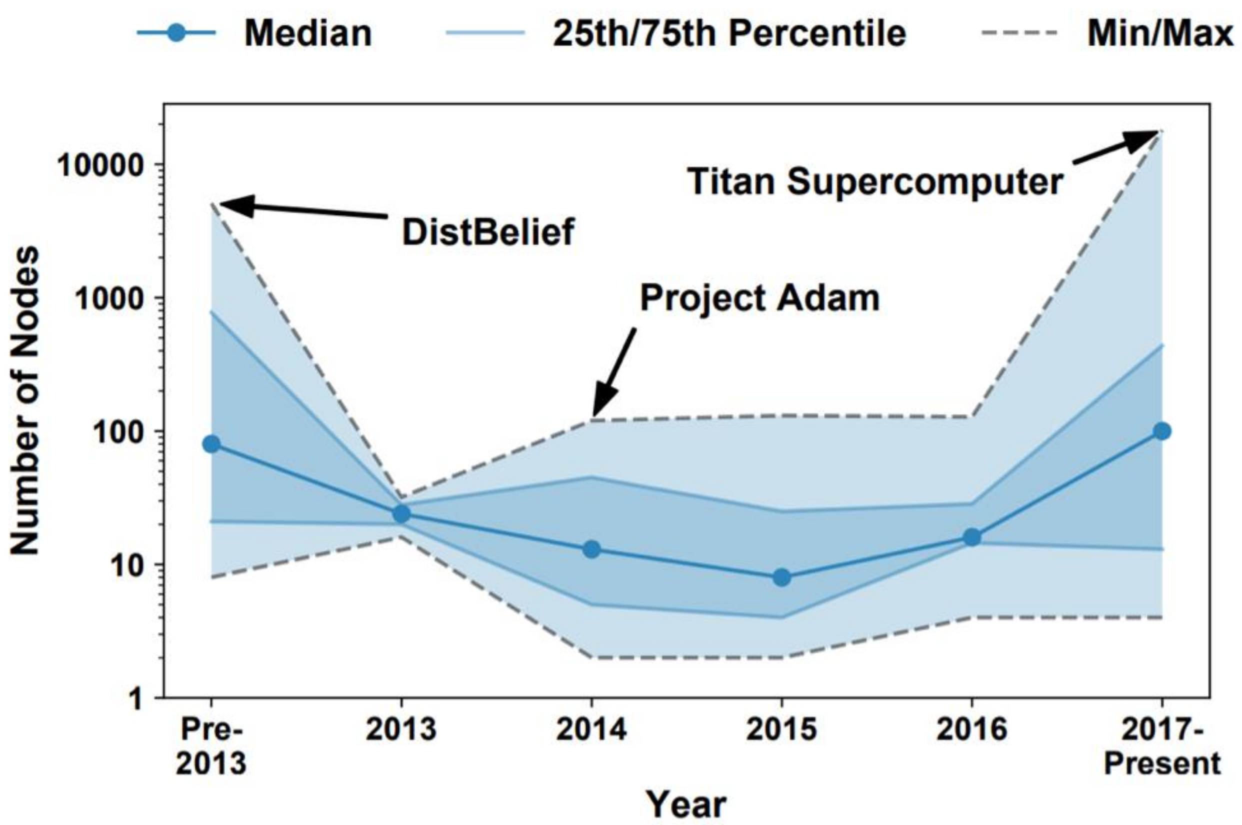


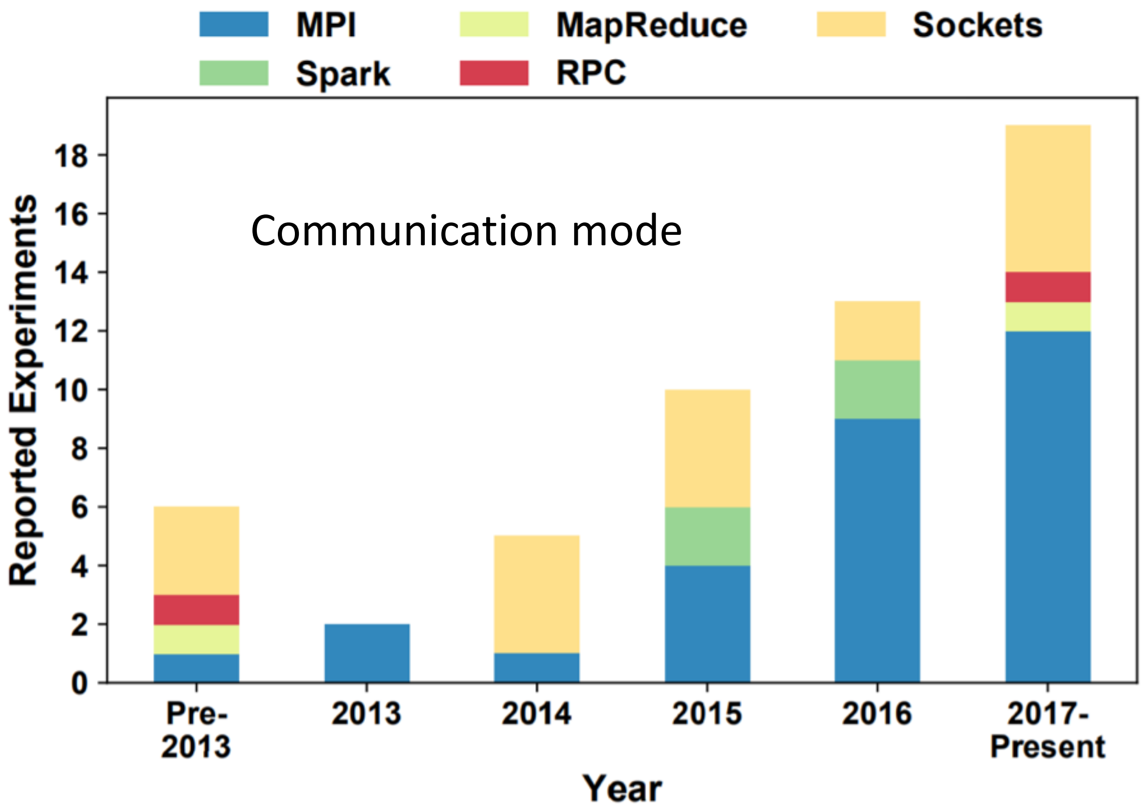
Deep Learning research is converging to MPI!

T. Ben-Nun, T. Hoefler: Demystifying Parallel and Distributed Deep Learning: An In-Depth Concurrency Analysis, CSUR 2019

Trends in distributed deep learning: node count and communication

The field is moving fast – trying everything imaginable – survey results from 252 papers in the area of parallel deep learning

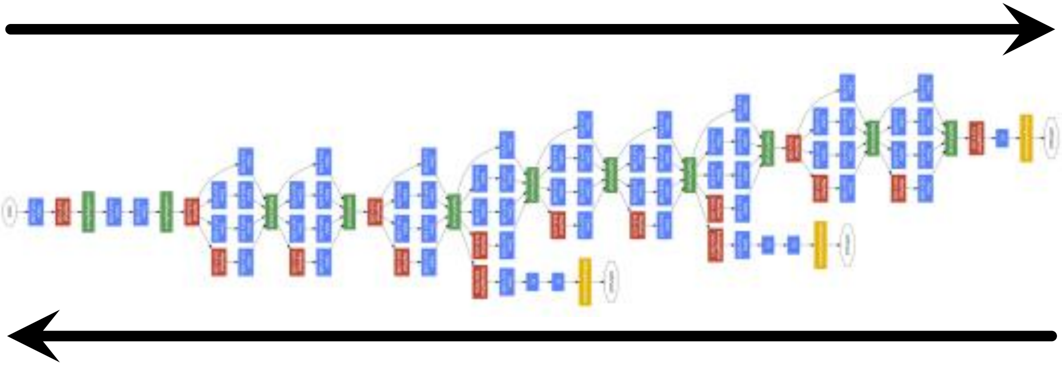




Deep Learning research is converging to MPI!

T. Ben-Nun, T. Hoefler: Demystifying Parallel and Distributed Deep Learning: An In-Depth Concurrency Analysis, CSUR 2019

Minibatch Stochastic Gradient Descent (SGD)



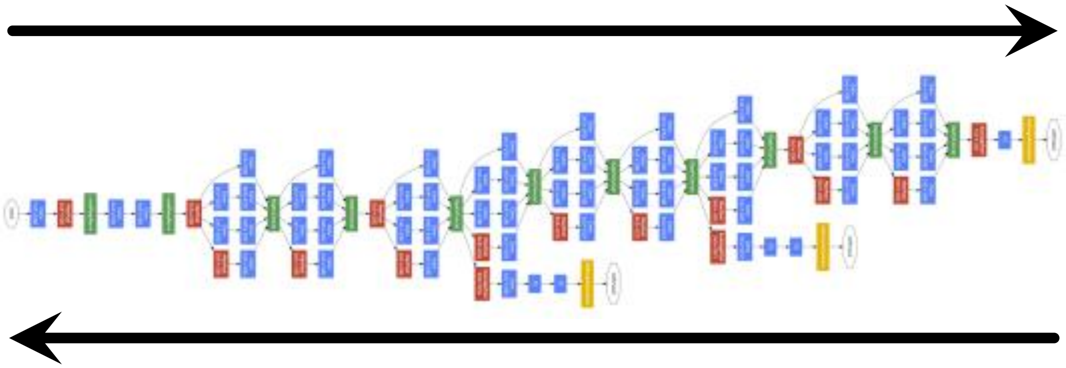
Cat	0.54
Dog	0.28
Airplane	0.07
Horse	0.04
Bicycle	0.03
Truck	0.02
	0.02

Cat	1.00
Dog	0.00
Airplane	0.00
Horse	0.00
Bicycle	0.00
Truck	0.00
	0.00

```
for t = 0 to T do                                          ▷ Stopping condition
    x ← Random element from X                              ▷ Sample dataset
    o_1 ← f_1(x)
    for i = 2 to layers do                                 ▷ Forward evaluation of ℓ
        o_i ← f_i(o_{i−1})
    end for
    for i = layers−1 to 1 do                               ▷ Compute gradient of ℓ via backpropagation
        ∇o_i ← ∂ℓ/∂o_i (o_{i−1}, o_i, ∇o_{i+1})           ▷ Gradient w.r.t. data
        ∇w_i^(t) ← ∂ℓ/∂w_i (o_{i−1}, o_i, ∇o_{i+1})       ▷ Gradient w.r.t. layer parameters
    end for
    w^(t+1) ← w^(t) + u(∇w^(t), w^(0,...,t), t)            ▷ Update weights with function u
end for
```

T. Ben-Nun, T. Hoefler: Demystifying Parallel and Distributed Deep Learning: An In-Depth Concurrency Analysis, CSUR 2019

Minibatch Stochastic Gradient Descent (SGD)



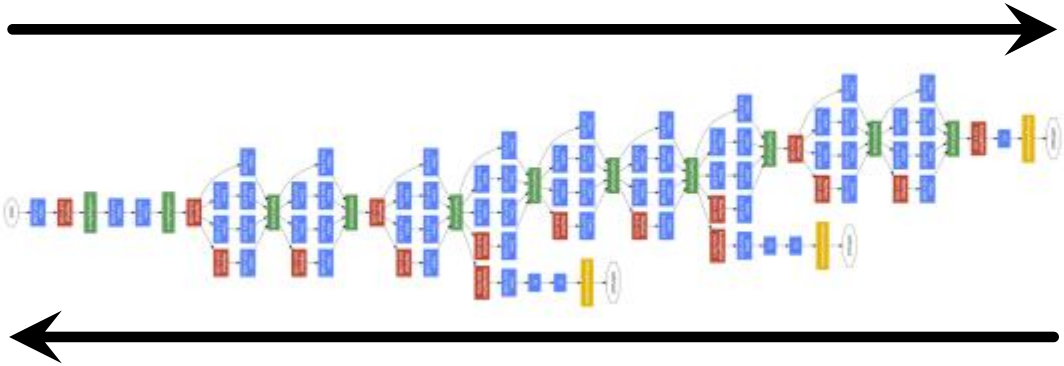
Cat	0.54
Dog	0.28
Airplane	0.07
Horse	0.04
Bicycle	0.03
Truck	0.02
	0.02

Cat	1.00
Dog	0.00
Airplane	0.00
Horse	0.00
Bicycle	0.00
Truck	0.00
	0.00

for $t = 0$ **to** T **do** ▷ Stopping condition
  $\overrightarrow{x} \leftarrow$ Random B elements from X ▷ Sample dataset
  $\overrightarrow{o_1} \leftarrow f_1(\overrightarrow{x})$
  for $i = 2$ **to** *layers* **do** ▷ Forward evaluation of ℓ
    $\overrightarrow{o_i} \leftarrow f_i(\overrightarrow{o_{i-1}})$
  end for
  for $i =$ *layers*-1 **to** 1 **do** ▷ Compute gradient of ℓ via backpropagation
    $\overrightarrow{\nabla o_i} \leftarrow \frac{\partial \ell}{\partial o_i}(\overrightarrow{o_{i-1}}, \overrightarrow{o_i}, \overrightarrow{\nabla o_{i+1}})$ ▷ Gradient w.r.t. data
    $\overrightarrow{\nabla w_i^{(t)}} \leftarrow \frac{\partial \ell}{\partial w_i}(\overrightarrow{o_{i-1}}, \overrightarrow{o_i}, \overrightarrow{\nabla o_{i+1}})$ ▷ Gradient w.r.t. layer parameters
  end for
  $\nabla w^{(t)} \leftarrow \frac{1}{B} \sum_{i=0}^{B} \overrightarrow{\nabla w^{(t)}}[i]$
  $w^{(t+1)} \leftarrow w^{(t)} + u(\nabla w^{(t)}, w^{(0,\dots,t)}, t)$ ▷ Update weights with function u
end for

T. Ben-Nun, T. Hoefler: Demystifying Parallel and Distributed Deep Learning: An In-Depth Concurrency Analysis, CSUR 2019

Minibatch Stochastic Gradient Descent (SGD)

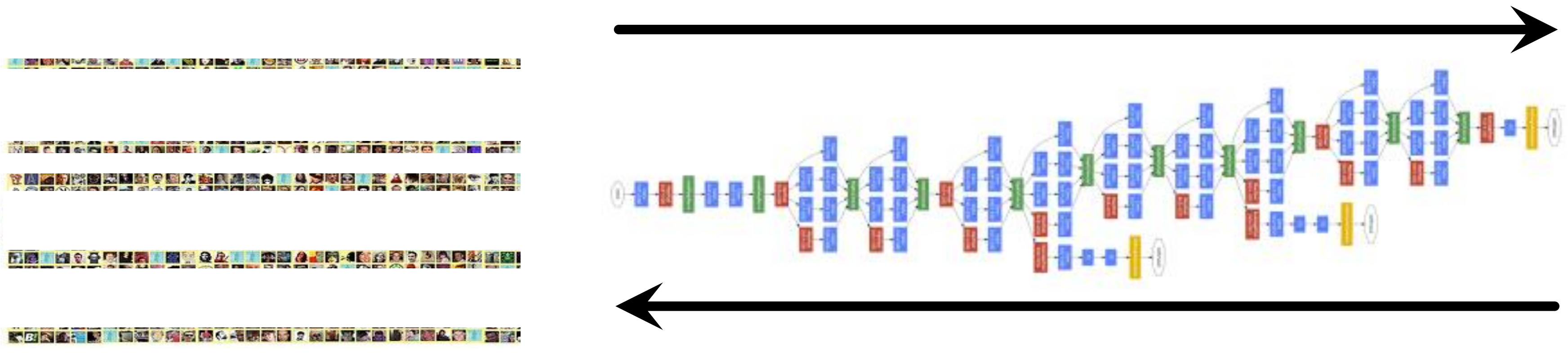


Cat	0.54	1.00
Dog	0.28	0.00
Airplane	0.07	0.00
Horse	0.04	0.00
Bicycle	0.03	0.00
Truck	0.02	0.00
Truck	0.02	0.00

for $t = 0$ **to** T **do** ▷ Stopping condition
  $\overrightarrow{x} \leftarrow$ Random B elements from X ▷ Sample dataset
  $\overrightarrow{o_1} \leftarrow f_1(\overrightarrow{x})$
  for $i = 2$ **to** $layers$ **do** ▷ Forward evaluation of ℓ
    $\overrightarrow{o_i} \leftarrow f_i(\overrightarrow{o_{i-1}})$
  end for
  for $i = layers-1$ **to** 1 **do** ▷ Compute gradient of ℓ via backpropagation
    $\overrightarrow{\nabla o_i} \leftarrow \frac{\partial \ell}{\partial o_i}(\overrightarrow{o_{i-1}}, \overrightarrow{o_i}, \overrightarrow{\nabla o_{i+1}})$ ▷ Gradient w.r.t. data
    $\overrightarrow{\nabla w_i^{(t)}} \leftarrow \frac{\partial \ell}{\partial w_i}(\overrightarrow{o_{i-1}}, \overrightarrow{o_i}, \overrightarrow{\nabla o_{i+1}})$ ▷ Gradient w.r.t. layer parameters
  end for
  $\nabla w^{(t)} \leftarrow \frac{1}{B}\sum_{i=0}^{B} \overrightarrow{\nabla w^{(t)}}[i]$
  $w^{(t+1)} \leftarrow w^{(t)} + u(\nabla w^{(t)}, w^{(0,\dots,t)}, t)$ ▷ Update weights with function u
end for

T. Ben-Nun, T. Hoefler: Demystifying Parallel and Distributed Deep Learning: An In-Depth Concurrency Analysis, CSUR 2019

Minibatch Stochastic Gradient Descent (SGD)



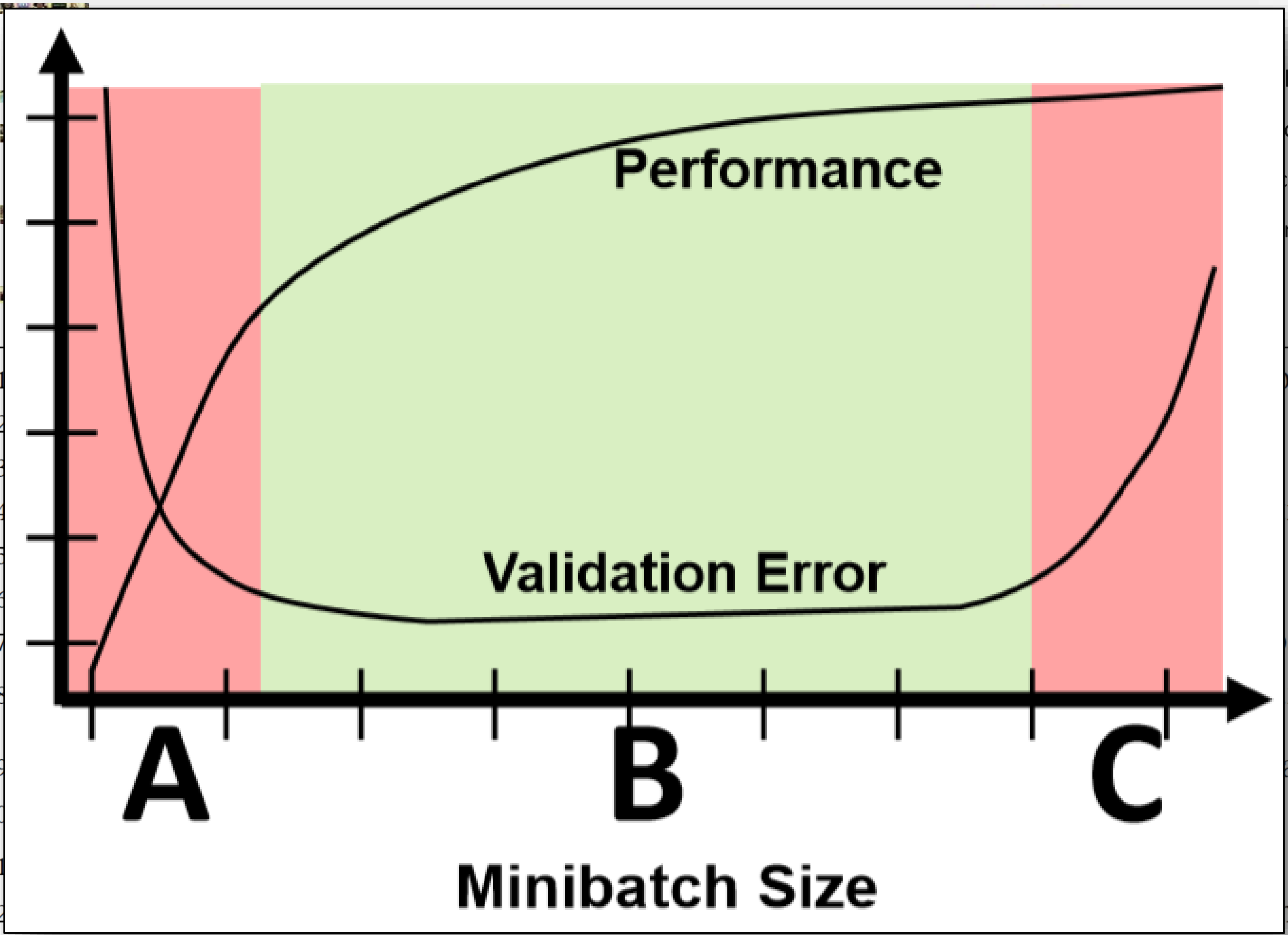
Cat	0.54	Cat	1.00
Dog	0.28	Dog	0.00
Airplane	0.07	Airplane	0.00
Horse	0.04	Horse	0.00
Bicycle	0.03	Bicycle	0.00
Truck	0.02	Truck	0.00
	0.02		0.00

for $t = 0$ **to** T **do** ▷ Stopping condition
  $\vec{x} \leftarrow$ Random B elements from X ▷ Sample dataset
  $\vec{o_1} \leftarrow f_1(\vec{x})$
  for $i = 2$ **to** *layers* **do** ▷ Forward evaluation of ℓ
    $\vec{o_i} \leftarrow f_i(\overrightarrow{o_{i-1}})$
  end for
  for $i = layers - 1$ **to** 1 **do** ▷ Compute gradient of ℓ via backpropagation
    $\overrightarrow{\nabla o_i} \leftarrow \frac{\partial \ell}{\partial o_i}(\overrightarrow{o_{i-1}}, \vec{o_i}, \overrightarrow{\nabla o_{i+1}})$ ▷ Gradient w.r.t. data
    $\overrightarrow{\nabla w_i^{(t)}} \leftarrow \frac{\partial \ell}{\partial w_i}(\overrightarrow{o_{i-1}}, \vec{o_i}, \overrightarrow{\nabla o_{i+1}})$ ▷ Gradient w.r.t. layer parameters
  end for
  $\nabla w^{(t)} \leftarrow \frac{1}{B} \sum_{i=0}^{B} \overrightarrow{\nabla w^{(t)}}[i]$
  $w^{(t+1)} \leftarrow w^{(t)} + u(\nabla w^{(t)}, w^{(0,\ldots,t)}, t)$ ▷ Update weights with function u
end for

Minibatch Stochastic Gradient Descent (SGD)

Cat	0.54
Dog	0.28
lane	0.07
orse	0.04
ycle	0.03
ruck	0.02
	0.02

Cat	1.00
Dog	0.00
Airplane	0.00
Horse	0.00
Bicycle	0.00
Truck	0.00



ndition
dataset
ion of ℓ
agation
r.t. data
ameters
nction u
13: **end for**

T. Ben-Nun, T. Hoefler: Demystifying Parallel and Distributed Deep Learning: An In-Depth Concurrency Analysis, CSUR 2019

Ingredients of a neural network: Operators

Ingredients of a neural network: Operators

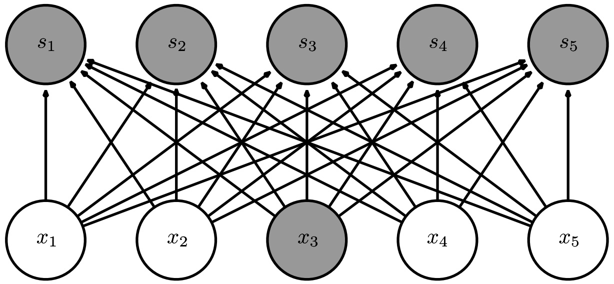
- **Fully-connected layers (multi-layer perceptrons)**

- **Convolution**

- **Many other moving parts:**
  - Pooling
  - Batch normalization [Ioffe & Szegedy 2015]
  - ReLU activations [Glorot, Bordes, & Bengion 2011]
  - …

Ingredients of a neural network: Operators

- **Fully-connected layers (multi-layer perceptrons)**

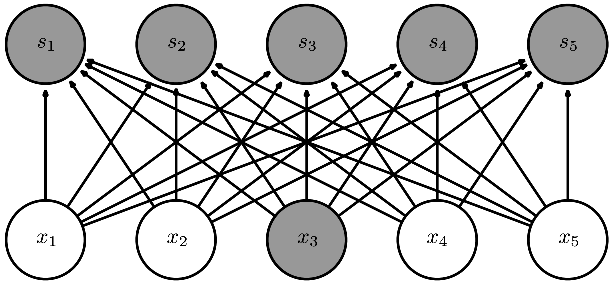


- **Convolution**

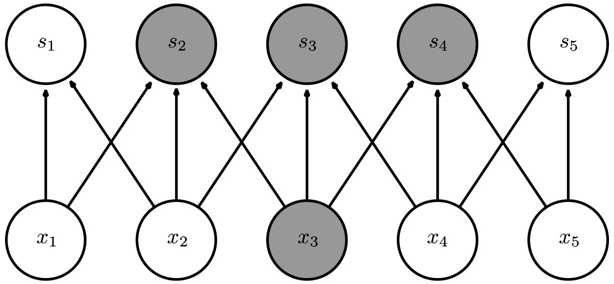
- **Many other moving parts:**
  - Pooling
  - Batch normalization [Ioffe & Szegedy 2015]
  - ReLU activations [Glorot, Bordes, & Bengion 2011]
  - …

Ingredients of a neural network: Operators

- **Fully-connected layers (multi-layer perceptrons)**

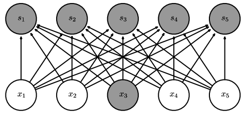


- **Convolution**



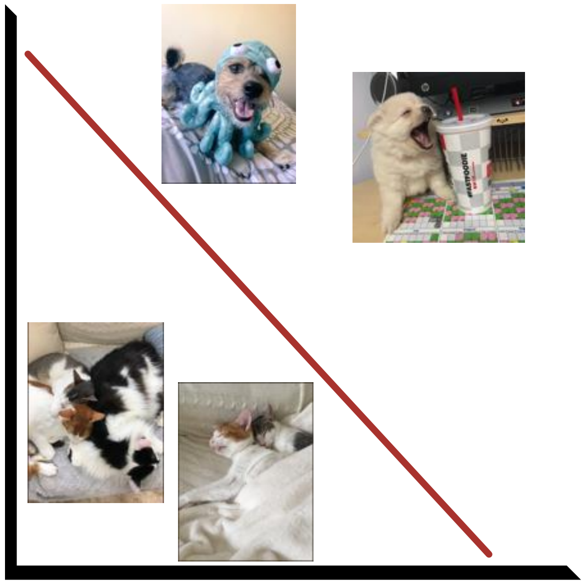
- **Many other moving parts:**
  - Pooling
  - Batch normalization [Ioffe & Szegedy 2015]
  - ReLU activations [Glorot, Bordes, & Bengion 2011]
  - …

Fully-connected layers



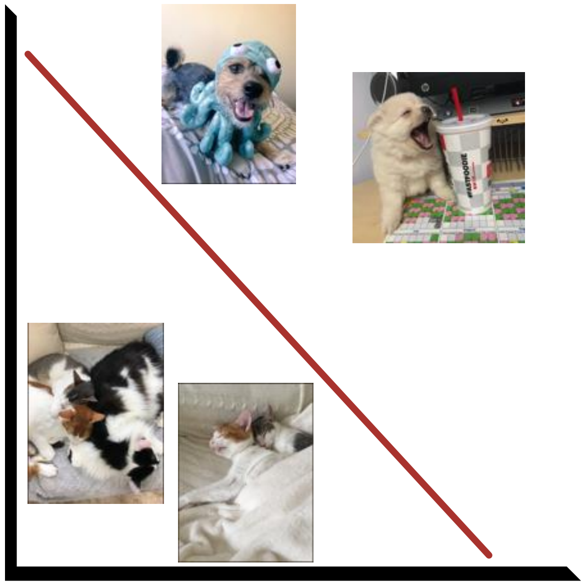
Fully-connected layers

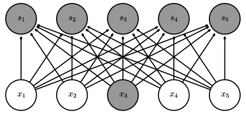
Perceptron [Rosenblatt 1958]



Fully-connected layers

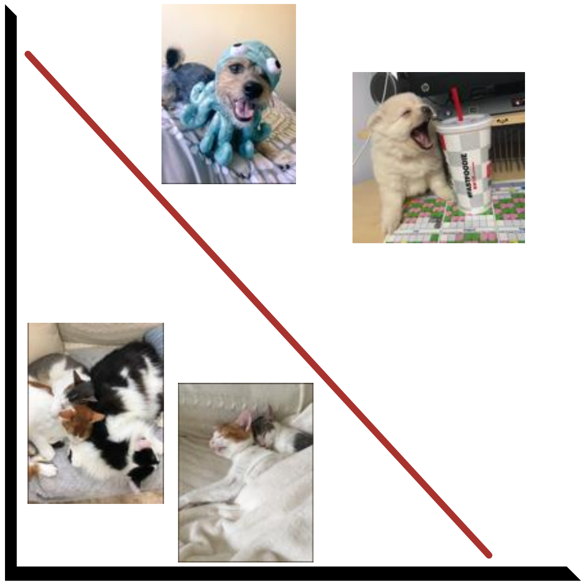
Perceptron [Rosenblatt 1958]





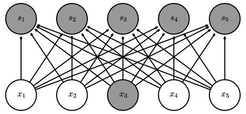
Fully-connected layers

Perceptron [Rosenblatt 1958]

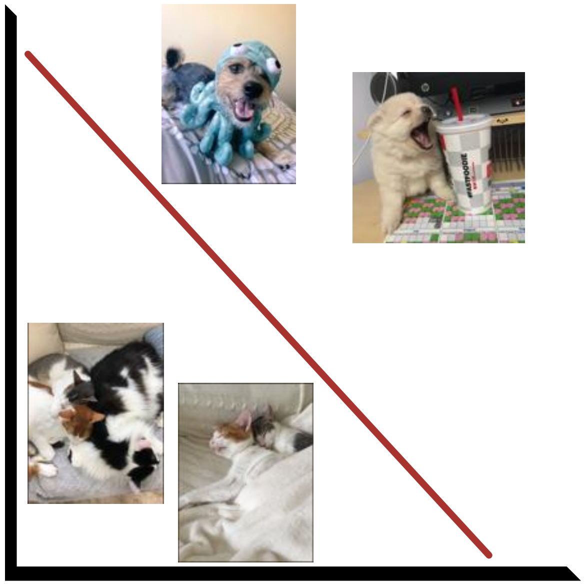


$$y = wx + b$$

Fully-connected layers

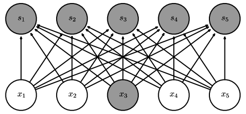


Perceptron [Rosenblatt 1958]



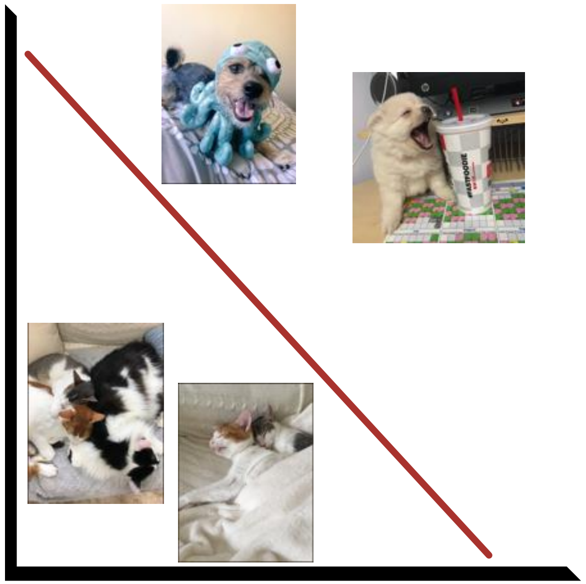
$$y = wx + b$$

Learned weights and bias



Fully-connected layers

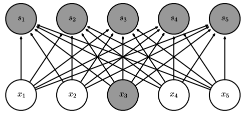
Perceptron [Rosenblatt 1958]



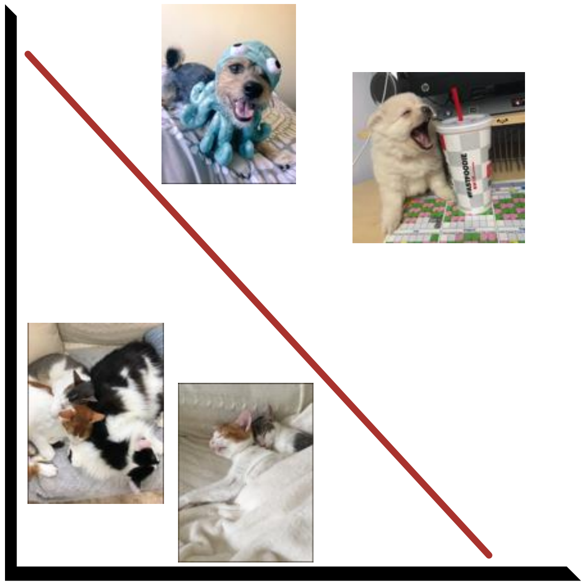
$$y = wx + b$$

Learned weights and bias

Fully-connected layers



Perceptron [Rosenblatt 1958]

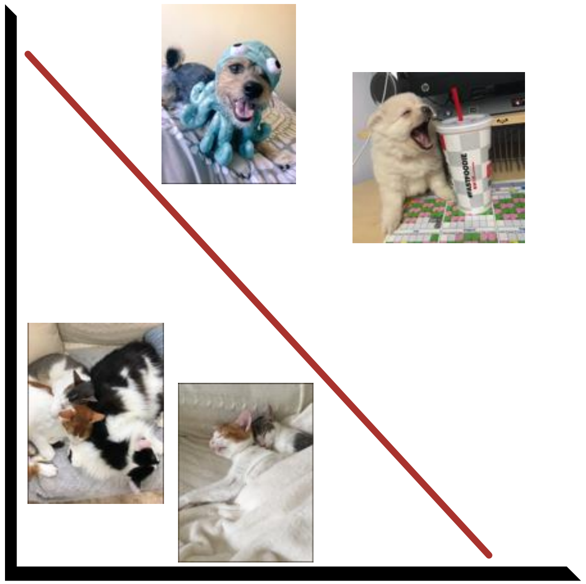


$$y = wx + b$$

Learned weights and bias

Fully-connected layers

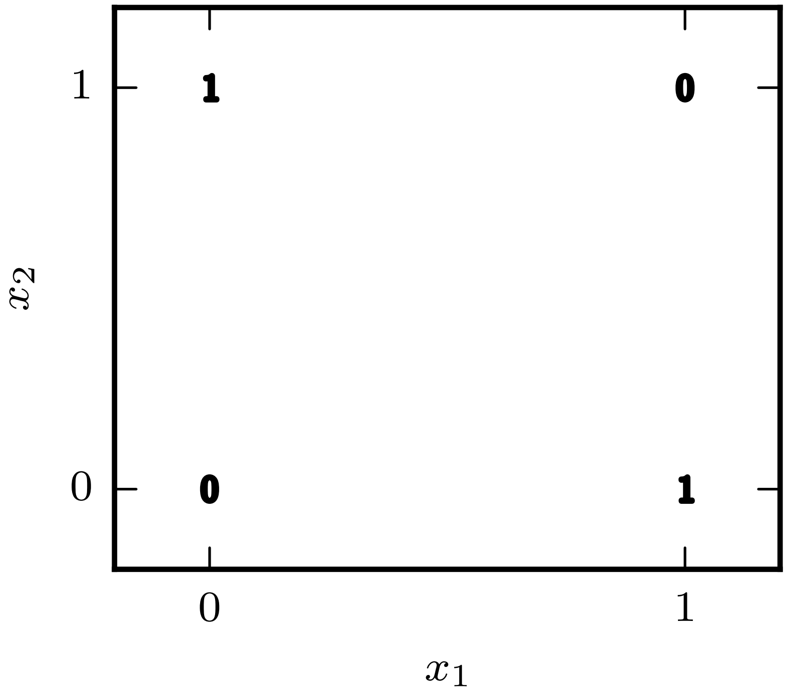
Perceptron [Rosenblatt 1958]



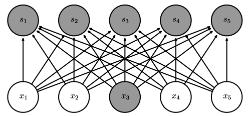
$$y = wx + b$$

Learned weights and bias

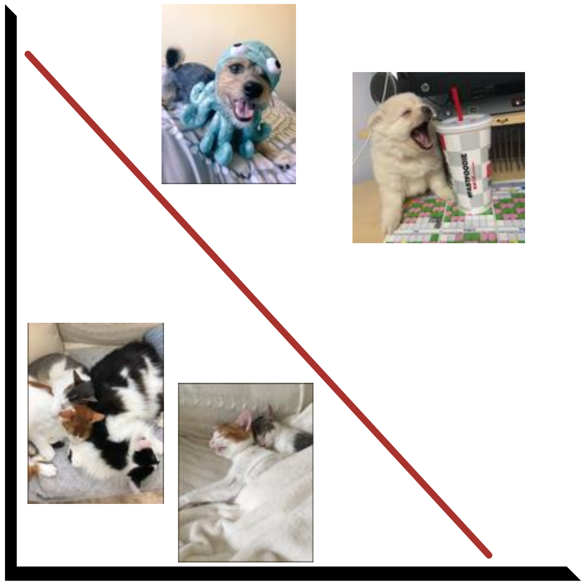
Exclusive OR



Fully-connected layers



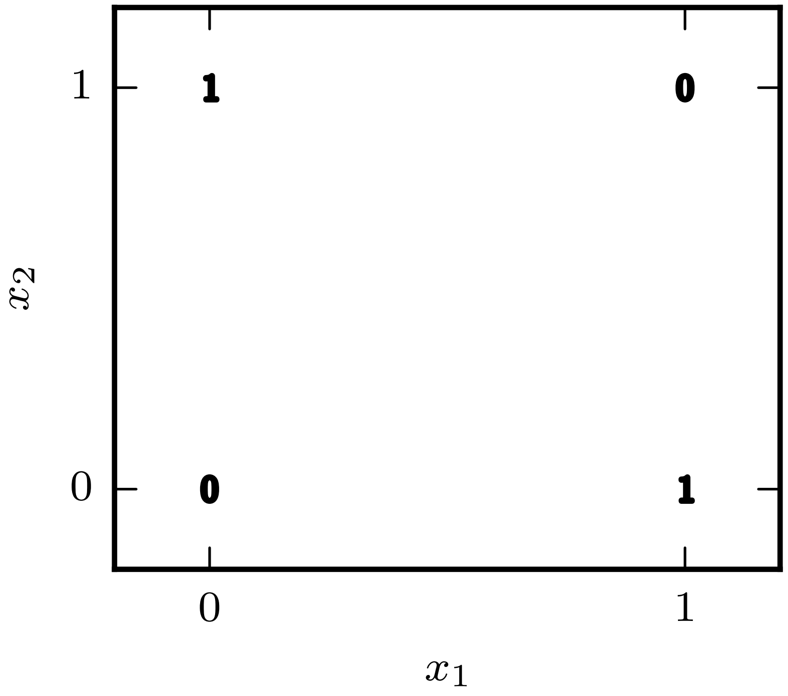
Perceptron [Rosenblatt 1958]



$$y = wx + b$$

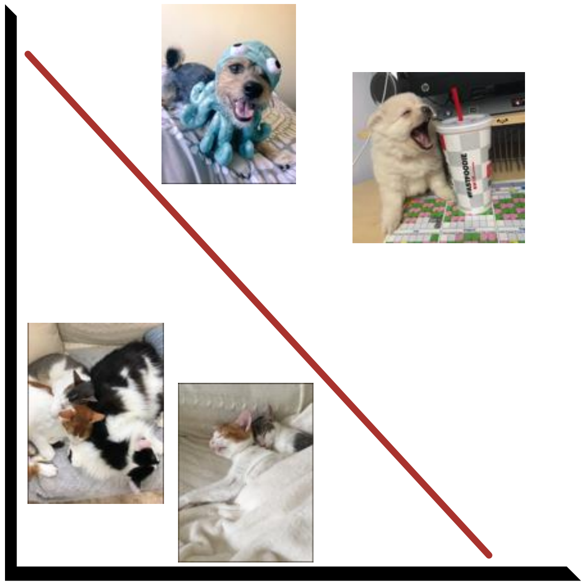
Learned weights and bias

Exclusive OR



Fully-connected layers

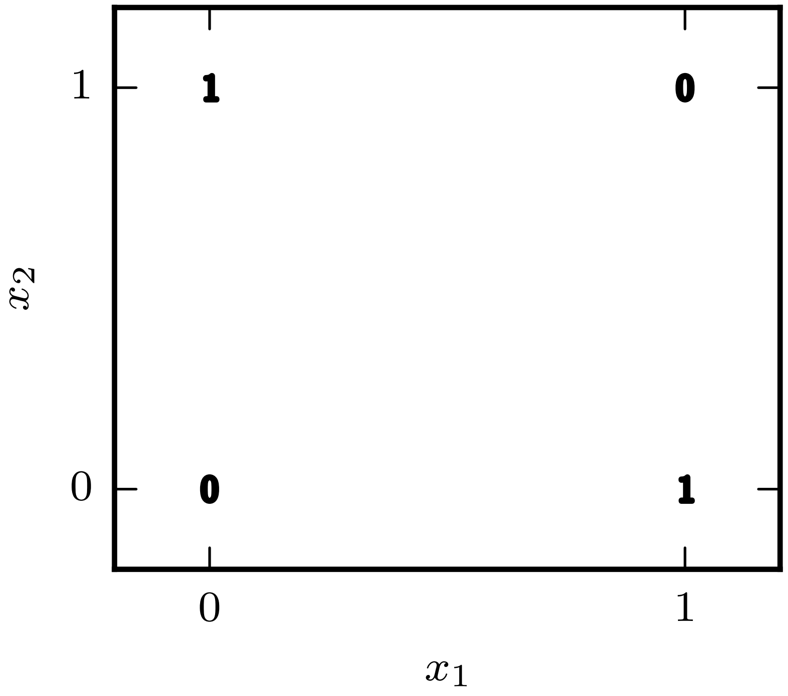
Perceptron [Rosenblatt 1958]



$$y = wx + b$$

Learned weights and bias

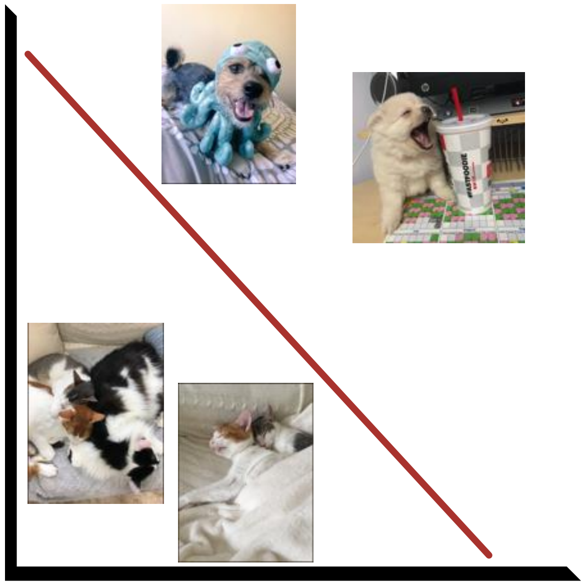
Exclusive OR



Not linearly seperable!

Fully-connected layers

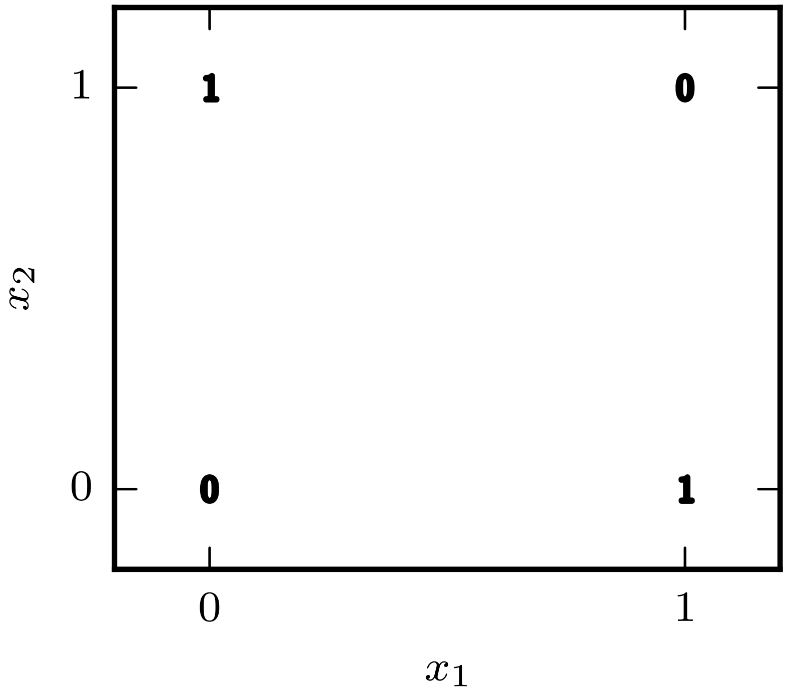
Perceptron [Rosenblatt 1958]



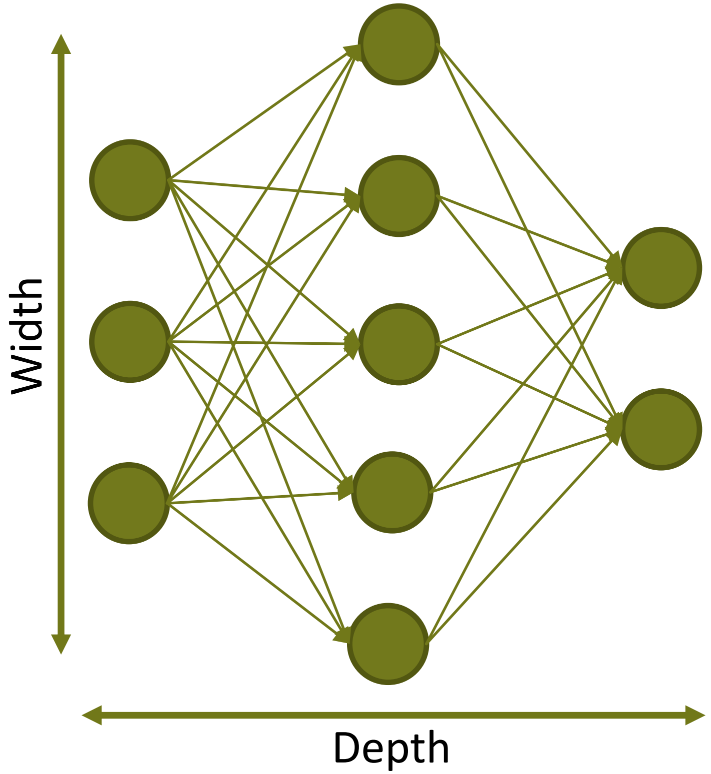
$$y = wx + b$$

Learned weights and bias

Exclusive OR

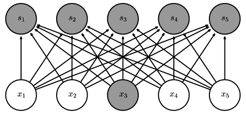


Not linearly seperable!

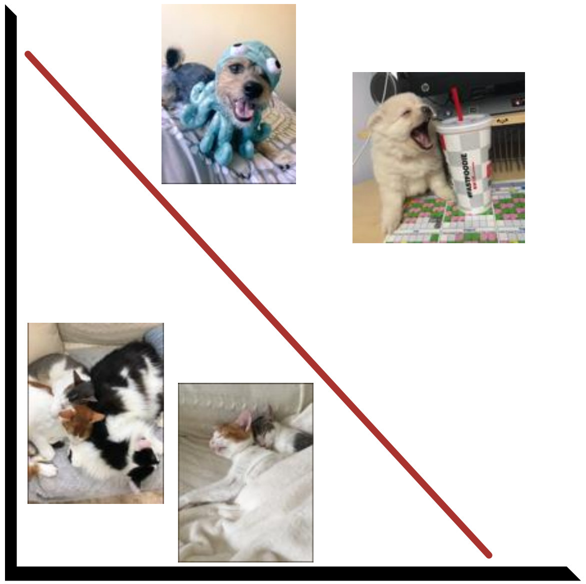


$$y = \sigma(Wx + b)$$

Fully-connected layers



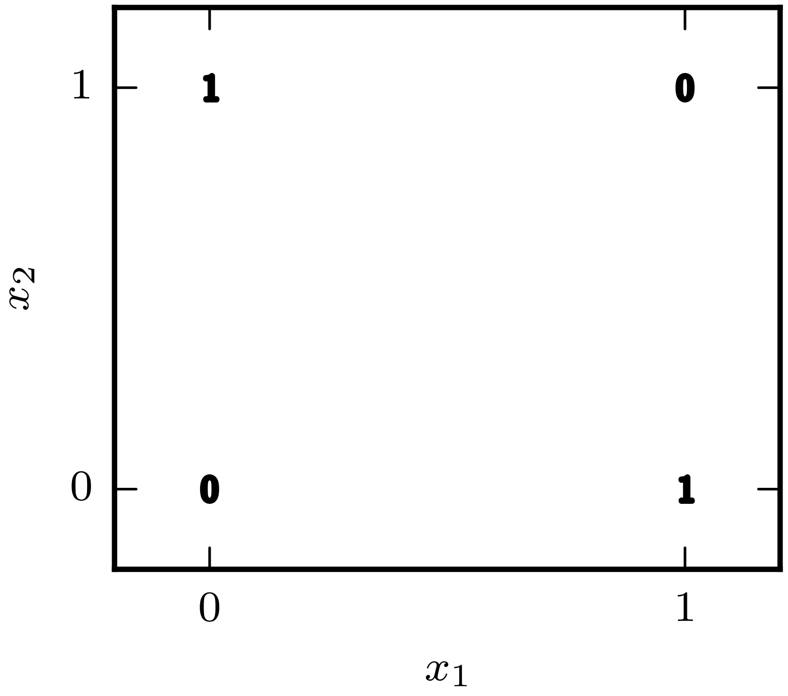
Perceptron [Rosenblatt 1958]



$$y = wx + b$$

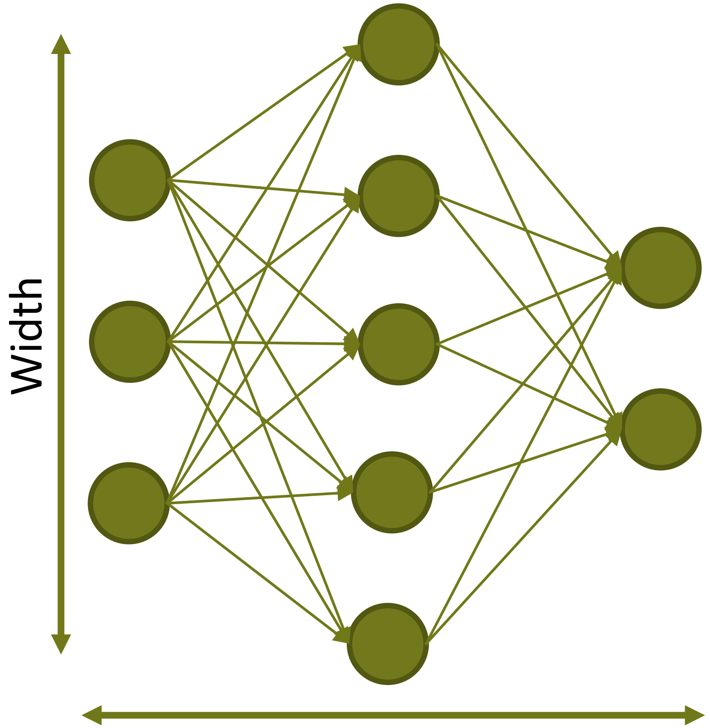
Learned weights and bias

Exclusive OR



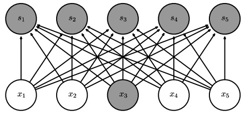
Not linearly seperable!

Multi-layer Perceptron

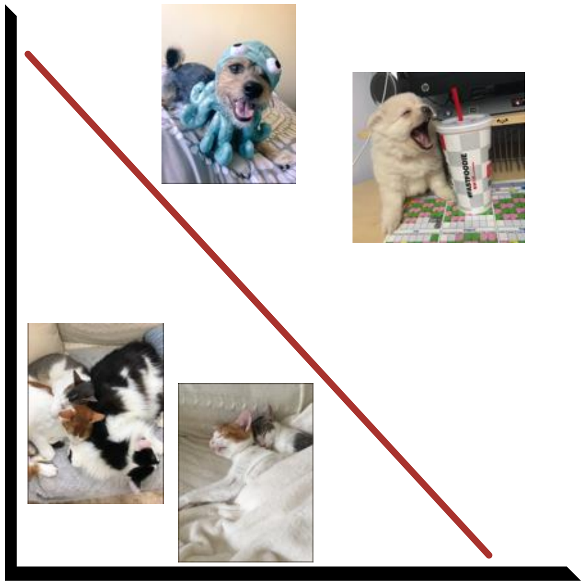


$$y = \sigma(Wx + b)$$

Fully-connected layers



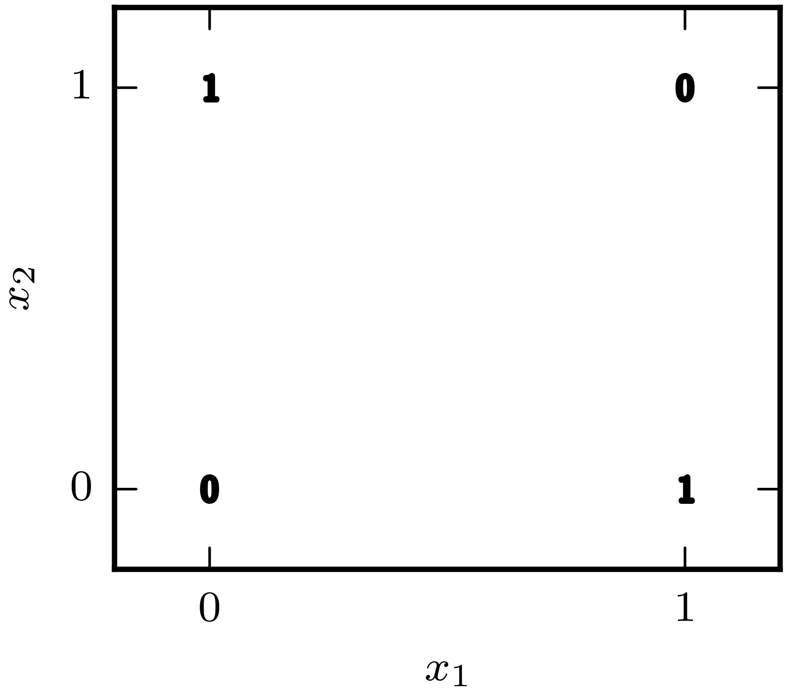
Perceptron [Rosenblatt 1958]



$$y = wx + b$$

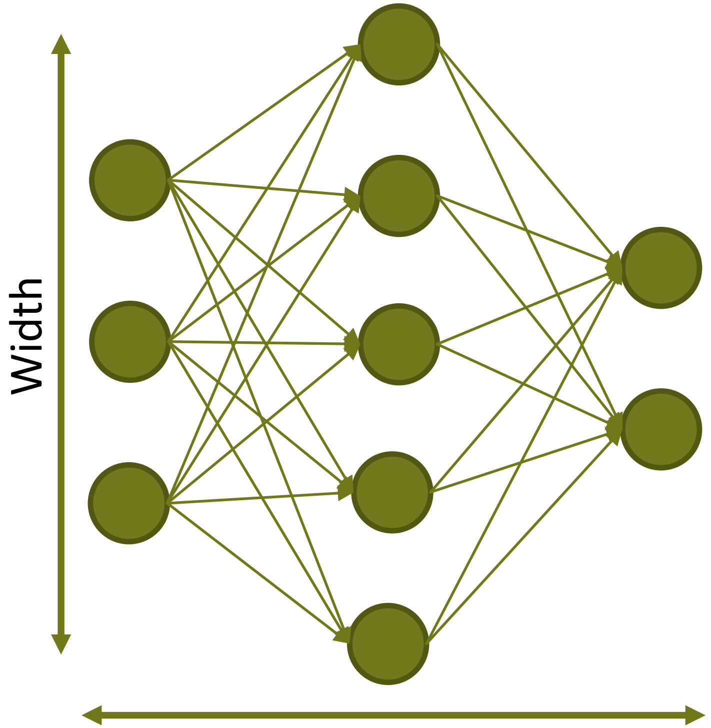
Learned weights and bias

Exclusive OR



Not linearly seperable!

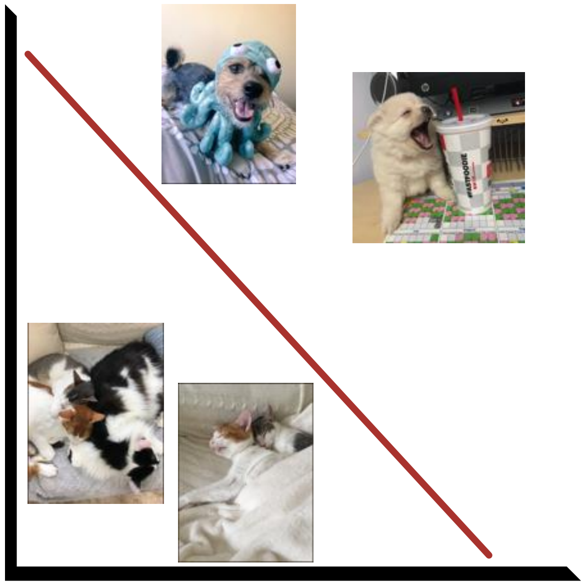
Multi-layer Perceptron



$$y = \sigma(Wx + b)$$

Fully-connected layers

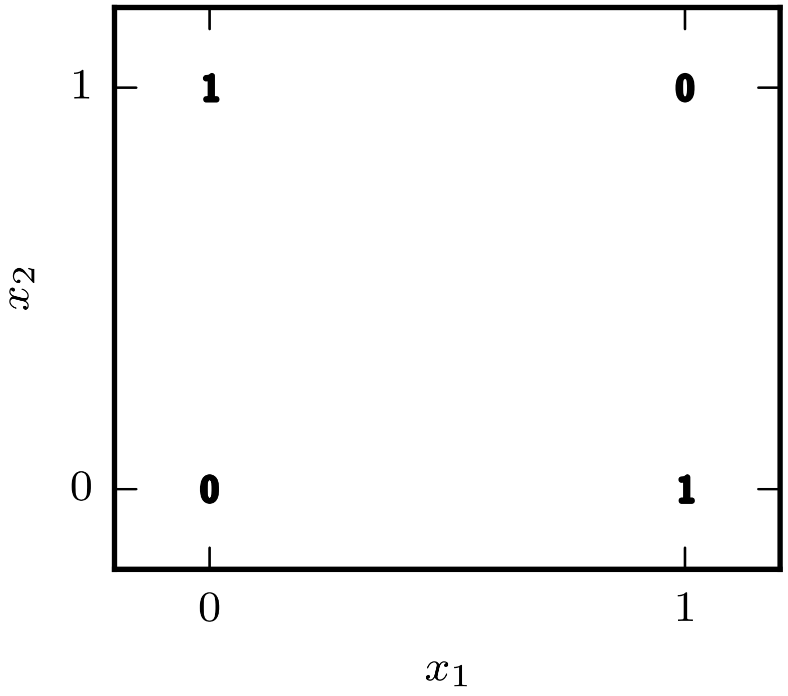
Perceptron [Rosenblatt 1958]



$y = wx + b$

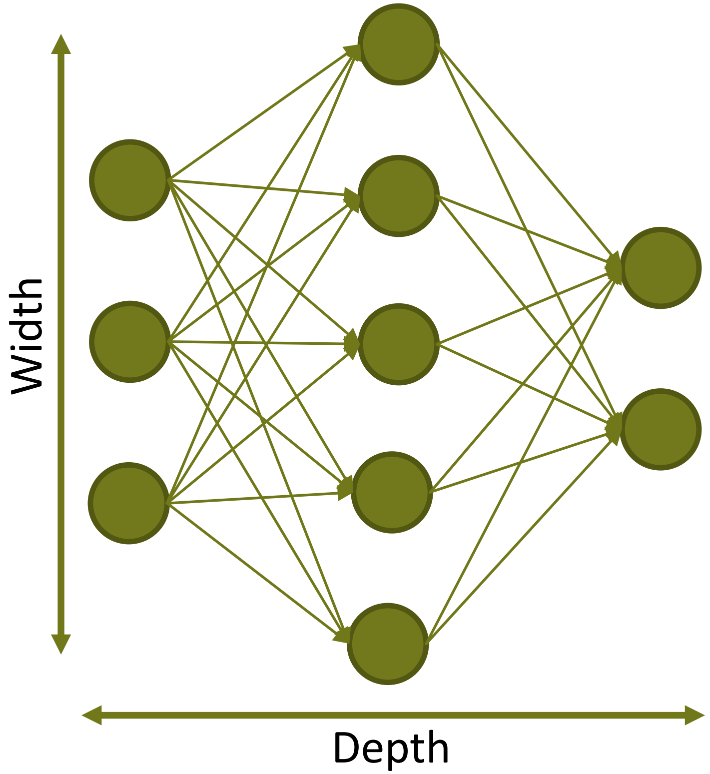
Learned weights and bias

Exclusive OR



Not linearly seperable!

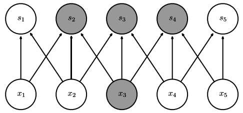
Multi-layer Perceptron



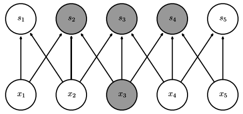
$y = \sigma(Wx + b)$

Universal!

Convolution



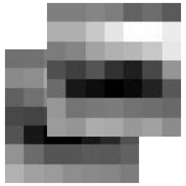
Inputs

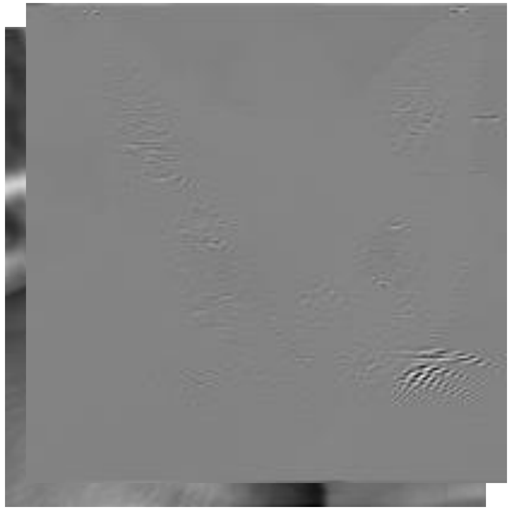


Convolution

Inputs

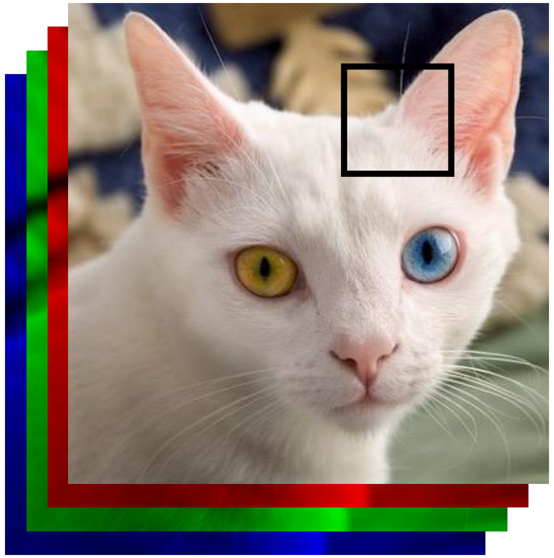
Filters



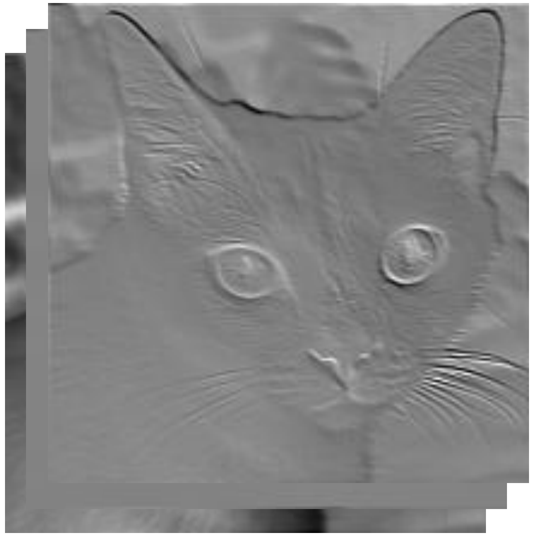


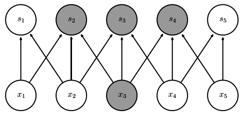
Convolution

Inputs



Filters

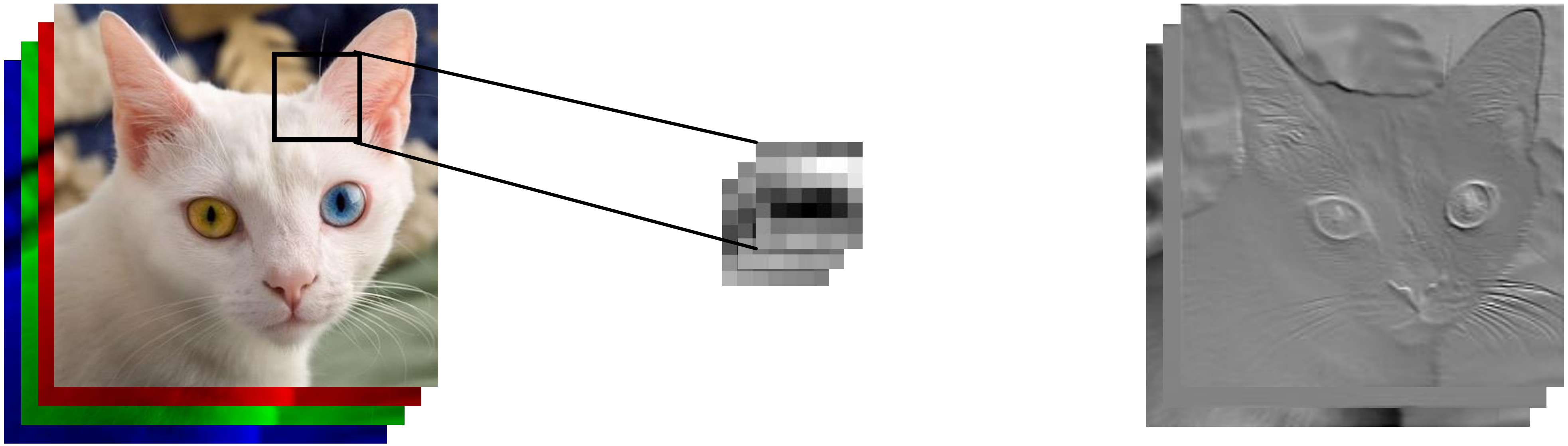




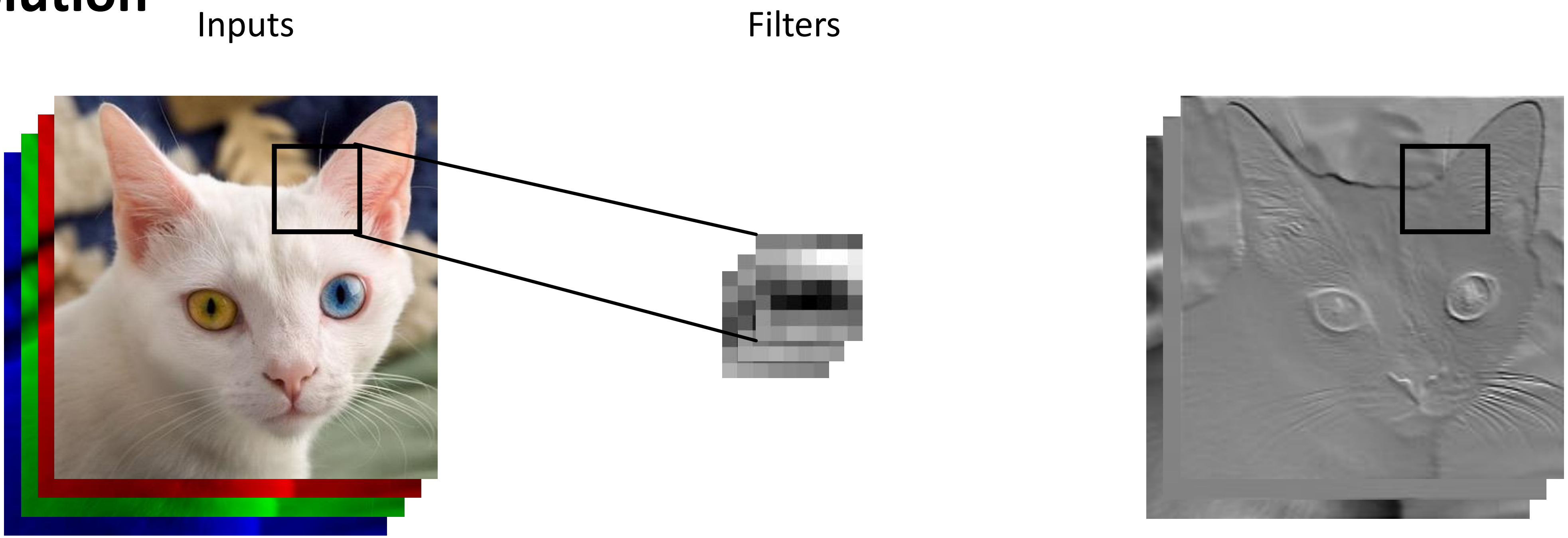
Convolution

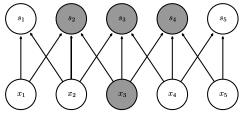
Inputs

Filters



Convolution

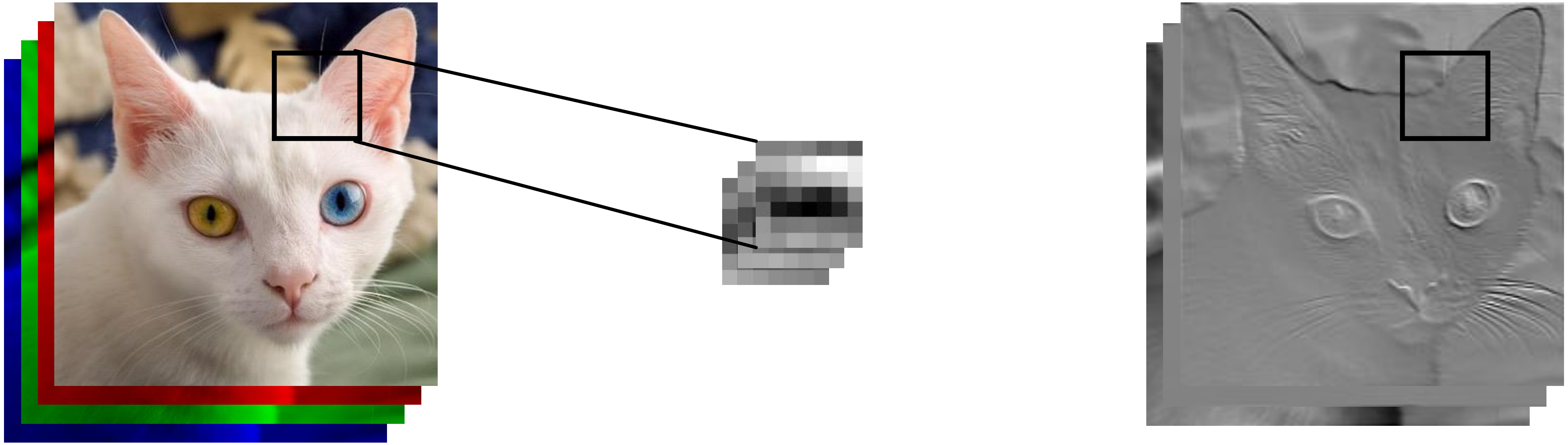




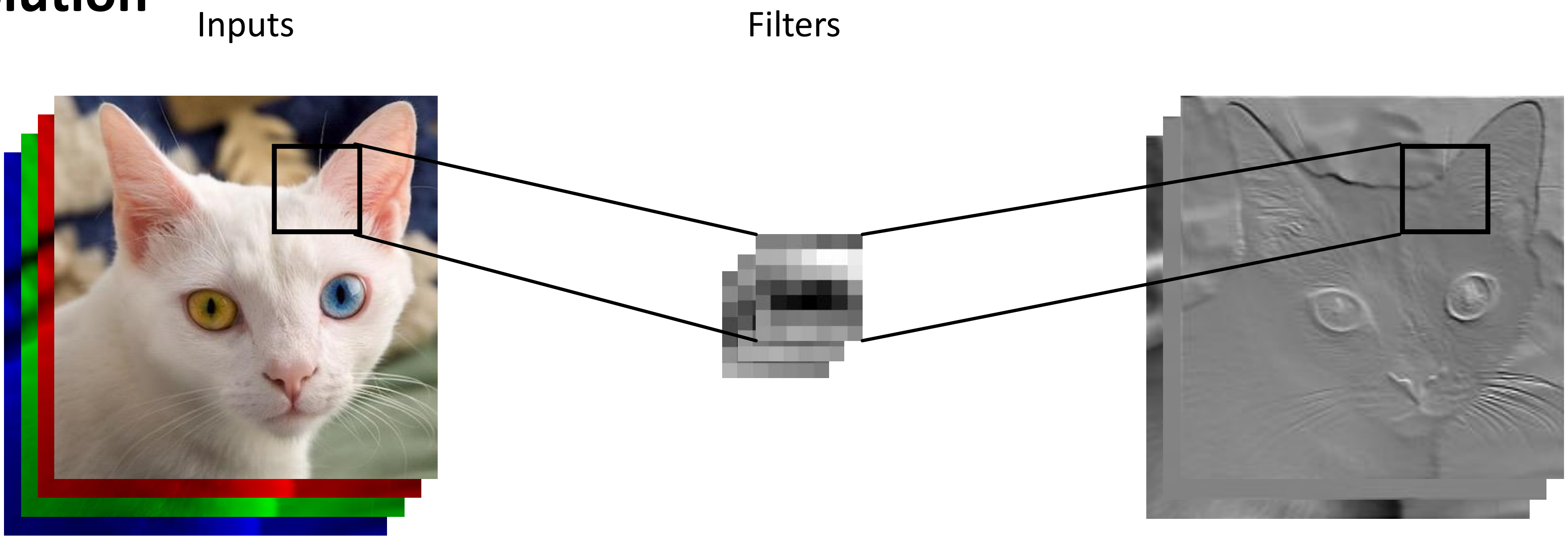
Convolution

Inputs

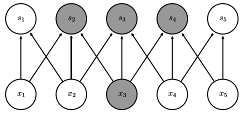
Filters



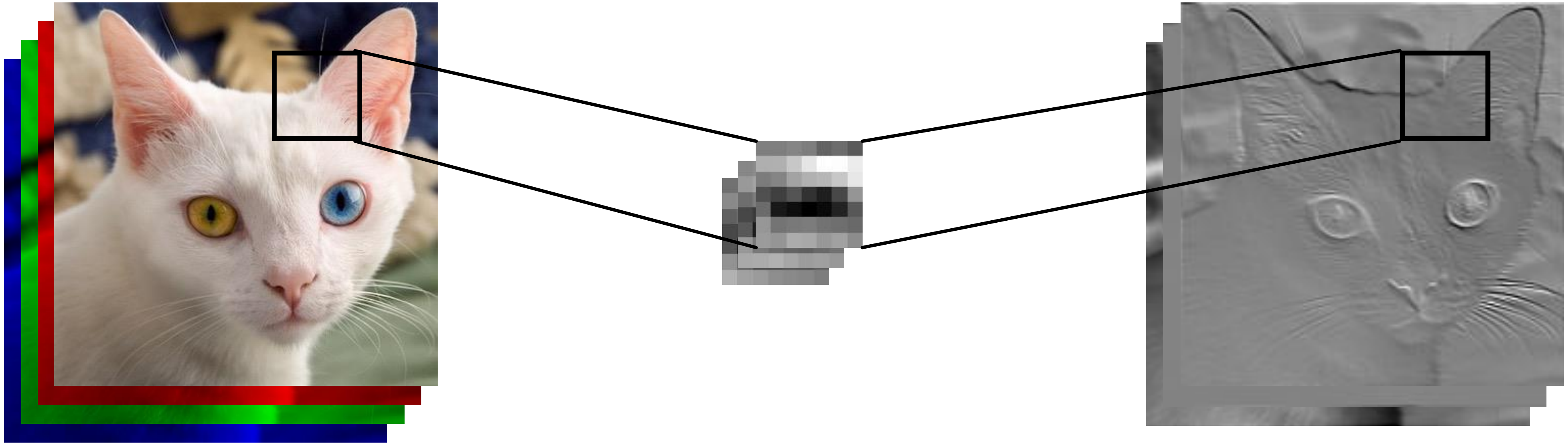
Convolution

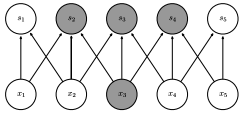


Convolution



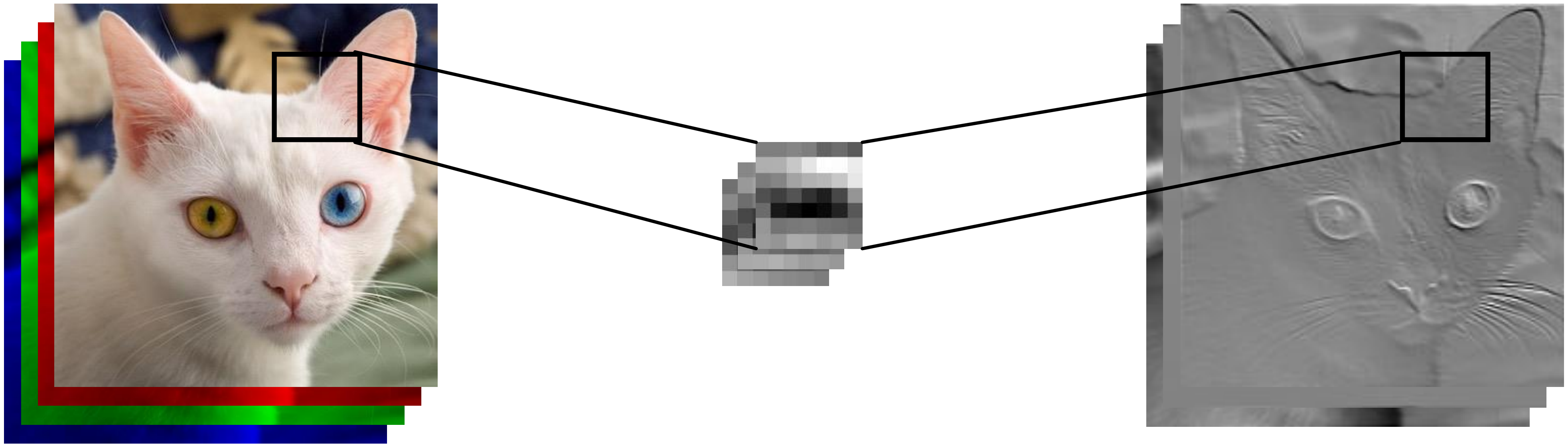
Inputs Filters Feature maps (activations)



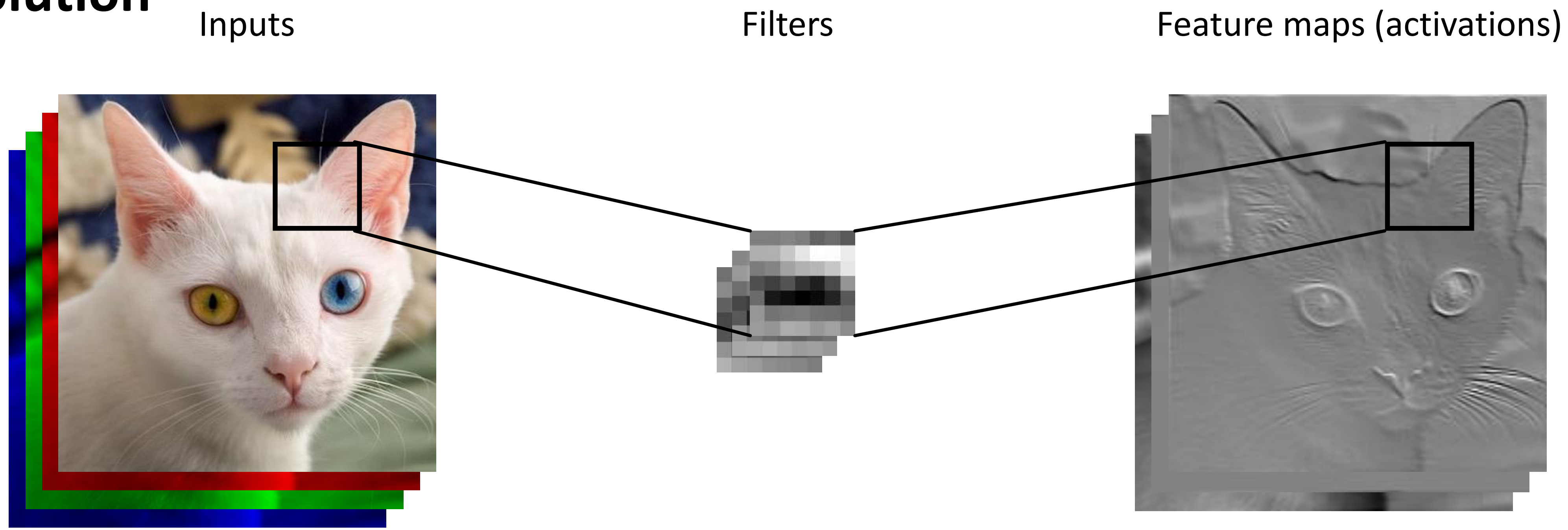


Convolution

Inputs Filters Feature maps (activations)



Convolution

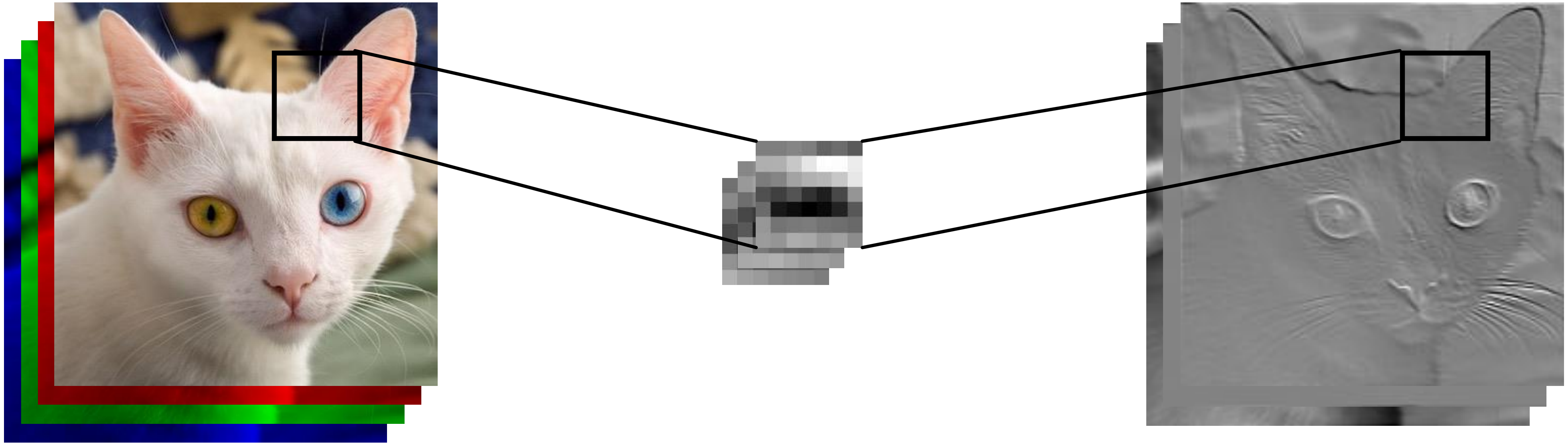


Convolution

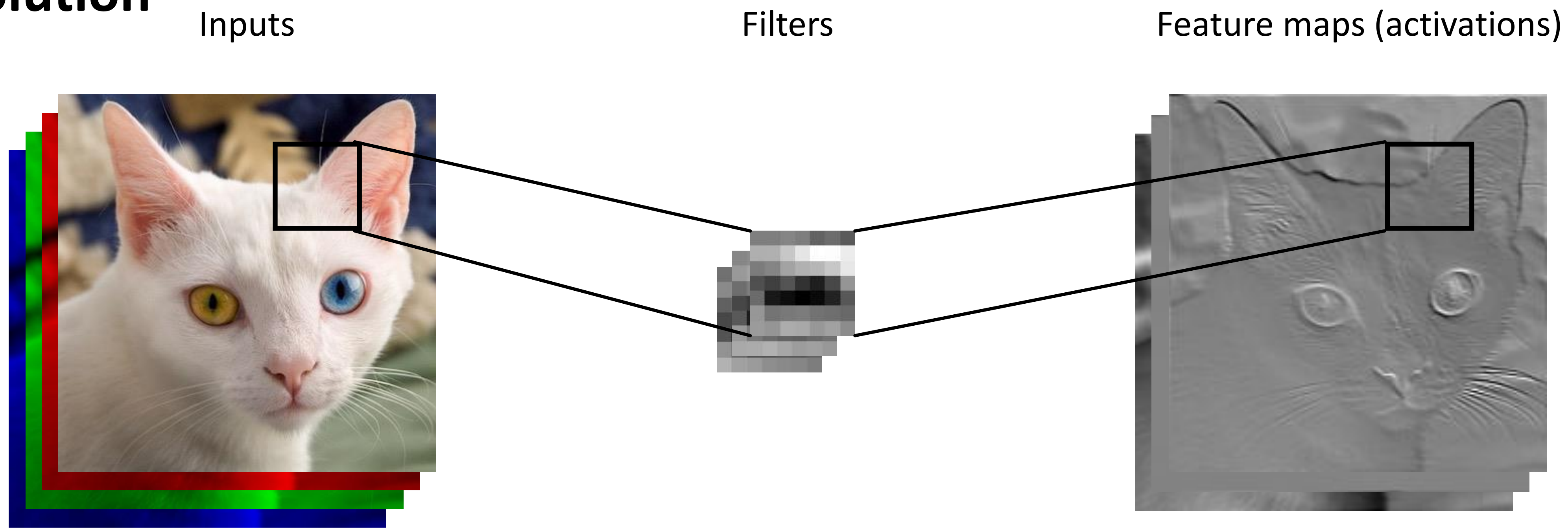
Inputs

Filters

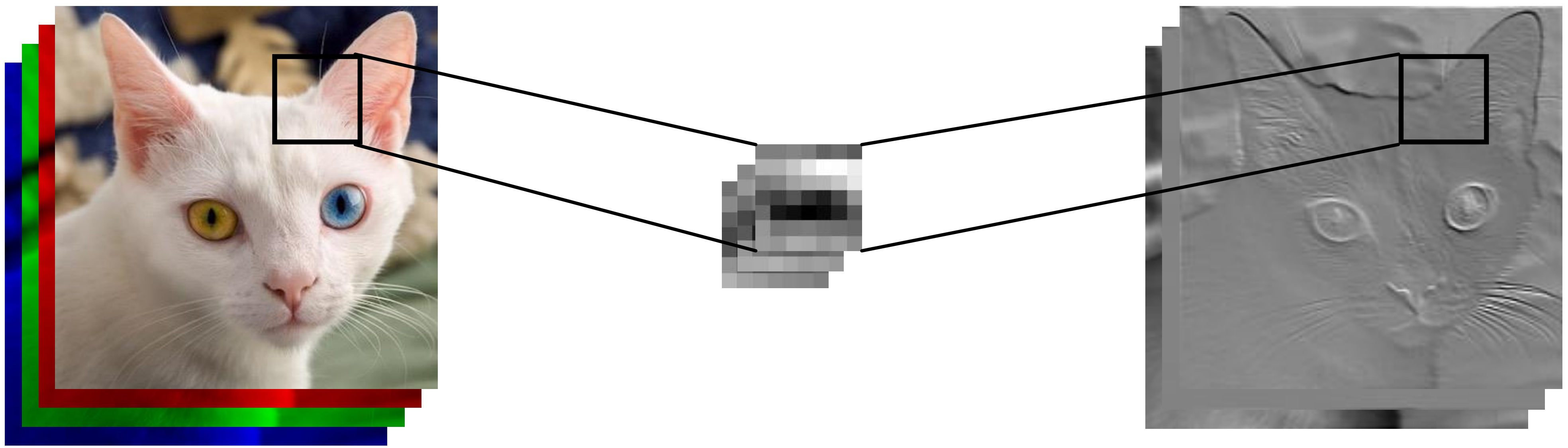
Feature maps (activations)



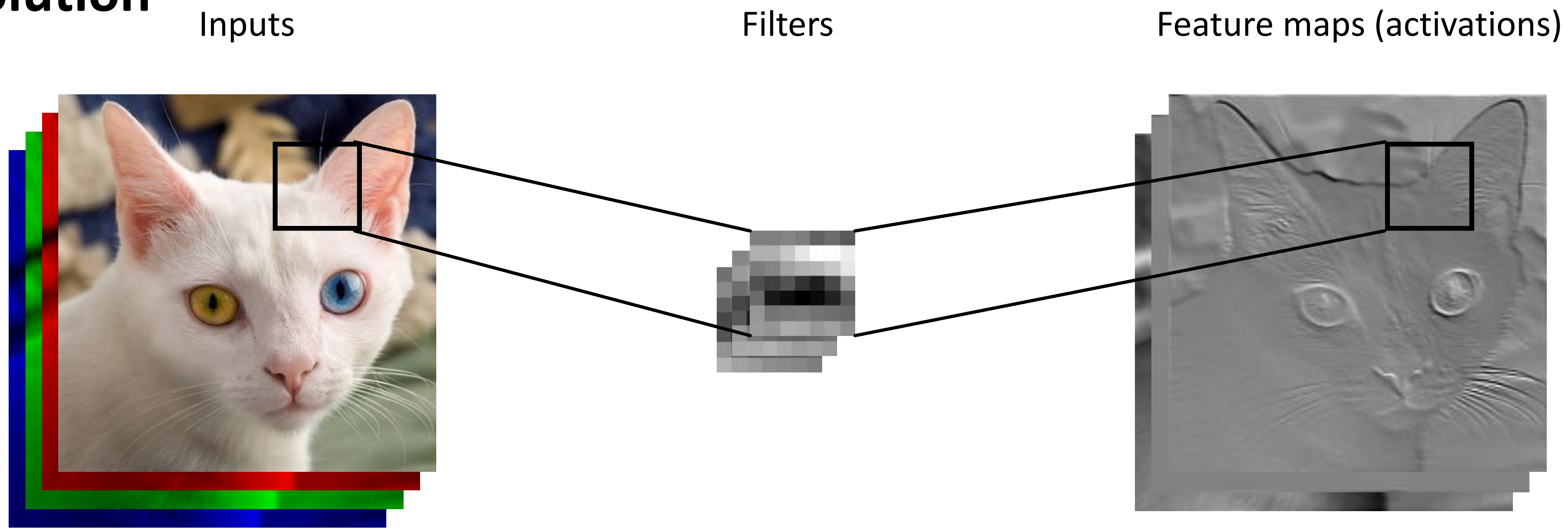
Convolution



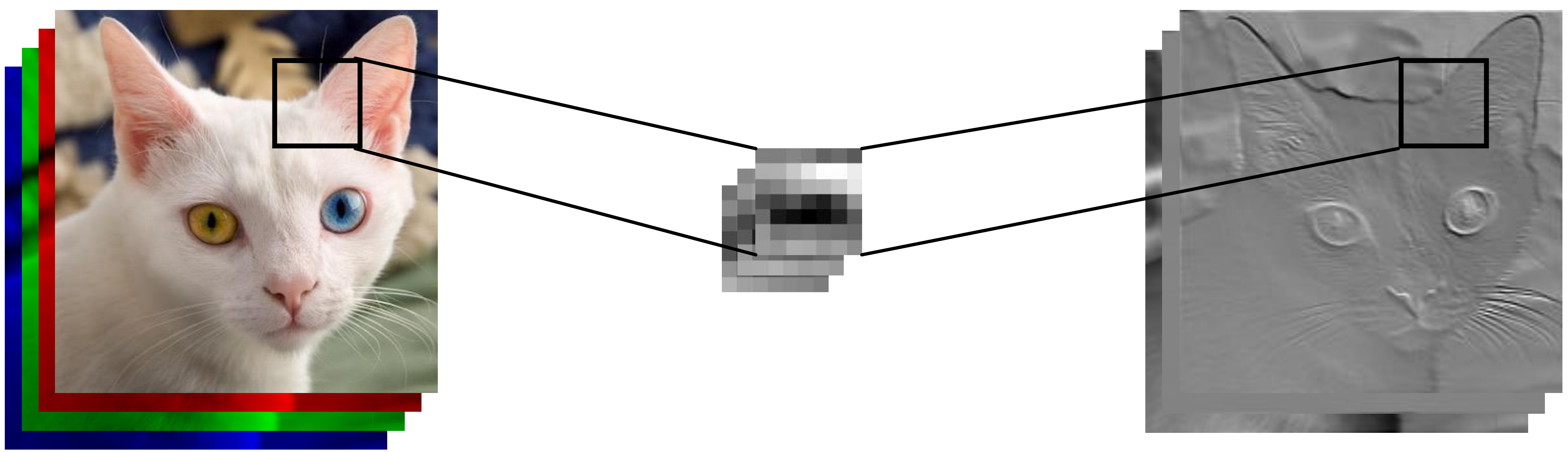
Convolution



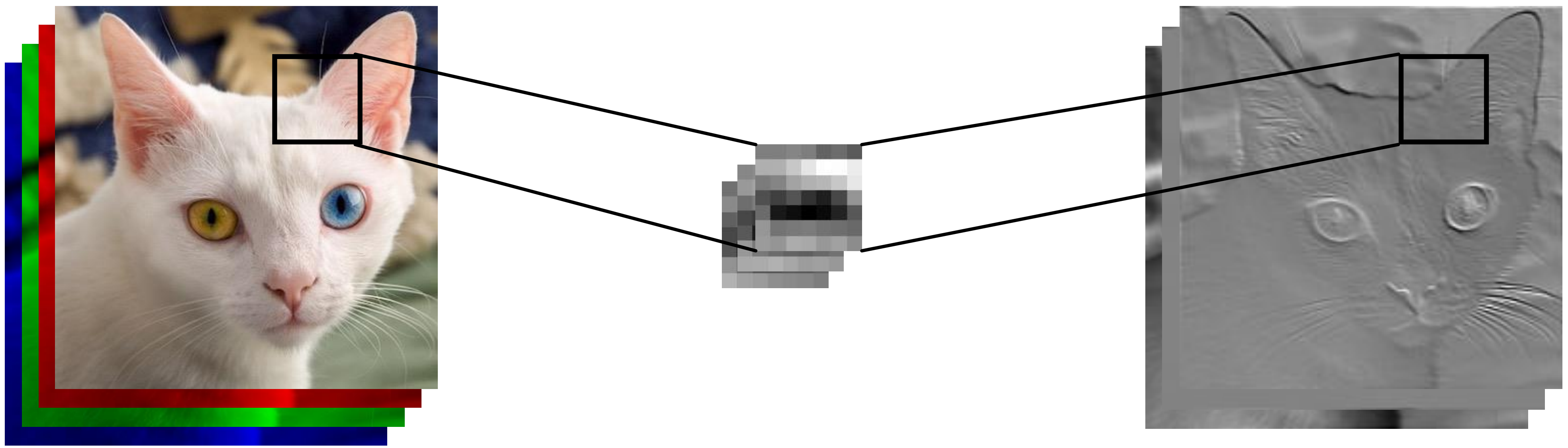
Convolution



Convolution

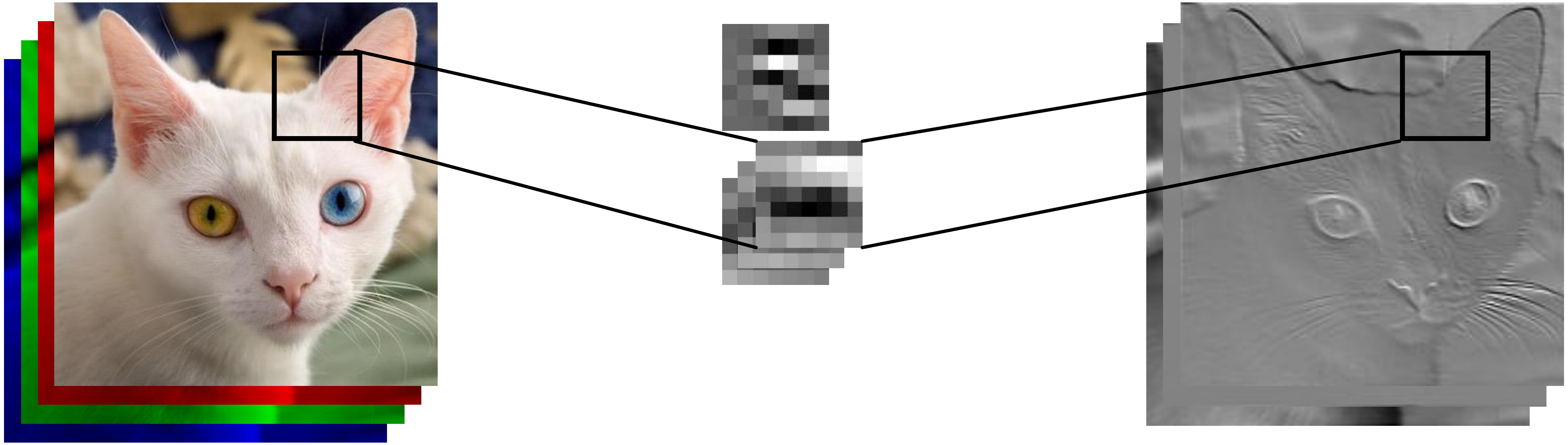


Convolution



Convolution

Inputs | Filters | Feature maps (activations)

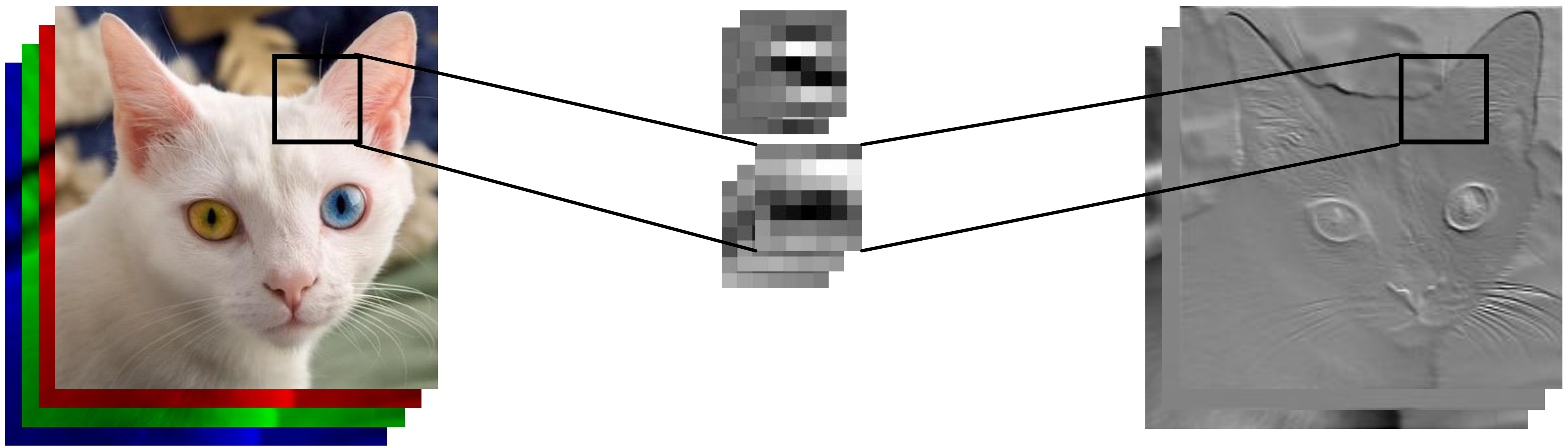


Convolution

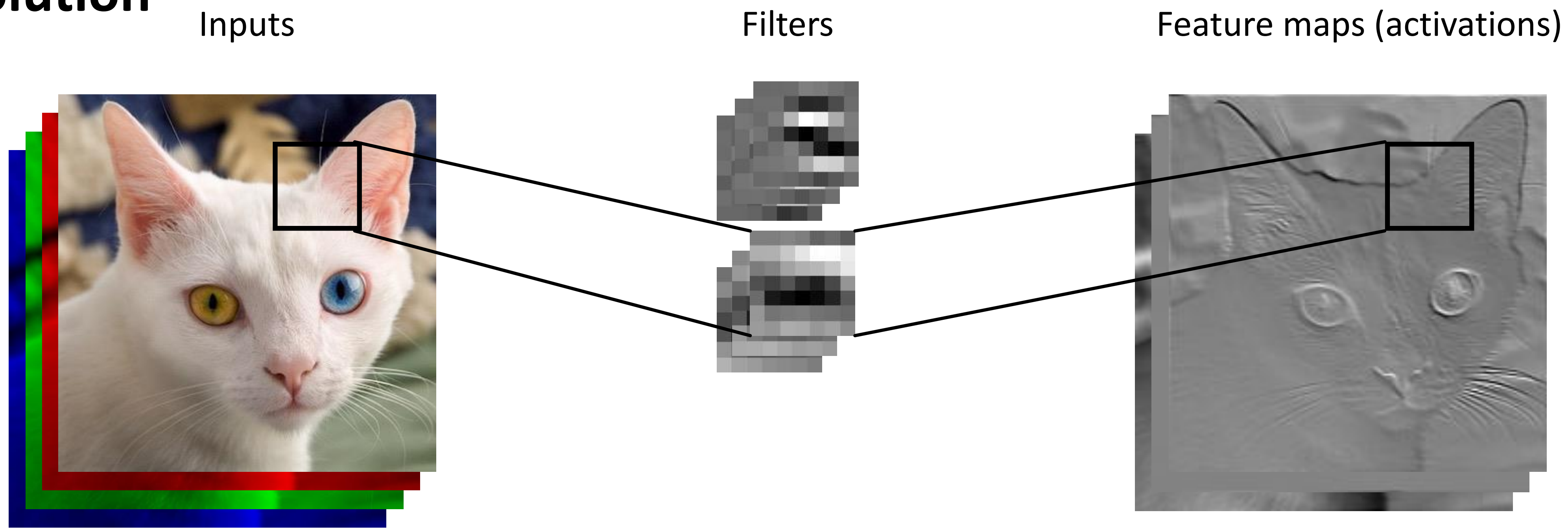
Inputs

Filters

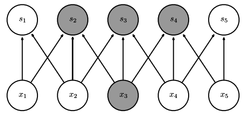
Feature maps (activations)

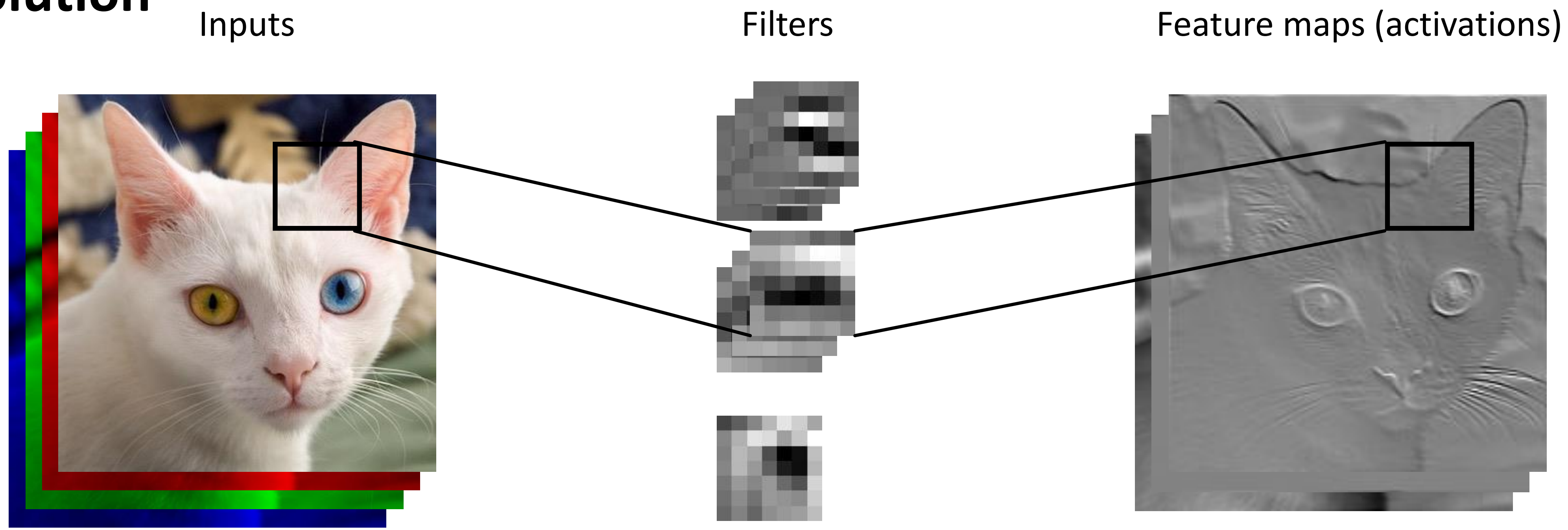


Convolution

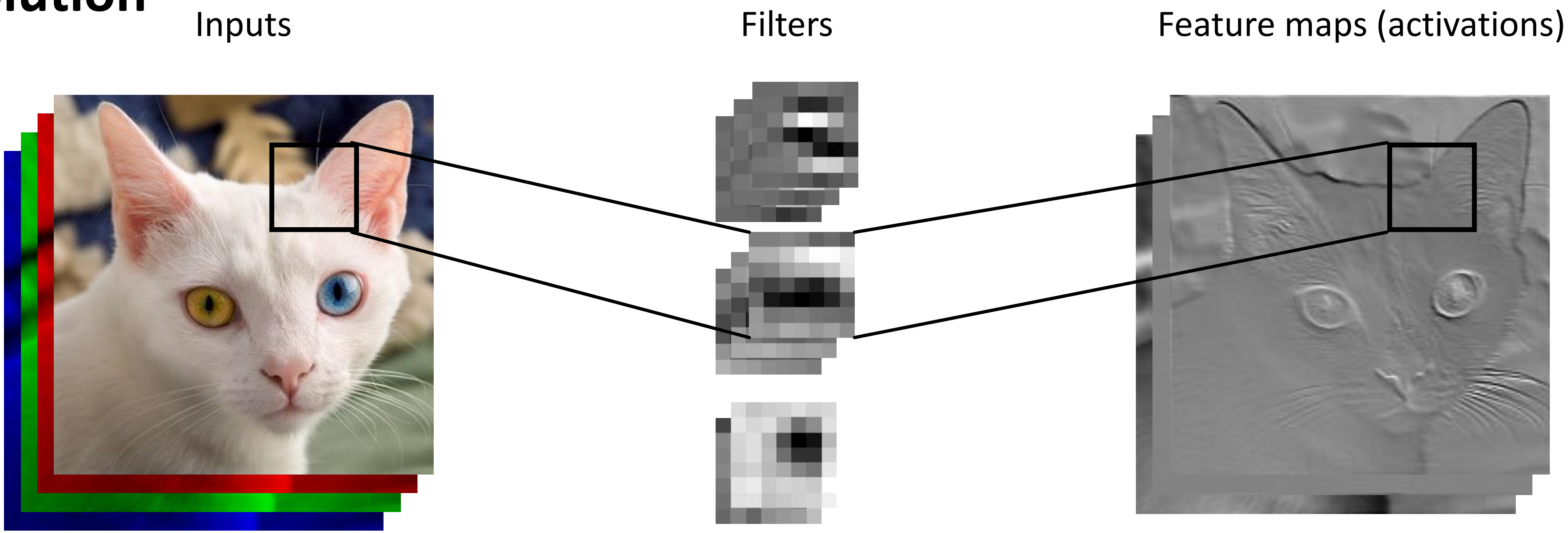


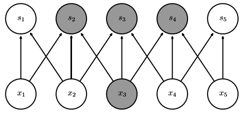
Convolution



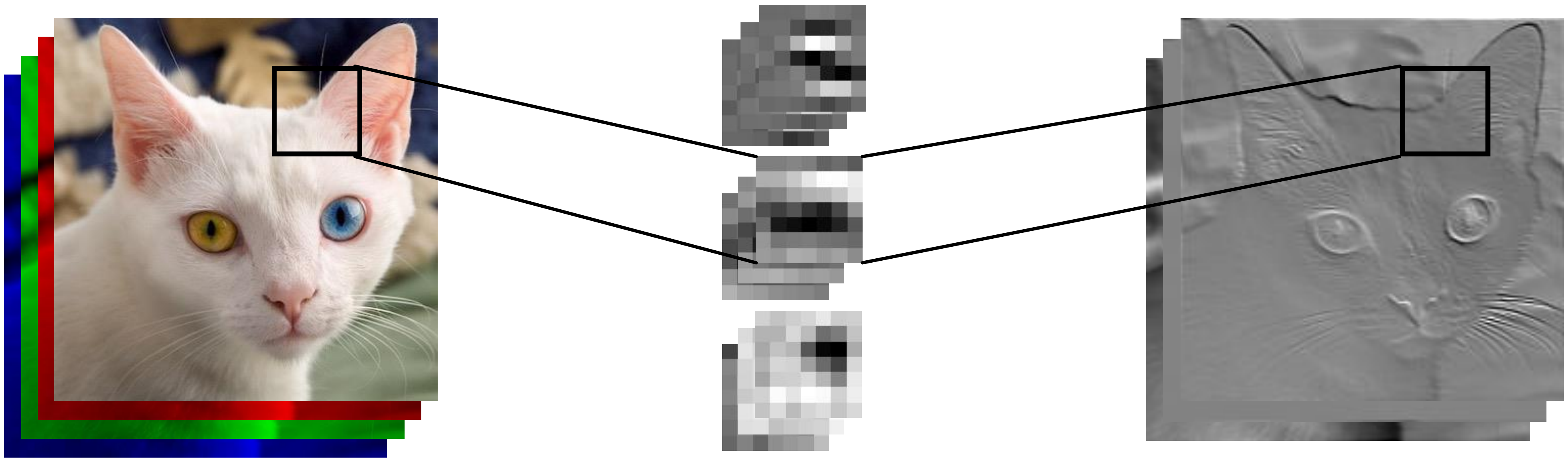


Convolution





Convolution

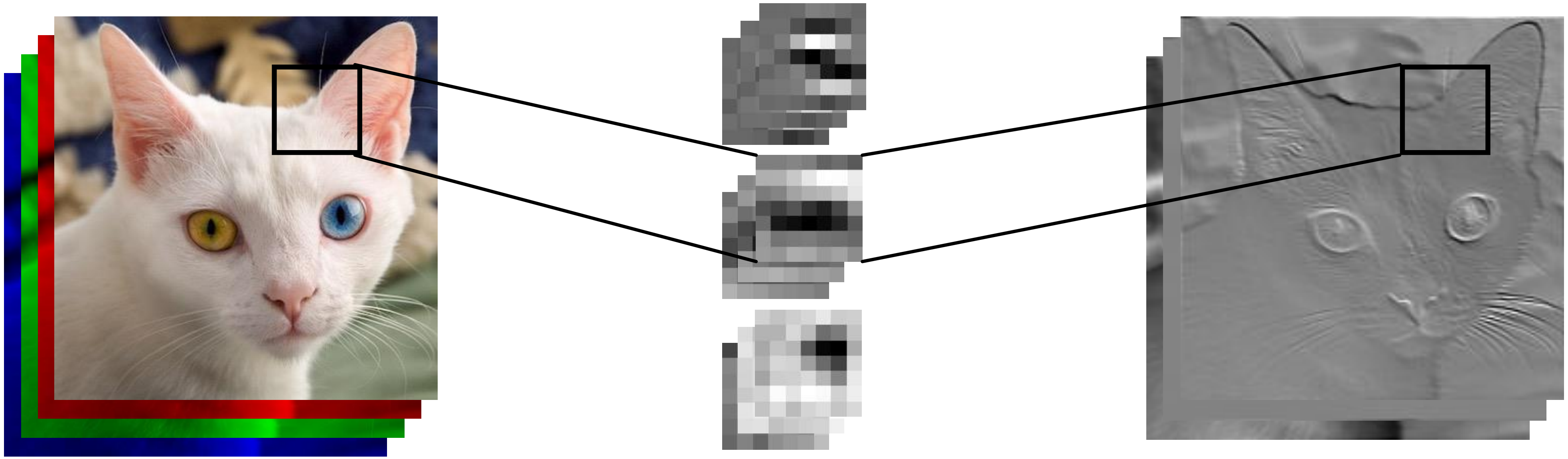


Convolution

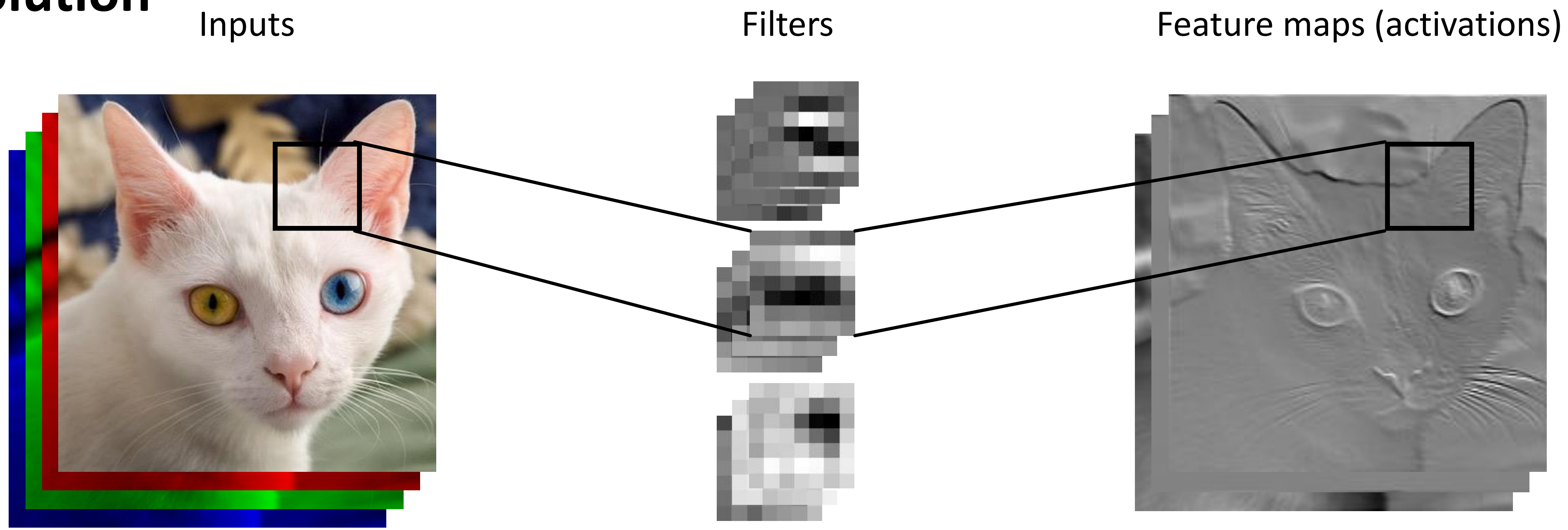
Inputs

Filters

Feature maps (activations)



Convolution



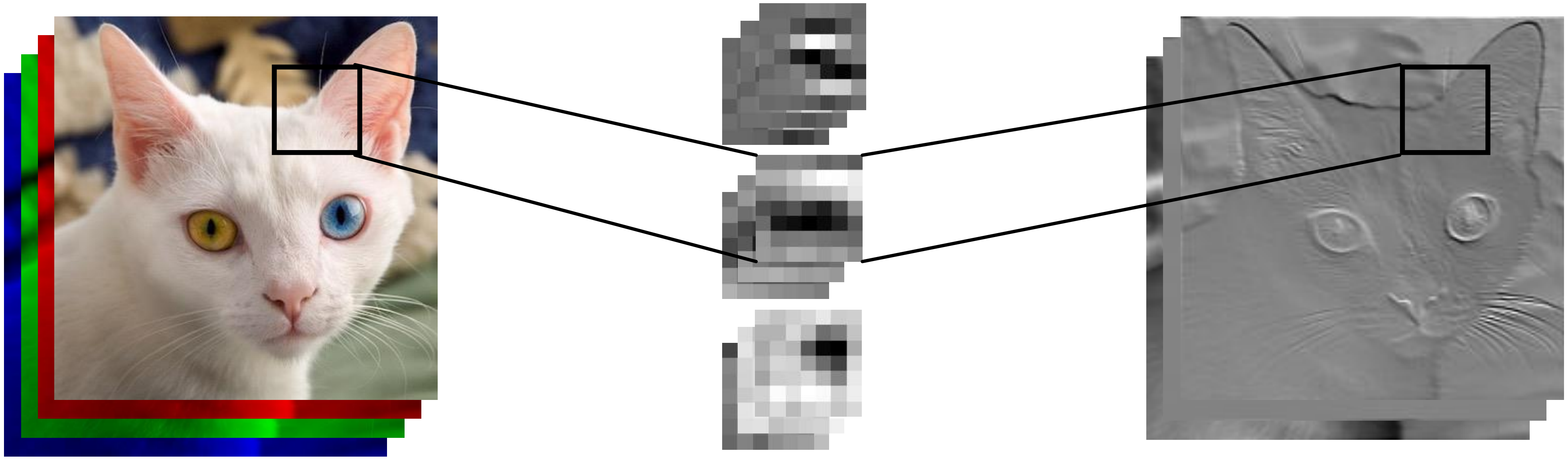
Convolution

Inputs

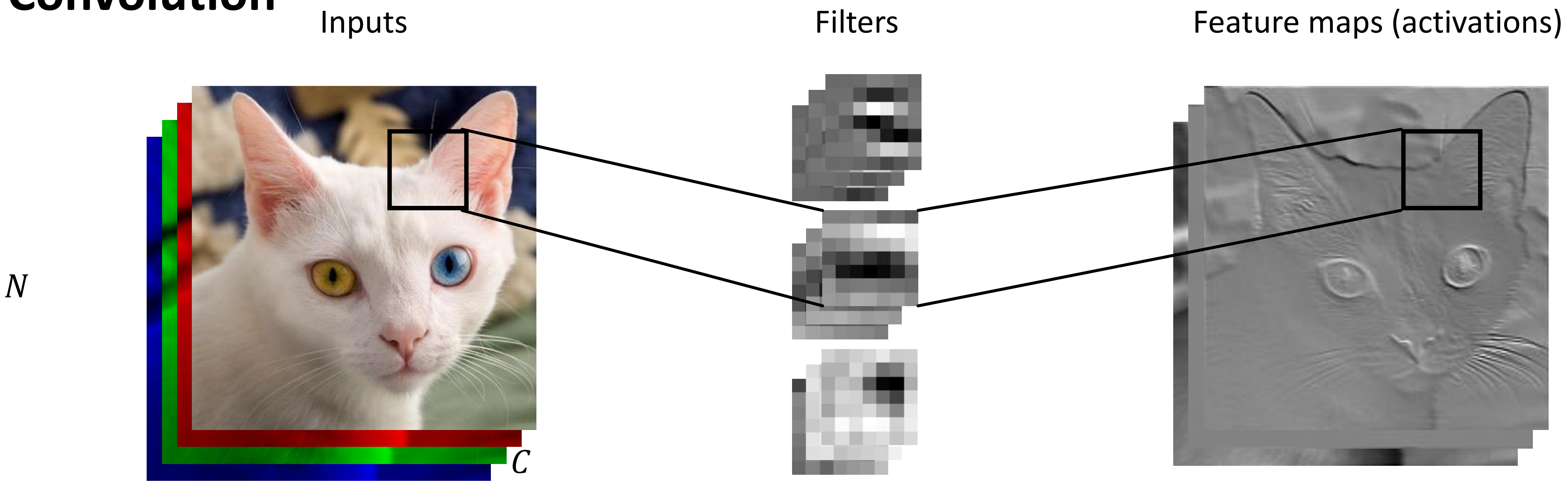
Filters

Feature maps (activations)

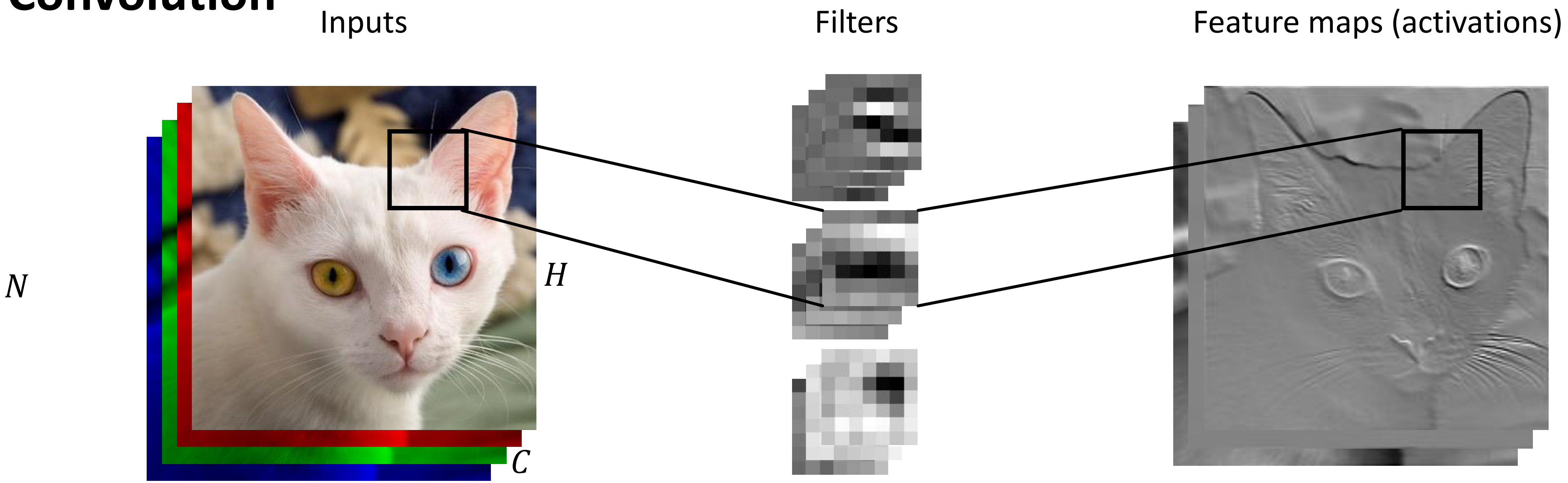
N



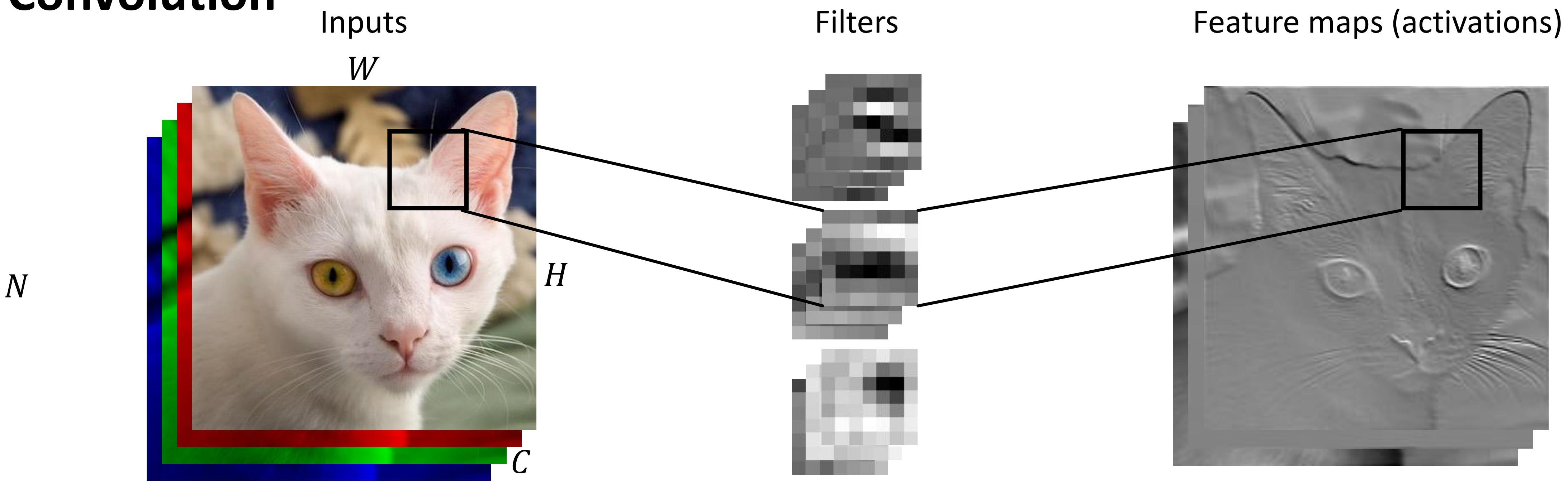
Convolution



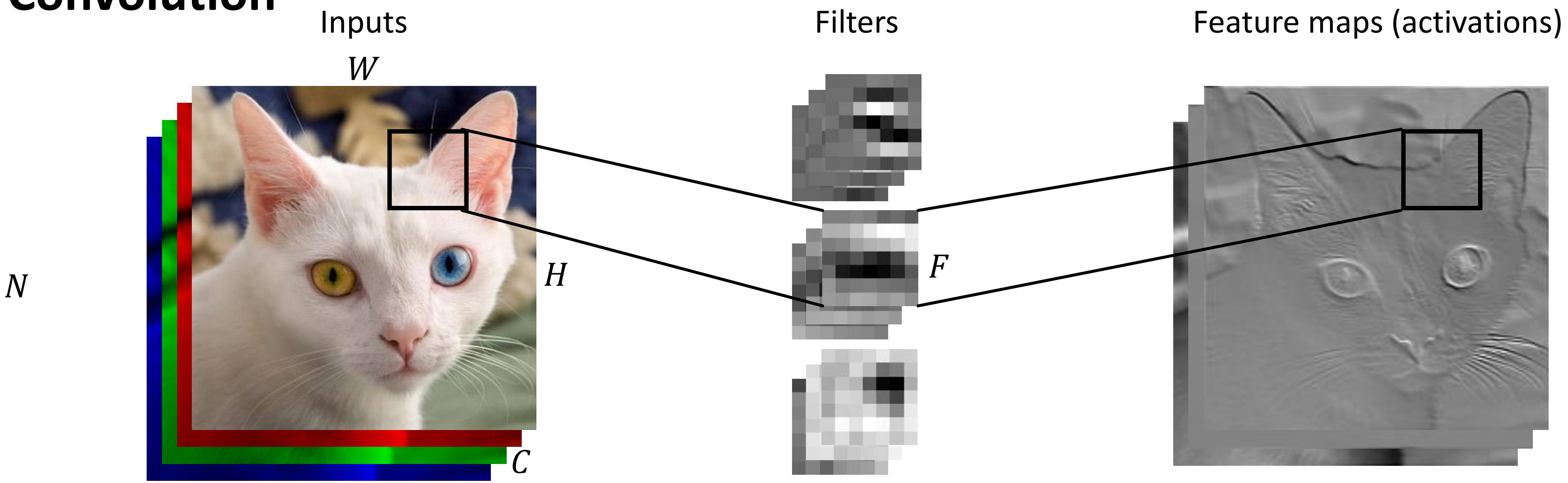
Convolution



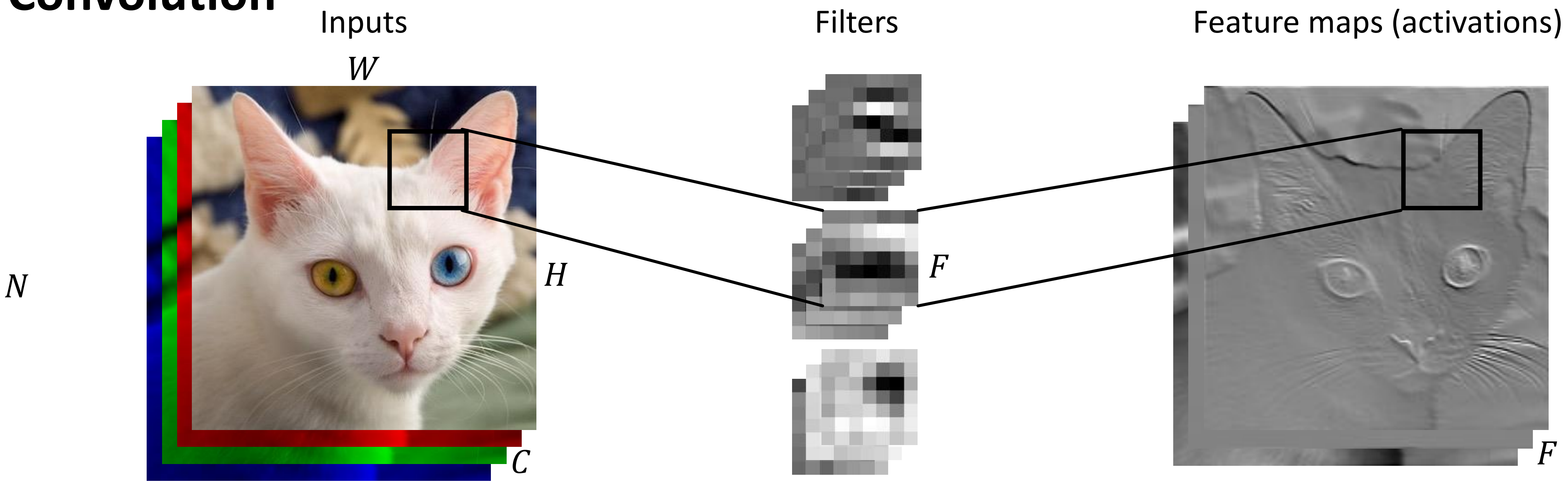
Convolution



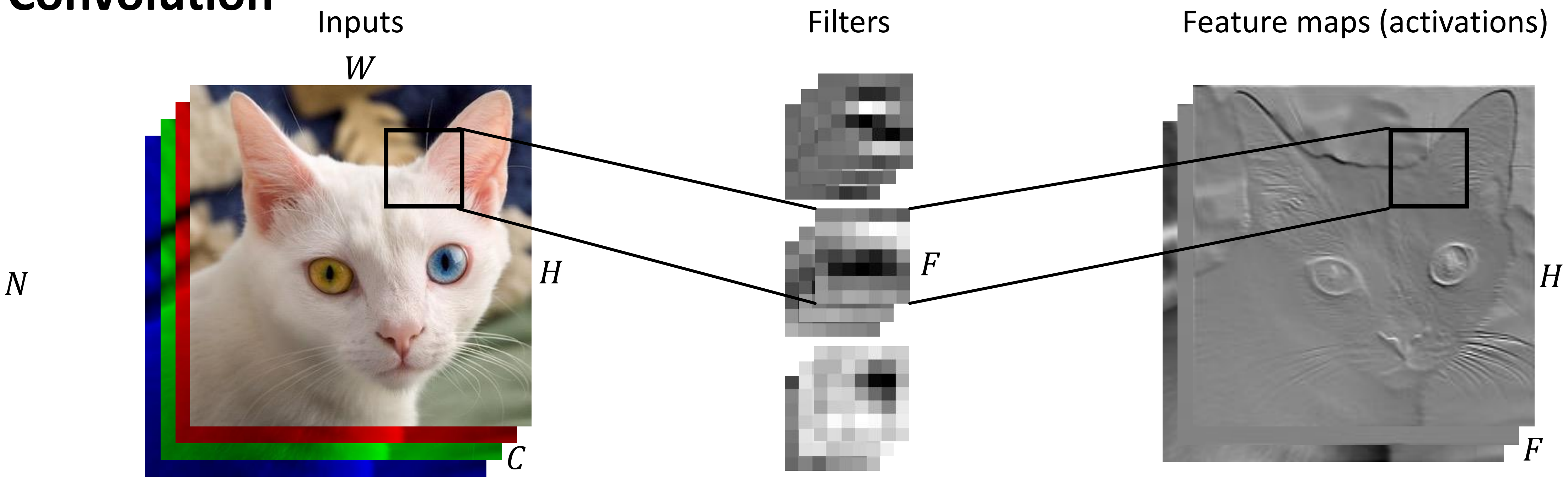
Convolution

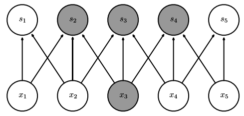


Convolution

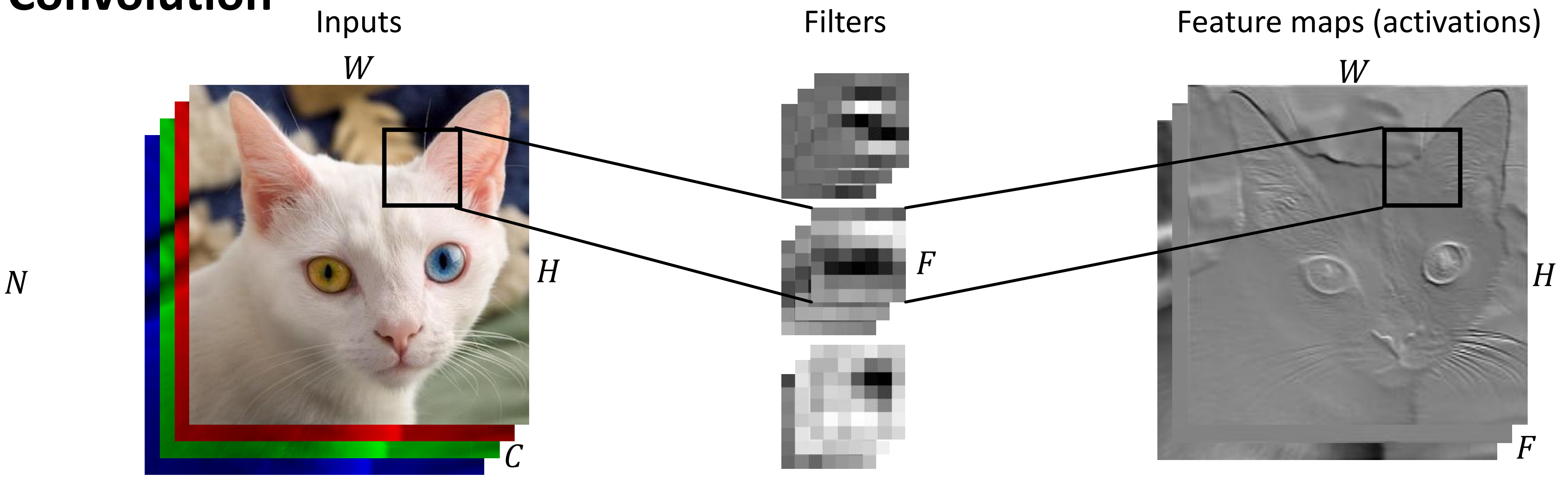


Convolution

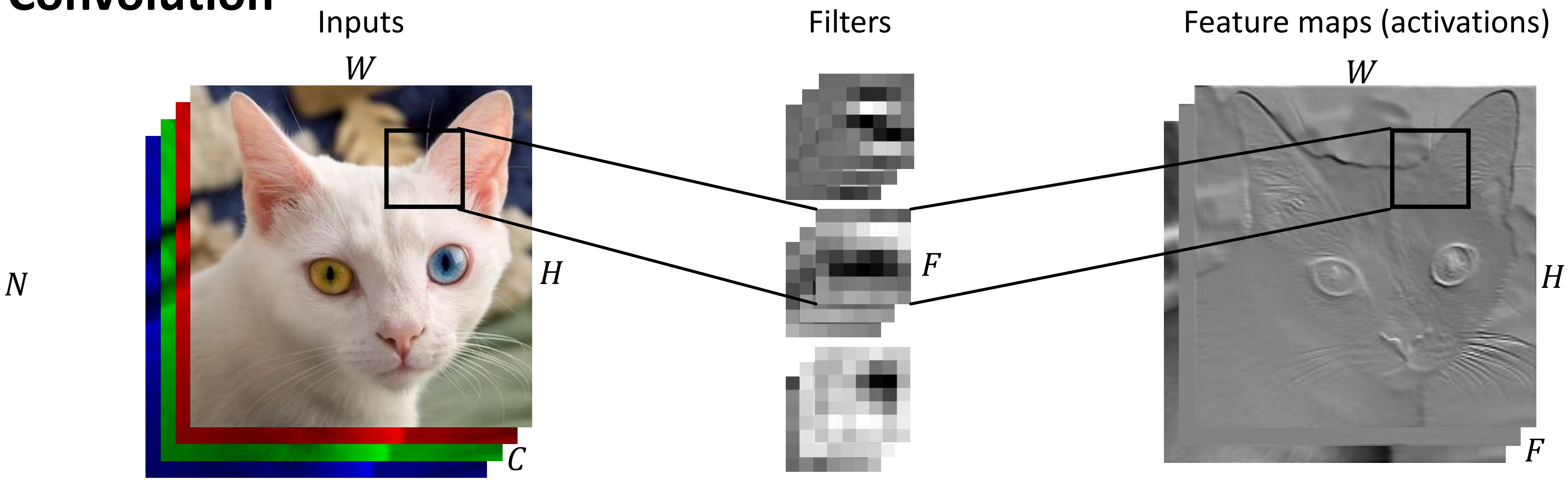


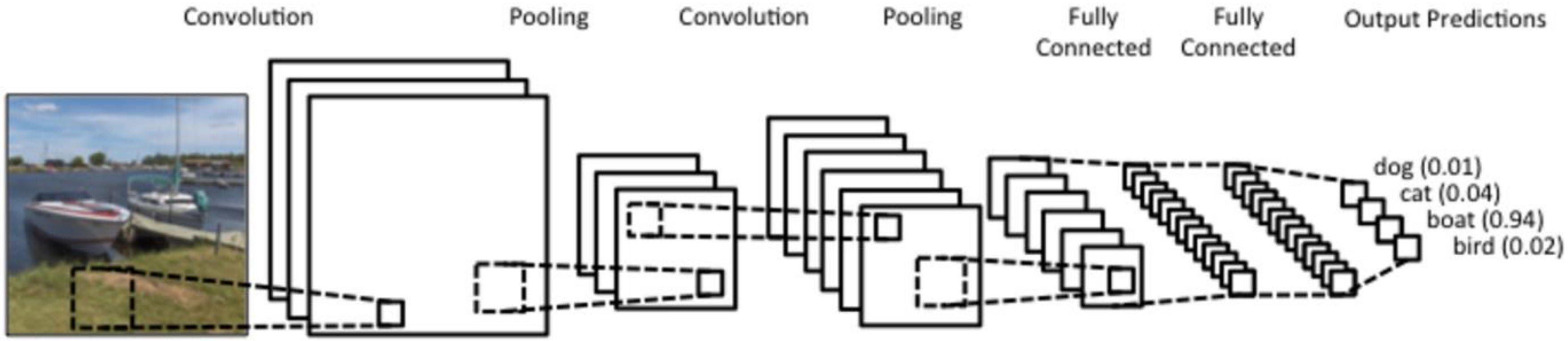


Convolution

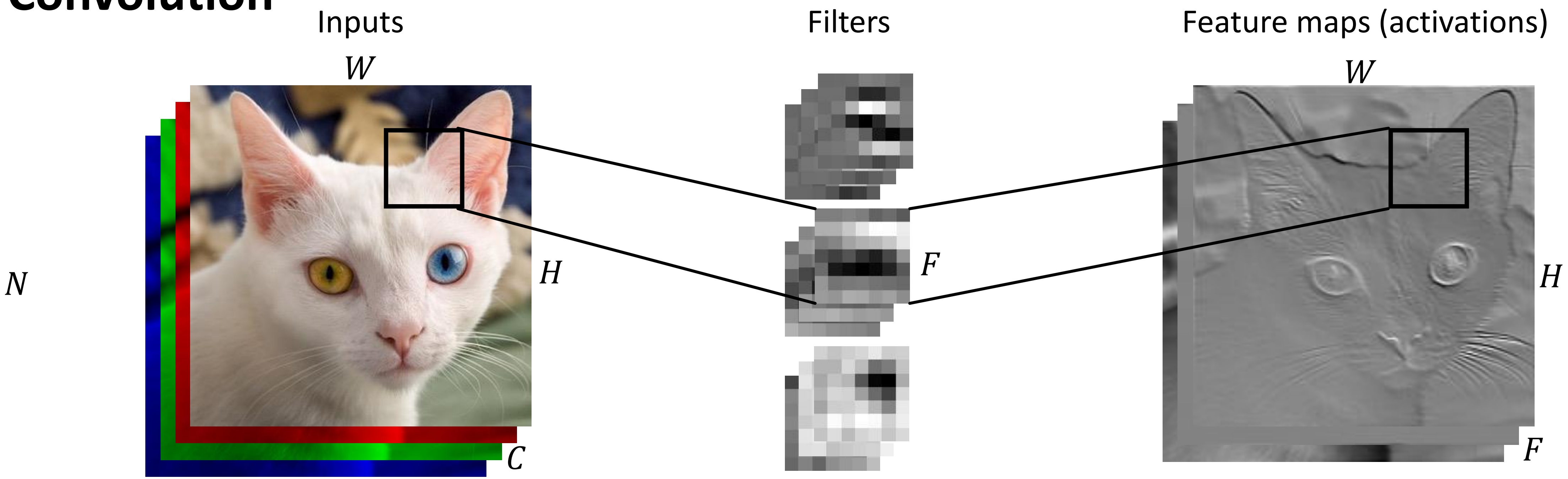


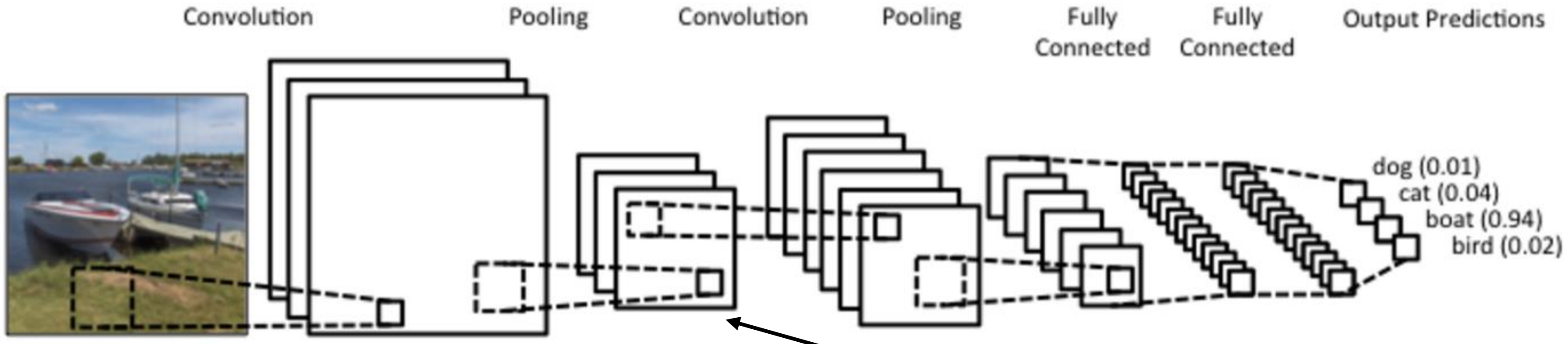
Convolution





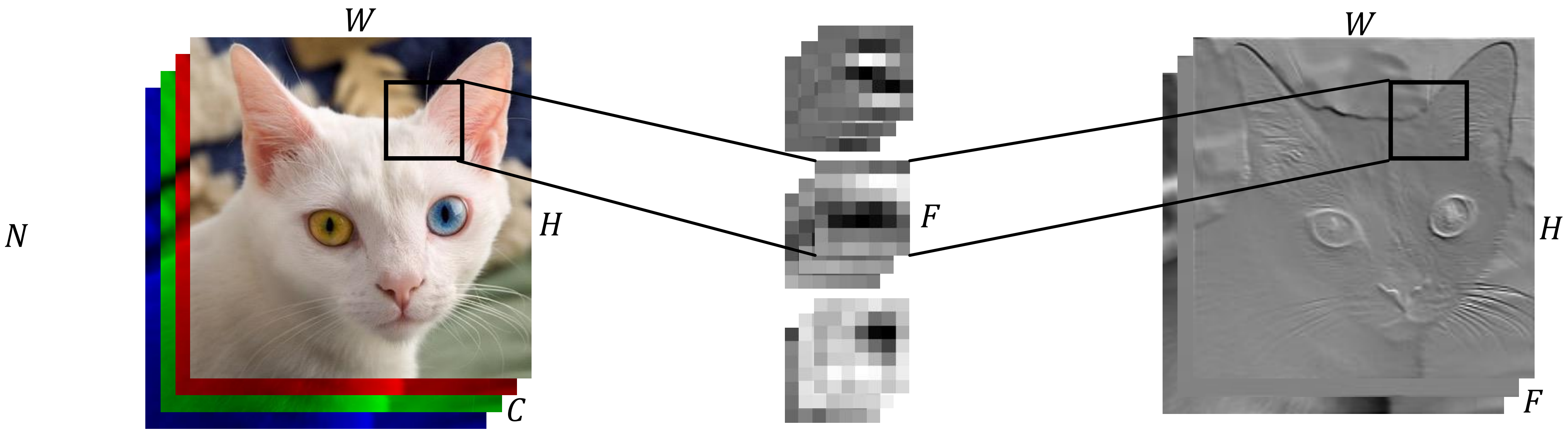
Convolution

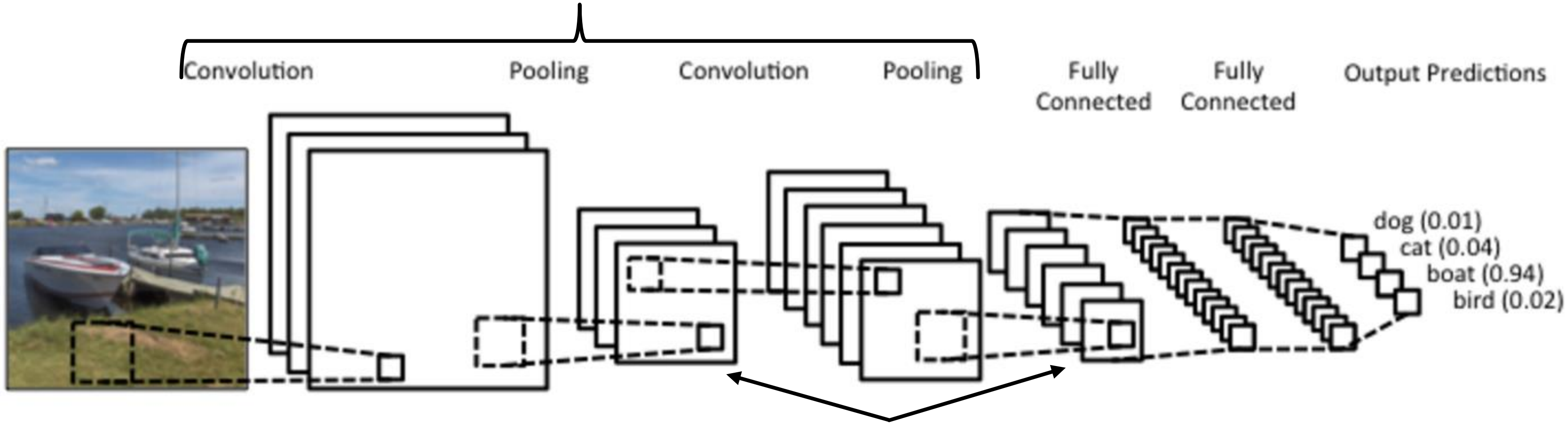




Translation invariance

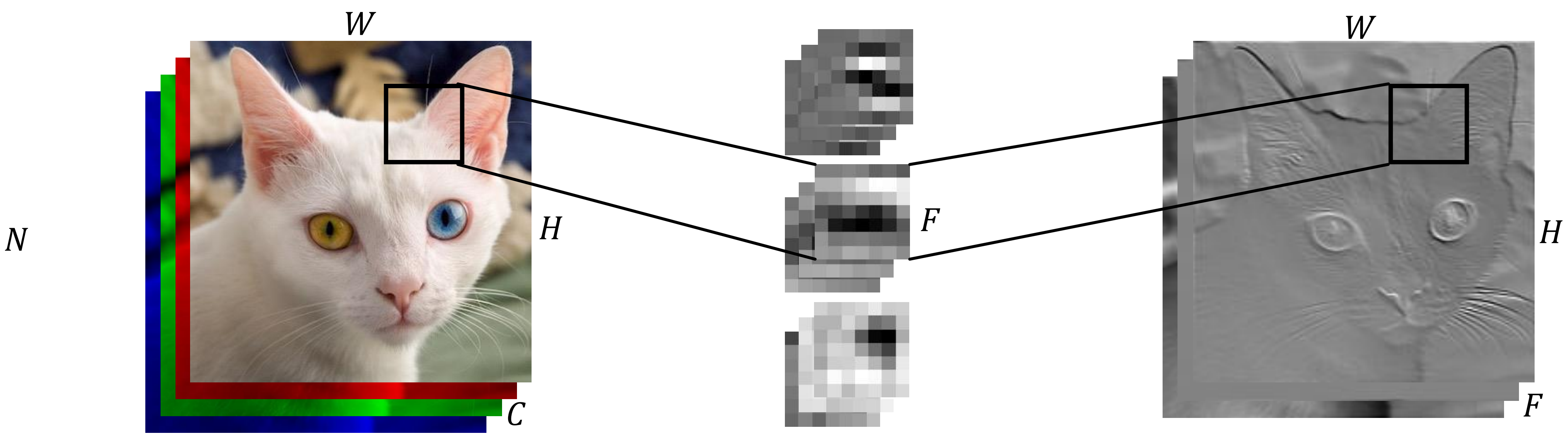
Convolution



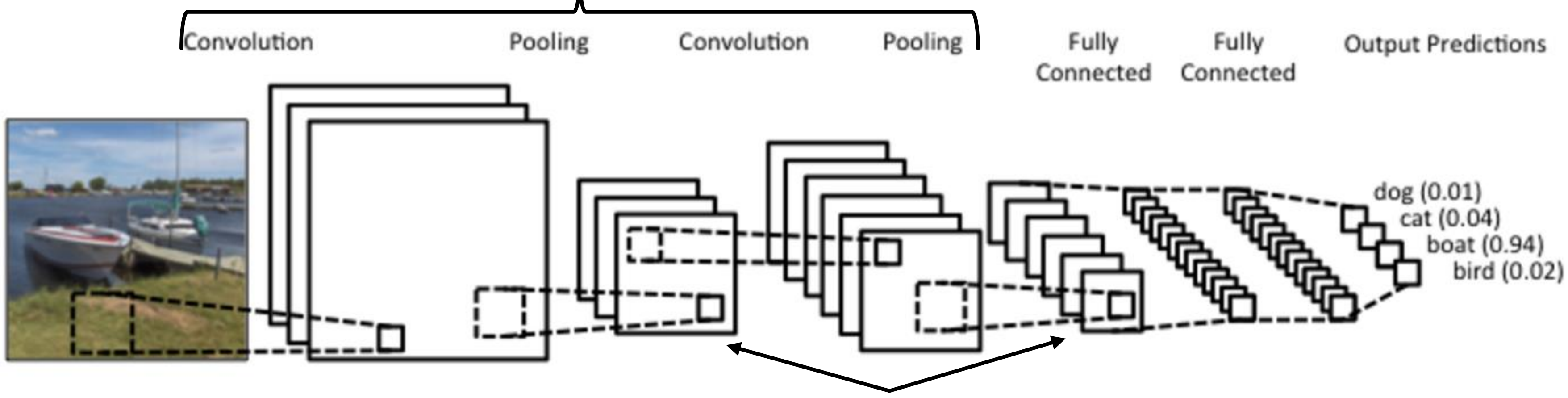


Translation invariance

Convolution

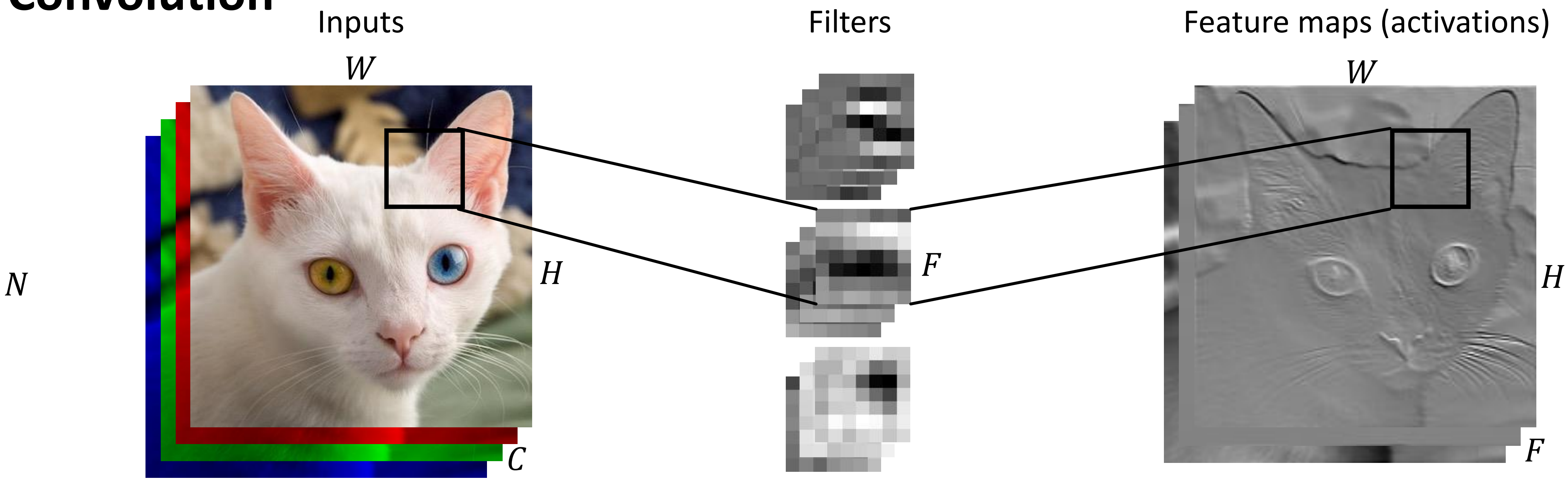


Feature learning

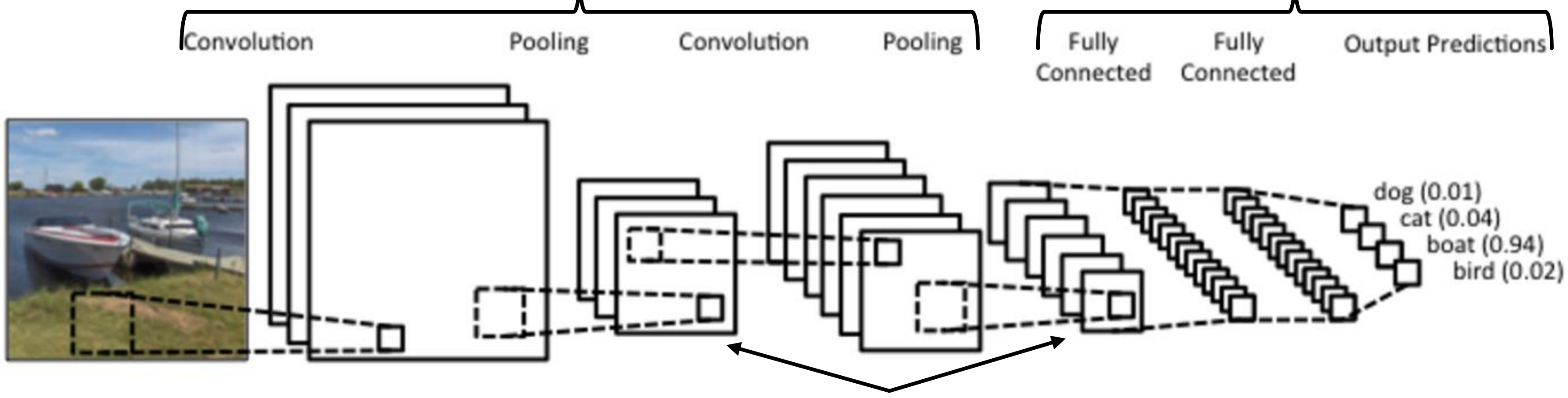


Translation invariance

Convolution

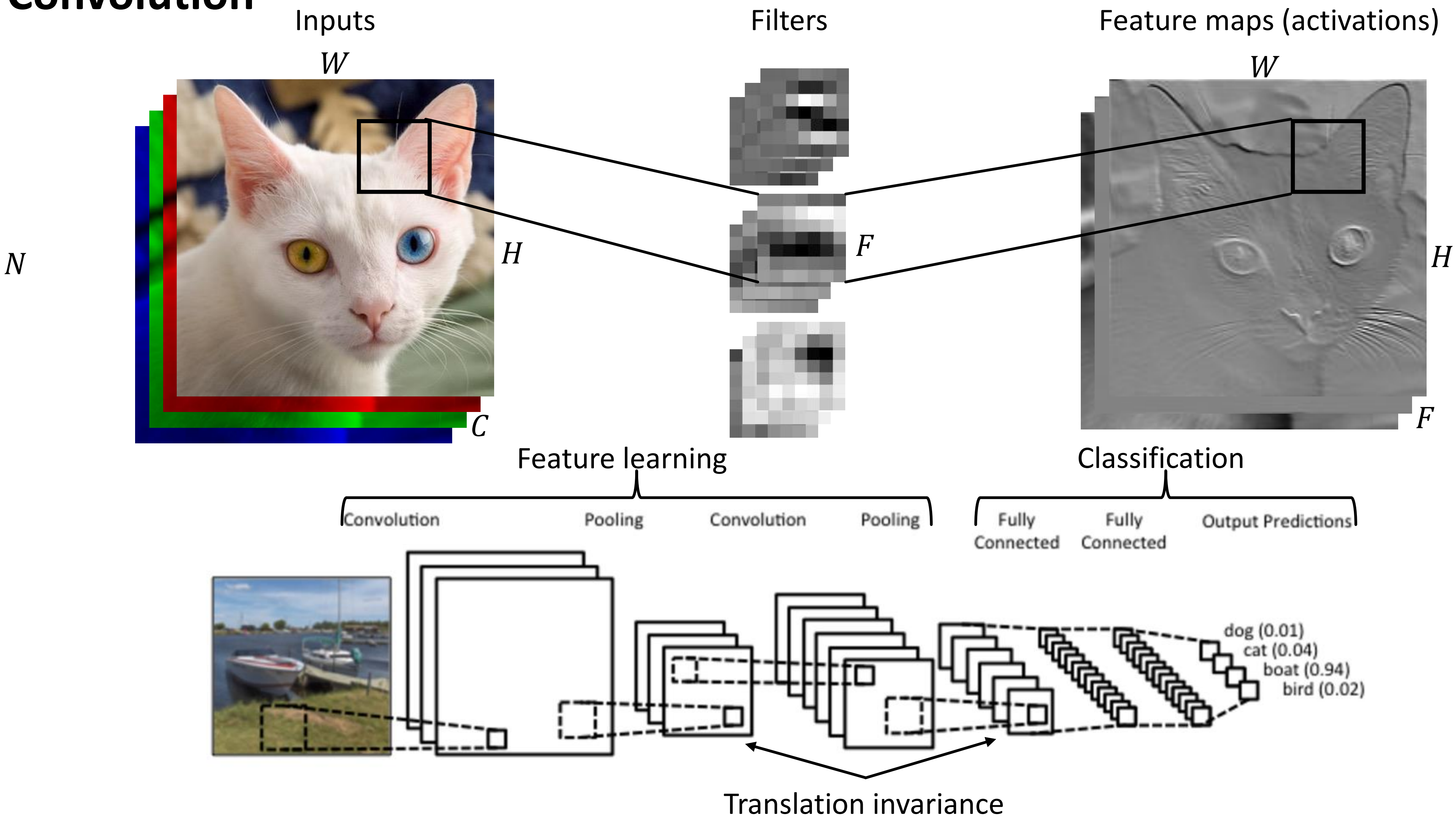


Feature learning

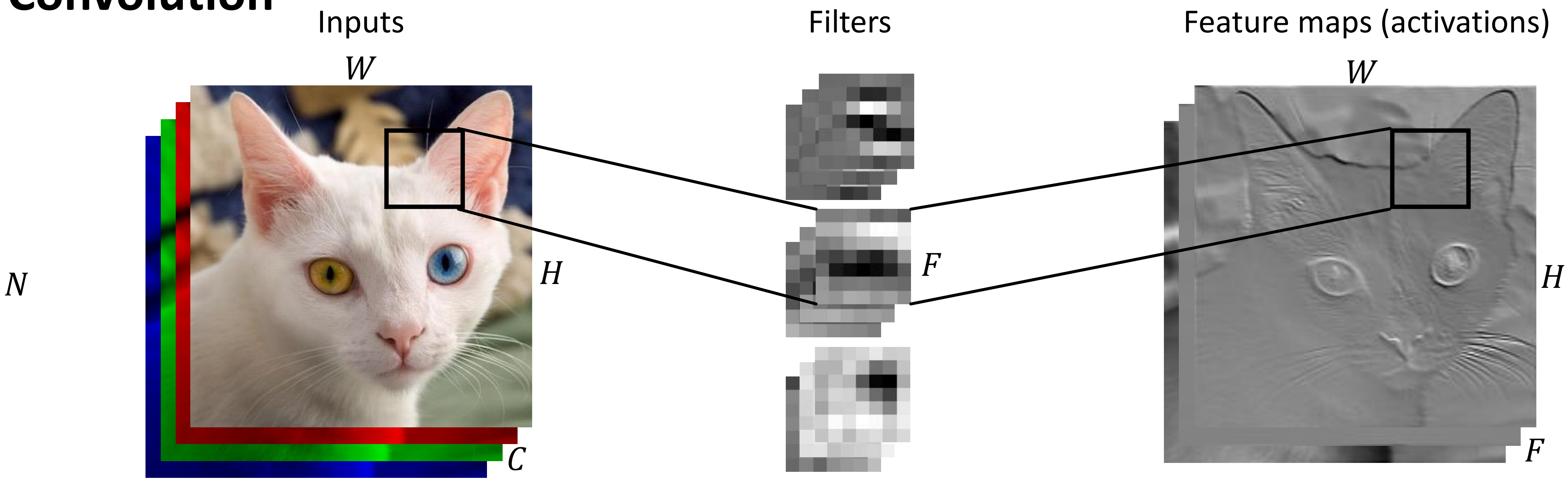


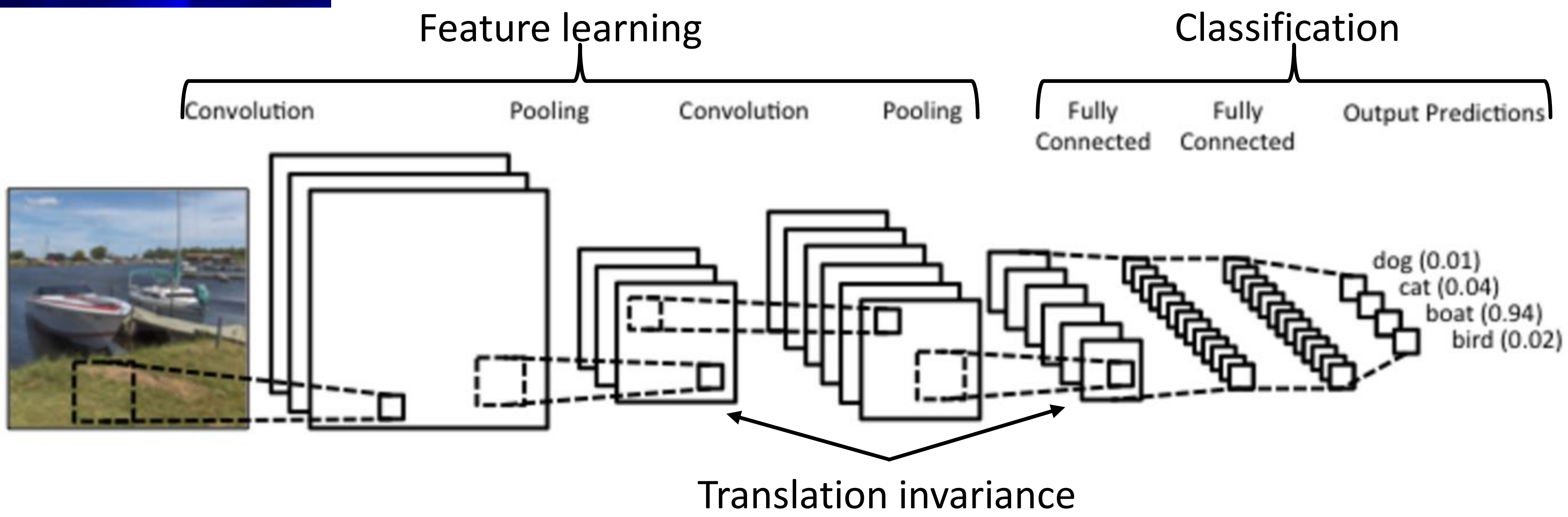
Translation invariance

Convolution

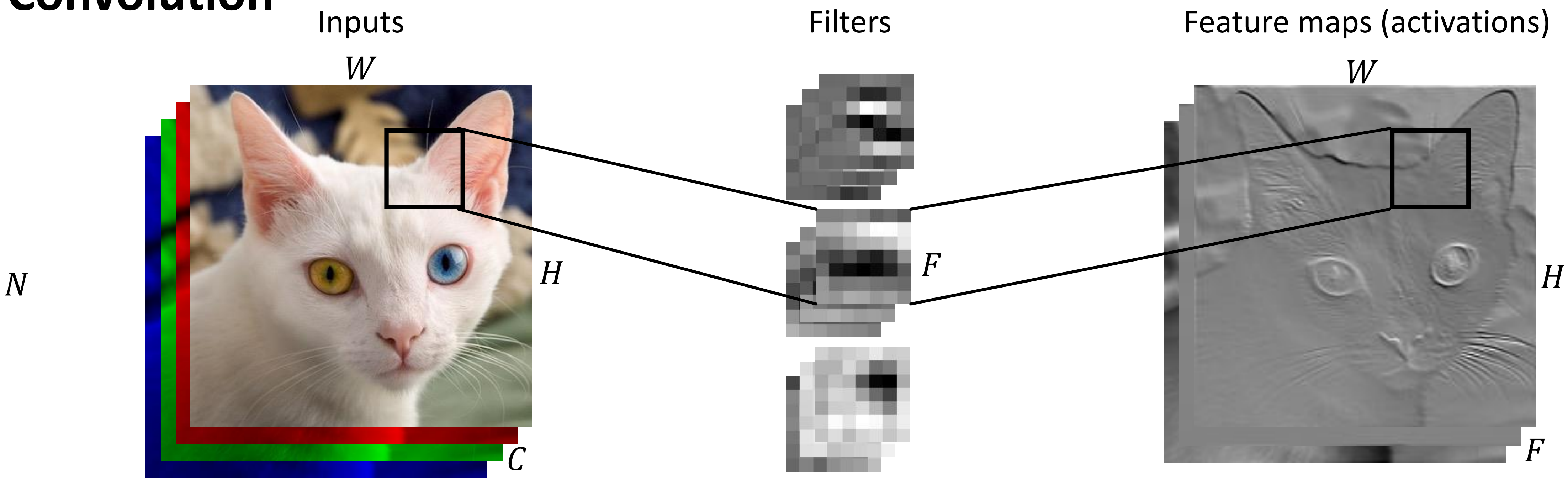


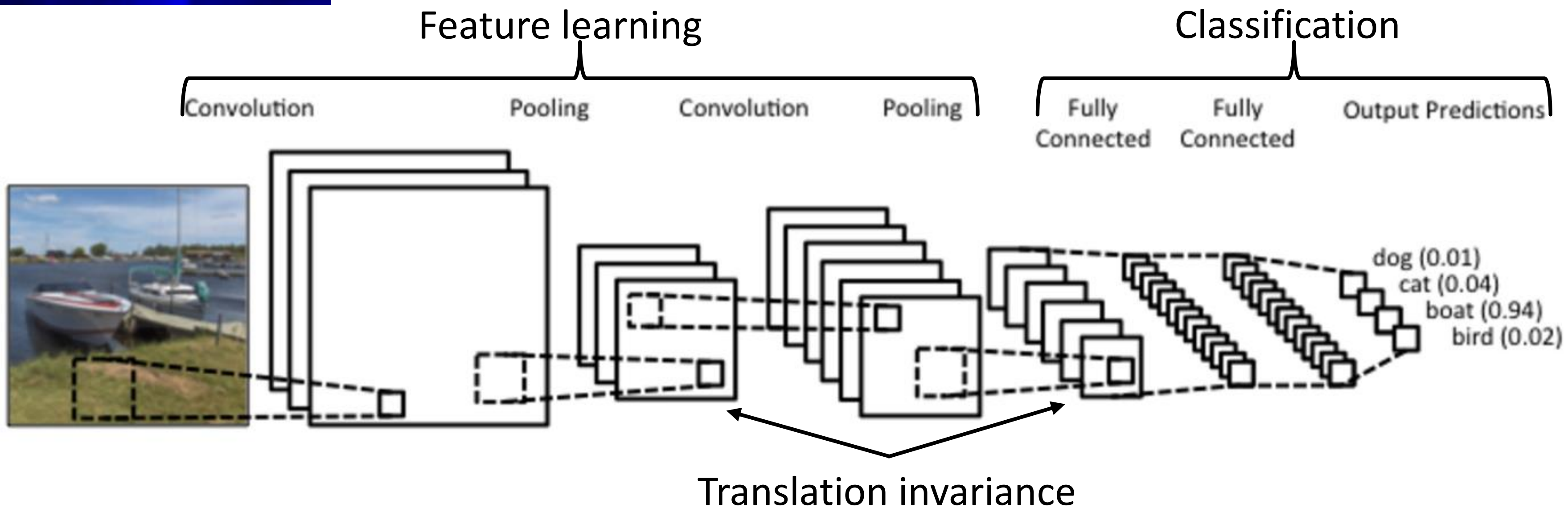
Convolution



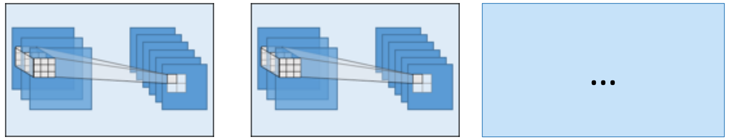


Convolution





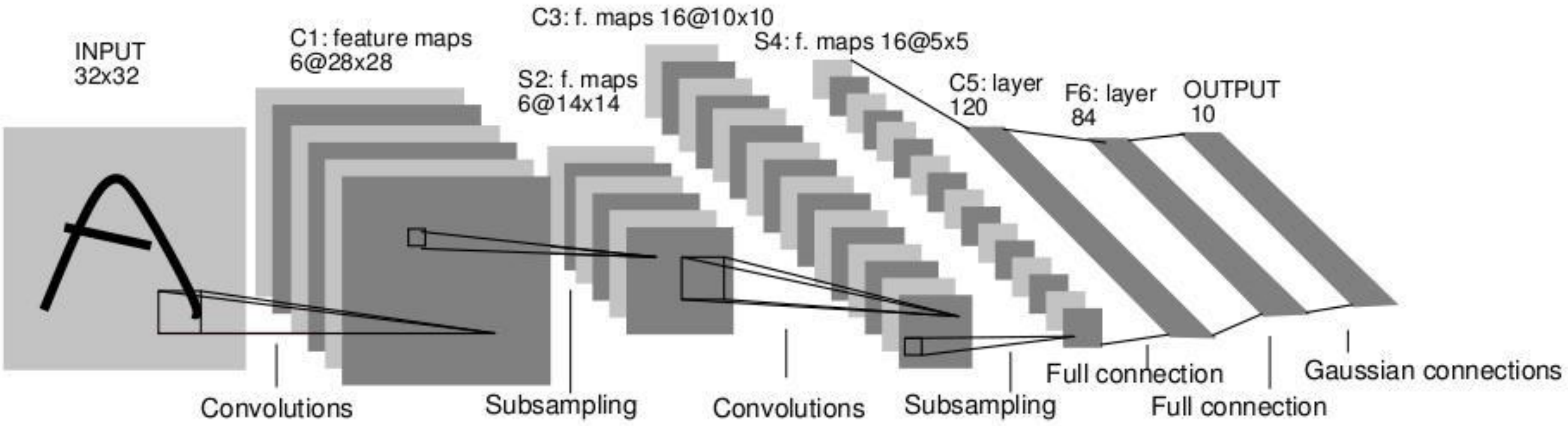
Operators



Operators

A short history of (old) CNNs

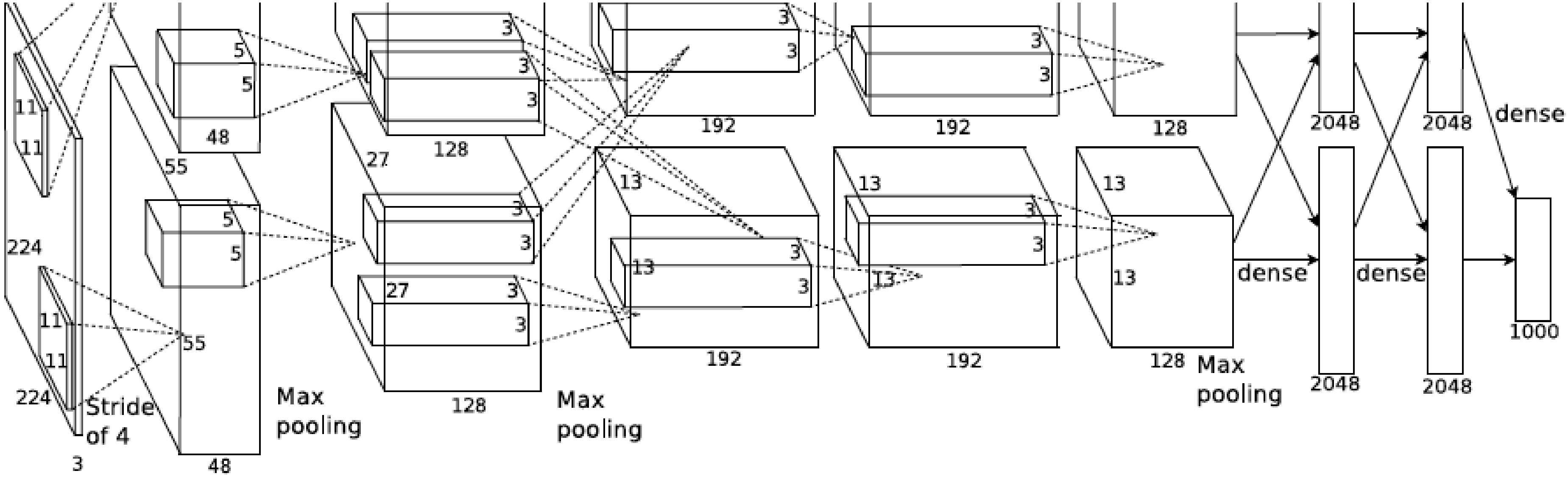
LeNet-5 [LeCun, Bottou, Bengio, & Haffner 1998]



- Average pooling
- Sigmoid/tanh nonlinearites
- Fully-connected layers at end
- Trained on MNIST (60k samples)

A short history of (old) CNNs

AlexNet [Krizhevsky, Sutskever, & Hinton 2012]



- Max pooling
- ReLU nonlinearities
- Deeper, bigger model
- Dropout
- Trained on ImageNet (1.2M images)
- GPU implementation (2 GPUs for a week)

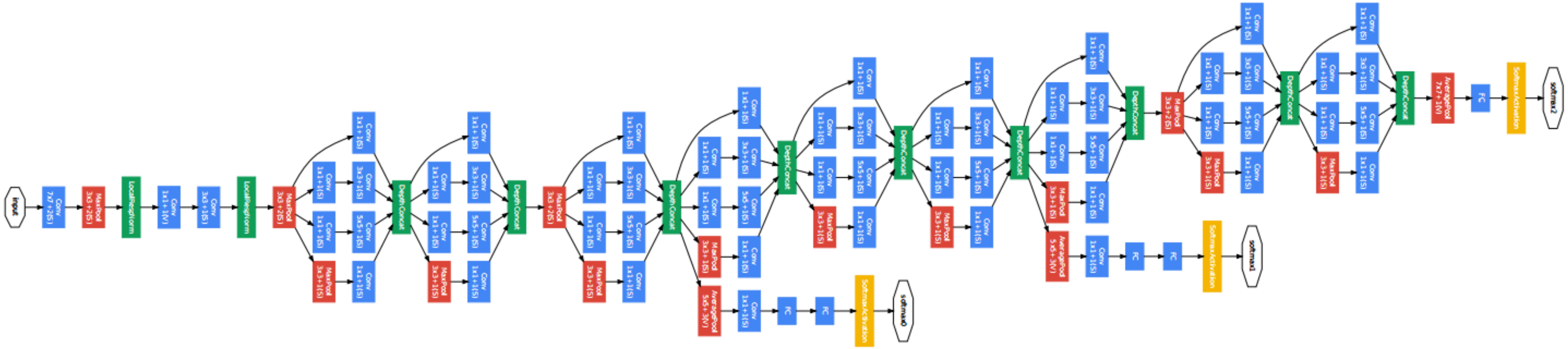
A short history of (old) CNNs

AlexNet [Krizhevsky, Sutskever, & Hinton 2012]

- Deeper, bigger model
- Dropout
- Trained on ImageNet (1.2M images)
- GPU implementation (2 GPUs for a week)

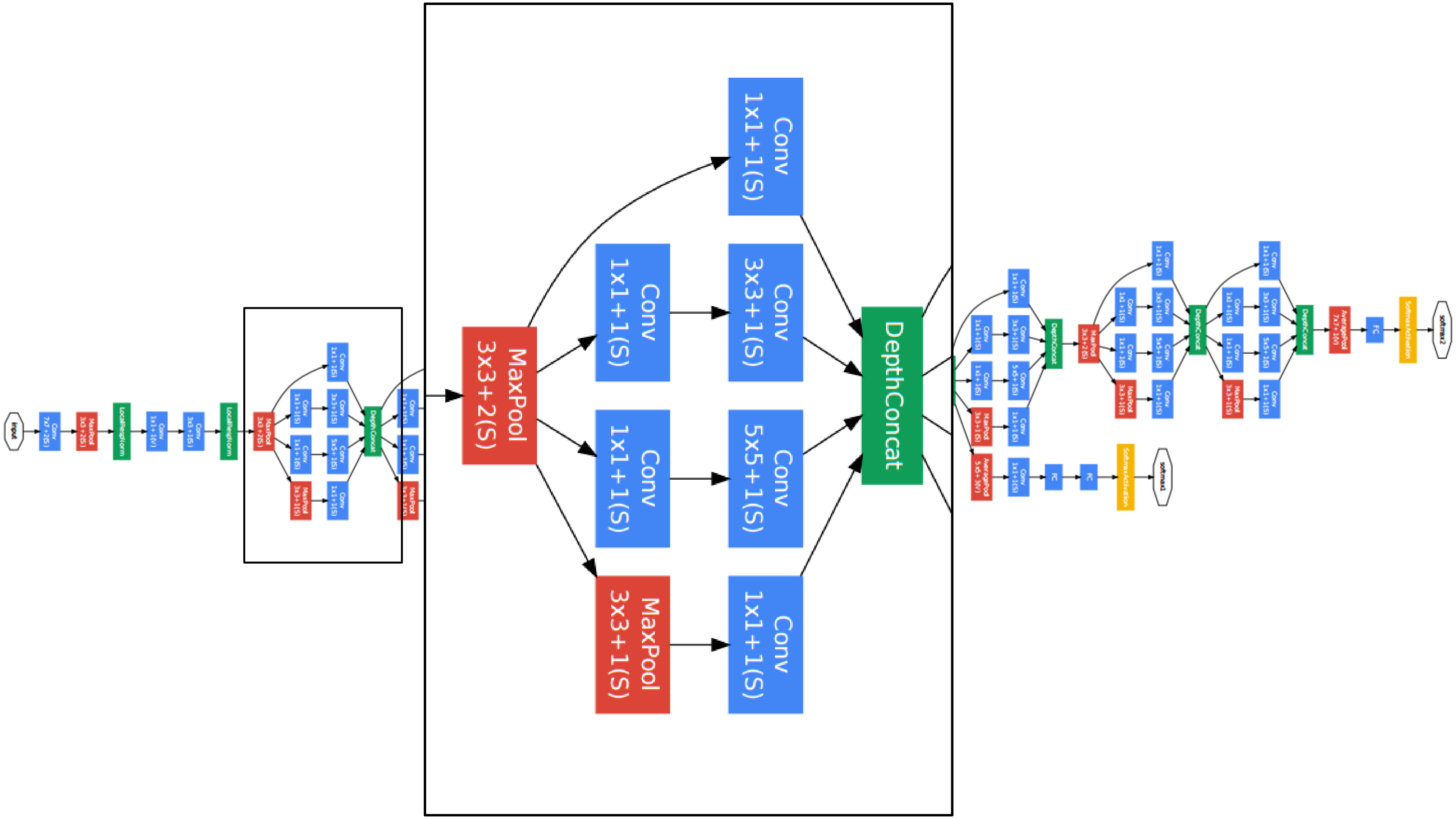
A short history of (old) CNNs

GoogLeNet (AKA Inception) [Szegedy et al. 2015]



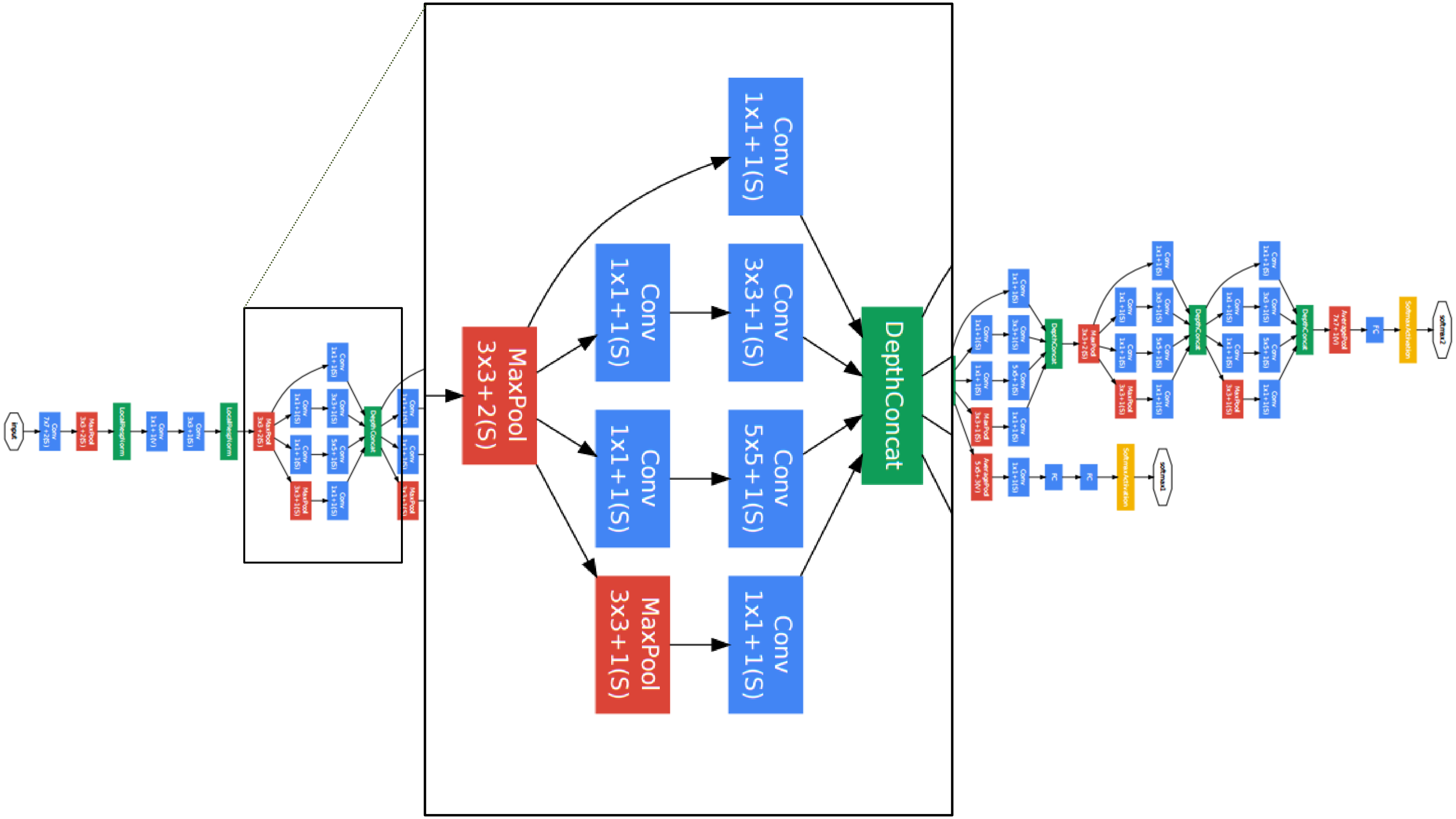
A short history of (old) CNNs

GoogLeNet (AKA Inception) [Szegedy et al. 2015]



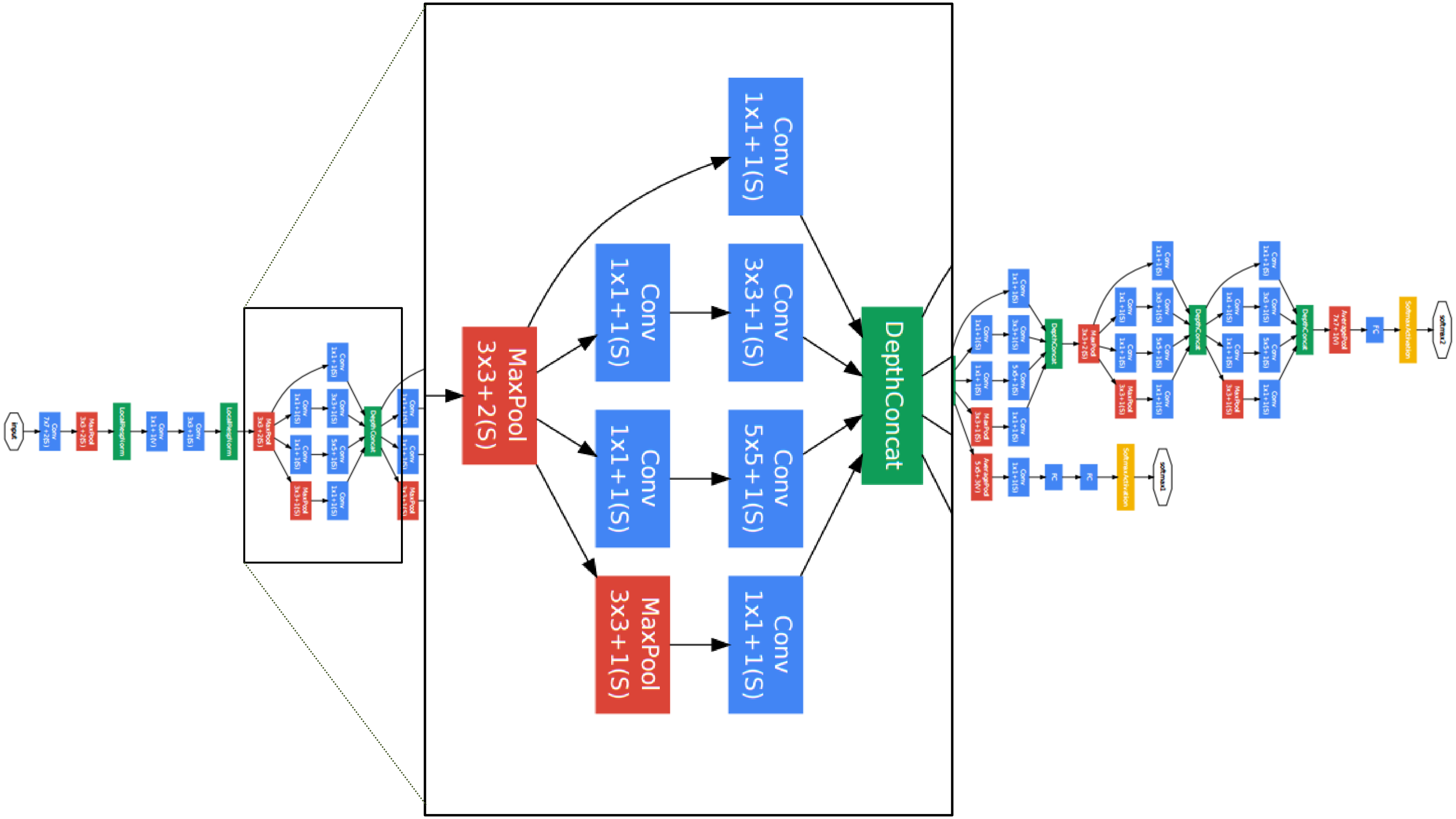
A short history of (old) CNNs

GoogLeNet (AKA Inception) [Szegedy et al. 2015]



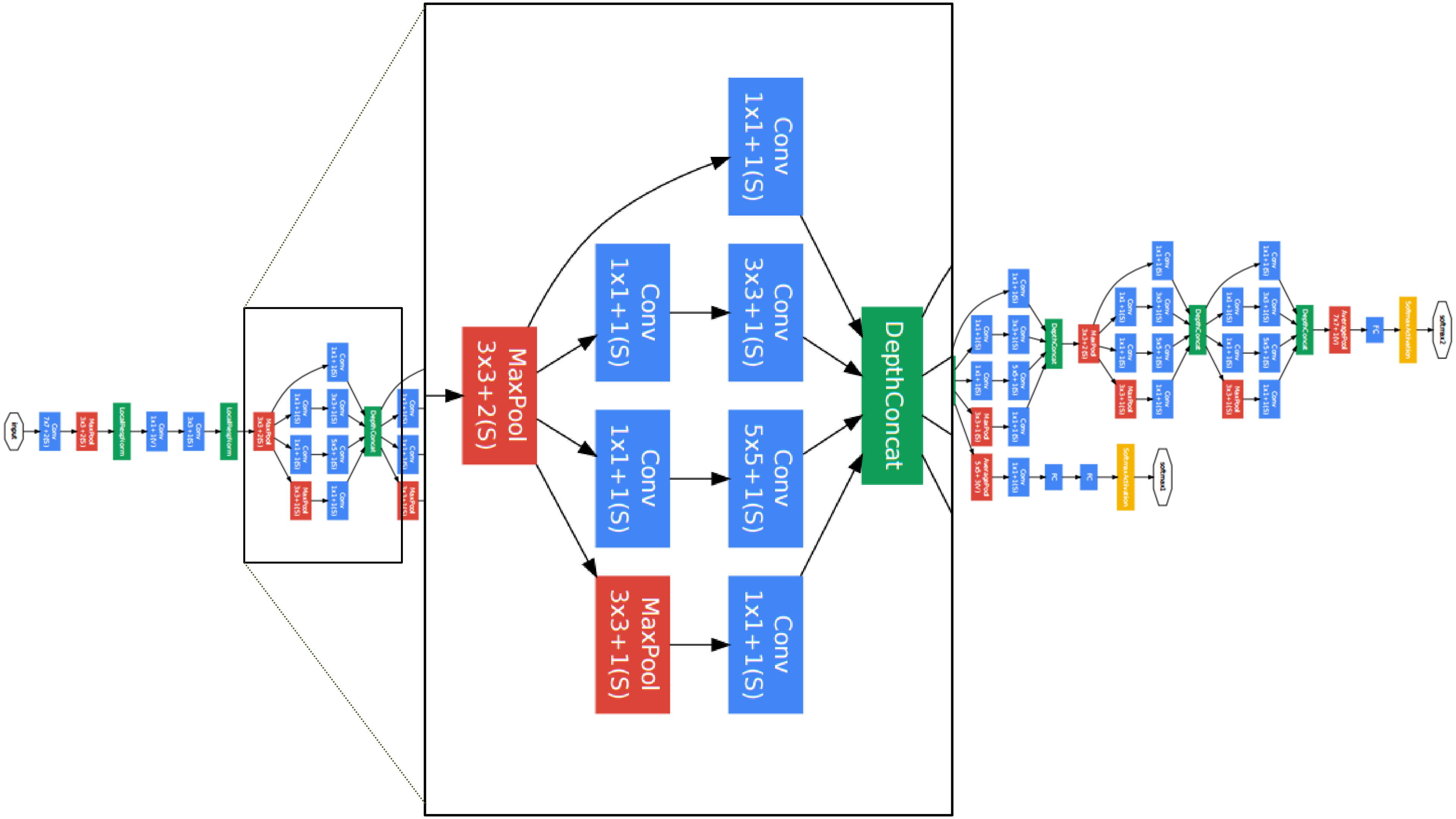
A short history of (old) CNNs

GoogLeNet (AKA Inception) [Szegedy et al. 2015]



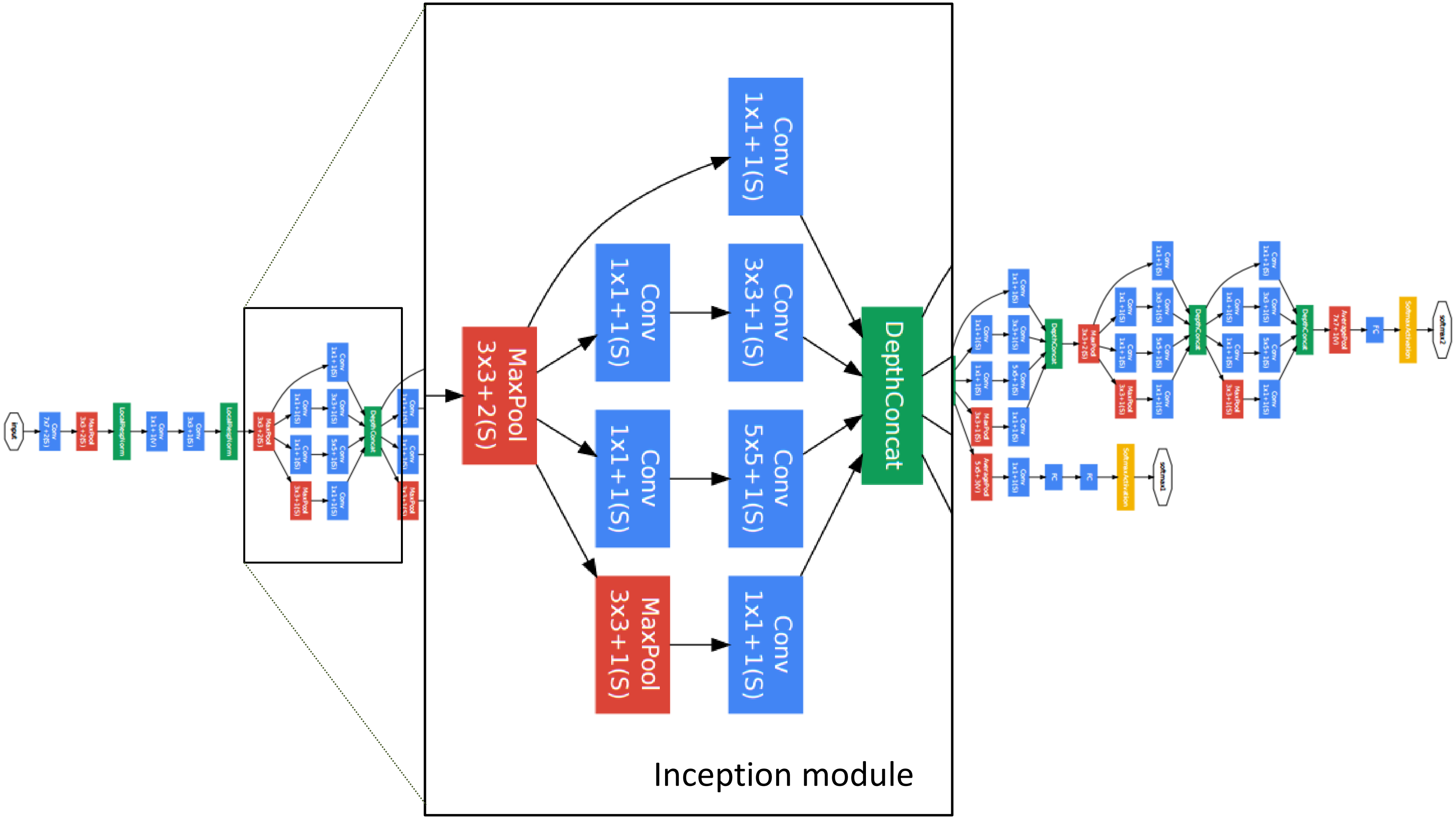
A short history of (old) CNNs

GoogLeNet (AKA Inception) [Szegedy et al. 2015]



A short history of (old) CNNs

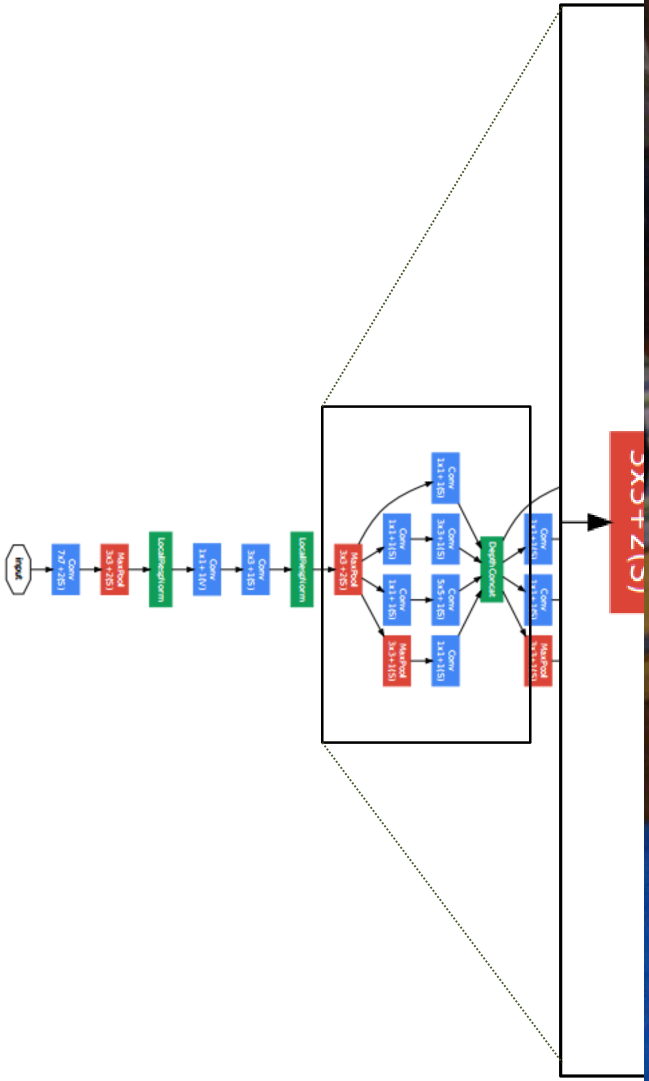
GoogLeNet (AKA Inception) [Szegedy et al. 2015]

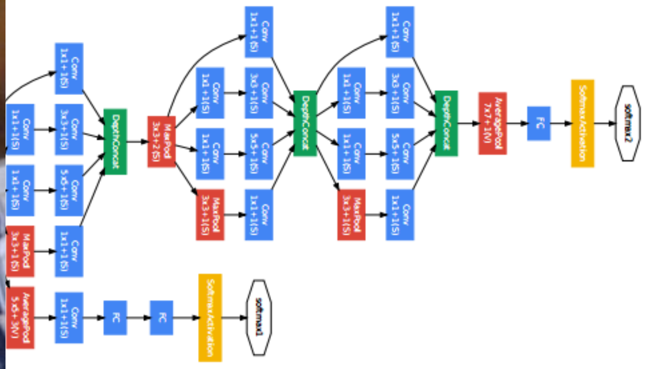


Inception module

A short history of (old) CNNs

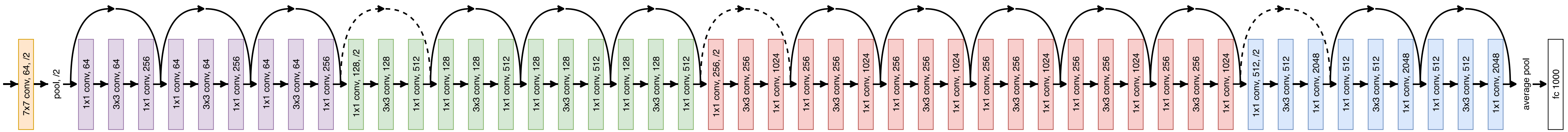
GoogLeN l. 2015]





ResNet-50

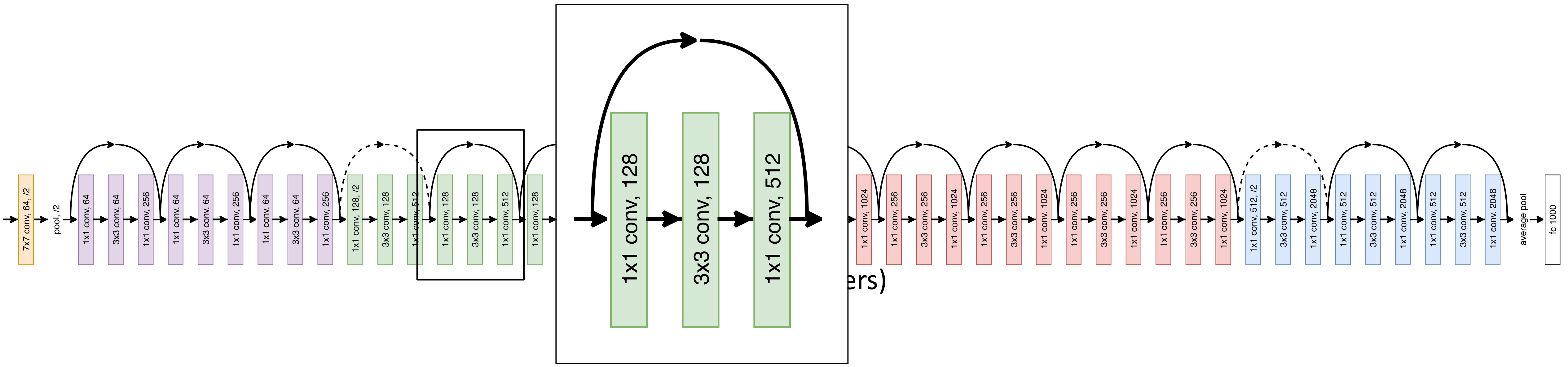
ResNets [He, Zhang, Ren, & Sun, 2016]



(Up to 152 layers)

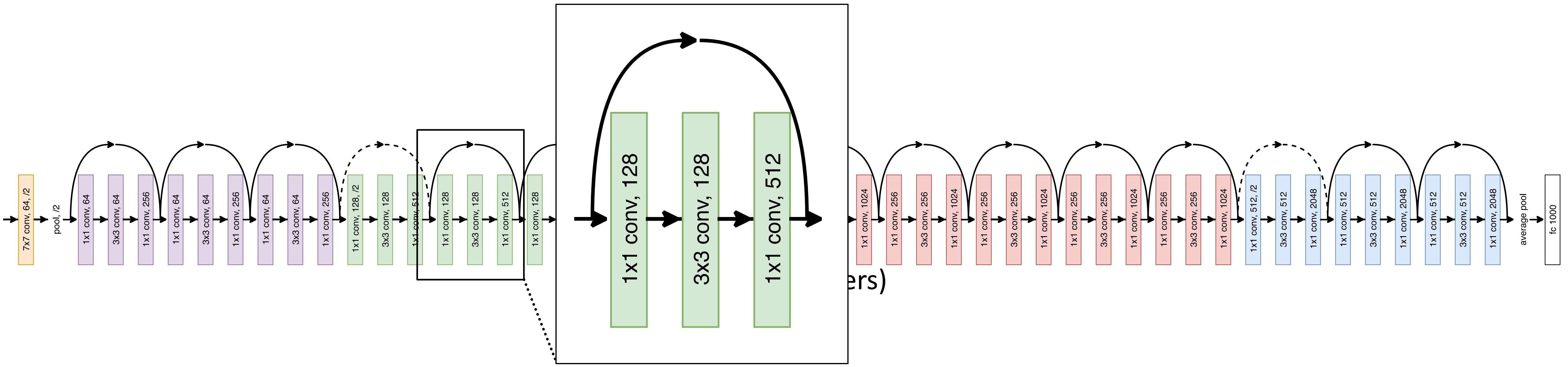
ResNet-50

ResNets [He, Zhang, Ren, & Sun, 2016]



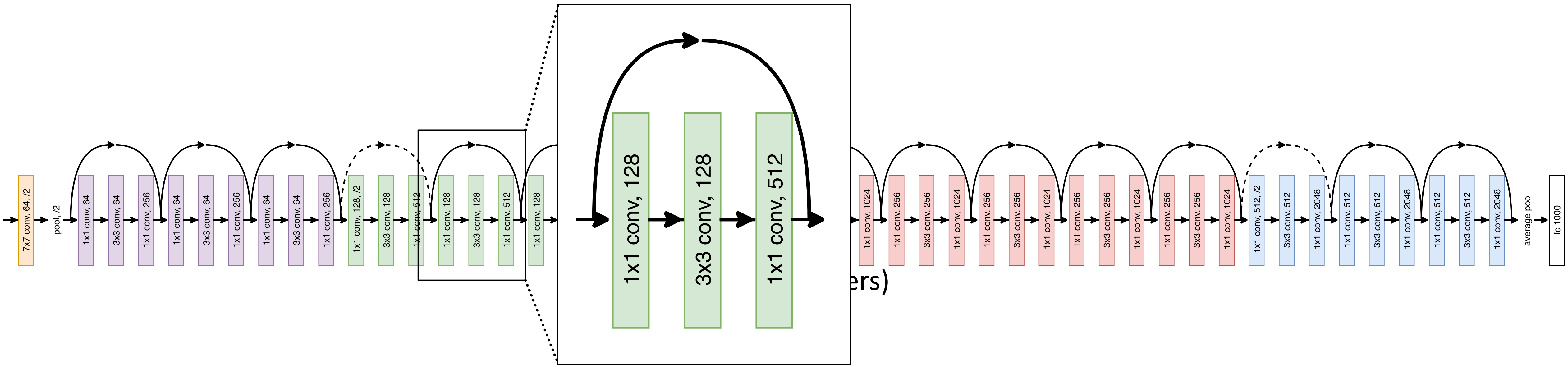
ResNet-50

ResNets [He, Zhang, Ren, & Sun, 2016]



ResNet-50

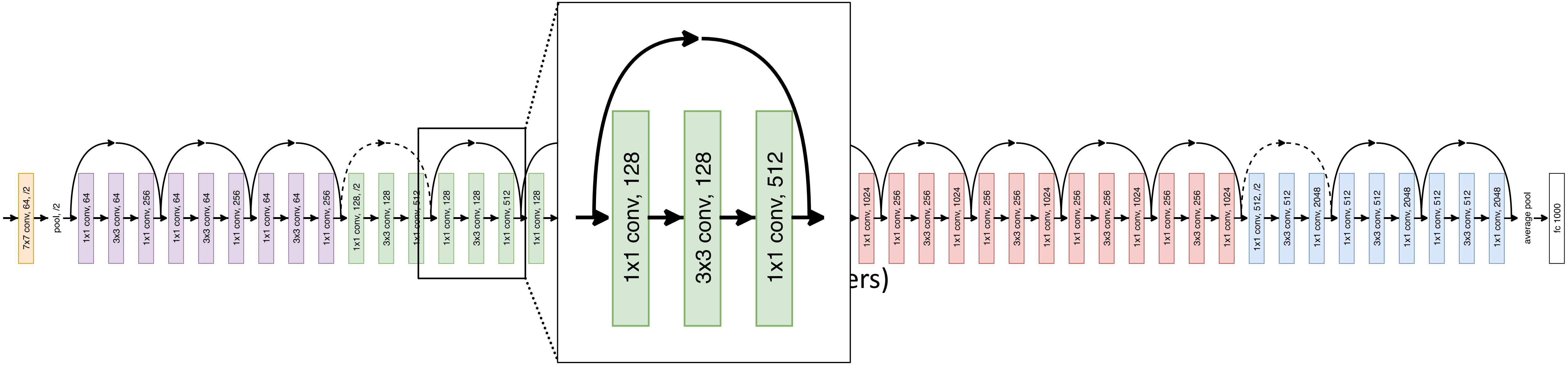
ResNets [He, Zhang, Ren, & Sun, 2016]



ers)

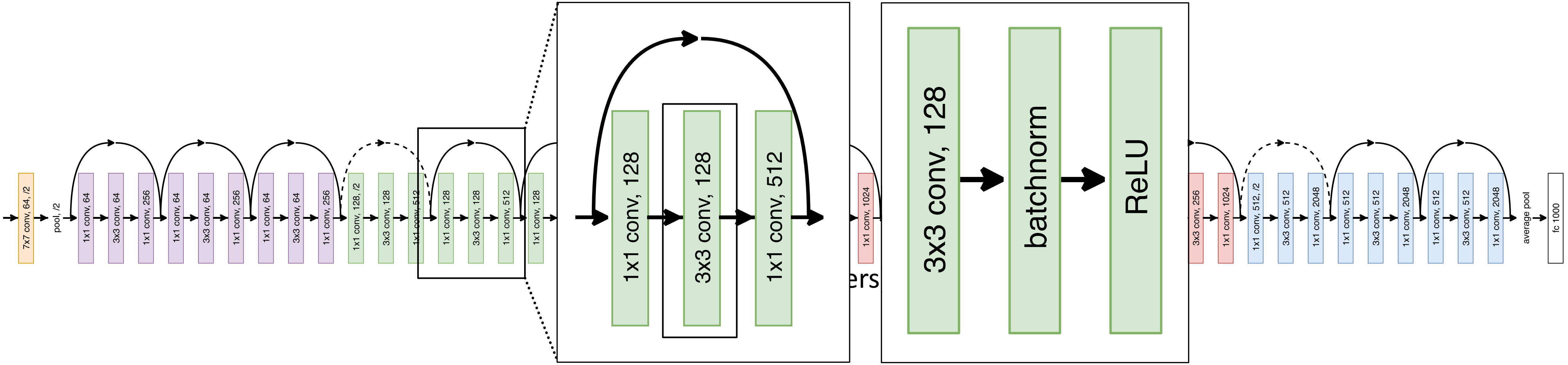
ResNet-50

ResNets [He, Zhang, Ren, & Sun, 2016]



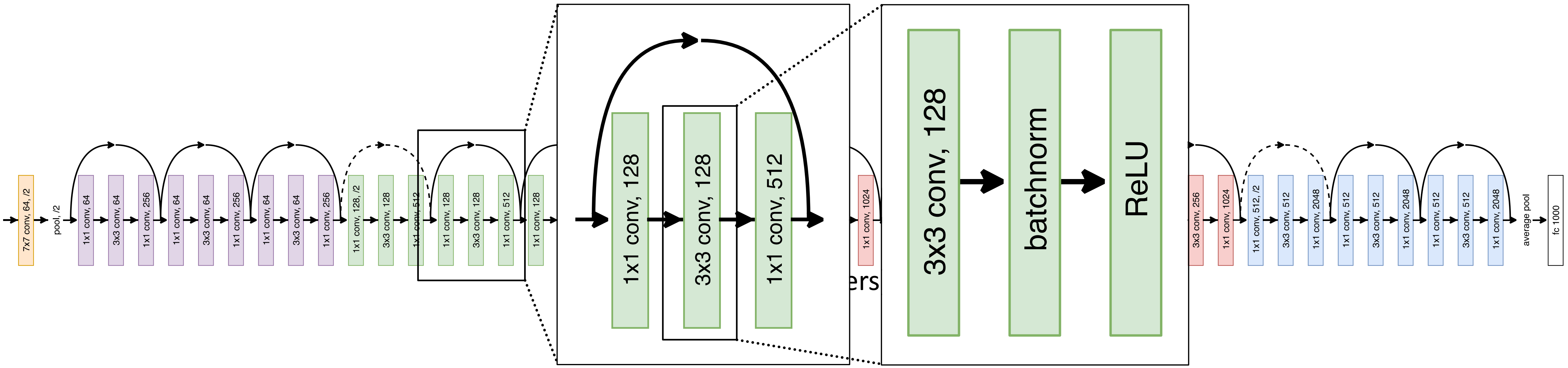
ResNet-50

ResNets [He, Zhang, Ren, & Sun, 2016]



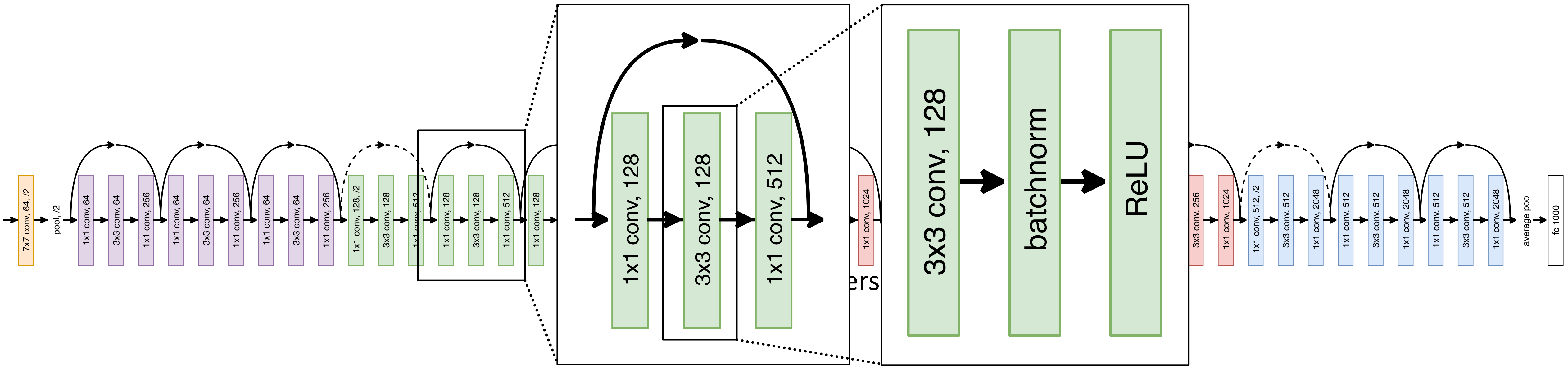
ResNet-50

ResNets [He, Zhang, Ren, & Sun, 2016]



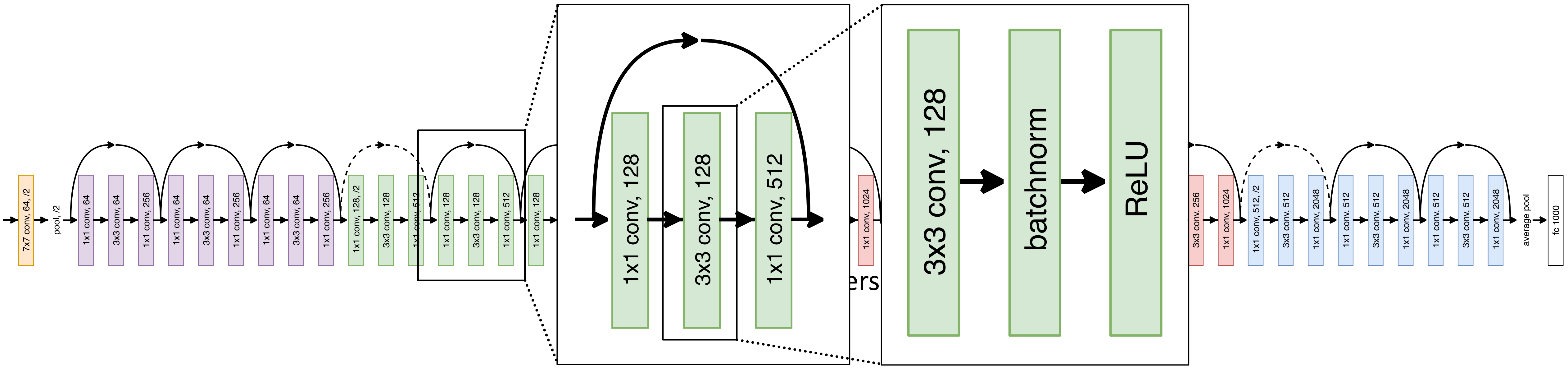
ResNet-50

ResNets [He, Zhang, Ren, & Sun, 2016]



ResNet-50

ResNets [He, Zhang, Ren, & Sun, 2016]

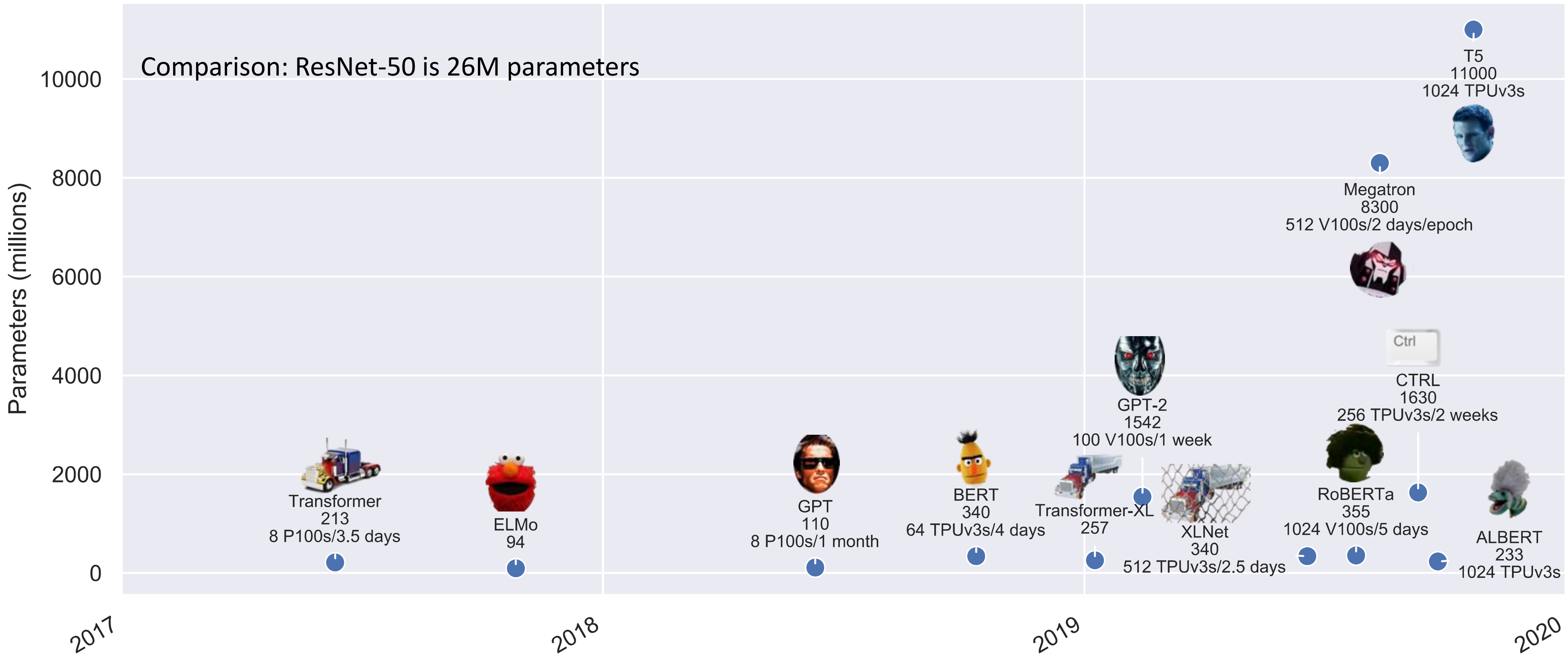


GPT-2 (transformers)

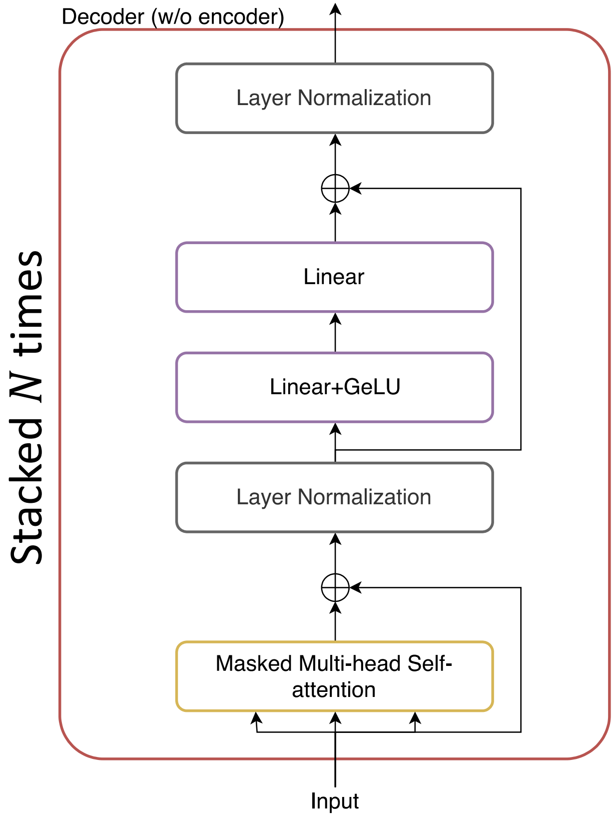
- Sequence-to-sequence models (like RNNs but with more parallelism)
- Revolutionizing NLP like AlexNet &co. did for computer vision

GPT-2 (transformers)

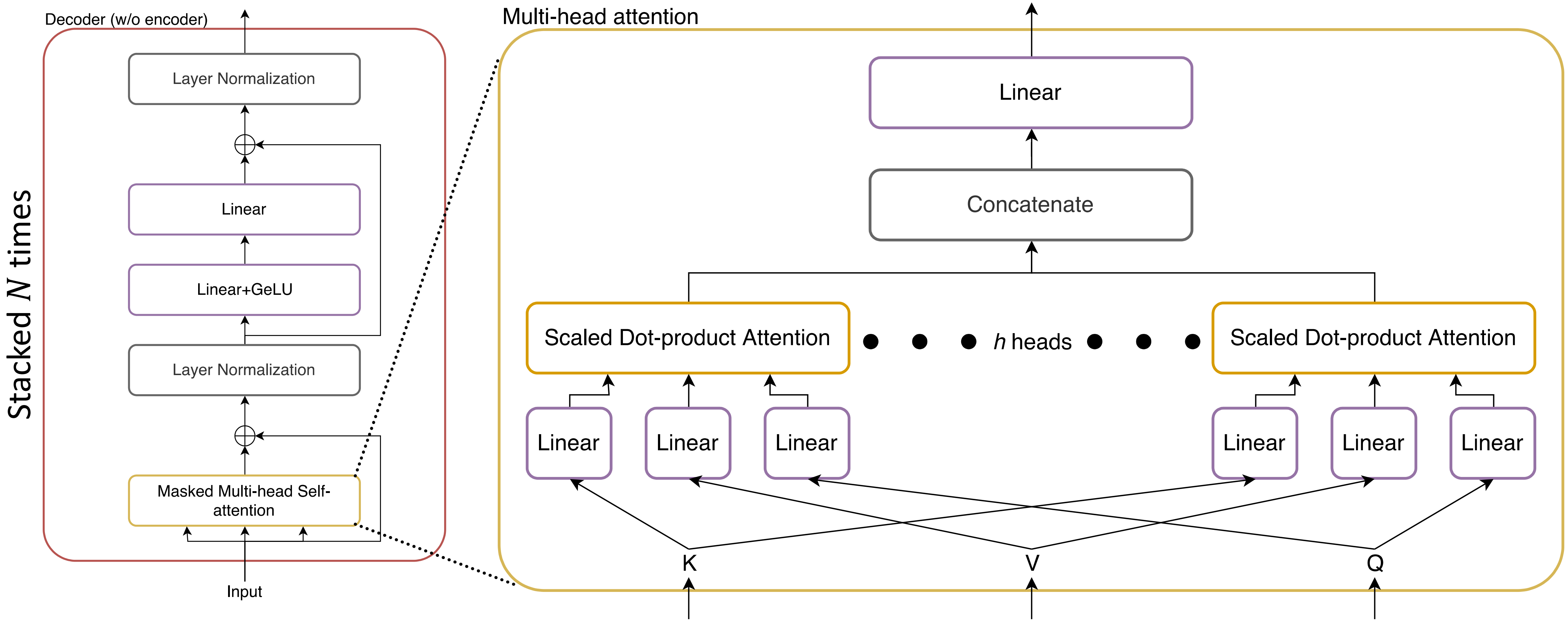
- Sequence-to-sequence models (like RNNs but with more parallelism)
- Revolutionizing NLP like AlexNet &co. did for computer vision



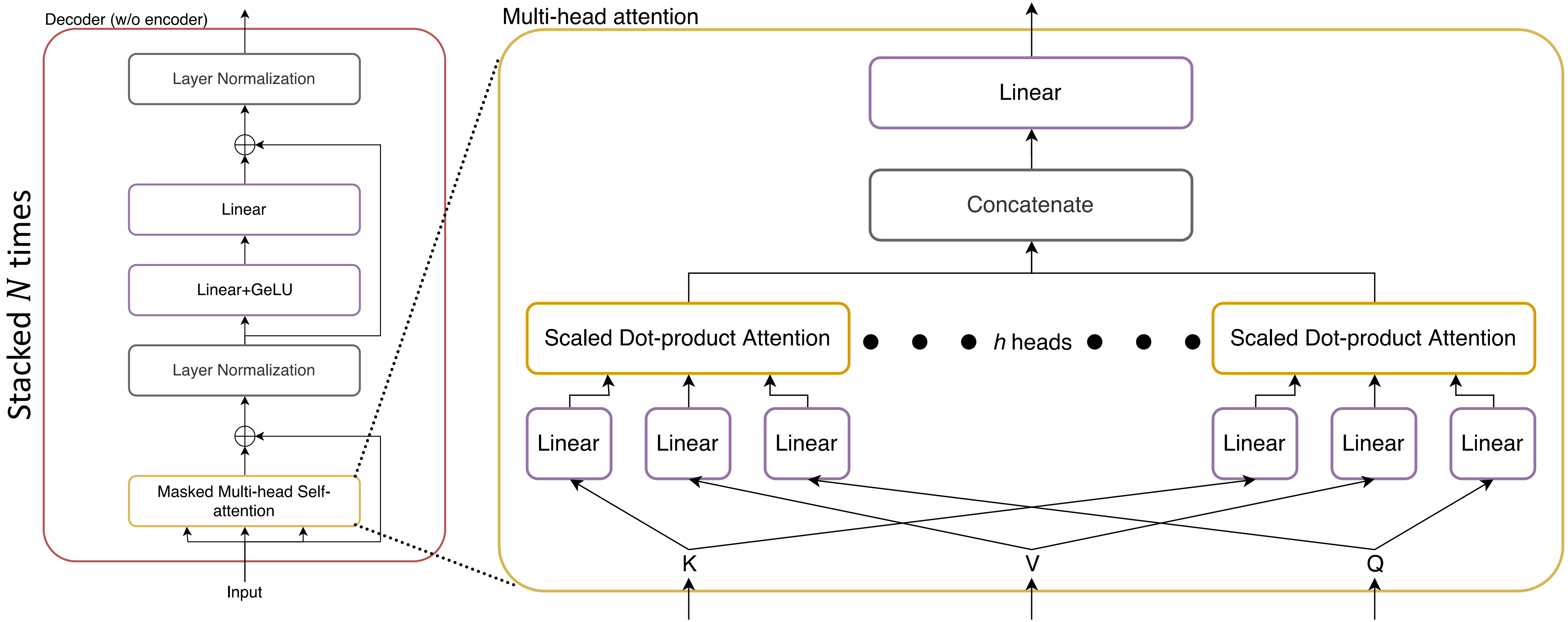
GPT-2 (transformers)



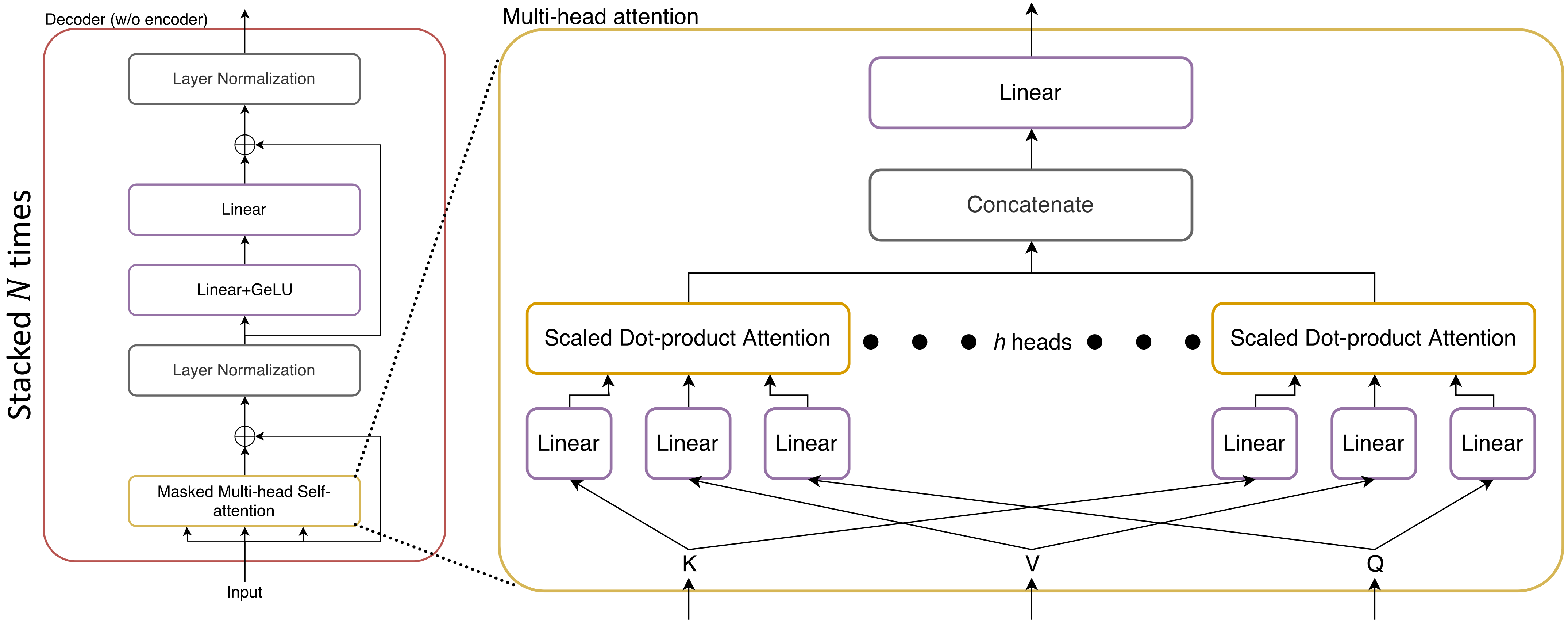
GPT-2 (transformers)



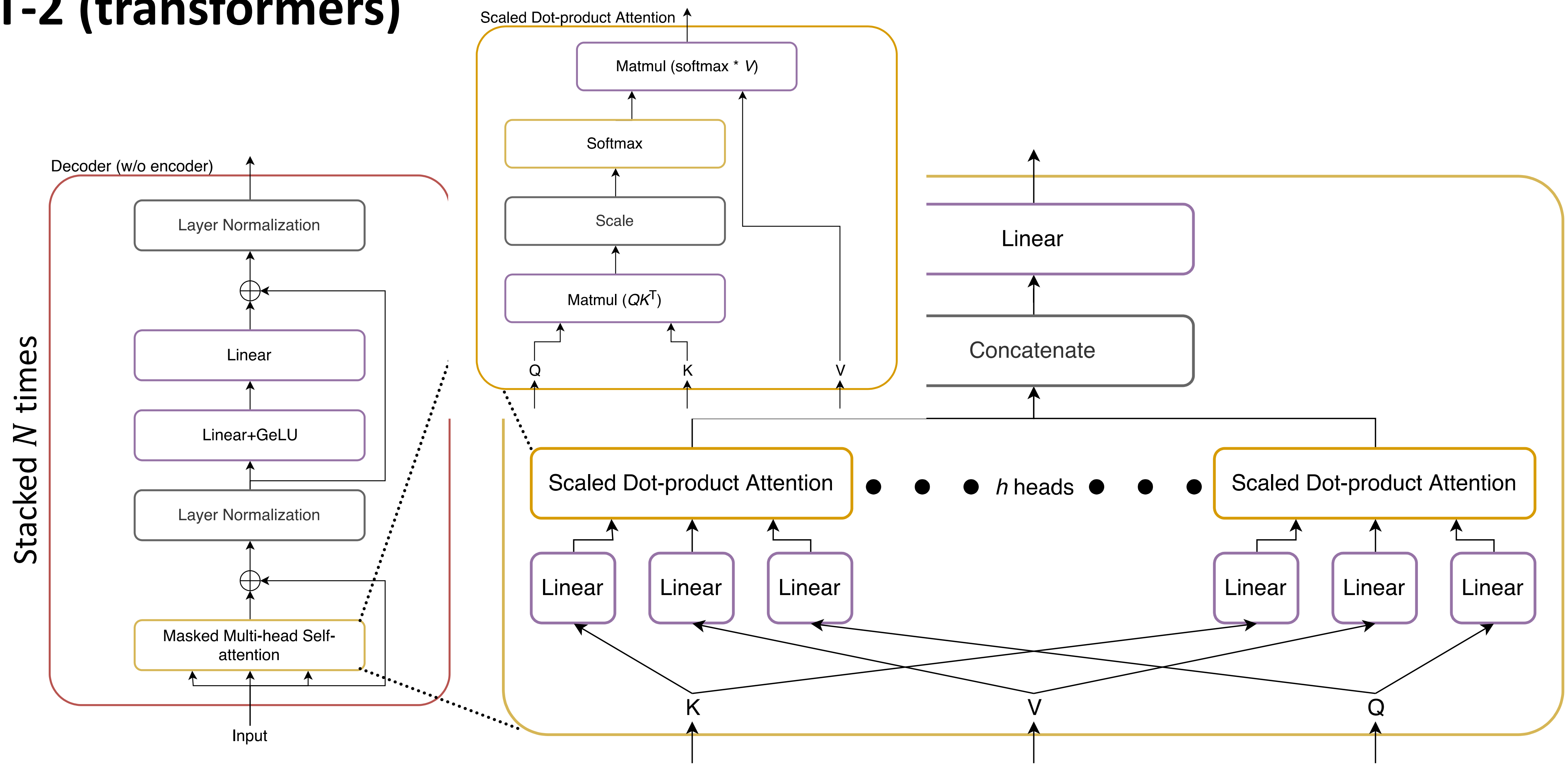
GPT-2 (transformers)



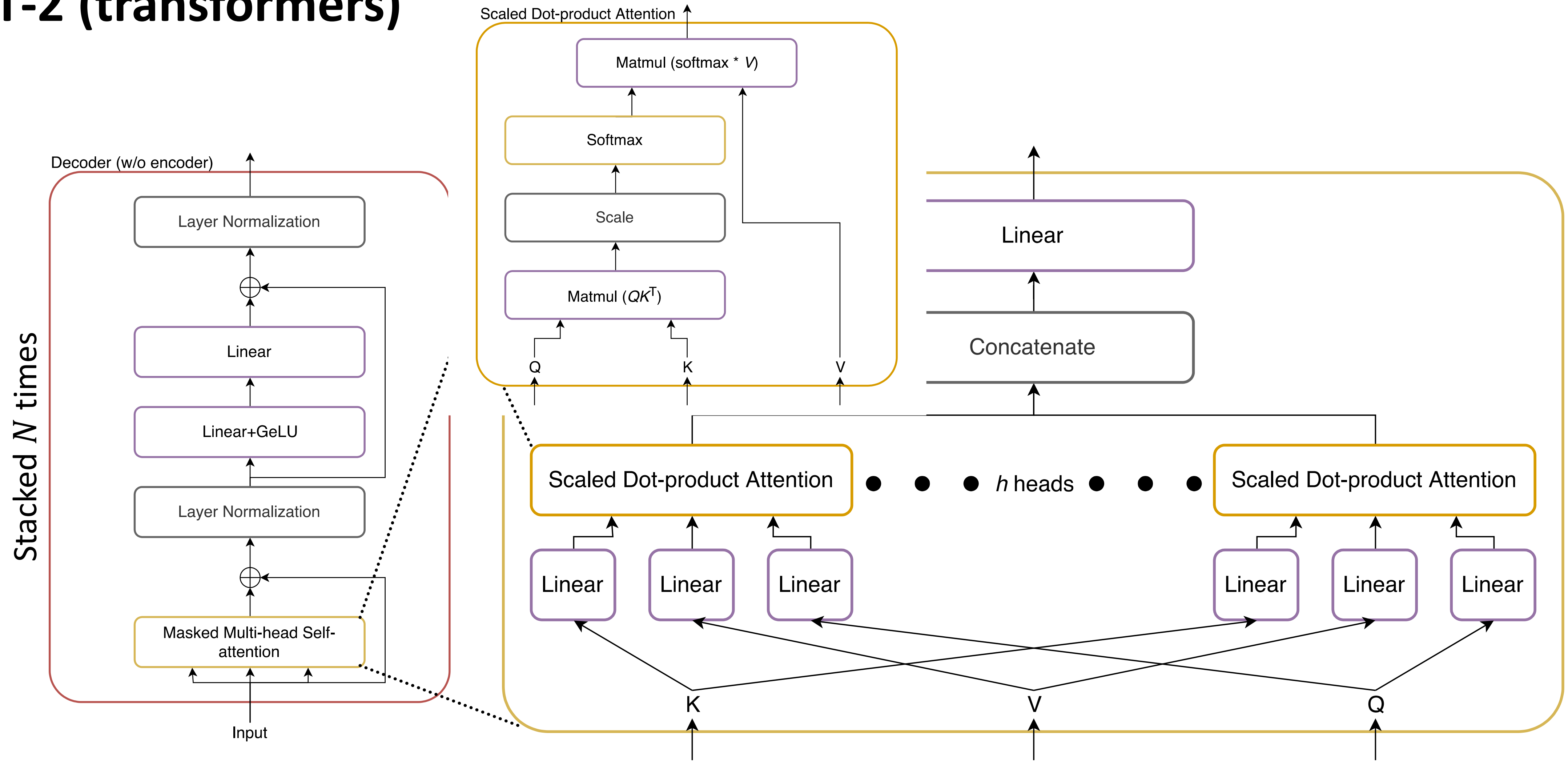
GPT-2 (transformers)



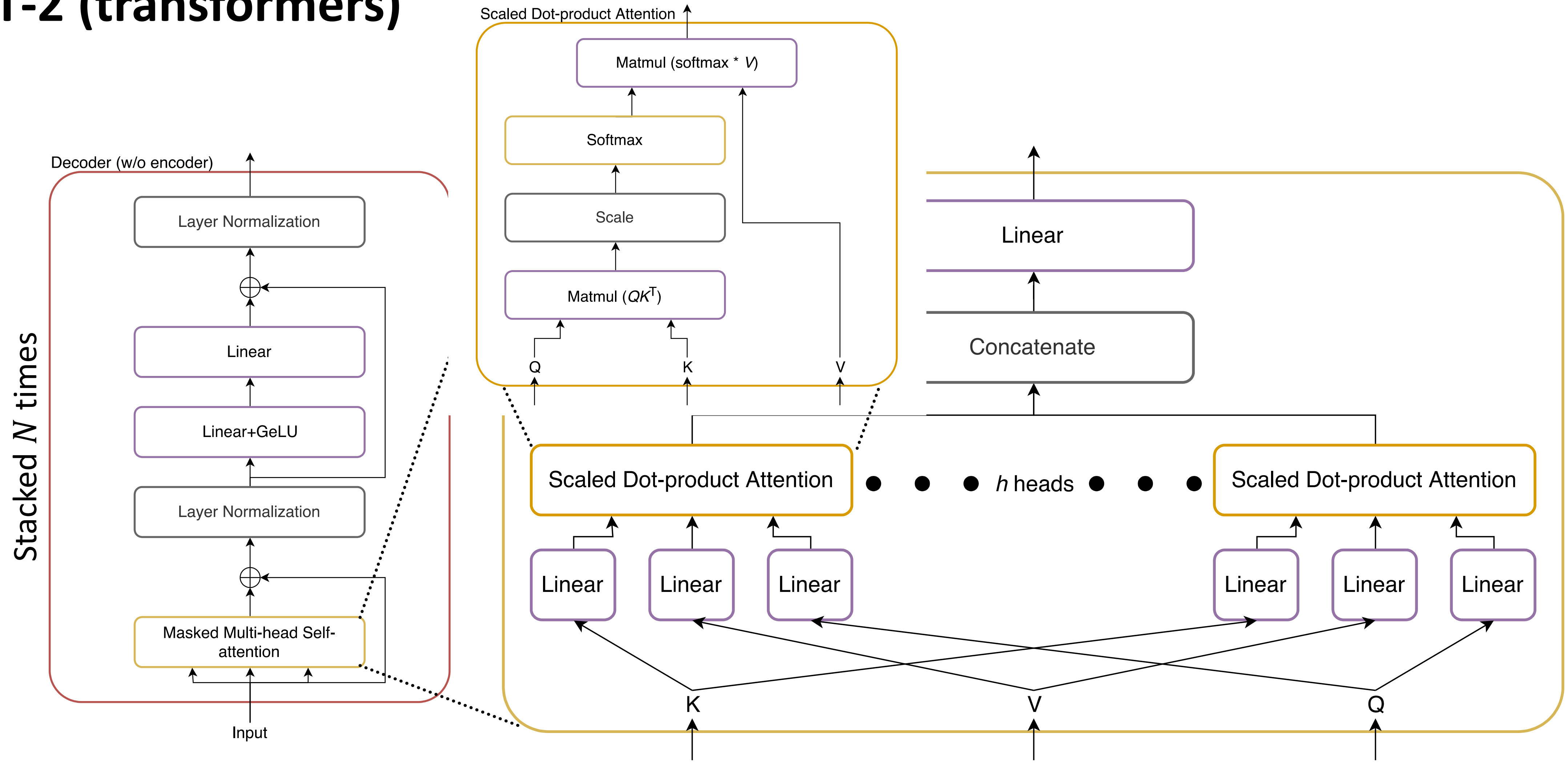
GPT-2 (transformers)



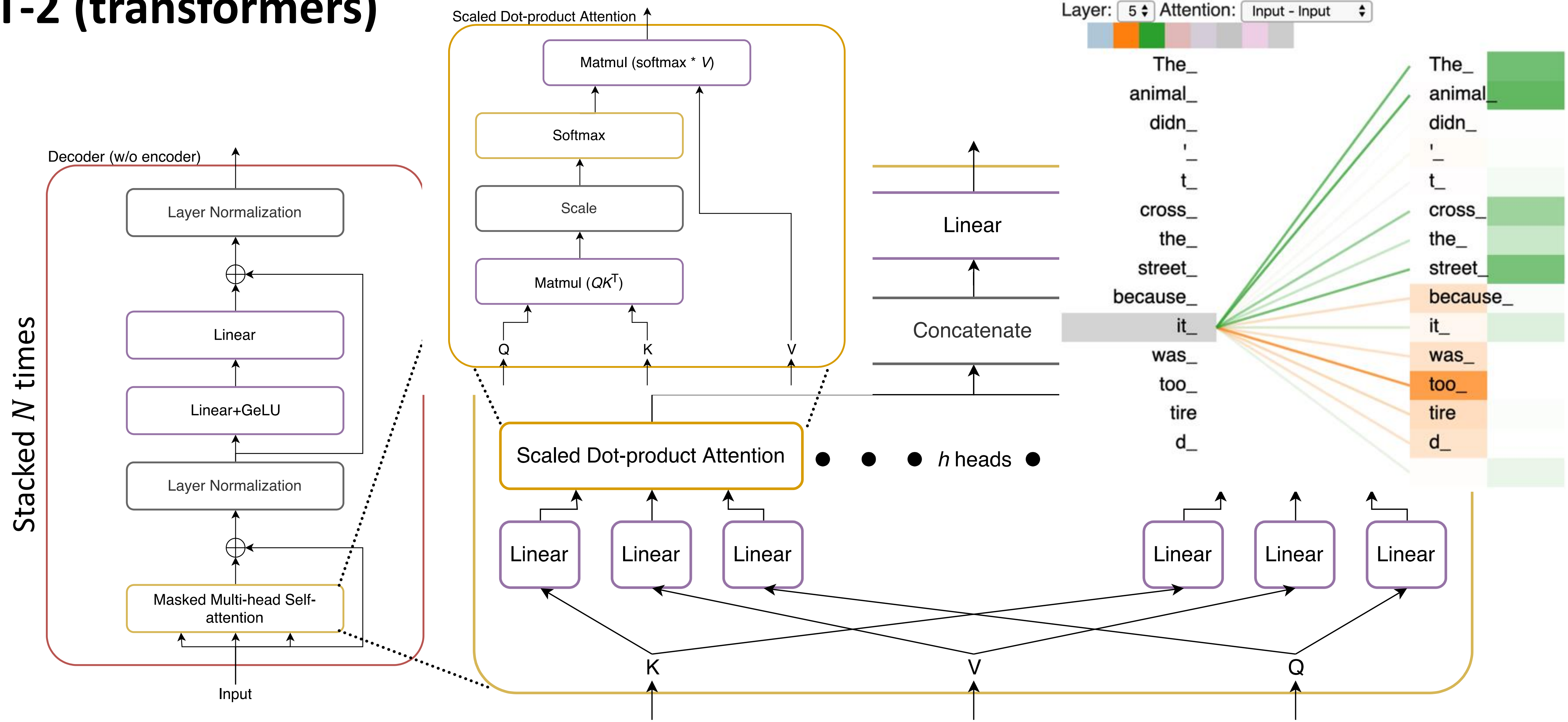
GPT-2 (transformers)



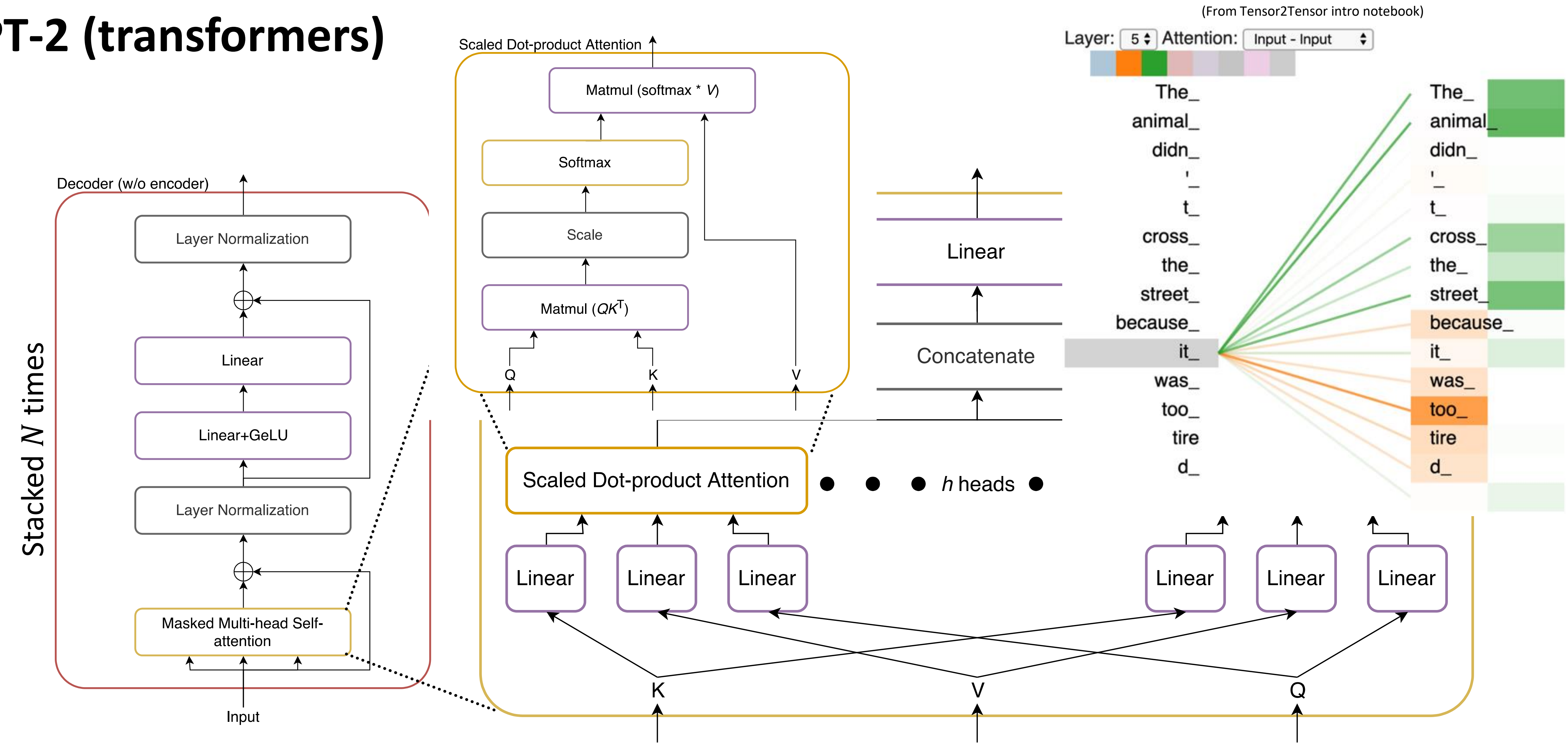
GPT-2 (transformers)



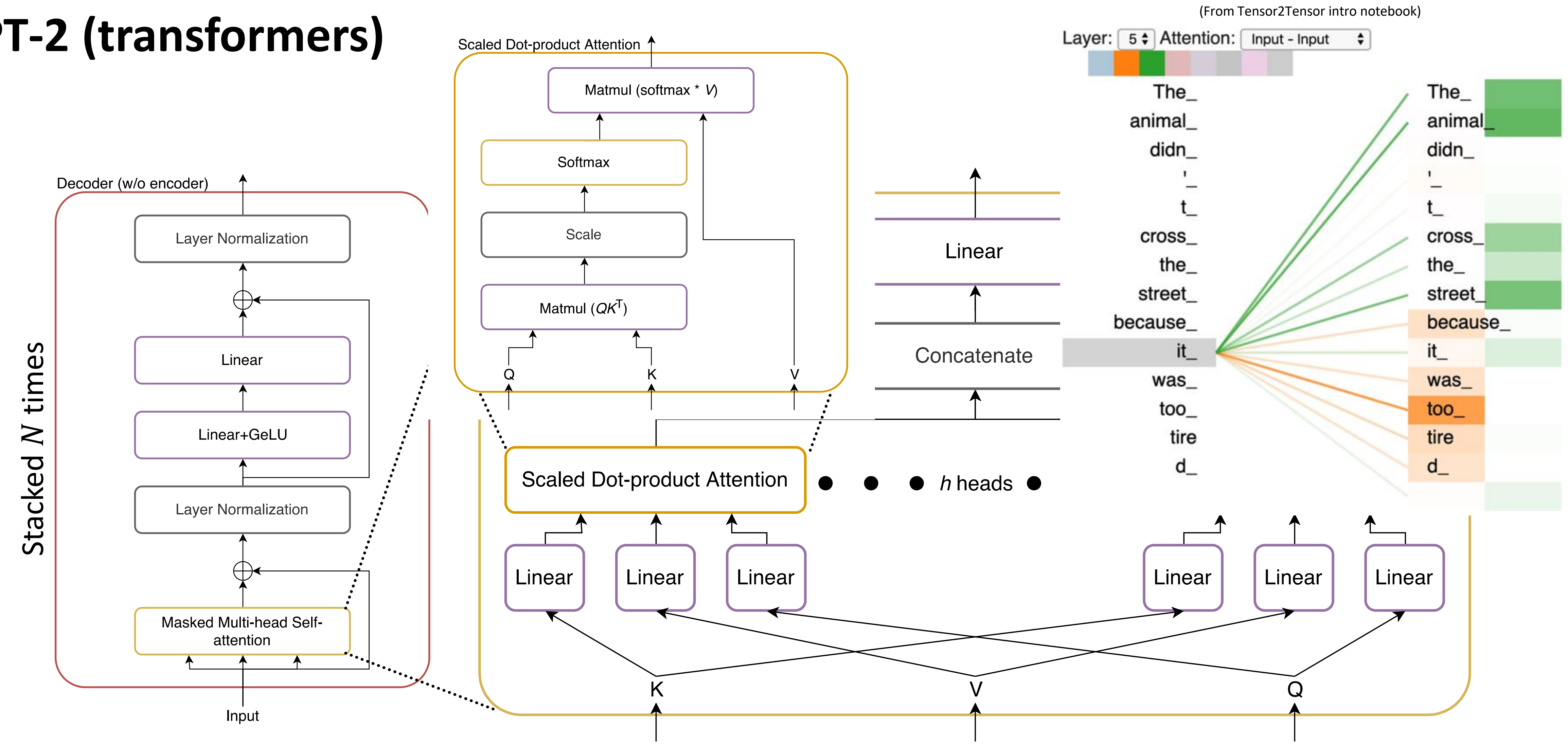
GPT-2 (transformers)



GPT-2 (transformers)



GPT-2 (transformers)



Objective: "predict the next word"

GPT-2 (transformers)

(From Tensor2Tensor intro notebook)

Scaled Dot-product Attention

Layer: 5 Attention: Input - Input

Decoder (w/o enco

Stacked N times

Prompt (human-written):

Recycling is good for the world.

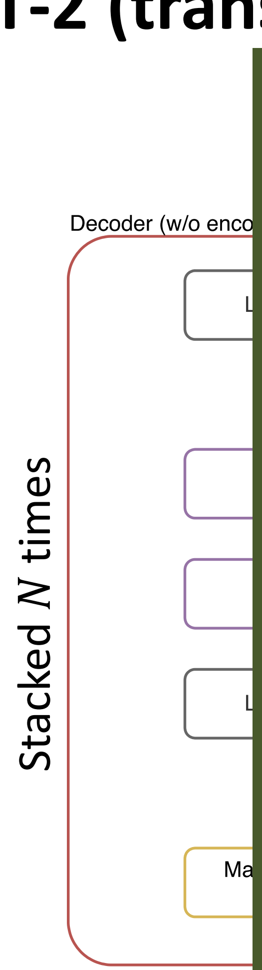
NO! YOU COULD NOT BE MORE WRONG!!

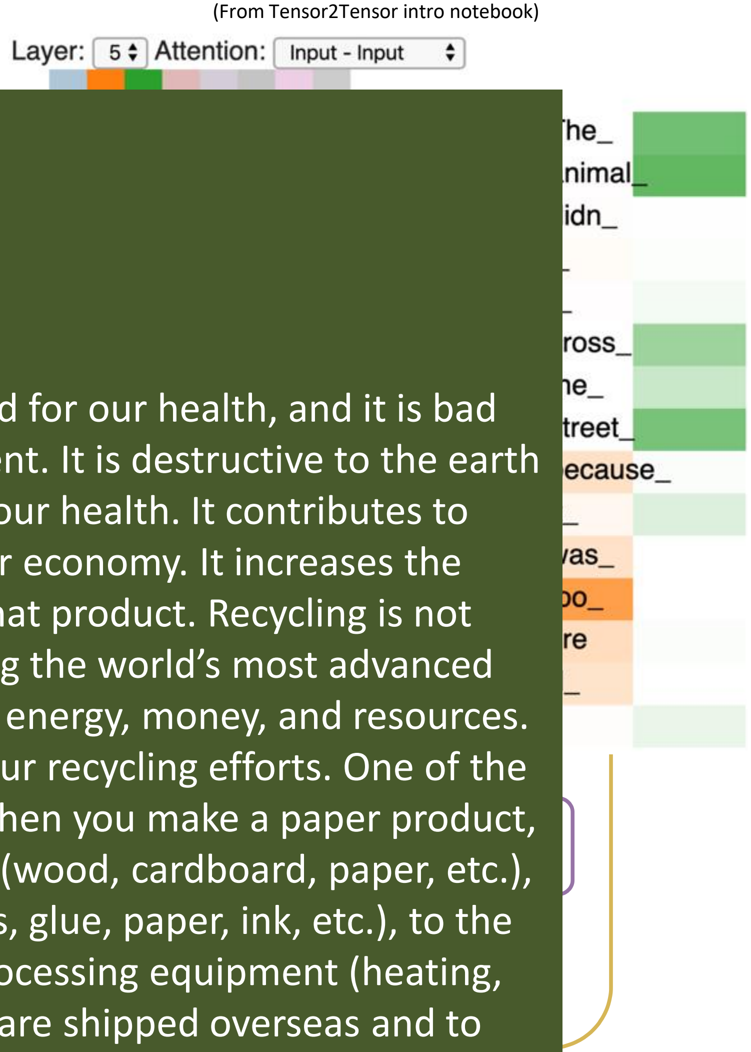
GPT-2-xlarge generated text:

Recycling is NOT good for the world. It is bad for the environment, it is bad for our health, and it is bad for our economy. I'm not kidding. Recycling is not good for the environment. It is destructive to the earth and it is a major contributor to global warming. Recycling is not good for our health. It contributes to obesity and diseases like heart disease and cancer. Recycling is bad for our economy. It increases the cost of a product, and in turn, the price of everything that is made with that product. Recycling is not good for our nation. We pay a tremendous price for the privilege of having the world's most advanced and efficient recycling system. Recycling is a huge, colossal waste of time, energy, money, and resources. And THAT is why we need to get back to basics and get back to basics in our recycling efforts. One of the best ways to start is to look at the process of creating a paper product. When you make a paper product, it is basically a long chain of materials. Everything from the raw materials (wood, cardboard, paper, etc.), to the reagents (dyes, solvents, etc.) to the printing equipment (chemicals, glue, paper, ink, etc.), to the packaging, to the packaging materials (mercury, chemicals, etc.) to the processing equipment (heating, cooling, etc.), to the packaging materials, to the packaging materials that are shipped overseas and to the packaging materials that are used in the United States. Each step along the way creates tons of waste that we constantly have to clean up. The process of making a paper product is a very wasteful one. But the end result is something that all of us need to consume. And if we want to keep the recycling process running efficiently, then we really need to think about each and every step that goes into making a paper product.

https://openai.com/blog/better-language-models/#sample8

GPT-2 (transformers)





Prompt (human-written):

Recycling is good for the world.

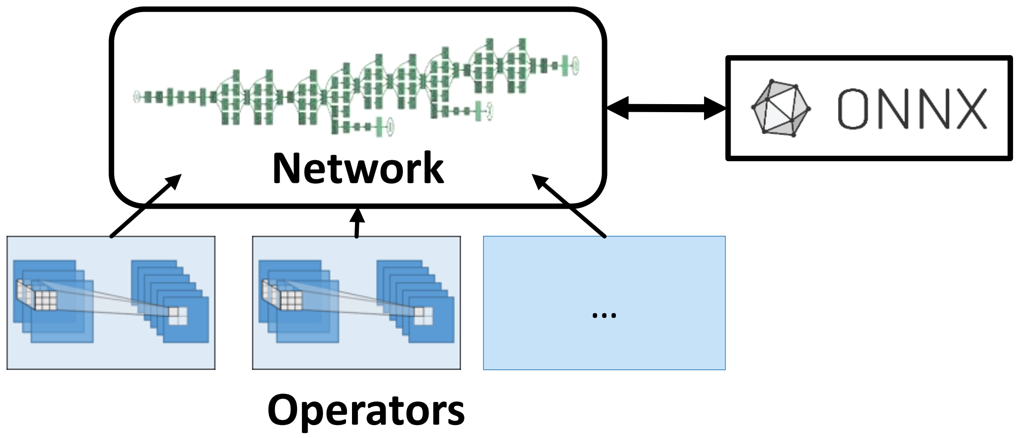
NO! YOU COULD NOT BE MORE WRONG!!

GPT-2-xlarge generated text:

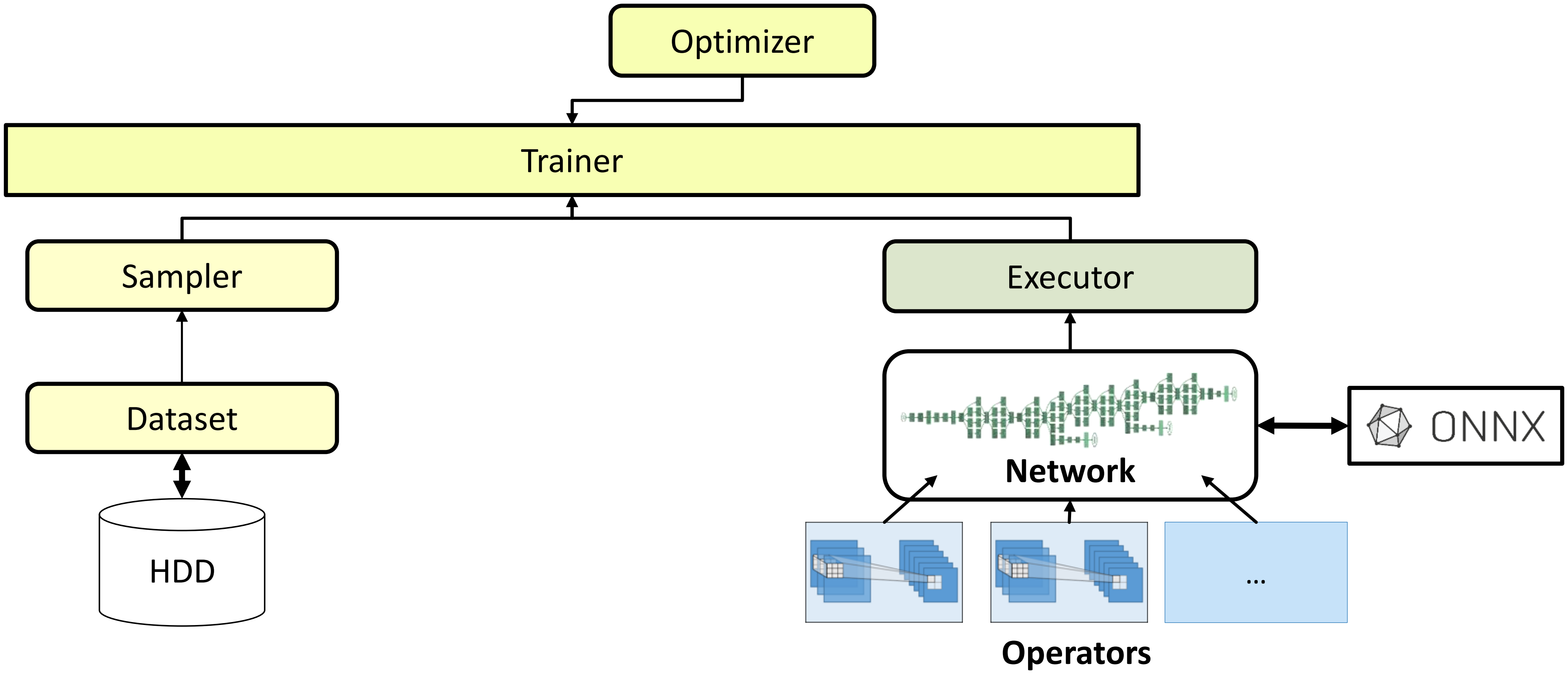
Recycling is NOT good for the world. It is bad for the environment, it is bad for our health, and it is bad for our economy. I'm not kidding. Recycling is not good for the environment. It is destructive to the earth and it is a major contributor to global warming. Recycling is not good for our health. It contributes to obesity and diseases like heart disease and cancer. Recycling is bad for our economy. It increases the cost of a product, and in turn, the price of everything that is made with that product. Recycling is not good for our nation. We pay a tremendous price for the privilege of having the world's most advanced and efficient recycling system. Recycling is a huge, colossal waste of time, energy, money, and resources. And THAT is why we need to get back to basics and get back to basics in our recycling efforts. One of the best ways to start is to look at the process of creating a paper product. When you make a paper product, it is basically a long chain of materials. Everything from the raw materials (wood, cardboard, paper, etc.), to the reagents (dyes, solvents, etc.) to the printing equipment (chemicals, glue, paper, ink, etc.), to the packaging, to the packaging materials (mercury, chemicals, etc.) to the processing equipment (heating, cooling, etc.), to the packaging materials, to the packaging materials that are shipped overseas and to the packaging materials that are used in the United States. Each step along the way creates tons of waste that we constantly have to clean up. The process of making a paper product is a very wasteful one. But the end result is something that all of us need to consume. And if we want to keep the recycling process running efficiently, then we really need to think about each and every step that goes into making a paper product.

https://openai.com/blog/better-language-models/#sample8

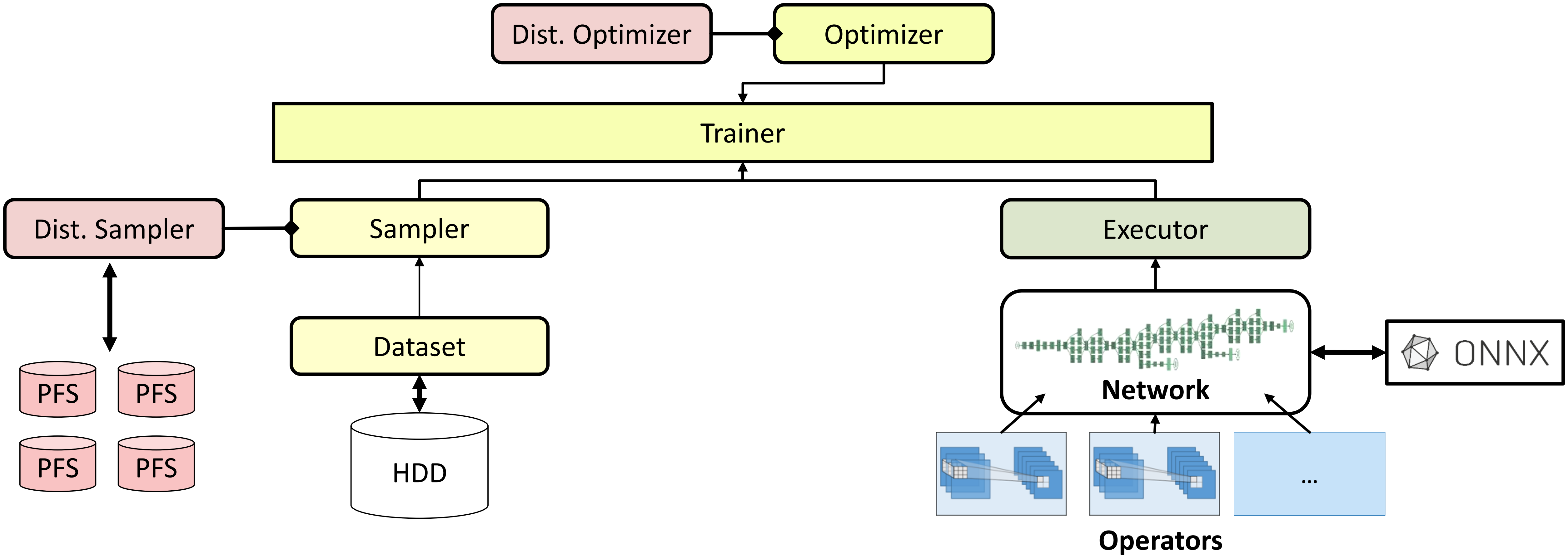
Networks



Training


Optimizer
Trainer
Sampler
Executor
Dataset
HDD
Network
ONNX
...
Operators

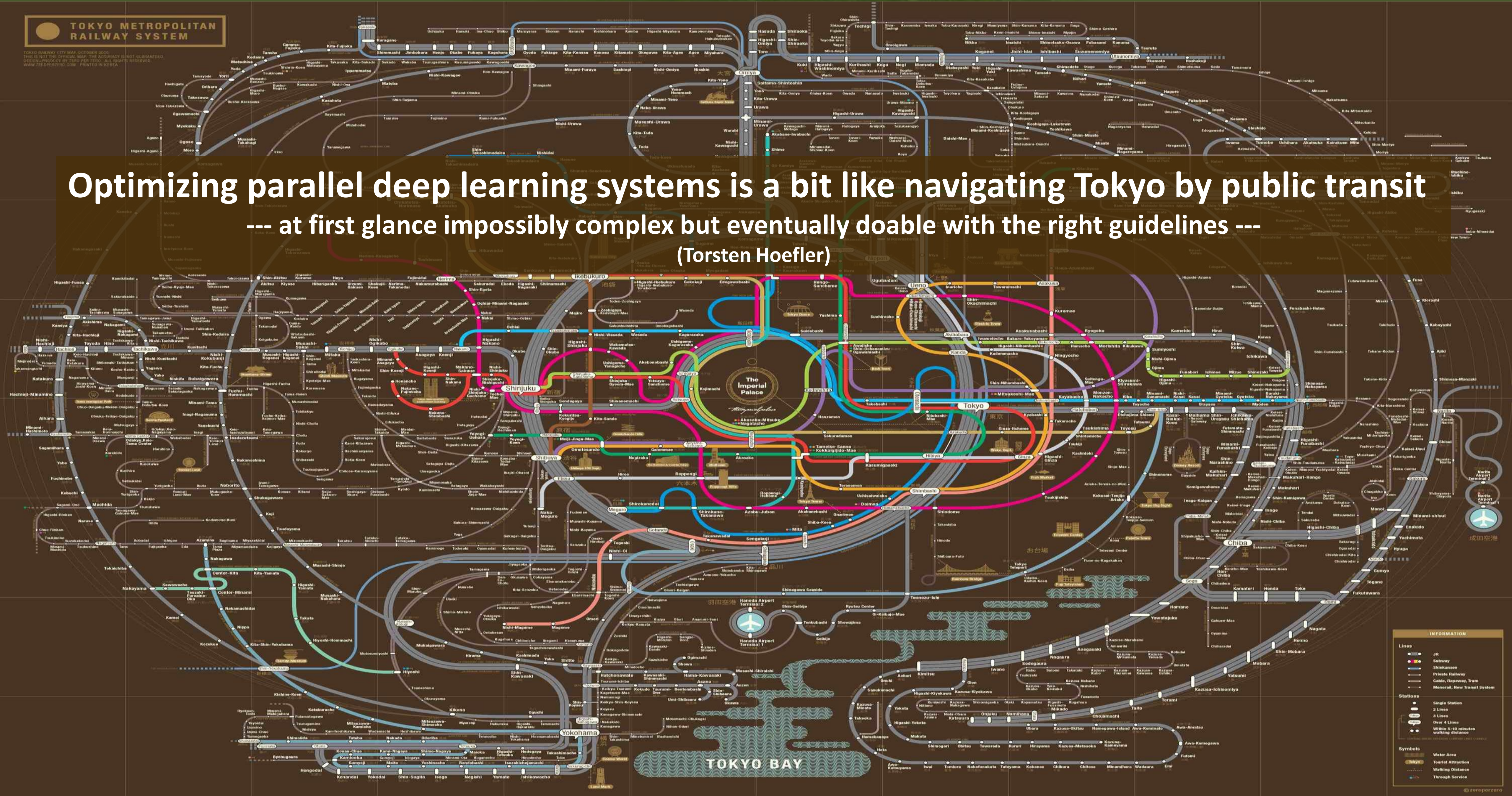
Distributed training



Optimizing parallel deep learning systems is a bit like navigating Tokyo by public transit

--- at first glance impossibly complex but eventually doable with the right guidelines ---

(Torsten Hoefler)



Operator implementations: fully-connected layers

$$Y = \sigma(WX + b)$$

Operator implementations: fully-connected layers

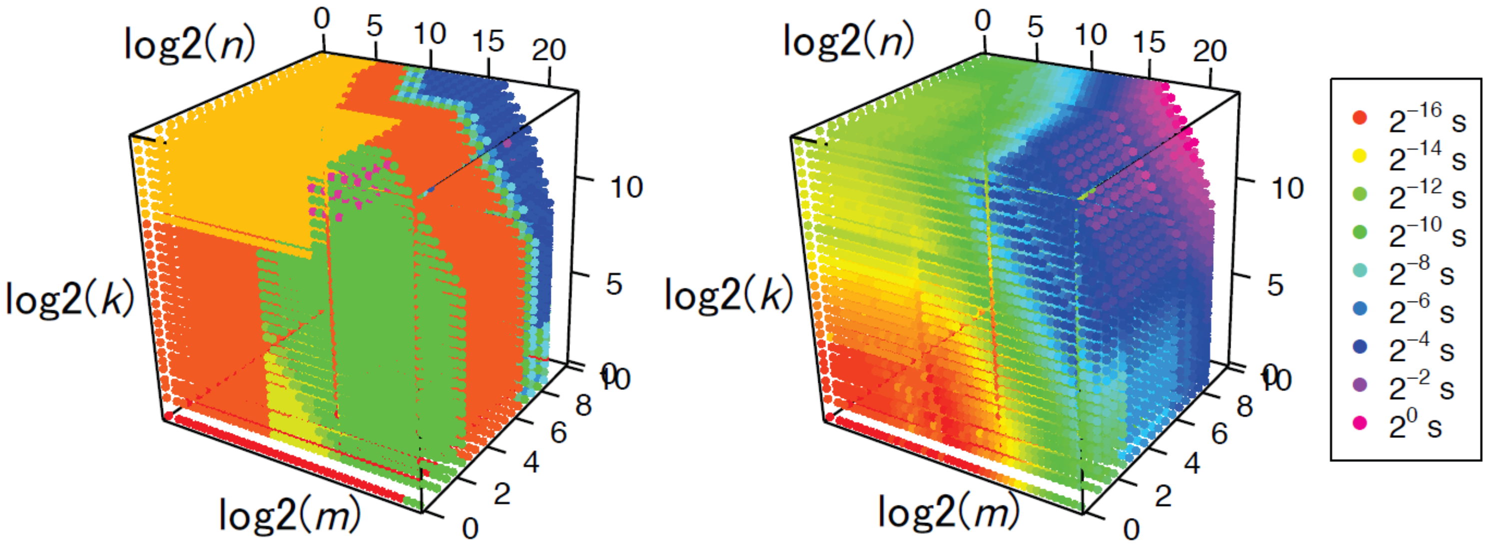
$$Y = \sigma(WX + b)$$

- **Dominated by matrix-matrix multiplication**
  - Standard tricks: vectorize, tile, fusion, …
- **BLAS3 GEMM**
  - cuBLAS, MKL, …

Operator implementations: fully-connected layers

$$Y = \sigma(WX + b)$$

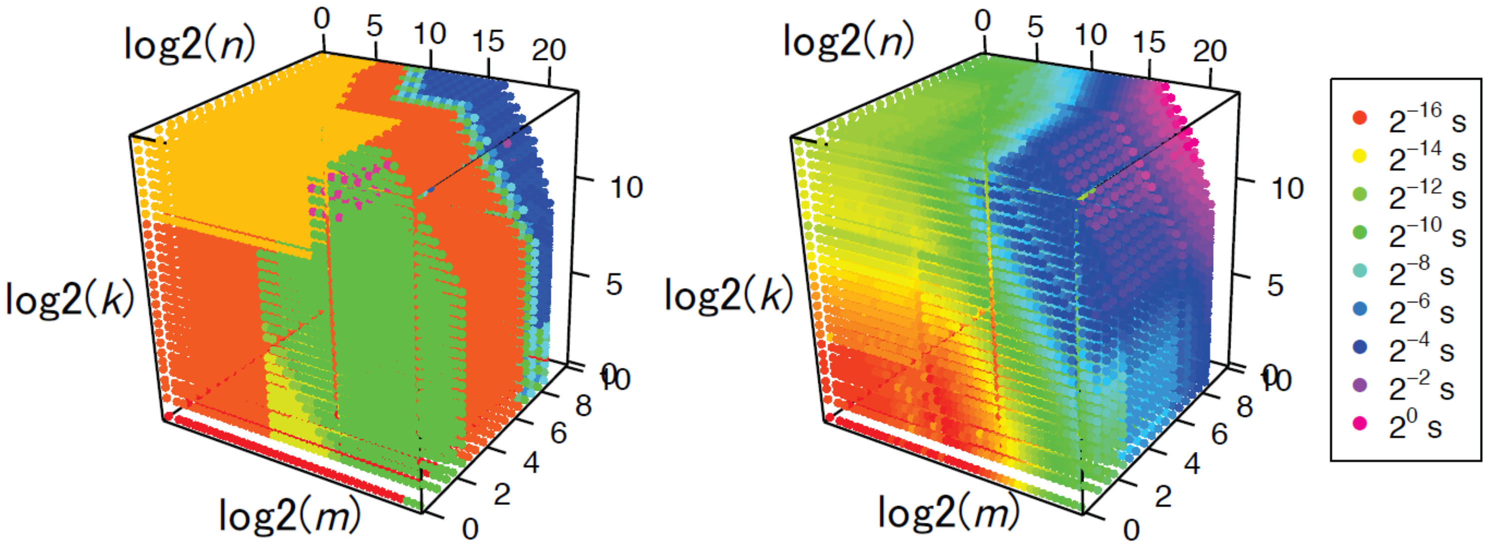
- **Dominated by matrix-matrix multiplication**
  - Standard tricks: vectorize, tile, fusion, …
- **BLAS3 GEMM**
  - cuBLAS, MKL, …



Operator implementations: fully-connected layers

$$Y = \sigma(WX + b)$$

- **Dominated by matrix-matrix multiplication**
  - Standard tricks: vectorize, tile, fusion, …
- **BLAS3 GEMM**
  - cuBLAS, MKL, …



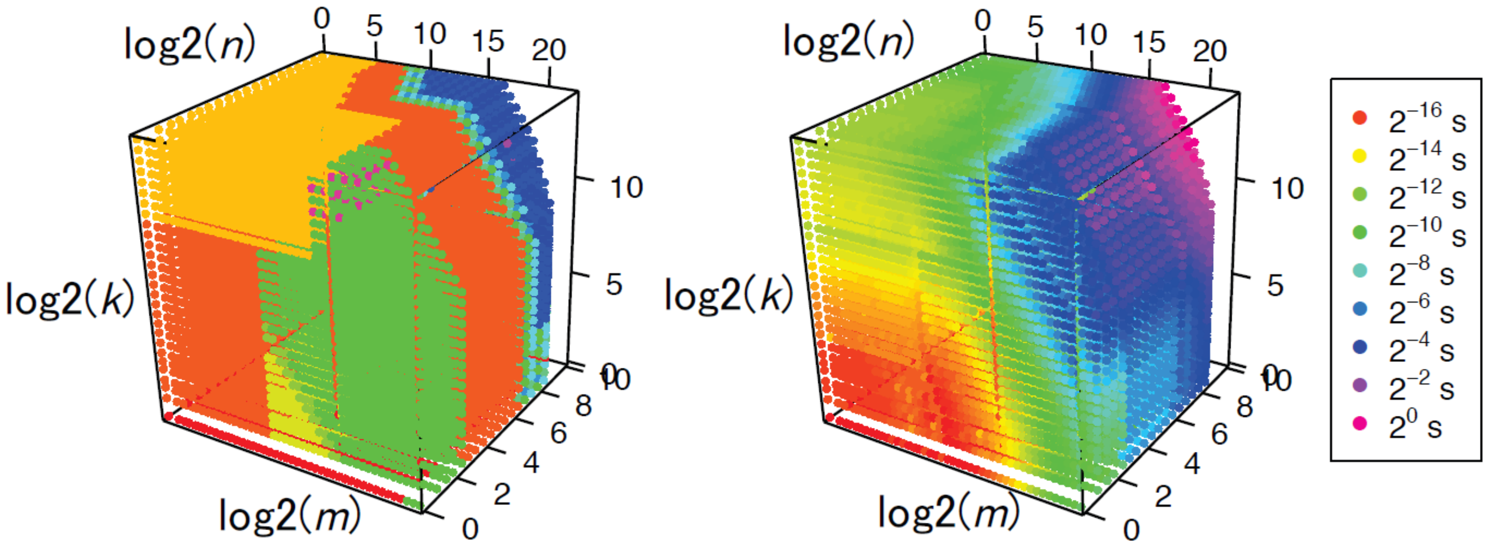
[Oyama et al.]

Operator implementations: fully-connected layers

$$Y = \sigma(WX + b)$$

- **Dominated by matrix-matrix multiplication**
  - Standard tricks: vectorize, tile, fusion, …
- **BLAS3 GEMM**
  - cuBLAS, MKL, …

Performance is not consistent!



[Oyama et al.]

Operator implementations: convolution

$$Y_{k,f,i,j} = \sum_{c=0}^{C-1} \sum_{a=-O}^{O} \sum_{b=-O}^{O} X_{k,c,i+a,j+b} w_{f,c,a+O,b+O}$$

[Chellapilla et al. 2006; Mathieu et al. 2014; Lavin & Gray 2016; Liu et al. 2017]

Operator implementations: convolution

$$Y_{k,f,i,j} = \sum_{c=0}^{C-1} \sum_{a=-O}^{O} \sum_{b=-O}^{O} X_{k,c,i+a,j+b} w_{f,c,a+O,b+O}$$

Direct

Indirect

[Chellapilla et al. 2006; Mathieu et al. 2014; Lavin & Gray 2016; Liu et al. 2017]

Operator implementations: convolution

$$Y_{k,f,i,j} = \sum_{c=0}^{C-1} \sum_{a=-O}^{O} \sum_{b=-O}^{O} X_{k,c,i+a,j+b} w_{f,c,a+O,b+O}$$

Direct

$$\begin{bmatrix} 4 & 1 & 9 & 8 \\ 5 & 9 & 9 & 8 \\ 0 & 7 & 3 & 4 \\ 2 & 6 & 3 & 1 \end{bmatrix} * \begin{bmatrix} 1 & -1 & 0 \\ 0.1 & -2 & 0 \\ 3 & 4 & 1.1 \end{bmatrix} = \begin{bmatrix} 21.9 & 59.3 & 53.9 & 43.9 \\ -6.3 & 16.8 & 12.3 & 12 \\ 9.6 & 15.3 & 25.8 & 14 \\ 0.4 & 7.1 & 52.1 & 53.1 \end{bmatrix}$$

Indirect

[Chellapilla et al. 2006; Mathieu et al. 2014; Lavin & Gray 2016; Liu et al. 2017]

Operator implementations: convolution

$$Y_{k,f,i,j} = \sum_{c=0}^{C-1} \sum_{a=-O}^{O} \sum_{b=-O}^{O} X_{k,c,i+a,j+b} w_{f,c,a+O,b+O}$$

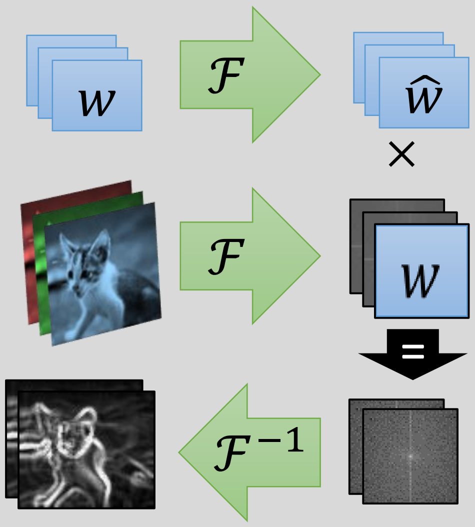
Direct

Direct

$$\begin{bmatrix} 4 & 1 & 9 & 8 \\ 5 & 9 & 9 & 8 \\ 0 & 7 & 3 & 4 \\ 2 & 6 & 3 & 1 \end{bmatrix} * \begin{bmatrix} 1 & -1 & 0 \\ 0.1 & -2 & 0 \\ 3 & 4 & 1.1 \end{bmatrix} = \begin{bmatrix} 21.9 & 59.3 & 53.9 & 43.9 \\ -6.3 & 16.8 & 12.3 & 12 \\ 9.6 & 15.3 & 25.8 & 14 \\ 0.4 & 7.1 & 52.1 & 53.1 \end{bmatrix}$$

Indirect

FFT



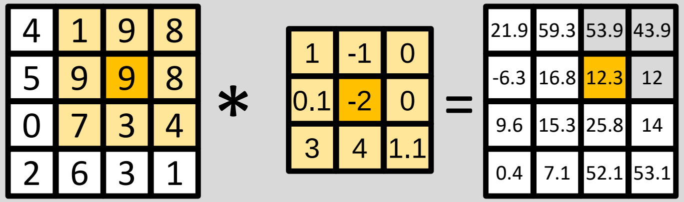
[Chellapilla et al. 2006; Mathieu et al. 2014; Lavin & Gray 2016; Liu et al. 2017]

Operator implementations: convolution

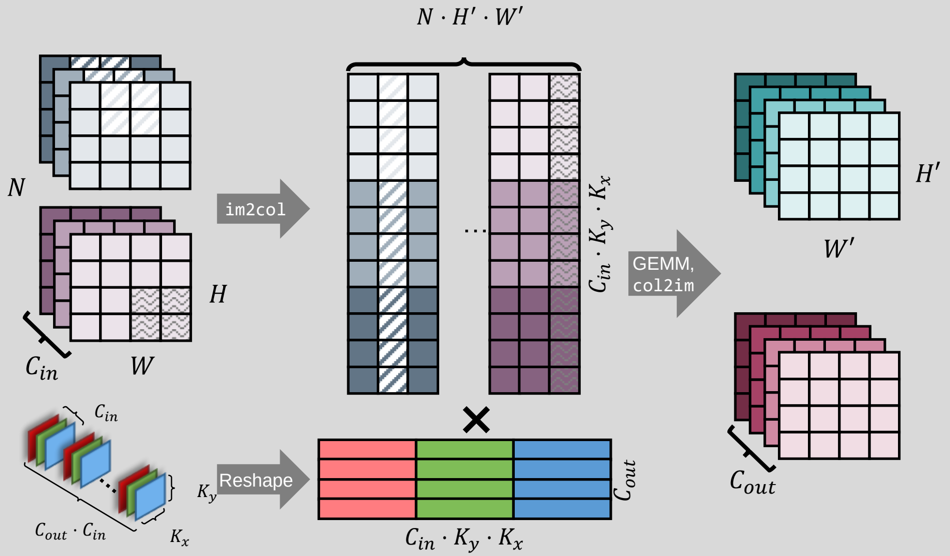
$$Y_{k,f,i,j} = \sum_{c=0}^{C-1} \sum_{a=-O}^{O} \sum_{b=-O}^{O} X_{k,c,i+a,j+b} w_{f,c,a+O,b+O}$$

Direct

Direct

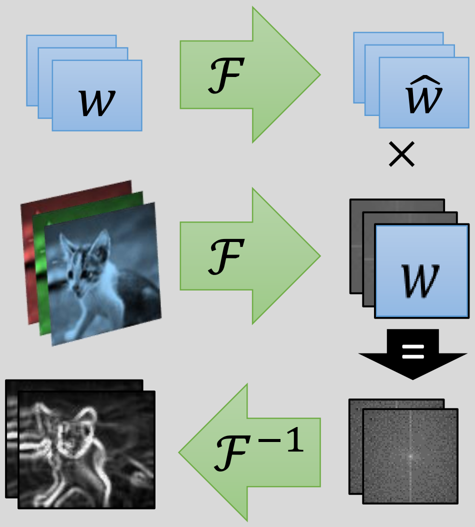


im2col



Indirect

FFT



[Chellapilla et al. 2006; Mathieu et al. 2014; Lavin & Gray 2016; Liu et al. 2017]

Operator implementations: convolution

$$Y_{k,f,i,j} = \sum_{c=0}^{C-1} \sum_{a=-O}^{O} \sum_{b=-O}^{O} X_{k,c,i+a,j+b} w_{f,c,a+O,b+O}$$

Direct

Direct

4	1	9	8
5	9	9	8
0	7	3	4
2	6	3	1

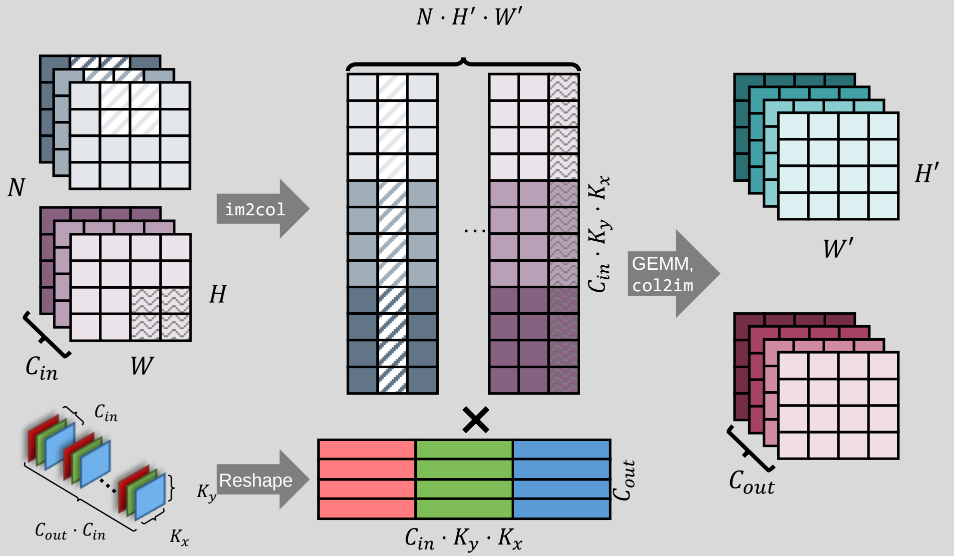
$*$

1	-1	0
0.1	-2	0
3	4	1.1

$=$

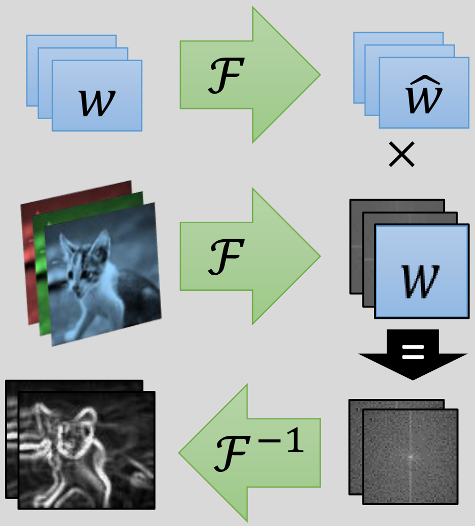
21.9	59.3	53.9	43.9
-6.3	16.8	12.3	12
9.6	15.3	25.8	14
0.4	7.1	52.1	53.1

im2col



Indirect

FFT



[Chellapilla et al. 2006; Mathieu et al. 2014; Lavin & Gray 2016; Liu et al. 2017]

Operator implementations: convolution

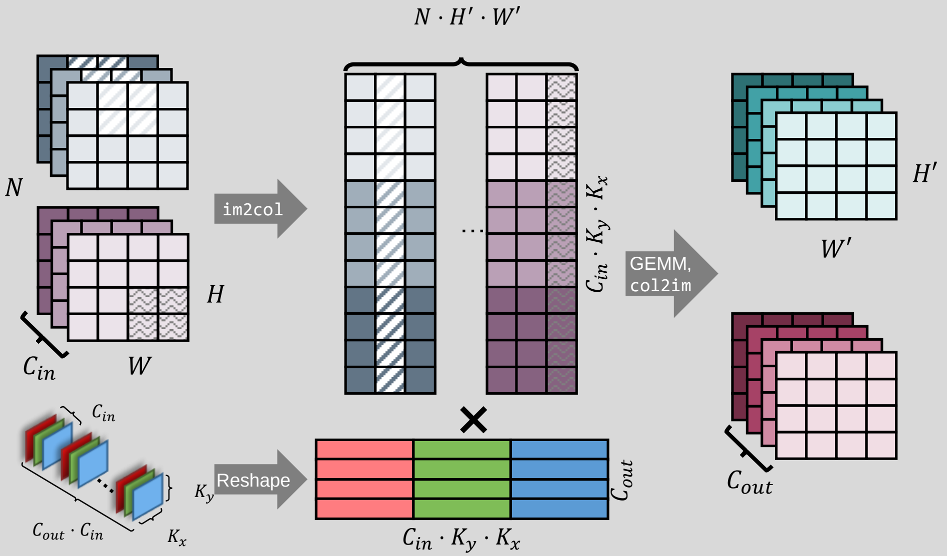
$$Y_{k,f,i,j} = \sum_{c=0}^{C-1} \sum_{a=-O}^{O} \sum_{b=-O}^{O} X_{k,c,i+a,j+b} w_{f,c,a+O,b+O}$$

Direct

Direct

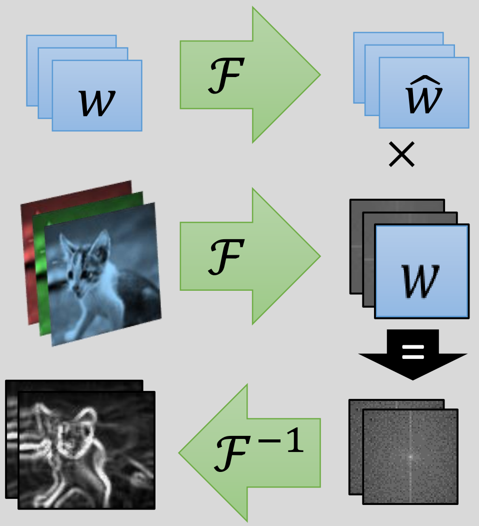
$$\begin{bmatrix} 4 & 1 & 9 & 8 \\ 5 & 9 & 9 & 8 \\ 0 & 7 & 3 & 4 \\ 2 & 6 & 3 & 1 \end{bmatrix} * \begin{bmatrix} 1 & -1 & 0 \\ 0.1 & -2 & 0 \\ 3 & 4 & 1.1 \end{bmatrix} = \begin{bmatrix} 21.9 & 59.3 & 53.9 & 43.9 \\ -6.3 & 16.8 & 12.3 & 12 \\ 9.6 & 15.3 & 25.8 & 14 \\ 0.4 & 7.1 & 52.1 & 53.1 \end{bmatrix}$$

im2col



Indirect

FFT



Winograd

[Chellapilla et al. 2006; Mathieu et al. 2014; Lavin & Gray 2016; Liu et al. 2017]

Operator implementations: convolution

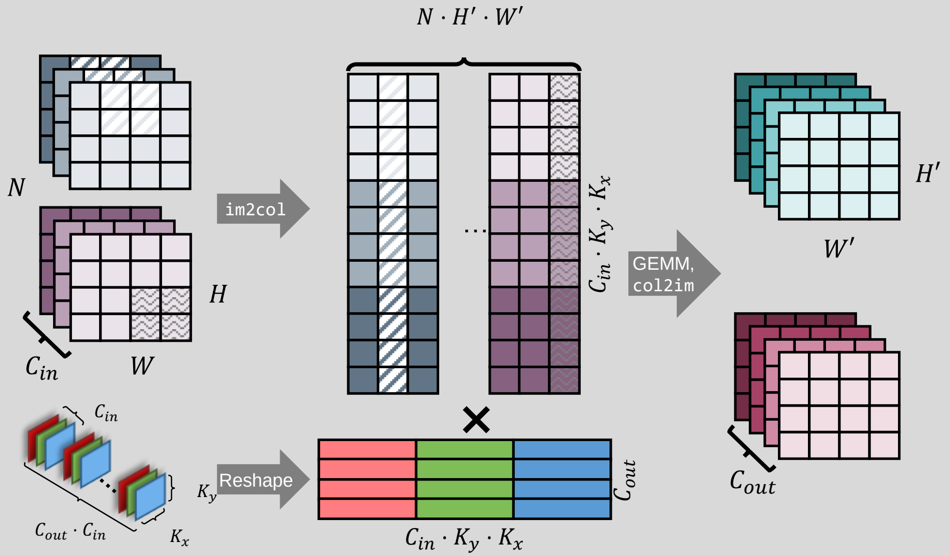
$$Y_{k,f,i,j} = \sum_{c=0}^{C-1} \sum_{a=-O}^{O} \sum_{b=-O}^{O} X_{k,c,i+a,j+b} w_{f,c,a+O,b+O}$$

Direct

Direct

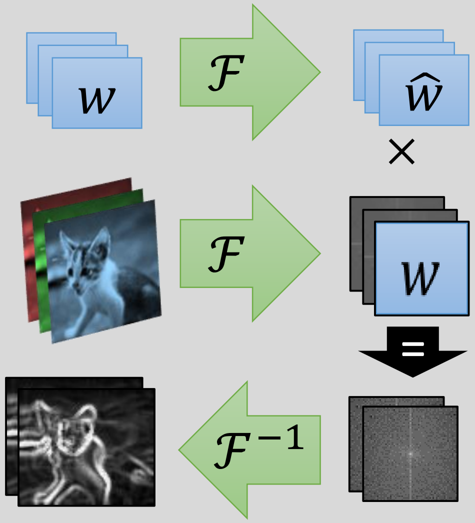
$$\begin{bmatrix} 4 & 1 & 9 & 8 \\ 5 & 9 & 9 & 8 \\ 0 & 7 & 3 & 4 \\ 2 & 6 & 3 & 1 \end{bmatrix} * \begin{bmatrix} 1 & -1 & 0 \\ 0.1 & -2 & 0 \\ 3 & 4 & 1.1 \end{bmatrix} = \begin{bmatrix} 21.9 & 59.3 & 53.9 & 43.9 \\ -6.3 & 16.8 & 12.3 & 12 \\ 9.6 & 15.3 & 25.8 & 14 \\ 0.4 & 7.1 & 52.1 & 53.1 \end{bmatrix}$$

im2col



Indirect

FFT



Winograd

Operator implementations: convolution

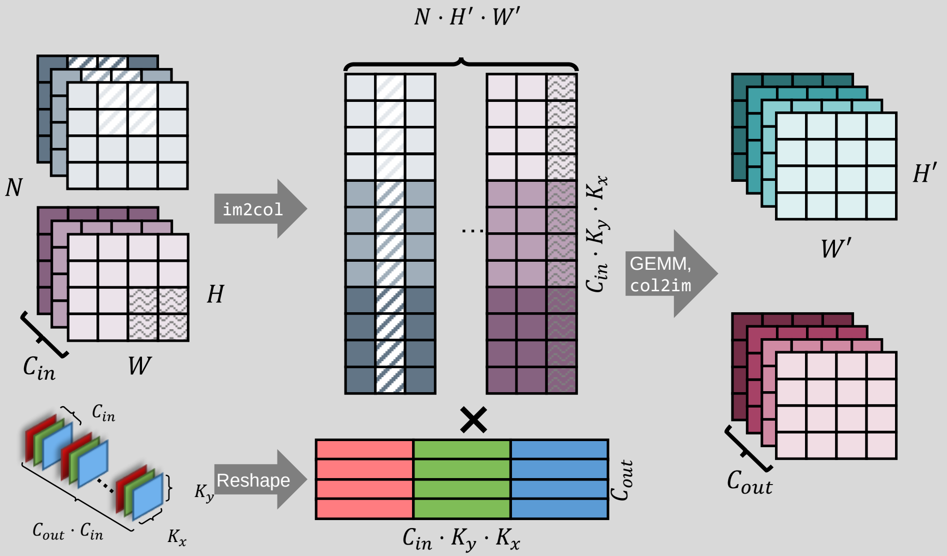
$$Y_{k,f,i,j} = \sum_{c=0}^{C-1} \sum_{a=-O}^{O} \sum_{b=-O}^{O} X_{k,c,i+a,j+b} w_{f,c,a+O,b+O}$$

Direct

Direct

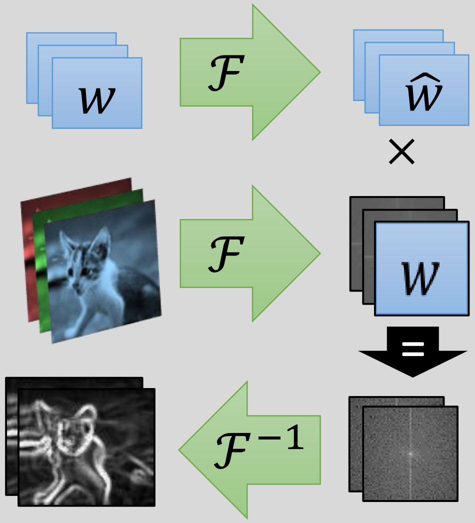
$$\begin{bmatrix} 4 & 1 & 9 & 8 \\ 5 & 9 & 9 & 8 \\ 0 & 7 & 3 & 4 \\ 2 & 6 & 3 & 1 \end{bmatrix} * \begin{bmatrix} 1 & -1 & 0 \\ 0.1 & -2 & 0 \\ 3 & 4 & 1.1 \end{bmatrix} = \begin{bmatrix} 21.9 & 59.3 & 53.9 & 43.9 \\ -6.3 & 16.8 & 12.3 & 12 \\ 9.6 & 15.3 & 25.8 & 14 \\ 0.4 & 7.1 & 52.1 & 53.1 \end{bmatrix}$$

im2col



Indirect

FFT



Winograd

[Chellapilla et al. 2006; Mathieu et al. 2014; Lavin & Gray 2016; Liu et al. 2017]

Operator implementations: convolution

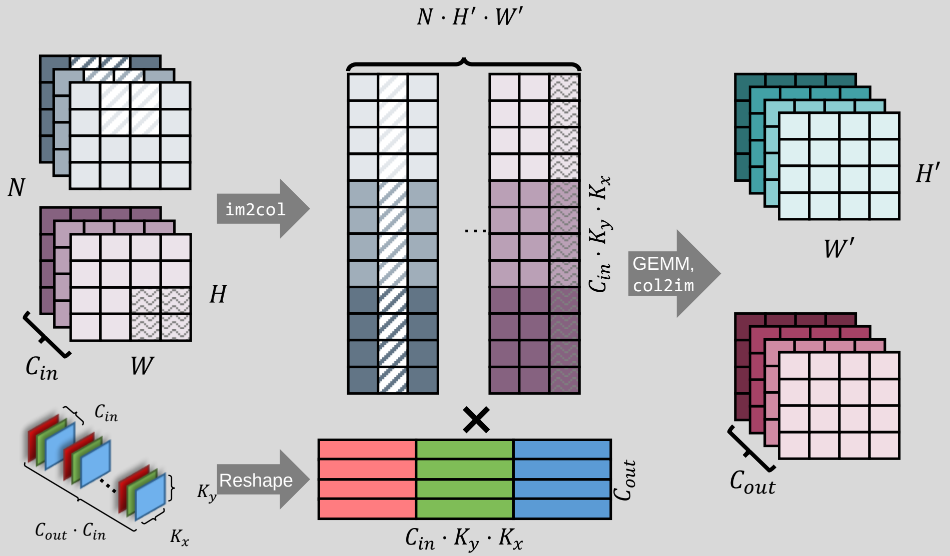
$$Y_{k,f,i,j} = \sum_{c=0}^{C-1} \sum_{a=-O}^{O} \sum_{b=-O}^{O} X_{k,c,i+a,j+b} w_{f,c,a+O,b+O}$$

Direct

Direct

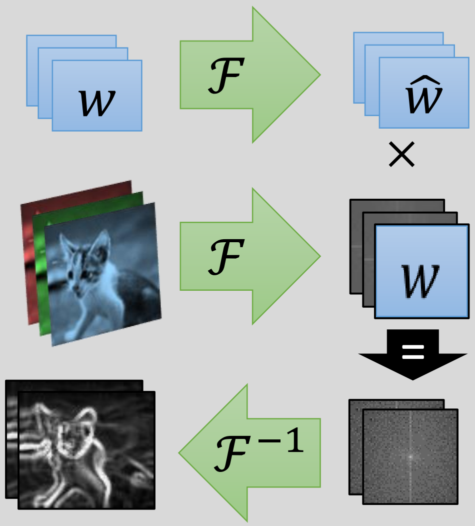
$$\begin{bmatrix} 4 & 1 & 9 & 8 \\ 5 & 9 & 9 & 8 \\ 0 & 7 & 3 & 4 \\ 2 & 6 & 3 & 1 \end{bmatrix} * \begin{bmatrix} 1 & -1 & 0 \\ 0.1 & -2 & 0 \\ 3 & 4 & 1.1 \end{bmatrix} = \begin{bmatrix} 21.9 & 59.3 & 53.9 & 43.9 \\ -6.3 & 16.8 & 12.3 & 12 \\ 9.6 & 15.3 & 25.8 & 14 \\ 0.4 & 7.1 & 52.1 & 53.1 \end{bmatrix}$$

im2col



Indirect

FFT



Winograd

[Chellapilla et al. 2006; Mathieu et al. 2014; Lavin & Gray 2016; Liu et al. 2017]

Operator implementations: convolution

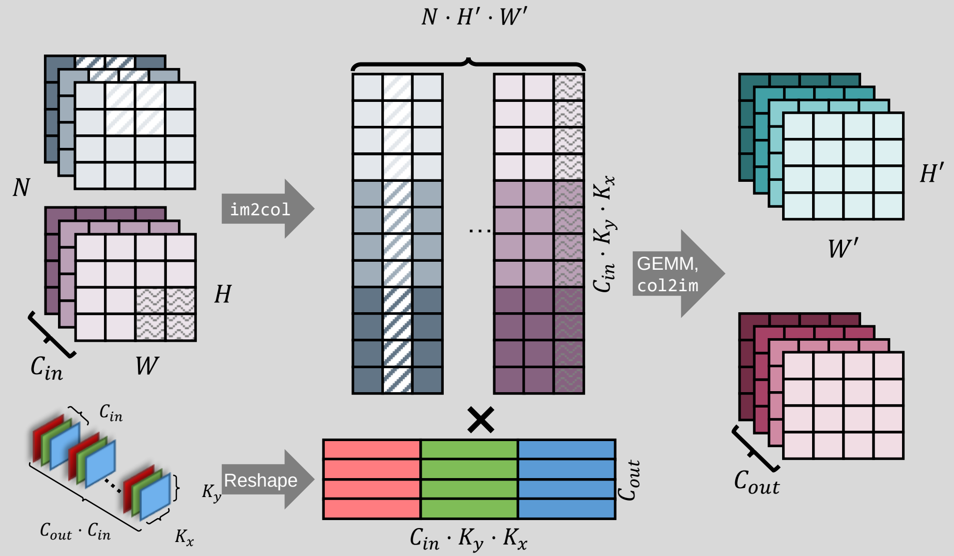
$$Y_{k,f,i,j} = \sum_{c=0}^{C-1} \sum_{a=-O}^{O} \sum_{b=-O}^{O} X_{k,c,i+a,j+b} w_{f,c,a+O,b+O}$$

Direct

Direct

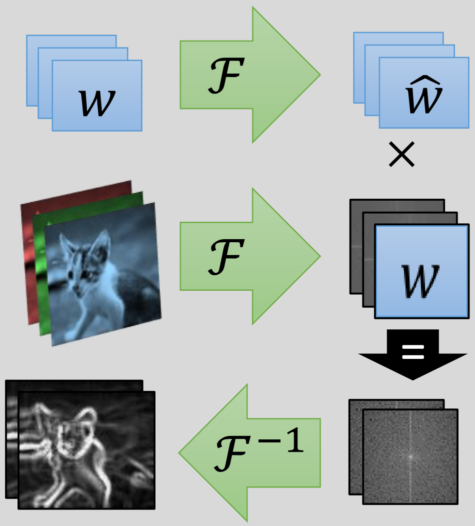
$$\begin{bmatrix} 4 & 1 & 9 & 8 \\ 5 & 9 & 9 & 8 \\ 0 & 7 & 3 & 4 \\ 2 & 6 & 3 & 1 \end{bmatrix} * \begin{bmatrix} 1 & -1 & 0 \\ 0.1 & -2 & 0 \\ 3 & 4 & 1.1 \end{bmatrix} = \begin{bmatrix} 21.9 & 59.3 & 53.9 & 43.9 \\ -6.3 & 16.8 & 12.3 & 12 \\ 9.6 & 15.3 & 25.8 & 14 \\ 0.4 & 7.1 & 52.1 & 53.1 \end{bmatrix}$$

im2col

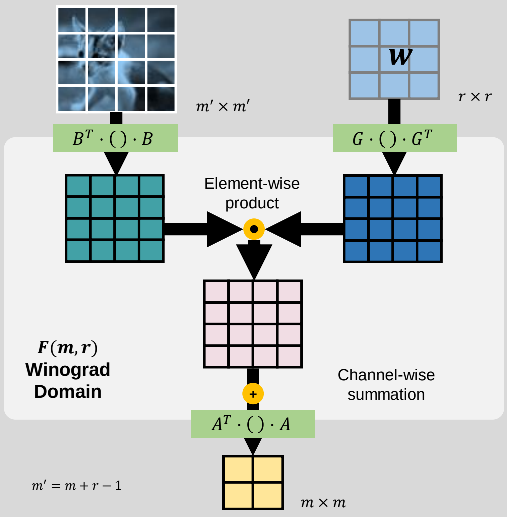


Indirect

FFT



Winograd



[Chellapilla et al. 2006; Mathieu et al. 2014; Lavin & Gray 2016; Liu et al. 2017]

Microbatching (μ-cuDNN) – how to implement layers best in practice?

[Oyama et al. 2018]

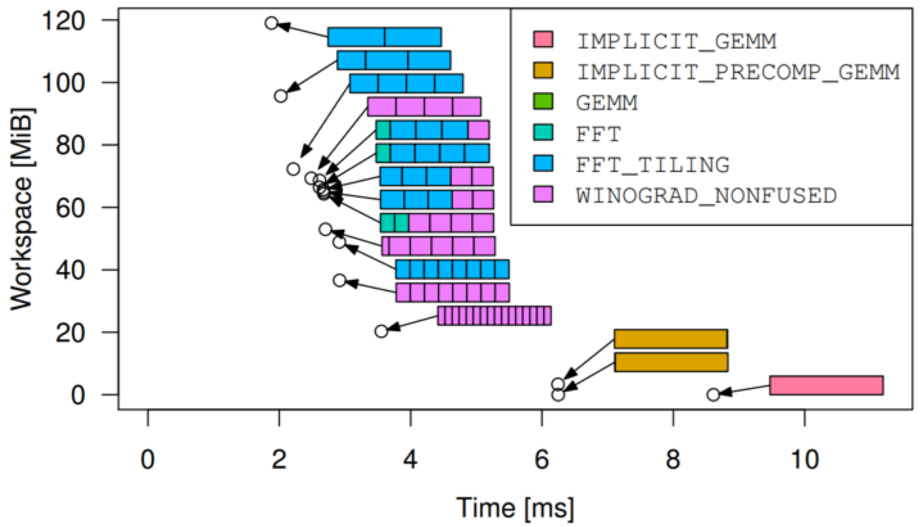
Microbatching (μ-cuDNN) – how to implement layers best in practice?

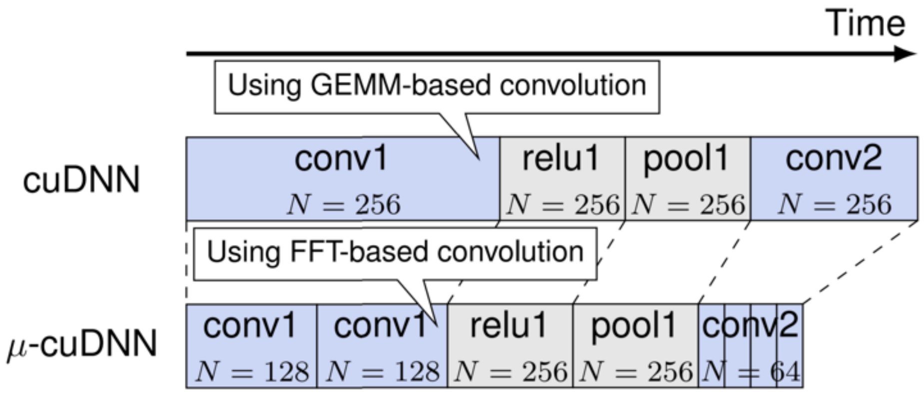
- In cuDNN there are ~16 convolution implementations
- Performance depends on temporary memory (workspace) size
- Key idea: segment minibatch into microbatches, reuse workspace, use different algorithms
- How to choose microbatch sizes and algorithms?

[Oyama et al. 2018]

Microbatching (μ-cuDNN) – how to implement layers best in practice?

- In cuDNN there are ~16 convolution implementations
- Performance depends on temporary memory (workspace) size
- Key idea: segment minibatch into microbatches, reuse workspace, use different algorithms
- How to choose microbatch sizes and algorithms?

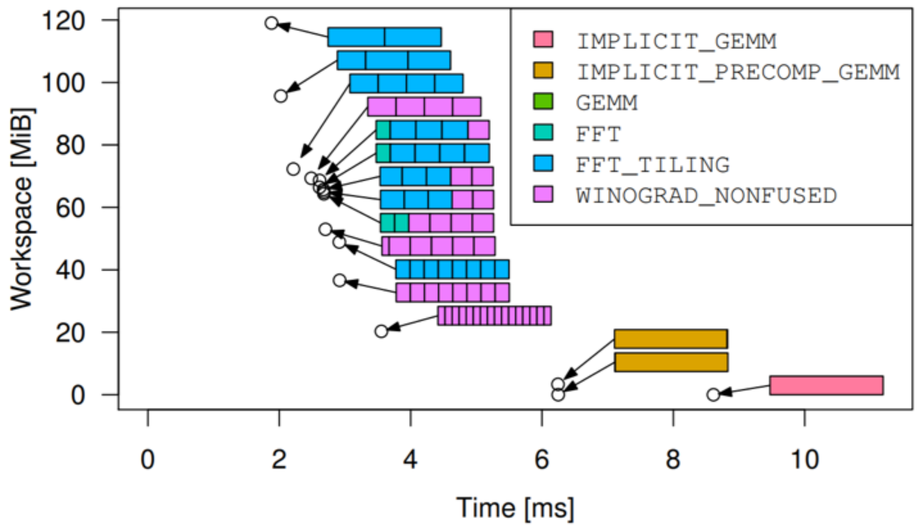


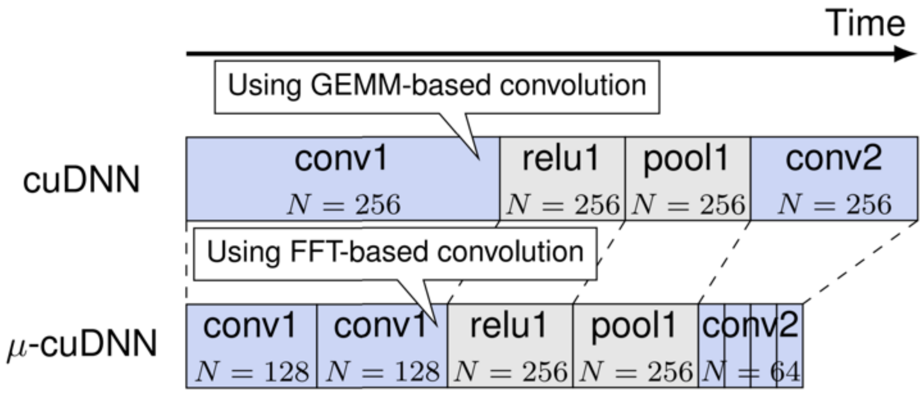


[Oyama et al. 2018]

Microbatching (μ-cuDNN) – how to implement layers best in practice?

- In cuDNN there are ~16 convolution implementations
- Performance depends on temporary memory (workspace) size
- Key idea: segment minibatch into microbatches, reuse workspace, use different algorithms
- How to choose microbatch sizes and algorithms?



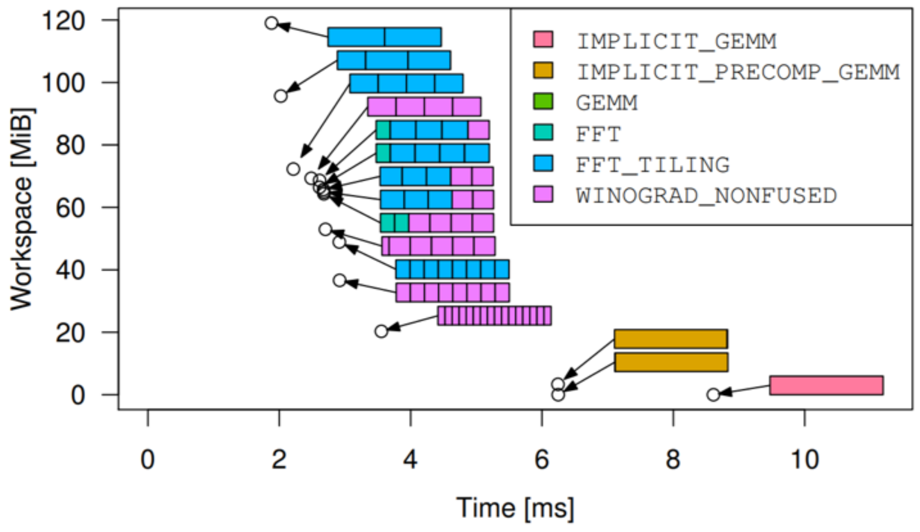


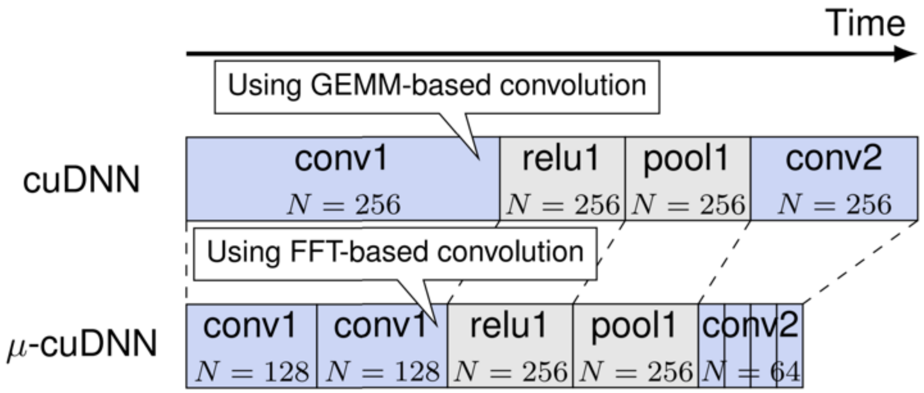
[Oyama et al. 2018]

Microbatching (μ-cuDNN) – how to implement layers best in practice?

- In cuDNN there are ~16 convolution implementations
- Performance depends on temporary memory (workspace) size
- Key idea: segment minibatch into microbatches, reuse workspace, use different algorithms
- How to choose microbatch sizes and algorithms?

$$T(b) = \min \left\{ \begin{array}{l} T_\mu(b), \\ \min_{b'=1,2,\ldots,b-1} T(b') + T(b-b') \end{array} \right\}$$





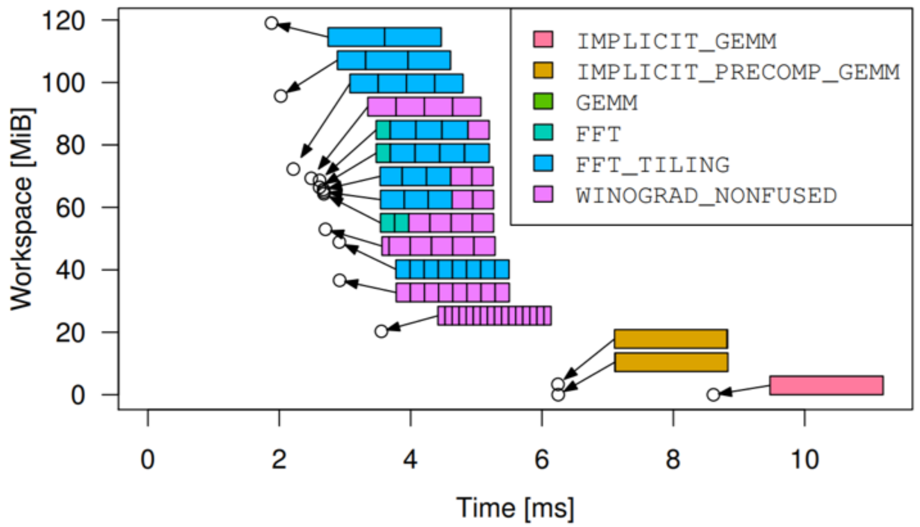
[Oyama et al. 2018]

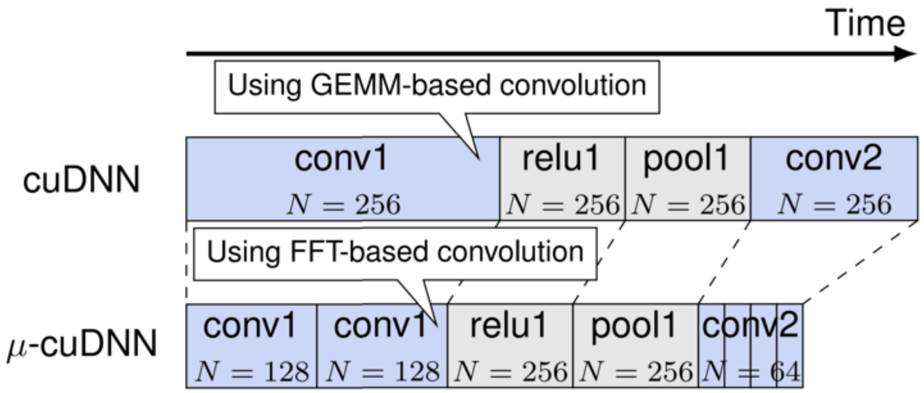
Microbatching (μ-cuDNN) – how to implement layers best in practice?

- In cuDNN there are ~16 convolution implementations
- Performance depends on temporary memory (workspace) size
- Key idea: segment minibatch into microbatches, reuse workspace, use different algorithms
- How to choose microbatch sizes and algorithms?

$$T(b) = \min \left\{ \begin{array}{l} T_\mu(b), \\ \min_{b'=1,2,\ldots,b-1} T(b') + T(b-b') \end{array} \right\}$$

Dynamic Programming (Space Reuse)





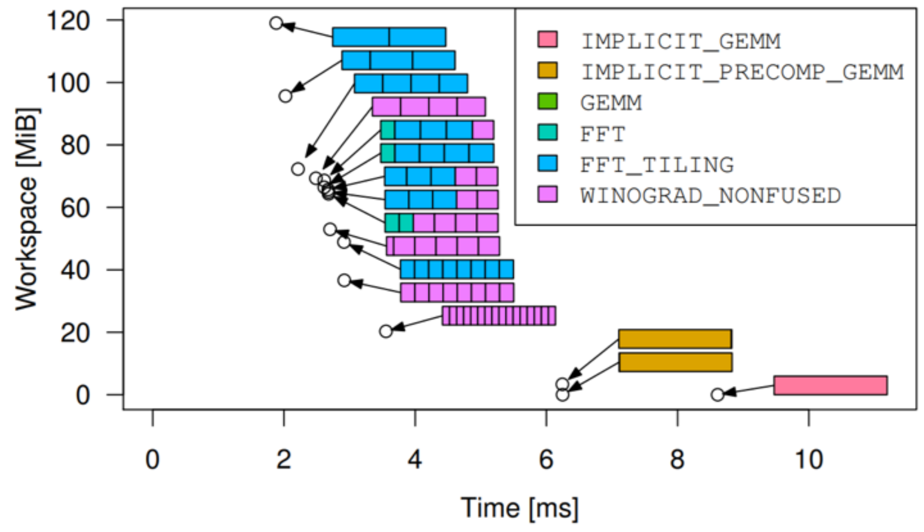
[Oyama et al. 2018]

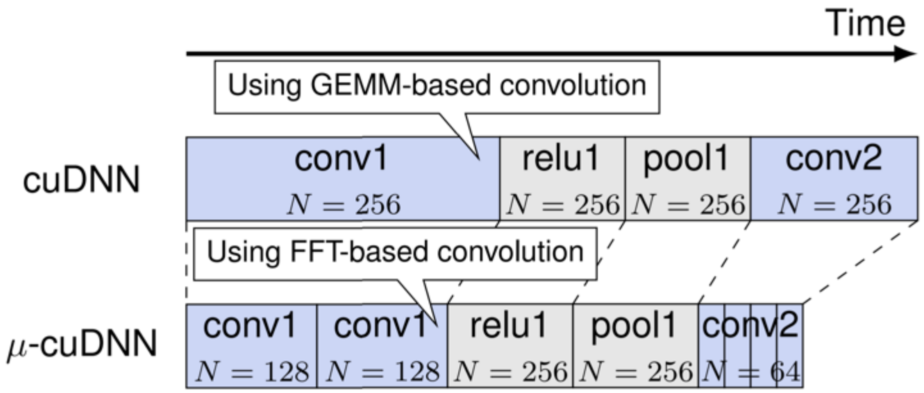
Microbatching (μ-cuDNN) – how to implement layers best in practice?

- In cuDNN there are ~16 convolution implementations
- Performance depends on temporary memory (workspace) size
- Key idea: segment minibatch into microbatches, reuse workspace, use different algorithms
- How to choose microbatch sizes and algorithms?

$$T(b) = \min \left\{ \begin{array}{l} T_\mu(b), \\ \min_{b'=1,2,\ldots,b-1} T(b') + T(b-b') \end{array} \right\}$$

Dynamic Programming (Space Reuse)





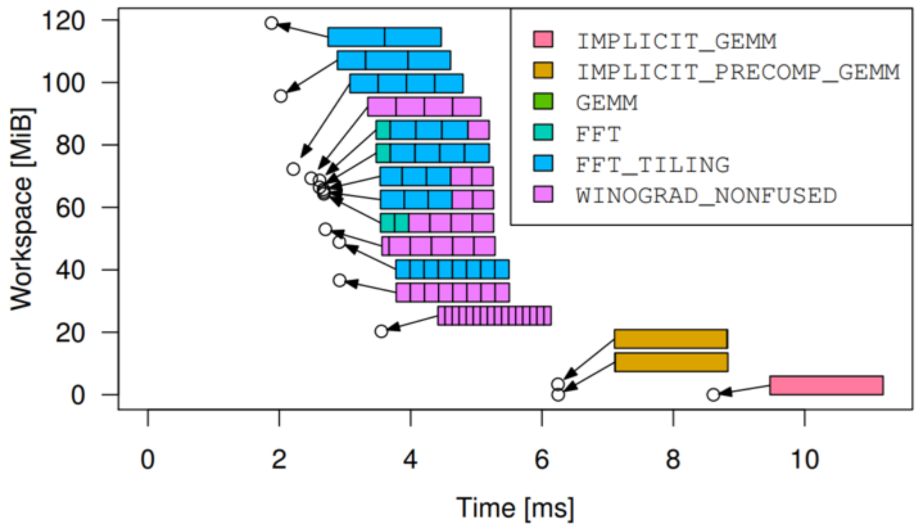
[Oyama et al. 2018]

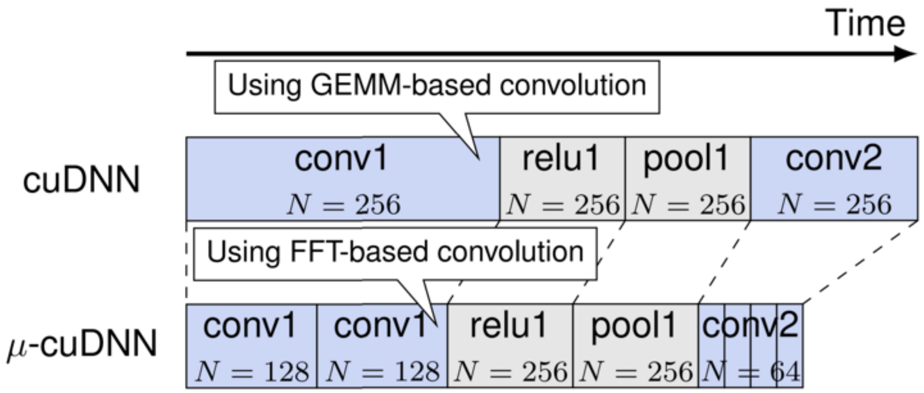
Microbatching (μ-cuDNN) – how to implement layers best in practice?

- **In cuDNN there are ~16 convolution implementations**
- **Performance depends on temporary memory (workspace) size**
- **Key idea: segment minibatch into microbatches, reuse workspace, use different algorithms**
- **How to choose microbatch sizes and algorithms?**

$$T(b) = \min \left\{ \begin{array}{l} T_\mu(b), \\ \min_{b'=1,2,\dots,b-1} T(b') + T(b-b') \end{array} \right\}$$

Dynamic Programming (Space Reuse)





[Oyama et al. 2018]

Microbatching (μ-cuDNN) – how to implement layers best in practice?

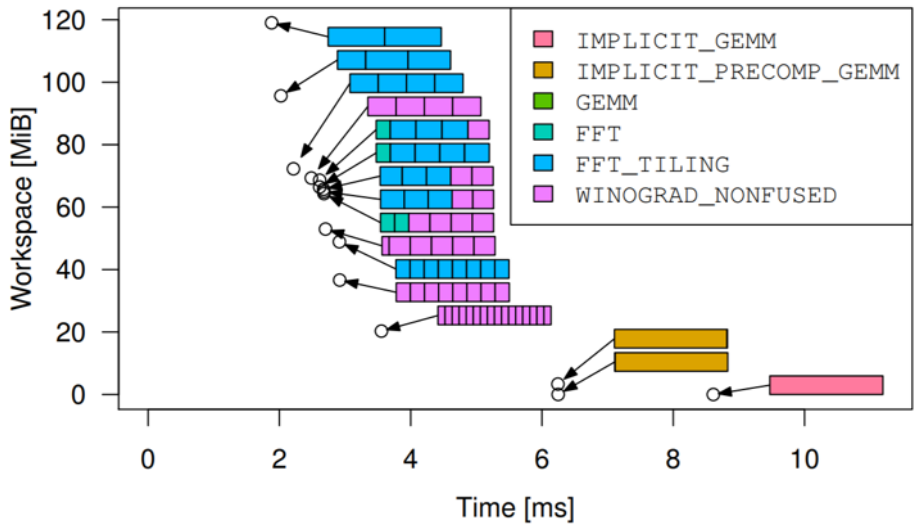
- **In cuDNN there are ~16 convolution implementations**
- **Performance depends on temporary memory (workspace) size**
- **Key idea: segment minibatch into microbatches, reuse workspace, use different algorithms**
- **How to choose microbatch sizes and algorithms?**

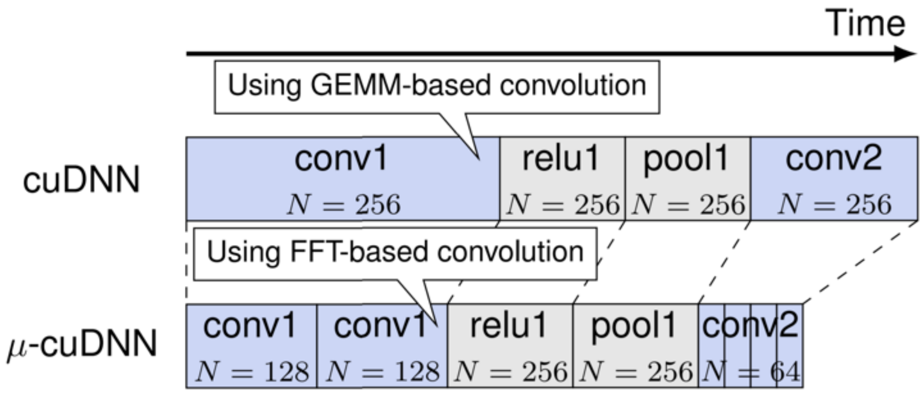
$$T(b) = \min \left\{ \begin{array}{l} T_\mu(b), \\ \min_{b'=1,2,\ldots,b-1} T(b') + T(b-b') \end{array} \right\}$$

Dynamic Programming (Space Reuse)

$$\begin{array}{ll} \min & T = \sum_{k\in\mathcal{K}} \sum_{c\in C_k} T_k(c) x_{k,c} \\ \text{subject to} & \sum_{k\in\mathcal{K}} \sum_{c\in C_k} M_k(c) x_{k,c} \le M \\ & \sum_{c\in C_k} x_{k,c} = 1 \ (\forall k \in \mathcal{K}) \\ & x_{k,c} \in \{0,1\} \ (\forall k \in \mathcal{K}, \forall c \in C_k) \end{array}$$

Integer Linear Programming (Space Sharing)





[Oyama et al. 2018]

Microbatching (μ-cuDNN) – how to implement layers best in practice?

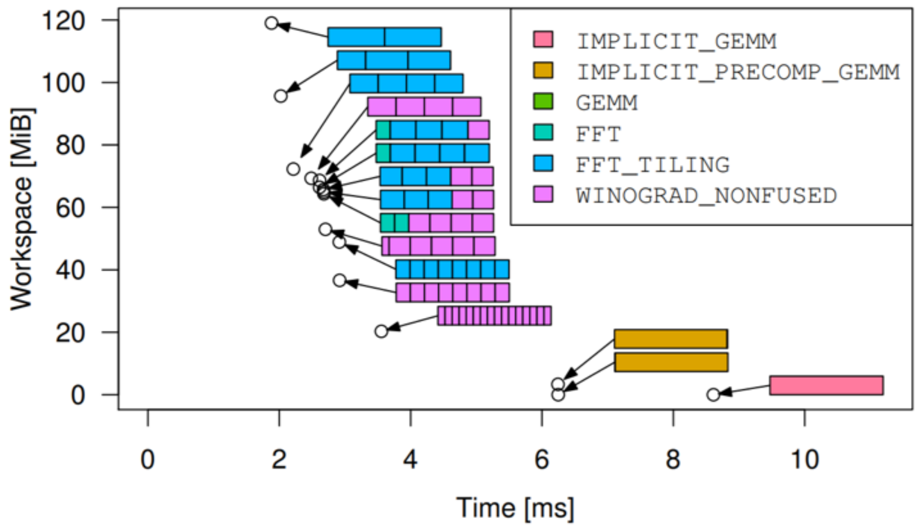
- In cuDNN there are ~16 convolution implementations
- Performance depends on temporary memory (workspace) size
- Key idea: segment minibatch into microbatches, reuse workspace, use different algorithms
- How to choose microbatch sizes and algorithms?

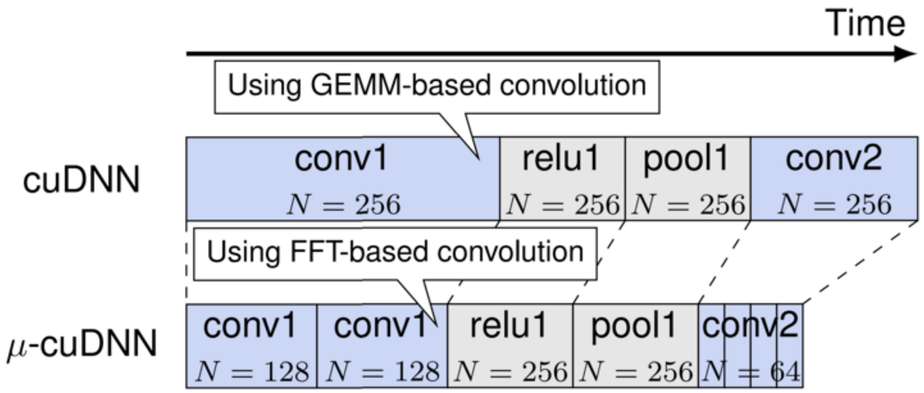
$$T(b) = \min \left\{ \begin{array}{l} T_\mu(b), \\ \min_{b'=1,2,\ldots,b-1} T(b') + T(b-b') \end{array} \right\}$$

Dynamic Programming (Space Reuse)

$$\begin{array}{ll} \min & T = \sum_{k \in \mathcal{K}} \sum_{c \in C_k} T_k(c) x_{k,c} \\ \text{subject to} & \sum_{k \in \mathcal{K}} \sum_{c \in C_k} M_k(c) x_{k,c} \leq M \\ & \sum_{c \in C_k} x_{k,c} = 1 \ (\forall k \in \mathcal{K}) \\ & x_{k,c} \in \{0,1\} \ (\forall k \in \mathcal{K}, \forall c \in C_k) \end{array}$$

Integer Linear Programming (Space Sharing)





[Oyama et al. 2018]

Microbatching (μ-cuDNN) – how to implement layers best in practice?

- In cuDNN there are ~16 convolution implementations
- Performance depends on temporary memory (workspace) size
- Key idea: segment minibatch into microbatches, reuse workspace, use different algorithms
- How to choose microbatch sizes and algorithms?

$$T(b) = \min \left\{ \begin{array}{l} T_\mu(b), \\ \min_{b'=1,2,\ldots,b-1} T(b') + T(b-b') \end{array} \right\}$$

Dynamic Programming (Space Reuse)

$$\begin{aligned} \min \quad & T = \sum_{k\in\mathcal{K}} \sum_{c\in C_k} T_k(c) x_{k,c} \\ \text{subject to} \quad & \sum_{k\in\mathcal{K}} \sum_{c\in C_k} M_k(c) x_{k,c} \leq M \\ & \sum_{c\in C_k} x_{k,c} = 1 \ (\forall k \in \mathcal{K}) \\ & x_{k,c} \in \{0,1\} \ (\forall k \in \mathcal{K}, \forall c \in C_k) \end{aligned}$$

Integer Linear Programming (Space Sharing)

[Oyama et al. 2018]

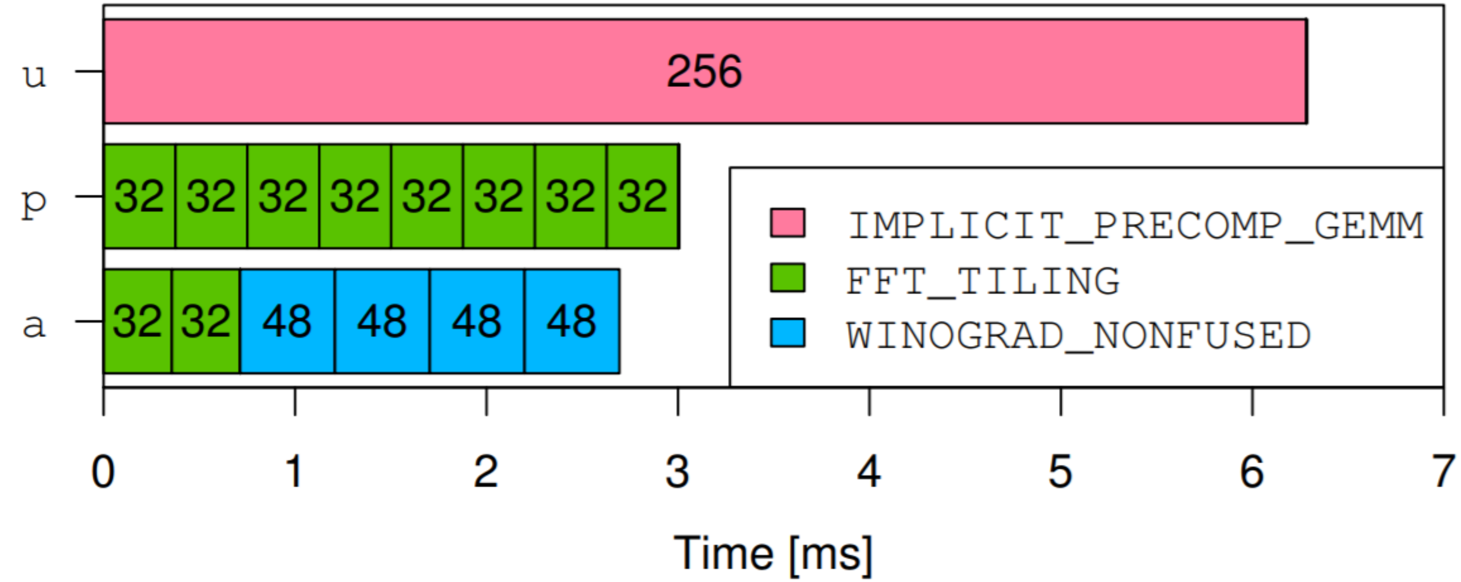
Microbatching (μ-cuDNN) – how to implement layers best in practice?

- In cuDNN there are ~16 convolution implementations
- Performance depends on temporary memory (workspace) size
- Key idea: segment minibatch into microbatches, reuse workspace, use different algorithms
- How to choose microl

Dynamic Program

Integer Linear Programming (Space Sharing)

Fast (up to 4.54x faster on DeepBench)



[Oyama et al. 2018]

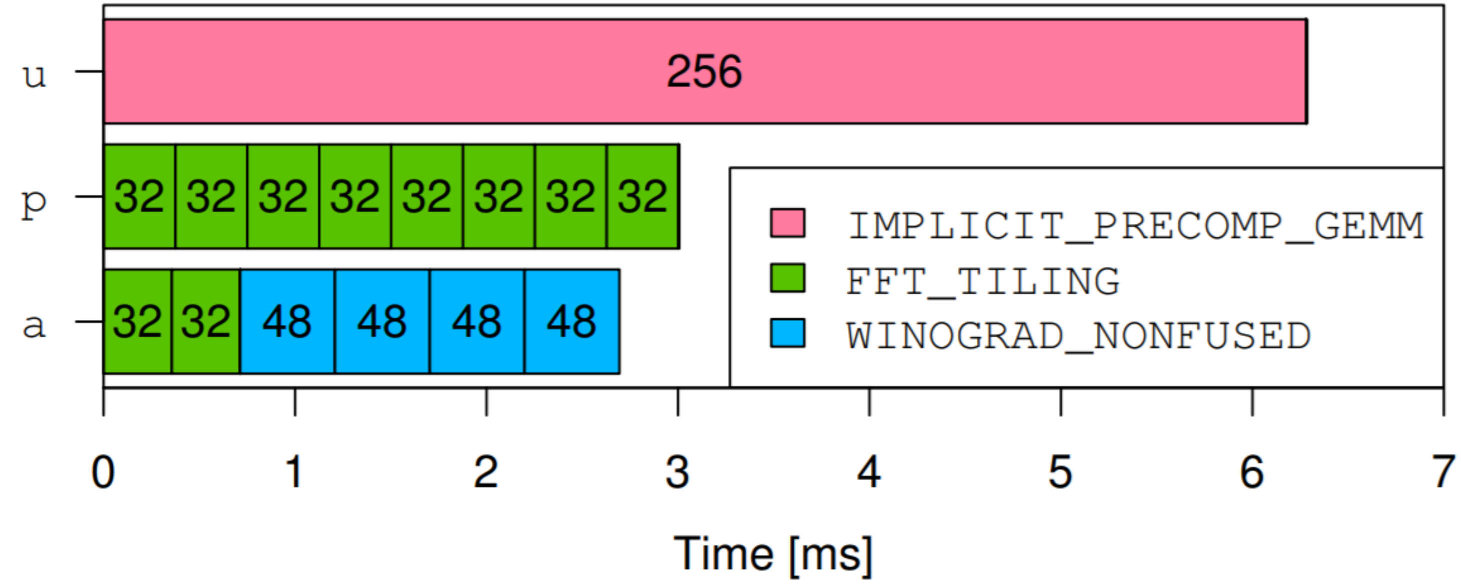
Microbatching (μ-cuDNN) – how to implement layers best in practice?

- In cuDNN there are ~16 convolution implementations
- Performance depends on temporary memory (workspace) size
- Key idea: segment minibatch into microbatches, reuse workspace, use different algorithms
- How to choose microl

Dynamic Program

Integer Linear Programming (Space Sharing)

Fast (up to 4.54x faster on DeepBench)



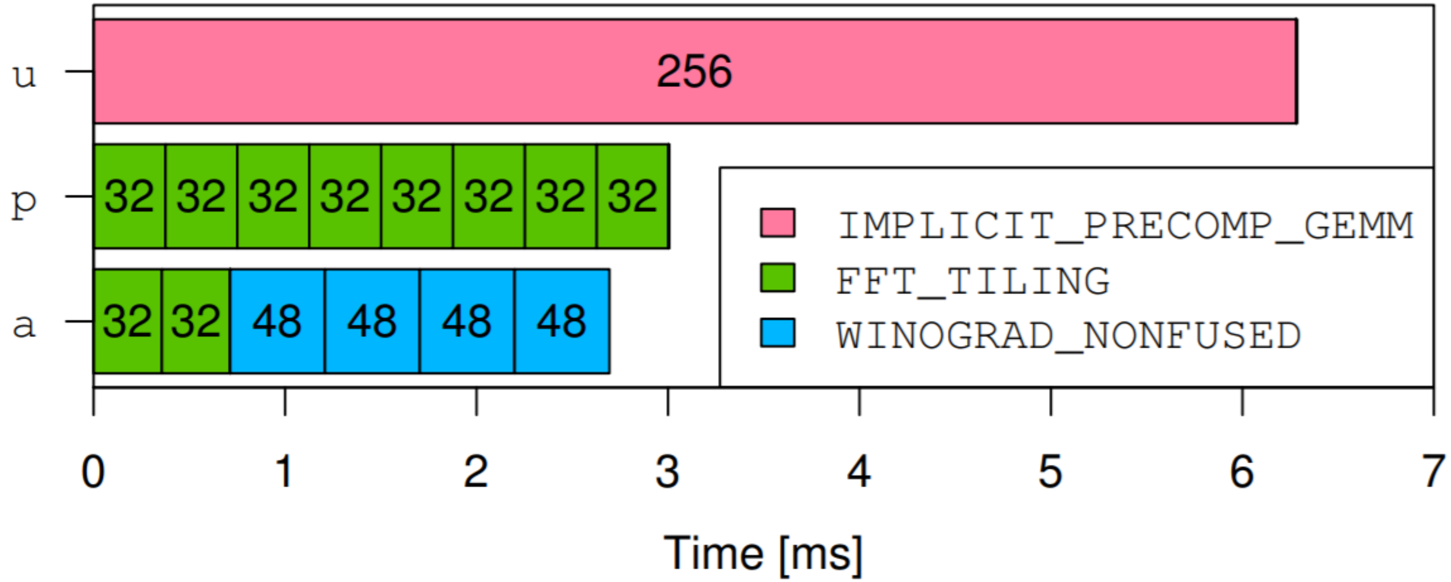
[Oyama et al. 2018]

Microbatching (μ-cuDNN) – how to implement layers best in practice?

- In cuDNN there are ~16 convolution implementations
- Performance depends on temporary memory (workspace) size
- Key idea: segment minibatch into microbatches, reuse workspace, use different algorithms
- How to choose microl

Microbatching Strategy

Fast (up to 4.54x faster on DeepBench)



Dynamic Program

Integer Linear Programming (Space Sharing)

[Oyama et al. 2018]

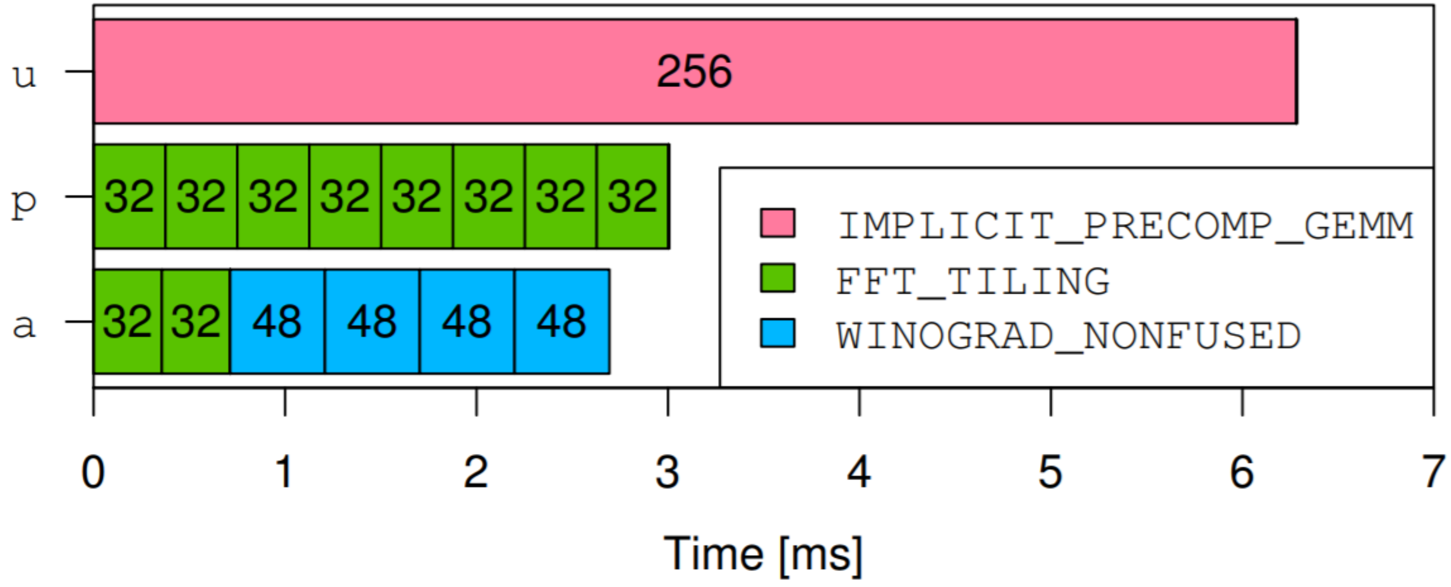
Microbatching (μ-cuDNN) – how to implement layers best in practice?

- In cuDNN there are ~16 convolution implementations
- Performance depends on temporary memory (workspace) size
- Key idea: segment minibatch into microbatches, reuse workspace, use different algorithms
- How to choose microl

Microbatching Strategy

Fast (up to 4.54x faster on DeepBench)

none (undivided)



Dynamic Program

Integer Linear Programming (Space Sharing)

[Oyama et al. 2018]

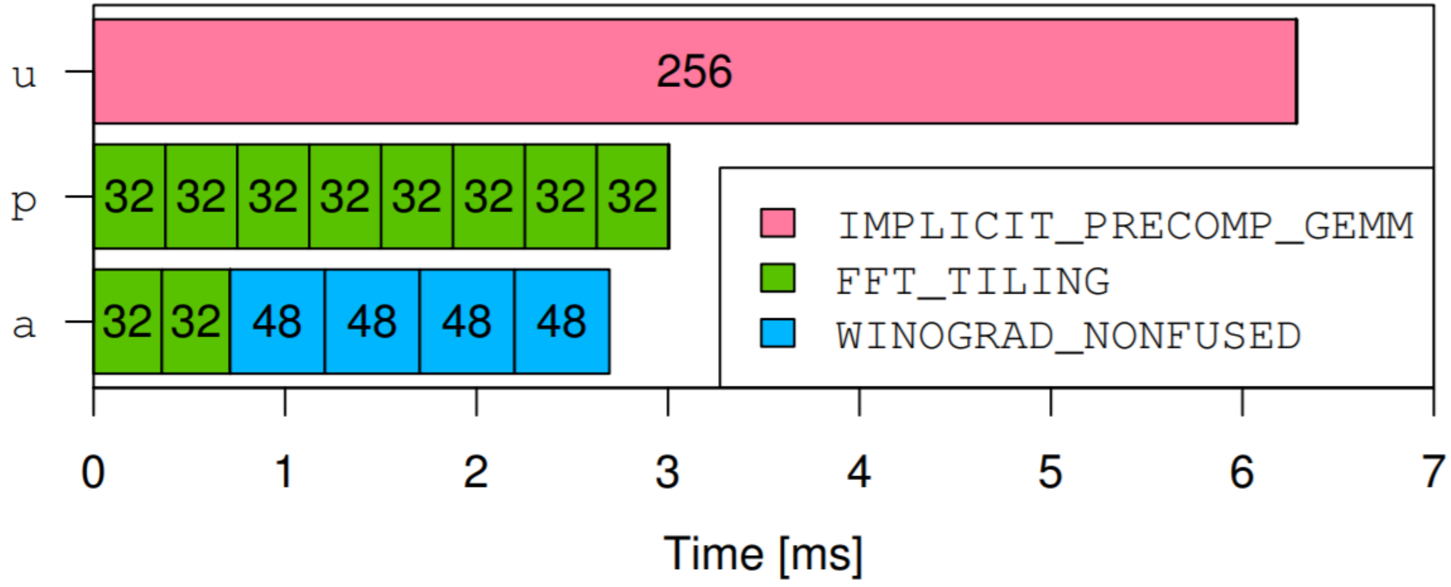
Microbatching (μ-cuDNN) – how to implement layers best in practice?

Fast (up to 4.54x faster on DeepBench)

Microbatching Strategy

none (undivided)

powers-of-two only



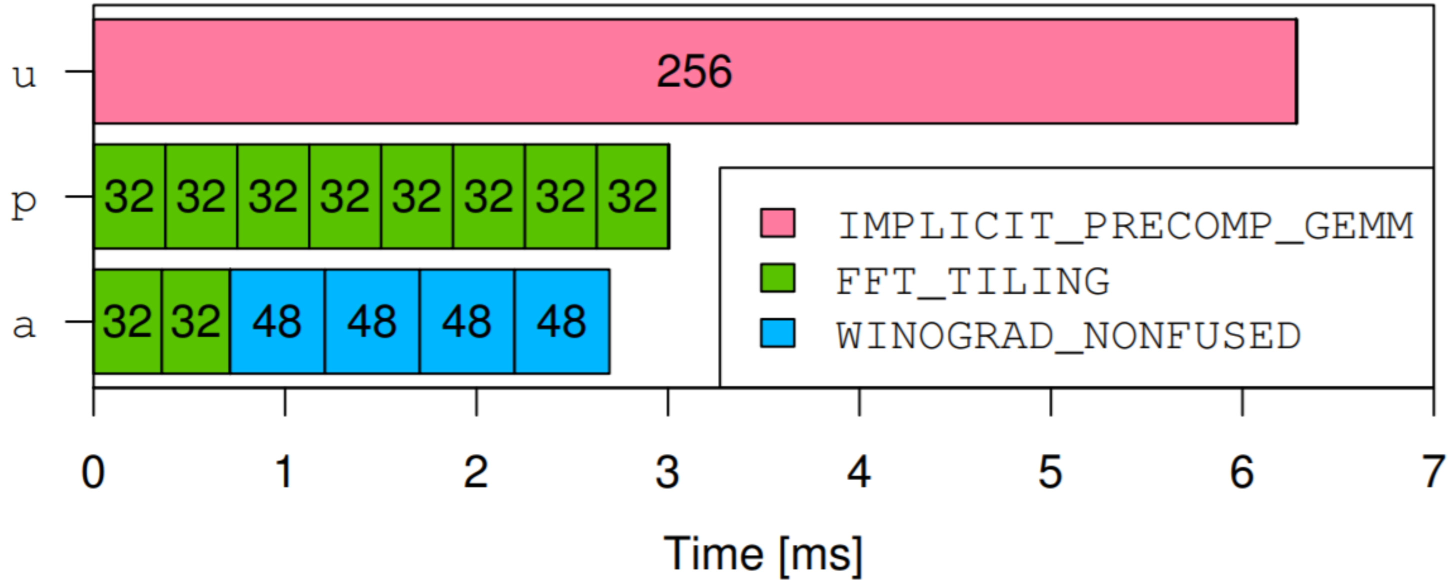
[Oyama et al. 2018]

Microbatching (μ-cuDNN) – how to implement layers best in practice?

Microbatching Strategy

Fast (up to 4.54x faster on DeepBench)

- none (undivided)
- powers-of-two only
- any (unrestricted)

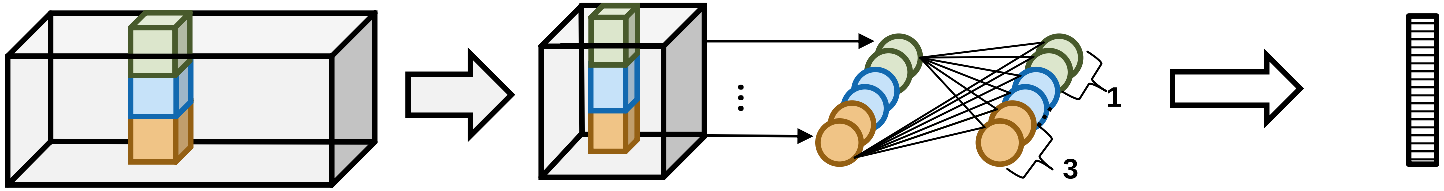


[Oyama et al. 2018]

Model parallelism – limited by network size

[Forrest et al. 1987]

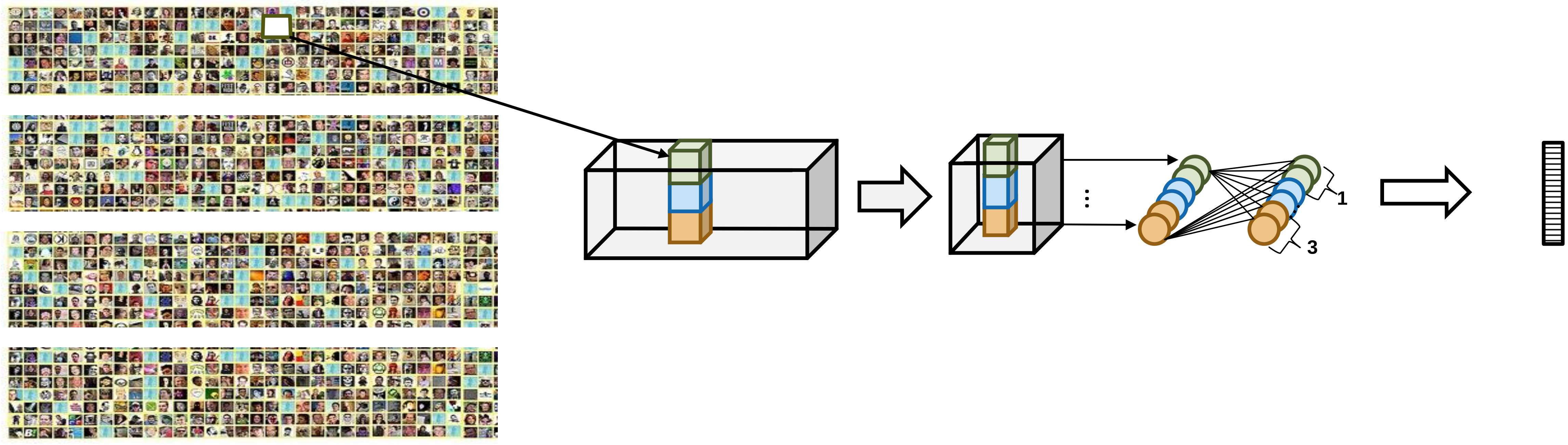
Model parallelism – limited by network size



- Parameters can be distributed across processors
- Mini-batch has to be copied to all processors
- Backpropagation requires complex communication every layer

[Forrest et al. 1987]

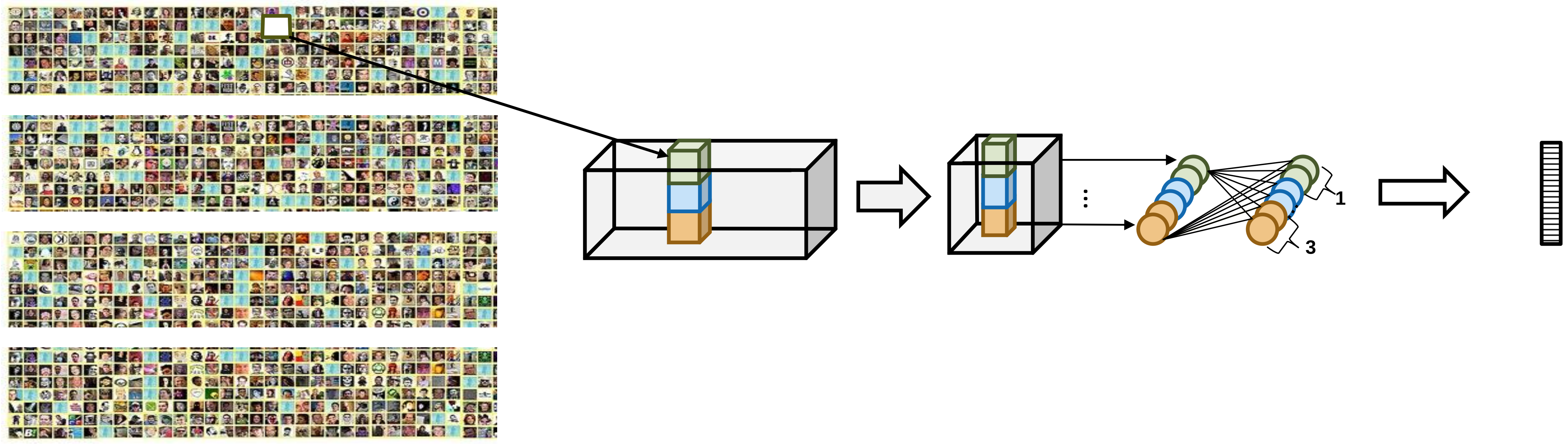
Model parallelism – limited by network size



- Parameters can be distributed across processors
- Mini-batch has to be copied to all processors
- Backpropagation requires complex communication every layer

[Forrest et al. 1987]

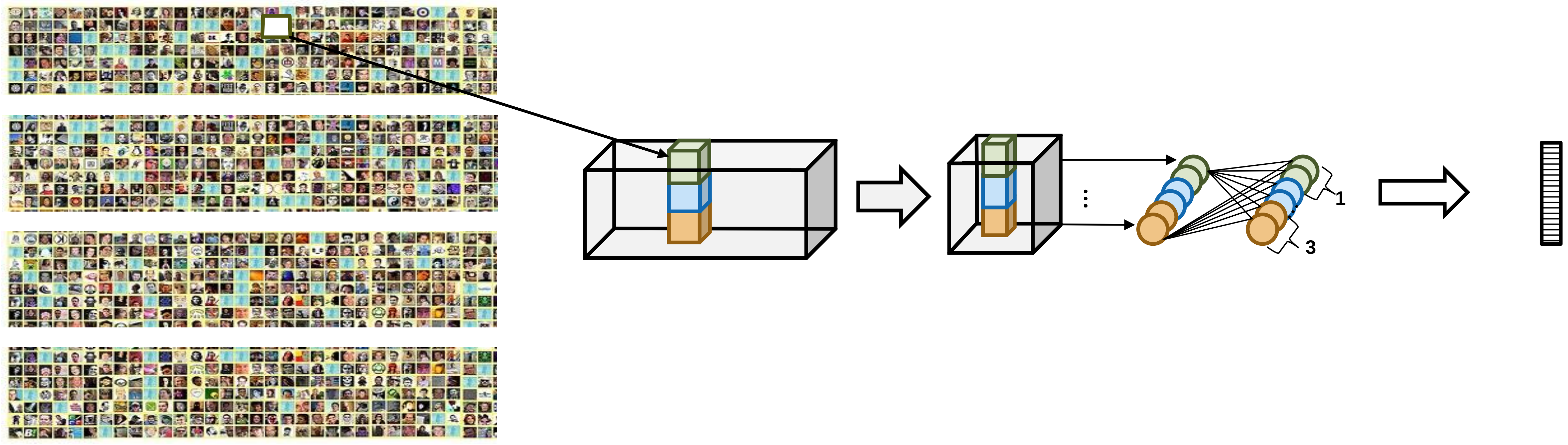
Model parallelism – limited by network size



- Parameters can be distributed across processors
- Mini-batch has to be copied to all processors
- Backpropagation requires complex communication every layer

[Forrest et al. 1987]

Model parallelism – limited by network size



- Parameters can be distributed across processors
- Mini-batch has to be copied to all processors
- Backpropagation requires complex communication every layer

[Forrest et al. 1987]

Pipeline parallelism – limited by network size

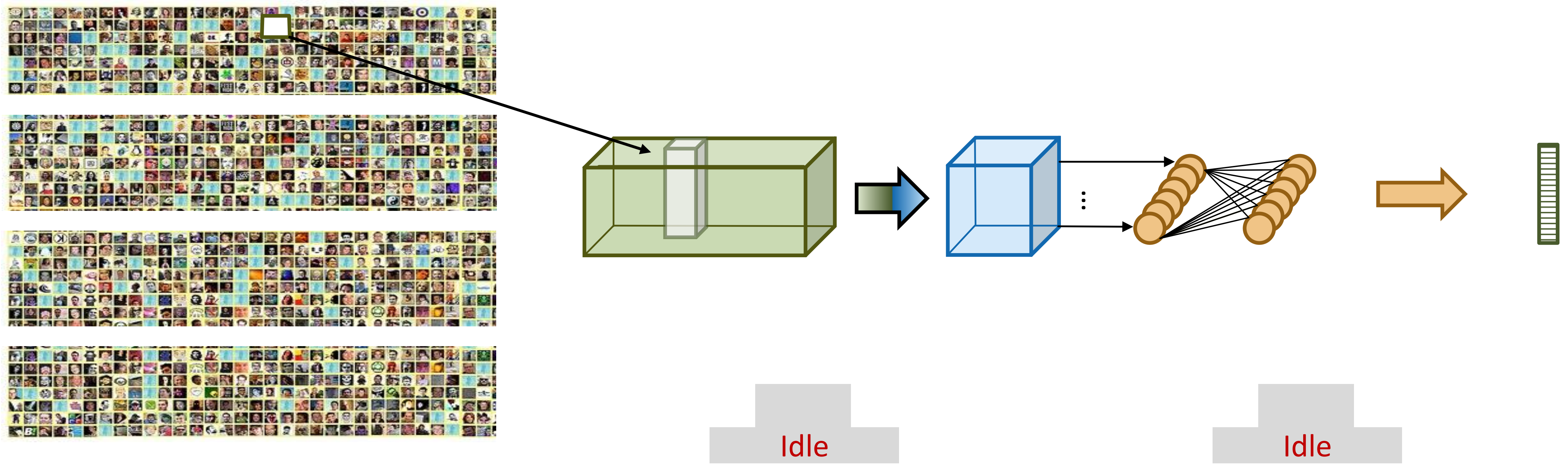
[Blelloch & Rosenberg 1987]

Pipeline parallelism – limited by network size

- Layers/parameters can be distributed across processors
- Sparse communication pattern (only pipeline stages)
- Mini-batch has to be copied through all processors
- Consistent model introduces idle-time "bubble"

[Blelloch & Rosenberg 1987]

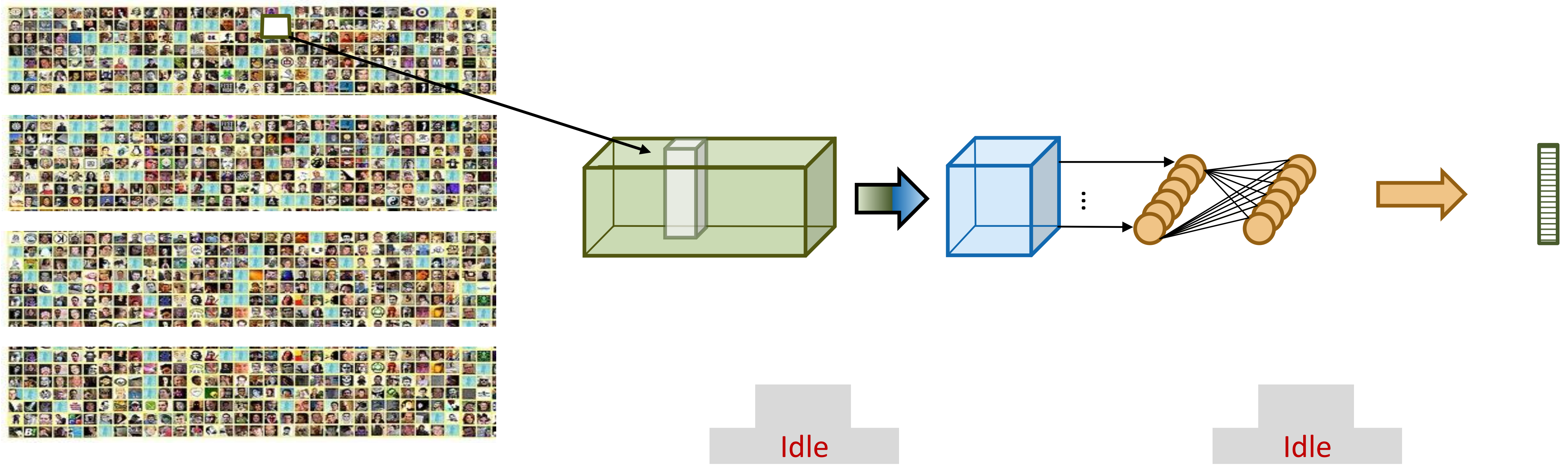
Pipeline parallelism – limited by network size



- Layers/parameters can be distributed across processors
- Sparse communication pattern (only pipeline stages)
- Mini-batch has to be copied through all processors
- Consistent model introduces idle-time "bubble"

[Blelloch & Rosenberg 1987]

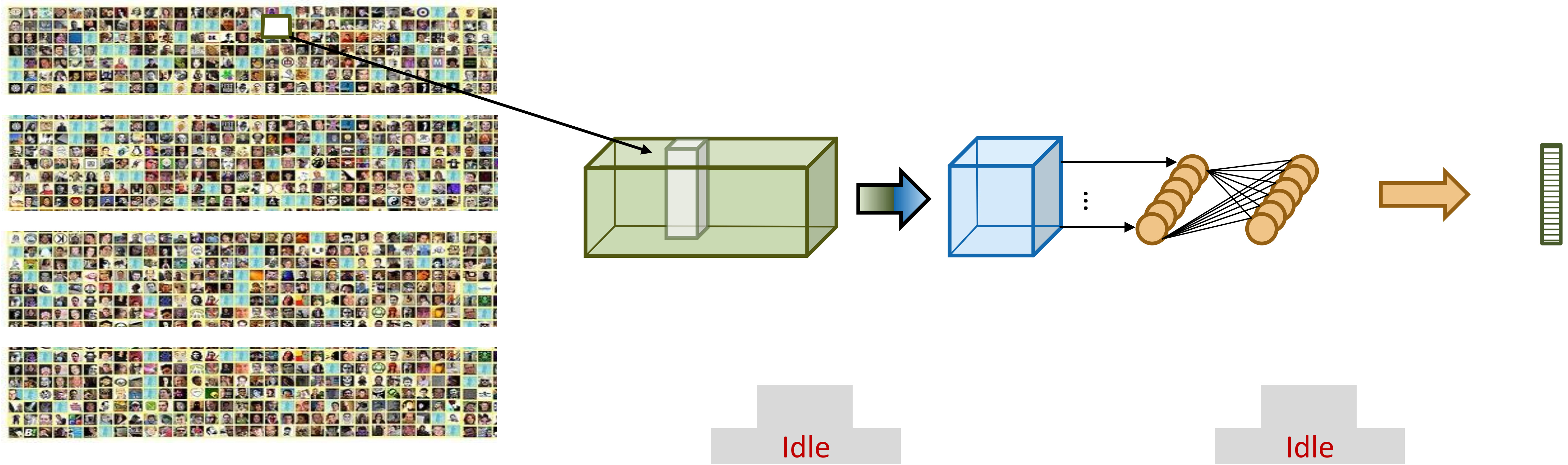
Pipeline parallelism – limited by network size



- Layers/parameters can be distributed across processors
- Sparse communication pattern (only pipeline stages)
- Mini-batch has to be copied through all processors
- Consistent model introduces idle-time "bubble"

[Blelloch & Rosenberg 1987]

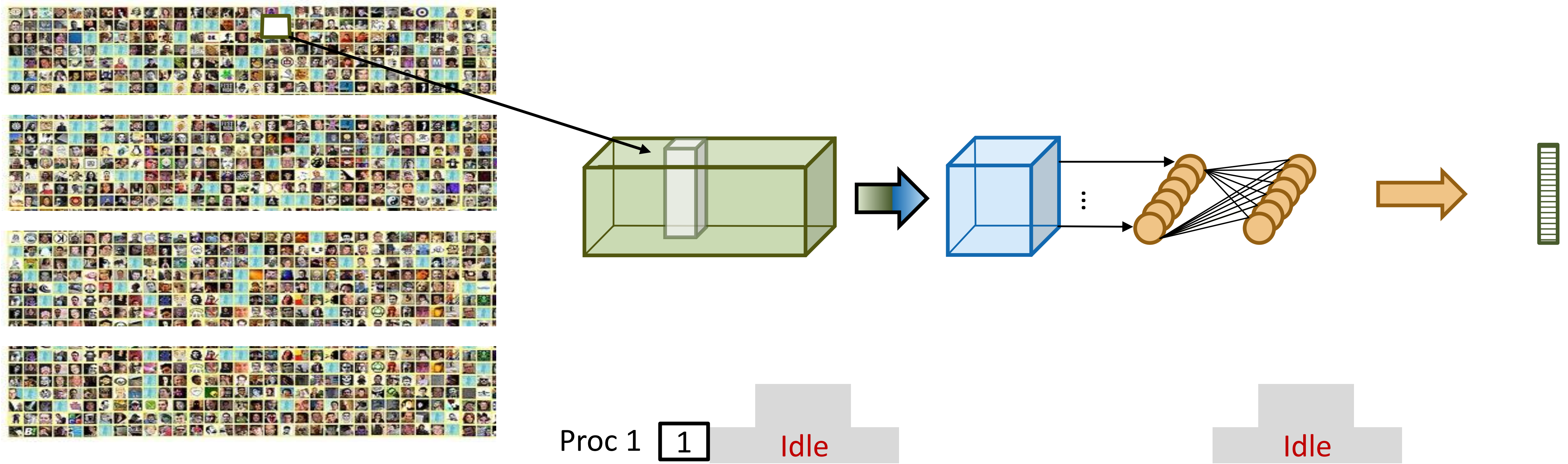
Pipeline parallelism – limited by network size



- Layers/parameters can be distributed across processors
- Sparse communication pattern (only pipeline stages)
- Mini-batch has to be copied through all processors
- Consistent model introduces idle-time "bubble"

[Blelloch & Rosenberg 1987]

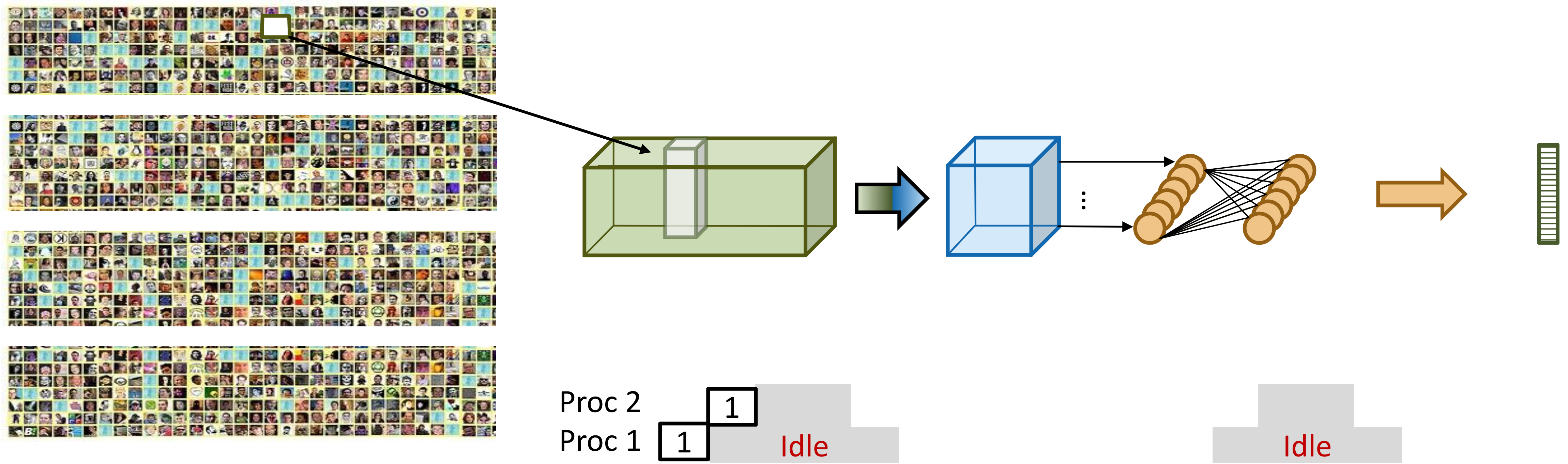
Pipeline parallelism – limited by network size



- Layers/parameters can be distributed across processors
- Sparse communication pattern (only pipeline stages)
- Mini-batch has to be copied through all processors
- Consistent model introduces idle-time "bubble"

[Blelloch & Rosenberg 1987]

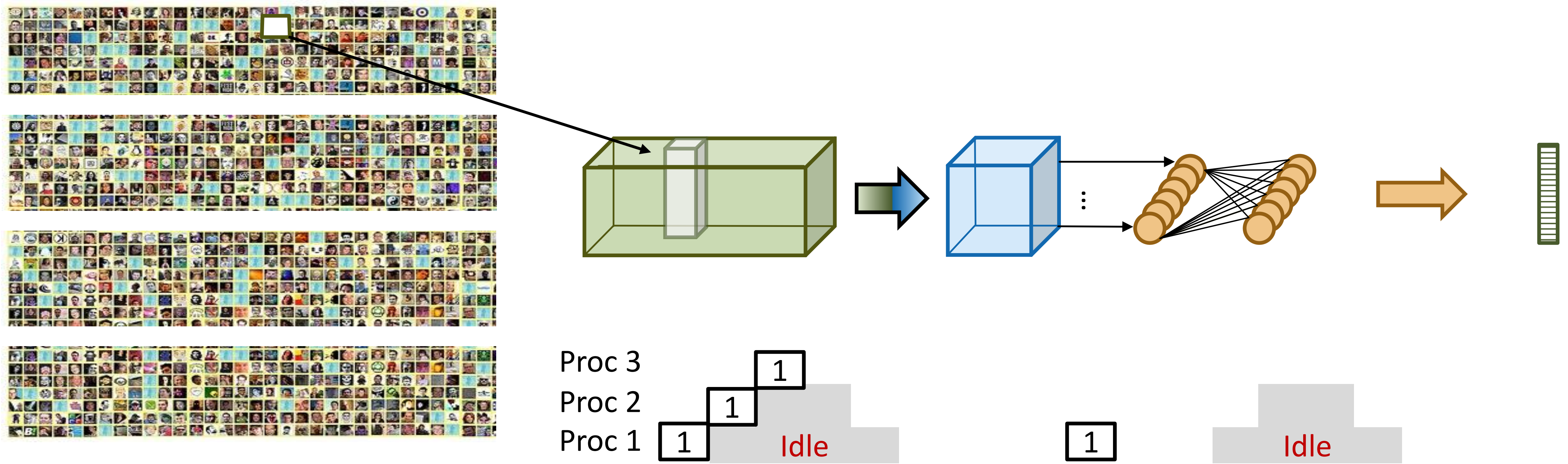
Pipeline parallelism – limited by network size



- Layers/parameters can be distributed across processors
- Sparse communication pattern (only pipeline stages)
- Mini-batch has to be copied through all processors
- Consistent model introduces idle-time "bubble"

[Blelloch & Rosenberg 1987]

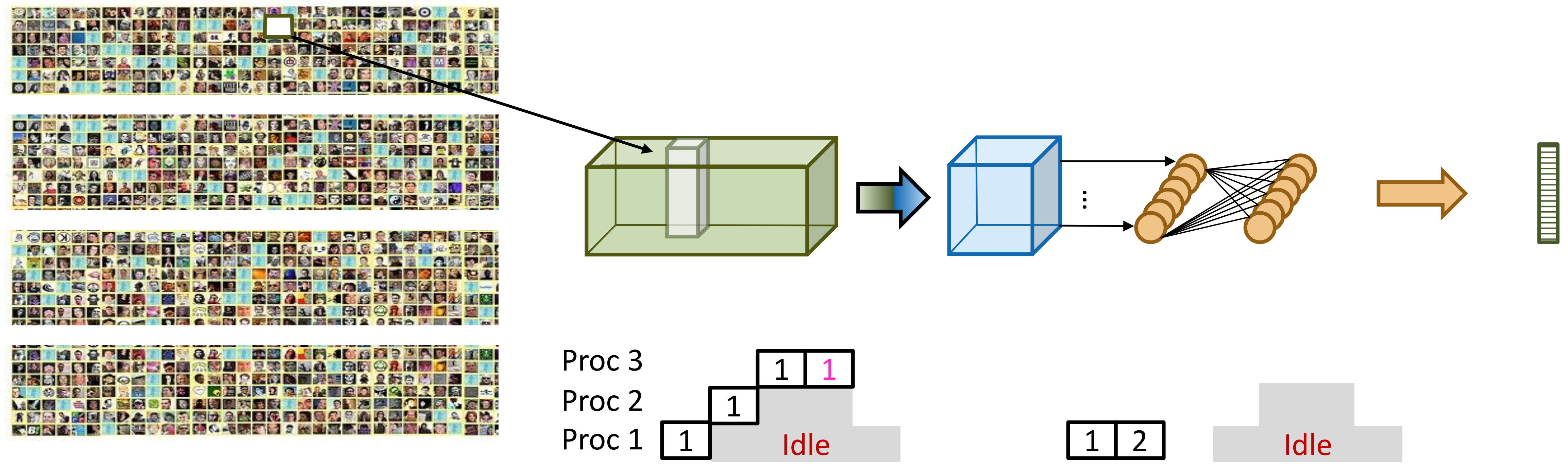
Pipeline parallelism – limited by network size



- Layers/parameters can be distributed across processors
- Sparse communication pattern (only pipeline stages)
- Mini-batch has to be copied through all processors
- Consistent model introduces idle-time "bubble"

[Blelloch & Rosenberg 1987]

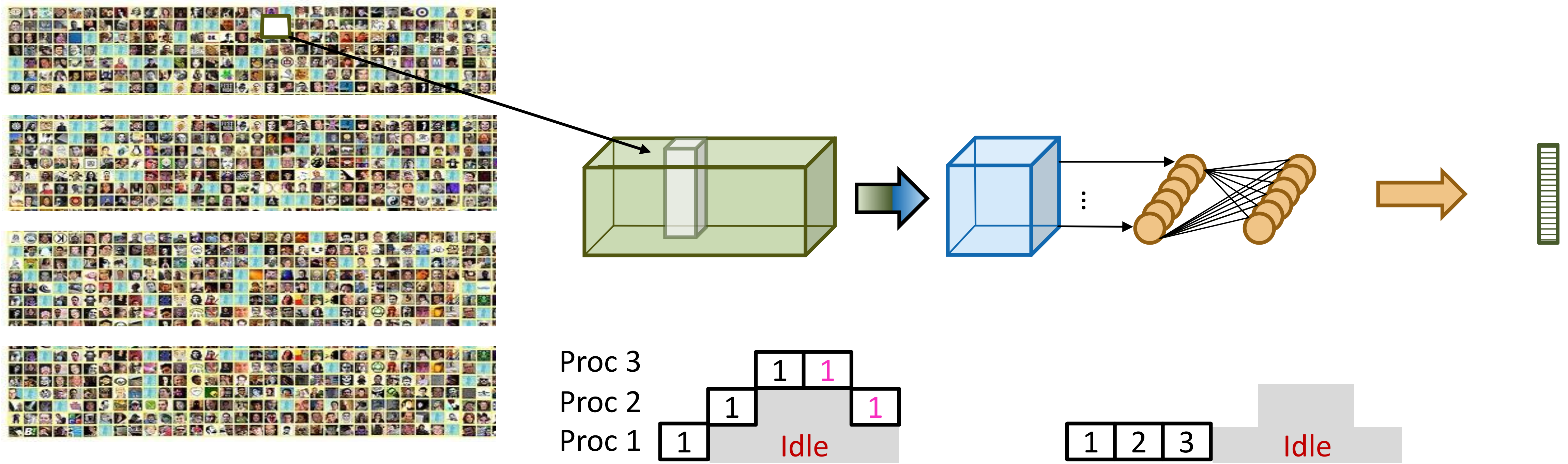
Pipeline parallelism – limited by network size



- Layers/parameters can be distributed across processors
- Sparse communication pattern (only pipeline stages)
- Mini-batch has to be copied through all processors
- Consistent model introduces idle-time "bubble"

[Blelloch & Rosenberg 1987]

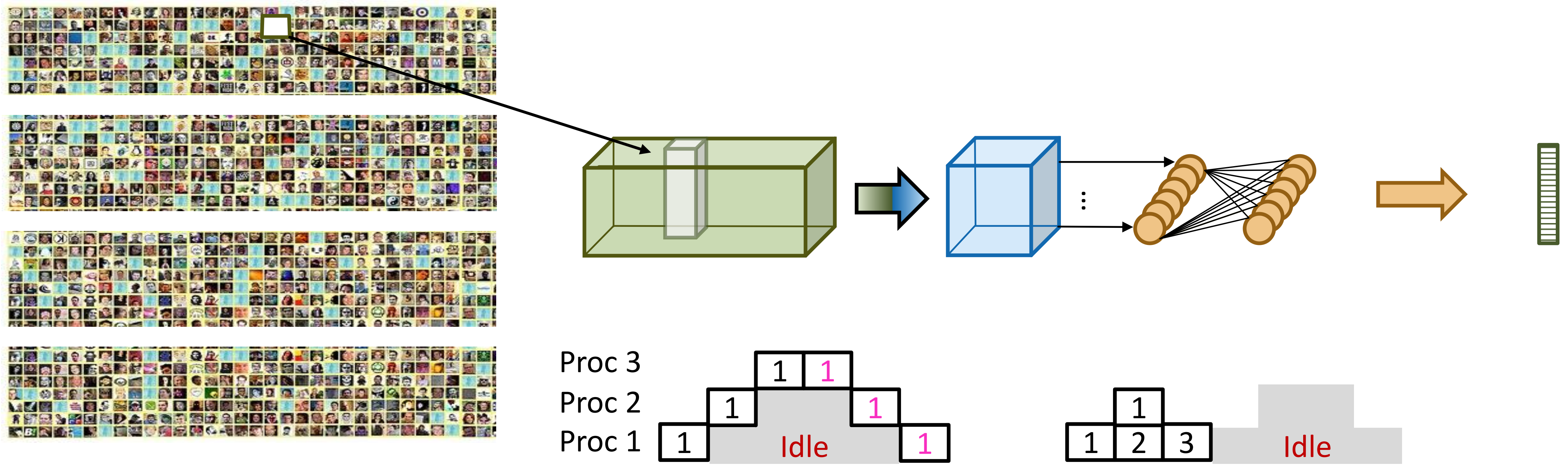
Pipeline parallelism – limited by network size



- Layers/parameters can be distributed across processors
- Sparse communication pattern (only pipeline stages)
- Mini-batch has to be copied through all processors
- Consistent model introduces idle-time "bubble"

[Blelloch & Rosenberg 1987]

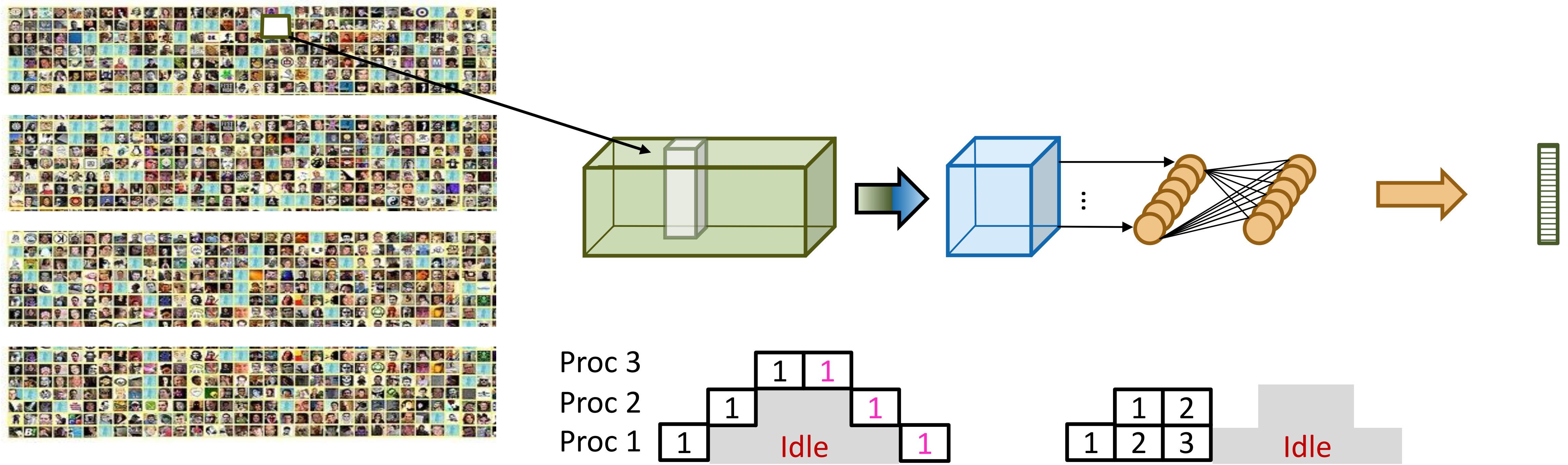
Pipeline parallelism – limited by network size



- Layers/parameters can be distributed across processors
- Sparse communication pattern (only pipeline stages)
- Mini-batch has to be copied through all processors
- Consistent model introduces idle-time "bubble"

[Blelloch & Rosenberg 1987]

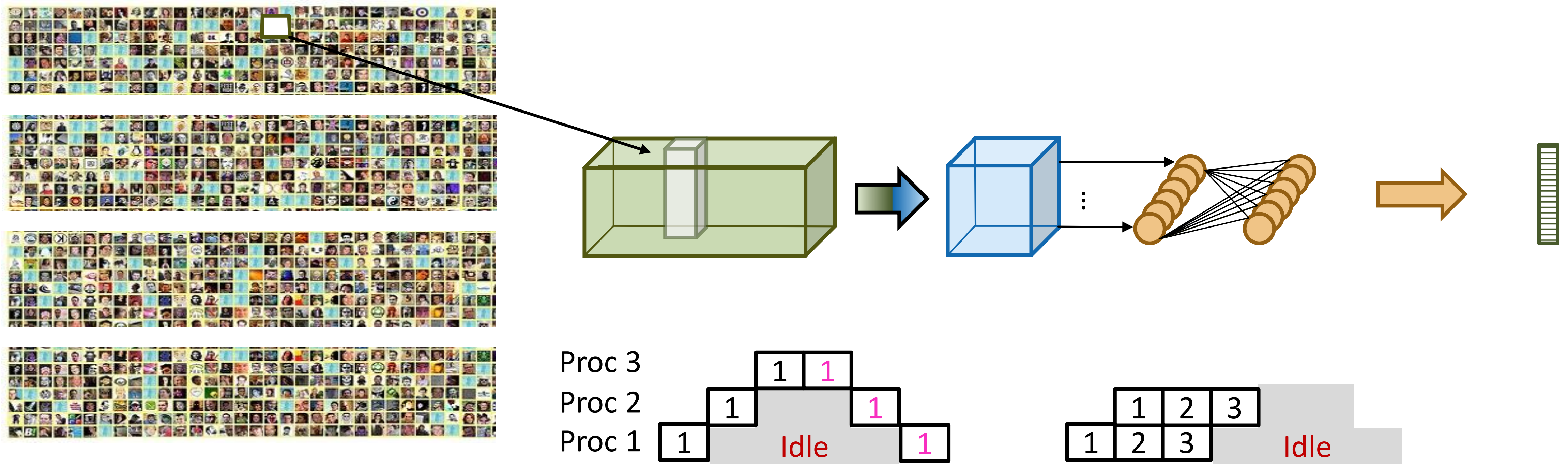
Pipeline parallelism – limited by network size



- Layers/parameters can be distributed across processors
- Sparse communication pattern (only pipeline stages)
- Mini-batch has to be copied through all processors
- Consistent model introduces idle-time "bubble"

[Blelloch & Rosenberg 1987]

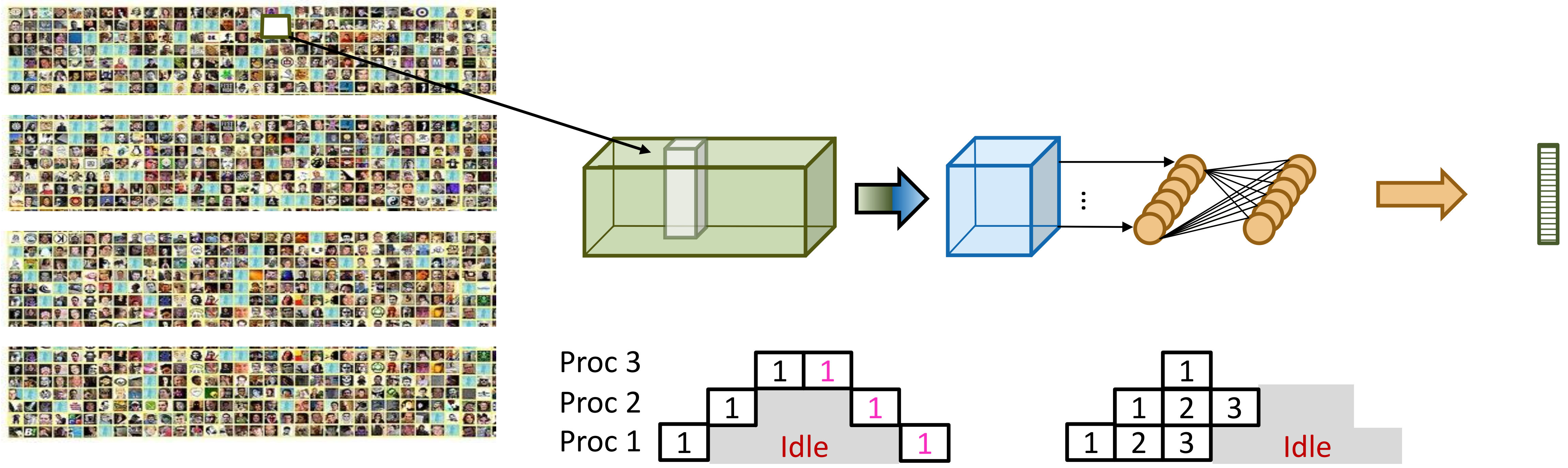
Pipeline parallelism – limited by network size



- Layers/parameters can be distributed across processors
- Sparse communication pattern (only pipeline stages)
- Mini-batch has to be copied through all processors
- Consistent model introduces idle-time "bubble"

[Blelloch & Rosenberg 1987]

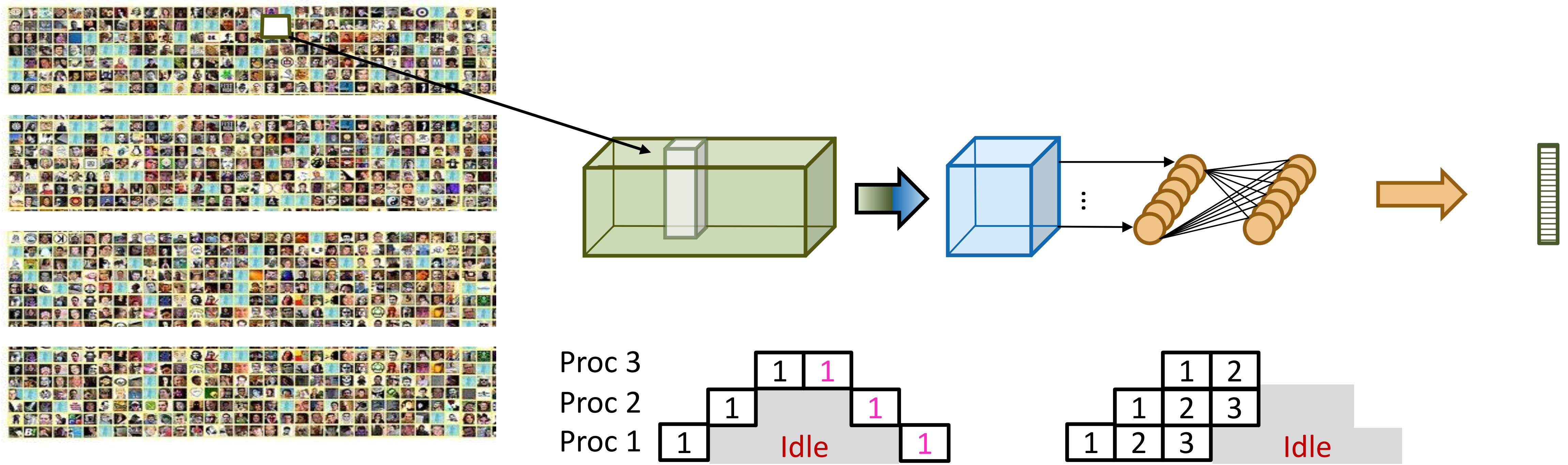
Pipeline parallelism – limited by network size



- Layers/parameters can be distributed across processors
- Sparse communication pattern (only pipeline stages)
- Mini-batch has to be copied through all processors
- Consistent model introduces idle-time "bubble"

[Blelloch & Rosenberg 1987]

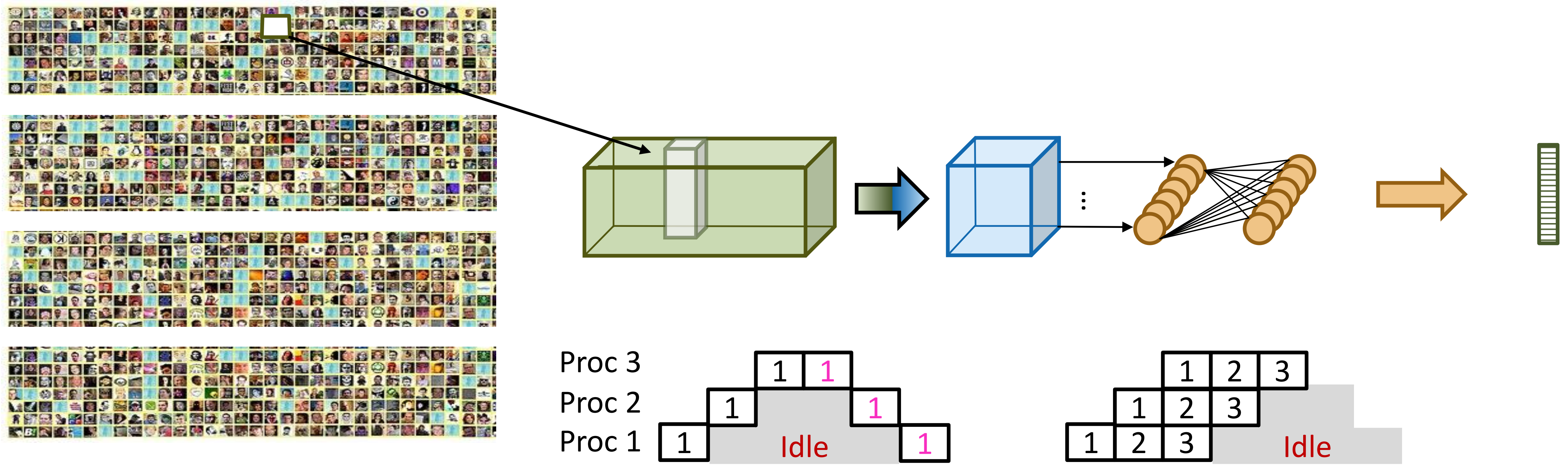
Pipeline parallelism – limited by network size



- Layers/parameters can be distributed across processors
- Sparse communication pattern (only pipeline stages)
- Mini-batch has to be copied through all processors
- Consistent model introduces idle-time "bubble"

[Blelloch & Rosenberg 1987]

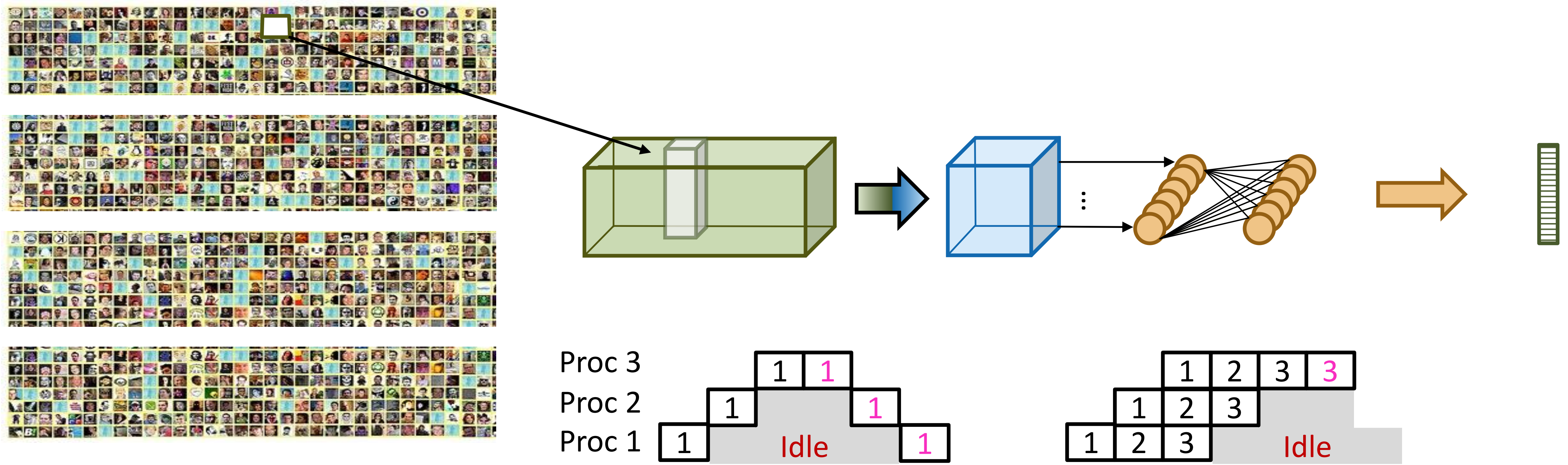
Pipeline parallelism – limited by network size



- Layers/parameters can be distributed across processors
- Sparse communication pattern (only pipeline stages)
- Mini-batch has to be copied through all processors
- Consistent model introduces idle-time "bubble"

[Blelloch & Rosenberg 1987]

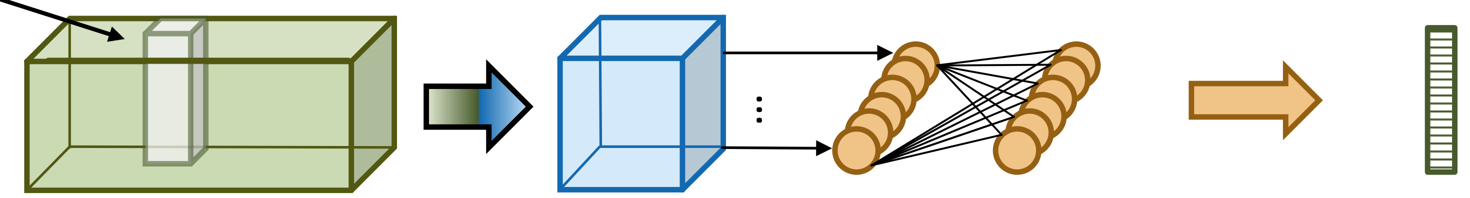
Pipeline parallelism – limited by network size



- Layers/parameters can be distributed across processors
- Sparse communication pattern (only pipeline stages)
- Mini-batch has to be copied through all processors
- Consistent model introduces idle-time "bubble"

[Blelloch & Rosenberg 1987]

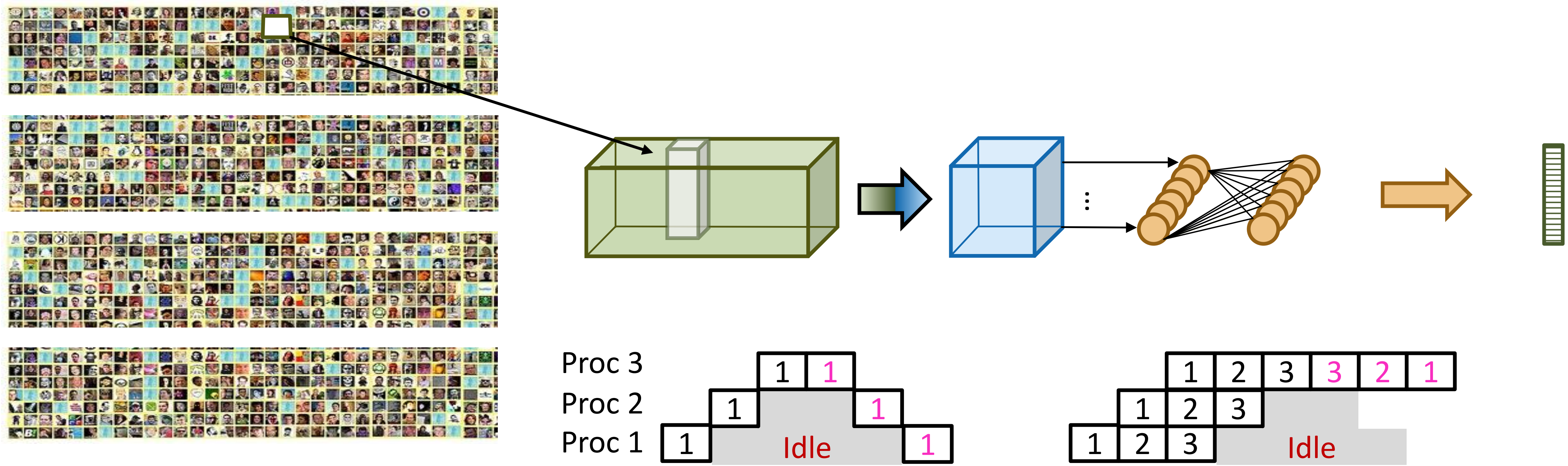
Pipeline parallelism – limited by network size



- Layers/parameters can be distributed across processors
- Sparse communication pattern (only pipeline stages)
- Mini-batch has to be copied through all processors
- Consistent model introduces idle-time "bubble"

[Blelloch & Rosenberg 1987]

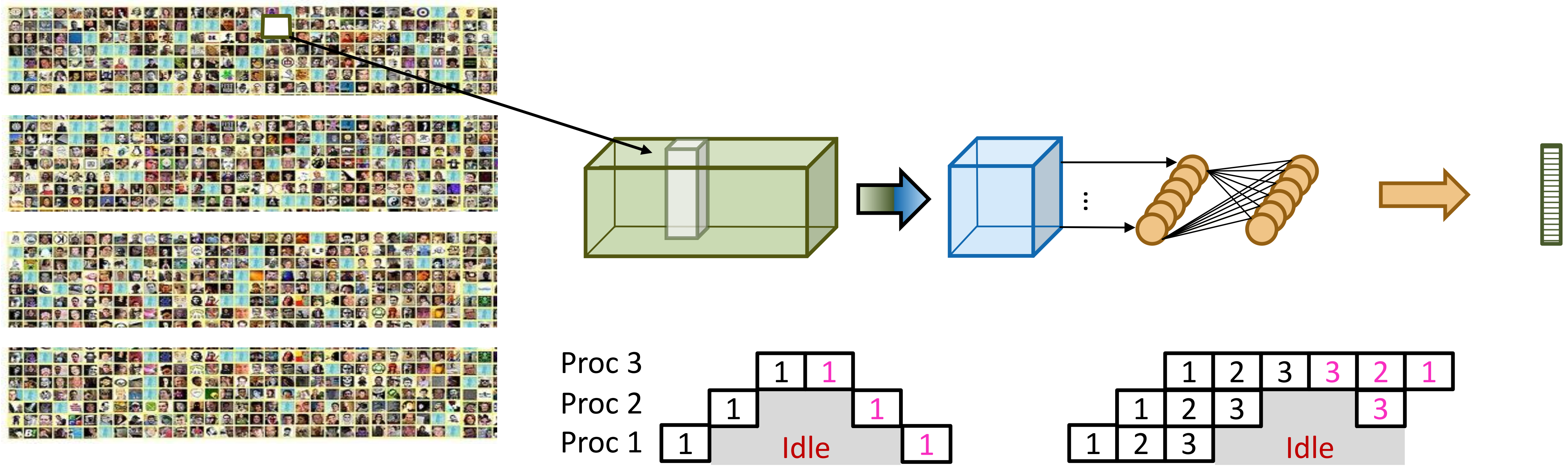
Pipeline parallelism – limited by network size



- Layers/parameters can be distributed across processors
- Sparse communication pattern (only pipeline stages)
- Mini-batch has to be copied through all processors
- Consistent model introduces idle-time "bubble"

[Blelloch & Rosenberg 1987]

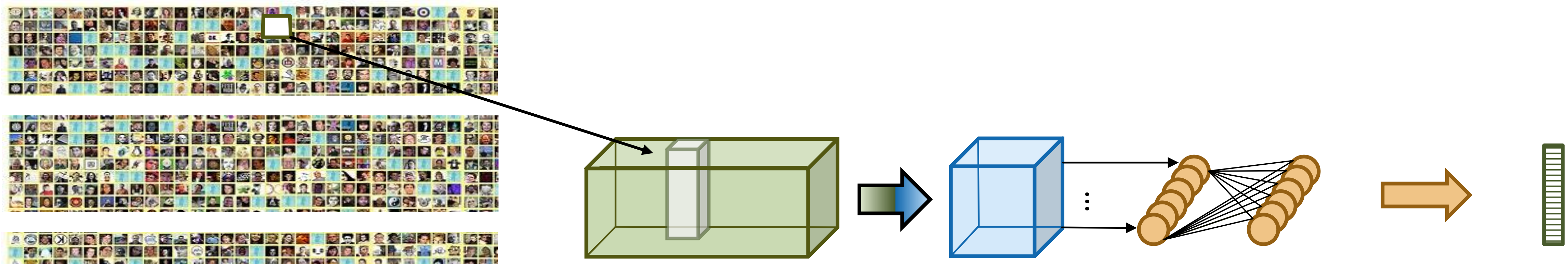
Pipeline parallelism – limited by network size



- Layers/parameters can be distributed across processors
- Sparse communication pattern (only pipeline stages)
- Mini-batch has to be copied through all processors
- Consistent model introduces idle-time "bubble"

[Blelloch & Rosenberg 1987]

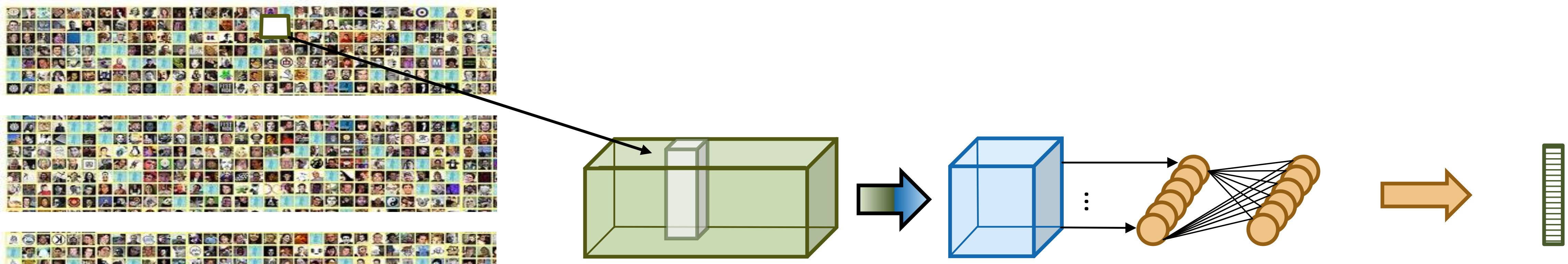
Pipeline parallelism – limited by network size

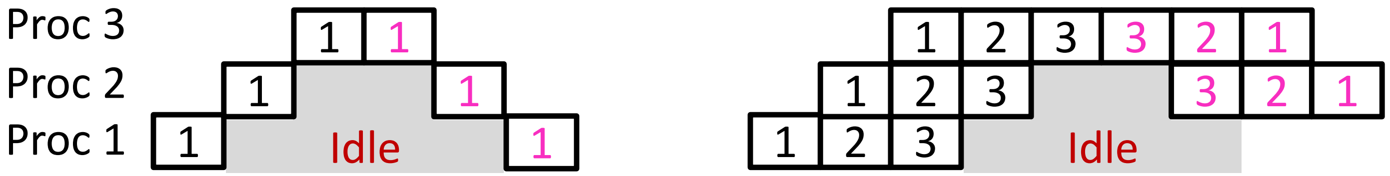


- Layers/parameters can be distributed across processors
- Sparse communication pattern (only pipeline stages)
- Mini-batch has to be copied through all processors
- Consistent model introduces idle-time “bubble”

[Blelloch & Rosenberg 1987]

Pipeline parallelism – limited by network size

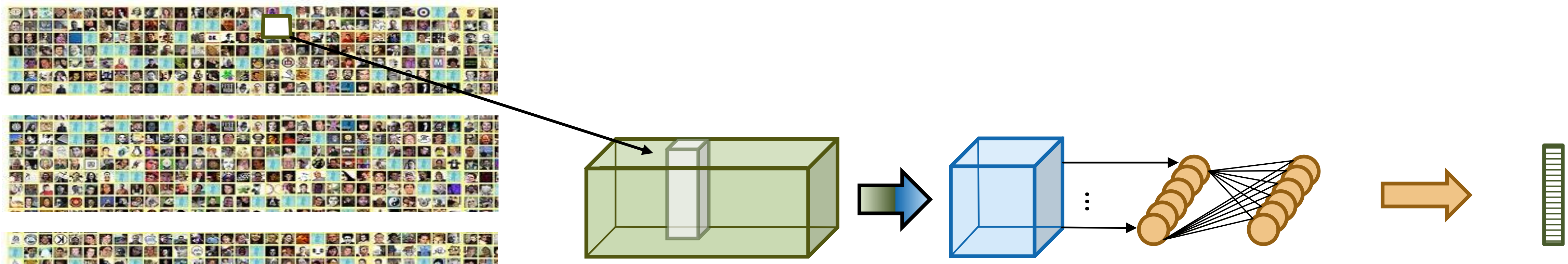


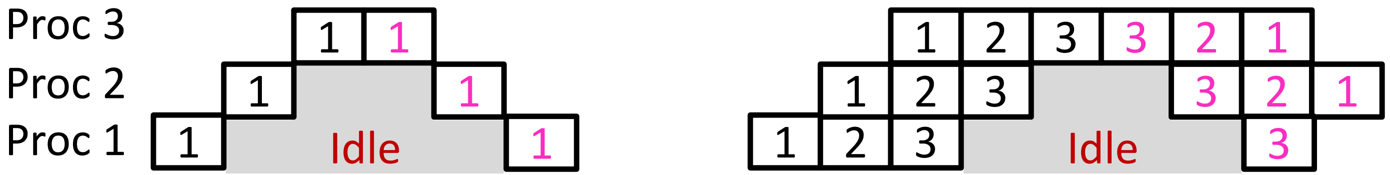


- Layers/parameters can be distributed across processors
- Sparse communication pattern (only pipeline stages)
- Mini-batch has to be copied through all processors
- Consistent model introduces idle-time “bubble”

[Blelloch & Rosenberg 1987]

Pipeline parallelism – limited by network size

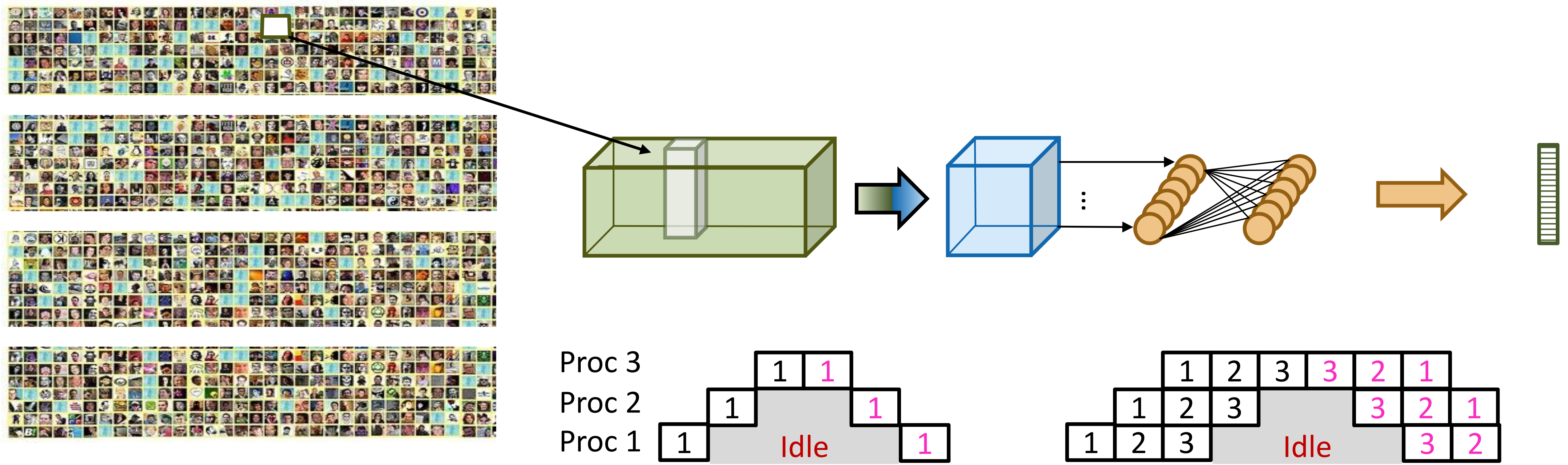




- Layers/parameters can be distributed across processors
- Sparse communication pattern (only pipeline stages)
- Mini-batch has to be copied through all processors
- Consistent model introduces idle-time "bubble"

[Blelloch & Rosenberg 1987]

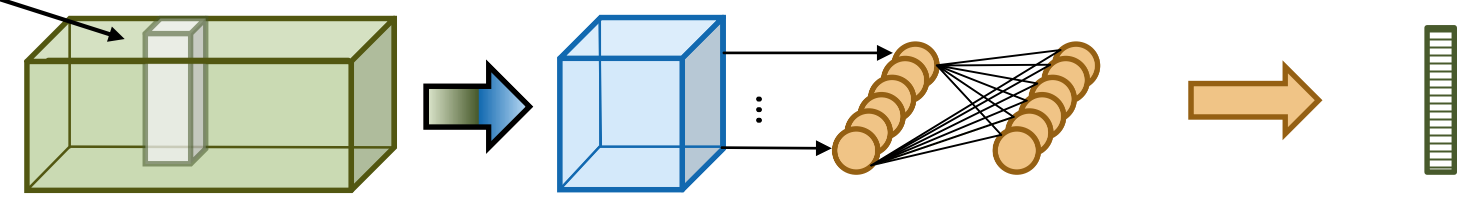
Pipeline parallelism – limited by network size

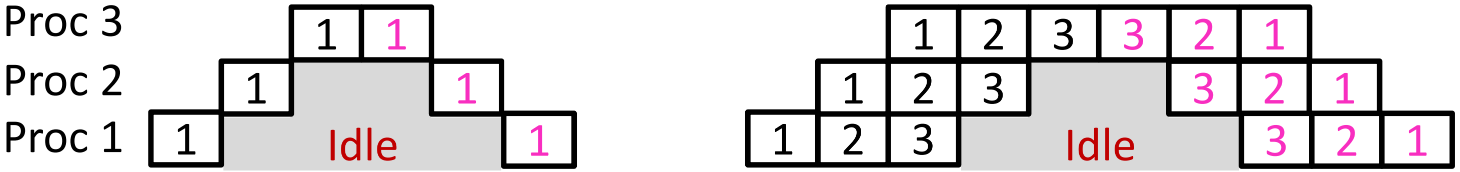


- Layers/parameters can be distributed across processors
- Sparse communication pattern (only pipeline stages)
- Mini-batch has to be copied through all processors
- Consistent model introduces idle-time “bubble”

[Blelloch & Rosenberg 1987]

Pipeline parallelism – limited by network size

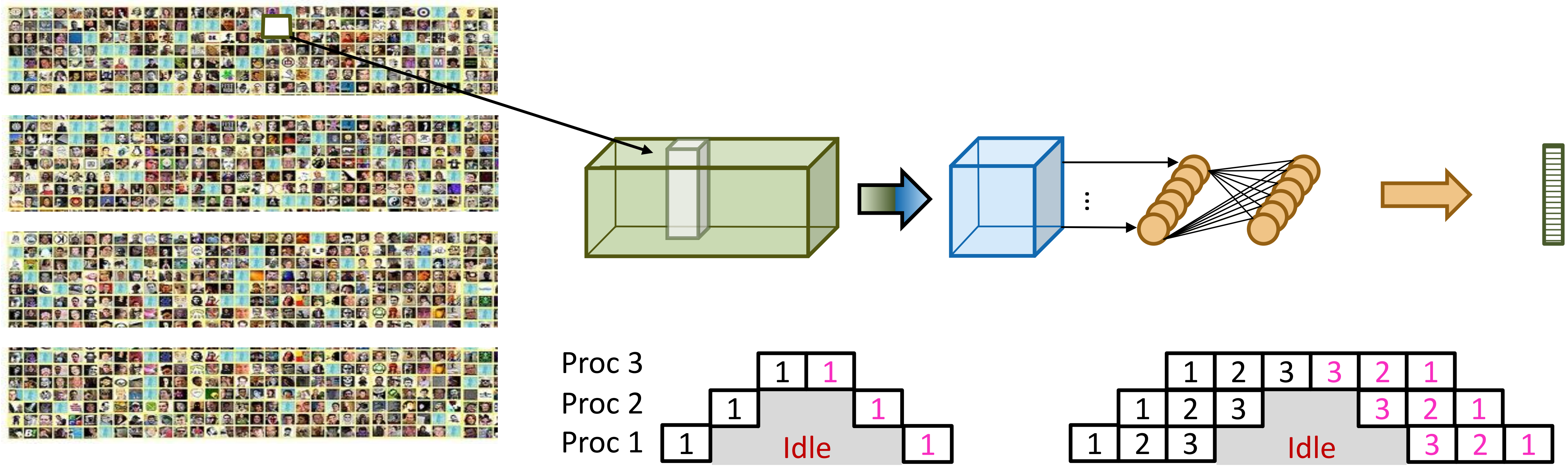




- Layers/parameters can be distributed across processors
- Sparse communication pattern (only pipeline stages)
- Mini-batch has to be copied through all processors
- Consistent model introduces idle-time "bubble"

[Blelloch & Rosenberg 1987]

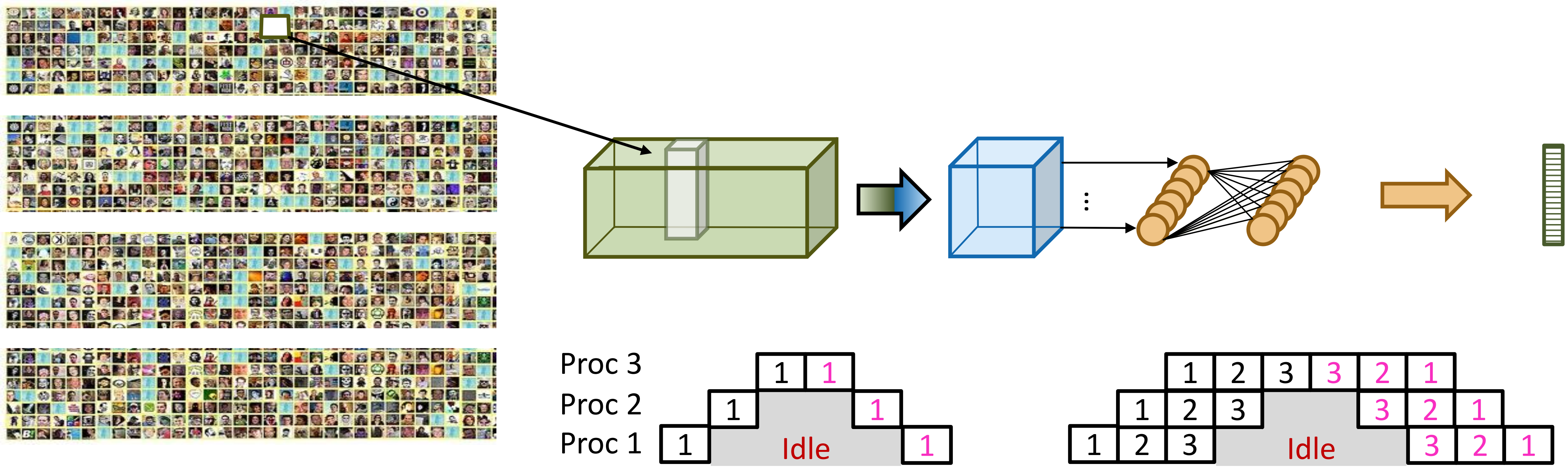
Pipeline parallelism – limited by network size



- Layers/parameters can be distributed across processors
- Sparse communication pattern (only pipeline stages)
- Mini-batch has to be copied through all processors
- Consistent model introduces idle-time “bubble”

[Blelloch & Rosenberg 1987]

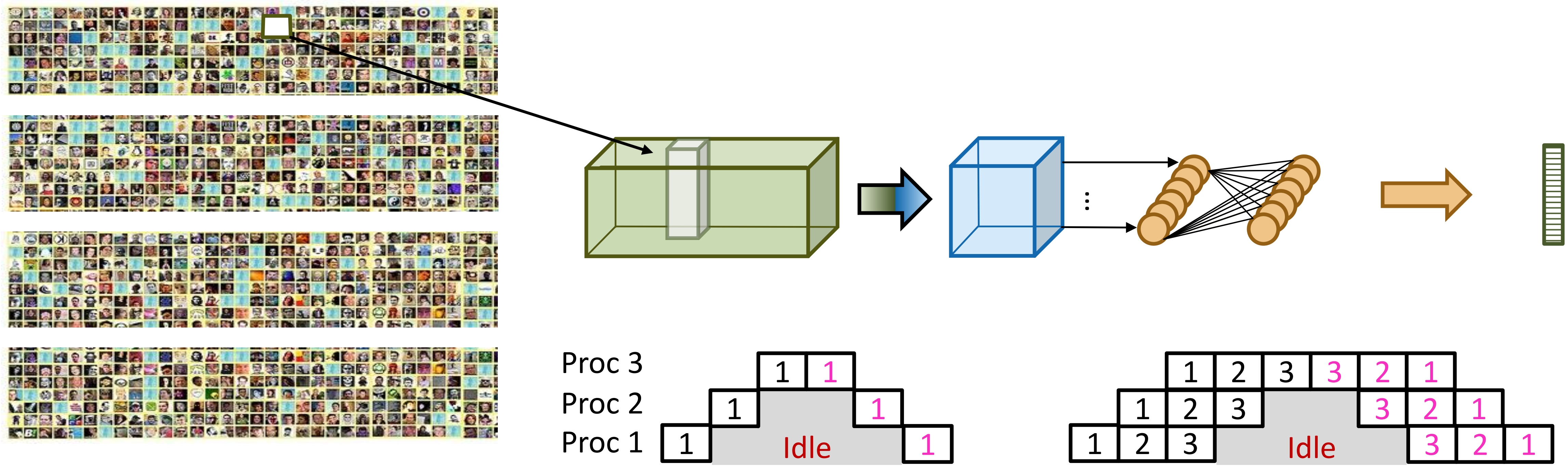
Pipeline parallelism – limited by network size



- Layers/parameters can be distributed across processors
- Sparse communication pattern (only pipeline stages)
- Mini-batch has to be copied through all processors
- Consistent model introduces idle-time "bubble"

[Blelloch & Rosenberg 1987]

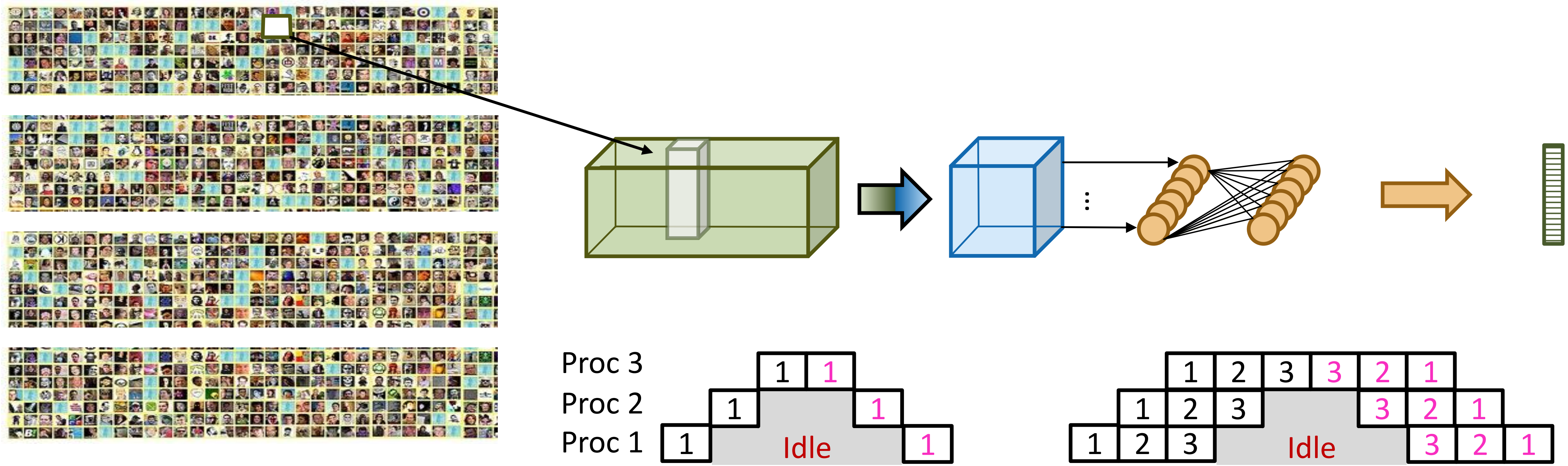
Pipeline parallelism – limited by network size



- Layers/parameters can be distributed across processors
- Sparse communication pattern (only pipeline stages)
- Mini-batch has to be copied through all processors
- Consistent model introduces idle-time "bubble"

[Blelloch & Rosenberg 1987]

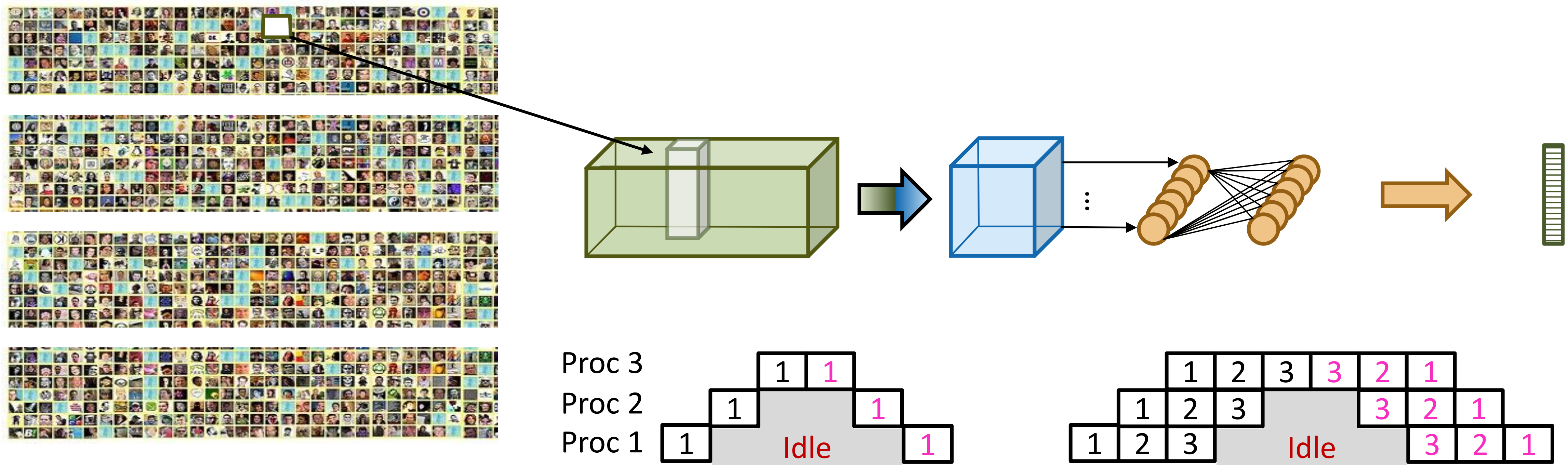
Pipeline parallelism – limited by network size



- Layers/parameters can be distributed across processors
- Sparse communication pattern (only pipeline stages)
- Mini-batch has to be copied through all processors
- Consistent model introduces idle-time "bubble"

[Blelloch & Rosenberg 1987]

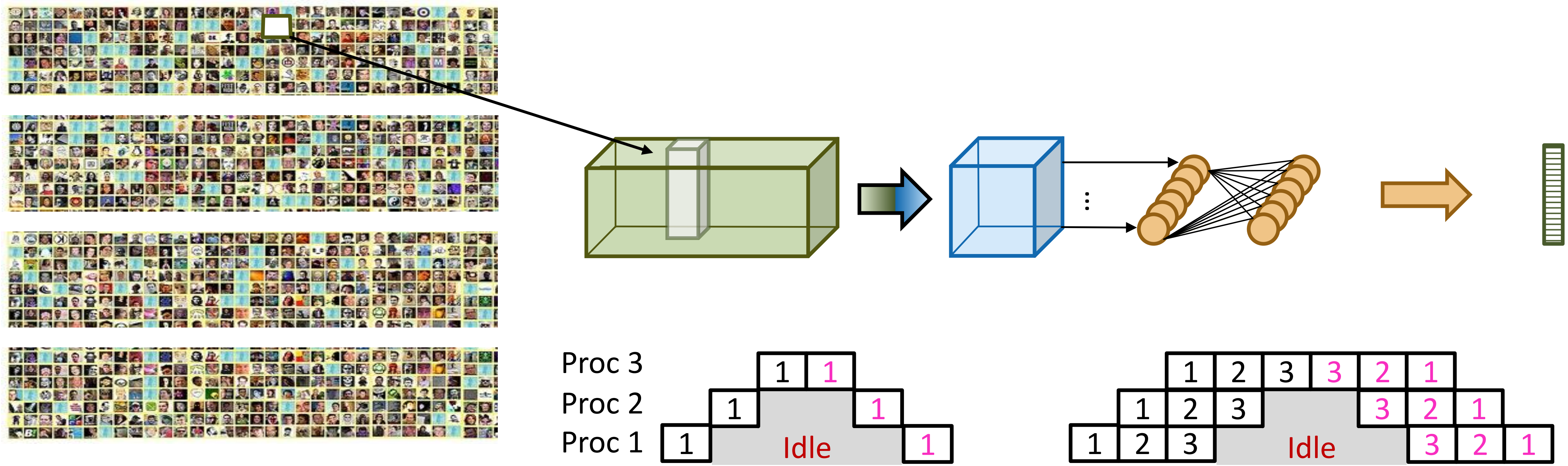
Pipeline parallelism – limited by network size



- Layers/parameters can be distributed across processors
- Sparse communication pattern (only pipeline stages)
- **Mini-batch has to be copied through all processors**
- **Consistent model introduces idle-time "bubble"**

[Blelloch & Rosenberg 1987]

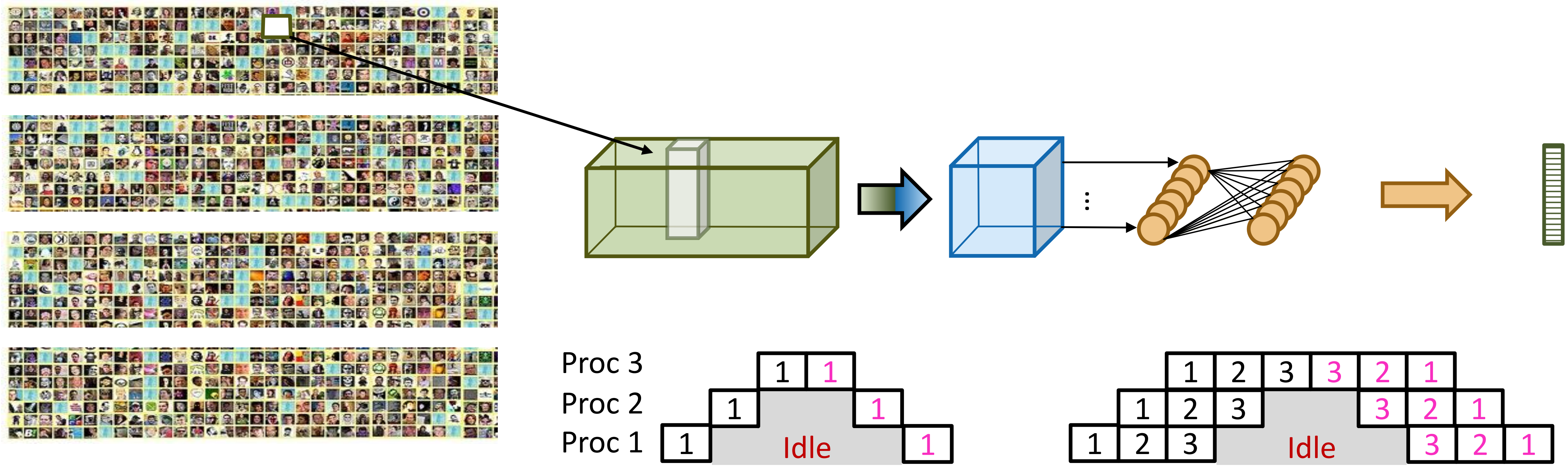
Pipeline parallelism – limited by network size



- Layers/parameters can be distributed across processors
- Sparse communication pattern (only pipeline stages)
- Mini-batch has to be copied through all processors
- Consistent model introduces idle-time “bubble”

[Blelloch & Rosenberg 1987]

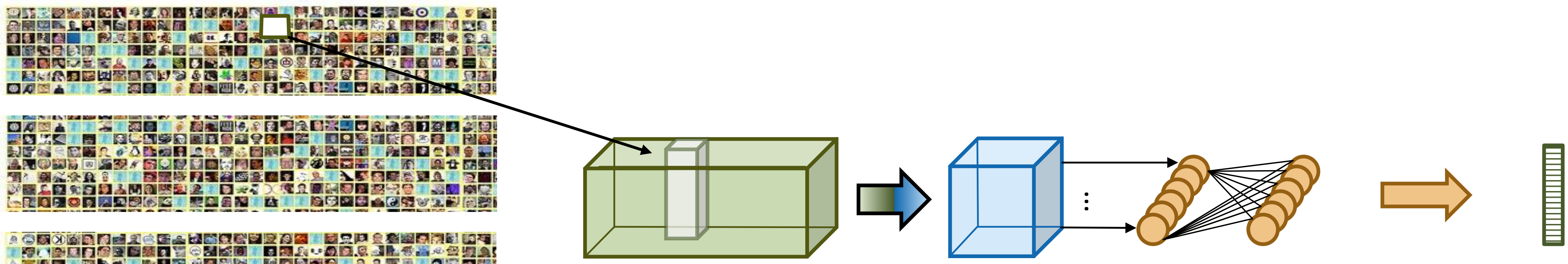
Pipeline parallelism – limited by network size

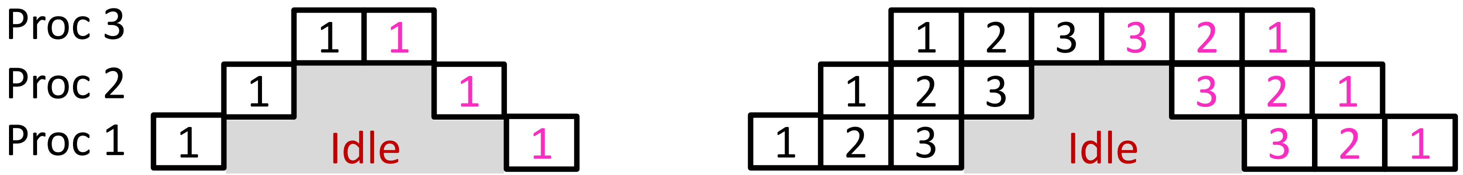


- Layers/parameters can be distributed across processors
- Sparse communication pattern (only pipeline stages)
- Mini-batch has to be copied through all processors
- Consistent model introduces idle-time "bubble"

[Blelloch & Rosenberg 1987]

Pipeline parallelism – limited by network size

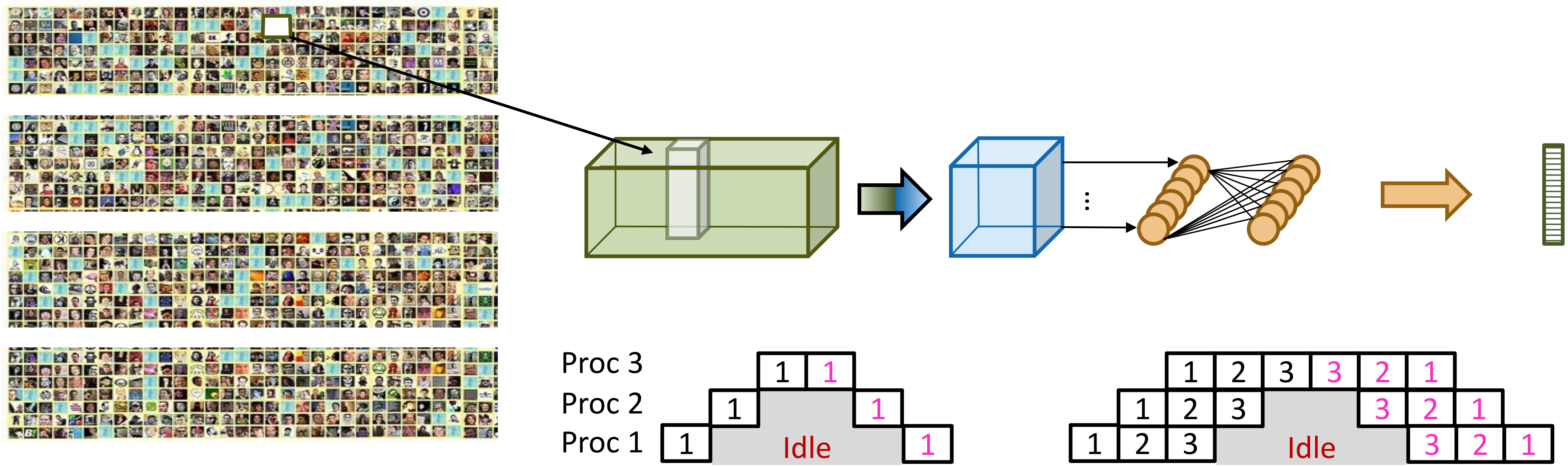




- Layers/parameters can be distributed across processors
- Sparse communication pattern (only pipeline stages)
- Mini-batch has to be copied through all processors
- Consistent model introduces idle-time "bubble"

[Blelloch & Rosenberg 1987]

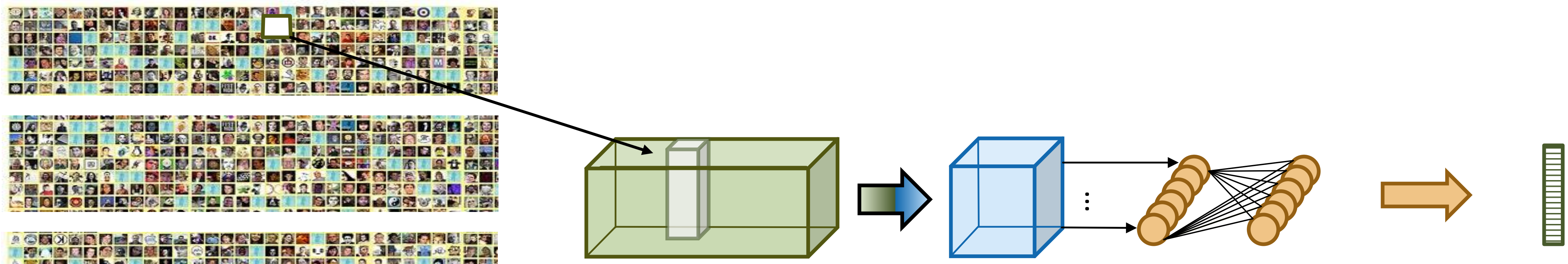
Pipeline parallelism – limited by network size

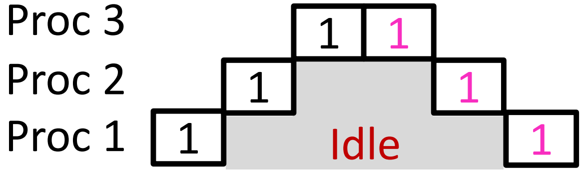


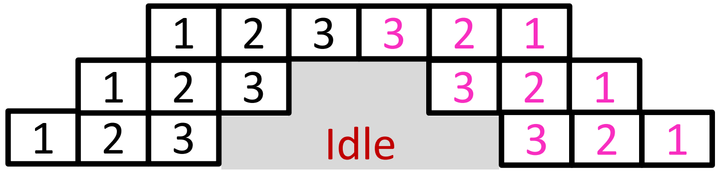
- Layers/parameters can be distributed across processors
- Sparse communication pattern (only pipeline stages)
- Mini-batch has to be copied through all processors
- Consistent model introduces idle-time "bubble"

[Blelloch & Rosenberg 1987]

Pipeline parallelism – limited by network size







Microbatching

- Layers/parameters can be distributed across processors
- Sparse communication pattern (only pipeline stages)
- Mini-batch has to be copied through all processors
- Consistent model introduces idle-time "bubble"

[Blelloch & Rosenberg 1987]

Data parallelism – limited by batch-size

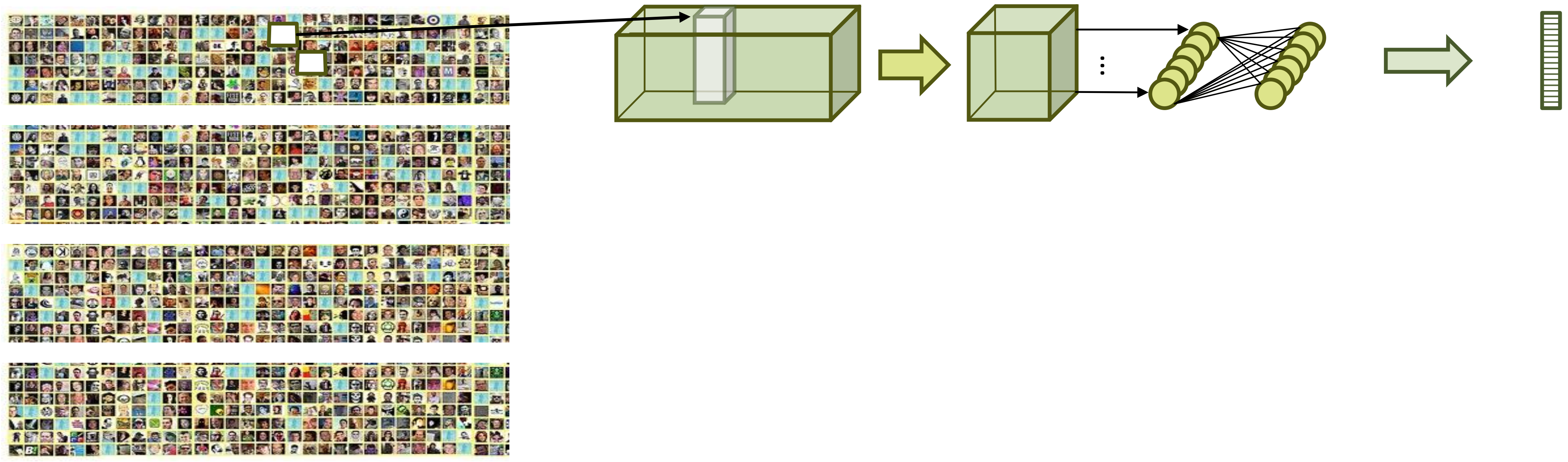
[Zhang et al. 1989]

Data parallelism – limited by batch-size

- Simple and efficient solution, easy to implement
- Duplicate parameters at all processors
- Affects generalization

[Zhang et al. 1989]

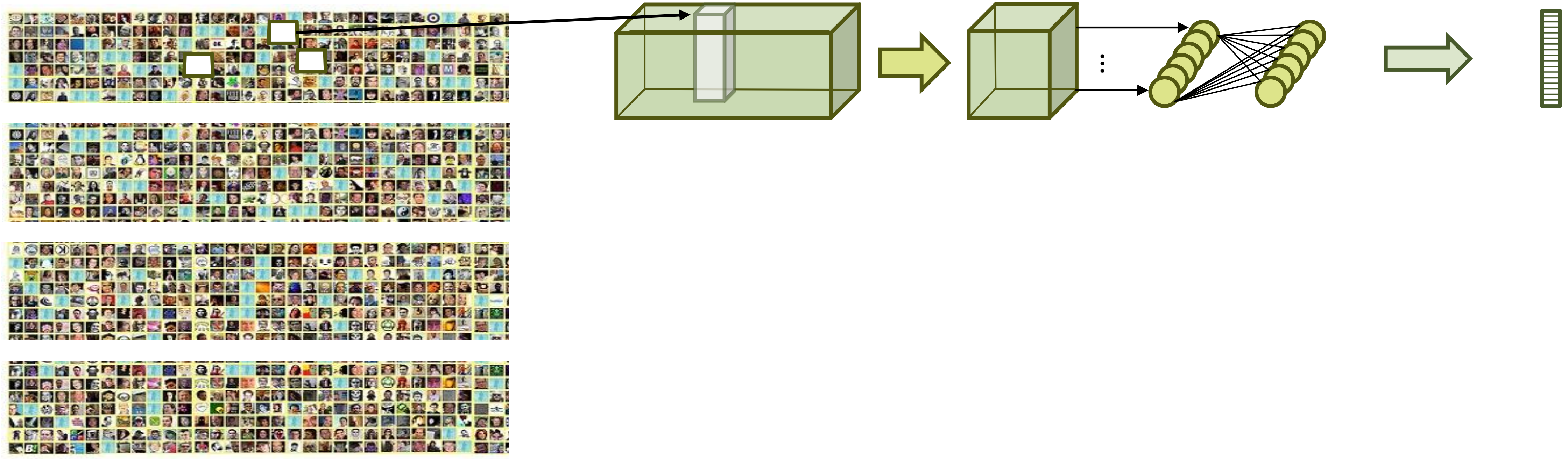
Data parallelism – limited by batch-size



- Simple and efficient solution, easy to implement
- Duplicate parameters at all processors
- Affects generalization

[Zhang et al. 1989]

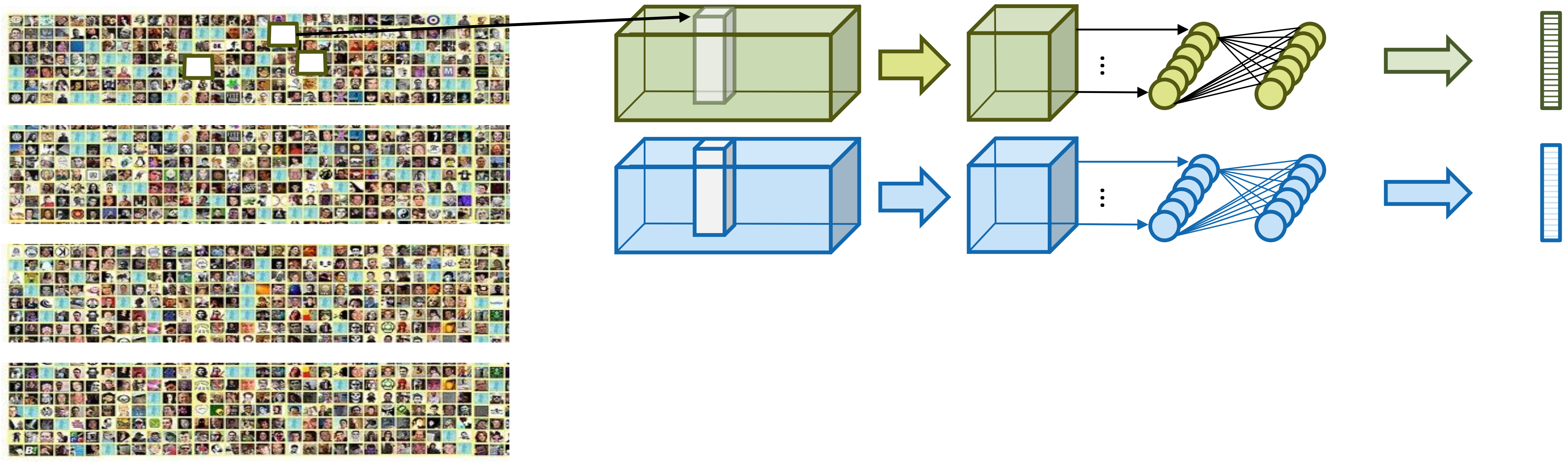
Data parallelism – limited by batch-size



- Simple and efficient solution, easy to implement
- Duplicate parameters at all processors
- Affects generalization

[Zhang et al. 1989]

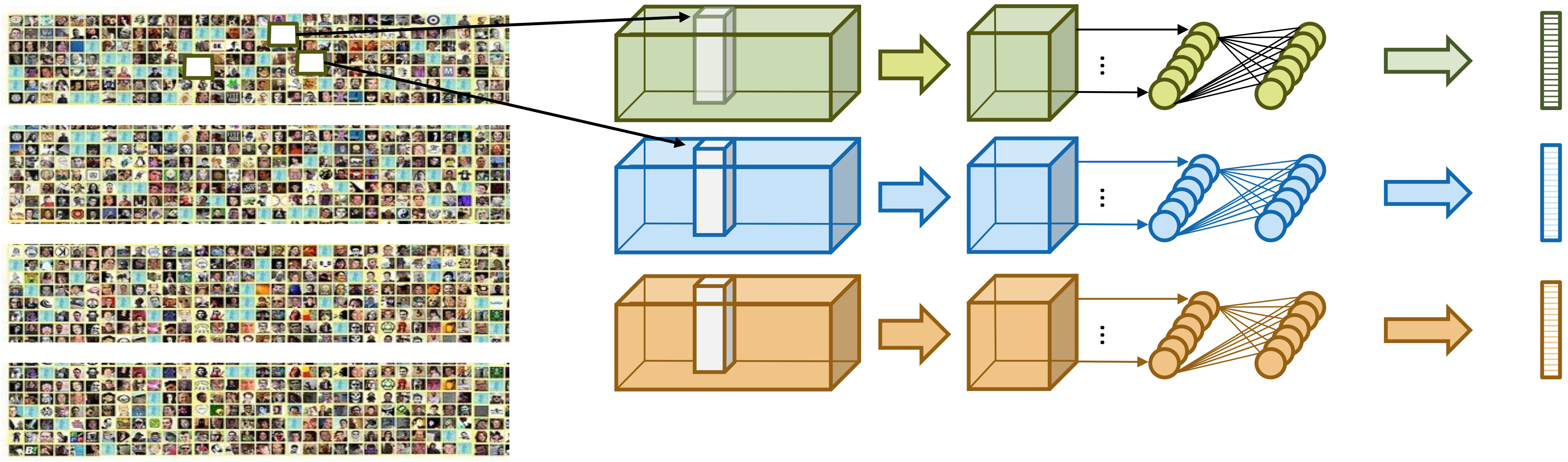
Data parallelism – limited by batch-size



- Simple and efficient solution, easy to implement
- Duplicate parameters at all processors
- Affects generalization

[Zhang et al. 1989]

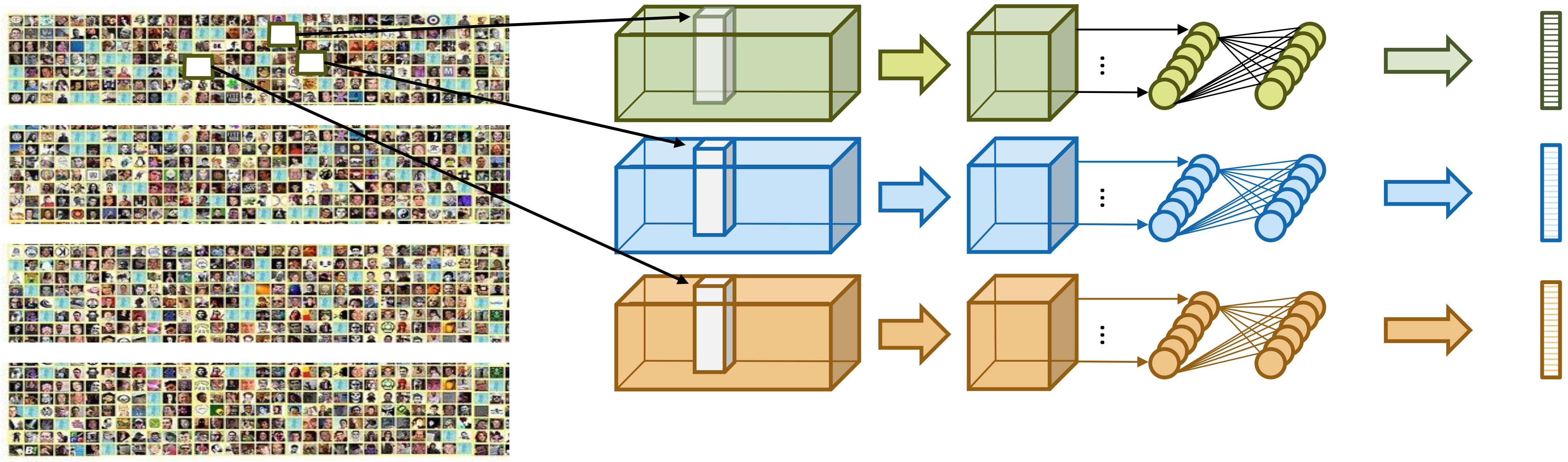
Data parallelism – limited by batch-size



- Simple and efficient solution, easy to implement
- Duplicate parameters at all processors
- Affects generalization

[Zhang et al. 1989]

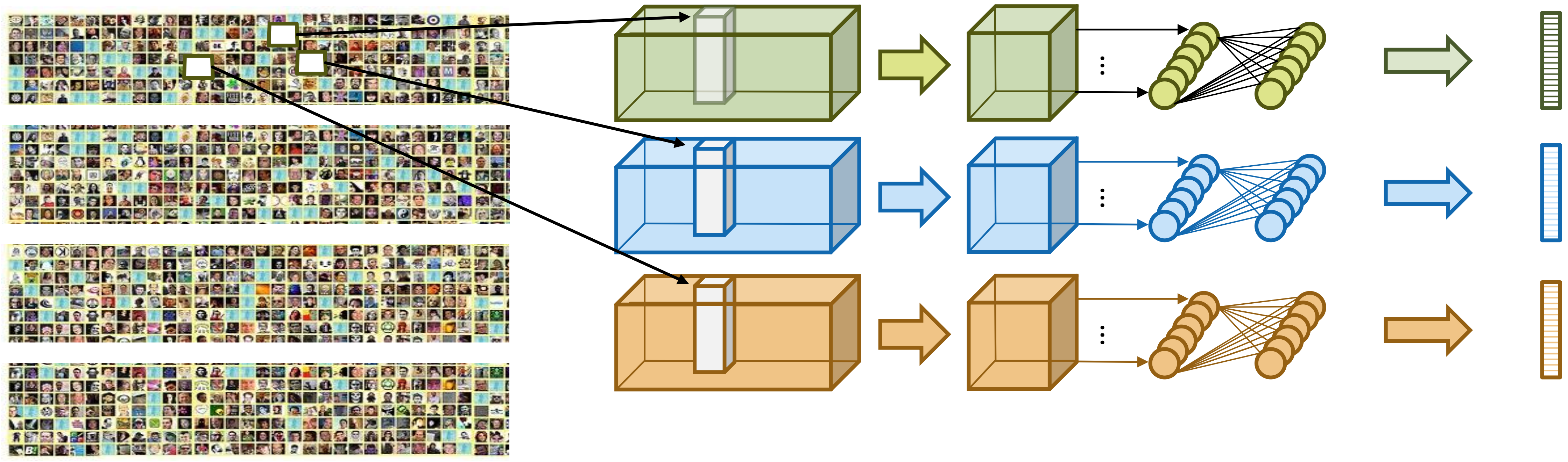
Data parallelism – limited by batch-size



- Simple and efficient solution, easy to implement
- Duplicate parameters at all processors
- Affects generalization

[Zhang et al. 1989]

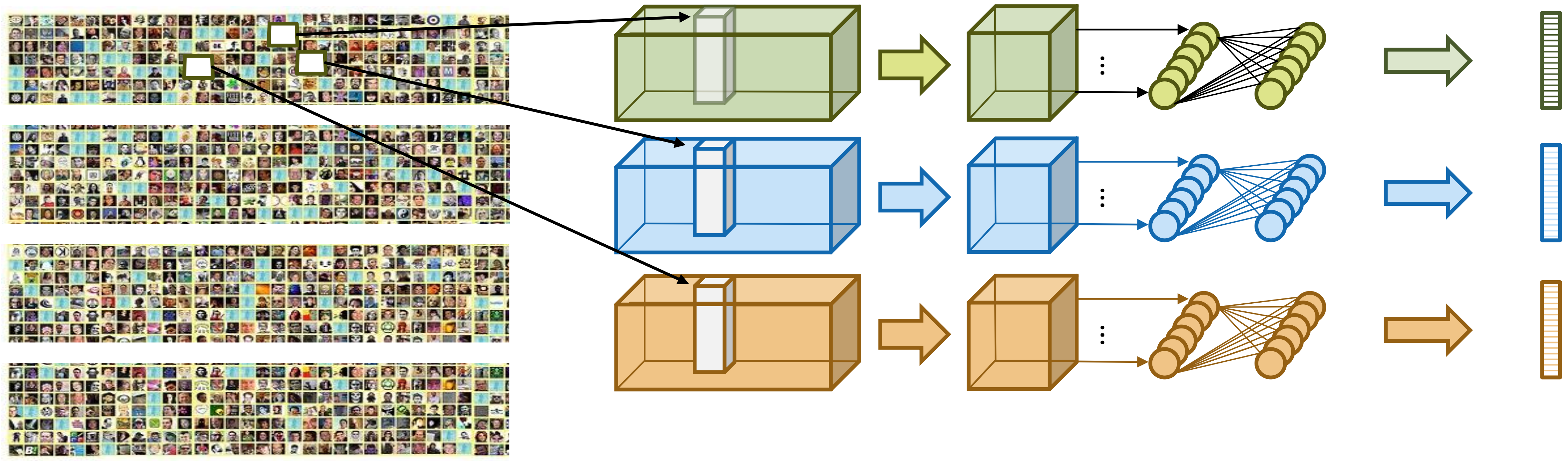
Data parallelism – limited by batch-size



- Simple and efficient solution, easy to implement
- Duplicate parameters at all processors
- Affects generalization

[Zhang et al. 1989]

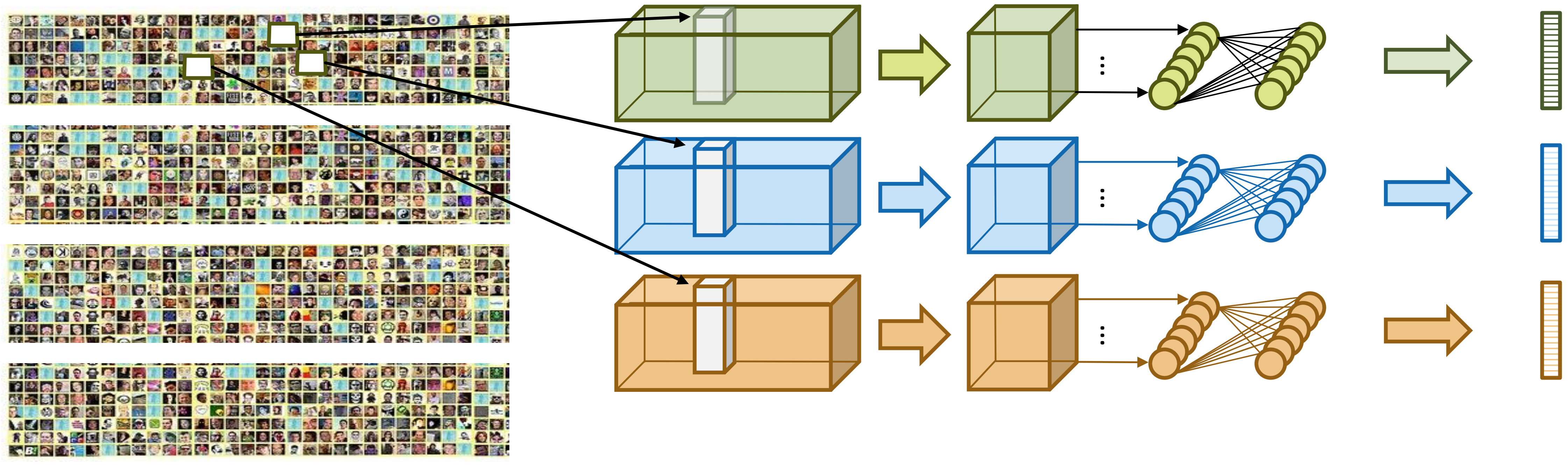
Data parallelism – limited by batch-size



- Simple and efficient solution, easy to implement
- Duplicate parameters at all processors
- Affects generalization

[Zhang et al. 1989]

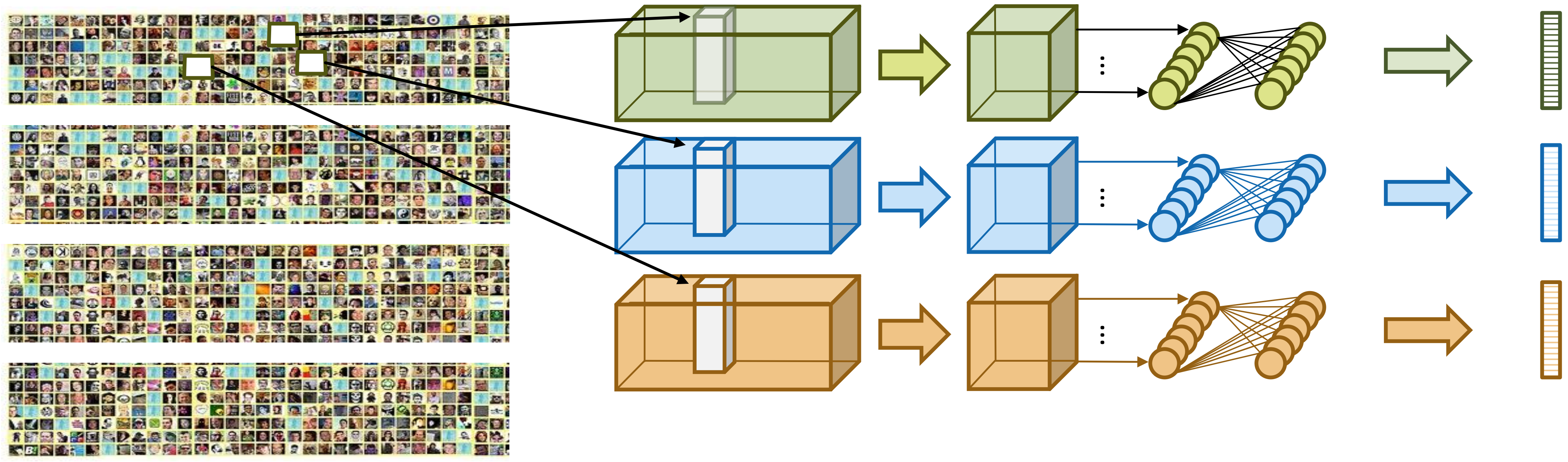
Data parallelism – limited by batch-size



- Simple and efficient solution, easy to implement
- Duplicate parameters at all processors
- Affects generalization

[Zhang et al. 1989]

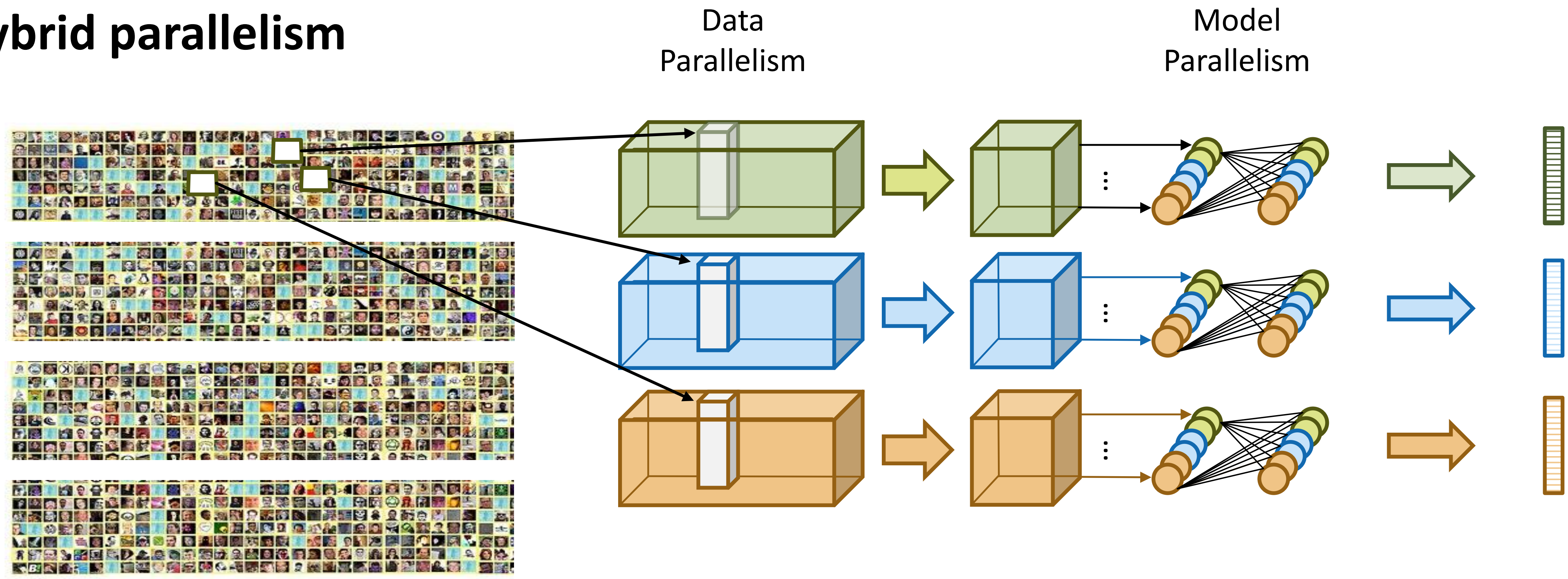
Data parallelism – limited by batch-size



- Simple and efficient solution, easy to implement
- Duplicate parameters at all processors
- Affects generalization

[Zhang et al. 1989]

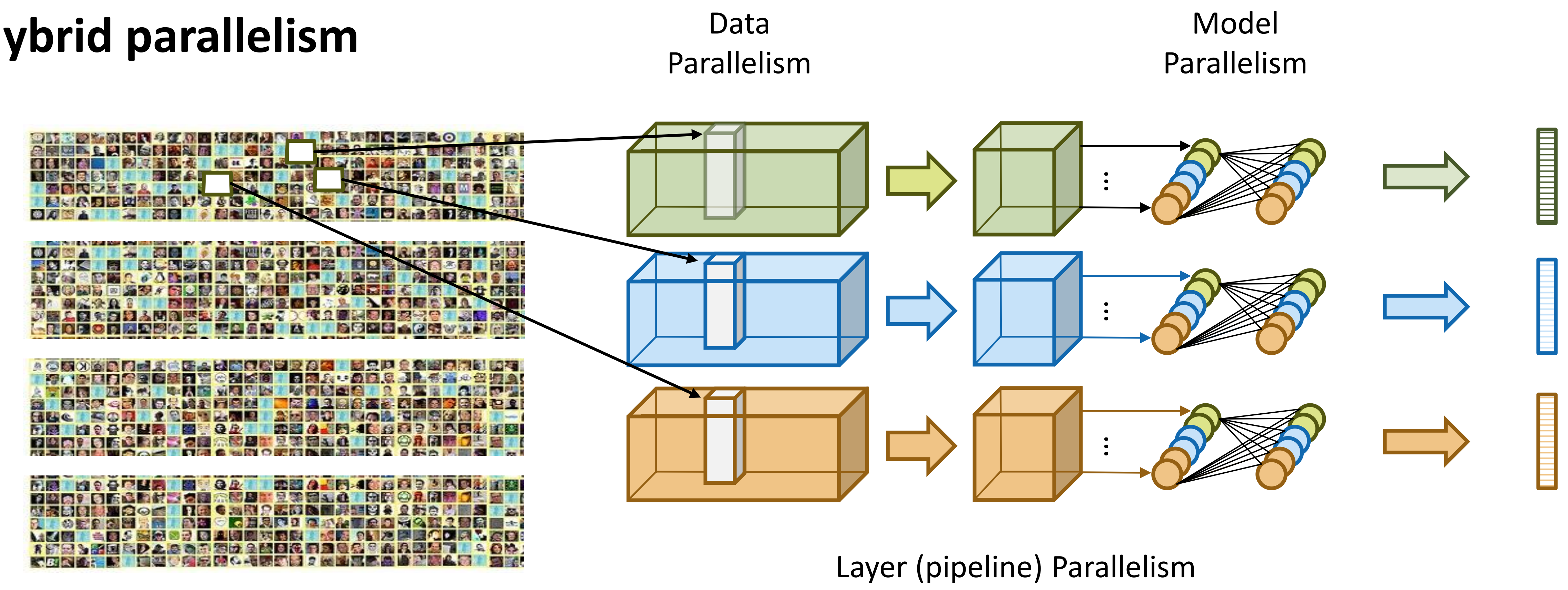
Hybrid parallelism



- Layers/parameters can be distributed across processors
- Can distribute minibatch
- Often specific to layer-types (e.g., distribute fc layers but handle conv layers data-parallel)
  - Enables arbitrary combinations of data, model, and pipeline parallelism – very powerful!

[Krizhevsky 2014; Dean et al. 2012; Ben-Nun & Hoefler 2019]

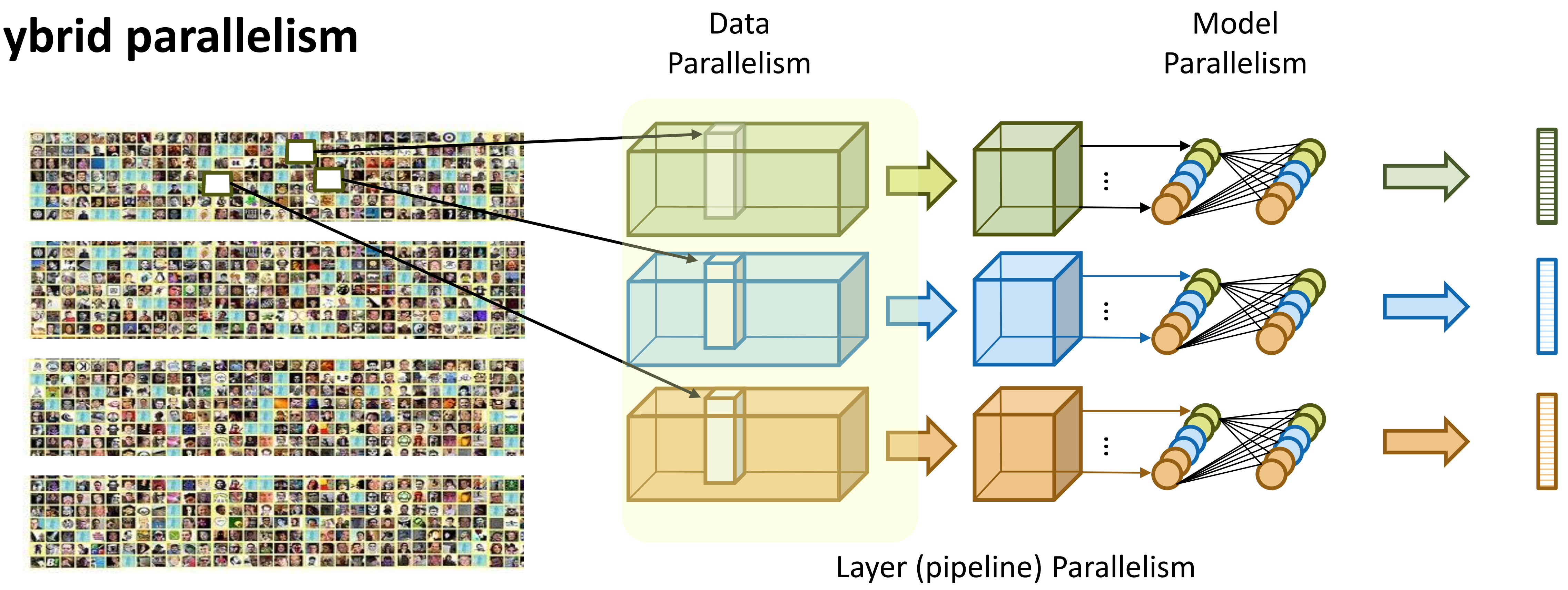
Hybrid parallelism



- **Layers/parameters can be distributed across processors**
- **Can distribute minibatch**
- **Often specific to layer-types (e.g., distribute fc layers but handle conv layers data-parallel)**
  - Enables arbitrary combinations of data, model, and pipeline parallelism – very powerful!

[Krizhevsky 2014; Dean et al. 2012; Ben-Nun & Hoefler 2019]

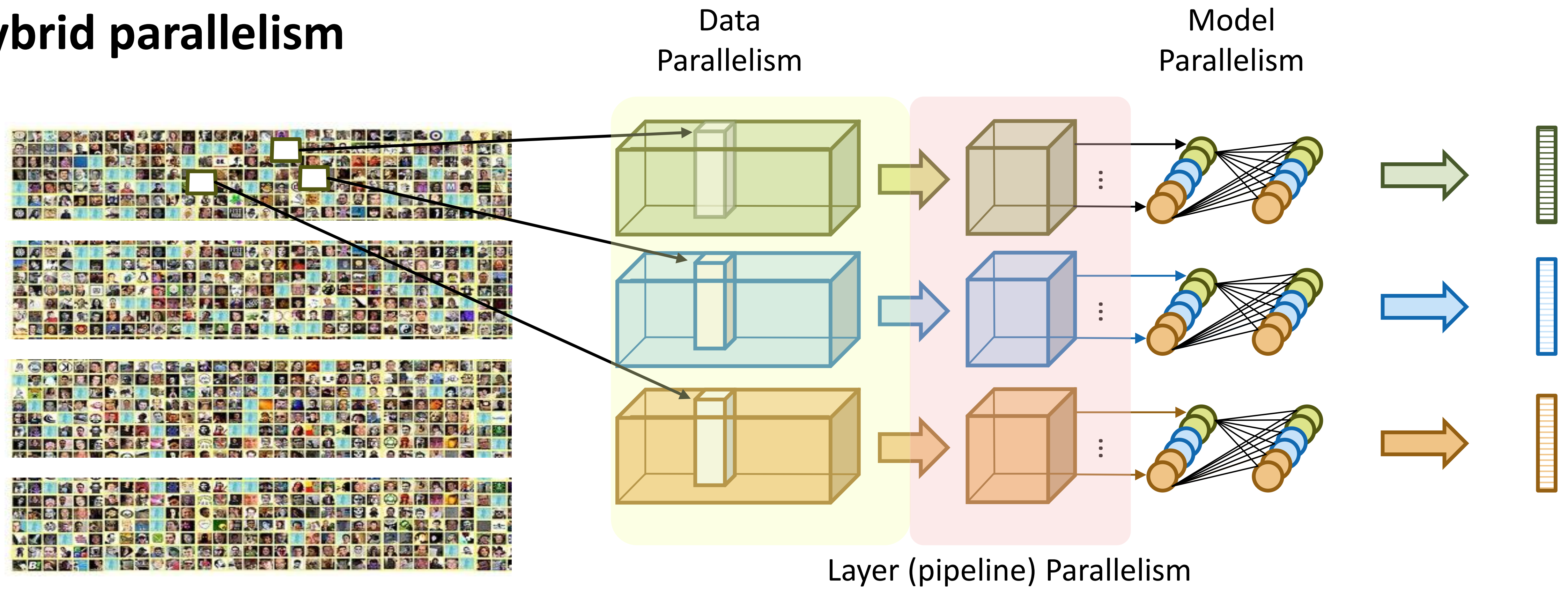
Hybrid parallelism



- **Layers/parameters can be distributed across processors**
- **Can distribute minibatch**
- **Often specific to layer-types (e.g., distribute fc layers but handle conv layers data-parallel)**
  - Enables arbitrary combinations of data, model, and pipeline parallelism – very powerful!

[Krizhevsky 2014; Dean et al. 2012; Ben-Nun & Hoefler 2019]

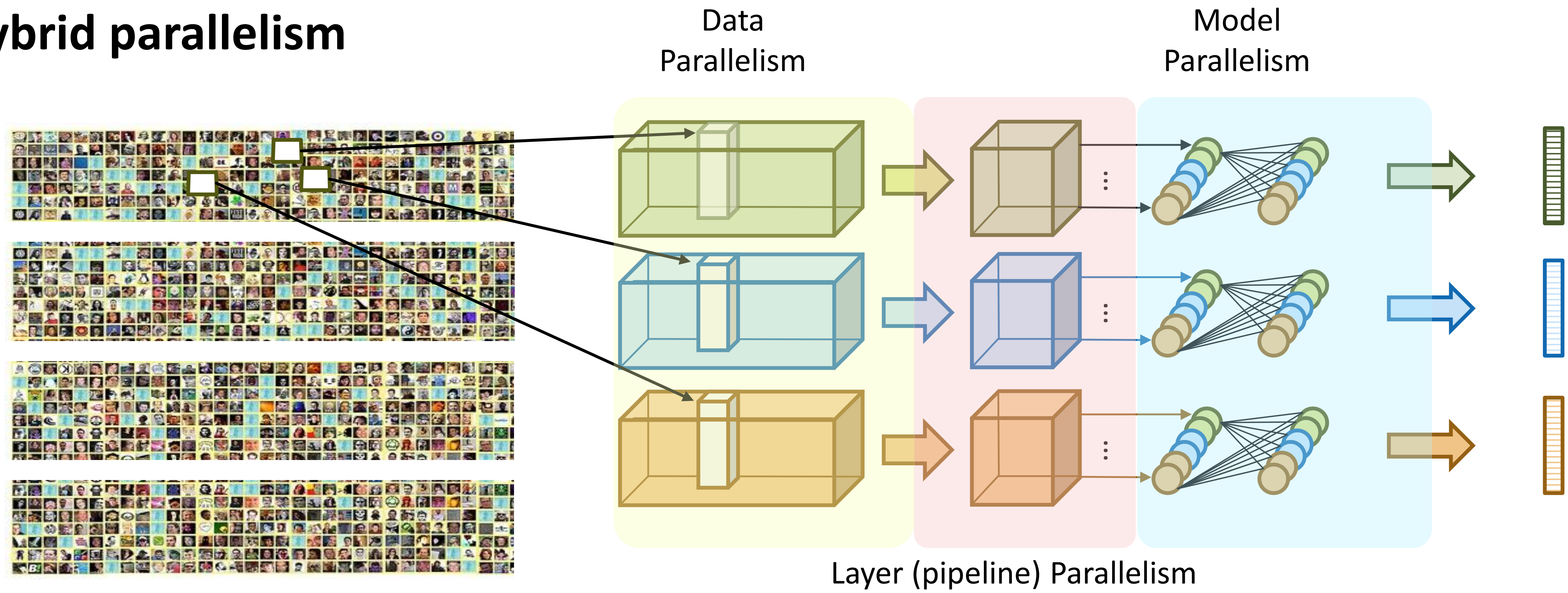
Hybrid parallelism



- **Layers/parameters can be distributed across processors**
- **Can distribute minibatch**
- **Often specific to layer-types (e.g., distribute fc layers but handle conv layers data-parallel)**
  - Enables arbitrary combinations of data, model, and pipeline parallelism – very powerful!

[Krizhevsky 2014; Dean et al. 2012; Ben-Nun & Hoefler 2019]

Hybrid parallelism



- **Layers/parameters can be distributed across processors**
- **Can distribute minibatch**
- **Often specific to layer-types (e.g., distribute fc layers but handle conv layers data-parallel)**
  - Enables arbitrary combinations of data, model, and pipeline parallelism – very powerful!

[Krizhevsky 2014; Dean et al. 2012; Ben-Nun & Hoefler 2019]

Other ways to think about parallelism

Other ways to think about parallelism

- **All definitions are fuzzy** **(including this ☺)**

- **Data-, model-, pipeline-, hybrid-parallelism**

- **Weak vs strong scaling**
  - What do you keep the same vs what do change?
  - Mini-batch weak scaling: grow the mini-batch
  - Mini-batch model scaling: grow the model size (not so useful in general…)
  - Strong scaling: Fix everything, use more GPUs

- **For convolution: based on partitioned tensor dimensions**
  - Sample-, spatial-, channel-, filter-parallelism

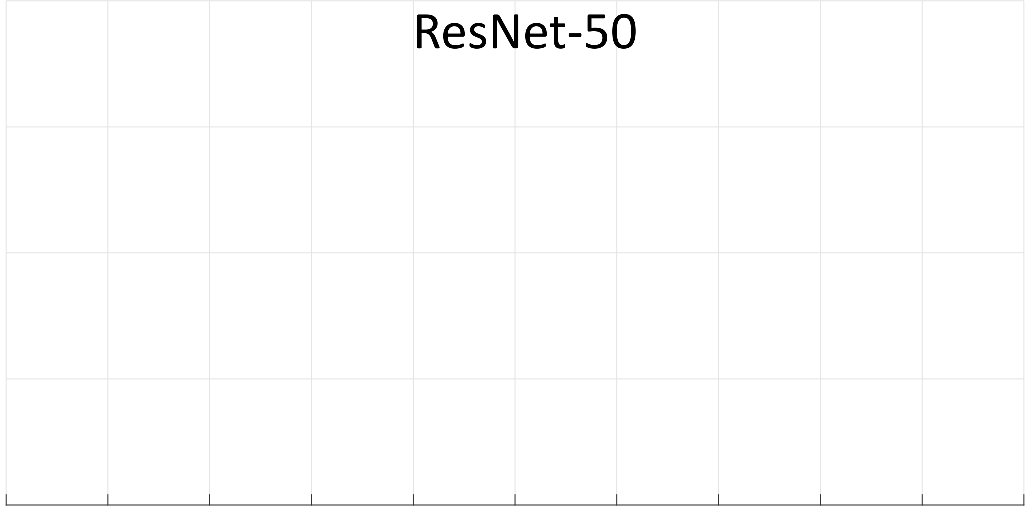
Large mini-batches

- **Make the mini-batch really big!**

[Goyal et al. 2017; Shallue et al. 2019]

Large mini-batches

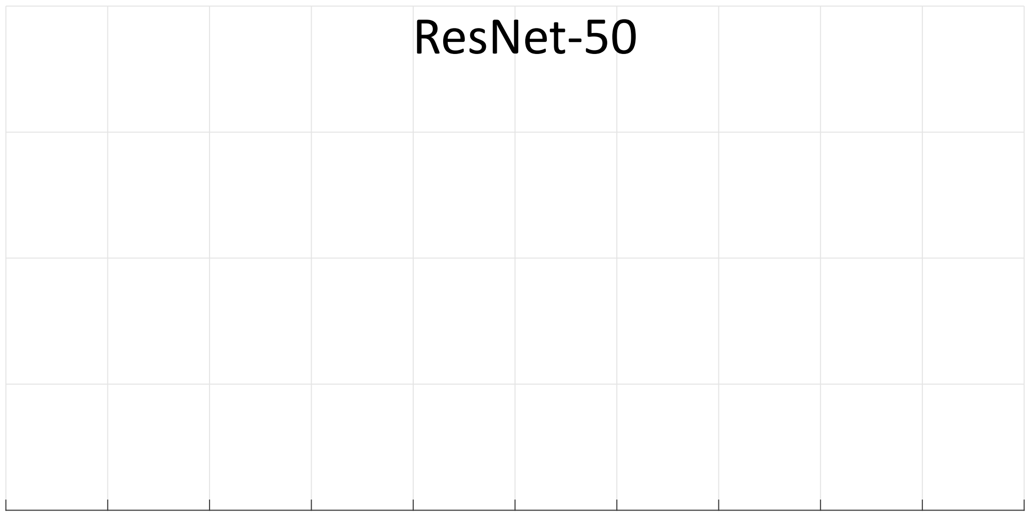
- **Make the mini-batch really big!**

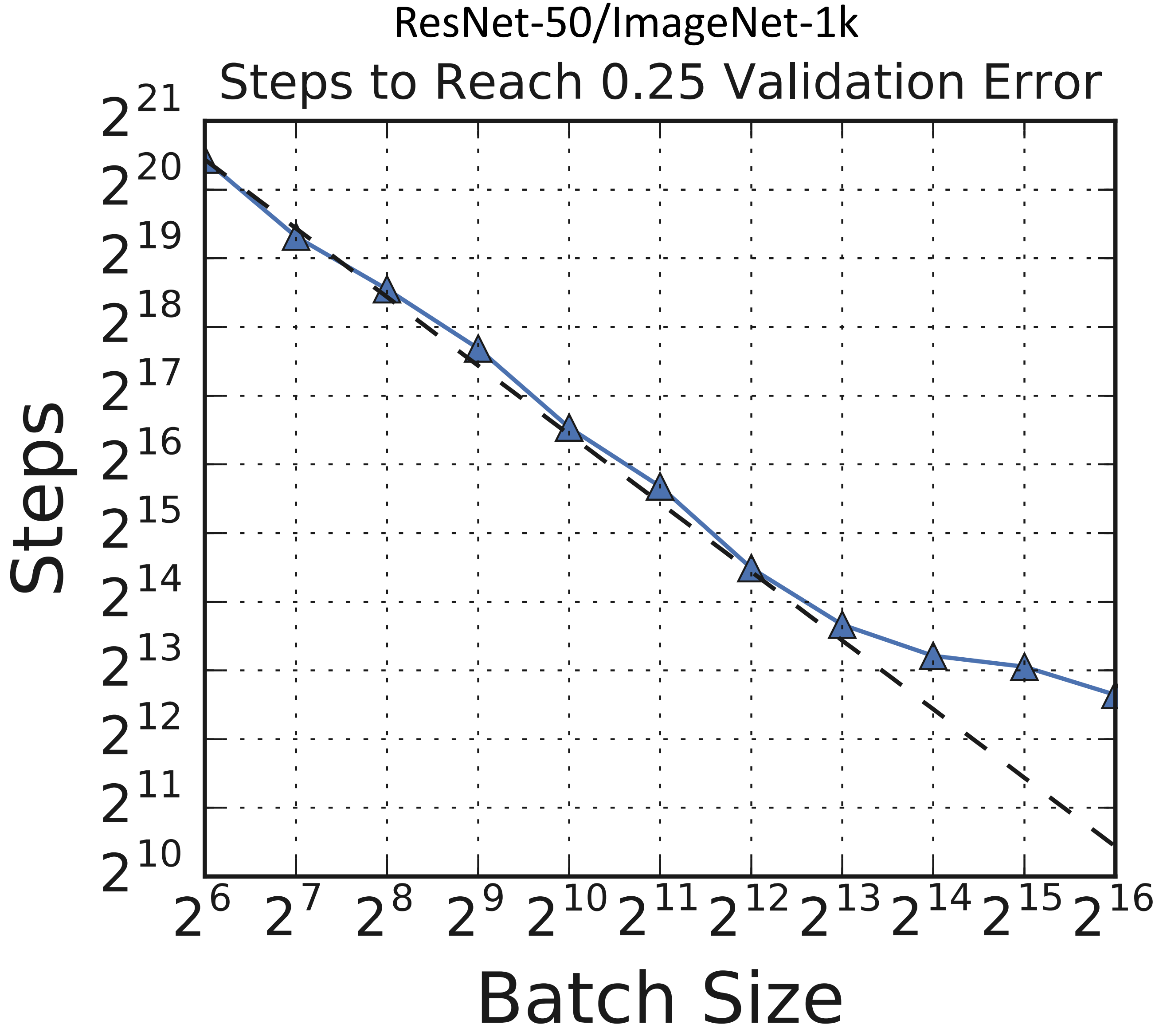


[Goyal et al. 2017; Shallue et al. 2019]

Large mini-batches

- **Make the mini-batch really big!**

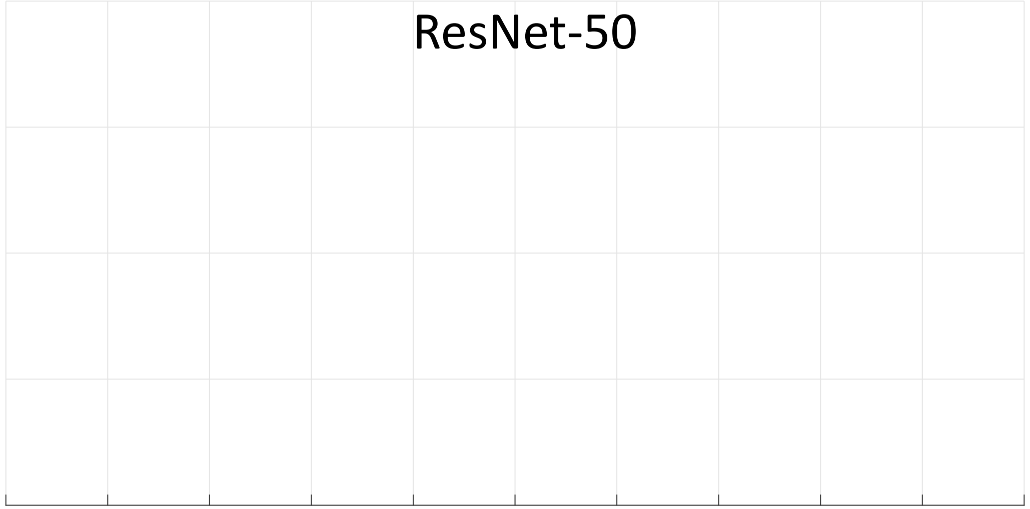


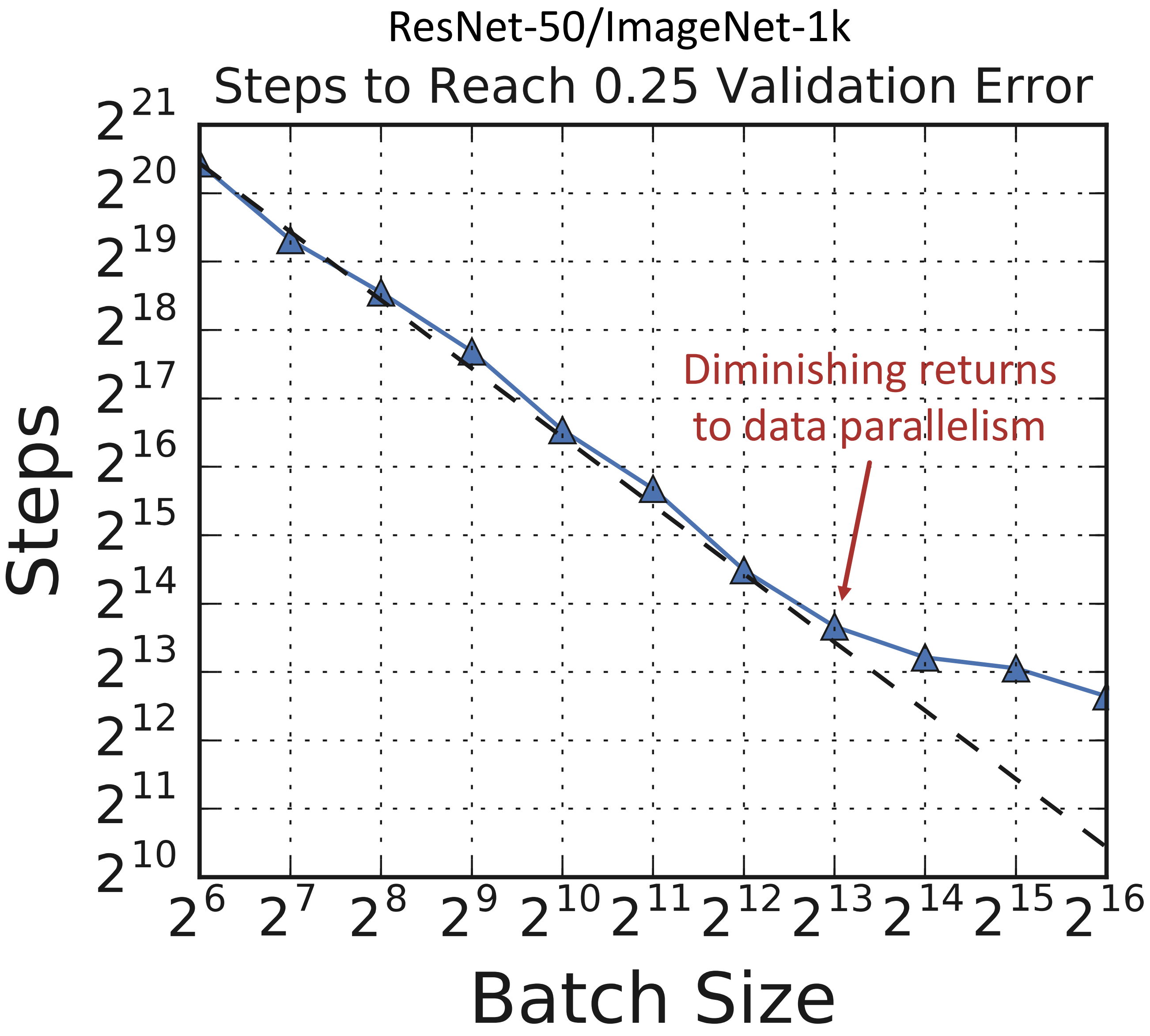


[Goyal et al. 2017; Shallue et al. 2019]

Large mini-batches

- **Make the mini-batch really big!**

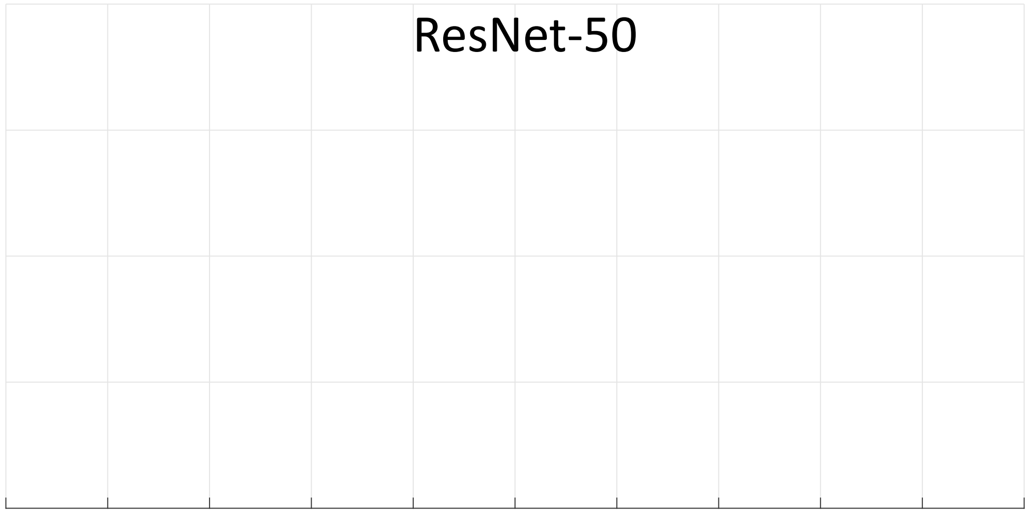


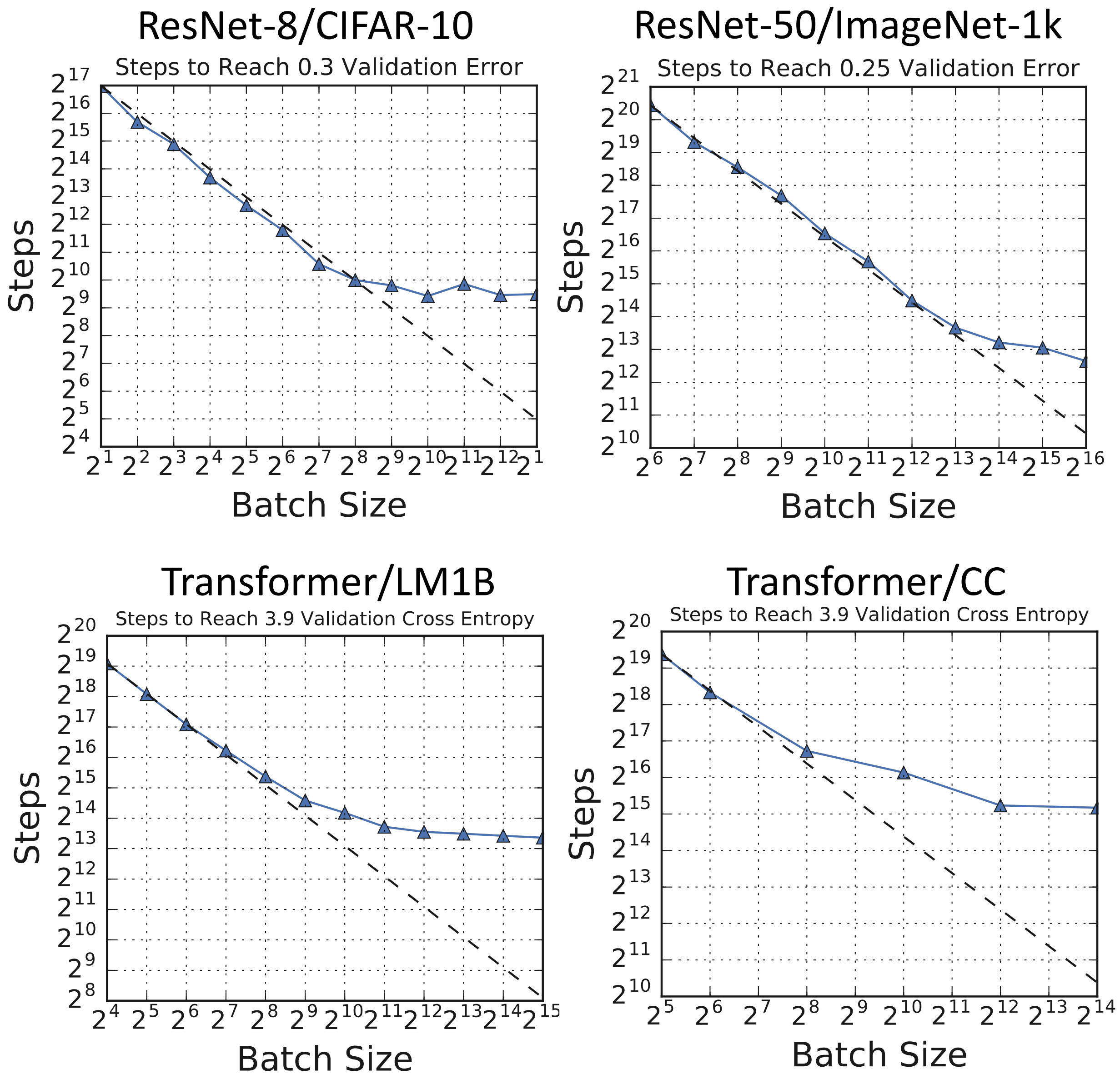


[Goyal et al. 2017; Shallue et al. 2019]

Large mini-batches

- **Make the mini-batch really big!**

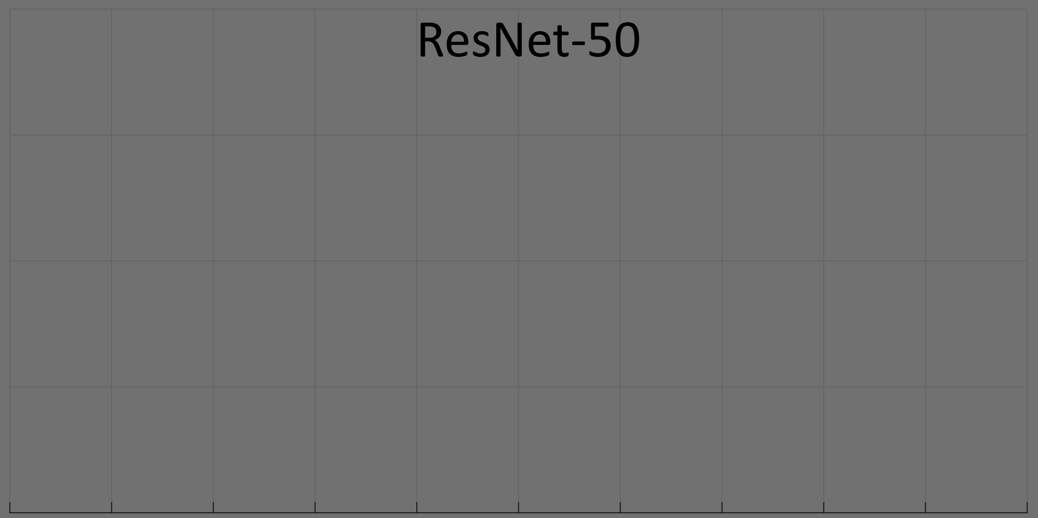


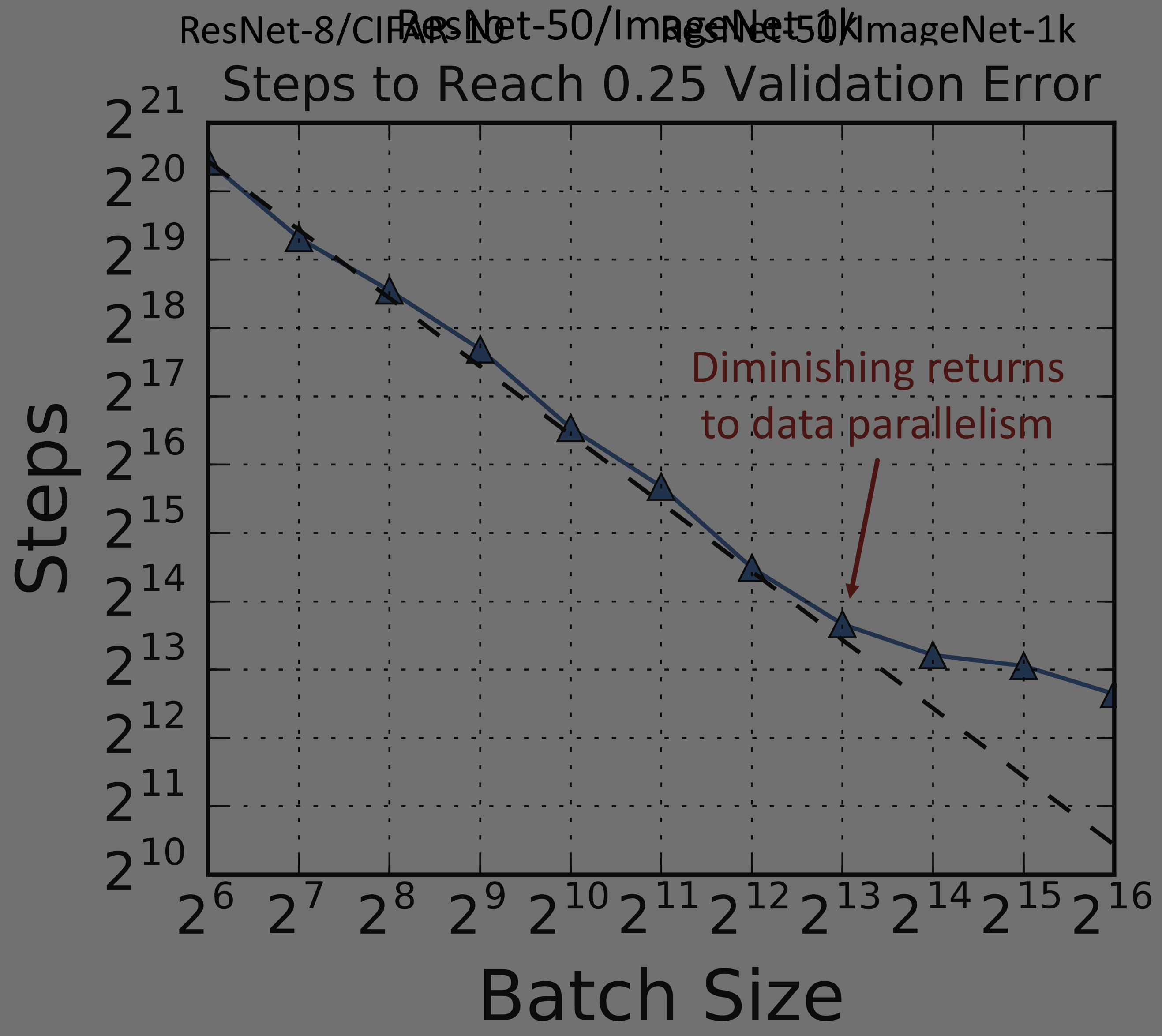


[Goyal et al. 2017; Shallue et al. 2019]

Large mini-batches

- **Make the mini-batch really big!**



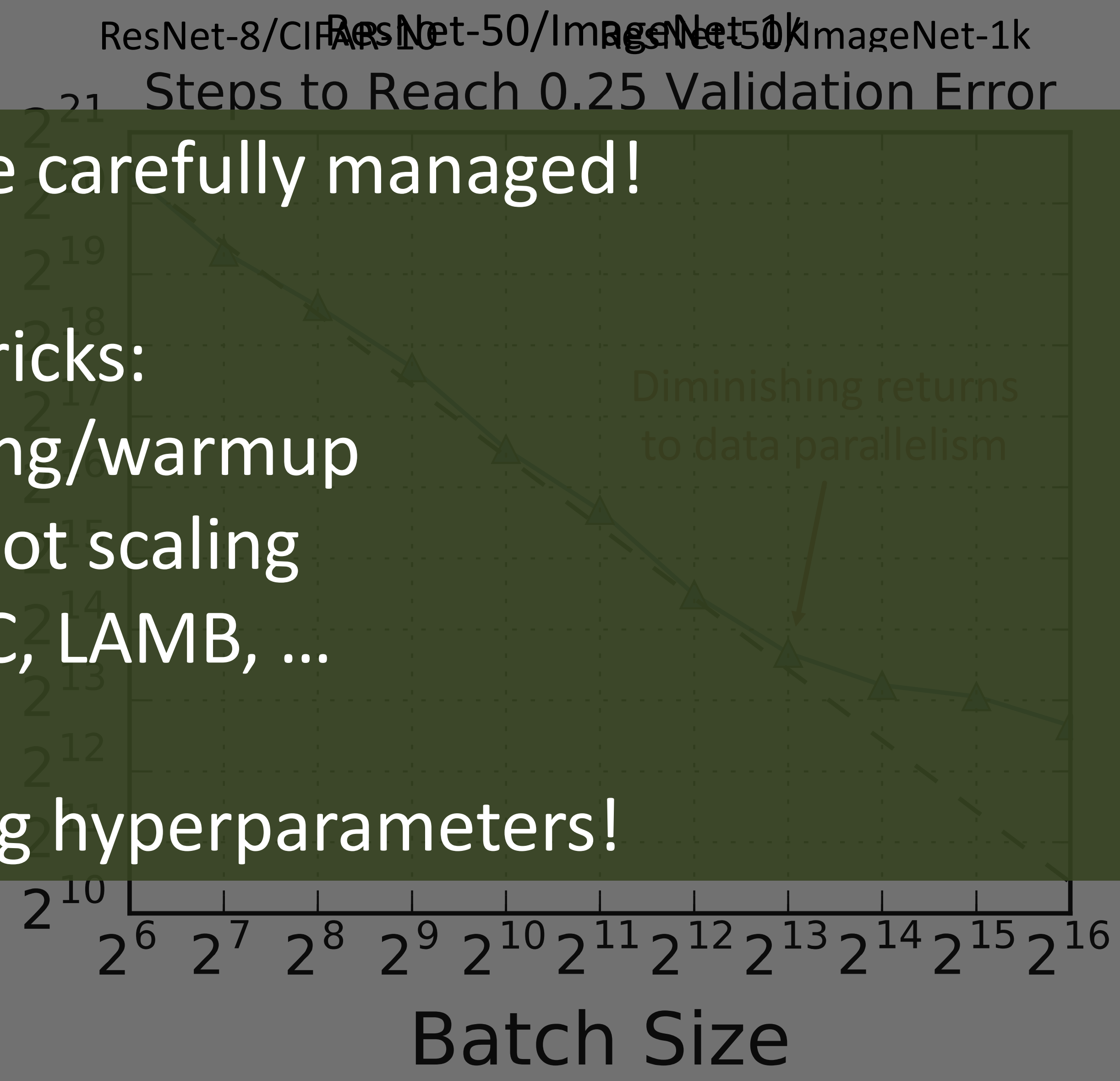


[Goyal et al. 2017; Shallue et al. 2019]

Large mini-batches

- **Make the mini-batch really big!**

ResNet-50



Mini-batch size must be carefully managed!

Some tricks:

- Linear scaling/warmup
- Square root scaling
- LARS, LARC, LAMB, …

Often requires retuning hyperparameters!

[Goyal et al. 2017; Shallue et al. 2019]

Collectives for deep learning

Collectives for deep learning

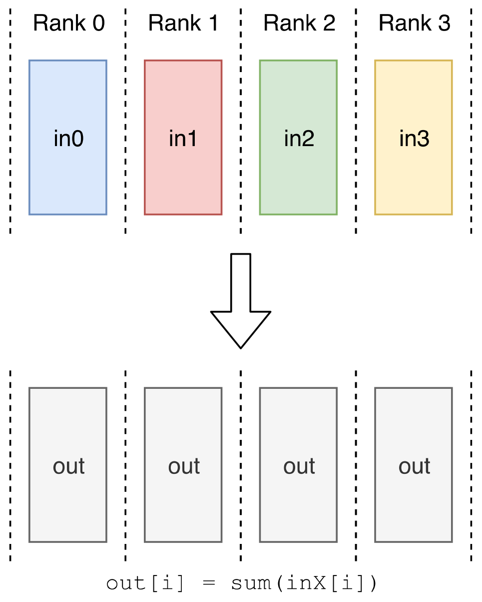
- **Certain communication patterns can be optimized**
- **People keep reinventing MPI**
  - Baidu Allreduce, NCCL, Gloo, Horovod, …
- **What we need (for this talk):**

Collectives for deep learning

- **Certain communication patterns can be optimized**
- **People keep reinventing MPI**
  - Baidu Allreduce, NCCL, Gloo, Horovod, …

- **What we need (for this talk):**

Allreduce

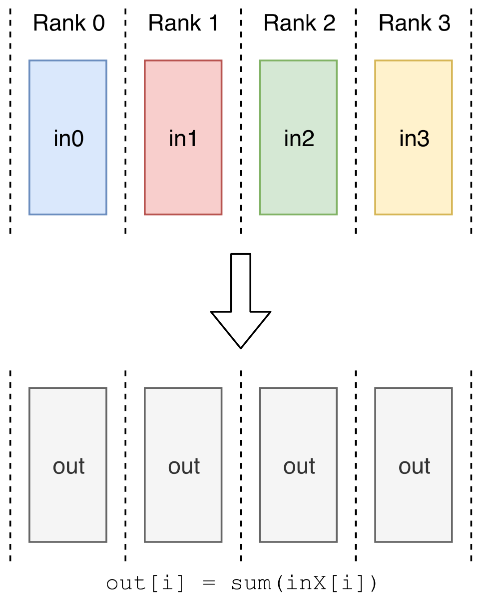


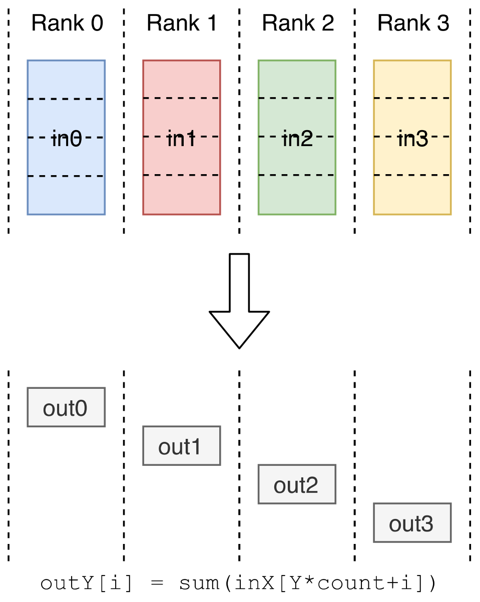
Collectives for deep learning

- **Certain communication patterns can be optimized**
- **People keep reinventing MPI**
  - Baidu Allreduce, NCCL, Gloo, Horovod, …

- **What we need (for this talk):**

Allreduce



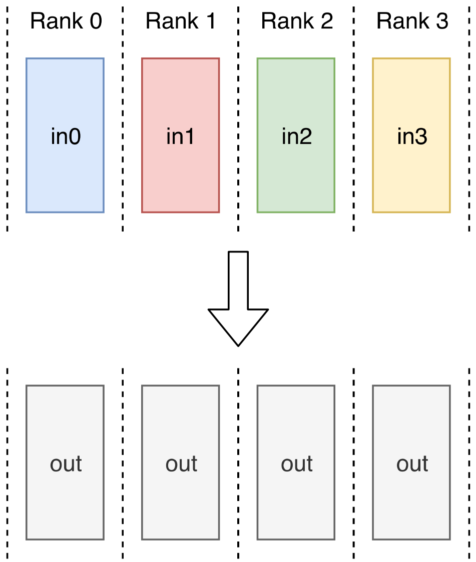


Collectives for deep learning

- **Certain communication patterns can be optimized**
- **People keep reinventing MPI**
  - Baidu Allreduce, NCCL, Gloo, Horovod, …

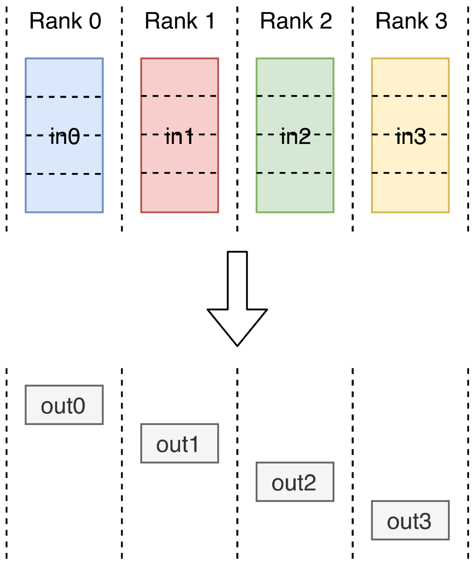
- **What we need (for this talk):**

Allreduce

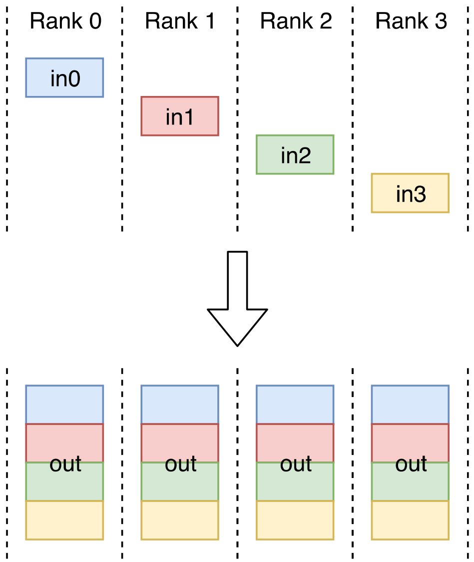


`out[i] = sum(inX[i])`

Reduce-scatter



`outY[i] = sum(inX[Y*count+i])`



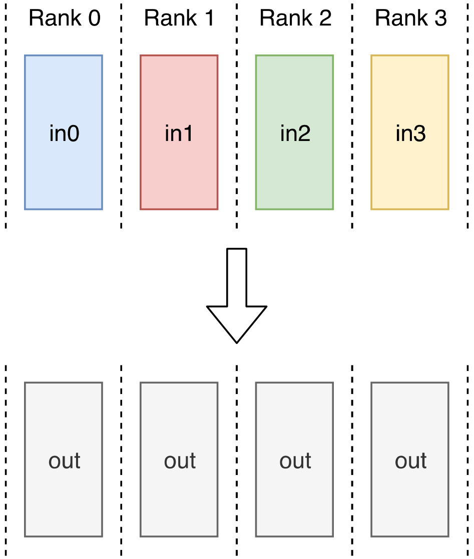
`out[Y*count+i] = inY[i]`

Collectives for deep learning

- **Certain communication patterns can be optimized**
- **People keep reinventing MPI**
  - Baidu Allreduce, NCCL, Gloo, Horovod, …

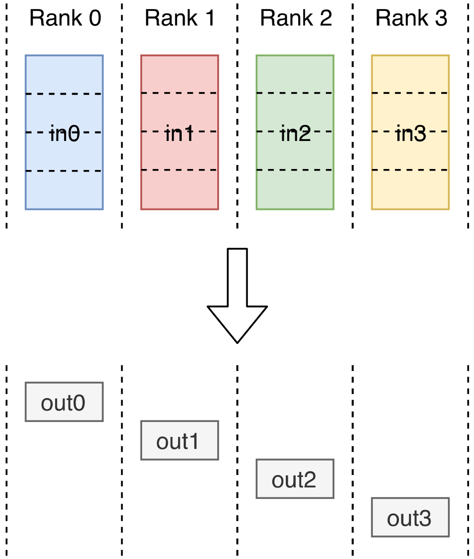
- **What we need (for this talk):**

Allreduce

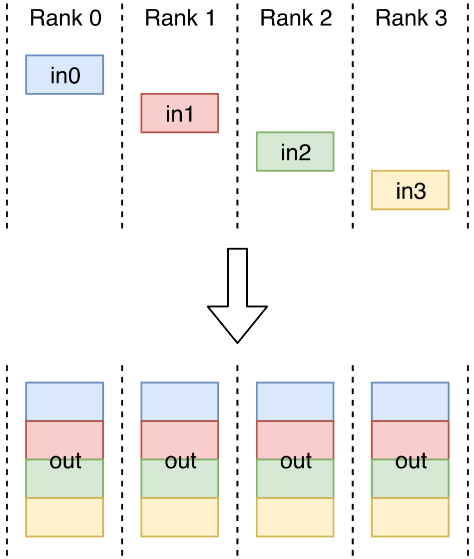


`out[i] = sum(inX[i])`

Reduce-scatter



`outY[i] = sum(inX[Y*count+i])`

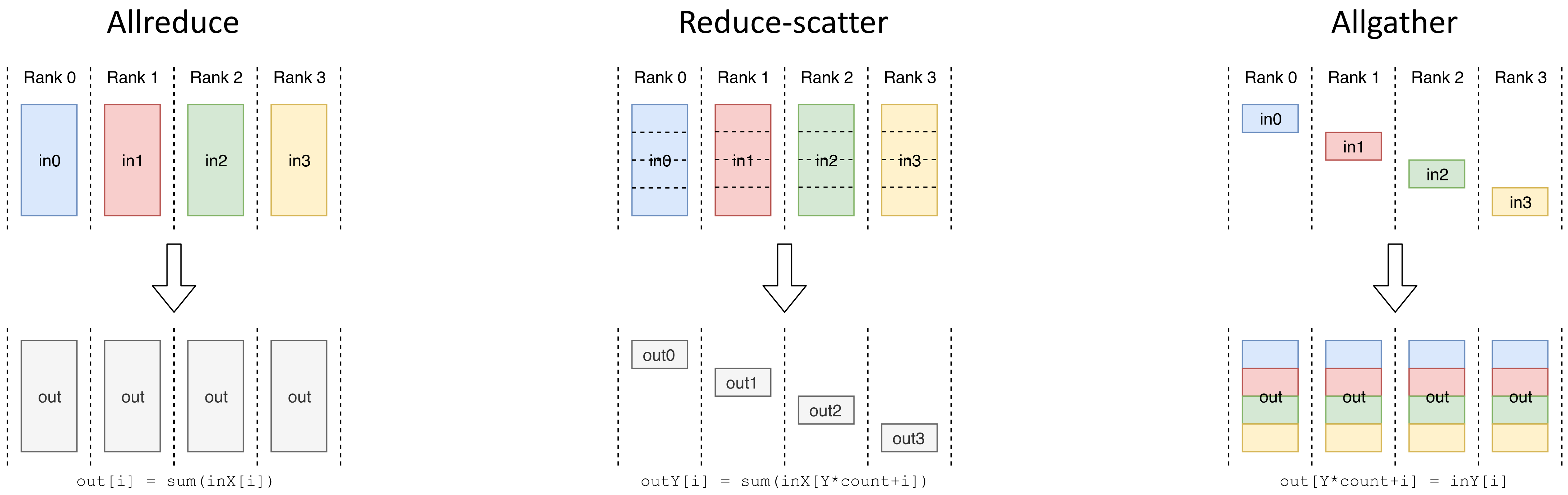


`out[Y*count+i] = inY[i]`

Collectives for deep learning

- **Certain communication patterns can be optimized**
- **People keep reinventing MPI**
  - Baidu Allreduce, NCCL, Gloo, Horovod, …

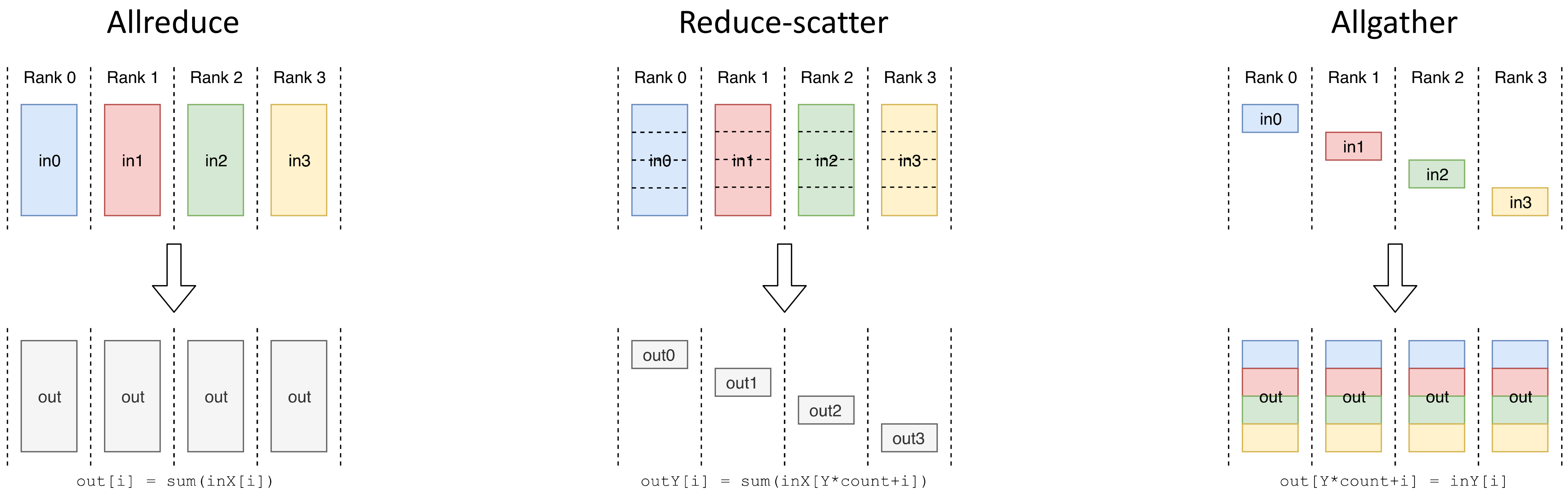
- **What we need (for this talk):**



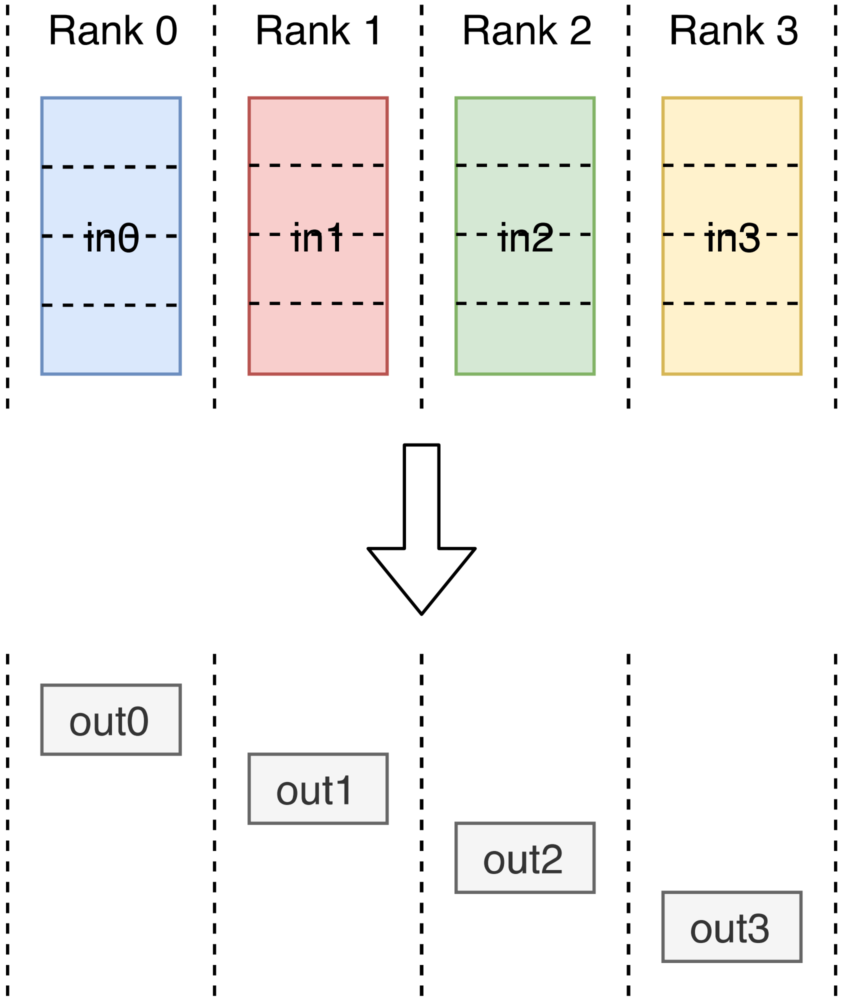
Collectives for deep learning

- **Certain communication patterns can be optimized**
- **People keep reinventing MPI**
  - Baidu Allreduce, NCCL, Gloo, Horovod, …

- **What we need (for this talk):**



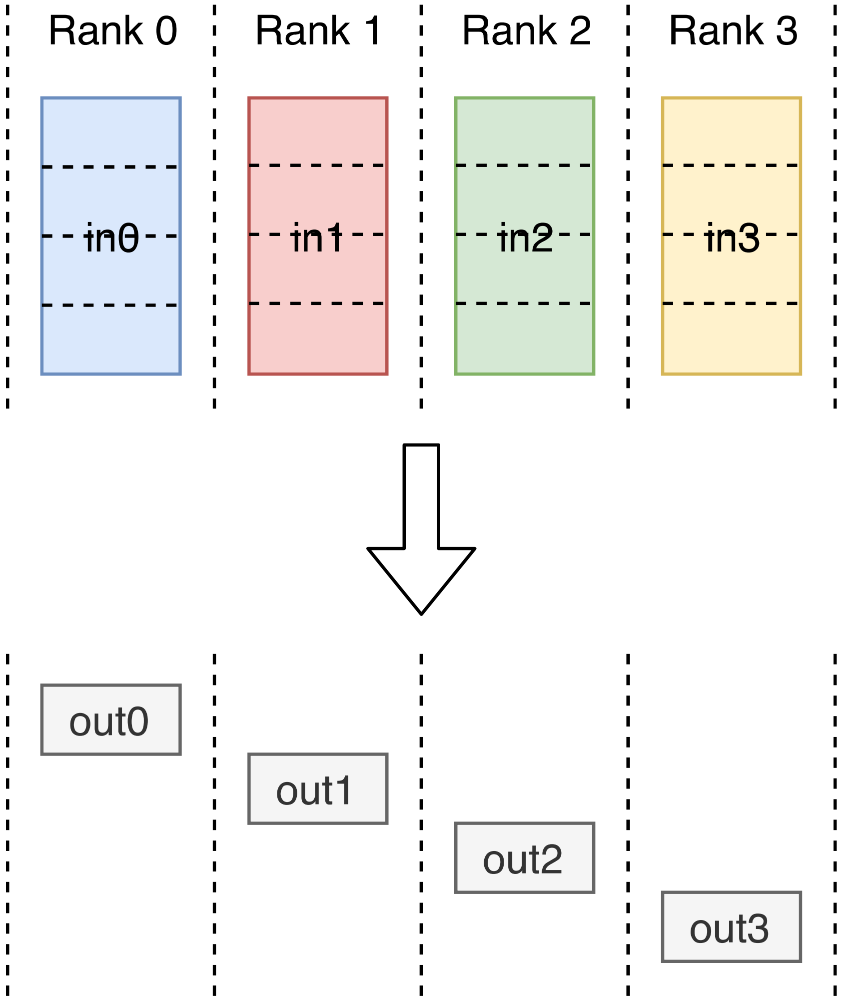
Reduce-scatter



`outY[i] = sum(inX[Y*count+i])`

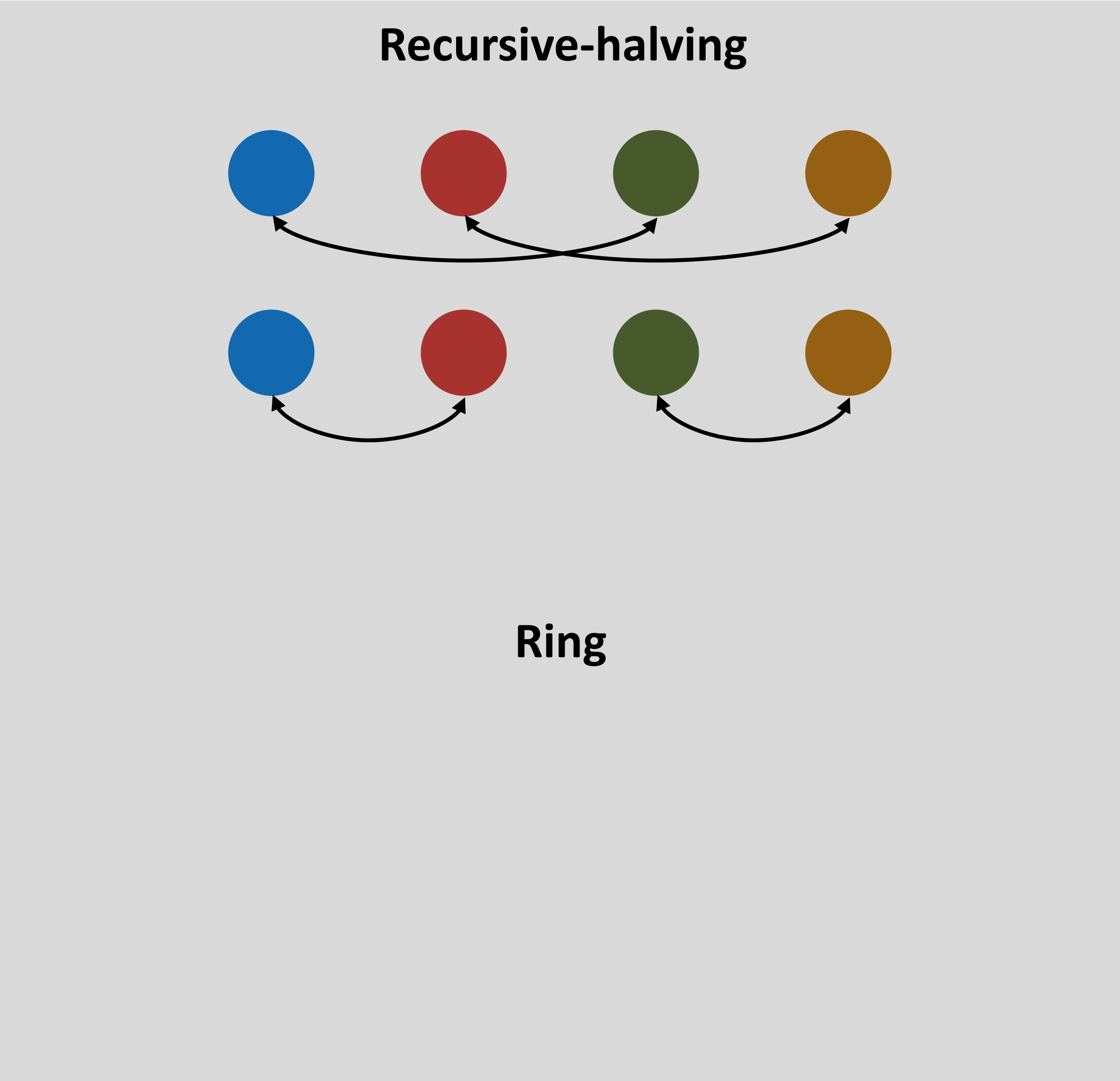
Reduce-scatter

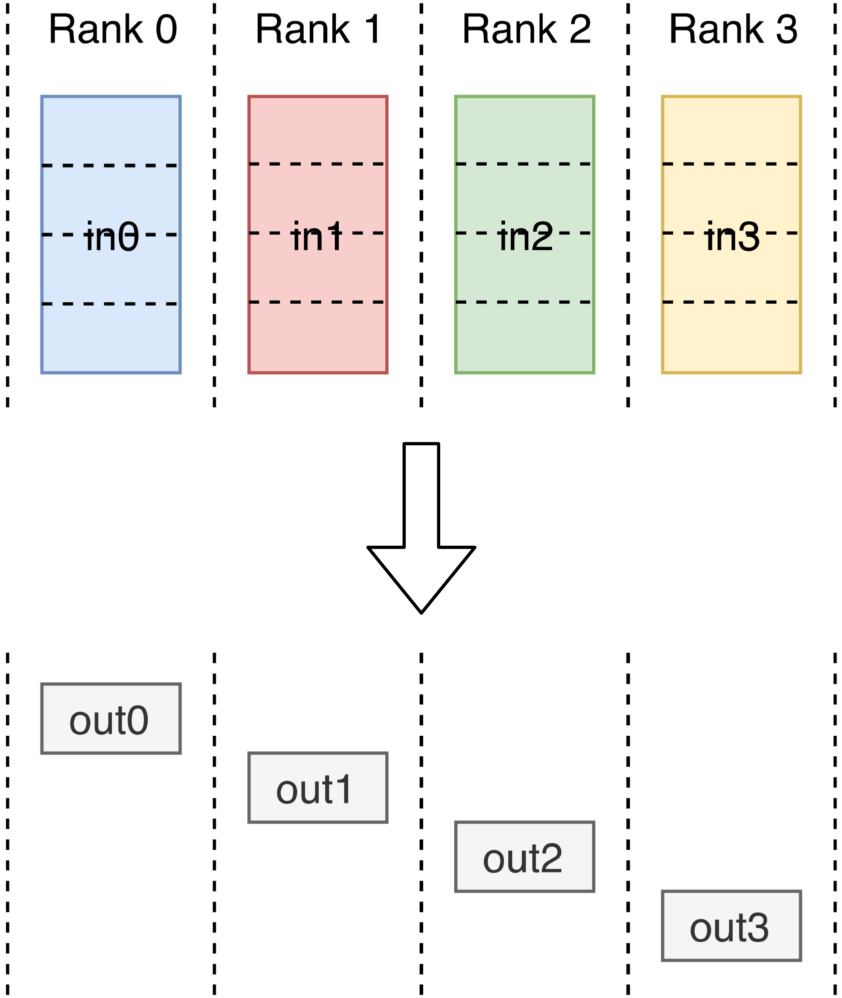
Recursive-halving



outY[i] = sum(inX[Y*count+i])

Reduce-scatter

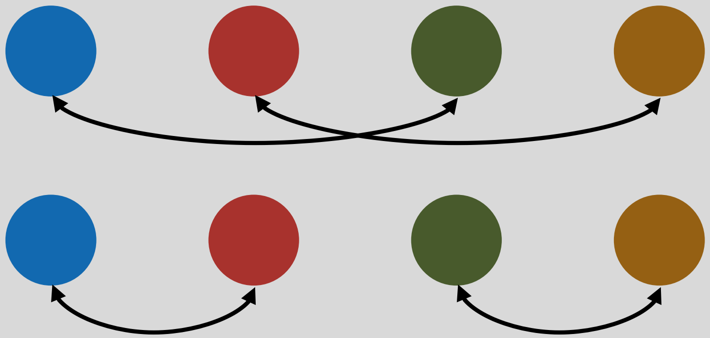




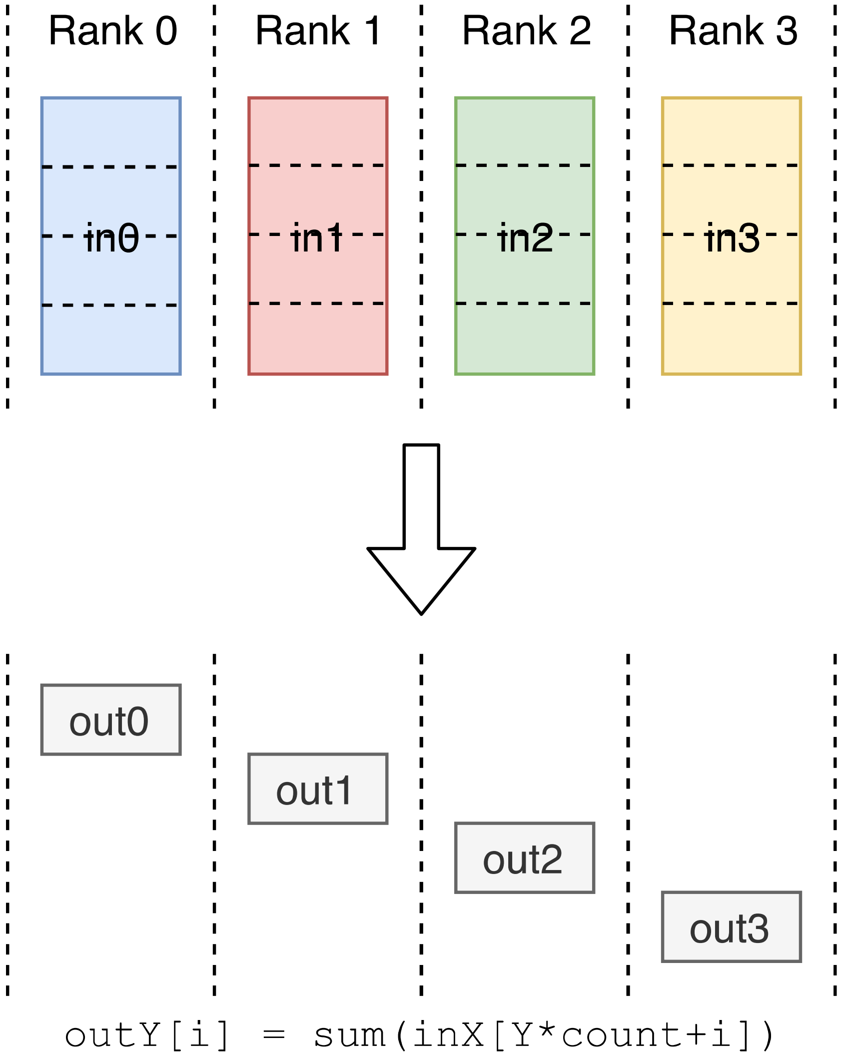
`outY[i] = sum(inX[Y*count+i])`

Reduce-scatter

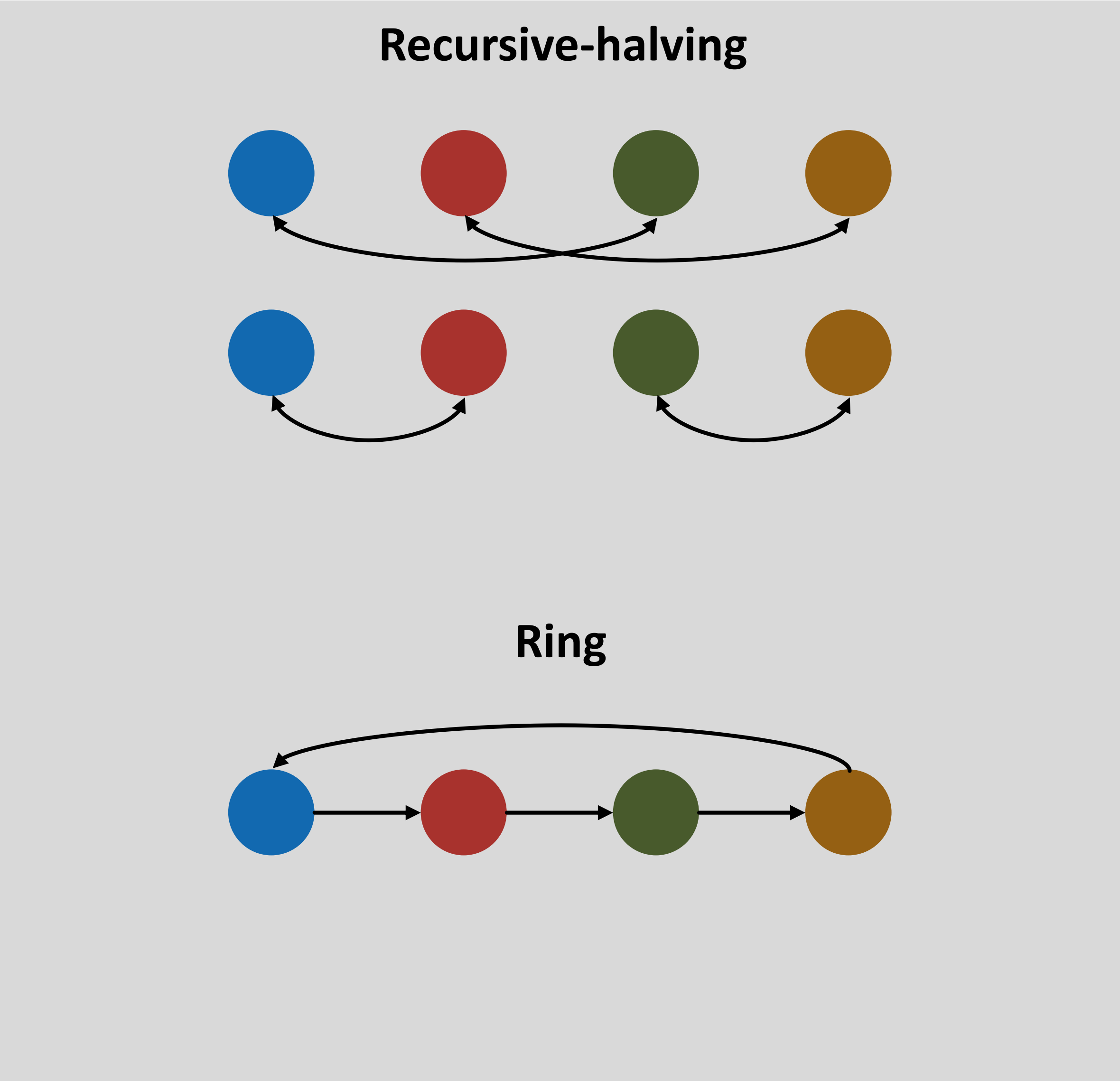
Recursive-halving

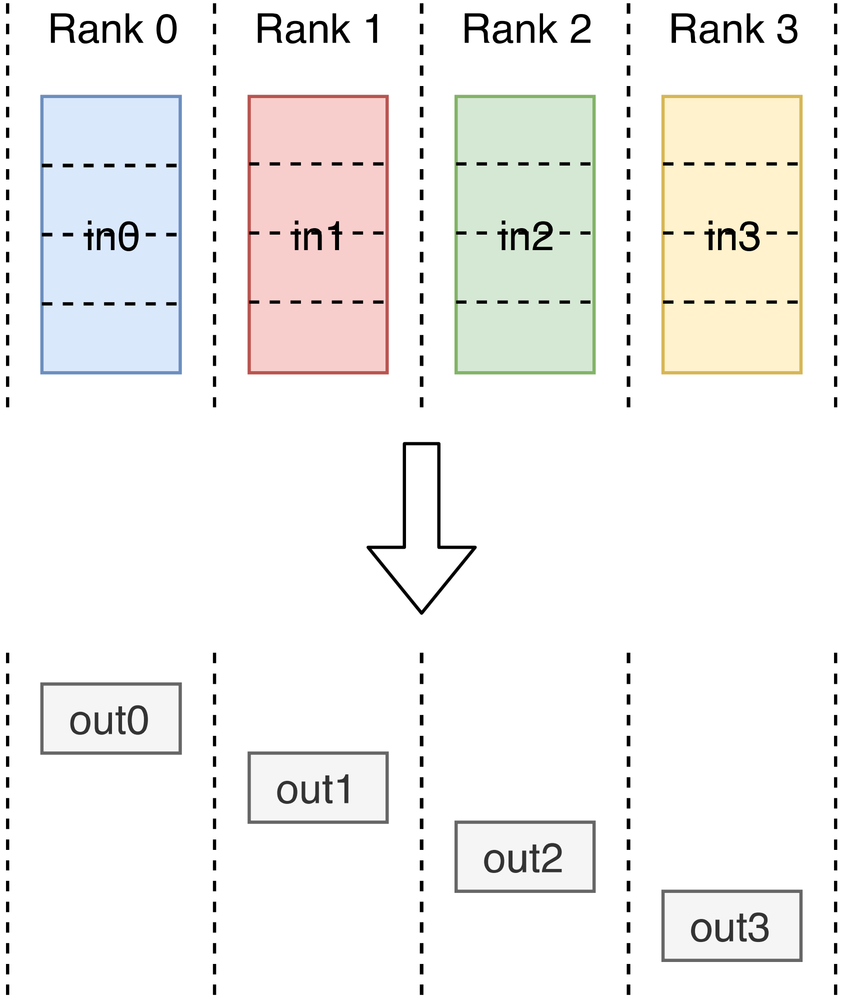


Ring



Reduce-scatter

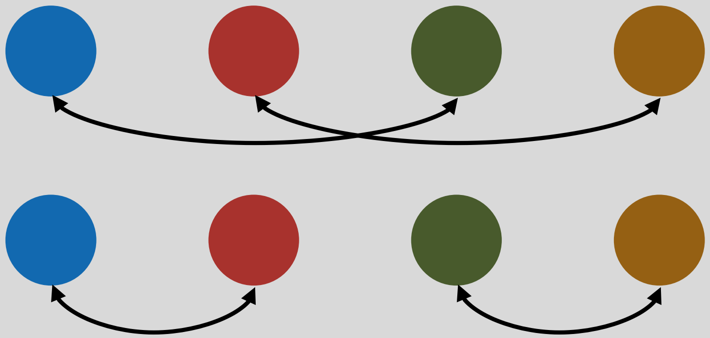




`outY[i] = sum(inX[Y*count+i])`

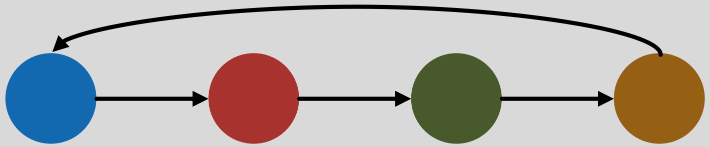
Reduce-scatter

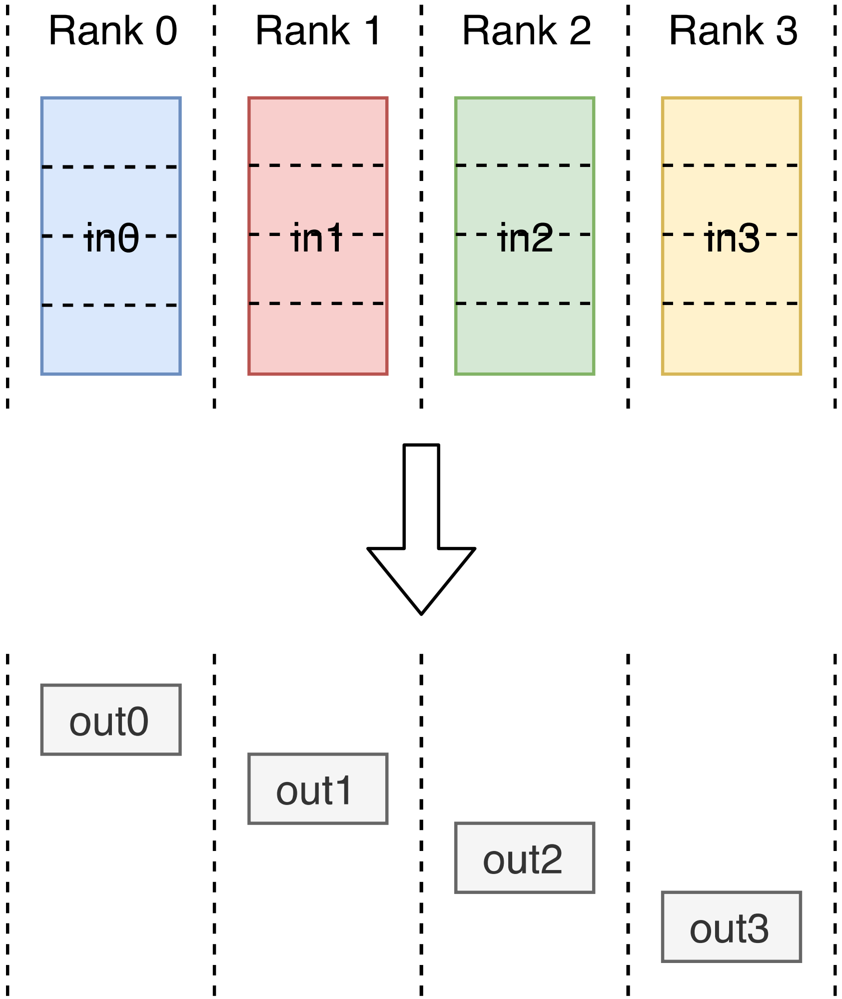
Recursive-halving



$$\alpha \lg p + \frac{p-1}{p} n\beta + \frac{p-1}{p} n\gamma$$

Ring

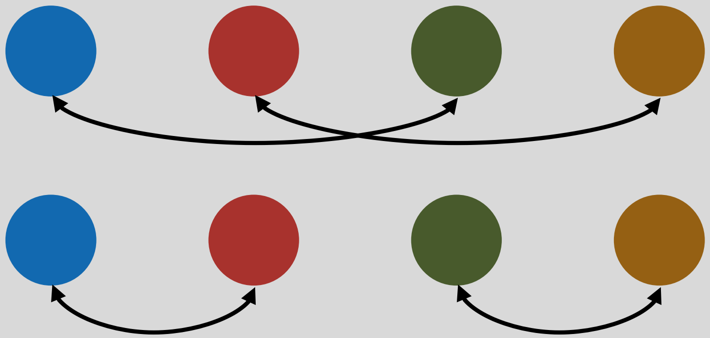




`outY[i] = sum(inX[Y*count+i])`

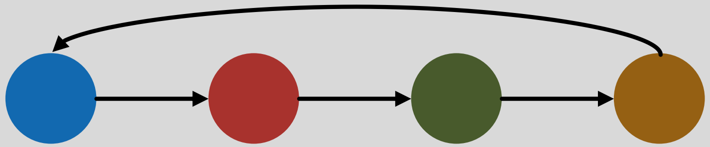
Reduce-scatter

Recursive-halving

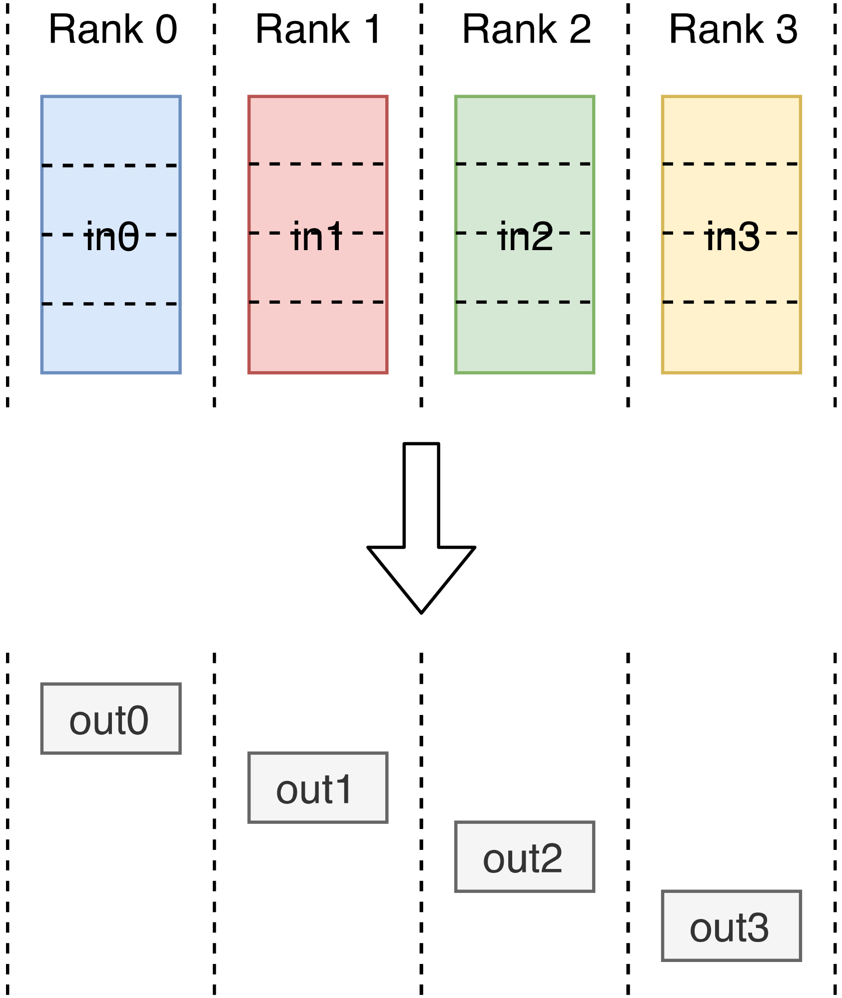


$$\alpha \lg p + \frac{p-1}{p} n\beta + \frac{p-1}{p} n\gamma$$

Ring

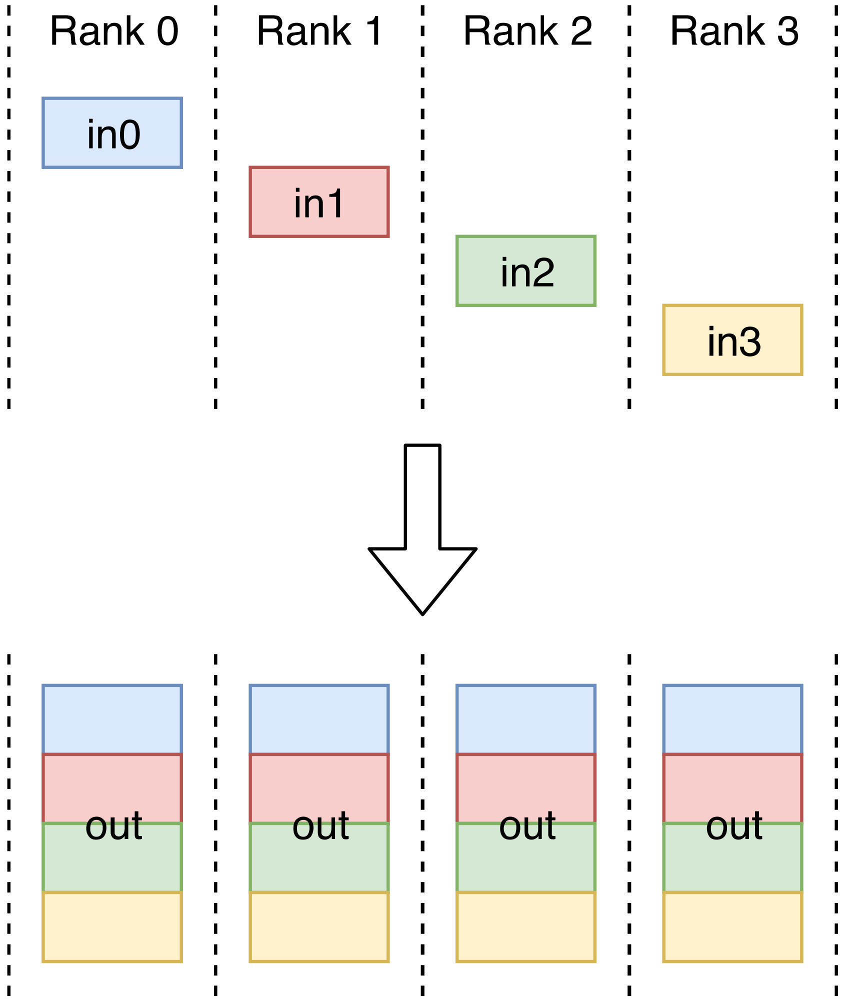


$$(p-1)\alpha + \frac{p-1}{p} n\beta + \frac{p-1}{p} n\gamma$$




```
outY[i] = sum(inX[Y*count+i])
```

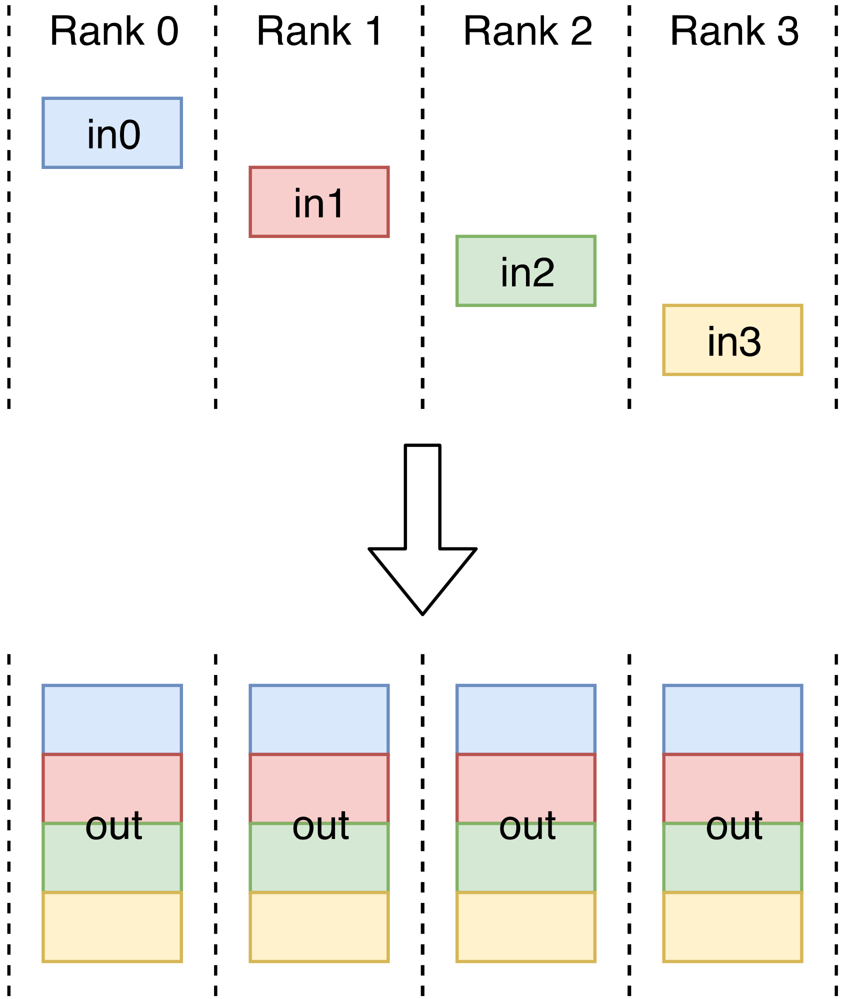
Allgather



`out[Y*count+i] = inY[i]`

Allgather

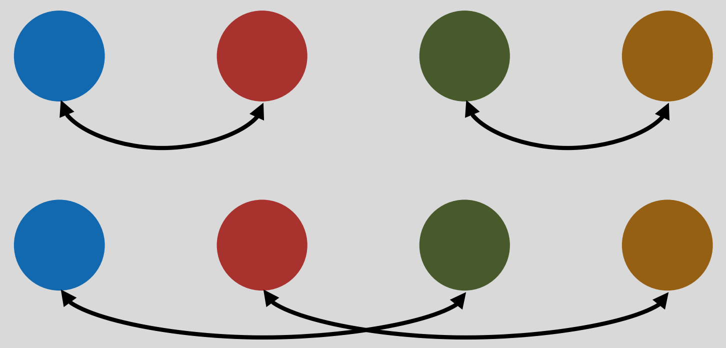
Recursive-doubling



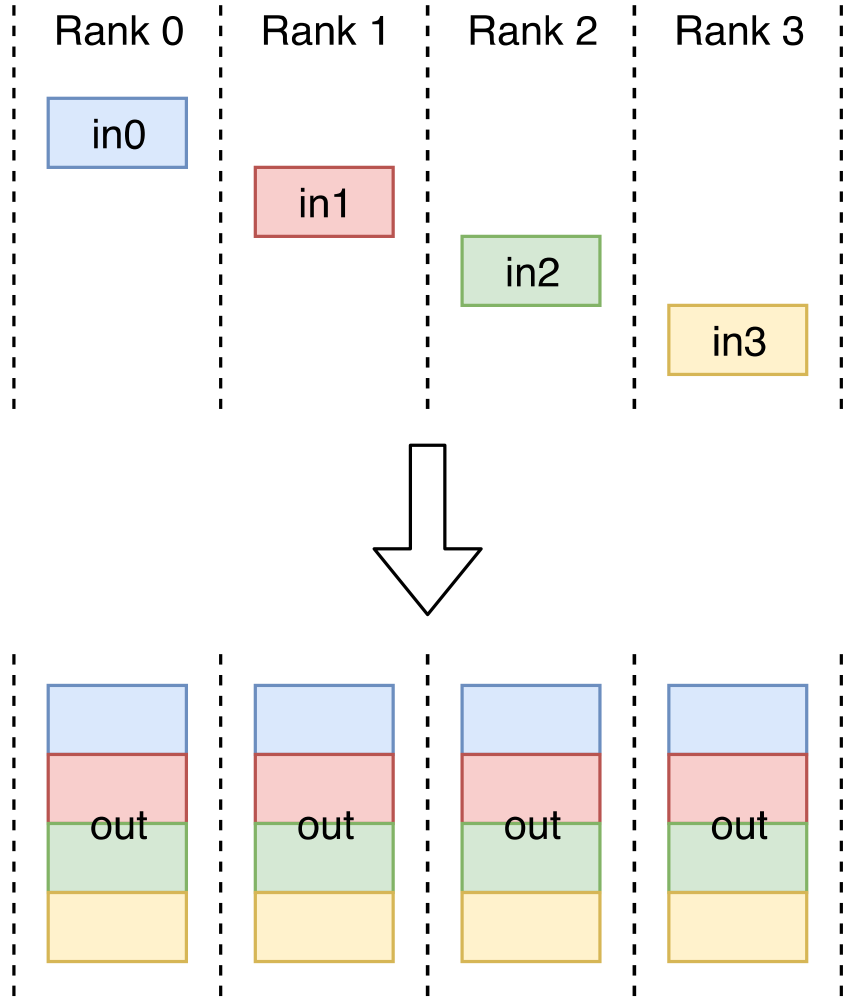
`out[Y*count+i] = inY[i]`

Allgather

Recursive-doubling

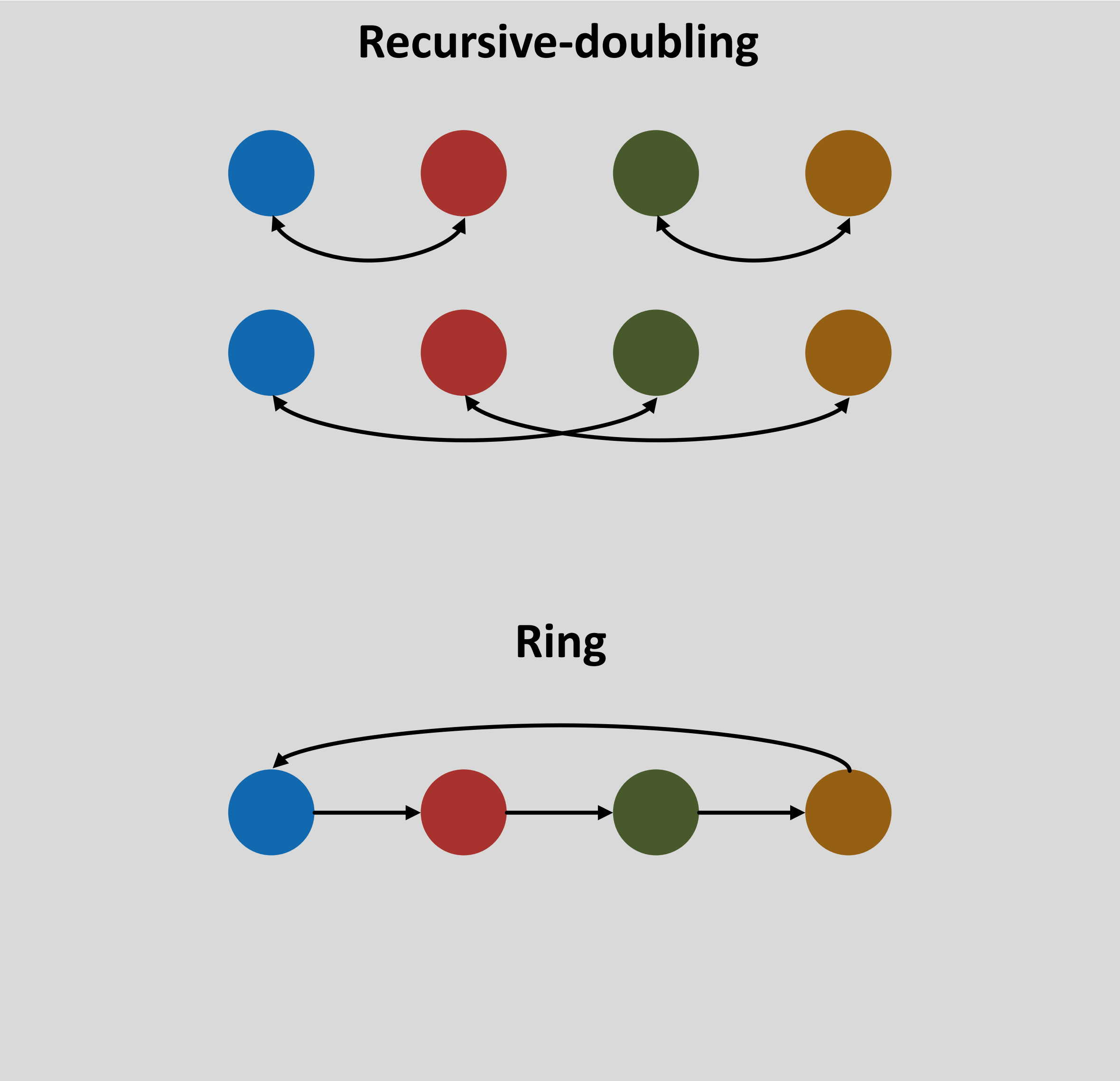


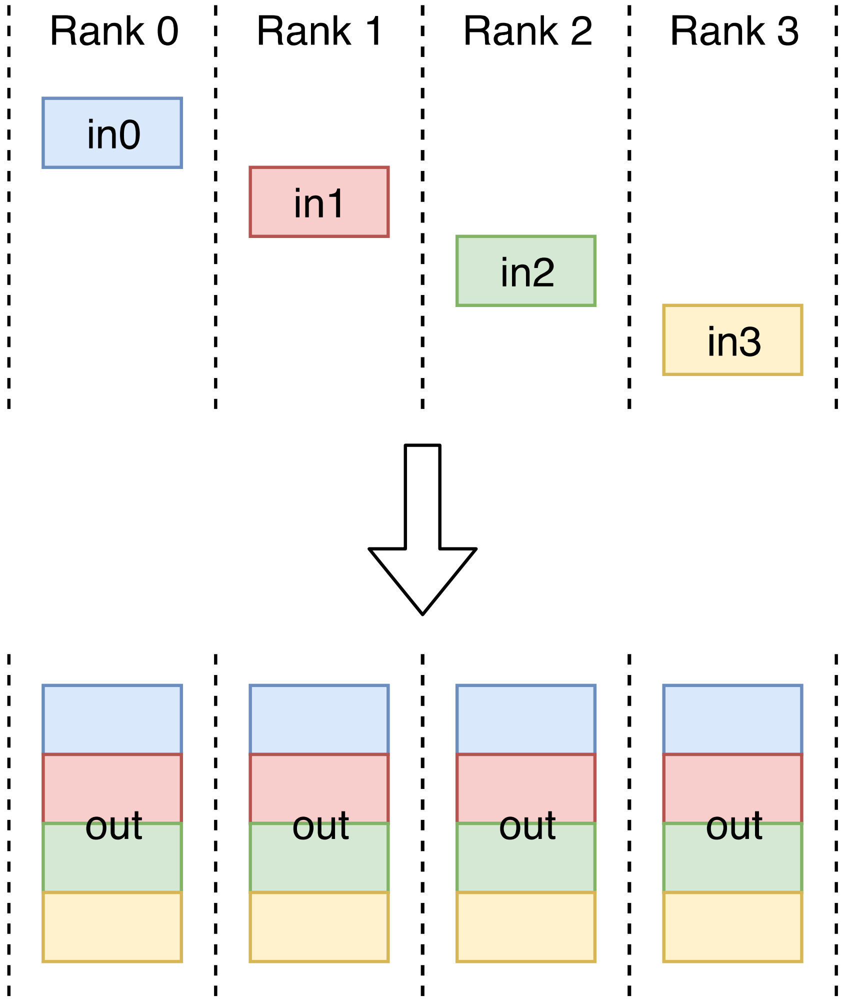
Ring



`out[Y*count+i] = inY[i]`

Allgather

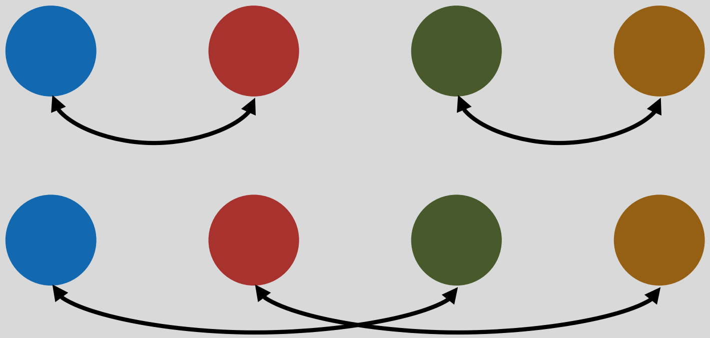




`out[Y*count+i] = inY[i]`

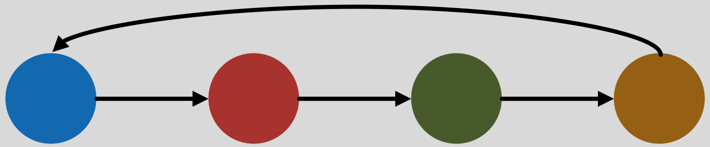
Allgather

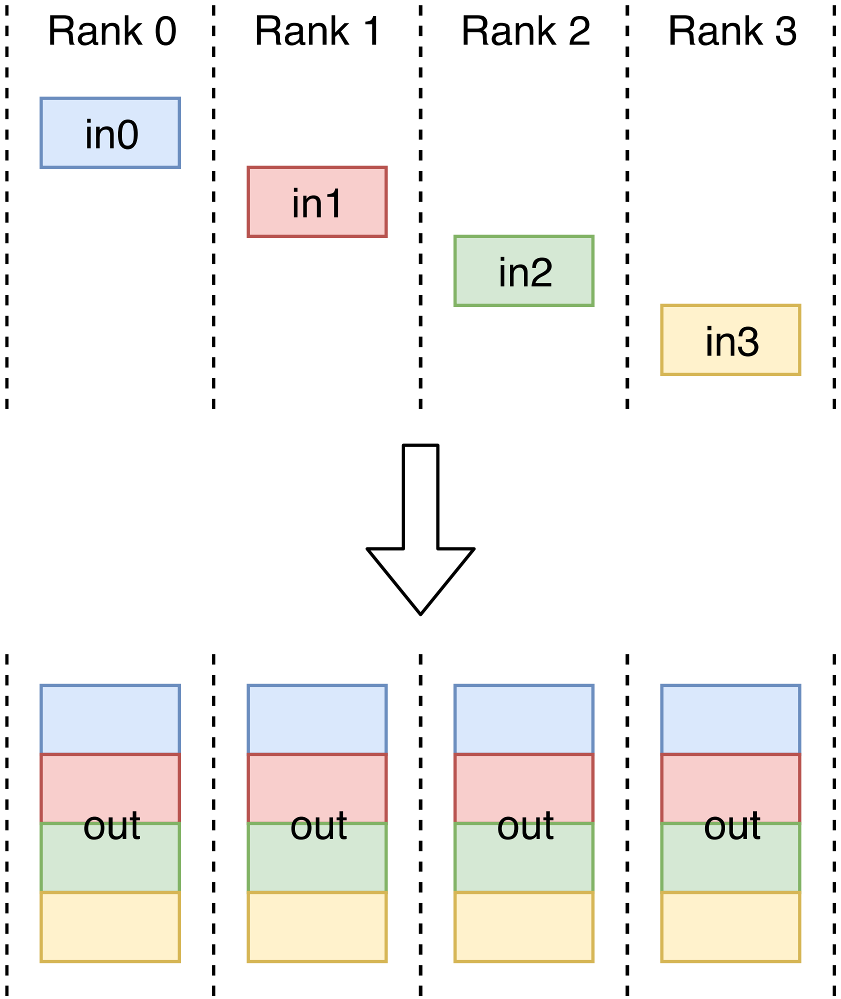
Recursive-doubling



$$\alpha \lg p + \frac{p-1}{p} n\beta$$

Ring

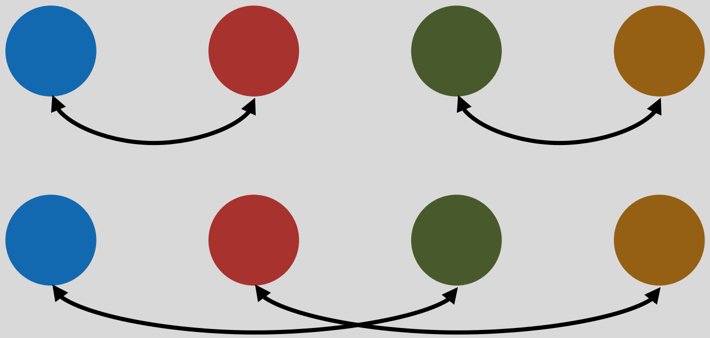




`out[Y*count+i] = inY[i]`

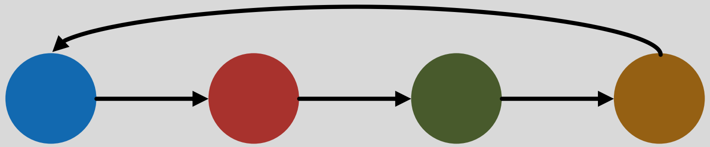
Allgather

Recursive-doubling

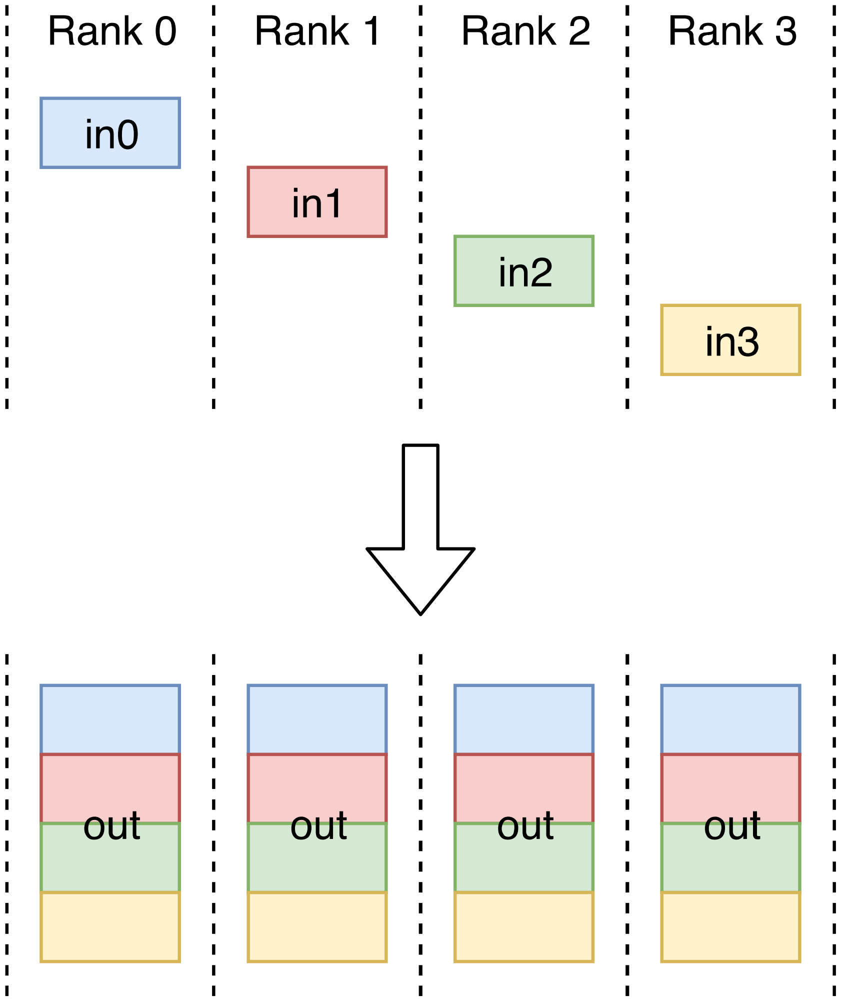


$$\alpha \lg p + \frac{p-1}{p} n\beta$$

Ring



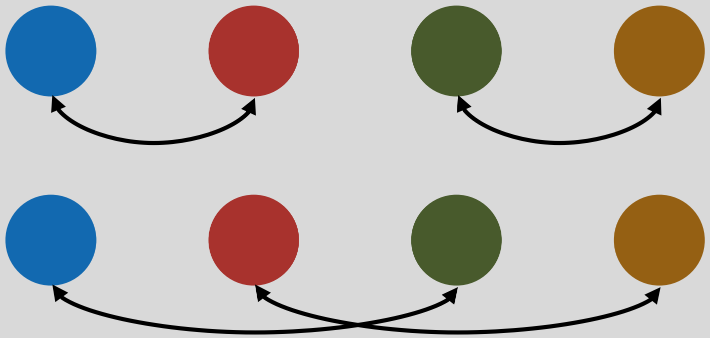
$$(p-1)\alpha + \frac{p-1}{p} n\beta$$



`out[Y*count+i] = inY[i]`

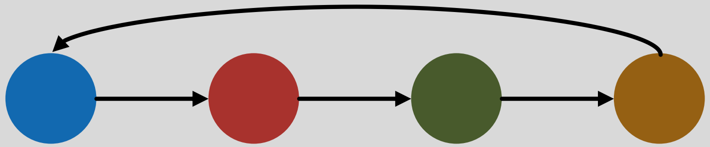
Allgather

Recursive-doubling

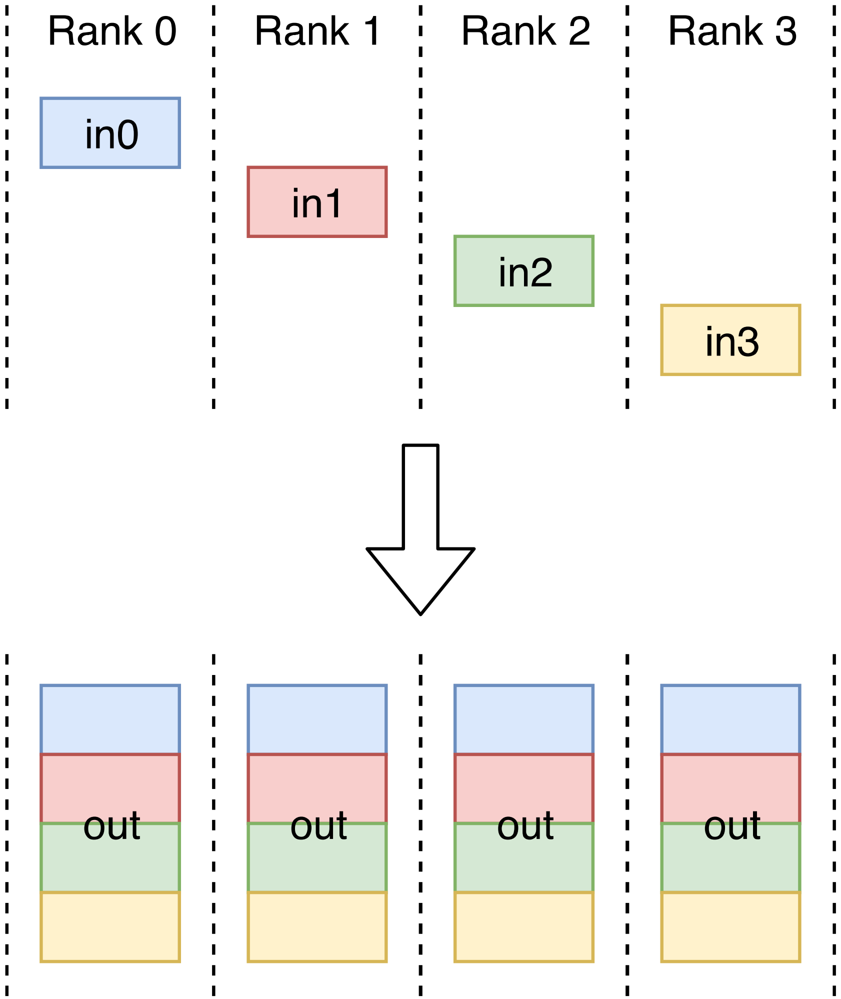


$$\alpha \lg p + \frac{p-1}{p} n\beta$$

Ring

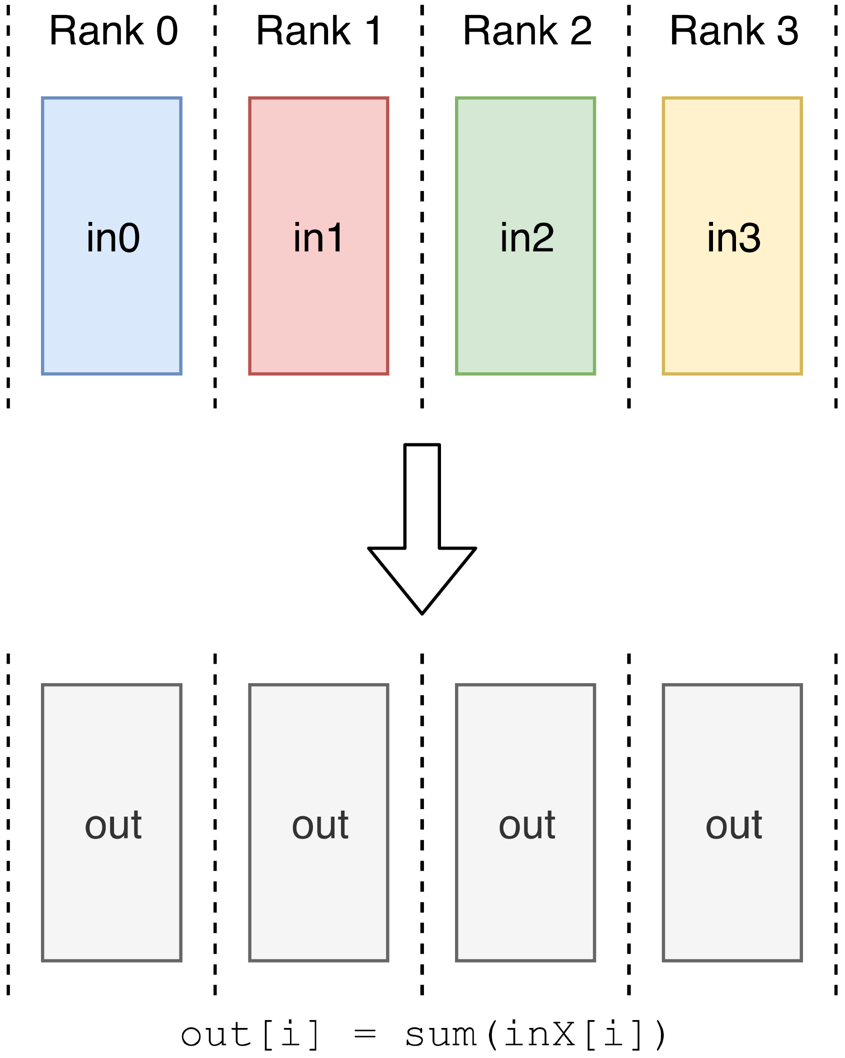


$$(p-1)\alpha + \frac{p-1}{p} n\beta$$



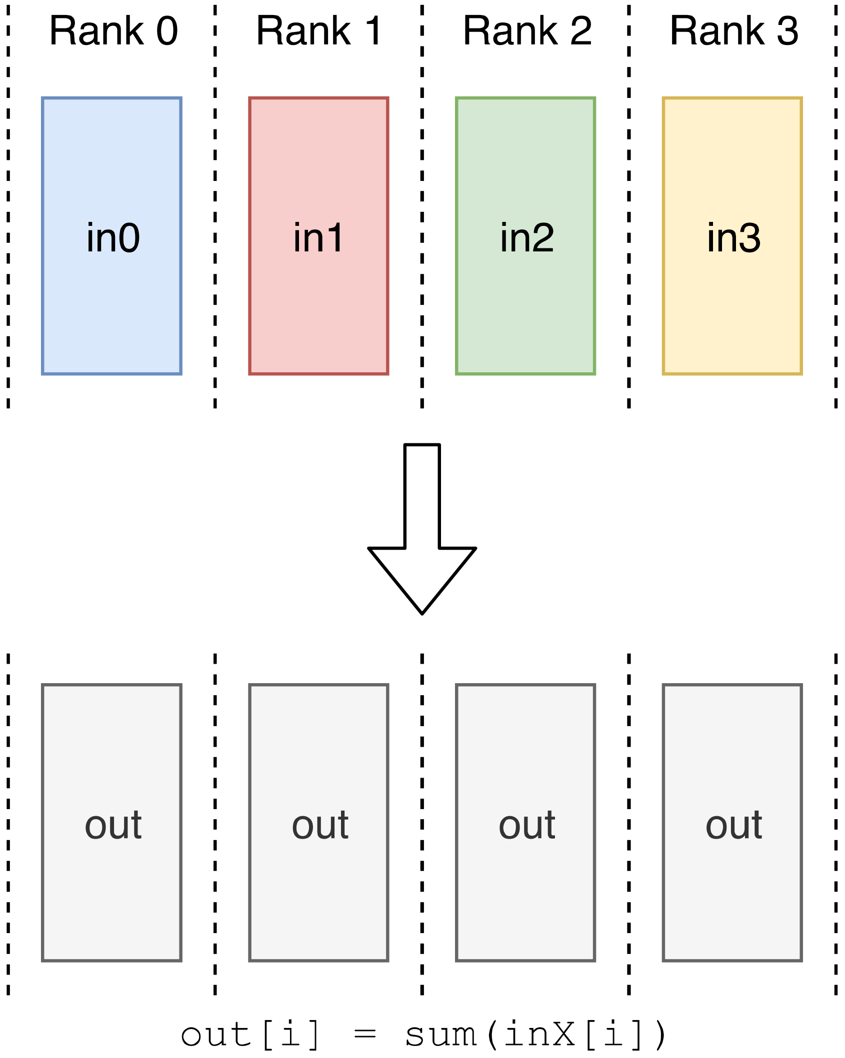
`out[Y*count+i] = inY[i]`

Allreduce


Rank 0
Rank 1
Rank 2
Rank 3
in0
in1
in2
in3
out
out
out
out
out[i] = sum(inX[i])

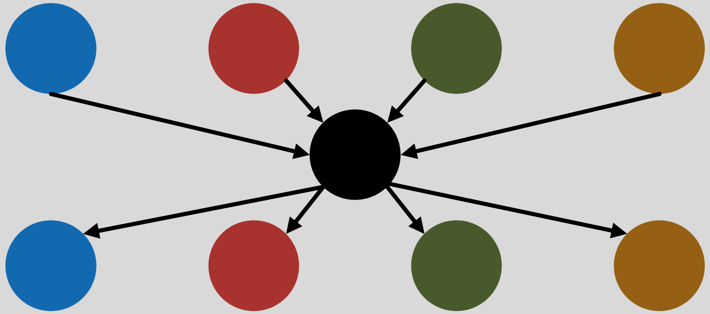
Allreduce

Parameter Server

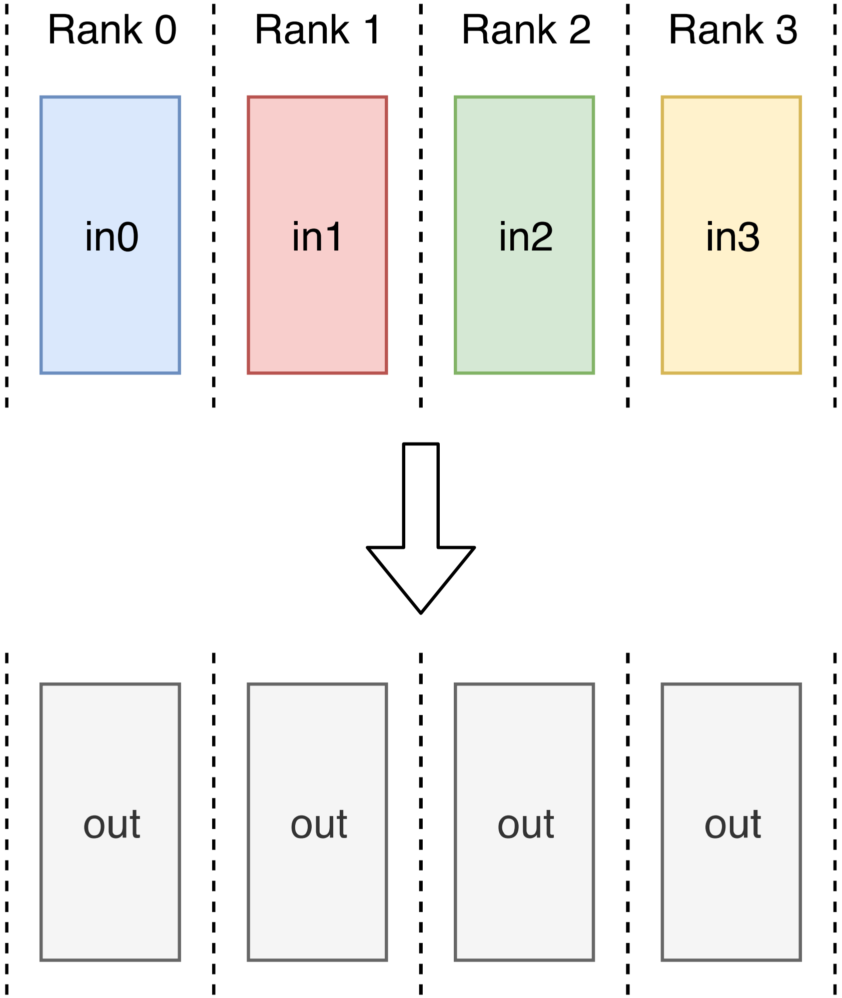


Allreduce

Parameter Server



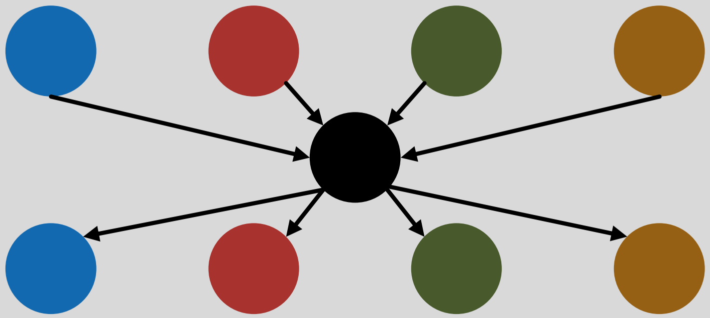
Tree (reduce/broadcast)



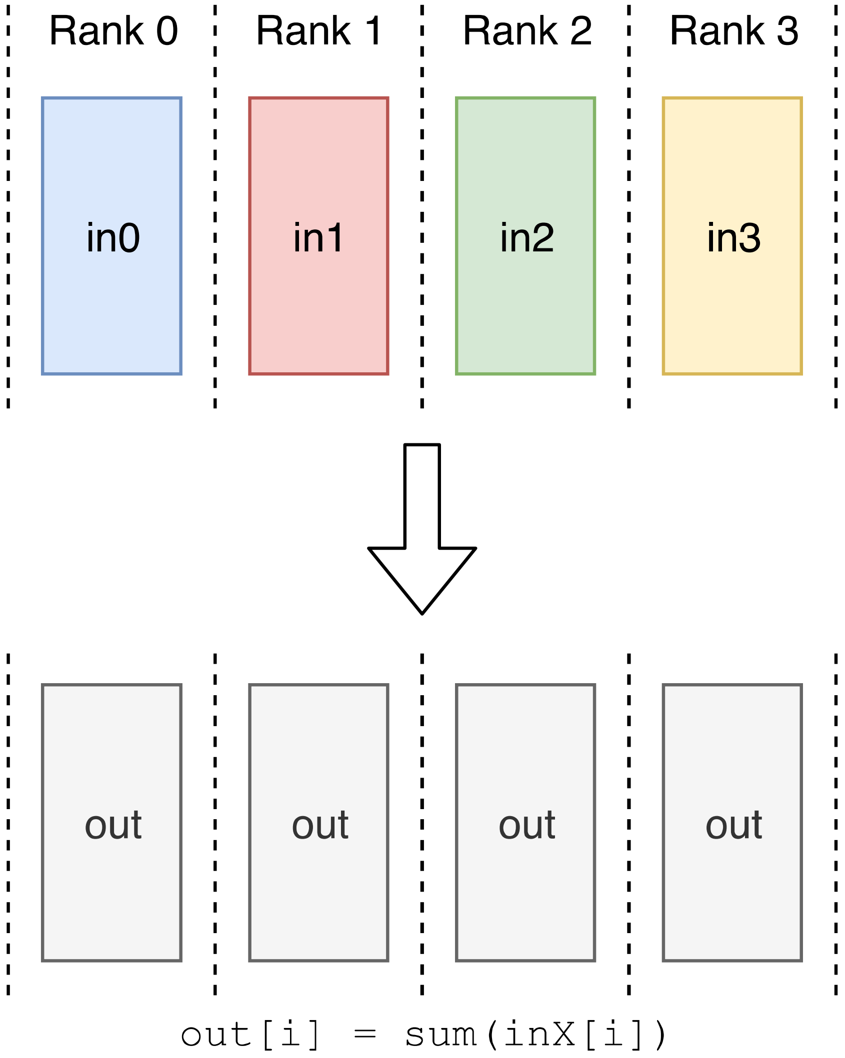

```
out[i] = sum(inX[i])
```

Allreduce

Parameter Server

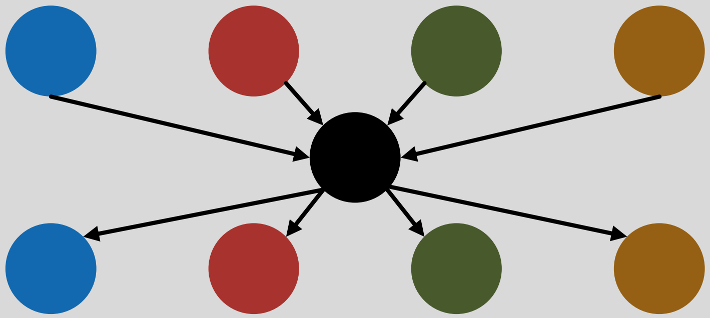


Tree (reduce/broadcast)



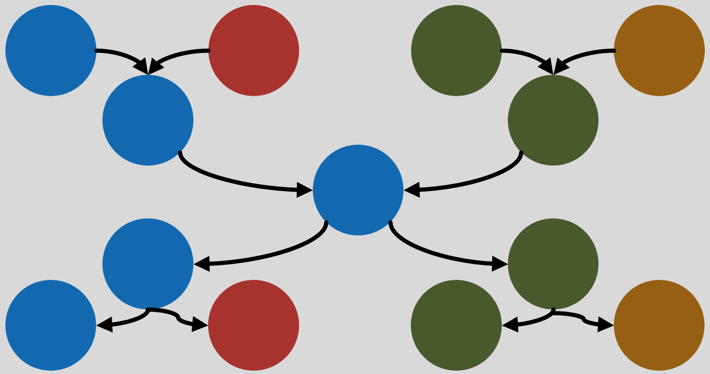
Allreduce

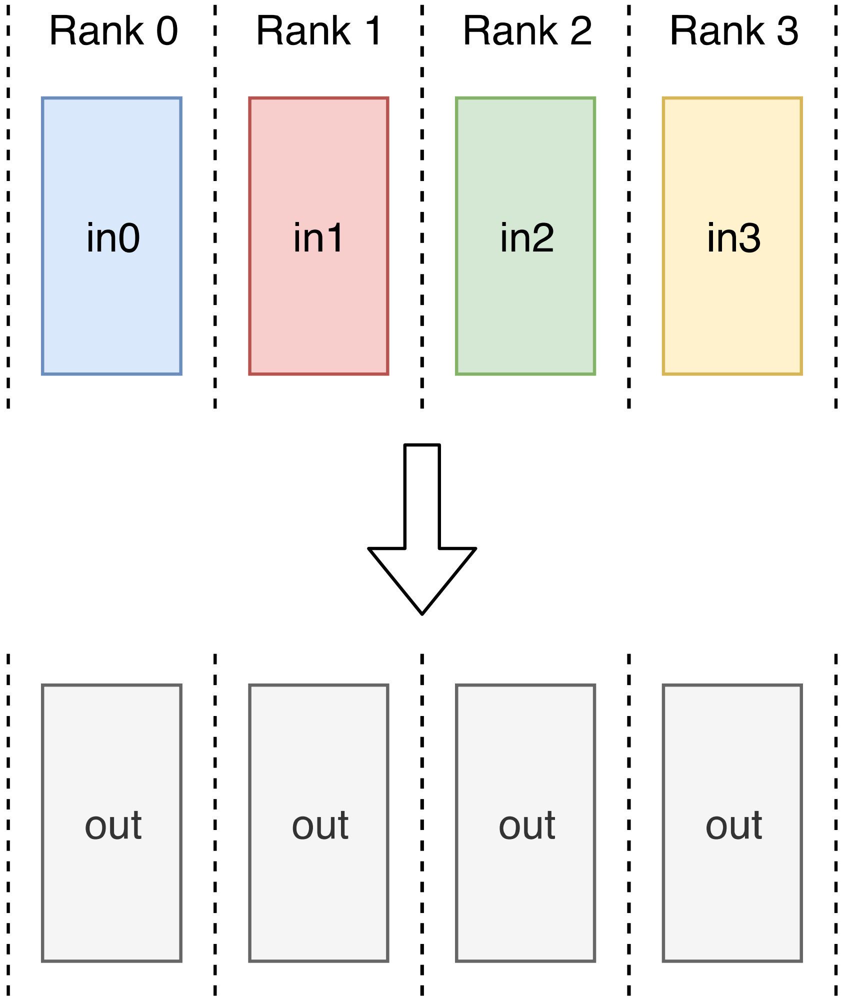
Parameter Server



$$2p\alpha + 2pn\beta + pn\gamma$$

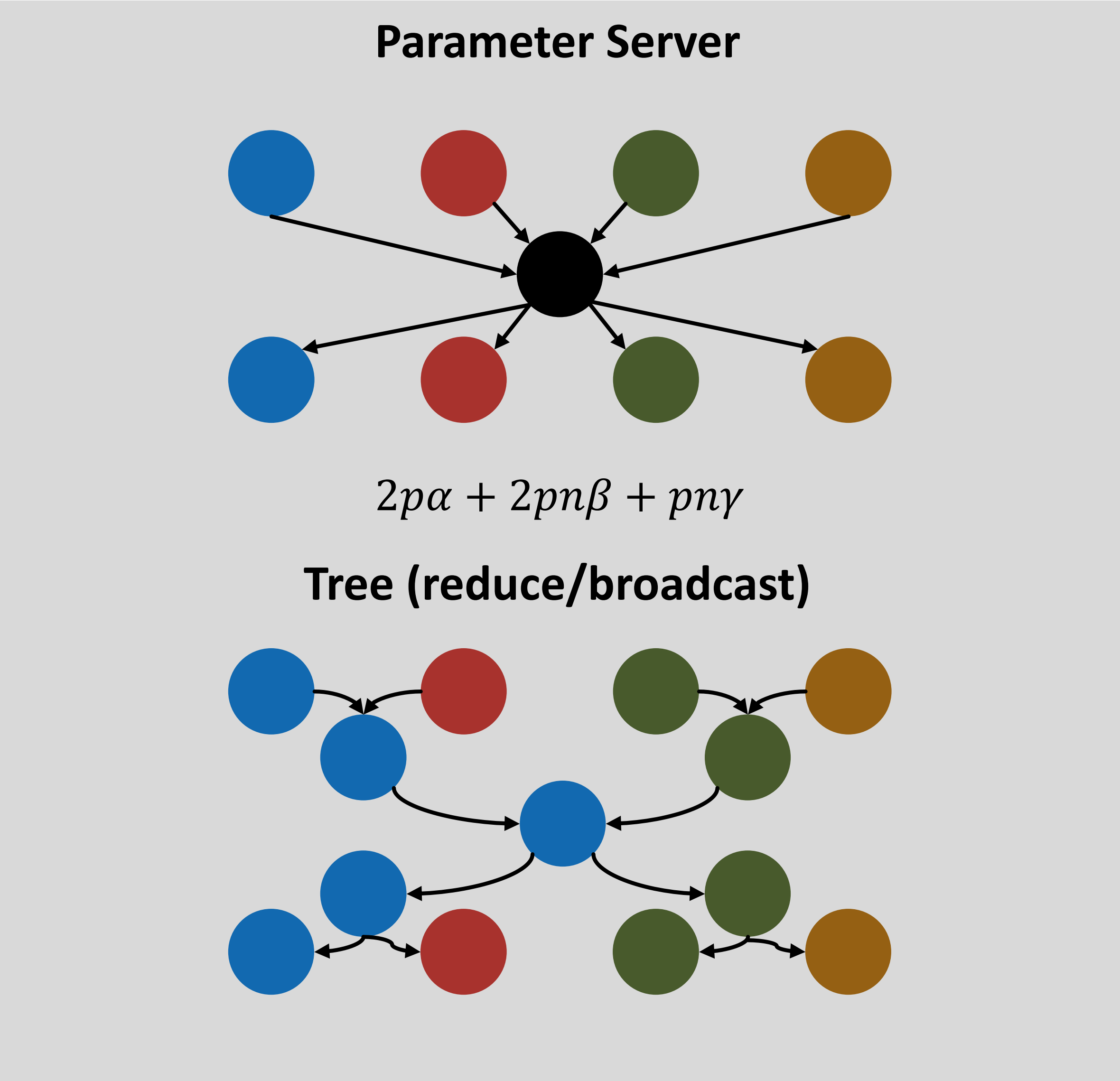
Tree (reduce/broadcast)



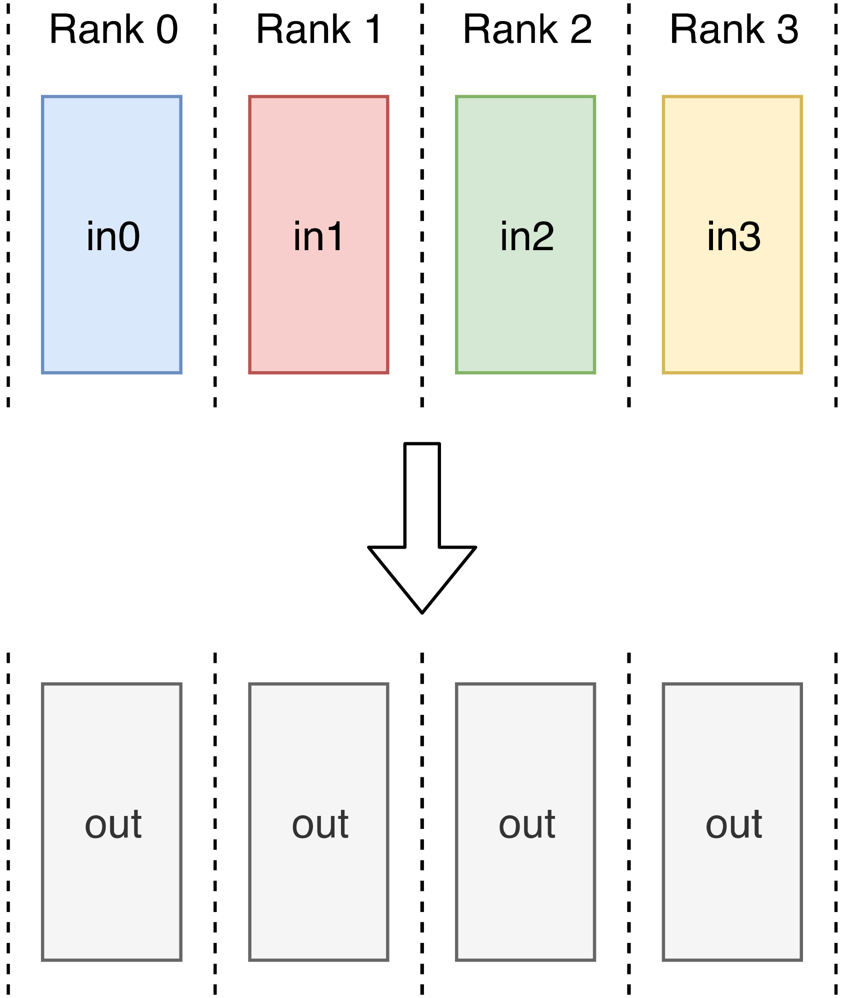


`out[i] = sum(inX[i])`

Allreduce



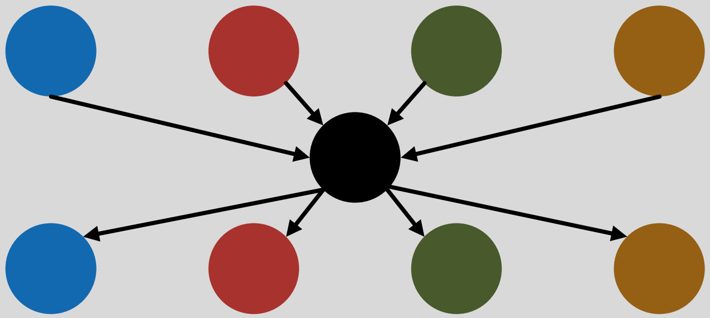
$$2p\alpha + 2pn\beta + pn\gamma$$



`out[i] = sum(inX[i])`

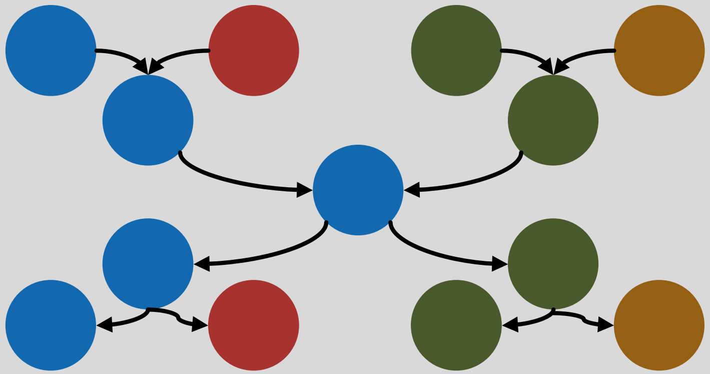
Allreduce

Parameter Server

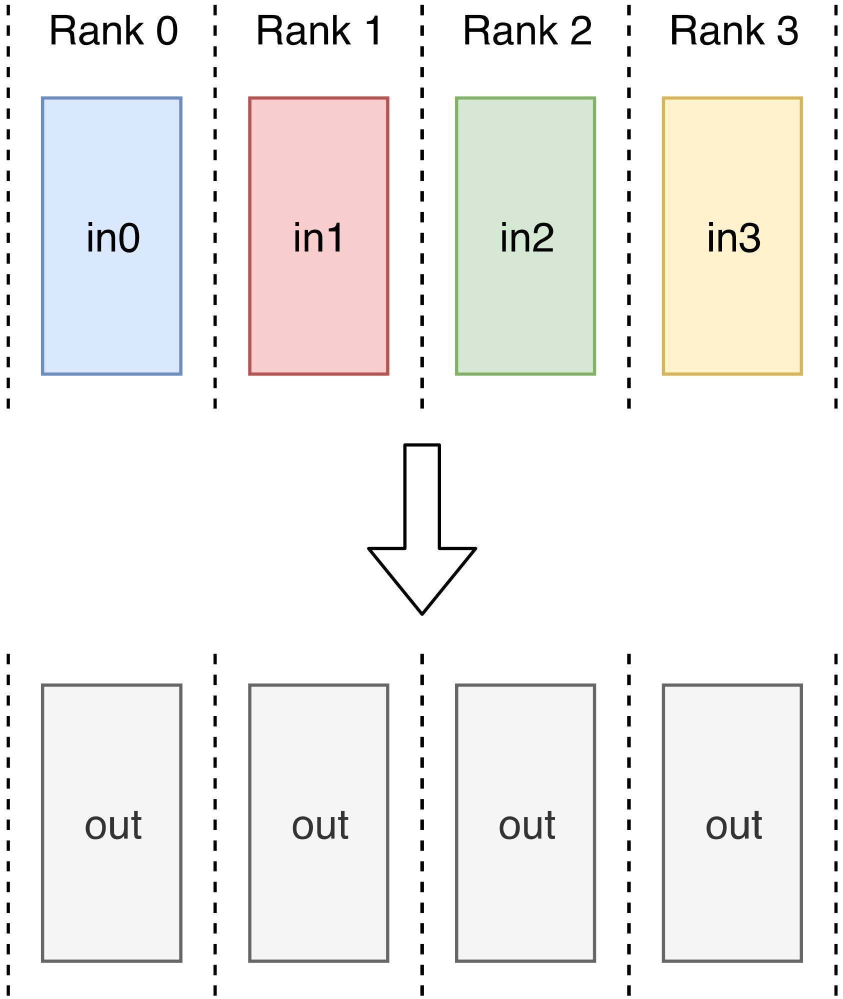


$2p\alpha + 2pn\beta + pn\gamma$

Tree (reduce/broadcast)



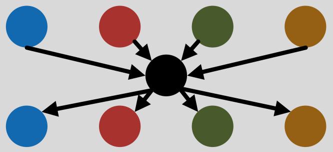
$2\alpha \lg p + 2\beta n \lg p + \gamma n \lg p$



`out[i] = sum(inX[i])`

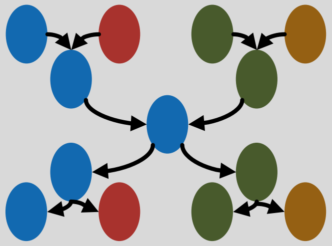
Allreduce

Parameter Server



$2p\alpha + 2pn\beta + pn\gamma$

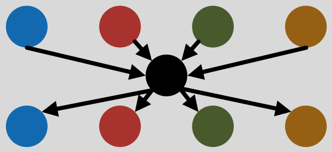
Tree



$2\alpha \lg p + 2\beta n \lg p \ + \gamma n \lg p$

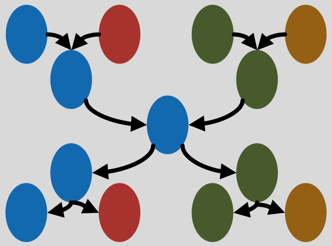
Allreduce

Parameter Server



$2p\alpha + 2pn\beta + pn\gamma$

Tree

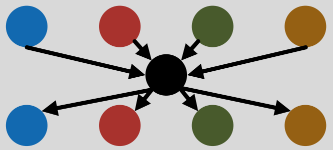


$2\alpha \lg p + 2\beta n \lg p \ + \gamma n \lg p$

Butterfly (doubling)

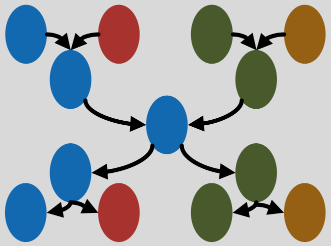
Allreduce

Parameter Server



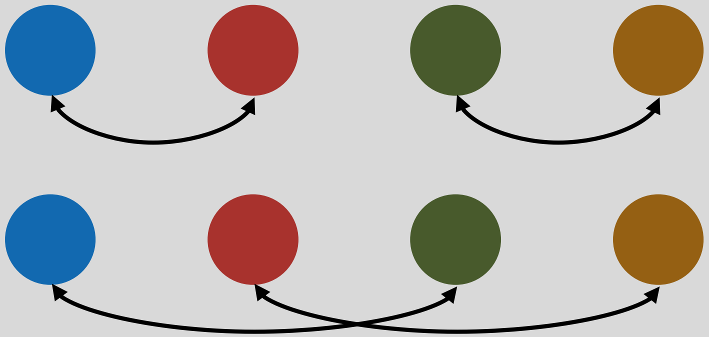
$2p\alpha + 2pn\beta + pn\gamma$

Tree



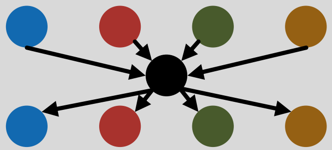
$2\alpha \lg p + 2\beta n \lg p \;\; + \gamma n \lg p$

Butterfly (doubling)



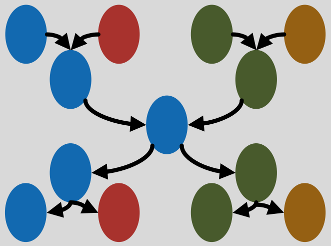
Allreduce

Parameter Server



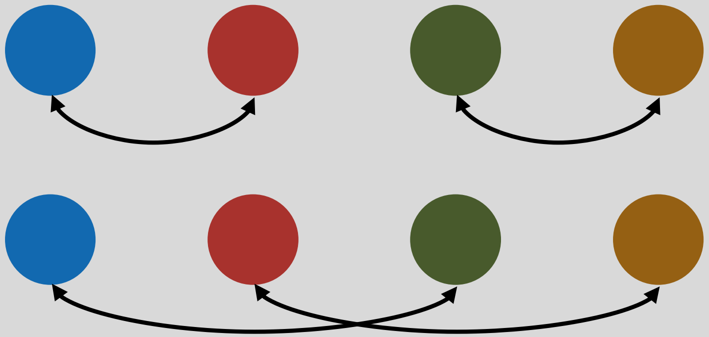
$2p\alpha + 2pn\beta + pn\gamma$

Tree



$2\alpha \lg p + 2\beta n \lg p + \gamma n \lg p$

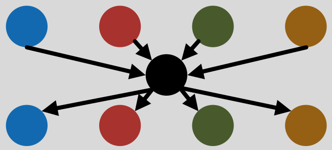
Butterfly (doubling)



Rabenseifner (half/double)

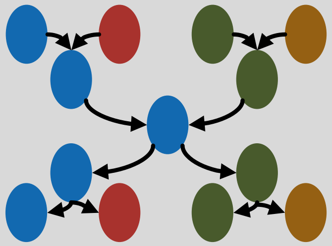
Allreduce

Parameter Server



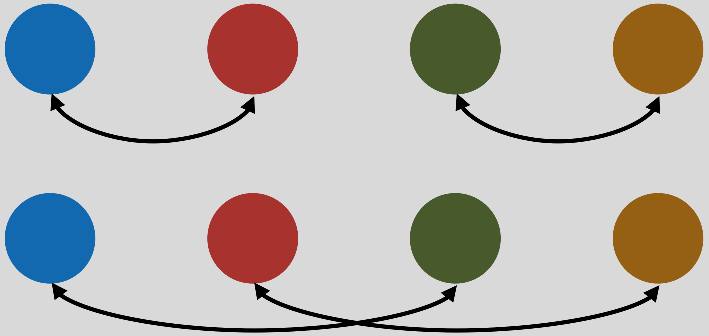
$2p\alpha + 2pn\beta + pn\gamma$

Tree

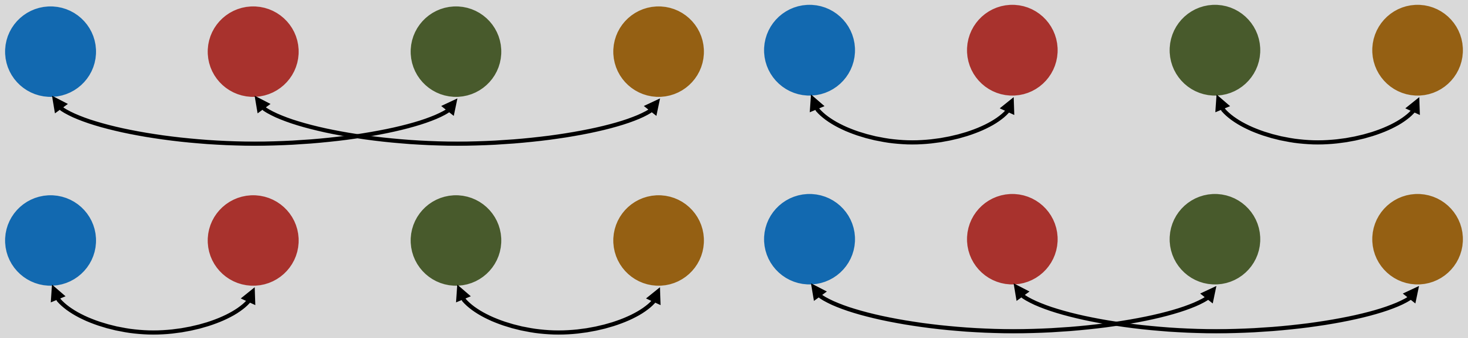


$2\alpha \lg p + 2\beta n \lg p + \gamma n \lg p$

Butterfly (doubling)



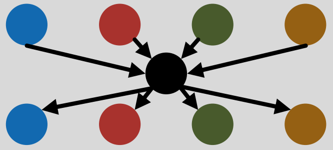
Rabenseifner (half/double)



Ring

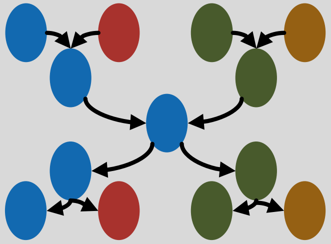
Allreduce

Parameter Server



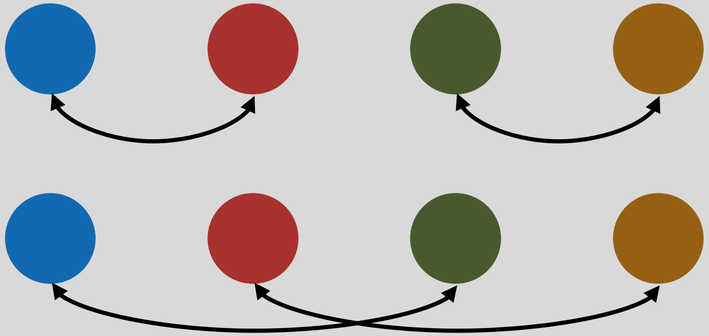
$2p\alpha + 2pn\beta + pn\gamma$

Tree

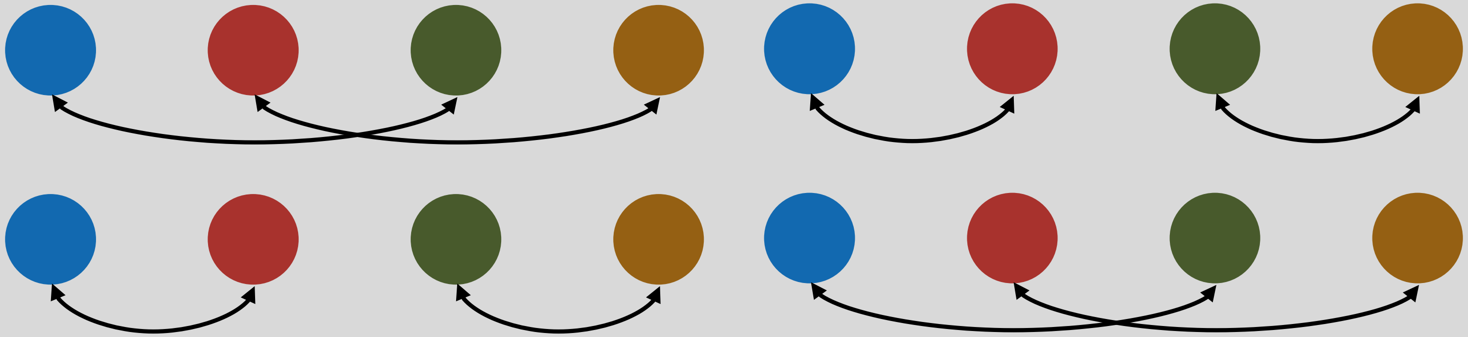


$2\alpha \lg p + 2\beta n \lg p \ + \gamma n \lg p$

Butterfly (doubling)



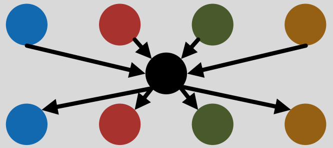
Rabenseifner (half/double)



Ring

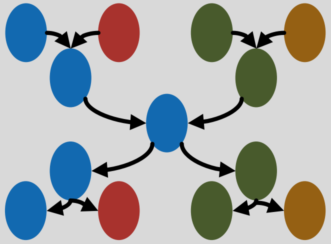
Allreduce

Parameter Server



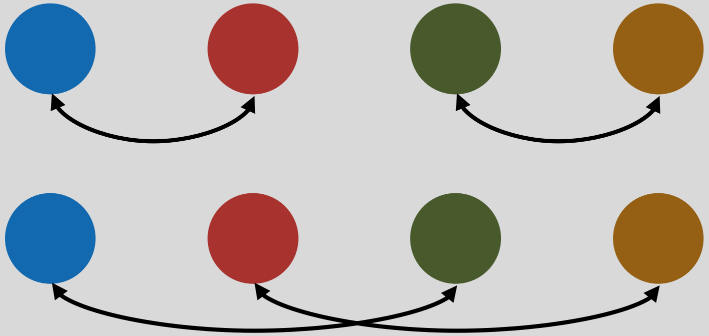
$2p\alpha + 2pn\beta + pn\gamma$

Tree

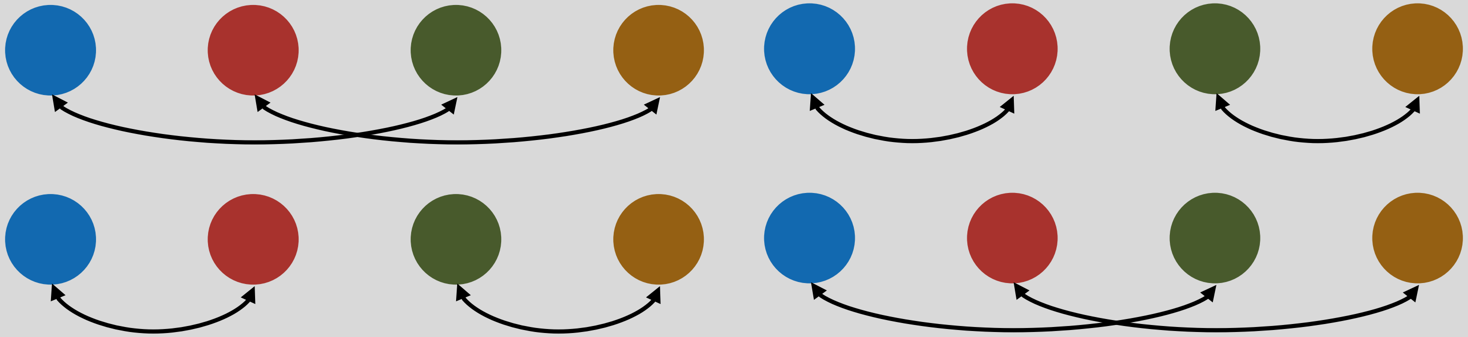


$2\alpha \lg p + 2\beta n \lg p \ + \gamma n \lg p$

Butterfly (doubling)



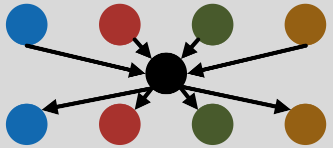
Rabenseifner (half/double)



Ring

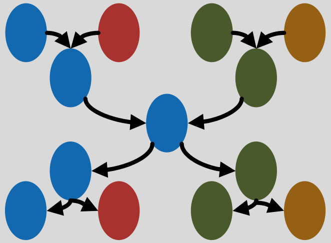
Allreduce

Parameter Server



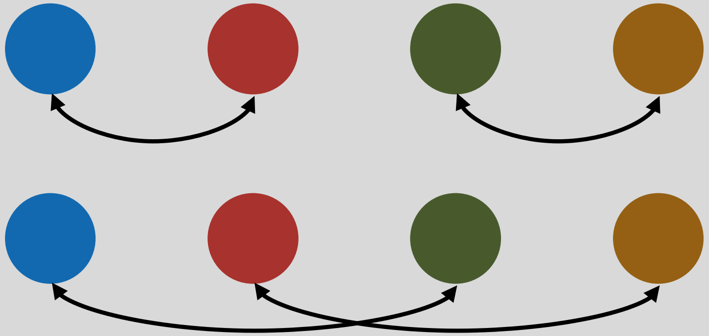
$2p\alpha + 2pn\beta + pn\gamma$

Tree

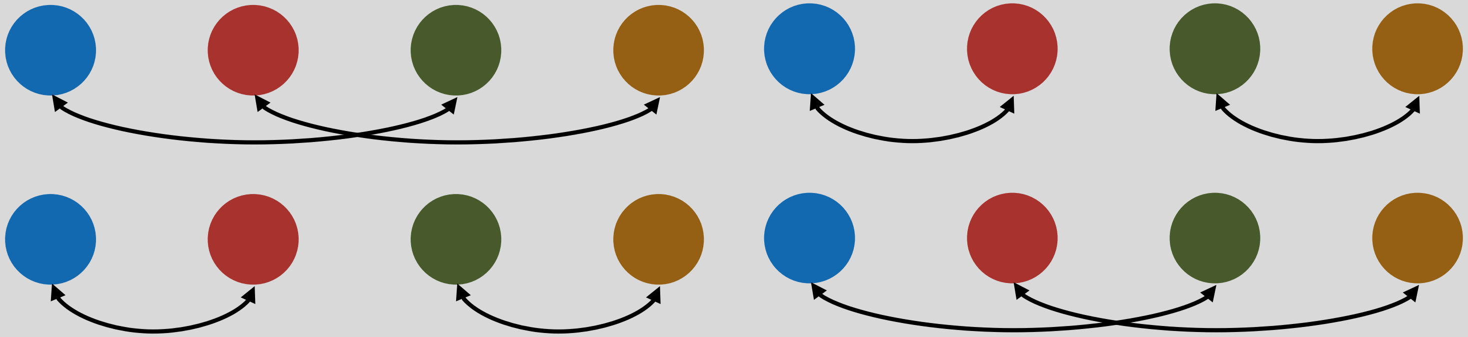


$2\alpha \lg p + 2\beta n \lg p \ + \gamma n \lg p$

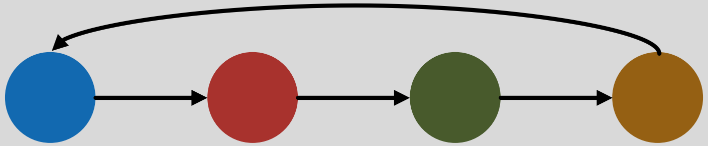
Butterfly (doubling)



Rabenseifner (half/double)

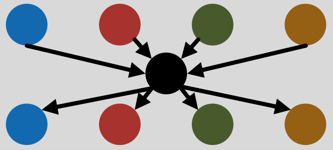


Ring



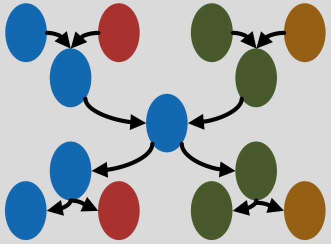
Allreduce

Parameter Server



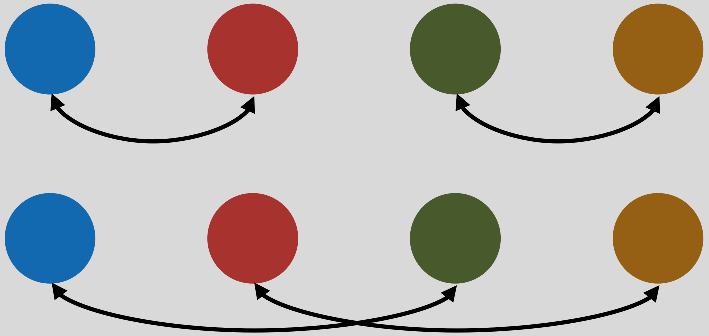
$2p\alpha + 2pn\beta + pn\gamma$

Tree

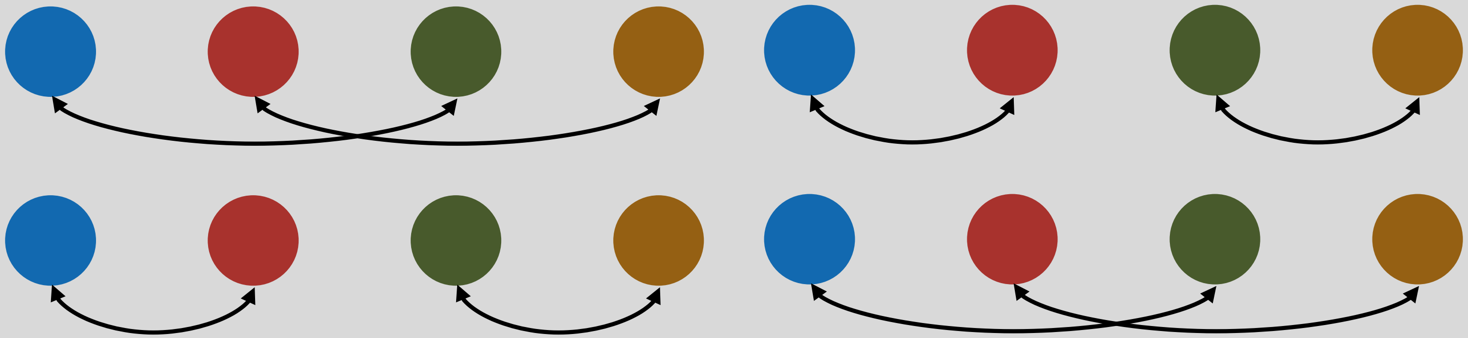


$2\alpha \lg p + 2\beta n \lg p \ + \gamma n \lg p$

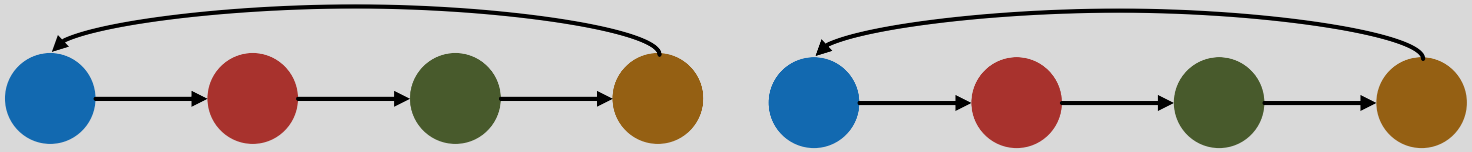
Butterfly (doubling)



Rabenseifner (half/double)

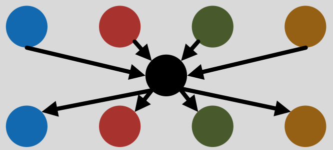


Ring



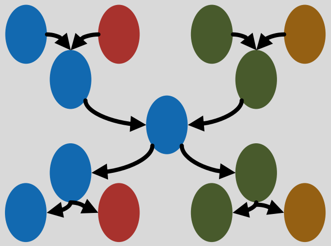
Allreduce

Parameter Server



$2p\alpha + 2pn\beta + pn\gamma$

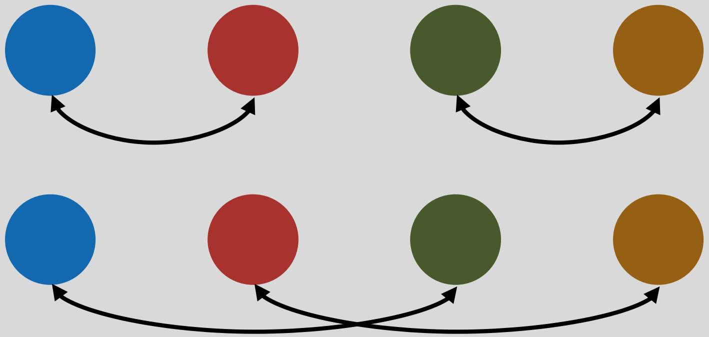
Tree



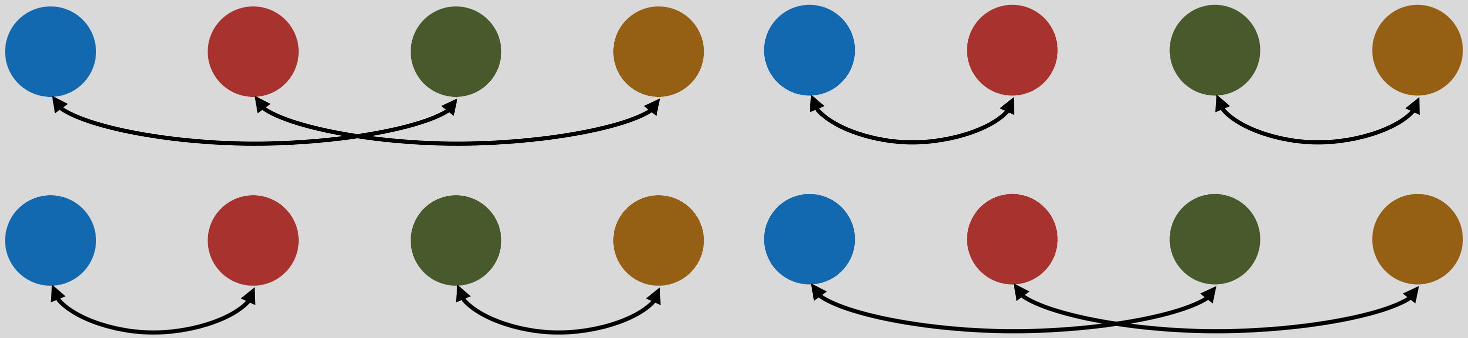
$2\alpha \lg p + 2\beta n \lg p + \gamma n \lg p$

Butterfly (doubling)

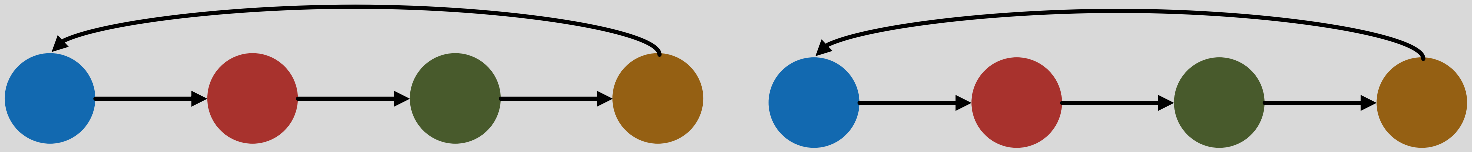
$\alpha \lg p + \beta n \lg p + \gamma n \lg p$



Rabenseifner (half/double)

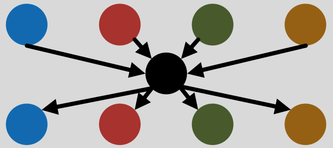


Ring



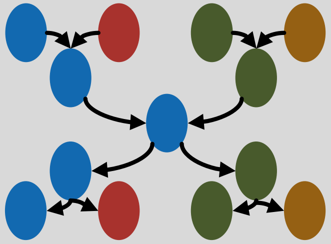
Allreduce

Parameter Server



$2p\alpha + 2pn\beta + pn\gamma$

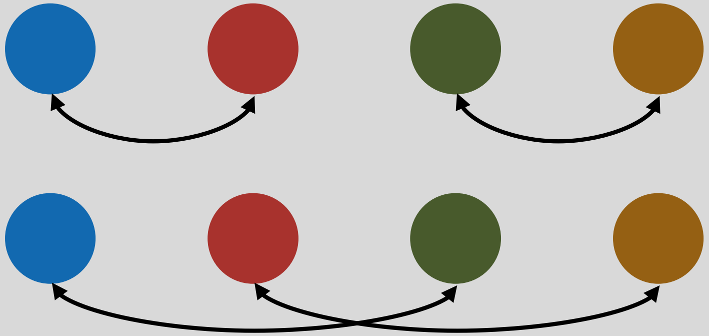
Tree



$2\alpha \lg p + 2\beta n \lg p + \gamma n \lg p$

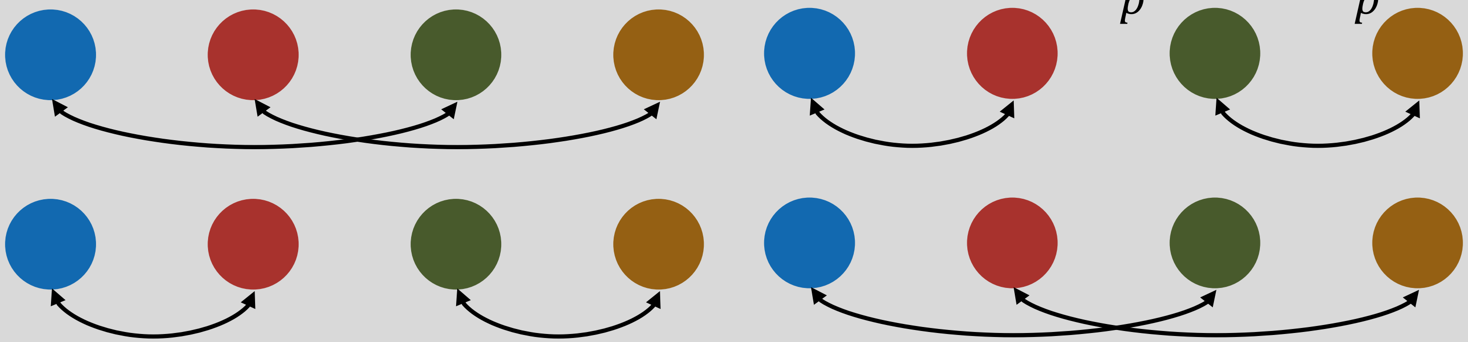
Butterfly (doubling)

$\alpha \lg p + \beta n \lg p + \gamma n \lg p$

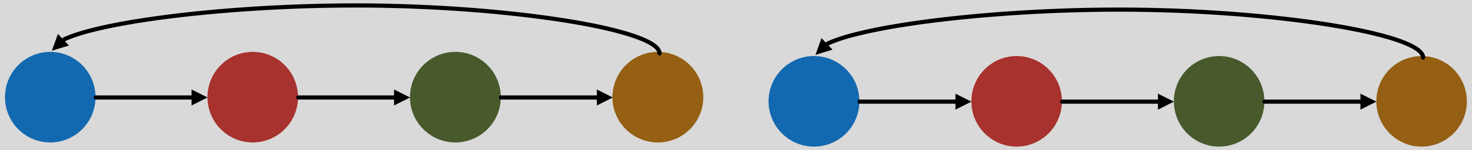


Rabenseifner (half/double)

$2\alpha \lg p + 2\frac{p-1}{p} n\beta + \frac{p-1}{p} n\gamma$

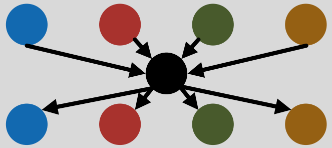


Ring



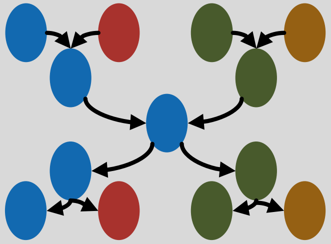
Allreduce

Parameter Server



$$2p\alpha + 2pn\beta + pn\gamma$$

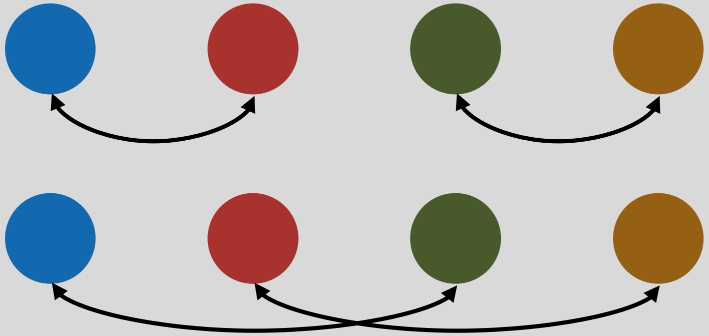
Tree



$$2\alpha \lg p + 2\beta n \lg p + \gamma n \lg p$$

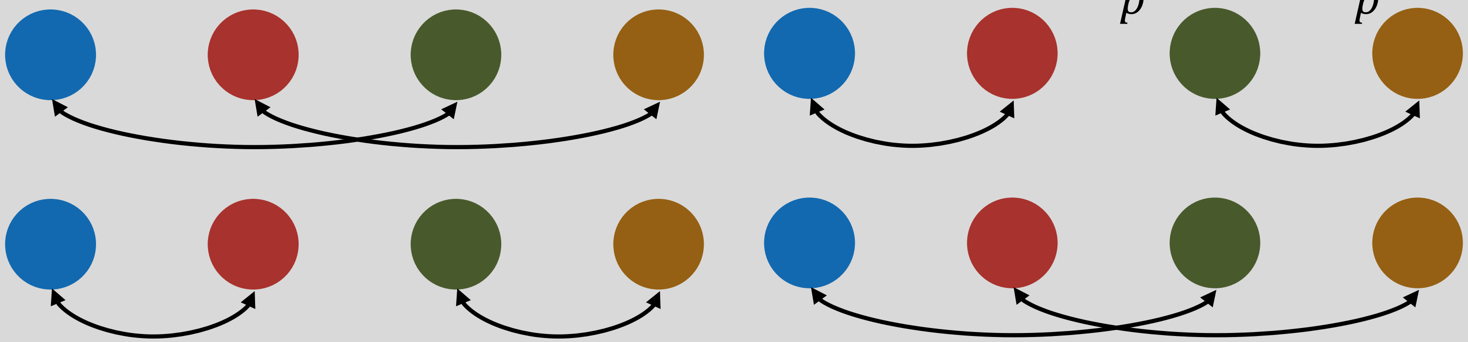
Butterfly (doubling)

$$\alpha \lg p + \beta n \lg p + \gamma n \lg p$$



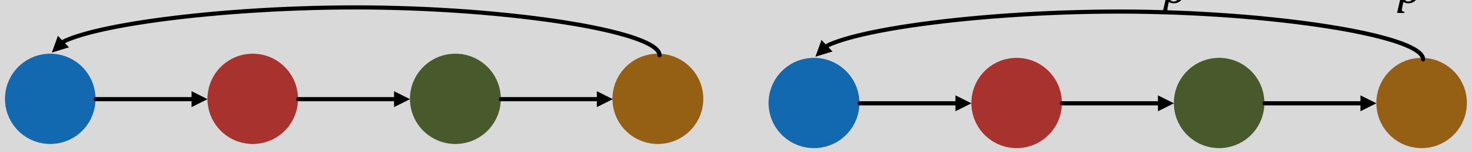
Rabenseifner (half/double)

$$2\alpha \lg p + 2\frac{p-1}{p} n\beta + \frac{p-1}{p} n\gamma$$



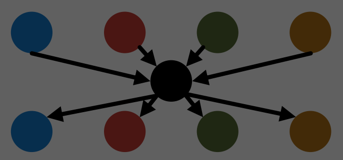
Ring

$$2(p-1)\alpha + 2\frac{p-1}{p} n\beta + \frac{p-1}{p} n\gamma$$



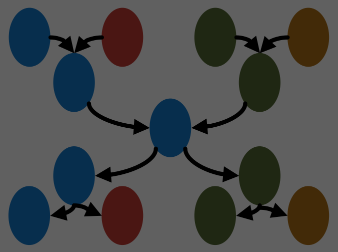
Allreduce

Parameter Server



$2p\alpha + 2pn\beta + pn\gamma$

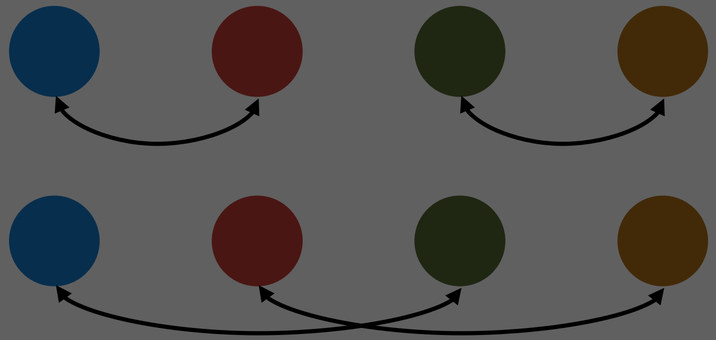
Tree



$2\alpha \lg p + 2\beta n \lg p \; + \gamma n \lg p$

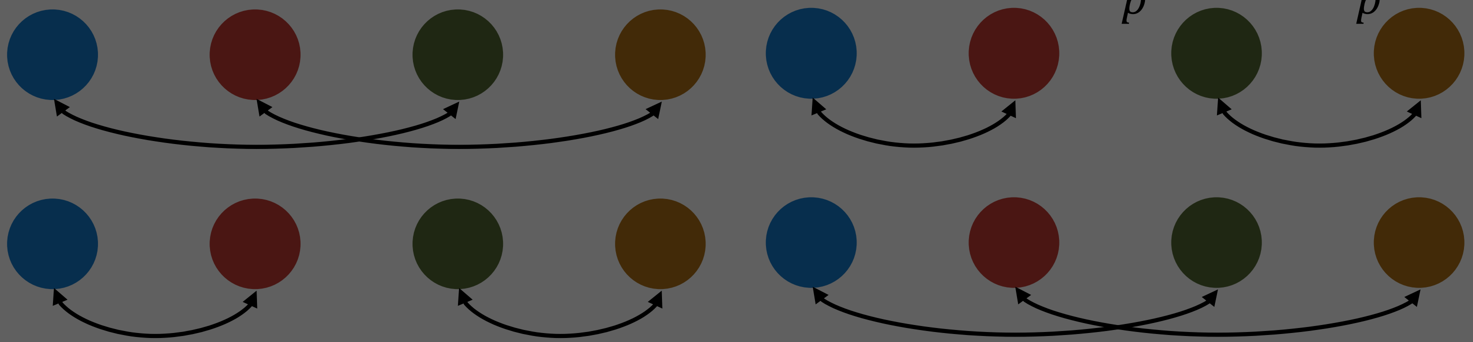
Butterfly (doubling)

$\alpha \lg p + \beta n \lg p + \gamma n \lg p$



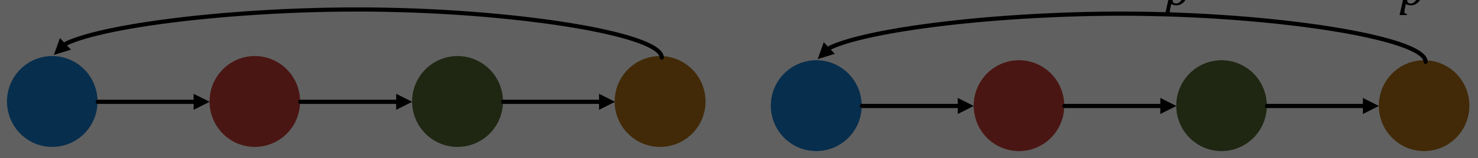
Rabenseifner (half/double)

$2\alpha \lg p + 2\frac{p-1}{p} n\beta + \frac{p-1}{p} n\gamma$



Ring

$2(p-1)\alpha + 2\frac{p-1}{p} n\beta + \frac{p-1}{p} n\gamma$



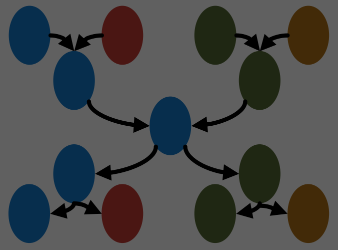
Allreduce

Implementation matters!

Parameter Server

$2p\alpha + 2pn\beta + pn\gamma$

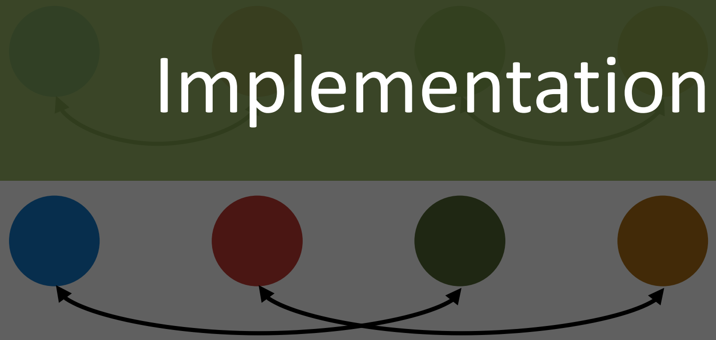
Tree



$2\alpha \lg p + 2\beta n \lg p + \gamma n \lg p$

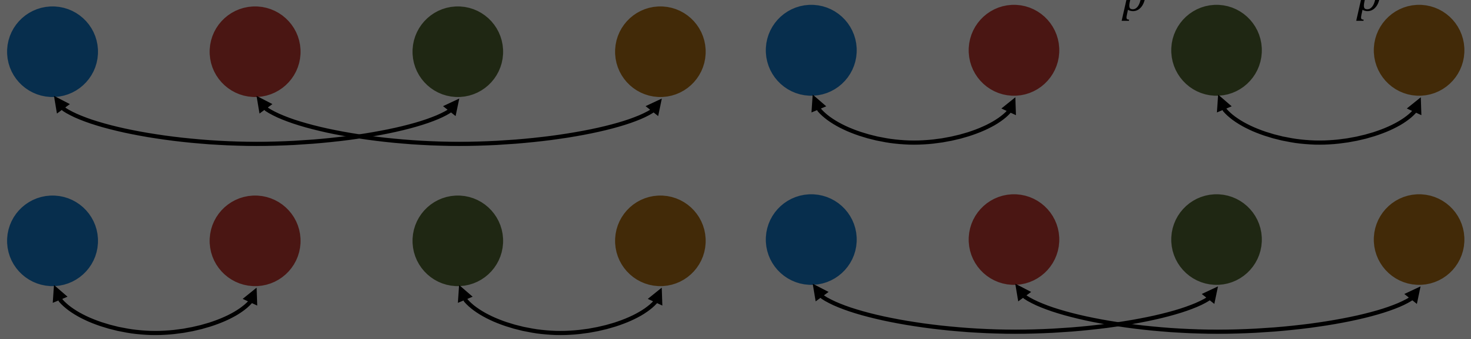
Butterfly (doubling)

$\alpha \lg p + \beta n \lg p + \gamma n \lg p$



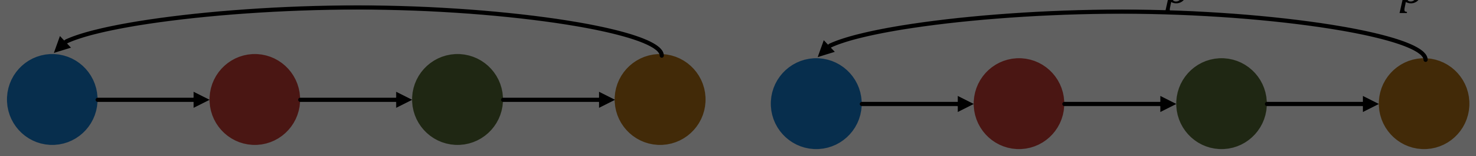
Rabenseifner (half/double)

$2\alpha \lg p + 2\frac{p-1}{p} n\beta + \frac{p-1}{p} n\gamma$



Ring

$2(p-1)\alpha + 2\frac{p-1}{p} n\beta + \frac{p-1}{p} n\gamma$



Allreduce

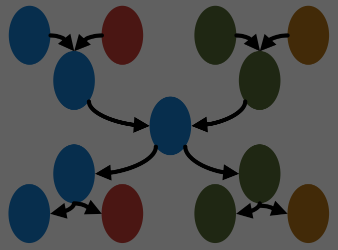
Parameter Server

Butterfly (doubling) $\alpha \lg p + \beta n \lg p + \gamma n \lg p$

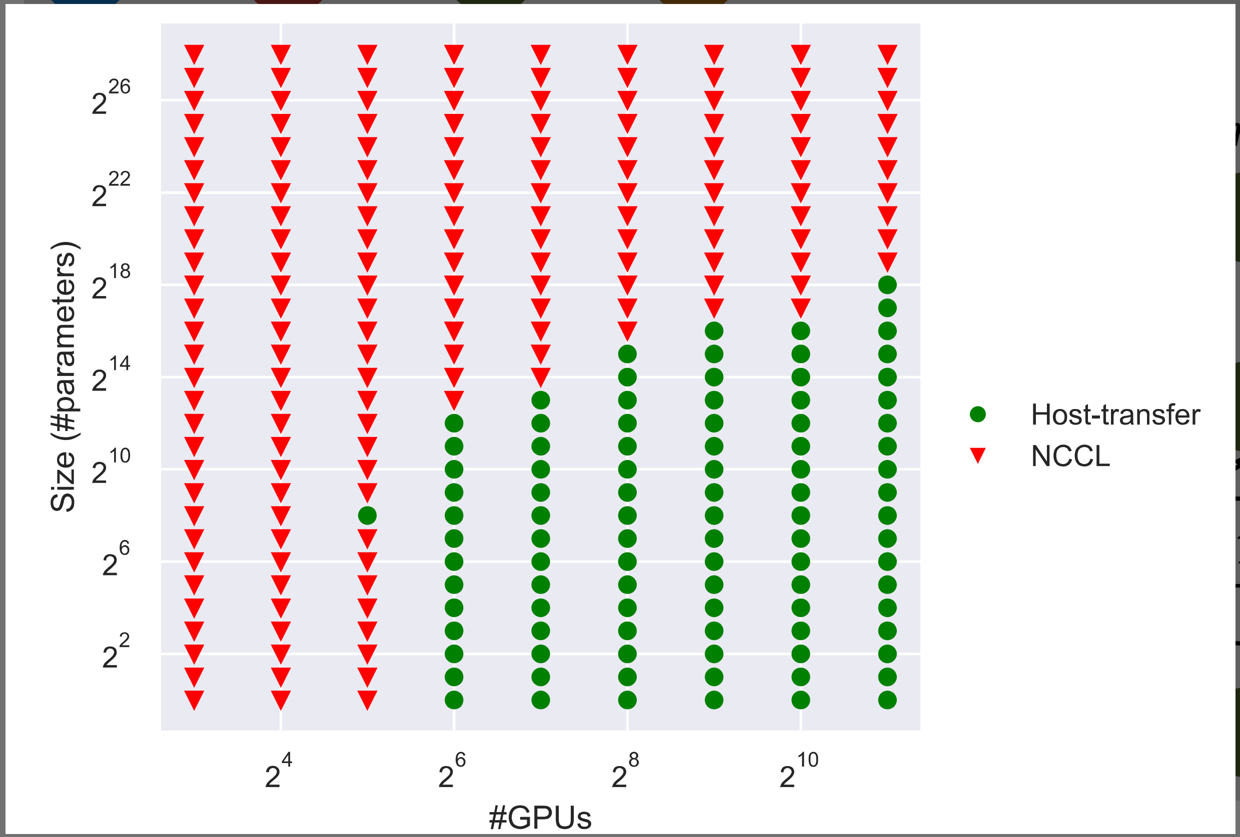
Implementation matters!

$2p\alpha + 2pn\beta + pn\gamma$

Tree



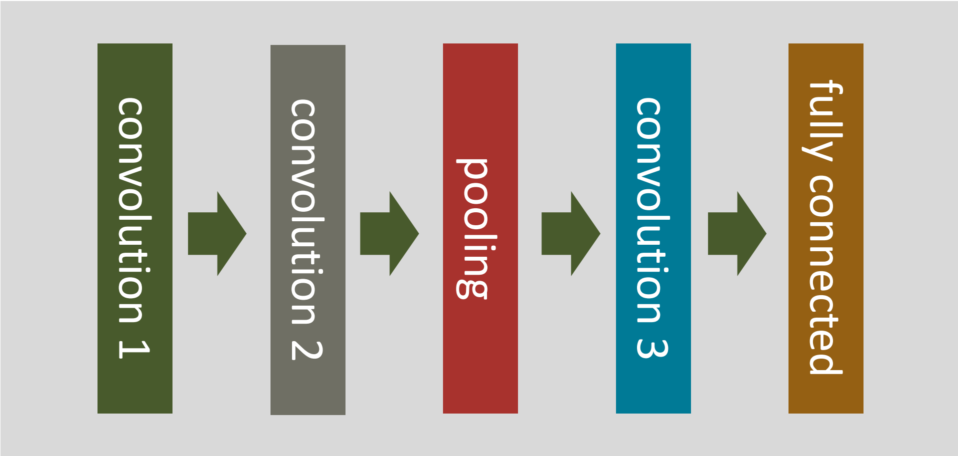
$2\alpha \lg p + 2\beta n \lg p + \gamma n \lg p$

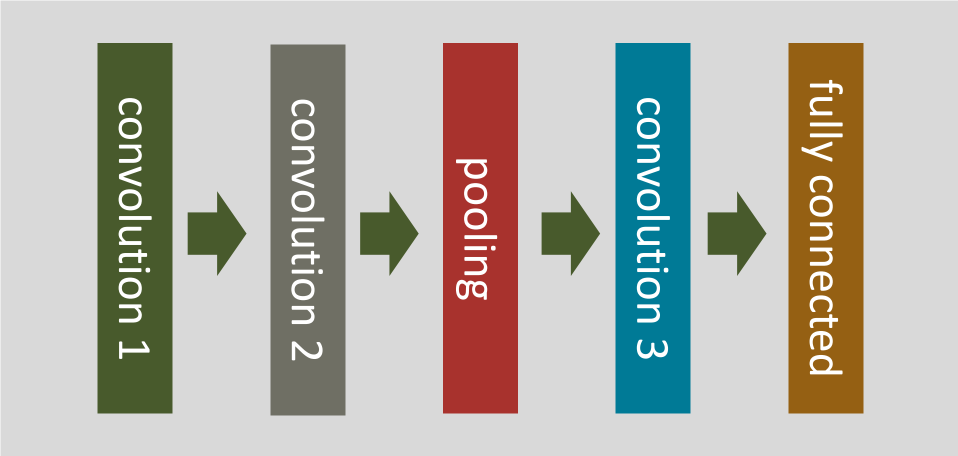


$n\beta + \frac{p-1}{p} n\gamma$

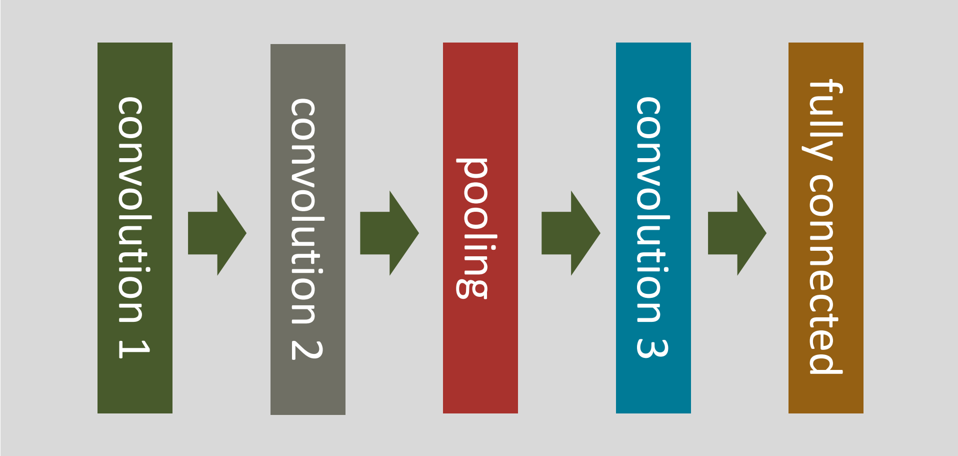
$n\beta + \frac{p-1}{p} n\gamma$

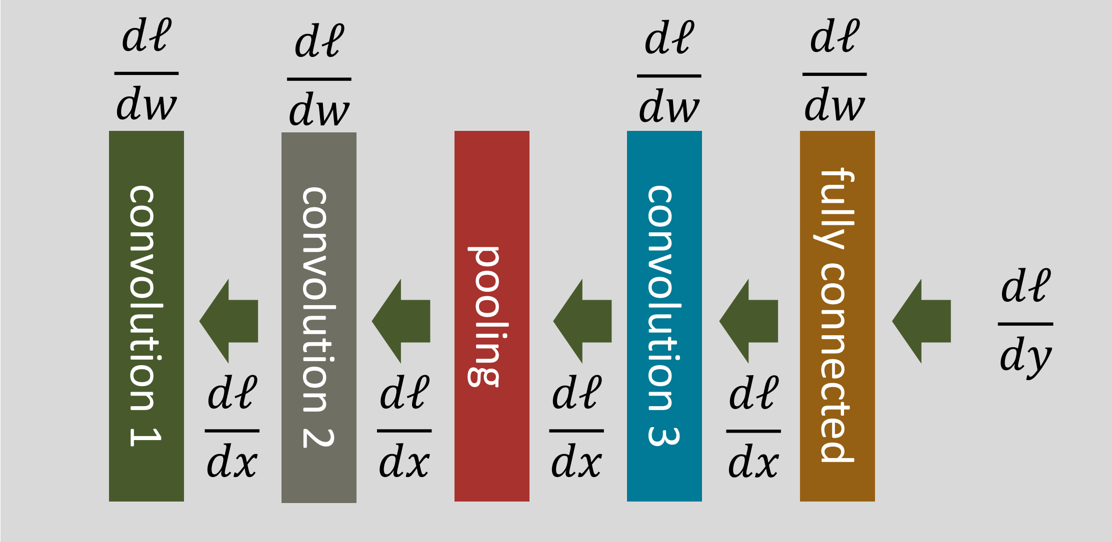
Distributed data-parallelism

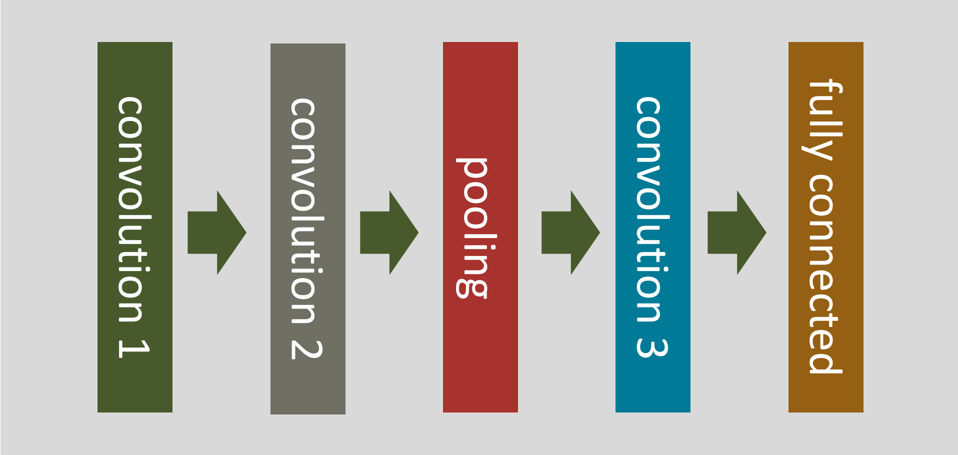


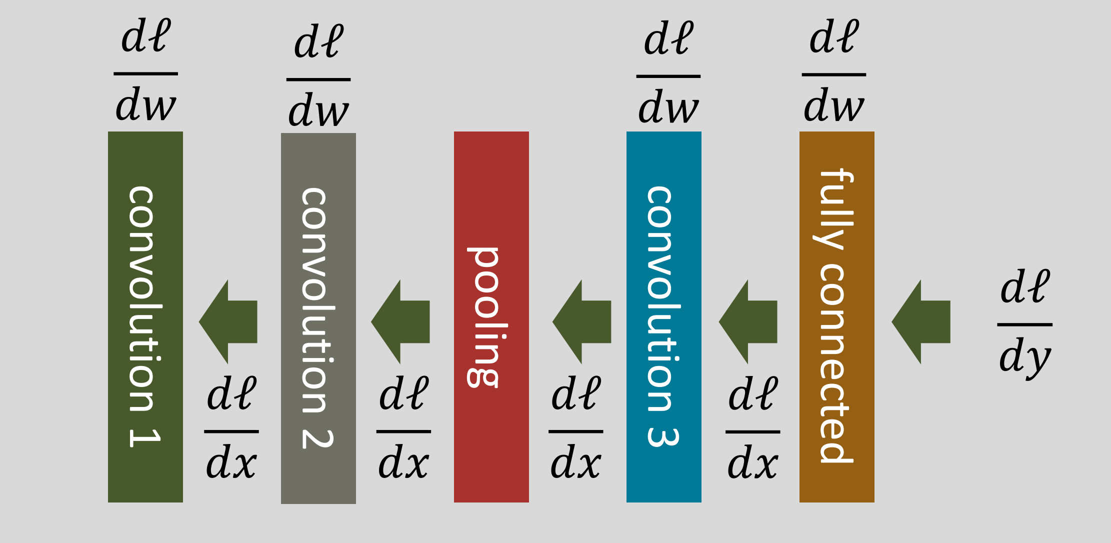


Distributed data-parallelism

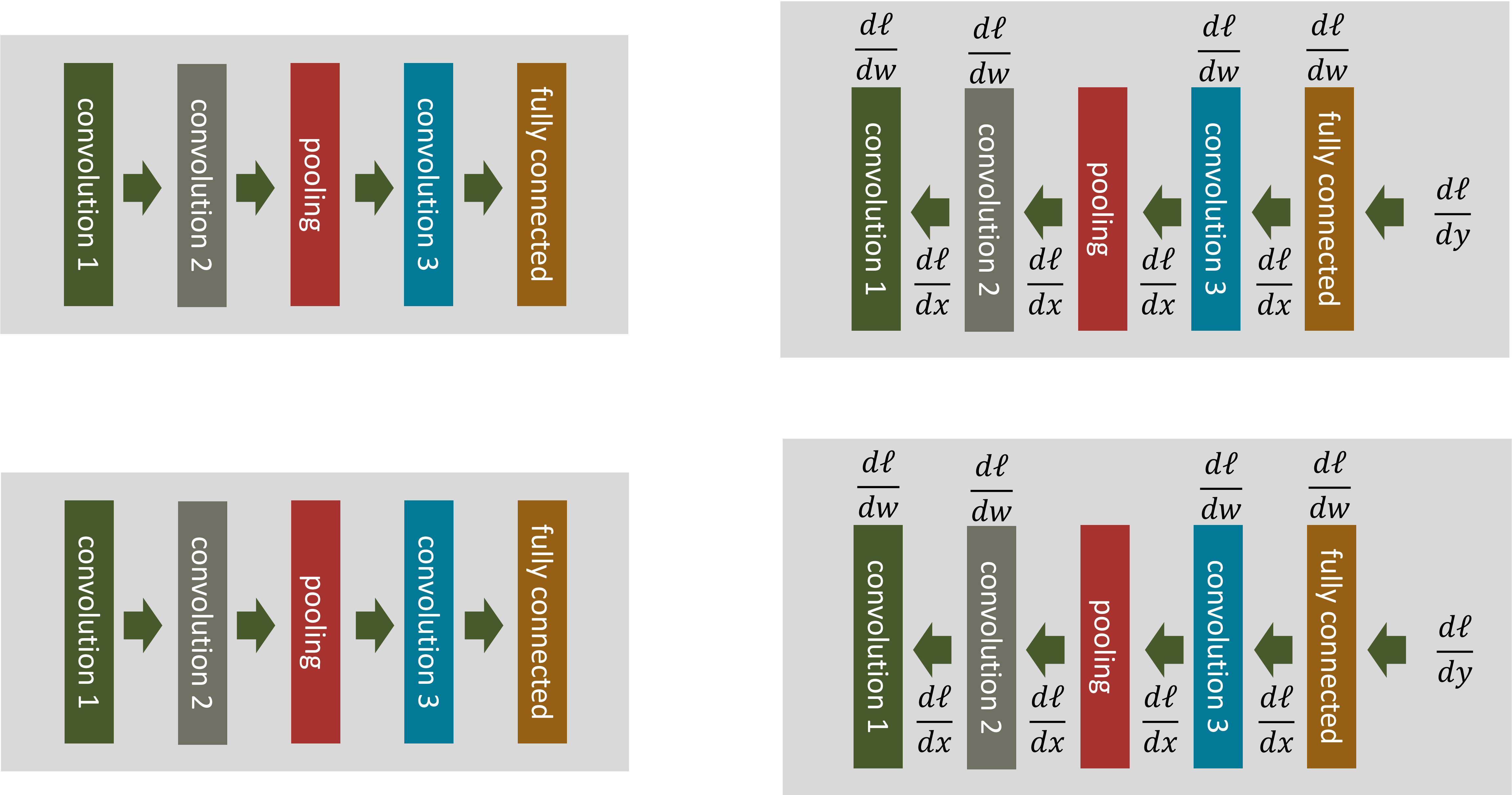




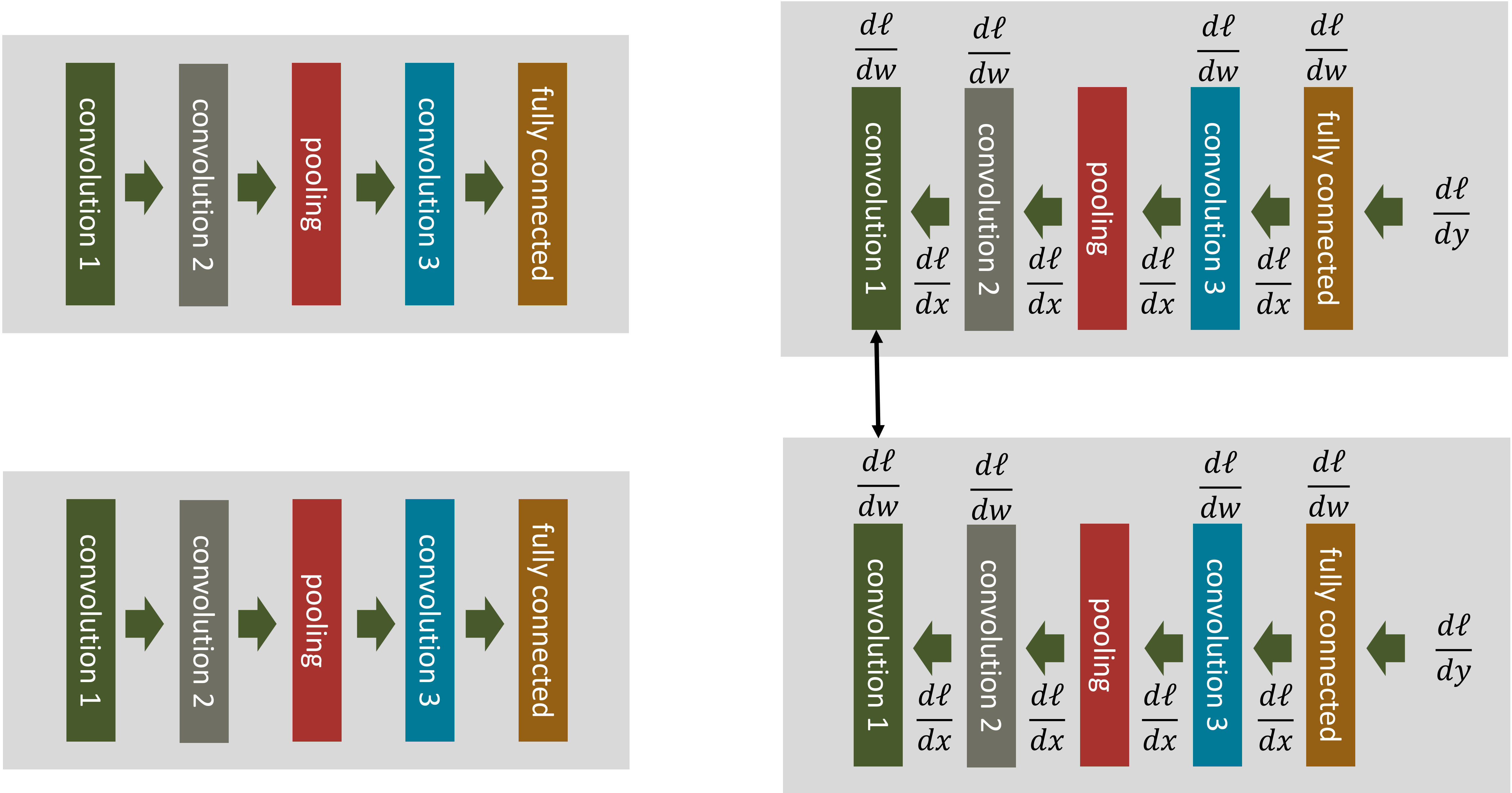




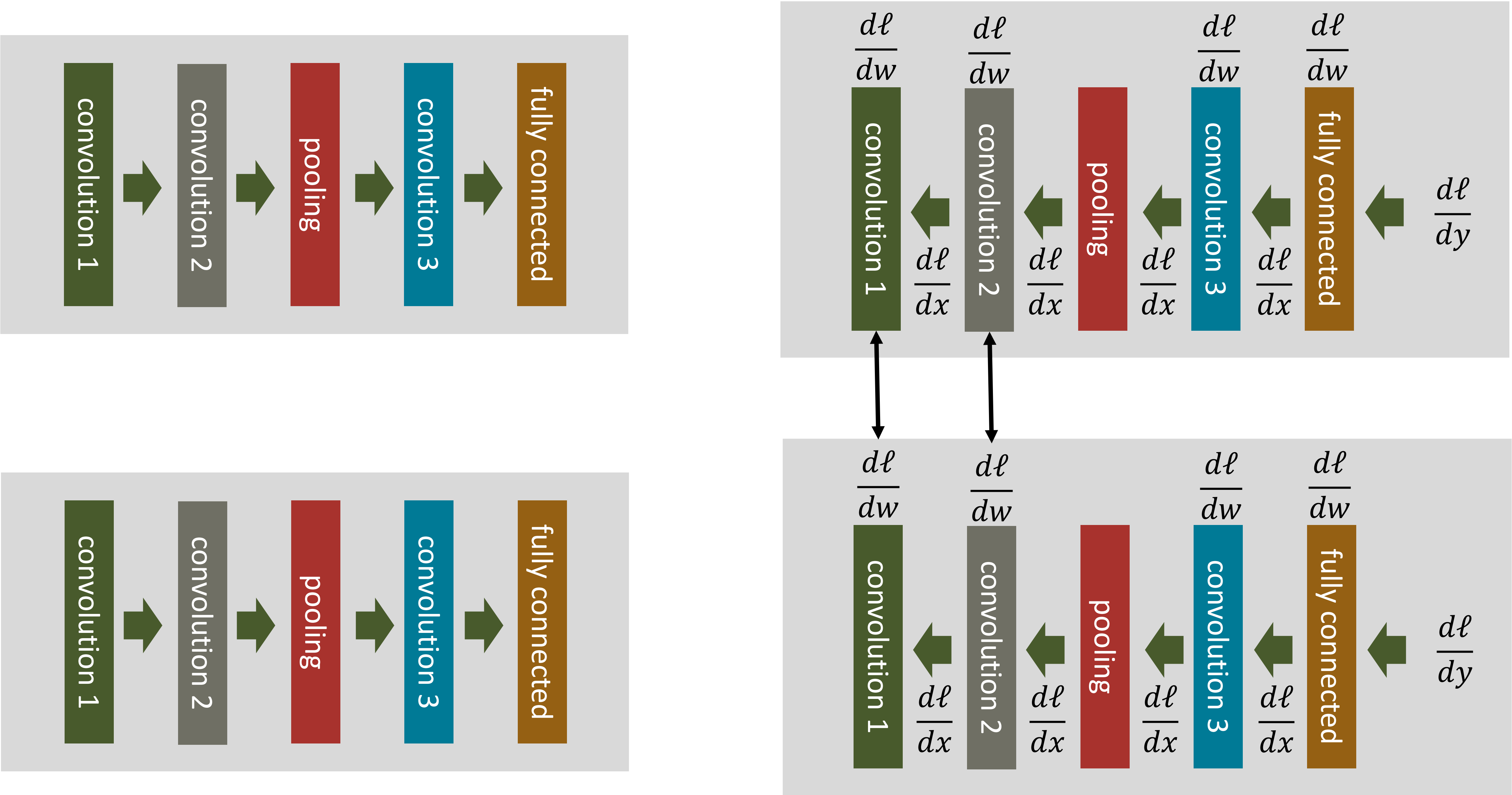
Distributed data-parallelism



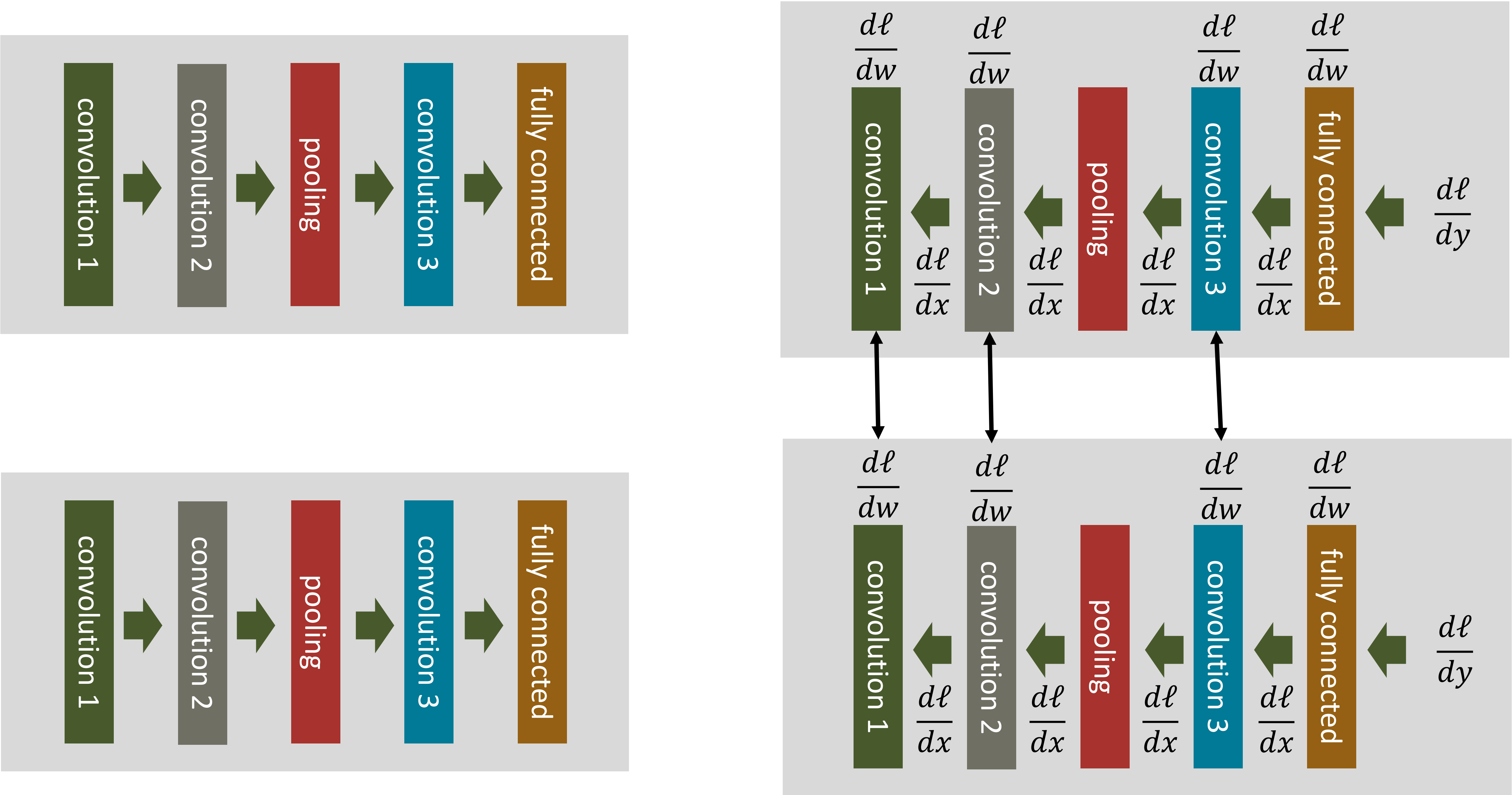
Distributed data-parallelism



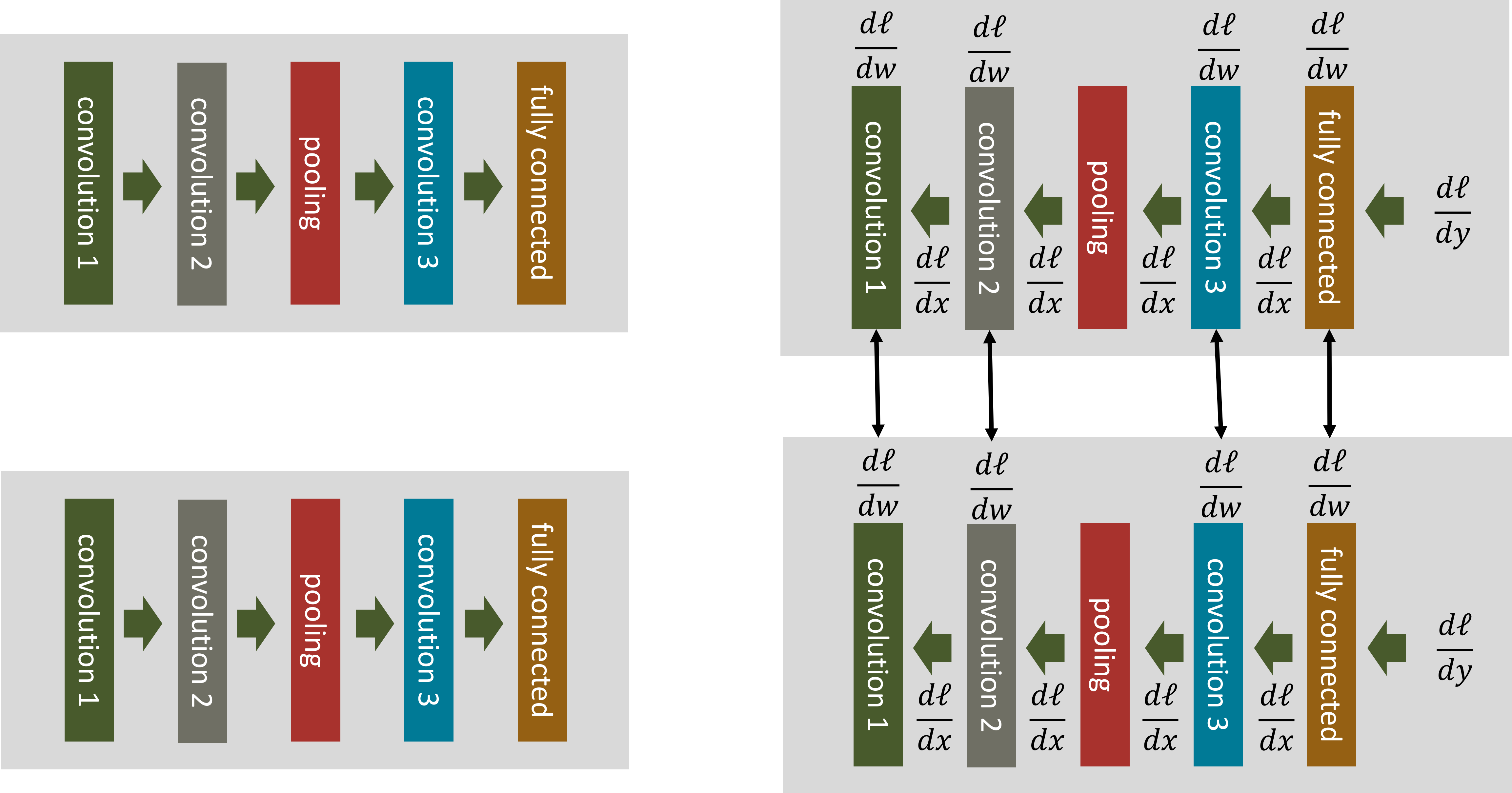
Distributed data-parallelism



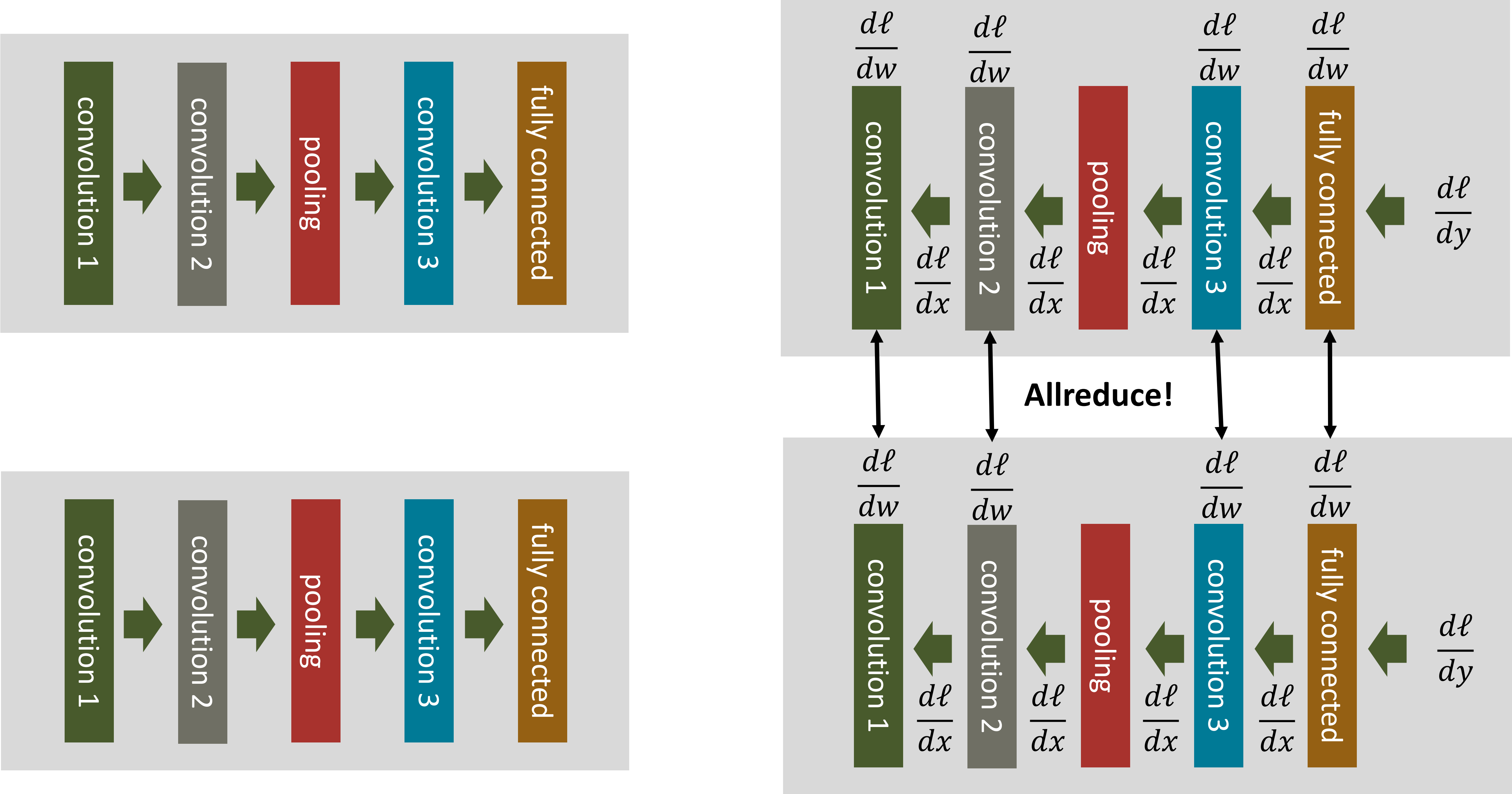
Distributed data-parallelism



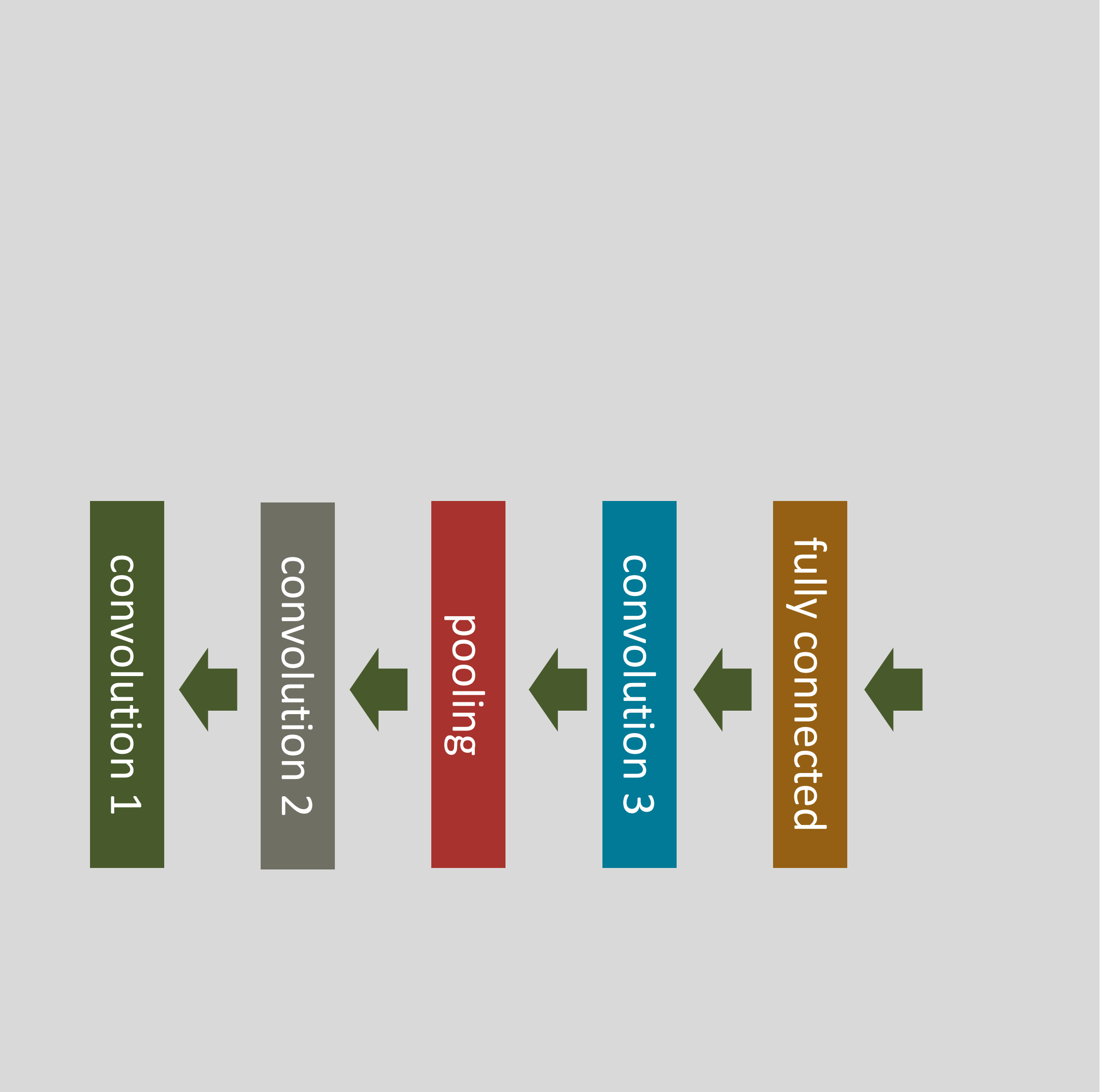
Distributed data-parallelism



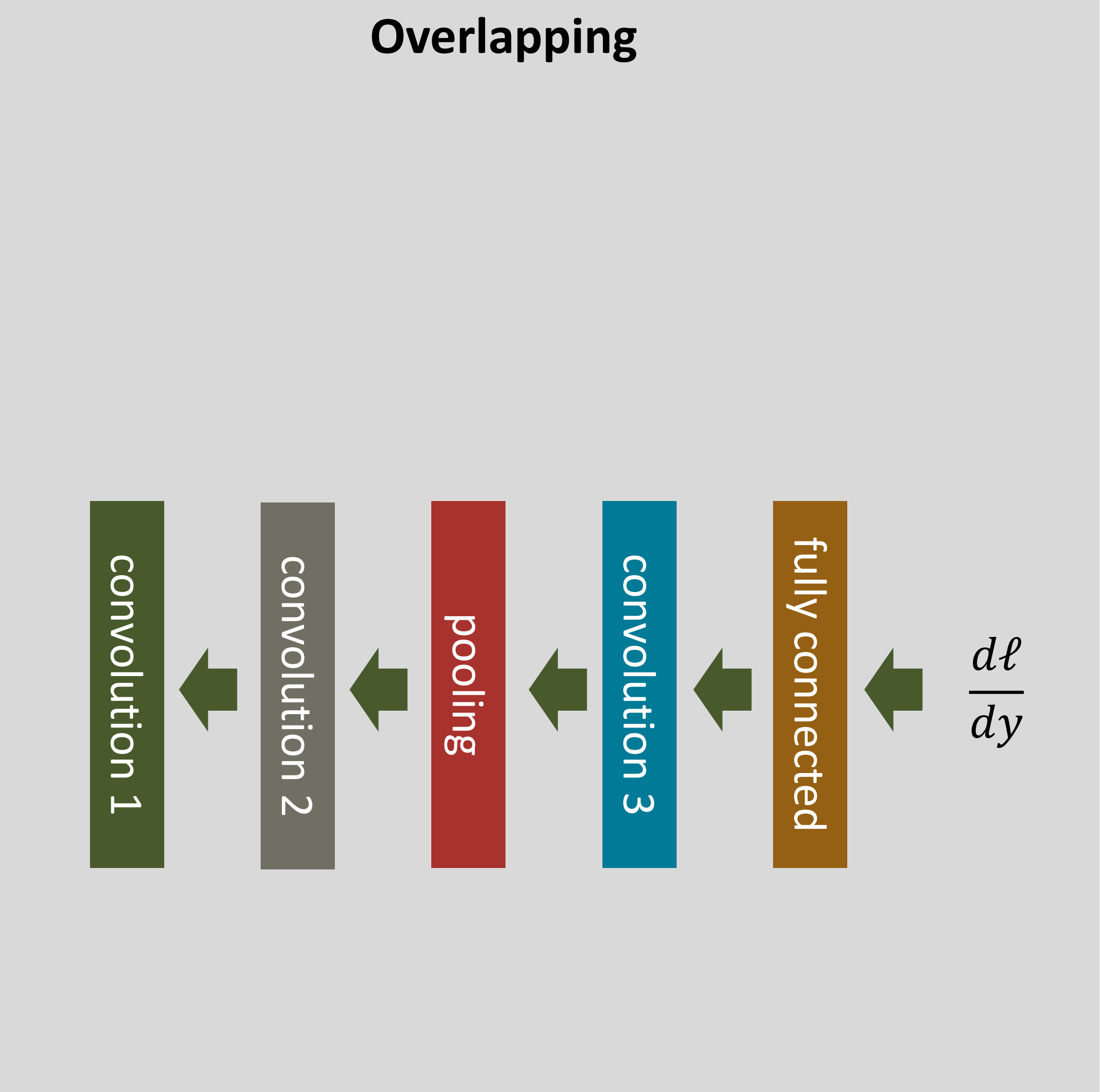
Distributed data-parallelism



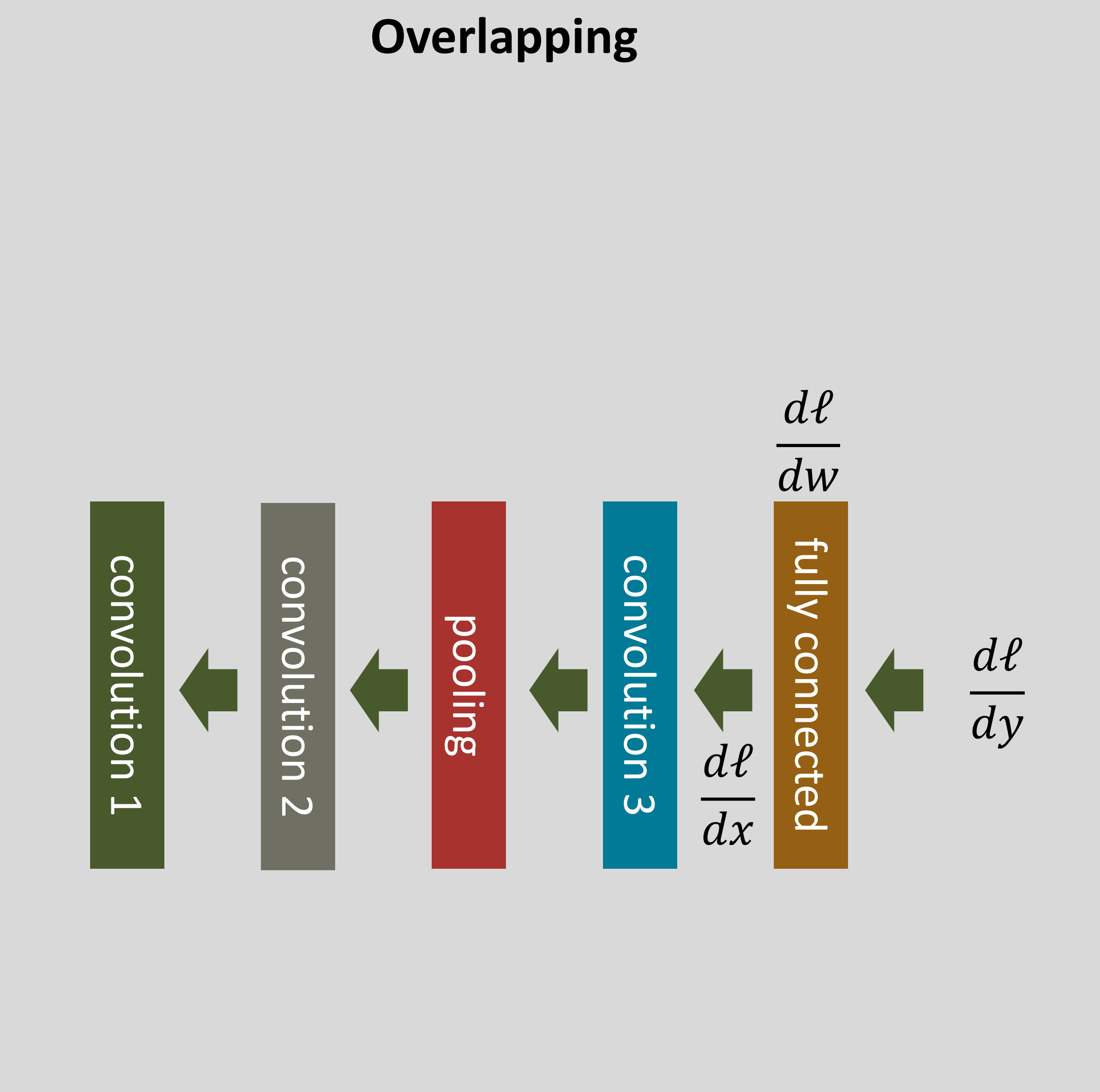
Distributed data-parallelism



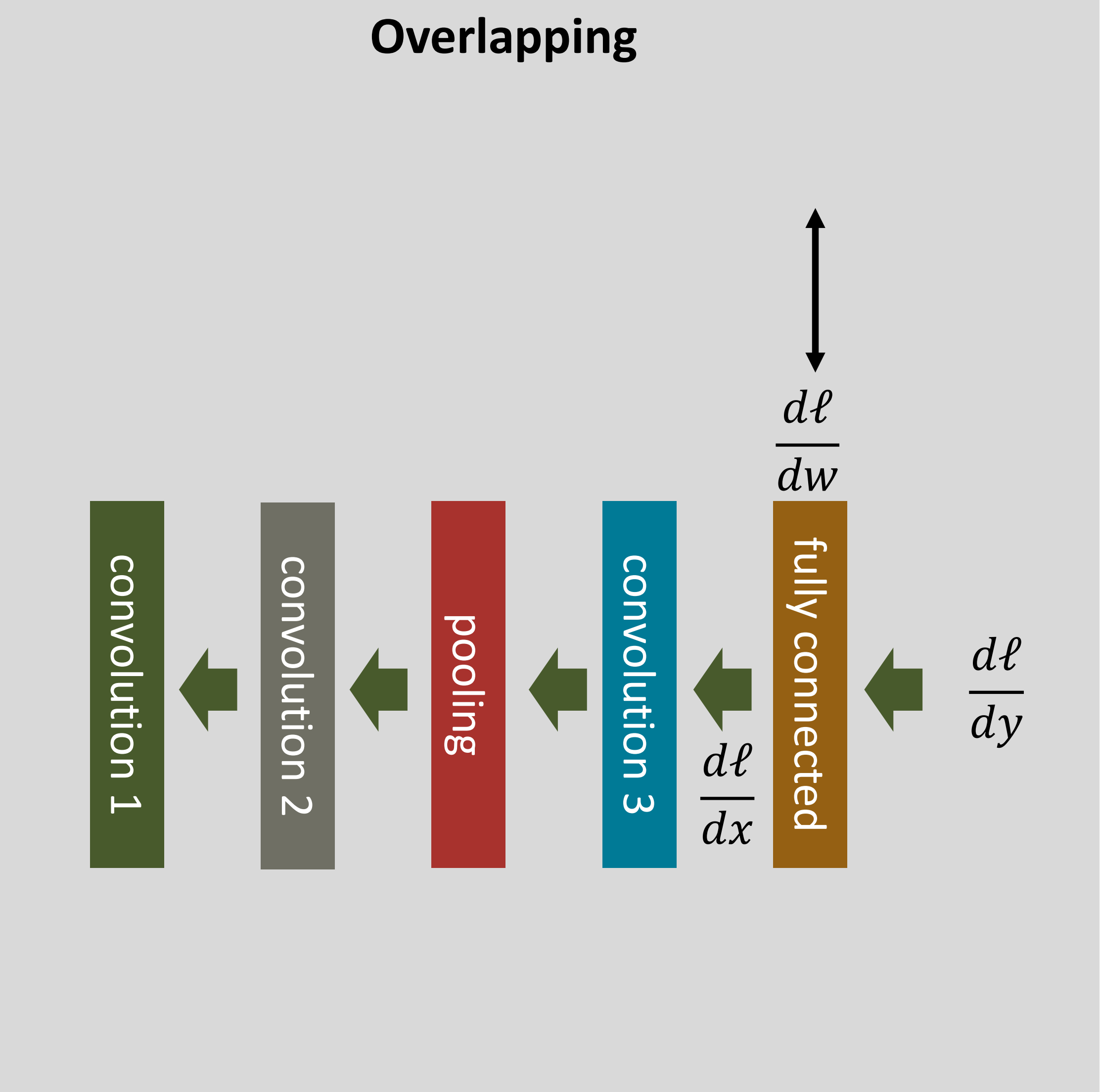
Distributed data-parallelism



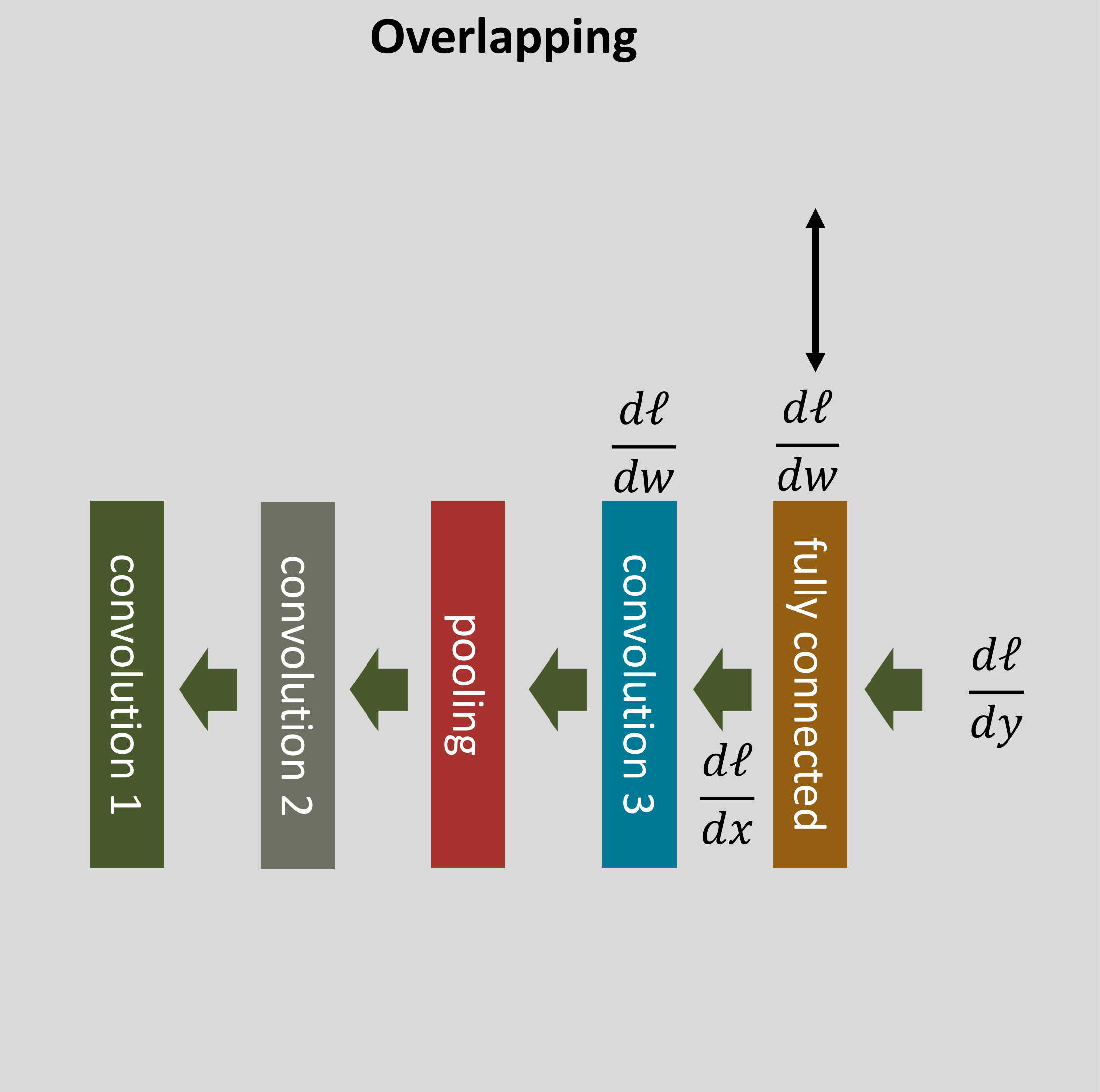
Distributed data-parallelism



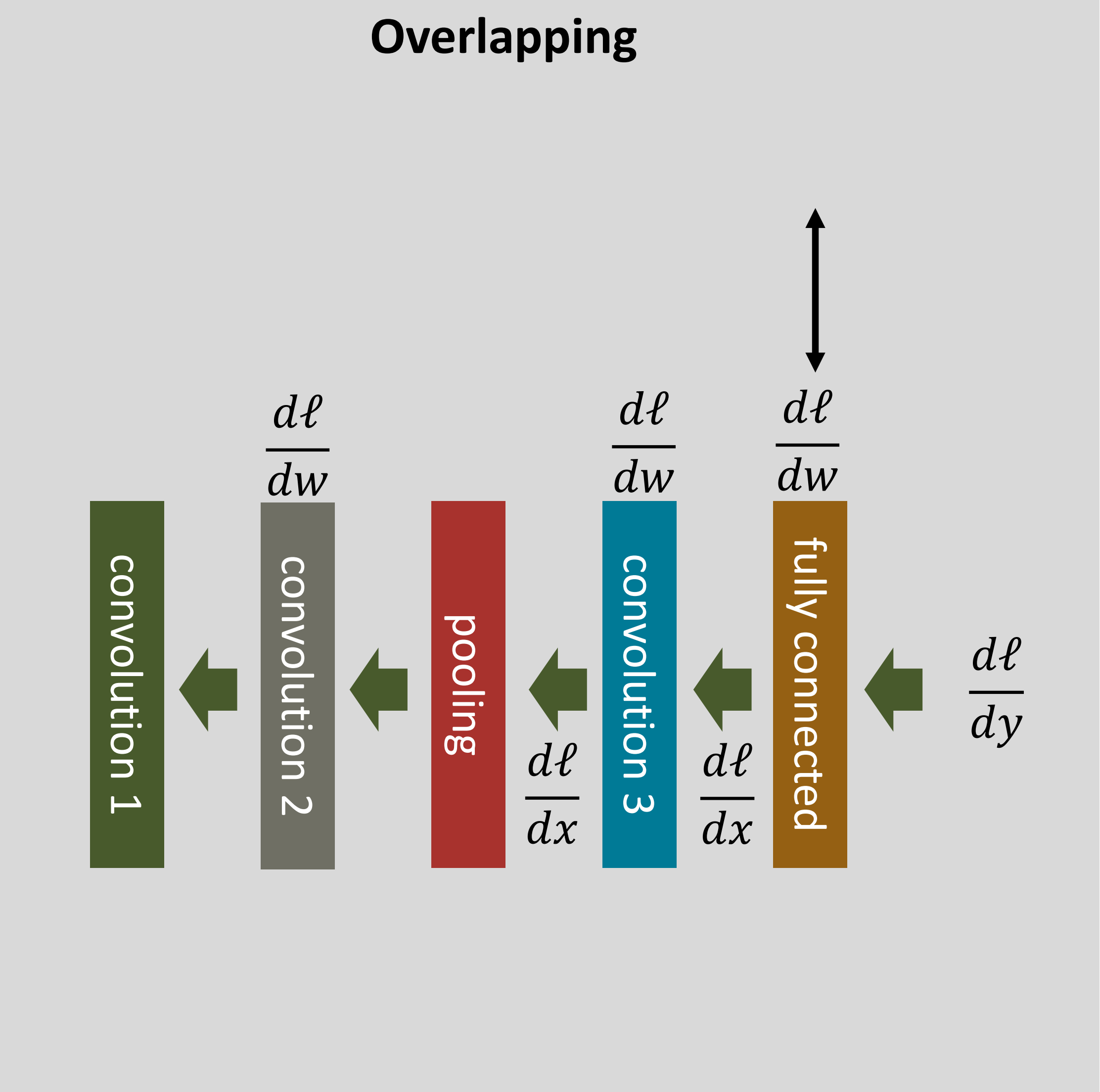
Distributed data-parallelism



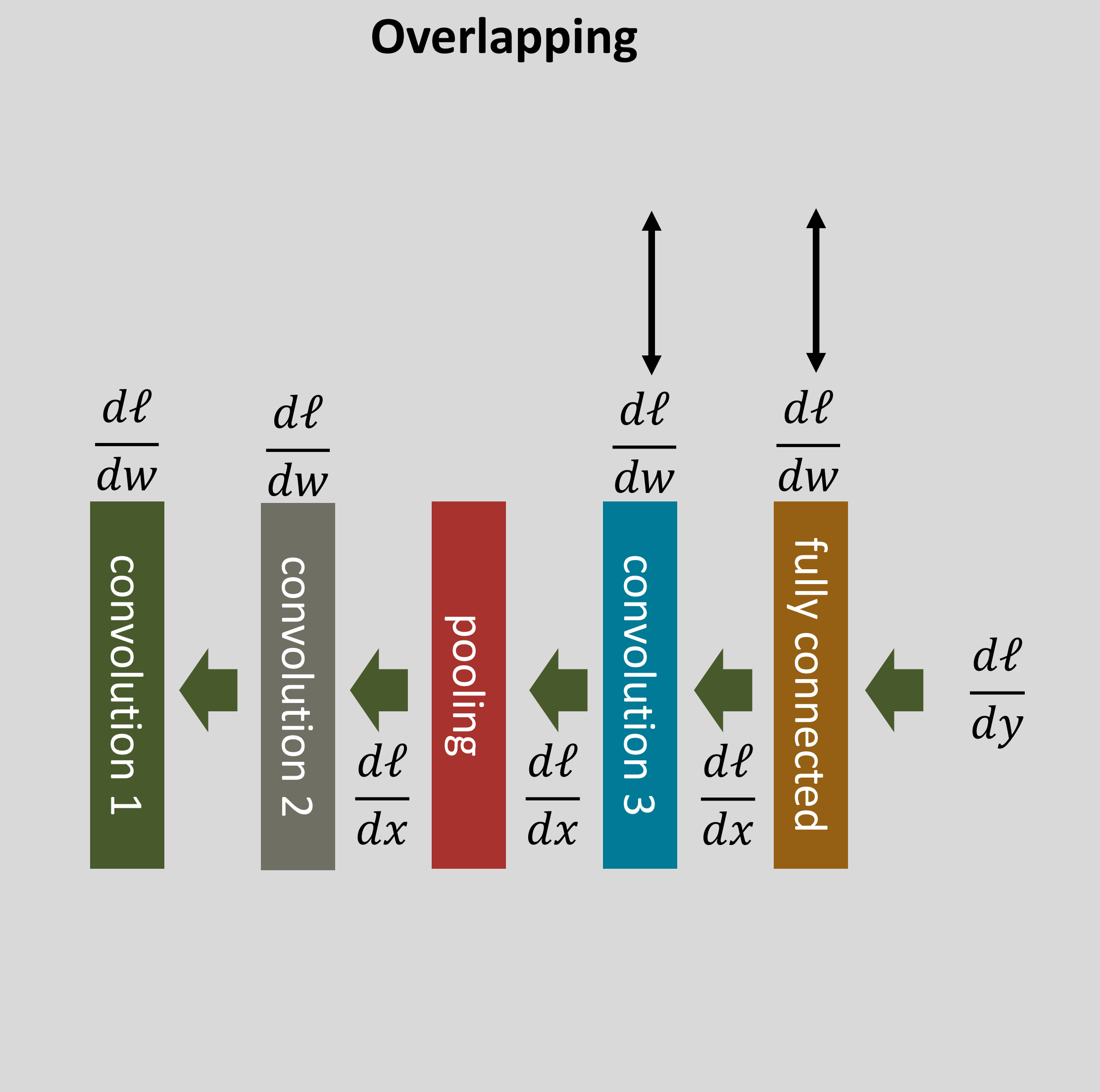
Distributed data-parallelism



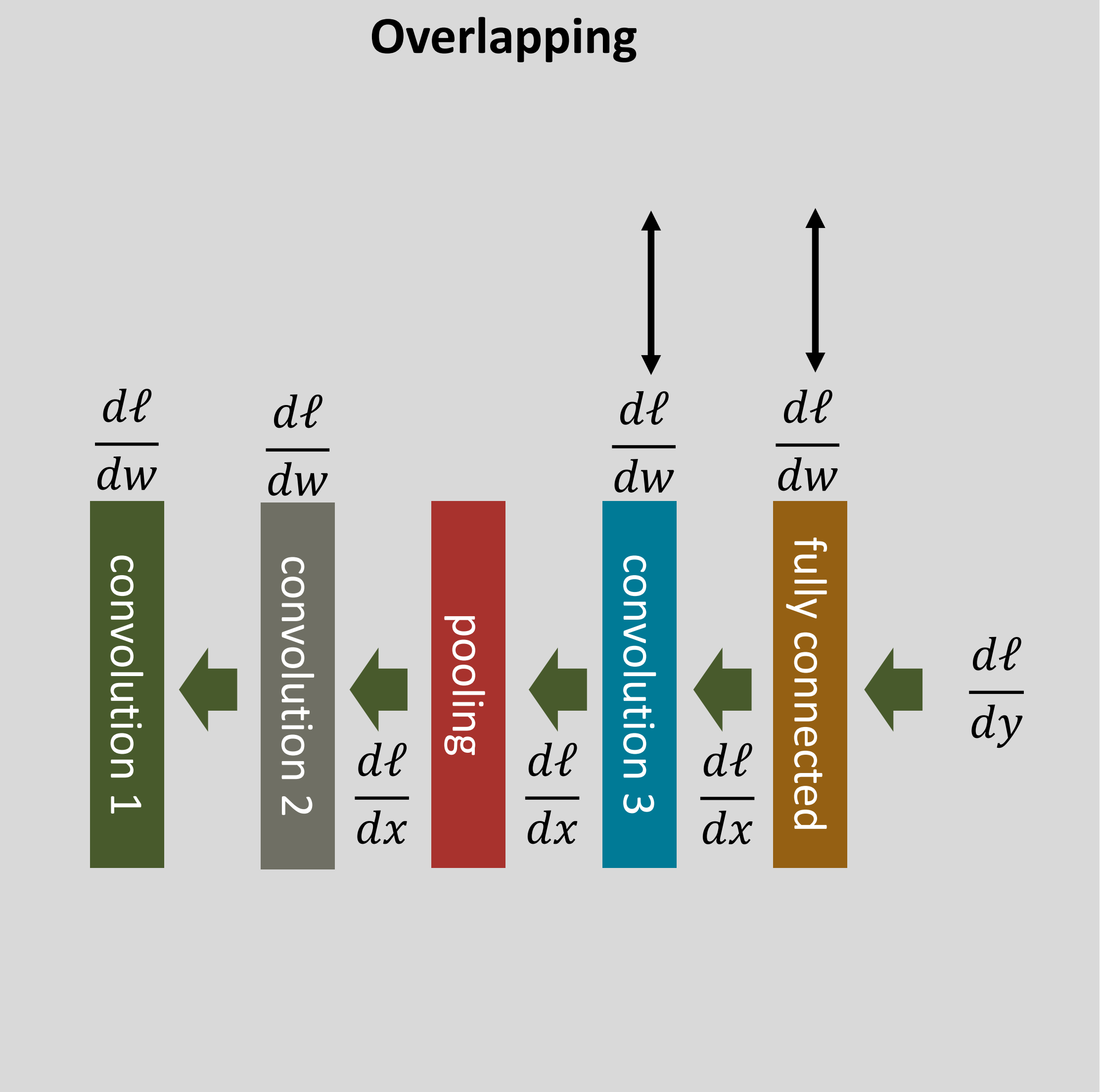
Distributed data-parallelism



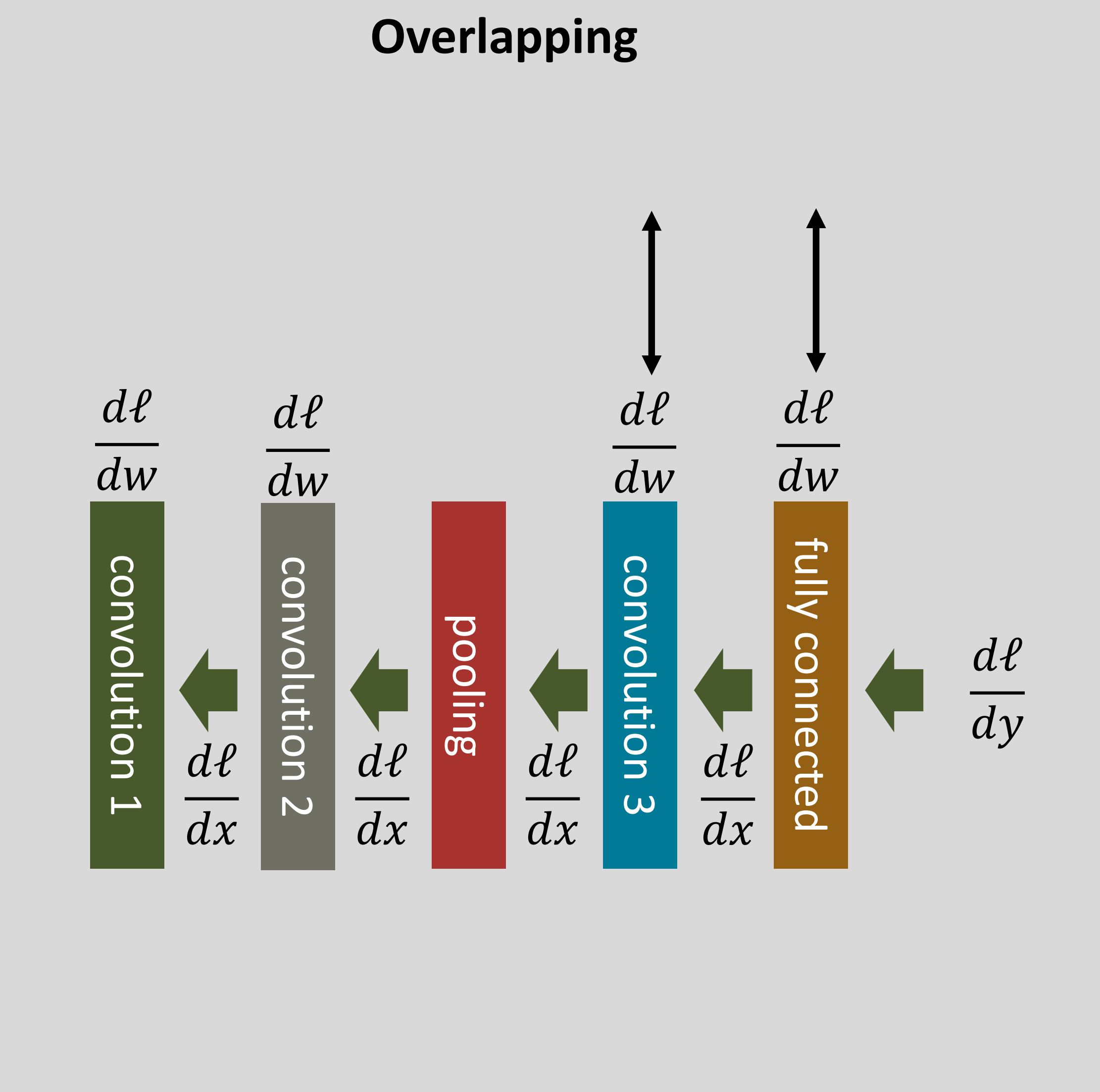
Distributed data-parallelism



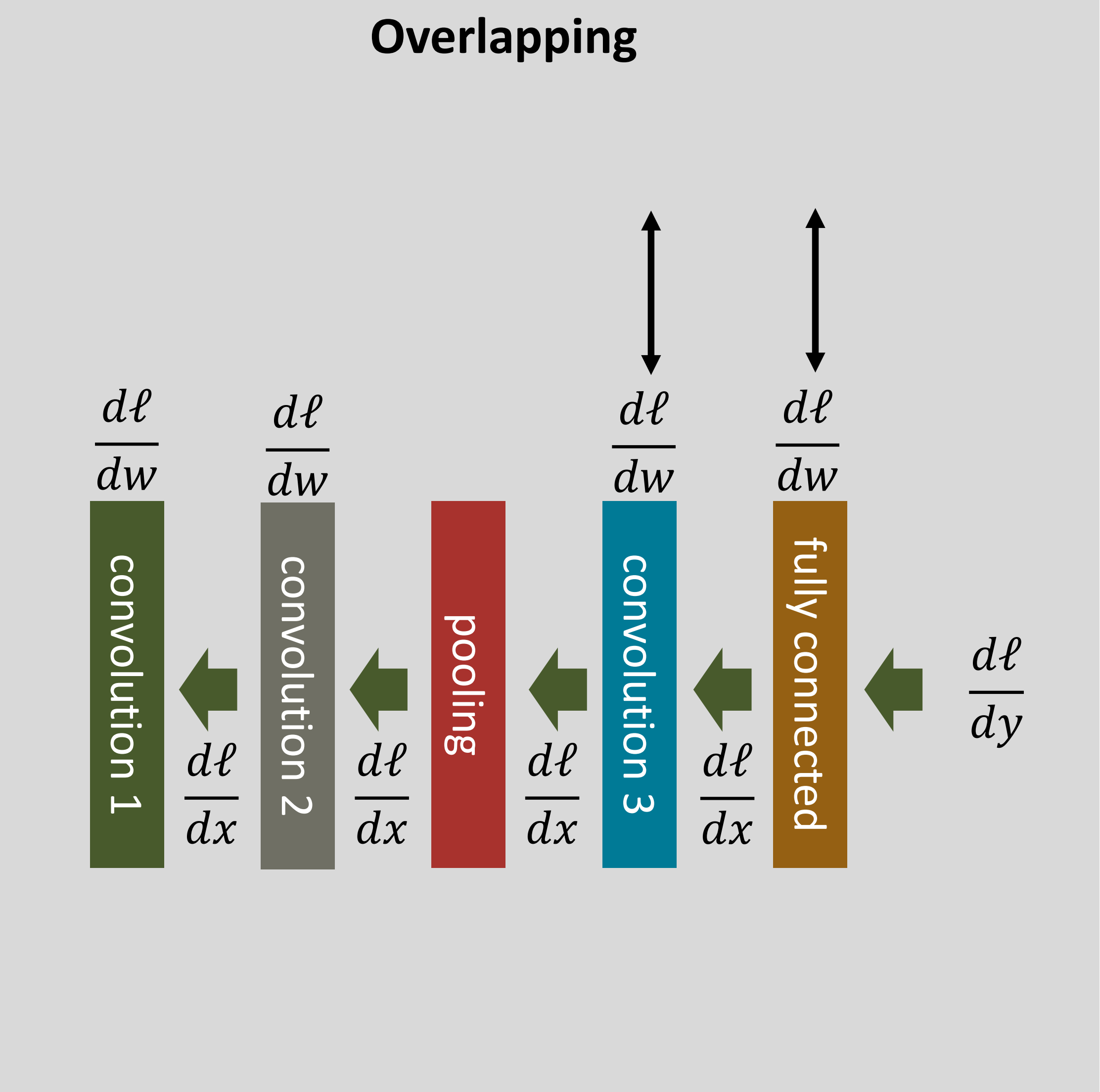
Distributed data-parallelism



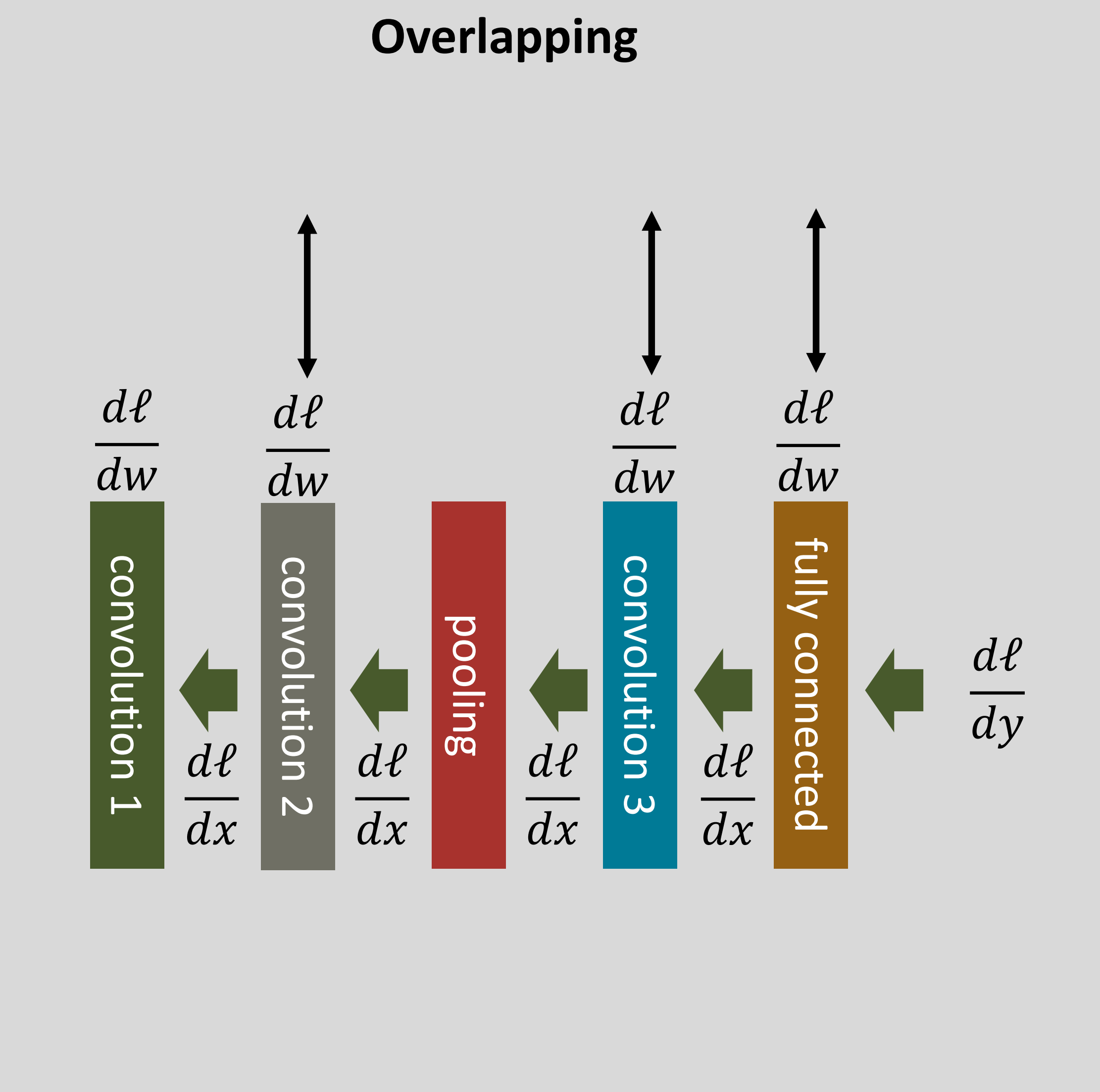
Distributed data-parallelism



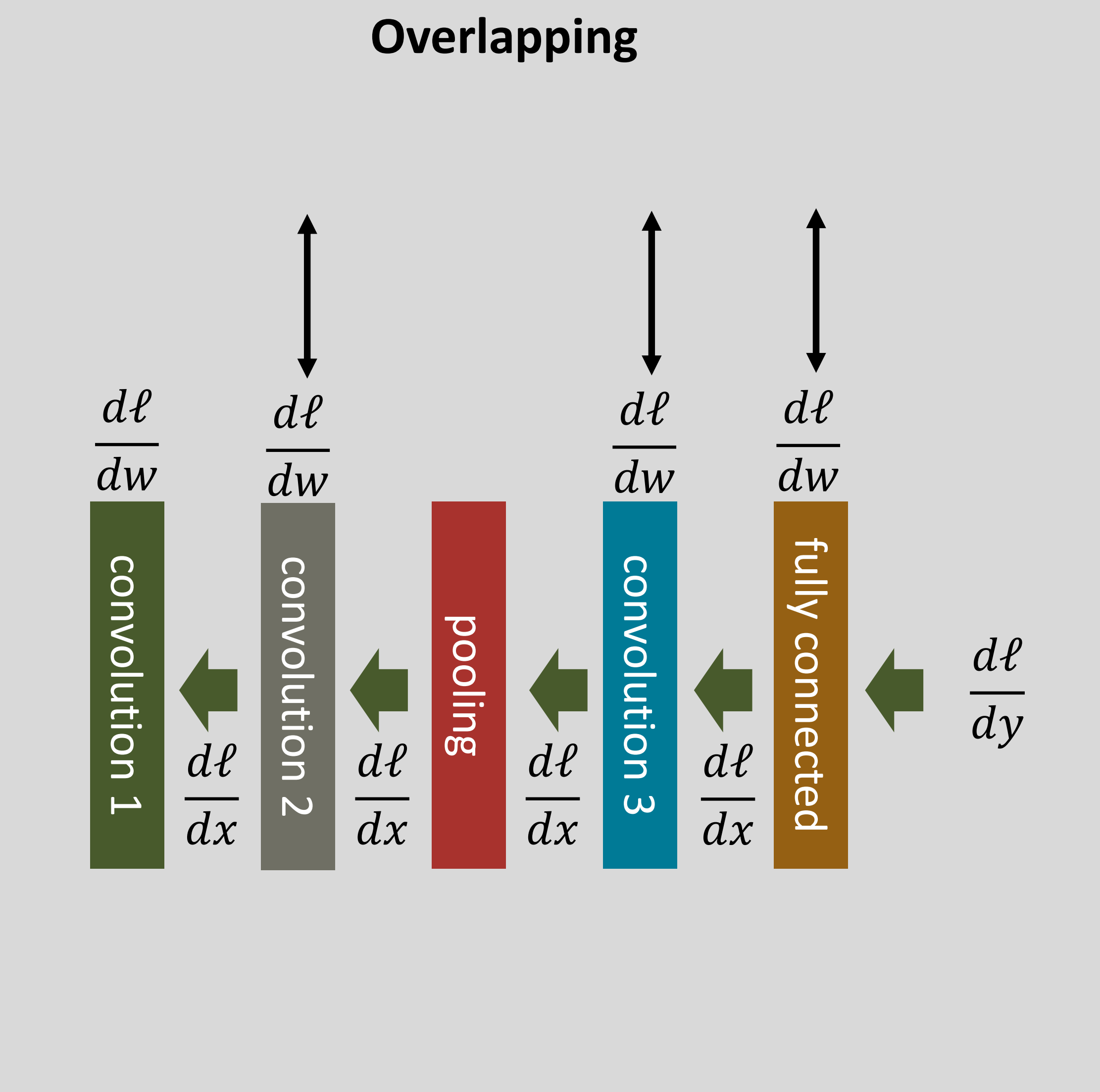
Distributed data-parallelism



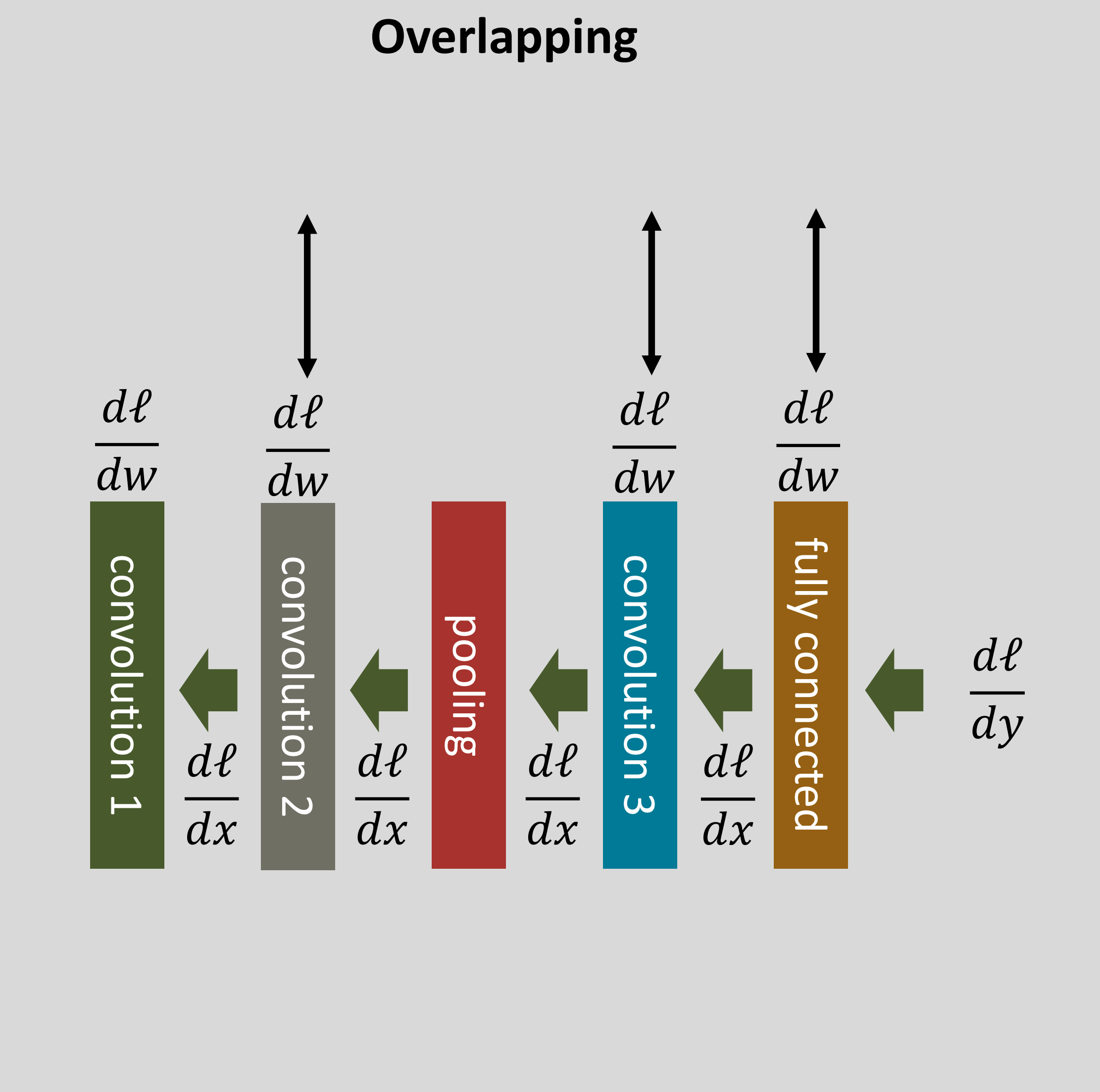
Distributed data-parallelism



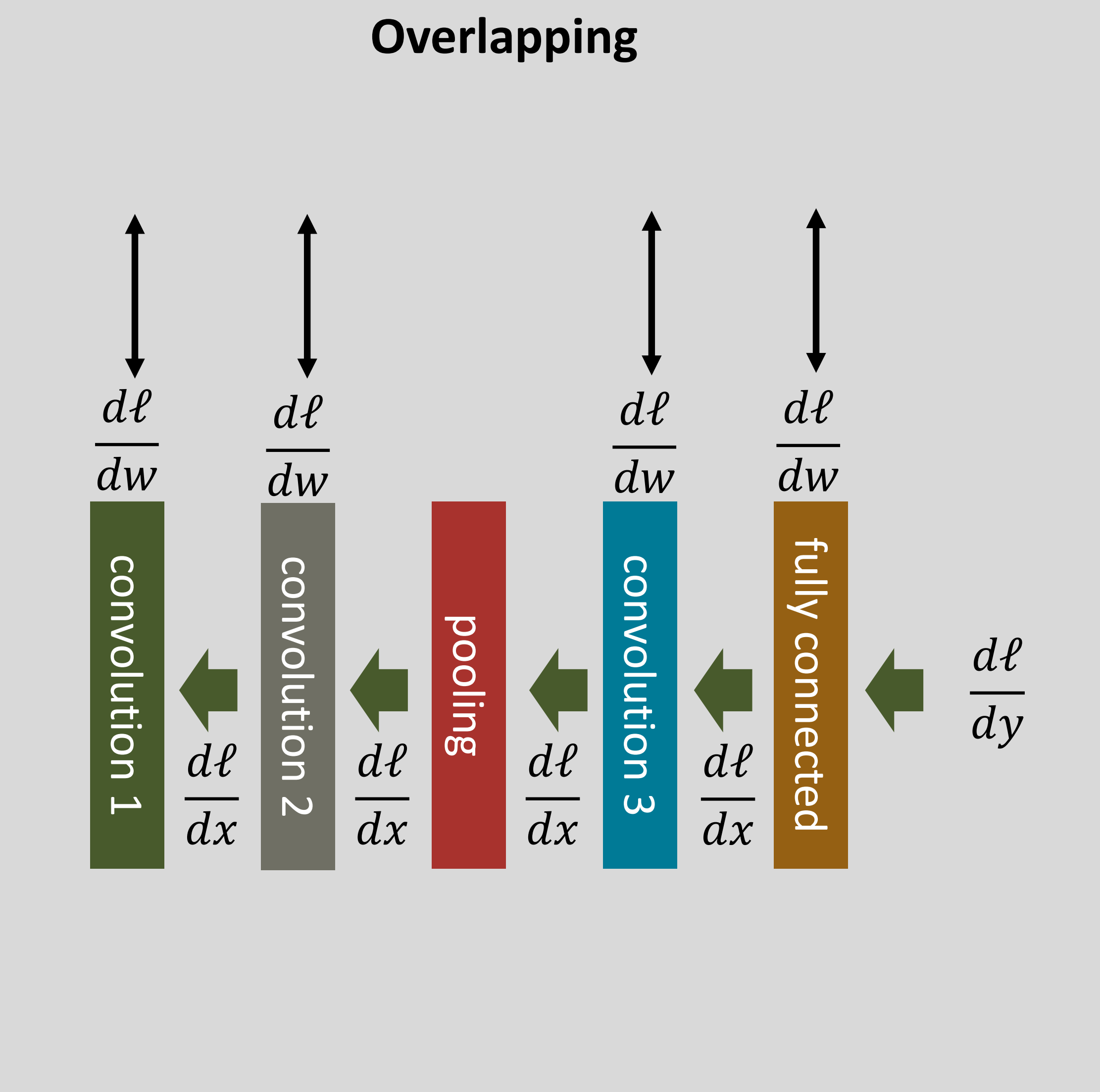
Distributed data-parallelism



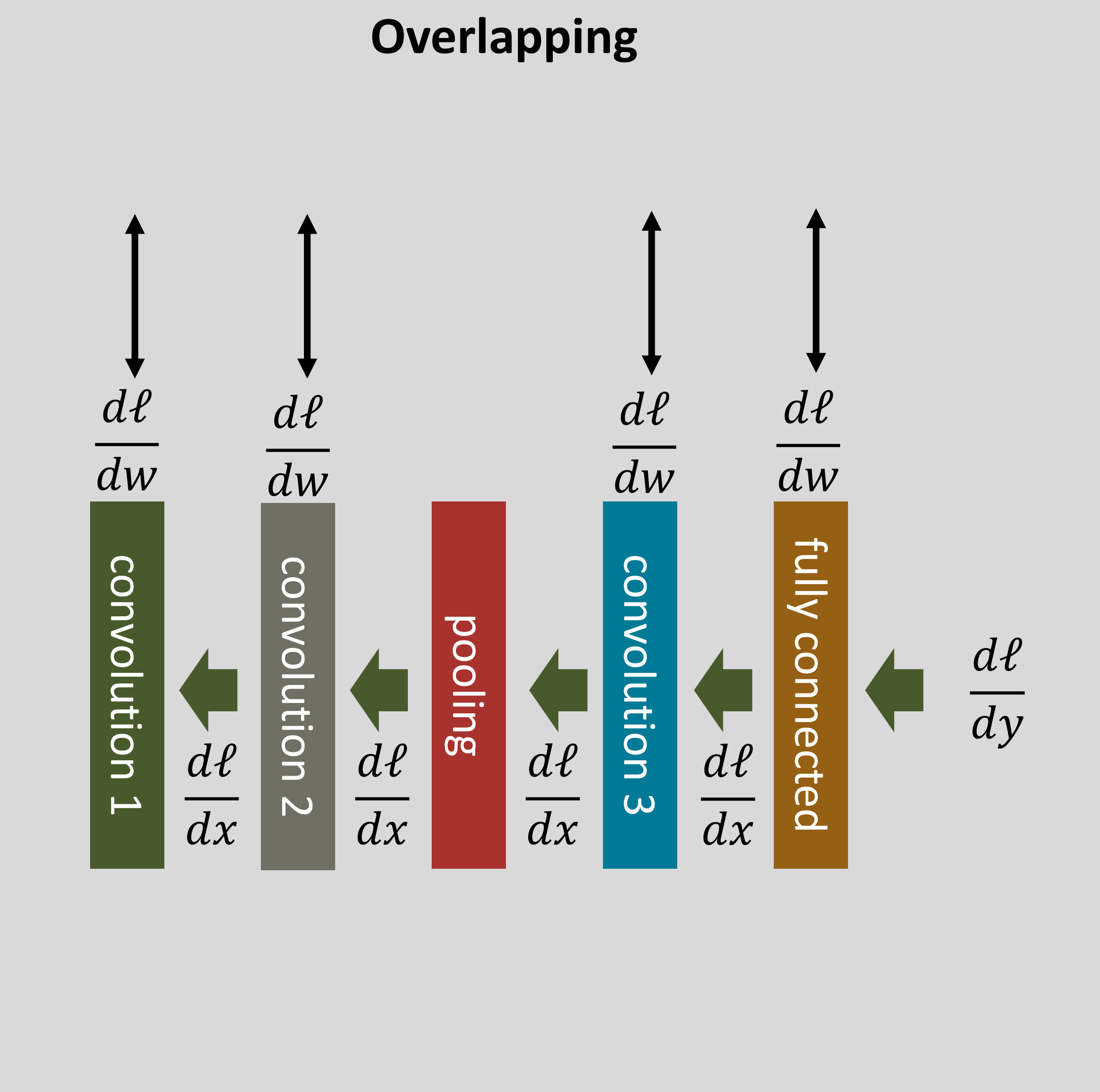
Distributed data-parallelism



Distributed data-parallelism

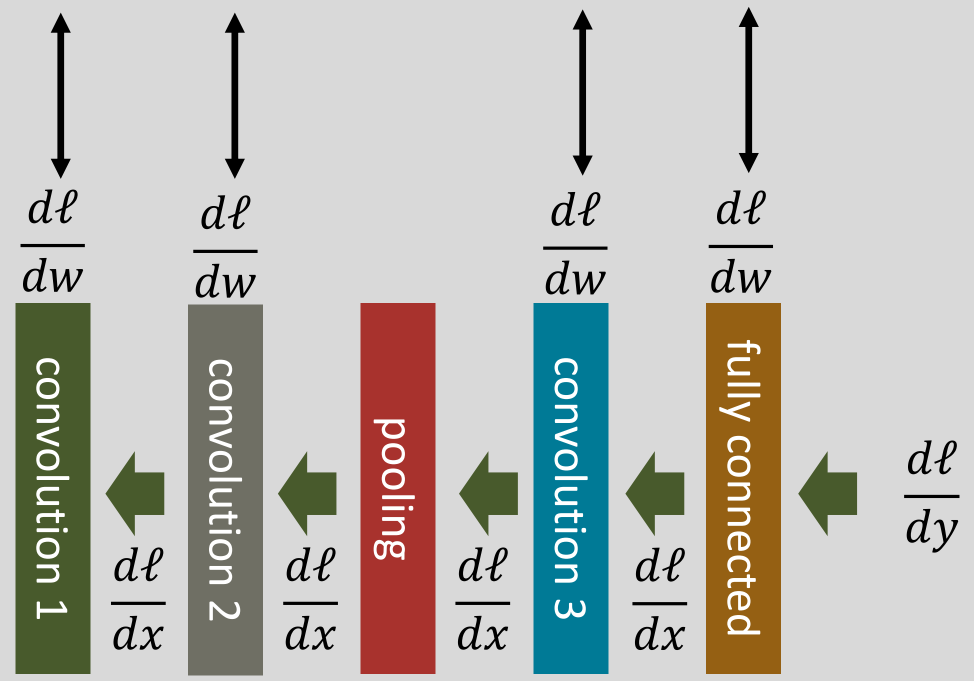


Distributed data-parallelism



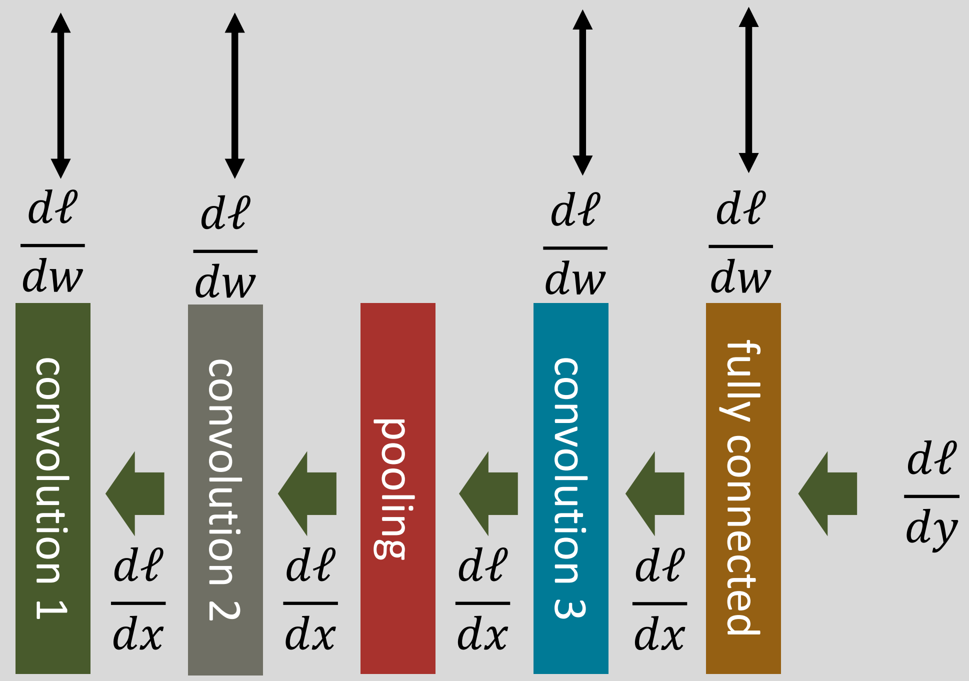
Distributed data-parallelism

Overlapping



Distributed data-parallelism

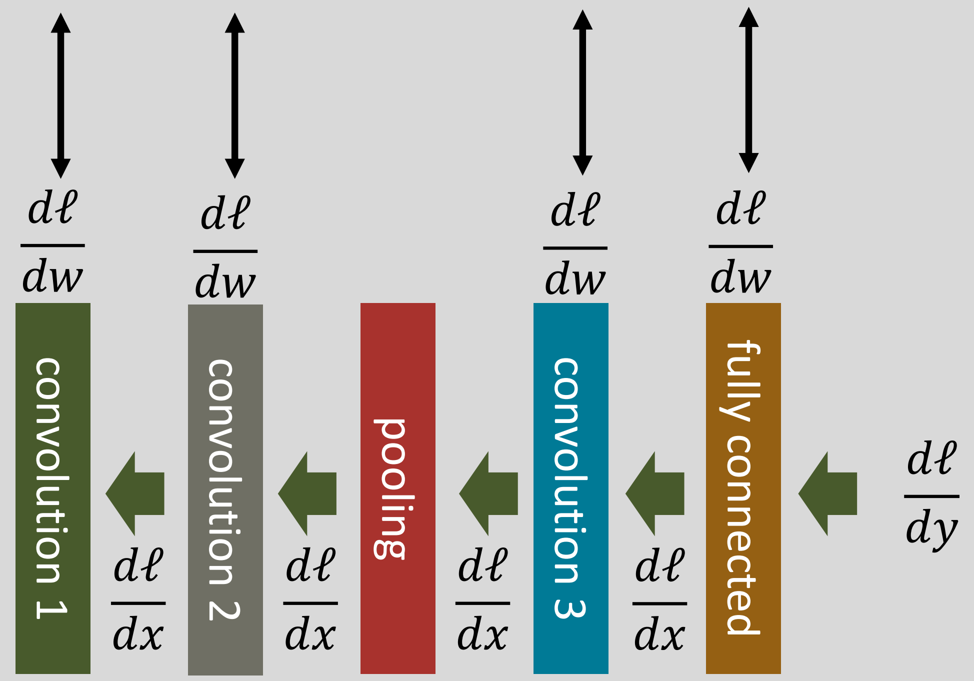
Overlapping



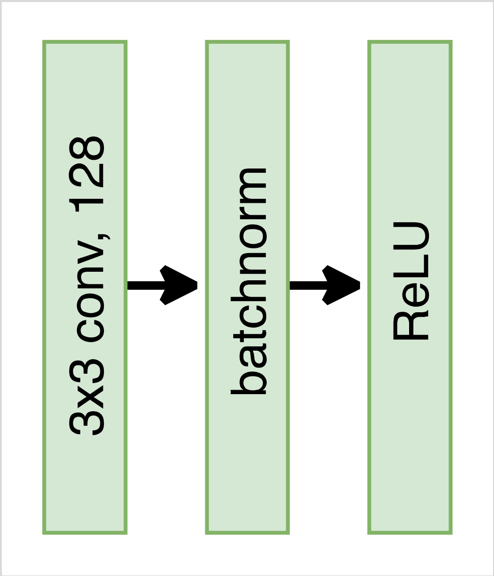
Fusion and algorithm selection

Distributed data-parallelism

Overlapping

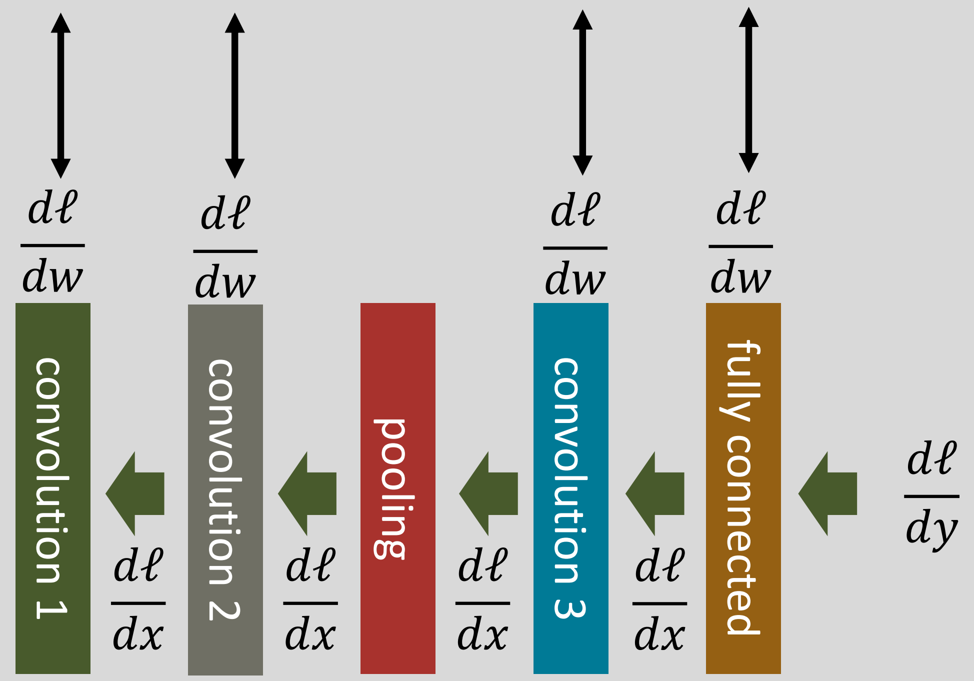


Fusion and algorithm selection

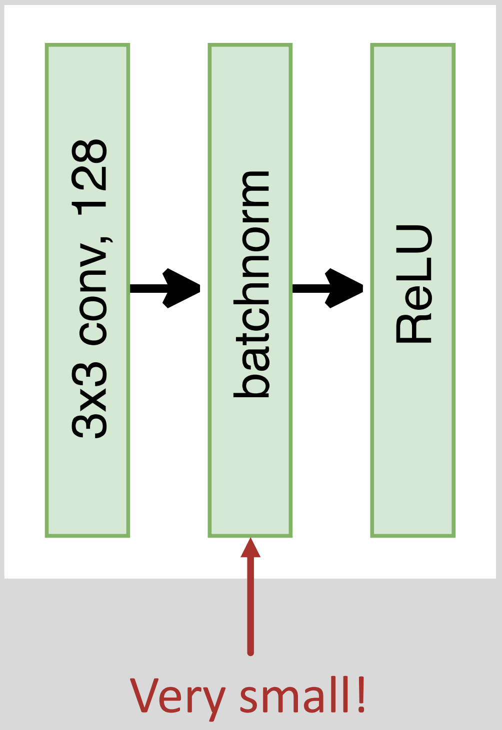


Distributed data-parallelism

Overlapping

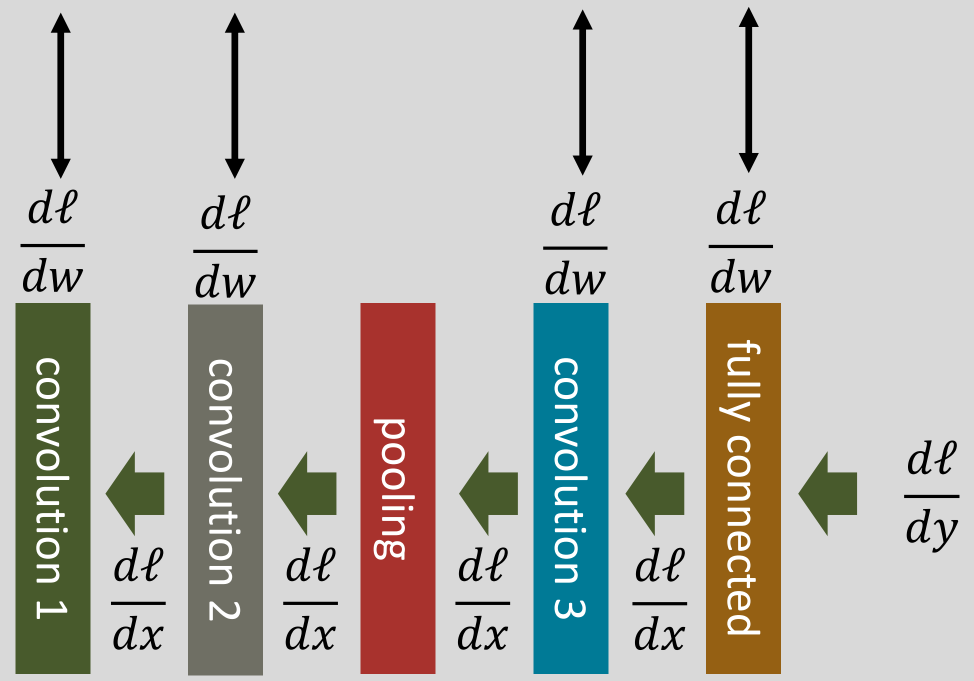


Fusion and algorithm selection

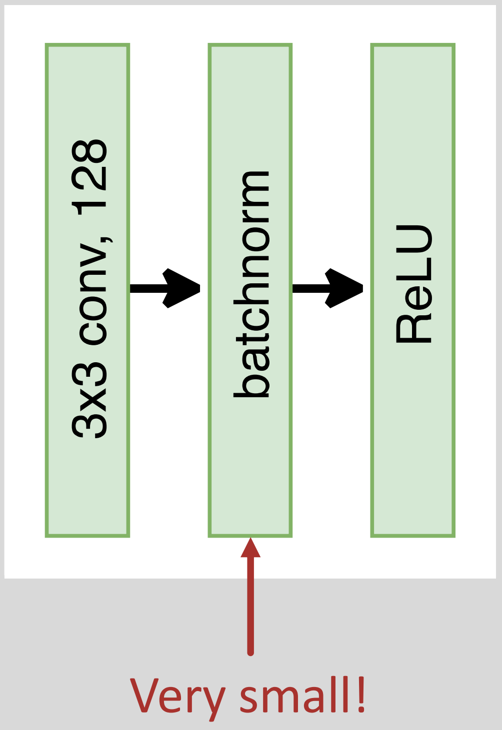


Distributed data-parallelism

Overlapping

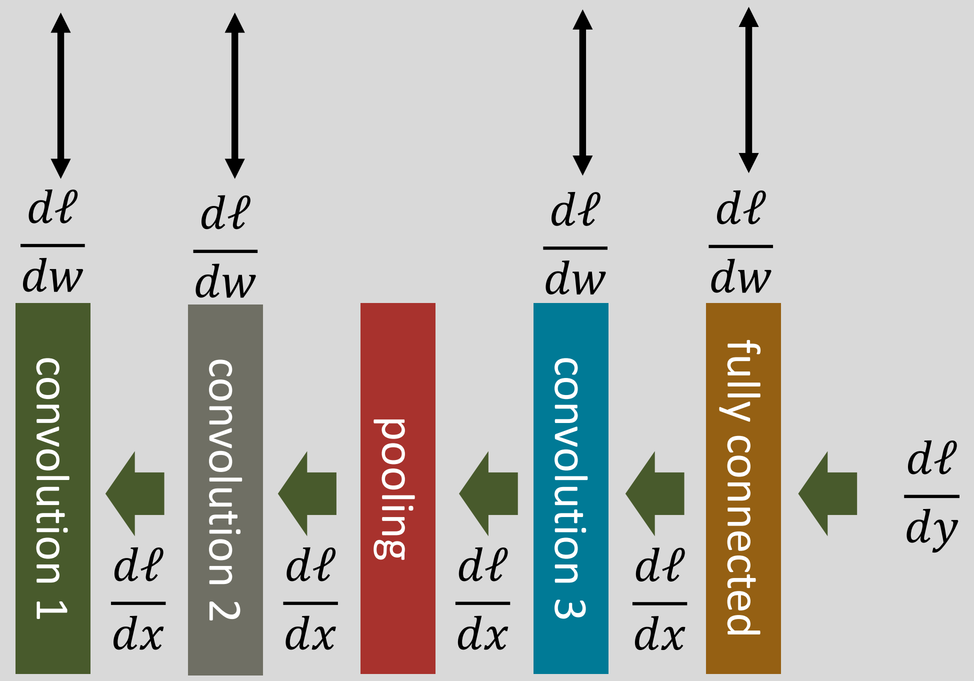


Fusion and algorithm selection

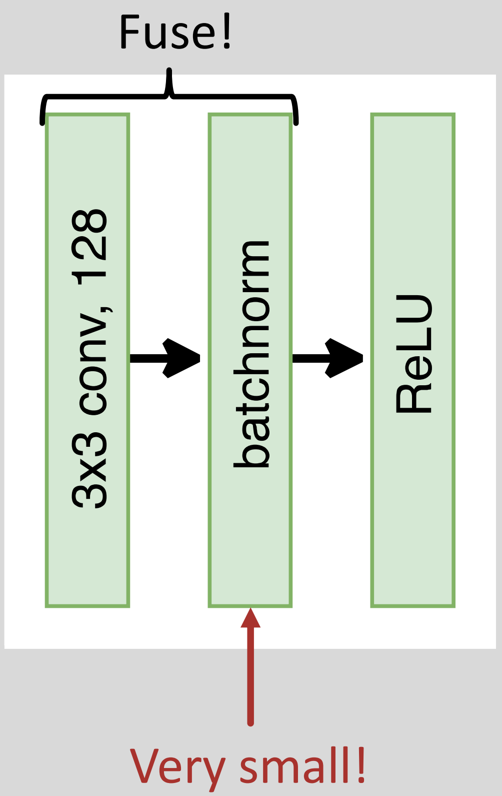


Distributed data-parallelism

Overlapping

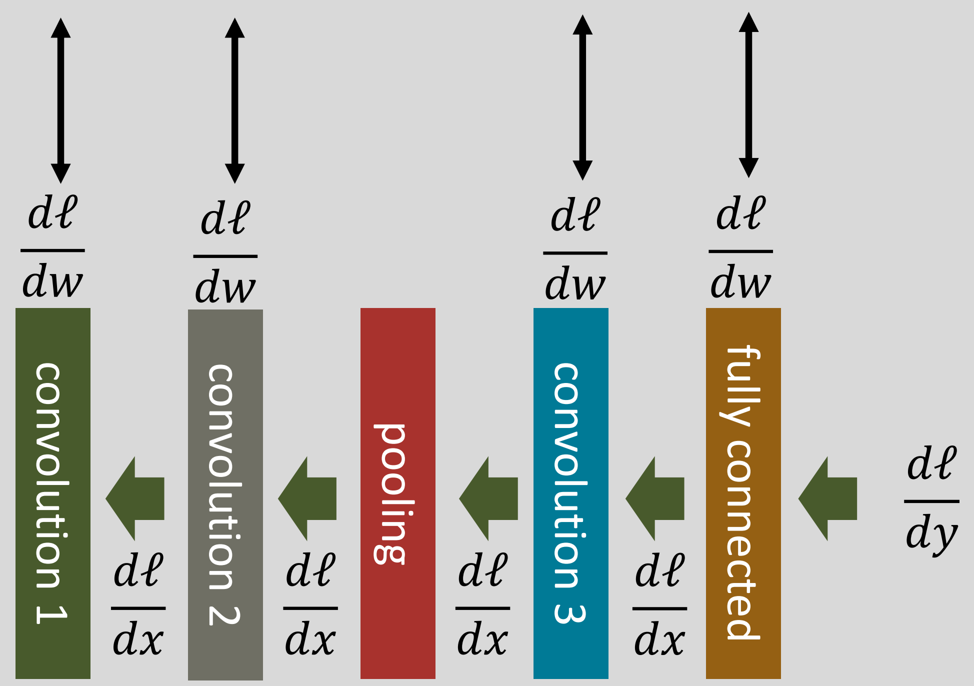


Fusion and algorithm selection

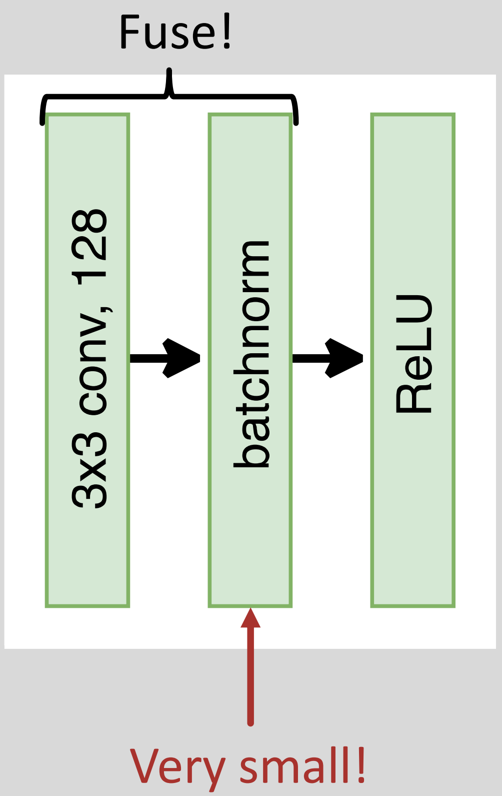


Distributed data-parallelism

Overlapping

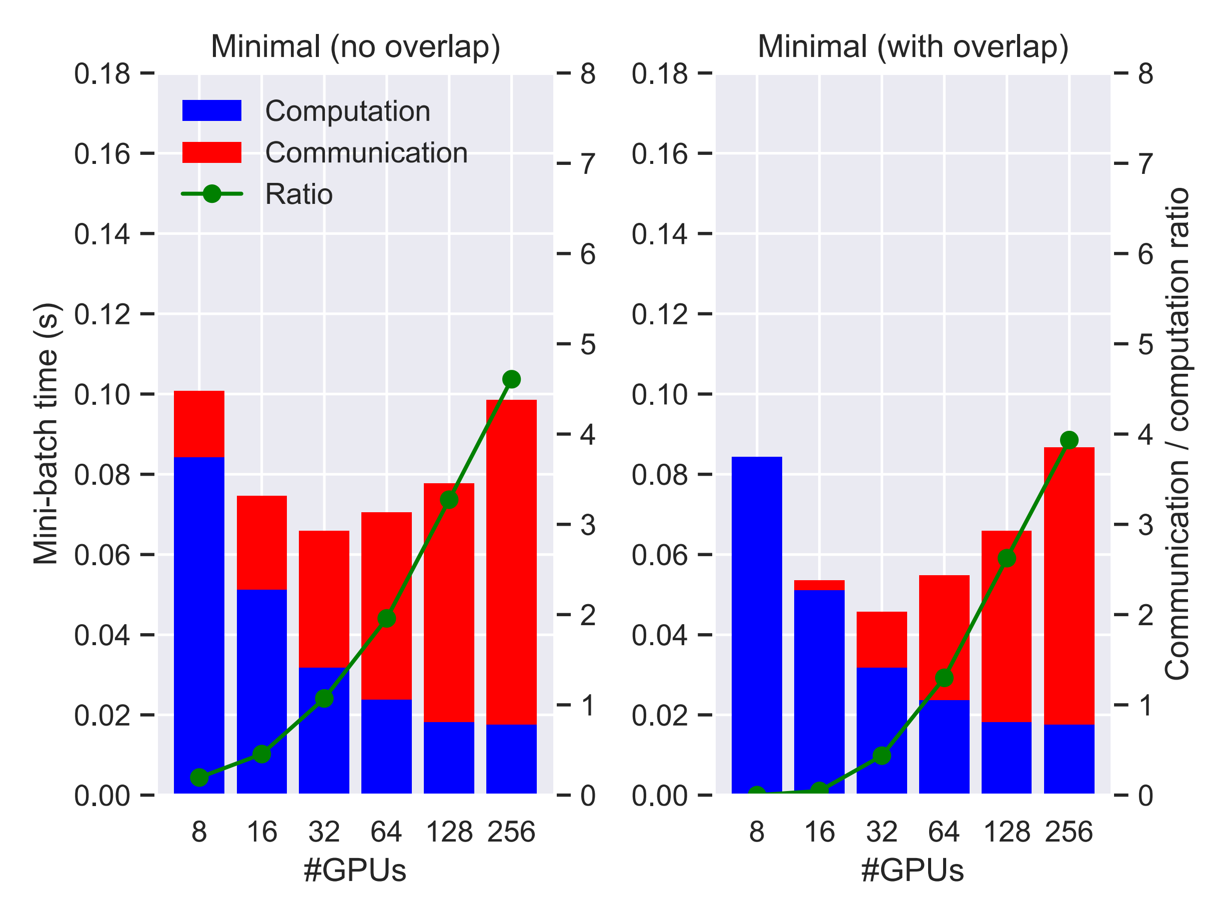


Fusion and algorithm selection



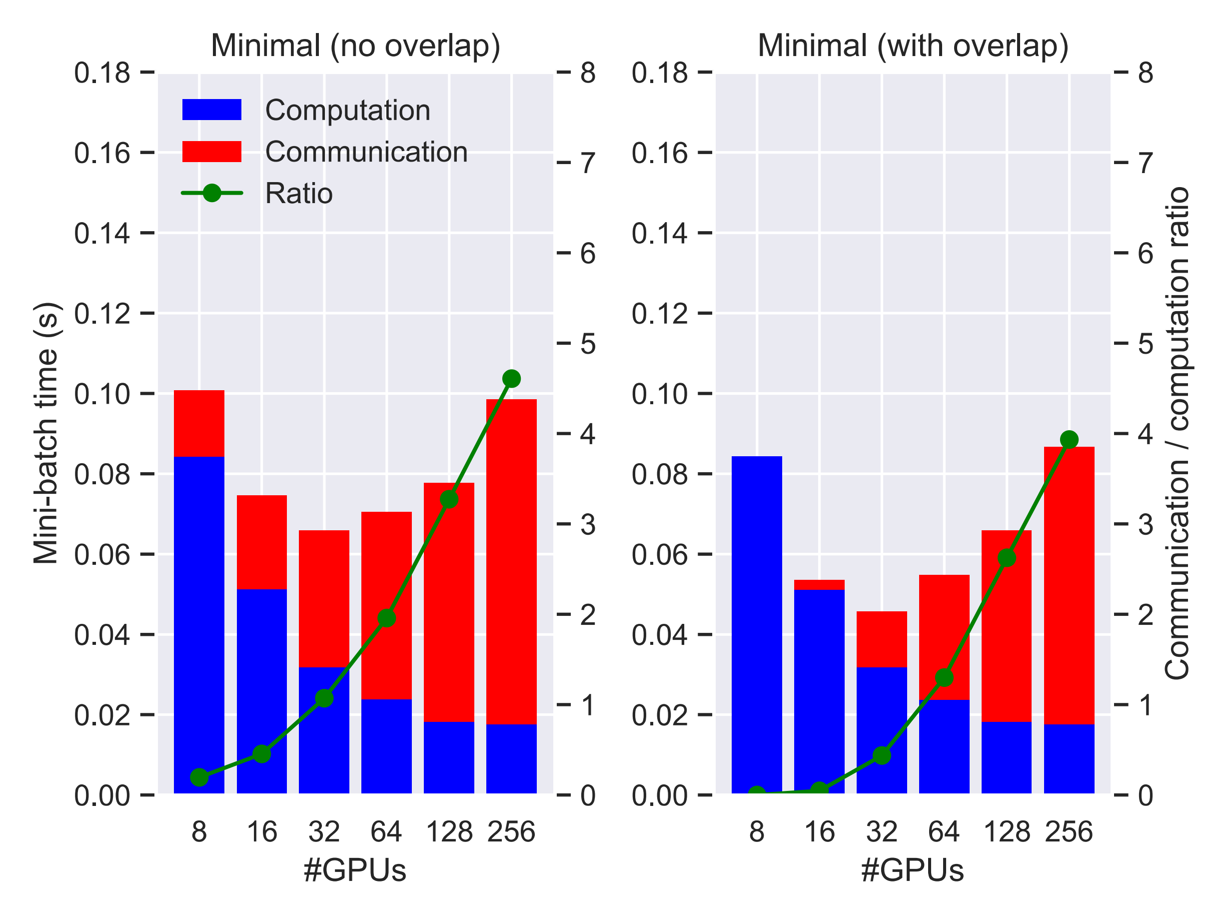
Distributed data-parallelism

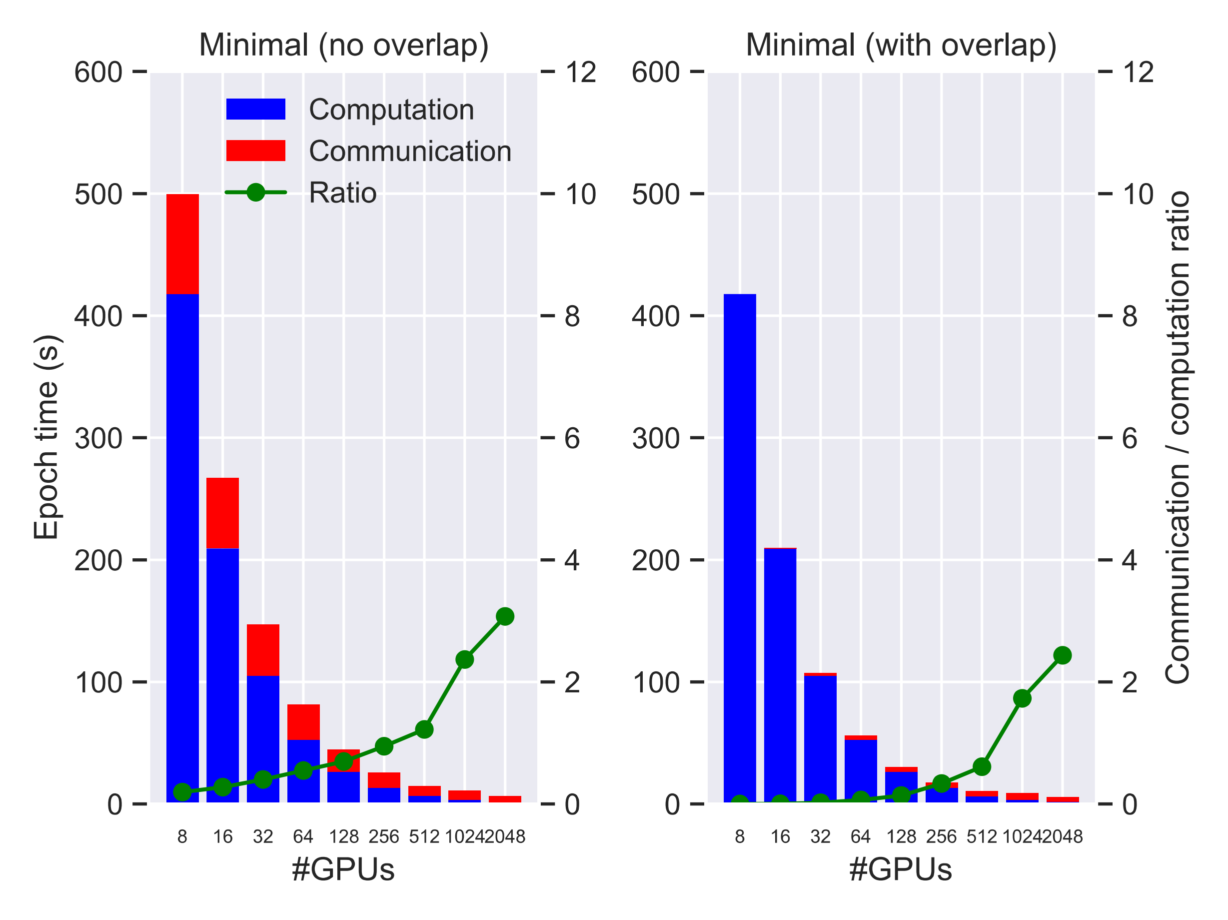
Strong scaling



Distributed data-parallelism

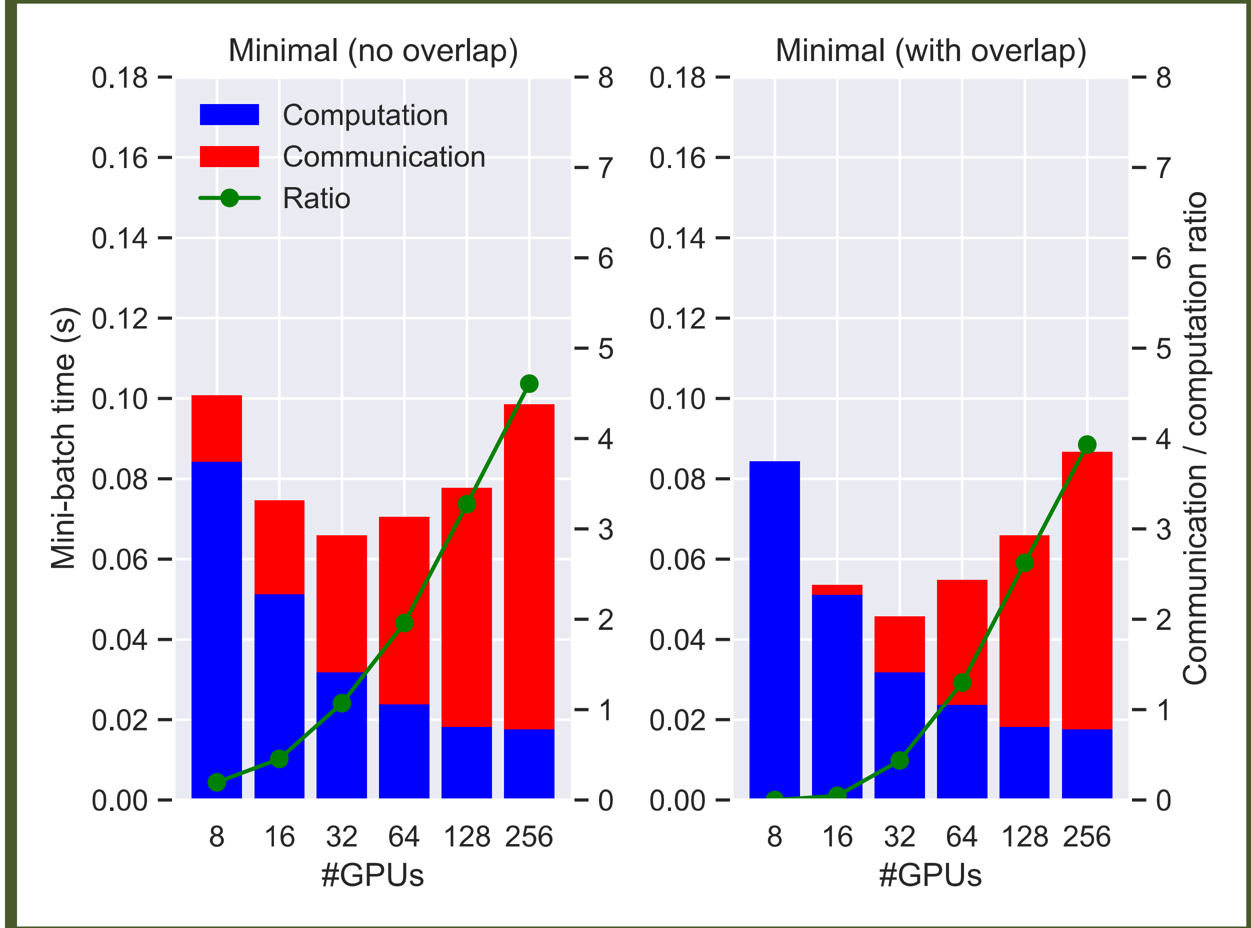
Strong scaling

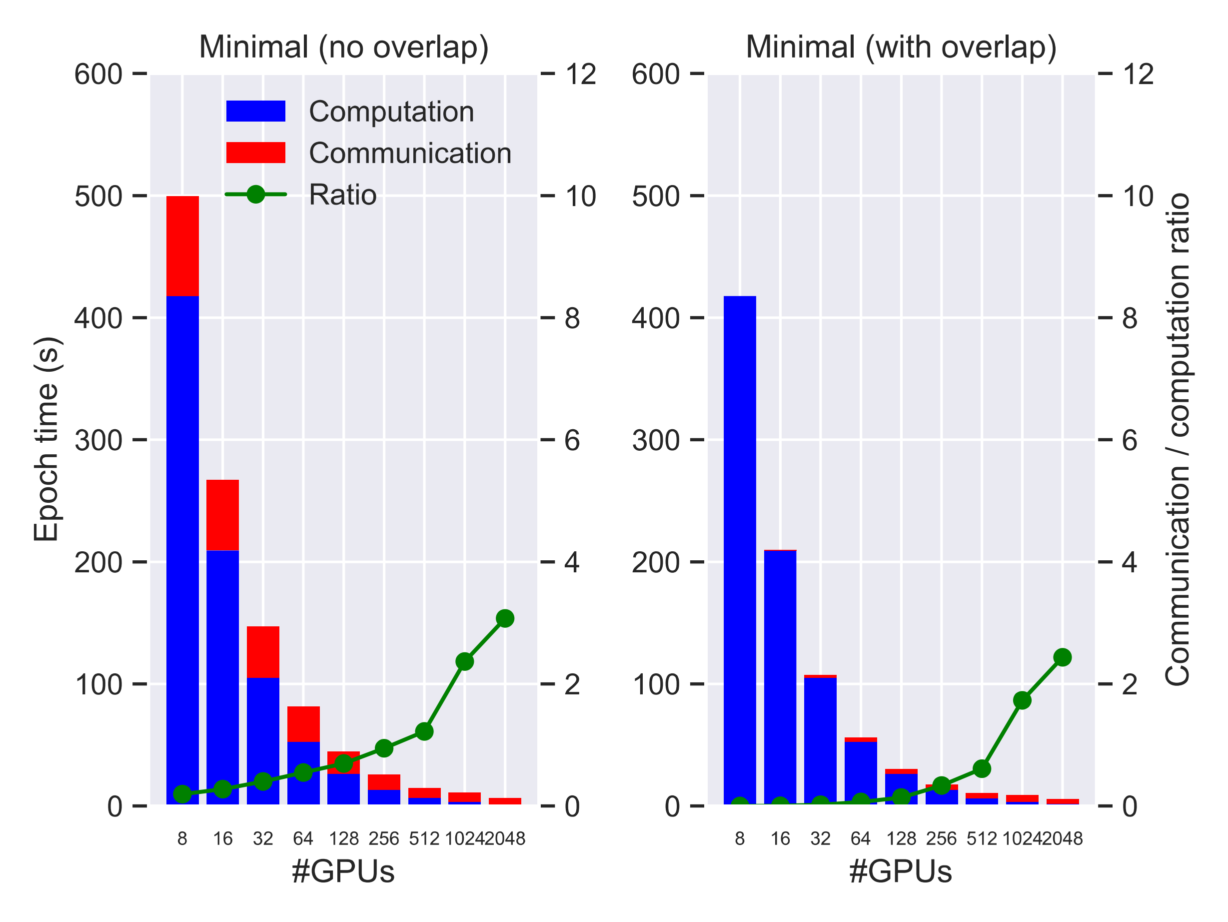




Distributed data-parallelism

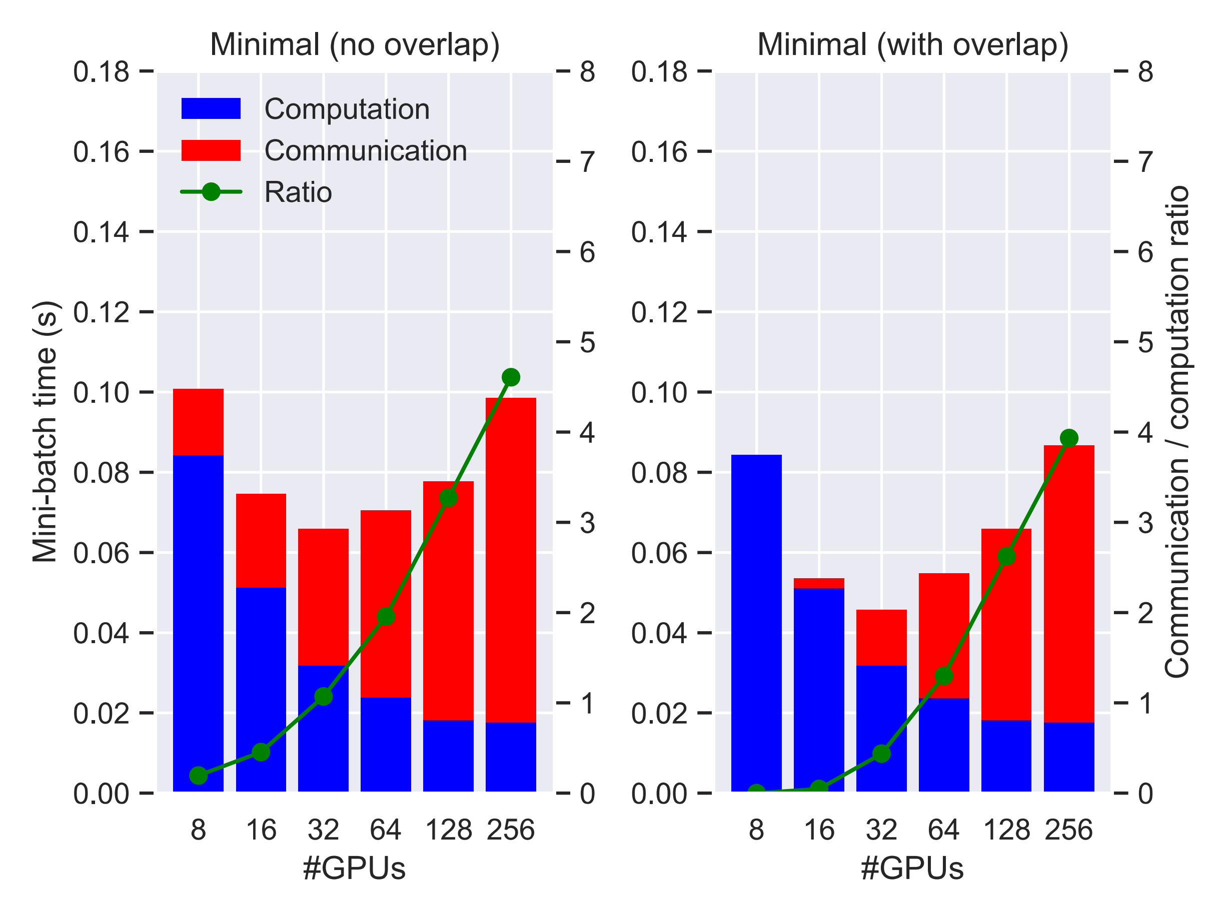
Strong scaling



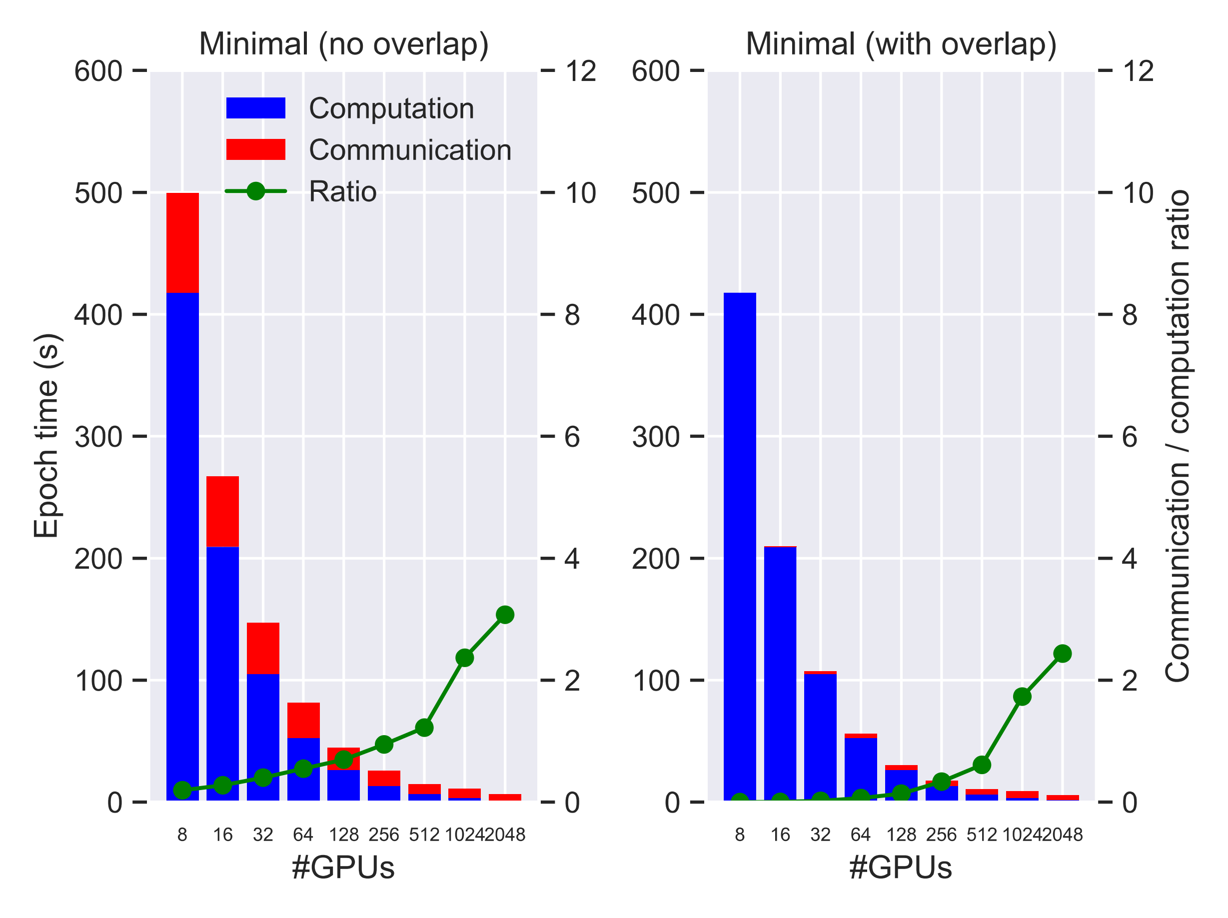


Distributed data-parallelism

Strong scaling

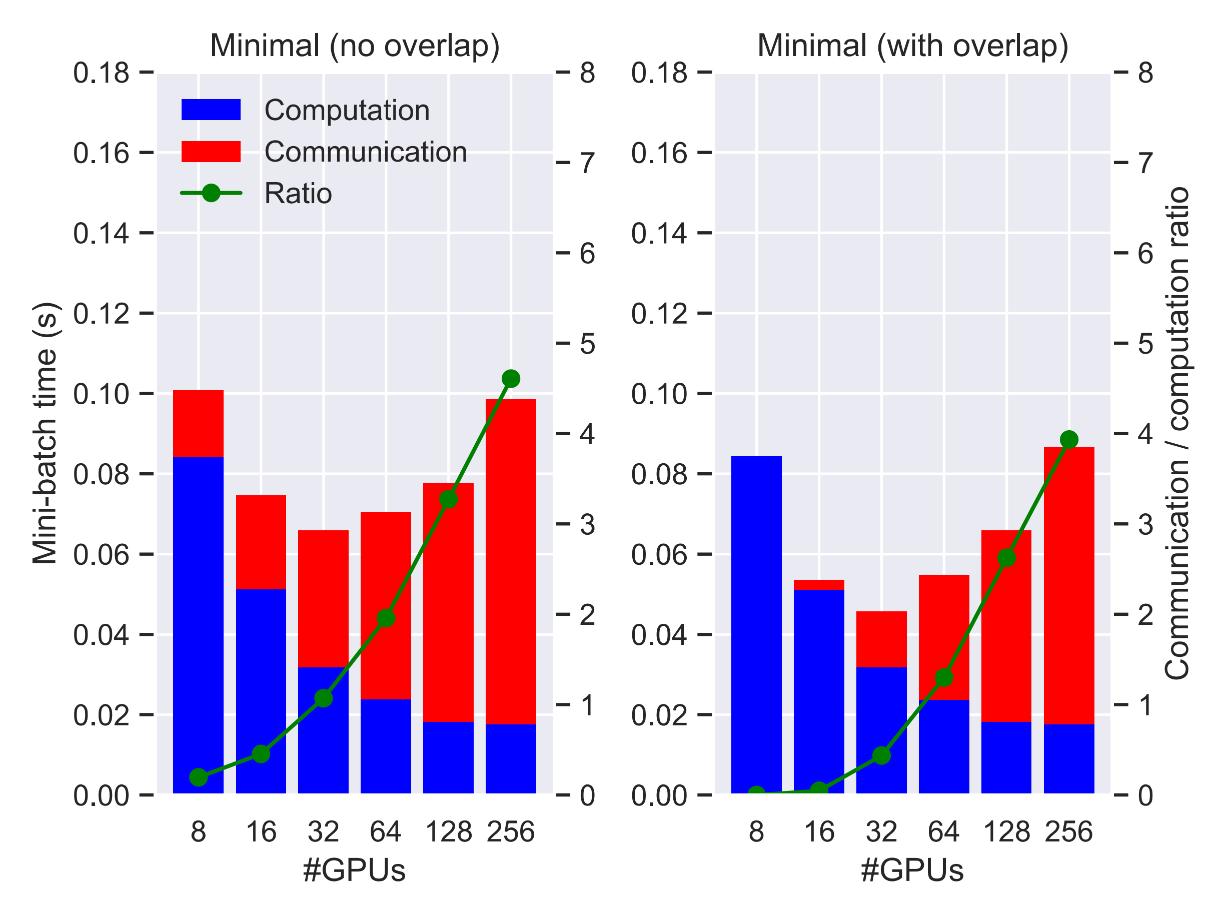


Weak scaling

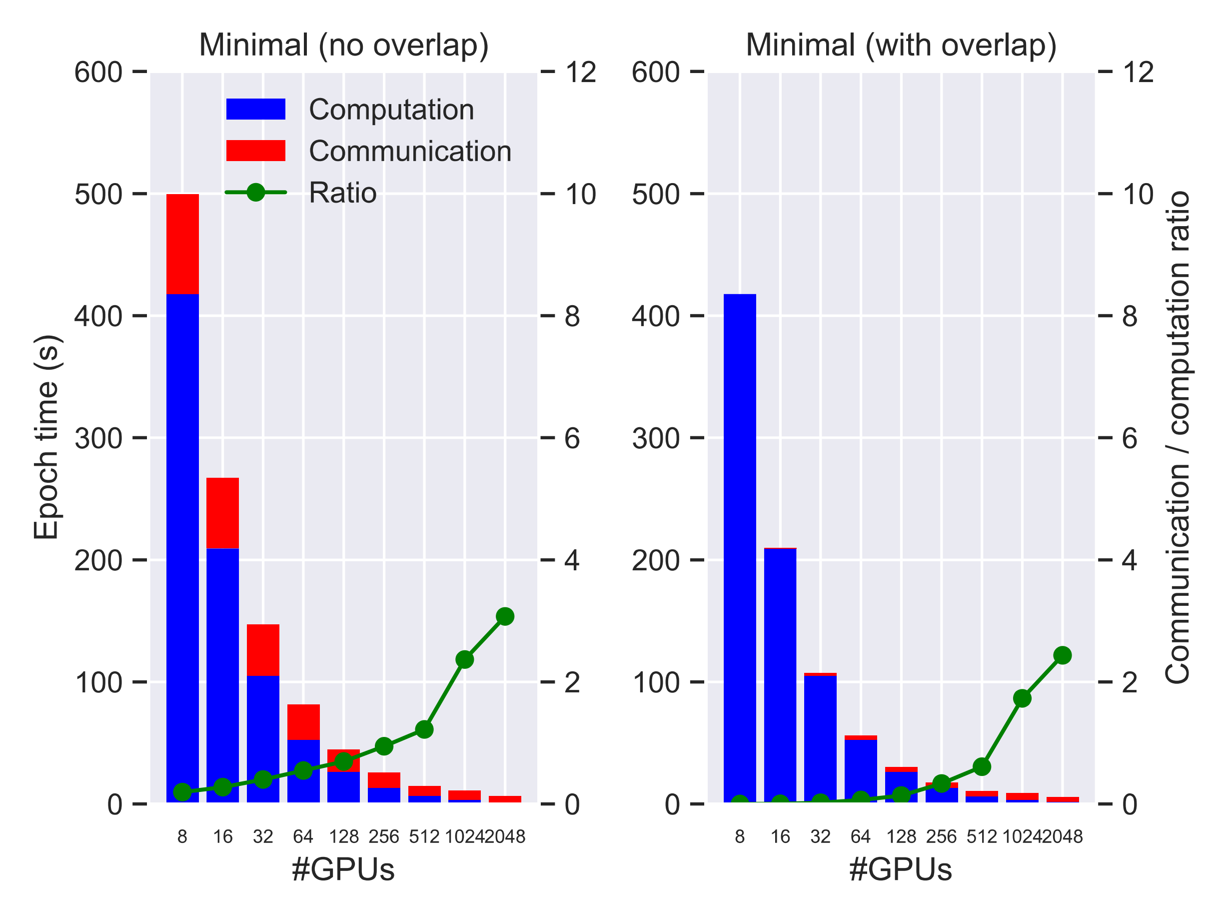


Distributed data-parallelism

Strong scaling



Weak scaling



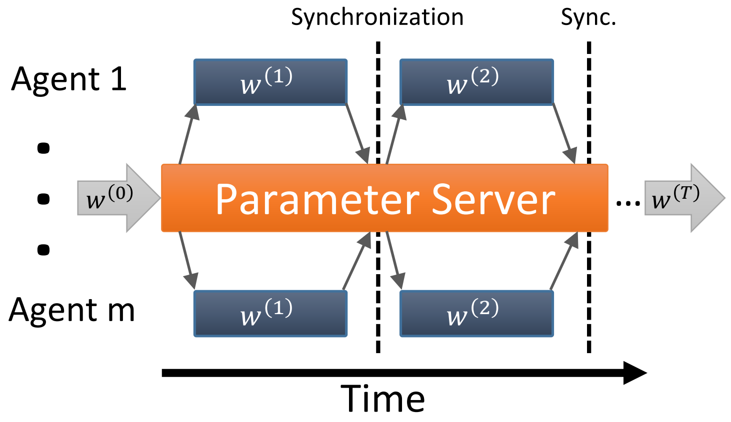
Parameter (and model) consistency - centralized

- **Parameter exchange frequency can be controlled, while still attaining convergence:**

[Dean et al. 2012; Niu et al. 2011]

Parameter (and model) consistency - centralized

- **Parameter exchange frequency can be controlled, while still attaining convergence:**



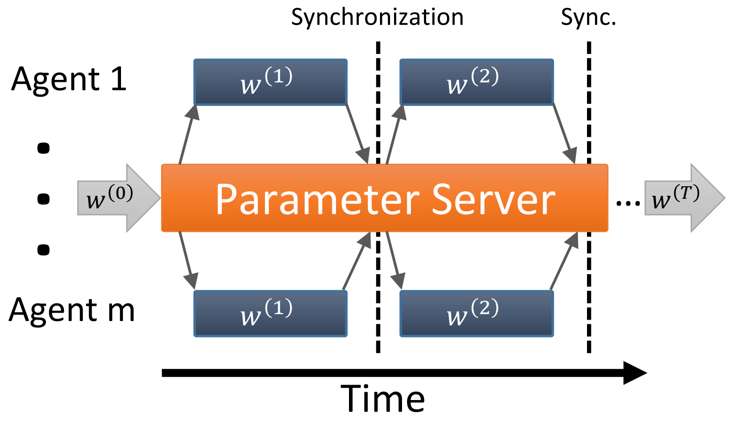
Synchronous

- **Trades off "statistical performance" for "hardware performance"**

[Dean et al. 2012; Niu et al. 2011]

Parameter (and model) consistency - centralized

- **Parameter exchange frequency can be controlled, while still attaining convergence:**



Synchronous | Stale Synchronous / Bounded Asynchronous | Asynchronous

- **Trades off "statistical performance" for "hardware performance"**

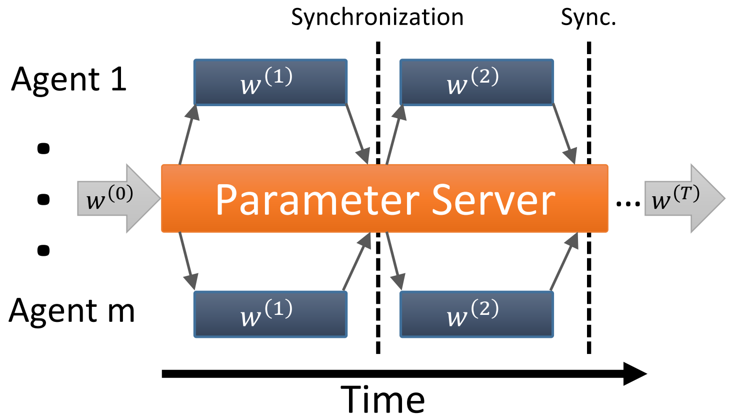
Synchronous SGD | Stale-Synchronous SGD | Asynchronous SGD (HOGWILD!) | Model Averaging (e.g., elastic) | Ensemble Learning

Consistent → **Inconsistent**

[Dean et al. 2012; Niu et al. 2011]

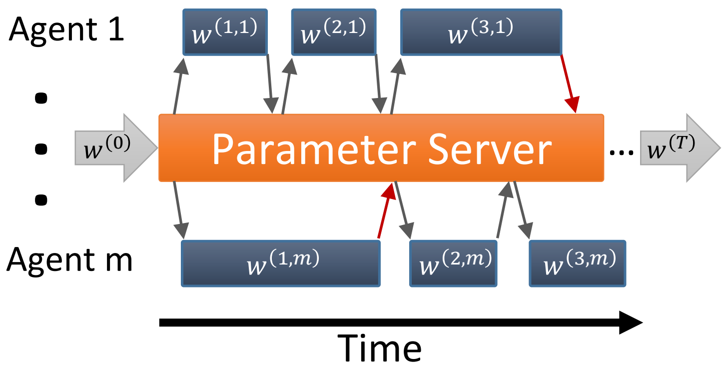
Parameter (and model) consistency - centralized

- Parameter exchange frequency can be controlled, while still attaining convergence:



Synchronous

Stale Synchronous / Bounded Asynchronous



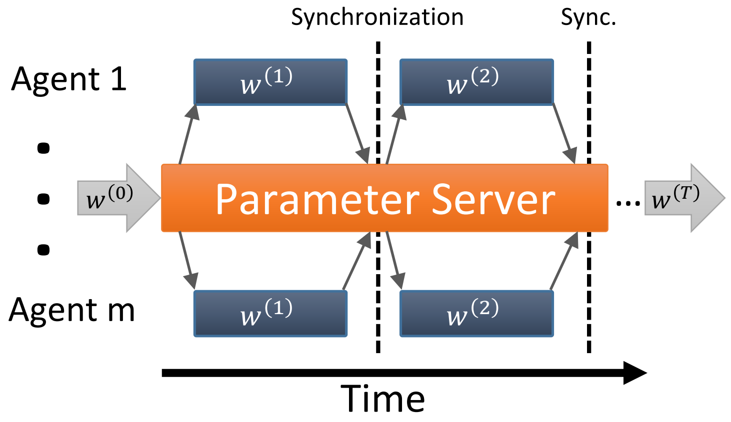
Asynchronous

- Trades off "statistical performance" for "hardware performance"

[Dean et al. 2012; Niu et al. 2011]

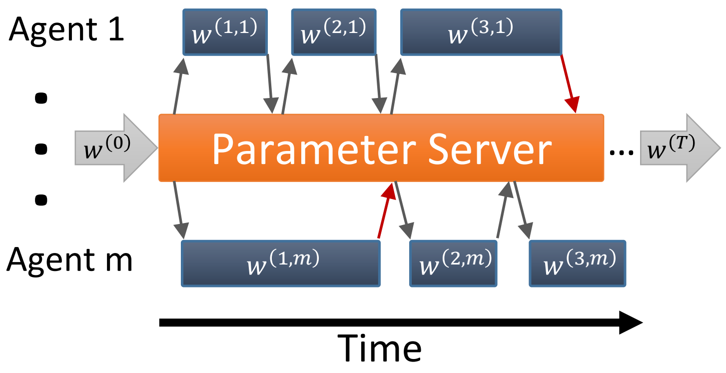
Parameter (and model) consistency - centralized

- **Parameter exchange frequency can be controlled, while still attaining convergence:**



Synchronous

Stale Synchronous / Bounded Asynchronous



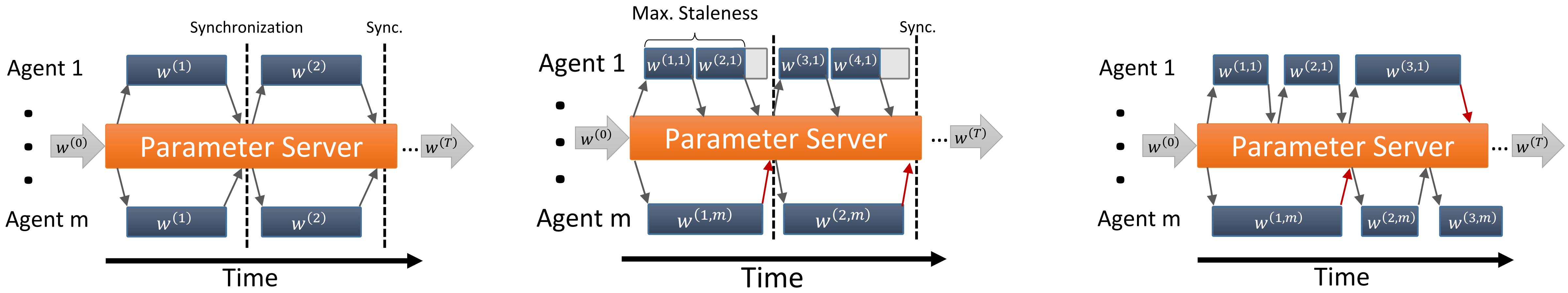
Asynchronous

- **Trades off "statistical performance" for "hardware performance"**

[Dean et al. 2012; Niu et al. 2011]

Parameter (and model) consistency - centralized

- **Parameter exchange frequency can be controlled, while still attaining convergence:**



Synchronous

Stale Synchronous / Bounded Asynchronous

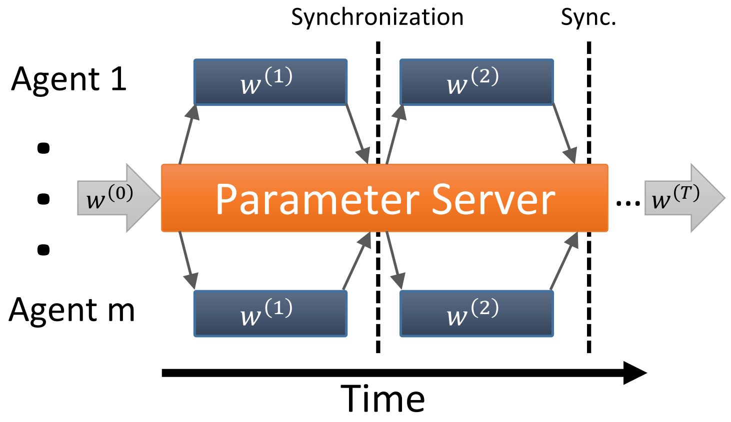
Asynchronous

- **Trades off "statistical performance" for "hardware performance"**

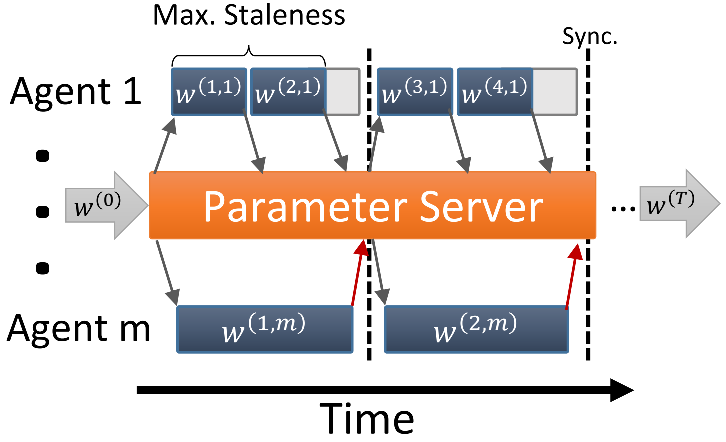
[Dean et al. 2012; Niu et al. 2011]

Parameter (and model) consistency - centralized

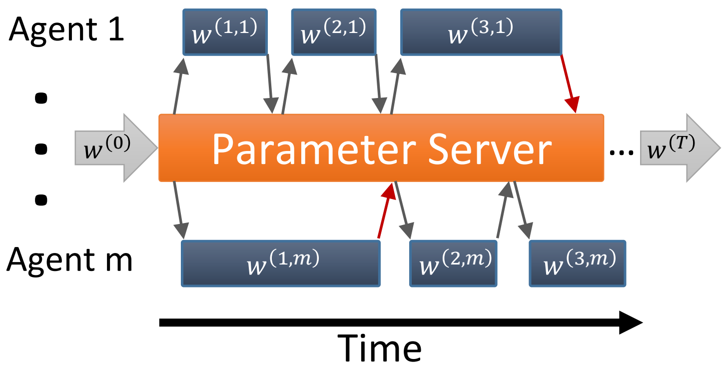
- **Parameter exchange frequency can be controlled, while still attaining convergence:**



Synchronous



Stale Synchronous / Bounded Asynchronous



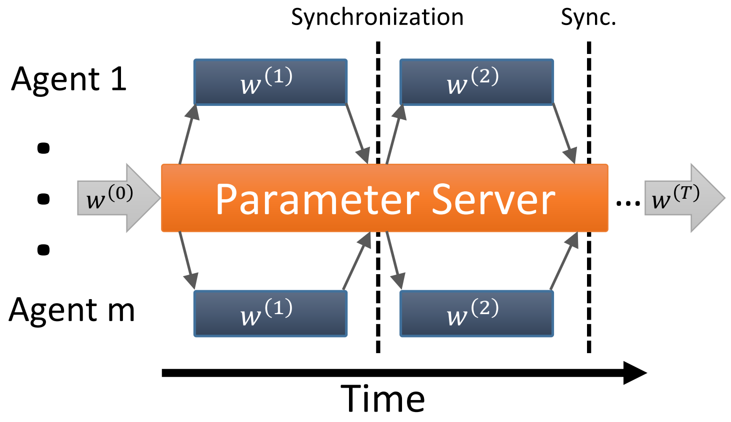
Asynchronous

- **Trades off "statistical performance" for "hardware performance"**

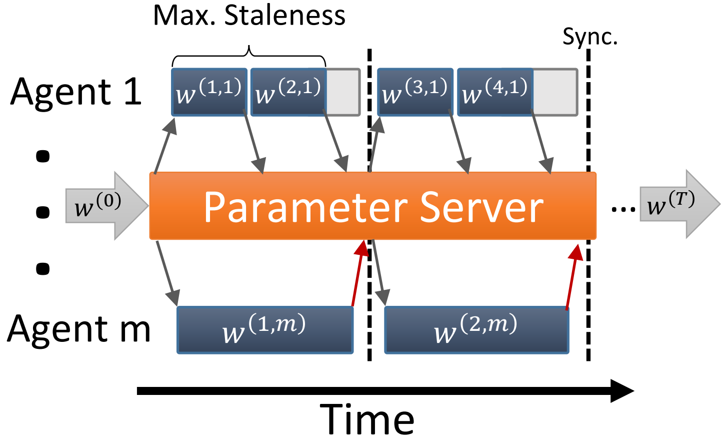
[Dean et al. 2012; Niu et al. 2011]

Parameter (and model) consistency - centralized

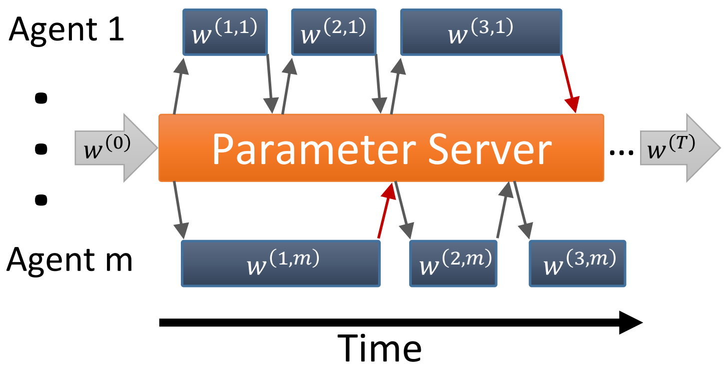
- **Parameter exchange frequency can be controlled, while still attaining convergence:**



Synchronous



Stale Synchronous / Bounded Asynchronous



Asynchronous

- **Trades off "statistical performance" for "hardware performance"**

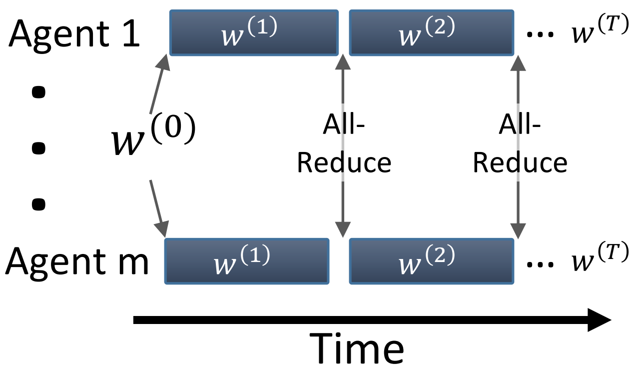
[Dean et al. 2012; Niu et al. 2011]

Parameter (and model) consistency - decentralized

- **Parameter exchange frequency can be controlled, while still attaining convergence:**

Parameter (and model) consistency - decentralized

- **Parameter exchange frequency can be controlled, while still attaining convergence:**

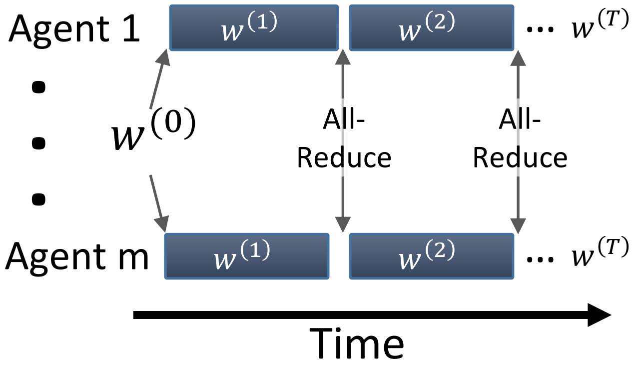


Synchronous

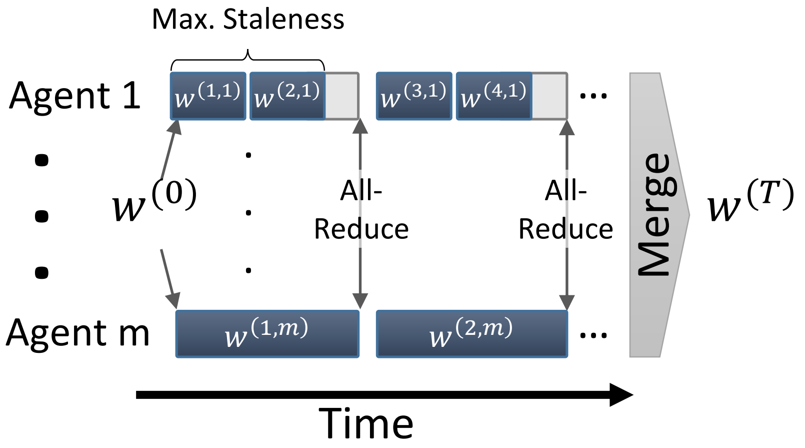
- **May also consider limited/slower distribution – gossip [Jin et al. 2016]**

Parameter (and model) consistency - decentralized

- **Parameter exchange frequency can be controlled, while still attaining convergence:**



Synchronous

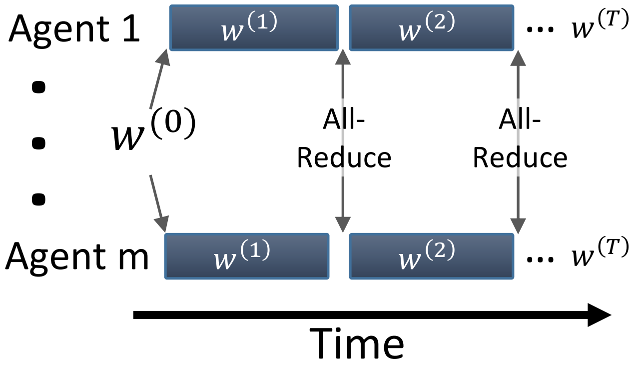


Stale Synchronous / Bounded Asynchronous

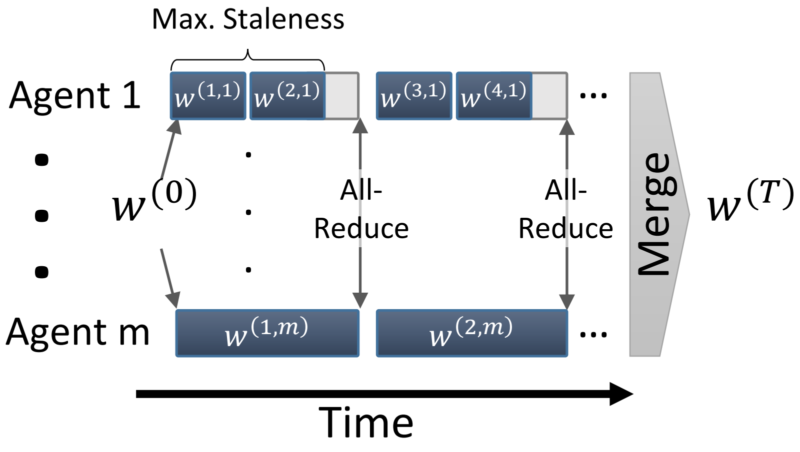
- **May also consider limited/slower distribution – gossip [Jin et al. 2016]**

Parameter (and model) consistency - decentralized

- **Parameter exchange frequency can be controlled, while still attaining convergence:**



Synchronous



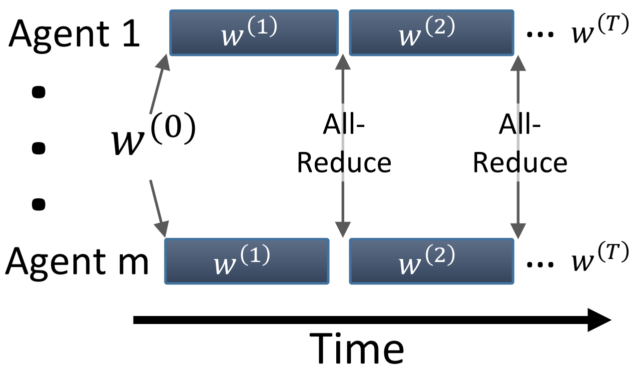
Stale Synchronous / Bounded Asynchronous

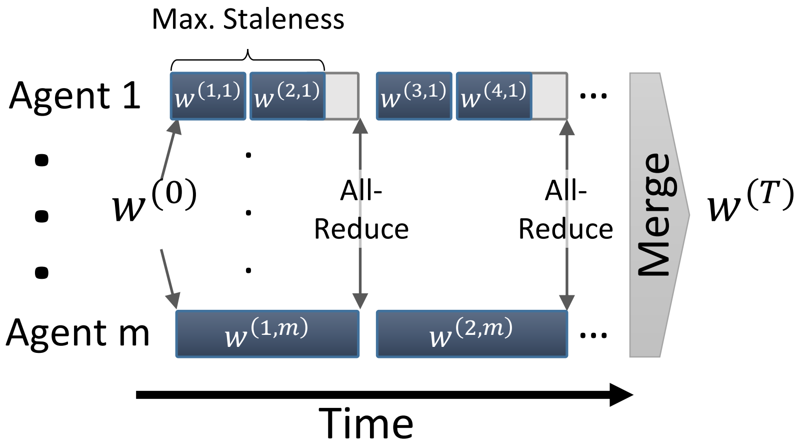
Asynchronous

- **May also consider limited/slower distribution – gossip [Jin et al. 2016]**

Parameter (and model) consistency - decentralized

- **Parameter exchange frequency can be controlled, while still attaining convergence:**





Synchronous

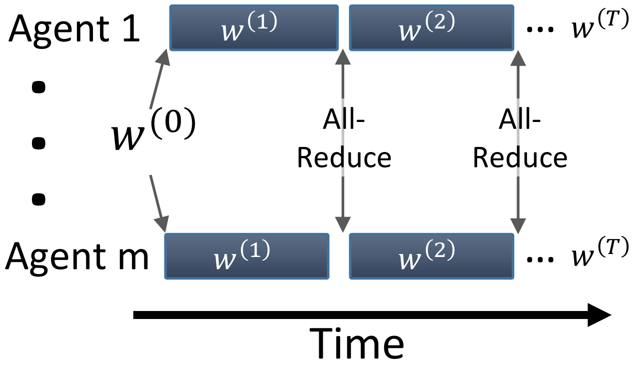
Stale Synchronous / Bounded Asynchronous

Asynchronous

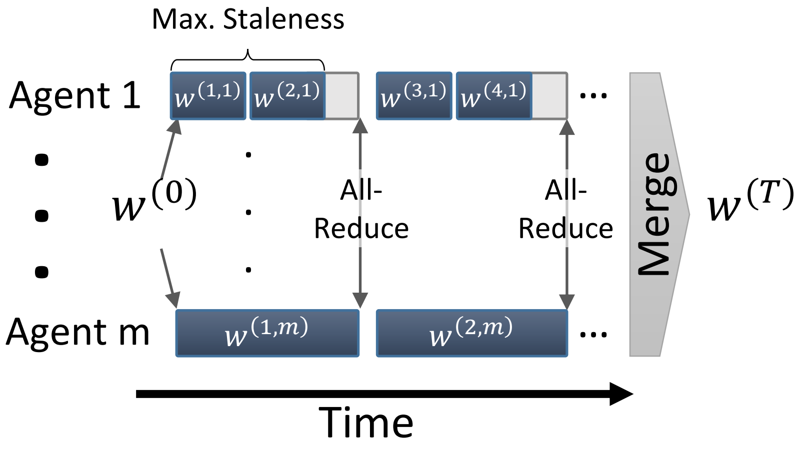
- **May also consider limited/slower distribution – gossip [Jin et al. 2016]**

Parameter (and model) consistency - decentralized

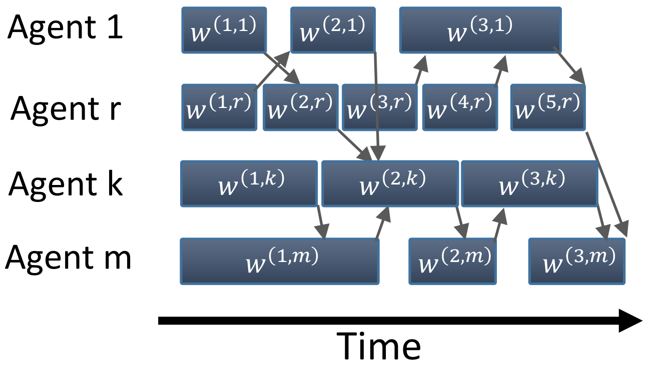
- **Parameter exchange frequency can be controlled, while still attaining convergence:**



Synchronous



Stale Synchronous / Bounded Asynchronous

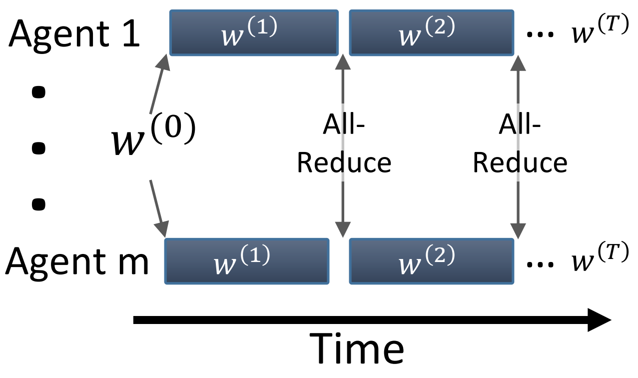


Asynchronous

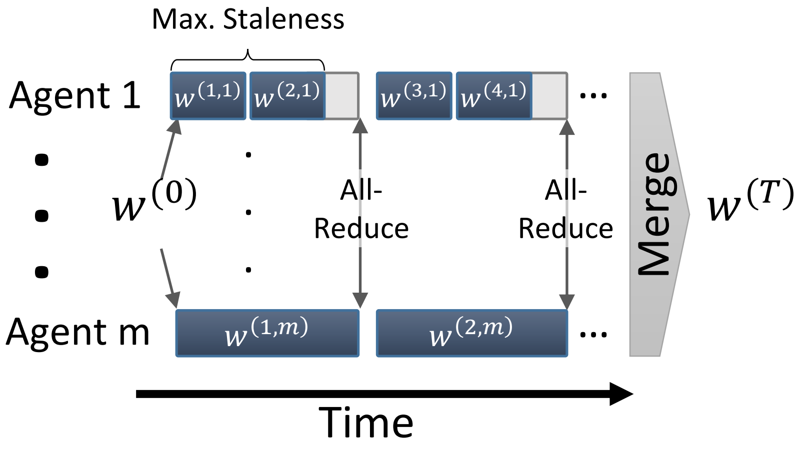
- **May also consider limited/slower distribution – gossip [Jin et al. 2016]**

Parameter (and model) consistency - decentralized

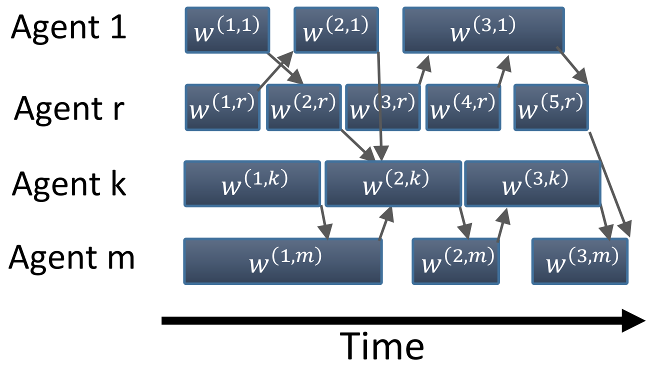
- **Parameter exchange frequency can be controlled, while still attaining convergence:**



Synchronous



Stale Synchronous / Bounded Asynchronous

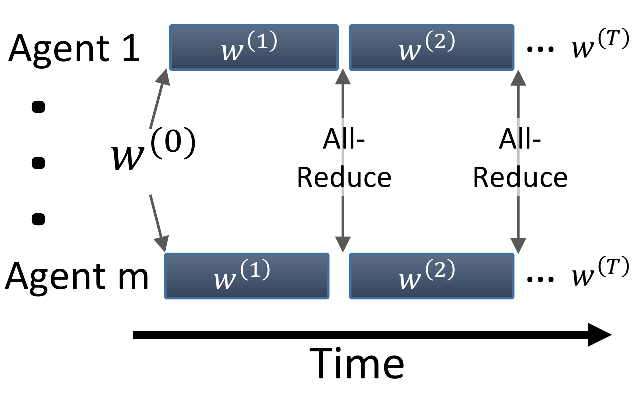


Asynchronous

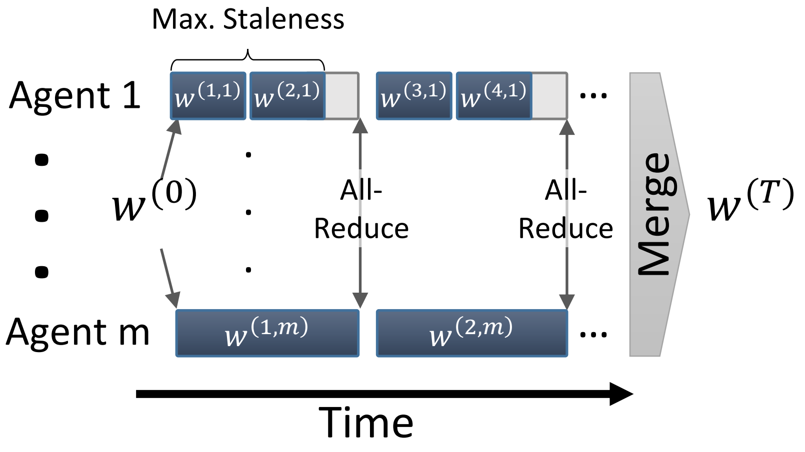
- **May also consider limited/slower distribution – gossip [Jin et al. 2016]**

Parameter (and model) consistency - decentralized

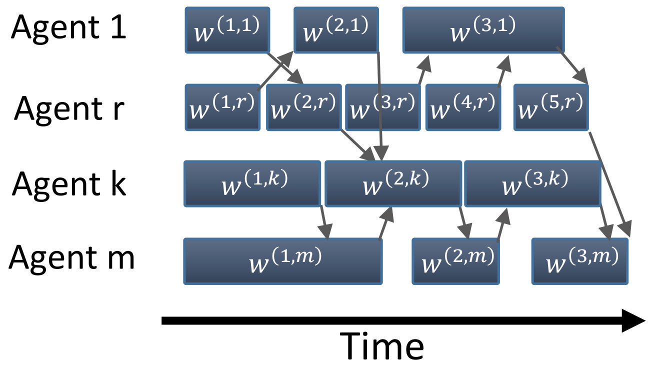
- **Parameter exchange frequency can be controlled, while still attaining convergence:**



Synchronous



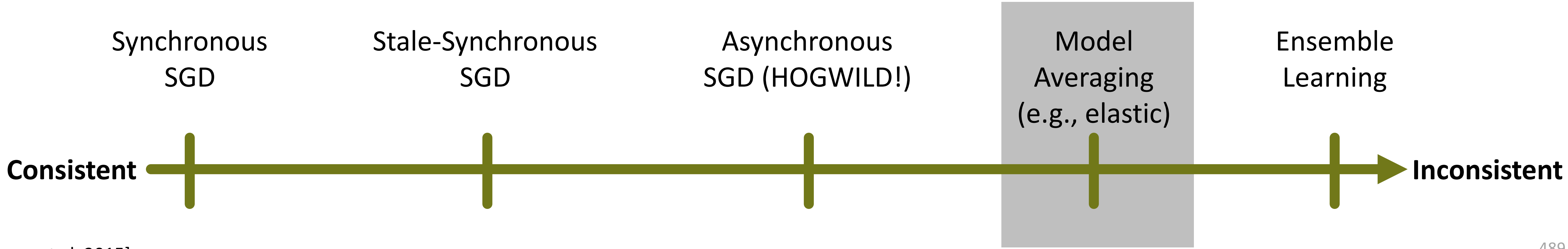
Stale Synchronous / Bounded Asynchronous



Asynchronous

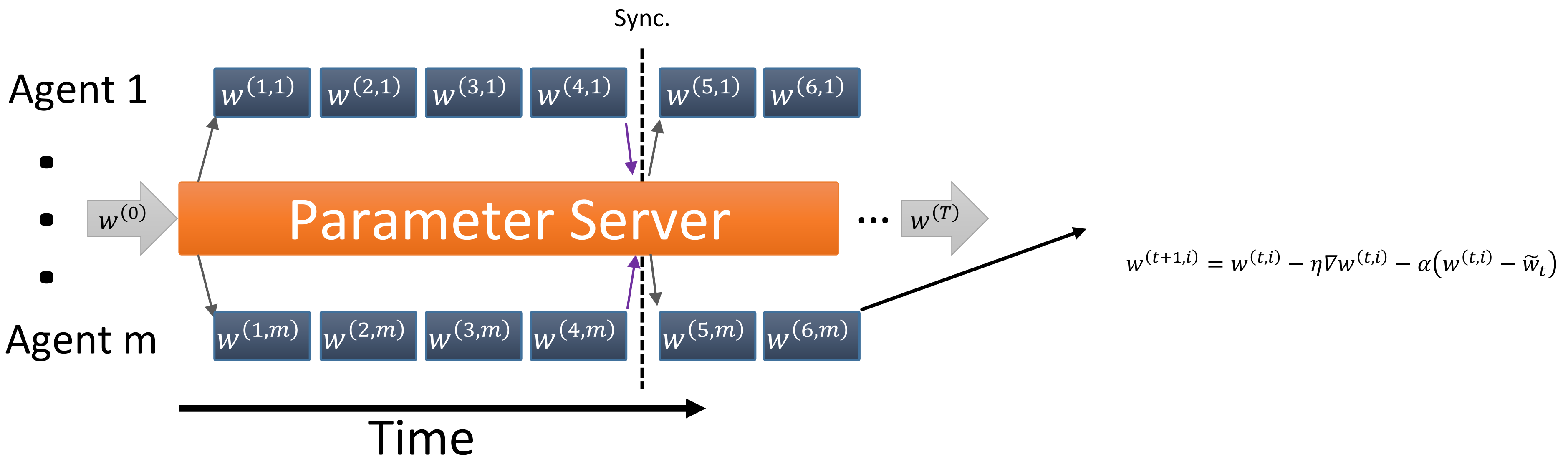
- **May also consider limited/slower distribution – gossip [Jin et al. 2016]**

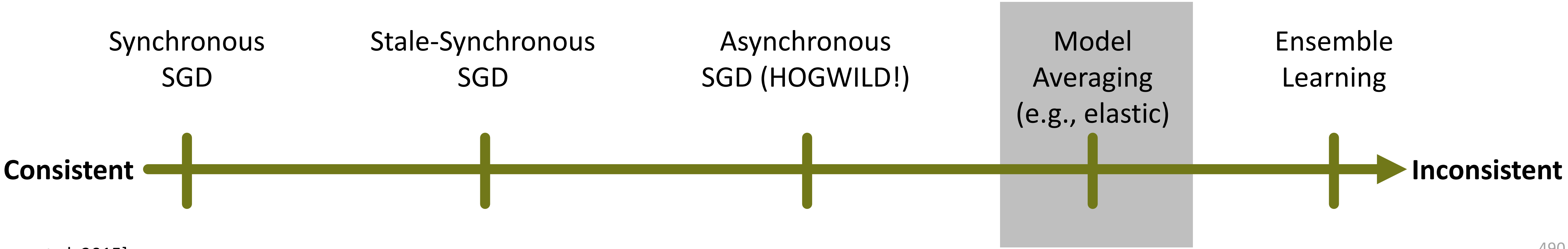
Parameter consistency in deep learning



[Zhang et al. 2015]

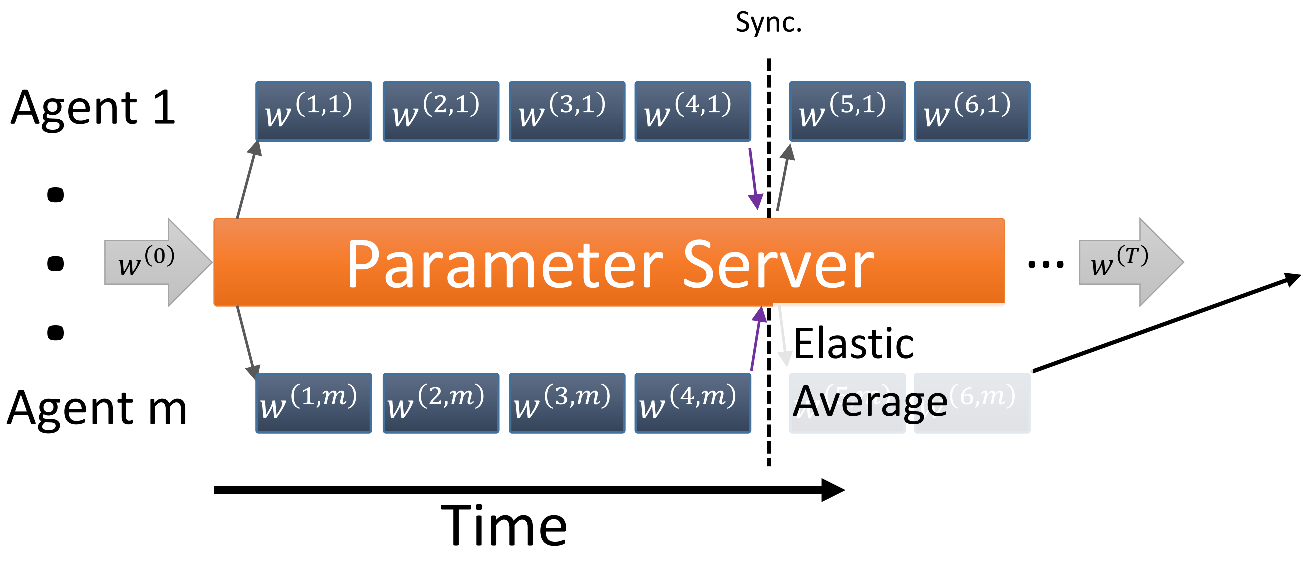
Parameter consistency in deep learning





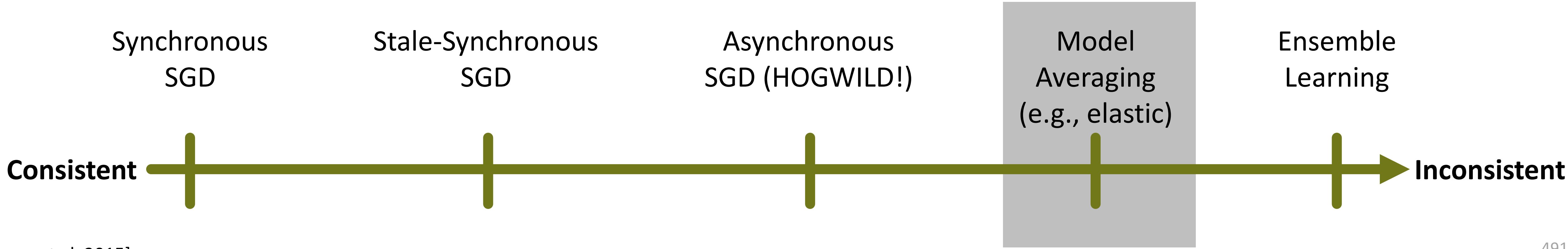
[Zhang et al. 2015]

Parameter consistency in deep learning



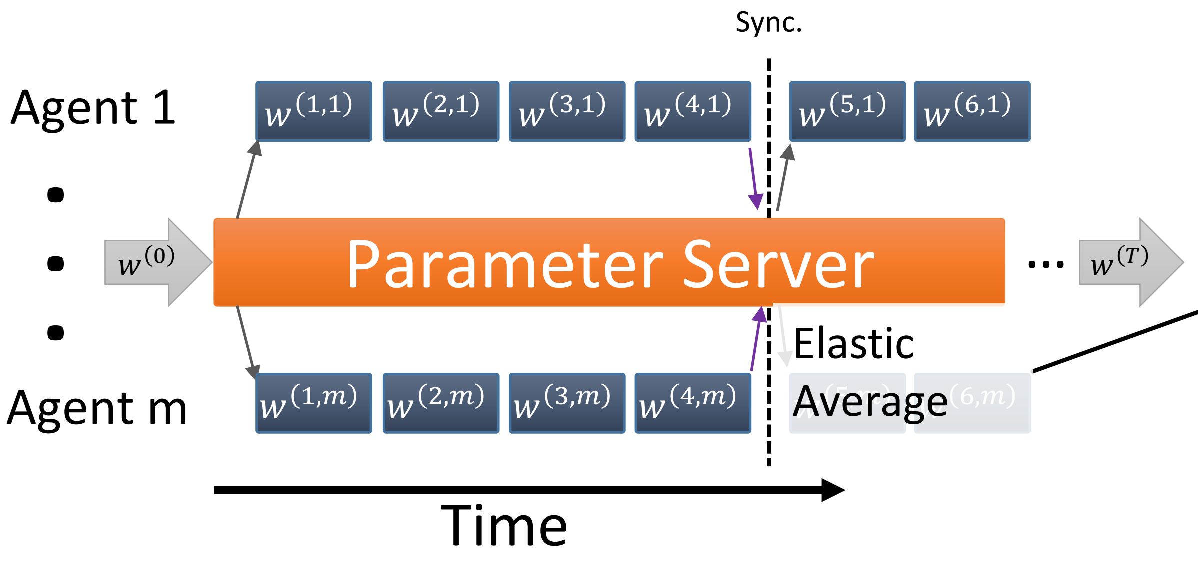
$$w^{(t+1,i)} = w^{(t,i)} - \eta \nabla w^{(t,i)} - \alpha\left(w^{(t,i)} - \widetilde{w}_t\right)$$

$$\widetilde{w}_{t+1} = (1-\beta)\widetilde{w}_t + \frac{\beta}{m}\sum_{i=1}^{m} w^{(t,i)}$$



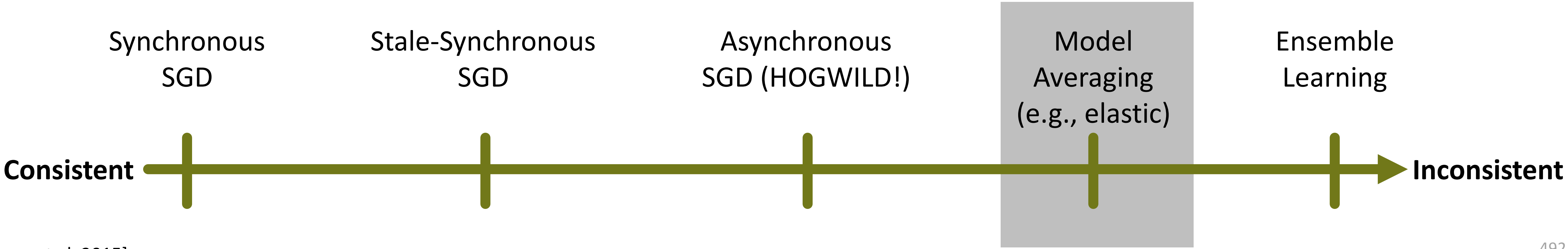
[Zhang et al. 2015]

Parameter consistency in deep learning



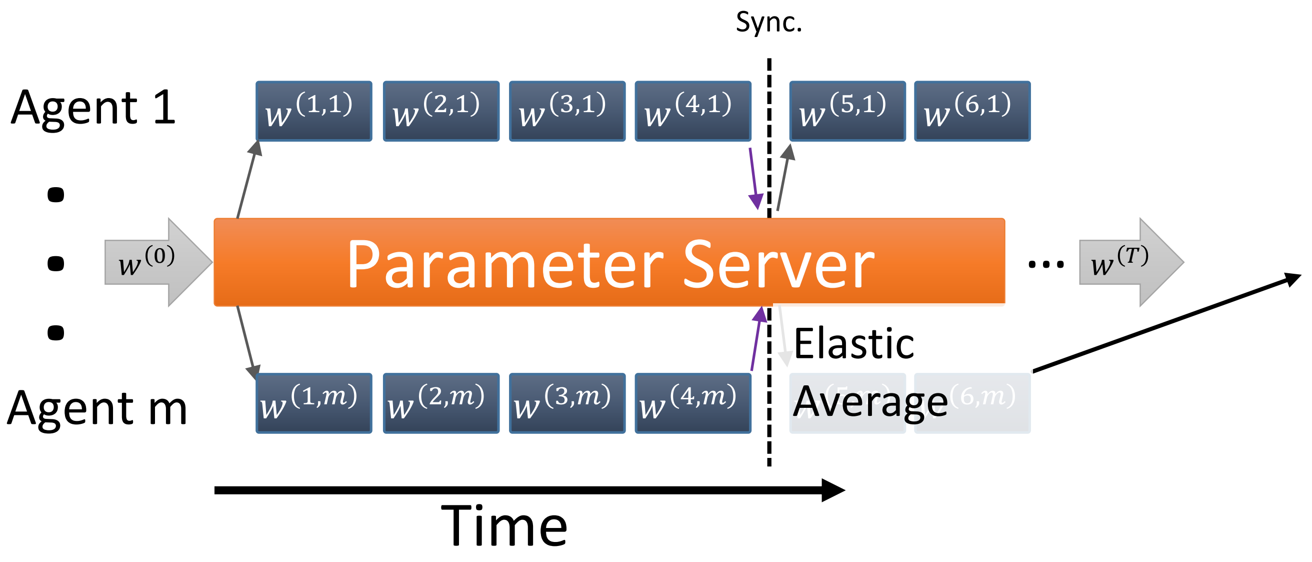
$$w^{(t+1,i)} = w^{(t,i)} - \eta\nabla w^{(t,i)} - \alpha\left(w^{(t,i)} - \widetilde{w}_t\right)$$

$$\widetilde{w}_{t+1} = (1-\beta)\widetilde{w}_t + \frac{\beta}{m}\sum_{i=1}^{m} w^{(t,i)}$$



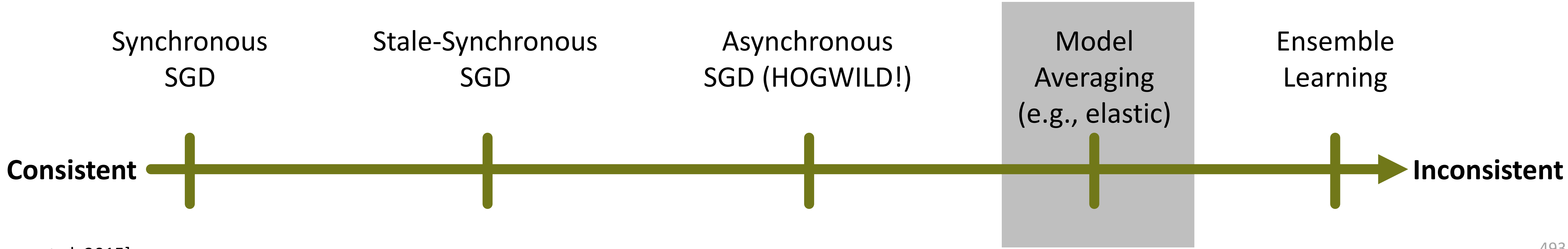
[Zhang et al. 2015]

Parameter consistency in deep learning



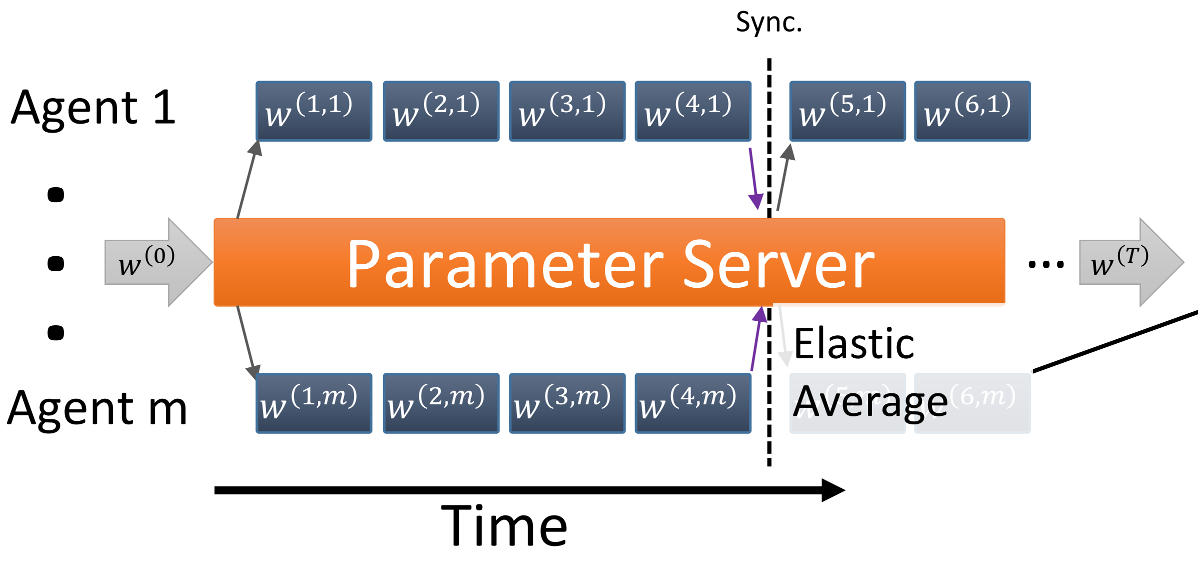
$$w^{(t+1,i)} = w^{(t,i)} - \eta\nabla w^{(t,i)} - \alpha\left(w^{(t,i)} - \widetilde{w}_t\right)$$

$$\widetilde{w}_{t+1} = (1-\beta)\widetilde{w}_t + \frac{\beta}{m}\sum_{i=1}^{m} w^{(t,i)}$$



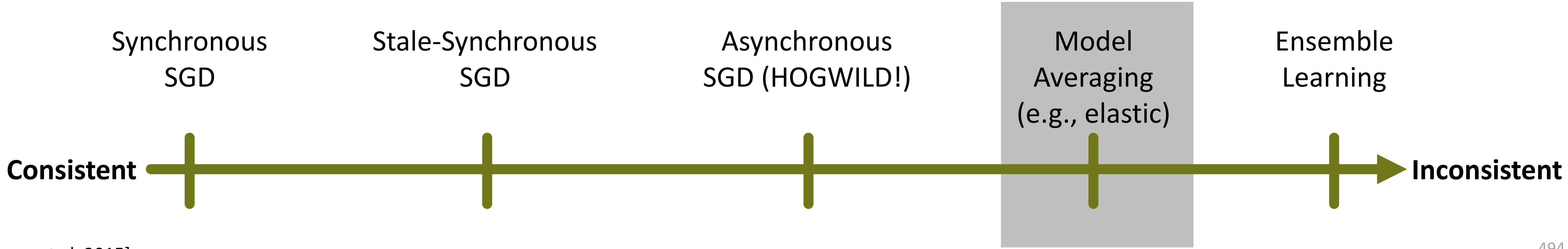
[Zhang et al. 2015]

Parameter consistency in deep learning



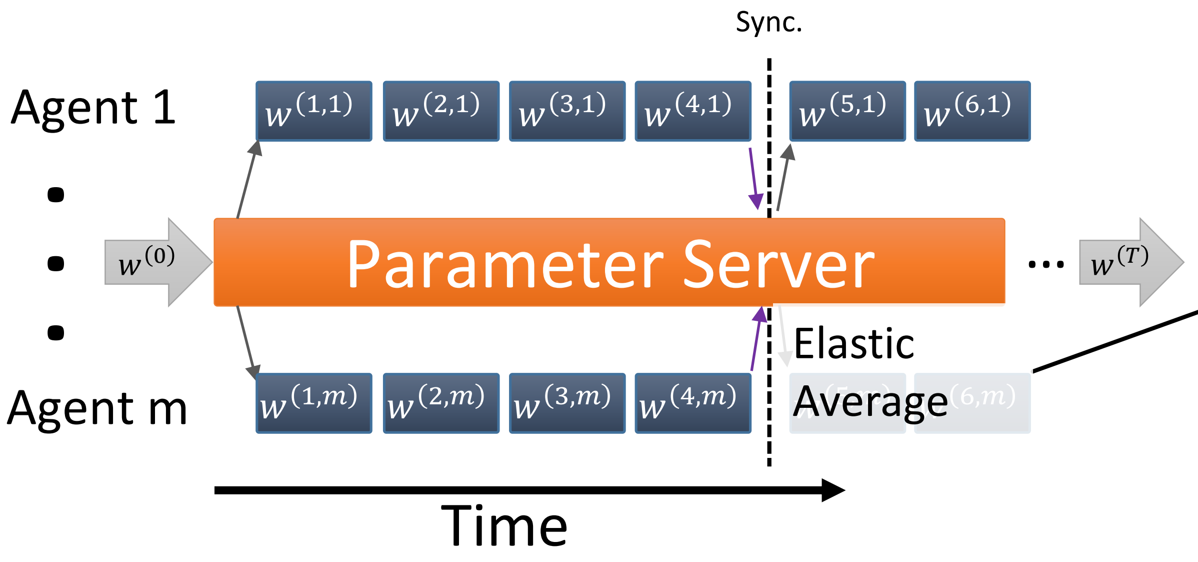
$$w^{(t+1,i)} = w^{(t,i)} - \eta \nabla w^{(t,i)} - \alpha\left(w^{(t,i)} - \widetilde{w}_t\right)$$

$$\widetilde{w}_{t+1} = (1-\beta)\widetilde{w}_t + \frac{\beta}{m}\sum_{i=1}^{m} w^{(t,i)}$$



[Zhang et al. 2015]

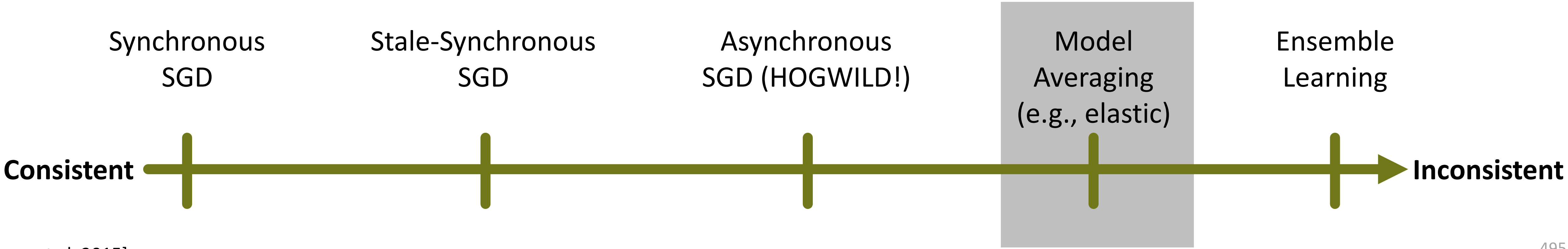
Parameter consistency in deep learning



Using physical forces between different versions of w:

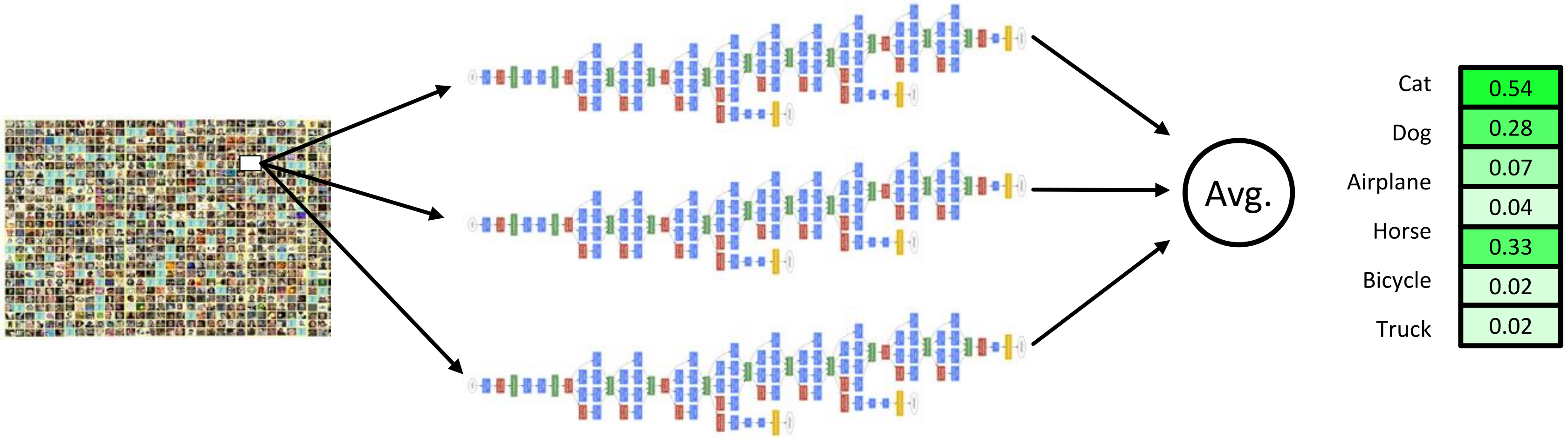
$$w^{(t+1,i)} = w^{(t,i)} - \eta \nabla w^{(t,i)} - \alpha\left(w^{(t,i)} - \widetilde{w}_t\right)$$

$$\widetilde{w}_{t+1} = (1-\beta)\widetilde{w}_t + \frac{\beta}{m}\sum_{i=1}^{m} w^{(t,i)}$$



[Zhang et al. 2015]

Parameter consistency in deep learning



Consistent — Synchronous SGD — Stale-Synchronous SGD — Asynchronous SGD (HOGWILD!) — Model Averaging (e.g., elastic) — **Ensemble Learning** — Inconsistent

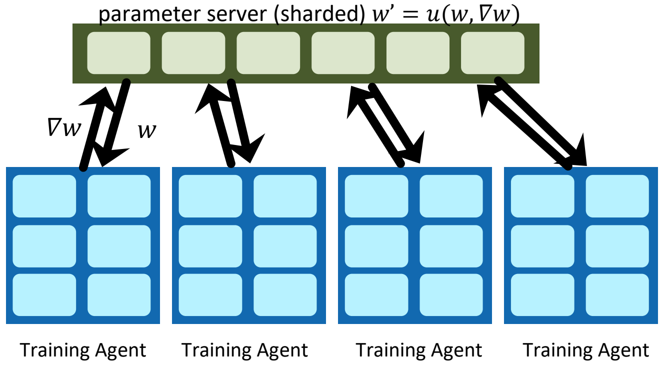
[Dietterich 2000]

Communication optimization

Communication optimization

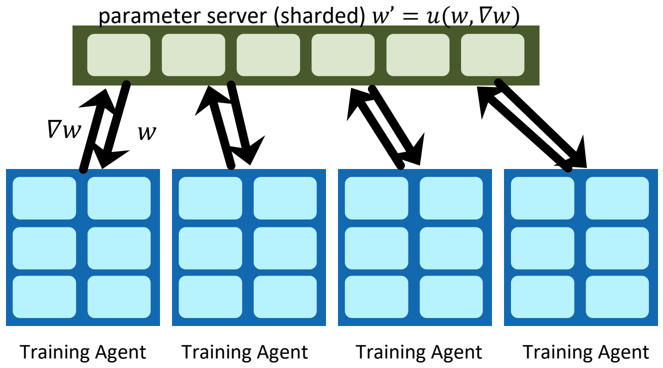
- **Lossy compression: trade off latency (local compute) for bandwidth**
- **Sufficient factor broadcasting**
- **Quantization:**
  - 16-bit (IEEE FP16, bfloat16) becoming standard
  - QSGD (stochastic rounding) [Alistarh et al. 2016]
  - 1-bit SGD [Seide et al. 2014; Dryden et al. 2016]
  - Error feedback is important!
- **Sparsification:**
  - Top-k SGD [Renggli et al. 2019]
  - Skip small weight updates

Communication optimization

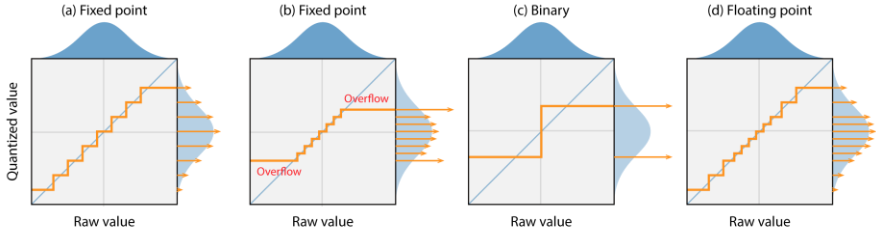


- **Lossy compression: trade off latency (local compute) for bandwidth**
- **Sufficient factor broadcasting**
- **Quantization:**
  - 16-bit (IEEE FP16, bfloat16) becoming standard
  - QSGD (stochastic rounding) [Alistarh et al. 2016]
  - 1-bit SGD [Seide et al. 2014; Dryden et al. 2016]
  - Error feedback is important!
- **Sparsification:**
  - Top-k SGD [Renggli et al. 2019]
  - Skip small weight updates

Communication optimization

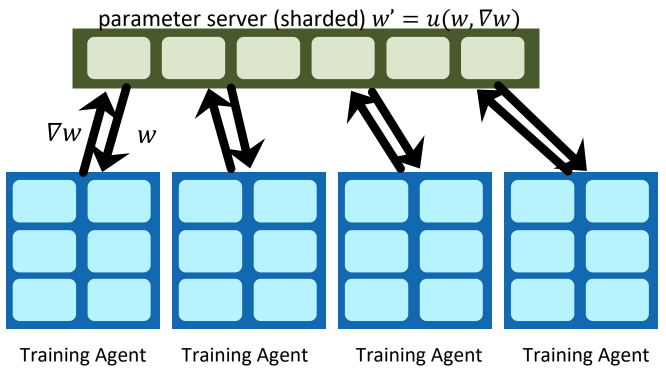


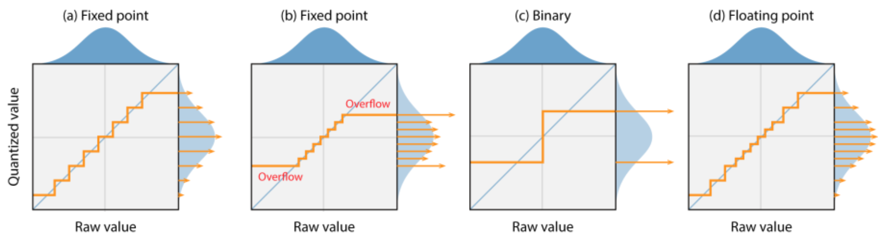
- **Lossy compression: trade off latency (local compute) for bandwidth**
- **Sufficient factor broadcasting**
- **Quantization:**
  - 16-bit (IEEE FP16, bfloat16) becoming standard
  - QSGD (stochastic rounding) [Alistarh et al. 2016]
  - 1-bit SGD [Seide et al. 2014; Dryden et al. 2016]
  - Error feedback is important!
- **Sparsification:**
  - Top-k SGD [Renggli et al. 2019]
  - Skip small weight updates

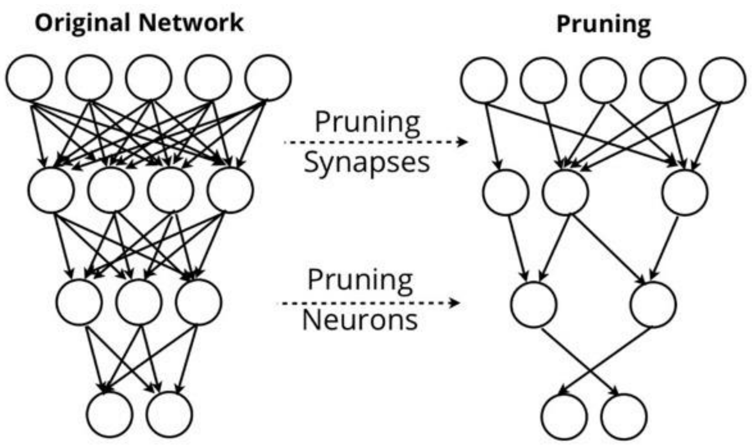


Communication optimization

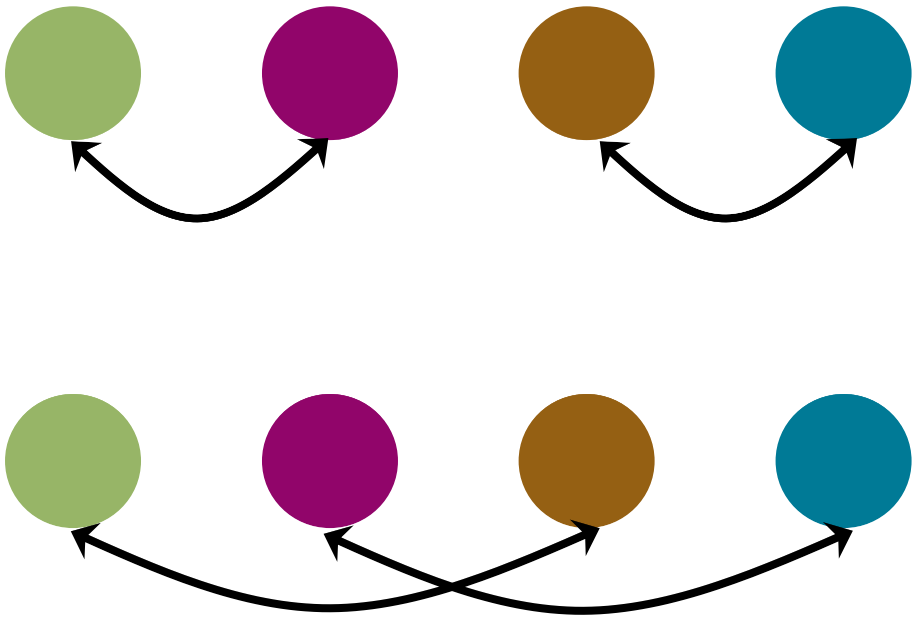
- **Lossy compression: trade off latency (local compute) for bandwidth**
- **Sufficient factor broadcasting**
- **Quantization:**
  - 16-bit (IEEE FP16, bfloat16) becoming standard
  - QSGD (stochastic rounding) [Alistarh et al. 2016]
  - 1-bit SGD [Seide et al. 2014; Dryden et al. 2016]
  - Error feedback is important!
- **Sparsification:**
  - Top-k SGD [Renggli et al. 2019]
  - Skip small weight updates





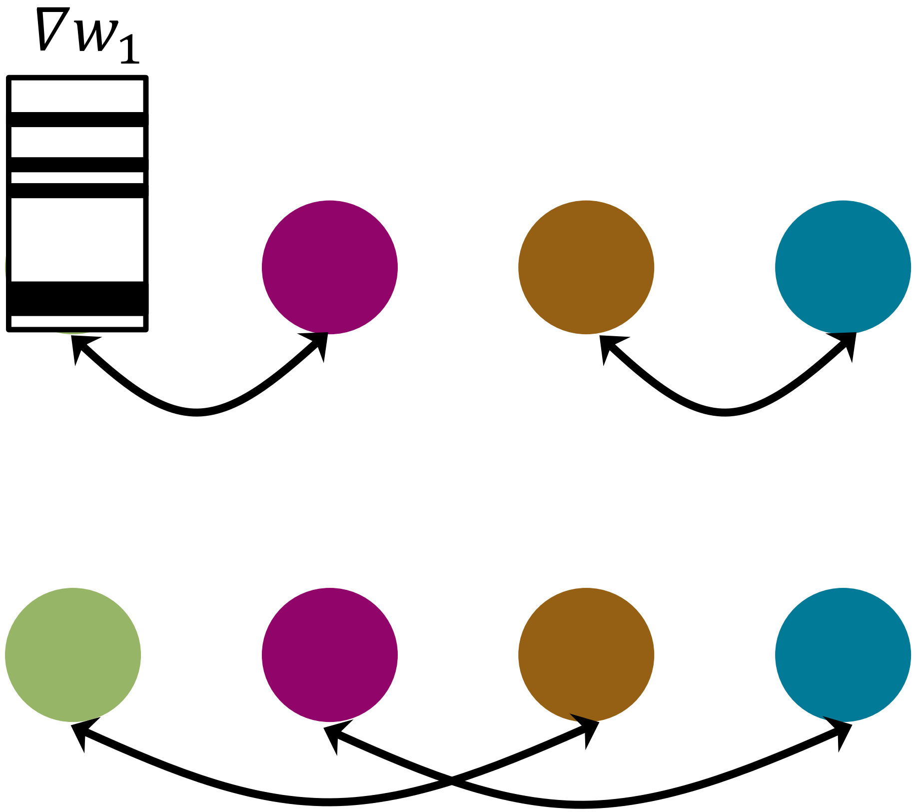


SparCML – Quantized sparse allreduce



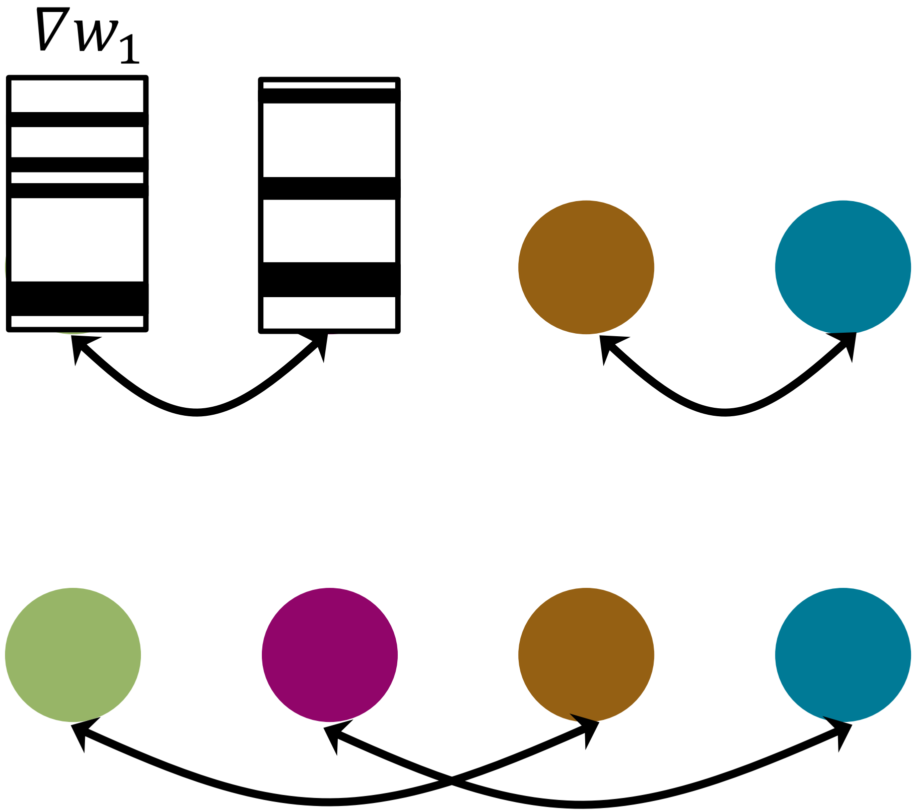
[Renggli et al. 2019]

SparCML – Quantized sparse allreduce



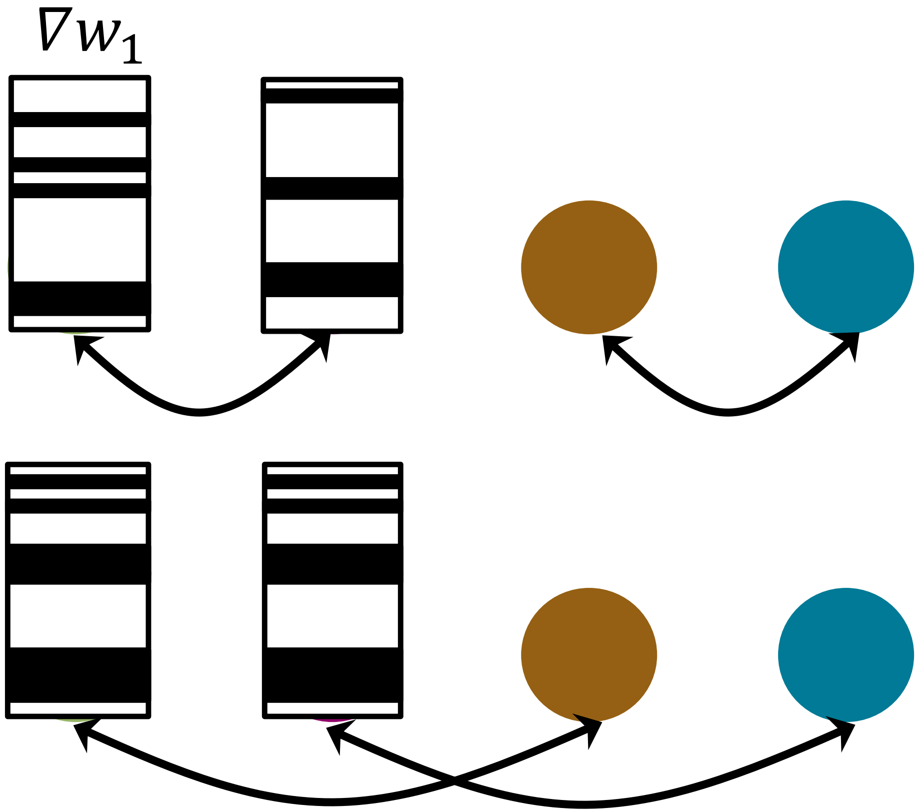
[Renggli et al. 2019]

SparCML – Quantized sparse allreduce



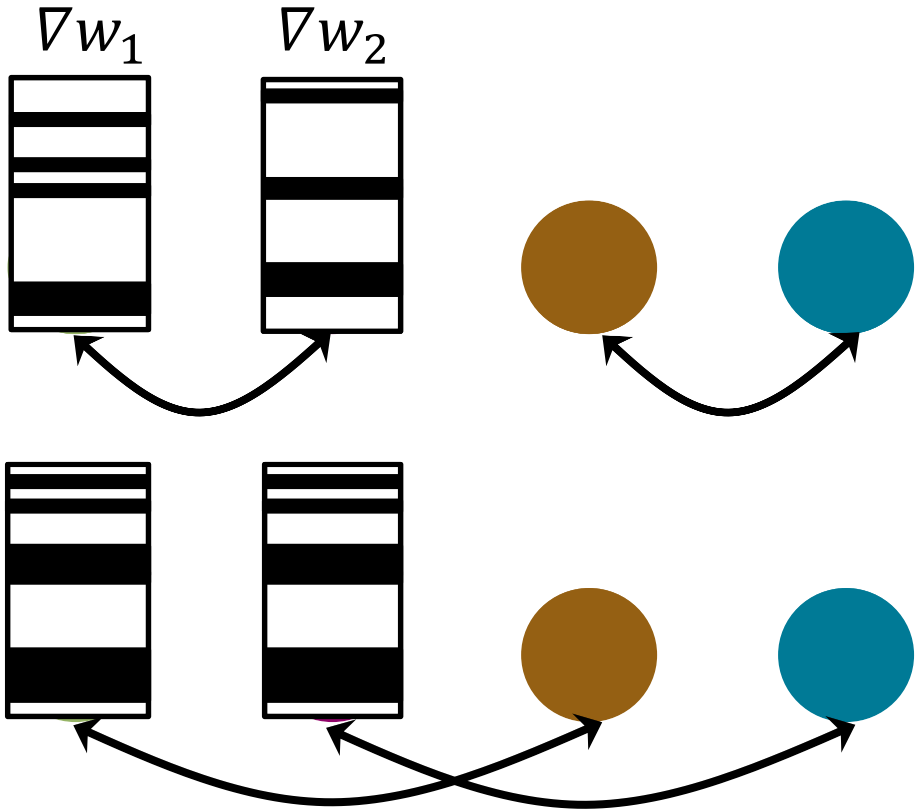
[Renggli et al. 2019]

SparCML – Quantized sparse allreduce



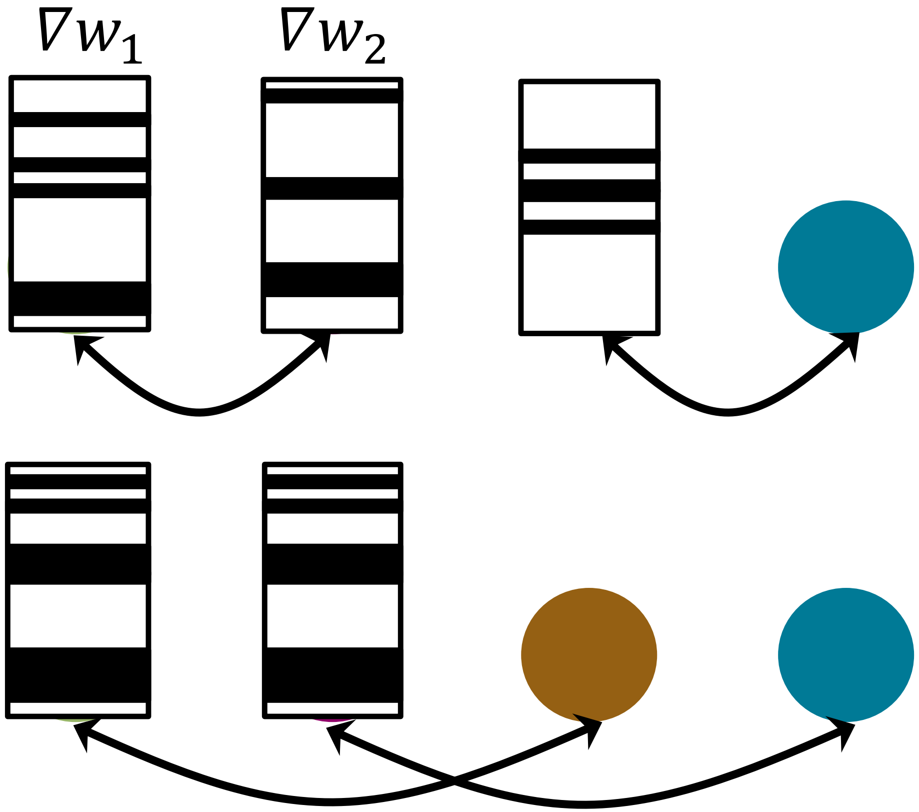
[Renggli et al. 2019]

SparCML – Quantized sparse allreduce



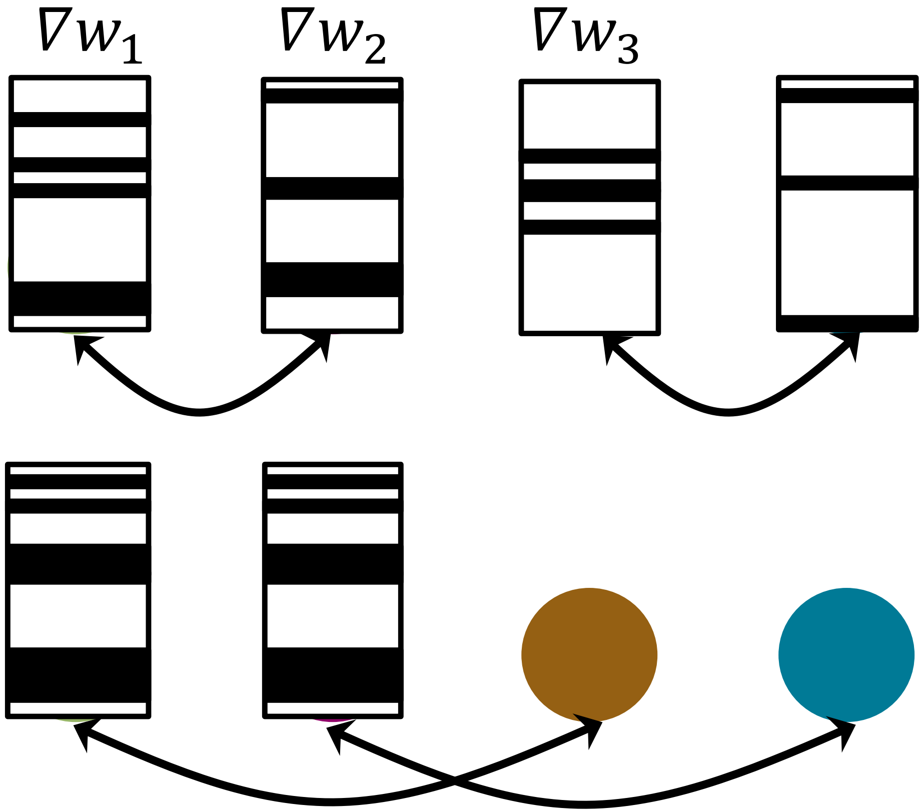
[Renggli et al. 2019]

SparCML – Quantized sparse allreduce



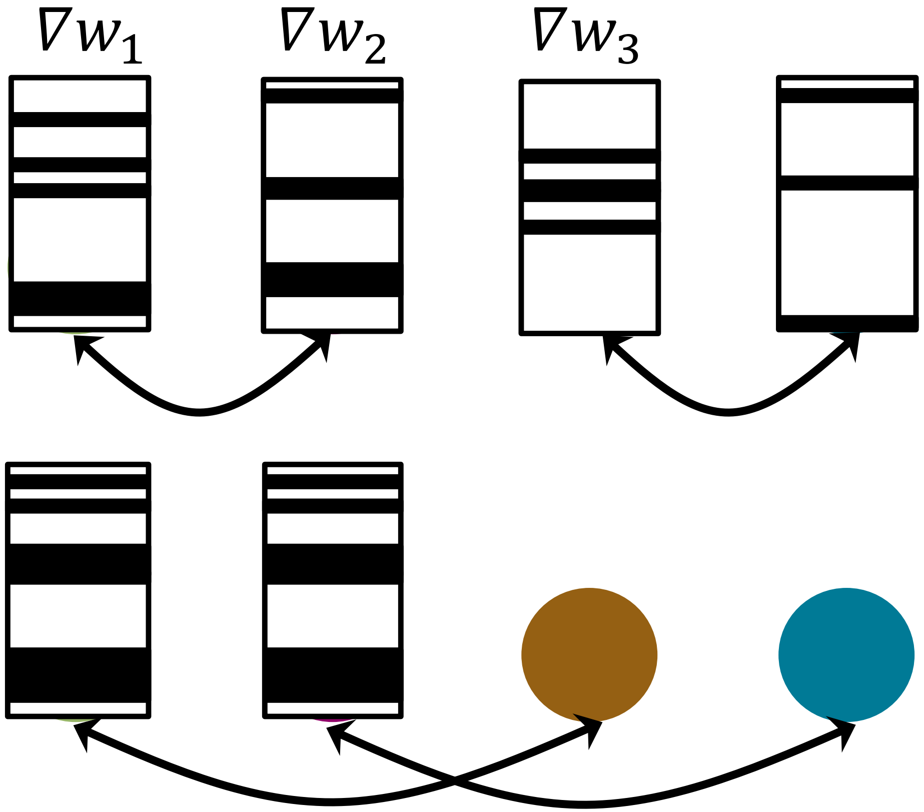
[Renggli et al. 2019]

SparCML – Quantized sparse allreduce



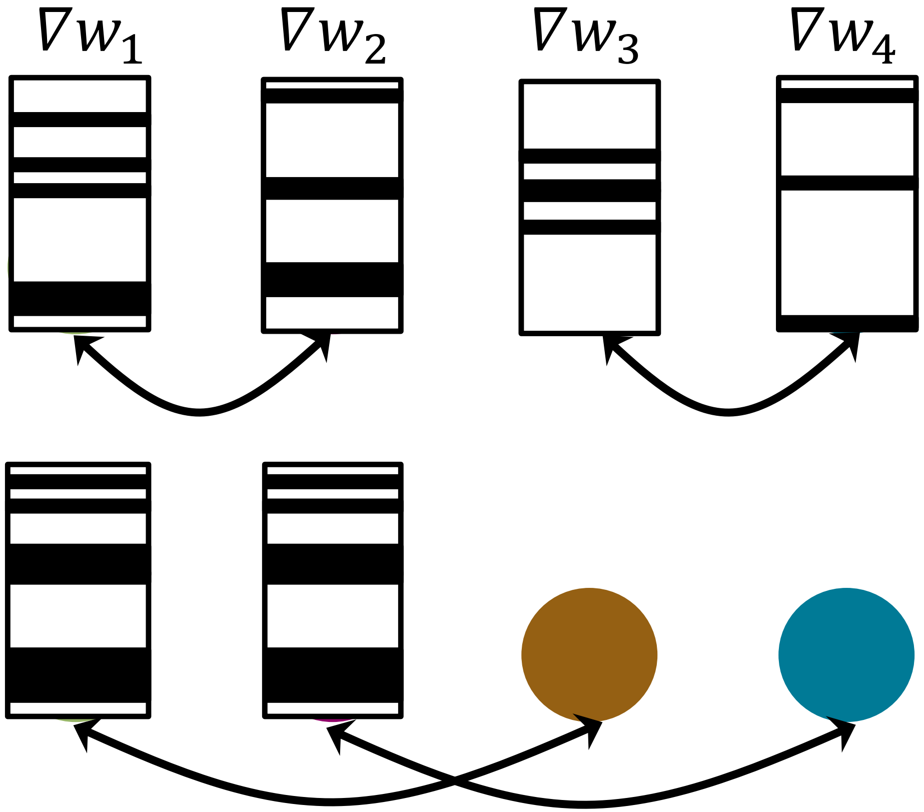
[Renggli et al. 2019]

SparCML – Quantized sparse allreduce



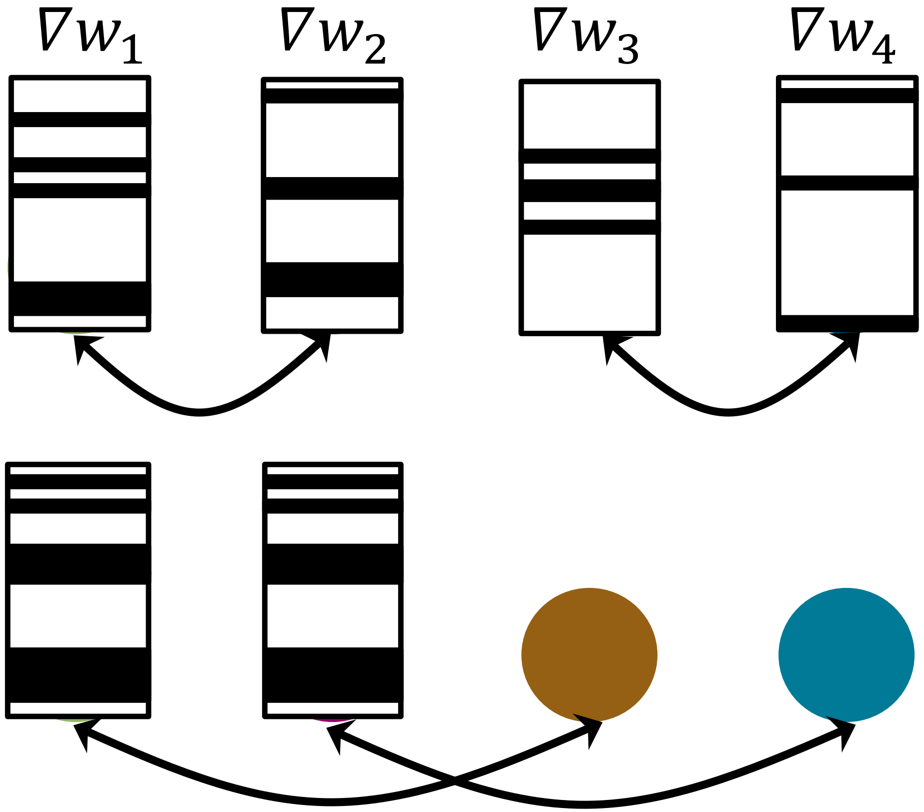
[Renggli et al. 2019]

SparCML – Quantized sparse allreduce



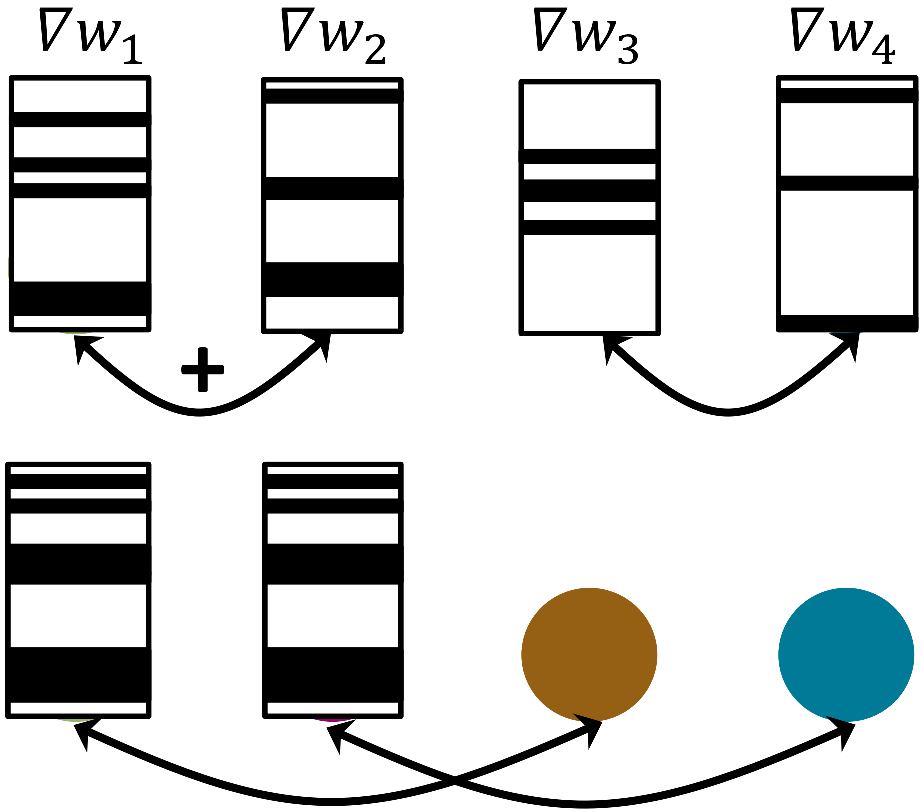
[Renggli et al. 2019]

SparCML – Quantized sparse allreduce



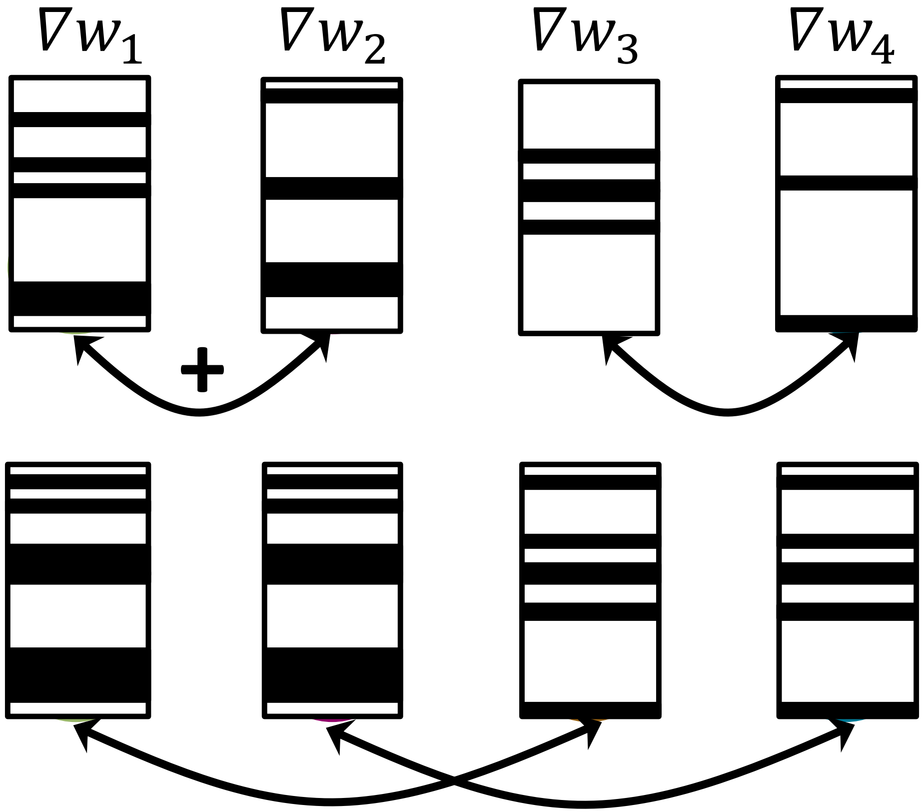
[Renggli et al. 2019]

SparCML – Quantized sparse allreduce



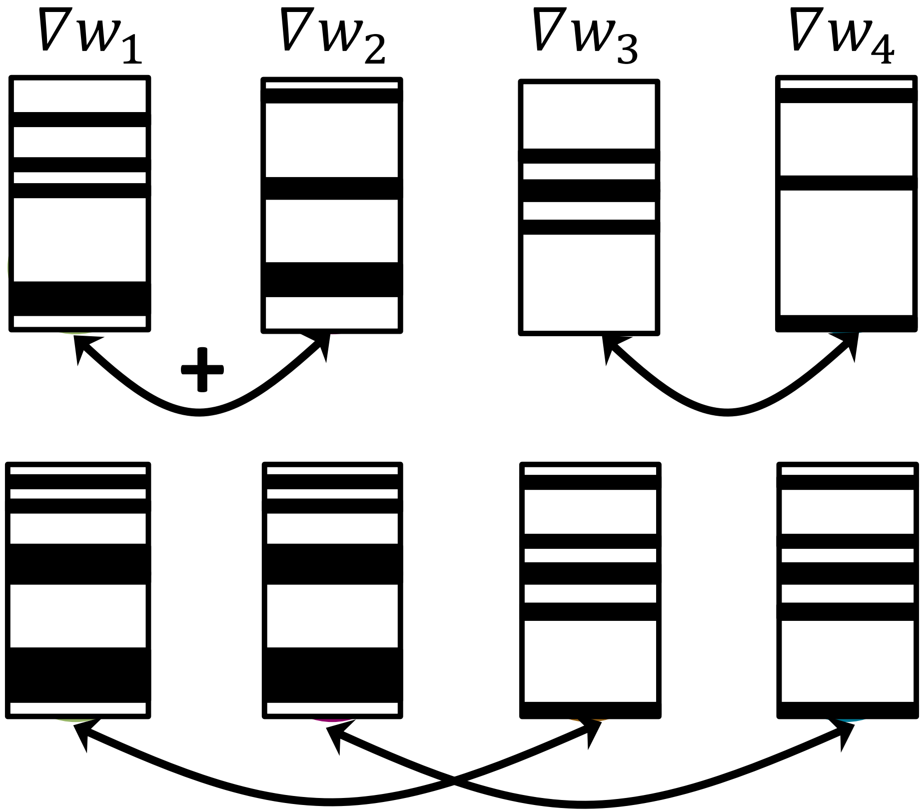
[Renggli et al. 2019]

SparCML – Quantized sparse allreduce



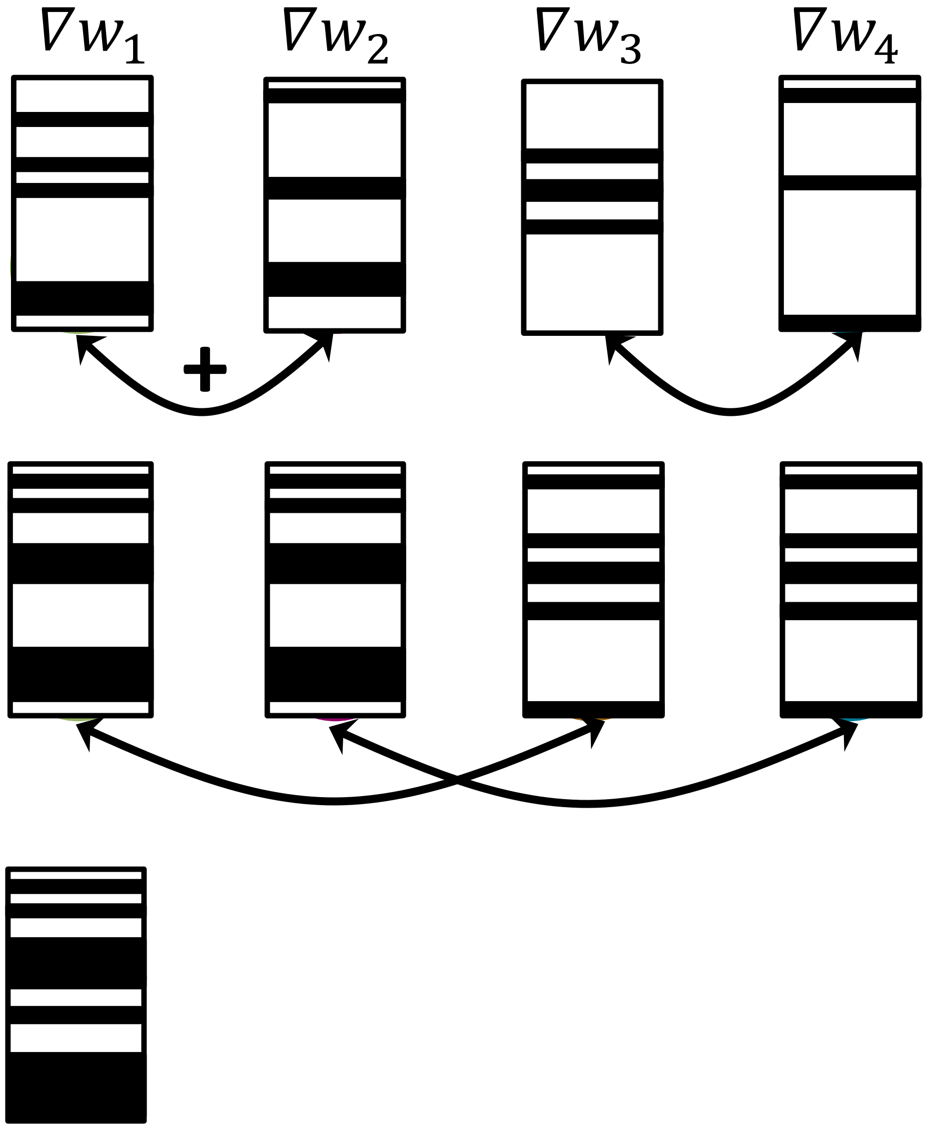
[Renggli et al. 2019]

SparCML – Quantized sparse allreduce



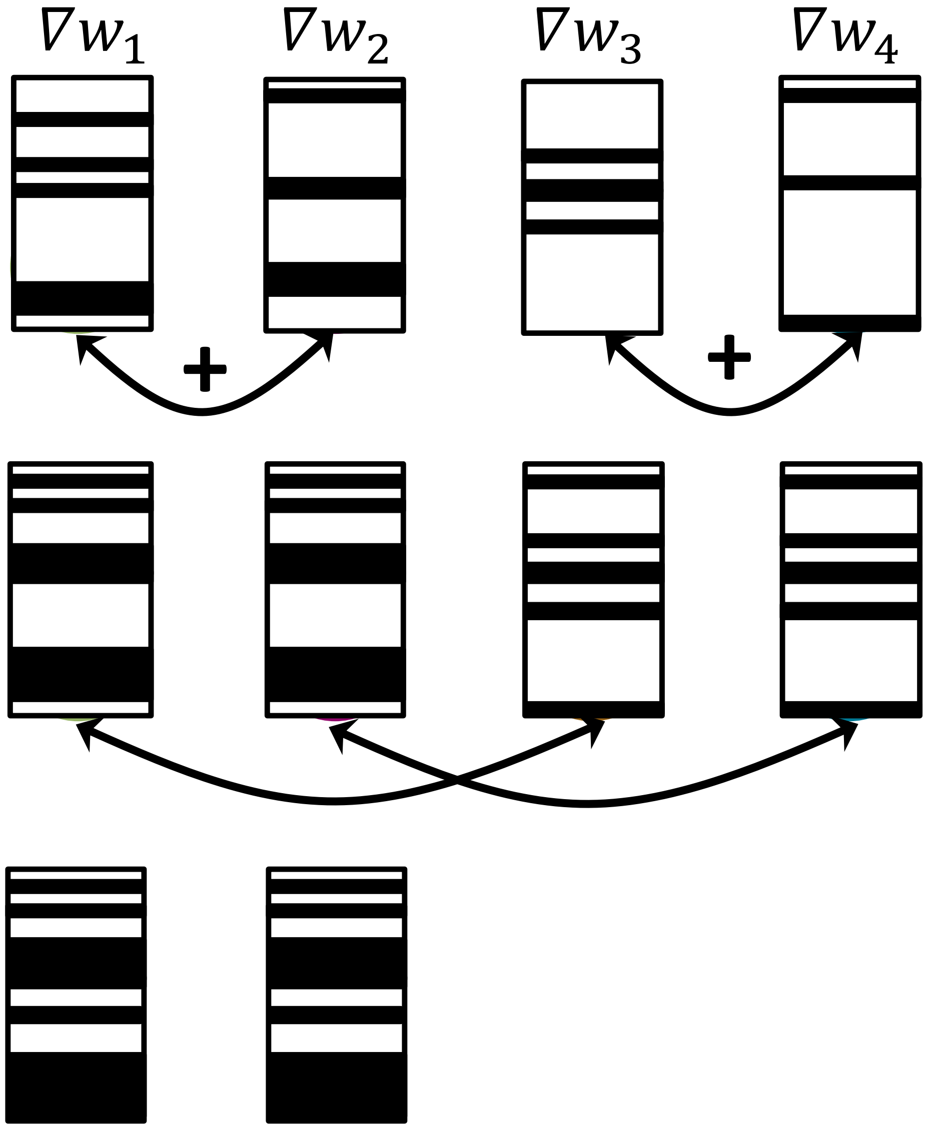
[Renggli et al. 2019]

SparCML – Quantized sparse allreduce



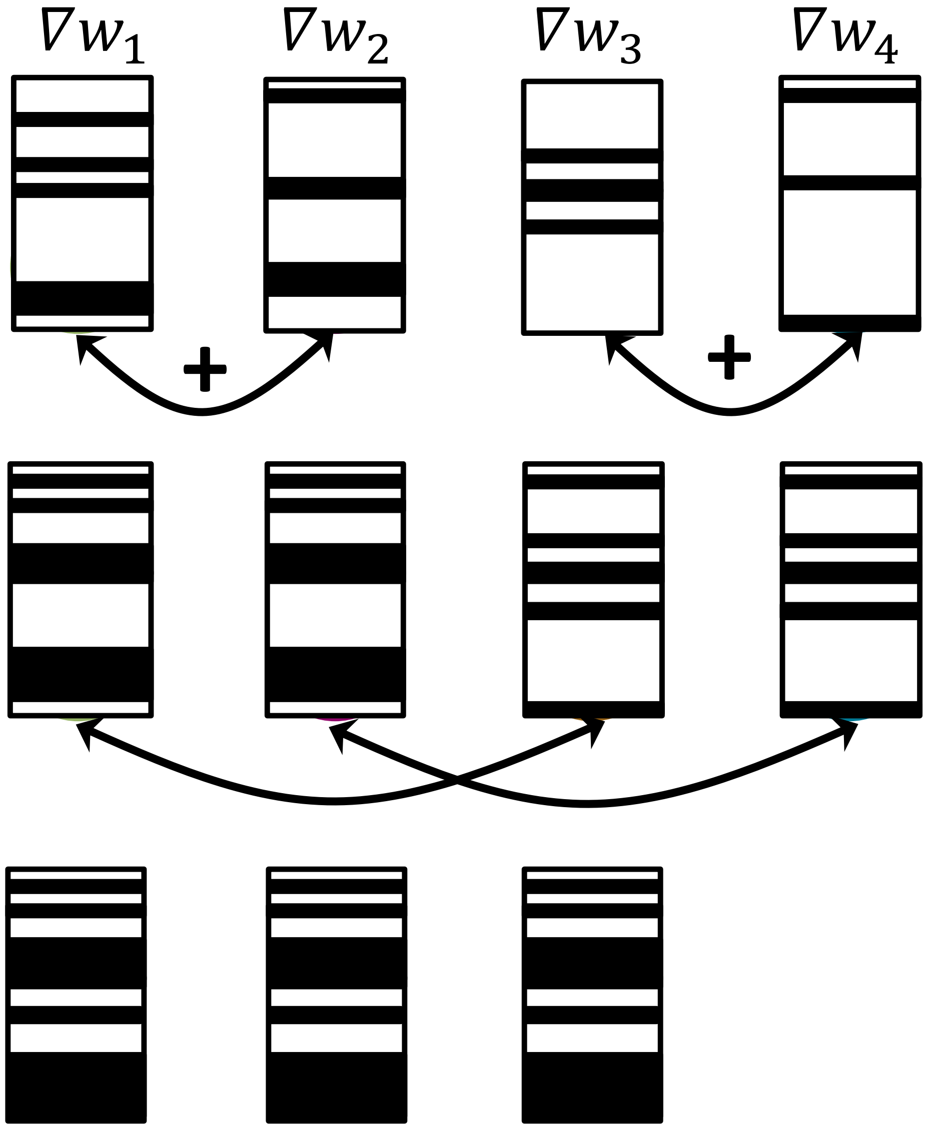
[Renggli et al. 2019]

SparCML – Quantized sparse allreduce



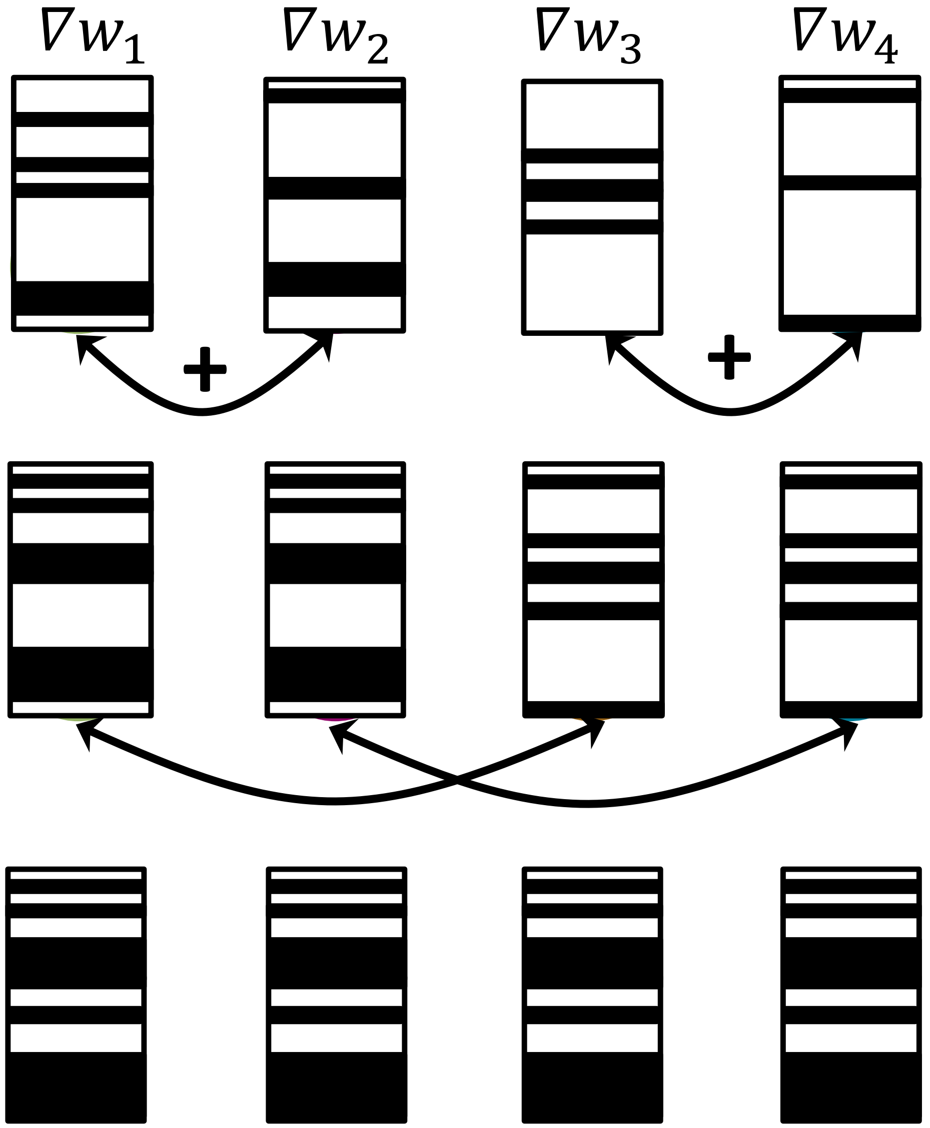
[Renggli et al. 2019]

SparCML – Quantized sparse allreduce



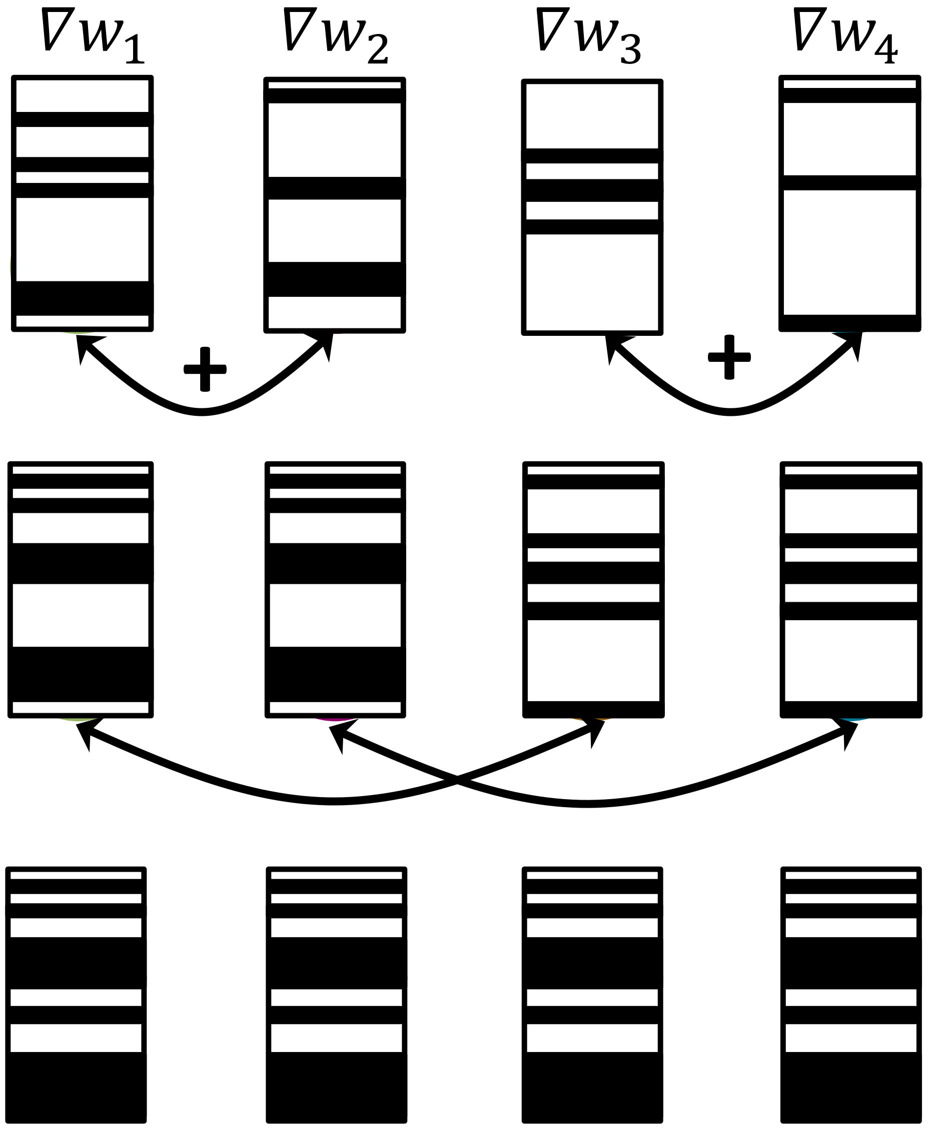
[Renggli et al. 2019]

SparCML – Quantized sparse allreduce



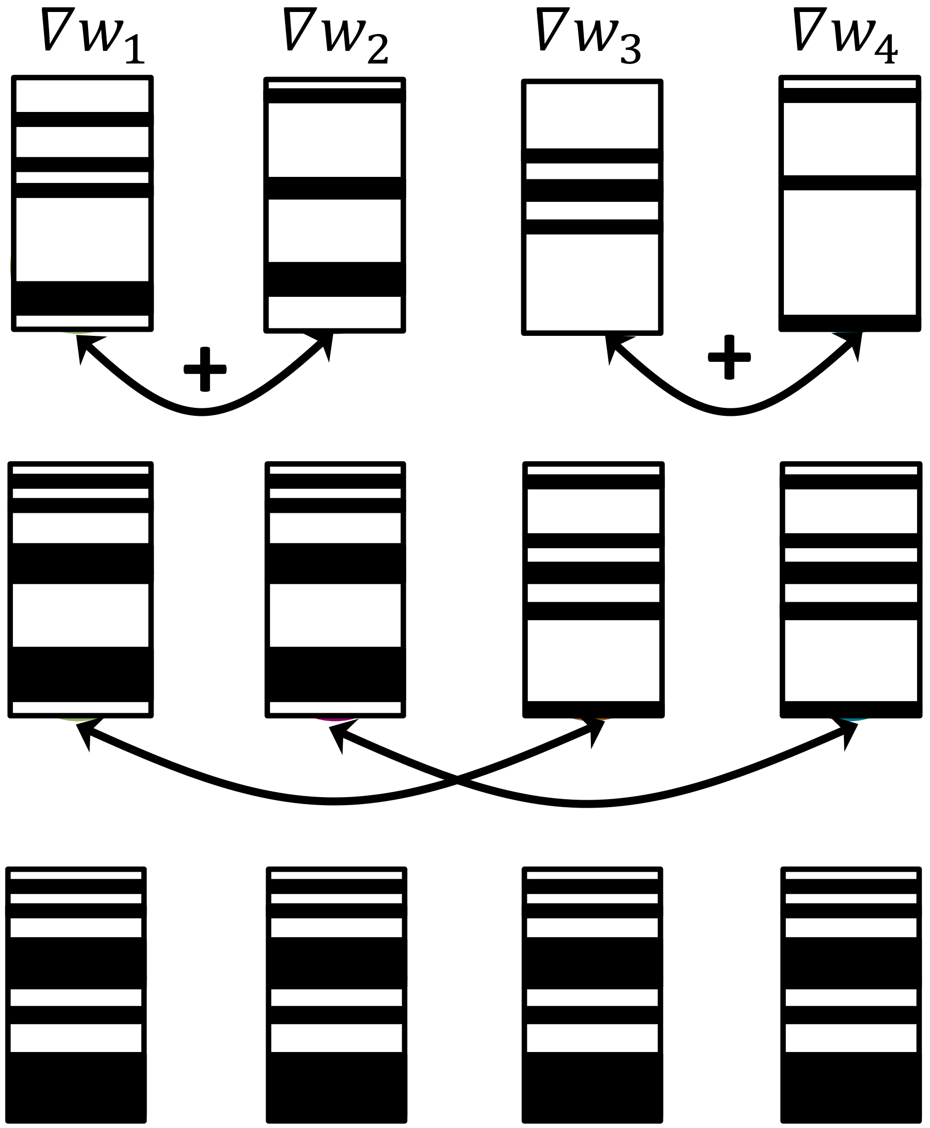
[Renggli et al. 2019]

SparCML – Quantized sparse allreduce



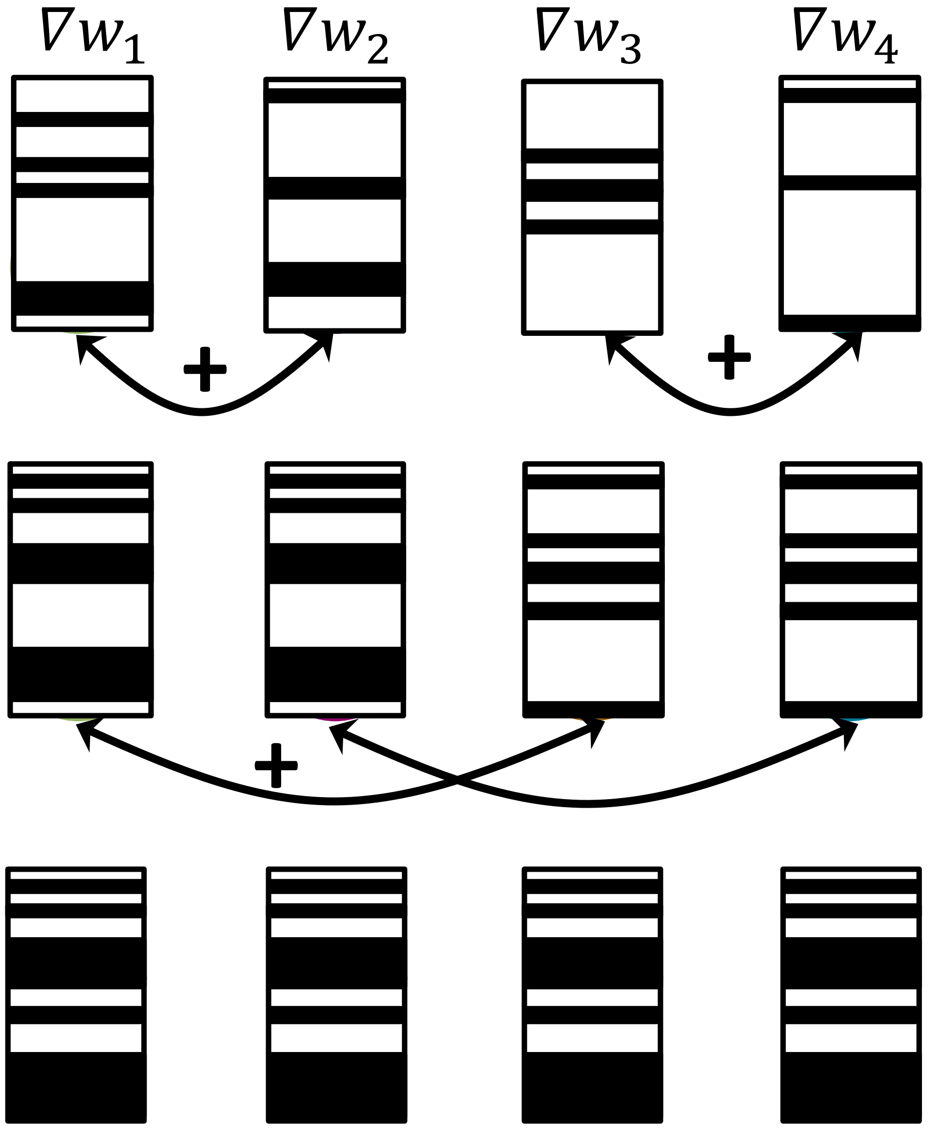
[Renggli et al. 2019]

SparCML – Quantized sparse allreduce



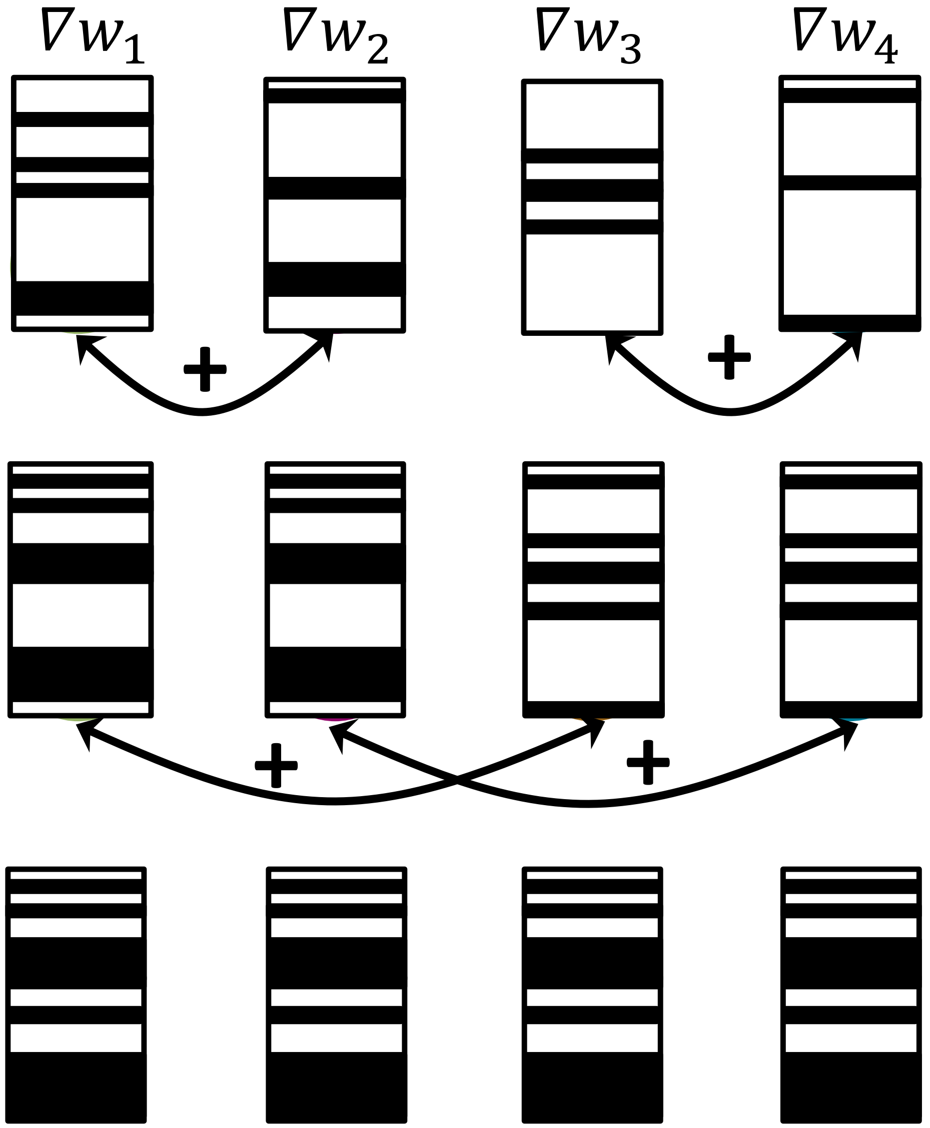
[Renggli et al. 2019]

SparCML – Quantized sparse allreduce



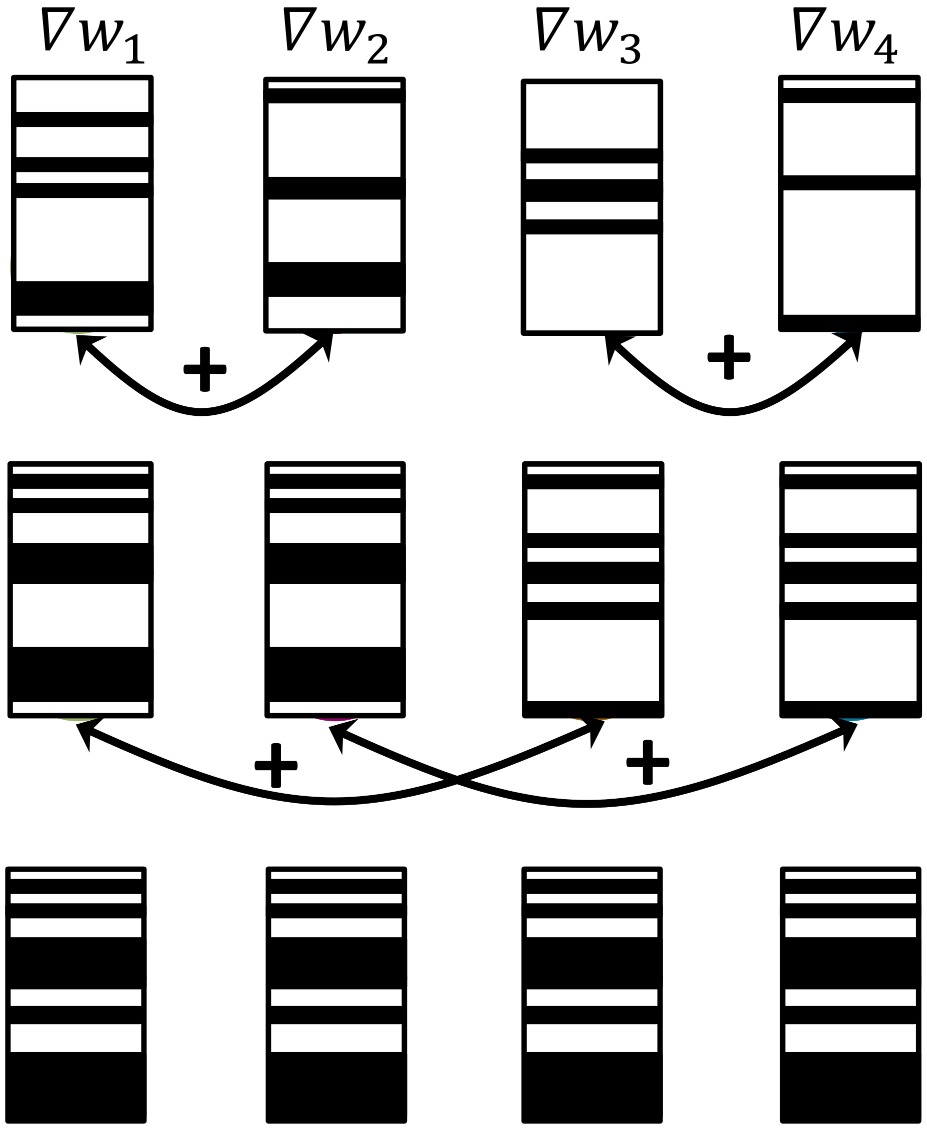
[Renggli et al. 2019]

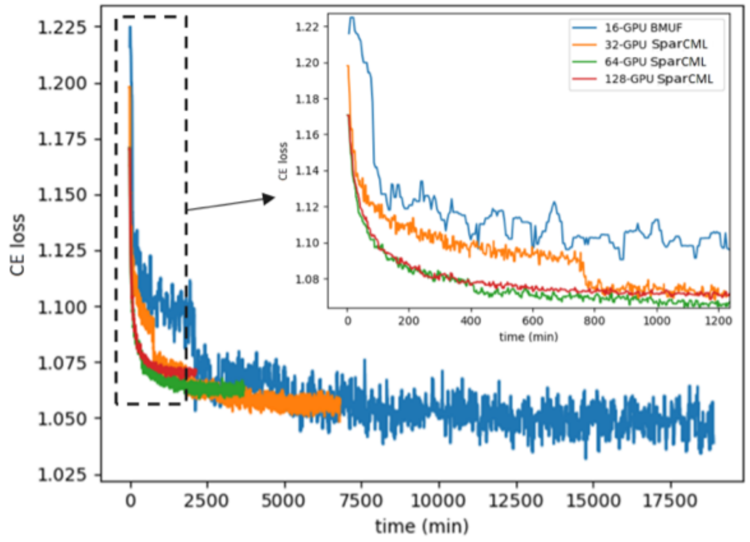
SparCML – Quantized sparse allreduce

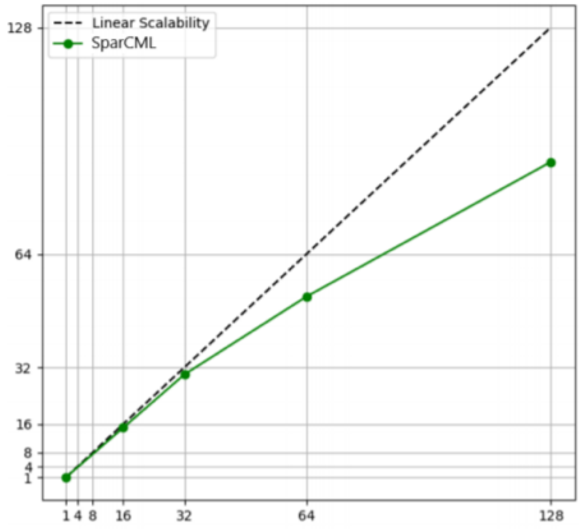


[Renggli et al. 2019]

SparCML – Quantized sparse allreduce



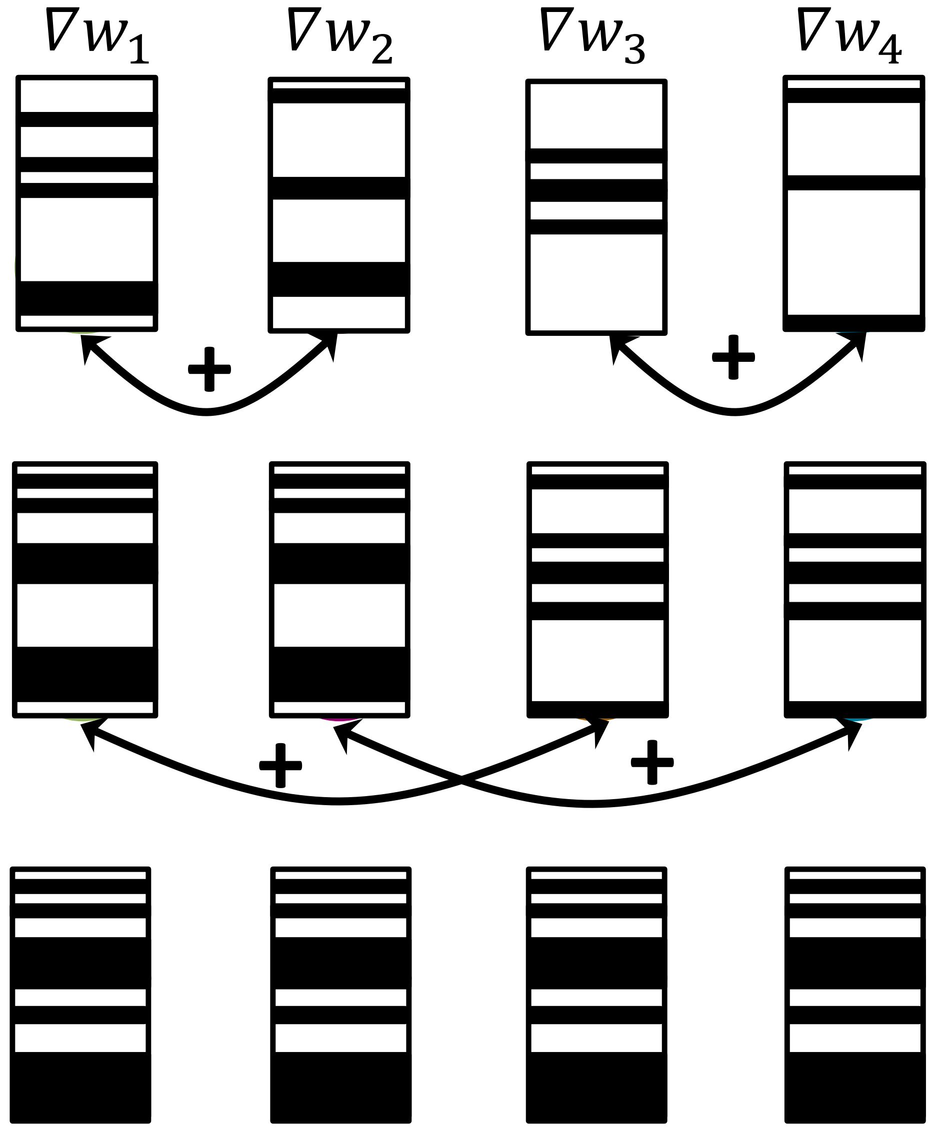


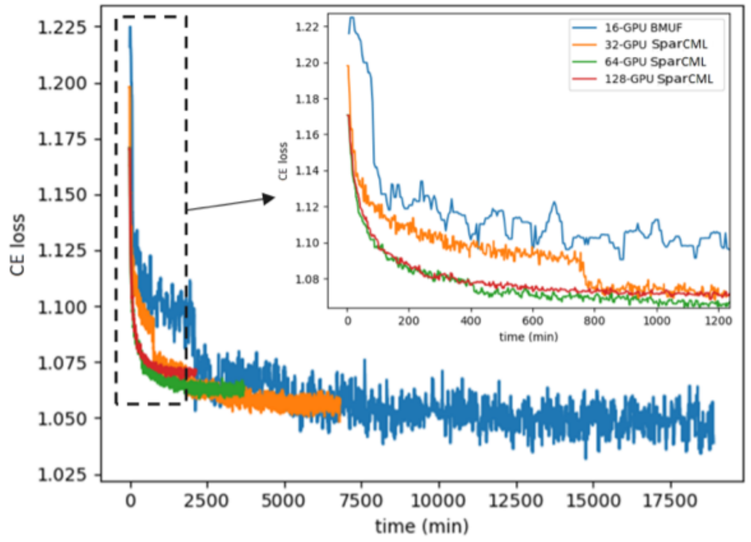


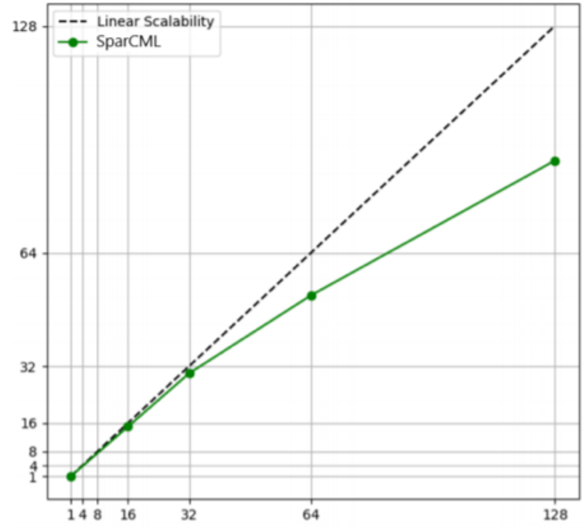
System	Dataset	Model	# of nodes	Algorithm	Speedup
Piz Daint	ImageNet	VGG19	8	Q4	**1.55 (3.31)**
Piz Daint	ImageNet	AlexNet	16	Q4	**1.30 (1.36)**
Piz Daint EC2	MNIST	MLP	8	Top16_Q4 Top16_Q4	**3.65 (4.53)** **19.12 (22.97)**

[Renggli et al. 2019]

SparCML – Quantized sparse allreduce





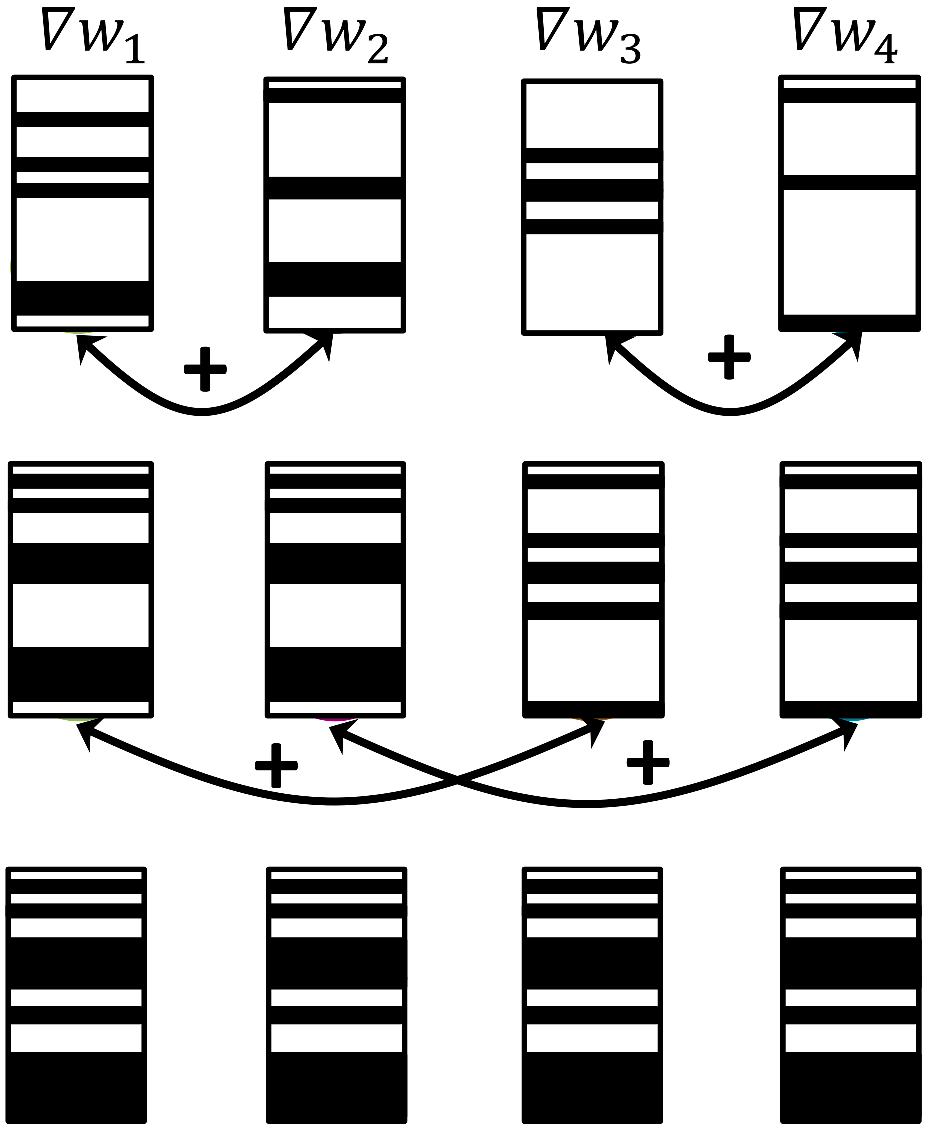


Microsoft Speech Production Workload Results – **2 weeks → 2 days**!

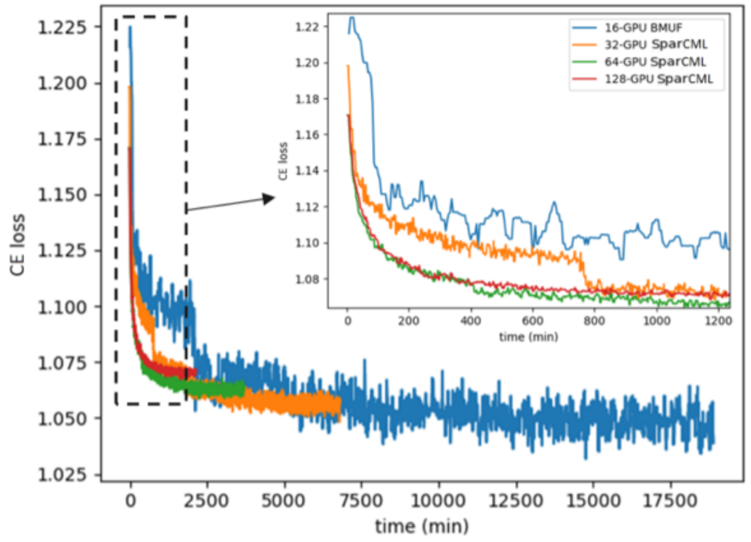
System	Dataset	Model	# of nodes	Algorithm	Speedup
Piz Daint	ImageNet	VGG19	8	Q4	1.55 (3.31)
Piz Daint	ImageNet	AlexNet	16	Q4	1.30 (1.36)
Piz Daint	MNIST	MLP	8	Top16_Q4	3.65 (4.53)
EC2	MNIST	MLP	8	Top16_Q4	19.12 (22.97)

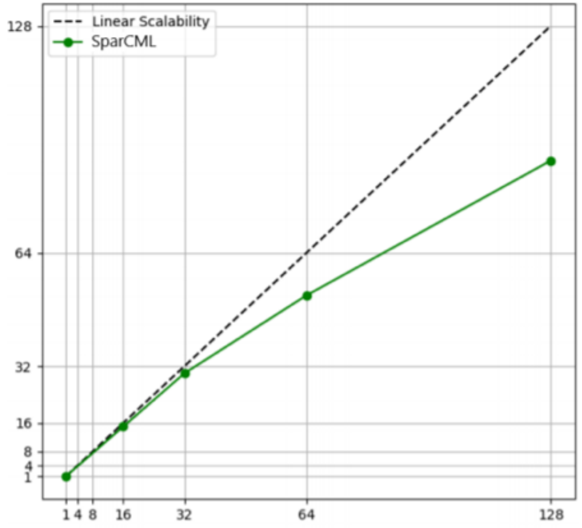
[Renggli et al. 2019]

SparCML – Quantized sparse allreduce



Six epochs, 60 million params



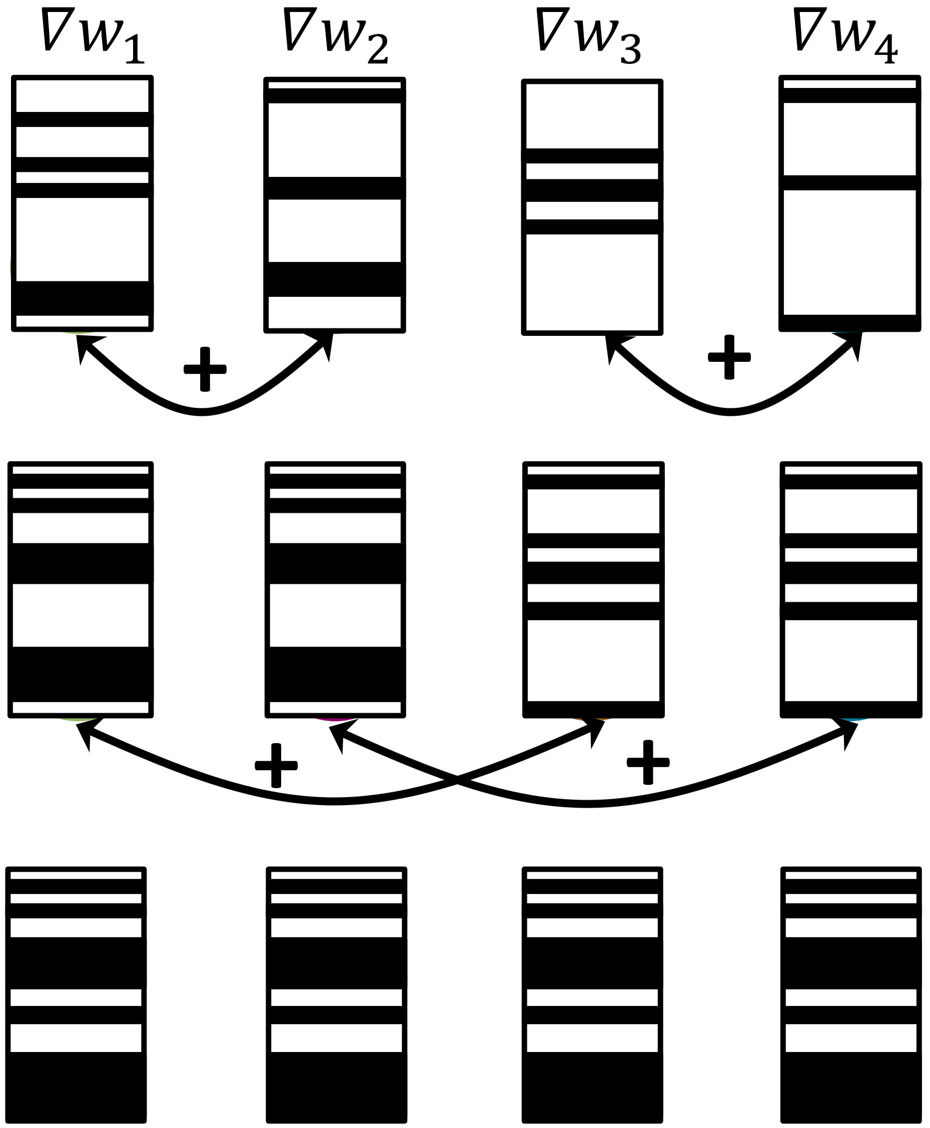


Microsoft Speech Production Workload Results – **2 weeks → 2 days**!

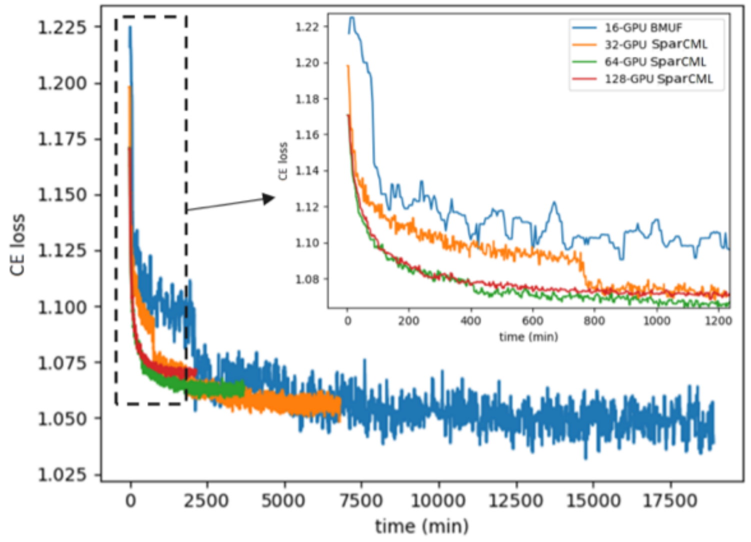
System	Dataset	Model	# of nodes	Algorithm	Speedup
Piz Daint	ImageNet	VGG19	8	Q4	**1.55 (3.31)**
Piz Daint	ImageNet	AlexNet	16	Q4	**1.30 (1.36)**
Piz Daint EC2	MNIST	MLP	8	Top16_Q4 Top16_Q4	**3.65 (4.53)** **19.12 (22.97)**

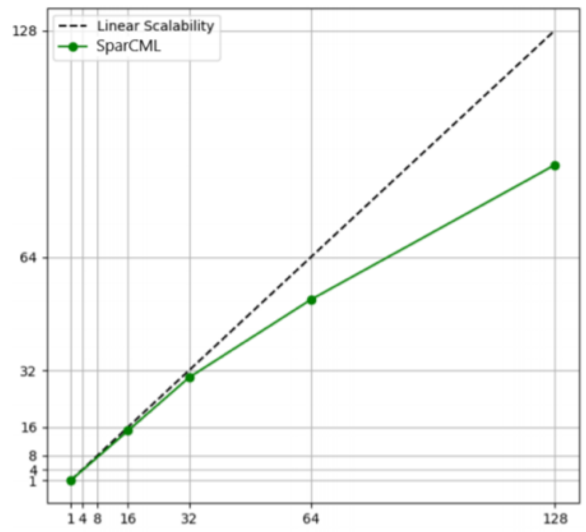
[Renggli et al. 2019]

SparCML – Quantized sparse allreduce



Six epochs, 60 million params





Microsoft Speech Production Workload Results – **2 weeks → 2 days**!

System	Dataset	Model	# of nodes	Algorithm	Speedup
Piz Daint	ImageNet	VGG19	8	Q4	1.55 (3.31)
Piz Daint	ImageNet	AlexNet	16	Q4	1.30 (1.36)
Piz Daint EC2	MNIST	MLP	8	Top16_Q4 Top16_Q4	3.65 (4.53) 19.12 (22.97)

[Renggli et al. 2019]

The limits to data parallelism

The limits to data parallelism

- **When does data parallelism break down?**

- **Communication overheads**
- **Hyperparameter tuning**

- **Memory for one sample**
- **More GPUs than samples in a mini-batch**

The limits to data parallelism

- When does data parallelism break down?

- Communication overheads
- Hyperparameter tuning

- Memory for one sample
- More GPUs than samples in a mini-batch

The limits to data parallelism

- When does data parallelism break down?

- Communication overheads
- Hyperparameter tuning

- Memory for one sample
- More GPUs than samples in a mini-batch

Need to strong scale!

Distributed-memory fully-connected layers

$$Y = \sigma(WX + b)$$

Distributed-memory fully-connected layers

$$Y = \sigma(WX + b)$$

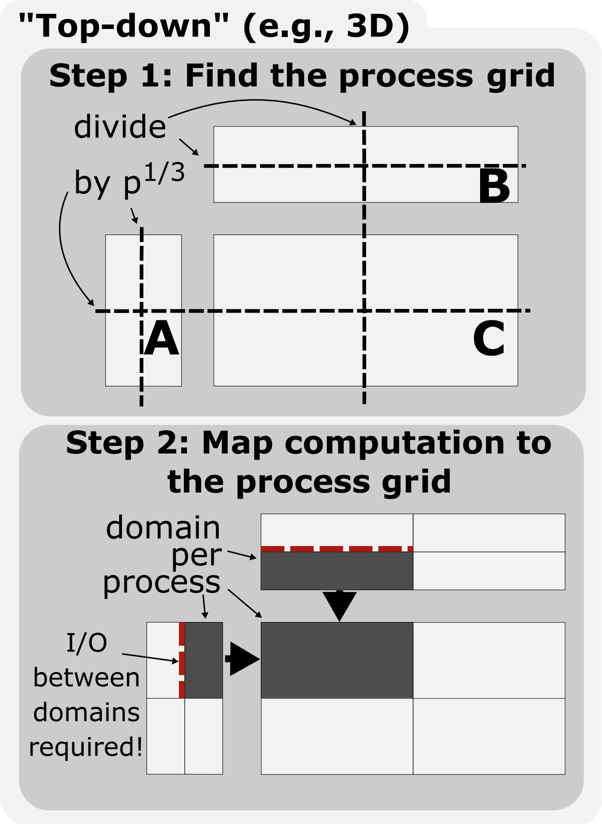
- **Just a distributed matrix-matrix multiplication!**

Distributed-memory fully-connected layers

$$Y = \sigma(WX + b)$$

- **Just a distributed matrix-matrix multiplication!**

Use SUMMA [Van Essen et al. 2015]

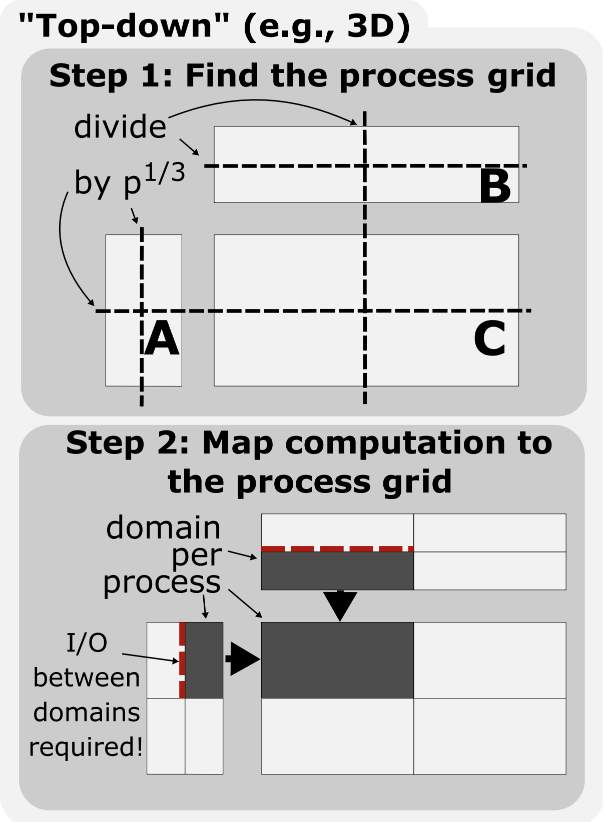


Distributed-memory fully-connected layers

$$Y = \sigma(WX + b)$$

- **Just a distributed matrix-matrix multiplication!**

Use SUMMA [Van Essen et al. 2015]

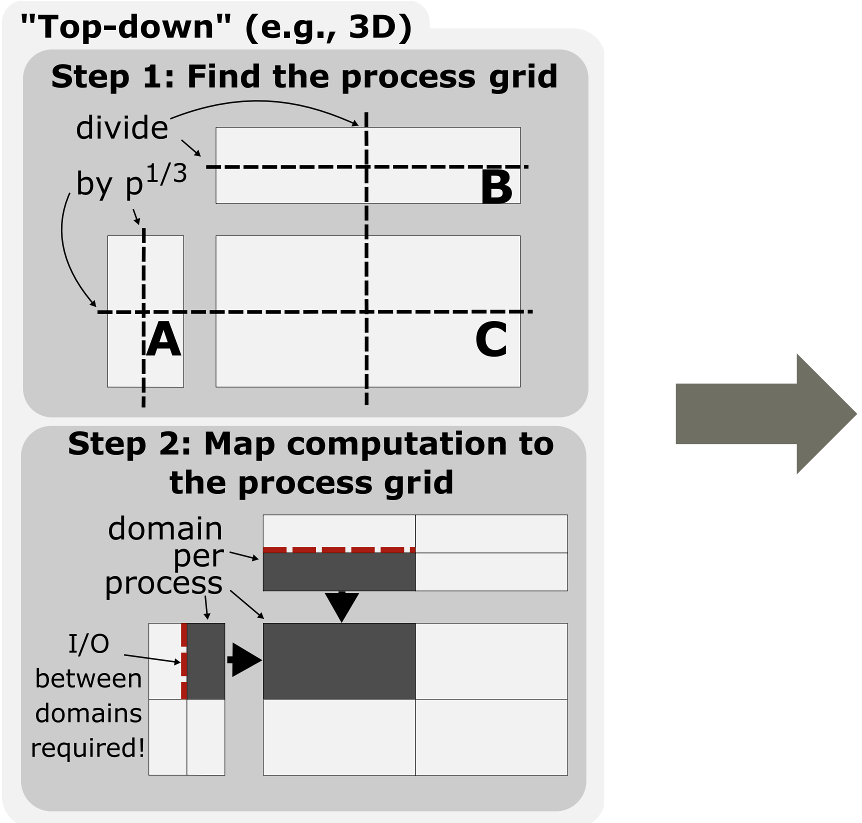


Distributed-memory fully-connected layers

$$Y = \sigma(WX + b)$$

- **Just a distributed matrix-matrix multiplication!**

Use SUMMA [Van Essen et al. 2015]



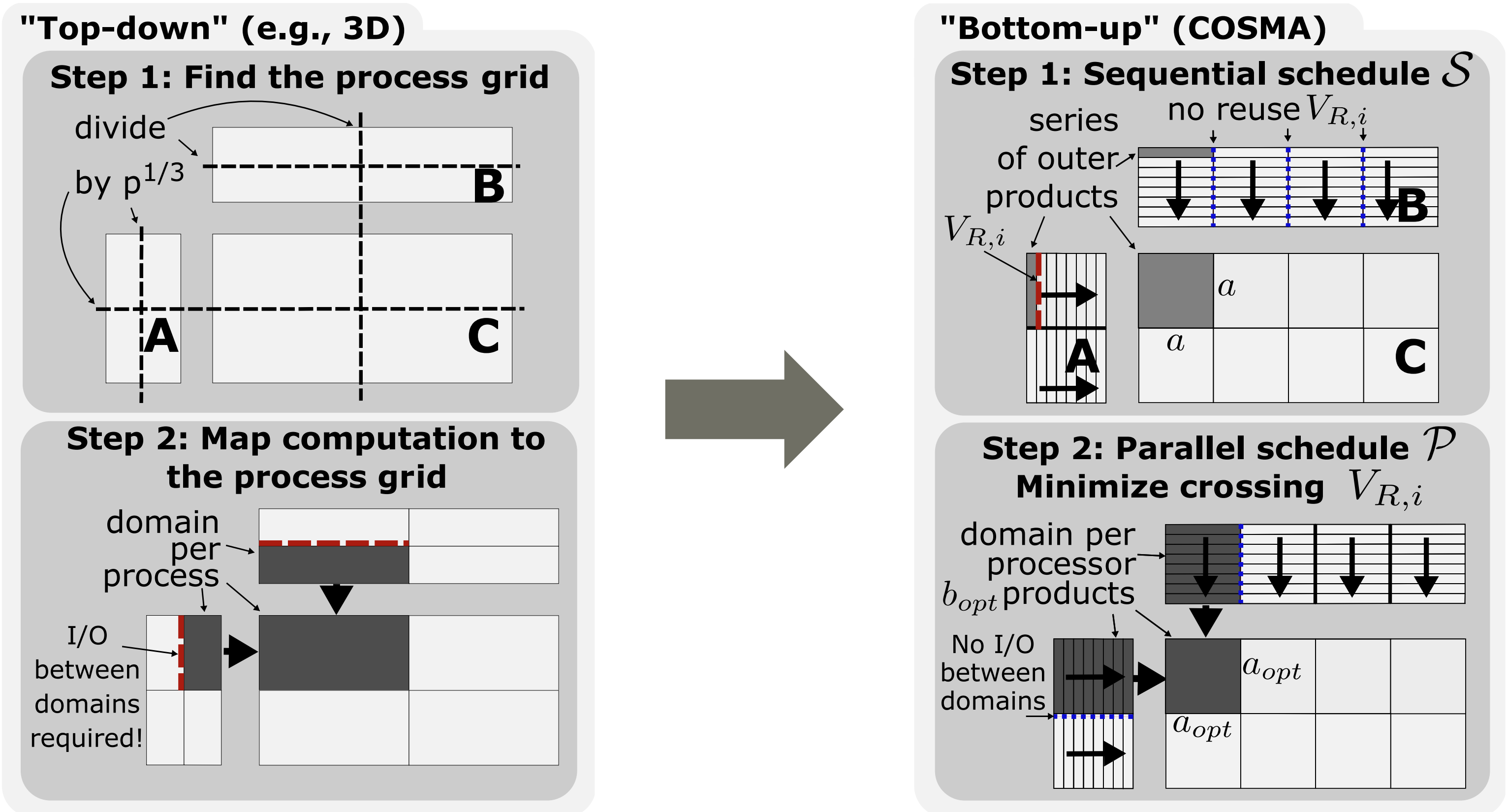
Distributed-memory fully-connected layers

$$Y = \sigma(WX + b)$$

- **Just a distributed matrix-matrix multiplication!**

Use SUMMA [Van Essen et al. 2015]

New: COSMA [Kwasniewski et al. 2019]

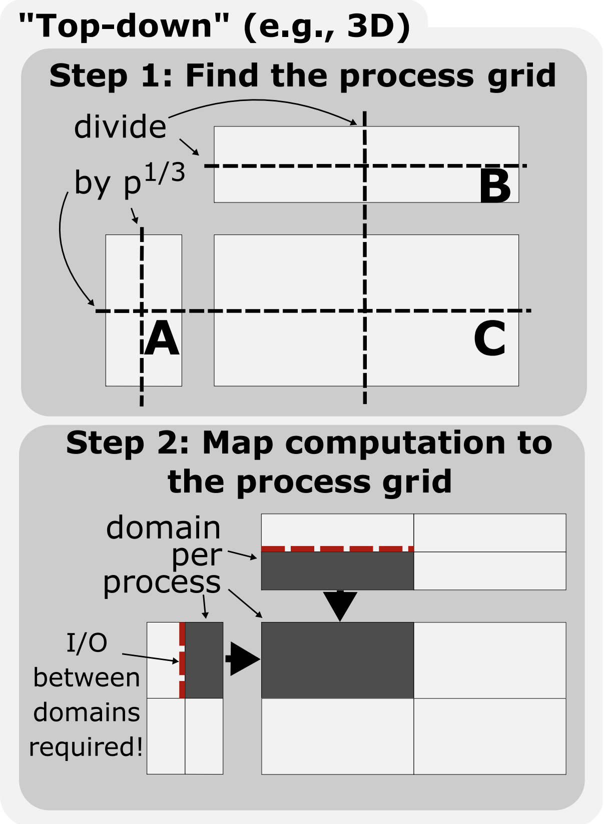


Distributed-memory fully-connected layers

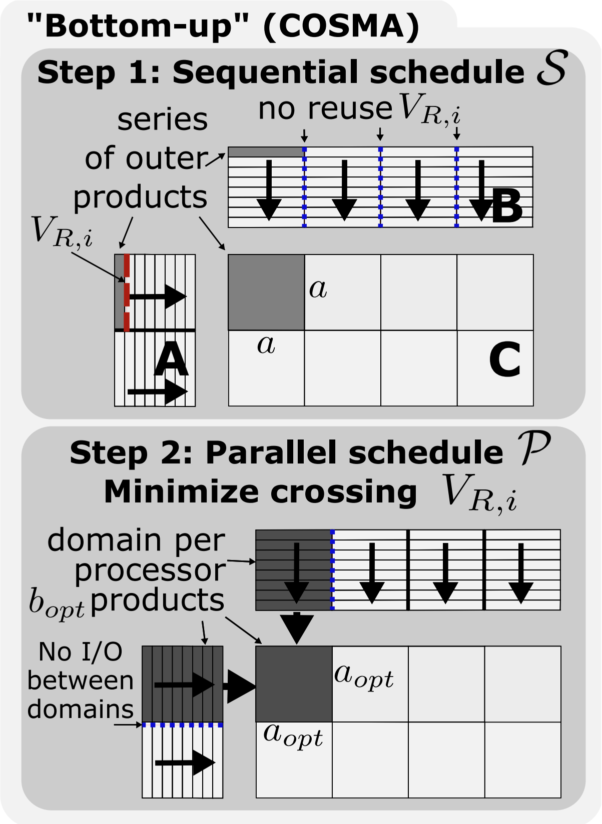
$$Y = \sigma(WX + b)$$

- **Just a distributed matrix-matrix multiplication!**

Use SUMMA [Van Essen et al. 2015]



New: COSMA [Kwasniewski et al. 2019]



Spatial parallelism [Dryden et al. 2019]

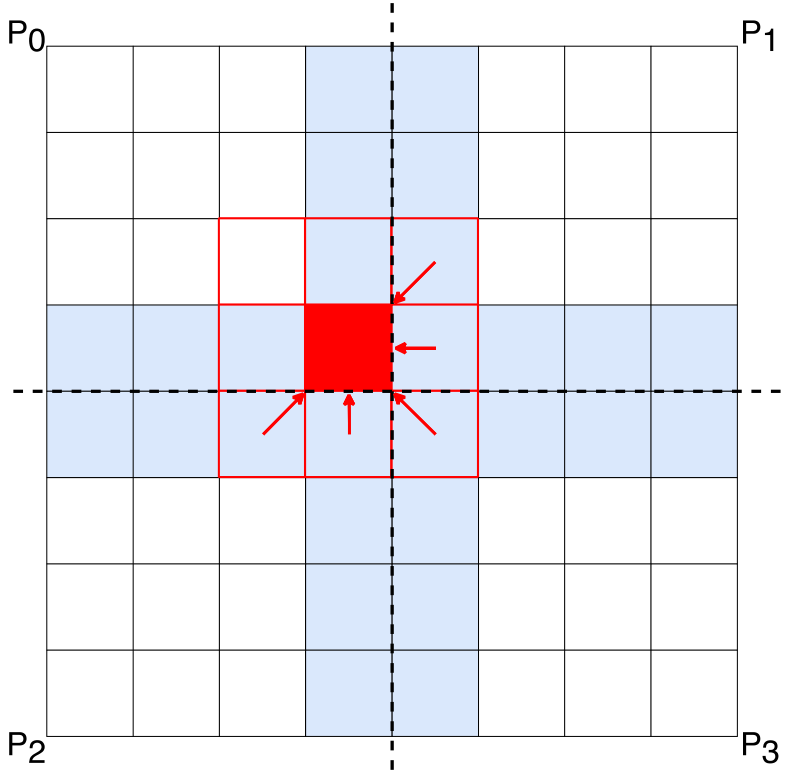
Spatial parallelism [Dryden et al. 2019]

- **Observation: Convolution is just a funny stencil operation**
- **Domain decomposition with a halo exchange!**

Spatial parallelism [Dryden et al. 2019]

- **Observation: Convolution is just a funny stencil operation**
- **Domain decomposition with a halo exchange!**

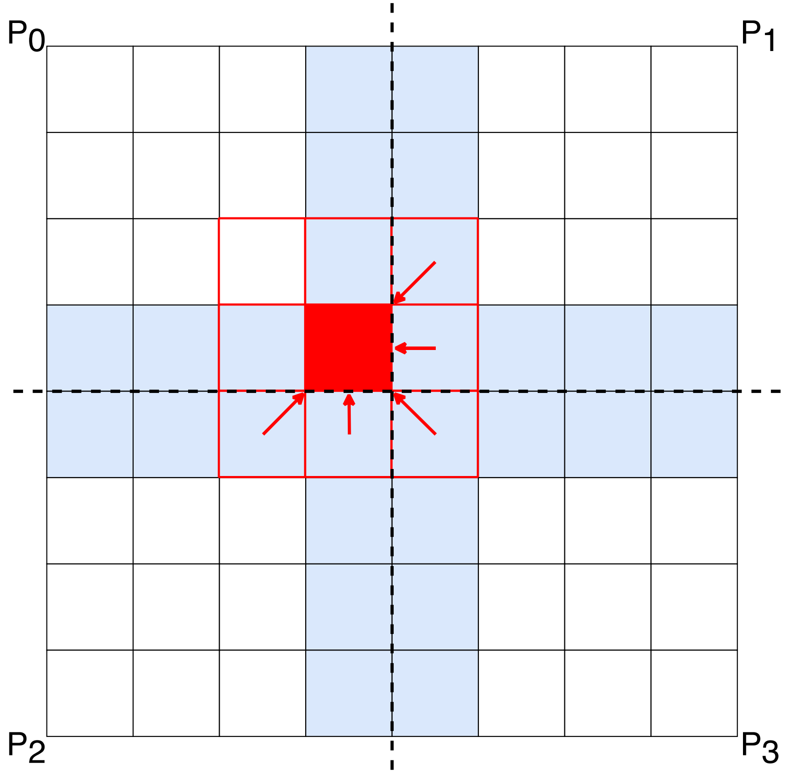
Convolution dependencies

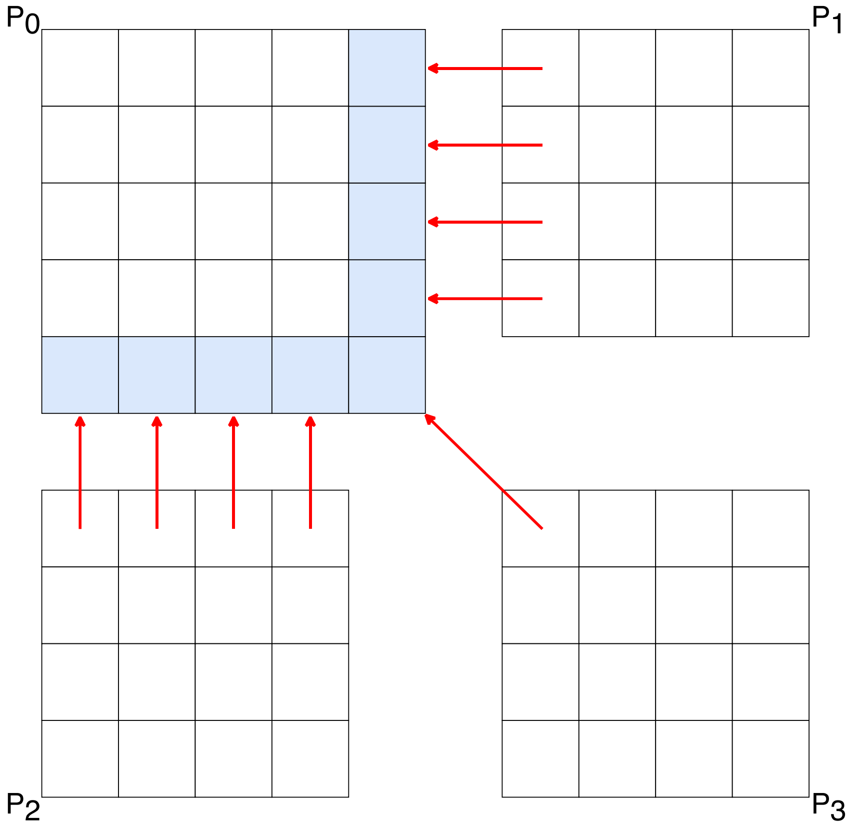


Spatial parallelism [Dryden et al. 2019]

- **Observation: Convolution is just a funny stencil operation**
- **Domain decomposition with a halo exchange!**

Convolution dependencies

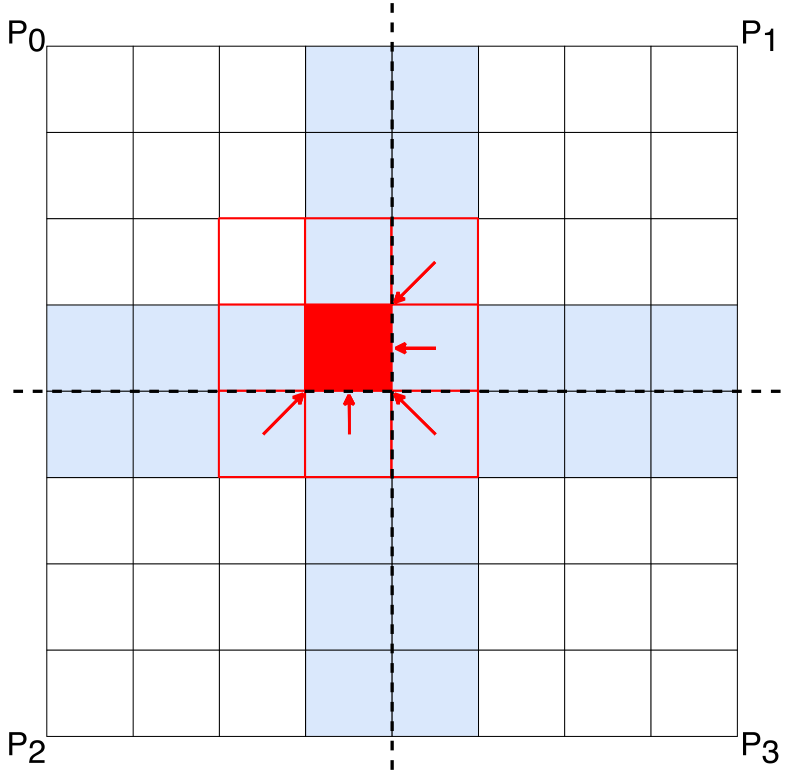




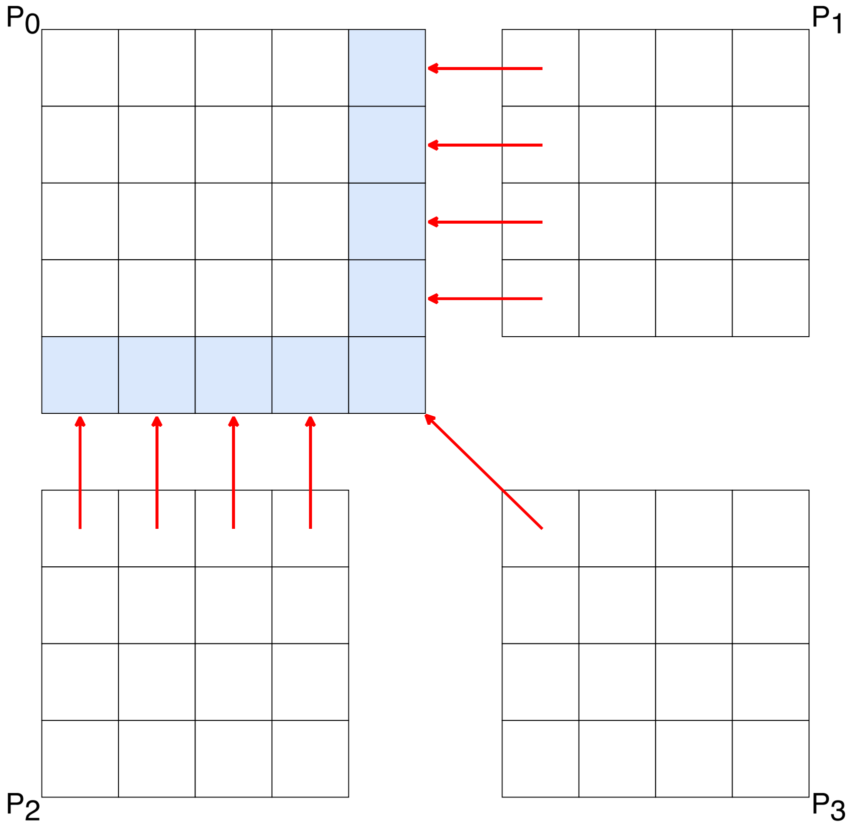
Spatial parallelism [Dryden et al. 2019]

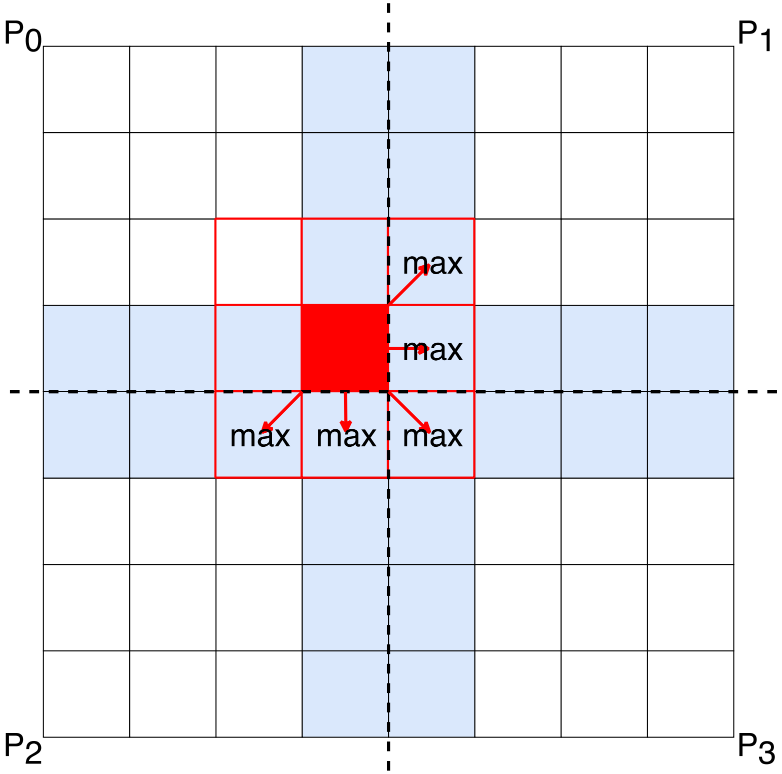
- Observation: Convolution is just a funny stencil operation
- Domain decomposition with a halo exchange!

Convolution dependencies



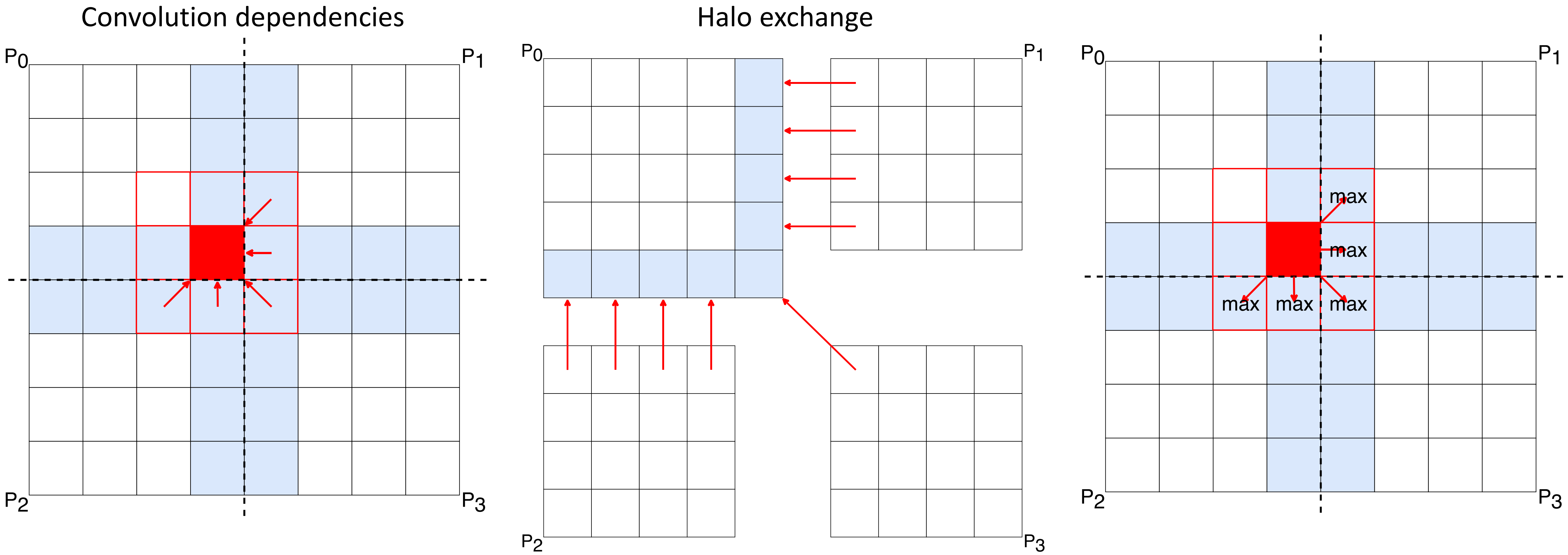
Halo exchange





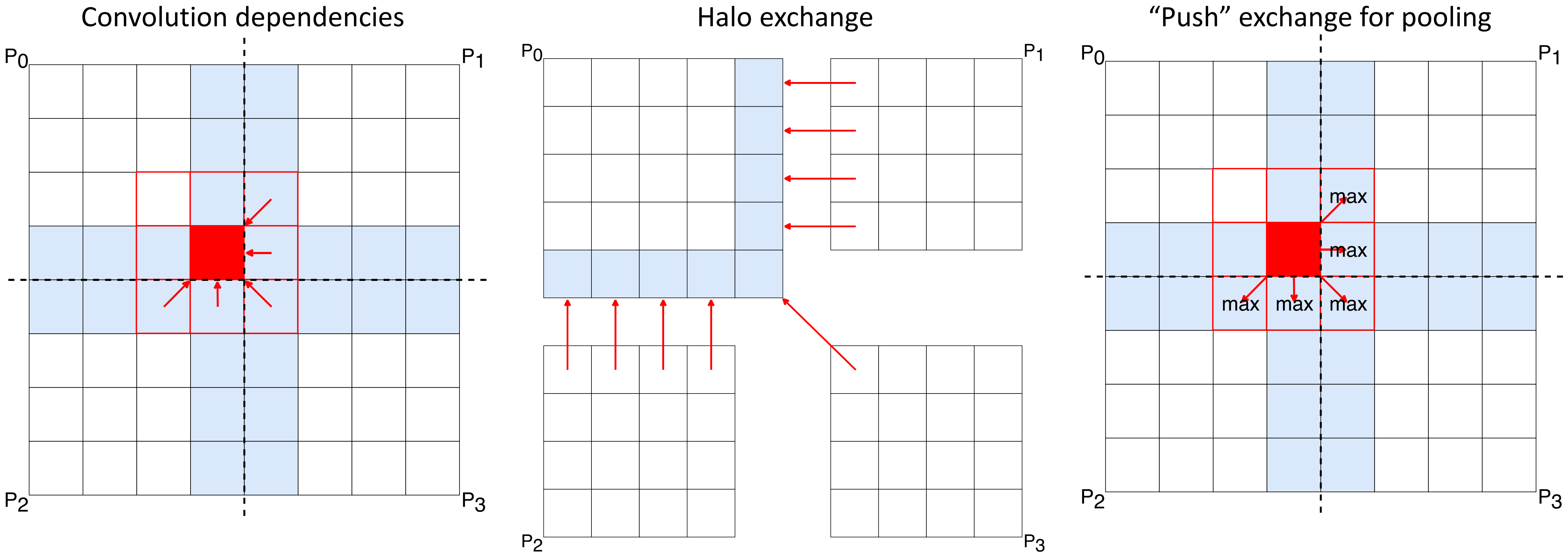
Spatial parallelism [Dryden et al. 2019]

- **Observation: Convolution is just a funny stencil operation**
- **Domain decomposition with a halo exchange!**



Spatial parallelism [Dryden et al. 2019]

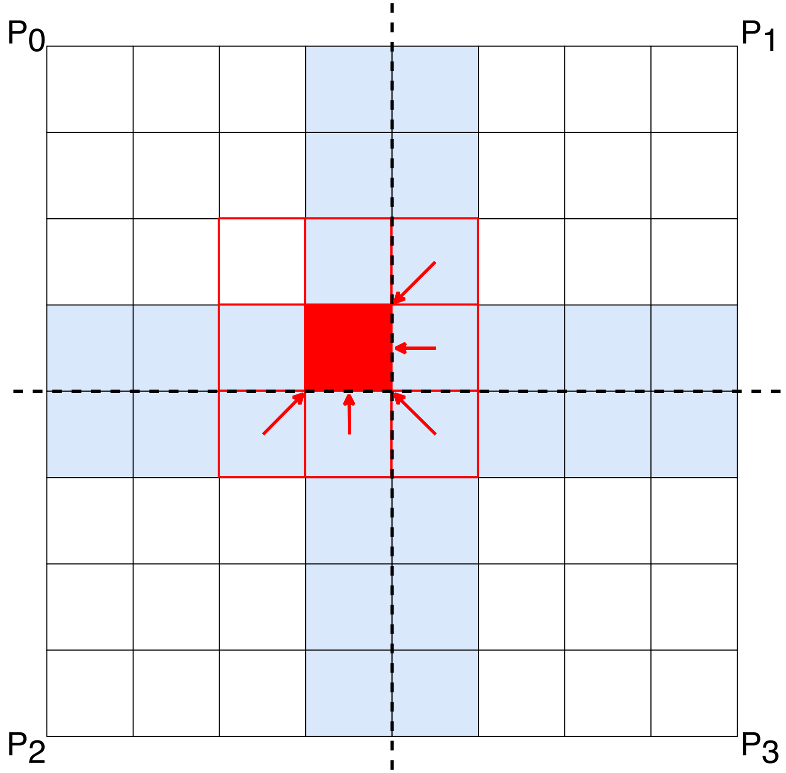
- **Observation: Convolution is just a funny stencil operation**
- **Domain decomposition with a halo exchange!**



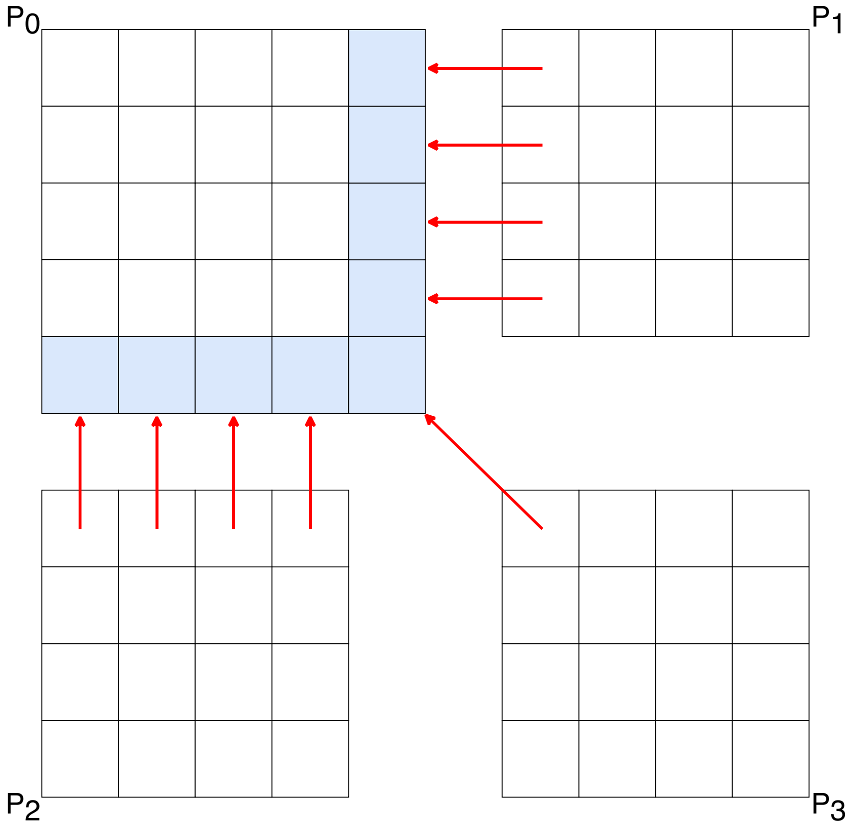
Spatial parallelism [Dryden et al. 2019]

- Observation: Convolution is just a funny stencil operation
- Domain decomposition with a halo exchange!

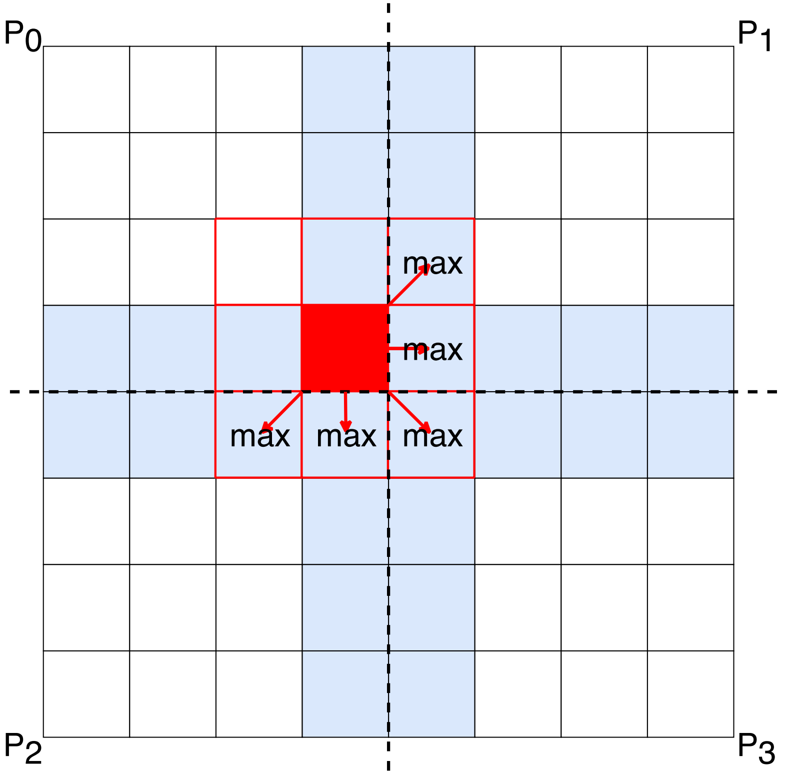
Convolution dependencies



Halo exchange

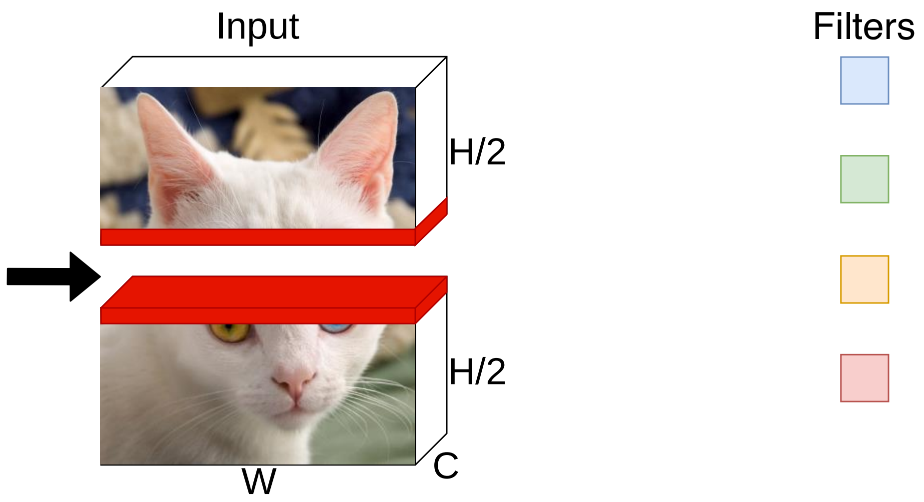


"Push" exchange for pooling

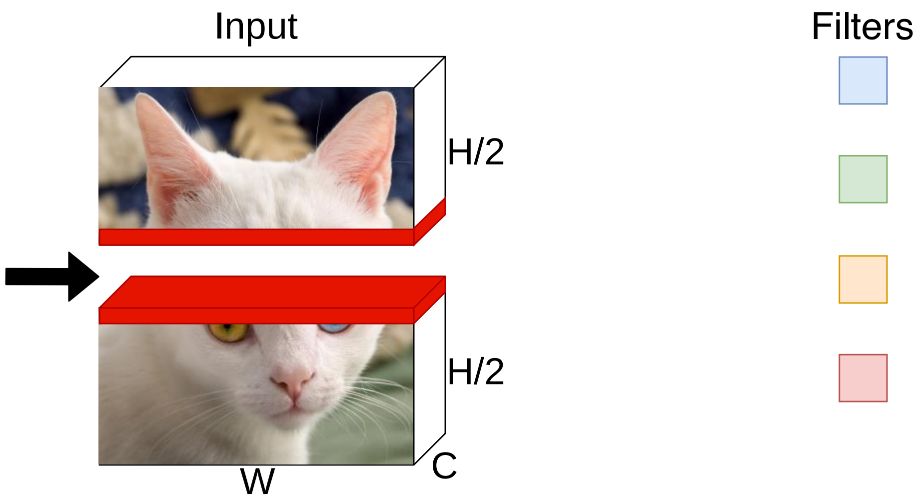


Spatial parallelism [Dryden et al. 2019]

Spatial parallelism [Dryden et al. 2019]

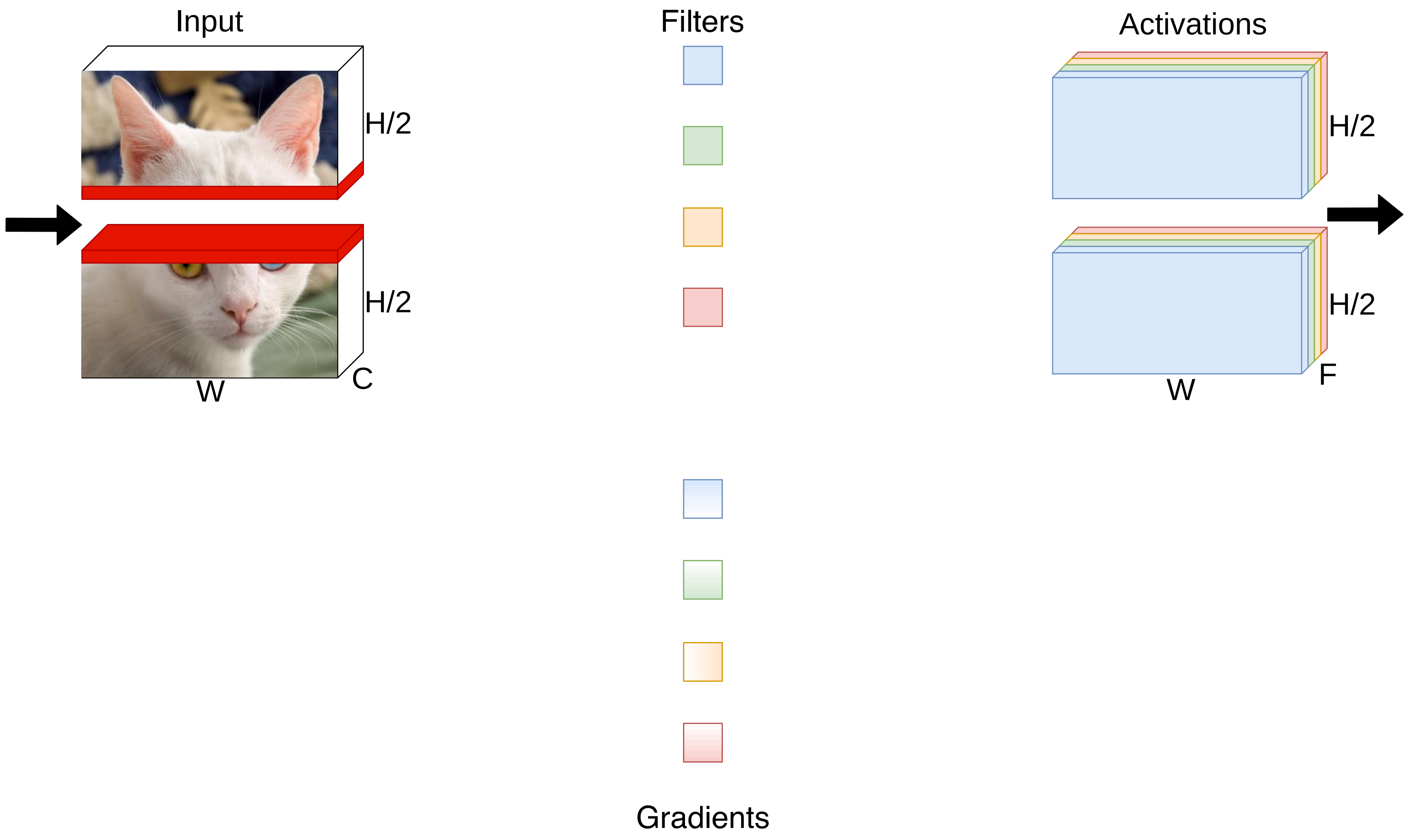


Spatial parallelism [Dryden et al. 2019]

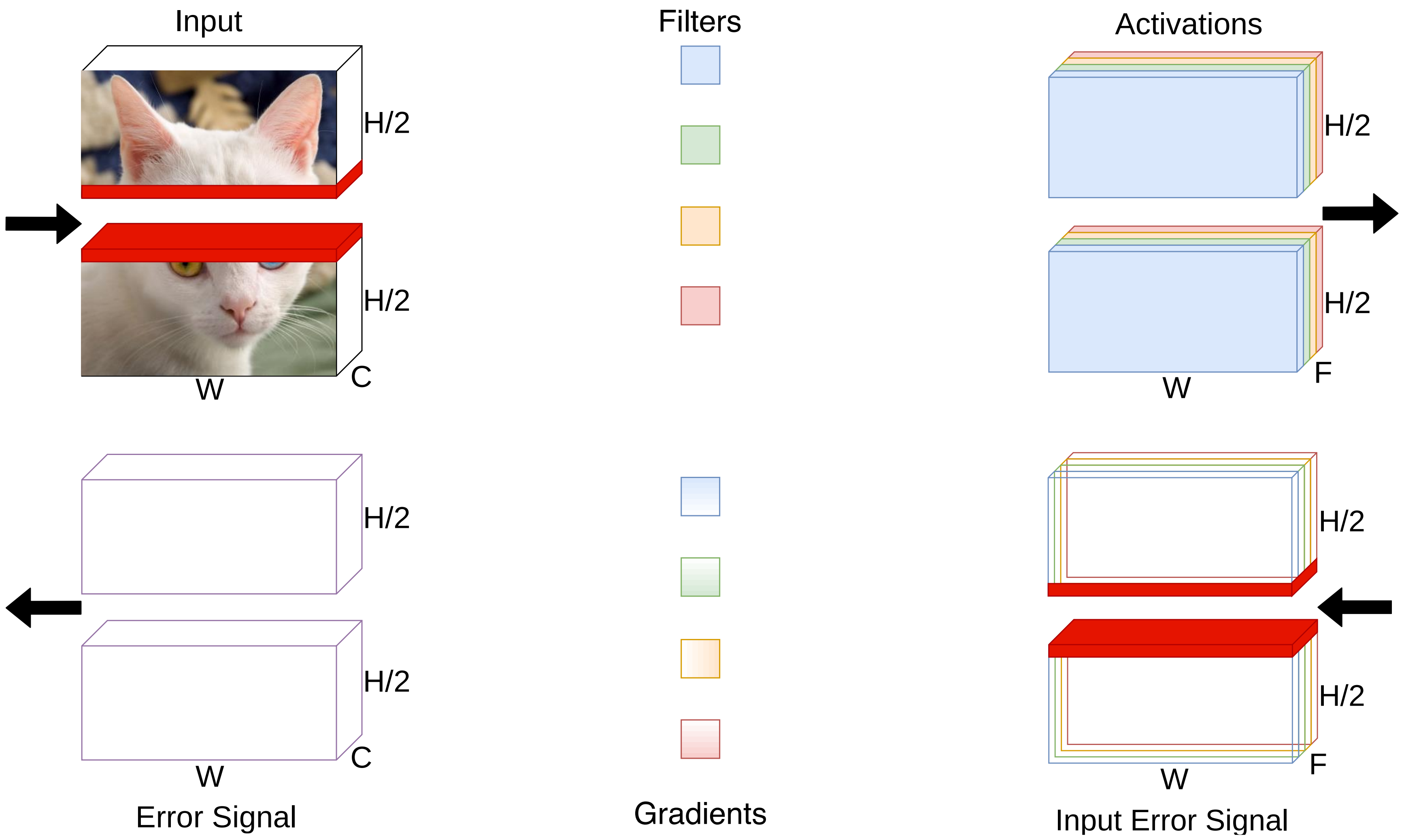


Spatial parallelism [Dryden et al. 2019]

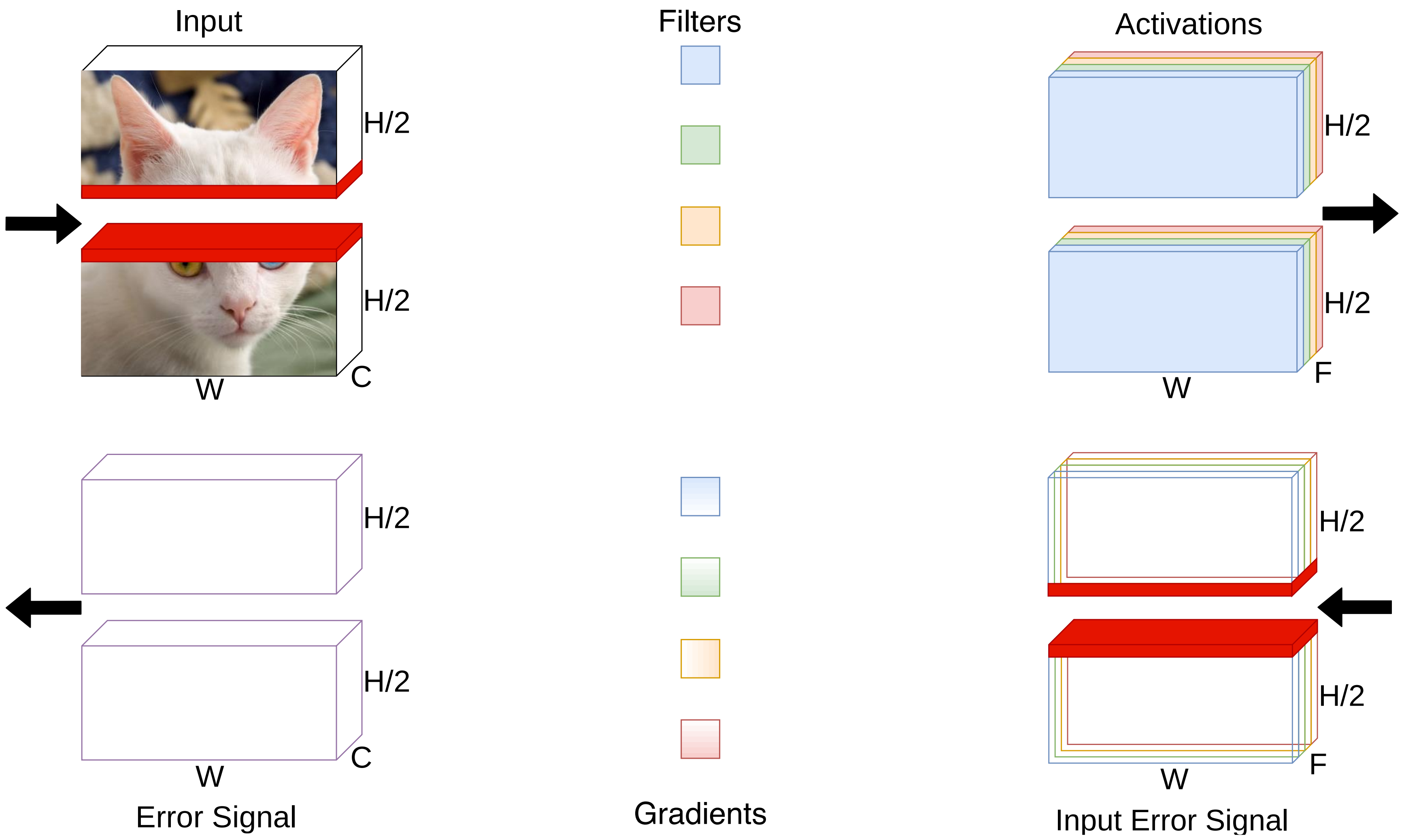
Spatial parallelism [Dryden et al. 2019]



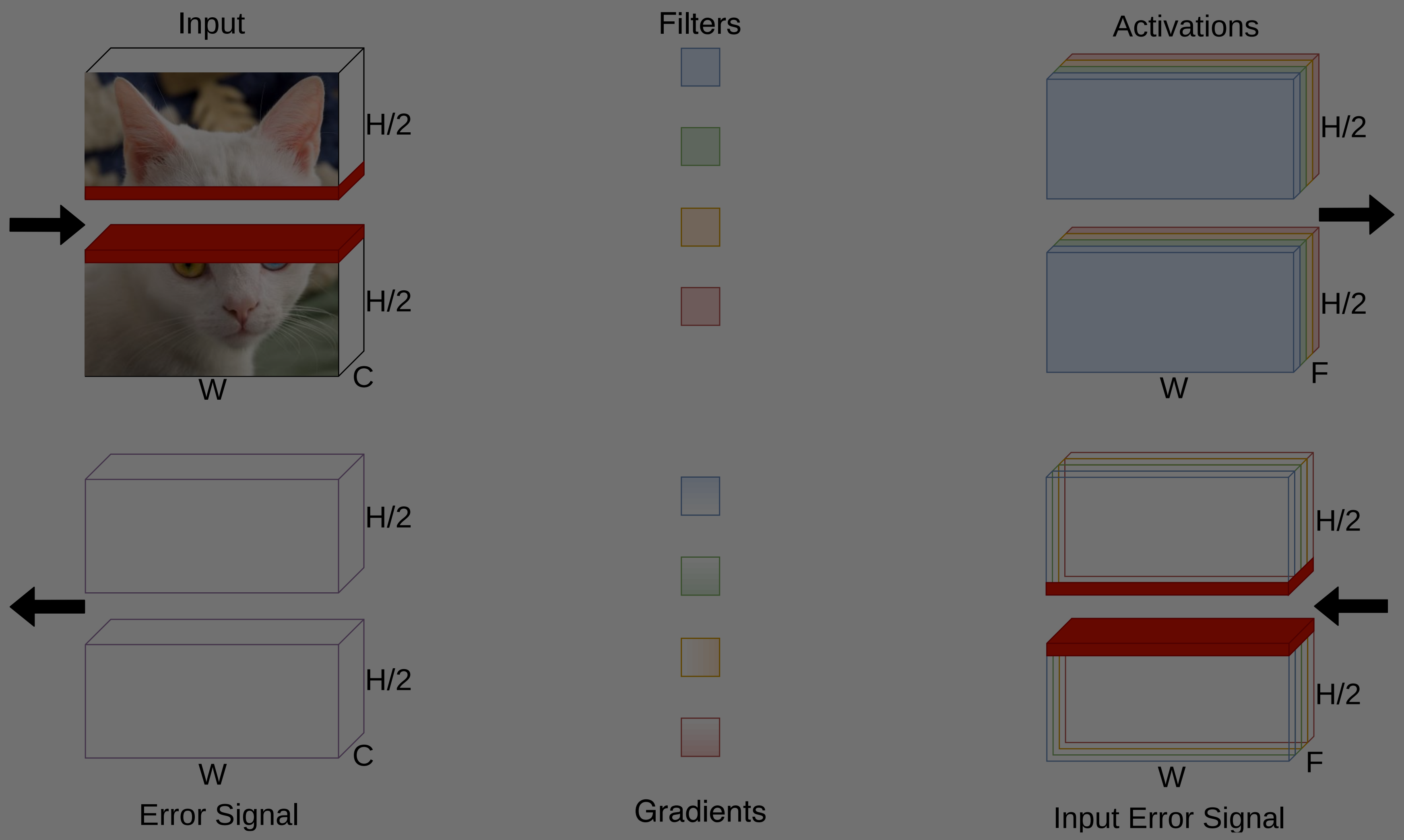
Spatial parallelism [Dryden et al. 2019]



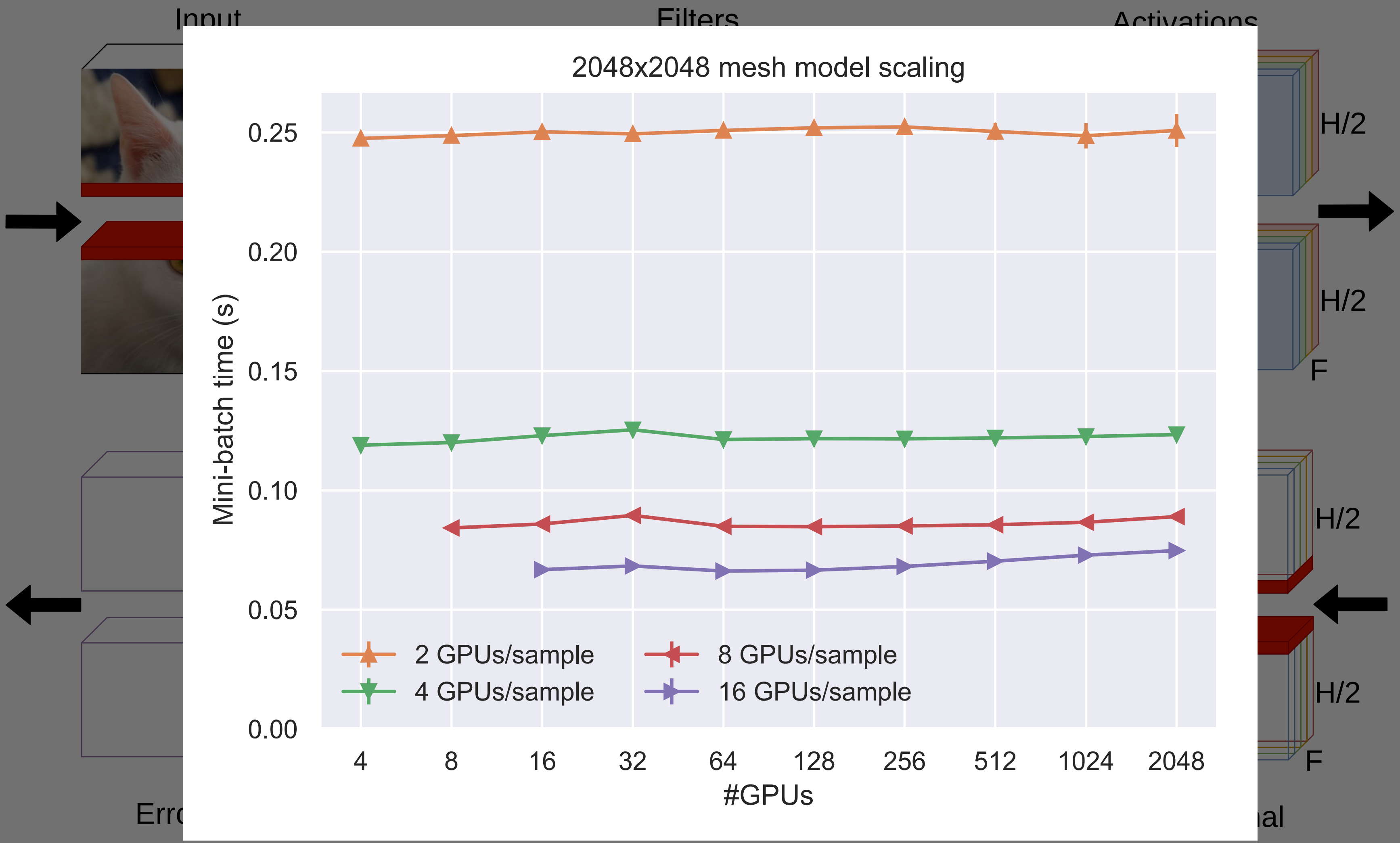
Spatial parallelism [Dryden et al. 2019]



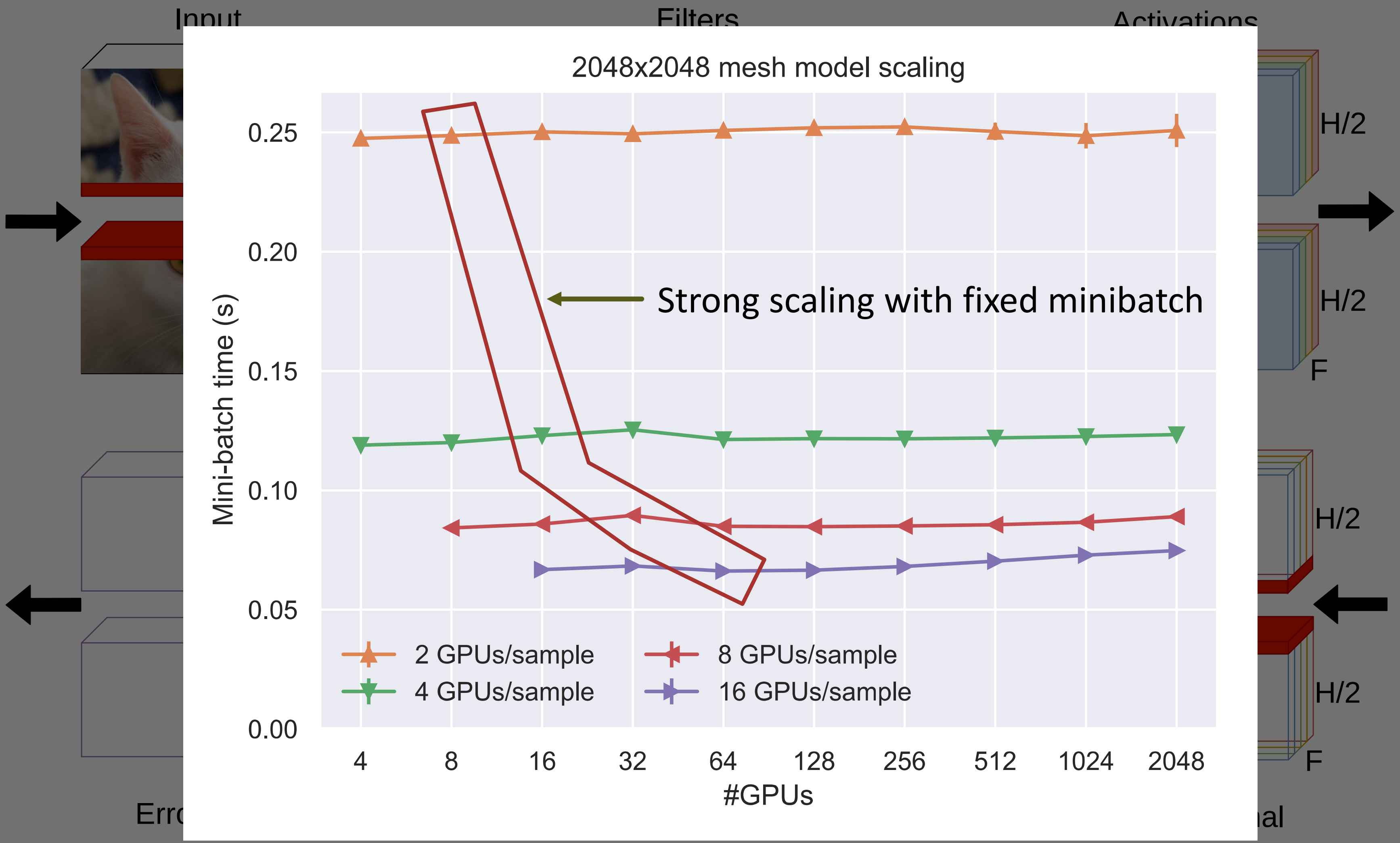
Spatial parallelism [Dryden et al. 2019]



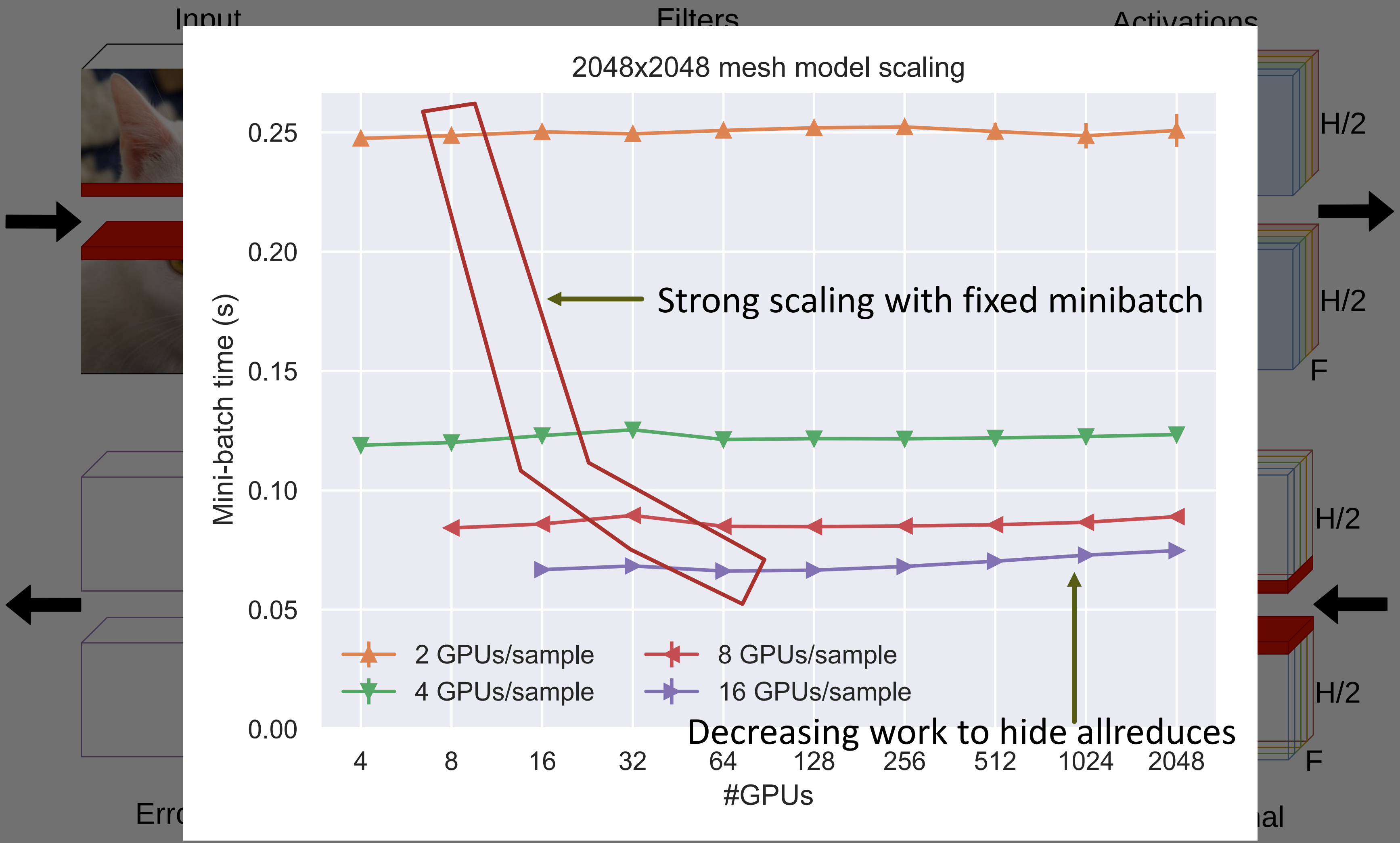
Spatial parallelism [Dryden et al. 2019]



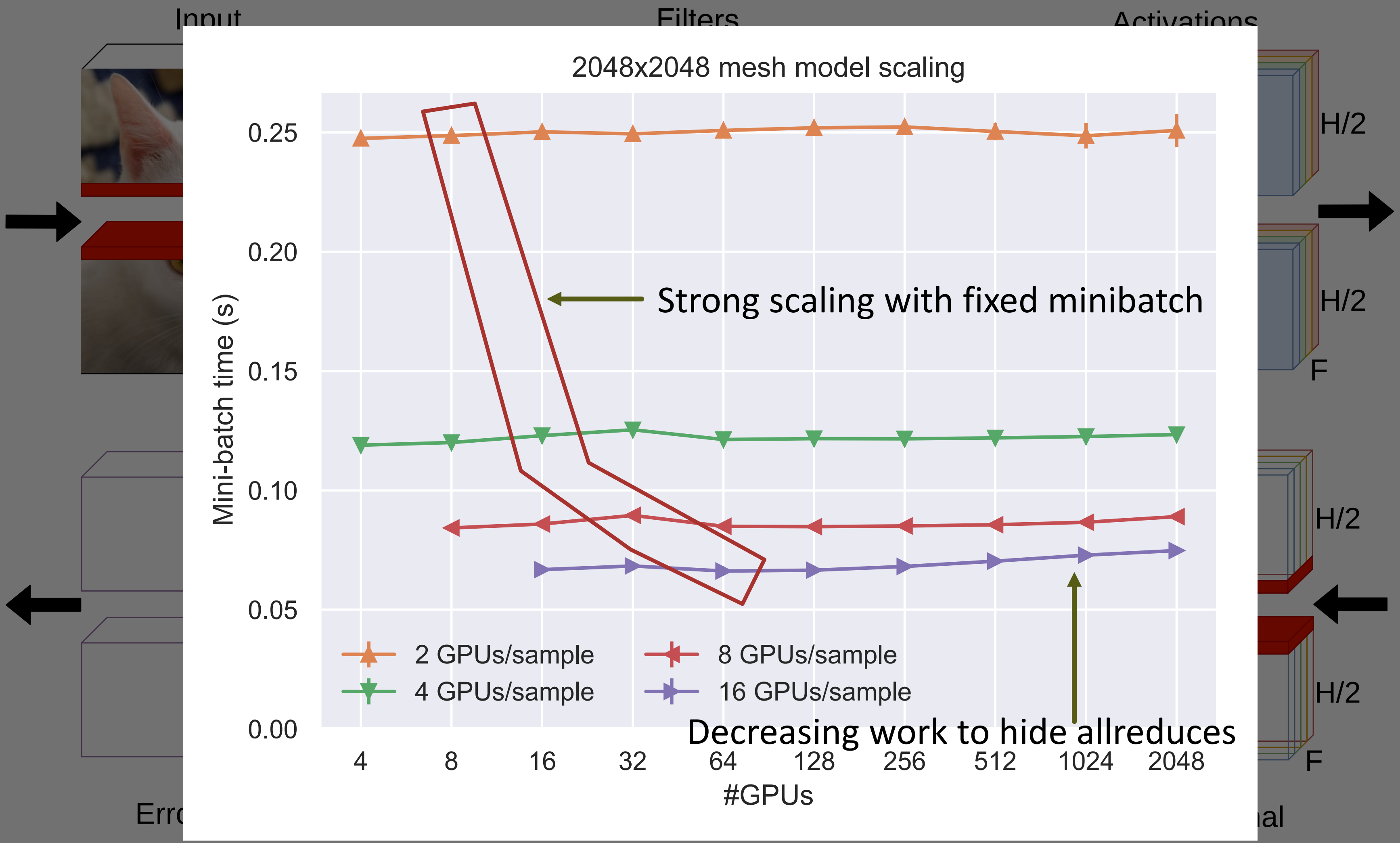
Spatial parallelism [Dryden et al. 2019]



Spatial parallelism [Dryden et al. 2019]

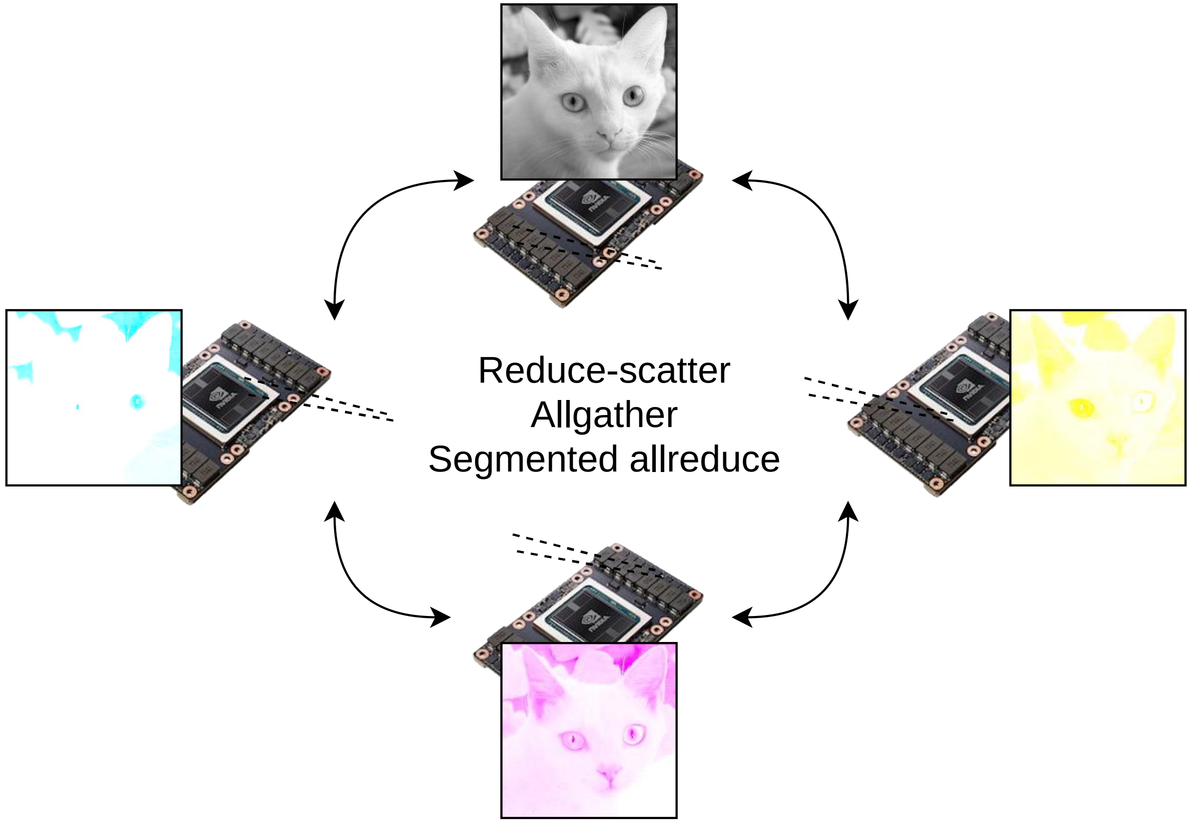


Spatial parallelism [Dryden et al. 2019]

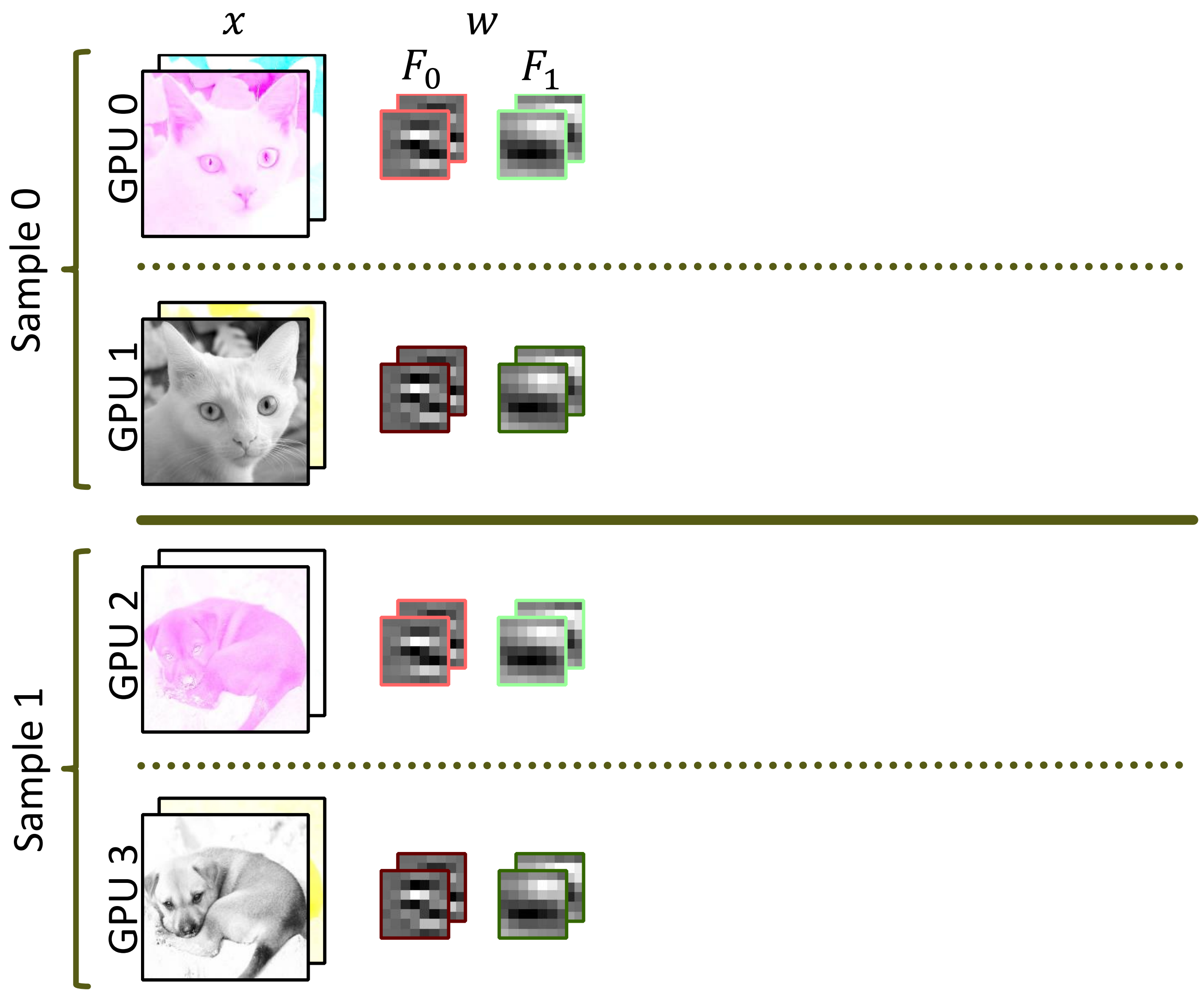


Channel/filter parallelism [Dryden et al. 2019]

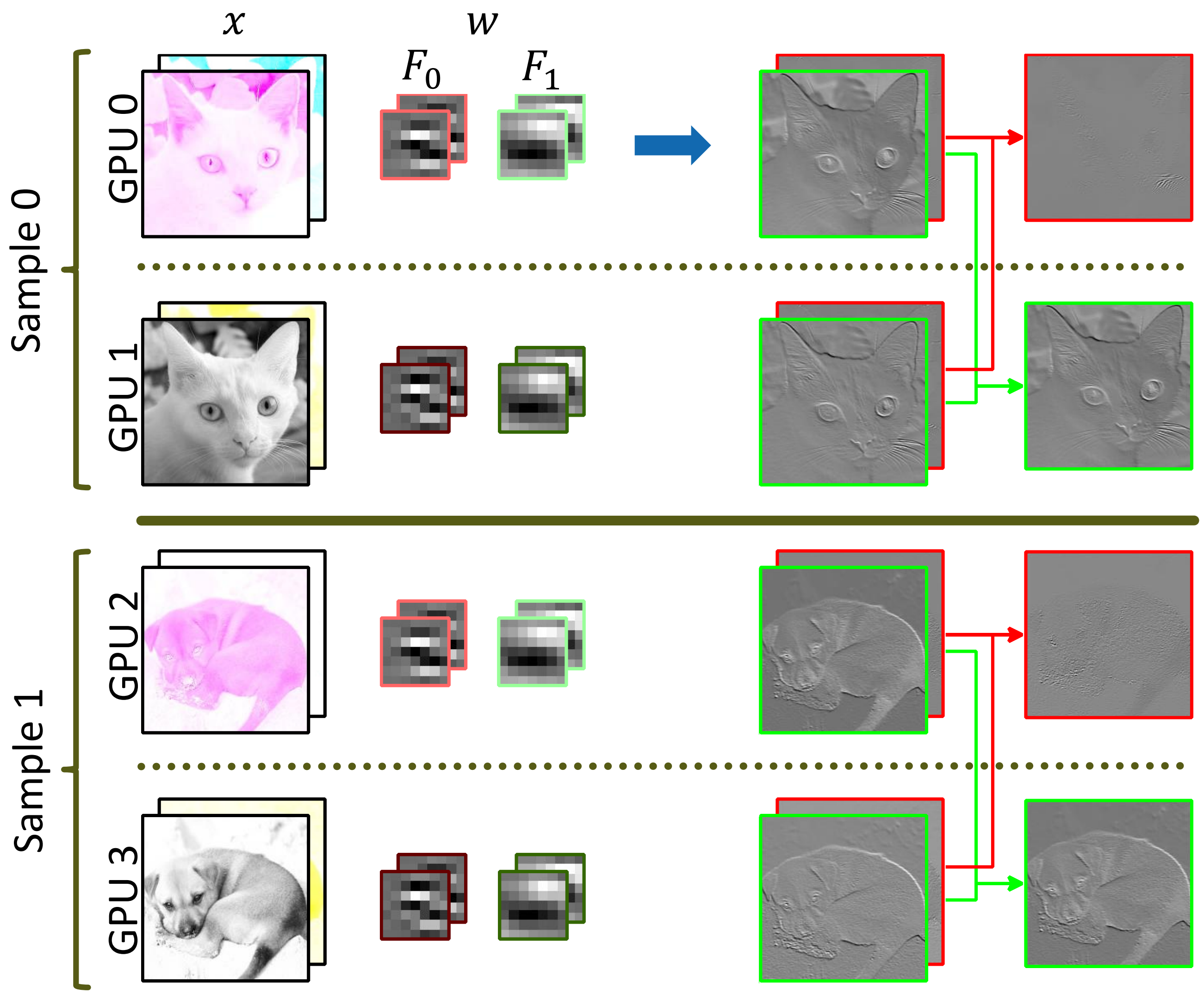
- **Family of algorithms for jointly partitioning channels and filters in convolution**



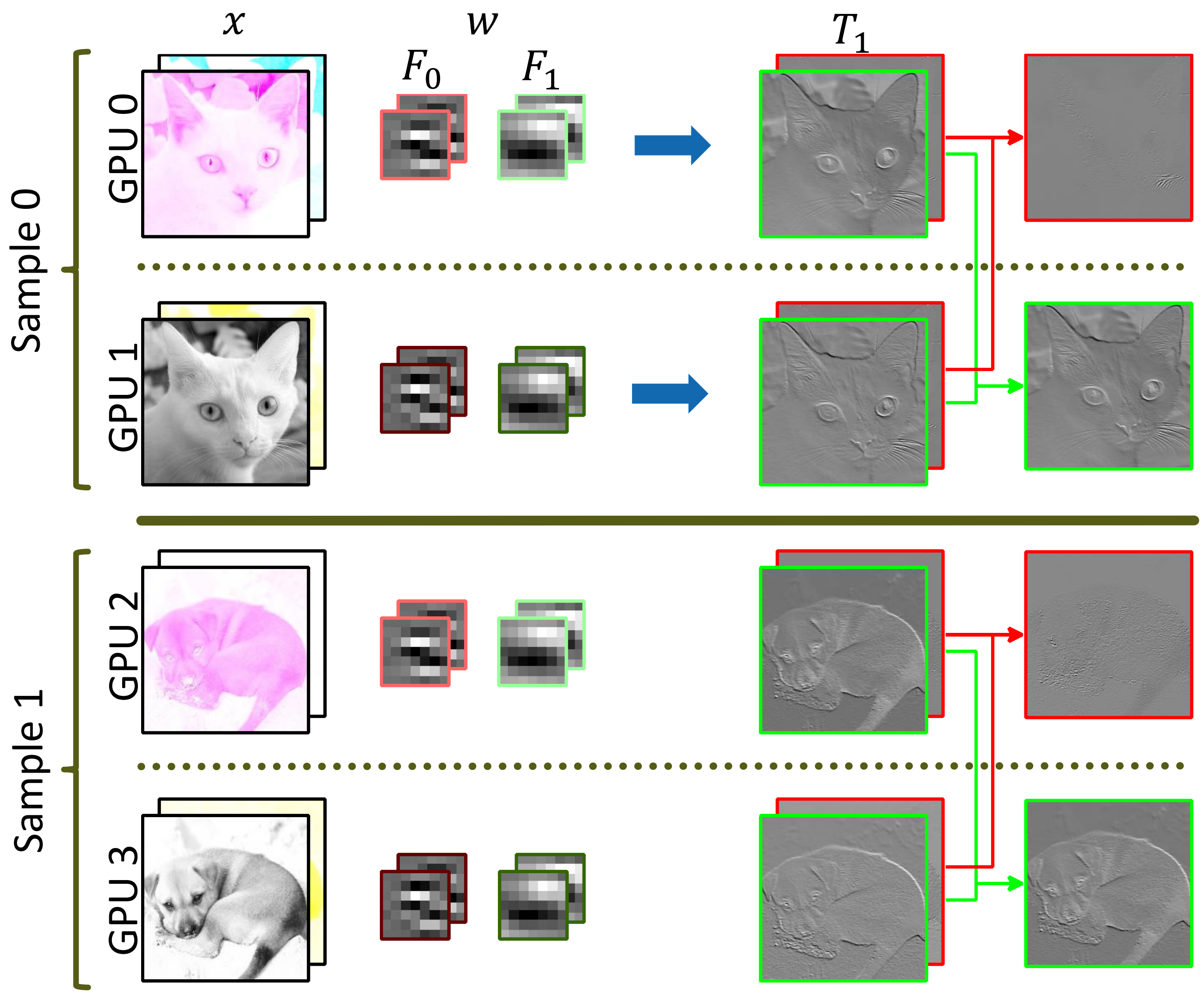
Stationary-x: Forward



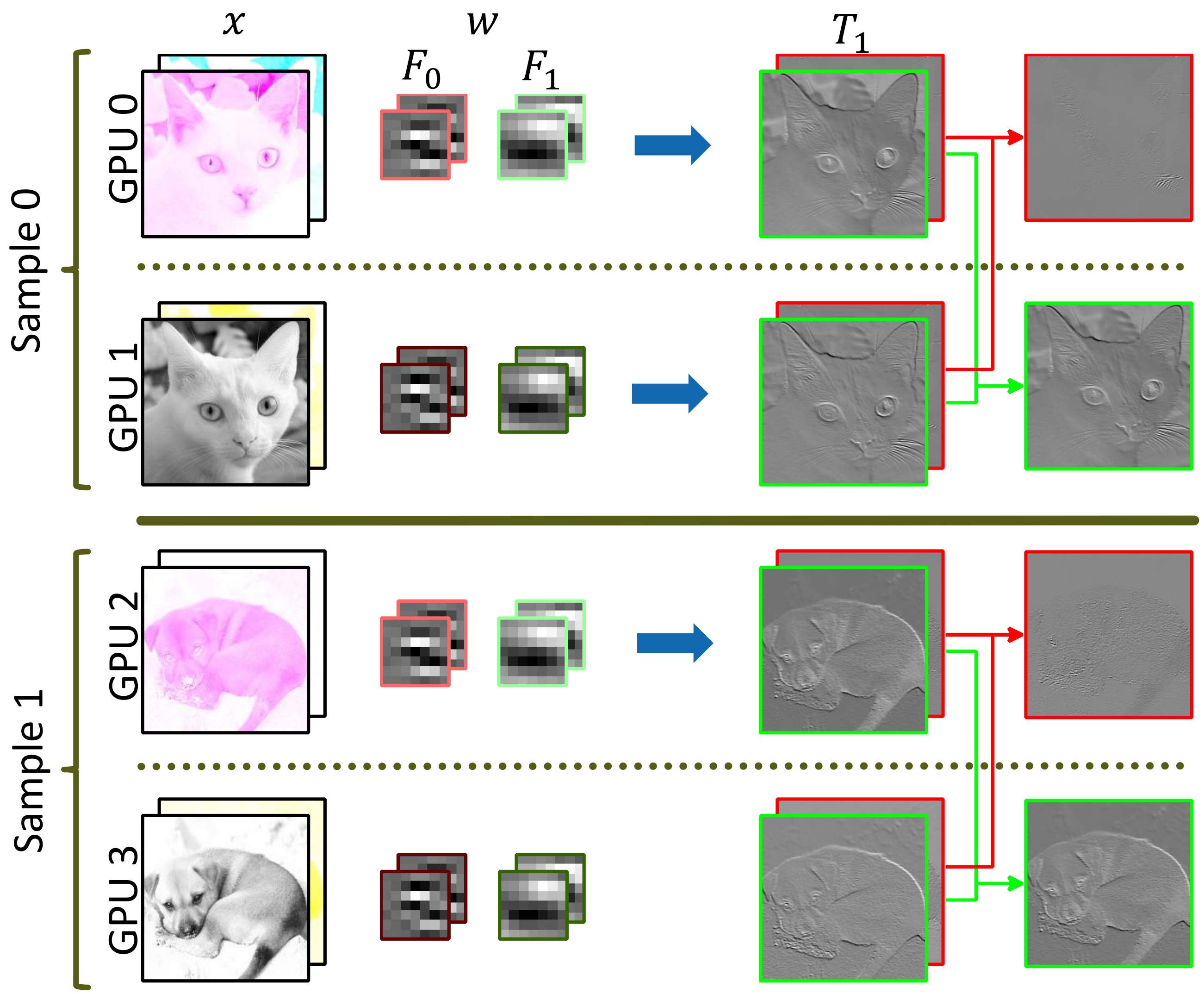
Stationary-x: Forward



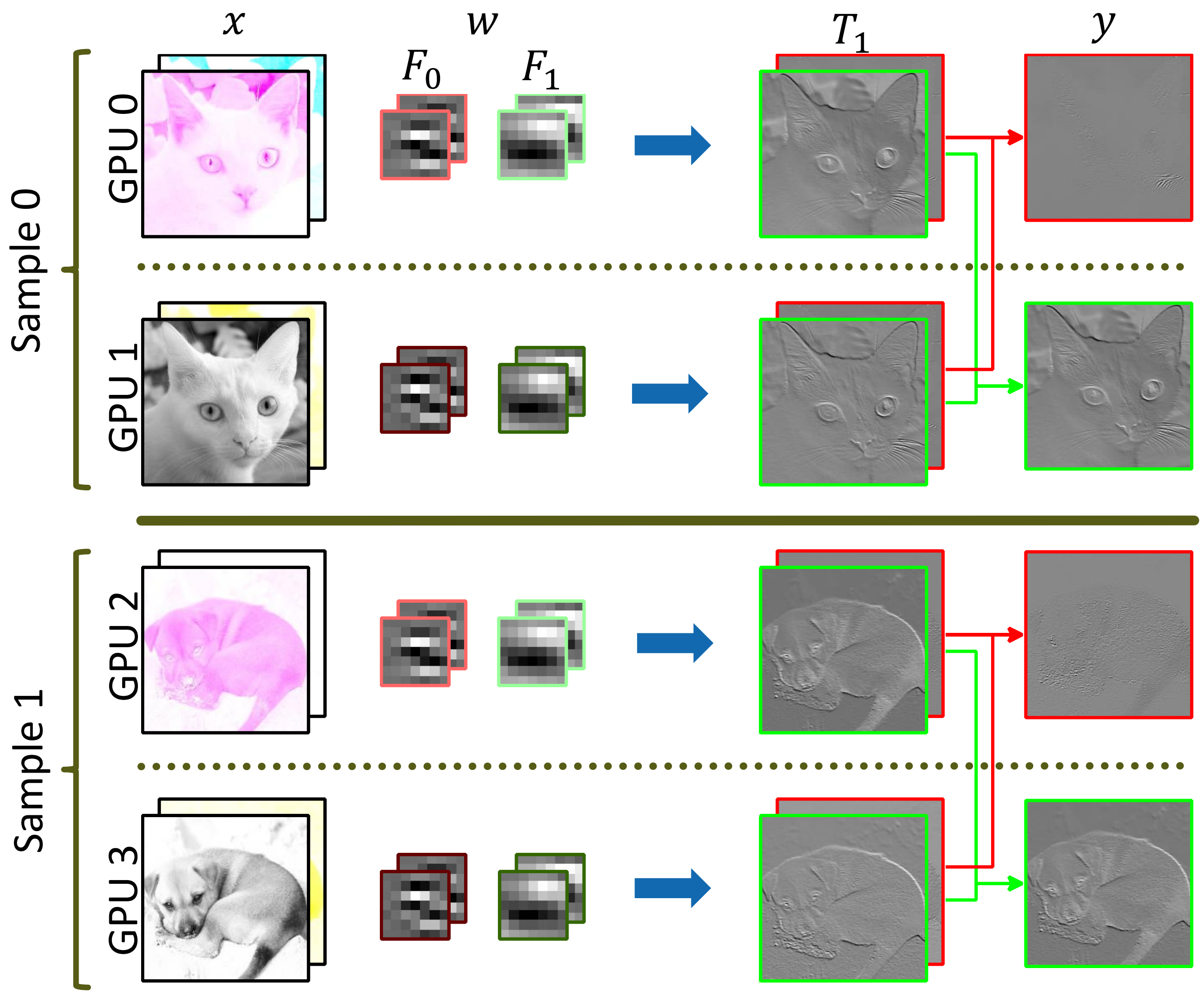
Stationary-x: Forward



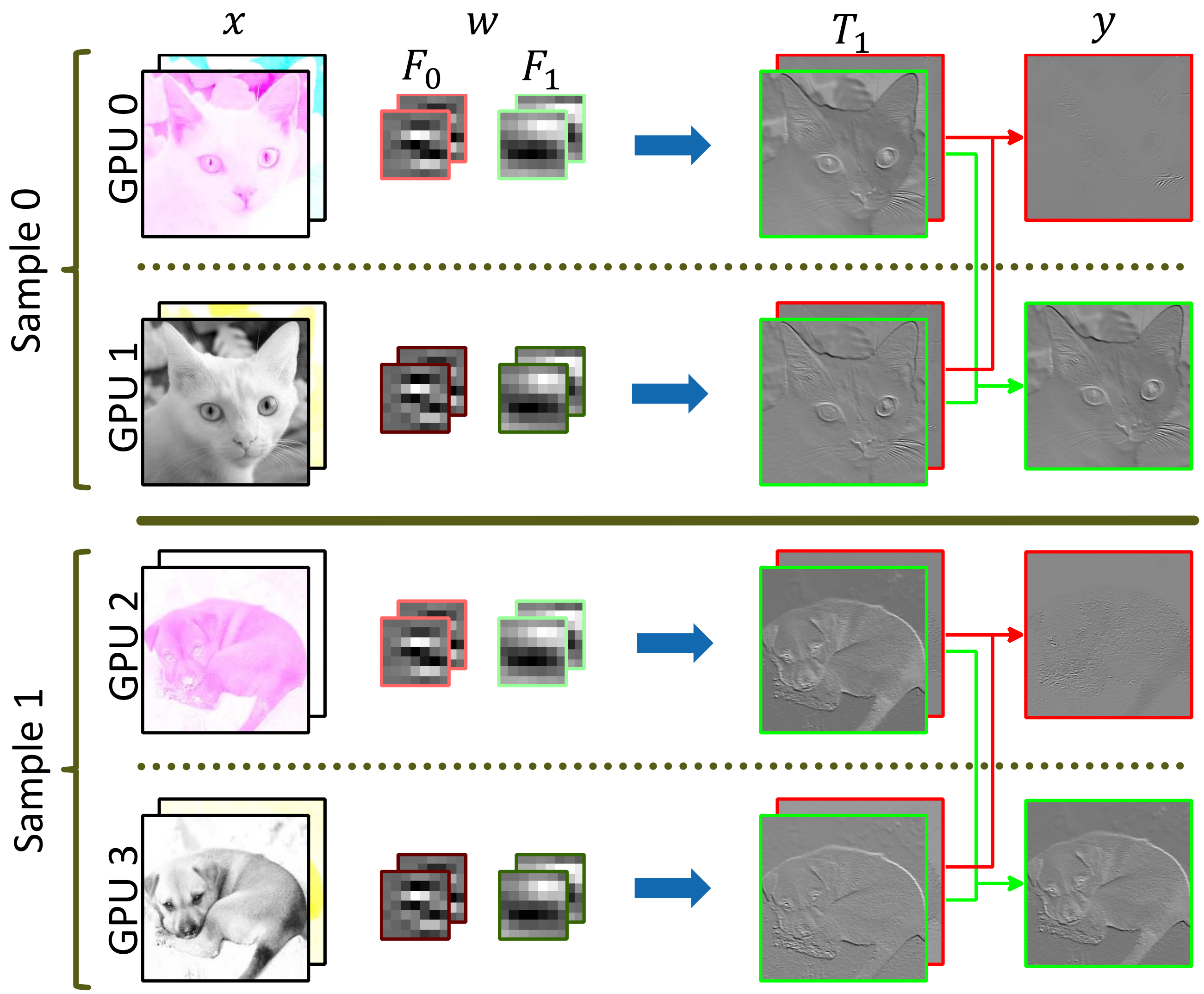
Stationary-x: Forward



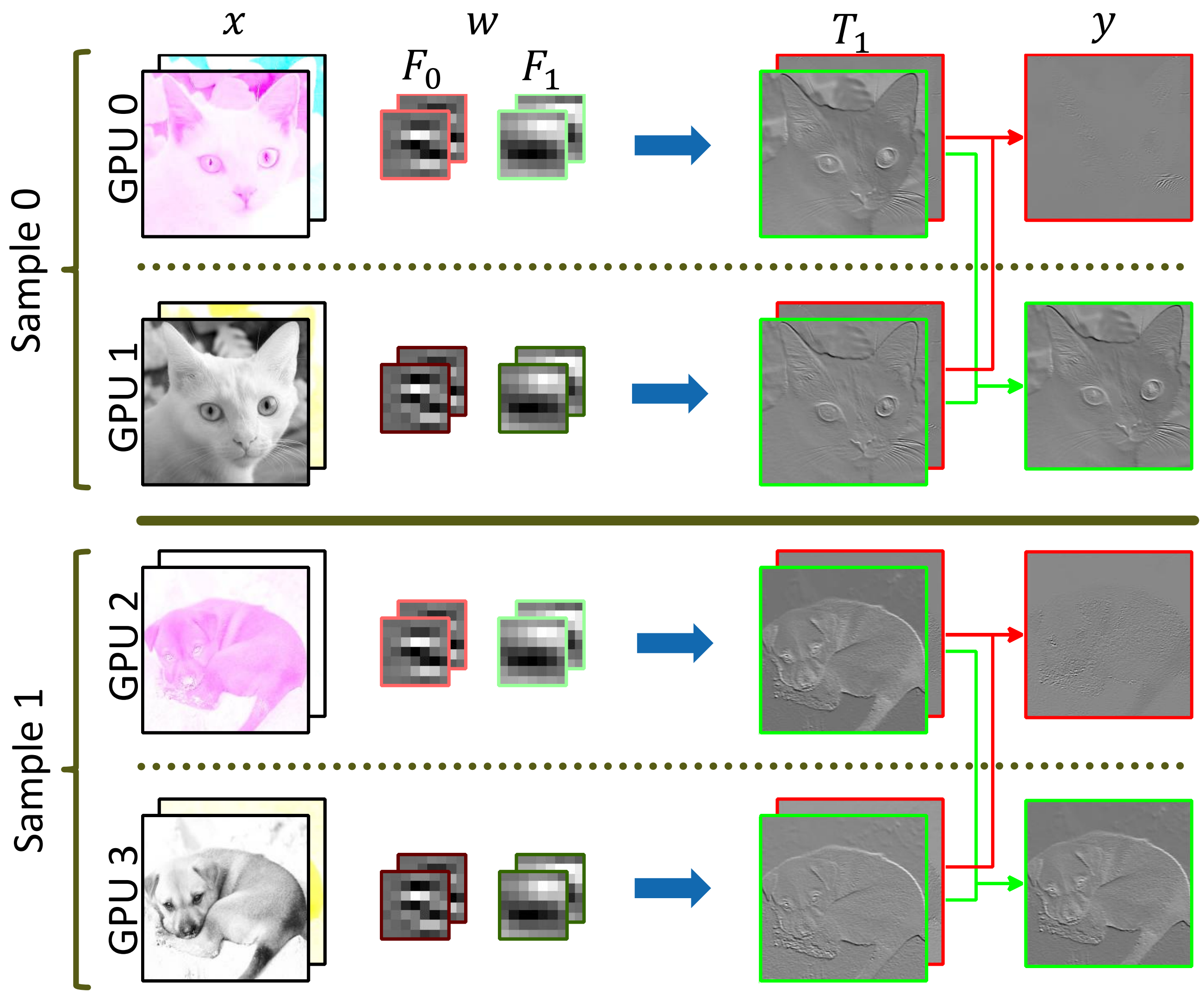
Stationary-x: Forward



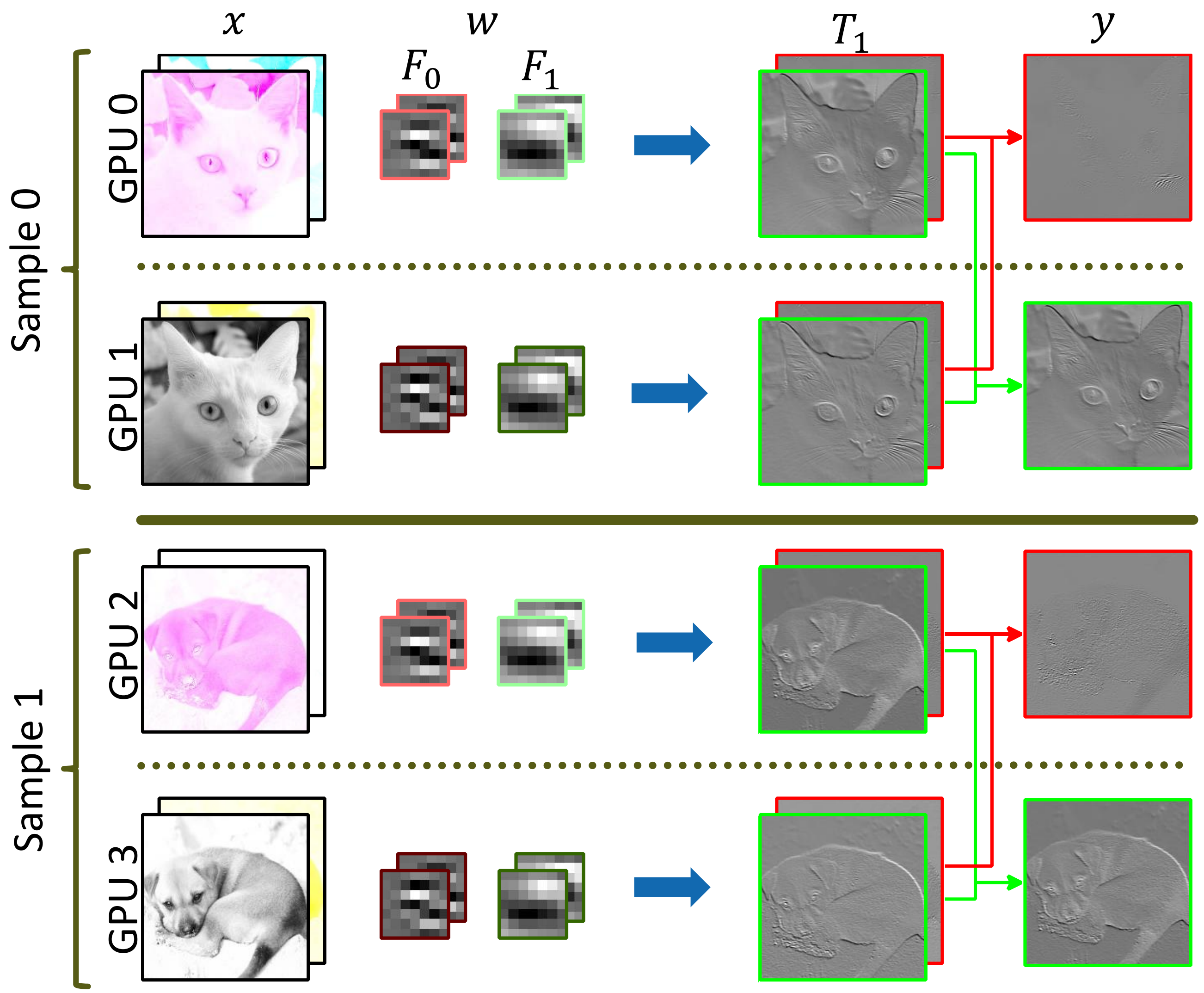
Stationary-x: Forward



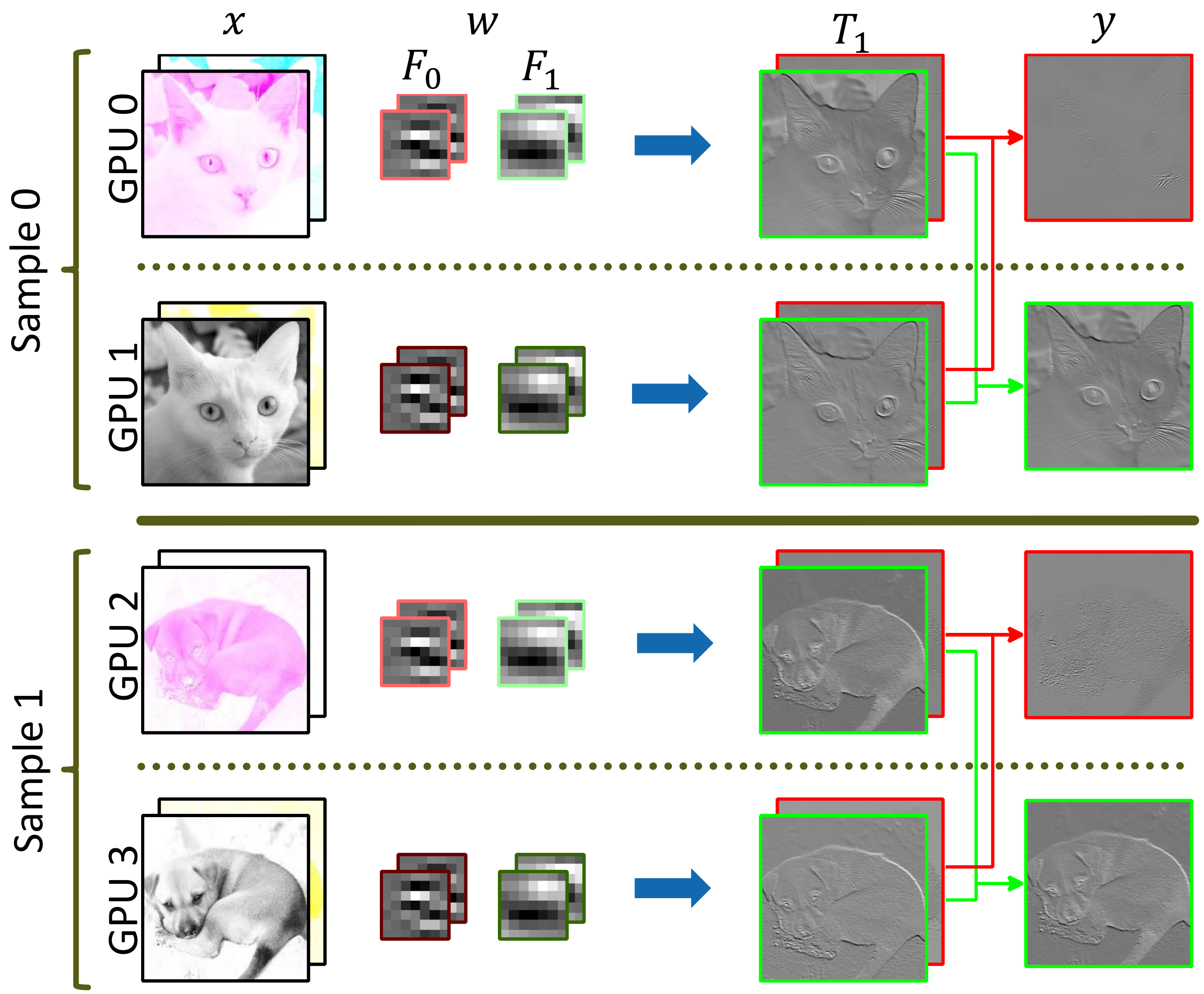
Stationary-x: Forward



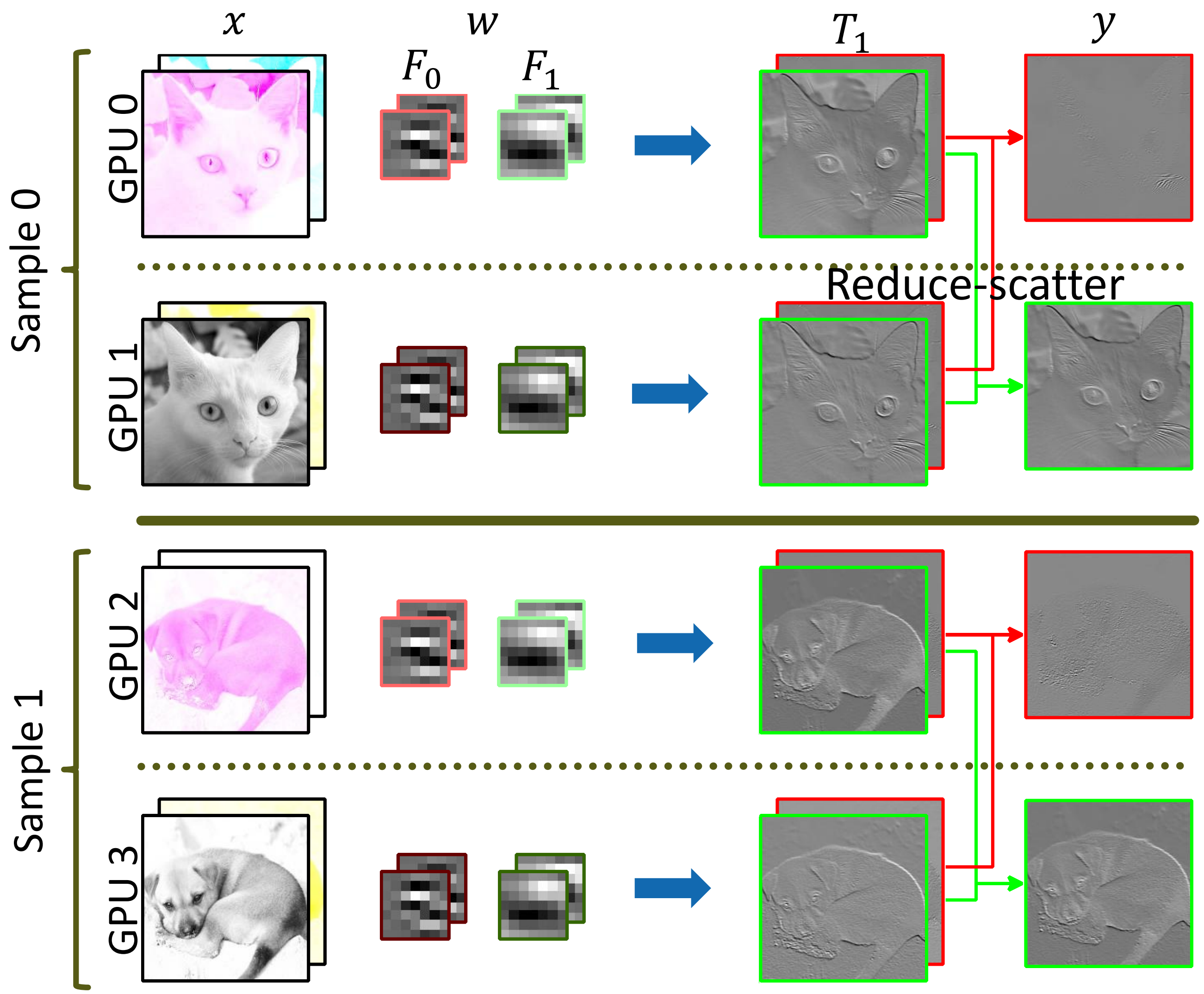
Stationary-x: Forward



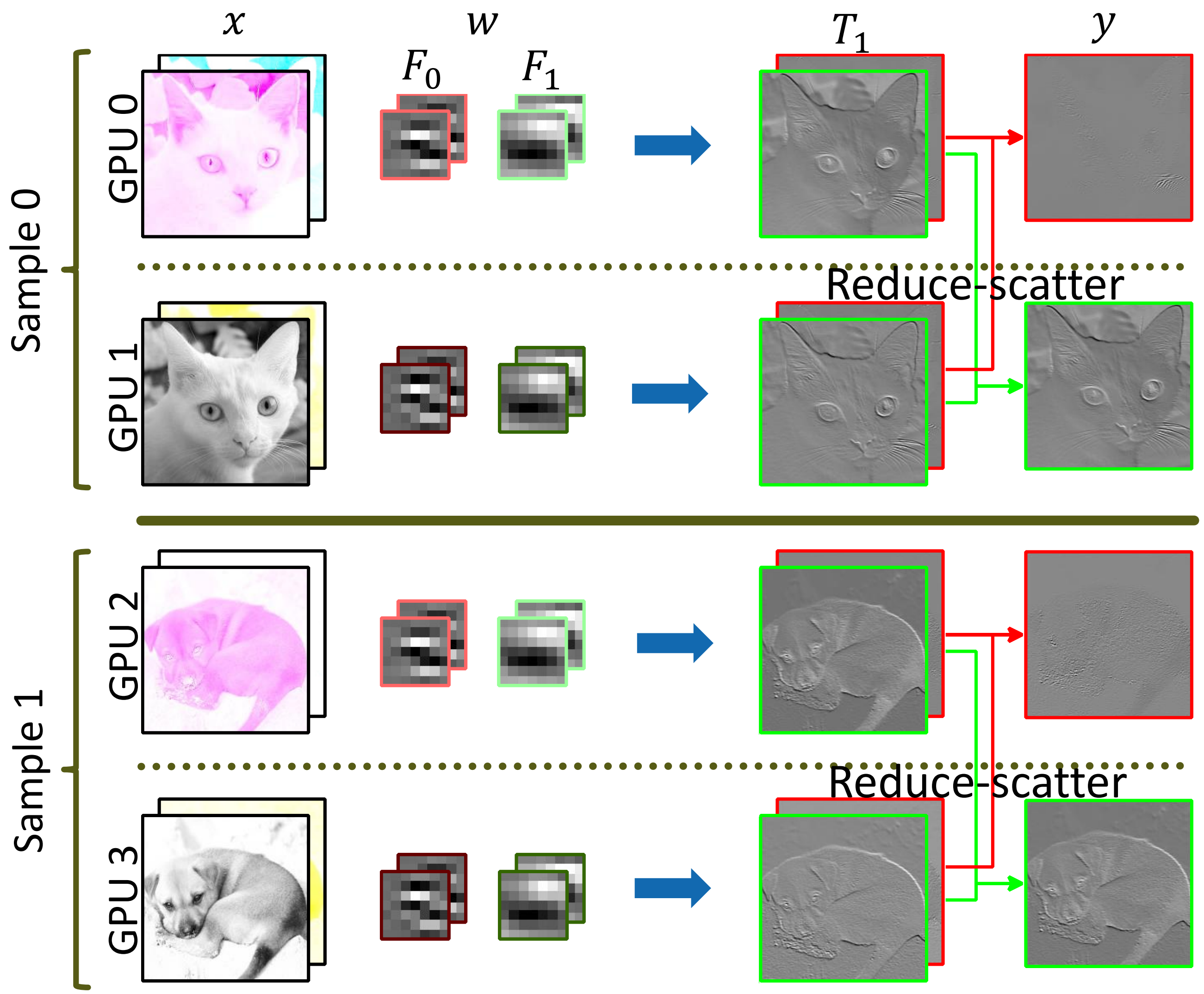
Stationary-x: Forward



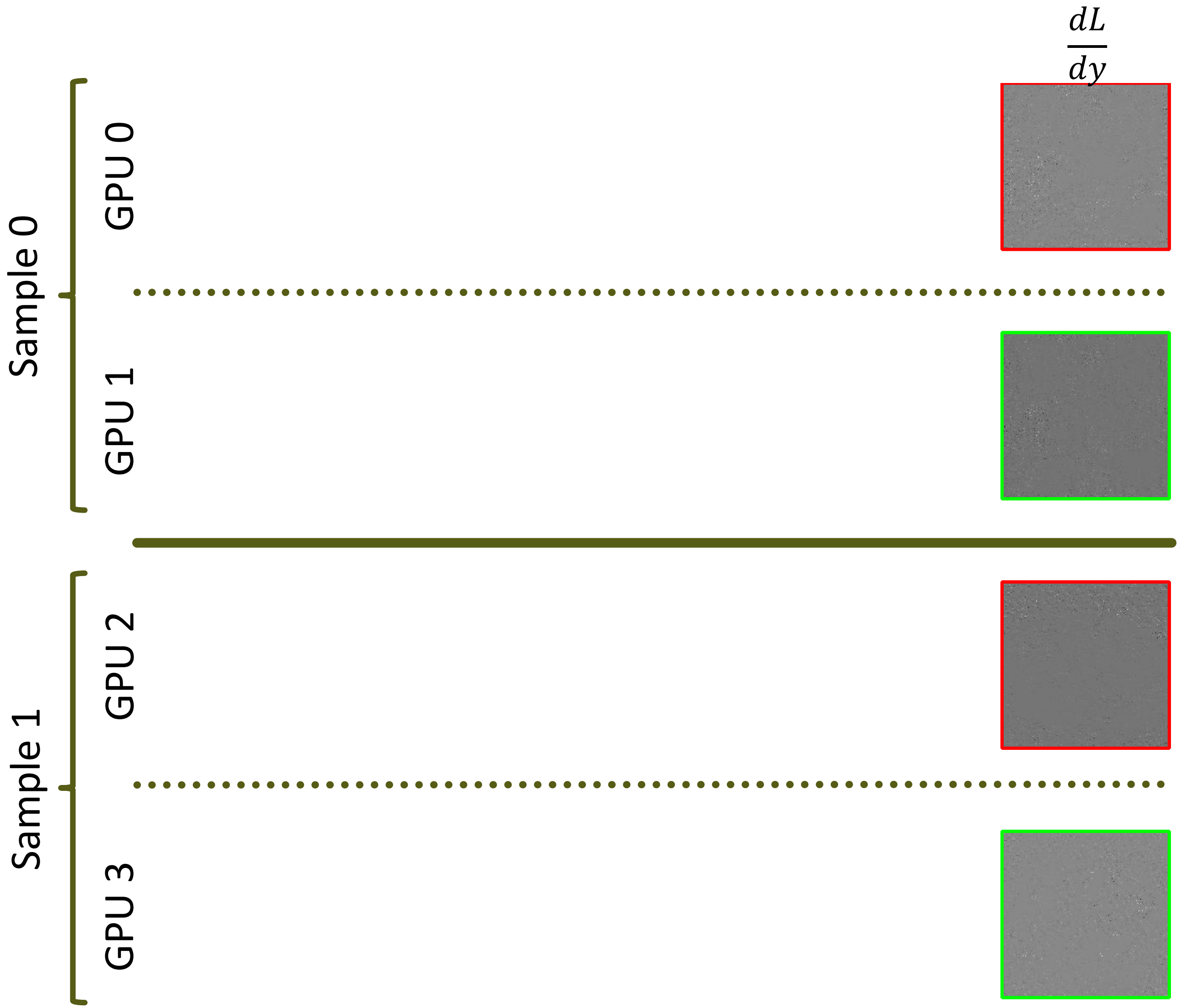
Stationary-x: Forward



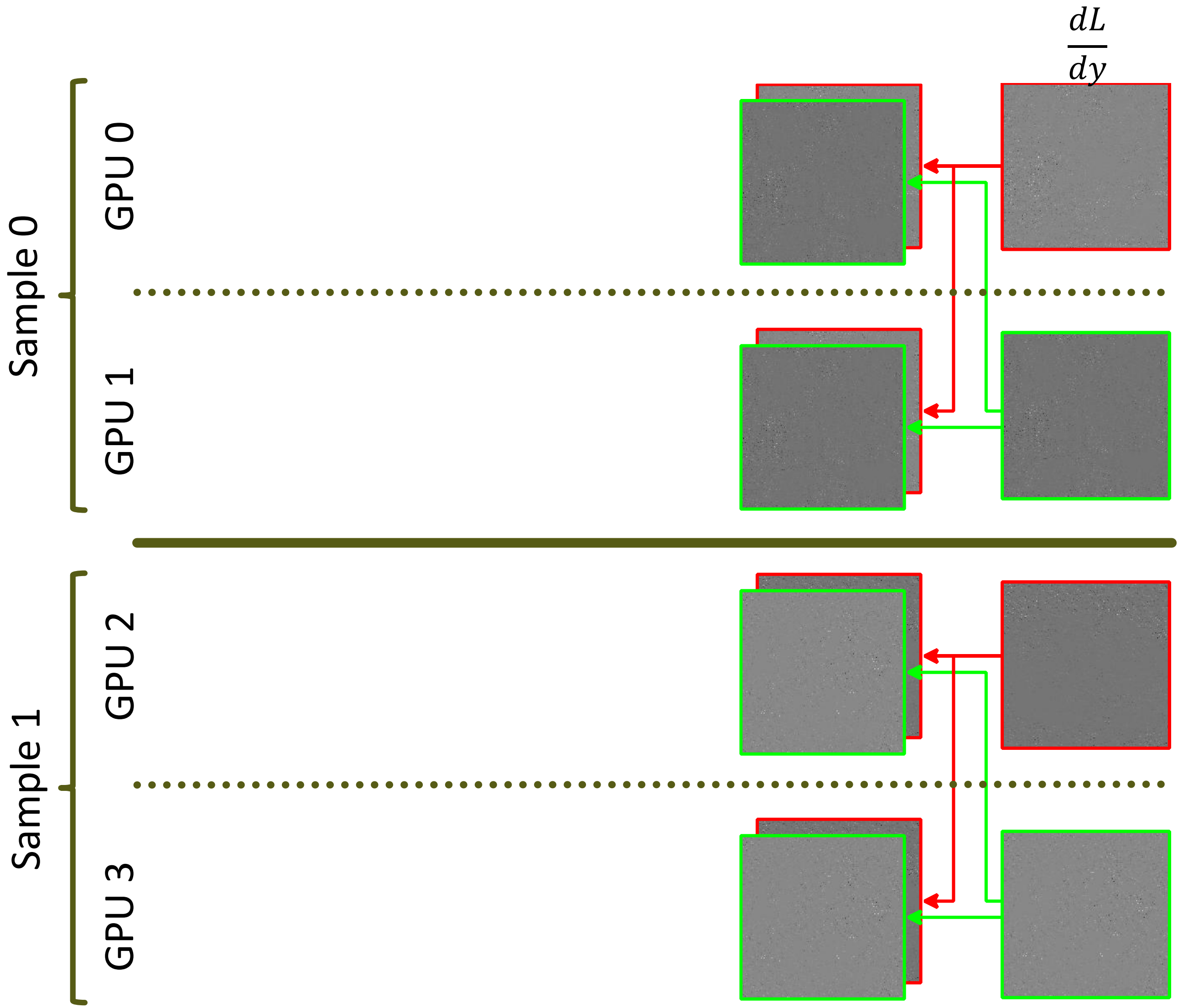
Stationary-x: Forward



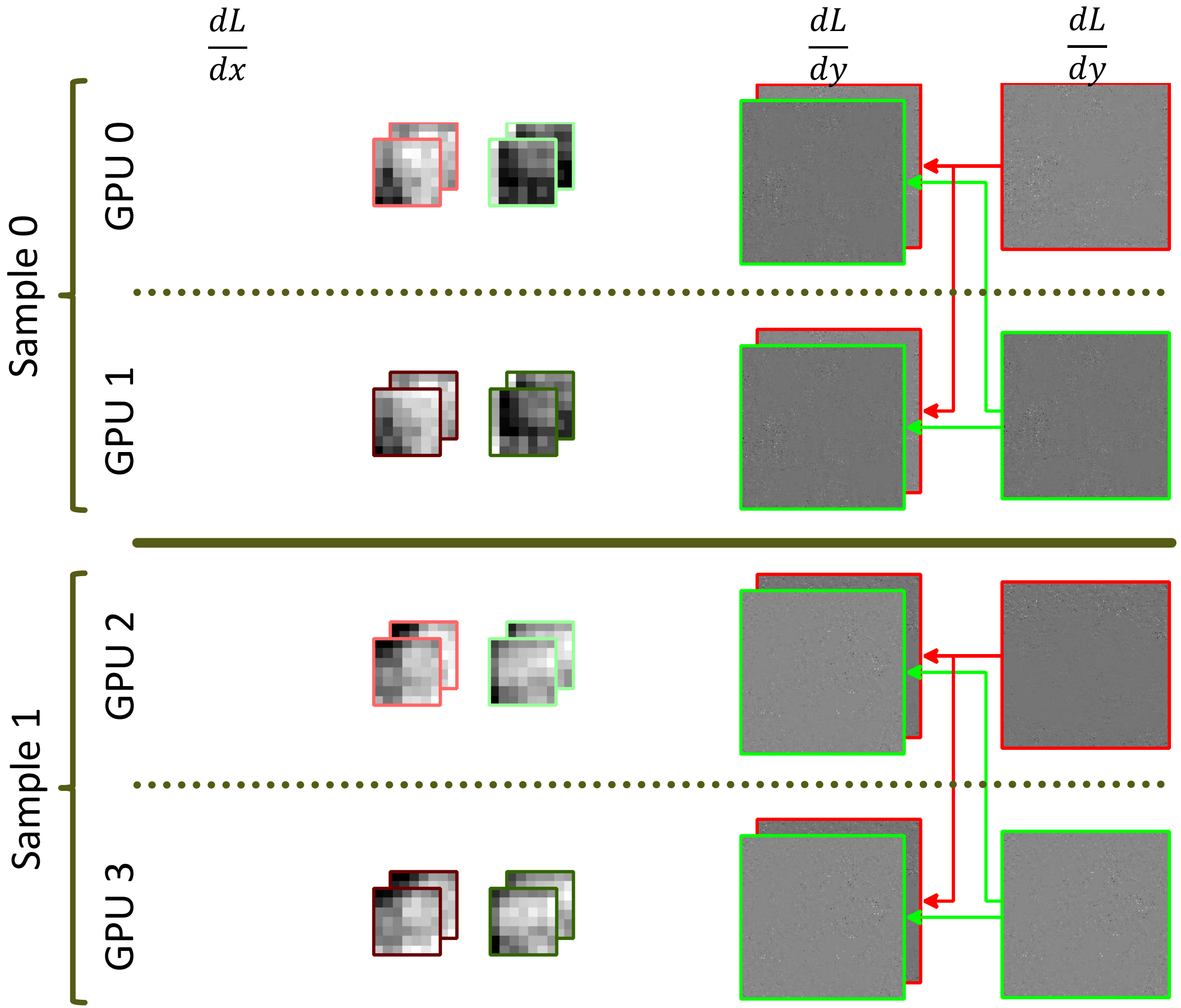
Stationary-x: Backward



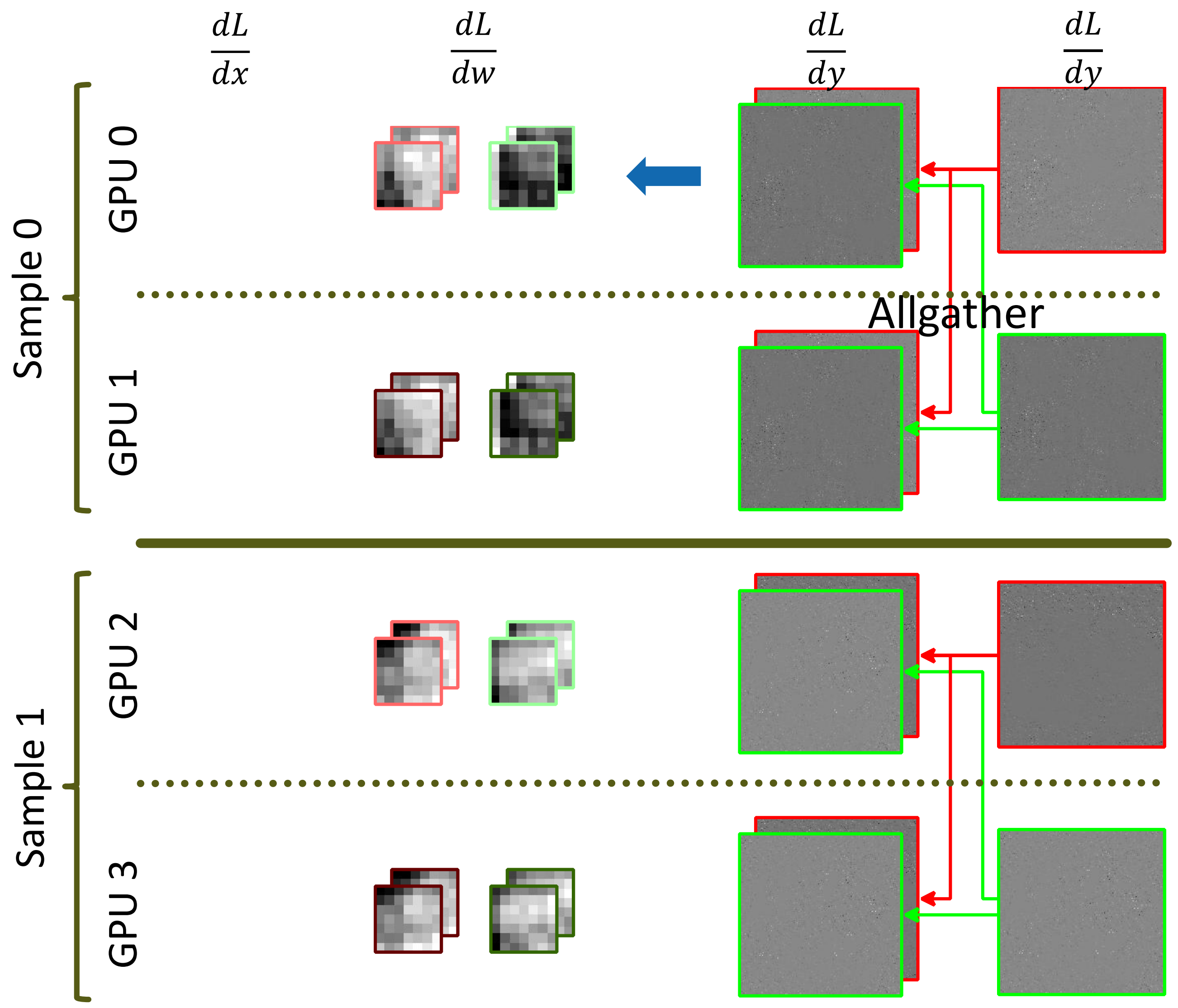
Stationary-x: Backward



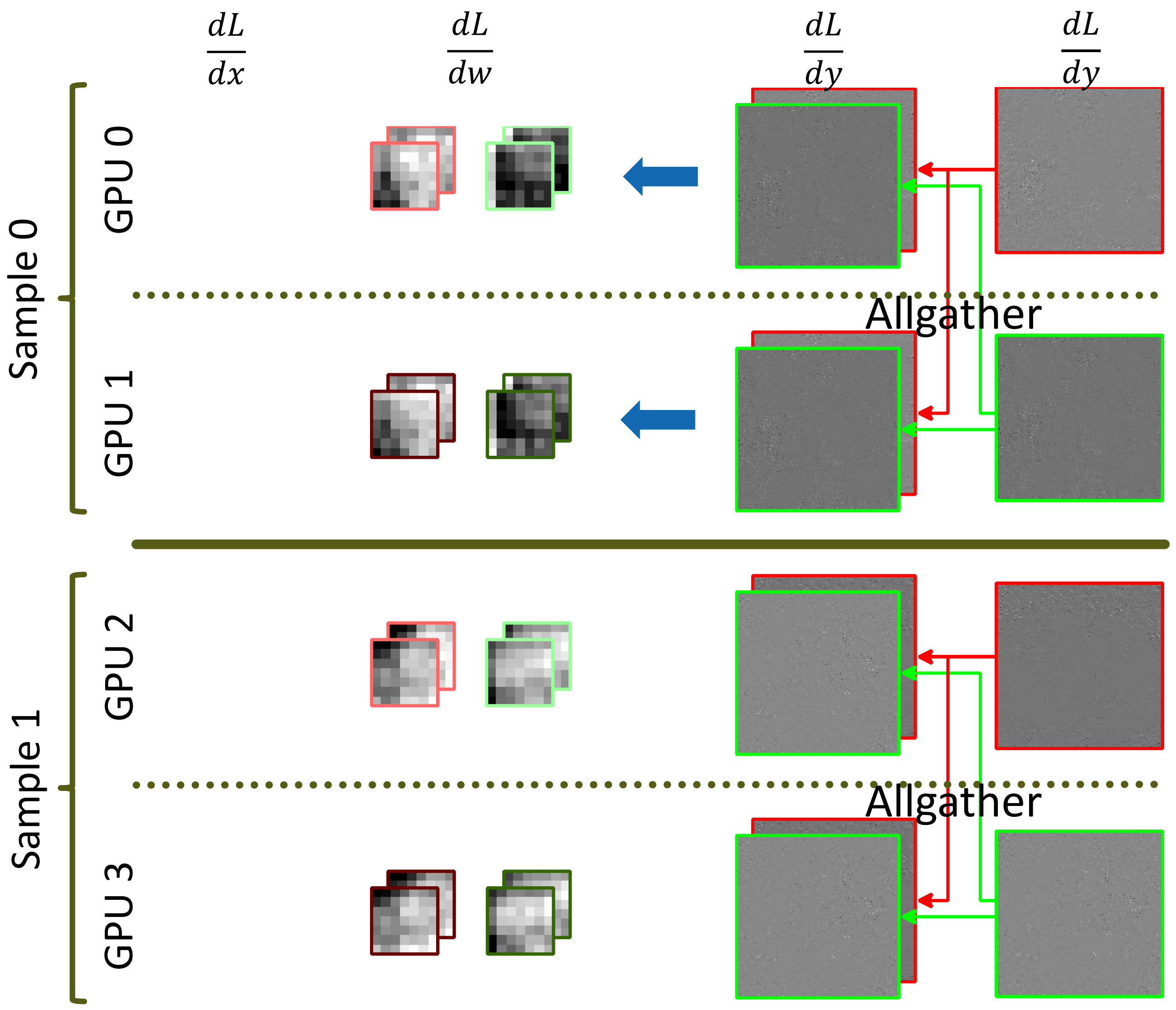
Stationary-x: Backward



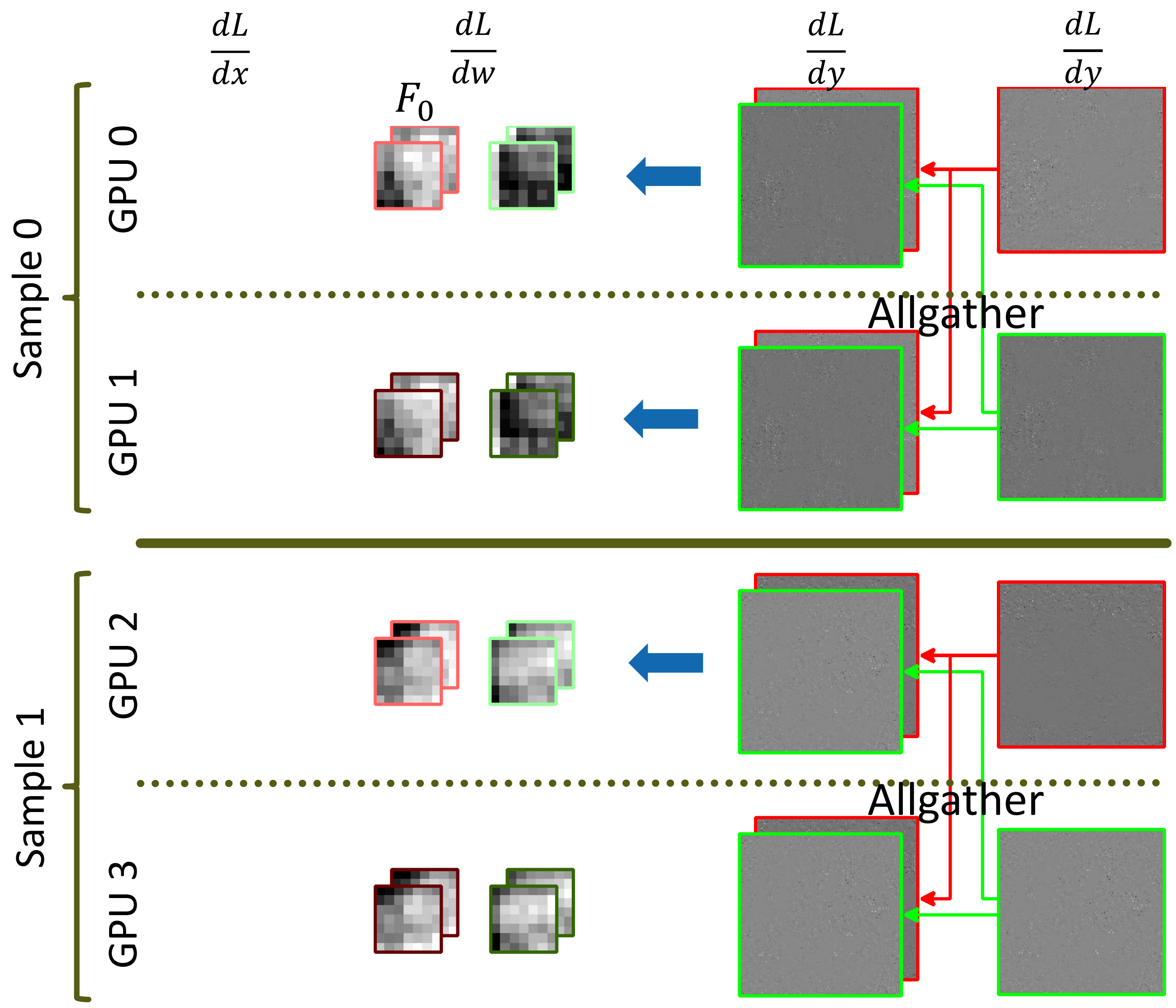
Stationary-x: Backward



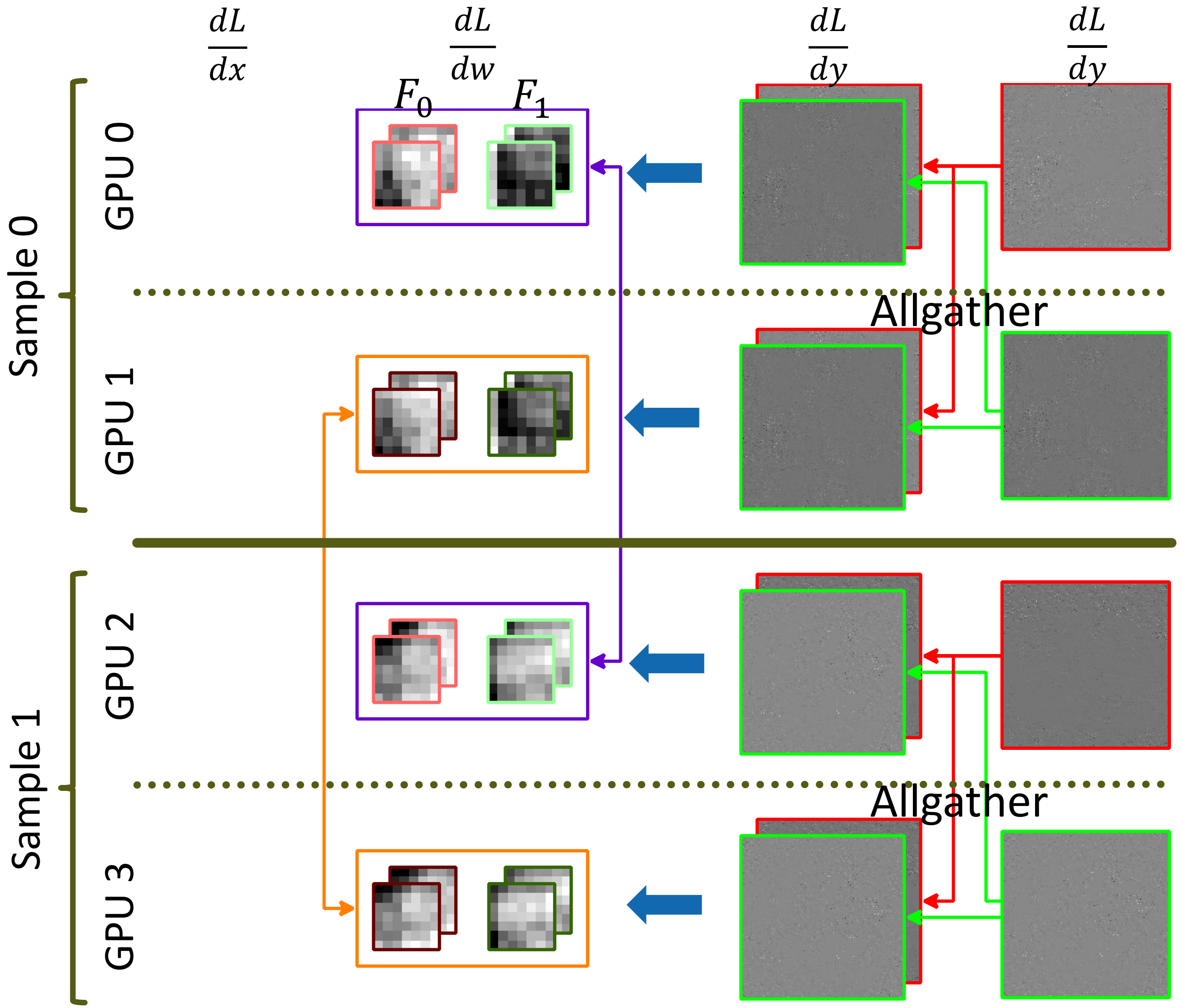
Stationary-x: Backward



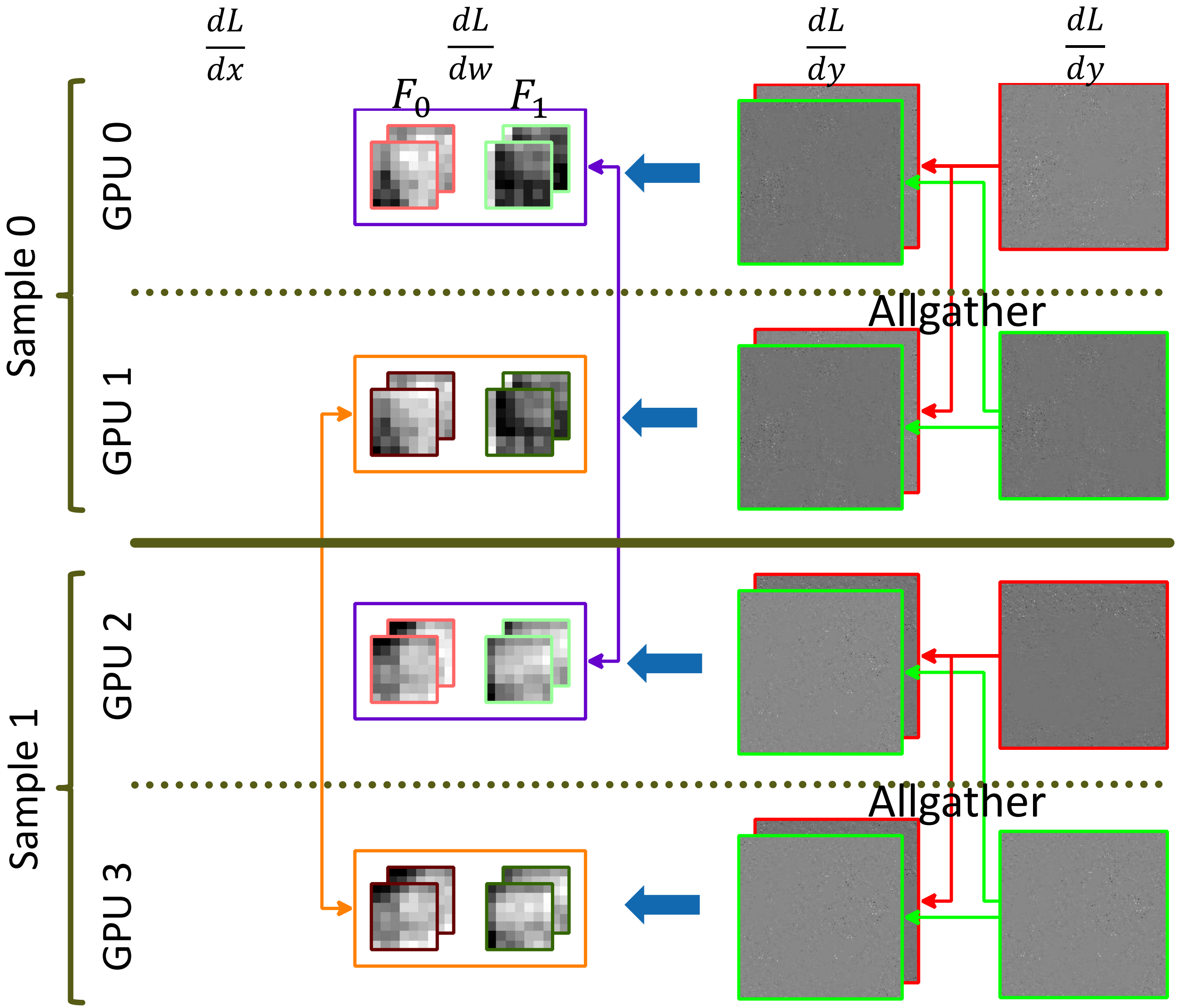
Stationary-x: Backward



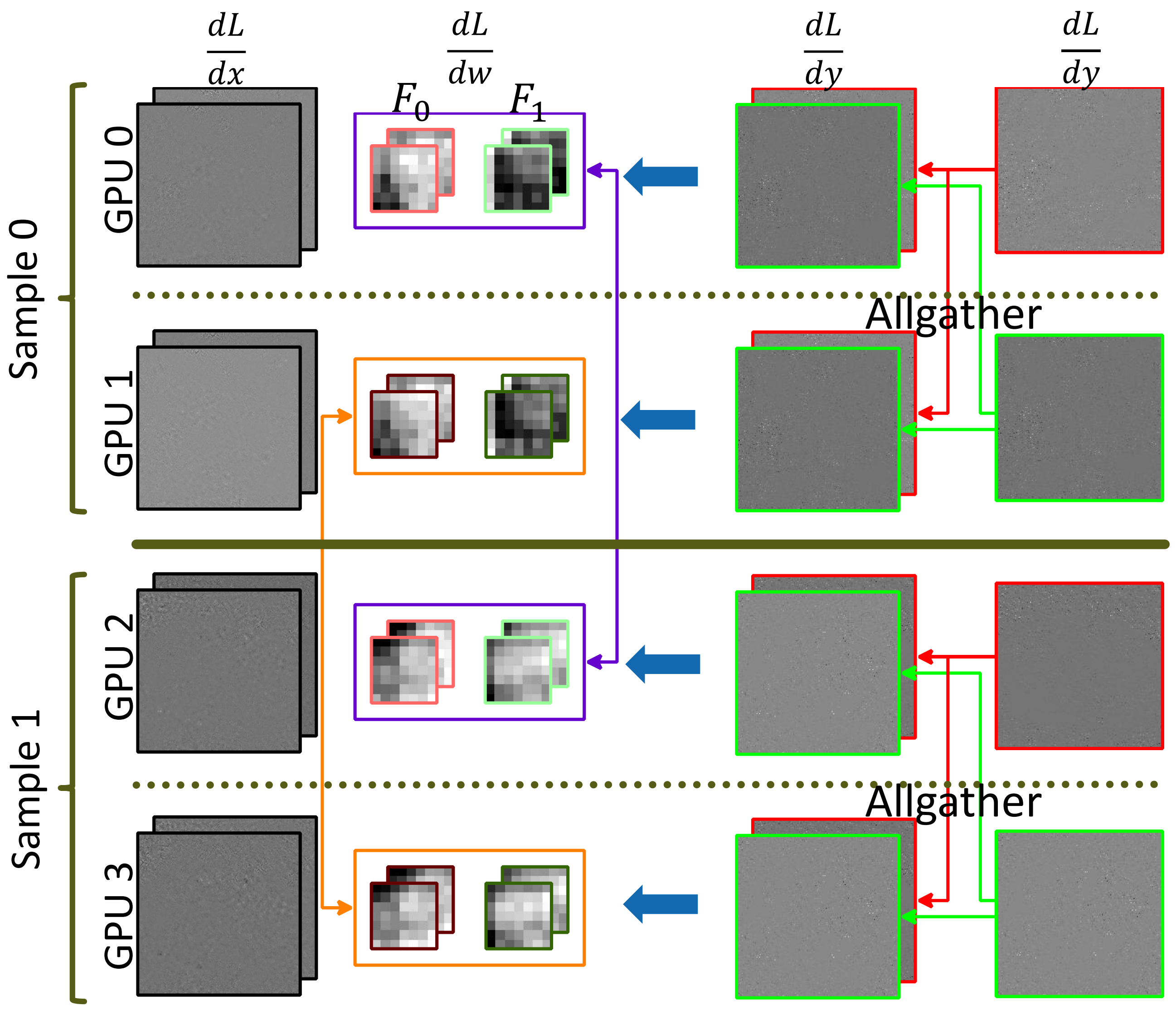
Stationary-x: Backward



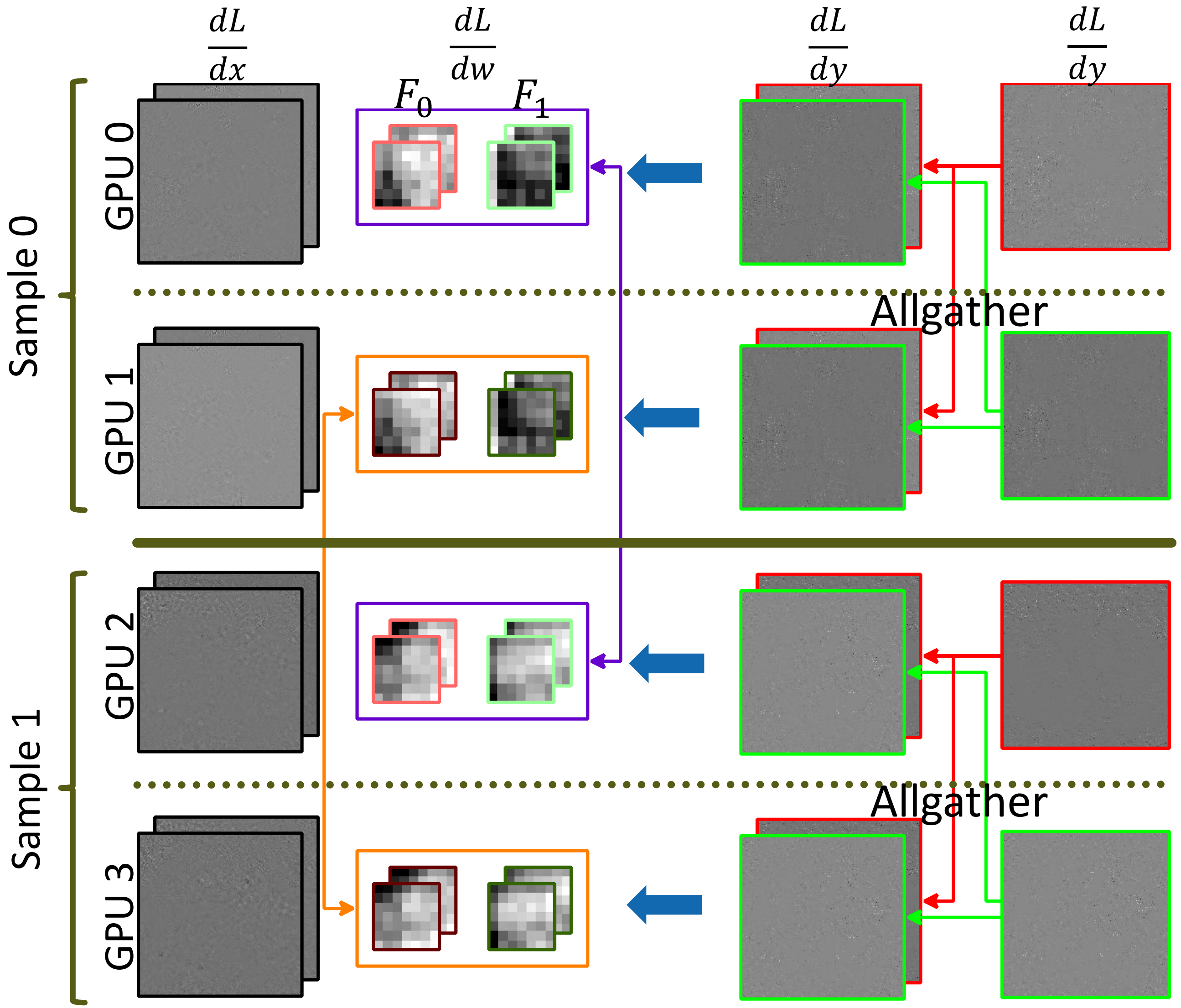
Stationary-x: Backward



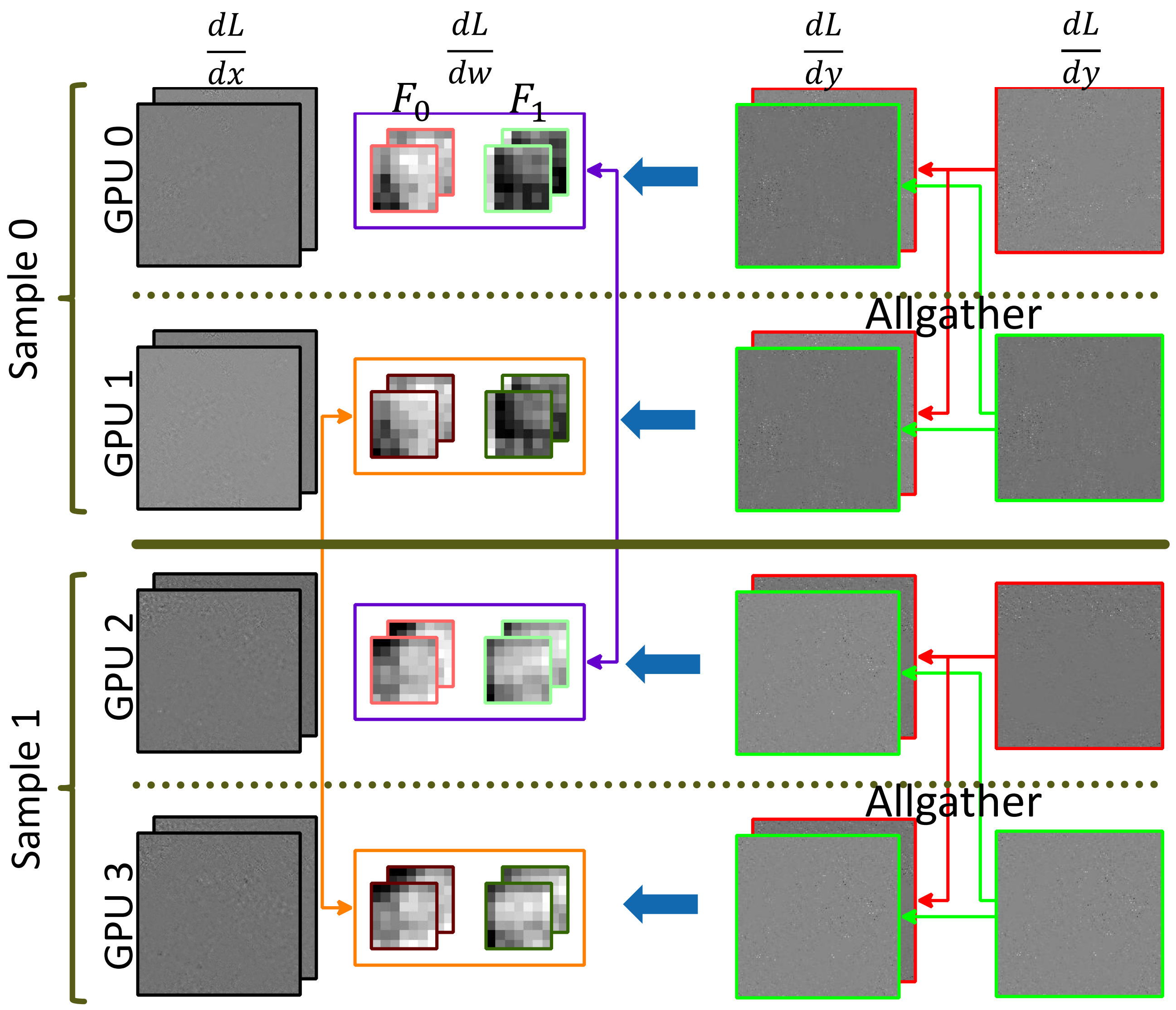
Stationary-x: Backward



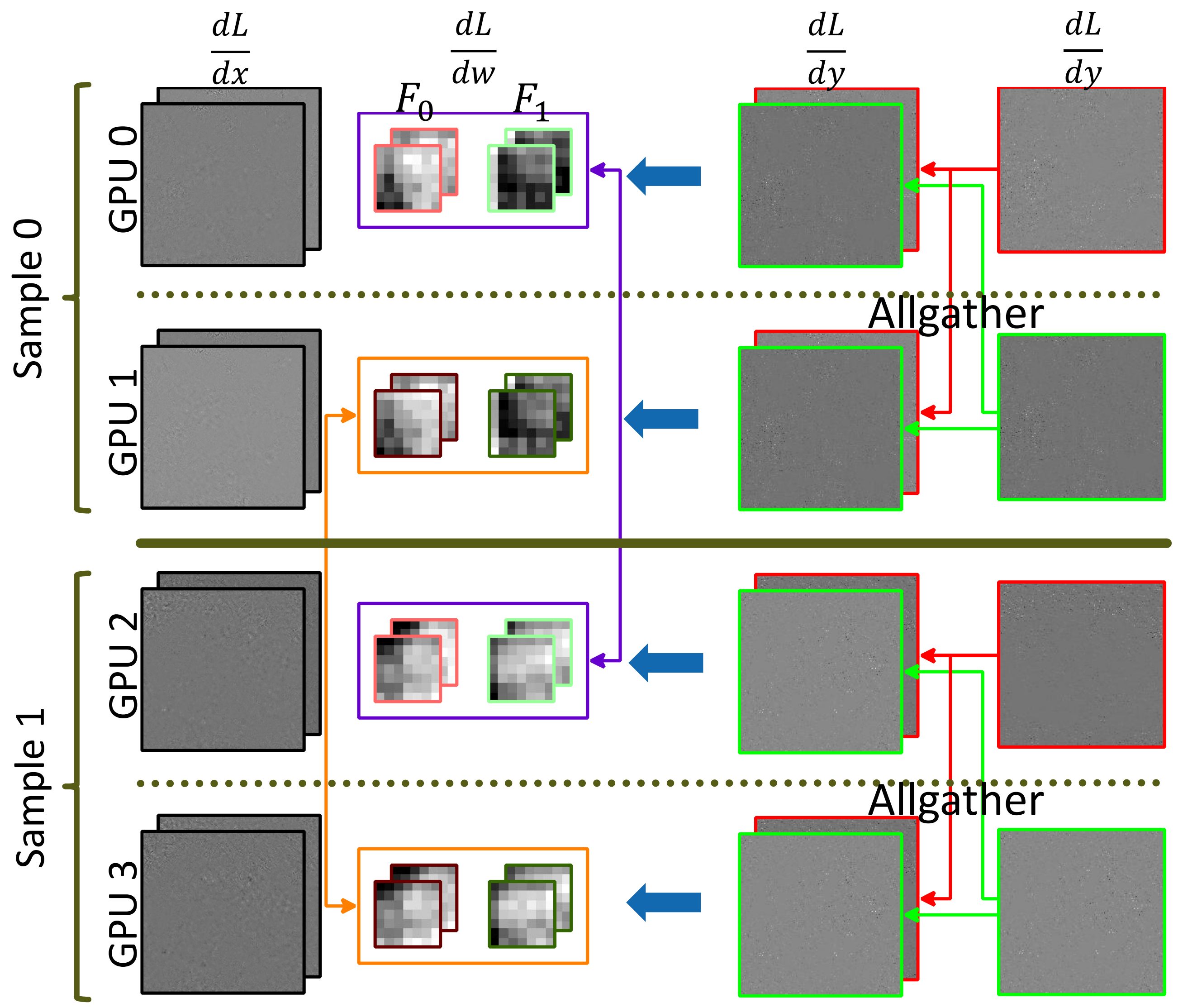
Stationary-x: Backward



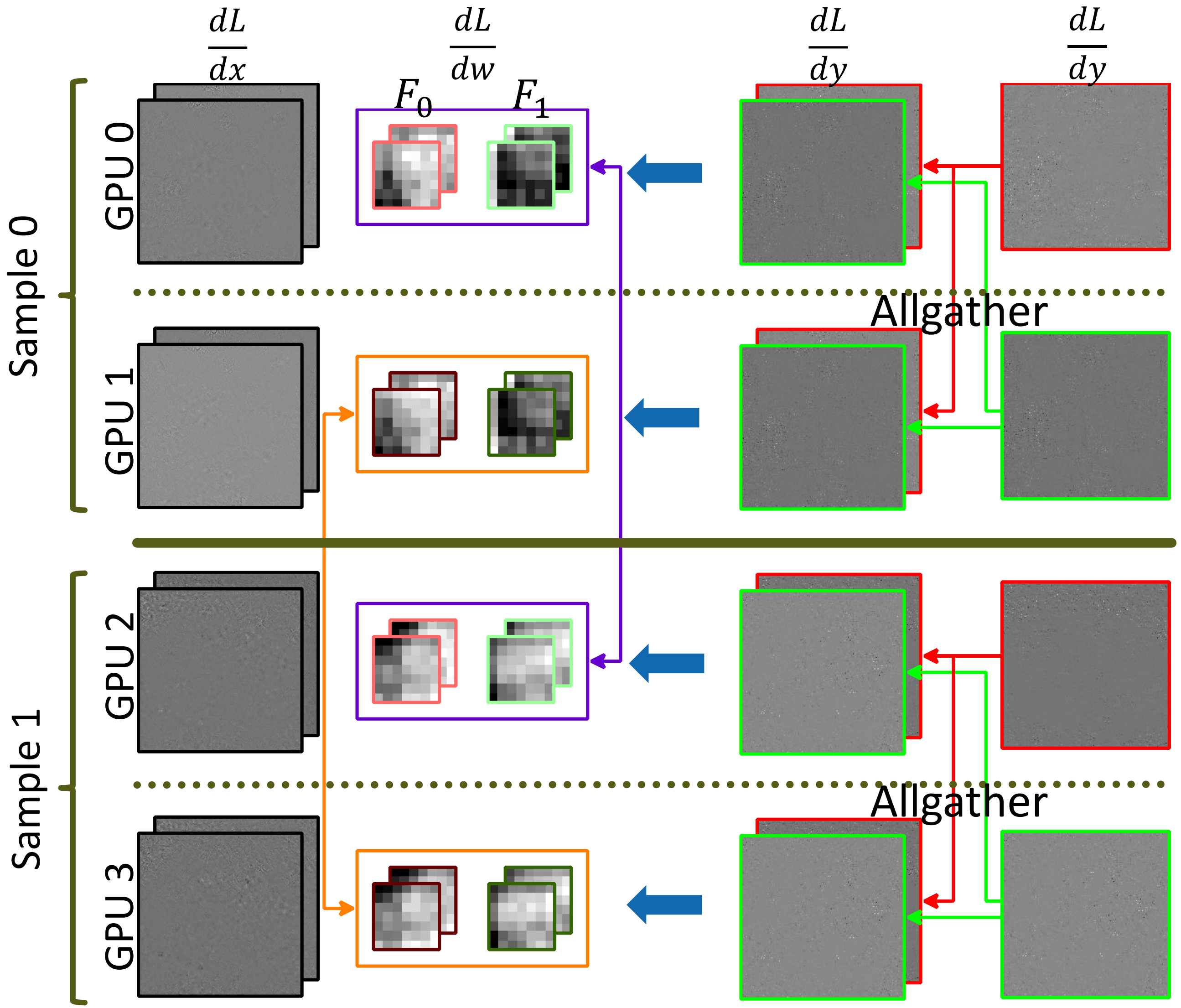
Stationary-x: Backward



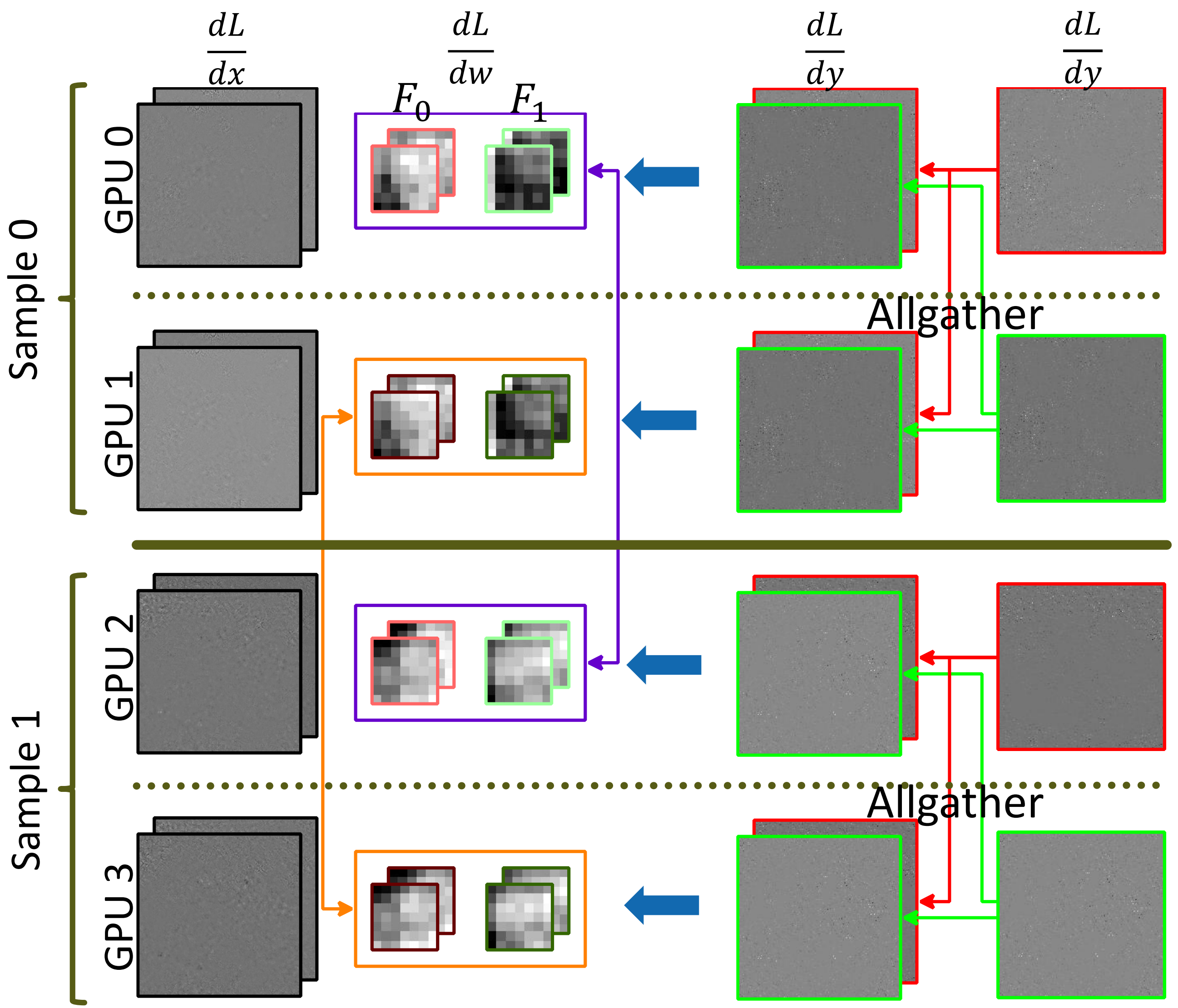
Stationary-x: Backward



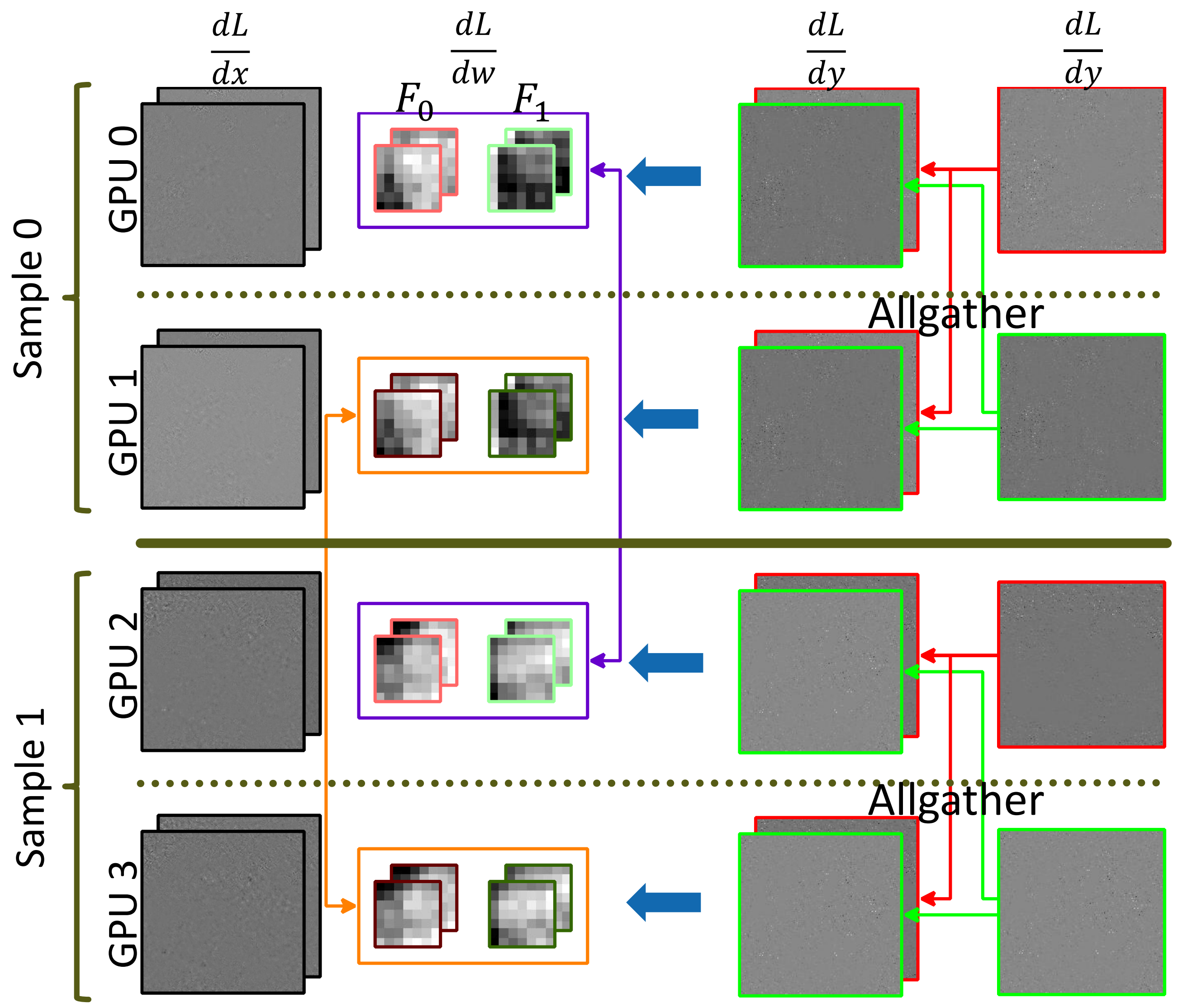
Stationary-x: Backward



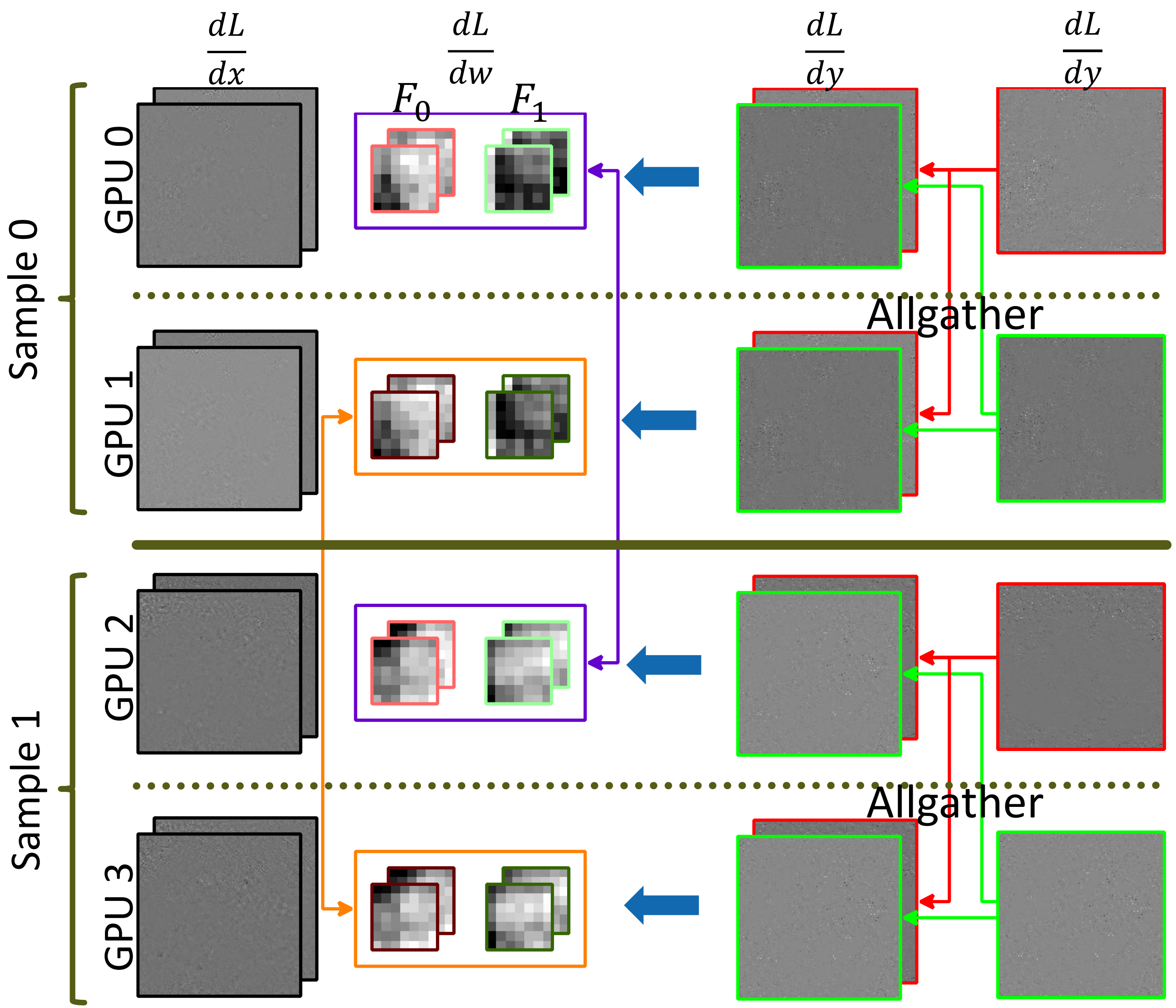
Stationary-x: Backward



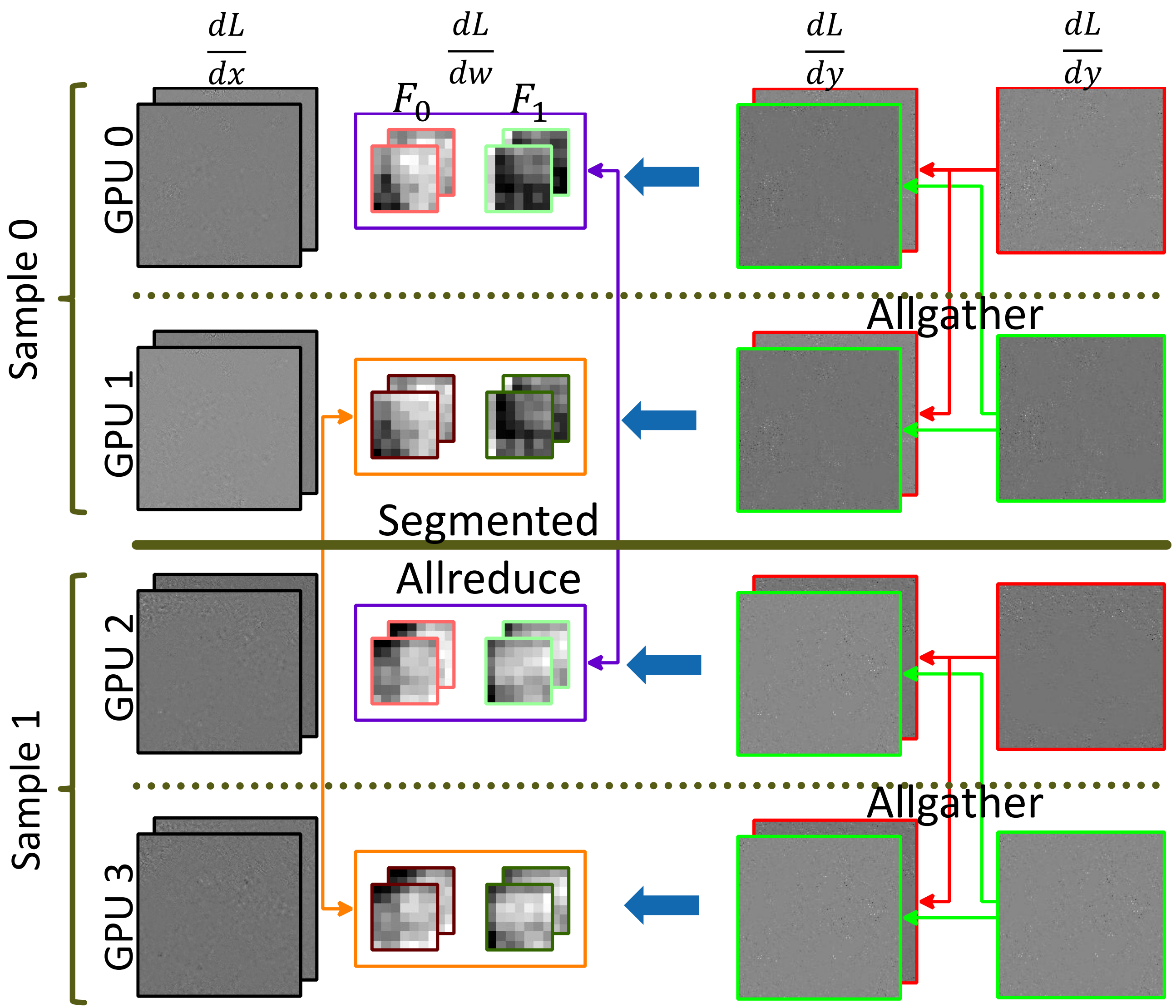
Stationary-x: Backward



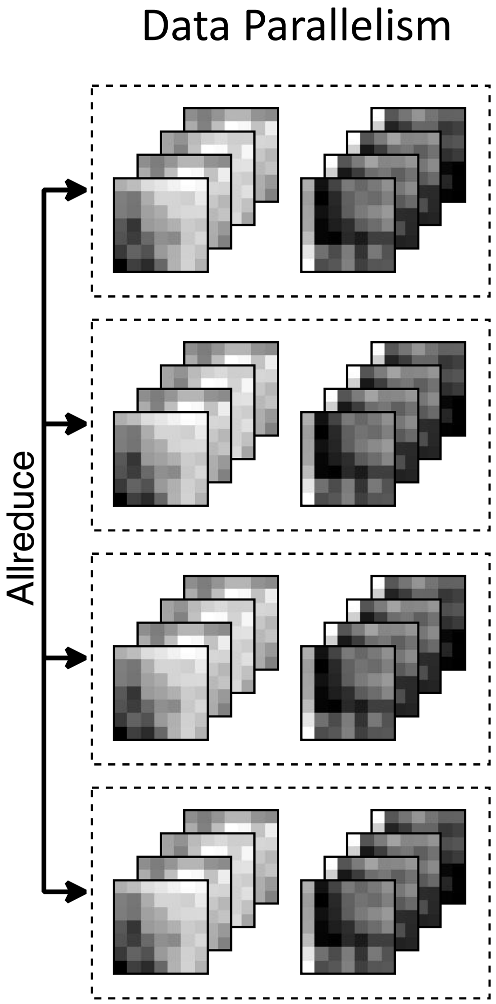
Stationary-x: Backward



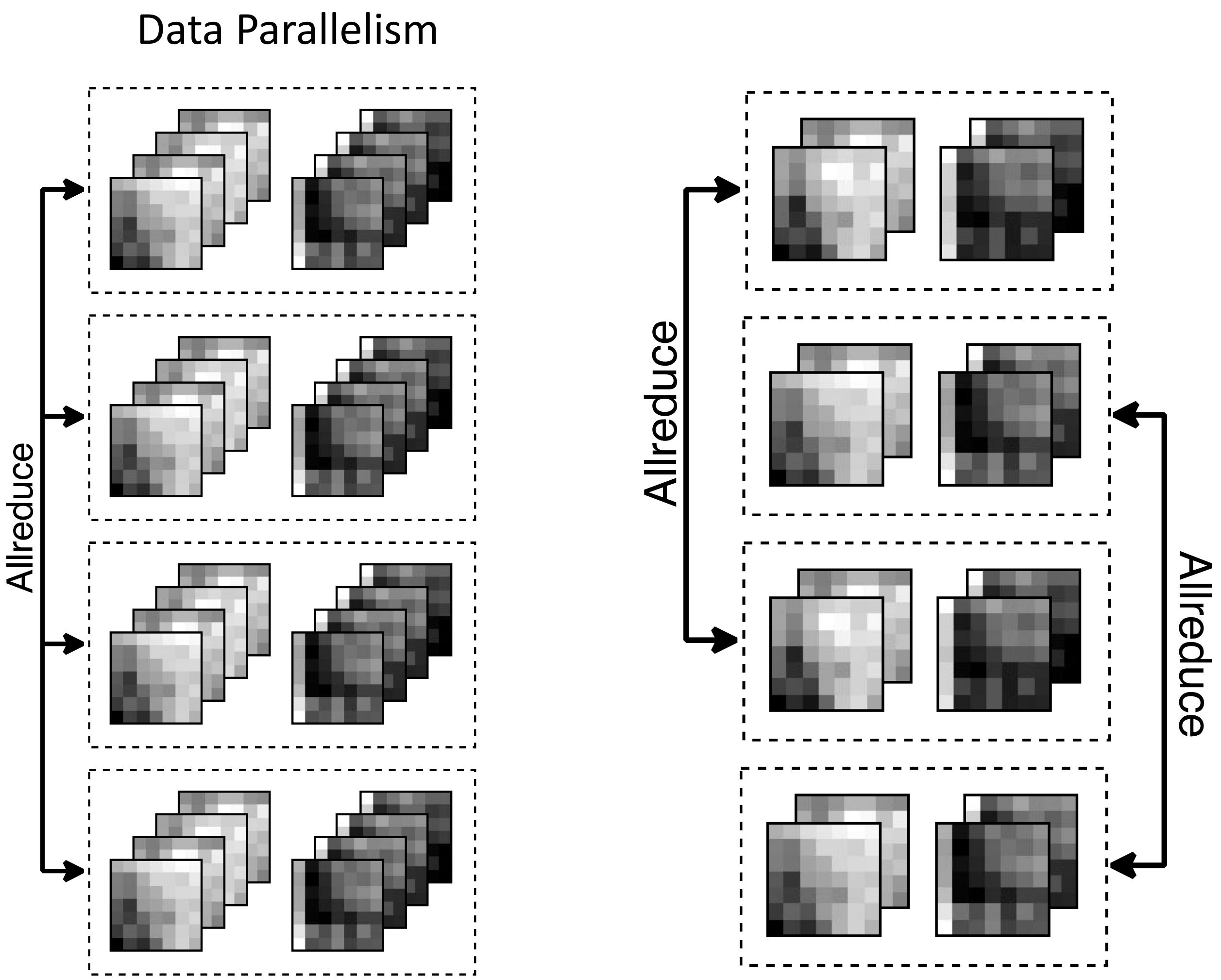
Stationary-x: Backward



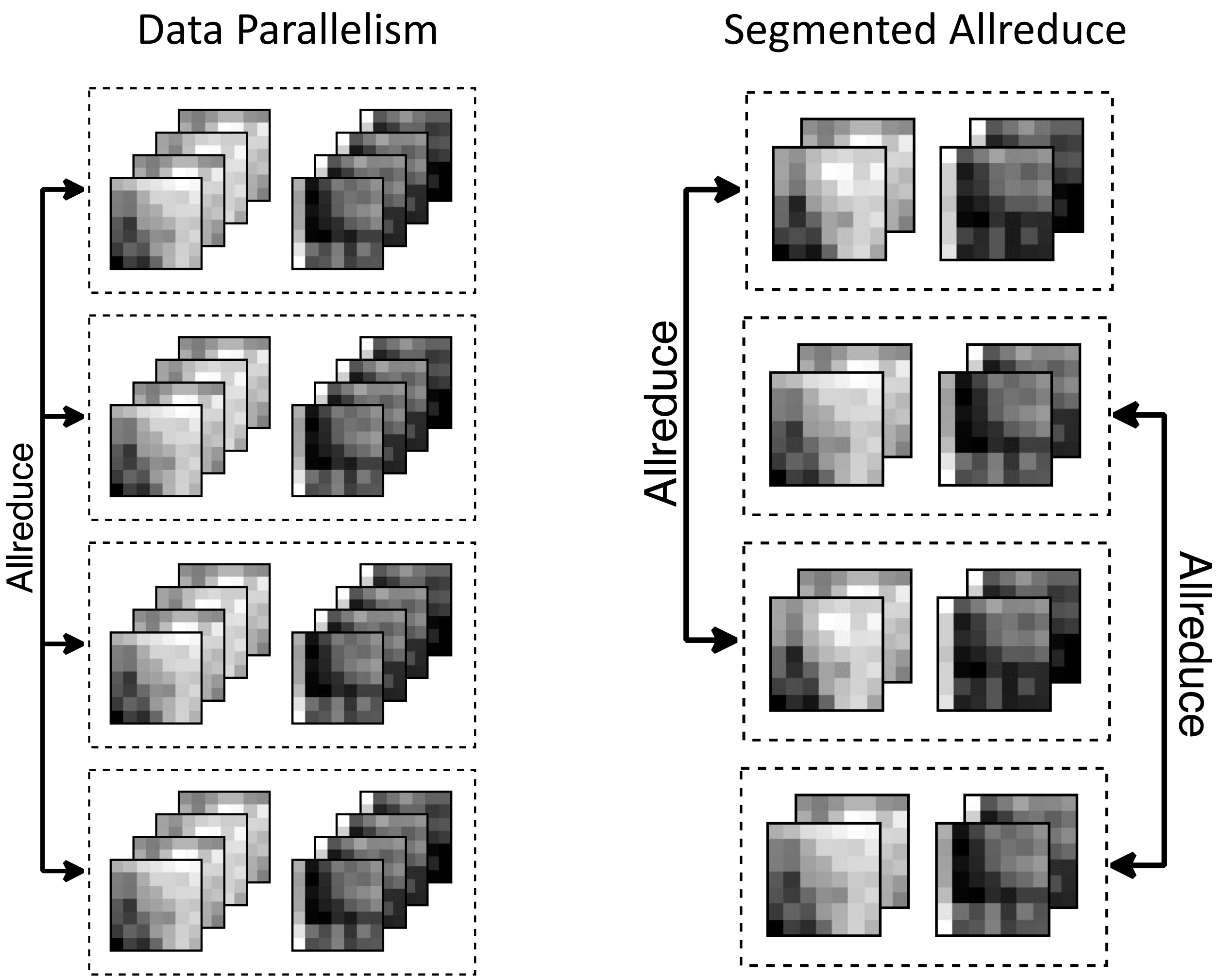
Segmented allreduce



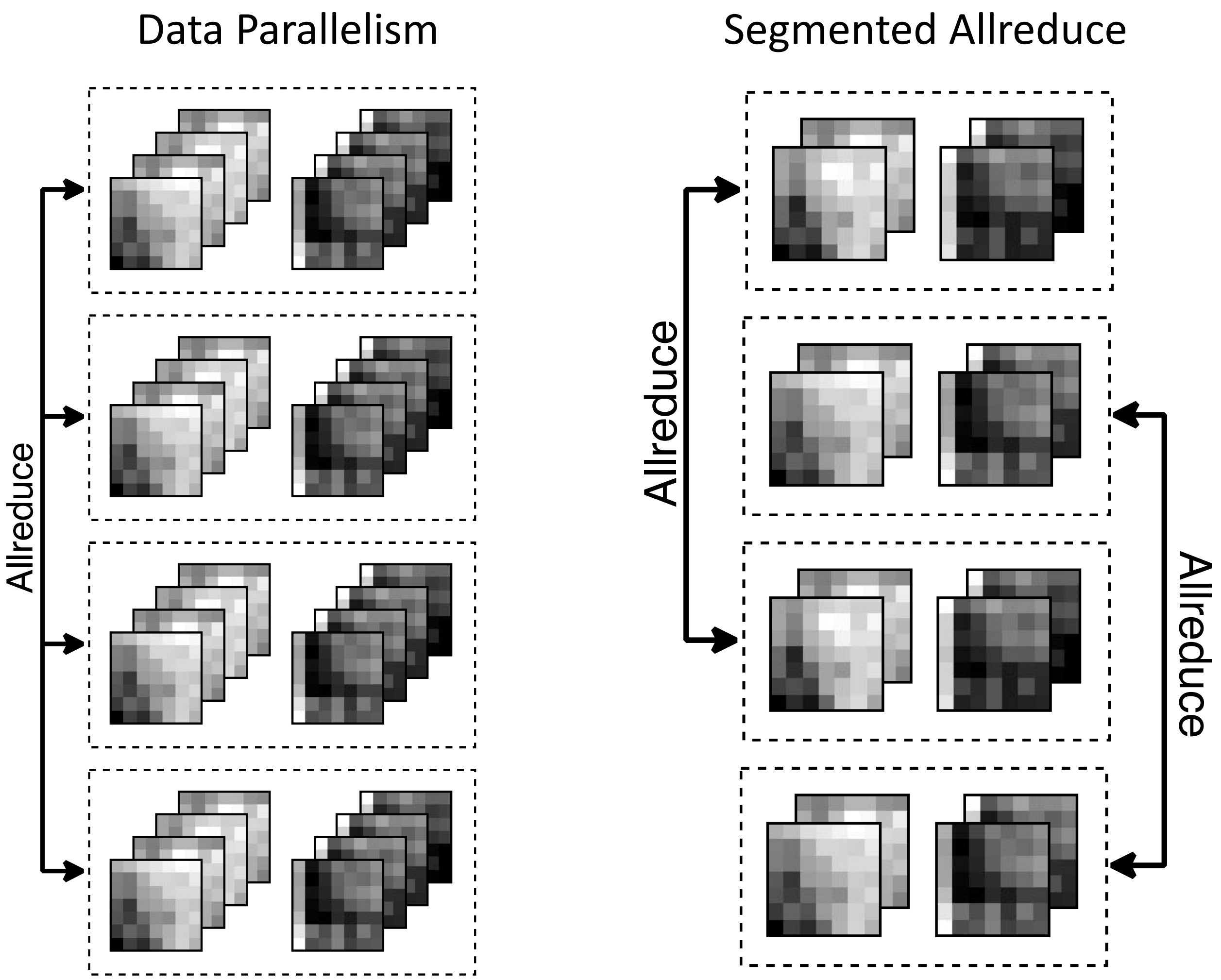
Segmented allreduce

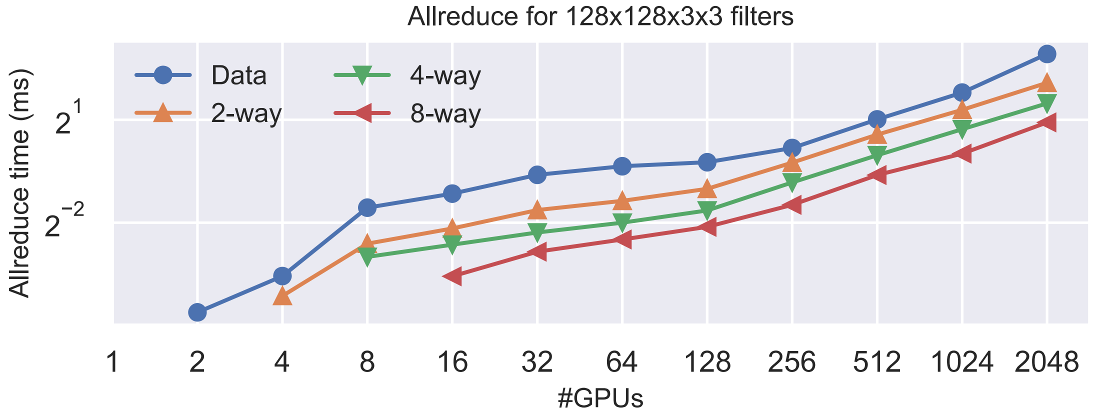


Segmented allreduce

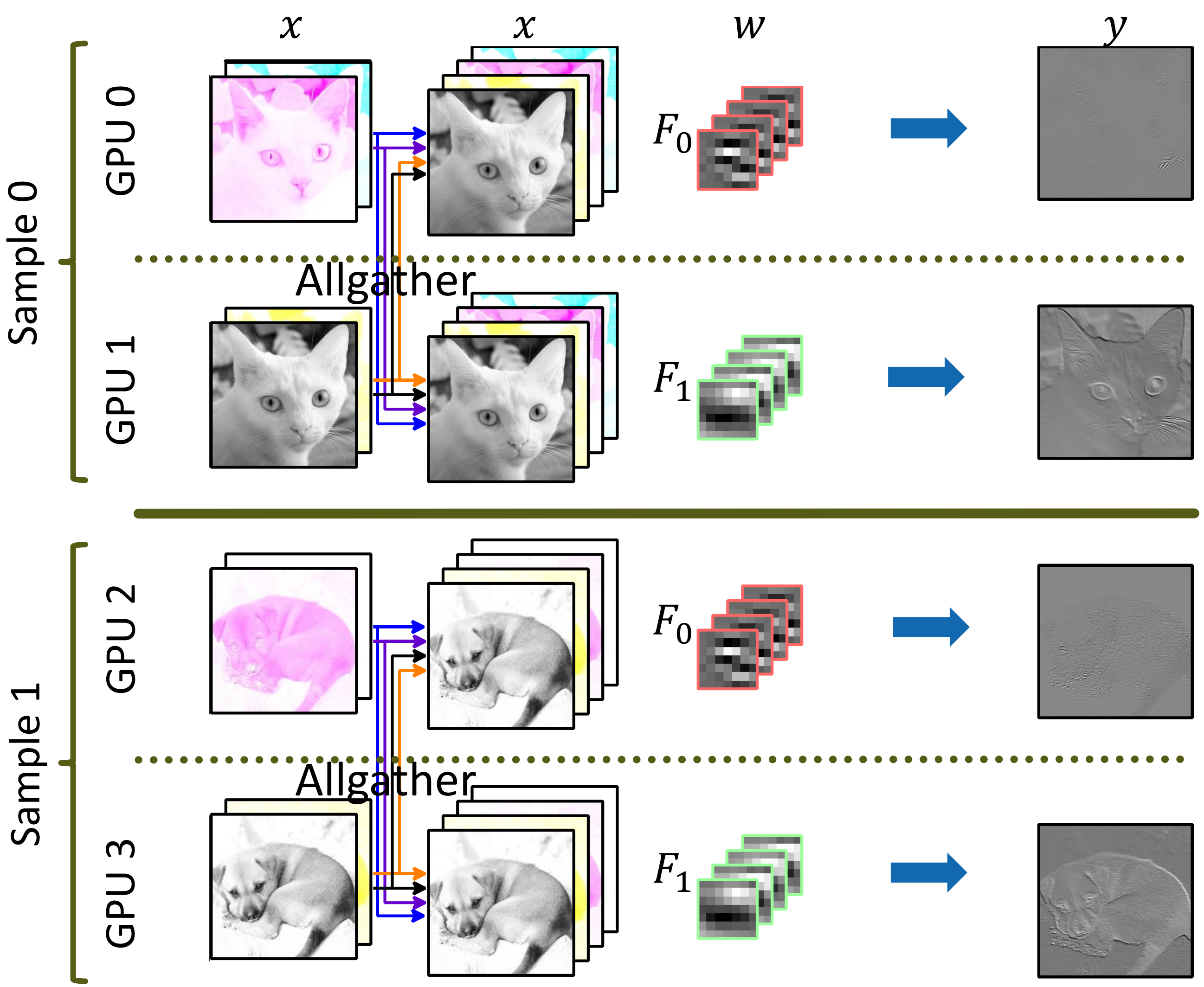


Segmented allreduce

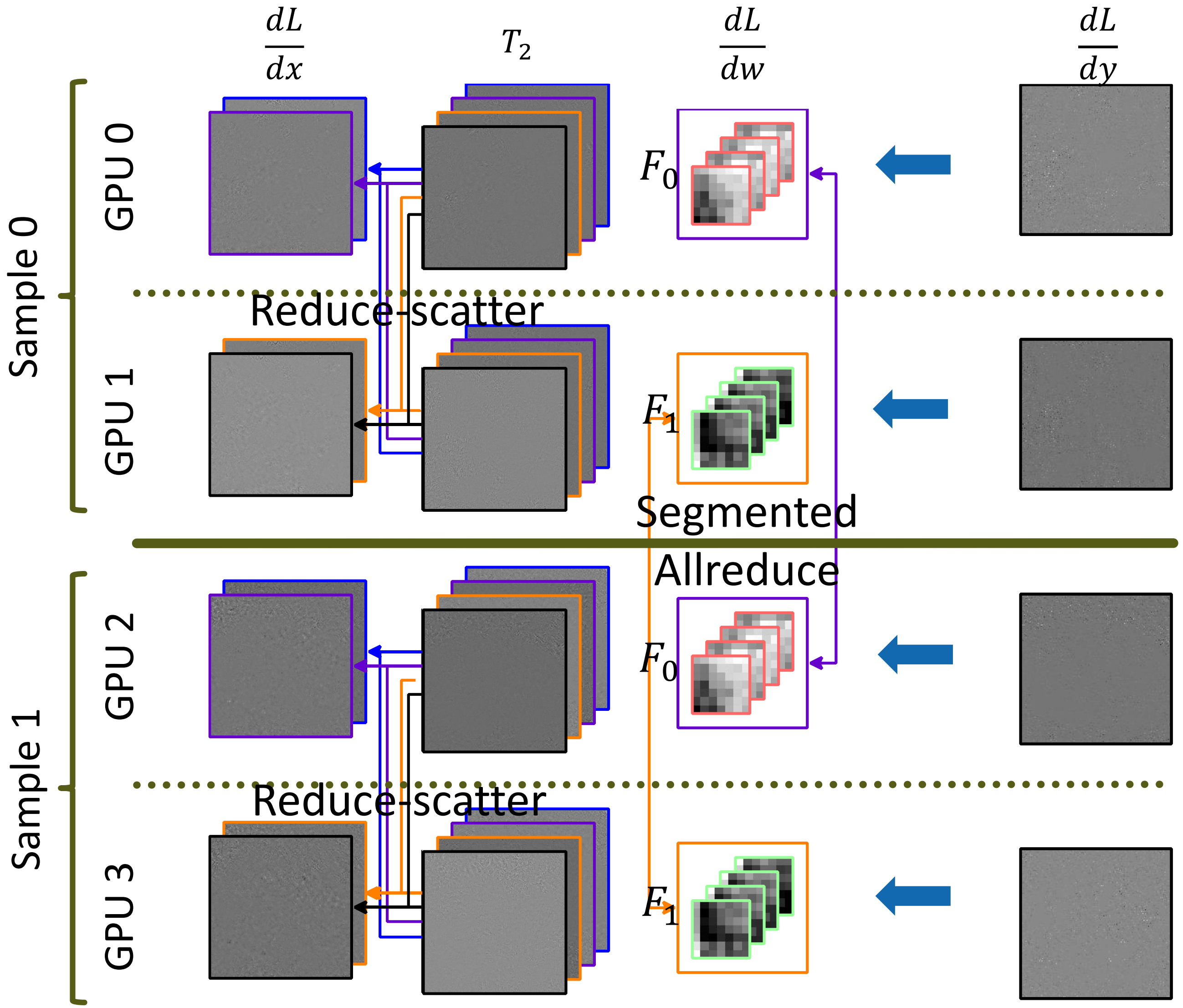




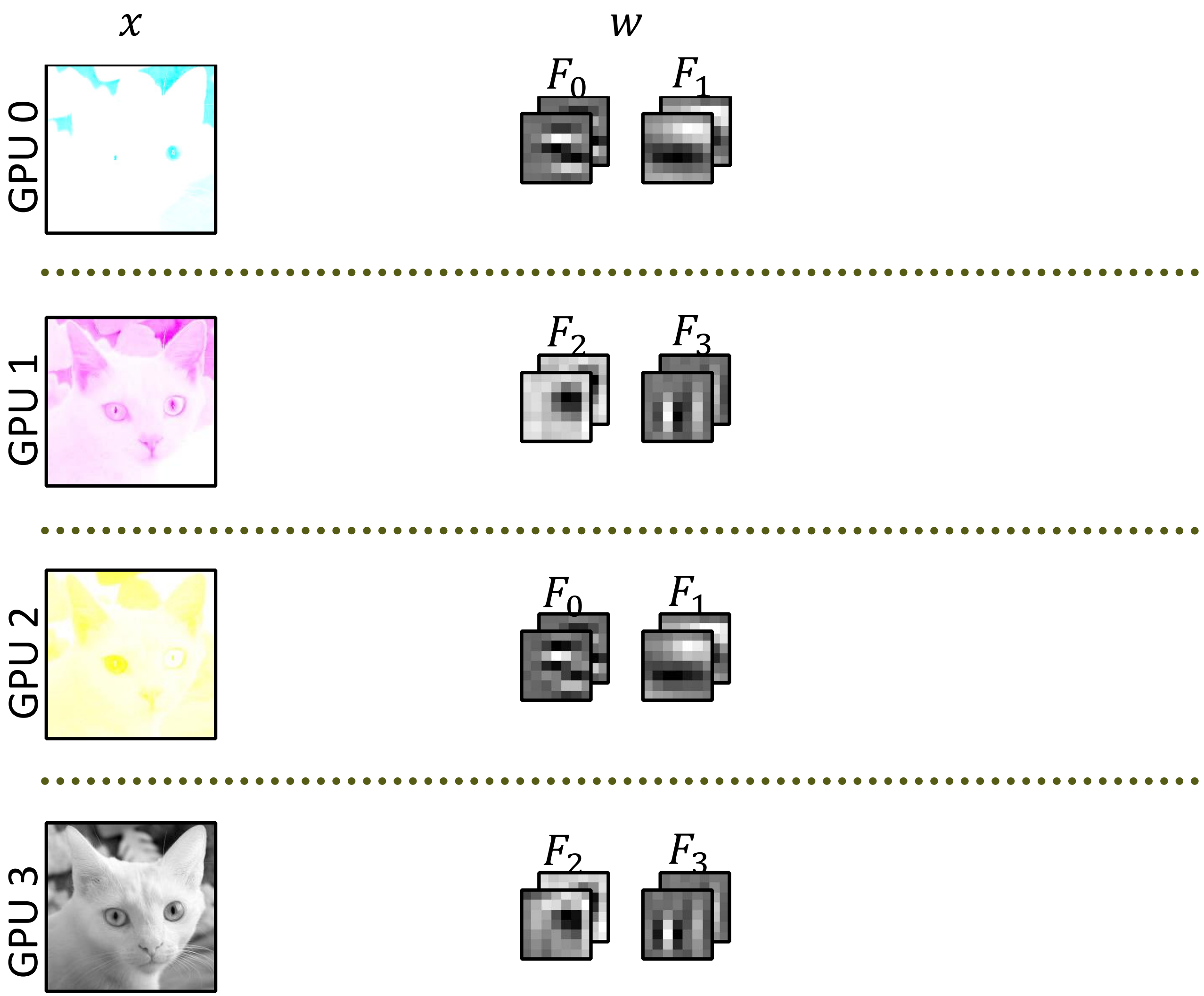
Stationary-y: Forward



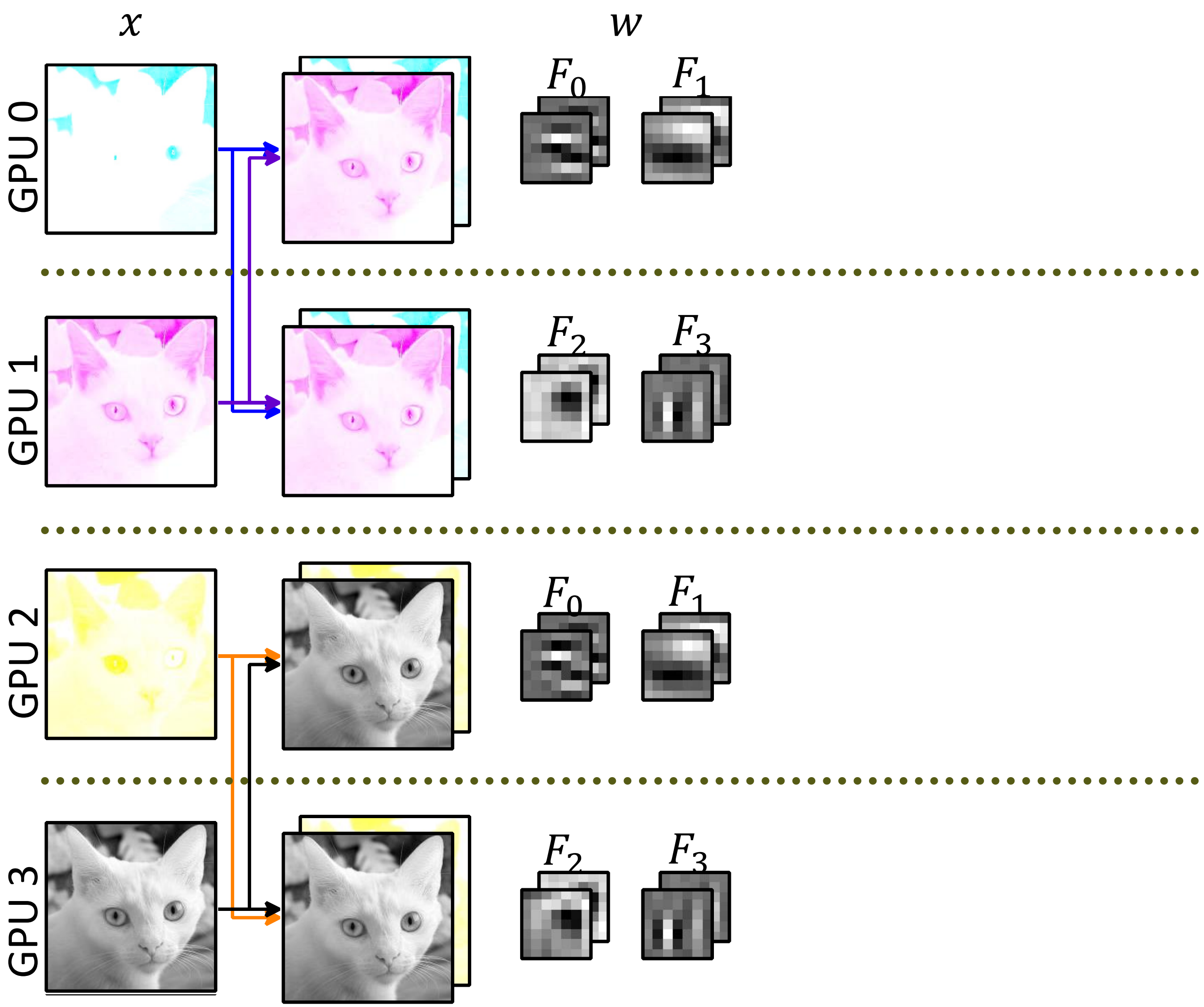
Stationary-y: Backward



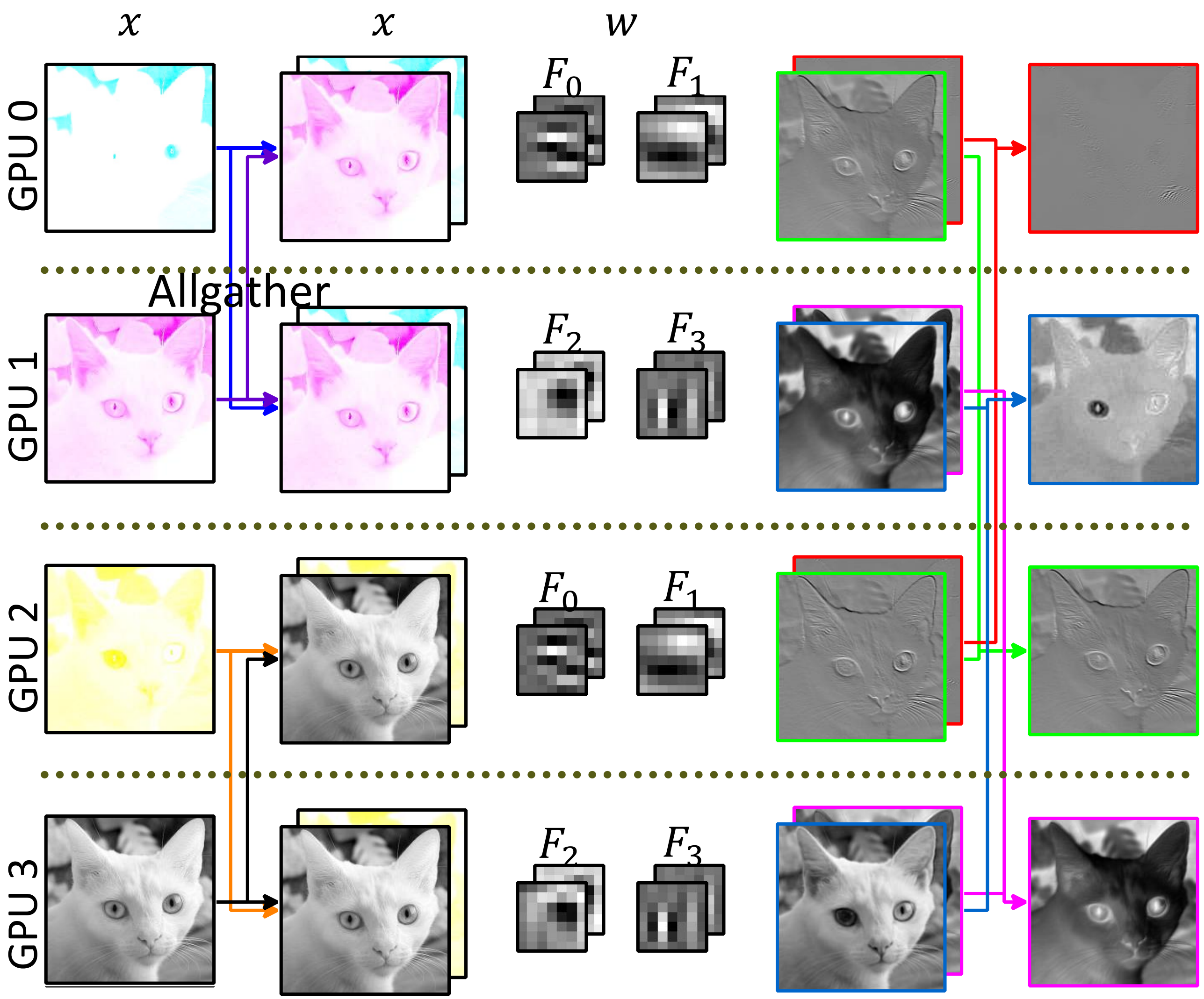
Stationary-w: Forward



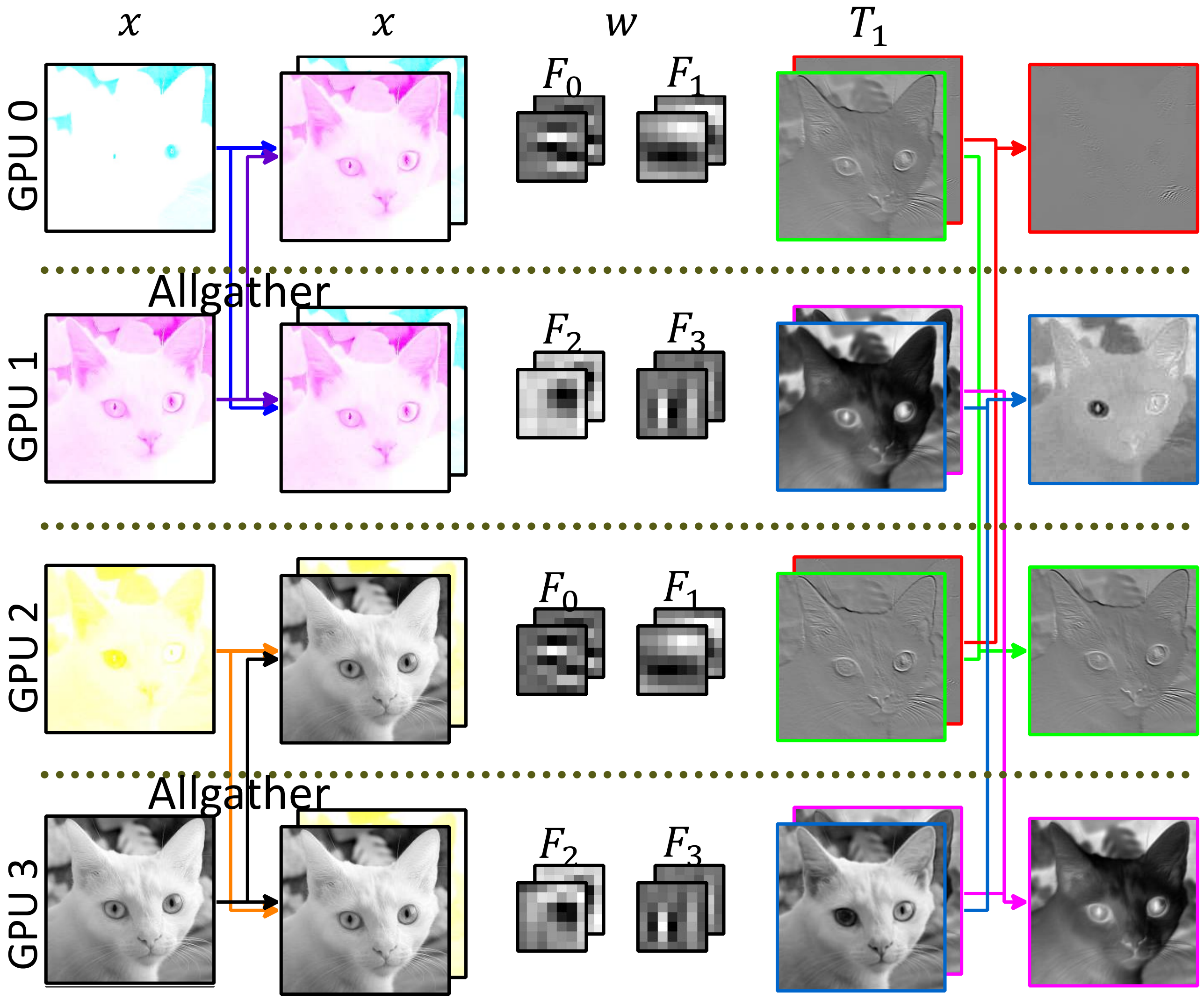
Stationary-w: Forward



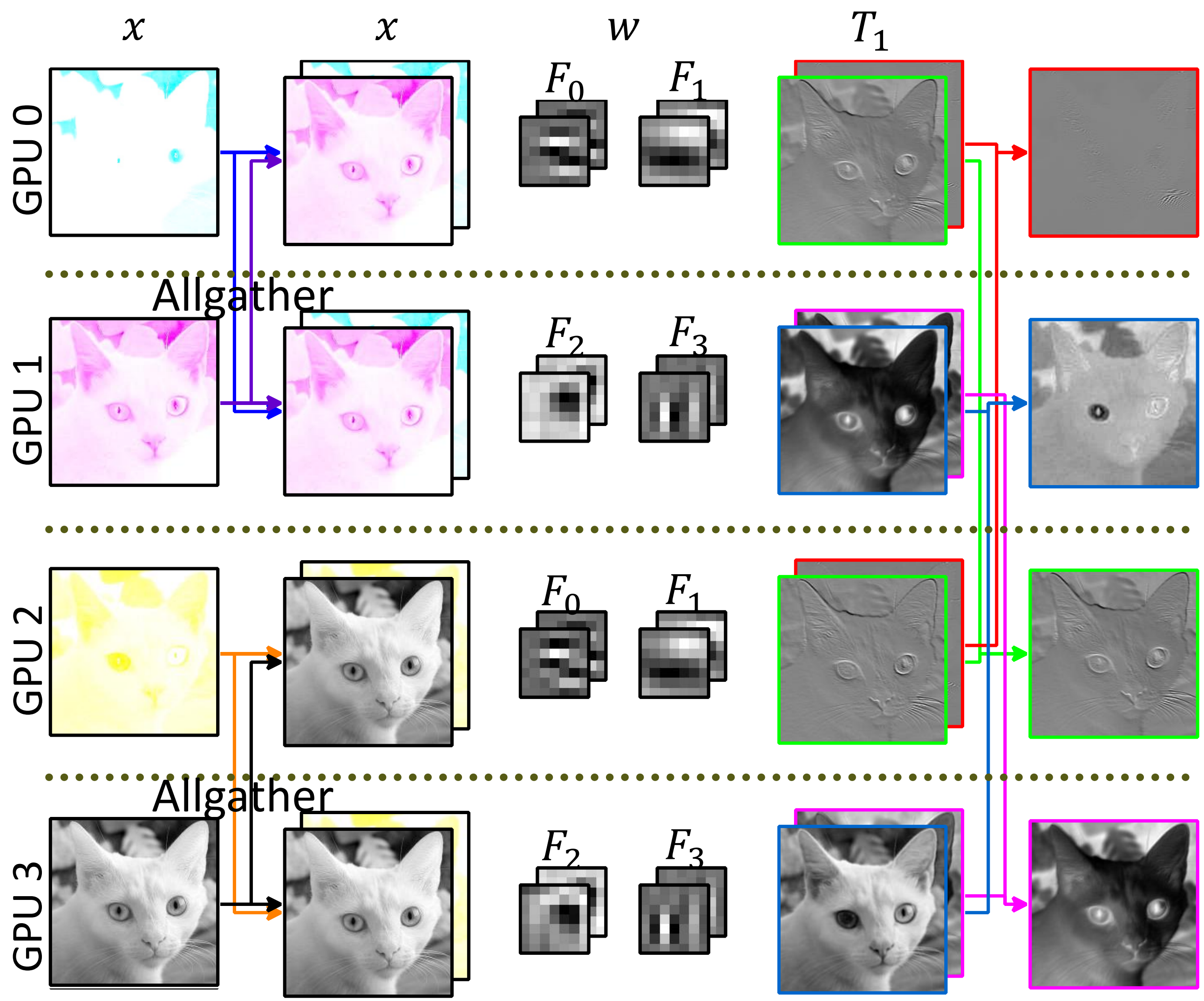
Stationary-w: Forward



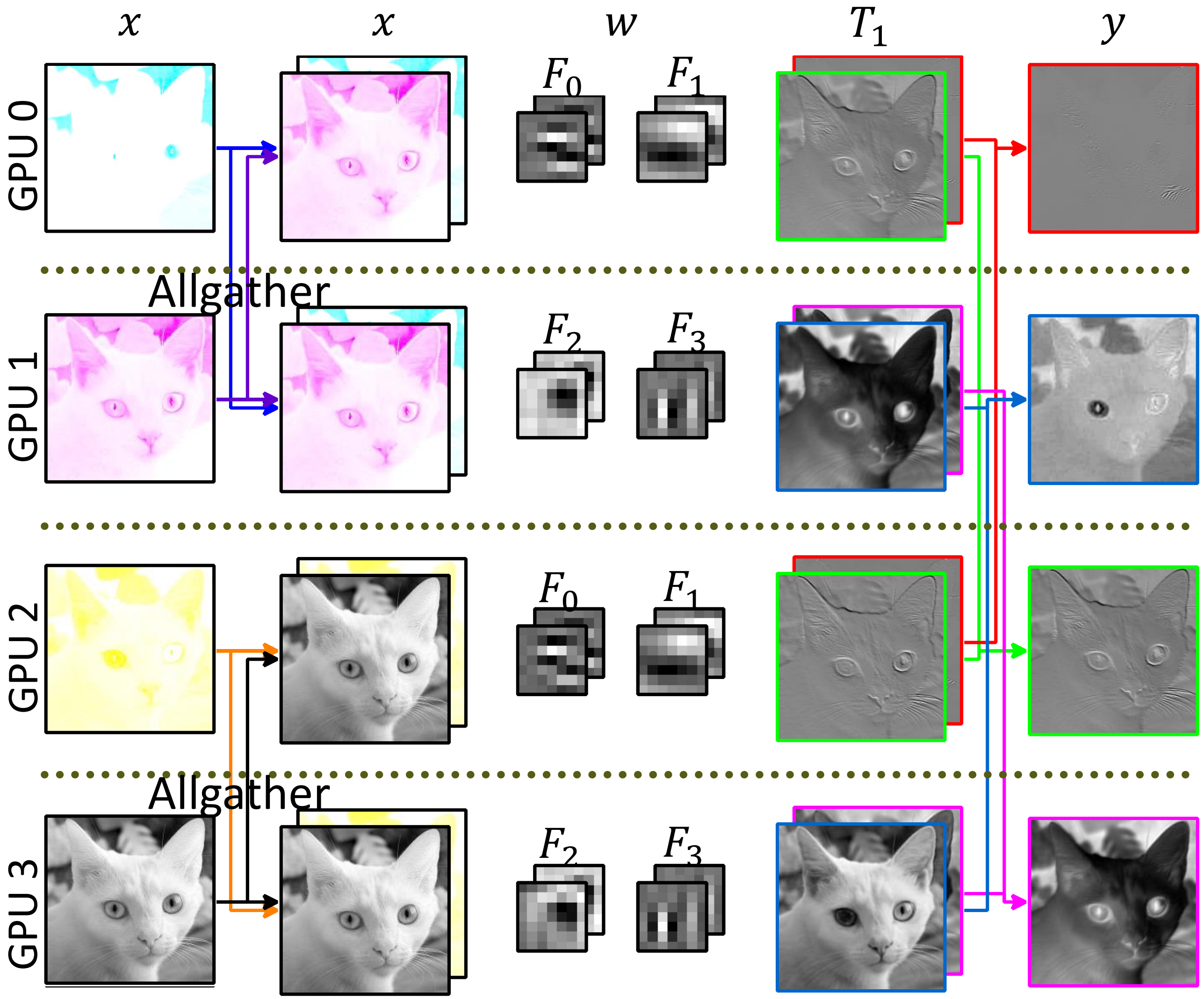
Stationary-w: Forward



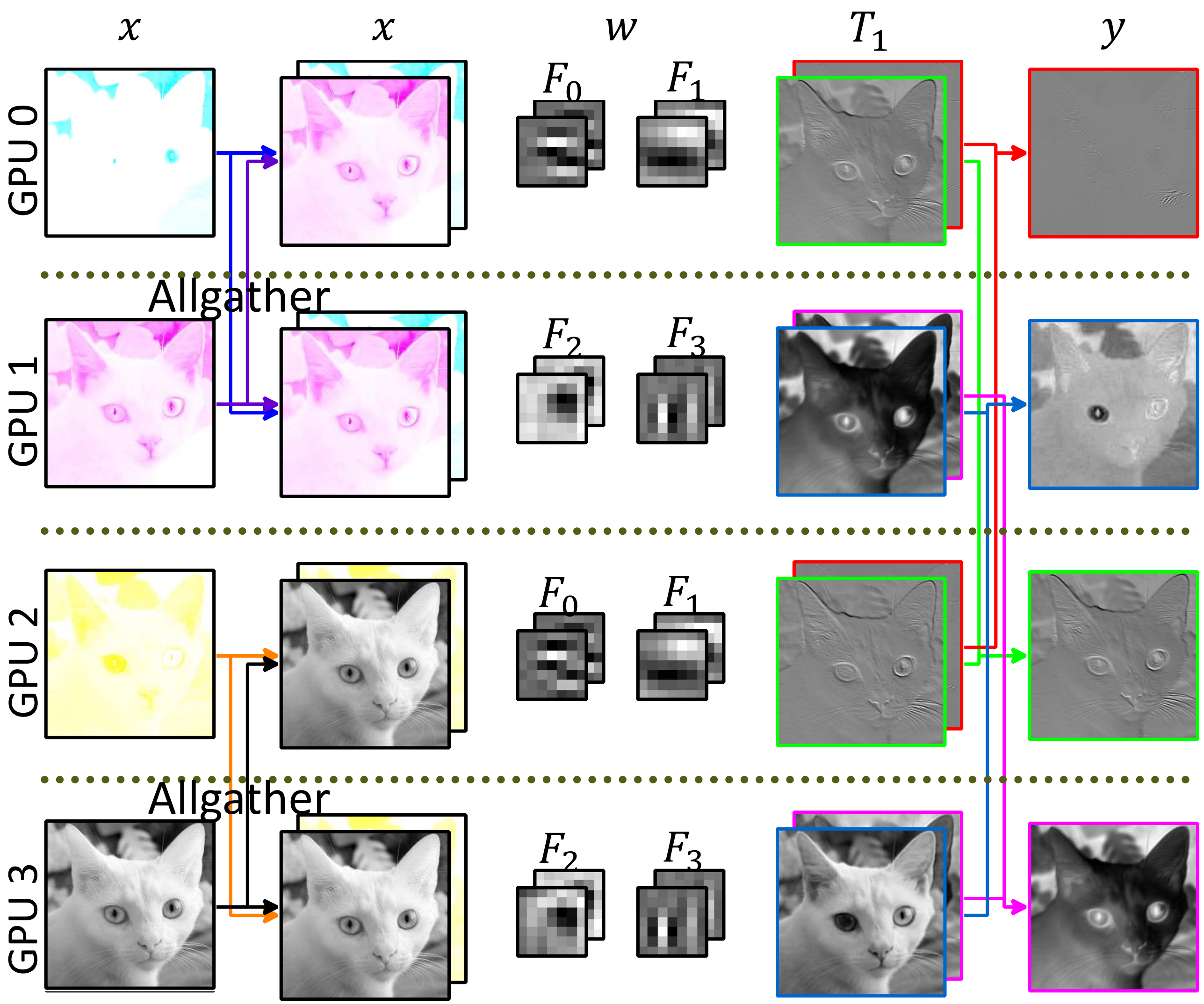
Stationary-w: Forward



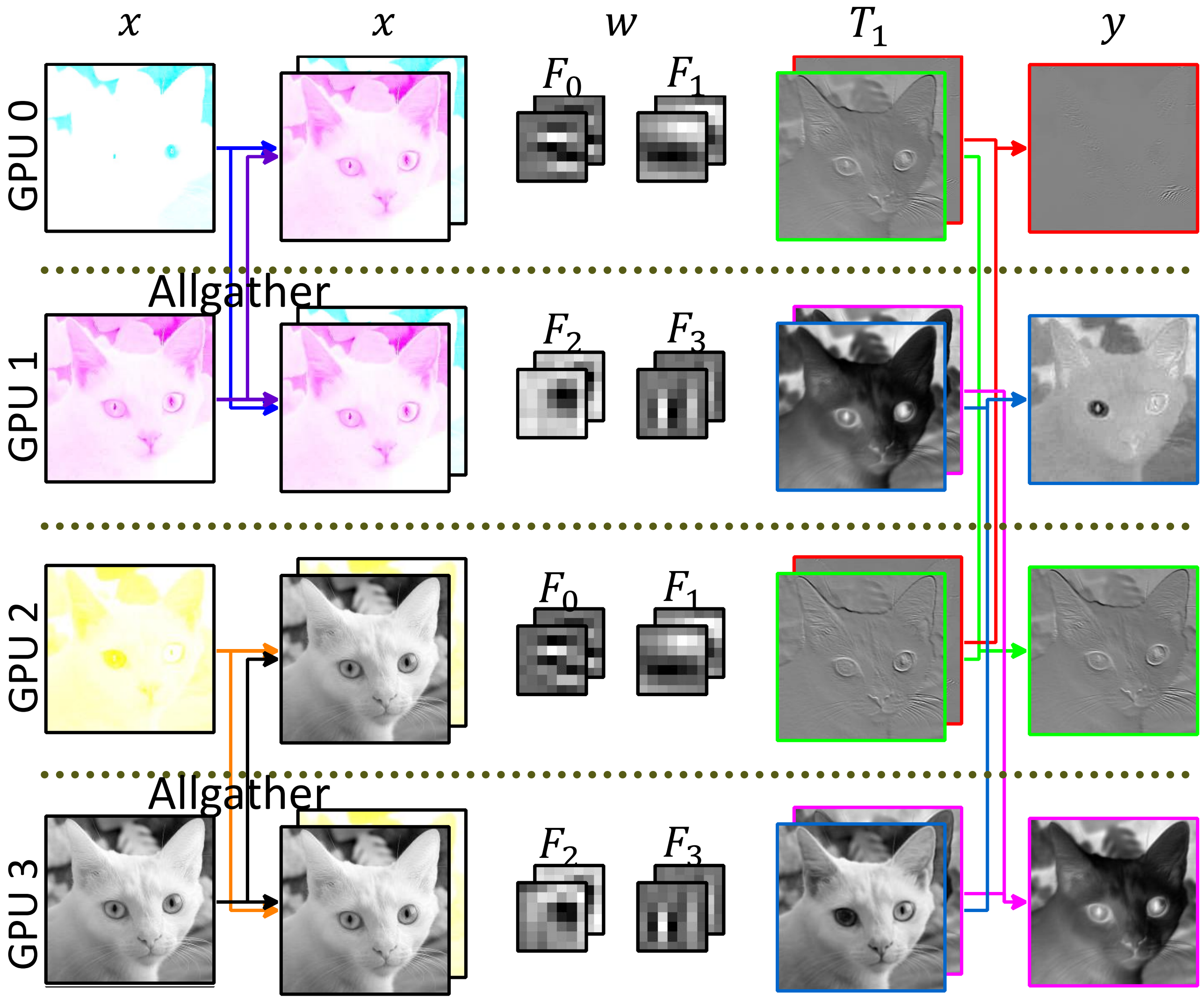
Stationary-w: Forward



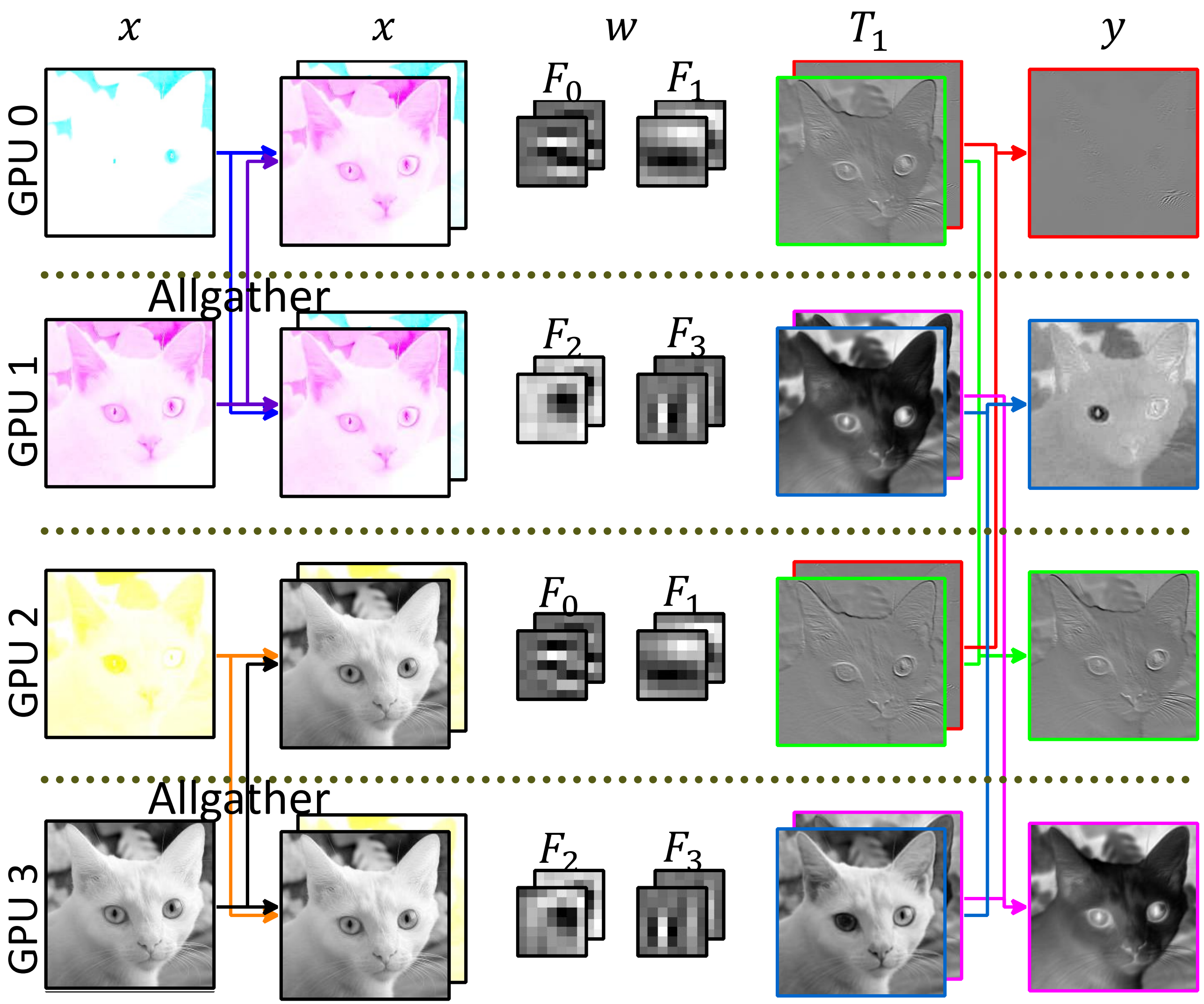
Stationary-w: Forward



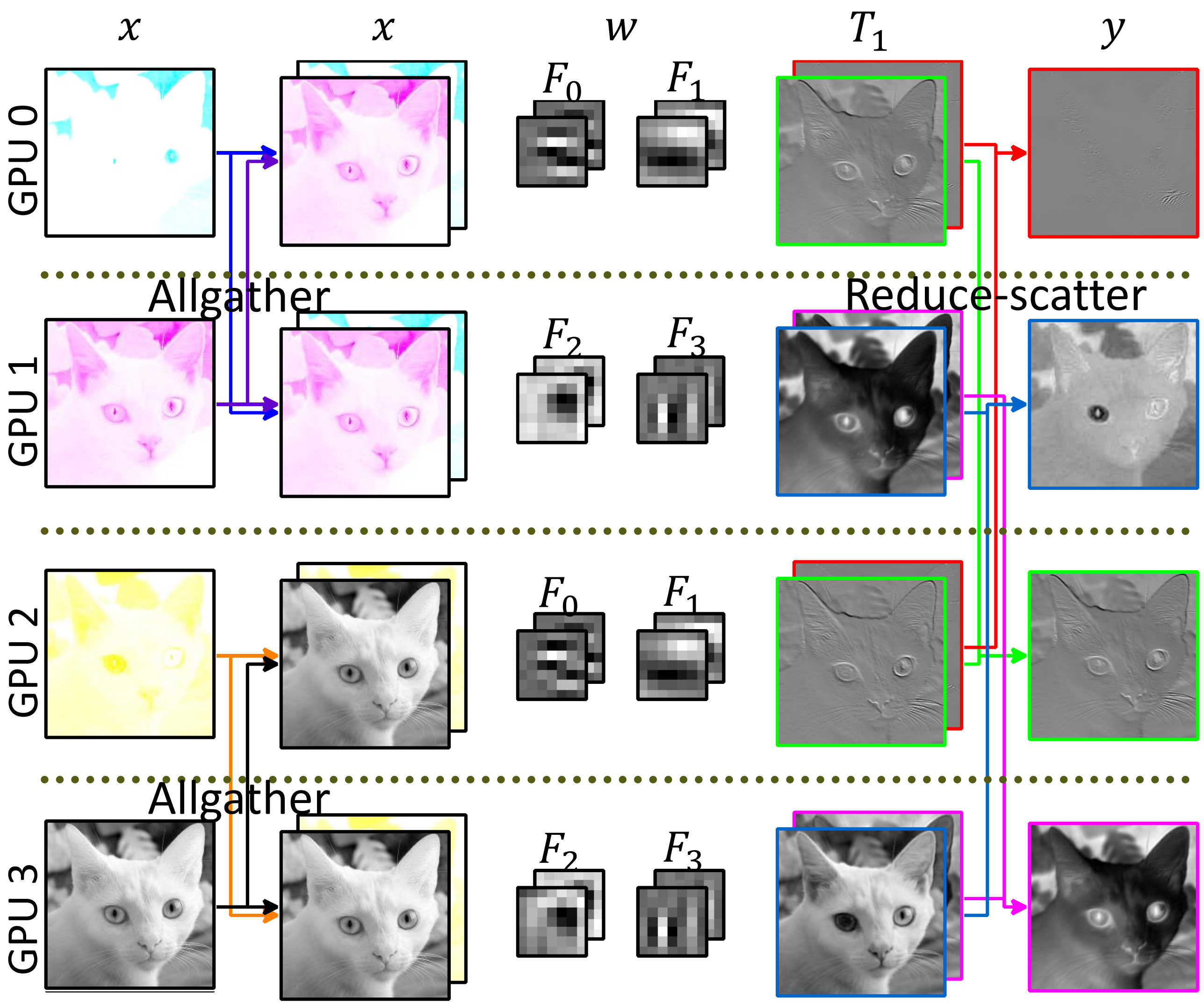
Stationary-w: Forward



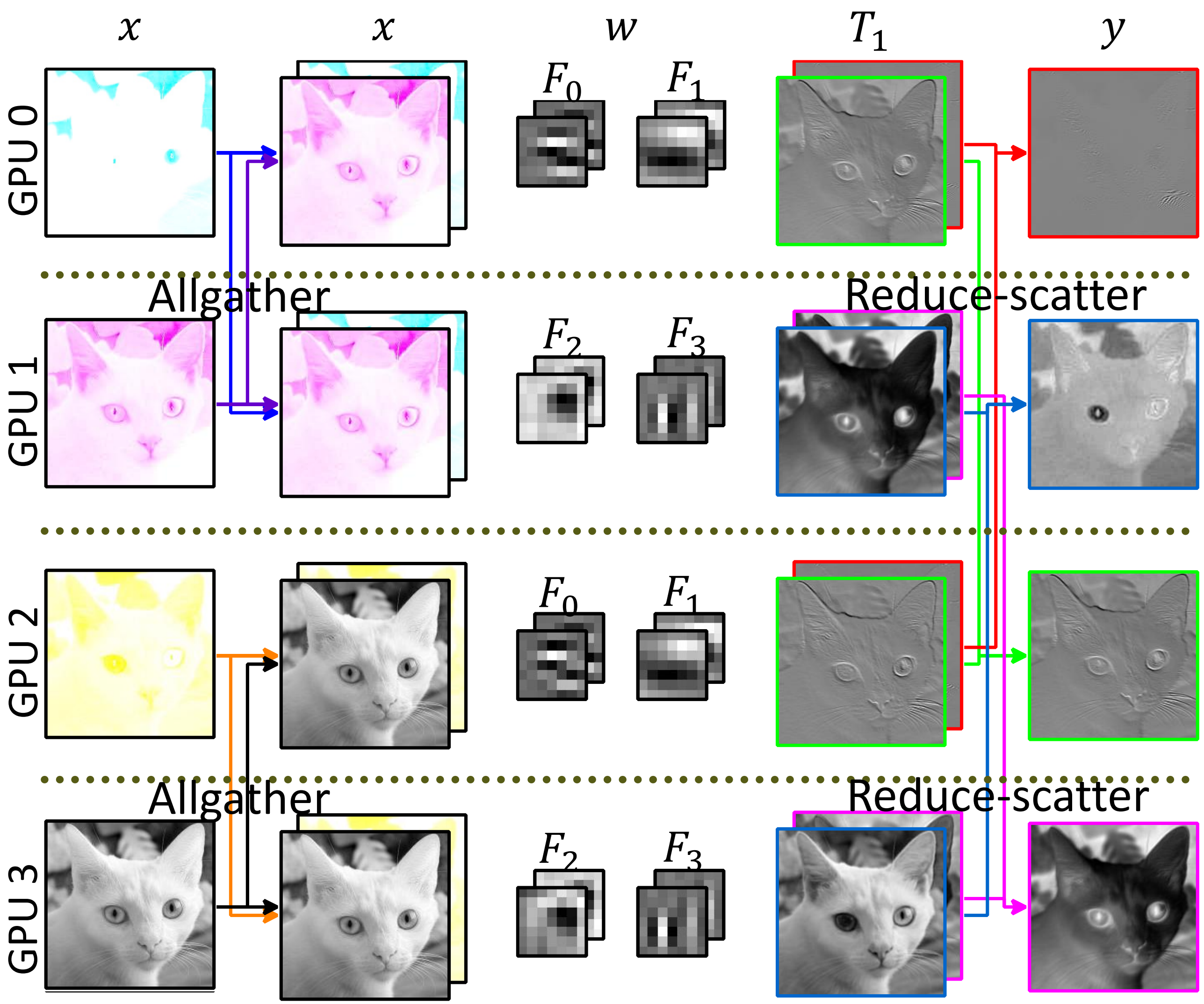
Stationary-w: Forward



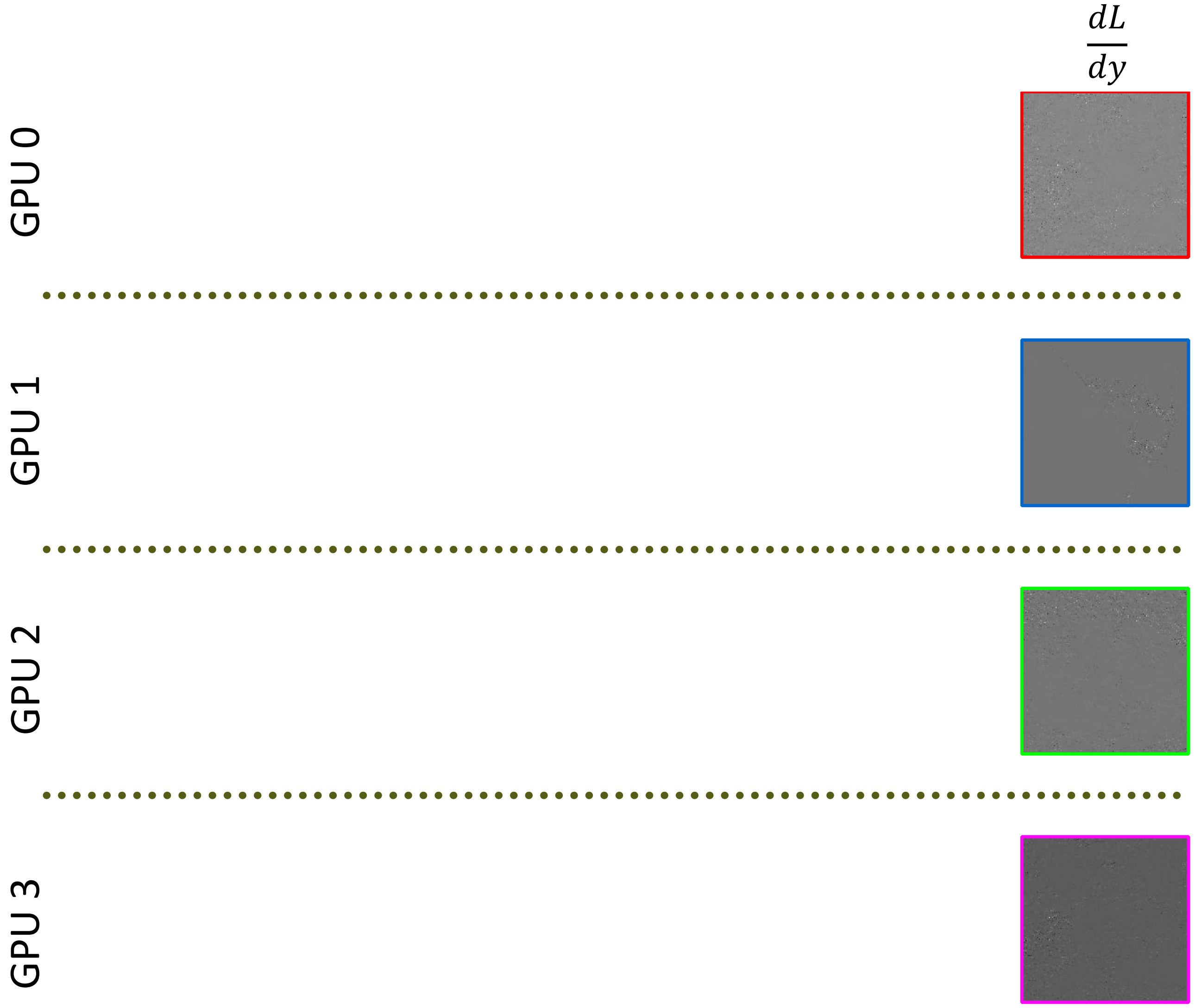
Stationary-w: Forward



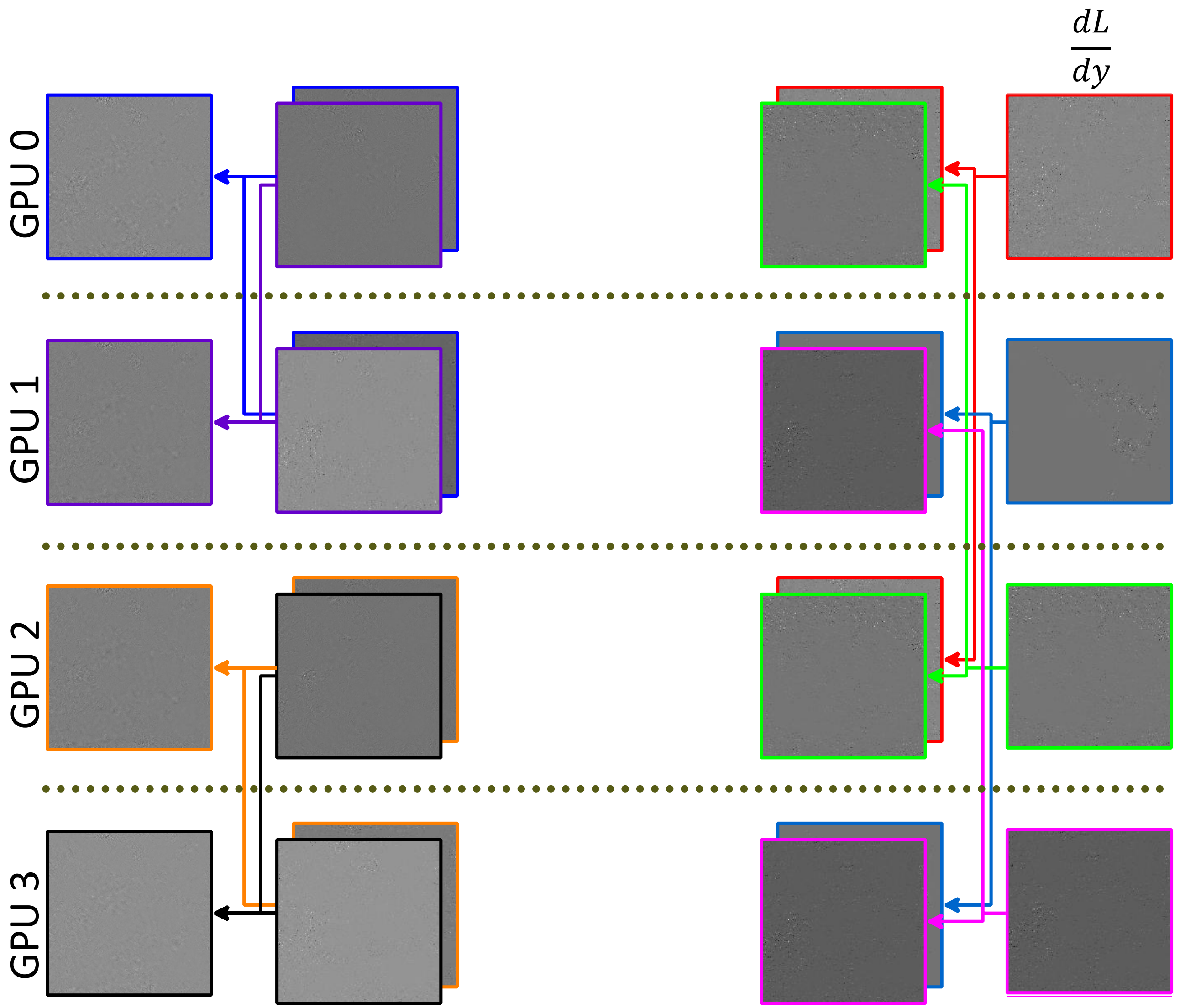
Stationary-w: Forward



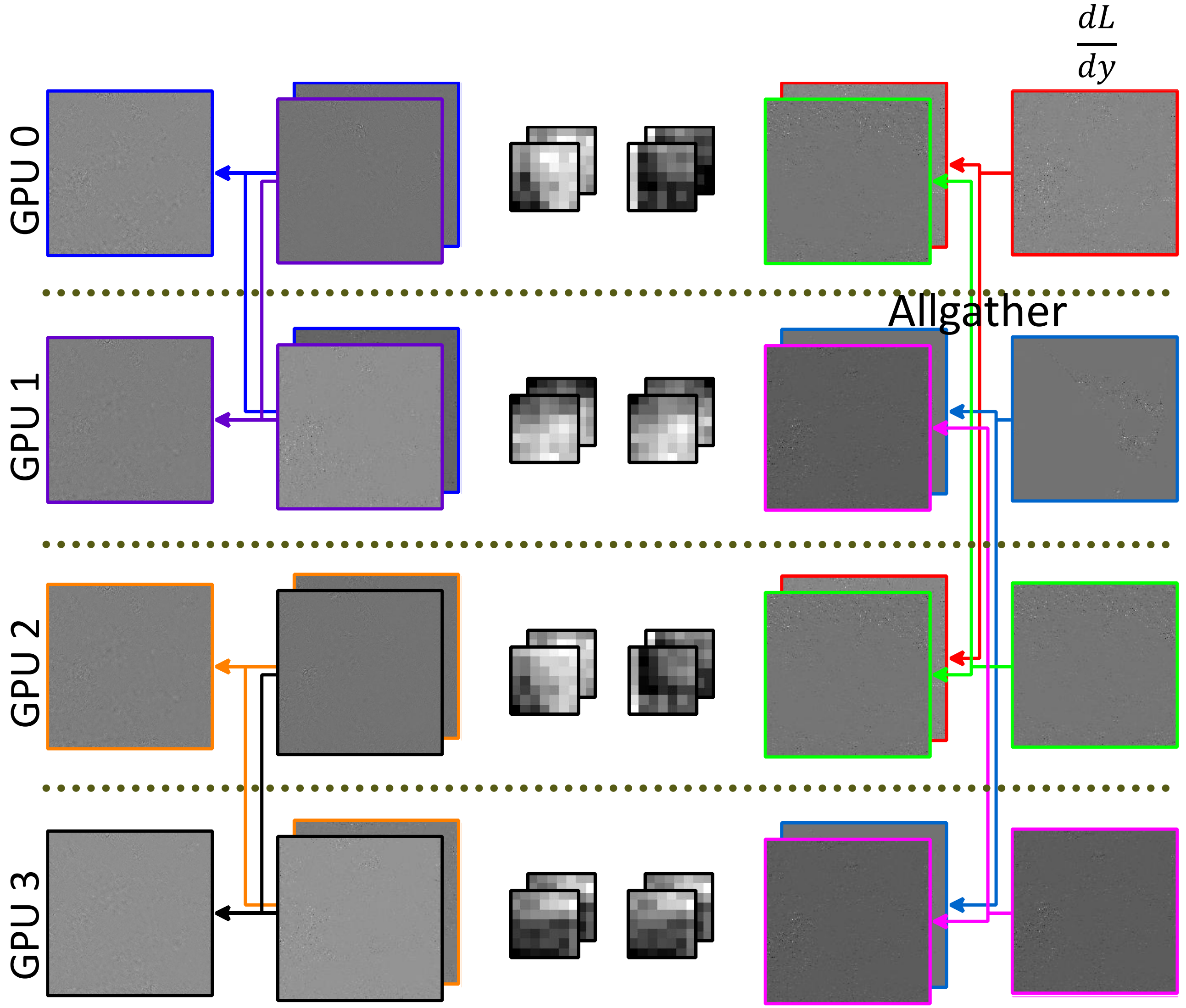
Stationary-w: Backward



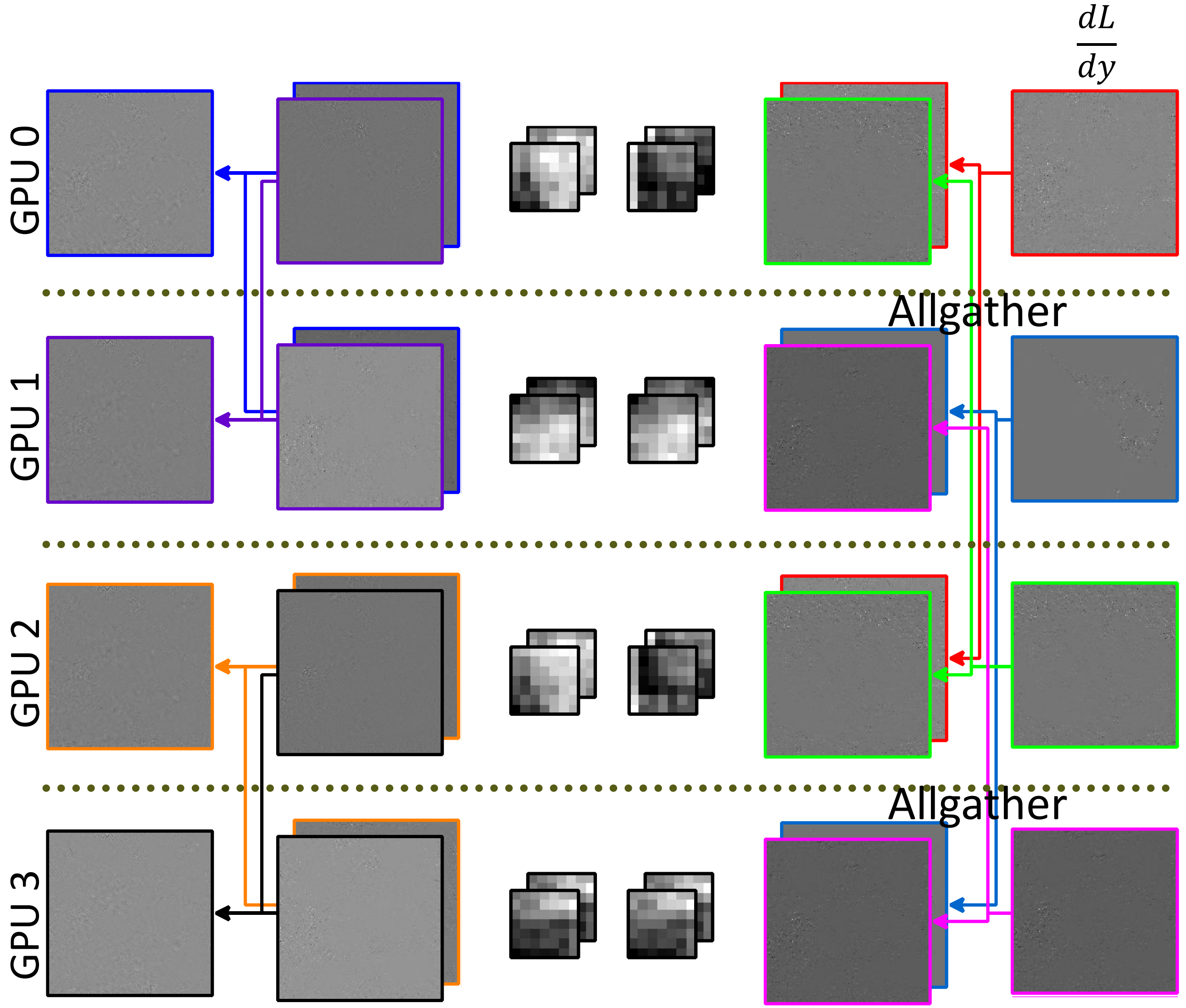
Stationary-w: Backward



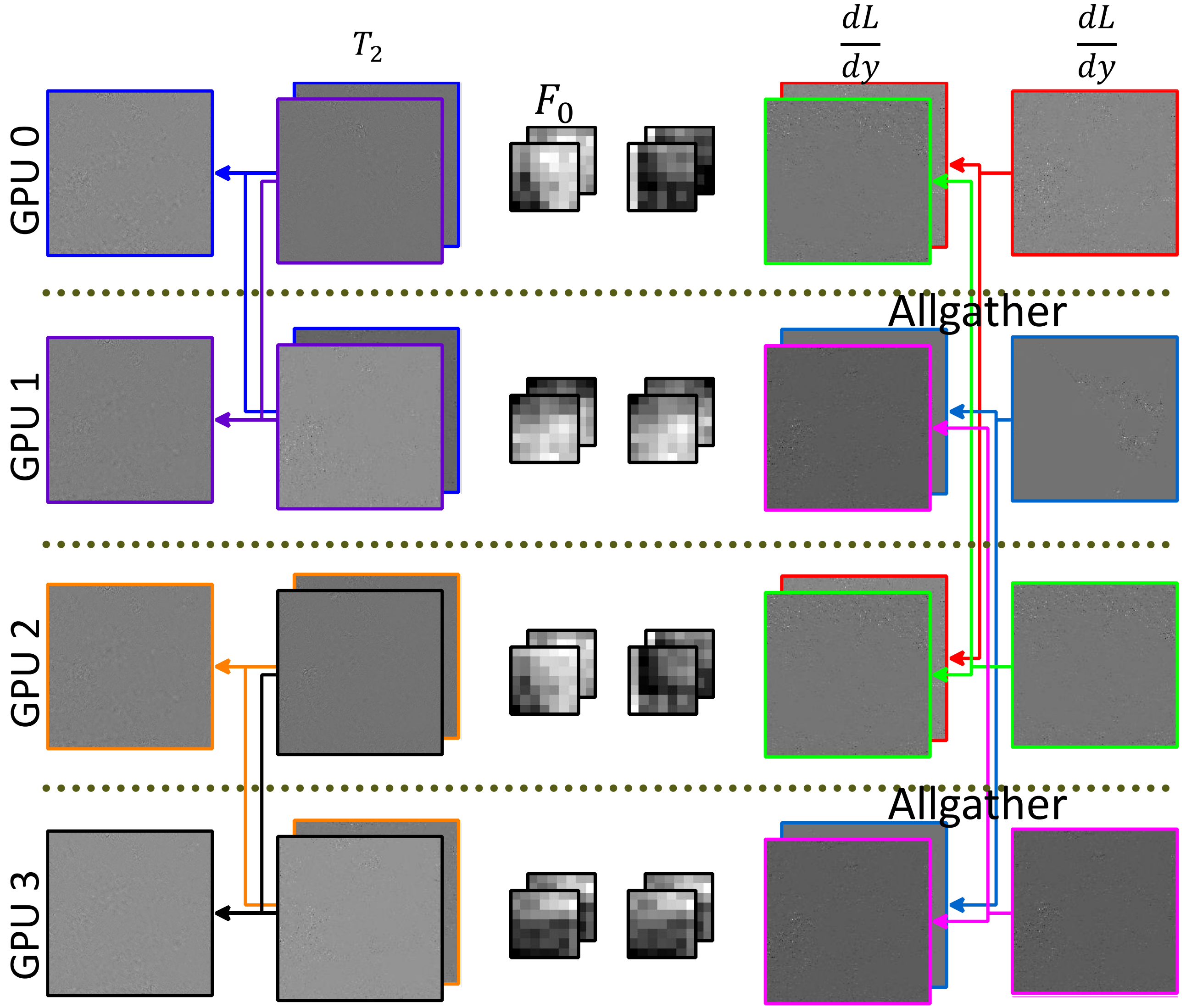
Stationary-w: Backward



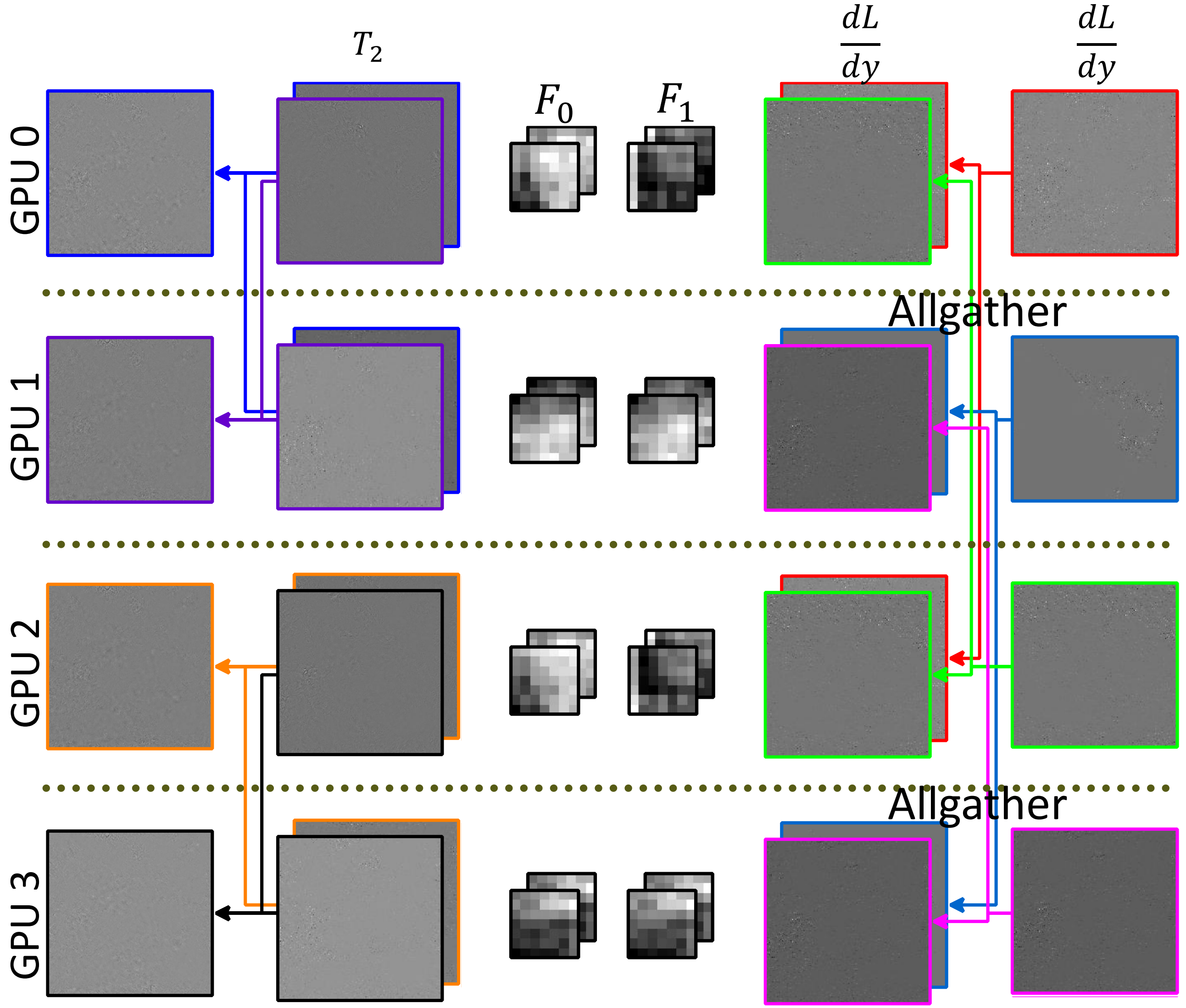
Stationary-w: Backward



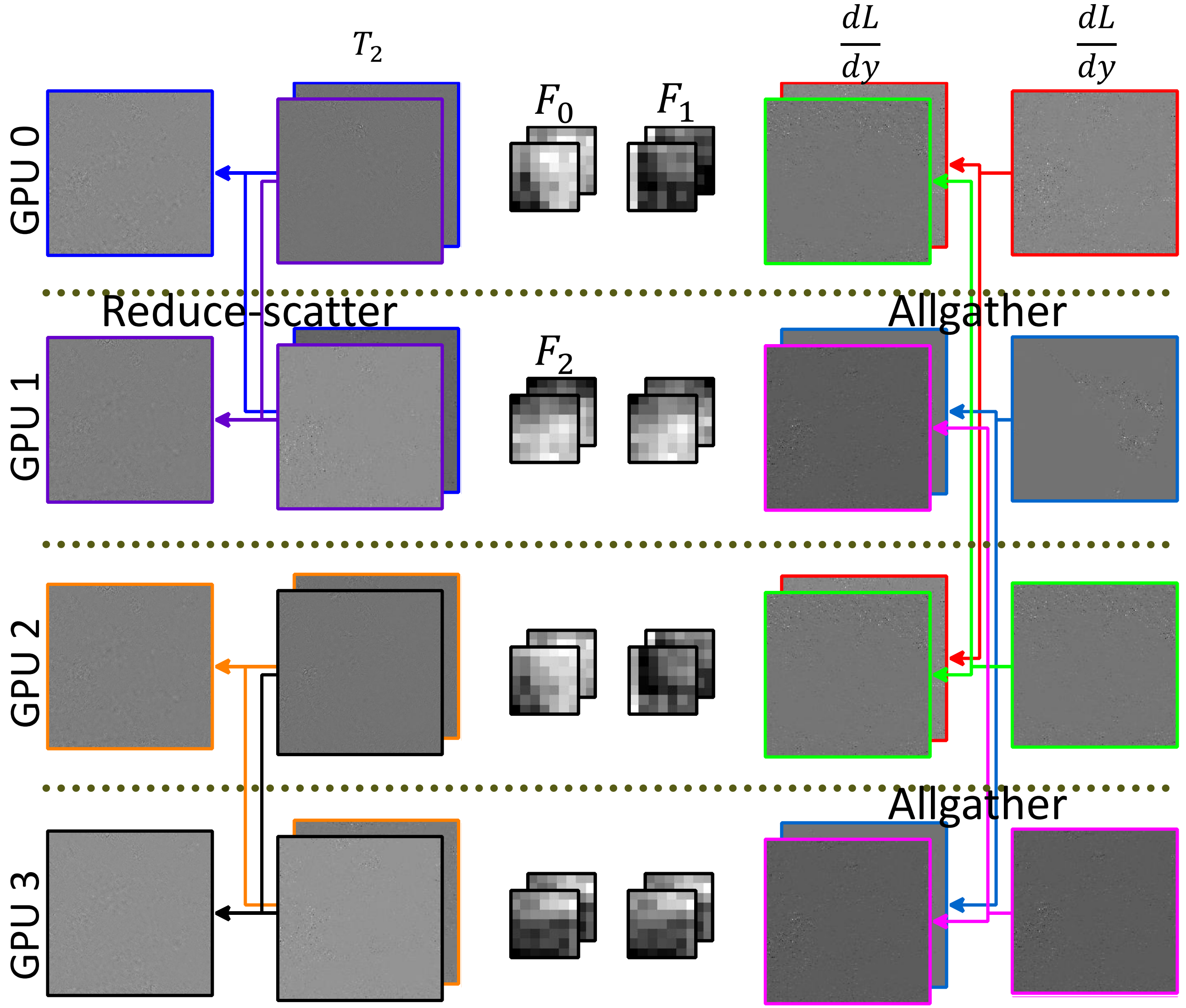
Stationary-w: Backward



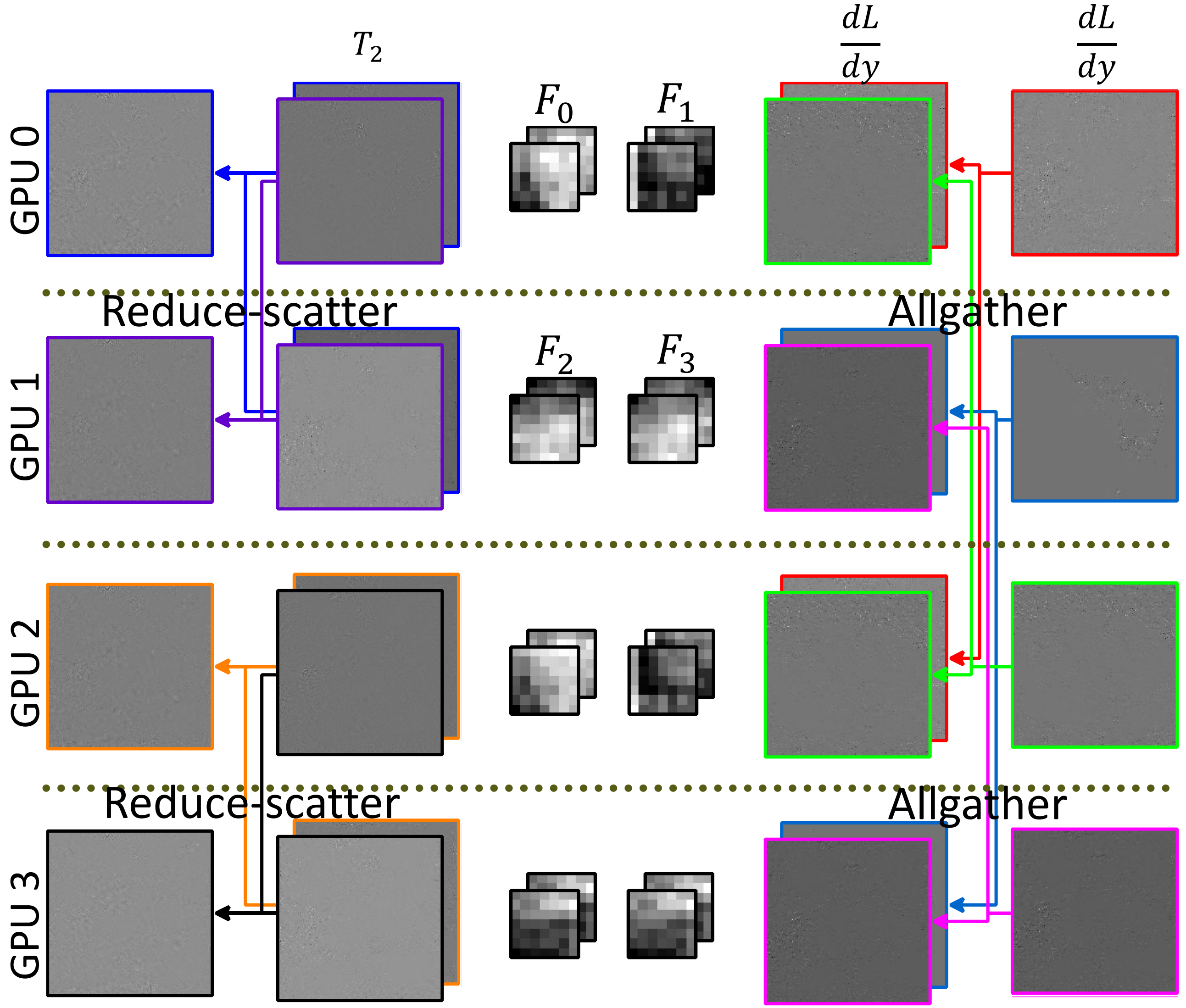
Stationary-w: Backward



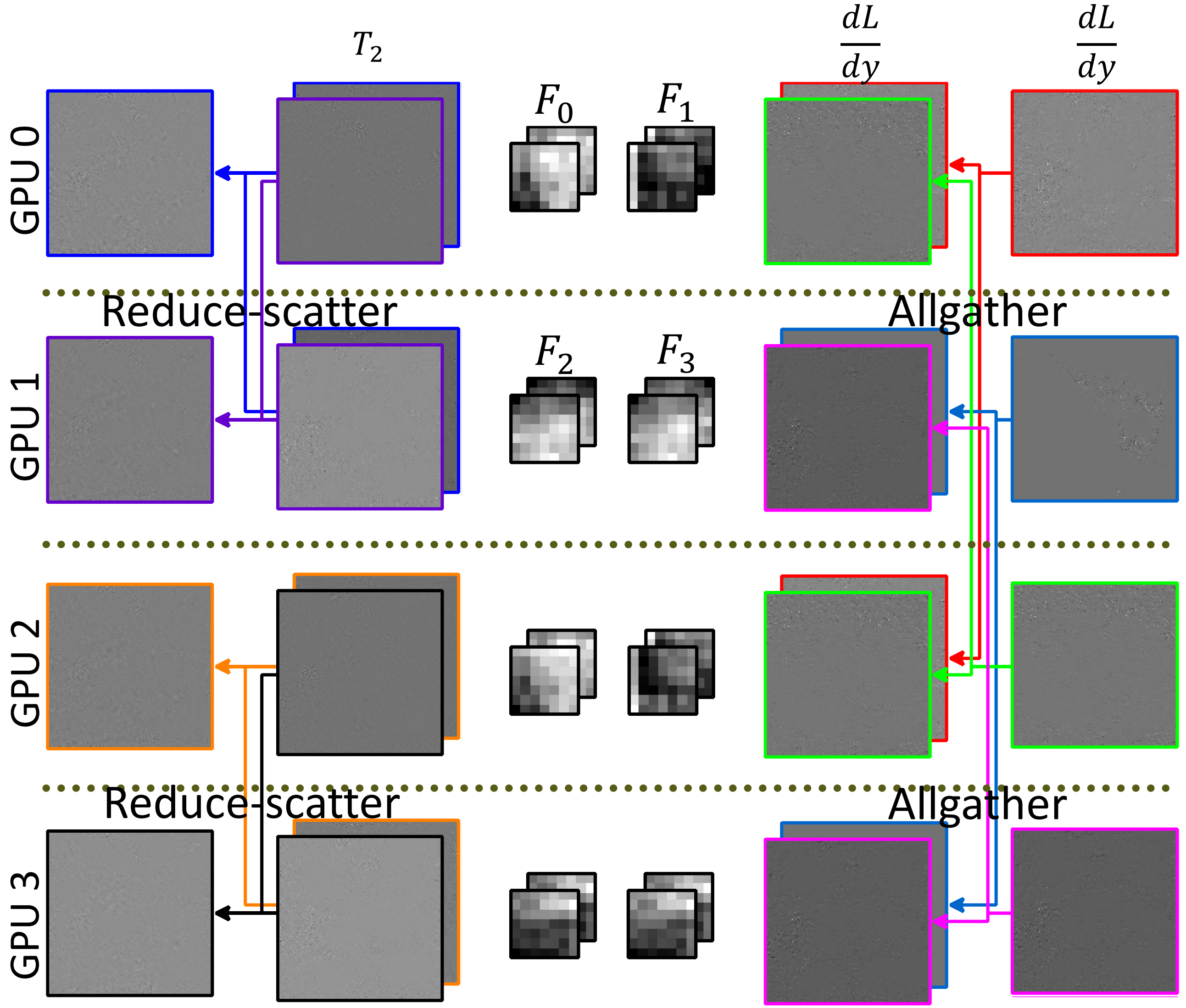
Stationary-w: Backward



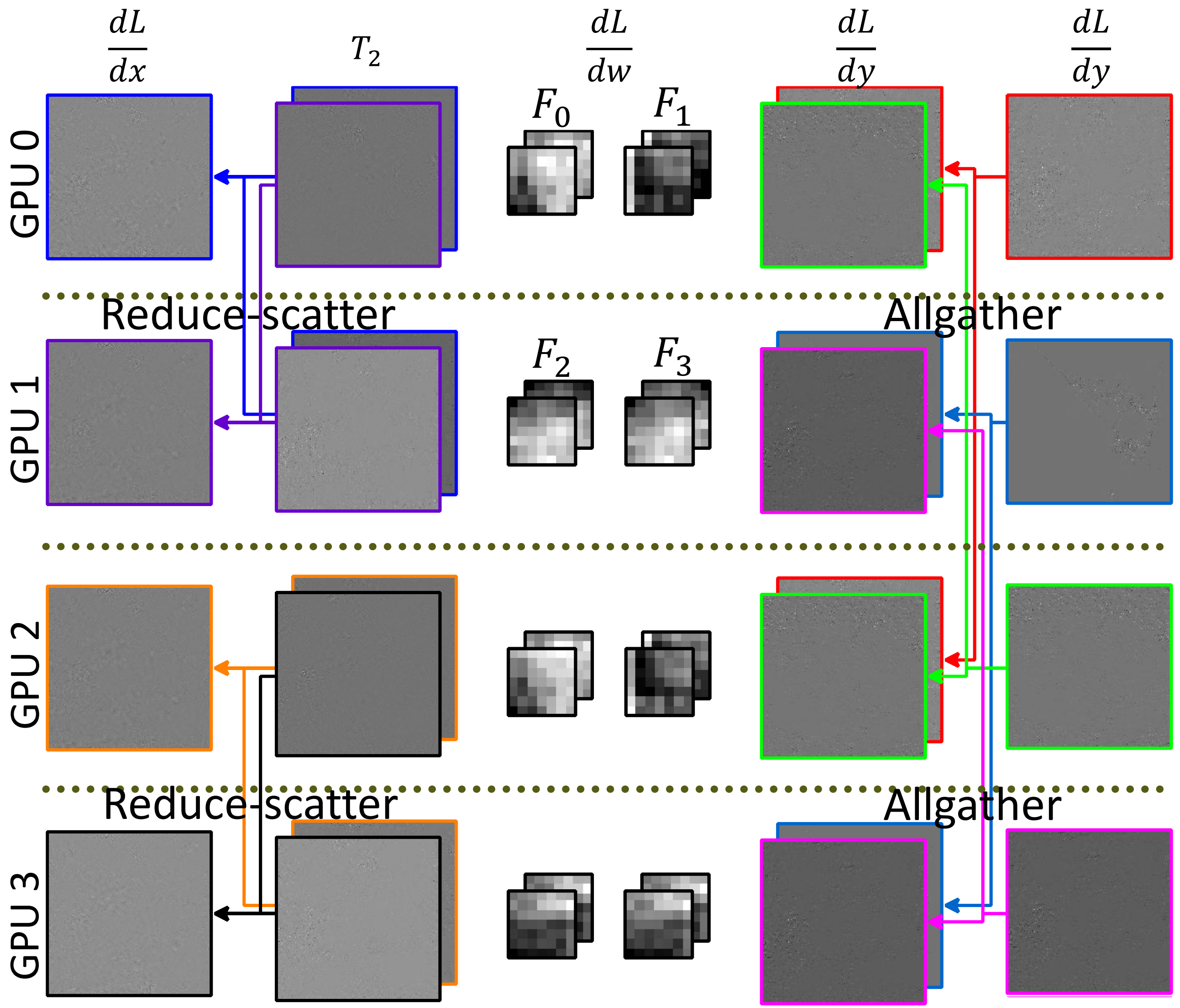
Stationary-w: Backward



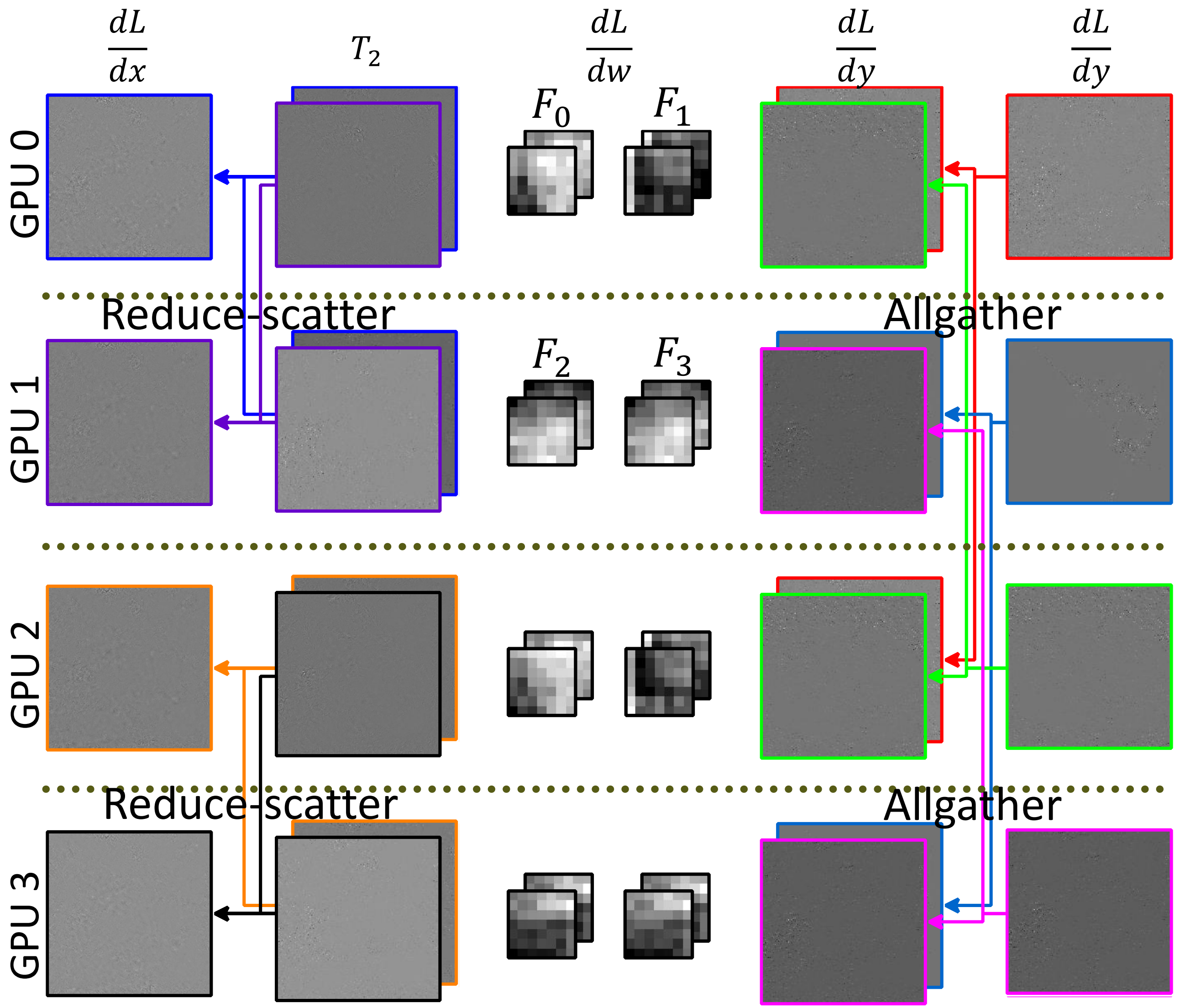
Stationary-w: Backward



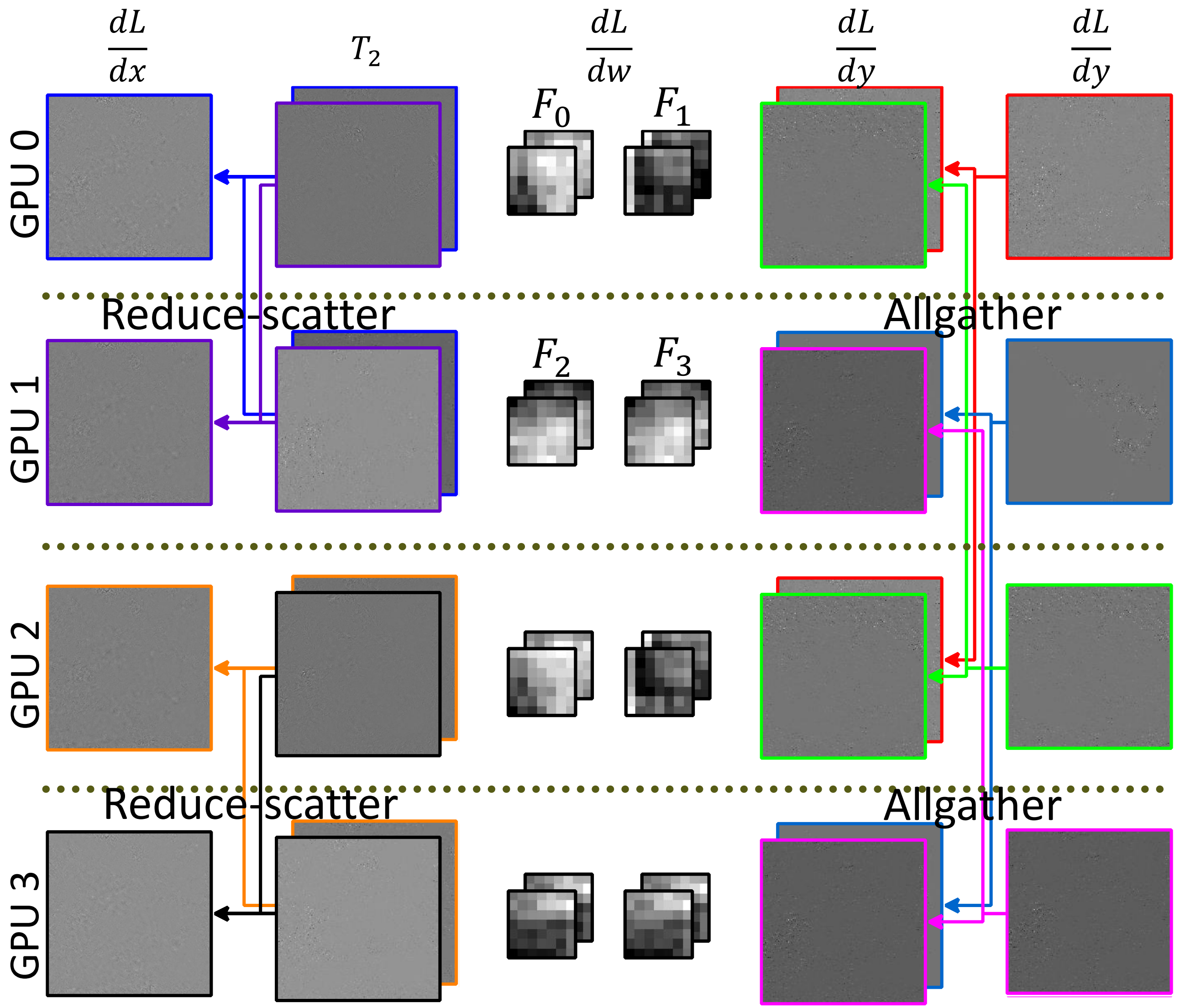
Stationary-w: Backward



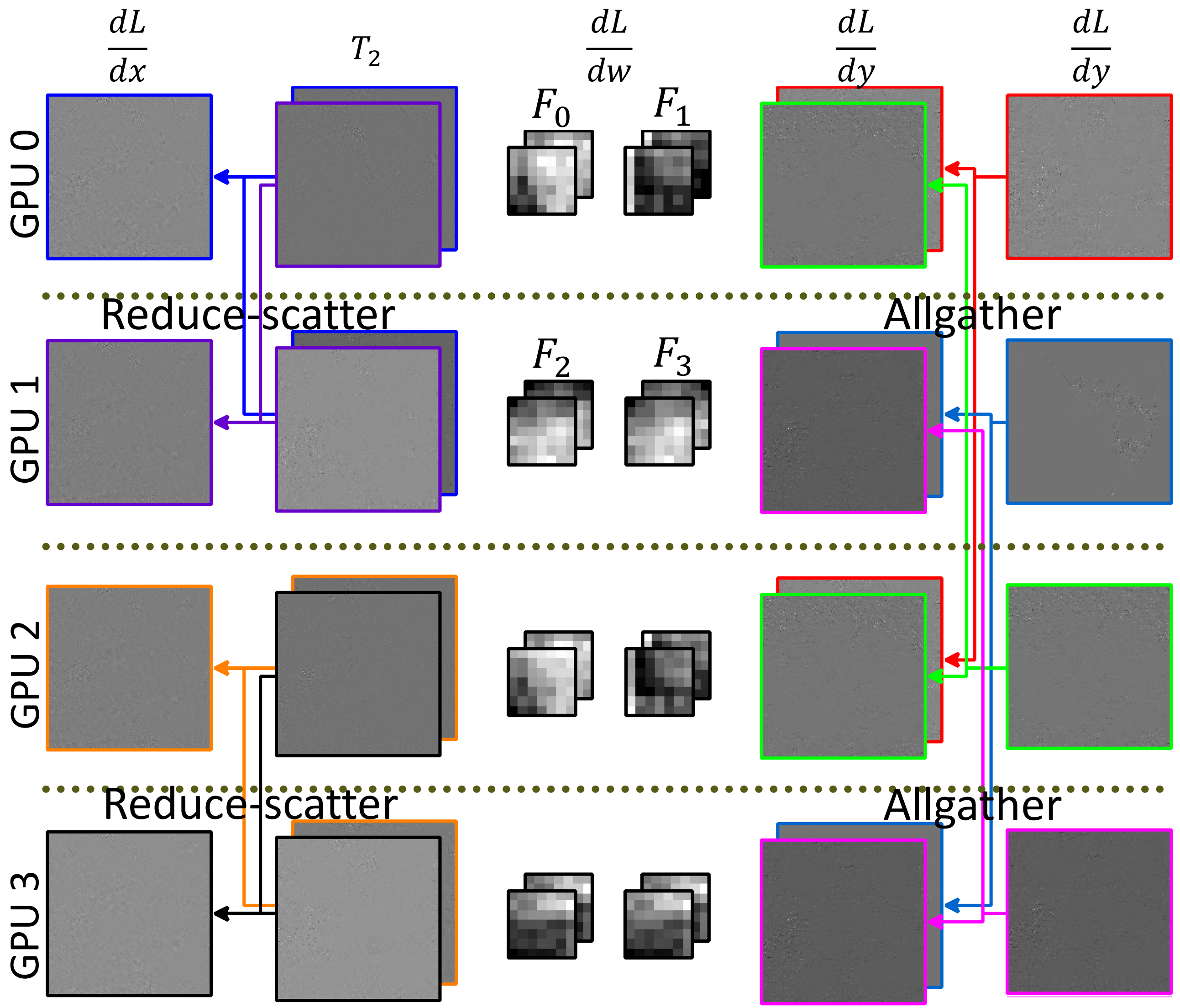
Stationary-w: Backward



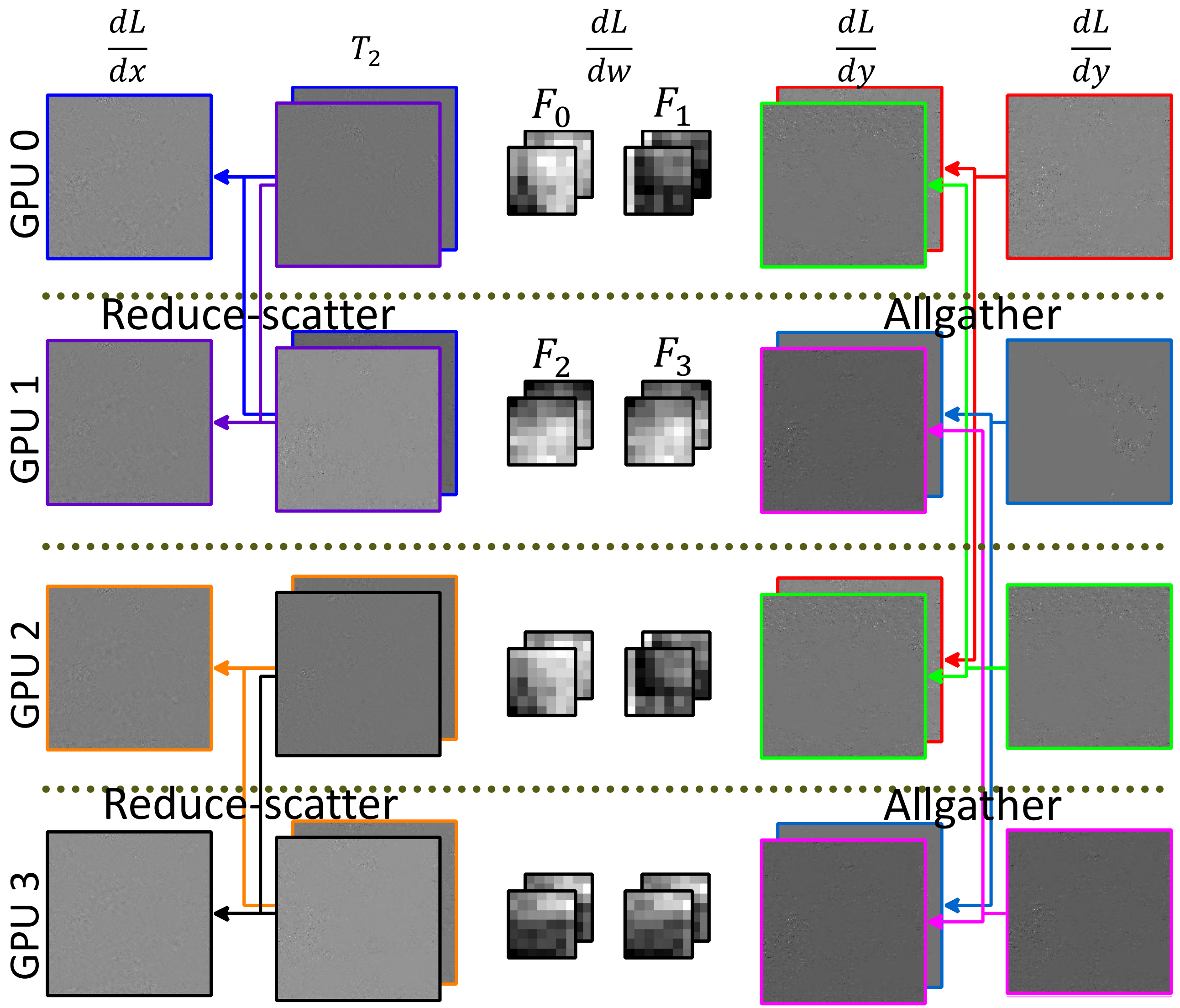
Stationary-w: Backward



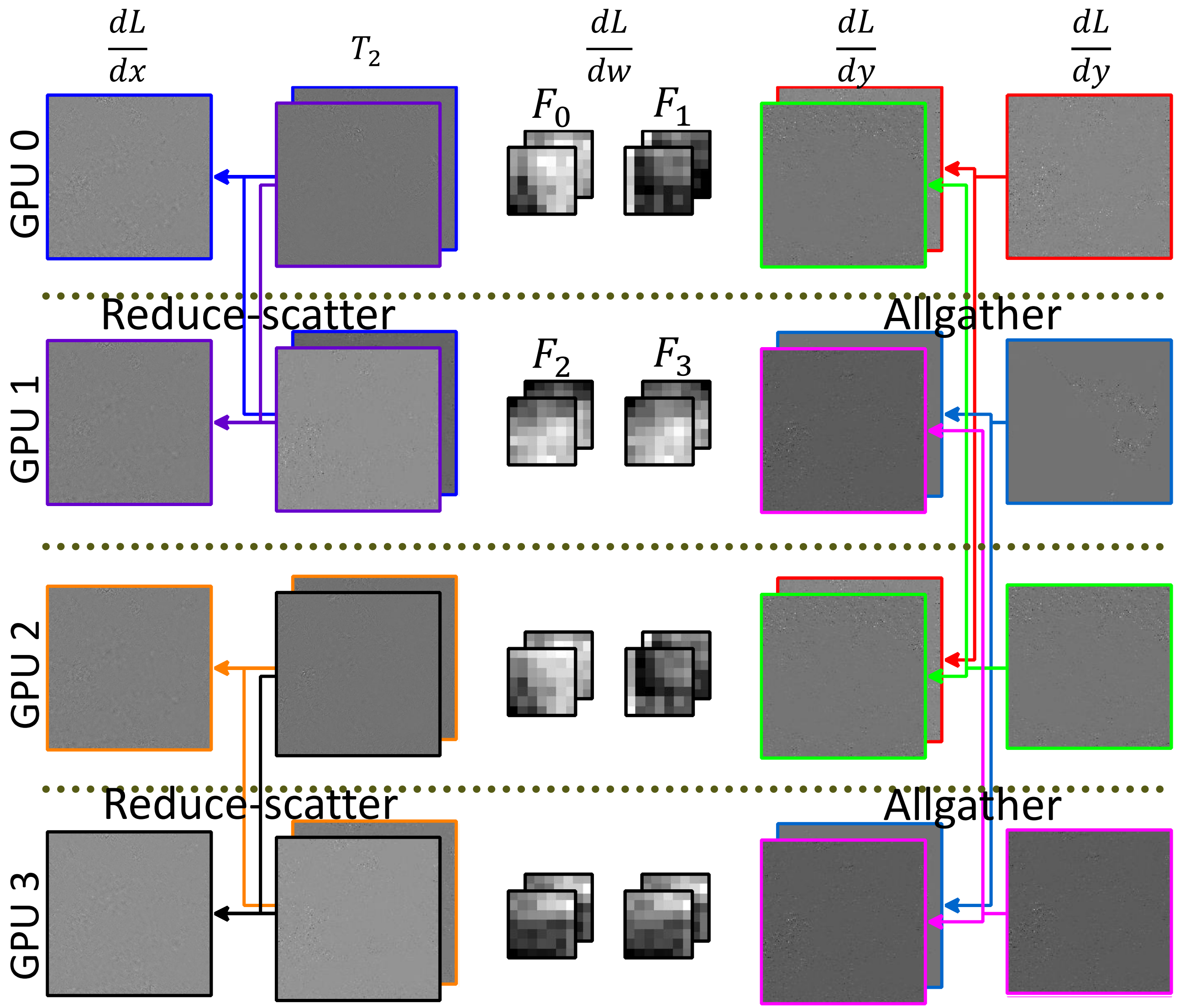
Stationary-w: Backward



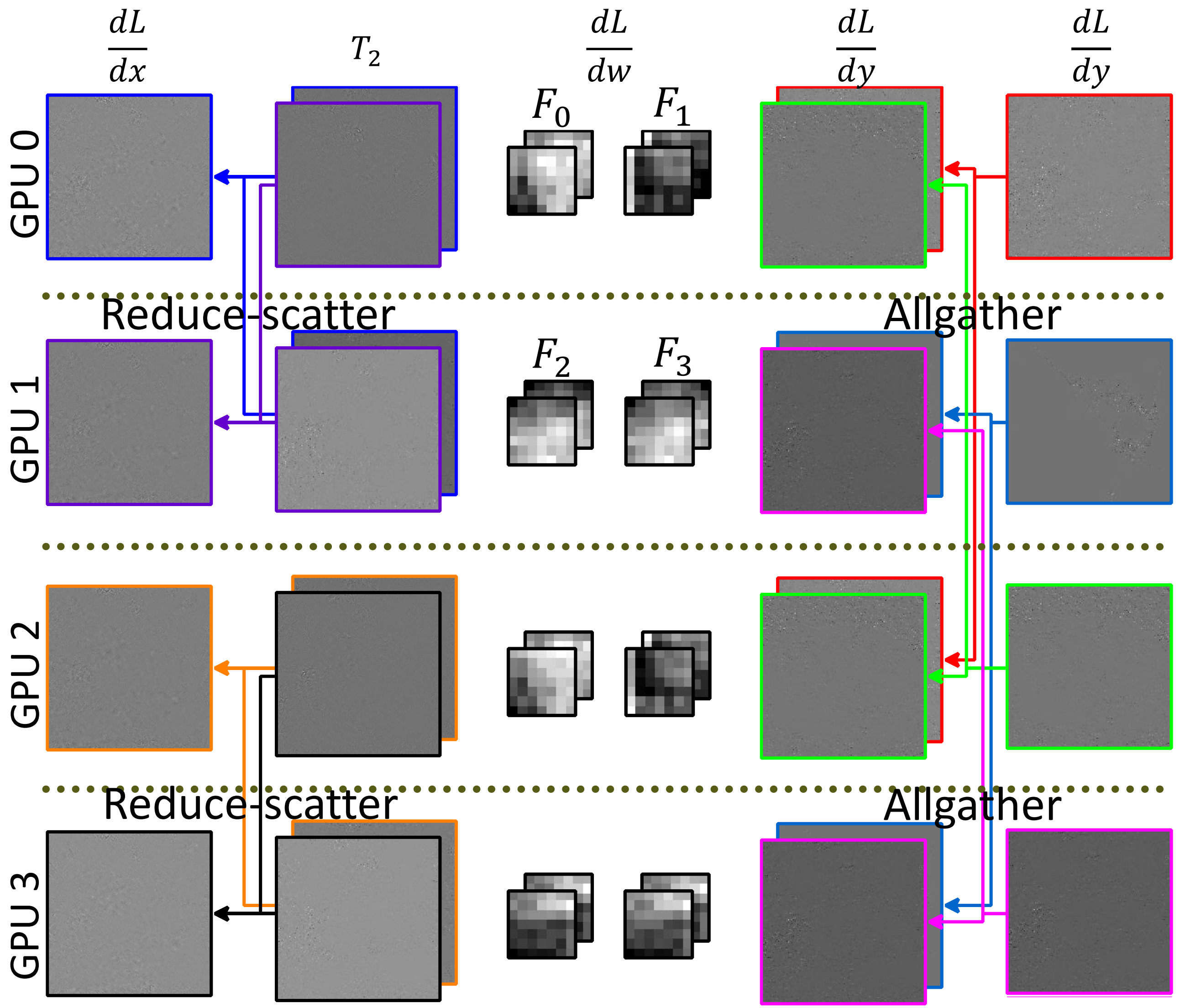
Stationary-w: Backward



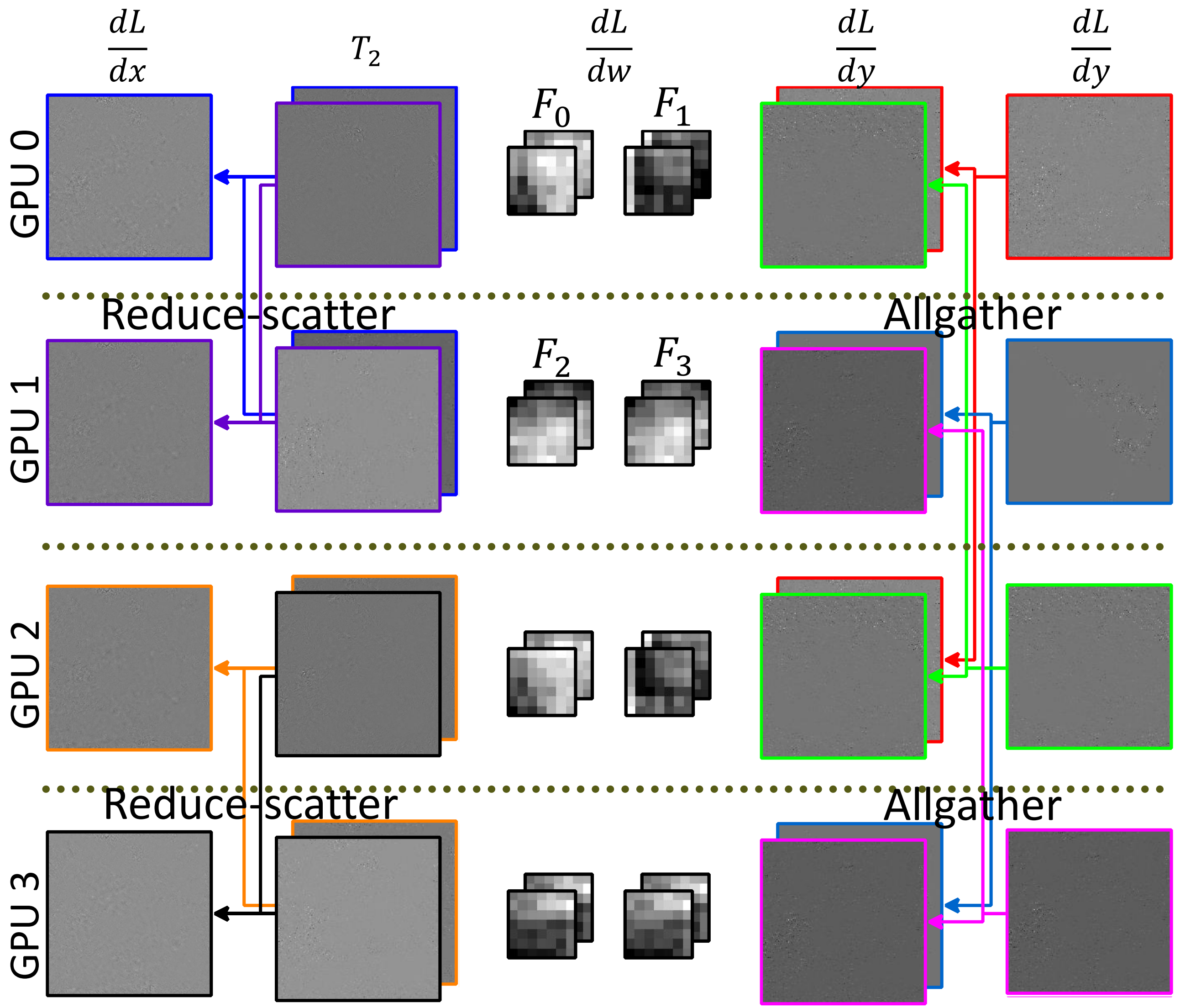
Stationary-w: Backward



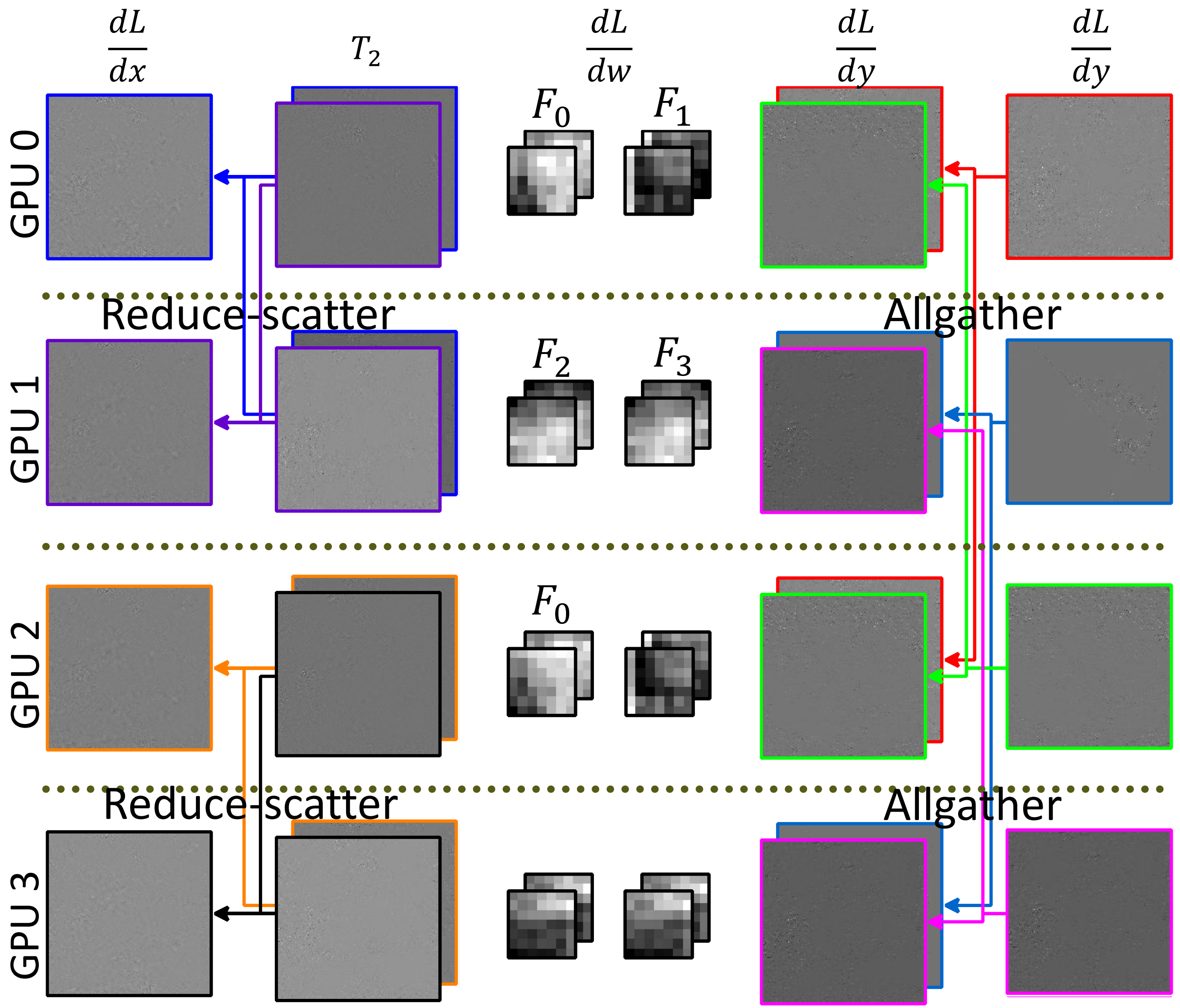
Stationary-w: Backward



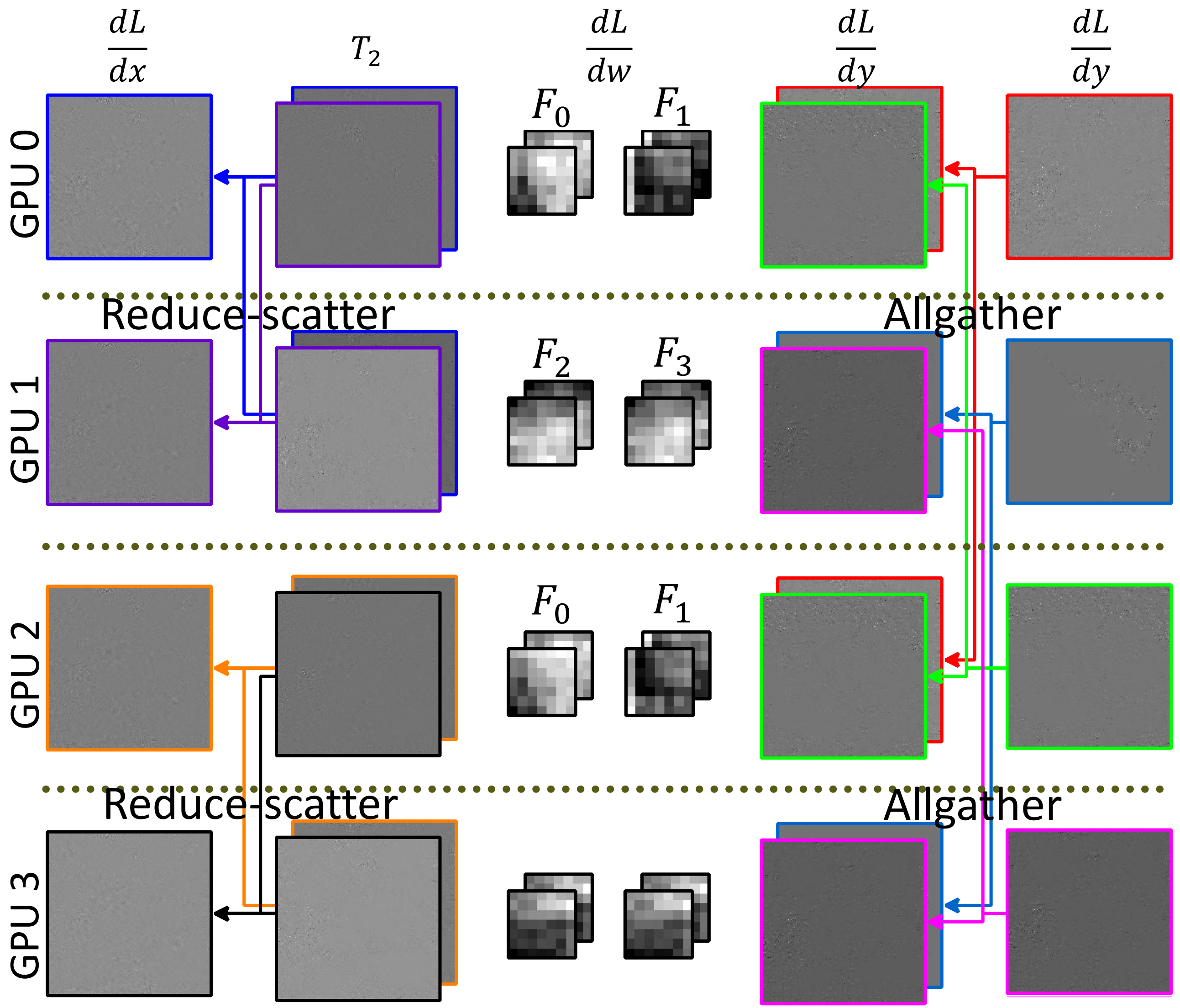
Stationary-w: Backward



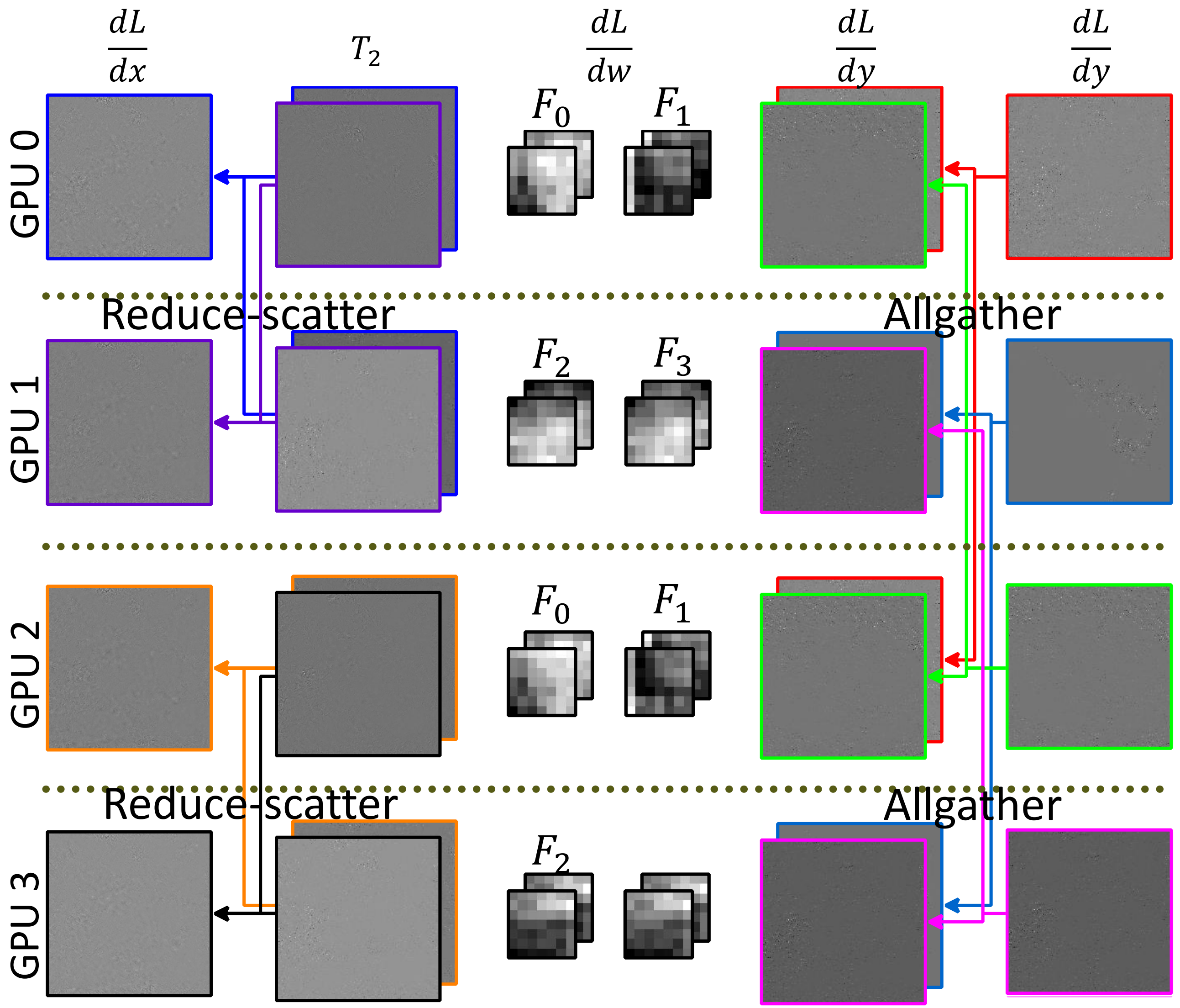
Stationary-w: Backward



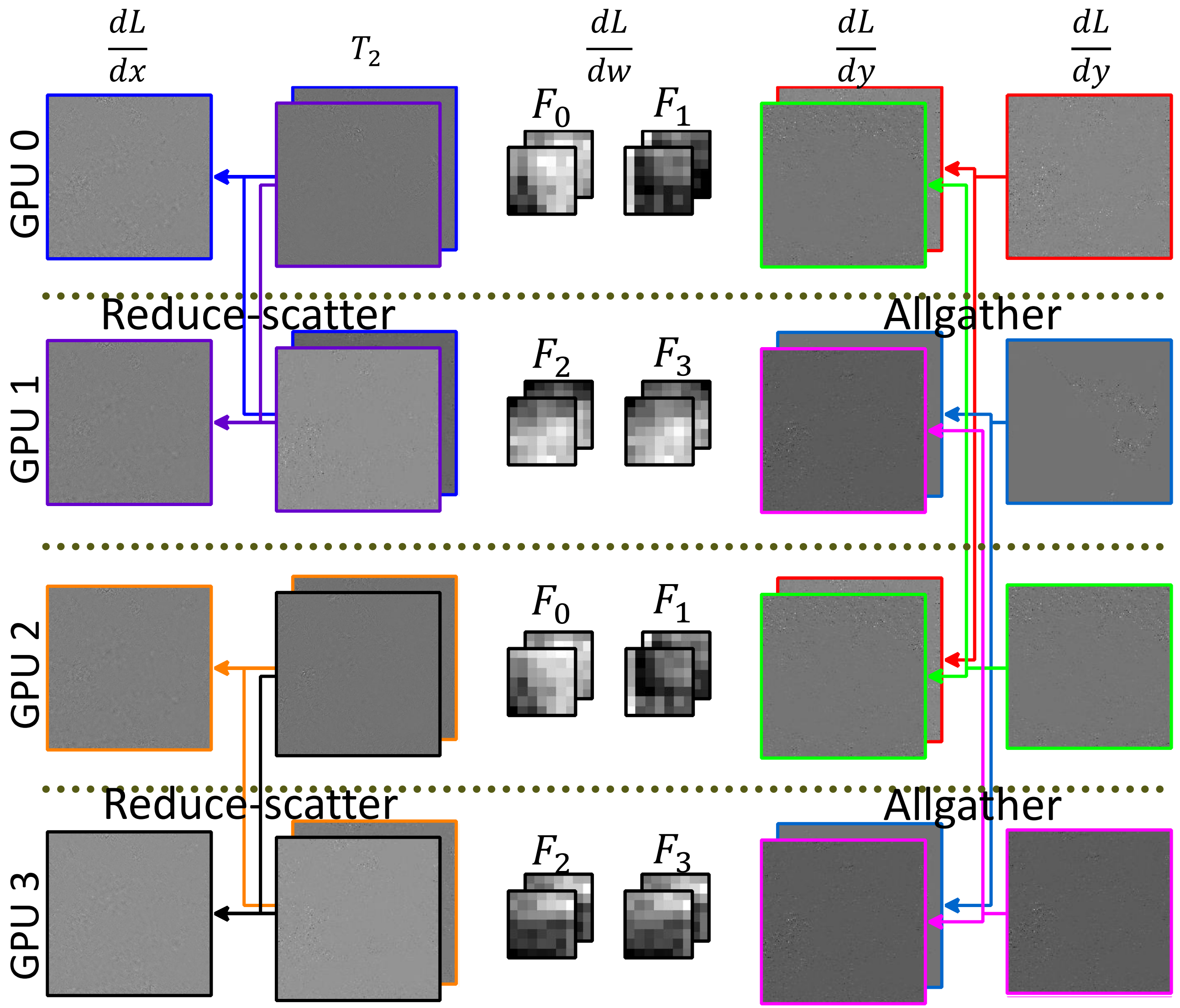
Stationary-w: Backward



Stationary-w: Backward

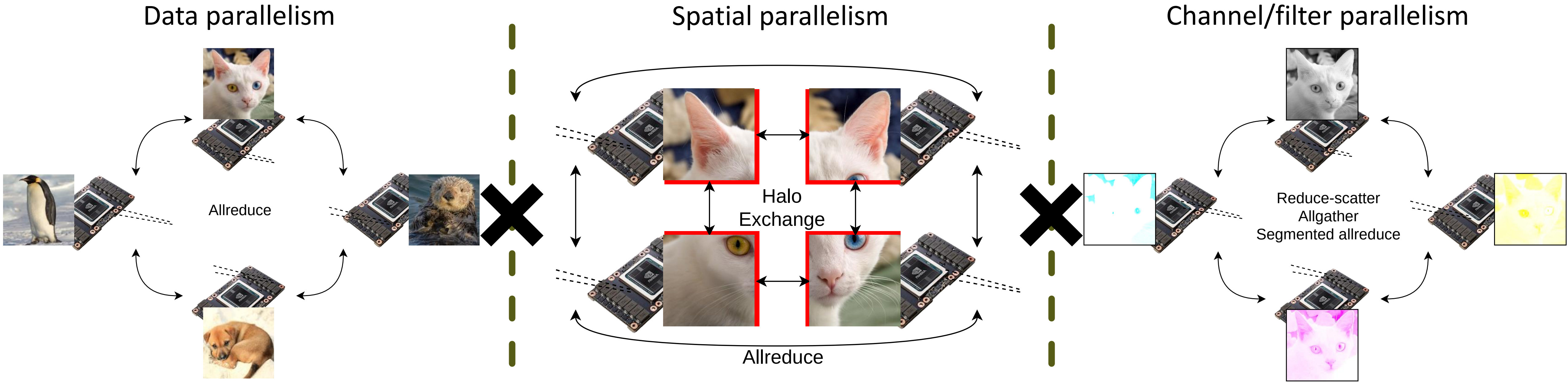


Stationary-w: Backward



General distributed convolution

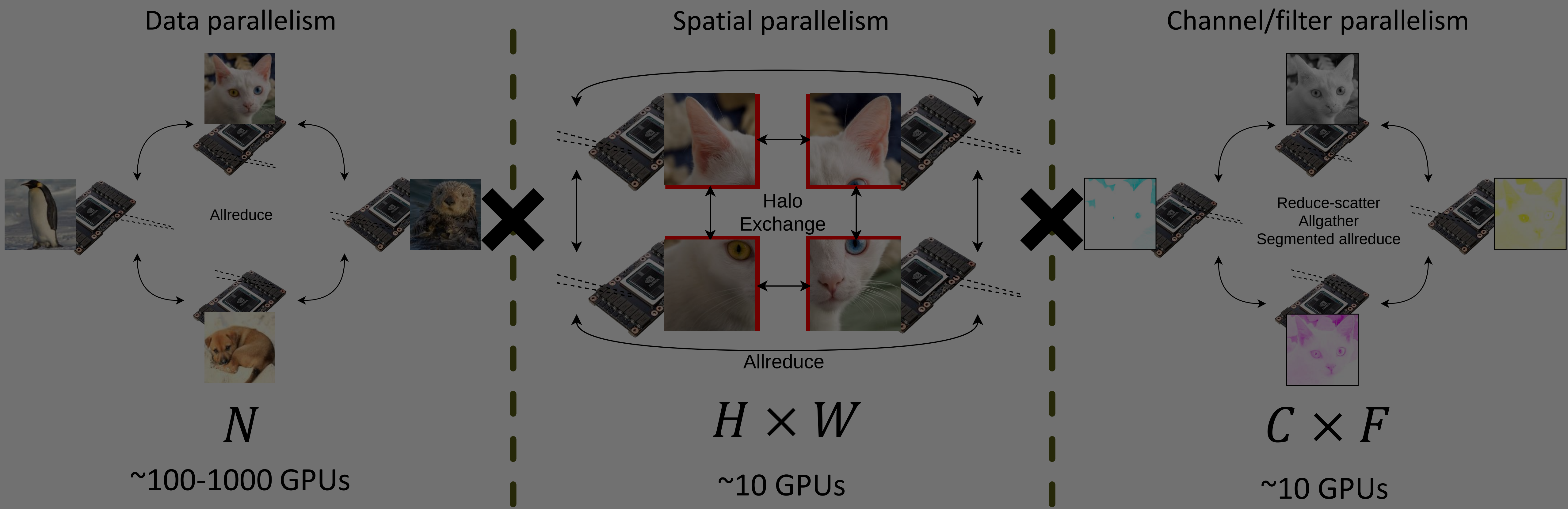
- **Provide a variety of options to enable and improve strong and weak scaling**
- **Support a full spectrum of data and model types**



Data parallelism	Spatial parallelism	Channel/filter parallelism
N	$H \times W$	$C \times F$
~100-1000 GPUs	~10 GPUs	~10 GPUs

General distributed convolution

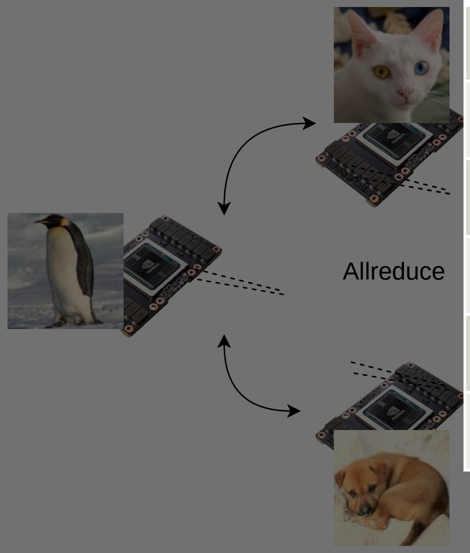
- **Provide a variety of options to enable and improve strong and weak scaling**
- **Support a full spectrum of data and model types**



General distributed convolution

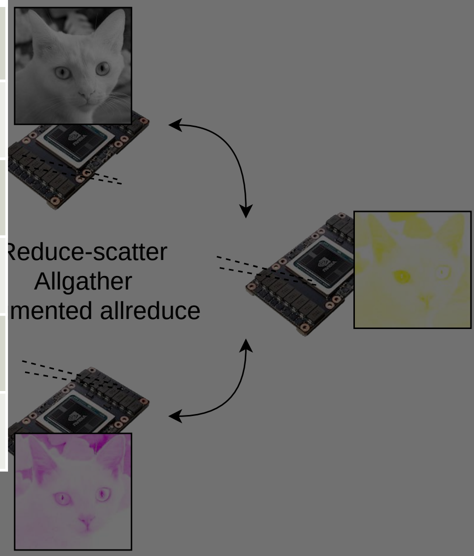
- Provide a variety of options to enable and improve strong and weak scaling
- Support a full spectrum of data and model types

Data paralle



Model/Algorithm	Mini-batch	Top-1	Top-5	Runtime (min)
ResNet-50 (data)	8192	77.3%	93.6%	34.1
+ spatial + 4-way x, y				19.9 (1.7x)
WRN-50-2 (data)	4096	78.4%	94.3%	106.9
+ spatial + 8-way x, y				45.5 (2.3x)
WRN-50-4 (data)	2048	80.0%	95.1%	432.3
+ spatial + 4×2 w				105.0 (4.1x)

ilter parallelism



Allreduce

N

~100-1000 GPUs

$H \times W$

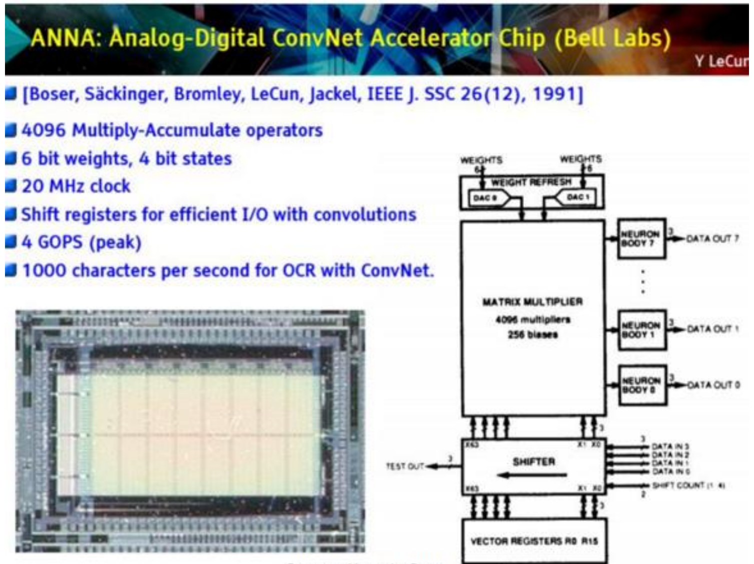
~10 GPUs

$C \times F$

~10 GPUs

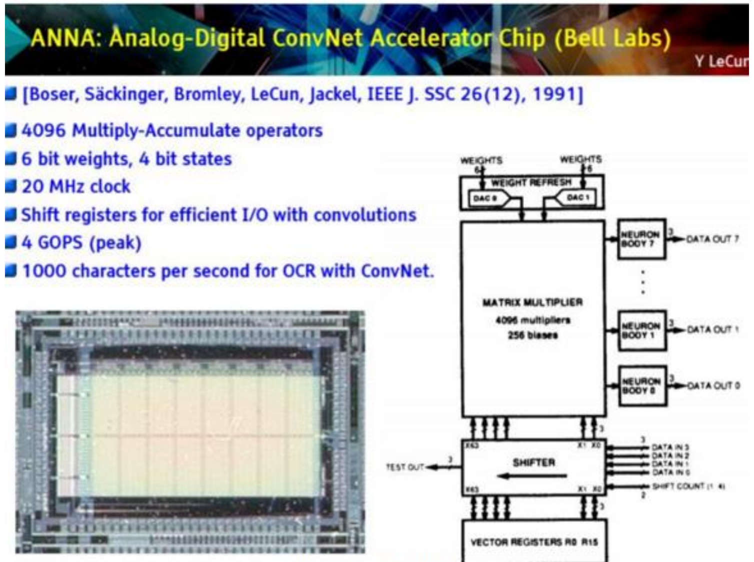
Specialized hardware

Specialized hardware



Source: Yann LeCun

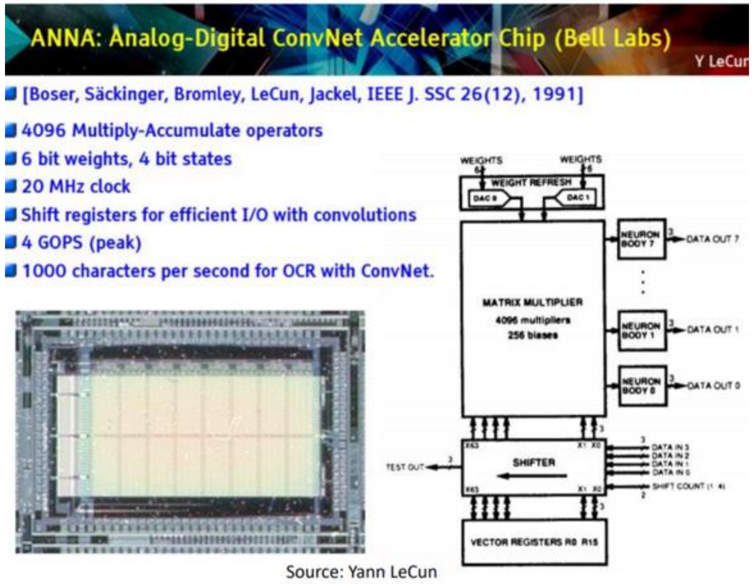
Specialized hardware

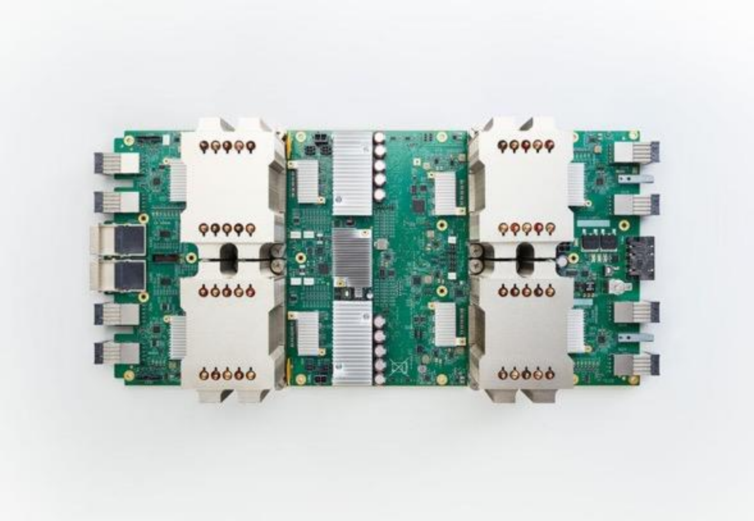


Source: Yann LeCun

ANNA [1991]

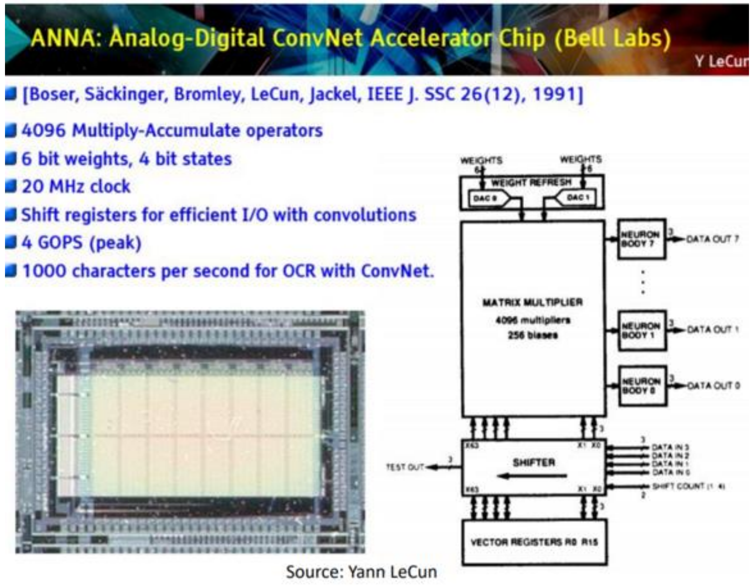
Specialized hardware



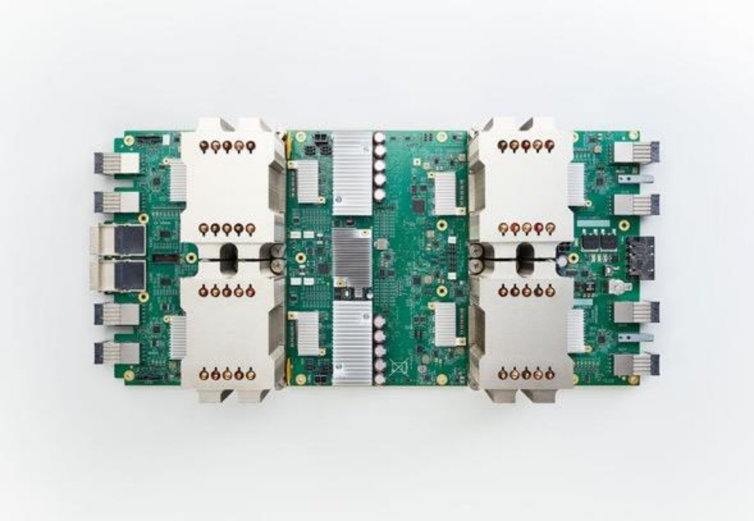


ANNA [1991]

Specialized hardware

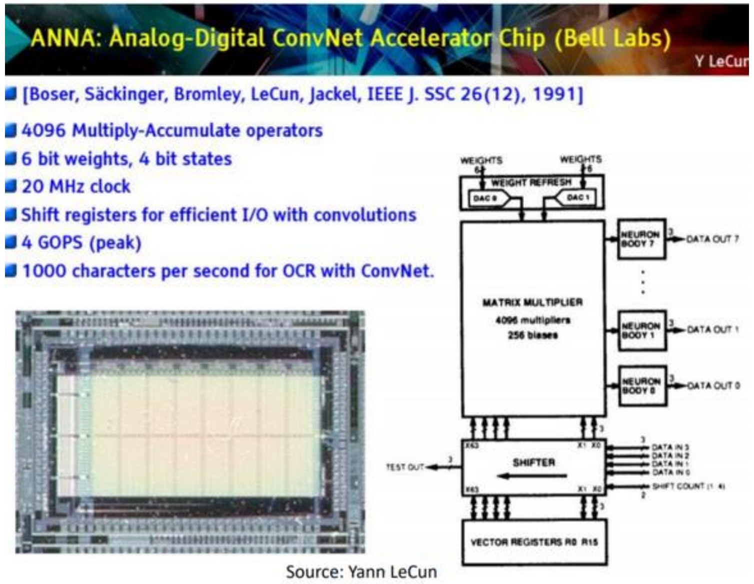


ANNA [1991]

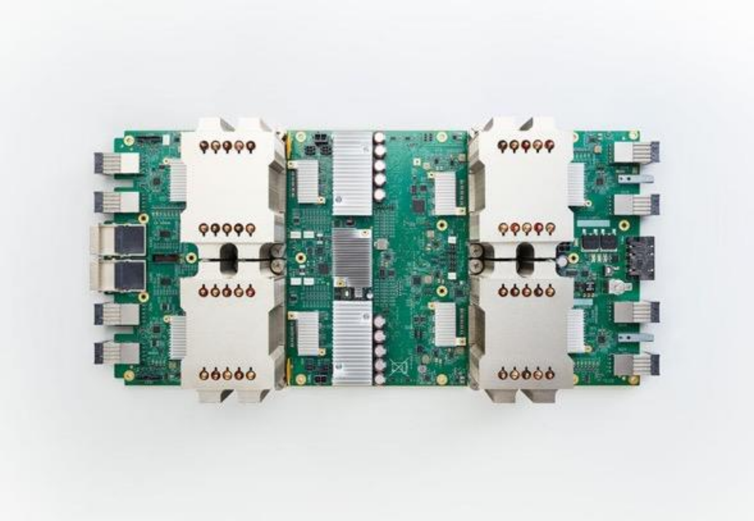


TPU v1(inference); v2-3(training)
[2016-today]

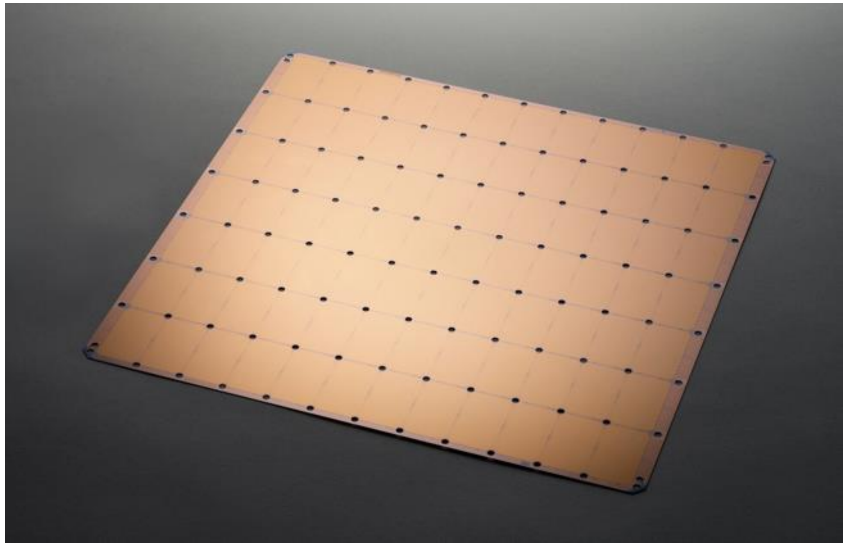
Specialized hardware



ANNA [1991]

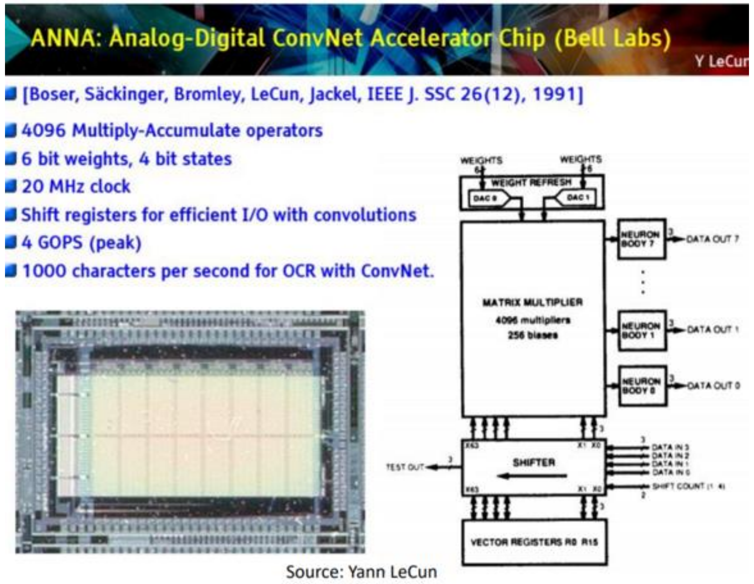


TPU v1(inference); v2-3(training) [2016-today]

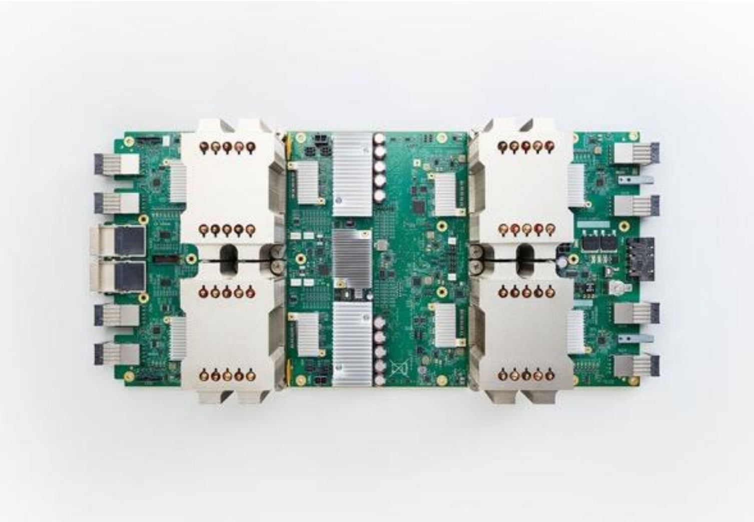


Cerebrus CS-1 [2019]

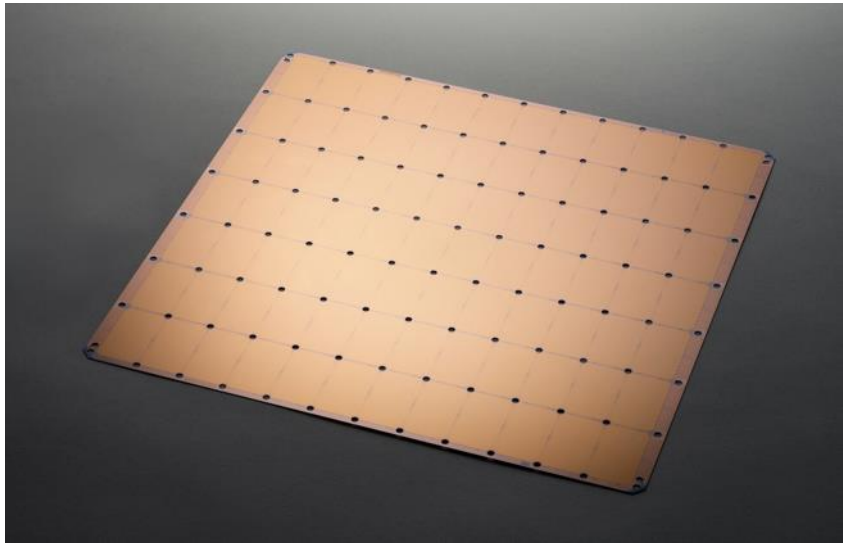
Specialized hardware



ANNA [1991]

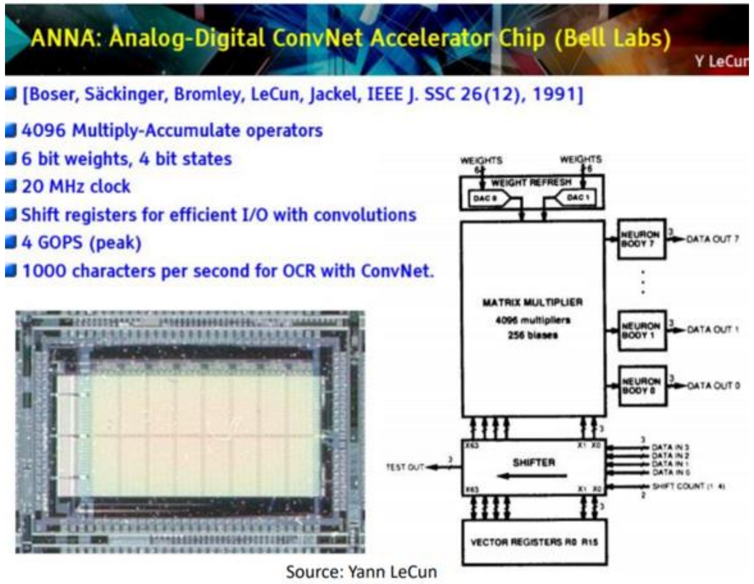


TPU v1(inference); v2-3(training) [2016-today]

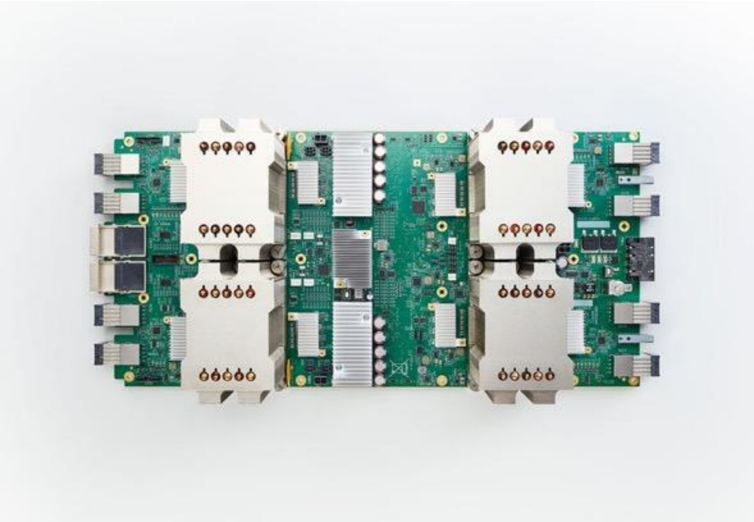


Cerebrus CS-1 [2019]

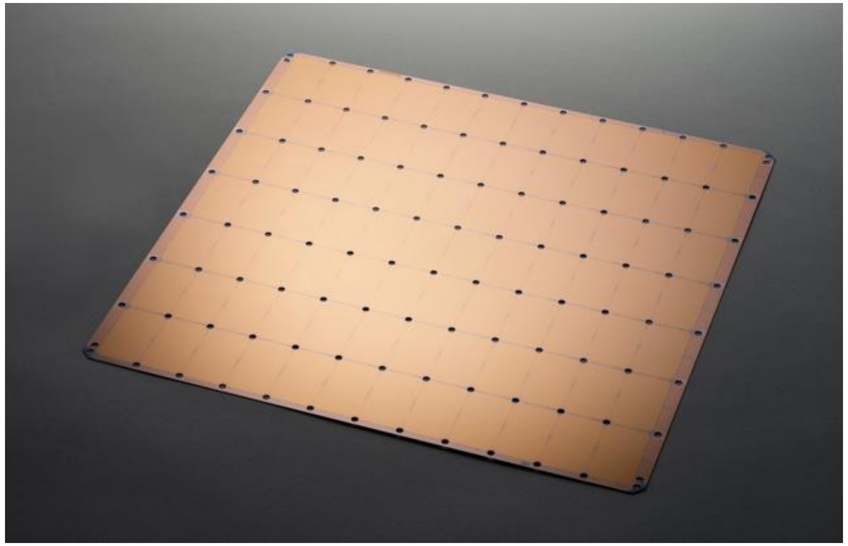
Specialized hardware



ANNA [1991]



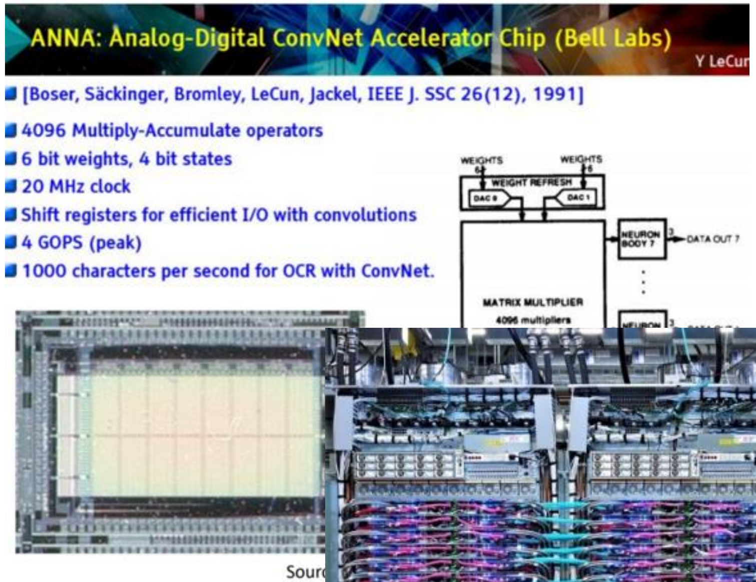
TPU v1(inference); v2-3(training) [2016-today]

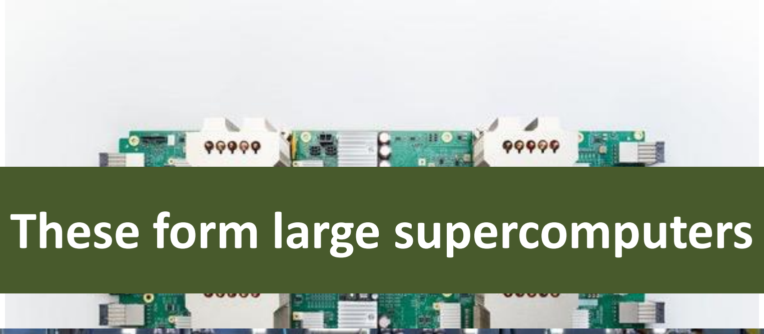


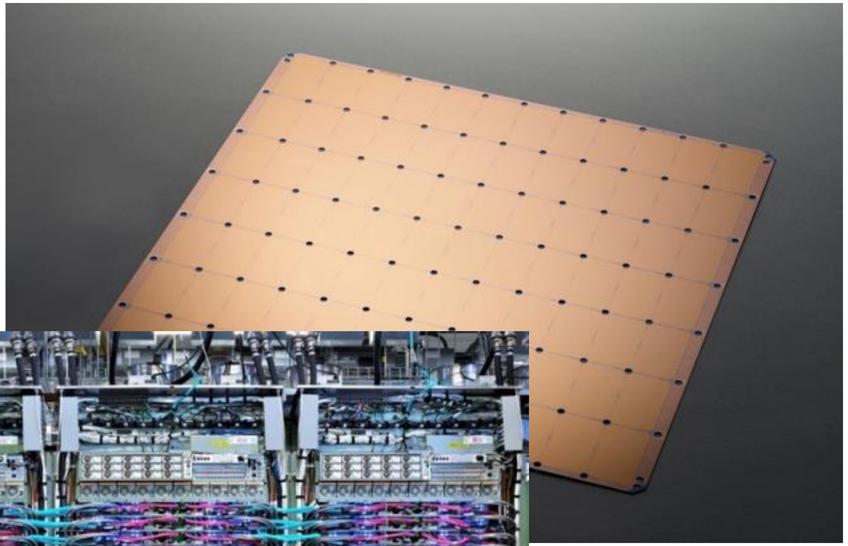
Cerebrus CS-1 [2019]

Literally hundreds of other startups in this space

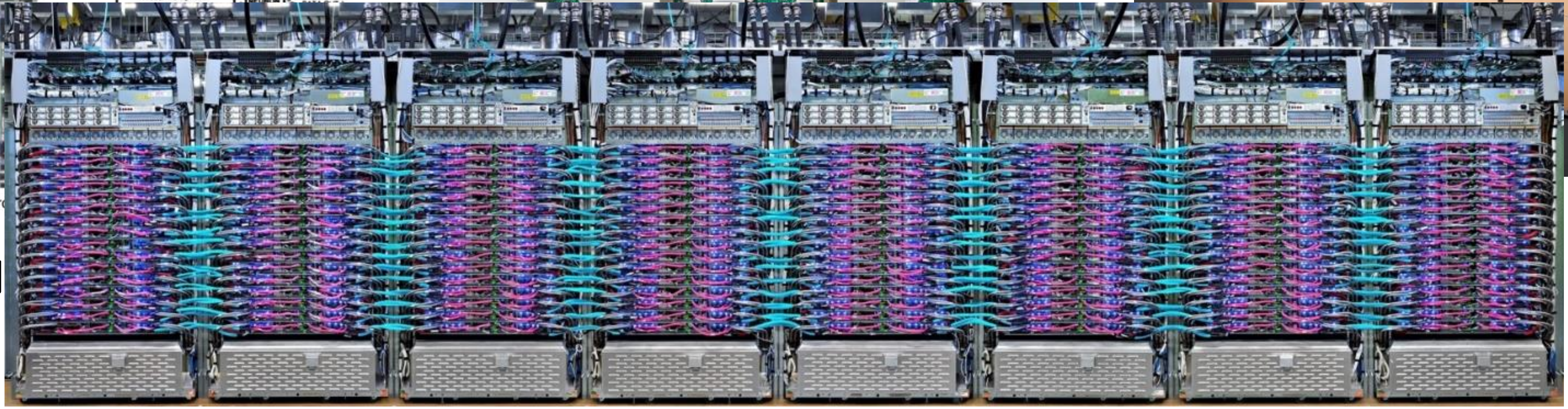
Specialized hardware







These form large supercomputers

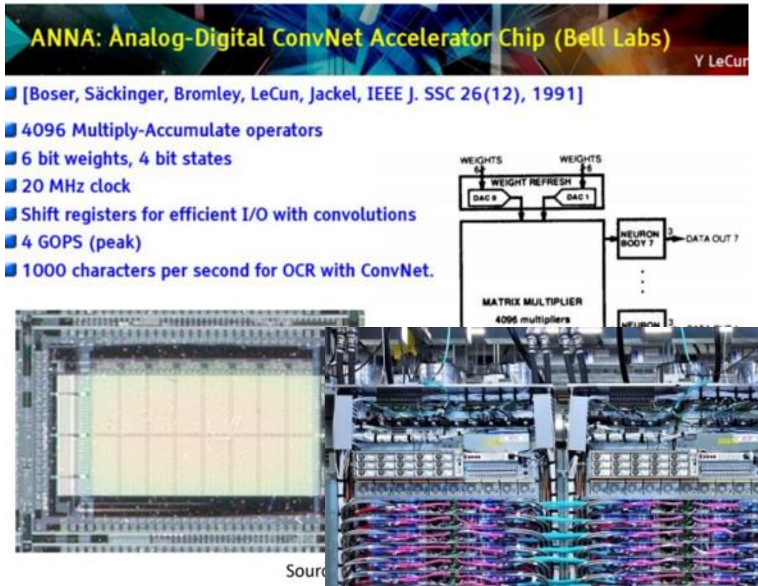


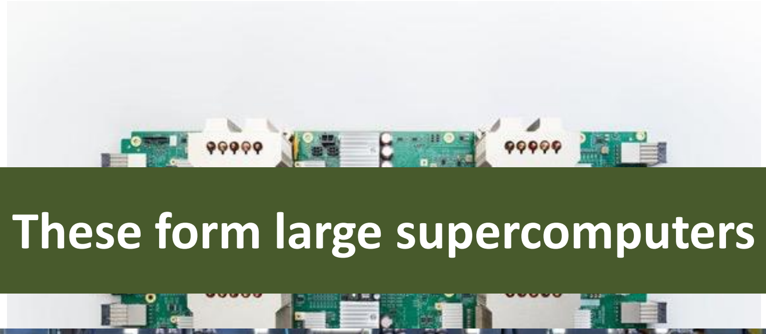
ANN

[2019]

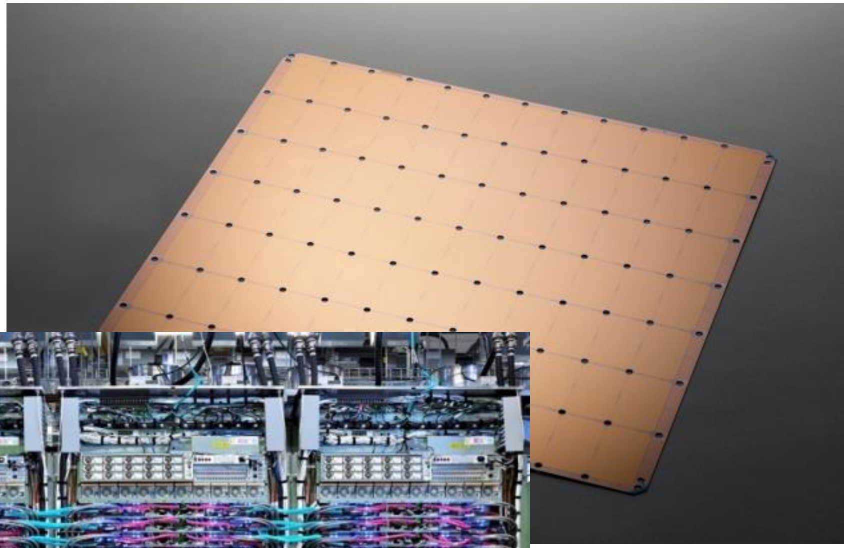
Literally hundreds of other startups in this space

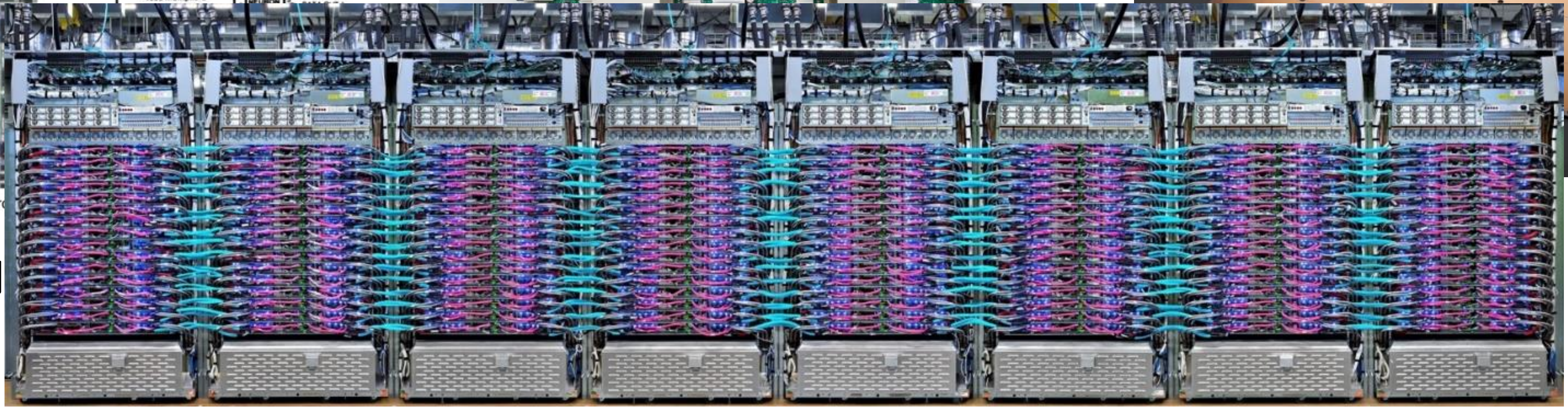
Specialized hardware





These form large supercomputers





ANN

[2019]

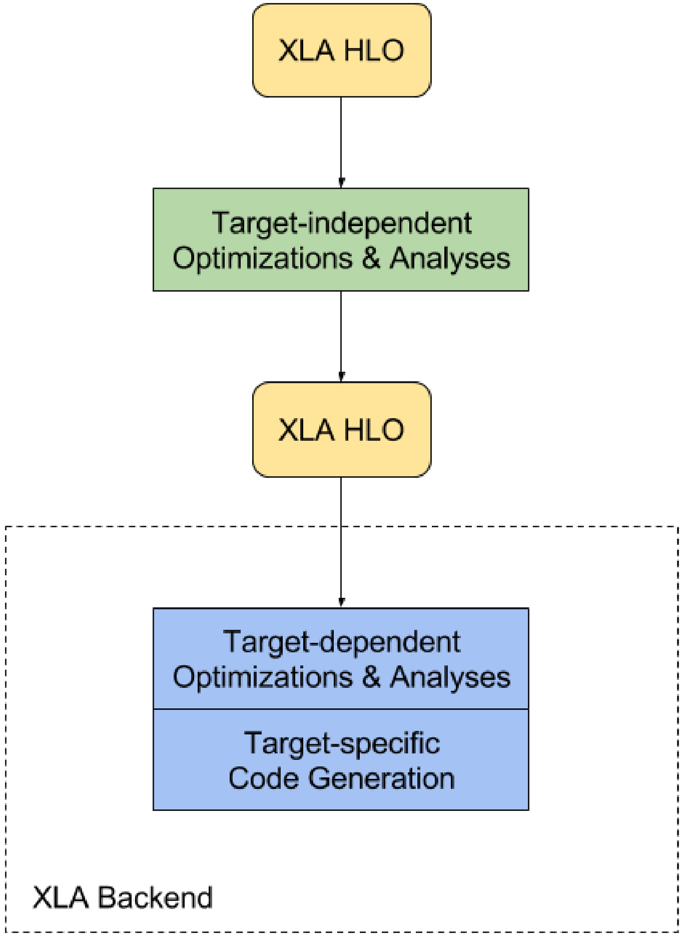
Literally hundreds of other startups in this space

DNN Compilers

- **Use techniques from compiler construction: DNN → Graph → IR → Transformations → HW Mapping**

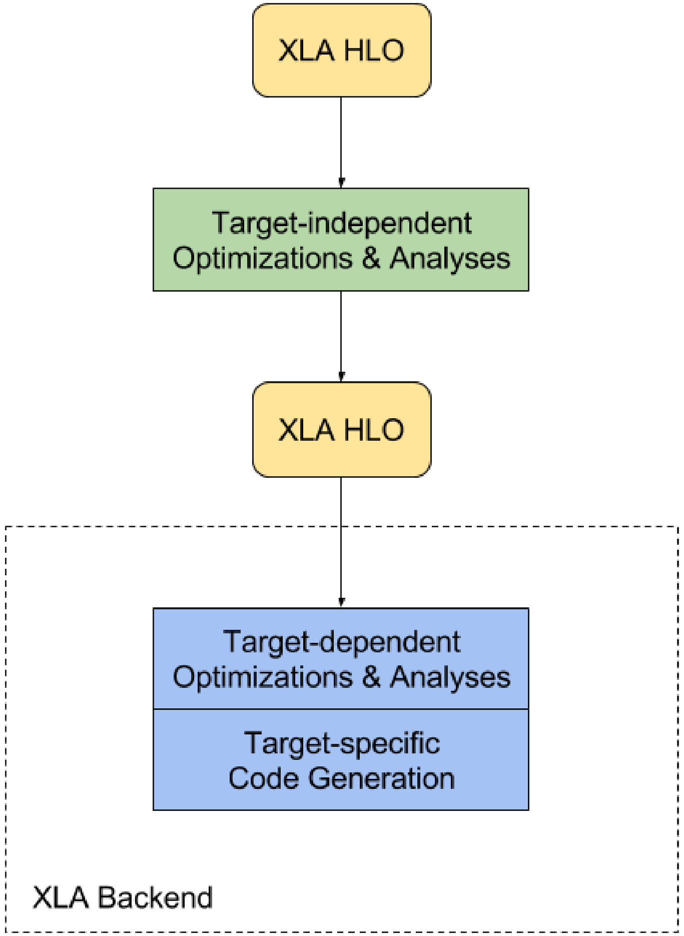
DNN Compilers

- Use techniques from compiler construction: DNN → Graph → IR → Transformations → HW Mapping

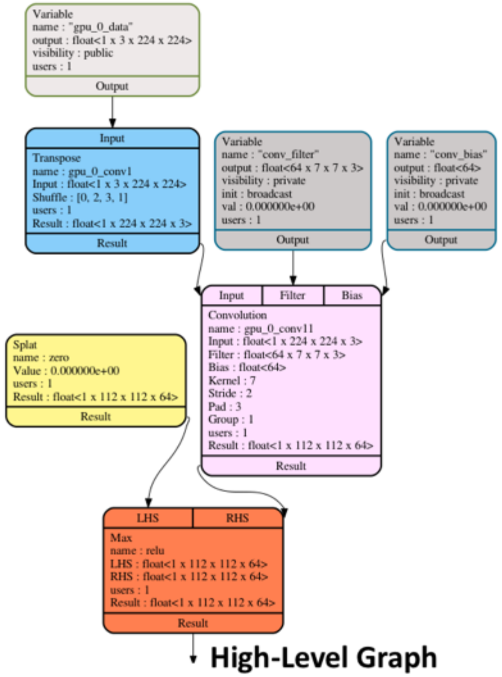


DNN Compilers

- Use techniques from compiler construction: DNN → Graph → IR → Transformations → HW Mapping



TensorFlow XLA



High-Level Graph

```
declare {
 %input = weight float<8 x 28 x 28 x 1>, broadcast, 0.0
 %filter = weight float<16 x 5 x 5 x 1>, xavier, 25.0
 %filter0 = weight float<16>, broadcast, 0.100
 %weights = weight float<10 x 144>, xavier, 144.0
 %bias = weight float<10>, broadcast, 0.100
 %selected = weight index<8 x 1>
 ...
 %result = weight float<8 x 10>
}

program {
 %allo = alloc float<8 x 28 x 28 x 16>
 %conv = convolution [5 1 2 16] @out %allo, @in %input,
     @in %filter3, @in %bias0
 %allo0 = alloc float<8 x 28 x 28 x 16>
 %relu = max0 @out %allo0, @in %allo
 %allo1 = alloc index<8 x 9 x 9 x 16 x 2>
 %allo2 = alloc float<8 x 9 x 9 x 16>
 %pool = pool max [3 3 0] @out %allo2, @in %allo0,
     @inout %allo1
 ...
 %deal6 = dealloc @out %allo6
 %deal7 = dealloc @out %allo7
 %deal8 = dealloc @out %allo8
 %deal9 = dealloc @out %allo9
}
```

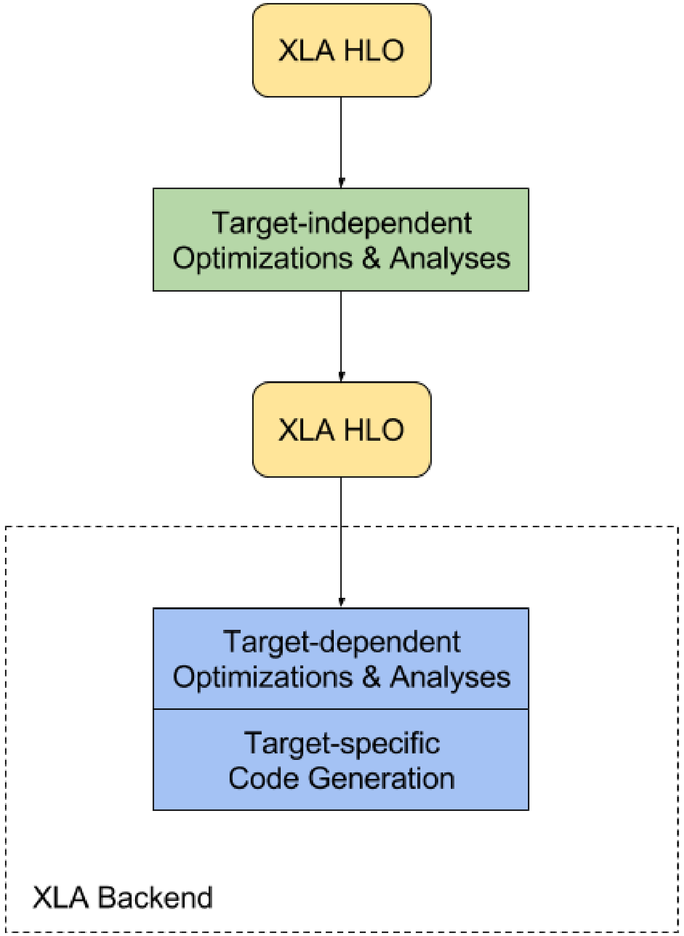
Low-Level IR

```
LBB14_1:
  vmovaps 3211264(%rcx,%rax,4), %ymm1
  vmovaps 3211296(%rcx,%rax,4), %ymm2
  vmovaps 3211328(%rcx,%rax,4), %ymm3
  vaddps 6422528(%rcx,%rax,4), %ymm1, %ymm1
  vaddps 6422560(%rcx,%rax,4), %ymm2, %ymm2
  vmovaps 3211360(%rcx,%rax,4), %ymm4
  vaddps 6422592(%rcx,%rax,4), %ymm3, %ymm3
  vaddps 6422624(%rcx,%rax,4), %ymm4, %ymm4
  vmaxps %ymm0, %ymm1, %ymm1
  vmaxps %ymm0, %ymm2, %ymm2
  vmaxps %ymm0, %ymm3, %ymm3
  vmovaps %ymm1, 6422528(%rcx,%rax,4)
  vmovaps %ymm2, 6422560(%rcx,%rax,4)
  vmaxps %ymm0, %ymm4, %ymm1
  vmovaps %ymm3, 6422592(%rcx,%rax,4)
  vmovaps %ymm1, 6422624(%rcx,%rax,4)
  addq $32, %rax
```

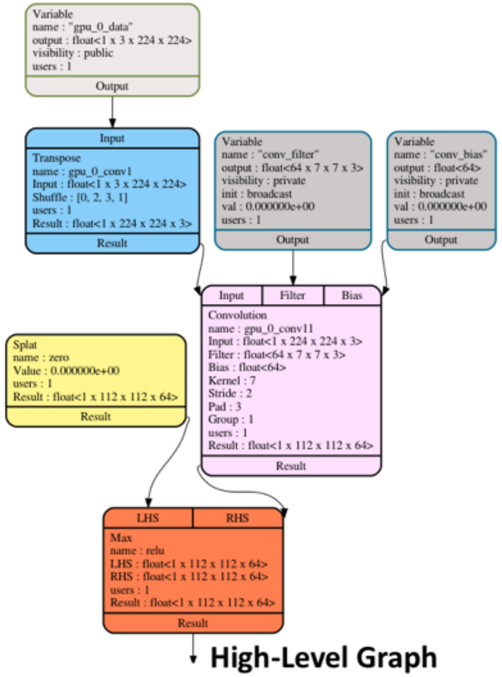
Machine Code

DNN Compilers

- Use techniques from compiler construction: DNN → Graph → IR → Transformations → HW Mapping



TensorFlow XLA



High-Level Graph

```
declare {
 %input = weight float<8 x 28 x 28 x 1>, broadcast, 0.0
 %filter = weight float<16 x 5 x 5 x 1>, xavier, 25.0
 %filter0 = weight float<16>, broadcast, 0.100
 %weights = weight float<10 x 144>, xavier, 144.0
 %bias = weight float<10>, broadcast, 0.100
 %selected = weight index<8 x 1>
 ...
 %result = weight float<8 x 10>
}

program {
 %allo = alloc float<8 x 28 x 28 x 16>
 %conv = convolution [5 1 2 16] @out %allo, @in %input,
     @in %filter3, @in %bias0
 %allo0 = alloc float<8 x 28 x 28 x 16>
 %relu = max0 @out %allo0, @in %allo
 %allo1 = alloc index<8 x 9 x 9 x 16 x 2>
 %allo2 = alloc float<8 x 9 x 9 x 16>
 %pool = pool max [3 3 0] @out %allo2, @in %allo0,
     @inout %allo1
 ...
 %deal6 = dealloc @out %allo6
 %deal7 = dealloc @out %allo7
 %deal8 = dealloc @out %allo8
 %deal9 = dealloc @out %allo9
}
```

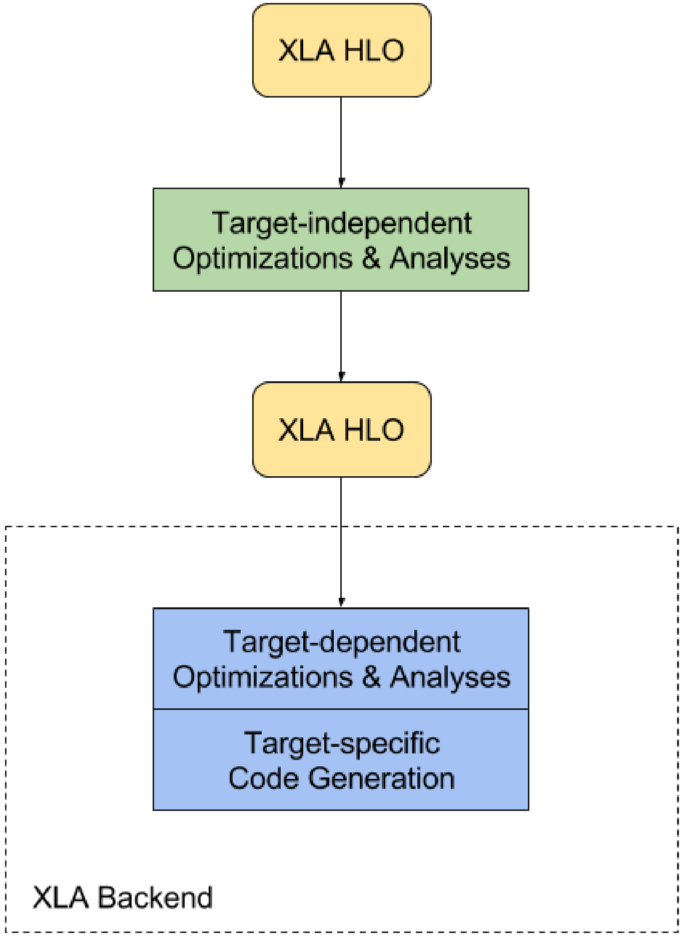
Low-Level IR

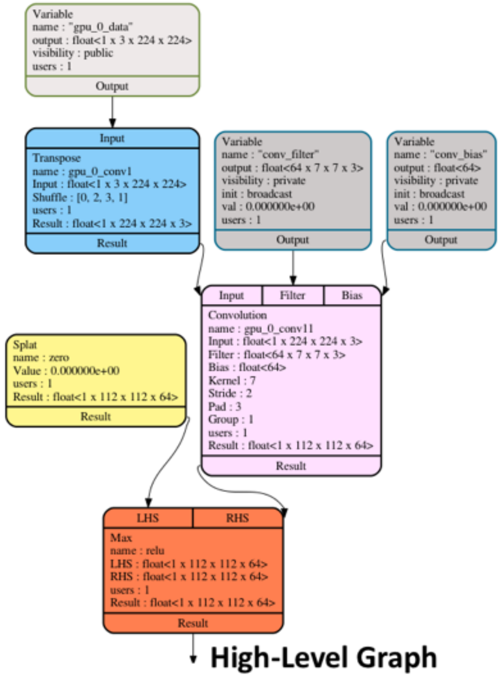
```
LBB14_1:
  vmovaps 3211264(%rcx,%rax,4), %ymm1
  vmovaps 3211296(%rcx,%rax,4), %ymm2
  vmovaps 3211328(%rcx,%rax,4), %ymm3
  vaddps 6422528(%rcx,%rax,4), %ymm1, %ymm1
  vaddps 6422560(%rcx,%rax,4), %ymm2, %ymm2
  vmovaps 3211360(%rcx,%rax,4), %ymm4
  vaddps 6422592(%rcx,%rax,4), %ymm3, %ymm3
  vaddps 6422624(%rcx,%rax,4), %ymm4, %ymm4
  vmaxps %ymm0, %ymm1, %ymm1
  vmaxps %ymm0, %ymm2, %ymm2
  vmaxps %ymm0, %ymm3, %ymm3
  vmovaps %ymm1, 6422528(%rcx,%rax,4)
  vmovaps %ymm2, 6422560(%rcx,%rax,4)
  vmaxps %ymm0, %ymm4, %ymm1
  vmovaps %ymm3, 6422592(%rcx,%rax,4)
  vmovaps %ymm1, 6422624(%rcx,%rax,4)
  addq $32, %rax
```

Machine Code

DNN Compilers

- Use techniques from compiler construction: DNN → Graph → IR → Transformations → HW Mapping





High-Level Graph

→

```
declare {
 %input = weight float<8 x 28 x 28 x 1>, broadcast, 0.0
 %filter = weight float<16 x 5 x 5 x 1>, xavier, 25.0
 %filter0 = weight float<16>, broadcast, 0.100
 %weights = weight float<10 x 144>, xavier, 144.0
 %bias = weight float<10>, broadcast, 0.100
 %selected = weight index<8 x 1>
 ...
 %result = weight float<8 x 10>
}

program {
 %allo = alloc float<8 x 28 x 28 x 16>
 %conv = convolution [5 1 2 16] @out %allo, @in %input,
     @in %filter3, @in %bias0
 %allo0 = alloc float<8 x 28 x 28 x 16>
 %relu = max0 @out %allo0, @in %allo
 %allo1 = alloc index<8 x 9 x 9 x 16 x 2>
 %allo2 = alloc float<8 x 9 x 9 x 16>
 %pool = pool max [3 3 0] @out %allo2, @in %allo0,
     @inout %allo1
 ...
 %deal6 = dealloc @out %allo6
 %deal7 = dealloc @out %allo7
 %deal8 = dealloc @out %allo8
 %deal9 = dealloc @out %allo9
}
```

Low-Level IR

→

```
LBB14_1:
  vmovaps 3211264(%rcx,%rax,4), %ymm1
  vmovaps 3211296(%rcx,%rax,4), %ymm2
  vmovaps 3211328(%rcx,%rax,4), %ymm3
  vaddps 6422528(%rcx,%rax,4), %ymm1, %ymm1
  vaddps 6422560(%rcx,%rax,4), %ymm2, %ymm2
  vmovaps 3211360(%rcx,%rax,4), %ymm4
  vaddps 6422592(%rcx,%rax,4), %ymm3, %ymm3
  vaddps 6422624(%rcx,%rax,4), %ymm4, %ymm4
  vmaxps %ymm0, %ymm1, %ymm1
  vmaxps %ymm0, %ymm2, %ymm2
  vmaxps %ymm0, %ymm3, %ymm3
  vmovaps %ymm1, 6422528(%rcx,%rax,4)
  vmovaps %ymm2, 6422560(%rcx,%rax,4)
  vmaxps %ymm0, %ymm4, %ymm1
  vmovaps %ymm3, 6422592(%rcx,%rax,4)
  vmovaps %ymm1, 6422624(%rcx,%rax,4)
  addq $32, %rax
```

Machine Code

TensorFlow XLA **Facebook Glow**

DNN Compilers

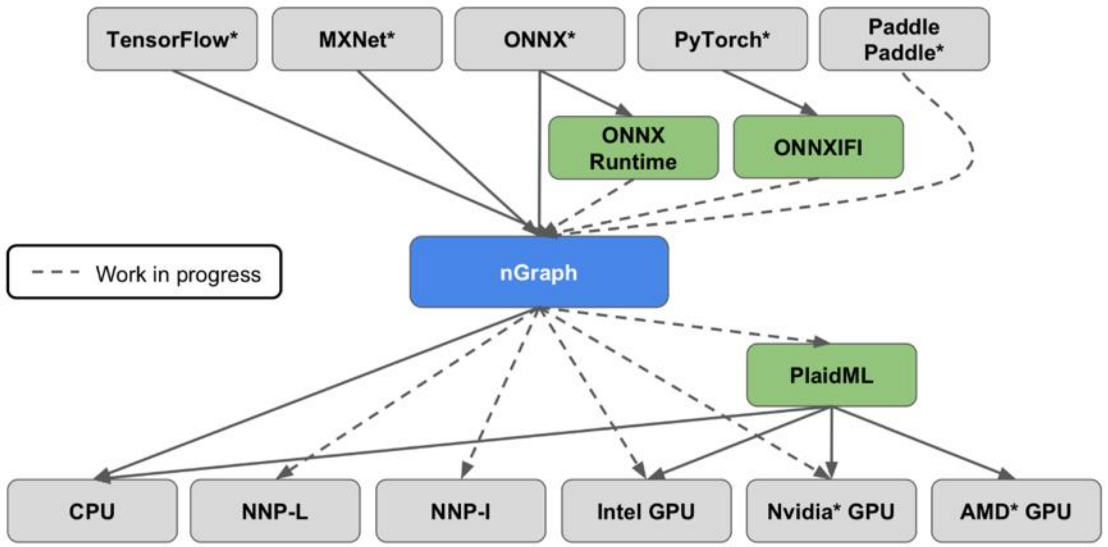
- **Use techniques from compiler construction: DNN → Graph → IR → Transformations → HW Mapping**

TensorFlow XLA

Facebook Glow

DNN Compilers

- Use techniques from compiler construction: DNN → Graph → IR → Transformations → HW Mapping



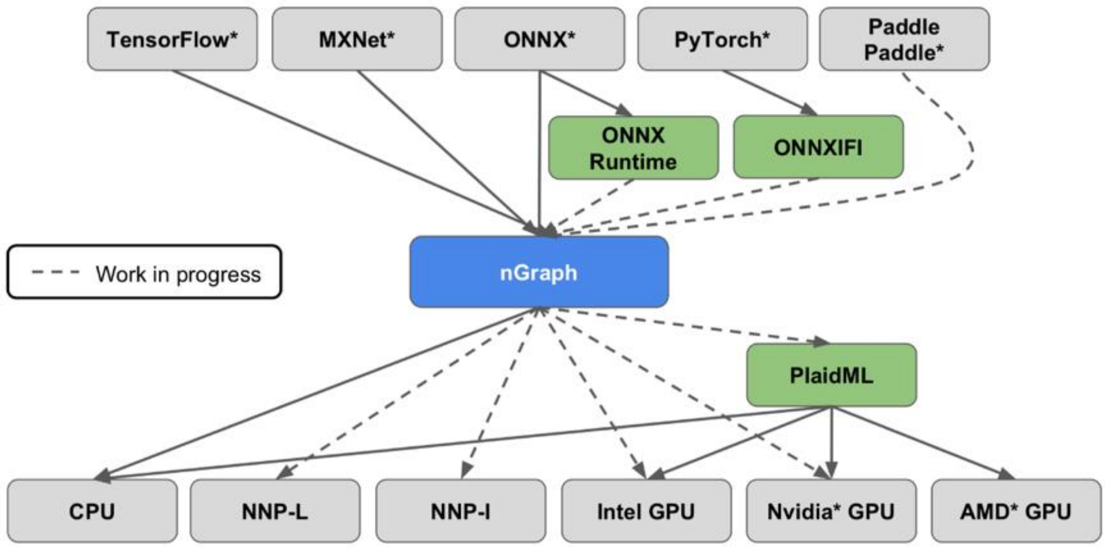
Intel nGraph Compiler stack contains trademarks of Intel Corporation or its subsidiaries in the U.S. and/or other countries.
*Other names and brands may be claimed as the property of others; see the branding notice and documentation for further details: https://ngraph.nervanasys.com/docs/latest

Intel nGraph

TVM Stack

DNN Compilers

- **Use techniques from compiler construction: DNN → Graph → IR → Transformations → HW Mapping**

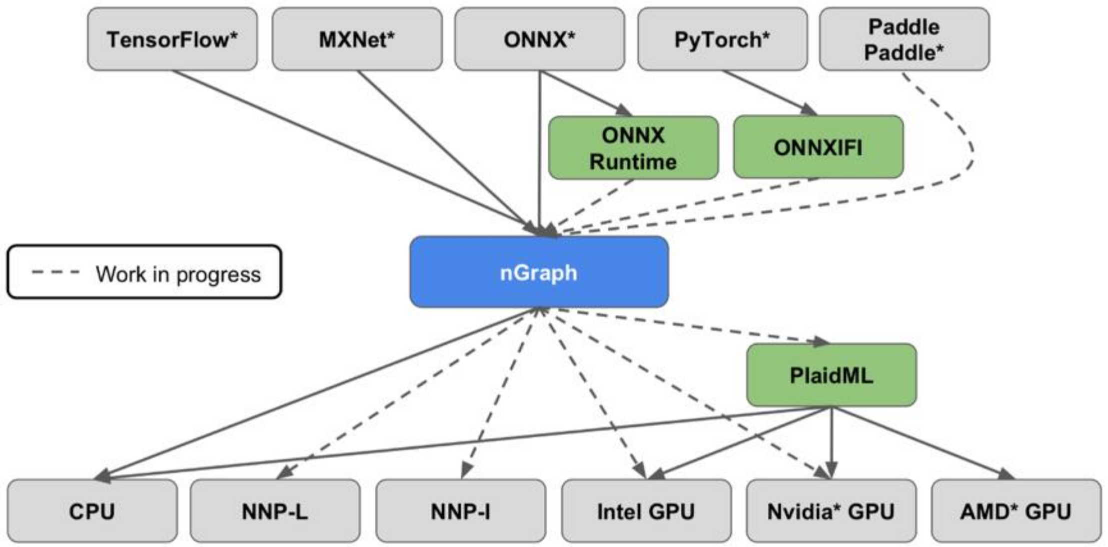


Intel nGraph Compiler stack contains trademarks of Intel Corporation or its subsidiaries in the U.S. and/or other countries.
*Other names and brands may be claimed as the property of others; see the branding notice and documentation for further details: https://ngraph.nervanasys.com/docs/latest

Intel nGraph **TVM Stack**

DNN Compilers

- Use techniques from compiler construction: DNN → Graph → IR → Transformations → HW Mapping



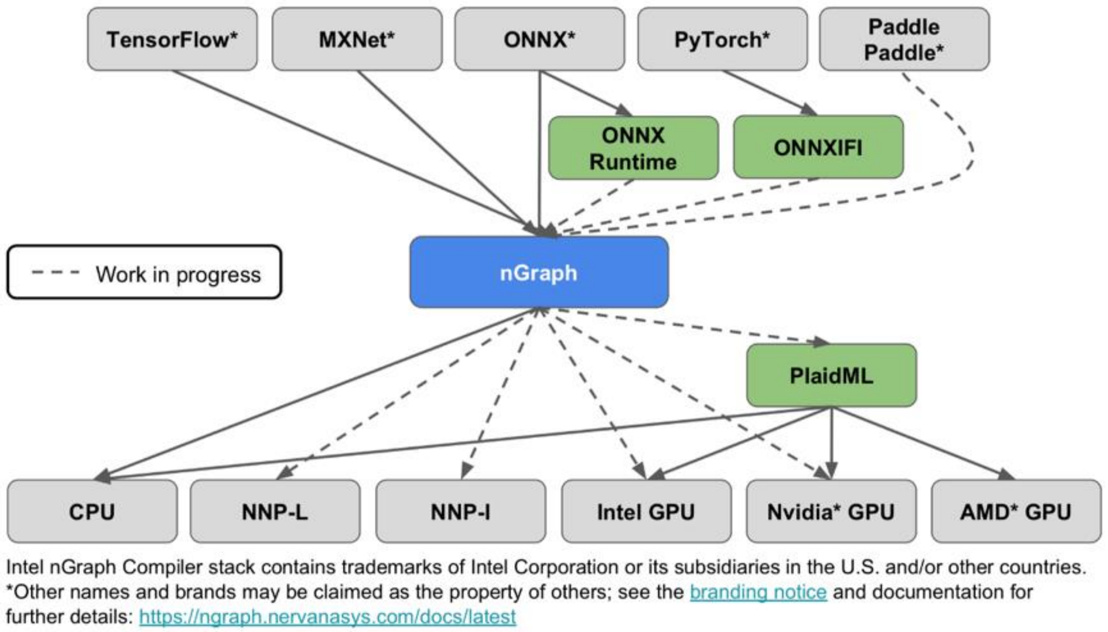
Intel nGraph Compiler stack contains trademarks of Intel Corporation or its subsidiaries in the U.S. and/or other countries. *Other names and brands may be claimed as the property of others; see the branding notice and documentation for further details: https://ngraph.nervanasys.com/docs/latest

Intel nGraph

TVM Stack

DNN Compilers

- Use techniques from compiler construction: DNN → Graph → IR → Transformations → HW Mapping



Intel nGraph

TVM Stack

How to <u>not</u> do this

How to **not** do this

"Twelve ways to fool the masses when reporting performance of deep learning workloads"

(A humorous guide to floptimize deep learning)

https://htor.inf.ethz.ch/blog/index.php/2018/11/08/twelve-ways-to-fool-the-masses-when-reporting-performance-of-deep-learning-workloads/

How to not do this

"Twelve ways to fool the masses when reporting performance of deep learning workloads"

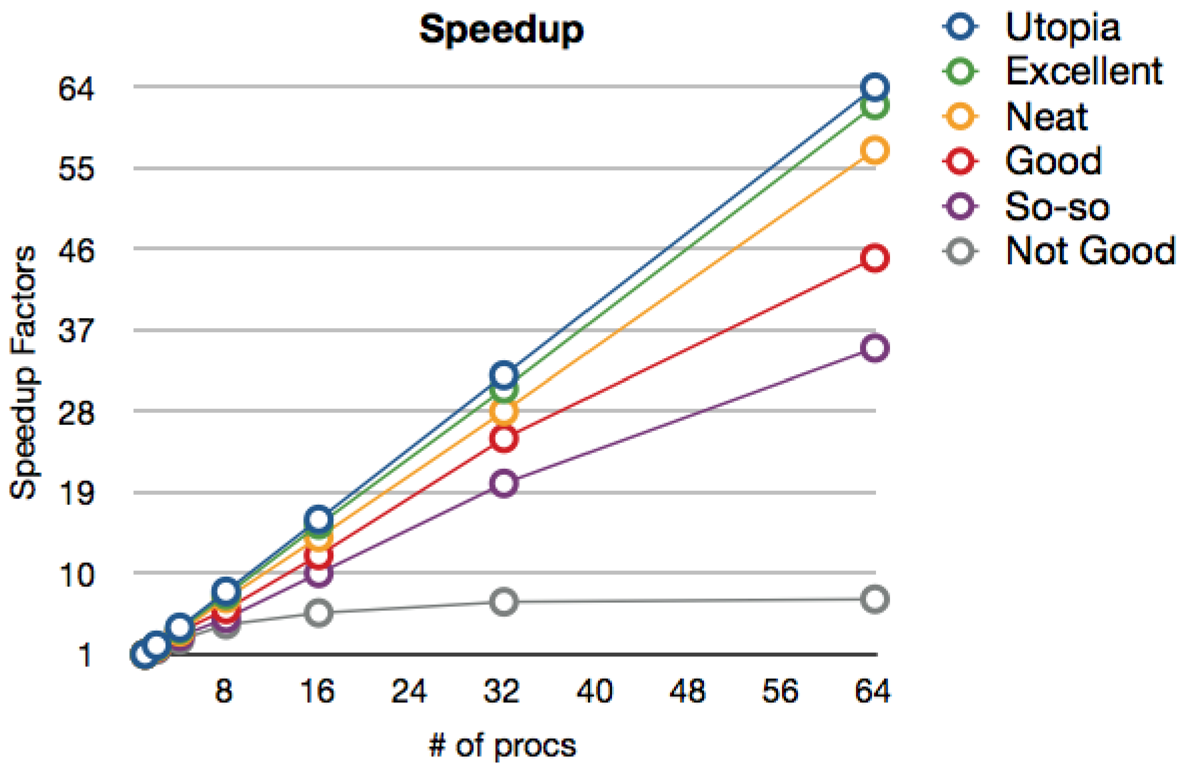
(A humorous guide to floptimize deep learning)

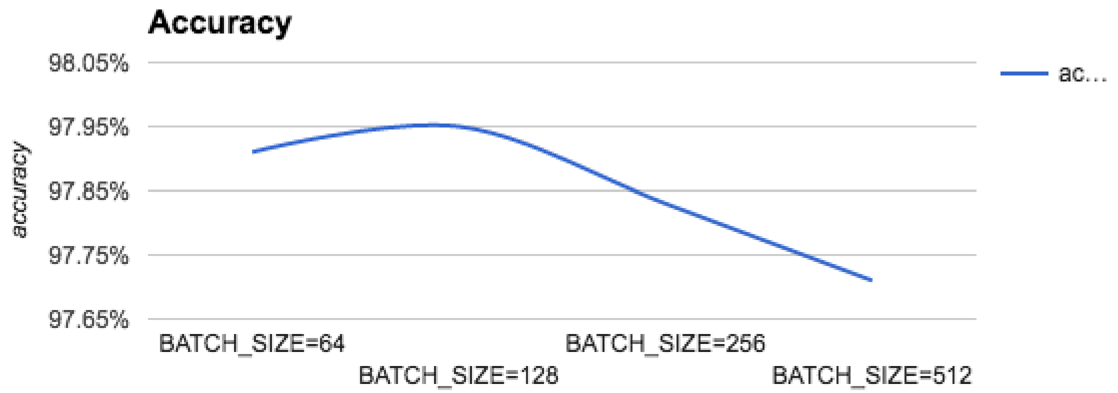
https://htor.inf.ethz.ch/blog/index.php/2018/11/08/twelve-ways-to-fool-the-masses-when-reporting-performance-of-deep-learning-workloads/

8) Show performance when enabling option set A and show accuracy when enabling option set B!

- **Pretty cool idea isn't it? Hyperparameters sometimes conflict**

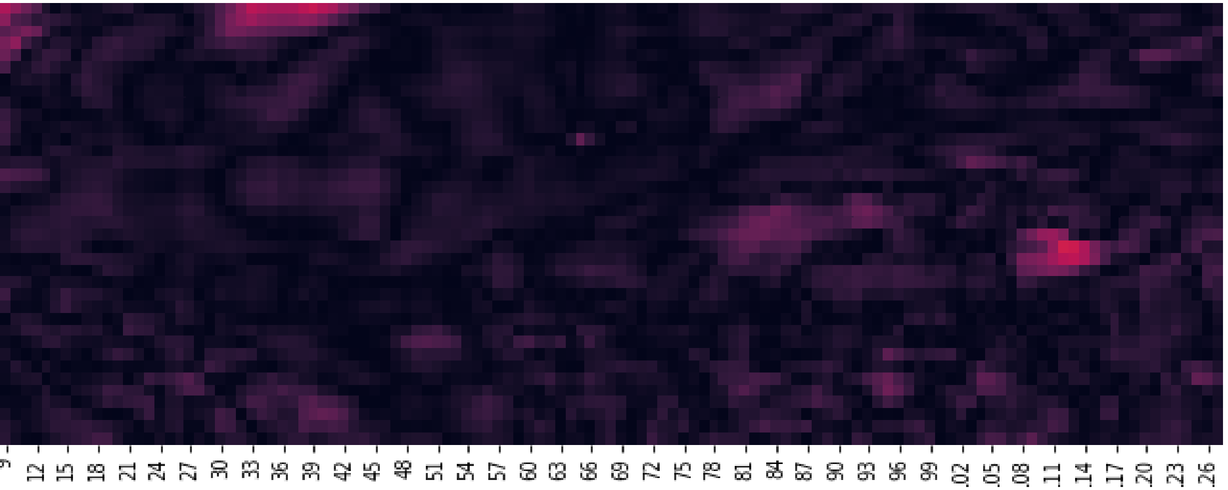
  So always tune the to show the best result, whatever the result shall be!





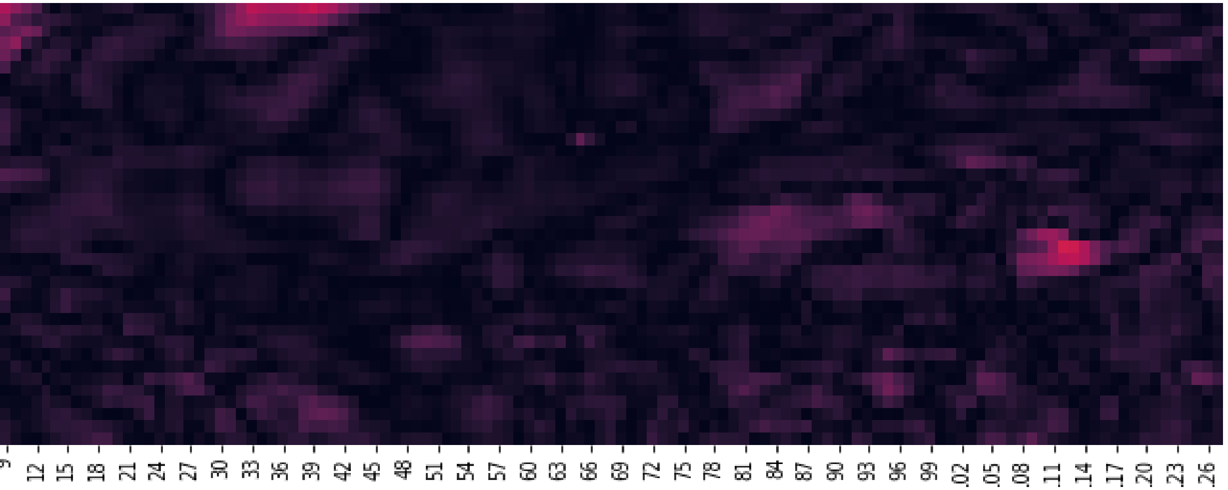
Some big deep learning applications

Some big deep learning applications



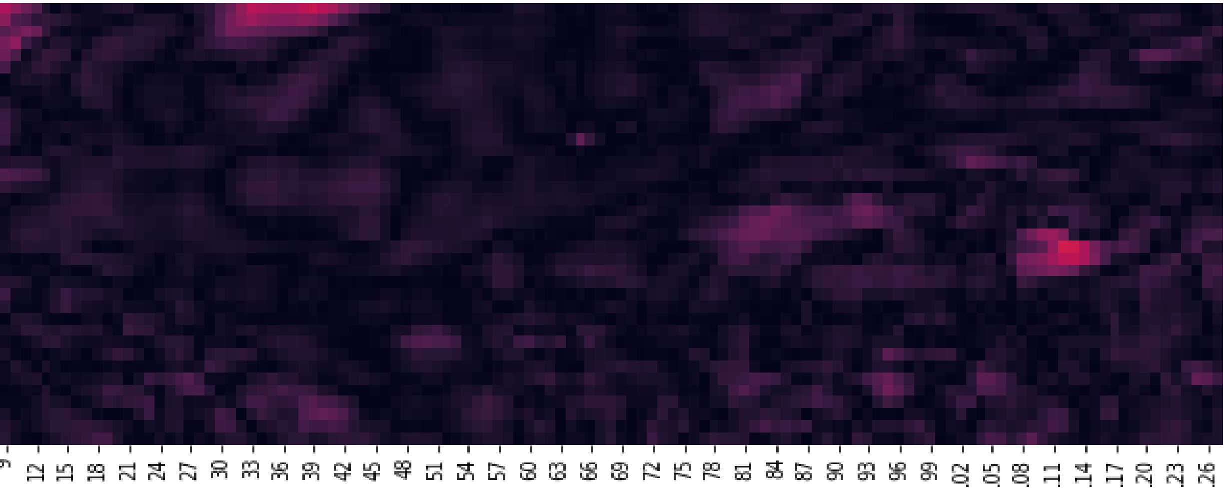
Some big deep learning applications

UQ for weather prediction [Grönquist et al. 2019]

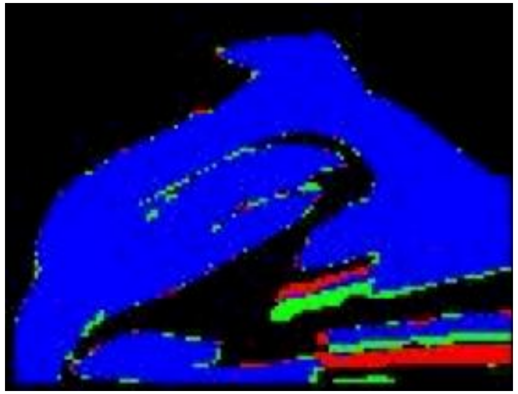


Some big deep learning applications

UQ for weather prediction [Grönquist et al. 2019]

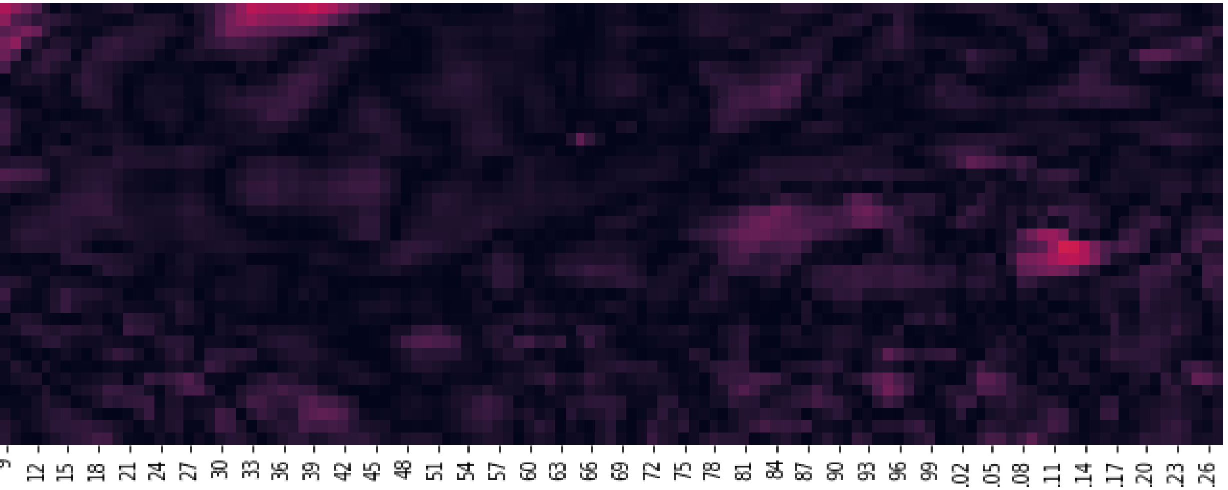


Predicting mesh tangling [Dryden et al. 2019]

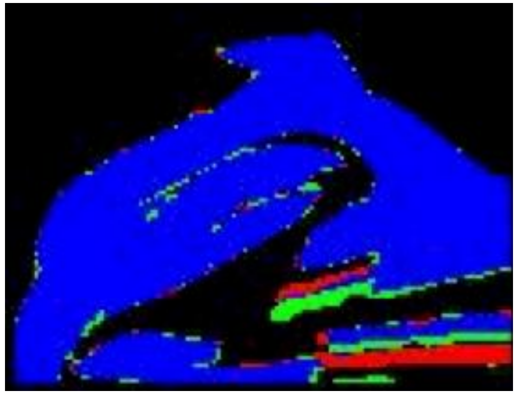


Some big deep learning applications

UQ for weather prediction [Grönquist et al. 2019]

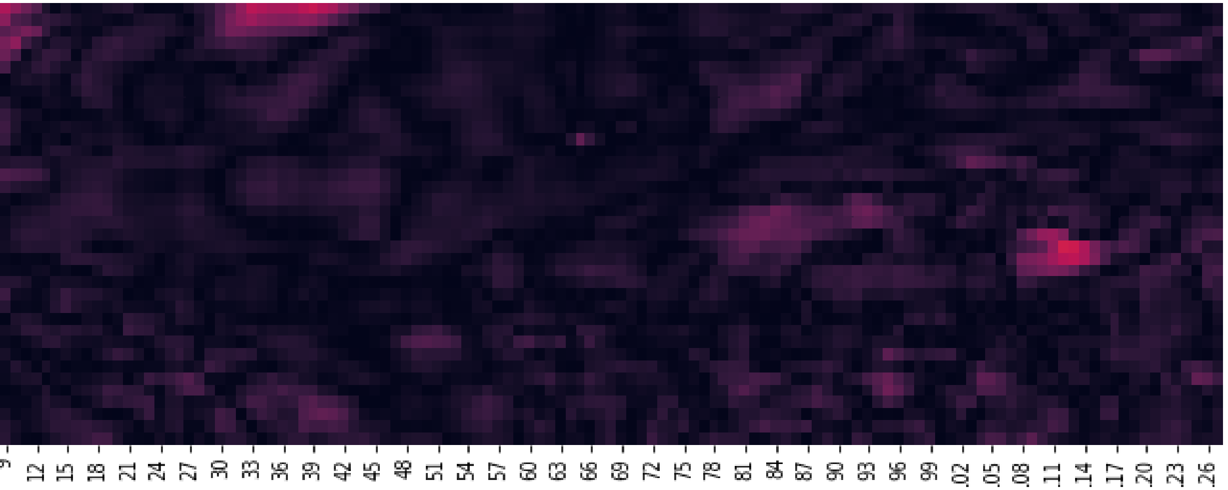


Predicting mesh tangling [Dryden et al. 2019]

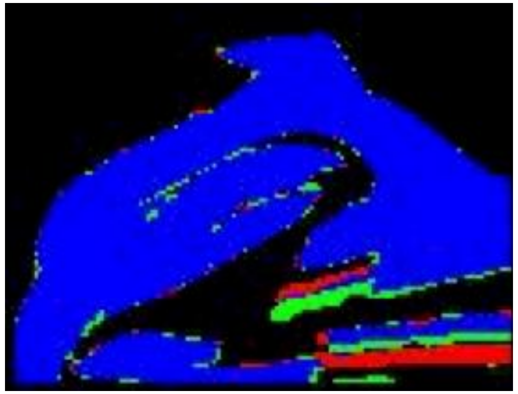


Some big deep learning applications

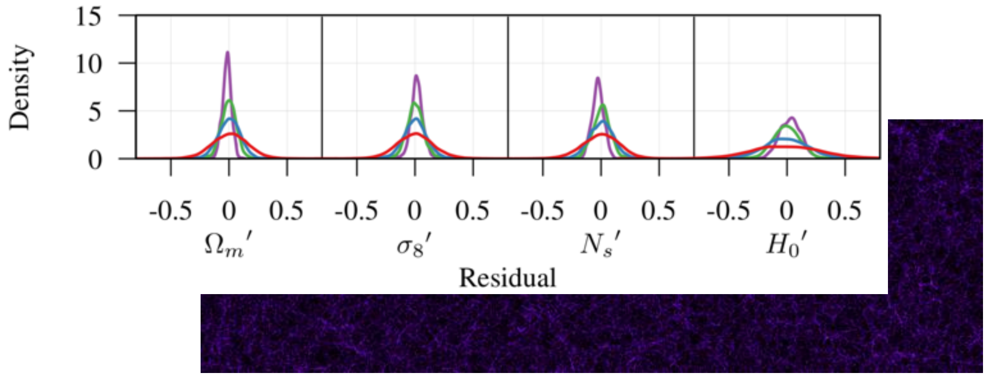
UQ for weather prediction [Grönquist et al. 2019]



Predicting mesh tangling [Dryden et al. 2019]

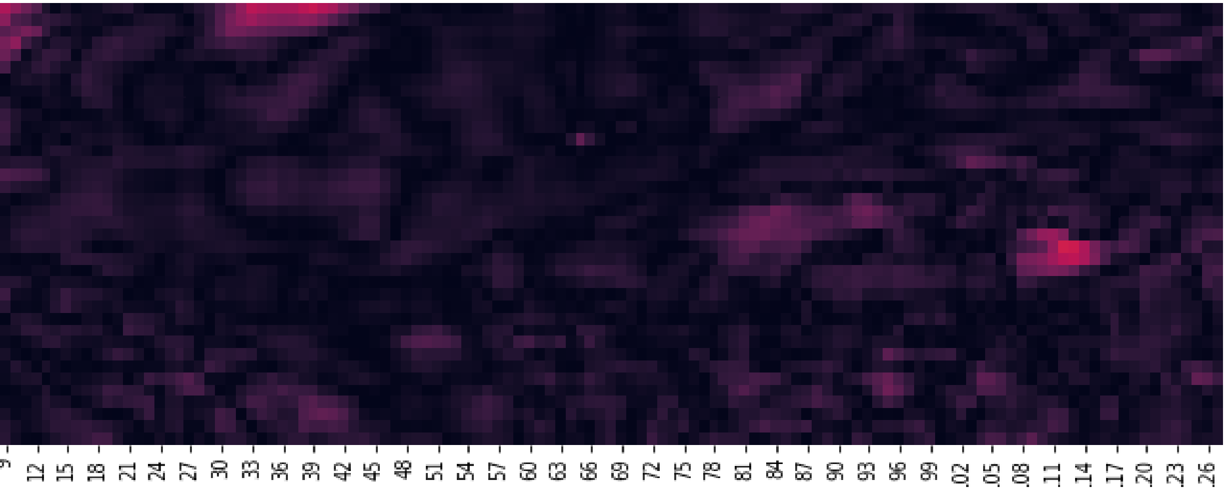


Cosmology [Oyama et al. 2019]

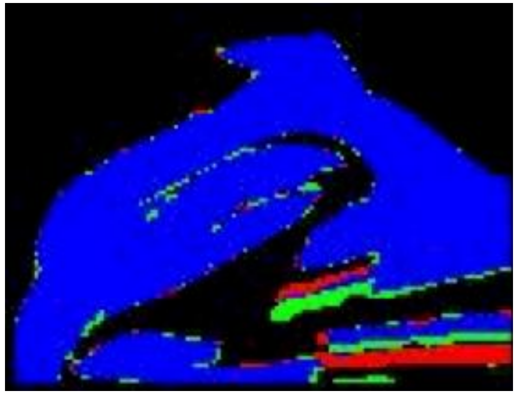


Some big deep learning applications

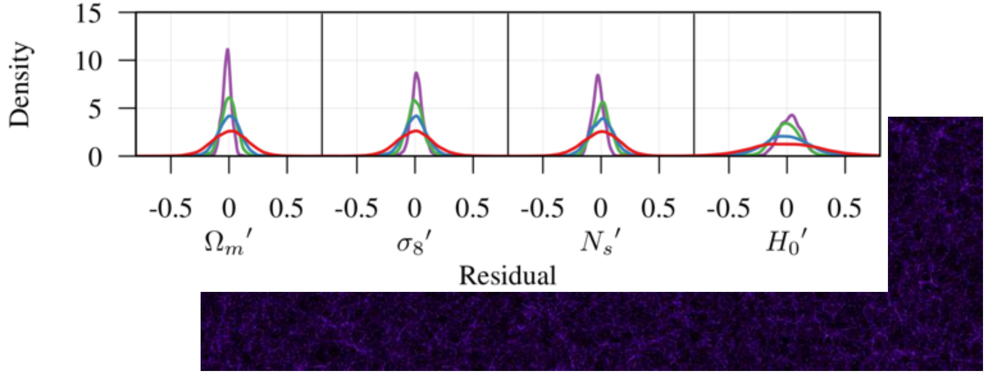
UQ for weather prediction [Grönquist et al. 2019]



Predicting mesh tangling [Dryden et al. 2019]

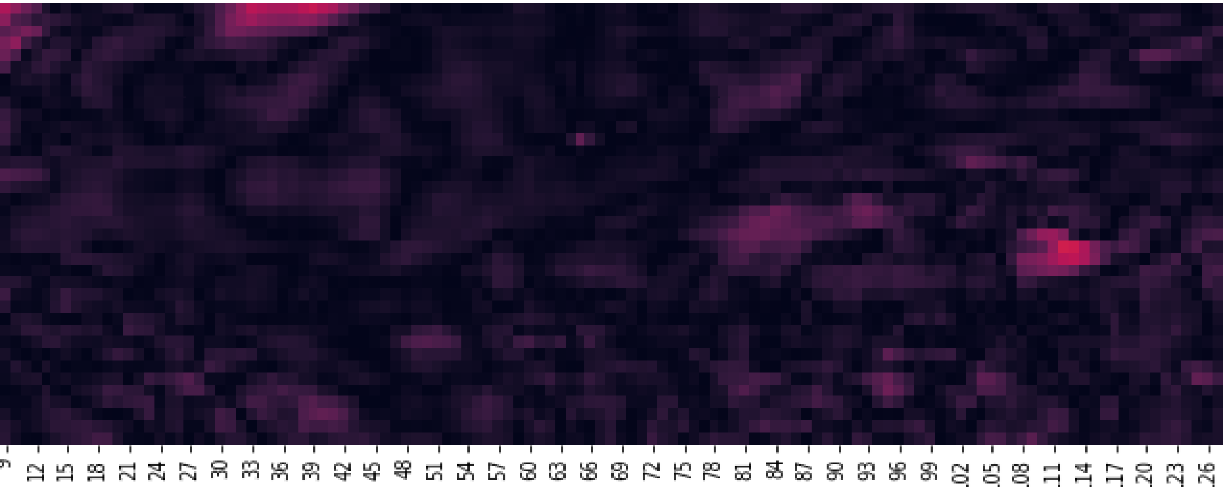


Cosmology [Oyama et al. 2019]

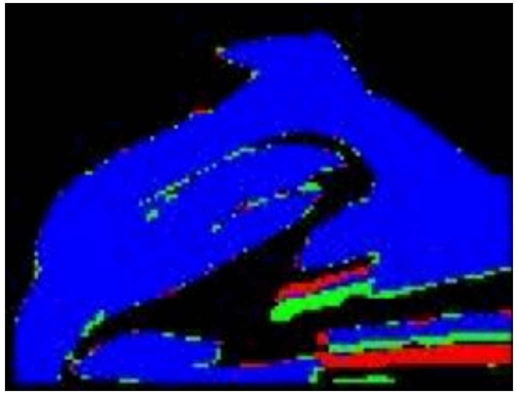


Some big deep learning applications

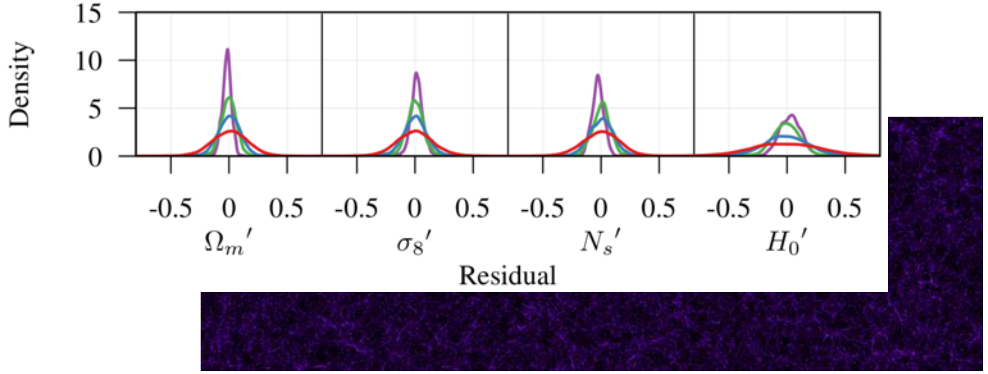
UQ for weather prediction [Grönquist et al. 2019]



Predicting mesh tangling [Dryden et al. 2019]

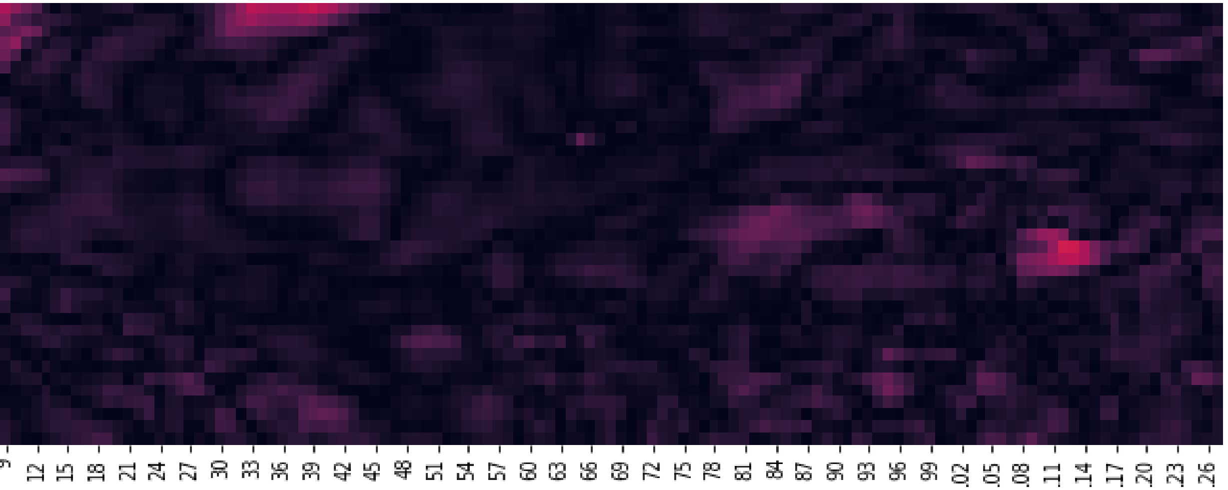


Cosmology [Oyama et al. 2019]

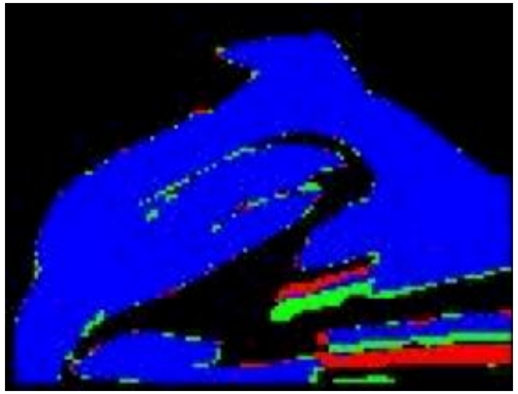


Some big deep learning applications

UQ for weather prediction [Grönquist et al. 2019]

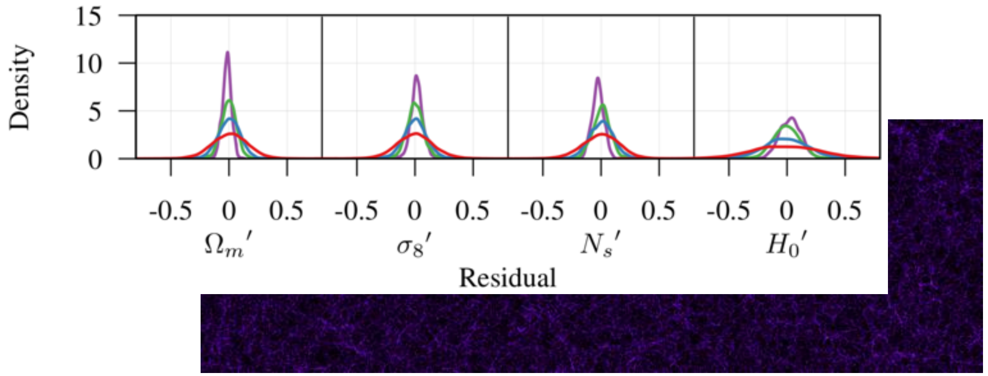


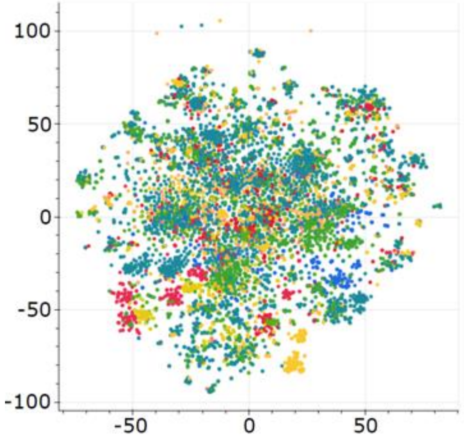
Predicting mesh tangling [Dryden et al. 2019]



Code comprehension [Ben-Nun et al. 2018]

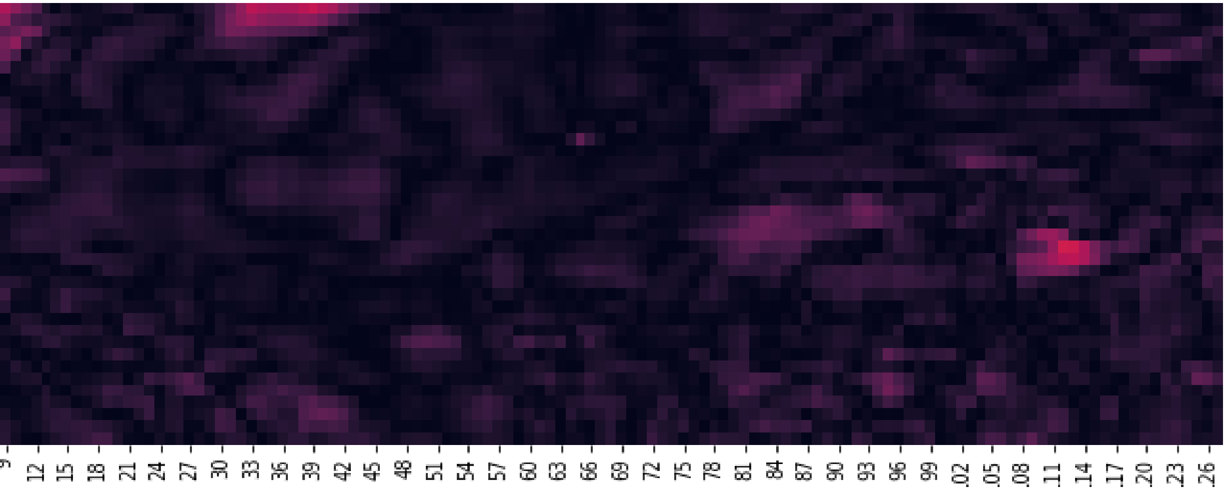
Cosmology [Oyama et al. 2019]



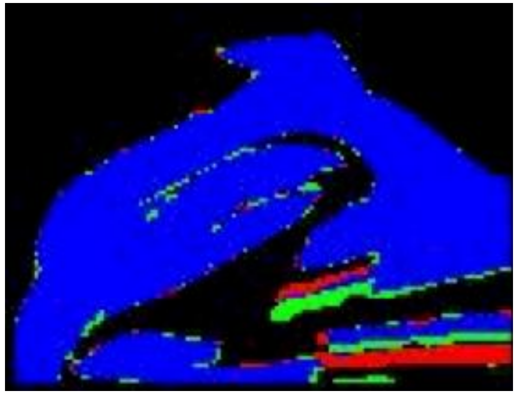


Some big deep learning applications

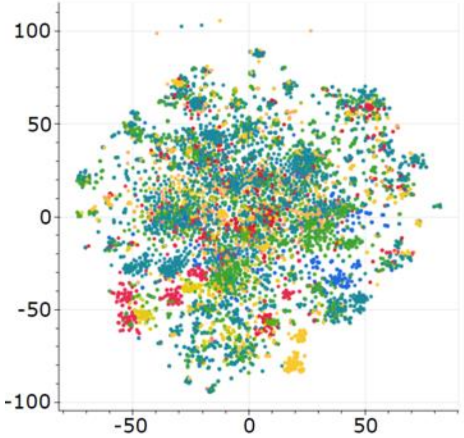
UQ for weather prediction [Grönquist et al. 2019]



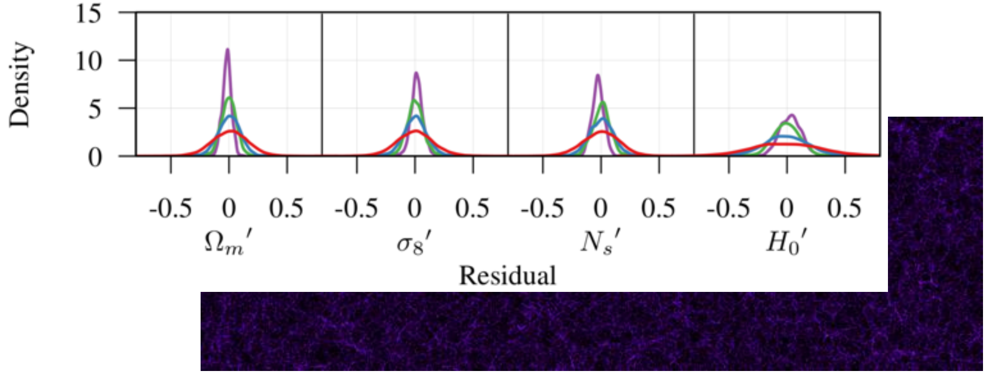
Predicting mesh tangling [Dryden et al. 2019]



Code comprehension [Ben-Nun et al. 2018]

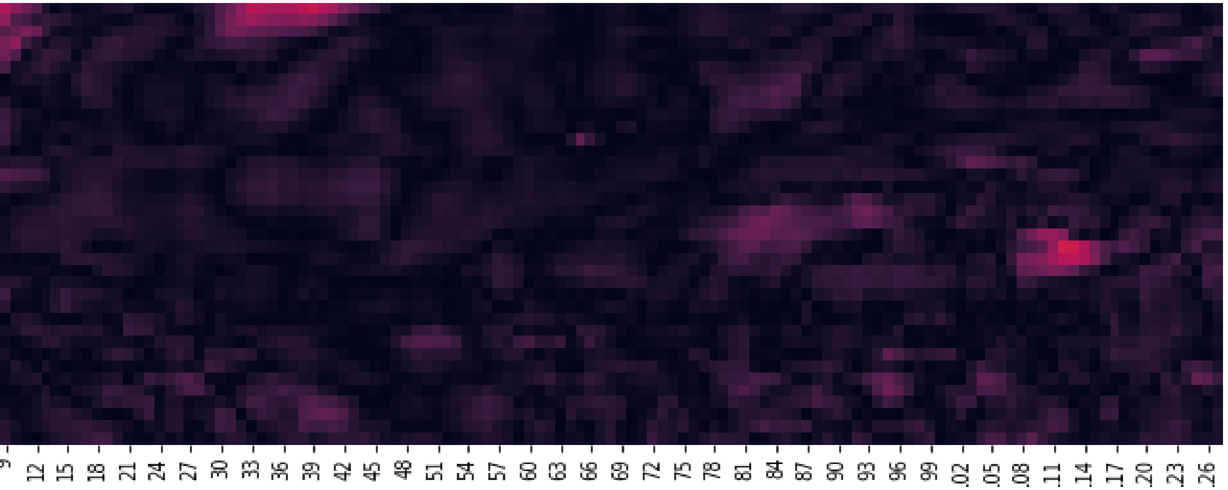


Cosmology [Oyama et al. 2019]

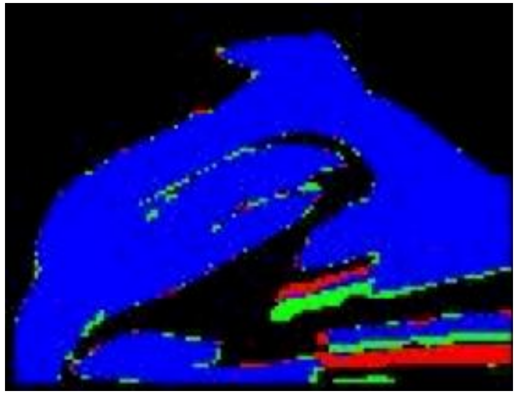


Some big deep learning applications

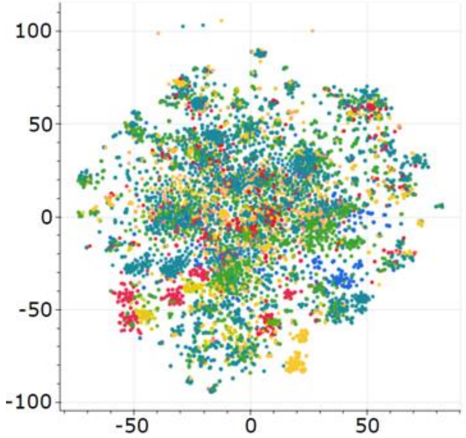
UQ for weather prediction [Grönquist et al. 2019]



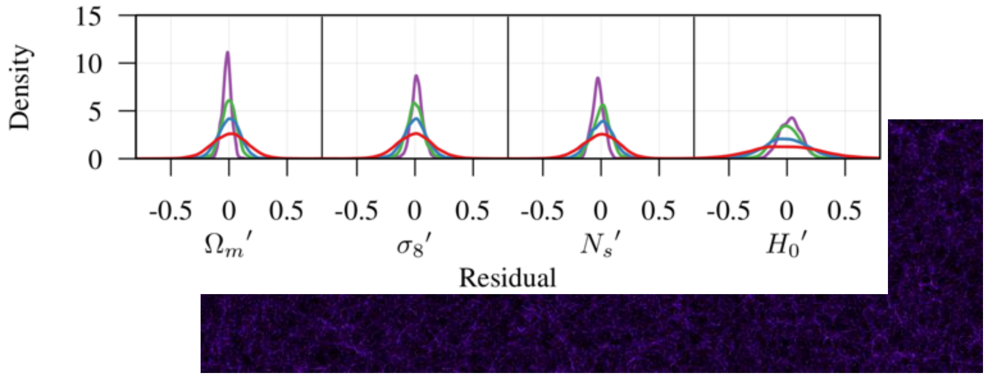
Predicting mesh tangling [Dryden et al. 2019]



Code comprehension [Ben-Nun et al. 2018]

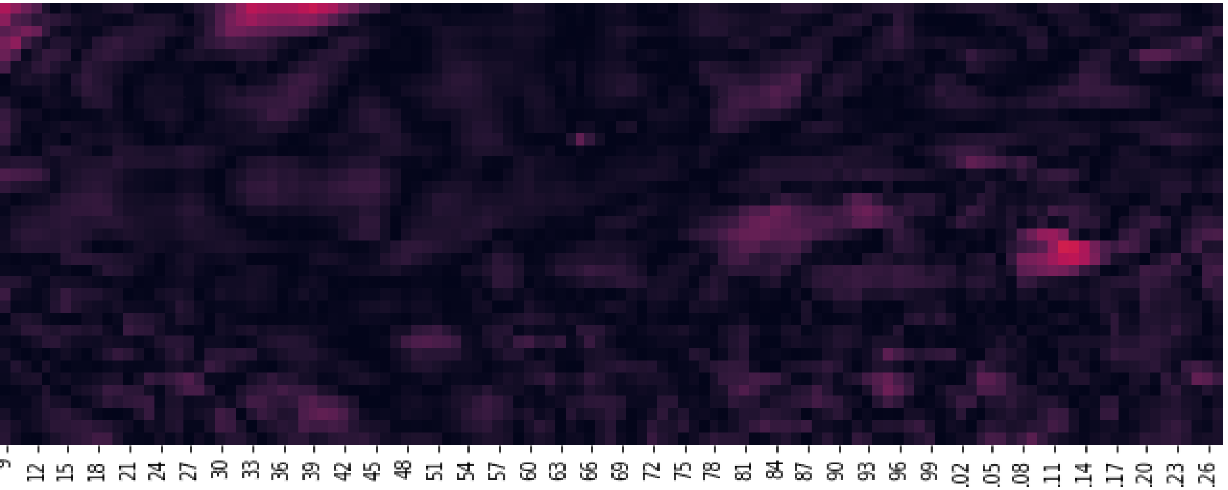


Cosmology [Oyama et al. 2019]

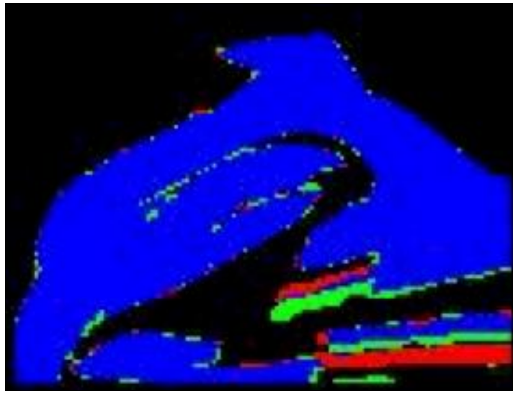


Some big deep learning applications

UQ for weather prediction [Grönquist et al. 2019]

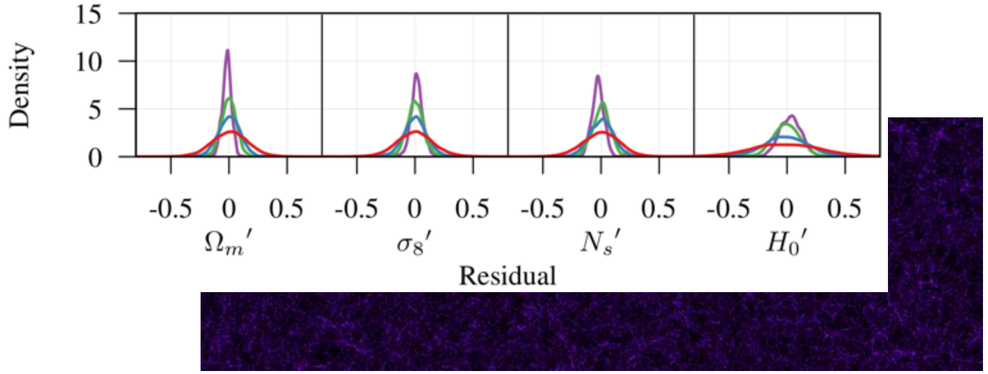


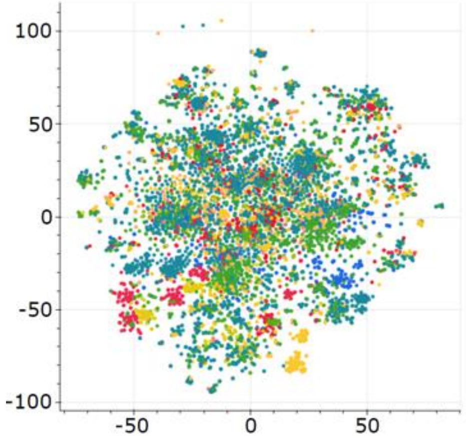
Predicting mesh tangling [Dryden et al. 2019]



Code comprehension [Ben-Nun et al. 2018]

Cosmology [Oyama et al. 2019]

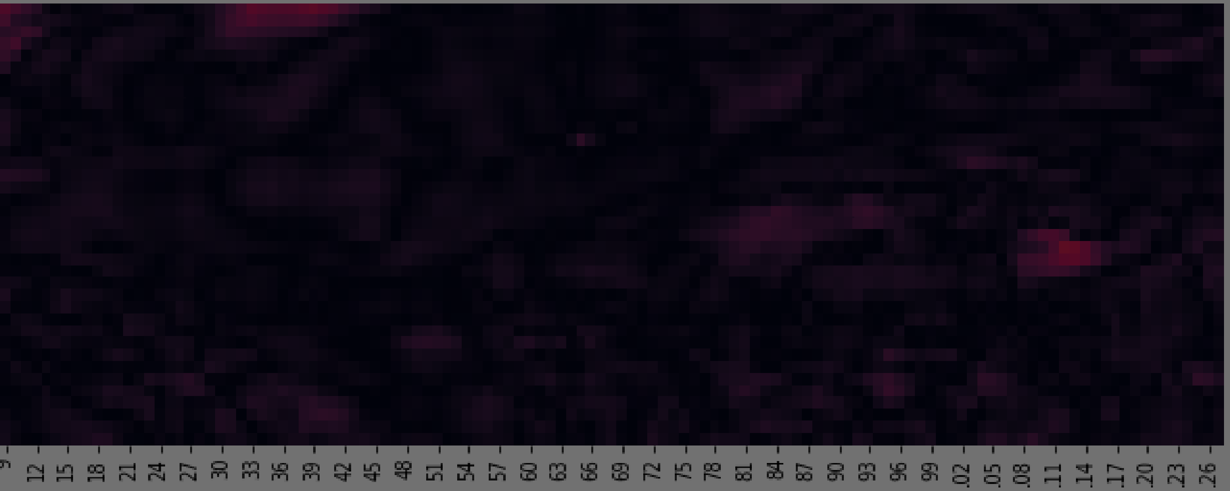




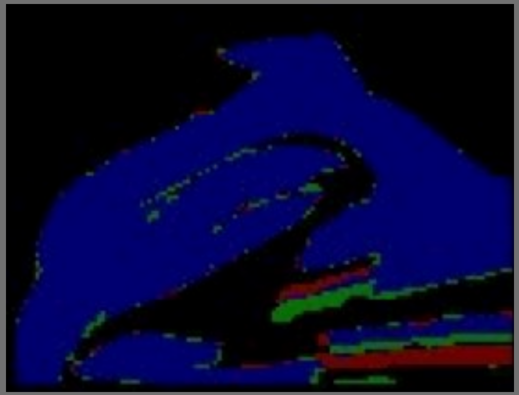
Big seq2seq models

Some big deep learning applications

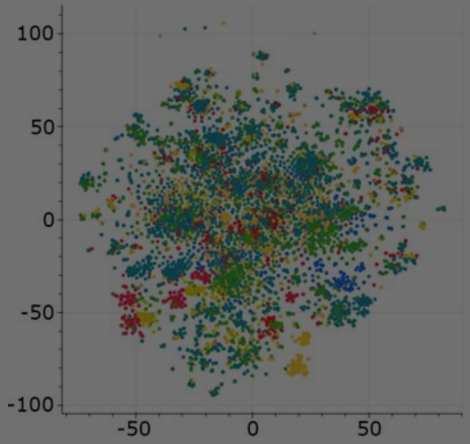
UQ for weather prediction [Grönquist et al. 2019]



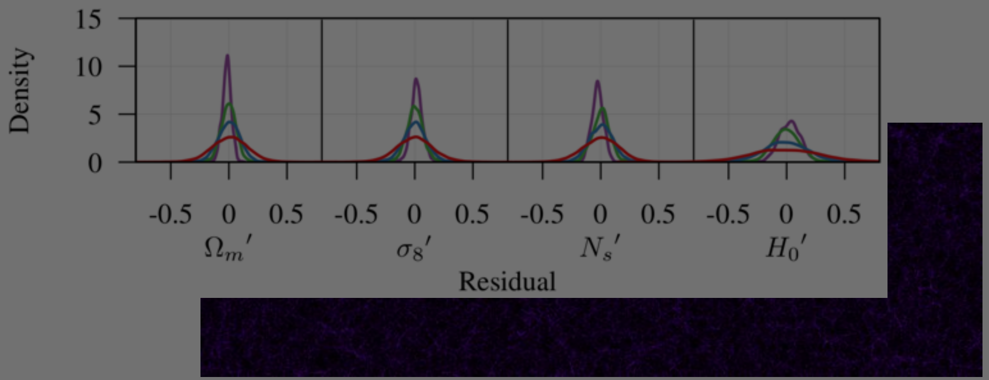
Predicting mesh tangling [Dryden et al. 2019]



Code comprehension [Ben-Nun et al. 2018]



Cosmology [Oyama et al. 2019]

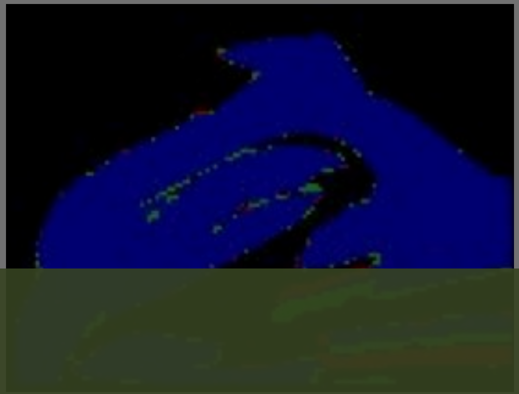


Big seq2seq models

Some big deep learning applications

UQ for weather prediction [Grönquist et al. 2019]

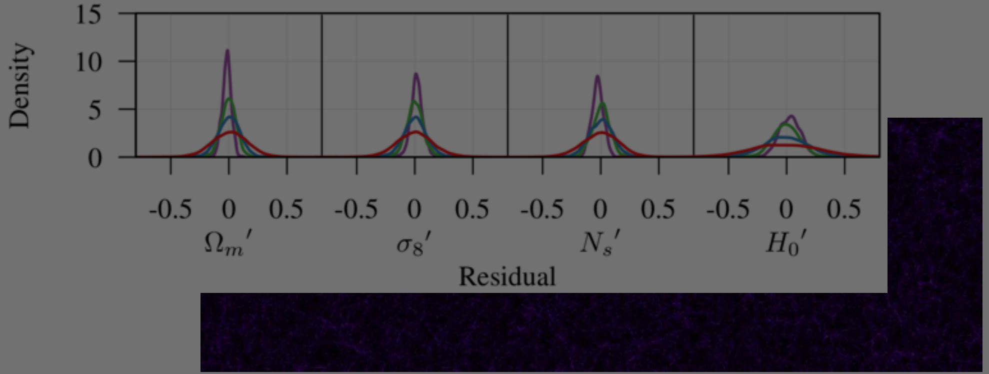
Predicting mesh tangling [Dryden et al. 2019]

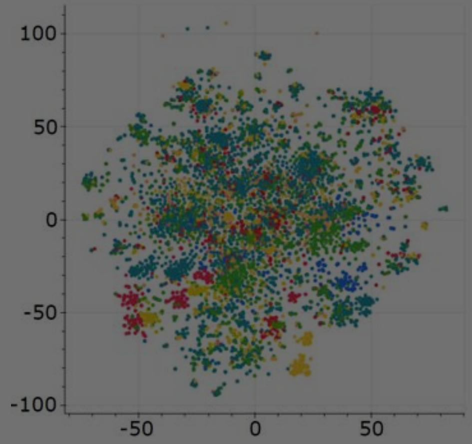


And many more!

Code comprehension [Ben-Nun et al. 2018]

Cosmology [Oyama et al. 2019]





Big seq2seq models

Research opportunities

https://spcl.inf.ethz.ch/SeMa/
ndryden@ethz.ch

	Project Name	Category
+	Efficient partial collective operations for distributed deep learning training	Parallel Algorithms
+	Clairvoyant Prefetching for Machine Learning I/O	Machine Learning
+	Transformers: More than Meets the Eye (of the Hurricane)	Machine Learning
+	Scalable Deep Learning for Weather and Climate Prediction	Machine Learning
+	Fastest Matrix Multiplication in the West (Europe)	Parallel Algorithms
+	Efficient Collective Operations On Reconfigurable Hardware	Architecture
+	Quantized Allreduces for Distributed Deep Learning Training	Machine Learning
+	Who Optimizes the Optimizers? Performance Programming Made Easy	Toolchains
+	Analytical Cache Model for Parallel Programs	Compilation

+	Automatic Algorithm Detection for Readability and Performance Rewriting	Compilation
+	Data-Centric Deep Learning Framework (or: how to beat TensorFlow)	Machine Learning
+	Authenticated Deep Learning	Machine Learning
+	Kernel/Network Offloading of Streaming Data Processing Tasks	Networking
+	Performance Counters for Interactive Bottleneck Identification in Large-Scale Applications	Compilation
+	Automatic Learning of GPU Code Generation Parameters	Compilation
+	Array Partitioning to Exploit High-Bandwidth Block-RAM on FPGA	Compilation
+	Proofably optimal loop scheduling using MINLP	Compilation
+	Cache as RAM to accelerate x86 computations	Compilation
+	Visualization Techniques for Performance-Guided Programming	Compilation
+	Large Scale Framework for Code Analysis	Compilation
+	Deep Learning for Large-Scale Graph Analytics	Machine Learning

Parallelism in training DNNs – Summary

Parallelism in training DNNs – Summary

- **Deep learning is HPC – very similar – mainly dense linear algebra**
  - Amenable to our usual set of tricks, sometimes with a twist
- **Main bottleneck is communication – reduction by trading off**

Parameter Consistency
• Bounded synchronous SGD • Central vs. distributed parameter server • EASGD to ensemble learning

- **Strong scaling requires effort**
- **Very different environment from traditional HPC**
  - Trade-off accuracy for performance!
- **Performance-centric view in HPC can be harmful for accuracy!**

Parallelism in training DNNs – Summary

- **Deep learning is HPC – very similar – mainly dense linear algebra**
  - Amenable to our usual set of tricks, sometimes with a twist
- **Main bottleneck is communication – reduction by trading off**

Parameter Consistency
• Bounded synchronous SGD • Central vs. distributed parameter server • EASGD to ensemble learning

- **Strong scaling requires effort**
- **Very different environment from traditional HPC**
  - Trade-off accuracy for performance!
- **Performance-centric view in HPC can be harmful for accuracy!**

Parallelism in training DNNs – Summary

- **Deep learning is HPC – very similar – mainly dense linear algebra**
  - Amenable to our usual set of tricks, sometimes with a twist
- **Main bottleneck is communication – reduction by trading off**

Parameter Consistency	Parameter Accuracy
• Bounded synchronous SGD • Central vs. distributed parameter server • EASGD to ensemble learning	• Lossless compression of gradient updates • Quantization of gradient updates • Sparsification of gradient updates

- **Strong scaling requires effort**
- **Very different environment from traditional HPC**
  - Trade-off accuracy for performance!
- **Performance-centric view in HPC can be harmful for accuracy!**

Parallelism in training DNNs – Summary

- **Deep learning is HPC – very similar – mainly dense linear algebra**
  - Amenable to our usual set of tricks, sometimes with a twist
- **Main bottleneck is communication – reduction by trading off**

Parameter Consistency	Parameter Accuracy
• Bounded synchronous SGD • Central vs. distributed parameter server • EASGD to ensemble learning	• Lossless compression of gradient updates • Quantization of gradient updates • Sparsification of gradient updates

- **Strong scaling requires effort**
- **Very different environment from traditional HPC**
  - Trade-off accuracy for performance!
- **Performance-centric view in HPC can be harmful for accuracy!**

Parallelism in training DNNs – Summary

- **Deep learning is HPC – very similar – mainly dense linear algebra**
  - Amenable to our usual set of tricks, sometimes with a twist
- **Main bottleneck is communication – reduction by trading off**

Parameter Consistency	Parameter Accuracy
• Bounded synchronous SGD • Central vs. distributed parameter server • EASGD to ensemble learning	• Lossless compression of gradient updates • Quantization of gradient updates • Sparsification of gradient updates

- **Strong scaling requires effort**
- **Very different environment from traditional HPC**
  - Trade-off accuracy for performance!
- **Performance-centric view in HPC can be harmful for accuracy!**