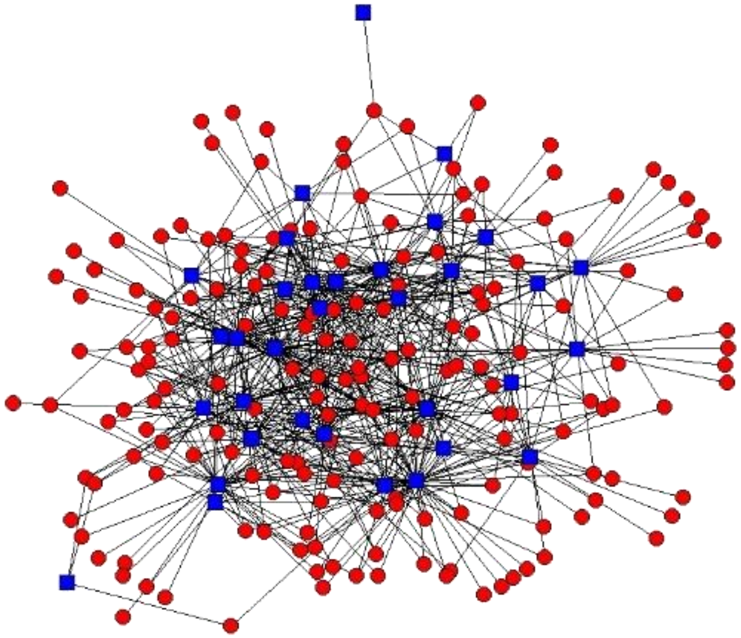
# High-Performance Distributed RMA Locks

PATRICK SCHMID, MACIEJ BESTA, TORSTEN HOEFLER

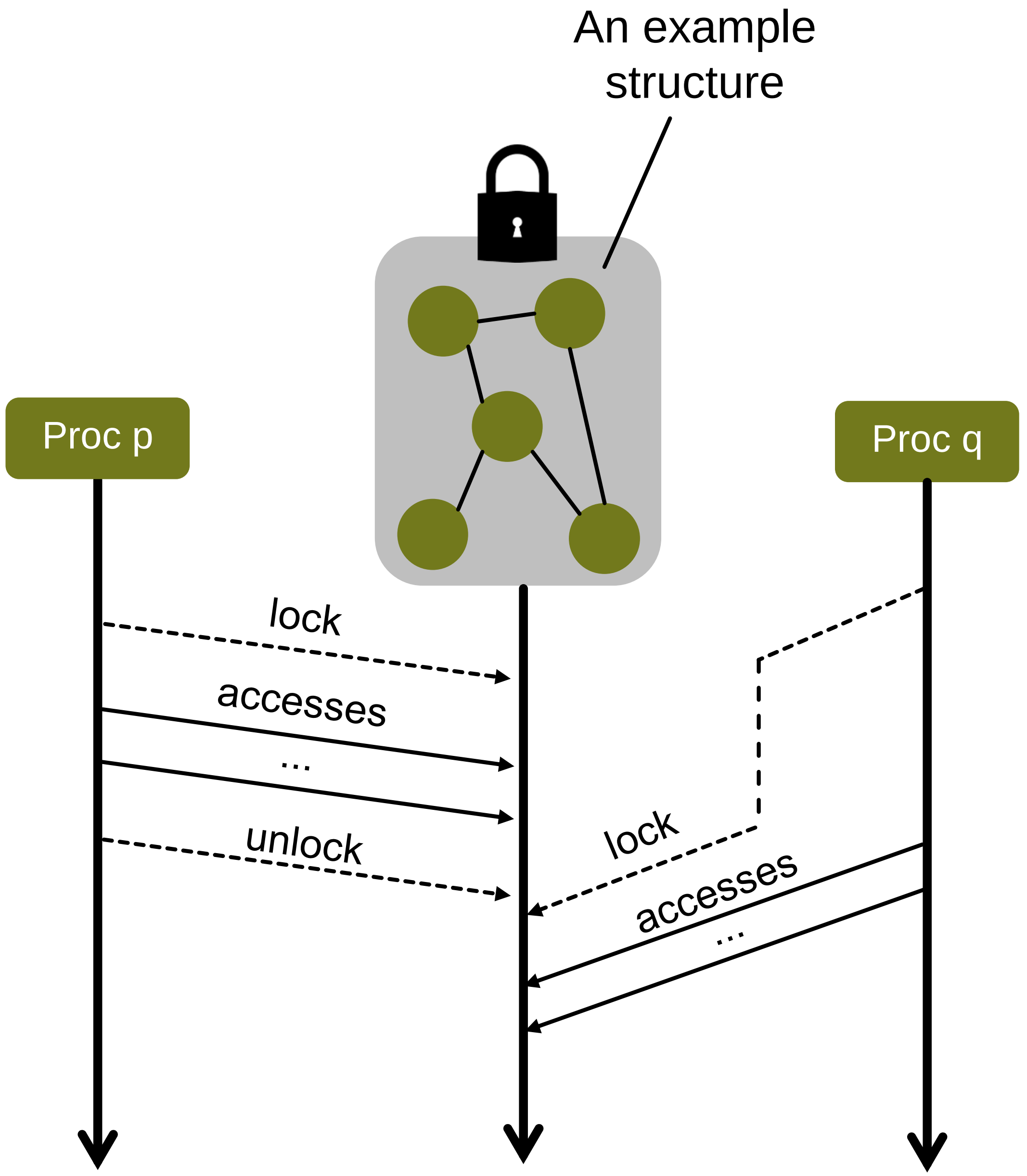
## Need for Efficient large-Scale Synchronization



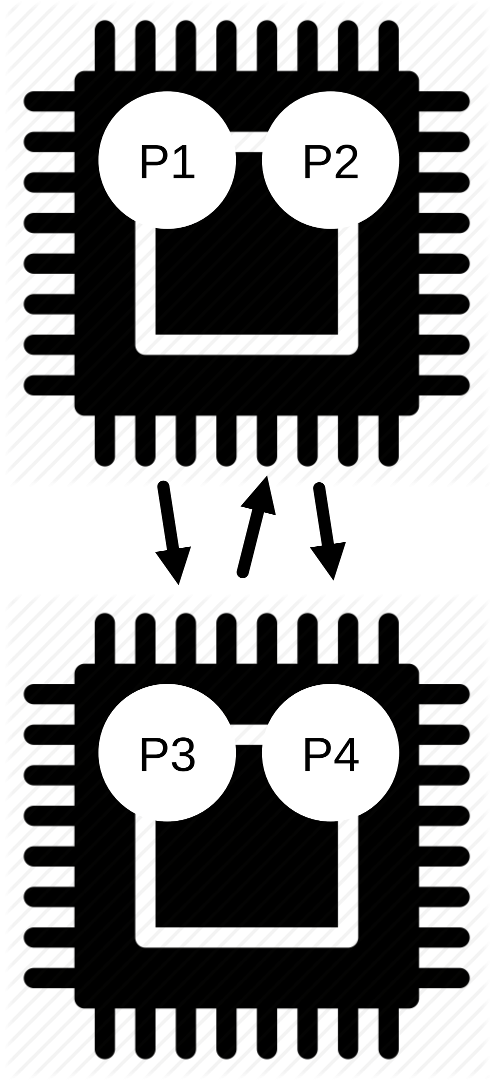
## Locks

Inuitive semantics

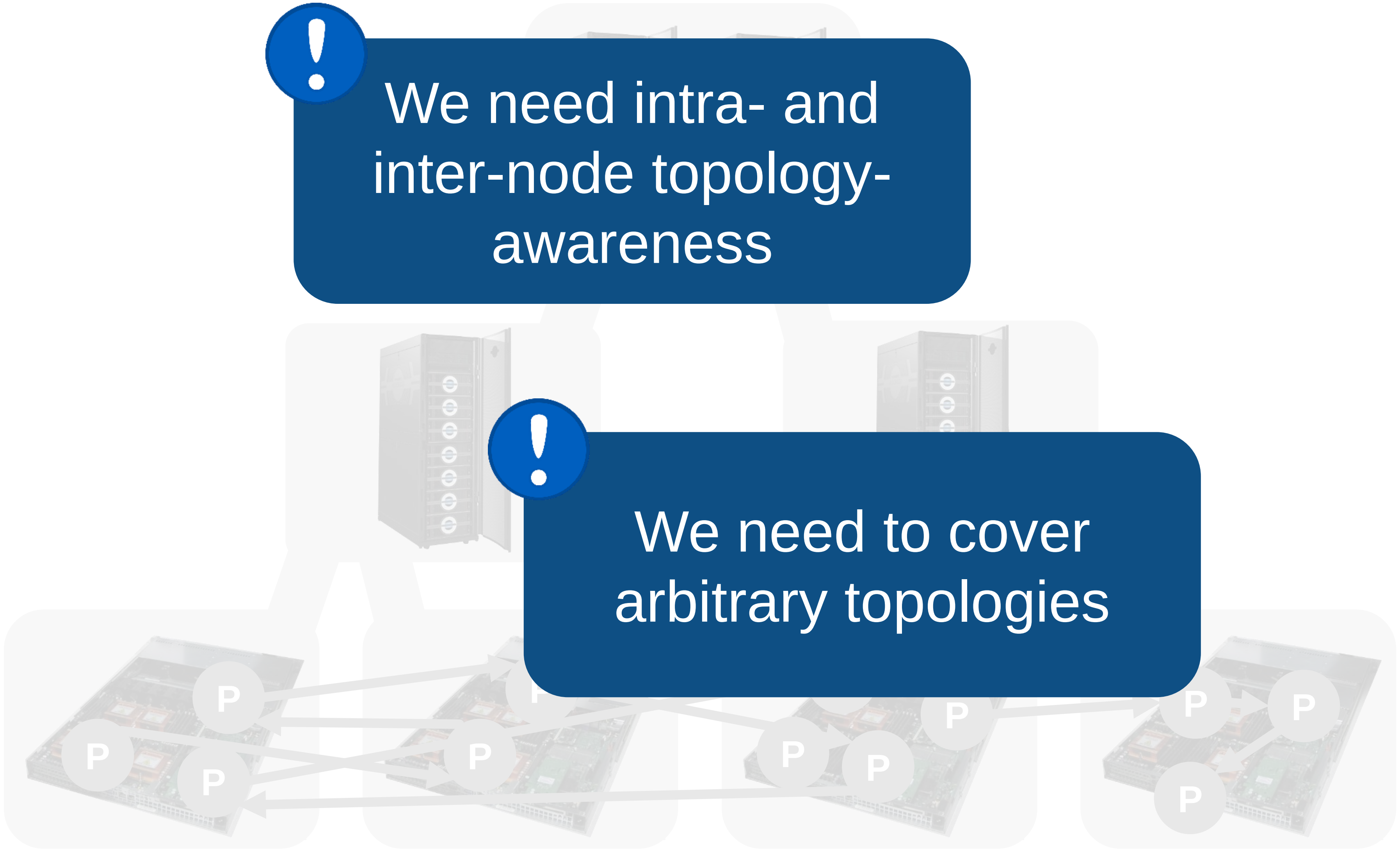
Various performance penalties



## Locks: Challenges



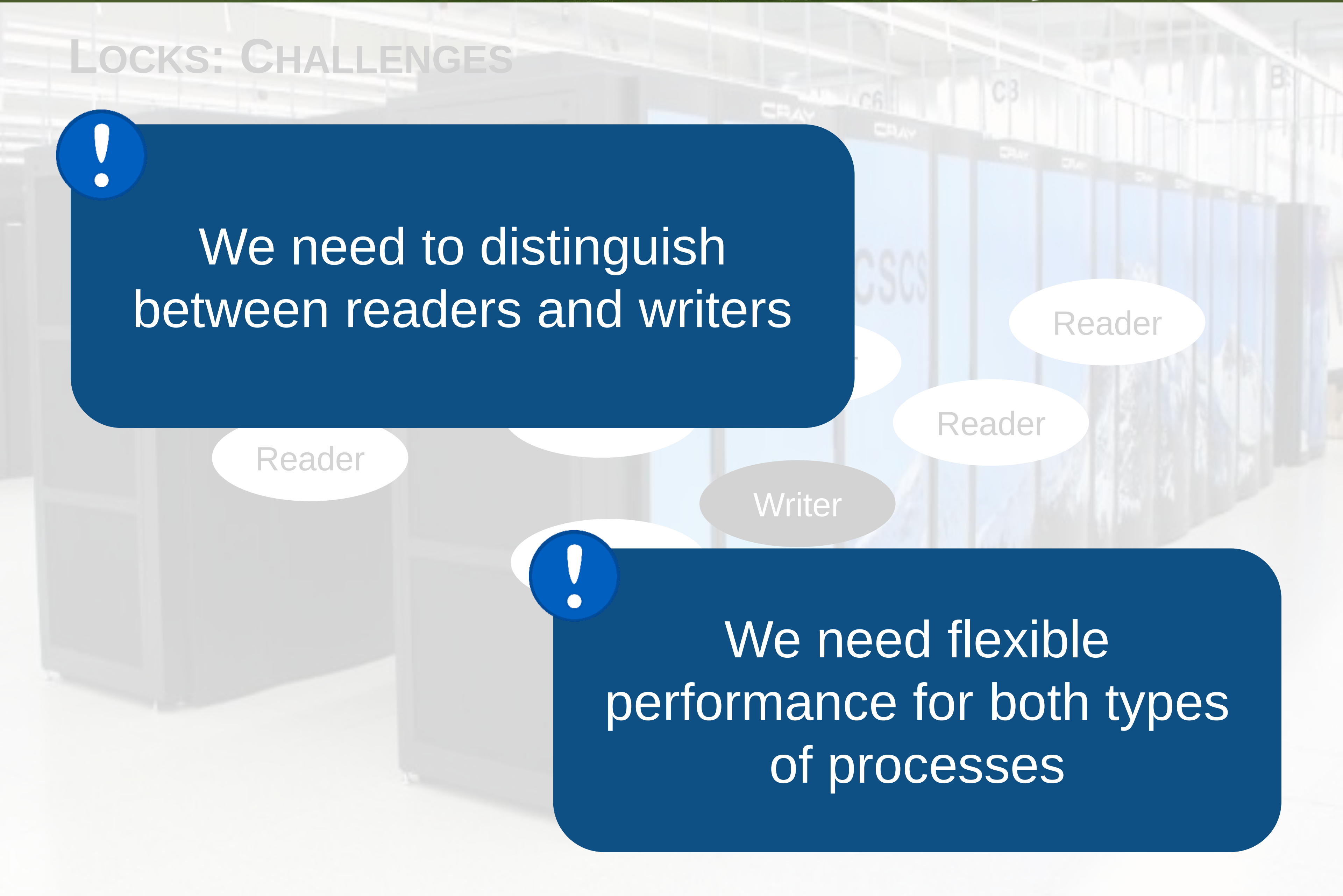
## LOCKS: CHALLENGES



We need intra- and inter-node topology-awareness

We need to cover arbitrary topologies

## LOCKS: CHALLENGES



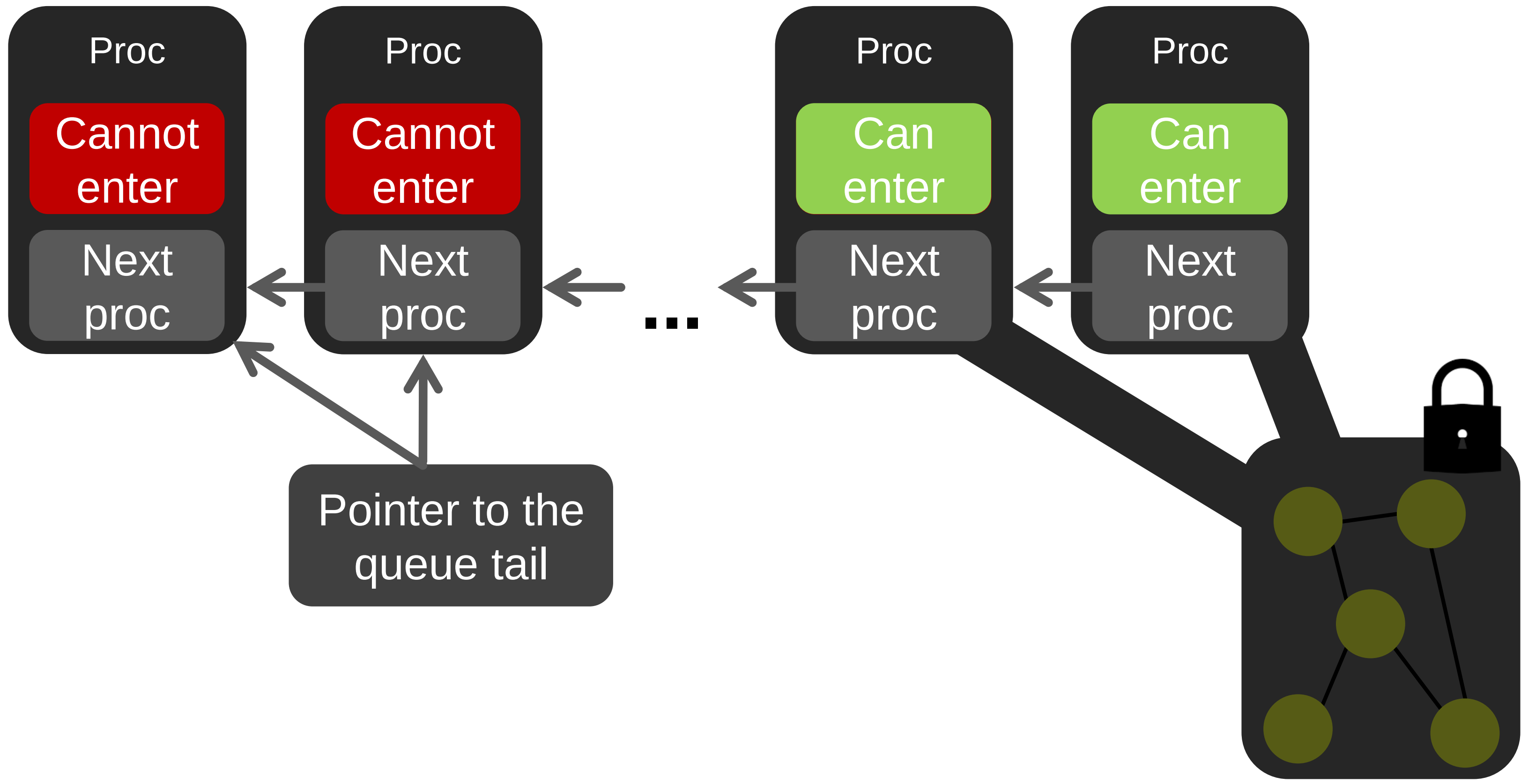
We need to distinguish between readers and writers

We need flexible performance for both types of processes

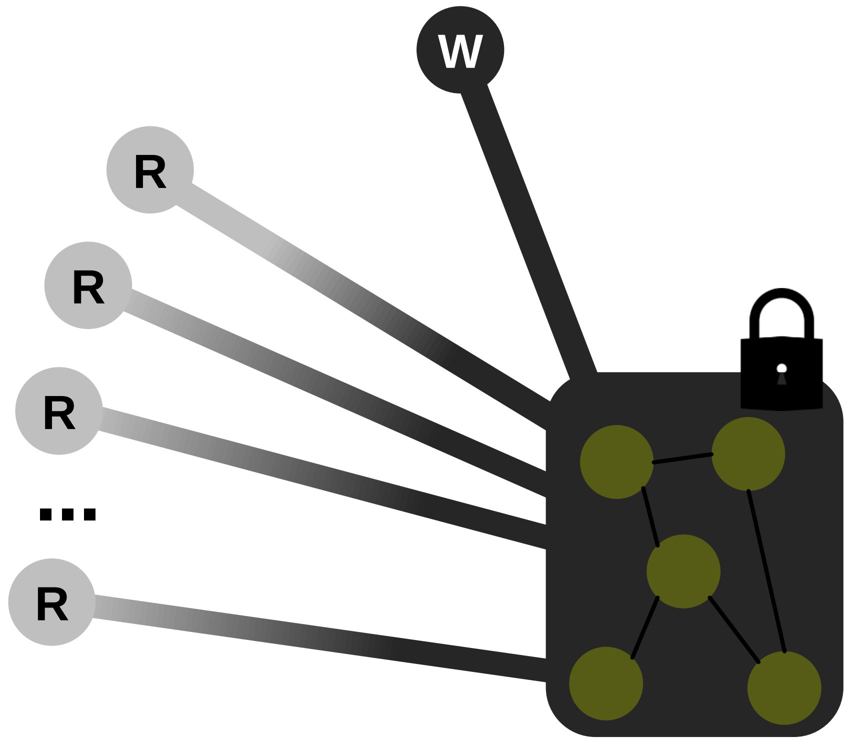
[1] V. Venkataramani et al. Tao: How facebook serves the social graph. SIGMOD'12.

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?

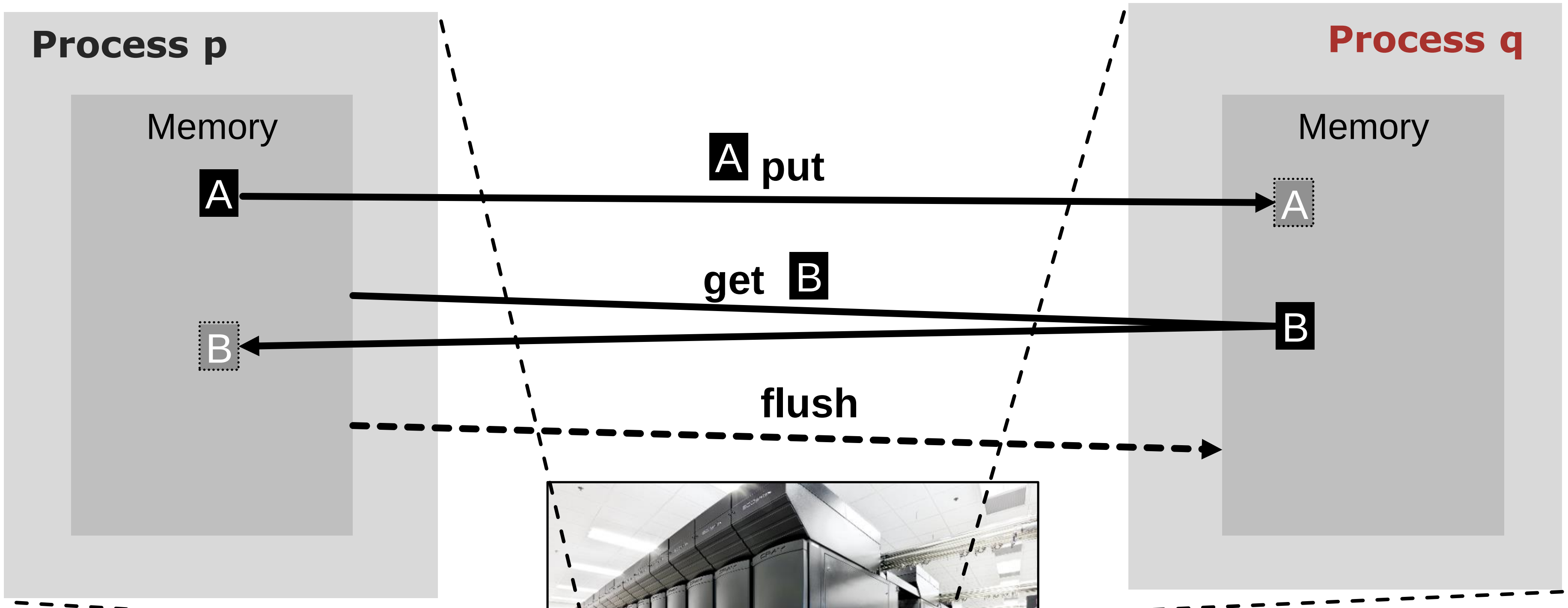
?

How to ensure tunable performance?

What mechanism to use for efficient implementation?

?

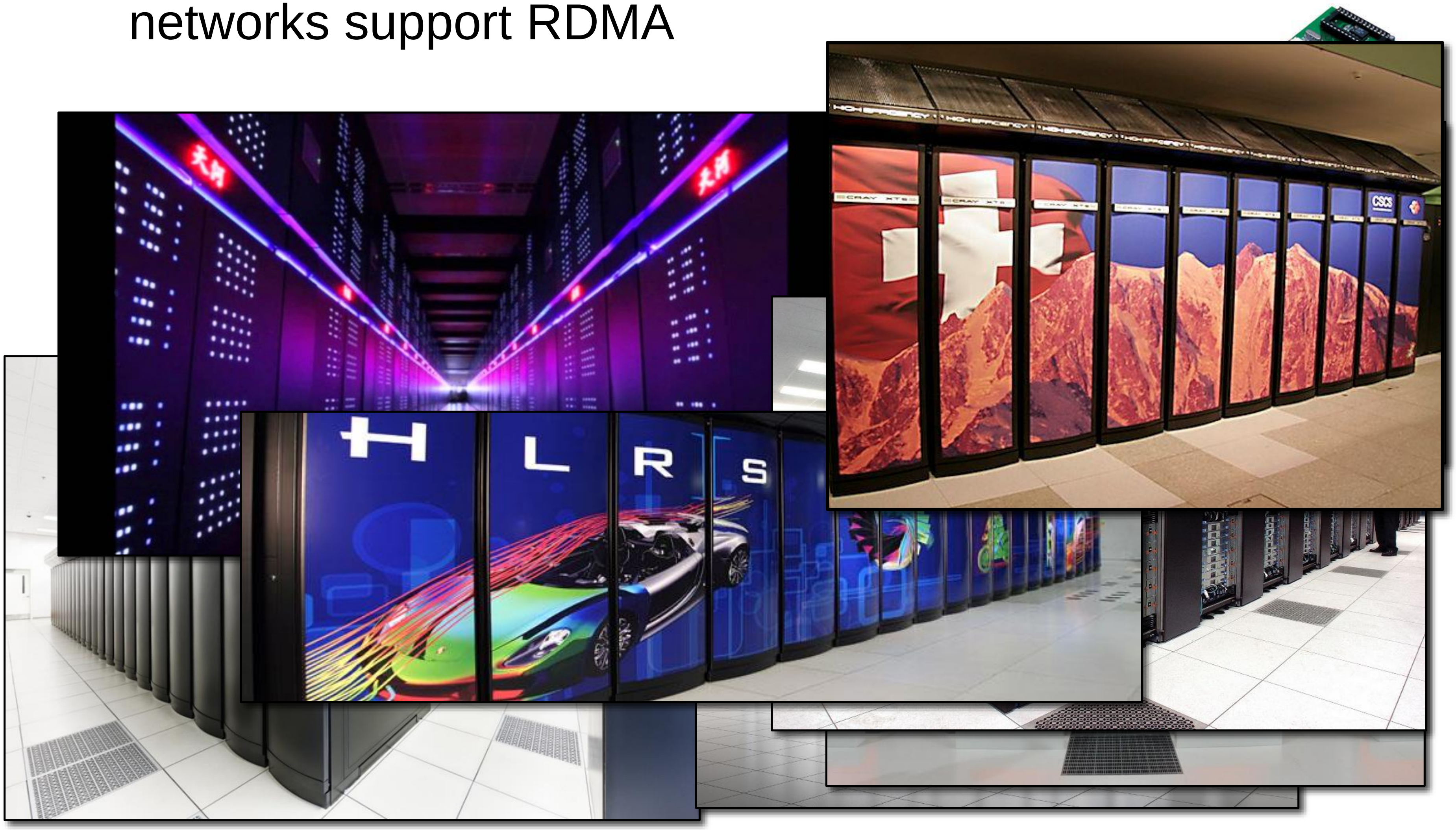
## Remote Memory Access (RMA) Programming



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?

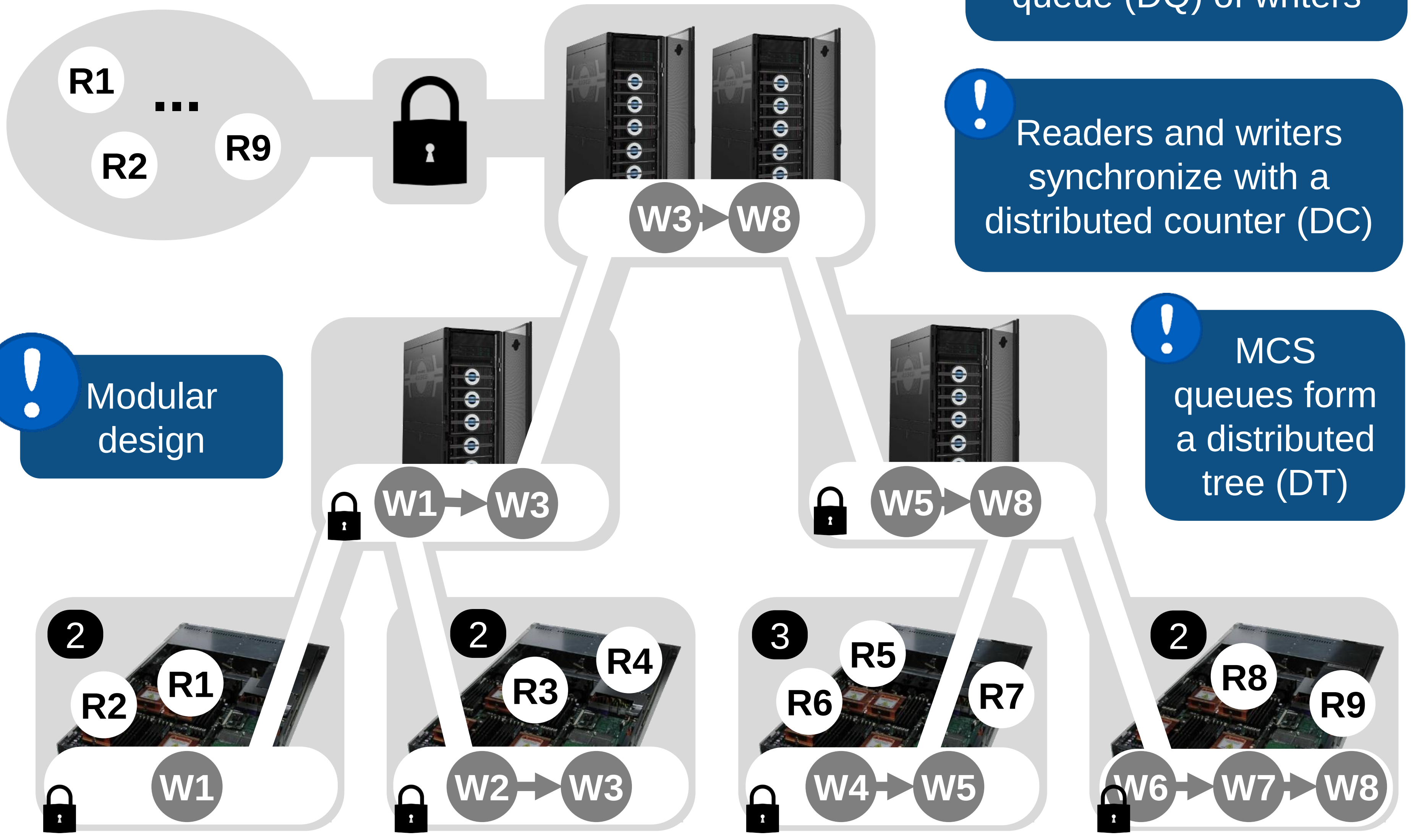
## How to manage the design complexity?

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



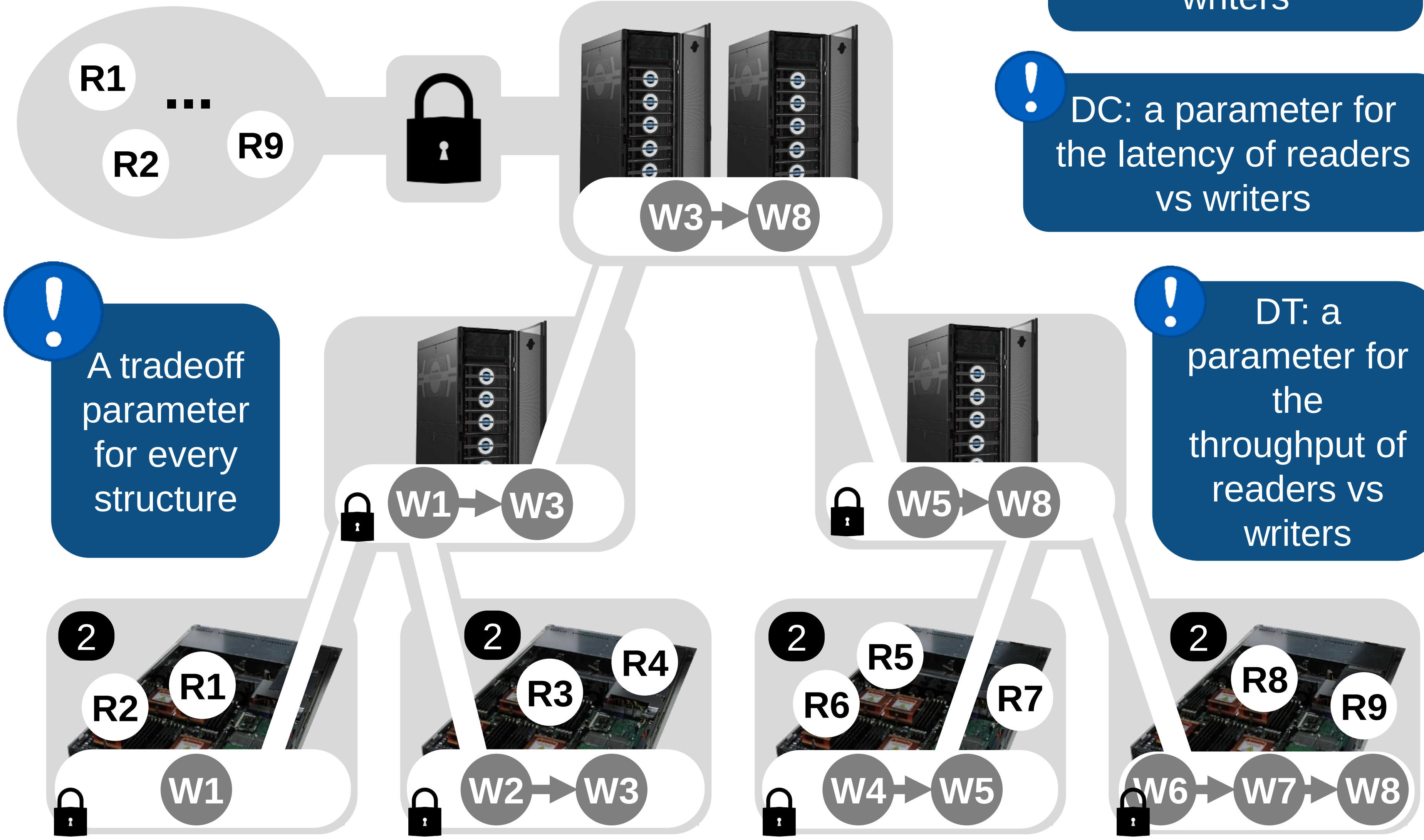
## How to ensure tunable performance?

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

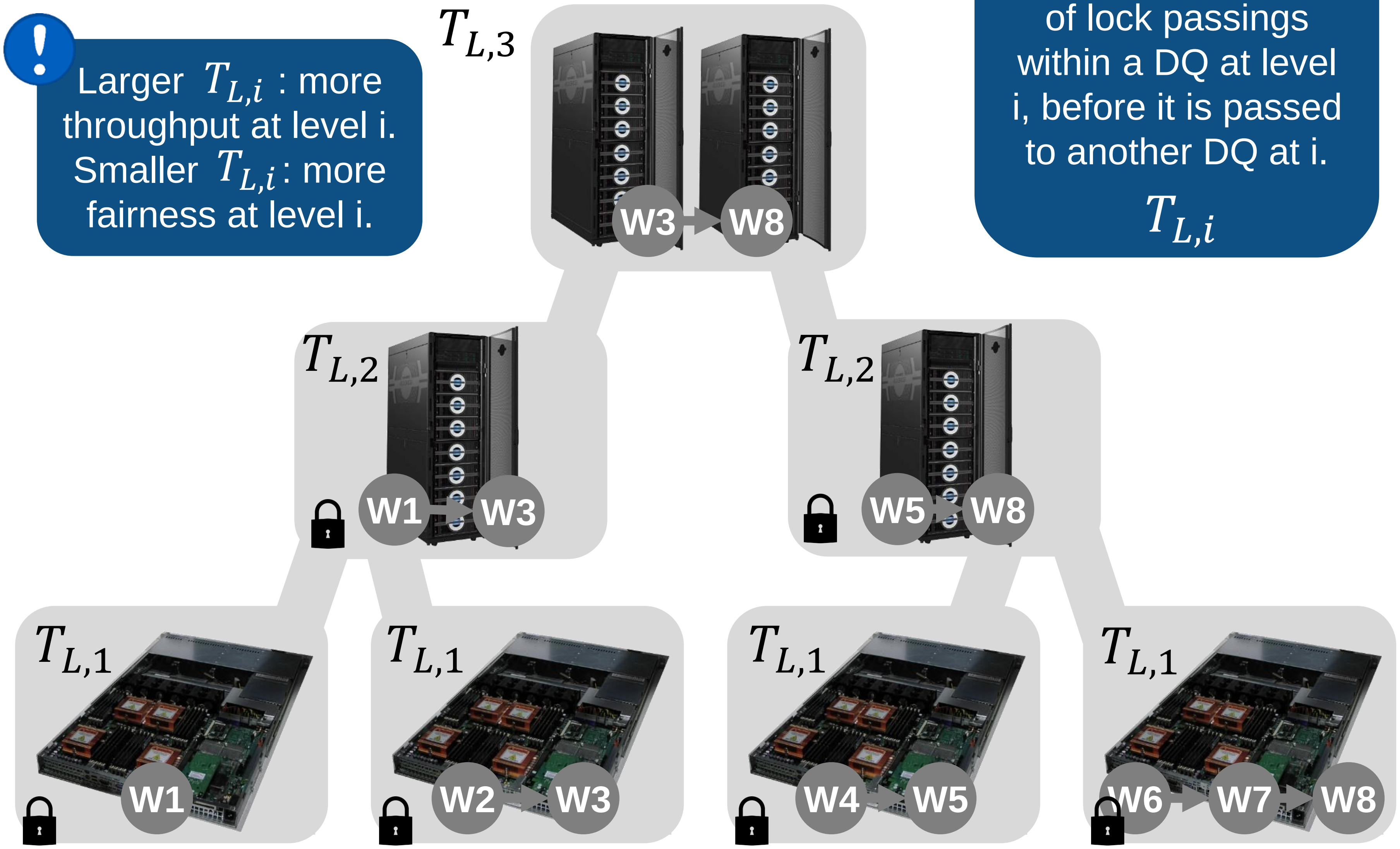


# Distributed MCS Queues (DQs)

## Throughput vs Fairness

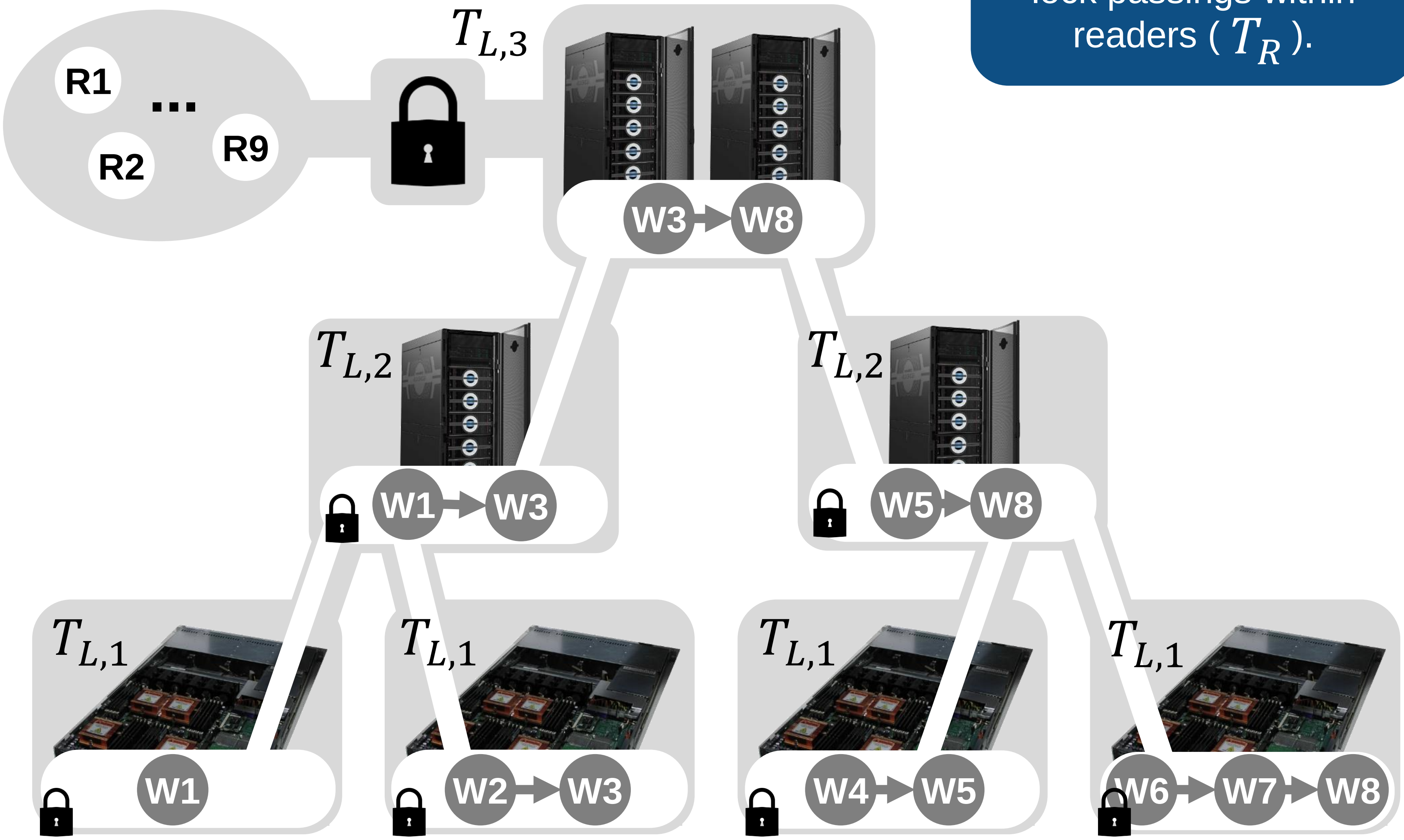
!
Larger $T_{L,i}$ : more throughput at level i. Smaller $T_{L,i}$ : more fairness at level i.

!
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

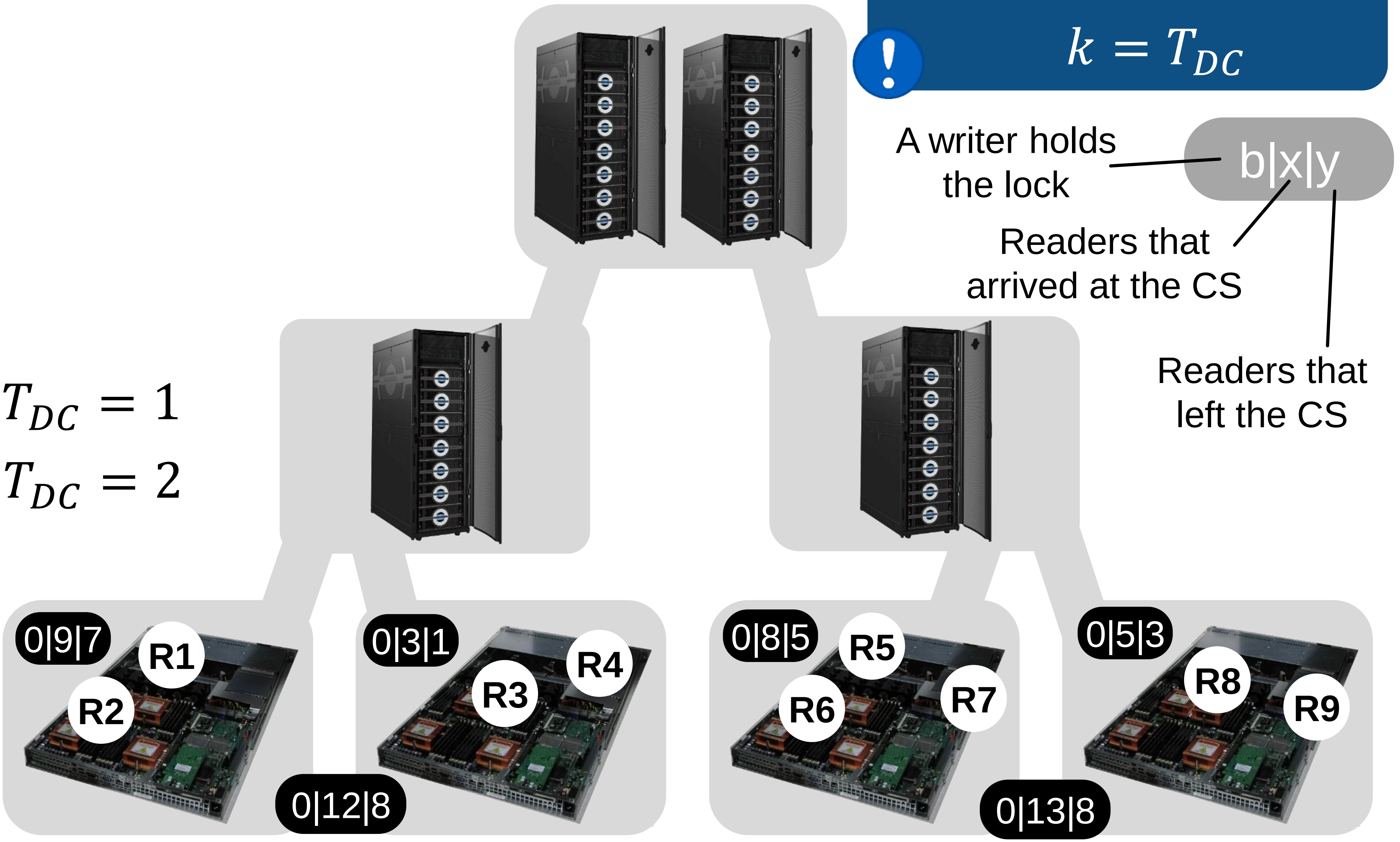
DT: The maximum number of consecutive lock passings within readers ( $T_R$ ).



# Distributed Counter (DC)
## Latency of readers vs writers

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

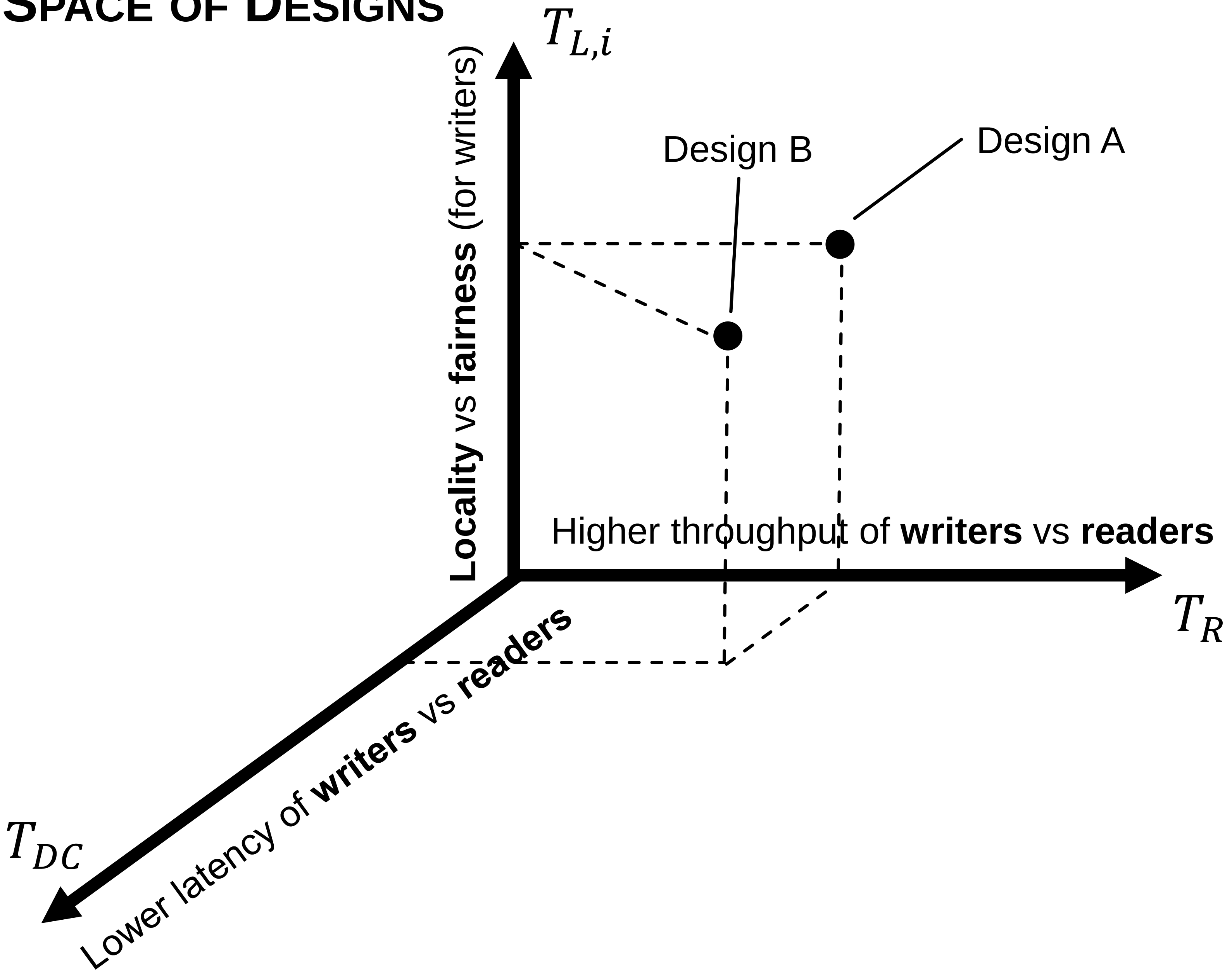
$$k = T_{DC}$$



$T_{DC} = 1$

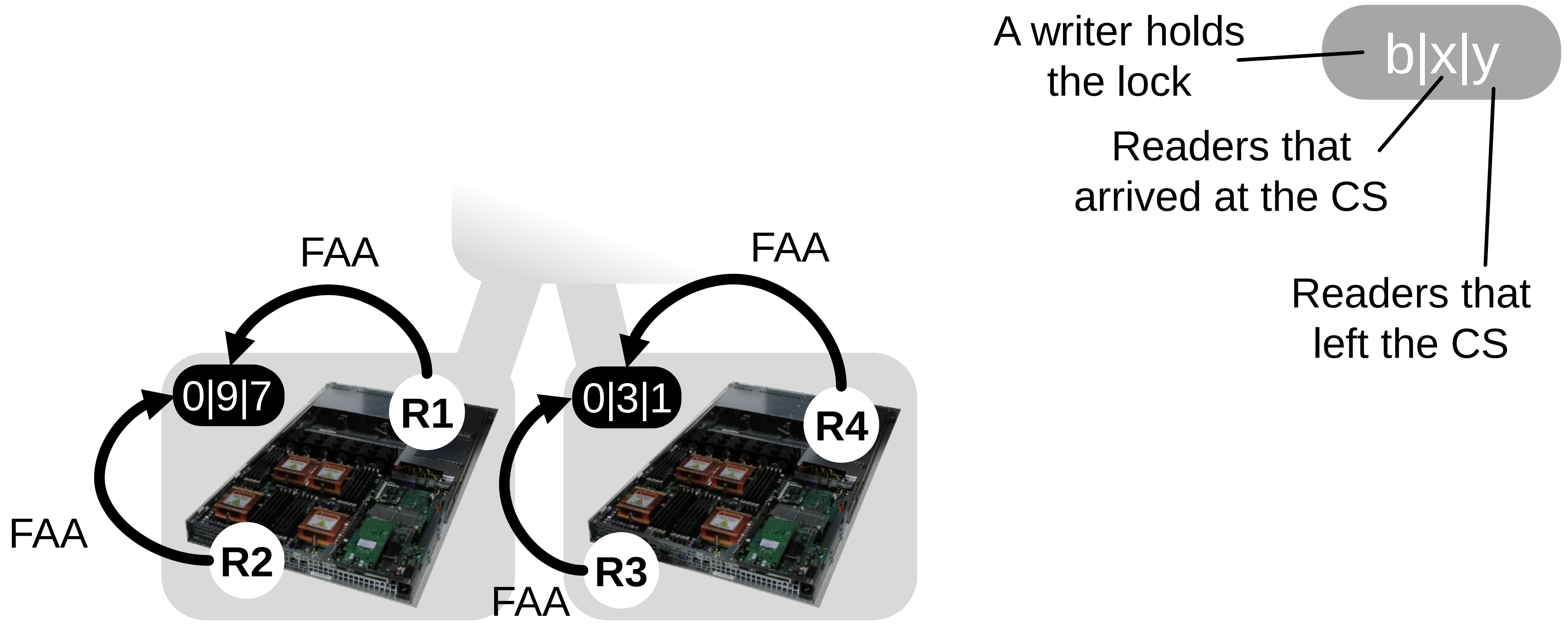
$T_{DC} = 2$

## The Space of Designs

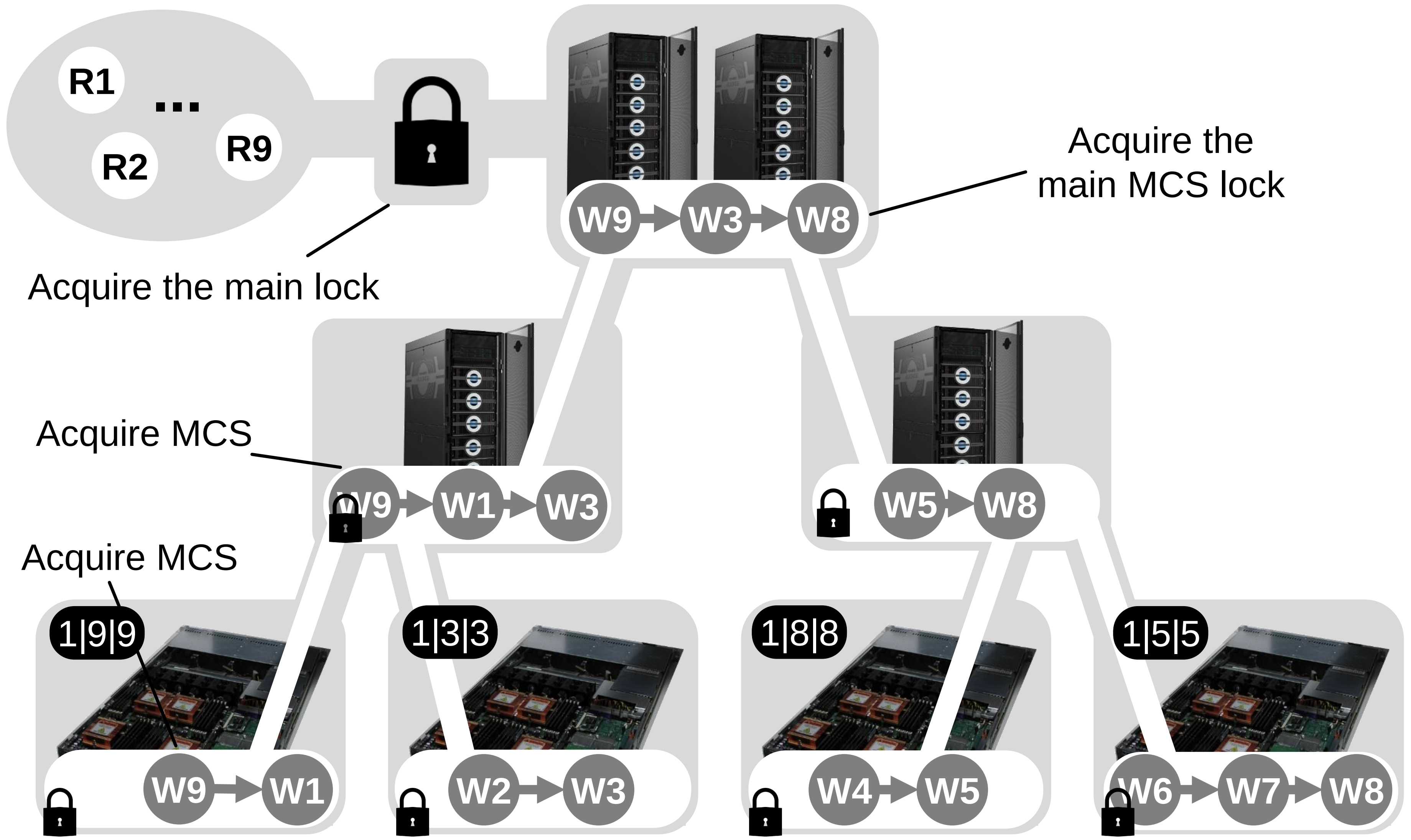

$T_{L,i}$
**Locality** vs **fairness** (for writers)
Design B
Design A
Higher throughput of **writers** vs **readers**
$T_R$
Lower latency of **writers** vs **readers**
$T_{DC}$

## 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

DHT

Distributed hashtable evaluation

**Throughput benchmarks:**

Empty-critical-section

Single-operation

Wait-after-release

Workload-critical-section

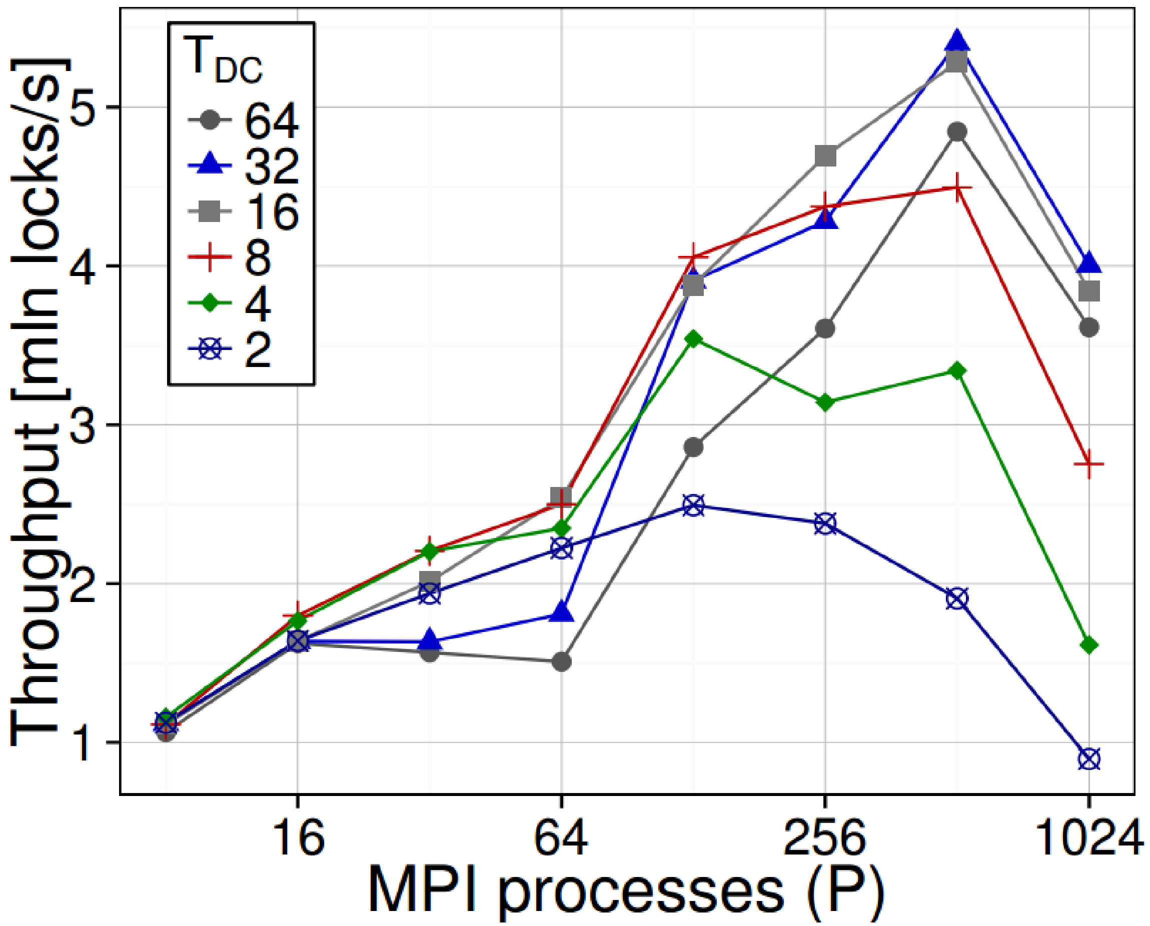
## Evaluation
### Distributed Counter Analysis

0|9|7 0|3|1

0|12|8

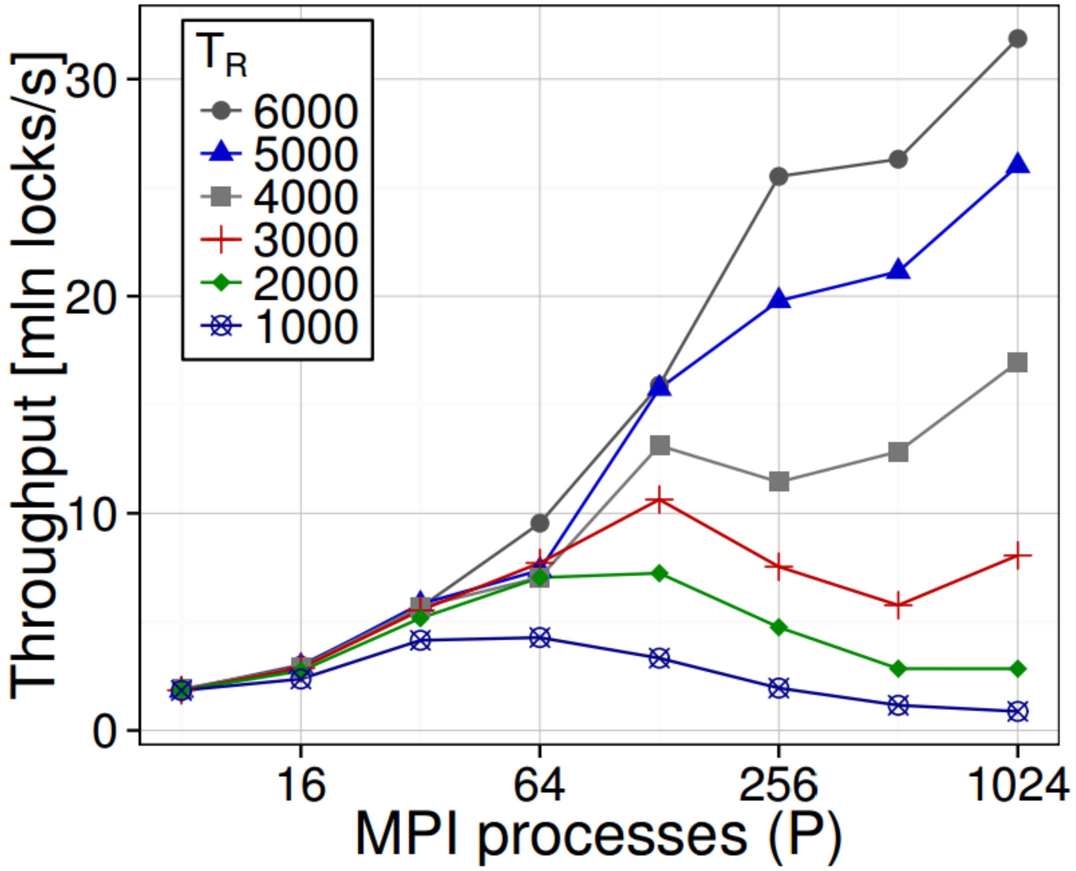
Throughput, 2% writers

Single-operation benchmark

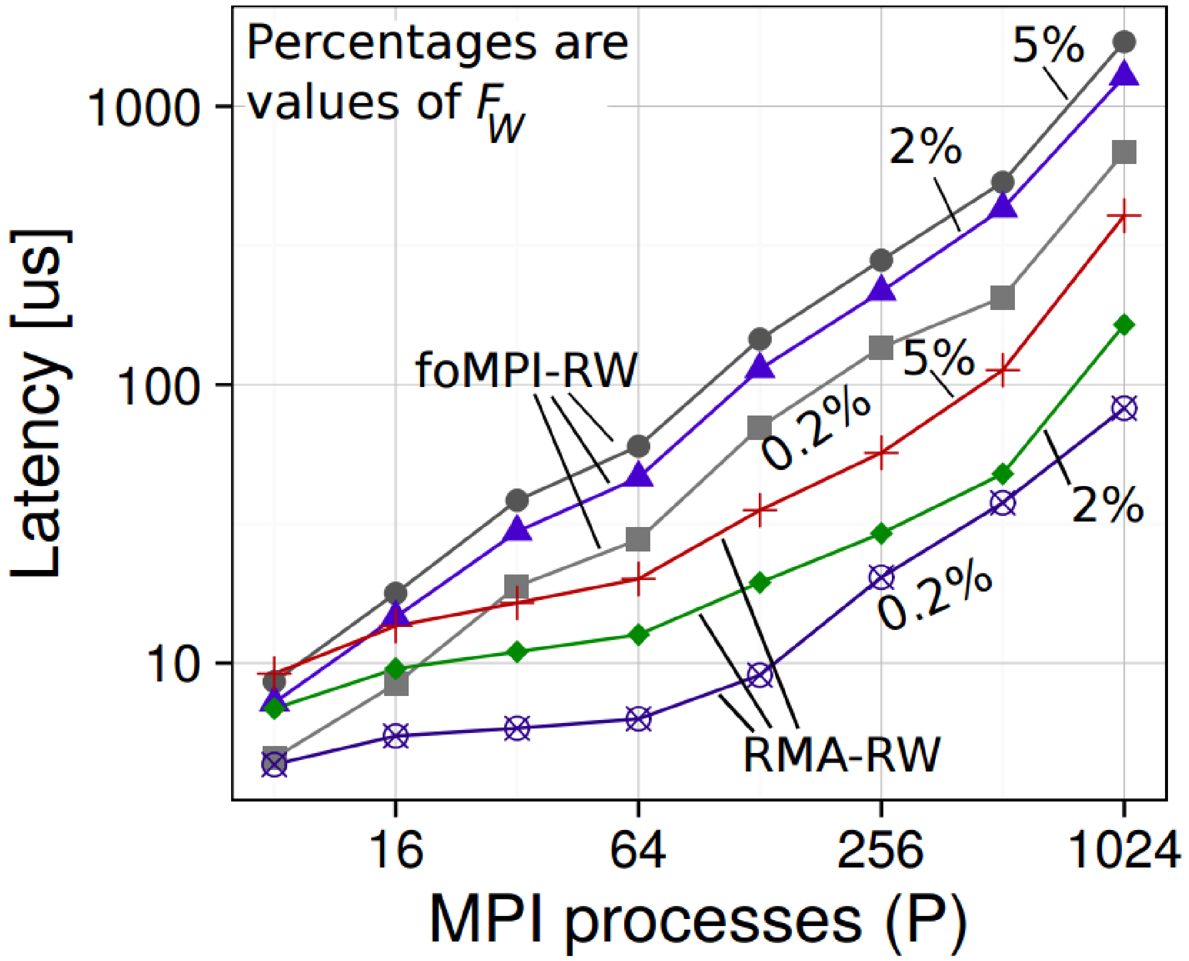


## Evaluation
### Reader Threshold Analysis

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



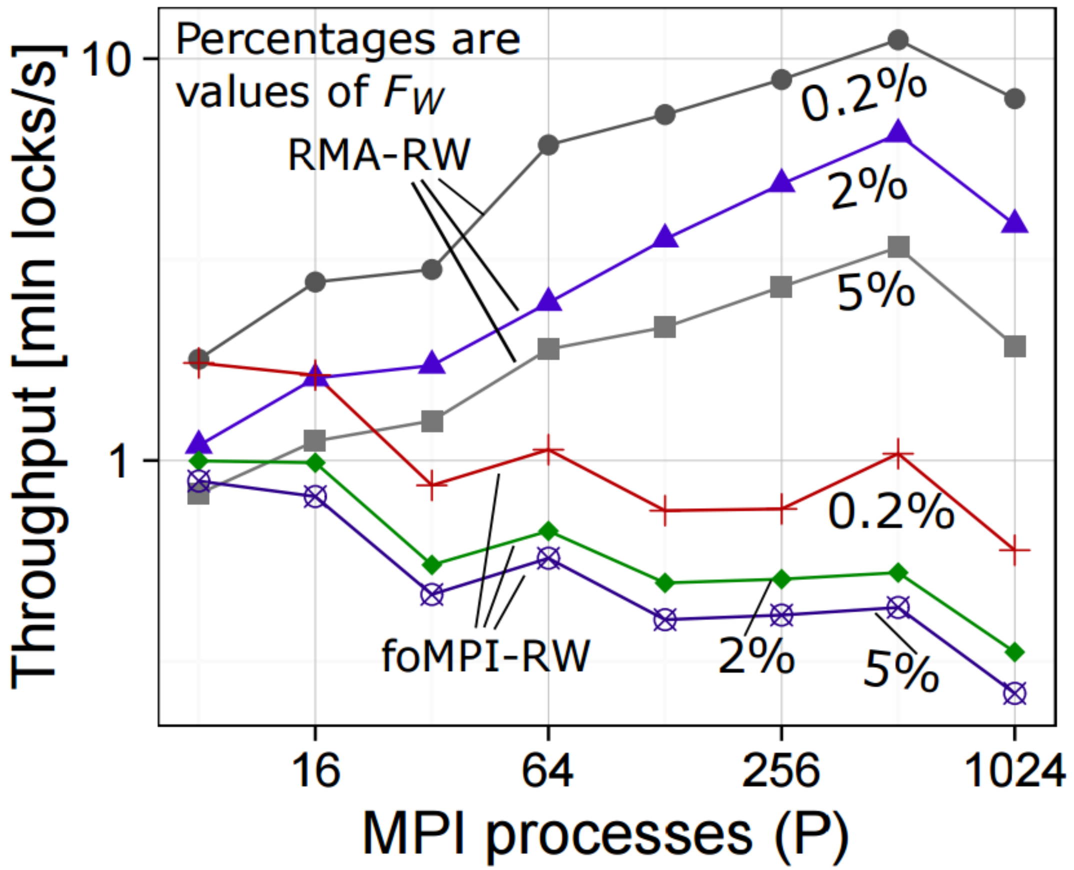
## Evaluation
### Comparison to the State-of-the-Art



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

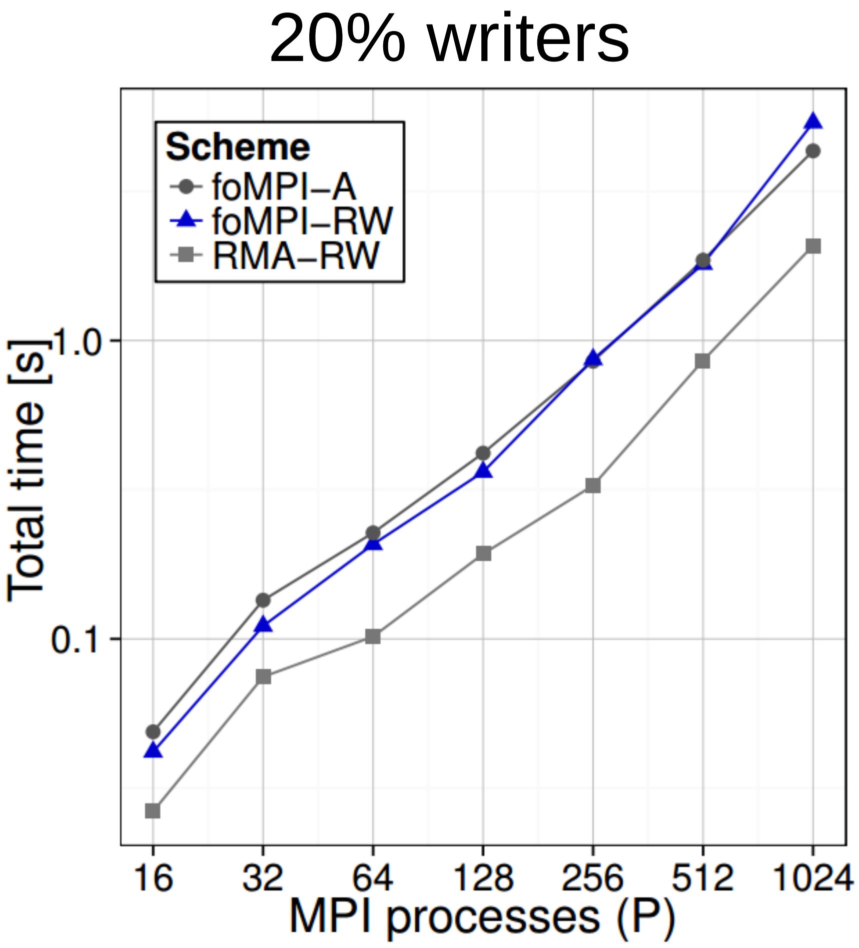
## EVALUATION
### COMPARISON TO THE STATE-OF-THE-ART

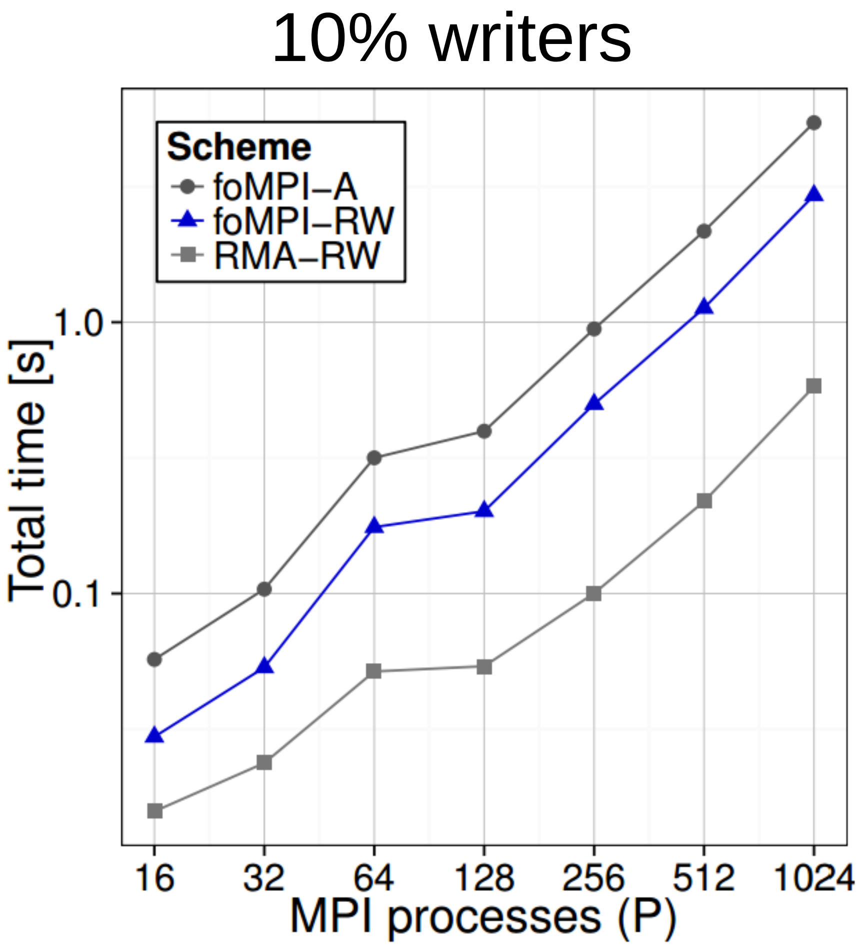
Throughput, single-operation benchmark



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

## EVALUATION
### DISTRIBUTED HASHTABLE

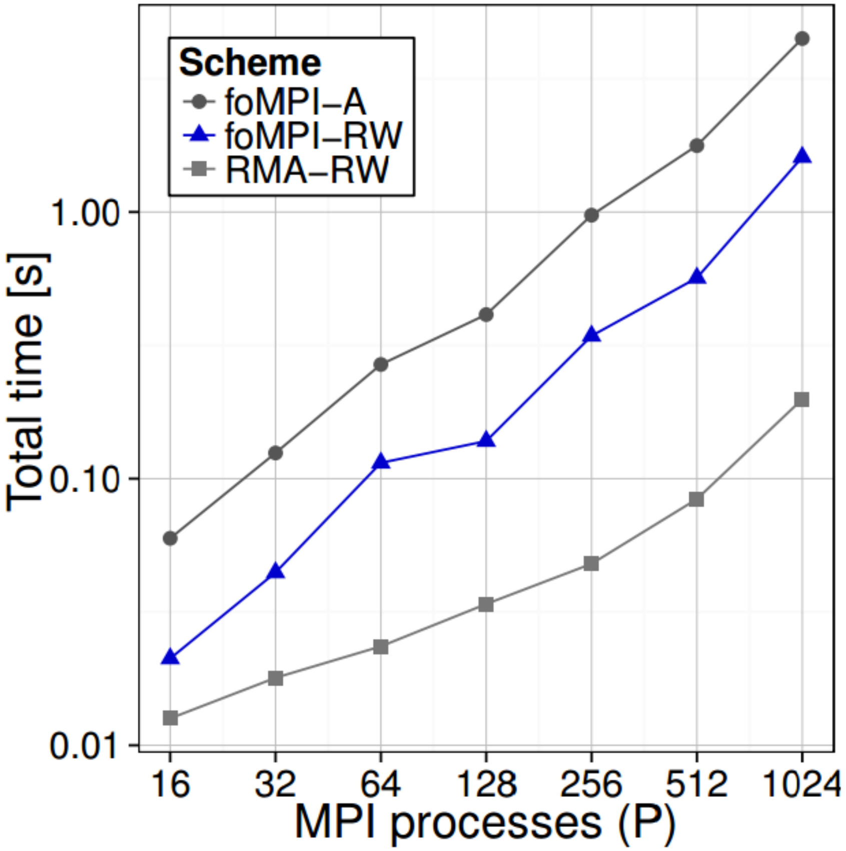




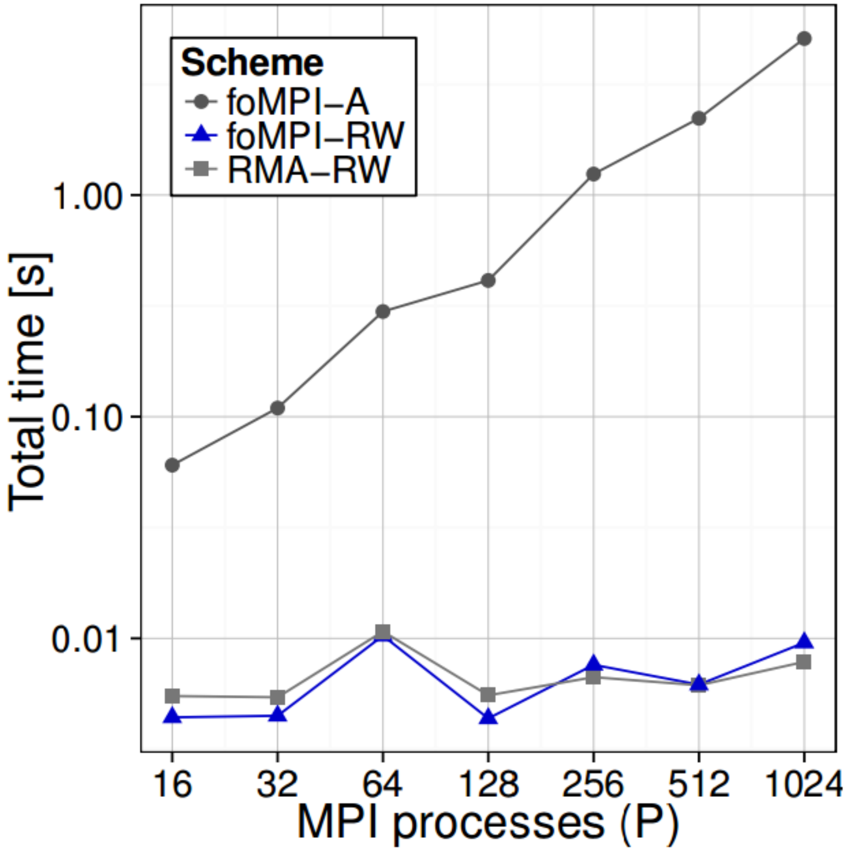
[1] R. Gerstenberger et al. Enabling Highly-scalable Remote Memory Access Programming with MPI-3 One Sided. ACM/IEEE Supercomputing 2013.

## EVALUATION
### DISTRIBUTED HASHTABLE

2% of writers

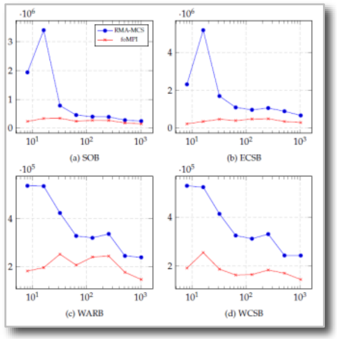


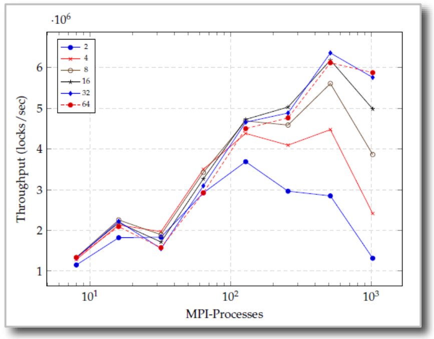
0% of writers

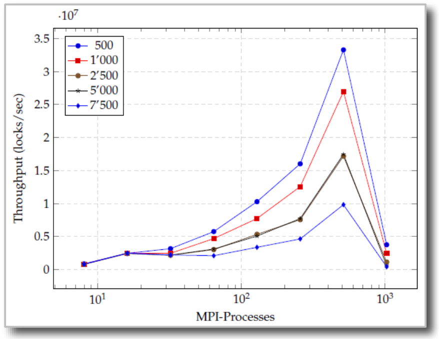


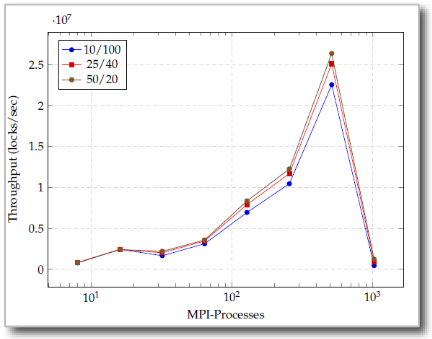
[1] R. Gerstenberger et al. Enabling Highly-scalable Remote Memory Access Programming with MPI-3 One Sided. ACM/IEEE Supercomputing 2013.

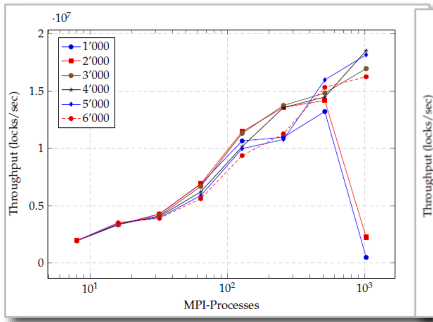
## Other Analyses

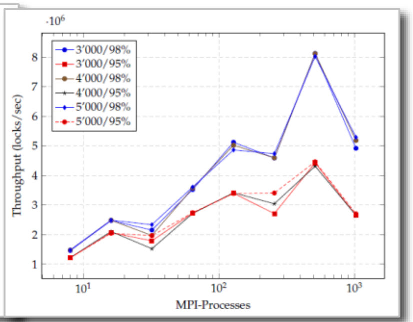


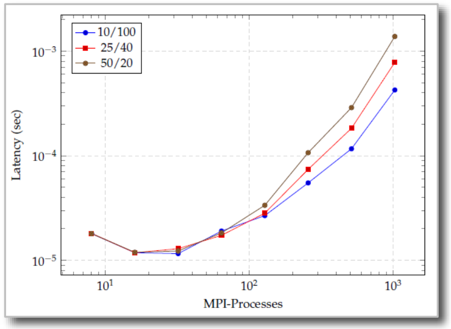


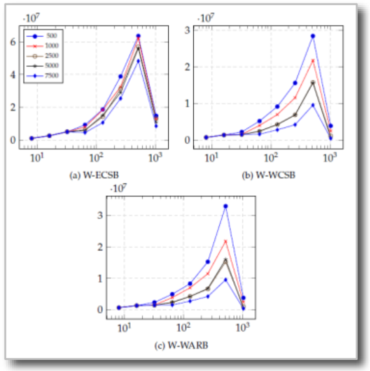


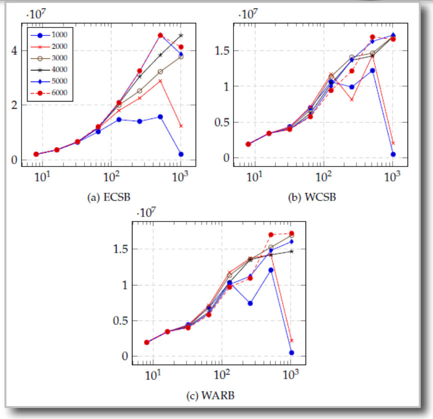


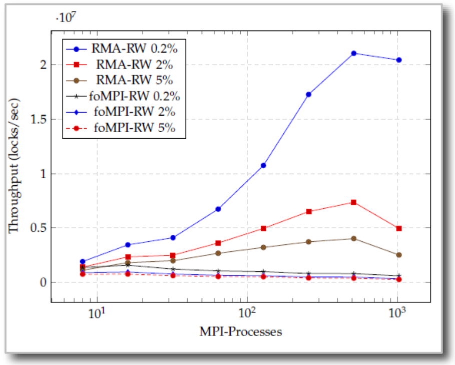


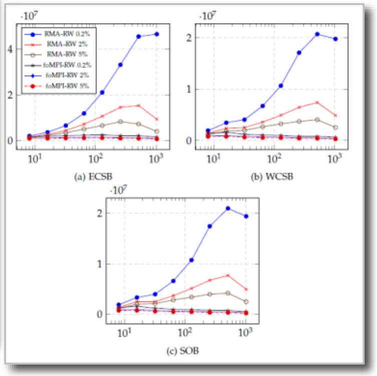


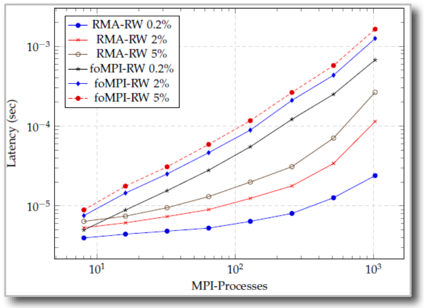




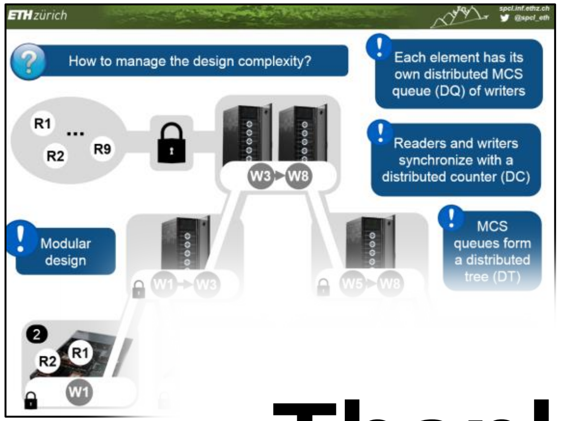








## CONCLUSIONS

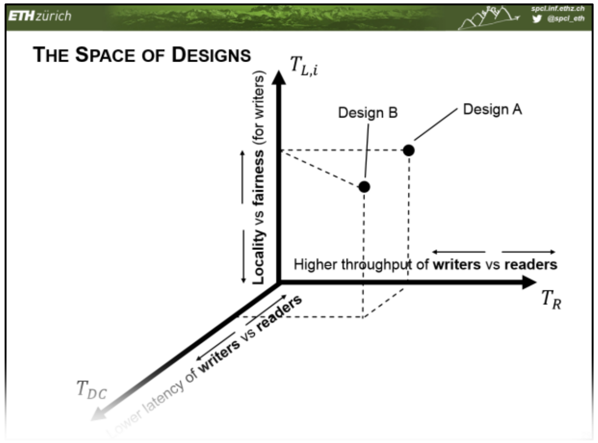

How to manage the design complexity?

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


THE SPACE OF DESIGNS

$T_{L,i}$
Design B
Design A
Locality vs fairness (for writers)
Higher throughput of writers vs readers
$T_R$
Lower latency of writers vs readers
$T_{DC}$

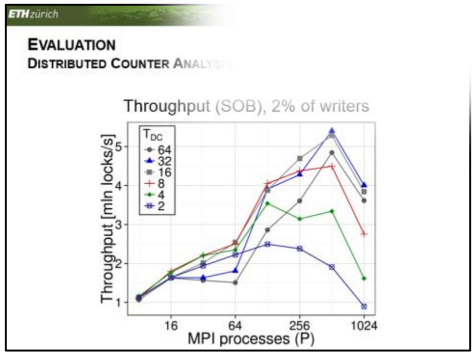
Modular d

corre

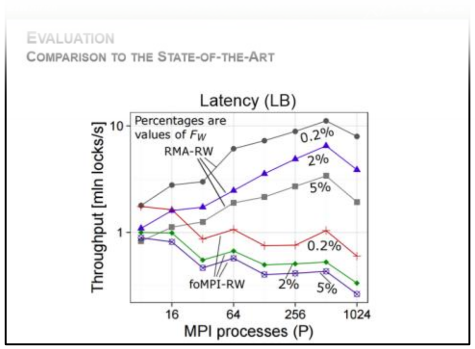
ible

es

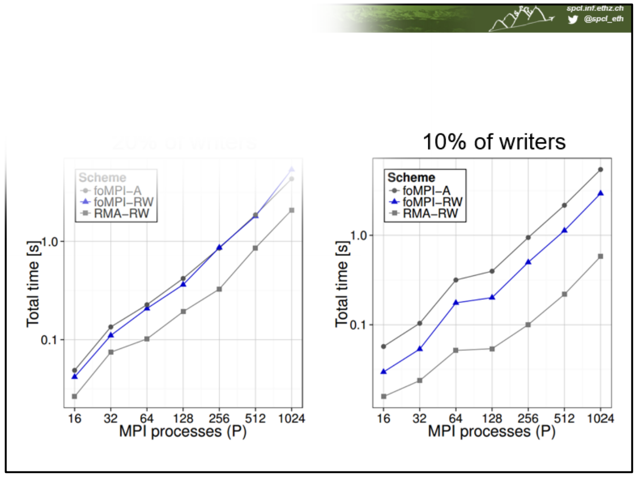
# Thank you for your attention


EVALUATION

DISTRIBUTED COUNTER ANALYSIS


EVALUATION

COMPARISON TO THE STATE-OF-THE-ART



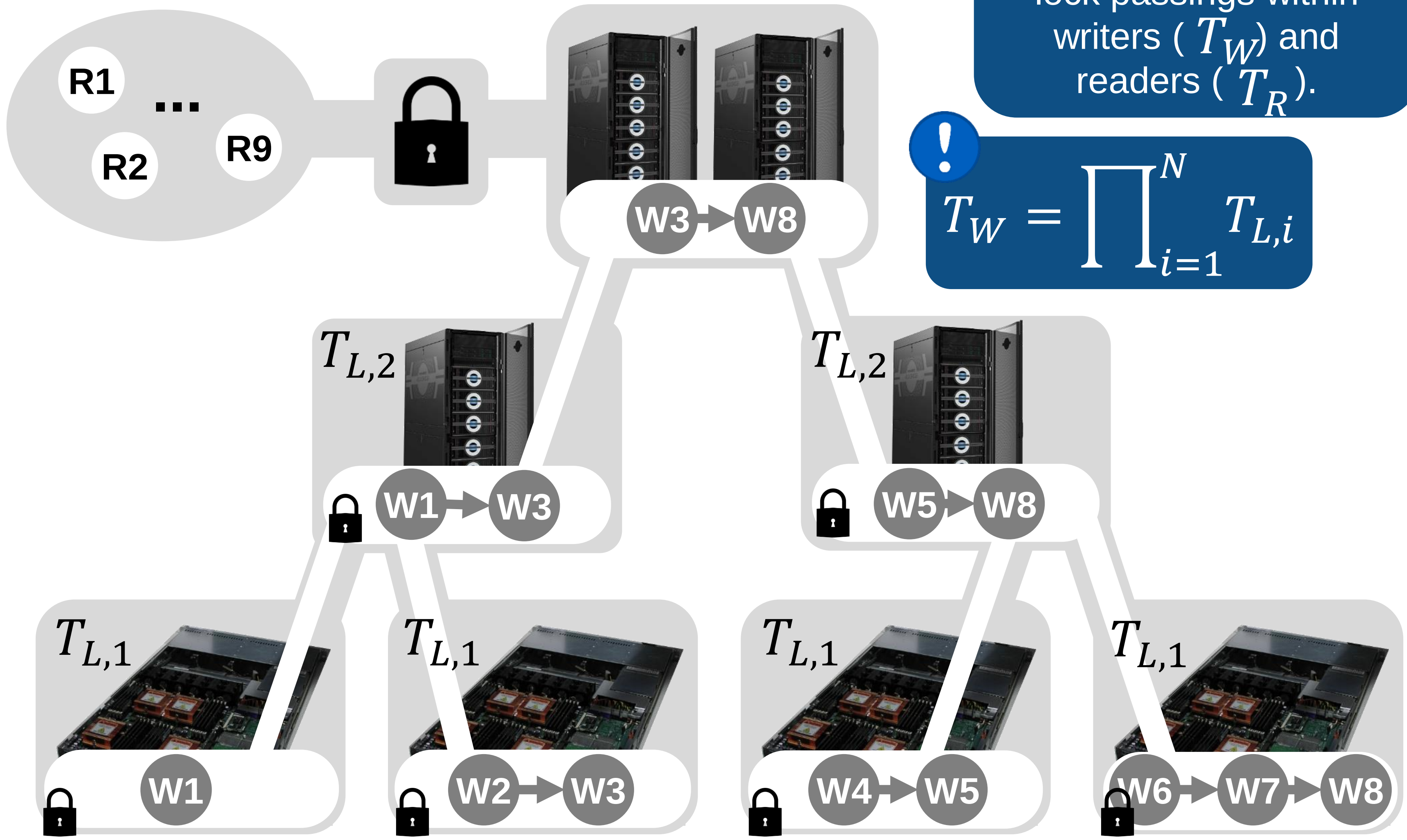
Improves latency and throughput over state-of-the-art

Enables high-performance distributed hashtabled

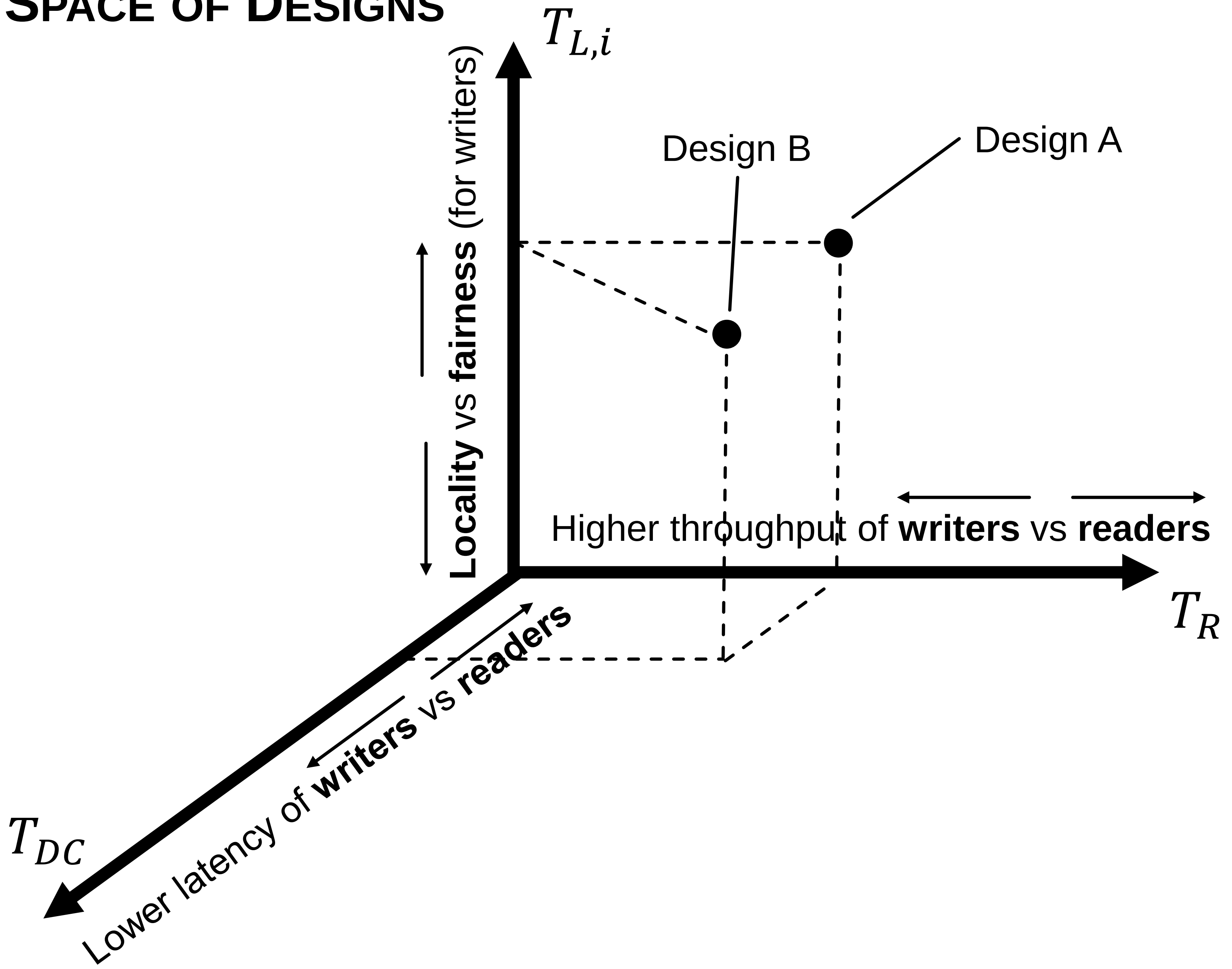
## DISTRIBUTED TREE OF QUEUES (DT)
### Throughput of readers vs writers

DT: The maximum number of consecutive lock passings within writers ( $T_W$) and readers ( $T_R$ ).

$$T_W = \prod_{i=1}^{N} T_{L,i}$$

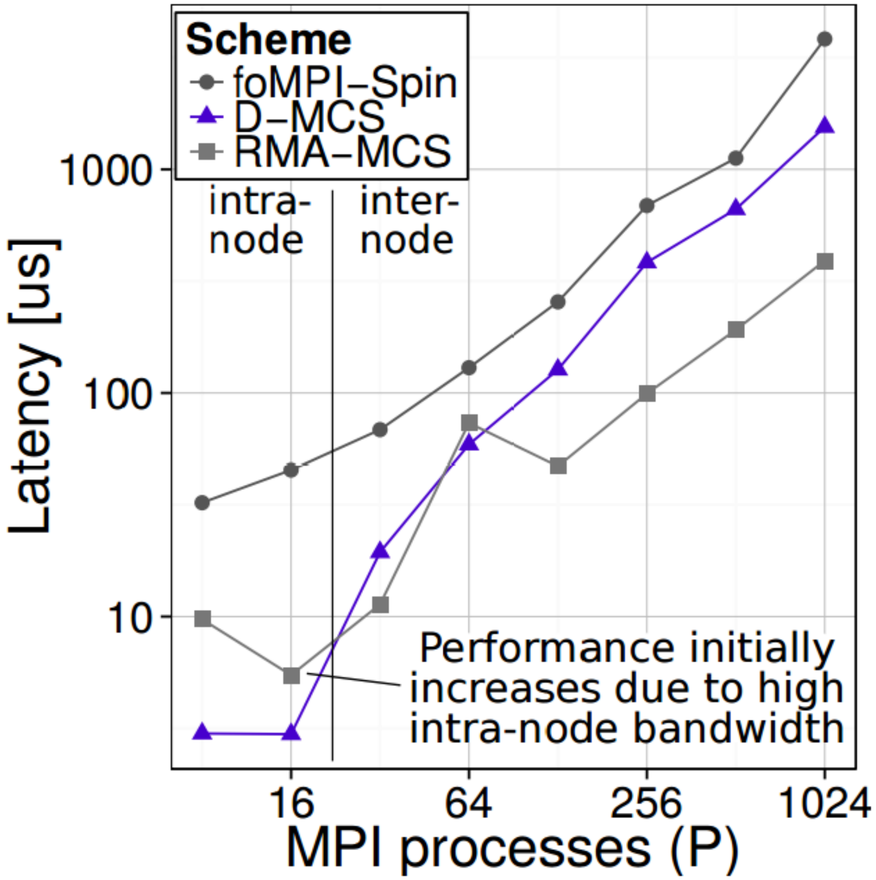


## The Space of Designs

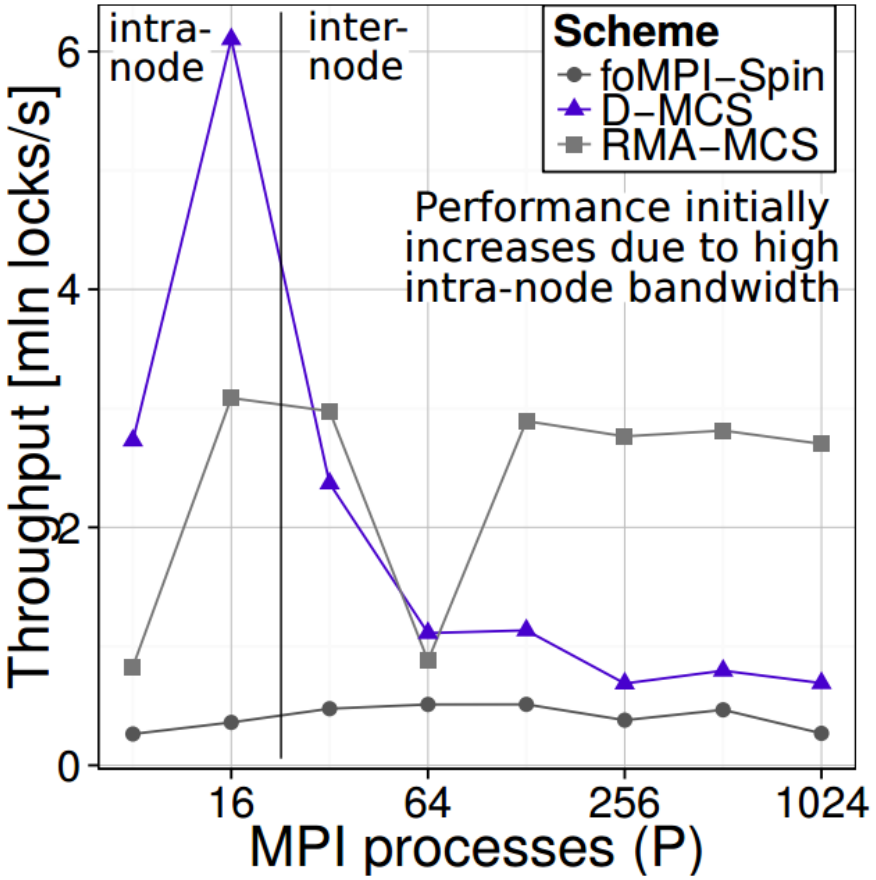

$T_{L,i}$
Locality vs fairness (for writers)
Design B
Design A
Higher throughput of writers vs readers
$T_R$
$T_{DC}$
Lower latency of writers vs readers

## EVALUATION
### D-MCS VS OTHERS

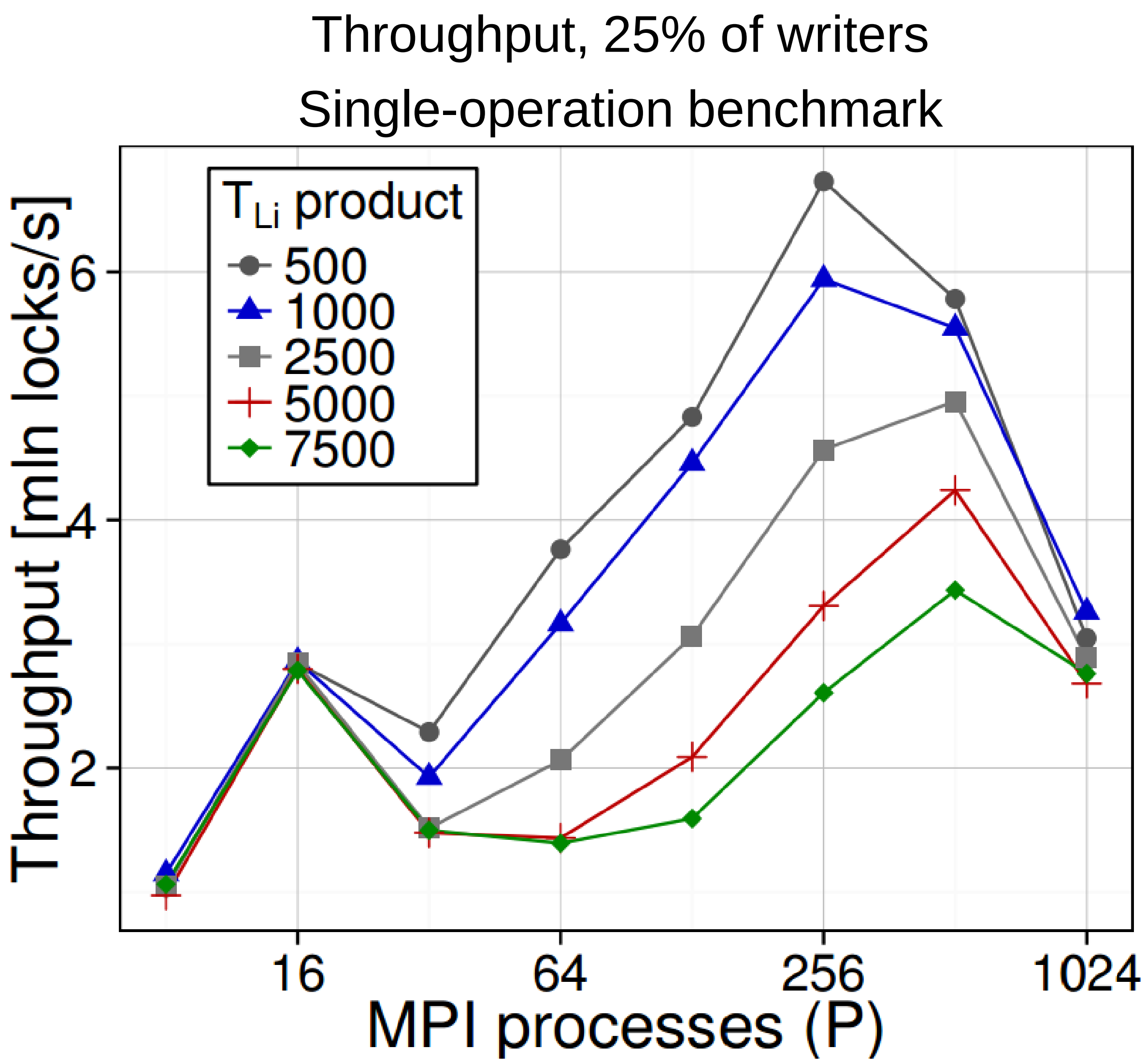
Latency (LB)



Throughput (ECSB)

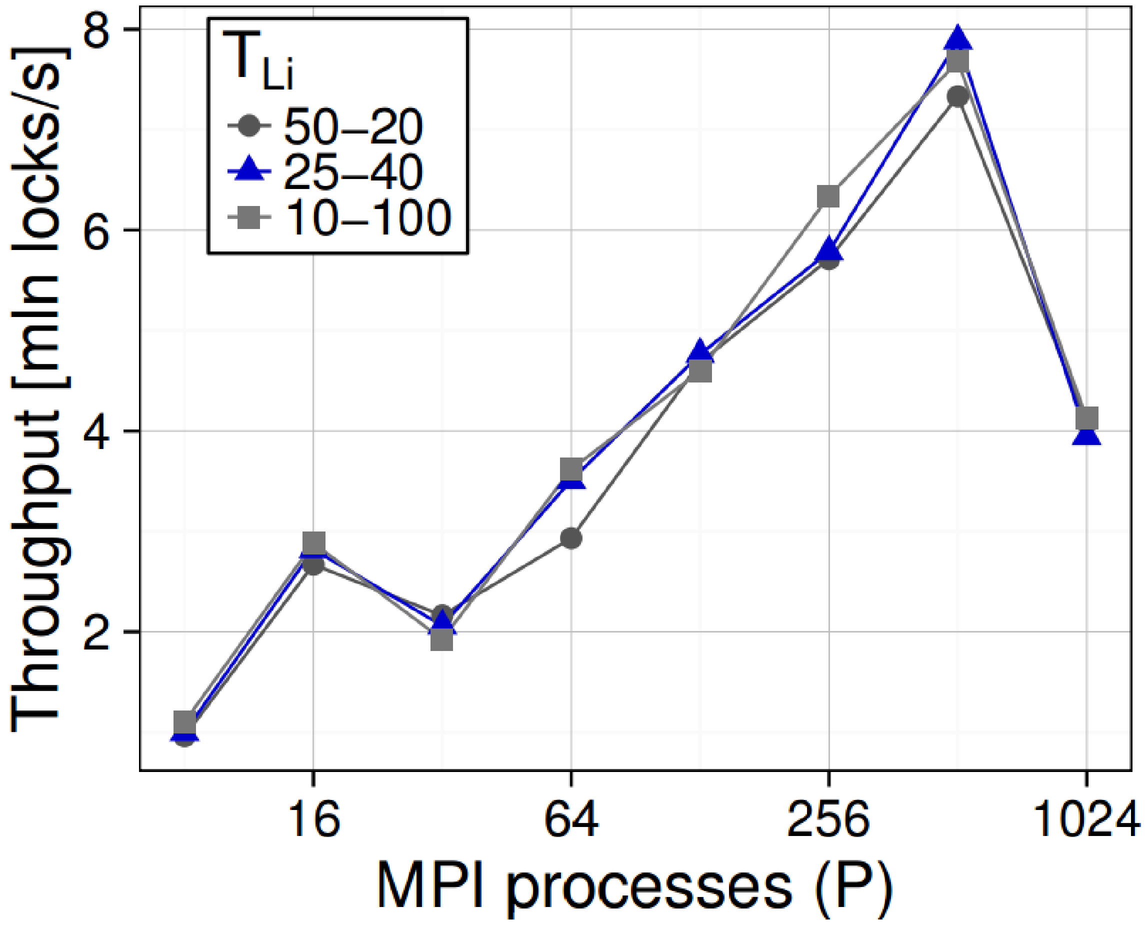


## EVALUATION
### WRITER THRESHOLD ANALYSIS



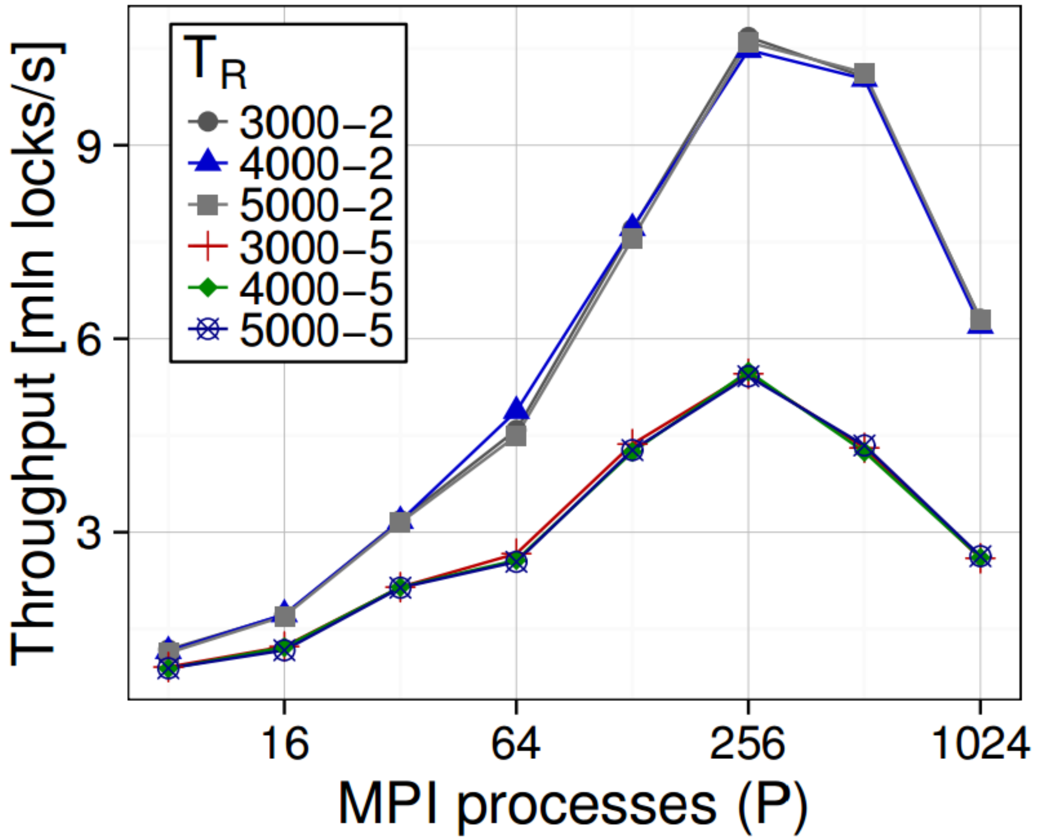
# EVALUATION
## FAIRNESS VS THROUGHPUT ANALYSIS

Throughput, 25% of writers,
Single-operation benchmark



# Evaluation
## Reader Threshold Analysis

Throughput, 2% and 5% writers,
Empty-critical-section benchmark



## FEASIBILITY ANALYSIS

| | UPC (standard) [44] | Berkeley UPC [1] | SHMEM [4] |
|---|---|---|---|
| Put | UPC_SET | bupc_atomicX_set_RS | shmem_swap |
| Get | UPC_GET | bupc_atomicX_read_RS | shmem_mswap |
| Accumulate | UPC_INC | bupc_atomicX_fetchadd_RS | shmem_fadd |
| FAO (SUM) | UPC_INC, UPC_DEC | bupc_atomicX_fetchadd_RS | shmem_fadd |
| FAO (REPLACE) | UPC_SET | bupc_atomicX_swap_RS | shmem_swap |
| CAS | UPC_CSWAP | bupc_atomicX_cswap_RS | shmem_cswap |

| | Fortran 2008 [27] | Linux RDMA/IB [33, 43] | iWARP [18, 41] |
|---|---|---|---|
| Put | atomic_define | MskCmpSwap | masked CmpSwap |
| Get | atomic_ref | MskCmpSwap | masked CmpSwap |
| Accumulate | atomic_add | FetchAdd | FetchAdd |
| FAO (SUM) | atomic_add | FetchAdd | FetchAdd |
| FAO (REPLACE) | atomic_define* | MskCmpSwap | masked CmpSwap |
| CAS | atomic_cas | CmpSwap | CmpSwap |