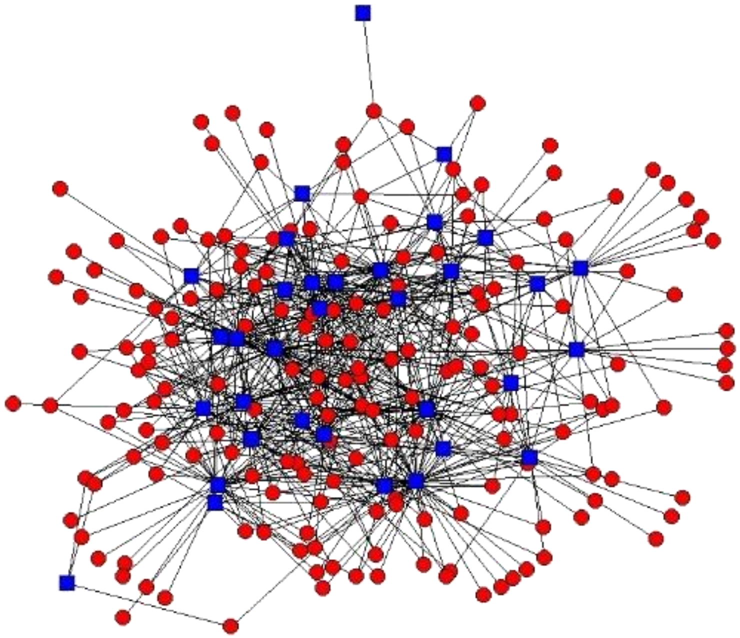
Towards scalable RDMA locking on a NIC

TORSTEN HOEFLER

with support of Patrick Schmid, Maciej Besta, Salvatore di Girolamo @ SPCL

presented at HP Labs, Palo Alto, CA, USA

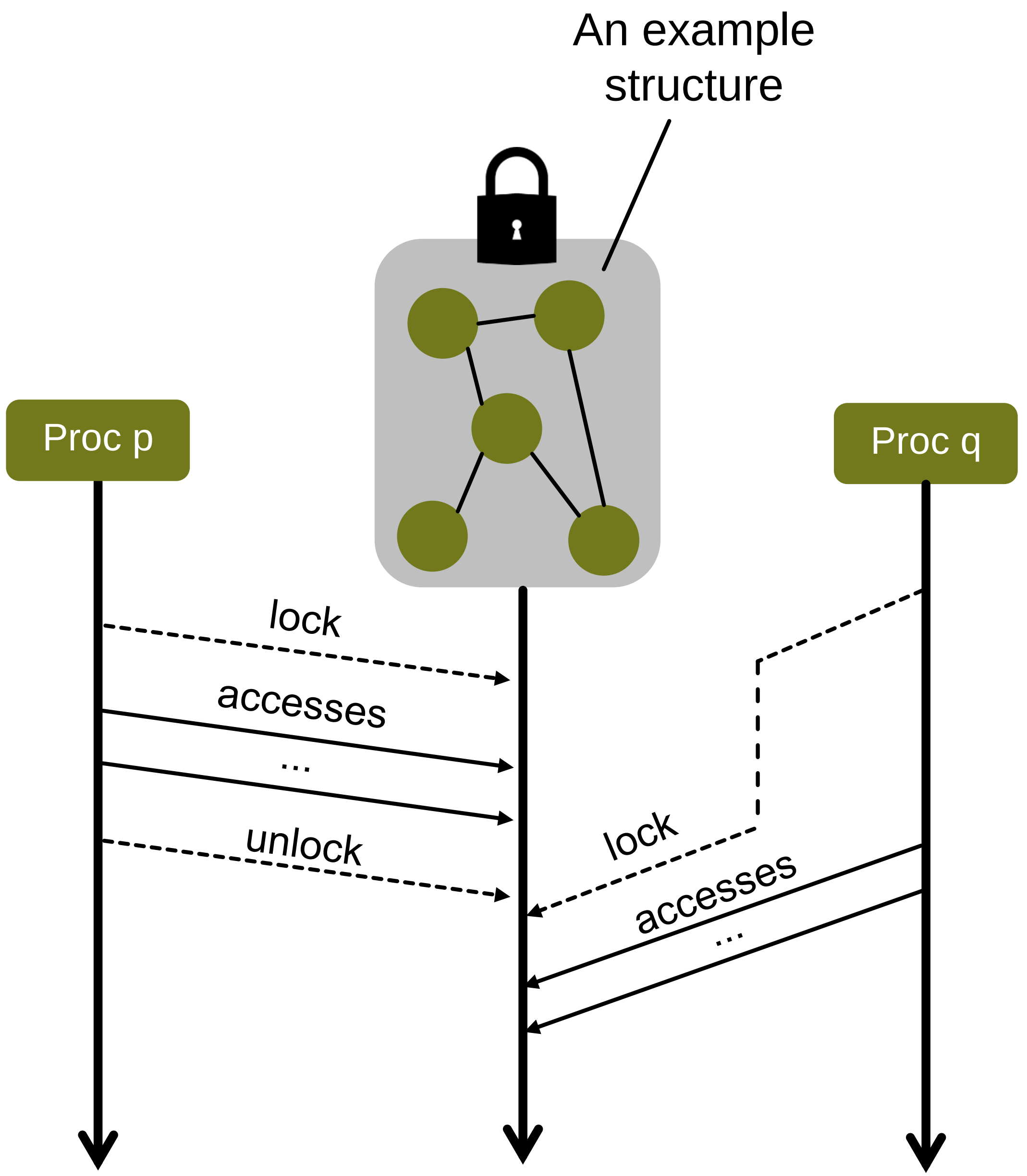
NEED FOR EFFICIENT LARGE-SCALE SYNCHRONIZATION



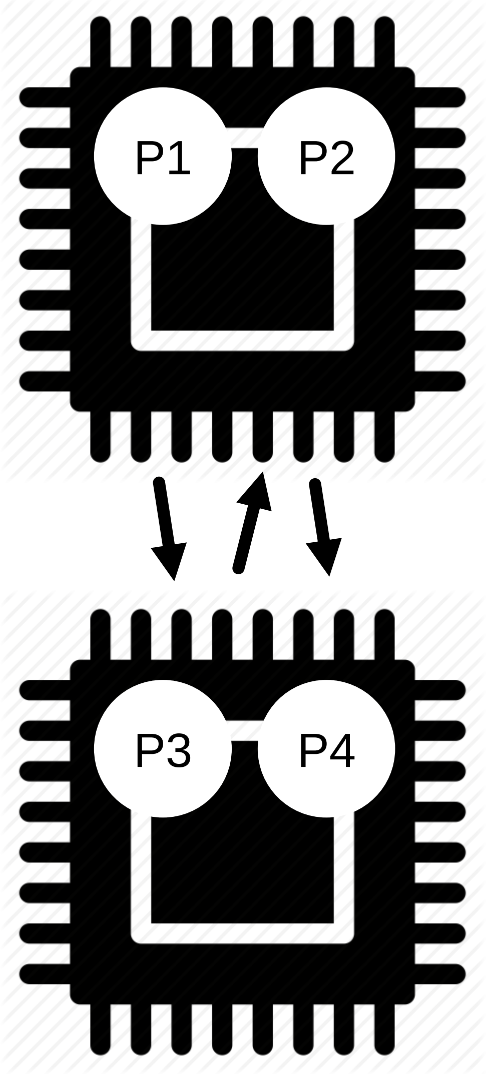
Locks

Inuitive semantics

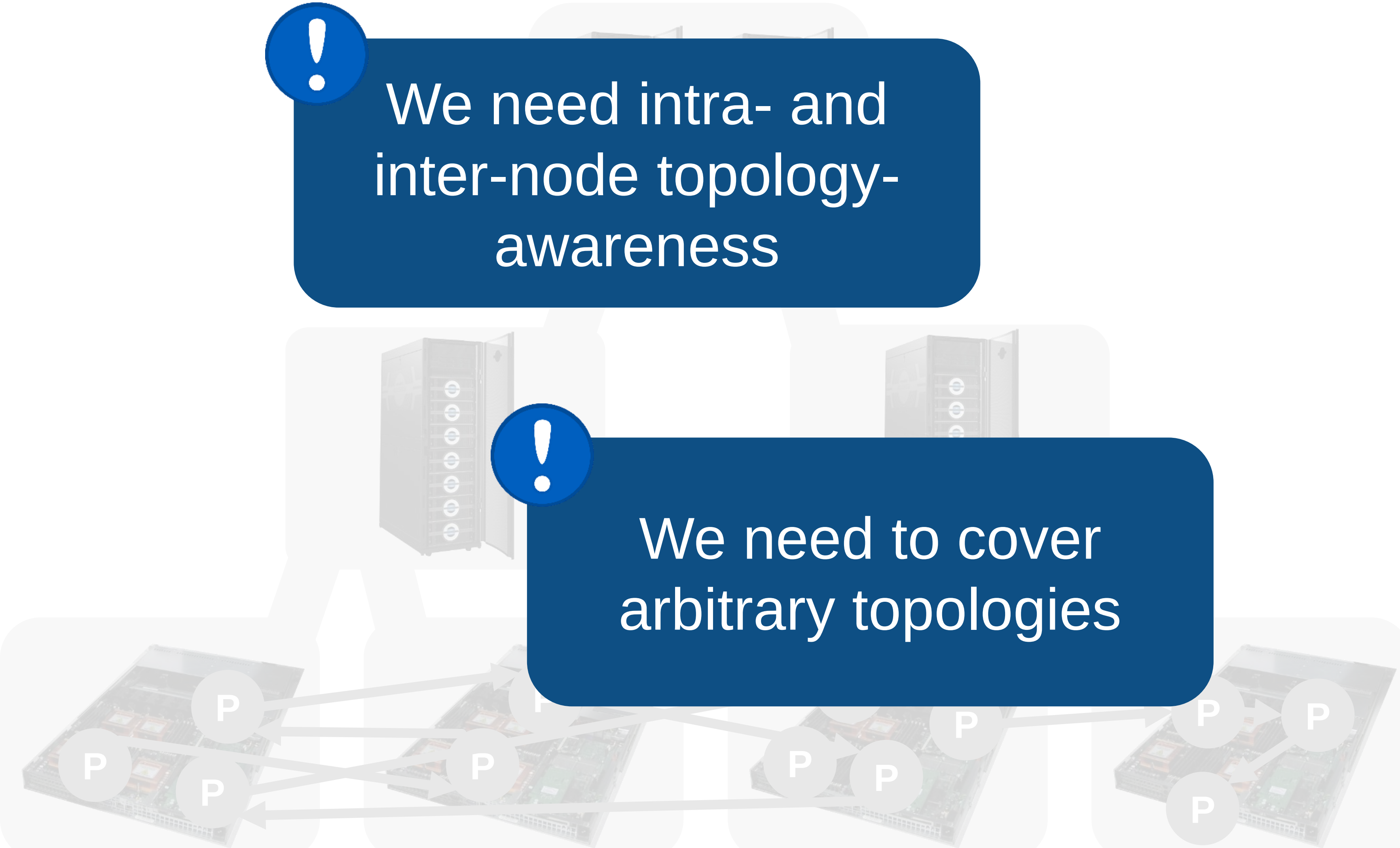
Various performance penalties



LOCKS: CHALLENGES



LOCKS: CHALLENGES



We need intra- and inter-node topology-awareness

We need to cover arbitrary topologies

Locks: Challenges

Reader

Reader

Reader

Writer

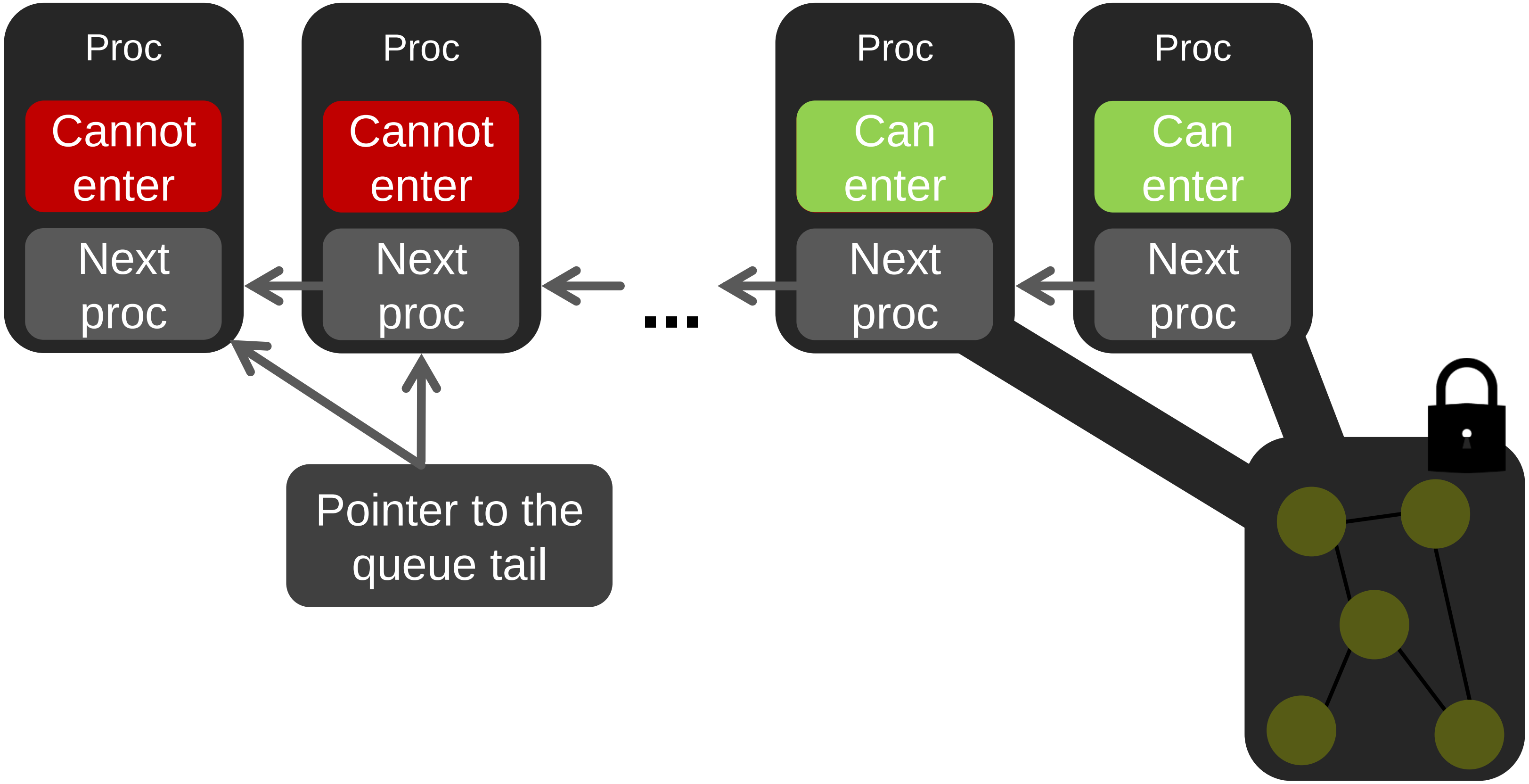
We need to distinguish between readers and writers

We need flexible performance for both types of processes

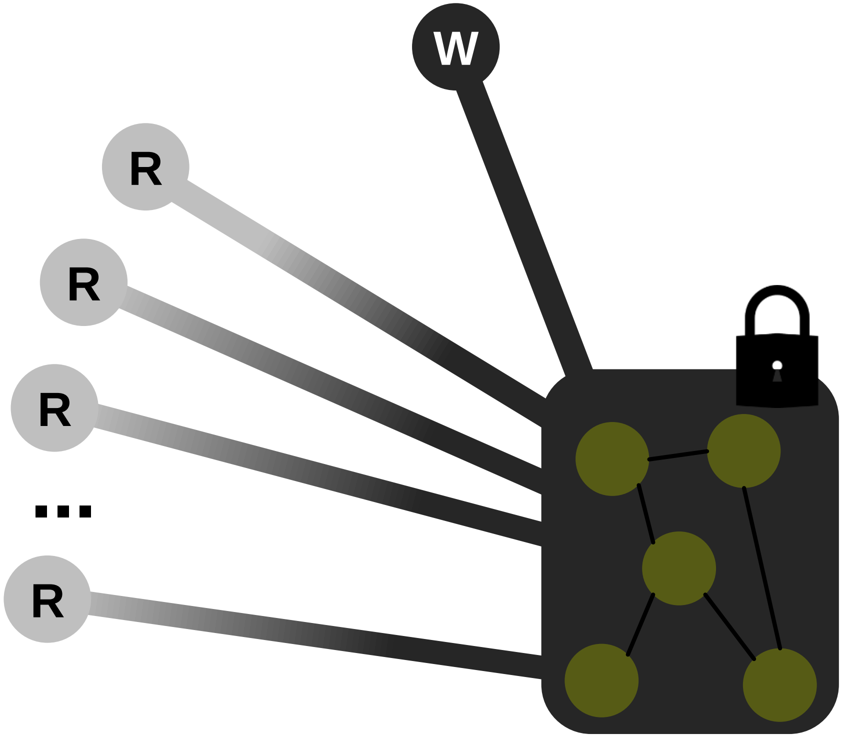
[1] V. Venkataramani et al. Tao: How facebook serves the social graph. SIGMOD'12.

What will we use in the design?

What we will use
MCS Locks



WHAT WE WILL USE
Reader-Writer Locks

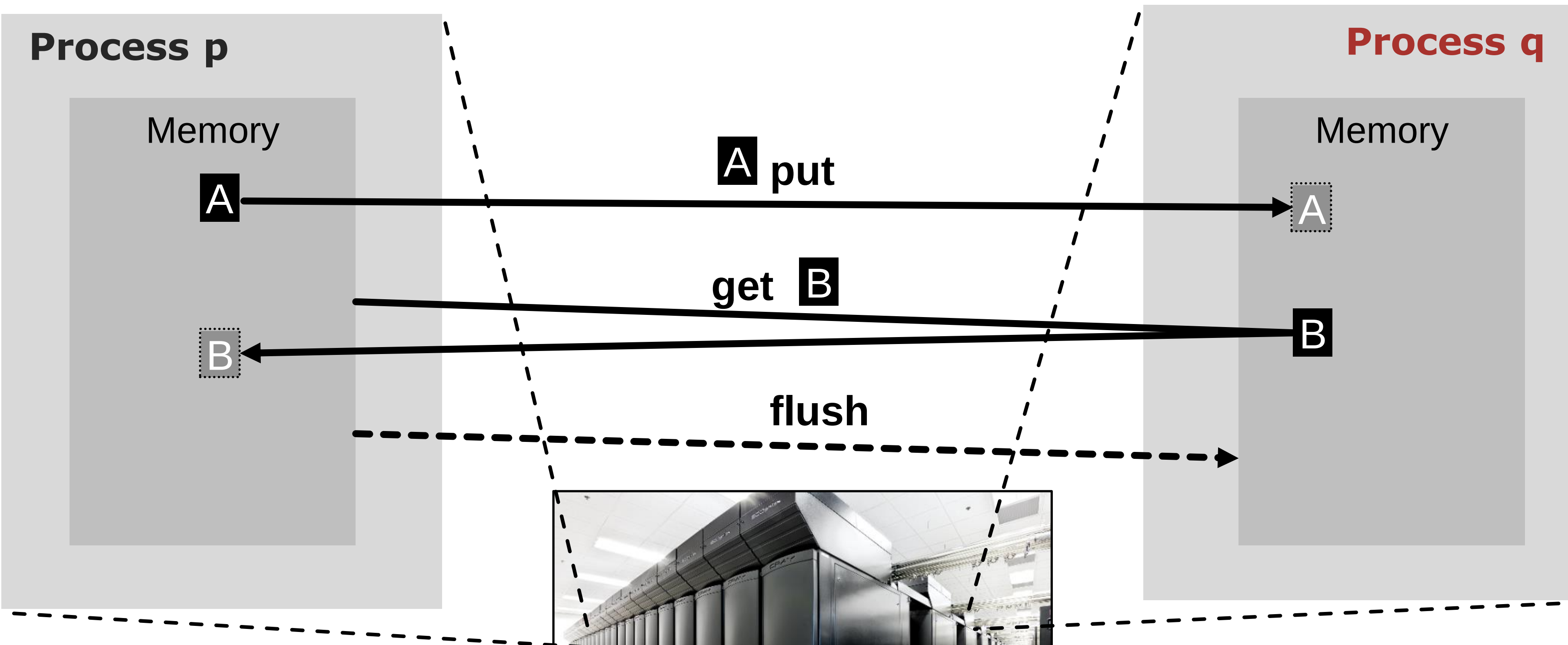


How to manage the design complexity?

How to ensure tunable performance?

What mechanism to use for efficient implementation?

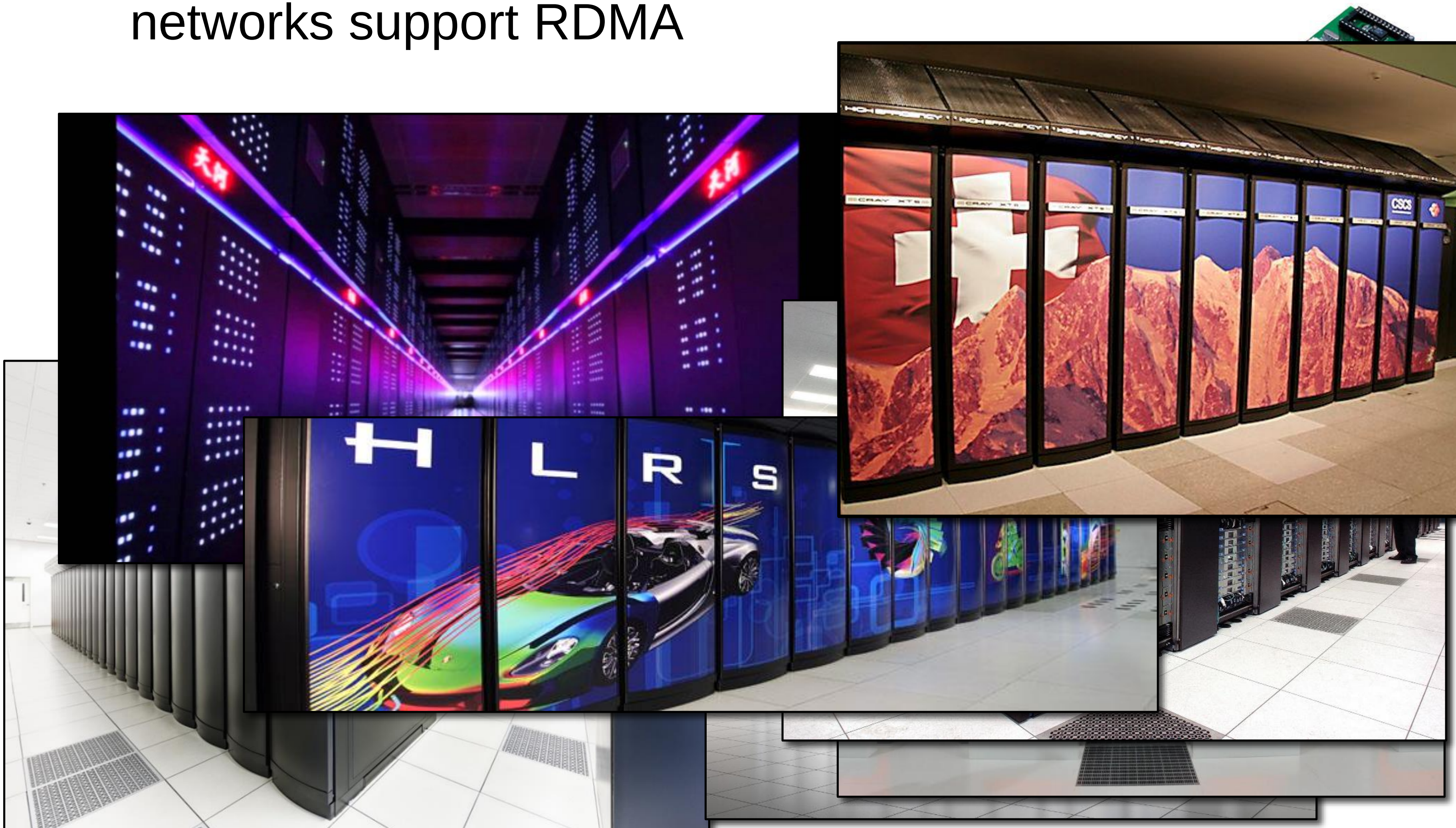
Remote Memory Access (RMA) Programming



Cray
BlueWaters

Remote Memory Access Programming

- Implemented in hardware in NICs in the majority of HPC networks support RDMA



How to manage the design complexity?

How to ensure tunable performance?

What mechanism to use for efficient implementation?

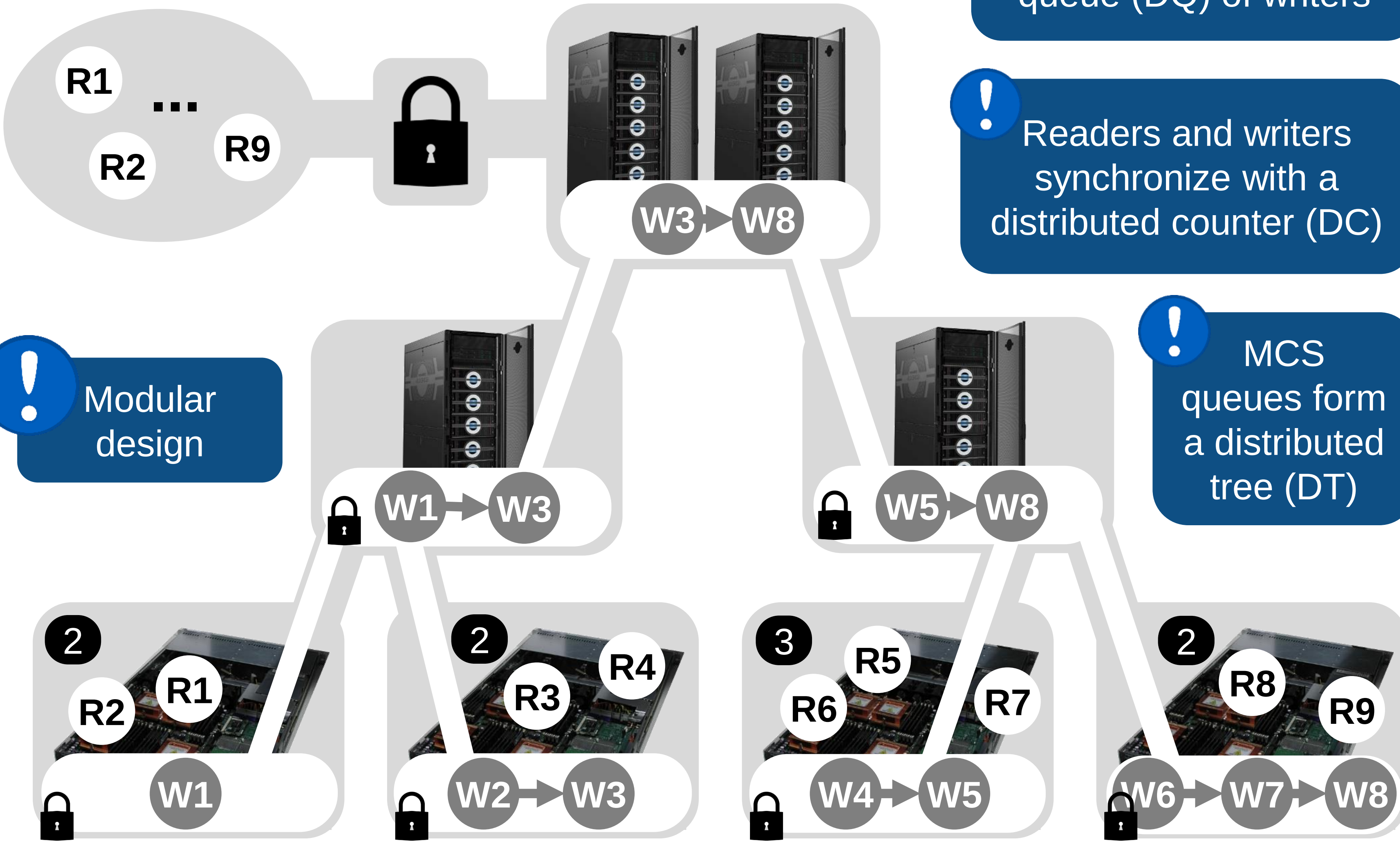
How to manage the design complexity?

Each element 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



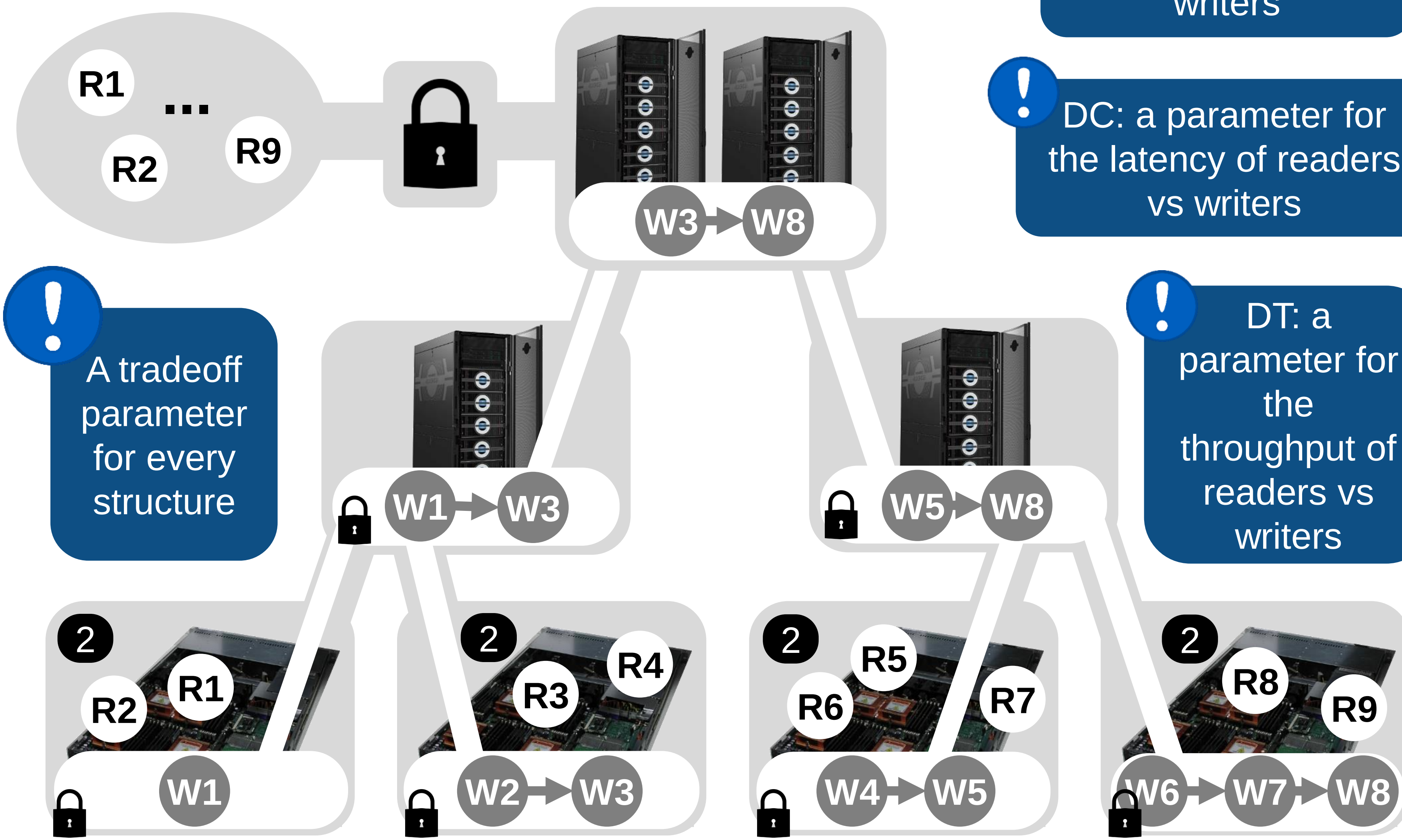
How to ensure tunable performance?

Each DQ: fairness vs throughput of writers

DC: a parameter for the latency of readers vs writers

A tradeoff parameter for every structure

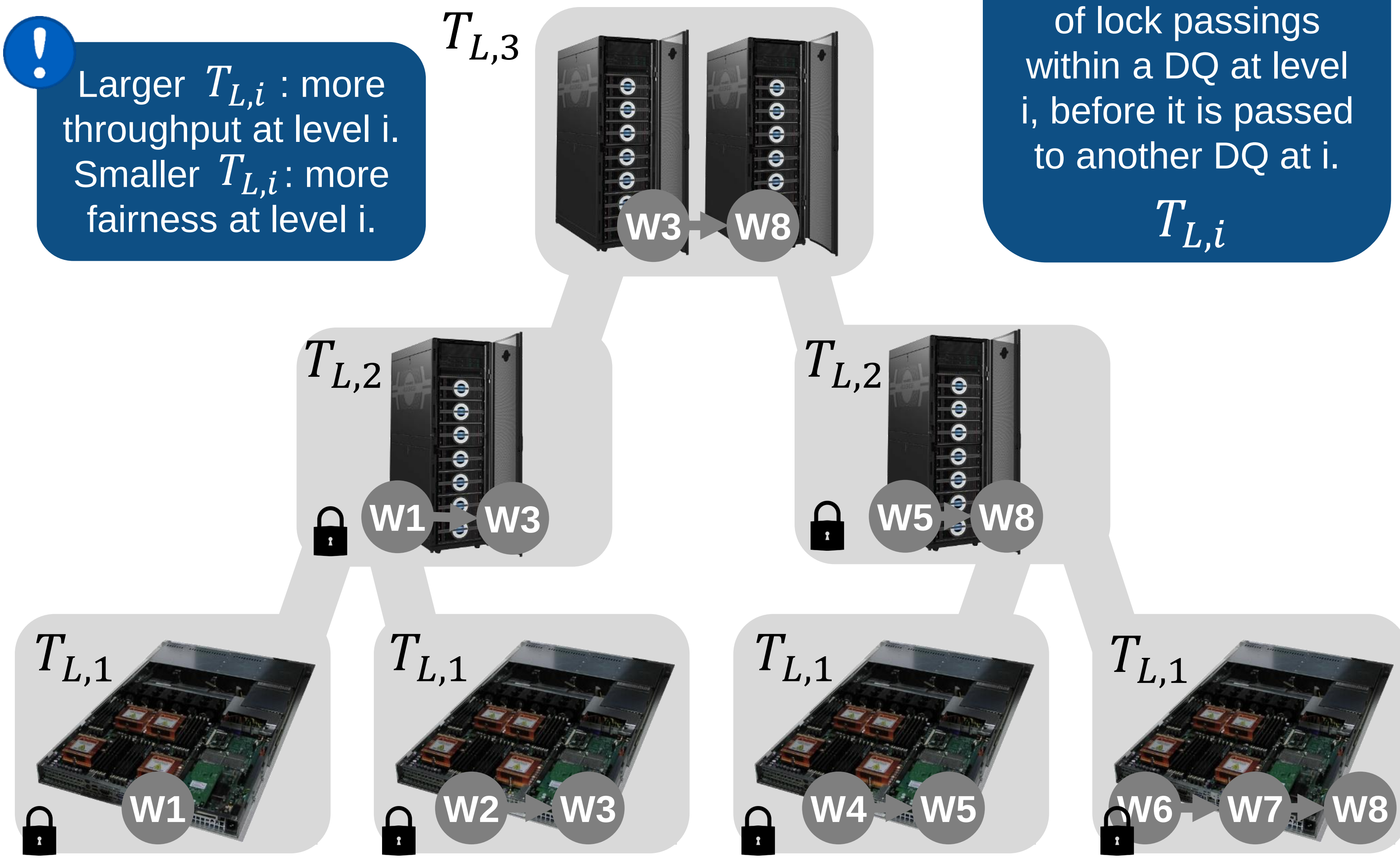
DT: a parameter for the throughput of readers vs writers



Distributed MCS Queues (DQs)
Throughput vs Fairness

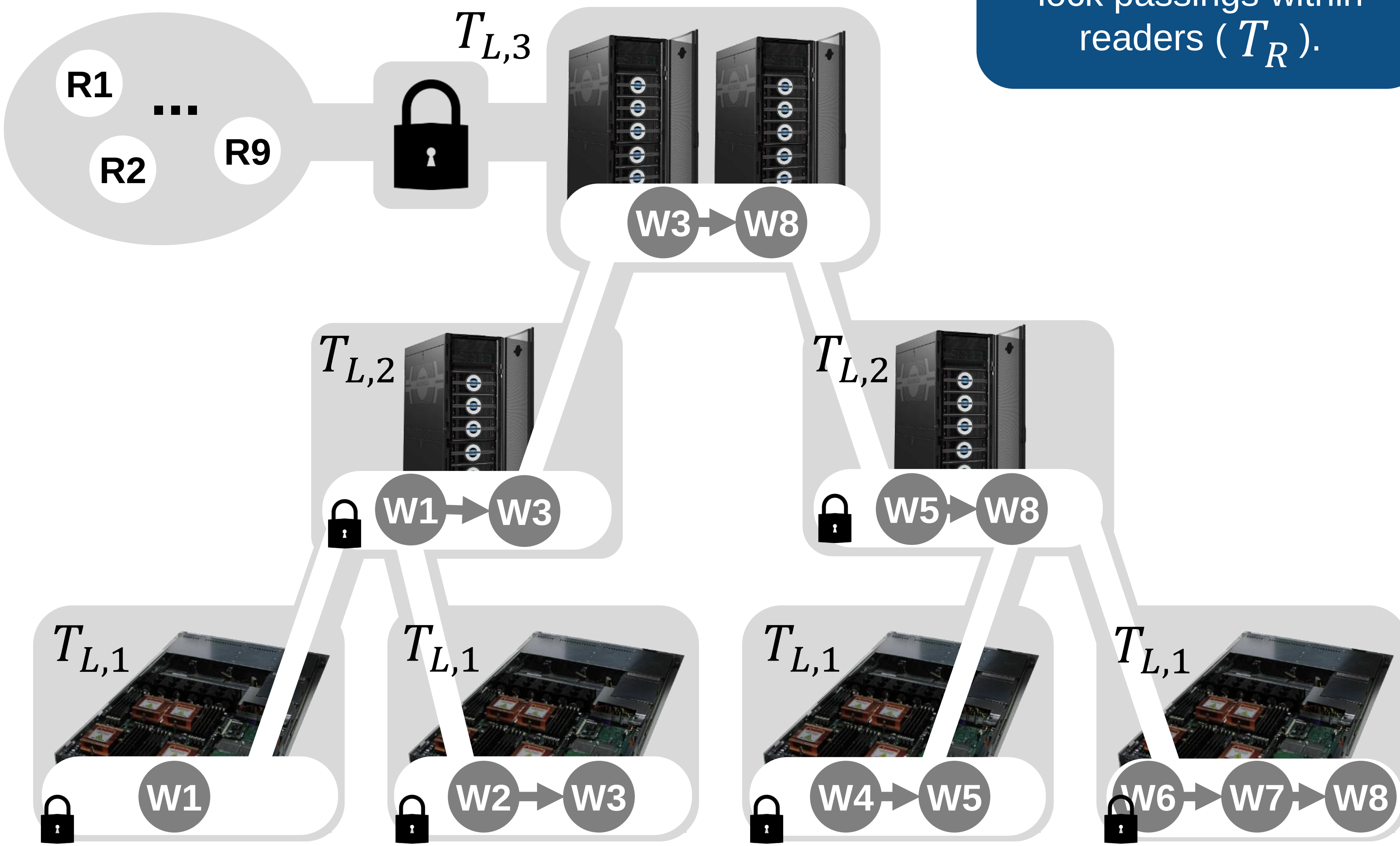
Larger $T_{L,i}$: more throughput at level i. Smaller $T_{L,i}$: more fairness at level i.

Each DQ: The maximum number of lock passings within a DQ at level i, before it is passed to another DQ at i.
$T_{L,i}$



Distributed Tree of Queues (DT)
Throughput of readers vs writers

DT: The maximum number of consecutive lock passings within readers (T_R).



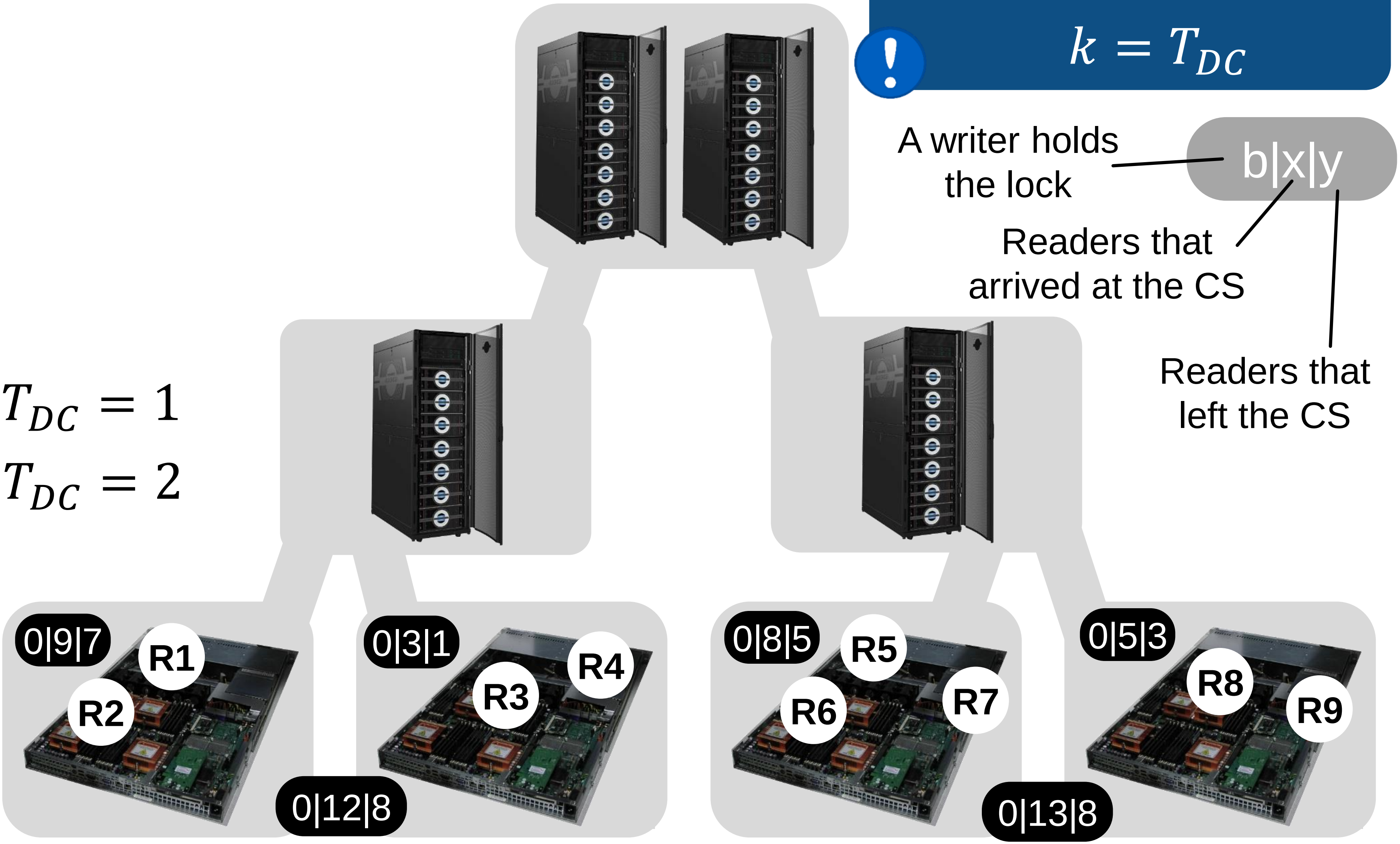
Distributed Counter (DC)
Latency of readers vs writers

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

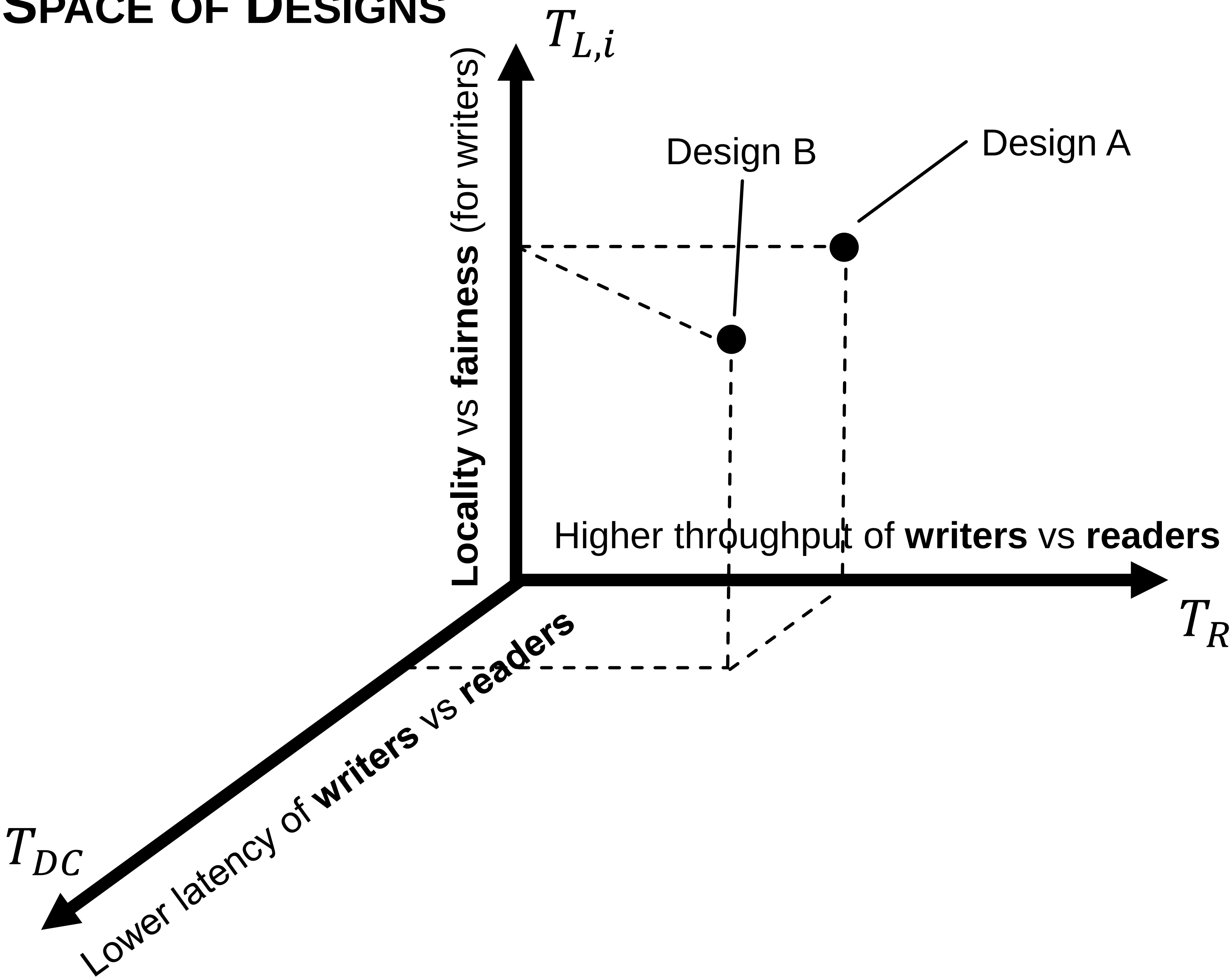
$$k = T_{DC}$$

$T_{DC} = 1$

$T_{DC} = 2$

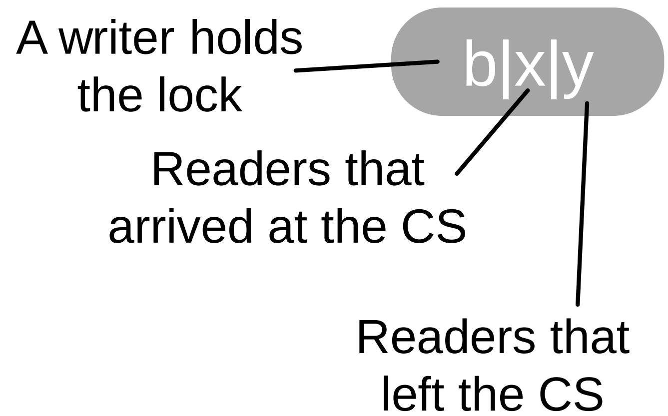


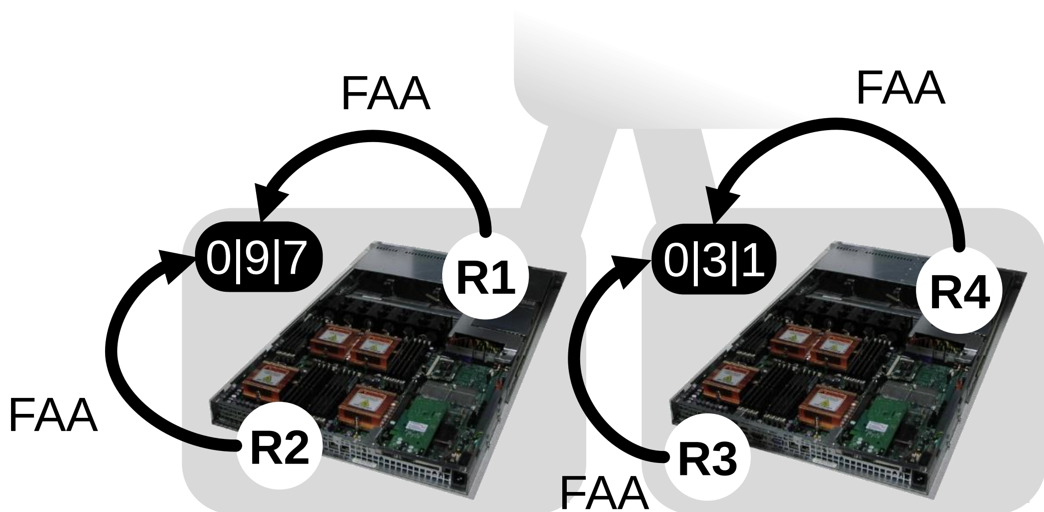
The Space of Designs



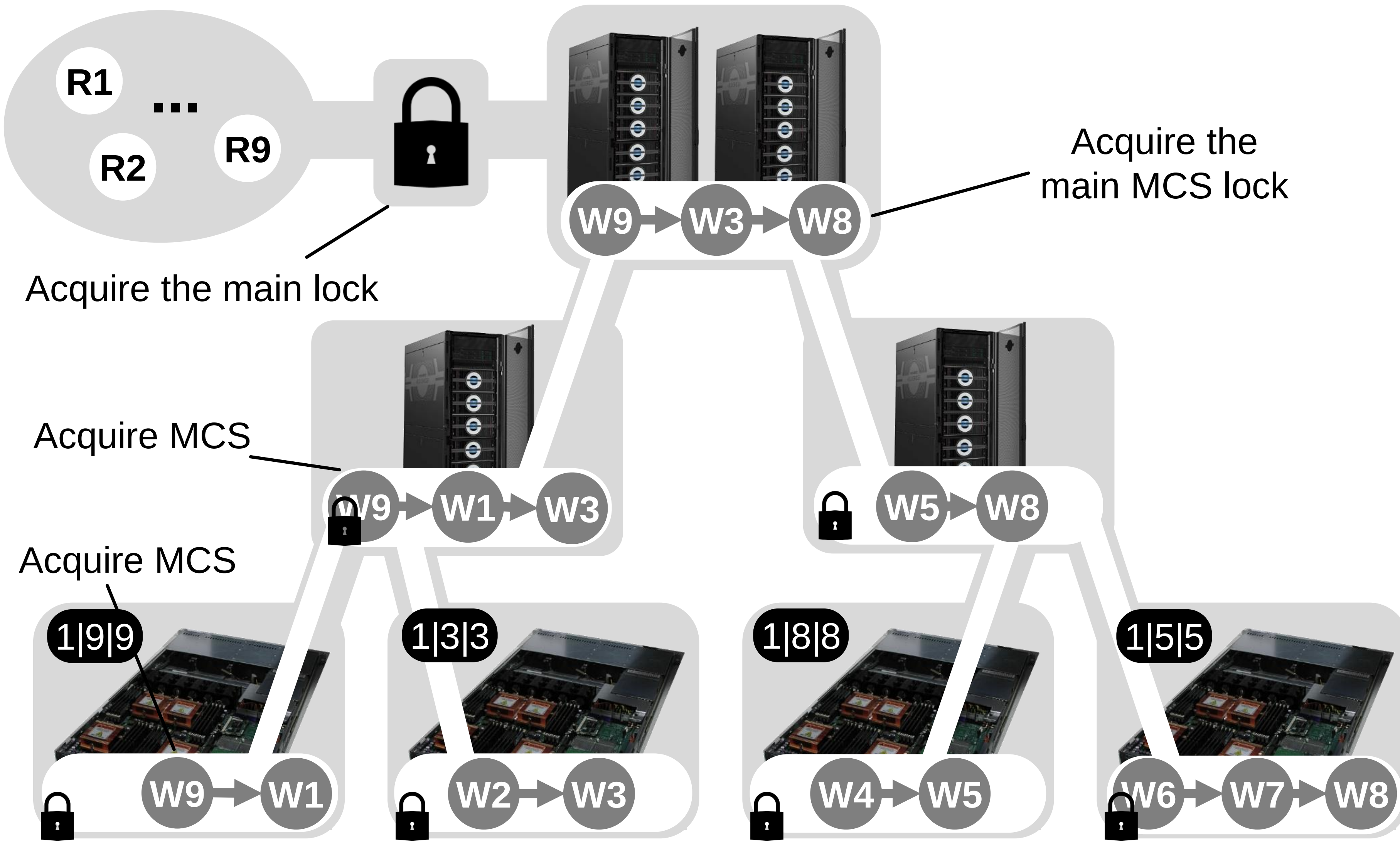
Lock Acquire by Readers

A lightweight acquire protocol for readers: only one atomic fetch-and-add (FAA) operation



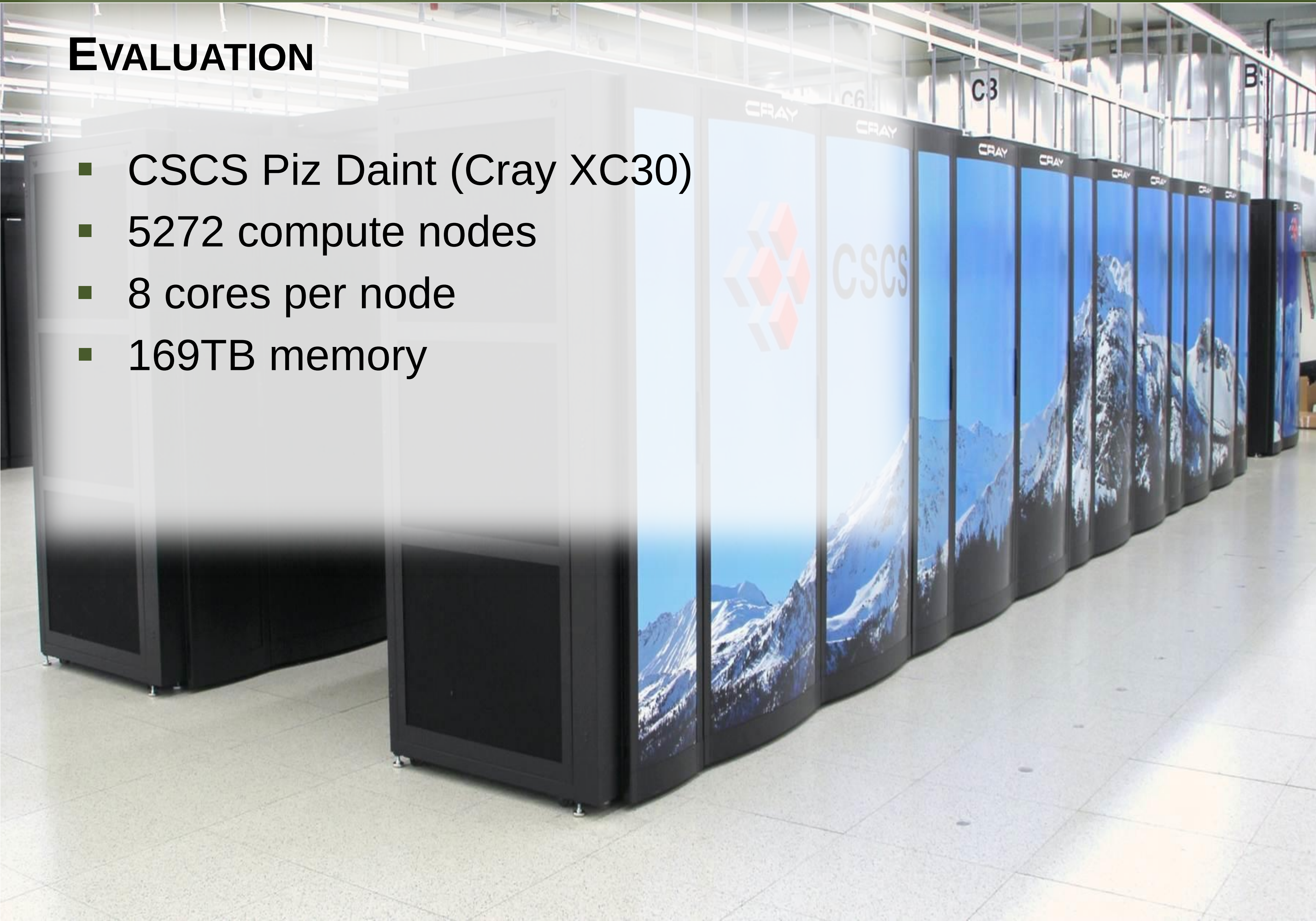


Lock Acquire by Writers



EVALUATION

- CSCS Piz Daint (Cray XC30)
- 5272 compute nodes
- 8 cores per node
- 169TB memory



EVALUATION
CONSIDERED BENCHMARKS

The **latency** benchmark

Throughput benchmarks:

Empty-critical-section

Single-operation

Wait-after-release

Workload-critical-section

DHT

Distributed hashtable evaluation

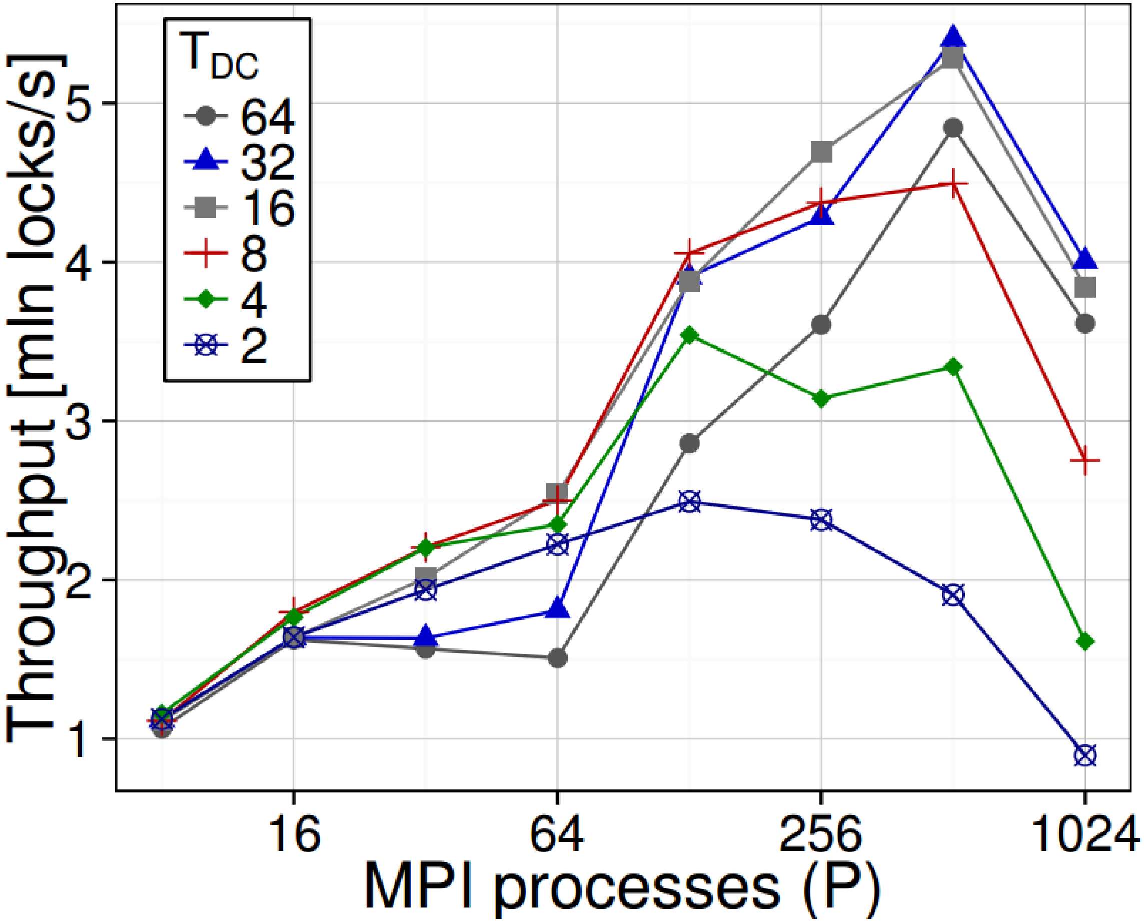
EVALUATION
DISTRIBUTED COUNTER ANALYSIS

0|9|7 0|3|1

0|12|8

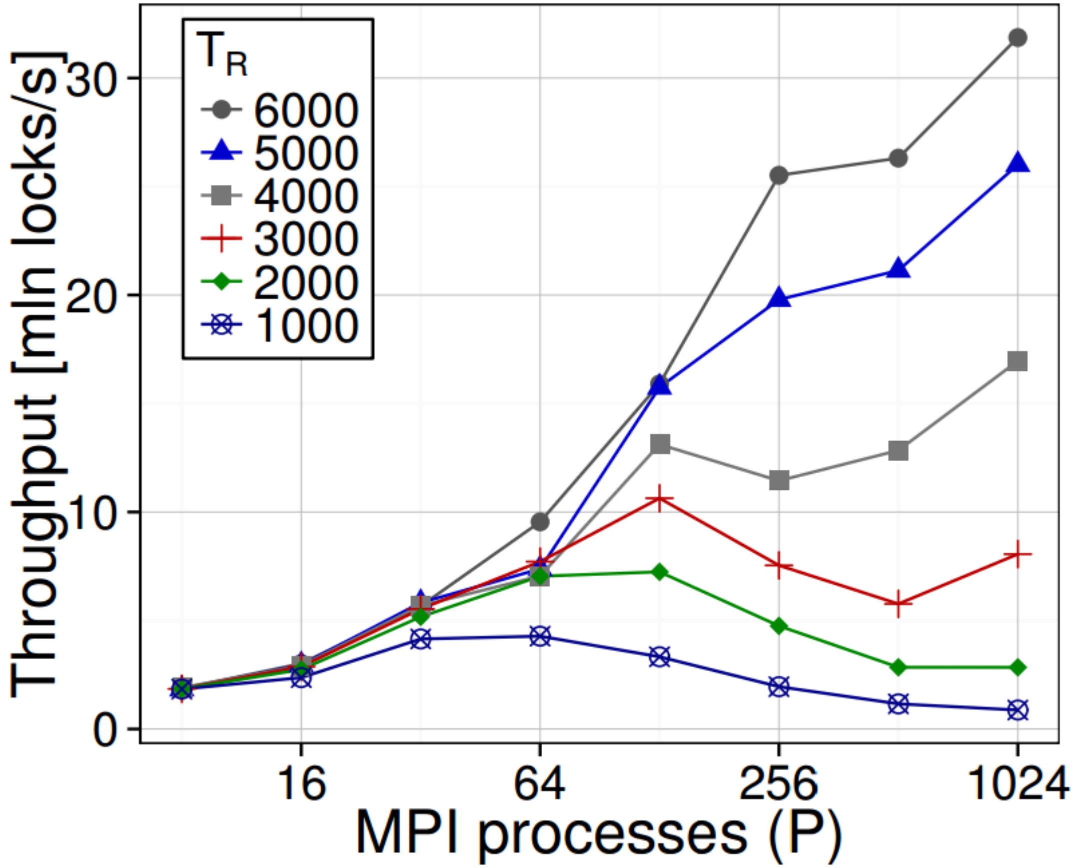
Throughput, 2% writers

Single-operation benchmark

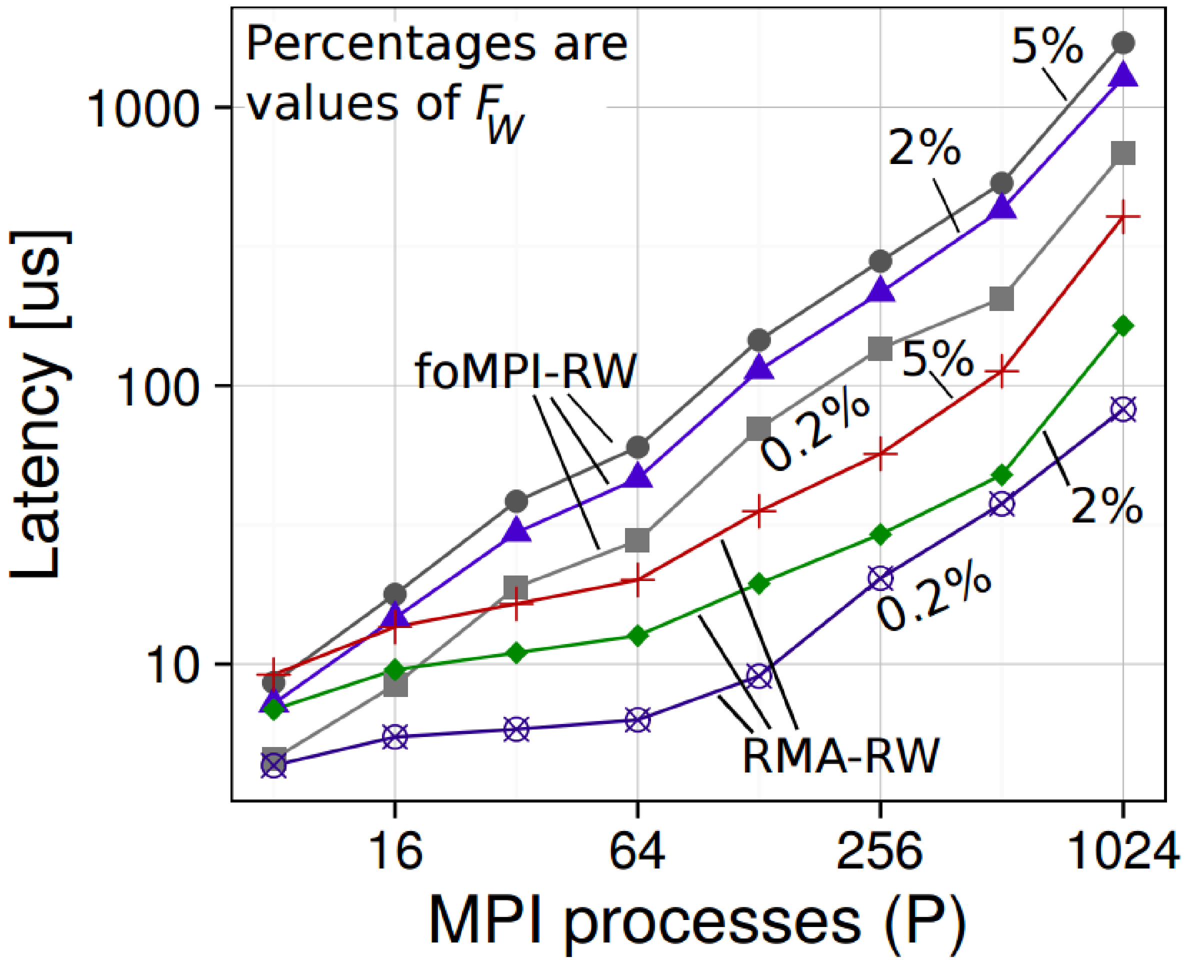


EVALUATION
READER THRESHOLD ANALYSIS

Throughput, 0.2% writers,
Empty-critical-section benchmark



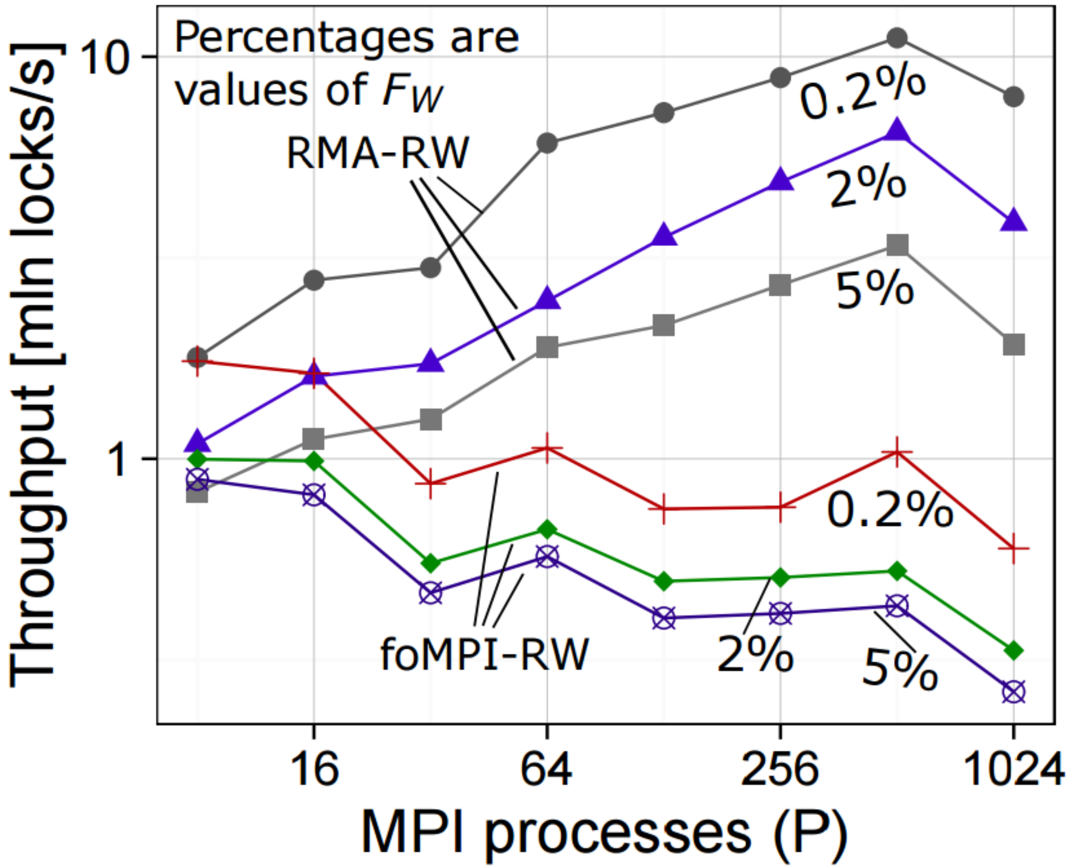
Evaluation
Comparison to the State-of-the-Art



[1] R. Gerstenberger et al. Enabling Highly-scalable Remote Memory Access Programming with MPI-3 One Sided. ACM/IEEE Supercomputing 2013.

Evaluation
Comparison to the State-of-the-Art

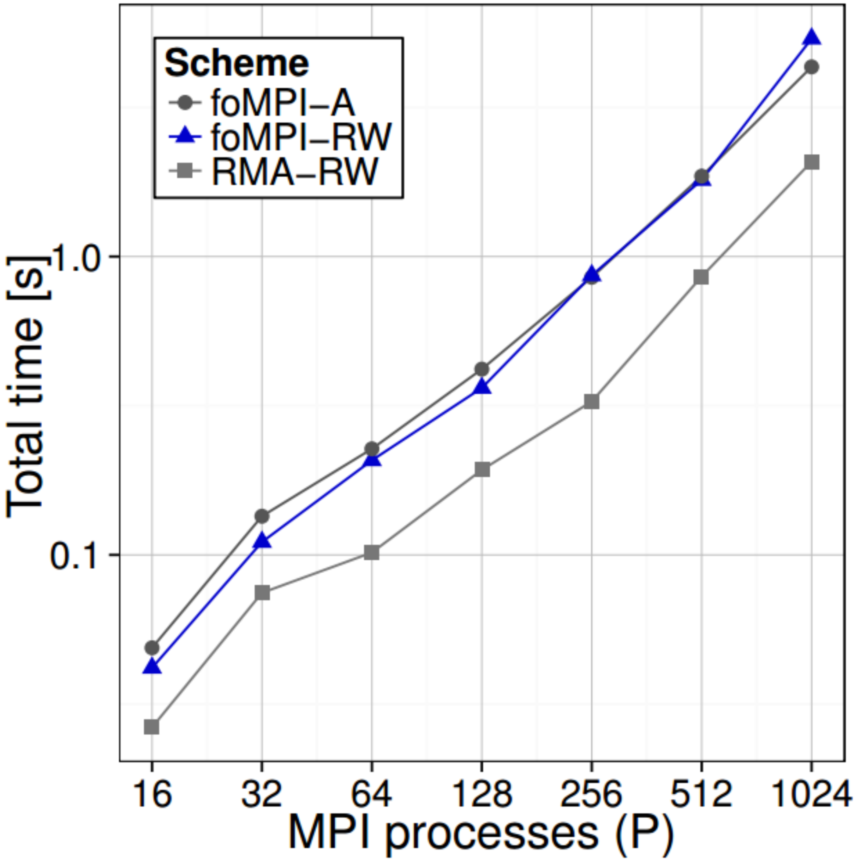
Throughput, single-operation benchmark



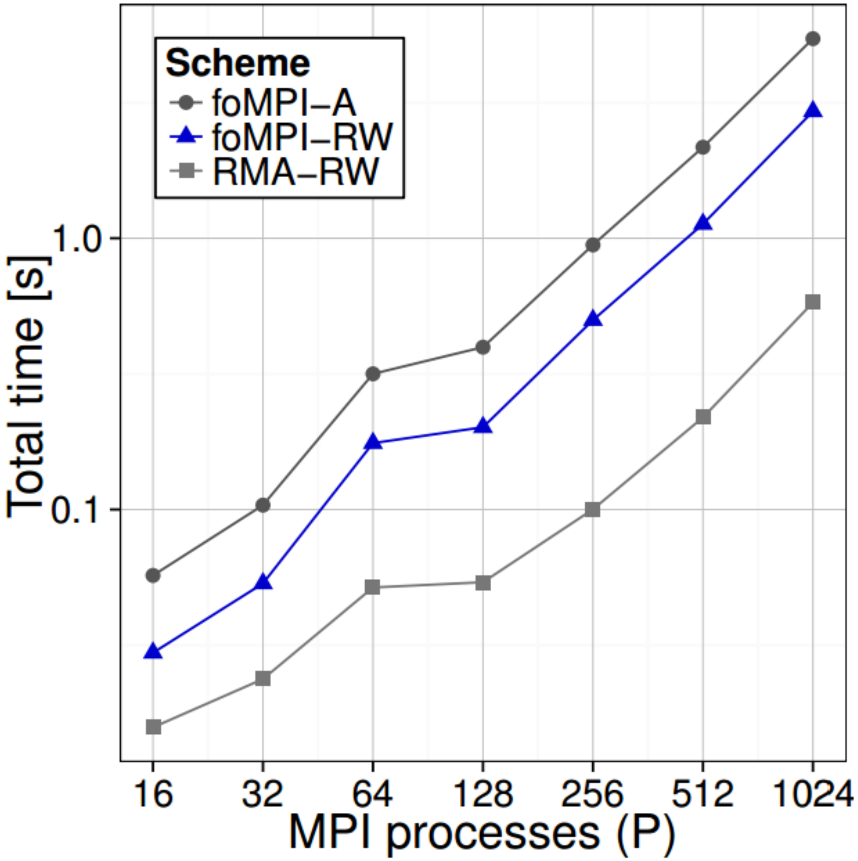
[1] R. Gerstenberger et al. Enabling Highly-scalable Remote Memory Access Programming with MPI-3 One Sided. ACM/IEEE Supercomputing 2013.

EVALUATION
DISTRIBUTED HASHTABLE

20% writers



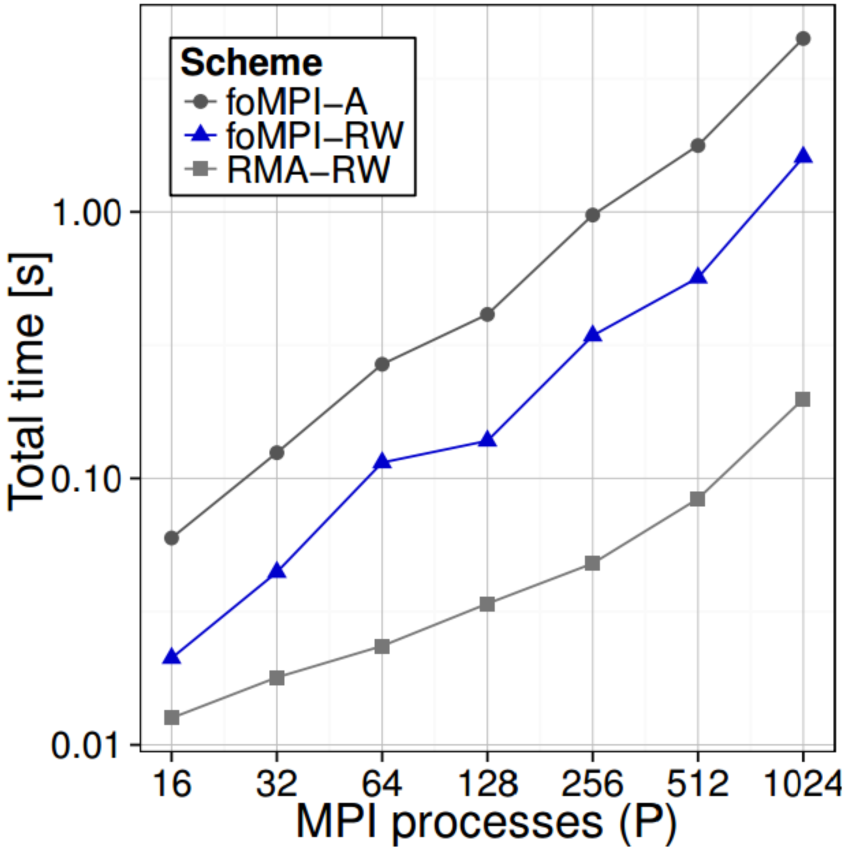
10% writers



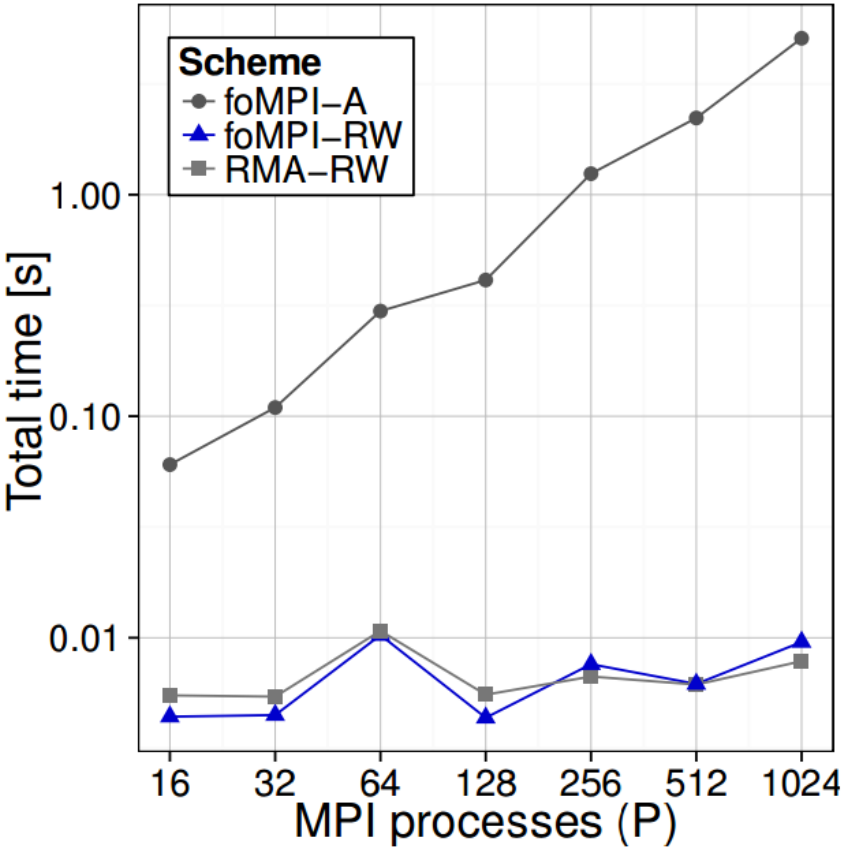
[1] R. Gerstenberger et al. Enabling Highly-scalable Remote Memory Access Programming with MPI-3 One Sided. ACM/IEEE Supercomputing 2013.

Evaluation
Distributed Hashtable

2% of writers

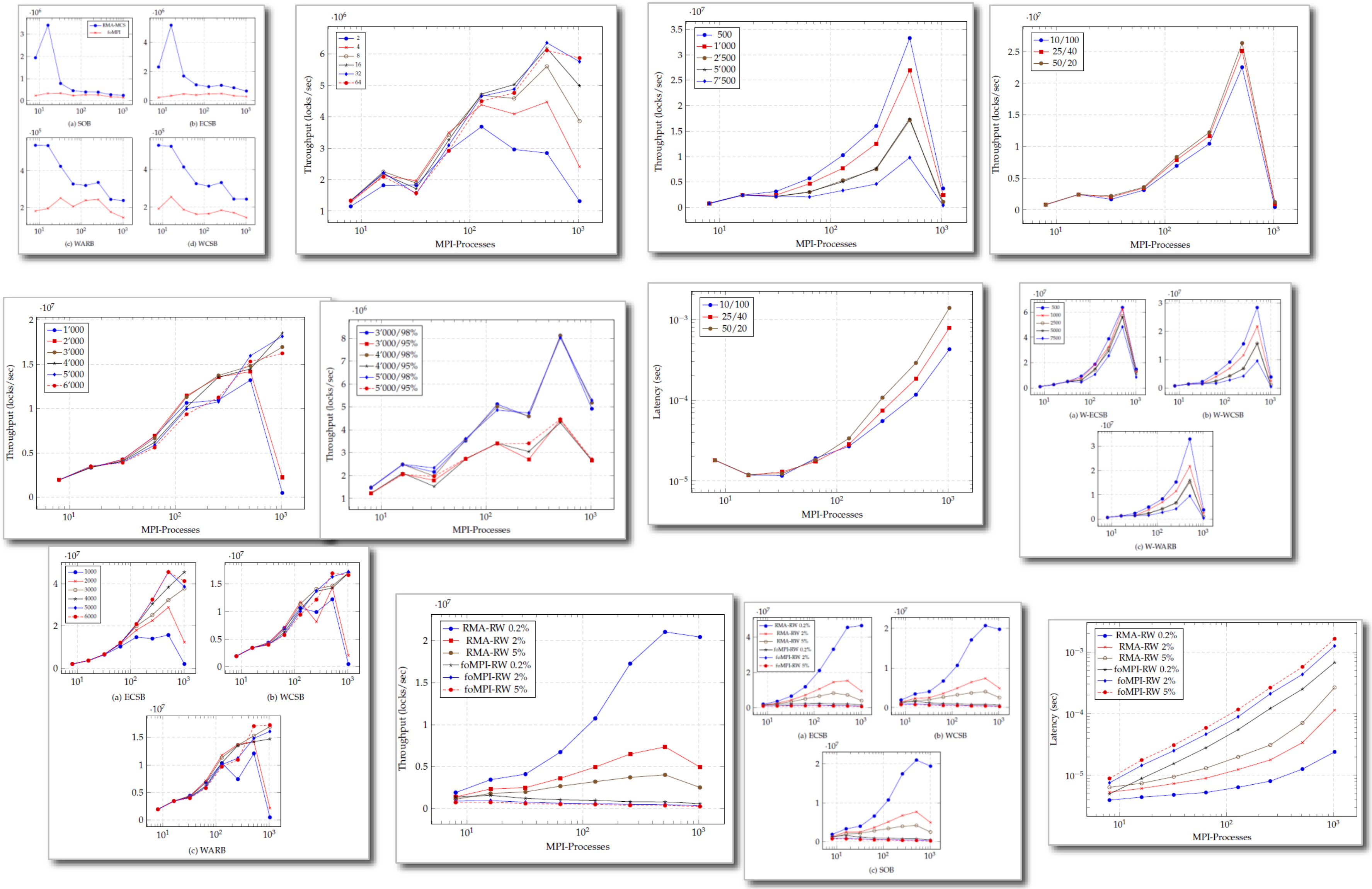


0% of writers



[1] R. Gerstenberger et al. Enabling Highly-scalable Remote Memory Access Programming with MPI-3 One Sided. ACM/IEEE Supercomputing 2013.

OTHER ANALYSES



But why stop at RDMA -- A brief history

How to program QsNet?

How to offload in Portals 4?

How to offload in libfabric?

Lossy Networks
Ethernet

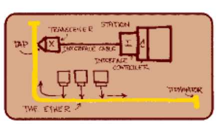
1980's

Lossless N
RDM

200

vice Programming
Offload

2020's



Mellanox TECHNOLOGIES

intel

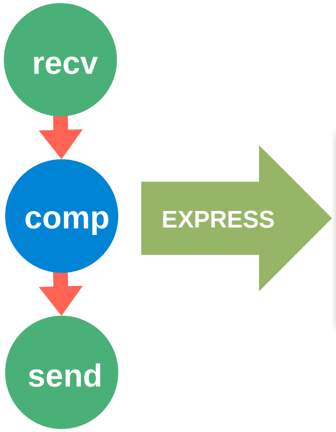
portals

OPENFABRICS ALLIANCE

Communications
(non-blocking)

Computations

Dependencies




```
L0: recv a from P1;
L1: b = compute f(buff, a);
L2: send b to P1;
L0 and CPU-> L1
L1 -> L2
```

CPU

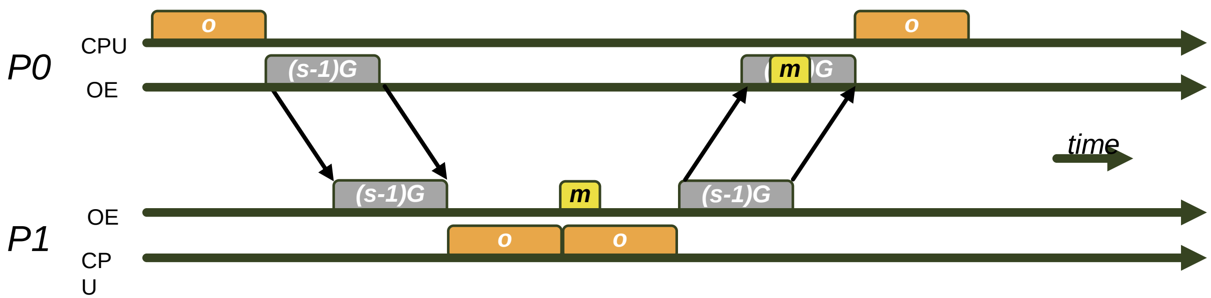
OFFLOAD

Offload Engine

portals

OPENFABRICS ALLIANCE

Performance Model




```
P0{
   L0: recv m1 from P1;
   L1: send m2 to P1;
}
```

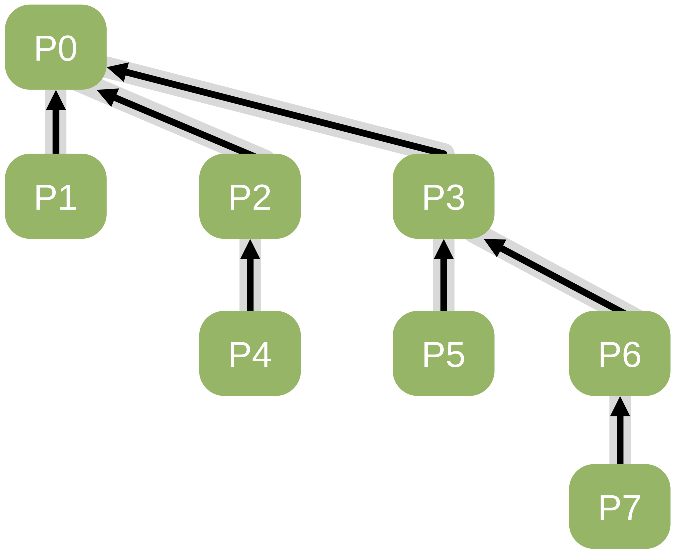
```
P1{
   L0: recv m1 from P1;
   L1: send m2 to P1;
   L0 -> L1
}
```

[1] A. Alexandrov et al. "LogGP: incorporating long messages into the LogP model—one step closer towards a realistic model for parallel computation.", Proceedings of the seventh annual ACM symposium on Parallel algorithms and architectures. ACM, 1995.

Fully Offloaded Collectives

Collective communication: A communication that involves a group of processes

Non-blocking collective: Once initiated the operation may progress independently of any computation or other communication at participating processes



Fully Offloaded Collectives

Collective communication: A communication that involves a group of processes

Non-blocking collective: Once initiated the operation may progress independently of any computation or other communication at participating processes

P0

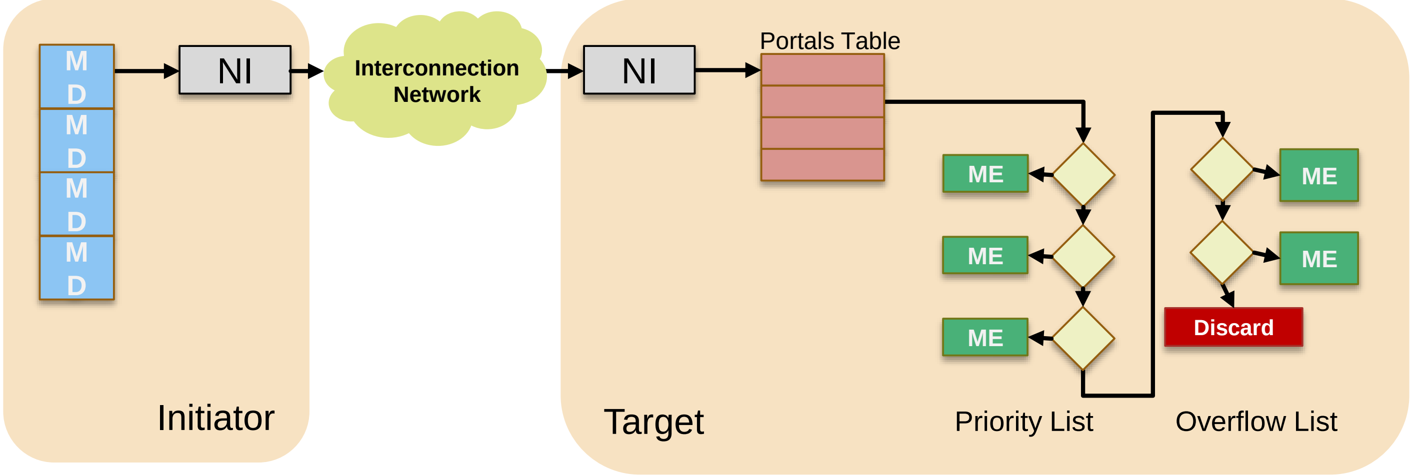
Fully Offloading:

1. *No synchronization* is required in order to start the collective operation
2. Once a collective operation is started, *no further CPU intervention* is required in order to progress or complete it.

P7

A Case Study: Portals 4

- Based on the one-sided communication model
- Matching/Non-Matching semantics can be adopted



[2] "The Portal 4.0.2 Network Programming Interface"

A Case Study: Portals 4

Communication primitives
- Put/Get operations are natively supported by Portals 4
- One-sided + matching semantic

Atomic operations
- Operands are the data specified by the MD at the initiator and by the ME at the target
- Available operators: *min, max, sum, prod, swap, and, or, …*

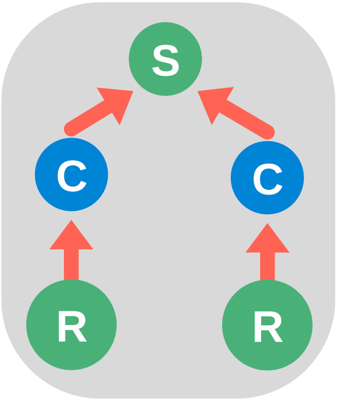
Counters
- Associated with MDs or MEs
- Count specific events (e.g., operation completion)

Triggered operations
- Put/Get/Atomic associated with a counter
- Executed when the associated counter reaches the specified threshold

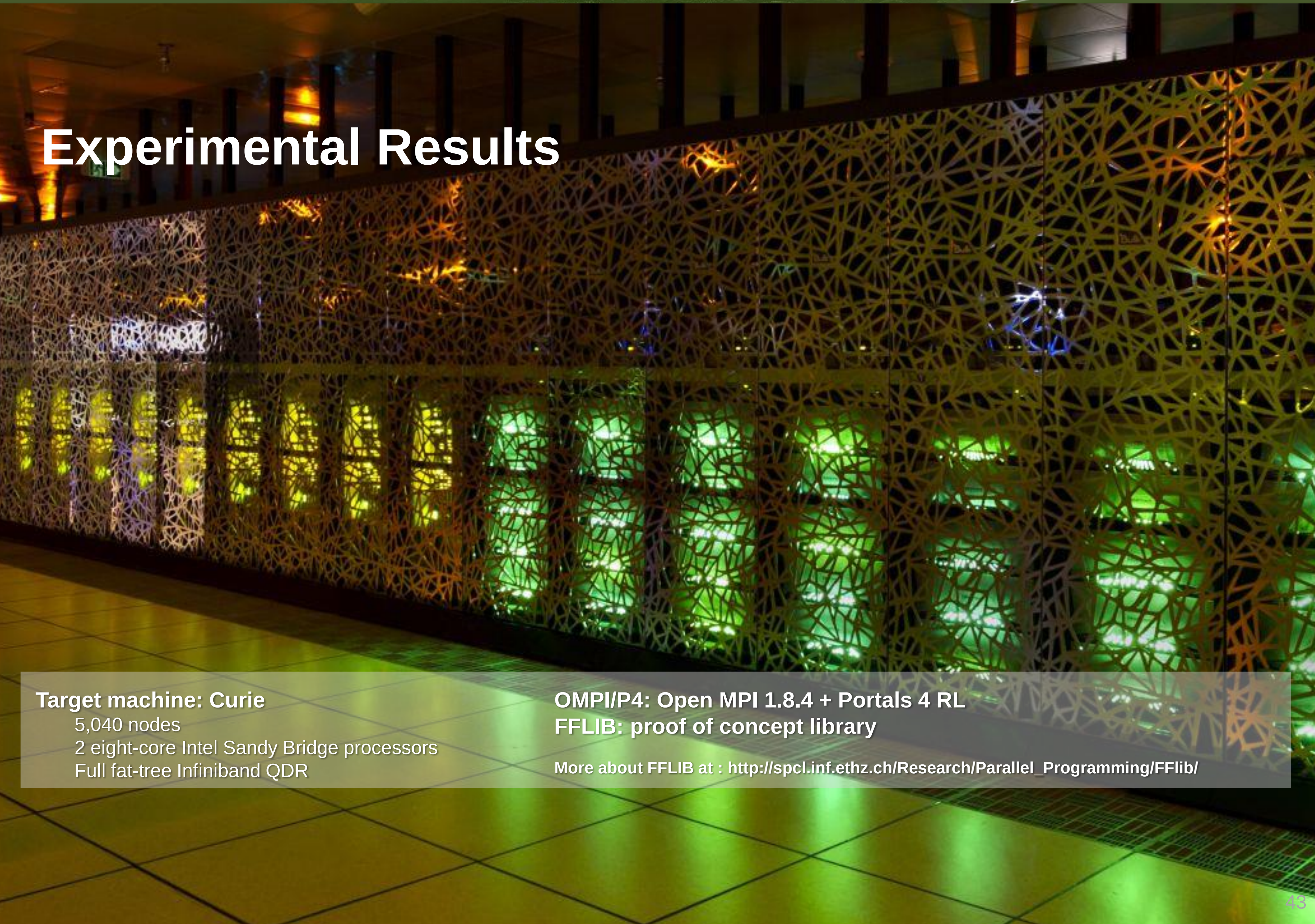
FFlib: An Example

Proof of concept library implemented on top of Portals 4

```
ff_schedule_h sched = ff_schedule_create(…);

ff_op_h r1 = ff_op_create_recv(tmp + blocksize, blocksize, child1, tag);
ff_op_h r2 = ff_op_create_recv(tmp + 2*blocksize, blocksize, child2, tag);

ff_op_h c1 = ff_op_create_computation(rbuff, blocksize, tmp + blocksize, blocksize, operator, datatype, tag)
ff_op_h c2 = ff_op_create_computation(rbuff, blocksize, tmp + 2*blocksize, blocksize, operator, datatype, tag)

ff_op_h s = ff_op_create_send(rbuff, blocksize, parent, tag)

ff_op_hb(r1, c1)
ff_op_hb(r2, c2)
ff_op_hb(c1, s)
ff_op_hb(c2, s)

ff_schedule_add(sched, r1)
ff_schedule_add(sched, r2)
ff_schedule_add(sched, c1)
ff_schedule_add(sched, c2)
ff_schedule_add(sched, s)
```



Experimental Results



Target machine: Curie

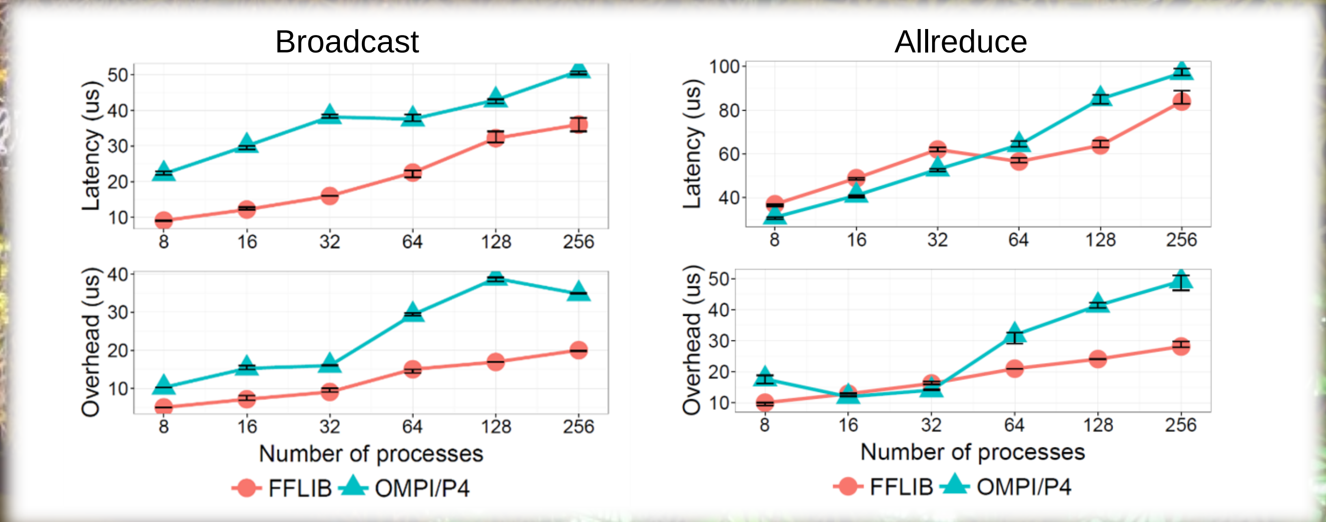
- 5,040 nodes
- 2 eight-core Intel Sandy Bridge processors
- Full fat-tree Infiniband QDR

OMPI/P4: Open MPI 1.8.4 + Portals 4 RL

FFLIB: proof of concept library

More about FFLIB at : http://spcl.inf.ethz.ch/Research/Parallel_Programming/FFlib/

Experimental Results: Latency/Overhead



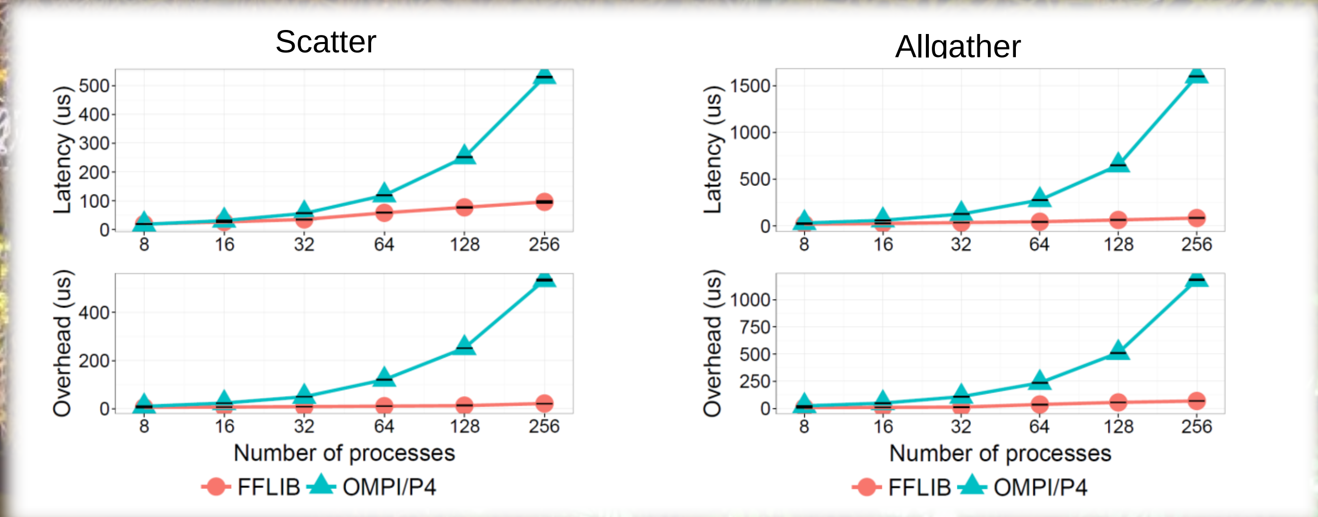
Target machine: Curie

- 5,040 nodes
- 2 eight-core Intel Sandy Bridge processors
- Full fat-tree Infiniband QDR

OMPI/P4: Open MPI 1.8.4 + Portals 4 RL
FFLIB: proof of concept library

More about FFLIB at : http://spcl.inf.ethz.ch/Research/Parallel_Programming/FFlib/

Experimental Results: Latency/Overhead



Target machine: Curie
- 5,040 nodes
- 2 eight-core Intel Sandy Bridge processors
- Full fat-tree Infiniband QDR

OMPI/P4: Open MPI 1.8.4 + Portals 4 RL
FFLIB: proof of concept library

More about FFLIB at : http://spcl.inf.ethz.ch/Research/Parallel_Programming/FFlib/

Experimental Results: Micro-Benchmarks

3DFFT

PGMRES

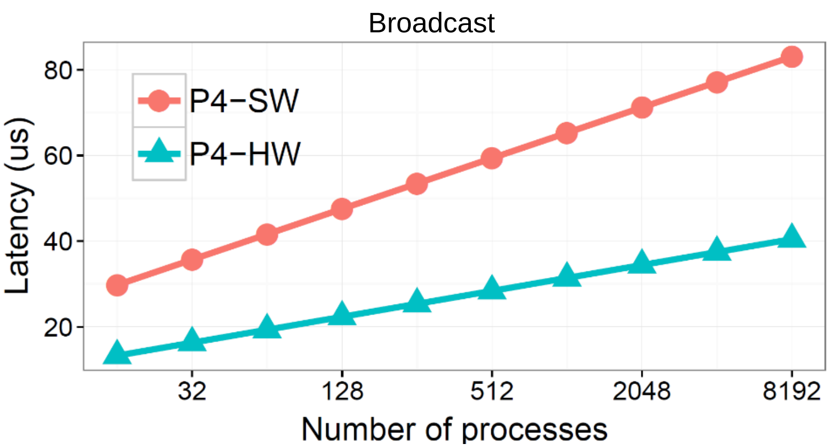
Target machine: Curie
- 5,040 nodes
- 2 eight-core Intel Sandy Bridge processors
- Full fat-tree Infiniband QDR

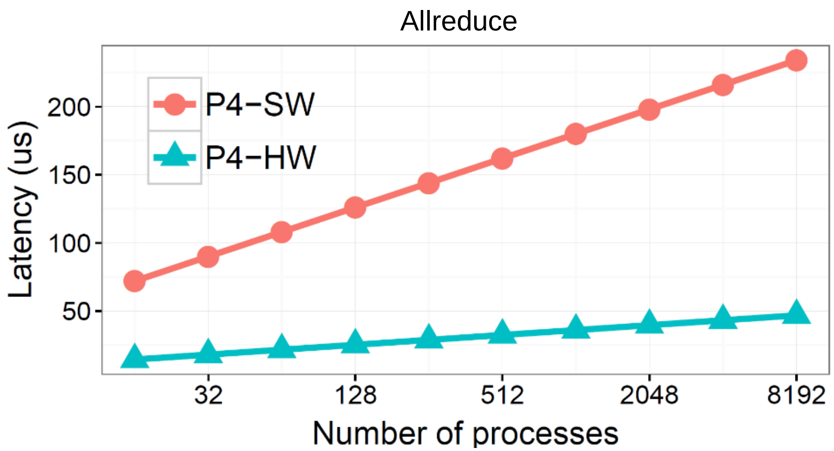
OMPI/P4: Open MPI 1.8.4 + Portals 4 RL
FFLIB: proof of concept library

More about FFLIB at : http://spcl.inf.ethz.ch/Research/Parallel_Programming/FFlib/

Simulations

- **Why?** To study offloaded collectives at large scale
- **How?** Extending the LogGOPSim to simulate Portals 4 functionalities

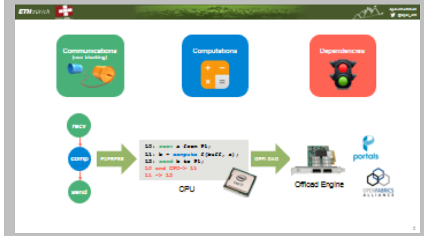


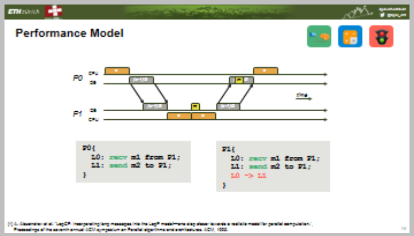


	L	o	g	G	m
P4-SW	$5\mu s$	$6\mu s$	$6\mu s$	$0.4ns$	$0.9ns$
P4-HW	$2.7\mu s$	$1.2\mu s$	$0.5\mu s$	$0.4ns$	$0.3ns$ [4]

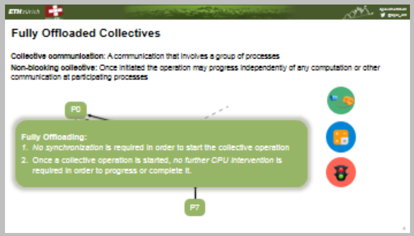
[3] T. Hoefler, T. Schneider, A. Lumsdaine. “LogGOPSim - Simulating Large-Scale Applications in the LogGOPS Model”, In *Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing* (HPDC '10). ACM, 2010.
[4] Underwood et al., "Enabling Flexible Collective Communication Offload with Triggered Operations“, *IEEE 19th Annual Symposium on High Performance Interconnects* (HOTI ‘11). IEEE, 2011.

Abstract Machine Model

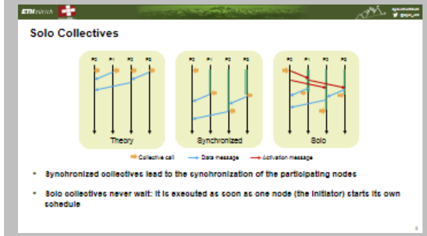


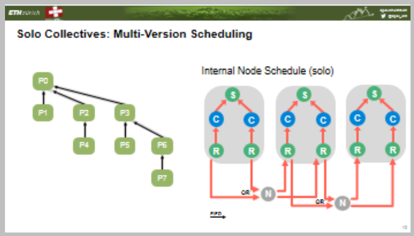


Offloading Collectives

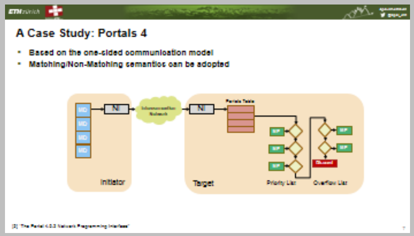


Solo Collectives

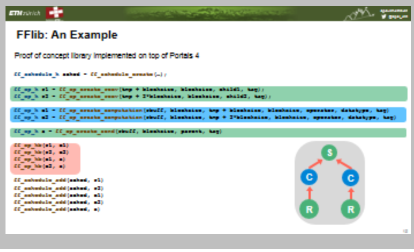




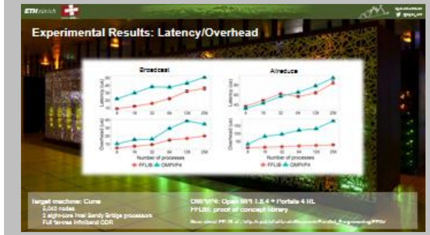
Mapping to Portals 4

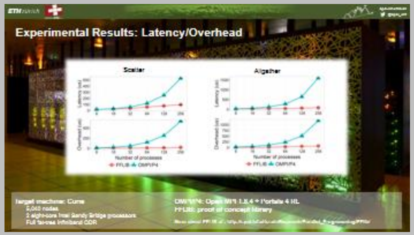


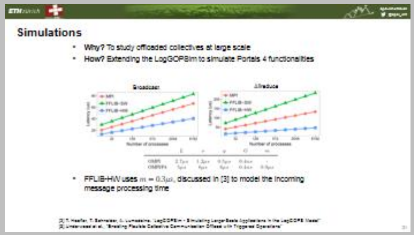
FFlib



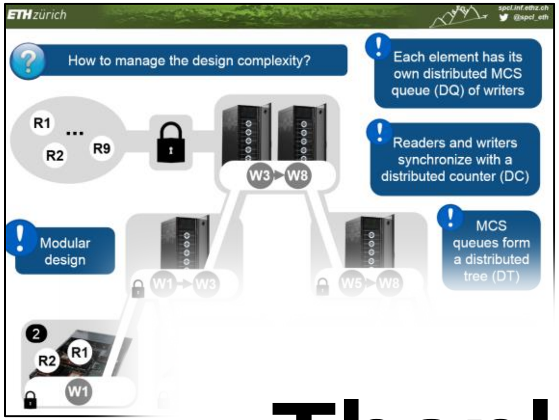
Results



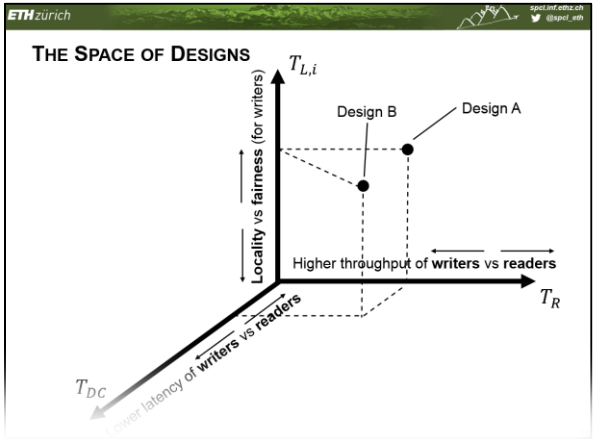




Conclusions

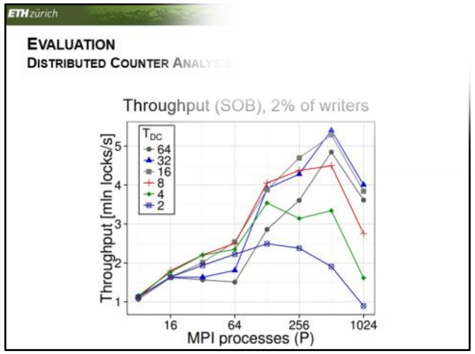


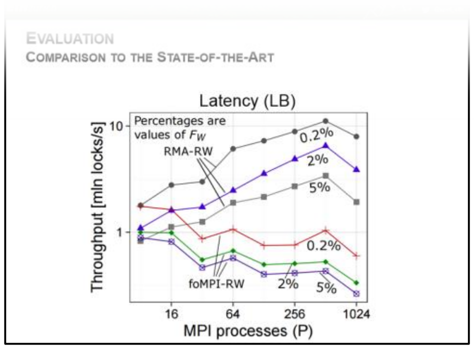
Modular d
corre



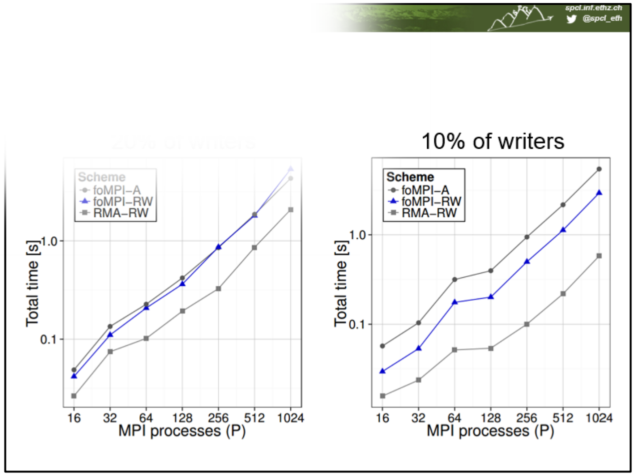
ible
es

Thank you for your attention





Improves latency and throughput over state-of-the-art



Enables high-performance distributed hashtabled