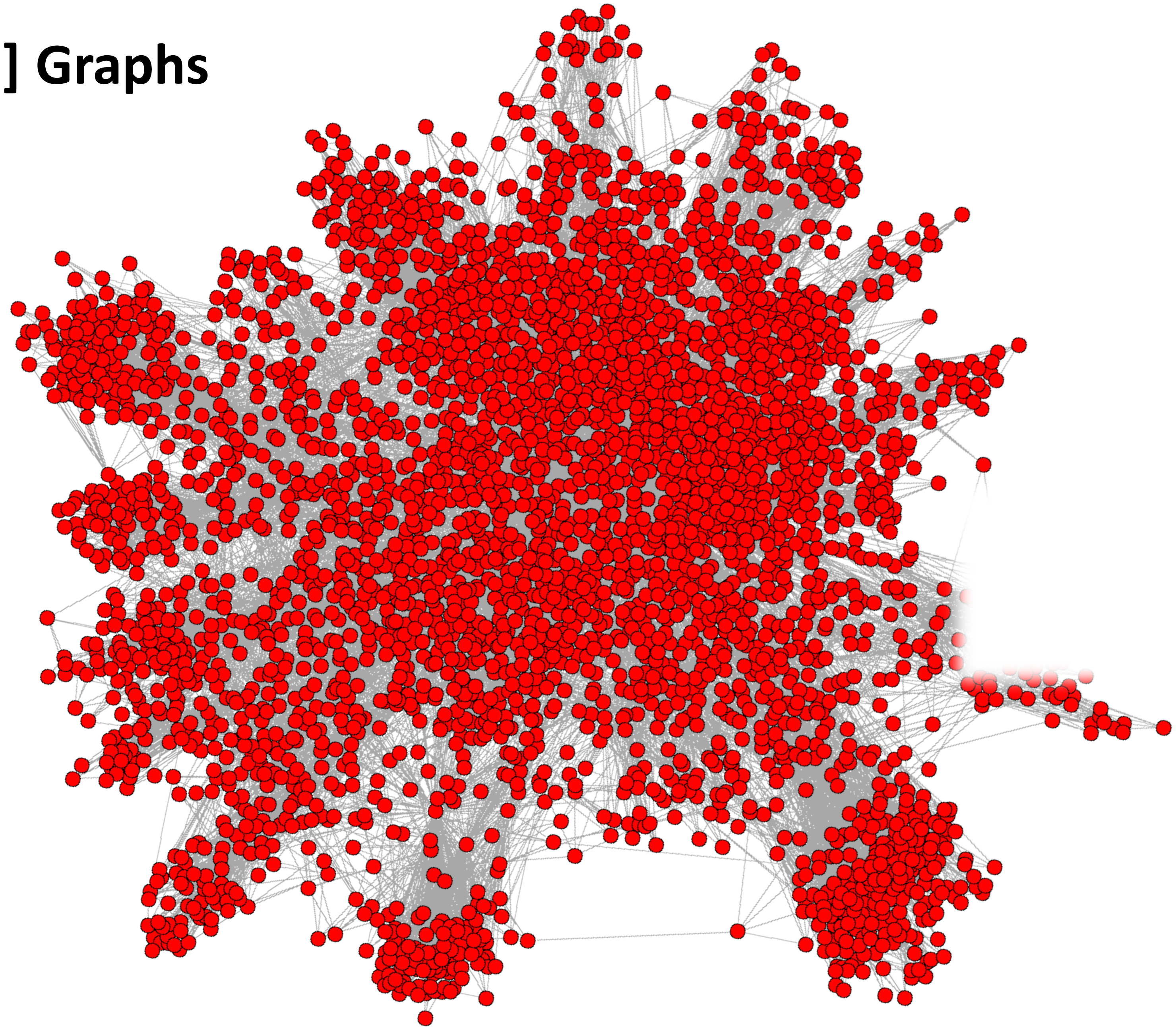
M. Besta, T. Hoefler

Towards High-Performance Processing, Storage, and Analytics of Extreme-Scale Graphs

With contributions from Dimitri Stanojevic, Simon Weber, Lukas Gianinazzi, Andrey Ivanov, Marc Fischer, Robert Gerstenberger, Heng Lin, Xiaowei Zhu, Bowen Yu, Xiongchao Tang, Wei Xue, Wenguang Chen, Lufei Zhang, Xiaosong Ma, Xin Liu, Weimin Zheng, and Jingfang Xu and others.

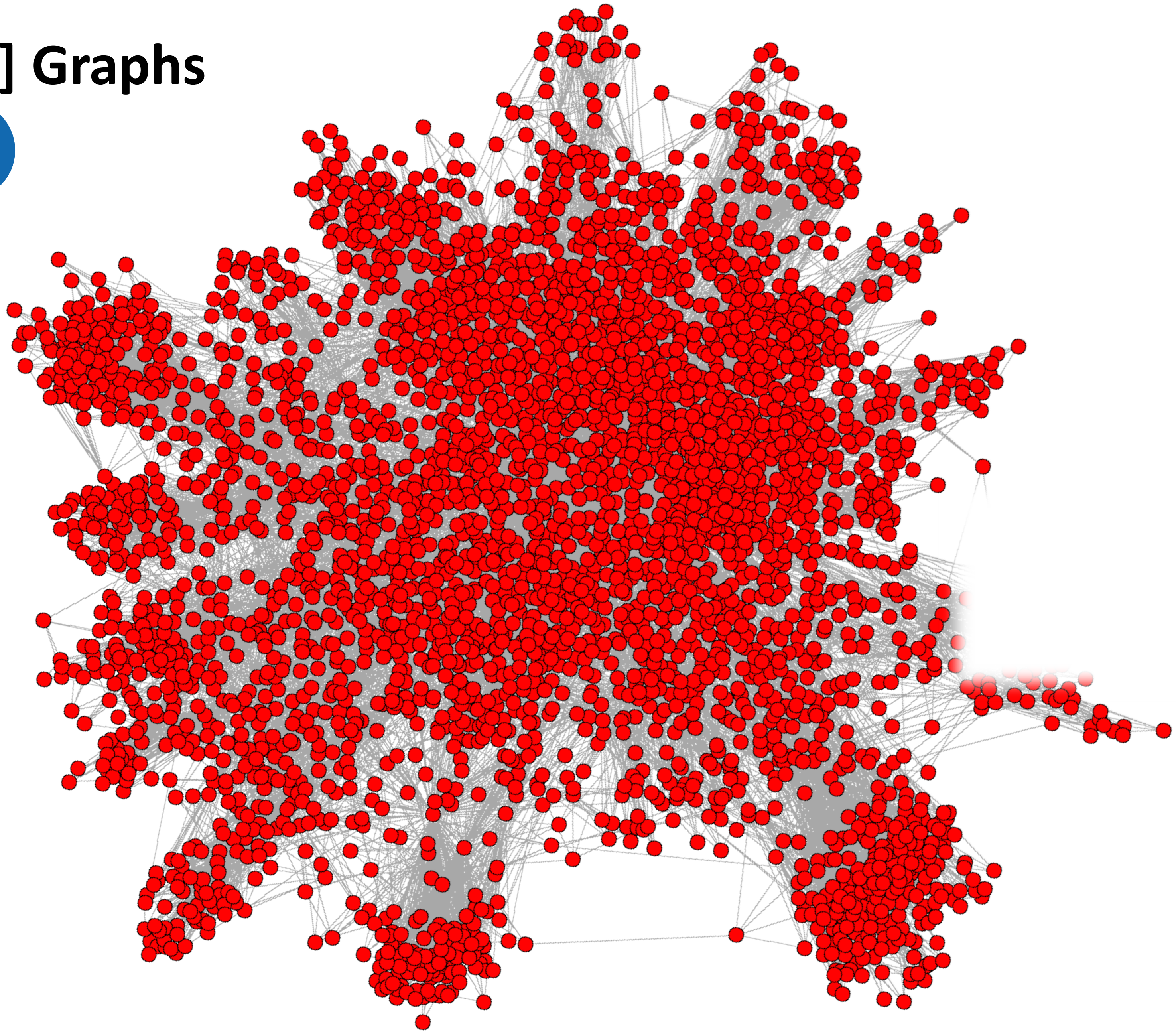
[Extreme-Scale] Graphs

[Extreme-Scale] Graphs



[Extreme-Scale] Graphs

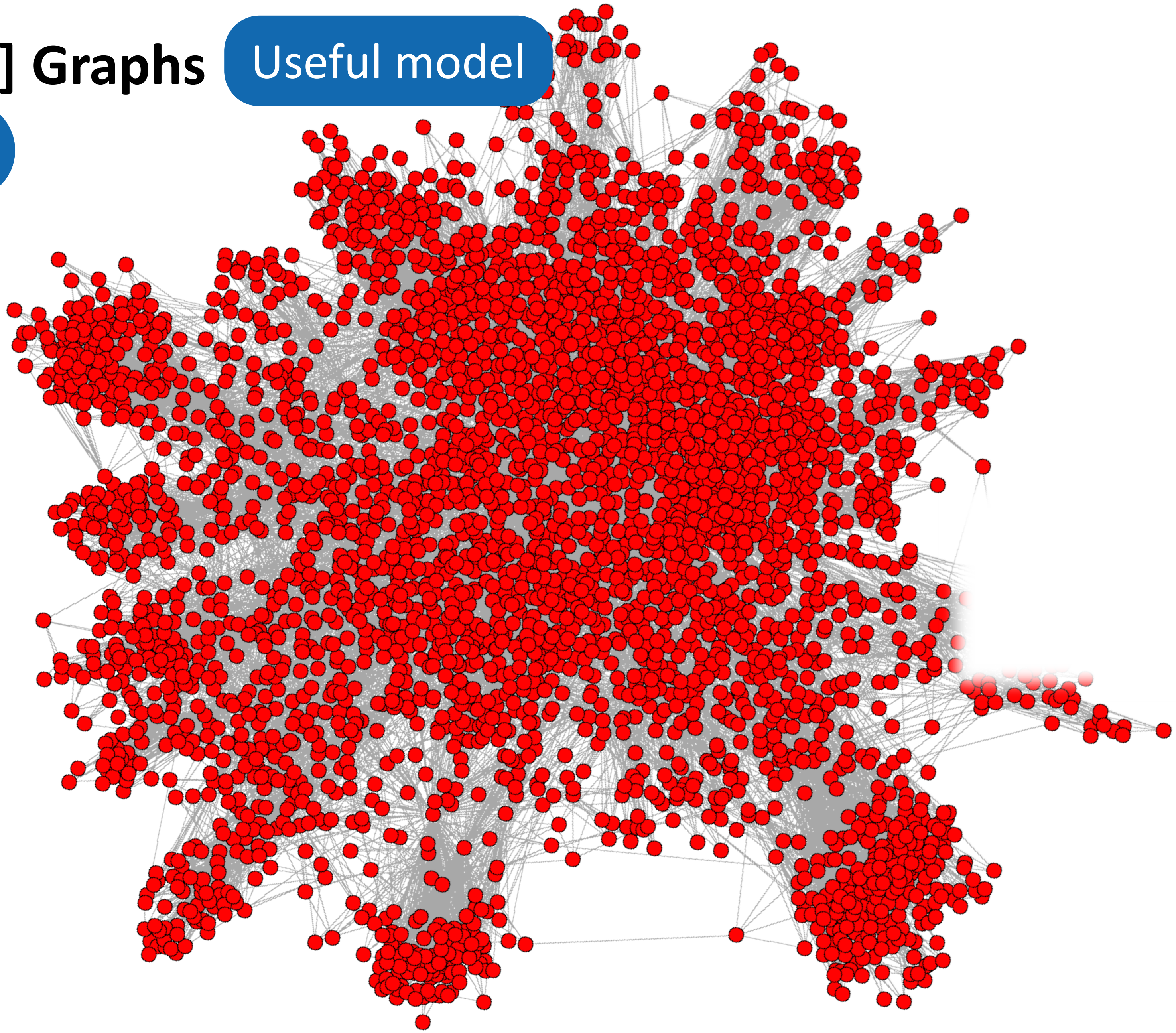
Why do we care?



[Extreme-Scale] Graphs

Useful model

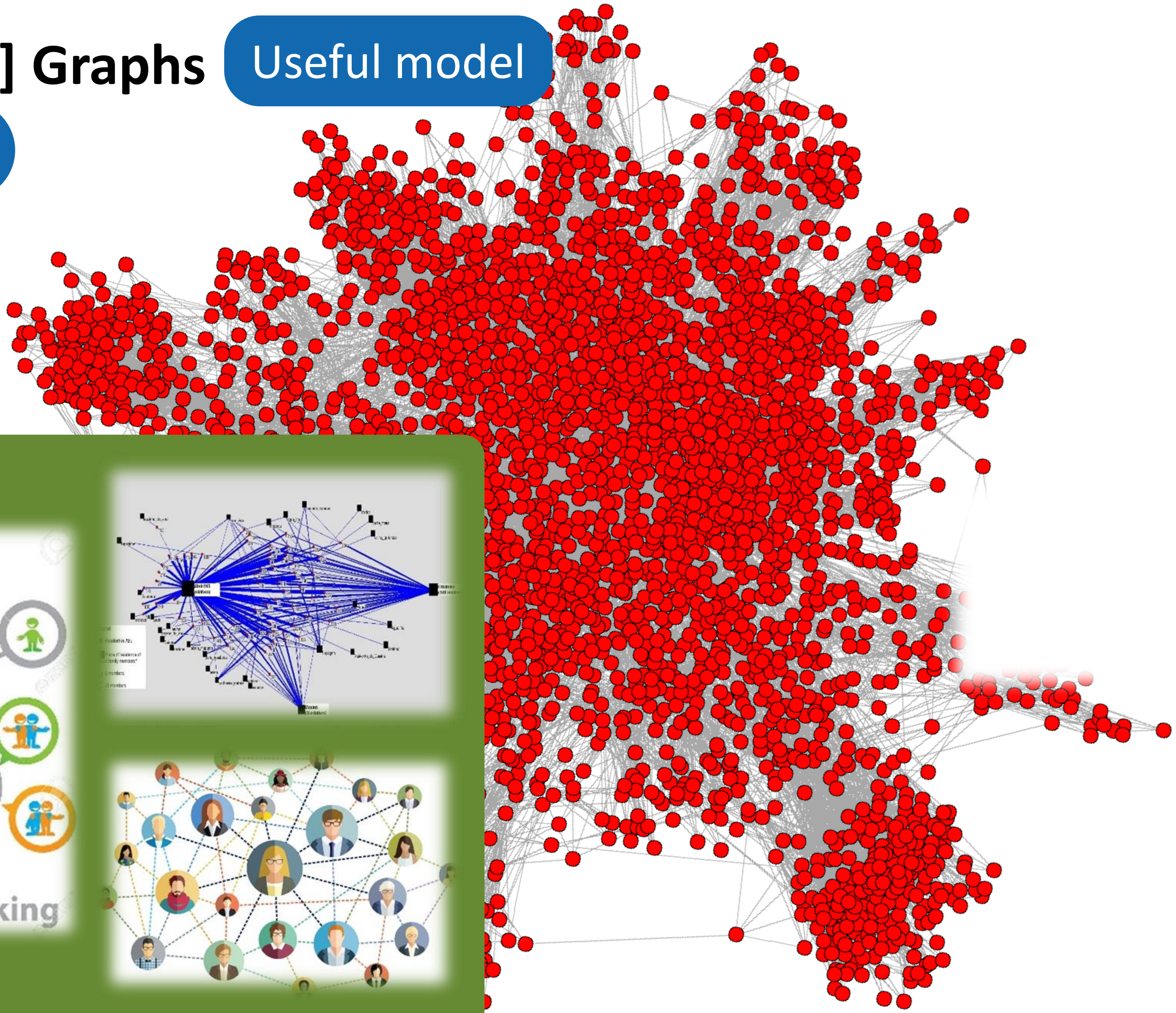
Why do we care?



[Extreme-Scale] Graphs

Useful model

Why do we care?

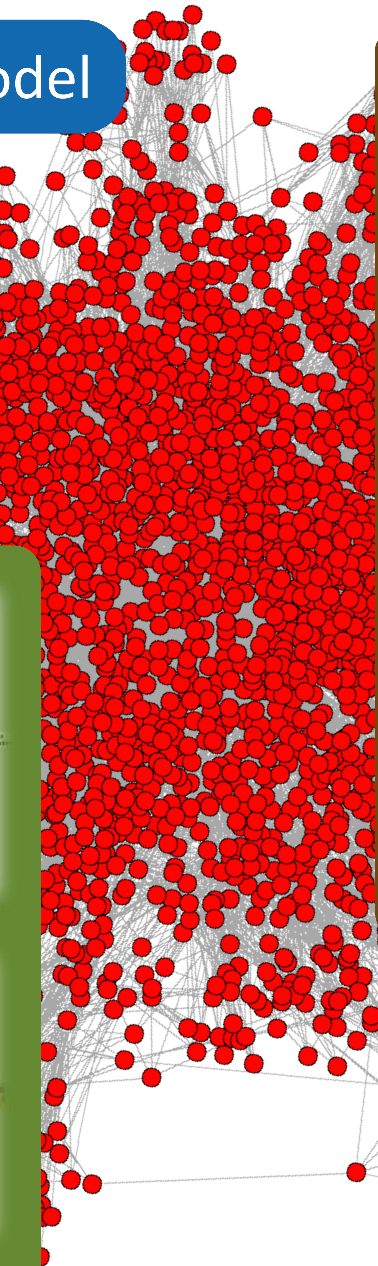


Social networks

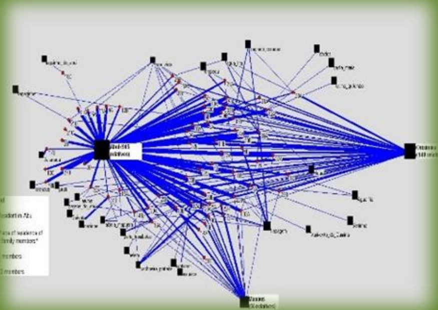
[Extreme-Scale] Graphs

Useful model

Why do we care?



Social networks



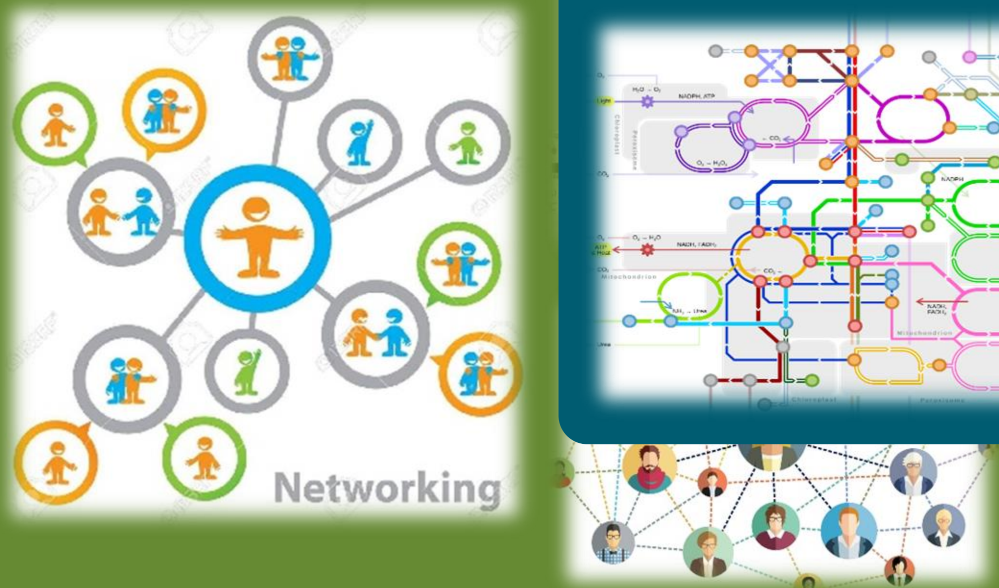
Engineering networks

[Extreme-Scale] Graphs

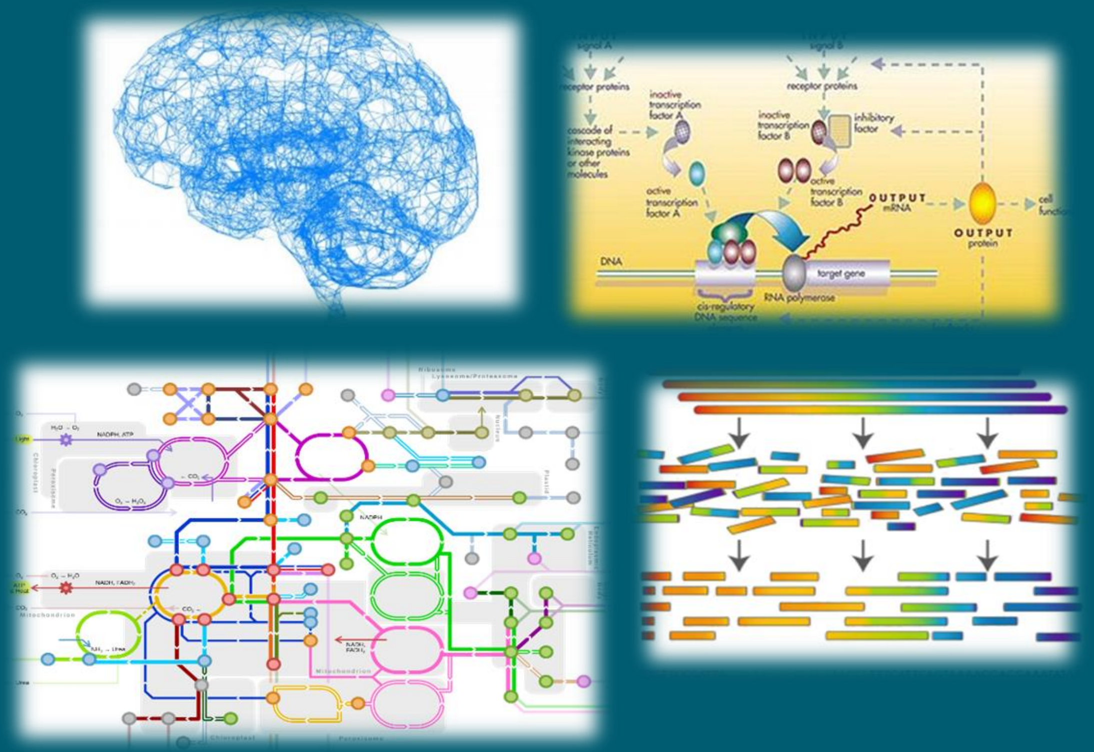
Useful model

Why do we care?

Social networks



Biological networks



Engineering networks

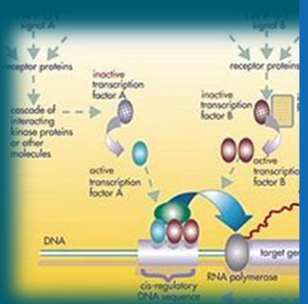
[Extreme-Scale] Graphs

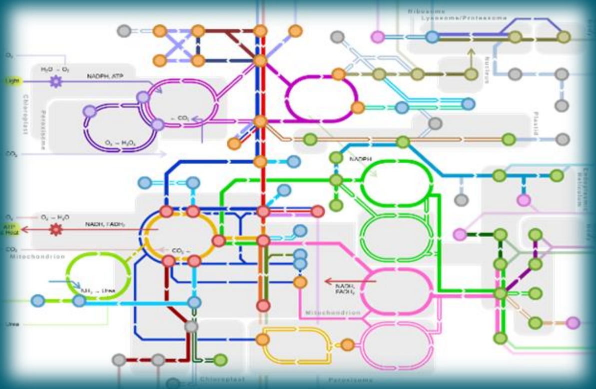
Useful model

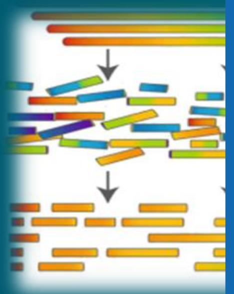
Why do we care?

Social networks

Biological networks

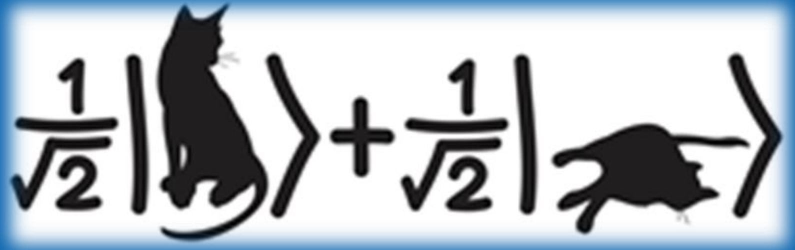


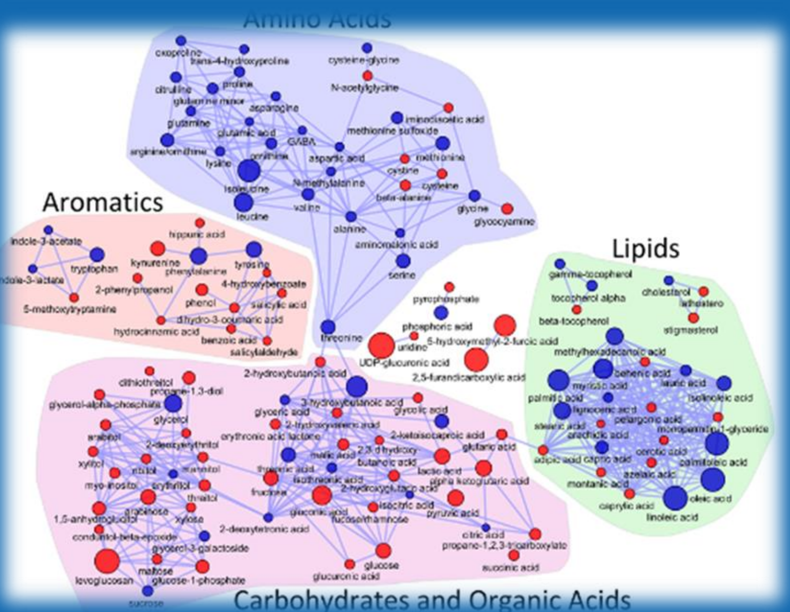




Engineering networks

Physics, chemistry





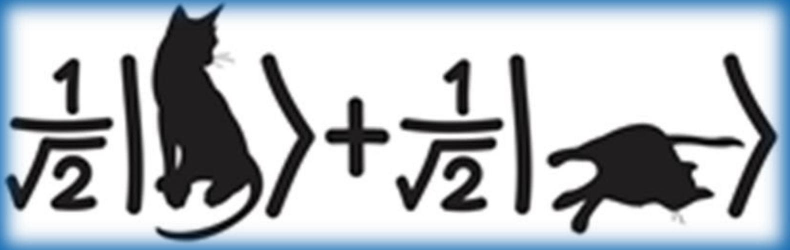
[Extreme-Scale] Graphs

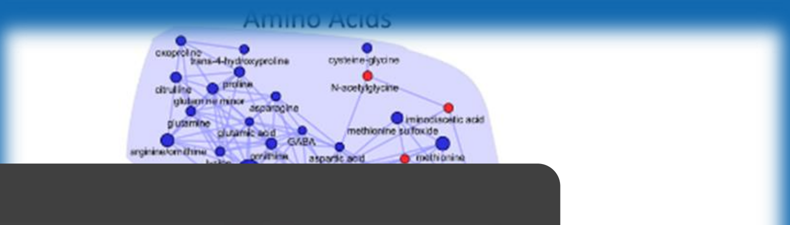
Useful model

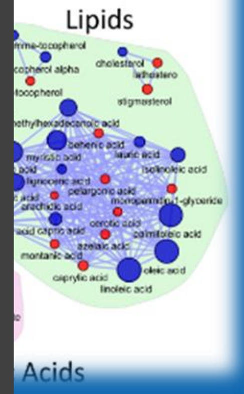
Why do we care?

Engineering networks

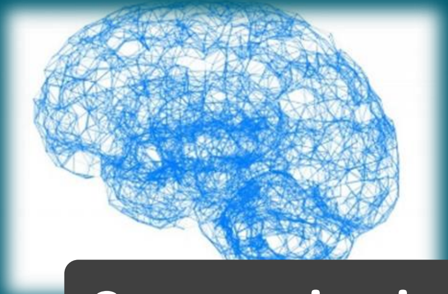
Physics, chemistry

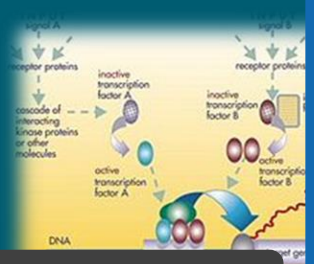






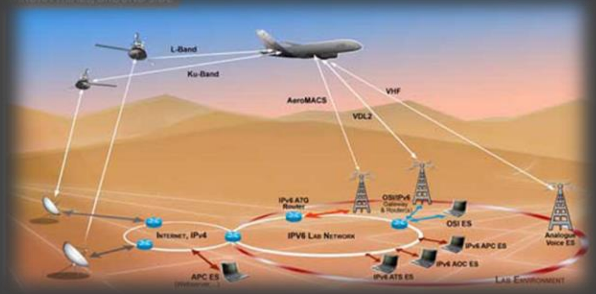
Biological networks

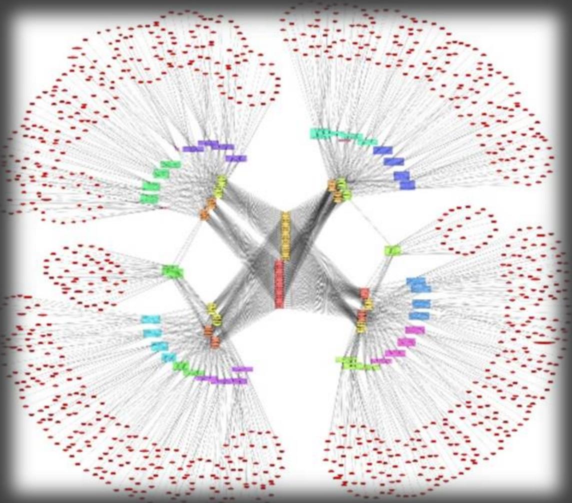




Social networks

Communication networks

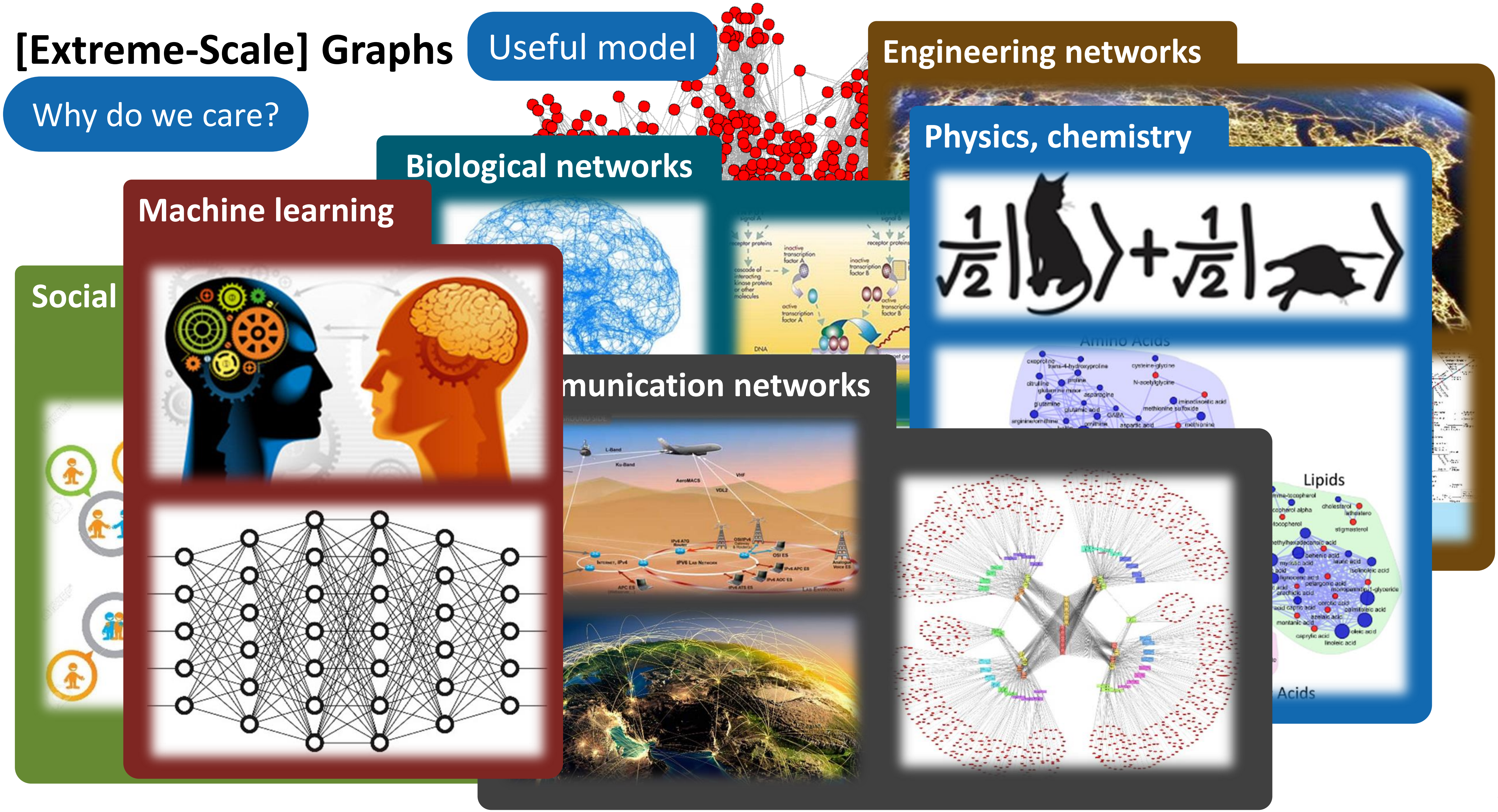




[Extreme-Scale] Graphs

Useful model

Why do we care?



[Extreme-Scale] Graphs

Useful model

Why do we care?

Engineering networks

...even philosophy ☺

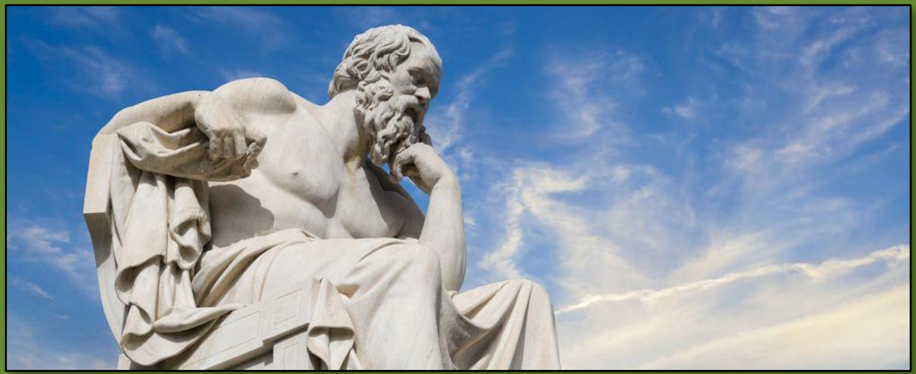
hysics, chemistry

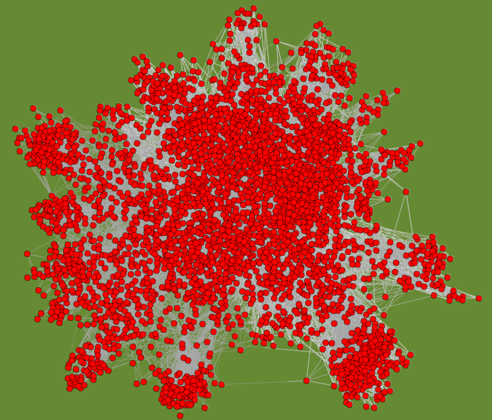
Biological ne

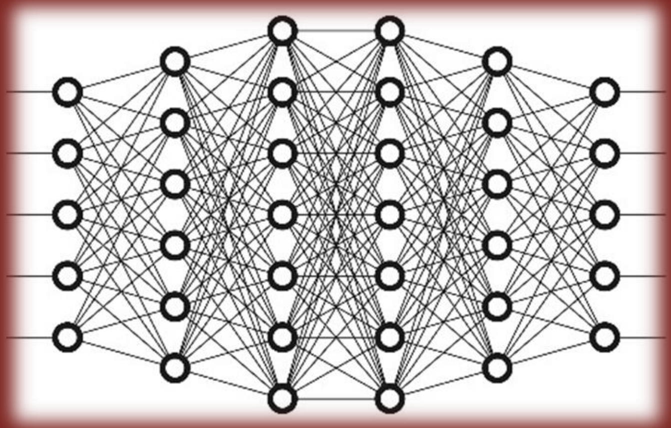
Machine learning

Social

m







FOSDEM 2016 / Schedule / Events / Developer rooms / Graph Processing / Modeling a Philosophical Inquiry: from MySQL to a graph database

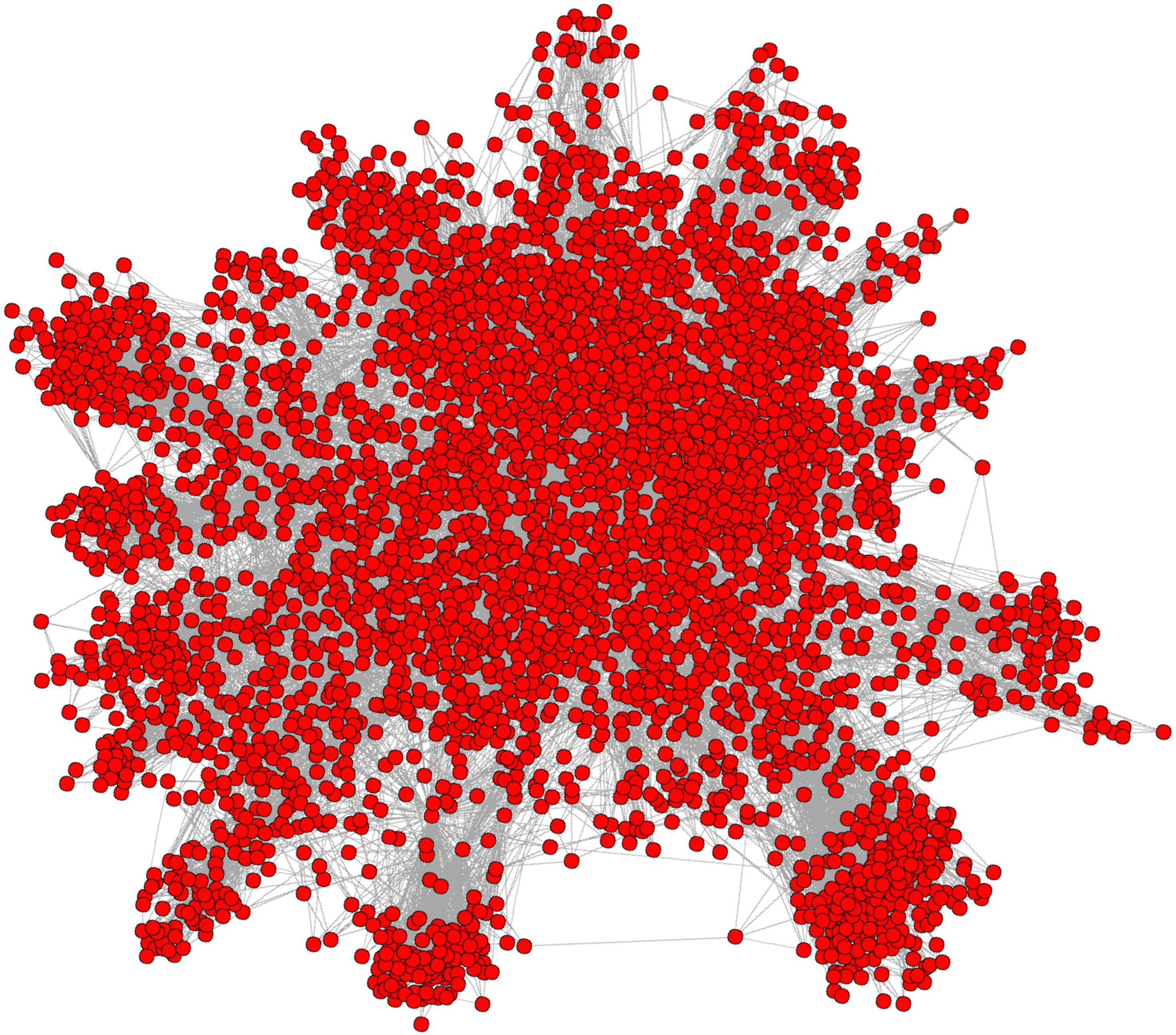
Modeling a Philosophical Inquiry: from MySQL to a graph database

The short story of a long refactoring process

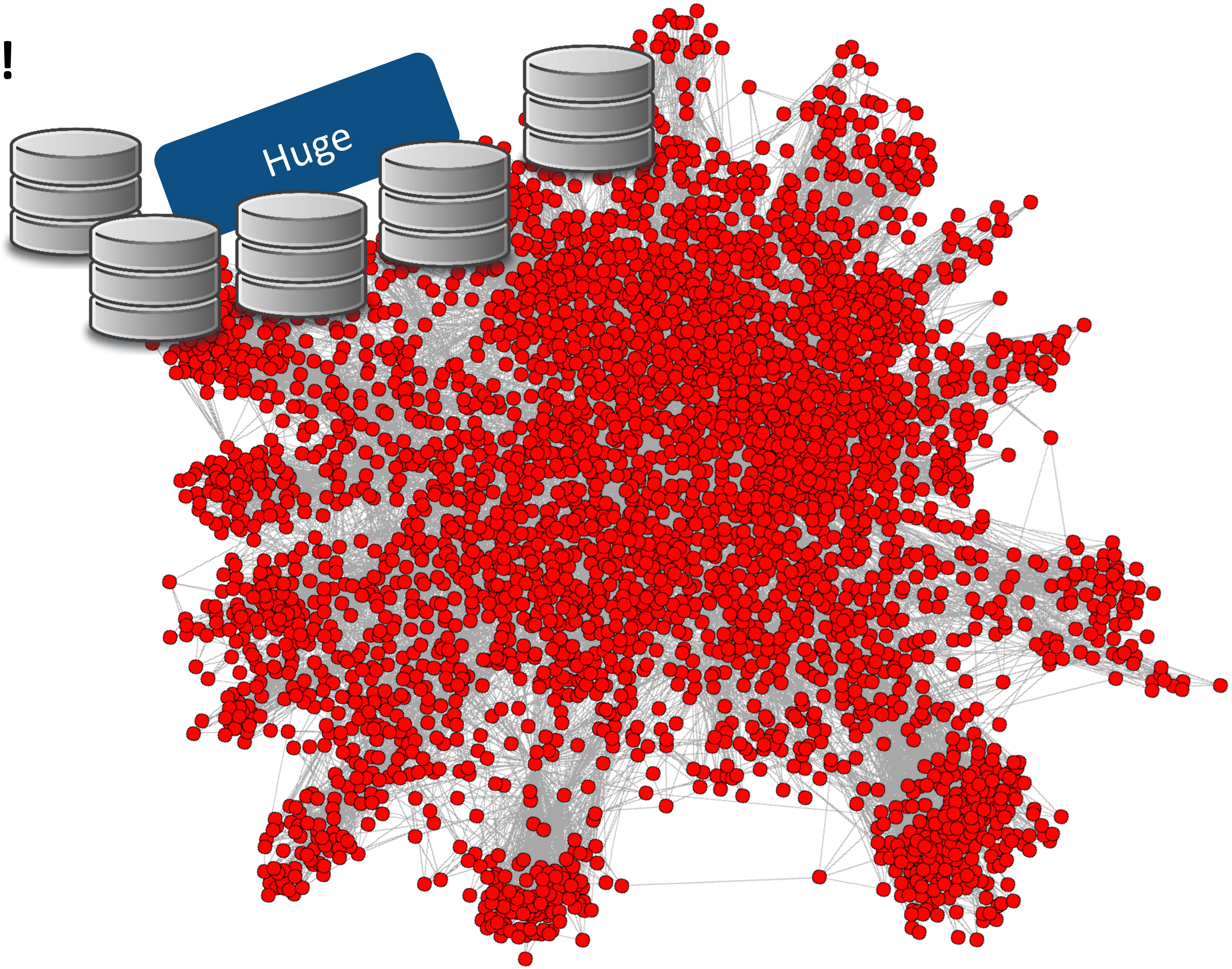
- **Track**: Graph Processing devroom
- **Room**: AW1.126
- **Day**: Saturday
- **Start**: 12:45
- **End**: 13:35

Bruno Latour wrote a book about philosophy (an inquiry into modes of existence). He decided that the paper book was no place for the numerous footnotes, documentation or glossary, instead giving access to all this information surrounding the book through a web application which would present itself as a reading companion. He also offered to the community of readers to submit their contributions to his inquiry by writing new documents to be added to the platform. The first version

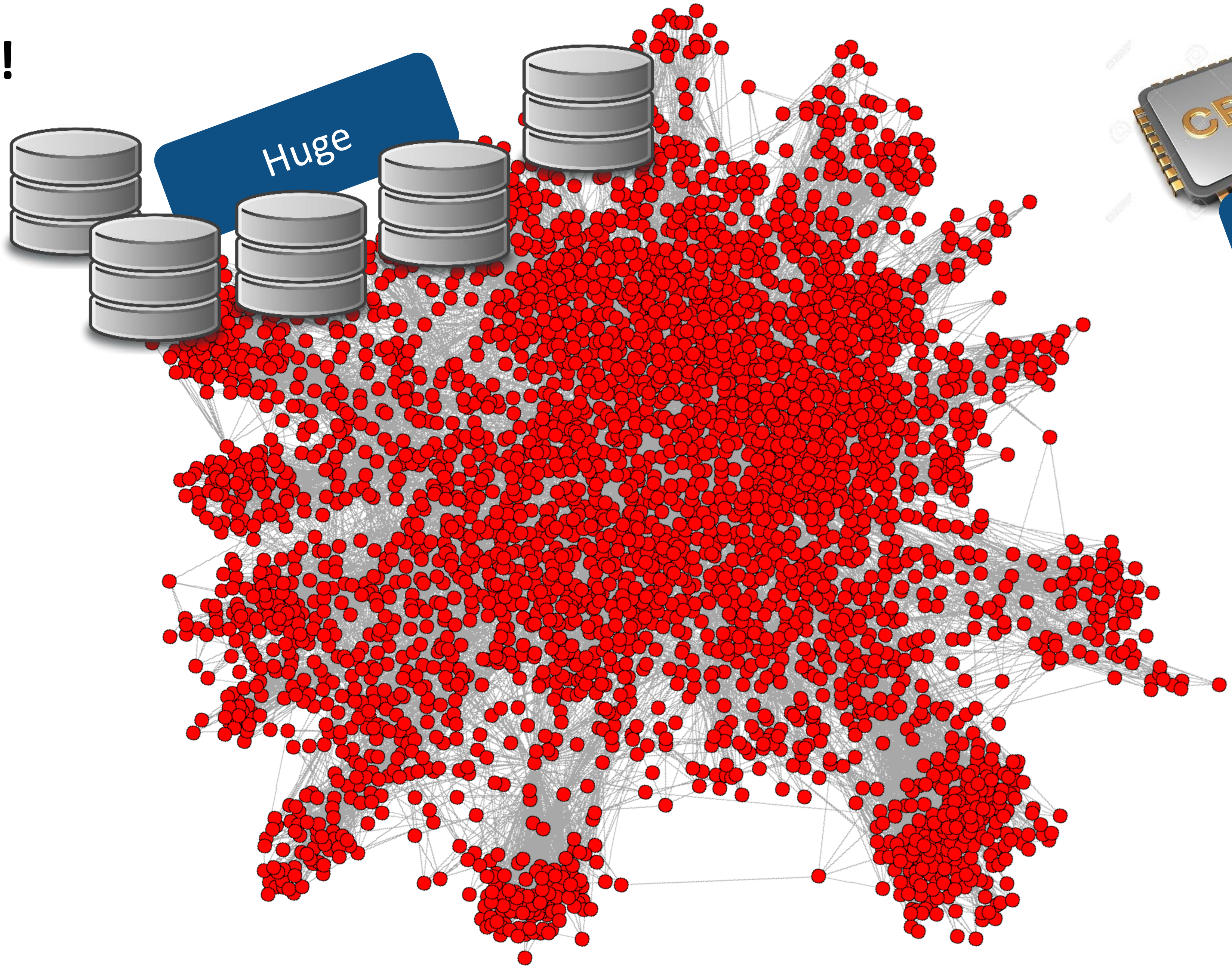
Problems!

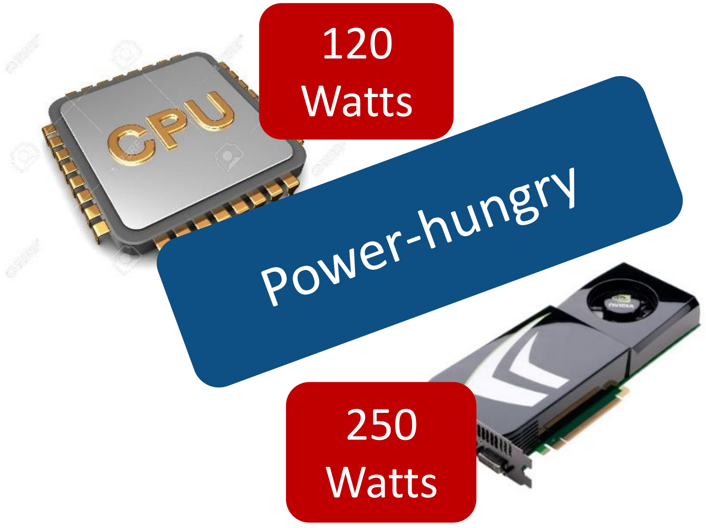


Problems!



Problems!





Problems!


Huge
Communication-heavy
CPU
120 Watts
Power-hungry
250 Watts

Problems!


Huge
Communication-heavy
Synchronization-heavy
Power-hungry
CPU
120
Watts
250
Watts

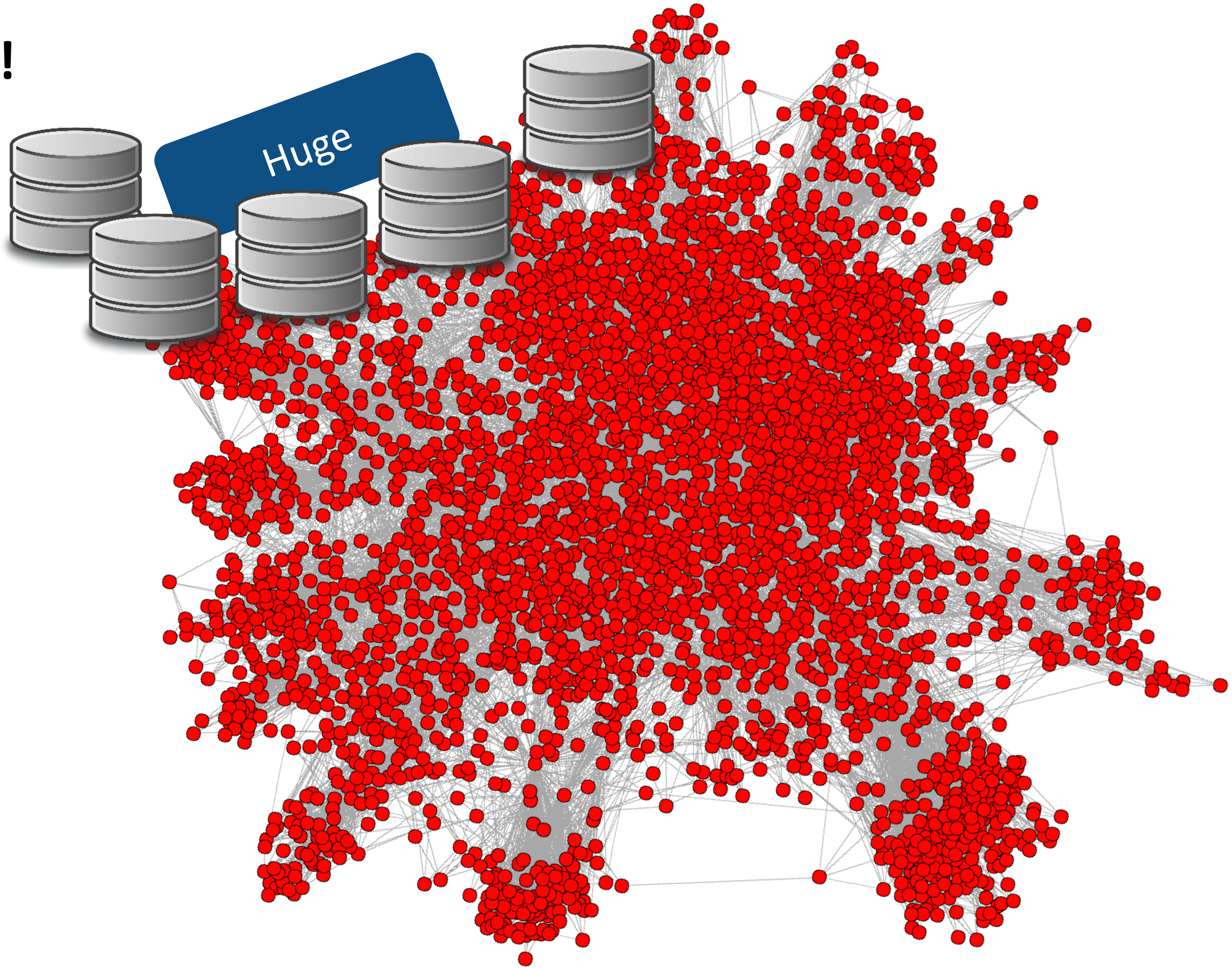
Problems!


Huge
Power-hungry
120 Watts
250 Watts
Communication-heavy
Synchronization-heavy
Irregular

Problems!


Huge
120
Watts
CPU
Power-hungry
250
Watts
It's Complicated...
Communication-heavy
Synchronization-heavy
Irregular

Problems!



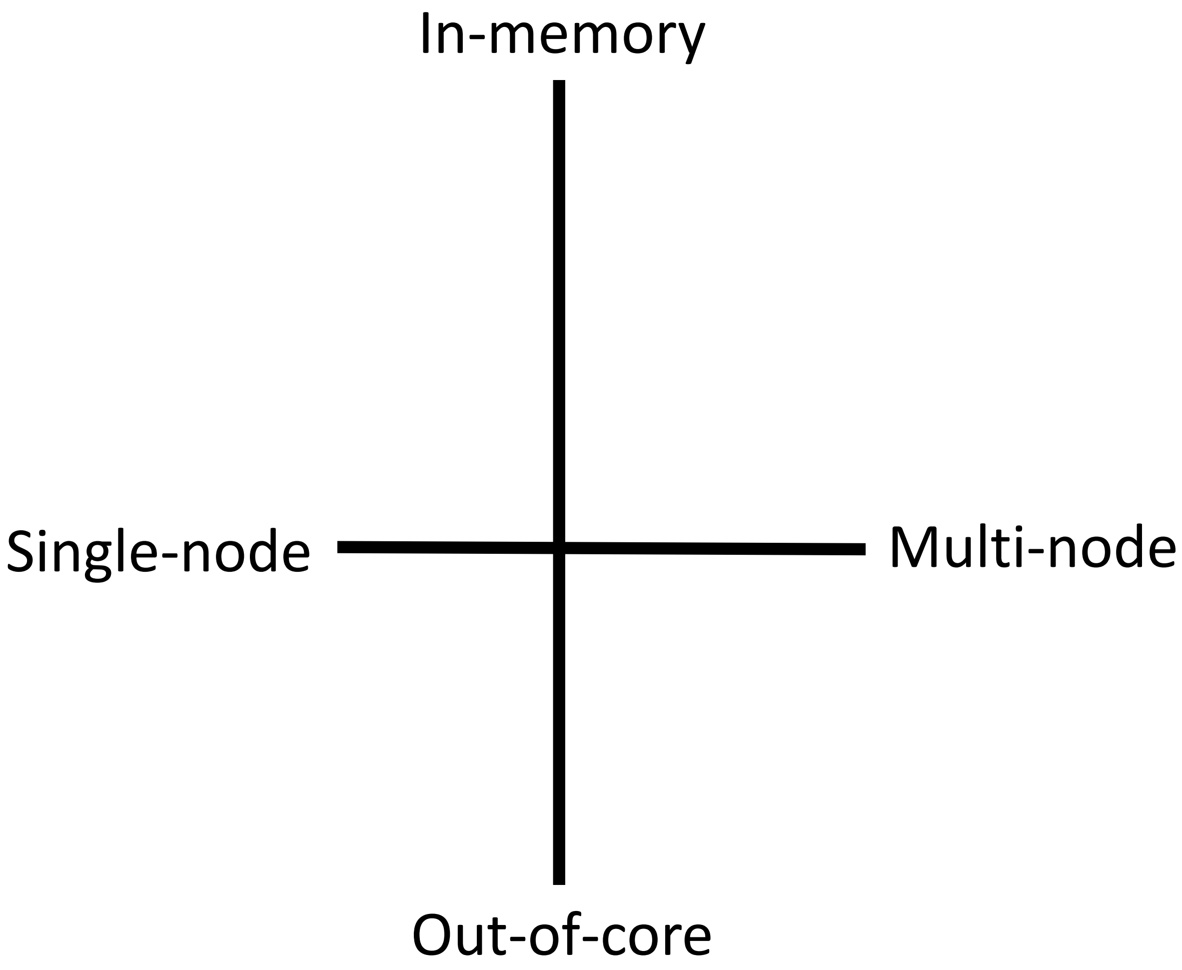
Problems!

What does "huge" mean?

Practice of Extreme-Scale Graph Processing

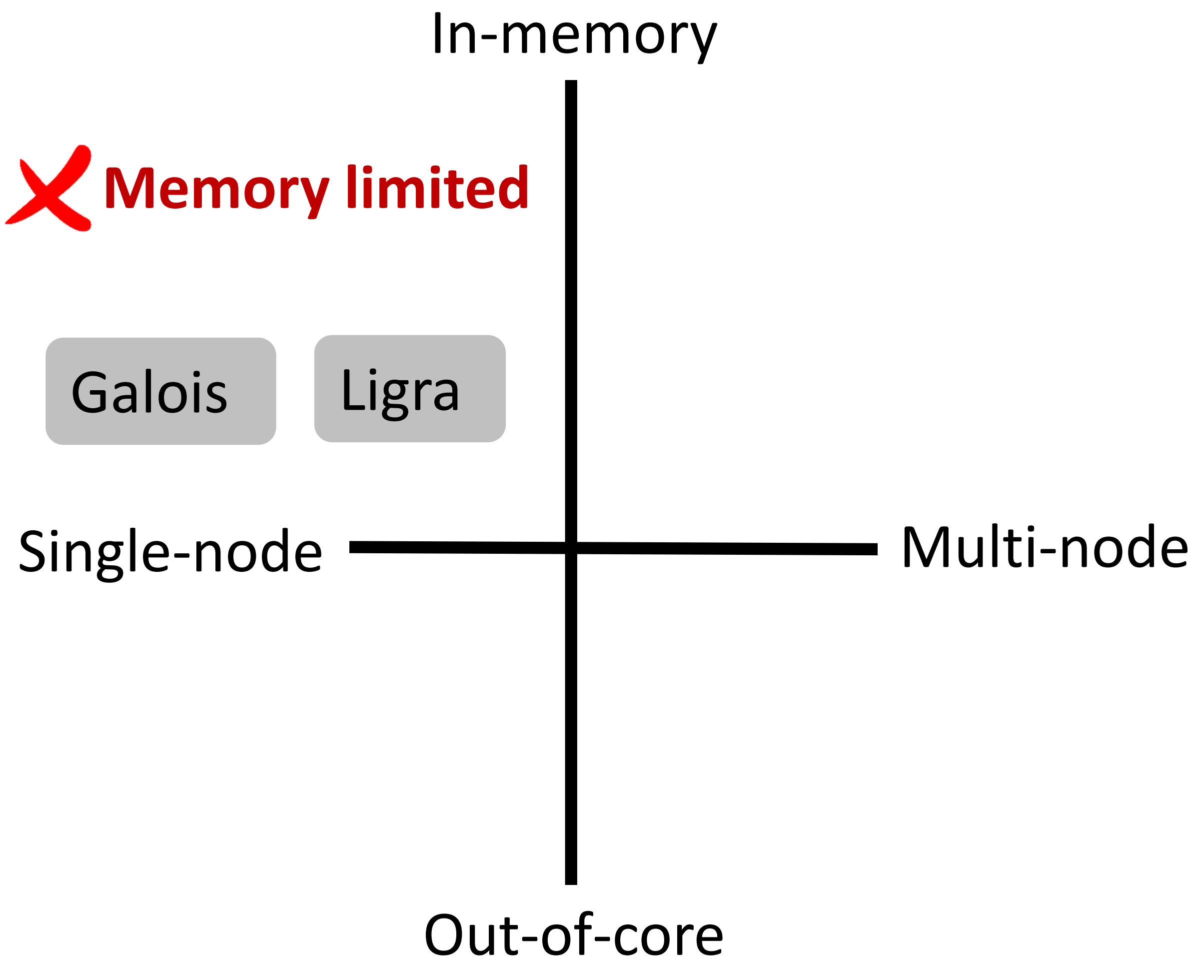
[1] Heng Lin et al.: ShenTu: Processing Multi-Trillion Edge Graphs on Millions of Cores in Seconds, SC18, Gordon Bell Finalist

Practice of Extreme-Scale Graph Processing



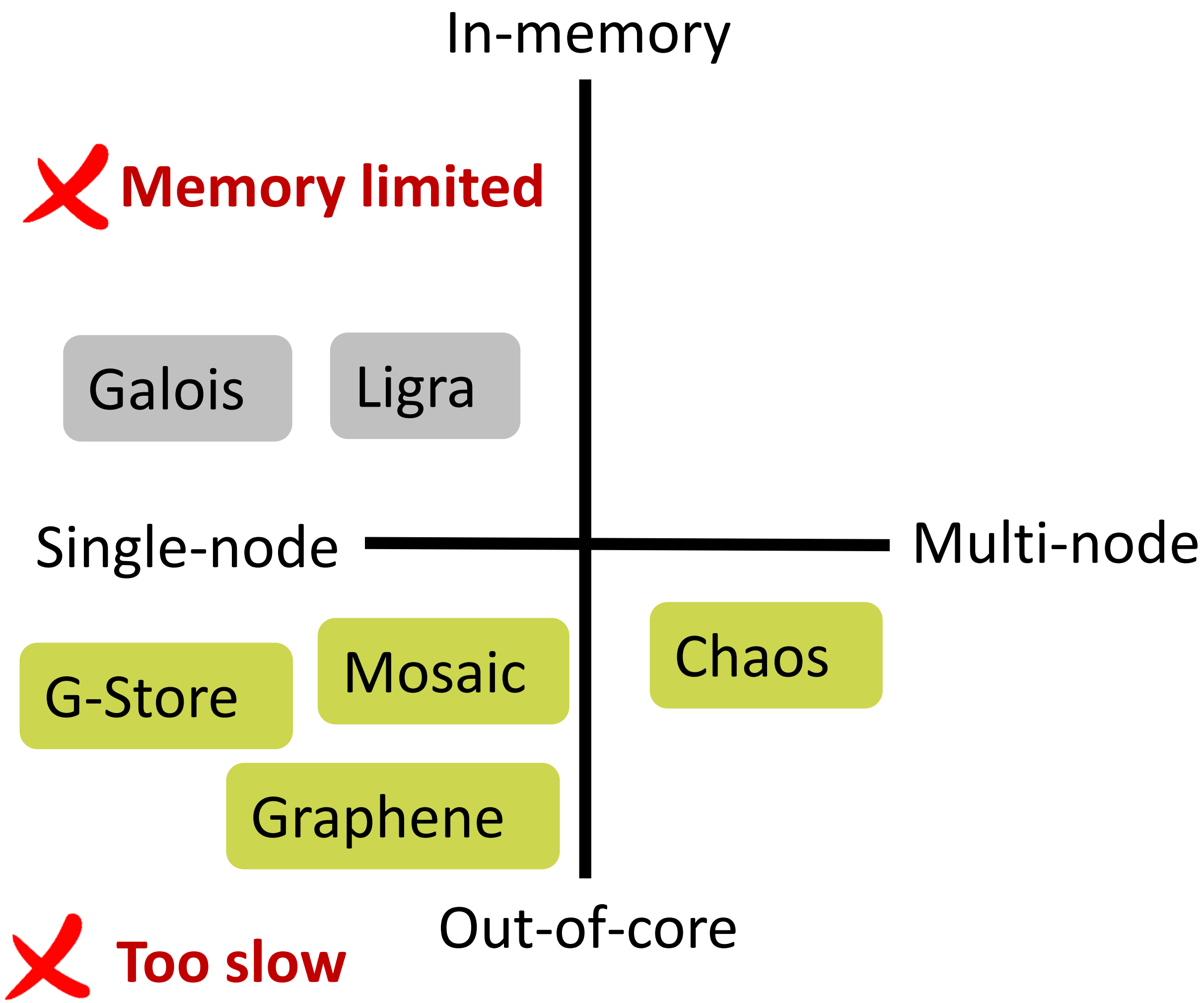
[1] Heng Lin et al.: ShenTu: Processing Multi-Trillion Edge Graphs on Millions of Cores in Seconds, SC18, Gordon Bell Finalist

Practice of Extreme-Scale Graph Processing



[1] Heng Lin et al.: ShenTu: Processing Multi-Trillion Edge Graphs on Millions of Cores in Seconds, SC18, Gordon Bell Finalist

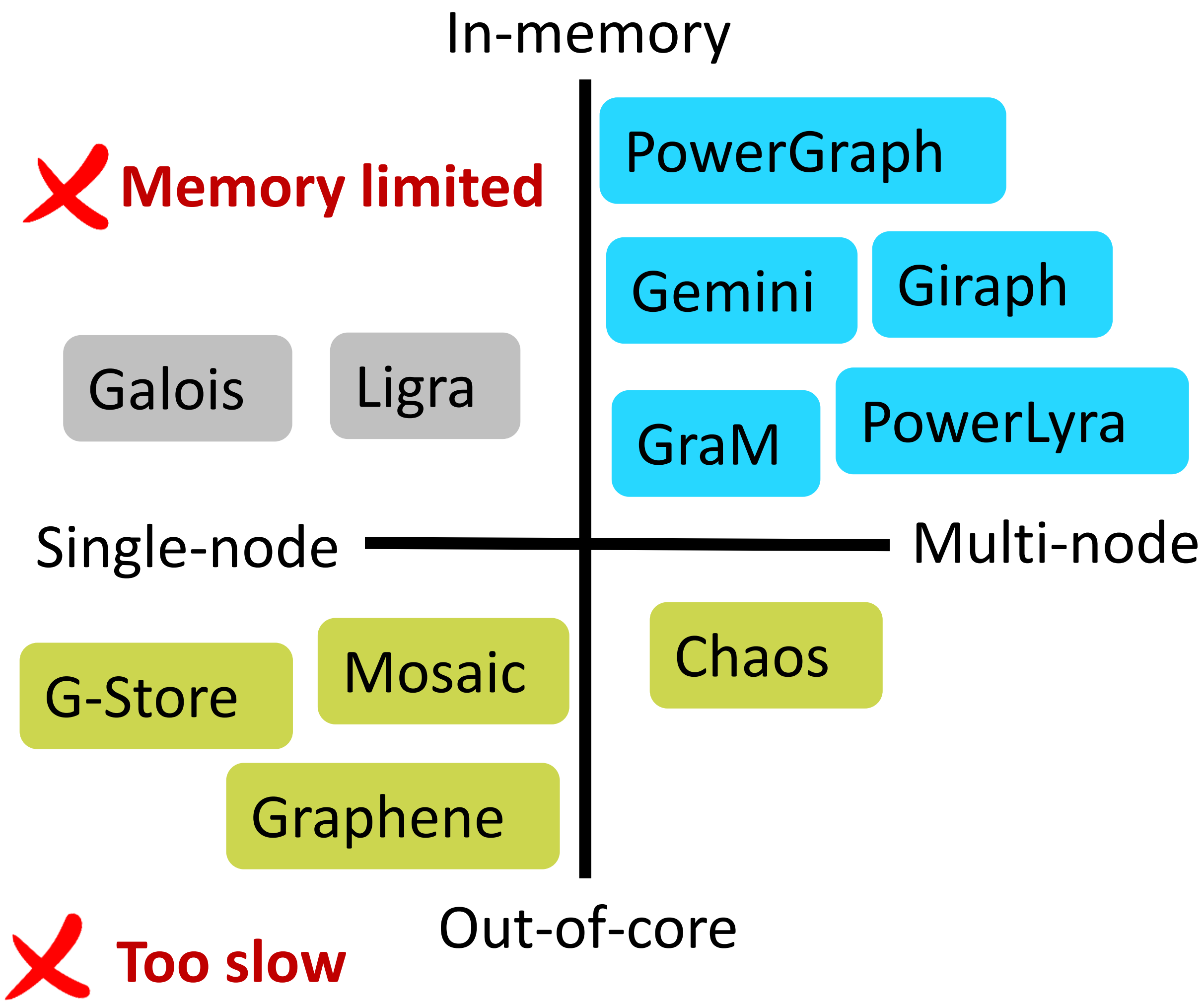
Practice of Extreme-Scale Graph Processing



(> 20 mins per PageRank iteration on 1-trillion-edge graph)

[1] Heng Lin et al.: ShenTu: Processing Multi-Trillion Edge Graphs on Millions of Cores in Seconds, SC18, Gordon Bell Finalist

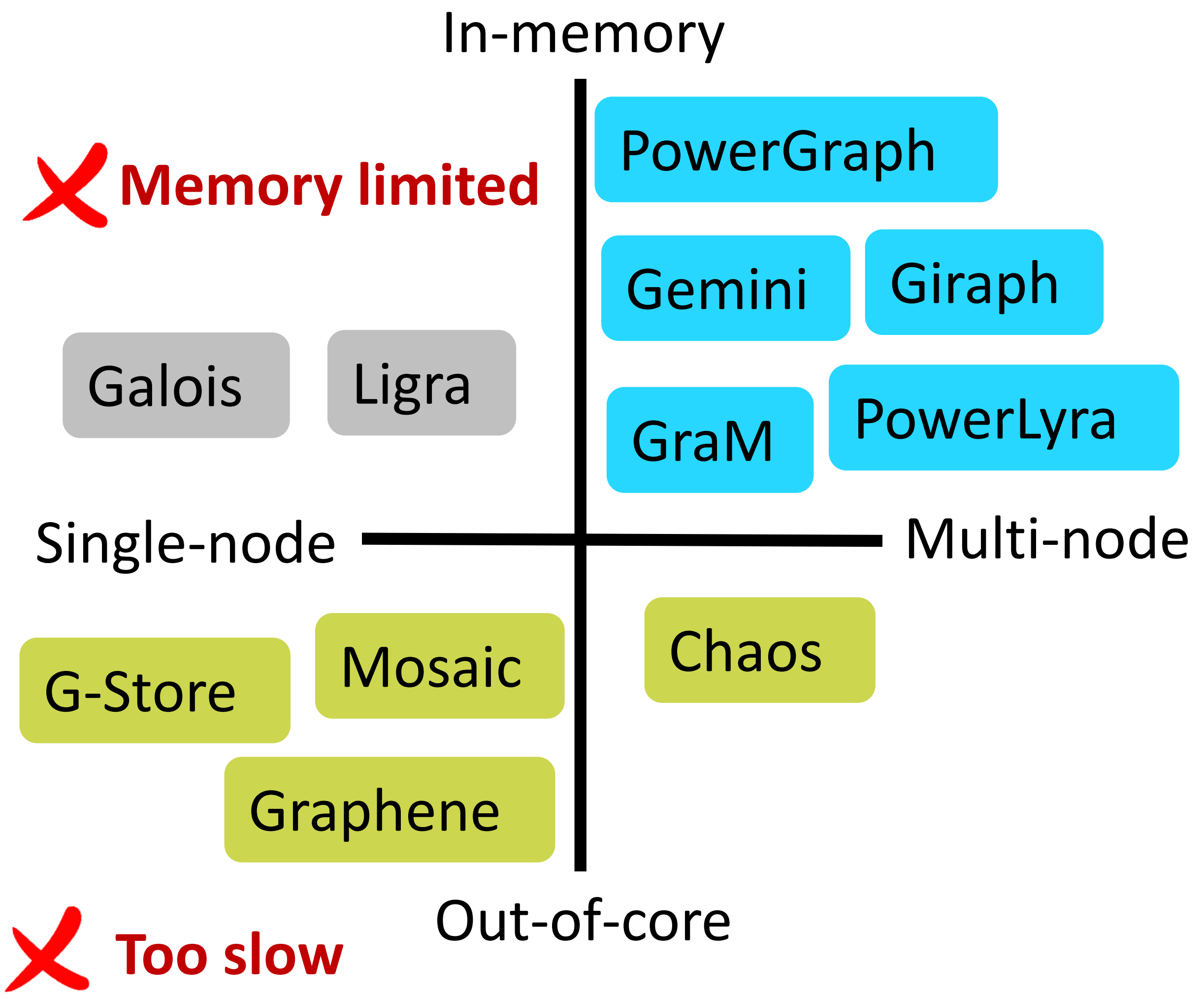
Practice of Extreme-Scale Graph Processing



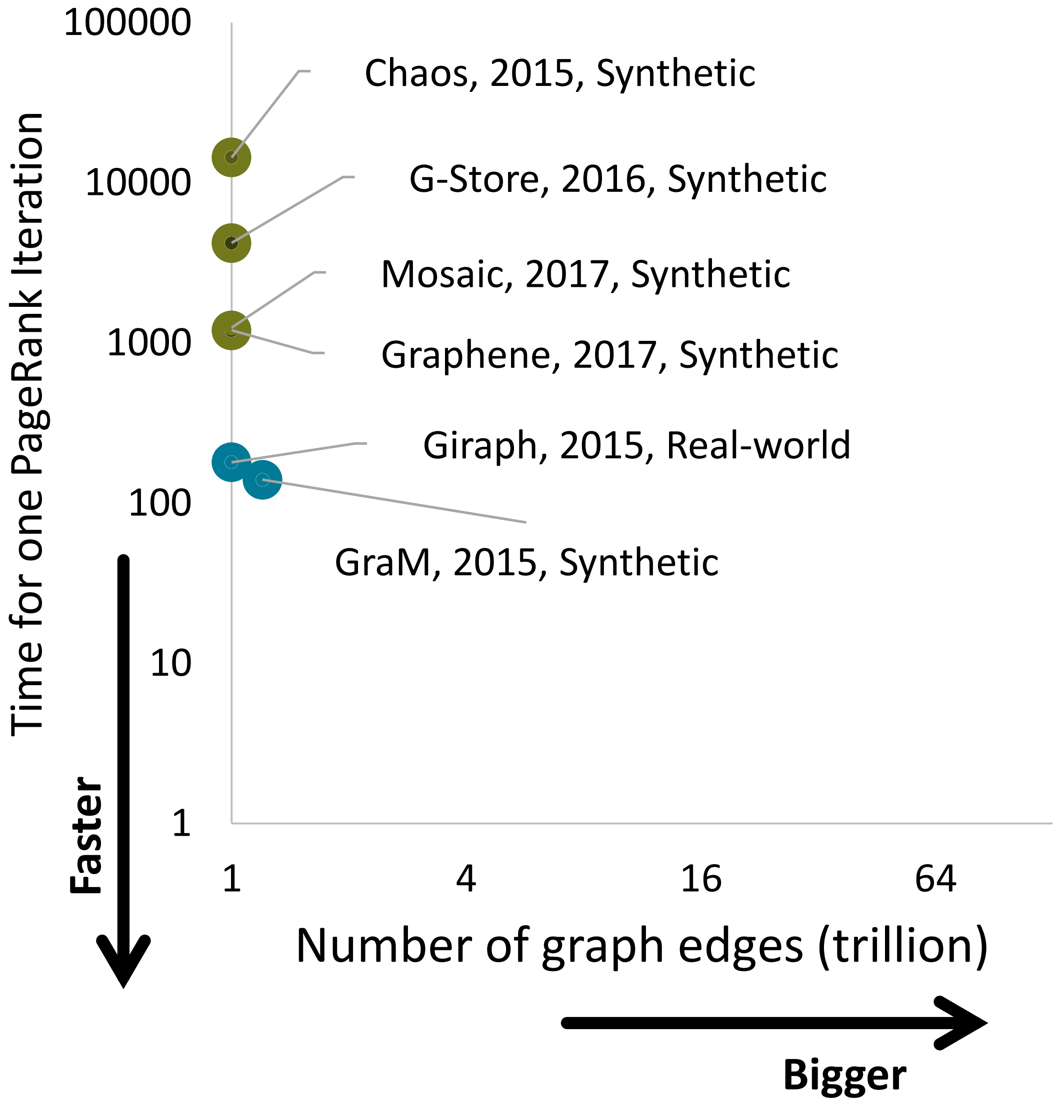
(> 20 mins per PageRank iteration on 1-trillion-edge graph)

[1] Heng Lin et al.: ShenTu: Processing Multi-Trillion Edge Graphs on Millions of Cores in Seconds, SC18, Gordon Bell Finalist

Practice of Extreme-Scale Graph Processing

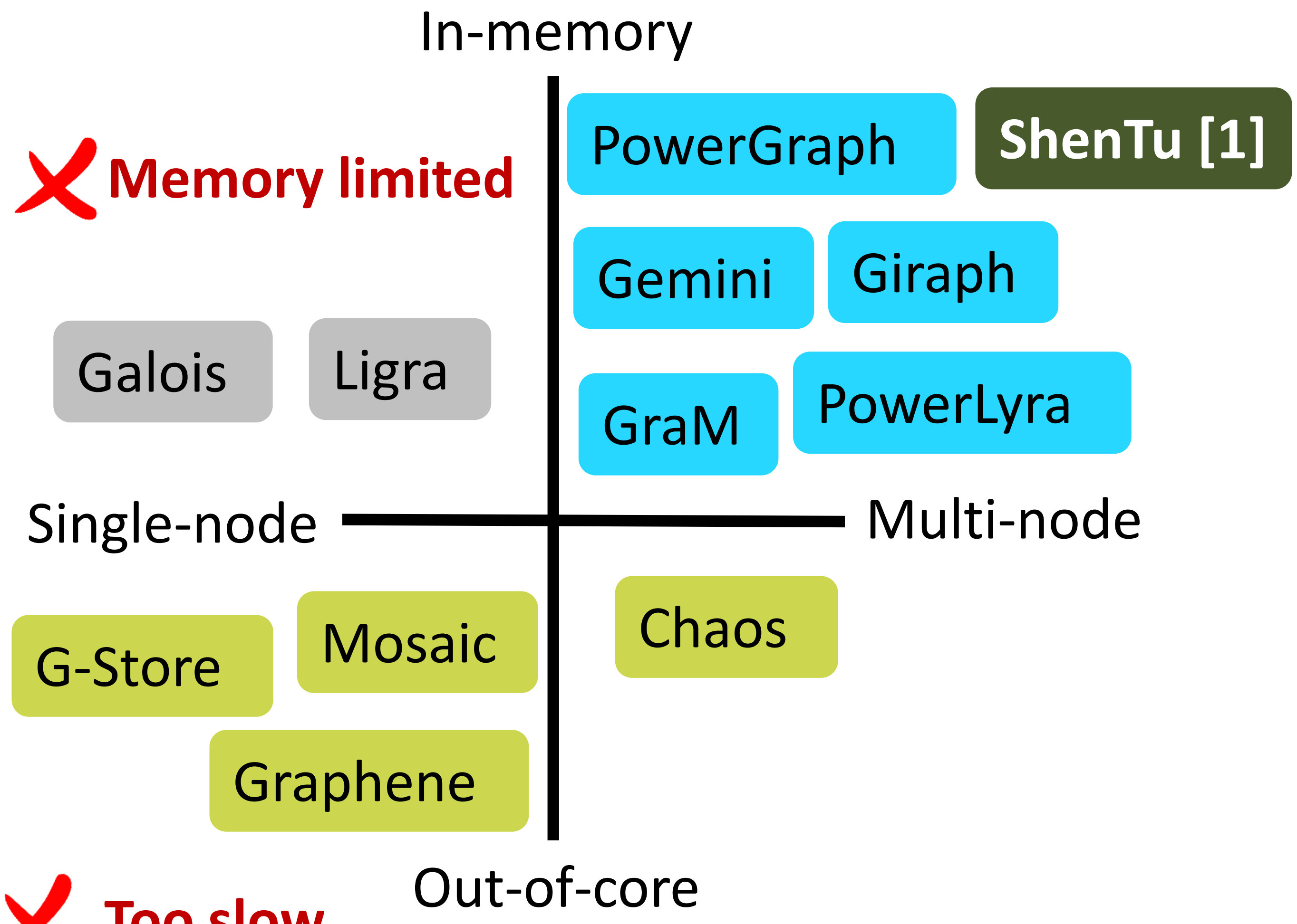


(> 20 mins per PageRank iteration on 1-trillion-edge graph)



[1] Heng Lin et al.: ShenTu: Processing Multi-Trillion Edge Graphs on Millions of Cores in Seconds, SC18, Gordon Bell Finalist

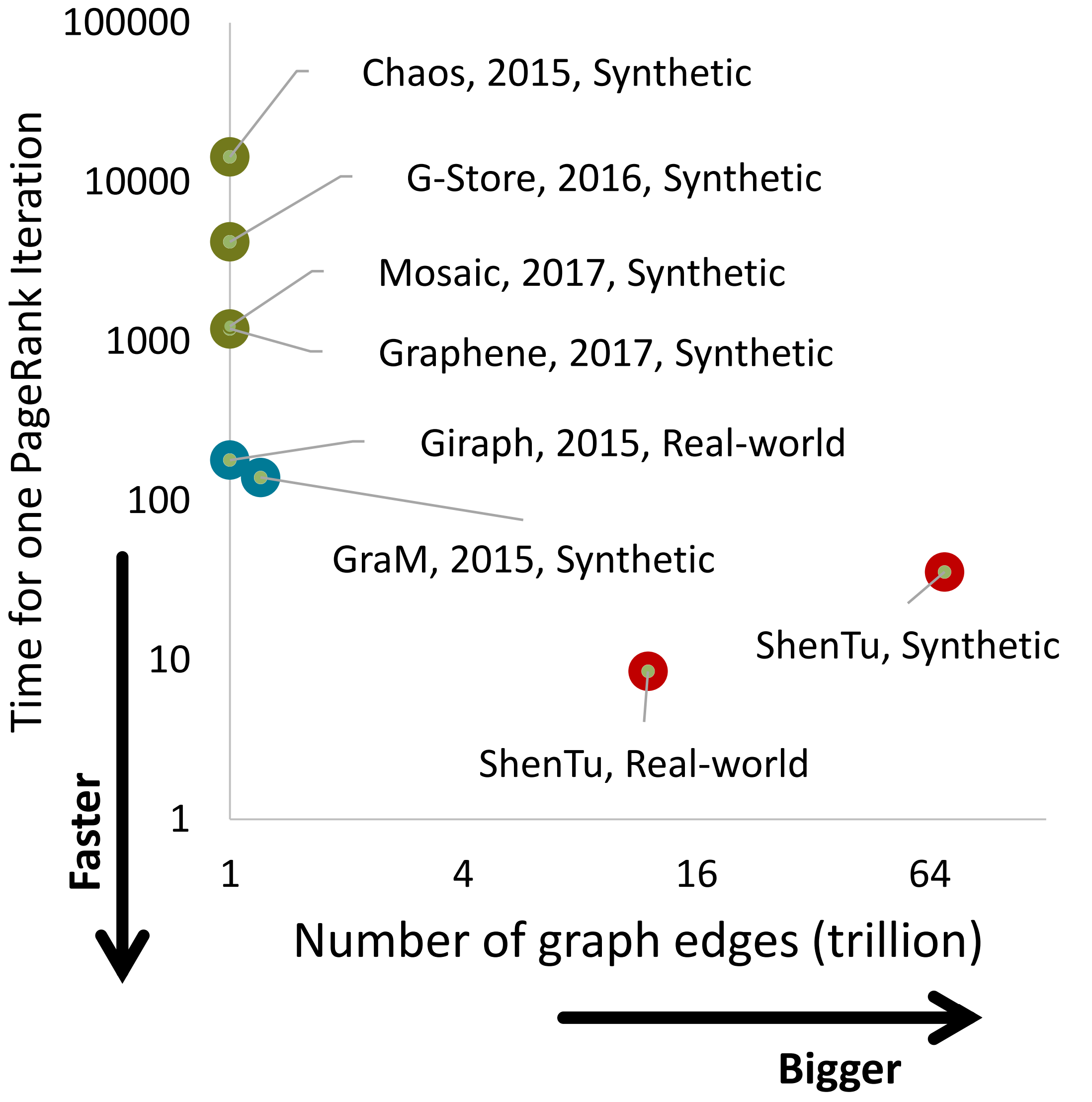
Practice of Extreme-Scale Graph Processing



Memory limited

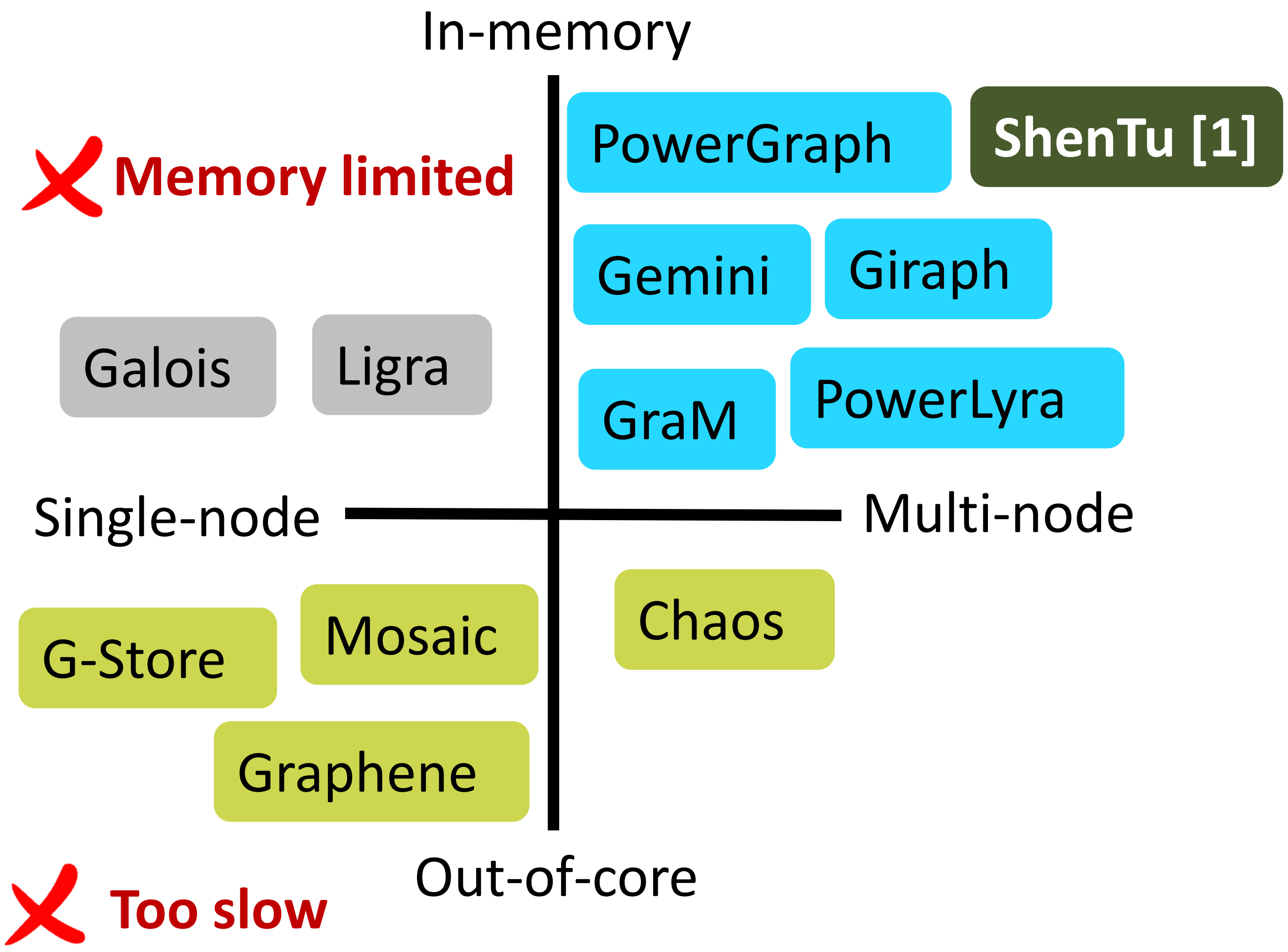
Too slow

(> 20 mins per PageRank iteration on 1-trillion-edge graph)

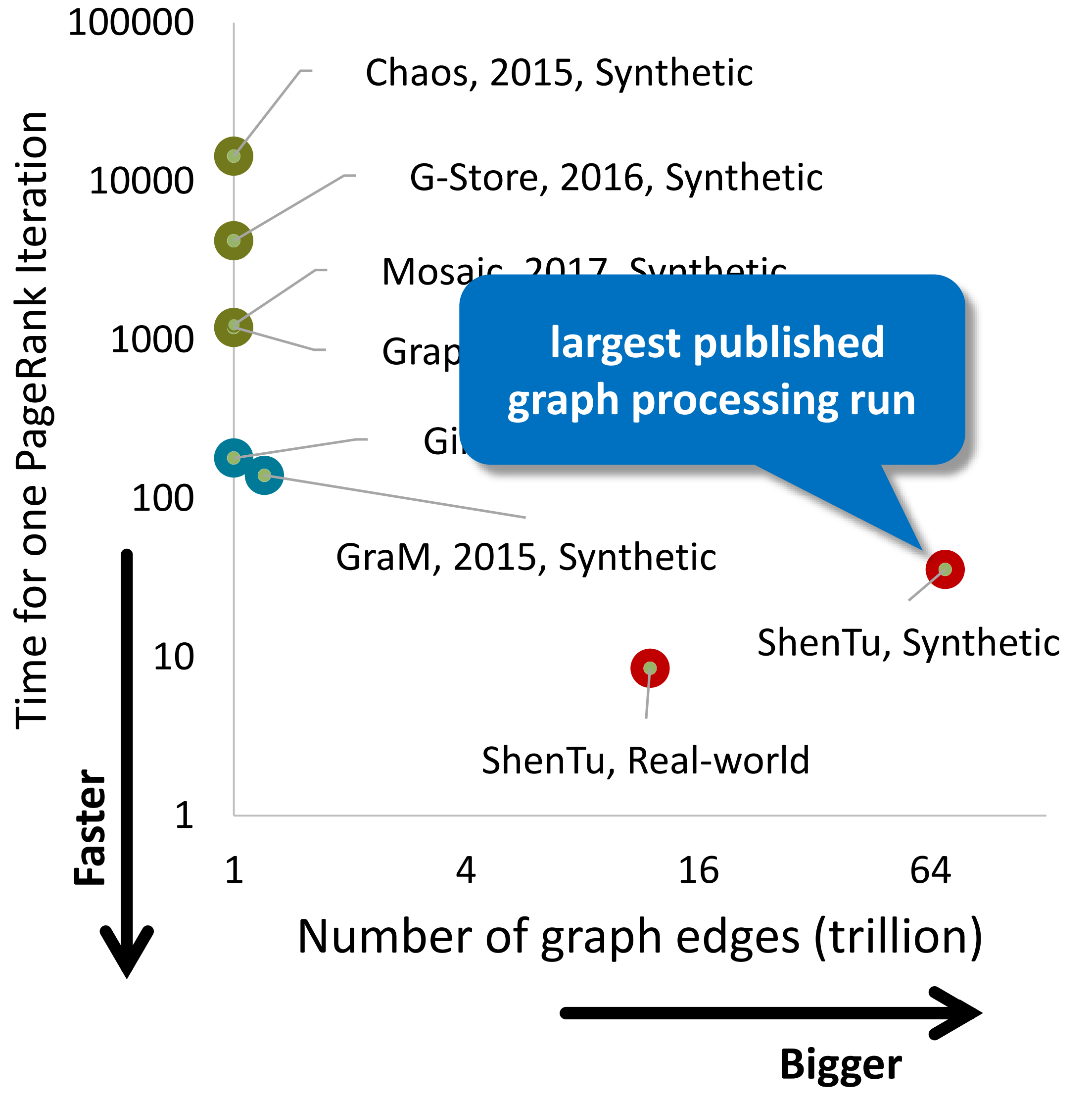


[1] Heng Lin et al.: ShenTu: Processing Multi-Trillion Edge Graphs on Millions of Cores in Seconds, SC18, Gordon Bell Finalist

Practice of Extreme-Scale Graph Processing



(> 20 mins per PageRank iteration on 1-trillion-edge graph)



[1] Heng Lin et al.: ShenTu: Processing Multi-Trillion Edge Graphs on Millions of Cores in Seconds, SC18, Gordon Bell Finalist

How large are extreme-scale graphs today?

How large are extreme-scale graphs today?

Webgraph datasets

Graph	Crawl date	Nodes	Arcs
uk-2014	2014	787 801 471	47 614 527 250
eu-2015	2015	1 070 557 254	91 792 261 600
gsh-2015	2015	988 490 691	33 877 399 152

KONECT graph datasets

en	Wikipedia edits (en)	50,757,442	572,591,272
TW	Twitter (WWW)	41,652,230	1,468,365,182
TF	Twitter (MPI)	52,579,682	1,963,263,821
FR	Friendster	68,349,466	2,586,147,869
UL	UK domain (2007)	105,153,952	3,301,876,564

Web data commons datasets

Granularity	#Nodes	#Arcs
Page	3,563 million	128,736 million
Host	101 million	2,043 million
Pay-Level-Domain	43 million	623 million

How large are extreme-scale graphs today?

Webgraph datasets

Graph	Crawl date	Nodes	Arcs
uk-2014	2014	787 801 471	47 614 527 250
eu-2015	2015	1 070 557 254	91 792 261 600
gsh-2015	2015	988 490 691	33 877 399 152

KONECT graph datasets

en	Wikipedia edits (en)	50,757,442	572,591,272
TW	Twitter (WWW)	41,652,230	1,468,365,182
TF	Twitter (MPI)	52,579,682	1,963,263,821
FR	Friendster	68,349,466	2,586,147,869
UL	UK domain (2007)	105,153,952	3,301,876,564

Web data commons datasets

Granularity	#Nodes	#Arcs
Page	3,563 million	128,736 million
Host	101 million	2,043 million
Pay-Level-Domain	43 million	623 million

How large are extreme-scale graphs today?

Largest Published Graph Computation [1]
Gordon Bell Finalist 2018
ShenTu on Sunway TaihuLight

Webgraph datasets

Graph	Crawl date	Nodes	Arcs
uk-2014	2014	787 801 471	47 614 527 250
eu-2015	2015	1 070 557 254	91 792 261 600
gsh-2015	2015	988 490 691	33 877 399 152

KONECT graph datasets

en	Wikipedia edits (en)	B	C	50,757,442	572,591,272
TW	Twitter (WWW)	D		41,652,230	1,468,365,182
TF	Twitter (MPI)	D		52,579,682	1,963,263,821
FR	Friendster	D		68,349,466	2,586,147,869
UL	UK domain (2007)	U		105,153,952	3,301,876,564

Web data commons datasets

Granularity	#Nodes	#Arcs
Page	3,563 million	128,736 million
Host	101 million	2,043 million
Pay-Level-Domain	43 million	623 million

[1] Heng Lin et al.: ShenTu: Processing Multi-Trillion Edge Graphs on Millions of Cores in Seconds, SC18, Gordon Bell Finalist

How large are extreme-scale graphs today?

Largest Published Graph Computation [1]
Gordon Bell Finalist 2018
ShenTu on Sunway TaihuLight

Sogou 搜狗 271 billion vertices, 12 trillion edges

Kronecker graph: 4.4 trillion vertices, 70 trillion edges

Webgraph datasets

Graph	Crawl date	Nodes	Arcs
uk-2014	2014	787 801 471	47 614 527 250
eu-2015	2015	1 070 557 254	91 792 261 600
gsh-2015	2015	988 490 691	33 877 399 152

KONECT graph datasets

en	Wikipedia edits (en)	50,757,442	572,591,272
TW	Twitter (WWW)	41,652,230	1,468,365,182
TF	Twitter (MPI)	52,579,682	1,963,263,821
FR	Friendster	68,349,466	2,586,147,869
UL	UK domain (2007)	105,153,952	3,301,876,564

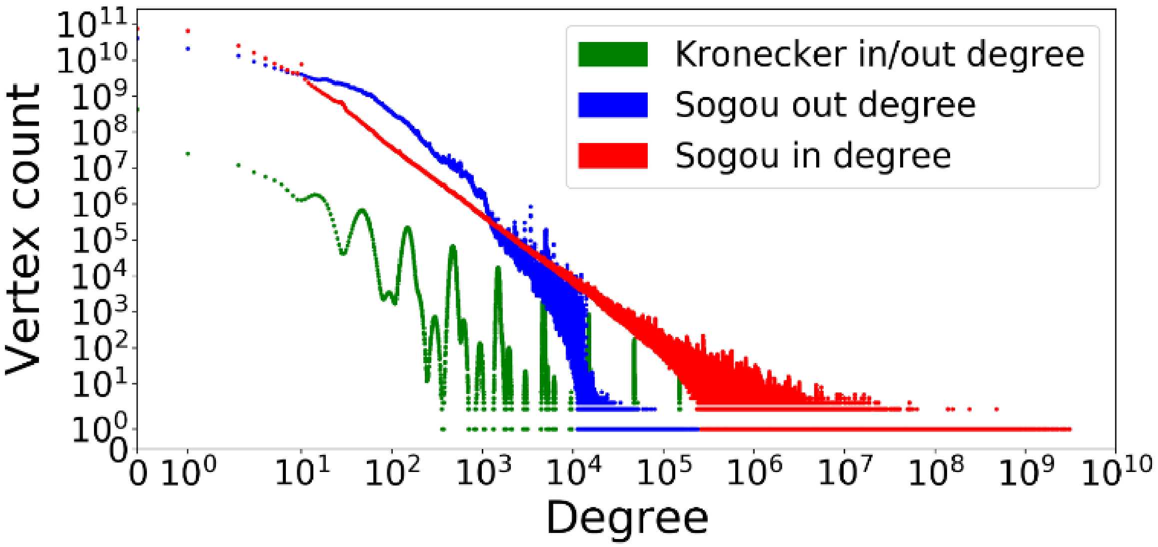
Web data commons datasets

Granularity	#Nodes	#Arcs
Page	3,563 million	128,736 million
Host	101 million	2,043 million
Pay-Level-Domain	43 million	623 million

[1] Heng Lin et al.: ShenTu: Processing Multi-Trillion Edge Graphs on Millions of Cores in Seconds, SC18, Gordon Bell Finalist

How large are extreme-scale graphs today?

Largest Published Graph Computation [1]
Gordon Bell Finalist 2018
ShenTu on Sunway TaihuLight



Sogou 搜狗 271 billion vertices, 12 trillion edges

Kronecker graph: 4.4 trillion vertices, 70 trillion edges

Webgraph datasets

Graph	Crawl date	Nodes	Arcs
uk-2014	2014	787 801 471	47 614 527 250
eu-2015	2015	1 070 557 254	91 792 261 600
gsh-2015	2015	988 490 691	33 877 399 152

KONECT graph datasets

en	Wikipedia edits (en)	B — C	50,757,442	572,591,272
TW	Twitter (WWW)	D —	41,652,230	1,468,365,182
TF	Twitter (MPI)	D —	52,579,682	1,963,263,821
FR	Friendster	D —	68,349,466	2,586,147,869
UL	UK domain (2007)	U —	105,153,952	3,301,876,564

Web data commons datasets

Granularity	#Nodes	#Arcs
Page	3,563 million	128,736 million
Host	101 million	2,043 million
Pay-Level-Domain	43 million	623 million

[1] Heng Lin et al.: ShenTu: Processing Multi-Trillion Edge Graphs on Millions of Cores in Seconds, SC18, Gordon Bell Finalist

Sunway TaihuLight

Heng Lin et al.: ShenTu: Processing Multi-Trillion Edge Graphs on Millions of Cores in Seconds, SC18, Gordon Bell Finalist

Sunway TaihuLight

TaihuLight Top500 ranking: #3 (2018 Nov), #1 (2016, 2017)

Heng Lin et al.: ShenTu: Processing Multi-Trillion Edge Graphs on Millions of Cores in Seconds, SC18, Gordon Bell Finalist

Sunway TaihuLight

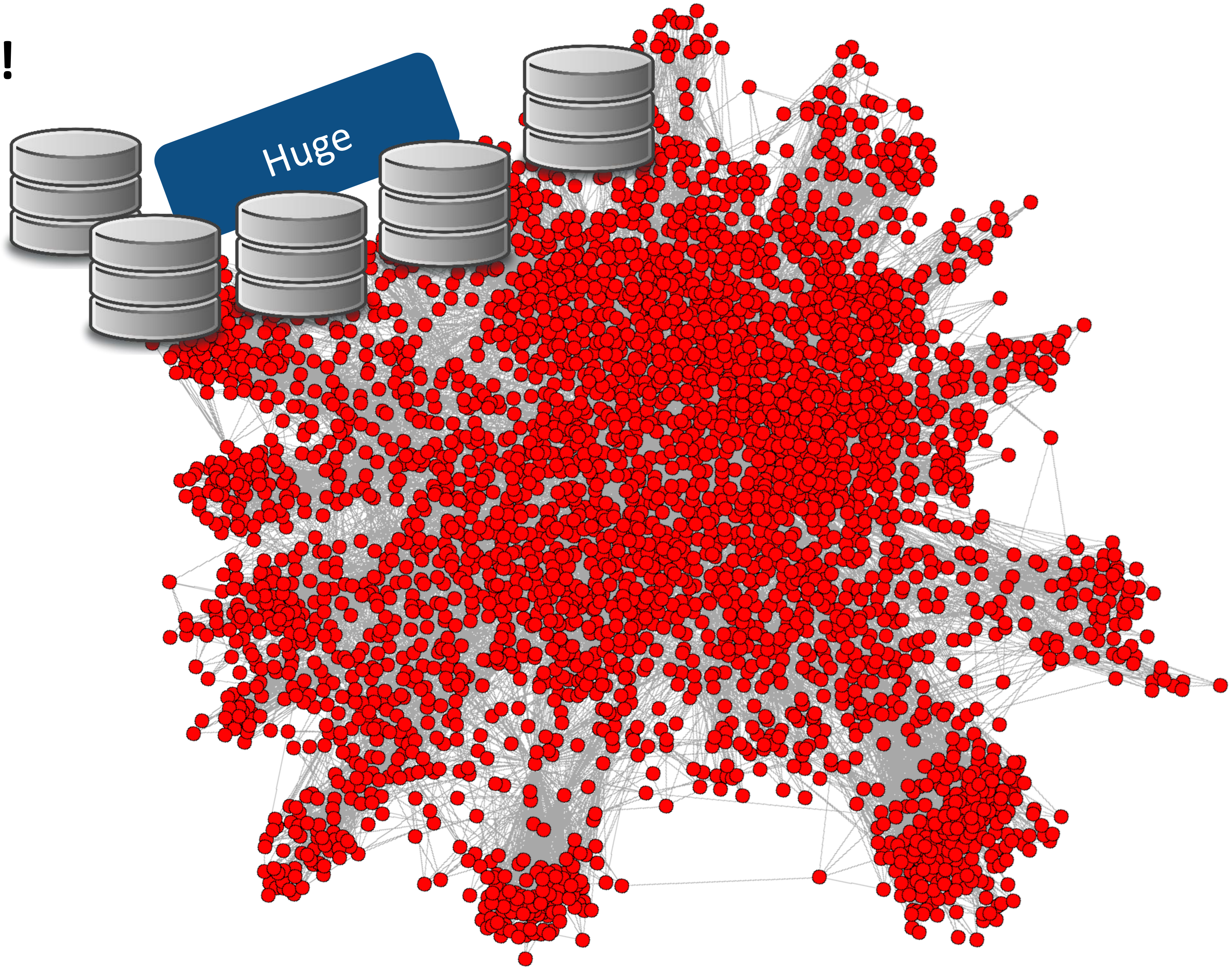
Nearly full system run (~41,000 nodes, ~1 million cores)

- PageRank iteration on 12 trillion edges in 8.5s (1.4 TPEPS)
- On 70 trillion edges, nearly 2 TPEPS for PageRank and WCC
  - 774 GPEPS for BFS

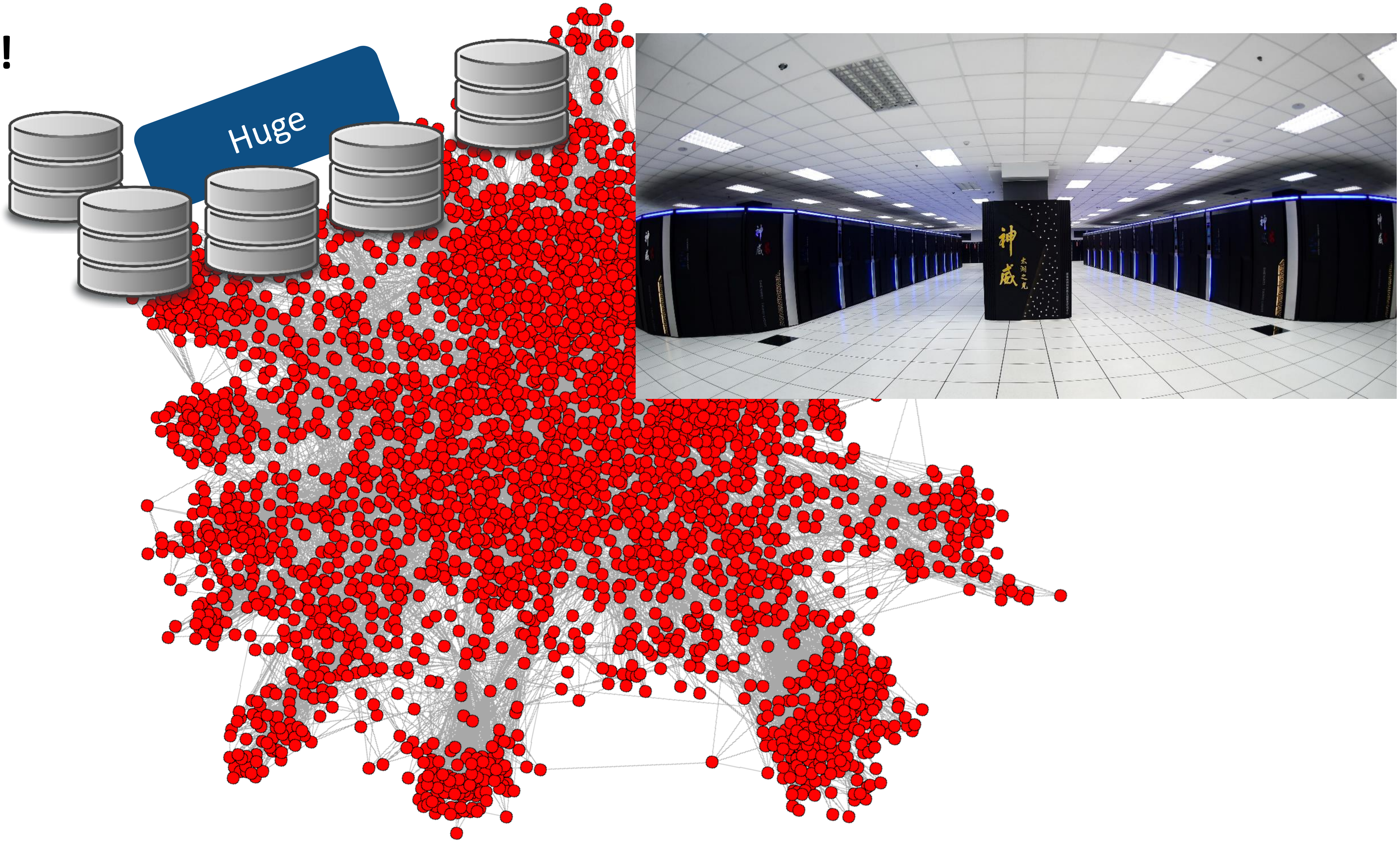
  (PEPS = processes edges per second as opposed to TEPS)

Heng Lin et al.: ShenTu: Processing Multi-Trillion Edge Graphs on Millions of Cores in Seconds, SC18, Gordon Bell Finalist

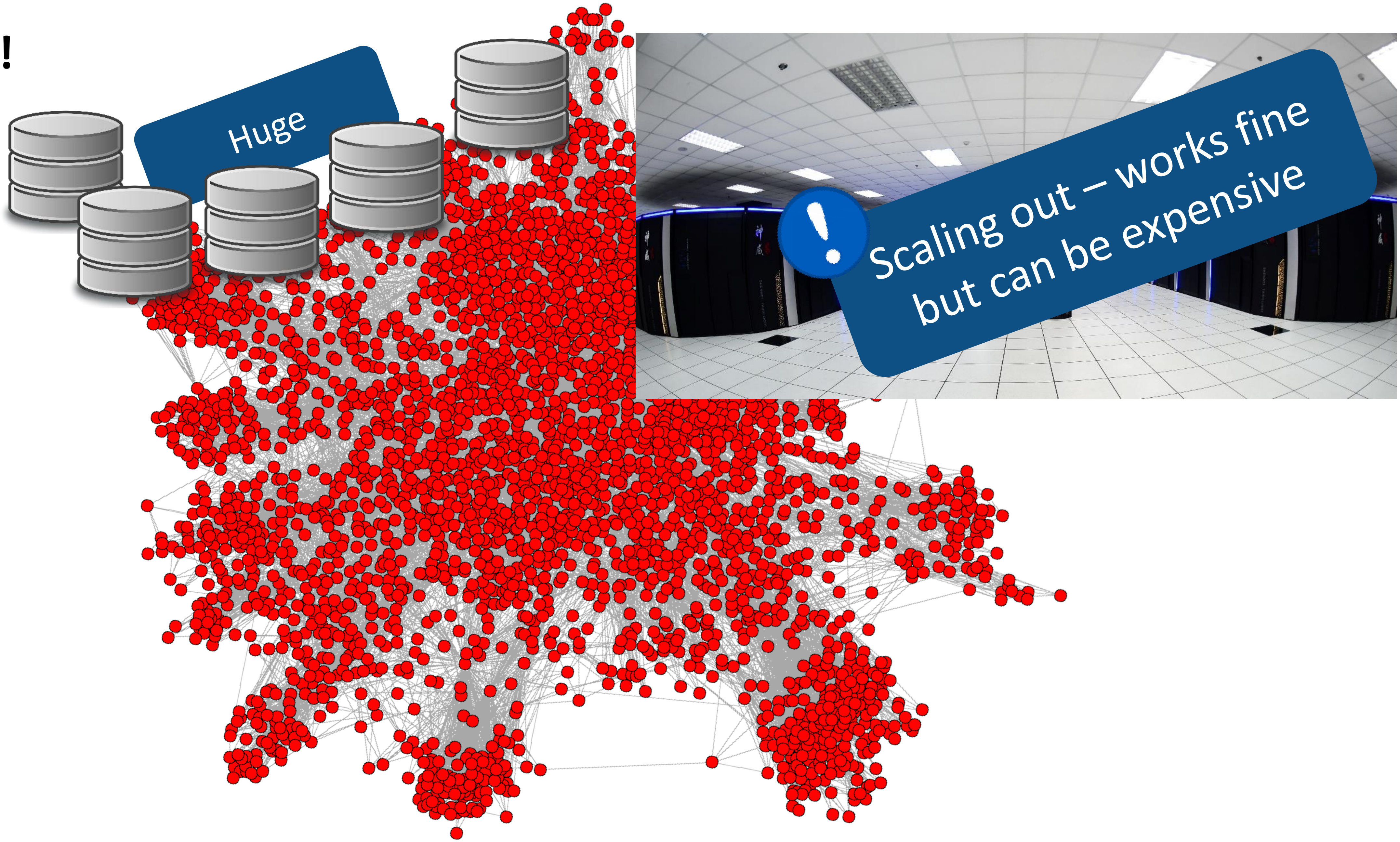
Problems!



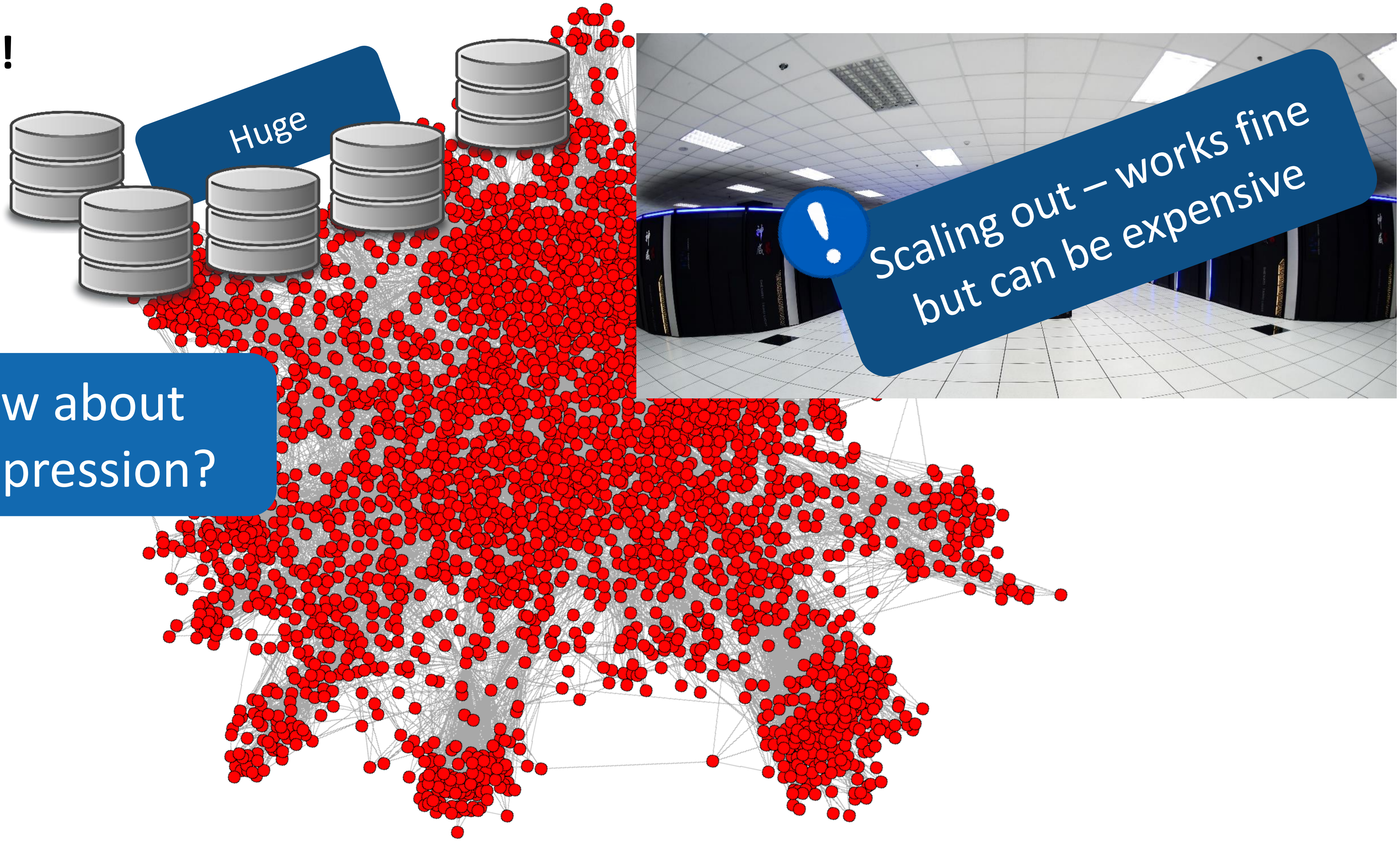
Problems!



Problems!



Problems!



Scaling out – works fine but can be expensive

How about compression?

Problems!

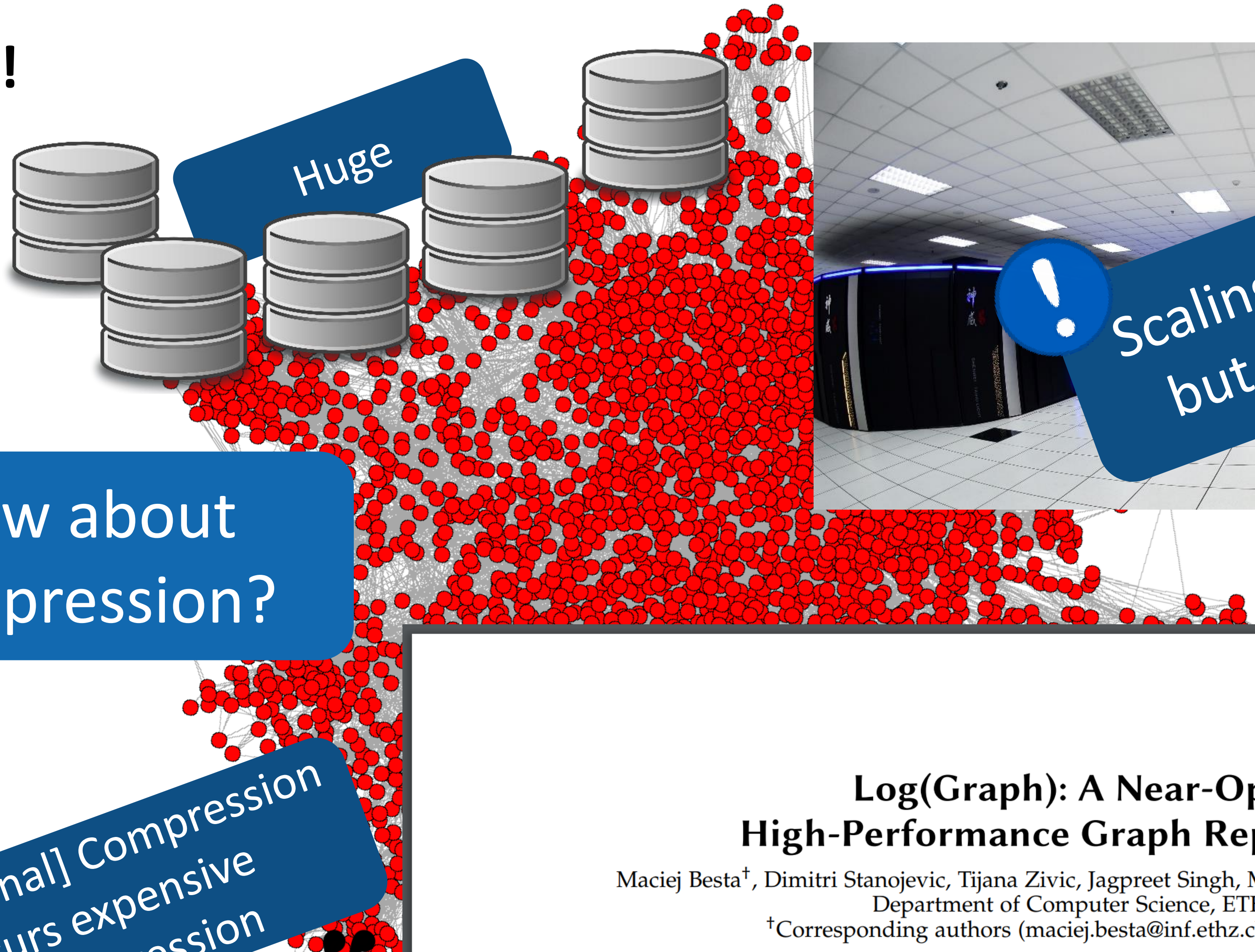
Huge

Scaling out – works fine but can be expensive

How about compression?

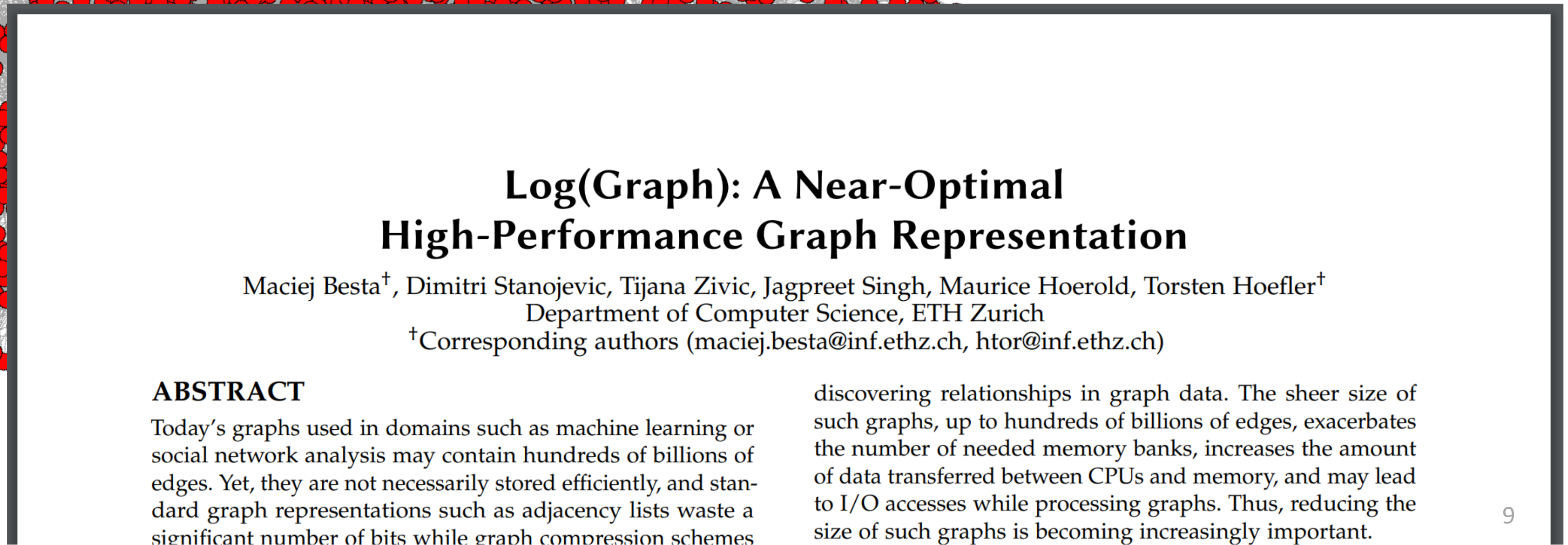
[Traditional] Compression incurs expensive decompression

Problems!



How about compression?

[Traditional] Compression incurs expensive decompression



Log(Graph): A Near-Optimal High-Performance Graph Representation

Maciej Besta[†], Dimitri Stanojevic, Tijana Zivic, Jagpreet Singh, Maurice Hoerold, Torsten Hoefler[†]
Department of Computer Science, ETH Zurich
[†]Corresponding authors (maciej.besta@inf.ethz.ch, htor@inf.ethz.ch)

ABSTRACT

Today's graphs used in domains such as machine learning or social network analysis may contain hundreds of billions of edges. Yet, they are not necessarily stored efficiently, and standard graph representations such as adjacency lists waste a significant number of bits while graph compression schemes discovering relationships in graph data. The sheer size of such graphs, up to hundreds of billions of edges, exacerbates the number of needed memory banks, increases the amount of data transferred between CPUs and memory, and may lead to I/O accesses while processing graphs. Thus, reducing the size of such graphs is becoming increasingly important.

Problems!

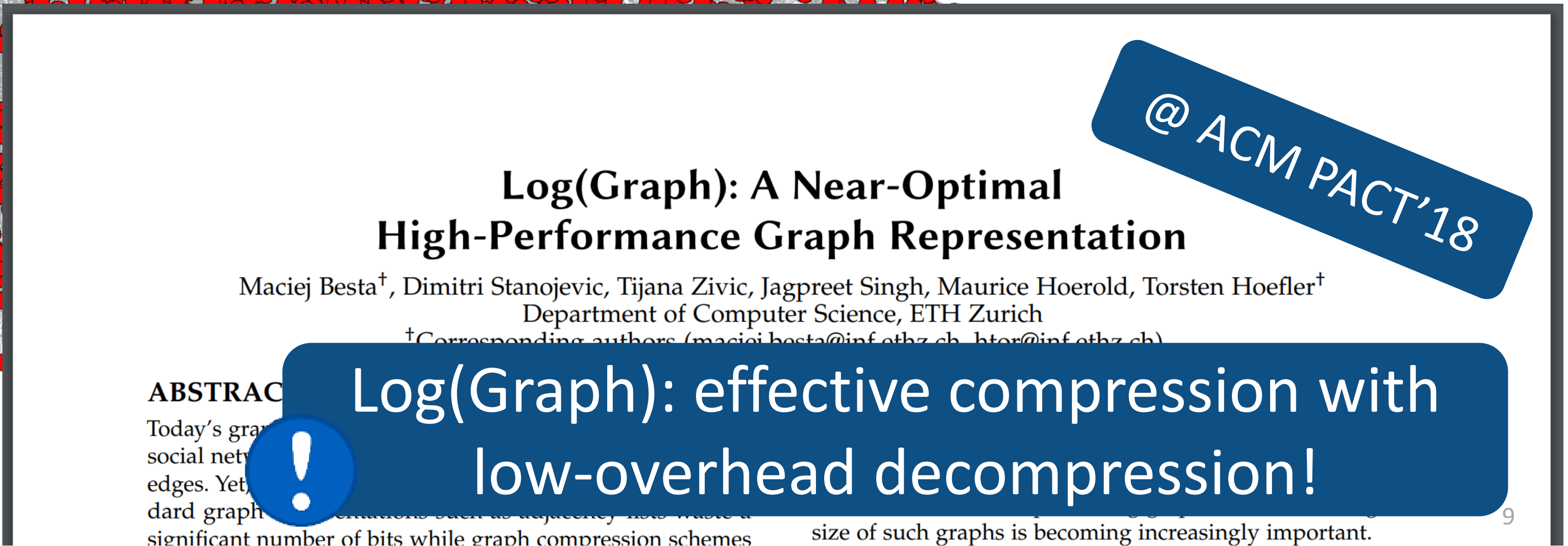
Huge

Scaling out – works fine but can be expensive

How about compression?

[Traditional] Compression incurs expensive decompression

@ ACM PACT'18



Log(Graph): A Near-Optimal High-Performance Graph Representation

Maciej Besta†, Dimitri Stanojevic, Tijana Zivic, Jagpreet Singh, Maurice Hoerold, Torsten Hoefler†
Department of Computer Science, ETH Zurich
†Corresponding authors (maciej.besta@inf.ethz.ch, htor@inf.ethz.ch)

ABSTRAC
Today's gra
social net
edges. Yet
dard graph
significant number of bits while graph compression schemes
size of such graphs is becoming increasingly important.

Log(Graph): effective compression with low-overhead decompression!

M. Besta et al.: “Log(Graph): A Near-Optimal High-Performance Graph Representation”, PACT’18

What is **the lowest storage** we can (hope to) use to store a graph?

M. Besta et al.: "Log(Graph): A Near-Optimal High-Performance Graph Representation", PACT'18

What is **the lowest storage** we can (hope to) use to store a graph?

The storage **lower bound** Ω

M. Besta et al.: "Log(Graph): A Near-Optimal High-Performance Graph Representation", PACT'18

What is **the lowest storage** we can (hope to) use to store a graph?

The storage **lower bound** Ω

Which one? ☺

M. Besta et al.: "Log(Graph): A Near-Optimal High-Performance Graph Representation", PACT'18

What is **the lowest storage** we can (hope to) use to store a graph?

The storage **lower bound** Ω

Which one? ☺

Shannon's approach
logarithmic
(one needs at least log|S| bits to store an object from an arbitrary set S)

M. Besta et al.: "Log(Graph): A Near-Optimal High-Performance Graph Representation", PACT'18

What is **the lowest storage** we can (hope to) use to store a graph?

The storage **lower bound** Ω

Which one? ☺

Shannon's approach
logarithmic
(one needs at least log|S| bits to store an object from an arbitrary set S)

$S = \{x_1, x_2, x_3, \dots\}$

$x_1 \rightarrow 0 \dots 01$
$x_2 \rightarrow 0 \dots 10$
$x_3 \rightarrow 0 \dots 11$
…

What is **the lowest storage** we can (hope to) use to store a graph?

The storage **lower bound** Ω

Which one? ☺

Shannon's approach **logarithmic** (one needs at least log|S| bits to store an object from an arbitrary set S)

$S = \{x_1, x_2, x_3, \ldots\}$

$x_1 \rightarrow 0 \ldots 01$
$x_2 \rightarrow 0 \ldots 10$
$x_3 \rightarrow 0 \ldots 11$
$\ldots$

Key idea

M. Besta et al.: "Log(Graph): A Near-Optimal High-Performance Graph Representation", PACT'18

What is **the lowest storage** we can (hope to) use to store a graph?

The storage **lower bound** Ω

Which one? ☺

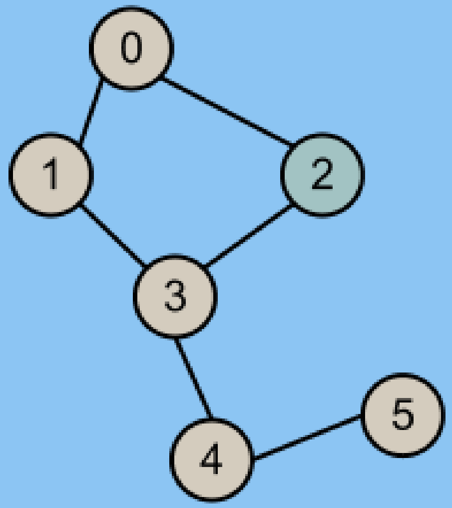
Shannon's approach **logarithmic** (one needs at least log|S| bits to store an object from an arbitrary set S)

$S = \{x_1, x_2, x_3, \dots\}$

$x_1 \rightarrow 0 \dots 01$
$x_2 \rightarrow 0 \dots 10$
$x_3 \rightarrow 0 \dots 11$
$\dots$

Key idea

Encode different parts of a graph representation using (logarithmic) **storage lower bounds**



M. Besta et al.: "Log(Graph): A Near-Optimal High-Performance Graph Representation", PACT'18

What is **the lowest storage** we can (hope to) use to store a graph?

The storage **lower bound** Ω

Which one? ☺

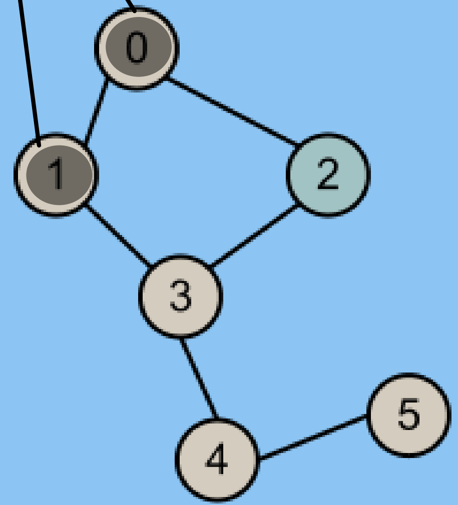
Shannon's approach **logarithmic** (one needs at least log|S| bits to store an object from an arbitrary set S)

$$S = \{x_1, x_2, x_3, \dots\} \quad \begin{array}{l} x_1 \rightarrow 0 \dots 01 \\ x_2 \rightarrow 0 \dots 10 \\ x_3 \rightarrow 0 \dots 11 \\ \quad\dots \end{array}$$

Key idea

Encode different parts of a graph representation using (logarithmic) **storage lower bounds**

Vertex labels



M. Besta et al.: "Log(Graph): A Near-Optimal High-Performance Graph Representation", PACT'18

What is **the lowest storage** we can (hope to) use to store a graph?

The storage **lower bound** Ω

Which one? ☺

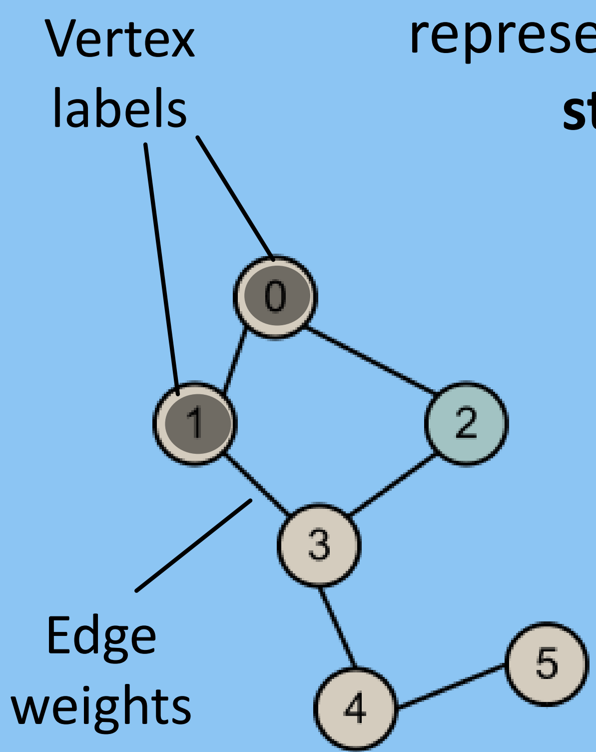
Shannon's approach **logarithmic** (one needs at least log|S| bits to store an object from an arbitrary set S)

$S = \{x_1, x_2, x_3, \dots\}$

$x_1 \rightarrow 0 \dots 01$
$x_2 \rightarrow 0 \dots 10$
$x_3 \rightarrow 0 \dots 11$
$\dots$

Key idea

Encode different parts of a graph representation using (logarithmic) **storage lower bounds**



M. Besta et al.: "Log(Graph): A Near-Optimal High-Performance Graph Representation", PACT'18

What is **the lowest storage** we can (hope to) use to store a graph?

The storage **lower bound** Ω

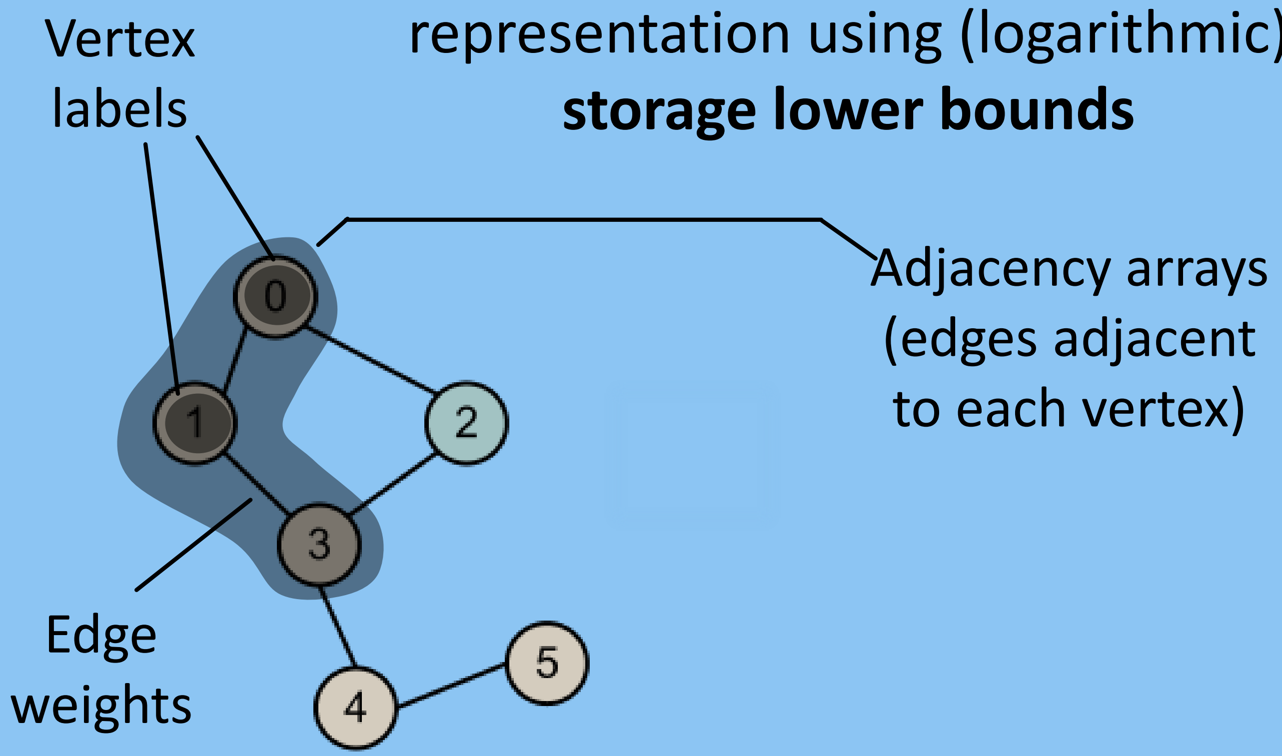
Which one? ☺

Shannon's approach **logarithmic** (one needs at least log|S| bits to store an object from an arbitrary set S)

$$S = \{x_1, x_2, x_3, \ldots\} \quad \begin{array}{l} x_1 \rightarrow 0\ldots01 \\ x_2 \rightarrow 0\ldots10 \\ x_3 \rightarrow 0\ldots11 \\ \ldots \end{array}$$

Key idea

Encode different parts of a graph representation using (logarithmic) **storage lower bounds**



What is **the lowest storage** we can (hope to) use to store a graph?

$$S = \{x_1, x_2, x_3, \dots\} \quad \begin{array}{l} x_1 \rightarrow 0 \dots 01 \\ x_2 \rightarrow 0 \dots 10 \\ x_3 \rightarrow 0 \dots 11 \\ \quad \dots \end{array}$$

The storage **lower bound** Ω

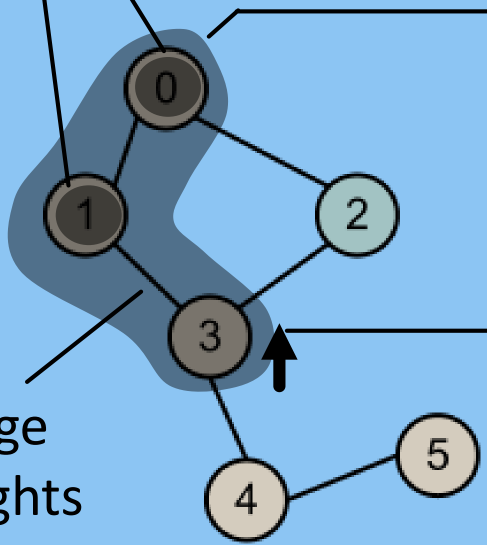
Which one? ☺

Shannon's approach **logarithmic** (one needs at least log|S| bits to store an object from an arbitrary set S)

Key idea

Encode different parts of a graph representation using (logarithmic) **storage lower bounds**

Vertex labels



Adjacency arrays (edges adjacent to each vertex)

Edge weights

Offsets (locations) of adj. arrays

What is **the lowest storage** we can (hope to) use to store a graph?

$$S = \{x_1, x_2, x_3, \dots\} \quad \begin{aligned} x_1 &\rightarrow 0 \dots 01 \\ x_2 &\rightarrow 0 \dots 10 \\ x_3 &\rightarrow 0 \dots 11 \\ &\dots \end{aligned}$$

The storage **lower bound** Ω

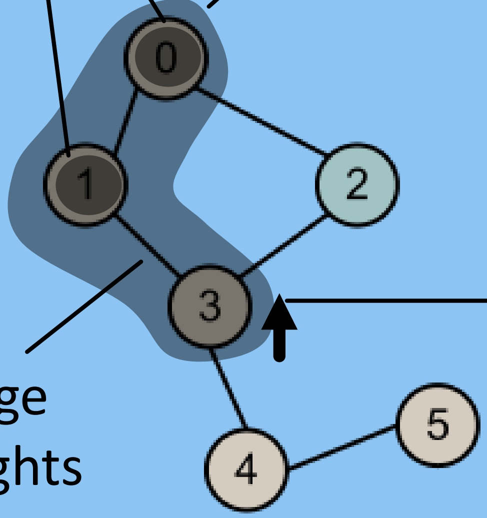
Which one? ☺

Shannon's approach **logarithmic** (one needs at least log|S| bits to store an object from an arbitrary set S)

Key idea

Encode different parts of a graph representation using (logarithmic) **storage lower bounds**

Log (Vertex labels **)**



Adjacency arrays (edges adjacent to each vertex)

Offsets (locations) of adj. arrays

Edge weights

M. Besta et al.: "Log(Graph): A Near-Optimal High-Performance Graph Representation", PACT'18

What is **the lowest storage** we can (hope to) use to store a graph?

$$S = \{x_1, x_2, x_3, \dots\} \quad \begin{array}{l} x_1 \rightarrow 0 \dots 01 \\ x_2 \rightarrow 0 \dots 10 \\ x_3 \rightarrow 0 \dots 11 \\ \dots \end{array}$$

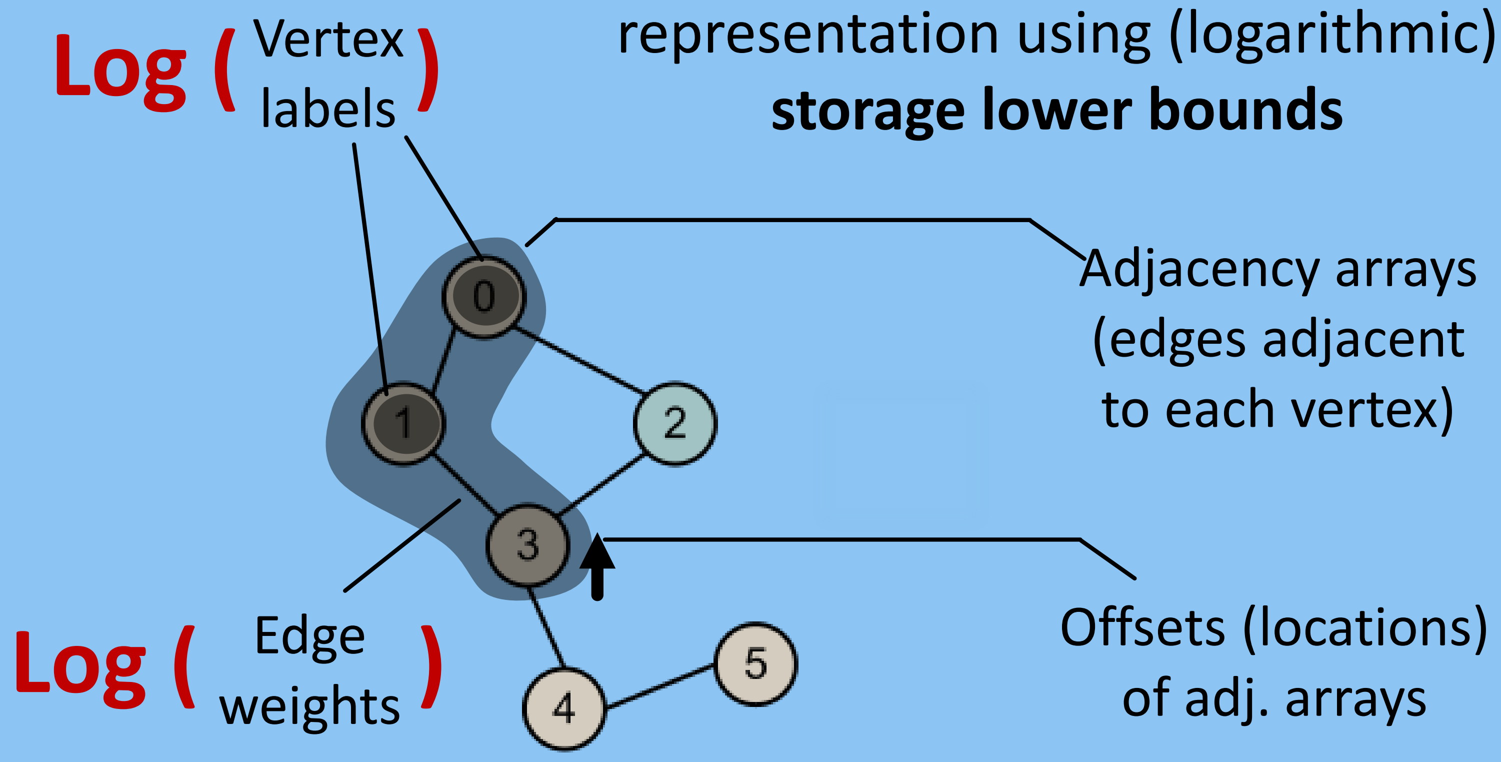
The storage **lower bound** Ω

Which one? ☺

Shannon's approach **logarithmic** (one needs at least log|S| bits to store an object from an arbitrary set S)

Key idea

Encode different parts of a graph representation using (logarithmic) **storage lower bounds**



M. Besta et al.: "Log(Graph): A Near-Optimal High-Performance Graph Representation", PACT'18

What is **the lowest storage** we can (hope to) use to store a graph?

The storage **lower bound** Ω

Which one? ☺

Shannon's approach **logarithmic** (one needs at least log|S| bits to store an object from an arbitrary set S)

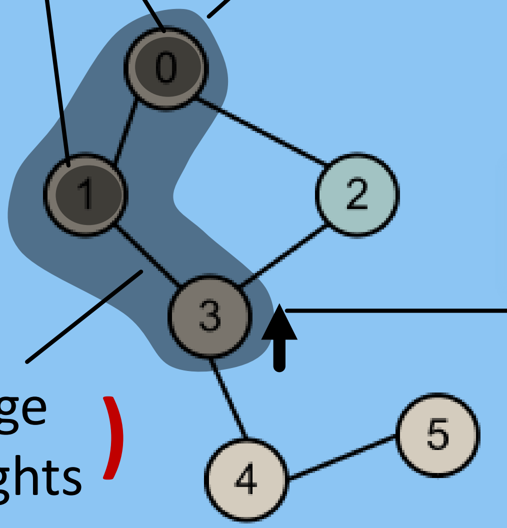
$$S = \{x_1, x_2, x_3, \dots\}$$

$x_1 \rightarrow 0 \dots 01$
$x_2 \rightarrow 0 \dots 10$
$x_3 \rightarrow 0 \dots 11$
...

Key idea

Encode different parts of a graph representation using (logarithmic) **storage lower bounds**

Log (Vertex labels **)**



Log (Adjacency arrays (edges adjacent to each vertex) **)**

Log (Edge weights **)**

Offsets (locations) of adj. arrays

M. Besta et al.: "Log(Graph): A Near-Optimal High-Performance Graph Representation", PACT'18

What is **the lowest storage** we can (hope to) use to store a graph?

The storage **lower bound** Ω

Which one? ☺

Shannon's approach **logarithmic** (one needs at least log|S| bits to store an object from an arbitrary set S)

$S = \{x_1, x_2, x_3, \ldots\}$

$x_1 \rightarrow 0 \ldots 01$
$x_2 \rightarrow 0 \ldots 10$
$x_3 \rightarrow 0 \ldots 11$
$\ldots$

Key idea

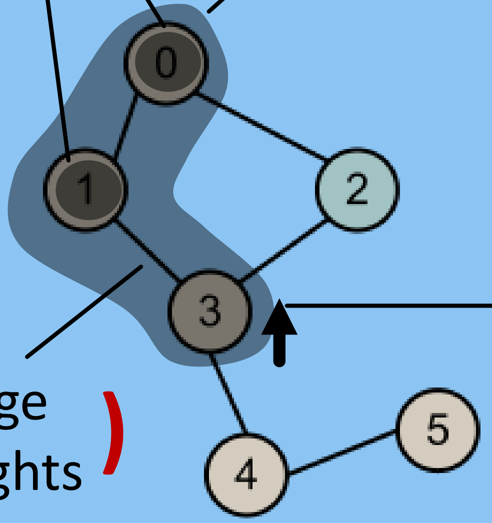
Encode different parts of a graph representation using (logarithmic) **storage lower bounds**

Log (Vertex labels **)**

Log (Adjacency arrays (edges adjacent to each vertex) **)**

Log (Edge weights **)**

Log (Offsets (locations) of adj. arrays **)**



ADJACENCY ARRAY
GRAPH REPRESENTATION

M. Besta et al.: “Log(Graph): A Near-Optimal High-Performance Graph Representation”, PACT’18

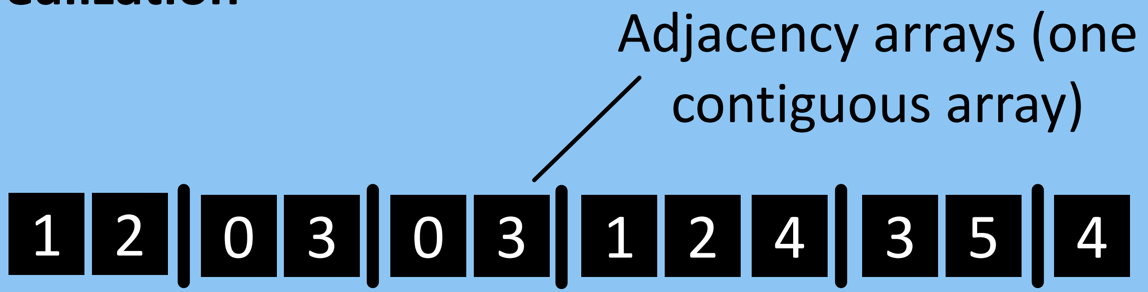
ADJACENCY ARRAY GRAPH REPRESENTATION

Physical realization

M. Besta et al.: “Log(Graph): A Near-Optimal High-Performance Graph Representation”, PACT’18

Adjacency Array
Graph Representation

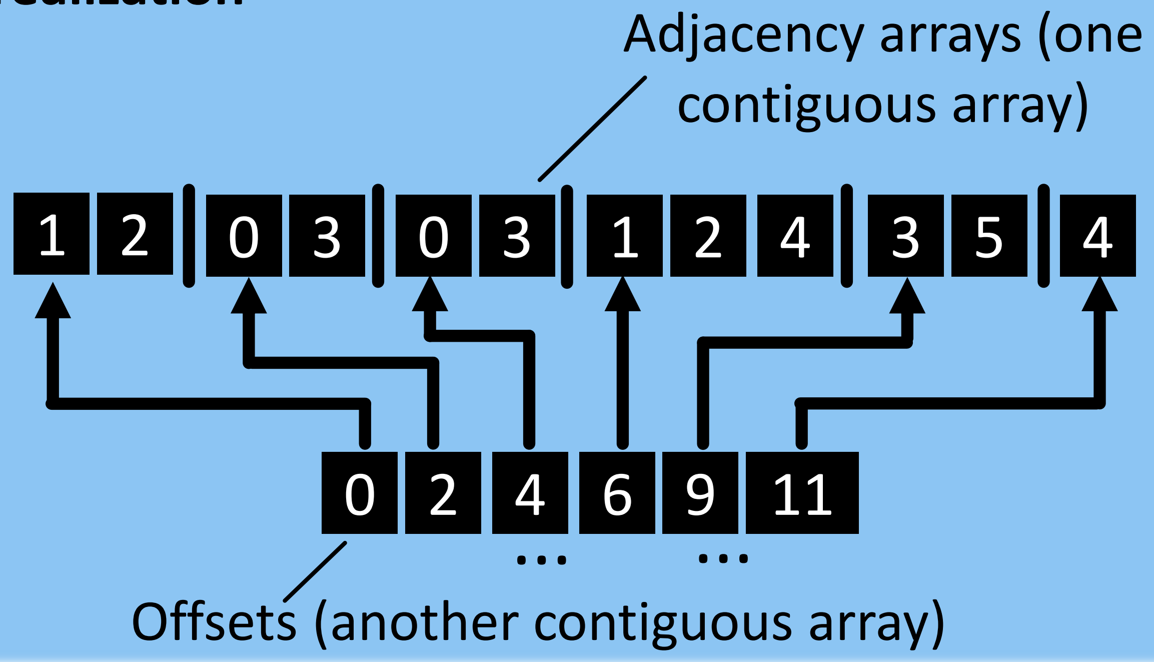
Physical realization



M. Besta et al.: "Log(Graph): A Near-Optimal High-Performance Graph Representation", PACT'18

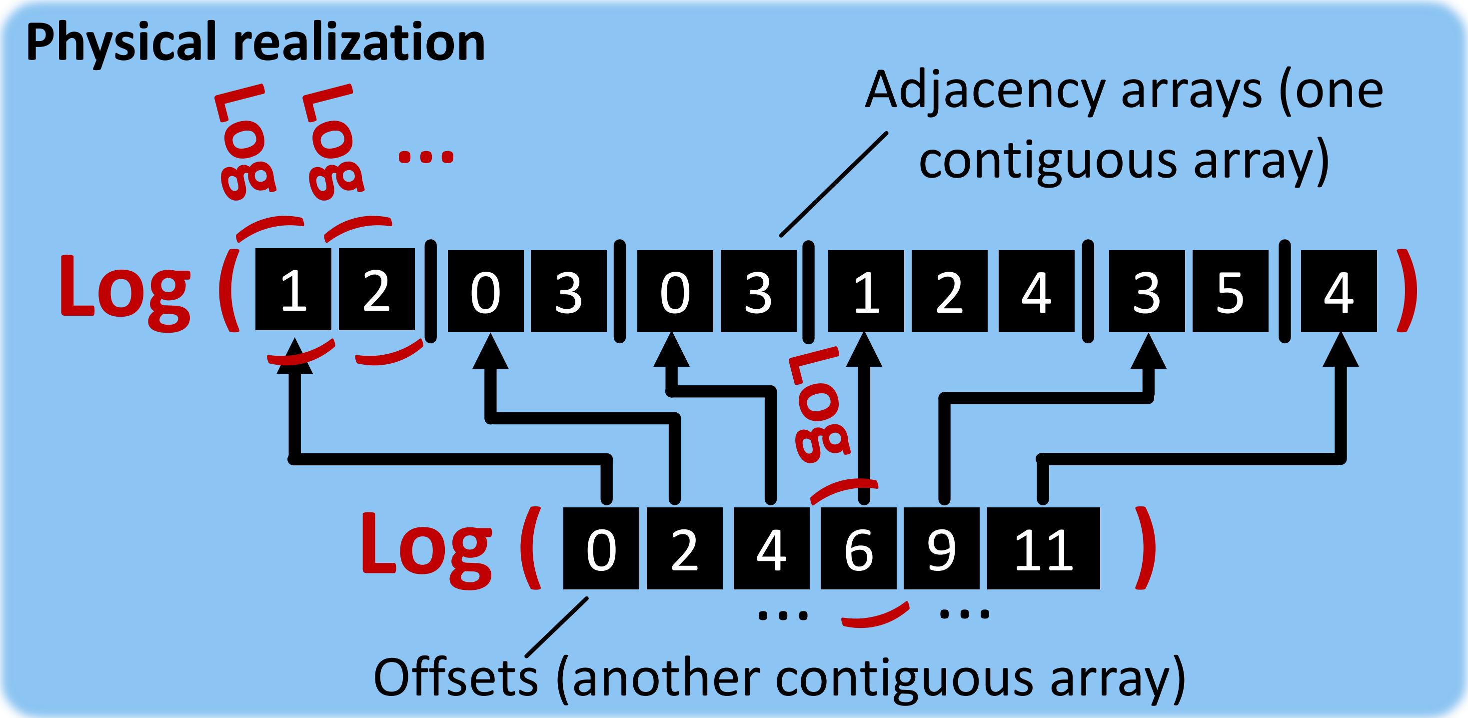
Adjacency Array
Graph Representation

Physical realization



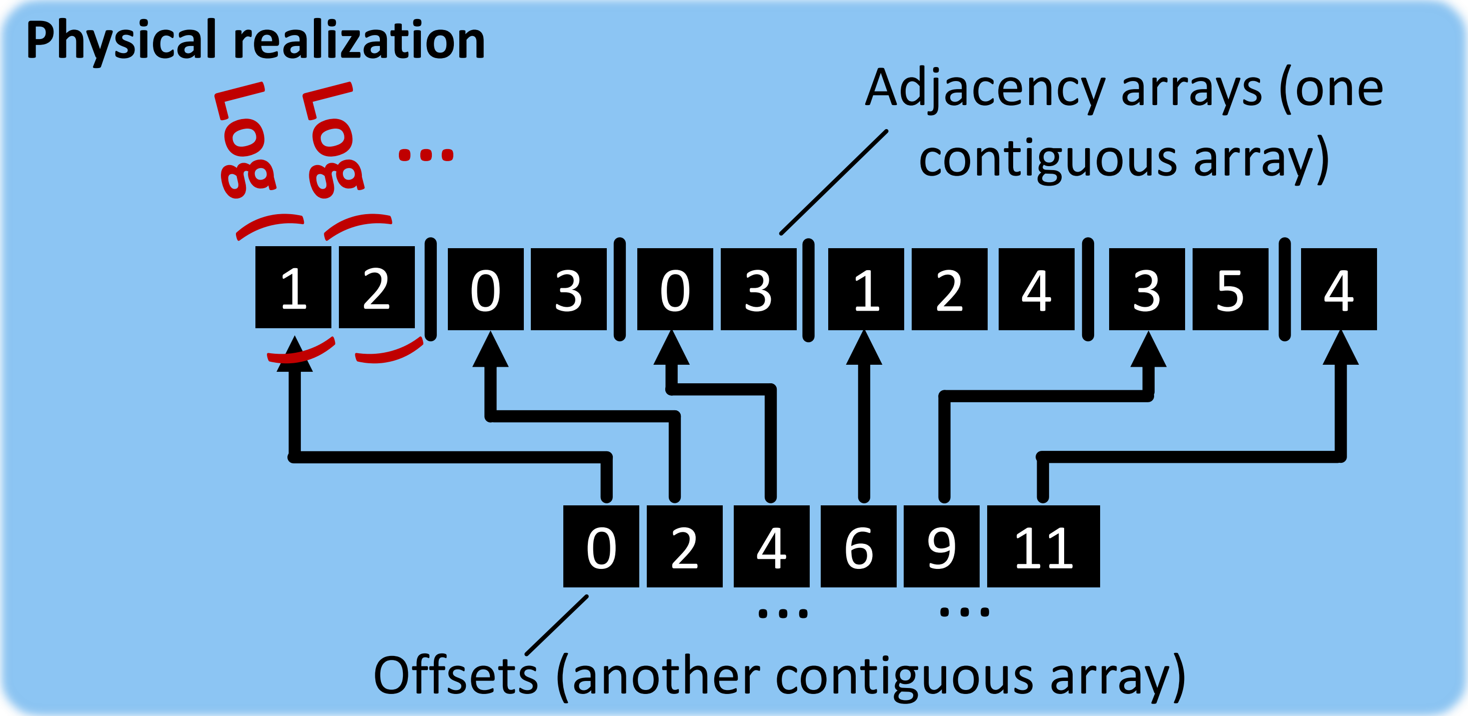
M. Besta et al.: “Log(Graph): A Near-Optimal High-Performance Graph Representation”, PACT’18

Adjacency Array
Graph Representation



M. Besta et al.: “Log(Graph): A Near-Optimal High-Performance Graph Representation”, PACT’18

Adjacency Array
Graph Representation

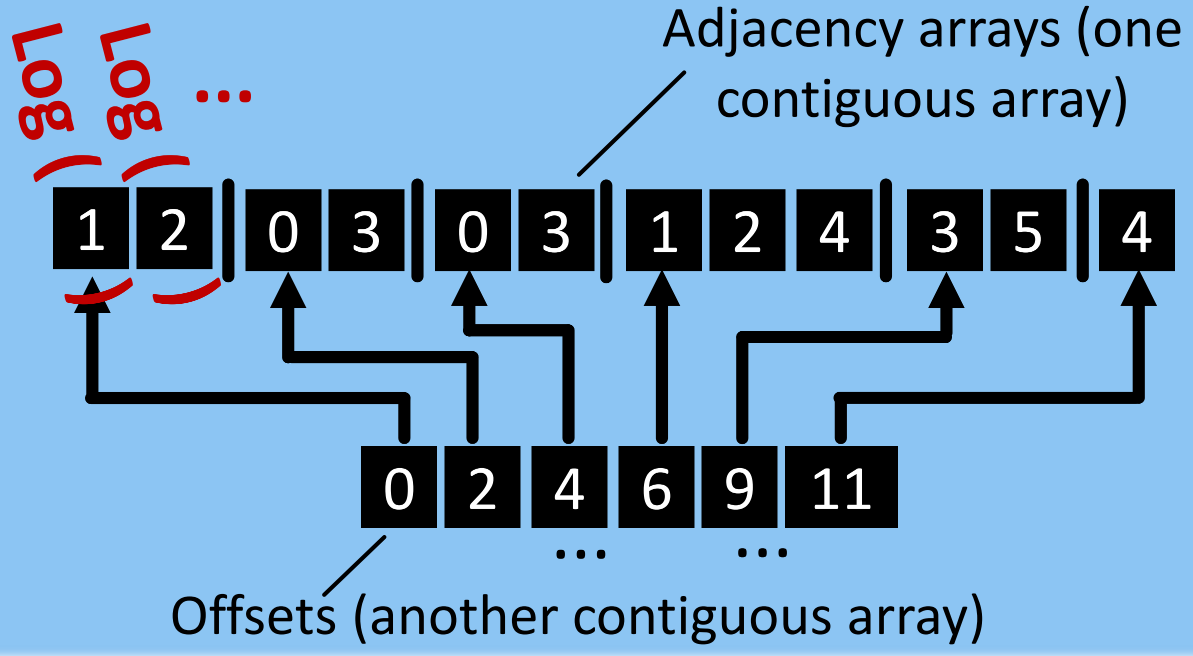


M. Besta et al.: “Log(Graph): A Near-Optimal High-Performance Graph Representation”, PACT’18

Adjacency Array Graph Representation

1 **Log (Vertex labels), Log (Edge weights)**

Physical realization

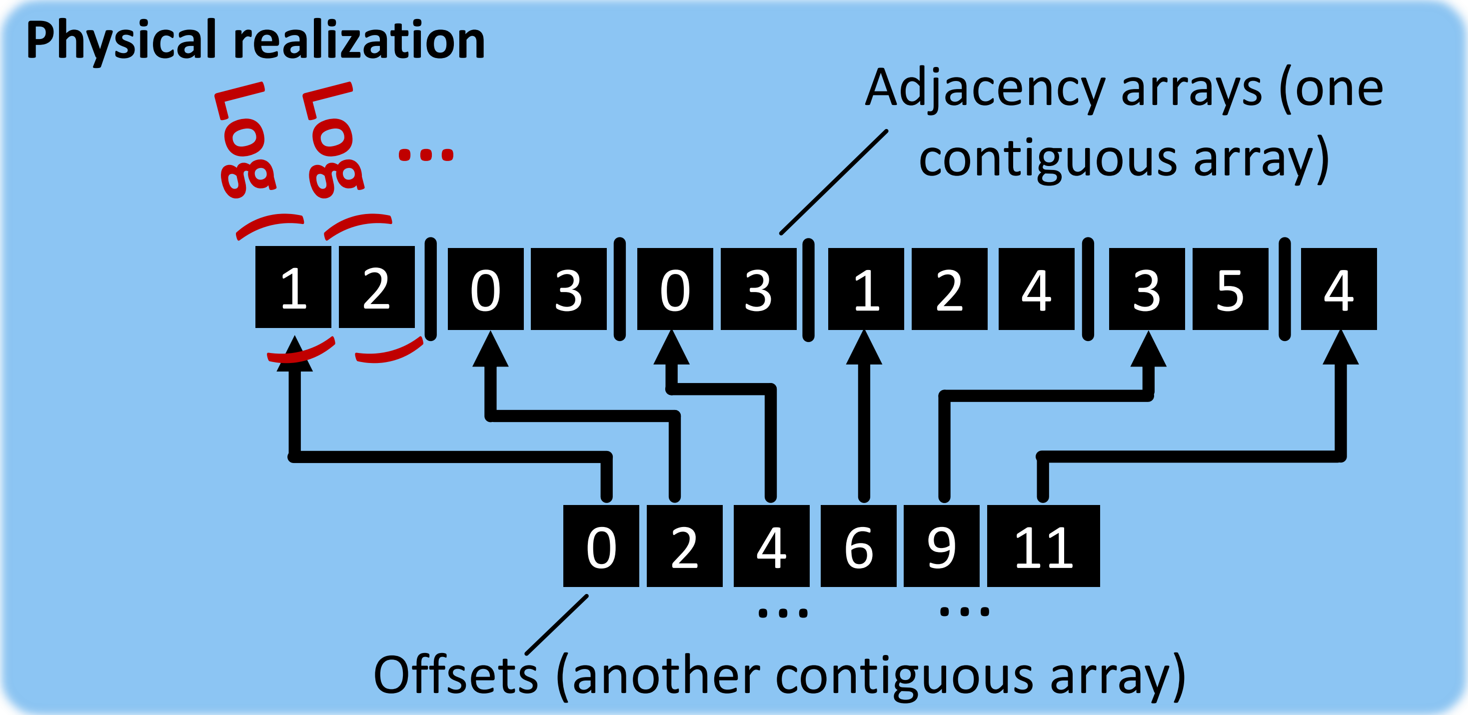


M. Besta et al.: “Log(Graph): A Near-Optimal High-Performance Graph Representation”, PACT’18

Adjacency Array Graph Representation

1 Log (Vertex labels), Log (Edge weights)

Global bound $\lceil \log n \rceil$

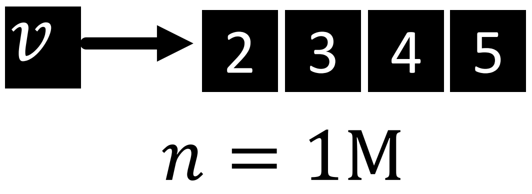


M. Besta et al.: "Log(Graph): A Near-Optimal High-Performance Graph Representation", PACT'18

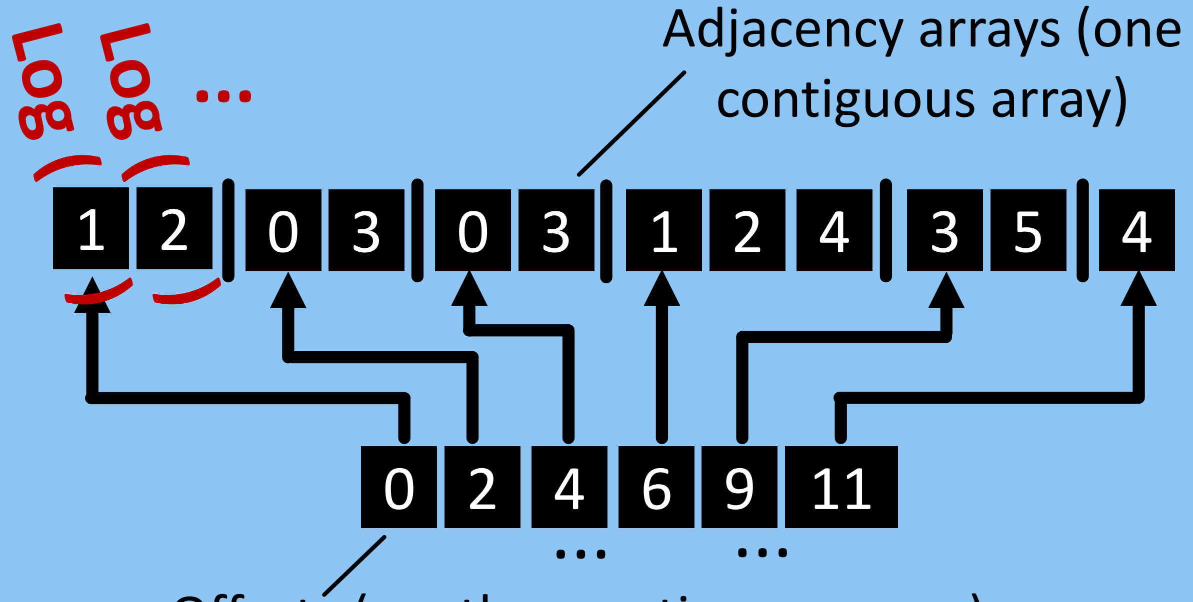
Adjacency Array Graph Representation

1 Log (Vertex labels), Log (Edge weights)

Global bound $\lceil \log n \rceil$



Physical realization



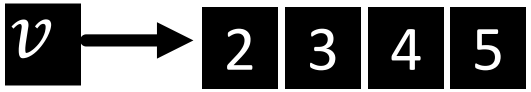
M. Besta et al.: "Log(Graph): A Near-Optimal High-Performance Graph Representation", PACT'18

Adjacency Array Graph Representation

1 Log (Vertex labels), Log (Edge weights)

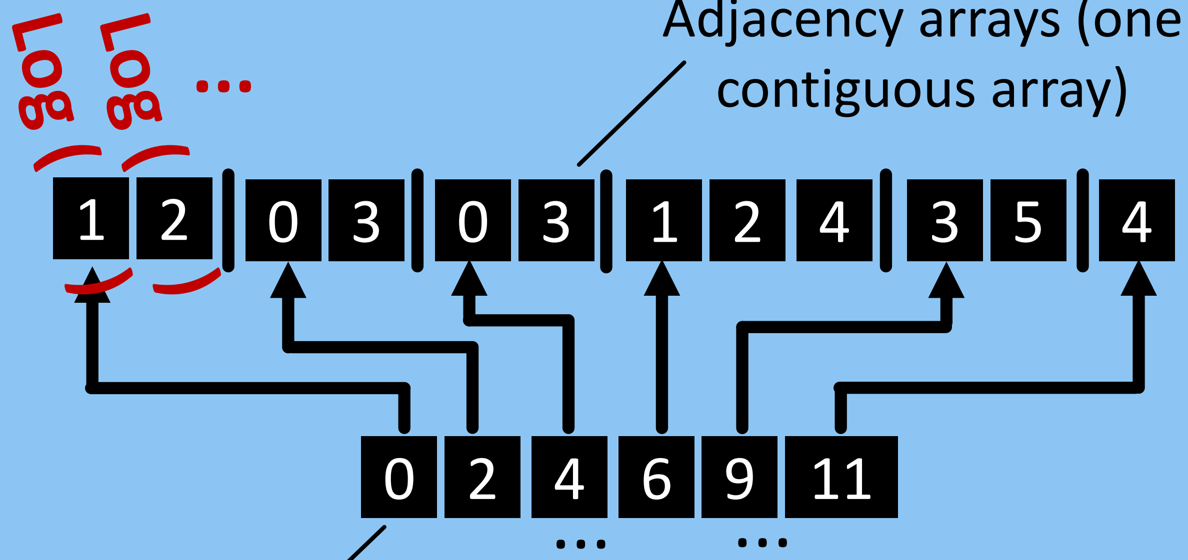
Global bound $\lceil \log n \rceil$

Local bounds $\lceil \log(\max(v_i)) \rceil$



$n = 1\text{M}$

Physical realization

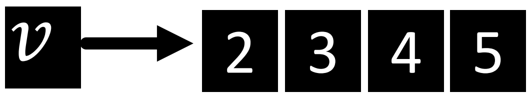


Offsets (another contiguous array)

M. Besta et al.: "Log(Graph): A Near-Optimal High-Performance Graph Representation", PACT'18

Adjacency Array Graph Representation

1 Log (Vertex labels), Log (Edge weights)



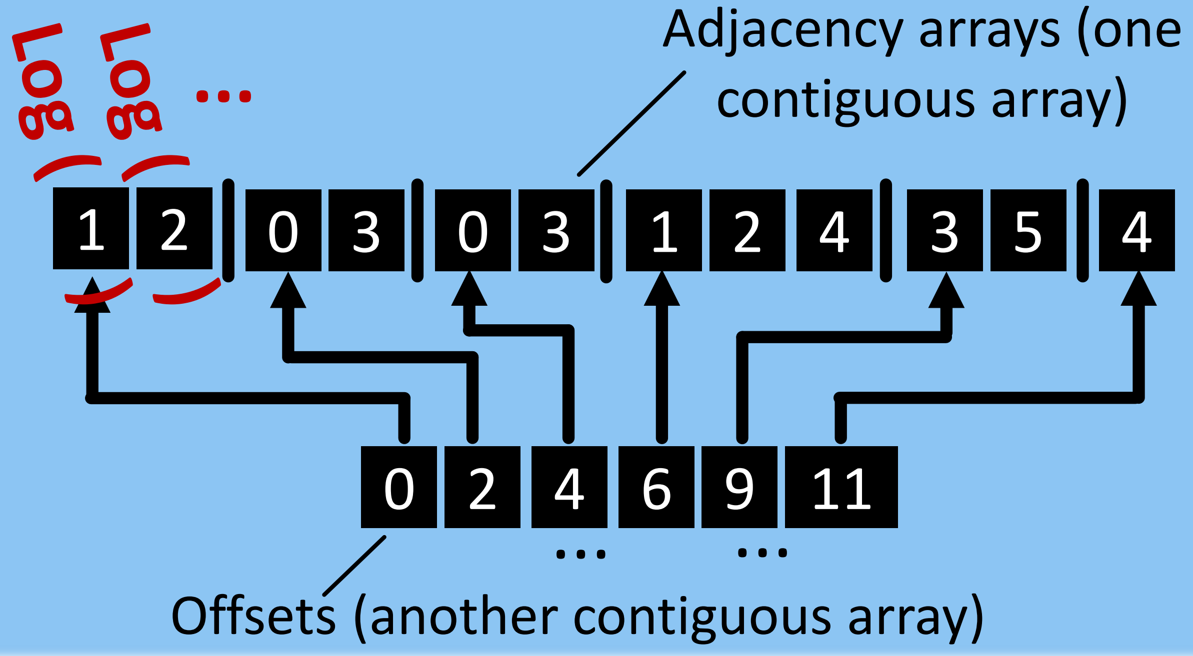
$n = 1\text{M}$

$n = 1\text{M}$

Global bound $\lceil \log n \rceil$

Local bounds $\lceil \log(\max(v_i)) \rceil$

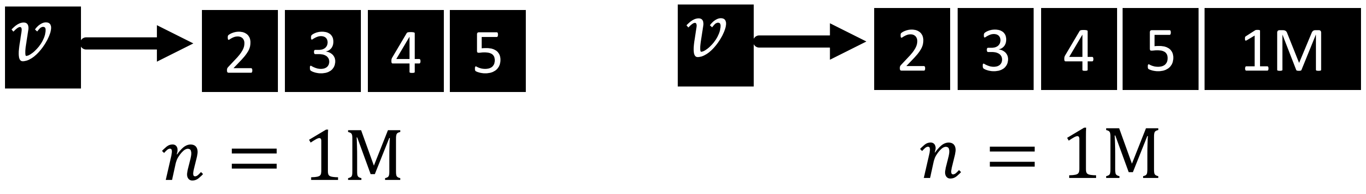
Physical realization

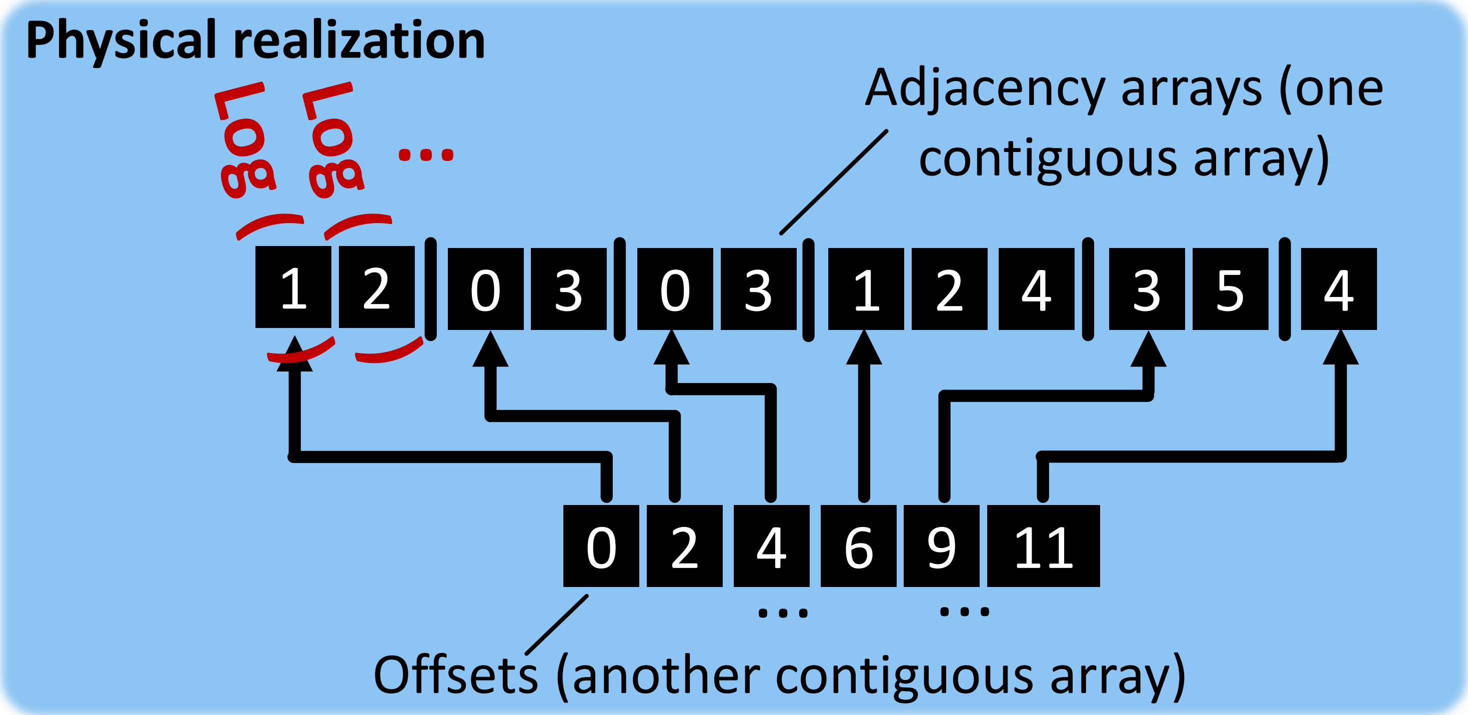


M. Besta et al.: “Log(Graph): A Near-Optimal High-Performance Graph Representation”, PACT’18

Adjacency Array Graph Representation

1 **Log (** Vertex labels **), Log (** Edge weights **)**





Global bound $\lceil \log n \rceil$

Local bounds $\lceil \log(\max(v_i)) \rceil$

Permute vertex labels **to reduce such maximum labels** in as many neighborhoods as possible

M. Besta et al.: "Log(Graph): A Near-Optimal High-Performance Graph Representation", PACT'18

2 **Log (** Offset structure **)**

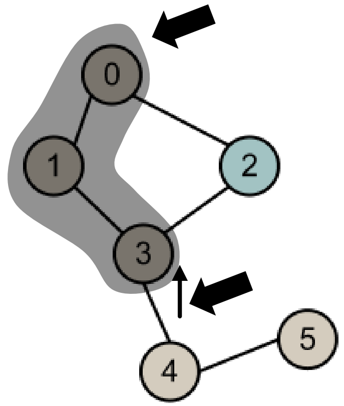
3 **Log (** Adjacency structure **)**

[1] G. J. Jacobson. Succinct Static Data Structures. 1988

M. Besta et al.: "Log(Graph): A Near-Optimal High-Performance Graph Representation", PACT'18

2 **Log (** Offset structure **)**

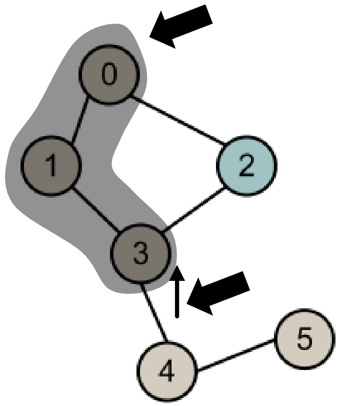
3 **Log (** Adjacency structure **)**



[1] G. J. Jacobson. Succinct Static Data Structures. 1988
M. Besta et al.: "Log(Graph): A Near-Optimal High-Performance Graph Representation", PACT'18

2 **Log (** Offset structure **)**

3 **Log (** Adjacency structure **)**



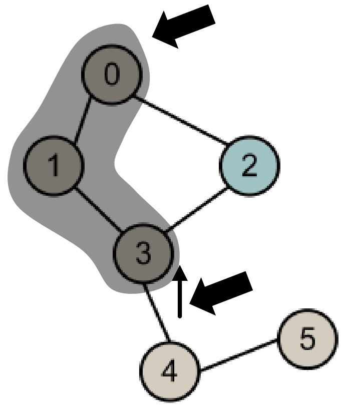
Succinct data structures [1]

[1] G. J. Jacobson. Succinct Static Data Structures. 1988

M. Besta et al.: “Log(Graph): A Near-Optimal High-Performance Graph Representation”, PACT’18

2 **Log (** Offset structure **)**

3 **Log (** Adjacency structure **)**



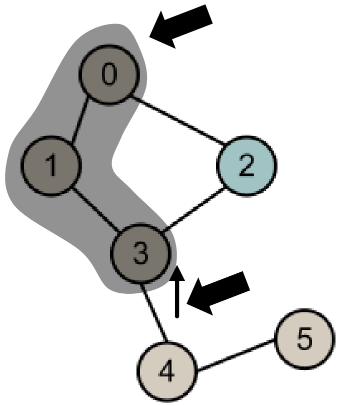
Succinct data structures [1]

They use **[Q] + *o*(Q) bits** ([Q] - lower bound), they answer various queries in ***o*(Q) time**.

[1] G. J. Jacobson. Succinct Static Data Structures. 1988

2 Log (Offset structure)

3 Log (Adjacency structure)



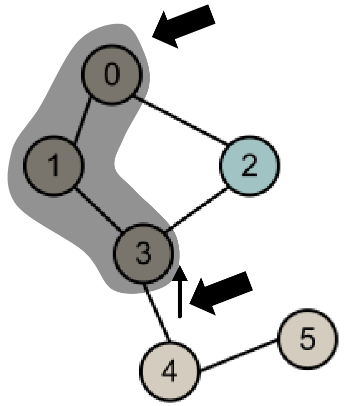
Succinct data structures [1]

They use **[Q] + *o*(Q) bits** ([Q] - lower bound), they answer various queries in ***o*(Q) time**.

Compact data structures [1]

[1] G. J. Jacobson. Succinct Static Data Structures. 1988

2 Log (Offset structure) **3** Log (Adjacency structure)



Succinct data structures [1]

They use **[Q] + *o*(Q) bits** ([Q] - lower bound), they answer various queries in ***o*(Q) time**.

Compact data structures [1]

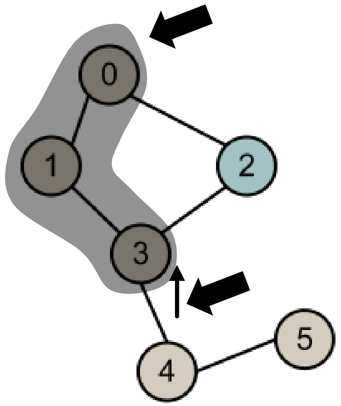
They use ***O*(Q) bits** ([Q] - lower bound), they answer various queries in ***o*(Q) time**.

[1] G. J. Jacobson. Succinct Static Data Structures. 1988

M. Besta et al.: “Log(Graph): A Near-Optimal High-Performance Graph Representation”, PACT’18

2 **Log (** Offset structure **)**

3 **Log (** Adjacency structure **)**



Succinct data structures [1]

They use **[Q] + *o*(Q) bits** ([Q] - lower bound), they answer various queries in ***o*(Q) time**.

Compact data structures [1]

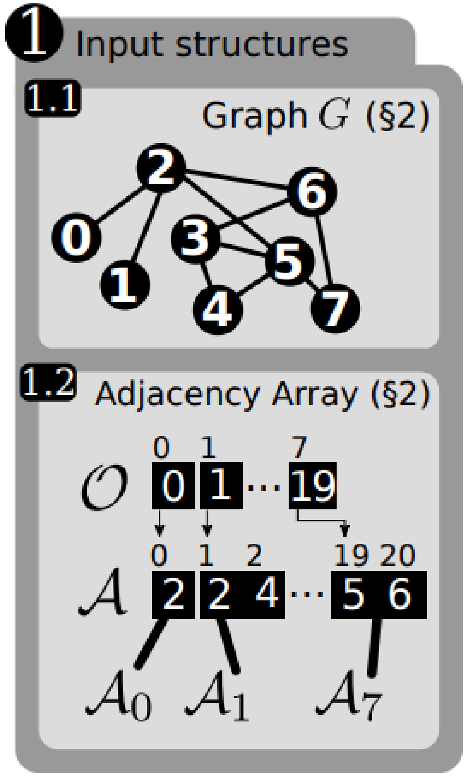
They use ***O*(Q) bits** ([Q] - lower bound), they answer various queries in ***o*(Q) time**.

We show that they are in practice both small and fast!

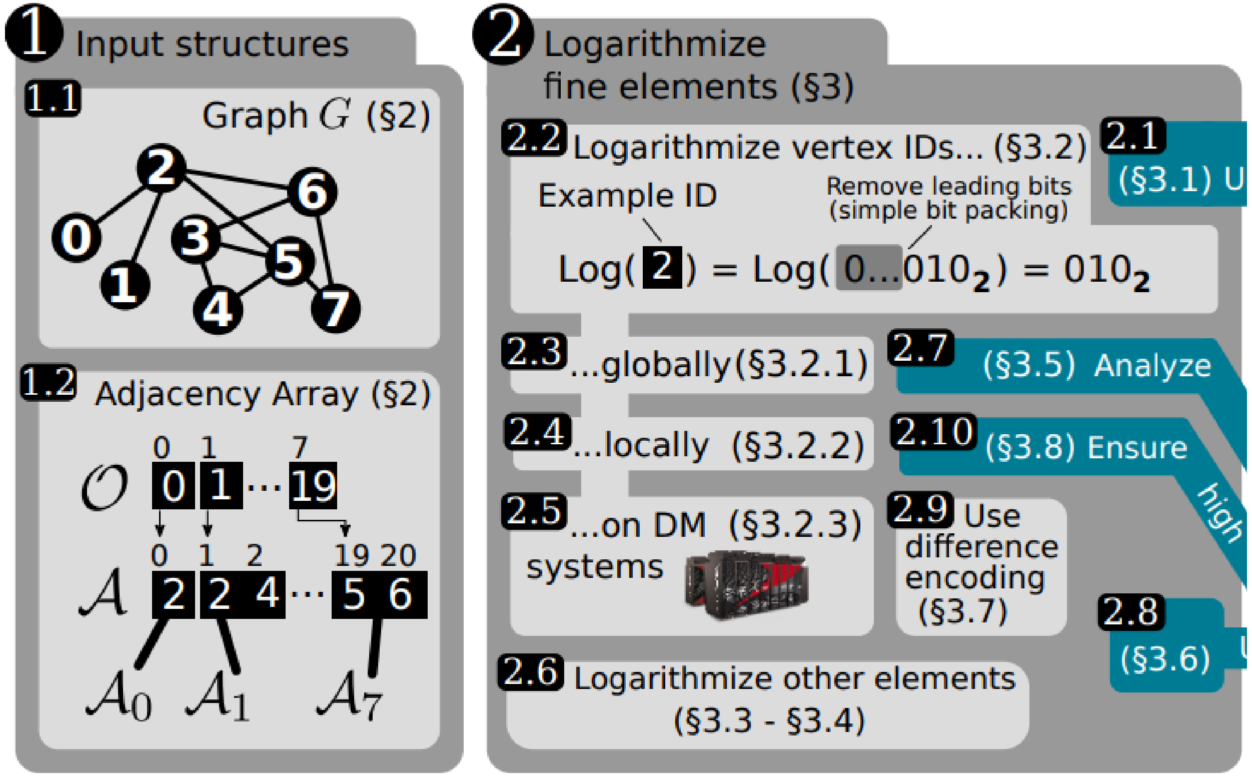
[1] G. J. Jacobson. Succinct Static Data Structures. 1988
M. Besta et al.: "Log(Graph): A Near-Optimal High-Performance Graph Representation", PACT'18

Overview of Full Log(Graph) Design

Overview of Full Log(Graph) Design

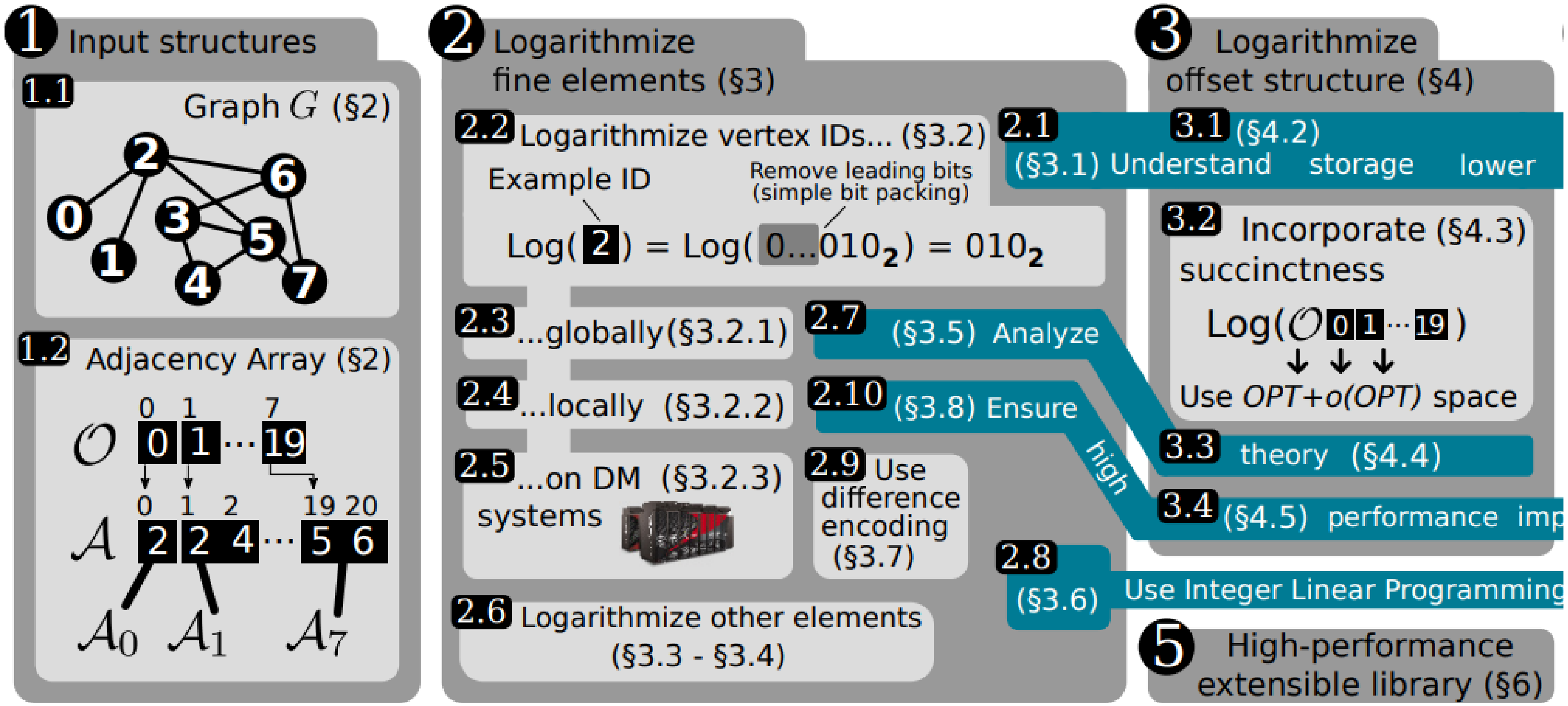


OVERVIEW OF FULL LOG(GRAPH) DESIGN



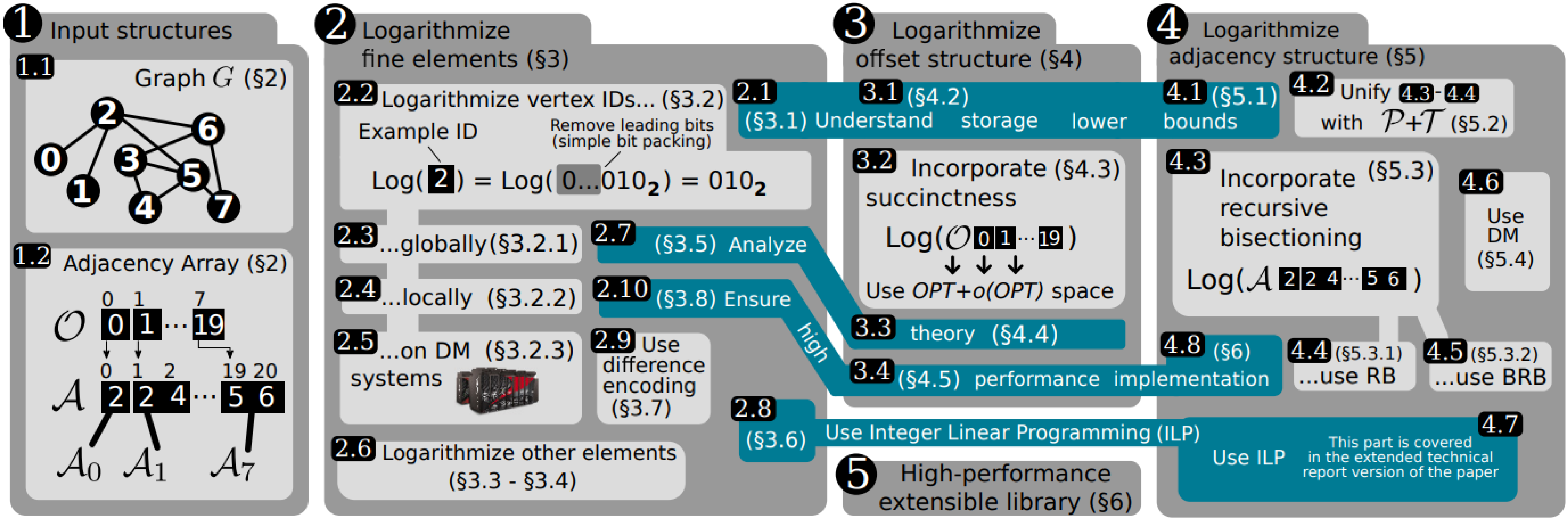
M. Besta et al.: “Log(Graph): A Near-Optimal High-Performance Graph Representation”, PACT’18

Overview of Full Log(Graph) Design



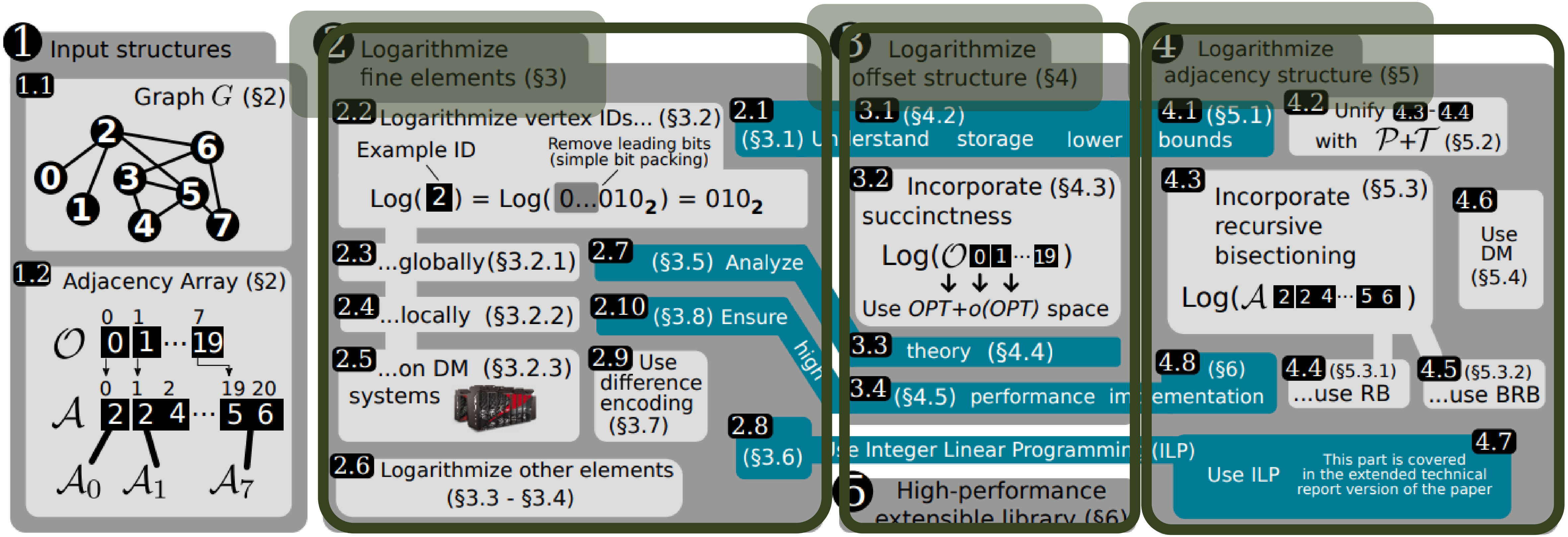
M. Besta et al.: "Log(Graph): A Near-Optimal High-Performance Graph Representation", PACT'18

Overview of Full Log(Graph) Design



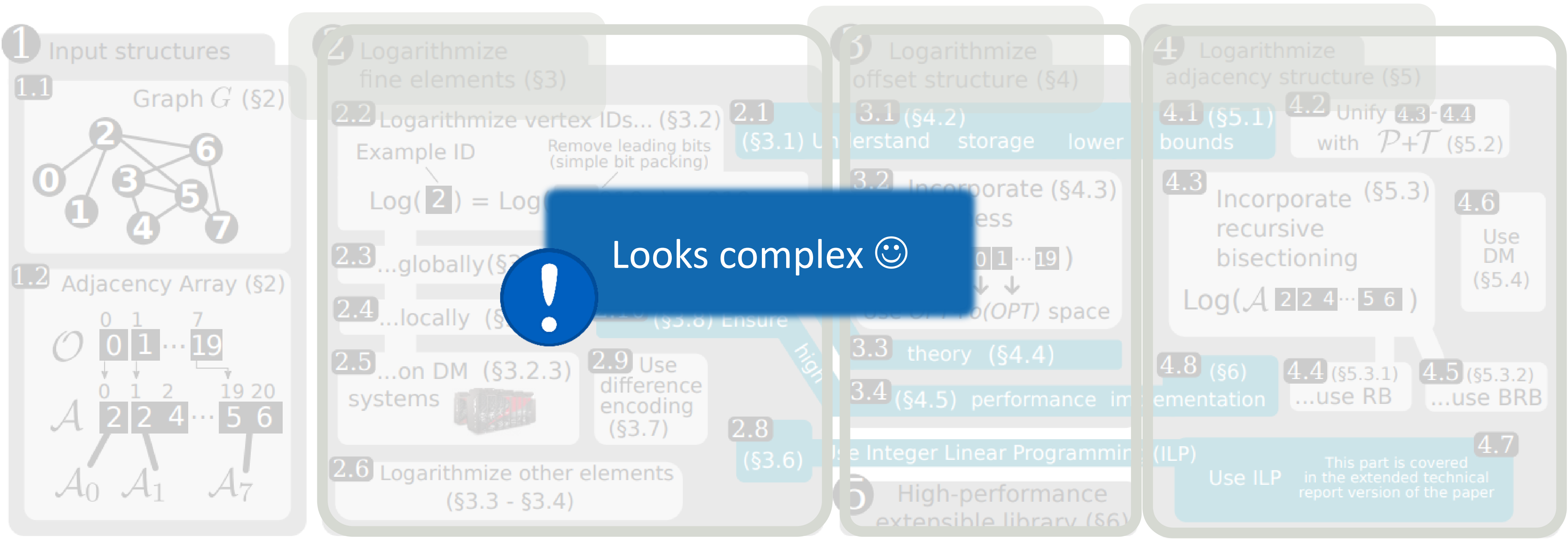
M. Besta et al.: "Log(Graph): A Near-Optimal High-Performance Graph Representation", PACT'18

OVERVIEW OF FULL LOG(GRAPH) DESIGN



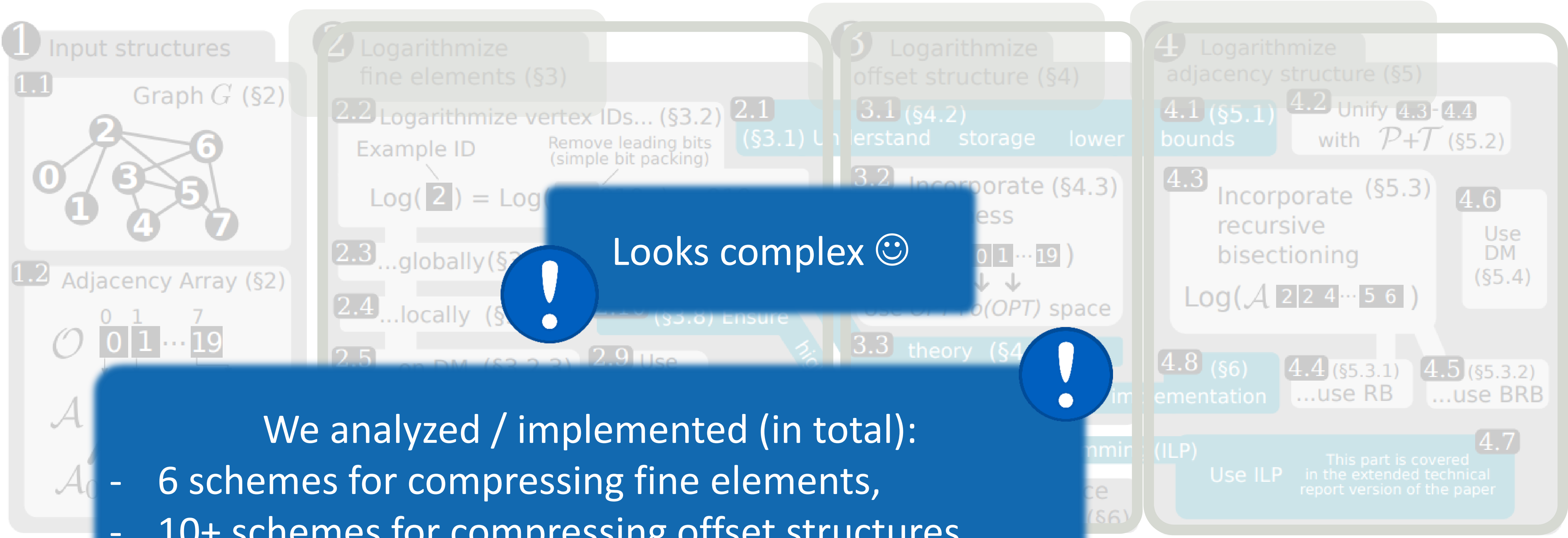
M. Besta et al.: “Log(Graph): A Near-Optimal High-Performance Graph Representation”, PACT’18

Overview of Full Log(Graph) Design



M. Besta et al.: “Log(Graph): A Near-Optimal High-Performance Graph Representation”, PACT’18

Overview of Full Log(Graph) Design



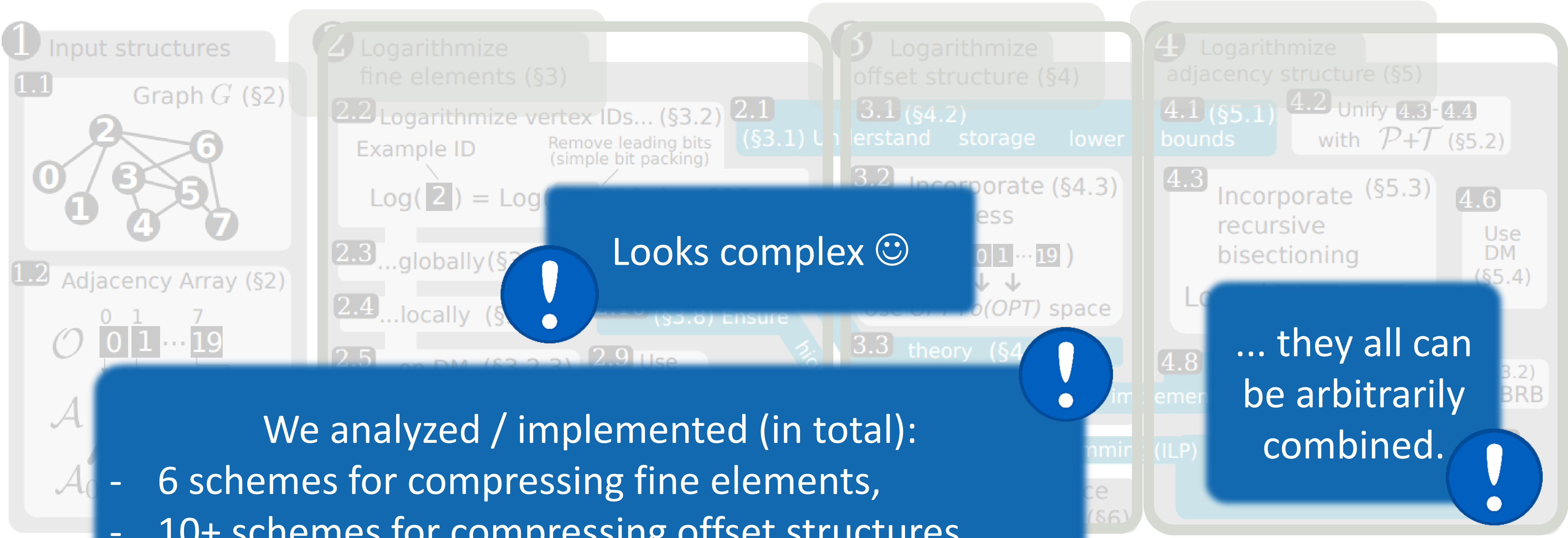
Looks complex ☺

We analyzed / implemented (in total):

- 6 schemes for compressing fine elements,
- 10+ schemes for compressing offset structures,
- 4+ schemes for compressing adjacency structures

M. Besta et al.: "Log(Graph): A Near-Optimal High-Performance Graph Representation", PACT'18

Overview of Full Log(Graph) Design



Looks complex ☺

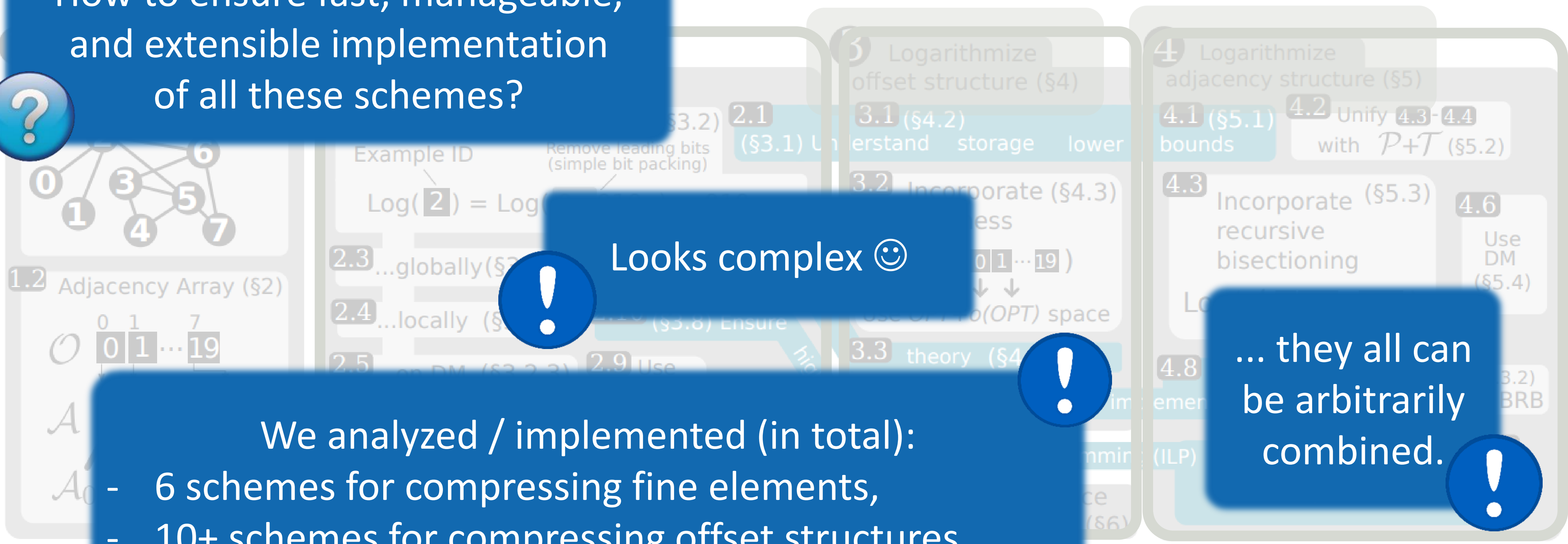
We analyzed / implemented (in total):

- 6 schemes for compressing fine elements,
- 10+ schemes for compressing offset structures,
- 4+ schemes for compressing adjacency structures

... they all can be arbitrarily combined.

M. Besta et al.: "Log(Graph): A Near-Optimal High-Performance Graph Representation", PACT'18

Overview of Full Log(Graph) Design



How to ensure fast, manageable, and extensible implementation of all these schemes?

Looks complex ☺

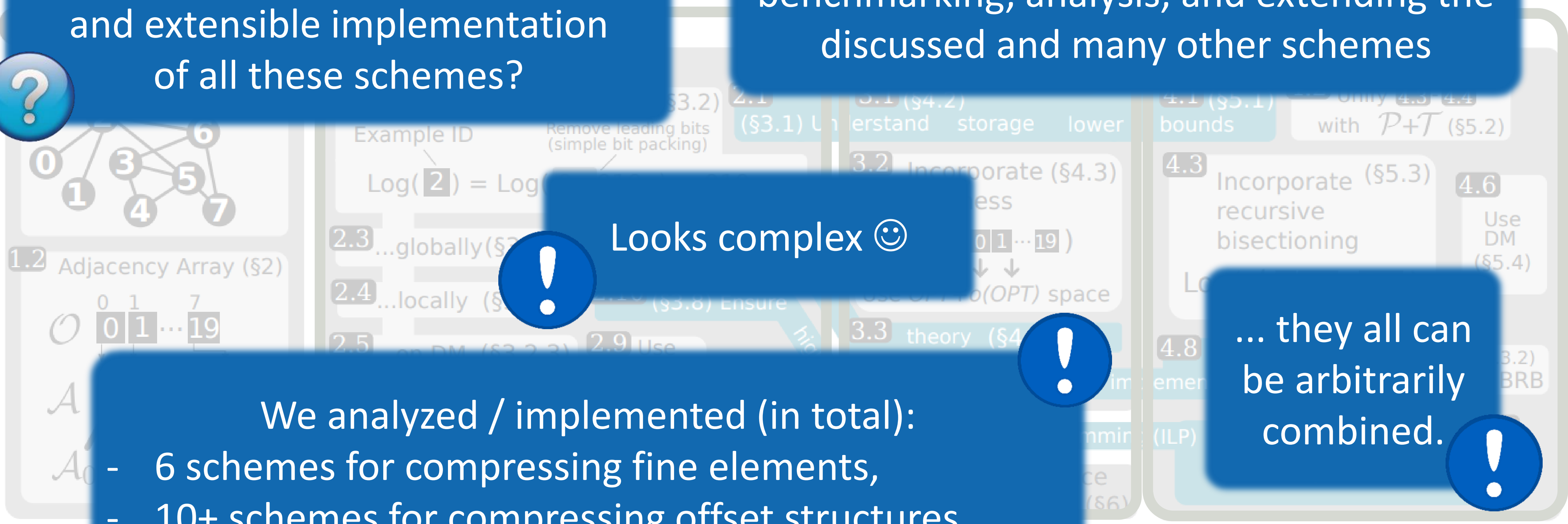
We analyzed / implemented (in total):

- 6 schemes for compressing fine elements,
- 10+ schemes for compressing offset structures,
- 4+ schemes for compressing adjacency structures

... they all can be arbitrarily combined.

M. Besta et al.: “Log(Graph): A Near-Optimal High-Performance Graph Representation”, PACT’18

Overview of Full Log(Graph) Design



How to ensure fast, manageable, and extensible implementation of all these schemes?

We use C++ templates to develop a platform that facilitates implementation, benchmarking, analysis, and extending the discussed and many other schemes

Looks complex ☺

We analyzed / implemented (in total):

- 6 schemes for compressing fine elements,
- 10+ schemes for compressing offset structures,
- 4+ schemes for compressing adjacency structures

... they all can be arbitrarily combined.

M. Besta et al.: “Log(Graph): A Near-Optimal High-Performance Graph Representation”, PACT’18

Overview of Full Log(Graph) Design

How to ensure fast, manageable, and extensible implementation of all these schemes?

We use C++ templates to develop a platform that facilitates implementation, benchmarking, analysis, and extending the discussed and many other schemes

COMING SOON

Looks complex ☺

... they all can be arbitrarily combined.

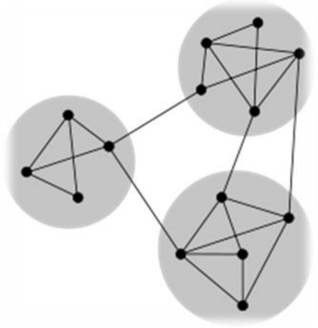
We analyzed / implemented (in total):

- 6 schemes for compressing fine elements,
- 10+ schemes for compressing offset structures,
- 4+ schemes for compressing adjacency structures

M. Besta et al.: "Log(Graph): A Near-Optimal High-Performance Graph Representation", PACT'18

1 Log (Vertex labels), Log (Edge weights) Storage, Performance

SSSP

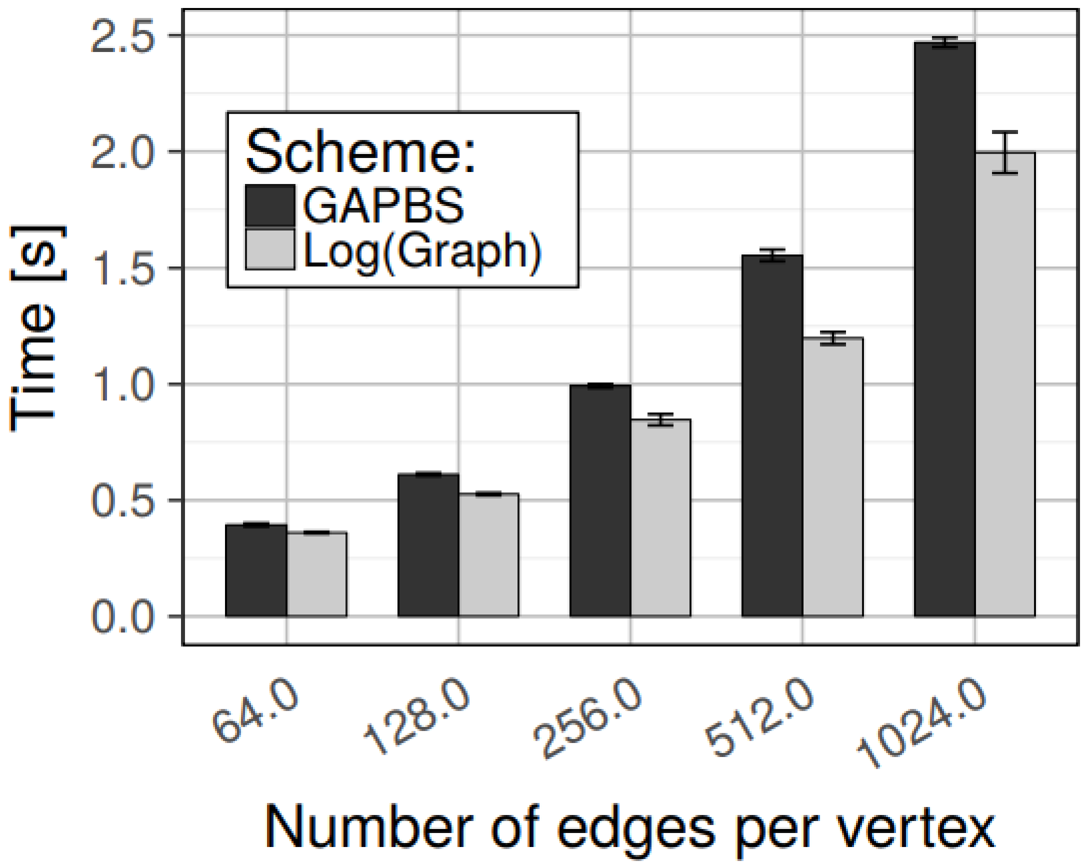


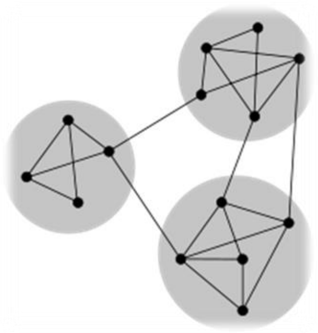
Kronecker graphs
Number of vertices: 4M

M. Besta et al.: "Log(Graph): A Near-Optimal High-Performance Graph Representation", PACT'18

SSSP

1 **Log (Vertex labels), Log (Edge weights)** **Storage, Performance**



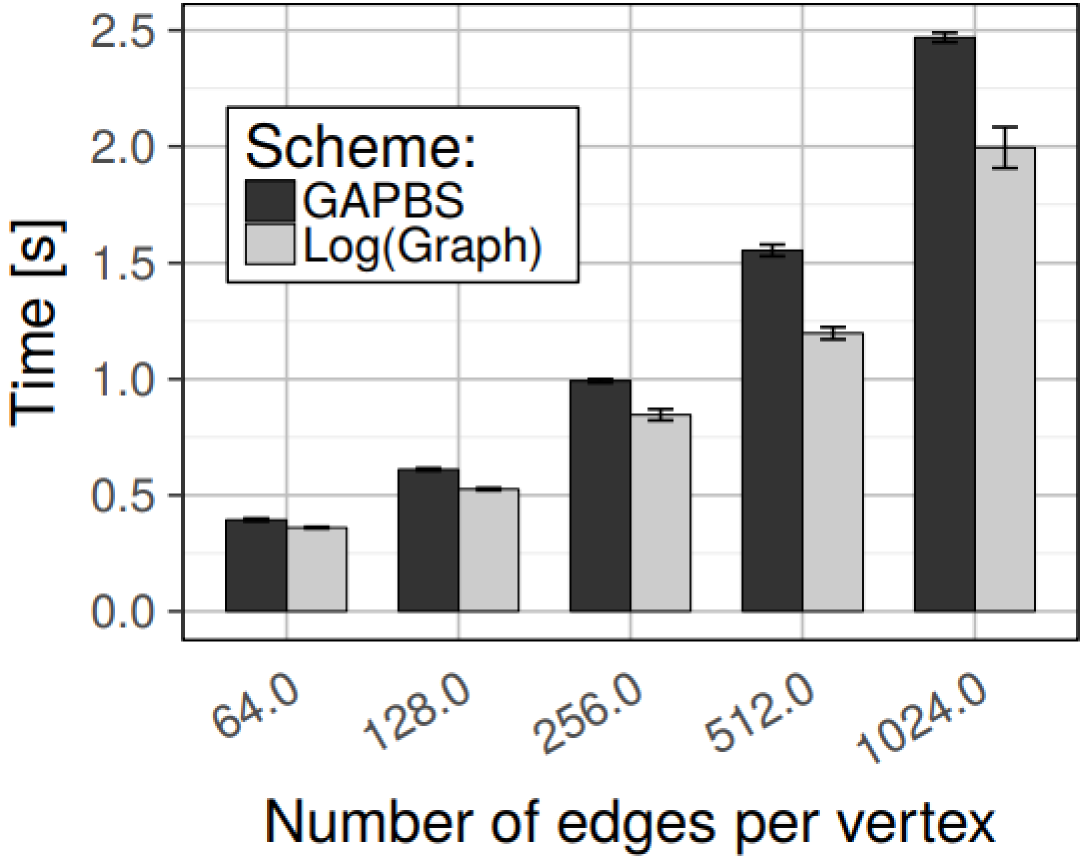


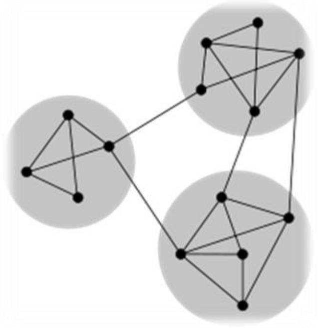
Kronecker graphs
Number of vertices: 4M

M. Besta et al.: "Log(Graph): A Near-Optimal High-Performance Graph Representation", PACT'18

SSSP

1 **Log (Vertex labels), Log (Edge weights)** **Storage, Performance**





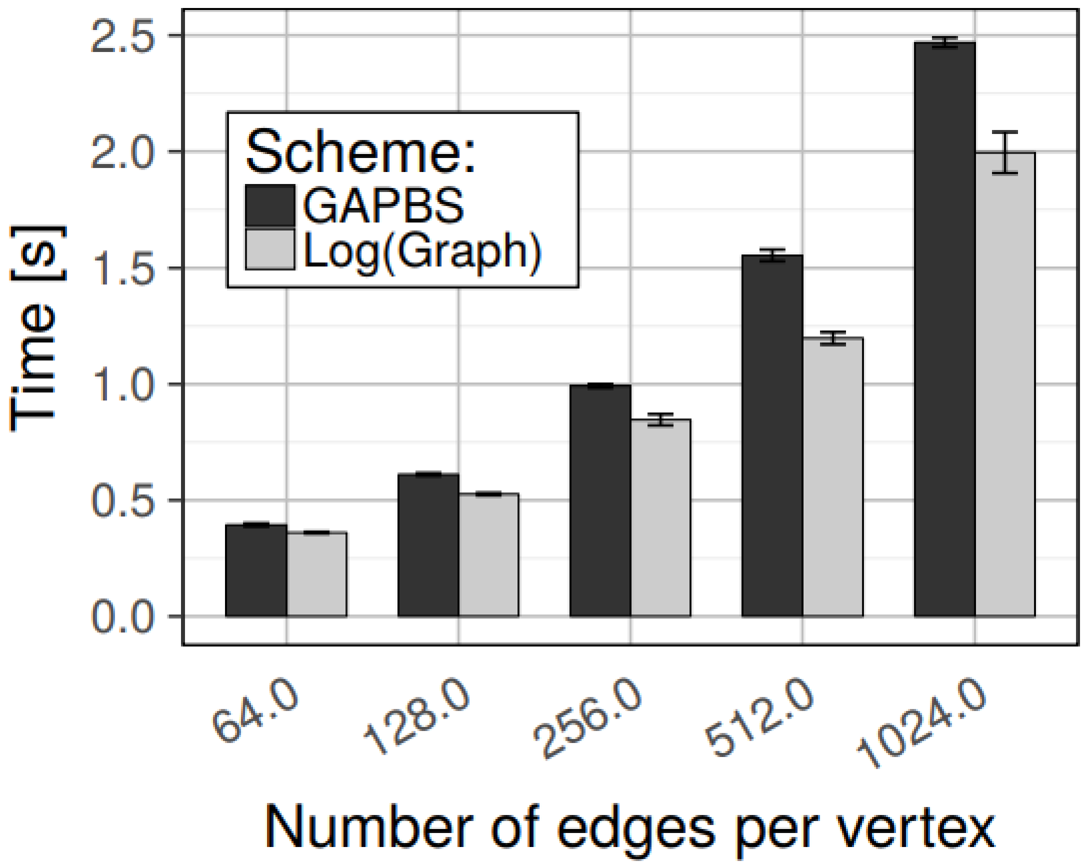
Kronecker graphs
Number of vertices: 4M

Log(Graph) consistently reduces storage overhead (by 20-35%)

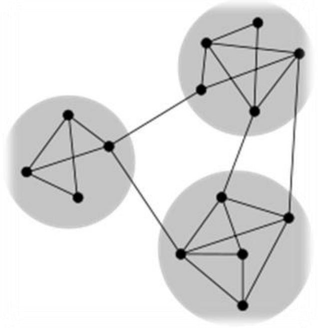
M. Besta et al.: "Log(Graph): A Near-Optimal High-Performance Graph Representation", PACT'18

SSSP

1 Log (Vertex labels), Log (Edge weights) Storage, Performance



Log(Graph) **accelerates** GAPBS



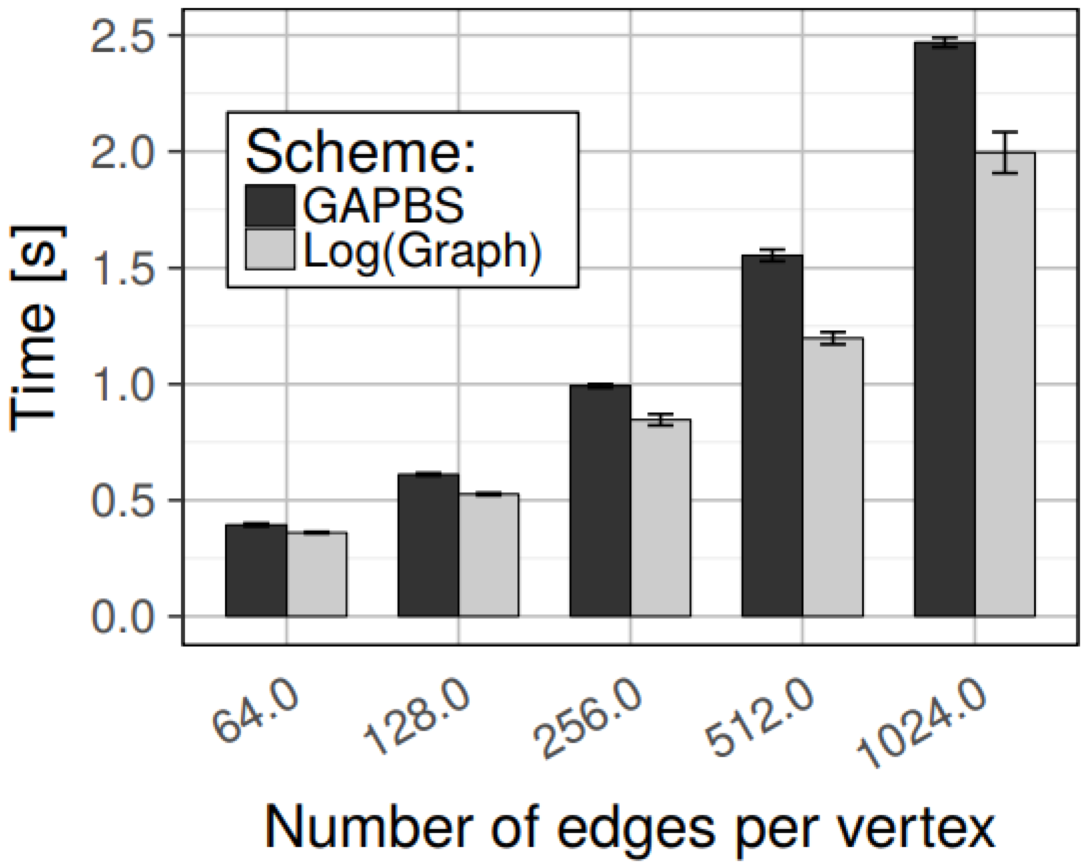
Kronecker graphs
Number of vertices: 4M

Log(Graph) consistently reduces storage overhead (by 20-35%)

M. Besta et al.: “Log(Graph): A Near-Optimal High-Performance Graph Representation”, PACT’18

1 Log (Vertex labels), Log (Edge weights) Storage, Performance

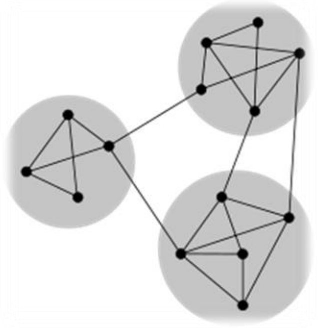
SSSP



Log(Graph) **accelerates** GAPBS

Both storage and performance are improved **simultaneously**

Log(Graph) consistently reduces storage overhead (by 20-35%)



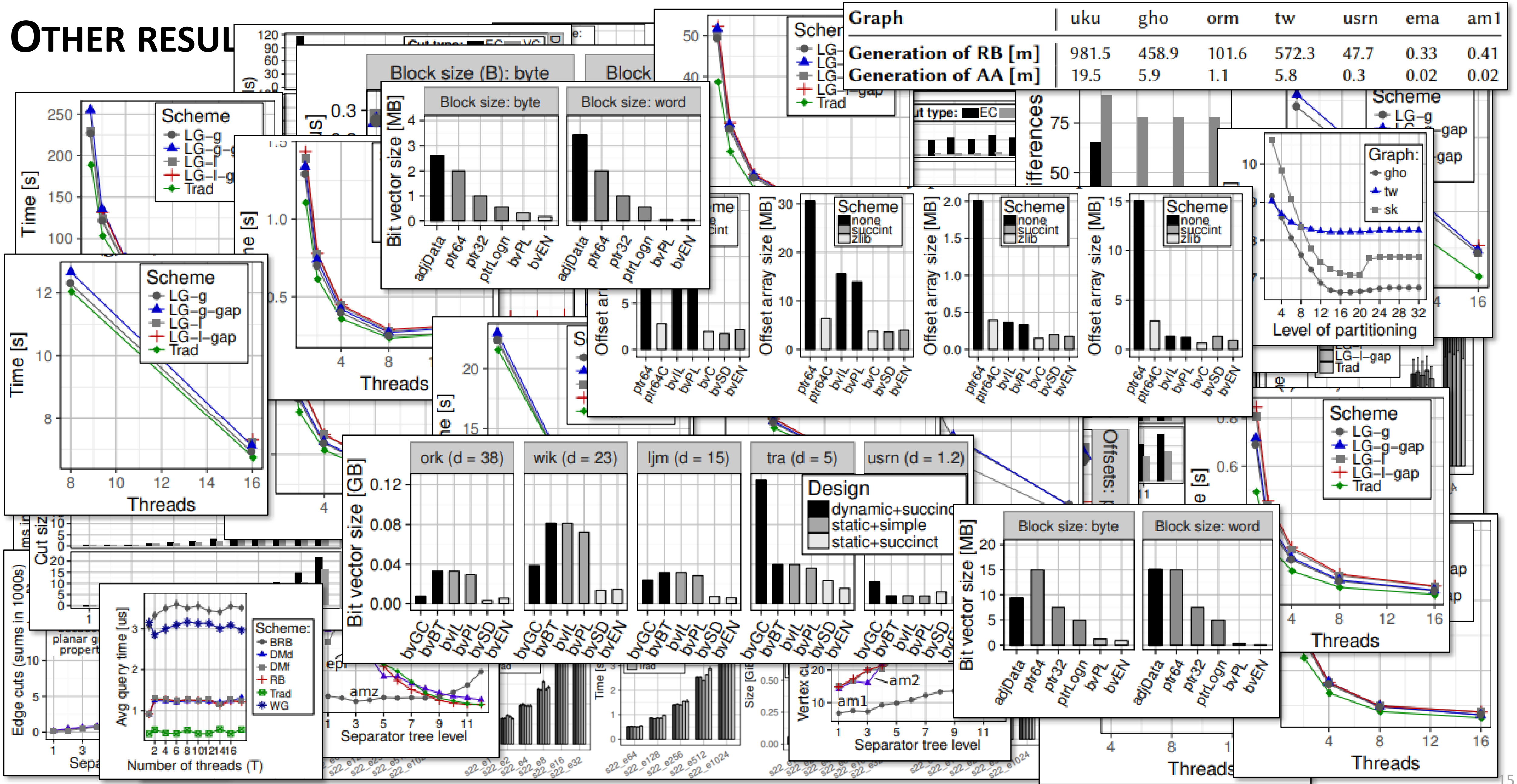
Kronecker graphs
Number of vertices: 4M

M. Besta et al.: "Log(Graph): A Near-Optimal High-Performance Graph Representation", PACT'18

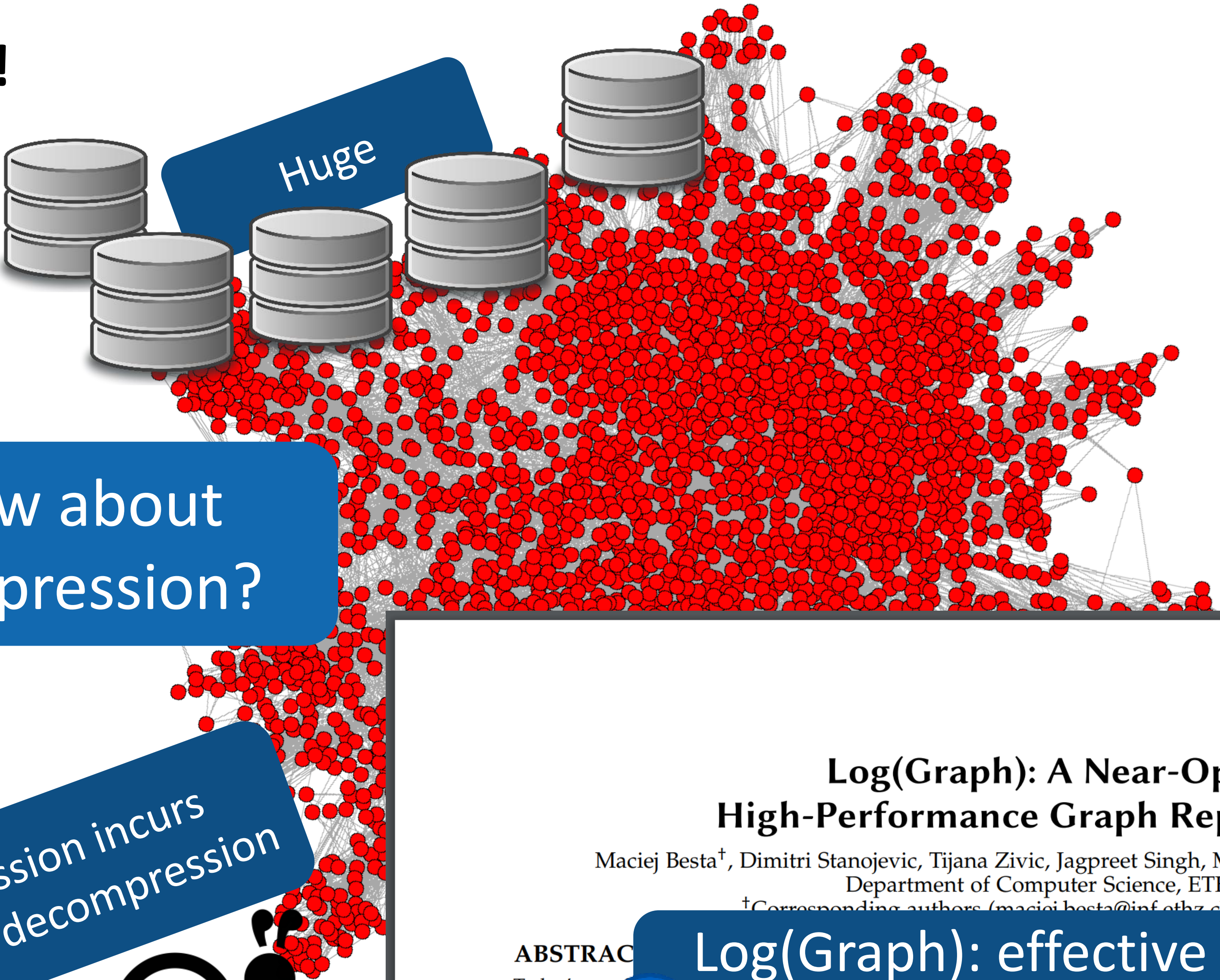
OTHER RESULTS

OTHER RESULTS

Graph	uku	gho	orm	tw	usrn	ema	am1
Generation of RB [m]	981.5	458.9	101.6	572.3	47.7	0.33	0.41
Generation of AA [m]	19.5	5.9	1.1	5.8	0.3	0.02	0.02



Problems!



Huge

How about compression?

Compression incurs expensive decompression

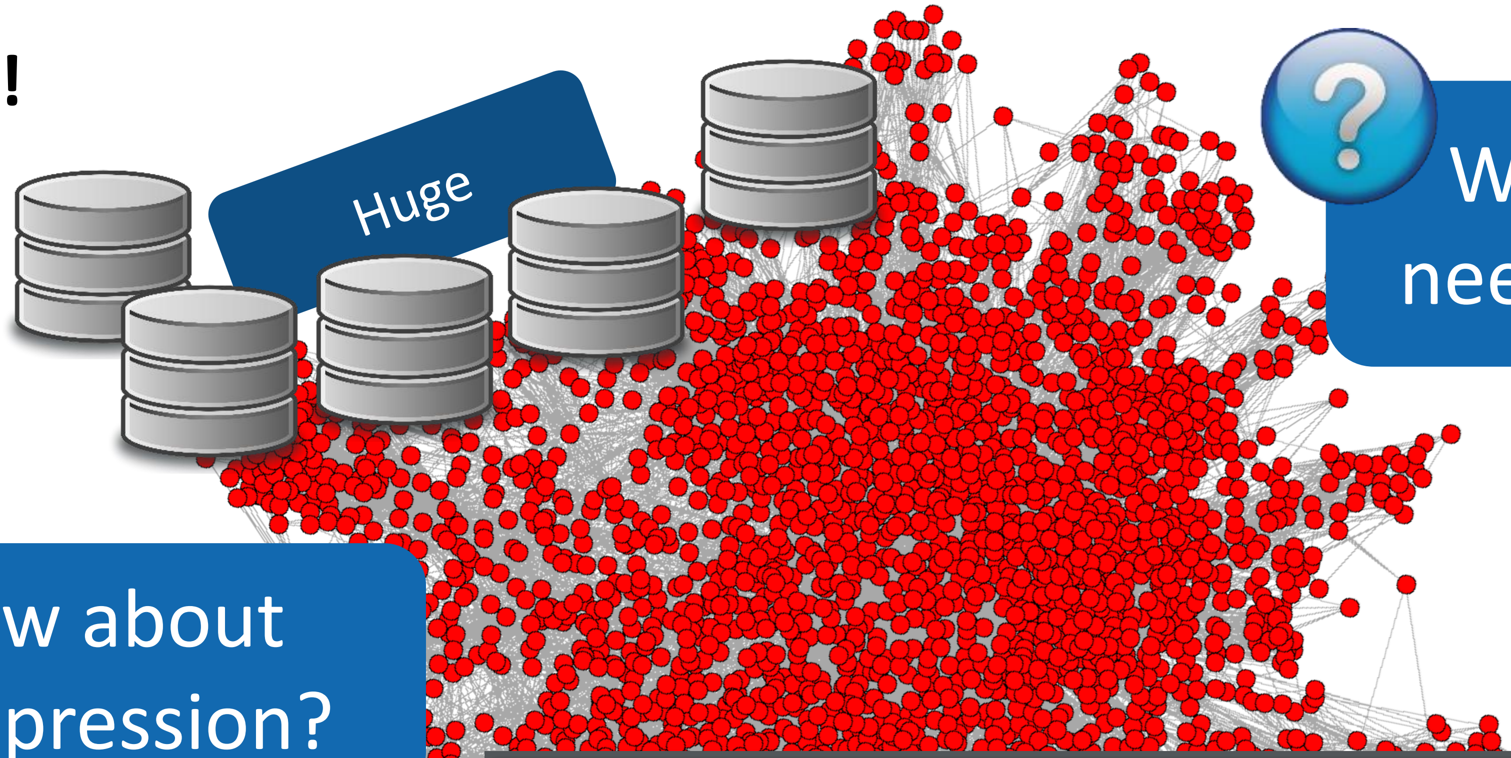
Log(Graph): A Near-Optimal High-Performance Graph Representation

Maciej Besta†, Dimitri Stanojevic, Tijana Zivic, Jagpreet Singh, Maurice Hoerold, Torsten Hoefler†
Department of Computer Science, ETH Zurich
†Corresponding authors (maciej.besta@inf.ethz.ch, htor@inf.ethz.ch)

ABSTRAC
Today's gra
social netw
edges. Yet
dard graph ... entations such as adjacency lists waste a
significant number of bits while graph compression schemes
size of such graphs is becoming increasingly important.

Log(Graph): effective compression with low-overhead decompression!

Problems!



What if we don't need full precision?

How about compression?

Compression incurs expensive decompression

Log(Graph): A Near-Optimal High-Performance Graph Representation

Maciej Besta†, Dimitri Stanojevic, Tijana Zivic, Jagpreet Singh, Maurice Hoerold, Torsten Hoefler†
Department of Computer Science, ETH Zurich
†Corresponding authors (maciej.besta@inf.ethz.ch, htor@inf.ethz.ch)

ABSTRAC
Today's gra
social netw
edges. Yet
dard graph
significant number of bits while graph compression schemes
size of such graphs is becoming increasingly important.

Log(Graph): effective compression with low-overhead decompression!

Problems!

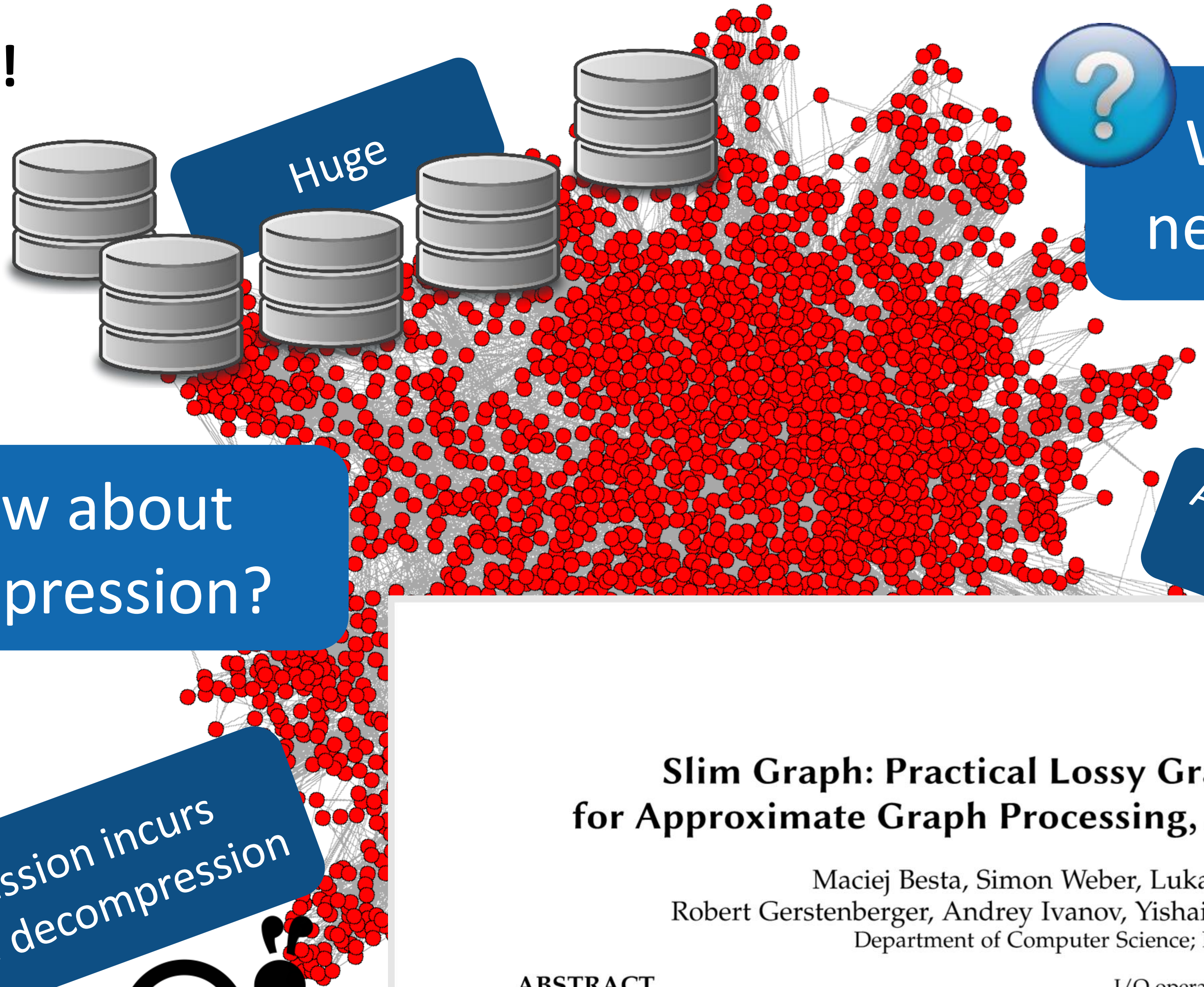
Huge

What if we don't need full precision?

How about compression?

Compression incurs expensive decompression

Problems!



Huge

What if we don't need full precision?

How about compression?

Compression incurs expensive decompression

Accepted @ ACM/IEEE SC19, Best paper finalist

Slim Graph: Practical Lossy Graph Compression for Approximate Graph Processing, Storage, and Analytics

Maciej Besta, Simon Weber, Lukas Gianinazzi,
Robert Gerstenberger, Andrey Ivanov, Yishai Oltchik, Torsten Hoefler
Department of Computer Science; ETH Zurich

ABSTRACT

We propose Slim Graph: the first programming model and framework for practical lossy graph compression that facilitates high-performance approximate graph processing, storage, and analytics. Slim Graph enables the developer to

I/O operations, the amount of data communicated over the network, and by storing a larger fraction of data in caches.

There exist many *lossless* schemes for graph compression, including WebGraph [21], k^2-trees [25], and others [17]. They provide various degrees of storage reductions. Unfortunately,

JPEG compression level: 0%

M. Besta et al.: “Slim Graph: Practical Lossy Graph Compression for Approximate Graph Processing, Storage, and Analytics”, ACM/IEEE SC’19

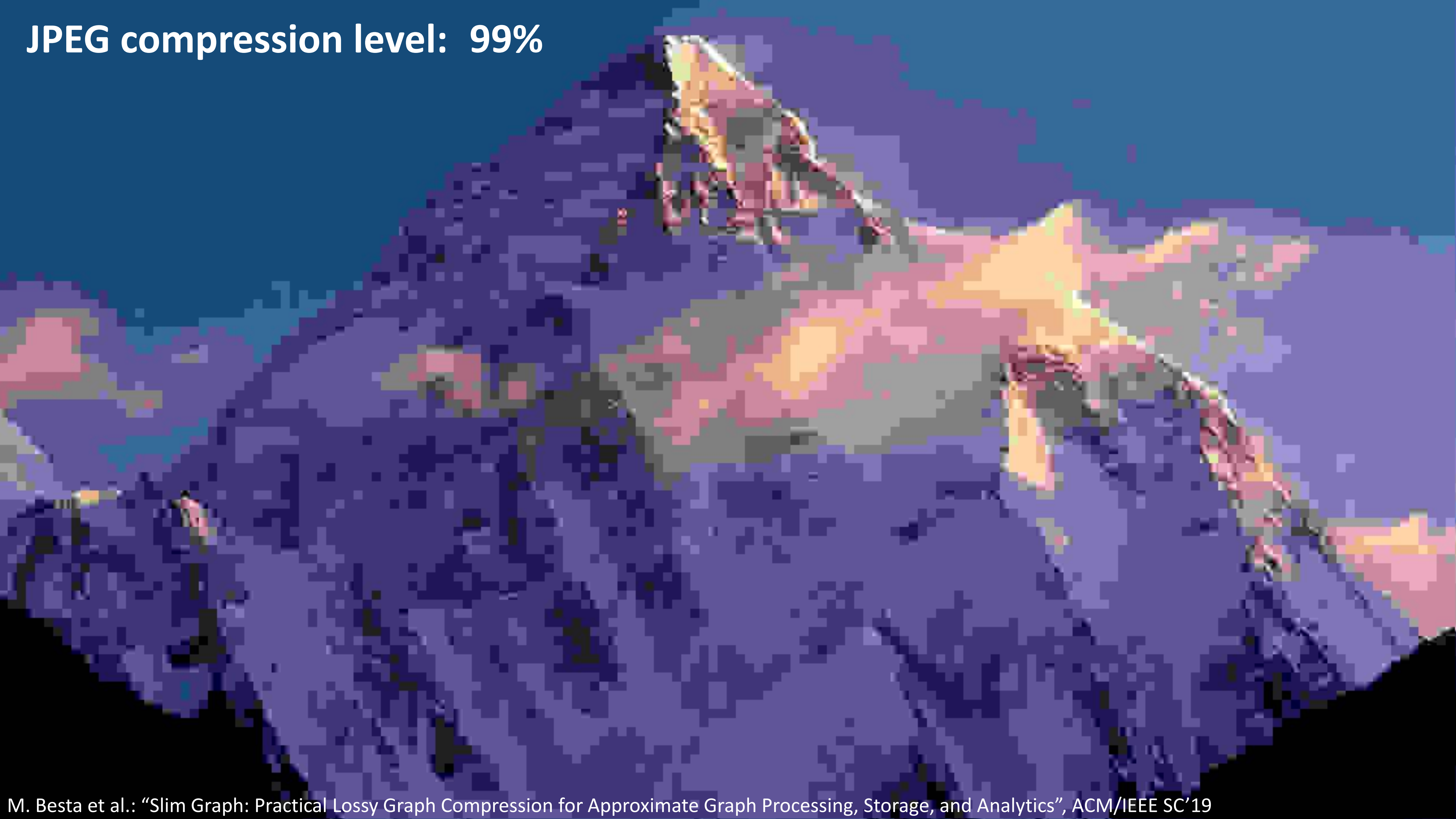
JPEG compression level: 50%

M. Besta et al.: “Slim Graph: Practical Lossy Graph Compression for Approximate Graph Processing, Storage, and Analytics”, ACM/IEEE SC’19

JPEG compression level: 90%

M. Besta et al.: “Slim Graph: Practical Lossy Graph Compression for Approximate Graph Processing, Storage, and Analytics”, ACM/IEEE SC’19

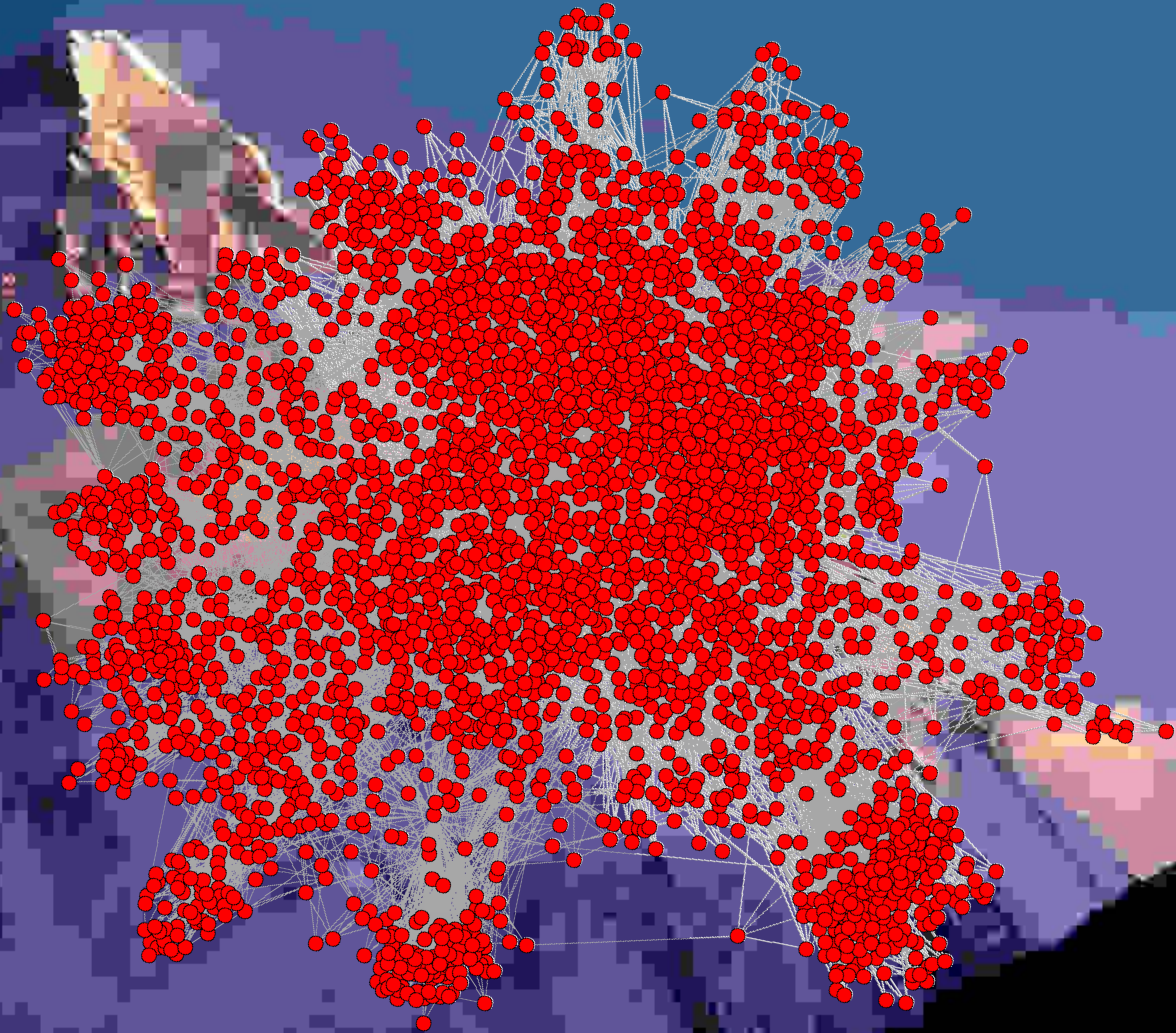
JPEG compression level: 99%



M. Besta et al.: “Slim Graph: Practical Lossy Graph Compression for Approximate Graph Processing, Storage, and Analytics”, ACM/IEEE SC’19

JPEG compression level: 99%

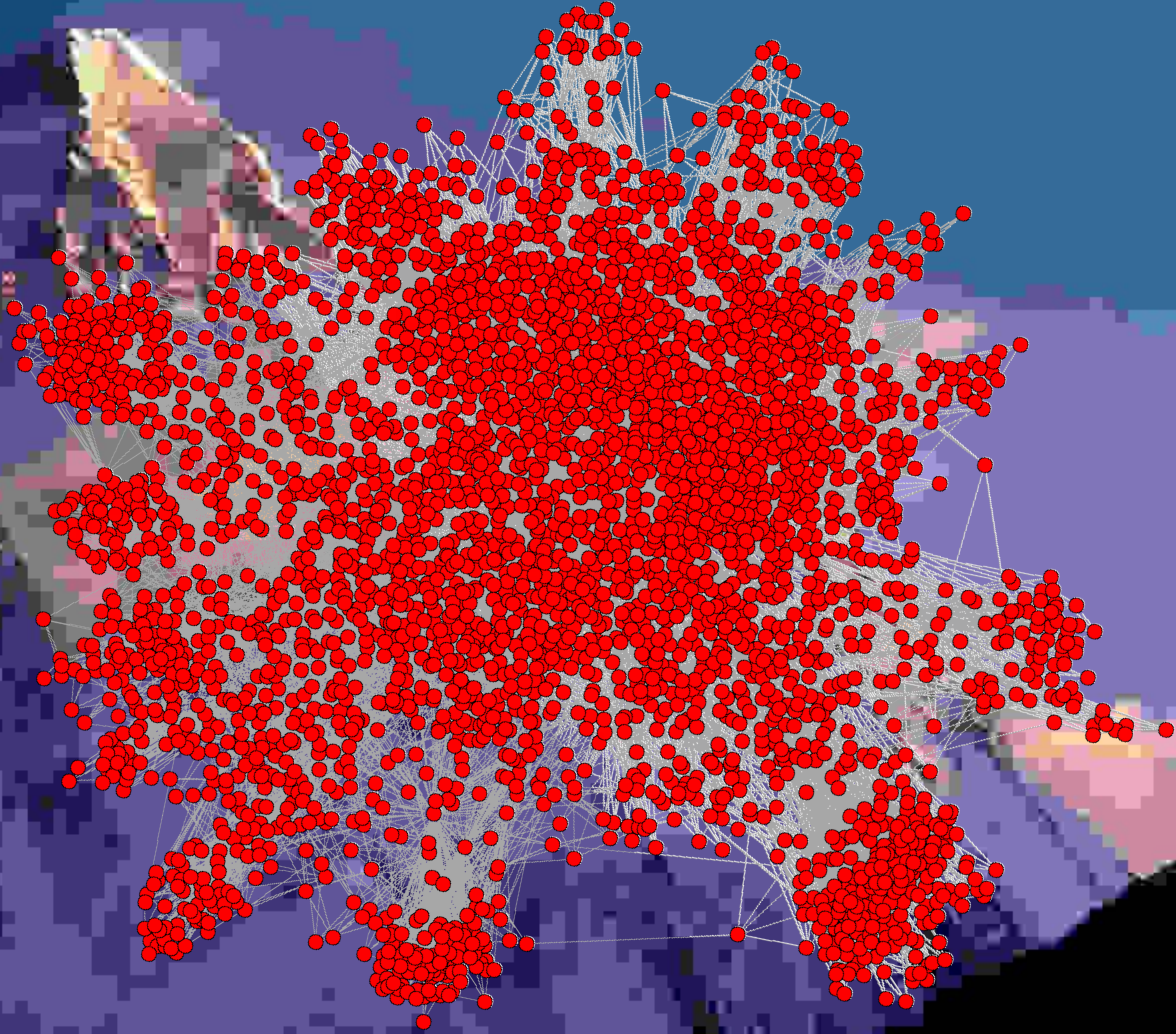
Can we apply a similar reasoning to graphs? How?



M. Besta et al.: "Slim Graph: Practical Lossy Graph Compression for Approximate Graph Processing, Storage, and Analytics", ACM/IEEE SC'19

JPEG compression level: 99%

Can we apply a similar reasoning to graphs? How?



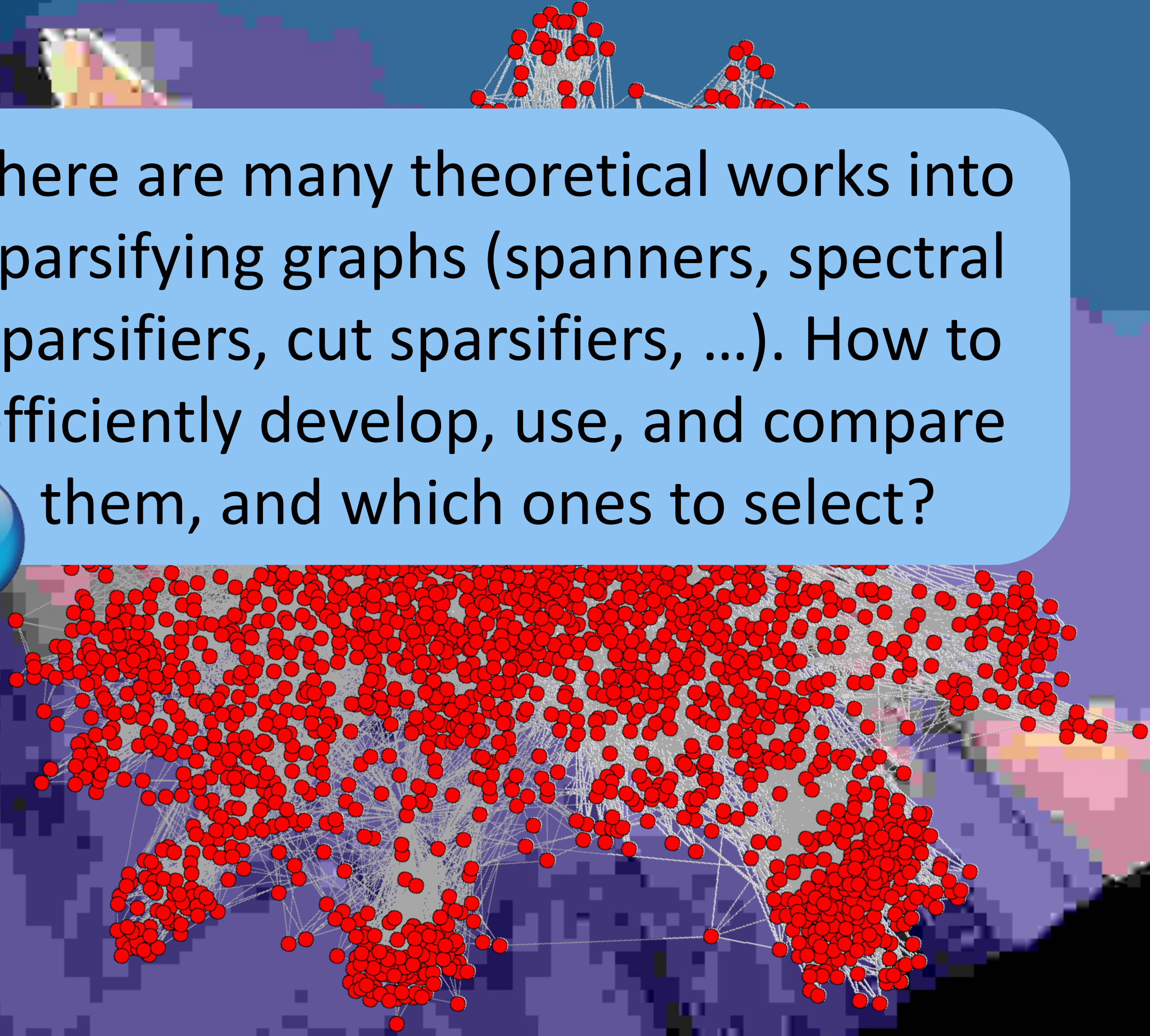
What should we pay attention to? (there is no “visual similarity” measure in this case…)

M. Besta et al.: “Slim Graph: Practical Lossy Graph Compression for Approximate Graph Processing, Storage, and Analytics”, ACM/IEEE SC’19

JPEG compression level: 99%

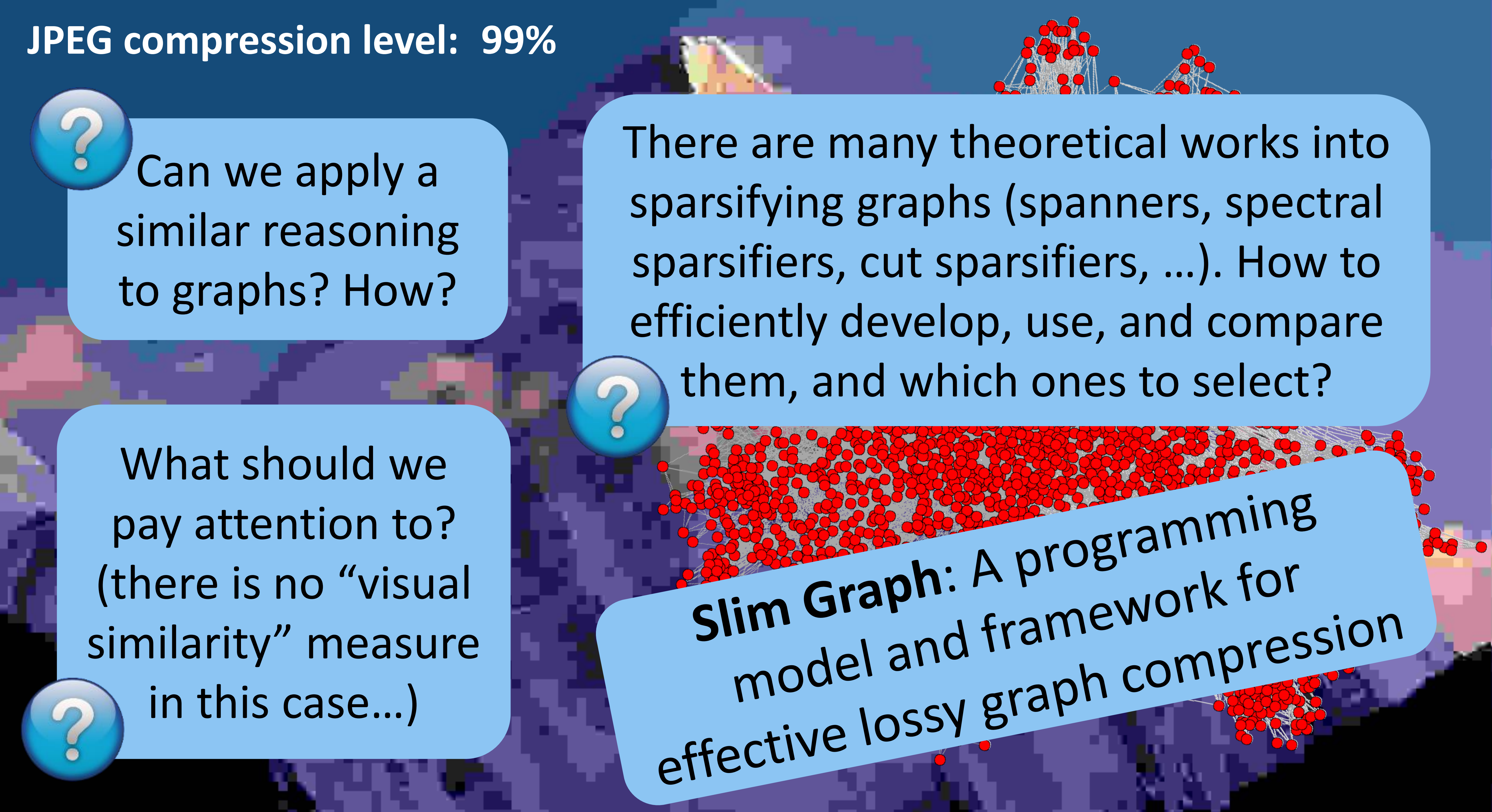
Can we apply a similar reasoning to graphs? How?

There are many theoretical works into sparsifying graphs (spanners, spectral sparsifiers, cut sparsifiers, …). How to efficiently develop, use, and compare them, and which ones to select?



What should we pay attention to? (there is no “visual similarity” measure in this case…)

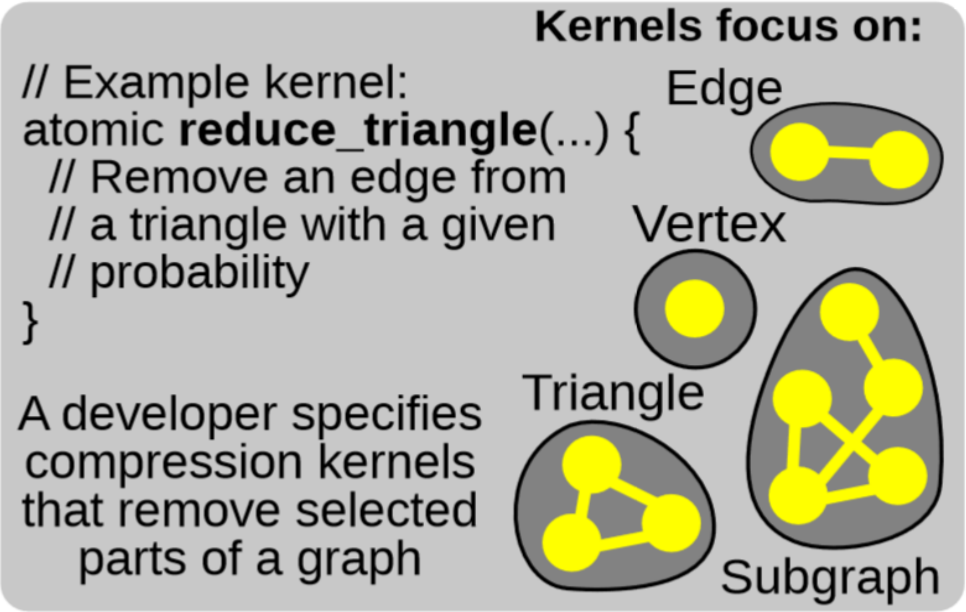
M. Besta et al.: “Slim Graph: Practical Lossy Graph Compression for Approximate Graph Processing, Storage, and Analytics”, ACM/IEEE SC’19



M. Besta et al.: “Slim Graph: Practical Lossy Graph Compression for Approximate Graph Processing, Storage, and Analytics”, ACM/IEEE SC’19

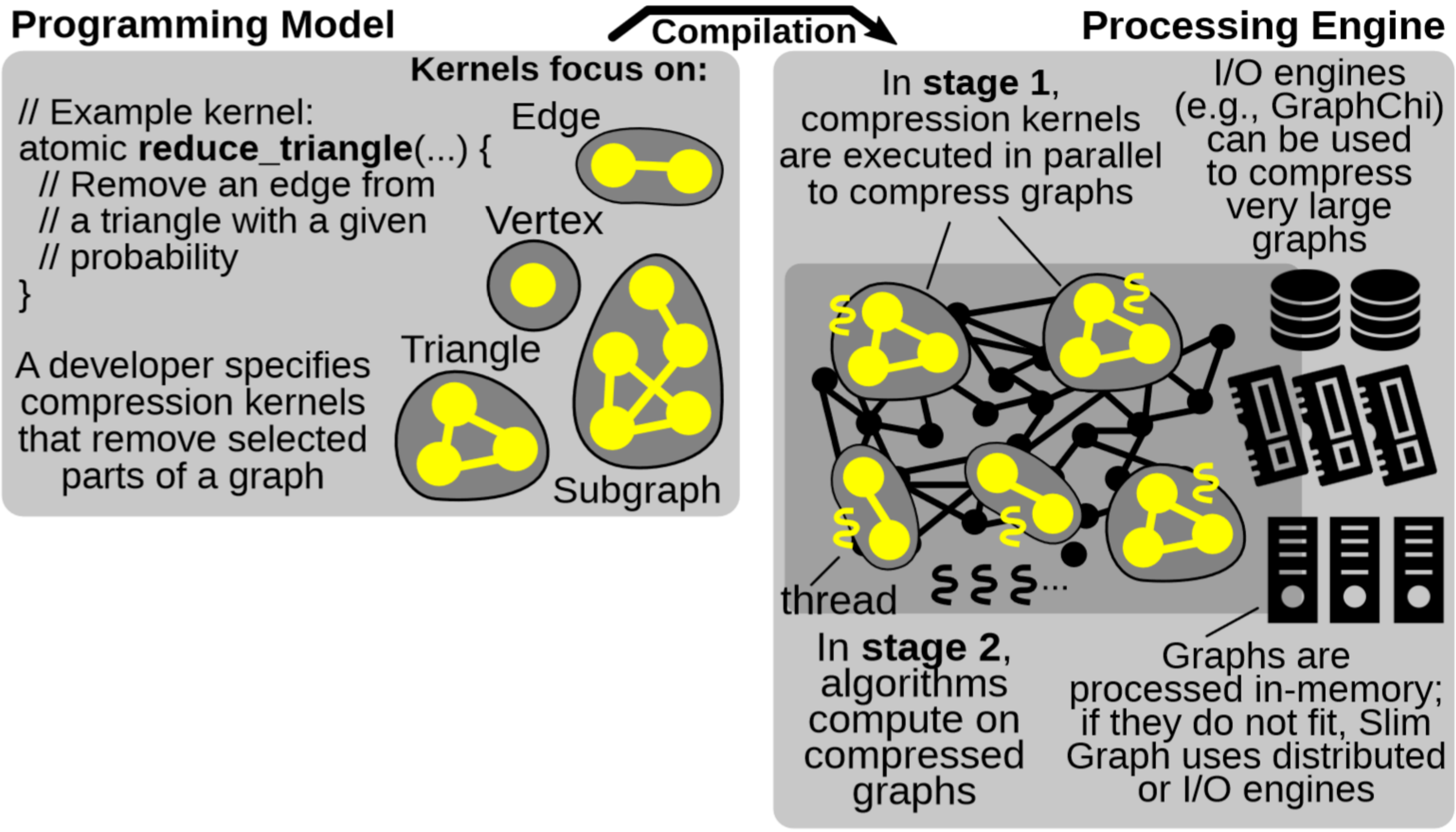
Slim Graph Overview

Programming Model



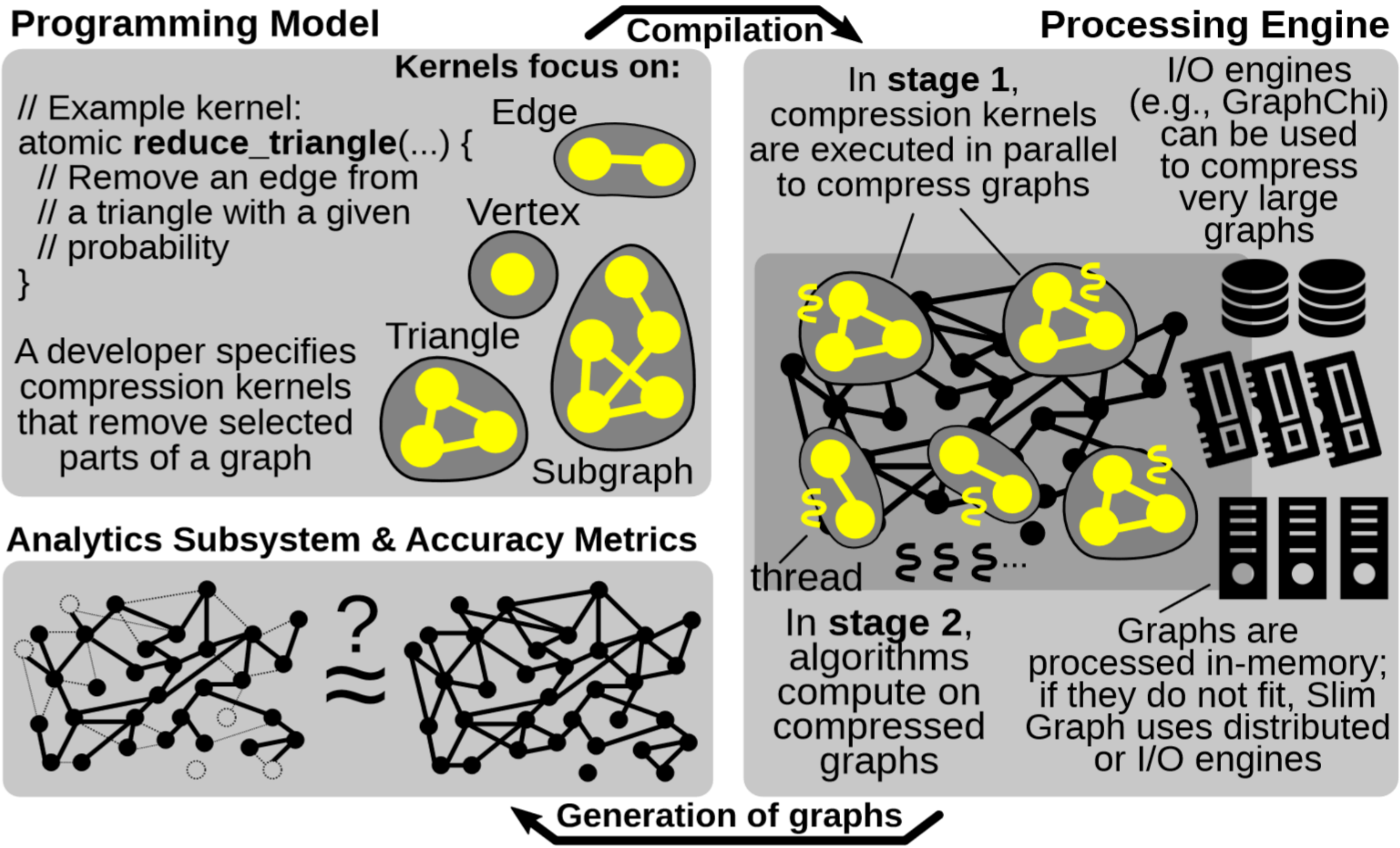
M. Besta et al.: "Slim Graph: Practical Lossy Graph Compression for Approximate Graph Processing, Storage, and Analytics", ACM/IEEE SC'19

Slim Graph Overview



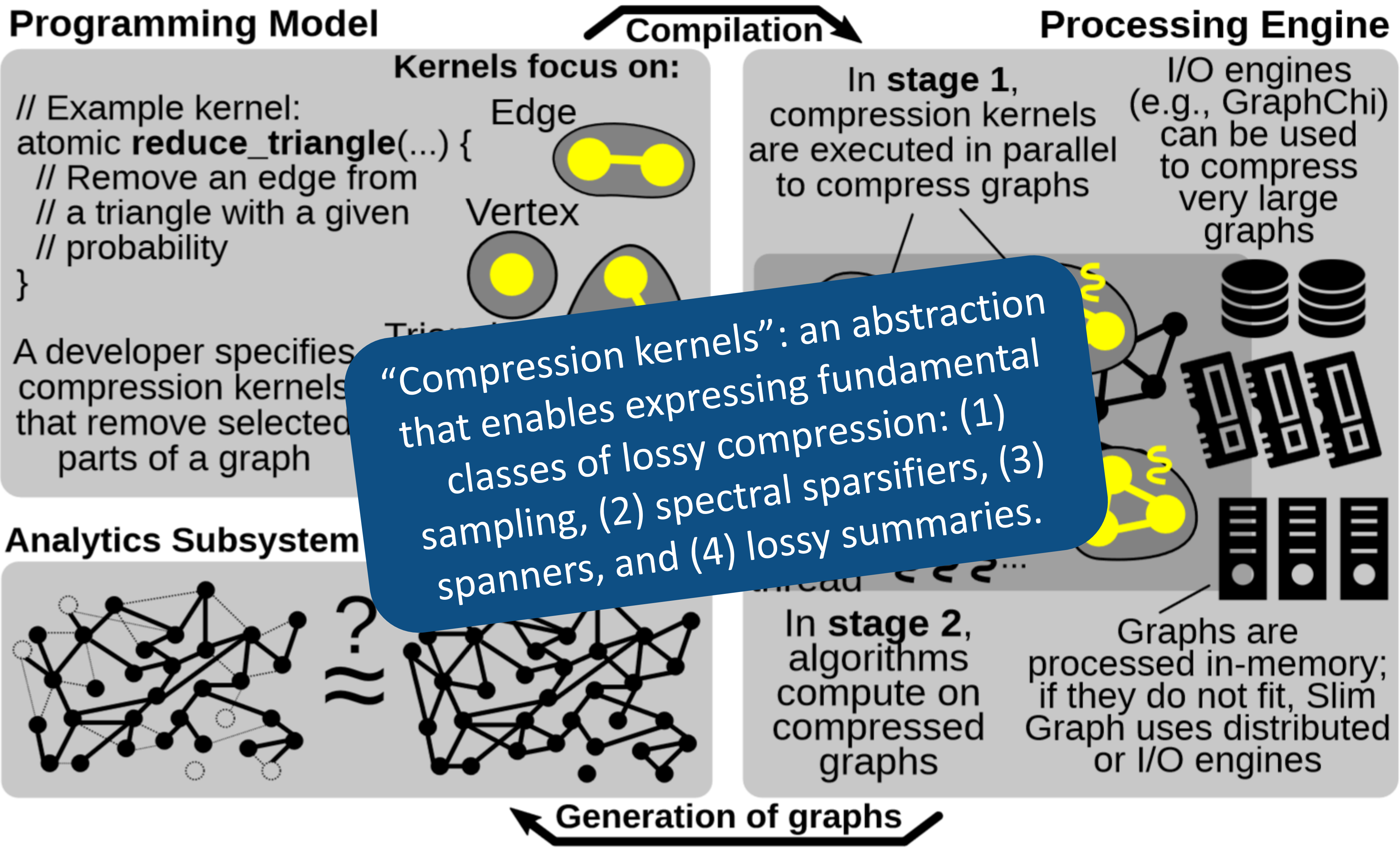
M. Besta et al.: "Slim Graph: Practical Lossy Graph Compression for Approximate Graph Processing, Storage, and Analytics", ACM/IEEE SC'19

Slim Graph Overview



M. Besta et al.: “Slim Graph: Practical Lossy Graph Compression for Approximate Graph Processing, Storage, and Analytics”, ACM/IEEE SC’19

Slim Graph Overview



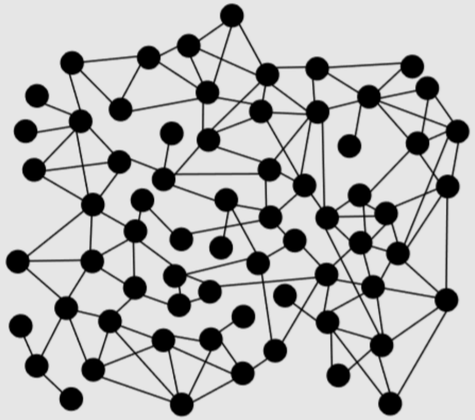
"Compression kernels": an abstraction that enables expressing fundamental classes of lossy compression: (1) sampling, (2) spectral sparsifiers, (3) spanners, and (4) lossy summaries.

M. Besta et al.: "Slim Graph: Practical Lossy Graph Compression for Approximate Graph Processing, Storage, and Analytics", ACM/IEEE SC'19

Example Compression Kernel: Triangle Reduction

Input graph:

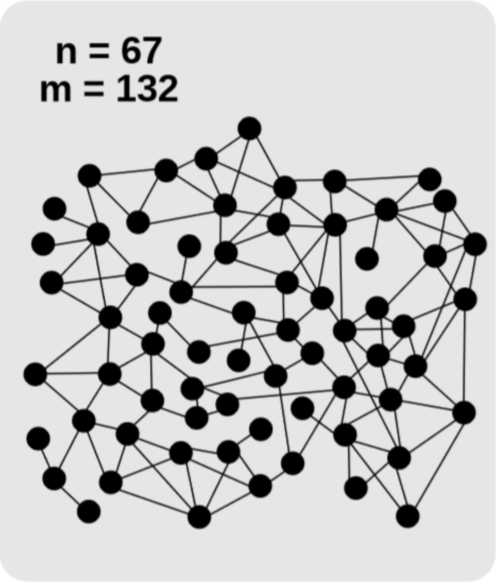
n = 67
m = 132



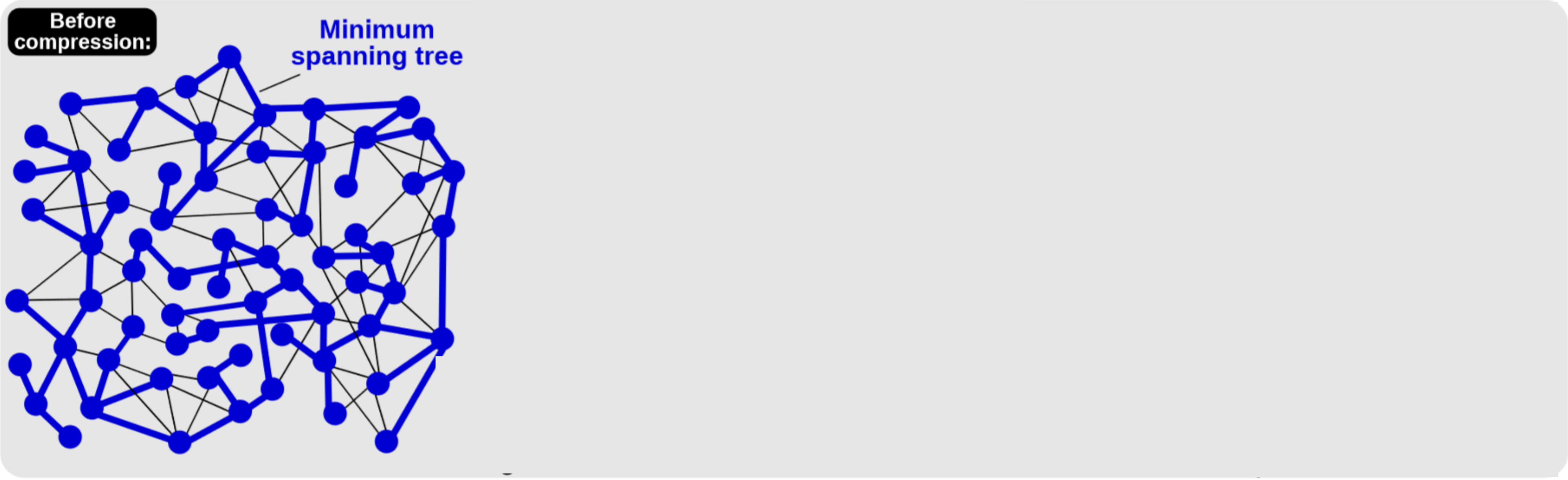
(§ 4.3) Triangle Compression Kernels (implementing Triangle Reduction, a novel graph compression method proposed in this work):

Example Compression Kernel: Triangle Reduction

Input graph:

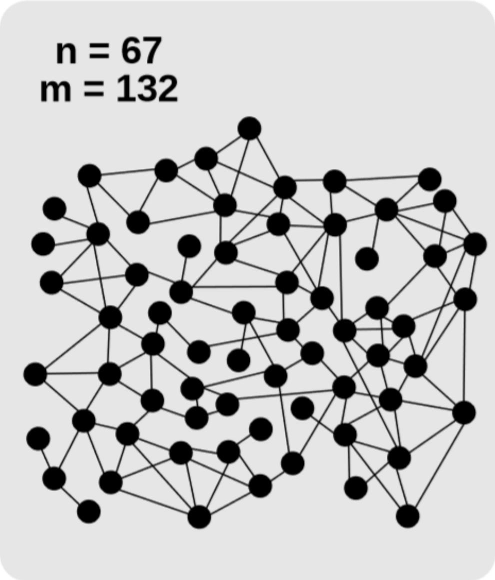


(§ 4.3) Triangle Compression Kernels (implementing Triangle Reduction, a novel graph compression method proposed in this work):

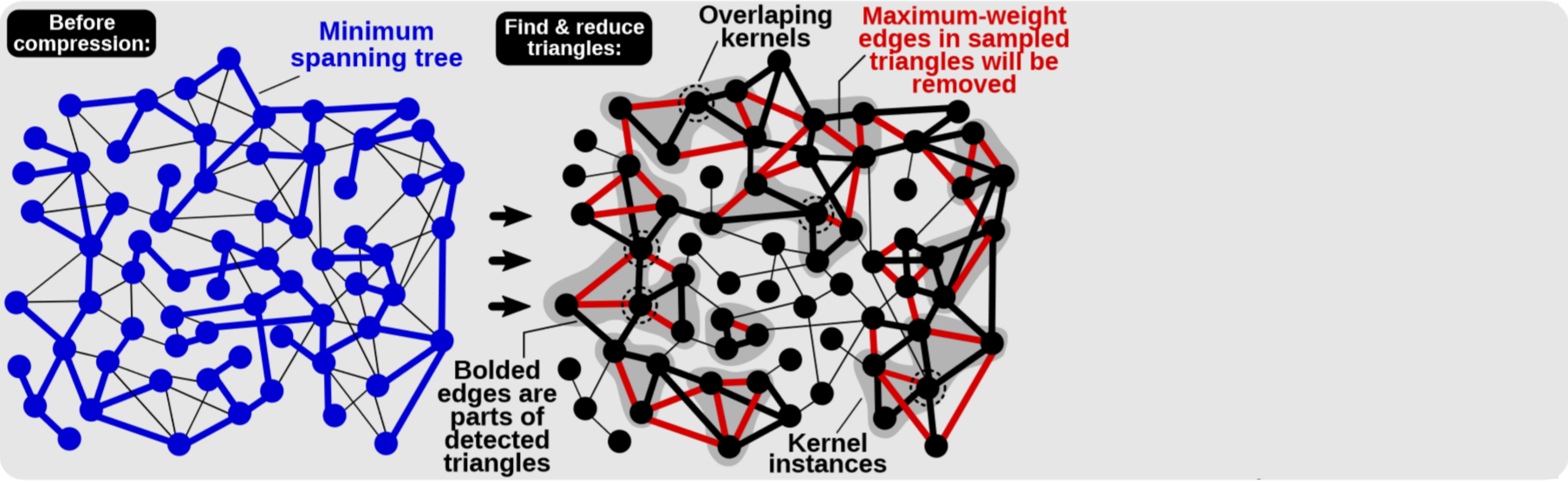


Example Compression Kernel: Triangle Reduction

Input graph:

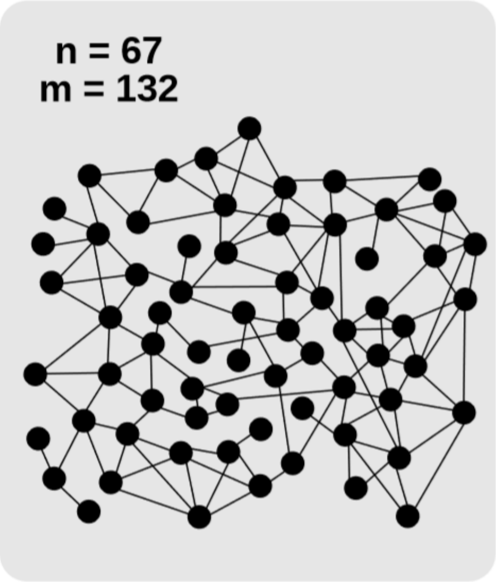


(§ 4.3) Triangle Compression Kernels (implementing Triangle Reduction, a novel graph compression method proposed in this work):

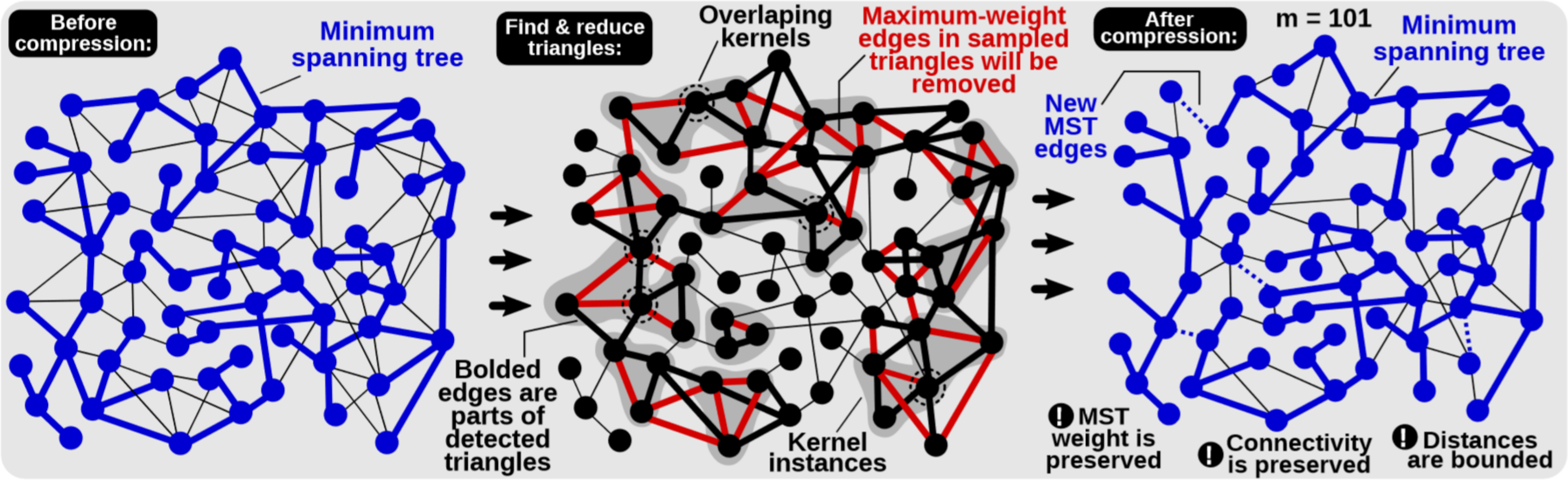


Example Compression Kernel: Triangle Reduction

Input graph:

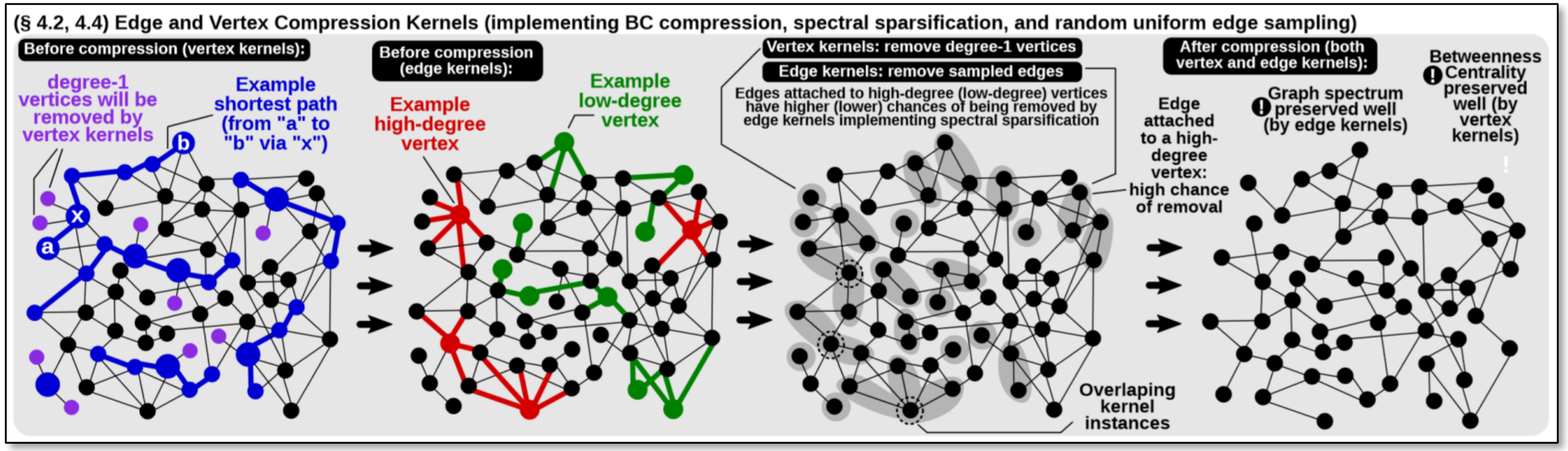


(§ 4.3) Triangle Compression Kernels (implementing Triangle Reduction, a novel graph compression method proposed in this work):

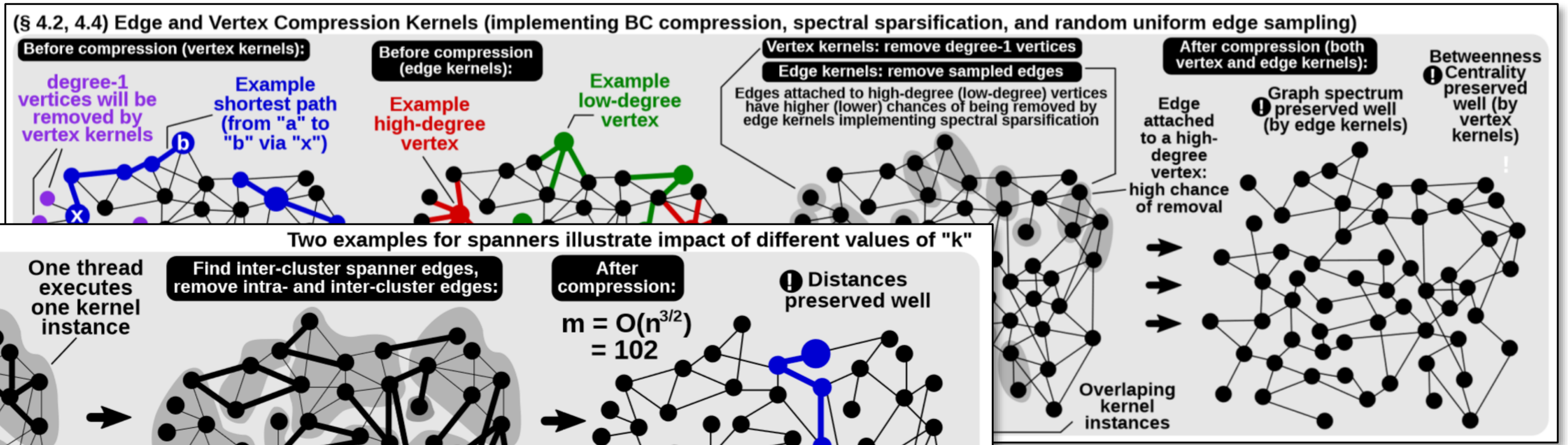


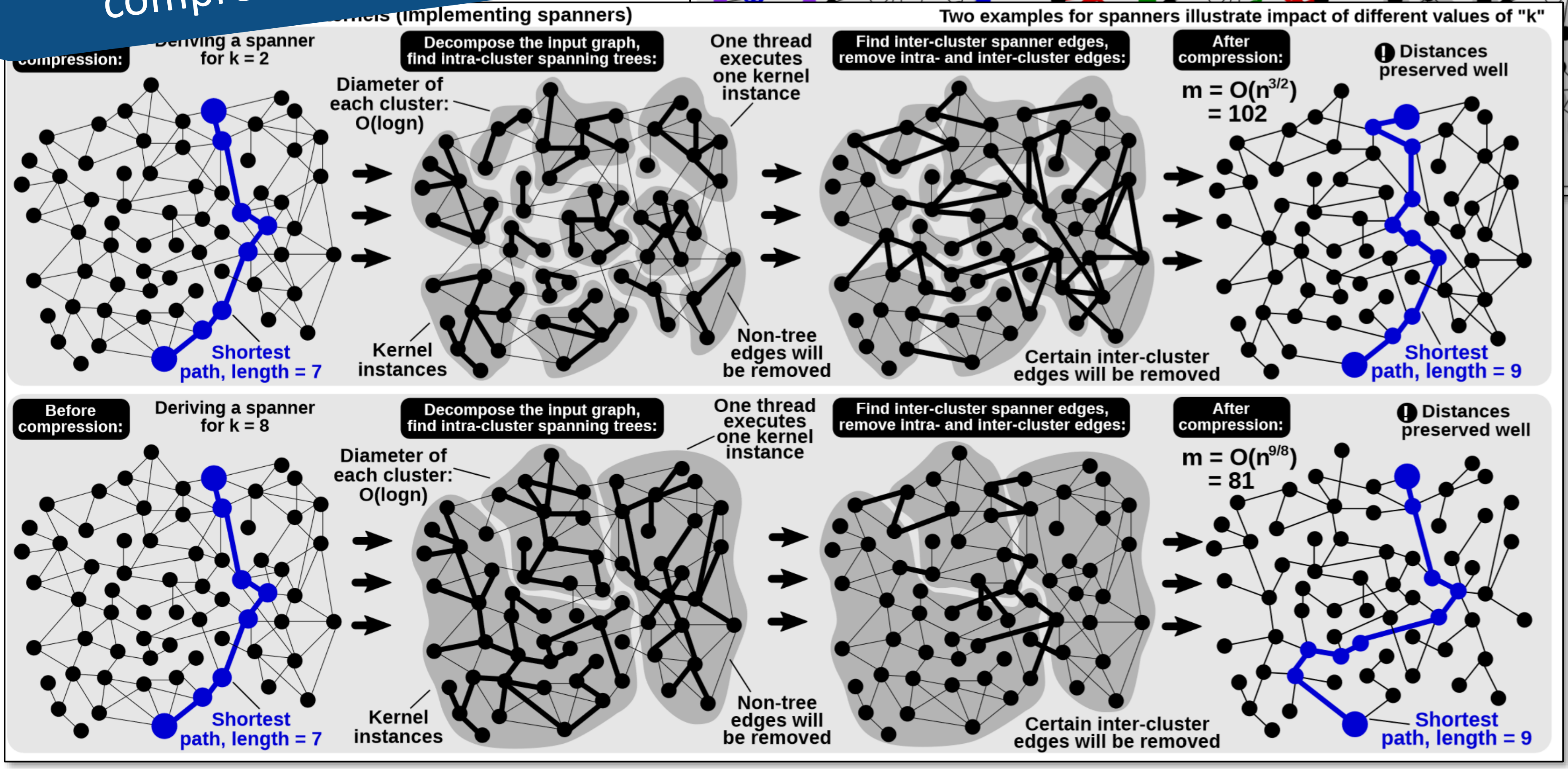
Other types of compression kernels enable expressing and implementing other lossy graph compression schemes

Other types of compression kernels enable expressing and implementing other lossy graph compression schemes

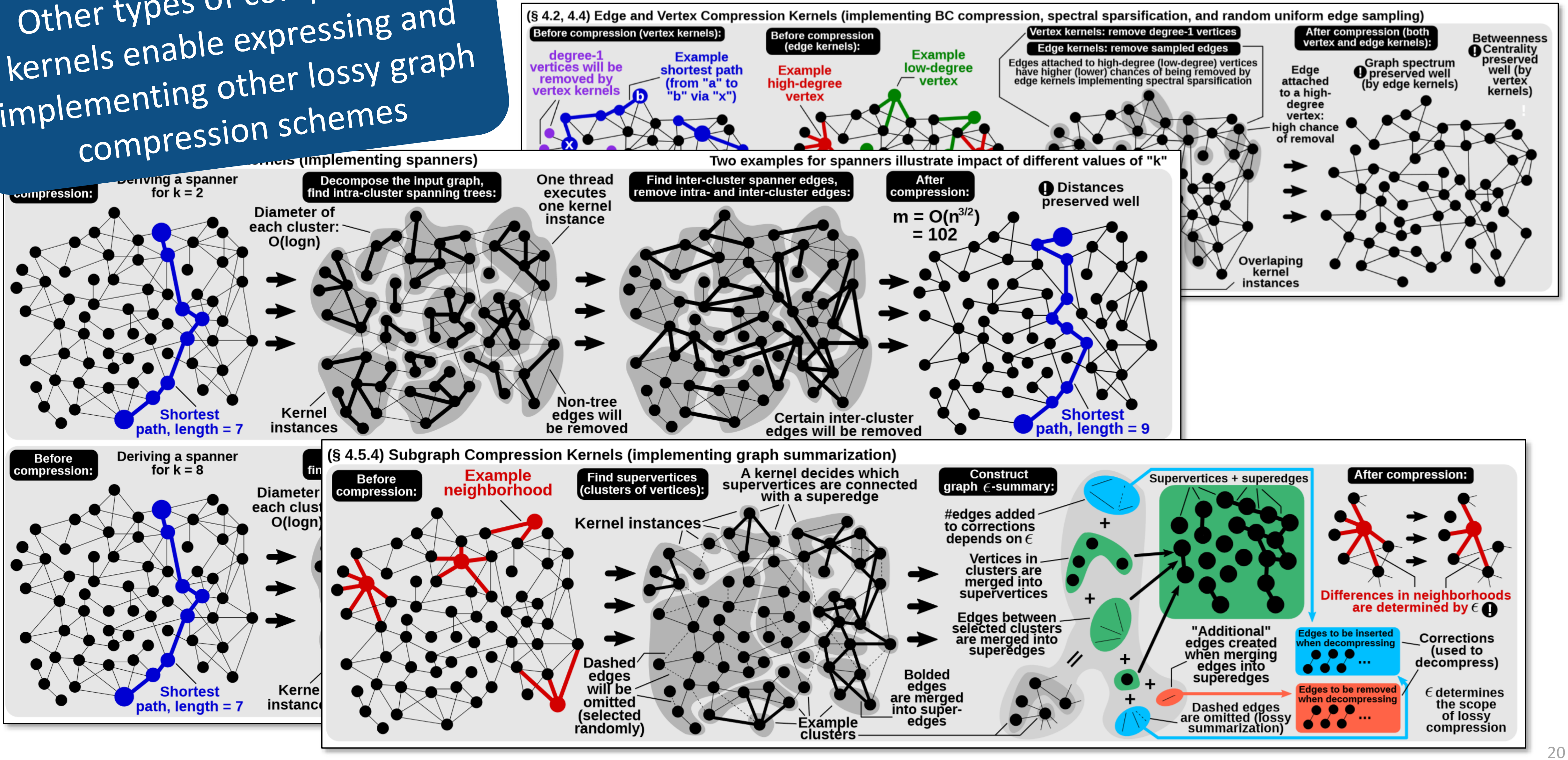


Other types of compression kernels enable expressing and implementing other lossy graph compression schemes

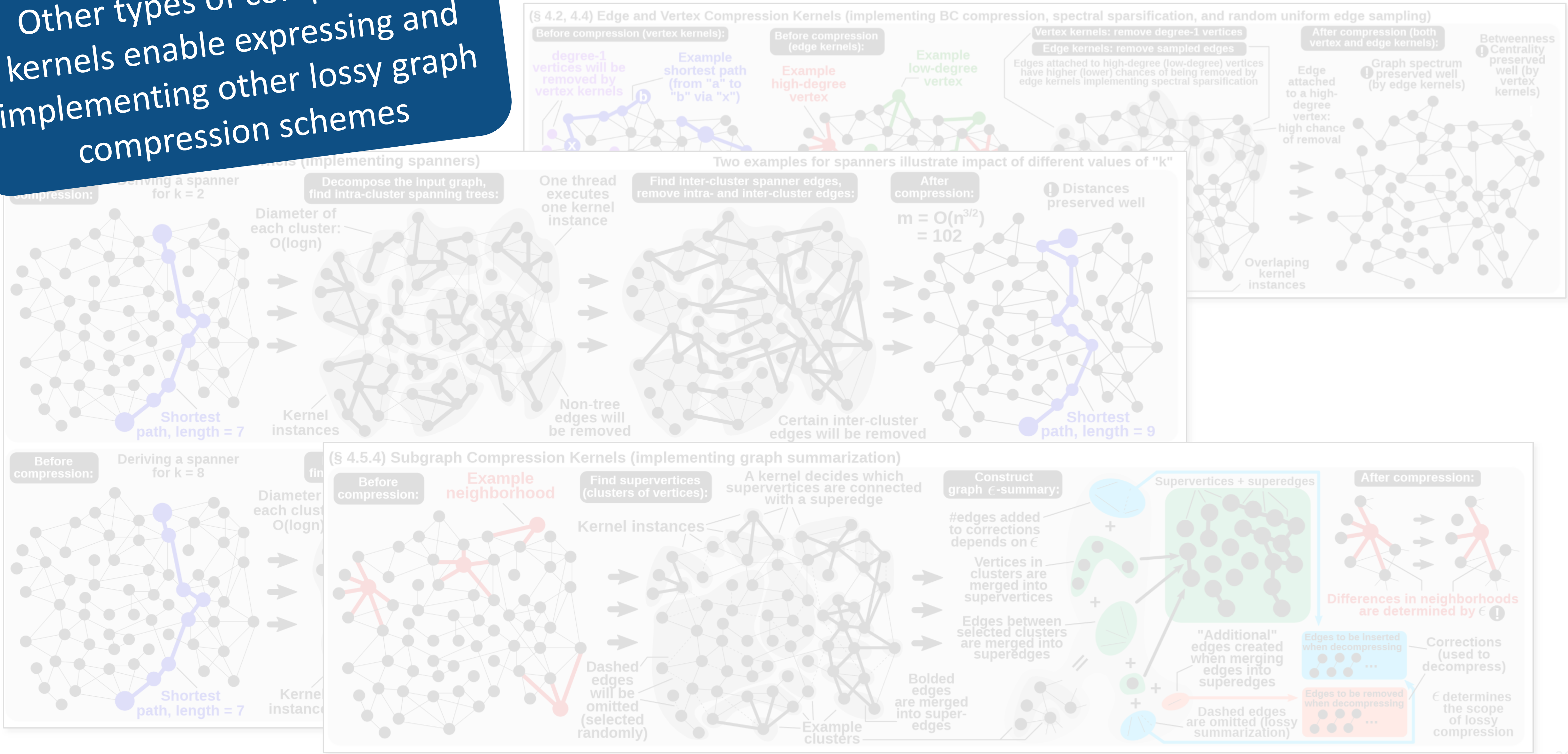




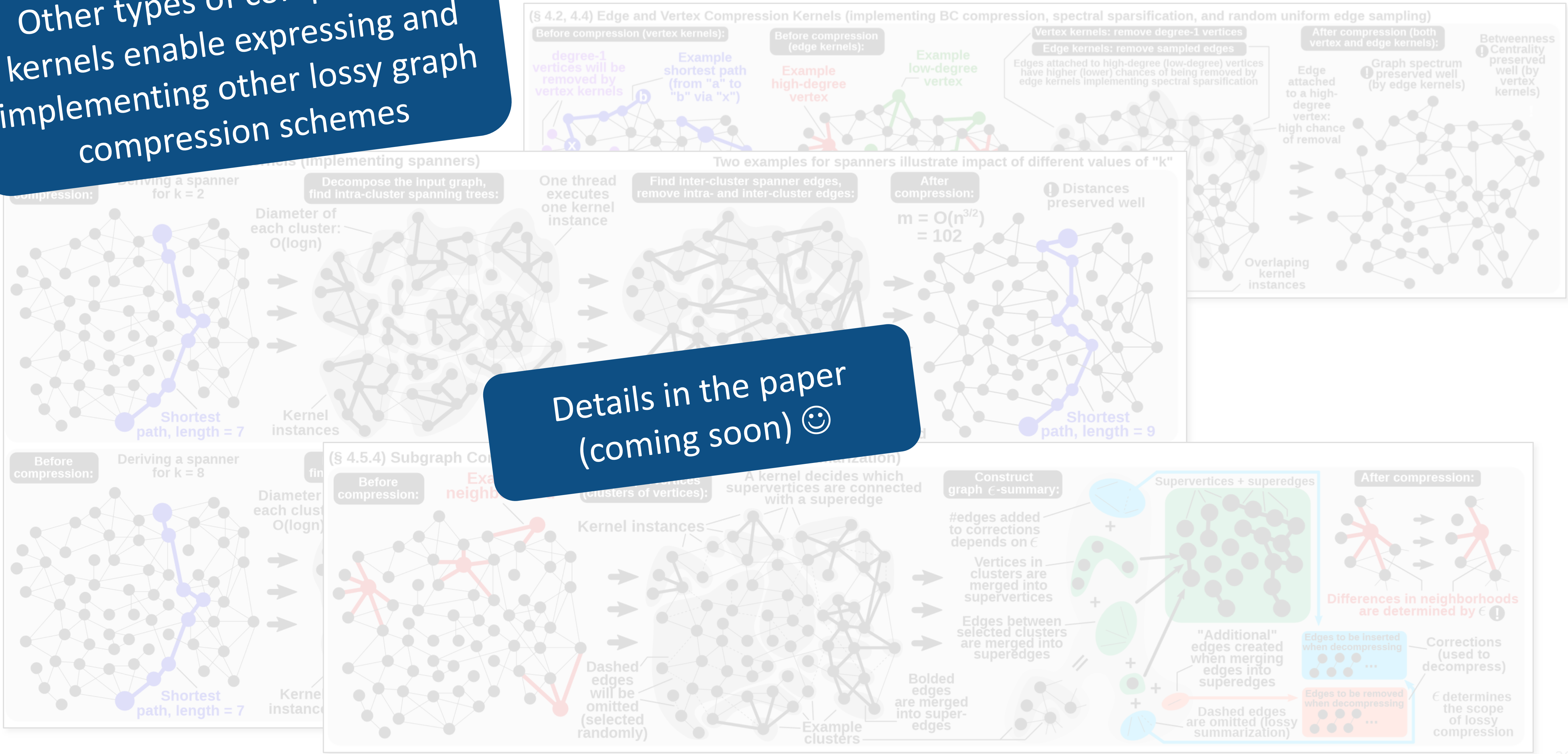
Other types of compression kernels enable expressing and implementing other lossy graph compression schemes



Other types of compression kernels enable expressing and implementing other lossy graph compression schemes



Other types of compression kernels enable expressing and implementing other lossy graph compression schemes



Details in the paper (coming soon) ☺

Storage Reductions vs. Speedups vs. Accuracy Loss [vs. Compression Overhead]

Various real-world graphs are used

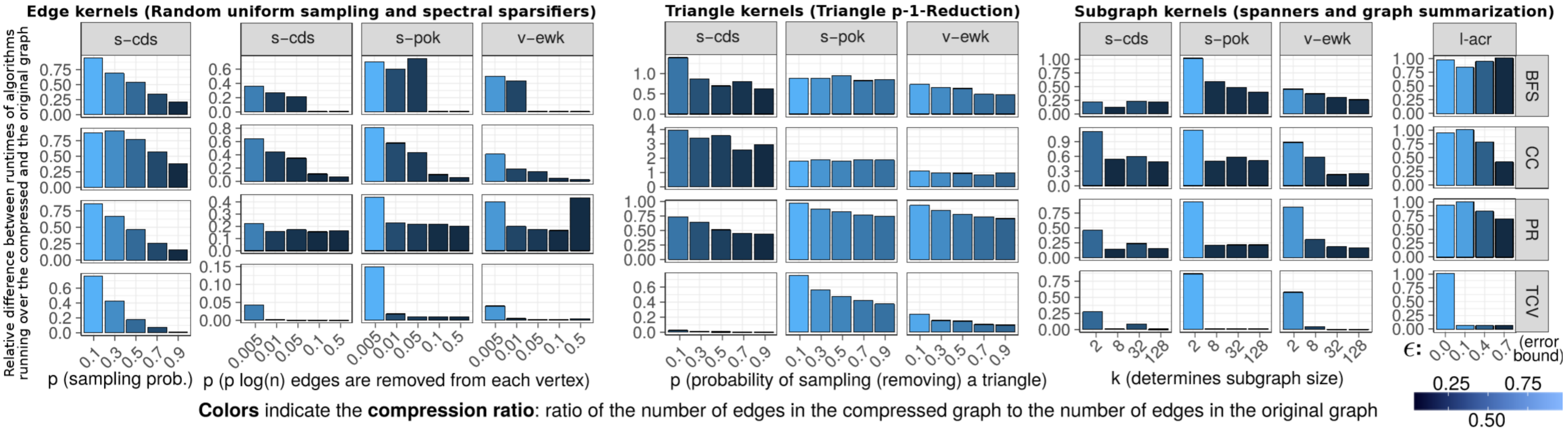


Figure 4: Analysis of **storage and performance** tradeoffs of various lossy compression schemes implemented in Slim Graph (**when varying compression parameters**).

Storage Reductions vs. Speedups vs. Accuracy Loss [vs. Compression Overhead]

Various real-world graphs are used

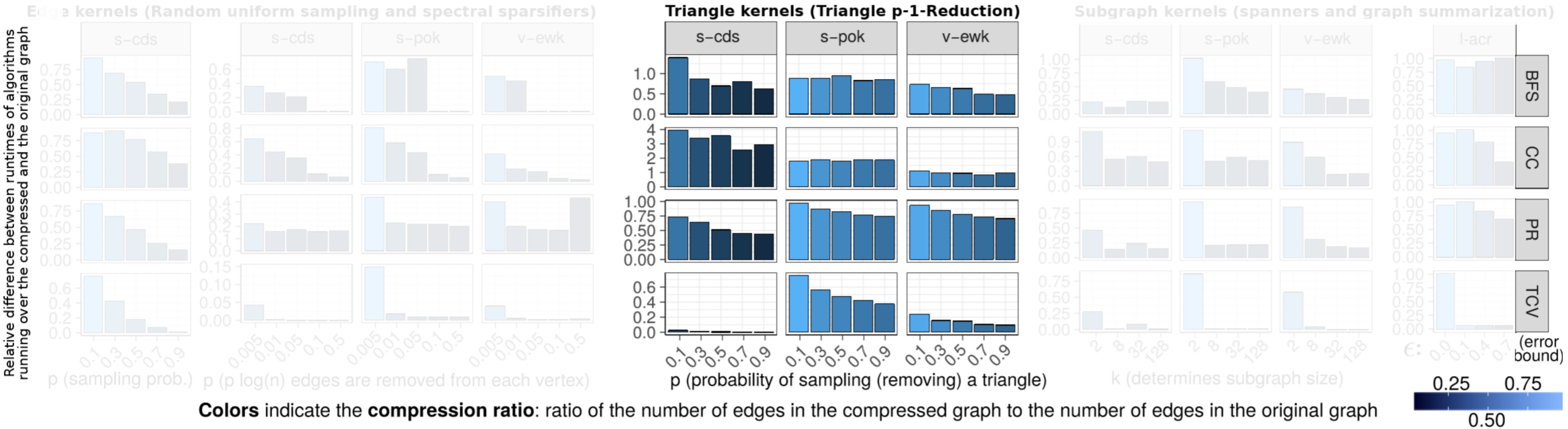
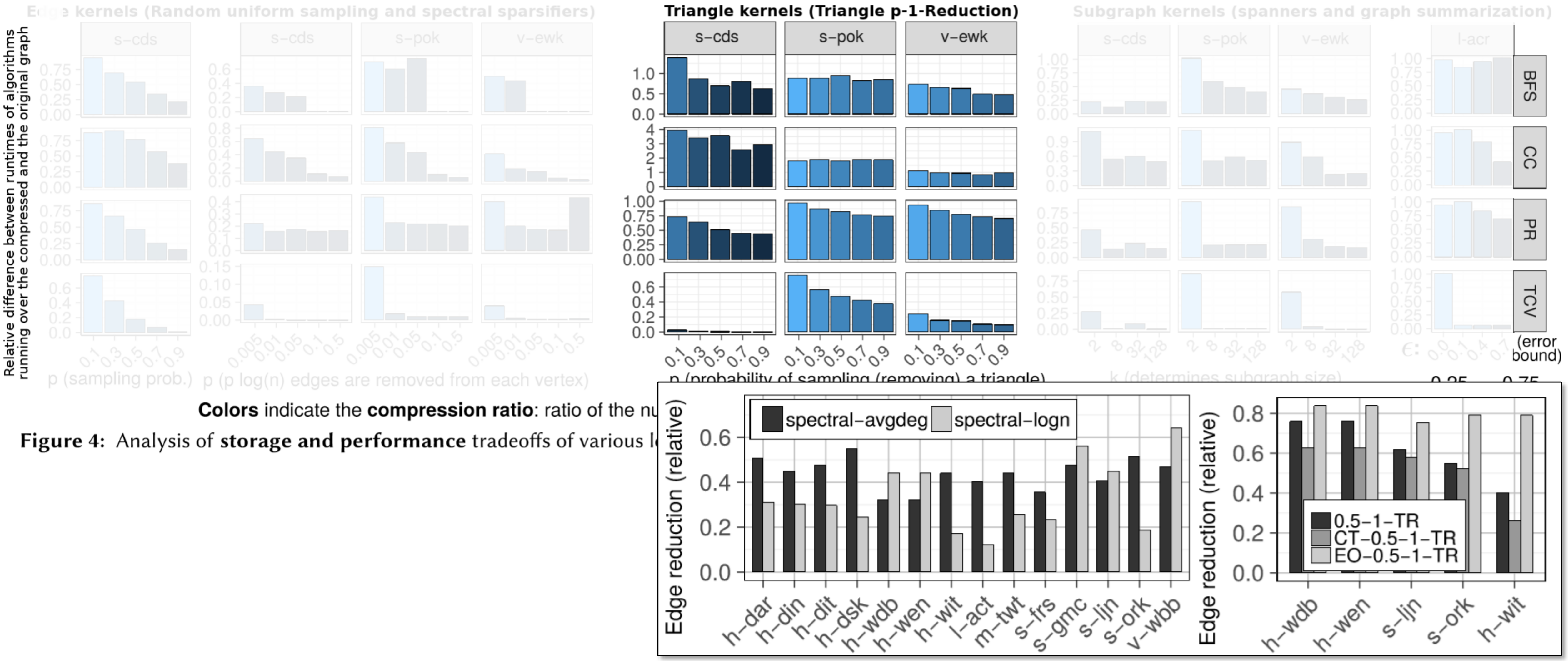


Figure 4: Analysis of **storage and performance** tradeoffs of various lossy compression schemes implemented in Slim Graph (**when varying compression parameters**).

Storage Reductions vs. Speedups vs. Accuracy Loss [vs. Compression Overhead]

Various real-world graphs are used

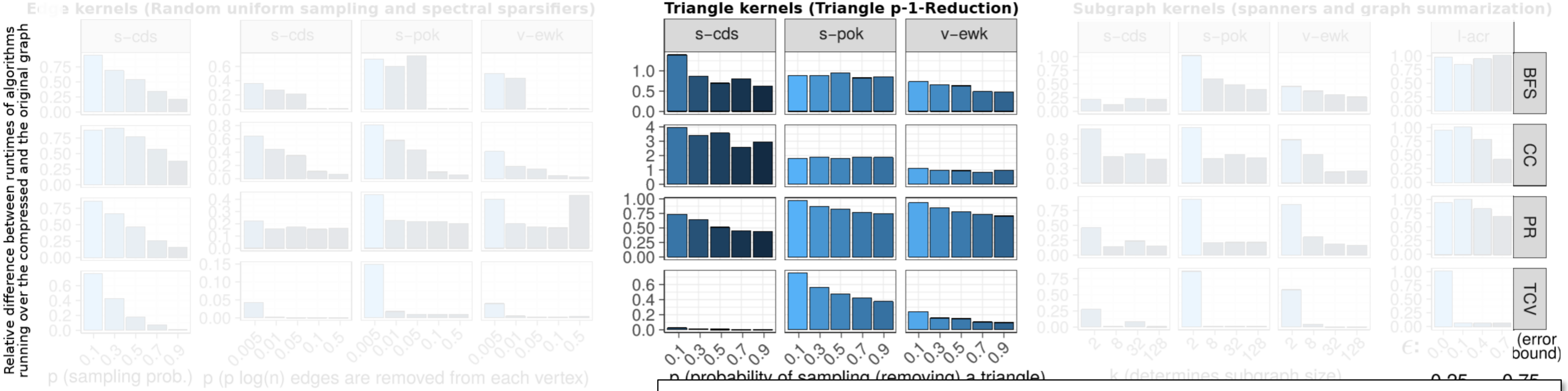


Colors indicate the **compression ratio**: ratio of the nu

Figure 4: Analysis of **storage and performance** tradeoffs of various l

Storage Reductions vs. Speedups vs. Accuracy Loss [vs. Compression Overhead]

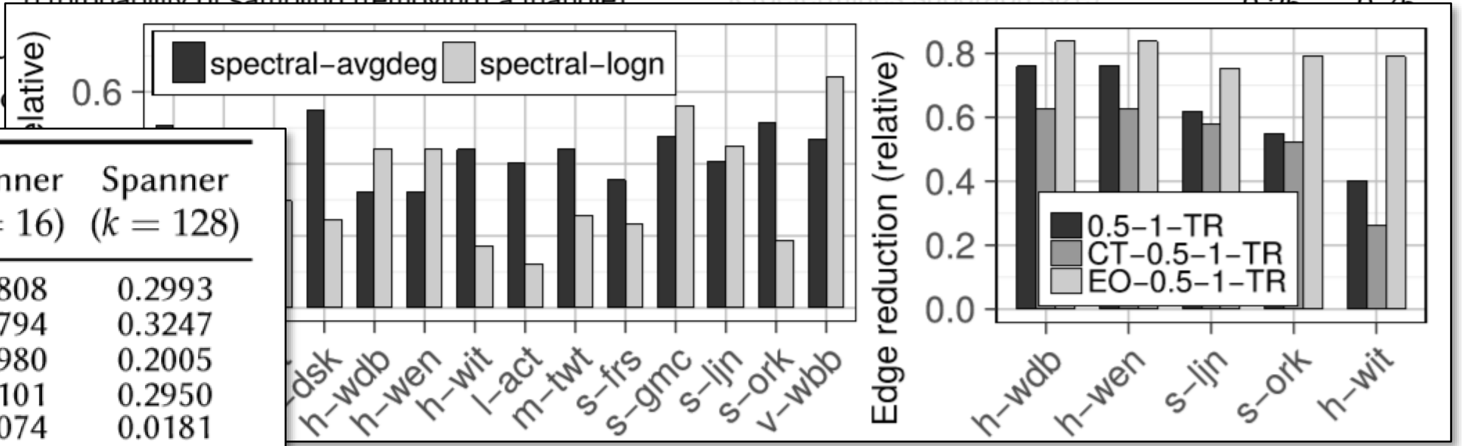
Various real-world graphs are used



Colors indicate the **compression ratio**: ratio of the nu

Figure 4: Analysis of **storage and performance** tradeoffs of various l

Graph	EO 0.8-1-TR	EO 1.0-1-TR	Uniform ($p = 0.2$)	Uniform ($p = 0.5$)	Spanner ($k = 2$)	Spanner ($k = 16$)	Spanner ($k = 128$)
s-you	0.0121	0.0167	0.1932	0.6019	0.0054	0.2808	0.2993
h-hud	0.0187	0.0271	0.0477	0.1633	0.0340	0.2794	0.3247
il-dbl	0.0459	0.0674	0.0749	0.2929	0.0080	0.1980	0.2005
v-skt	0.0410	0.0643	0.0674	0.2695	0.0311	0.1101	0.2950
v-usa	0.0089	0.0100	0.1392	0.5945	0.0000	0.0074	0.0181



Storage Reductions vs. Speedups vs. Accuracy Loss [vs. Compression Overhead]

Various real-world graphs are used

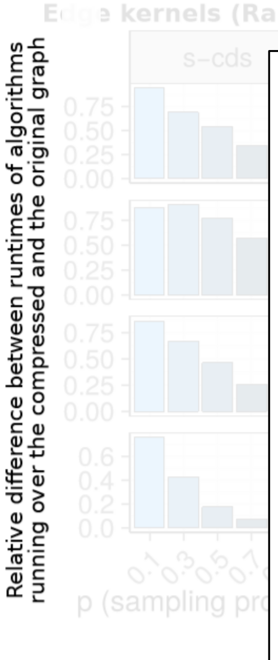
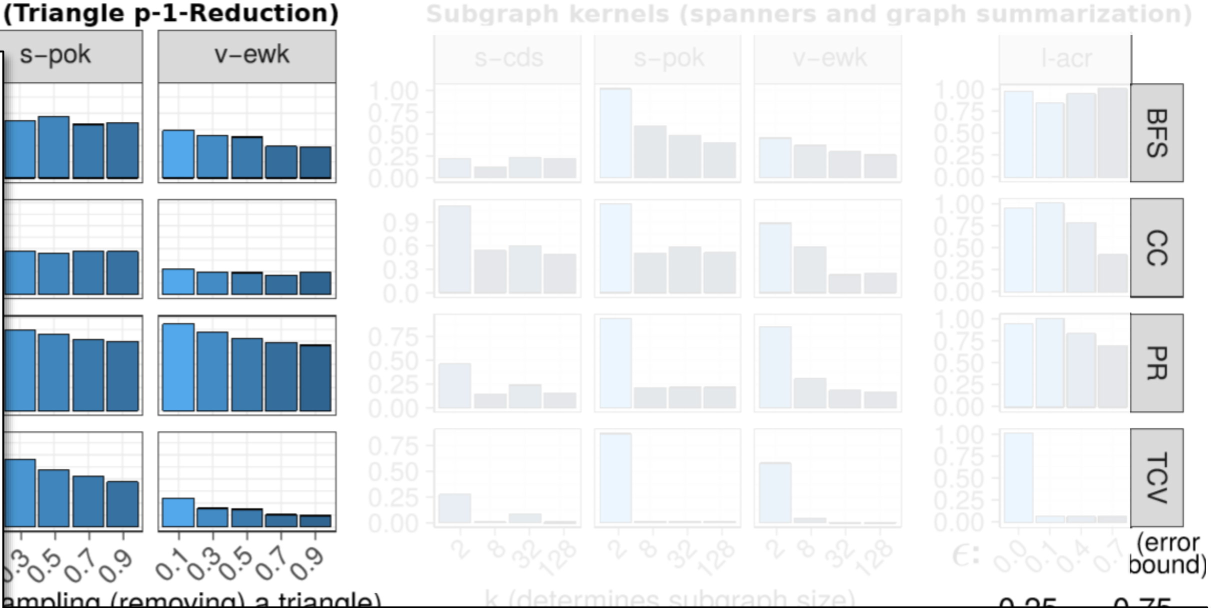



Figure 4: Analy

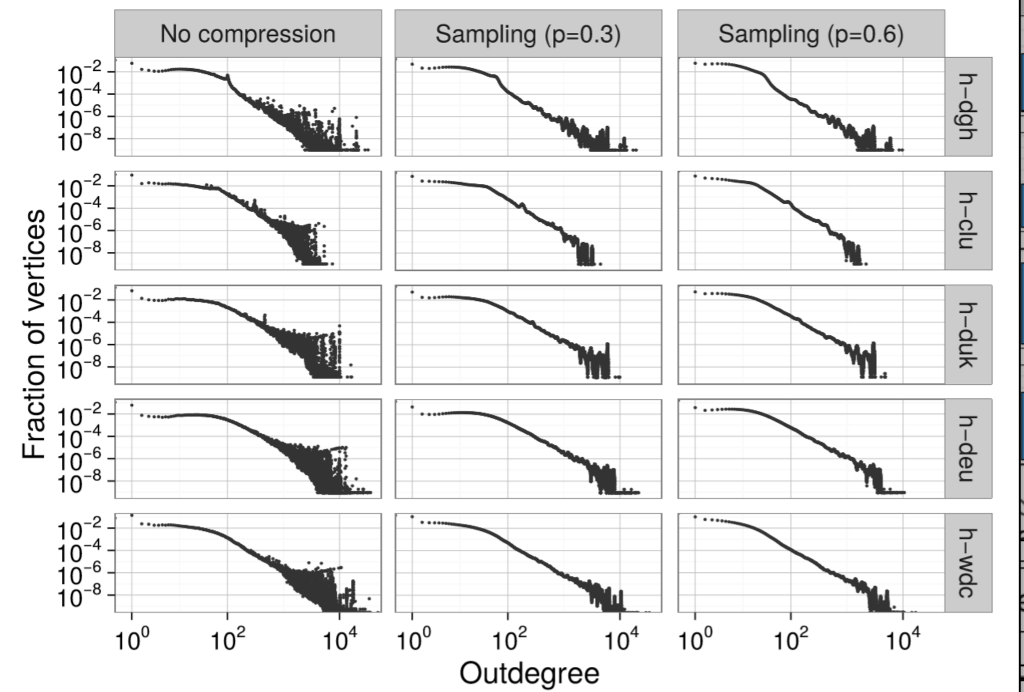
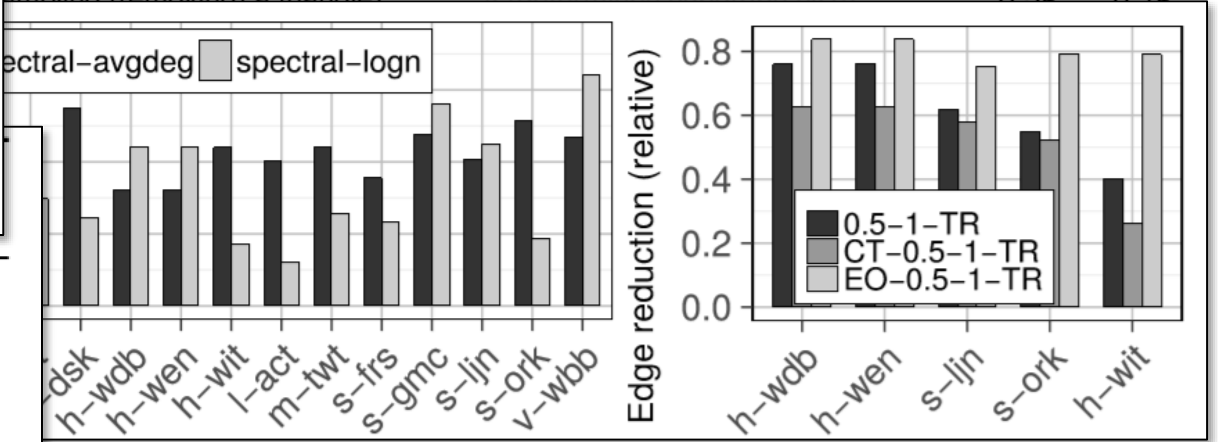


Figure 7: (Accuracy) Impact of random uniform sampling on the degree distribution of large graphs (the largest, h-wdc, has ≈128B edges).

Graph	EO 0.8-1-						
s-you	0.0121	0.0167	0.1932	0.6019	0.0054	0.2808	0.2993
h-hud	0.0187	0.0271	0.0477	0.1633	0.0340	0.2794	0.3247
il-dbl	0.0459	0.0674	0.0749	0.2929	0.0080	0.1980	0.2005
v-skt	0.0410	0.0643	0.0674	0.2695	0.0311	0.1101	0.2950
v-usa	0.0089	0.0100	0.1392	0.5945	0.0000	0.0074	0.0181



Storage Reductions vs. Speedups vs. Accuracy Loss [vs. Compression Overhead]

Various real-world graphs are used

	$\lvert V\rvert$	$\lvert E\rvert$	Shortest s-t path length	Average path length	Diameter	Average degree	Maximum degree	#Triangles	#Connected components	Chromatic number	Max. indep. set size	Max. cardinal. matching size
Original graph	n	m	$\mathcal{P}$	$\overline{\mathcal{P}}$	D	$\bar{d}$	d	T	$\mathcal{C}$	C_R	$\widehat{I}_S$	$\widehat{M}_C$
Lossy ϵ-summary	n	$m \pm 2\epsilon m$	$1,\dots,\infty$	$1,\dots,\infty$	$1,\dots,\infty$	$\bar{d} \pm \epsilon\bar{d}$	$d \pm \epsilon d$	$T \pm 2\epsilon m$	$\mathcal{C} \pm 2\epsilon m$	$C_R \pm 2\epsilon m$	$\widehat{I}_S \pm 2\epsilon m$	$\widehat{M}_C \pm 2\epsilon m$
Simple p–sampling	n	$(1-p)m$	∞	∞	∞	$(1-p)\bar{d}$	$(1-p)d$	$(1-p^3)T$	$\leq \mathcal{C} + pm$	$\geq C_R - pm$	$\leq \widehat{I}_S + pm$	$\geq \widehat{M}_C - pm$
Spectral ϵ-sparsifier	n	$\tilde{O}(n/\epsilon^2)$	$\leq n$	$\leq n$	$\leq n$	$\tilde{O}(1/\epsilon^2)$	$\geq d/2(1+\epsilon)$	$\tilde{O}(n^{3/2}/\epsilon^3)$	$\overset{w.h.p.}{=} \mathcal{C}$	$\leq d/2(1+\epsilon)$	$\geq 2(1+\epsilon)n/d$	≥ 0
$O(k)$–spanner	n	$O(n^{1+1/k})$	$O(k\mathcal{P})$	$O(k\overline{\mathcal{P}})$	$O(kD)$	$O(n^{1/k})$	$\leq d$	$O(n^{1+2/k})$	$\mathcal{C}$	$O(n^{1/k}\log n)$	$\Omega\left(\frac{n^{1-1/k}}{\log n}\right)$	≥ 0
EO p–1–Triangle Red.	n	$\leq m - \frac{pT}{3d}$	$\overset{w.h.p.}{\leq} \mathcal{P} + p\mathcal{P}$	$\leq \overline{\mathcal{P}} + \frac{pT}{n(n-1)}$	$\overset{w.h.p.}{\leq} D + pD$	$\leq \bar{d} - \frac{pT}{dn}$	$\geq d/2$	$\leq (1 - \frac{p}{d})T$	$\mathcal{C}$	$\geq C_R - pT$	$\leq \widehat{I}_S + pT$	$\geq \widehat{M}_C/2$
remove k deg-1 vertices	$n-k$	$m-k$	$\mathcal{P}$	$\geq \overline{\mathcal{P}} - \frac{kD}{n}$	$\geq D-2$	$\geq \bar{d} - \frac{k}{n}$	d	T	$\mathcal{C}$	C_R	$\geq \widehat{I}_S - k$	$\geq \widehat{M}_C - k$

Table 3: The impact of various compression schemes on the outcome of selected graph algorithms. Bounds that do not include inequalities hold deterministically. If not otherwise stated, the other bounds hold in expectation. Bounds annotated with w.h.p. hold w.h.p. (if the involved quantities are large enough). Note that since the listed compression schemes (except the scheme where we remove the degree 1 vertices) return a subgraph of the original graph, m, C_R, $\bar{d}$, d, T, and $\widehat{M}_C$ never increase. Moreover, $\mathcal{P}$, $\overline{\mathcal{P}}$, D, $\mathcal{C}$, and $\widehat{I}_S$ never decrease during compression. ϵ is a parameter that controls how well a spectral sparsifier approximates the original graph spectrum.

Figure 4: Analy

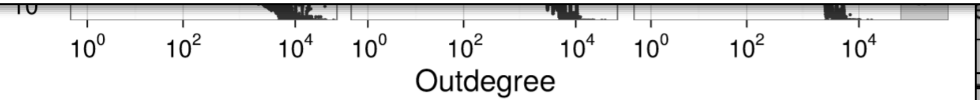
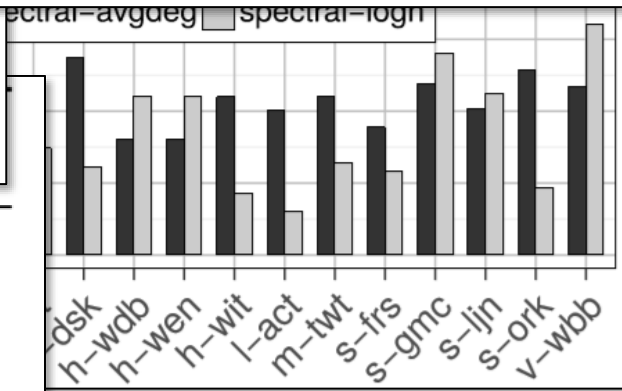
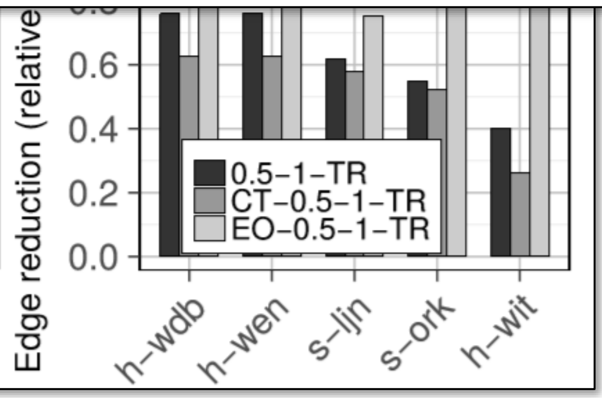


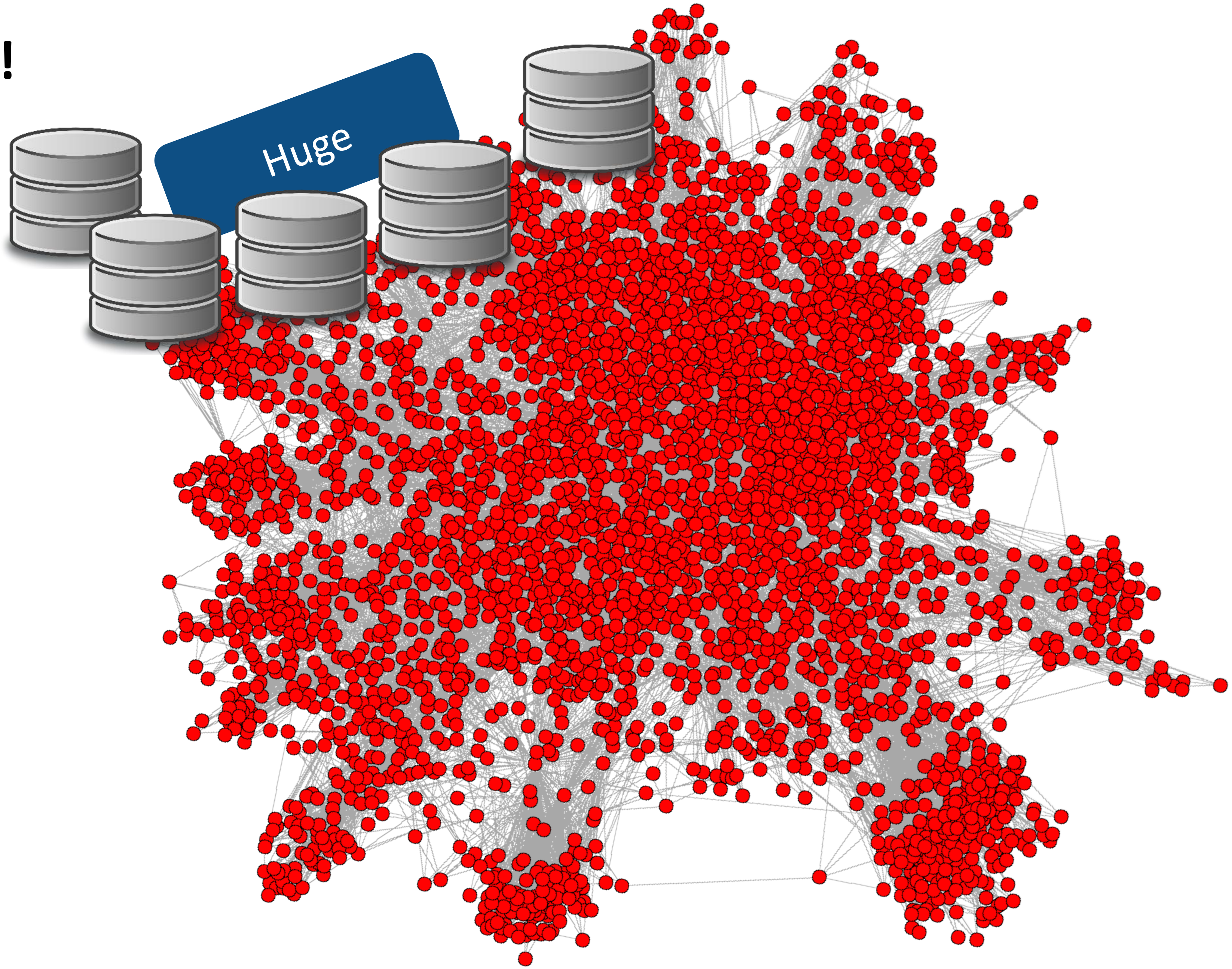
Figure 7: (Accuracy) Impact of random uniform sampling on the degree distribution of large graphs (the largest, h-wdc, has ≈128B edges).

Graph	EO 0.8-1-						
s-you	0.0121	0.0167	0.1932	0.6019	0.0054	0.2808	0.2993
h-hud	0.0187	0.0271	0.0477	0.1633	0.0340	0.2794	0.3247
il-dbl	0.0459	0.0674	0.0749	0.2929	0.0080	0.1980	0.2005
v-skt	0.0410	0.0643	0.0674	0.2695	0.0311	0.1101	0.2950
v-usa	0.0089	0.0100	0.1392	0.5945	0.0000	0.0074	0.0181

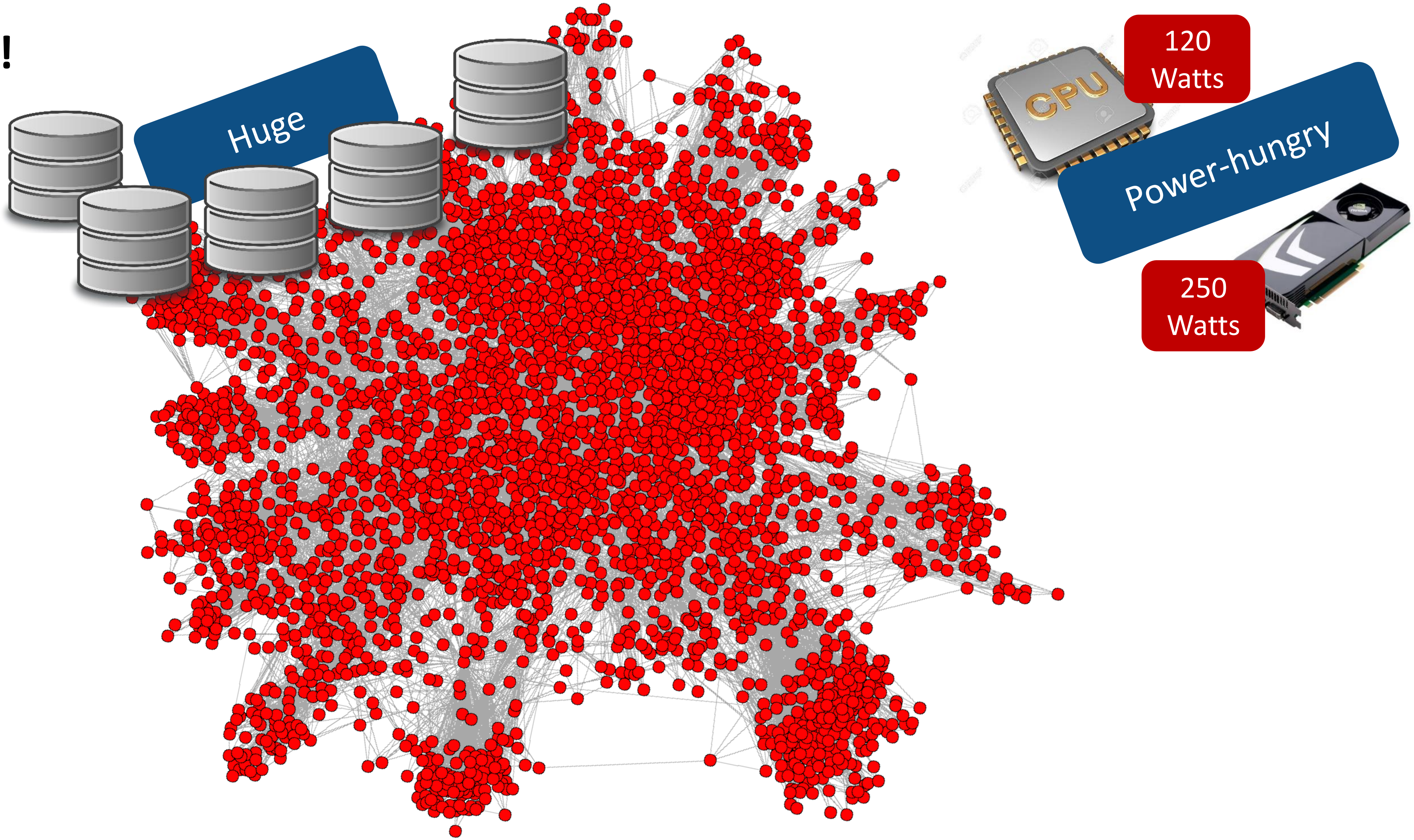




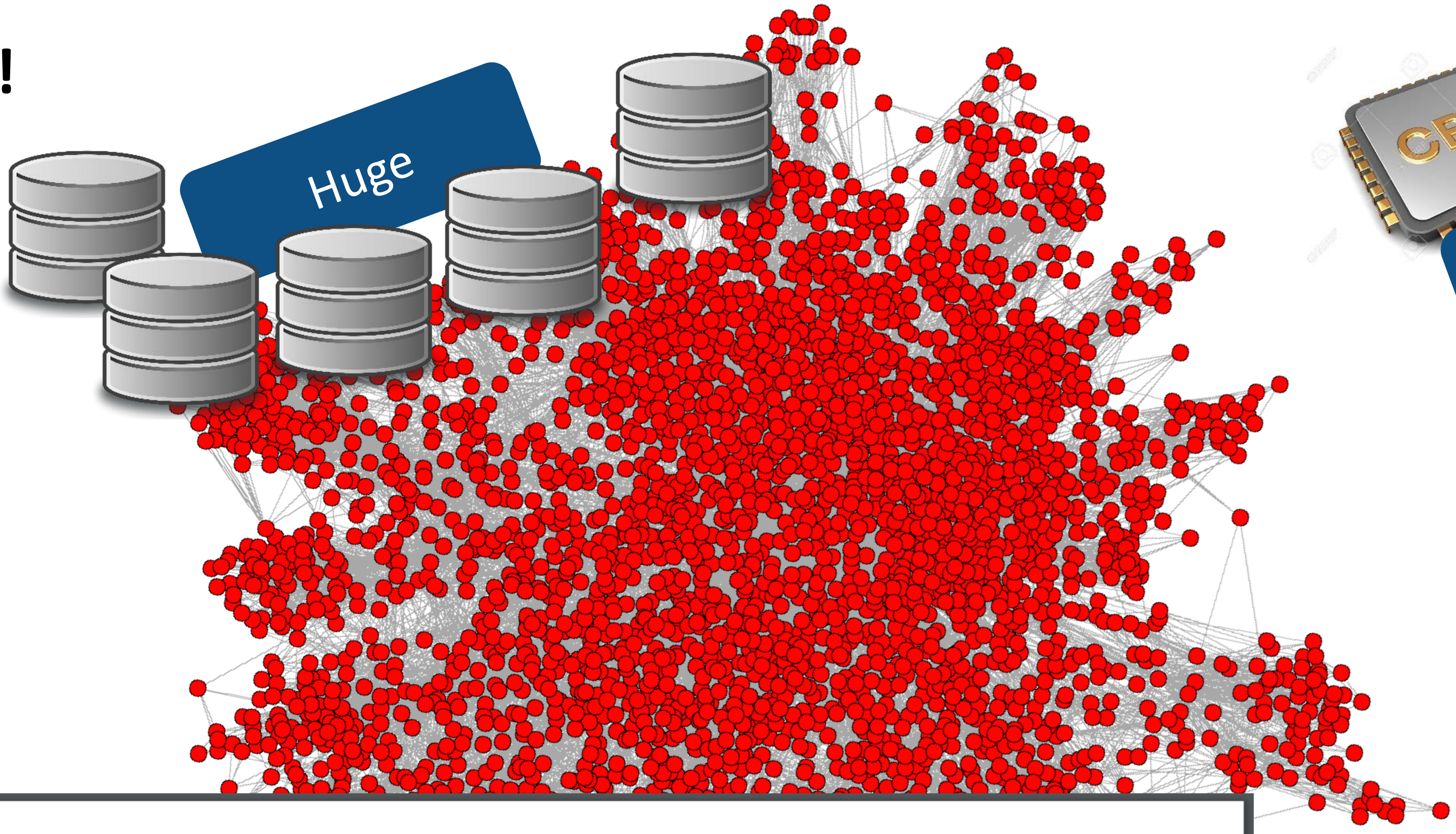
Problems!

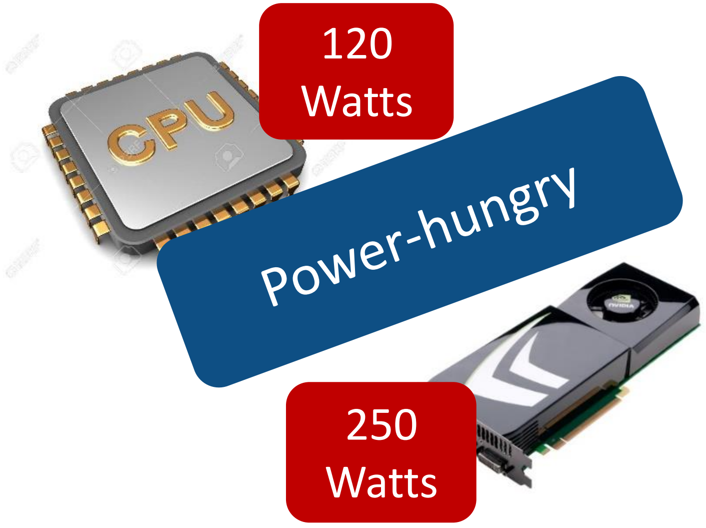


Problems!



Problems!





Substream-Centric Maximum Matchings on FPGA

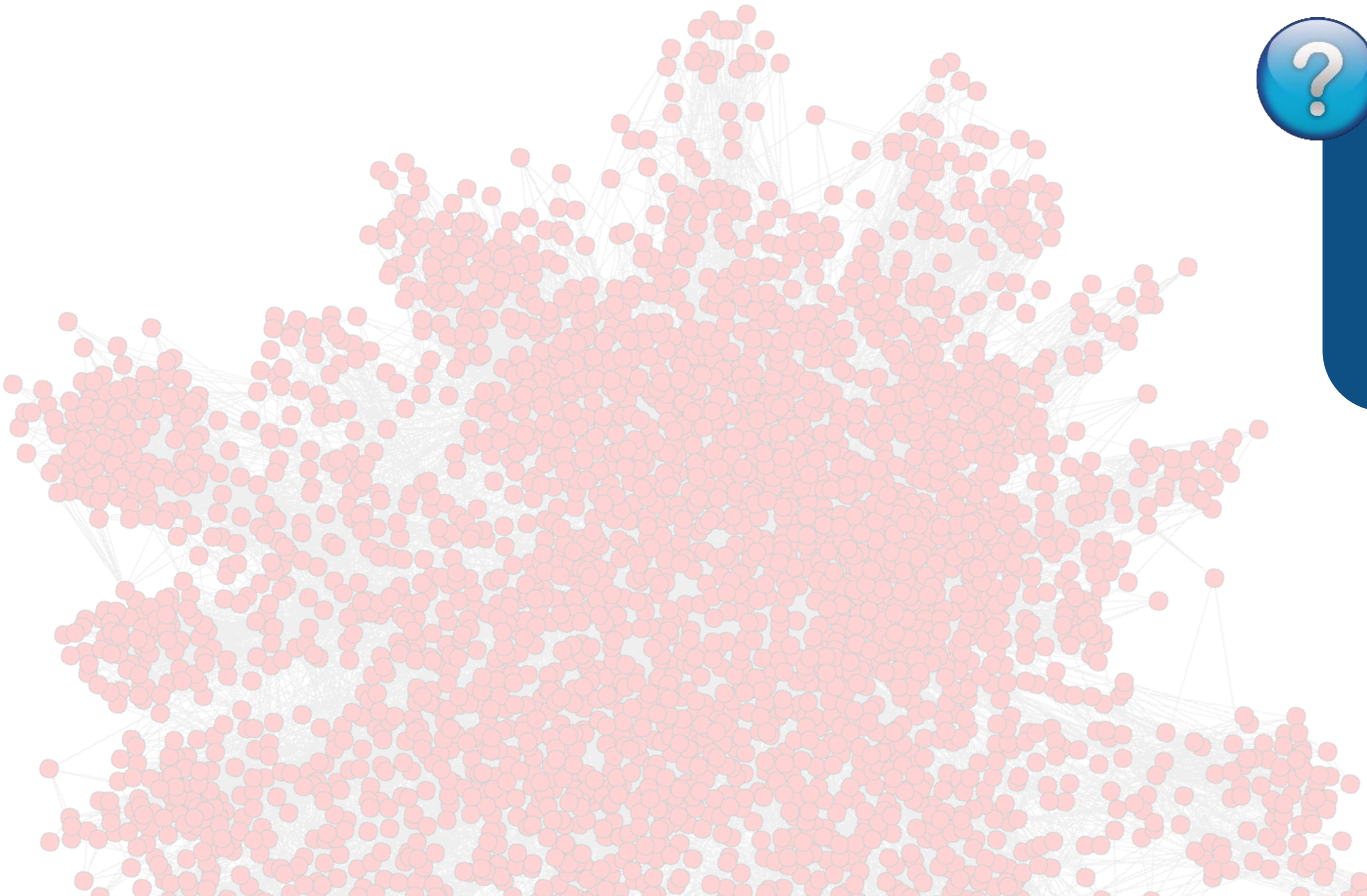
Maciej Besta, Marc Fischer, Tal Ben-Nun, Johannes De Fine Licht, Torsten Hoefler
Department of Computer Science, ETH Zurich

ABSTRACT

Developing high-performance and energy-efficient algorithms for maximum matchings is becoming increasingly important in social network analysis, computational sciences, schedul-

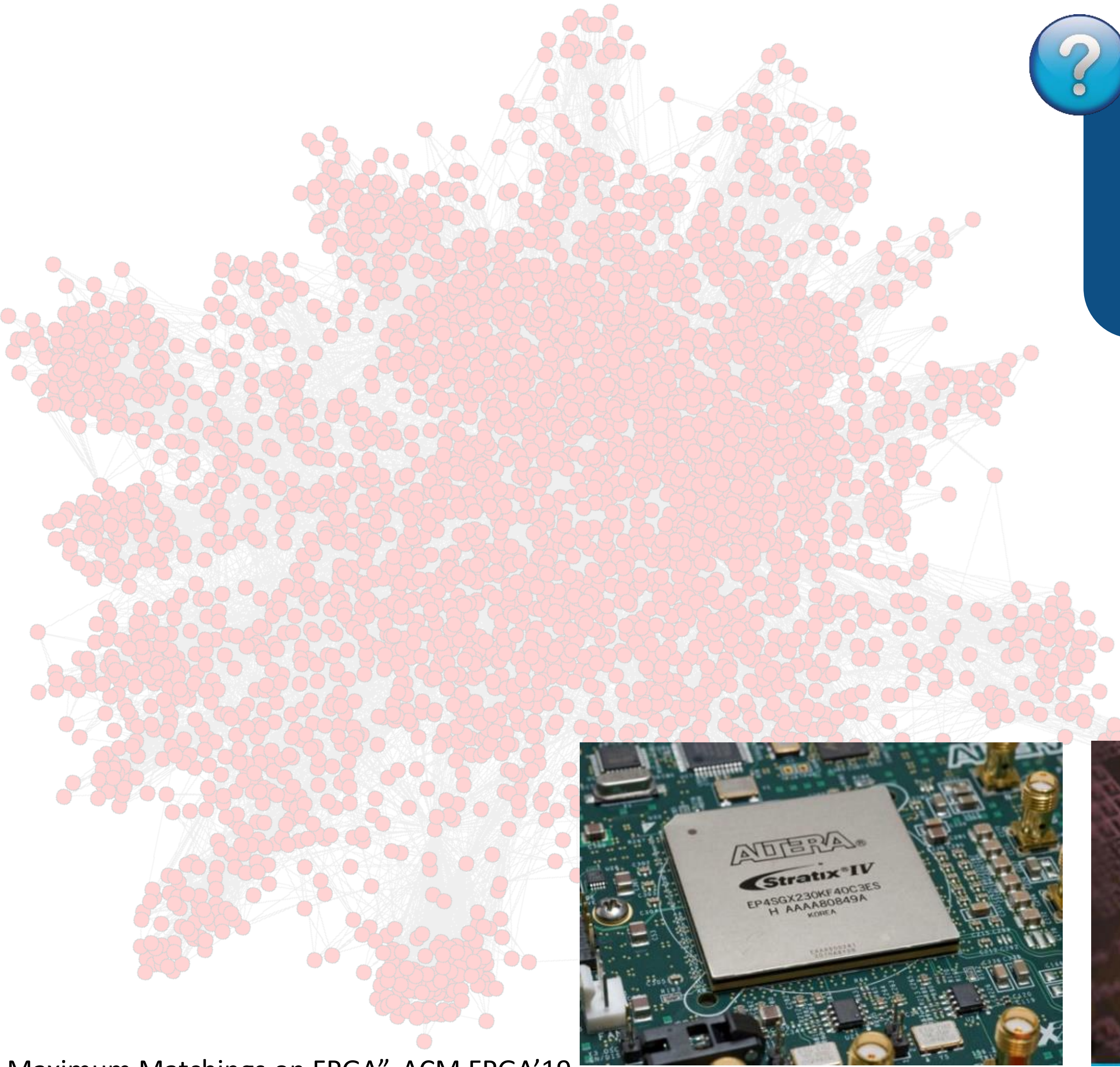
compound can be used to show the locations of double bonds in the chemical structure [59]. As deriving the exact MM is usually computationally expensive, significant focus has been placed on developing fast approximate solutions [17].

What graph programming paradigm for FPGAs and why?



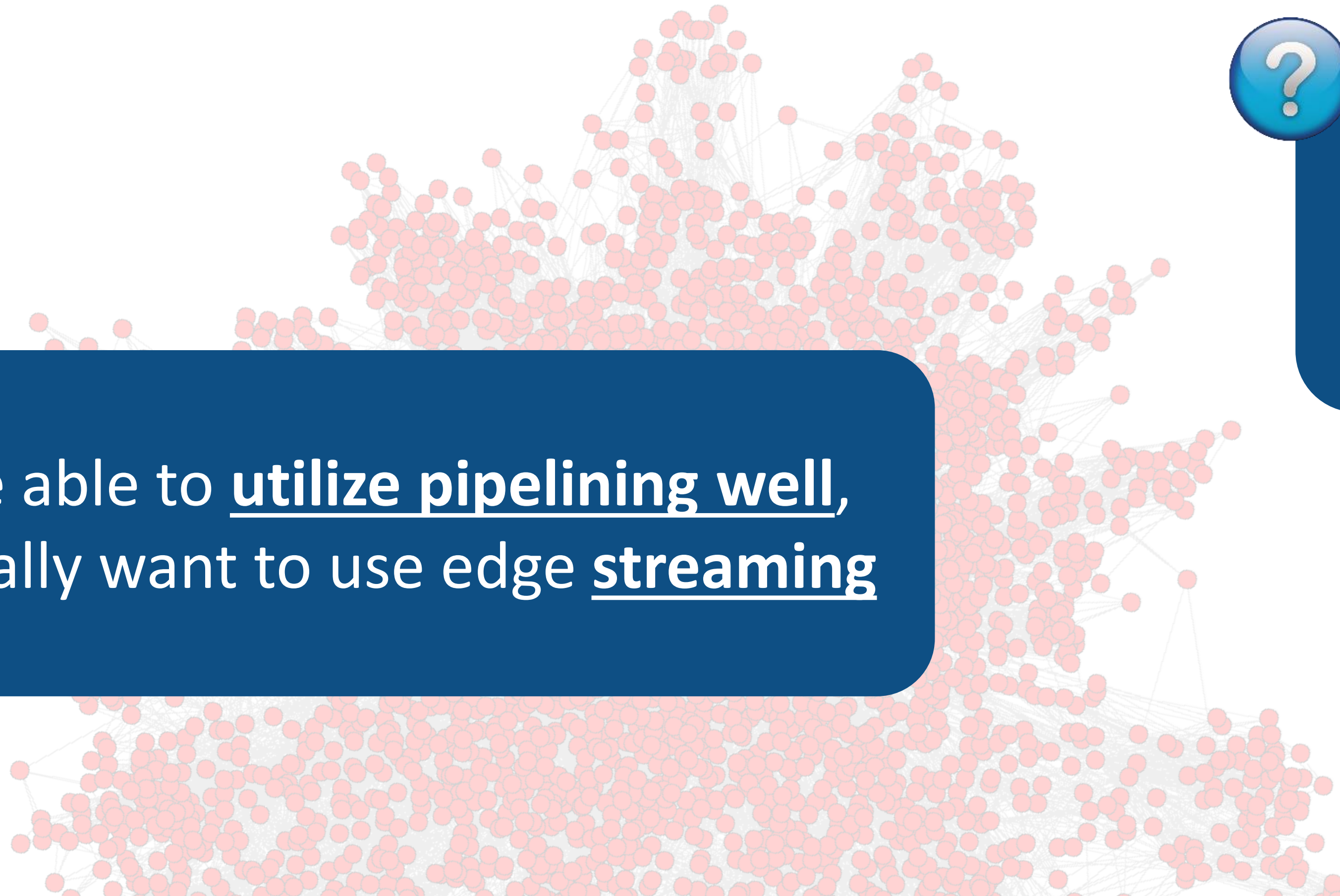
M. Besta et al.: “Substream-Centric Maximum Matchings on FPGA”, ACM FPGA’19

What graph programming paradigm for FPGAs and why?



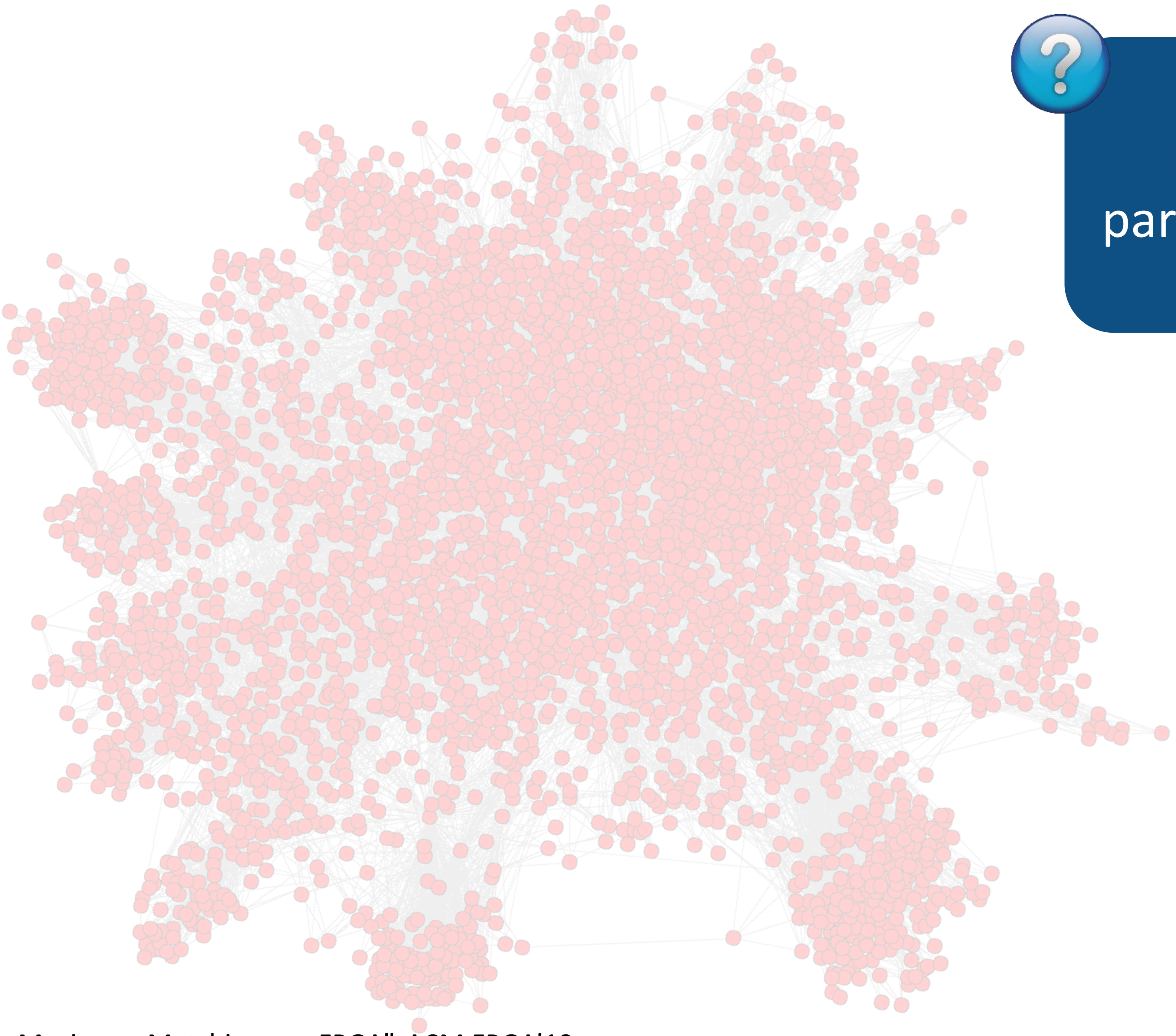
M. Besta et al.: “Substream-Centric Maximum Matchings on FPGA”, ACM FPGA’19

What graph programming paradigm for FPGAs and why?



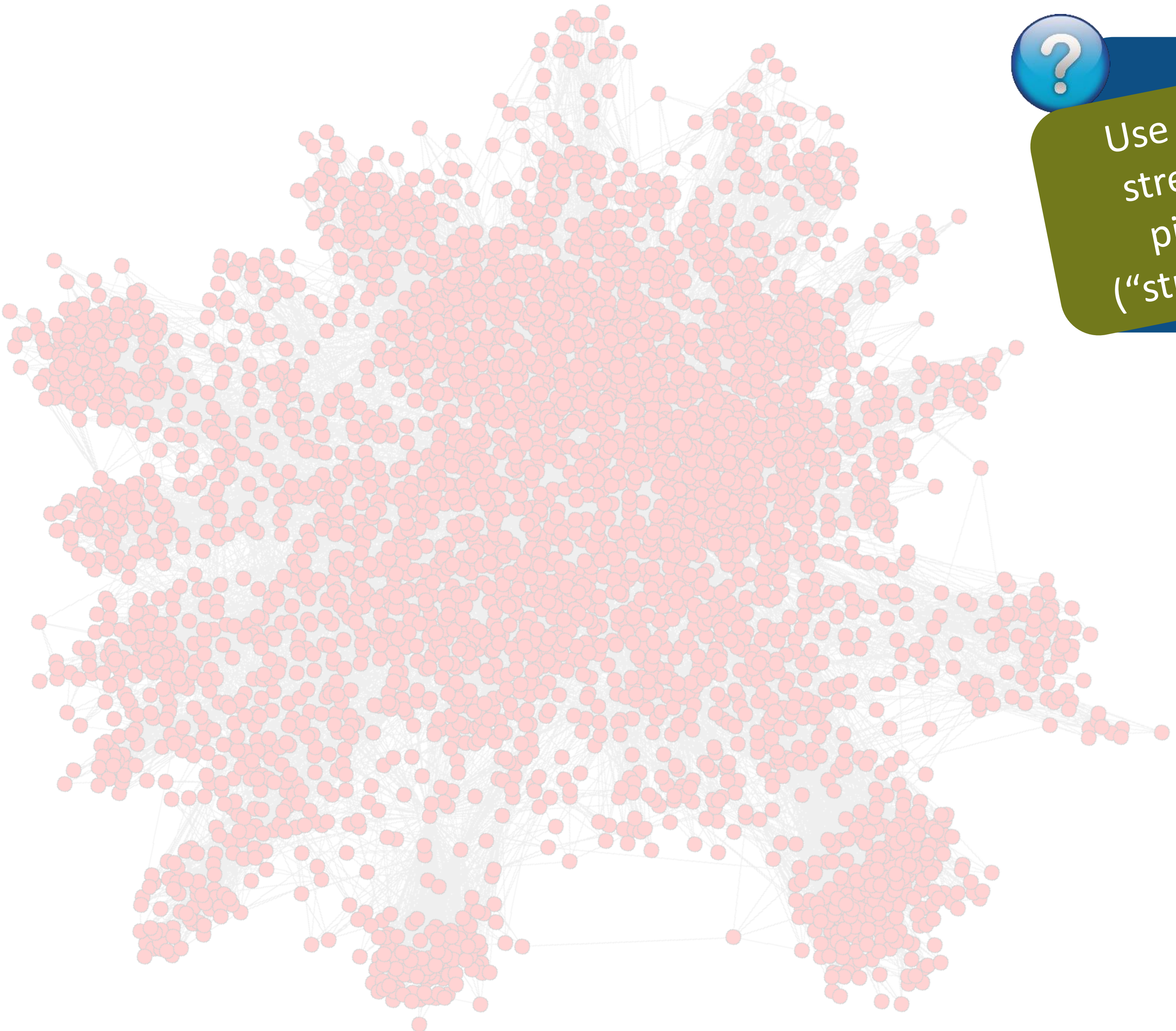
To be able to **utilize pipelining well**, we really want to use edge **streaming**

M. Besta et al.: "Substream-Centric Maximum Matchings on FPGA", ACM FPGA'19



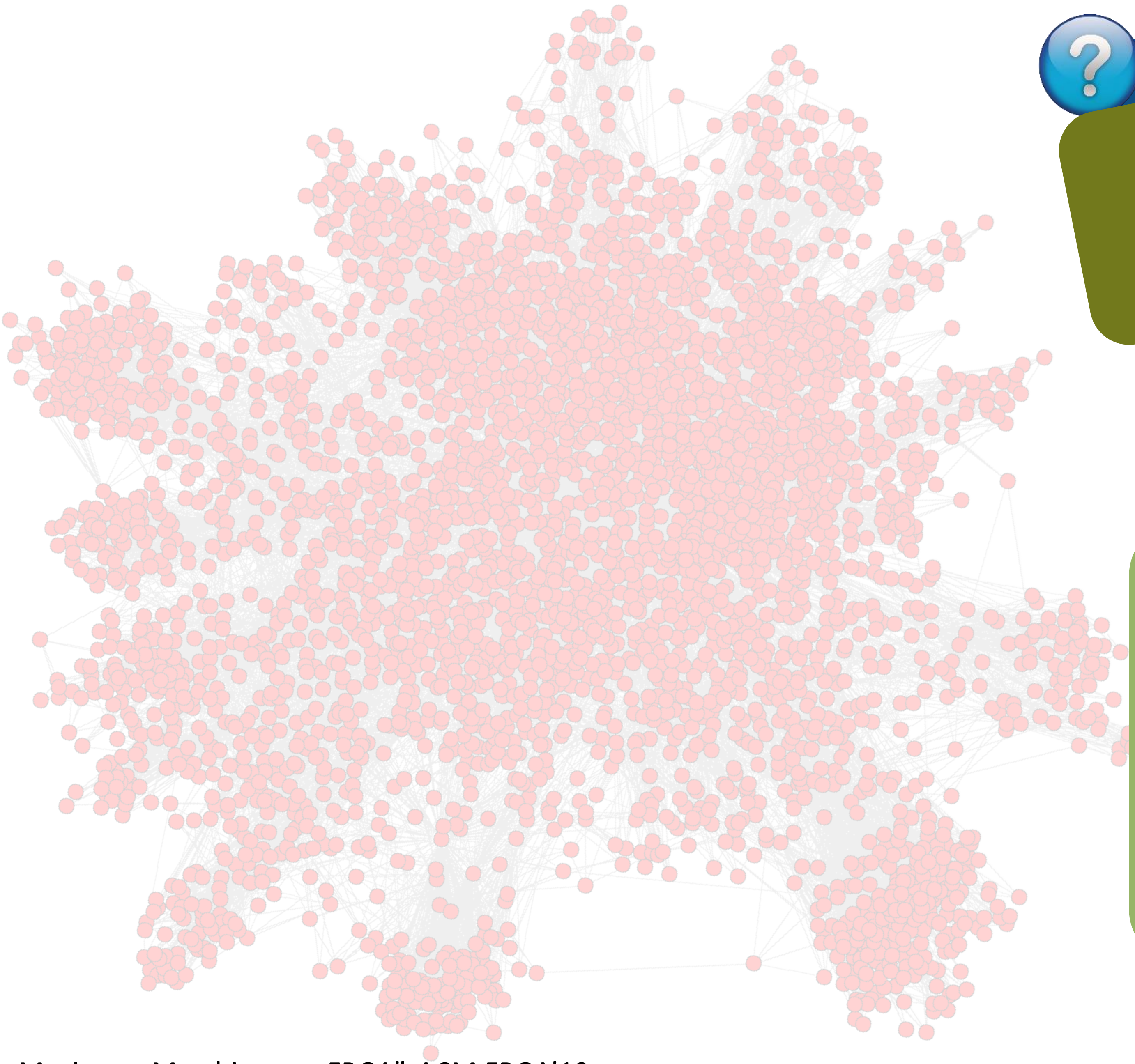
What graph programming paradigm for FPGAs and why?

M. Besta et al.: “Substream-Centric Maximum Matchings on FPGA”, ACM FPGA’19



Use some form of edge streaming; we can use pipelining efficiently ("streaming ≈ pipelining")

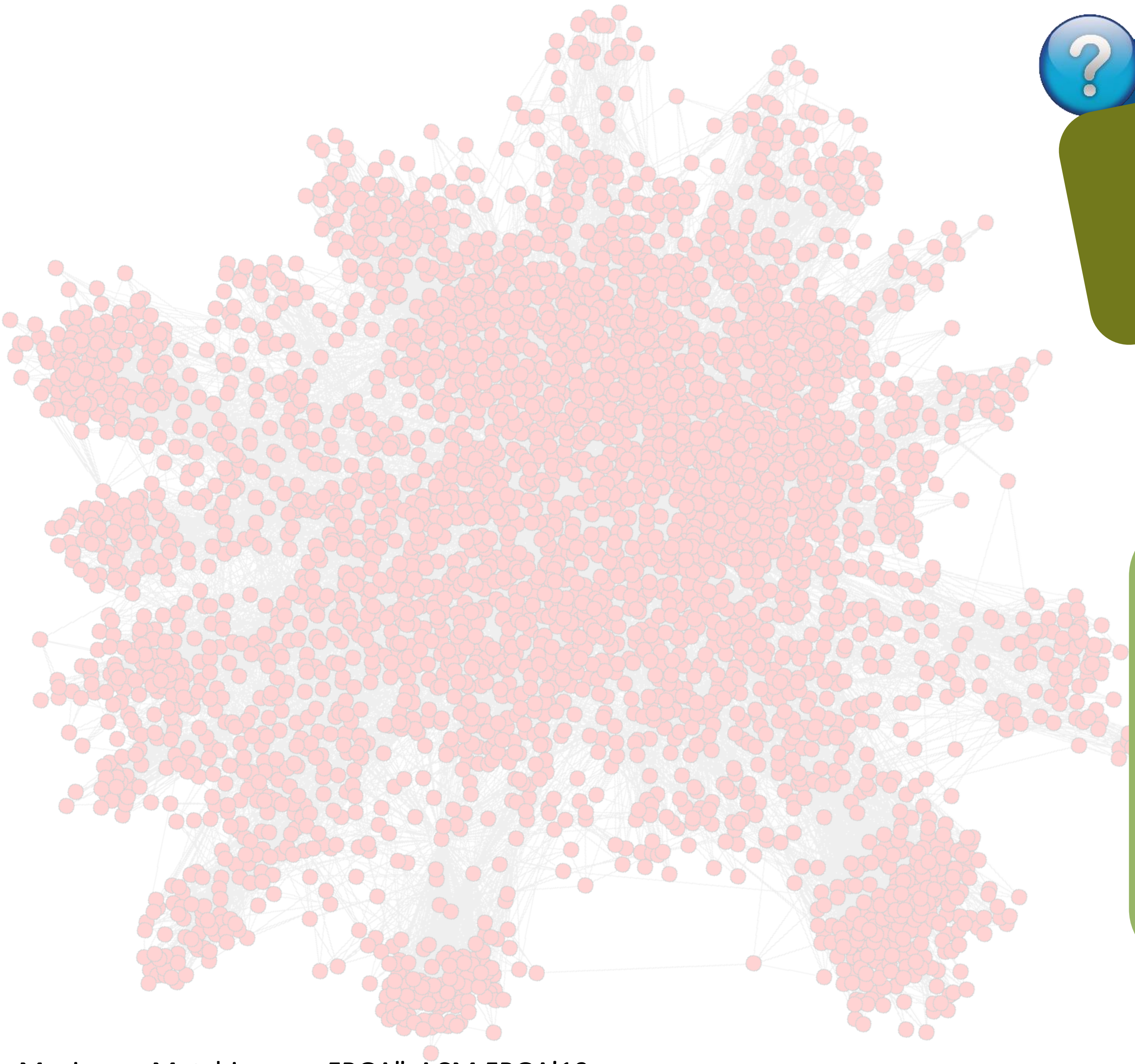
M. Besta et al.: "Substream-Centric Maximum Matchings on FPGA", ACM FPGA'19



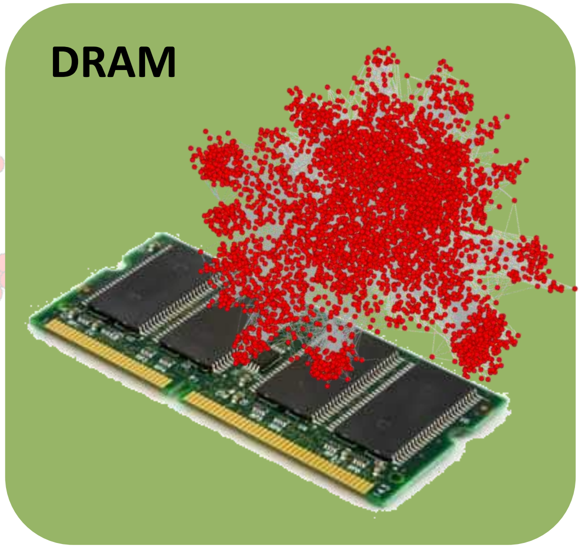
Use some form of edge streaming; we can use pipelining efficiently ("streaming ≈ pipelining")

DRAM

M. Besta et al.: "Substream-Centric Maximum Matchings on FPGA", ACM FPGA'19



Use some form of edge streaming; we can use pipelining efficiently ("streaming ≈ pipelining")

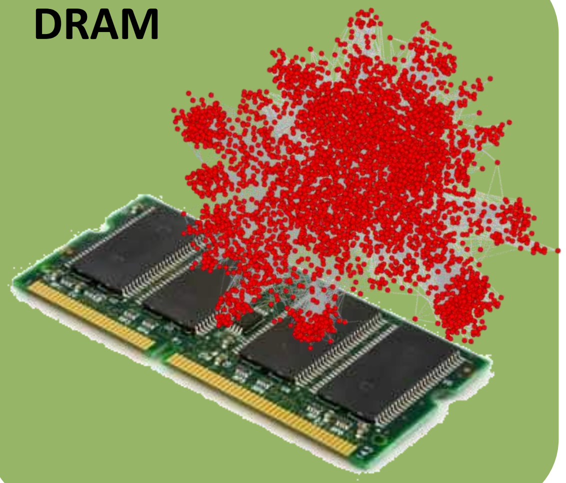


M. Besta et al.: "Substream-Centric Maximum Matchings on FPGA", ACM FPGA'19

Use some form of edge streaming; we can use pipelining efficiently ("streaming ≈ pipelining")

Some processing unit (CPU, GPU, FPGA, …, for a moment we don't care)

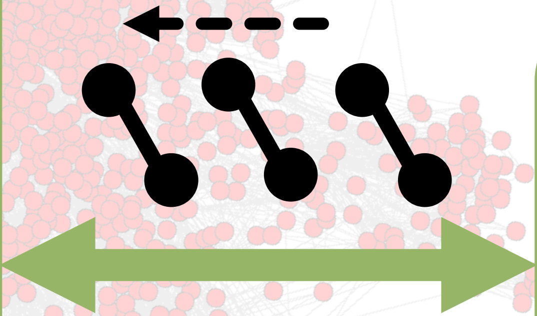
DRAM



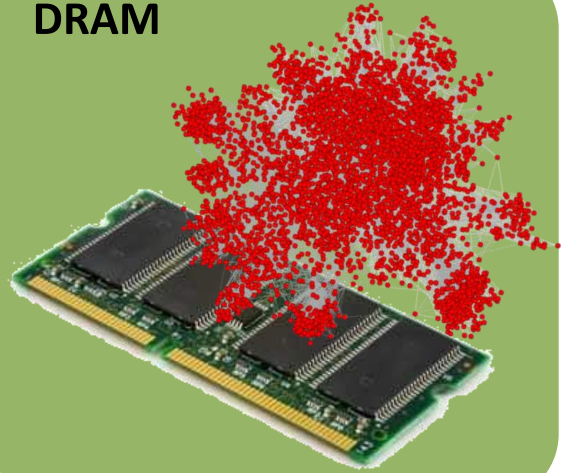
M. Besta et al.: "Substream-Centric Maximum Matchings on FPGA", ACM FPGA'19

Use some form of edge streaming; we can use pipelining efficiently ("streaming ≈ pipelining")

Some processing unit (CPU, GPU, FPGA, …, for a moment we don't care)

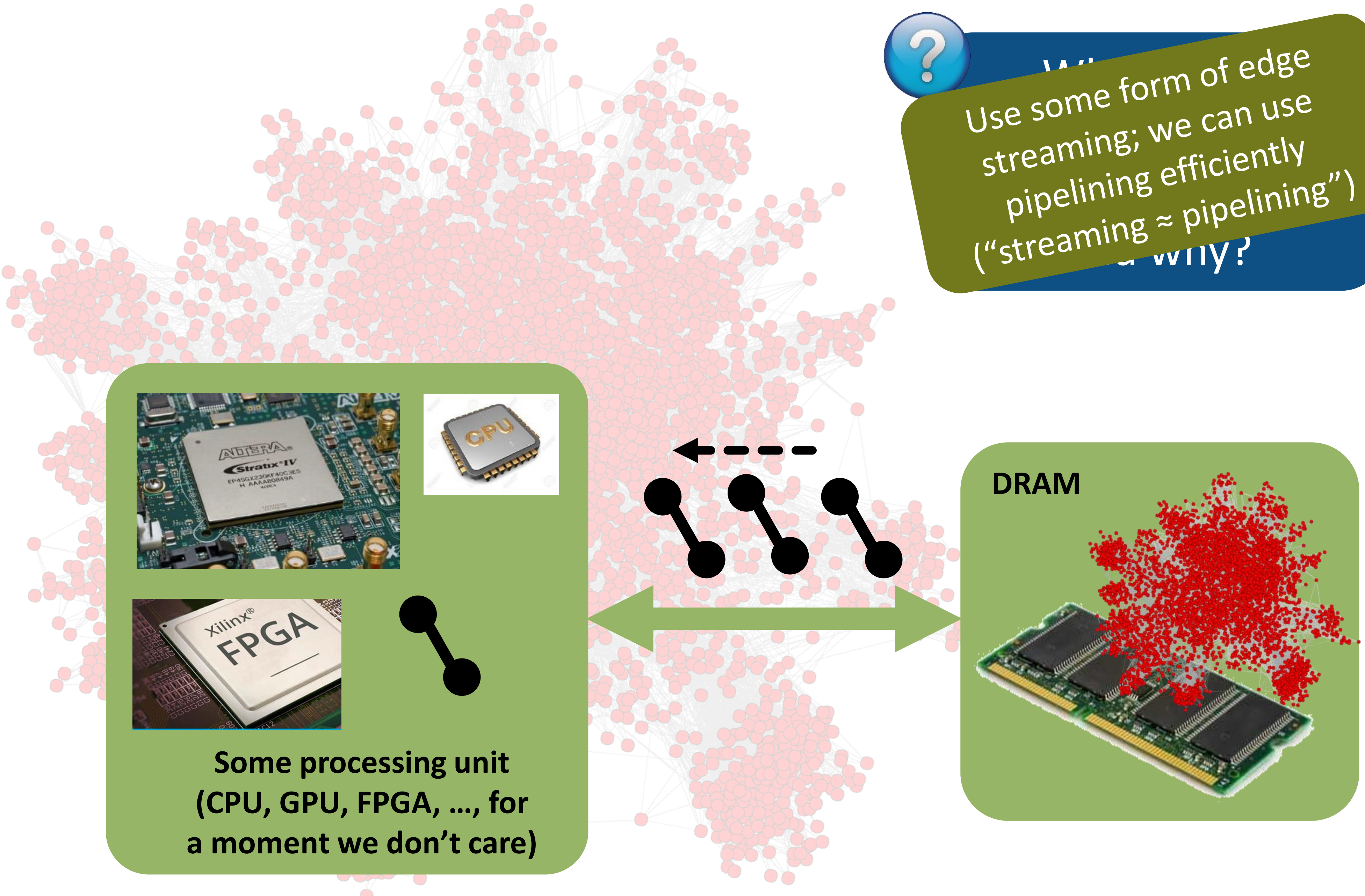


DRAM



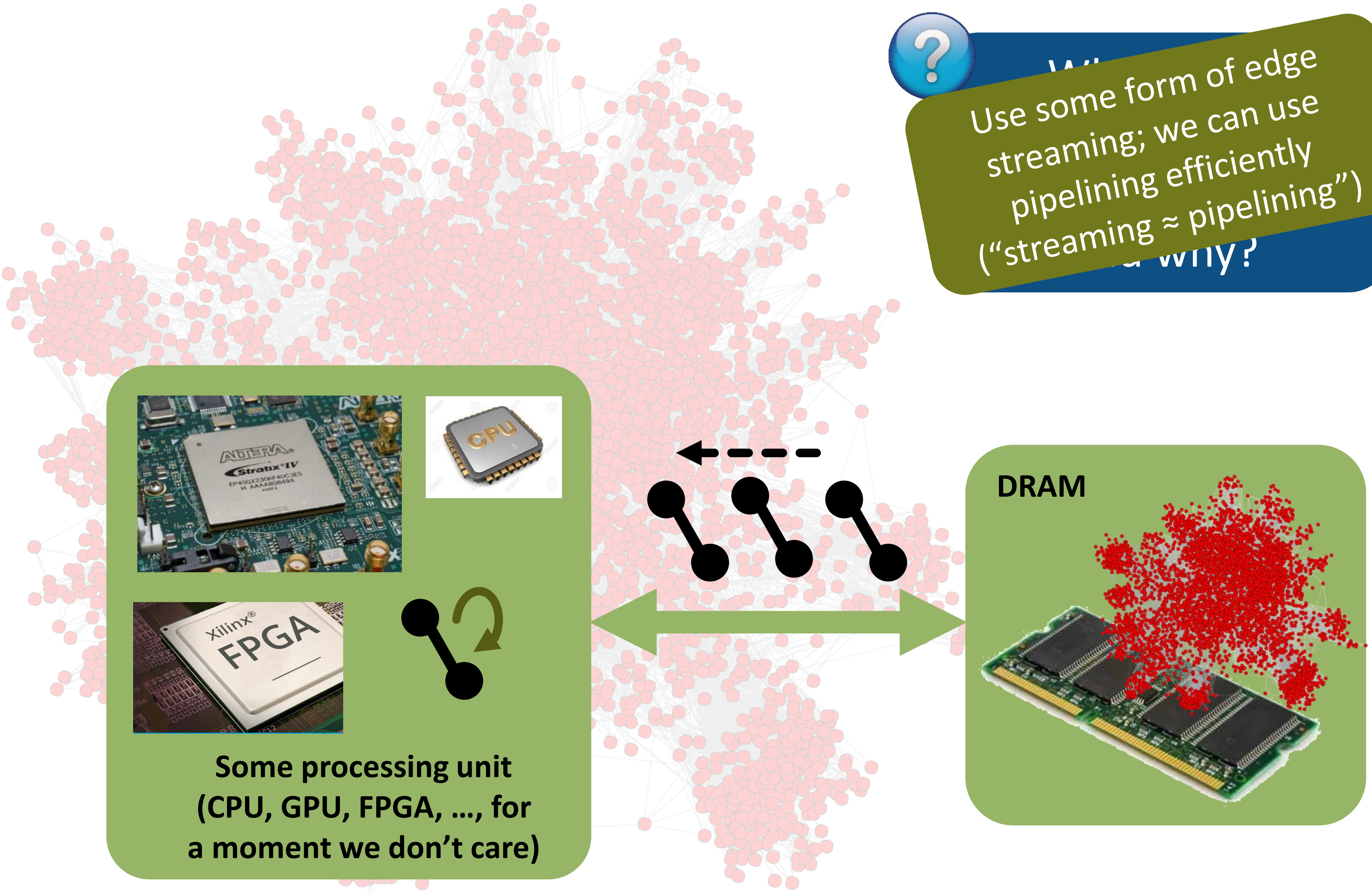
M. Besta et al.: "Substream-Centric Maximum Matchings on FPGA", ACM FPGA'19

Use some form of edge streaming; we can use pipelining efficiently ("streaming ≈ pipelining")



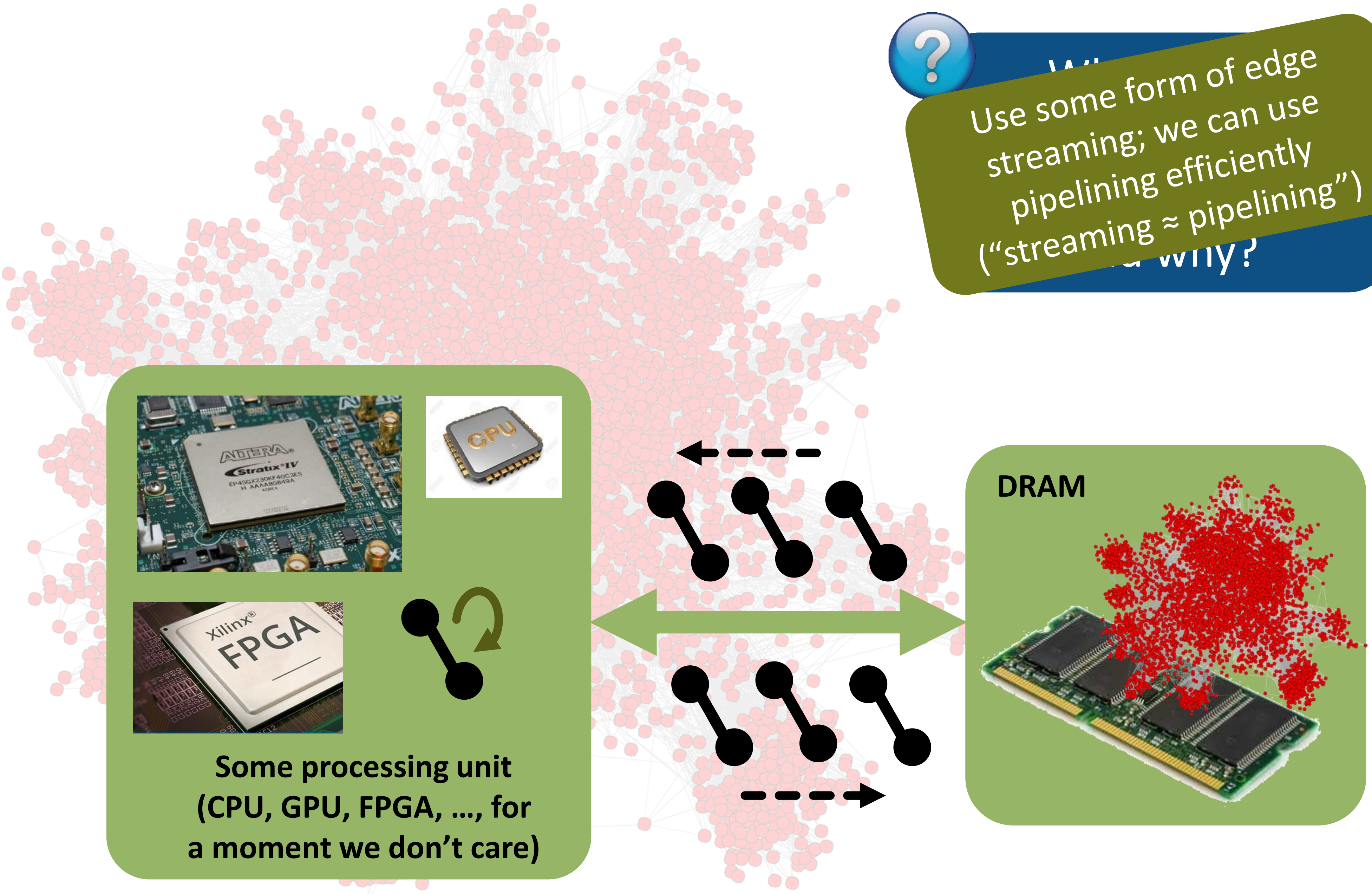
M. Besta et al.: "Substream-Centric Maximum Matchings on FPGA", ACM FPGA'19

Use some form of edge streaming; we can use pipelining efficiently (“streaming ≈ pipelining”)



M. Besta et al.: “Substream-Centric Maximum Matchings on FPGA”, ACM FPGA’19

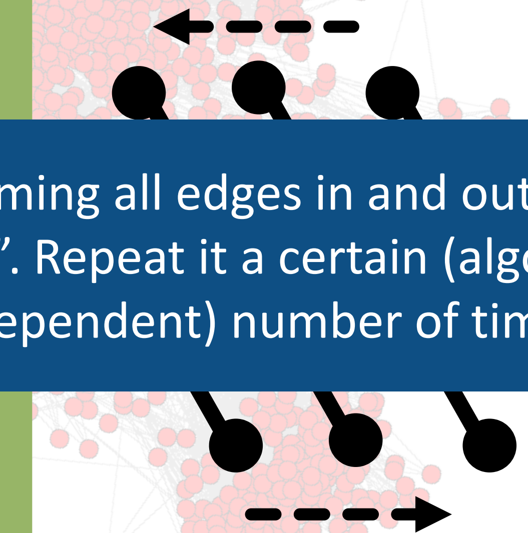
Use some form of edge streaming; we can use pipelining efficiently ("streaming ≈ pipelining")



M. Besta et al.: "Substream-Centric Maximum Matchings on FPGA", ACM FPGA'19

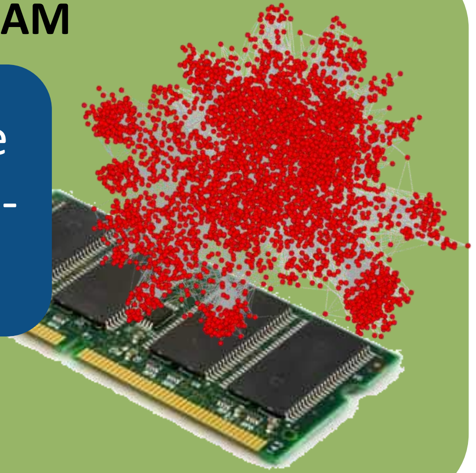
Use some form of edge streaming; we can use pipelining efficiently ("streaming ≈ pipelining")

Some processing unit (CPU, GPU, FPGA, …, for a moment we don't care)



Streaming all edges in and out is one "**pass**". Repeat it a certain (algorithm-dependent) number of times

DRAM



M. Besta et al.: "Substream-Centric Maximum Matchings on FPGA", ACM FPGA'19

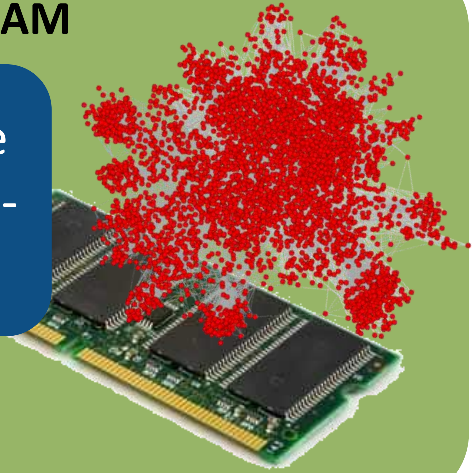
Use some form of edge streaming; we can use pipelining efficiently ("streaming ≈ pipelining")

Issues…

Some processing unit (CPU, GPU, FPGA, …, for a moment we don't care)

Streaming all edges in and out is one "**pass**". Repeat it a certain (algorithm-dependent) number of times

DRAM



M. Besta et al.: "Substream-Centric Maximum Matchings on FPGA", ACM FPGA'19

…How to minimize the number of "passes" over edges? This can get **really** bad in the "traditional" edge-centric approach (e.g., BFS needs *D* passes; *D* = diameter [1]).

Use some form of edge streaming; we can use pipelining efficiently ("streaming ≈ pipelining")

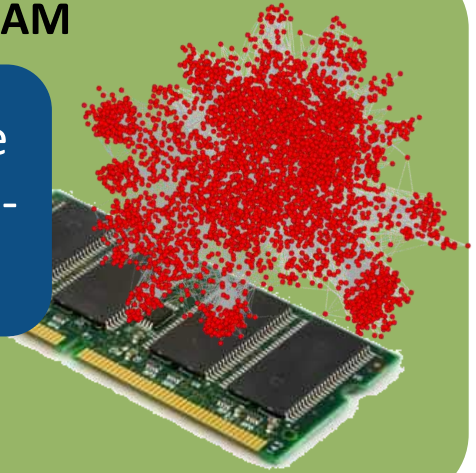
…Processing edges is sequential – **how to incorporate parallelism**?

Issues…

Some processing unit (CPU, GPU, FPGA, …, for a moment we don't care)

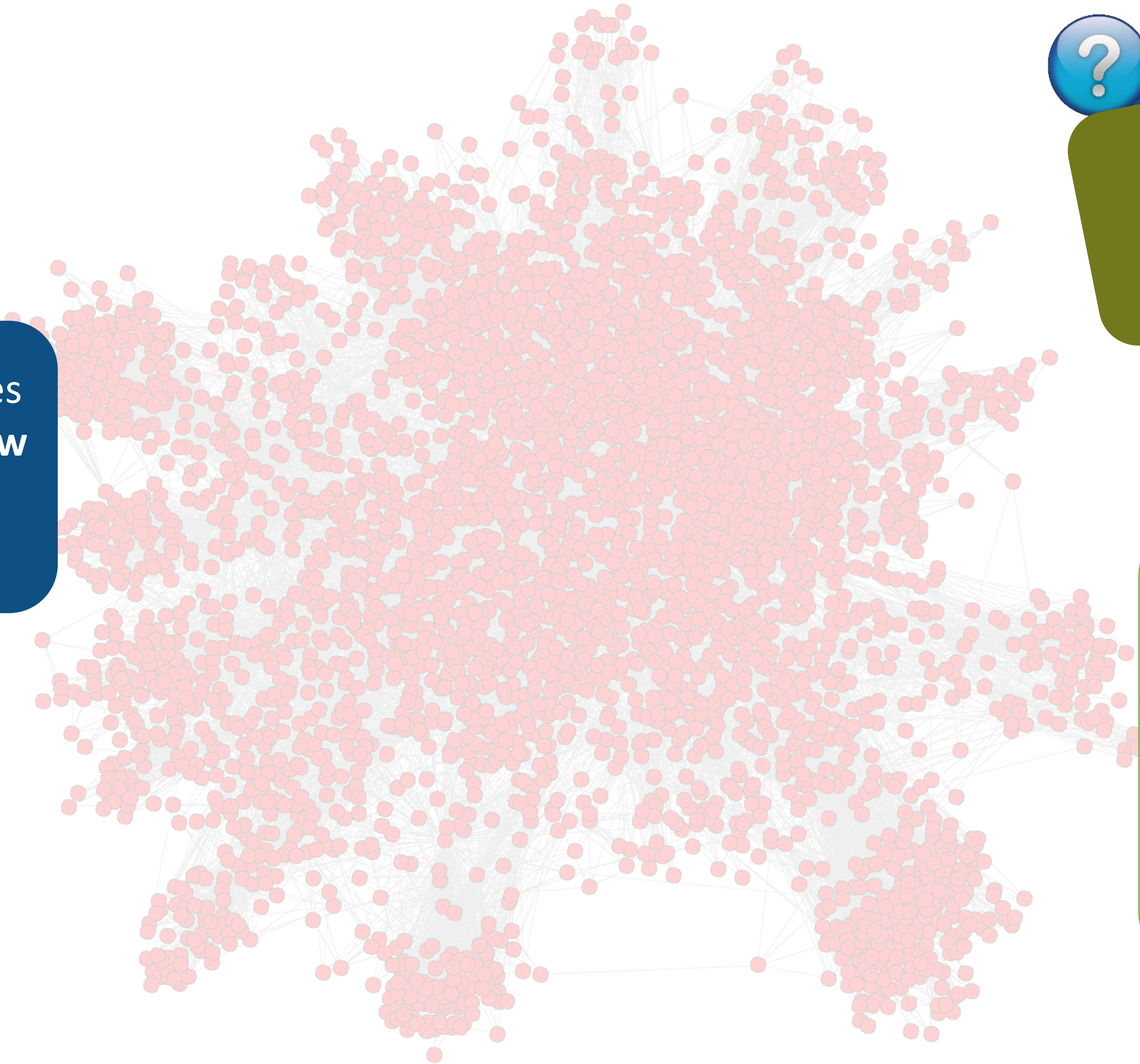
Streaming all edges in and out is one "**pass**". Repeat it a certain (algorithm-dependent) number of times

DRAM

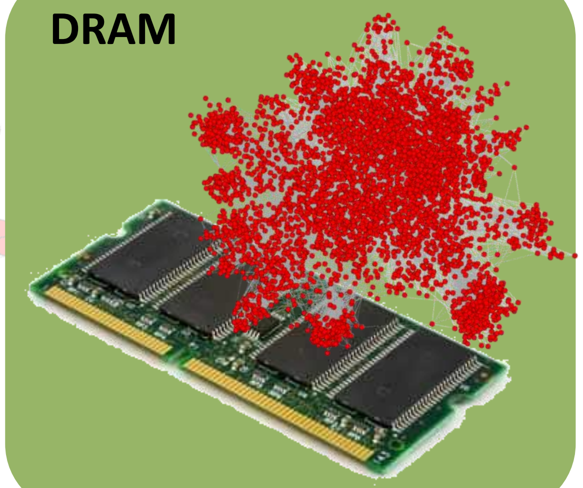


[1] A. Roy et al. X-stream: Edge-Centric Graph Processing using Streaming Partitions. SOSP. 2013.

…Processing edges is sequential – **how to incorporate parallelism**?



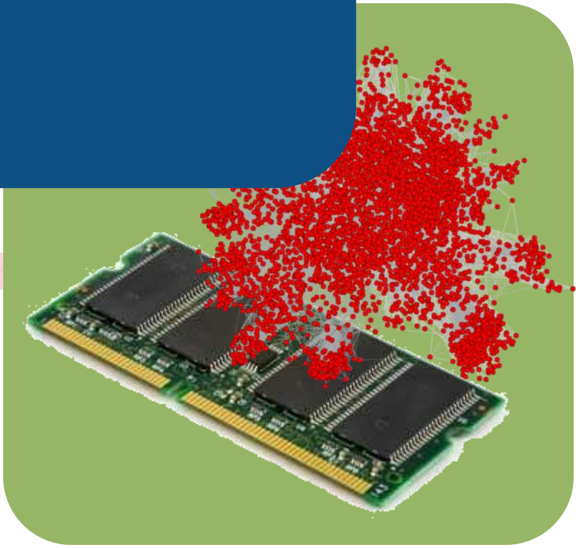
Use some form of edge streaming; we can use pipelining efficiently (“streaming ≈ pipelining”)



Use some form of edge streaming; we can use pipelining efficiently
ing ≈ pipelining”)

…Pro
is seq
to i
pa

Substream-Centric: A new paradigm for processing graphs

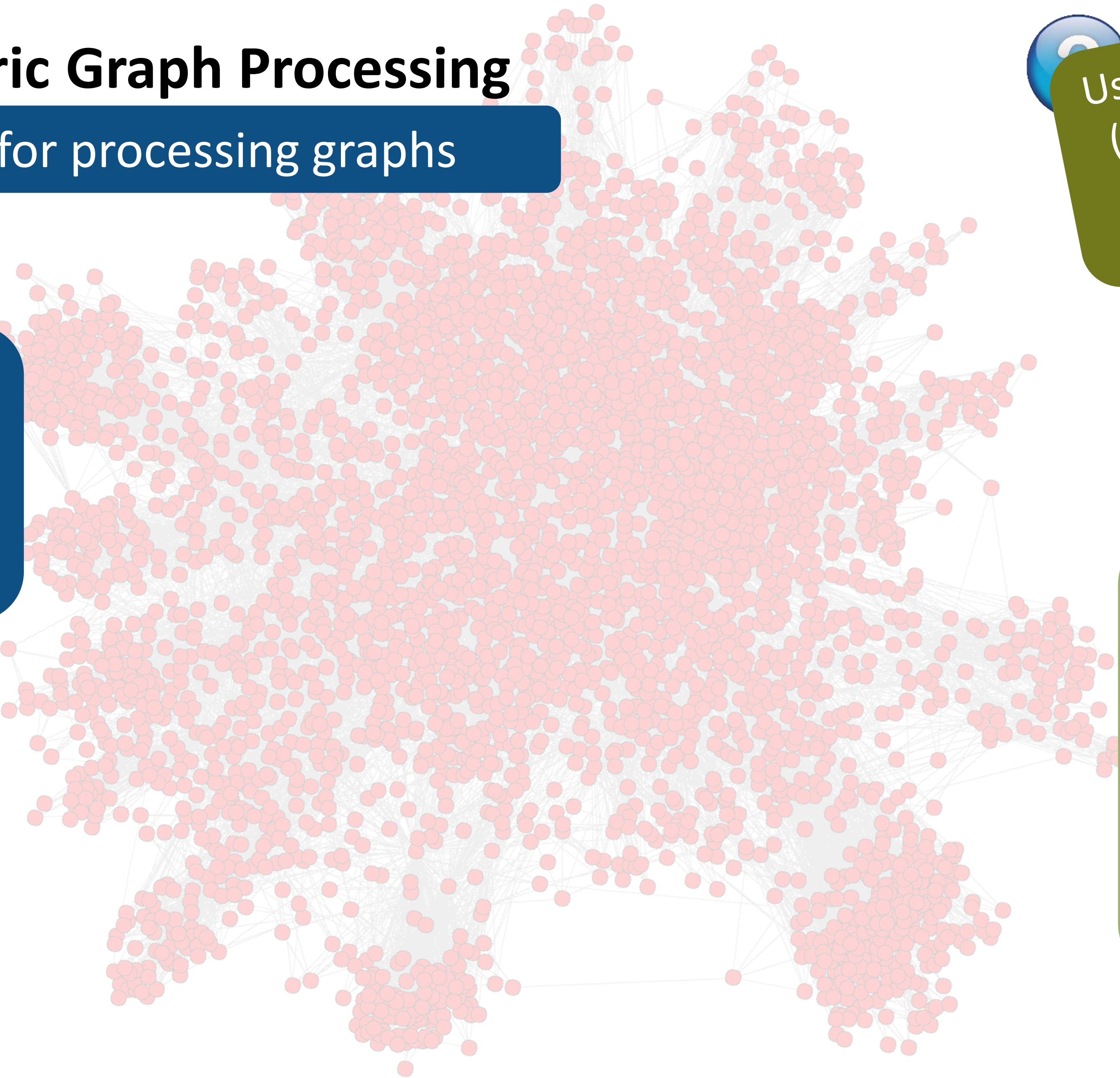


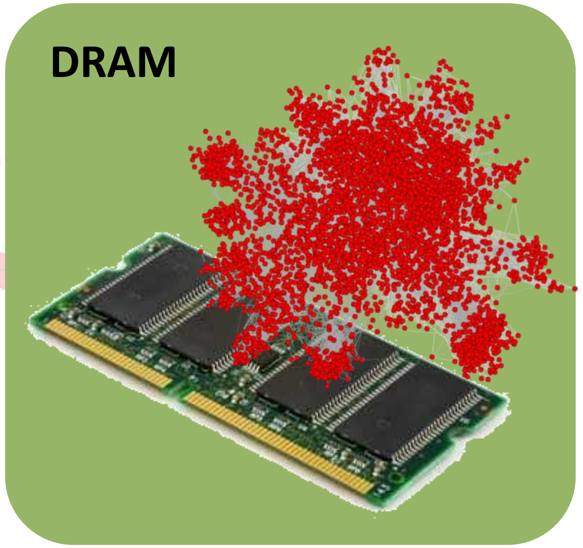
Substream-Centric Graph Processing

A new paradigm for processing graphs

…Processing edges is sequential – **how to incorporate parallelism**?

Use some form of streaming (aka **edge-centric**); we can use pipelining efficiently ("streaming ≈ pipelining")





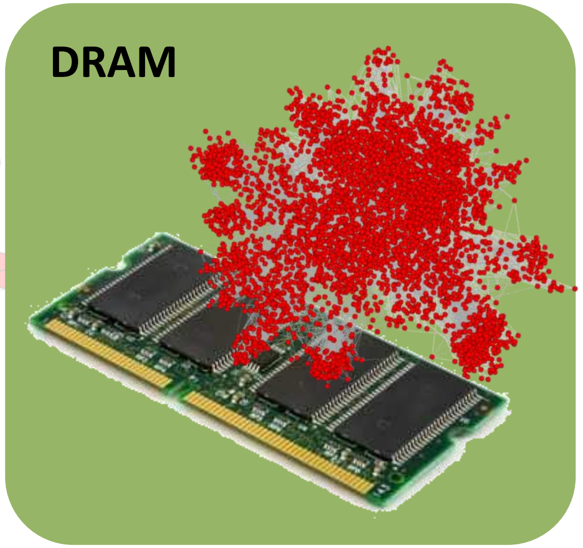
Substream-Centric Graph Processing

A new paradigm for processing graphs

It enhances edge-centric streaming approaches

Use some form of streaming (aka **edge-centric**); we can use pipelining efficiently ("streaming ≈ pipelining")

…Processing edges is sequential – **how to incorporate parallelism**?



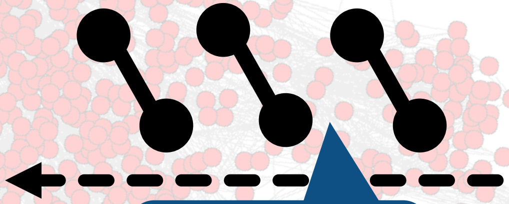
Substream-Centric Graph Processing

A new paradigm for processing graphs

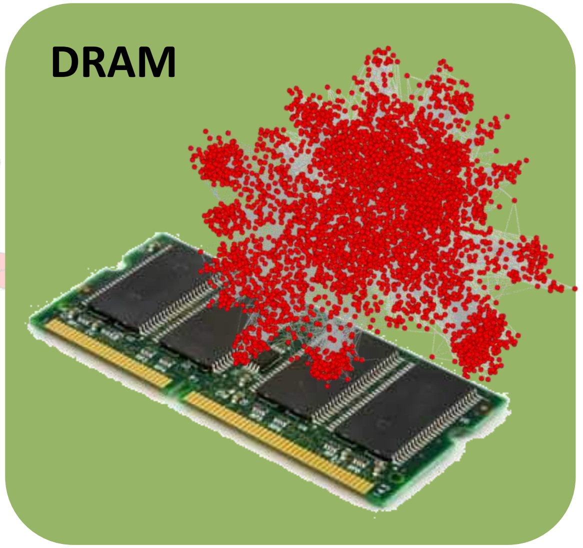
It enhances edge-centric streaming approaches

Use some form of streaming (aka **edge-centric**); we can use pipelining efficiently (“streaming ≈ pipelining”)

…Processing edges is sequential – **how to incorporate parallelism**?



Weighted edges



Substream-Centric Graph Processing

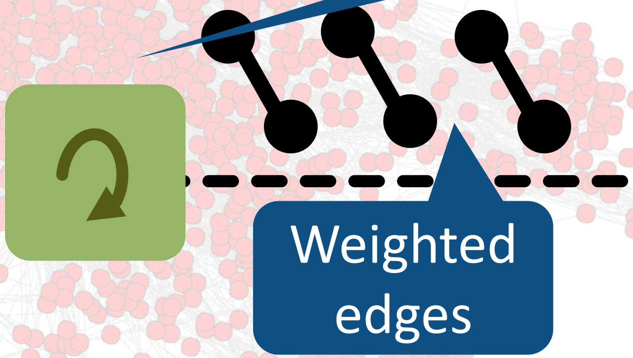
A new paradigm for processing graphs

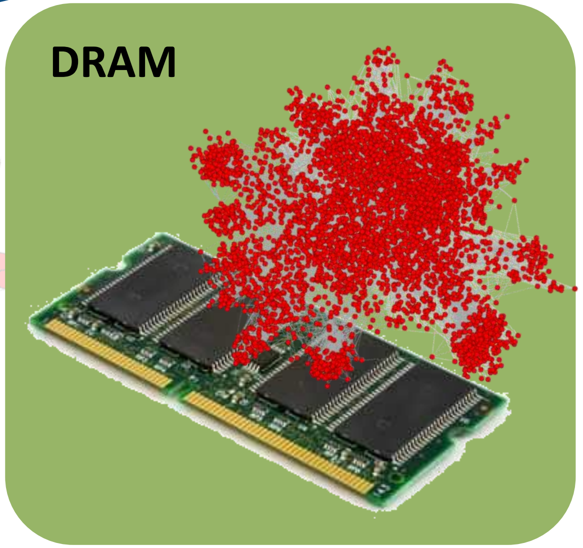
It enhances edge-centric streaming approaches

Use some form of streaming (aka **edge-centric**); we can use pipelining efficiently (“streaming ≈ pipelining”)

…Processing edges is sequential – **how to incorporate parallelism**?

Divide the input stream of edges according to some (algorithm-specific) pattern





Substream-Centric Graph Processing

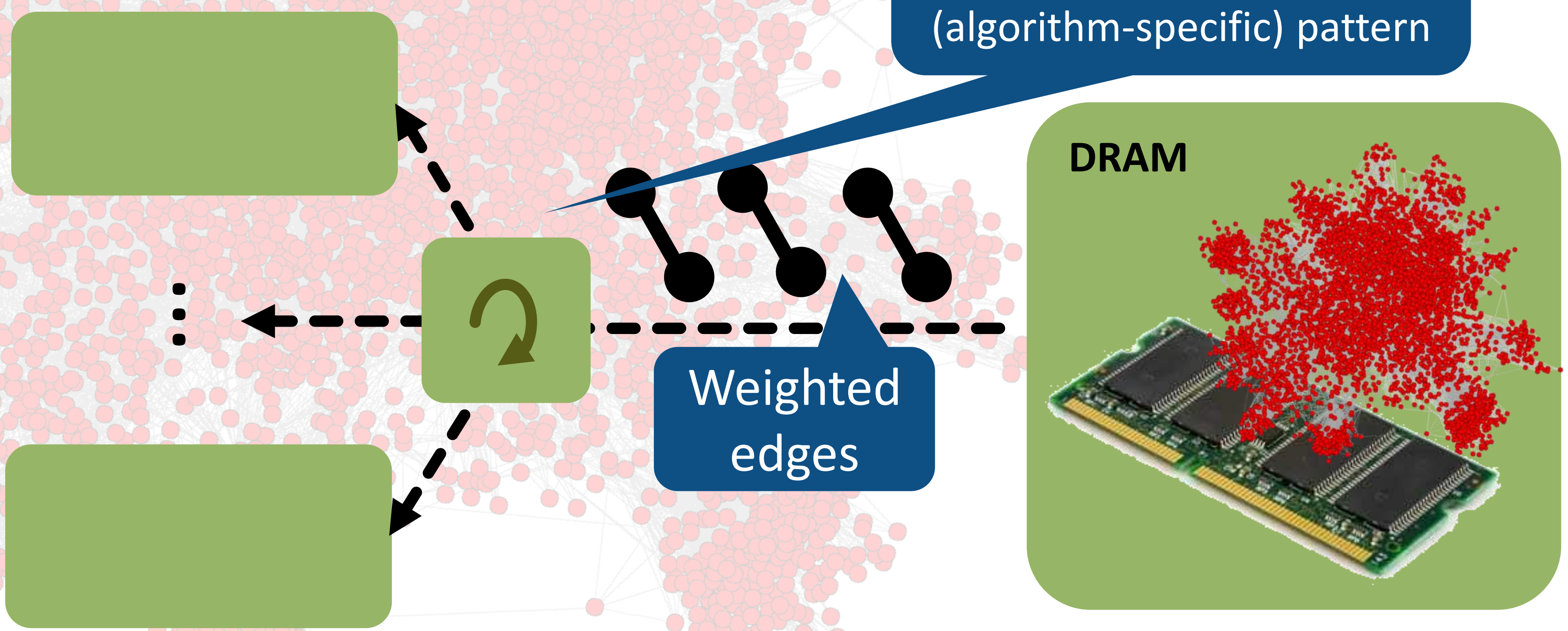
A new paradigm for processing graphs

It enhances edge-centric streaming approaches

Use some form of streaming (aka **edge-centric**); we can use pipelining efficiently ("streaming ≈ pipelining")

…Processing edges is sequential – **how to incorporate parallelism**?

Divide the input stream of edges according to some (algorithm-specific) pattern



Substream-Centric Graph Processing

A new paradigm for processing graphs

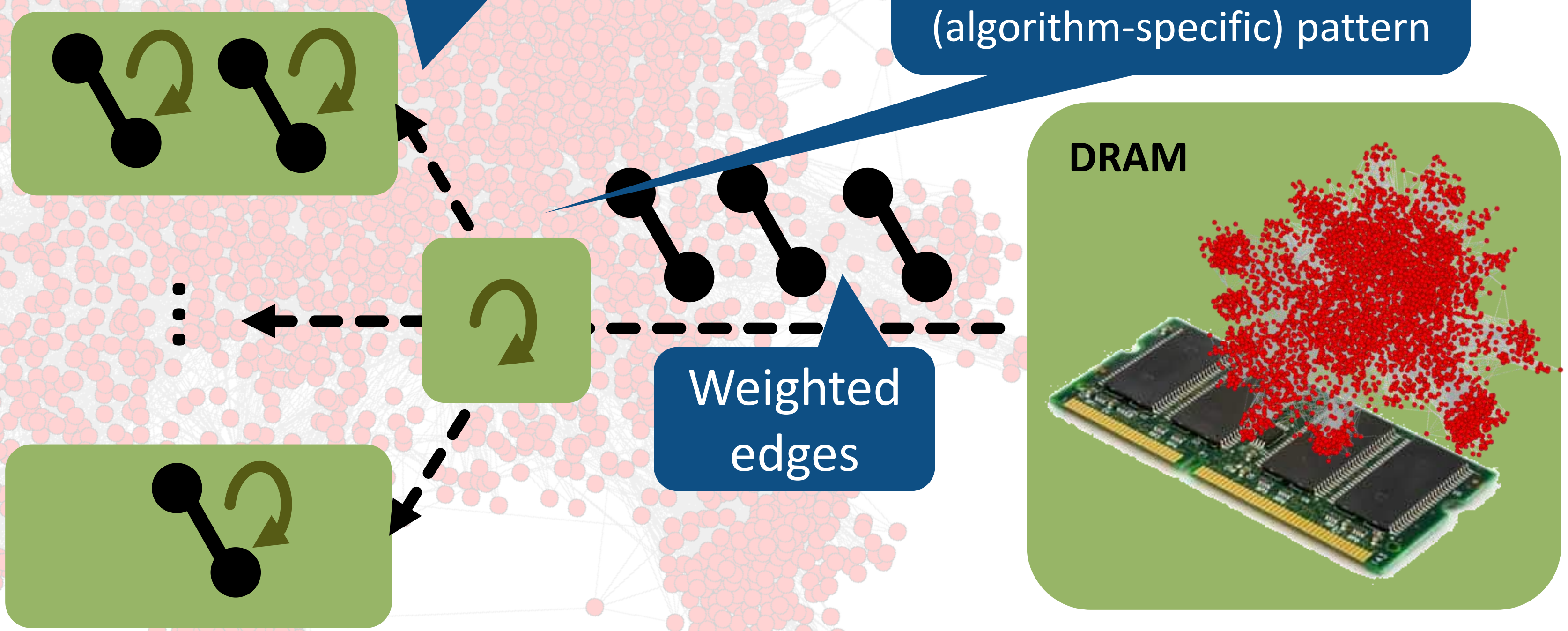
It enhances edge-centric streaming approaches

Use some form of streaming (aka **edge-centric**); we can use pipelining efficiently ("streaming ≈ pipelining")

…Processing edges is sequential – **how to incorporate parallelism**?

Process "substreams" independently

Divide the input stream of edges according to some (algorithm-specific) pattern



Weighted edges

Substream-Centric Graph Processing

A new paradigm for processing graphs

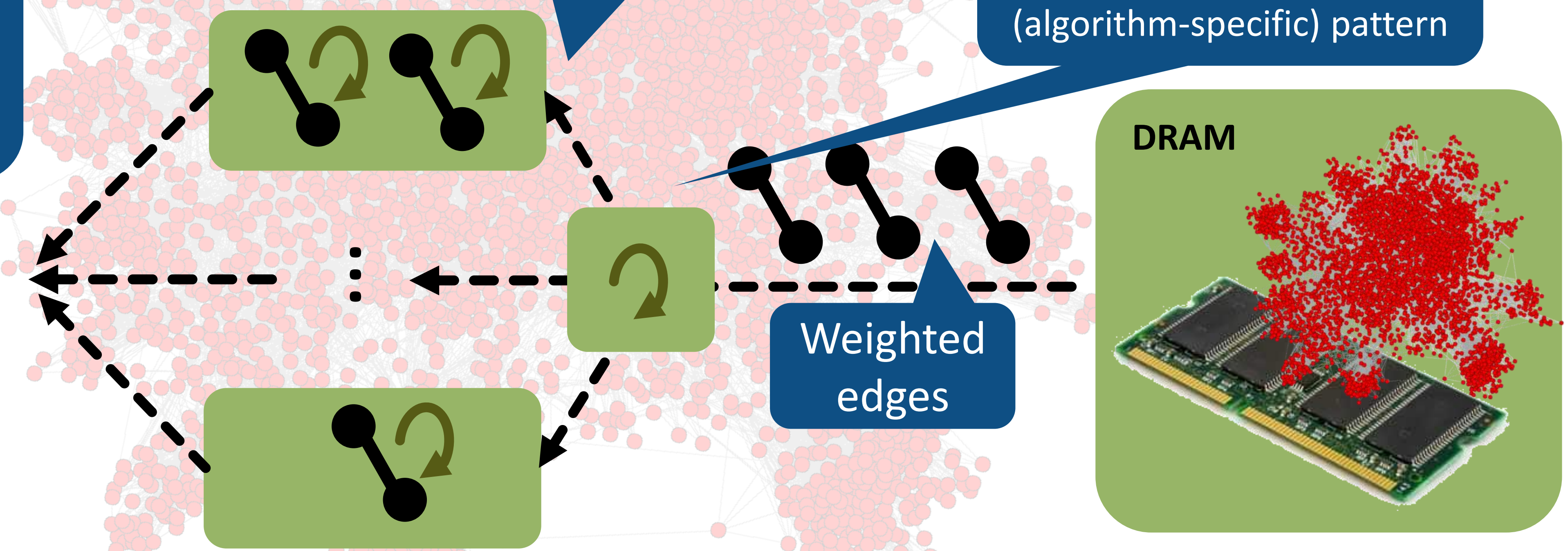
It enhances edge-centric streaming approaches

Use some form of streaming (aka **edge-centric**); we can use pipelining efficiently ("streaming ≈ pipelining")

Process "substreams" independently

...Processing edges is sequential – **how to incorporate parallelism**?

Divide the input stream of edges according to some (algorithm-specific) pattern



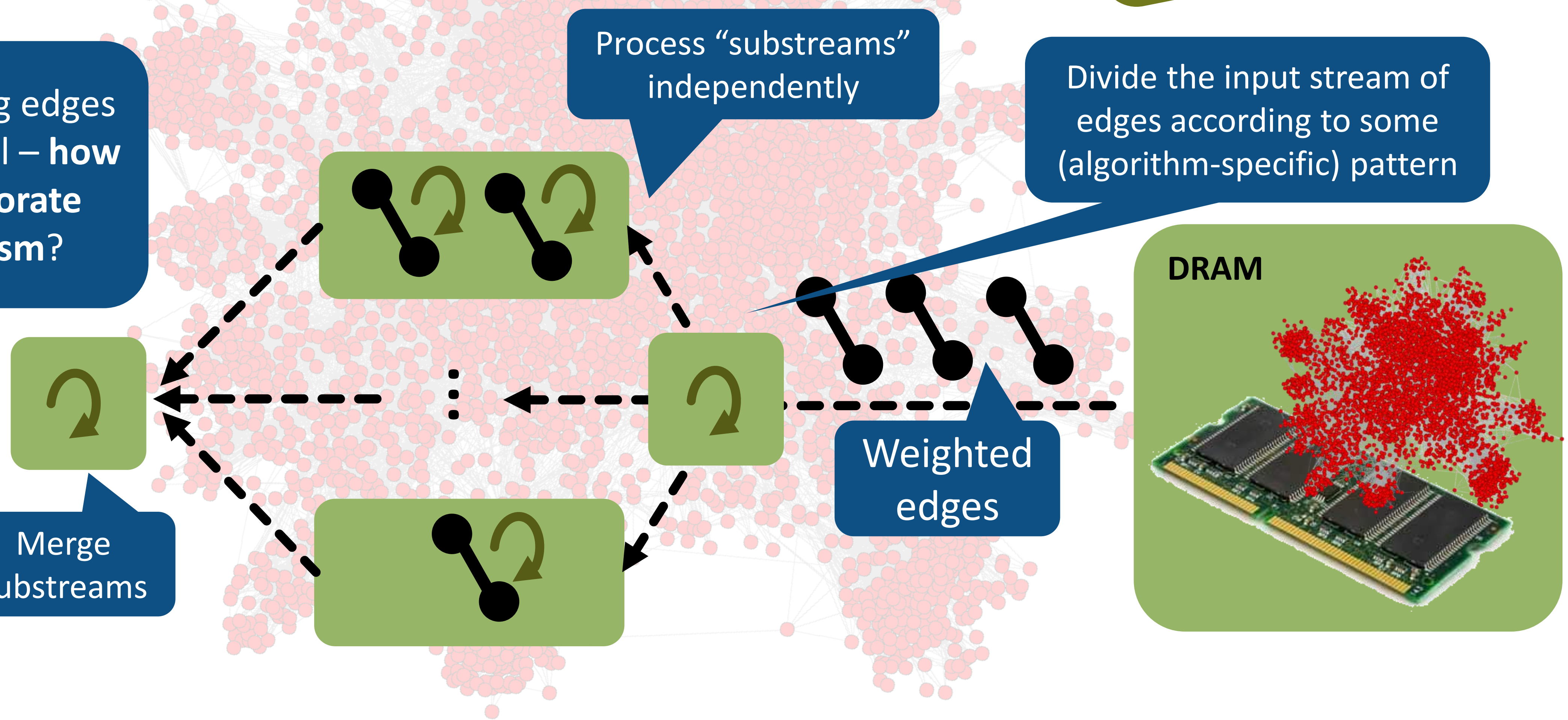
Substream-Centric Graph Processing

A new paradigm for processing graphs

It enhances edge-centric streaming approaches

Use some form of streaming (aka **edge-centric**); we can use pipelining efficiently (“streaming ≈ pipelining”)

…Processing edges is sequential – **how to incorporate parallelism**?



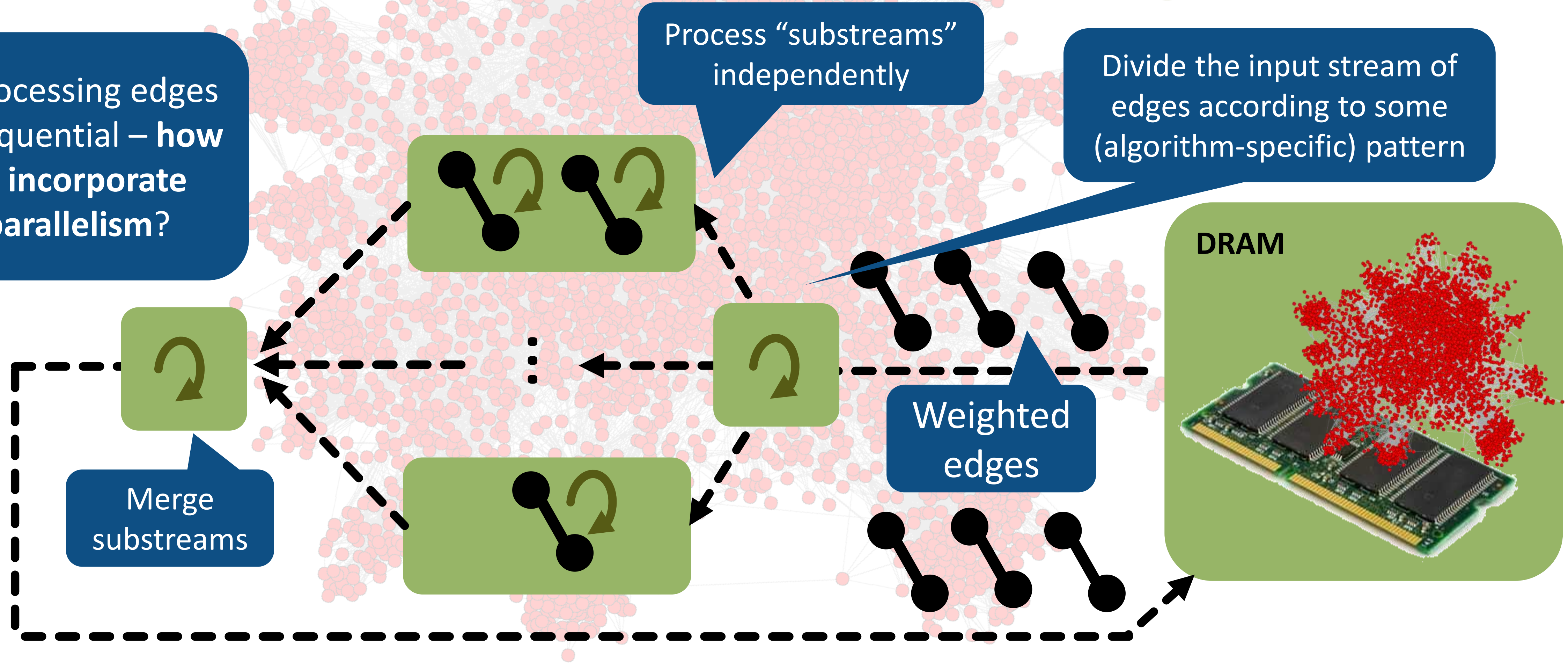
Substream-Centric Graph Processing

A new paradigm for processing graphs

It enhances edge-centric streaming approaches

Use some form of streaming (aka **edge-centric**); we can use pipelining efficiently ("streaming ≈ pipelining")

…Processing edges is sequential – **how to incorporate parallelism**?



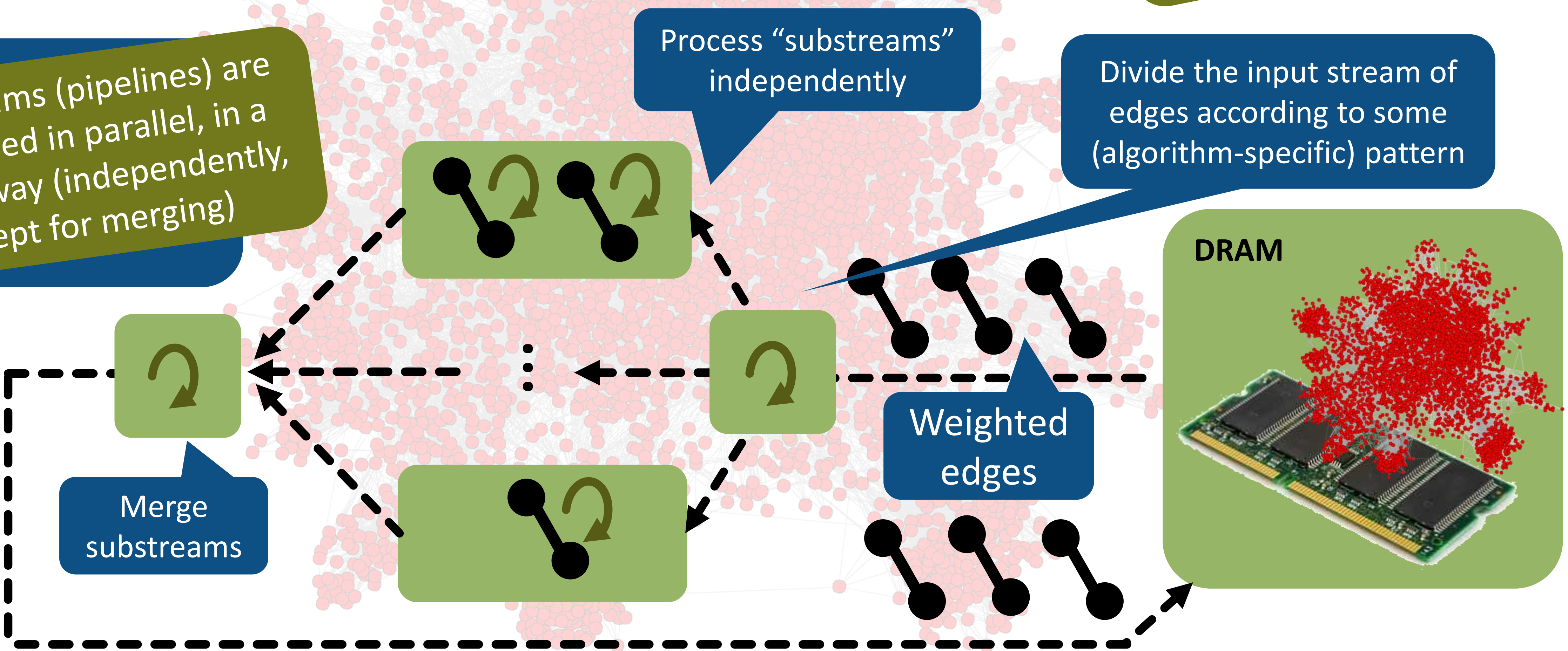
Substream-Centric Graph Processing

A new paradigm for processing graphs

It enhances edge-centric streaming approaches

Use some form of streaming (aka **edge-centric**); we can use pipelining efficiently ("streaming ≈ pipelining")

Substreams (pipelines) are processed in parallel, in a simple way (independently, except for merging)



Substream-Centric Graph Processing

A new paradigm for processing graphs

It enhances edge-centric streaming app aches

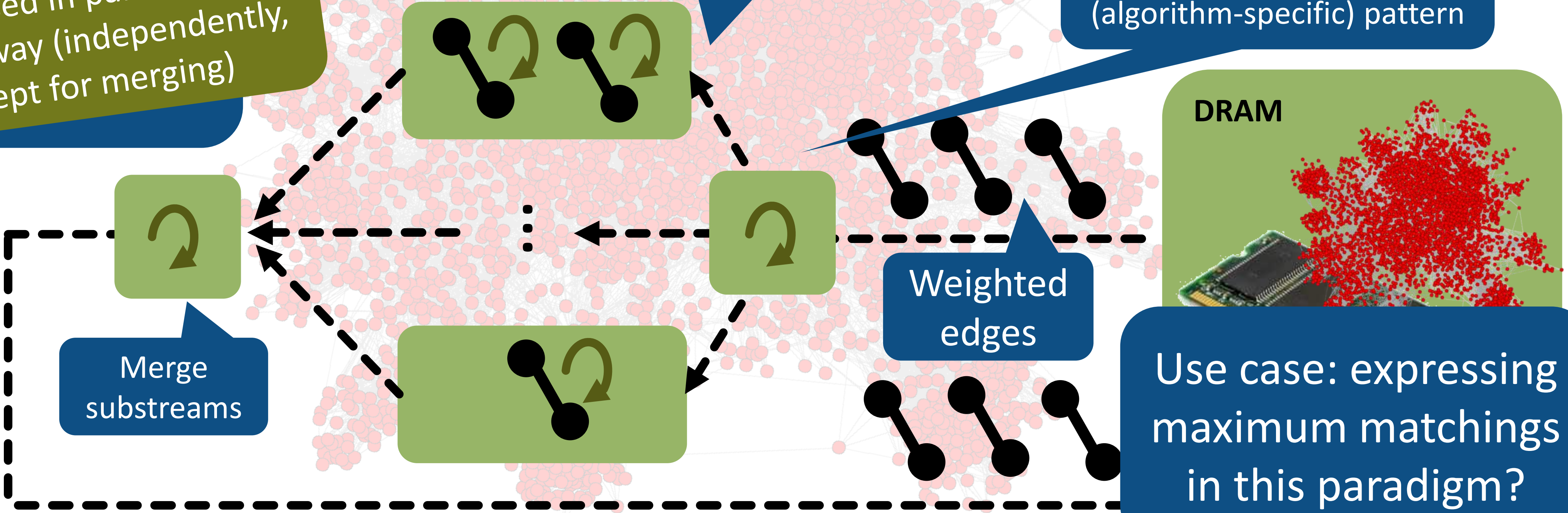
Use some form of streaming (aka **edge-centric**); we can use pipelining efficiently (“streaming ≈ pipelining”)

Also, it enables (tunable) approximation and a (tunable) number of passes

streams” independently

Substreams (pipelines) are processed in parallel, in a simple way (independently, except for merging)

Divide the input stream of edges according to some (algorithm-specific) pattern



Use case: expressing maximum matchings in this paradigm?

Substream-Centric Graph Processing

A new paradigm for processing graphs

It enhances edge-centric streaming approaches

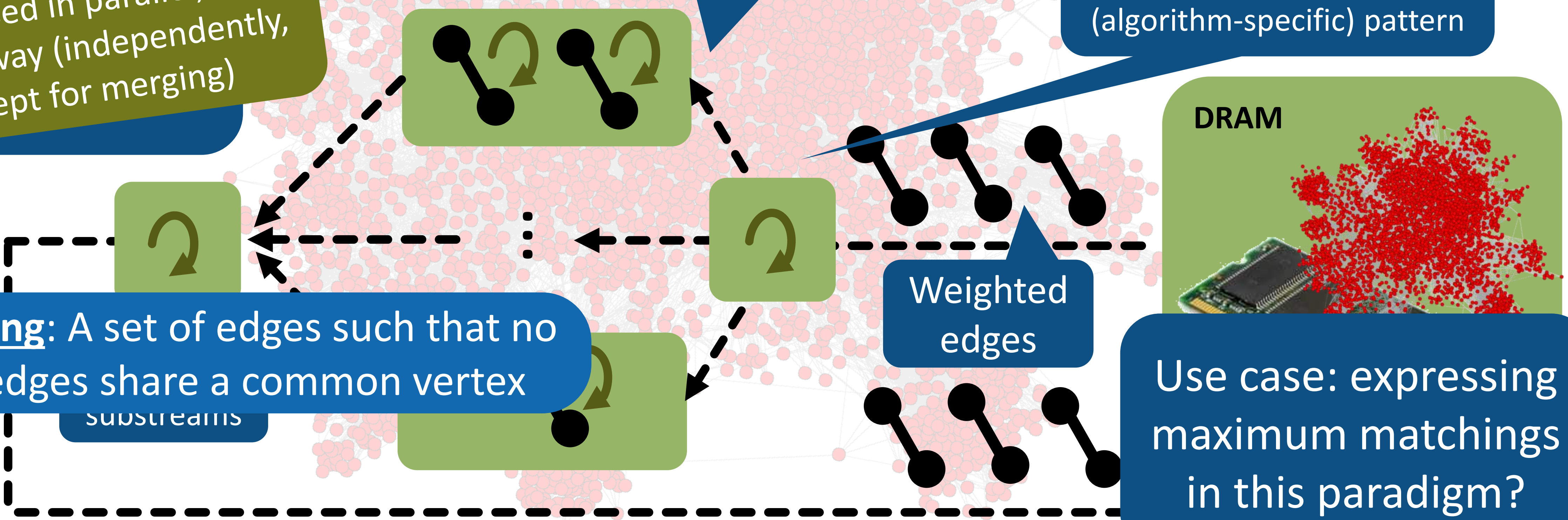
Use some form of streaming (aka **edge-centric**); we can use pipelining efficiently (“streaming ≈ pipelining”)

Also, it enables (tunable) approximation and a (tunable) number of passes

Substreams (pipelines) are processed in parallel, in a simple way (independently, except for merging)

…streams” independently

Divide the input stream of edges according to some (algorithm-specific) pattern



Matching: A set of edges such that no two edges share a common vertex

Use case: expressing maximum matchings in this paradigm?

Substream-Centric Graph Processing

A new paradigm for processing graphs

It enhances edge-centric streaming approaches

Use some form of streaming (aka **edge-centric**); we can use pipelining efficiently ("streaming ≈ pipelining")

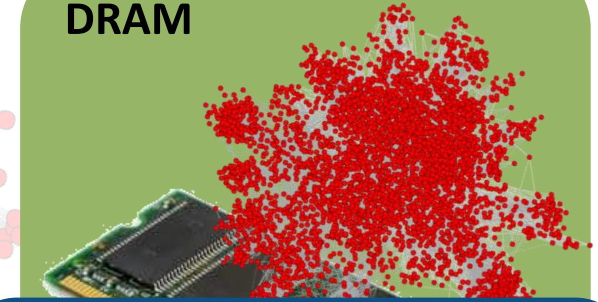
Also, it enables (tunable) approximation and a (tunable) number of passes

…streams" independently

Substreams (pipelines) are processed in parallel, in a simple way (independently, except for merging)

Divide the input stream of edges according to some (algorithm-specific) pattern

DRAM



Weighted edges

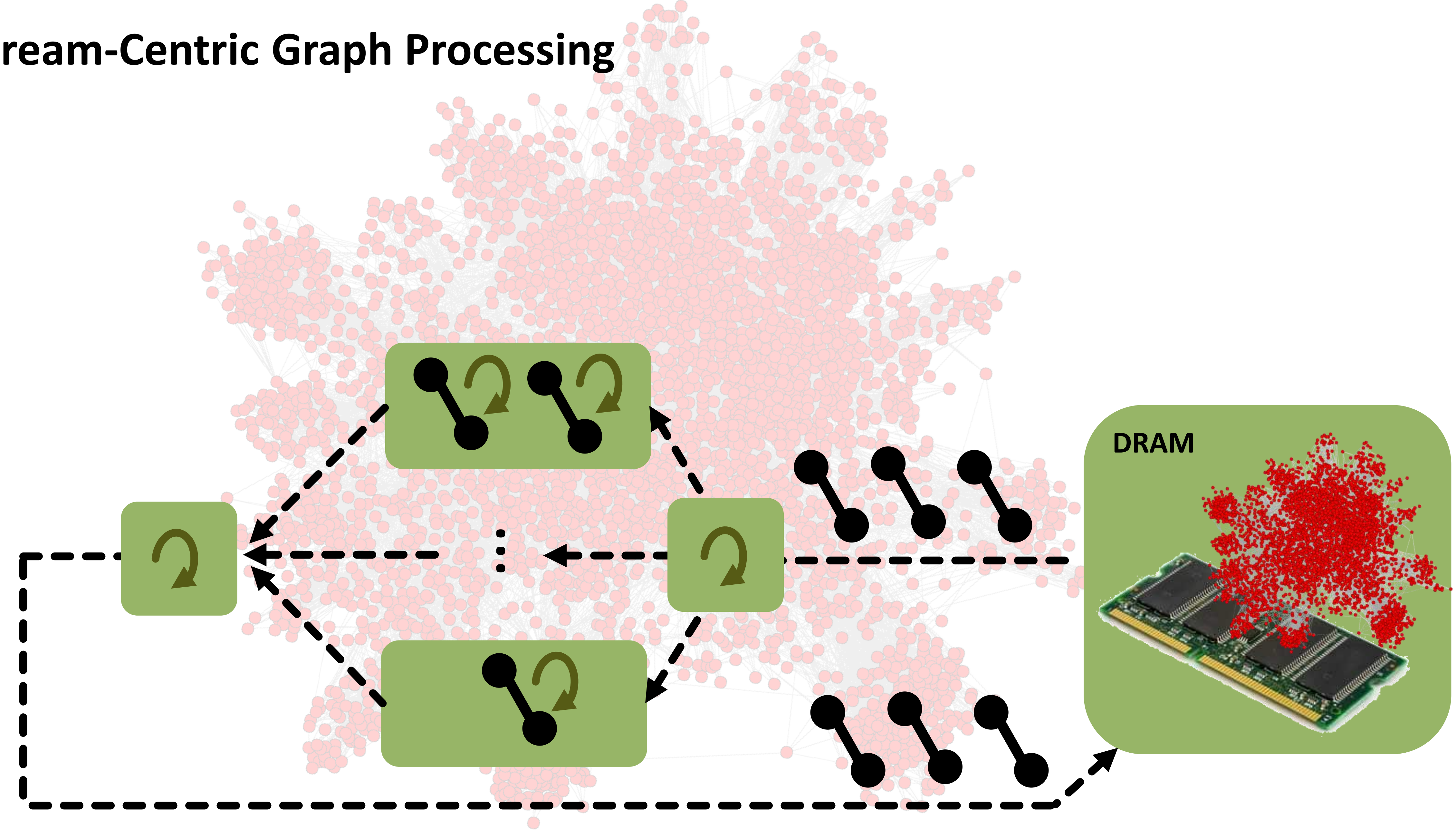
Matching: A set of edges such that no two edges share a common vertex

substreams

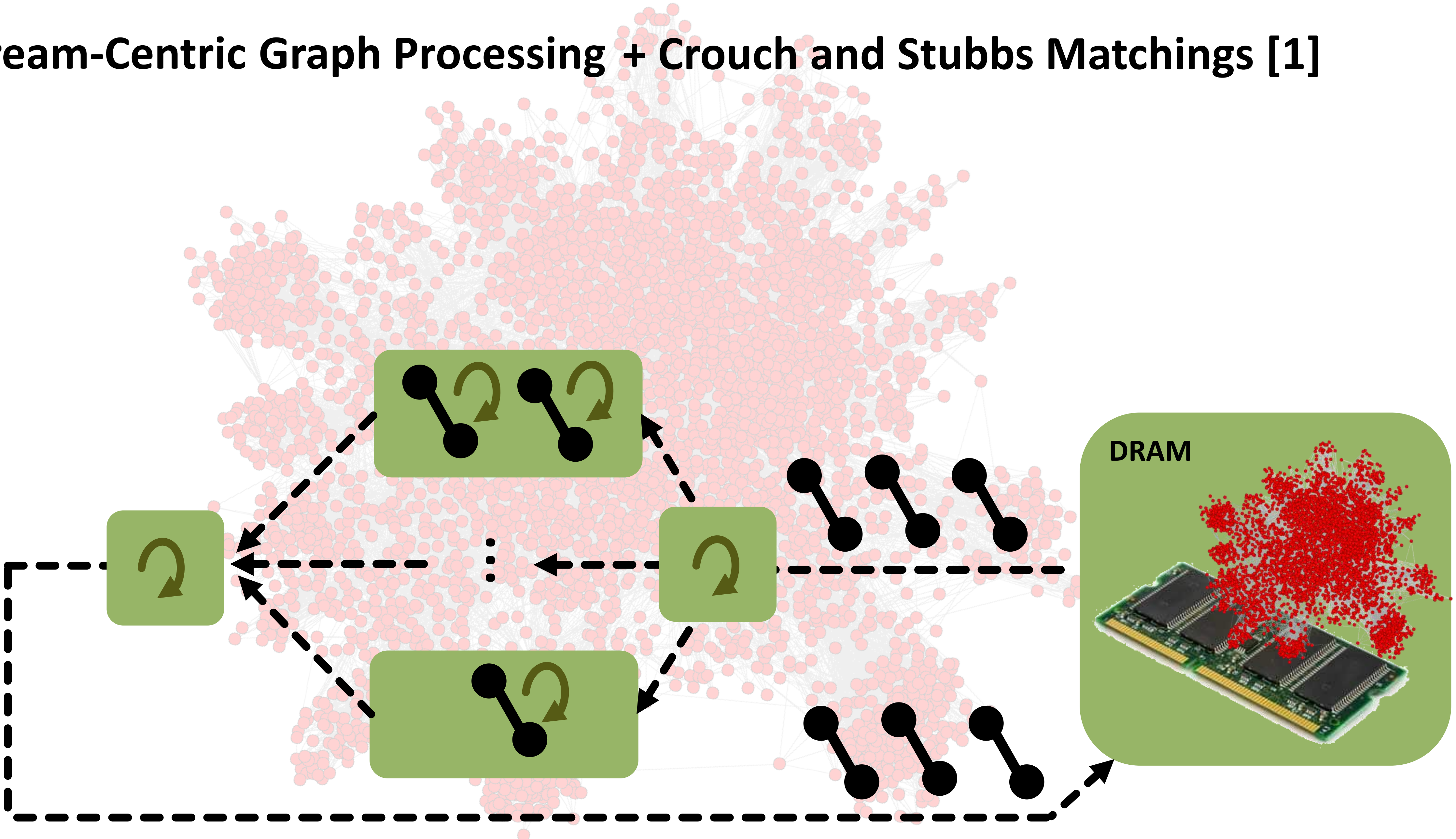
Maximum Weighted Matching (MWM): A matching such that the sum of the edge weights is maximized

Use case: expressing maximum matchings in this paradigm?

Substream-Centric Graph Processing

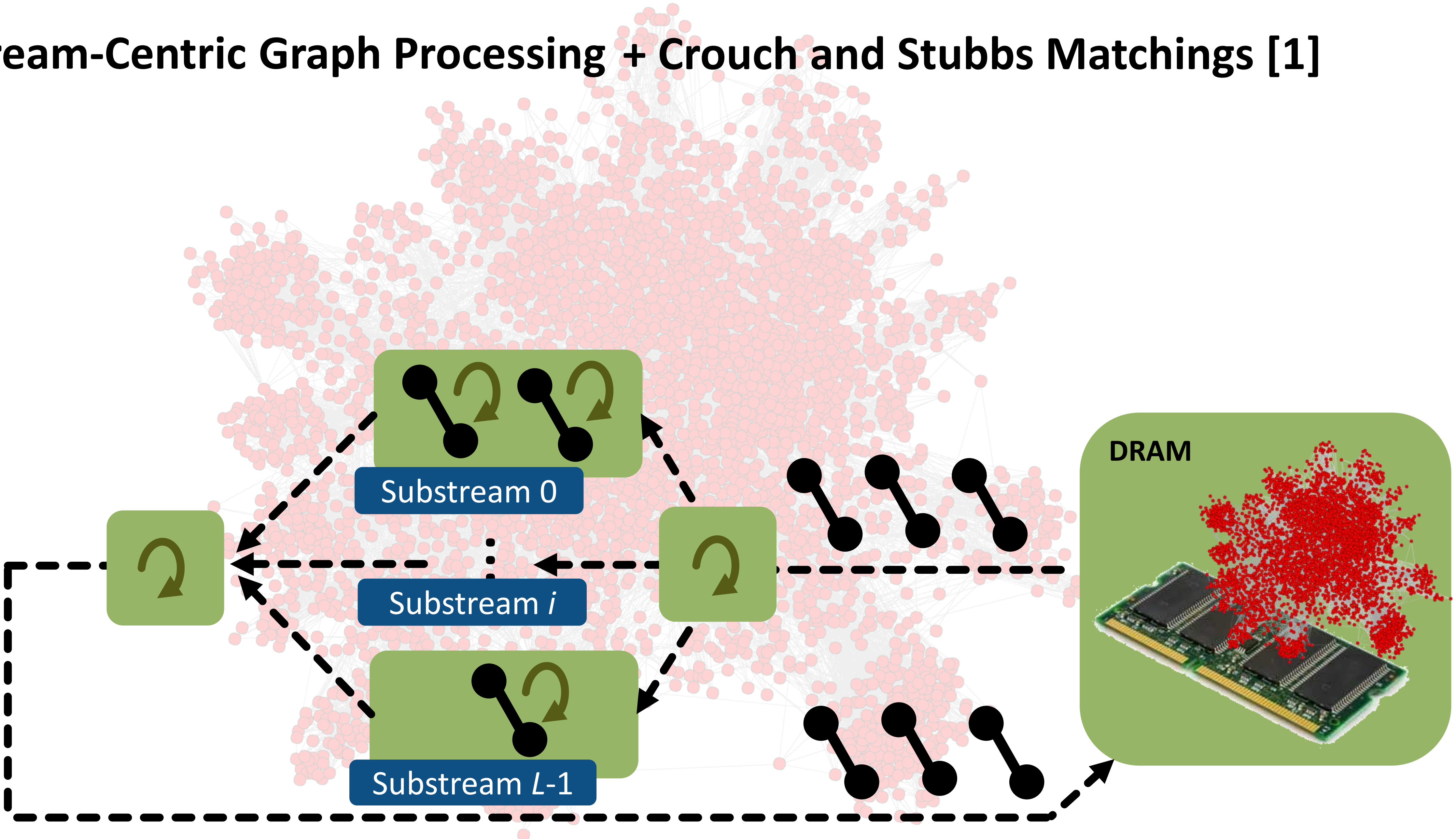


Substream-Centric Graph Processing + Crouch and Stubbs Matchings [1]



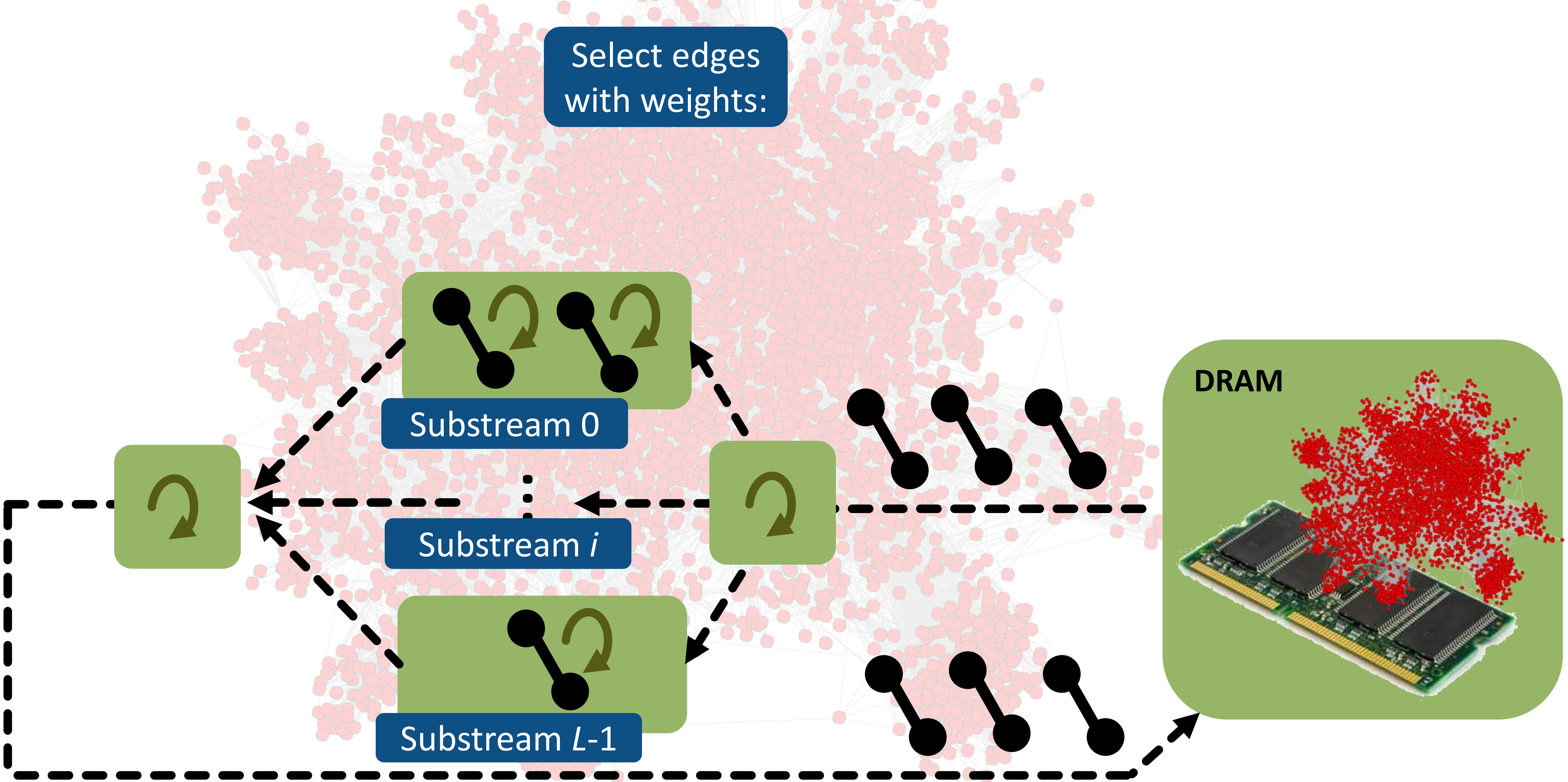
[1] M. Crouch and D. M. Stubbs. Improved streaming Algorithms for weighted Matching, via unweighted Matching. LIPIcs-Leibniz Informatics. 2014.

Substream-Centric Graph Processing + Crouch and Stubbs Matchings [1]



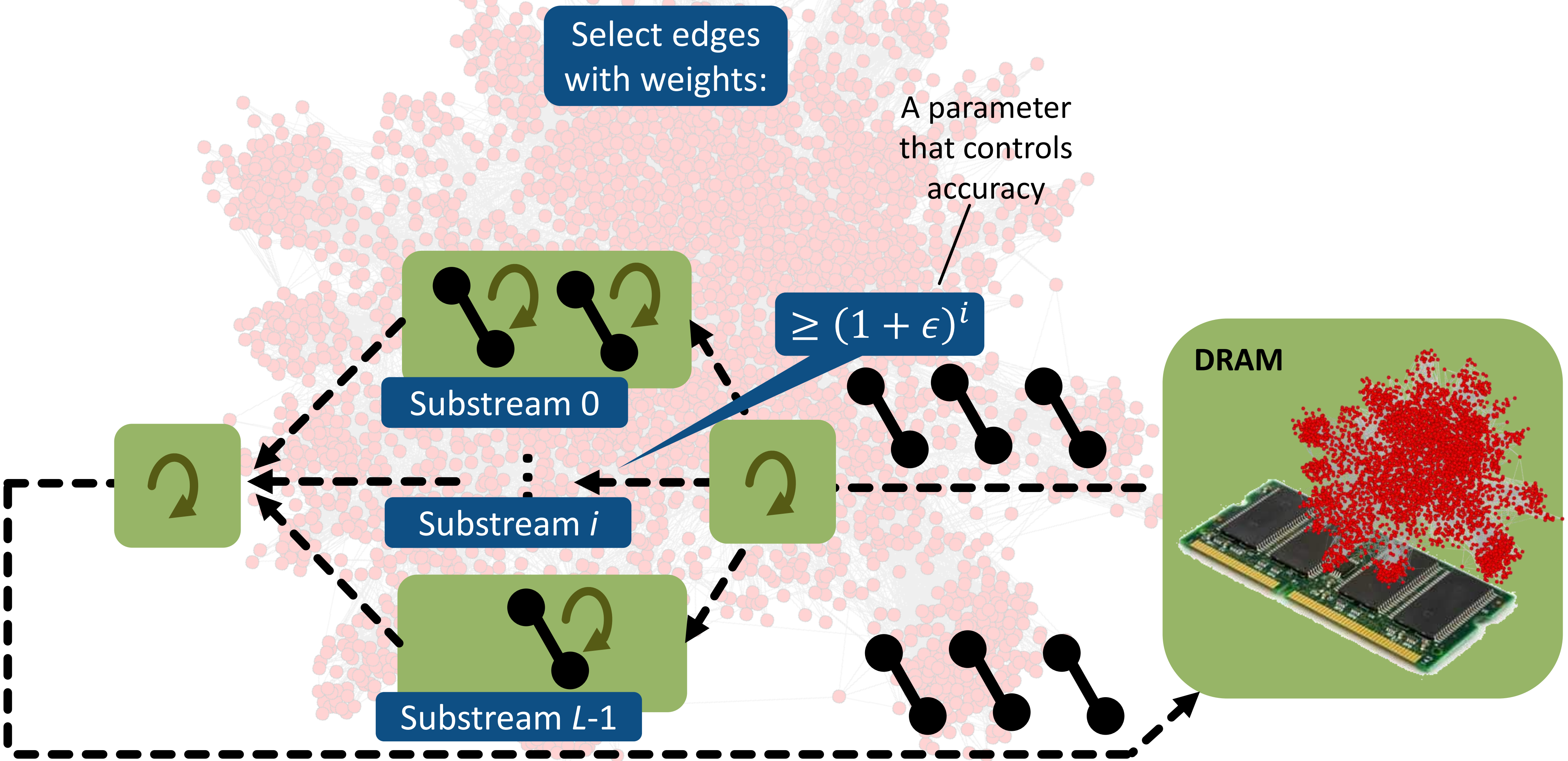
[1] M. Crouch and D. M. Stubbs. Improved streaming Algorithms for weighted Matching, via unweighted Matching. LIPIcs-Leibniz Informatics. 2014.

Substream-Centric Graph Processing + Crouch and Stubbs Matchings [1]



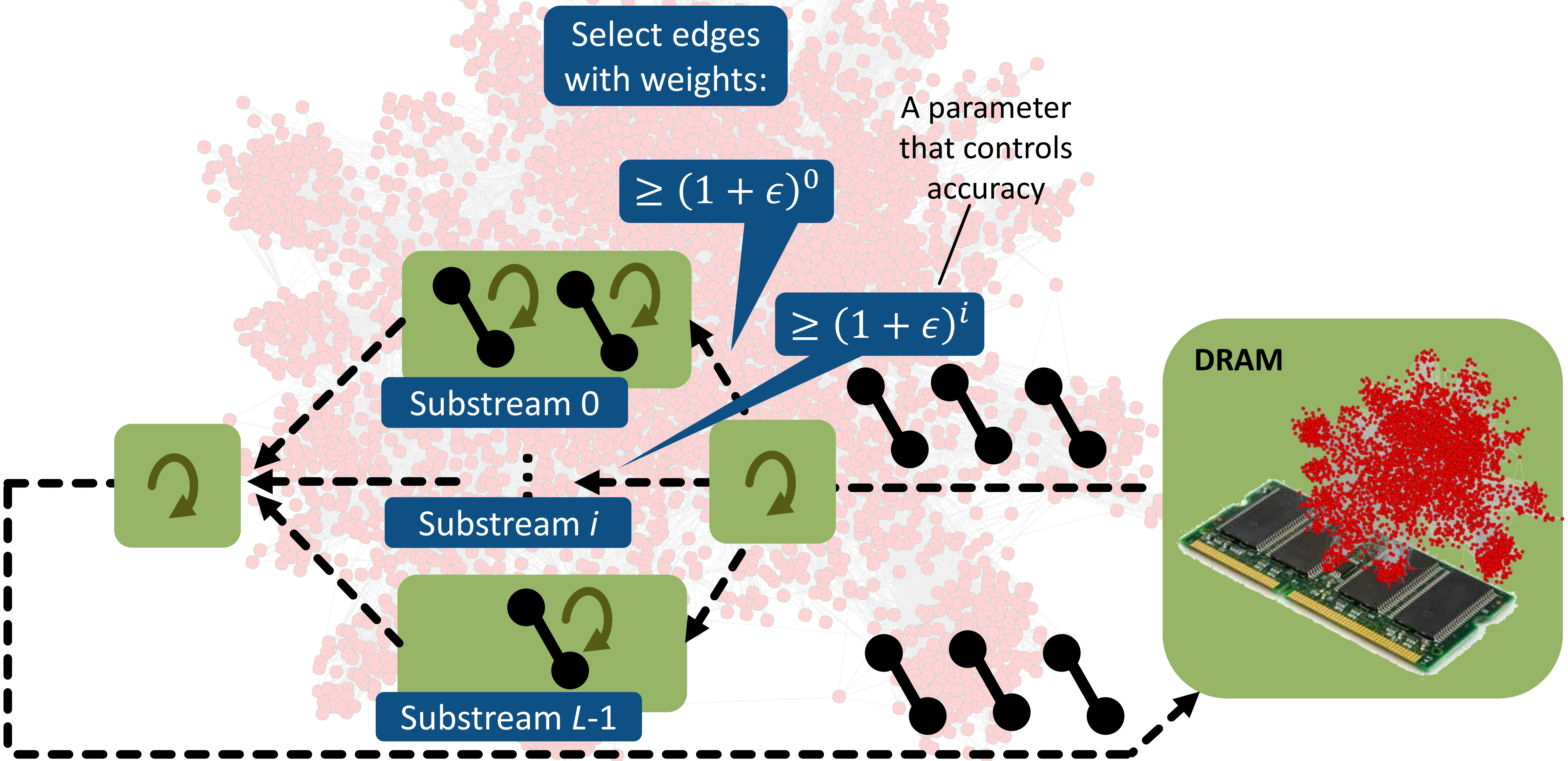
[1] M. Crouch and D. M. Stubbs. Improved streaming Algorithms for weighted Matching, via unweighted Matching. LIPIcs-Leibniz Informatics. 2014.

Substream-Centric Graph Processing + Crouch and Stubbs Matchings [1]



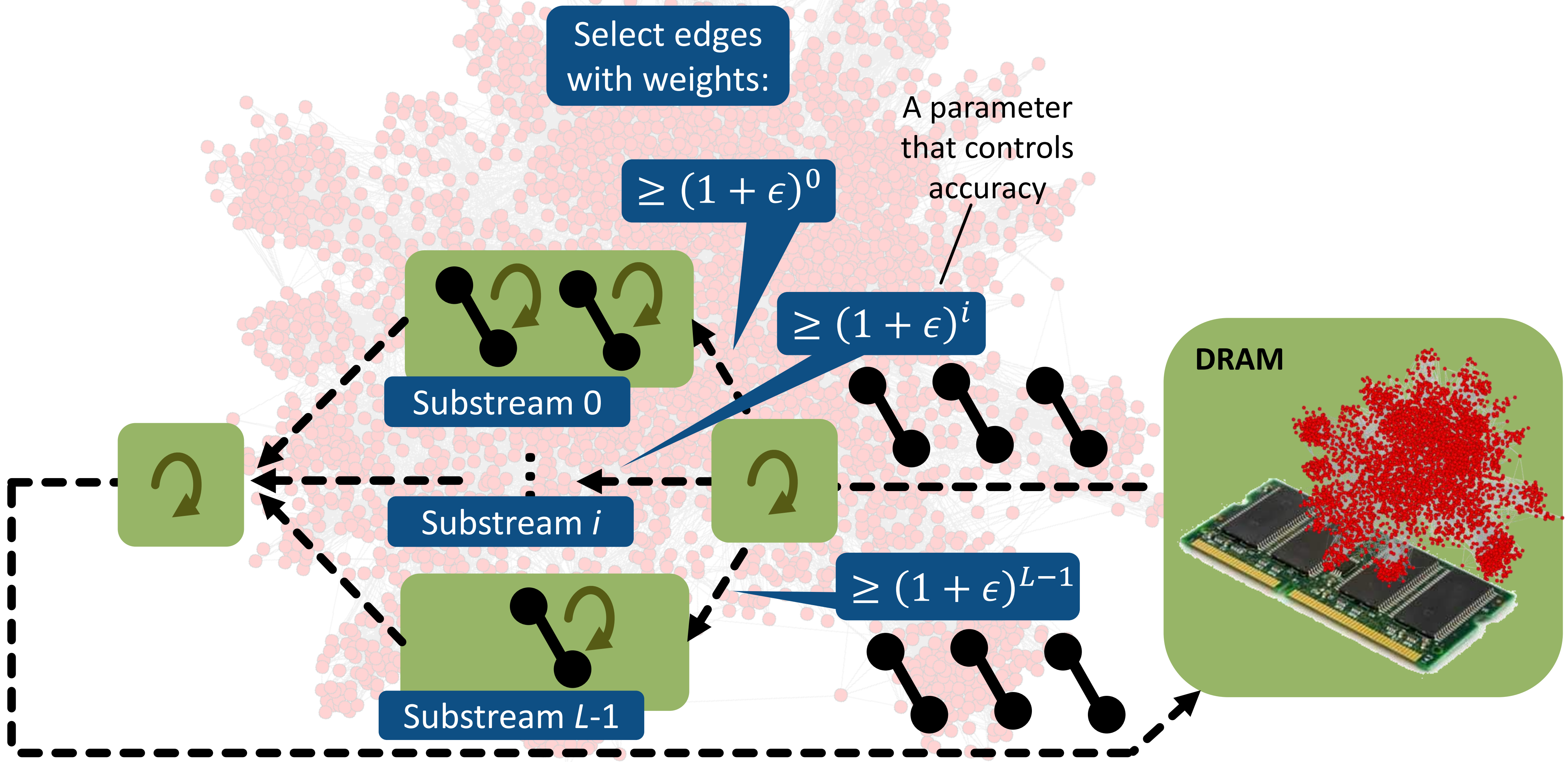
[1] M. Crouch and D. M. Stubbs. Improved streaming Algorithms for weighted Matching, via unweighted Matching. LIPIcs-Leibniz Informatics. 2014.

Substream-Centric Graph Processing + Crouch and Stubbs Matchings [1]



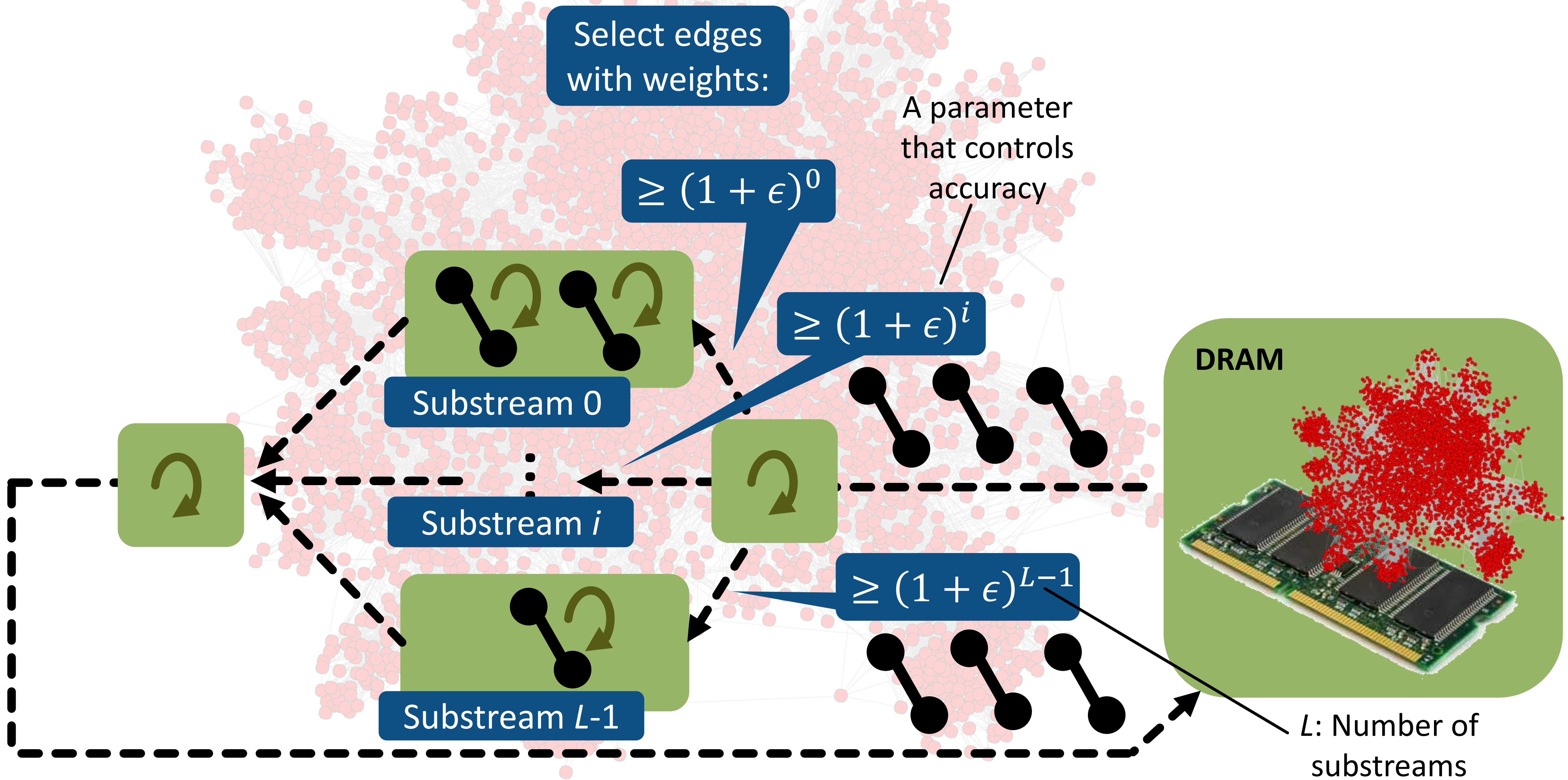
[1] M. Crouch and D. M. Stubbs. Improved streaming Algorithms for weighted Matching, via unweighted Matching. LIPIcs-Leibniz Informatics. 2014.

Substream-Centric Graph Processing + Crouch and Stubbs Matchings [1]



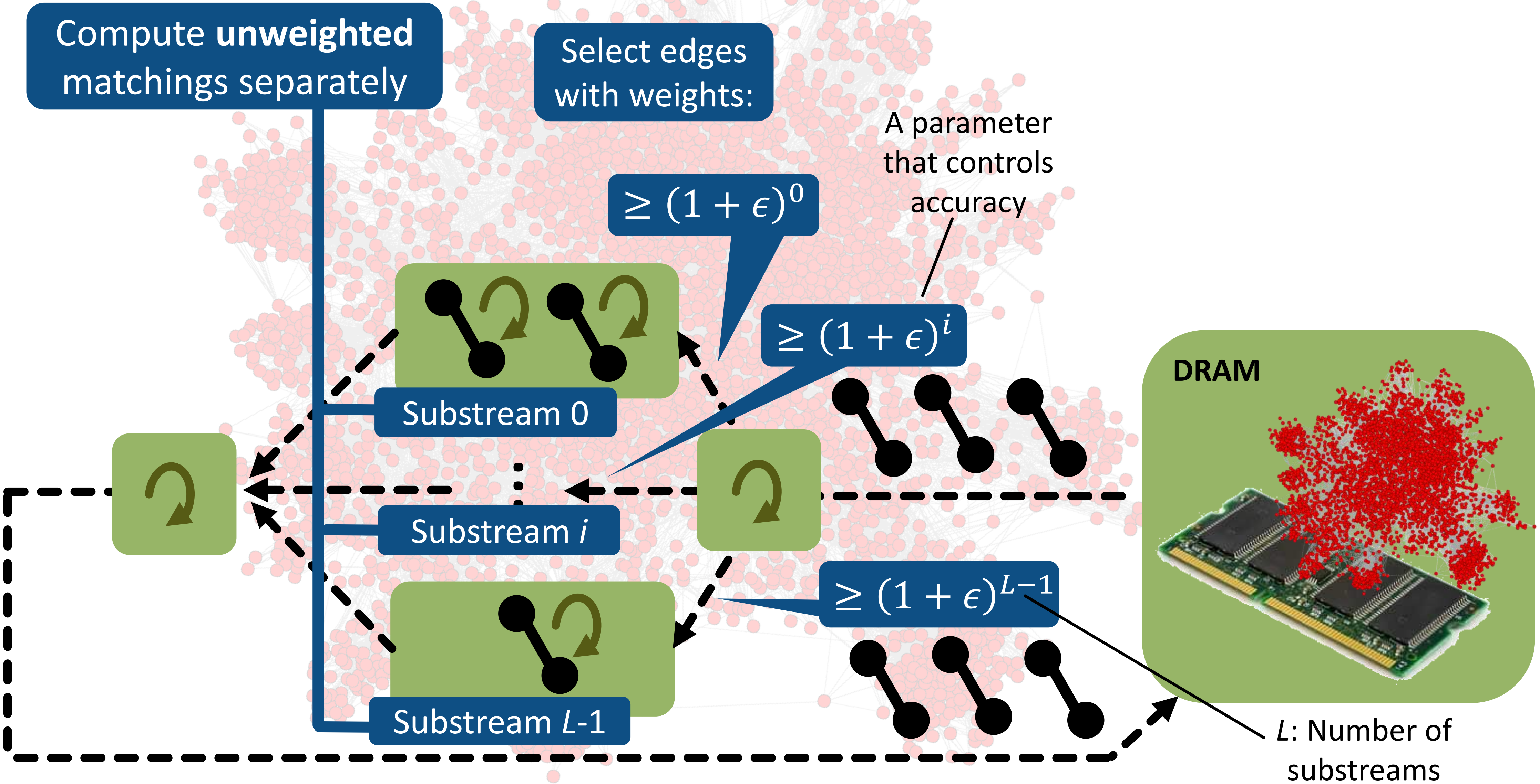
[1] M. Crouch and D. M. Stubbs. Improved streaming Algorithms for weighted Matching, via unweighted Matching. LIPIcs-Leibniz Informatics. 2014.

Substream-Centric Graph Processing + Crouch and Stubbs Matchings [1]



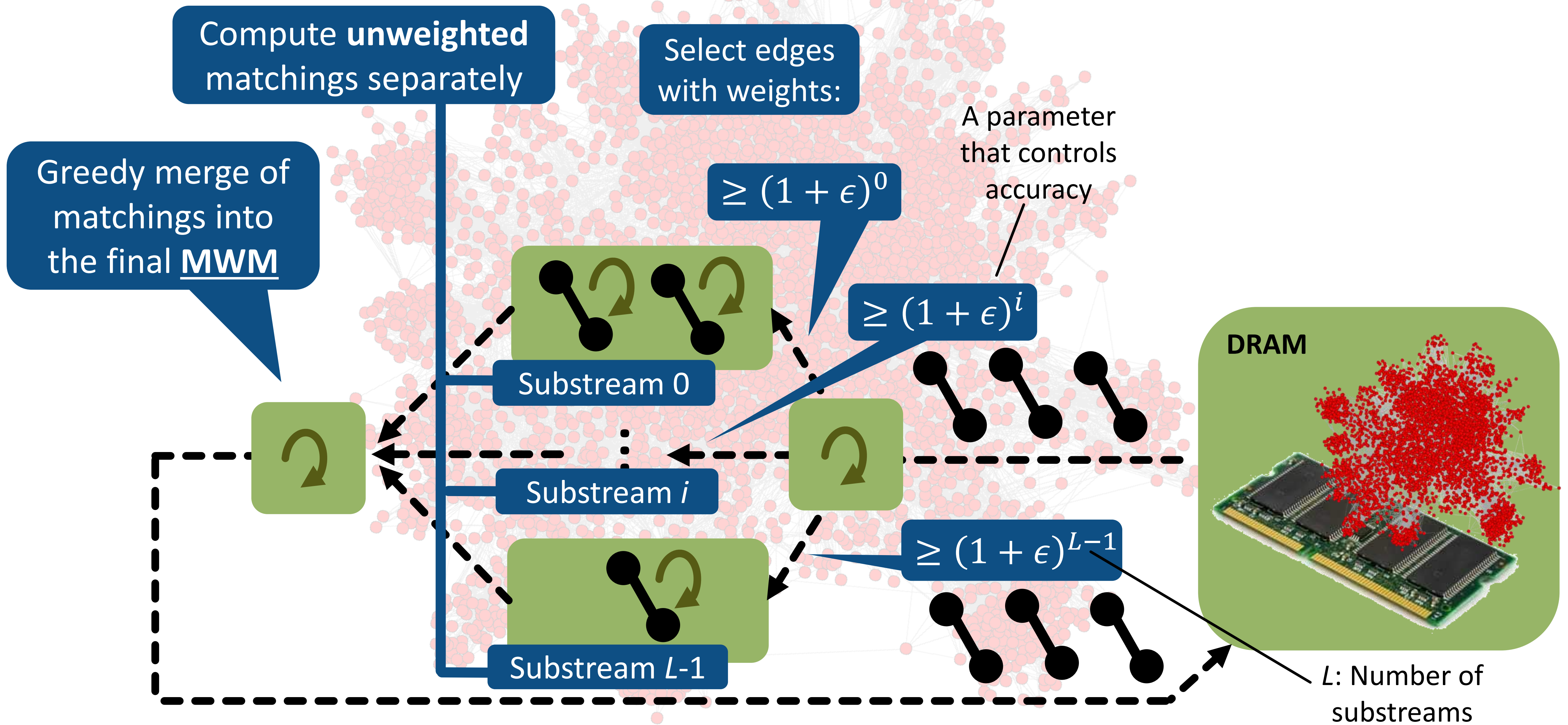
[1] M. Crouch and D. M. Stubbs. Improved streaming Algorithms for weighted Matching, via unweighted Matching. LIPIcs-Leibniz Informatics. 2014.

Substream-Centric Graph Processing + Crouch and Stubbs Matchings [1]



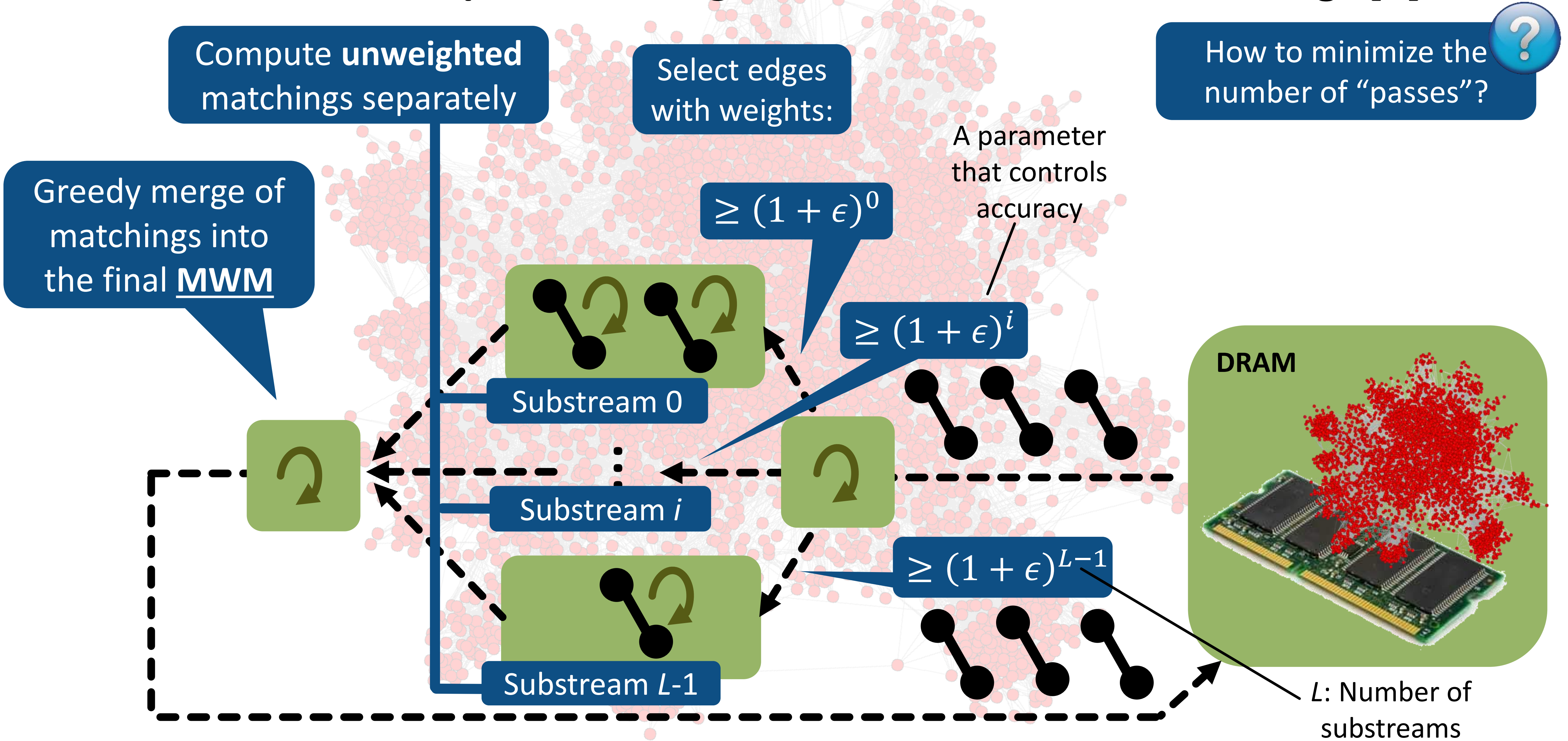
[1] M. Crouch and D. M. Stubbs. Improved streaming Algorithms for weighted Matching, via unweighted Matching. LIPIcs-Leibniz Informatics. 2014.

Substream-Centric Graph Processing + Crouch and Stubbs Matchings [1]



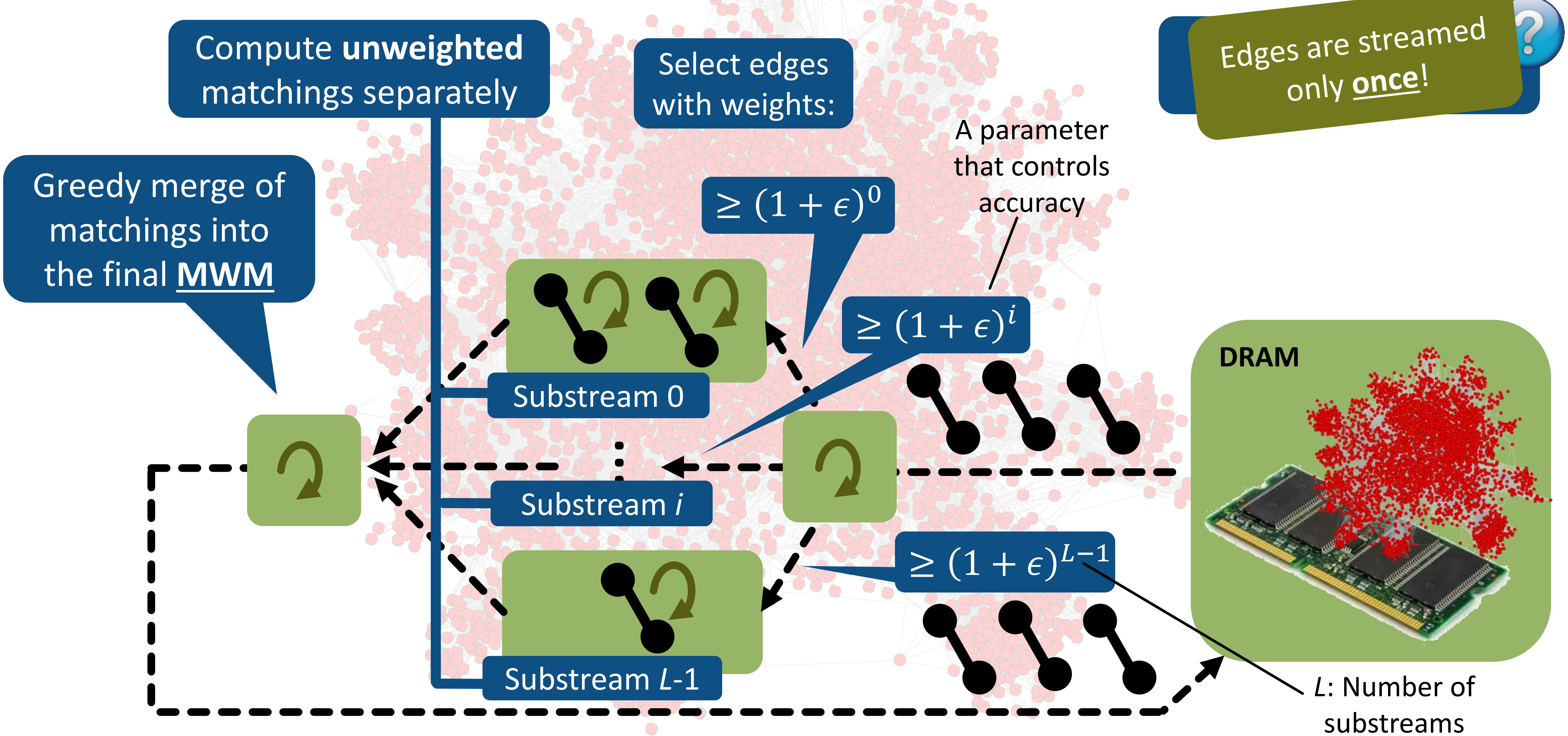
[1] M. Crouch and D. M. Stubbs. Improved streaming Algorithms for weighted Matching, via unweighted Matching. LIPIcs-Leibniz Informatics. 2014.

Substream-Centric Graph Processing + Crouch and Stubbs Matchings [1]



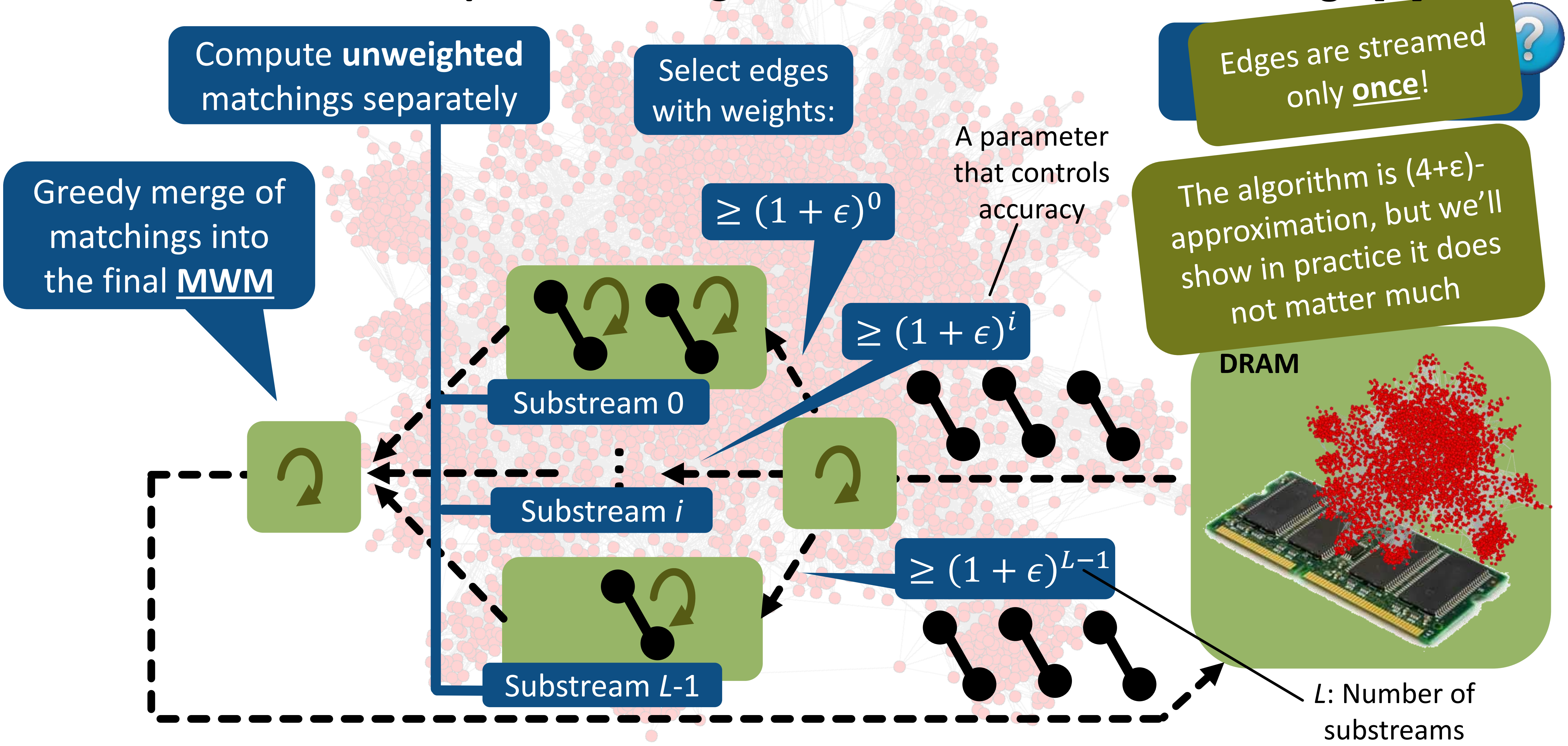
[1] M. Crouch and D. M. Stubbs. Improved streaming Algorithms for weighted Matching, via unweighted Matching. LIPIcs-Leibniz Informatics. 2014.

Substream-Centric Graph Processing + Crouch and Stubbs Matchings [1]



[1] M. Crouch and D. M. Stubbs. Improved streaming Algorithms for weighted Matching, via unweighted Matching. LIPIcs-Leibniz Informatics. 2014.

Substream-Centric Graph Processing + Crouch and Stubbs Matchings [1]

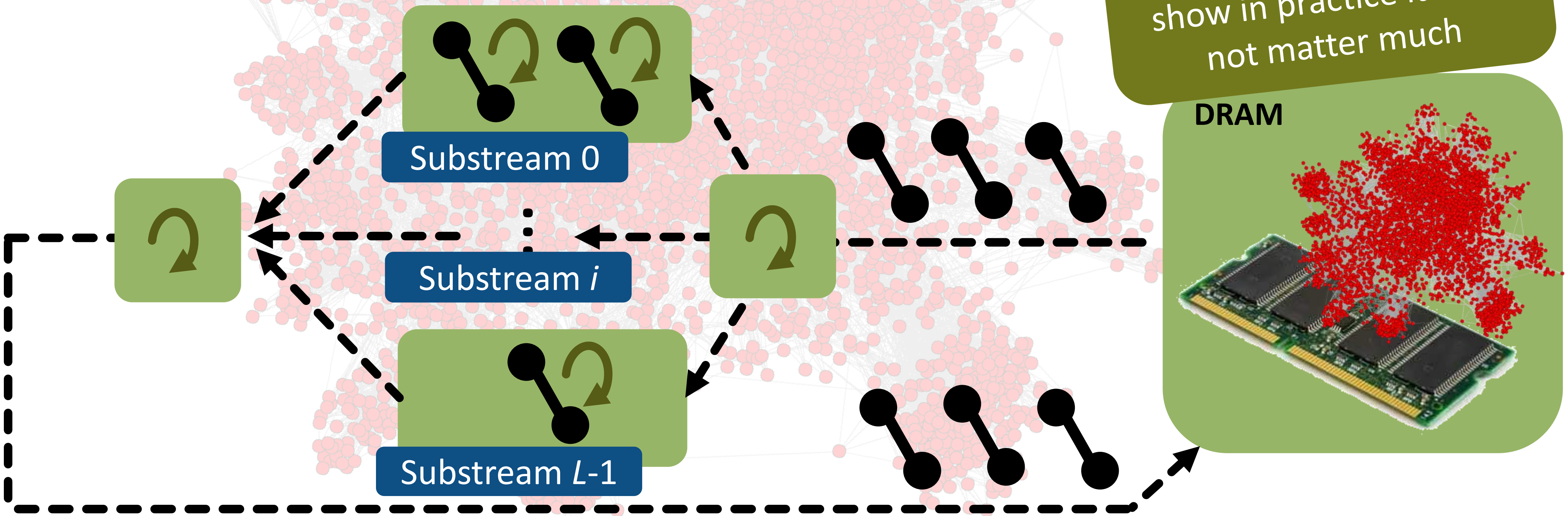


[1] M. Crouch and D. M. Stubbs. Improved streaming Algorithms for weighted Matching, via unweighted Matching. LIPIcs-Leibniz Informatics. 2014.

Substream-Centric Graph Processing + Crouch and Stubbs Matchings [1]

Edges are streamed only **once**!

The algorithm is (4+ε)-approximation, but we'll show in practice it does not matter much

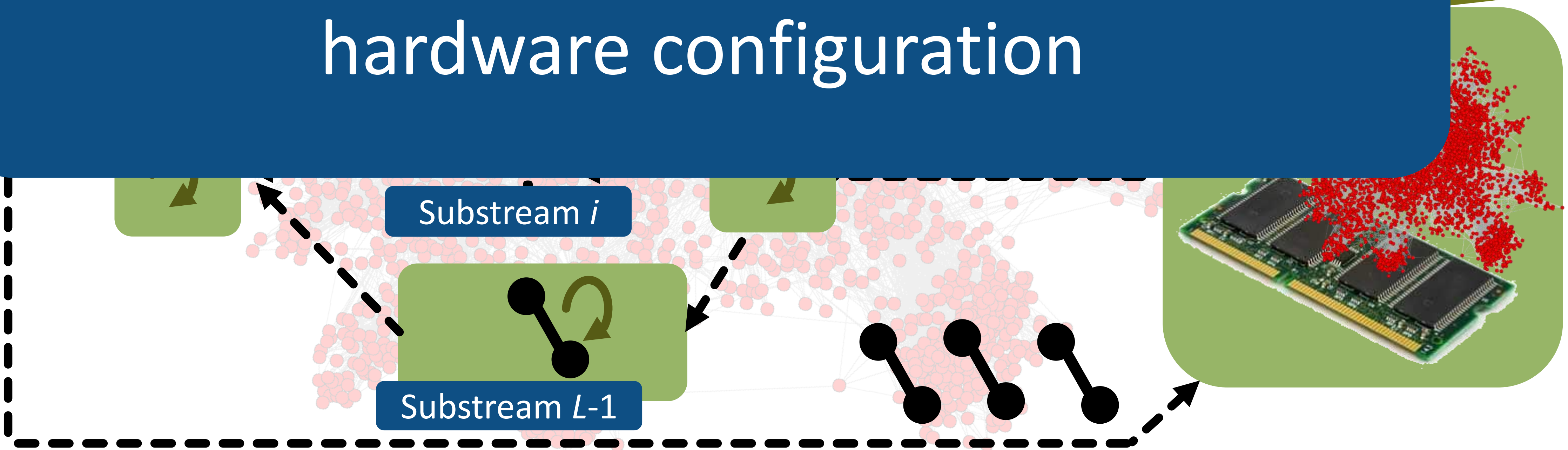


[1] M. Crouch and D. M. Stubbs. Improved streaming Algorithms for weighted Matching, via unweighted Matching. LIPIcs-Leibniz Informatics. 2014.

Substream-Centric Graph Processing + Crouch and Stubbs Matchings [1]

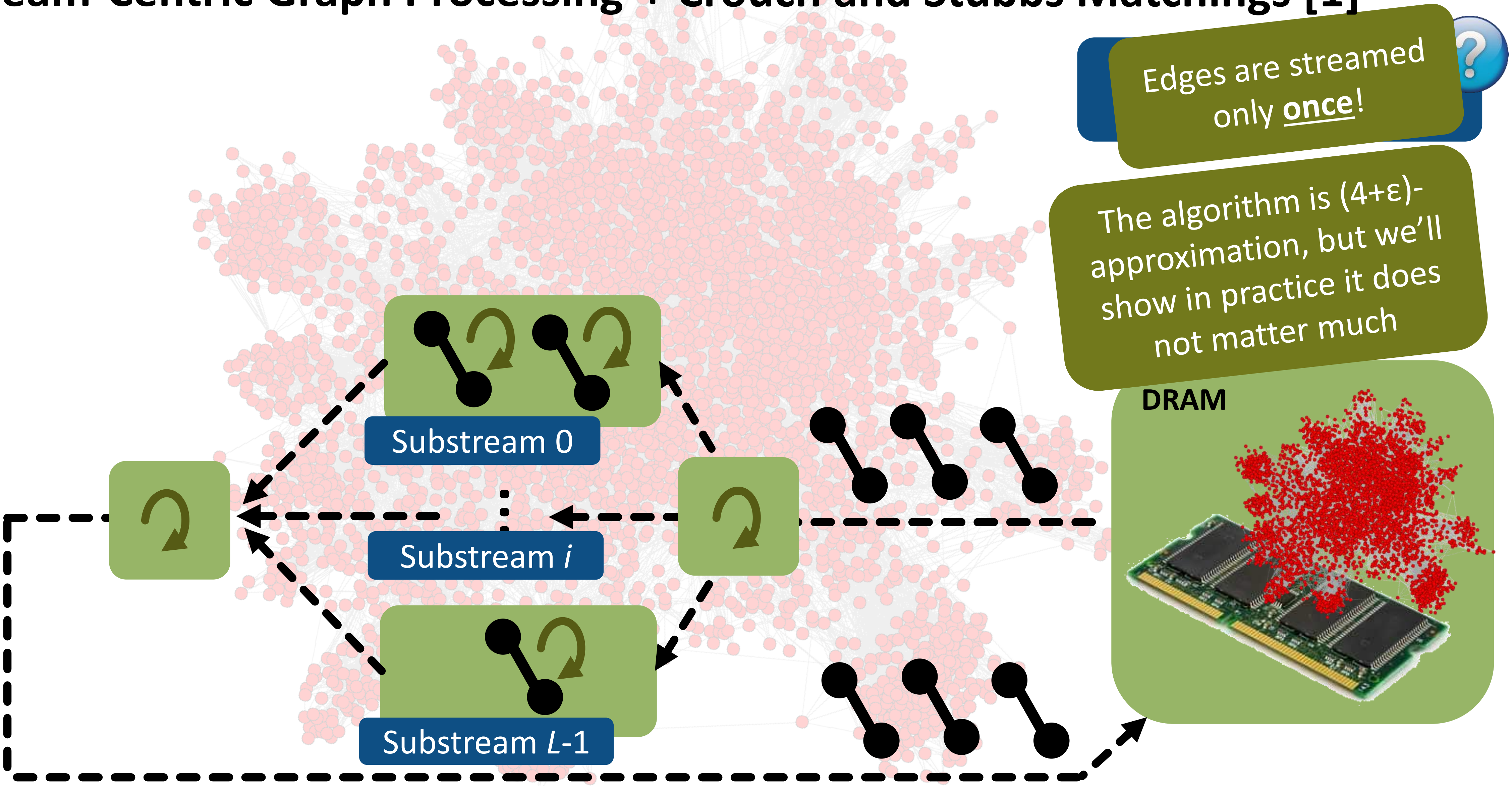
Edges are streamed only **once**!

Mapping the algorithm to the „right” hardware configuration



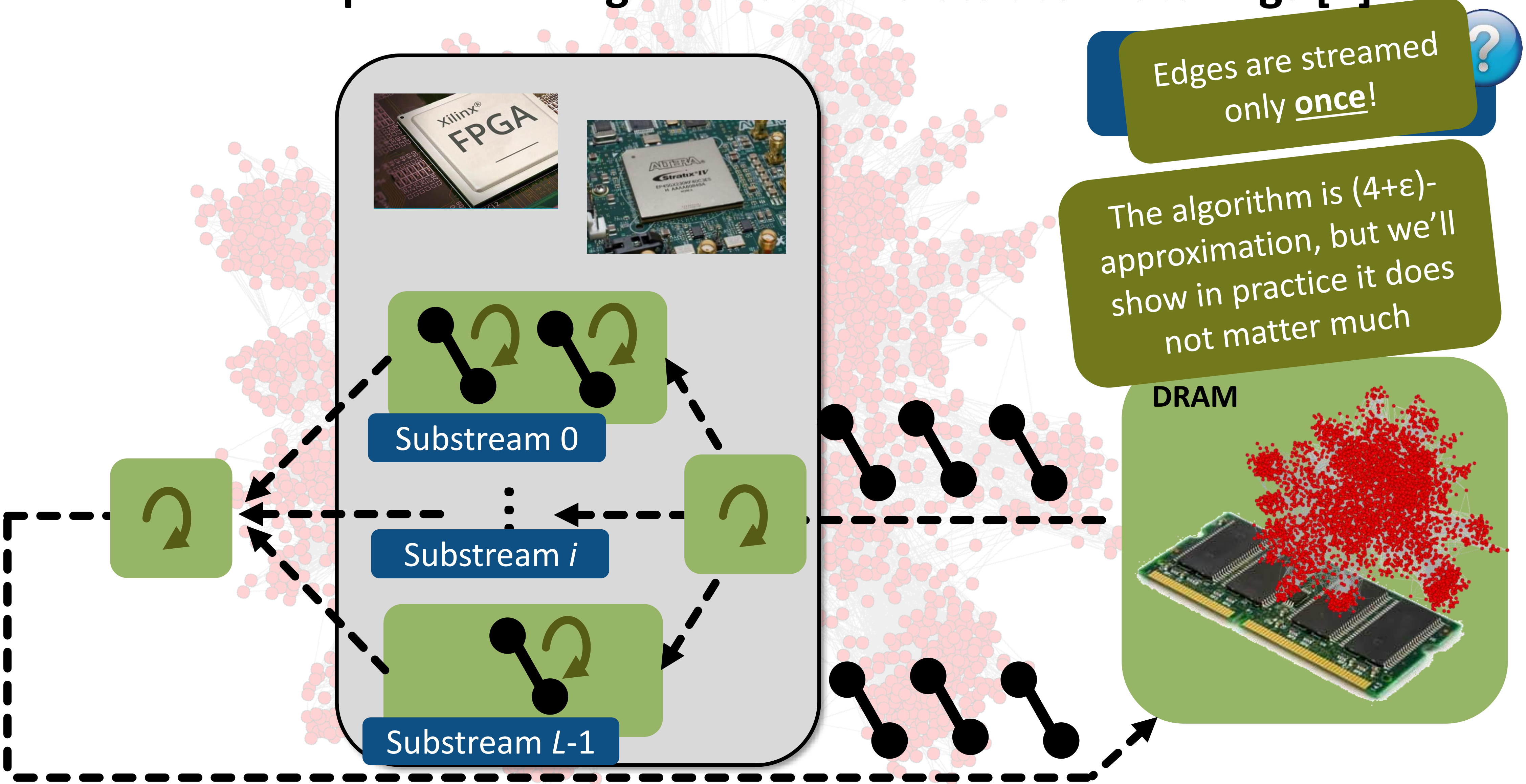
[1] M. Crouch and D. M. Stubbs. Improved streaming Algorithms for weighted Matching, via unweighted Matching. LIPIcs-Leibniz Informatics. 2014.

Substream-Centric Graph Processing + Crouch and Stubbs Matchings [1]



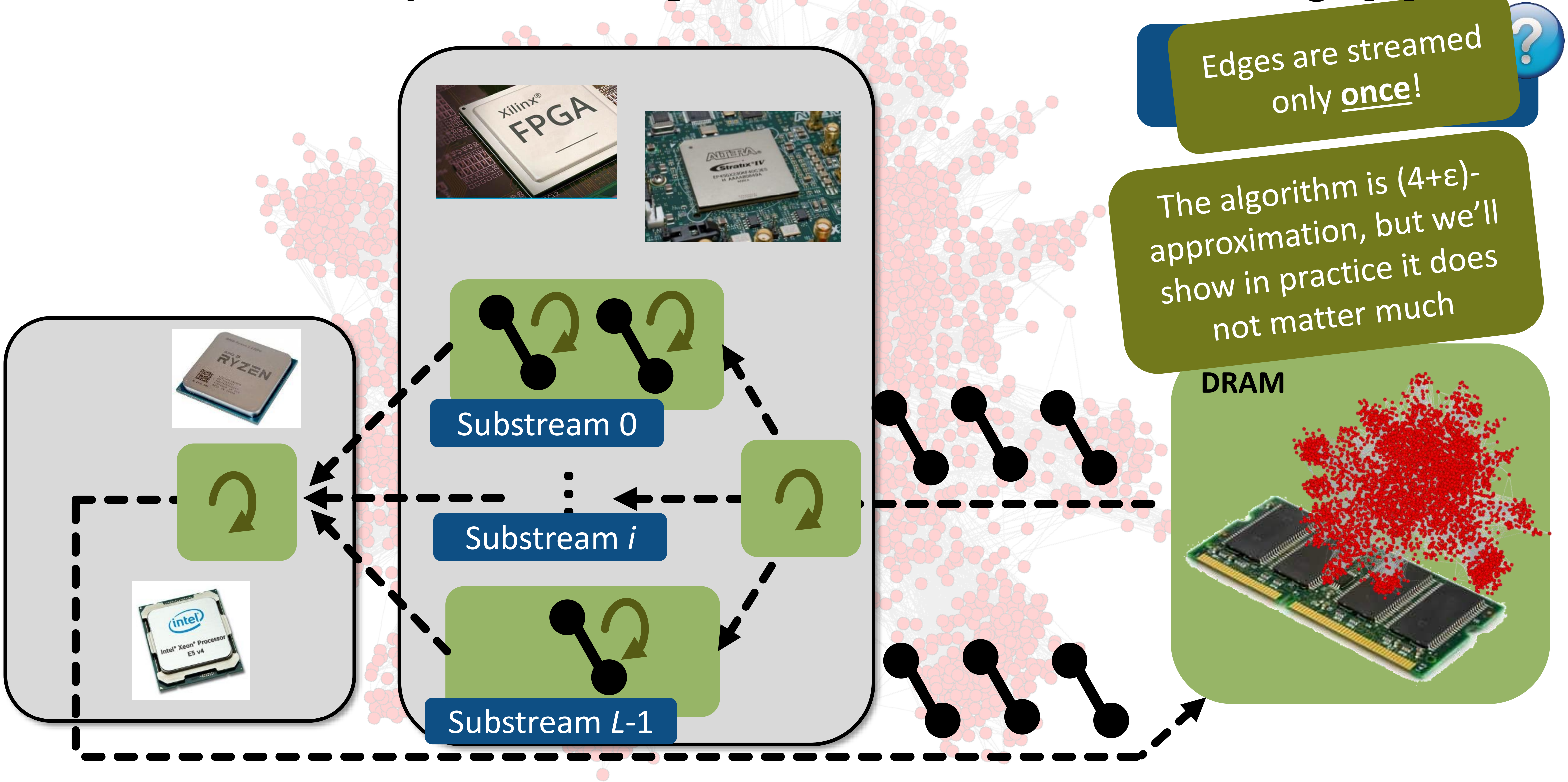
[1] M. Crouch and D. M. Stubbs. Improved streaming Algorithms for weighted Matching, via unweighted Matching. LIPIcs-Leibniz Informatics. 2014.

Substream-Centric Graph Processing + Crouch and Stubbs Matchings [1]



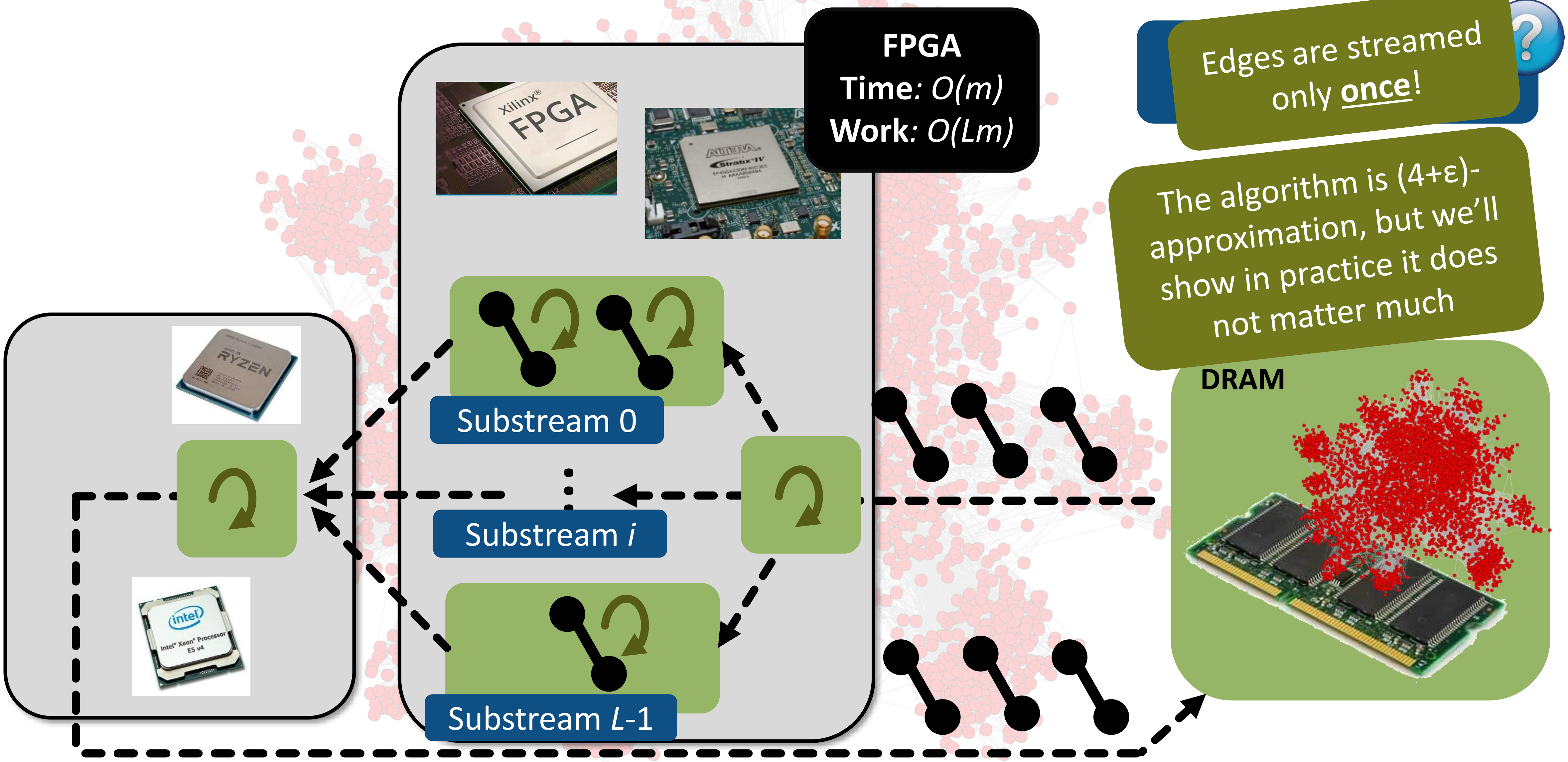
[1] M. Crouch and D. M. Stubbs. Improved streaming Algorithms for weighted Matching, via unweighted Matching. LIPIcs-Leibniz Informatics. 2014.

Substream-Centric Graph Processing + Crouch and Stubbs Matchings [1]



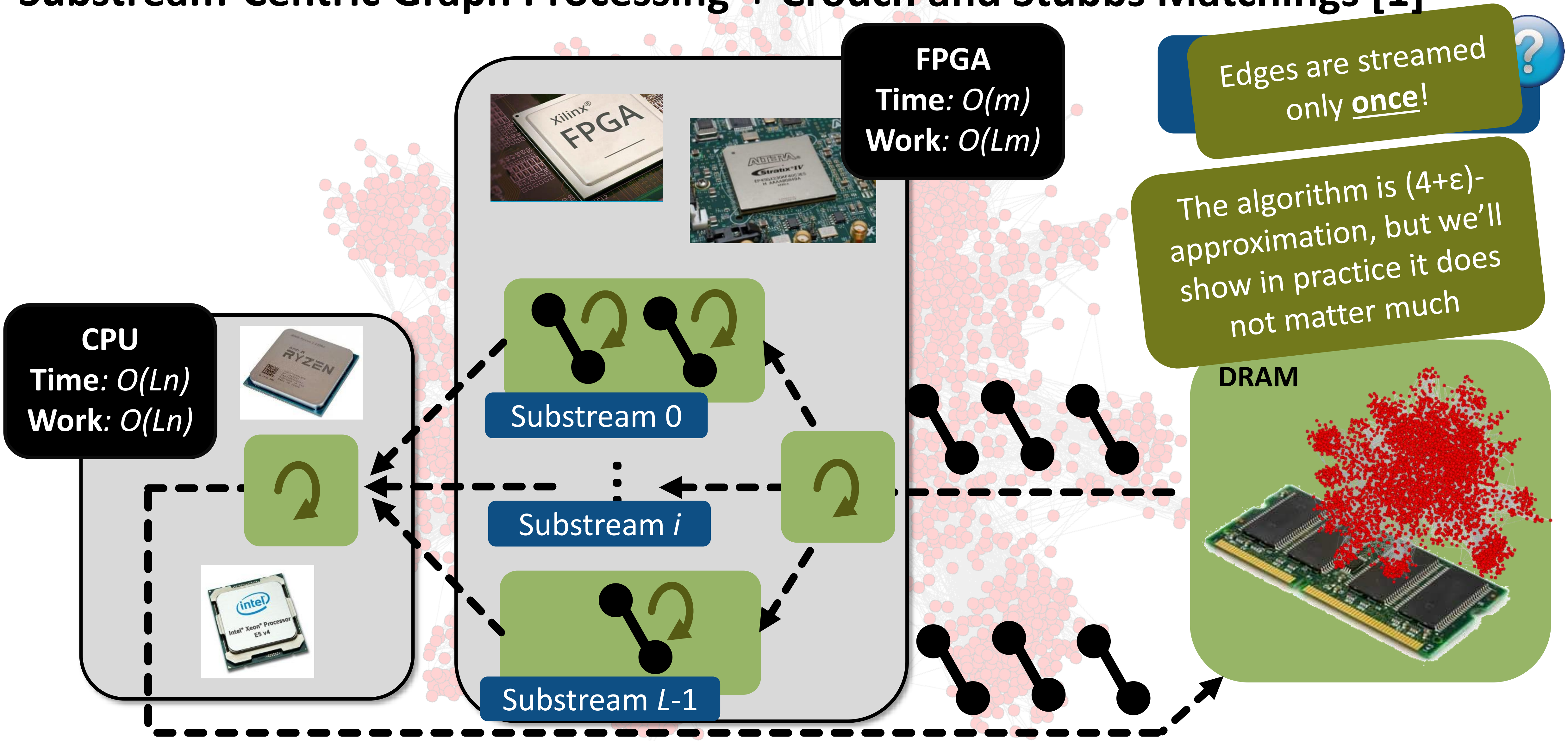
[1] M. Crouch and D. M. Stubbs. Improved streaming Algorithms for weighted Matching, via unweighted Matching. LIPIcs-Leibniz Informatics. 2014.

Substream-Centric Graph Processing + Crouch and Stubbs Matchings [1]



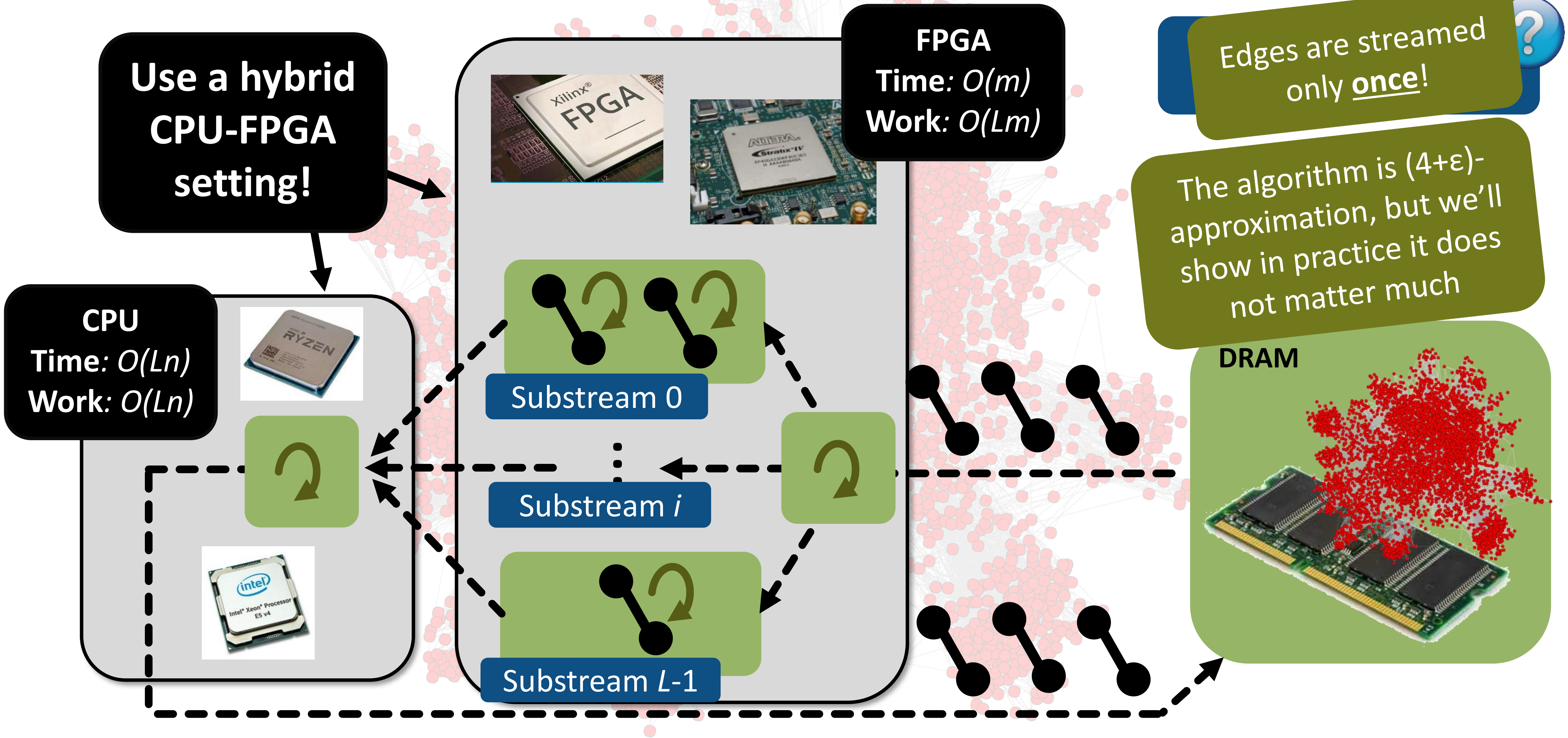
[1] M. Crouch and D. M. Stubbs. Improved streaming Algorithms for weighted Matching, via unweighted Matching. LIPIcs-Leibniz Informatics. 2014.

Substream-Centric Graph Processing + Crouch and Stubbs Matchings [1]



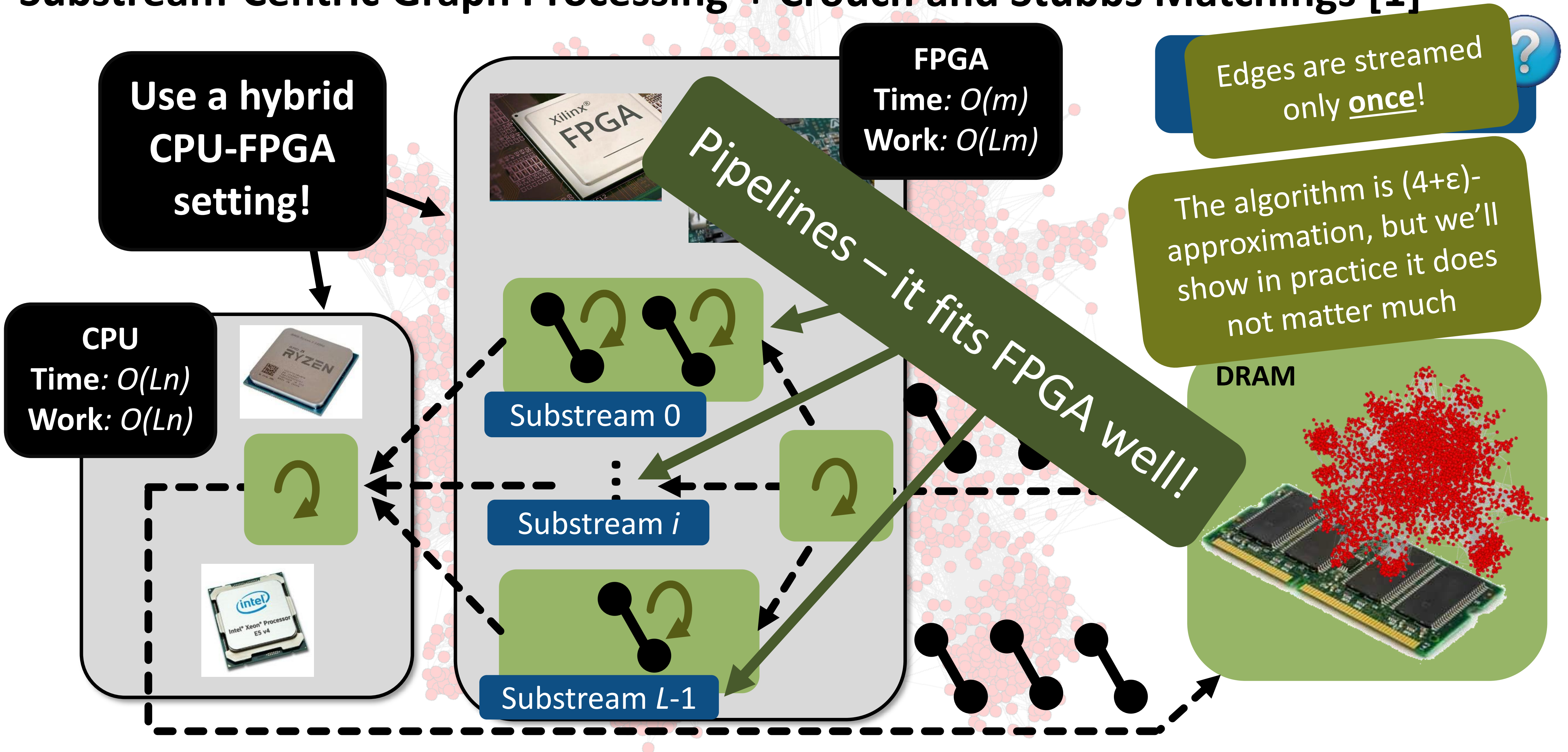
[1] M. Crouch and D. M. Stubbs. Improved streaming Algorithms for weighted Matching, via unweighted Matching. LIPIcs-Leibniz Informatics. 2014.

Substream-Centric Graph Processing + Crouch and Stubbs Matchings [1]



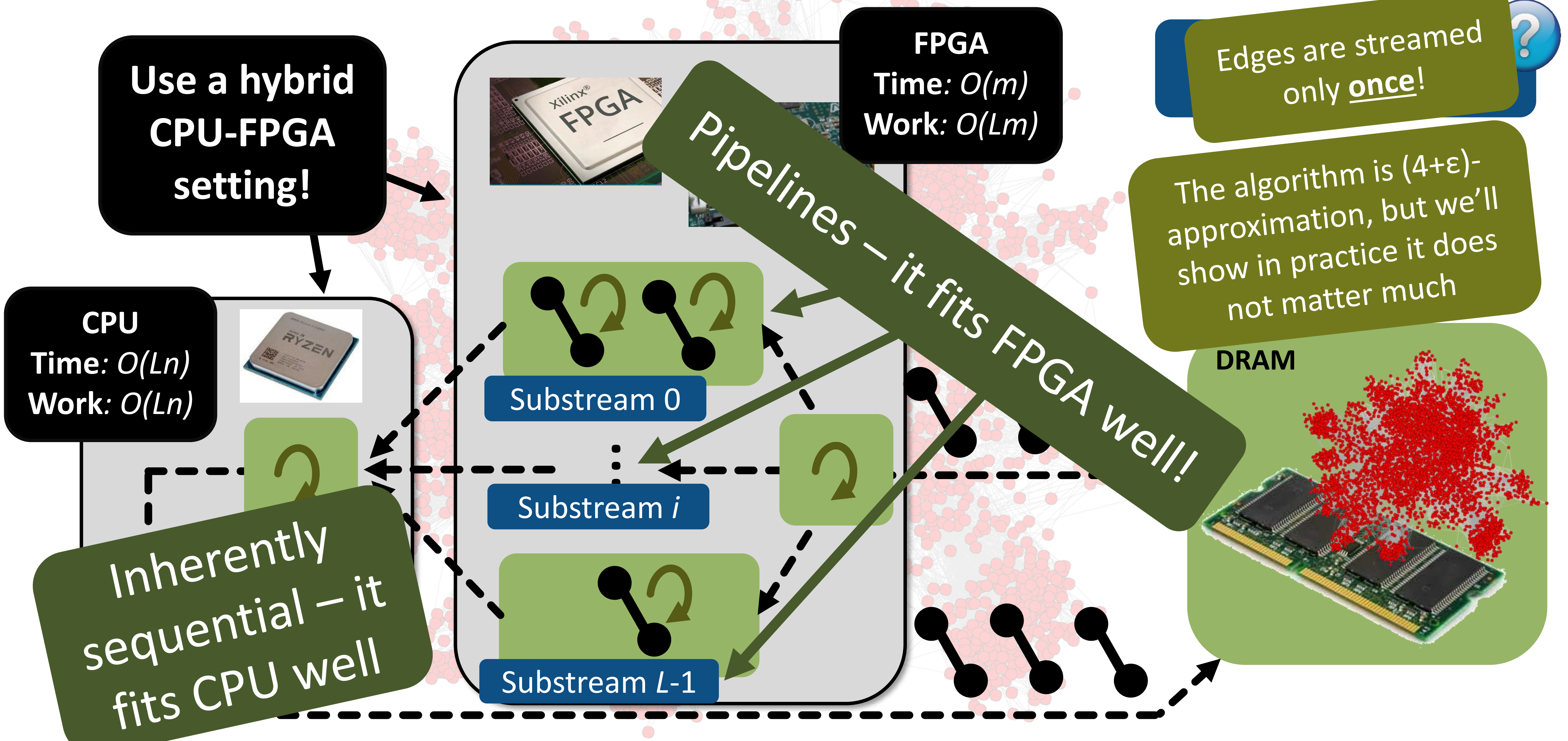
[1] M. Crouch and D. M. Stubbs. Improved streaming Algorithms for weighted Matching, via unweighted Matching. LIPIcs-Leibniz Informatics. 2014.

Substream-Centric Graph Processing + Crouch and Stubbs Matchings [1]



[1] M. Crouch and D. M. Stubbs. Improved streaming Algorithms for weighted Matching, via unweighted Matching. LIPIcs-Leibniz Informatics. 2014.

Substream-Centric Graph Processing + Crouch and Stubbs Matchings [1]



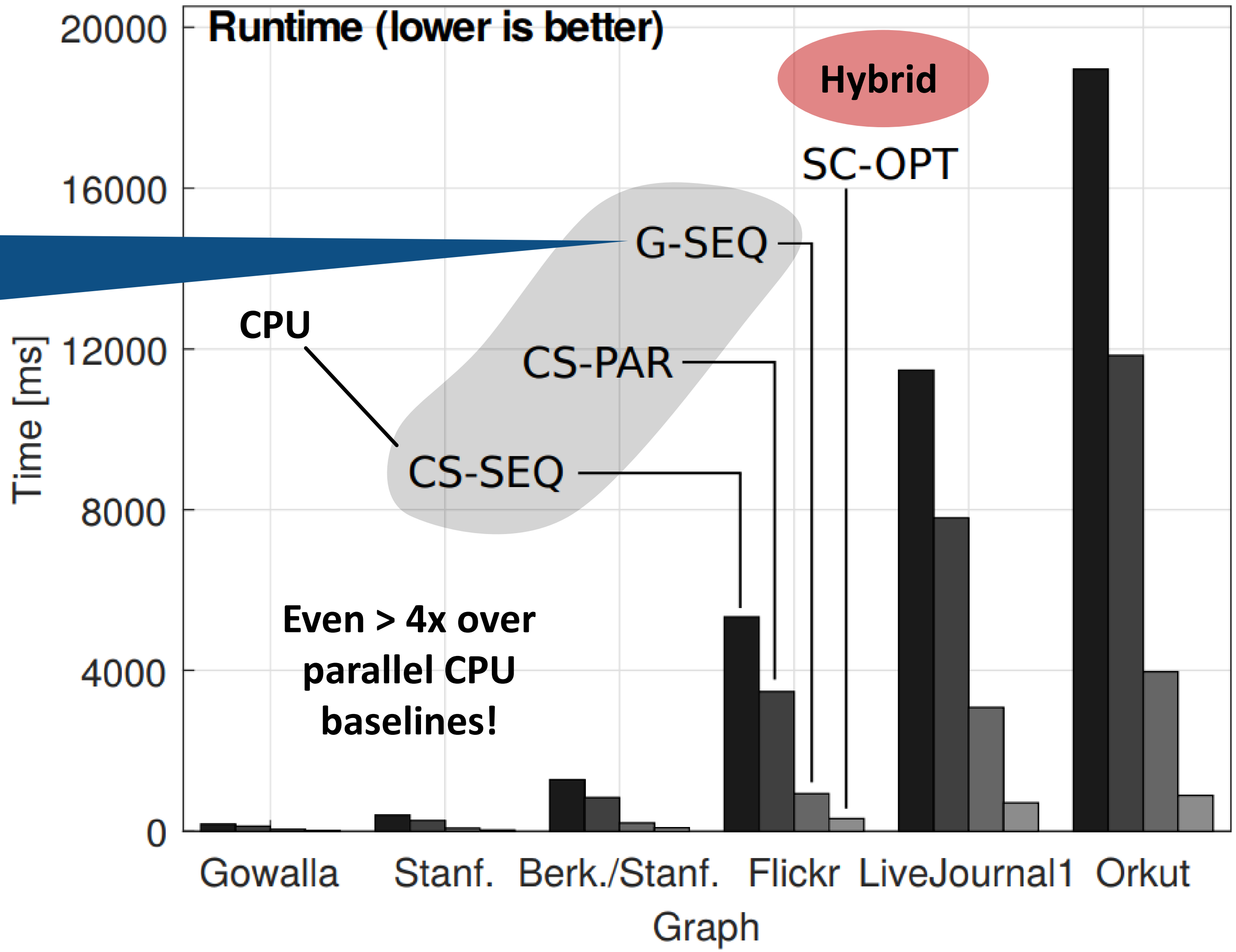
[1] M. Crouch and D. M. Stubbs. Improved streaming Algorithms for weighted Matching, via unweighted Matching. LIPIcs-Leibniz Informatics. 2014.

Performance Analysis
Various Graphs

Parameters:
#Substreams = 64, #Threads = 4

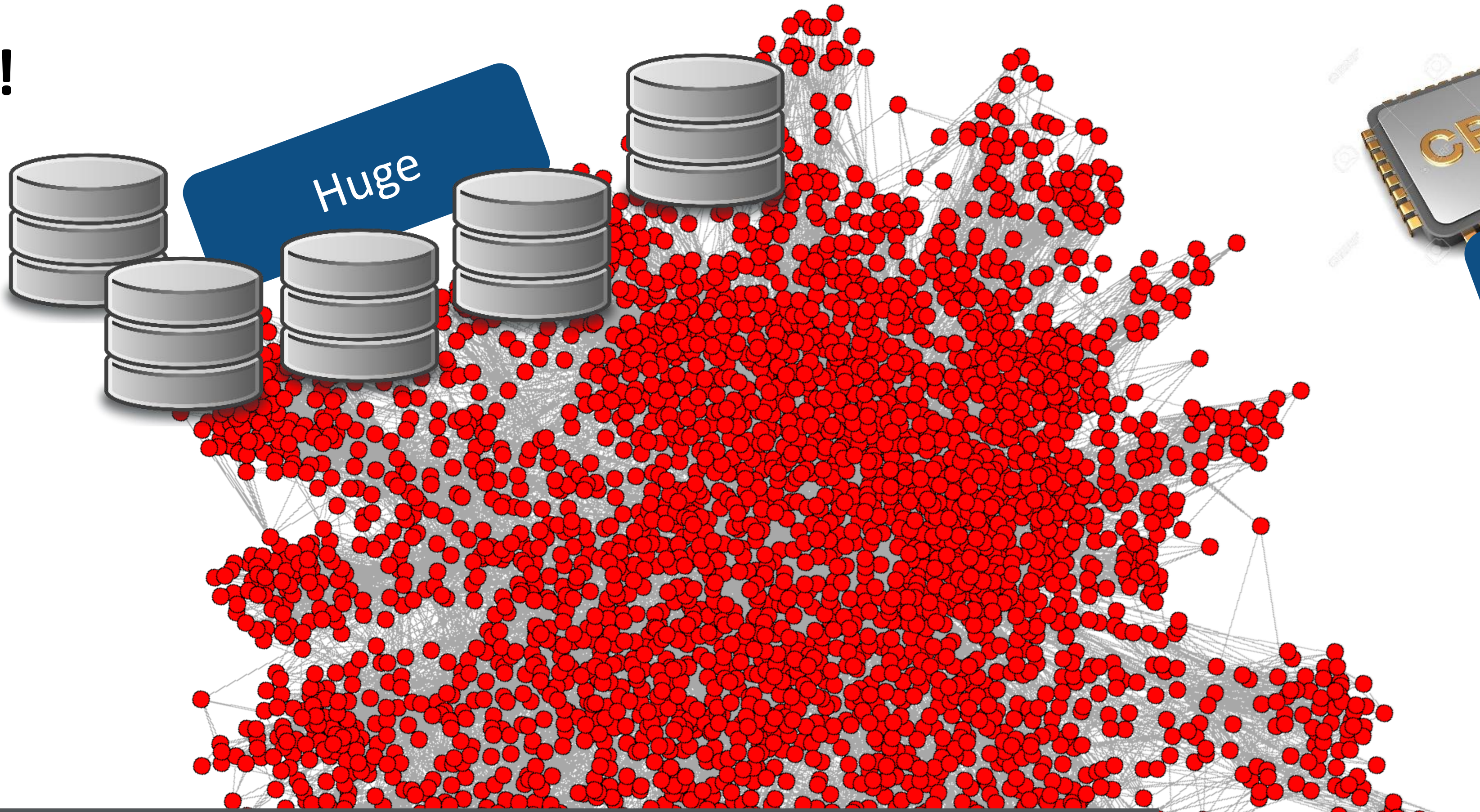
SC-OPT secures highest performance

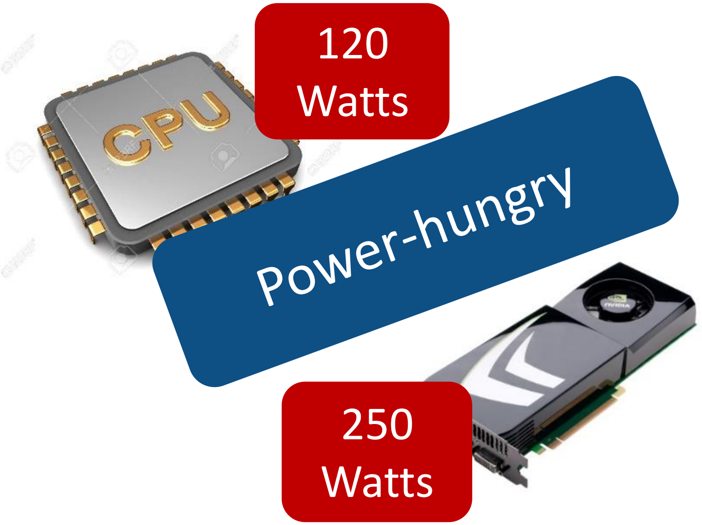
State-of-the-art MWM algorithm, space-optimal, **time-optimal (O(m))**, (2+ε)-approximation



Graph	Type	m	n
Kronecker	Synthetic power-law	$\approx 48n$	$2^k; k = 16, \ldots, 21$
Gowalla	Social network	950,327	196,591
Flickr	Social network	33,140,017	2,302,925
LiveJournal1	Social network	68,993,773	4,847,571
Orkut	Social network	117,184,899	3,072,441
Stanford	Hyperlink graph	2,312,497	281,903
Berkeley	Hyperlink graph	7,600,595	685,230
arXiv hep-th	Citation graph	352,807	27,770

Problems!





Substream-Centric Maximum Matchings on FPGA

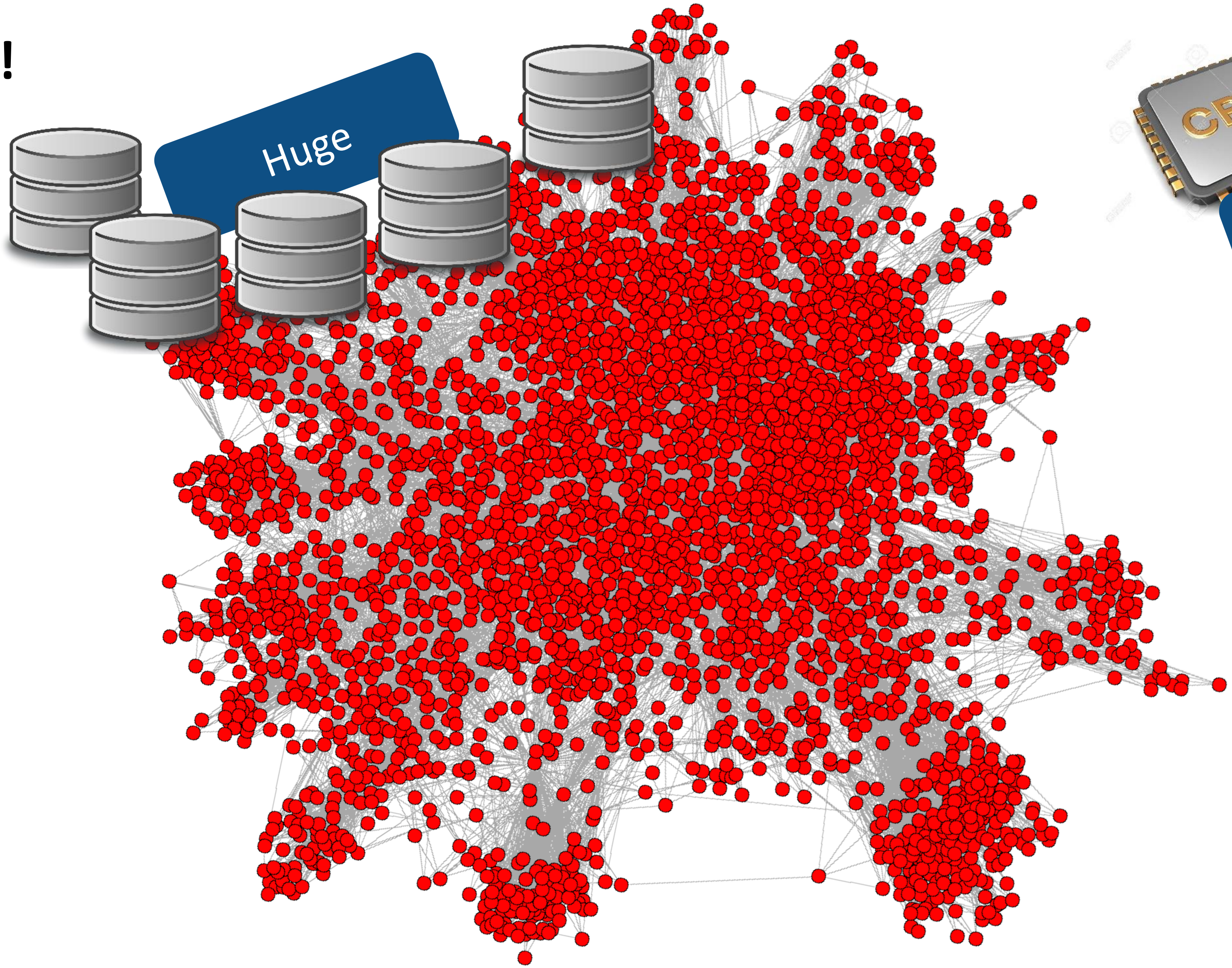
Maciej Besta, Marc Fischer, Tal Ben-Nun, Johannes De Fine Licht, Torsten Hoefler
Department of Computer Science, ETH Zurich

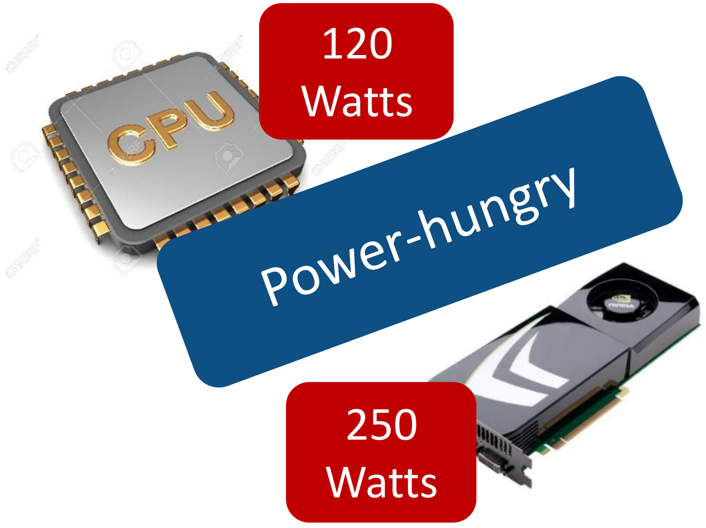
ABSTRACT

Developing high-performance and energy-efficient algorithms for maximum matchings is becoming increasingly important in social network analysis, computational sciences, schedul-

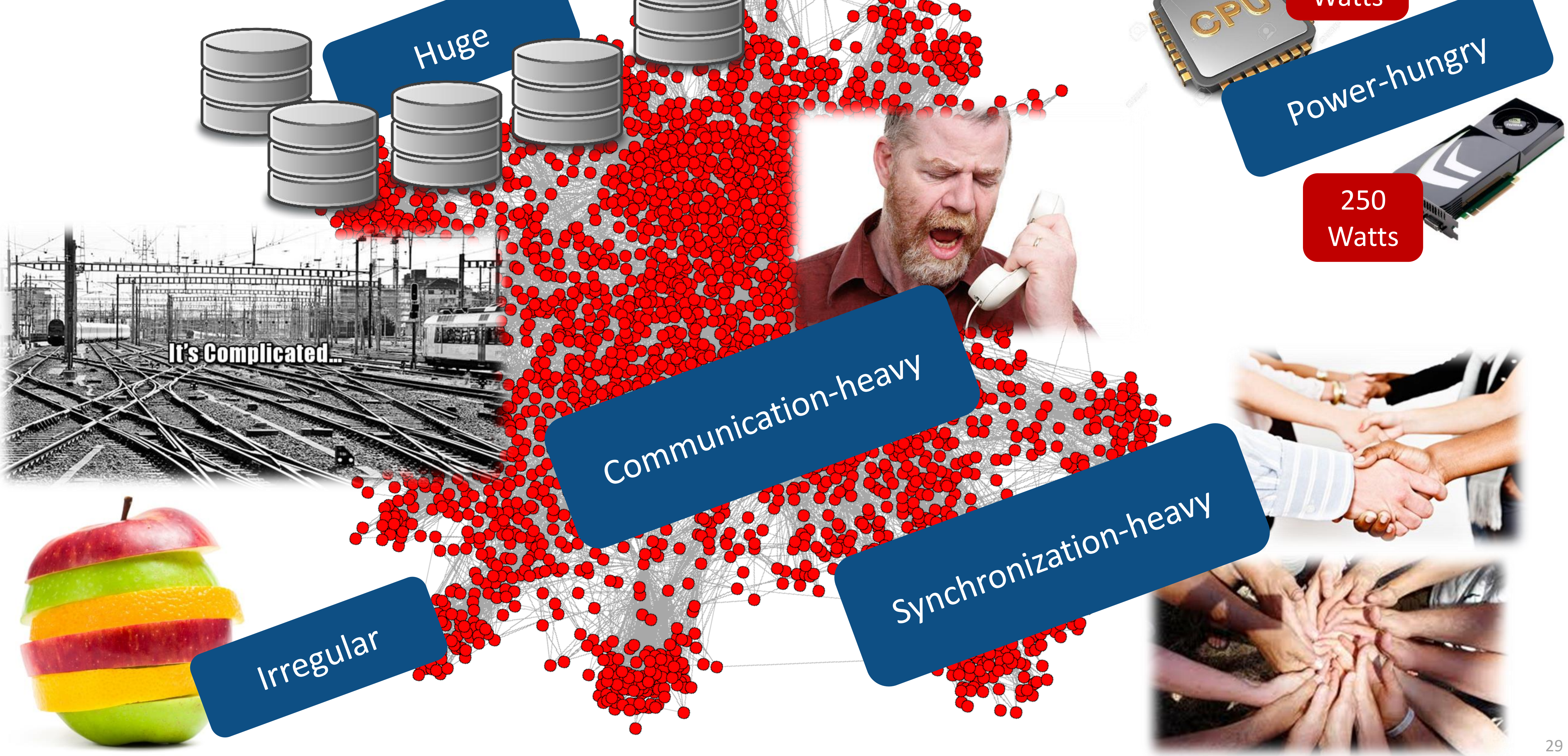
compound can be used to show the locations of double bonds in the chemical structure [59]. As deriving the exact MM is usually computationally expensive, significant focus has been placed on developing fast approximate solutions [17].

Problems!

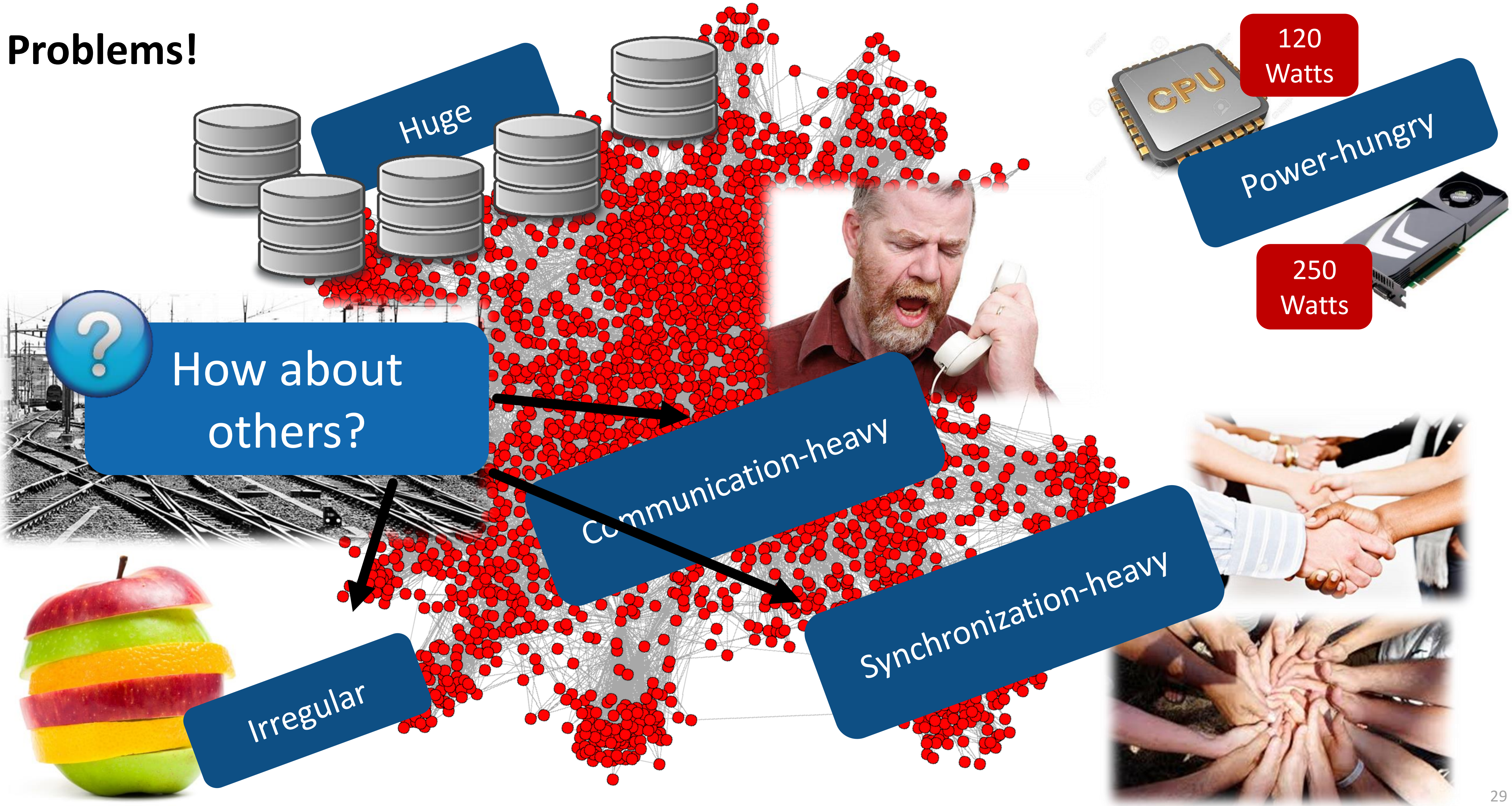




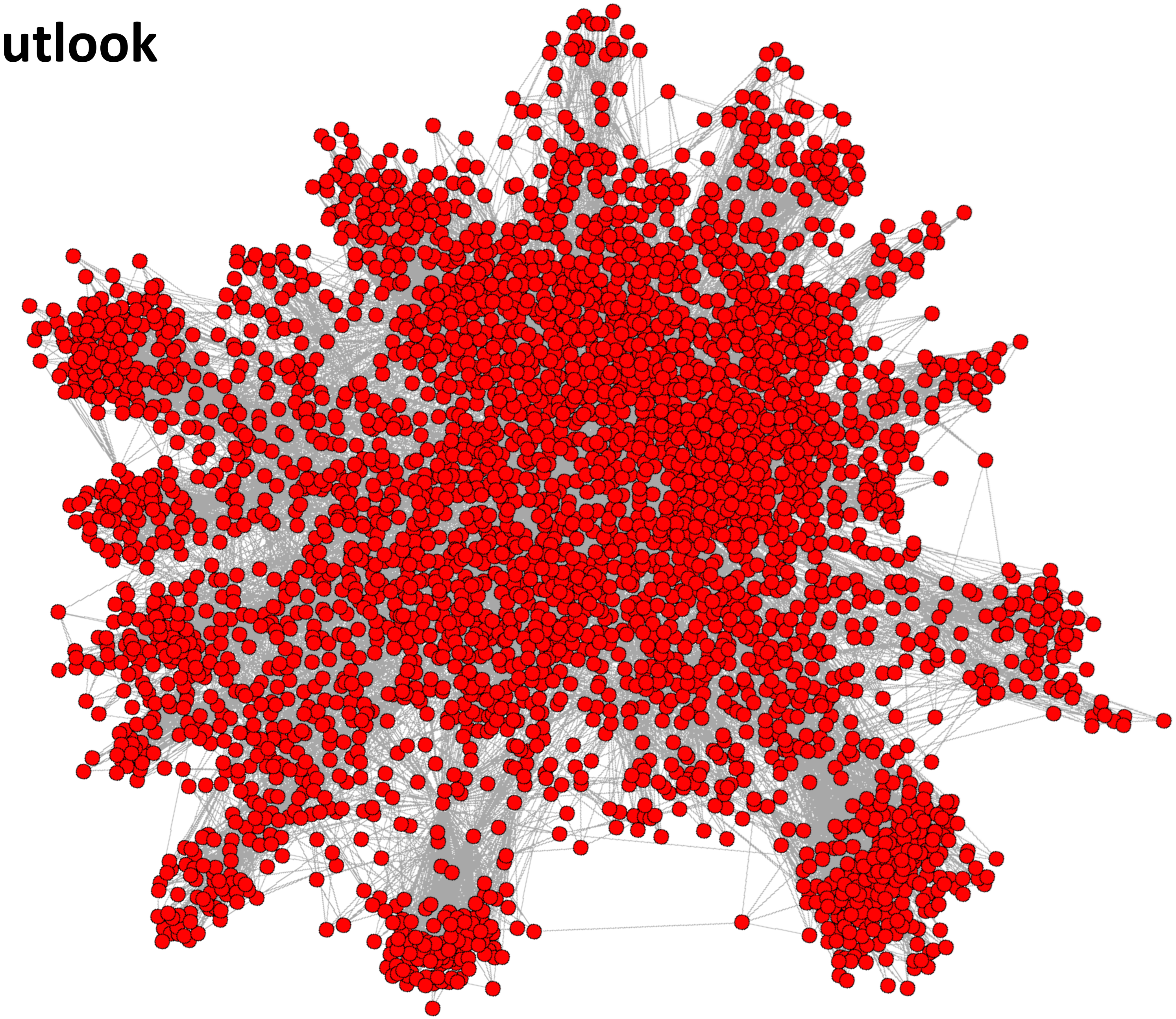
Problems!


Huge
120
Watts
CPU
Power-hungry
250
Watts
It's Complicated...
Communication-heavy
Synchronization-heavy
Irregular

Problems!


Huge
120
Watts
Power-hungry
250
Watts
How about
others?
Communication-heavy
Synchronization-heavy
Irregular

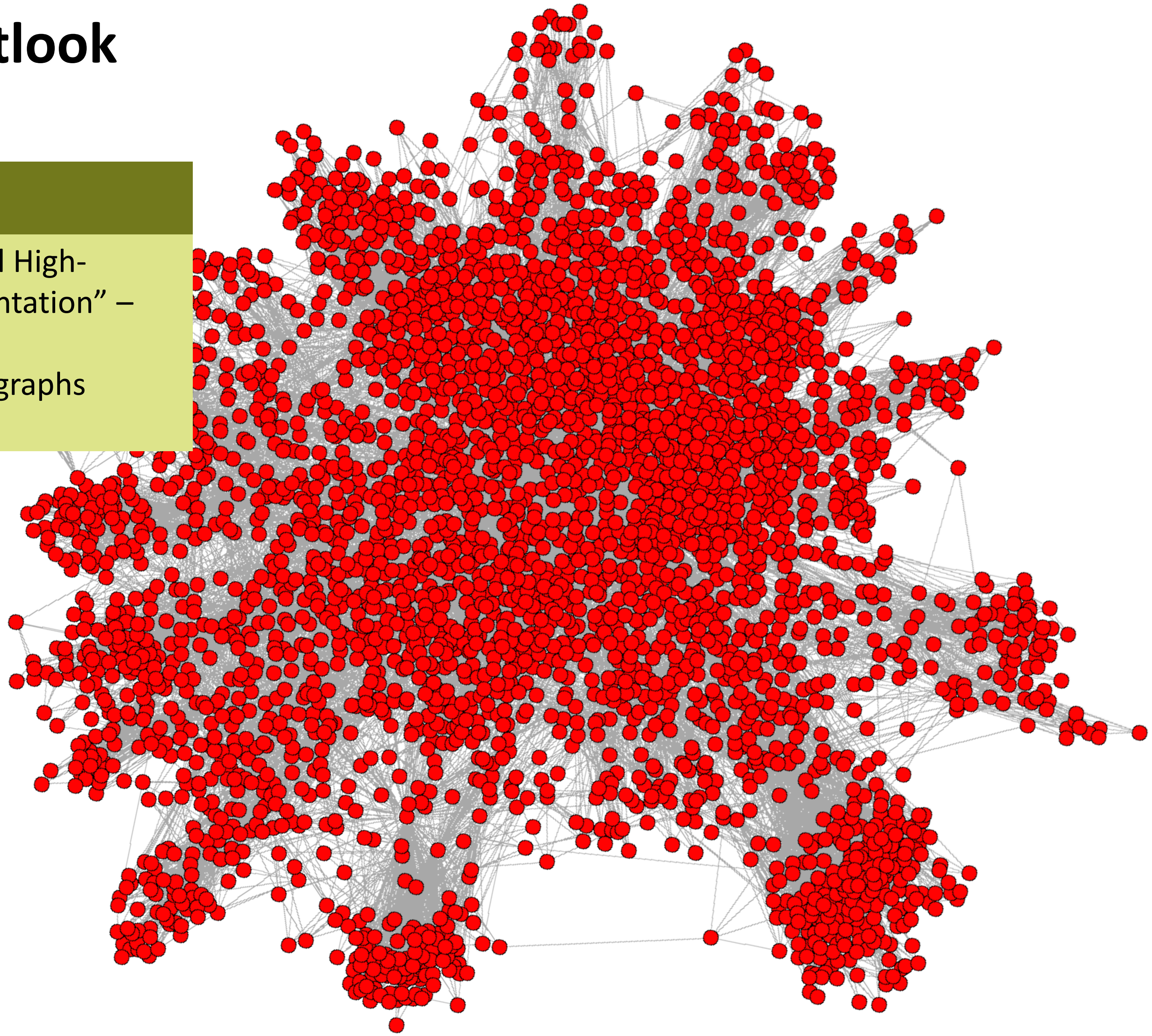
Summary and outlook



Summary and outlook

Log(Graph)

- "Log(Graph): A Near-Optimal High-Performance Graph Representation" – ACM PACT'18
- Minimal storage bounds for graphs during processing



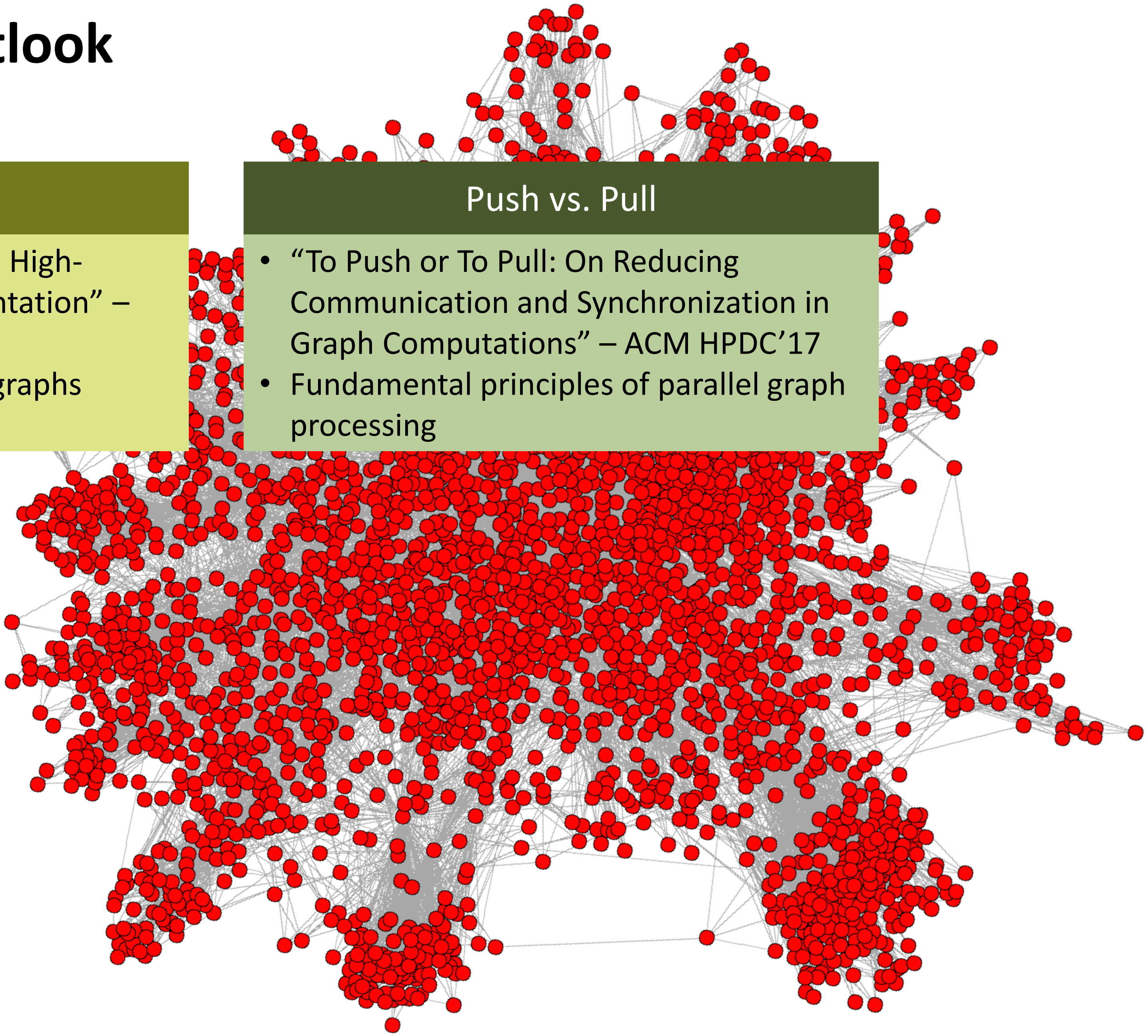
Summary and outlook

Log(Graph)

- “Log(Graph): A Near-Optimal High-Performance Graph Representation” – ACM PACT’18
- Minimal storage bounds for graphs during processing

Push vs. Pull

- “To Push or To Pull: On Reducing Communication and Synchronization in Graph Computations” – ACM HPDC’17
- Fundamental principles of parallel graph processing



Summary and outlook

Log(Graph)

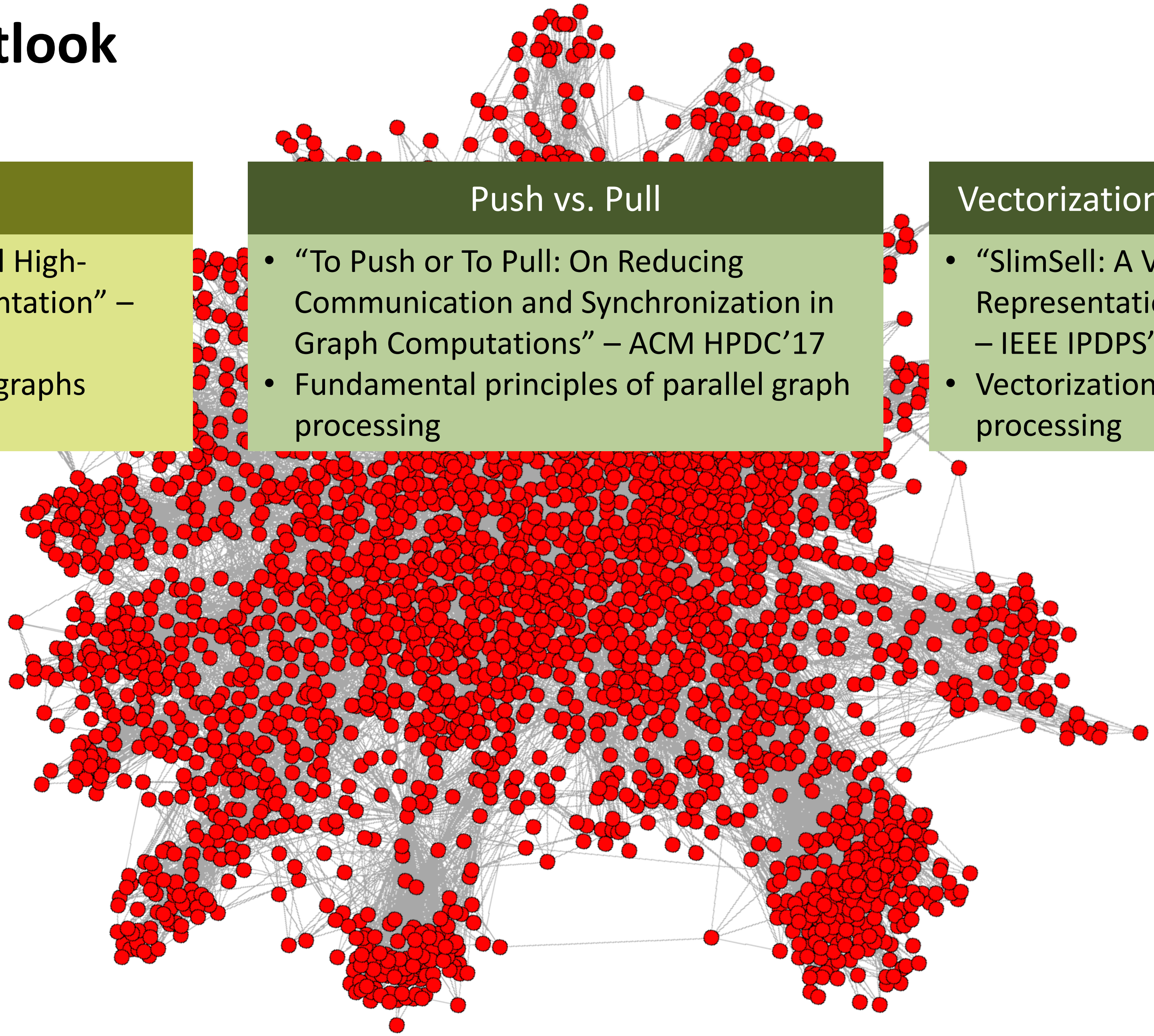
- “Log(Graph): A Near-Optimal High-Performance Graph Representation” – ACM PACT’18
- Minimal storage bounds for graphs during processing

Push vs. Pull

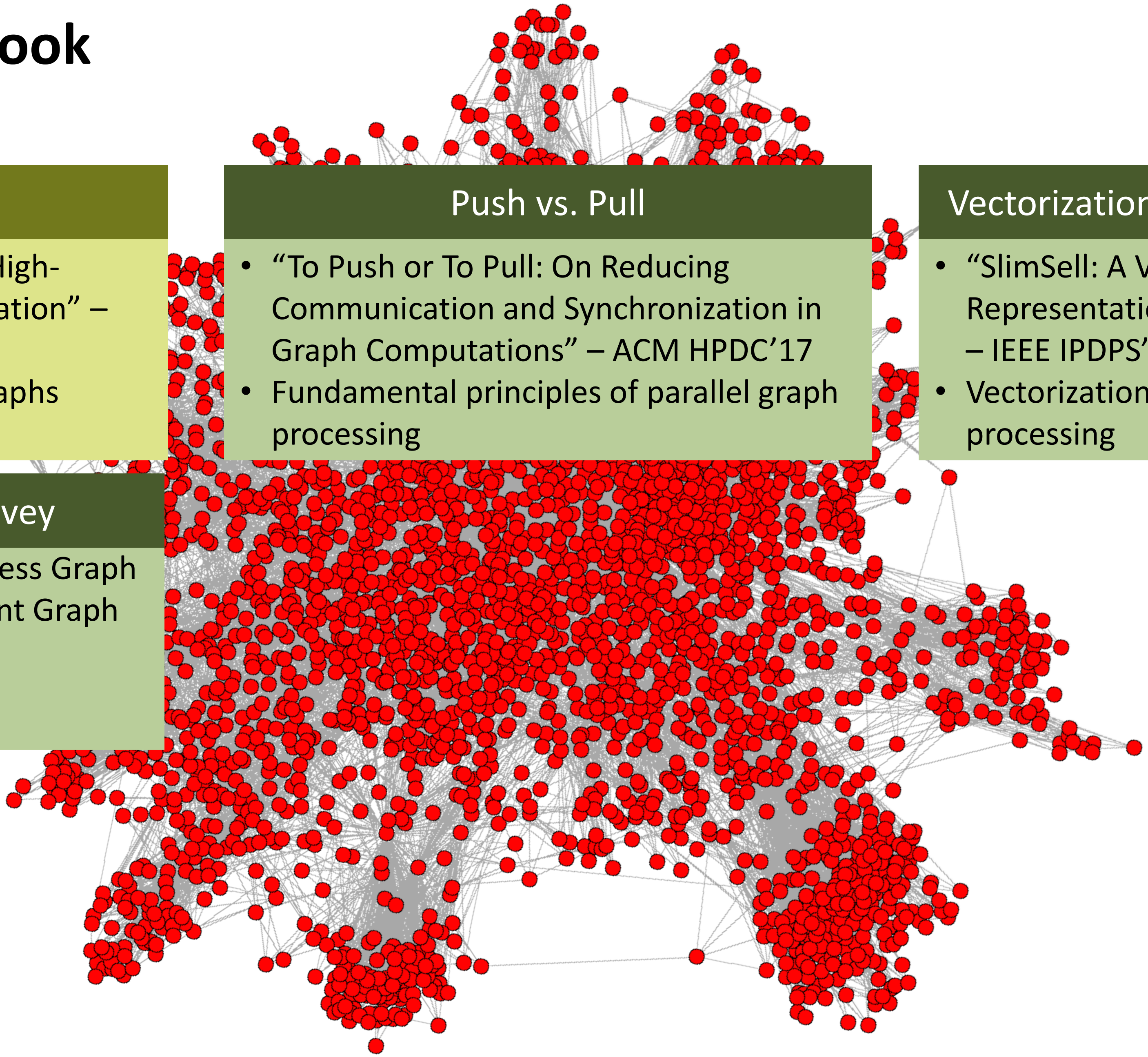
- “To Push or To Pull: On Reducing Communication and Synchronization in Graph Computations” – ACM HPDC’17
- Fundamental principles of parallel graph processing

Vectorization of Graph Computations

- “SlimSell: A Vectorized Graph Representation for Breadth-First Search” – IEEE IPDPS’17
- Vectorization schemes for parallel graph processing



Summary and outlook



Log(Graph)

- “Log(Graph): A Near-Optimal High-Performance Graph Representation” – ACM PACT’18
- Minimal storage bounds for graphs during processing

Push vs. Pull

- “To Push or To Pull: On Reducing Communication and Synchronization in Graph Computations” – ACM HPDC’17
- Fundamental principles of parallel graph processing

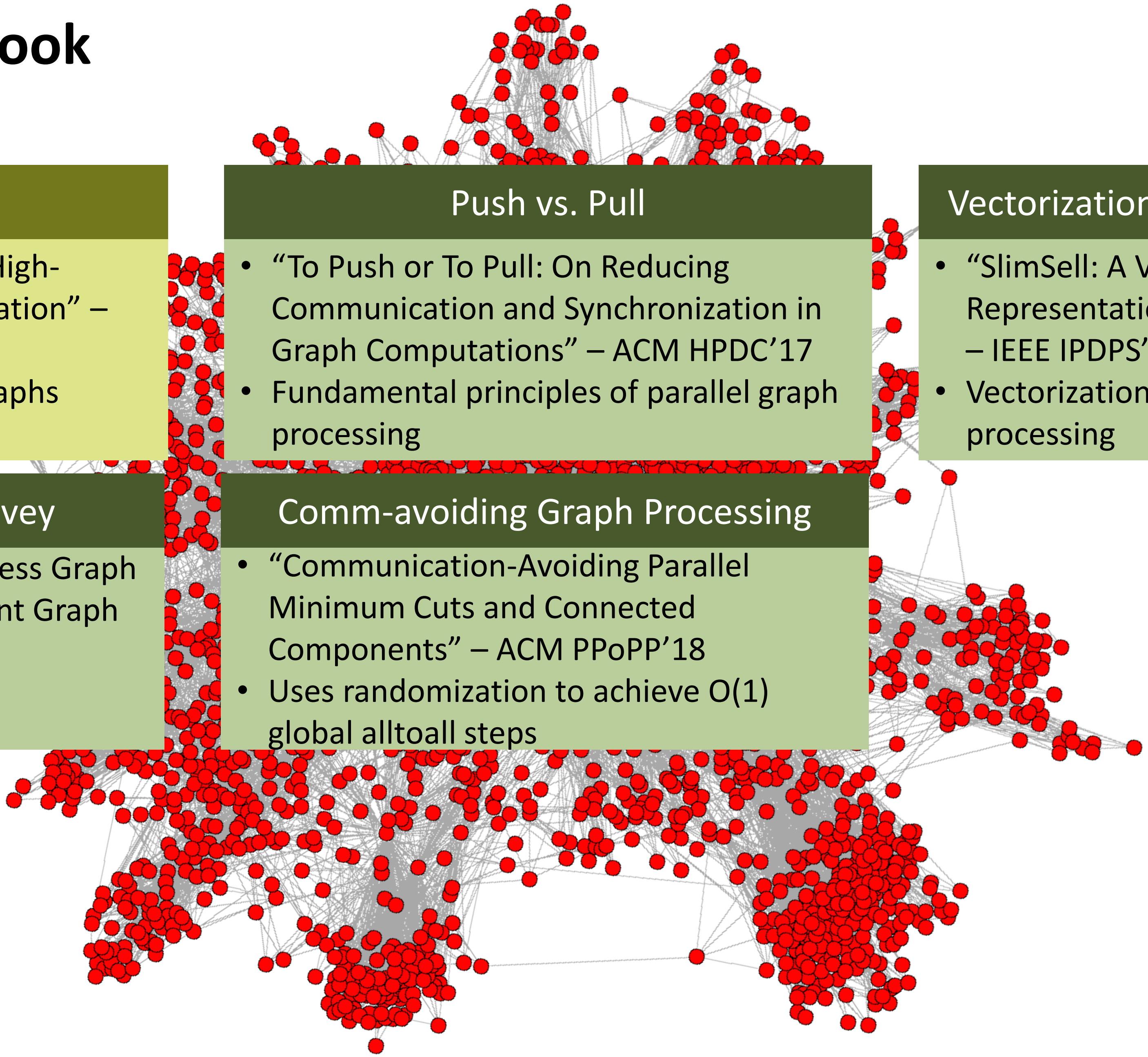
Vectorization of Graph Computations

- “SlimSell: A Vectorized Graph Representation for Breadth-First Search” – IEEE IPDPS’17
- Vectorization schemes for parallel graph processing

Graph Compression Survey

- “Survey and Taxonomy of Lossless Graph Compression and Space-Efficient Graph Representations” – arXiv
- Comprehensive overview
  - 54 pages, 465 references

Summary and outlook



Log(Graph)

- "Log(Graph): A Near-Optimal High-Performance Graph Representation" – ACM PACT'18
- Minimal storage bounds for graphs during processing

Push vs. Pull

- "To Push or To Pull: On Reducing Communication and Synchronization in Graph Computations" – ACM HPDC'17
- Fundamental principles of parallel graph processing

Vectorization of Graph Computations

- "SlimSell: A Vectorized Graph Representation for Breadth-First Search" – IEEE IPDPS'17
- Vectorization schemes for parallel graph processing

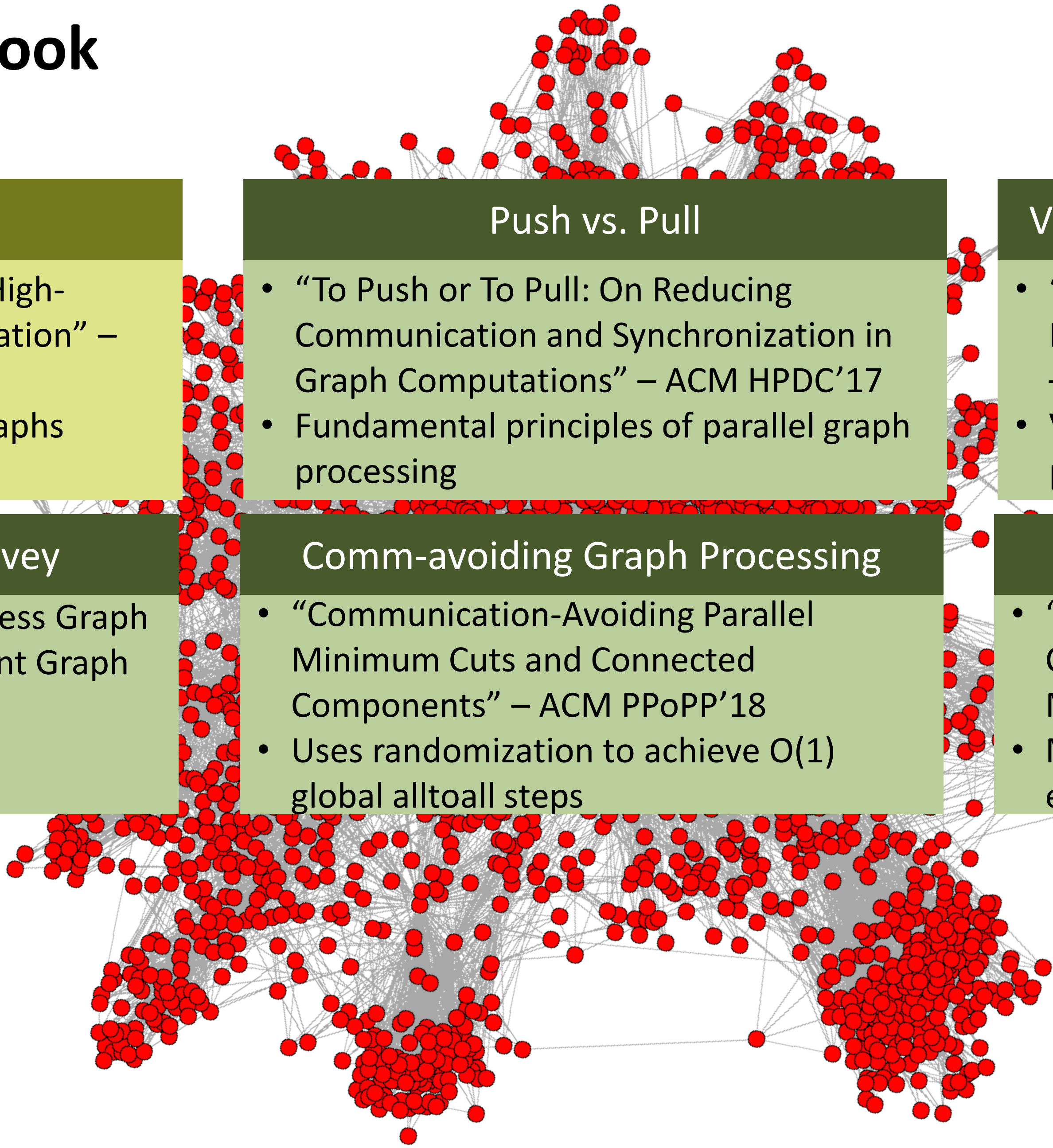
Graph Compression Survey

- "Survey and Taxonomy of Lossless Graph Compression and Space-Efficient Graph Representations" – arXiv
- Comprehensive overview
  - 54 pages, 465 references

Comm-avoiding Graph Processing

- "Communication-Avoiding Parallel Minimum Cuts and Connected Components" – ACM PPoPP'18
- Uses randomization to achieve O(1) global alltoall steps

Summary and outlook



Log(Graph)

- “Log(Graph): A Near-Optimal High-Performance Graph Representation” – ACM PACT’18
- Minimal storage bounds for graphs during processing

Push vs. Pull

- “To Push or To Pull: On Reducing Communication and Synchronization in Graph Computations” – ACM HPDC’17
- Fundamental principles of parallel graph processing

Vectorization of Graph Computations

- “SlimSell: A Vectorized Graph Representation for Breadth-First Search” – IEEE IPDPS’17
- Vectorization schemes for parallel graph processing

Graph Compression Survey

- “Survey and Taxonomy of Lossless Graph Compression and Space-Efficient Graph Representations” – arXiv
- Comprehensive overview
  - 54 pages, 465 references

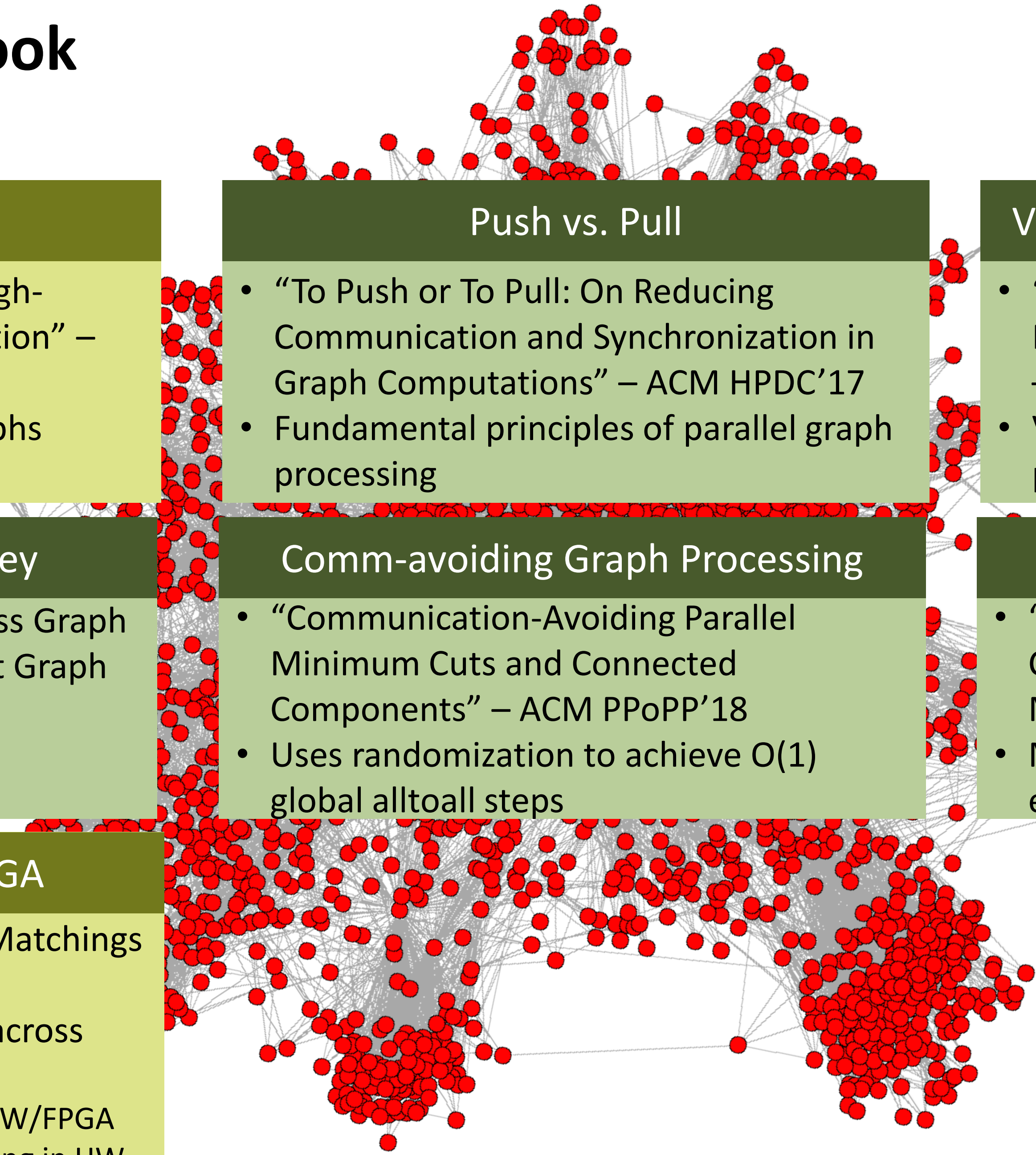
Comm-avoiding Graph Processing

- “Communication-Avoiding Parallel Minimum Cuts and Connected Components” – ACM PPoPP’18
- Uses randomization to achieve O(1) global alltoall steps

Algebraic Betweenness Centrality

- “Scaling Betweenness Centrality using Communication-Efficient Sparse Matrix Multiplication” – ACM SC’17
- More on the algebraic view – complex example, large-scale sparse matrices

Summary and outlook



Log(Graph)
- "Log(Graph): A Near-Optimal High-Performance Graph Representation" – ACM PACT'18
- Minimal storage bounds for graphs during processing

Push vs. Pull
- "To Push or To Pull: On Reducing Communication and Synchronization in Graph Computations" – ACM HPDC'17
- Fundamental principles of parallel graph processing

Vectorization of Graph Computations
- "SlimSell: A Vectorized Graph Representation for Breadth-First Search" – IEEE IPDPS'17
- Vectorization schemes for parallel graph processing

Graph Compression Survey
- "Survey and Taxonomy of Lossless Graph Compression and Space-Efficient Graph Representations" – arXiv
- Comprehensive overview
  - 54 pages, 465 references

Comm-avoiding Graph Processing
- "Communication-Avoiding Parallel Minimum Cuts and Connected Components" – ACM PPoPP'18
- Uses randomization to achieve O(1) global alltoall steps

Algebraic Betweenness Centrality
- "Scaling Betweenness Centrality using Communication-Efficient Sparse Matrix Multiplication" – ACM SC'17
- More on the algebraic view – complex example, large-scale sparse matrices

Streaming Graphs on FPGA
- "Substream-Centric Maximum Matchings on FPGA" – FPGA'19
- New paradigm for parallelizing across substreams
  - Integrates with pipelining in HW/FPGA
  - Blueprint for efficient processing in HW

Summary and outlook

Log(Graph)

- "Log(Graph): A Near-Optimal High-Performance Graph Representation" – ACM PACT'18
- Minimal storage bounds for graphs during processing

Push vs. Pull

- "To Push or To Pull: On Reducing Communication and Synchronization in Graph Computations" – ACM HPDC'17
- Fundamental principles of parallel graph processing

Vectorization of Graph Computations

- "SlimSell: A Vectorized Graph Representation for Breadth-First Search" – IEEE IPDPS'17
- Vectorization schemes for parallel graph processing

Graph Compression Survey

- "Survey and Taxonomy of Lossless Graph Compression and Space-Efficient Graph Representations" – arXiv
- Comprehensive overview
  - 54 pages, 465 references

Comm-avoiding Graph Processing

- "Communication-Avoiding Parallel Minimum Cuts and Connected Components" – ACM PPoPP'18
- Uses randomization to achieve O(1) global alltoall steps

Algebraic Betweenness Centrality

- "Scaling Betweenness Centrality using Communication-Efficient Sparse Matrix Multiplication" – ACM SC'17
- More on the algebraic view – complex example, large-scale sparse matrices

Streaming Graphs on FPGA

- "Substream-Centric Maximum Matchings on FPGA" – FPGA'19
- New paradigm for parallelizing across substreams
  - Integrates with pipelining in HW/FPGA
  - Blueprint for efficient processing in HW

Streaming Graphs Survey

- "Theory and Practice of Streaming Graph Processing" – soon on arXiv
- Overview of streaming algorithms, approximations, research gaps
  - Way forward for FPGA?

Summary and outlook

Log(Graph)
- "Log(Graph): A Near-Optimal High-Performance Graph Representation" – ACM PACT'18
- Minimal storage bounds for graphs during processing

Push vs. Pull
- "To Push or To Pull: On Reducing Communication and Synchronization in Graph Computations" – ACM HPDC'17
- Fundamental principles of parallel graph processing

Vectorization of Graph Computations
- "SlimSell: A Vectorized Graph Representation for Breadth-First Search" – IEEE IPDPS'17
- Vectorization schemes for parallel graph processing

Graph Compression Survey
- "Survey and Taxonomy of Lossless Graph Compression and Space-Efficient Graph Representations" – arXiv
- Comprehensive overview
  - 54 pages, 465 references

Comm-avoiding Graph Processing
- "Communication-Avoiding Parallel Minimum Cuts and Connected Components" – ACM PPoPP'18
- Uses randomization to achieve O(1) global alltoall steps

Algebraic Betweenness Centrality
- "Scaling Betweenness Centrality using Communication-Efficient Sparse Matrix Multiplication" – ACM SC'17
- More on the algebraic view – complex example, large-scale sparse matrices

Streaming Graphs on FPGA
- "Substream-Centric Maximum Matchings on FPGA" – FPGA'19
- New paradigm for parallelizing across substreams
  - Integrates with pipelining in HW/FPGA
  - Blueprint for efficient processing in HW

Streaming Graphs Survey
- "Theory and Practice of Streaming Graph Processing" – soon on arXiv
- Overview of streaming algorithms, approximations, research gaps
  - Way forward for FPGA?

Graphs on FPGA Survey
- "Graph Processing on FPGAs: Taxonomy, Survey, Challenges" – arXiv
- Relatively young field of graph processing on FPGAs / in hardware
  - Identify research opportunities

Summary and outlook

Hardware Transactions for Graphs

- "Accelerating Irregular Computations with Hardware Transactional Memory and Active Messages" – ACM HPDC'15

Log(Graph)

- "Log(Graph): A Near-Optimal High-Performance Graph Representation" – ACM PACT'18
- Minimal storage bounds for graphs during processing

Push vs. Pull

- "To Push or To Pull: On Reducing Communication and Synchronization in Graph Computations" – ACM HPDC'17
- Fundamental principles of parallel graph processing

Vectorization of Graph Computations

- "SlimSell: A Vectorized Graph Representation for Breadth-First Search" – IEEE IPDPS'17
- Vectorization schemes for parallel graph processing

Graph Compression Survey

- "Survey and Taxonomy of Lossless Graph Compression and Space-Efficient Graph Representations" – arXiv
- Comprehensive overview
  - 54 pages, 465 references

Comm-avoiding Graph Processing

- "Communication-Avoiding Parallel Minimum Cuts and Connected Components" – ACM PPoPP'18
- Uses randomization to achieve O(1) global alltoall steps

Algebraic Betweenness Centrality

- "Scaling Betweenness Centrality using Communication-Efficient Sparse Matrix Multiplication" – ACM SC'17
- More on the algebraic view – complex example, large-scale sparse matrices

Streaming Graphs on FPGA

- "Substream-Centric Maximum Matchings on FPGA" – FPGA'19
- New paradigm for parallelizing across substreams
  - Integrates with pipelining in HW/FPGA
  - Blueprint for efficient processing in HW

Streaming Graphs Survey

- "Theory and Practice of Streaming Graph Processing" – soon on arXiv
- Overview of streaming algorithms, approximations, research gaps
  - Way forward for FPGA?

Graphs on FPGA Survey

- "Graph Processing on FPGAs: Taxonomy, Survey, Challenges" – arXiv
- Relatively young field of graph processing on FPGAs / in hardware
  - Identify research opportunities

Summary and outlook

Slim Graph
- "Slim Graph: Practical Lossy Graph Compression for Approximate Graph Processing, Storage, and Analytics" – ACM/IEEE SC'19

Hardware Transactions for Graphs
- "Accelerating Irregular Computations with Hardware Transactional Memory and Active Messages" – ACM HPDC'15

Log(Graph)
- "Log(Graph): A Near-Optimal High-Performance Graph Representation" – ACM PACT'18
- Minimal storage bounds for graphs during processing

Push vs. Pull
- "To Push or To Pull: On Reducing Communication and Synchronization in Graph Computations" – ACM HPDC'17
- Fundamental principles of parallel graph processing

Vectorization of Graph Computations
- "SlimSell: A Vectorized Graph Representation for Breadth-First Search" – IEEE IPDPS'17
- Vectorization schemes for parallel graph processing

Graph Compression Survey
- "Survey and Taxonomy of Lossless Graph Compression and Space-Efficient Graph Representations" – arXiv
- Comprehensive overview
  - 54 pages, 465 references

Comm-avoiding Graph Processing
- "Communication-Avoiding Parallel Minimum Cuts and Connected Components" – ACM PPoPP'18
- Uses randomization to achieve O(1) global alltoall steps

Algebraic Betweenness Centrality
- "Scaling Betweenness Centrality using Communication-Efficient Sparse Matrix Multiplication" – ACM SC'17
- More on the algebraic view – complex example, large-scale sparse matrices

Streaming Graphs on FPGA
- "Substream-Centric Maximum Matchings on FPGA" – FPGA'19
- New paradigm for parallelizing across substreams
  - Integrates with pipelining in HW/FPGA
  - Blueprint for efficient processing in HW

Streaming Graphs Survey
- "Theory and Practice of Streaming Graph Processing" – soon on arXiv
- Overview of streaming algorithms, approximations, research gaps
  - Way forward for FPGA?

Graphs on FPGA Survey
- "Graph Processing on FPGAs: Taxonomy, Survey, Challenges" – arXiv
- Relatively young field of graph processing on FPGAs / in hardware
  - Identify research opportunities

Summary and outlook

Slim Graph
- "Slim Graph: Practical Lossy Graph Compression for Approximate Graph Processing, Storage, and Analytics" – ACM/IEEE SC'19

Hardware Transactions for Graphs
- "Accelerati… with Hardw… and Active…

…and others ☺

Log(Graph)
- "Log(Graph): A Near-Optimal High-Performance Graph Representation" – ACM PACT'18
- Minimal storage bounds for graphs during processing

Push vs. Pull
- "To Push or To Pull: On Reducing Communication and Synchronization in Graph Computations" – ACM HPDC'17
- Fundamental principles of parallel graph processing

Vectorization of Graph Computations
- "SlimSell: A Vectorized Graph Representation for Breadth-First Search" – IEEE IPDPS'17
- Vectorization schemes for parallel graph processing

Graph Compression Survey
- "Survey and Taxonomy of Lossless Graph Compression and Space-Efficient Graph Representations" – arXiv
- Comprehensive overview
  - 54 pages, 465 references

Comm-avoiding Graph Processing
- "Communication-Avoiding Parallel Minimum Cuts and Connected Components" – ACM PPoPP'18
- Uses randomization to achieve O(1) global alltoall steps

Algebraic Betweenness Centrality
- "Scaling Betweenness Centrality using Communication-Efficient Sparse Matrix Multiplication" – ACM SC'17
- More on the algebraic view – complex example, large-scale sparse matrices

Streaming Graphs on FPGA
- "Substream-Centric Maximum Matchings on FPGA" – FPGA'19
- New paradigm for parallelizing across substreams
  - Integrates with pipelining in HW/FPGA
  - Blueprint for efficient processing in HW

Streaming Graphs Survey
- "Theory and Practice of Streaming Graph Processing" – soon on arXiv
- Overview of streaming algorithms, approximations, research gaps
  - Way forward for FPGA?

Graphs on FPGA Survey
- "Graph Processing on FPGAs: Taxonomy, Survey, Challenges" – arXiv
- Relatively young field of graph processing on FPGAs / in hardware
  - Identify research opportunities

Summary and outlook

Slim Graph
- "Slim Graph: Practical Lossy Graph Compression for Approximate Graph Processing, Storage, and Analytics" – ACM/IEEE SC'19

Hardware Transactions for Graphs
- "Accelerati… with Hardw… and Active…

…and others ☺

Log(Graph)
- "Log(Graph): A Near-Optimal High-Performance Graph Representation" ACM PACT'18
- Minimal storage bounds for graphs during processing

Push vs. Pull
- "To Push or To Pull: On Reducing

Vectorization of Graph Computations
- "SlimSell: A Vectorized Graph …esentation for Breadth-First Search" … IPDPS'17
- …rization schemes for parallel graph …ssing

A big challenges ahead: develop a framework to integrate all techniques!

SPCL's approach: *stateful dataflow graphs*

We're always hiring excellent PhD students and postdocs at SPCL/ETH at spcl.inf.ethz.ch/Jobs

Graph Compression Survey
- "Survey and Taxonomy of Lossless … Compression and Space-Efficient G… Representations" – arXiv
- Comprehensive overview
  - 54 pages, 465 references

…braic Betweenness Centrality
- …g Betweenness Centrality using …unication-Efficient Sparse Matrix …lication" – ACM SC'17
- …on the algebraic view – complex …ple, large-scale sparse matrices

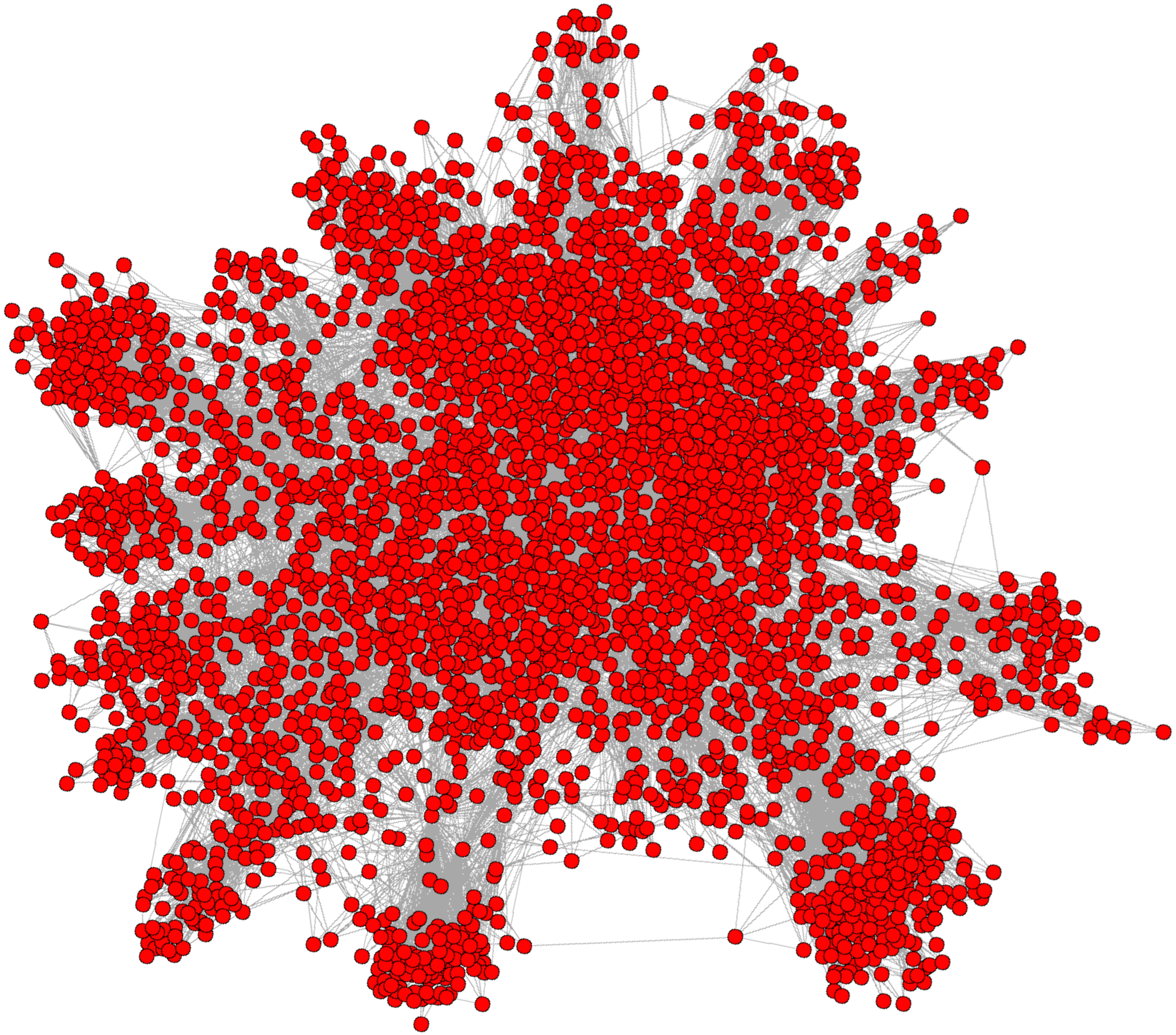
Streaming Graphs on FPGA
- "Substream-Centric Maximum Matchings on FPGA" – FPGA'19
- New paradigm for parallelizing across substreams
  - Integrates with pipelining in HW/FPGA
  - Blueprint for efficient processing in HW

Streaming Graphs Survey
- "Theory and Practice of Streaming Graph Processing" – soon on arXiv
- Overview of streaming algorithms, approximations, research gaps
  - Way forward for FPGA?

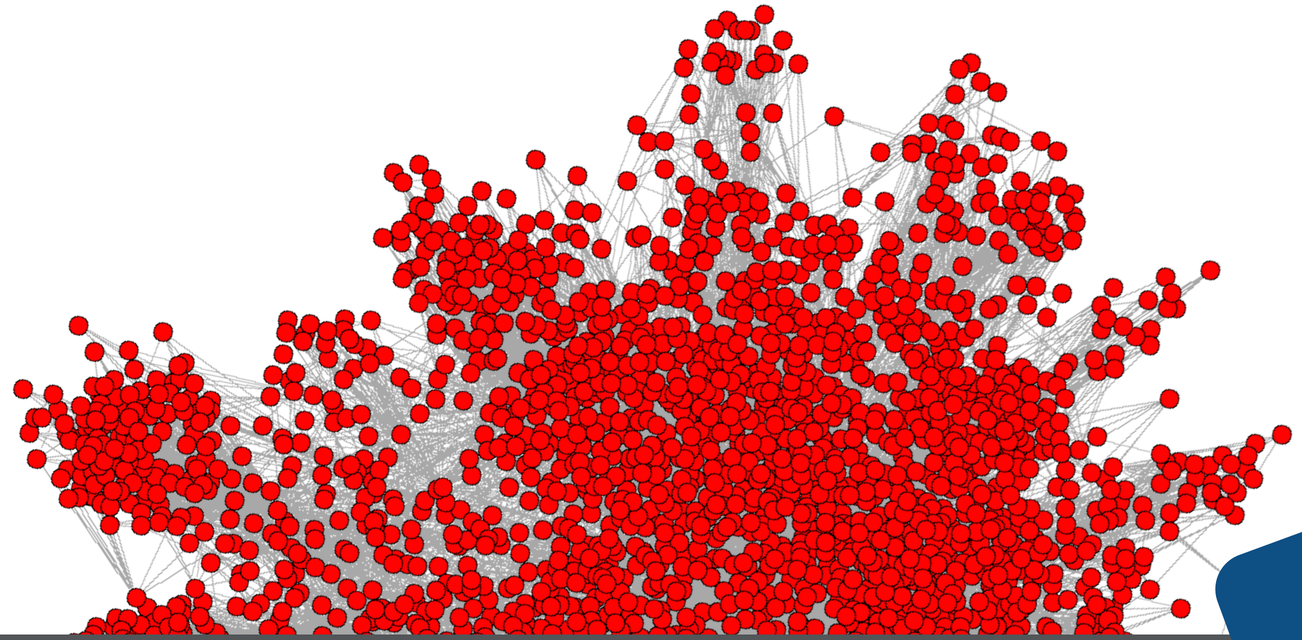
Graphs on FPGA Survey
- "Graph Processing on FPGAs: Taxonomy, Survey, Challenges" – arXiv
- Relatively young field of graph processing on FPGAs / in hardware
  - Identify research opportunities

Backup

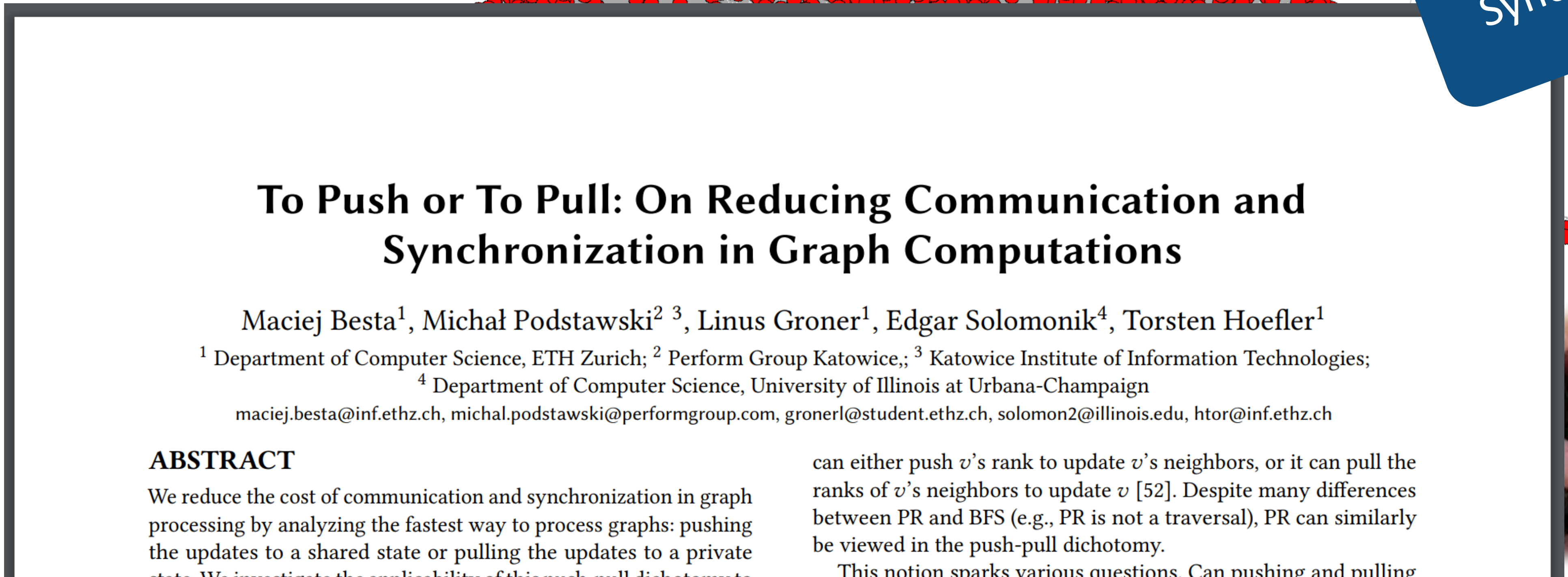
Problems!



Problems!



Synchronization-heavy



To Push or To Pull: On Reducing Communication and Synchronization in Graph Computations

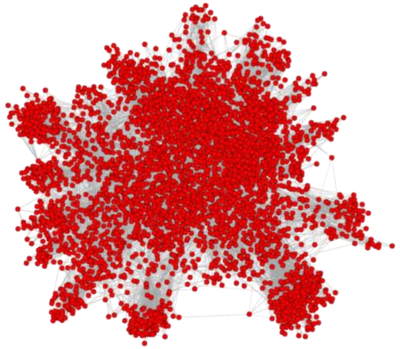
Maciej Besta[1], Michał Podstawski[2 3], Linus Groner[1], Edgar Solomonik[4], Torsten Hoefler[1]
[1] Department of Computer Science, ETH Zurich; [2] Perform Group Katowice,; [3] Katowice Institute of Information Technologies;
[4] Department of Computer Science, University of Illinois at Urbana-Champaign
maciej.besta@inf.ethz.ch, michal.podstawski@performgroup.com, gronerl@student.ethz.ch, solomon2@illinois.edu, htor@inf.ethz.ch

ABSTRACT
We reduce the cost of communication and synchronization in graph processing by analyzing the fastest way to process graphs: pushing the updates to a shared state or pulling the updates to a private

can either push v's rank to update v's neighbors, or it can pull the ranks of v's neighbors to update v [52]. Despite many differences between PR and BFS (e.g., PR is not a traversal), PR can similarly be viewed in the push-pull dichotomy.

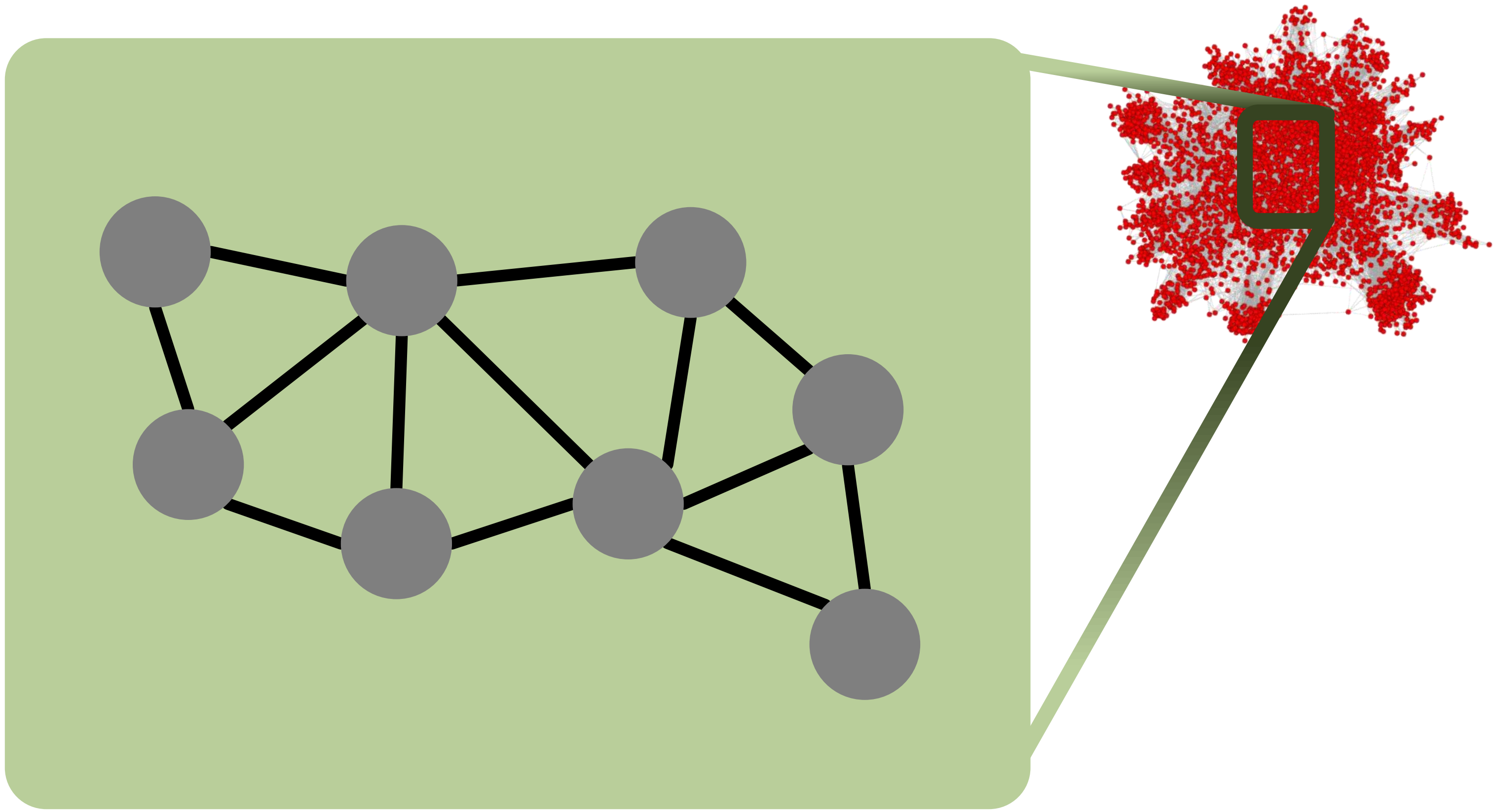
PAGERANK

PAGERANK



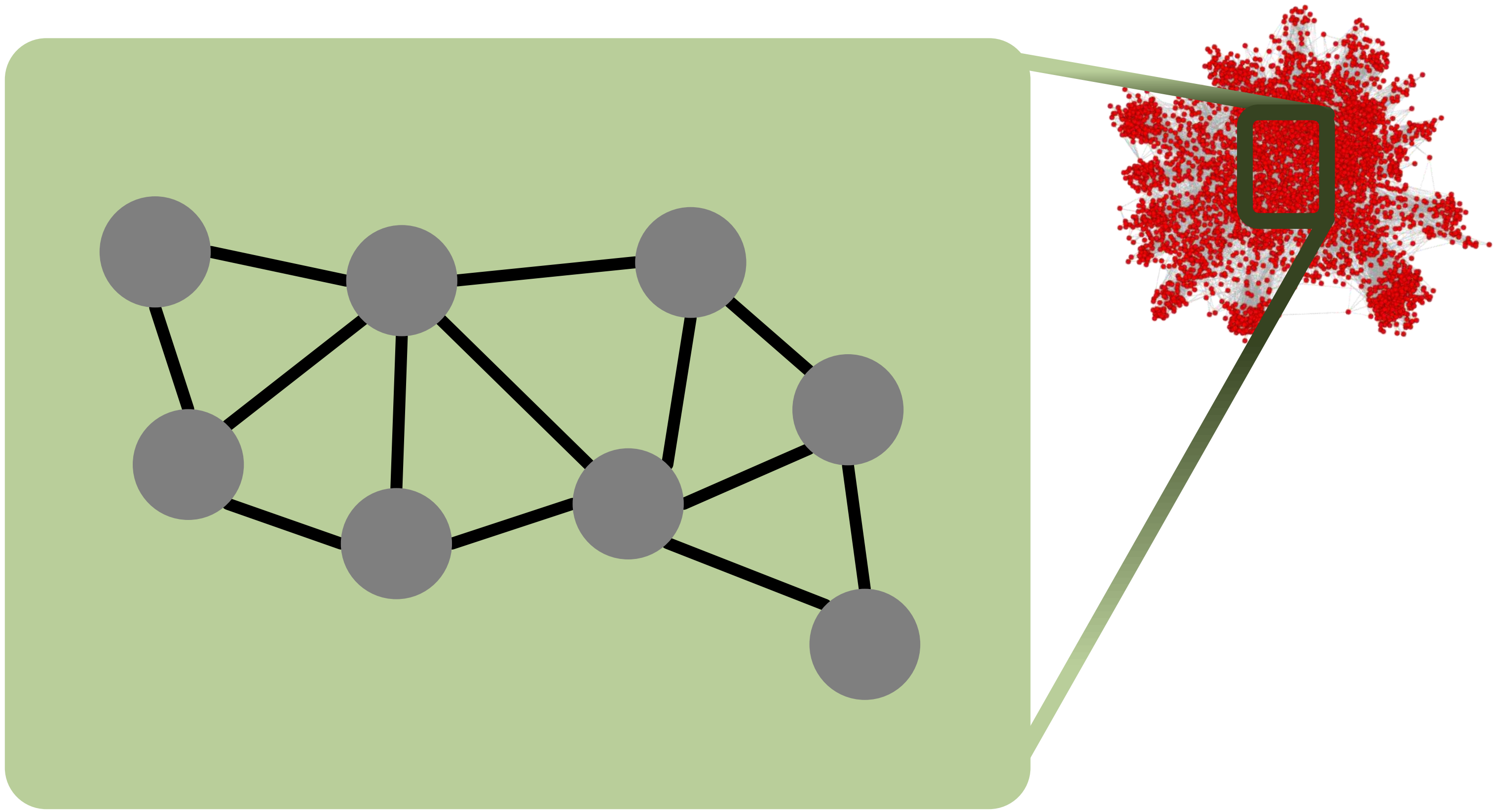
PAGERANK

PAGERANK



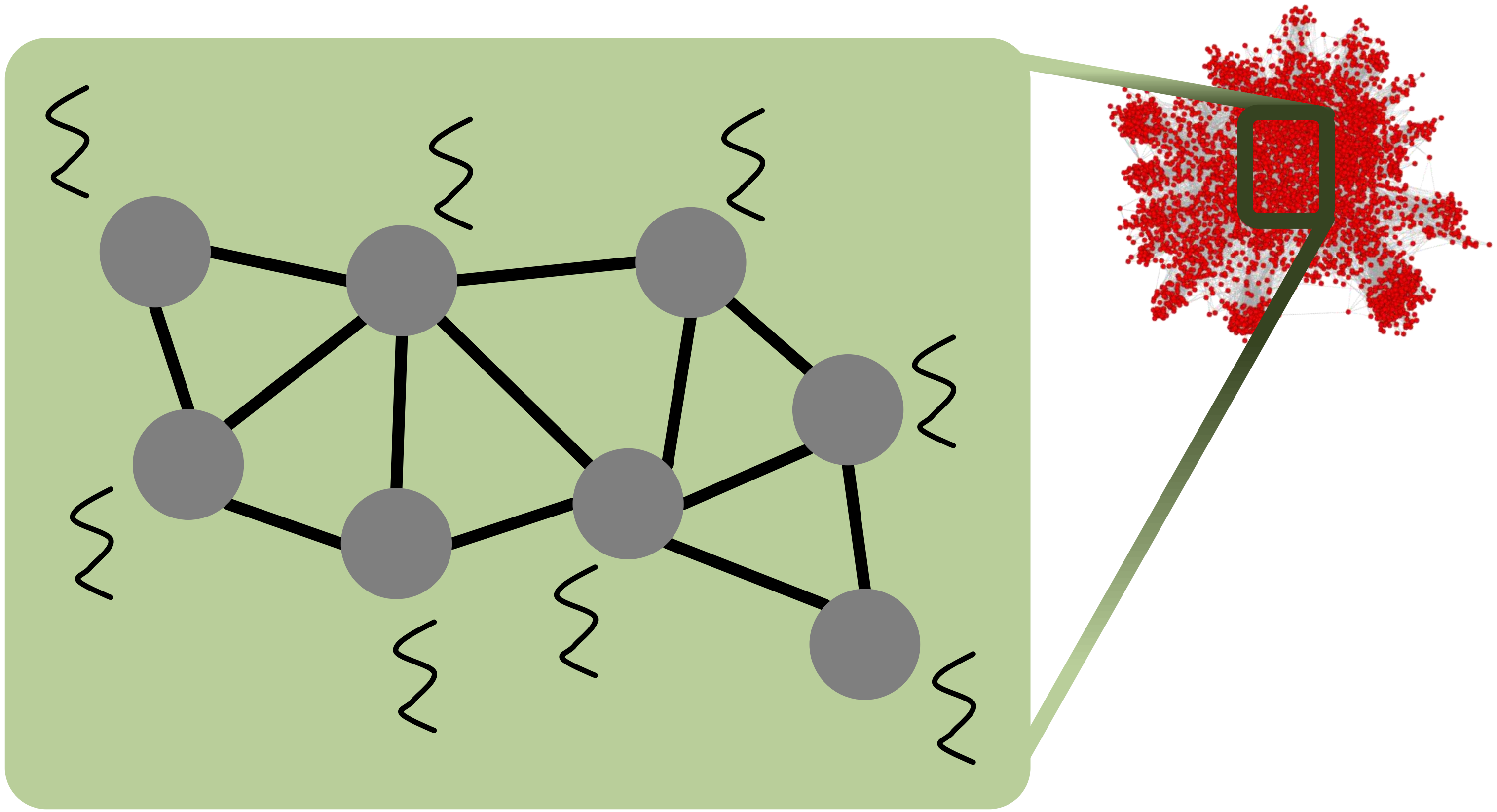
PageRank

P threads are used



PAGERANK

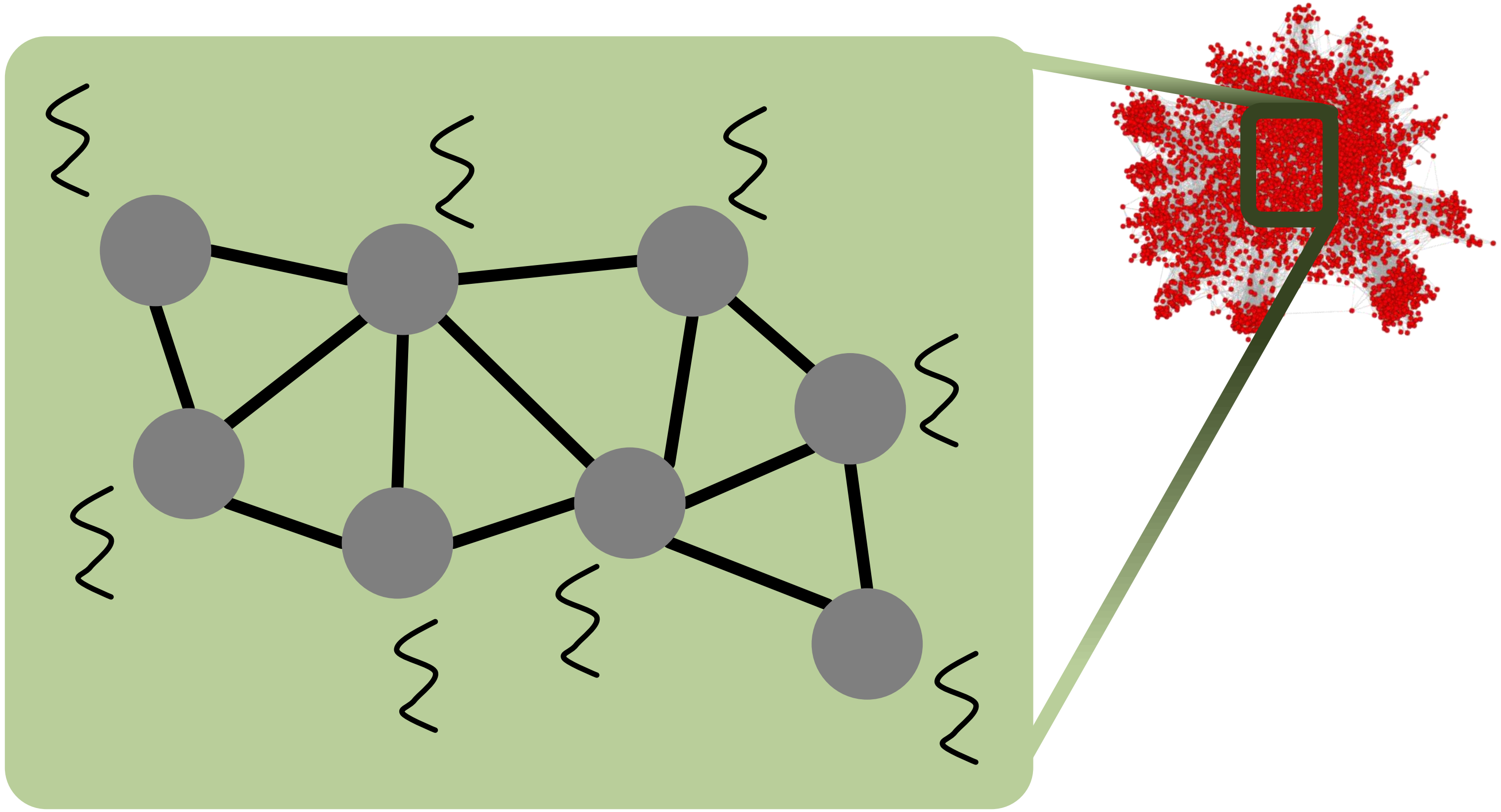
P threads are used



[1] J. J. Whang et al. Scalable Data-Driven PageRank: Algorithms, System Issues, and Lessons Learned. Euro-Par 2015.

PageRank

P threads are used

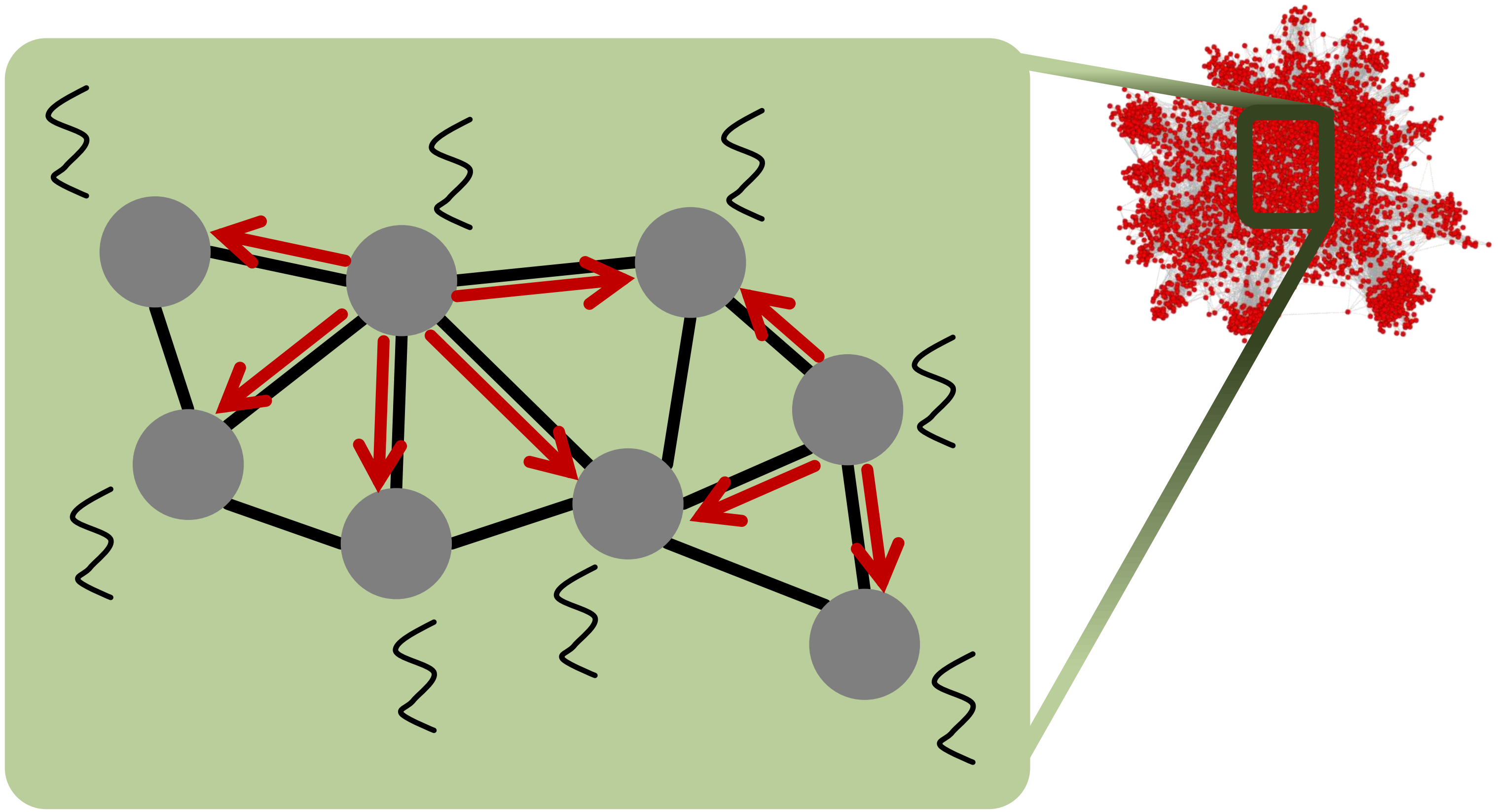


Pushing

[1] J. J. Whang et al. Scalable Data-Driven PageRank: Algorithms, System Issues, and Lessons Learned. Euro-Par 2015.

PageRank

P threads are used

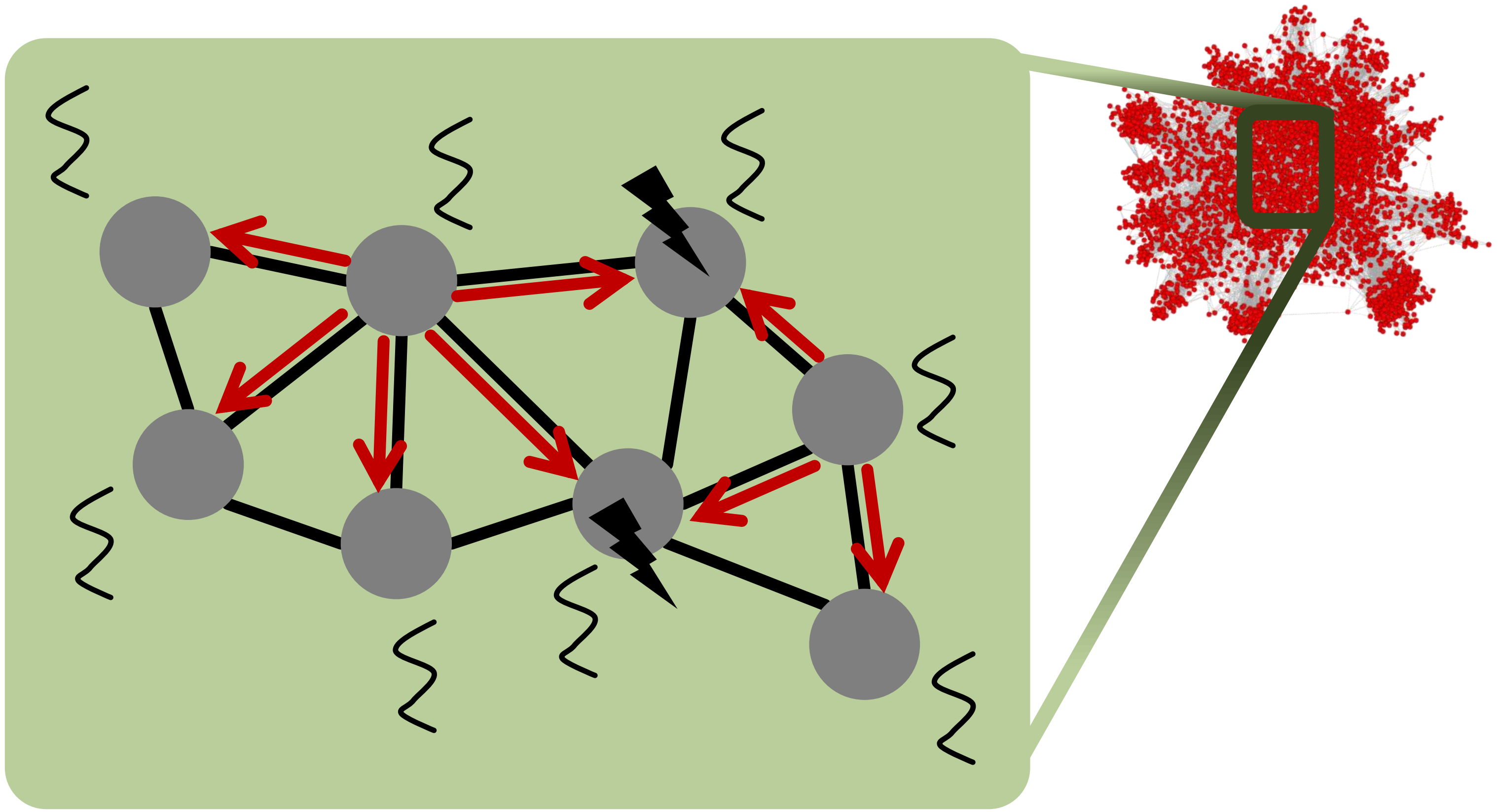


Pushing

[1] J. J. Whang et al. Scalable Data-Driven PageRank: Algorithms, System Issues, and Lessons Learned. Euro-Par 2015.

PageRank

P threads are used



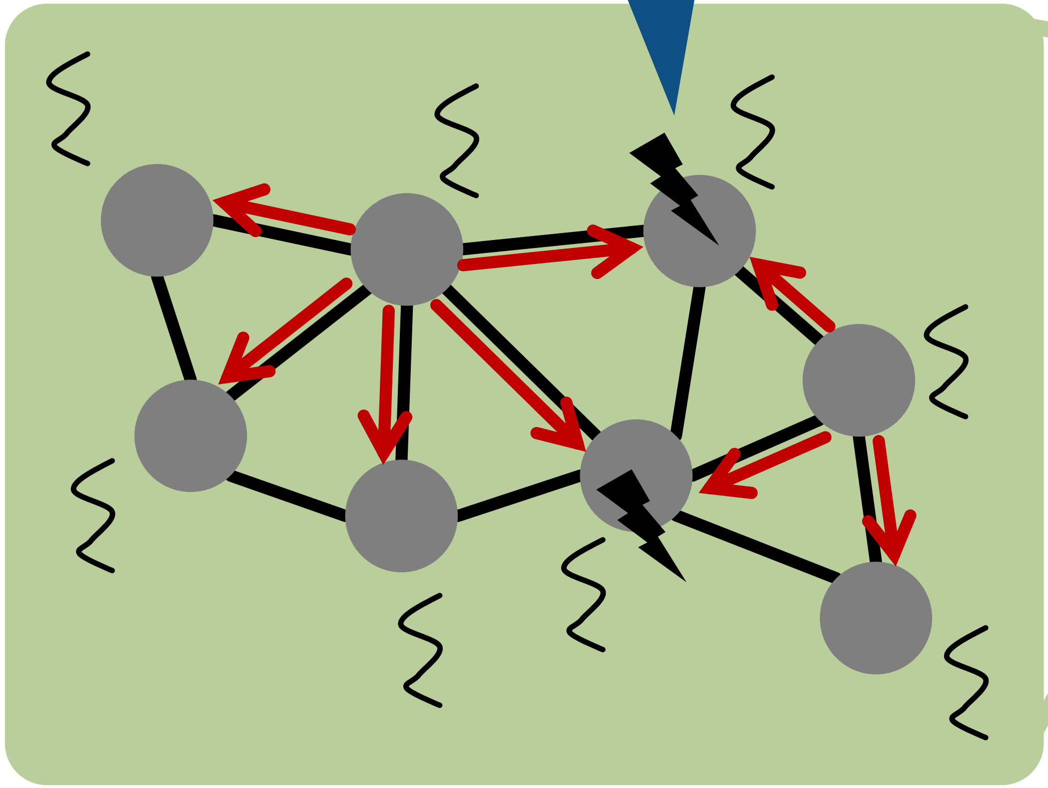
Pushing

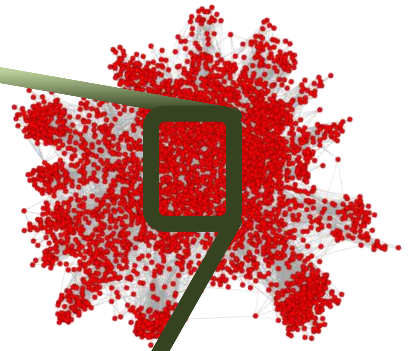
[1] J. J. Whang et al. Scalable Data-Driven PageRank: Algorithms, System Issues, and Lessons Learned. Euro-Par 2015.

PAGERANK

Write conflicts

P threads are used



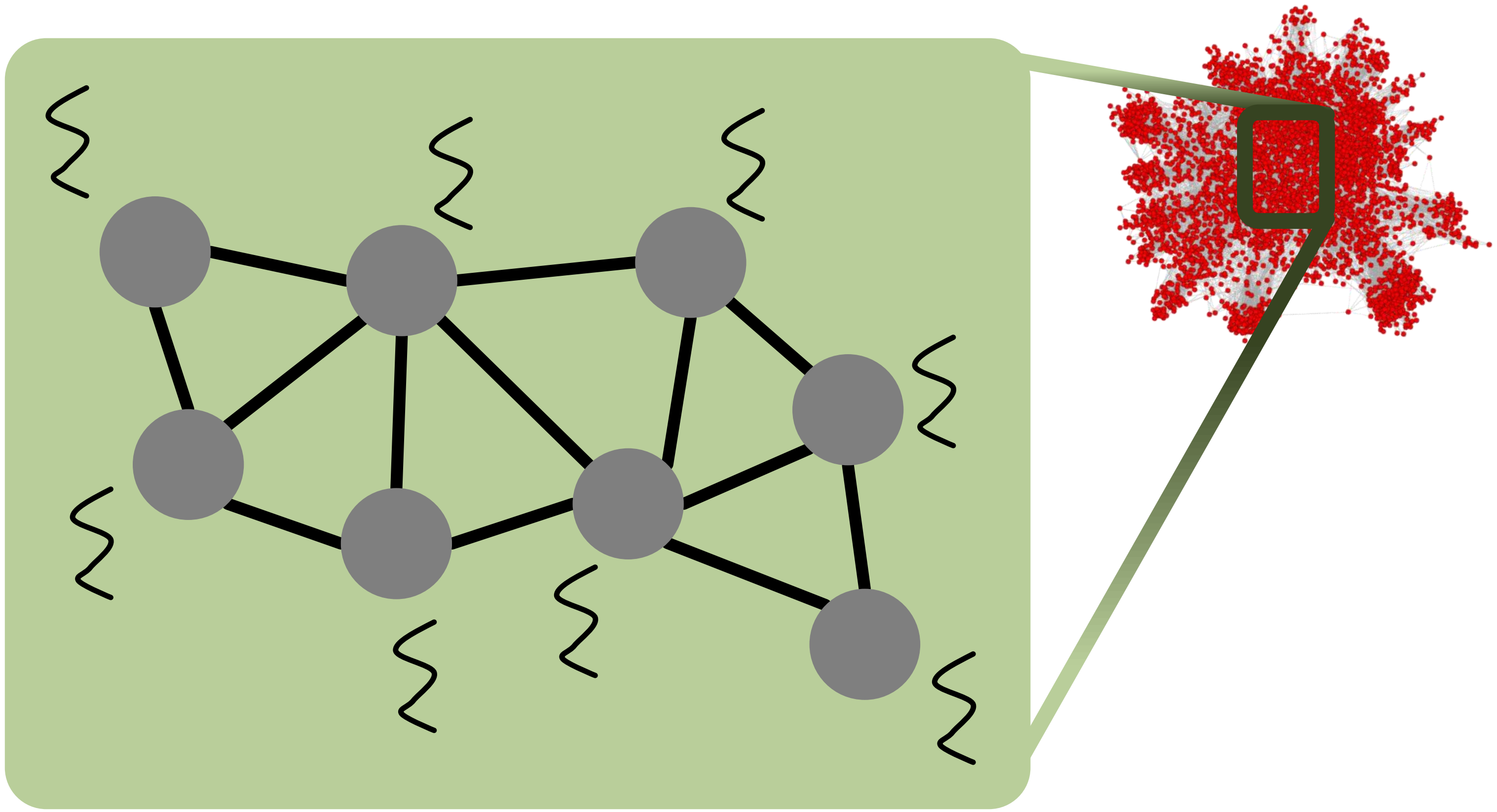


Pushing

[1] J. J. Whang et al. Scalable Data-Driven PageRank: Algorithms, System Issues, and Lessons Learned. Euro-Par 2015.

PageRank

P threads are used

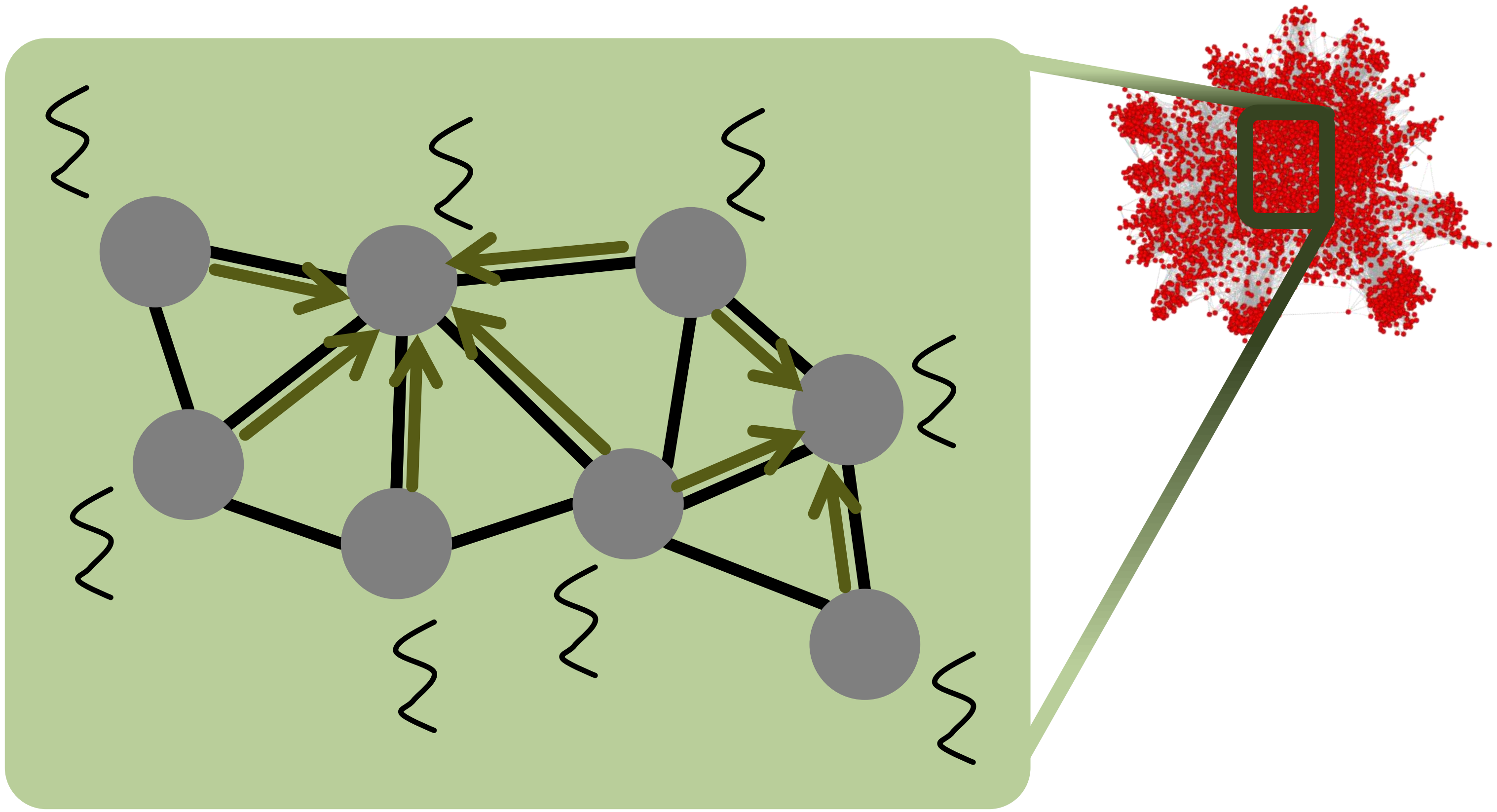


Pulling

[1] J. J. Whang et al. Scalable Data-Driven PageRank: Algorithms, System Issues, and Lessons Learned. Euro-Par 2015.

PAGERANK

P threads are used

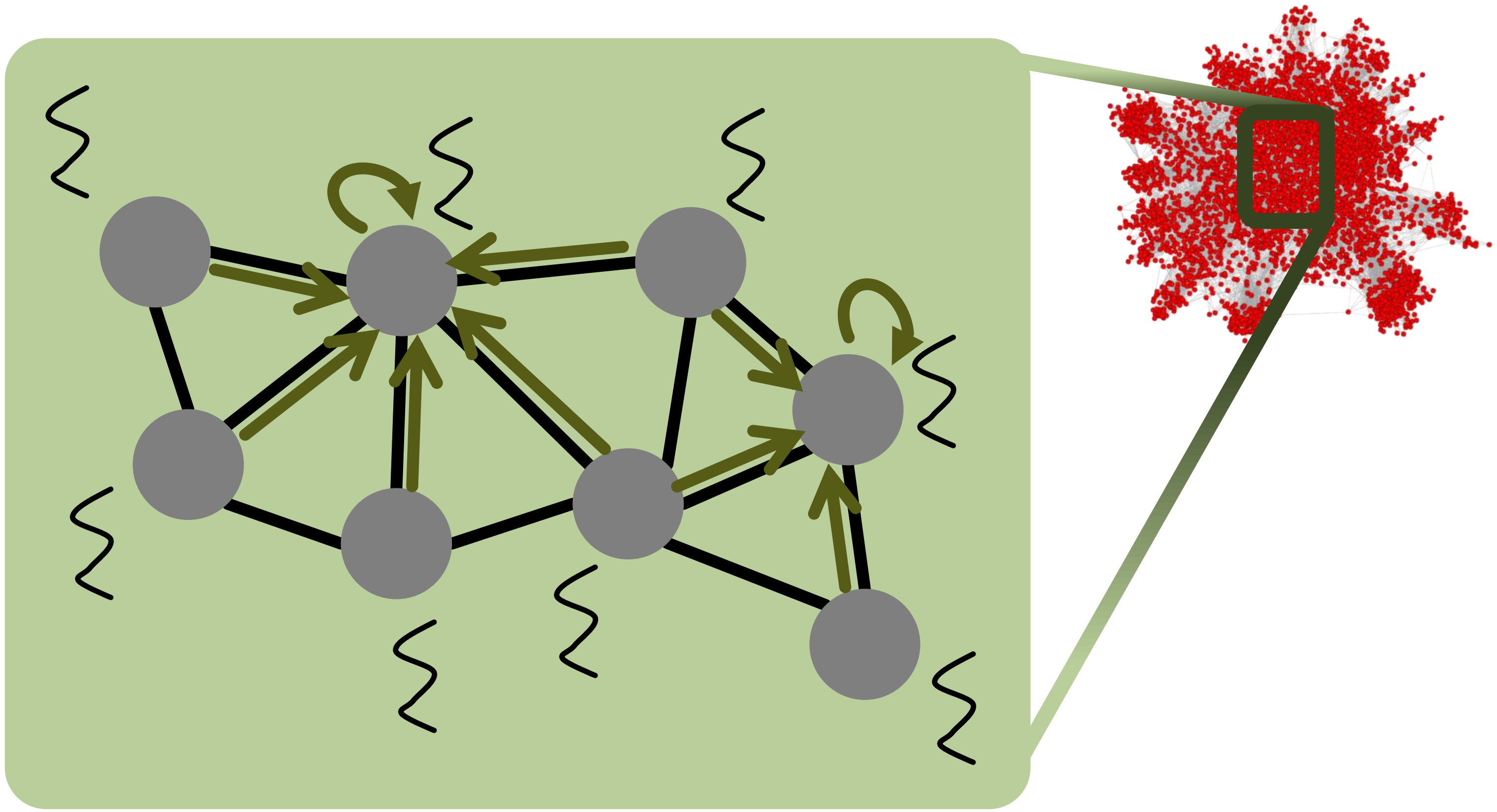


Pulling

[1] J. J. Whang et al. Scalable Data-Driven PageRank: Algorithms, System Issues, and Lessons Learned. Euro-Par 2015.

PAGERANK

P threads are used

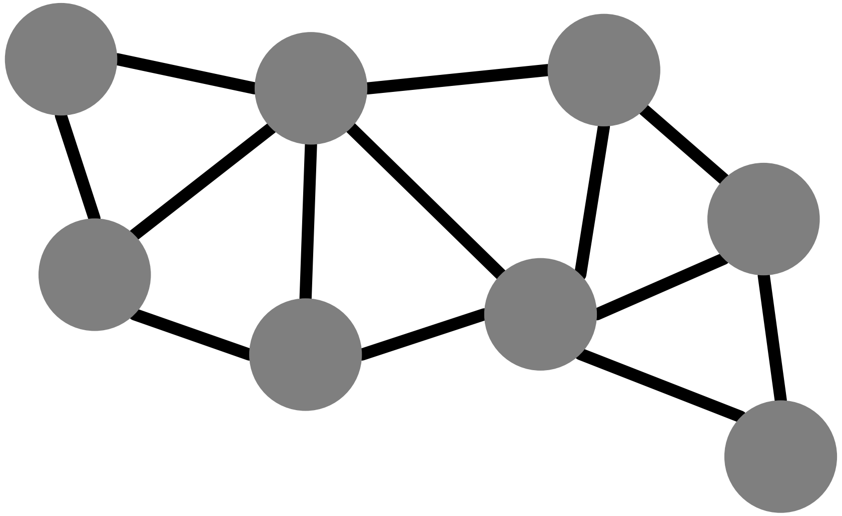


Pulling

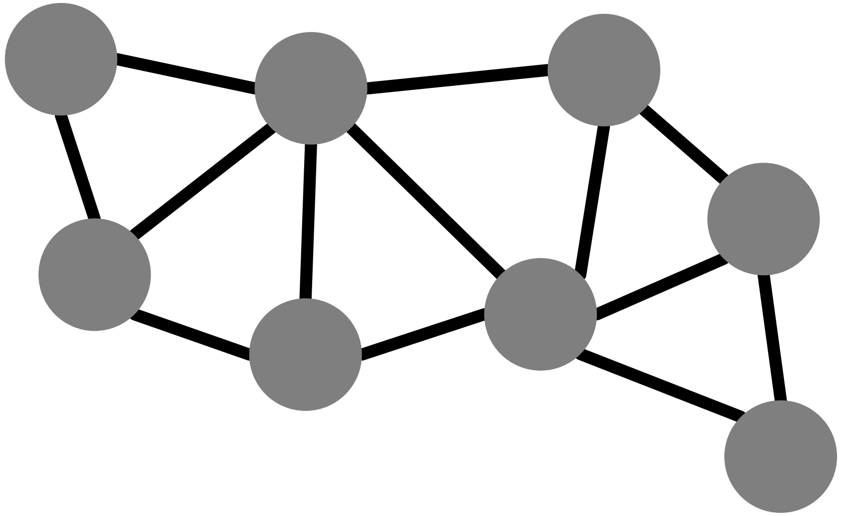
[1] J. J. Whang et al. Scalable Data-Driven PageRank: Algorithms, System Issues, and Lessons Learned. Euro-Par 2015.

BFS

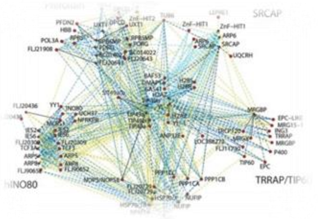
Top-Down vs. Bottom-Up [1]

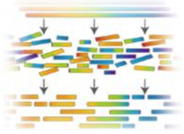


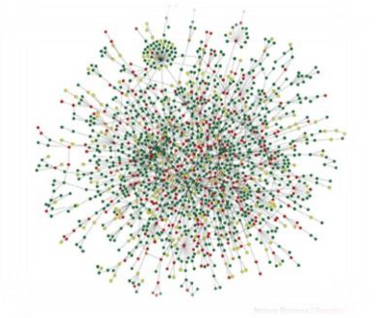
BFS
Top-Down vs. Bottom-Up [1]

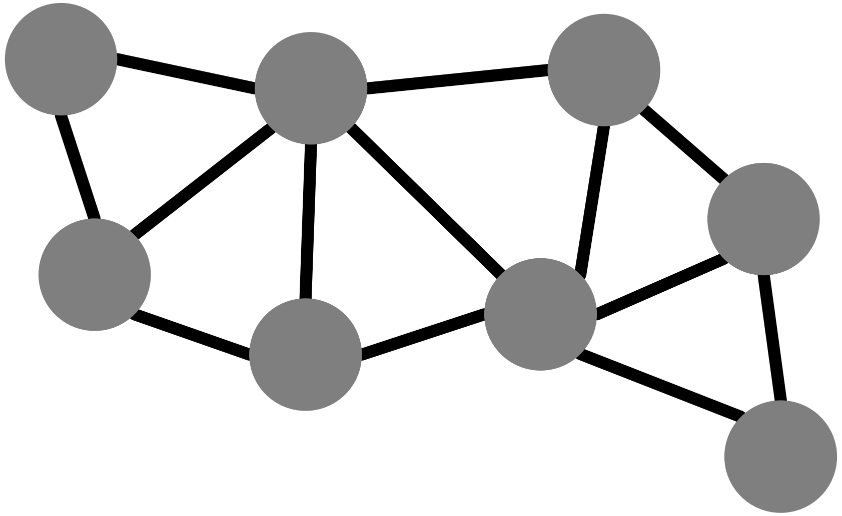


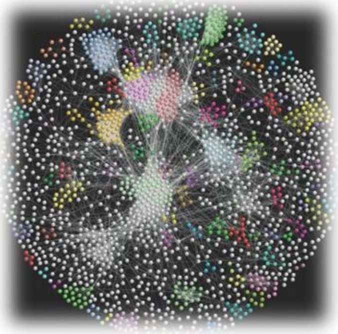
BFS
Top-Down vs. Bottom-Up [1]





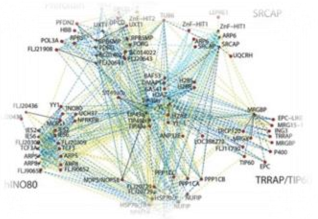


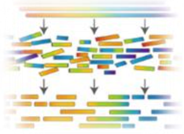


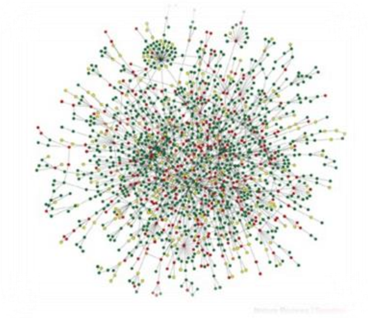


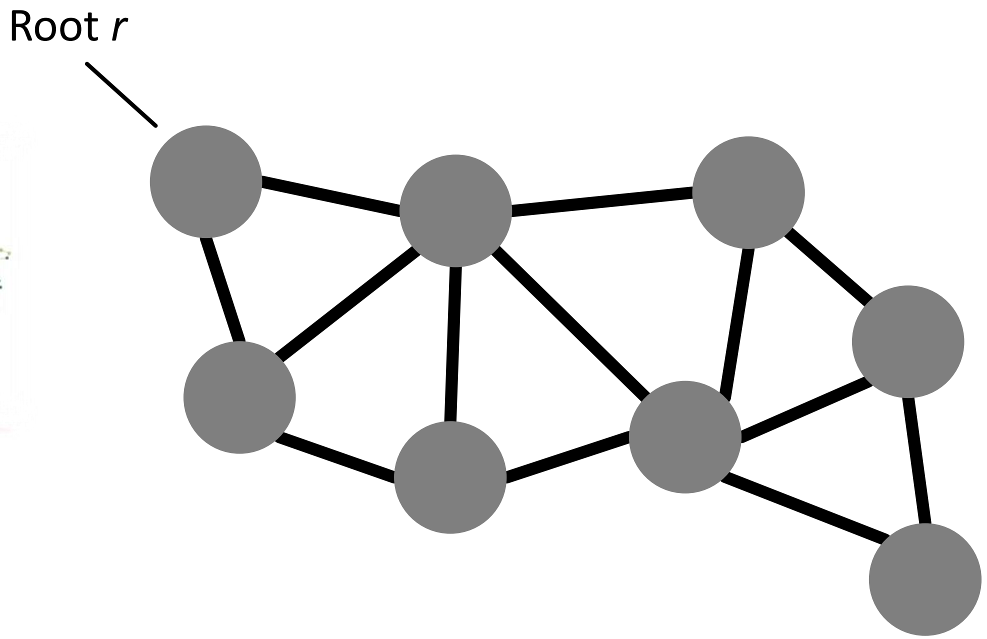
BFS

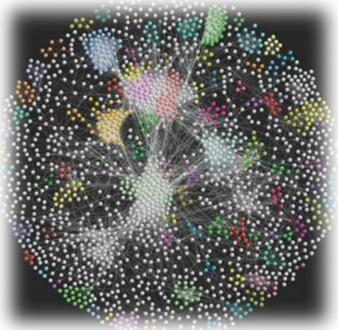
TOP-DOWN VS. BOTTOM-UP [1]



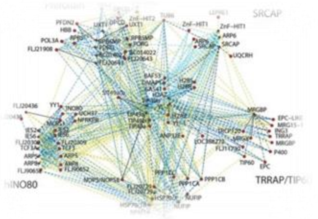


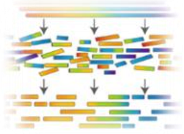


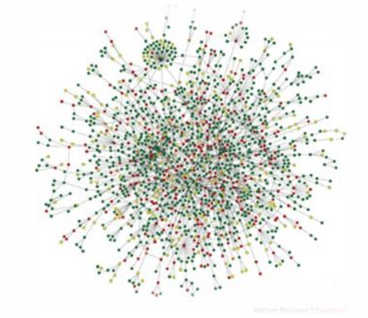


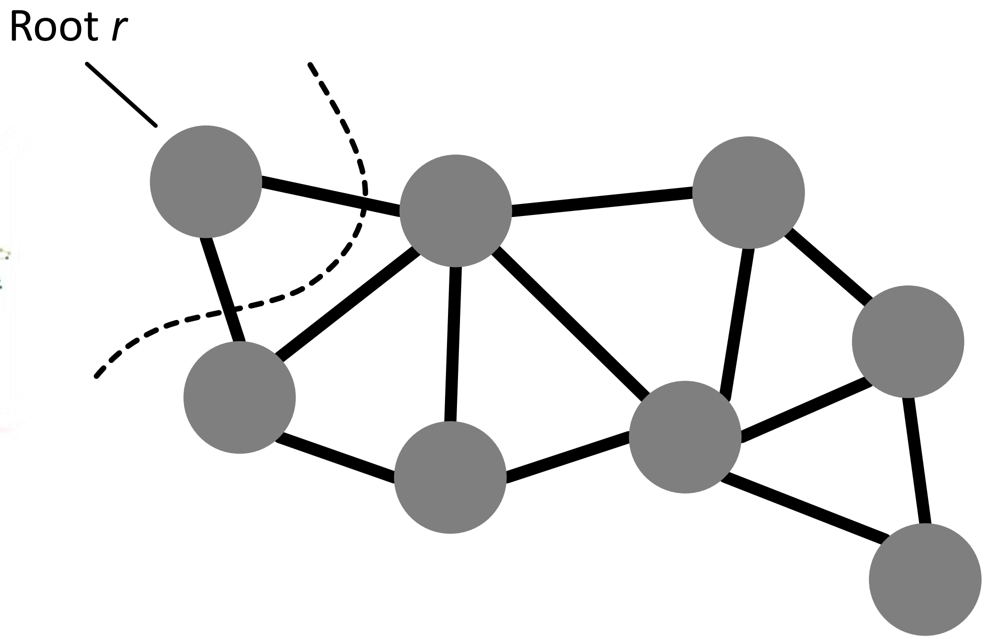


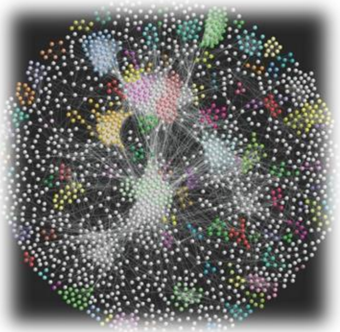
BFS
Top-Down vs. Bottom-Up [1]





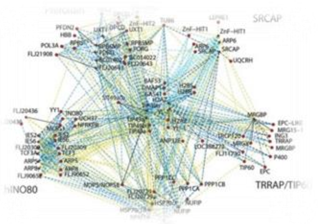


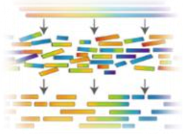


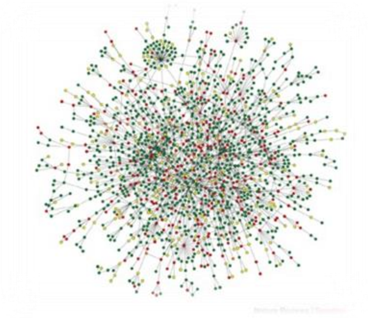


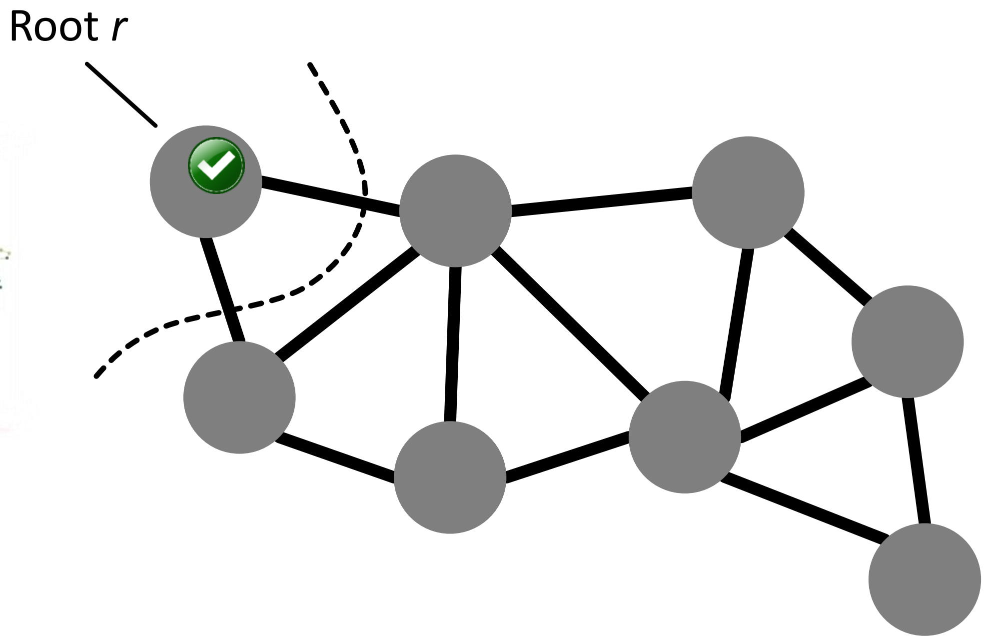
BFS

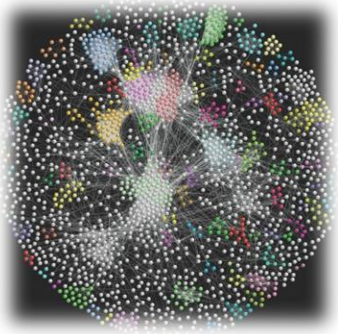
Top-Down vs. Bottom-Up [1]



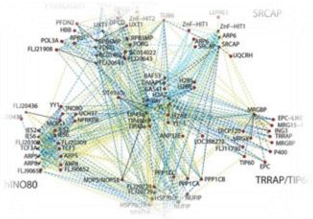


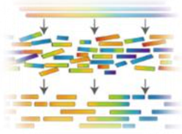


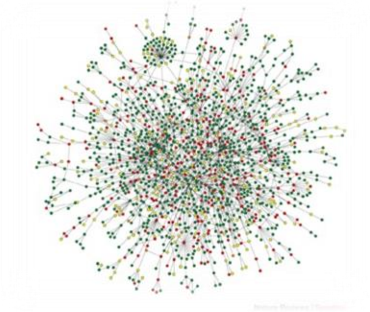


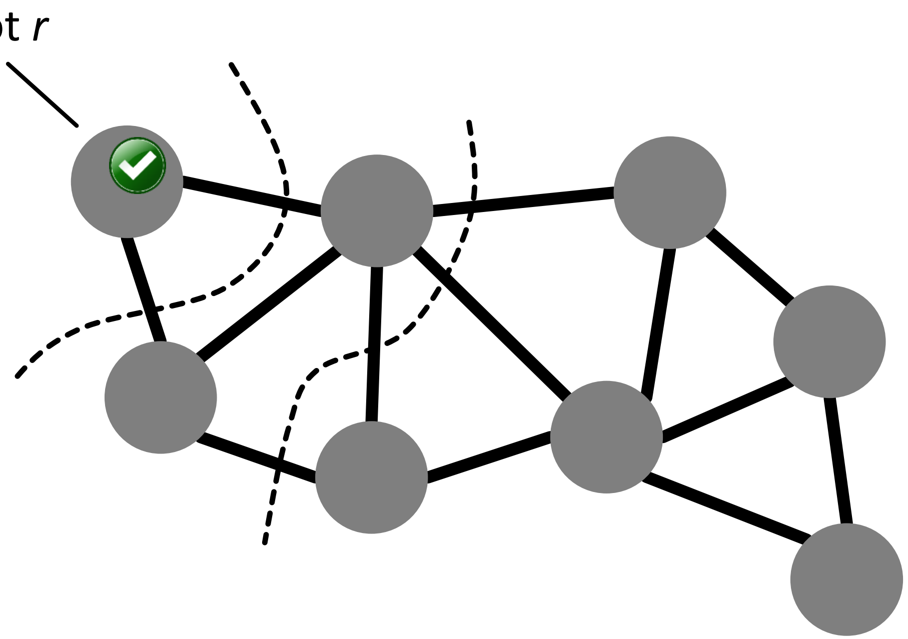


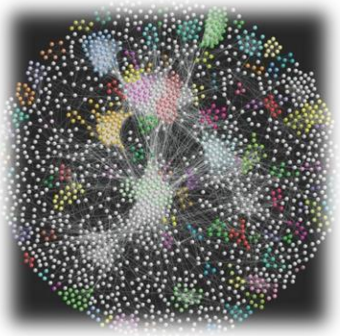
BFS
Top-Down vs. Bottom-Up [1]







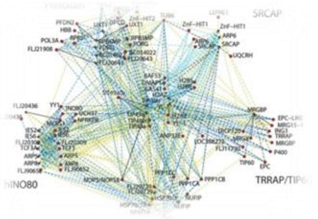


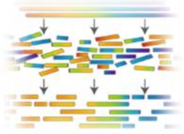


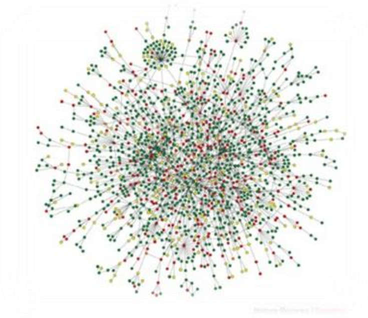
GRAPH 500

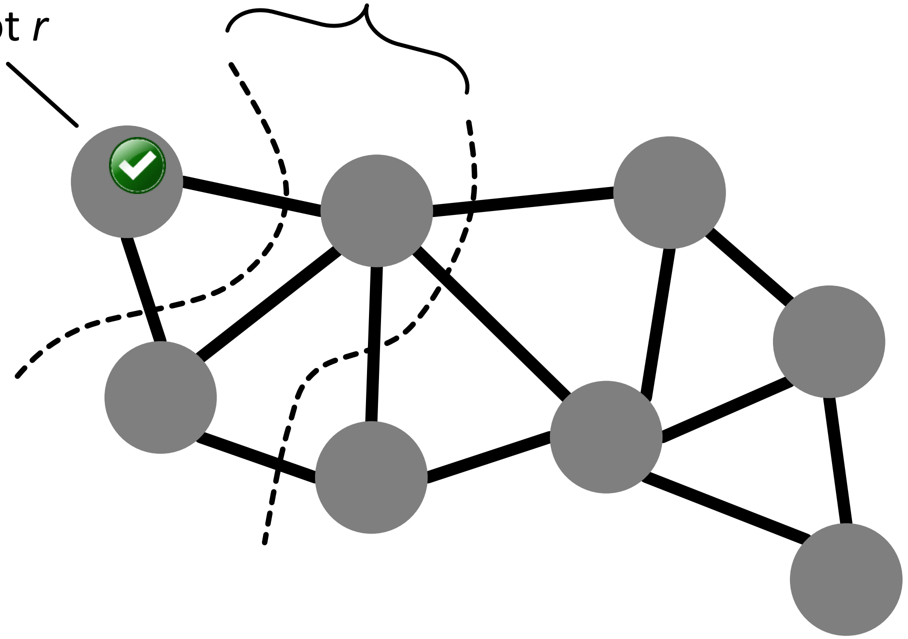
BFS

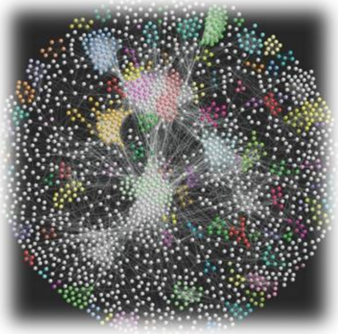
Top-Down vs. Bottom-Up [1]



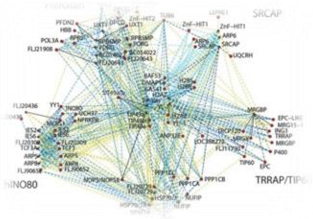


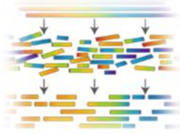


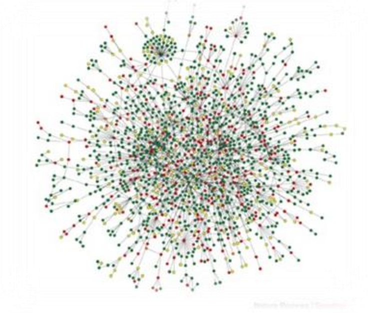


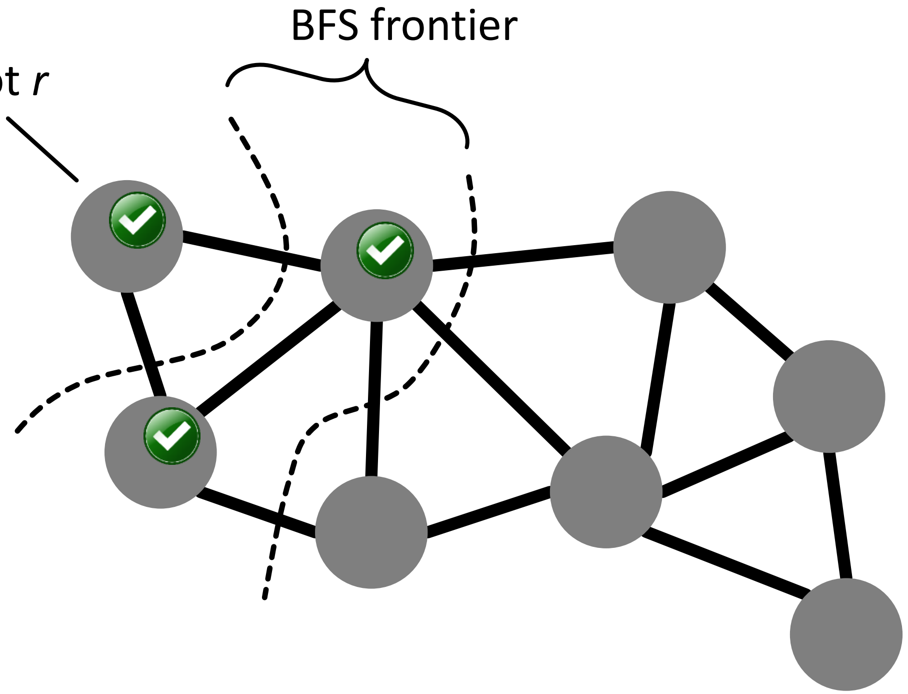


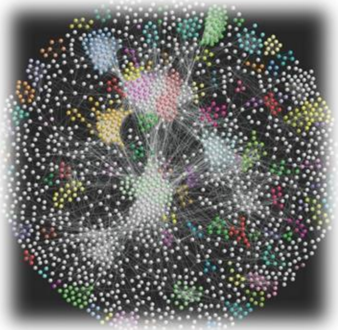
BFS
Top-Down vs. Bottom-Up [1]



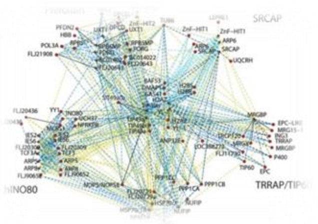


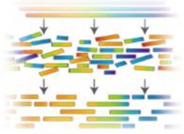


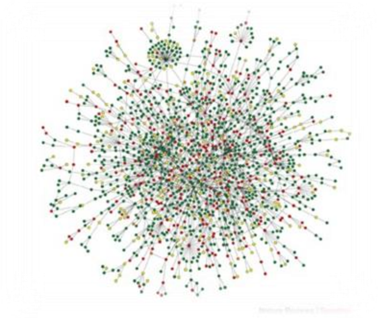


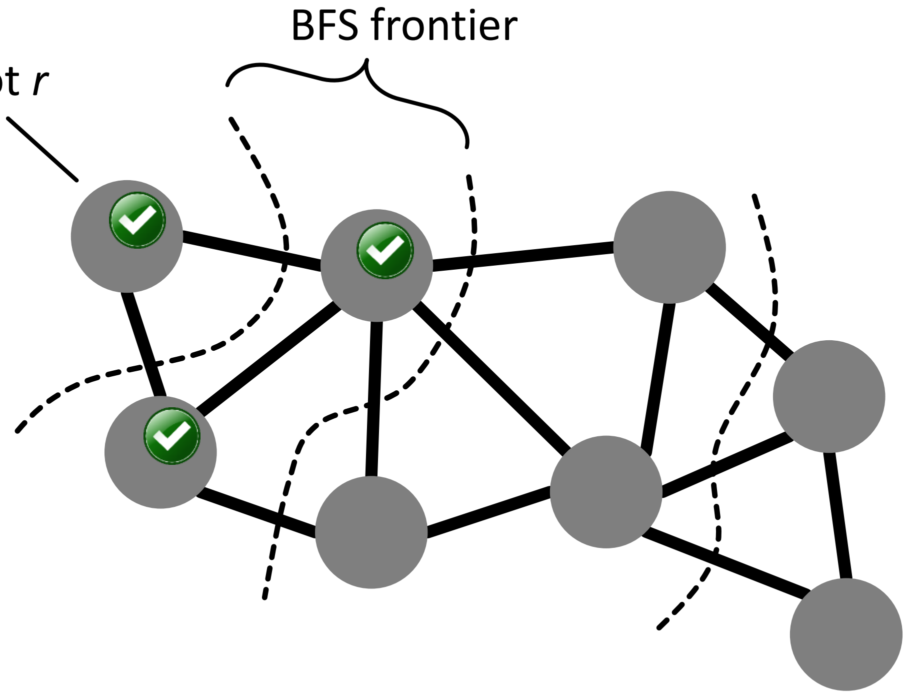


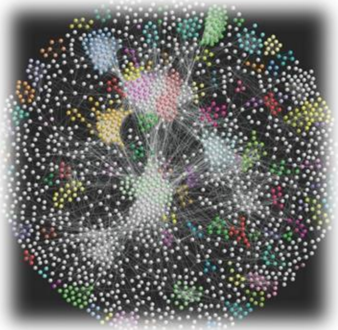
BFS
TOP-DOWN VS. BOTTOM-UP [1]



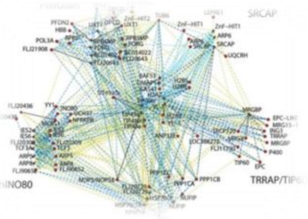


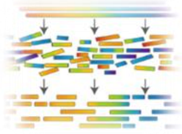


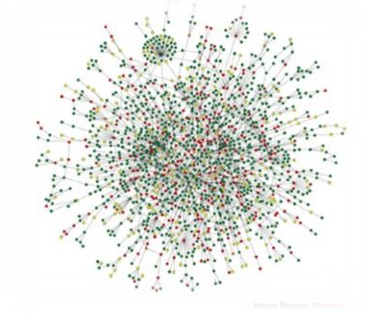


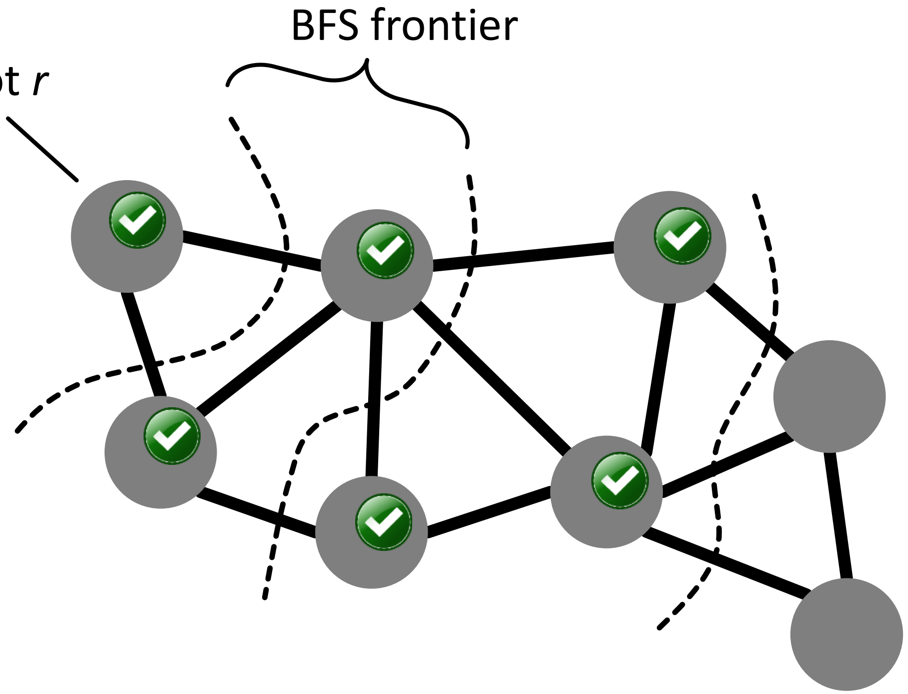


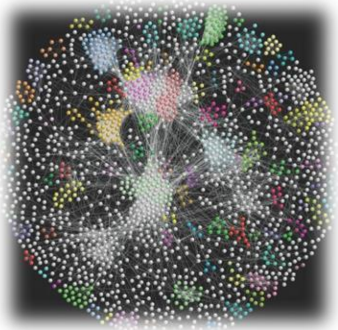
BFS
Top-Down vs. Bottom-Up [1]





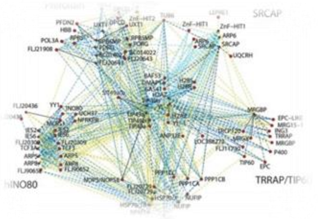


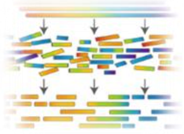


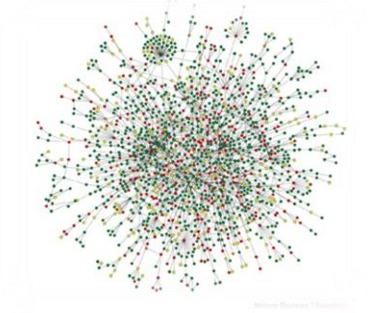


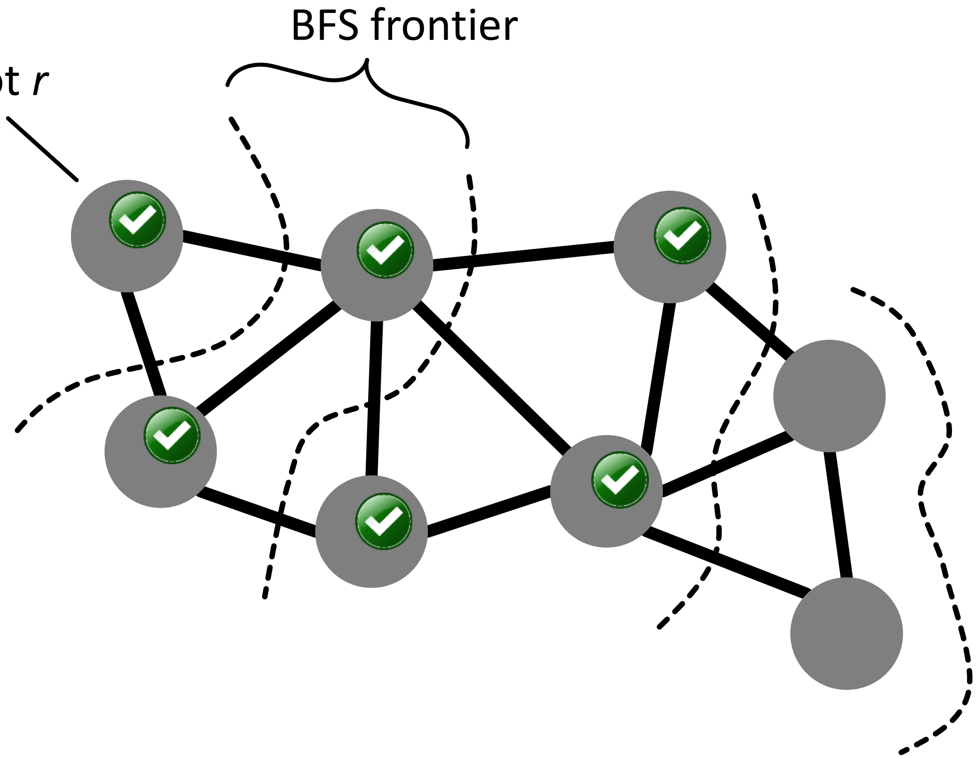
BFS

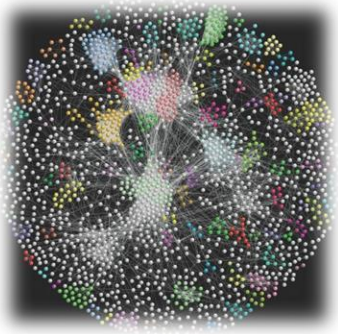
TOP-DOWN VS. BOTTOM-UP [1]



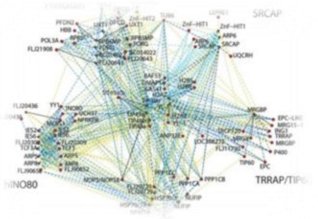


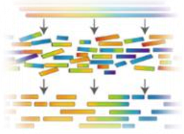


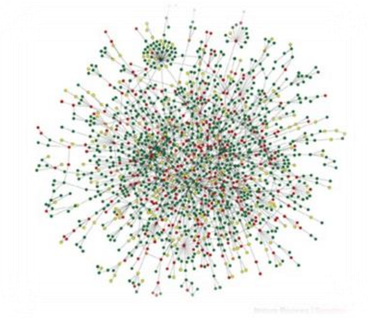


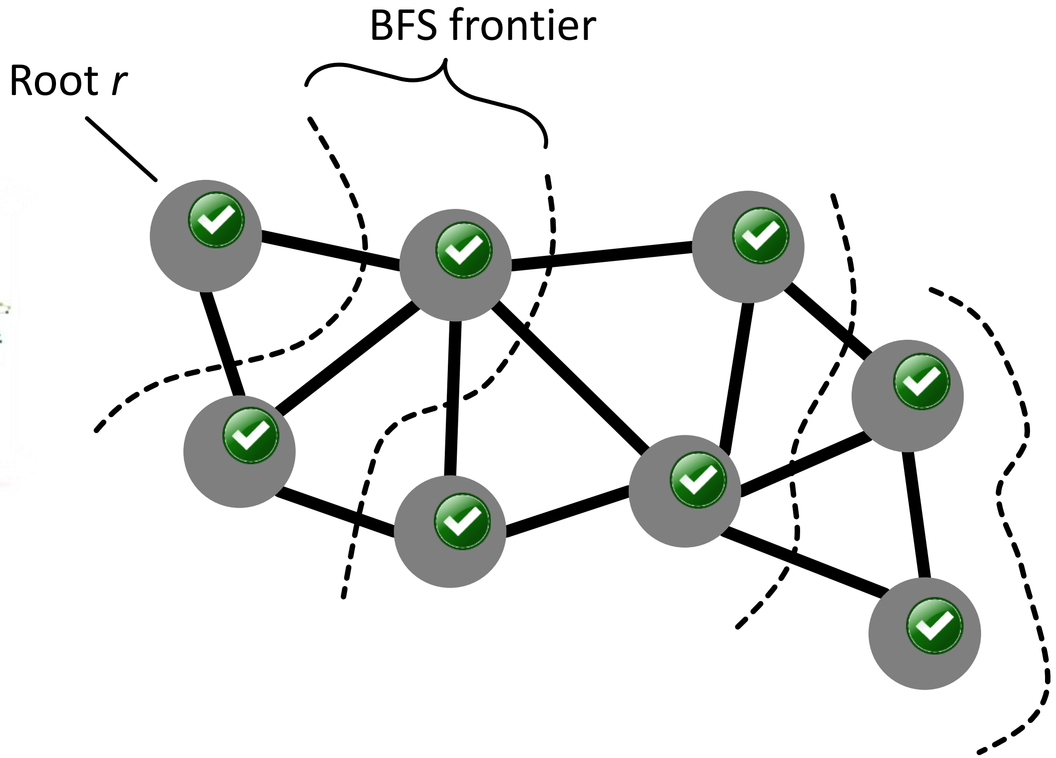


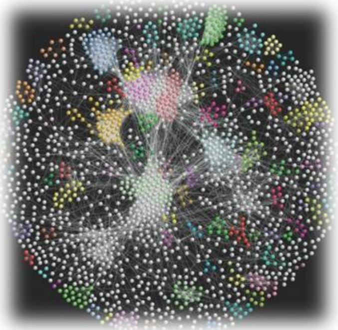
BFS
Top-Down vs. Bottom-Up [1]





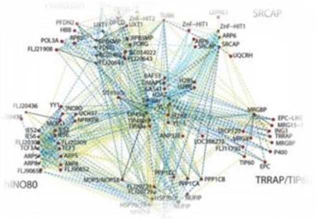


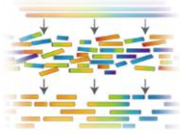


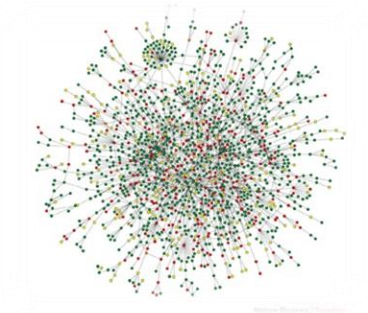


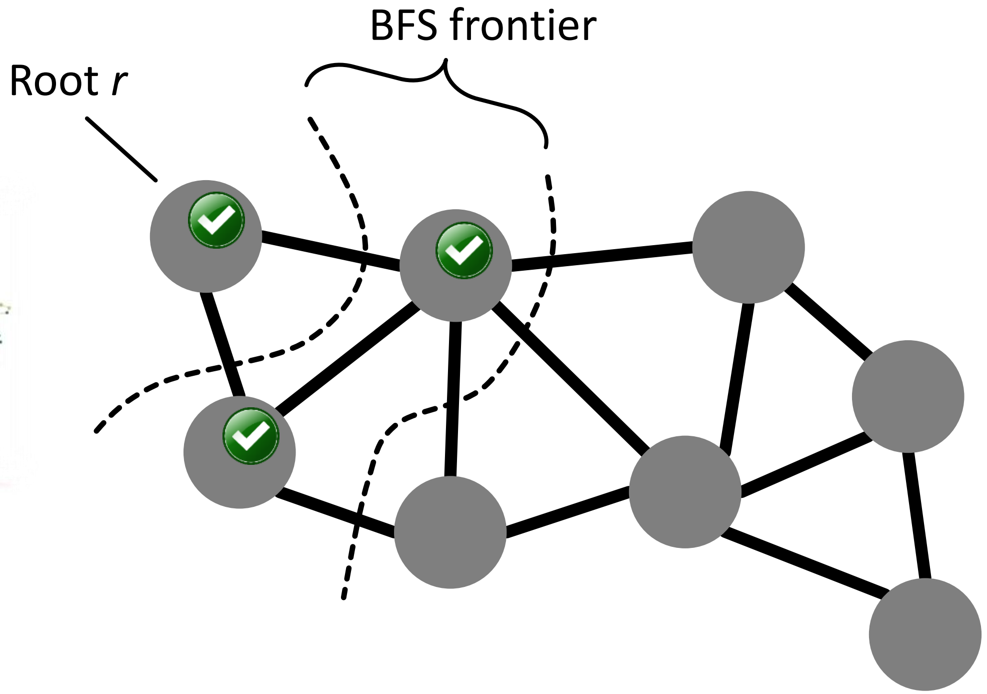
BFS

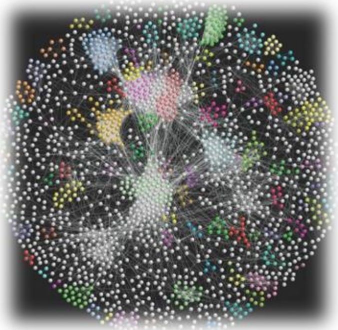
TOP-DOWN VS. BOTTOM-UP [1]



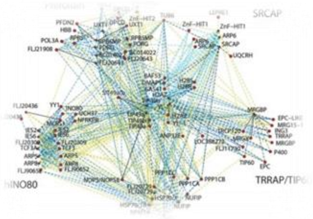


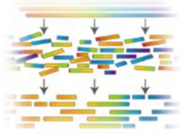




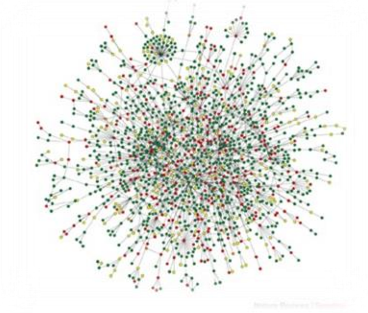


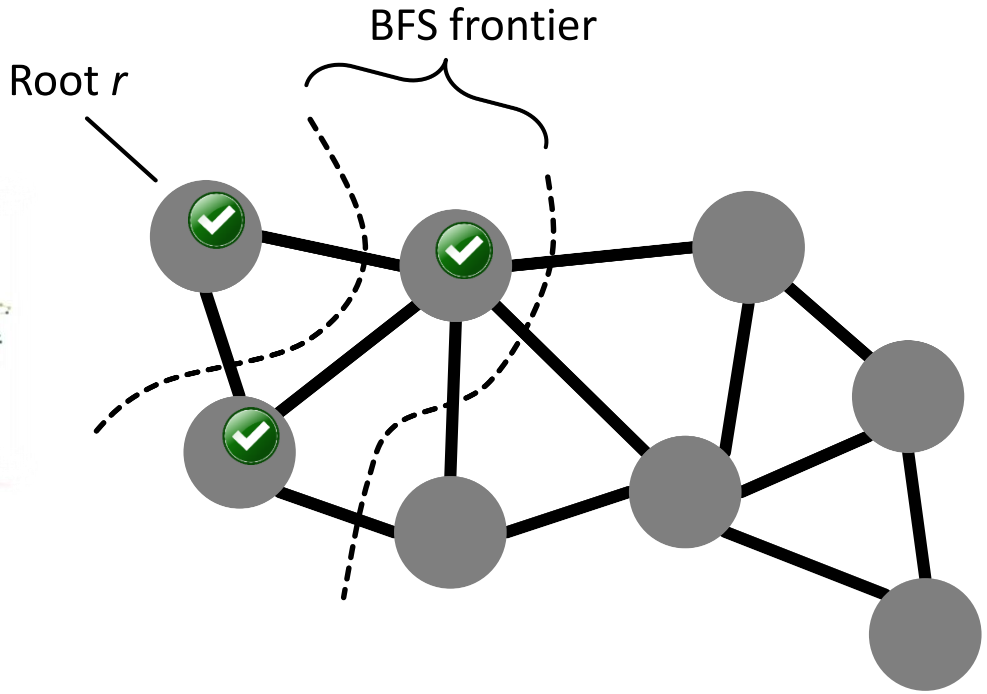
BFS
Top-Down vs. Bottom-Up [1]

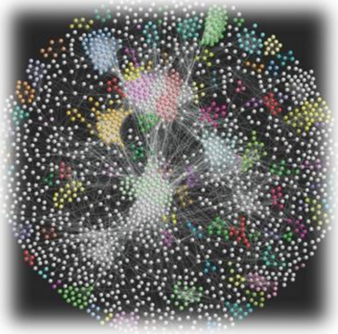




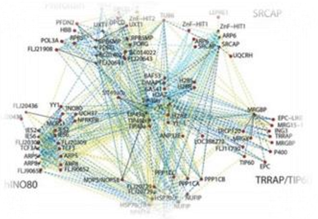
Pushing or pulling when expanding a frontier

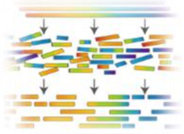


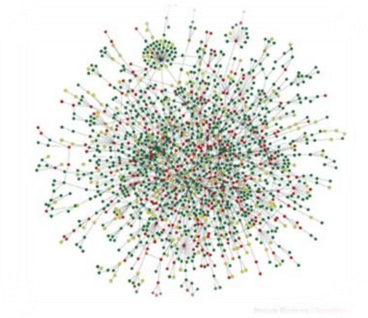




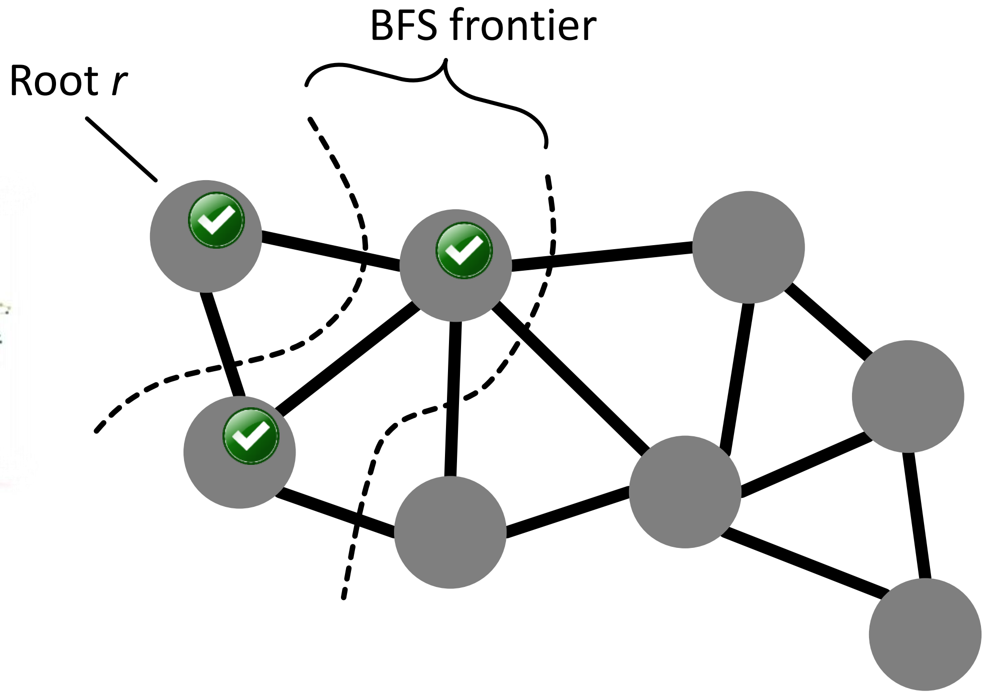
BFS
Top-Down vs. Bottom-Up [1]

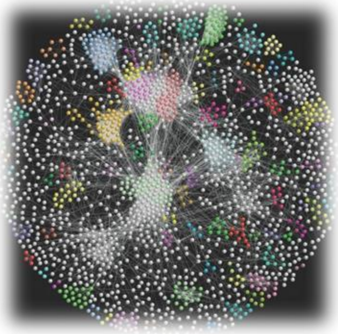






Pushing or pulling when expanding a frontier



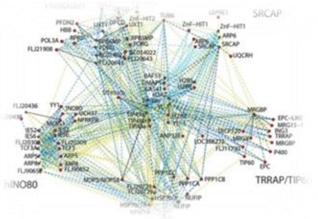


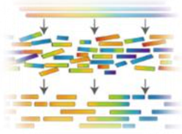
Pushing

[1] S. Beamer, K. Asanović, and D. Patterson. Direction-optimizing breadth-first search. SC12.

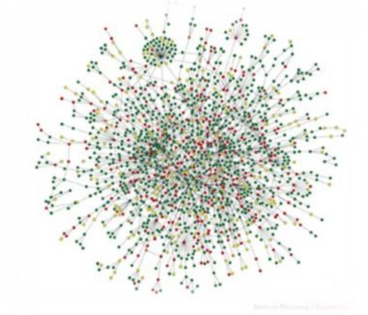
BFS

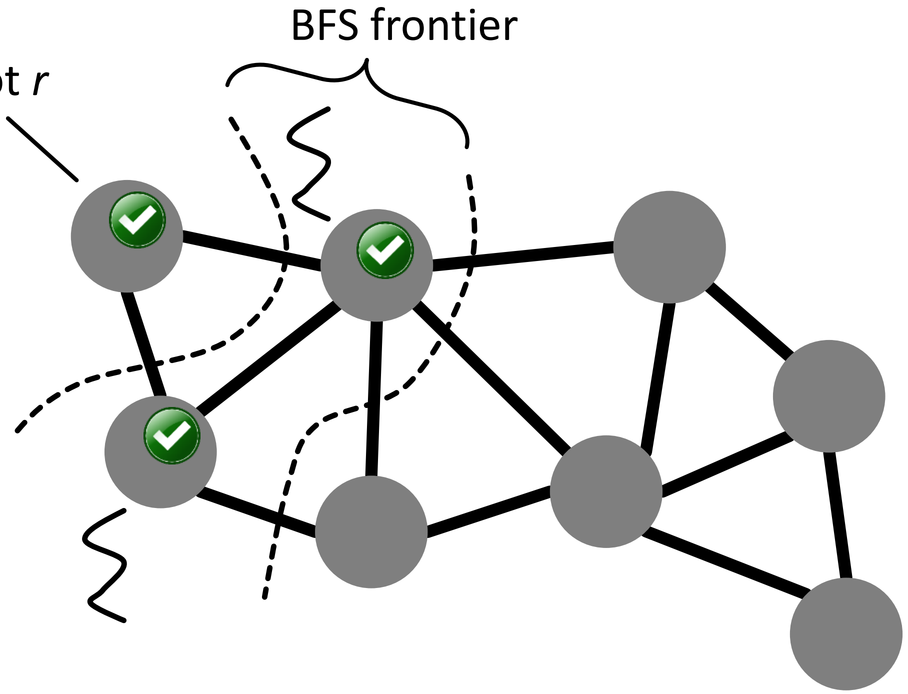
Top-Down vs. Bottom-Up [1]

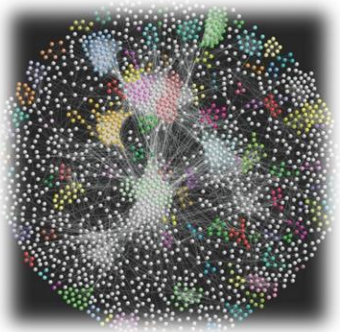




Pushing or pulling when expanding a frontier





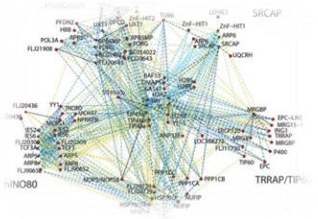


Pushing

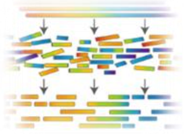
[1] S. Beamer, K. Asanović, and D. Patterson. Direction-optimizing breadth-first search. SC12.

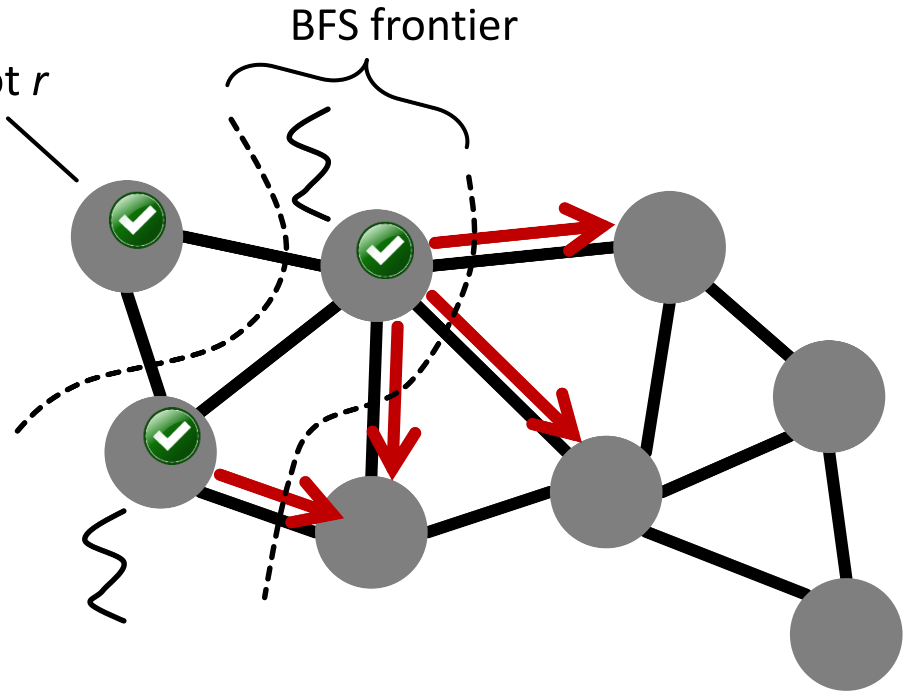
BFS

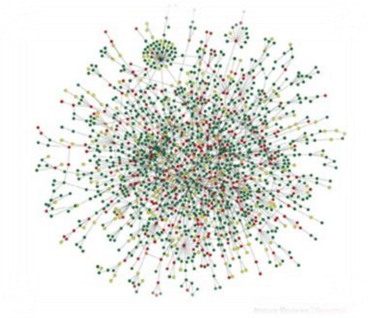
Top-Down vs. Bottom-Up [1]

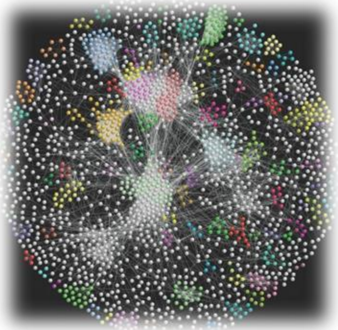


Pushing or pulling when expanding a frontier







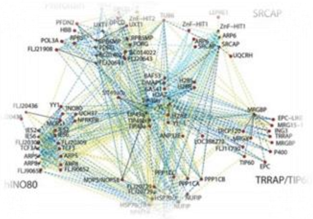


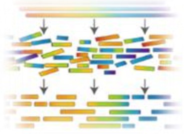
Pushing

[1] S. Beamer, K. Asanović, and D. Patterson. Direction-optimizing breadth-first search. SC12.

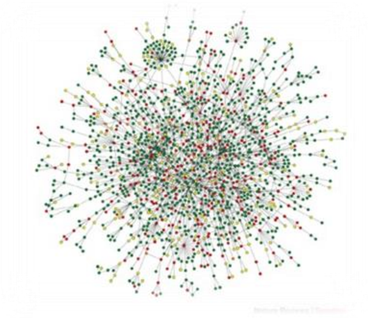
BFS

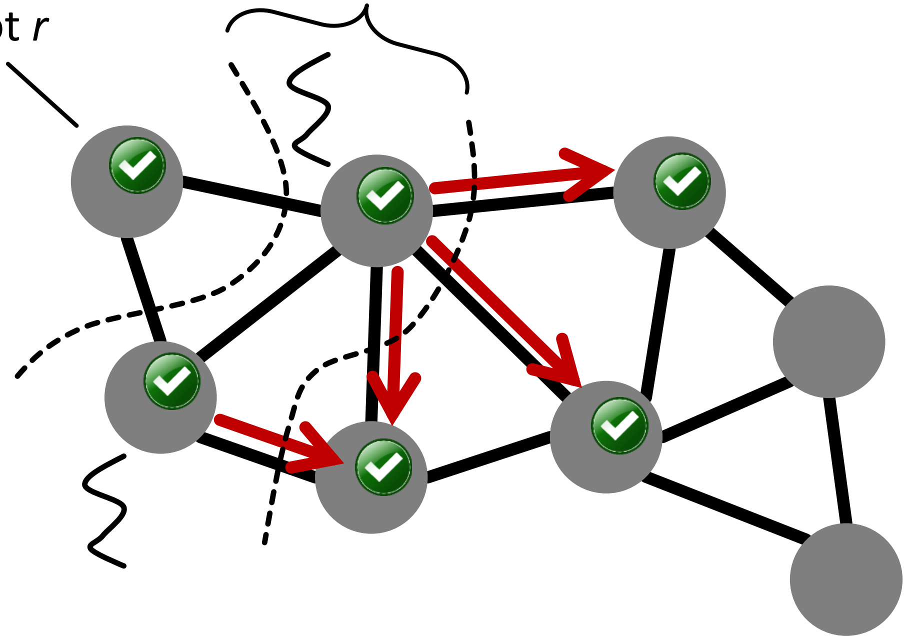
Top-Down vs. Bottom-Up [1]

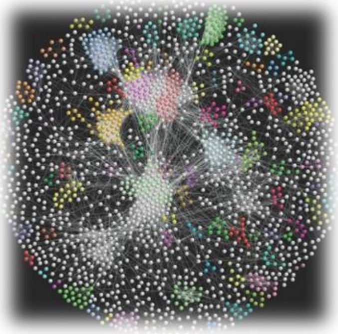




Pushing or pulling when expanding a frontier



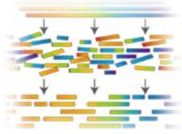


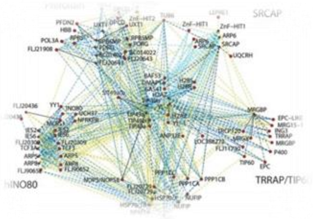


Pushing

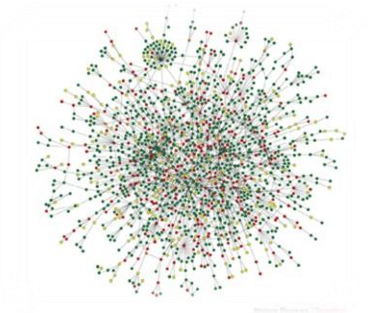
[1] S. Beamer, K. Asanović, and D. Patterson. Direction-optimizing breadth-first search. SC12.

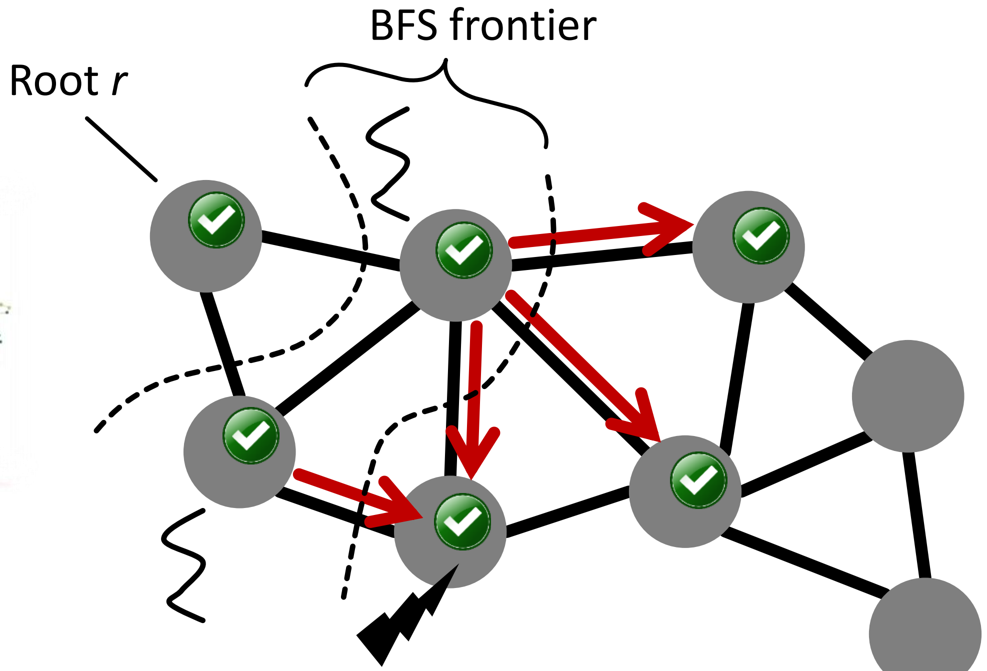
BFS
Top-Down vs. Bottom-Up [1]

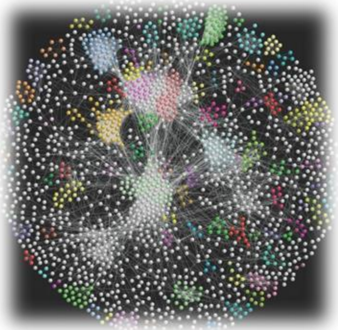




Pushing or pulling when expanding a frontier



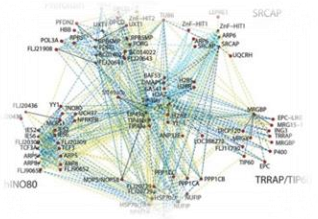


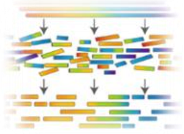


Pushing

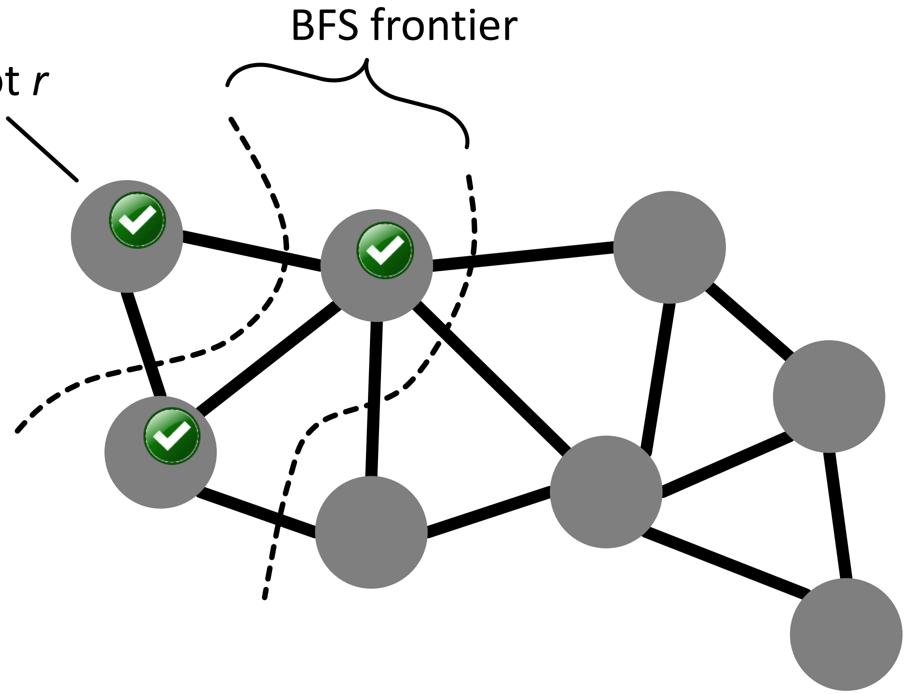
[1] S. Beamer, K. Asanović, and D. Patterson. Direction-optimizing breadth-first search. SC12.

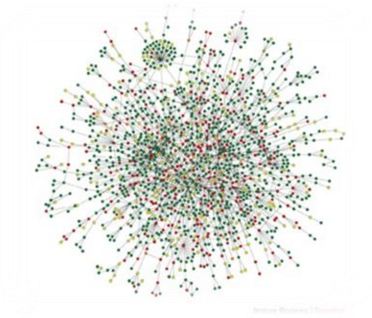
BFS
Top-Down vs. Bottom-Up [1]

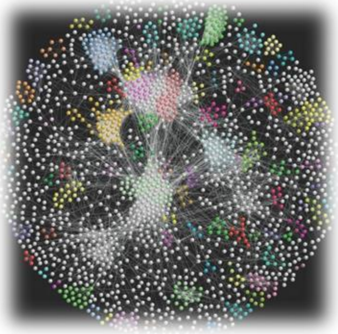




Pushing or pulling when expanding a frontier

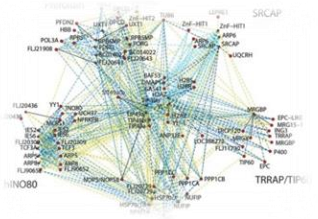


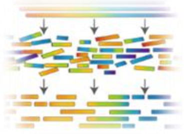




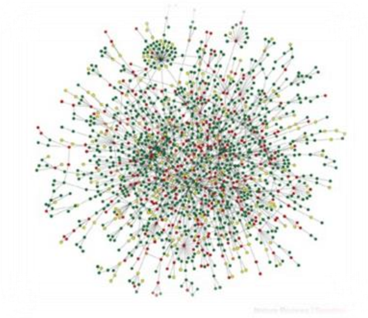
[1] S. Beamer, K. Asanović, and D. Patterson. Direction-optimizing breadth-first search. SC12.

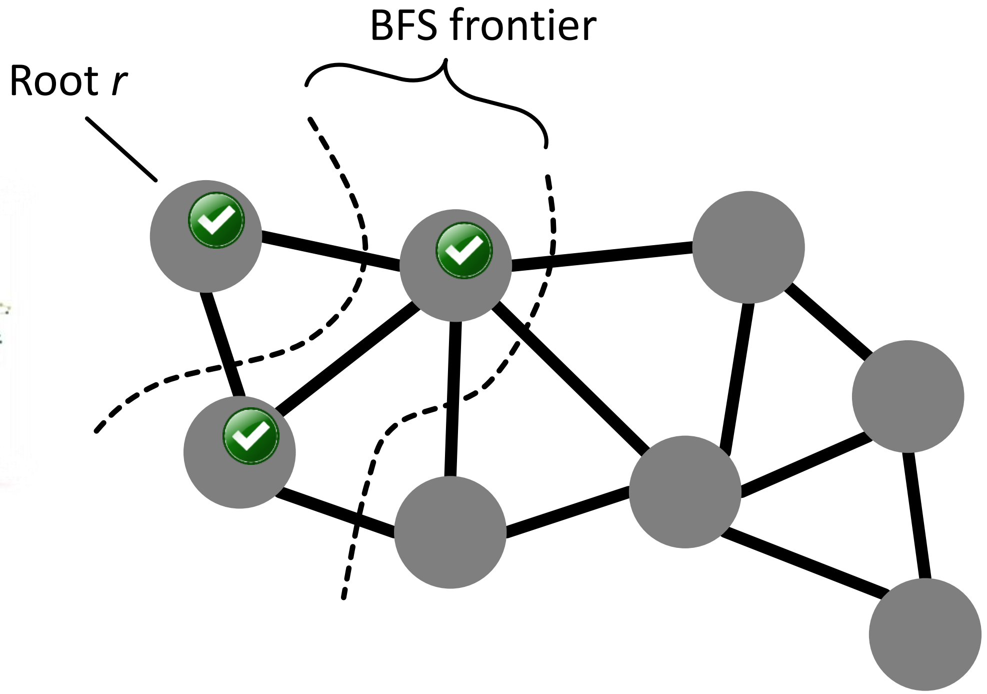
BFS
Top-Down vs. Bottom-Up [1]

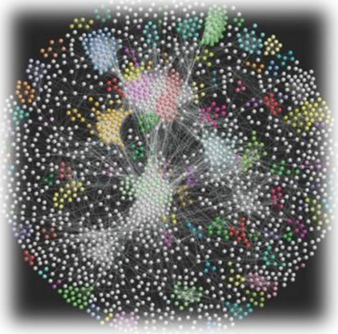




Pushing or pulling when expanding a frontier



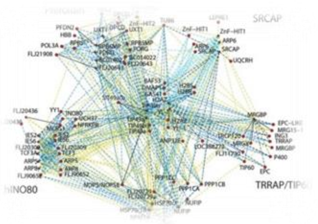


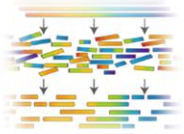


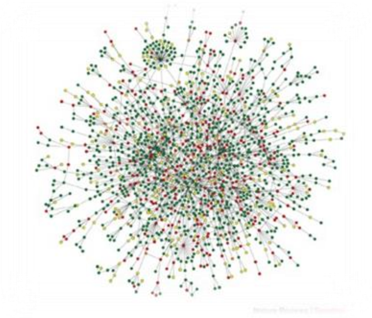
Pulling

[1] S. Beamer, K. Asanović, and D. Patterson. Direction-optimizing breadth-first search. SC12.

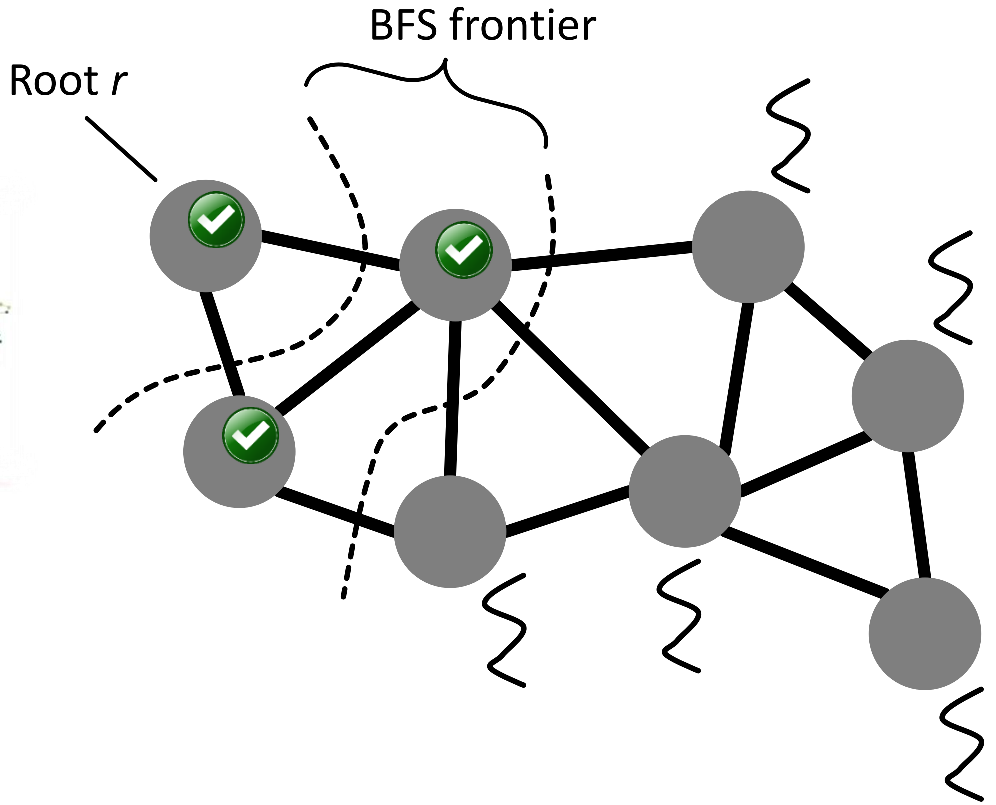
BFS
Top-Down vs. Bottom-Up [1]

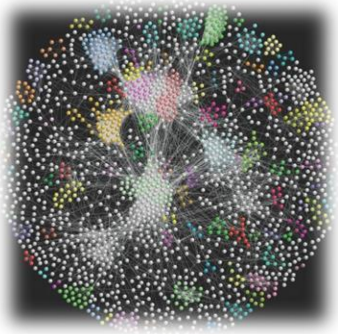






Pushing or pulling when expanding a frontier

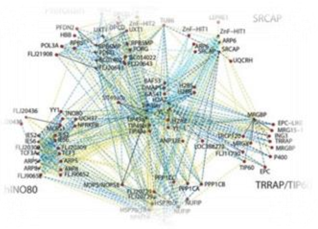


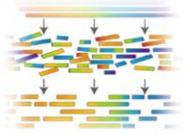


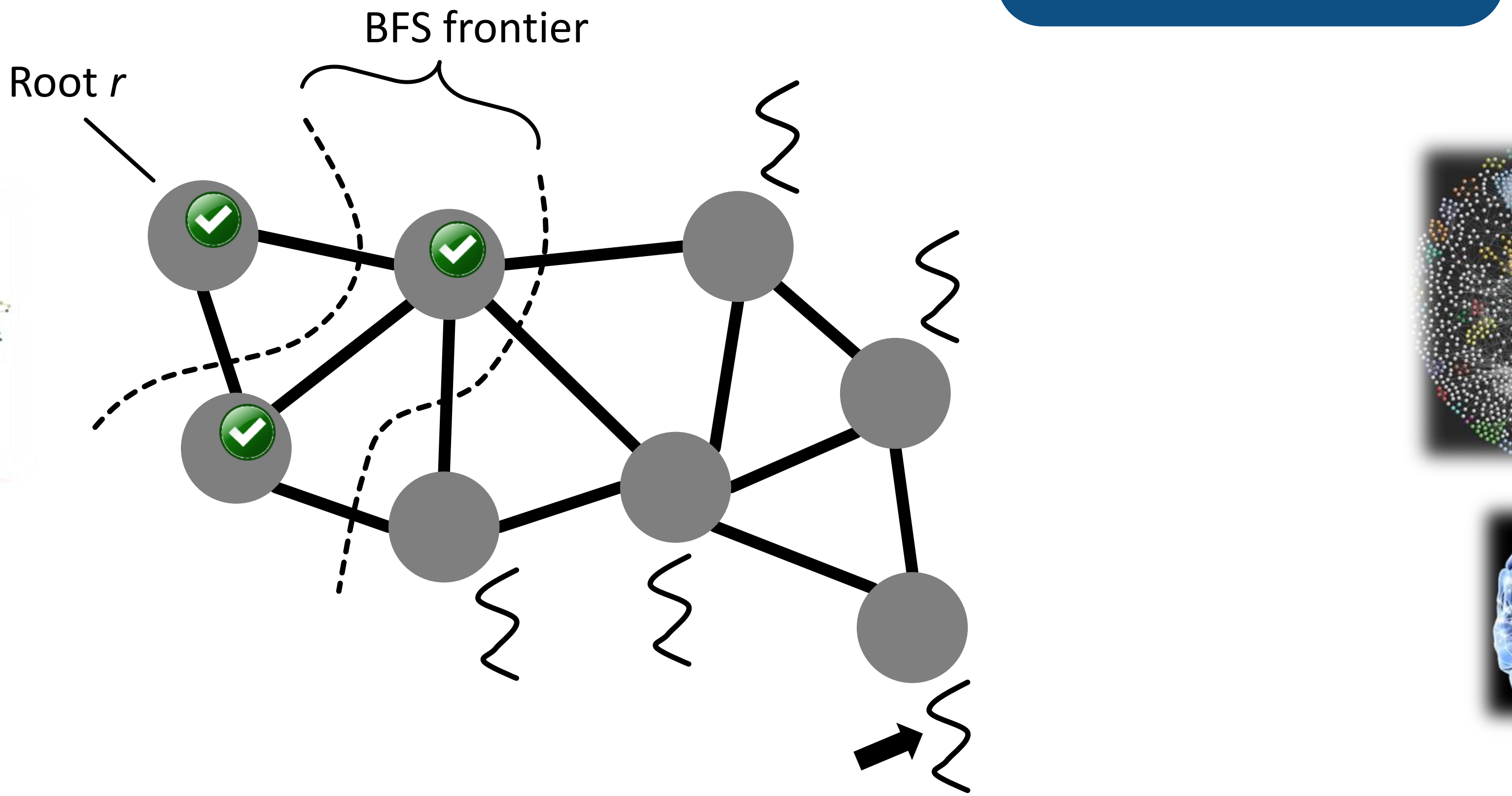
Pulling

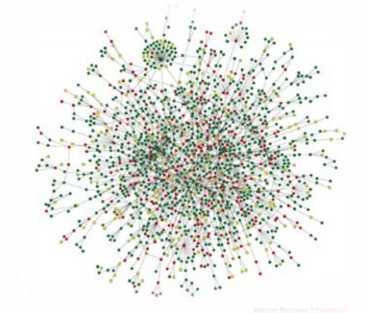
[1] S. Beamer, K. Asanović, and D. Patterson. Direction-optimizing breadth-first search. SC12.

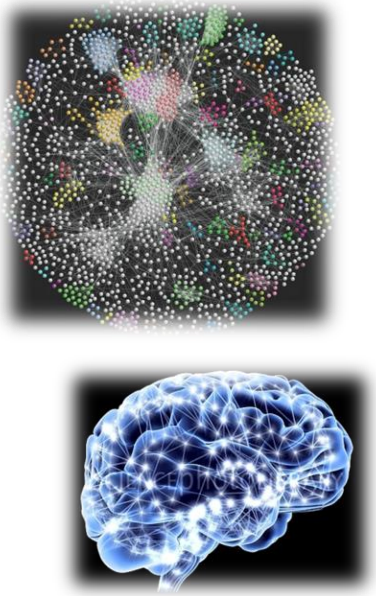
BFS
Top-Down vs. Bottom-Up [1]









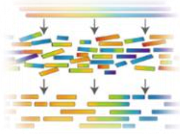


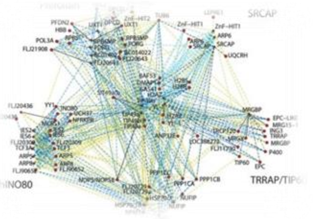
Pulling

[1] S. Beamer, K. Asanović, and D. Patterson. Direction-optimizing breadth-first search. SC12.

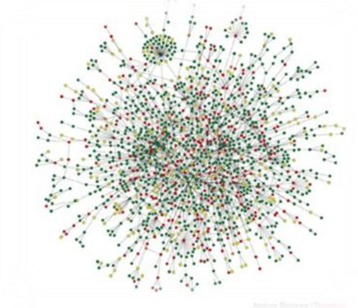
BFS

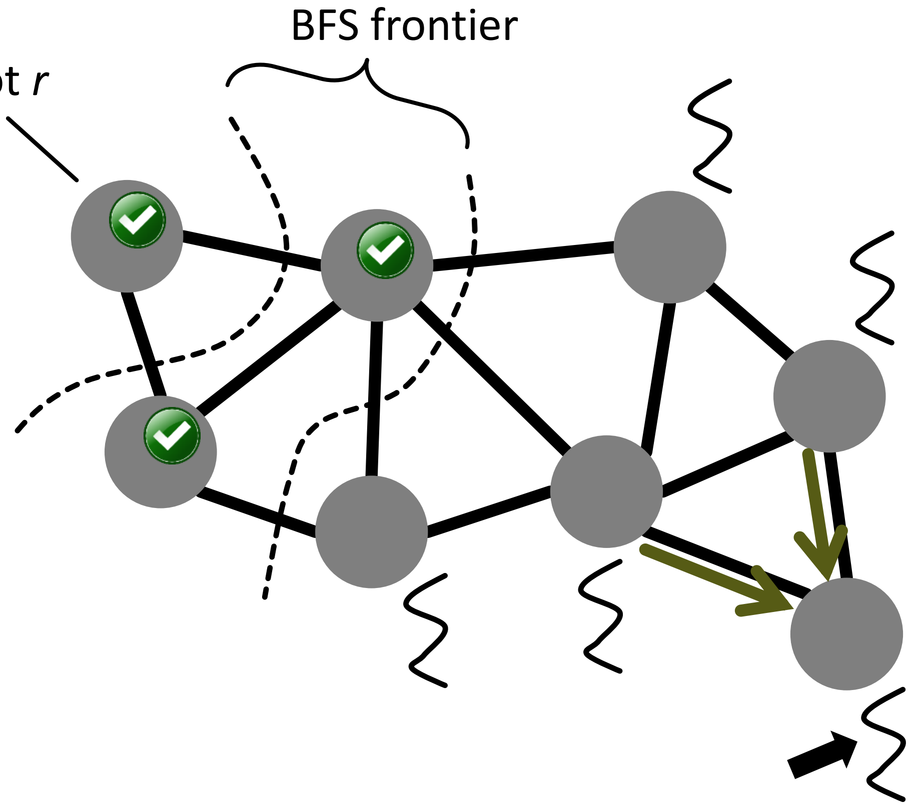
Top-Down vs. Bottom-Up [1]

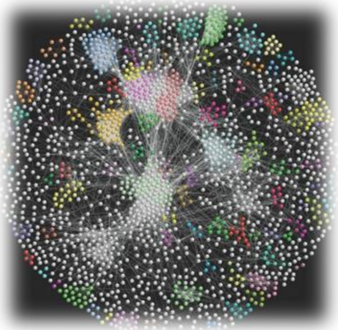




Pushing or pulling when expanding a frontier





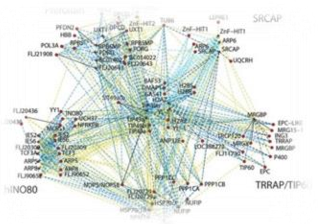


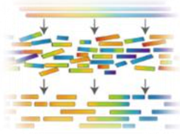
Pulling

[1] S. Beamer, K. Asanović, and D. Patterson. Direction-optimizing breadth-first search. SC12.

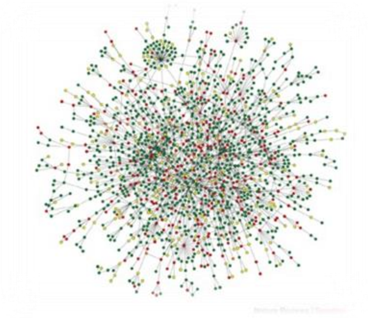
BFS

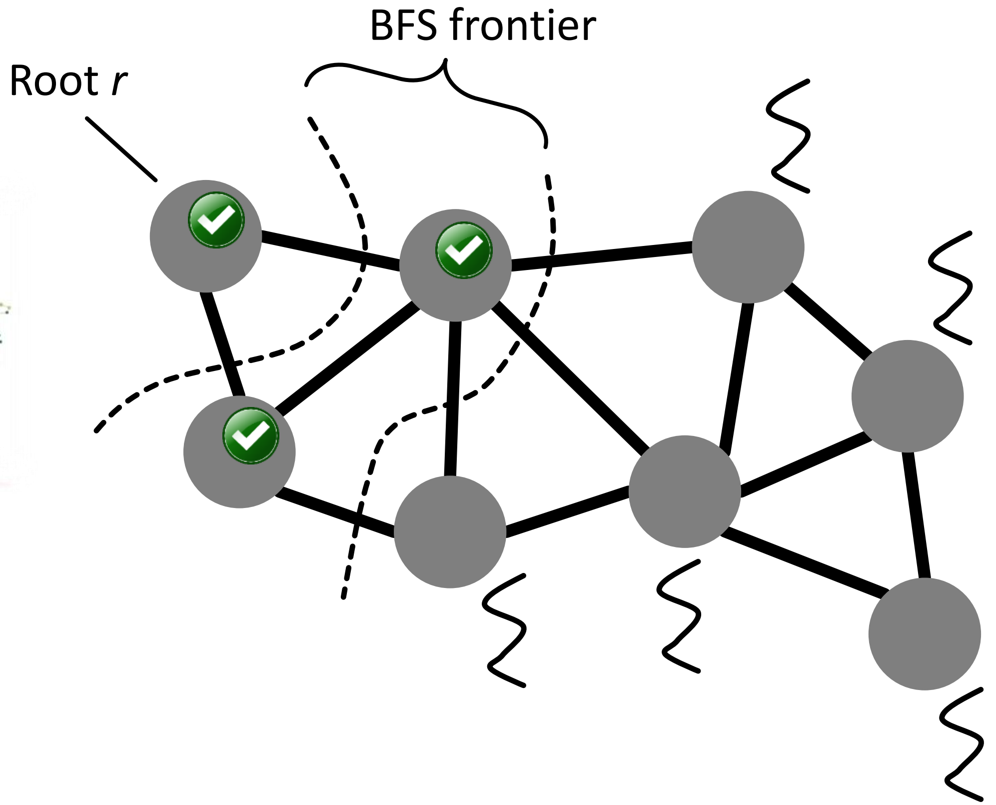
Top-Down vs. Bottom-Up [1]

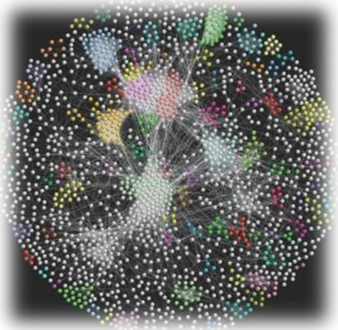




Pushing or pulling when expanding a frontier



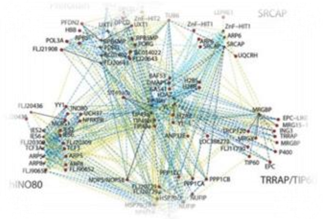


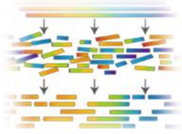


Pulling

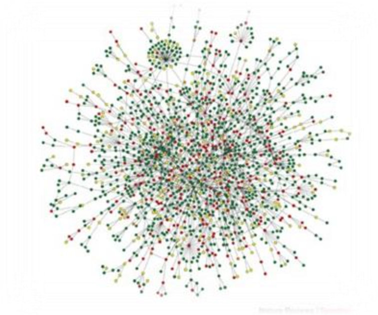
[1] S. Beamer, K. Asanović, and D. Patterson. Direction-optimizing breadth-first search. SC12.

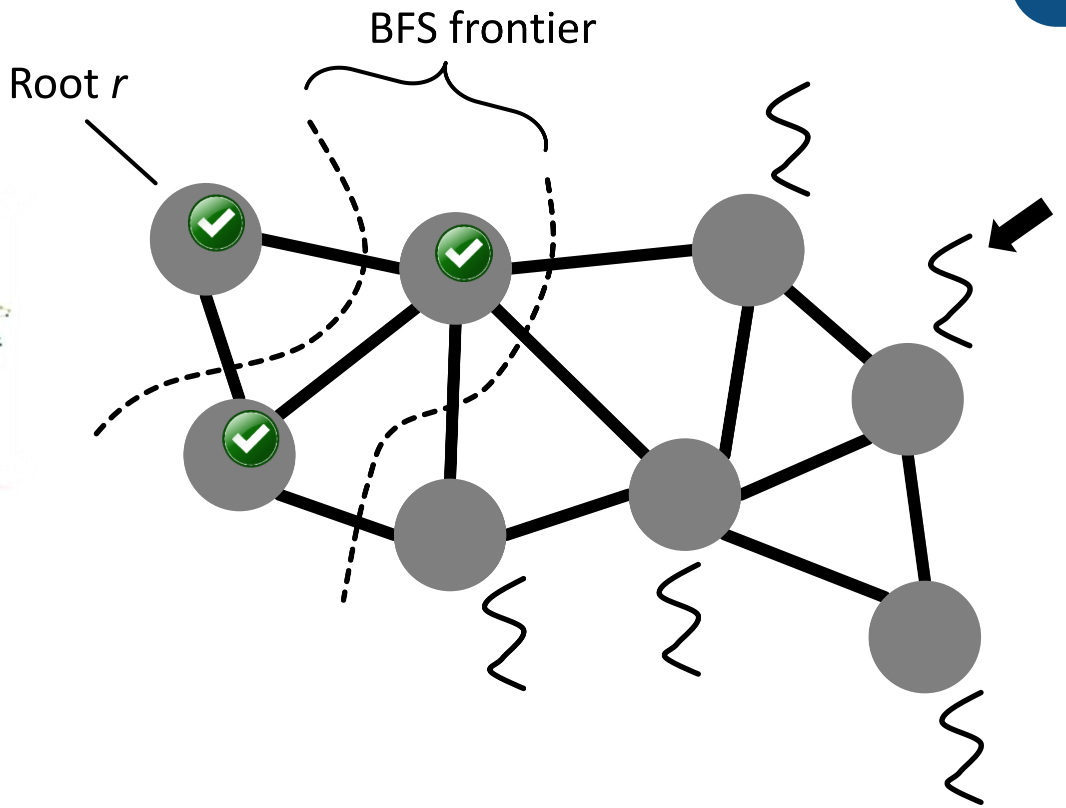
BFS
Top-Down vs. Bottom-Up [1]

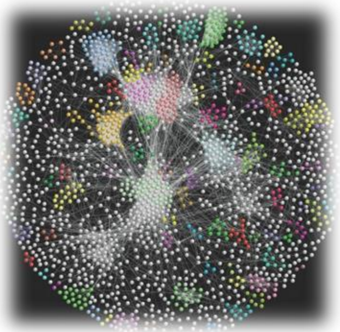




Pushing or pulling when expanding a frontier



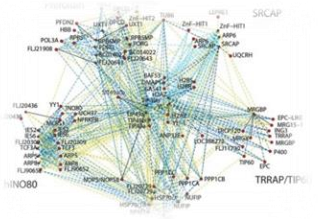


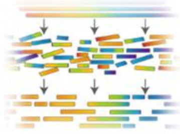


Pulling

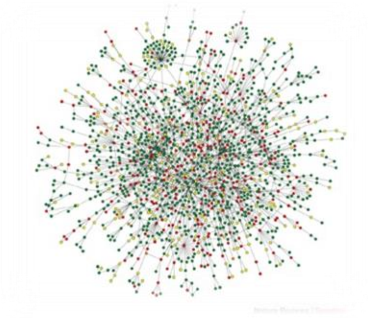
[1] S. Beamer, K. Asanović, and D. Patterson. Direction-optimizing breadth-first search. SC12.

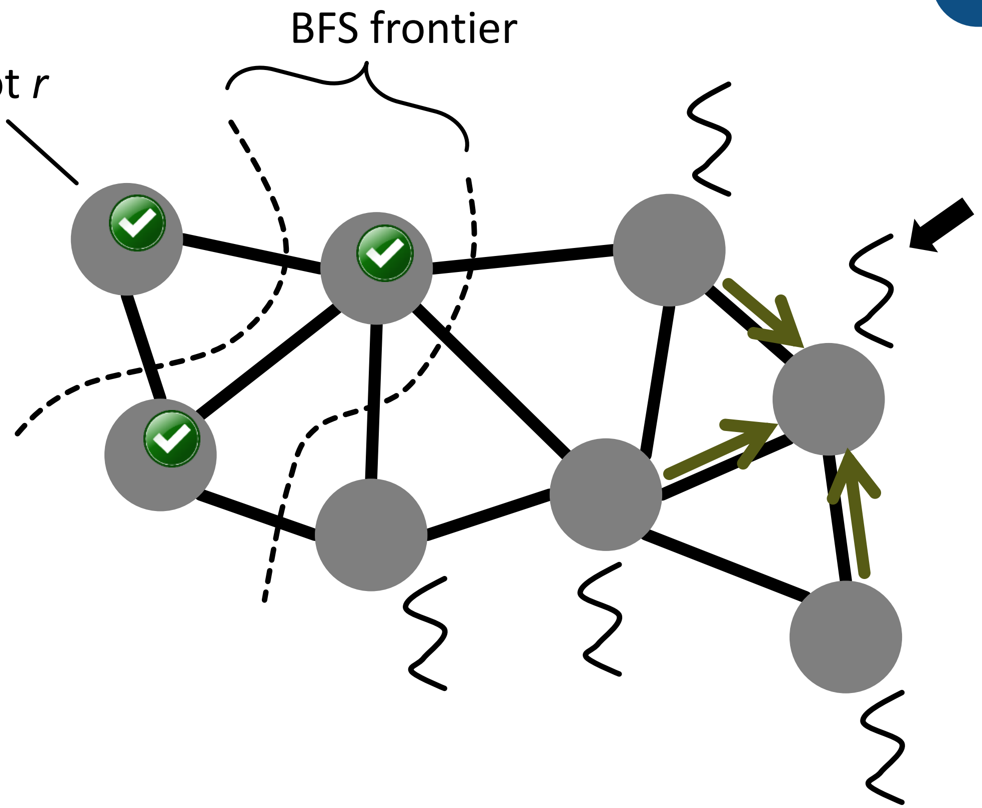
BFS
Top-Down vs. Bottom-Up [1]

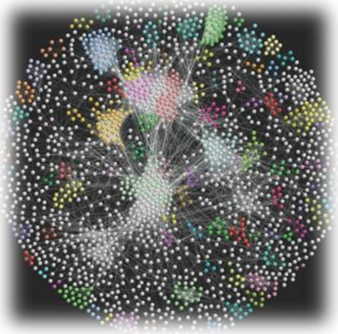




Pushing or pulling when expanding a frontier



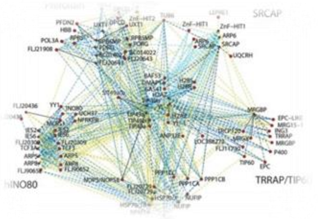


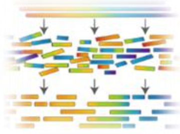


Pulling

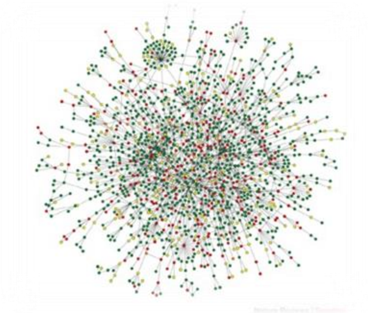
[1] S. Beamer, K. Asanović, and D. Patterson. Direction-optimizing breadth-first search. SC12.

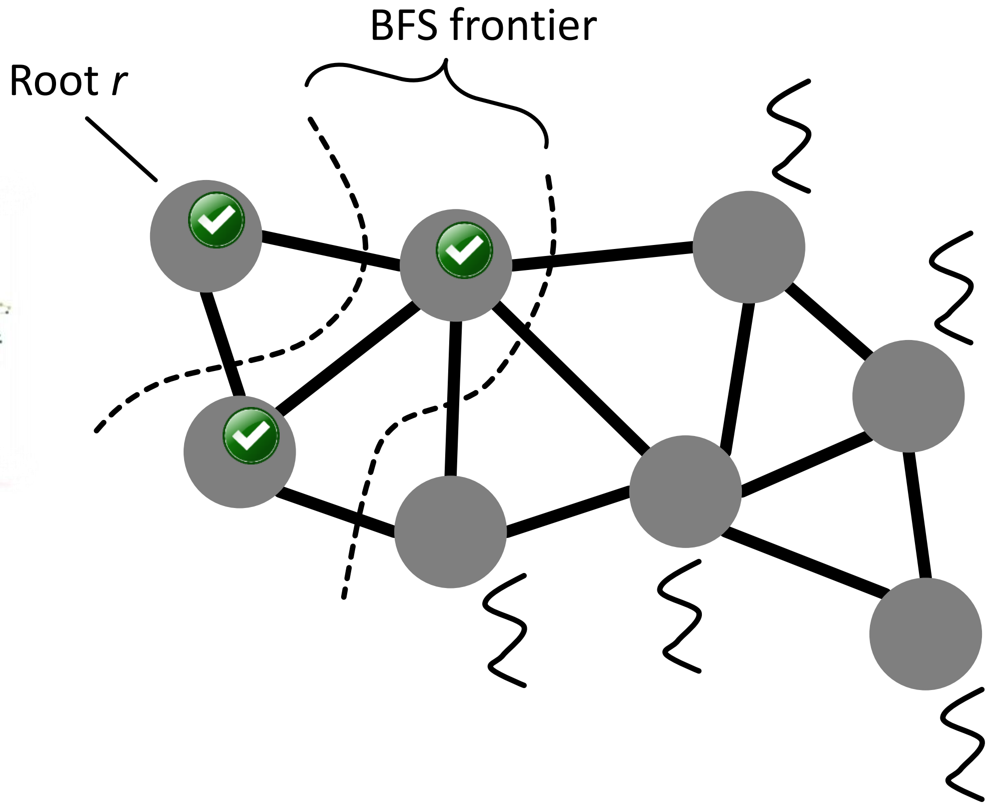
BFS
TOP-DOWN VS. BOTTOM-UP [1]

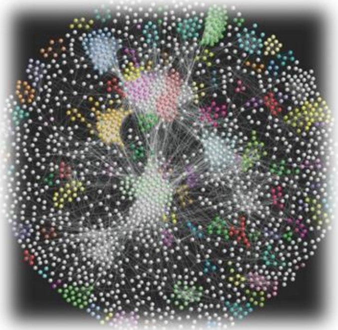




Pushing or pulling when expanding a frontier





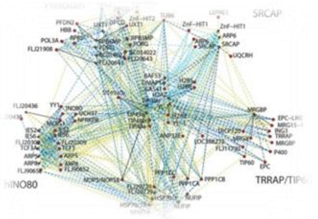


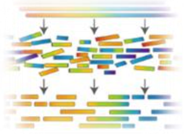
Pulling

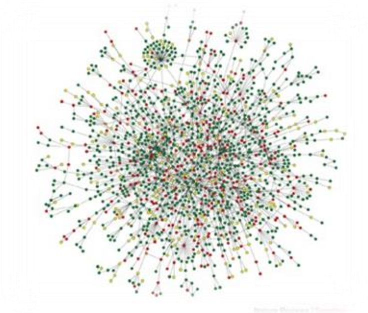
[1] S. Beamer, K. Asanović, and D. Patterson. Direction-optimizing breadth-first search. SC12.

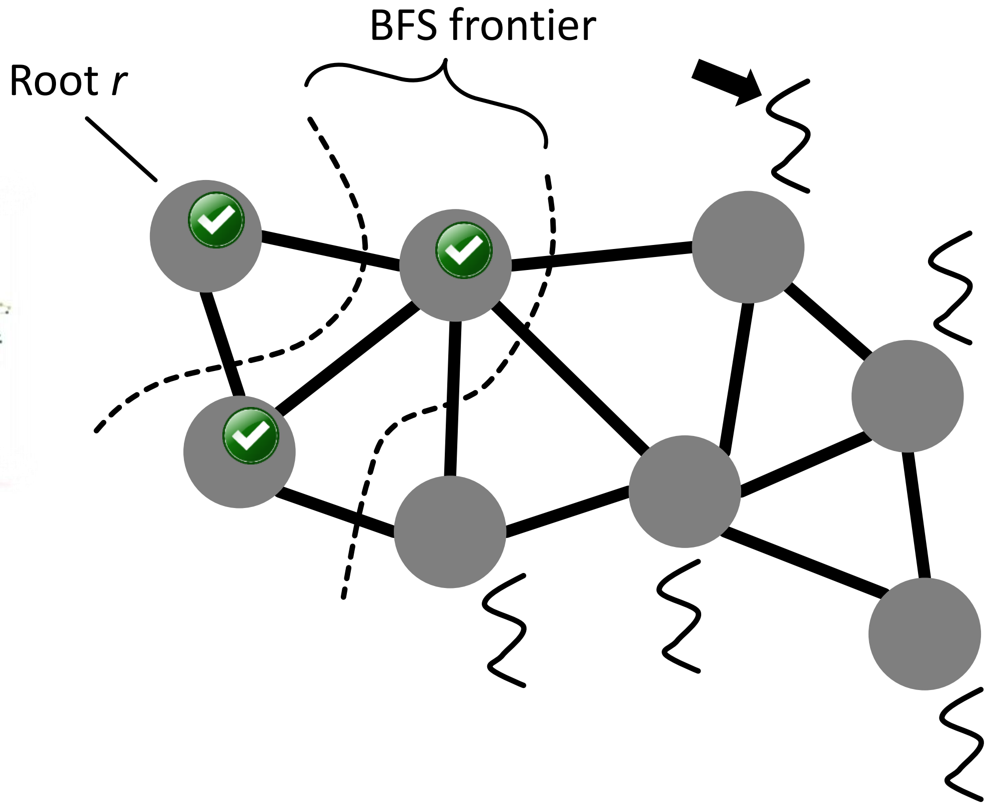
BFS

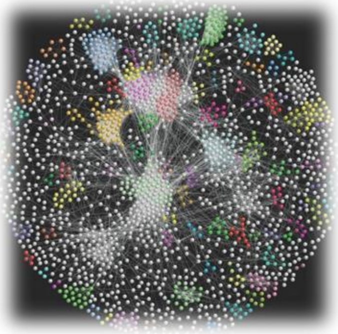
TOP-DOWN VS. BOTTOM-UP [1]









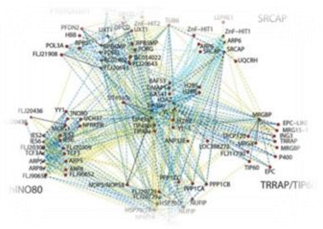


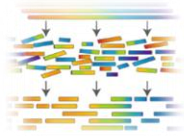
Pulling

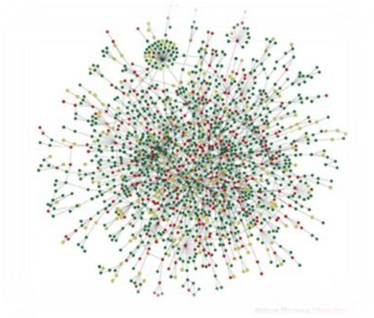
[1] S. Beamer, K. Asanović, and D. Patterson. Direction-optimizing breadth-first search. SC12.

BFS

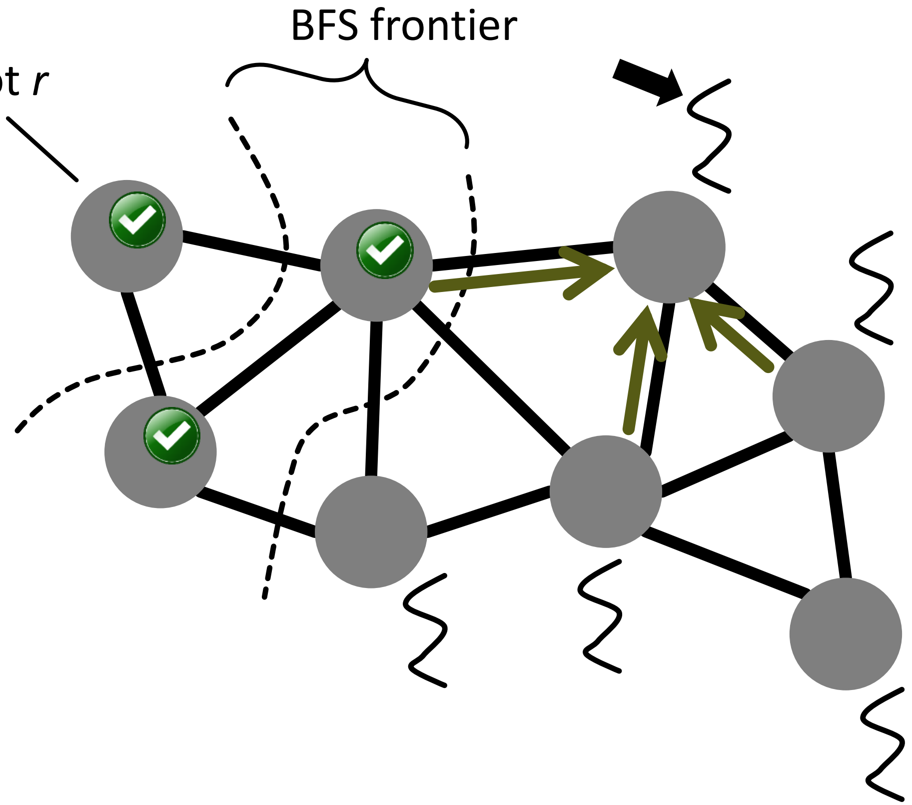
Top-Down vs. Bottom-Up [1]

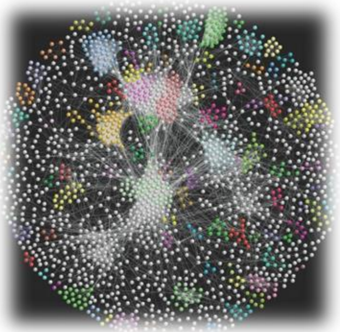






Pushing or pulling when expanding a frontier

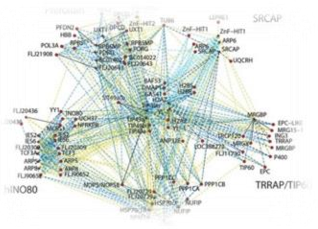


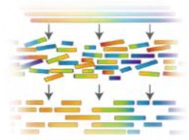


Pulling

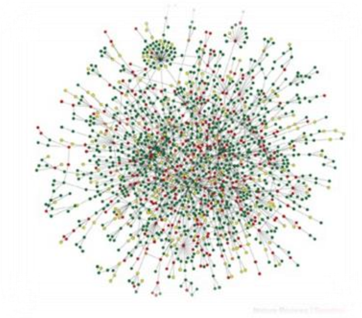
[1] S. Beamer, K. Asanović, and D. Patterson. Direction-optimizing breadth-first search. SC12.

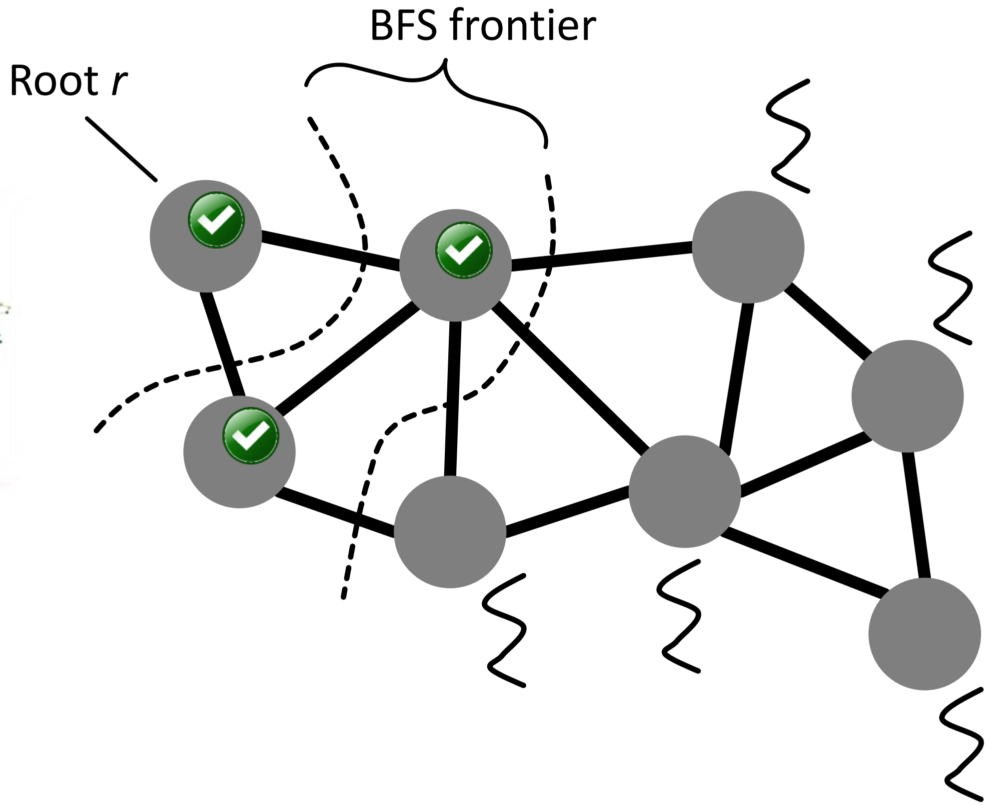
BFS
Top-Down vs. Bottom-Up [1]

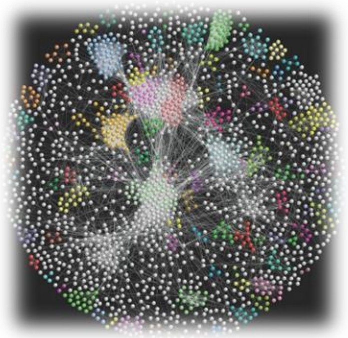




Pushing or pulling when expanding a frontier





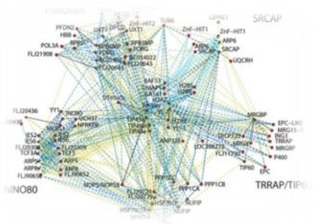


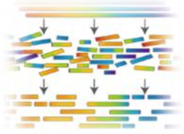
Pulling

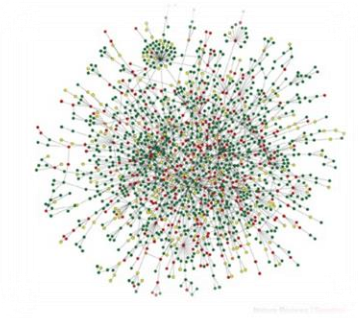
[1] S. Beamer, K. Asanović, and D. Patterson. Direction-optimizing breadth-first search. SC12.

BFS

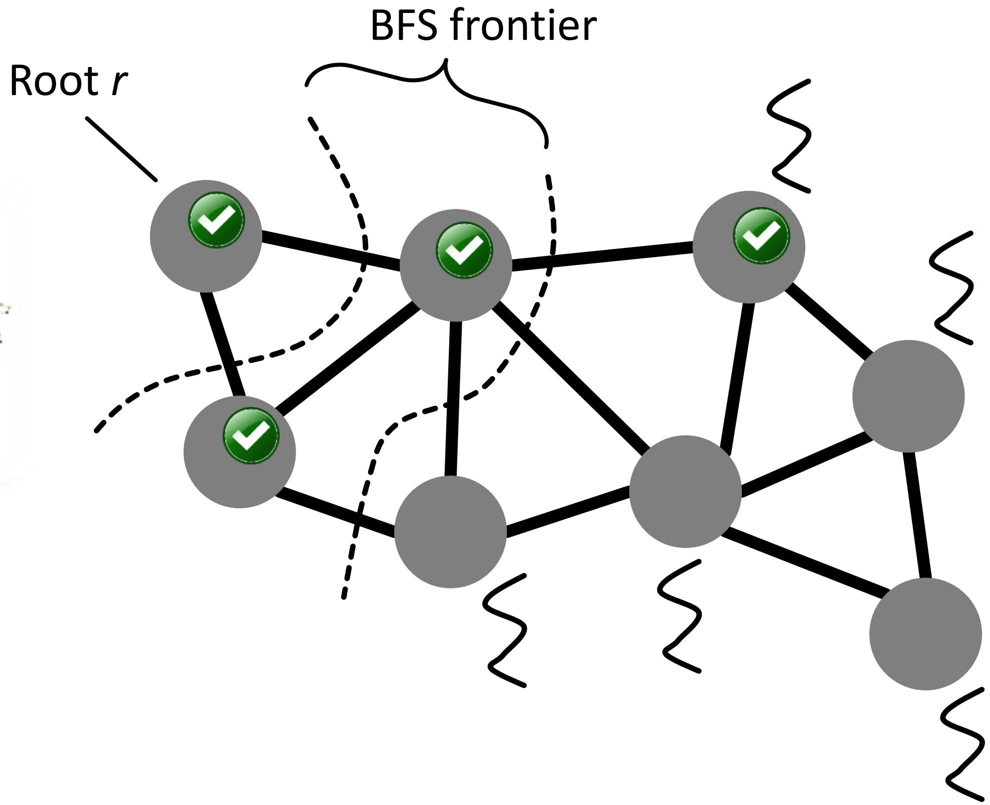
Top-Down vs. Bottom-Up [1]

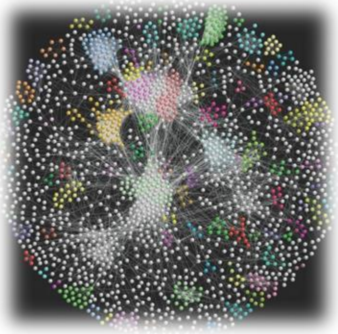






Pushing or pulling when expanding a frontier



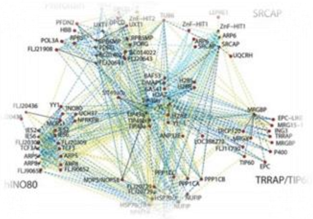


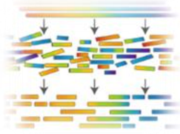
Pulling

[1] S. Beamer, K. Asanović, and D. Patterson. Direction-optimizing breadth-first search. SC12.

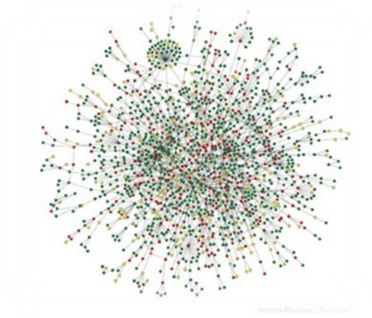
BFS

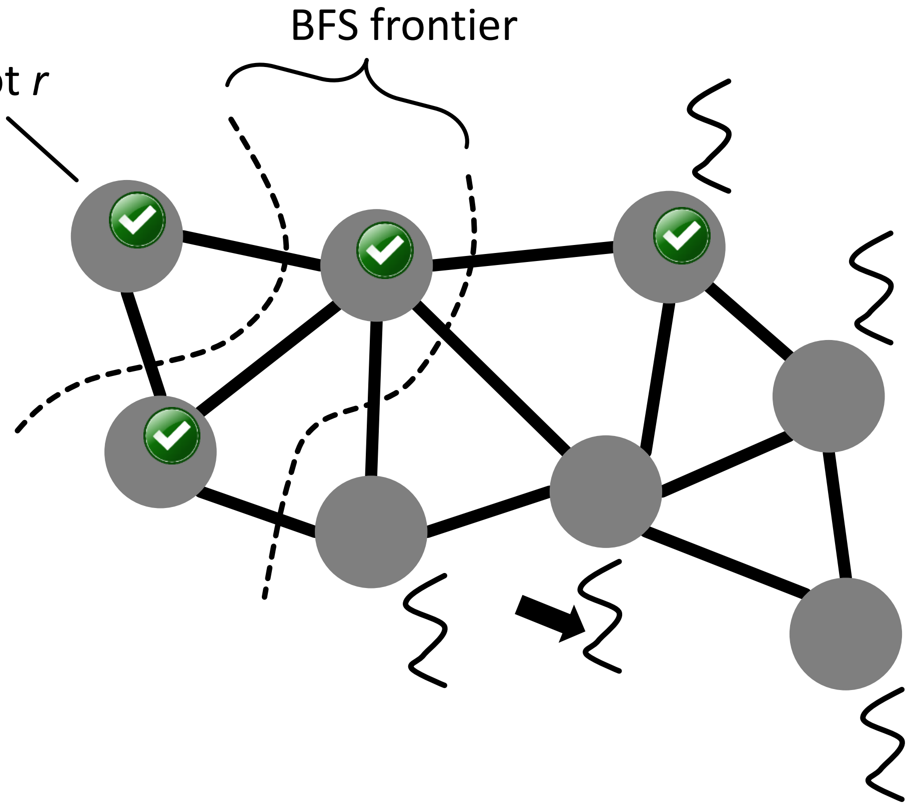
Top-Down vs. Bottom-Up [1]

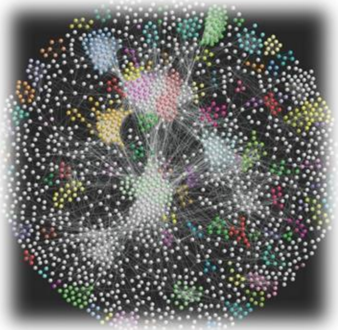




Pushing or pulling when expanding a frontier



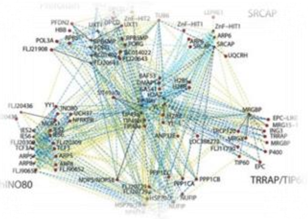


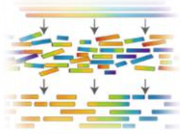


Pulling

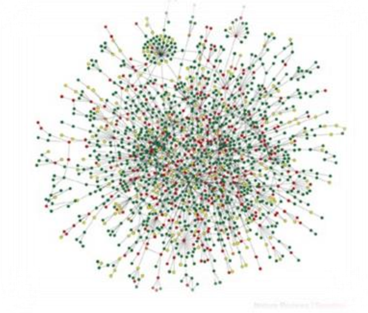
[1] S. Beamer, K. Asanović, and D. Patterson. Direction-optimizing breadth-first search. SC12.

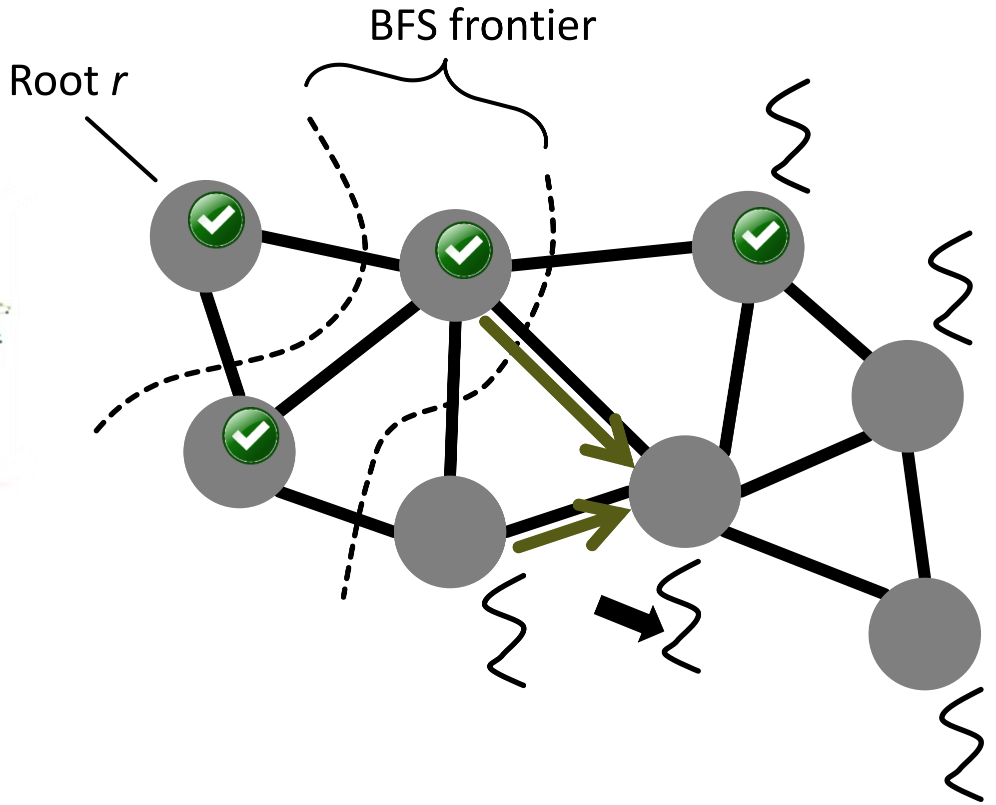
BFS
Top-Down vs. Bottom-Up [1]

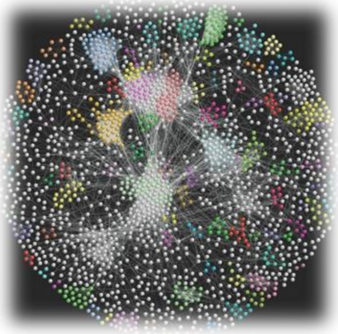




Pushing or pulling when expanding a frontier







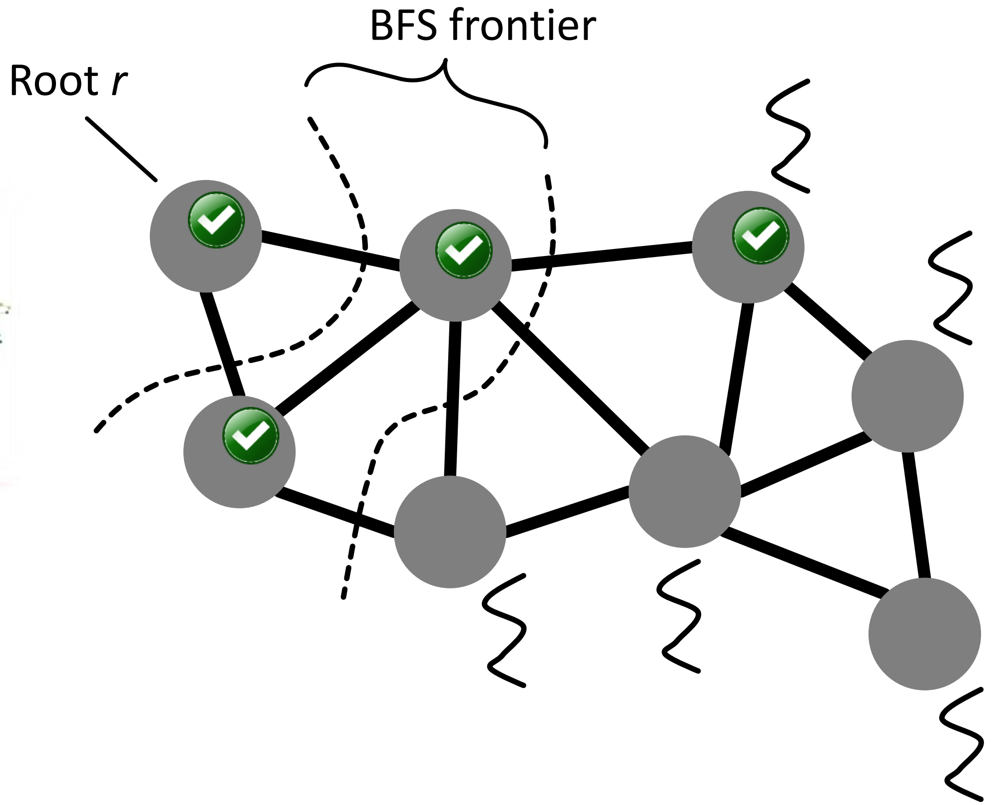
Pulling

[1] S. Beamer, K. Asanović, and D. Patterson. Direction-optimizing breadth-first search. SC12.

BFS

Top-Down vs. Bottom-Up [1]

Pushing or pulling when expanding a frontier

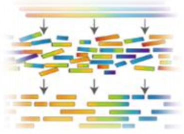


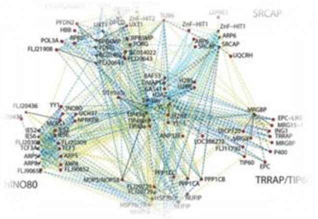
Pulling

[1] S. Beamer, K. Asanović, and D. Patterson. Direction-optimizing breadth-first search. SC12.

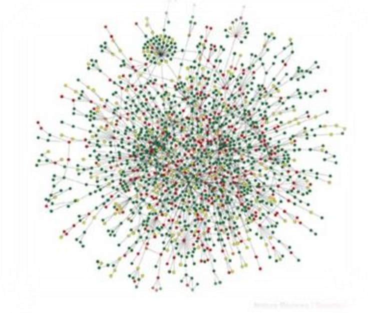
BFS

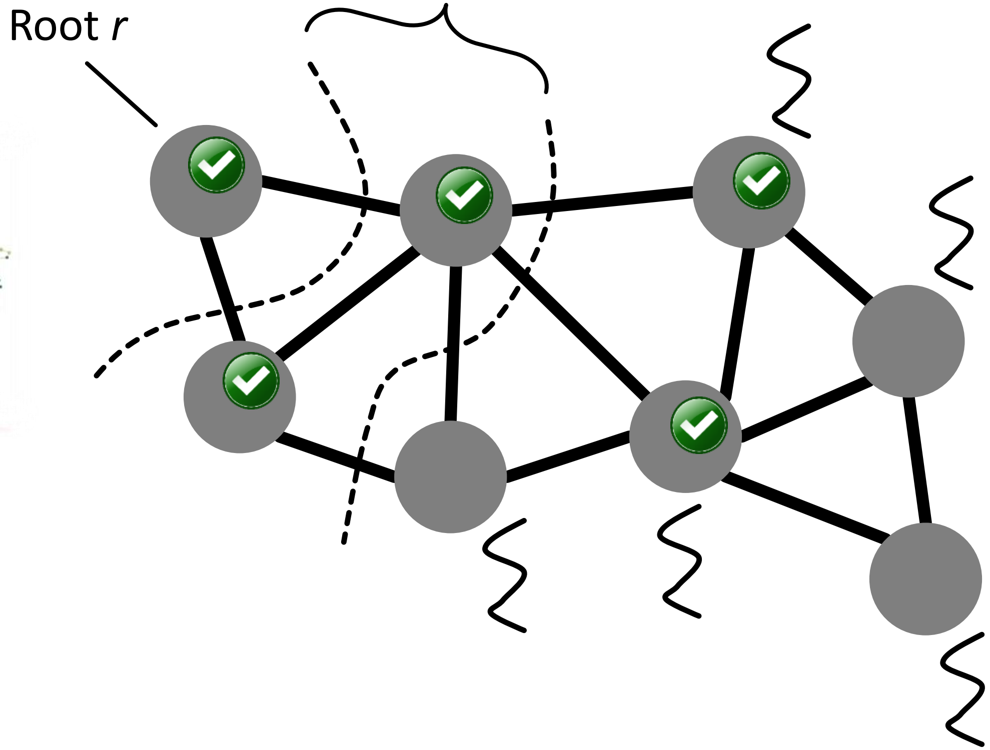
TOP-DOWN VS. BOTTOM-UP [1]

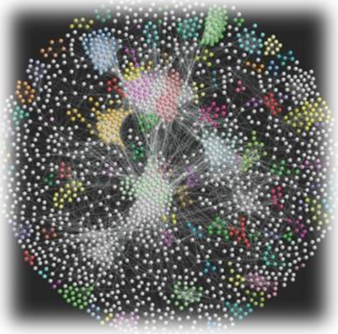




Pushing or pulling when expanding a frontier



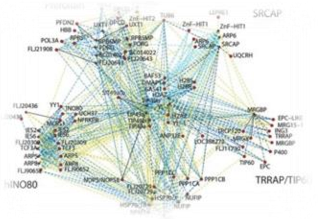


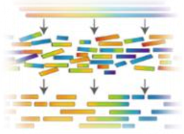


Pulling

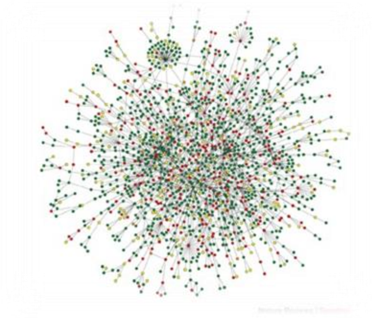
[1] S. Beamer, K. Asanović, and D. Patterson. Direction-optimizing breadth-first search. SC12.

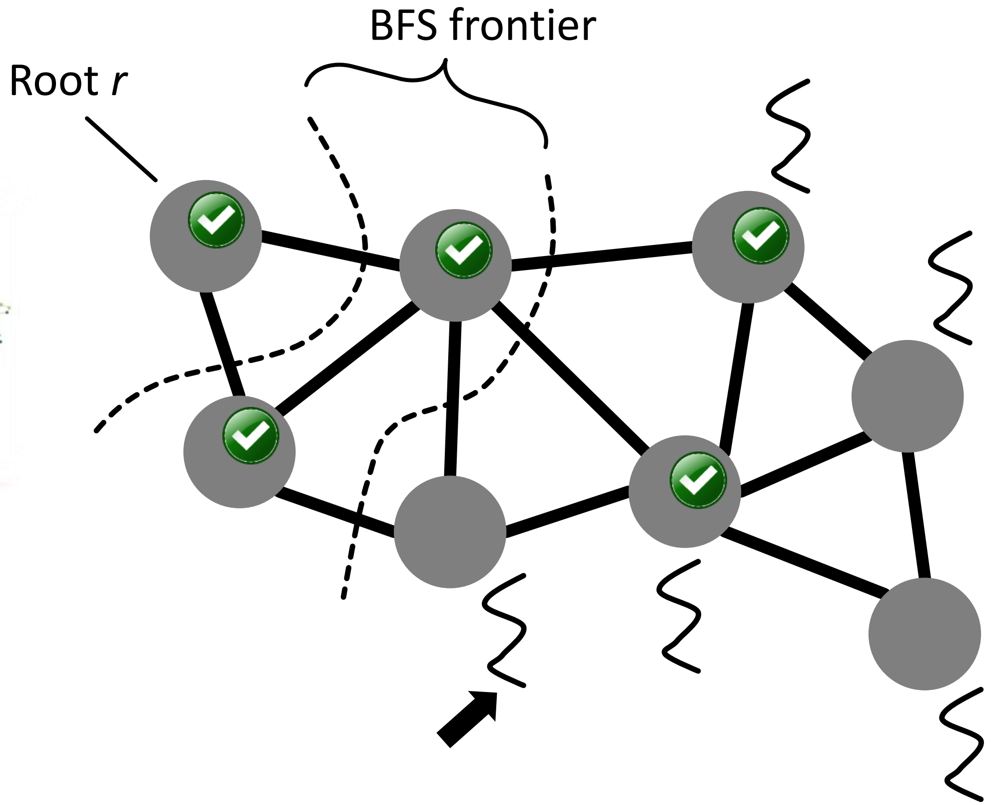
BFS
Top-Down vs. Bottom-Up [1]

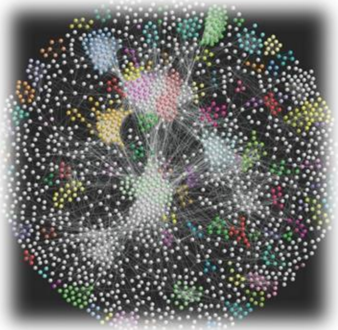




Pushing or pulling when expanding a frontier



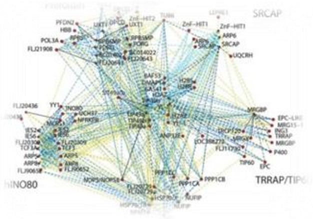


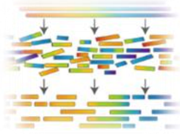


Pulling

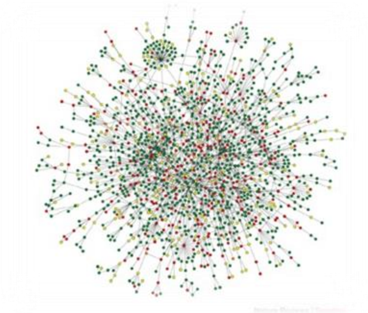
[1] S. Beamer, K. Asanović, and D. Patterson. Direction-optimizing breadth-first search. SC12.

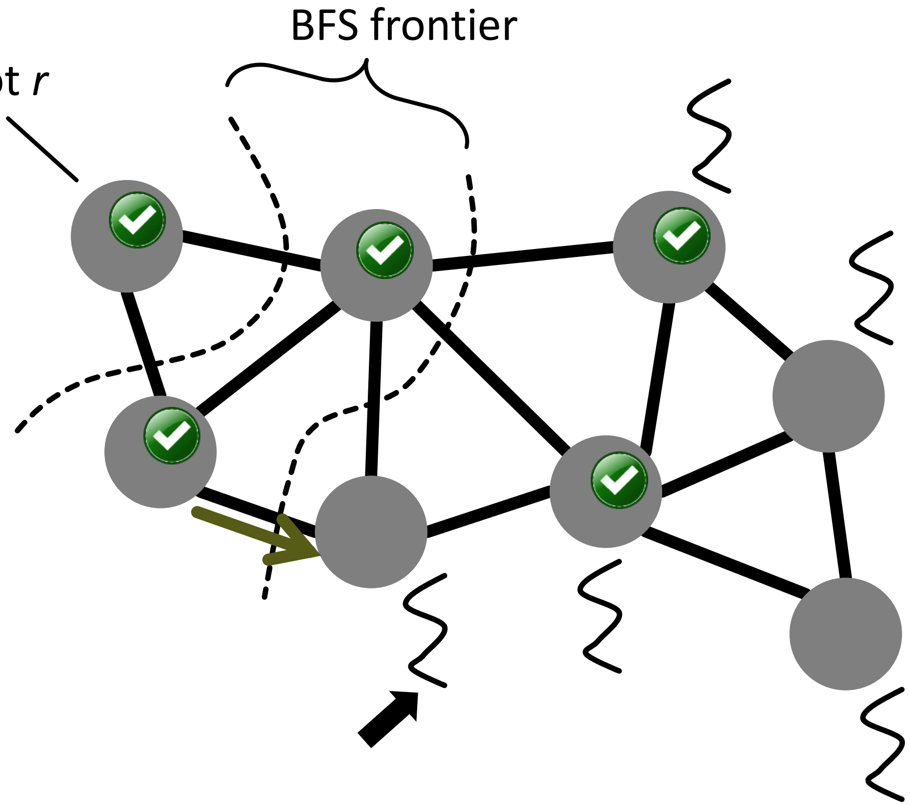
BFS
TOP-DOWN VS. BOTTOM-UP [1]

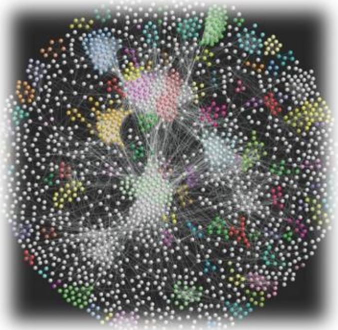




Pushing or pulling when expanding a frontier





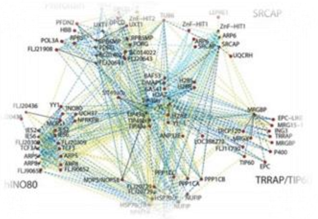


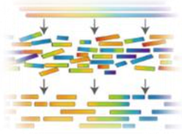
Pulling

[1] S. Beamer, K. Asanović, and D. Patterson. Direction-optimizing breadth-first search. SC12.

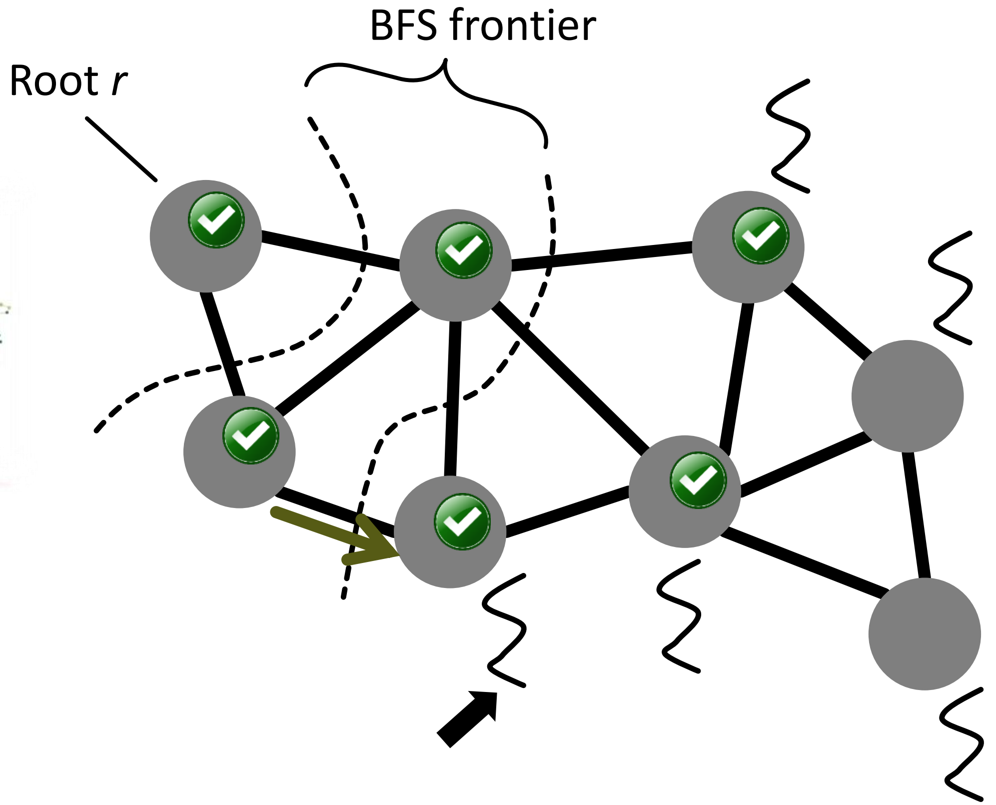
BFS

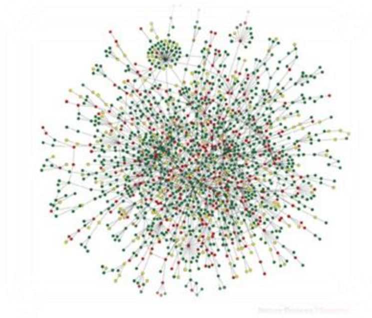
Top-Down vs. Bottom-Up [1]

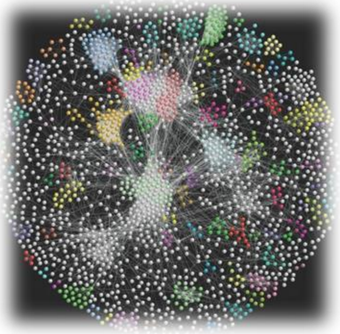




Pushing or pulling when expanding a frontier



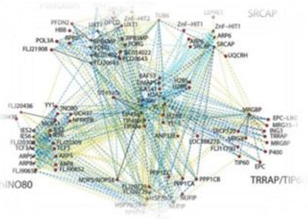


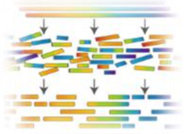


Pulling

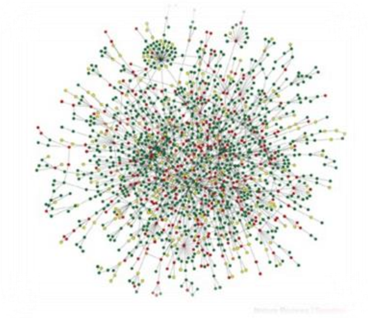
[1] S. Beamer, K. Asanović, and D. Patterson. Direction-optimizing breadth-first search. SC12.

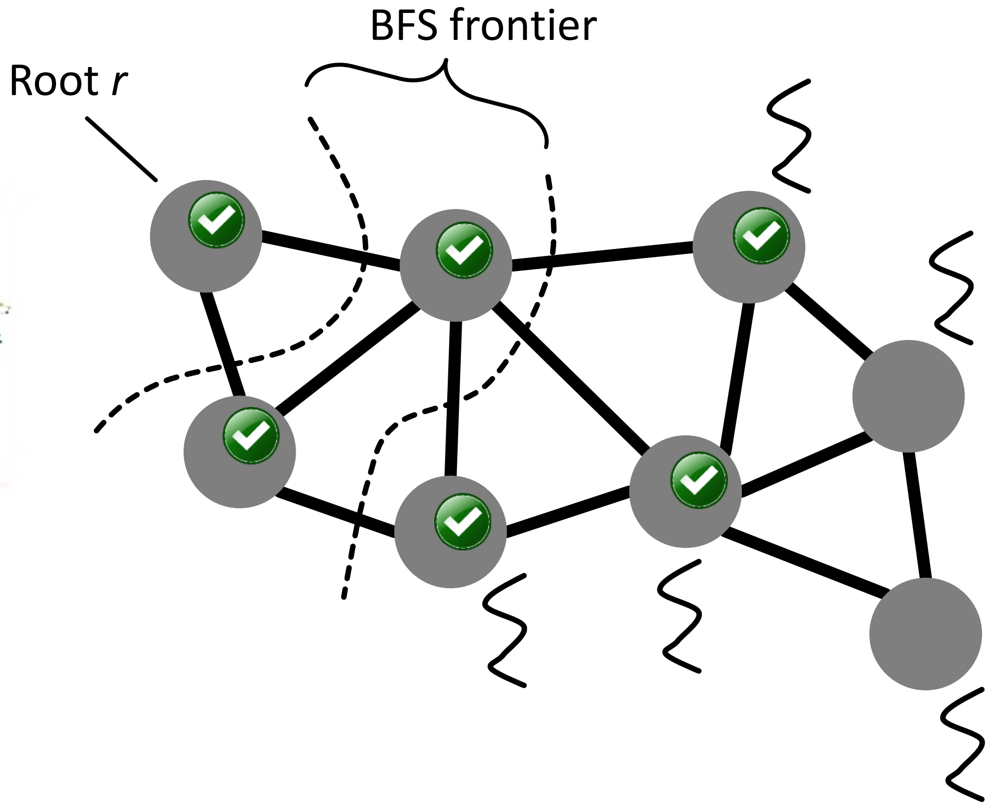
BFS
Top-Down vs. Bottom-Up [1]

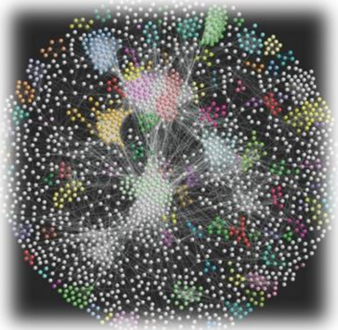




Pushing or pulling when expanding a frontier





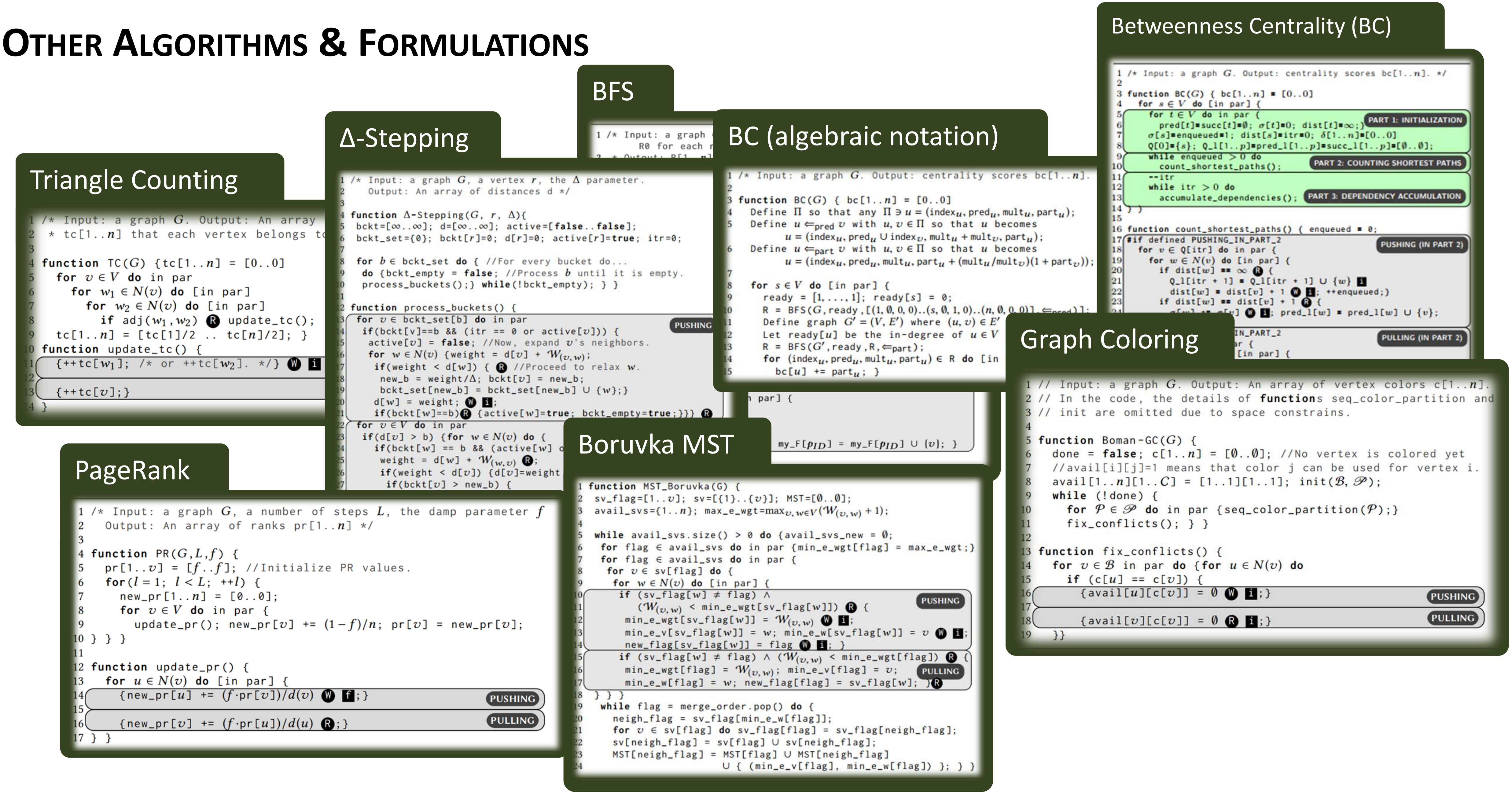


Pulling

[1] S. Beamer, K. Asanović, and D. Patterson. Direction-optimizing breadth-first search. SC12.

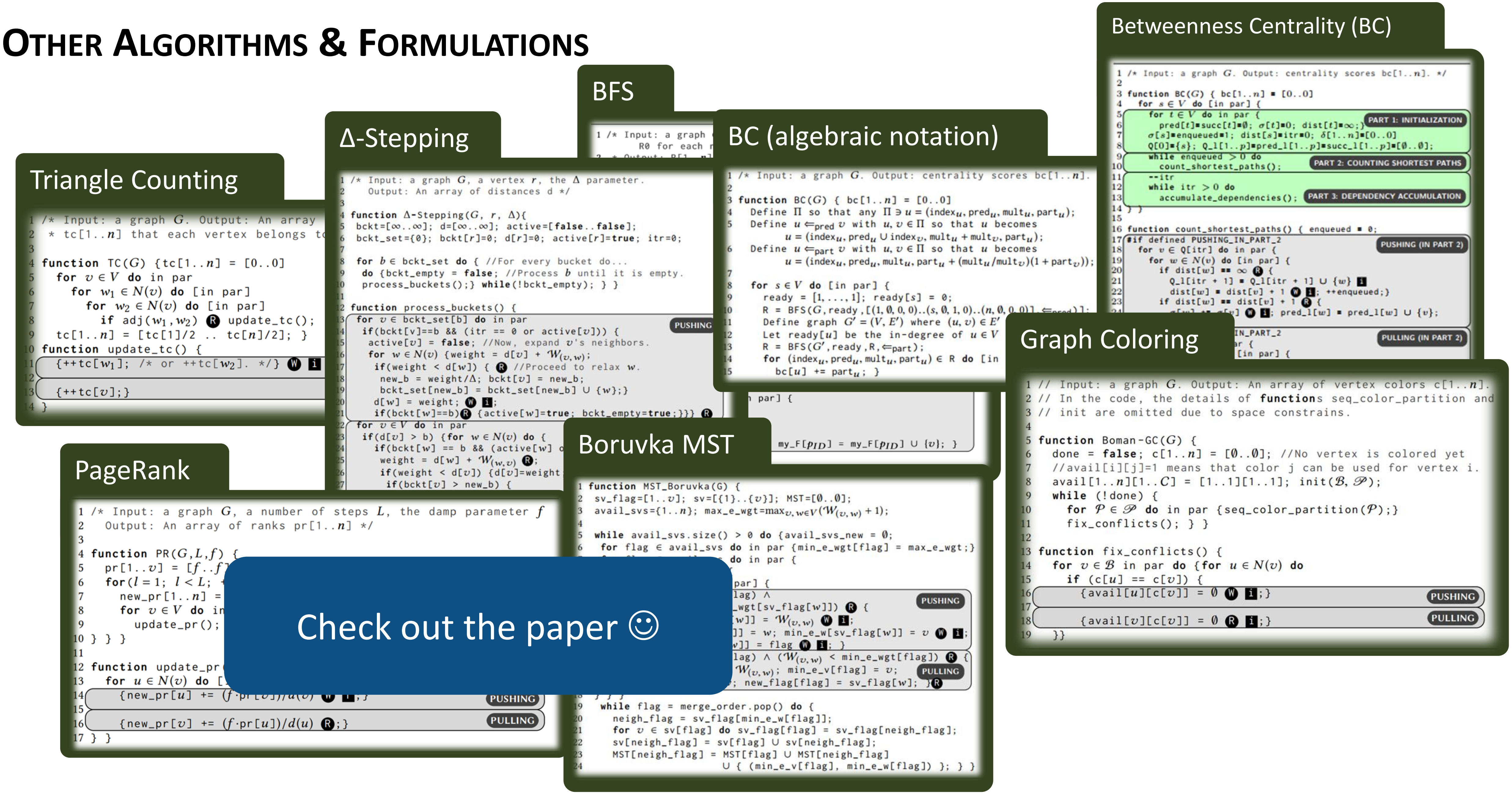
OTHER ALGORITHMS & FORMULATIONS

OTHER ALGORITHMS & FORMULATIONS



M. Besta et al.: “To Push or To Pull: On Reducing Communication and Synchronization in Graph Computations”, HPDC’17

Other Algorithms & Formulations



M. Besta et al.: "To Push or To Pull: On Reducing Communication and Synchronization in Graph Computations", HPDC'17

PUSHING VS. PULLING

GENERIC DIFFERENCES

M. Besta et al.: “To Push or To Pull: On Reducing Communication and Synchronization in Graph Computations”, HPDC’17

PUSHING VS. PULLING

GENERIC DIFFERENCES

What pushing vs. pulling *really* is?

M. Besta et al.: "To Push or To Pull: On Reducing Communication and Synchronization in Graph Computations", HPDC'17

Pushing vs. Pulling

Generic Differences

- Vertices: $v \in V$
- $t \leadsto v \Leftrightarrow t$ modifies v
- $t[v]$: a thread that owns v

What pushing vs. pulling *really* is?

M. Besta et al.: "To Push or To Pull: On Reducing Communication and Synchronization in Graph Computations", HPDC'17

Pushing vs. Pulling

Generic Differences

- Vertices: $v \in V$
- $t \leadsto v \Leftrightarrow t$ modifies v
- $t[v]$: a thread that owns v

What pushing vs. pulling *really* is?

Algorithm uses pushing ⇔

$$(\exists t\ \exists v \in V:\ \ t \leadsto v \land t \neq t[v])$$

M. Besta et al.: “To Push or To Pull: On Reducing Communication and Synchronization in Graph Computations”, HPDC’17

Pushing vs. Pulling
Generic Differences

- Vertices: $v \in V$
- $t \rightsquigarrow v \Leftrightarrow t$ modifies v
- $t[v]$: a thread that owns v

What pushing vs. pulling *really* is?

Algorithm uses pushing $\Leftrightarrow$

$$(\exists t\ \exists v \in V:\ \ t \rightsquigarrow v \wedge t \neq t[v])$$

Algorithm uses pulling $\Leftrightarrow$

$$(\forall t\ \forall v \in V:\ \ t \rightsquigarrow v \Rightarrow t = t[v])$$

M. Besta et al.: “To Push or To Pull: On Reducing Communication and Synchronization in Graph Computations”, HPDC’17

PUSHING VS. PULLING
GENERIC DIFFERENCES

- Vertices: $v \in V$
- $t \rightsquigarrow v \Leftrightarrow t$ modifies v
- $t[v]$: a thread that owns v

What pushing vs. pulling *really* is?

Algorithm uses pushing ⇔

$$(\exists t\ \exists v \in V:\ \ t \rightsquigarrow v \land t \neq t[v])$$

Algorithm uses pulling ⇔

$$(\forall t\ \forall v \in V:\ \ t \rightsquigarrow v \Rightarrow t = t[v])$$

This *is* the actual dichotomy

M. Besta et al.: "To Push or To Pull: On Reducing Communication and Synchronization in Graph Computations", HPDC'17

Pushing vs. Pulling
Generic Differences

- Vertices: $v \in V$
- $t \leadsto v \Leftrightarrow t$ modifies v
- $t[v]$: a thread that owns v

What pushing vs. pulling *really* is?

Algorithm uses pushing ⇔

$$\sim \left[(\exists t\, \exists v \in V\colon\ t \leadsto v \land t \neq t[v]) \right]$$

This *is* the actual dichotomy

Algorithm uses pulling ⇔

$$(\forall t\, \forall v \in V\colon\ t \leadsto v \Rightarrow t = t[v])$$

M. Besta et al.: “To Push or To Pull: On Reducing Communication and Synchronization in Graph Computations”, HPDC’17

Pushing vs. Pulling
Generic Differences

- Vertices: $v \in V$
- $t \rightsquigarrow v \Leftrightarrow t$ modifies v
- $t[v]$: a thread that owns v

What pushing vs. pulling *really* is?

Algorithm uses pushing ⇔

$$\sim\left[(\exists t\ \exists v \in V:\ \ t \rightsquigarrow v \wedge t \neq t[v])\right]$$

$$||$$

Algorithm uses pulling ⇔

$$(\forall t\ \forall v \in V:\ \ t \rightsquigarrow v \Rightarrow t = t[v])$$

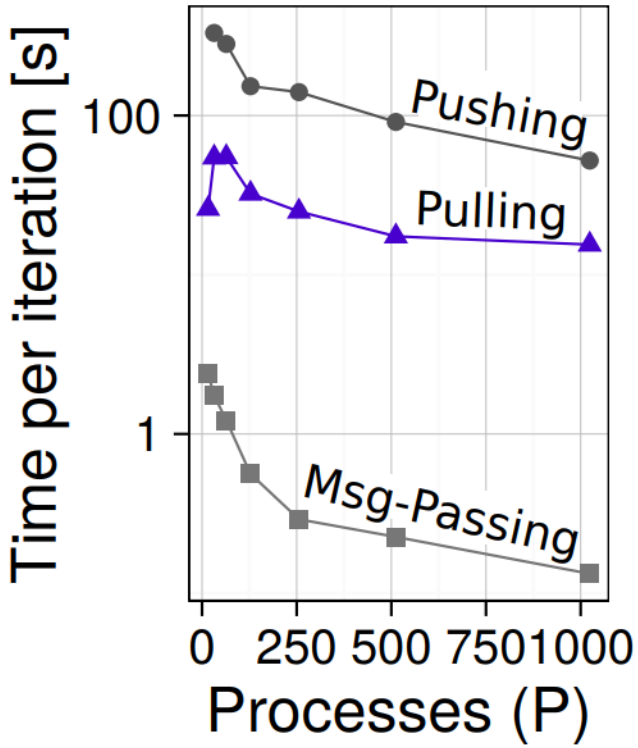
This *is* the actual dichotomy

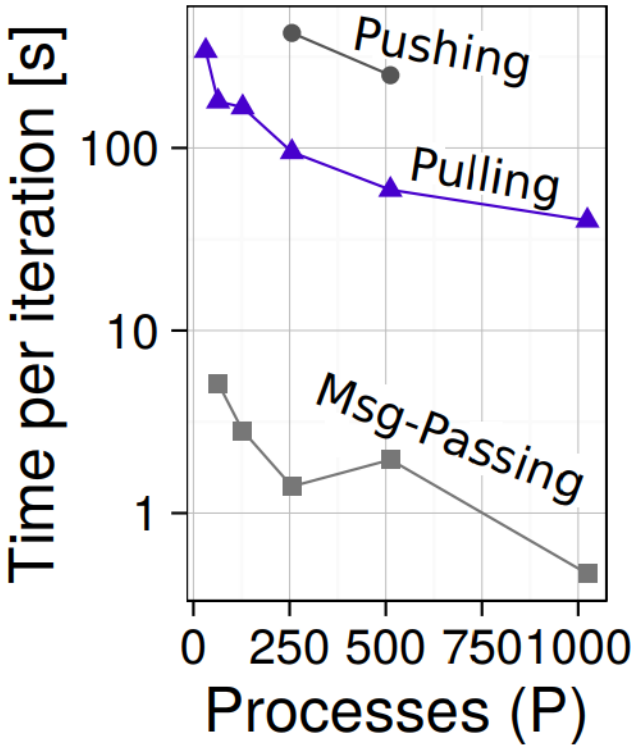
M. Besta et al.: "To Push or To Pull: On Reducing Communication and Synchronization in Graph Computations", HPDC'17

PERFORMANCE ANALYSIS
PAGERANK

Kronecker graphs

Distributed-Memory





M. Besta et al.: “To Push or To Pull: On Reducing Communication and Synchronization in Graph Computations”, HPDC’17

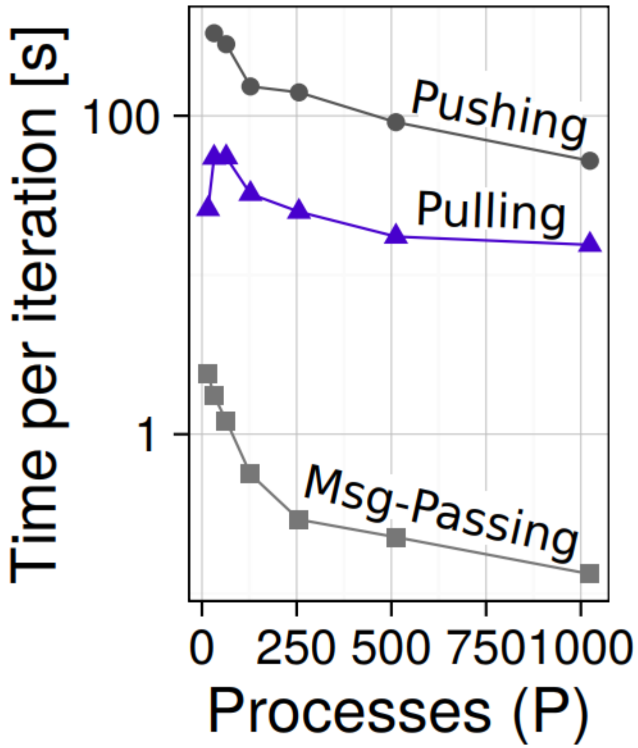
Performance Analysis

PageRank

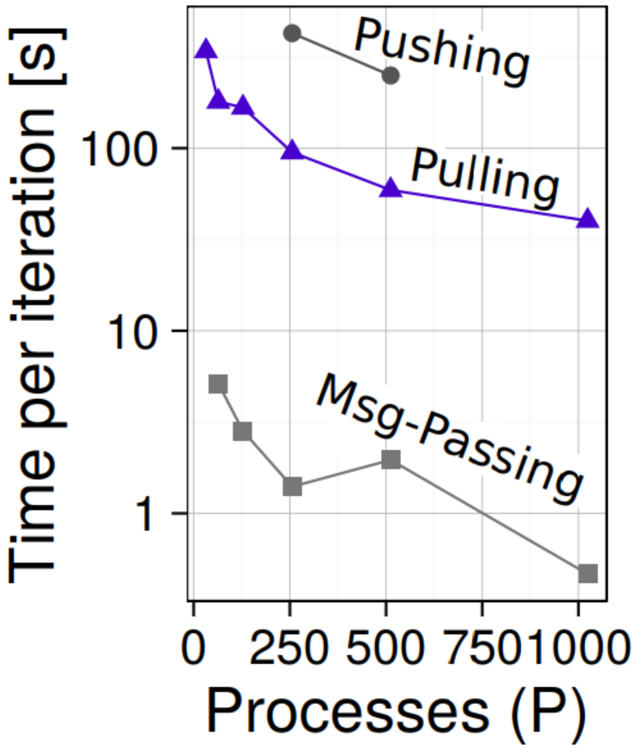
Kronecker graphs

Distributed-Memory

$n = 2^{25}, m = 2^{27}$



$n = 2^{27}, m = 2^{29}$



M. Besta et al.: “To Push or To Pull: On Reducing Communication and Synchronization in Graph Computations”, HPDC’17

Performance Analysis

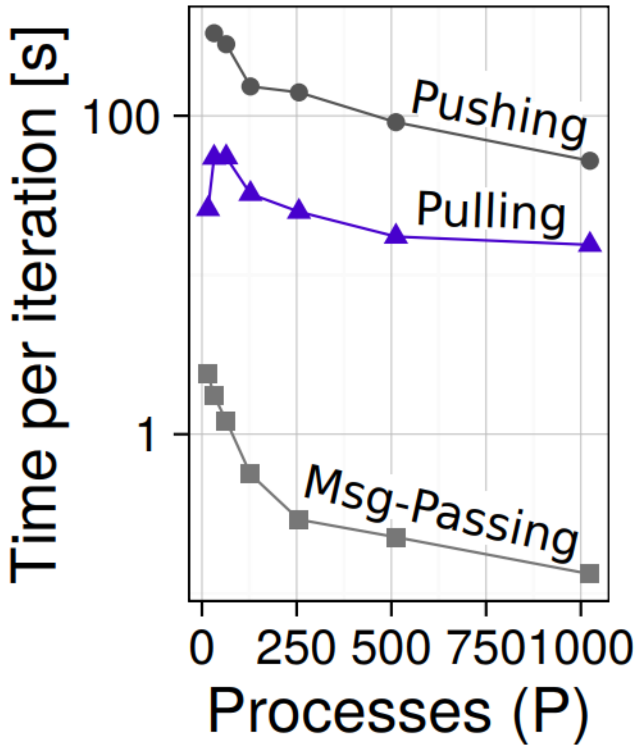
PageRank

Kronecker graphs

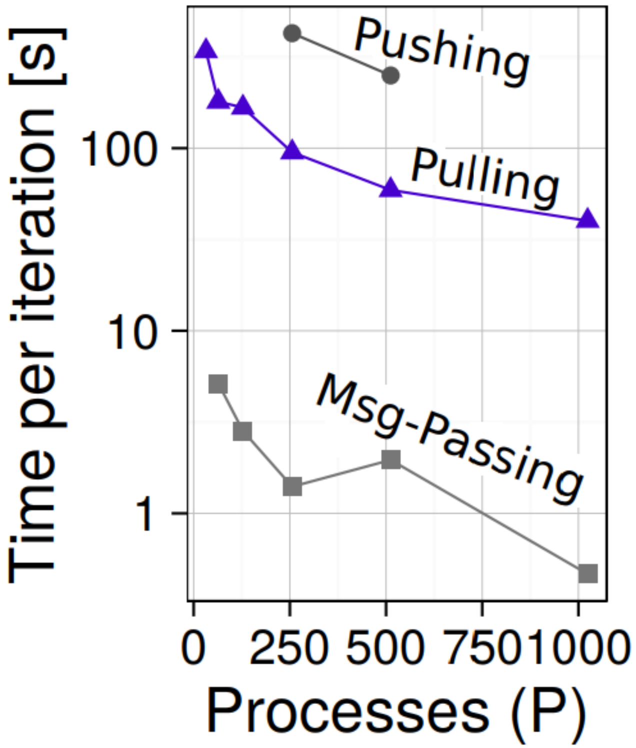
Distributed-Memory

Msg-Passing fastest

$n = 2^{25}, m = 2^{27}$



$n = 2^{27}, m = 2^{29}$



M. Besta et al.: "To Push or To Pull: On Reducing Communication and Synchronization in Graph Computations", HPDC'17

Performance Analysis
PageRank

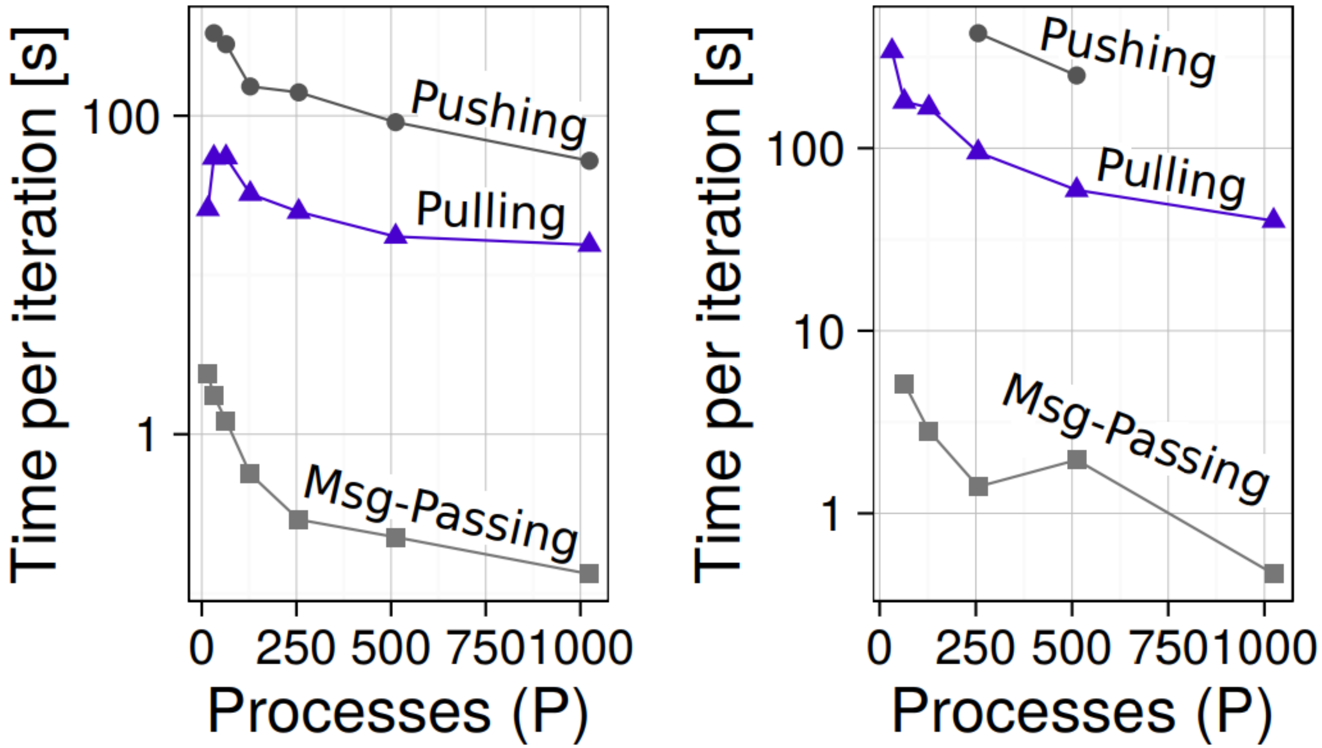
Kronecker graphs

Distributed-Memory

Msg-Passing fastest

Pulling incurs more communication while pushing expensive underlying locking

$n = 2^{25}, m = 2^{27}$ $\quad$ $n = 2^{27}, m = 2^{29}$



M. Besta et al.: "To Push or To Pull: On Reducing Communication and Synchronization in Graph Computations", HPDC'17

Performance Analysis
PageRank

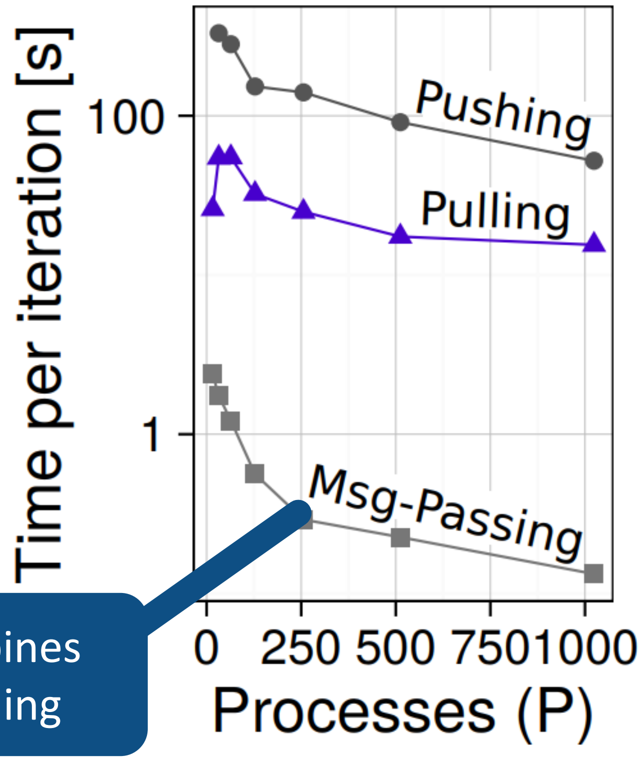
Kronecker graphs

Distributed-Memory

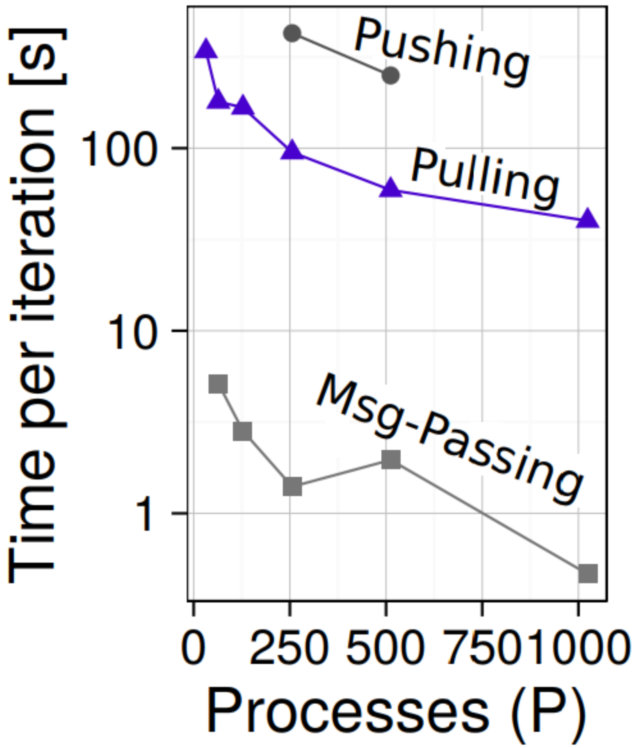
! Msg-Passing fastest

Pulling incurs more communication while pushing expensive underlying locking

$n = 2^{25}, m = 2^{27}$



$n = 2^{27}, m = 2^{29}$



! Collectives: combines pushing and pulling

M. Besta et al.: "To Push or To Pull: On Reducing Communication and Synchronization in Graph Computations", HPDC'17

To Push or To Pull?

M. Besta et al.: "To Push or To Pull: On Reducing Communication and Synchronization in Graph Computations", HPDC'17

To Push or To Pull?

If the complexities match: pull

M. Besta et al.: "To Push or To Pull: On Reducing Communication and Synchronization in Graph Computations", HPDC'17

To Push or To Pull?

If the complexities match: pull

Otherwise: push

M. Besta et al.: “To Push or To Pull: On Reducing Communication and Synchronization in Graph Computations”, HPDC’17

To Push or To Pull?

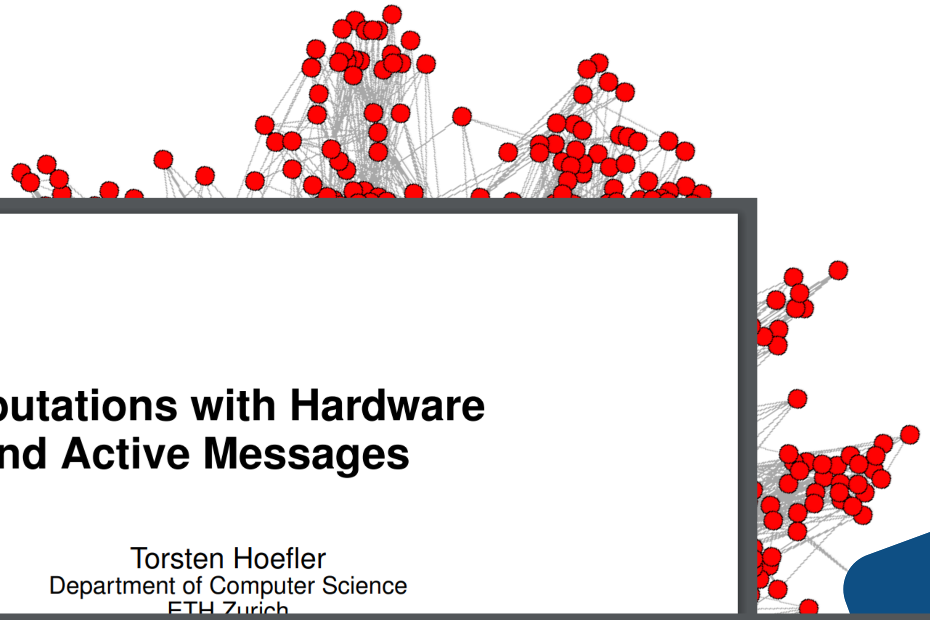
If the complexities match: pull

Otherwise: push

+ check your hardware ☺

M. Besta et al.: "To Push or To Pull: On Reducing Communication and Synchronization in Graph Computations", HPDC'17

Moving on …



Accelerating Irregular Computations with Hardware Transactional Memory and Active Messages

Maciej Besta
Department of Computer Science
ETH Zurich

Torsten Hoefler
Department of Computer Science
ETH Zurich

…chronization-heavy

To Push or To…
Synchr…

Maciej Besta[1], Michał …
[1] Department of Computer Science…
[4] Departm…
maciej.besta@inf.ethz.ch, michal.p…

ABSTRACT

We reduce the cost of communication an…
processing by analyzing the fastest way t…
the updates to a shared state or pulling…

High-Performance Distributed RMA Locks

Patrick Schmid*
Department of Computer Science
ETH Zurich
patrick.schmid@ieffects.com

Maciej Besta*
Department of Computer Science
ETH Zurich
bestam@inf.ethz.ch

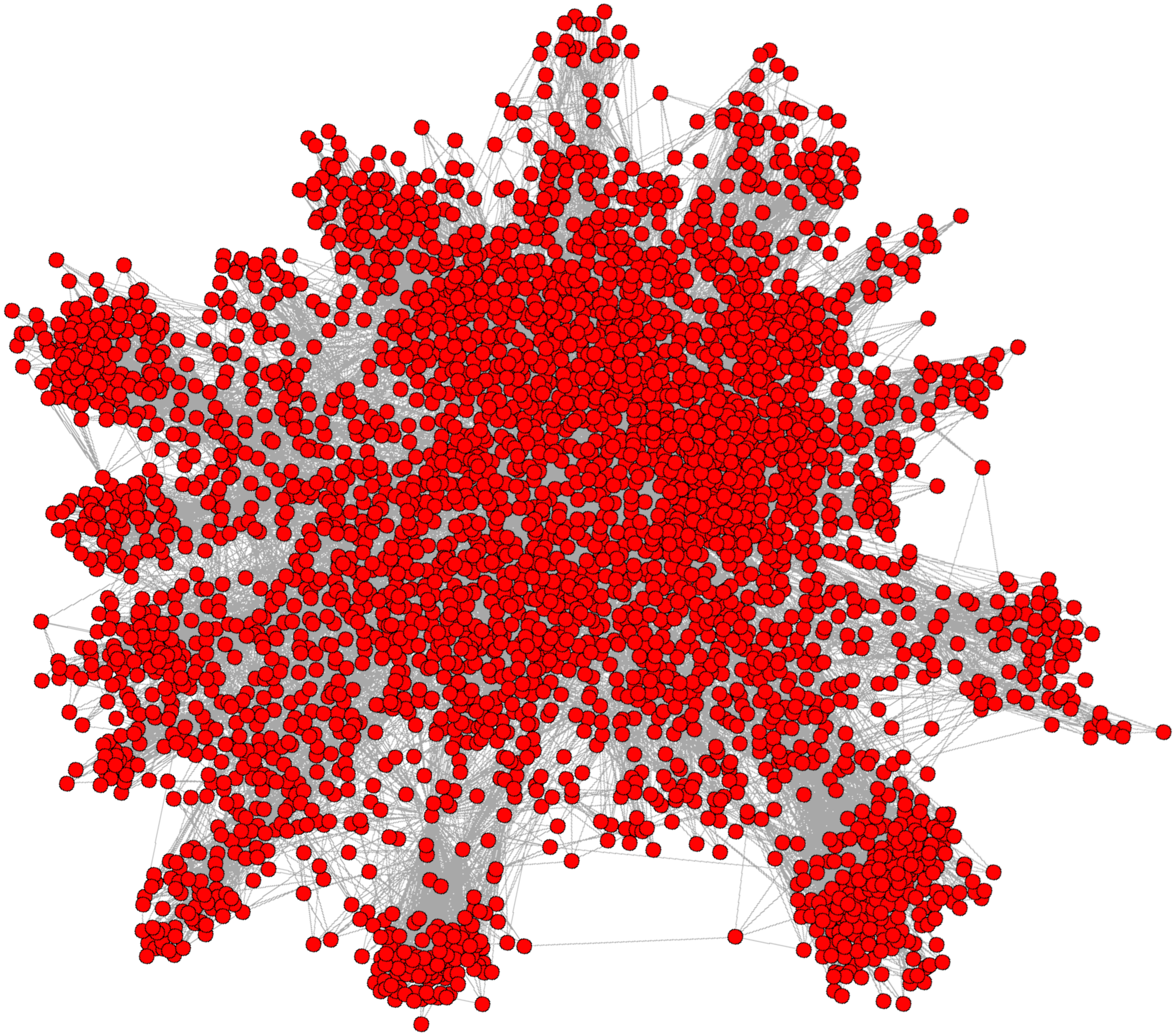
Torsten Hoefler
Department of Computer Science
ETH Zurich
htor@inf.ethz.ch

ABSTRACT

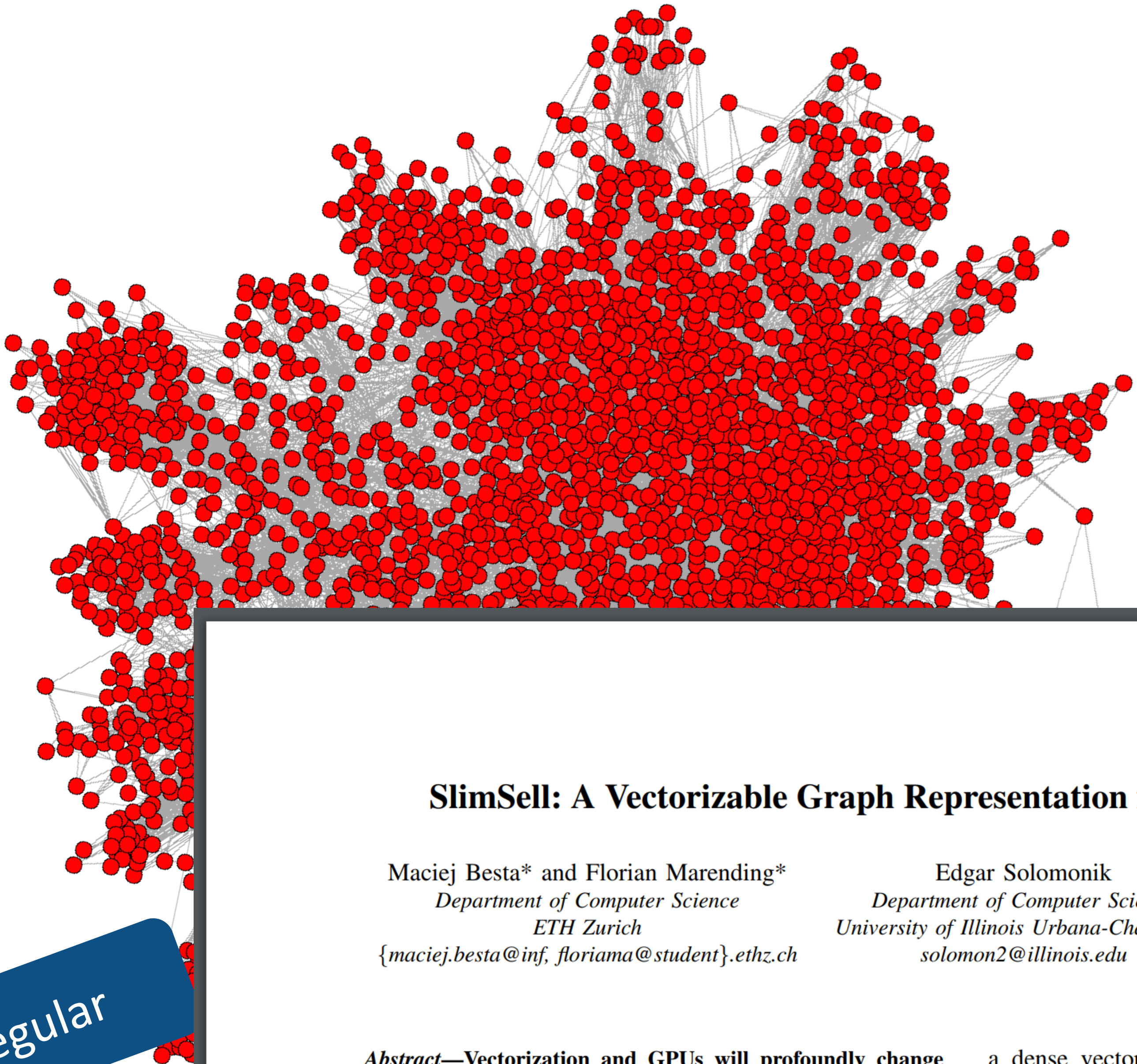
We propose a topology-aware distributed Reader-Writer lock

cesses competing for the same lock. Assume that two of them (A and B) run on one socket and the remaining two (C

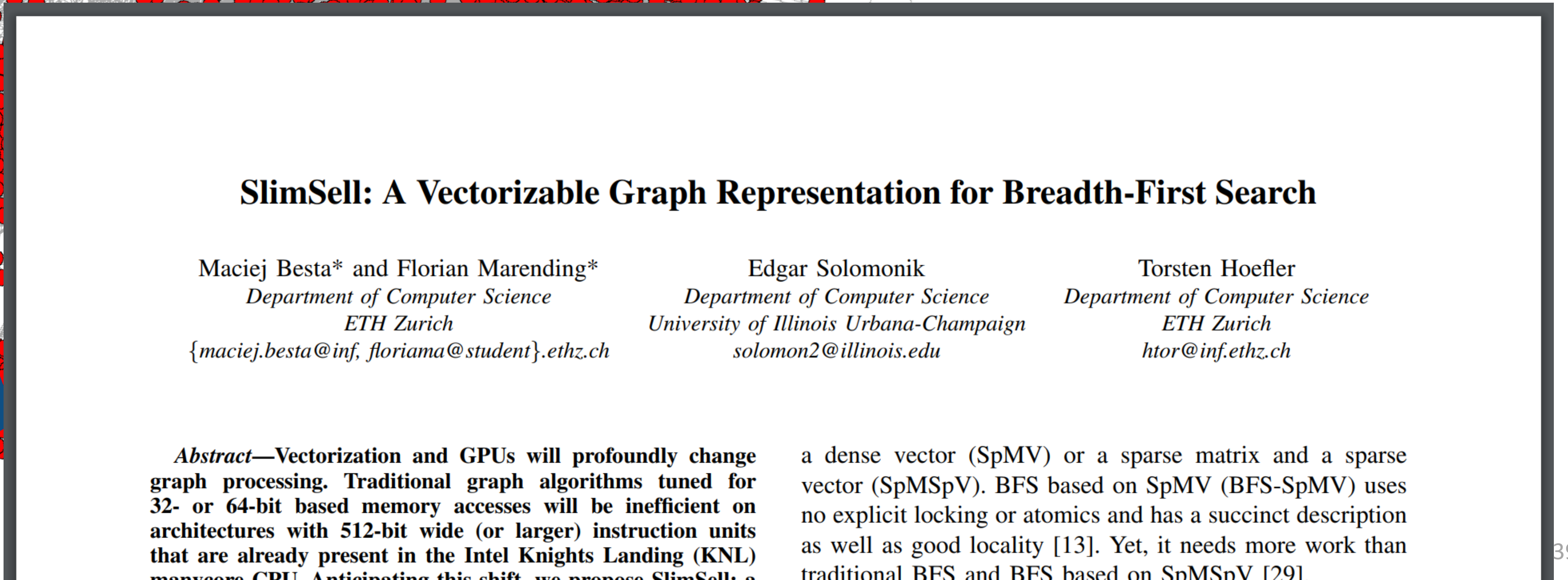
Moving on …



Moving on …



Irregular



SlimSell: A Vectorizable Graph Representation for Breadth-First Search

Maciej Besta* and Florian Marending*
Department of Computer Science
ETH Zurich
{maciej.besta@inf, floriama@student}.ethz.ch

Edgar Solomonik
Department of Computer Science
University of Illinois Urbana-Champaign
solomon2@illinois.edu

Torsten Hoefler
Department of Computer Science
ETH Zurich
htor@inf.ethz.ch

***Abstract*—Vectorization and GPUs will profoundly change graph processing. Traditional graph algorithms tuned for 32- or 64-bit based memory accesses will be inefficient on architectures with 512-bit wide (or larger) instruction units that are already present in the Intel Knights Landing (KNL) manycore CPU. Anticipating this shift, we propose SlimSell: a**

a dense vector (SpMV) or a sparse matrix and a sparse vector (SpMSpV). BFS based on SpMV (BFS-SpMV) uses no explicit locking or atomics and has a succinct description as well as good locality [13]. Yet, it needs more work than traditional BFS and BFS based on SpMSpV [29].

VECTORIZATION

VECTORIZATION

- Deployed in various hardware

Vectorization

- Deployed in various hardware
- Becoming more popular

VECTORIZATION

- Deployed in various hardware
- Becoming more popular

AVX

$C = 8$ (SIMD width)

VECTORIZATION

- Deployed in various hardware
- Becoming more popular

AVX

$C\ = 16$ (SIMD width)

AVX

$C\ = 8$ (SIMD width)

Vectorization

- Deployed in various hardware
- Becoming more popular

AVX

$C = 16$ (SIMD width)

warps

$C = 32$ (warp size)

AVX

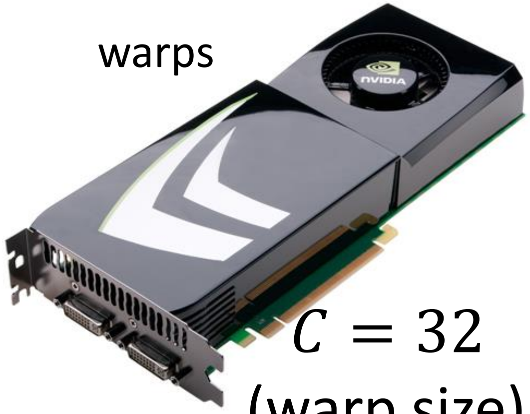
$C = 8$ (SIMD width)

C: „Chunk" size: SIMD width (CPUs, KNLs), warp size (GPUs)

VECTORIZATION

- Deployed in various hardware
- Becoming more popular
- Offers a lot of „regular" compute power

warps



$C\ = 32$
(warp size)

AVX

$C\ = 16$ (SIMD width)

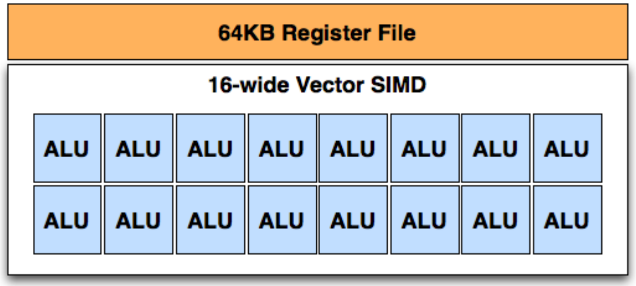
AVX

$C\ = 8$ (SIMD width)

C: **„Chunk" size: SIMD width (CPUs, KNLs), warp size (GPUs)**

VECTORIZATION

- Deployed in various hardware
- Becoming more popular
- Offers a lot of „regular” compute power



warps

$C = 32$ (warp size)

AVX

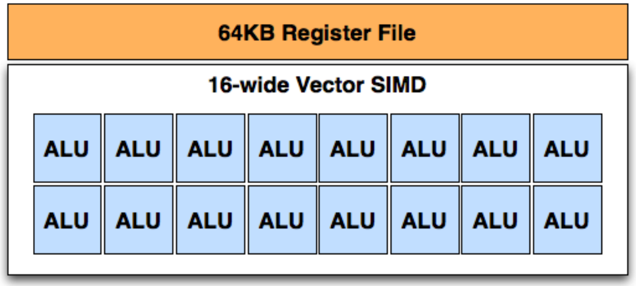
$C = 16$ (SIMD width)

AVX

$C = 8$ (SIMD width)

C: „Chunk” size: SIMD width (CPUs, KNLs), warp size (GPUs)

VECTORIZATION



- Deployed in various hardware
- Becoming more popular
- Offers a lot of „regular" compute power

Regular

warps

$C = 32$ (warp size)

AVX

$C = 16$ (SIMD width)

AVX

$C = 8$ (SIMD width)

C: „Chunk" size: SIMD width (CPUs, KNLs), warp size (GPUs)

VECTORIZATION

- Deployed in various hardware
- Becoming more popular
- Offers a lot of „regular” compute power

Regular

warps

$C = 32$
(warp size)

AVX

$C = 16$ (SIMD width)

AVX

$C = 8$ (SIMD width)

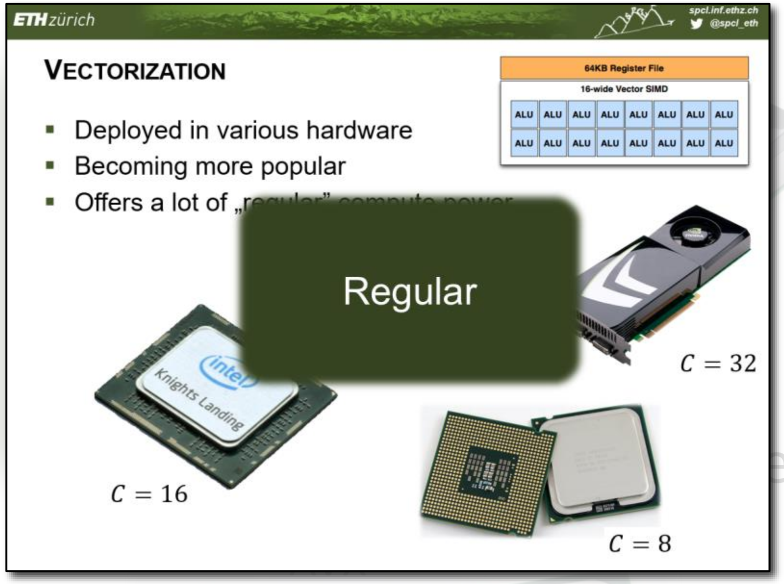
C: „Chunk” size: SIMD width (CPUs, KNLs), warp size (GPUs)

Vectorization

- Deployed in various hardware
- Becoming more popular
- Offers a lot of „regular" compute power

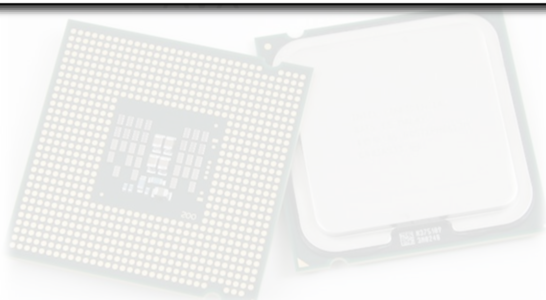
Regu

+



AVX

$C = 16$ (SIMD width)



$C = 8$ (SIMD width)

C: „Chunk" size: SIMD width (CPUs, KNLs), warp size (GPUs)

Breadth-First Search

Traditional Formulation

Breadth-First Search

Traditional Formulation

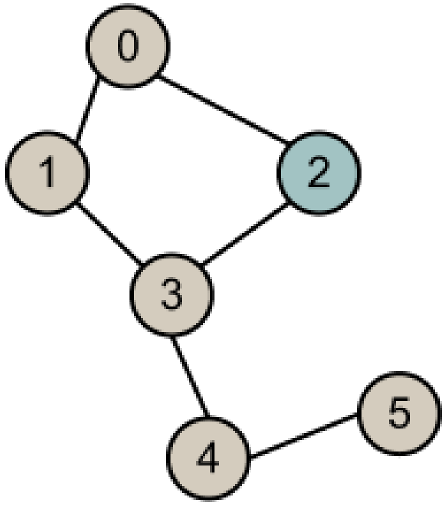
- BFS is based on primitives such as queues

M. Besta et al.: “SlimSell: A Vectorized Graph Representation for Breadth-First Search”, IPDPS’17

Breadth-First Search

Traditional Formulation

- BFS is based on primitives such as queues

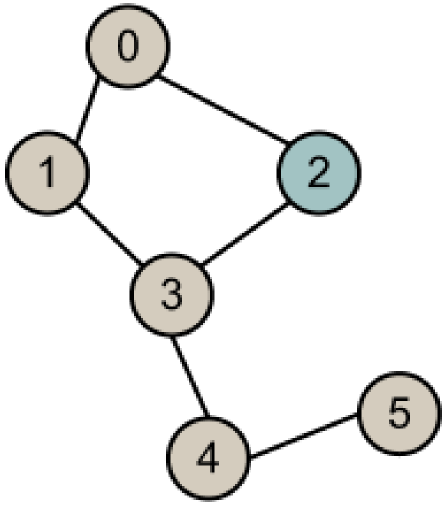


M. Besta et al.: "SlimSell: A Vectorized Graph Representation for Breadth-First Search", IPDPS'17

Breadth-First Search

Traditional Formulation

- BFS is based on primitives such as queues



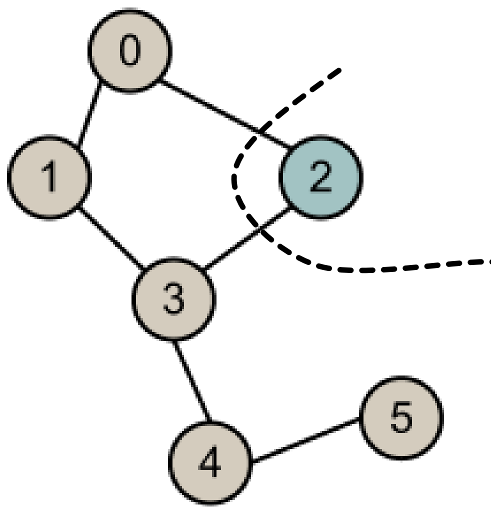
1) F = {}

M. Besta et al.: "SlimSell: A Vectorized Graph Representation for Breadth-First Search", IPDPS'17

Breadth-First Search

Traditional Formulation

- BFS is based on primitives such as queues



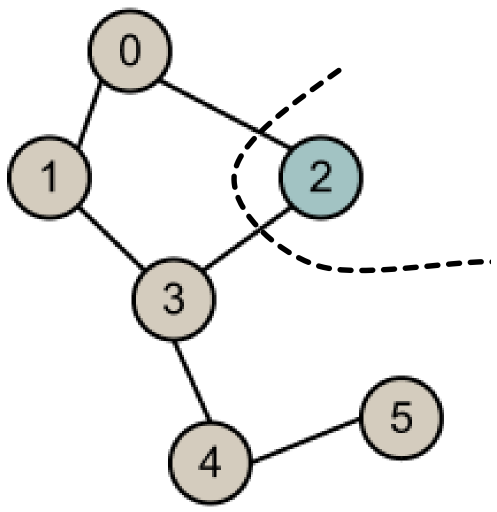
1) F = {}

M. Besta et al.: “SlimSell: A Vectorized Graph Representation for Breadth-First Search”, IPDPS’17

Breadth-First Search

Traditional Formulation

- BFS is based on primitives such as queues

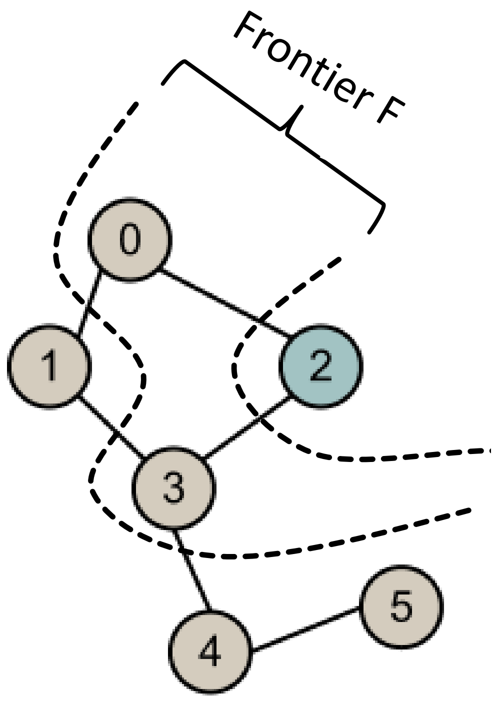


1) F = {}
2) F = {2}

M. Besta et al.: "SlimSell: A Vectorized Graph Representation for Breadth-First Search", IPDPS'17

Breadth-First Search
Traditional Formulation

- BFS is based on primitives such as queues



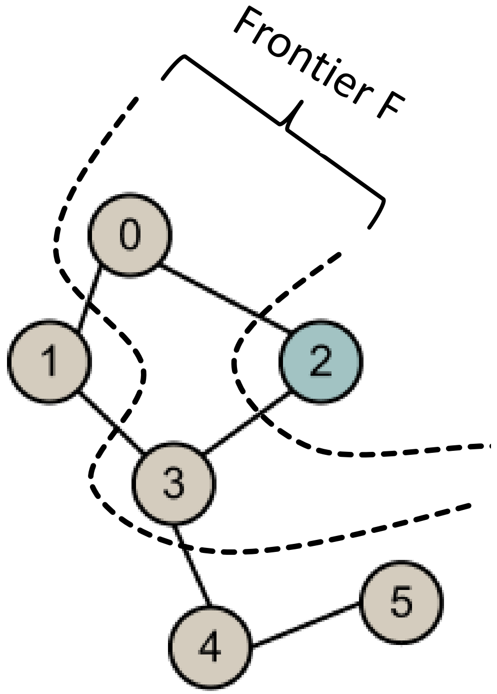
1) F = {}
2) F = {2}

M. Besta et al.: "SlimSell: A Vectorized Graph Representation for Breadth-First Search", IPDPS'17

Breadth-First Search

Traditional Formulation

- BFS is based on primitives such as queues

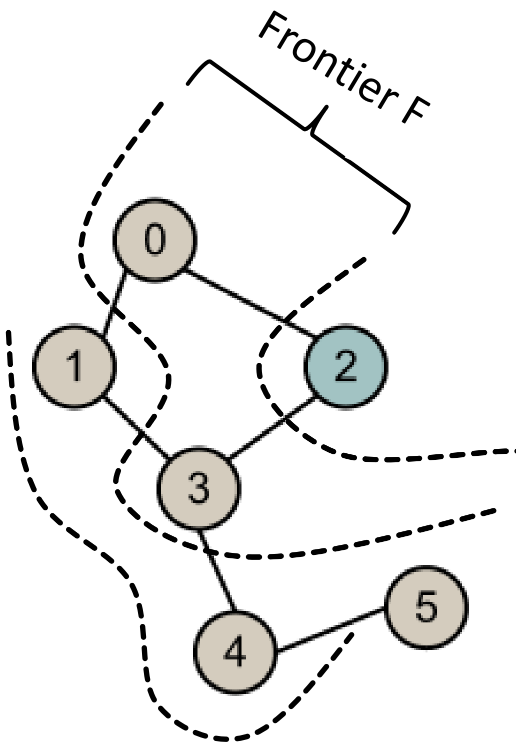


1) F = {}
2) F = {2}
3) F = {0,3}

M. Besta et al.: "SlimSell: A Vectorized Graph Representation for Breadth-First Search", IPDPS'17

Breadth-First Search
Traditional Formulation

- BFS is based on primitives such as queues



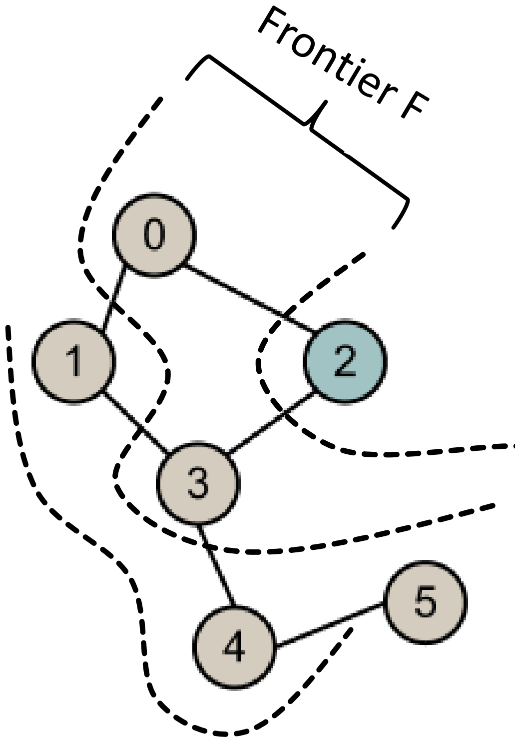
1) F = {}
2) F = {2}
3) F = {0,3}

M. Besta et al.: “SlimSell: A Vectorized Graph Representation for Breadth-First Search”, IPDPS’17

Breadth-First Search

Traditional Formulation

- BFS is based on primitives such as queues



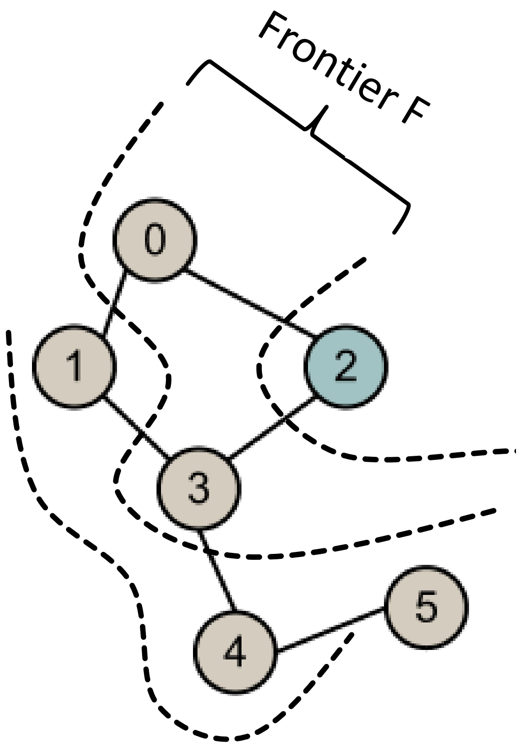
1) F = {}
2) F = {2}
3) F = {0,3}
4) F = {1,4}

M. Besta et al.: “SlimSell: A Vectorized Graph Representation for Breadth-First Search”, IPDPS’17

Breadth-First Search

Traditional Formulation

- BFS is based on primitives such as queues



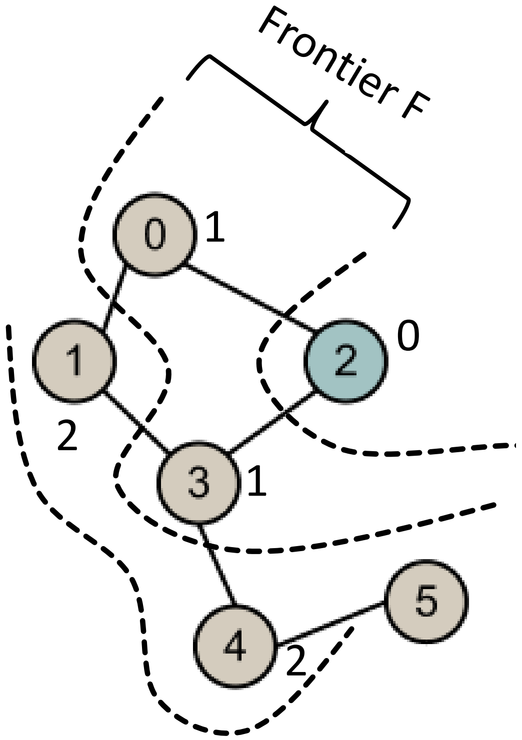
1) F = {}
2) F = {2}
3) F = {0,3}
4) F = {1,4}

Distances from the root

M. Besta et al.: "SlimSell: A Vectorized Graph Representation for Breadth-First Search", IPDPS'17

Breadth-First Search
Traditional Formulation

- BFS is based on primitives such as queues



1) F = {}
2) F = {2}
3) F = {0,3}
4) F = {1,4}

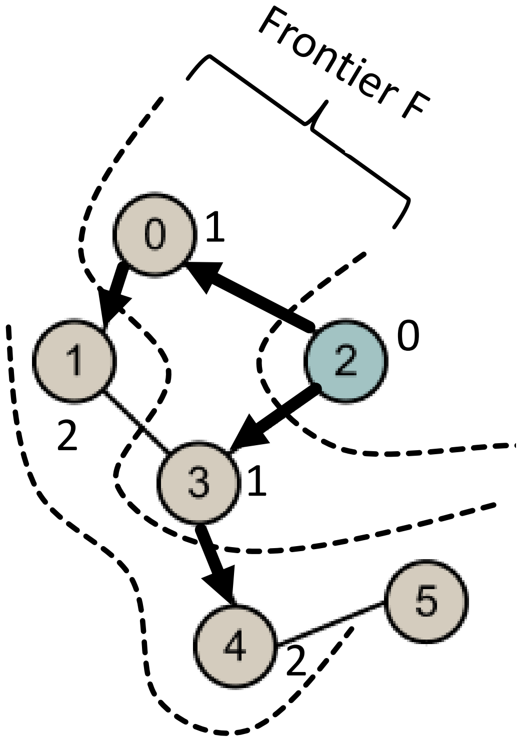
Distances from the root

M. Besta et al.: “SlimSell: A Vectorized Graph Representation for Breadth-First Search”, IPDPS’17

Breadth-First Search

Traditional Formulation

- BFS is based on primitives such as queues



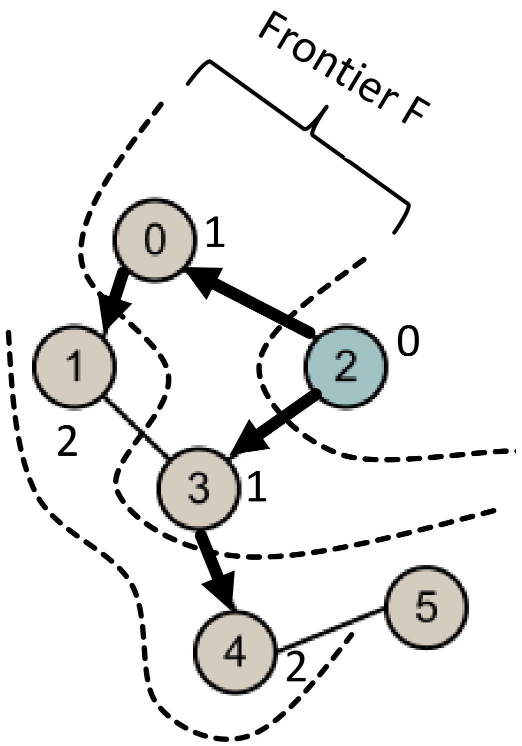
1) F = {}
2) F = {2}
3) F = {0,3}
4) F = {1,4}

Distances from the root

M. Besta et al.: "SlimSell: A Vectorized Graph Representation for Breadth-First Search", IPDPS'17

Breadth-First Search
Traditional Formulation

- BFS is based on primitives such as queues



1) F = {}
2) F = {2}
3) F = {0,3}
4) F = {1,4}

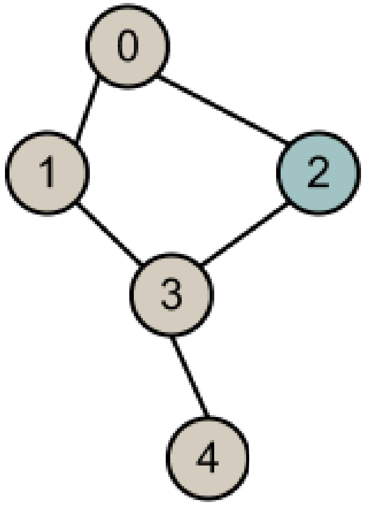
Distances from the root

Parents (predecessors) in the traversal tree

M. Besta et al.: "SlimSell: A Vectorized Graph Representation for Breadth-First Search", IPDPS'17

Breadth-First Search

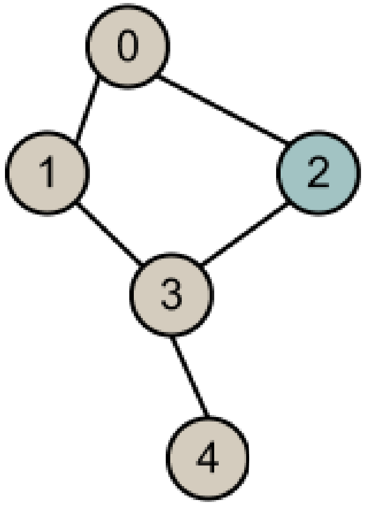
Algebraic Formulation



M. Besta et al.: “SlimSell: A Vectorized Graph Representation for Breadth-First Search”, IPDPS’17

Breadth-First Search
Algebraic Formulation

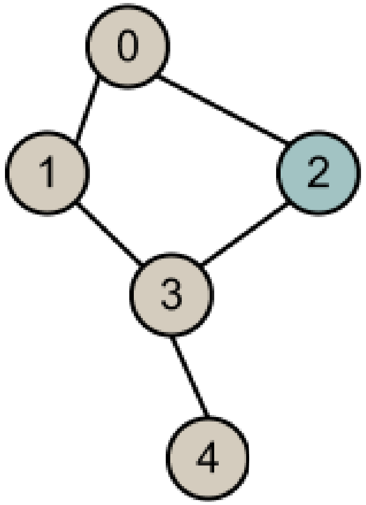
- BFS is a series of matrix-vector products



M. Besta et al.: "SlimSell: A Vectorized Graph Representation for Breadth-First Search", IPDPS'17

Breadth-First Search
Algebraic Formulation

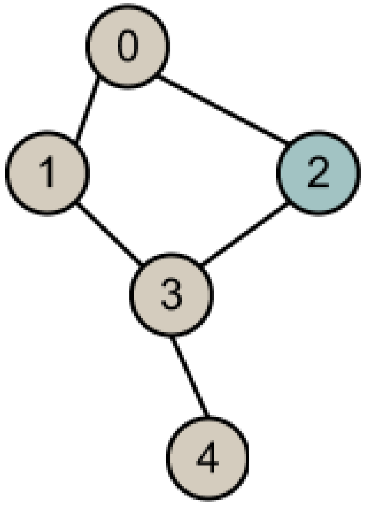
- BFS is a series of matrix-vector products
- Graph is modeled by an adjacency matrix



M. Besta et al.: "SlimSell: A Vectorized Graph Representation for Breadth-First Search", IPDPS'17

Breadth-First Search
Algebraic Formulation

- BFS is a series of matrix-vector products
- Graph is modeled by an adjacency matrix



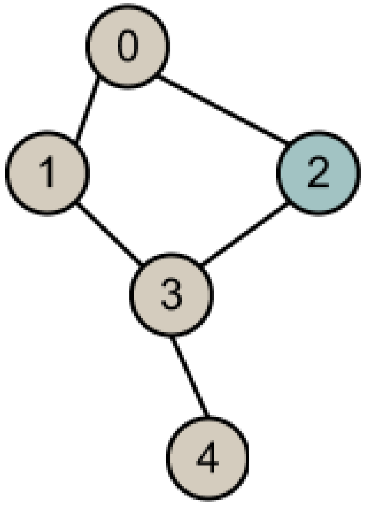
Adjacency Matrix:

$$A = \begin{bmatrix} 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 \end{bmatrix}$$

M. Besta et al.: "SlimSell: A Vectorized Graph Representation for Breadth-First Search", IPDPS'17

Breadth-First Search
Algebraic Formulation

- BFS is a series of matrix-vector products
- Graph is modeled by an adjacency matrix
- Multiplication is done over a semiring



Adjacency Matrix:

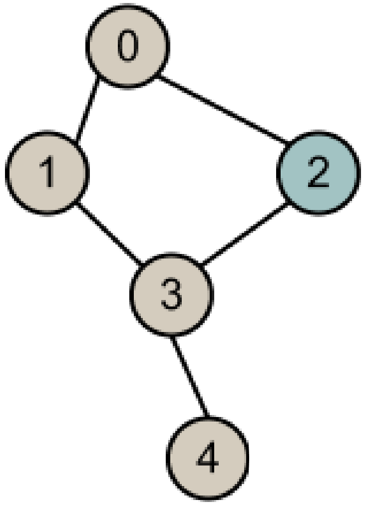
$$A = \begin{bmatrix} 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 \end{bmatrix}$$

M. Besta et al.: "SlimSell: A Vectorized Graph Representation for Breadth-First Search", IPDPS'17

Breadth-First Search

Algebraic Formulation

- BFS is a series of matrix-vector products
- Graph is modeled by an adjacency matrix
- Multiplication is done over a semiring



Adjacency Matrix:

$$A = \begin{bmatrix} 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 \end{bmatrix}$$

Semiring:

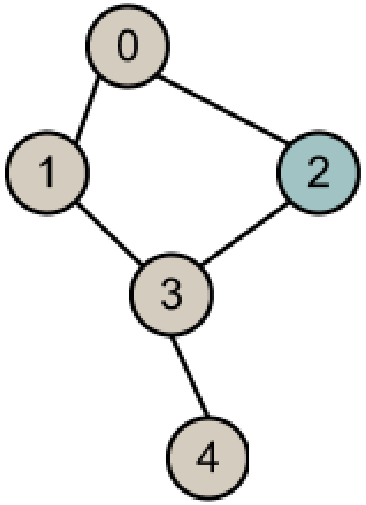
$$(\mathbb{R}\,, op_1, op_2, el_1, el_2)$$

M. Besta et al.: “SlimSell: A Vectorized Graph Representation for Breadth-First Search”, IPDPS’17

Breadth-First Search

Algebraic Formulation

- BFS is a series of matrix-vector products
- Graph is modeled by an adjacency matrix
- Multiplication is done over a semiring



Adjacency Matrix:

$$A = \begin{bmatrix} 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 \end{bmatrix}$$

Semiring:

$$(\mathbb{R}\,, op_1, op_2, el_1, el_2)$$

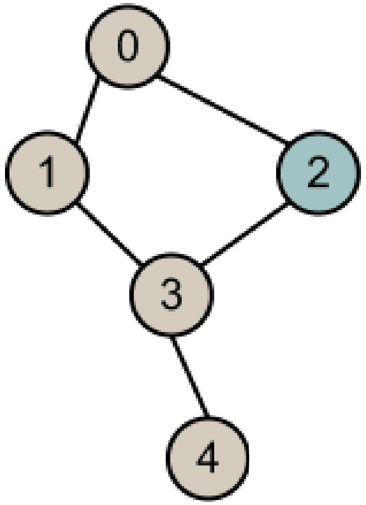
$$\begin{bmatrix} 1 & 2 \\ 3 & 1 \end{bmatrix} \begin{bmatrix} 0 \\ 2 \end{bmatrix} = \begin{bmatrix} 4 \\ 2 \end{bmatrix}$$

M. Besta et al.: “SlimSell: A Vectorized Graph Representation for Breadth-First Search”, IPDPS’17

Breadth-First Search

Algebraic Formulation

- BFS is a series of matrix-vector products
- Graph is modeled by an adjacency matrix
- Multiplication is done over a semiring



Adjacency Matrix:

$$A = \begin{bmatrix} 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 \end{bmatrix}$$

Semiring:

$$(\mathbb{R}\,, op_1, op_2, el_1, el_2)$$

$$(\mathbb{R}, +, \cdot, 0, 1)$$

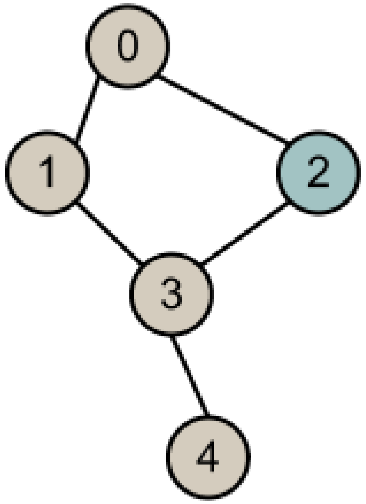
$$\begin{bmatrix} 1 & 2 \\ 3 & 1 \end{bmatrix} \begin{bmatrix} 0 \\ 2 \end{bmatrix} = \begin{bmatrix} 4 \\ 2 \end{bmatrix}$$

M. Besta et al.: "SlimSell: A Vectorized Graph Representation for Breadth-First Search", IPDPS'17

Breadth-First Search
Algebraic Formulation

Tropical Semiring

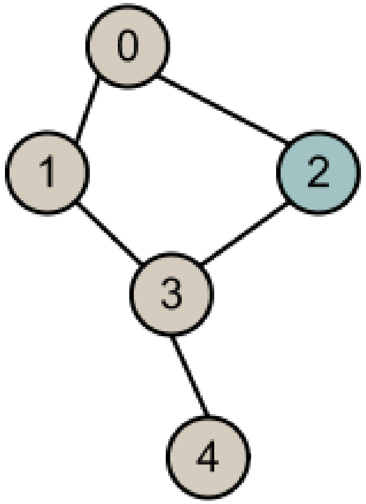
$(\mathbb{R} \cup \{\infty\}, min, +, \infty, 0)$



Breadth-First Search
Algebraic Formulation

Tropical Semiring

$(\mathbb{R} \cup \{\infty\}, min, +, \infty, 0)$



$$A' = \begin{bmatrix} 0 & 1 & 1 & \infty & \infty \\ 1 & 0 & \infty & 1 & \infty \\ 1 & \infty & 0 & 1 & \infty \\ \infty & 1 & 1 & 0 & 1 \\ \infty & \infty & \infty & 1 & 0 \end{bmatrix}$$

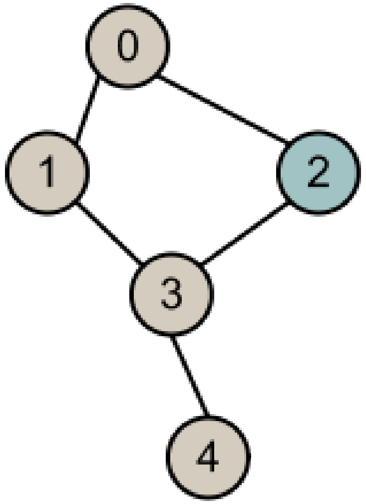
M. Besta et al.: "SlimSell: A Vectorized Graph Representation for Breadth-First Search", IPDPS'17

Breadth-First Search
Algebraic Formulation

Tropical Semiring

$(\mathbb{R} \cup \{\infty\}, min, +, \infty, 0)$

$$A = \begin{bmatrix} 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 \end{bmatrix}$$



$$A' = \begin{bmatrix} 0 & 1 & 1 & \infty & \infty \\ 1 & 0 & \infty & 1 & \infty \\ 1 & \infty & 0 & 1 & \infty \\ \infty & 1 & 1 & 0 & 1 \\ \infty & \infty & \infty & 1 & 0 \end{bmatrix}$$

M. Besta et al.: “SlimSell: A Vectorized Graph Representation for Breadth-First Search”, IPDPS’17

Breadth-First Search
Algebraic Formulation

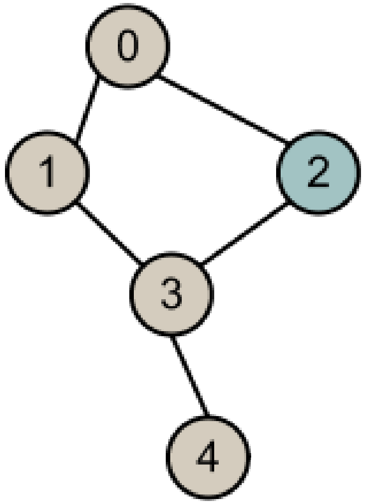
Tropical Semiring

$(\mathbb{R} \cup \{\infty\}, min, +, \infty, 0)$

$$A = \begin{bmatrix} 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 \end{bmatrix}$$

Usually stored using a sparse format

$$A' = \begin{bmatrix} 0 & 1 & 1 & \infty & \infty \\ 1 & 0 & \infty & 1 & \infty \\ 1 & \infty & 0 & 1 & \infty \\ \infty & 1 & 1 & 0 & 1 \\ \infty & \infty & \infty & 1 & 0 \end{bmatrix}$$



Breadth-First Search
Algebraic Formulation

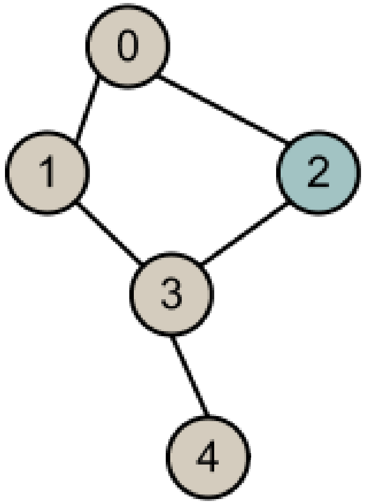
Tropical Semiring

$(\mathbb{R} \cup \{\infty\}, min, +, \infty, 0)$

$$A = \begin{bmatrix} 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 \end{bmatrix}$$

Usually stored using a sparse format

$$A' = \begin{bmatrix} 0 & 1 & 1 & \infty & \infty \\ 1 & 0 & \infty & 1 & \infty \\ 1 & \infty & 0 & 1 & \infty \\ \infty & 1 & 1 & 0 & 1 \\ \infty & \infty & \infty & 1 & 0 \end{bmatrix} \qquad f_0 = \begin{pmatrix} \infty \\ \infty \\ 0 \\ \infty \\ \infty \end{pmatrix}$$



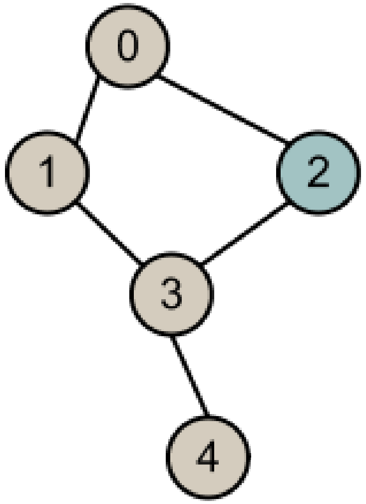
M. Besta et al.: "SlimSell: A Vectorized Graph Representation for Breadth-First Search", IPDPS'17

Breadth-First Search
Algebraic Formulation

Tropical Semiring

$(\mathbb{R} \cup \{\infty\}, min, +, \infty, 0)$

$$A = \begin{bmatrix} 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 \end{bmatrix}$$



Usually stored using a sparse format

$$A' = \begin{bmatrix} 0 & 1 & 1 & \infty & \infty \\ 1 & 0 & \infty & 1 & \infty \\ 1 & \infty & 0 & 1 & \infty \\ \infty & 1 & 1 & 0 & 1 \\ \infty & \infty & \infty & 1 & 0 \end{bmatrix}$$

Stored with a dense or a sparse format

$$f_0 = \begin{pmatrix} \infty \\ \infty \\ 0 \\ \infty \\ \infty \end{pmatrix}$$

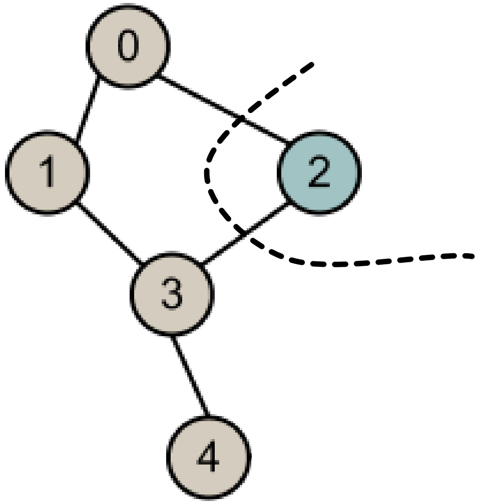
M. Besta et al.: “SlimSell: A Vectorized Graph Representation for Breadth-First Search”, IPDPS’17

Breadth-First Search
Algebraic Formulation

Tropical Semiring

$(\mathbb{R} \cup \{\infty\}, min, +, \infty, 0)$

$$A = \begin{bmatrix} 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 \end{bmatrix}$$



Usually stored using a sparse format

$$A' = \begin{bmatrix} 0 & 1 & 1 & \infty & \infty \\ 1 & 0 & \infty & 1 & \infty \\ 1 & \infty & 0 & 1 & \infty \\ \infty & 1 & 1 & 0 & 1 \\ \infty & \infty & \infty & 1 & 0 \end{bmatrix}$$

Stored with a dense or a sparse format

$$f_0 = \begin{pmatrix} \infty \\ \infty \\ 0 \\ \infty \\ \infty \end{pmatrix}$$

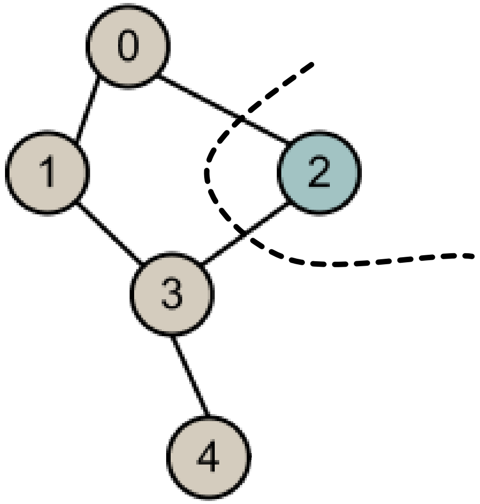
M. Besta et al.: “SlimSell: A Vectorized Graph Representation for Breadth-First Search”, IPDPS’17

Breadth-First Search
Algebraic Formulation

Tropical Semiring

$(\mathbb{R} \cup \{\infty\}, min, +, \infty, 0)$

$$A = \begin{bmatrix} 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 \end{bmatrix}$$



Usually stored using a sparse format

Stored with a dense or a sparse format

$$A' = \begin{bmatrix} 0 & 1 & 1 & \infty & \infty \\ 1 & 0 & \infty & 1 & \infty \\ 1 & \infty & 0 & 1 & \infty \\ \infty & 1 & 1 & 0 & 1 \\ \infty & \infty & \infty & 1 & 0 \end{bmatrix} \qquad f_0 = \begin{pmatrix} \infty \\ \infty \\ 0 \\ \infty \\ \infty \end{pmatrix}$$

$$f_1 = A'^T \otimes_T f_0 = \begin{pmatrix} \\ \\ \\ \end{pmatrix}$$

M. Besta et al.: “SlimSell: A Vectorized Graph Representation for Breadth-First Search”, IPDPS’17

Breadth-First Search
Algebraic Formulation

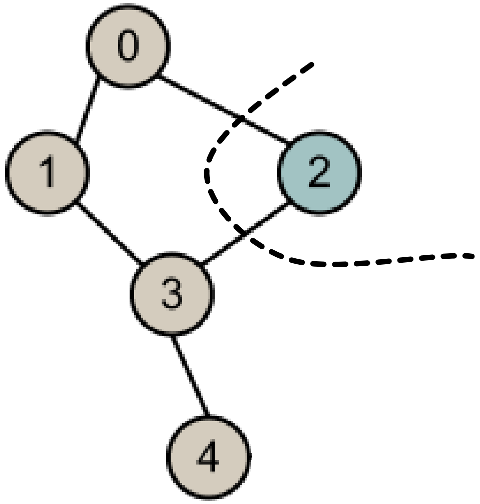
Tropical Semiring

$(\mathbb{R} \cup \{\infty\}, min, +, \infty, 0)$

$$A = \begin{bmatrix} 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 \end{bmatrix}$$

Usually stored using a sparse format

Stored with a dense or a sparse format



$$A' = \begin{bmatrix} 0 & 1 & 1 & \infty & \infty \\ 1 & 0 & \infty & 1 & \infty \\ 1 & \infty & 0 & 1 & \infty \\ \infty & 1 & 1 & 0 & 1 \\ \infty & \infty & \infty & 1 & 0 \end{bmatrix} \qquad f_0 = \begin{pmatrix} \infty \\ \infty \\ 0 \\ \infty \\ \infty \end{pmatrix}$$

$$f_1 = A'^T \otimes_T f_0 = \begin{pmatrix} \\ \\ \\ \end{pmatrix}$$

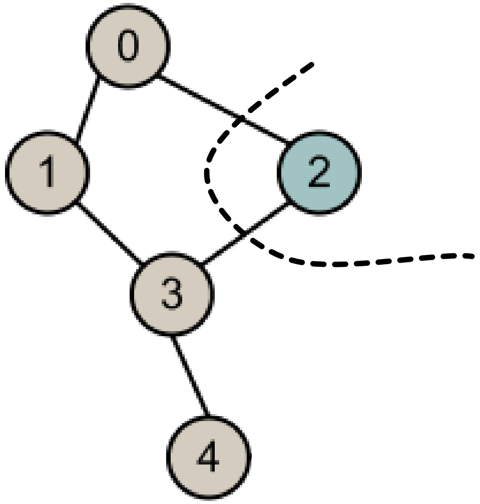
M. Besta et al.: “SlimSell: A Vectorized Graph Representation for Breadth-First Search”, IPDPS’17

Breadth-First Search
Algebraic Formulation

Tropical Semiring

$(\mathbb{R} \cup \{\infty\}, min, +, \infty, 0)$

$$A = \begin{bmatrix} 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 \end{bmatrix}$$



Usually stored using a sparse format

Stored with a dense or a sparse format

$$A' = \begin{bmatrix} 0 & 1 & 1 & \infty & \infty \\ 1 & 0 & \infty & 1 & \infty \\ 1 & \infty & 0 & 1 & \infty \\ \infty & 1 & 1 & 0 & 1 \\ \infty & \infty & \infty & 1 & 0 \end{bmatrix} \qquad f_0 = \begin{pmatrix} \infty \\ \infty \\ 0 \\ \infty \\ \infty \end{pmatrix}$$

$$f_1 = A'^T \otimes_T f_0 = \begin{pmatrix} 1 \\ \\ \\ \\ \end{pmatrix}$$

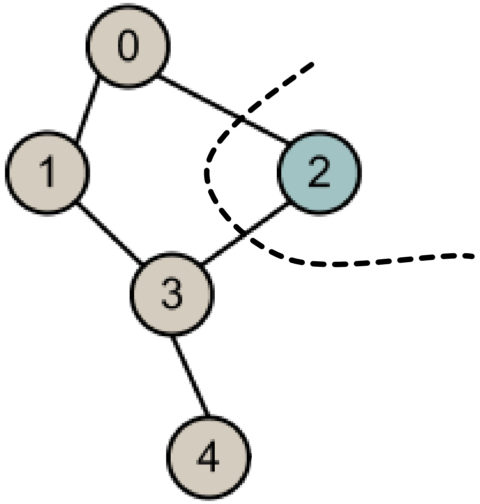
M. Besta et al.: “SlimSell: A Vectorized Graph Representation for Breadth-First Search”, IPDPS’17

Breadth-First Search
Algebraic Formulation

Tropical Semiring

$(\mathbb{R} \cup \{\infty\}, min, +, \infty, 0)$

$$A = \begin{bmatrix} 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 \end{bmatrix}$$



Usually stored using a sparse format

Stored with a dense or a sparse format

$$A' = \begin{bmatrix} 0 & 1 & 1 & \infty & \infty \\ 1 & 0 & \infty & 1 & \infty \\ 1 & \infty & 0 & 1 & \infty \\ \infty & 1 & 1 & 0 & 1 \\ \infty & \infty & \infty & 1 & 0 \end{bmatrix} \quad f_0 = \begin{pmatrix} \infty \\ \infty \\ 0 \\ \infty \\ \infty \end{pmatrix}$$

$$f_1 = A'^T \otimes_T f_0 = \begin{pmatrix} 1 \\ \infty \\ 0 \\ 1 \\ \infty \end{pmatrix}$$

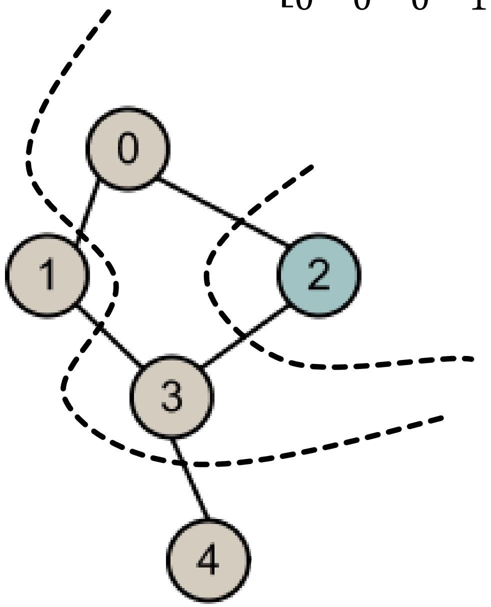
M. Besta et al.: “SlimSell: A Vectorized Graph Representation for Breadth-First Search”, IPDPS’17

Breadth-First Search
Algebraic Formulation

Tropical Semiring

$(\mathbb{R} \cup \{\infty\}, min, +, \infty, 0)$

$$A = \begin{bmatrix} 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 \end{bmatrix}$$



Usually stored using a sparse format

Stored with a dense or a sparse format

$$A' = \begin{bmatrix} 0 & 1 & 1 & \infty & \infty \\ 1 & 0 & \infty & 1 & \infty \\ 1 & \infty & 0 & 1 & \infty \\ \infty & 1 & 1 & 0 & 1 \\ \infty & \infty & \infty & 1 & 0 \end{bmatrix} \qquad f_0 = \begin{pmatrix} \infty \\ \infty \\ 0 \\ \infty \\ \infty \end{pmatrix}$$

$$f_1 = A'^T \otimes_T f_0 = \begin{pmatrix} 1 \\ \infty \\ 0 \\ 1 \\ \infty \end{pmatrix}$$

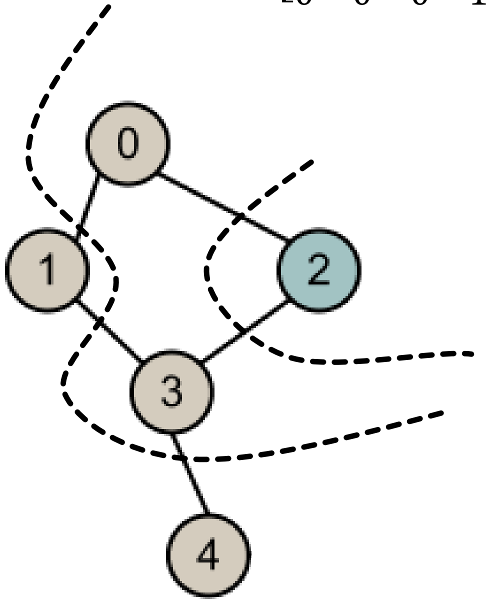
M. Besta et al.: “SlimSell: A Vectorized Graph Representation for Breadth-First Search”, IPDPS’17

Breadth-First Search
Algebraic Formulation

Tropical Semiring

$(\mathbb{R} \cup \{\infty\}, min, +, \infty, 0)$

$$A = \begin{bmatrix} 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 \end{bmatrix}$$



Usually stored using a sparse format

Stored with a dense or a sparse format

$$A' = \begin{bmatrix} 0 & 1 & 1 & \infty & \infty \\ 1 & 0 & \infty & 1 & \infty \\ 1 & \infty & 0 & 1 & \infty \\ \infty & 1 & 1 & 0 & 1 \\ \infty & \infty & \infty & 1 & 0 \end{bmatrix} \qquad f_0 = \begin{pmatrix} \infty \\ \infty \\ 0 \\ \infty \\ \infty \end{pmatrix}$$

$$f_1 = A'^T \otimes_T f_0 = \begin{pmatrix} 1 \\ \infty \\ 0 \\ 1 \\ \infty \end{pmatrix} \qquad f_2 = \begin{pmatrix} 1 \\ 2 \\ 0 \\ 1 \\ 2 \end{pmatrix}$$

M. Besta et al.: “SlimSell: A Vectorized Graph Representation for Breadth-First Search”, IPDPS’17

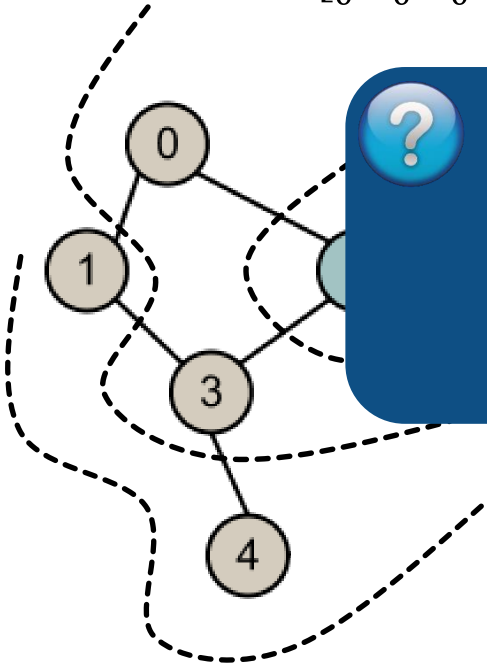
Breadth-First Search

Algebraic Formulation

Tropical Semiring

$(\mathbb{R} \cup \{\infty\}, min, +, \infty, 0)$

$$A = \begin{bmatrix} 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 \end{bmatrix}$$



Usually stored using a sparse format

Stored with a dense or a sparse format

$$\begin{bmatrix} 0 & 1 & 1 & \infty & \infty \end{bmatrix}$$

$$f_0 = \begin{pmatrix} \infty \\ \infty \\ 0 \\ \infty \\ \infty \end{pmatrix}$$

How to do this in practice?

$$f_1 = A'^T \otimes_T f_0 = \begin{pmatrix} \infty \\ 0 \\ 1 \\ \infty \end{pmatrix}$$

$$f_2 = \begin{pmatrix} 1 \\ 2 \\ 0 \\ 1 \\ 2 \end{pmatrix}$$

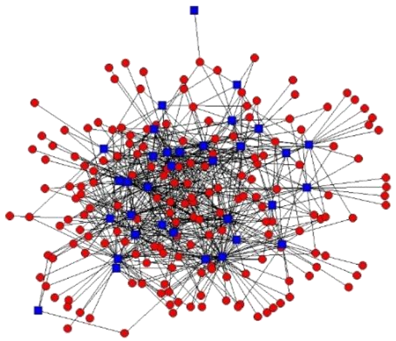
M. Besta et al.: “SlimSell: A Vectorized Graph Representation for Breadth-First Search”, IPDPS’17

GRAPH REPRESENTATIONS

COMPRESSED SPARSE ROW (CSR)

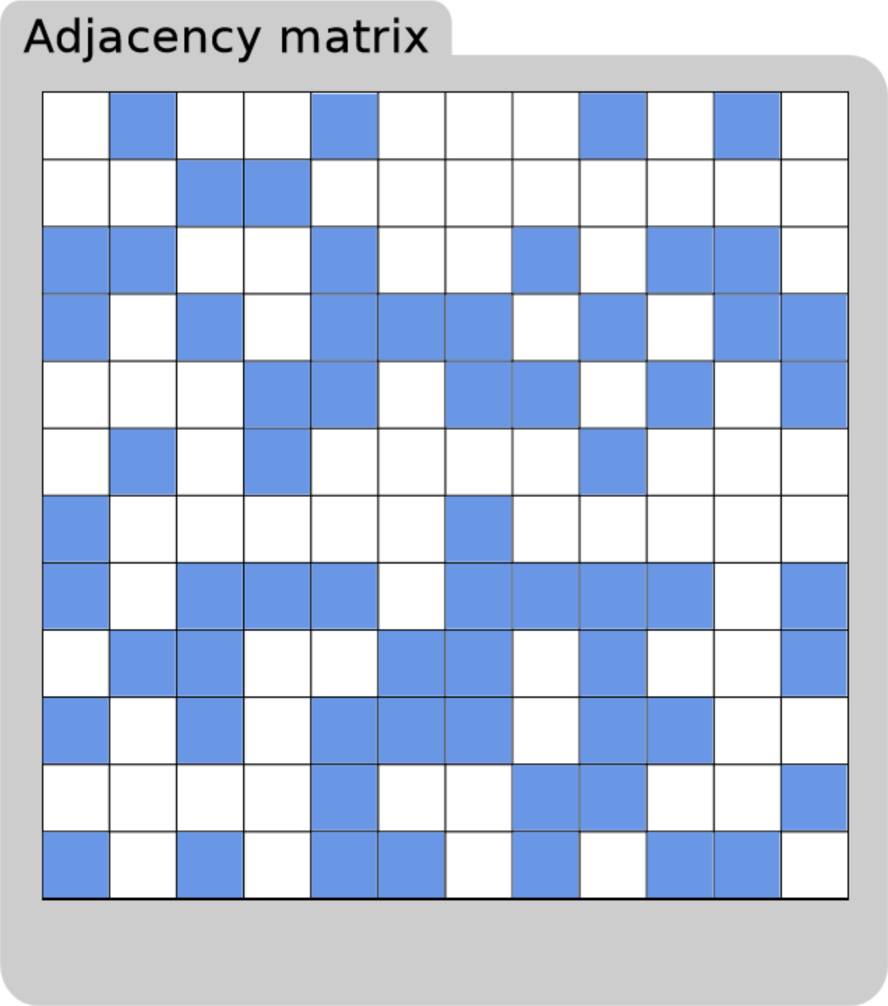
M. Besta et al.: "SlimSell: A Vectorized Graph Representation for Breadth-First Search", IPDPS'17

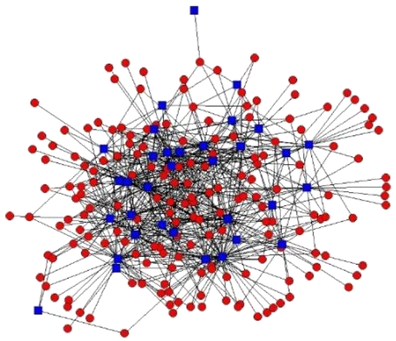
Graph Representations
Compressed Sparse Row (CSR)



M. Besta et al.: "SlimSell: A Vectorized Graph Representation for Breadth-First Search", IPDPS'17

Graph Representations
Compressed Sparse Row (CSR)

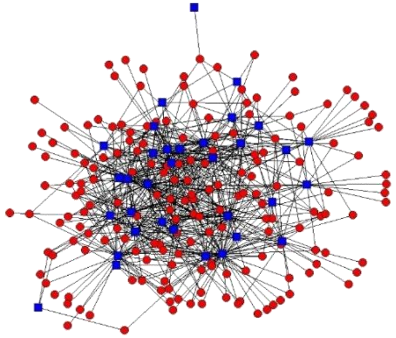


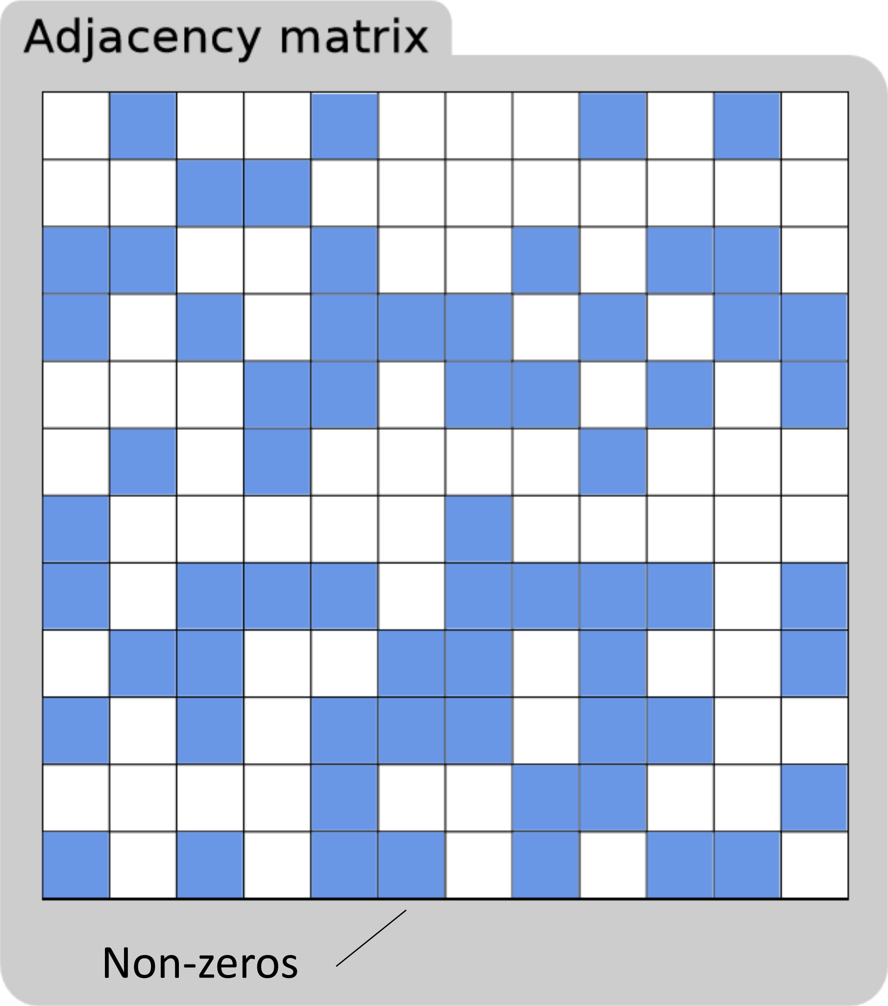


M. Besta et al.: “SlimSell: A Vectorized Graph Representation for Breadth-First Search”, IPDPS’17

Graph Representations

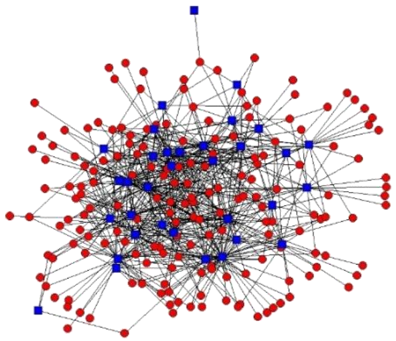
Compressed Sparse Row (CSR)

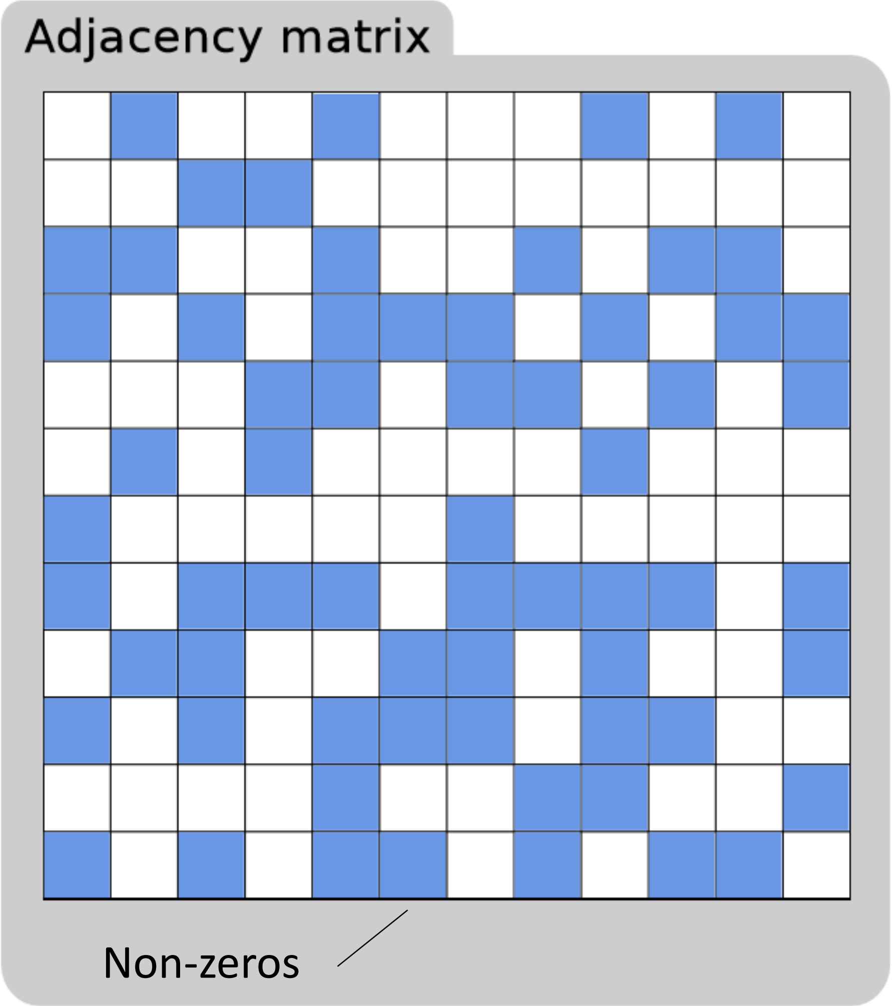




M. Besta et al.: “SlimSell: A Vectorized Graph Representation for Breadth-First Search”, IPDPS’17

Graph Representations
Compressed Sparse Row (CSR)





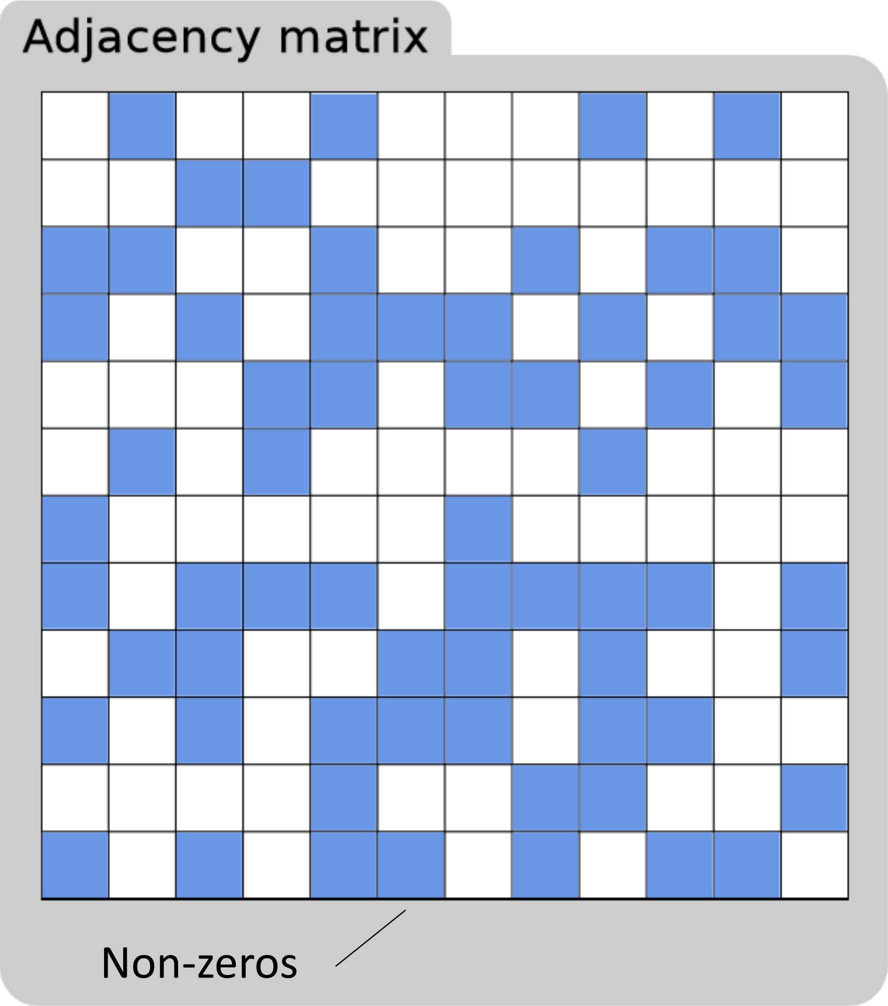
Non-zeros are stored in the *val* array

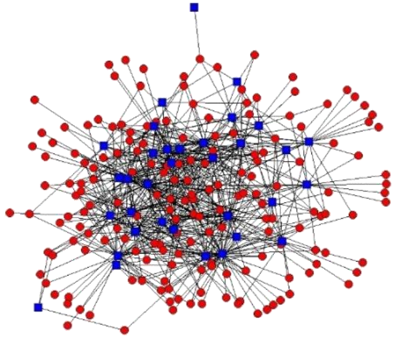
size: *2m* cells

n: number of vertices
m: number of edges

M. Besta et al.: “SlimSell: A Vectorized Graph Representation for Breadth-First Search”, IPDPS’17

Graph Representations
Compressed Sparse Row (CSR)





Non-zeros are stored in the *val* array

size: *2m* cells

Column indices stored in the *col* array

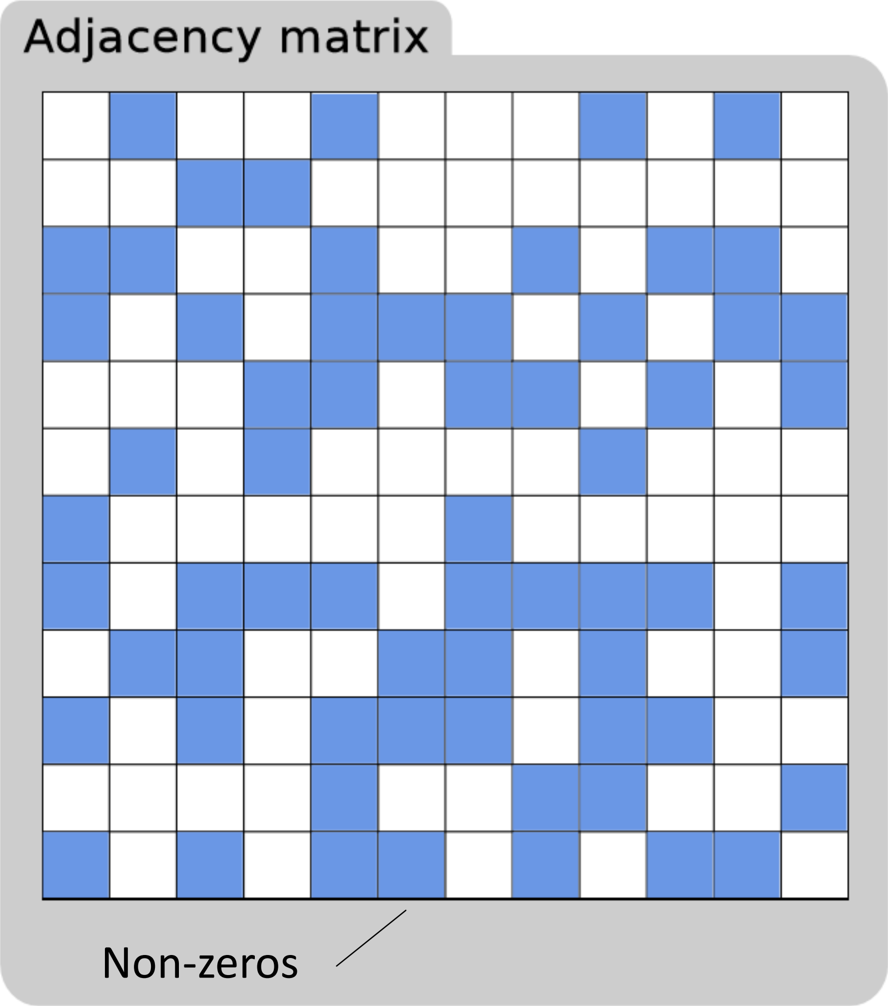
size: *2m* cells

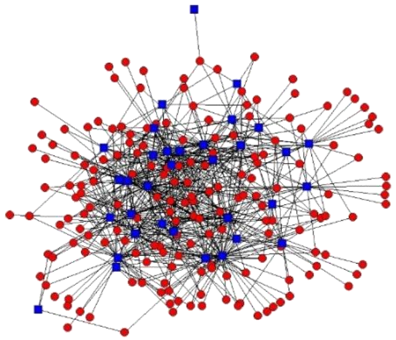
n: number of vertices
m: number of edges

M. Besta et al.: “SlimSell: A Vectorized Graph Representation for Breadth-First Search”, IPDPS’17

Graph Representations

Compressed Sparse Row (CSR)

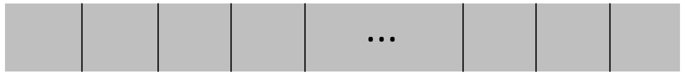




Non-zeros are stored in the *val* array — size: *2m* cells

Column indices stored in the *col* array — size: *2m* cells

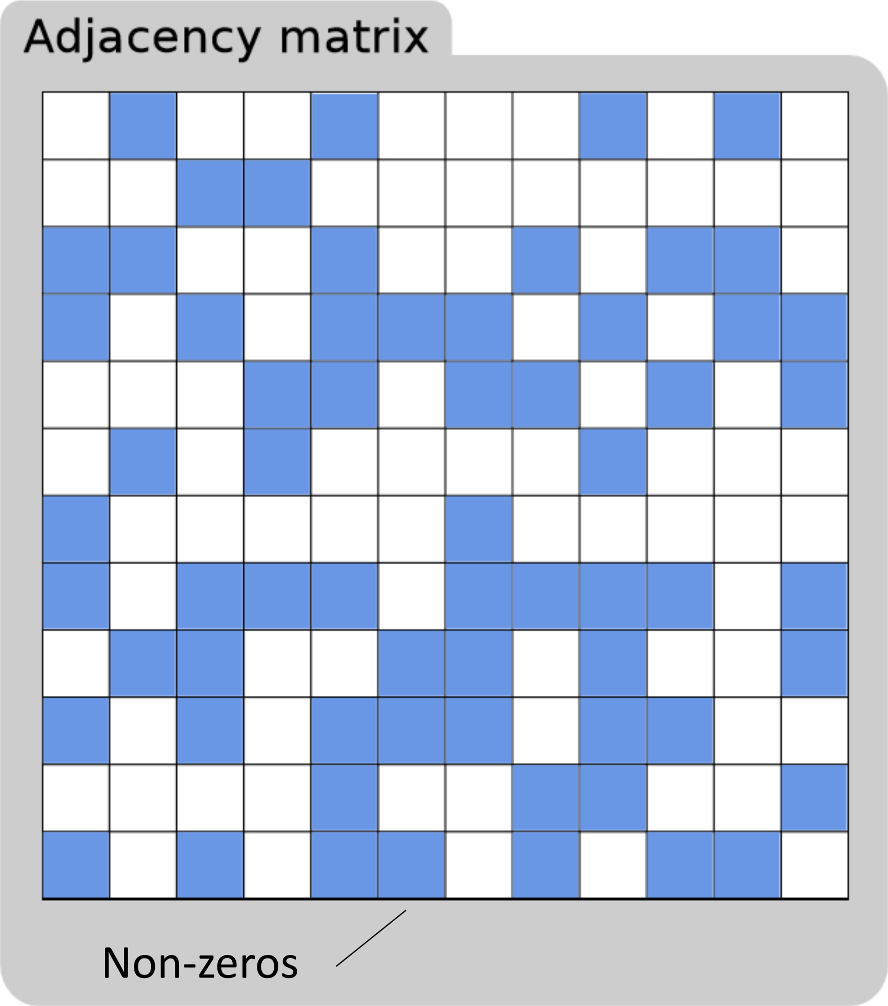
Row indices are stored in the *row* array — size: *n* cells

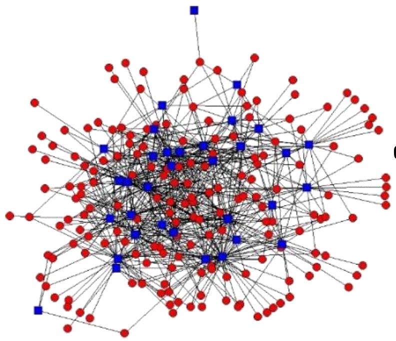


n: number of vertices
m: number of edges

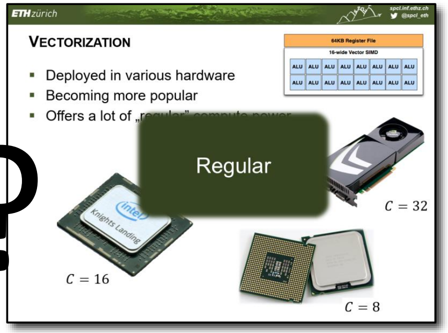
M. Besta et al.: “SlimSell: A Vectorized Graph Representation for Breadth-First Search”, IPDPS’17

Graph Representations
Compressed Sparse Row (CSR)





?



Non-zeros are stored in the *val* array

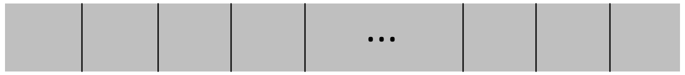
size: *2m* cells

Column indices stored in the *col* array

size: *2m* cells

Row indices are stored in the *row* array

size: *n* cells



n: number of vertices

m: number of edges

M. Besta et al.: "SlimSell: A Vectorized Graph Representation for Breadth-First Search", IPDPS'17

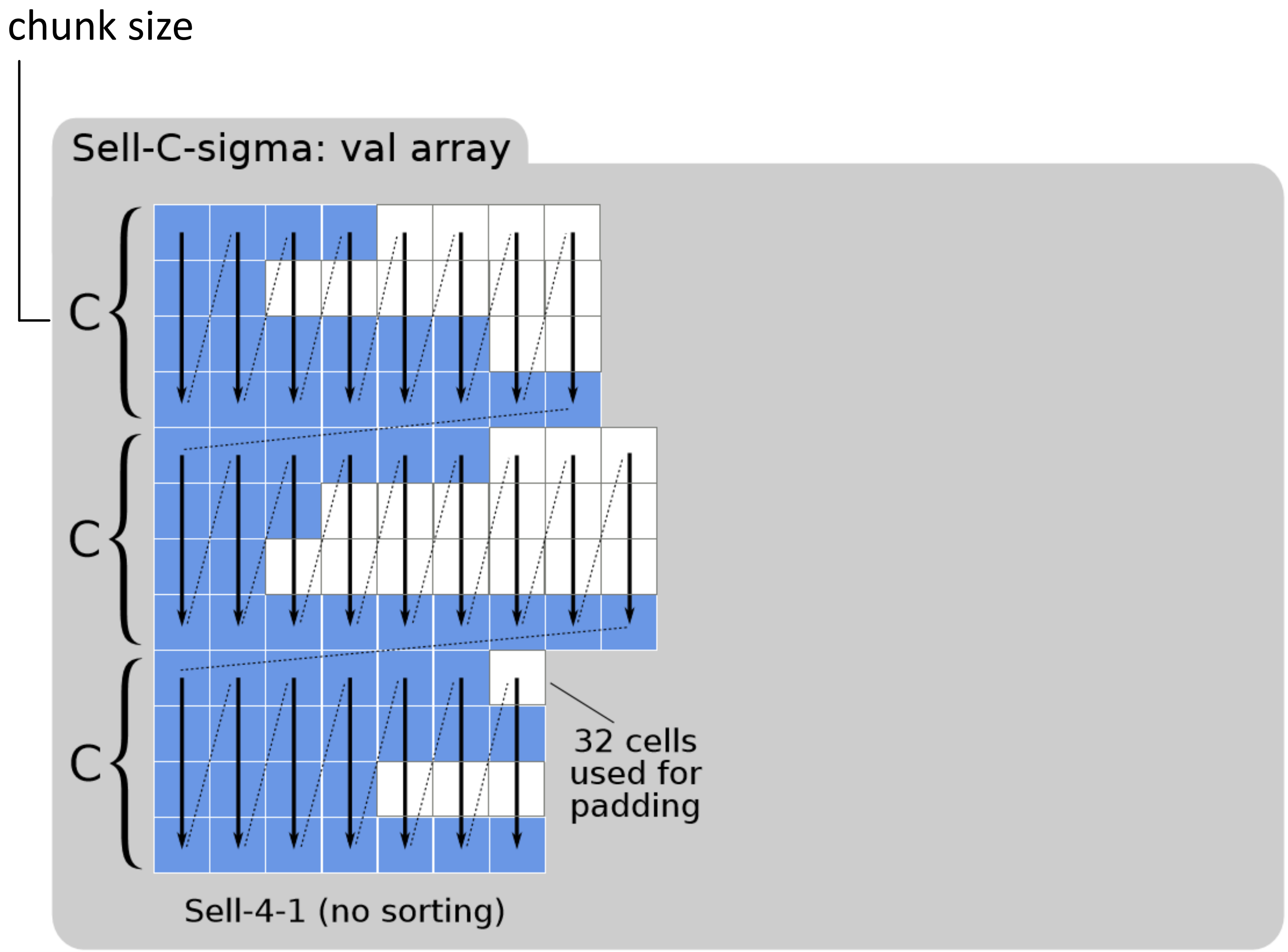
Graph Representations

Sell-C-Sigma

M. Besta et al.: “SlimSell: A Vectorized Graph Representation for Breadth-First Search”, IPDPS’17

Graph Representations

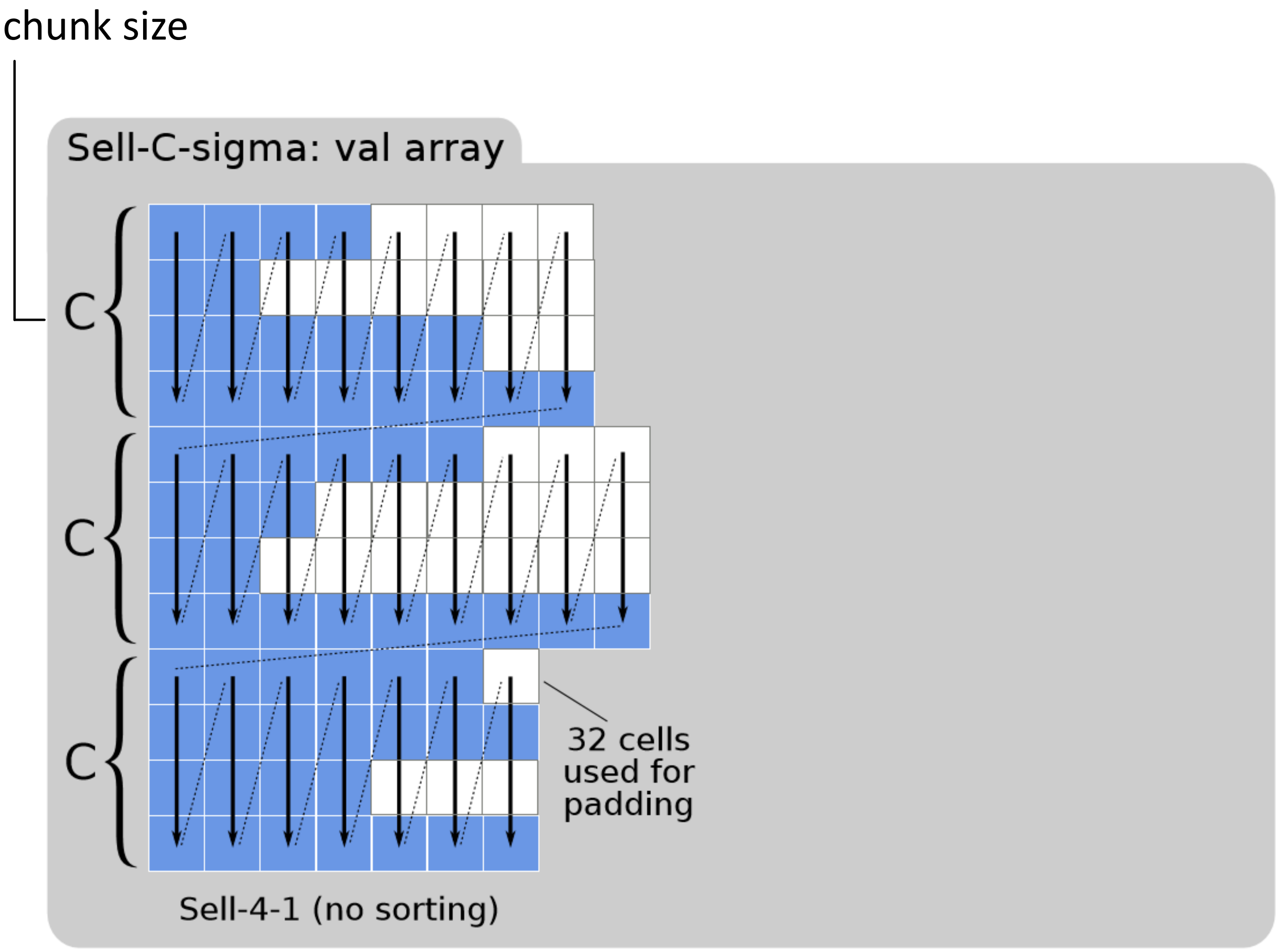
Sell-C-Sigma

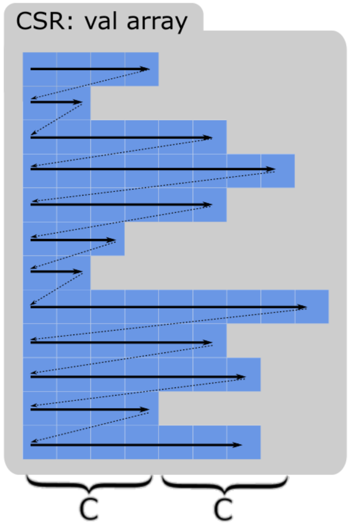


M. Besta et al.: “SlimSell: A Vectorized Graph Representation for Breadth-First Search”, IPDPS’17

Graph Representations

Sell-C-Sigma

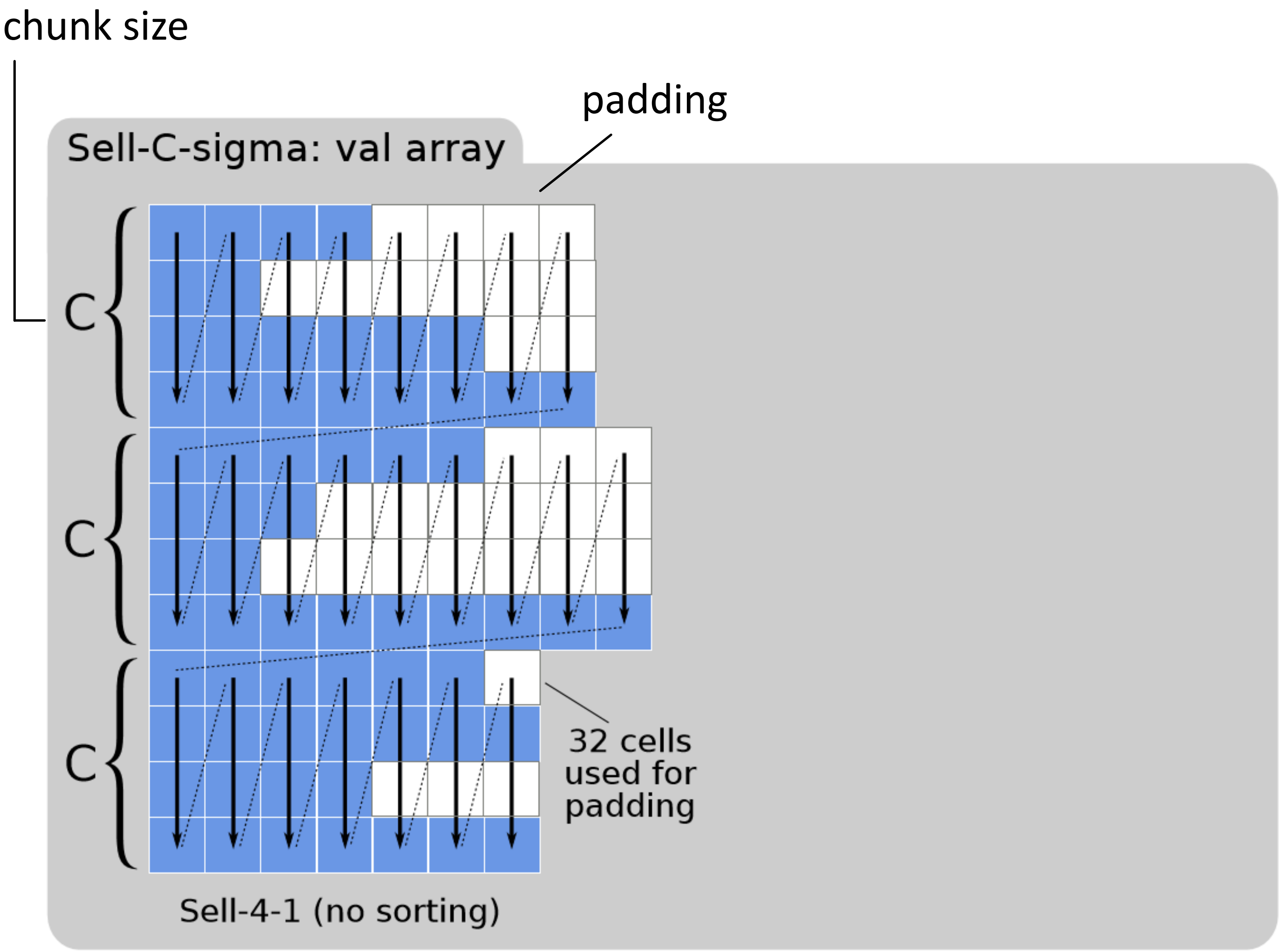


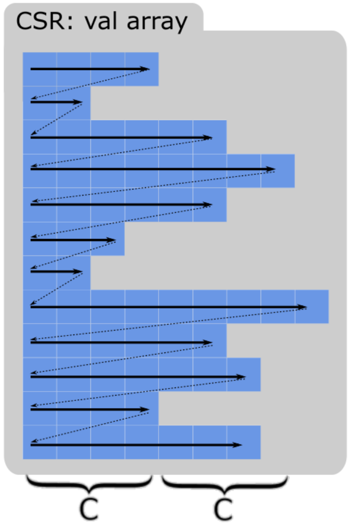


M. Besta et al.: "SlimSell: A Vectorized Graph Representation for Breadth-First Search", IPDPS'17

Graph Representations

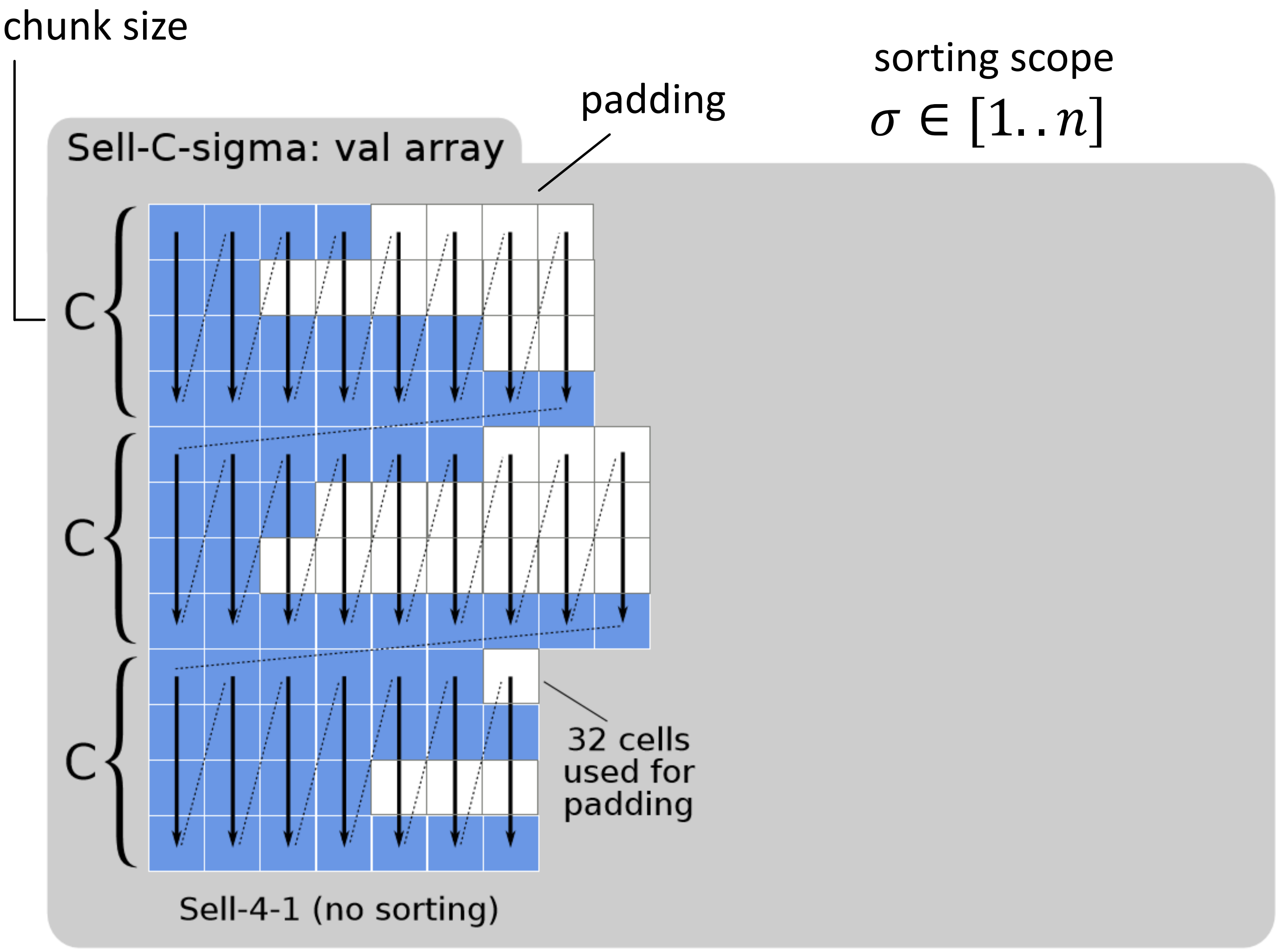
Sell-C-Sigma

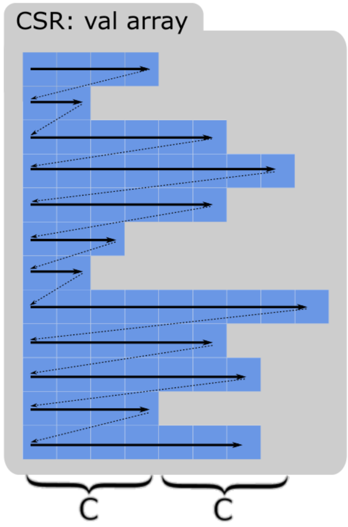




M. Besta et al.: “SlimSell: A Vectorized Graph Representation for Breadth-First Search”, IPDPS’17

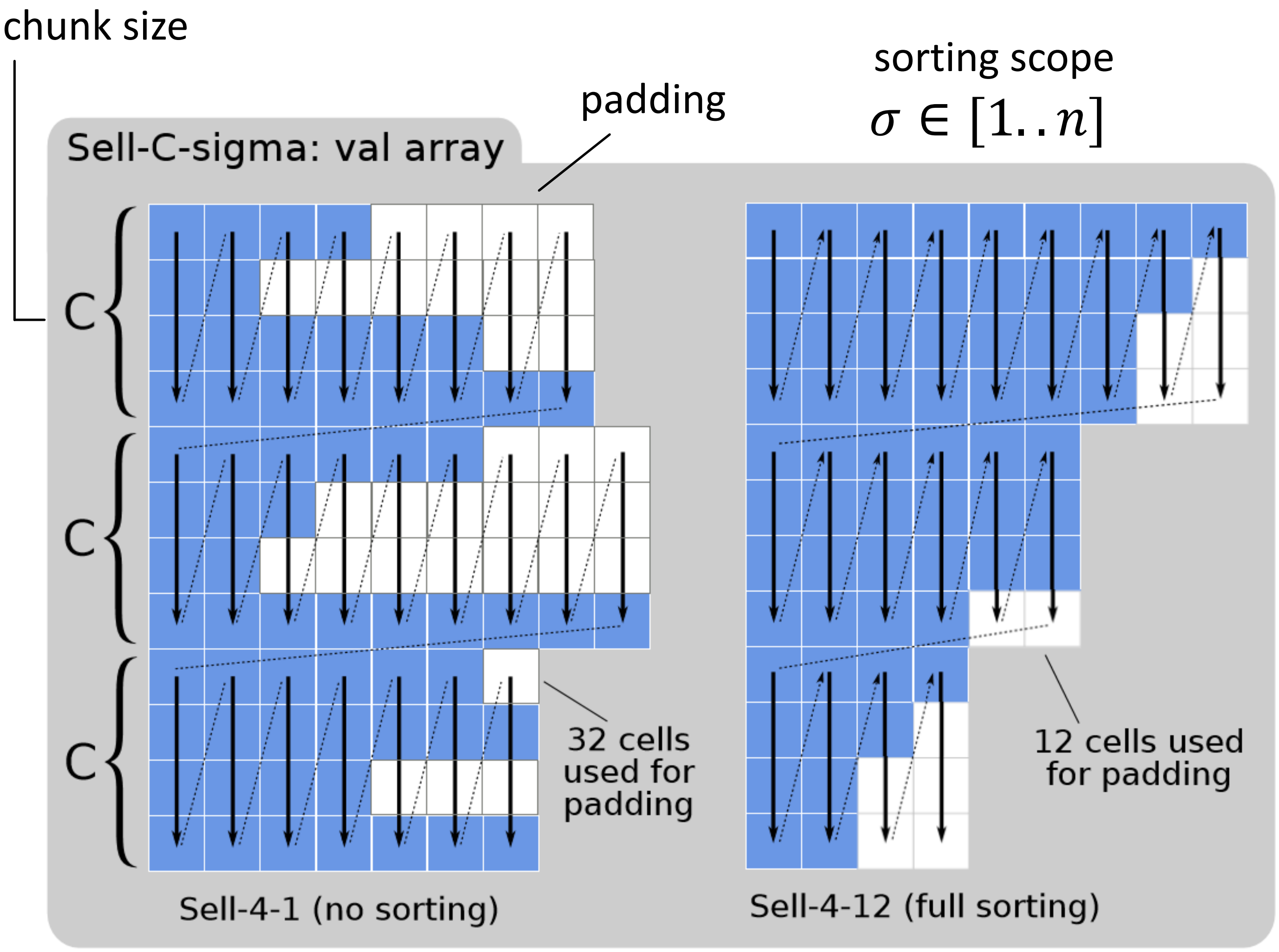
Graph Representations
Sell-C-Sigma

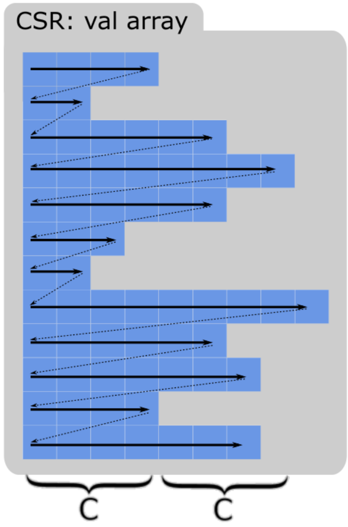




M. Besta et al.: "SlimSell: A Vectorized Graph Representation for Breadth-First Search", IPDPS'17

Graph Representations
Sell-C-Sigma

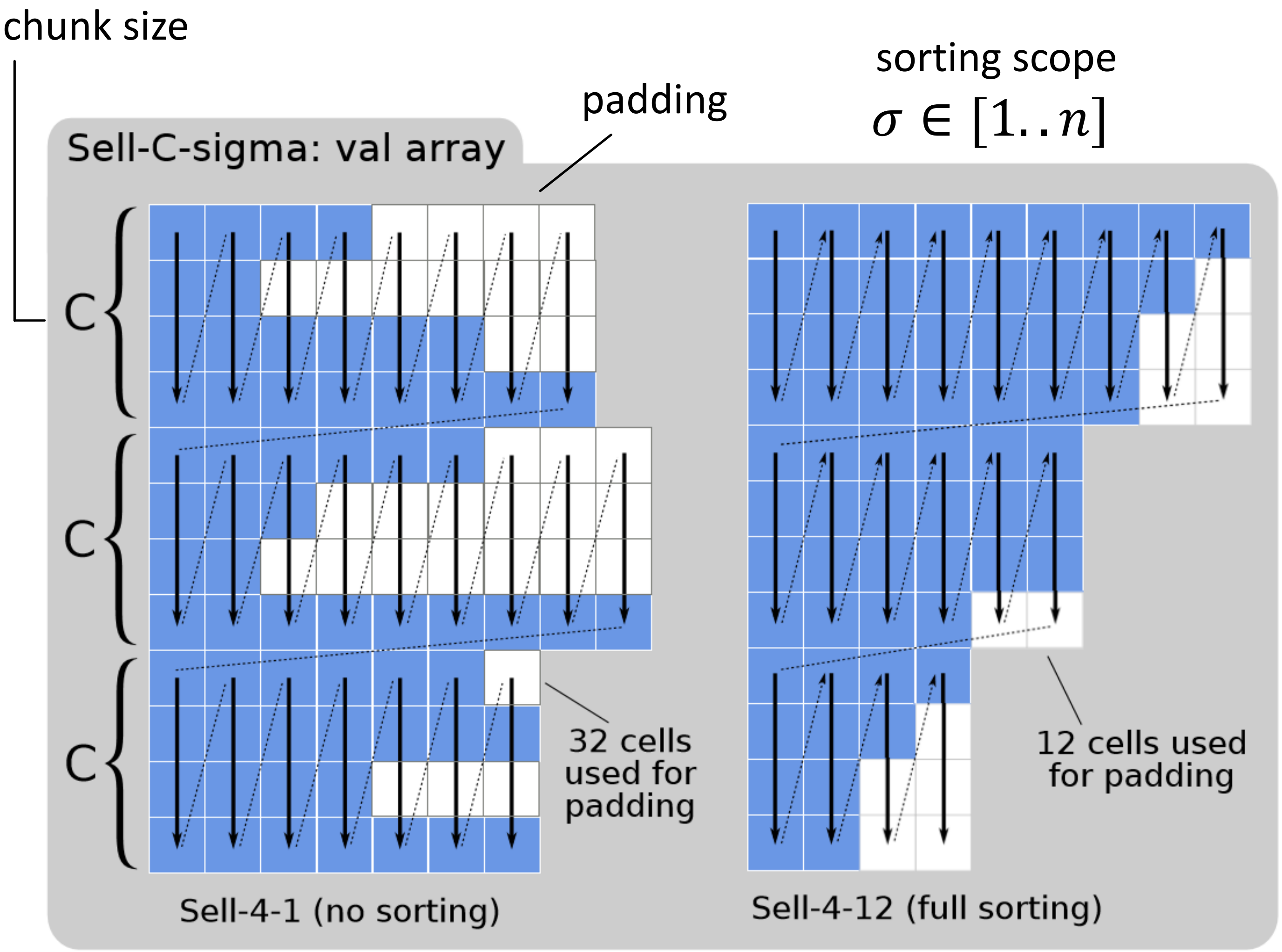


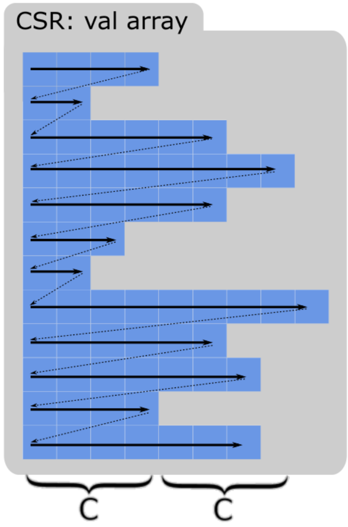


M. Besta et al.: "SlimSell: A Vectorized Graph Representation for Breadth-First Search", IPDPS'17

Graph Representations

Sell-C-Sigma



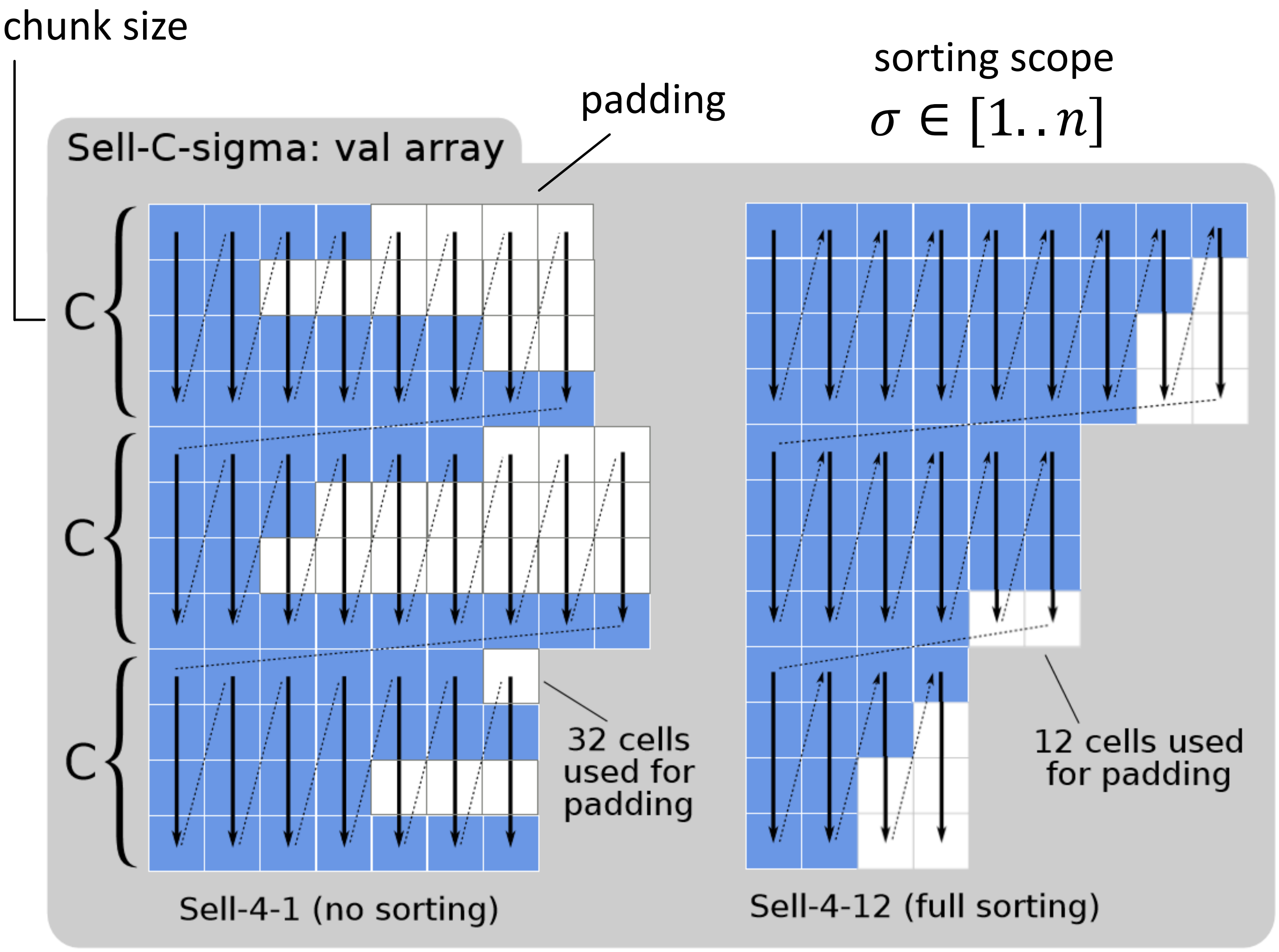


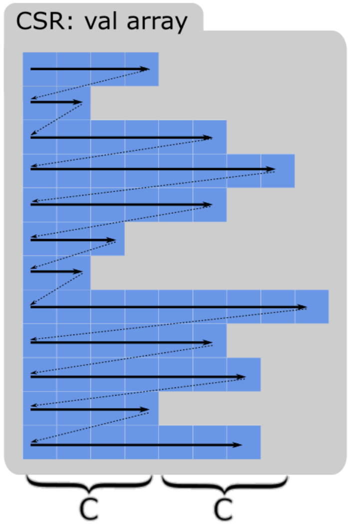
Reductions fast with SIMD operations

M. Besta et al.: "SlimSell: A Vectorized Graph Representation for Breadth-First Search", IPDPS'17

Graph Representations

Sell-C-Sigma





Reductions fast with SIMD operations

Portable

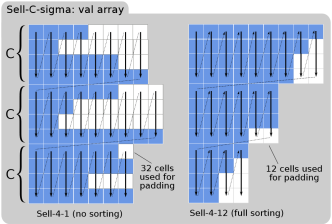
M. Besta et al.: “SlimSell: A Vectorized Graph Representation for Breadth-First Search”, IPDPS’17

SELL-C-SIGMA + SEMIRINGS + (…) = SLIMSELL

FORMULATIONS

SELL-C-SIGMA + SEMIRINGS + (…) = SLIMSELL

FORMULATIONS



+

$(X, op_1, op_2, el_1, el_2)$

$(\mathbb{R} \cup \{\infty\}, min, +, \infty, 0)$

$(\mathbb{R}, +, \cdot, 0, 1)$

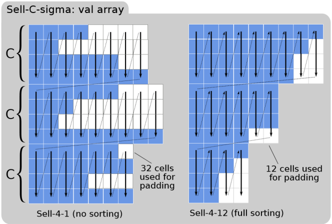
$(\{0,1\}, |, \&, 0, 1)$

$(\mathbb{R}, max, \cdot, -\infty, 1)$

M. Besta et al.: “SlimSell: A Vectorized Graph Representation for Breadth-First Search”, IPDPS’17

SELL-C-SIGMA + SEMIRINGS + (…) = SLIMSELL

FORMULATIONS



+

$(X, op_1, op_2, el_1, el_2)$

$(\mathbb{R} \cup \{\infty\}, min, +, \infty, 0)$

$(\mathbb{R}, +, \cdot, 0, 1)$

$(\{0,1\}, |, \&, 0, 1)$

$(\mathbb{R}, max, \cdot, -\infty, 1)$

```
    // Compute x_k (versions differ based on the used semiring):
#ifdef USE_TROPICAL_SEMIRING
    x = MIN(ADD(rhs, vals), x);
```
TROPICAL SEMIRING

```
#elif defined USE_BOOLEAN_SEMIRING
    x = OR(AND(rhs, vals), x);
```
BOOLEAN SEMIRING

```
#elif defined USE_SELMAX_SEMIRING
    x = MAX(MUL(rhs, vals), x);
```
SEL-MAX SEMIRING

```
#endif
    index += C;
  }
  // Now, derive f_k (versions differ based on the used semiring):
#ifdef USE_TROPICAL_SEMIRING
  STORE(&f_k[i*C], x); // Just a store.
```
TROPICAL SEMIRING

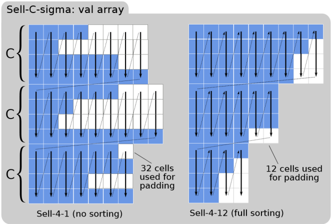
```
#elif defined USE_BOOLEAN_SEMIRING
  // First, derive f_k using filtering.
  V g = LOAD(&g_{k-1}[i*C]); // Load the filter g_{k-1}.
  x = CMP(AND(x, g), [0,0,...0], NEQ); STORE(&x_k[i*C], x);

  // Second, update distances d; depth is the iteration number.
  V x_mask = x; x = MUL(x, [depth,...,depth]);
  x = BLEND(LOAD(&d[i*C]), x, x_mask); STORE(&d[i*C], x);

  // Third, update the filtering term.
  g = AND(NOT(x_mask), g); STORE(&g_k[i*C], g);
```
BOOLEAN SEMIRING

```
#elif defined USE_SELMAX_SEMIRING:
  // Update parents.
  V pars = LOAD(&p_{k-1}[i*C]); // Load the required part of p_{k-1}
  V pnz = CMP(pars, [0,0,...,0], NEQ);
  pars = BLEND([0,0,...,0], pars, pnz); STORE(&p_k[i*C], pars);

  // Set new x_k vector.
  V tmpnz = CMP(x, [0,0,...,0], NEQ);
  x = BLEND(x, &v[i*C], tmpnz); STORE(&x_k[i*C], x);
#endif
```
SEL-MAX SEMIRING

M. Besta et al.: “SlimSell: A Vectorized Graph Representation for Breadth-First Search”, IPDPS’17

SELL-C-SIGMA + SEMIRINGS + (…) = SLIMSELL

FORMULATIONS



+

$(X, op_1, op_2, el_1, el_2)$

$(\mathbb{R} \cup \{\infty\}, min, +, \infty, 0)$

$(\mathbb{R}, +, \cdot, 0, 1)$

$(\{0,1\}, |, \&, 0, 1)$

$(\mathbb{R}, max, \cdot, -\infty, 1)$

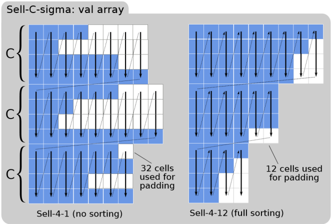
```
    // Compute x_k (versions differ based on the used semiring):
#ifdef USE_TROPICAL_SEMIRING                                TROPICAL SEMIRING
    x = MIN(ADD(rhs, vals), x);
#elif defined USE_BOOLEAN_SEMIRING                          BOOLEAN SEMIRING
    x = OR(AND(rhs, vals), x);
#elif defined USE_SELMAX_SEMIRING                           SEL-MAX SEMIRING
    x = MAX(MUL(rhs, vals), x);
#endif
    index += C;
  }
  // Now, derive f_k (versions differ based on the used semiring):
#ifdef USE_TROPICAL_SEMIRING                                TROPICAL SEMIRING
  STORE(&f_k[i*C], x); // Just a store.
#elif defined USE_BOOLEAN_SEMIRING                          BOOLEAN SEMIRING
  // First, derive f_k using filtering.
  V g = LOAD(&g_{k-1}[i*C]); // Load the filter g_{k-1}.
  x = CMP(AND(x, g), [0,0,...0], NEQ); STORE(&x_k[i*C], x);

  // Second, update distances d; depth is the iteration number.
  V x_mask = x; x = MUL(x, [depth,...,depth]);
  x = BLEND(LOAD(&d[i*C]), x, x_mask); STORE(&d[i*C], x);

  // Third, update the filtering term.
  g = AND(NOT(x_mask), g); STORE(&g_k[i*C], g);
#elif defined USE_SELMAX_SEMIRING:                          SEL-MAX SEMIRING
  // Update parents.
  V pars = LOAD(&p_{k-1}[i*C]); // Load the required part of p_{k-1}
  V pnz = CMP(pars, [0,0,...,0], NEQ);
  pars = BLEND([0,0,...,0], pars, pnz); STORE(&p_k[i*C], pars);

  // Set new x_k vector.
  V tmpnz = CMP(x, [0,0,...,0], NEQ);
  x = BLEND(x, &v[i*C], tmpnz); STORE(&x_k[i*C], x);
#endif
```

! Detailed formulations are in the paper ☺

M. Besta et al.: "SlimSell: A Vectorized Graph Representation for Breadth-First Search", IPDPS'17

Sell-C-Sigma + Semirings + (…) = SlimSell

Formulations



+

$(X, op_1, op_2, el_1, el_2)$

$(\mathbb{R} \cup \{\infty\}, min, +, \infty, 0)$

$(\mathbb{R}, +, \cdot, 0, 1)$

$(\{0,1\}, |, \&, 0, 1)$

$(\mathbb{R}, max, \cdot, -\infty, 1)$

```
    // Compute x_k (versions differ based on the used semiring):
#ifdef USE_TROPICAL_SEMIRING
    x = MIN(ADD(rhs, vals), x);
```

TROPICAL SEMIRING

```
#elif defined USE_BOOLEAN_SEMIRING
    x = OR(AND(rhs, vals), x);
```

BOOLEAN SEMIRING

```
#elif defined USE_SELMAX_SEMIRING
    x = MAX(MUL(rhs, vals), x);
```

SEL-MAX SEMIRING

```
#endif
    index += C;
  }
  // Now, derive f_k (versions differ based on the used semiring):
#ifdef USE_TROPICAL_SEMIRING
  STORE(&f_k[i*C], x); // Just a store.
```

TROPICAL SEMIRING

```
#elif defined USE_BOOLEAN_SEMIRING
  // First, derive f_k using filtering.
  V g = LOAD(&g_{k-1}[i*C]); // Load the filter g_{k-1}.
  x = CMP(AND(x, g), [0,0,...0], NEQ); STORE(&x_k[i*C], x);

  // Second, update distances d; depth is the iteration number.
  V x_mask = x; x = MUL(x, [depth,...,depth]);
  x = BLEND(LOAD(&d[i*C]), x, x_mask); STORE(&d[i*C], x);

  // Third, update the filtering term.
  g = AND(NOT(x_mask), g); STORE(&g_k[i*C], g);
```

BOOLEAN SEMIRING

```
#elif defined USE_SELMAX_SEMIRING:
  // Update parents.
  V pars = LOAD(&p_{k-1}[i*C]); // Load the required part of p_{k-1}
  V pnz = CMP(pars, [0,0,...,0], NEQ);
  pars = BLEND([0,0,...,0], pars, pnz); STORE(&p_k[i*C], pars);

  // Set new x_k vector.
  V tmpnz = CMP(x, [0,0,...,0], NEQ);
  x = BLEND(x, &v[i*C], tmpnz); STORE(&x_k[i*C], x);
#endif
```

SEL-MAX SEMIRING

What vector operations are required for each semiring when using Sell-C-sigma

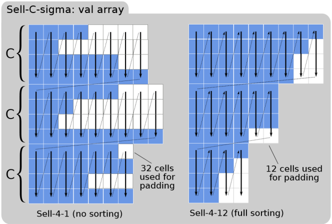
Detailed formulations are in the paper ☺

M. Besta et al.: “SlimSell: A Vectorized Graph Representation for Breadth-First Search”, IPDPS’17

SELL-C-SIGMA + SEMIRINGS + (…) = SLIMSELL

FORMULATIONS

```
   // Compute x_k (versions differ based on the used semiring):
#ifdef USE_TROPICAL_SEMIRING                                   TROPICAL SEMIRING
     x = MIN(ADD(rhs, vals), x);
#elif defined USE_BOOLEAN_SEMIRING                             BOOLEAN SEMIRING
     x = OR(AND(rhs, vals), x);
#elif defined USE_SELMAX_SEMIRING                              SEL-MAX SEMIRING
     x = MAX(MUL(rhs, vals), x);
#endif
     index += C;
  }
  // Now, derive f_k (versions differ based on the used semiring):
#ifdef USE_TROPICAL_SEMIRING                                   TROPICAL SEMIRING
  STORE(&f_k[i*C], x); // Just a store.
#elif defined USE_BOOLEAN_SEMIRING                             BOOLEAN SEMIRING
  // First, derive f_k using filtering.
  V g = LOAD(&g_{k-1}[i*C]); // Load the filter g_{k-1}.
  x = CMP(AND(x, g), [0,0,...0], NEQ); STORE(&x_k[i*C], x);

  // Second, update distances d; depth is the iteration number.
  V x_mask = x; x = MUL(x, [depth,...,depth]);
  x = BLEND(LOAD(&d[i*C]), x, x_mask); STORE(&d[i*C], x);

  // Third, update the filtering term.
  g = AND(NOT(x_mask), g); STORE(&g_k[i*C], g);
#elif defined USE_SELMAX_SEMIRING:                             SEL-MAX SEMIRING
  // Update parents.
  V pars = LOAD(&p_{k-1}[i*C]); // Load the required part of p_{k-1}
  V pnz = CMP(pars, [0,0,...,0], NEQ);
  pars = BLEND([0,0,...,0], pars, pnz); STORE(&p_k[i*C], pars);

  // Set new x_k vector.
  V tmpnz = CMP(x, [0,0,...,0], NEQ);
  x = BLEND(x, &v[i*C], tmpnz); STORE(&x_k[i*C], x);
#endif
```



+

$(X, op_1, op_2, el_1, el_2)$

$(\mathbb{R} \cup \{\infty\}, min, +, \infty, 0)$

$(\mathbb{R}, +, \cdot, 0, 1)$

$(\{0,1\}, |, \&, 0, 1)$

$(\mathbb{R}, max, \cdot, -\infty, 1)$

! What vector operations are required for each semiring when using Sell-C-sigma

! Detailed formulations are in the paper ☺

M. Besta et al.: "SlimSell: A Vectorized Graph Representation for Breadth-First Search", IPDPS'17

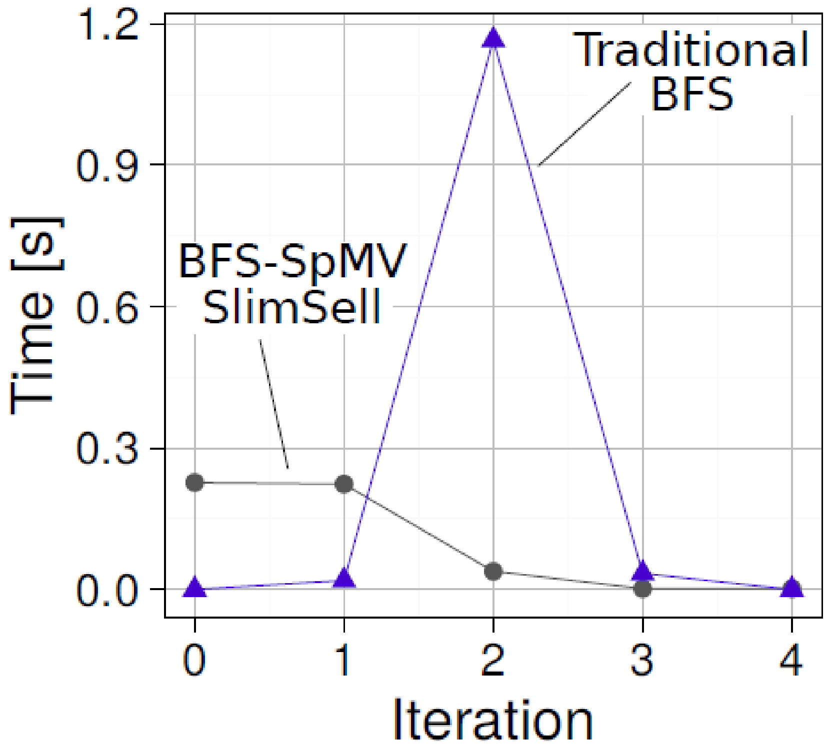
PERFORMANCE ANALYSIS
COMPARISON TO GRAPH500

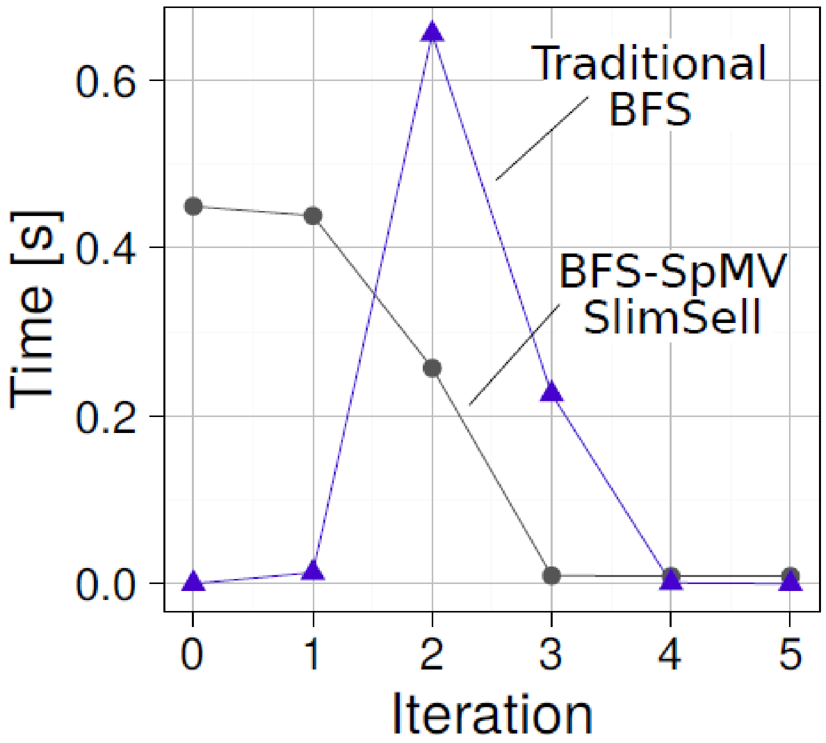
Kronecker power-law graphs

Intel KNL, $C = 16$

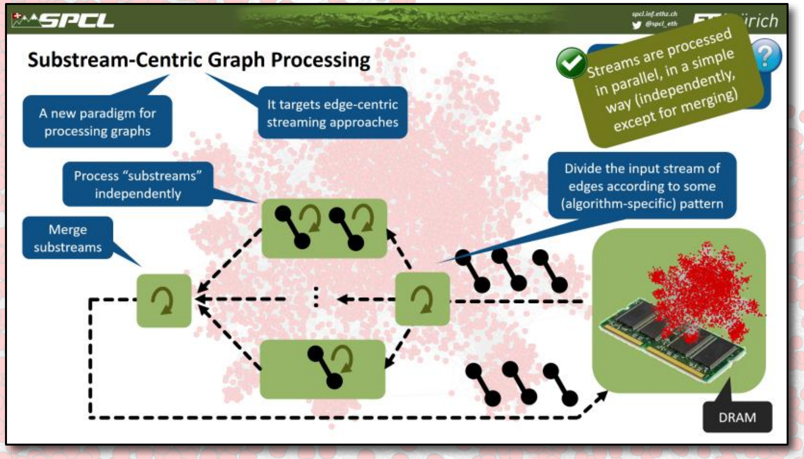
$\log \sigma \in \{20,21,22\}$

Dynamic scheduling



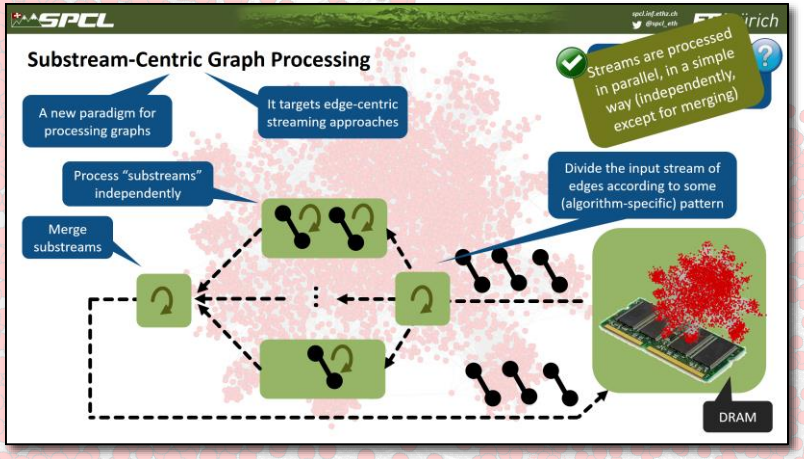


M. Besta et al.: “SlimSell: A Vectorized Graph Representation for Breadth-First Search”, IPDPS’17



SUBSTREAM-CENTRIC GRAPH PROCESSING PARADIGM:

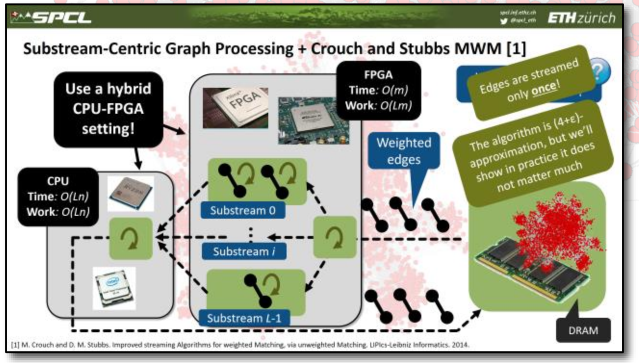
EXPOSES PARALLELISM, ENABLES EASY PIPELINING, SUPPORTS APPROXIMATION

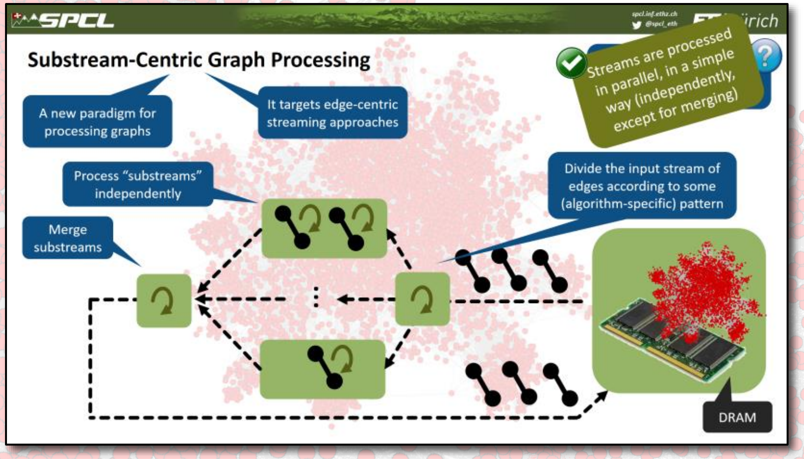


THEORY-INSPIRED MWM APPROXIMATE ALGORITHM ON A HYBRID CPU-FPGA SETTING

SUBSTREAM-CENTRIC GRAPH PROCESSING PARADIGM:

EXPOSES PARALLELISM, ENABLES EASY PIPELINING, SUPPORTS APPROXIMATION

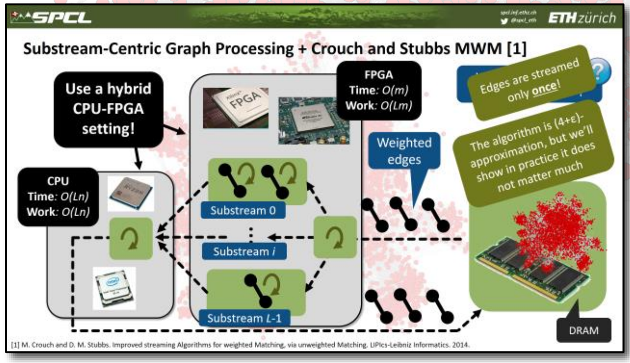


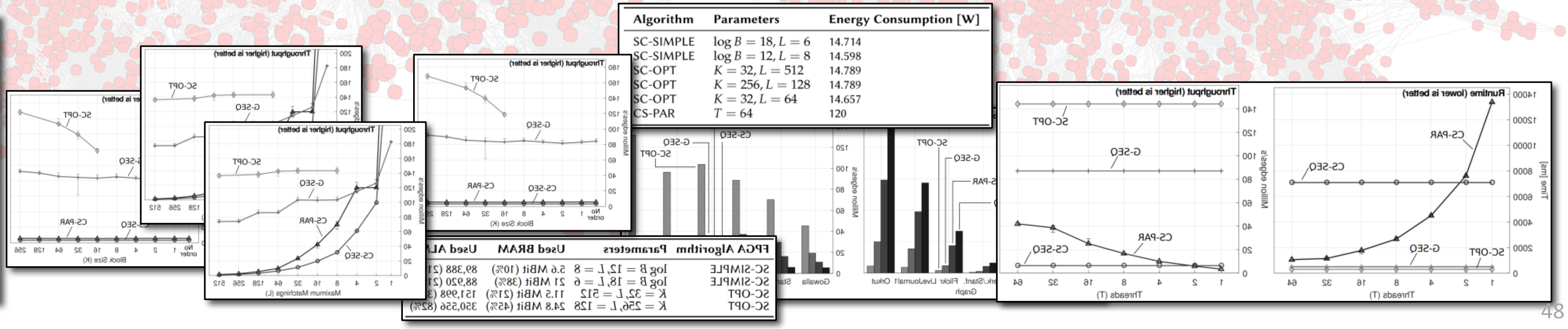


THEORY-INSPIRED MWM APPROXIMATE ALGORITHM ON A HYBRID CPU-FPGA SETTING

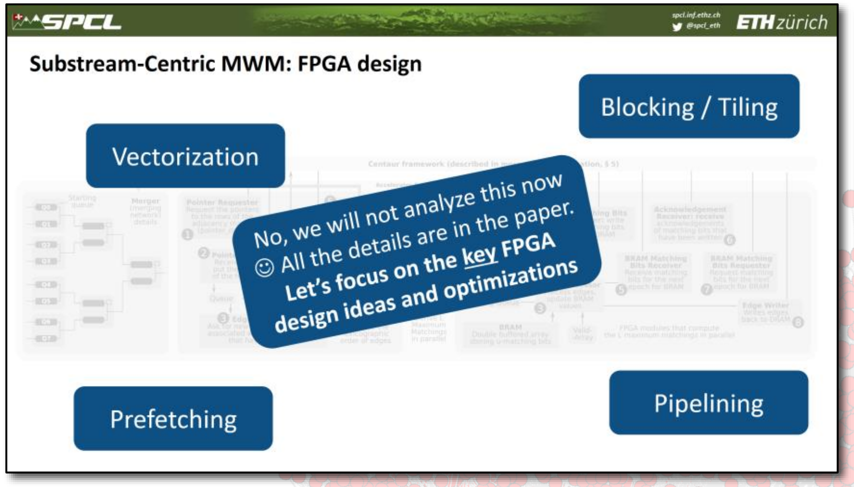
SUBSTREAM-CENTRIC GRAPH PROCESSING PARADIGM:

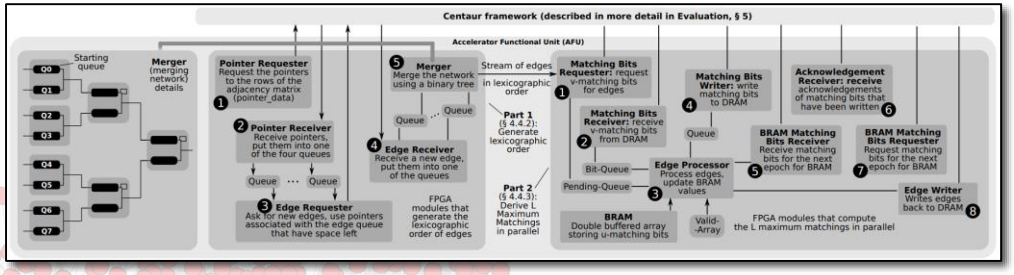
EXPOSES PARALLELISM, ENABLES EASY PIPELINING, SUPPORTS APPROXIMATION



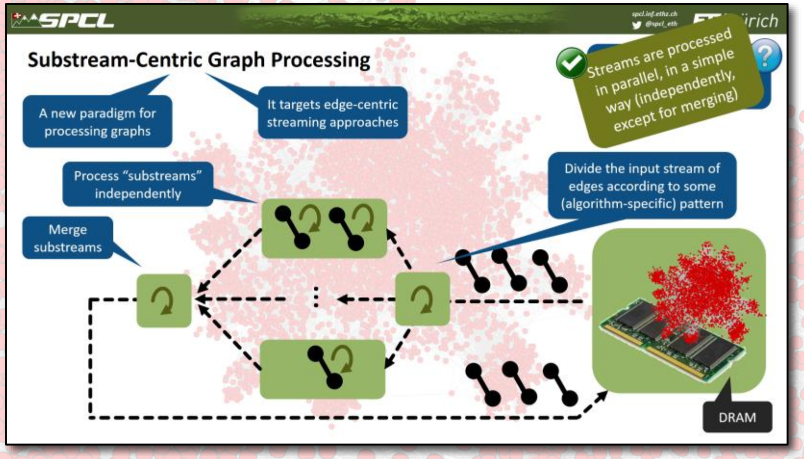


Algorithm	Parameters	Energy Consumption [W]
SC-SIMPLE	$\log B = 18, L = 6$	14.714
SC-SIMPLE	$\log B = 12, L = 8$	14.598
SC-OPT	$K = 32, L = 512$	14.789
SC-OPT	$K = 256, L = 128$	14.789
SC-OPT	$K = 32, L = 64$	14.657
CS-PAR	$T = 64$	120





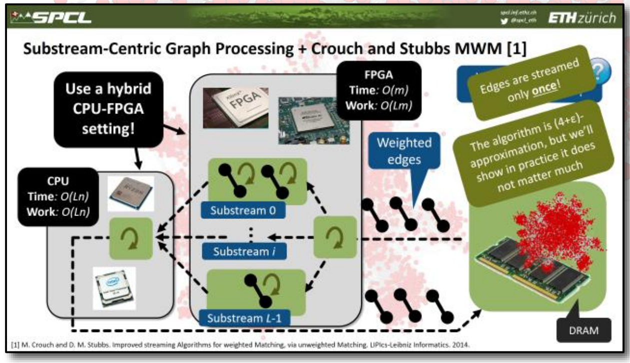
GENERIC FPGA DESIGN, CODE AVAILABLE



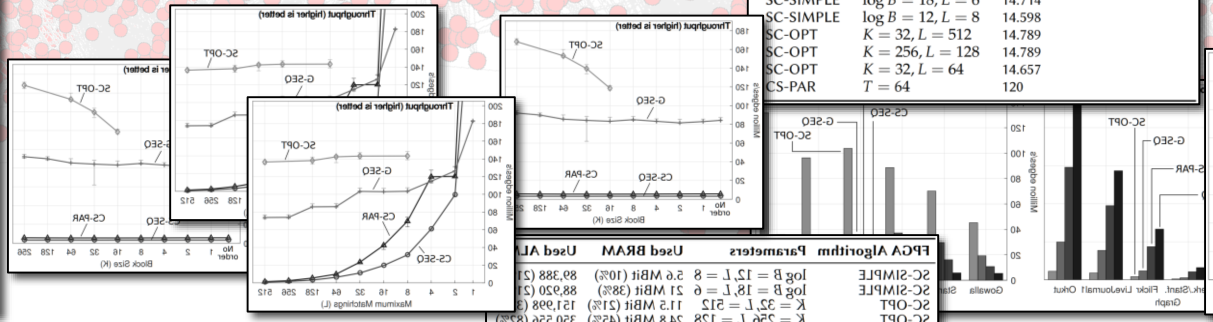
THEORY-INSPIRED MWM APPROXIMATE ALGORITHM ON A HYBRID CPU-FPGA SETTING

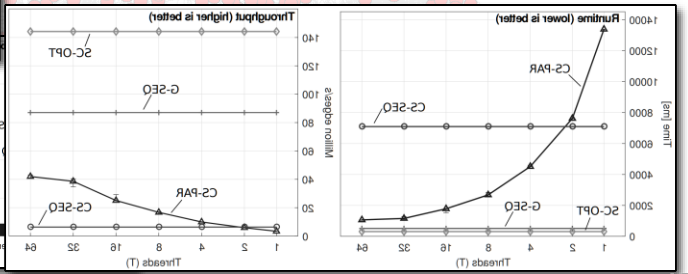
SUBSTREAM-CENTRIC GRAPH PROCESSING PARADIGM:

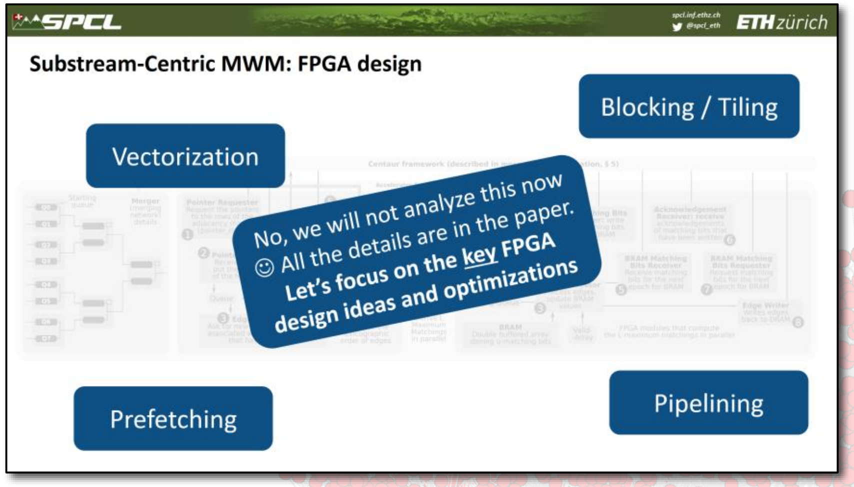
EXPOSES PARALLELISM, ENABLES EASY PIPELINING, SUPPORTS APPROXIMATION



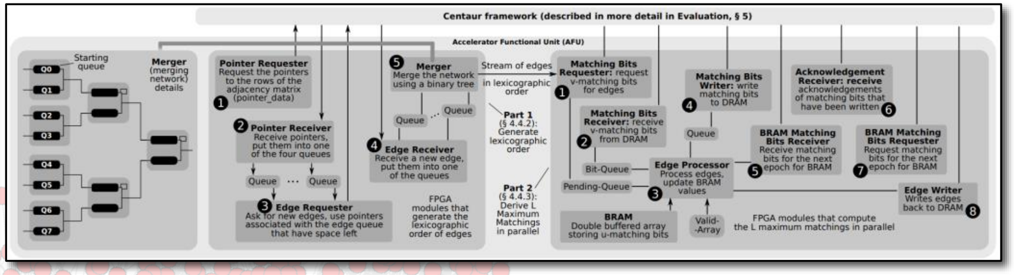
Algorithm	Parameters	Energy Consumption [W]
SC-SIMPLE	$\log B = 18, L = 6$	14.714
SC-SIMPLE	$\log B = 12, L = 8$	14.598
SC-OPT	$K = 32, L = 512$	14.789
SC-OPT	$K = 256, L = 128$	14.789
SC-OPT	$K = 32, L = 64$	14.657
CS-PAR	$T = 64$	120

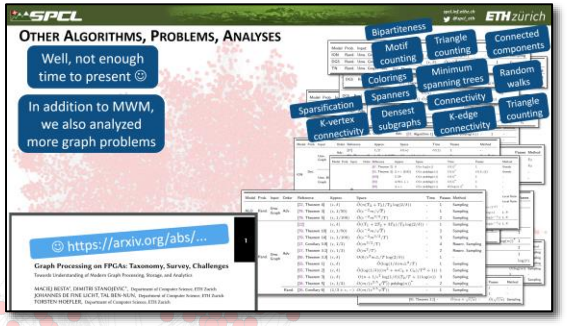




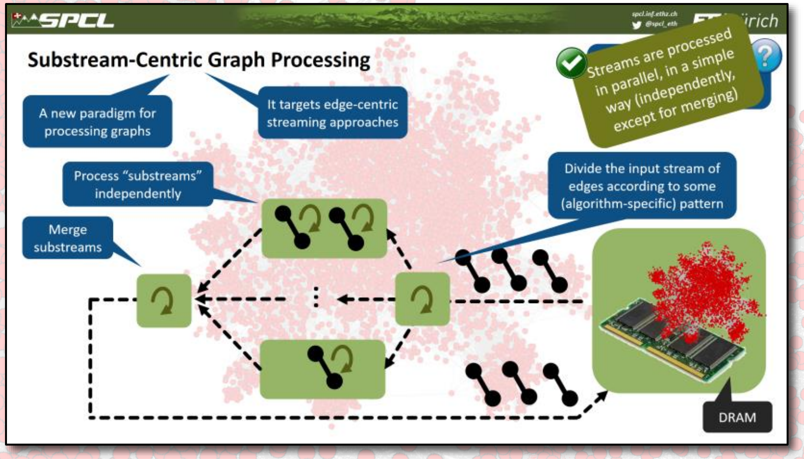


GENERIC FPGA DESIGN, CODE AVAILABLE





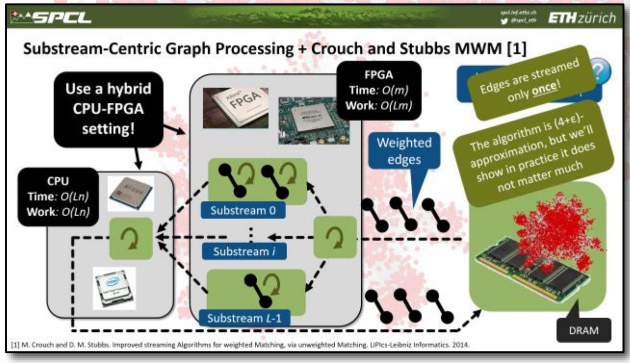
GENERALIZABILITY TO OTHER GRAPH PROBLEMS AND SETTINGS

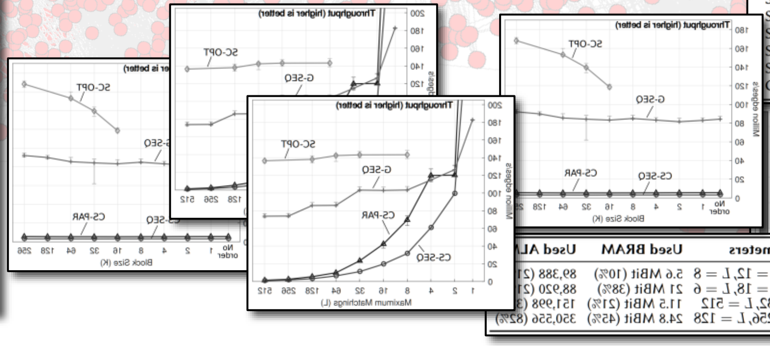


THEORY-INSPIRED MWM APPROXIMATE ALGORITHM ON A HYBRID CPU-FPGA SETTING

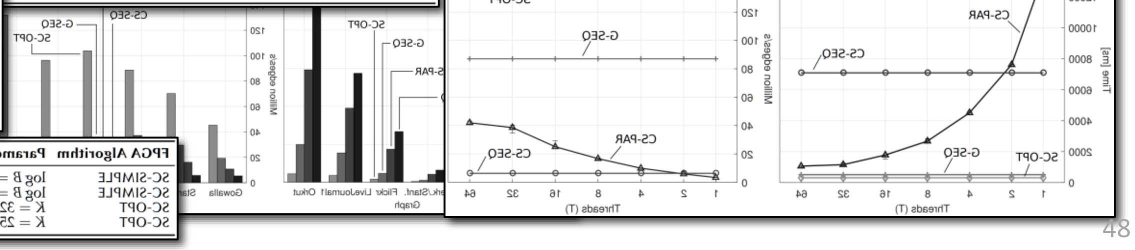
SUBSTREAM-CENTRIC GRAPH PROCESSING PARADIGM:

EXPOSES PARALLELISM, ENABLES EASY PIPELINING, SUPPORTS APPROXIMATION

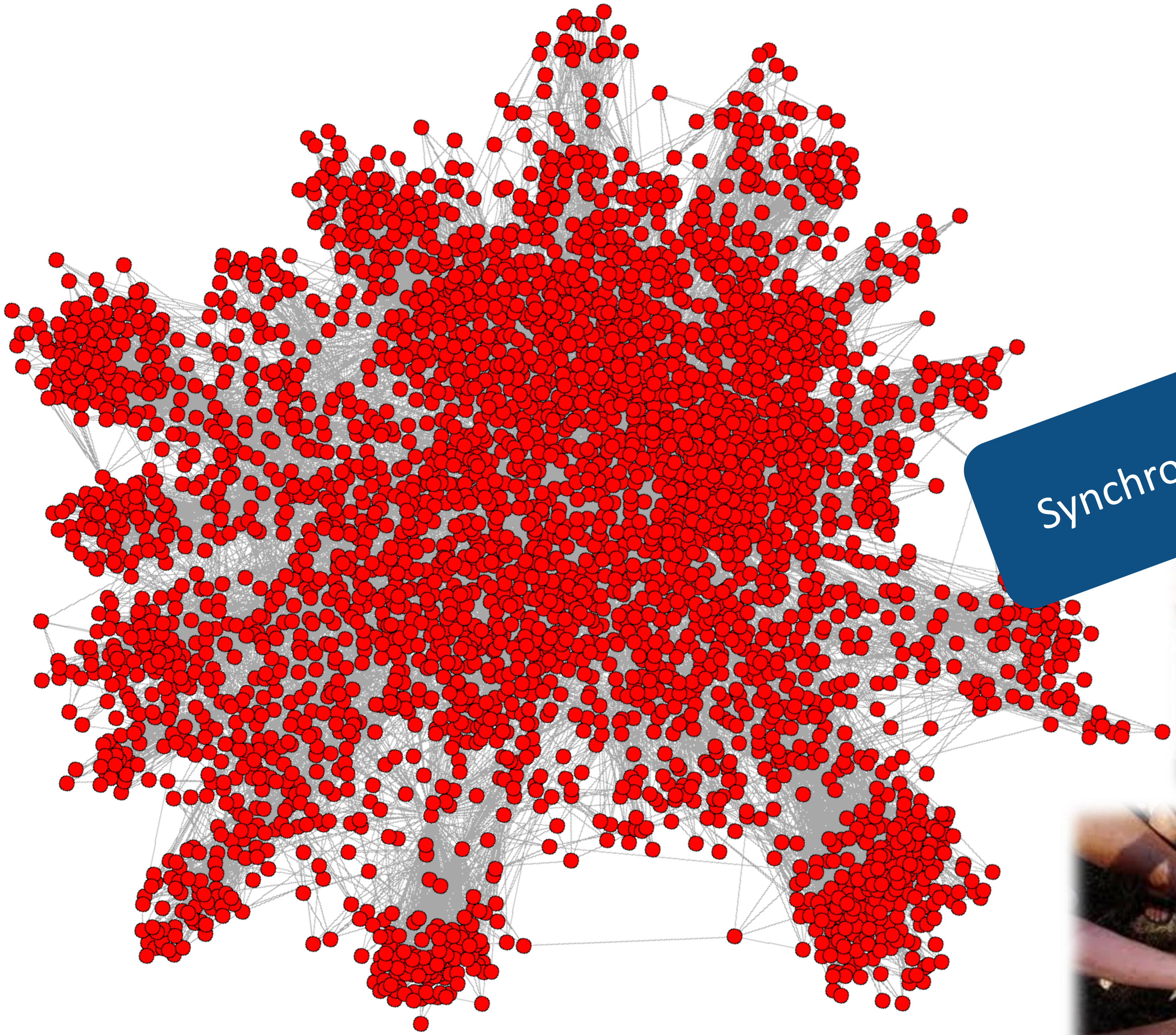




Algorithm	Parameters	Energy Consumption [W]
SC-SIMPLE	$\log B = 18, L = 6$	14.714
SC-SIMPLE	$\log B = 12, L = 8$	14.598
SC-OPT	$K = 32, L = 512$	14.789
SC-OPT	$K = 256, L = 128$	14.789
SC-OPT	$K = 32, L = 64$	14.657
CS-PAR	$T = 64$	120

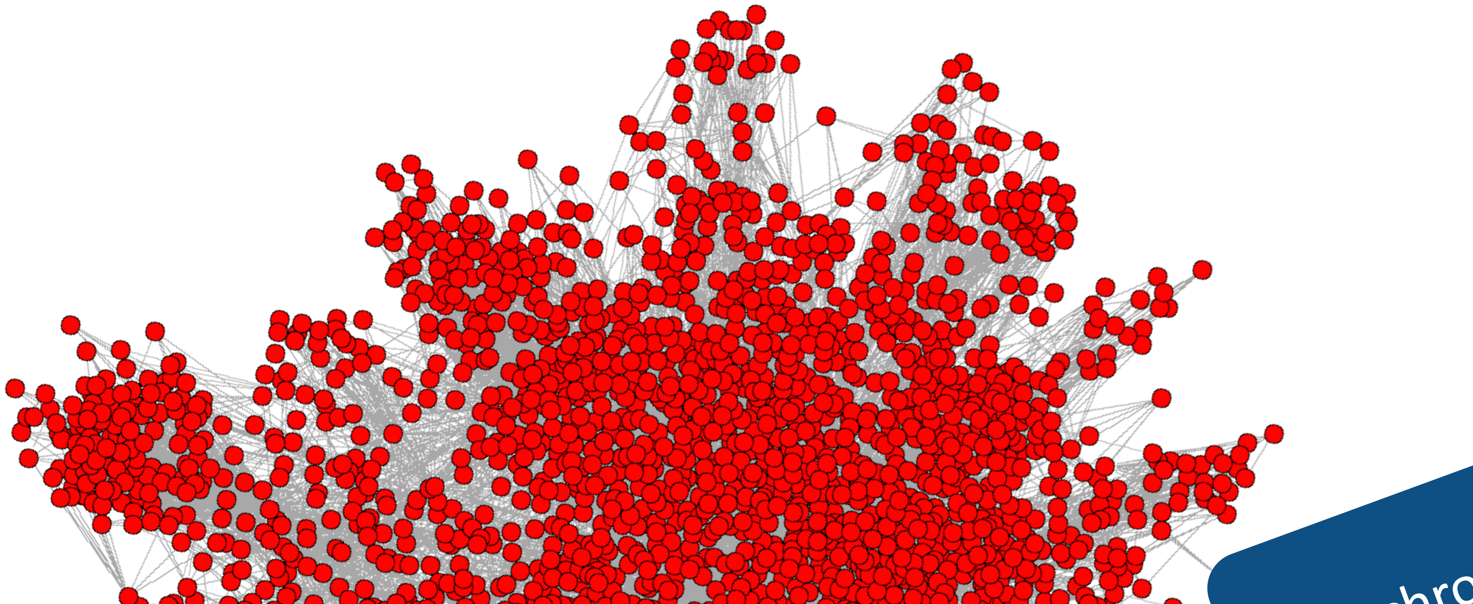


Problems!

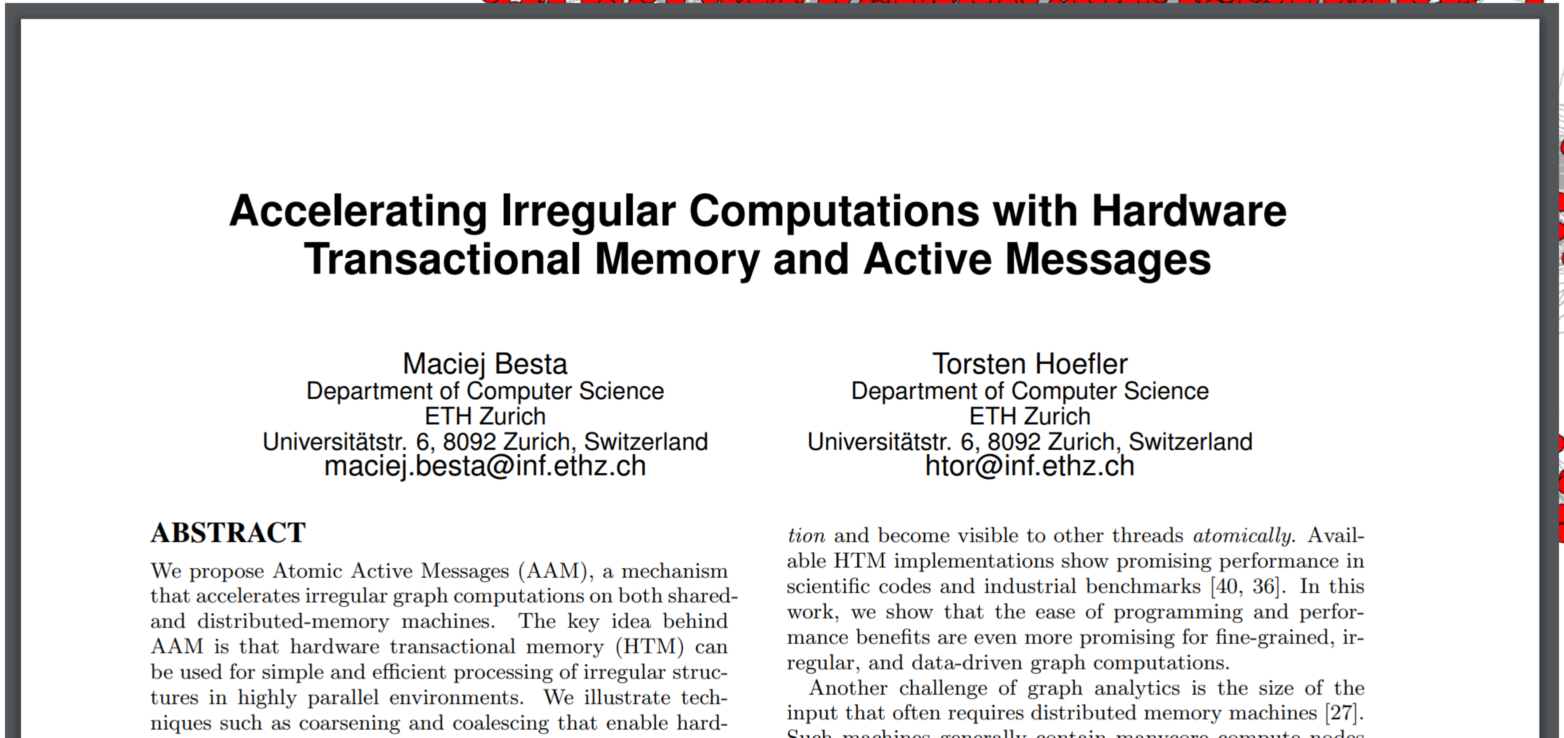


Synchronization-heavy

Problems!



Synchronization-heavy



Accelerating Irregular Computations with Hardware Transactional Memory and Active Messages

Maciej Besta
Department of Computer Science
ETH Zurich
Universitätstr. 6, 8092 Zurich, Switzerland
maciej.besta@inf.ethz.ch

Torsten Hoefler
Department of Computer Science
ETH Zurich
Universitätstr. 6, 8092 Zurich, Switzerland
htor@inf.ethz.ch

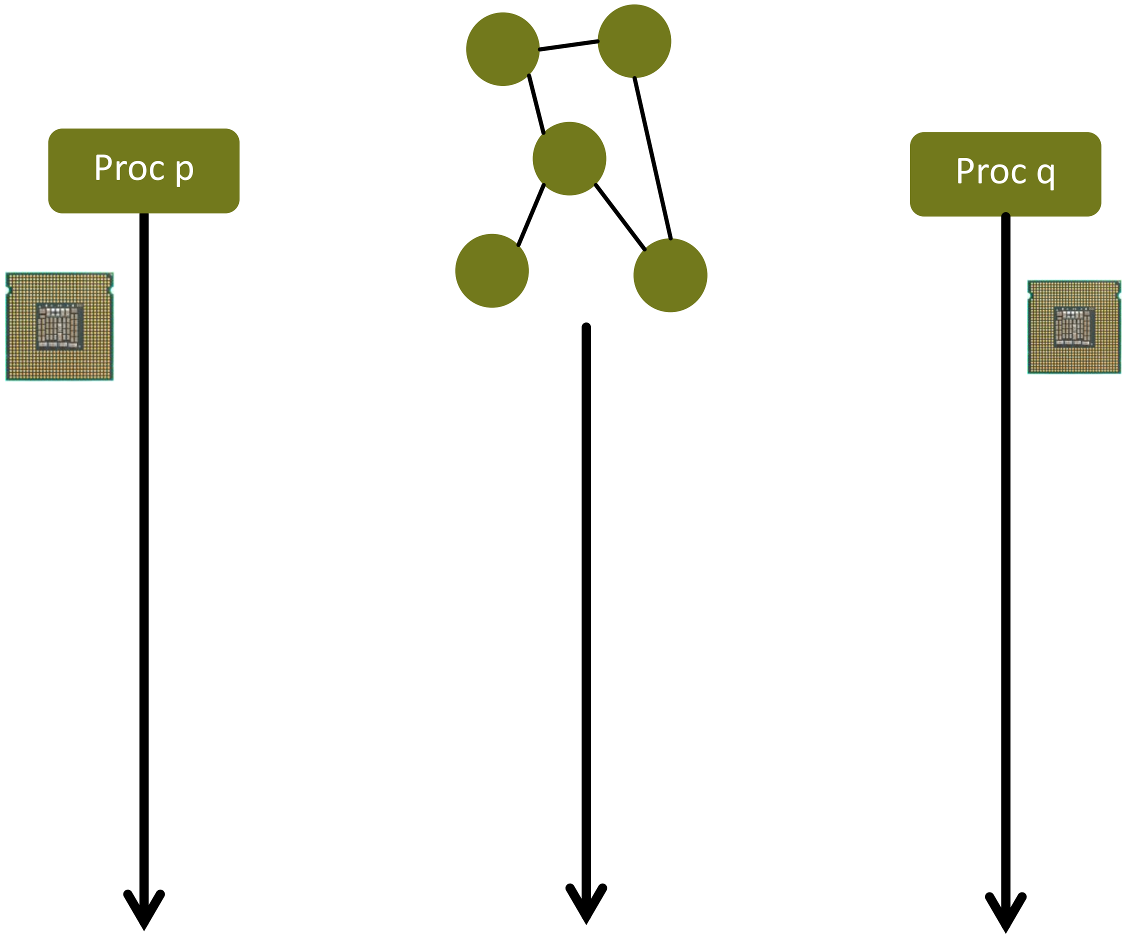
ABSTRACT

We propose Atomic Active Messages (AAM), a mechanism that accelerates irregular graph computations on both shared- and distributed-memory machines. The key idea behind AAM is that hardware transactional memory (HTM) can be used for simple and efficient processing of irregular structures in highly parallel environments. We illustrate techniques such as coarsening and coalescing that enable hard-

tion and become visible to other threads *atomically*. Available HTM implementations show promising performance in scientific codes and industrial benchmarks [40, 36]. In this work, we show that the ease of programming and performance benefits are even more promising for fine-grained, irregular, and data-driven graph computations.

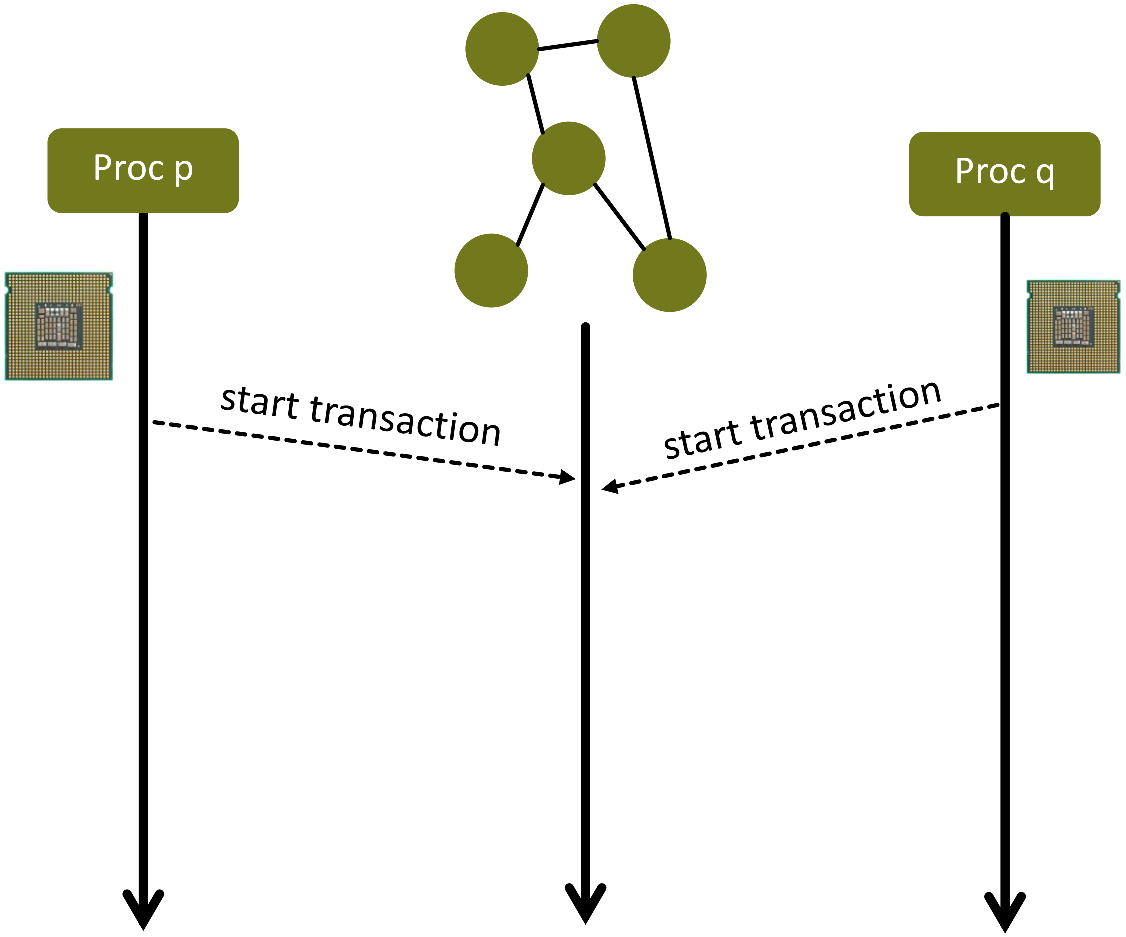
Another challenge of graph analytics is the size of the input that often requires distributed memory machines [27].

TRANSACTIONAL MEMORY [1]



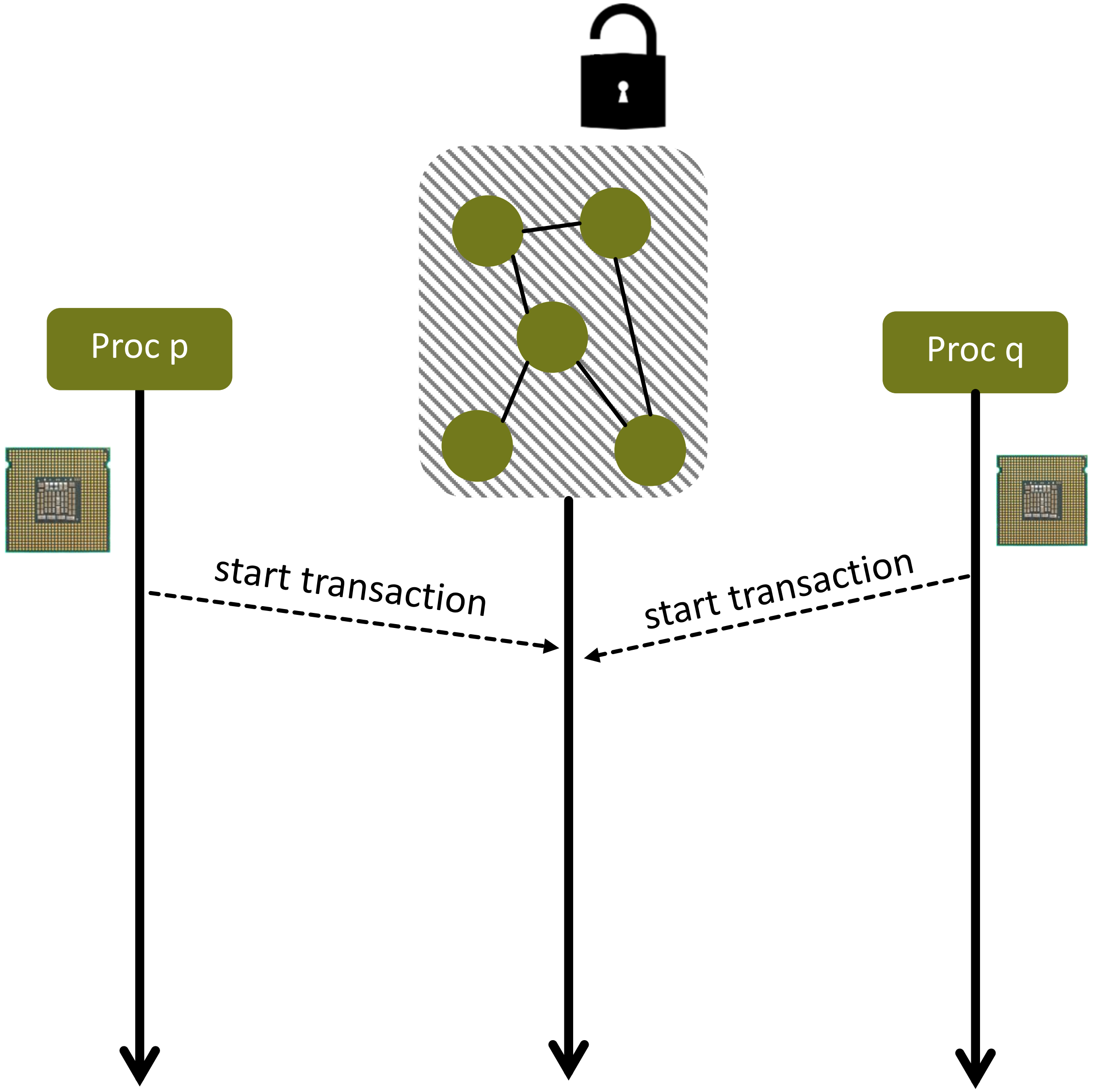
[1] N. Shavit and D. Touitou. Software transactional memory. PODC'95.

Transactional Memory [1]



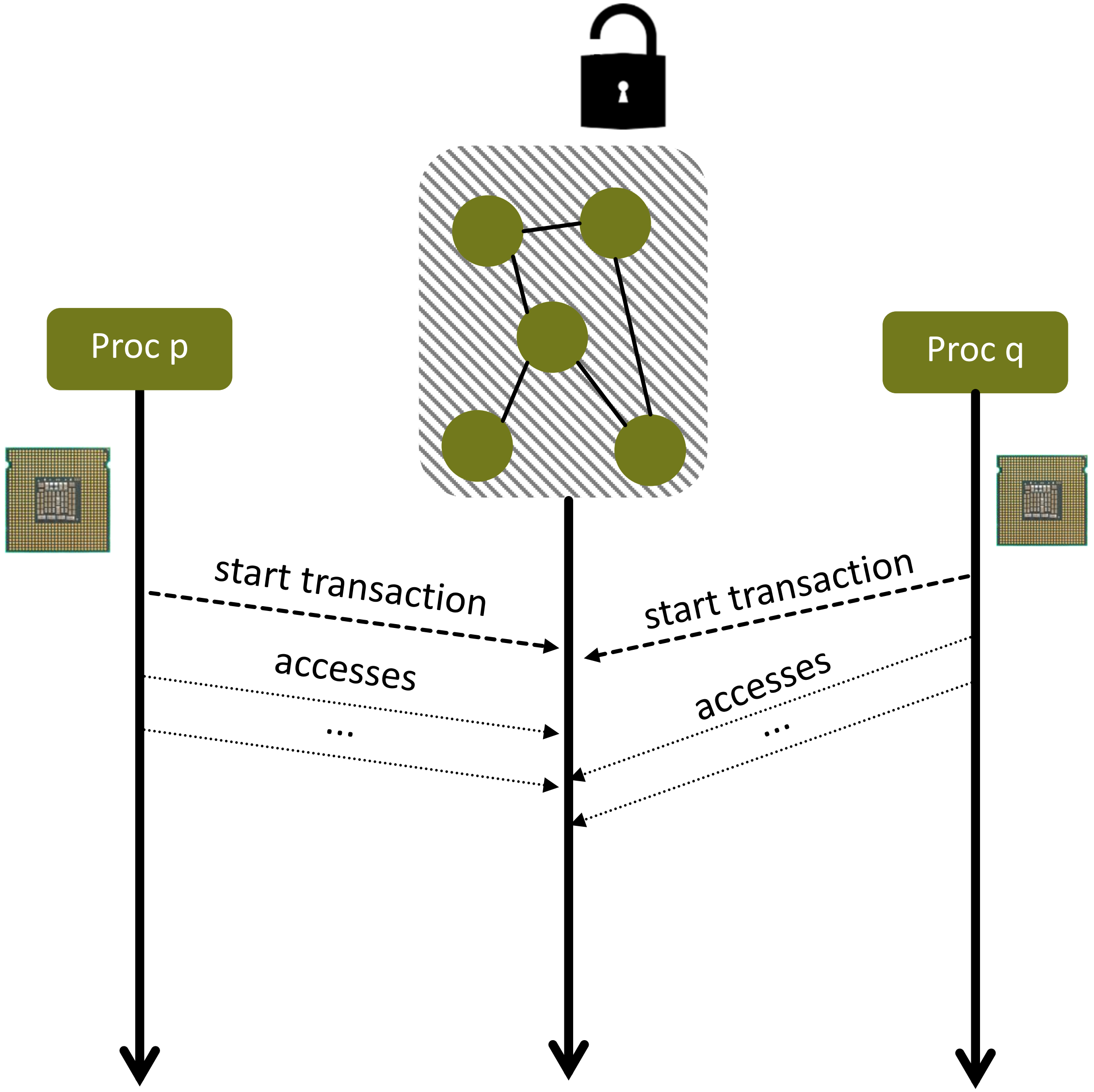
[1] N. Shavit and D. Touitou. Software transactional memory. PODC'95.

Transactional Memory [1]



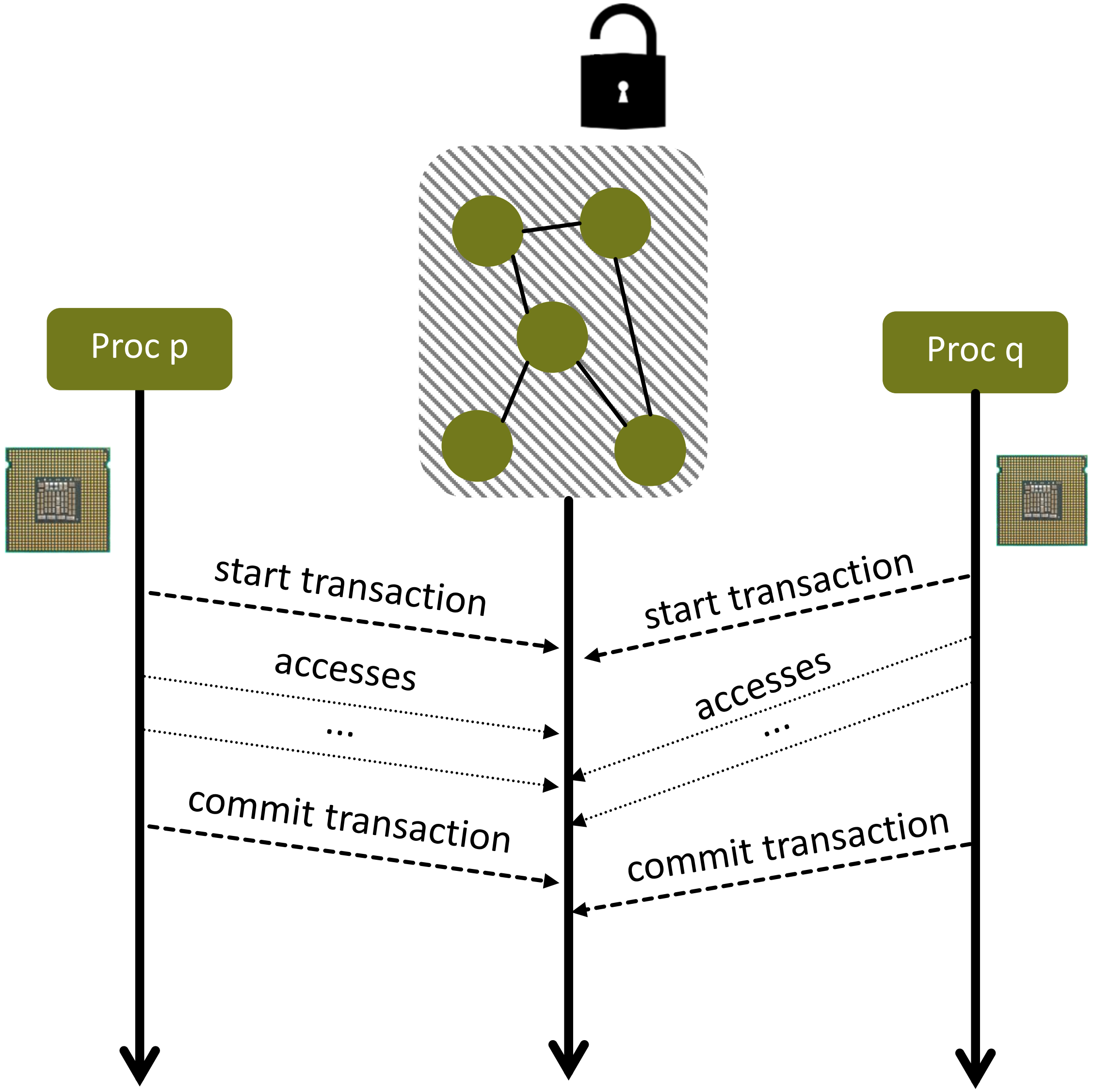
[1] N. Shavit and D. Touitou. Software transactional memory. PODC'95.

Transactional Memory [1]



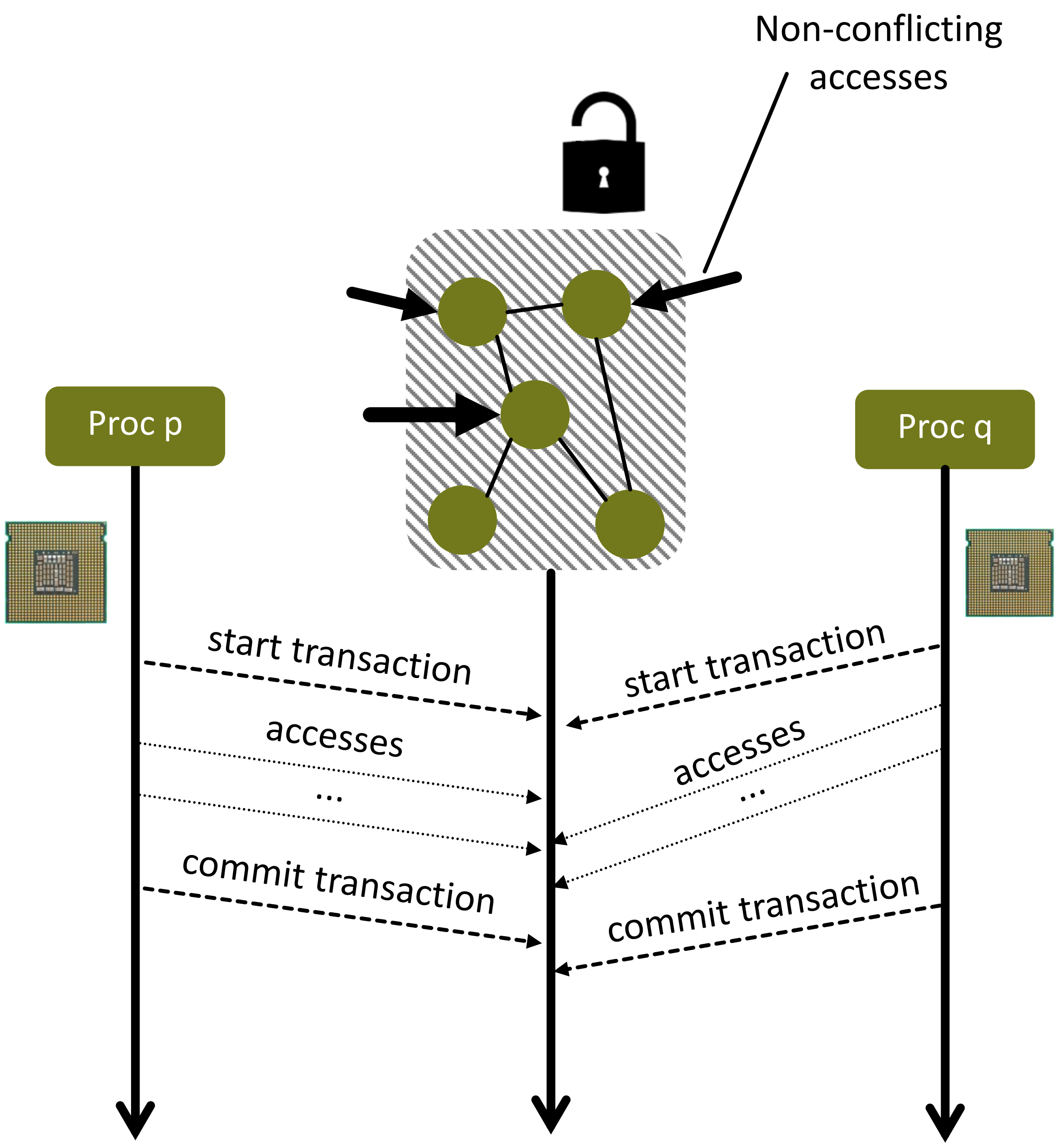
[1] N. Shavit and D. Touitou. Software transactional memory. PODC'95.

Transactional Memory [1]



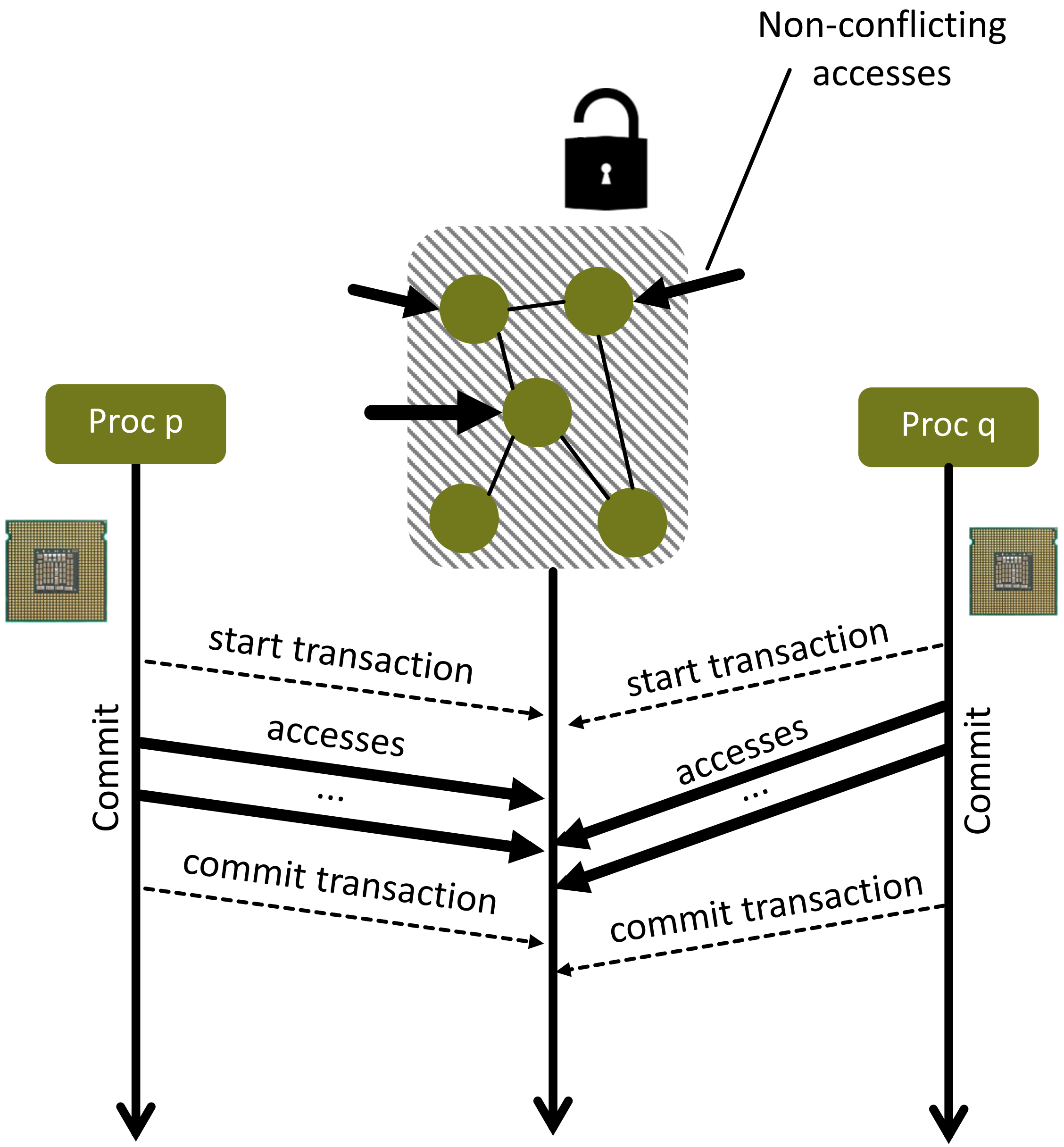
[1] N. Shavit and D. Touitou. Software transactional memory. PODC'95.

Transactional Memory [1]



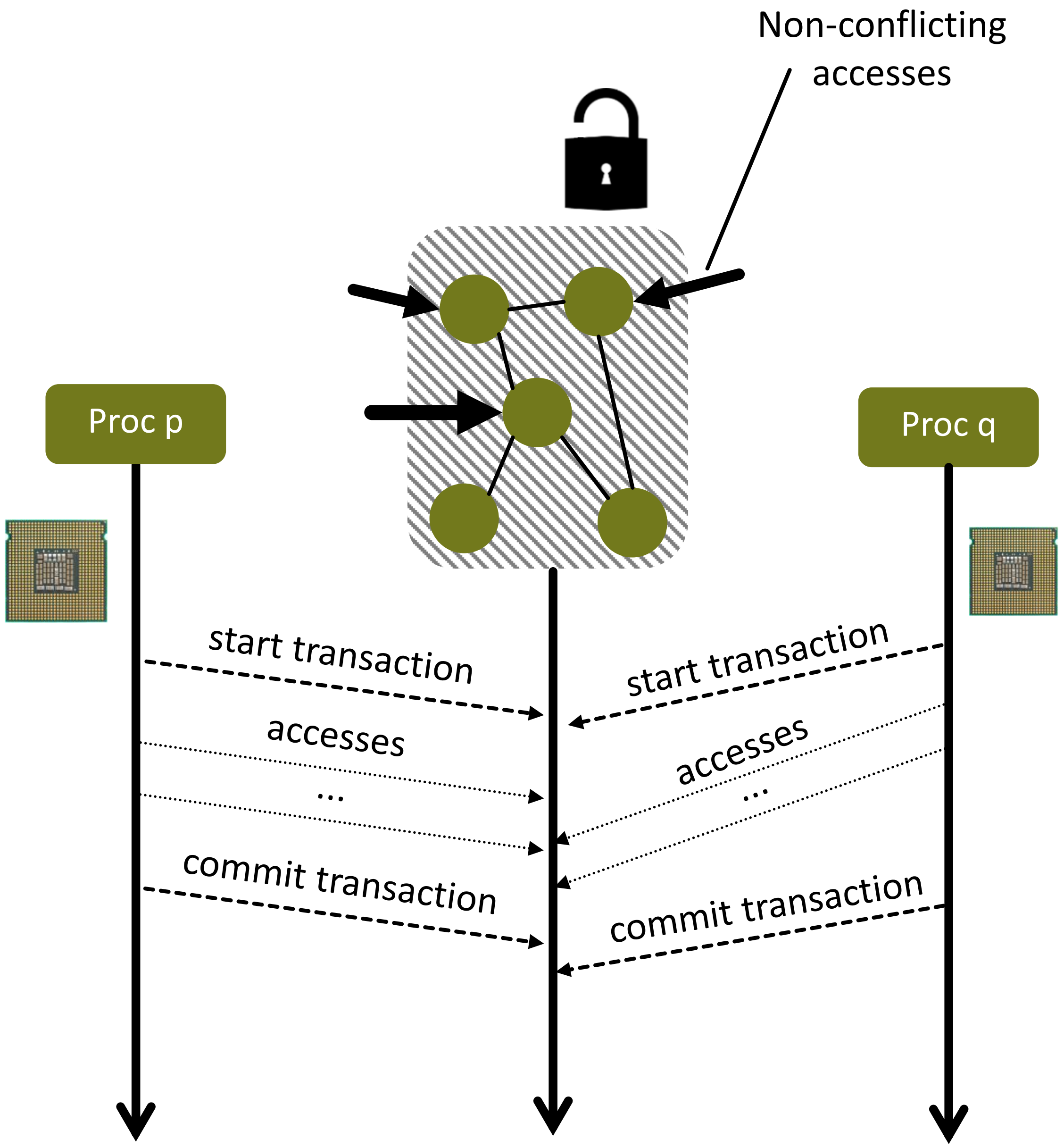
[1] N. Shavit and D. Touitou. Software transactional memory. PODC'95.

TRANSACTIONAL MEMORY [1]



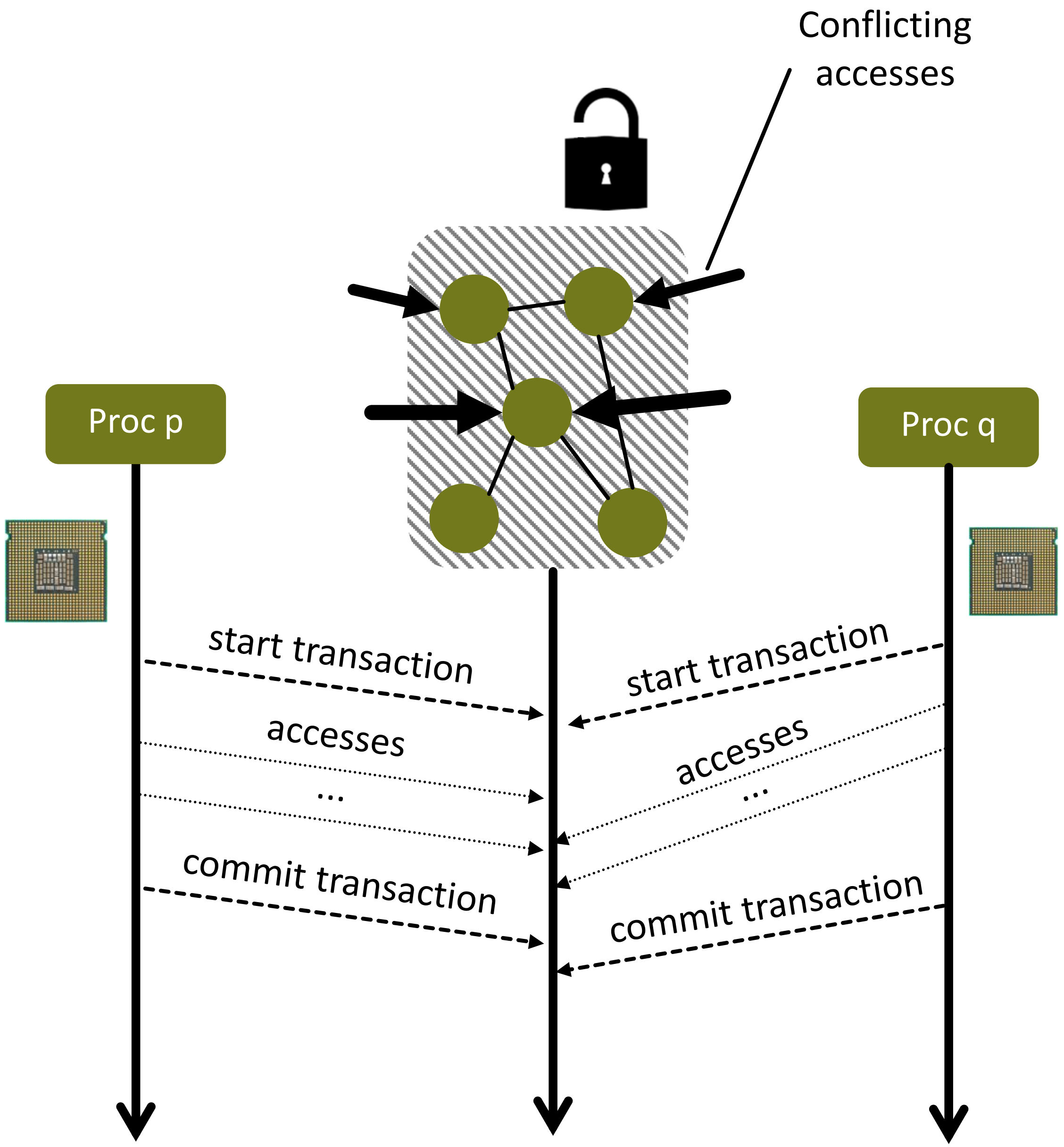
[1] N. Shavit and D. Touitou. Software transactional memory. PODC'95.

TRANSACTIONAL MEMORY [1]



[1] N. Shavit and D. Touitou. Software transactional memory. PODC'95.

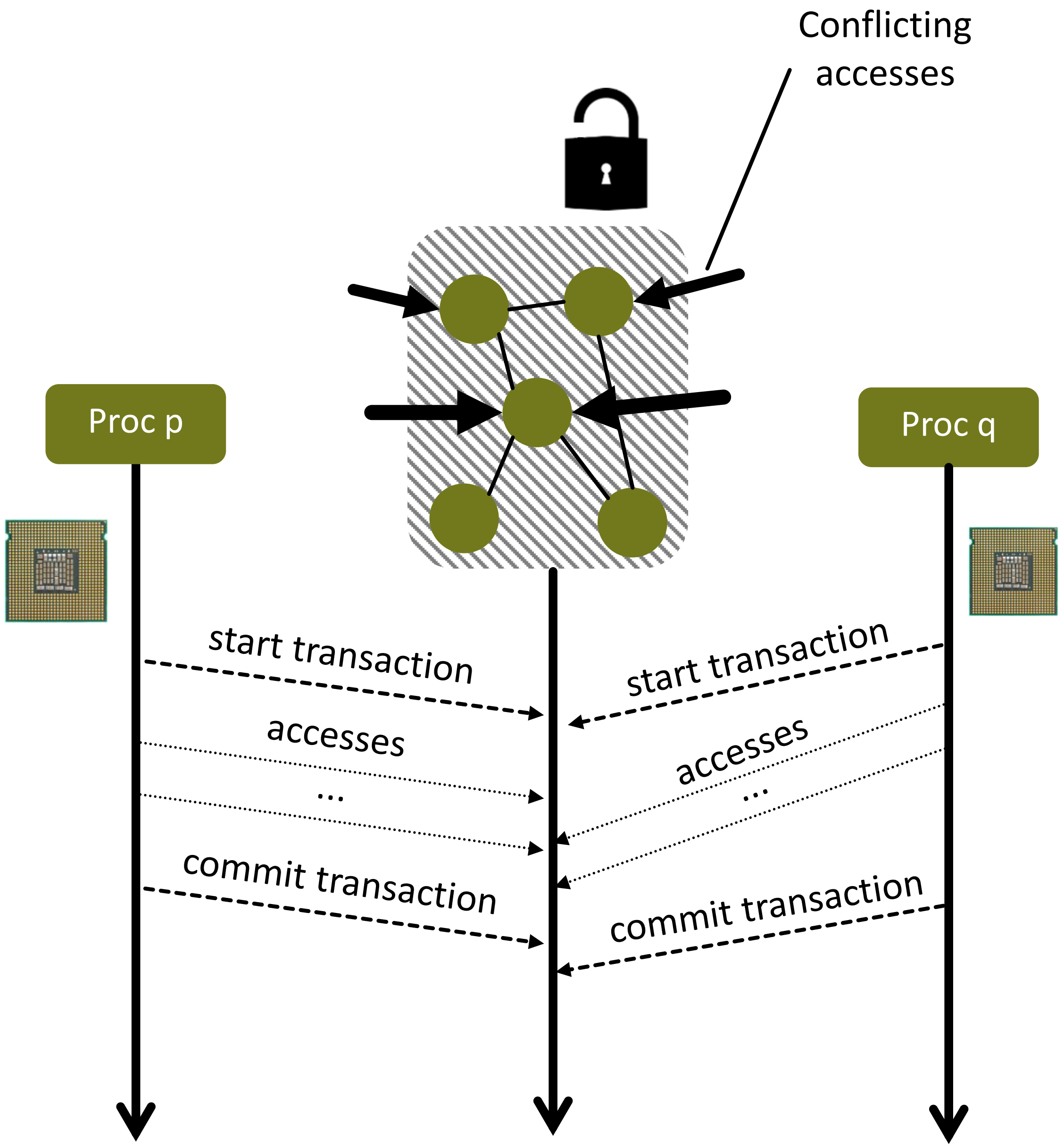
TRANSACTIONAL MEMORY [1]



[1] N. Shavit and D. Touitou. Software transactional memory. PODC’95.

Transactional Memory [1]

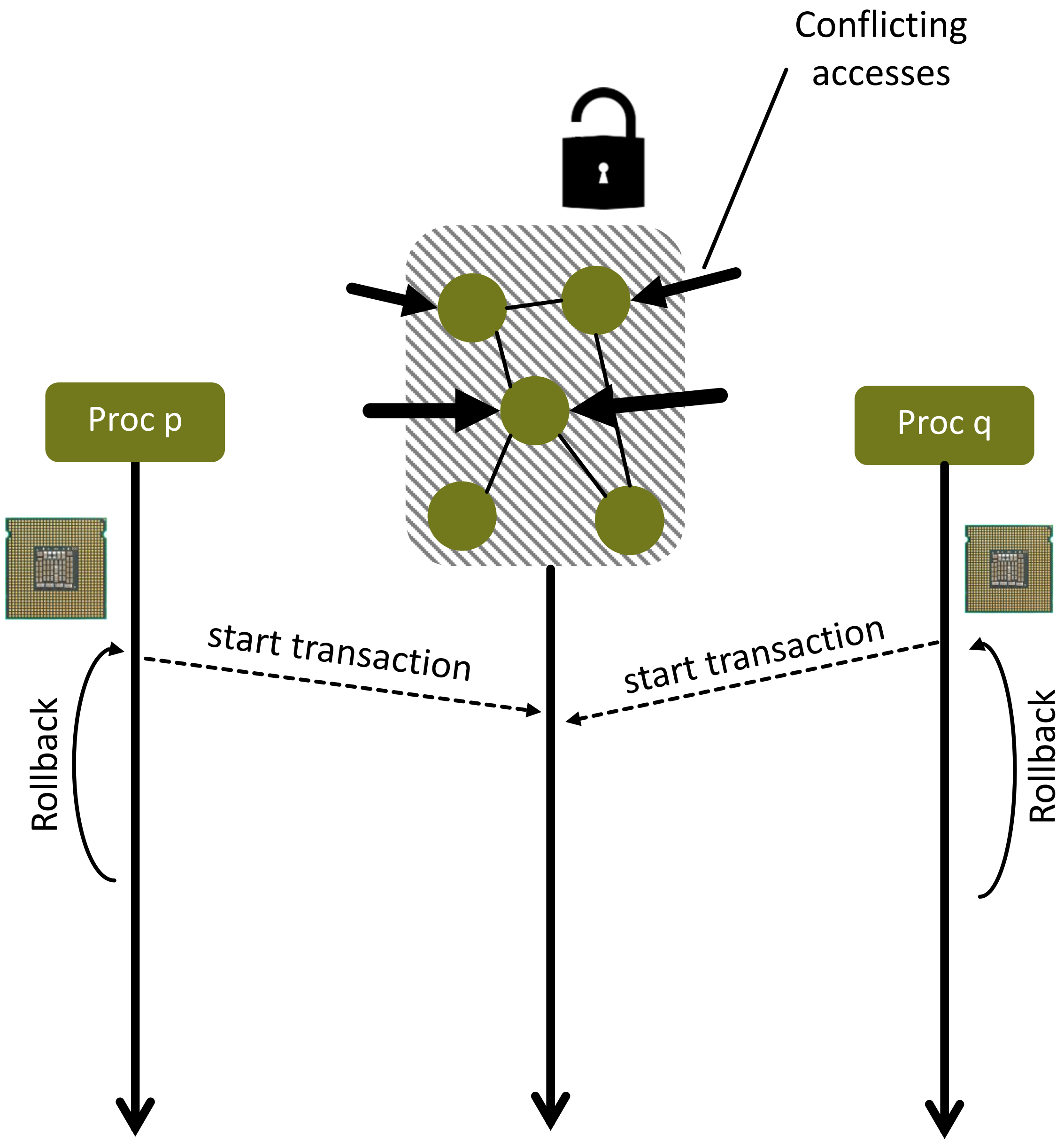
Conflicts solved with rollbacks and/or serialization.



[1] N. Shavit and D. Touitou. Software transactional memory. PODC'95.

TRANSACTIONAL MEMORY [1]

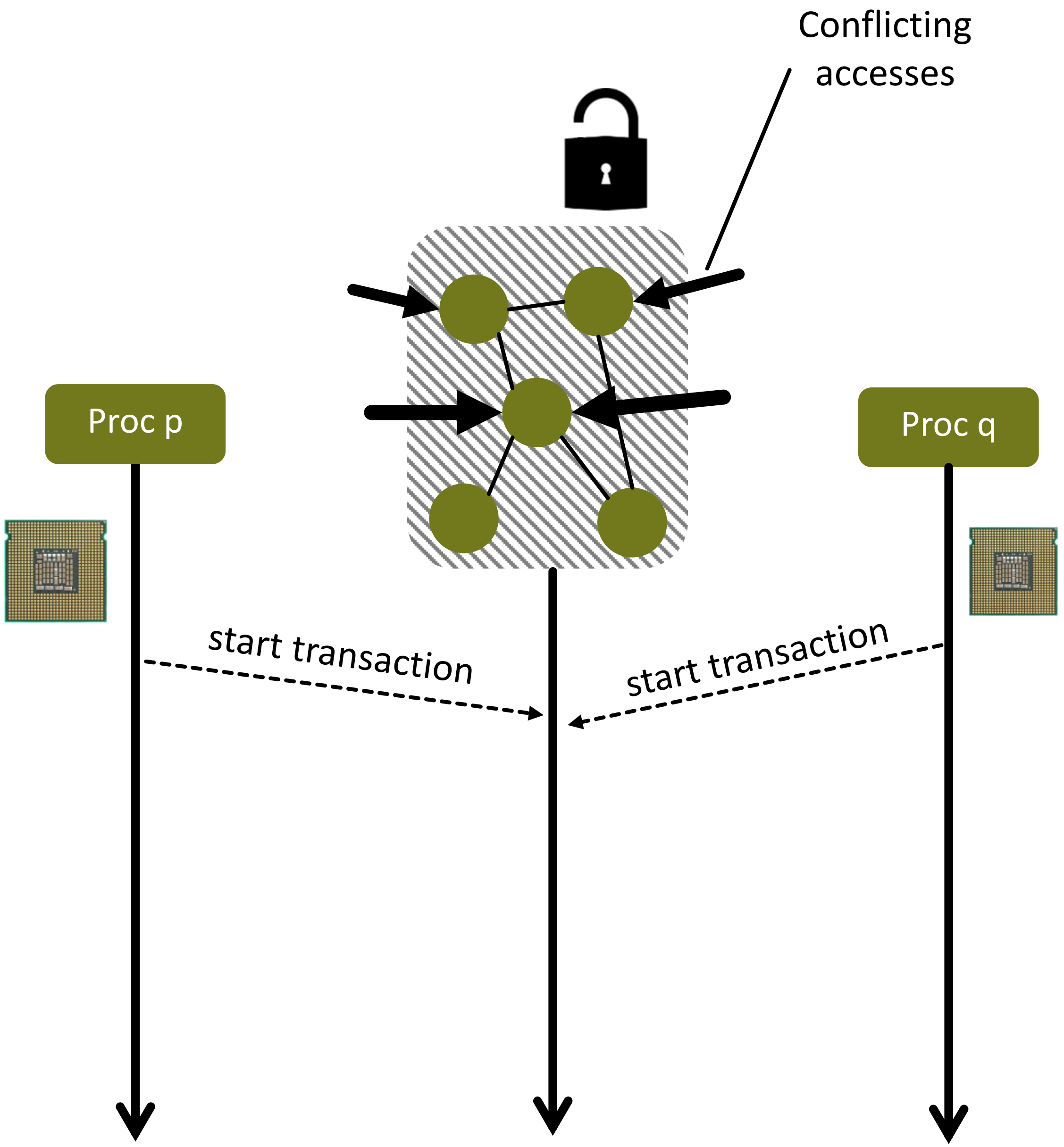
Conflicts solved with rollbacks and/or serialization.



[1] N. Shavit and D. Touitou. Software transactional memory. PODC'95.

TRANSACTIONAL MEMORY [1]

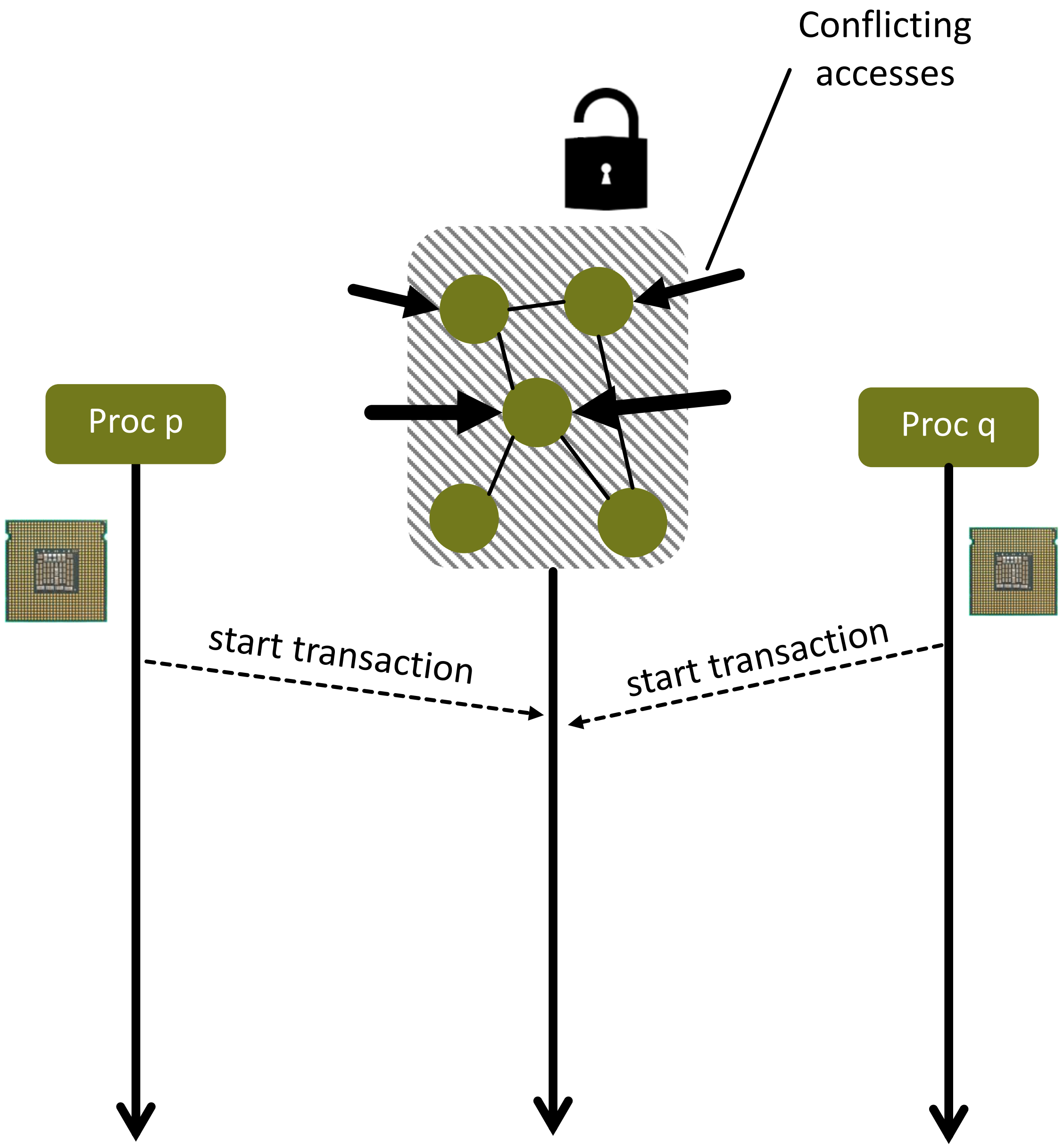
Conflicts solved with rollbacks and/or serialization.



[1] N. Shavit and D. Touitou. Software transactional memory. PODC'95.

Transactional Memory [1]

Conflicts solved with rollbacks and/or serialization.



Simple protocols

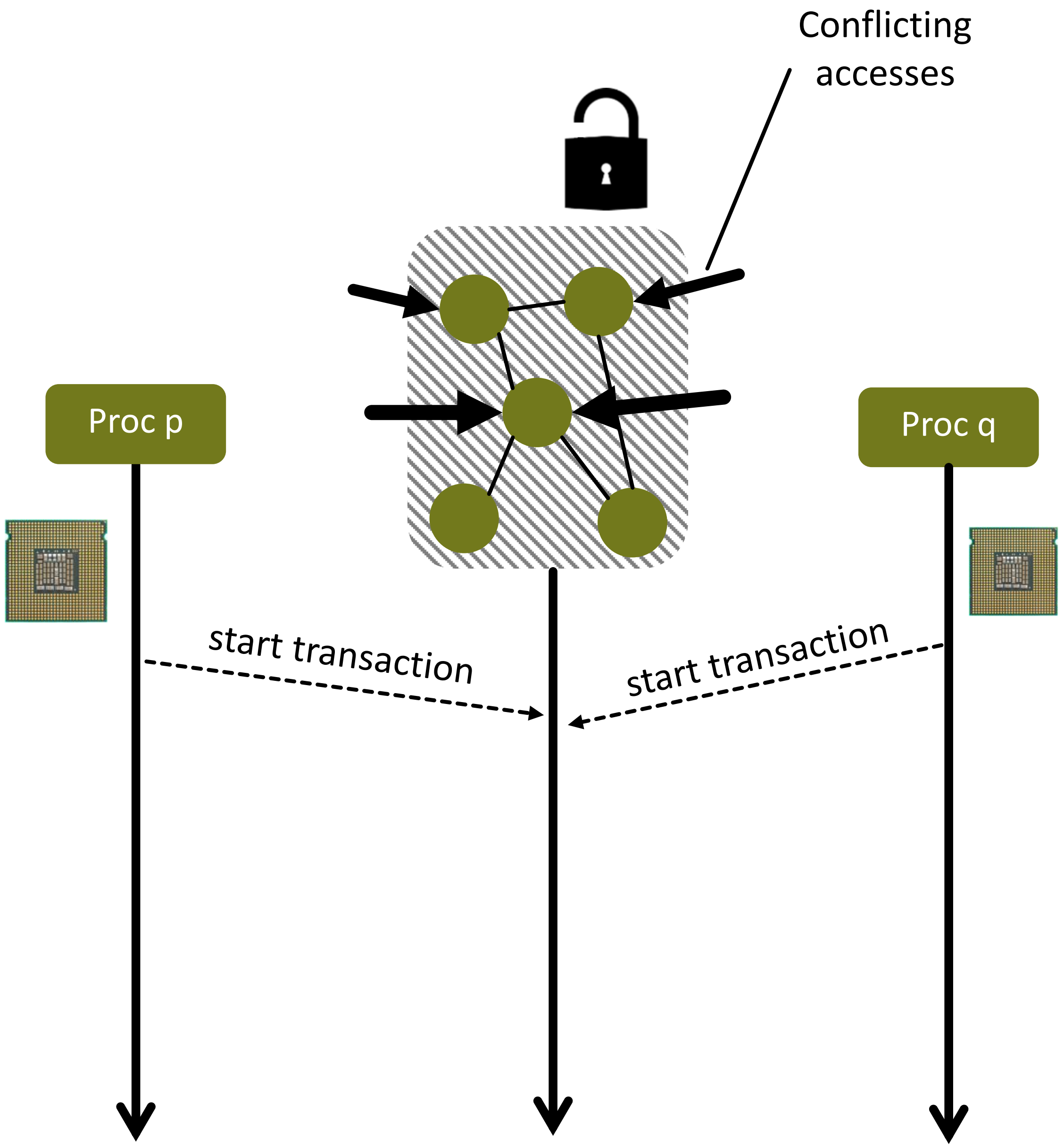
[1] N. Shavit and D. Touitou. Software transactional memory. PODC’95.

TRANSACTIONAL MEMORY [1]

Conflicts solved with rollbacks and/or serialization.

Software overheads

Simple protocols



[1] N. Shavit and D. Toutiou. Software transactional memory. PODC'95.

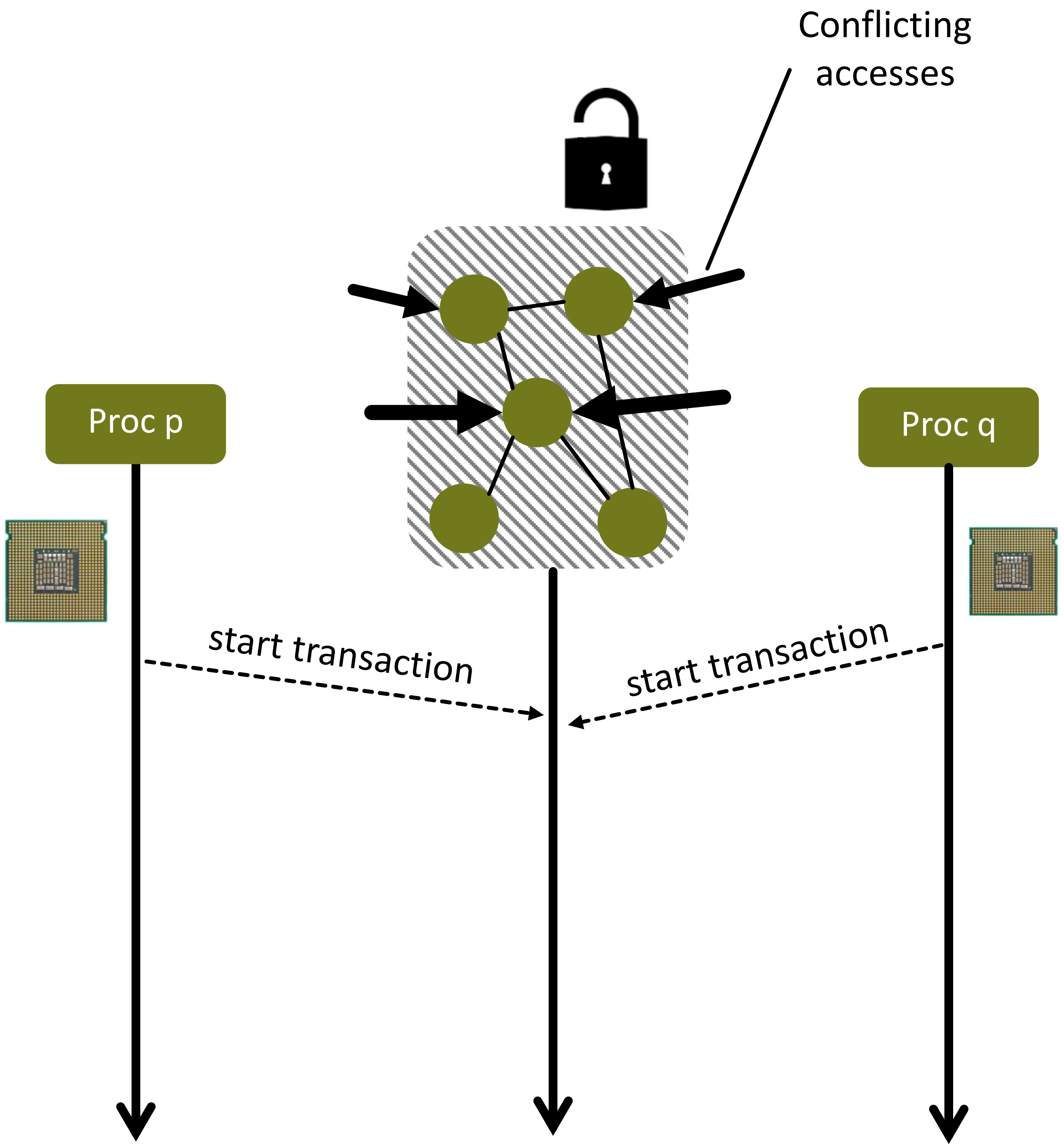
TRANSACTIONAL MEMORY [1]

Conflicts solved with rollbacks and/or serialization.

High performance? For graphs?

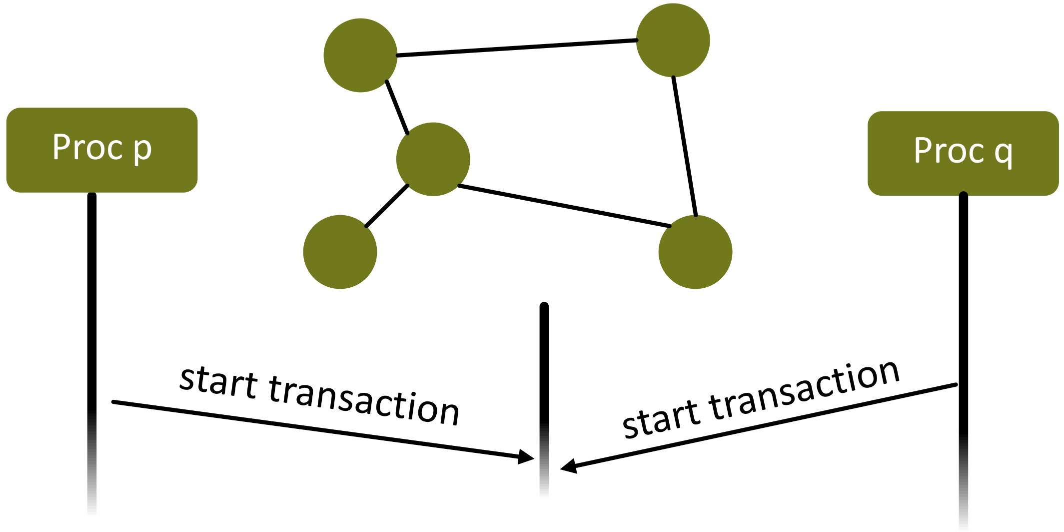
Software overheads

Simple protocols



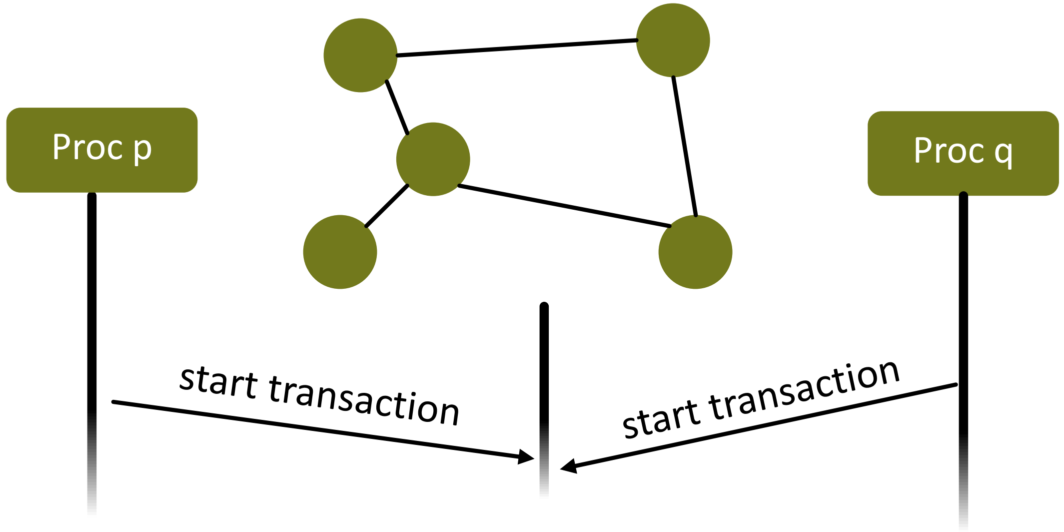
[1] N. Shavit and D. Touitou. Software transactional memory. PODC'95.

Shared- & Distributed-memory Machines



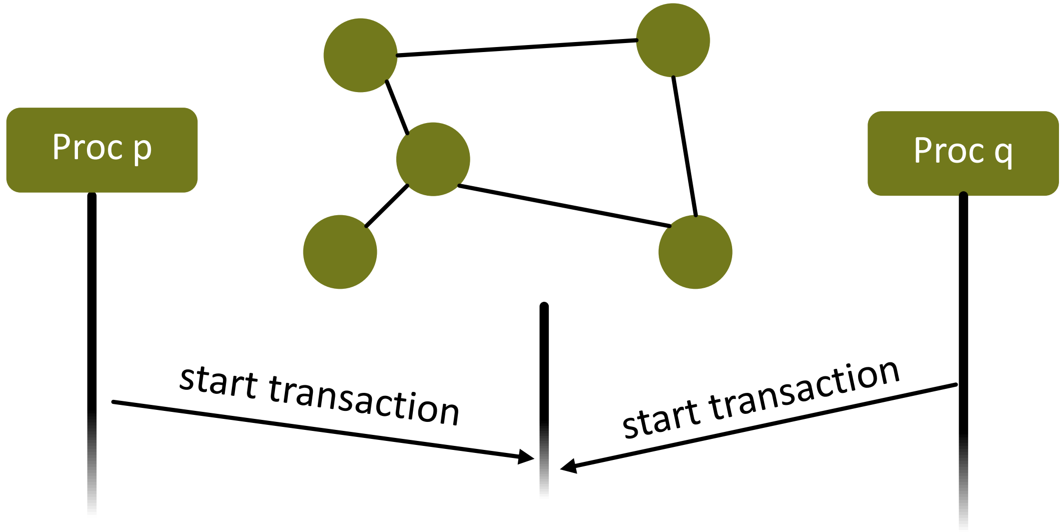
Shared- & Distributed-memory Machines

- HTM works fine for single shared-memory domains



SHARED- & DISTRIBUTED-MEMORY MACHINES

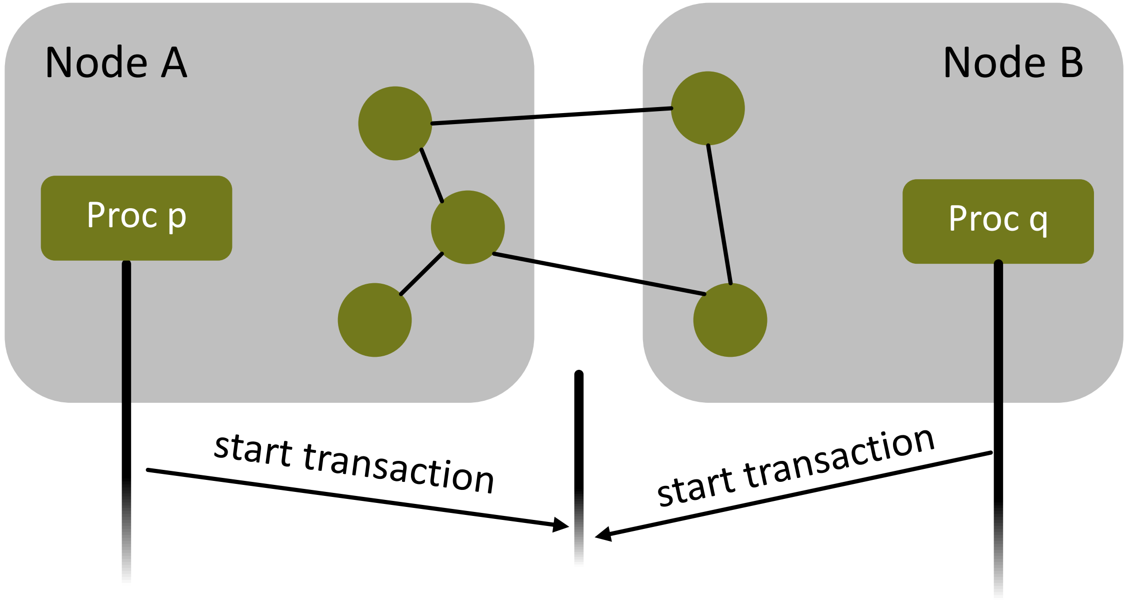
- HTM works fine for single shared-memory domains
  - Most graphs fit in such machines [1]



[1] Y. Perez et al. Ringo: Interactive Graph Analytics on Big-Memory Machines. SIGCOMM'14.

SHARED- & DISTRIBUTED-MEMORY MACHINES

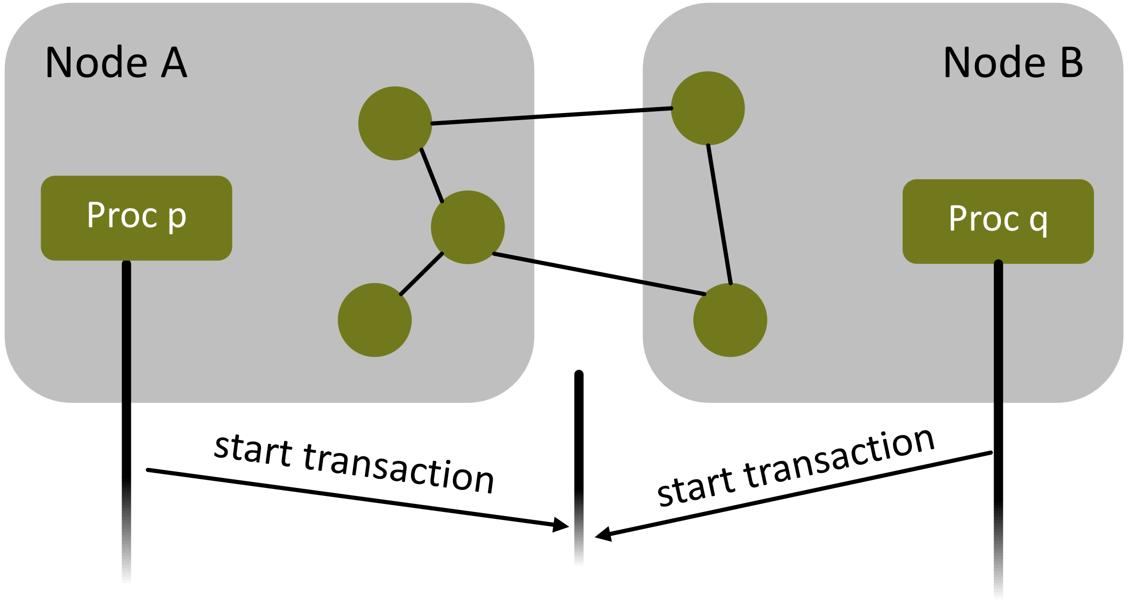
- HTM works fine for single shared-memory domains
  - Most graphs fit in such machines [1]



[1] Y. Perez et al. Ringo: Interactive Graph Analytics on Big-Memory Machines. SIGCOMM'14.

SHARED- & DISTRIBUTED-MEMORY MACHINES

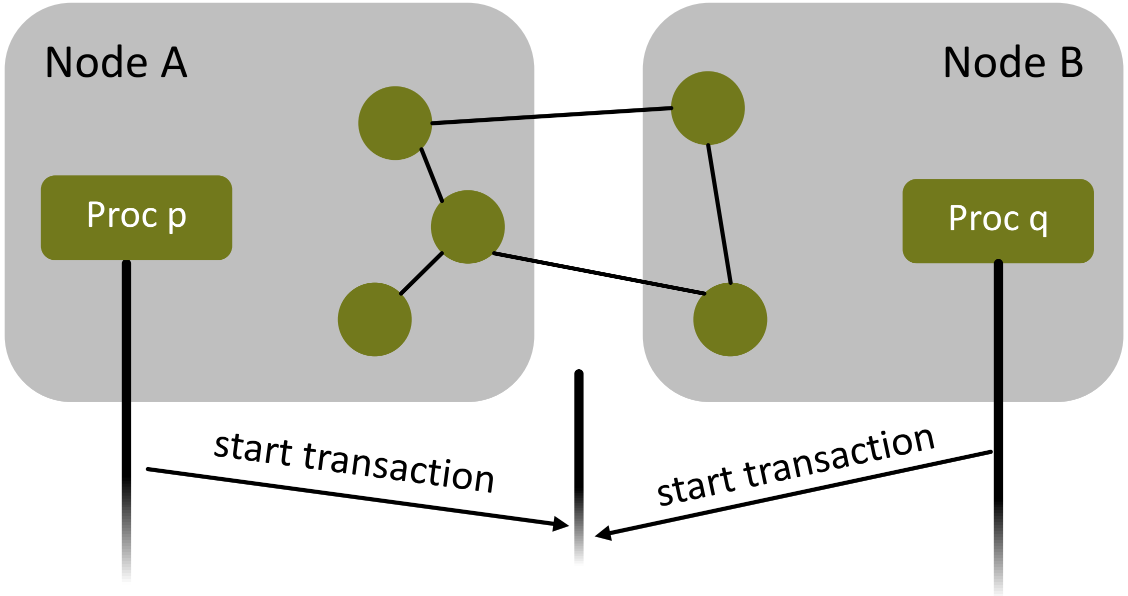
- HTM works fine for single shared-memory domains
  - Most graphs fit in such machines [1]
- However, some do not:



[1] Y. Perez et al. Ringo: Interactive Graph Analytics on Big-Memory Machines. SIGCOMM'14.

Shared- & Distributed-memory Machines

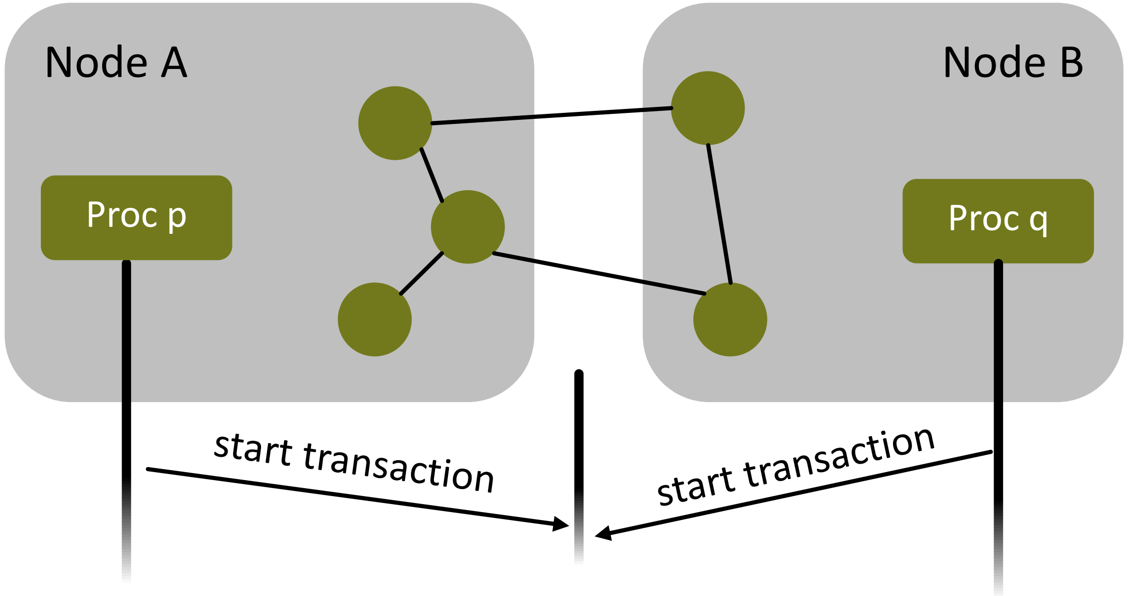
- HTM works fine for single shared-memory domains
  - Most graphs fit in such machines [1]
- However, some do not:
  - Very large instances



[1] Y. Perez et al. Ringo: Interactive Graph Analytics on Big-Memory Machines. SIGCOMM'14.

SHARED- & DISTRIBUTED-MEMORY MACHINES

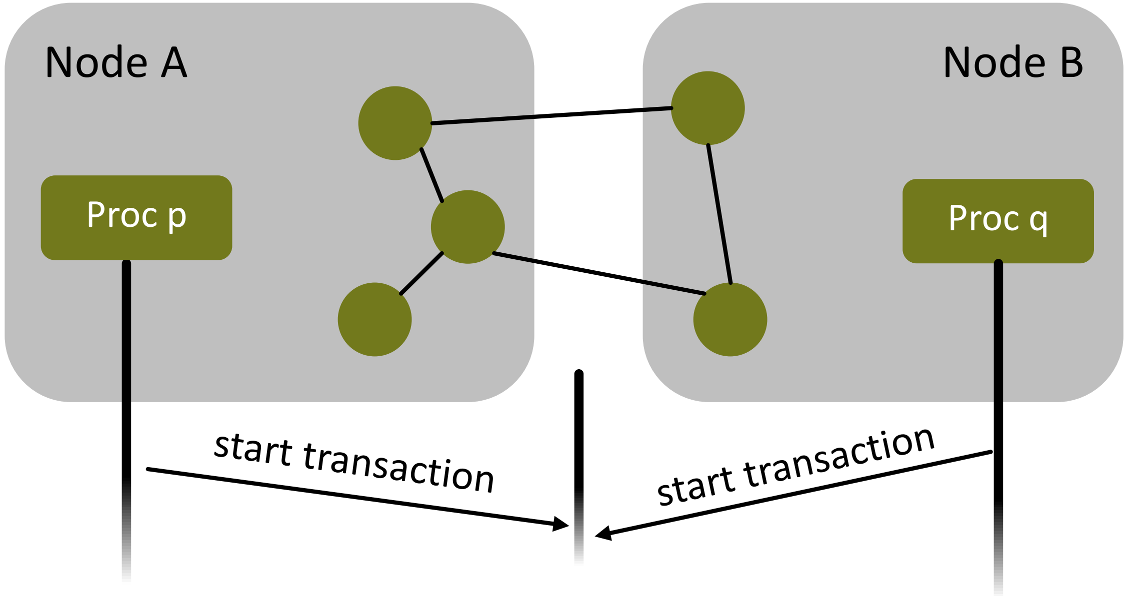
- HTM works fine for single shared-memory domains
  - Most graphs fit in such machines [1]
- However, some do not:
  - Very large instances
  - Rich vertex/edge data



[1] Y. Perez et al. Ringo: Interactive Graph Analytics on Big-Memory Machines. SIGCOMM'14.

SHARED- & DISTRIBUTED-MEMORY MACHINES

- HTM works fine for single shared-memory domains
  - Most graphs fit in such machines [1]
- However, some do not:
  - Very large instances
  - Rich vertex/edge data
- Fat nodes with lots of RAM still expensive ($35K for a machine with 1TB of RAM [1])

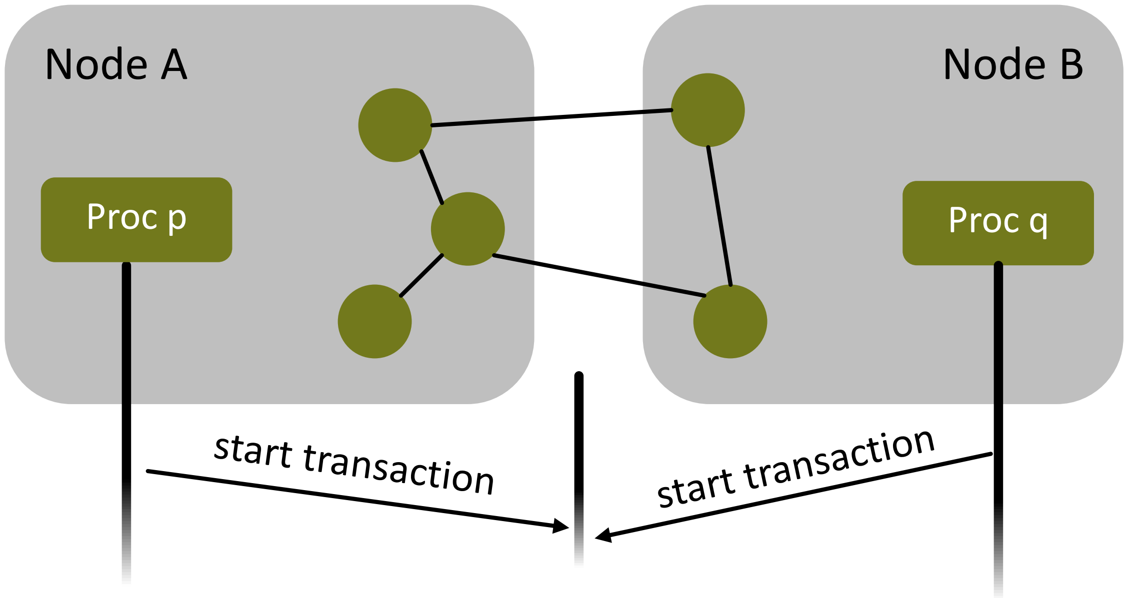


[1] Y. Perez et al. Ringo: Interactive Graph Analytics on Big-Memory Machines. SIGCOMM'14.

Shared- & Distributed-memory Machines

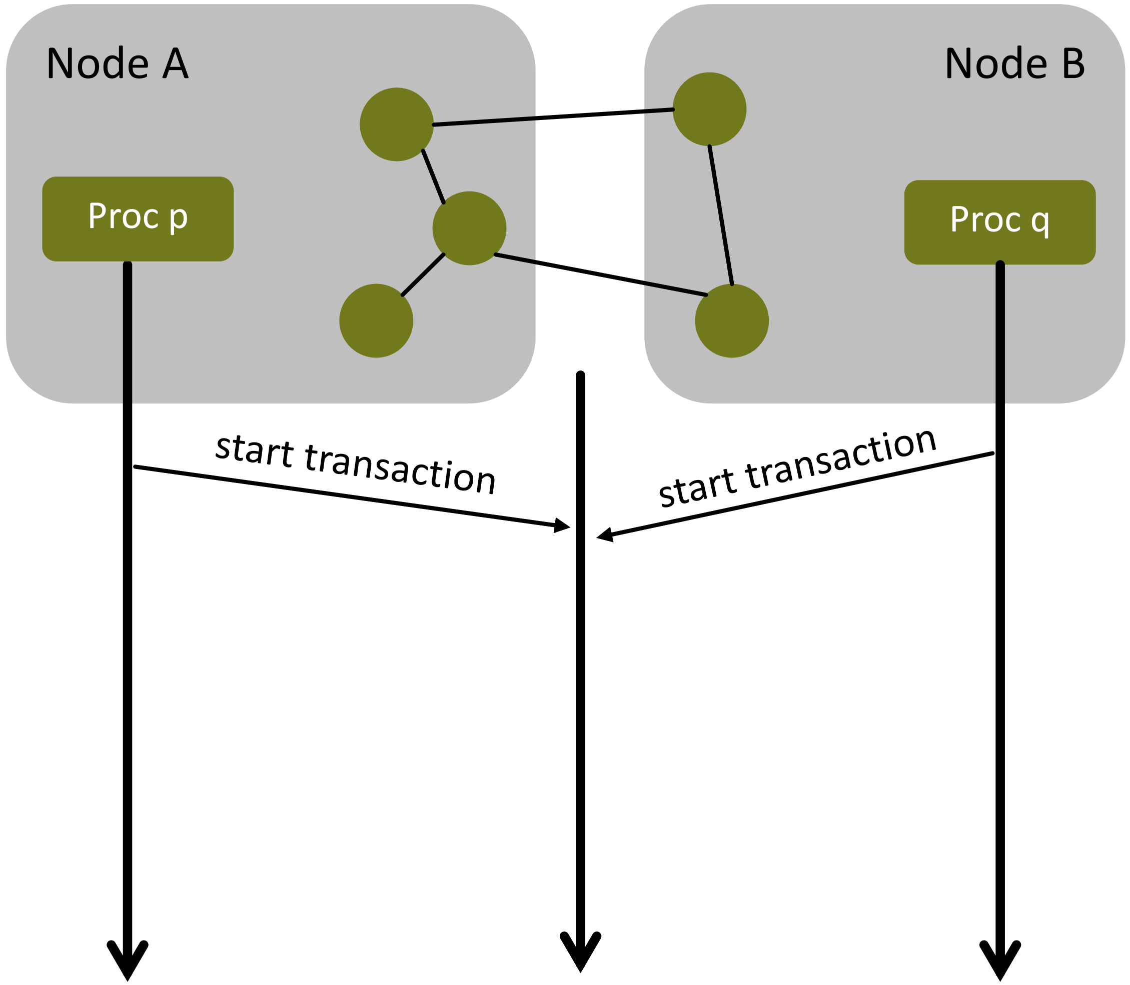
- HTM works fine for single shared-memory domains
  - Most graphs fit in such machines [1]
- However, some do not:
  - Very large instances
  - Rich vertex/edge data
- Fat nodes with lots of RAM still expensive ($35K for a machine with 1TB of RAM [1])

How to apply HTM in such a setting?

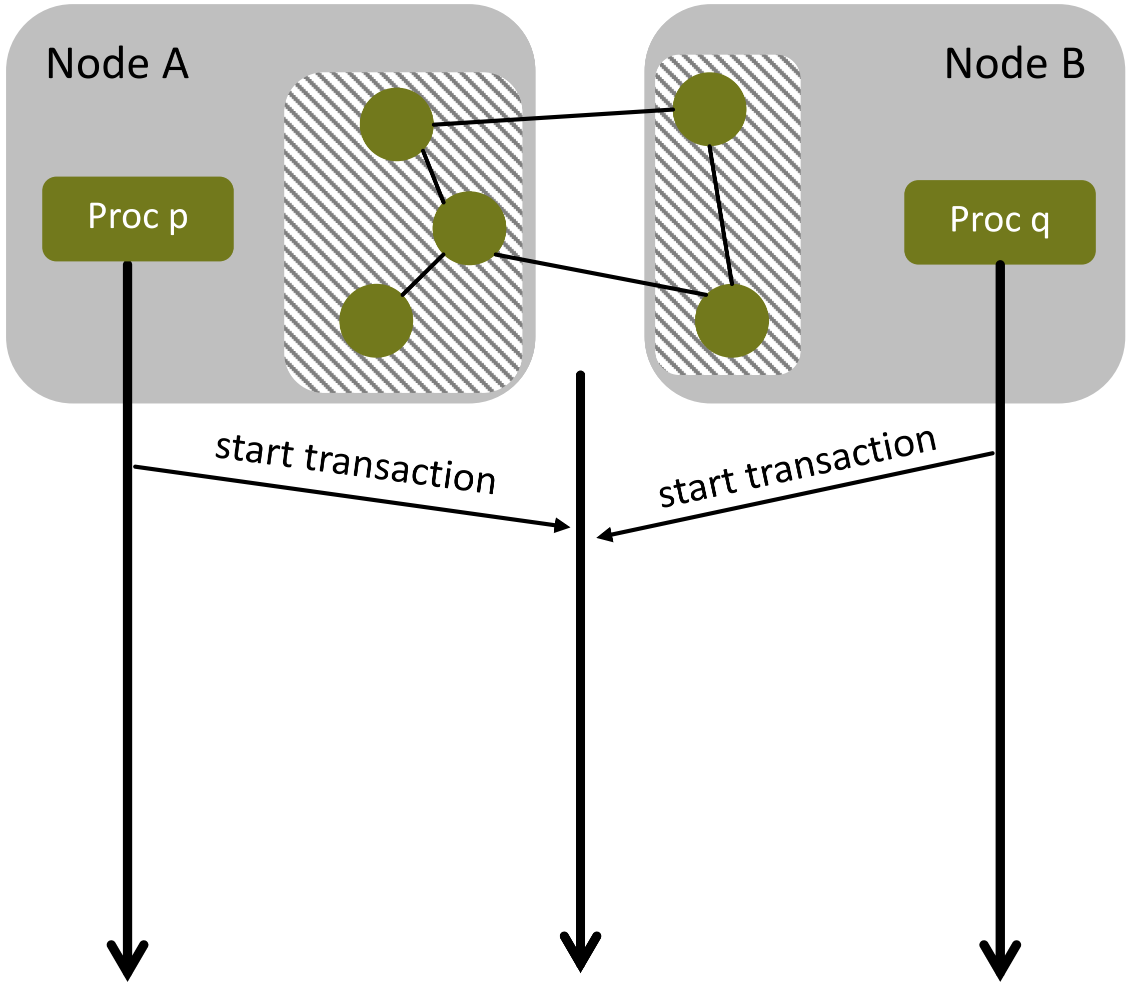


[1] Y. Perez et al. Ringo: Interactive Graph Analytics on Big-Memory Machines. SIGCOMM’14.

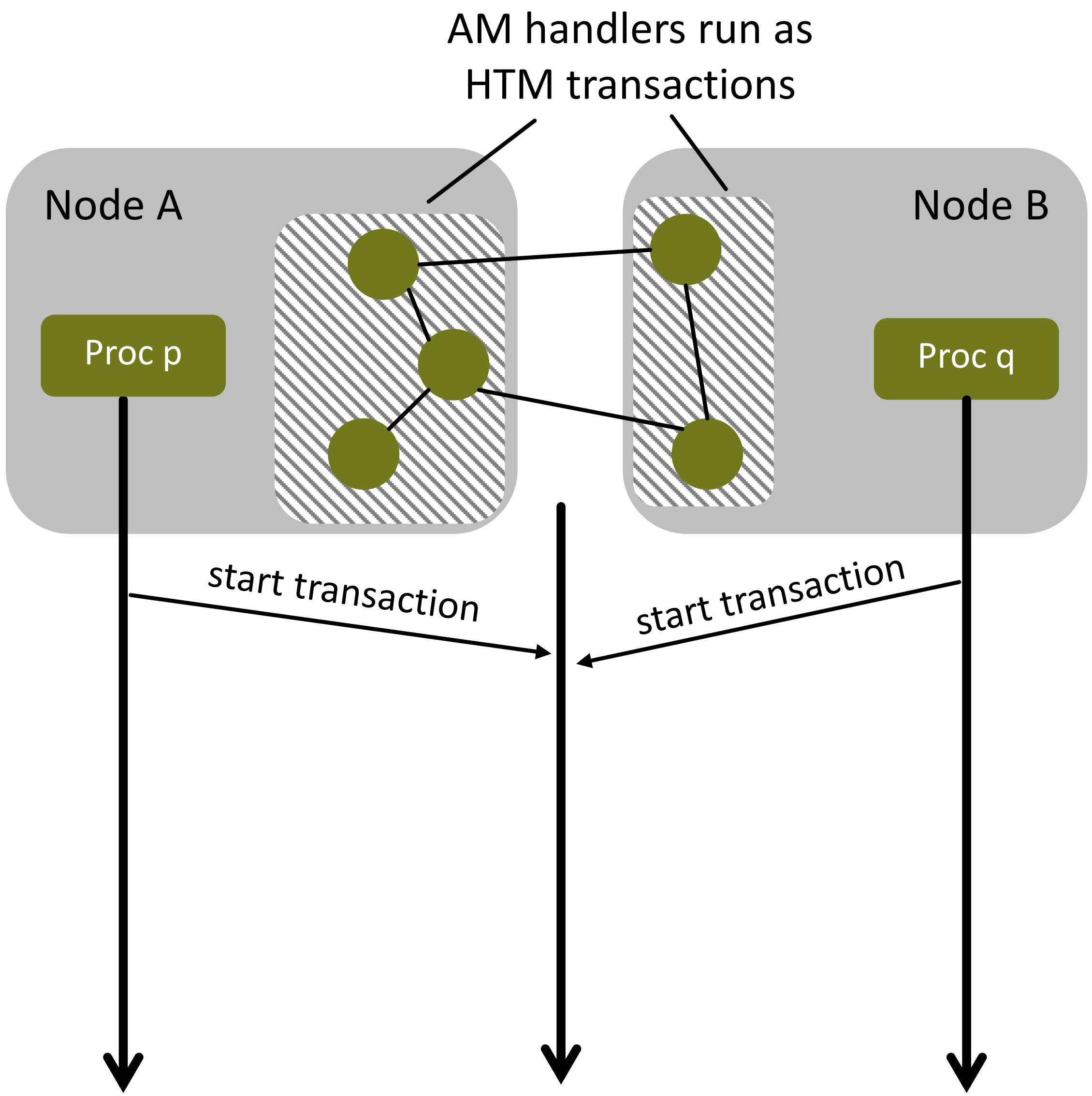
AM + HTM = ...


Node A
Node B
Proc p
Proc q
start transaction
start transaction

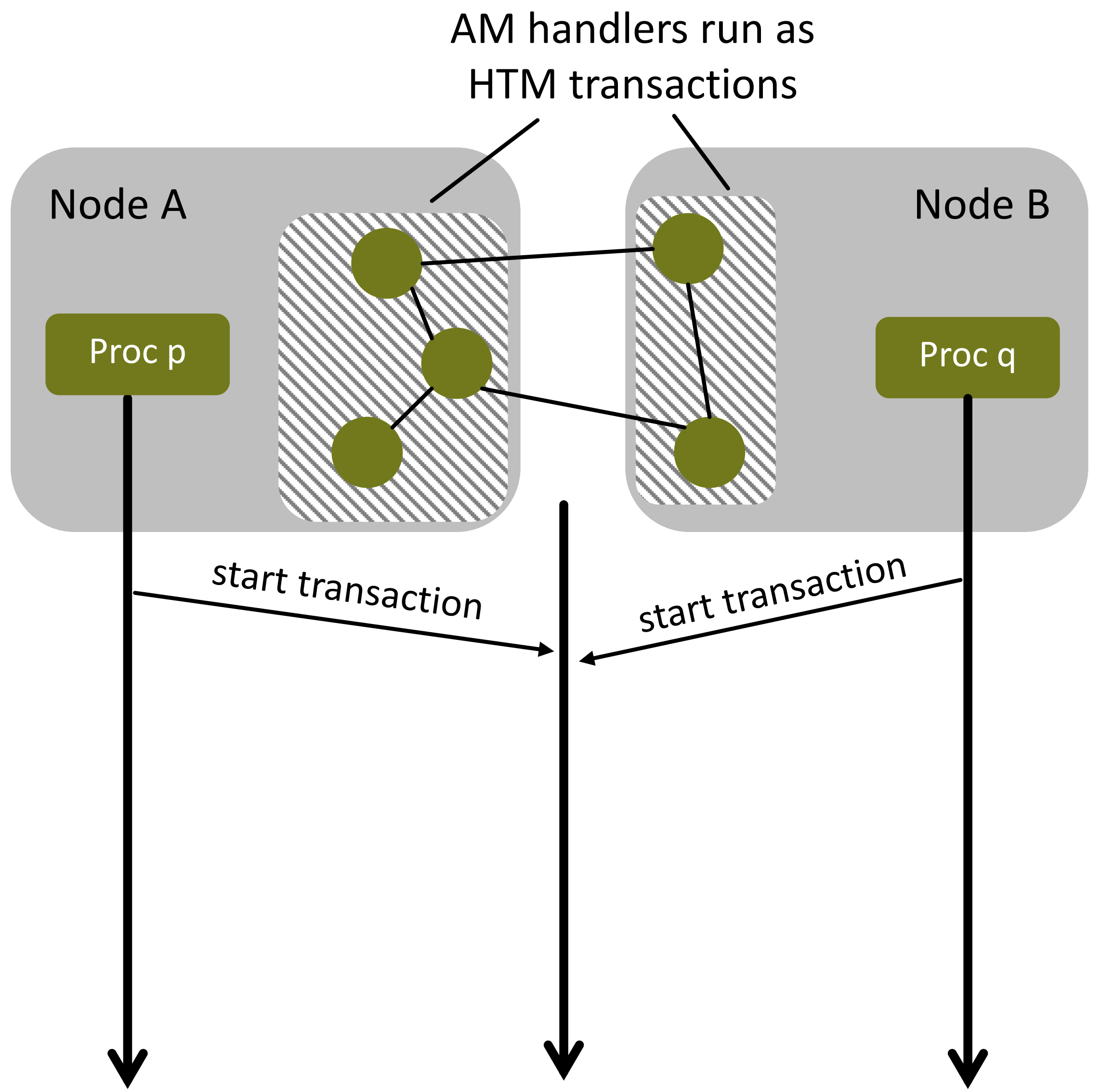
AM + HTM = ...


Node A
Node B
Proc p
Proc q
start transaction
start transaction

AM + HTM = ...

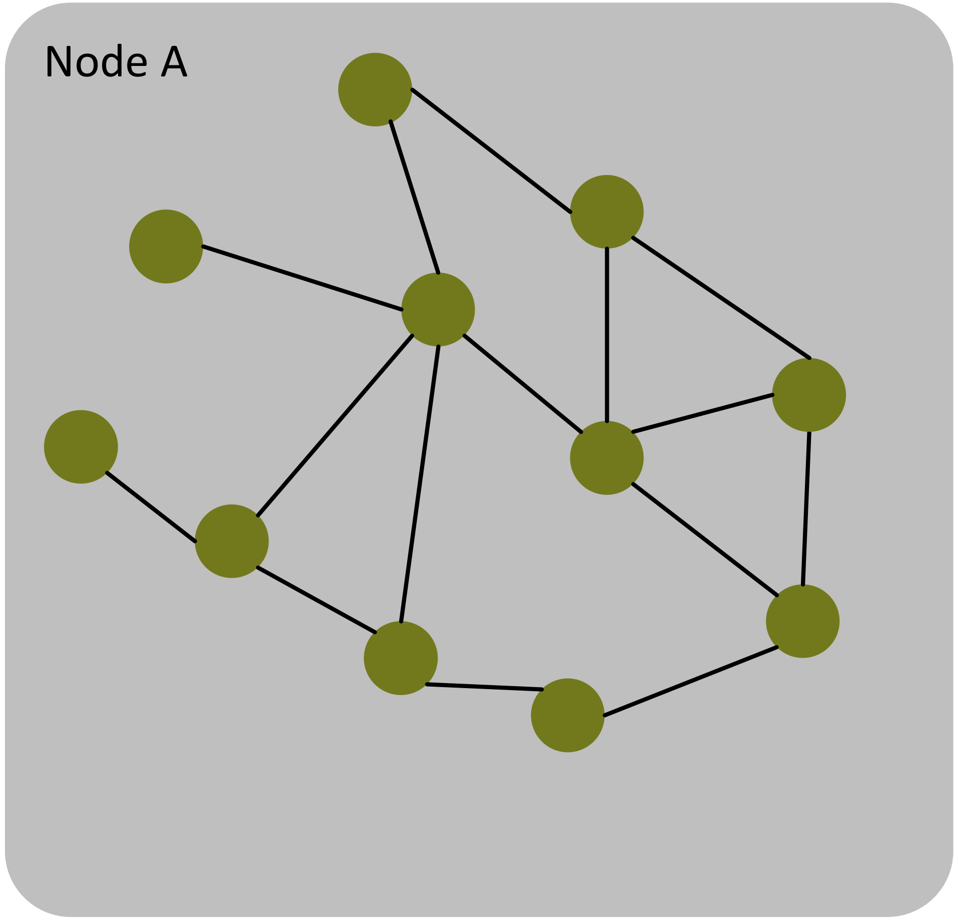

AM handlers run as
HTM transactions
Node A
Node B
Proc p
Proc q
start transaction
start transaction

AM + HTM = Atomic Active Messages



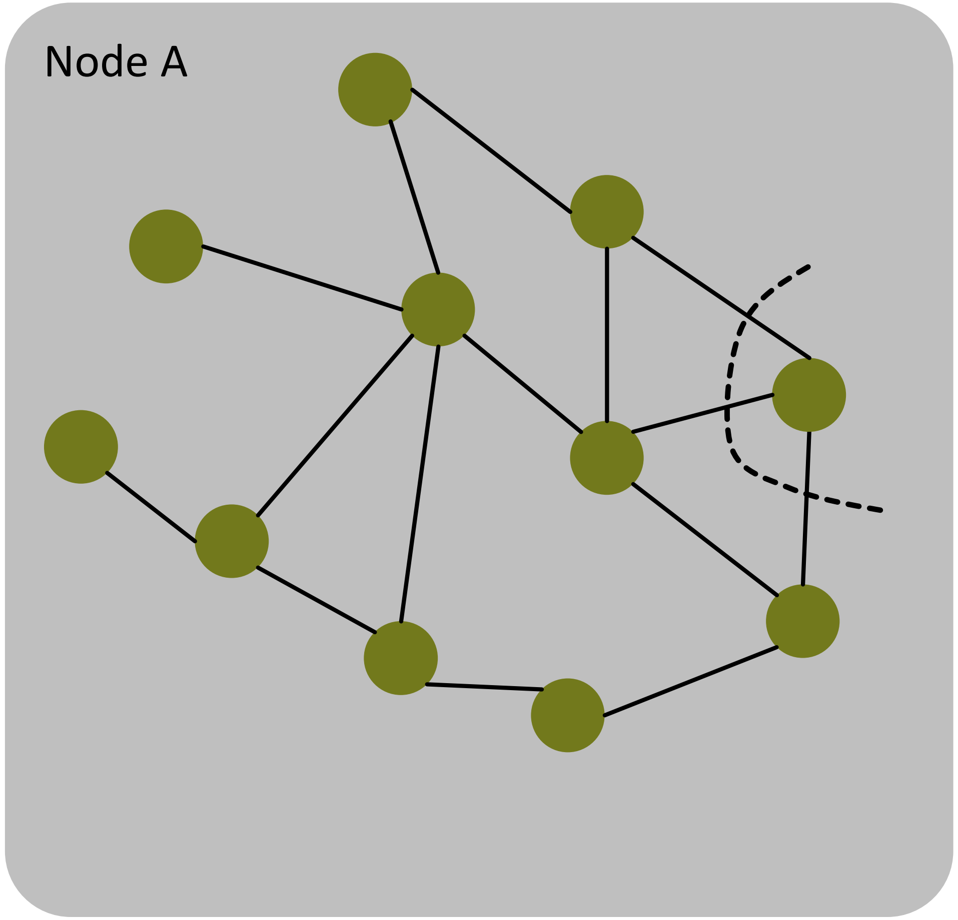
Accessing Multiple Vertices Atomically

Example: BFS



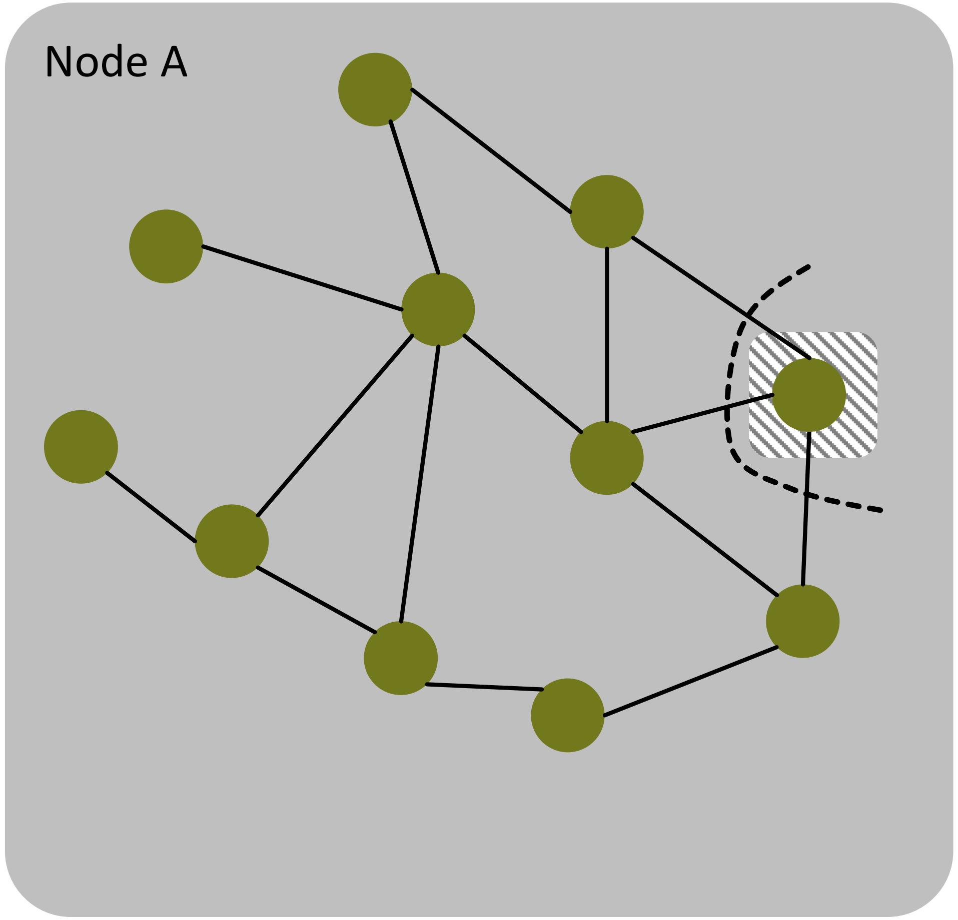
ACCESSING MULTIPLE VERTICES ATOMICALLY

Example: BFS



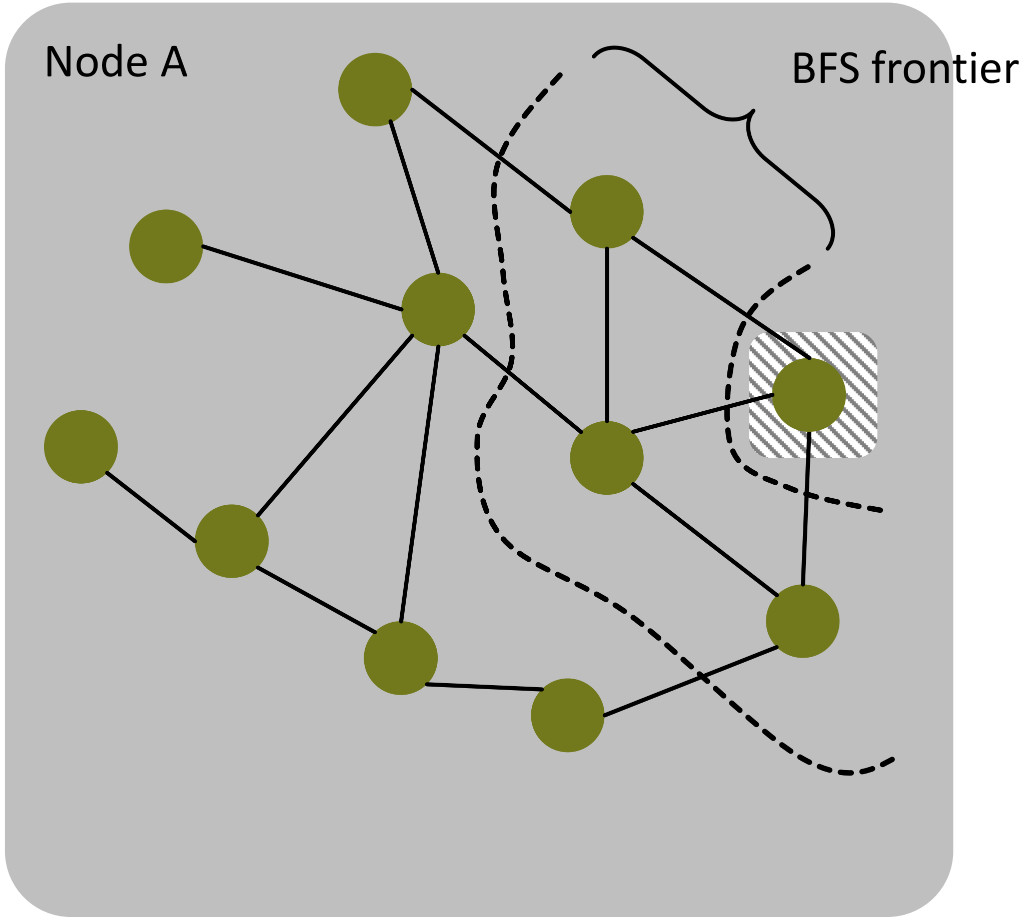
ACCESSING MULTIPLE VERTICES ATOMICALLY

Example: BFS



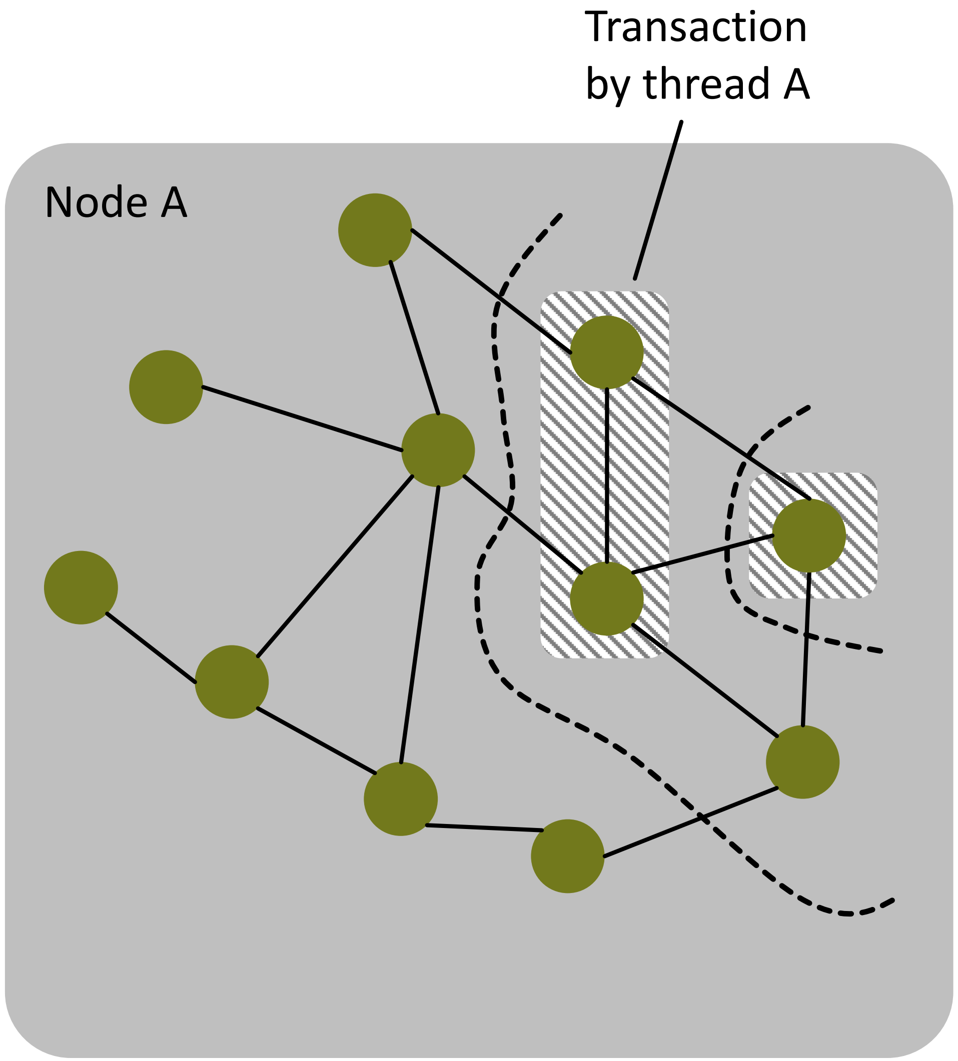
Accessing Multiple Vertices Atomically

Example: BFS



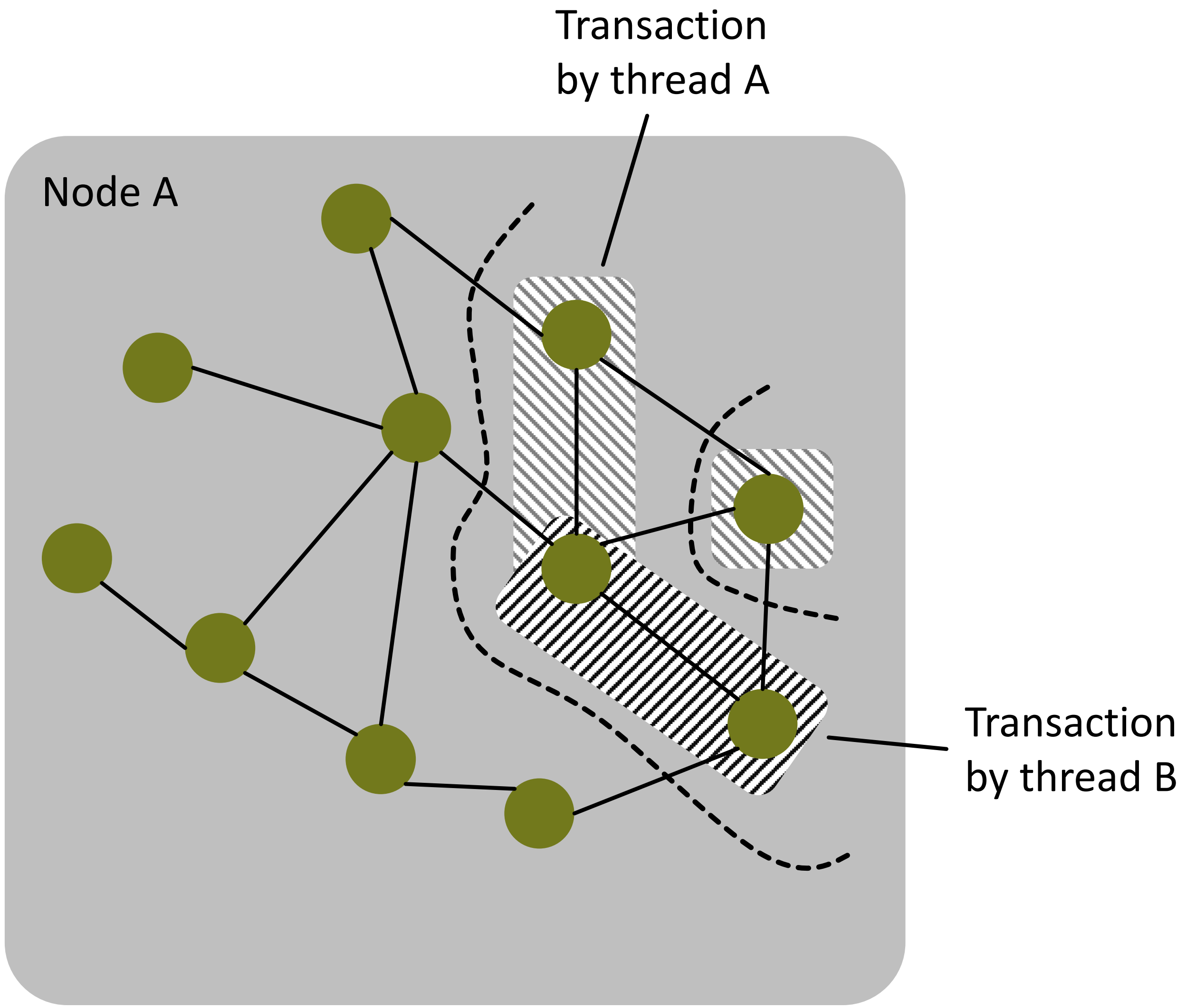
Accessing Multiple Vertices Atomically

Example: BFS



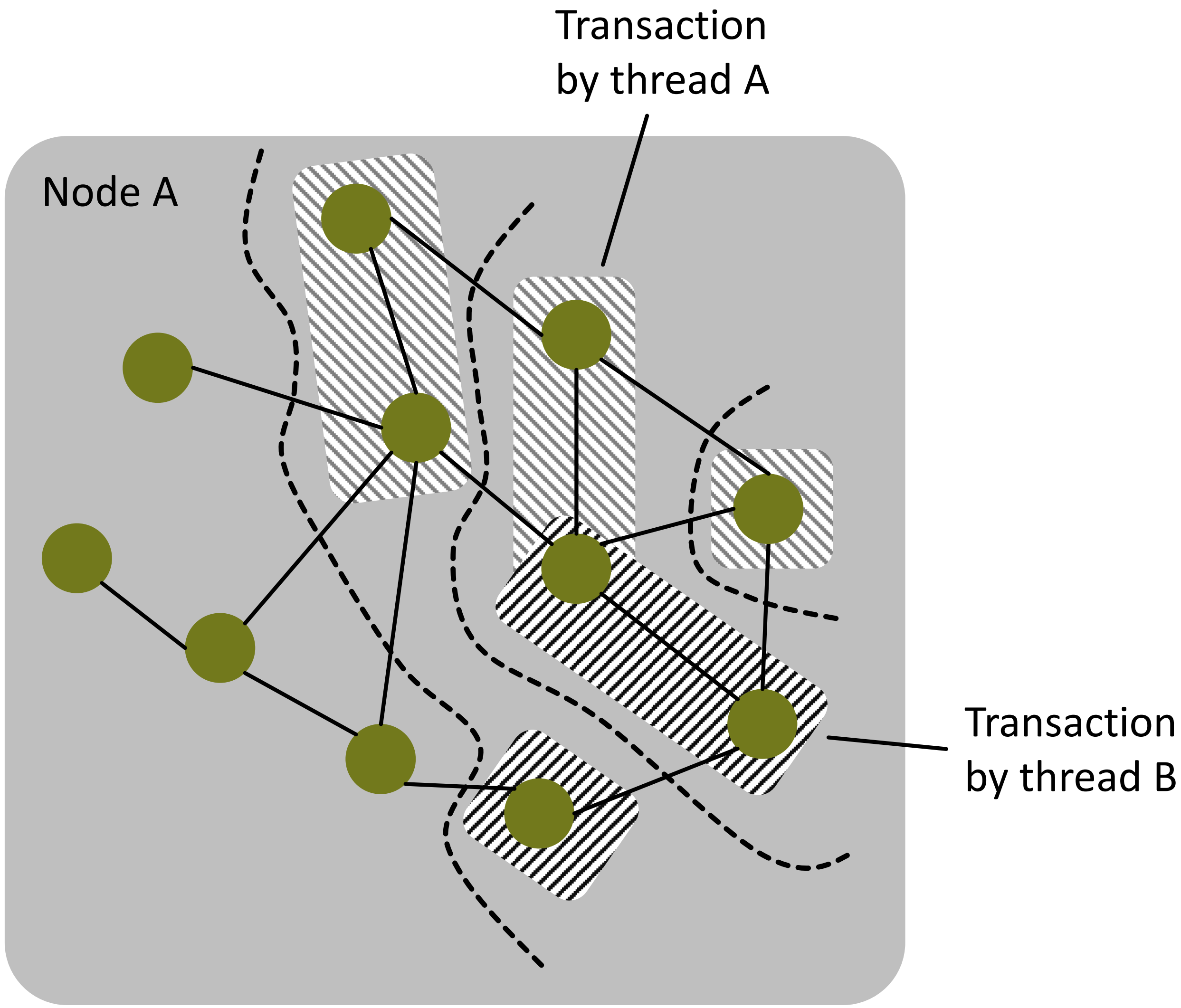
Accessing Multiple Vertices Atomically

Example: BFS



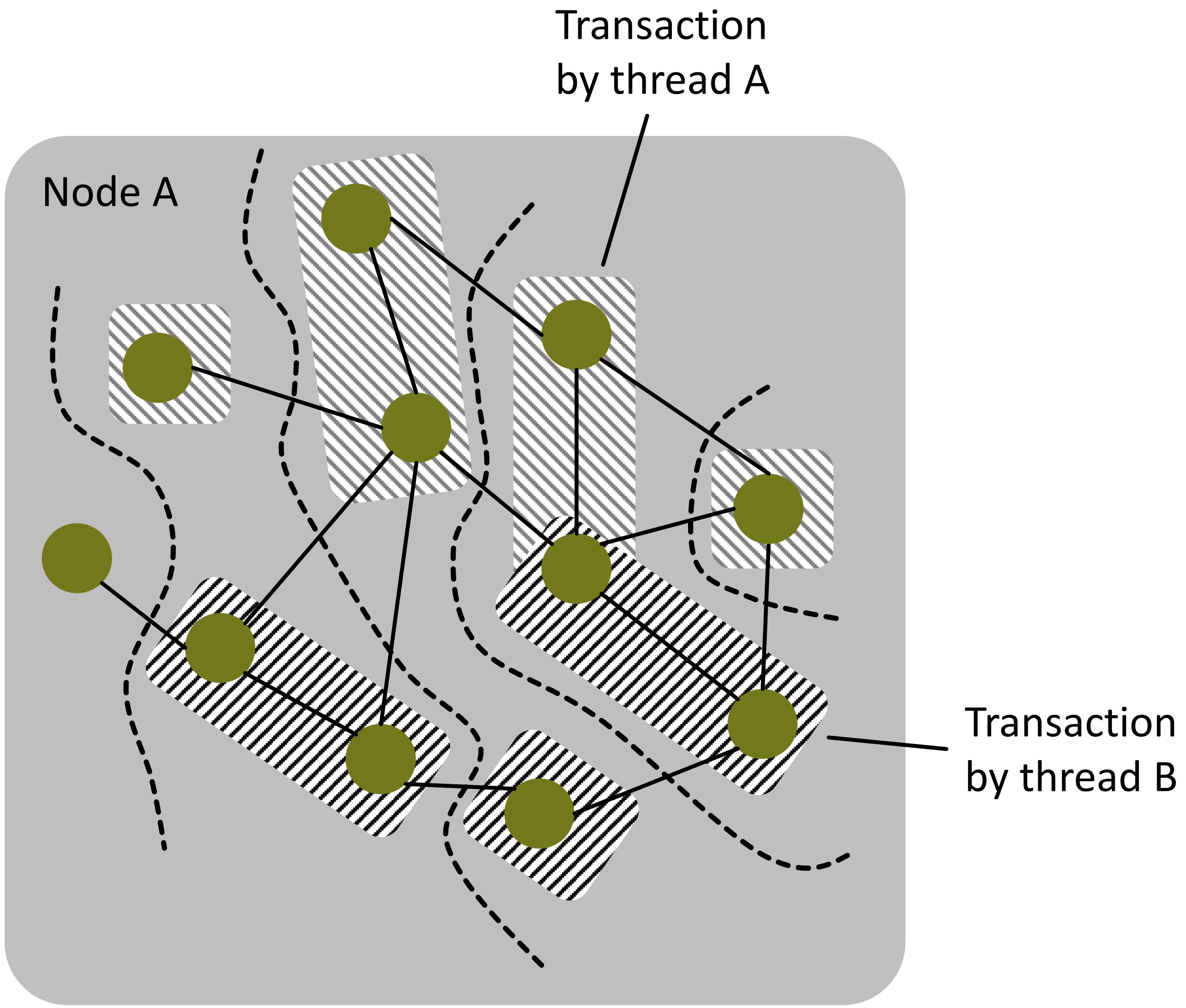
Accessing Multiple Vertices Atomically

Example: BFS



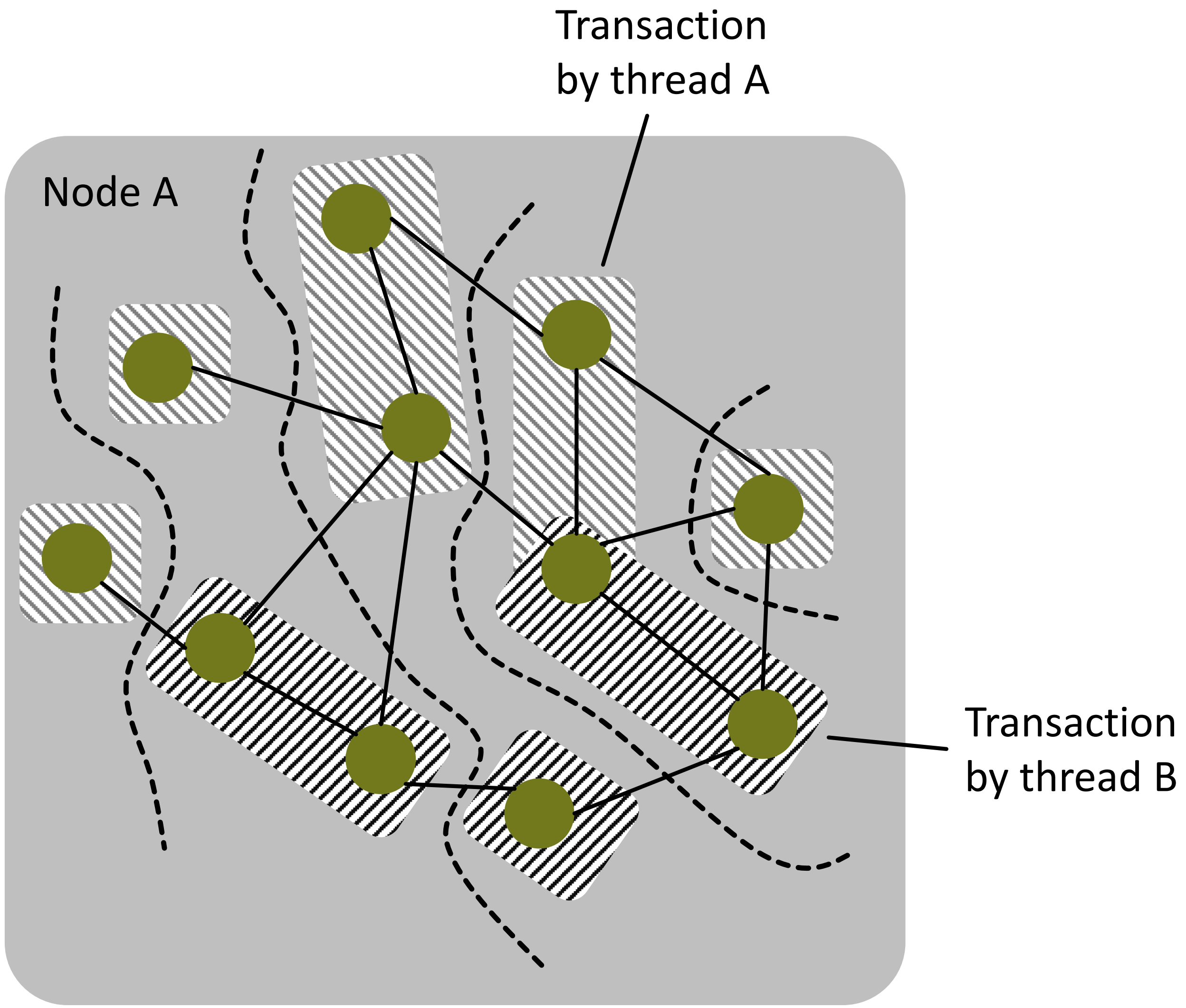
ACCESSING MULTIPLE VERTICES ATOMICALLY

Example: BFS



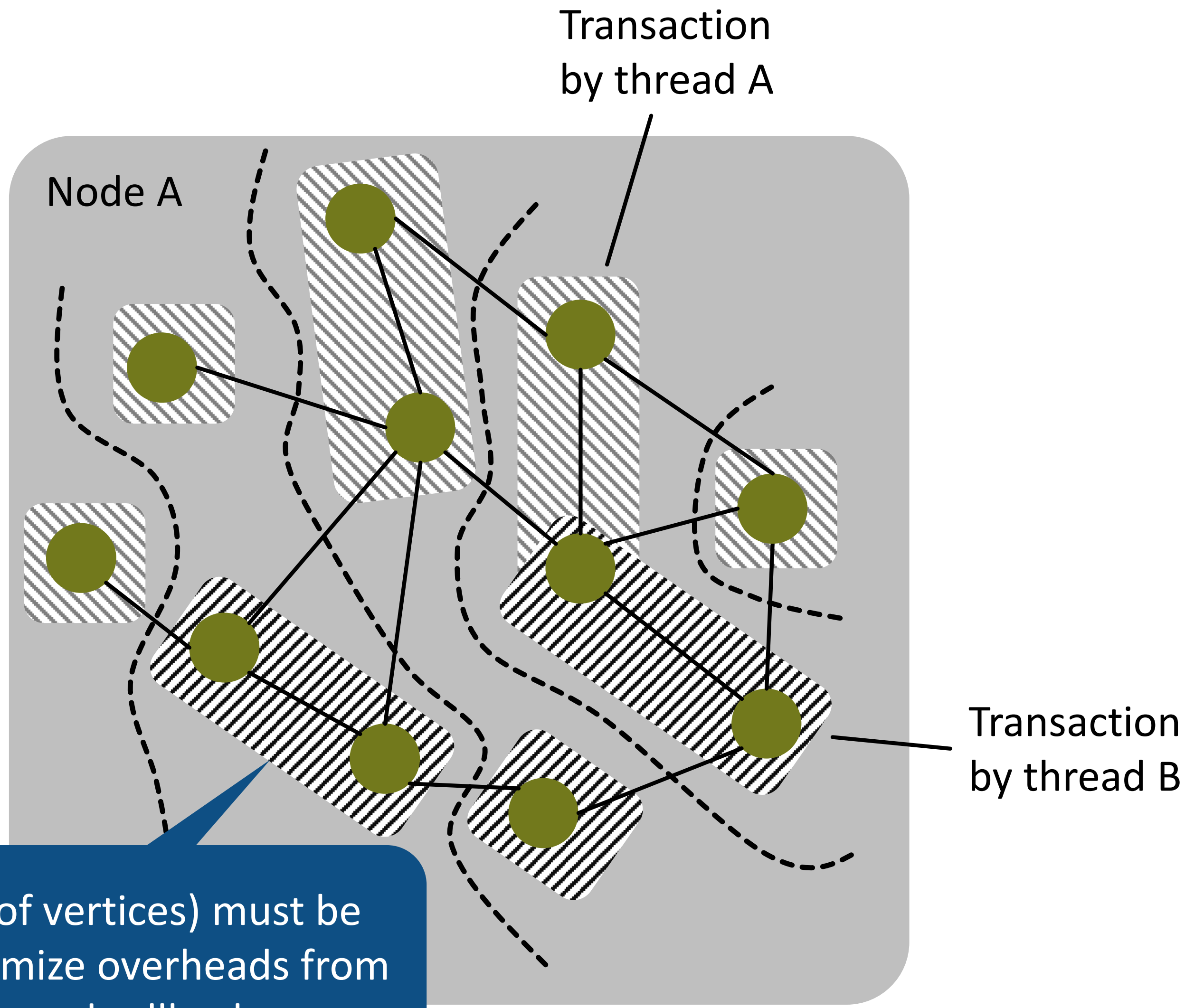
ACCESSING MULTIPLE VERTICES ATOMICALLY

Example: BFS



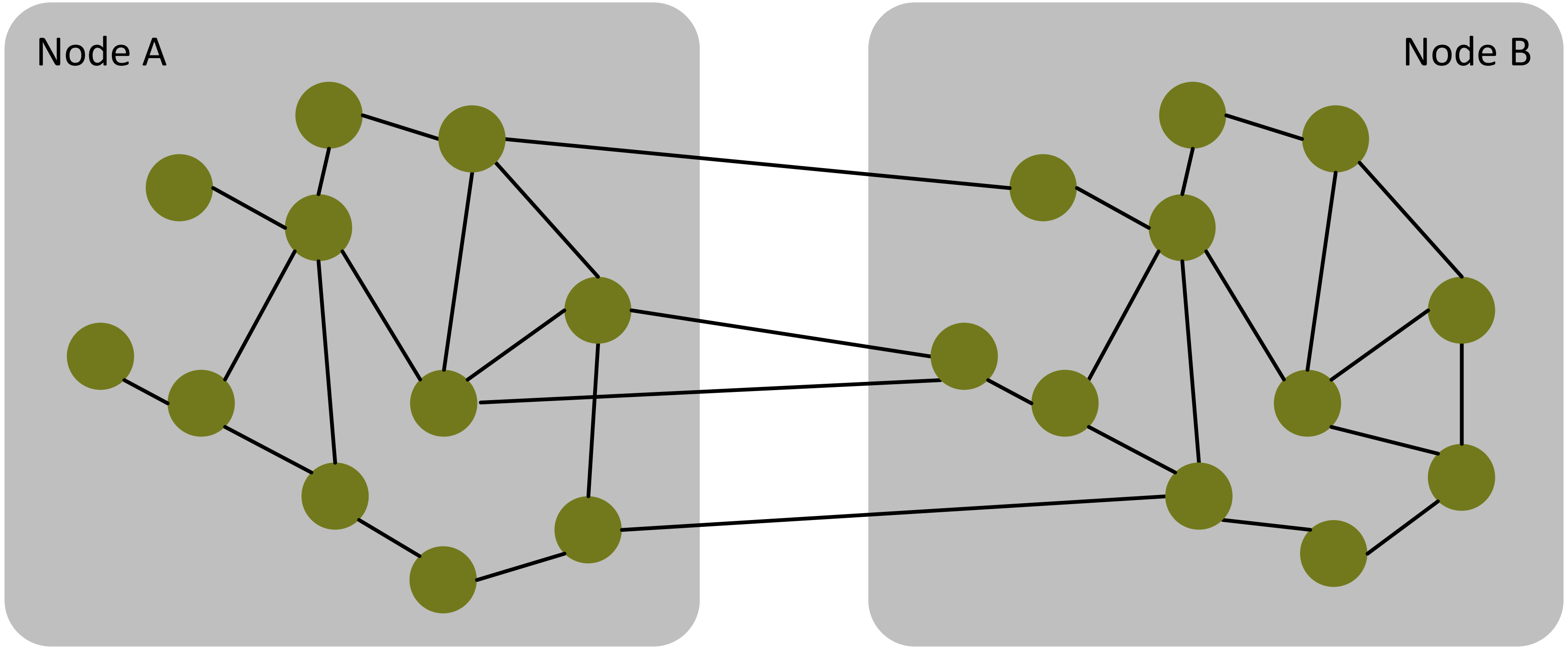
Accessing Multiple Vertices Atomically

Example: BFS

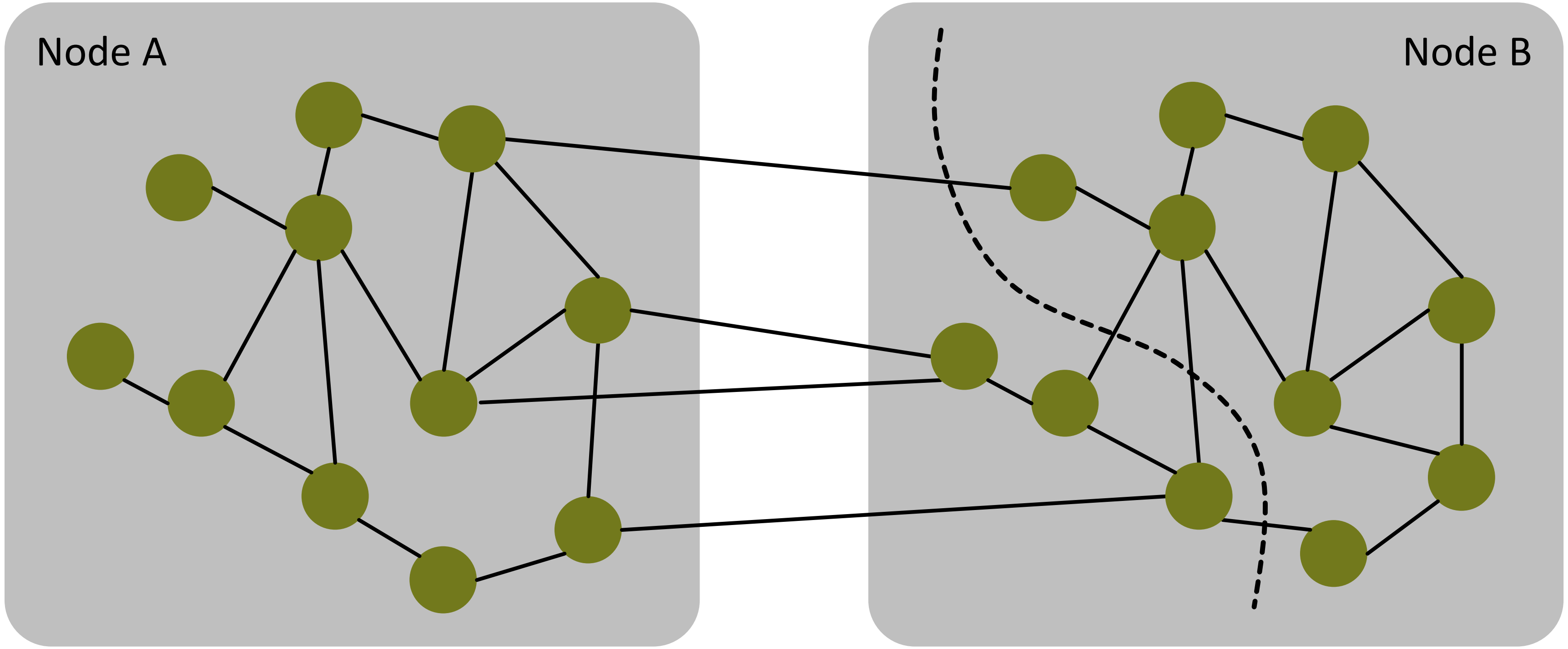


Size (the number of vertices) must be appropriate to minimize overheads from both commits and rollbacks

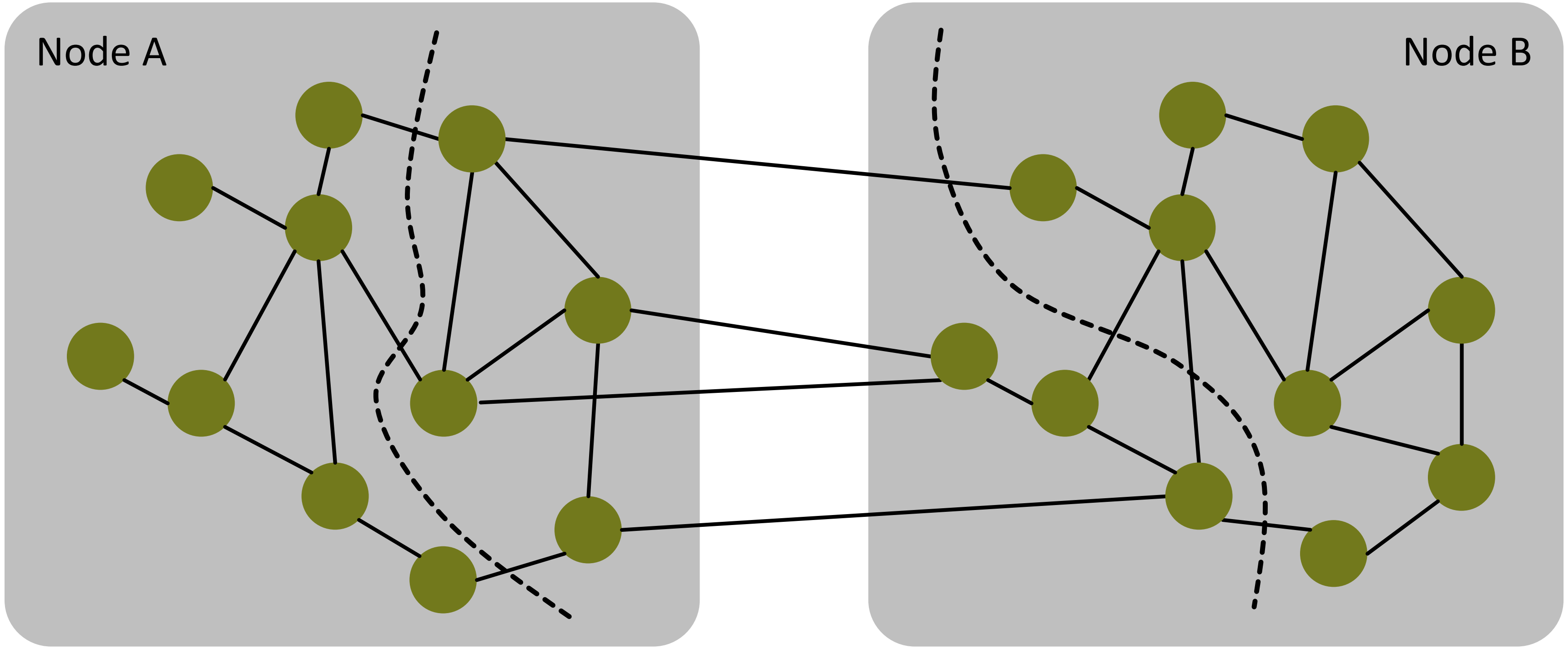
Executing Transactions on Multiple Nodes



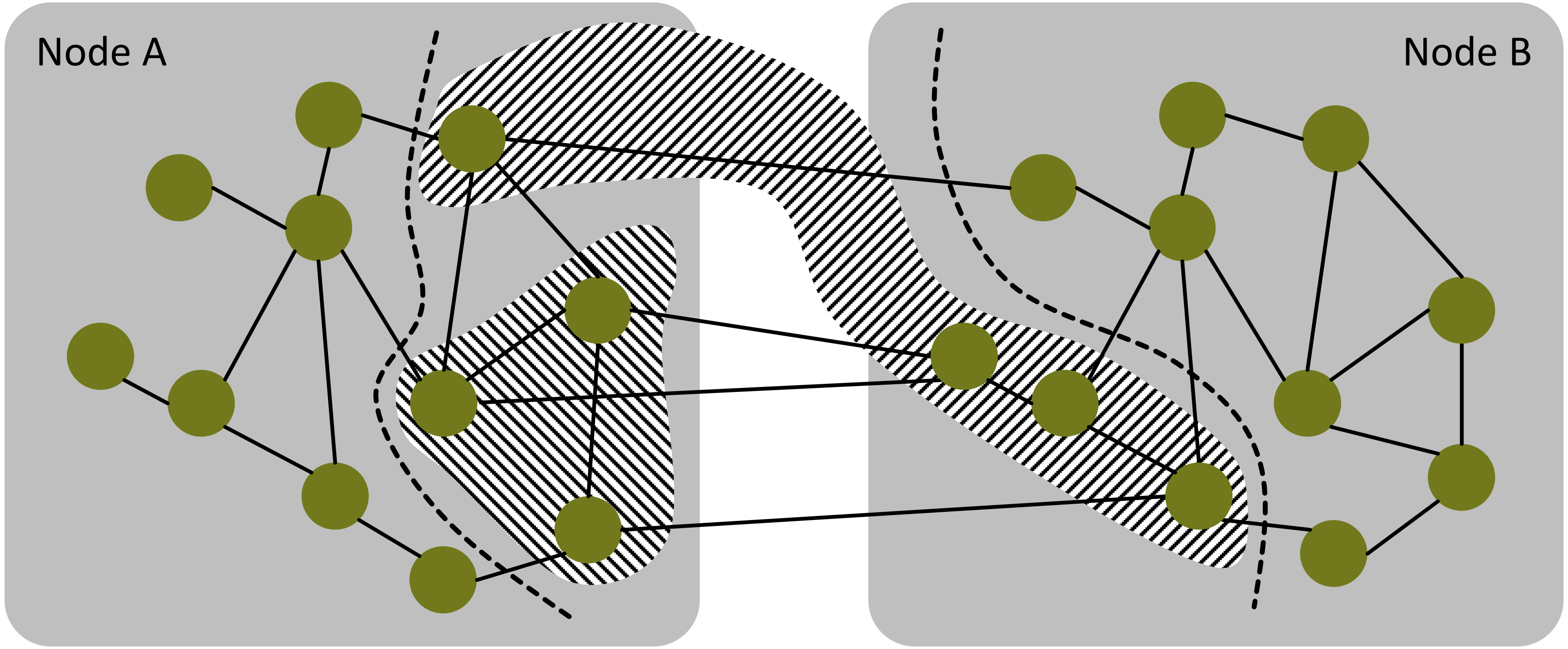
Executing Transactions on Multiple Nodes



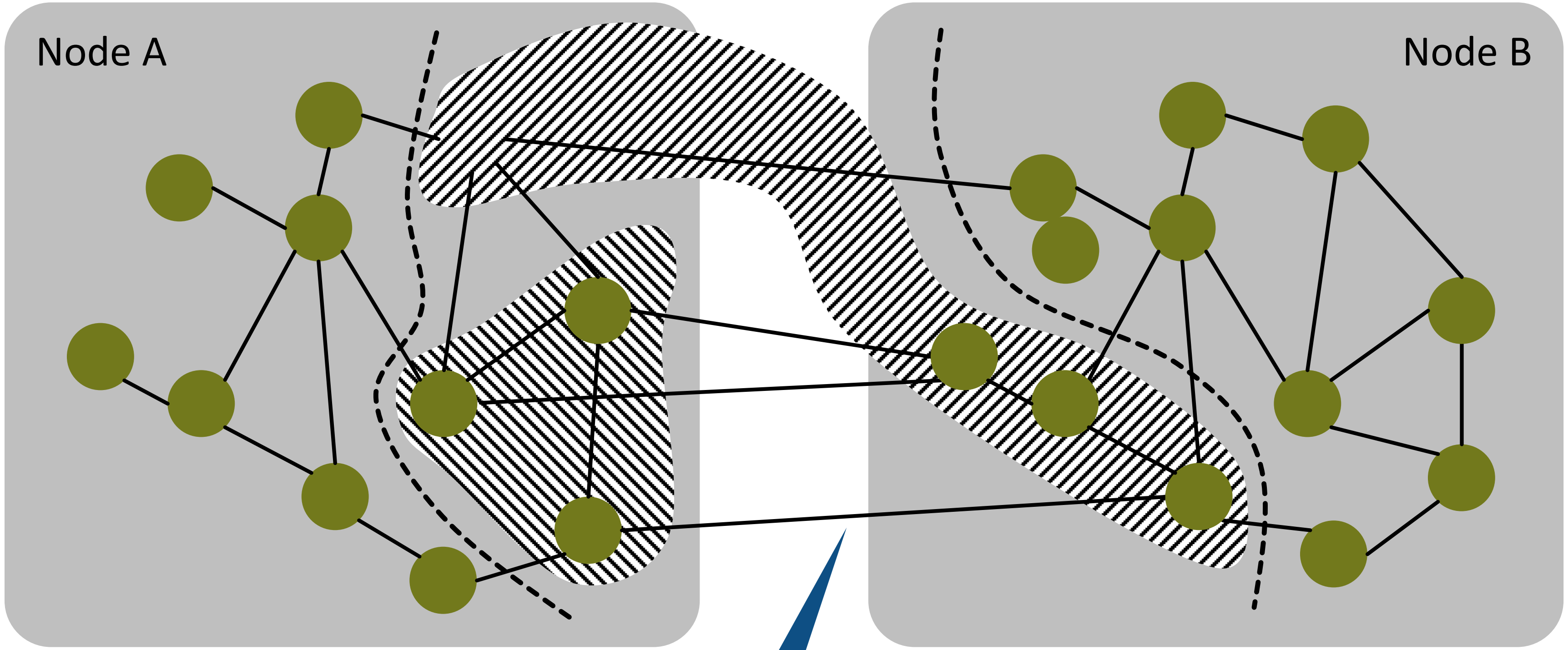
EXECUTING TRANSACTIONS ON MULTIPLE NODES



Executing Transactions on Multiple Nodes

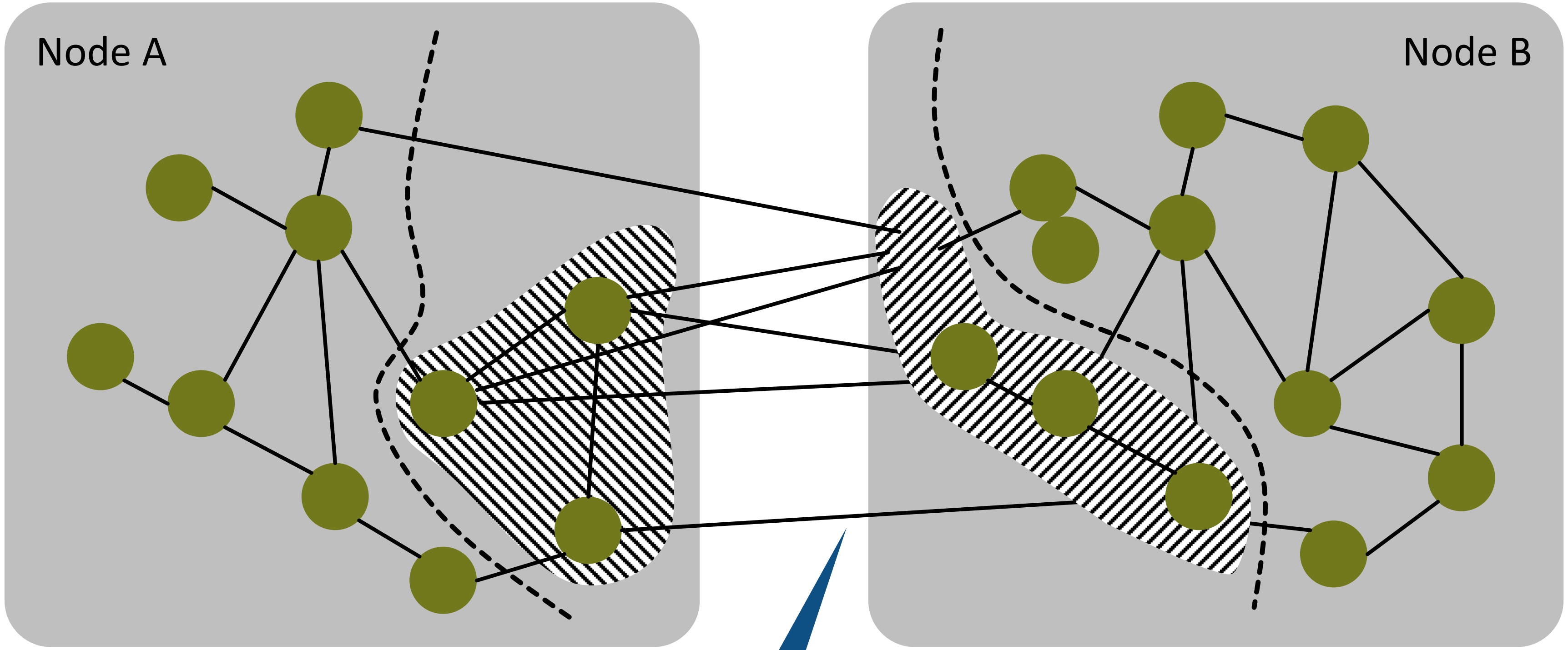


EXECUTING TRANSACTIONS ON MULTIPLE NODES



Vertices must be appropriately relocated to enable execution of a hardware transaction

Executing Transactions on Multiple Nodes



Vertices must be appropriately relocated to enable execution of a hardware transaction

Performance Model

Atomics vs transactions

Performance Model

Atomics vs transactions

Time to modify *N* vertices with atomics:

$$T_{AT}(N) = A_{AT}N + B_{AT}$$

Performance Model

Atomics vs transactions

Time to modify *N* vertices with atomics:

$$T_{AT}(N) = A_{AT}N + B_{AT}$$

Startup overheads

Performance Model

Atomics vs transactions

Time to modify *N* vertices with atomics:

$$T_{AT}(N) = A_{AT}N + B_{AT}$$

Overhead per vertex

Startup overheads

Performance Model
Atomics vs transactions

Time to modify *N* vertices with atomics:

$$T_{AT}(N) = A_{AT}N + B_{AT}$$

Overhead per vertex

Startup overheads

Time to modify *N* vertices with a transaction

$$T_{HTM}(N) = A_{HTM}N + B_{HTM}$$

Performance Model
Atomics vs transactions

Time to modify *N* vertices with atomics:

$$T_{AT}(N) = A_{AT}N + B_{AT}$$

Overhead per vertex

Startup overheads

Time to modify *N* vertices with a transaction

$$T_{HTM}(N) = A_{HTM}N + B_{HTM}$$

Overhead per vertex

Startup overheads

Performance Model
Atomics vs transactions

Time to modify *N* vertices with atomics:

$$T_{AT}(N) = A_{AT}N + B_{AT}$$

Overhead per vertex

Startup overheads

Time to modify *N* vertices with a transaction

$$T_{HTM}(N) = A_{HTM}N + B_{HTM}$$

Overhead per vertex

Startup overheads

We predict that:

$$B_{AT} < B_{HTM}$$

$$A_{AT} > A_{HTM}$$

Performance Model

Atomics vs transactions

Time to modify *N* vertices with atomics:

$$T_{AT}(N) = A_{AT}N + B_{AT}$$

Overhead per vertex

Startup overheads

Time to modify *N* vertices with a transaction

$$T_{HTM}(N) = A_{HTM}N + B_{HTM}$$

Overhead per vertex

Startup overheads

We predict that:

$$B_{AT} < B_{HTM}$$

$$A_{AT} > A_{HTM}$$

ansaction startup overheads dominate

Performance Model

Atomics vs transactions

Time to modify *N* vertices with atomics:

$$T_{AT}(N) = A_{AT}N + B_{AT}$$

Overhead per vertex

Startup overheads

Time to modify *N* vertices with a transaction

$$T_{HTM}(N) = A_{HTM}N + B_{HTM}$$

Overhead per vertex

Startup overheads

We predict that:

$$B_{AT} < B_{HTM}$$

$$A_{AT} > A_{HTM}$$

Transactions' cost grows slower

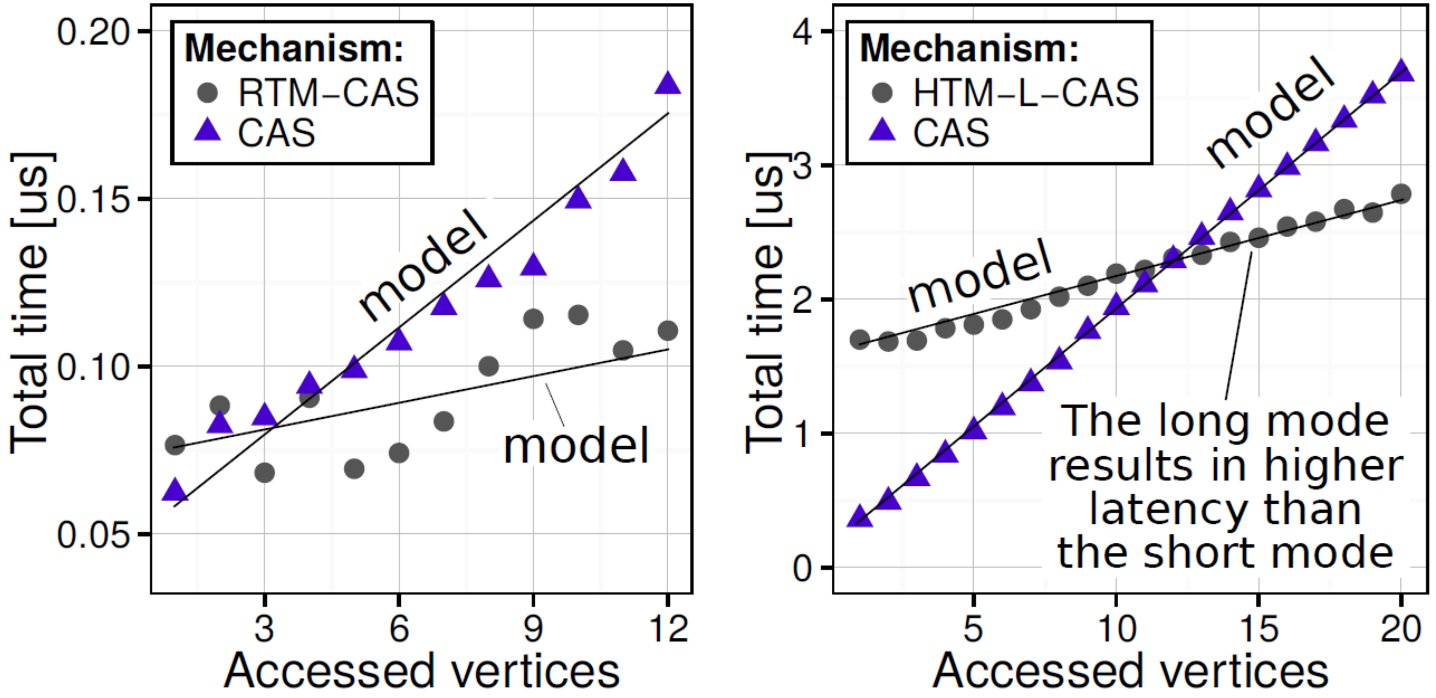
ansaction startup overheads dominate

PERFORMANCE MODEL

ATOMICS VS TRANSACTIONS

Performance Model

Atomics vs transactions



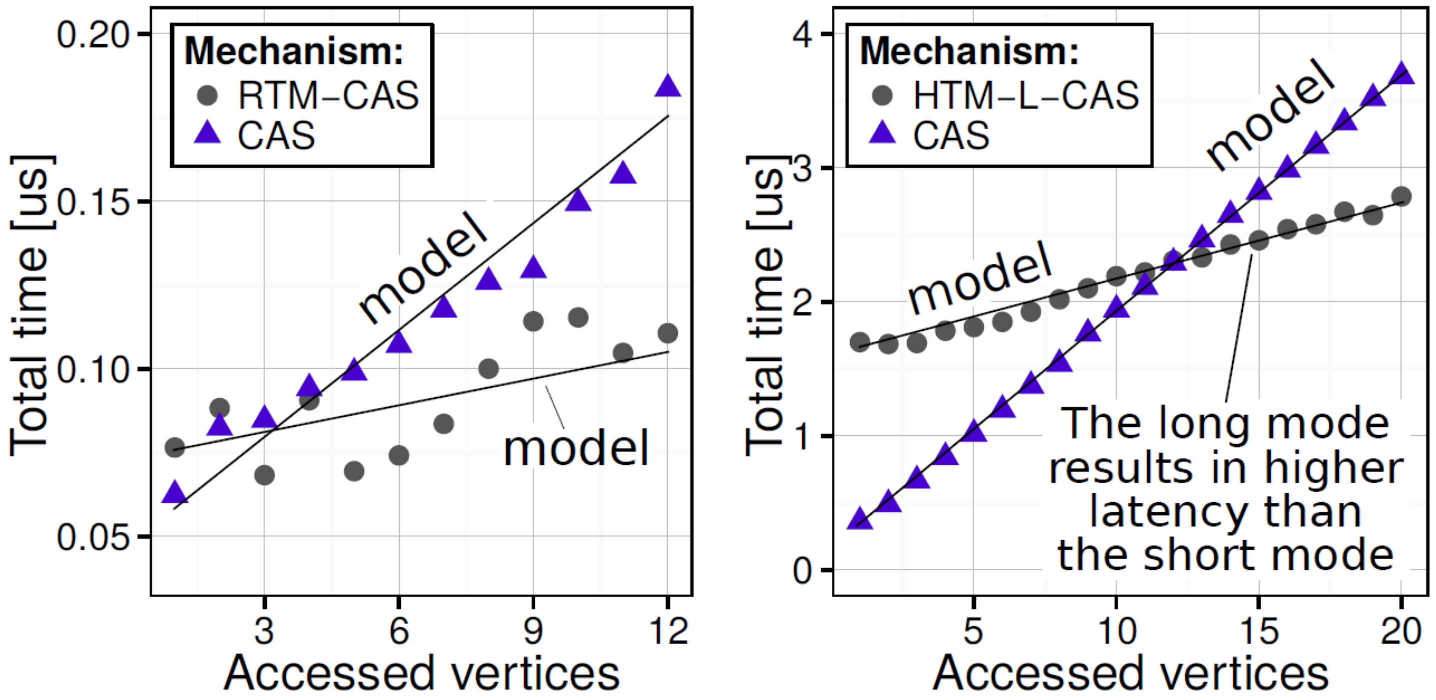
Performance Model

Atomics vs transactions

Indeed:

$$B_{AT} < B_{HTM}$$

$$A_{AT} > A_{HTM}$$



Performance Model

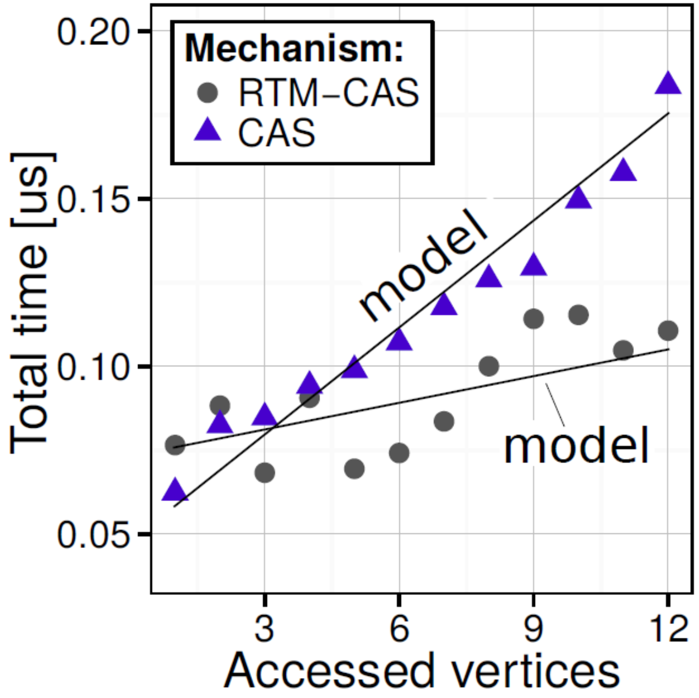
Atomics vs transactions

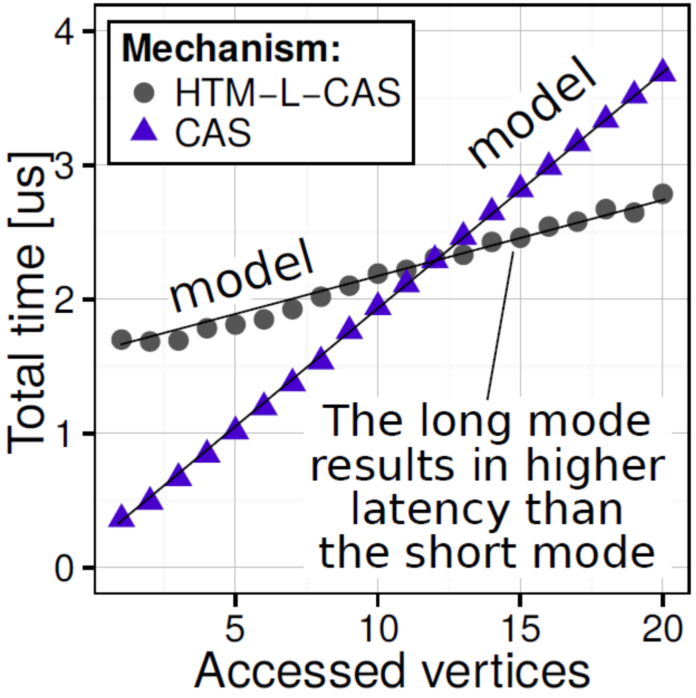
- Can we amortize HTM startup/commit overheads with larger transaction sizes?

Indeed:

$$B_{AT} < B_{HTM}$$

$$A_{AT} > A_{HTM}$$





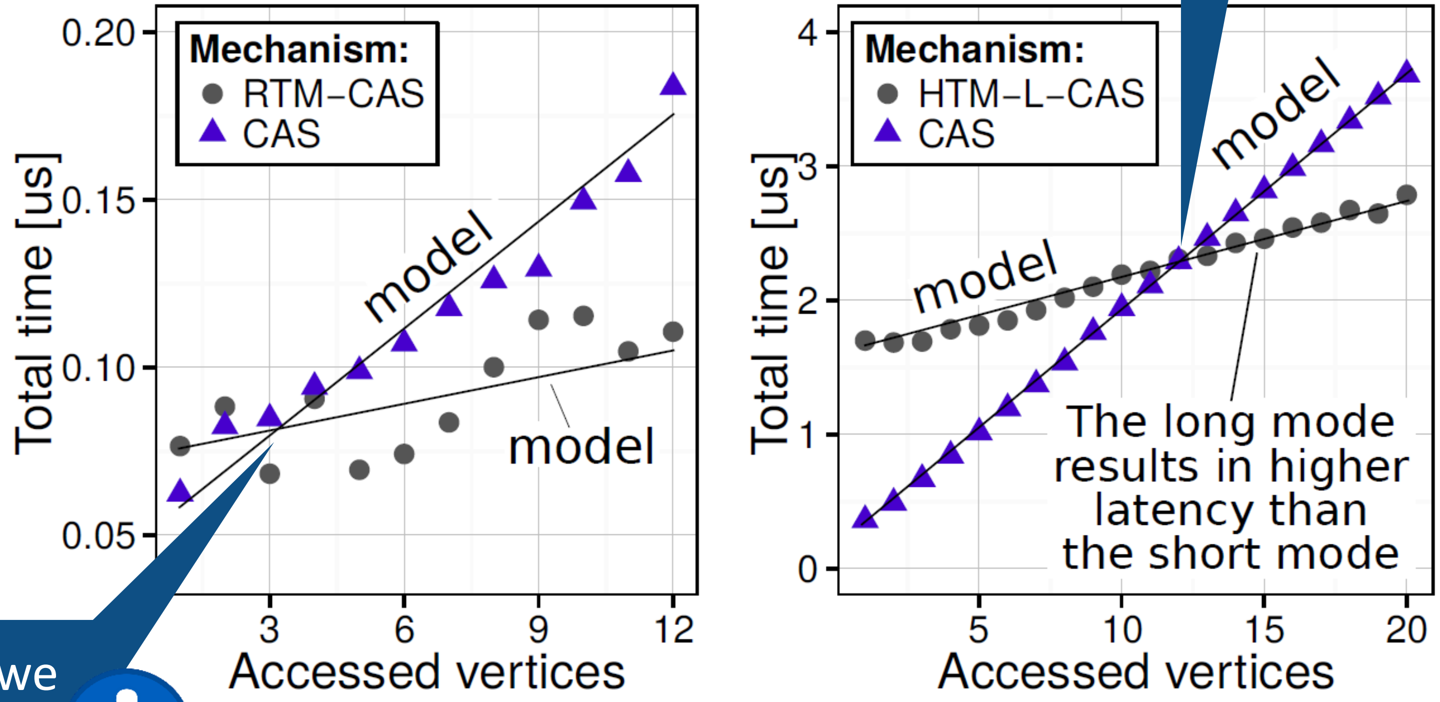
Performance Model
Atomics vs transactions

- Can we amortize HTM startup/commit overheads with larger transaction sizes?

Indeed:

$$B_{AT} < B_{HTM}$$
$$A_{AT} > A_{HTM}$$

Yes, we can!



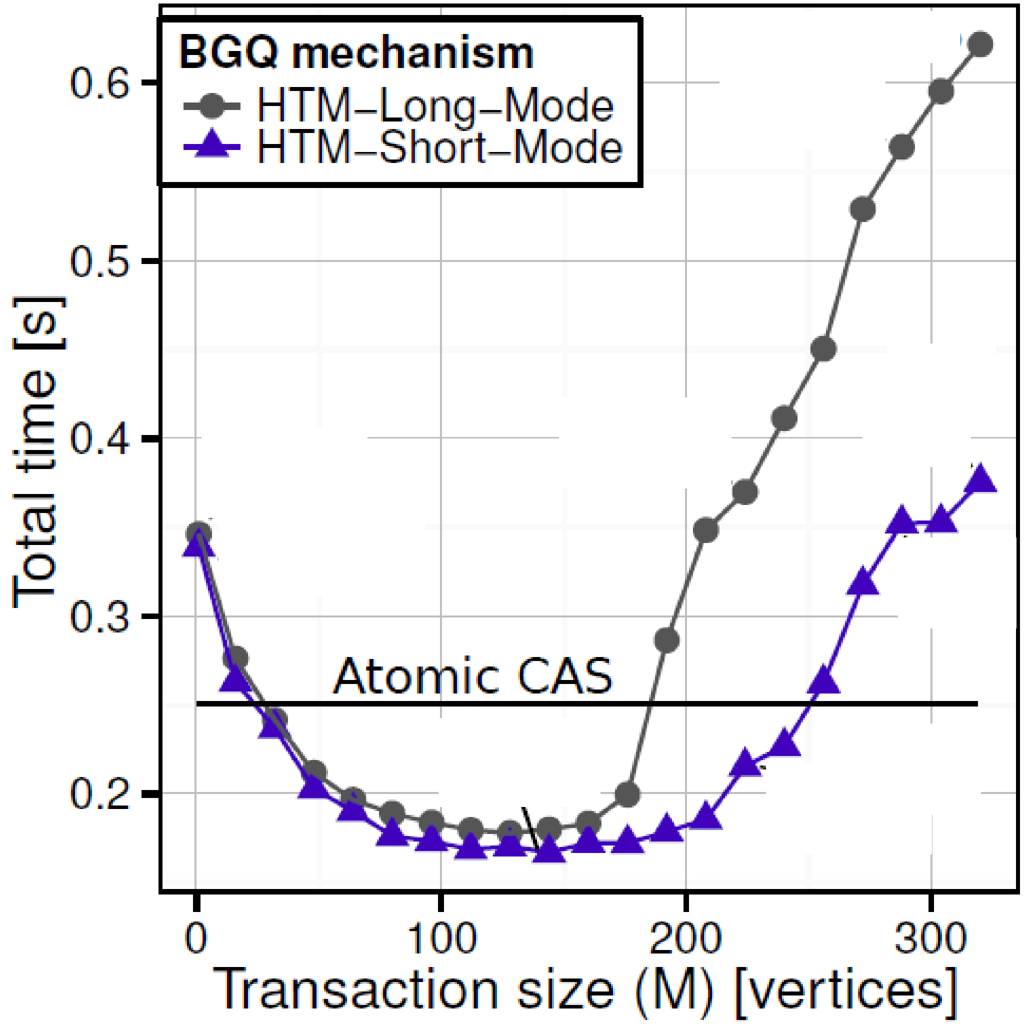
Yes, we can!

MULTI-VERTEX TRANSACTIONS

MARKING VERTICES AS VISITED

Multi-Vertex Transactions

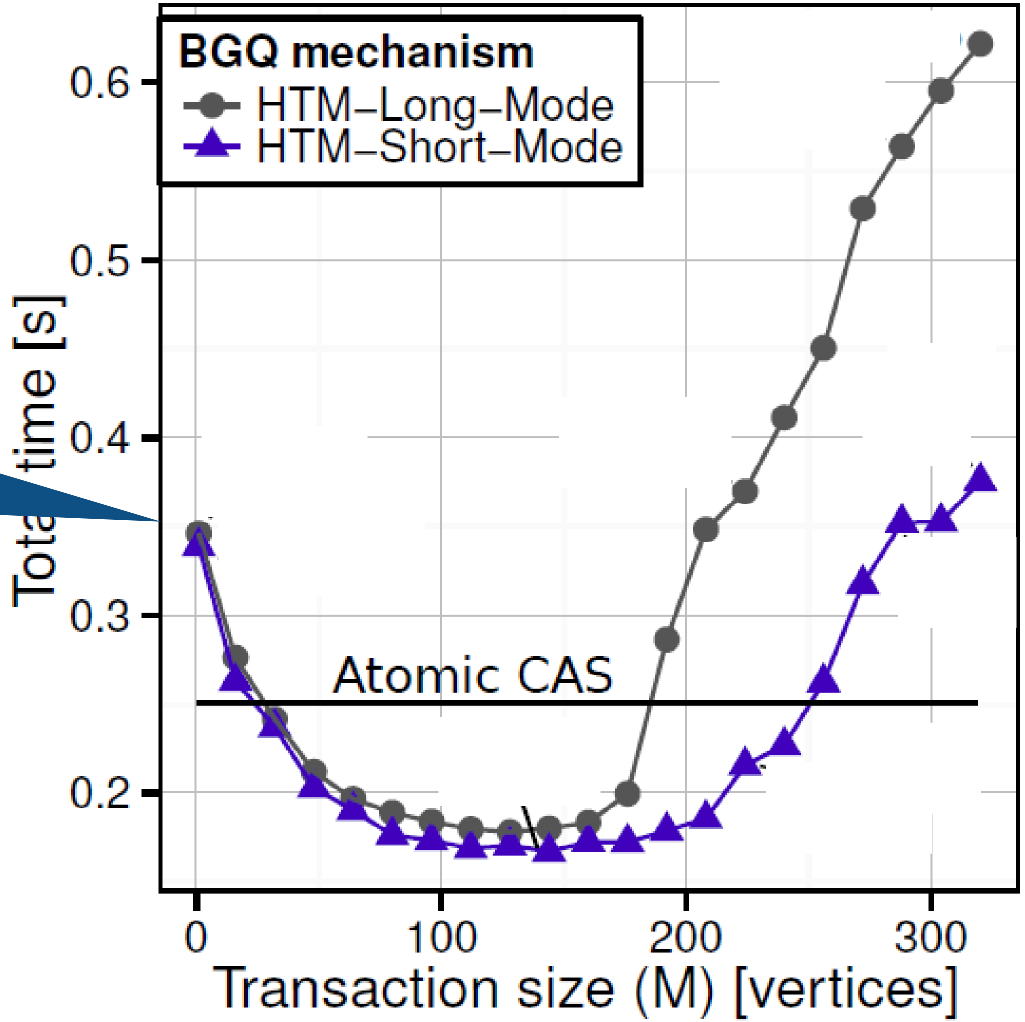
Marking vertices as visited



MULTI-VERTEX TRANSACTIONS

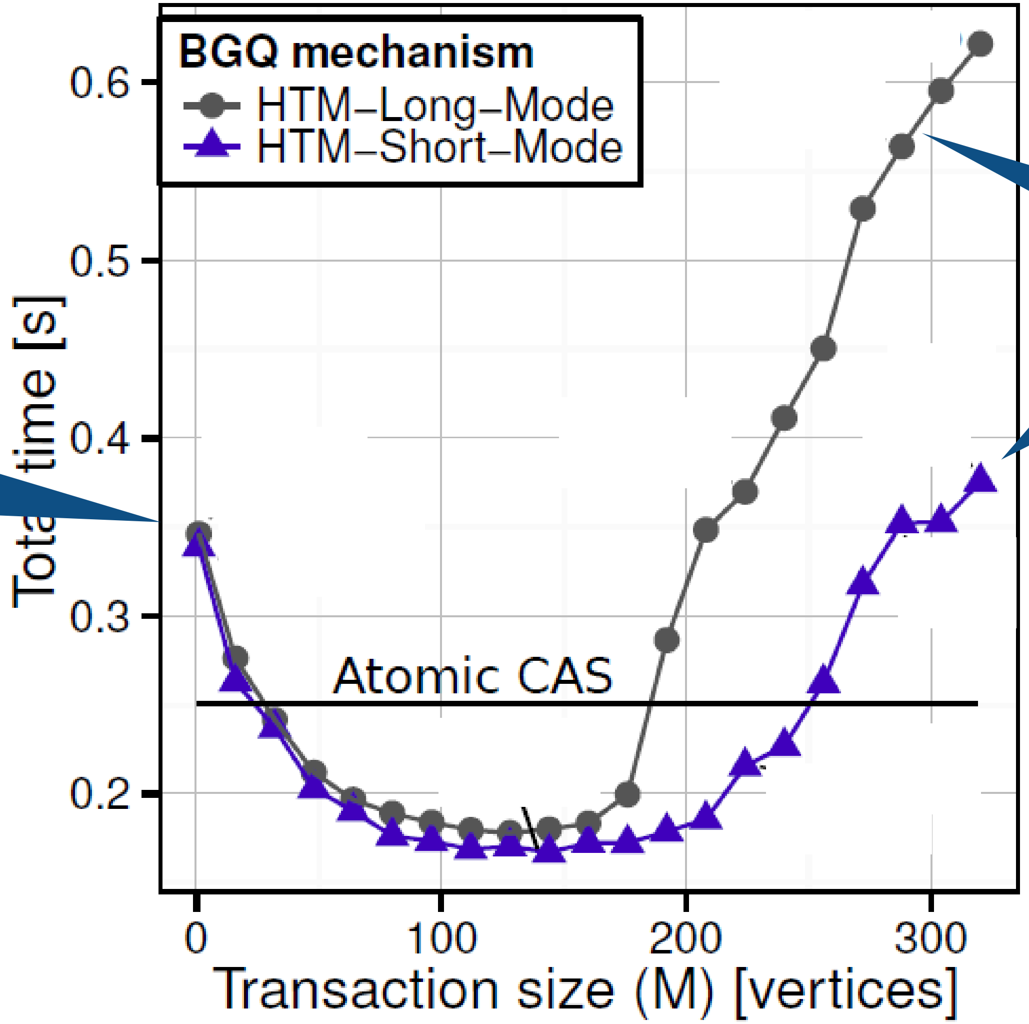
MARKING VERTICES AS VISITED

Startup and commit overheads



Multi-Vertex Transactions

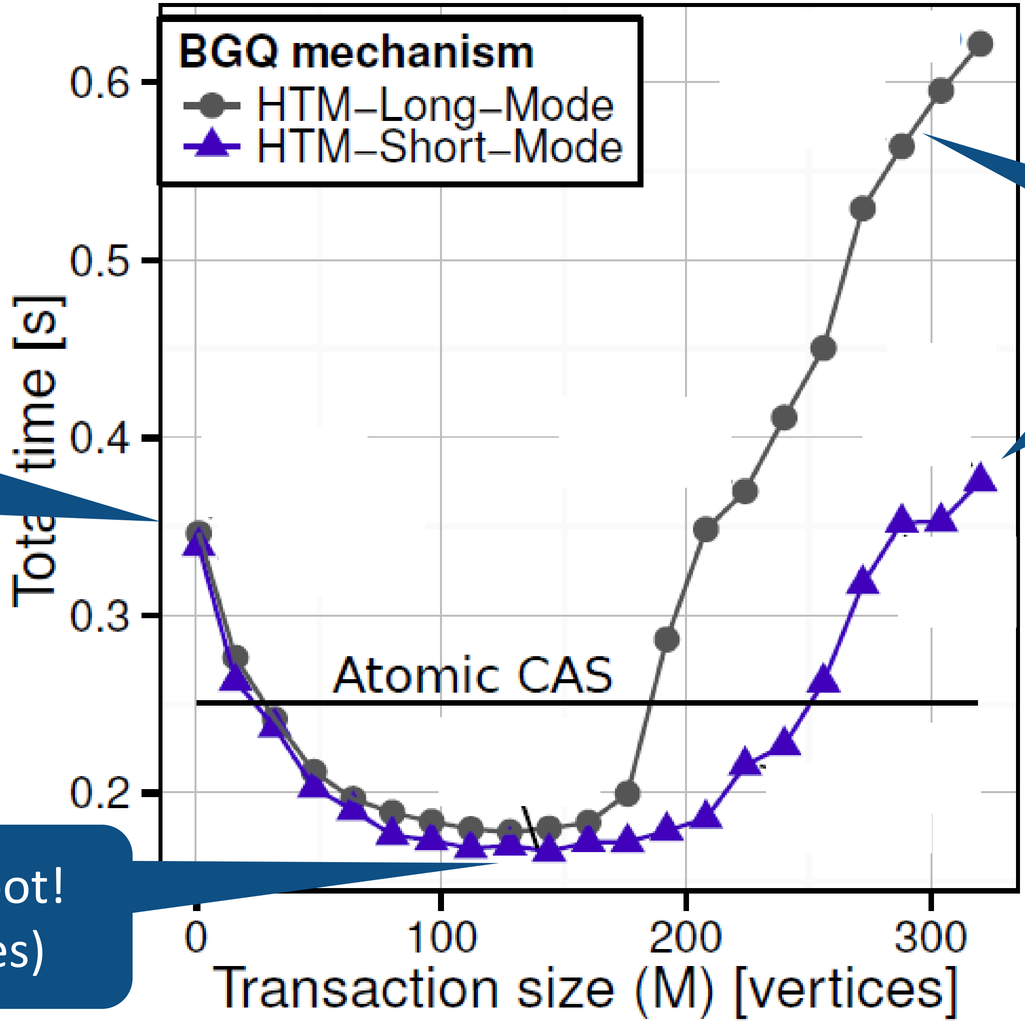
Marking vertices as visited



Startup and commit overheads

Abort and rollback overheads

Multi-Vertex Transactions
Marking vertices as visited



Startup and commit overheads

Abort and rollback overheads

The sweetspot! (144 vertices)