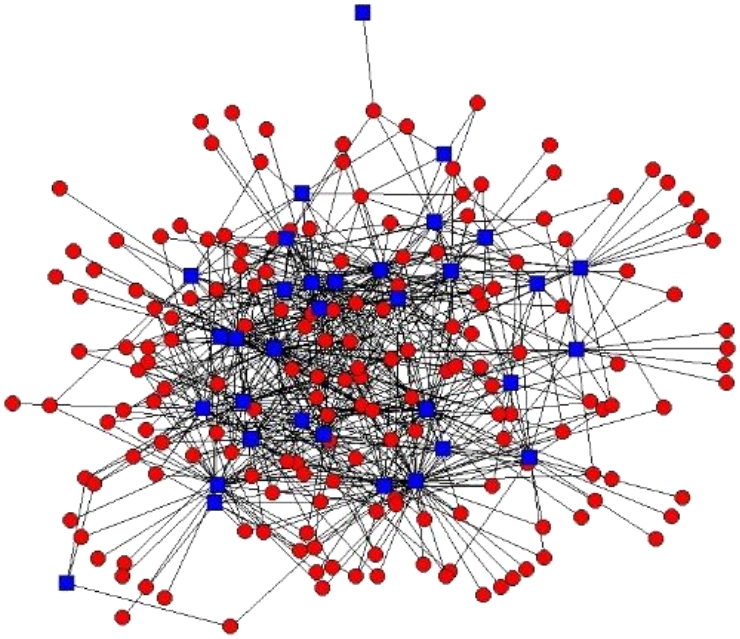
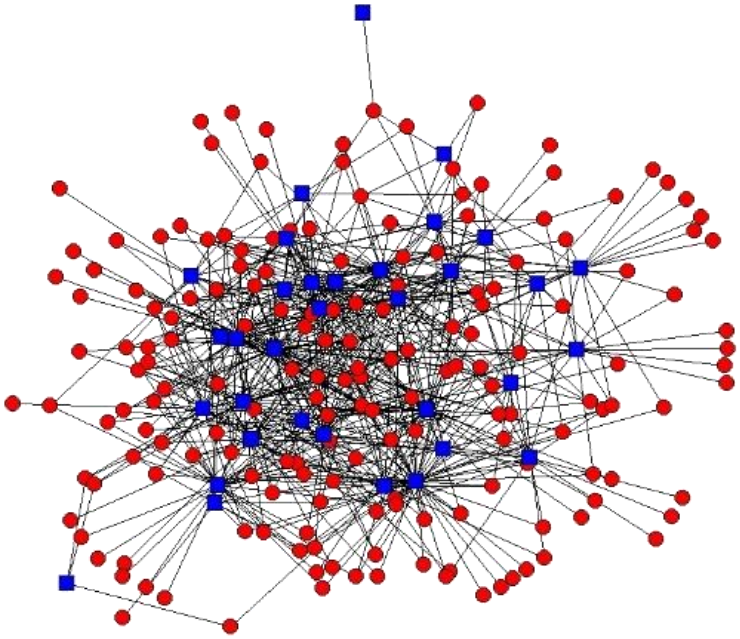
High-Performance Distributed RMA Locks

PATRICK SCHMID, MACIEJ BESTA, TORSTEN HOEFLER

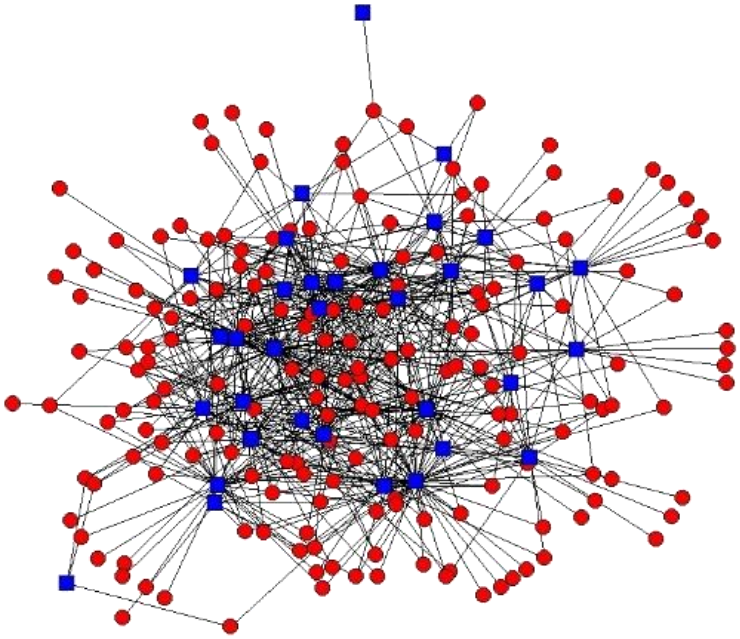
Need for Efficient large-Scale Synchronization



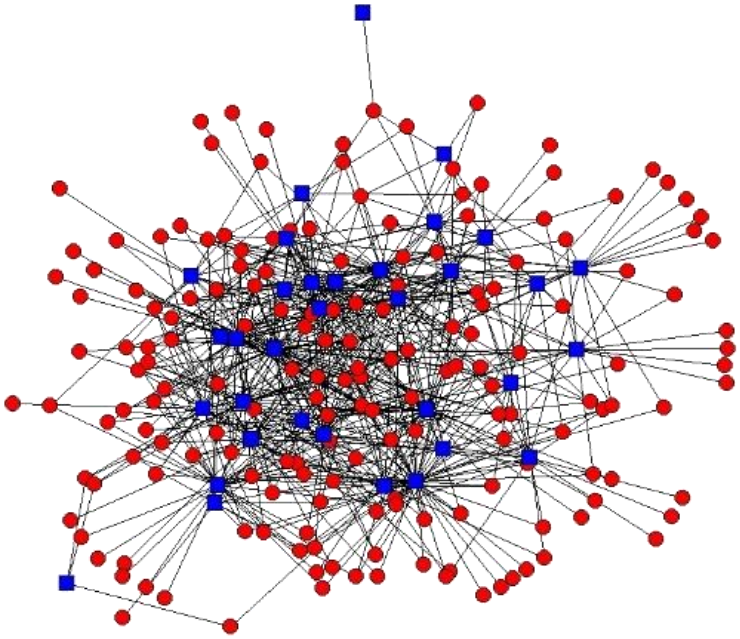
Need for Efficient large-Scale Synchronization



Need for Efficient large-Scale Synchronization

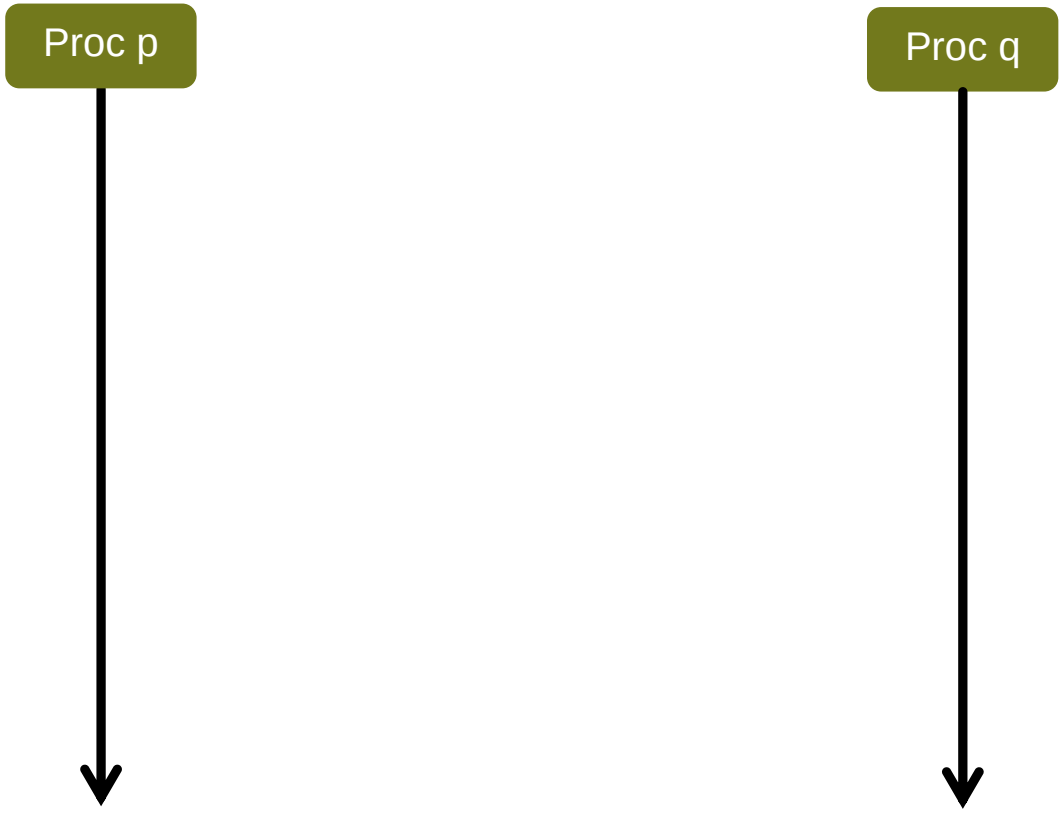


Need for Efficient large-Scale Synchronization

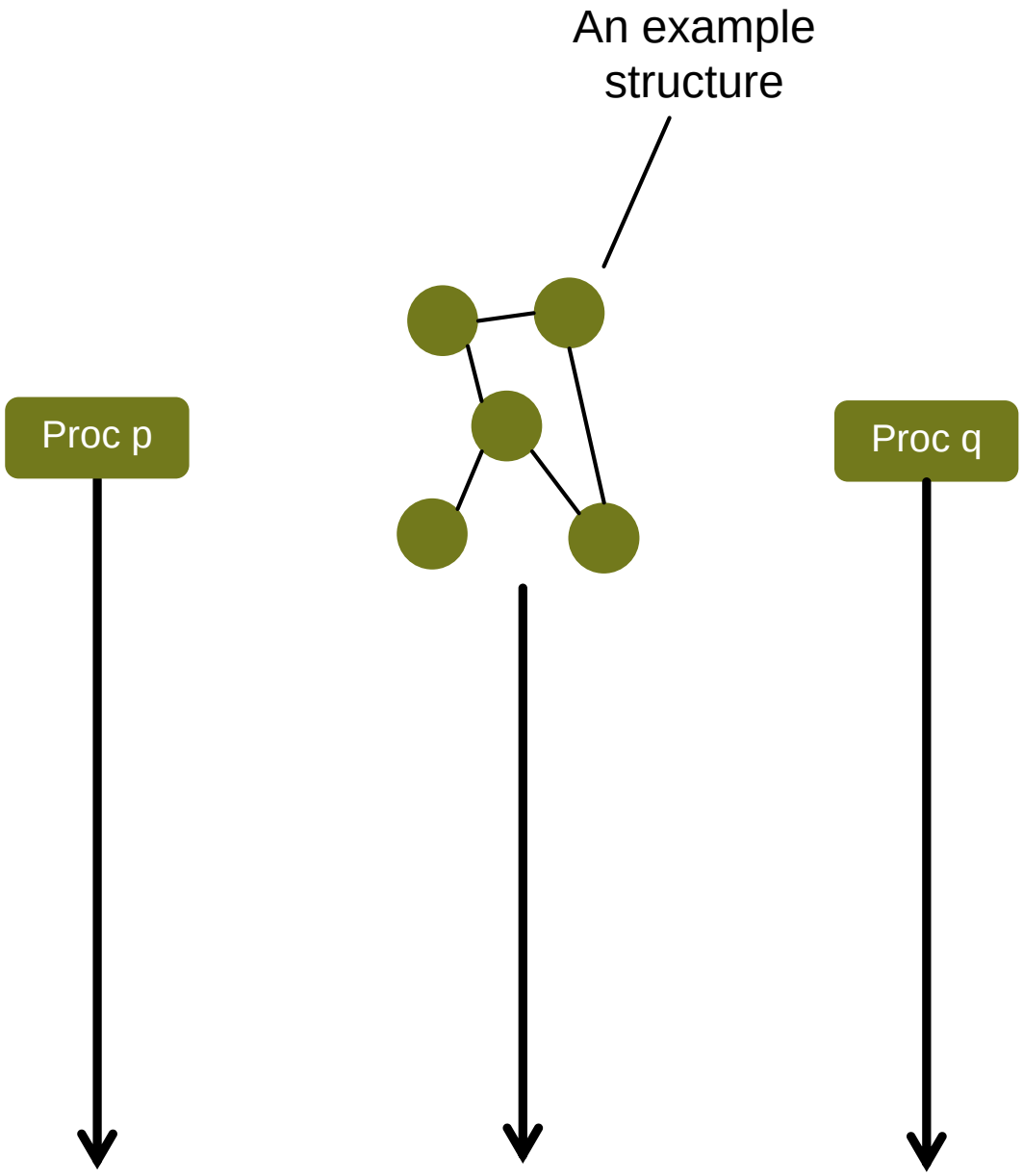


LOCKS

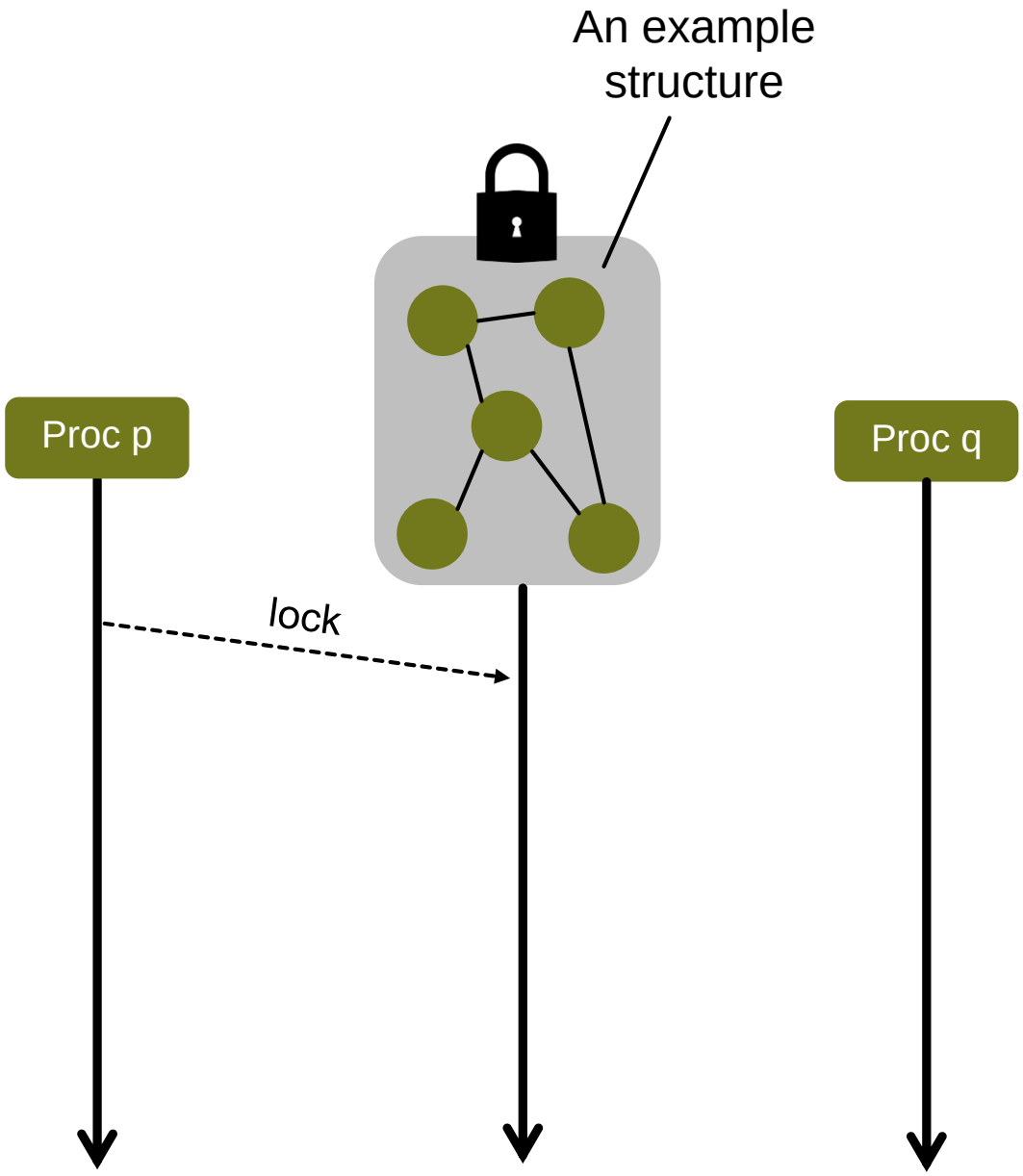
Locks



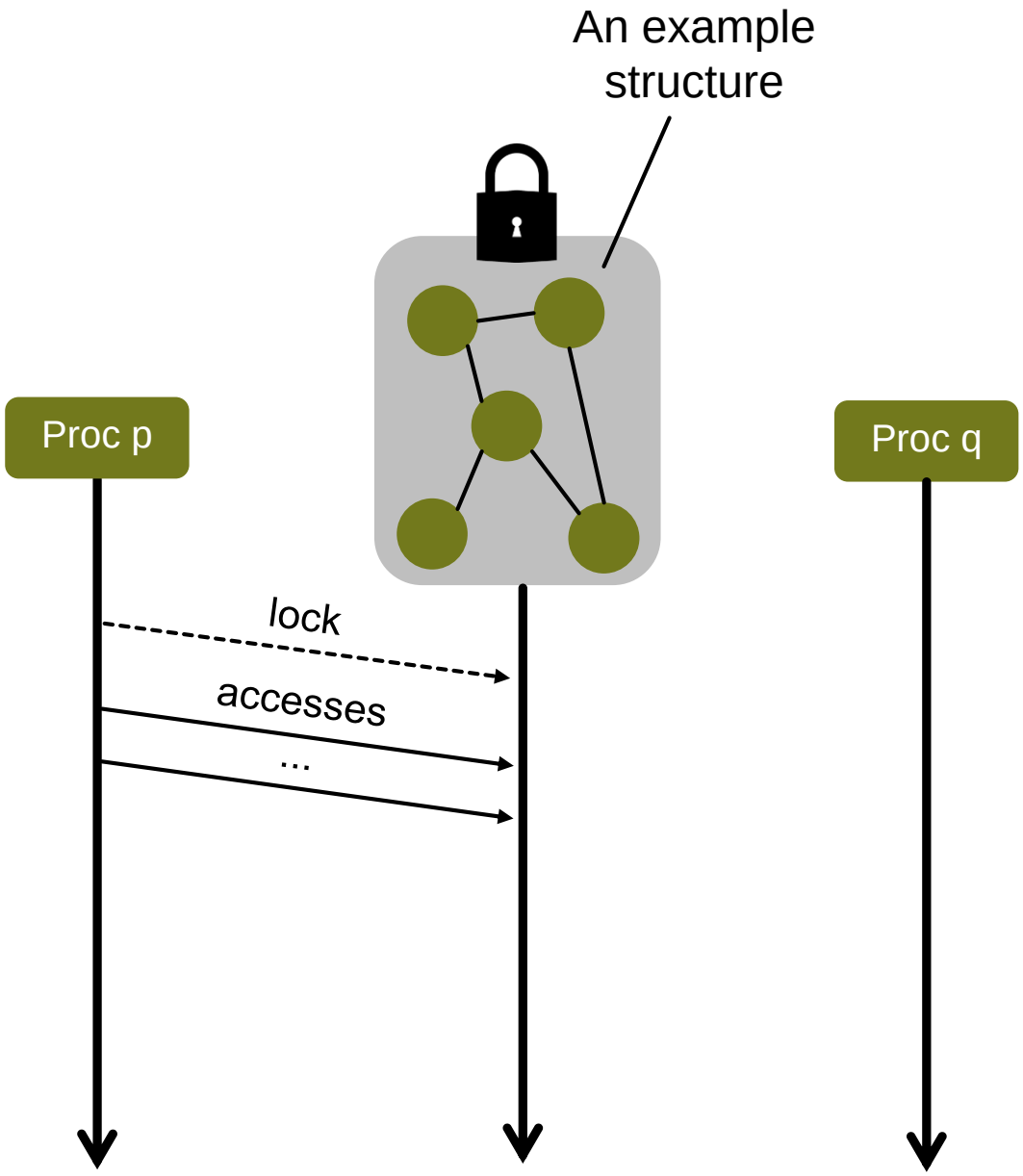
LOCKS



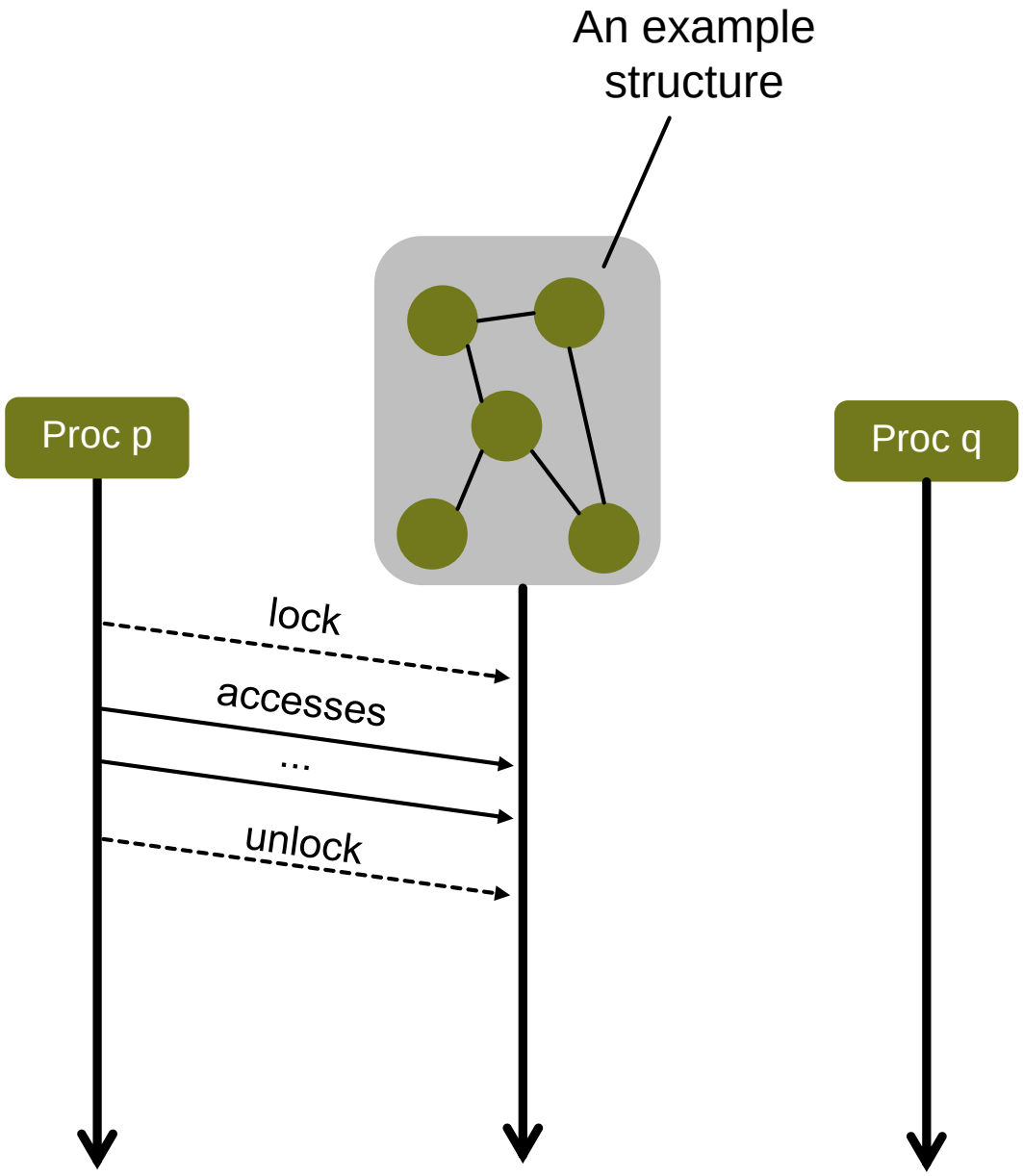
Locks



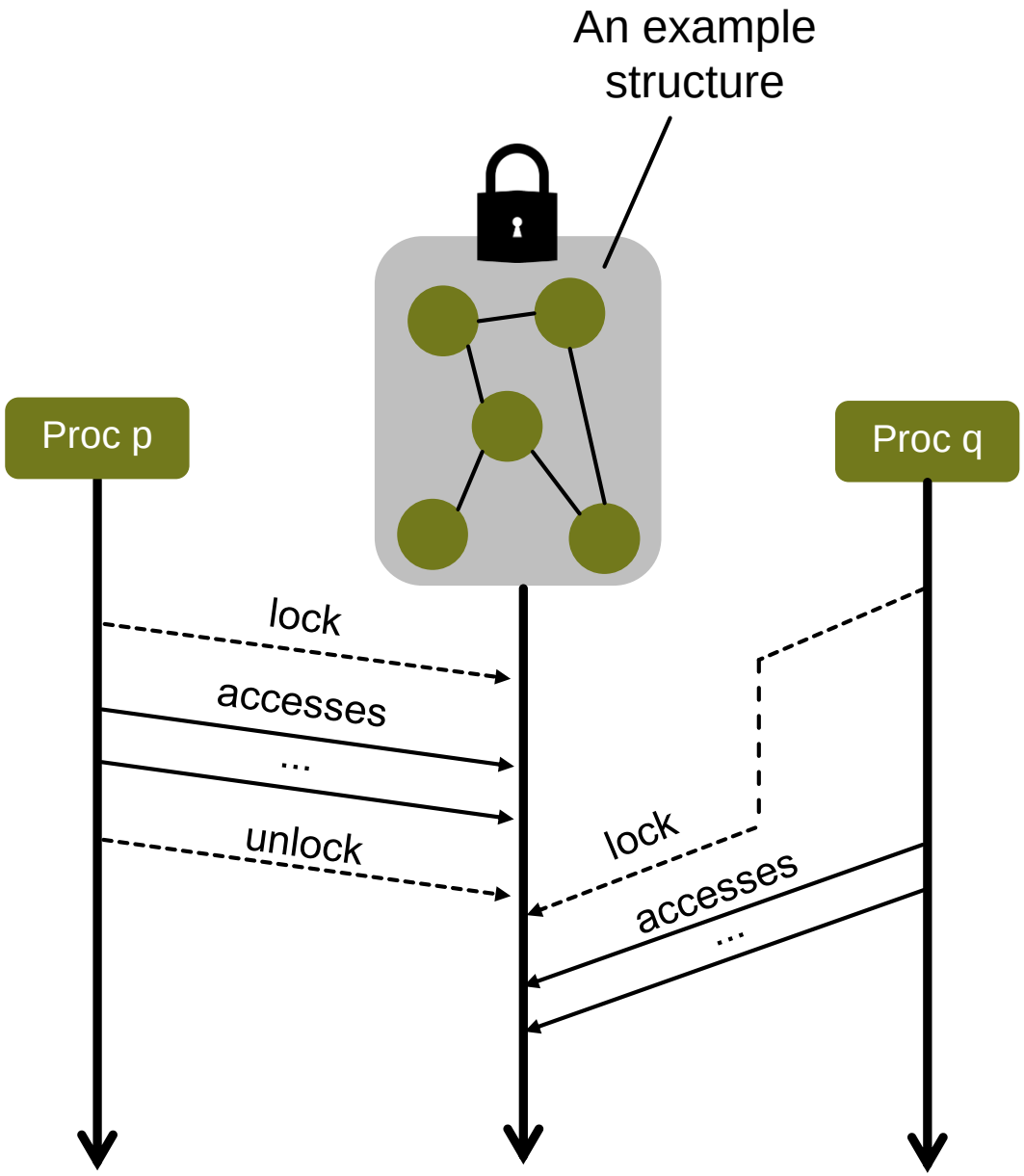
Locks



Locks

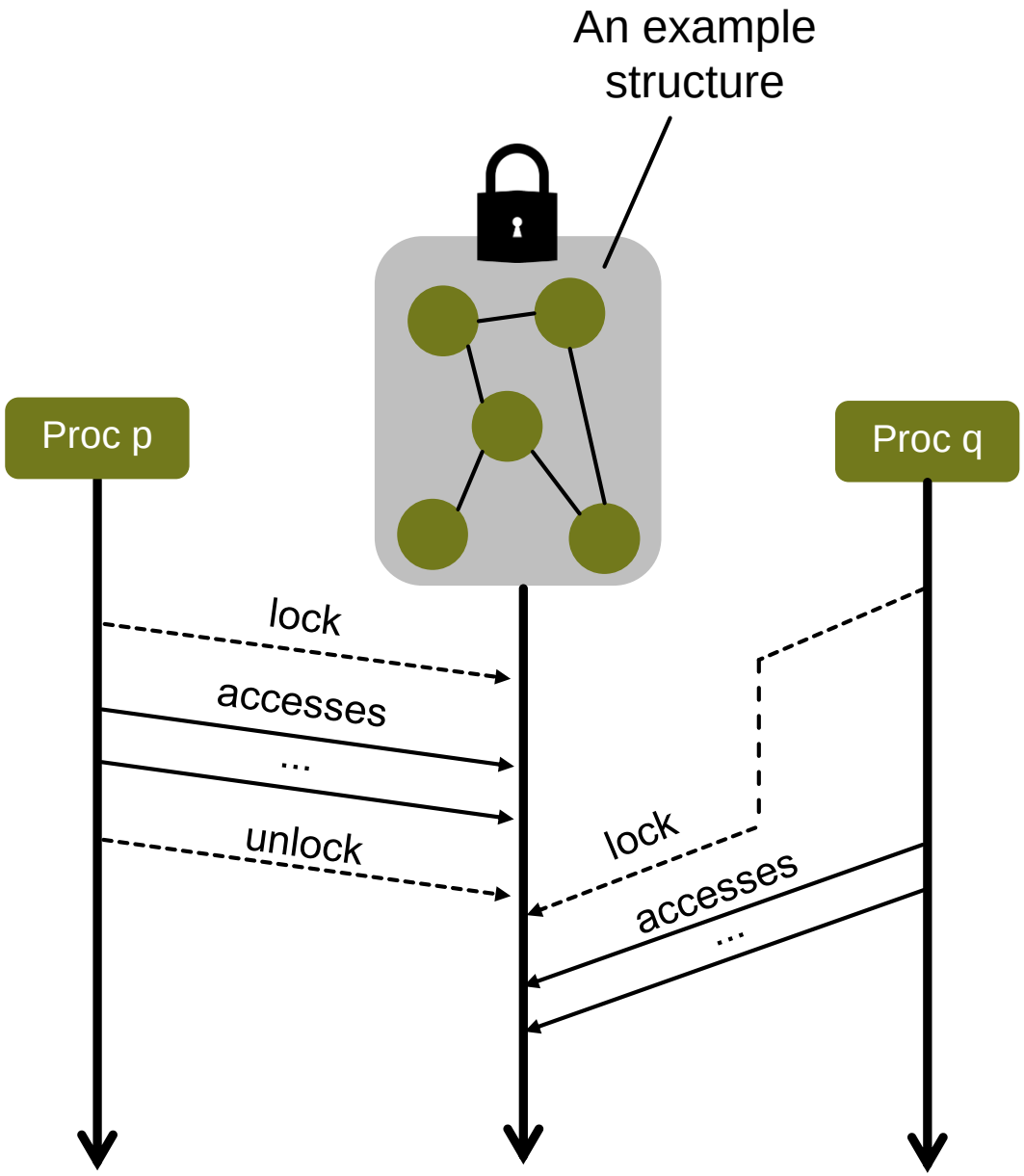


Locks


An example structure
Proc p
Proc q
lock
accesses
...
unlock
lock
accesses
...

Locks

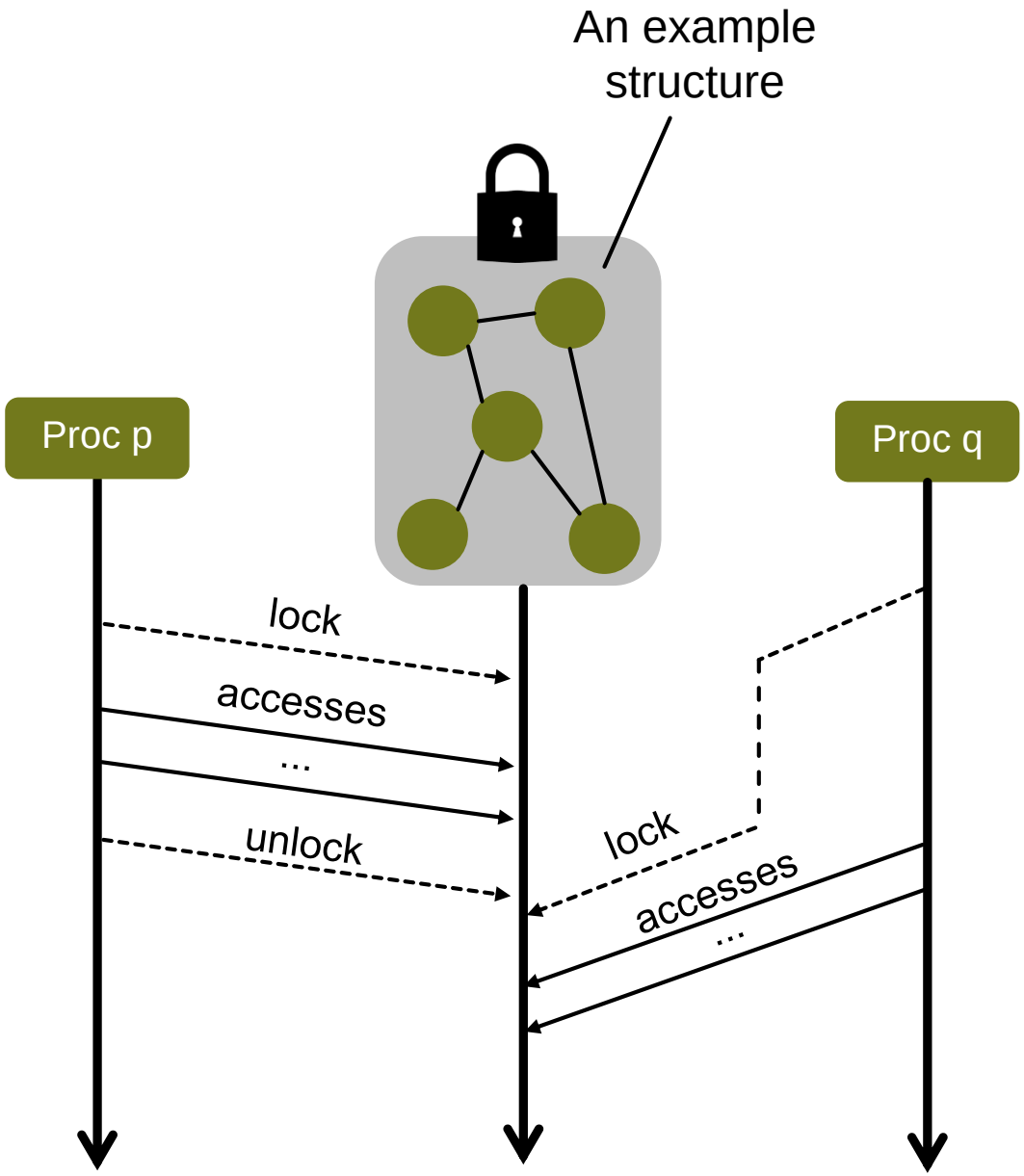

Inuitive
semantics


An example
structure
Proc p
Proc q
lock
accesses
...
unlock
lock
accesses
...

Locks

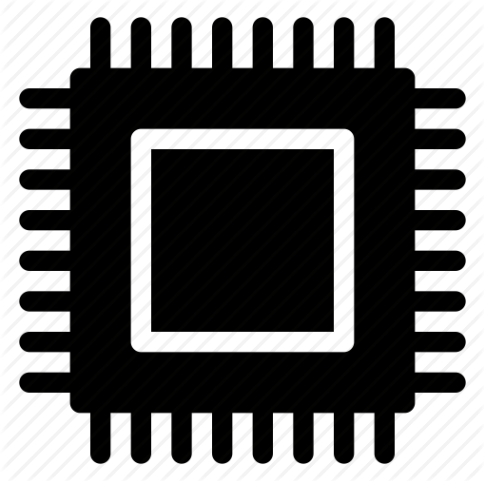
Inuitive semantics

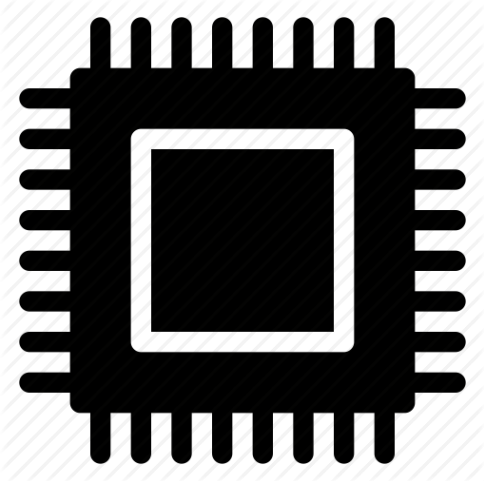
Various performance penalties



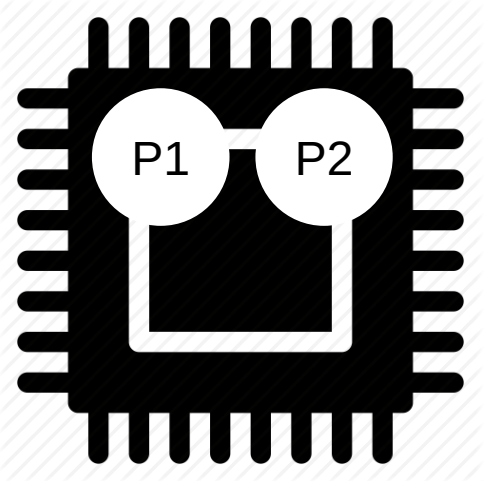
LOCKS: CHALLENGES

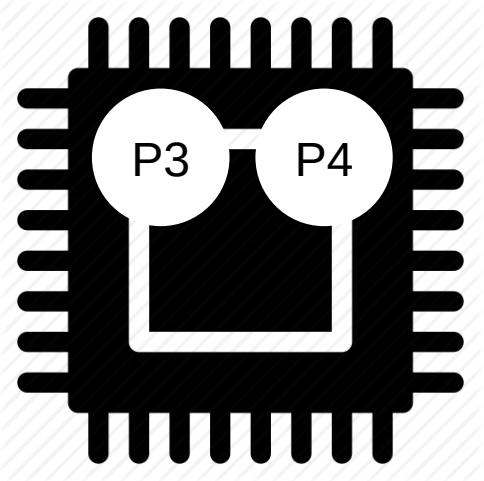
LOCKS: CHALLENGES



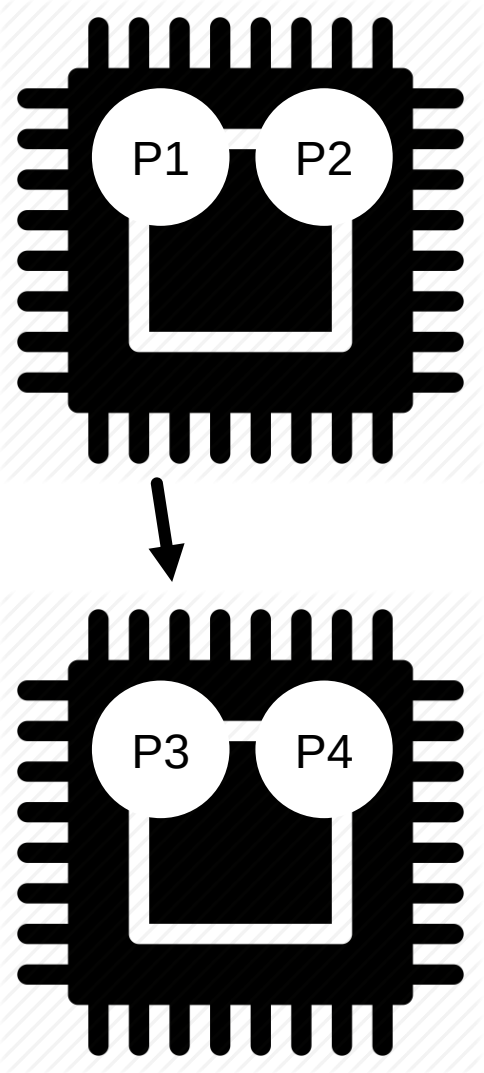


LOCKS: CHALLENGES

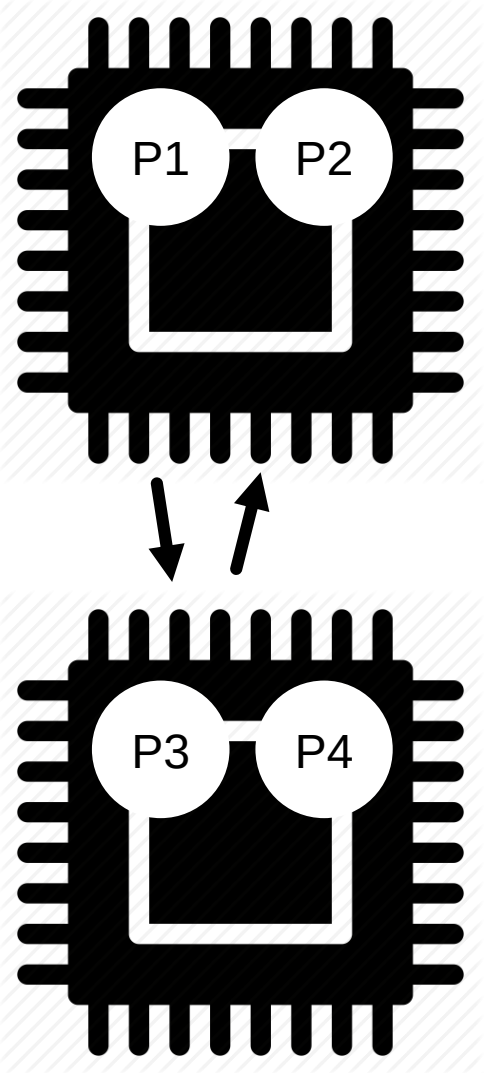




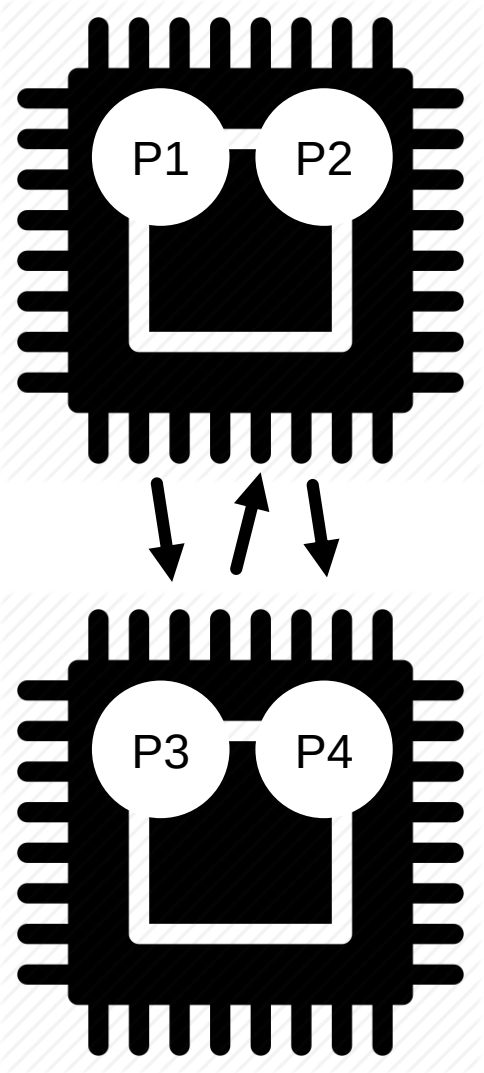
LOCKS: CHALLENGES



LOCKS: CHALLENGES

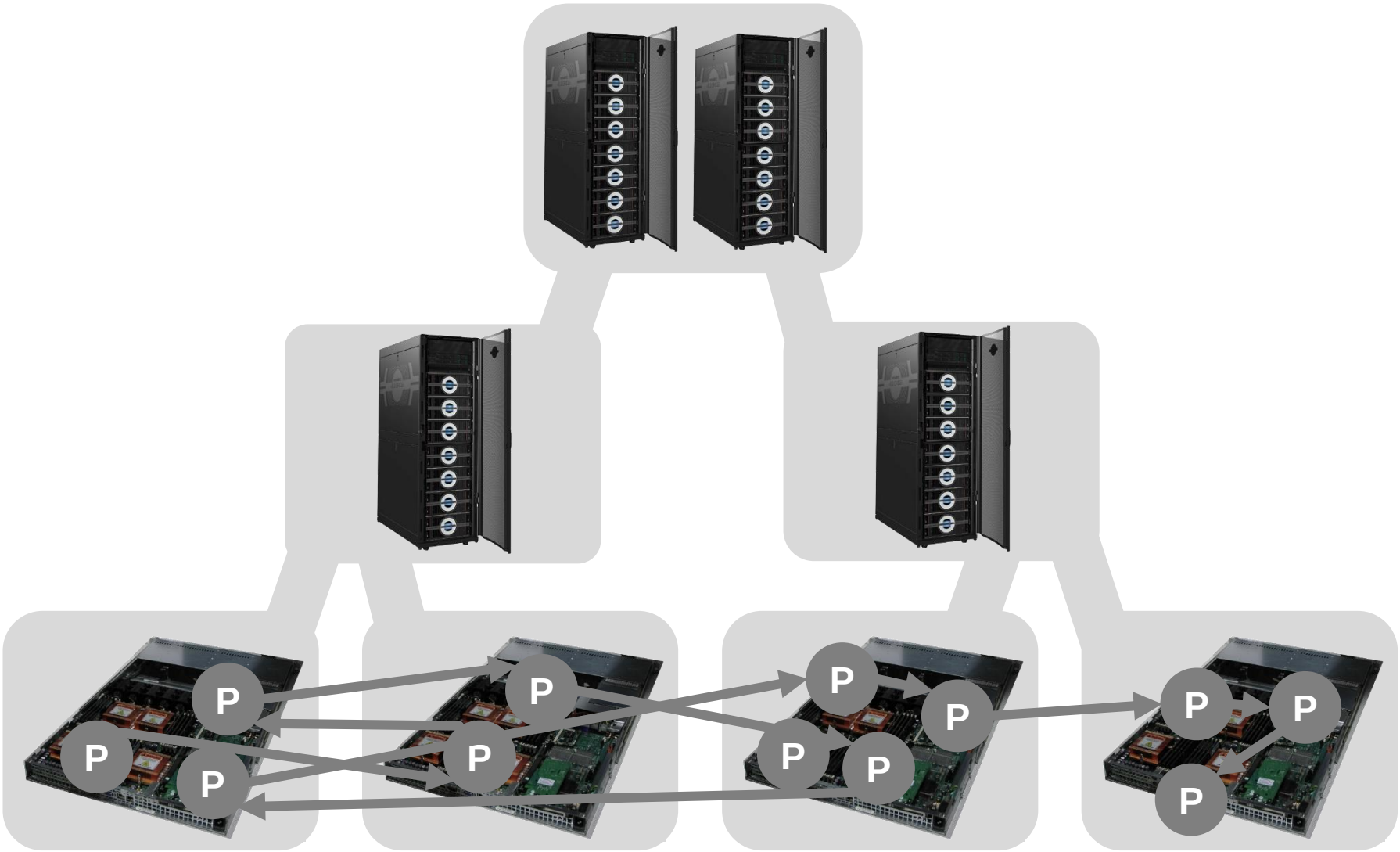


LOCKS: CHALLENGES



LOCKS: CHALLENGES

LOCKS: CHALLENGES



LOCKS: CHALLENGES

We need intra- and inter-node topology-awareness

We need to cover arbitrary topologies

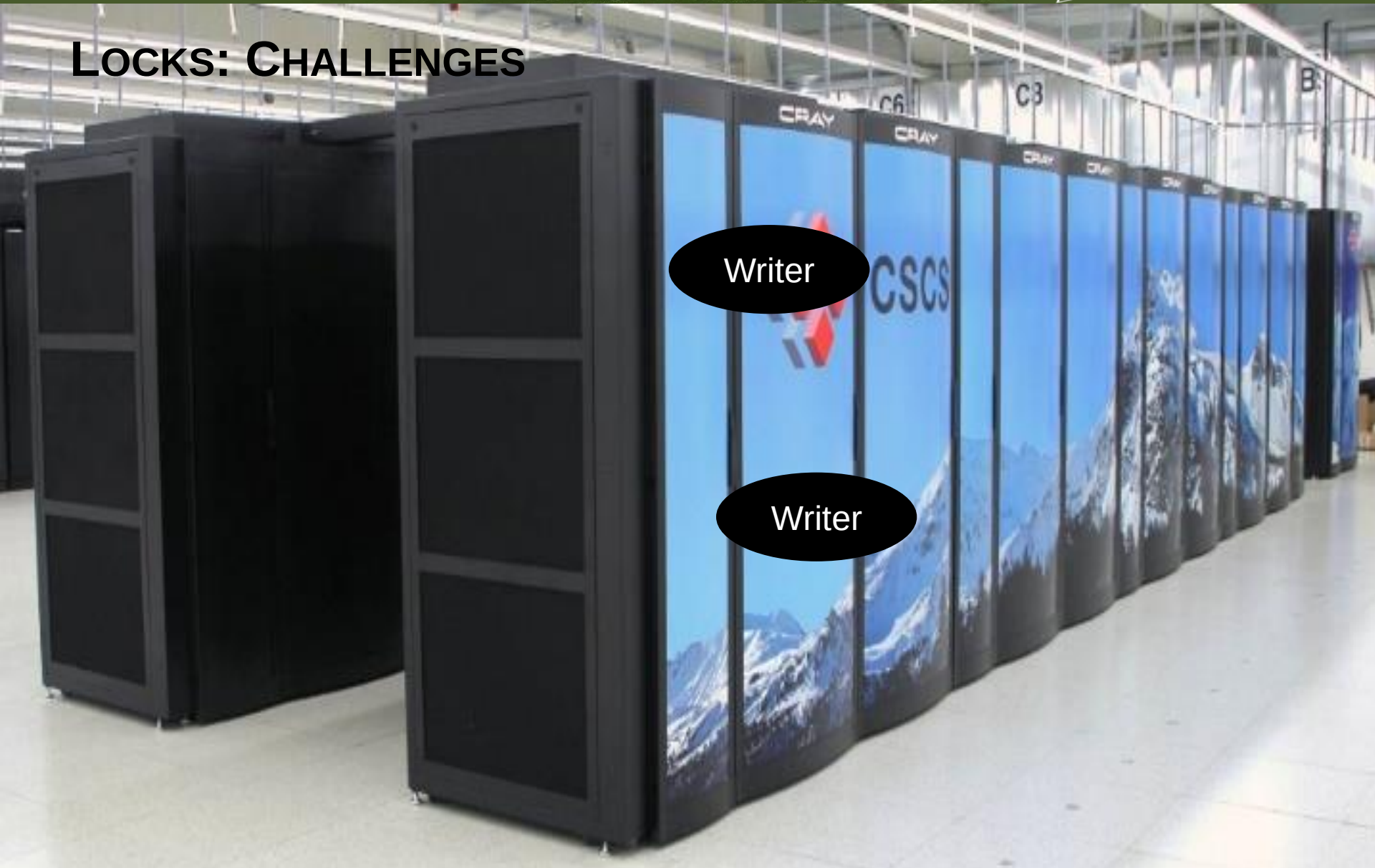
LOCKS: CHALLENGES

LOCKS: CHALLENGES

LOCKS: CHALLENGES

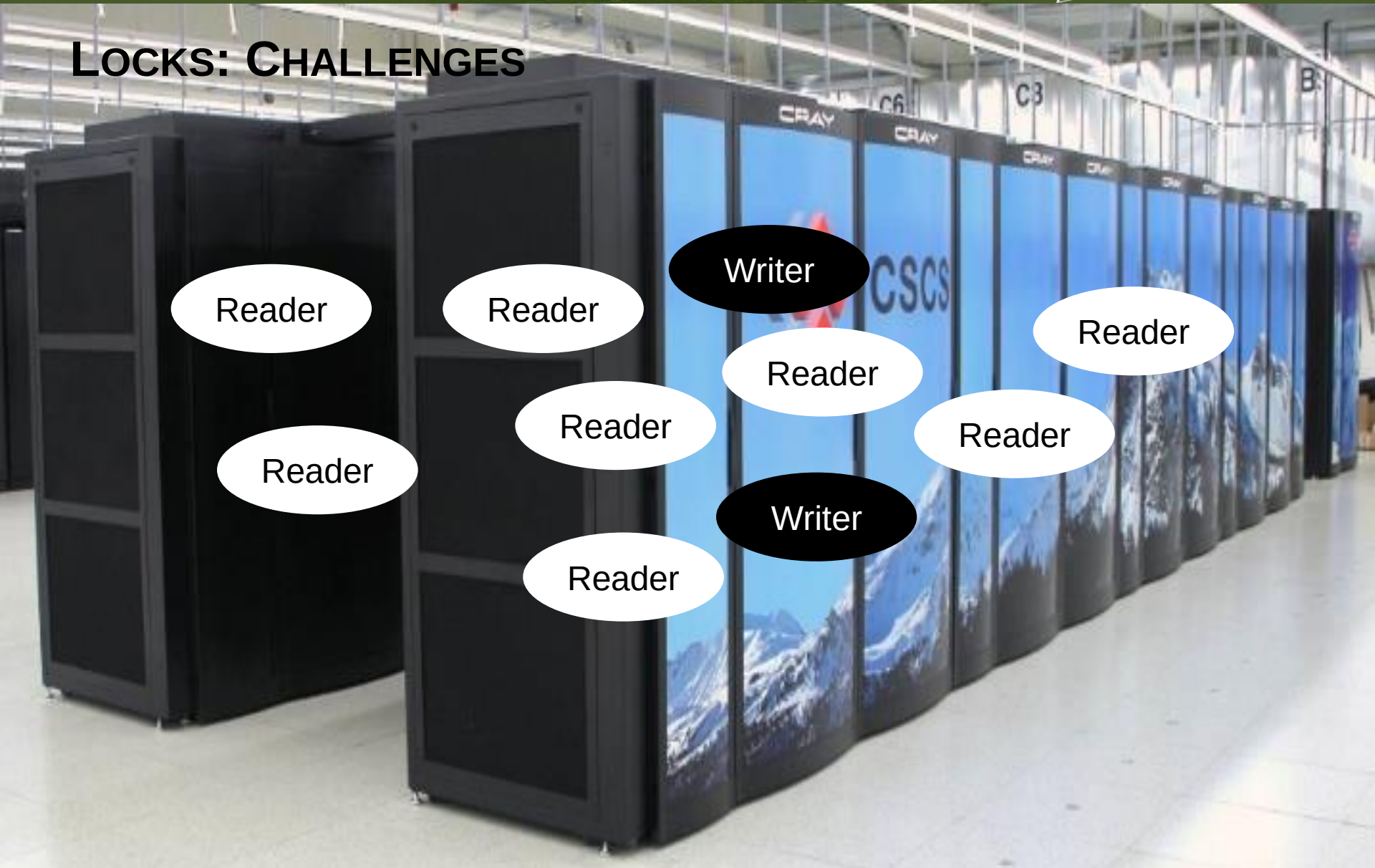
[1] V. Venkataramani et al. Tao: How facebook serves the social graph. SIGMOD'12.

LOCKS: CHALLENGES



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

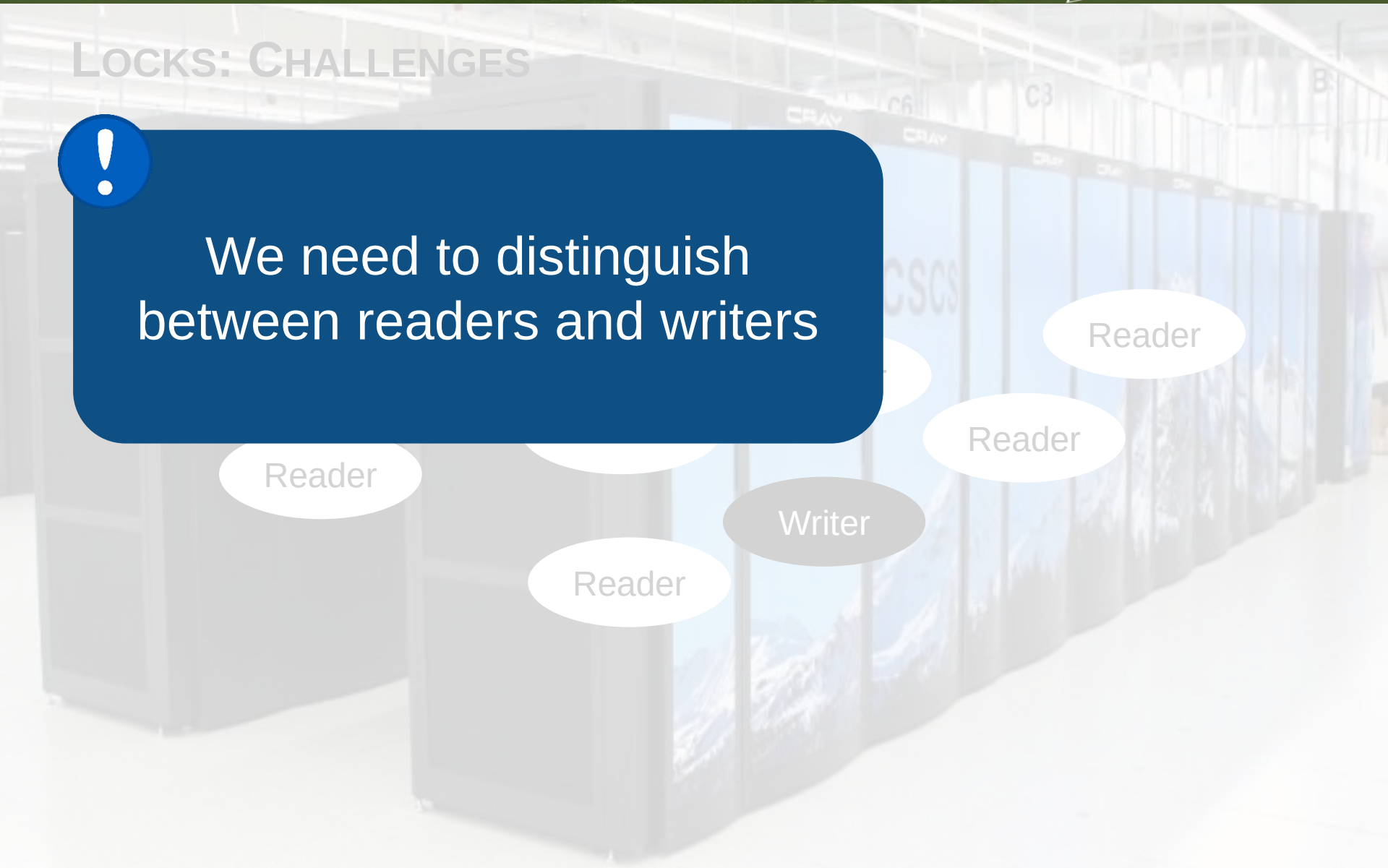
LOCKS: CHALLENGES



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

LOCKS: CHALLENGES

We need to distinguish between readers and writers



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

Locks: Challenges

We need to distinguish between readers and writers

Reader

Reader

Reader

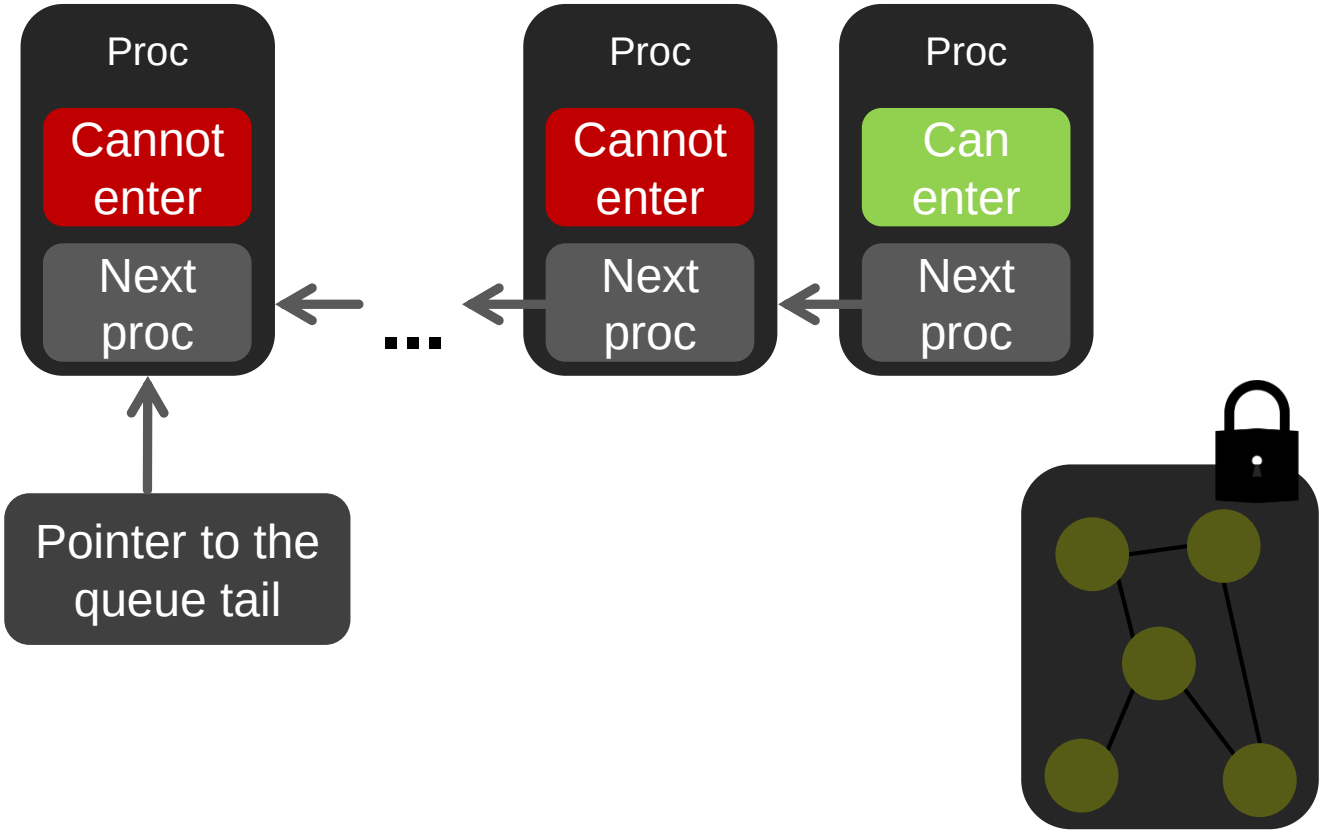
Writer

We need flexible performance for both types of processes

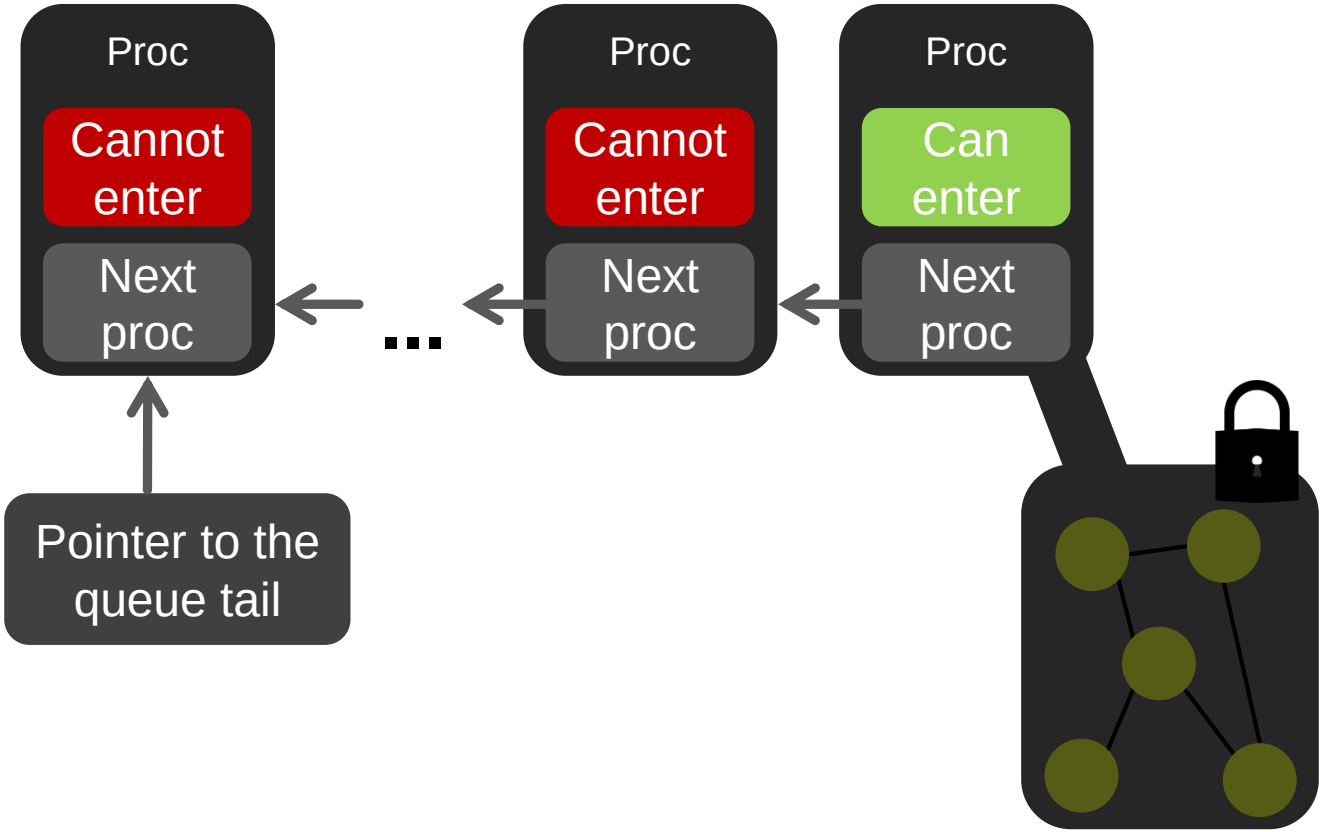
[1] V. Venkataramani et al. Tao: How facebook serves the social graph. SIGMOD'12.

What will we use in the design?

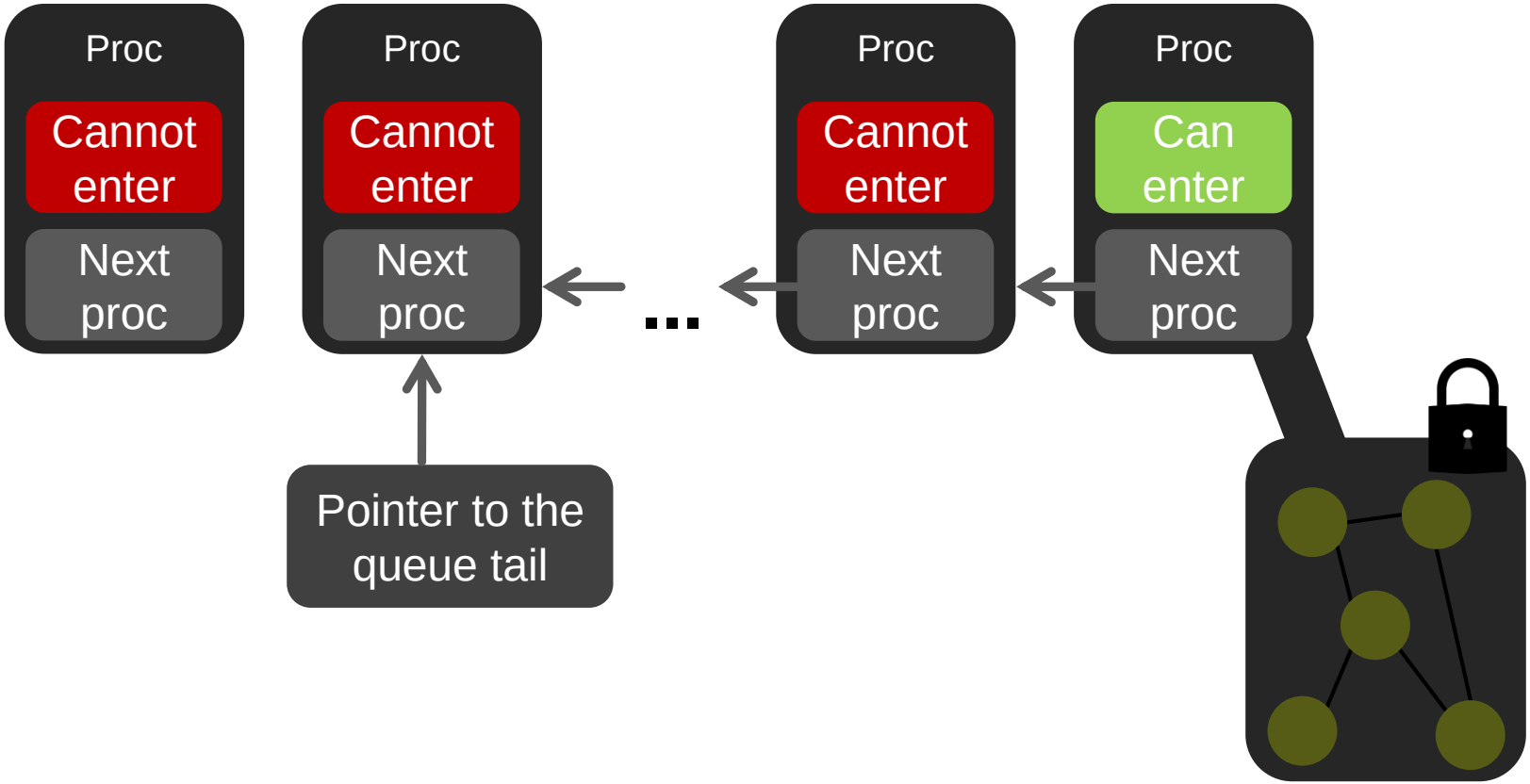
What we will use
MCS Locks



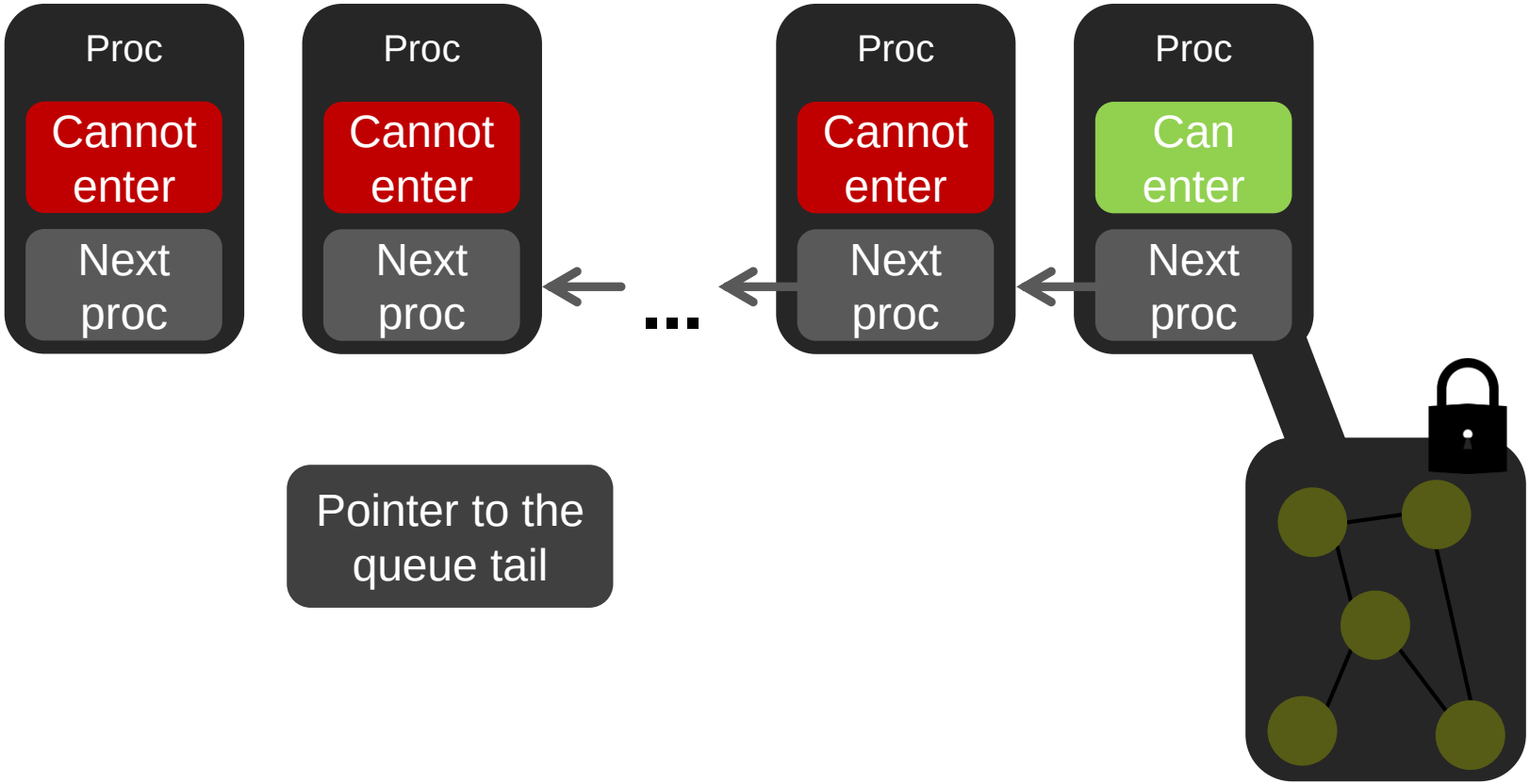
What we will use
MCS Locks



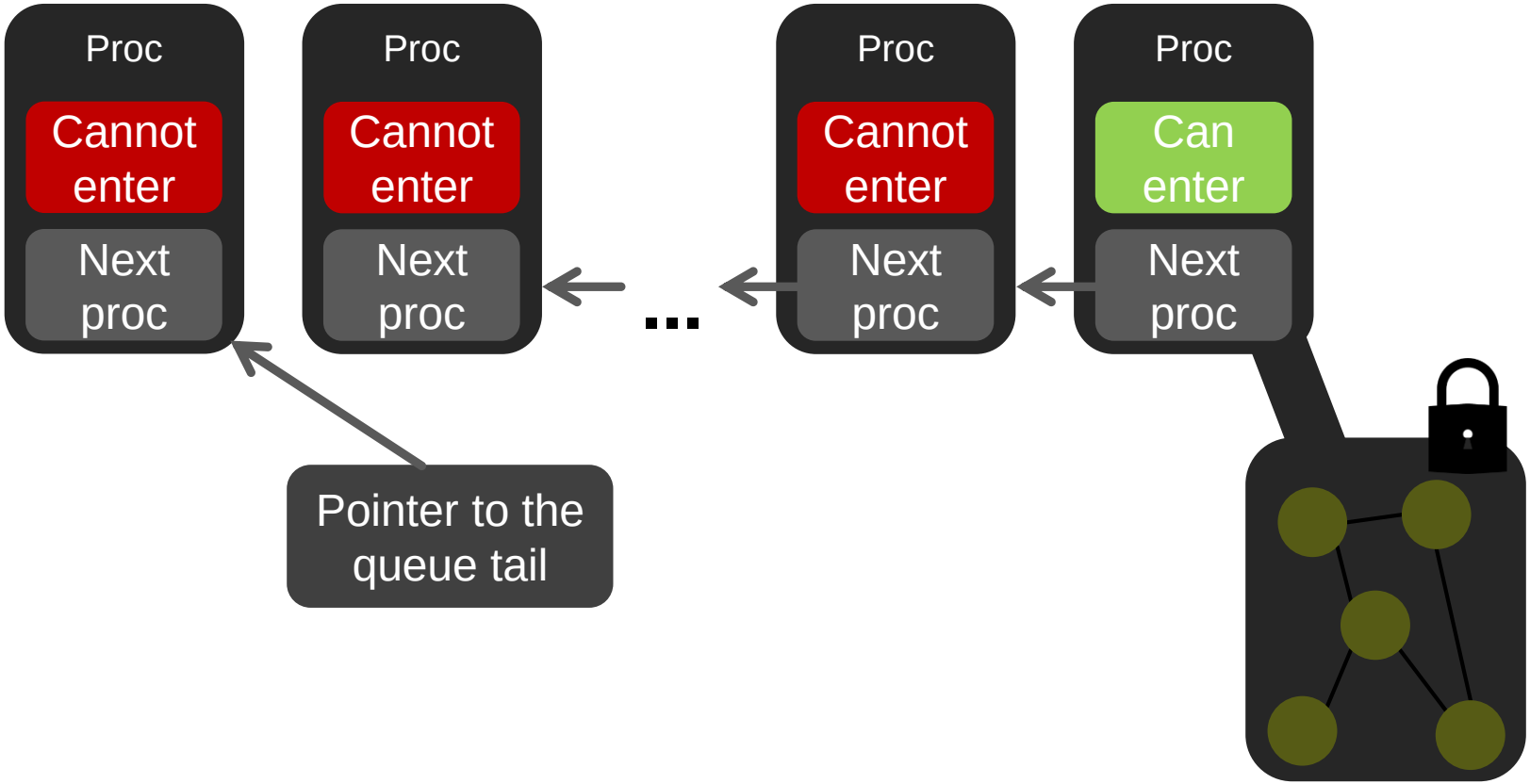
What we will use
MCS Locks



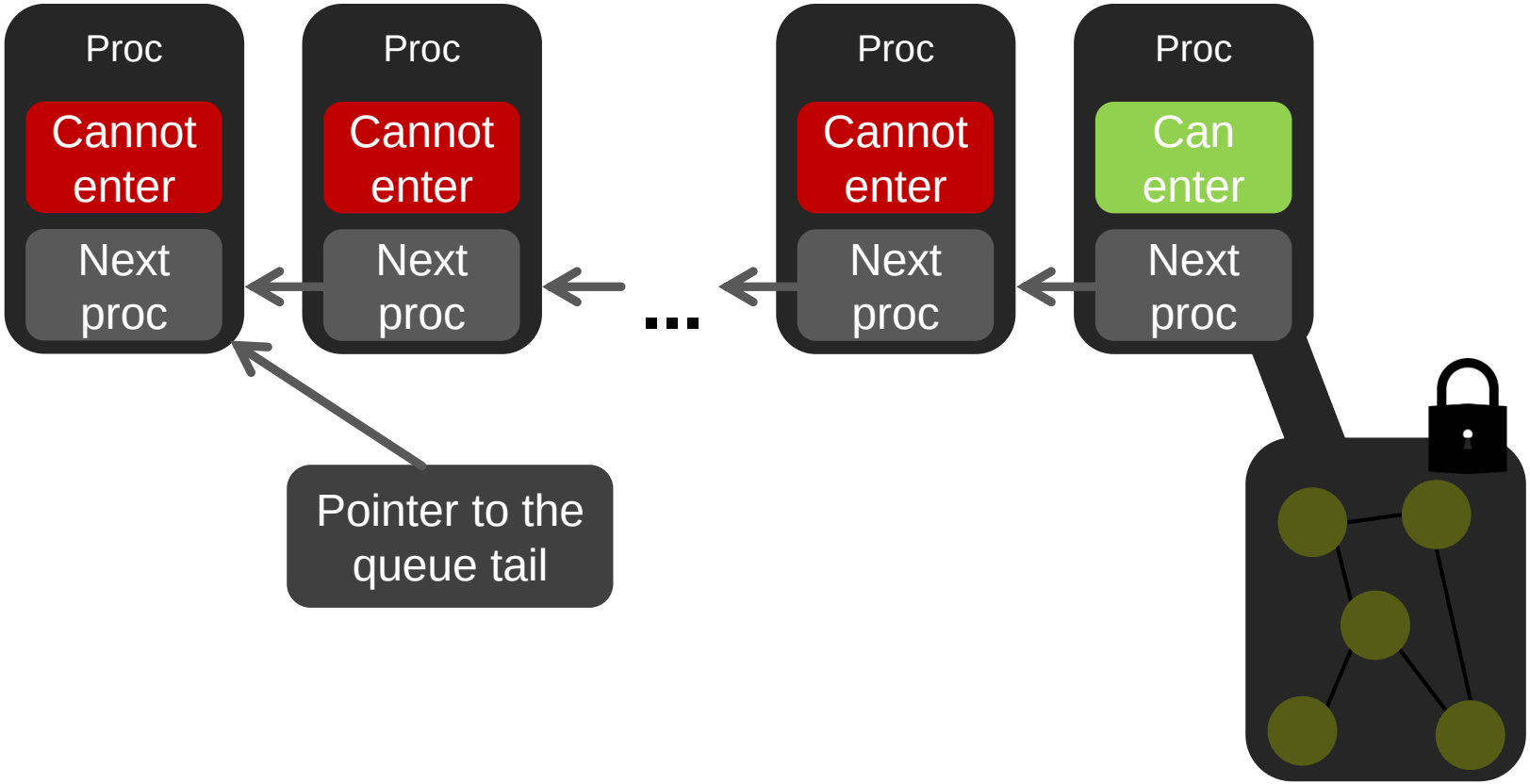
WHAT WE WILL USE
MCS Locks



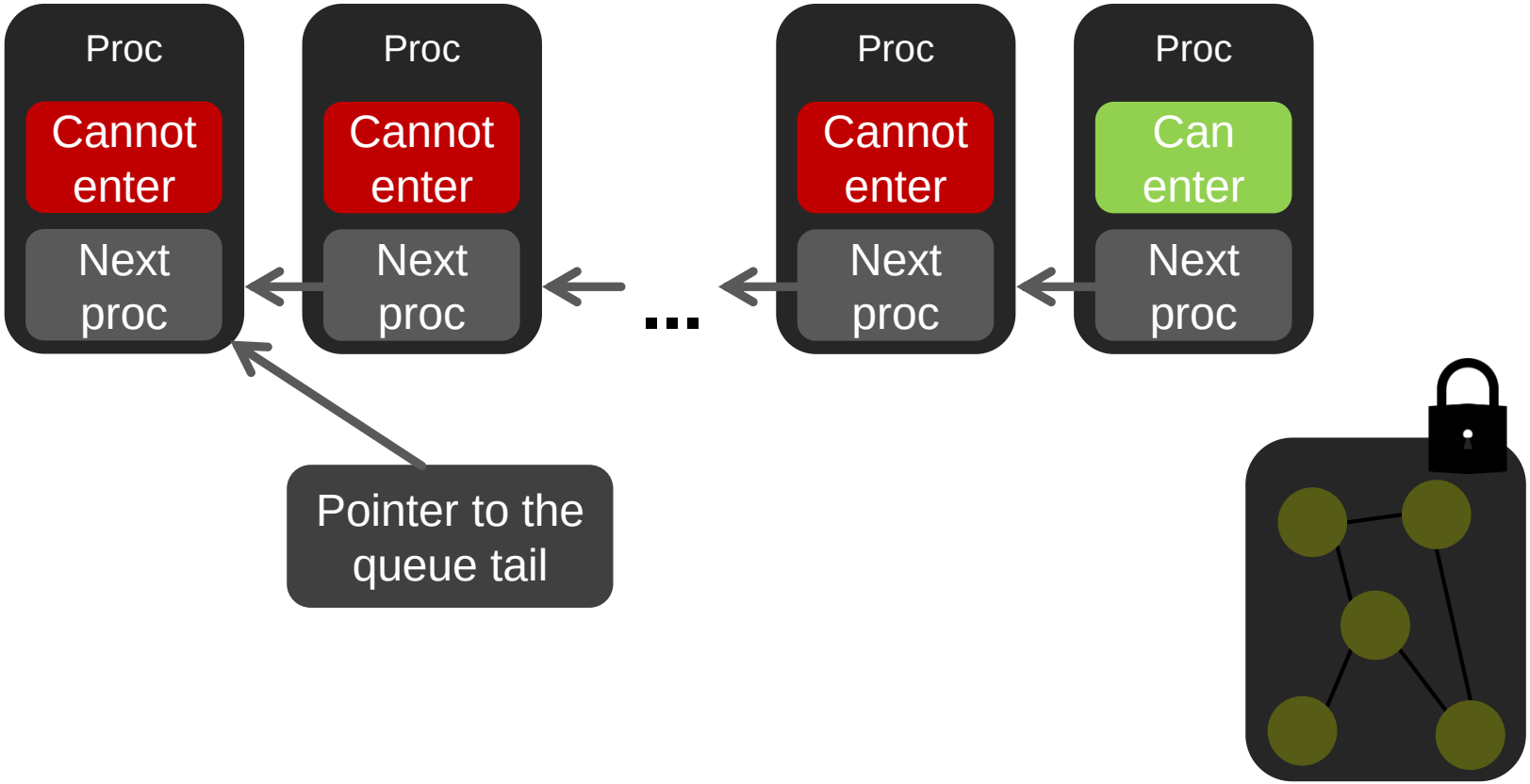
WHAT WE WILL USE
MCS Locks



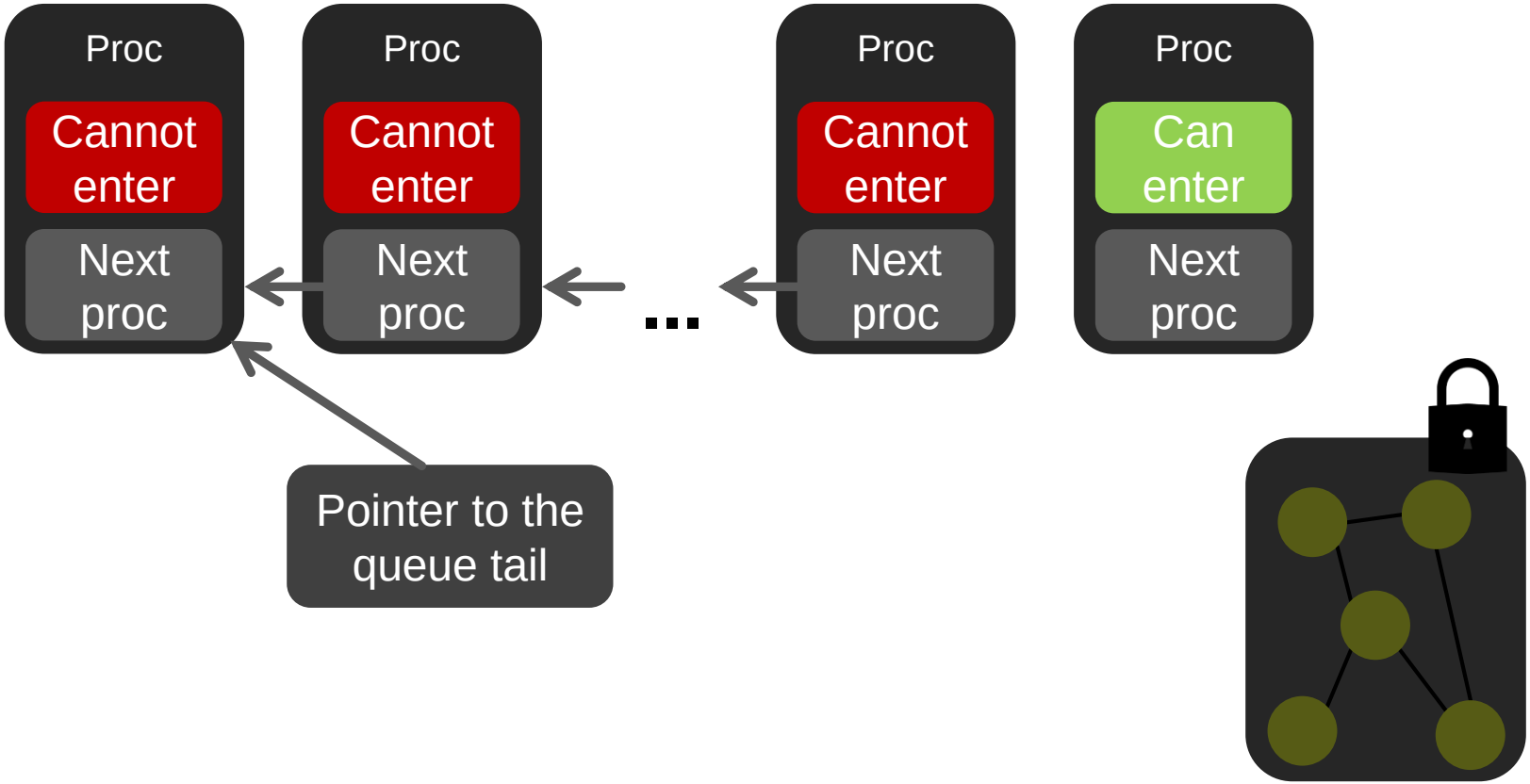
What we will use
MCS Locks



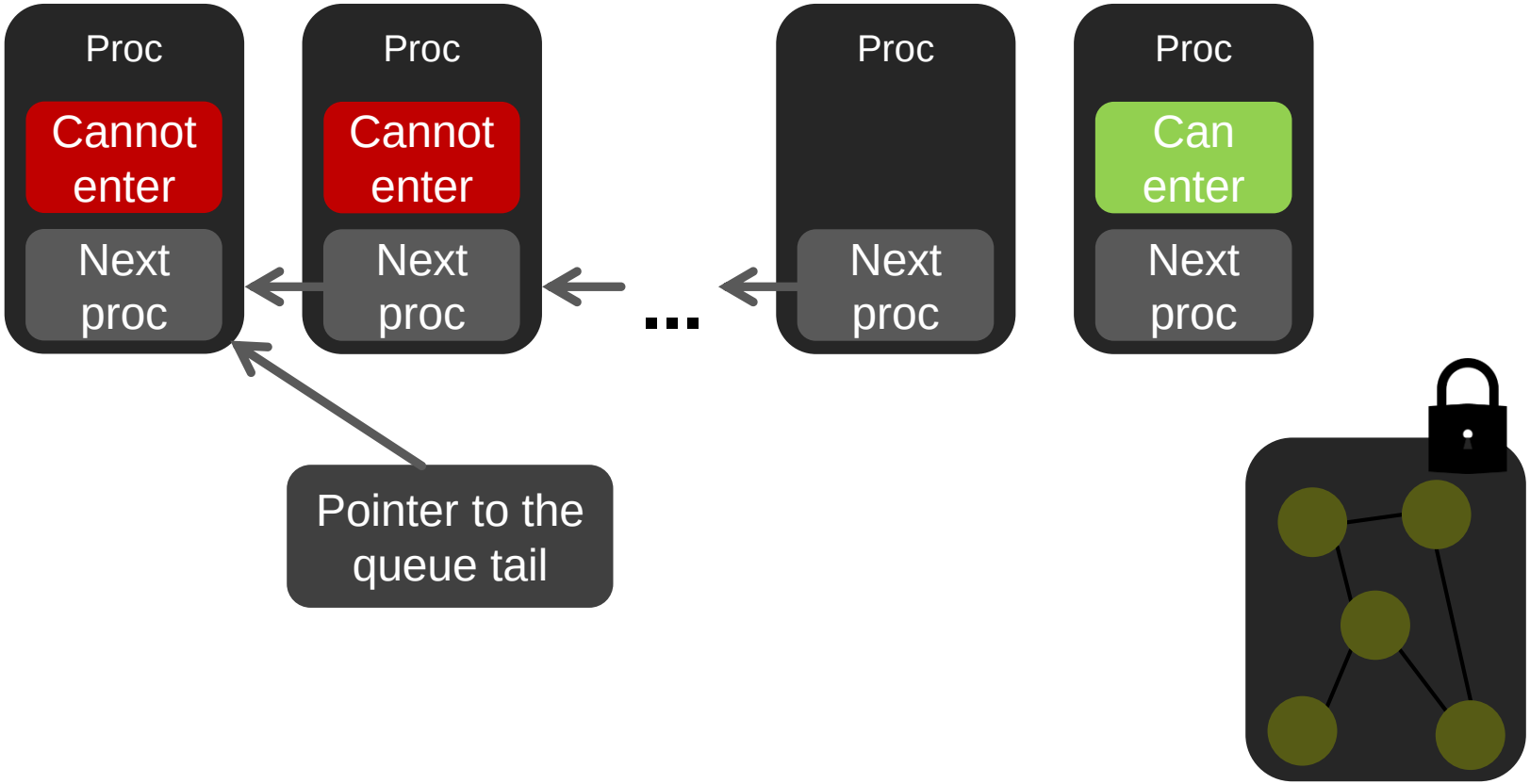
WHAT WE WILL USE
MCS Locks



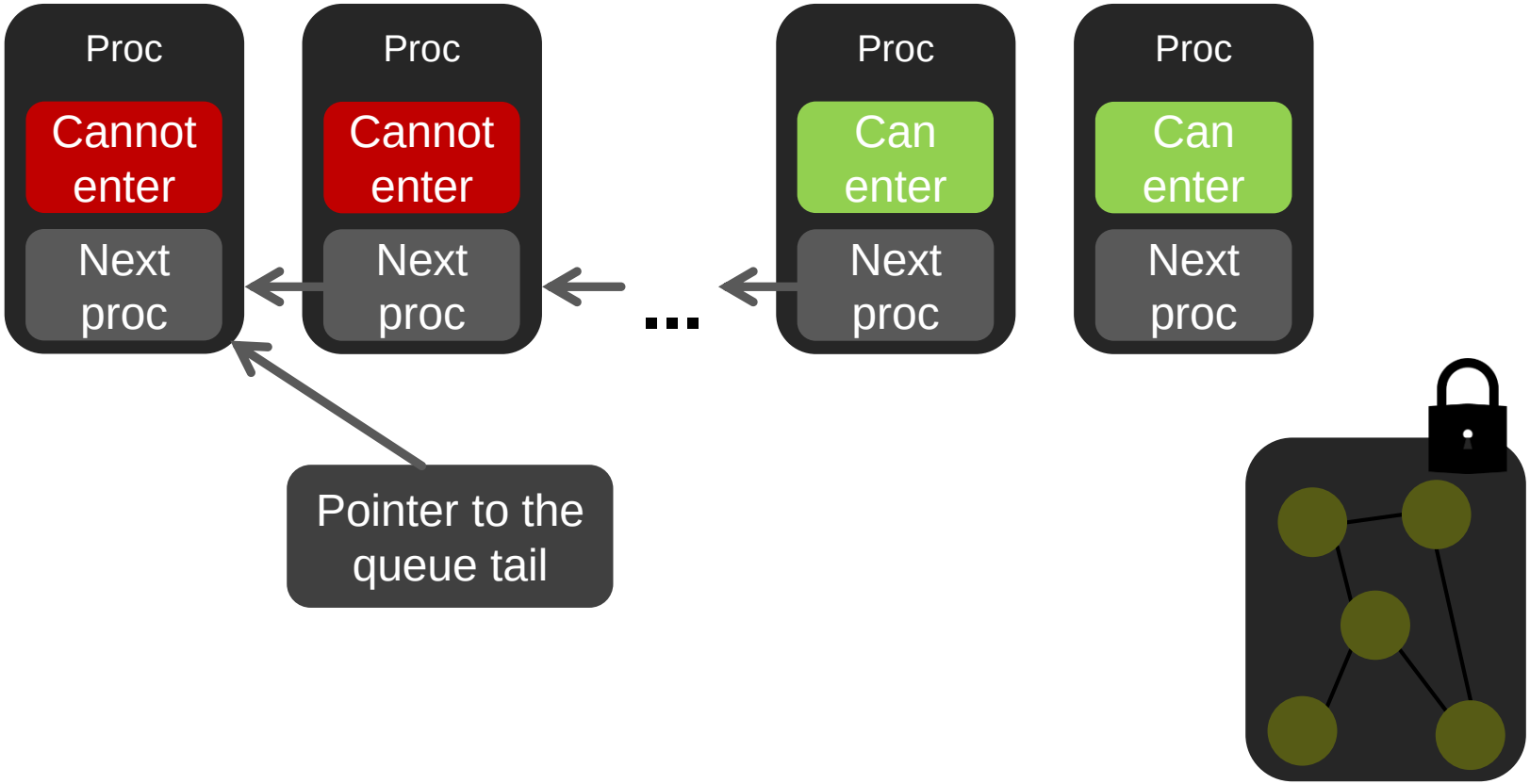
WHAT WE WILL USE
MCS Locks



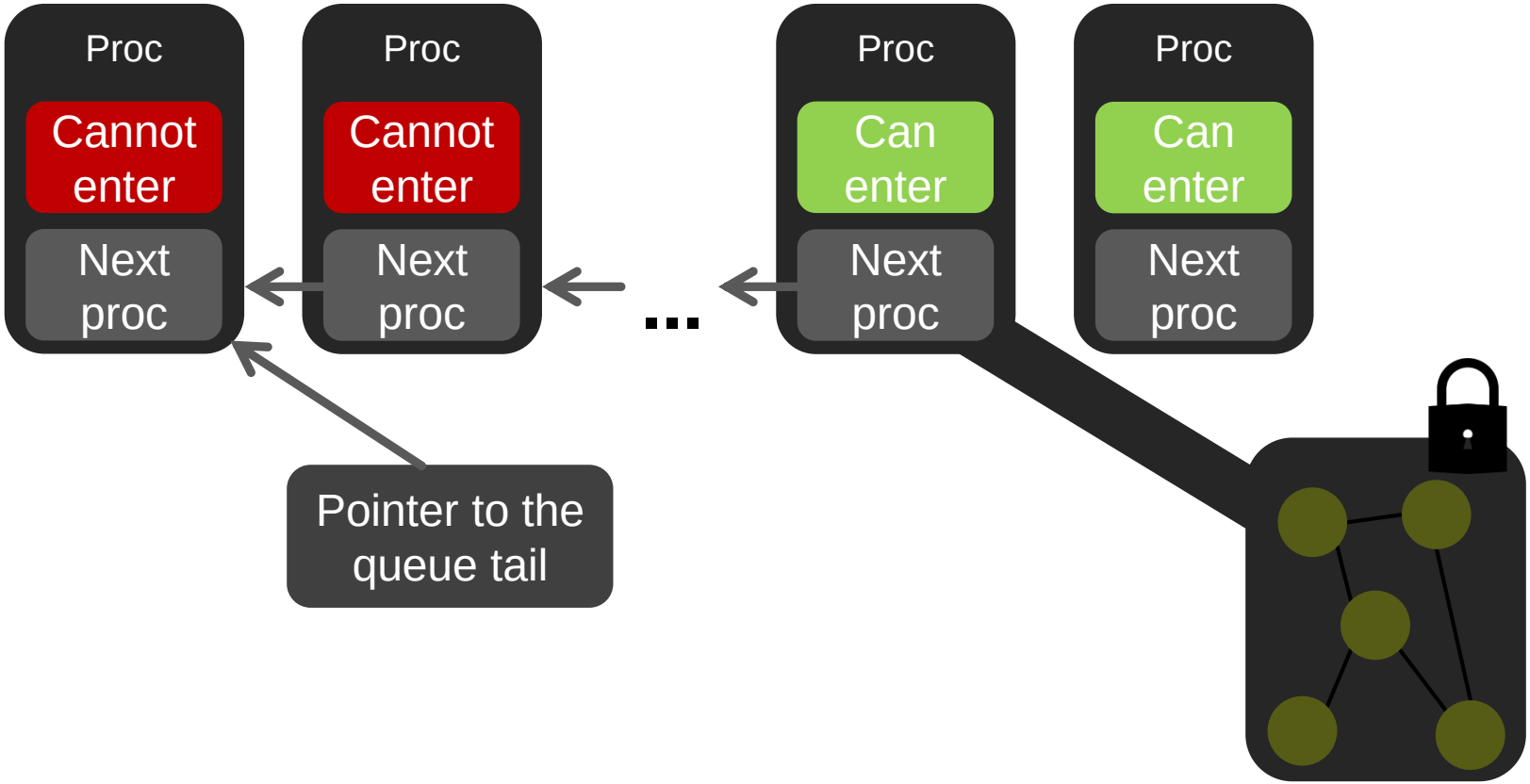
WHAT WE WILL USE
MCS Locks



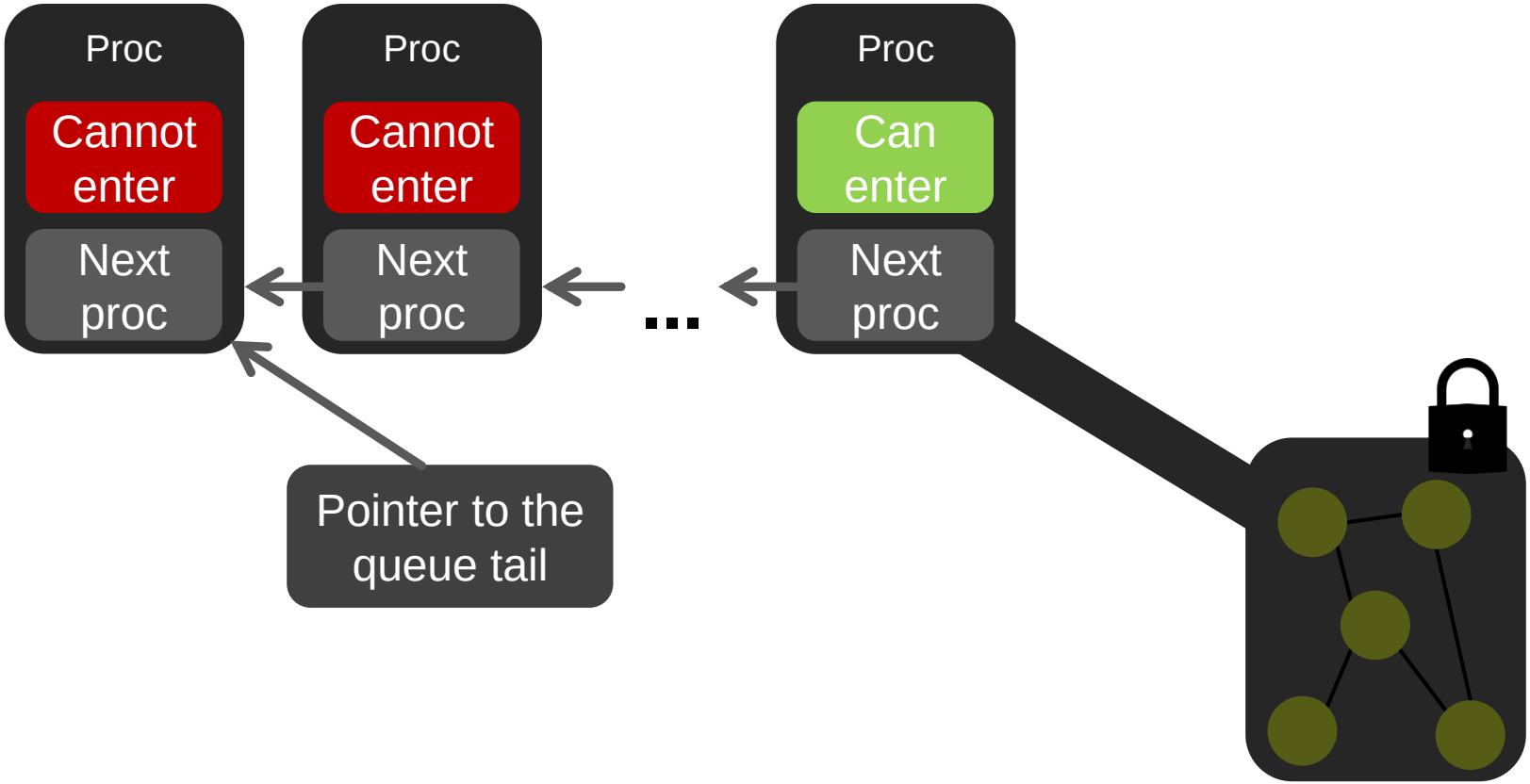
What we will use
MCS Locks



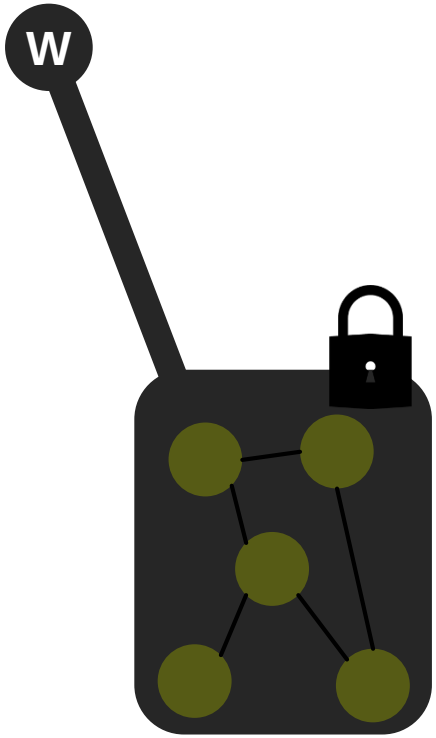
What we will use
MCS Locks



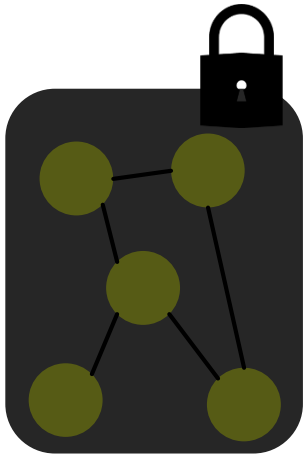
WHAT WE WILL USE
MCS Locks



What we will use
Reader-Writer Locks



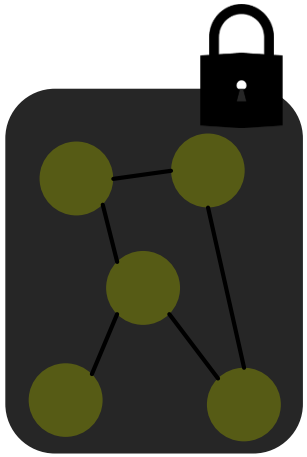
WHAT WE WILL USE
Reader-Writer Locks



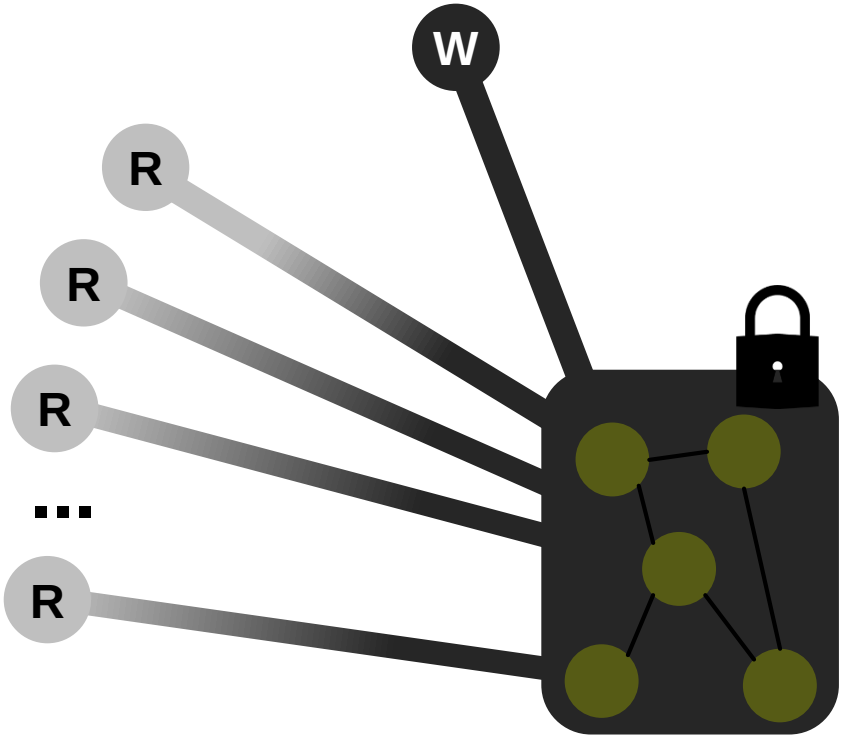
WHAT WE WILL USE
Reader-Writer Locks

R

...



What we will use
Reader-Writer Locks



How to manage the design complexity?

How to manage the design complexity?

What mechanism to use for efficient implementation?

How to manage the design complexity?

How to ensure tunable performance?

What mechanism to use for efficient implementation?

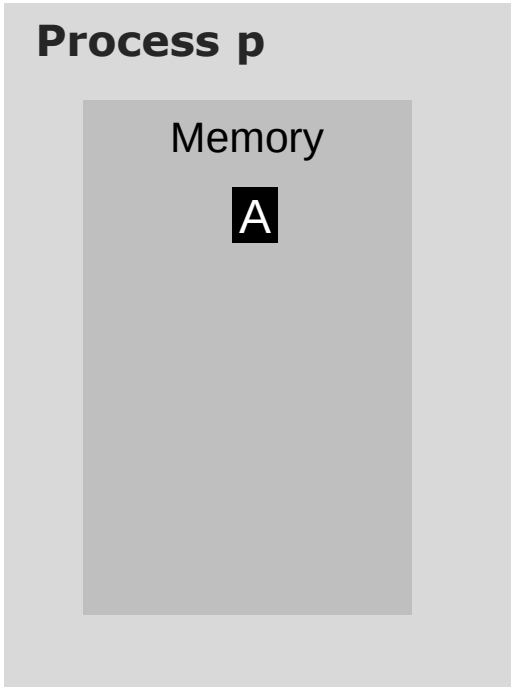
How to manage the design complexity?

How to ensure tunable performance?

What mechanism to use for efficient implementation?

Remote Memory Access (RMA) Programming

Remote Memory Access (RMA) Programming



Remote Memory Access (RMA) Programming

Remote Memory Access (RMA) Programming

Process p

Memory

A

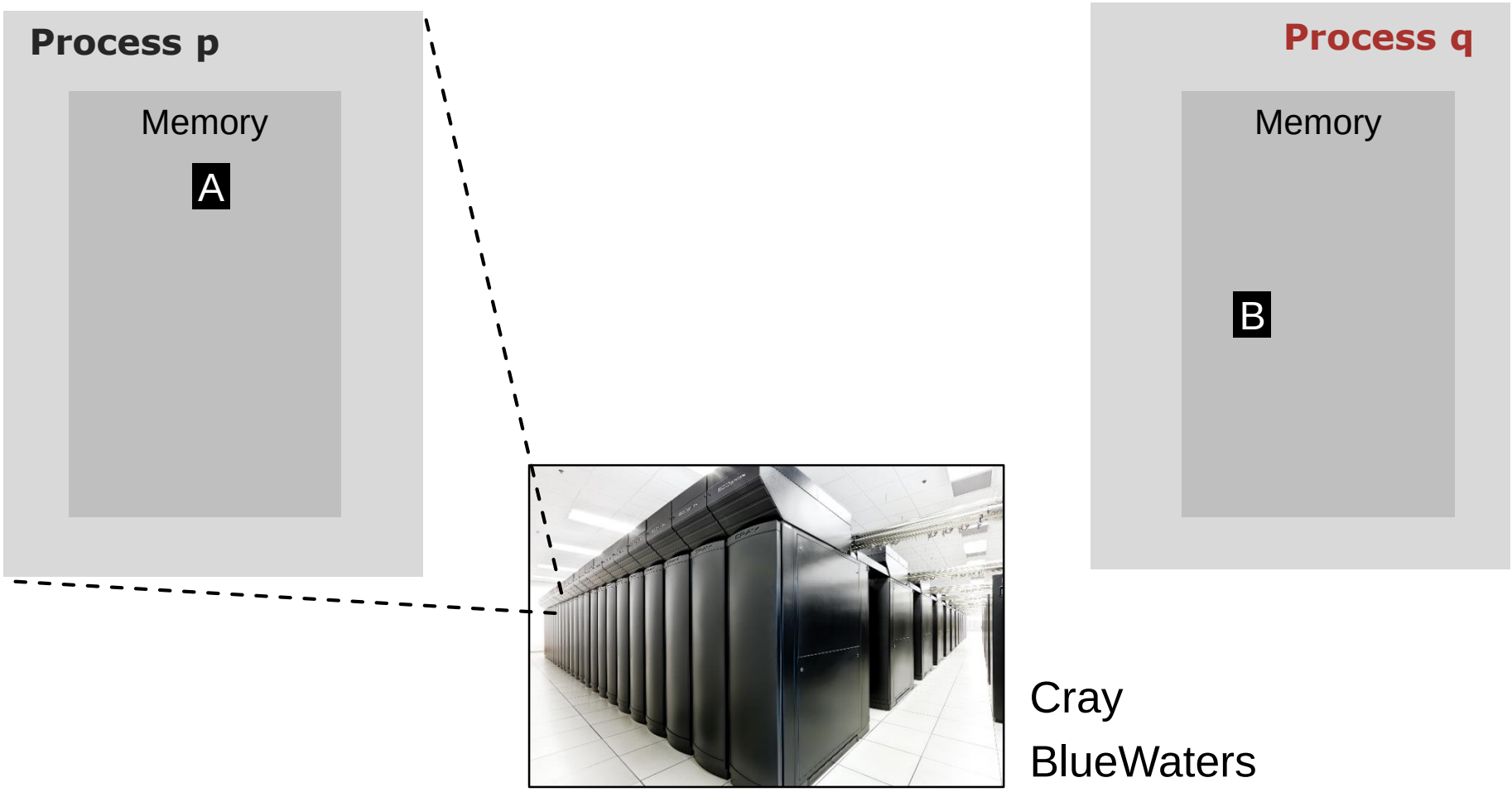
Process q

Memory

B

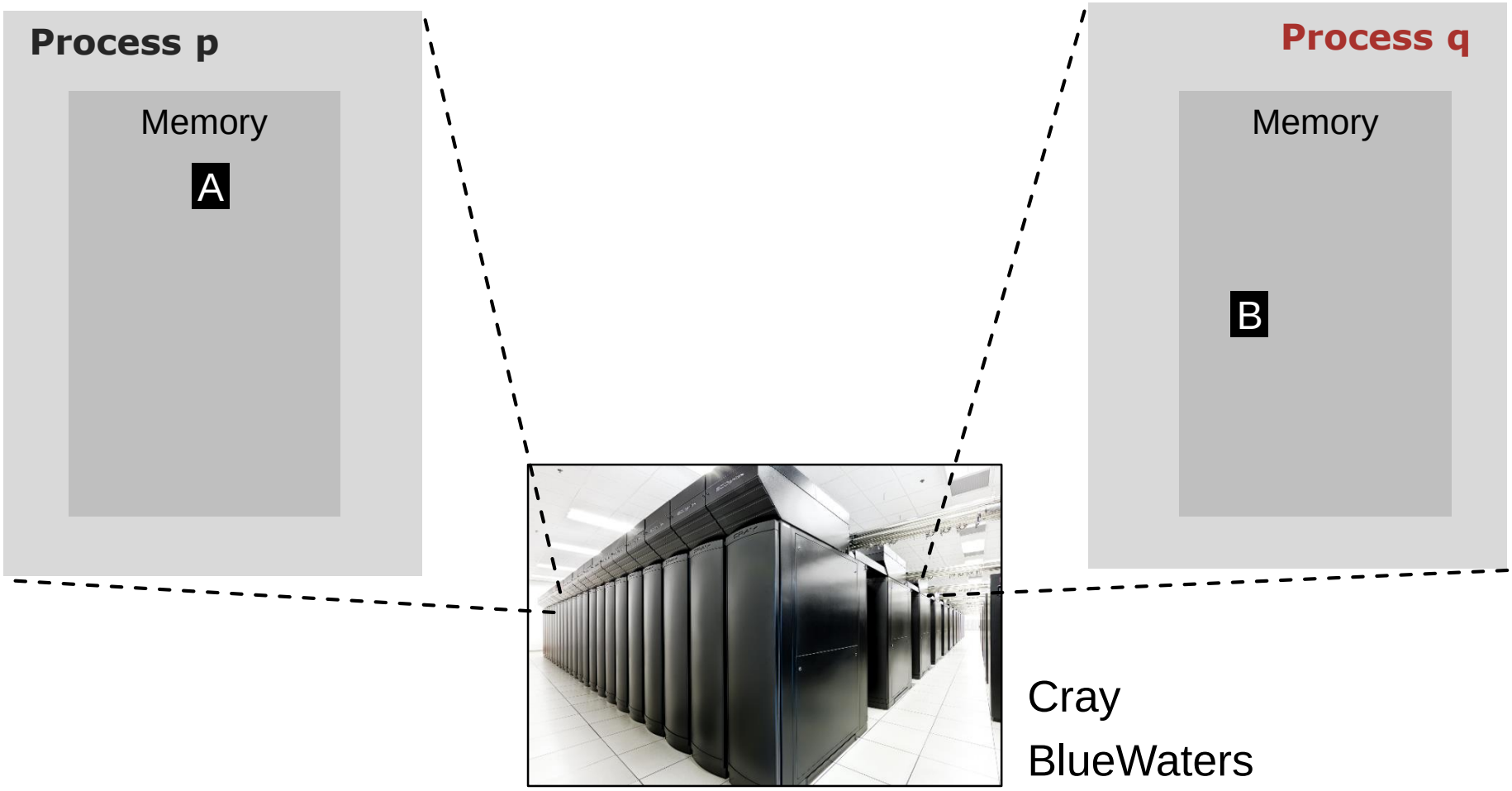
Cray BlueWaters

Remote Memory Access (RMA) Programming



Cray BlueWaters

Remote Memory Access (RMA) Programming



Remote Memory Access (RMA) Programming

Process p

Memory

A

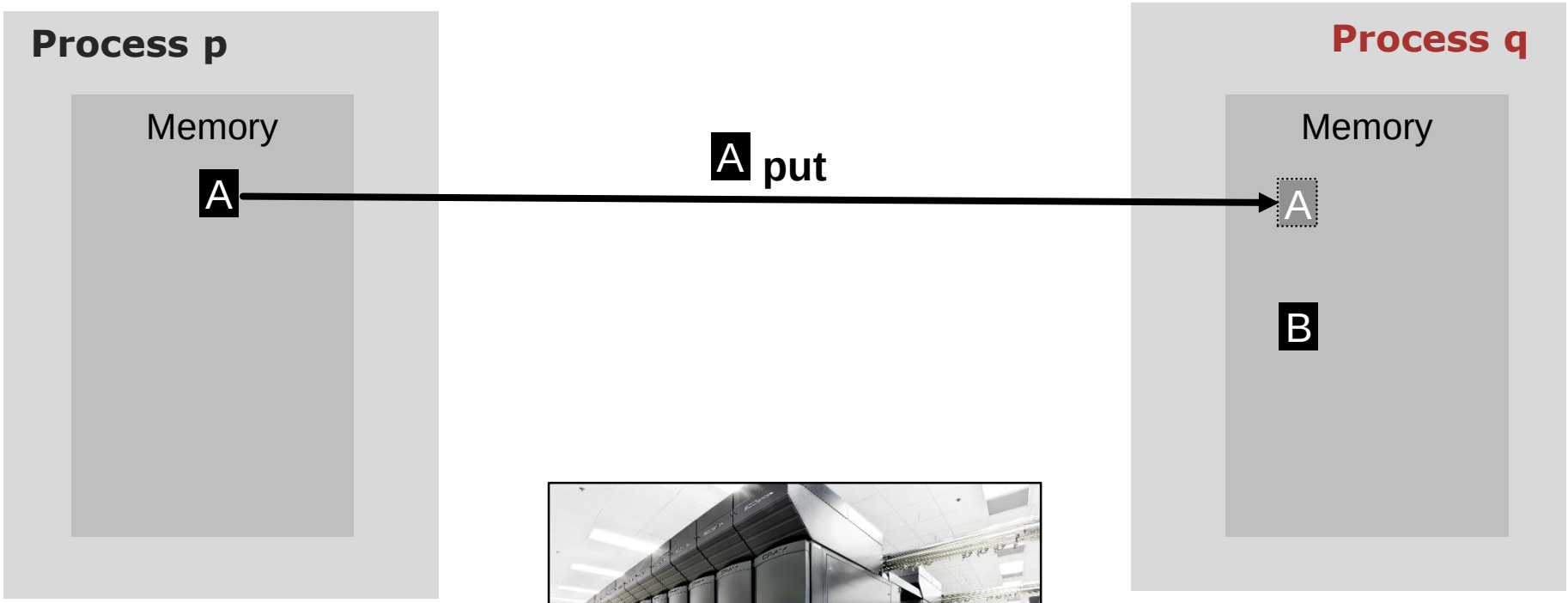
Process q

Memory

B

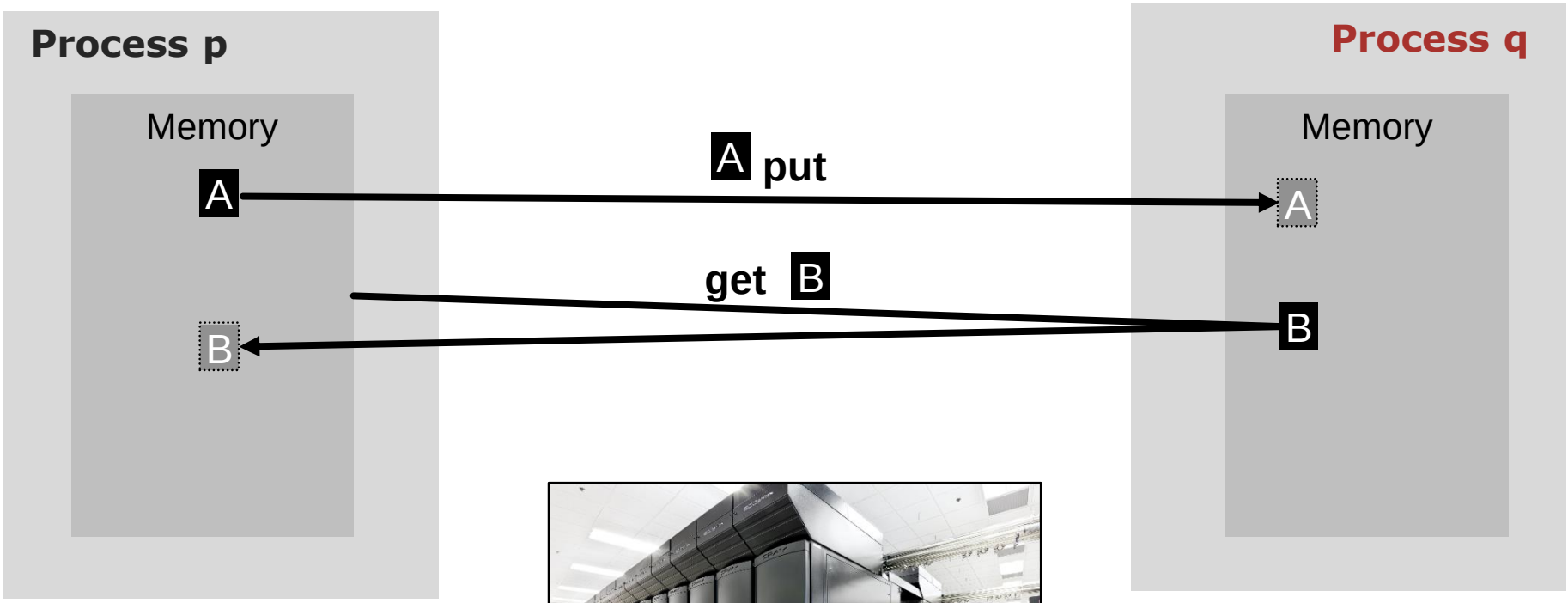
Cray BlueWaters

Remote Memory Access (RMA) Programming



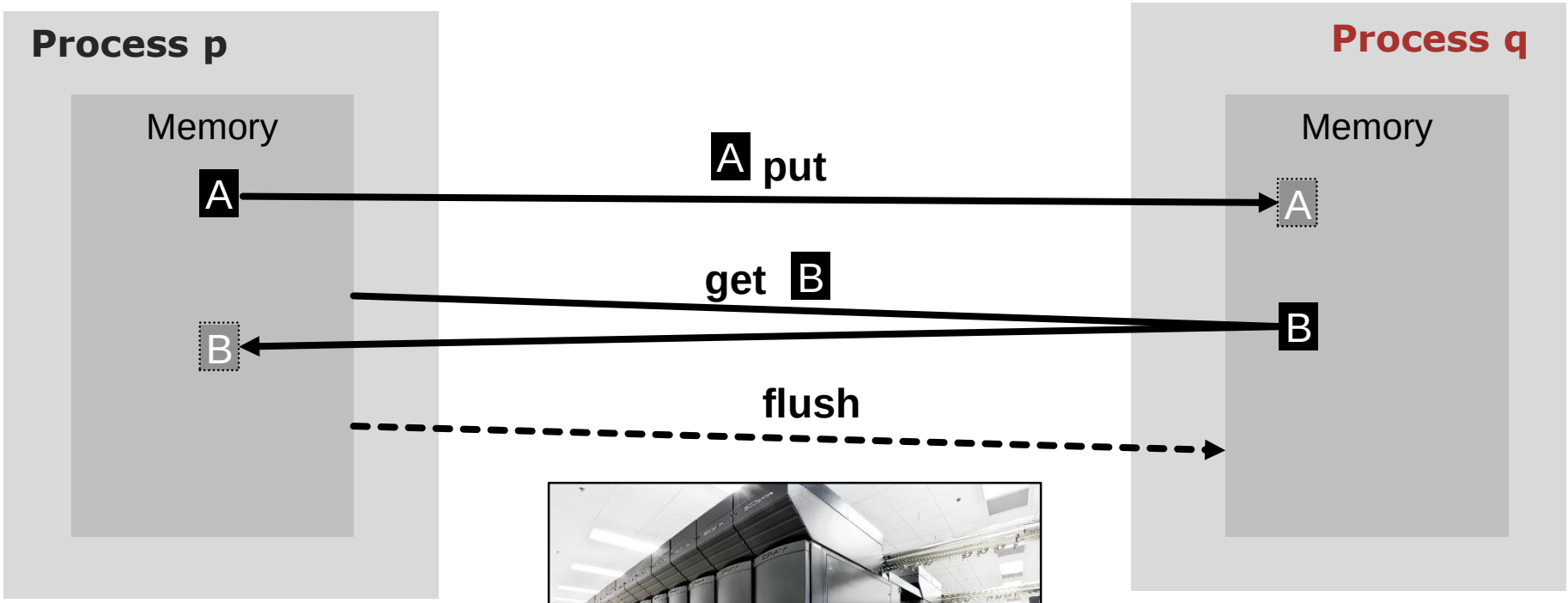
Cray BlueWaters

Remote Memory Access (RMA) Programming



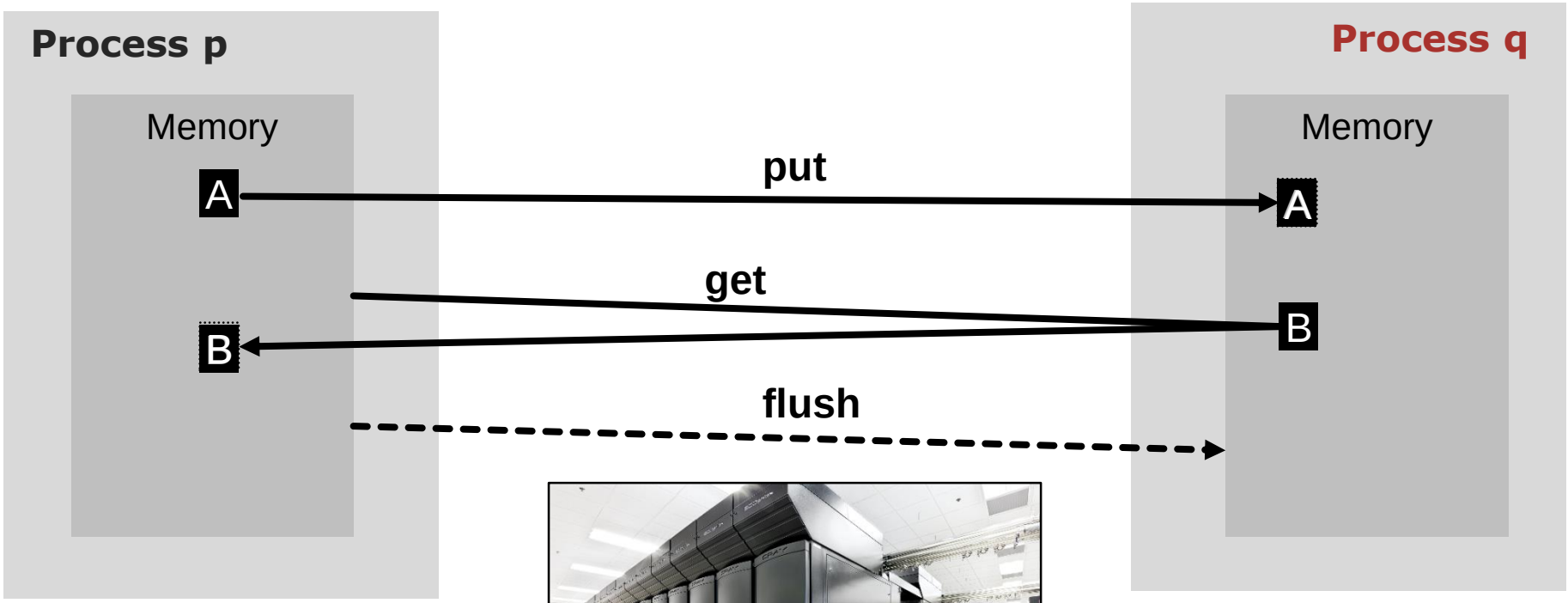
Cray BlueWaters

Remote Memory Access (RMA) Programming



Cray BlueWaters

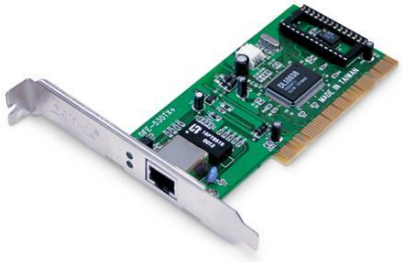
Remote Memory Access (RMA) Programming



Cray BlueWaters

Remote Memory Access Programming

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

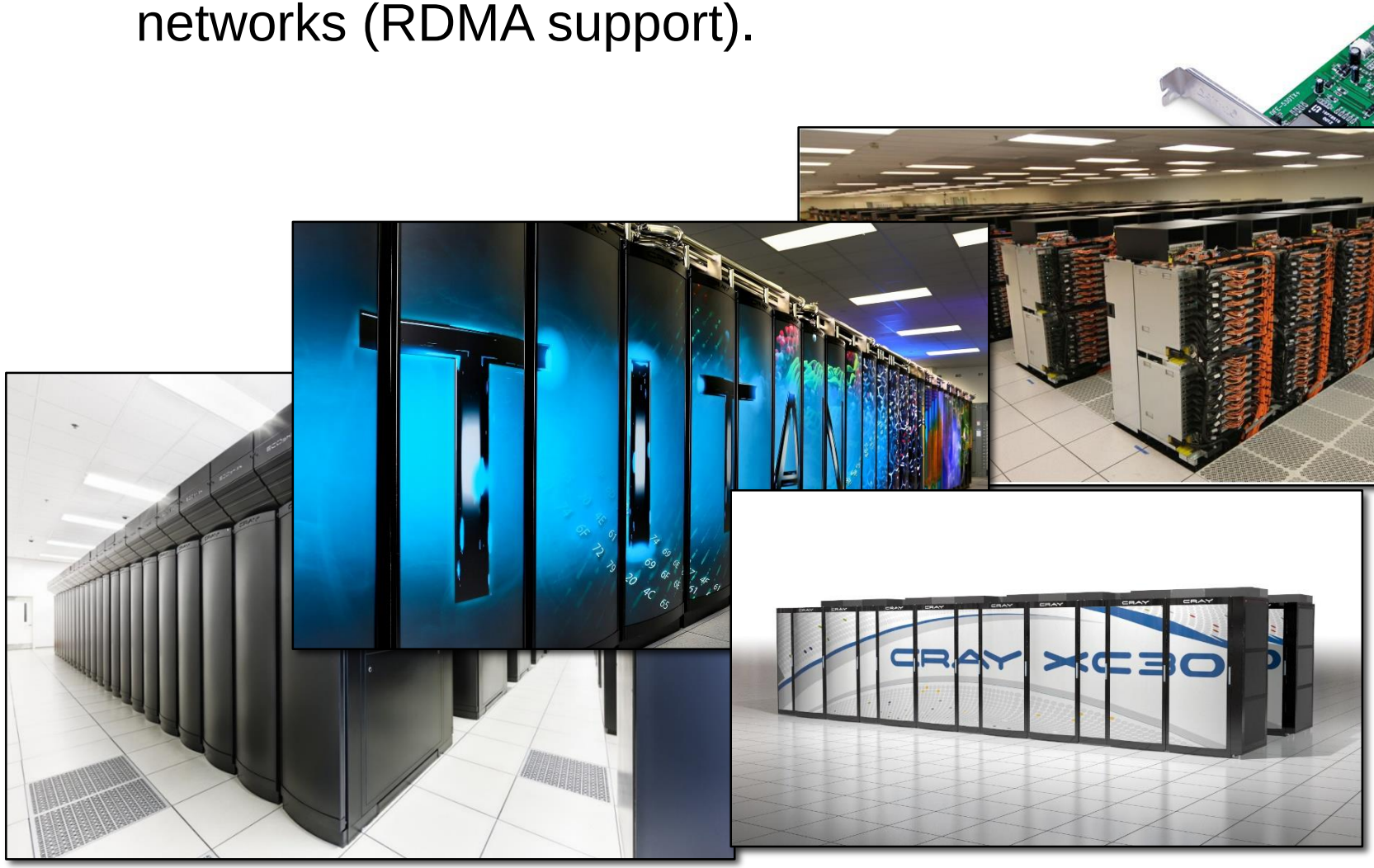


Remote Memory Access Programming

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

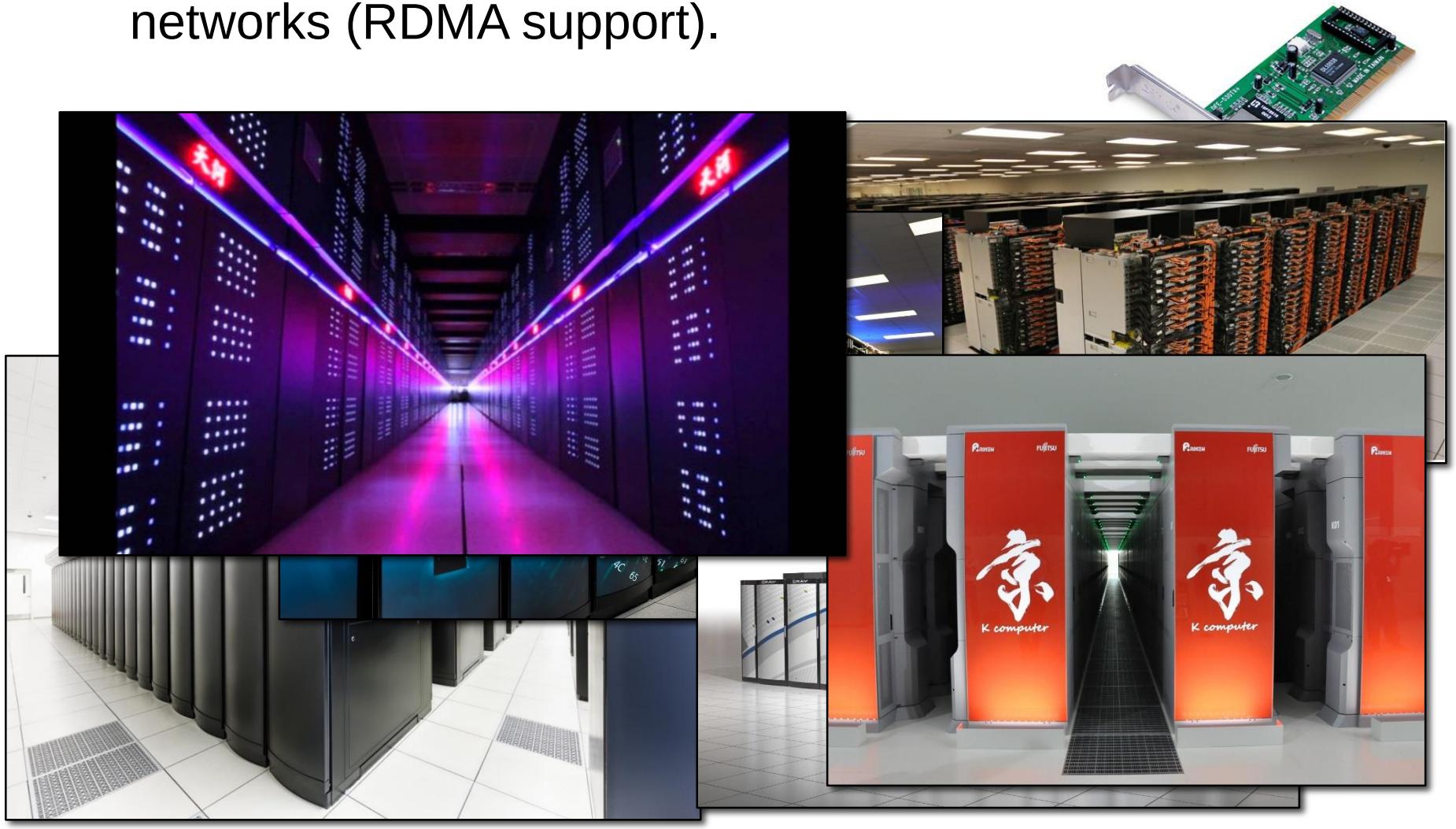
REMOTE MEMORY ACCESS PROGRAMMING

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



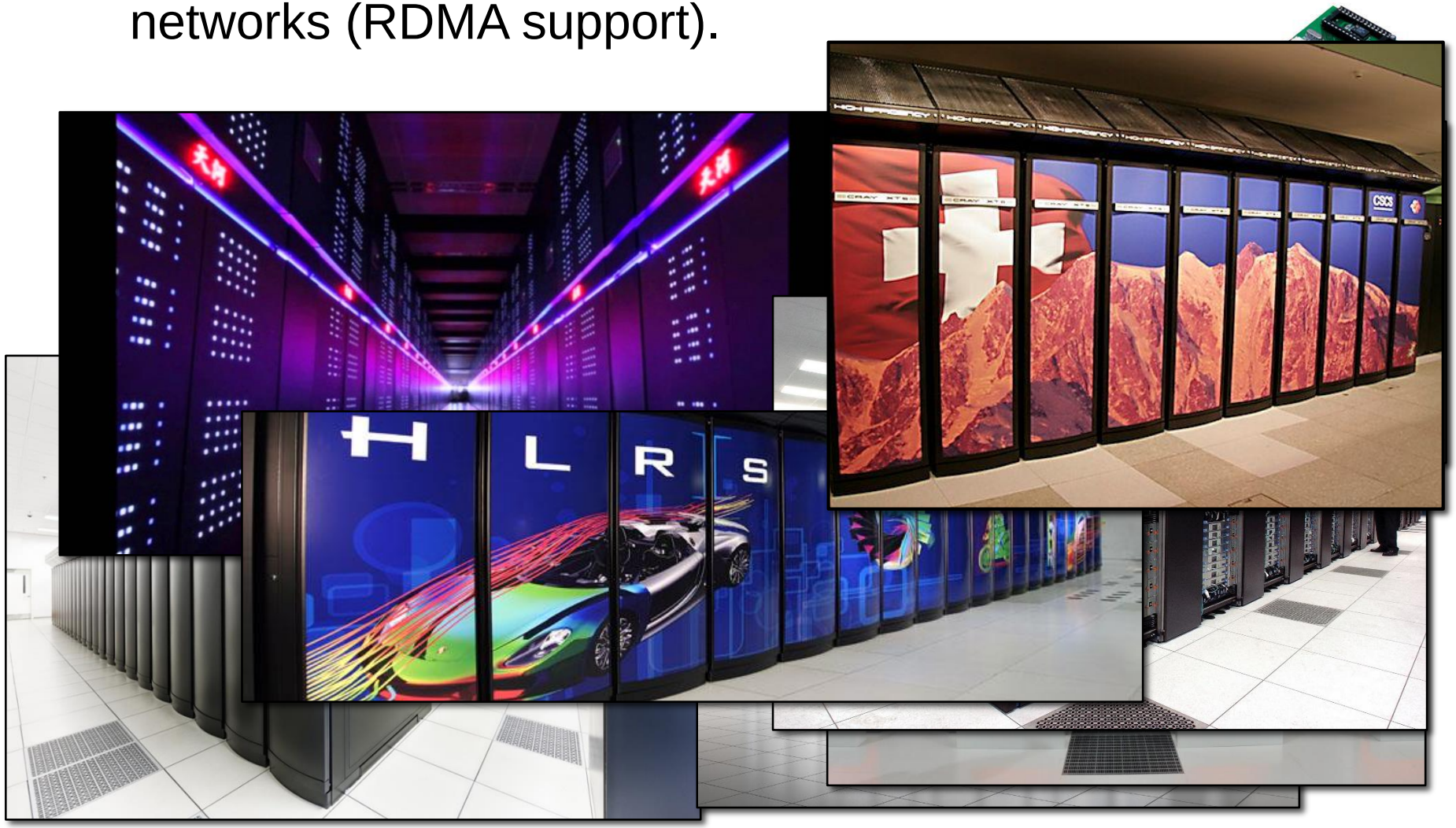
REMOTE MEMORY ACCESS PROGRAMMING

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



REMOTE MEMORY ACCESS PROGRAMMING

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



Remote Memory Access Programming

- Supported by many HPC libraries and languages

REMOTE MEMORY ACCESS PROGRAMMING

- Supported by many HPC libraries and languages

REMOTE MEMORY ACCESS PROGRAMMING

- Supported by many HPC libraries and languages

How to manage the design complexity?

How to ensure tunable performance?

What mechanism to use for efficient implementation?

How to manage the design complexity?

How to ensure tunable performance?

What mechanism to use for efficient implementation?

How to manage the design complexity?

How to ensure tunable performance?

What mechanism to use for efficient implementation?

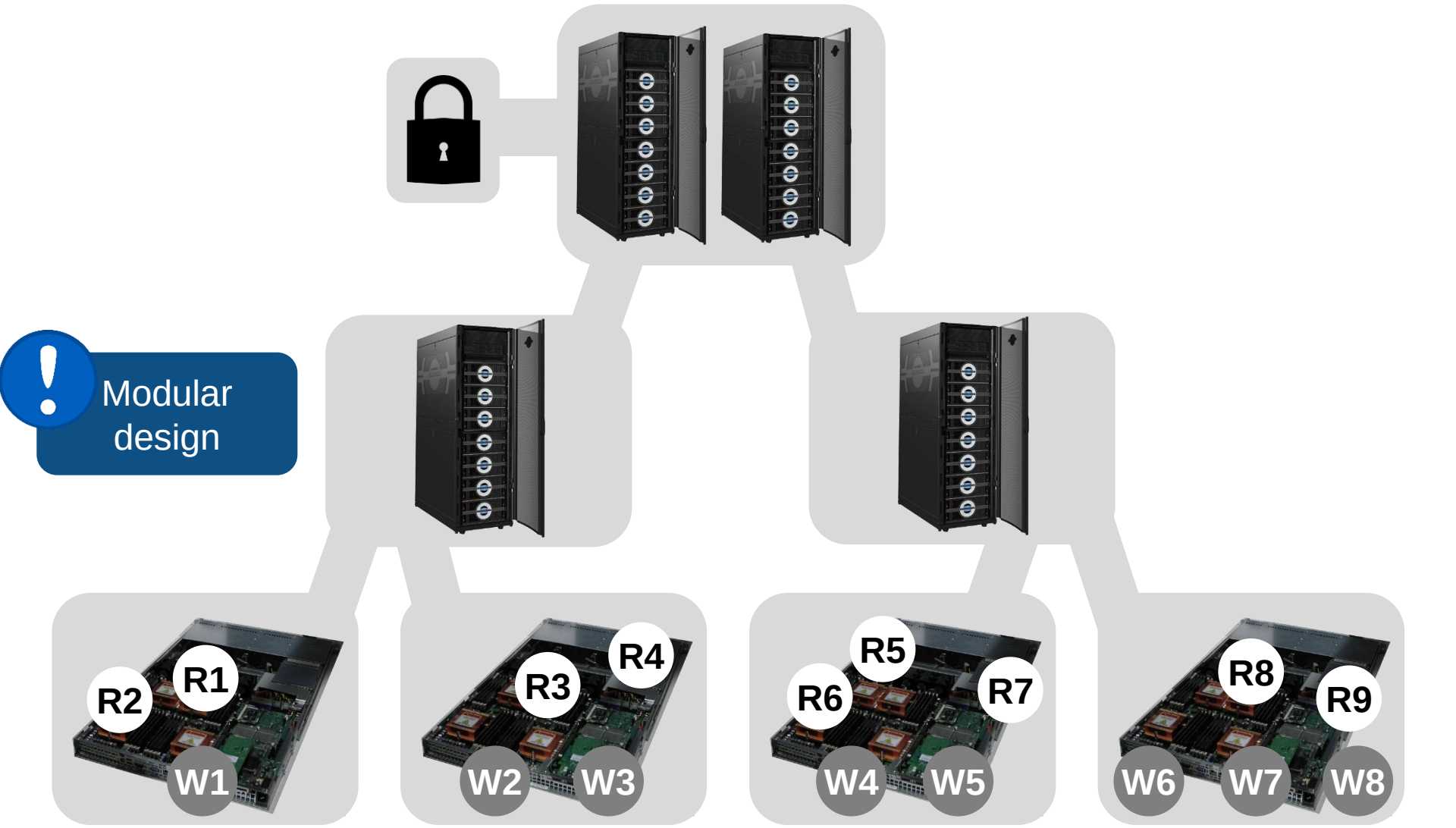
How to manage the design complexity?

How to manage the design complexity?

How to manage the design complexity?

How to manage the design complexity?

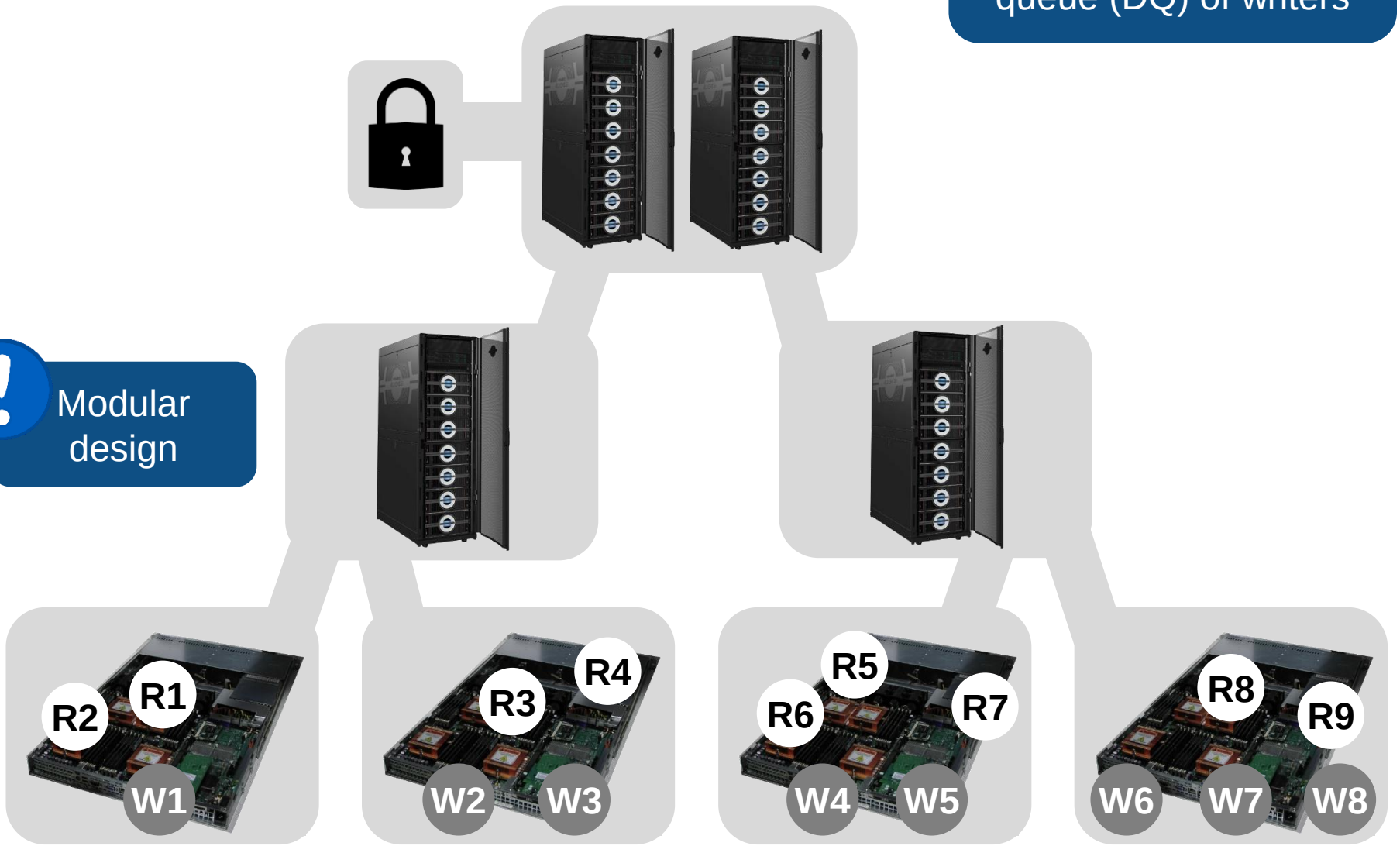
How to manage the design complexity?



How to manage the design complexity?

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

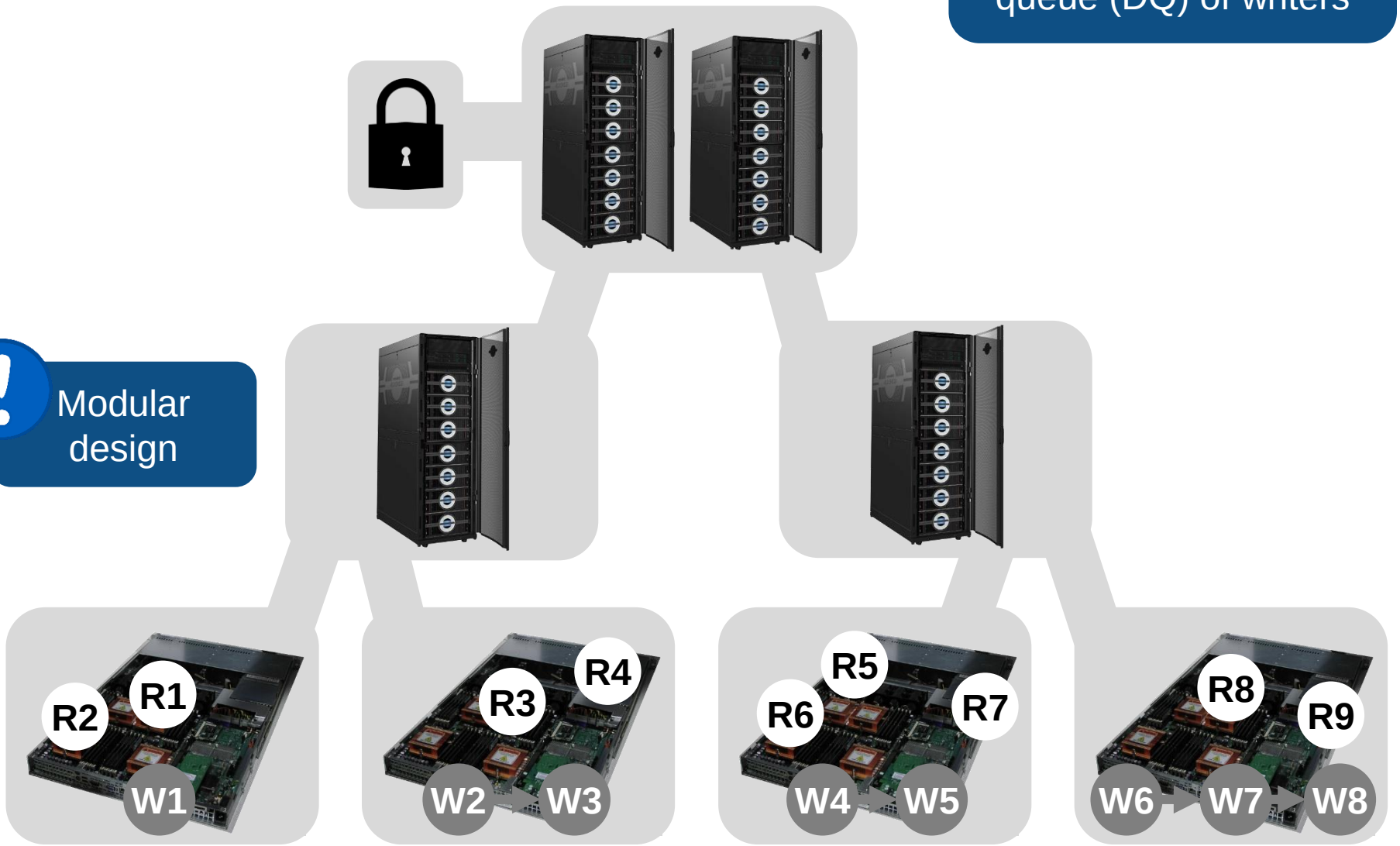
Modular design



How to manage the design complexity?

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

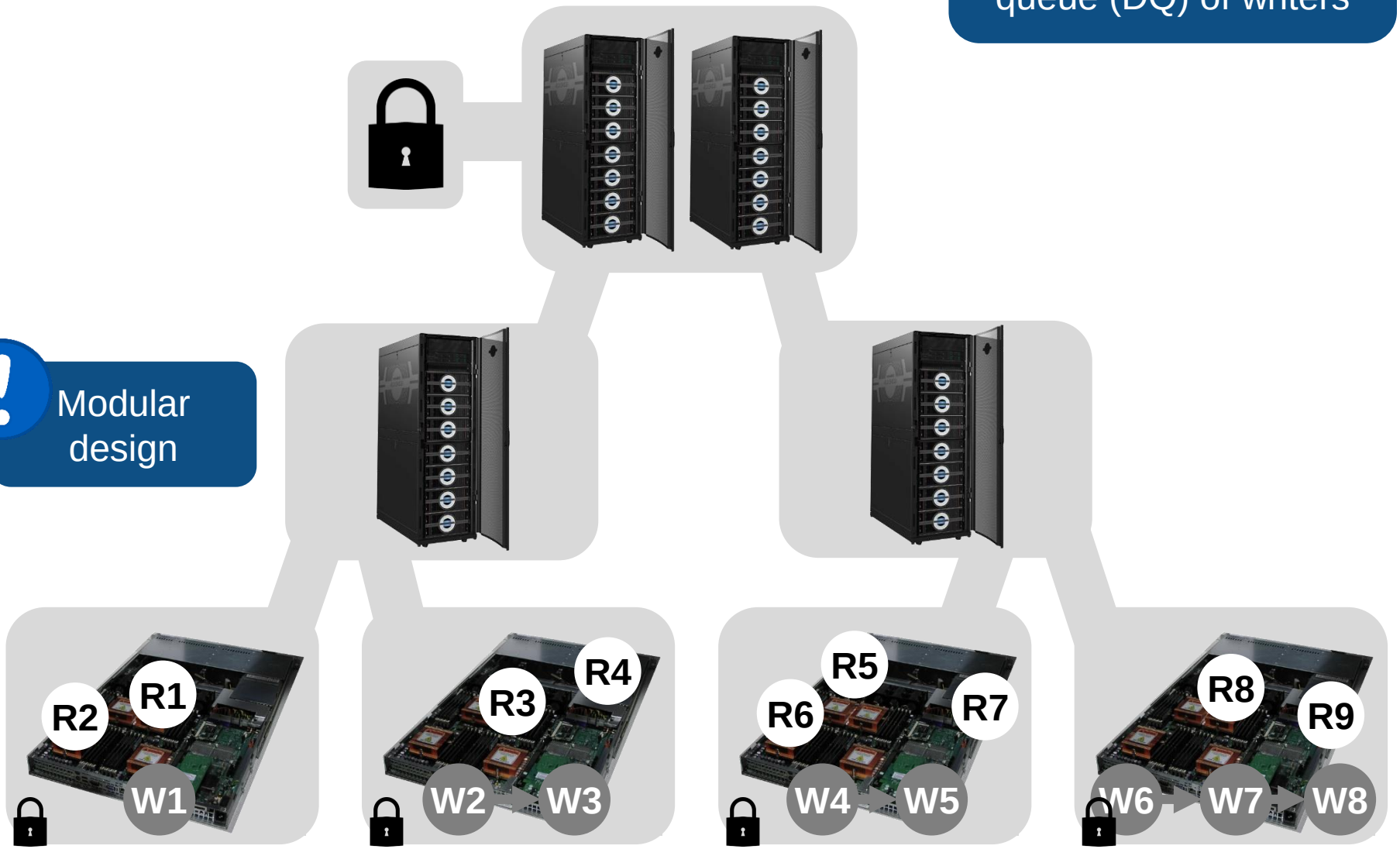
Modular design



How to manage the design complexity?

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

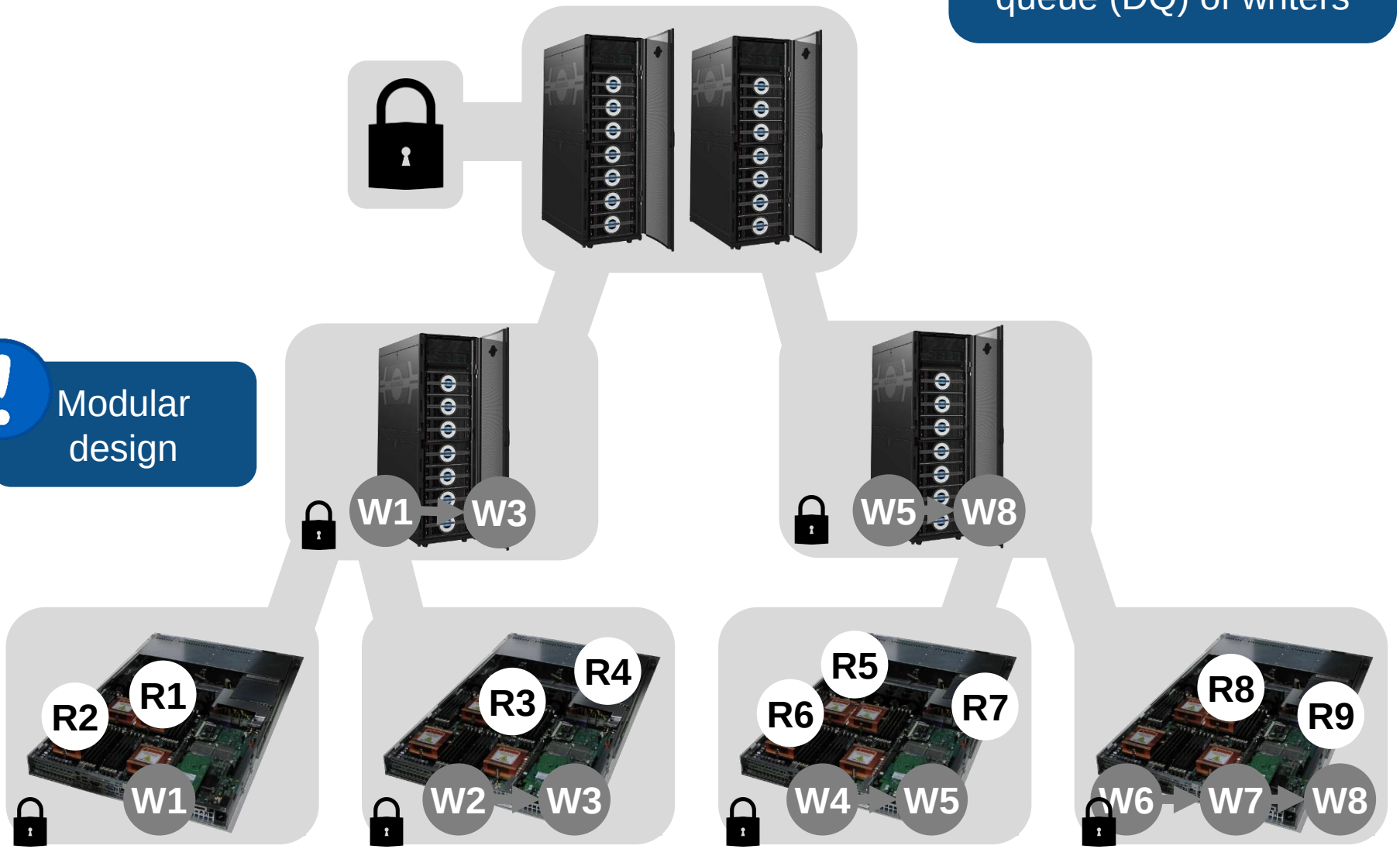
Modular design



How to manage the design complexity?

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

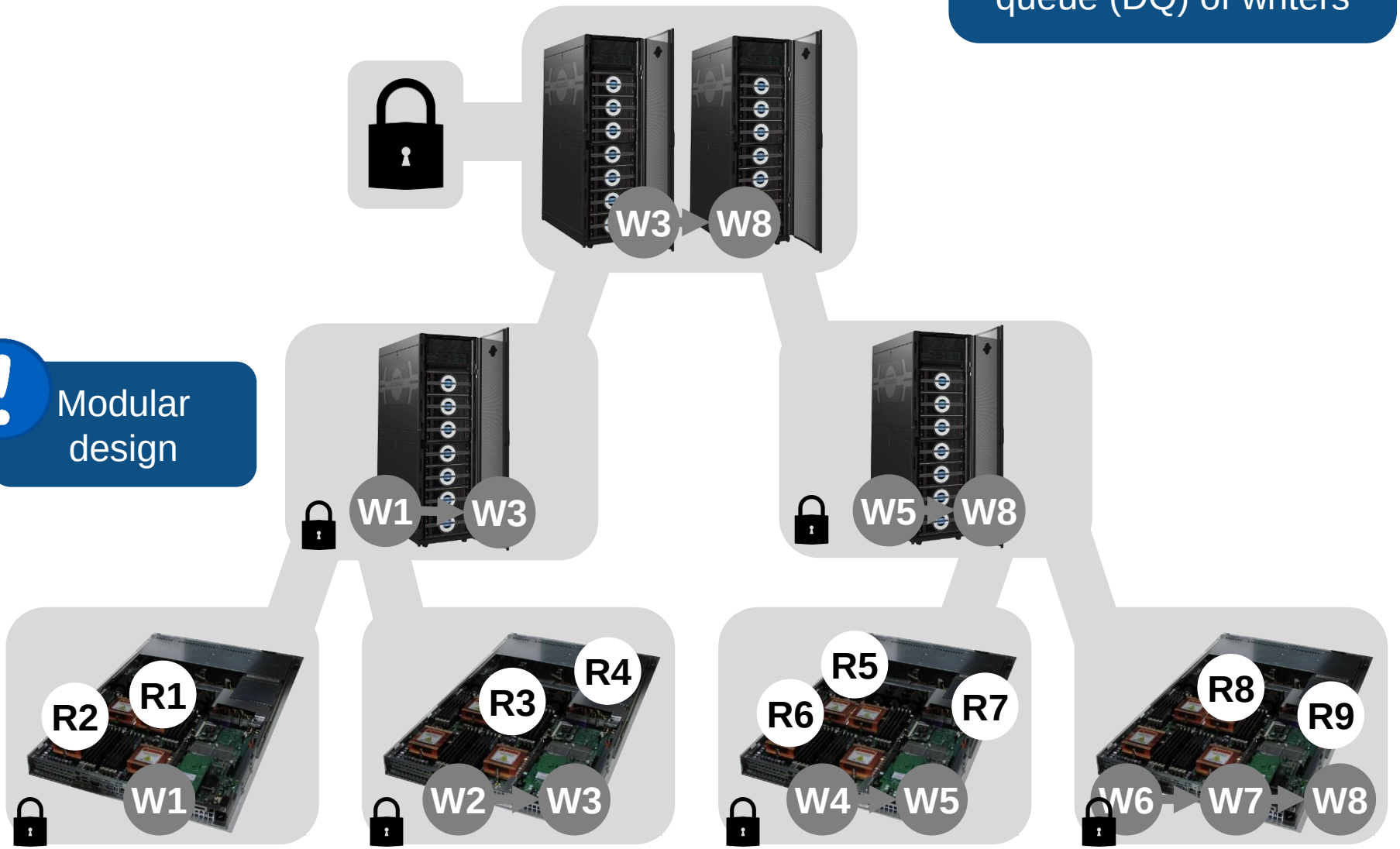
Modular design



How to manage the design complexity?

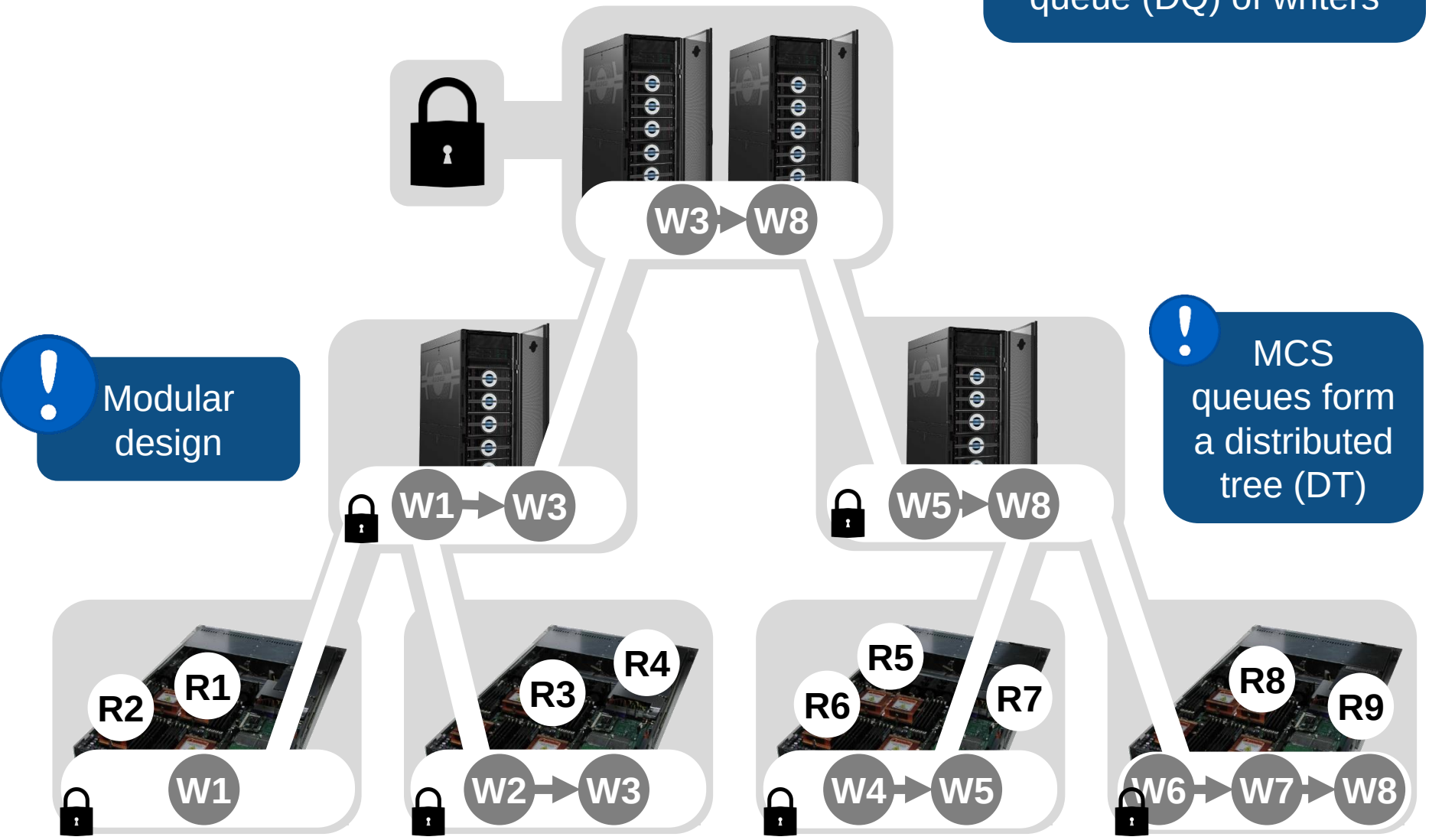
Each element has its own distributed MCS queue (DQ) of writers

Modular design



How to manage the design complexity?

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

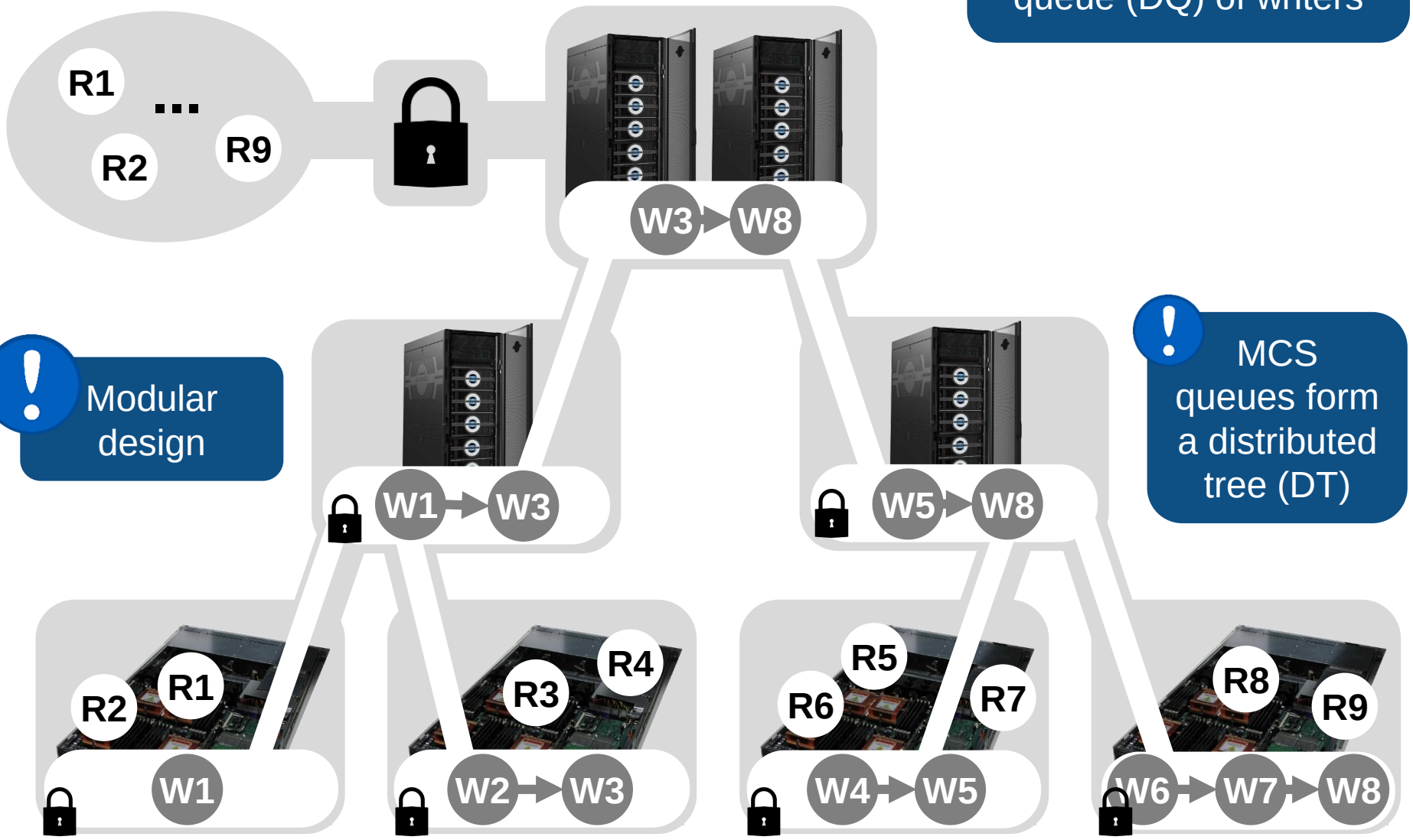


Modular design

MCS queues form a distributed tree (DT)

How to manage the design complexity?

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



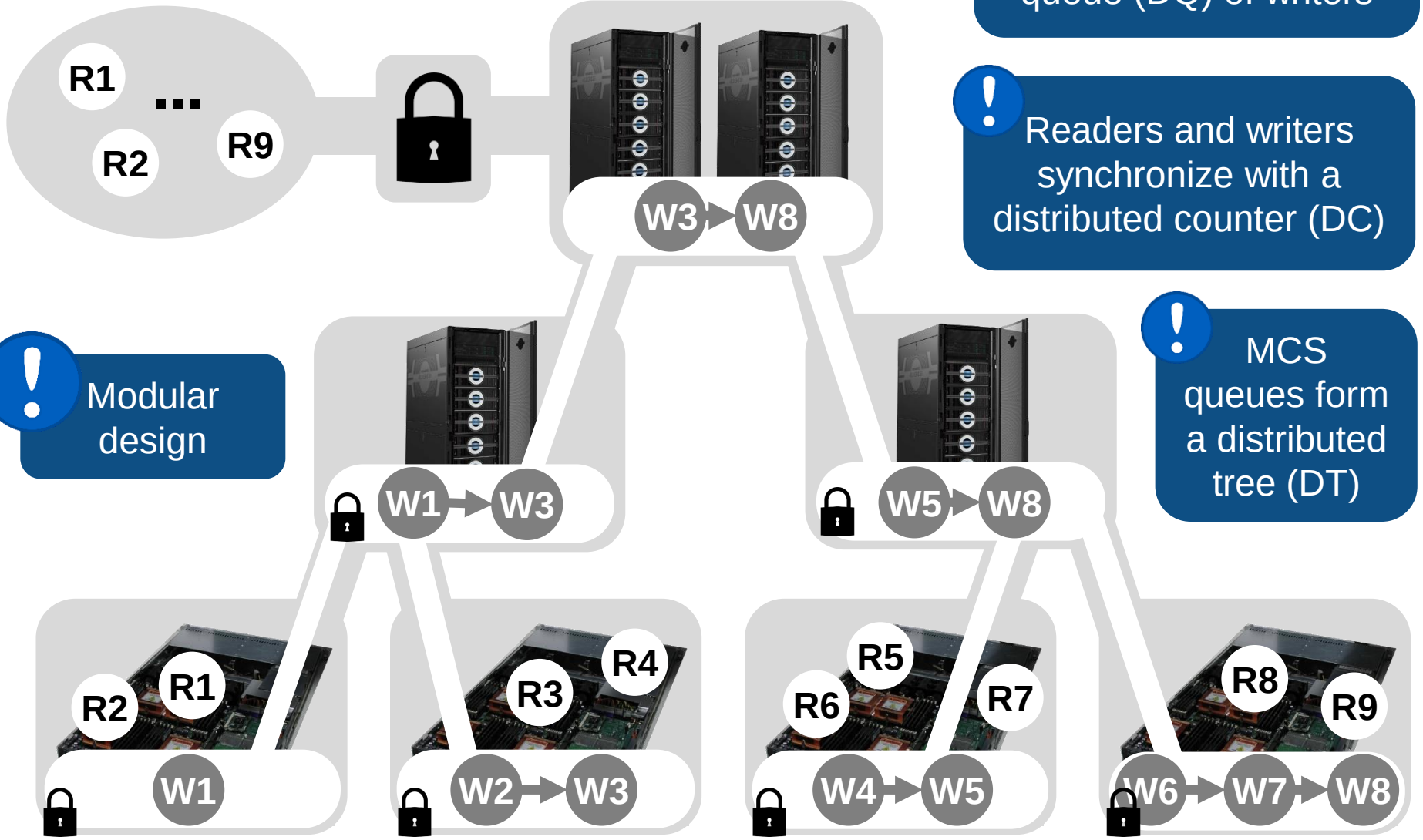
How to manage the design complexity?

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



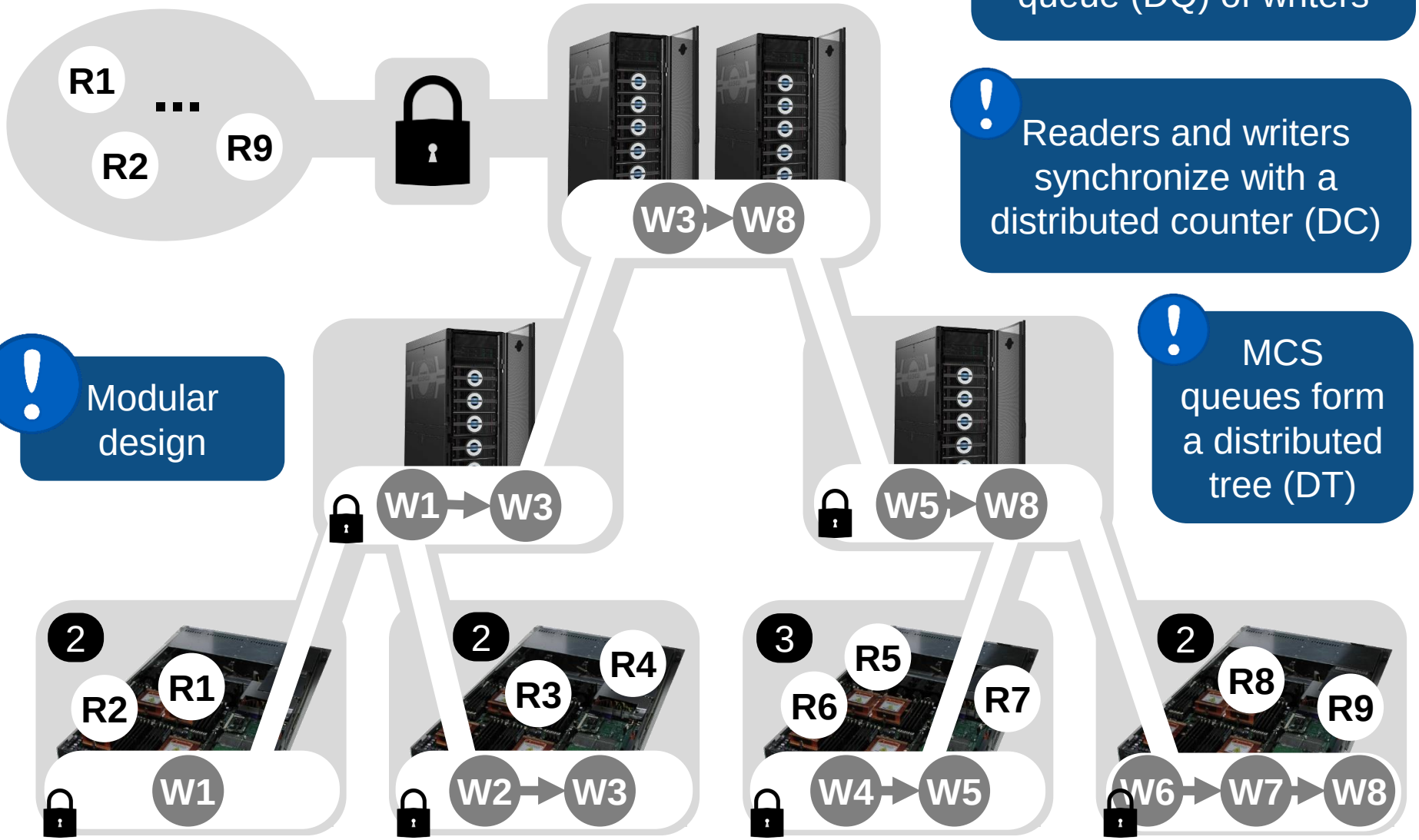
How to manage the design complexity?

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

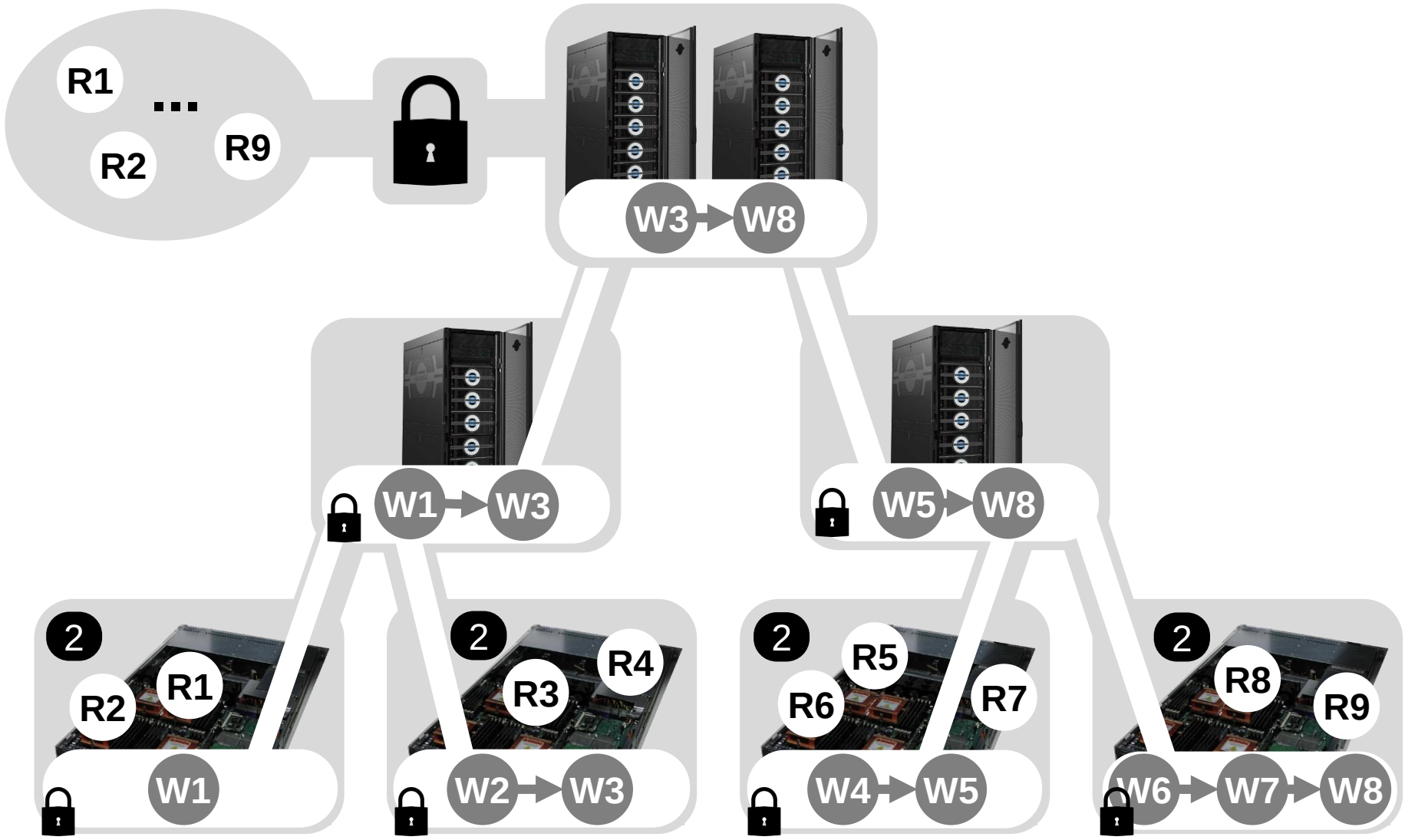
Readers and writers synchronize with a distributed counter (DC)

MCS queues form a distributed tree (DT)

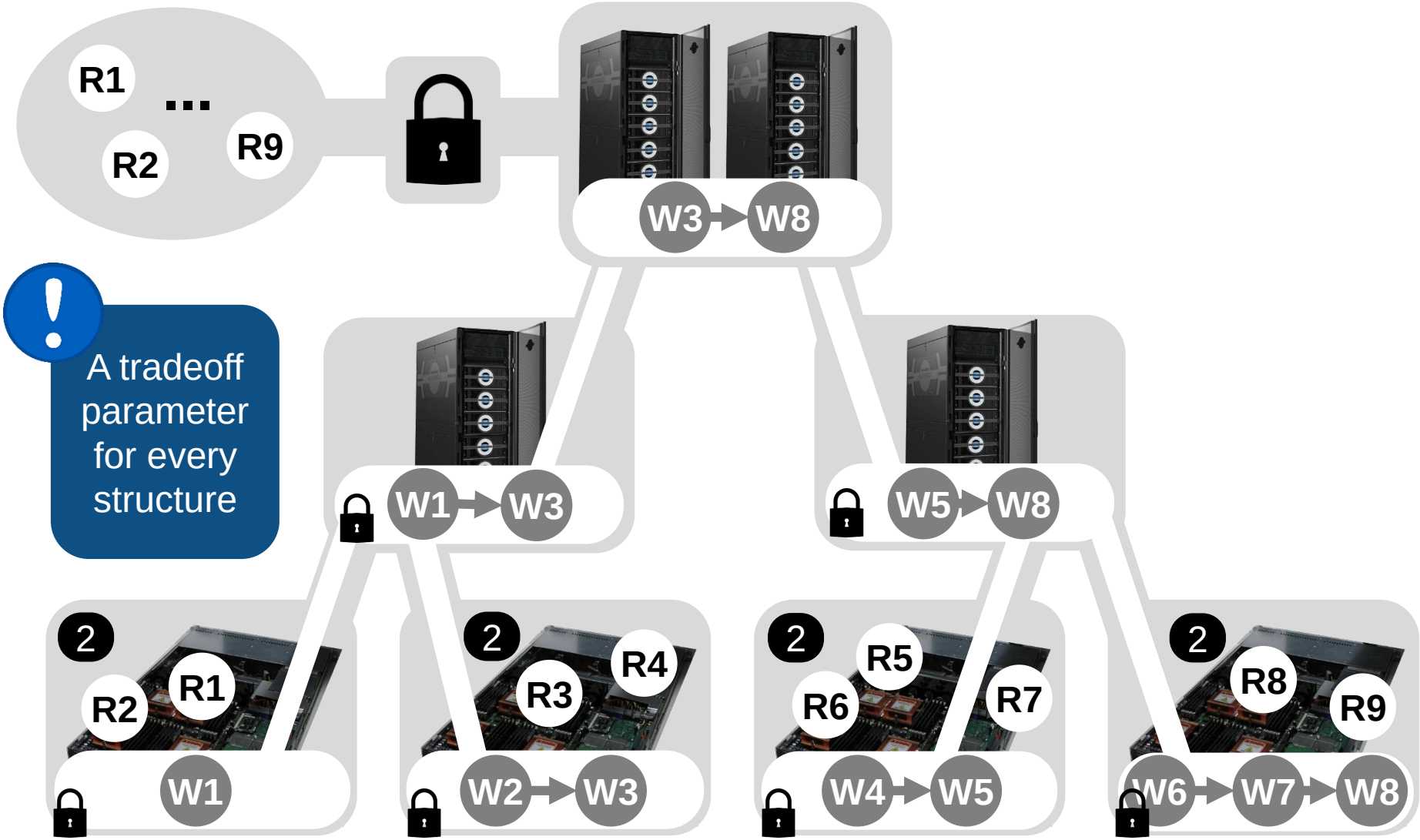
Modular design



How to ensure tunable performance?

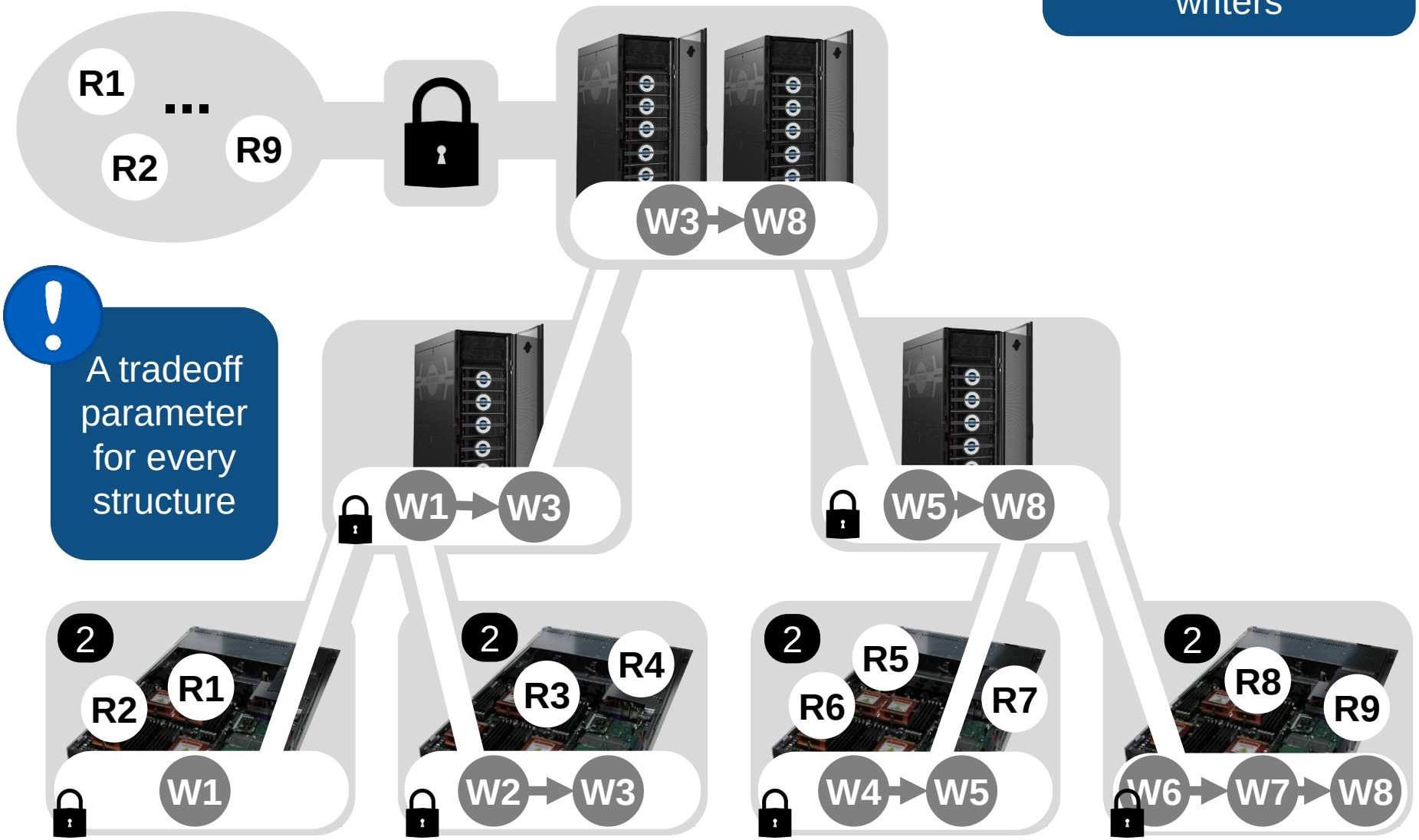


How to ensure tunable performance?



How to ensure tunable performance?

Each DQ: fairness vs throughput of writers

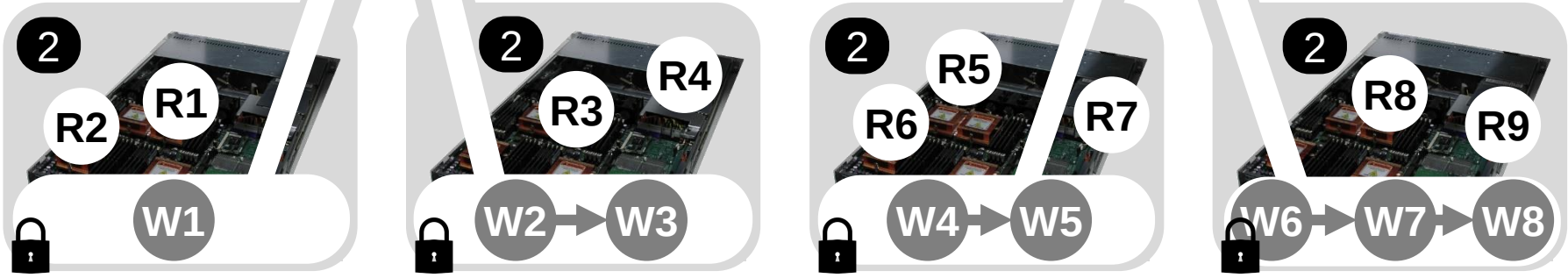


How to ensure tunable performance?

Each DQ: fairness vs throughput of writers

A tradeoff parameter for every structure

DT: a parameter for the throughput of readers vs writers



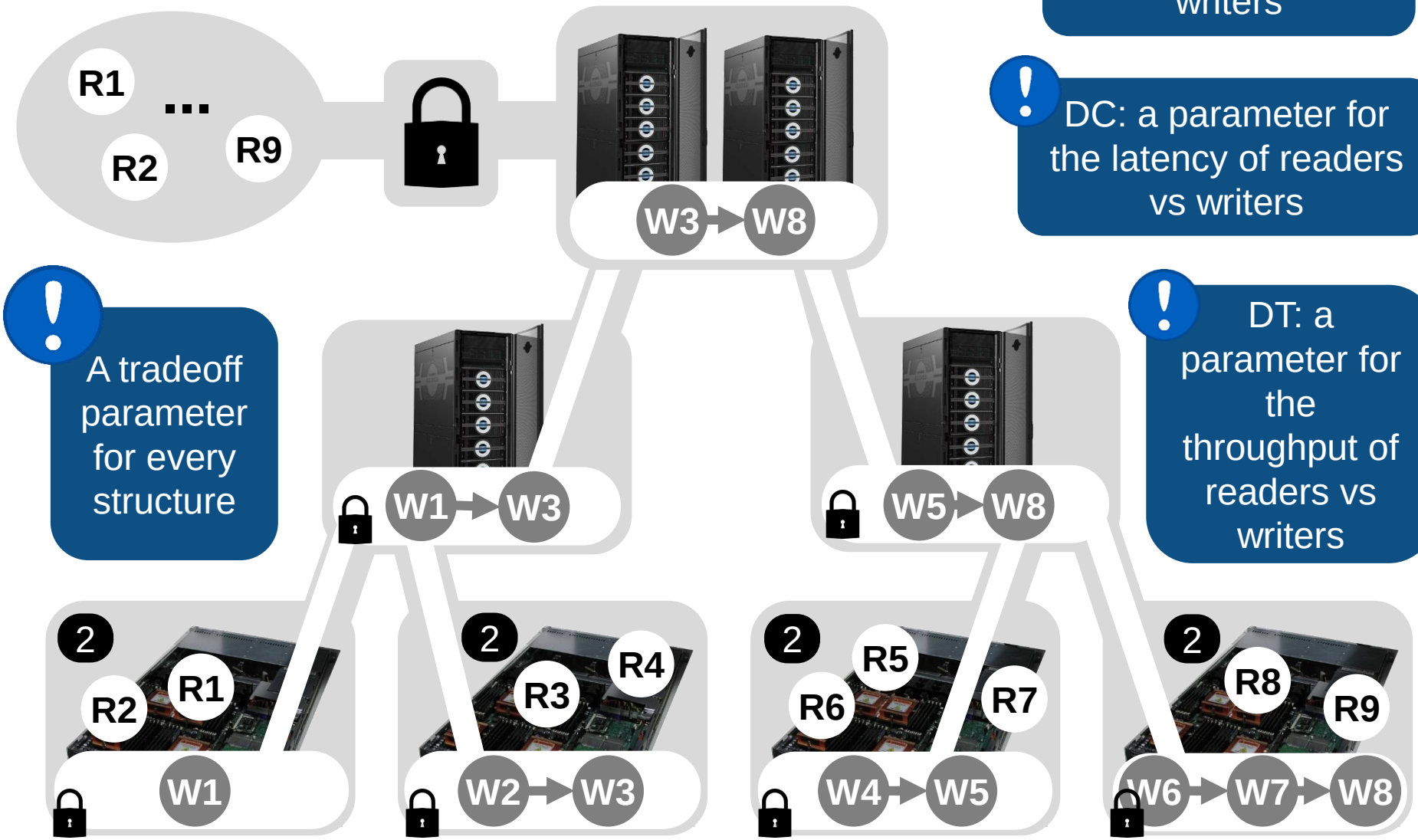
How to ensure tunable performance?

Each DQ: fairness vs throughput of writers

DC: a parameter for the latency of readers vs writers

DT: a parameter for the throughput of readers vs writers

A tradeoff parameter for every structure



DISTRIBUTED MCS QUEUES (DQS)

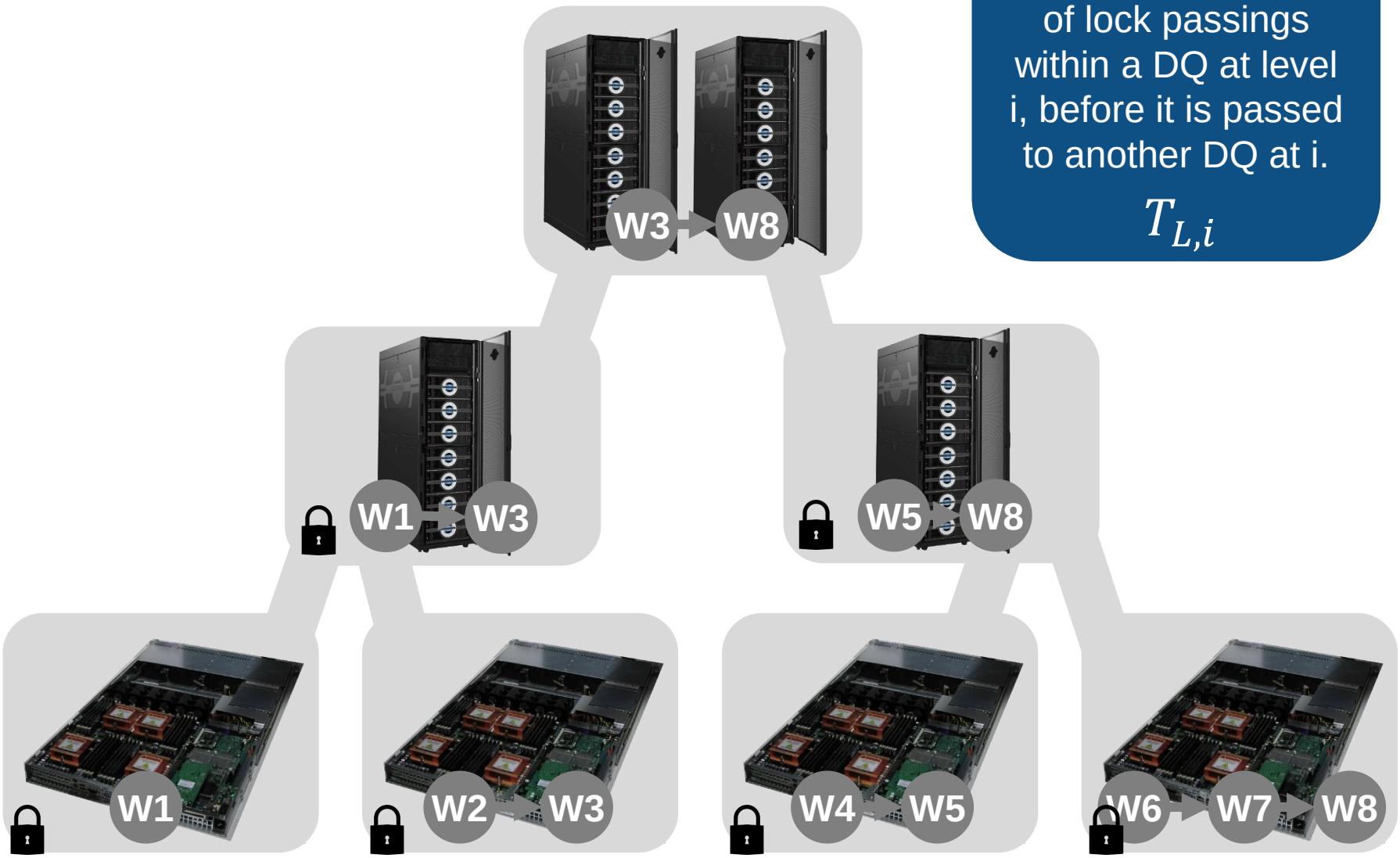
Throughput vs Fairness

DISTRIBUTED MCS QUEUES (DQS)

Throughput vs Fairness

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

$T_{L,i}$

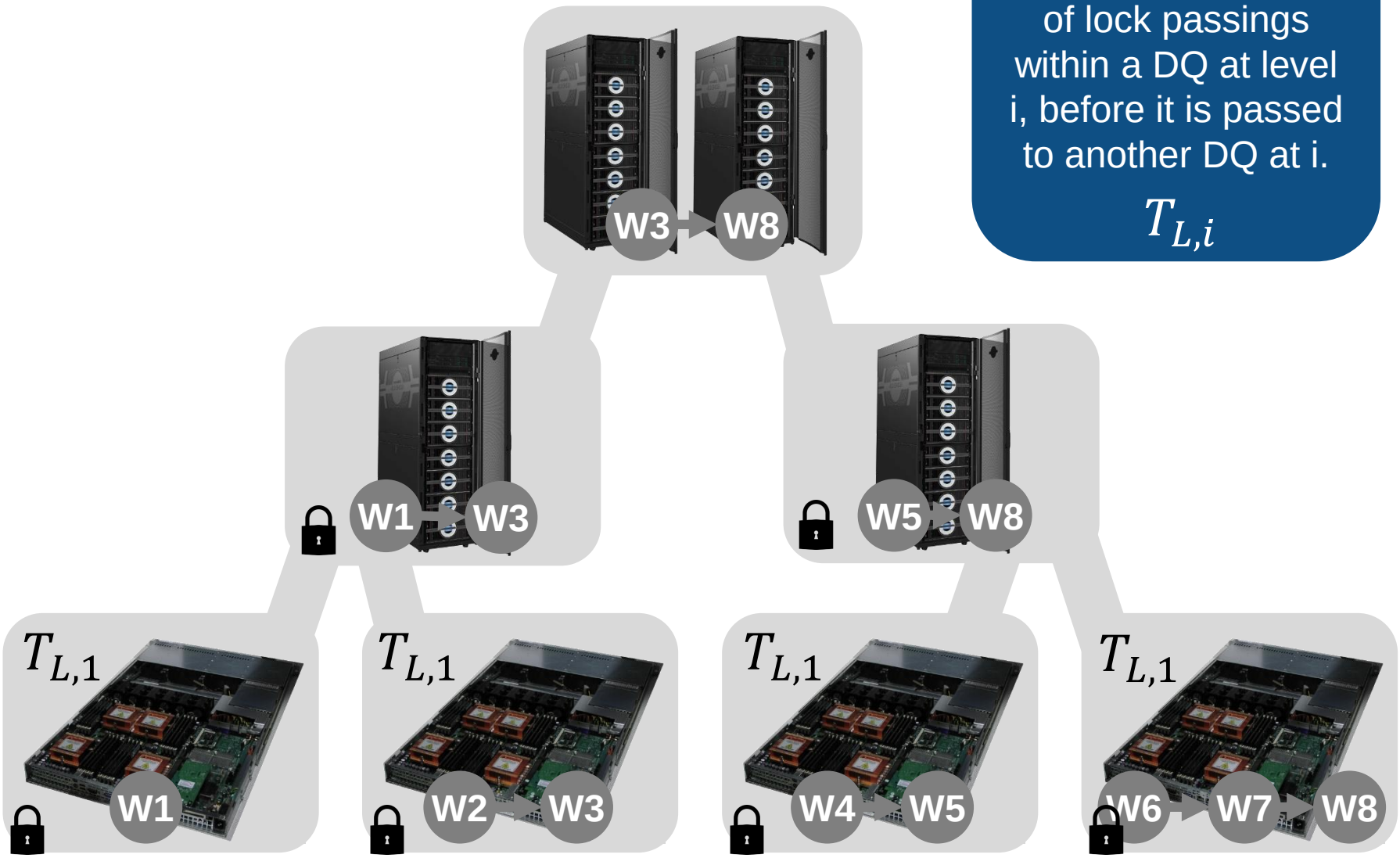


DISTRIBUTED MCS QUEUES (DQS)

Throughput vs Fairness

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

$T_{L,i}$

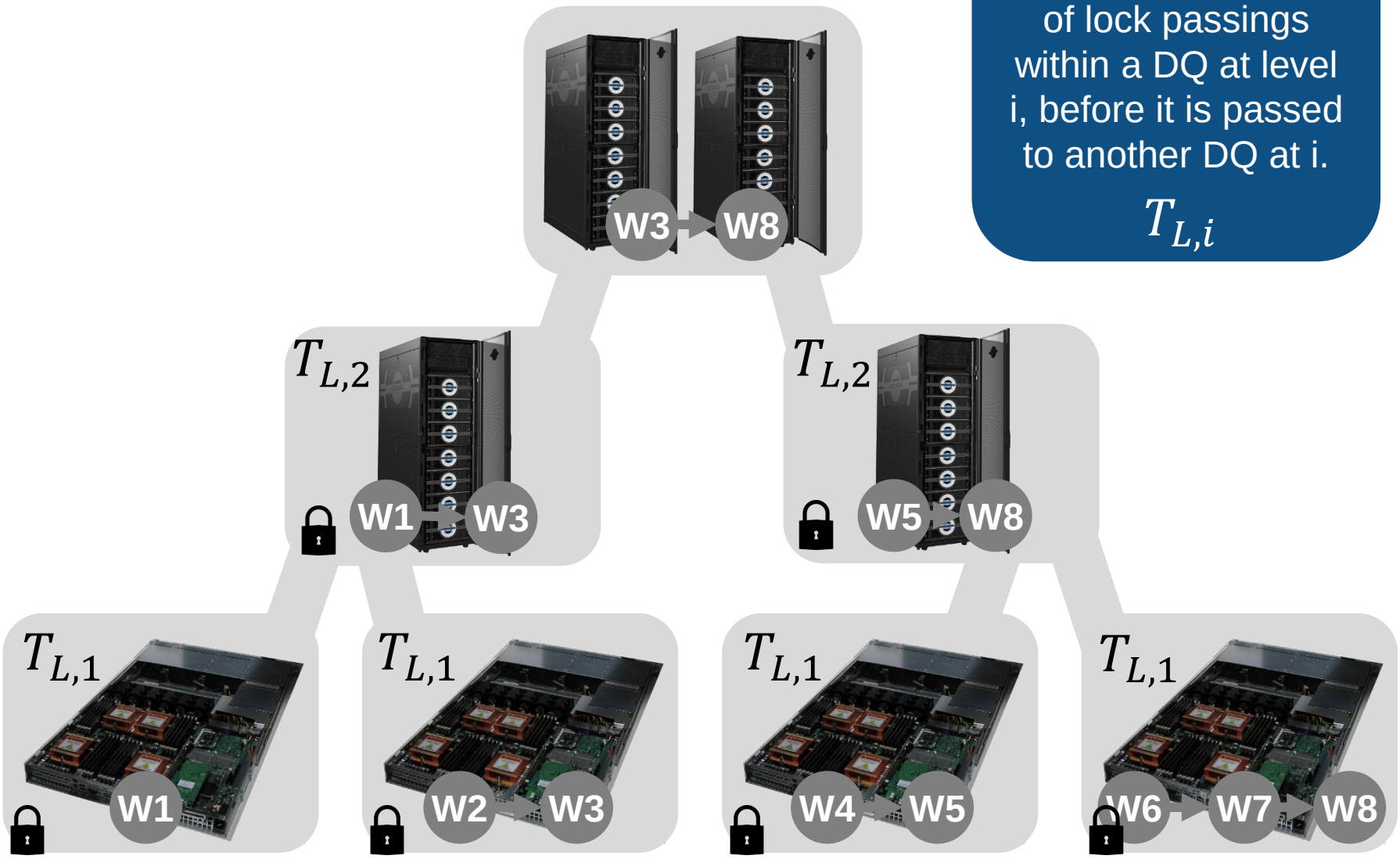


Distributed MCS Queues (DQs)

Throughput vs Fairness

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

$T_{L,i}$

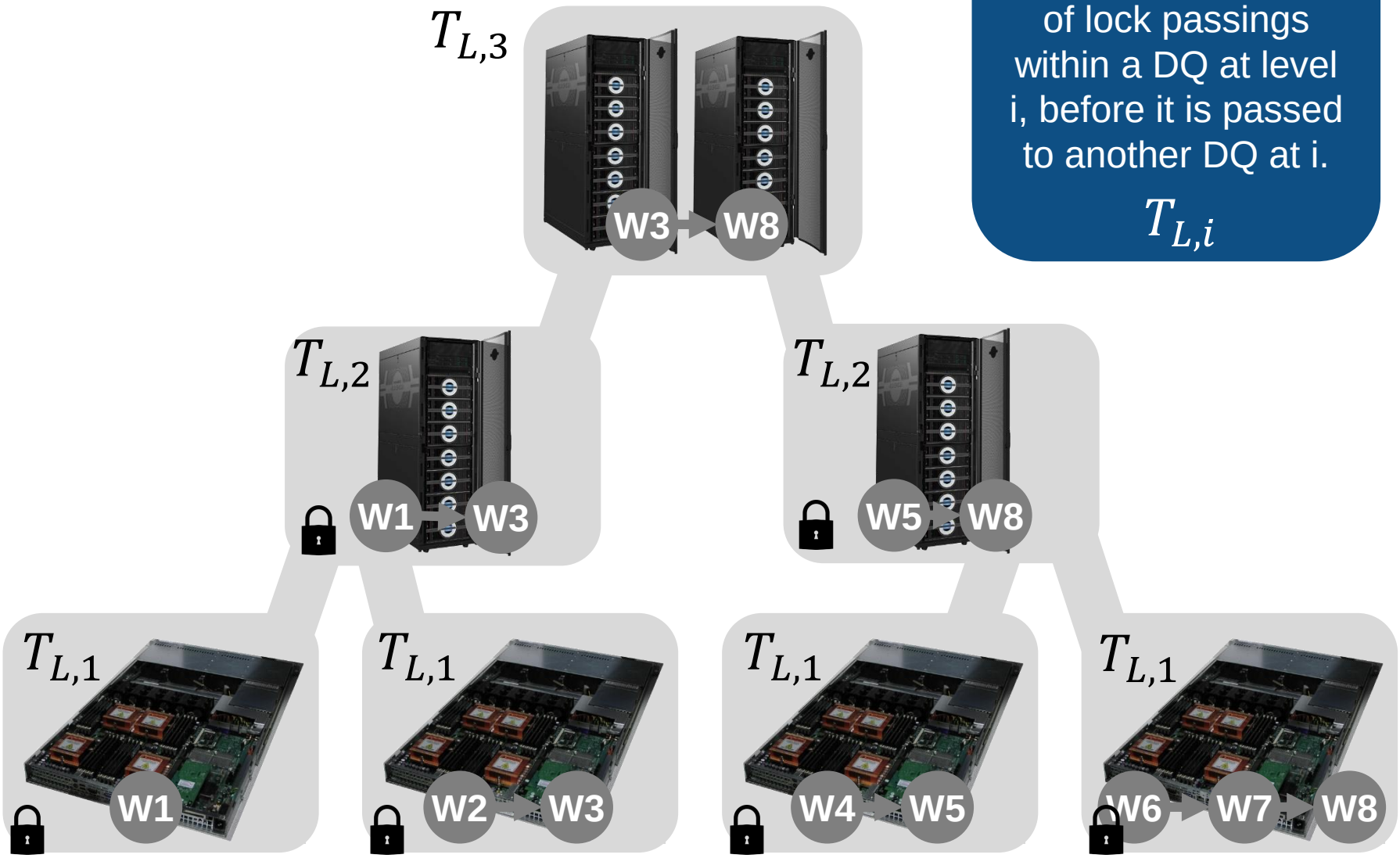


Distributed MCS Queues (DQs)

Throughput vs Fairness

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

$T_{L,i}$



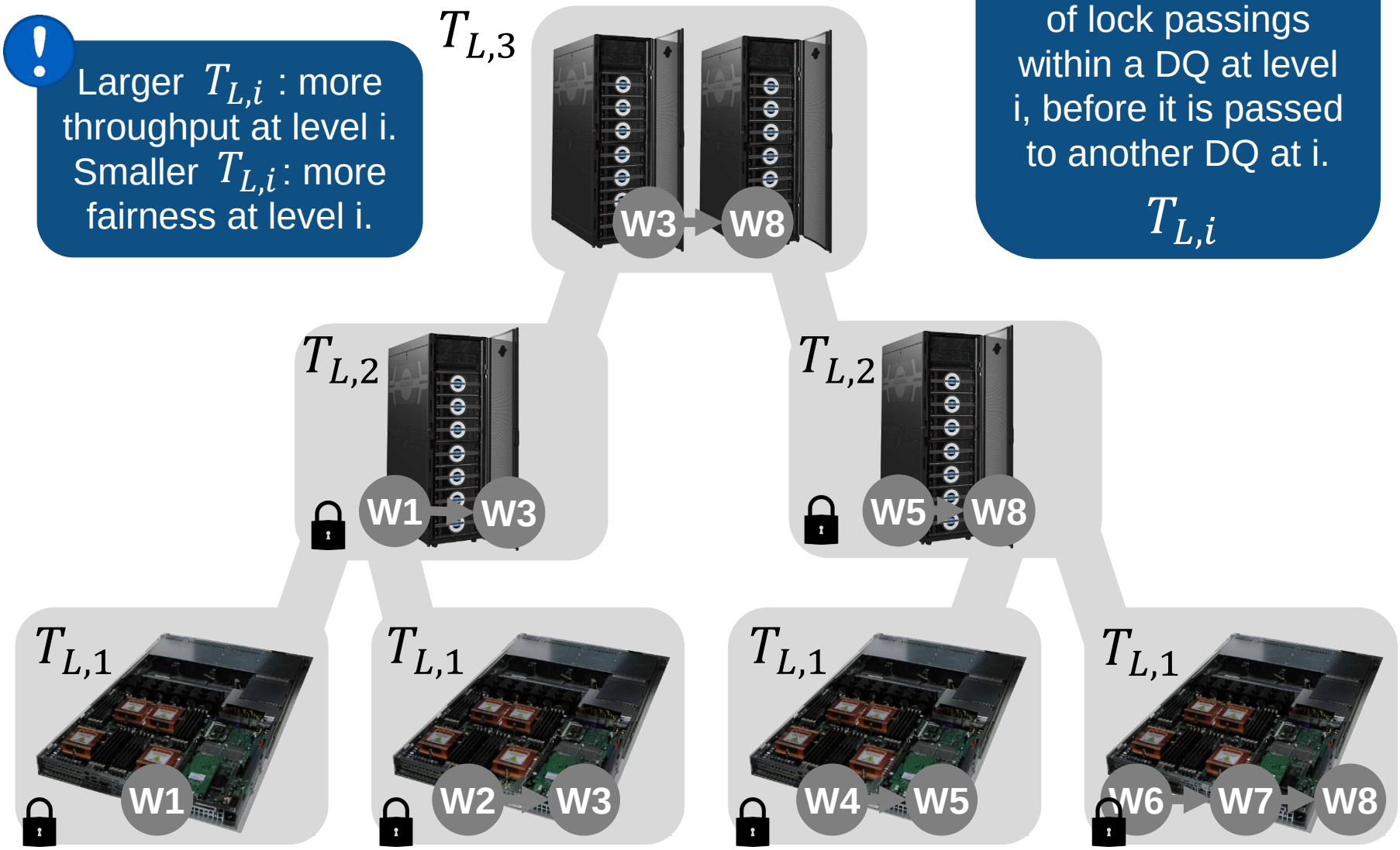
Distributed MCS Queues (DQs)

Throughput vs Fairness

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

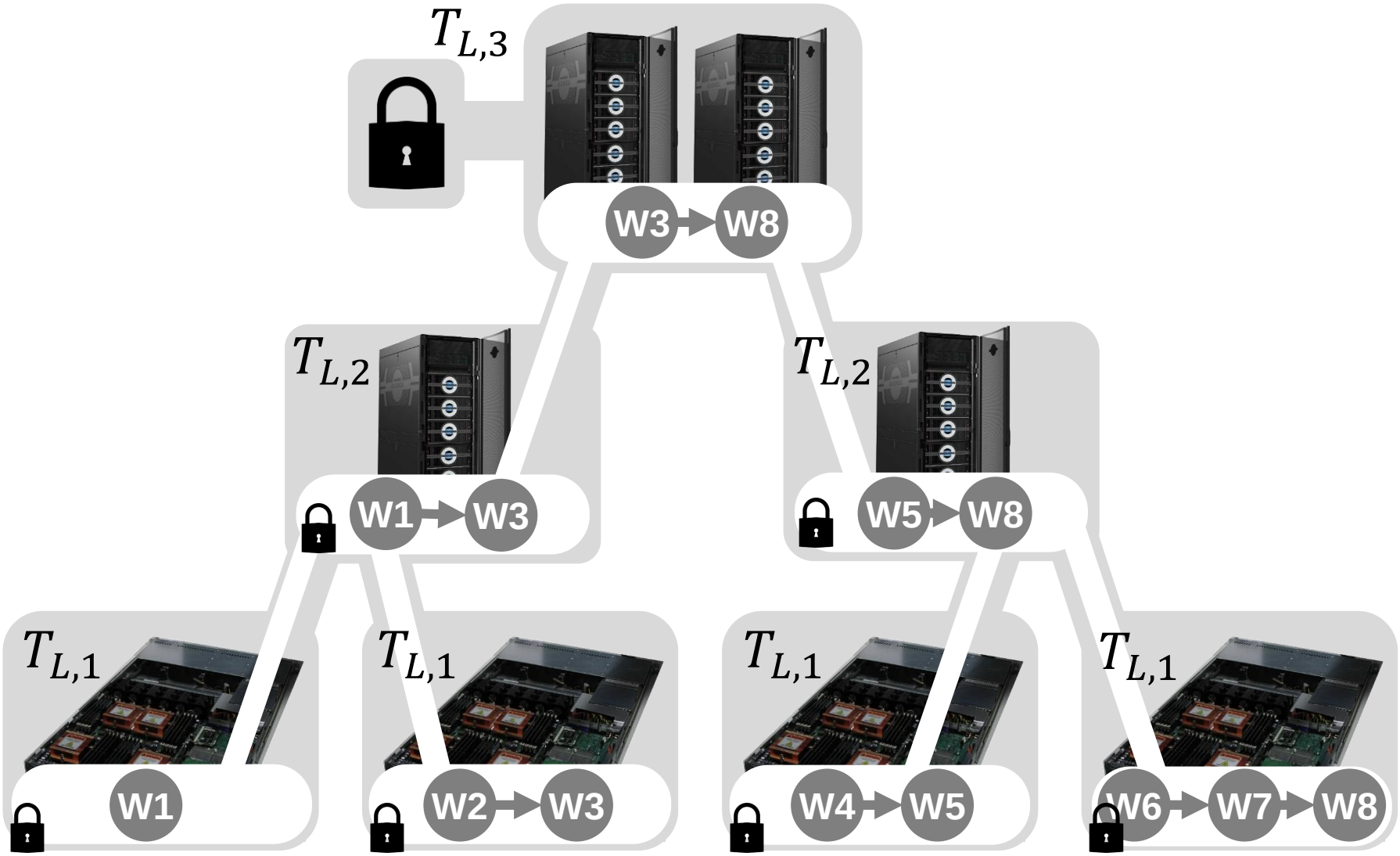
Each DQ: The maximum number of lock passings within a DQ at level i, before it is passed to another DQ at i.

$T_{L,i}$



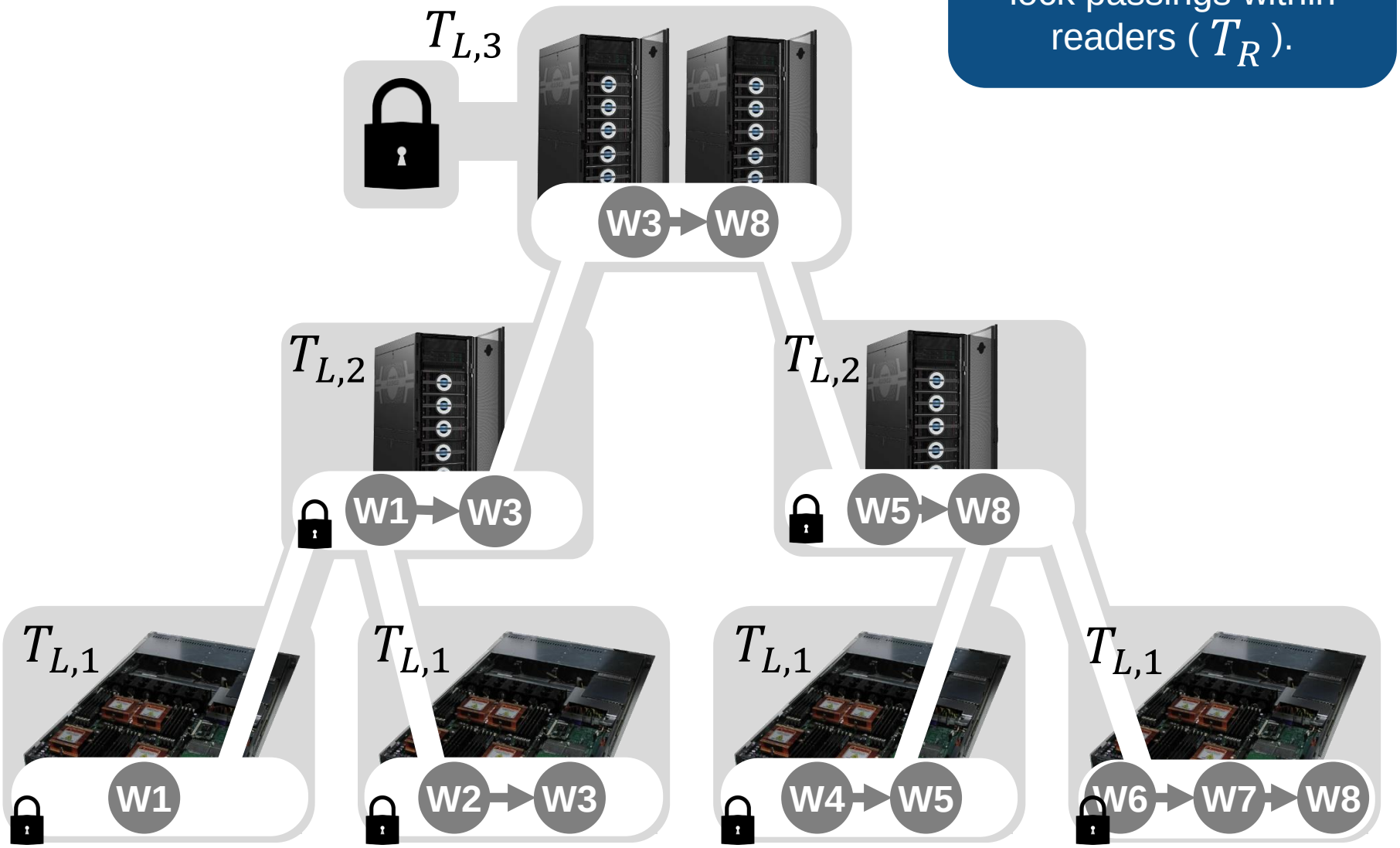
Distributed Tree of Queues (DT)

Throughput of readers vs writers



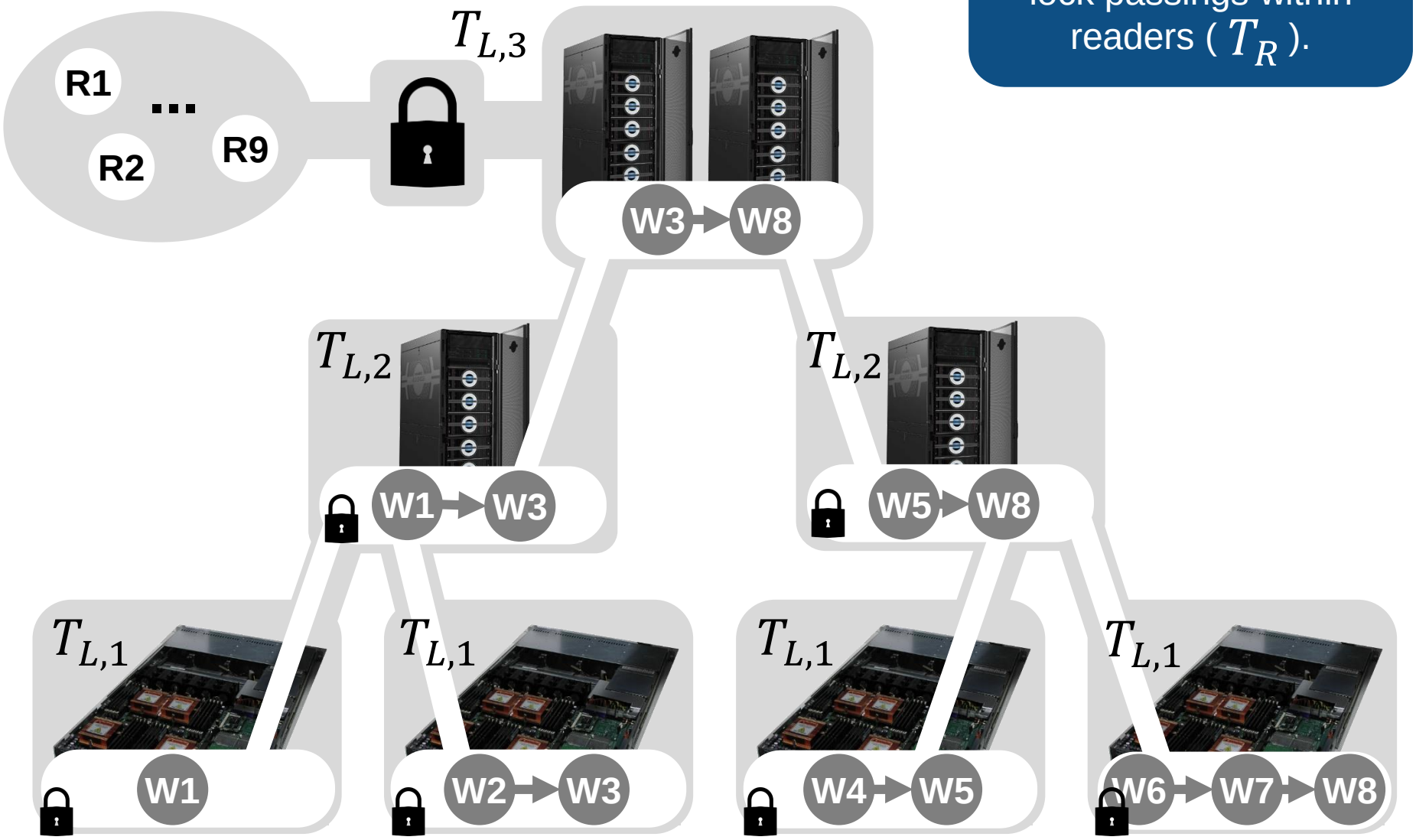
Distributed Tree of Queues (DT)
Throughput of readers vs writers

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



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

DISTRIBUTED COUNTER (DC)

Latency of readers vs writers

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

$$k = T_{DC}$$

DISTRIBUTED COUNTER (DC)
Latency of readers vs writers

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

$$k = T_{DC}$$

b|x|y

DISTRIBUTED COUNTER (DC)
Latency of readers vs writers

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

$$k = T_{DC}$$

A writer holds the lock — b|x|y

DISTRIBUTED COUNTER (DC)
Latency of readers vs writers

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

$$k = T_{DC}$$

A writer holds the lock

b|x|y

Readers that arrived at the CS

DISTRIBUTED COUNTER (DC)
Latency of readers vs writers

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

$$k = T_{DC}$$

DISTRIBUTED COUNTER (DC)
Latency of readers vs writers

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

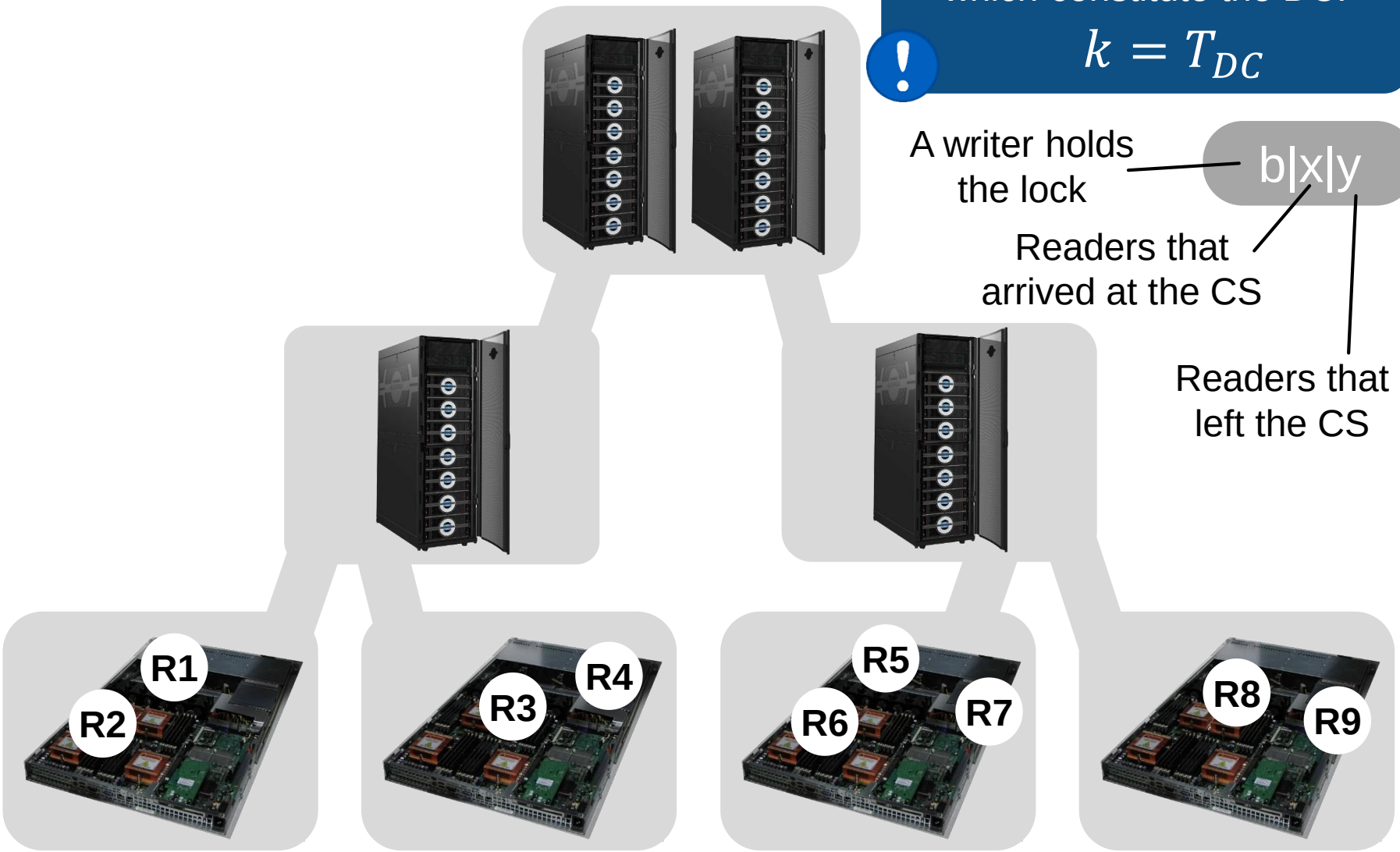
$$k = T_{DC}$$

A writer holds the lock

b|x|y

Readers that arrived at the CS

Readers that left the CS

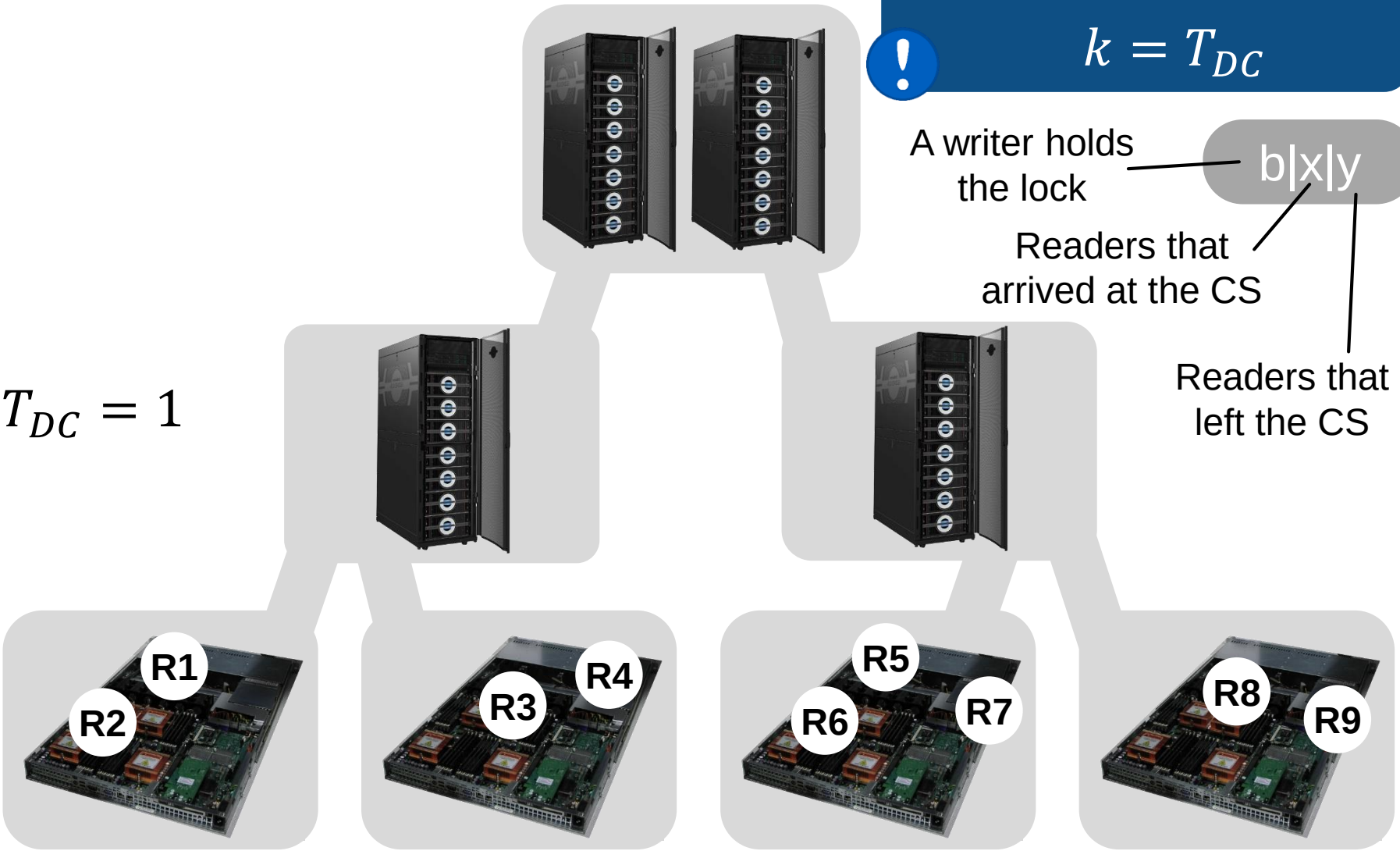


DISTRIBUTED COUNTER (DC)
Latency of readers vs writers

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

$$k = T_{DC}$$

$T_{DC} = 1$

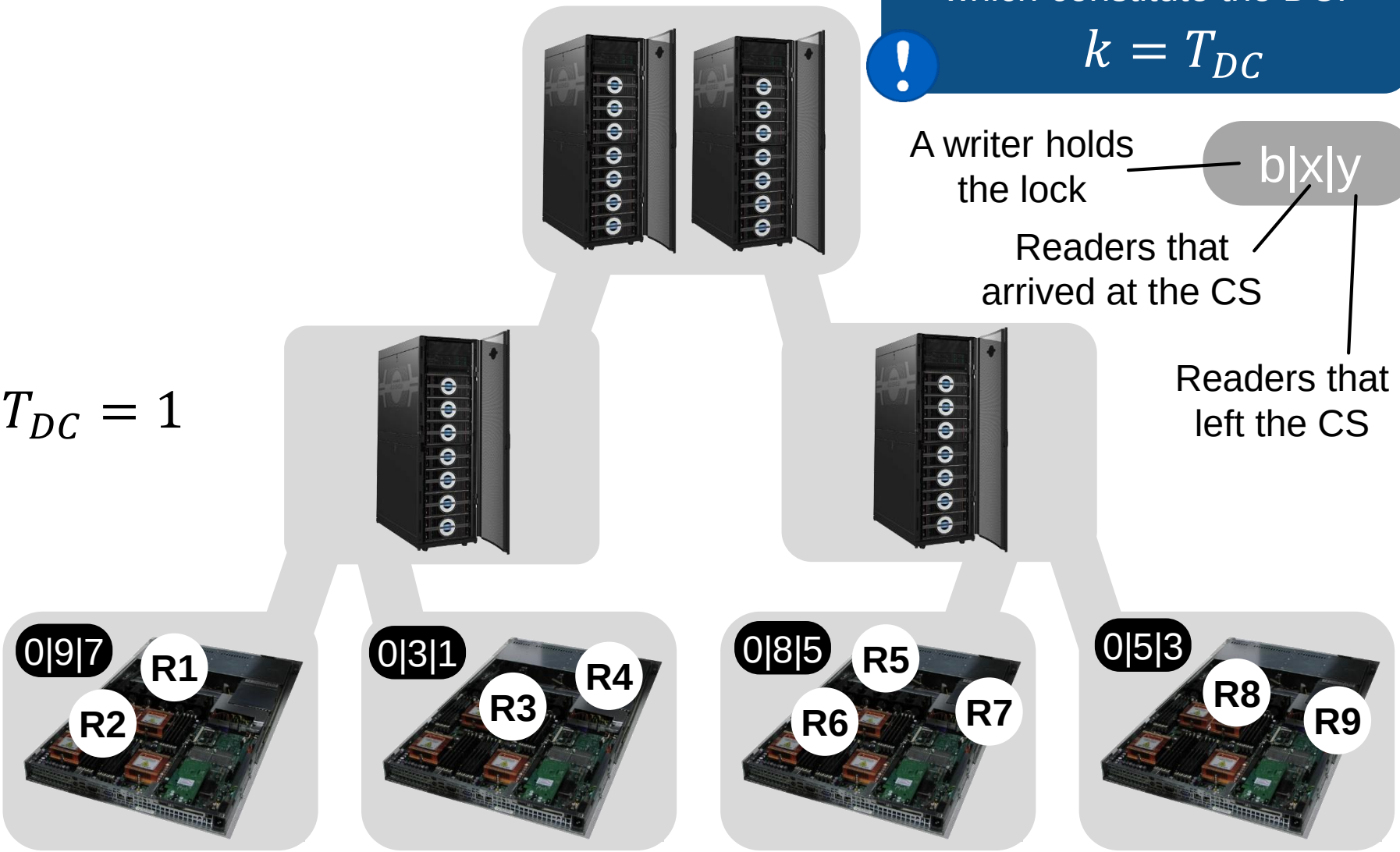


DISTRIBUTED COUNTER (DC)
Latency of readers vs writers

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

$$k = T_{DC}$$

$T_{DC} = 1$



DISTRIBUTED COUNTER (DC)
Latency of readers vs writers

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

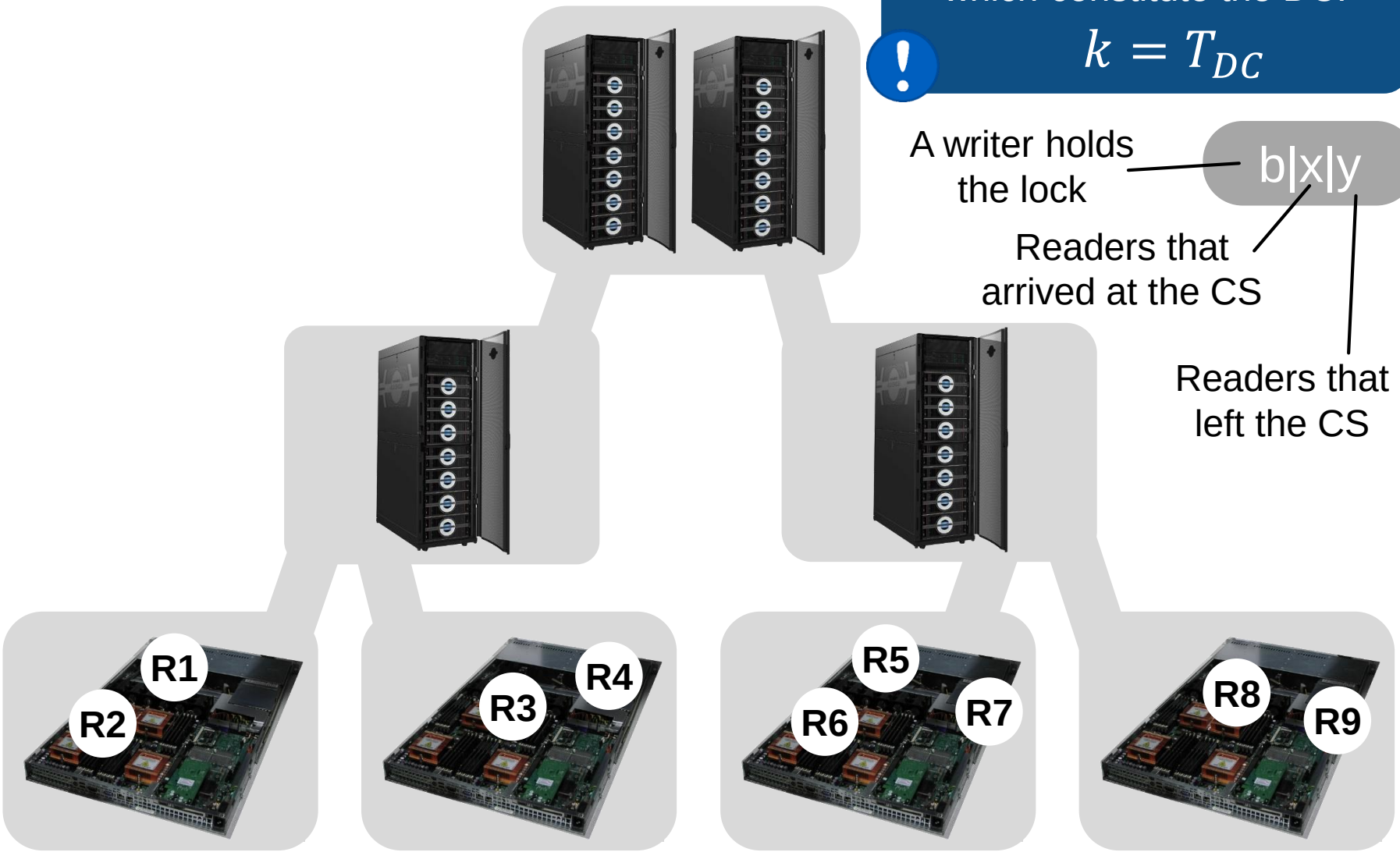
$$k = T_{DC}$$

A writer holds the lock

b|x|y

Readers that arrived at the CS

Readers that left the CS



DISTRIBUTED COUNTER (DC)
Latency of readers vs writers

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

$$k = T_{DC}$$

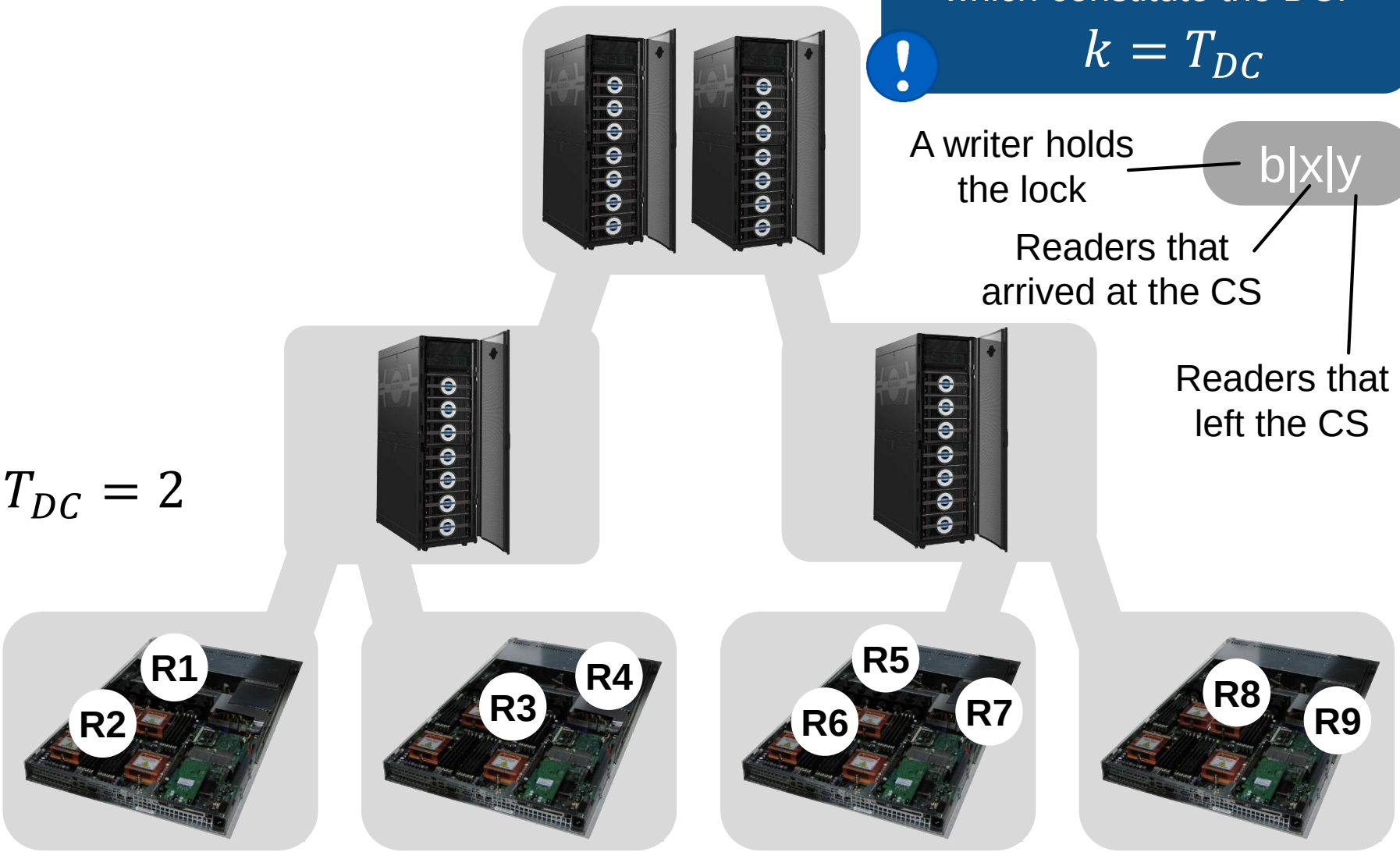
A writer holds the lock

b|x|y

Readers that arrived at the CS

Readers that left the CS

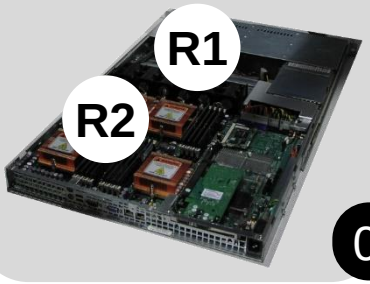
$T_{DC} = 2$

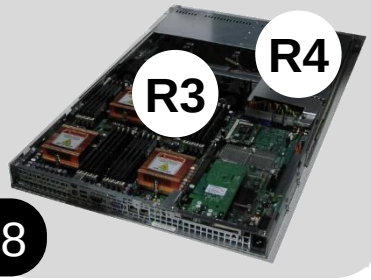


DISTRIBUTED COUNTER (DC)
Latency of readers vs writers

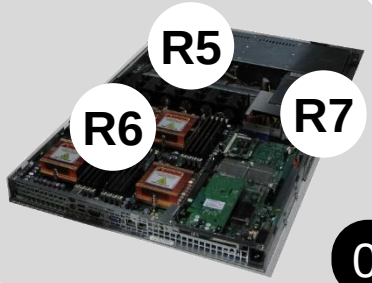
DC: every *k*th compute node hosts a partial counter, all of which constitute the DC.

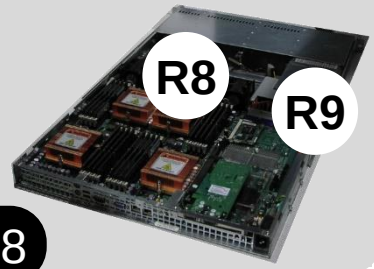
$$k = T_{DC}$$





0|12|8





0|13|8

A writer holds the lock

b|x|y

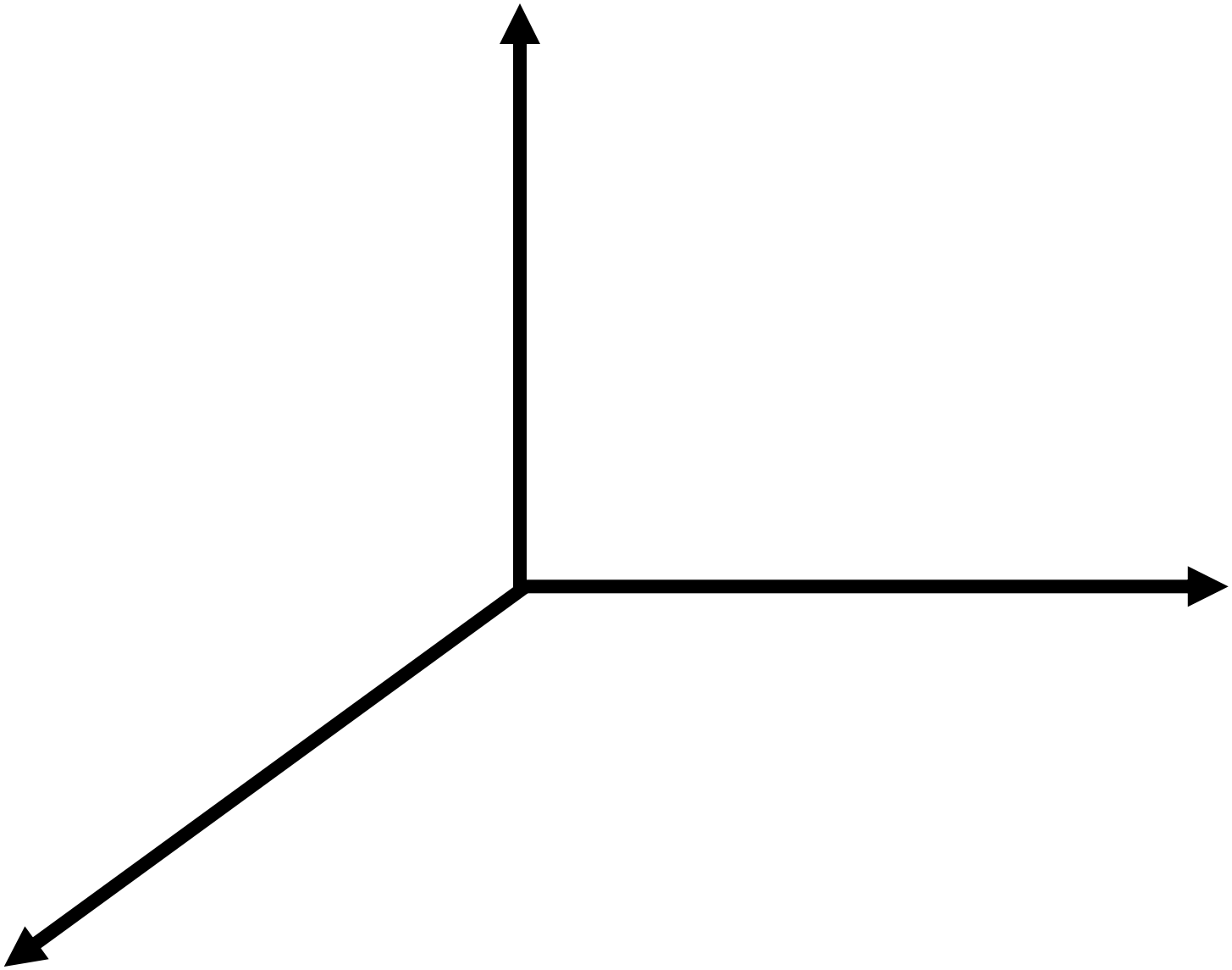
Readers that arrived at the CS

Readers that left the CS

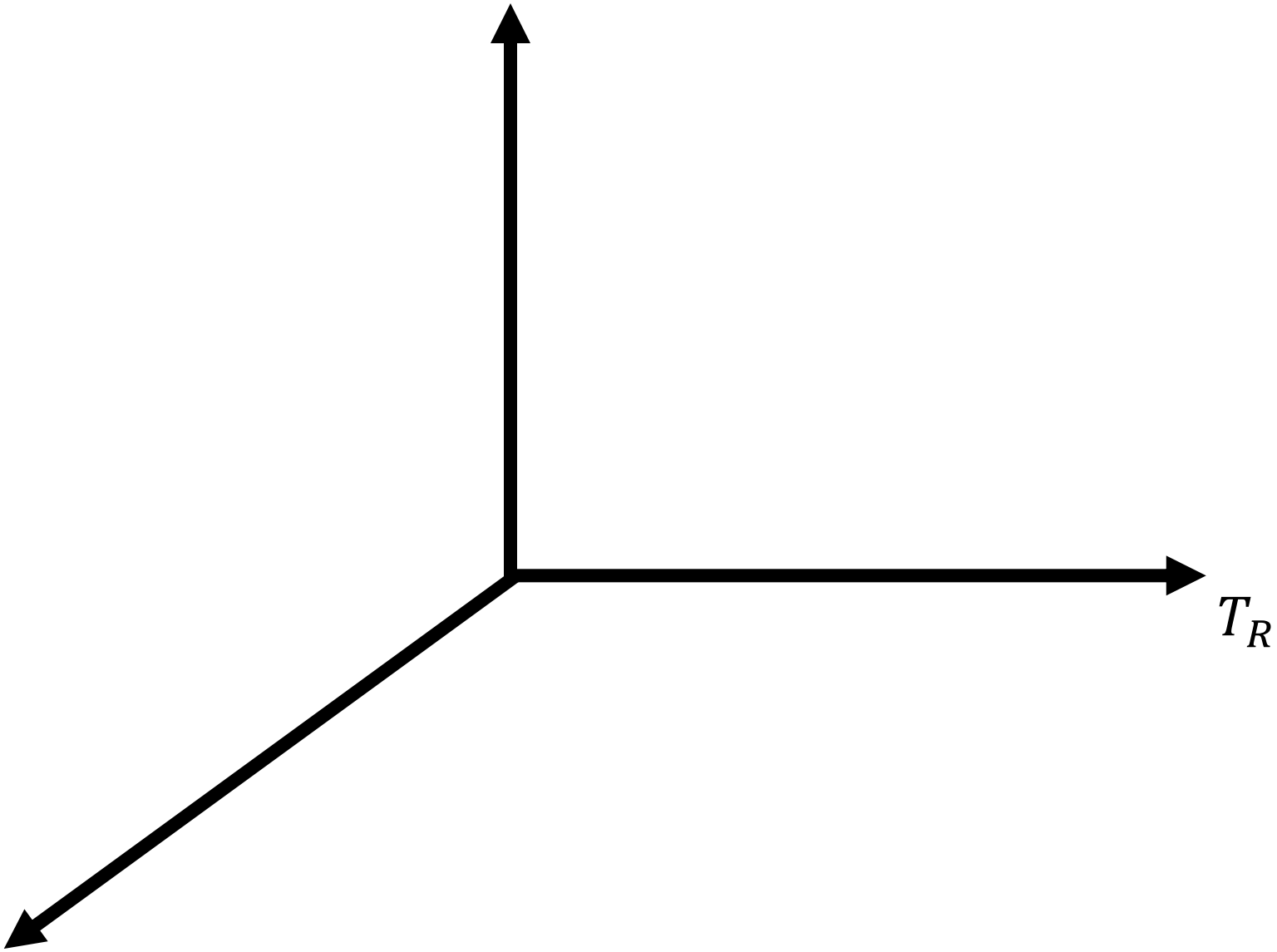
$T_{DC} = 2$

The Space of Designs

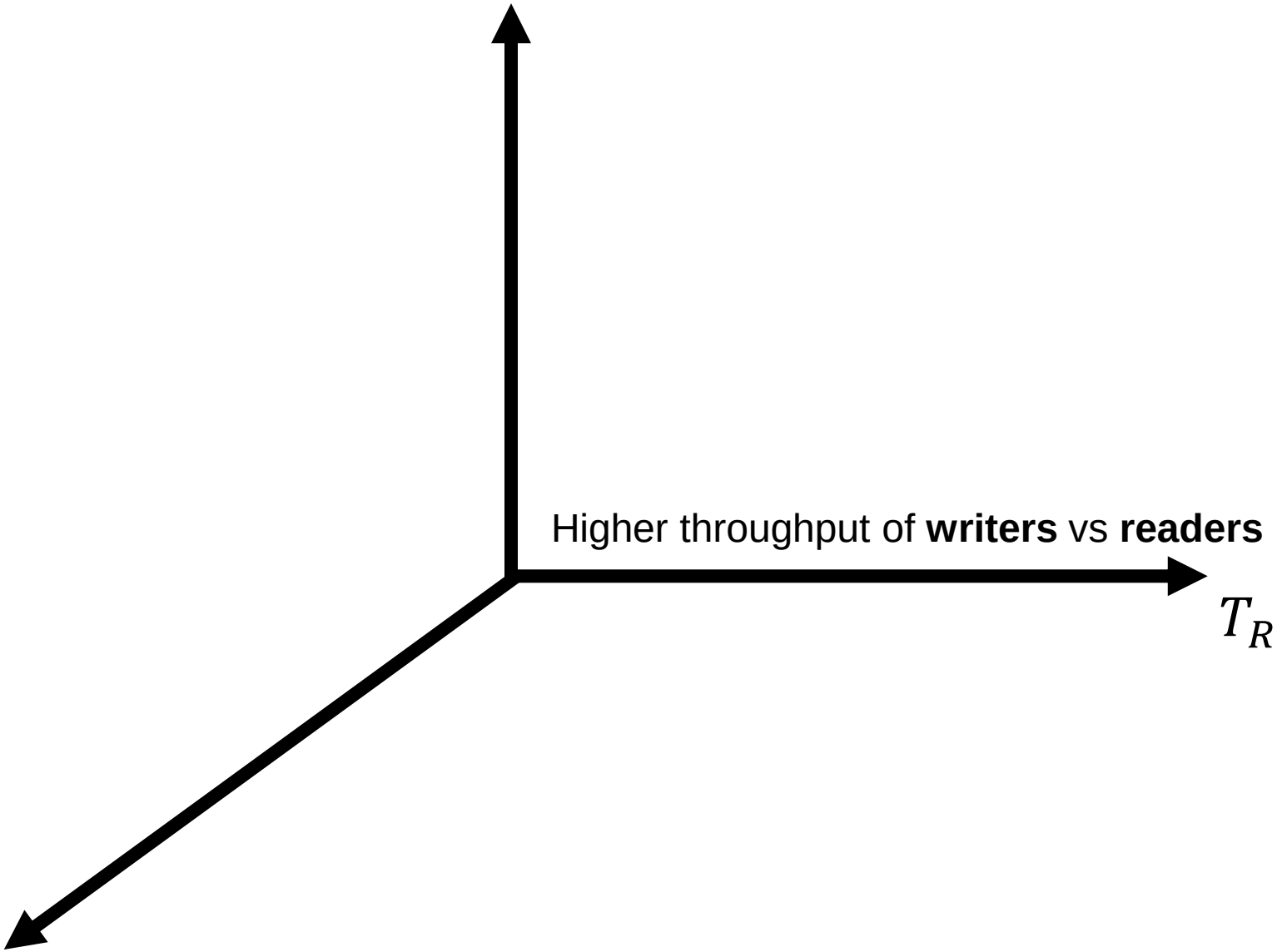
The Space of Designs



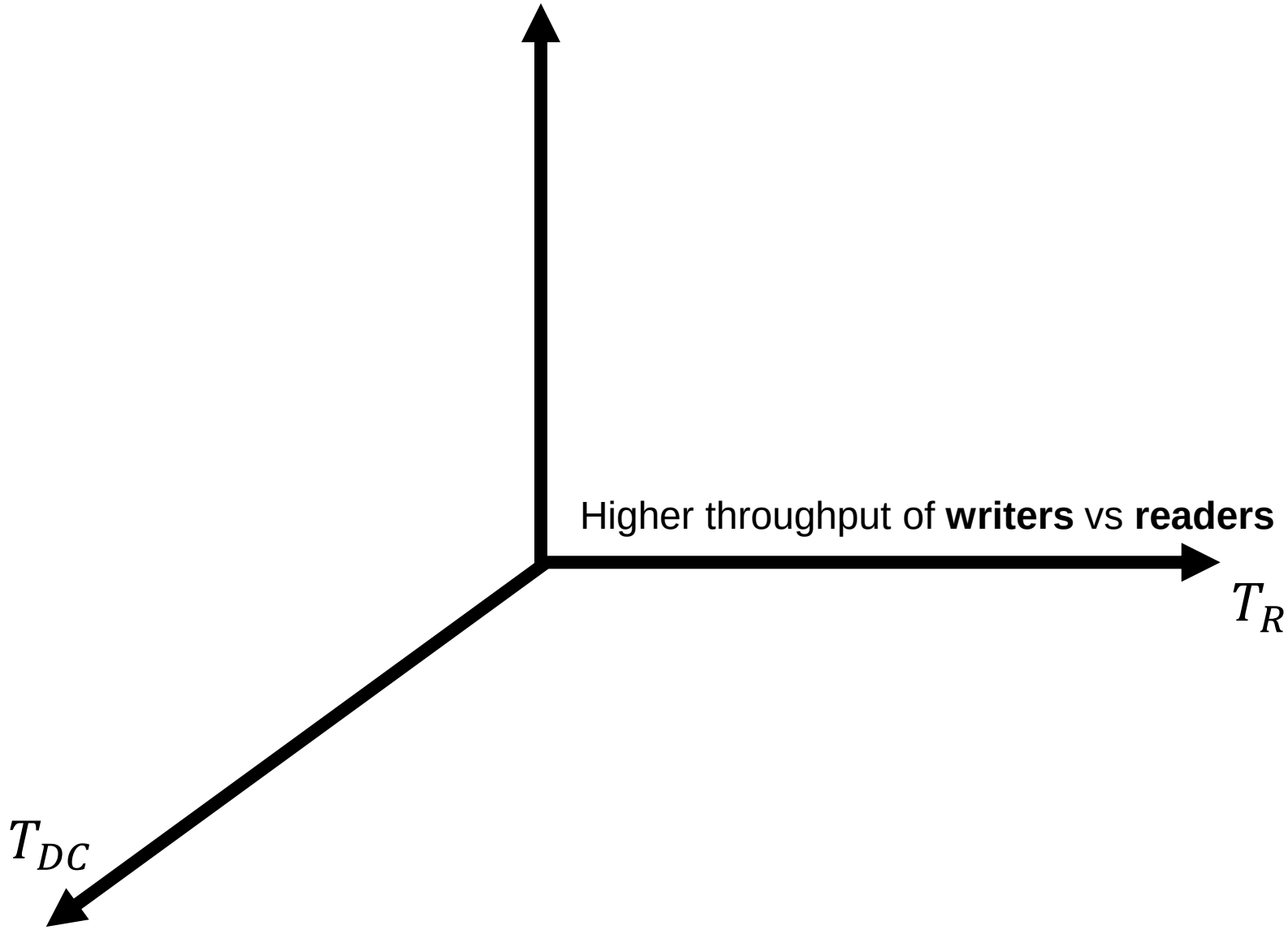
The Space of Designs



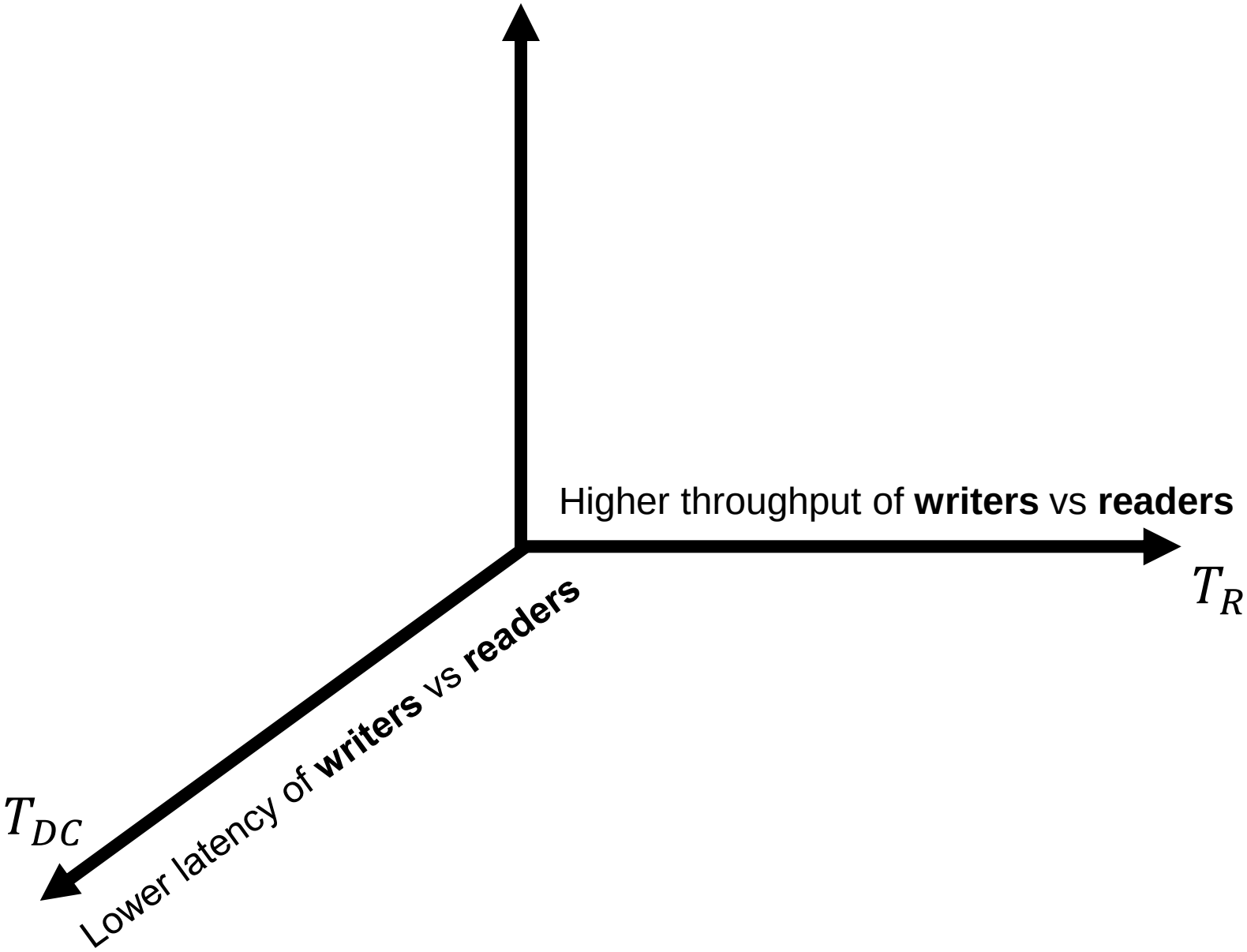
The Space of Designs



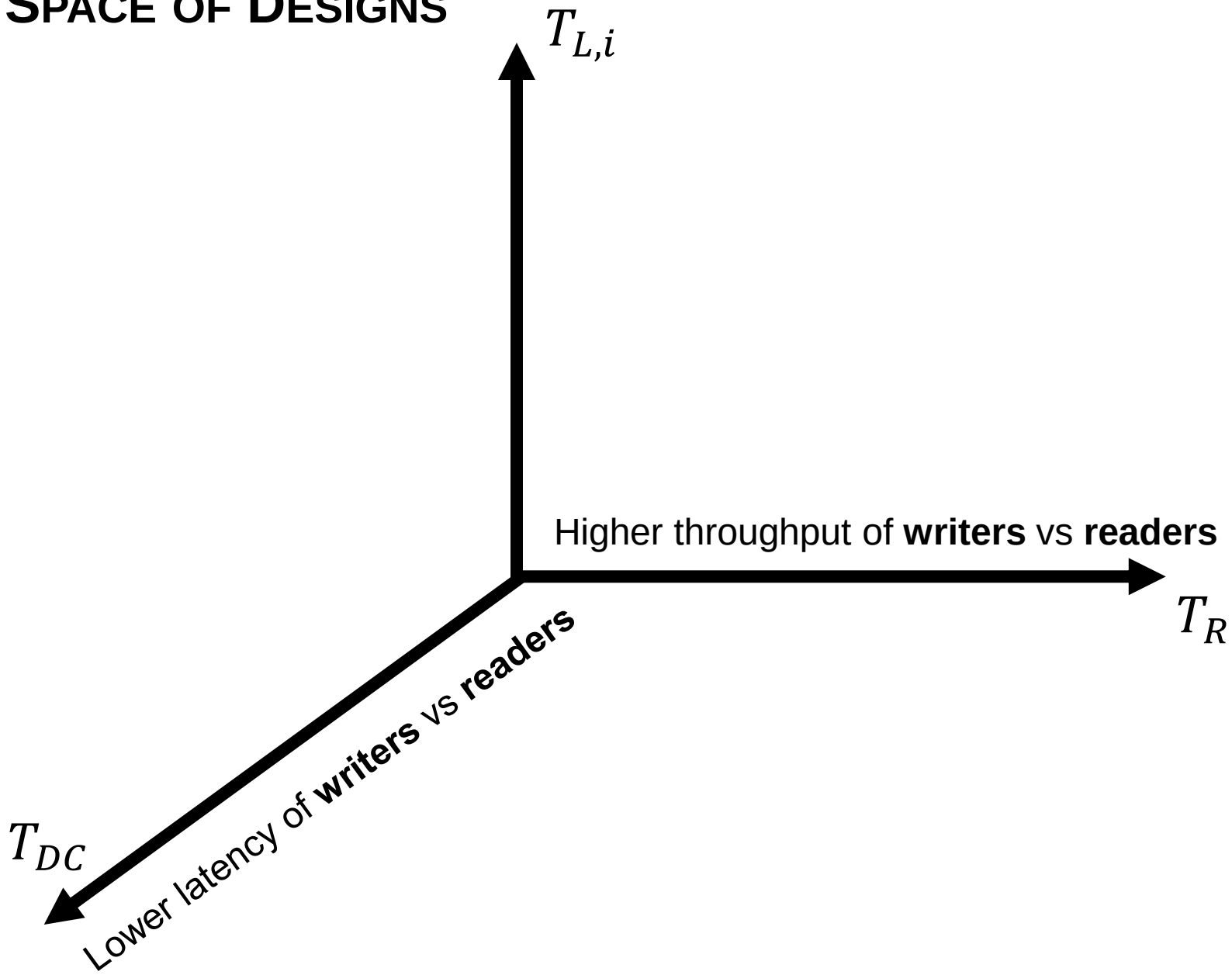
The Space of Designs



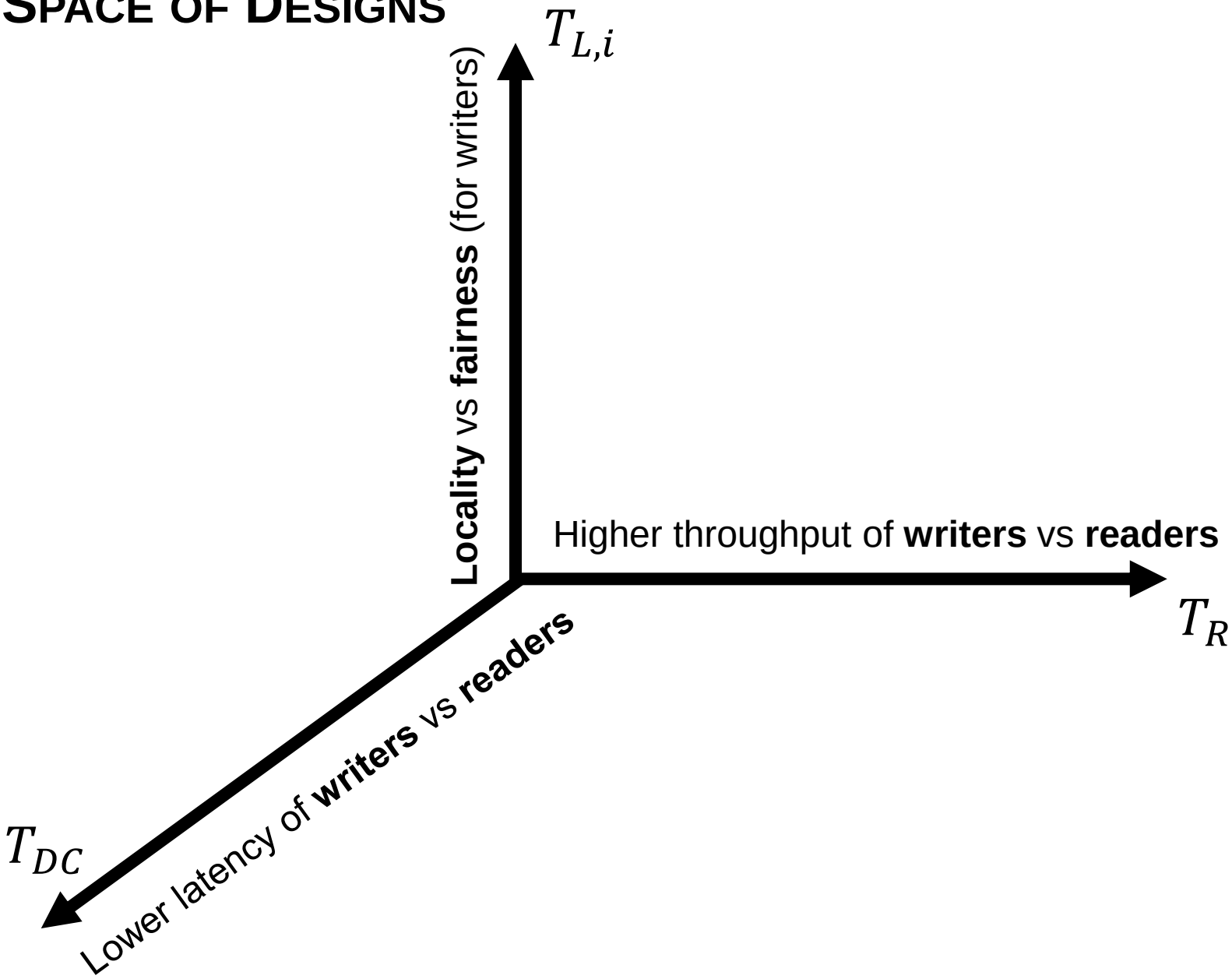
The Space of Designs



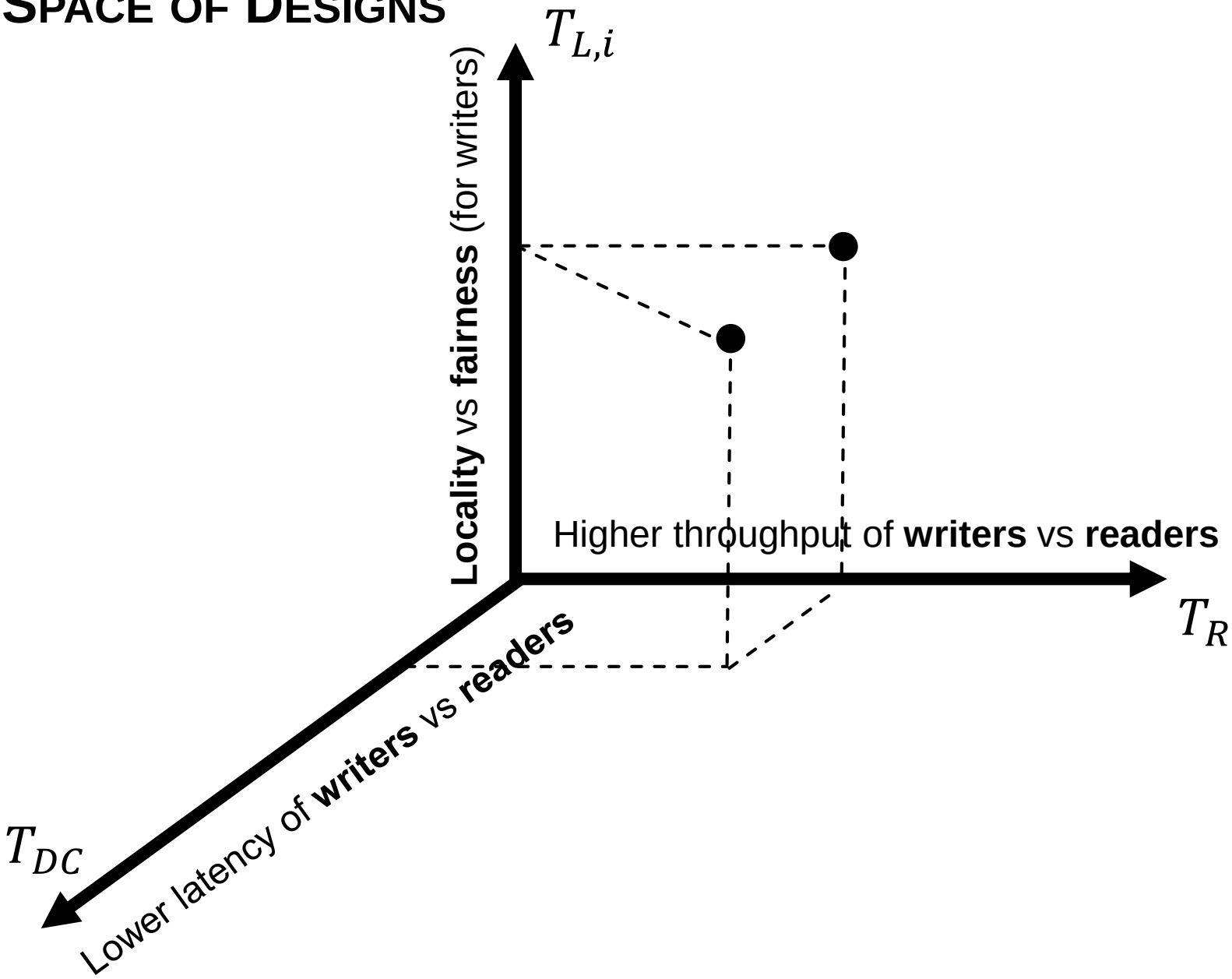
The Space of Designs



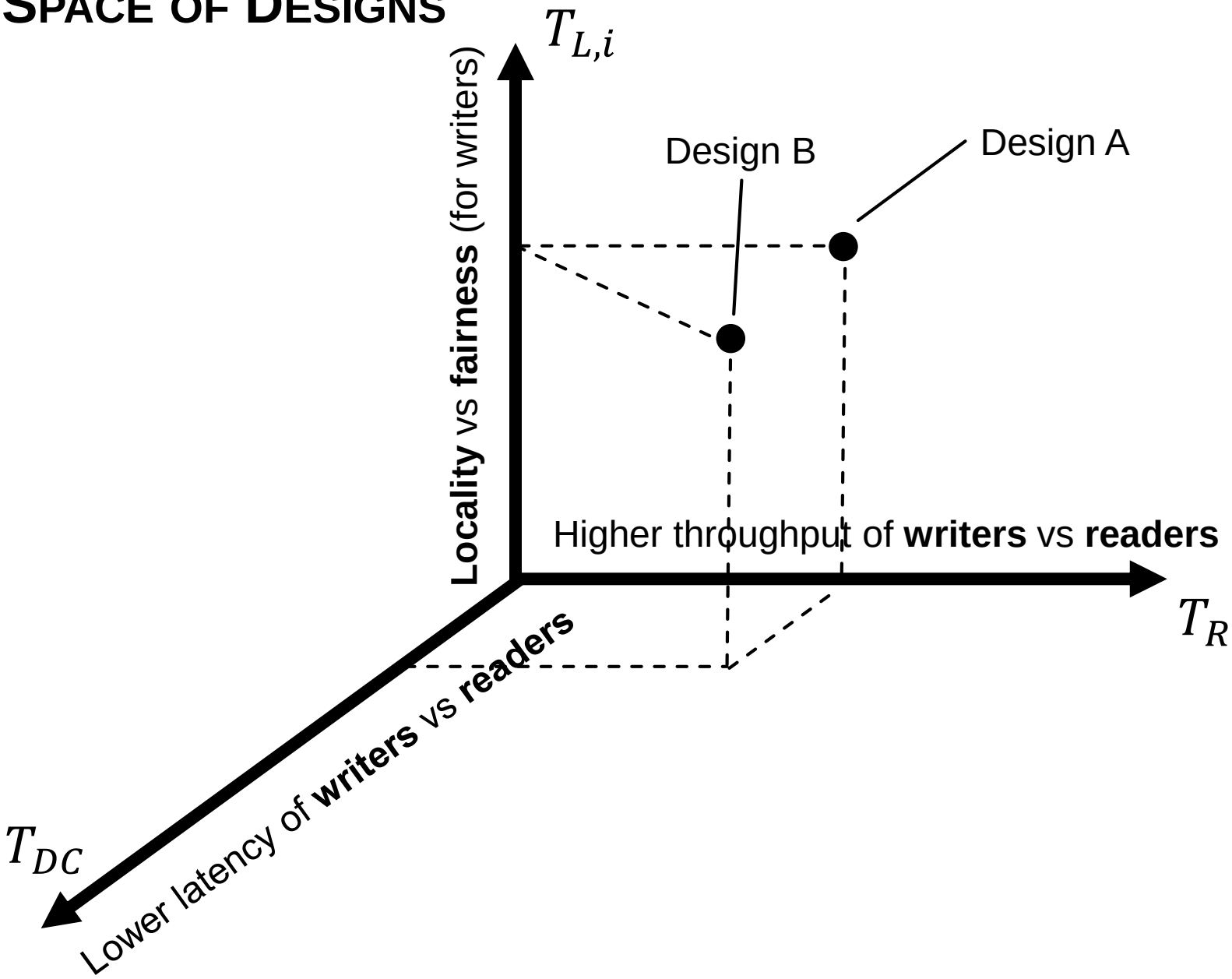
The Space of Designs



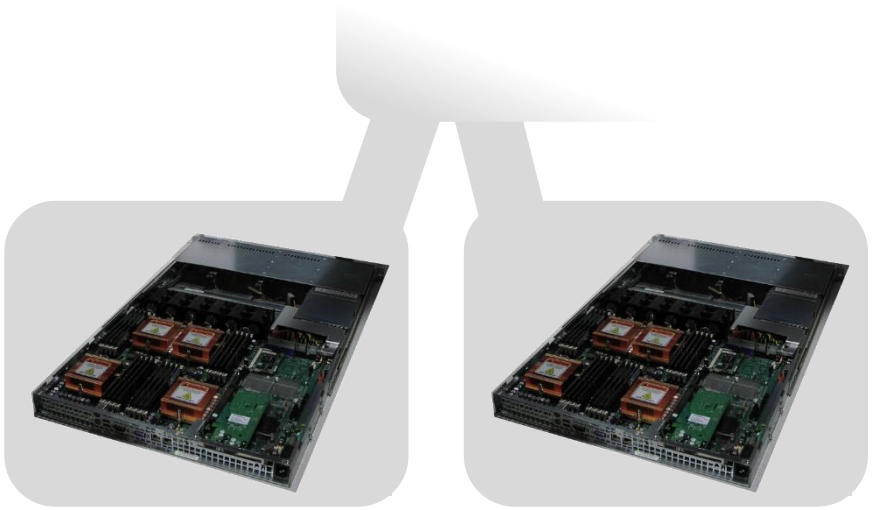
The Space of Designs



The Space of Designs

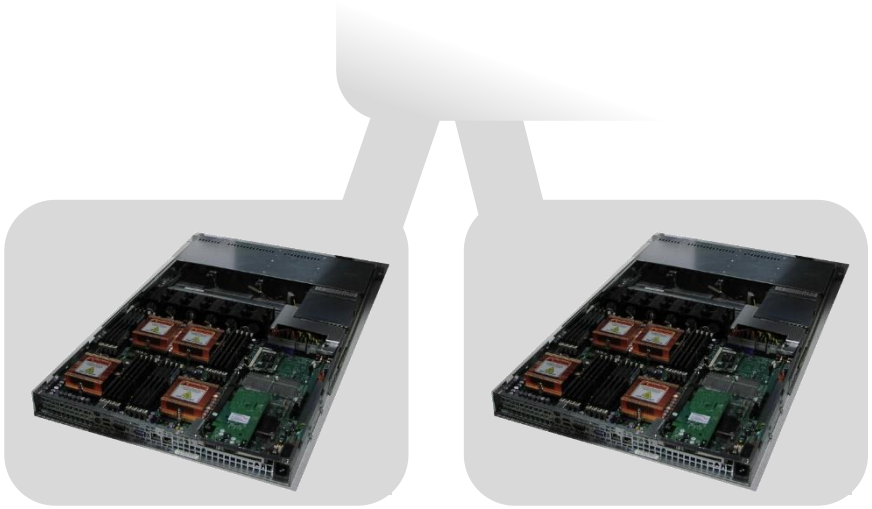


Lock Acquire by Readers



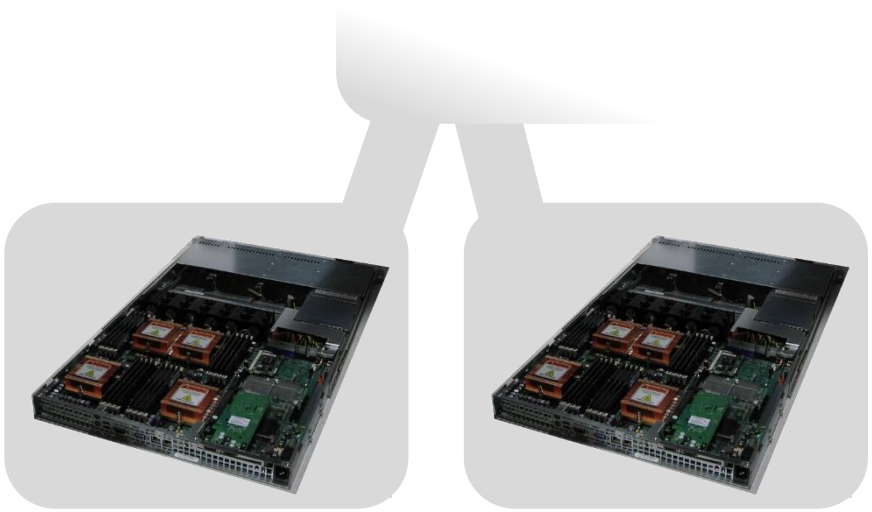
Lock Acquire by Readers

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



Lock Acquire by Readers

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



A writer holds the lock

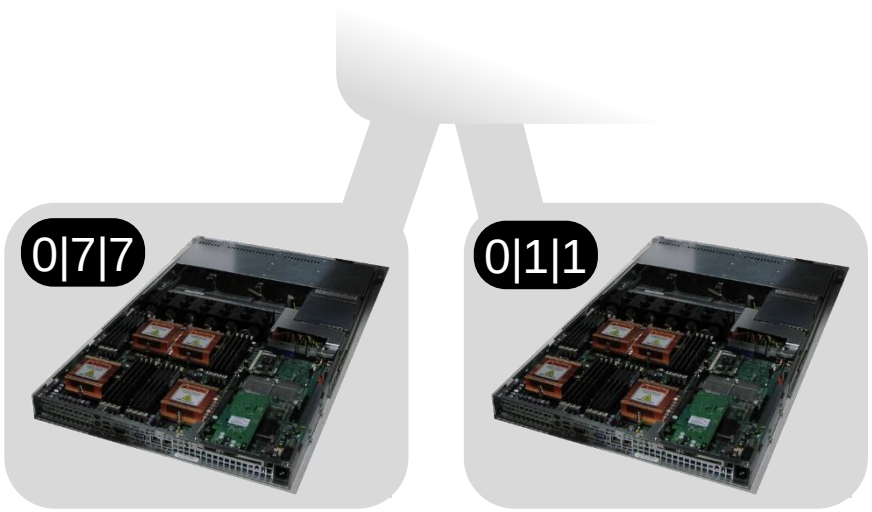
b|x|y

Readers that arrived at the CS

Readers that left the CS

Lock Acquire by Readers

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



A writer holds the lock

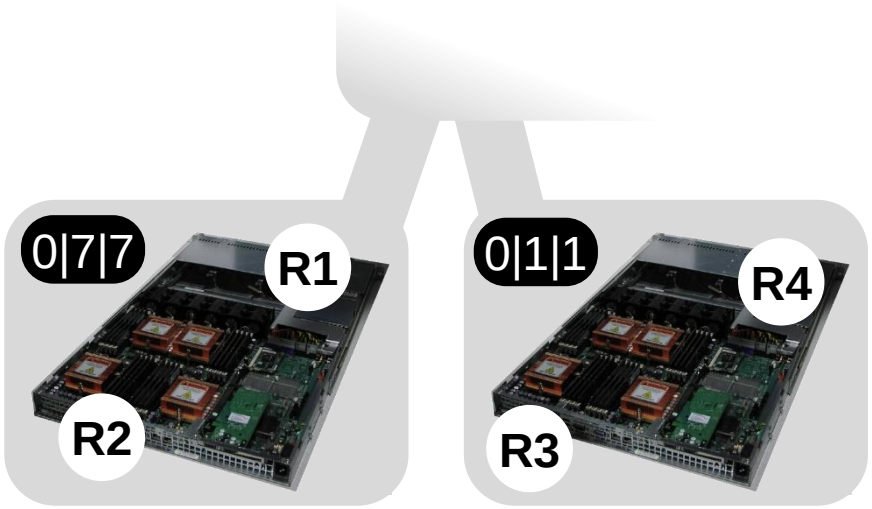
b|x|y

Readers that arrived at the CS

Readers that left the CS

Lock Acquire by Readers

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



A writer holds the lock

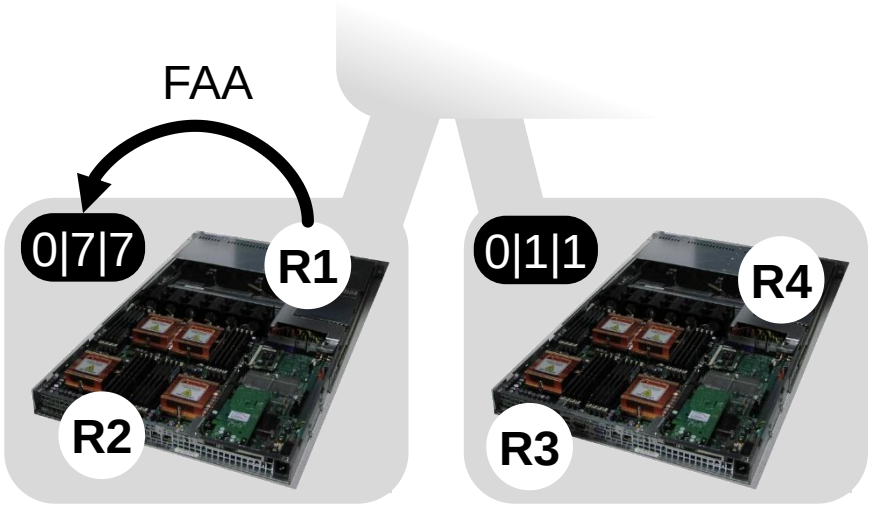
b|x|y

Readers that arrived at the CS

Readers that left the CS

Lock Acquire by Readers

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



A writer holds the lock

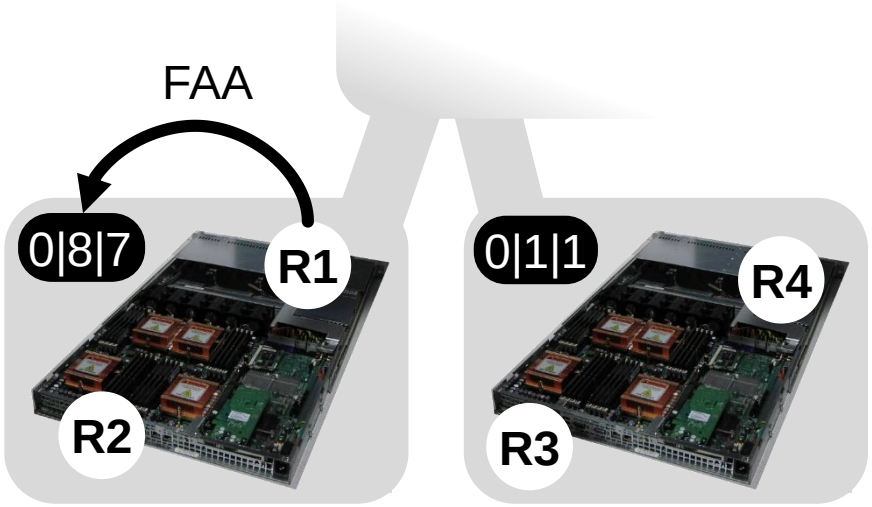
b|x|y

Readers that arrived at the CS

Readers that left the CS

Lock Acquire by Readers

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



A writer holds the lock

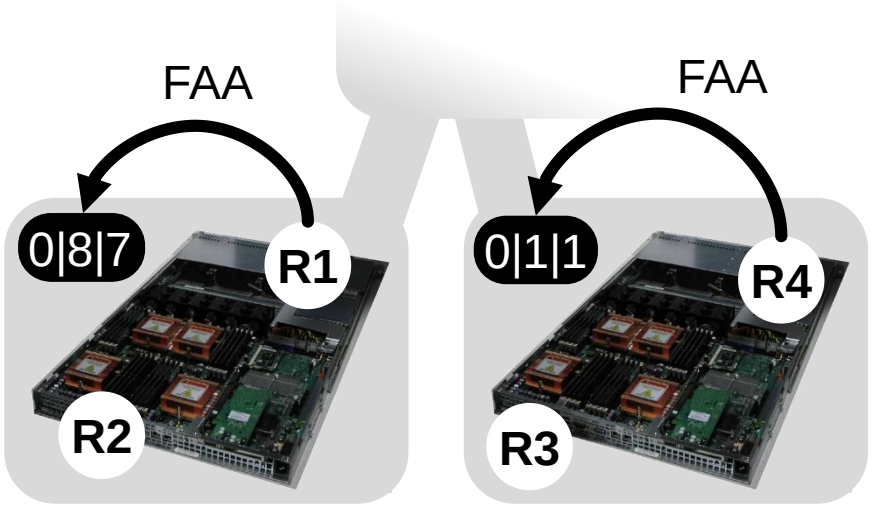
b|x|y

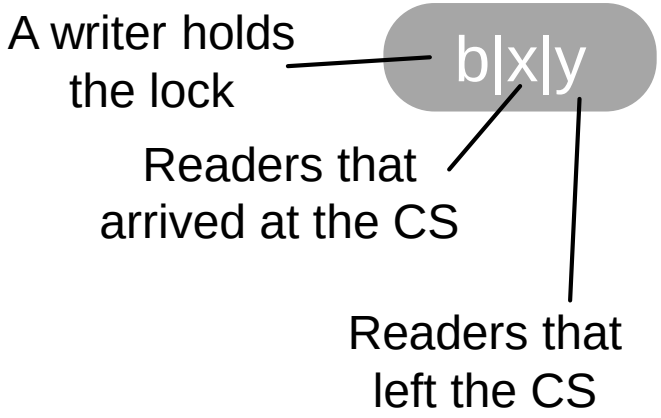
Readers that arrived at the CS

Readers that left the CS

Lock Acquire by Readers

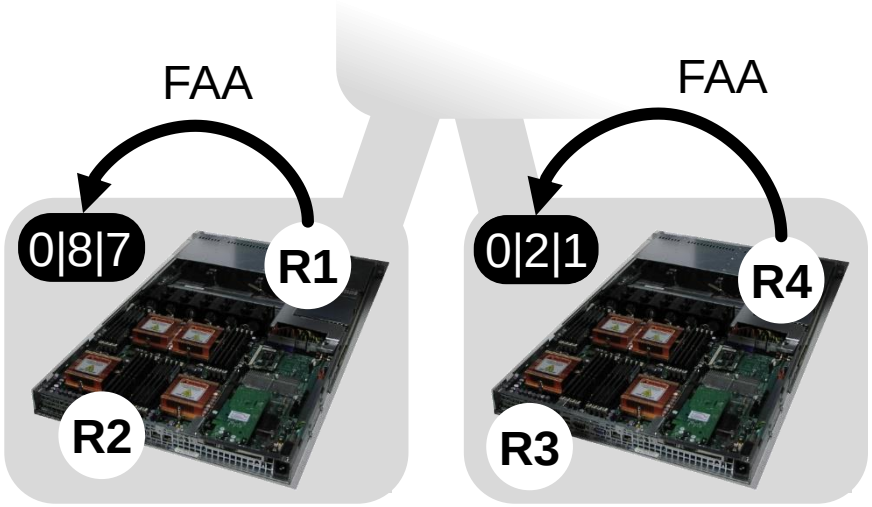
A lightweight acquire protocol for readers: only one atomic fetch-and-add (FAA) operation





Lock Acquire by Readers

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



A writer holds the lock

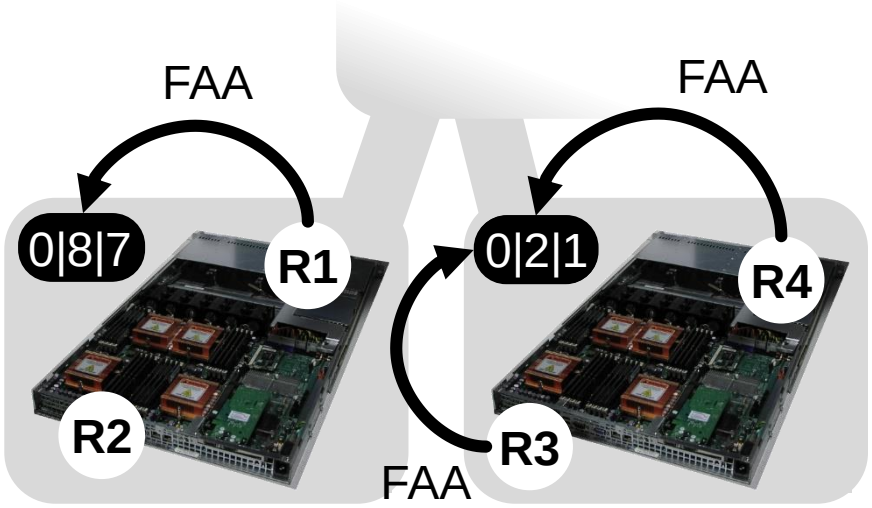
b|x|y

Readers that arrived at the CS

Readers that left the CS

Lock Acquire by Readers

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



A writer holds the lock

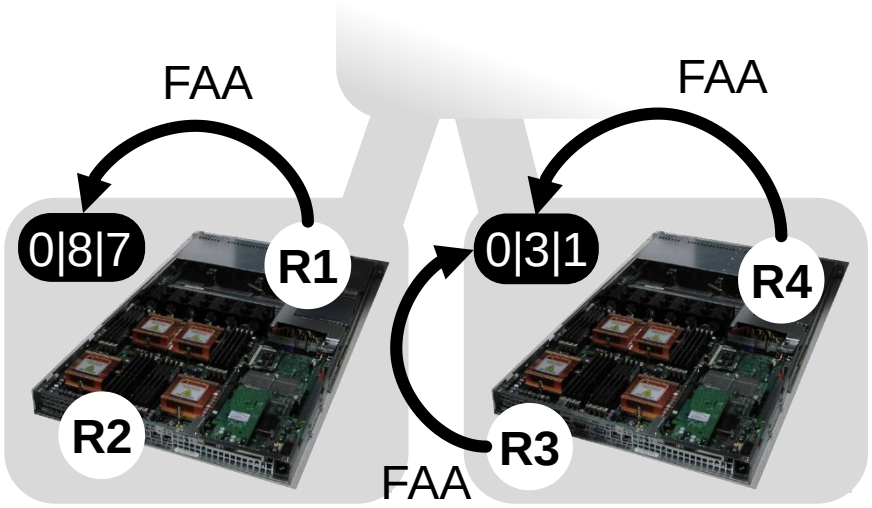
b|x|y

Readers that arrived at the CS

Readers that left the CS

Lock Acquire by Readers

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



A writer holds the lock

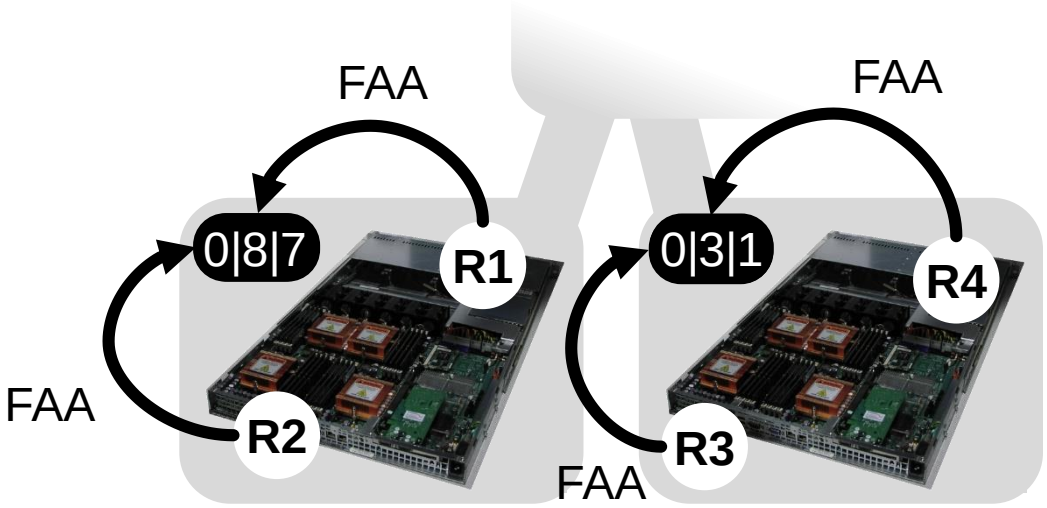
b|x|y

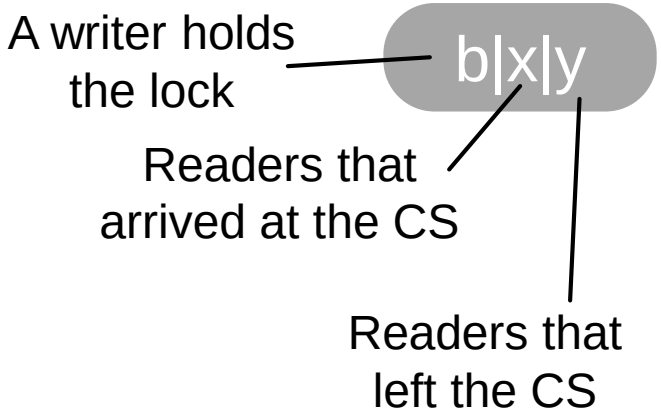
Readers that arrived at the CS

Readers that left the CS

Lock Acquire by Readers

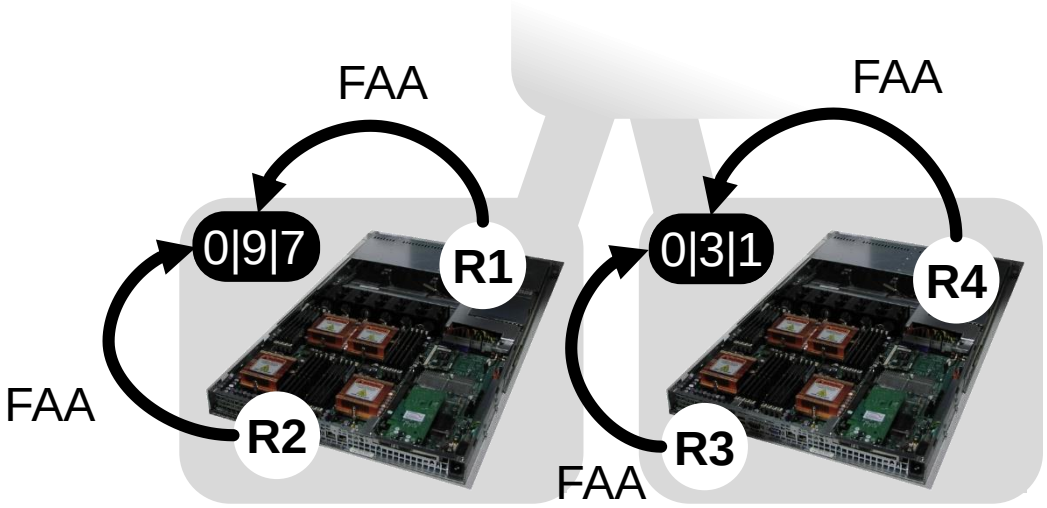
A lightweight acquire protocol for readers: only one atomic fetch-and-add (FAA) operation

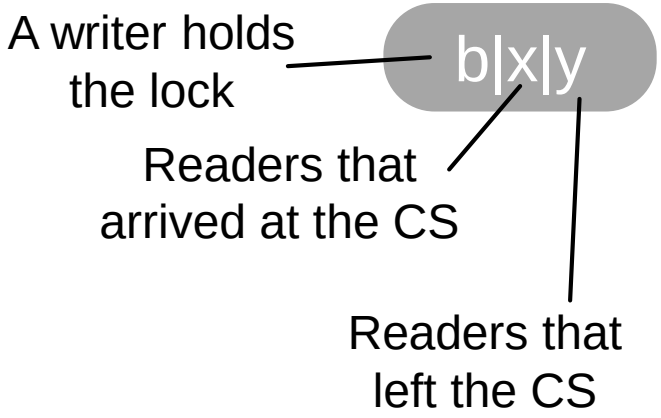




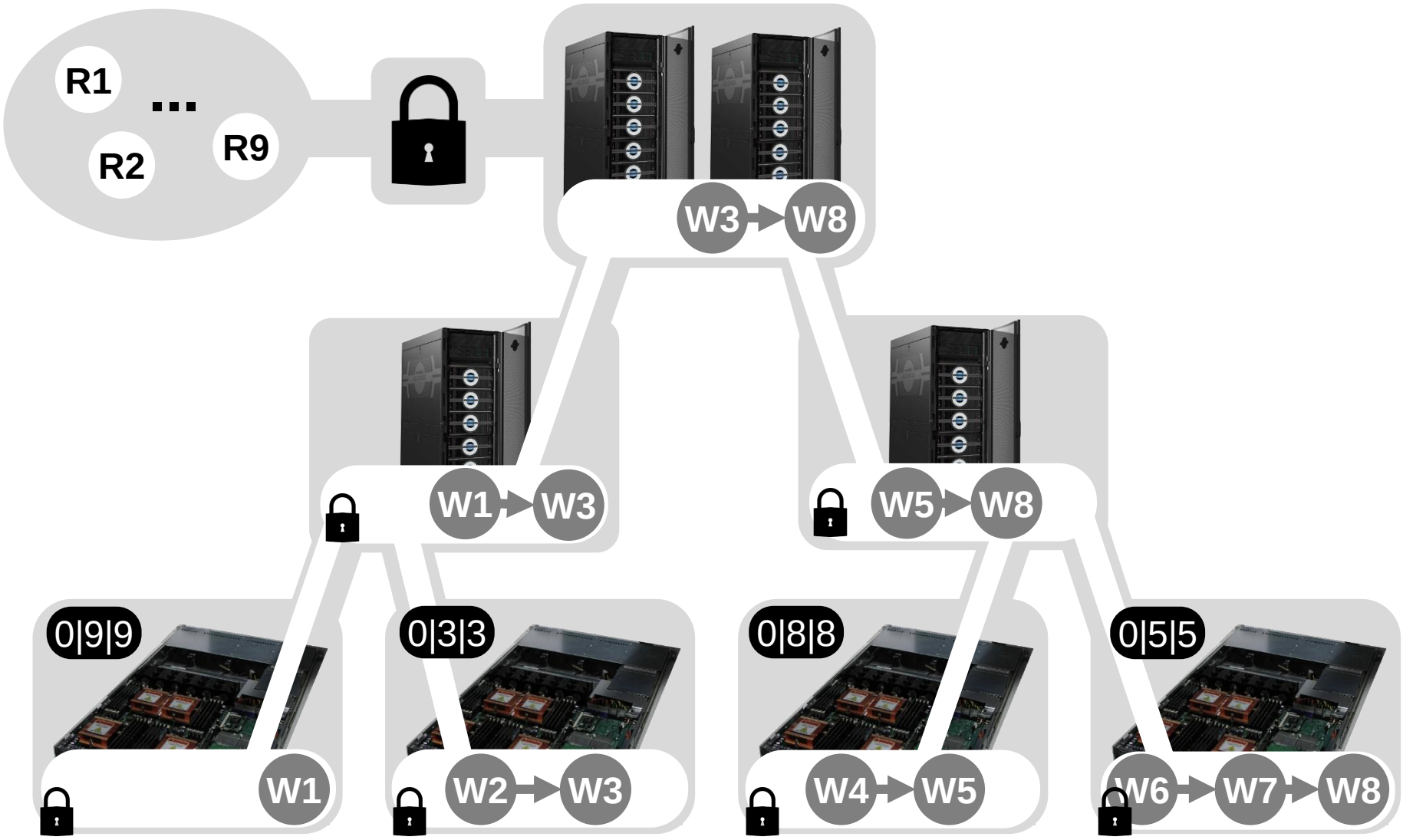
Lock Acquire by Readers

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

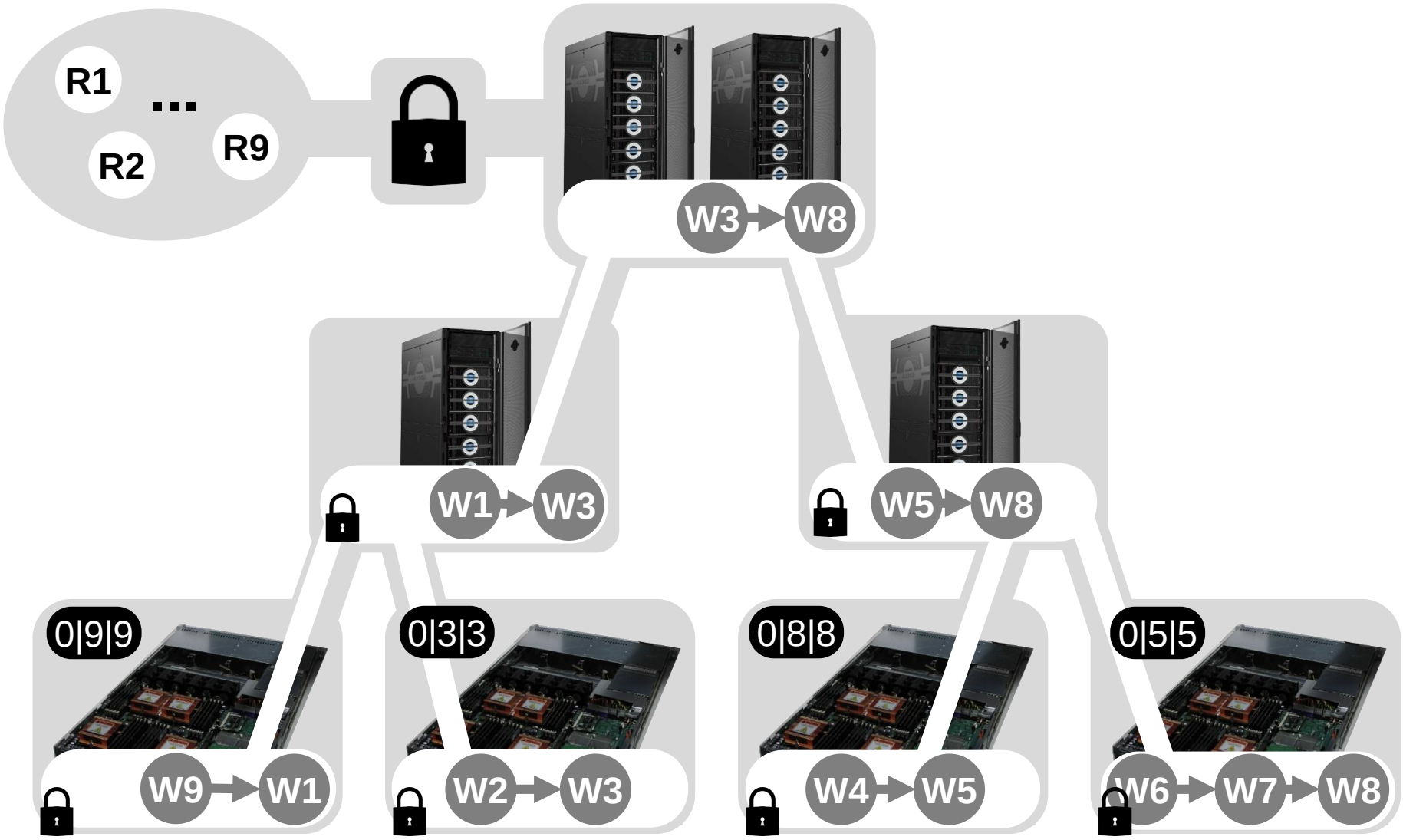




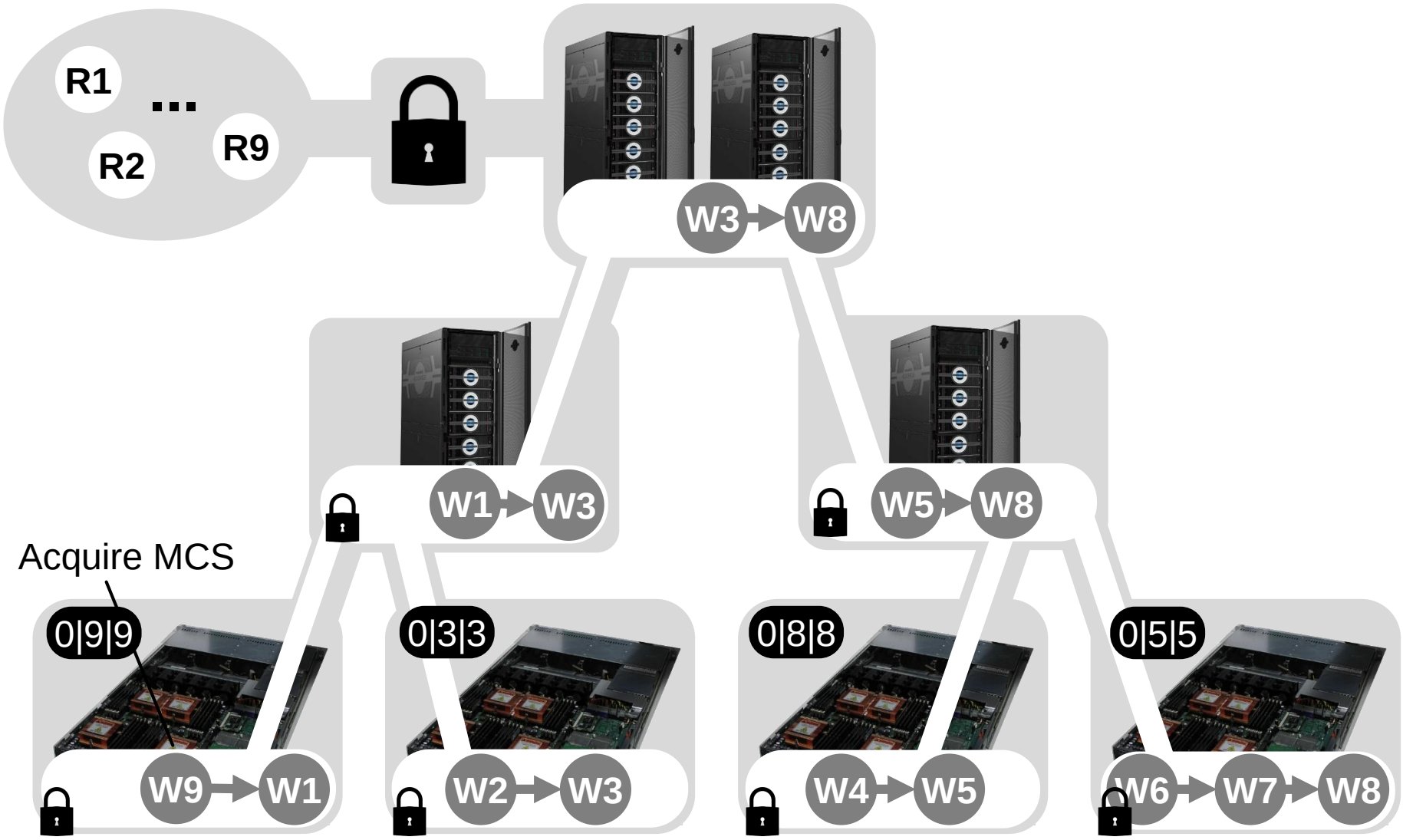
Lock Acquire by Writers



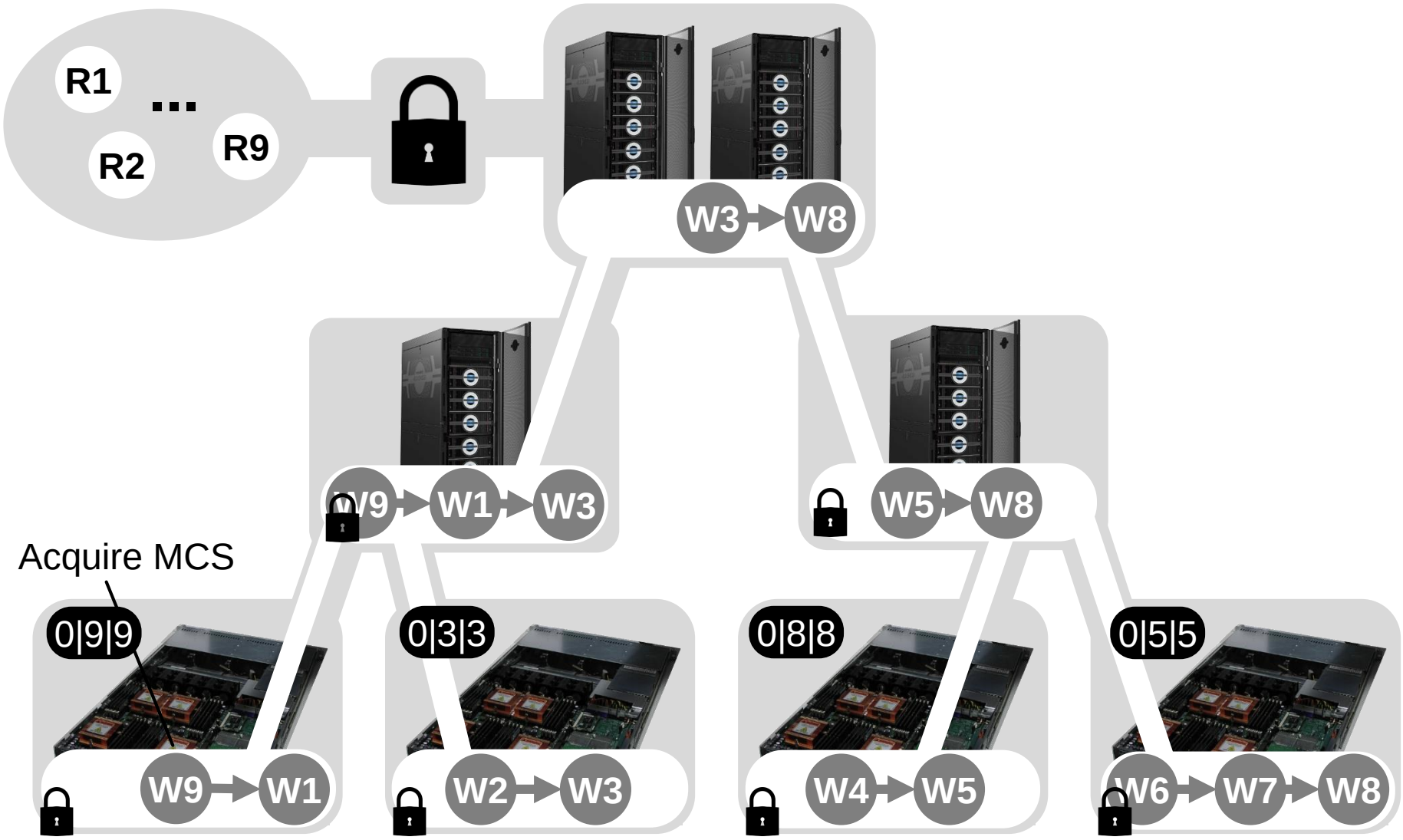
Lock Acquire by Writers



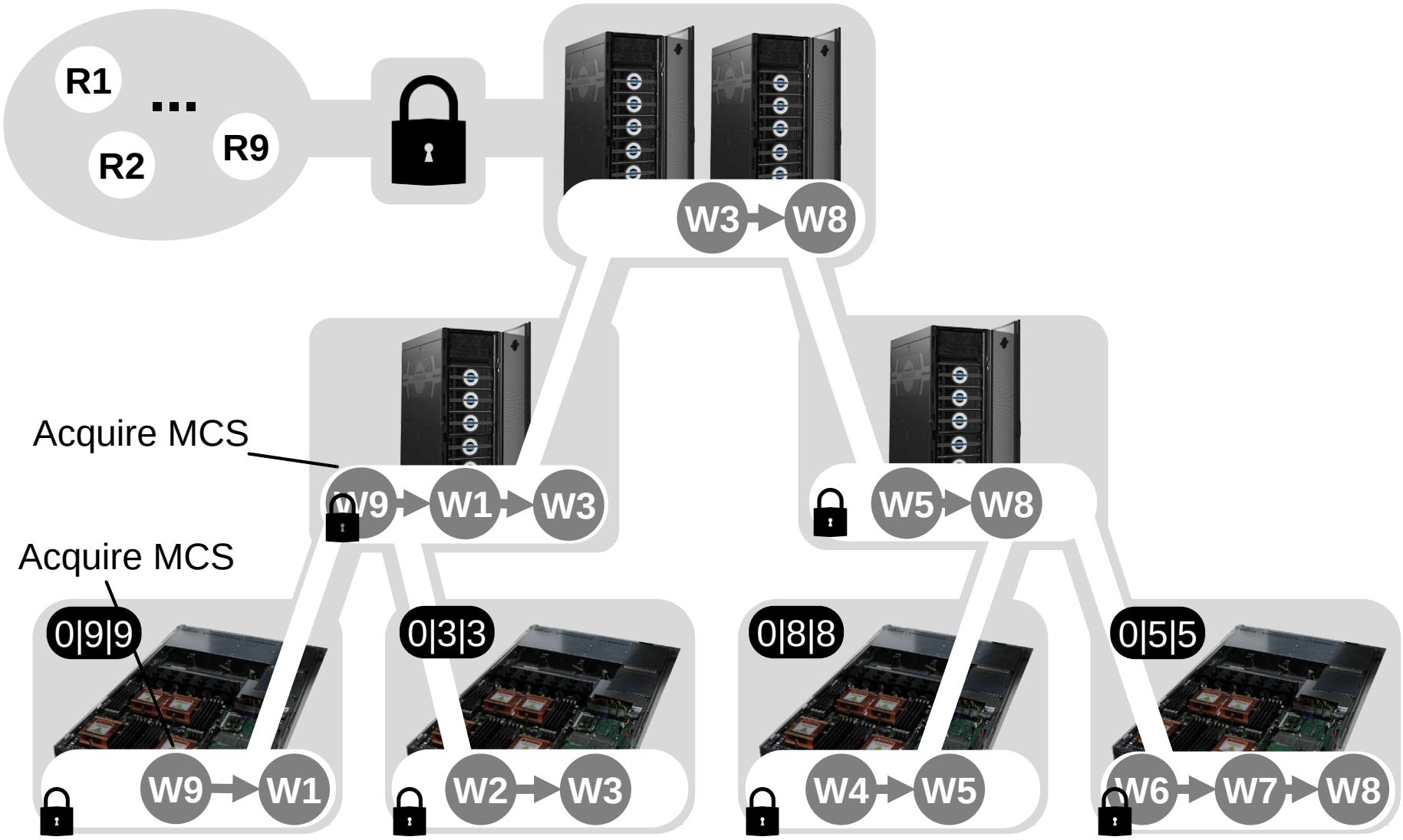
Lock Acquire by Writers



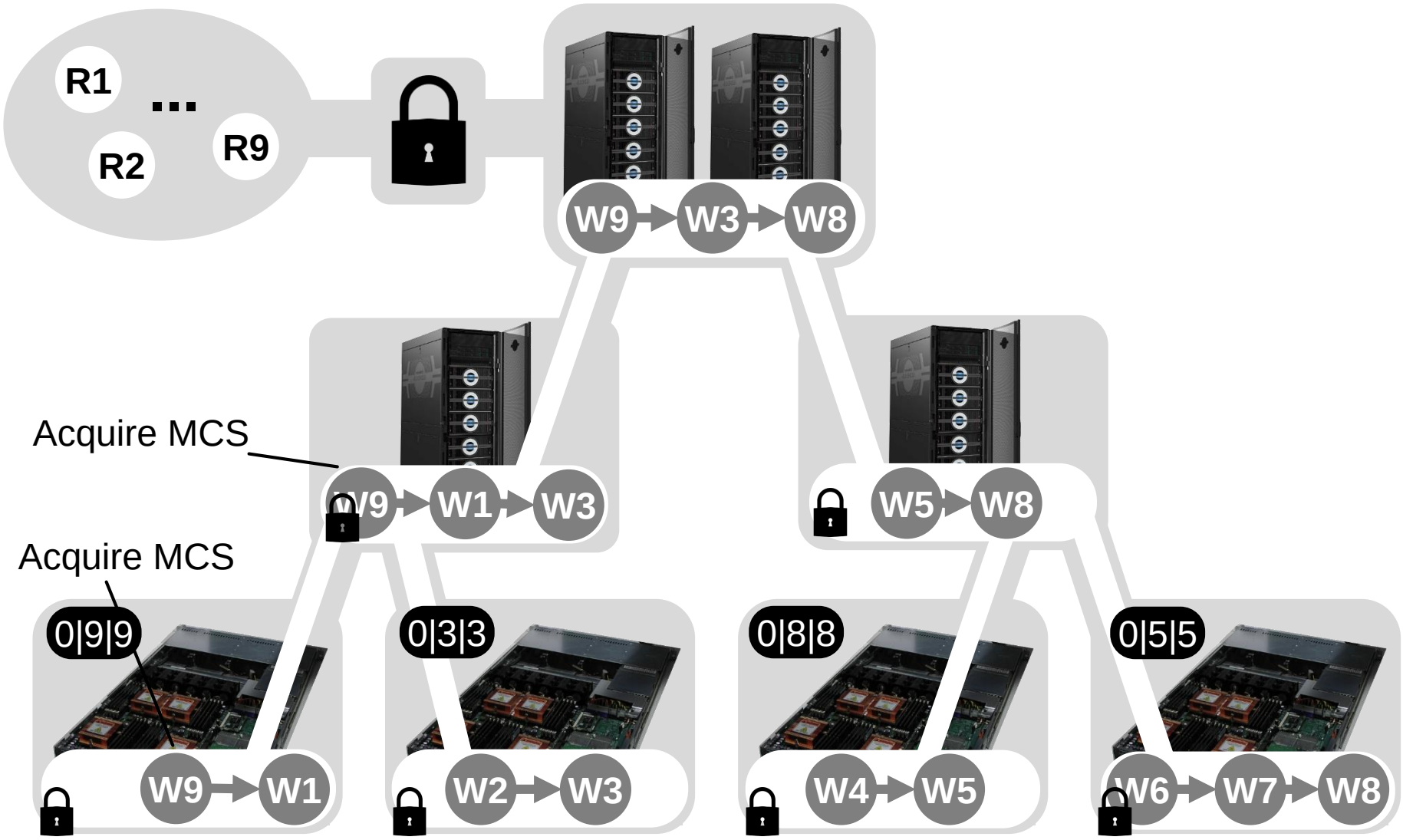
Lock Acquire by Writers



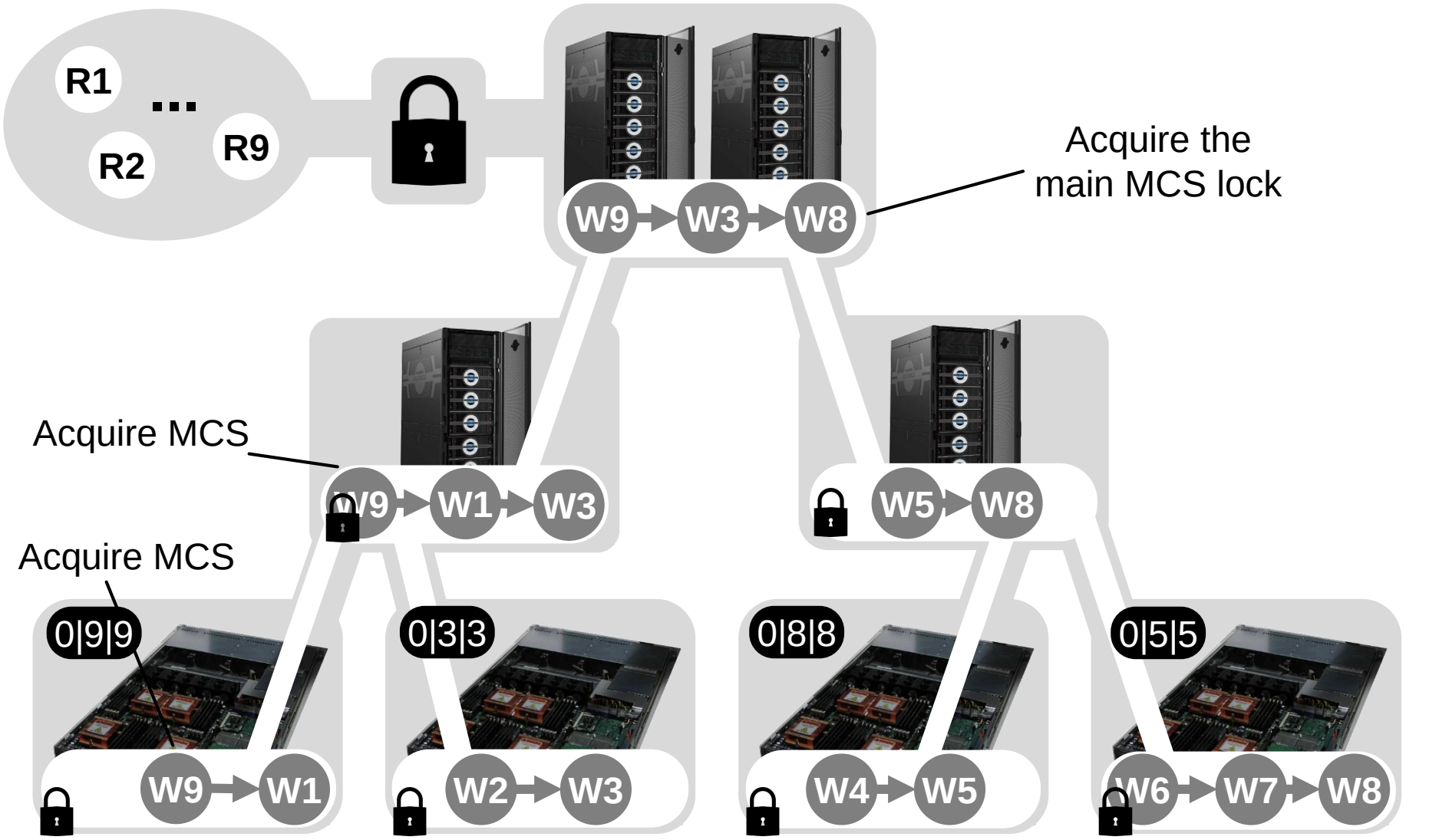
Lock Acquire by Writers



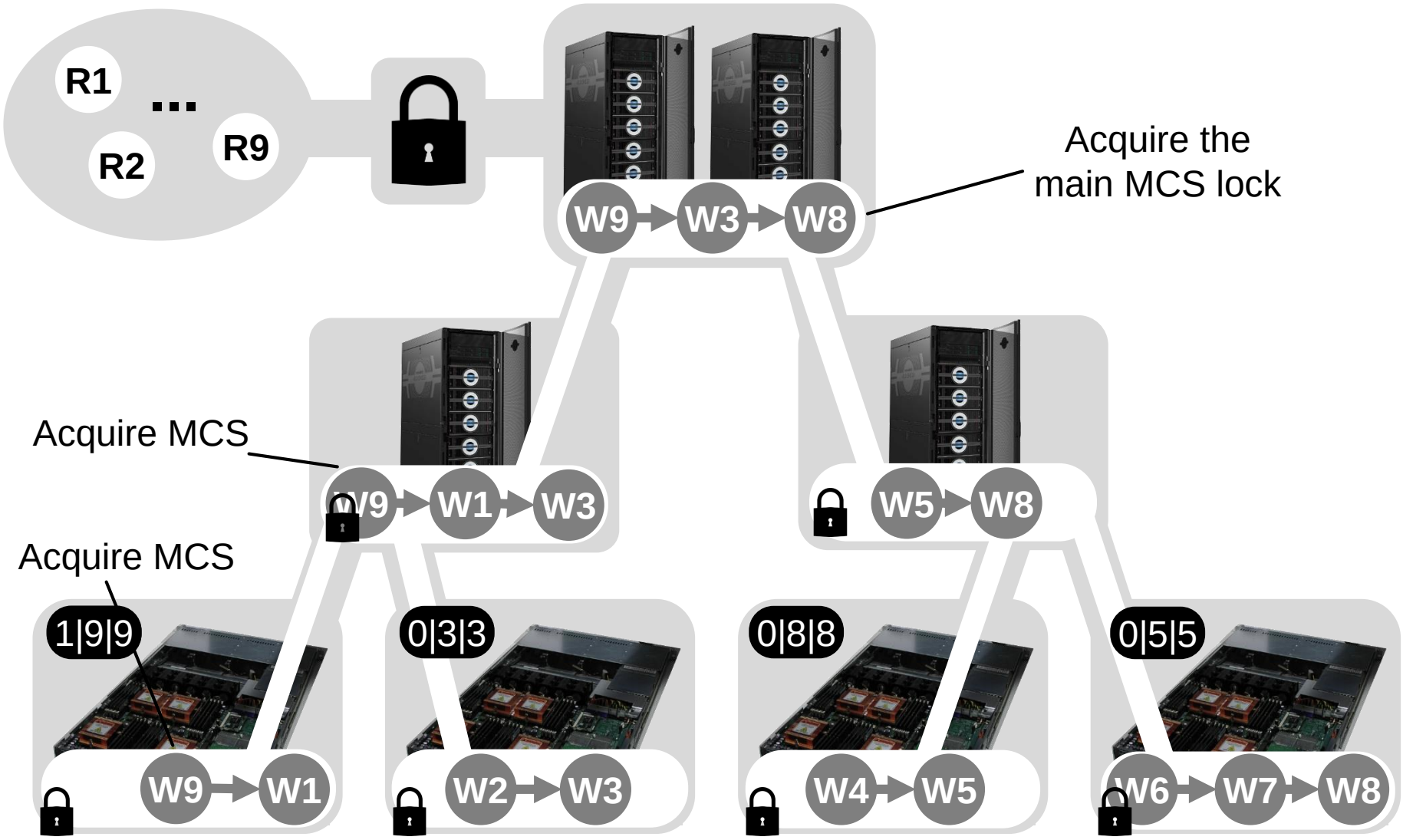
Lock Acquire by Writers



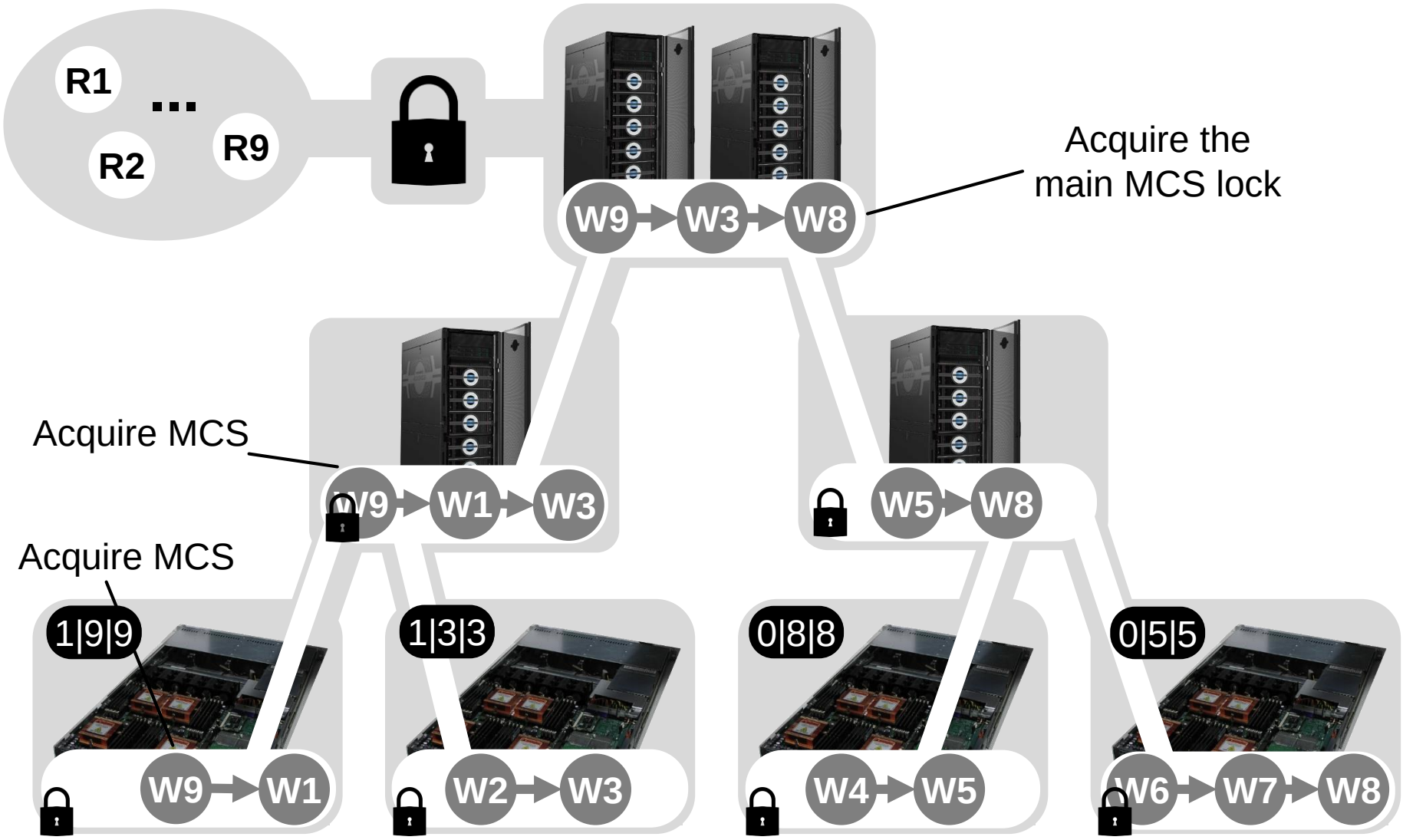
Lock Acquire by Writers



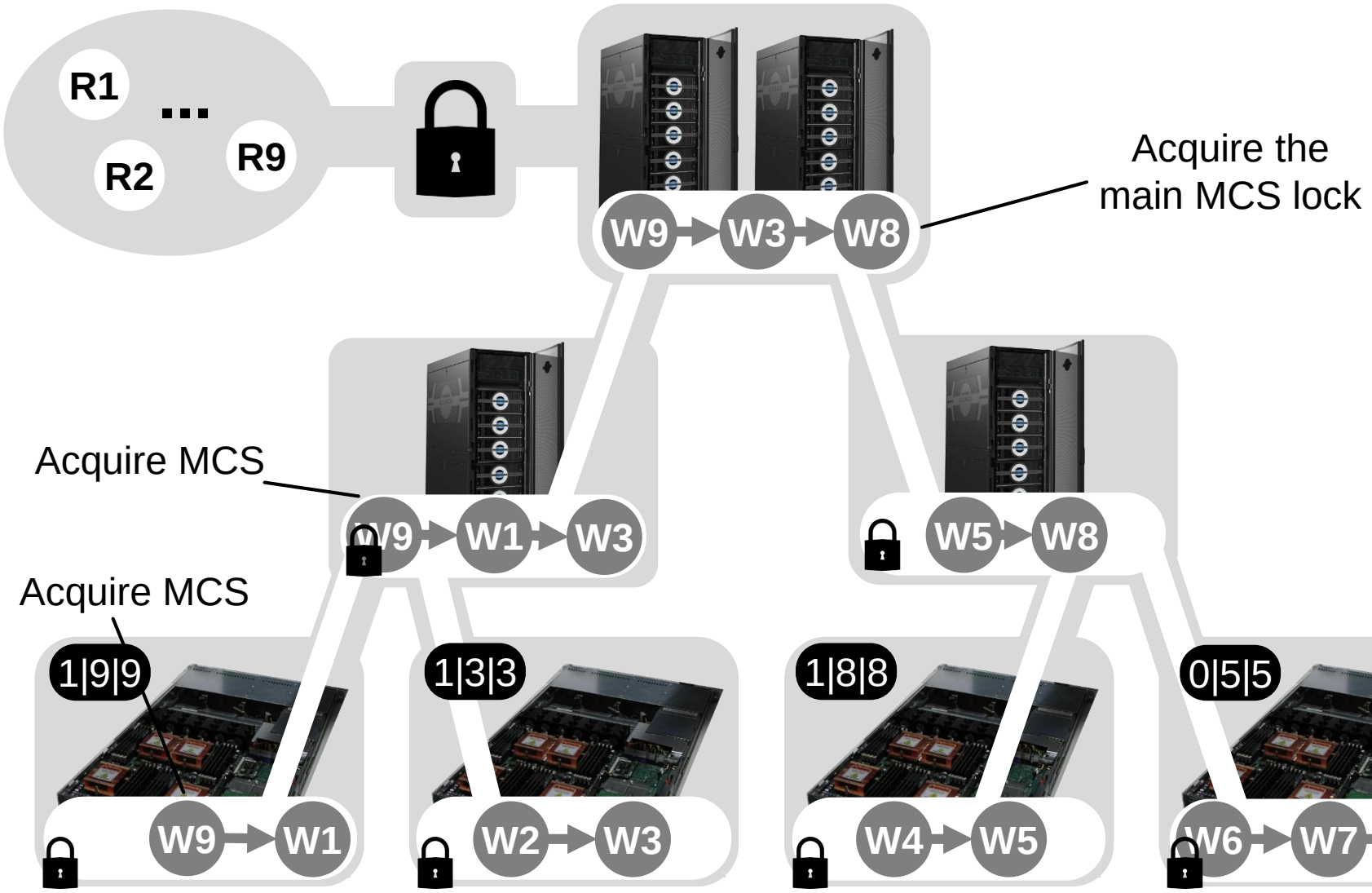
Lock Acquire by Writers



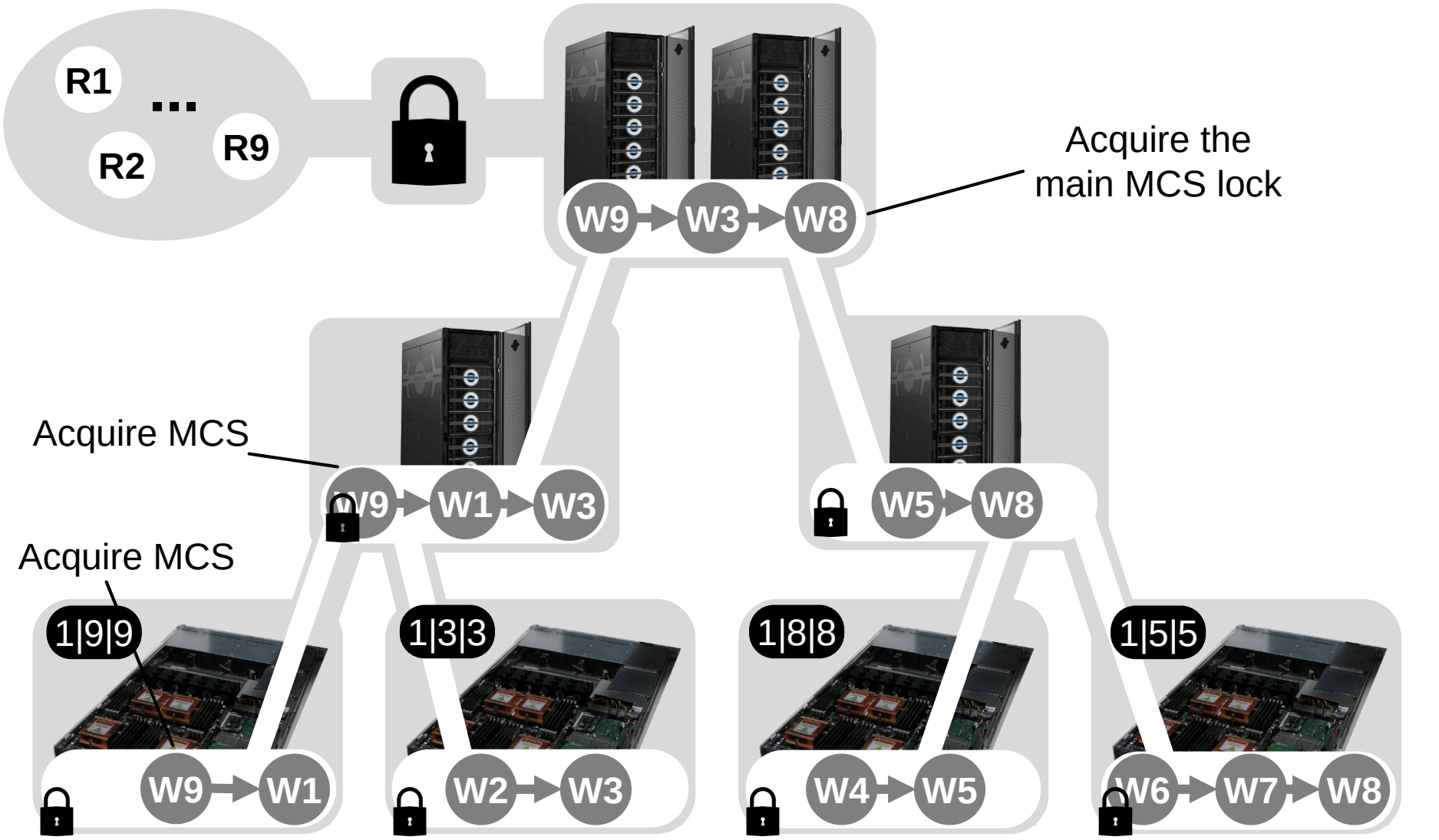
Lock Acquire by Writers



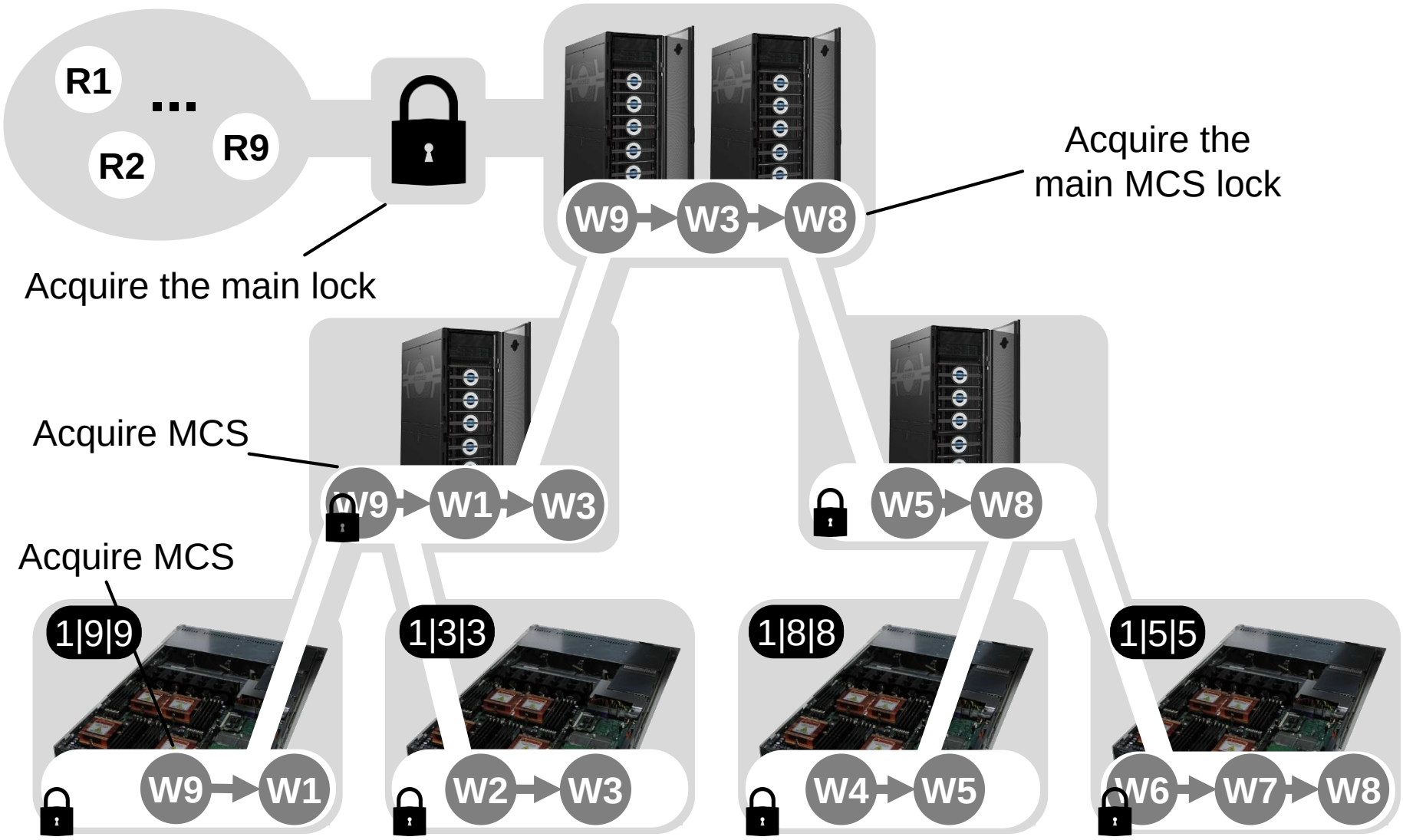
LOCK ACQUIRE BY WRITERS



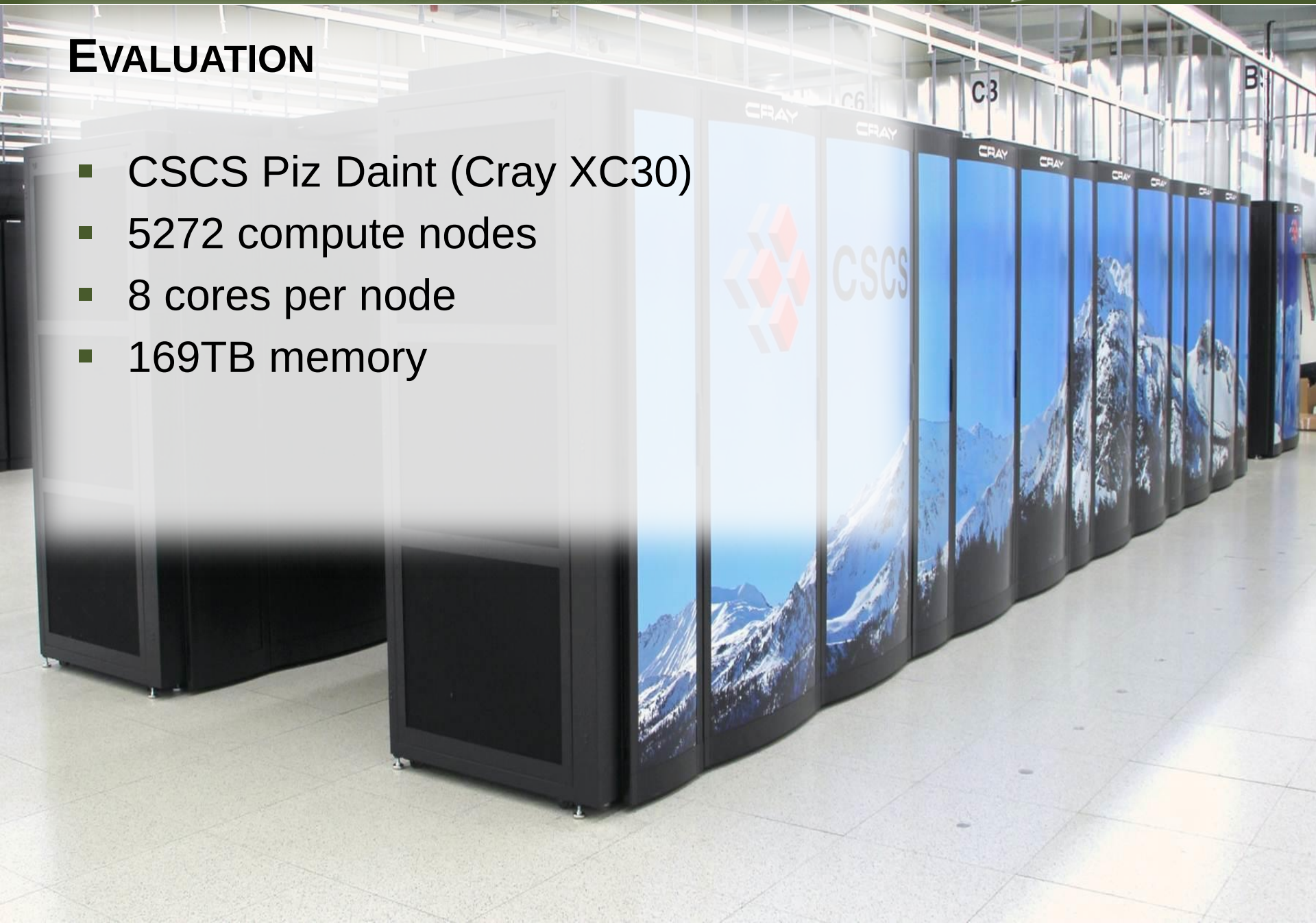
LOCK ACQUIRE BY WRITERS



Lock Acquire by Writers



EVALUATION



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

Evaluation
Considered Benchmarks

Evaluation
Considered Benchmarks

The **latency** benchmark

Evaluation
Considered Benchmarks

The **latency** benchmark

Throughput benchmarks:

- Empty-critical-section
- Single-operation
- Wait-after-release
- Workload-critical-section

Evaluation
Considered Benchmarks

The **latency** benchmark

Throughput benchmarks:

- Empty-critical-section
- Single-operation
- Wait-after-release
- Workload-critical-section

DHT

Distributed hashtable evaluation

EVALUATION

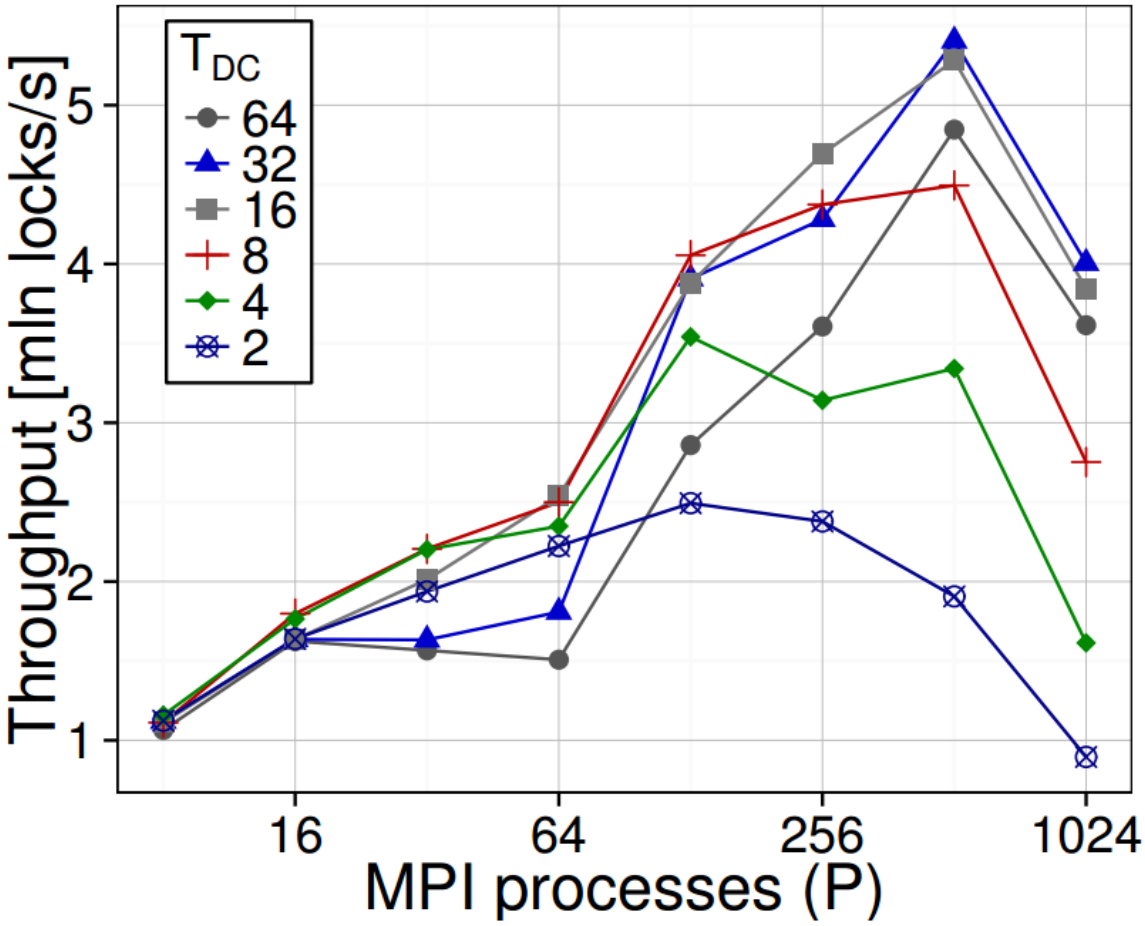
DISTRIBUTED COUNTER ANALYSIS

0|9|7 0|3|1

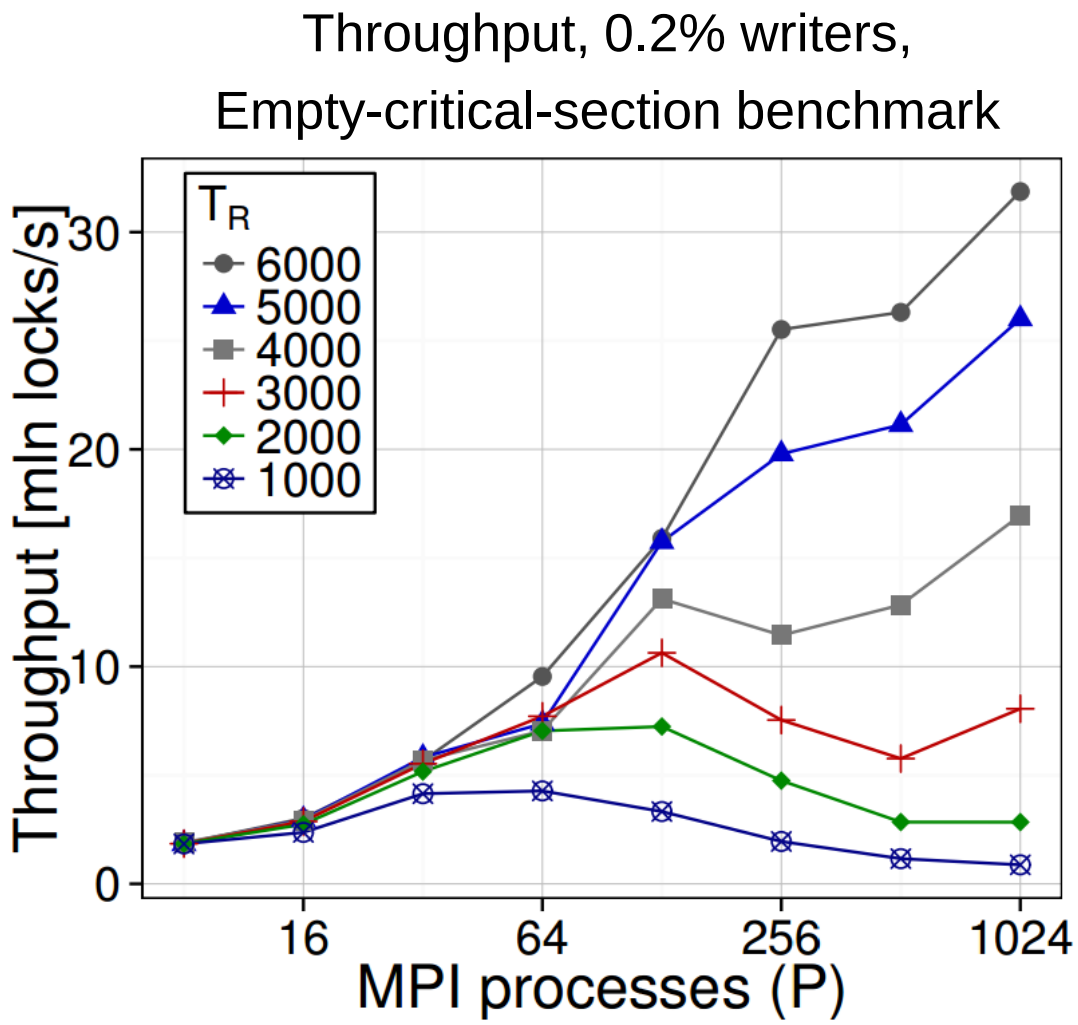
0|12|8

Throughput, 2% writers

Single-operation benchmark

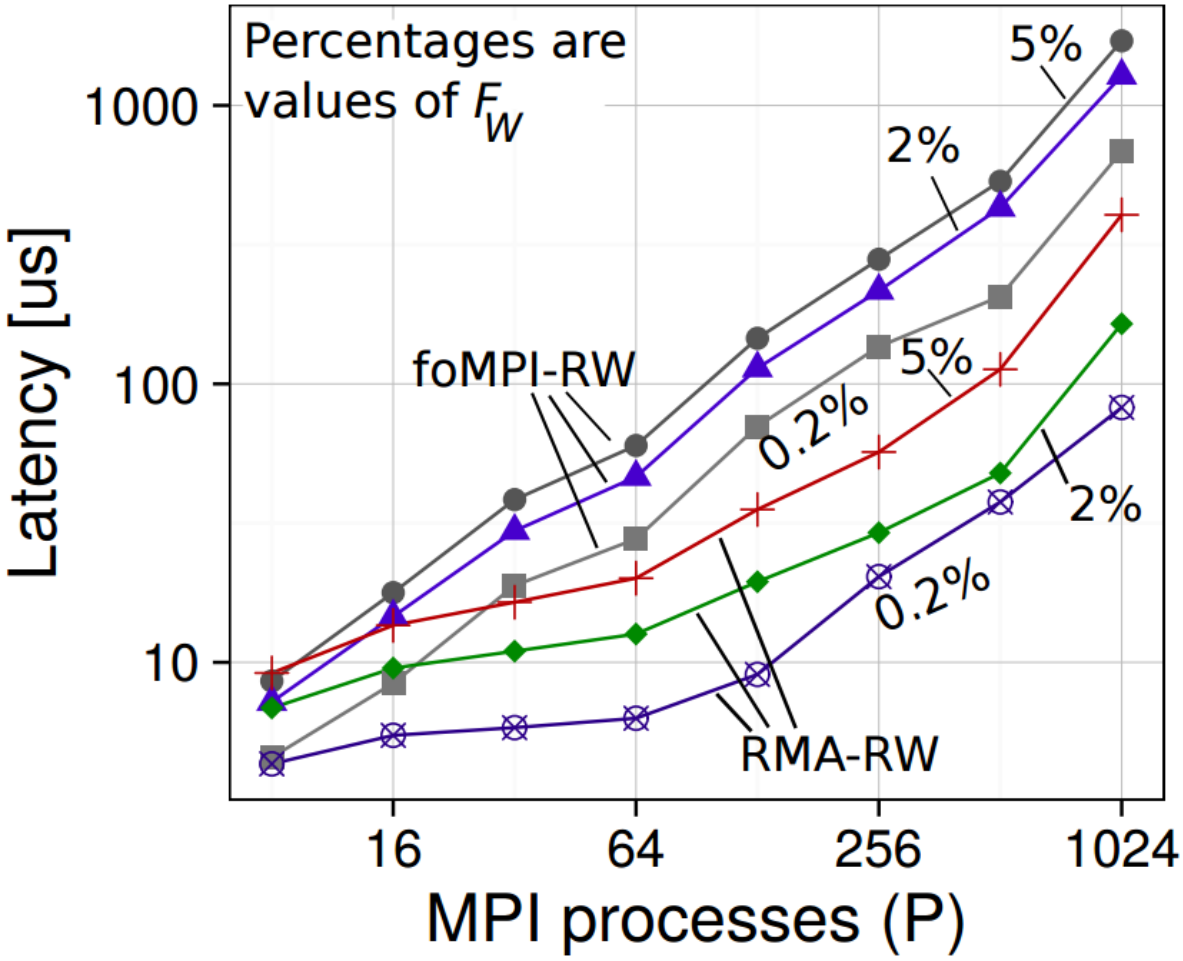


EVALUATION
READER THRESHOLD ANALYSIS



EVALUATION

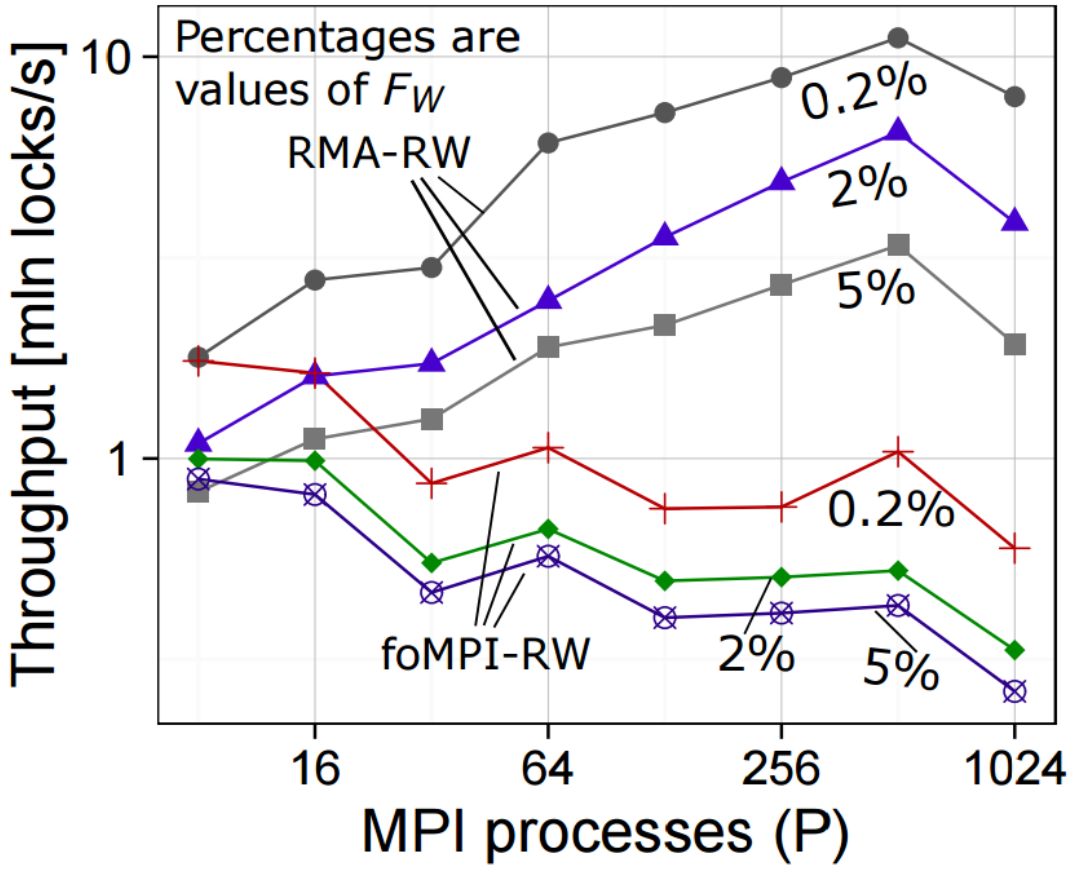
COMPARISON TO THE STATE-OF-THE-ART



[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

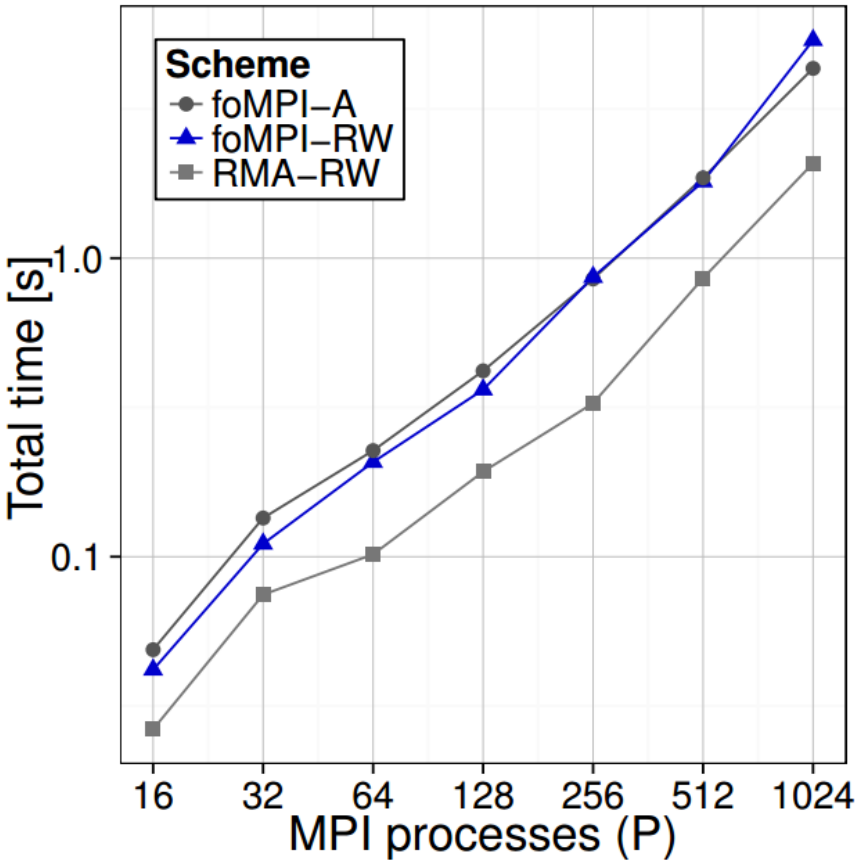
Throughput, single-operation benchmark



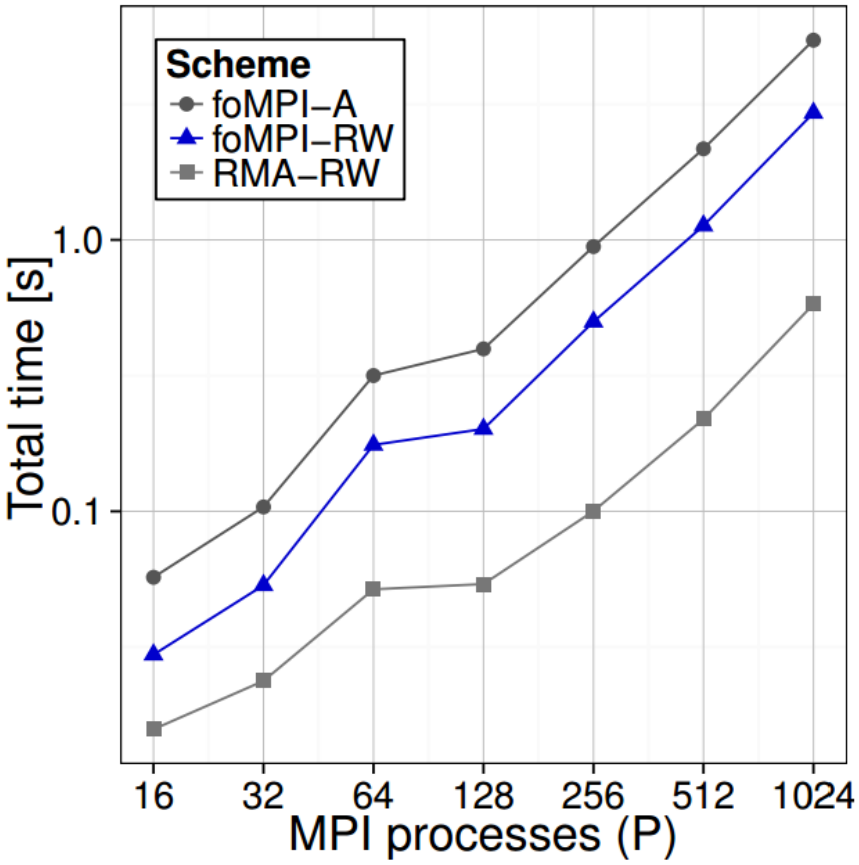
[1] R. Gerstenberger et al. Enabling Highly-scalable Remote Memory Access Programming with MPI-3 One Sided. ACM/IEEE Supercomputing 2013.

EVALUATION
DISTRIBUTED HASHTABLE

20% writers



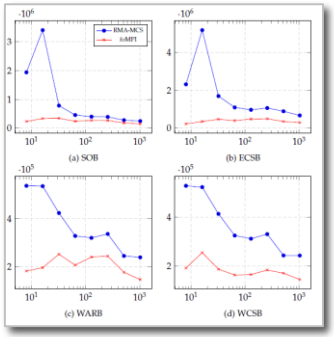
10% writers

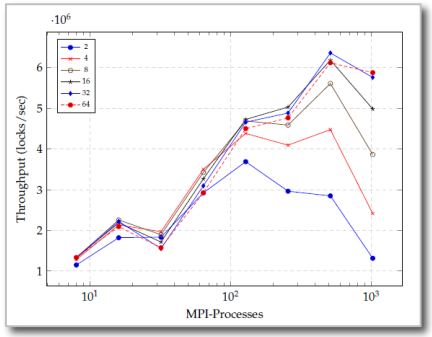


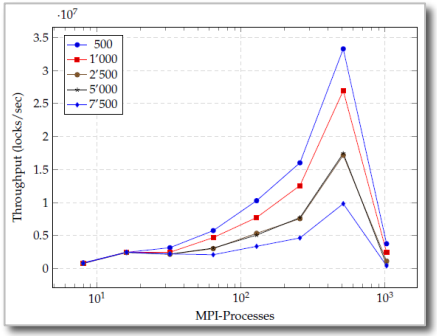
[1] R. Gerstenberger et al. Enabling Highly-scalable Remote Memory Access Programming with MPI-3 One Sided. ACM/IEEE Supercomputing 2013.

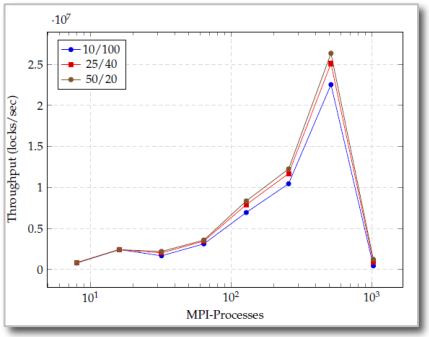
Other Analyses

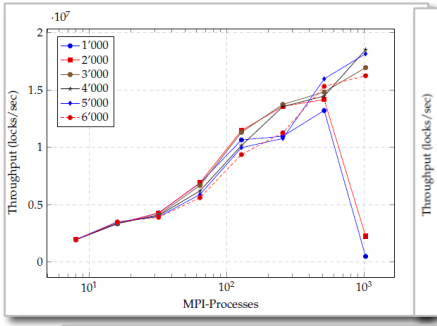
OTHER ANALYSES

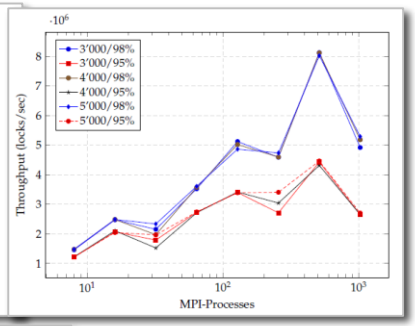


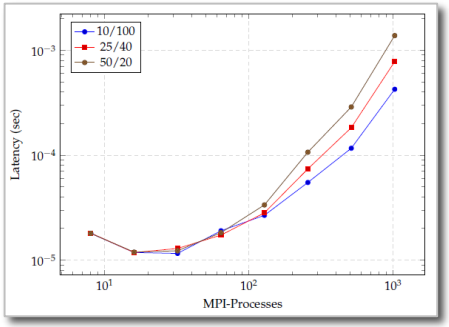


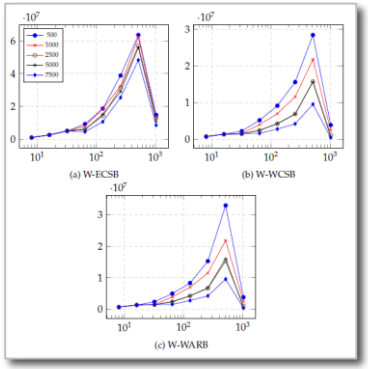


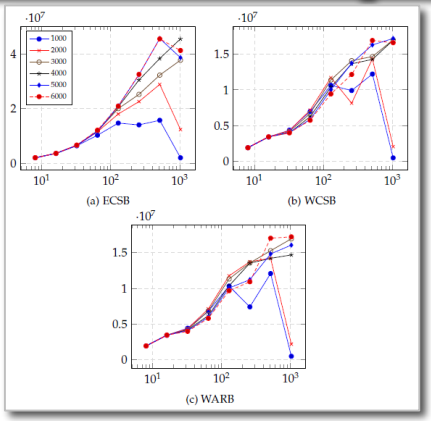


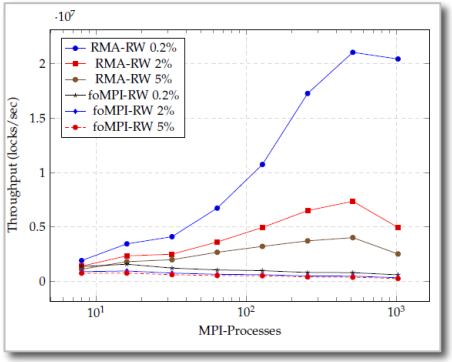


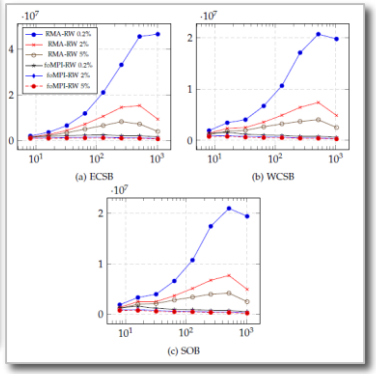


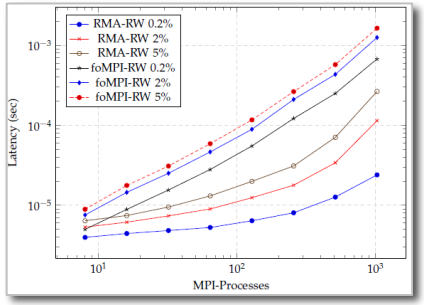






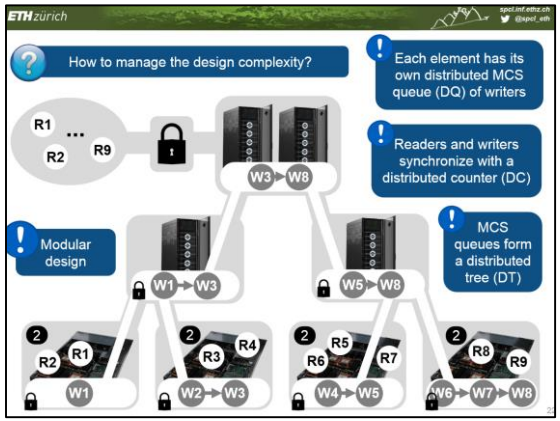






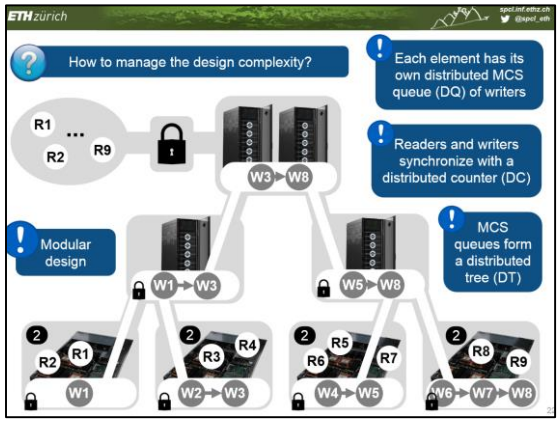
CONCLUSIONS

CONCLUSIONS

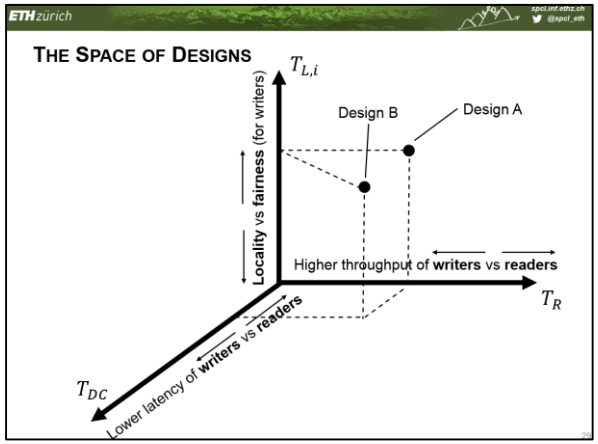


Modular distributed RMA lock, correctness with SPIN

CONCLUSIONS

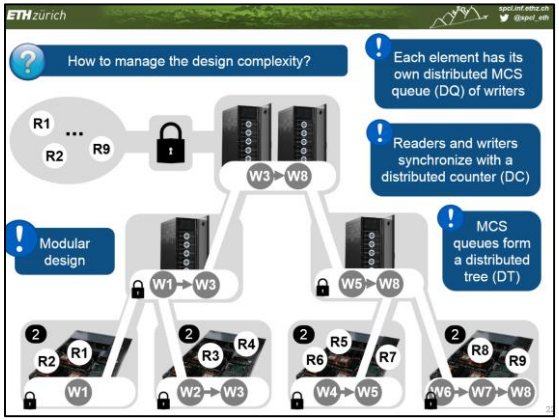


Modular distributed RMA lock, correctness with SPIN

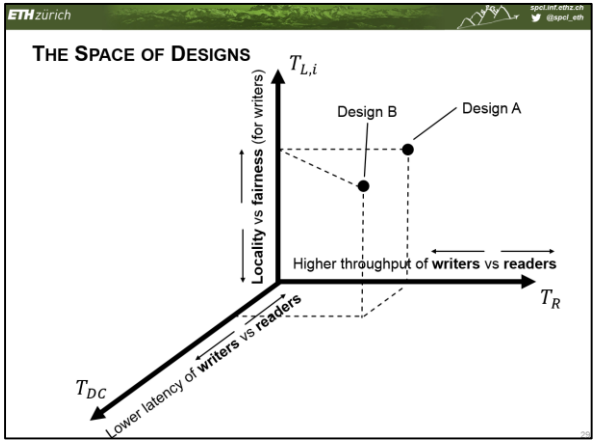


Parameter-based design, feasible with various RMA libs/languages

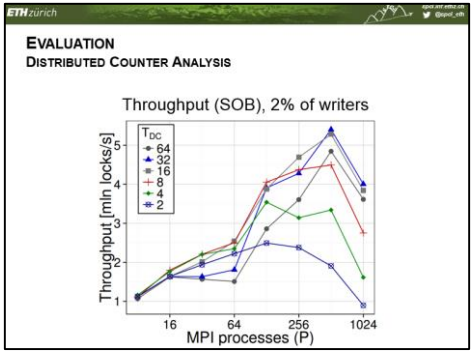
CONCLUSIONS

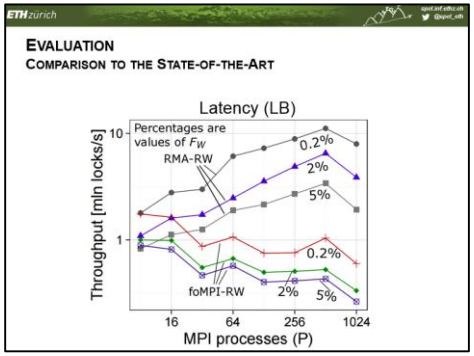


Modular distributed RMA lock, correctness with SPIN



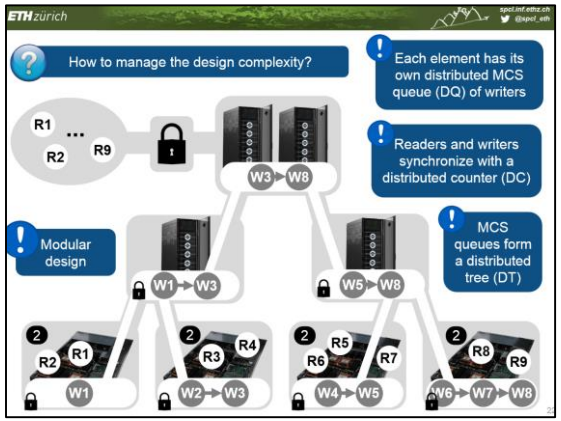
Parameter-based design, feasible with various RMA libs/languages



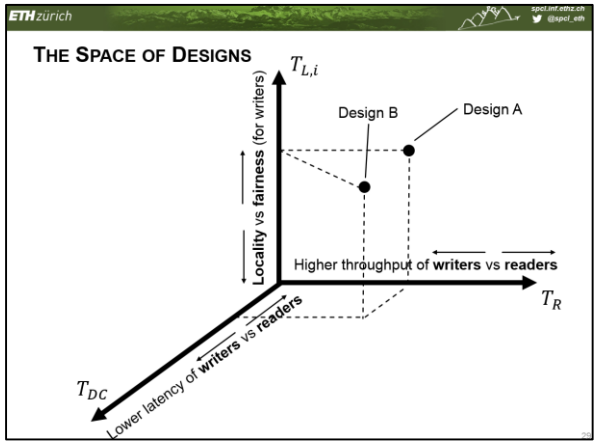


Improves latency and throughput over state-of-the-art

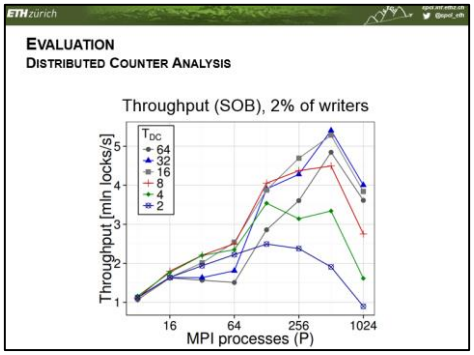
CONCLUSIONS

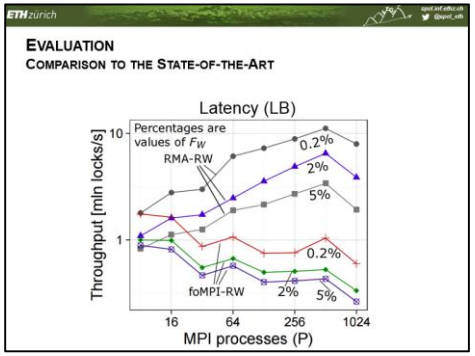


Modular distributed RMA lock, correctness with SPIN

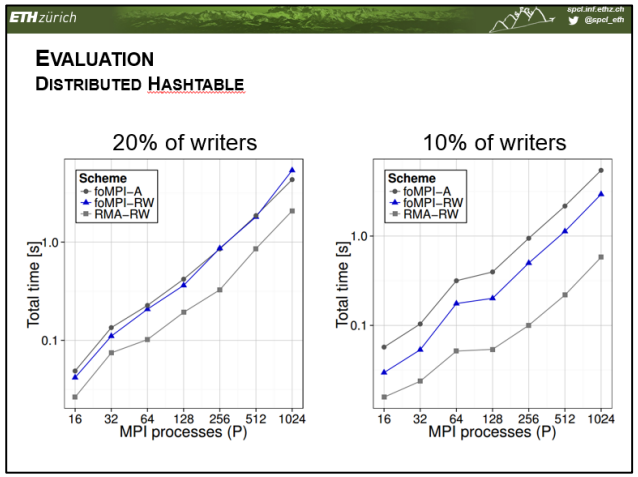


Parameter-based design, feasible with various RMA libs/languages



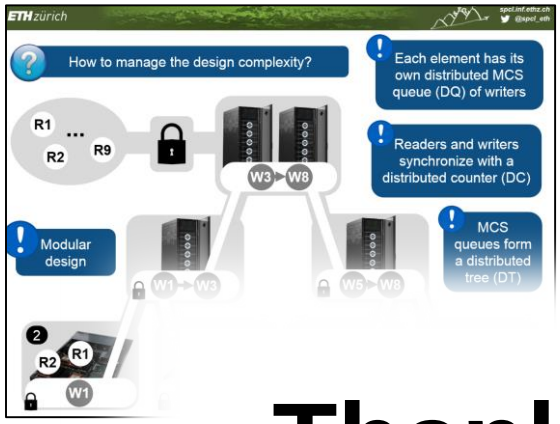


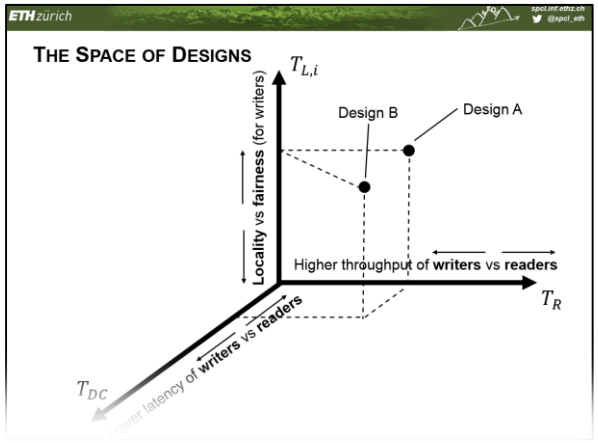
Improves latency and throughput over state-of-the-art



Enables high-performance distributed hashtabled

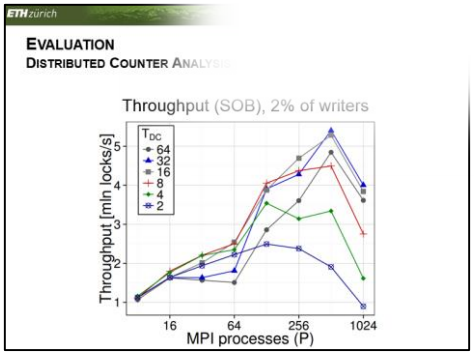
CONCLUSIONS



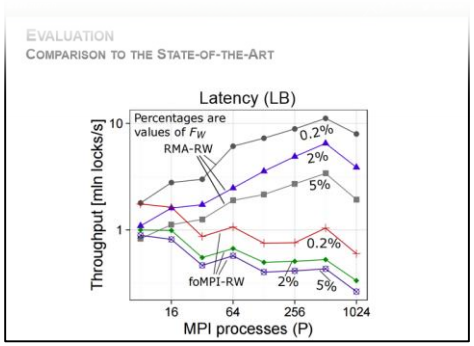


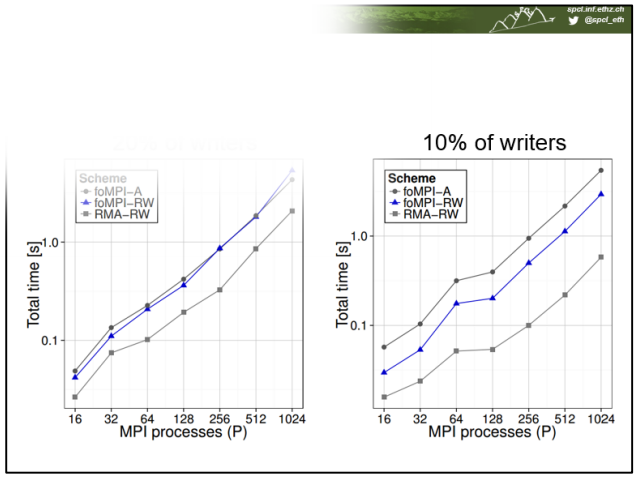
Modular d… corre… …ible …es

Thank you for your attention



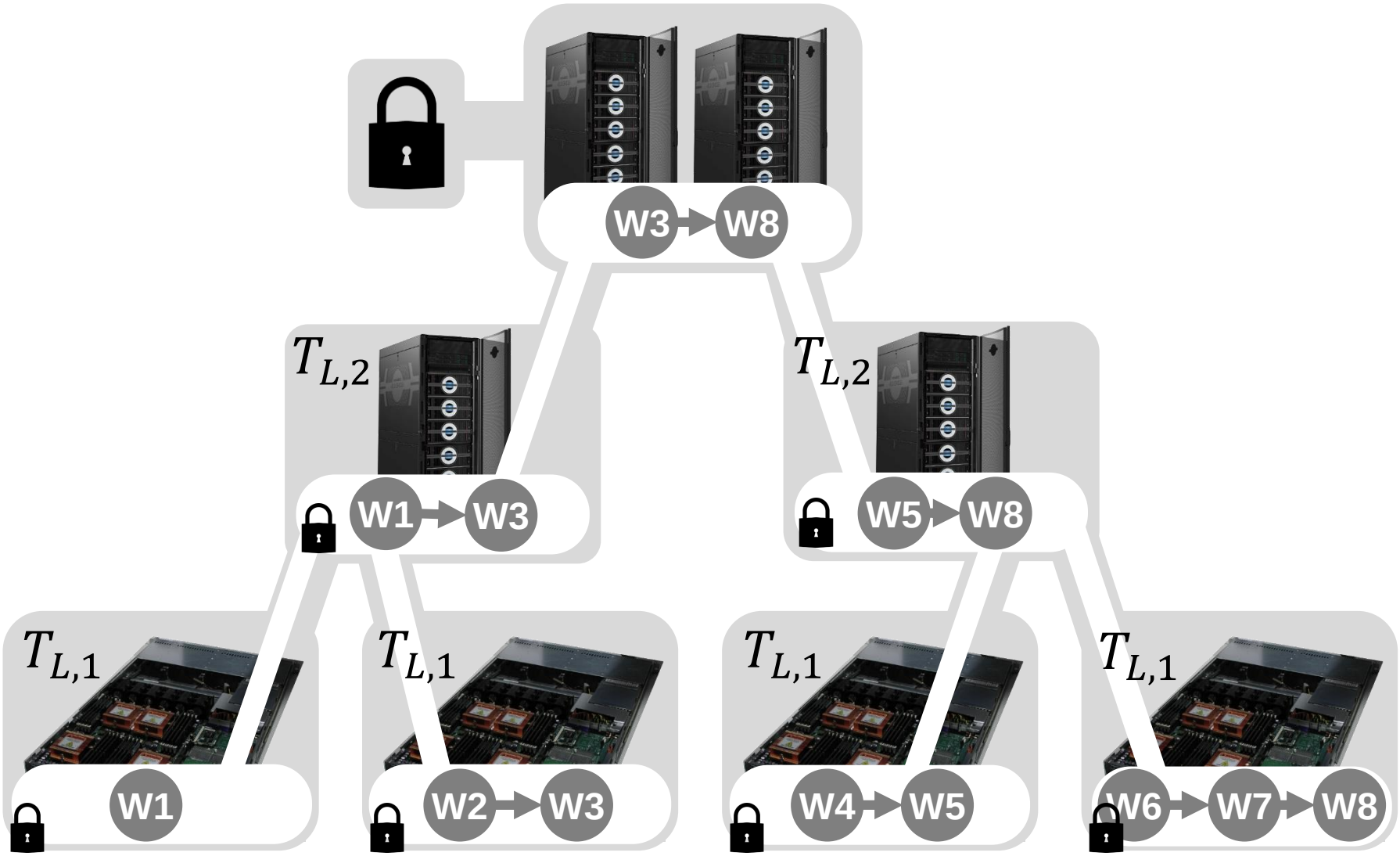
Improves latency and throughput over state-of-the-art





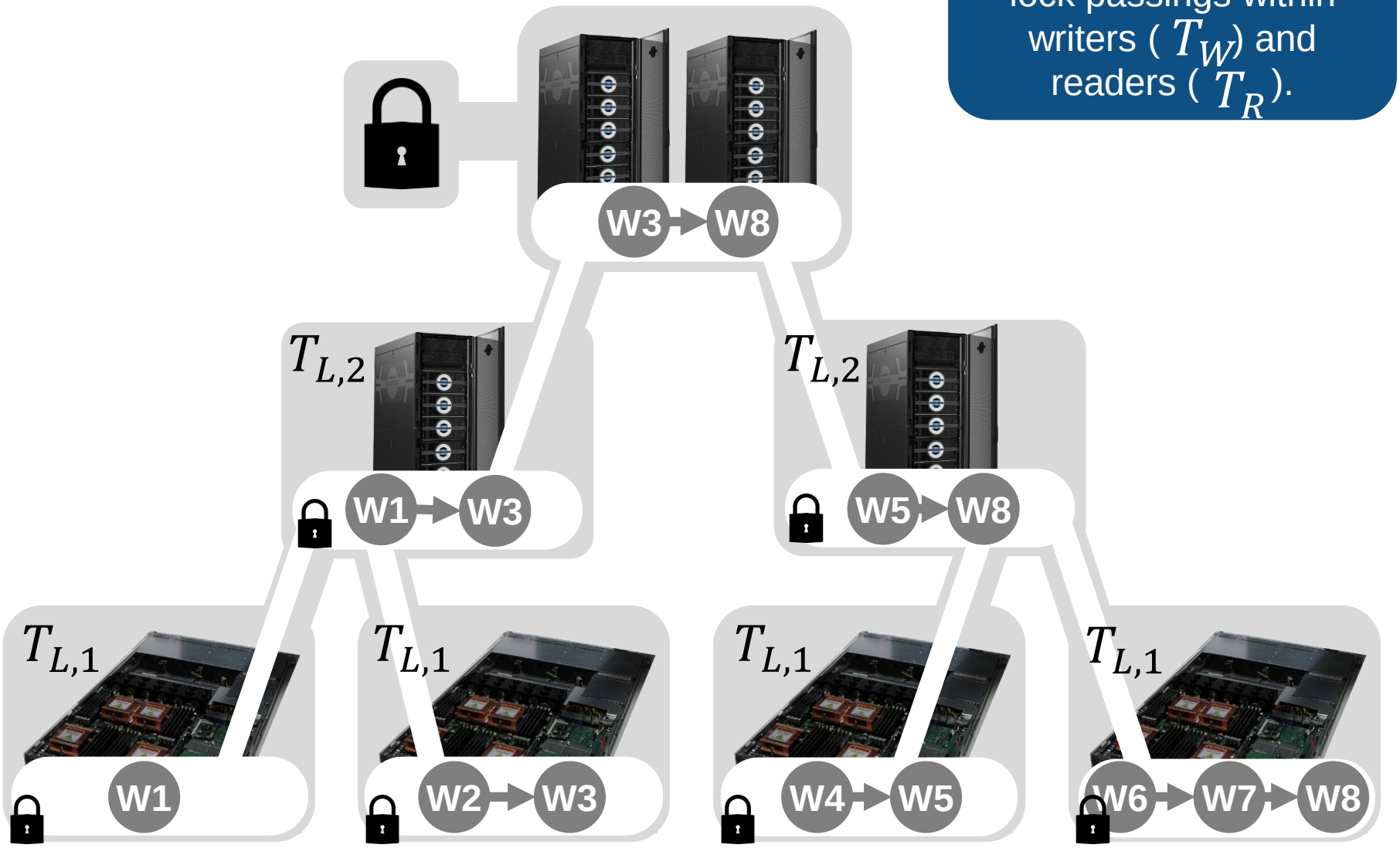
Enables high-performance distributed hashtabled

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



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

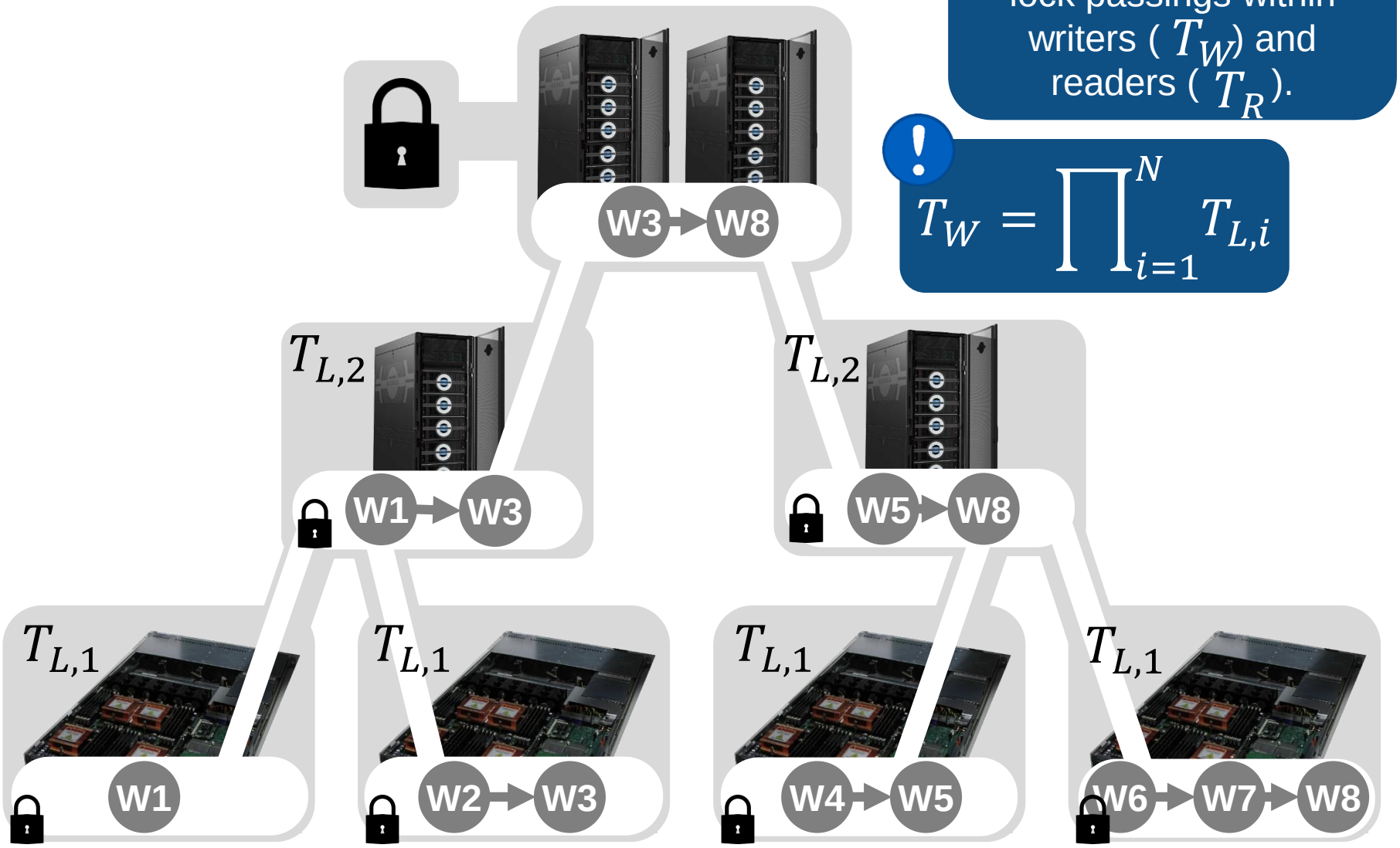
DT: The maximum number of consecutive lock passings within writers (T_W) and readers (T_R).



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).

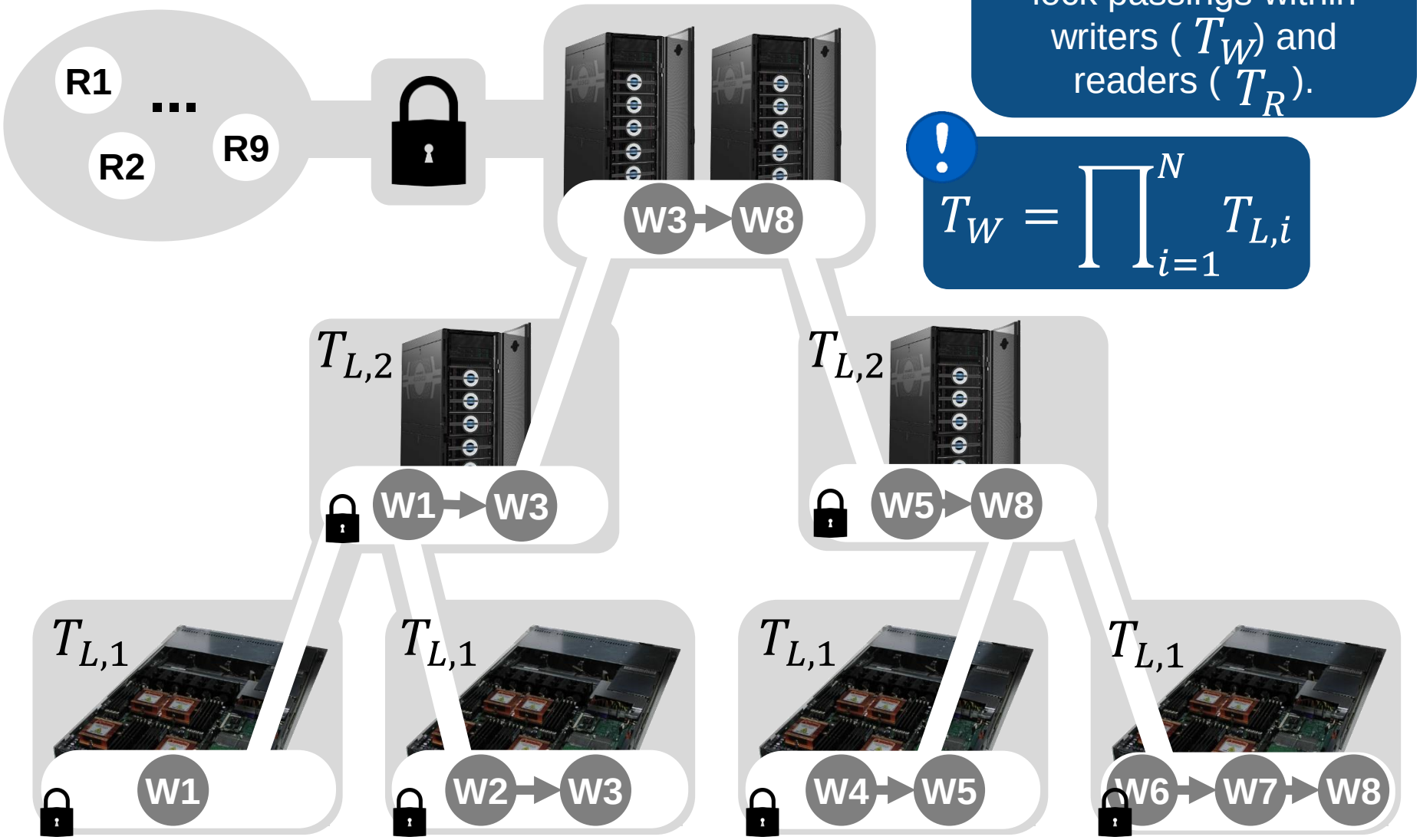
$$T_W = \prod_{i=1}^{N} T_{L,i}$$



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

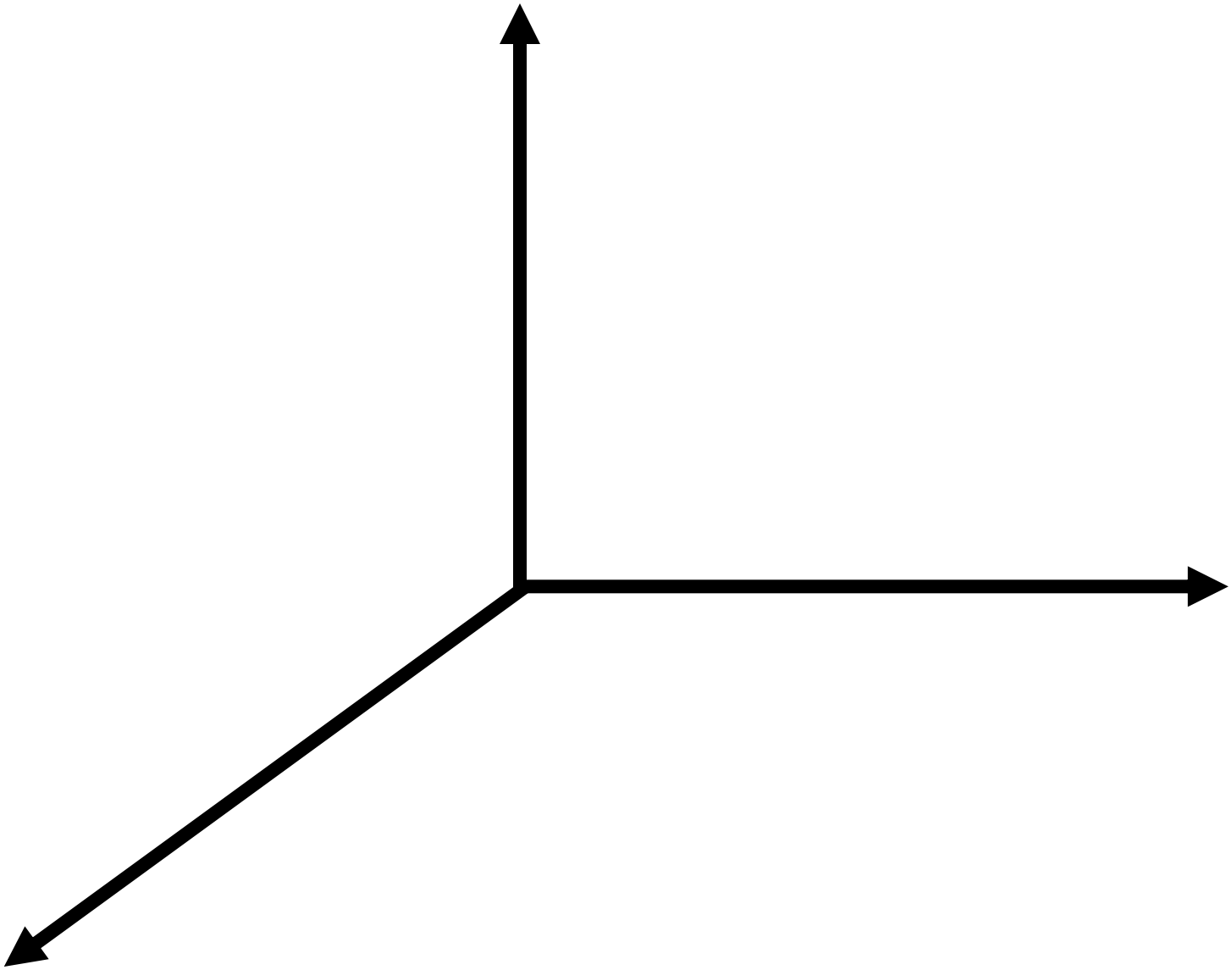
DT: The maximum number of consecutive lock passings within writers (T_W) and readers (T_R).

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

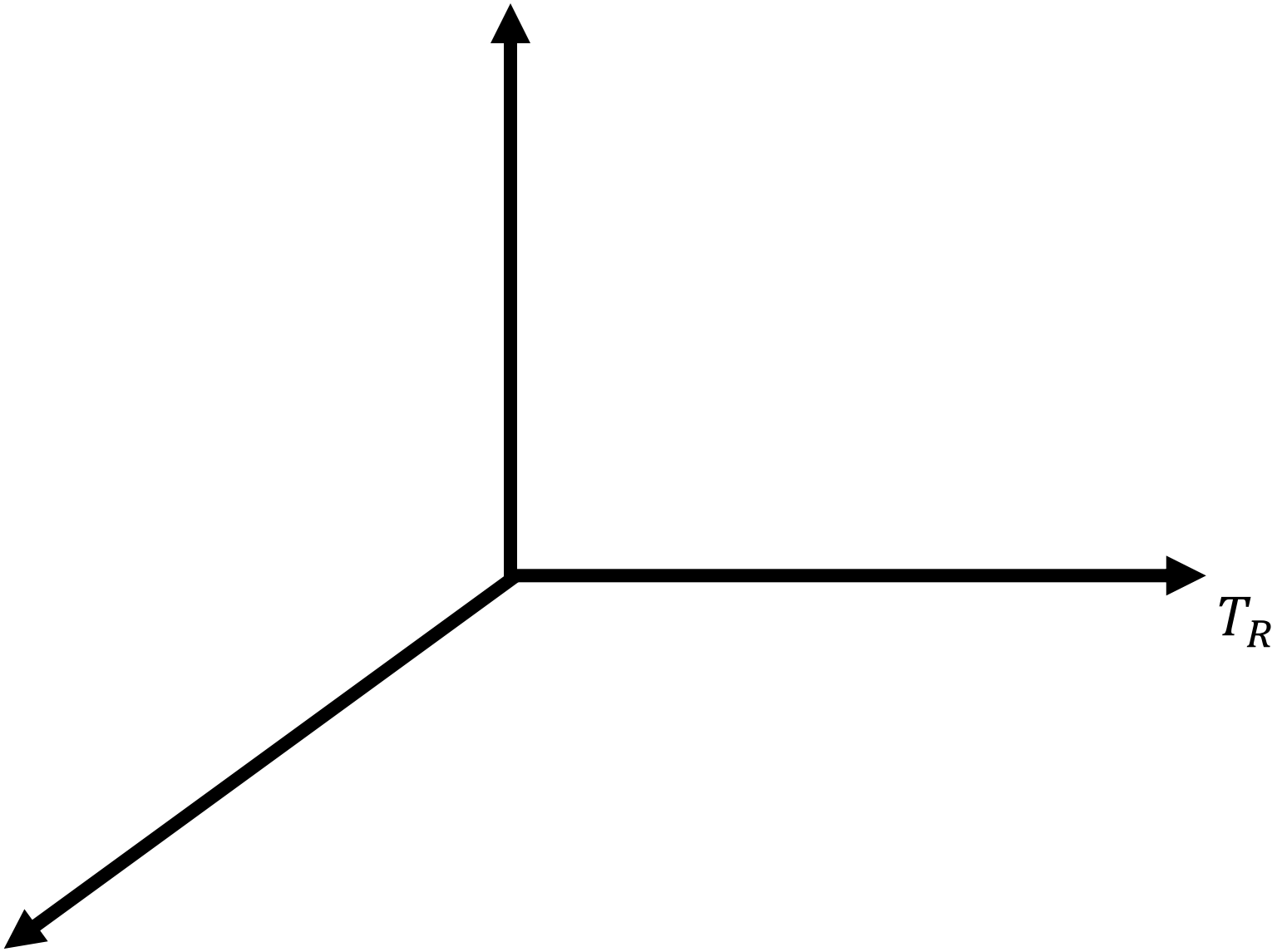


The Space of Designs

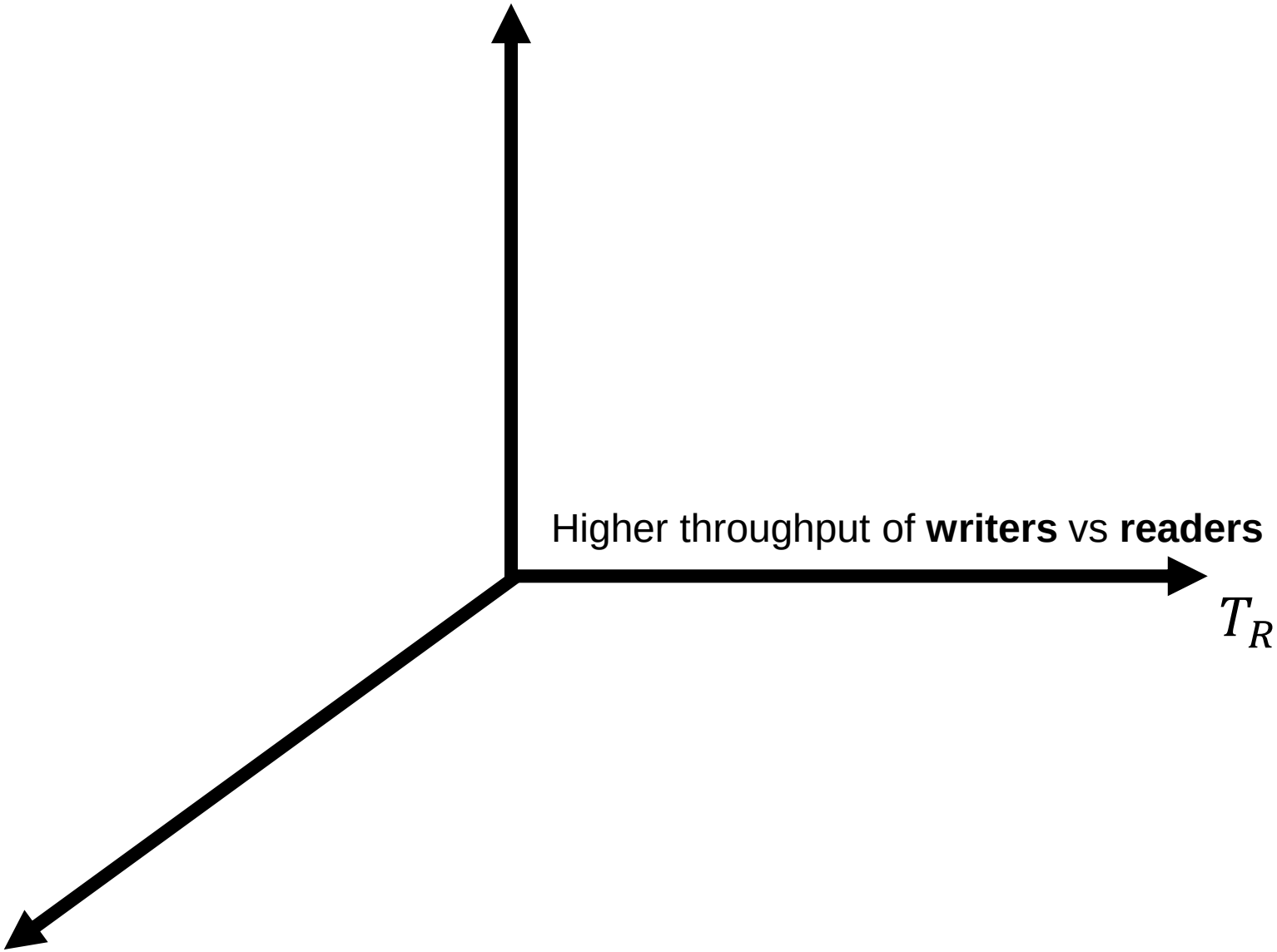
The Space of Designs



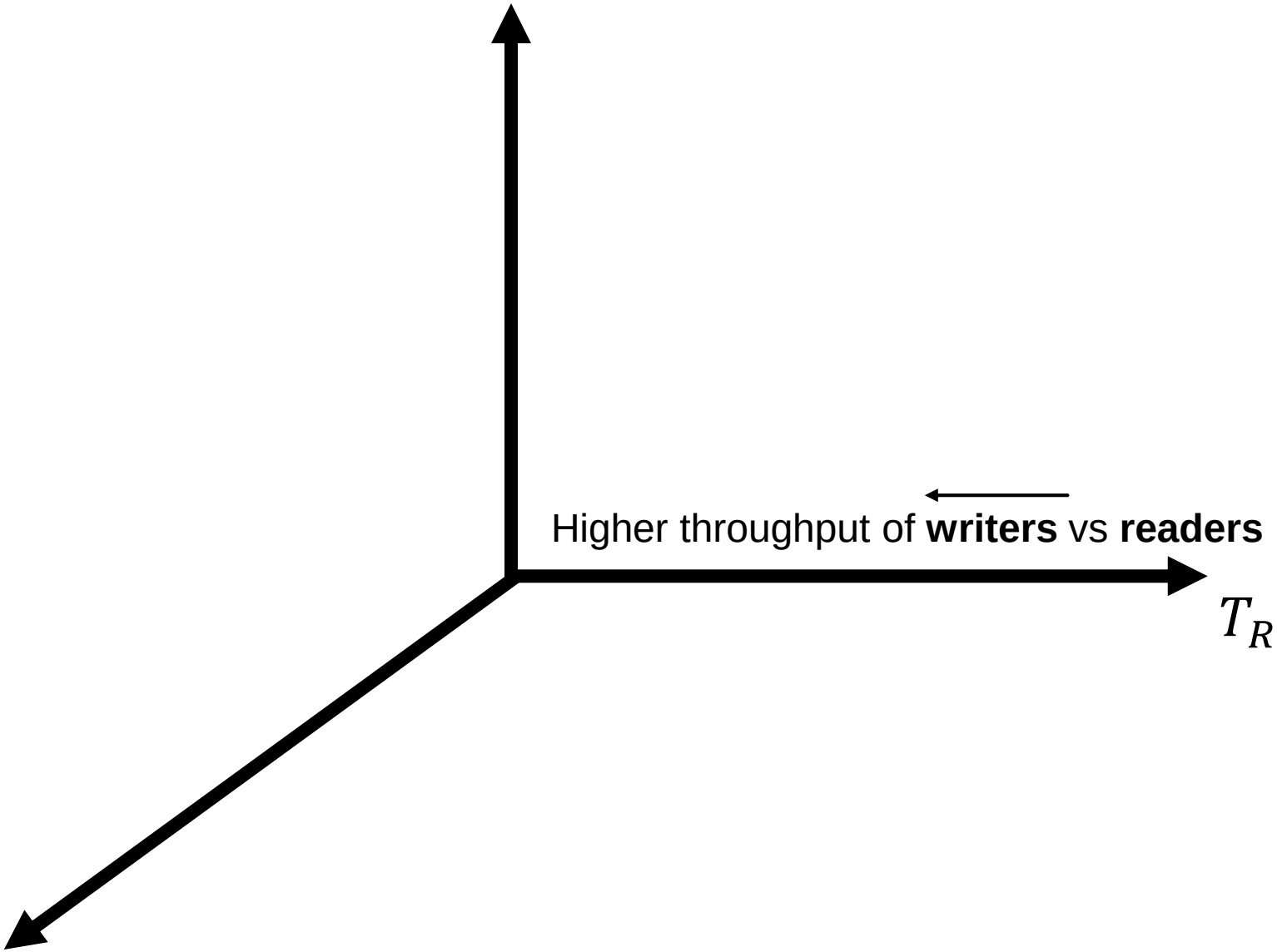
The Space of Designs



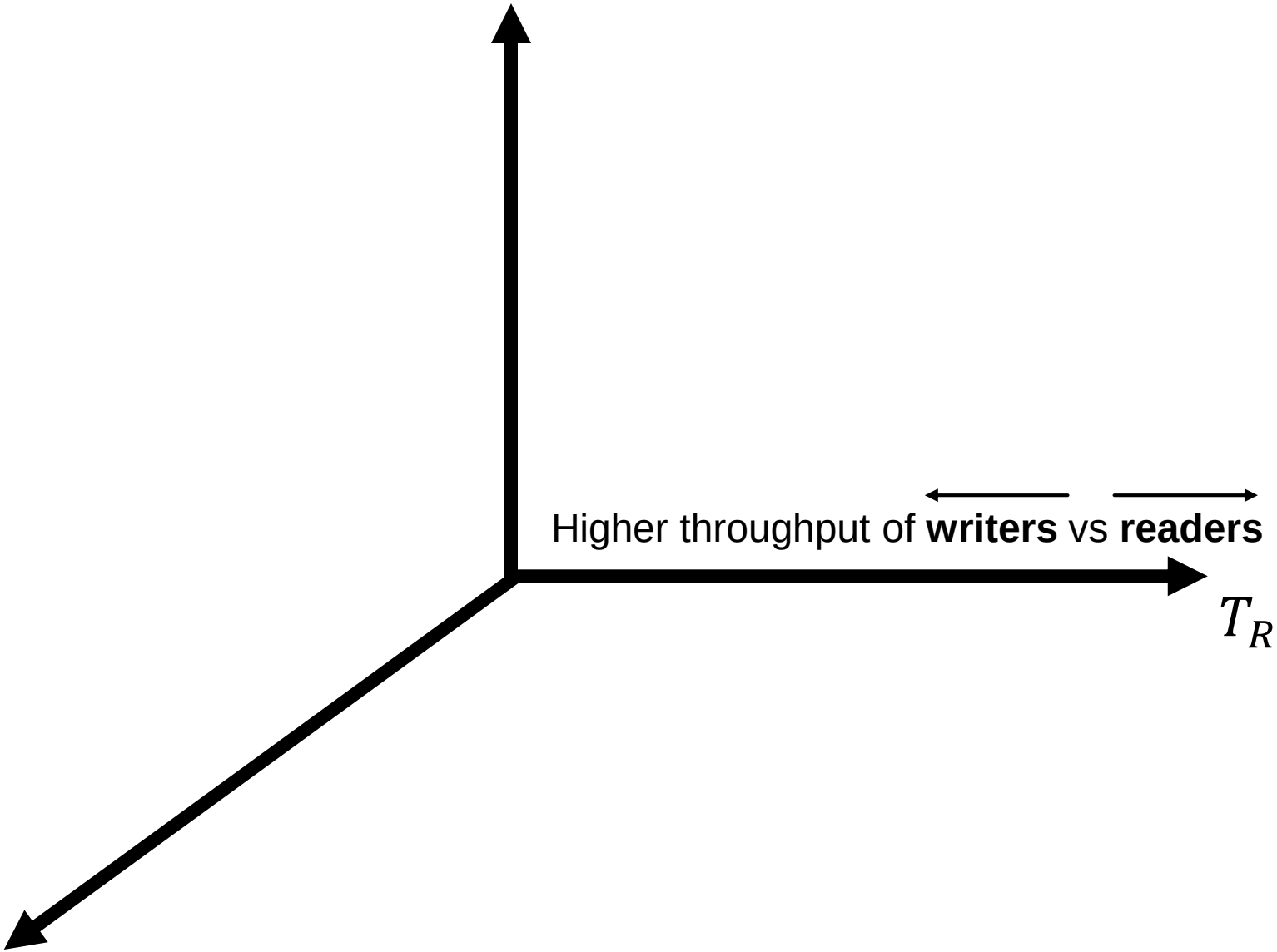
The Space of Designs



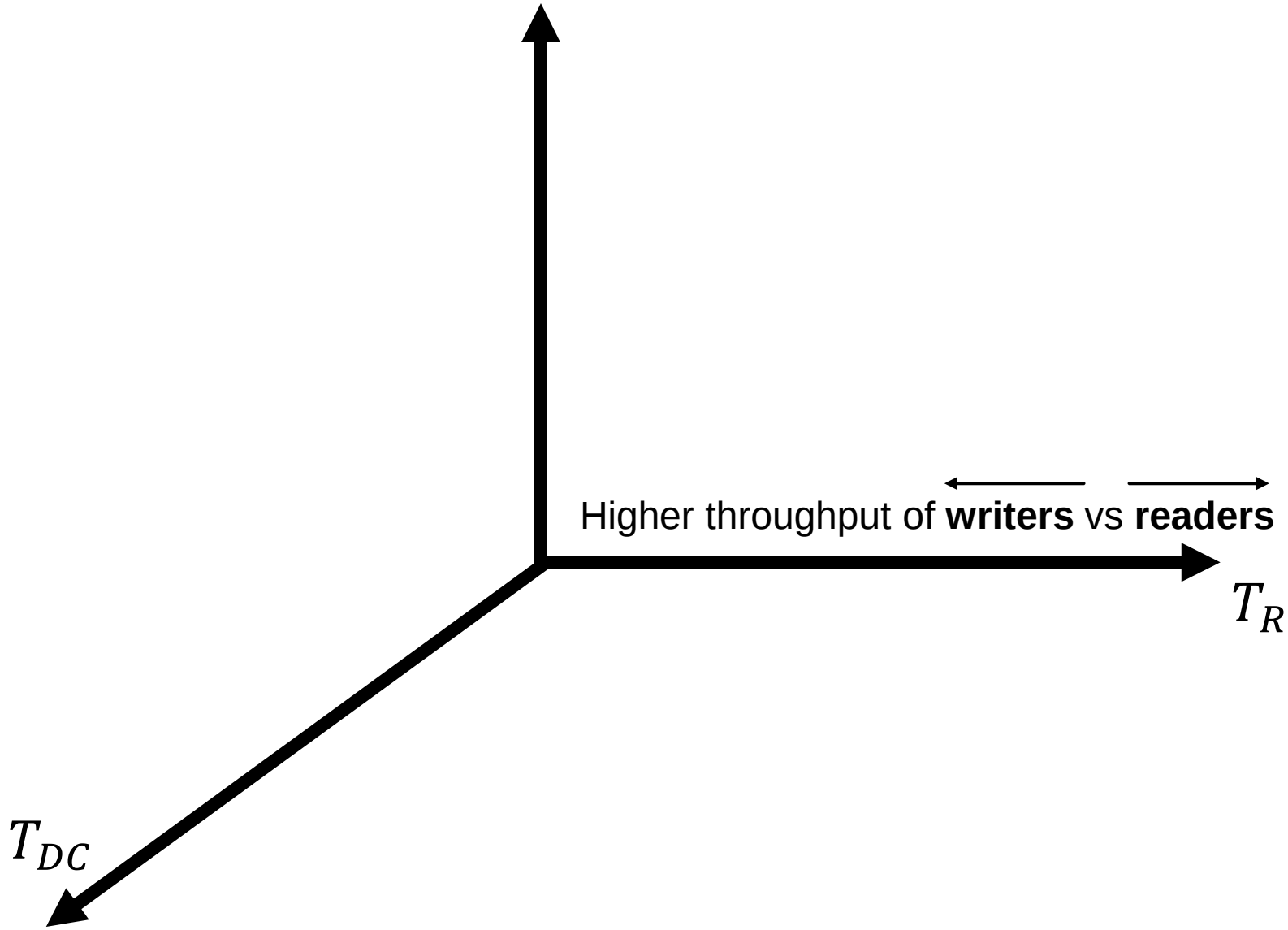
The Space of Designs



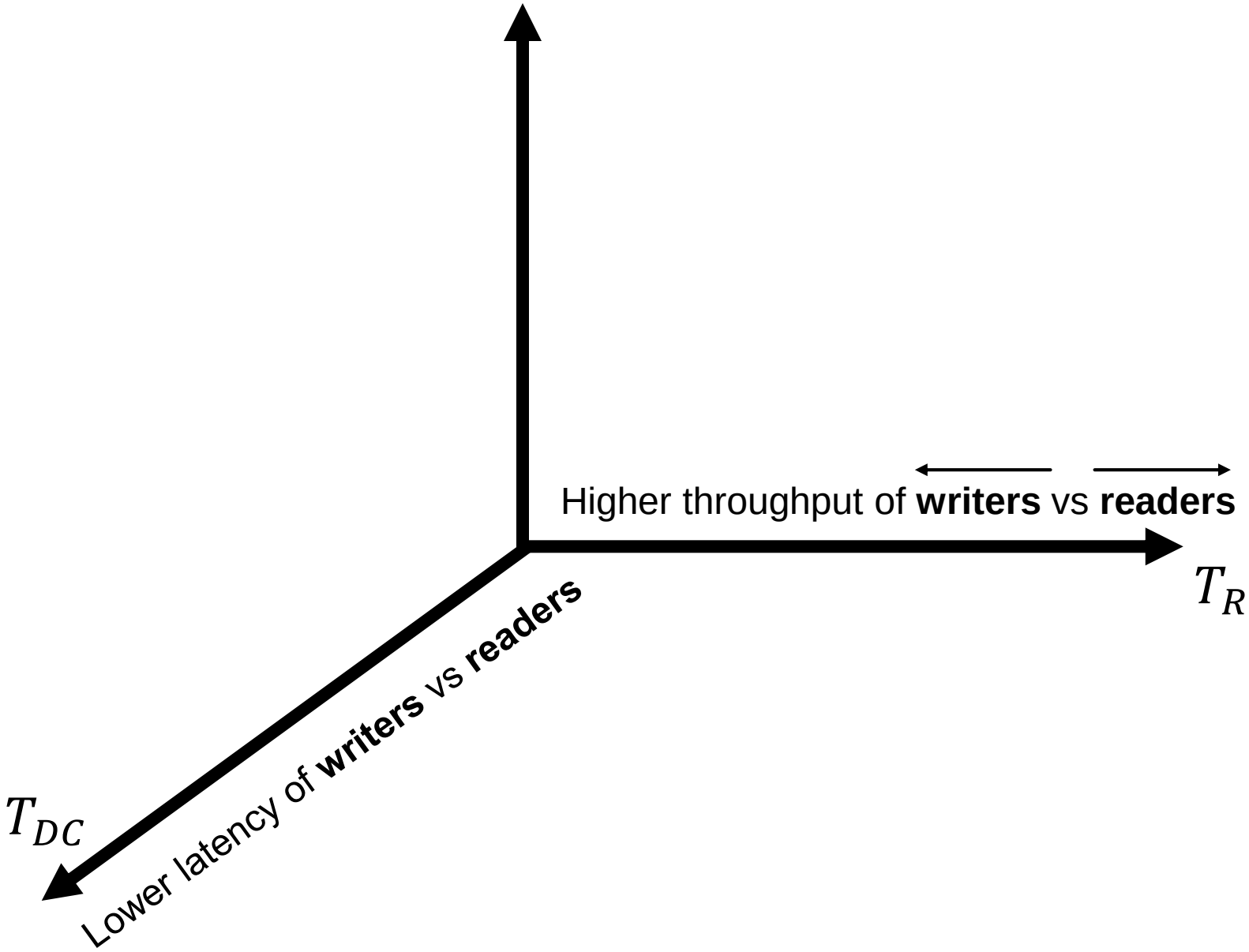
The Space of Designs



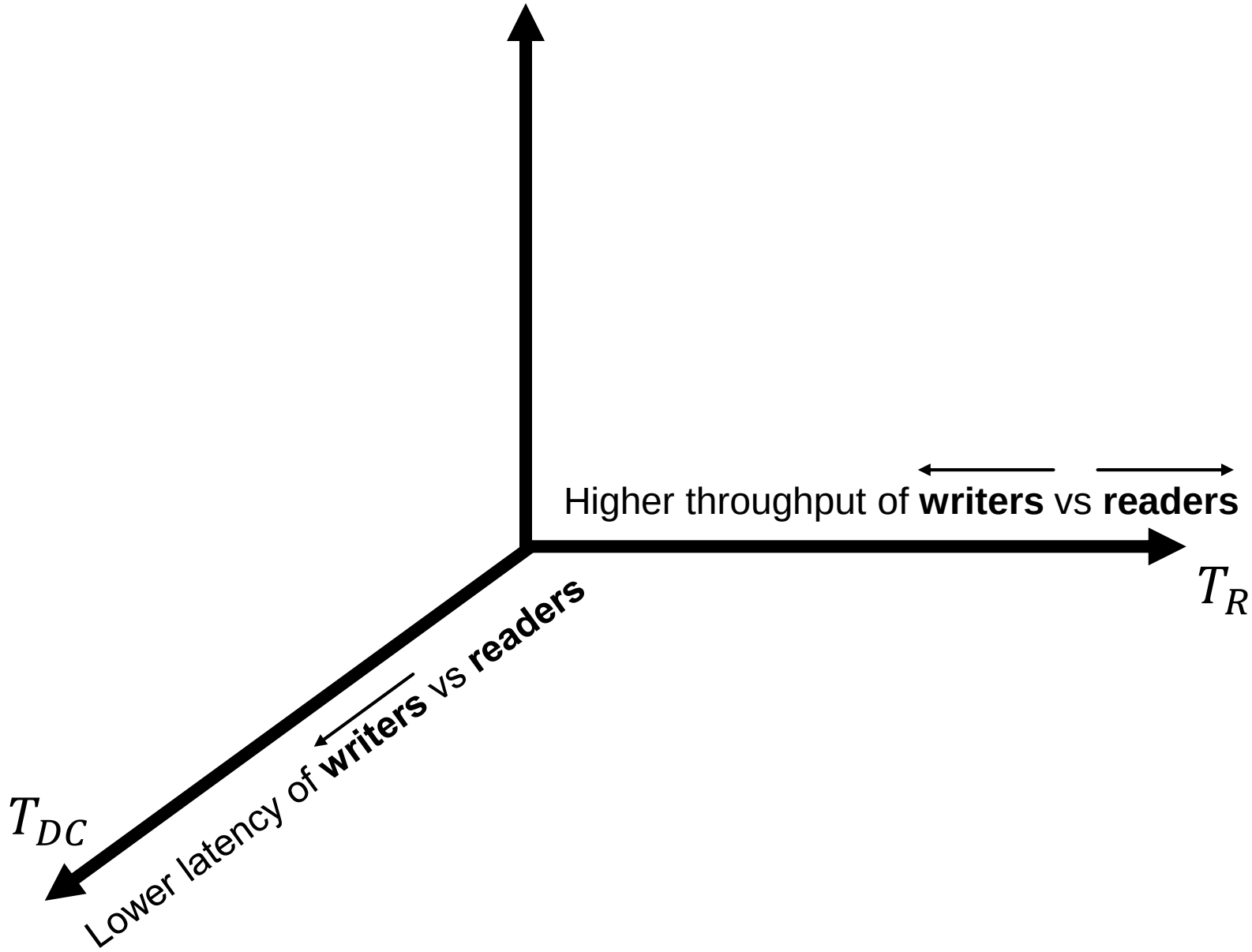
The Space of Designs



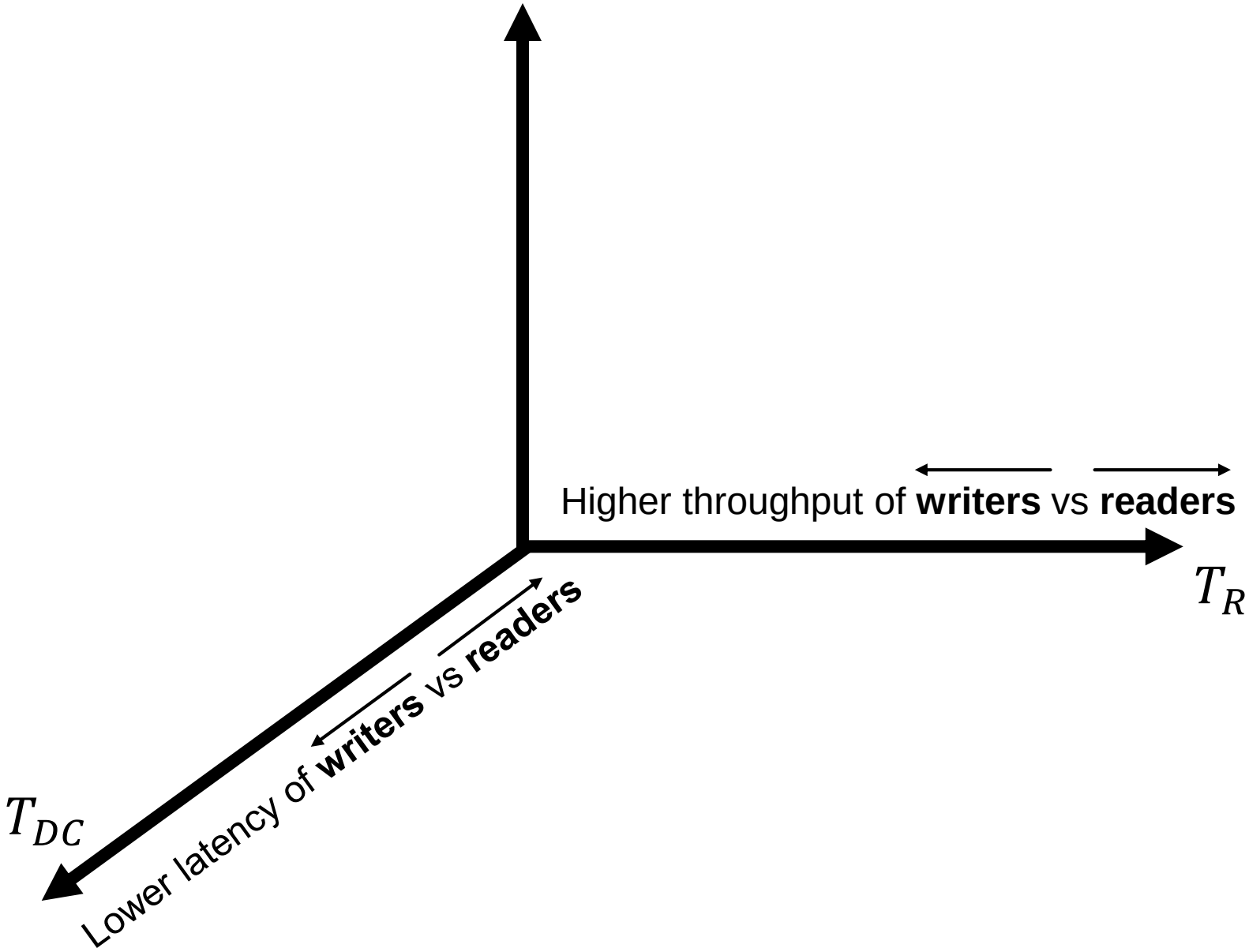
The Space of Designs



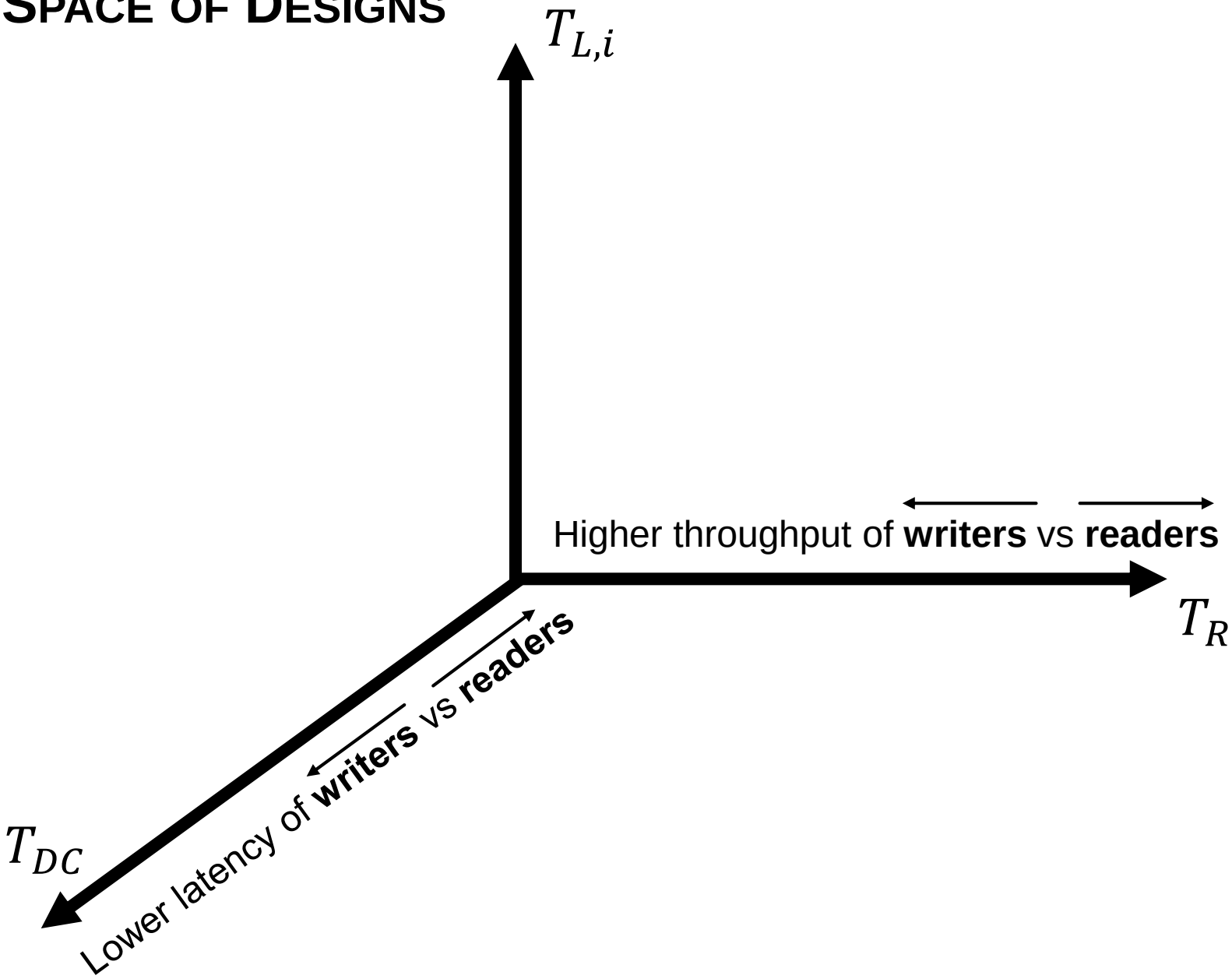
The Space of Designs



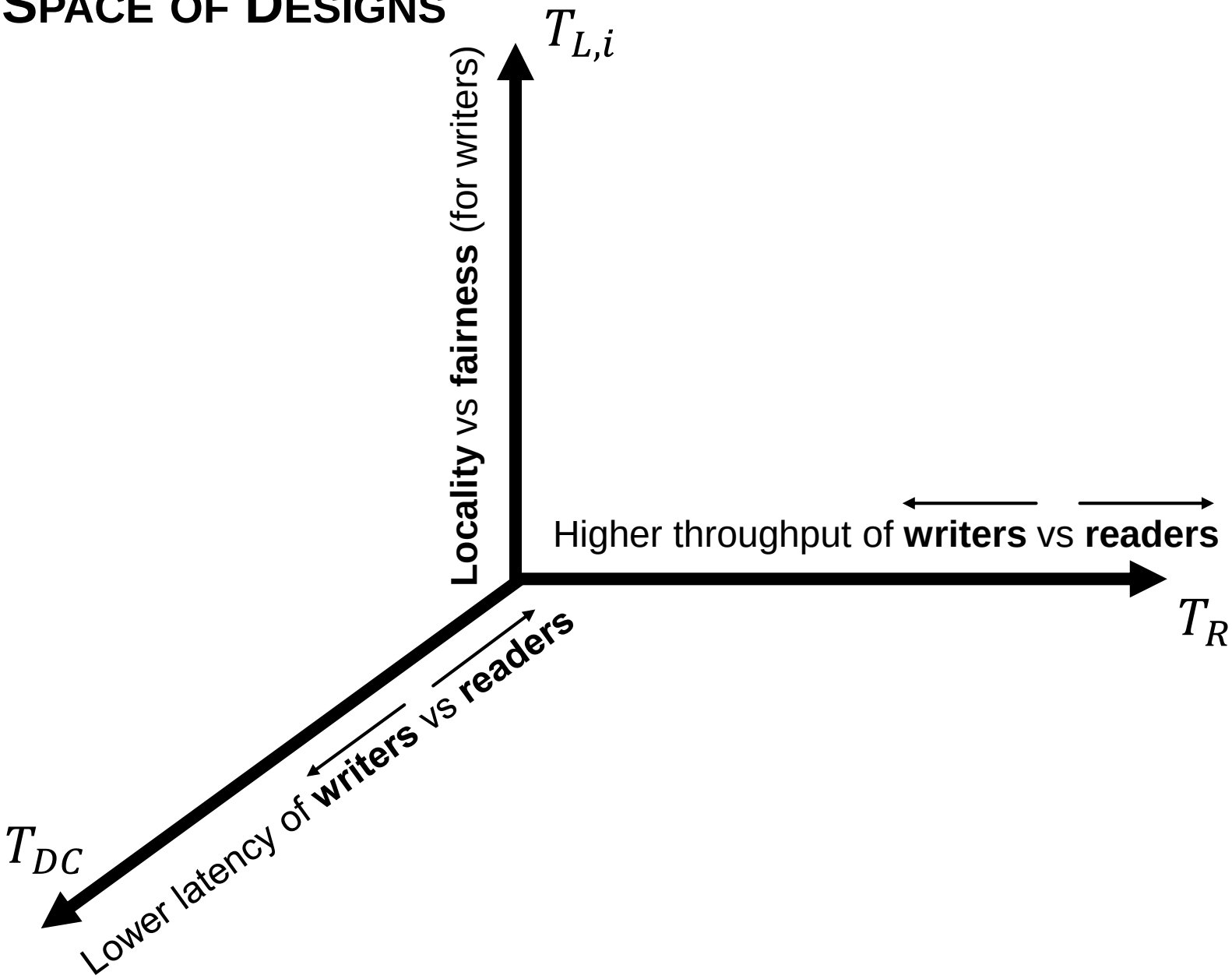
The Space of Designs



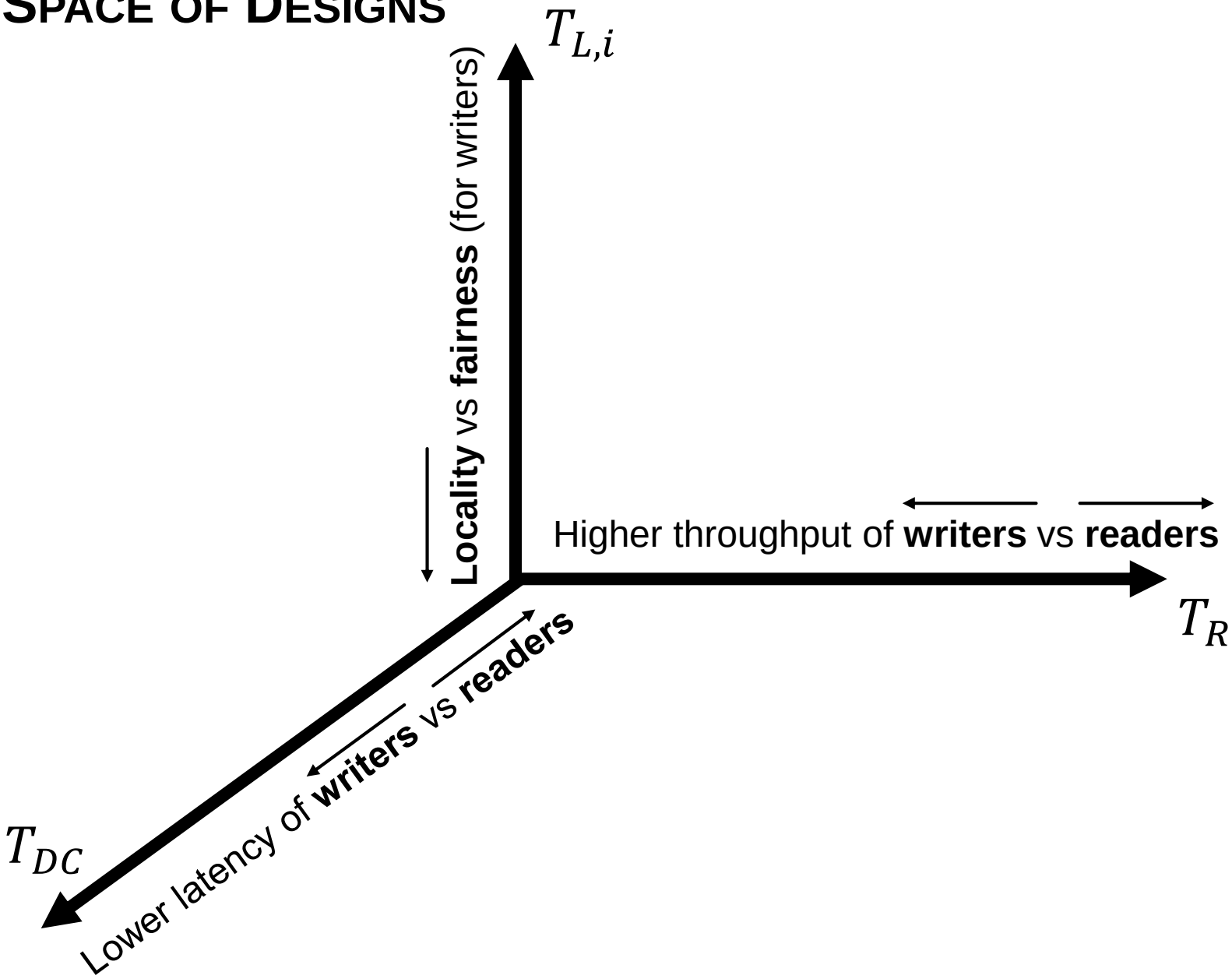
The Space of Designs



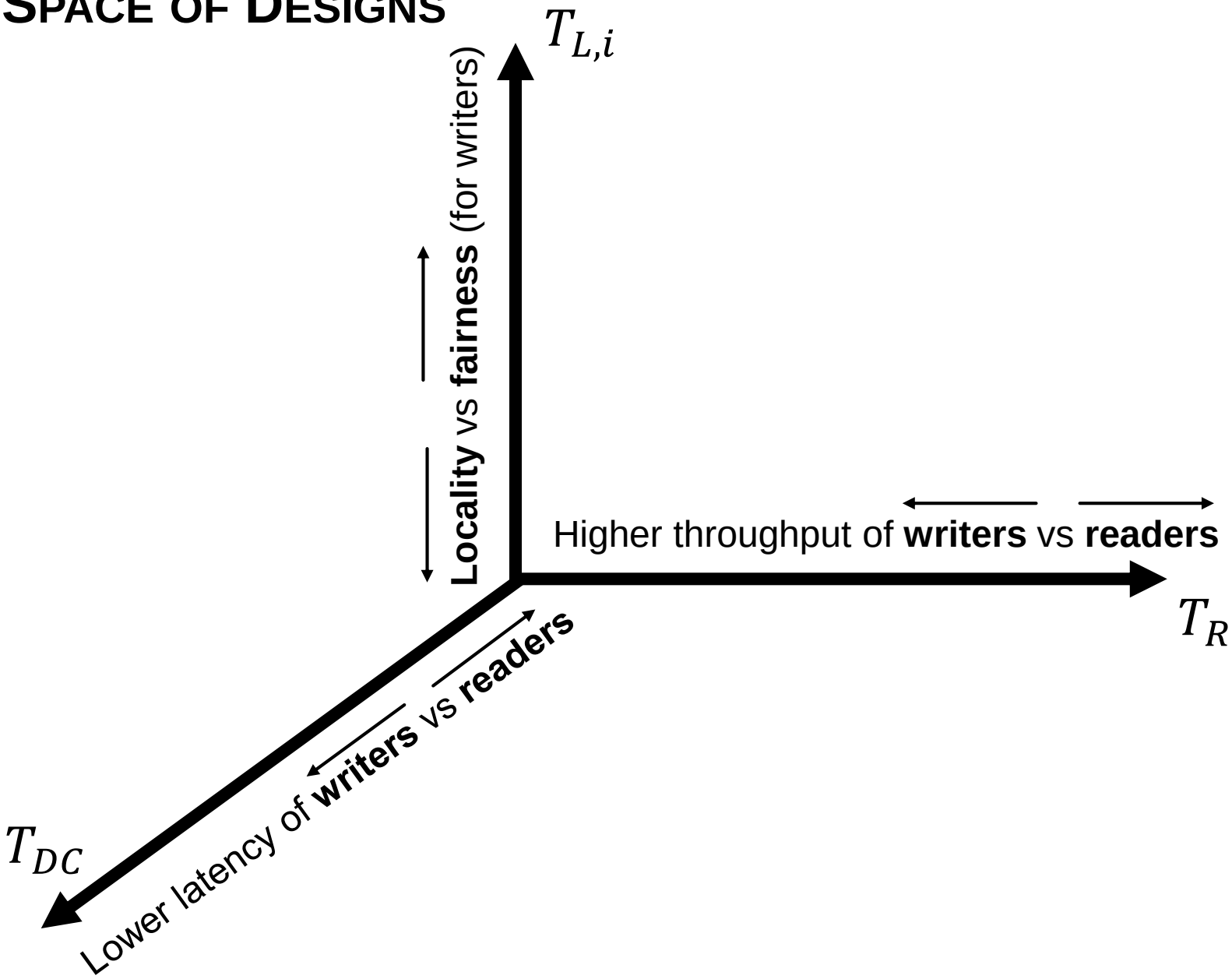
The Space of Designs



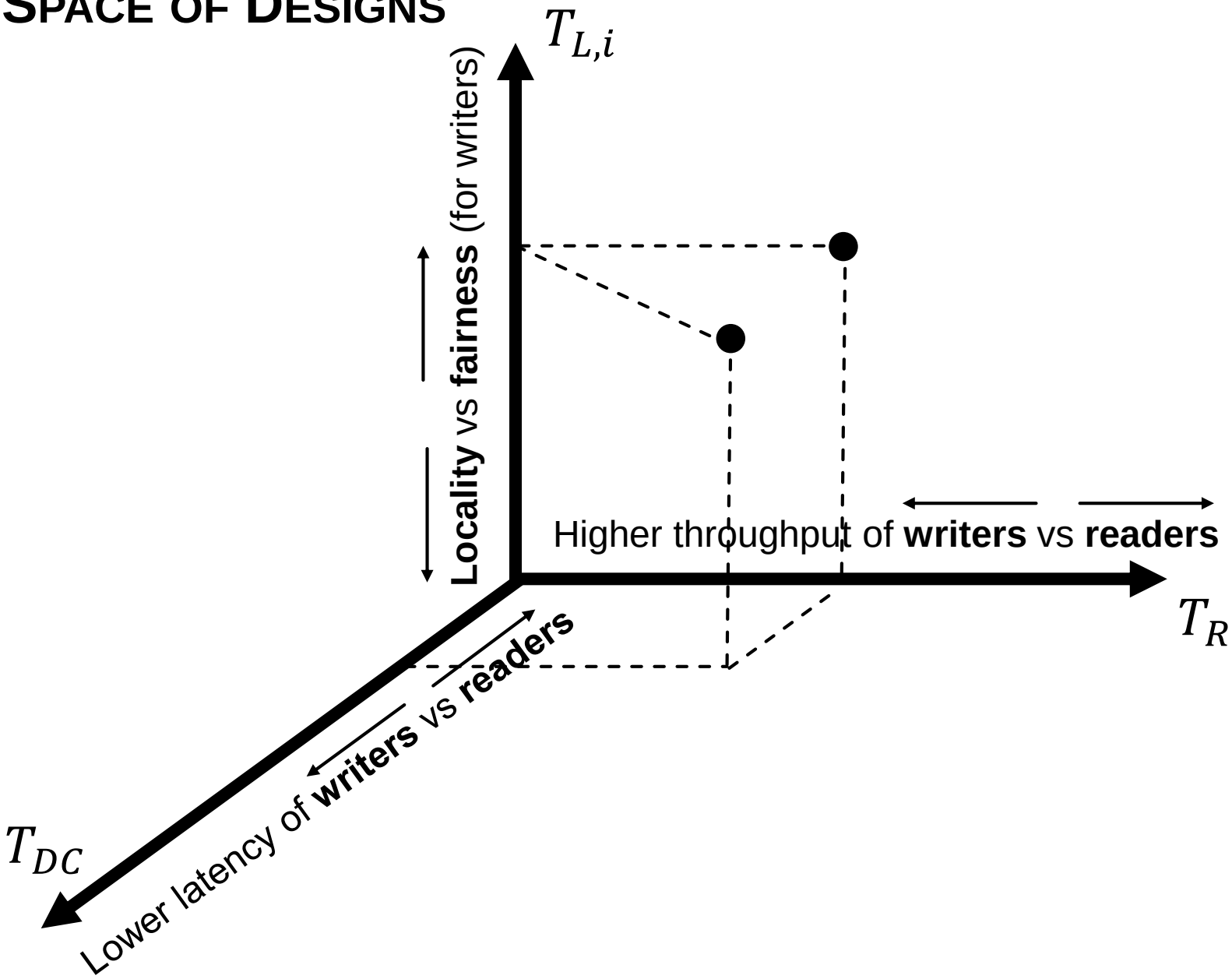
The Space of Designs



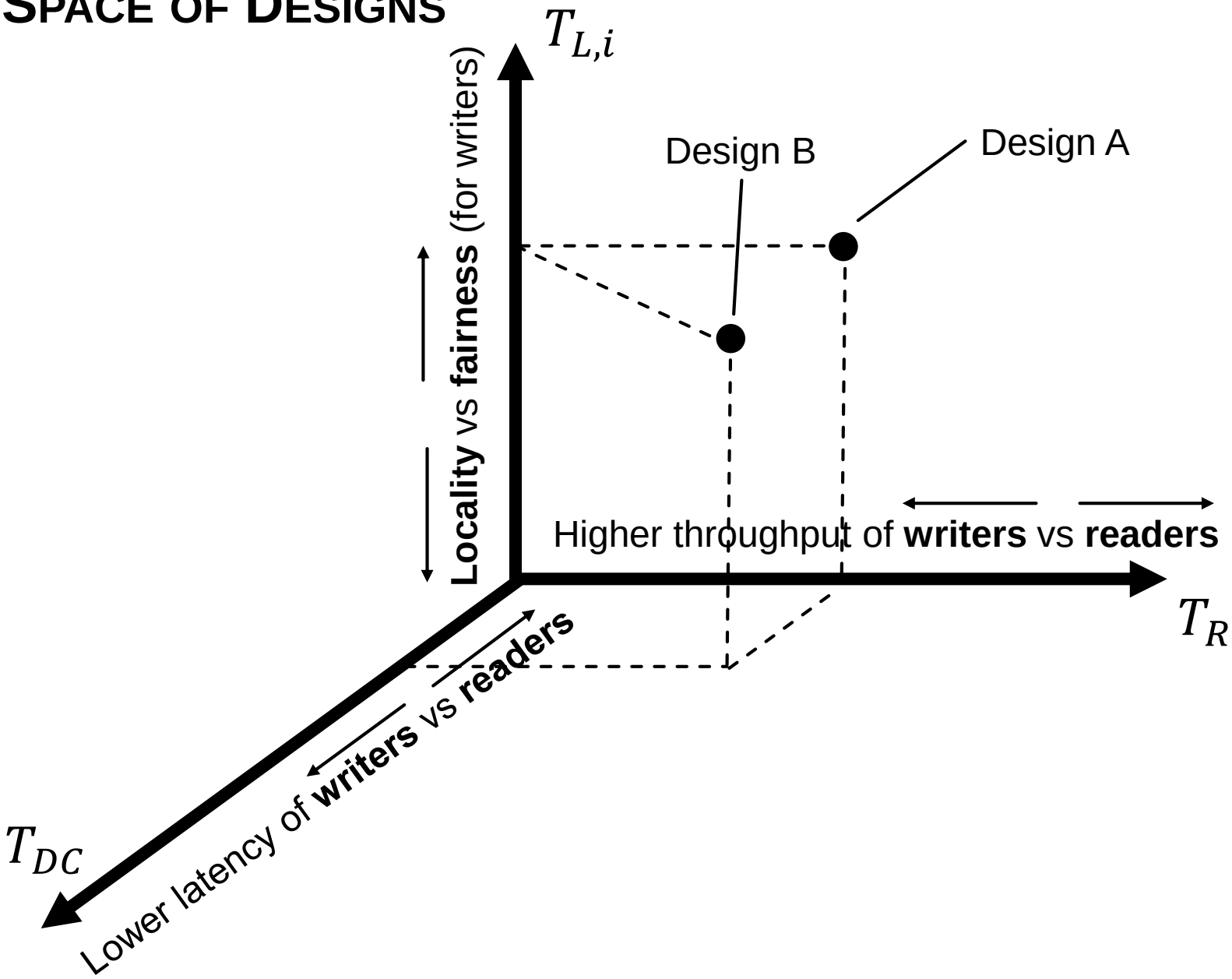
The Space of Designs



The Space of Designs

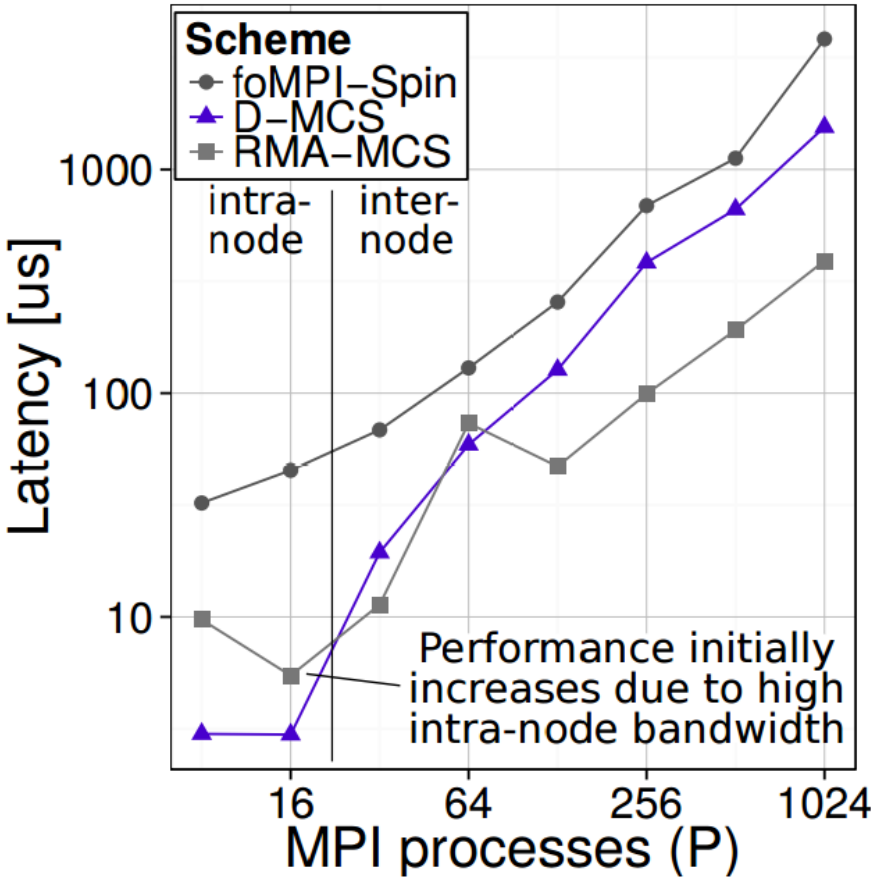


THE SPACE OF DESIGNS

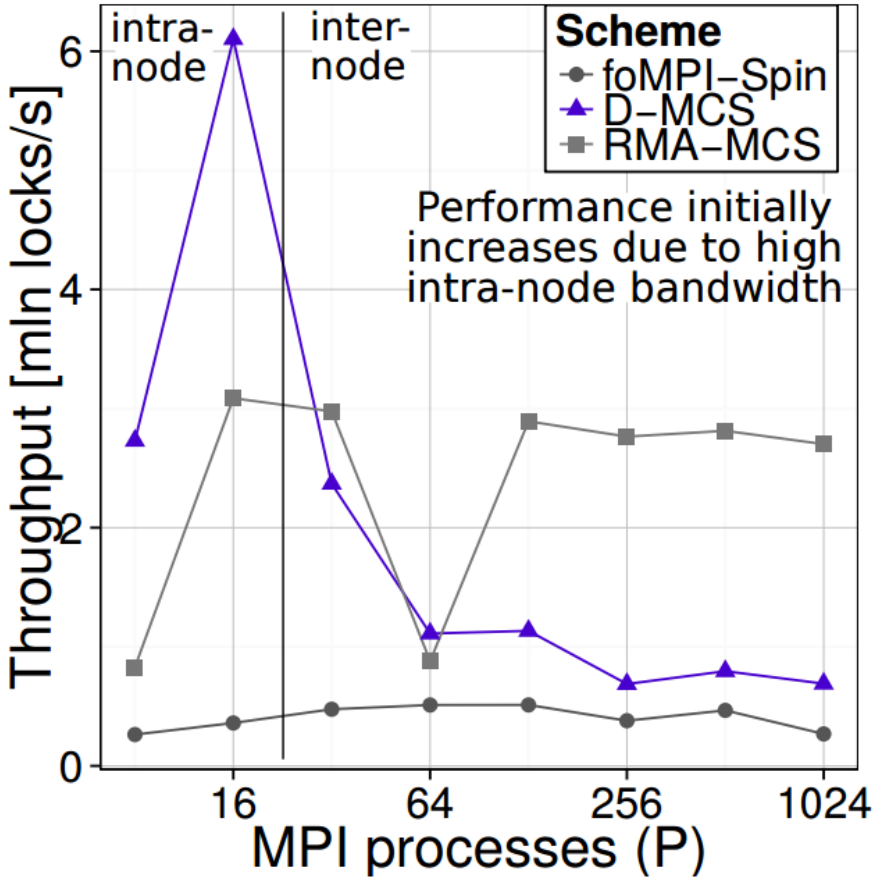


EVALUATION
D-MCS VS OTHERS

Latency (LB)



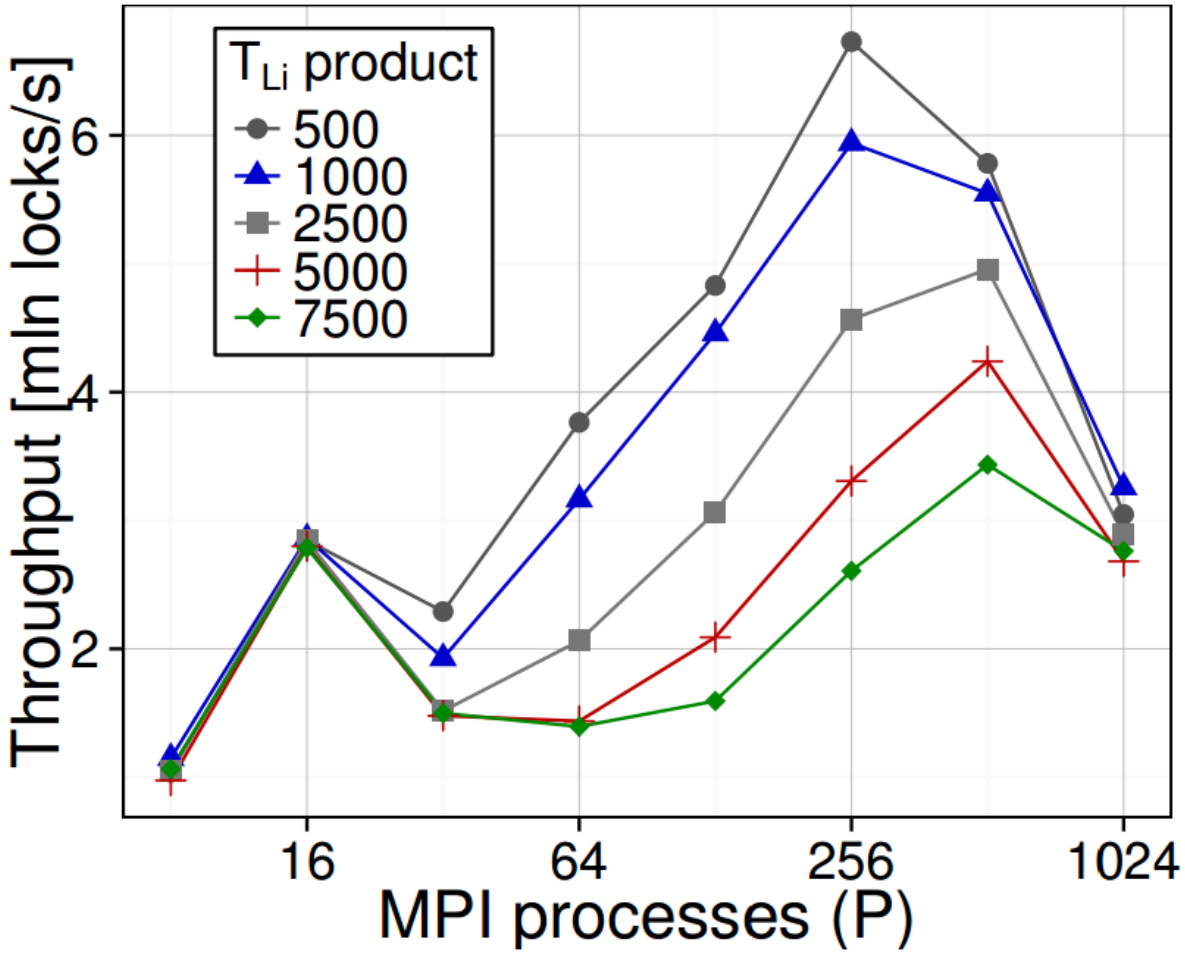
Throughput (ECSB)



EVALUATION
WRITER THRESHOLD ANALYSIS

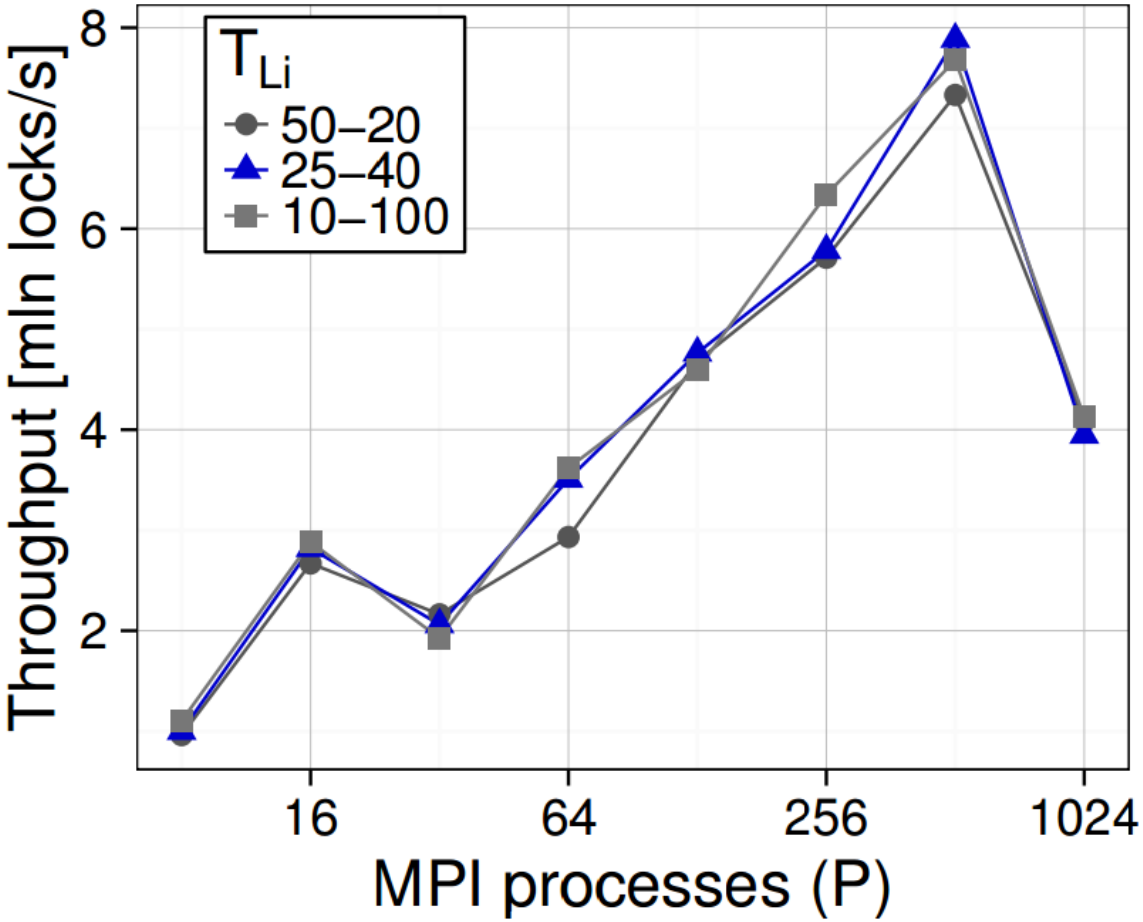
Throughput, 25% of writers

Single-operation benchmark



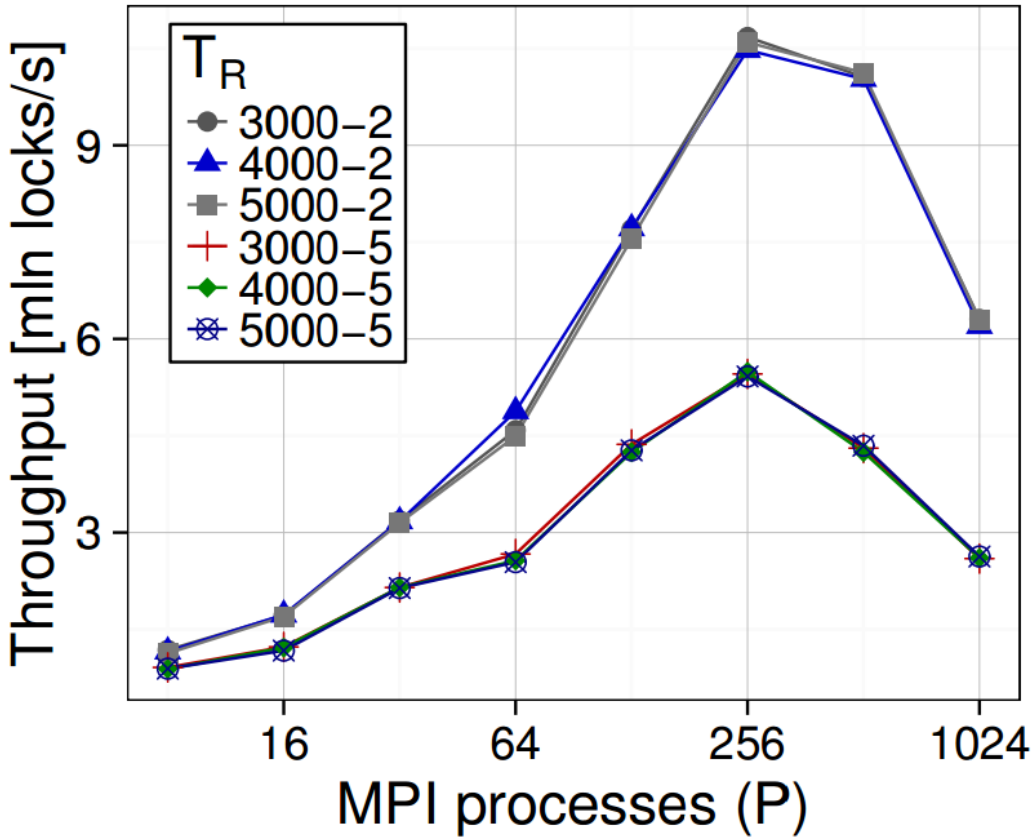
EVALUATION
FAIRNESS VS THROUGHPUT ANALYSIS

Throughput, 25% of writers,
Single-operation benchmark



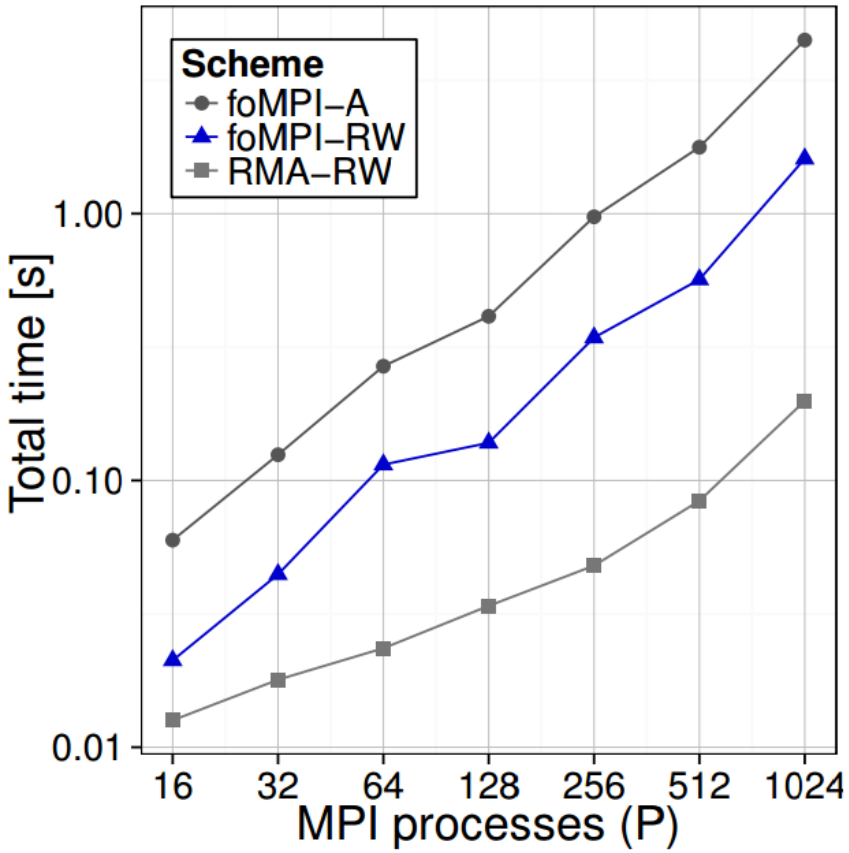
EVALUATION
READER THRESHOLD ANALYSIS

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

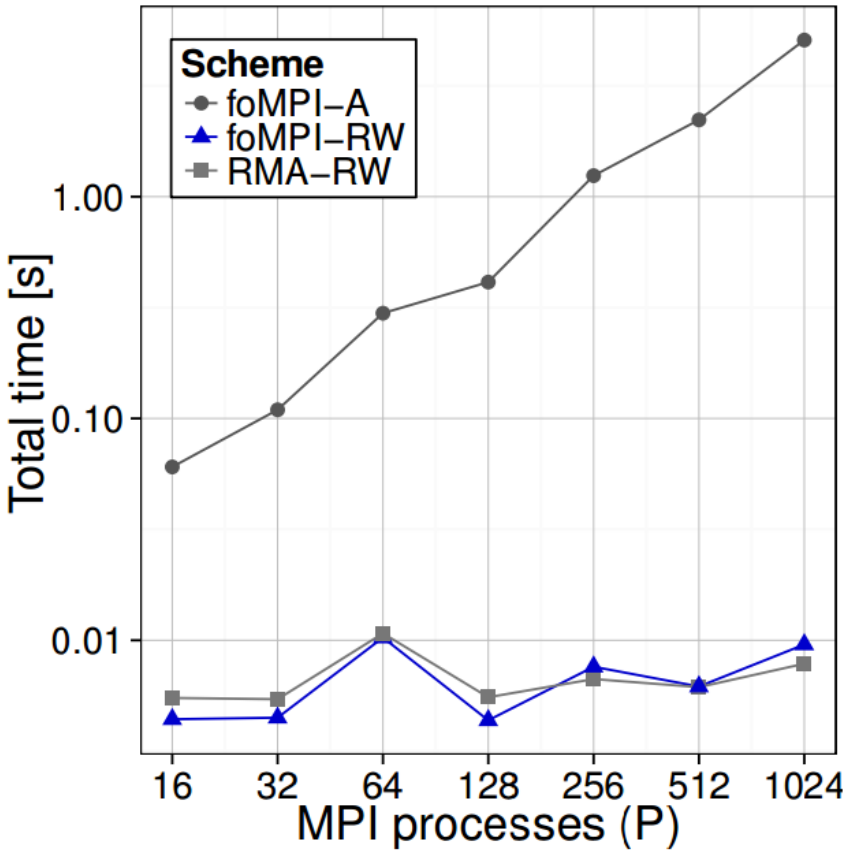


EVALUATION
DISTRIBUTED HASHTABLE

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.

FEASIBILITY ANALYSIS

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

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

RMA-RW
Required Operations

Process p

Memory

Process q

Memory

3

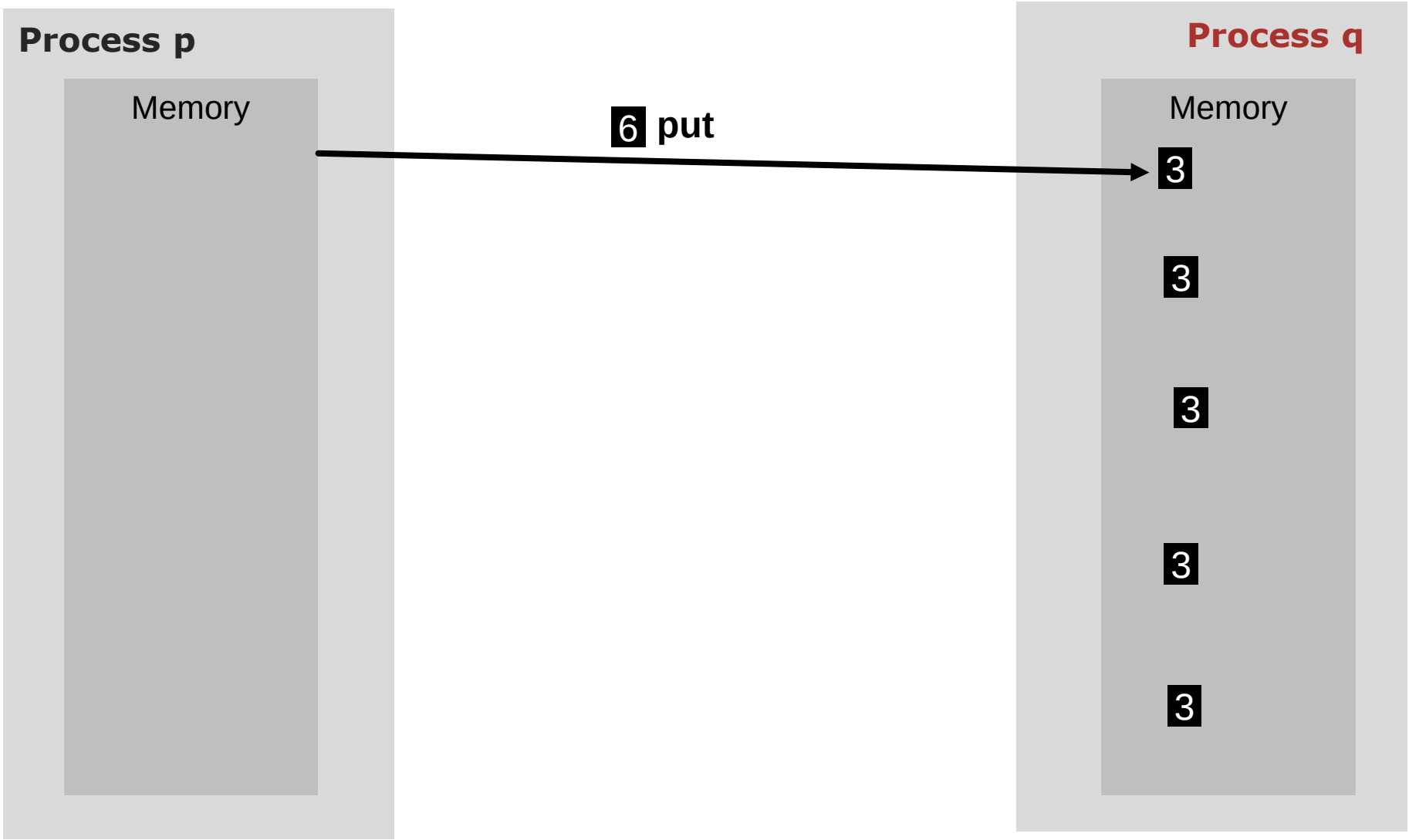
3

3

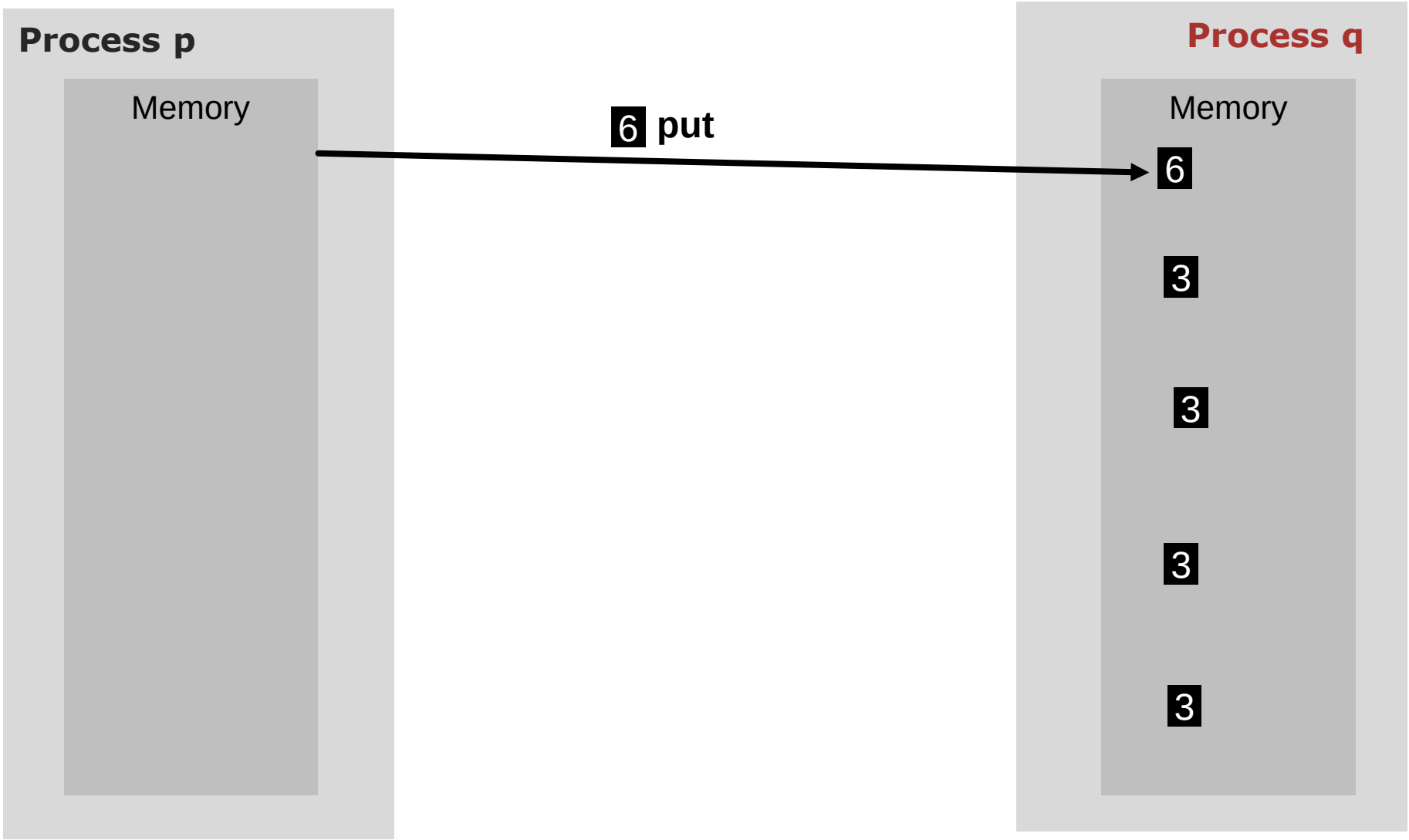
3

3

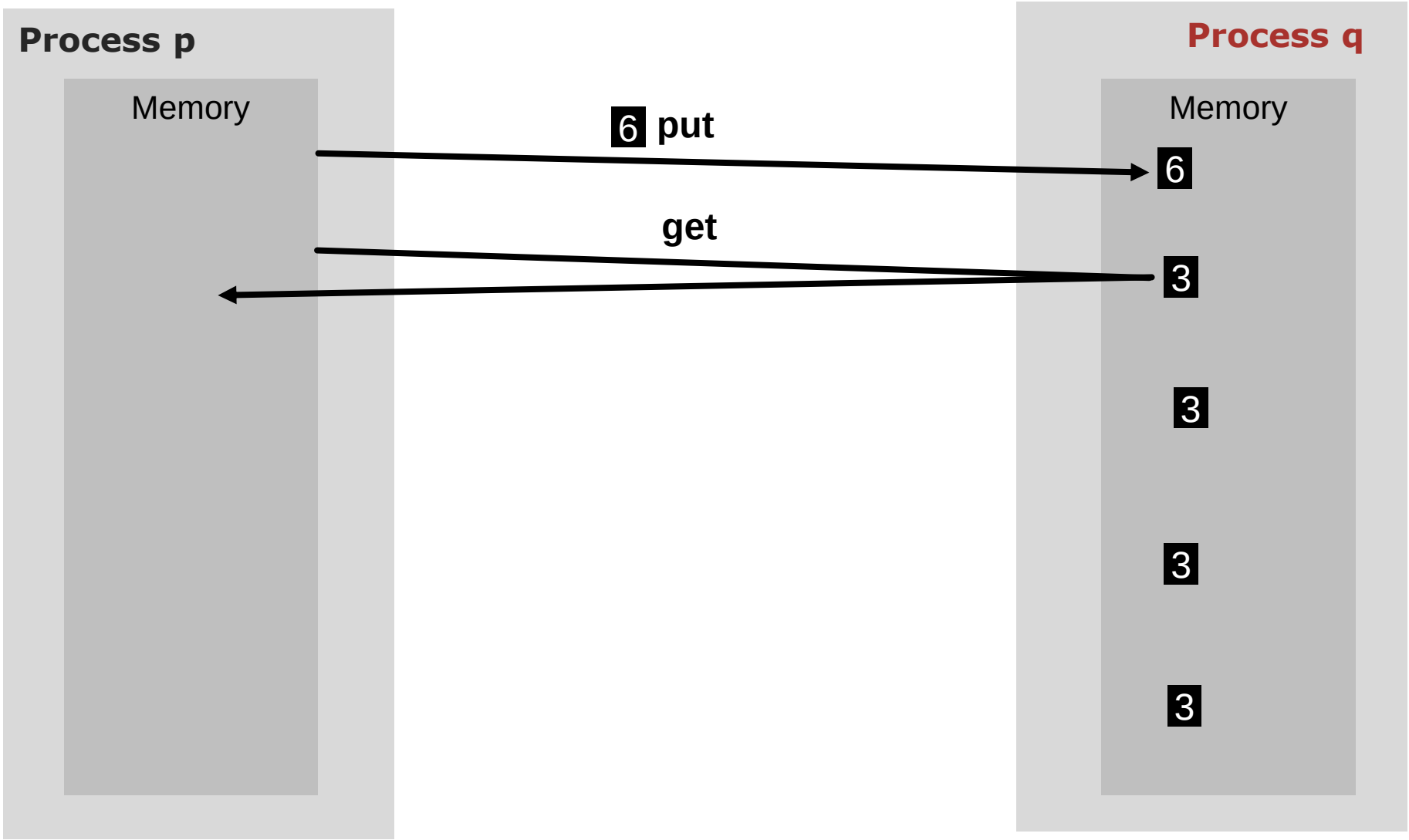
RMA-RW
Required Operations



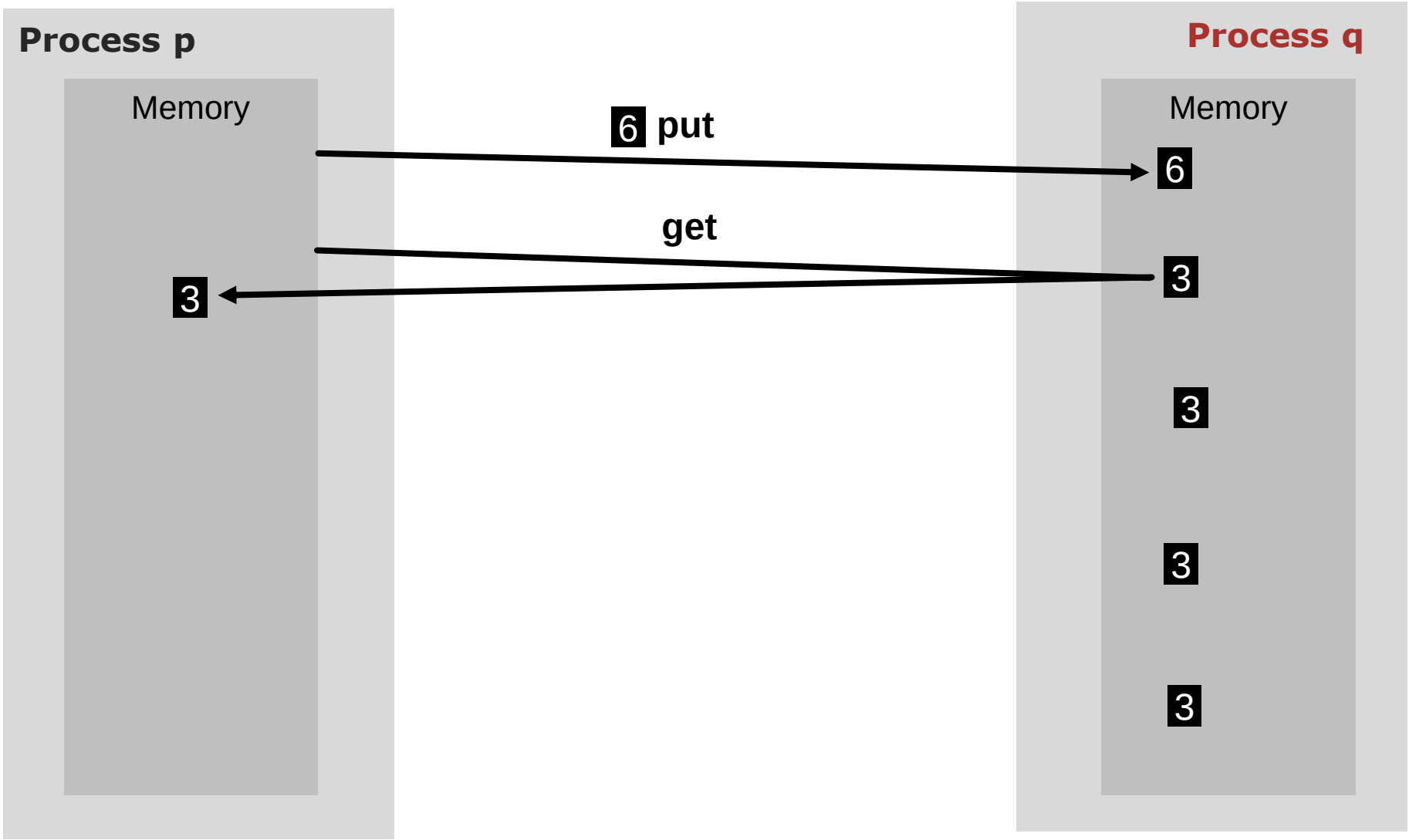
RMA-RW
Required Operations



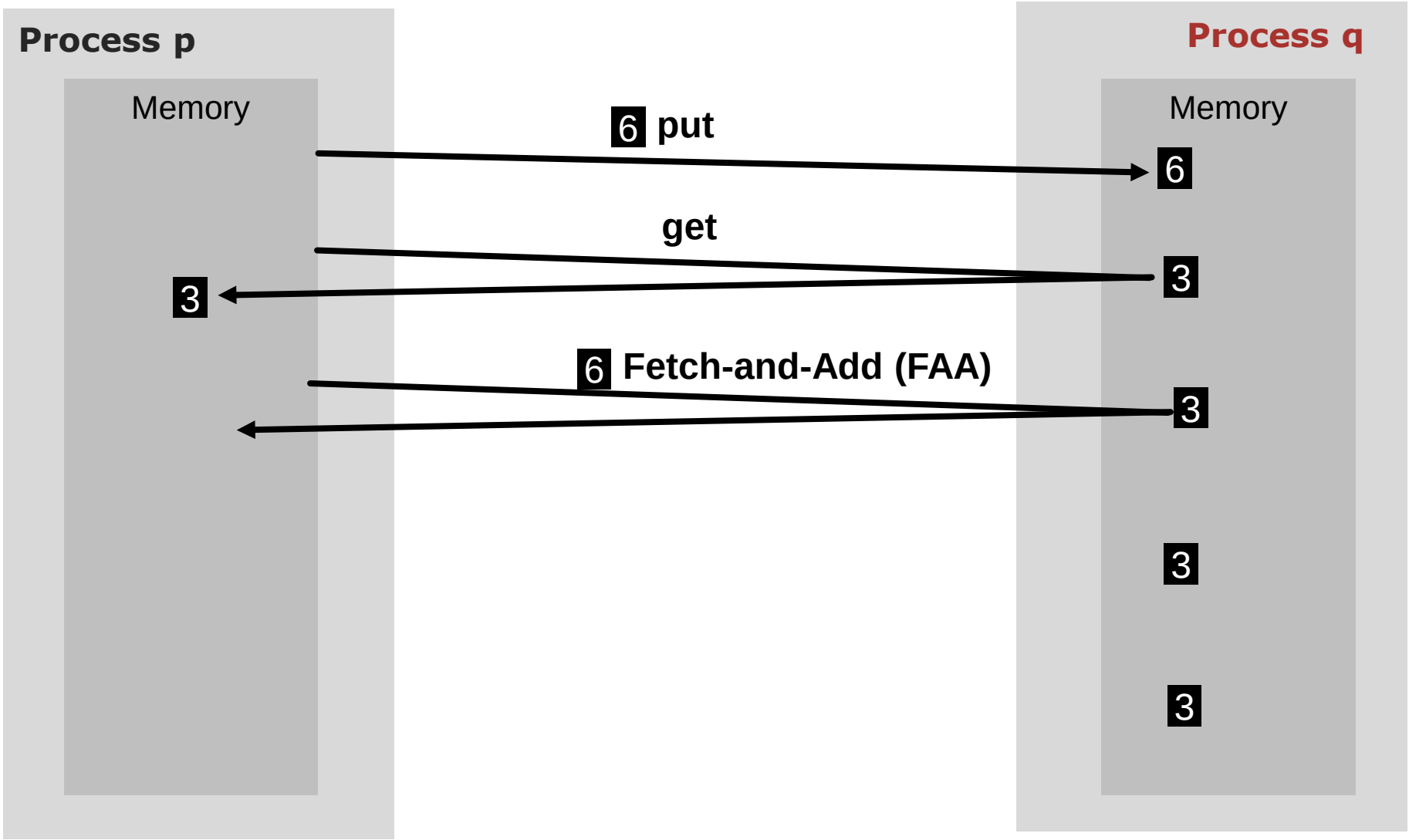
RMA-RW
Required Operations



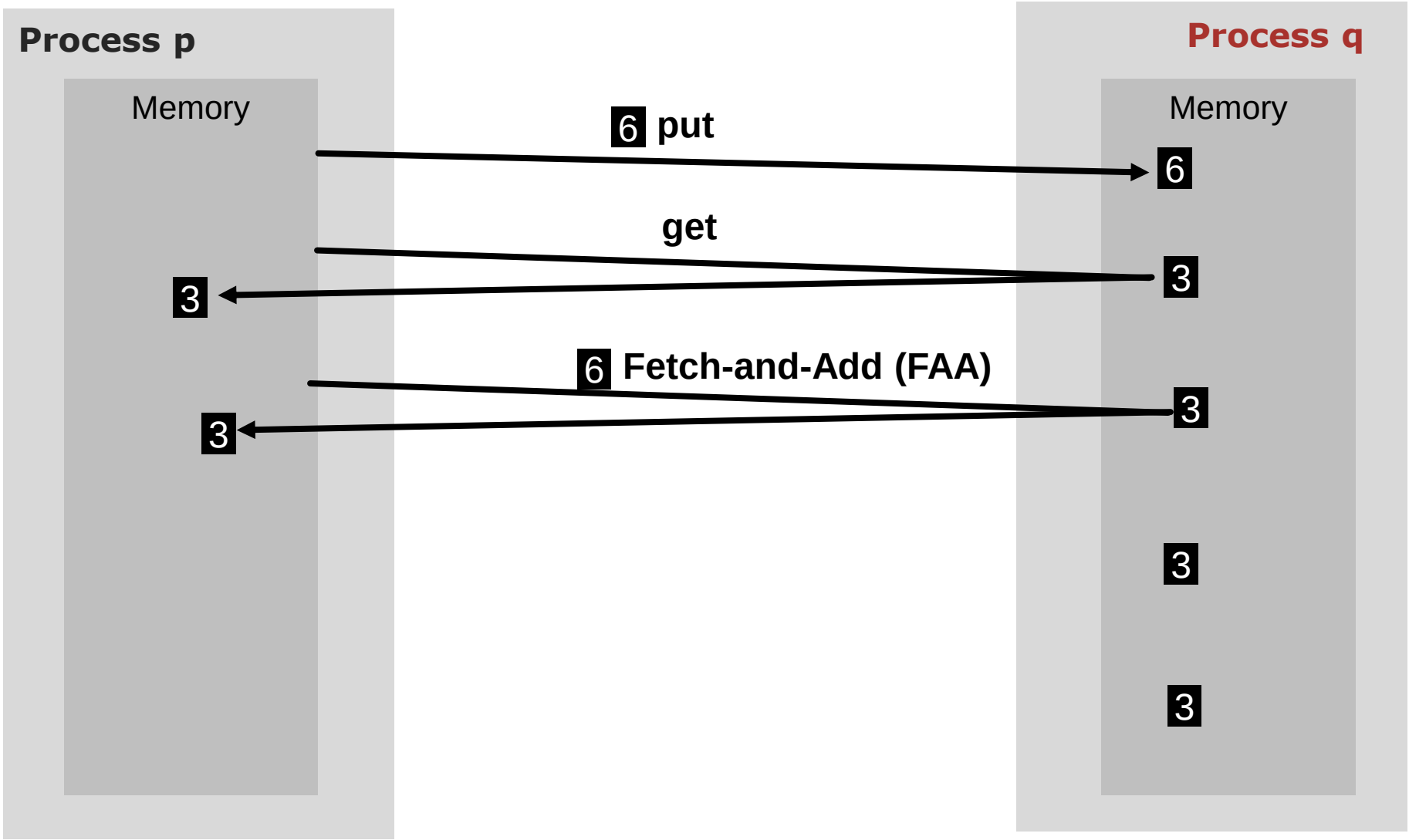
RMA-RW
Required Operations



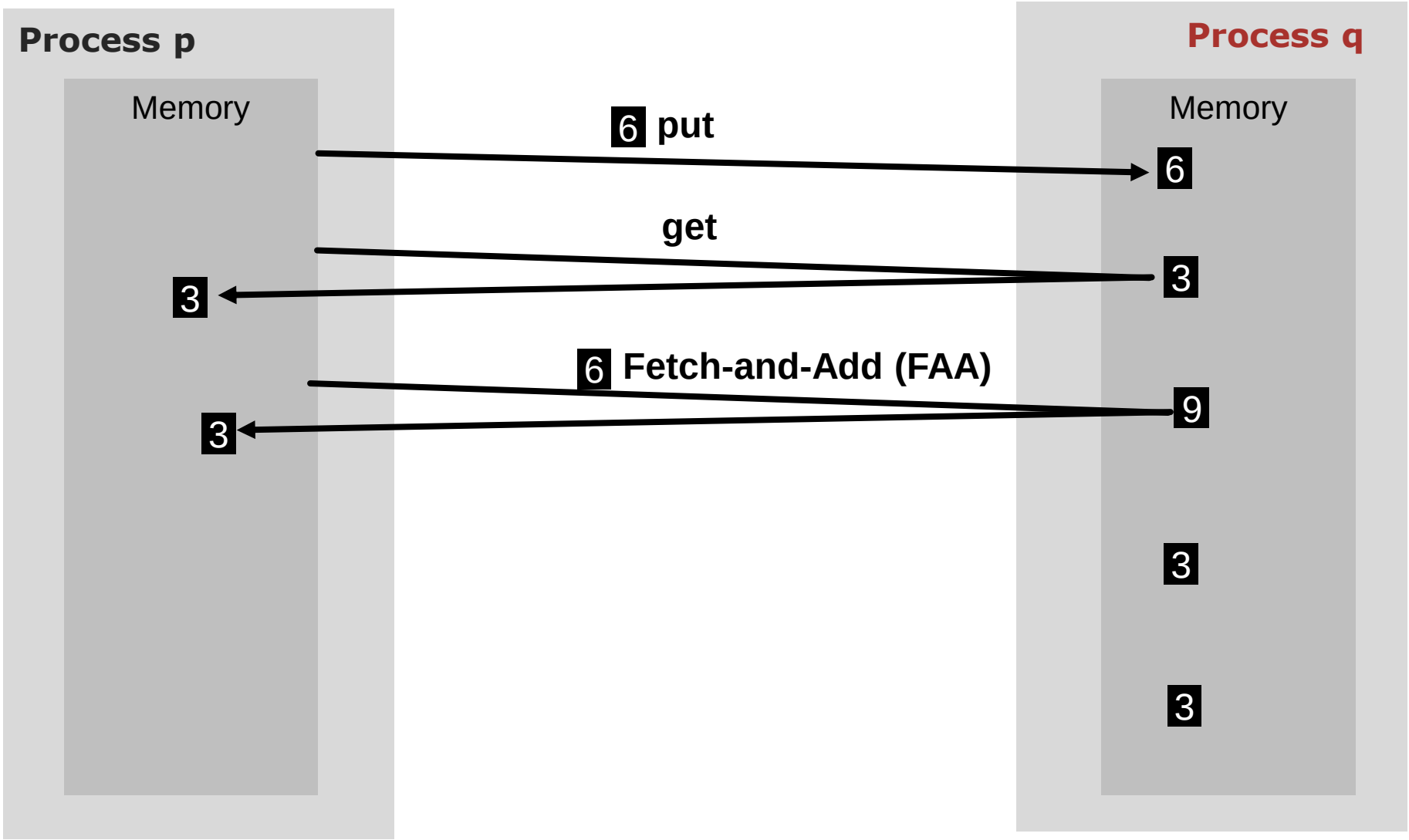
RMA-RW
Required Operations



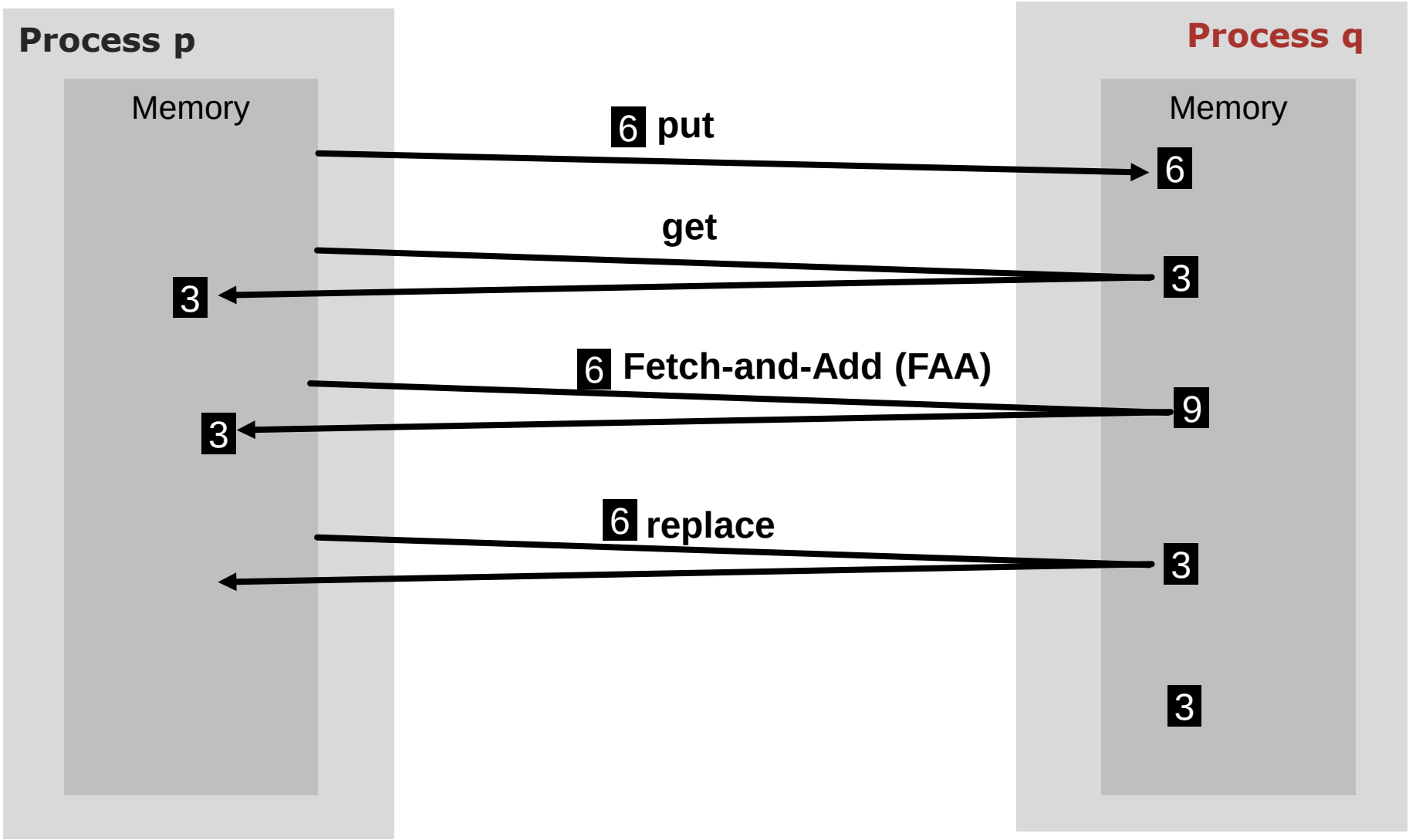
RMA-RW
Required Operations



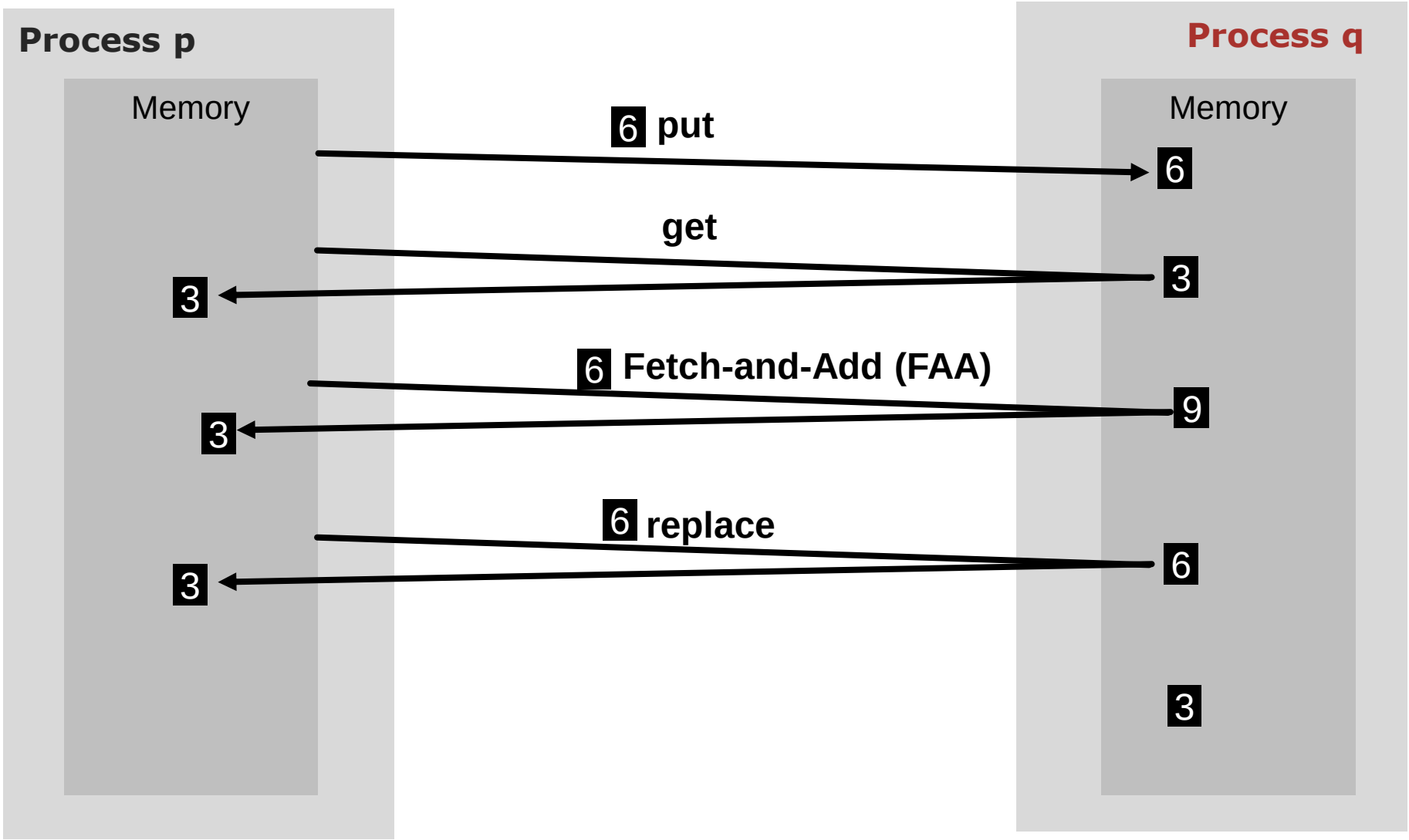
RMA-RW
Required Operations



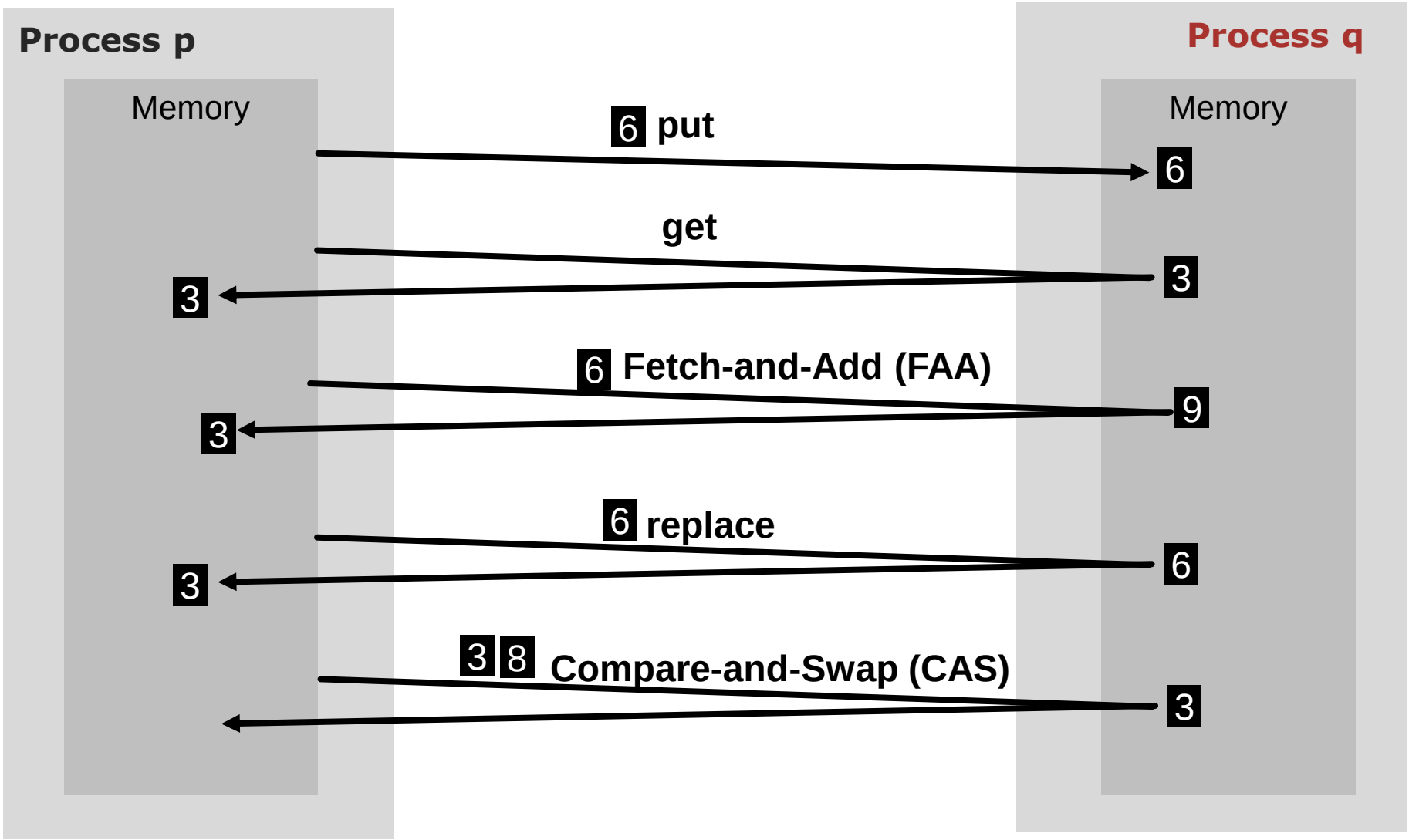
RMA-RW
Required Operations



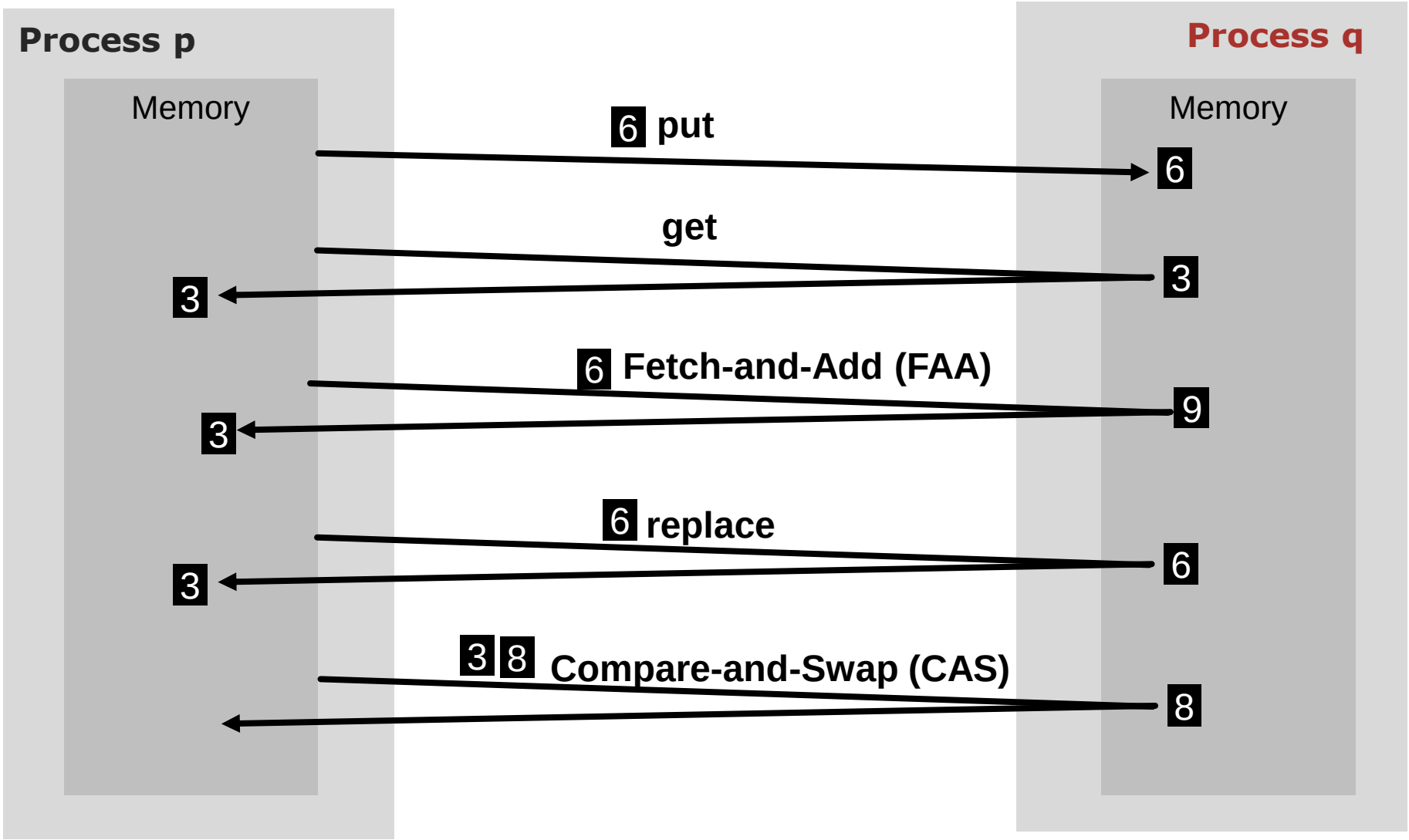
RMA-RW
Required Operations



RMA-RW
Required Operations



RMA-RW
Required Operations



RMA-RW
Required Operations