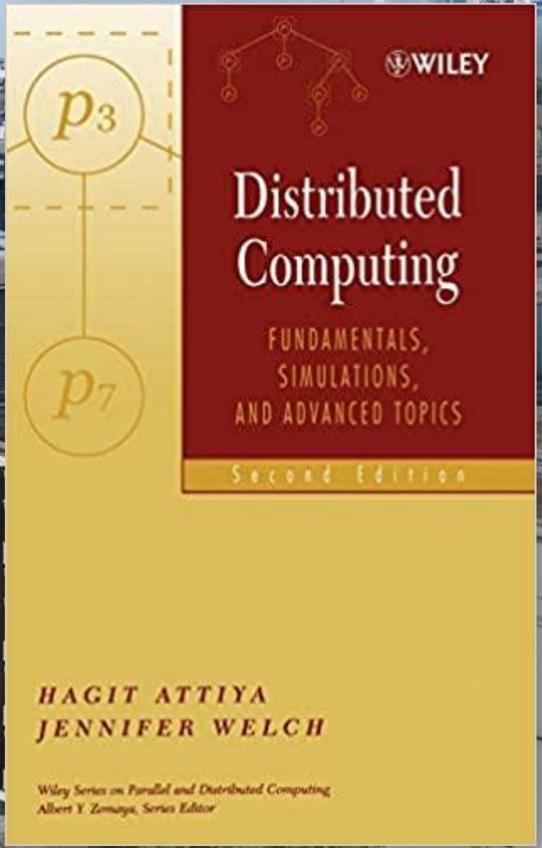
T. HOEFLER

WITH A. BARAK, Z. DREZNER, A. SHILOH, M. SNIR, B. GROPP, M. BESTA, S. DI GIROLAMO, K. TARANOV, G. KWASNIEWSKI, D. DE SENSI, T. SCHNEIDER, AND SPCL MEMBERS

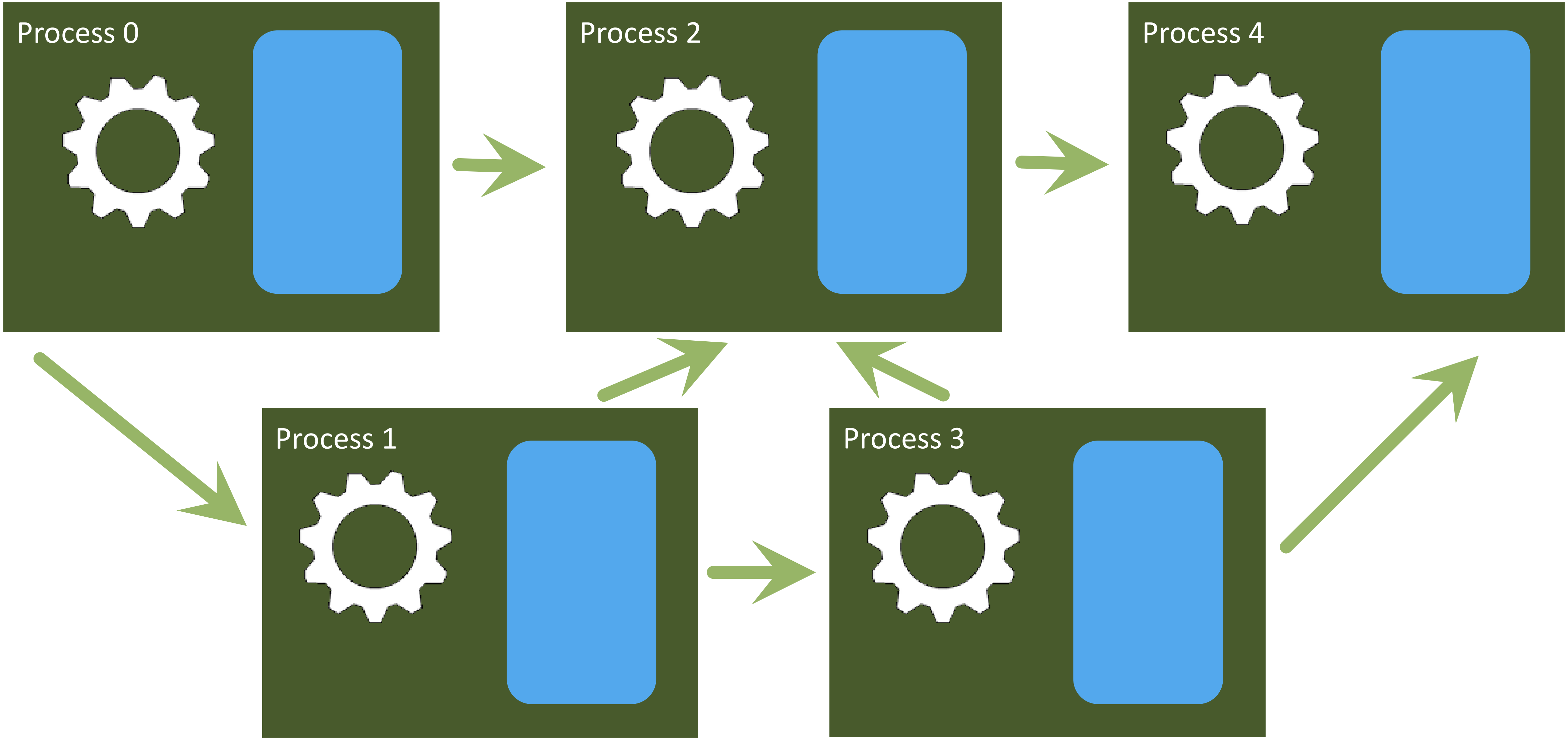
High-performance distributed memory systems - from supercomputers to data centers

Keynote at International Symposium on DIStributed Computing (DISC), Oct. 2020

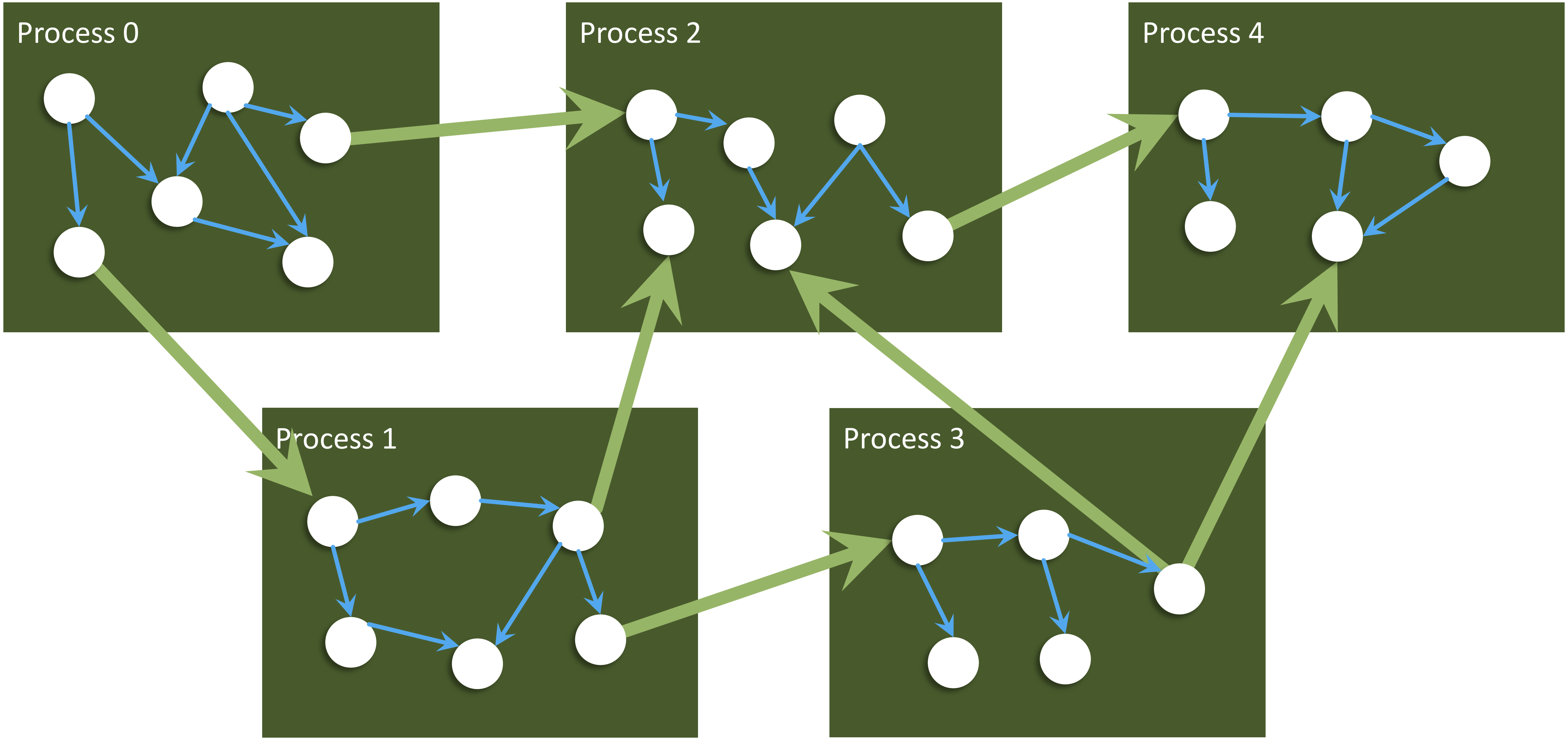


SPCL

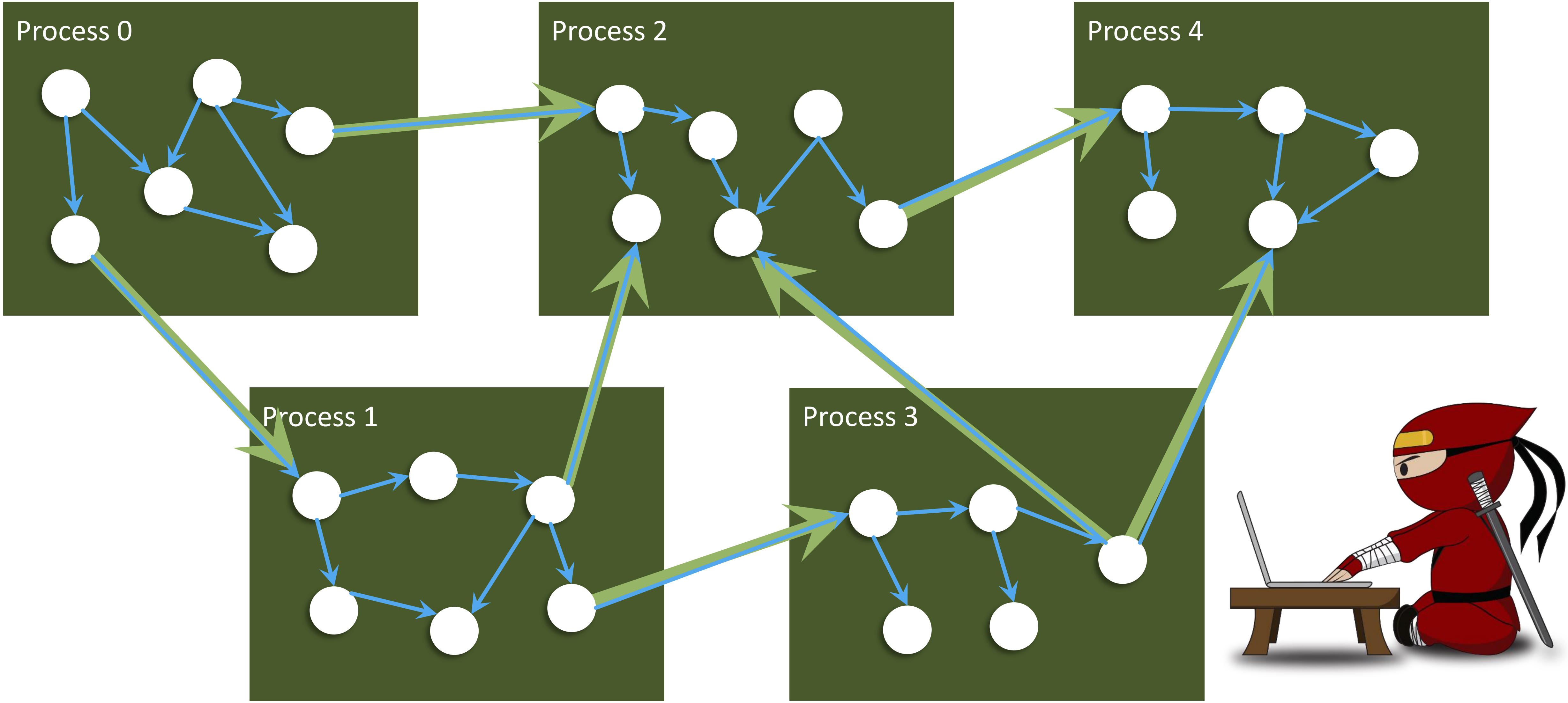
The Message Passing Interface – Communicating Processes



The Message Passing Interface – Communicating cDAGs



The Message Passing Interface – Distributed/Cut cDAGs



One step back – how to conquer the complexity of cDAGs?

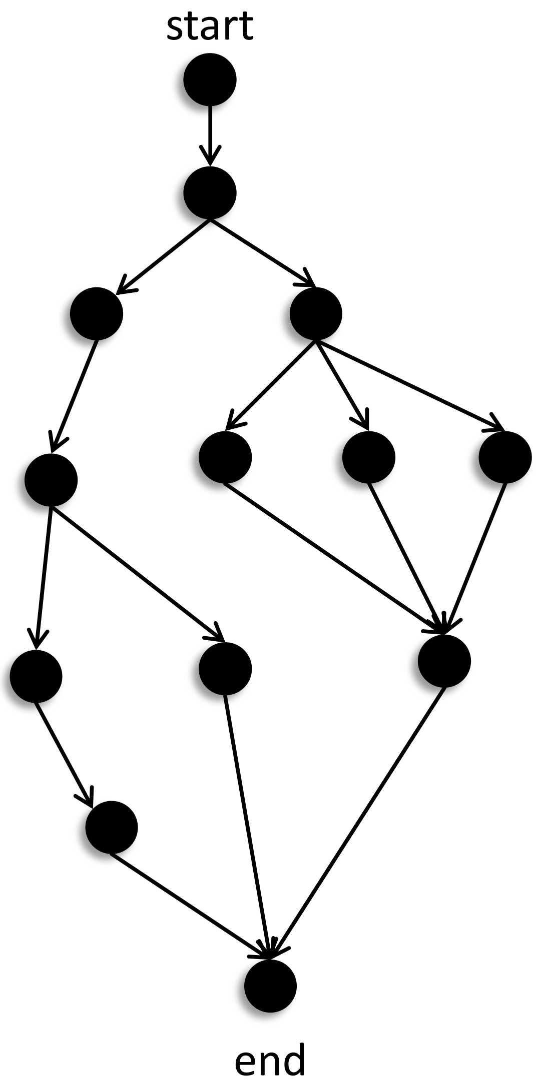
Work: $W = T_1$

Depth: $D = T_\infty$

Parallel efficiency: $E_p = \frac{T_1}{pT_p}$

Treewidth: usually small (2 for series parallel graphs)

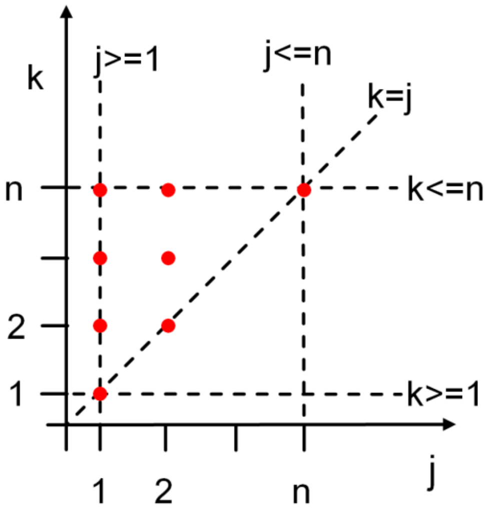
The generating program has an O(1) description



Side note: Analyzing cDAGs generated by programs – hard but doable!

```
for (j = 1; j <= n; j = j*2)
    for (k = j; k <= n; k = k++)
        operation(x,y)
```

Affine loop model



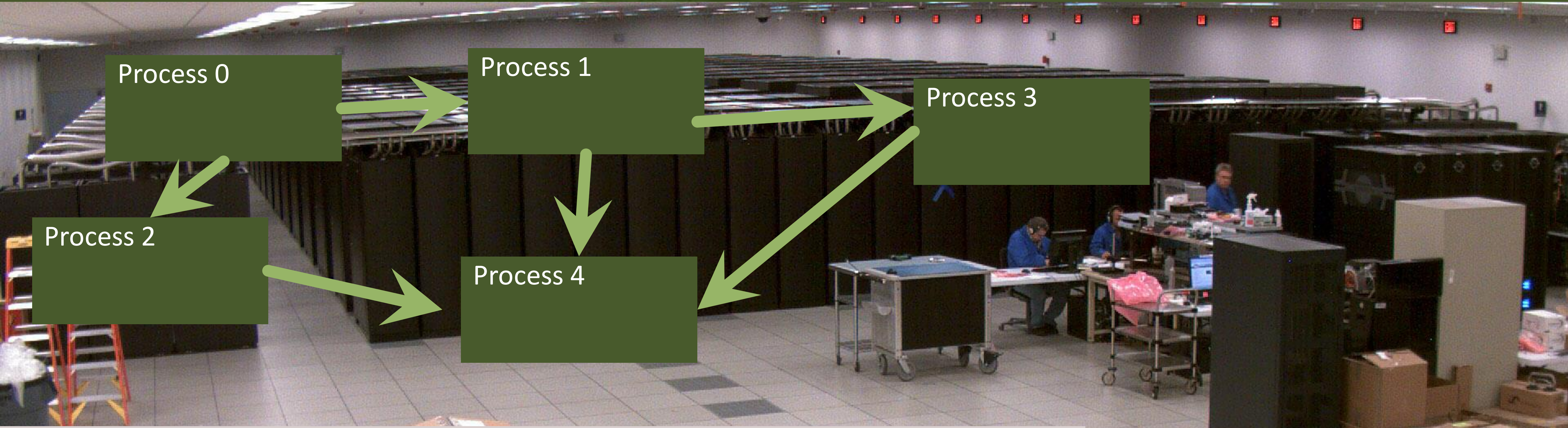
$j \in [1, n] \quad k \in [j, n]$

$$
\begin{aligned}
&\texttt{while}(c_1^T x < g_1)\ \{\\
&\quad x = A_1 x + b_1;\\
&\quad \texttt{while}(c_2^T x < g_2)\ \{\\
&\qquad \ldots\\
&\qquad\ x = A_{k-1} x + b_{k-1};\\
&\qquad\ \texttt{while}(c_k^T x < g_k)\ \{\\
&\qquad\qquad x = A_k x + b_k;\\
&\qquad\qquad \texttt{while}(c_{k+1}^T x < g_{k+1})\ \{\ldots\ \}\\
&\qquad\qquad x = U_k x + v_k;\ \}\\
&\qquad\ x = U_{k-1} x + v_{k-1};\\
&\qquad \ldots\}\\
&\quad x = U_1 x + v_1;\}
\end{aligned}
$$

$N = (n+1)\log_2 n - n + 2$

Automatic work-depth analysis for MPI (and other) programs!

TH, Grzegorz Kwasniewski: Automatic Complexity Analysis of Explicitly Parallel Programs, SPAA'14



UIUC/NCSA Blue Waters in 2012

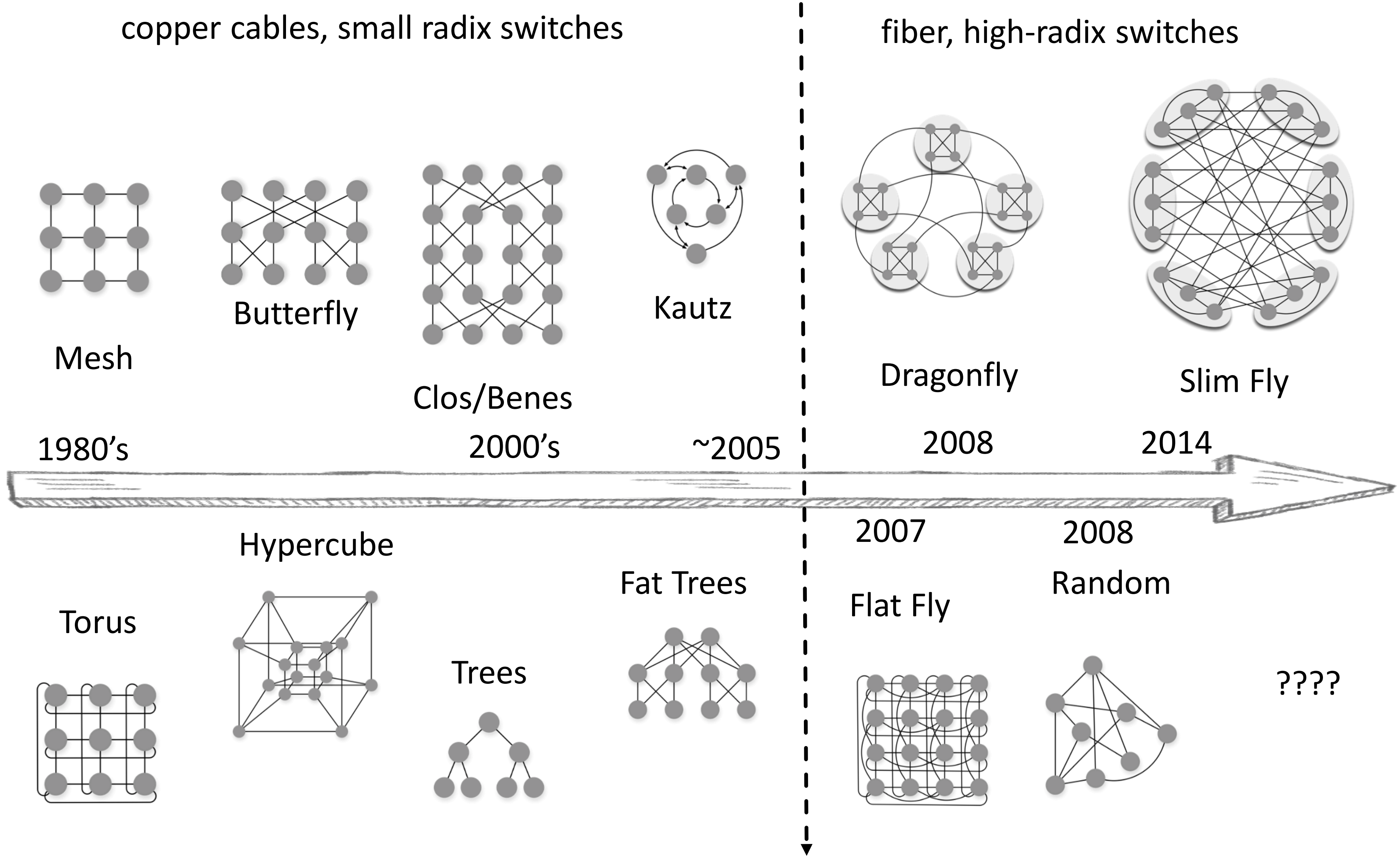
Total TCO ~$500M

49,000 AMD Bulldozer CPUs – 0.5 EB storage

Where do these processes go?

Understand supercomputer network architecture!

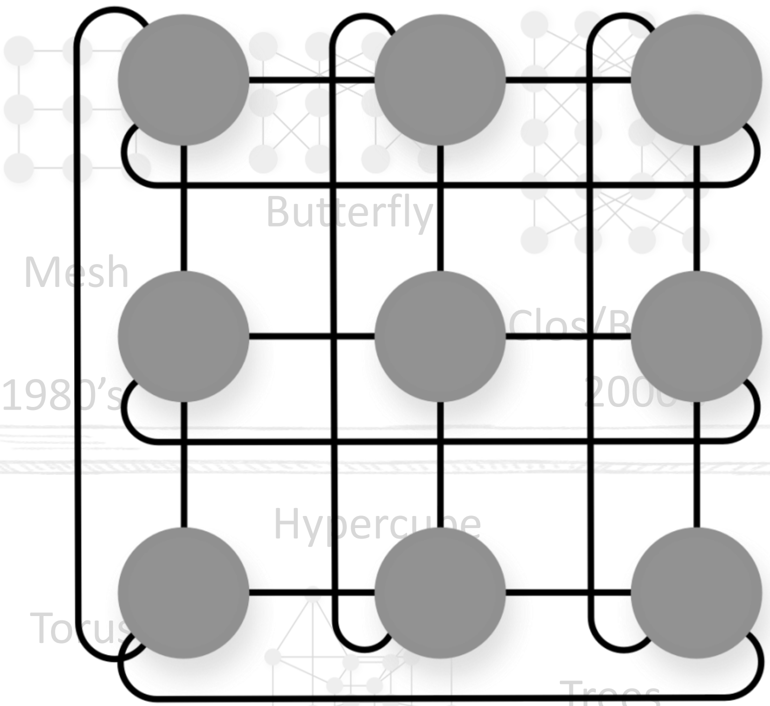
A BRIEF HISTORY OF NETWORK TOPOLOGIES

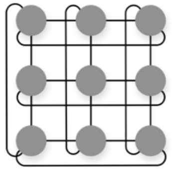


A BRIEF HISTORY OF NETWORK TOPOLOGIES

copper cables, small radix switches

fiber, high-radix switches



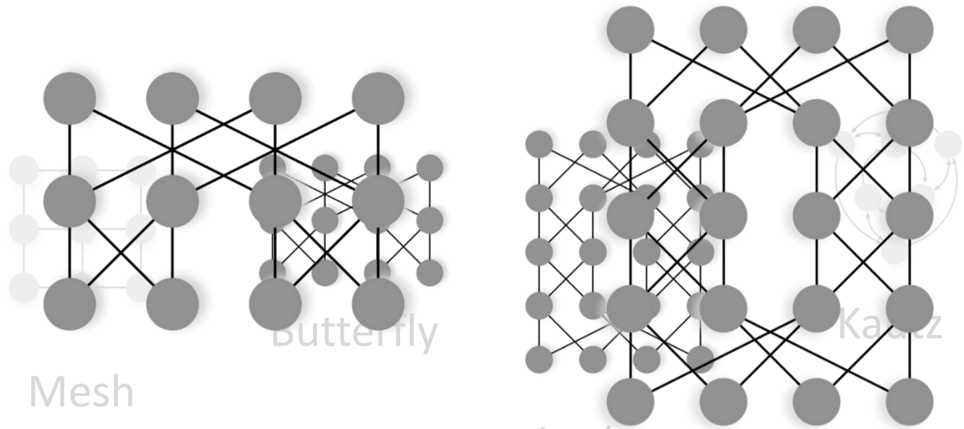


$$
\begin{aligned}
\text{Bandwidth} &= 2\sqrt[d]{N^{d-1}} \\
\text{Latency} &= \tfrac{d}{2}\sqrt[d]{N} \\
\text{Radix} &= 2d
\end{aligned}
$$

2008
2014

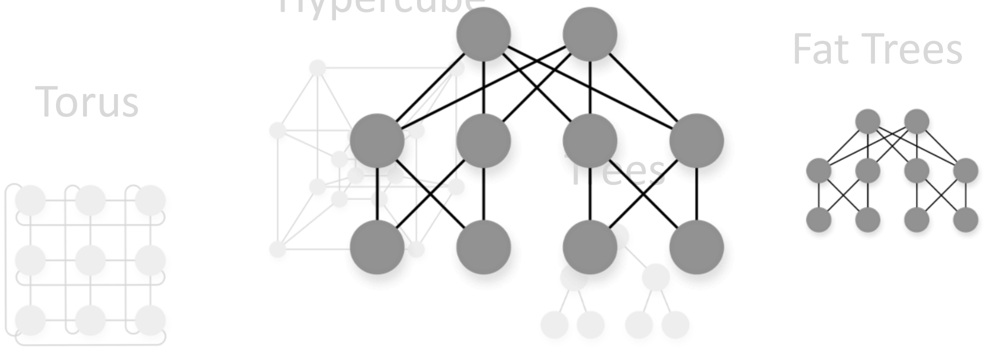
A BRIEF HISTORY OF NETWORK TOPOLOGIES

copper cables, small radix switches



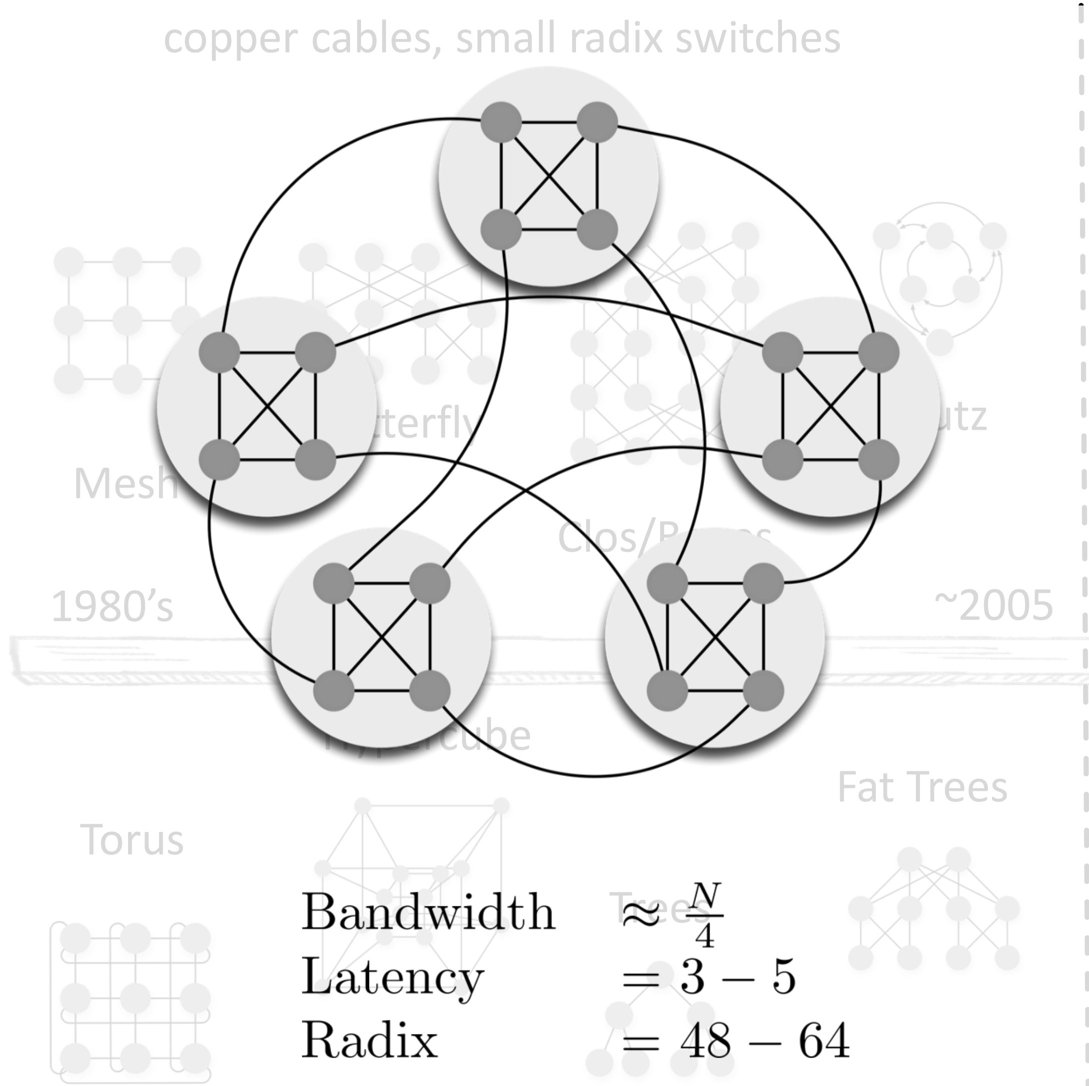
1980's 2000's ~2005

$$
\begin{aligned}
\text{Bandwidth} &= \frac{N}{2} \\
\text{Latency} &= 2\log_2 N \\
\text{Radix} &= 4
\end{aligned}
$$



2008 2014

A brief history of network topologies



$$\begin{aligned}\text{Bandwidth} &\approx \frac{N}{4}\\ \text{Latency} &= 3-5\\ \text{Radix} &= 48-64\end{aligned}$$

An In-Depth Analysis of the Slingshot Interconnect

Daniele De Sensi, Department of Computer Science, ETH Zurich, ddesensi@ethz.ch
Salvatore Di Girolamo, Department of Computer Science, ETH Zurich, salvatore.digirolamo@inf.ethz.ch
Kim H. McMahon, Hewlett Packard Enterprise, kim.mcmahon@hpe.com
Duncan Roweth, Hewlett Packard Enterprise, duncan.roweth@hpe.com
Torsten Hoefler, Department of Computer Science, ETH Zurich, torsten.hoefler@inf.ethz.ch

[cs.DC] 20 Aug 2020

***Abstract*—The interconnect is one of the most critical components in large scale computing systems, and its impact on the performance of applications is going to increase with the system size. In this paper, we will describe SLINGSHOT, an interconnection network for large scale computing systems. SLINGSHOT is based on high-radix switches, which allow building exascale and hyperscale datacenters networks with at most three switch-to-switch hops. Moreover, SLINGSHOT provides efficient adaptive routing and congestion control algorithms, and highly tunable traffic classes. SLINGSHOT uses an optimized Ethernet protocol, which allows it to be interoperable with standard Ethernet devices while providing high performance to HPC applications. We analyze the extent to which SLINGSHOT provides these features, evaluating it on microbenchmarks and on several applications from the datacenter and AI worlds, as well as on HPC applications. We find that applications running on SLINGSHOT are less affected by congestion compared to previous generation networks.**

***Index Terms*—interconnection network, dragonfly, exascale, datacenters, congestion**

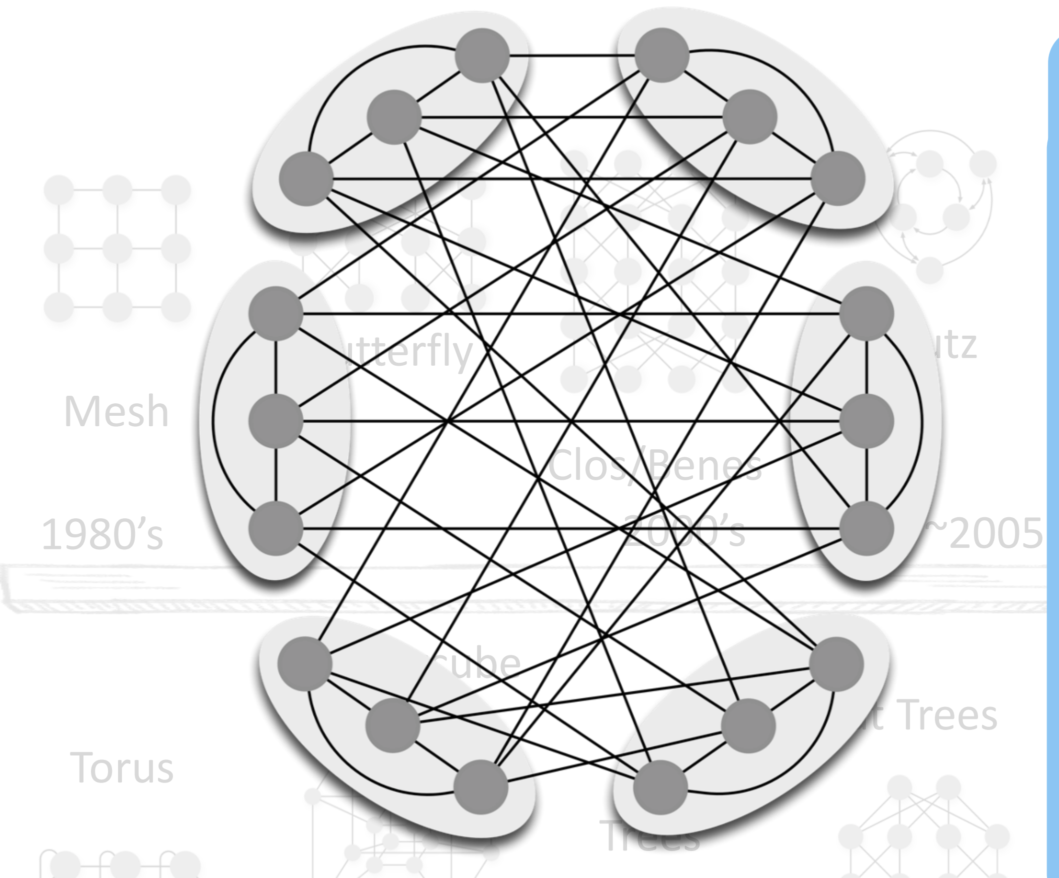
world. Due to the wide adoption of Ethernet in datacenters, interconnection networks should be compatible with standard Ethernet, so that they can be efficiently integrated with standard devices and storage systems. Moreover, many data center workloads are latency-sensitive. For such applications, *tail latency* is much more relevant than the best case or average latency. For example, web search nodes must provide 99^{th} percentile latencies of a few milliseconds [4]. This is also a relevant problem for HPC applications, whose performance may strongly depend on messages latency, especially when using many global or small messages synchronizations. Despite the efforts in improving the performance of interconnection networks, tail latency still severely affect large HPC and data center systems [4]–[7].

To address these issues, Cray[1] recently designed the SLINGSHOT interconnection network. SLINGSHOT will power all

A BRIEF HISTORY OF NETWORK TOPOLOGIES

copper cables, small radix switches

fiber, high-radix switches



$$\begin{aligned} \text{Bandwidth} &\approx \frac{N}{4} \\ \text{Latency} &= 2-4 \\ \text{Radix} &= k \end{aligned}$$

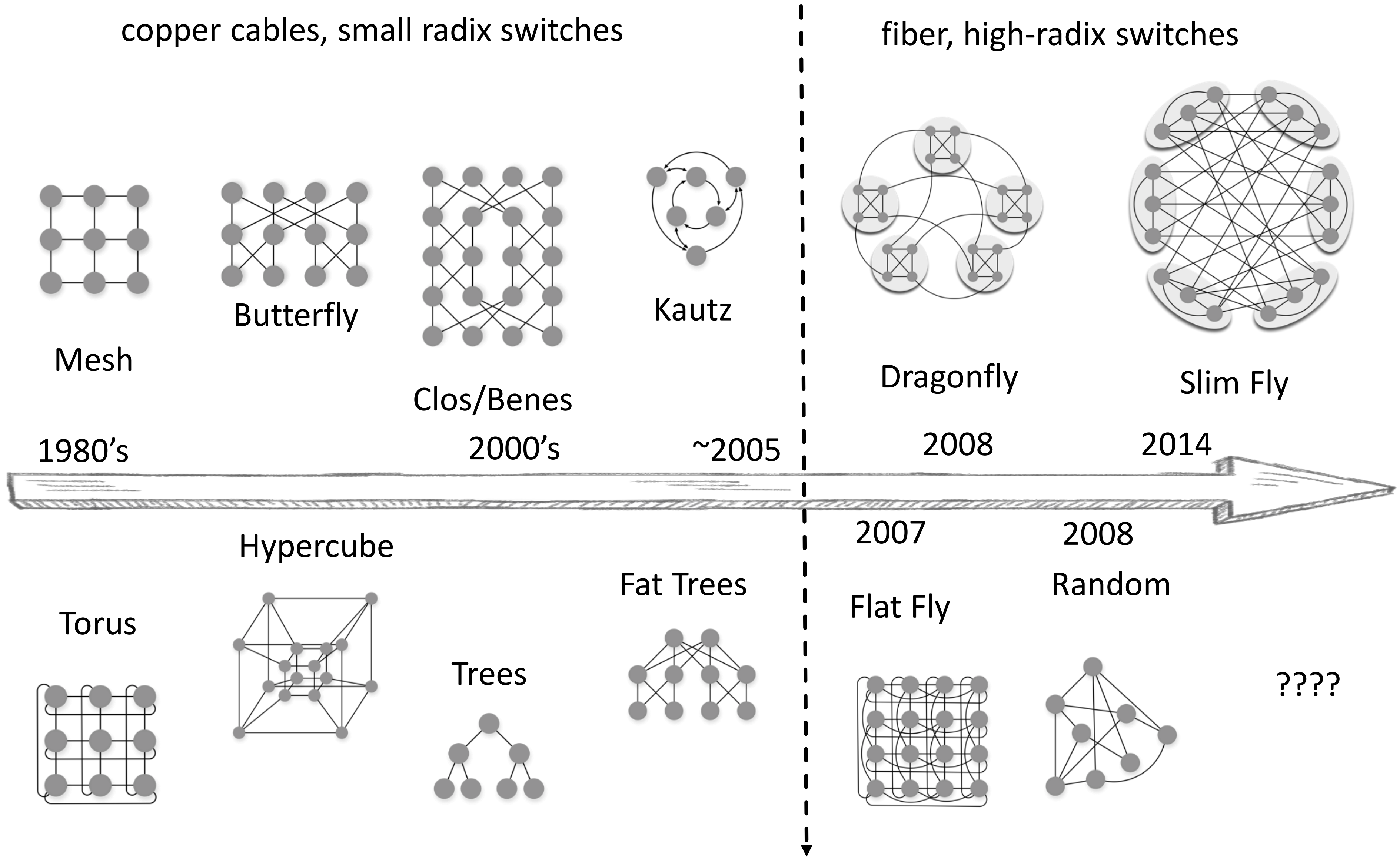
Key insight:

"It's the diameter, stupid"

Lower diameter:
→ Fewer cables traversed
→ Fewer cables needed
→ Fewer routers needed

Cost and energy savings:
→ Up to 50% over Fat Tree
→ Up to 33% over Dragonfly

A BRIEF HISTORY OF NETWORK TOPOLOGIES



Back to MPI processes – mapping them to nodes!

MPI programs cannot learn about the topology! They specify their communication topology instead and let the library map.

map to

Topology mapping is NP hard ☹

Chapter 5

An Overview of Topology Mapping Algorithms and Techniques in High-Performance Computing

Torsten Hoefler, Emmanuel Jeannot, Guillaume Mercier

$$Dilation(\Gamma) = \sum_{u,v \in V_G} \omega_G(uv) \cdot Dilation(uv)$$

$$Congestion(\Gamma) = \max_e Congestion(e)$$

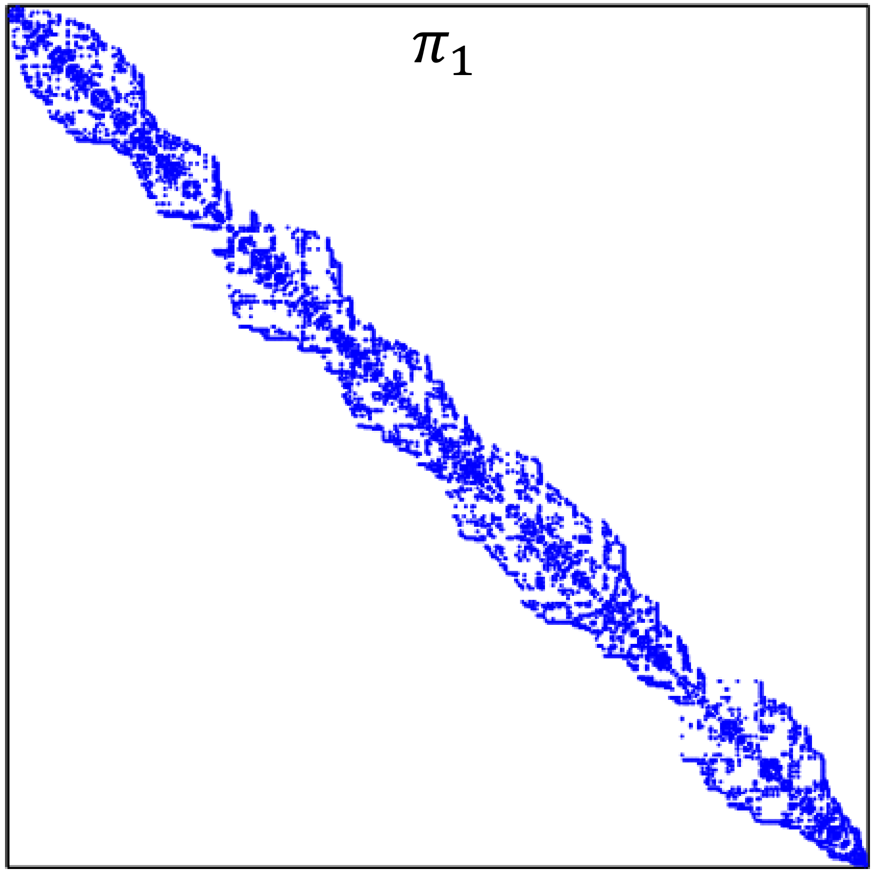
Measure of communication work!

Lower bound to the time of communication!

TH and Marc Snir: Generic Topology Mapping Strategies for Large-scale Parallel Architectures , ACM ICS'11

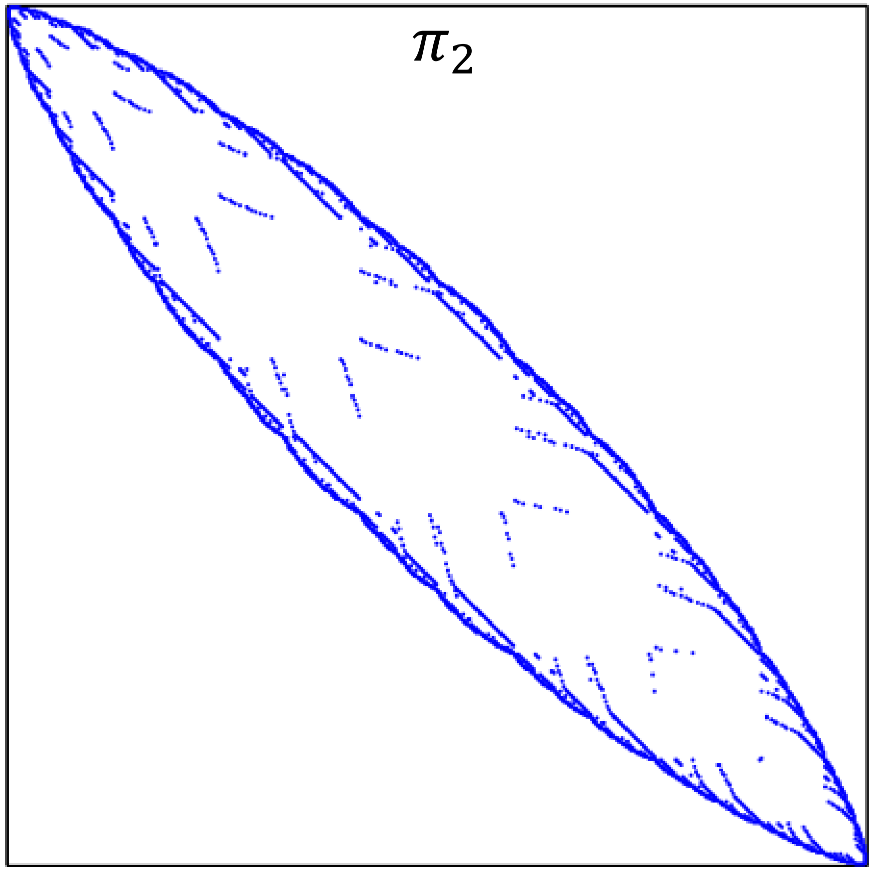
A new topology mapping heuristic – minimize bandwidth of both graphs

Application Graph (SpMV)



RCM Permutation

Network Graph (8x8x8 torus)

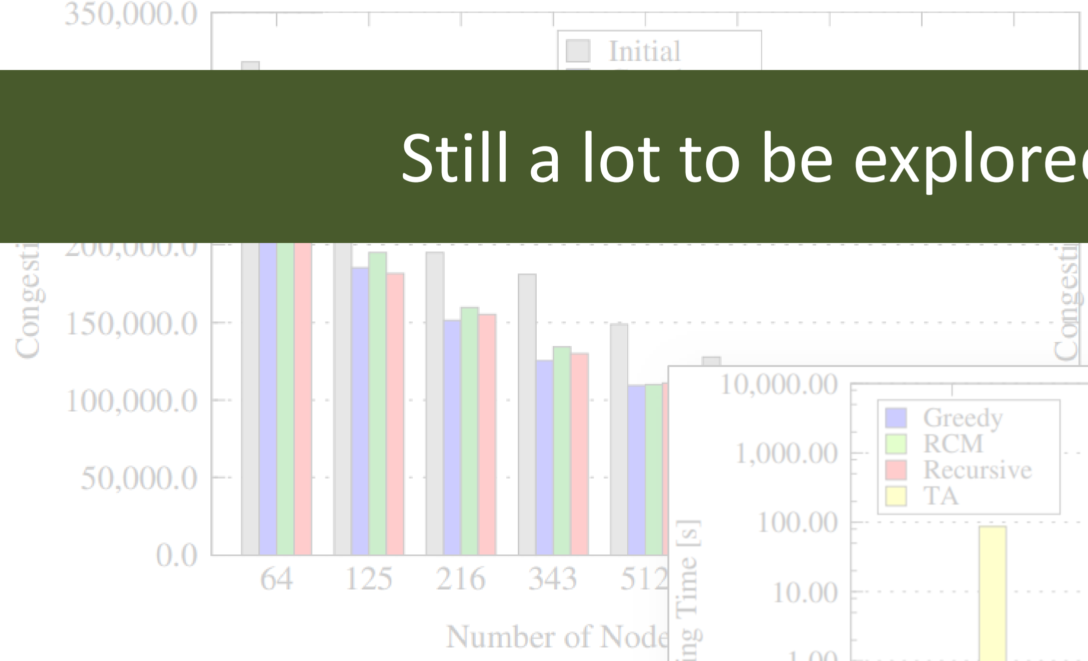


RCM Permutation

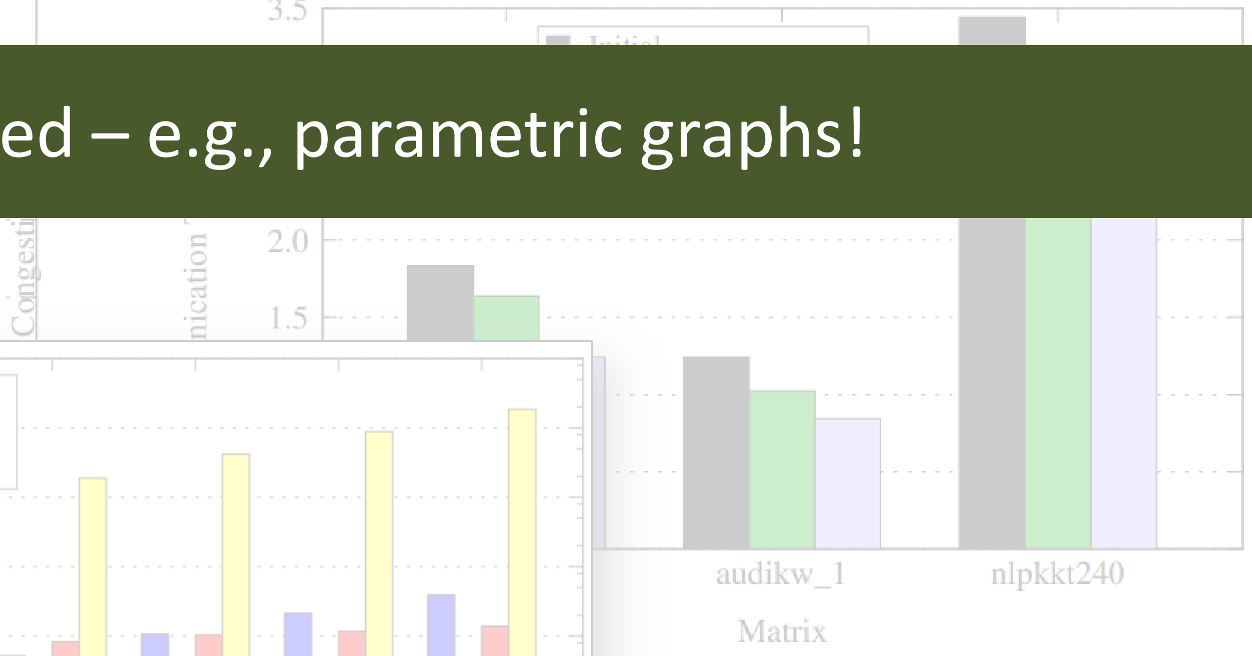
TH and Marc Snir: Generic Topology Mapping Strategies for Large-scale Parallel Architectures , ACM ICS'11

A new topology mapping heuristic – minimize bandwidth of both graphs

3D Torus

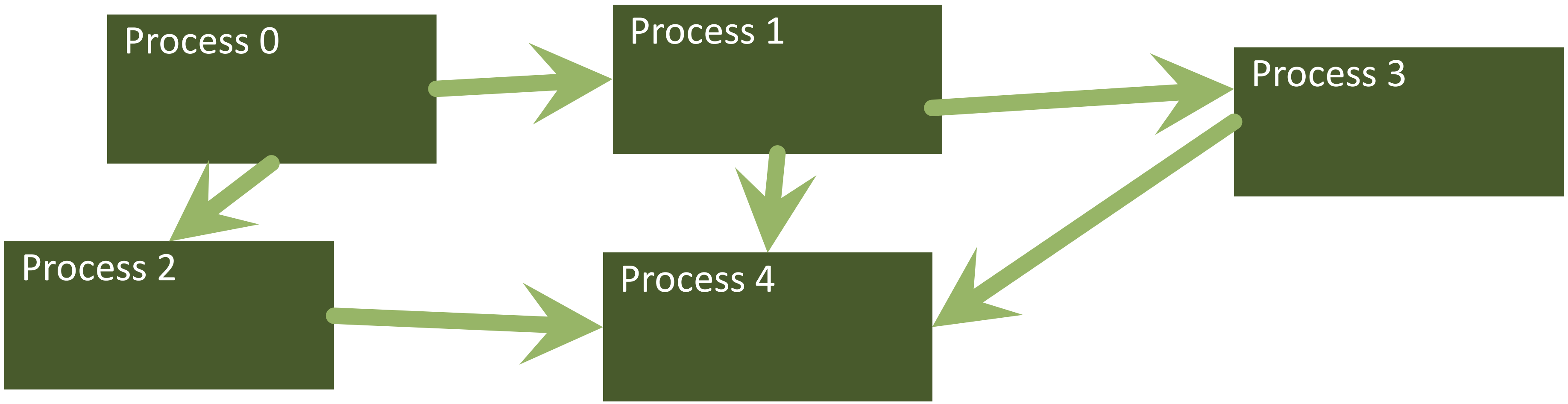


Real execution on a BlueGene/P (512 nodes, 3D Torus)



Still a lot to be explored – e.g., parametric graphs!

Assume processes are mapped nicely – structured communication



The generating program has an O(1) description → it has a lot of structure!

Bulk synchronous (single global state) thinking model works great for humans like me.
Communications there can often be described algorithmically as collective operations – MPI does so!

MPI_Allgather	**MPI_Allgatherv**	**MPI Allreduce**	**MPI_Alltoall**	**MPI_Alltoallv**
MPI_Alltoallw	**MPI_Barrier**	**MPI_Bcast**	**MPI_Gather**	**MPI_Gatherv**
MPI_Reduce	**MPI_Scatter**	**MPI_Scatterv**	**MPI_Exscan**	**MPI_Reduce_local**
MPI_Reduce_scatter	**MPI_Scan**	**MPI_Neighbor_allgather**	**MPI_Neighbor_alltoall**	

TH, D. Moor: Energy, Memory, and Runtime Tradeoffs for Implementing Collective Communication Operations

LogP – an accurate network model!

The LogP model family and the LogGOPS model [1]


A new parallel machine model reflects the critical technology trends underlying parallel computers

LogP

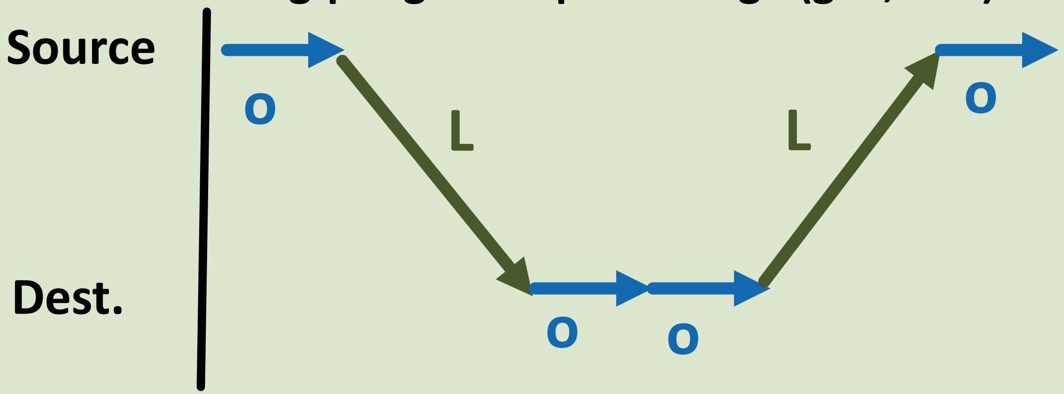
A PRACTICAL MODEL of PARALLEL COMPUTATION

OUR GOAL IS TO DEVELOP A MODEL OF PARALLEL COMPUTATION THAT WILL serve as a basis for the design and analysis of fast, portable parallel algorithms, such as algorithms that can be implemented effectively on a wide variety of current and future parallel machines. If we look at the body of parallel algorithms developed under current parallel models, many are impractical because they exploit artificial factors not present in any rea-

PRAM consists of a collection of processors which compute synchronously in parallel and communicate with a global random access

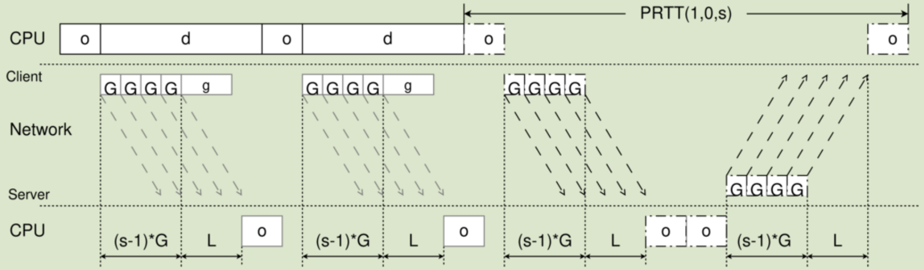
David E. Culler, Richard M. Karp, David Patterson, Abhijit Sahay, Eunice E. Santos, Klaus Erik Schauser, Ramesh Subramonian, and Thorsten von Eicken

Ping-pong in simplified LogP (g<o, P=2)



Finding LogGOPS parameters

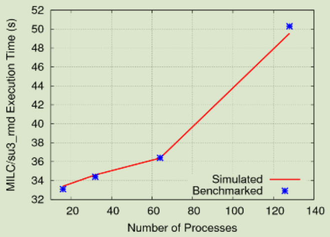
Netgauge [2], model from first principles, fit to data using special kernels

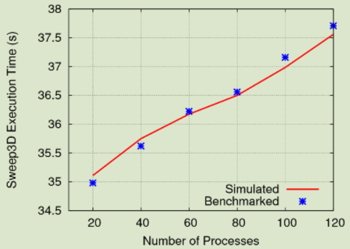


Large scale LogGOPS Simulation

LogGOPSim [1], simulates LogGOPS with 10 million MPI ranks

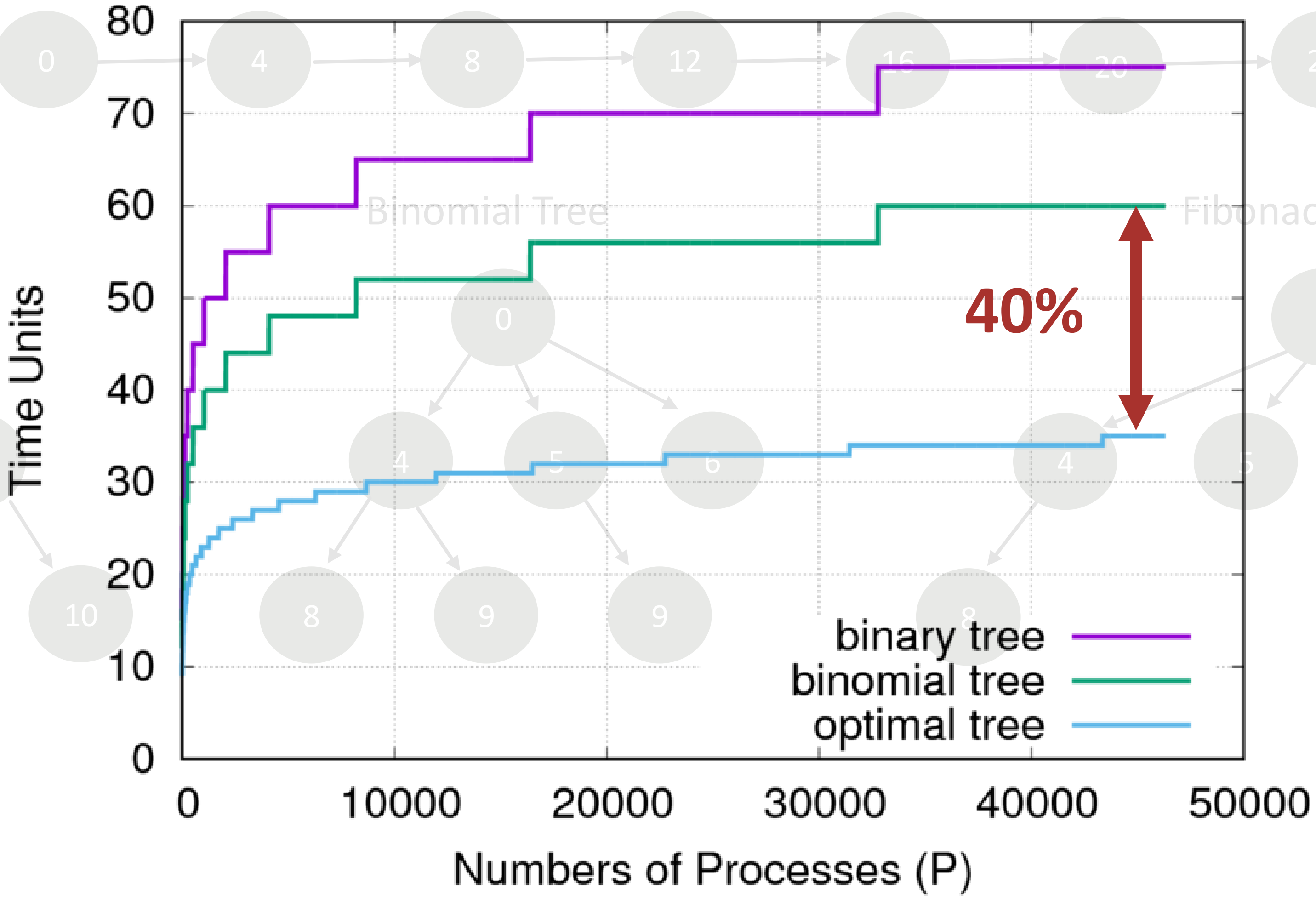
<5% error





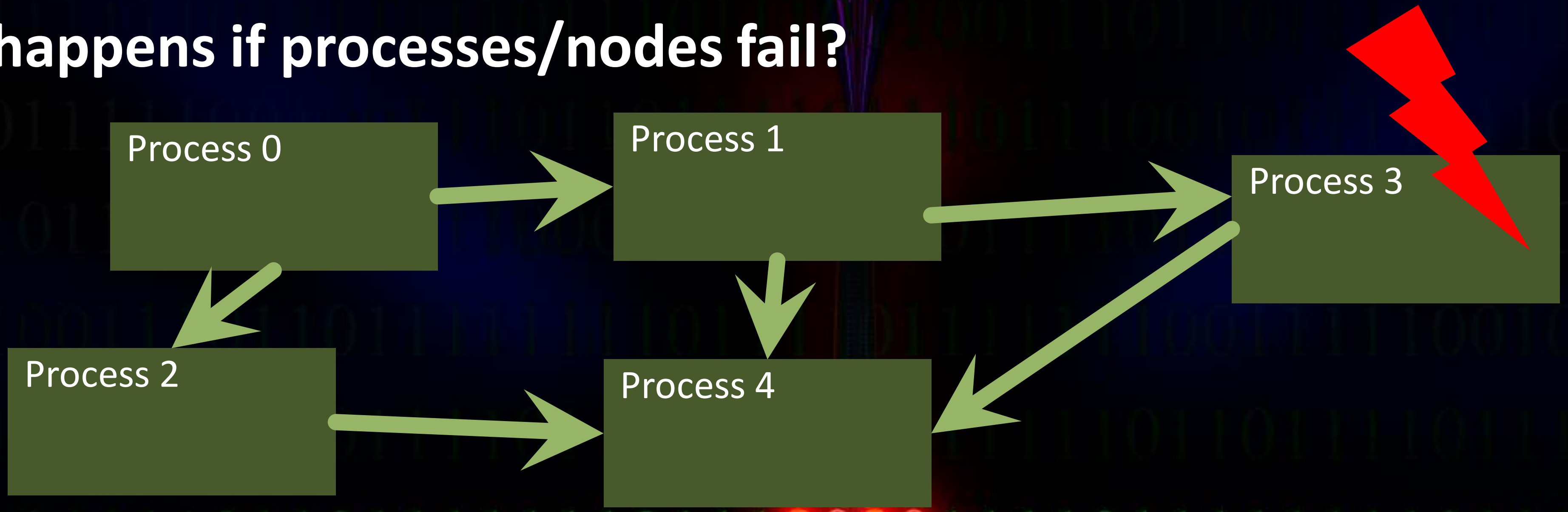
[1]: TH, T. Schneider and A. Lumsdaine: LogGOPSim - Simulating Large-Scale Applications in the LogGOPS Model, LSAP 2010, https://spcl.inf.ethz.ch/Research/Performance/LogGOPSim/
[2]: TH, T. Mehlan, A. Lumsdaine and W. Rehm: Netgauge: A Network Performance Measurement Framework, HPCC 2007, https://spcl.inf.ethz.ch/Research/Performance/Netgauge/

Designing an optimal small-message broadcast algorithm in LogP



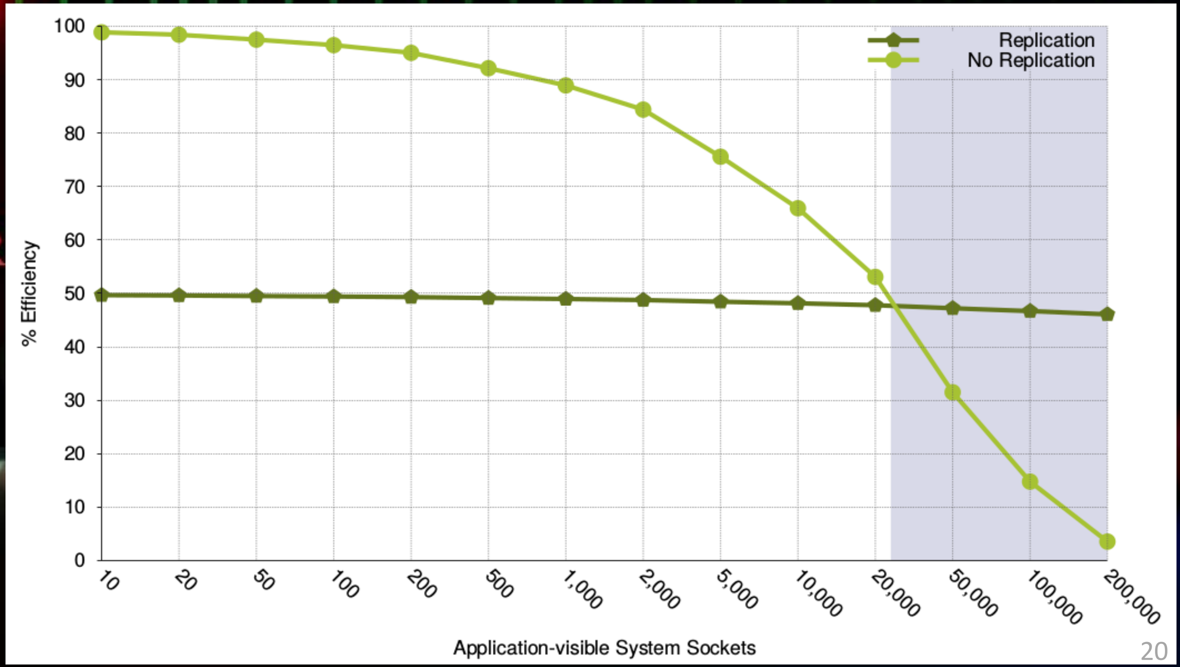
TH, D. Moor: Energy, Memory, and Runtime Tradeoffs for Implementing Collective Communication Operations, JSFI 2015

What happens if processes/nodes fail?



- **Things will fail!**
  - Wang et al., 2010: “Peta-scale systems: MTBF 1.25 hours”
  - Brightwell et al., 2011: “Next generation systems must be designed to handle failures without interrupting the workloads on the system or crippling the efficiency of the resource.”

    Checkpoint/restart will take longer than MTBF!
- **We need to enable applications to survive faults**
  - … to reach ~~Petascale~~ Exascale!
  - Like people did for decades in distributed systems!



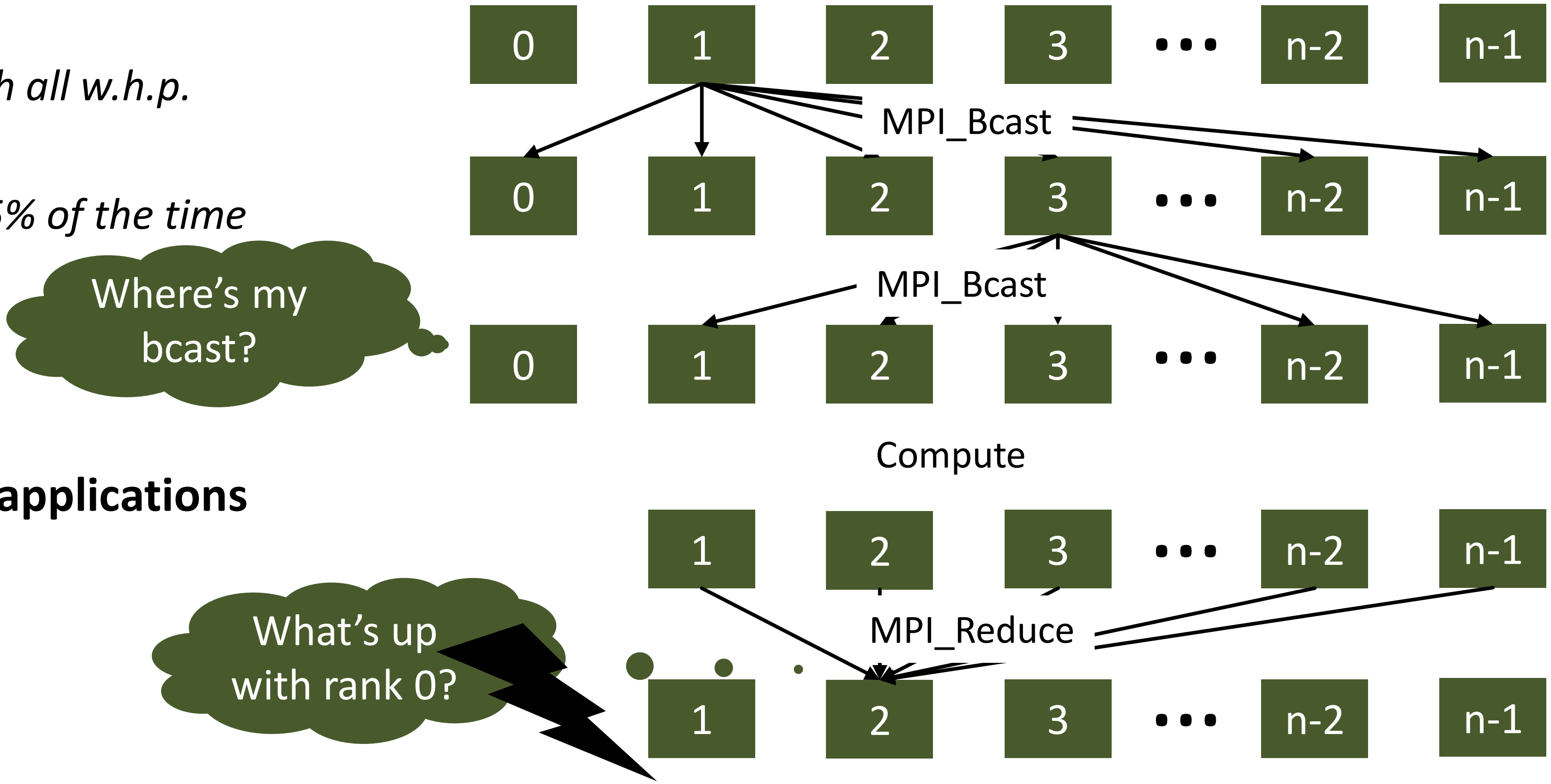
Ferreira et al.: Evaluating the Viability of Process Replication Reliability for Exascale Systems, SC11

A fast, low-work, fault-tolerant broadcast

- **Gossip?**
  - If root or message received: send to random other node until some global time expires
  - Proven to be very effective
  - Not strongly consistent ☹
  - Nice theory

    needs 1.64 $\log_2 n$ rounds to reach all w.h.p.
  - But for N=1000

    17 rounds only color all nodes 95% of the time

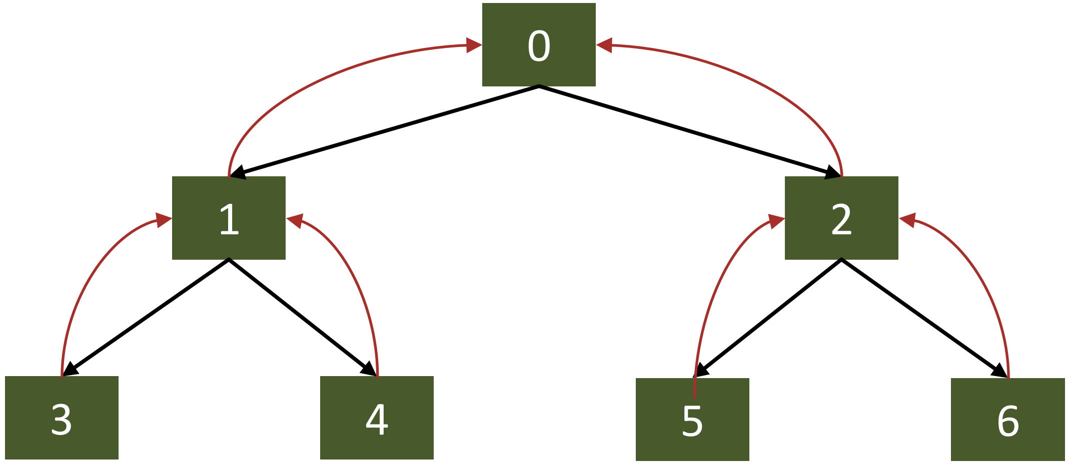


- **Very problematic for BSP-style applications**

Hoefler et al.: "Corrected Gossip Algorithms for Fast Reliable Broadcast on Unreliable Systems", IPDPS'17

But how does MPI (FT-MPICH) work then? Buntinas' FT broadcast!

- Uses a dynamic tree, each message contains information about children at next levels
- Children propagate back to root, relying on local failure-detectors

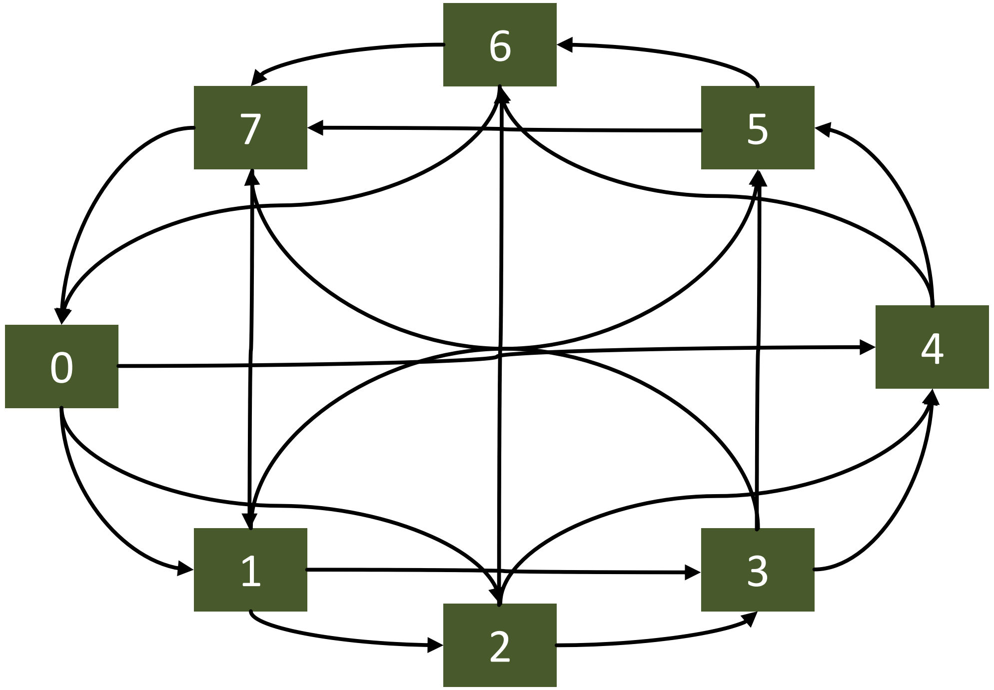


- Complex tree rebuild protocol
- Root failure results in bcast never delivered
- At least 2 $\log_2 n$ depth!

Hoefler et al.: "Corrected Gossip Algorithms for Fast Reliable Broadcast on Unreliable Systems", IPDPS'17

But how does MPI (FT-OpenMPI) work then? Binomial graph broadcast!

- **Use fixed graph, send along redundant edges**
- **Binomial graphs: each node sends to and receives from $\log_2 n$ neighbors**



- **Can survive up to $\log_2 n$ worst-case node failures**
  - In practice much more (not worst-case)

TH, Amnon Barak, A. Shiloh and Z. Drezner: "Corrected Gossip Algorithms for Fast Reliable Broadcast on Unreliable Systems", IPDPS'17

Both are far from optimal - from trees to gossip and back!

- **The power of randomness: gossip but <u>not just</u> gossip!**
- **Combine the probabilistic gossip protocol with a deterministic correction protocol**

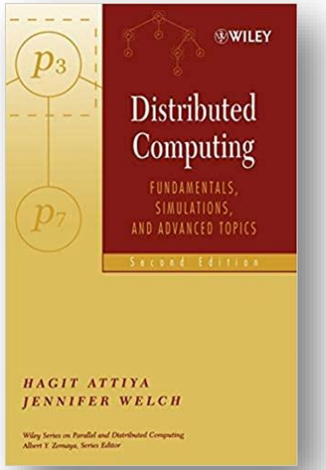
Corrected gossip turns Monte Carlo style gossiping algorithms into Las Vegas style deterministic algorithms!

- **But what is a fault-tolerant broadcast? Root failures, arbitrary failures?**
  - Assuming fail-stop, four criteria need to be fulfilled:

1. Integrity (all received messages have been sent)
2. No duplicates (each sent message is received only once)
3. Nonfaulty liveness (messages from a live node are received by all live nodes)
4. Faulty liveness (messages sent from a failed node are either received by all or none live nodes)

- **We relax 3+4 a bit: three levels of consistency**

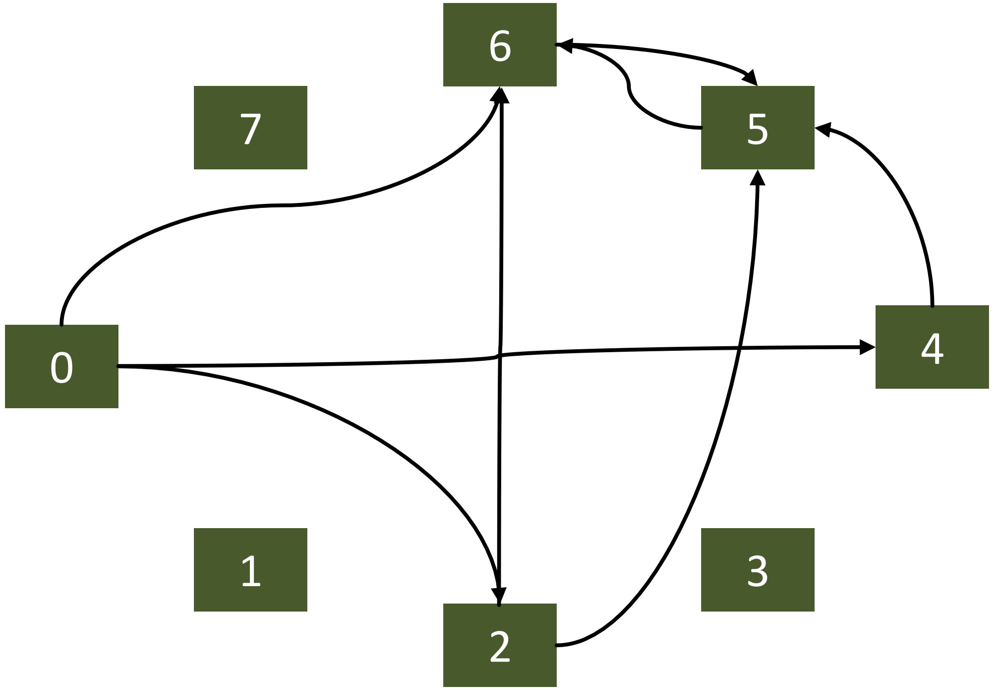
1. Not consistent (we provide an improvement over normal gossiping)
2. Nearly consistent (assuming no nodes fail during the correction phase, practical assumption)
3. Fully consistent (any failures allowed)



TH, Amnon Barak, A. Shiloh and Z. Drezner: "Corrected Gossip Algorithms for Fast Reliable Broadcast on Unreliable Systems", IPDPS'17

First algorithm: OCG (Opportunistic Corrected Gossip)

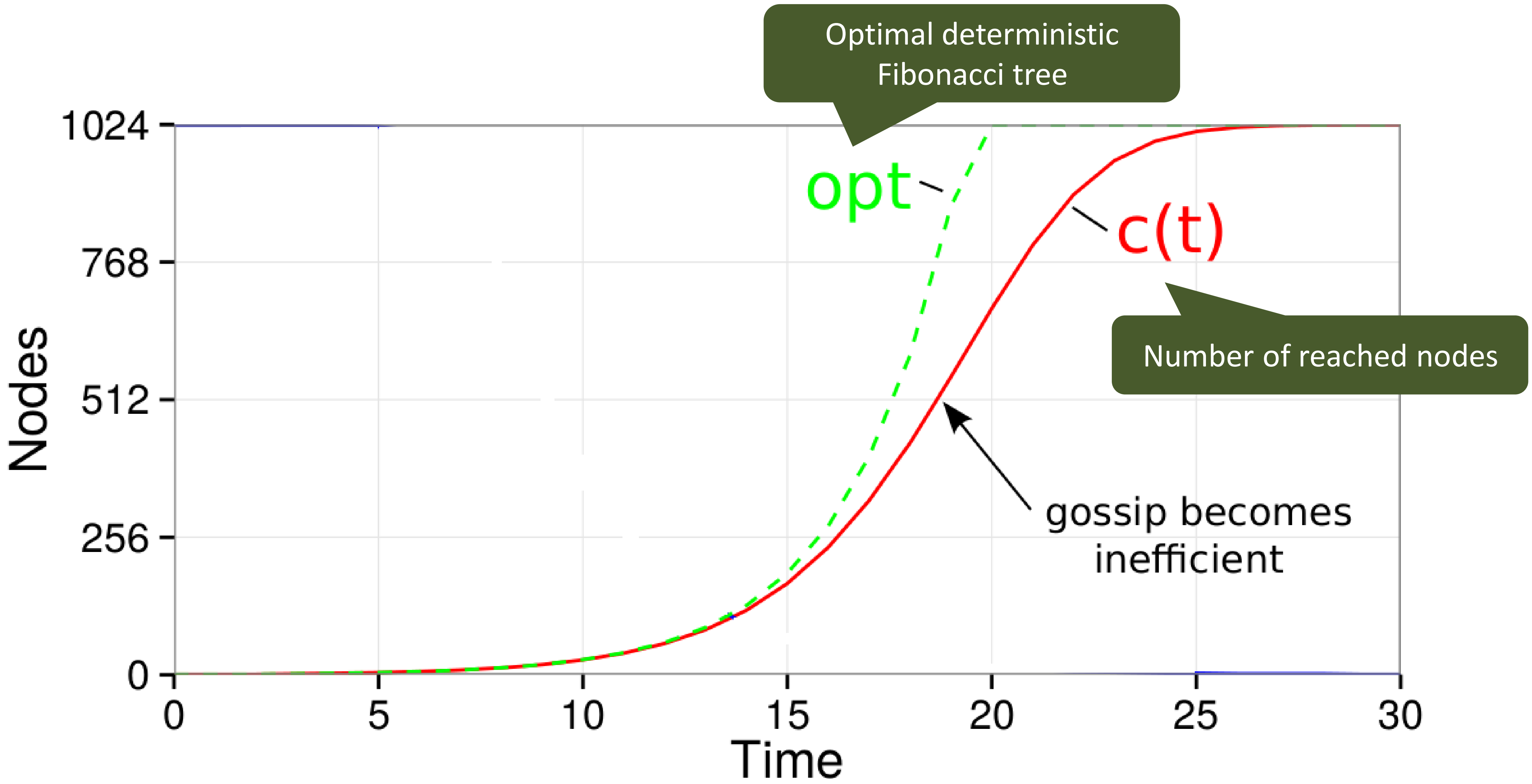
- **Not consistent, works w.h.p. --- let's first consider just gossiping**



Are all these redundant messages efficient?

TH, Amnon Barak, A. Shiloh and Z. Drezner: "Corrected Gossip Algorithms for Fast Reliable Broadcast on Unreliable Systems", IPDPS'17

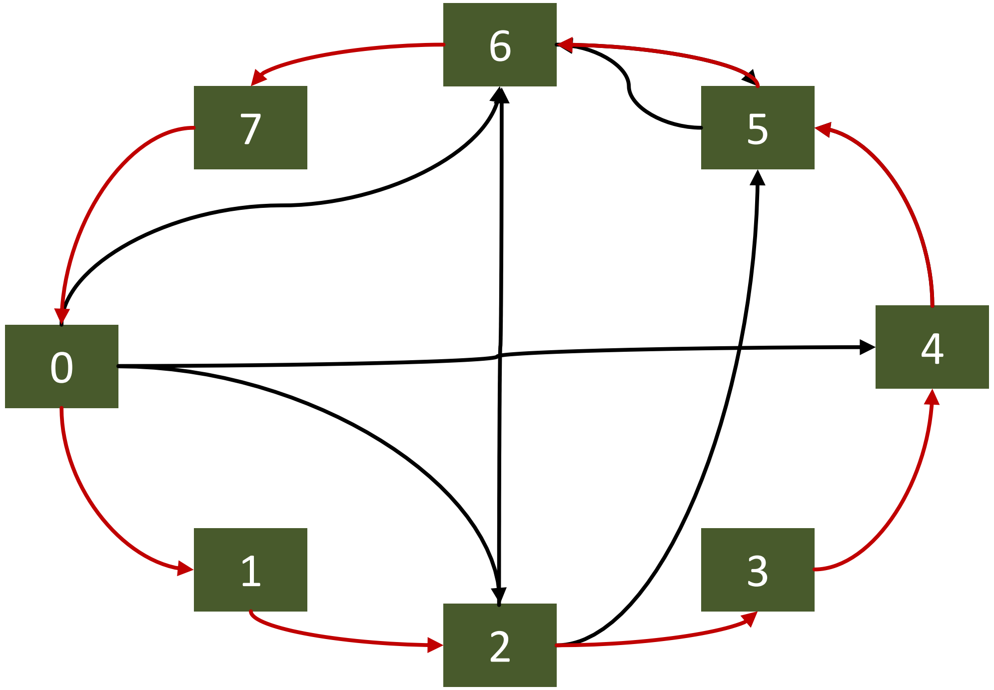
First algorithm: OCG (Opportunistic Corrected Gossip)



TH, Amnon Barak, A. Shiloh and Z. Drezner: “Corrected Gossip Algorithms for Fast Reliable Broadcast on Unreliable Systems”, IPDPS’17

First algorithm: OCG (Opportunistic Corrected Gossip)

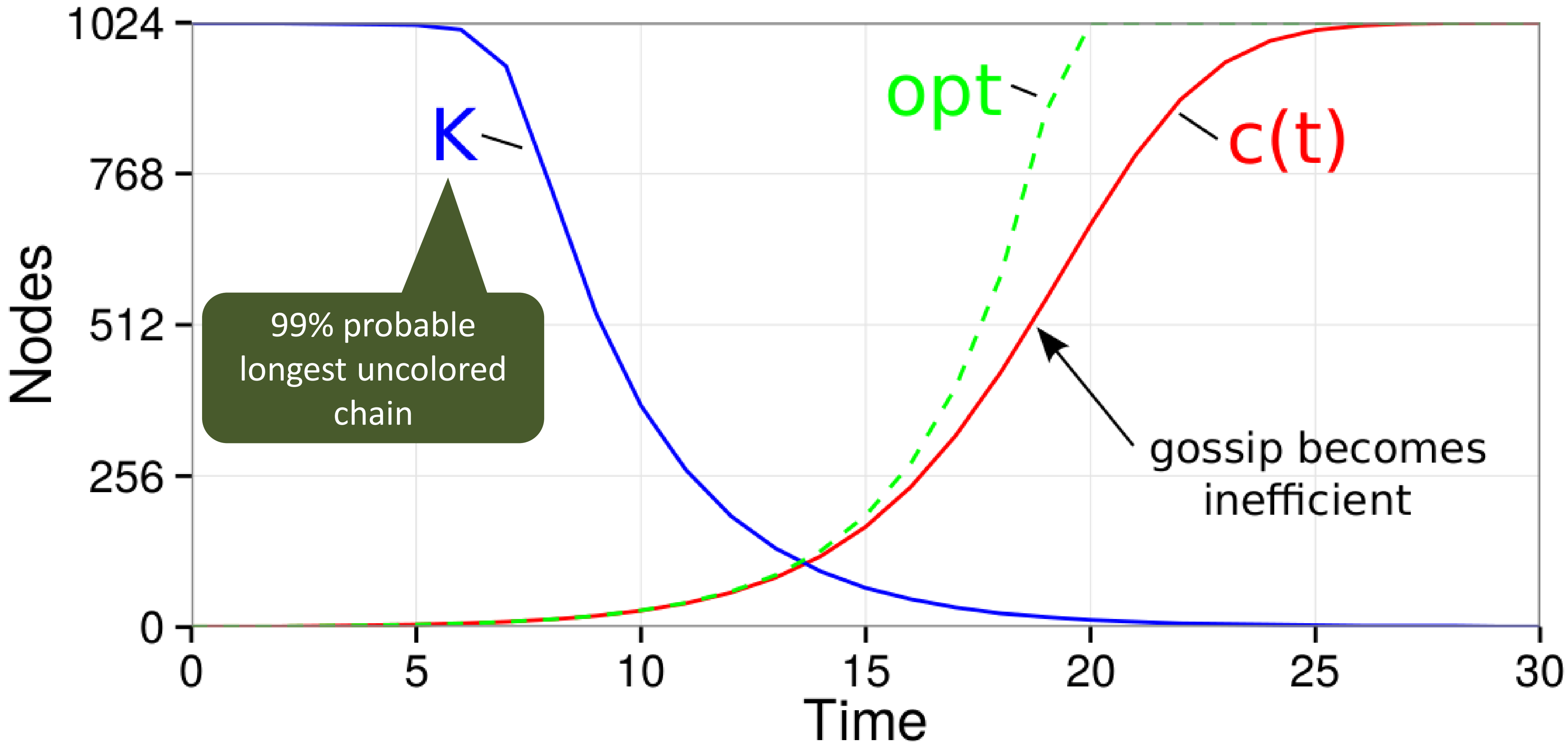
- **OCG main idea: run gossip for a while and then switch to a deterministic ring-correction protocol**
  - Every node that received a message sends it to (rank + 1) % nranks



- **Each message may be received twice**
  - But this depends on when we switch! But what is the longest uncolored chain?

TH, Amnon Barak, A. Shiloh and Z. Drezner: "Corrected Gossip Algorithms for Fast Reliable Broadcast on Unreliable Systems", IPDPS'17

The longest uncolored chain K!



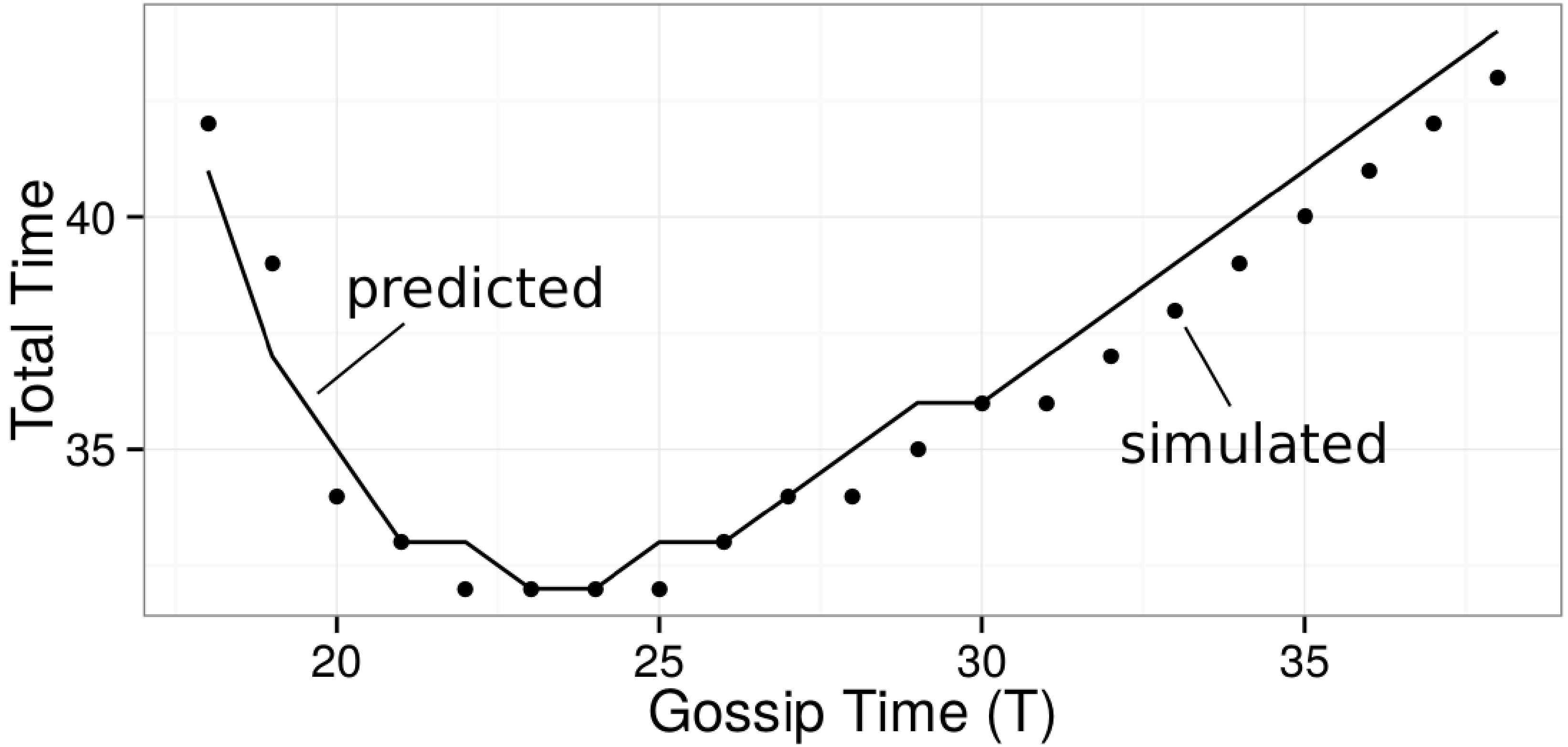
TH, Amnon Barak, A. Shiloh and Z. Drezner: “Corrected Gossip Algorithms for Fast Reliable Broadcast on Unreliable Systems”, IPDPS’17

First algorithm: OCG (Opportunistic Corrected Gossip)

- **When to switch from gossip to correction?**
  - Well, when the expected number of correction steps is small and gossip is inefficient
- **We can bound the probability of a longest chain of length k**
  - In terms of the LogP parameters, T (gossip time), and N (nranks)

$$T_{opt}^{OCG} = \underset{T}{\operatorname{argmin}}(T + 2L + (2 + \overline{K})O)$$

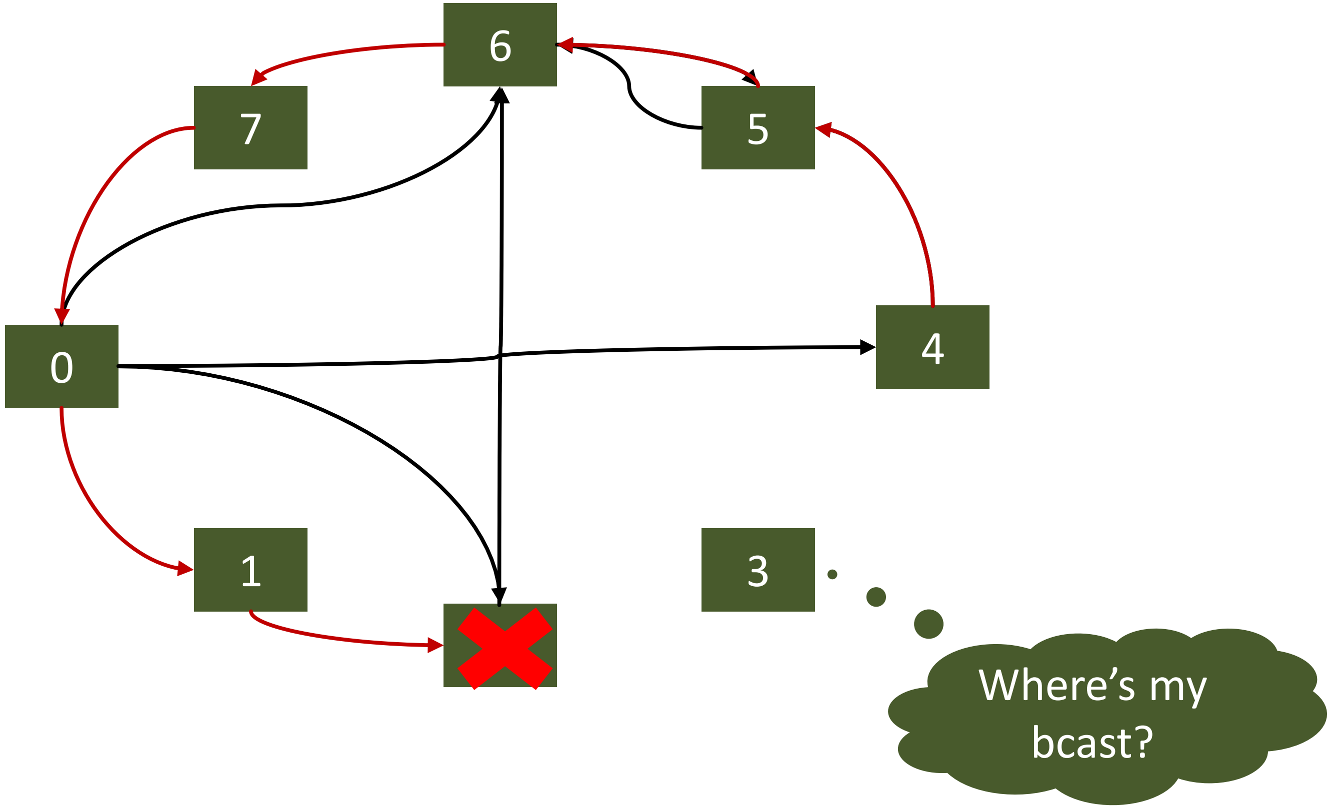
The optimal time to switch depends on L, O, and N



TH, Amnon Barak, A. Shiloh and Z. Drezner: "Corrected Gossip Algorithms for Fast Reliable Broadcast on Unreliable Systems", IPDPS'17

OCG Consistency

- **OCG is more efficient than gossip but does not guarantee that all nodes are reached (even w/o failures)**

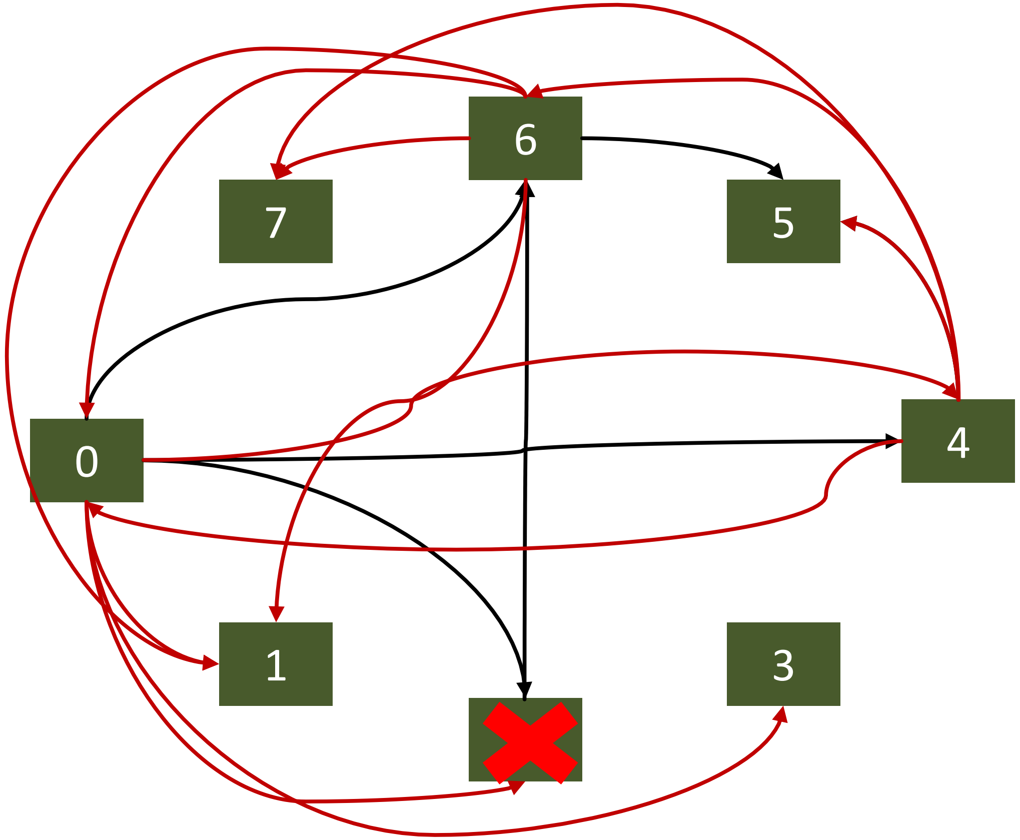


- **So we need to check that they were actually reached!**

TH, Amnon Barak, A. Shiloh and Z. Drezner: "Corrected Gossip Algorithms for Fast Reliable Broadcast on Unreliable Systems", IPDPS'17

Second algorithm: CCG (Checked Corrected Gossip)

- **CCG sends to the next node until it sent to a node it received from (i.e., knows that node was alive!)**
  - Since the node it received from also sent, it "knows" that all other nodes have been covered!

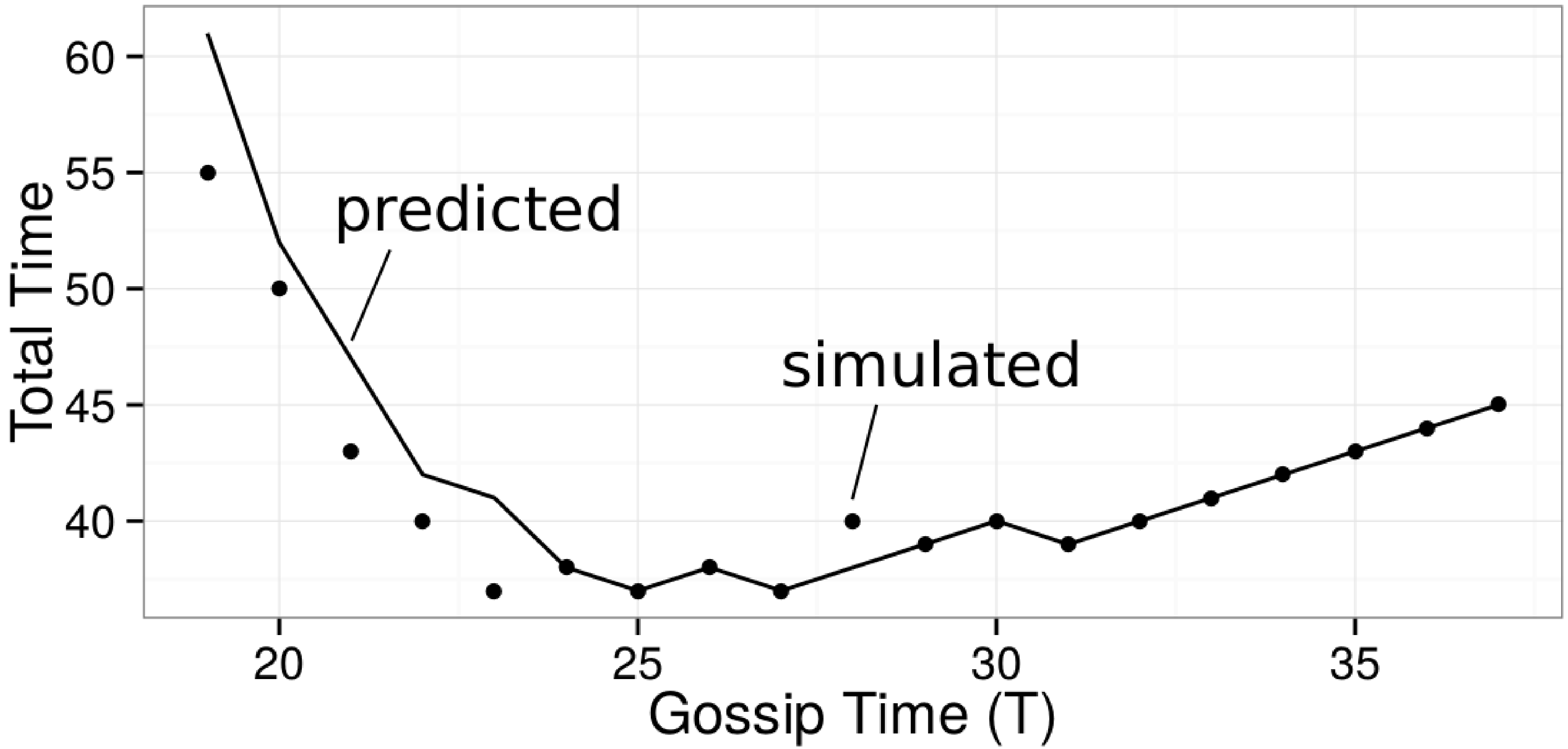


- **CCG guarantees that all nodes are reached unless a node dies in the middle of the correction phase!**
  - And another node assumes it finished its job!

TH, Amnon Barak, A. Shiloh and Z. Drezner: "Corrected Gossip Algorithms for Fast Reliable Broadcast on Unreliable Systems", IPDPS'17

Second algorithm: CCG (Checked Corrected Gossip)

- **When to switch from gossip to correction?**



- **A bit later than OCG**

TH, Amnon Barak, A. Shiloh and Z. Drezner: “Corrected Gossip Algorithms for Fast Reliable Broadcast on Unreliable Systems”, IPDPS’17

Third algorithm: FCG (Failure-proof Corrected Gossip)

- **FCG can protect from f failures – similar to CCG but instead of aborting to send when heard from one, it waits to hear from f+1 other nodes!**
- **So any f nodes can fail and it will still succeed (keep sending)**

- **Wait, what if there are less than f+1 nodes reached during gossip and they somehow die in the middle of the protocol?**
  - So we need to involve the non-gossip-colored nodes
  - They will wait to hear from a gossip-colored nodes to exit
  - If no such exit signal comes within a timeout period, panic!
  - In panic mode, send to every other node
  - Every node that receives panic messages also panics
  - This guarantees consistency (at a high cost)

- **Panic mode is extremely unlikely in practice (much less likely than the failing of binomial graphs)**
  - Likelihood can be reduced arbitrarily with gossiping time!
  - So panic is just a theoretical concern (to proof correctness)

TH, Amnon Barak, A. Shiloh and Z. Drezner: "Corrected Gossip Algorithms for Fast Reliable Broadcast on Unreliable Systems", IPDPS'17

Case study: TSUBAME 2.0

- **TiTech machine, published failure logs**
- **Node MTBF = 18,304 hours**
  - Assume 12 hour run on 4,096 nodes = 2.69 failures
- **We compare all algorithms and report**
  1. Expected latency
  2. Expected work
  3. Expected inconsistency

  For CCG/OCG/FCG, we simulate until the nonparameric CI was within 2% of the median

Gossip

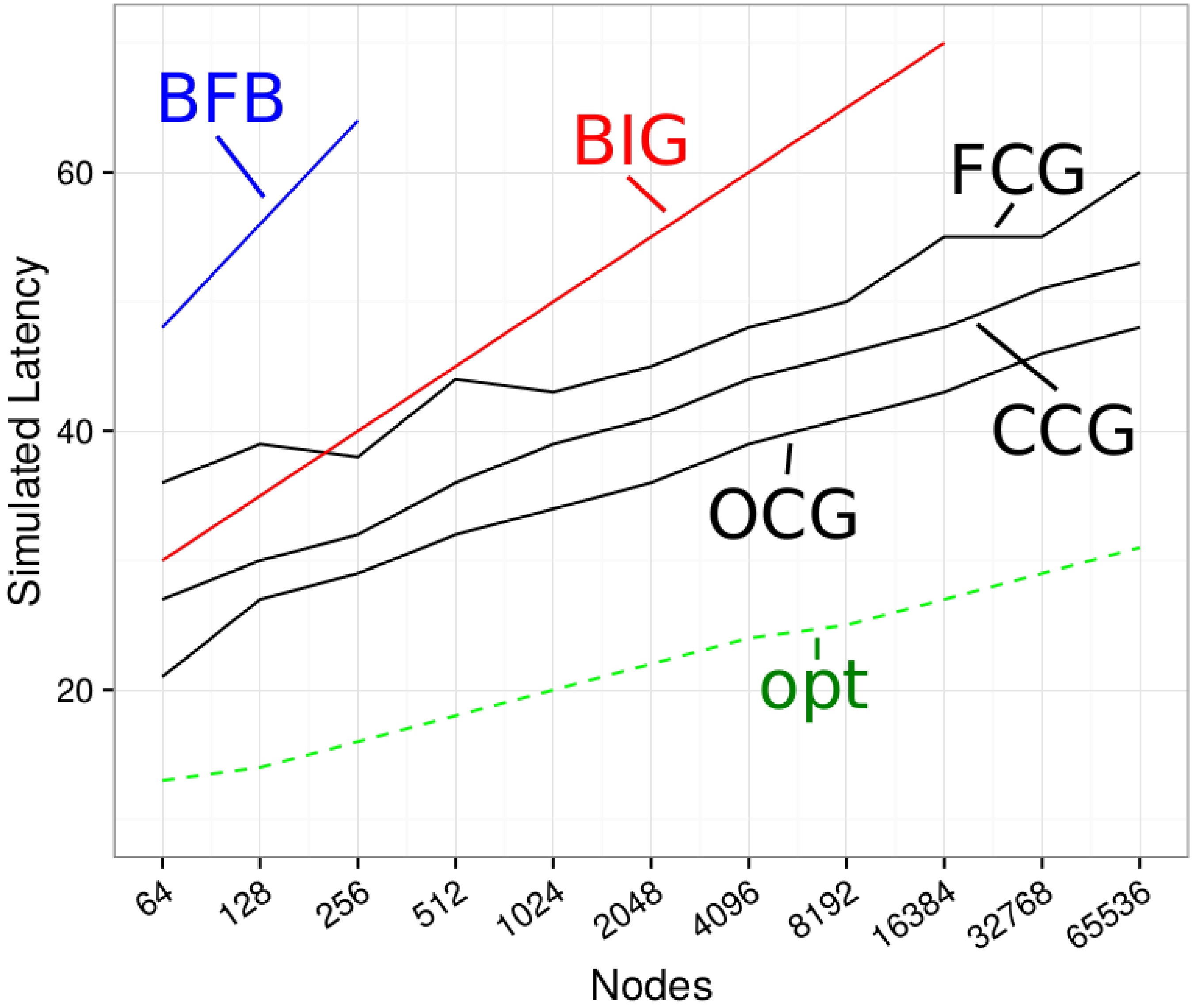
Binomial Graphs

Buntinas' Tree

algorithm	$\hat{f}$	T	lat	work	incon.
GOS [12]	0				
GOS [12]	3				
OCG	0				
OCG	3				
CCG	0				
CCG	3				
FCG	0				
FCG	3				
BIG [2]	0				
BIG [2]	3				
BFB [8]	0				
BFB [8]	3				

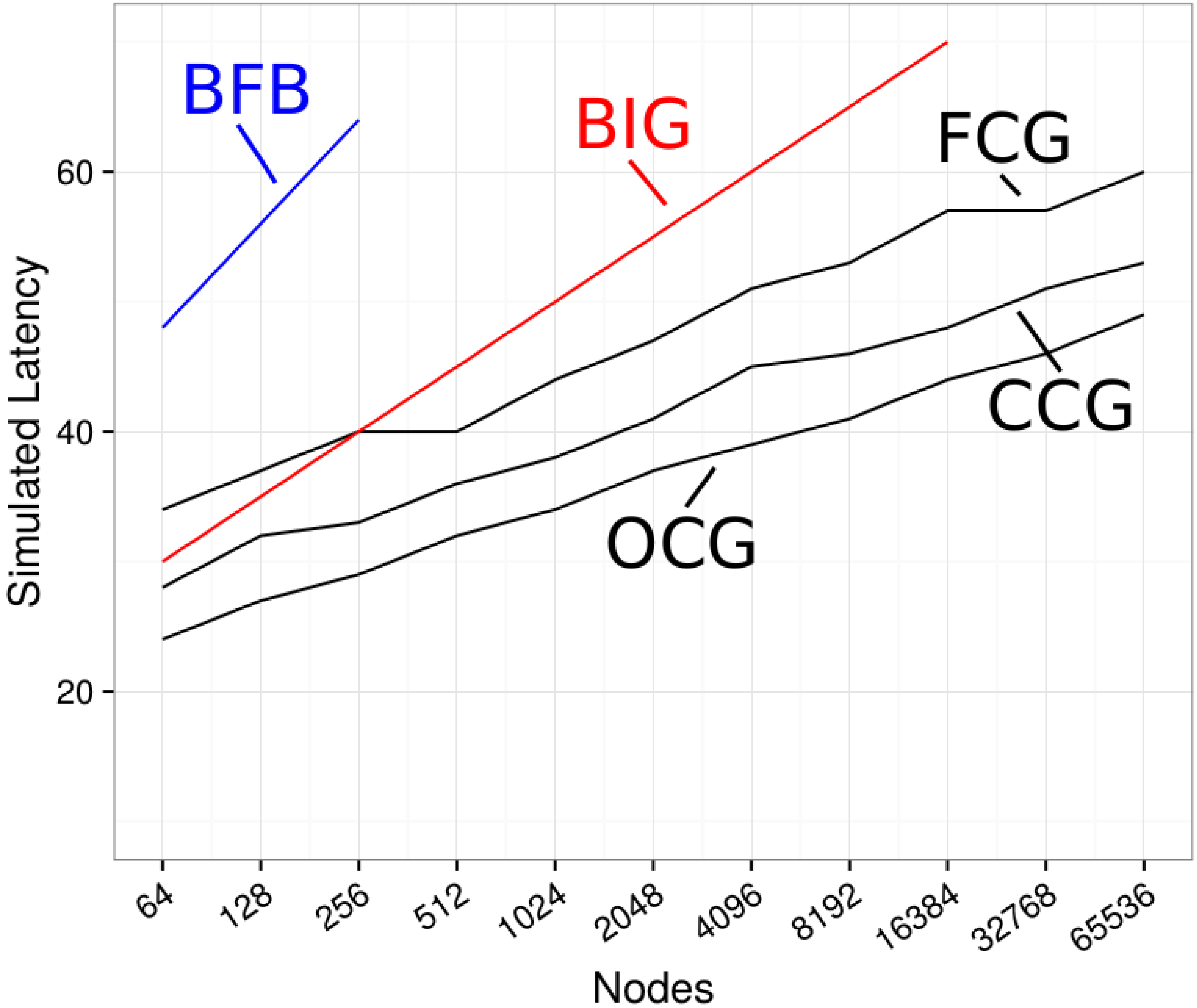
TH, Amnon Barak, A. Shiloh and Z. Drezner: "Corrected Gossip Algorithms for Fast Reliable Broadcast on Unreliable Systems", IPDPS'17

Scaling – Without failures



TH, Amnon Barak, A. Shiloh and Z. Drezner: “Corrected Gossip Algorithms for Fast Reliable Broadcast on Unreliable Systems”, IPDPS’17

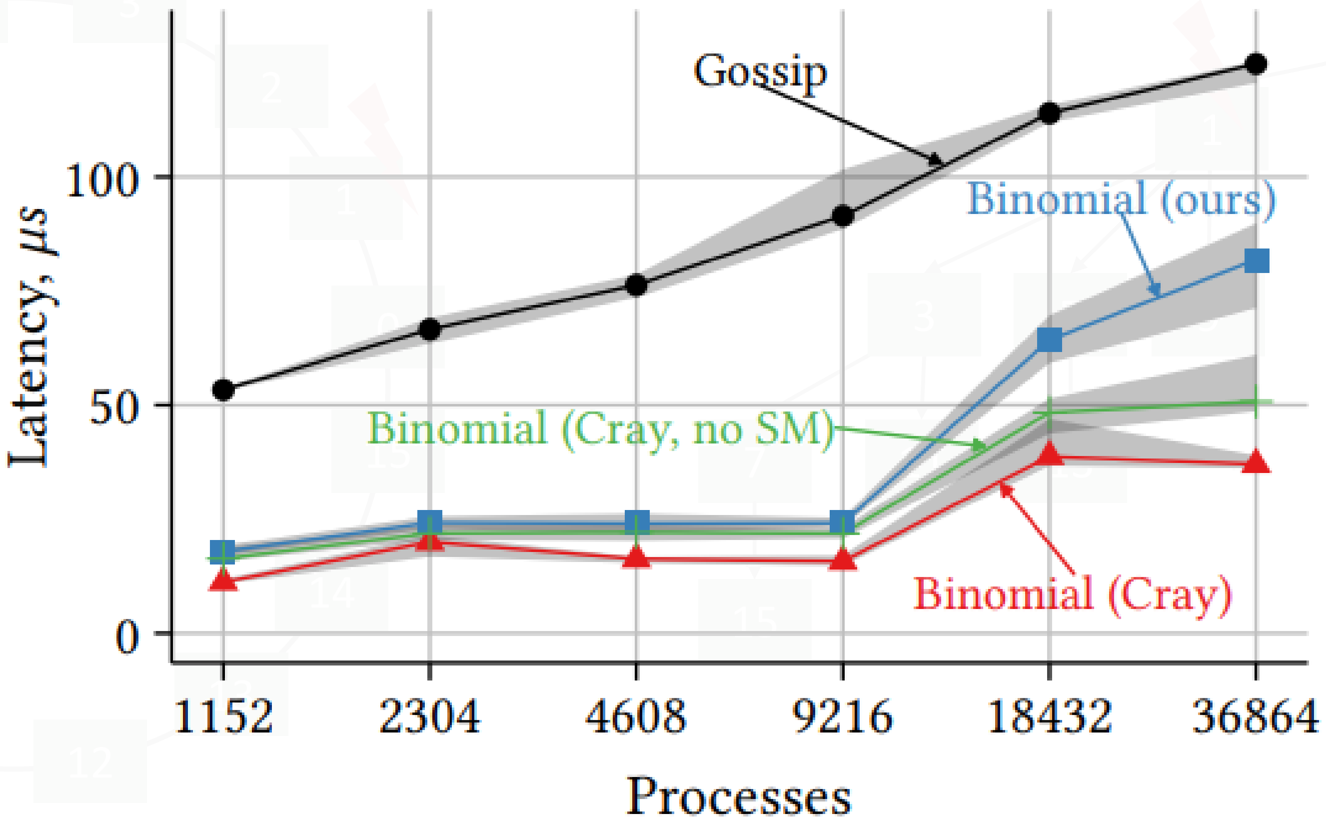
Scaling – With failures (expected for 12 hours on TSUBAME 2.0)



TH, Amnon Barak, A. Shiloh and Z. Drezner: "Corrected Gossip Algorithms for Fast Reliable Broadcast on Unreliable Systems", IPDPS'17

How to get to optimal? Corrected (optimal) trees!

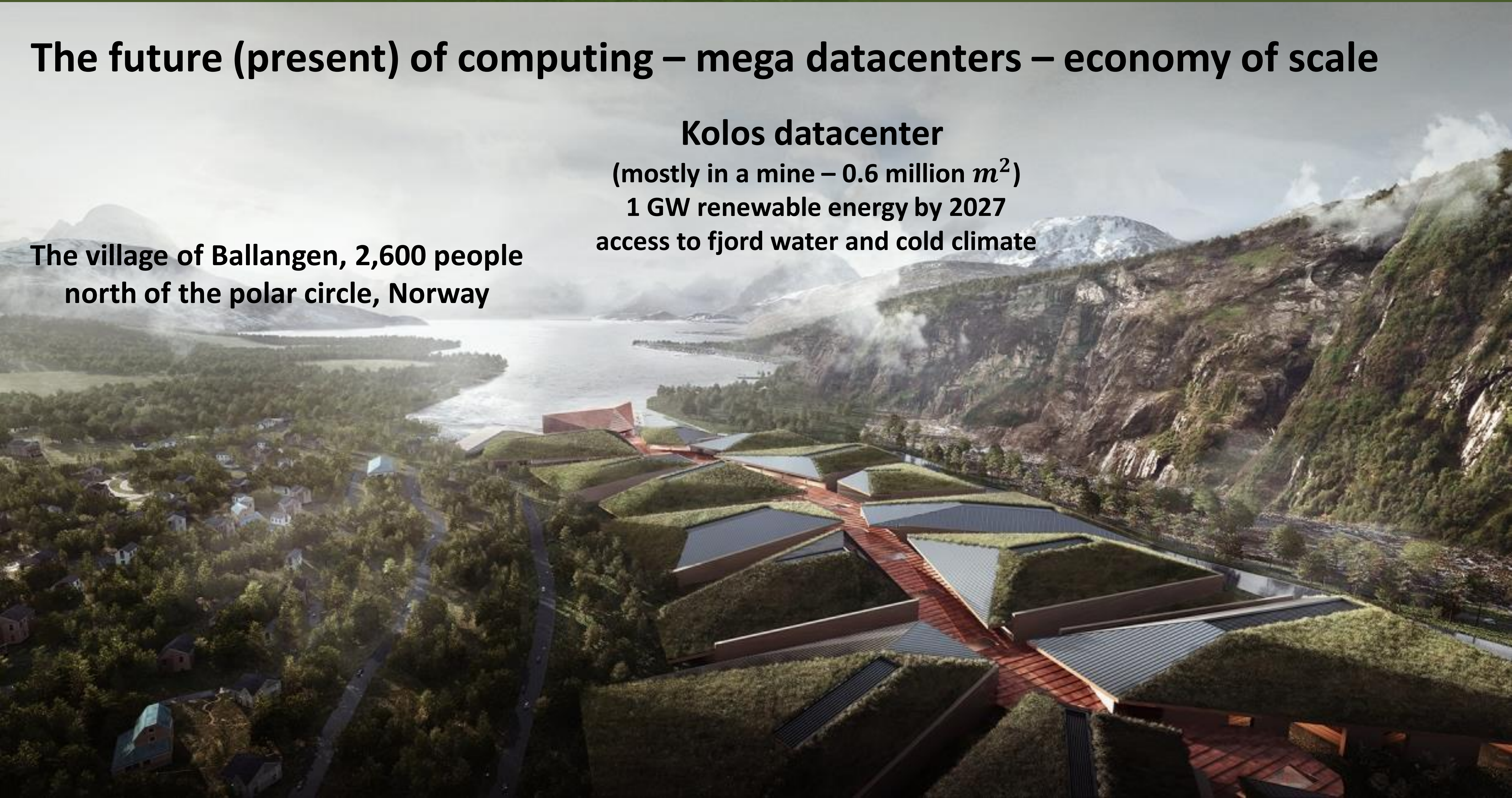
Piz Daint at CSCS



A single ring correction step reaches all nodes now! Generalizes to k steps with k failures. Tree numbering is key!

Martin Kuettler, Maksym Planeta, Jan Bierbaum, Carsten Weinhold, Hermann Haertig, Amnon Barak, TH: Corrected Trees for Reliable Group Communication, PPoPP'19

The future (present) of computing – mega datacenters – economy of scale



Kolos datacenter

(mostly in a mine – 0.6 million m^2)
1 GW renewable energy by 2027
access to fjord water and cold climate

The village of Ballangen, 2,600 people
north of the polar circle, Norway

“The network is the Computer” John Gage, Sun Microsystems, 1984

“Datacenters are not supercomputers yet, but eventually they will be.” (me, now)

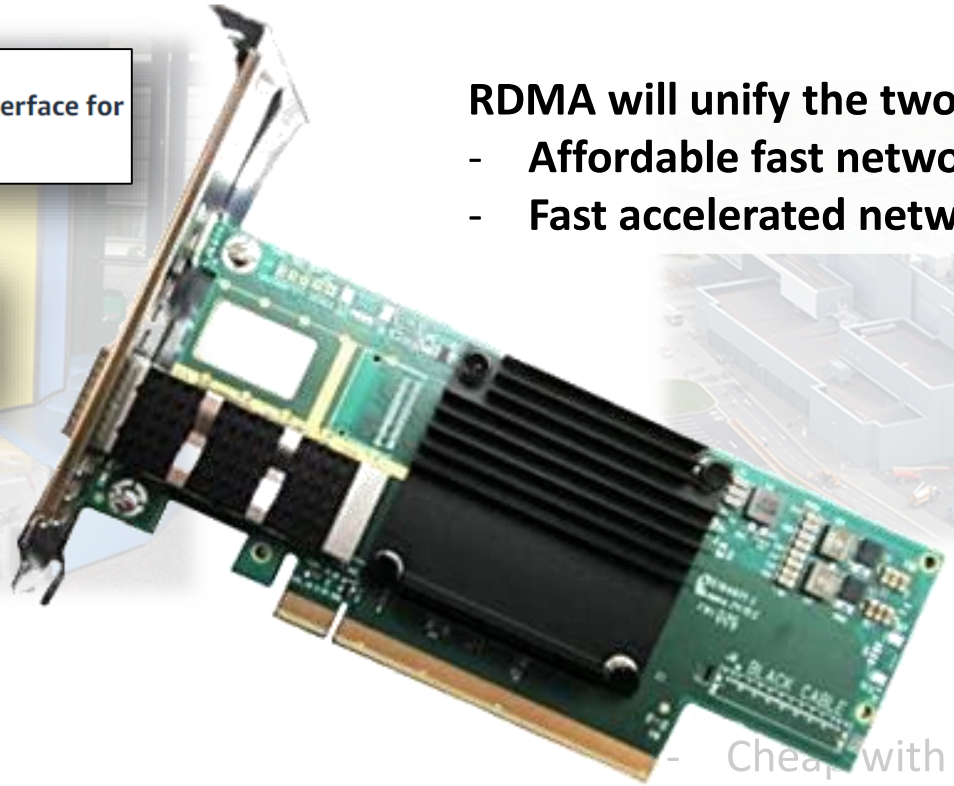
AWS News Blog

Elastic Network Adapter – High Performance Network Interface for Amazon EC2

by Jeff Barr | on 28 JUN 2016 | in Amazon EC2 | Permalink | Share

Microsoft to Drive RDMA Into Datacenters and Clouds

November 18, 2013 by Timothy Prickett Morgan



RDMA will unify the two:

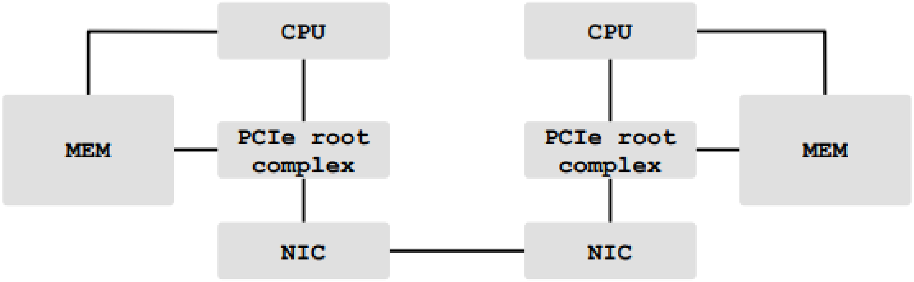
- **Affordable fast networking and distributed memory**
- **Fast accelerated networking (GPU, network acceleration)**

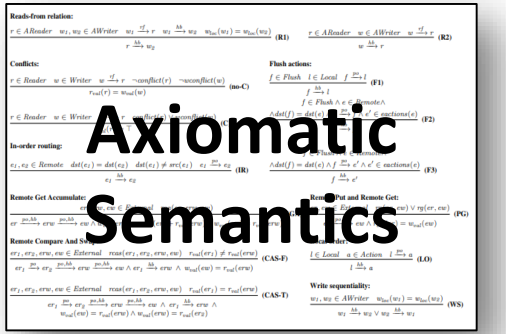
- Fast and cheap
- Performance first, productivity second

- Cheap with reasonable performance
- Productivity first, performance second

- **New research opportunities – RDMA networking offering RMA programming**
- **(actually, we are moving post-RDMA with Smart NICs/sPIN – but no time to discuss that now)**
- (cf. Next Platform: “Vertical integration is eating the datacenter, part two”, Feb. 2020)

Basics on R(D)MA memory models



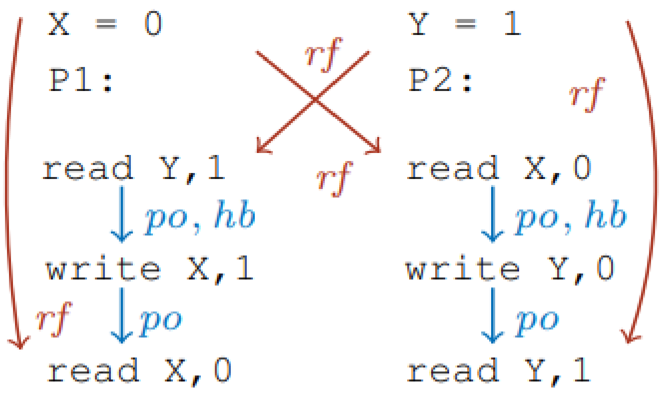


RMA	put	get	flush
DMAPP	dmapp_put_nbi	dmapp_get_nbi	dmapp_gsync_wait
OFED (IB)	ibv_wr_rdma_write	ibv_wr_rdma_read	ibv_reg_notify_cq
Portals 4	PtlPut	PtlGet	PtlCTWait
UPC	upc_memput	upc_memget	upc_fence
Fortran 2008	assignment	assignment	sync_memory
MPI-3 RMA	MPI_Put	MPI_Get	MPI_Win_flush

X = 0	Y = 1
P1:	P2:
X = get(Y^{P2})	Y = get(X^{P1})
a = X	b = Y

a = 0, b = 1

Non-sequentially consistent behavior!



Modeling and Analysis of Remote Memory Access Programming

Andrei Marian Dan
andrei.dan@inf.ethz.ch
ETH Zurich, Switzerland

Patrick Lam
patrick.lam@uwaterloo.ca
University of Waterloo, Canada

Torsten Hoefler
torsten.hoefler@inf.ethz.ch
ETH Zurich, Switzerland

Martin Vechev
martin.vechev@inf.ethz.ch
ETH Zurich, Switzerland

Abstract

Recent advances in networking hardware have led to a new generation of Remote Memory Access (RMA) networks in which processors from different machines can communicate directly, bypassing the operating system and allowing higher performance. Researchers and practitioners have proposed libraries and programming models for RMA to enable the development of applications running on these networks.

However, the memory models implied by these RMA libraries and languages are often loosely specified, poorly understood, and differ depending on the underlying network architecture and other factors. Hence, it is difficult to precisely reason about the semantics of RMA programs or how changes in the network architecture affect them.

Our work provides an important step towards understanding existing RMA networks, thus influencing the design of future RMA interfaces and hardware.

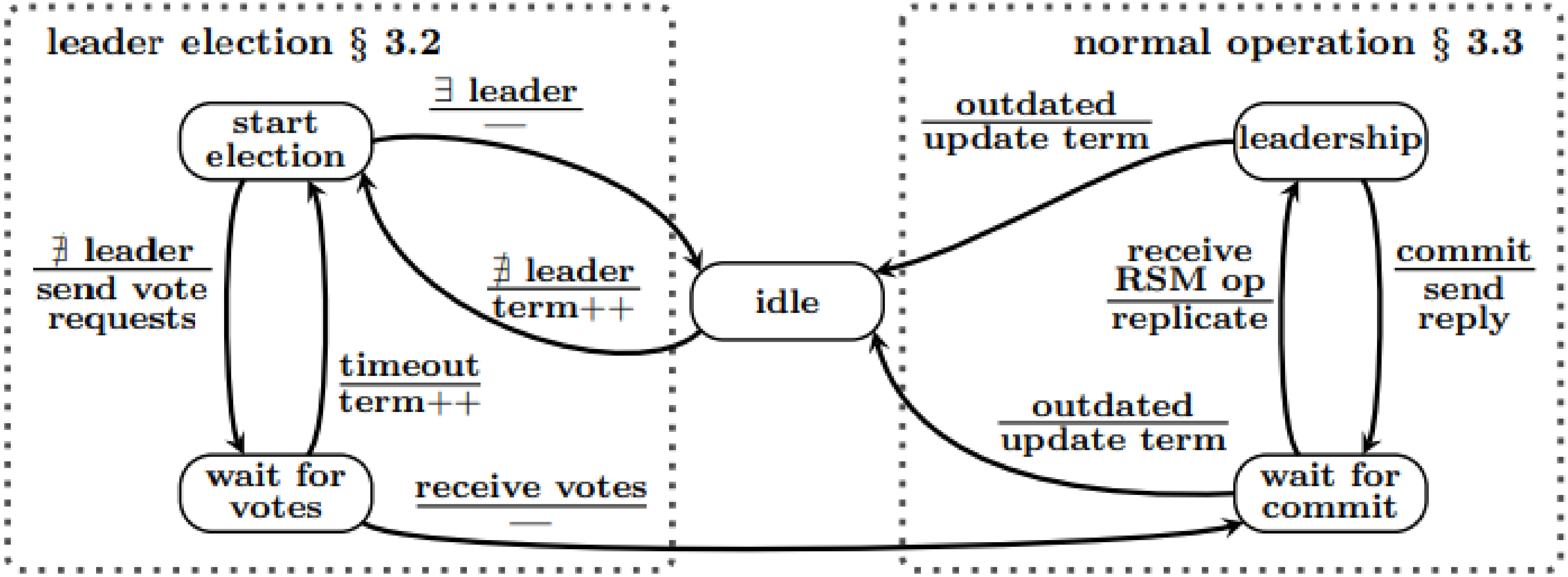
1. Introduction

Large-scale parallel systems are gaining importance for data center, big data, and scientific computations. The traditional programming models for such systems are message passing (e.g., through the Message Passing Interface—MPI) and TCP/IP sockets (as used by Hadoop, MapReduce, or Spark).

These models were designed for message-based interconnection networks such as Ethernet. Remote Direct Memory Access (RDMA) network interfaces, which have been used in High-Performance Computing for years, offer higher per-

Dan et al.: “Modeling and Analysis of Remote Memory Access Programming”, OOPSLA’16 outstanding paper

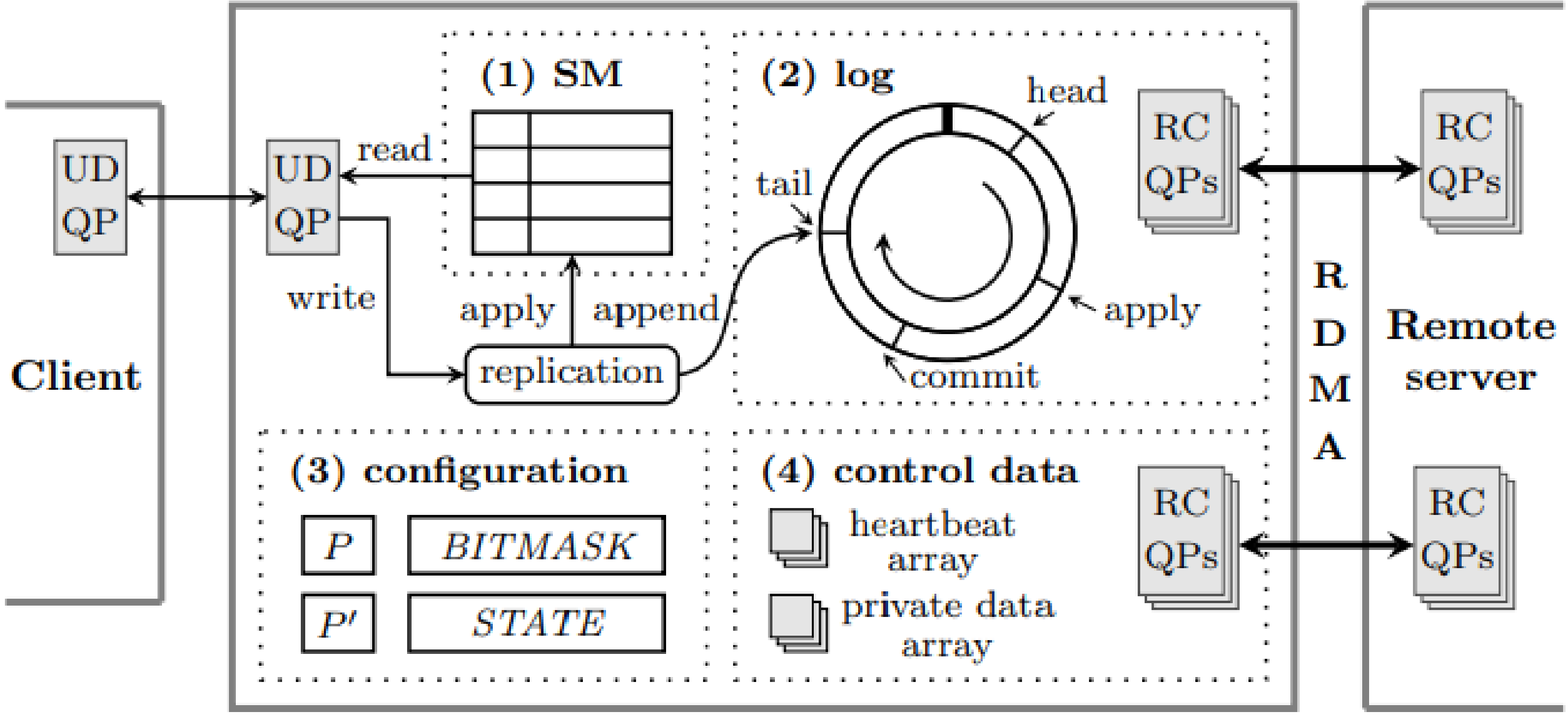
Direct Access REplication (DARE) – and RDMA consensus protocol



Leader-based replicated state machine – standard leader election (using RDMA as transport)

Poke, Hoefler: " DARE: High-Performance State Machine Replication on RDMA Networks", HPDC'15

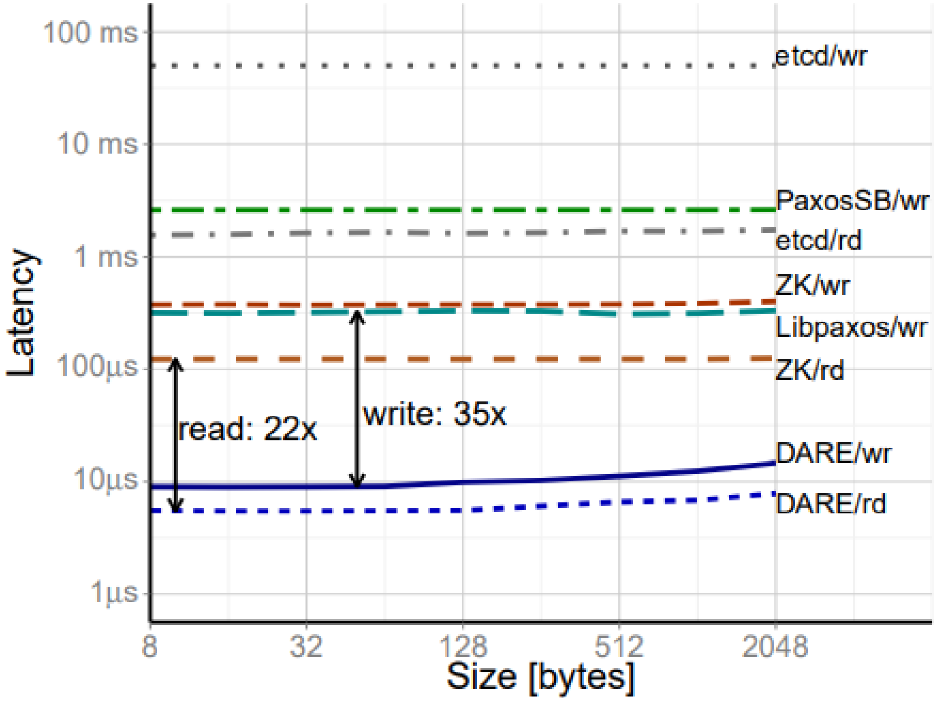
Direct Access REplication (DARE) – RDMA consensus protocol

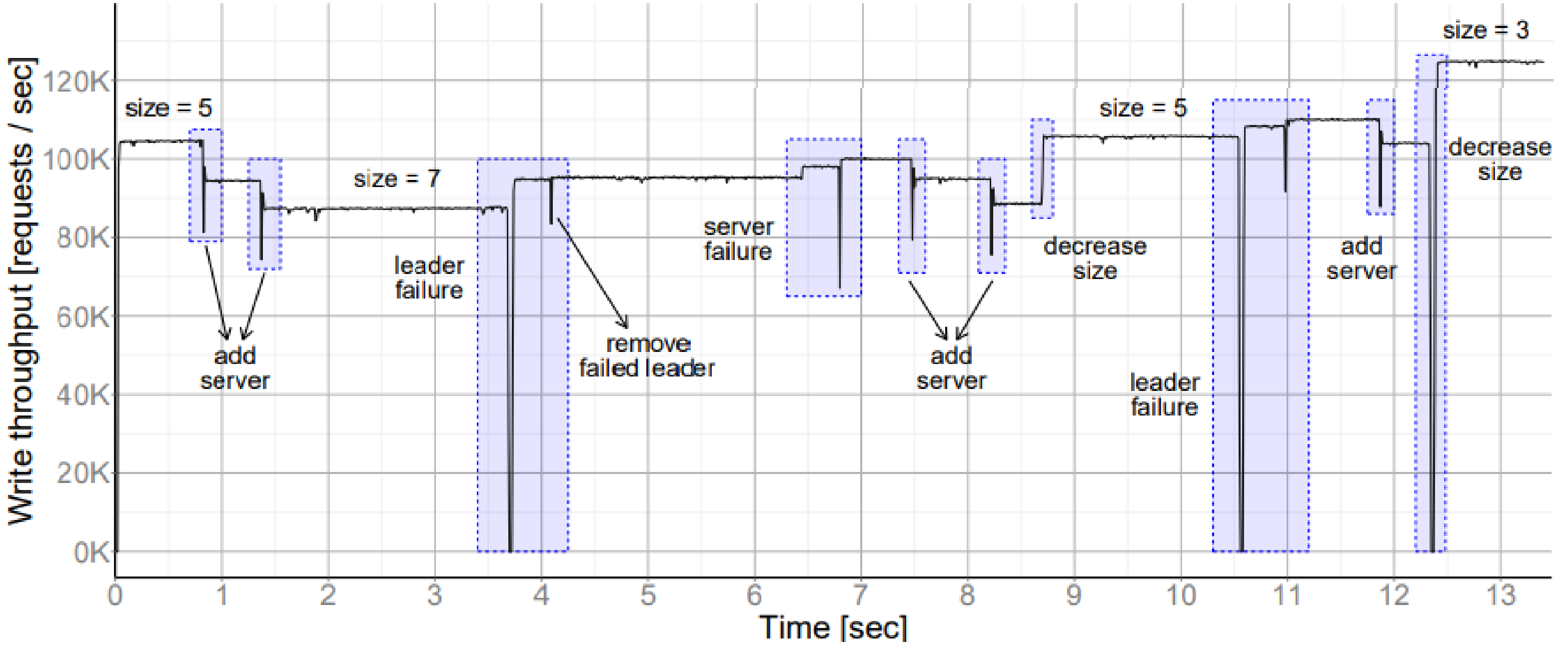


Log access via RDMA to remote servers, control and reconfiguration via direct RDMA accesses!

Poke, Hoefler: " DARE: High-Performance State Machine Replication on RDMA Networks", HPDC'15

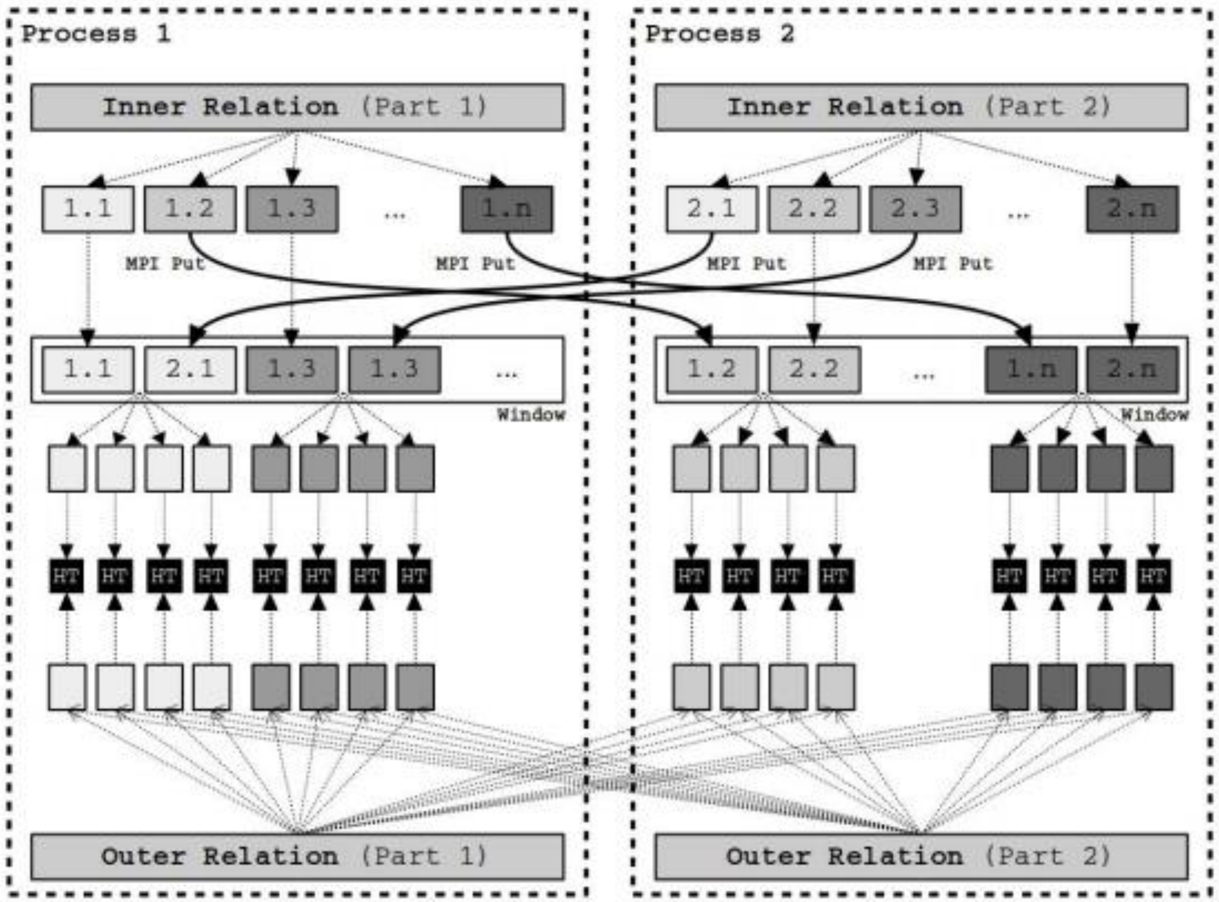
Direct Access REplication (DARE) – performance



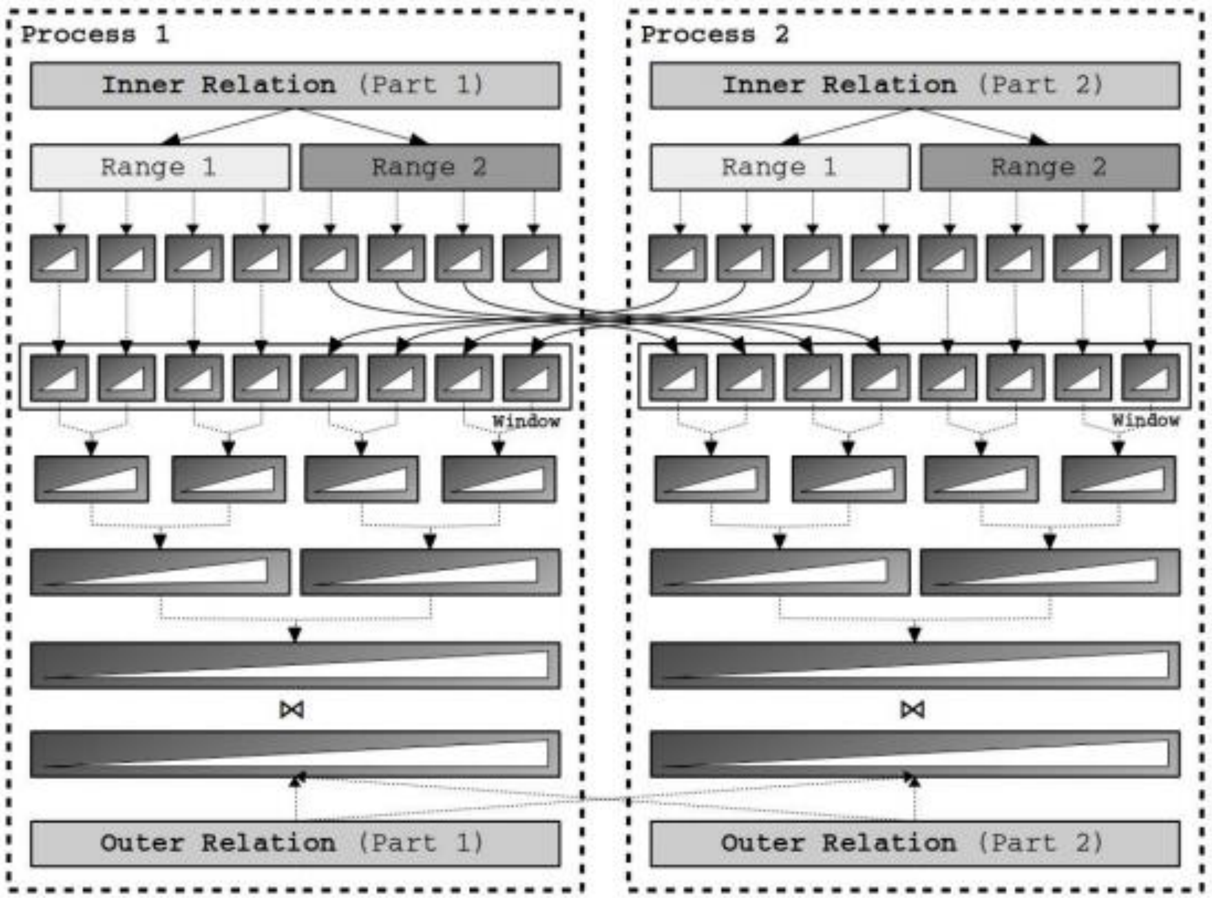


Poke, Hoefler: " DARE: High-Performance State Machine Replication on RDMA Networks", HPDC'15

RDMA join for distributed databases - algorithms

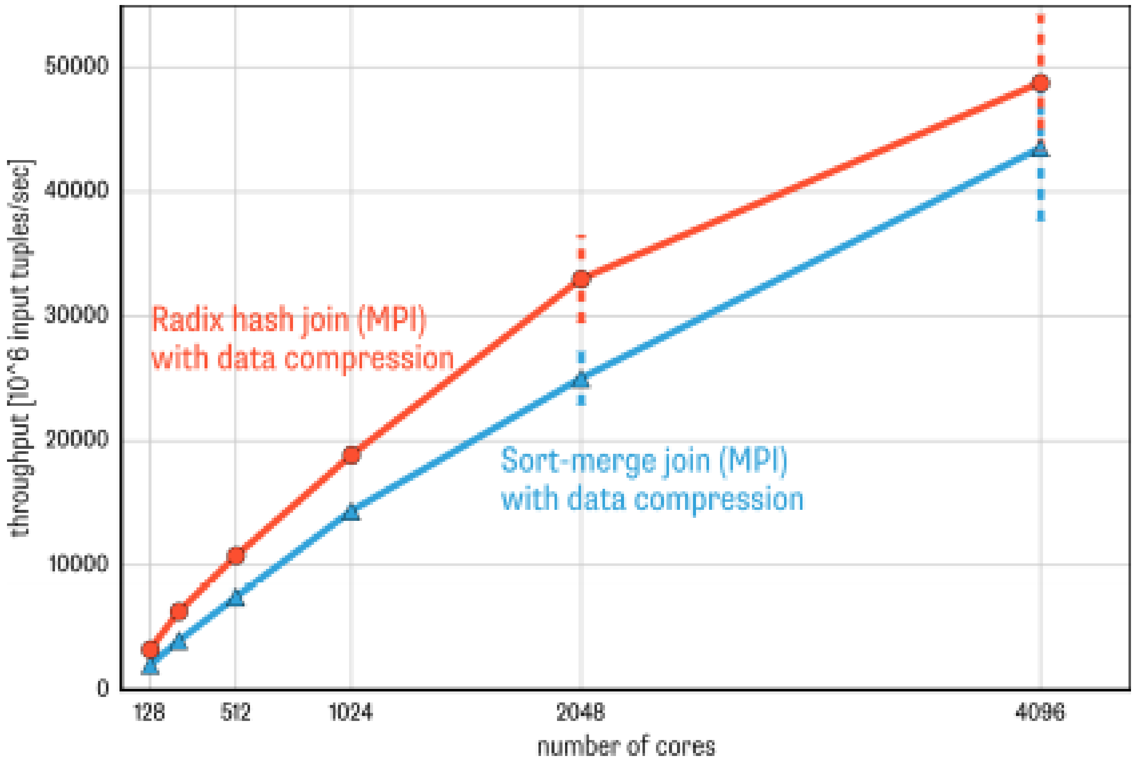


Distributed Direct-Access Radix Join

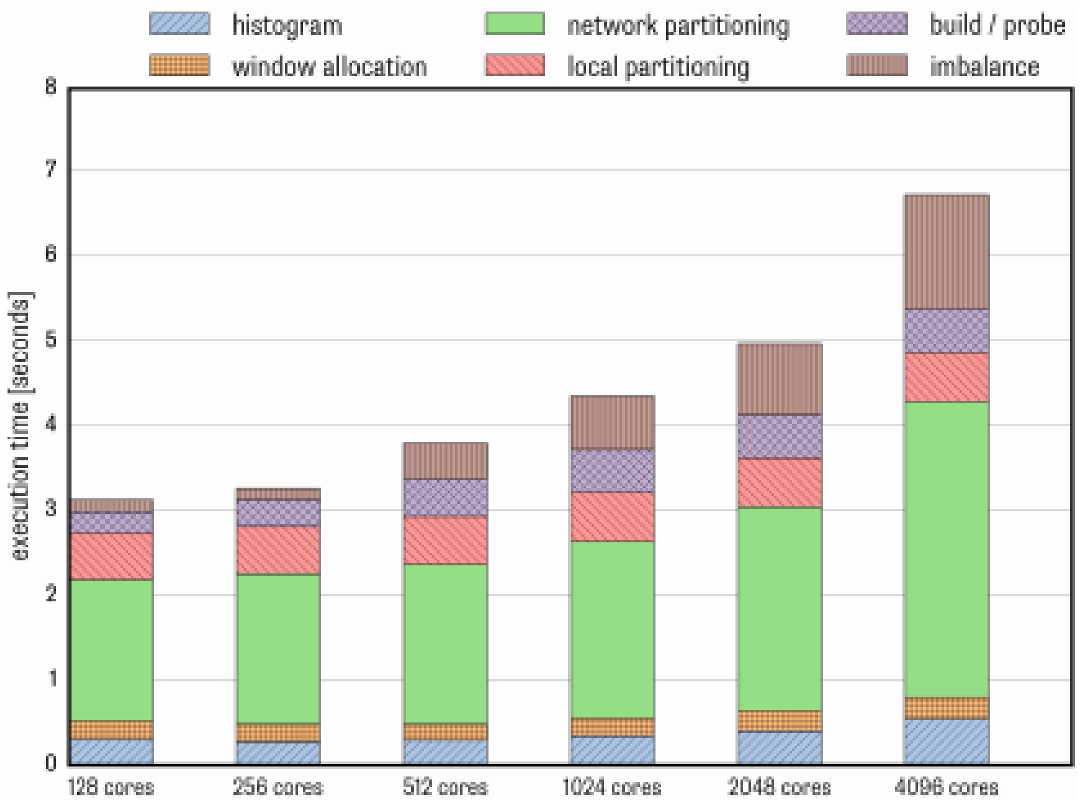


Distributed Direct-Access Sort-Merge Join

RDMA join for distributed databases - performance



Scaling joins to thousands of cores with billions of tuples/s throughput

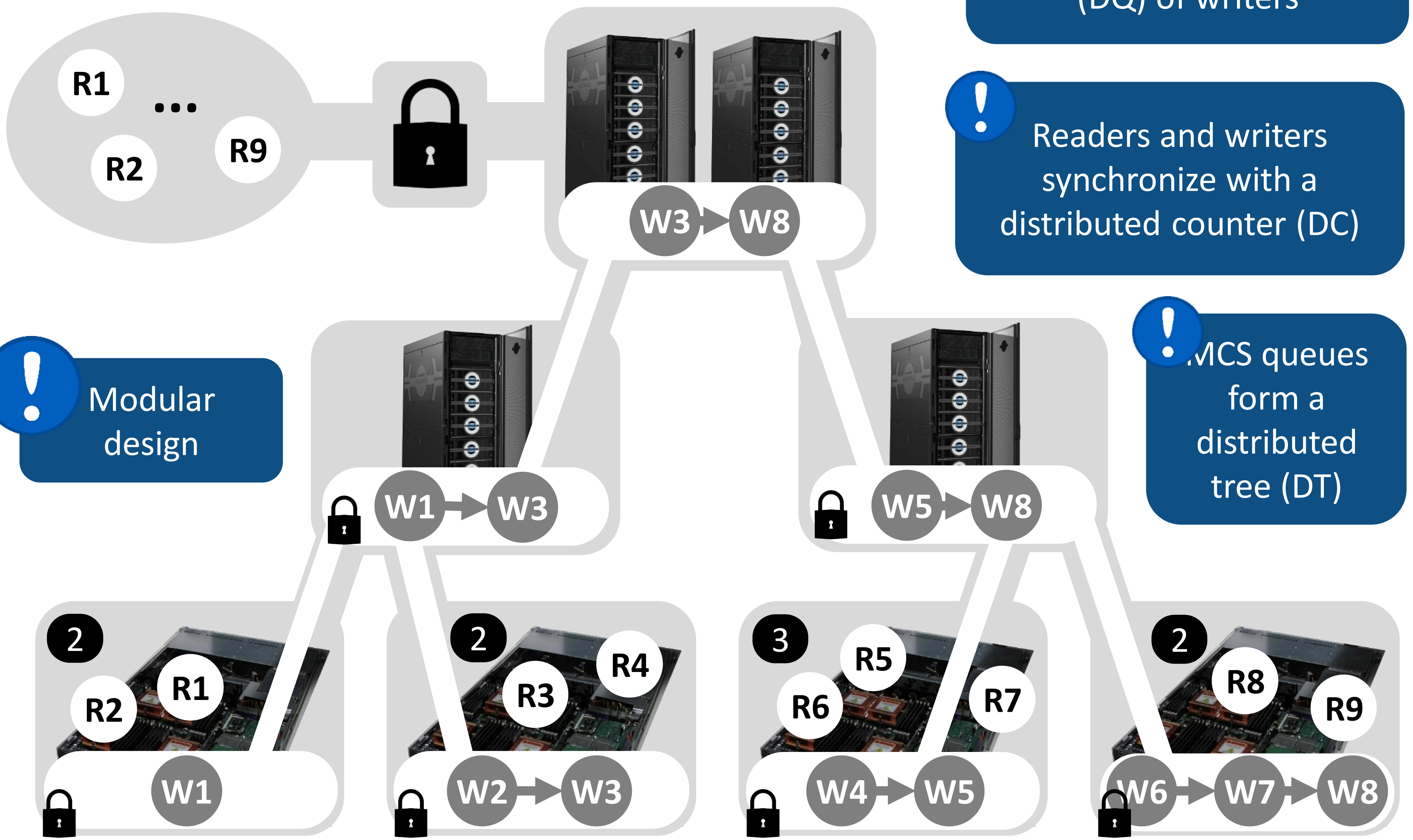


(a) Total execution

Detailed performance breakdown
network eventually limits performance

Barthels et al.: “Distributed Join Algorithms on Thousands of Cores”, VLDB’17

Large-scale RDMA Reader-Writer locking



Each lock has its own distributed MCS queue (DQ) of writers

Readers and writers synchronize with a distributed counter (DC)

MCS queues form a distributed tree (DT)

Modular design

Schmid et al.: "High-Performance Distributed RMA Locks", HPDC'16, Karsten Schwan Best Paper Award

Large-scale RDMA Reader-Writer locking

DC: every kth compute node hosts a partial counter, all of which constitute the DC.

$$k = T_{DC}$$

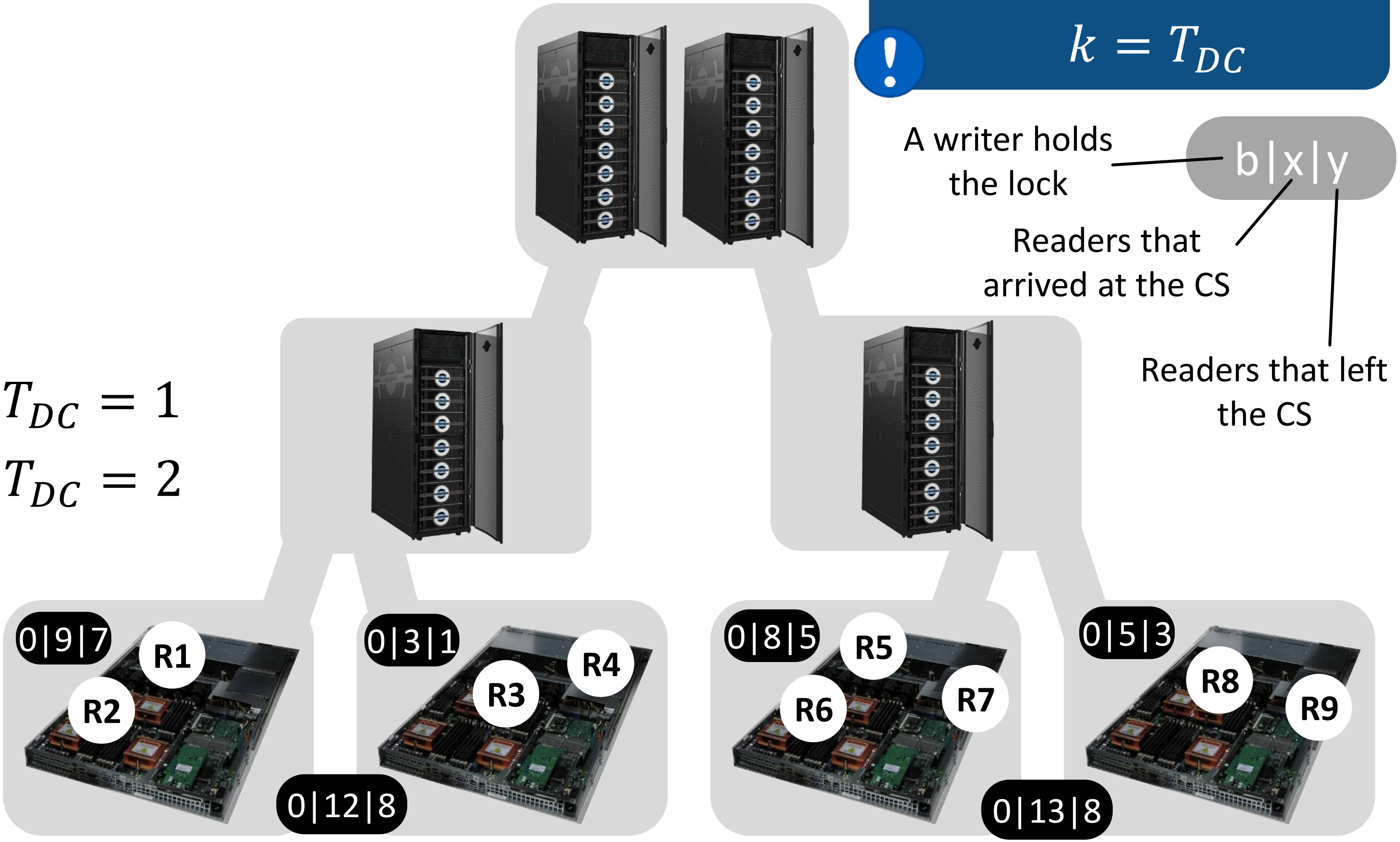
A writer holds the lock

Readers that arrived at the CS

Readers that left the CS

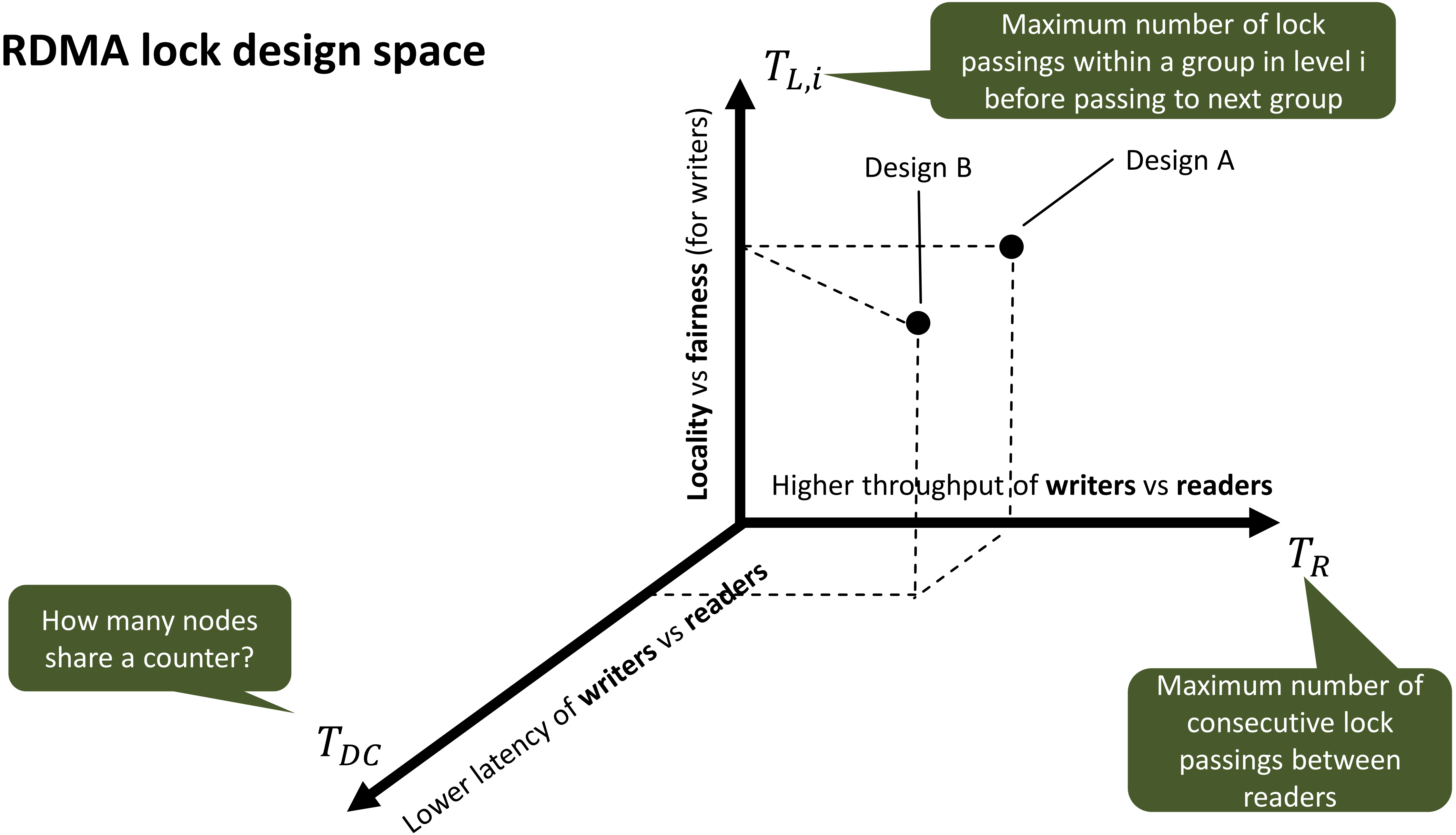
$T_{DC} = 1$

$T_{DC} = 2$

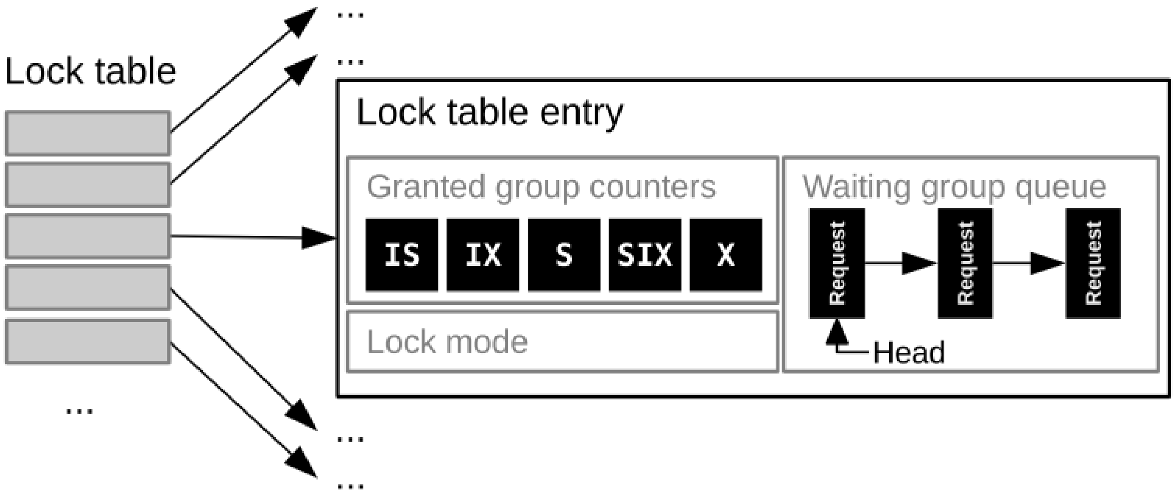


Schmid et al.: "High-Performance Distributed RMA Locks", HPDC'16, Karsten Schwan Best Paper Award

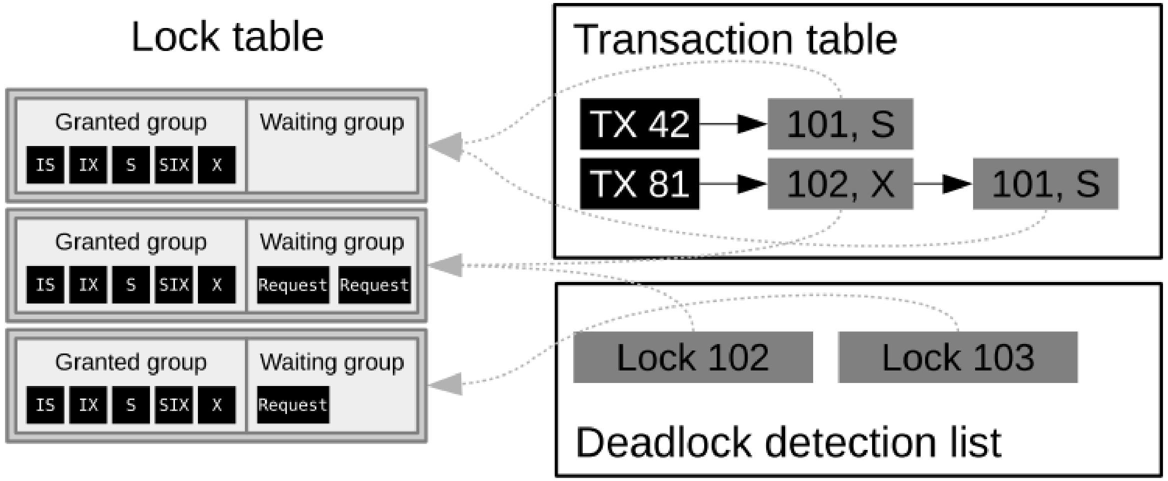
RDMA lock design space



Fast RDMA two-phase (database) locking - algorithms



(a) Lock table entry

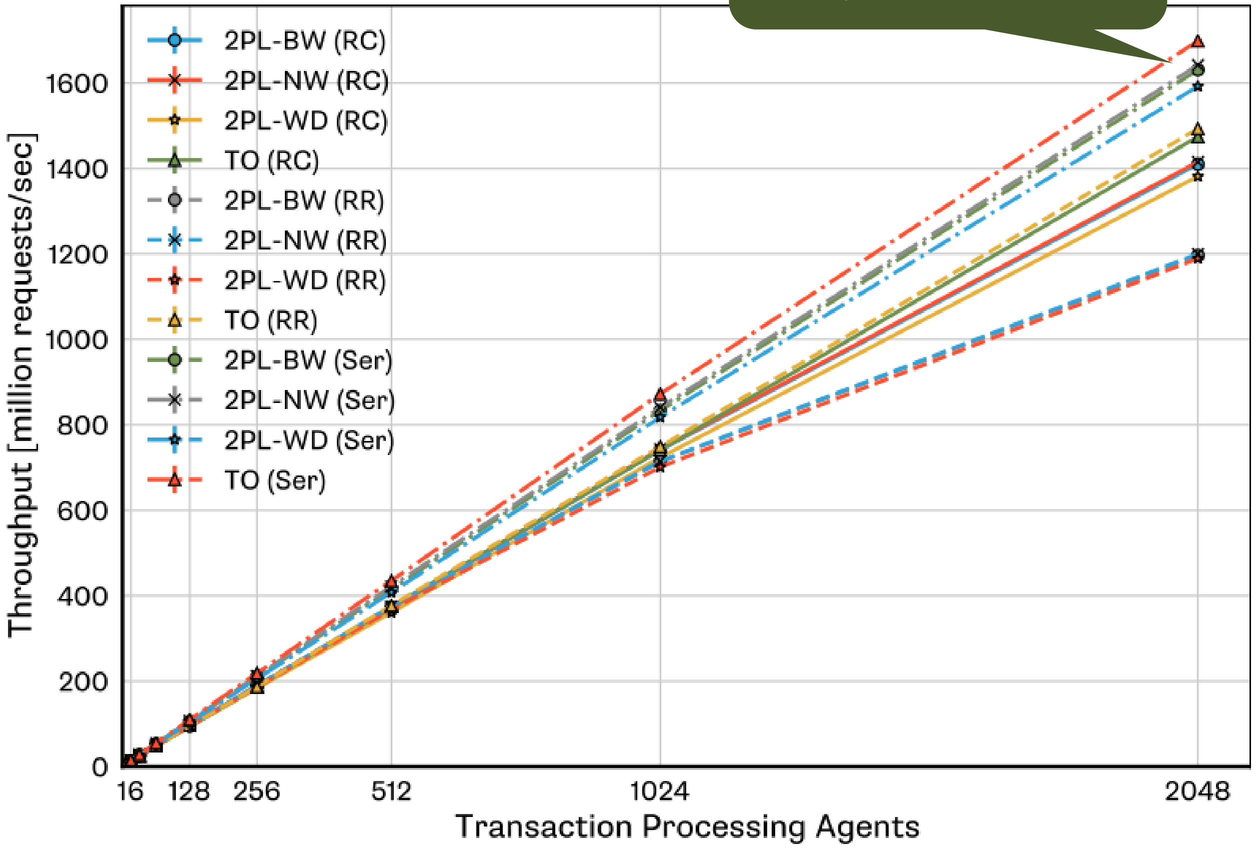


(b) Auxiliary data structures

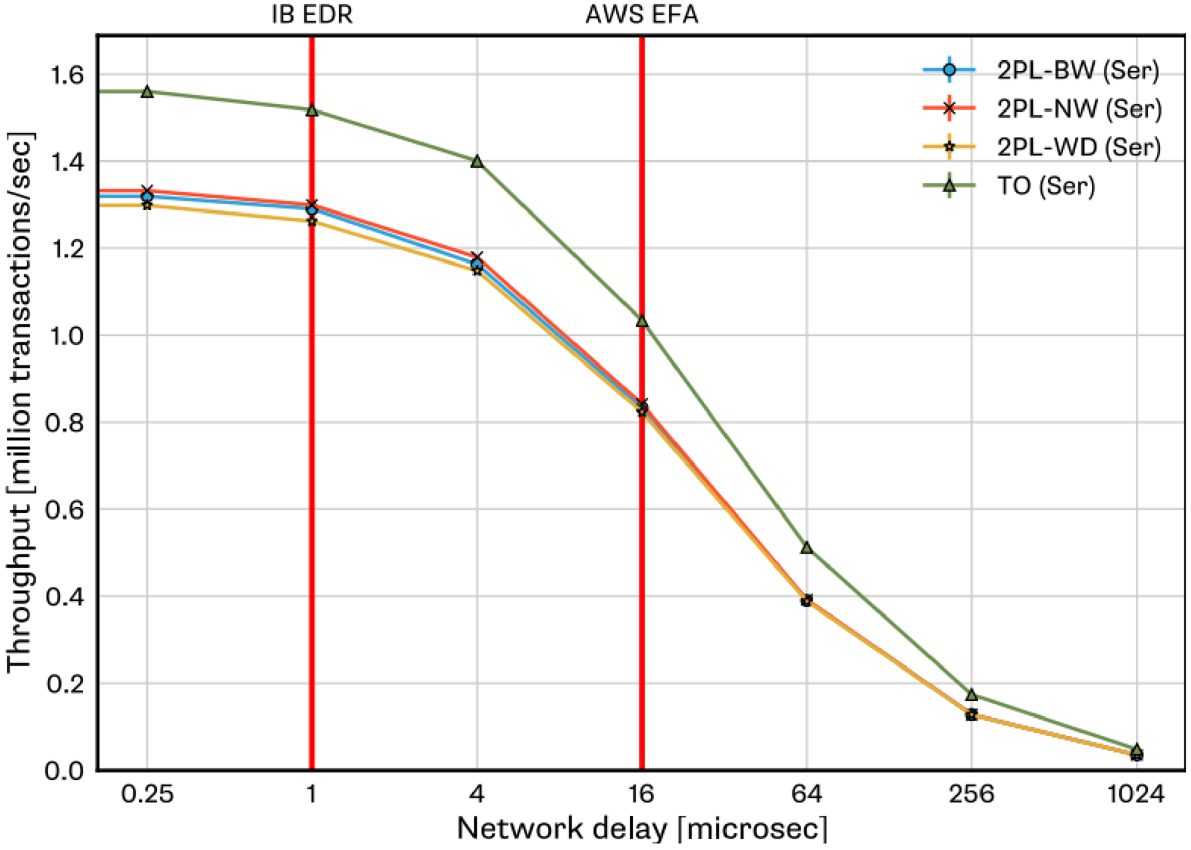
Barthels et al.: “Strong consistency is not hard to get: TwoPhase Locking and TwoPhase Commit on Thousands of Cores”, VLDB’19

Fast RDMA two-phase (database) locking - performance

38 M transactions per second



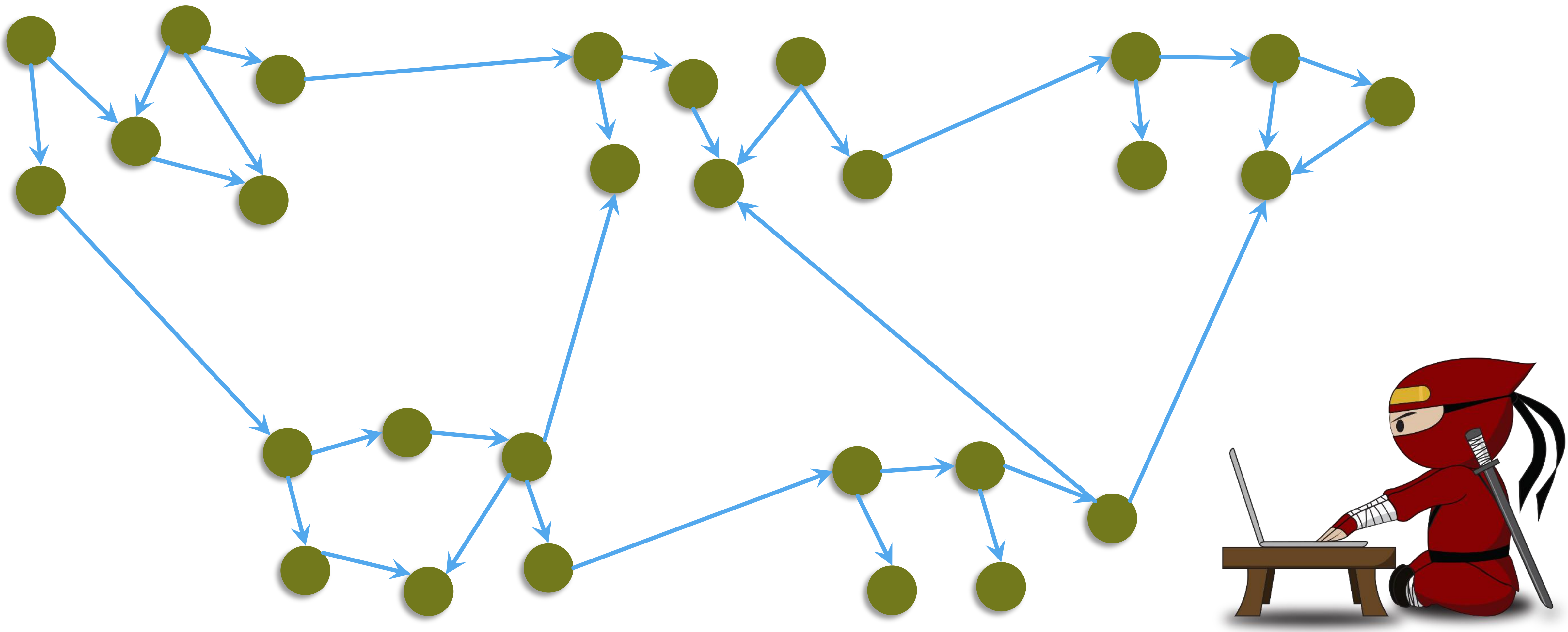
Lock requests per second on 2048 warehouses TPC-C



Latency sensitivity

Barthels et al.: “Strong consistency is not hard to get: TwoPhase Locking and TwoPhase Commit on Thousands of Cores”, VLDB’19

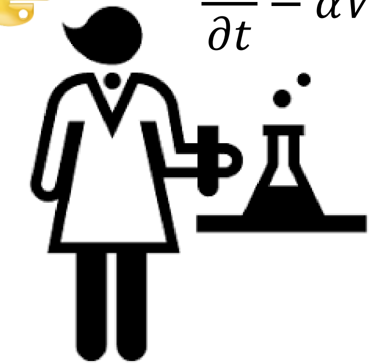
What if we could work with the cDAG abstraction directly?



The path ahead – use cDAGs directly!

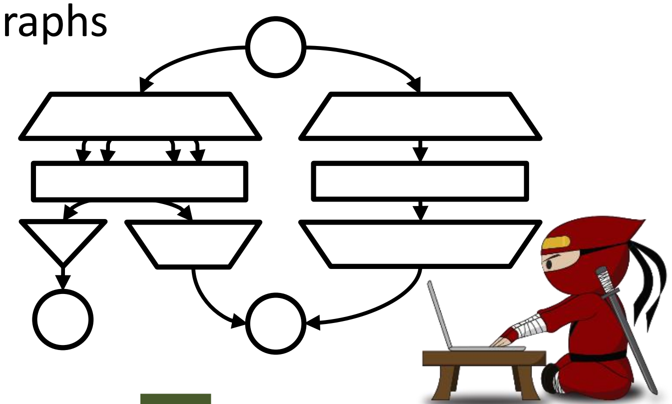
Domain-Specific Language

$$\frac{\partial u}{\partial t} - \alpha \nabla^2 u = 0$$

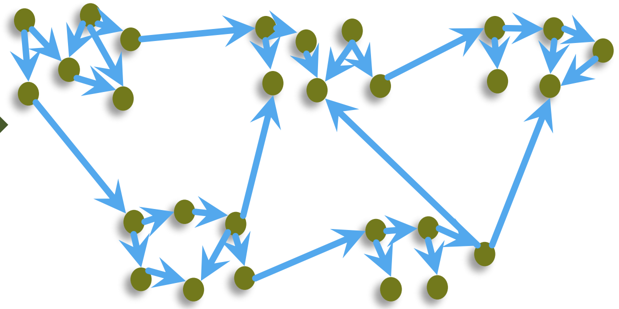


compile

Stateful (parametric) Dataflow Graphs



execute



optimize

optimize

optimize

Multi-core CPUs

GPUs

FPGA/RTL

DaCe

SPCL is hiring PhD students and highly-qualified postdocs to reach new heights!

https://spcl.inf.ethz.ch/Jobs/