Patronus: High-Performance and Protective Remote Memory

Bin Yan, Youyou Lu, Qing Wang, Minhui Xie, Jiwu Shu

Tsinghua University

Remote Memory

Remote memory architecture

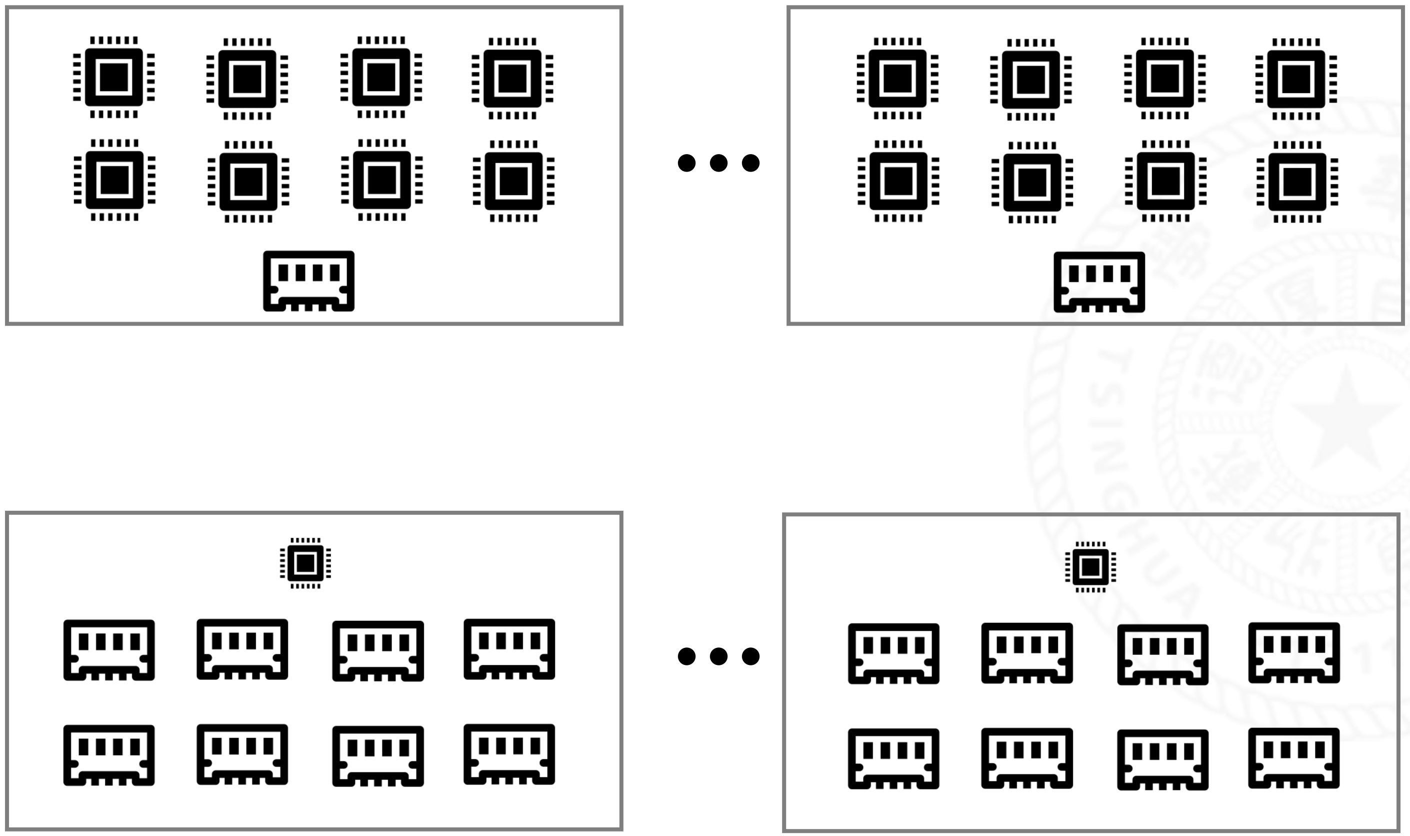
❖ Physically separate CPU and memory into network-attached components

many CPU cores,
small local DRAM

Compute Nodes
(CNs)

large DRAM,
several wimpy cores

Memory Nodes
(MNs)



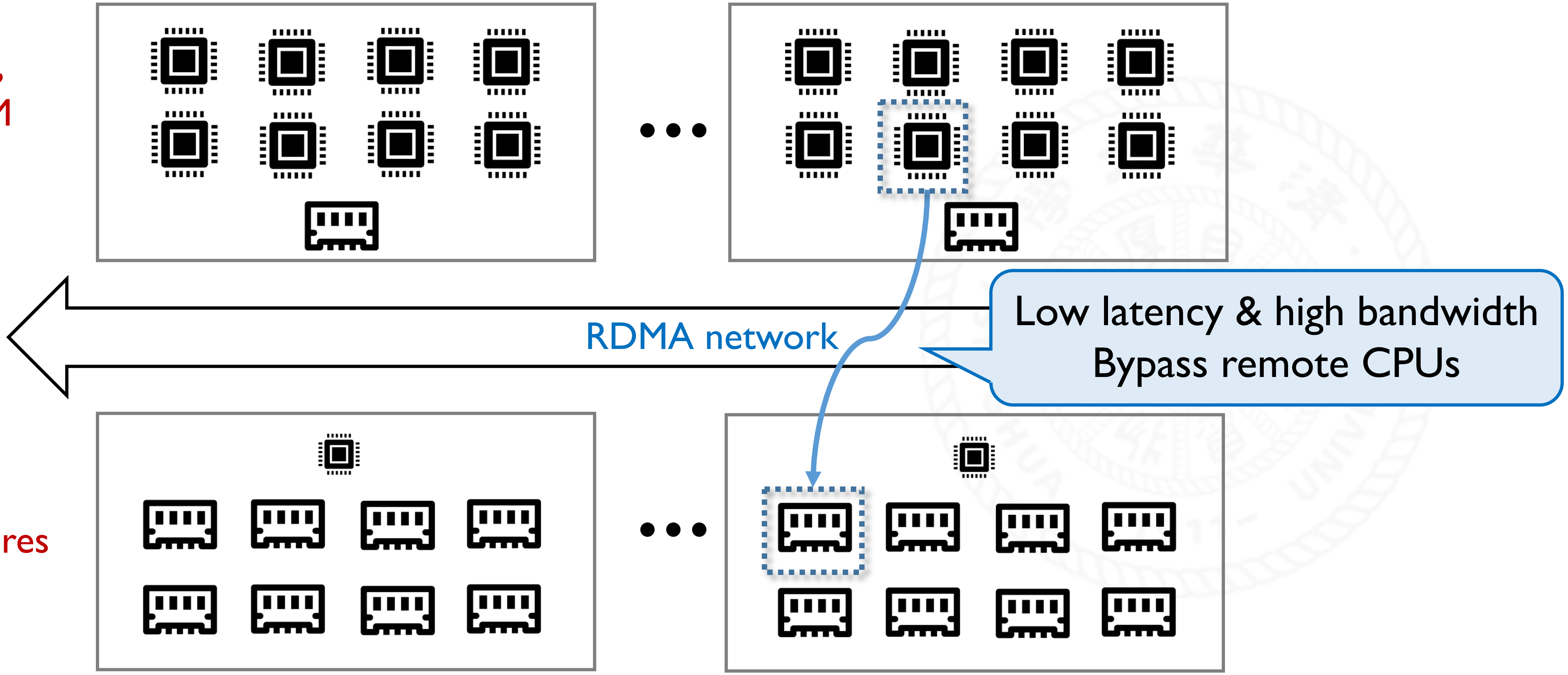
Remote Memory

Remote memory architecture

❖ Physically separate CPU and memory into network-attached components

many CPU cores,
small local DRAM

Compute Nodes
(CNs)



large DRAM,
several wimpy cores

Memory Nodes
(MNs)

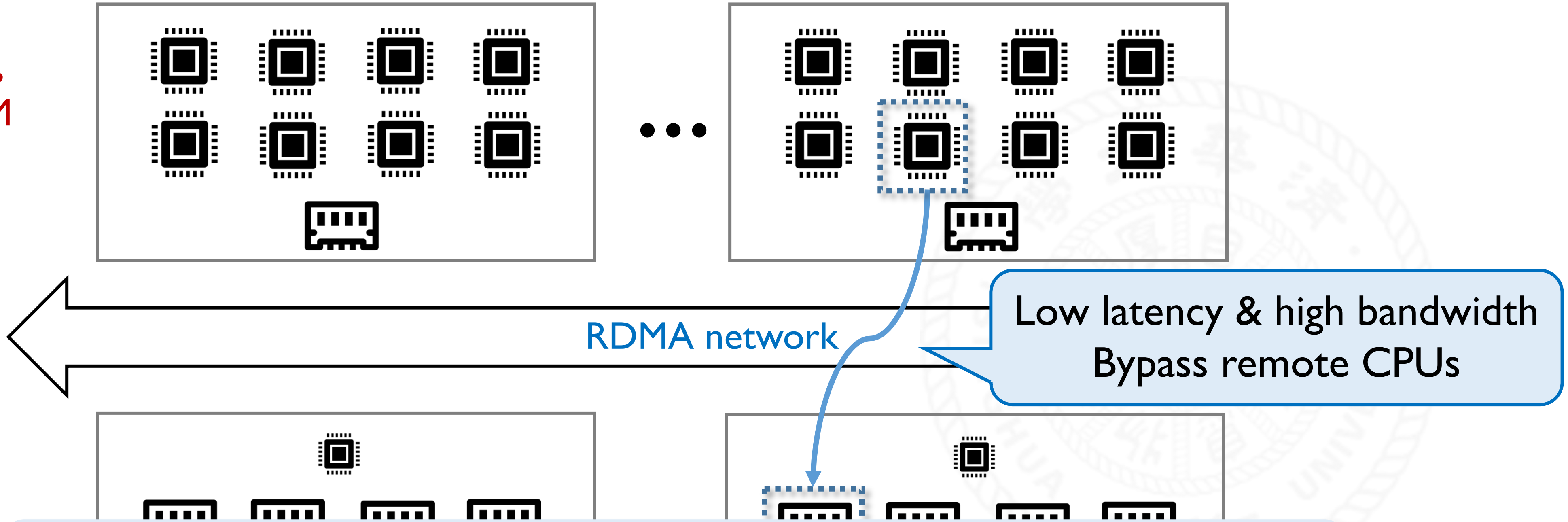
Remote Memory

Remote memory architecture

❖ Physically separate CPU and memory into network-attached components

many CPU cores,
small local DRAM

Compute Nodes
(CNs)



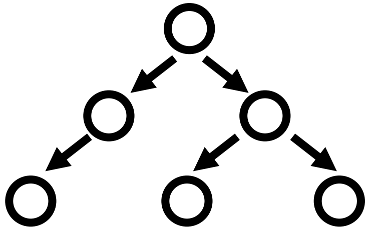
large DRAM,
several wimpy co

Memory Nodes
(MNs)

RDMA-enabled remote memory is widely deployed
to improve memory utilization

Remote Memory

Many efforts to make remote memory practical on multiple fronts



efficient remote indexes

[ATC'21,SIGMOD'22,HotOS'19]

easy-to-use

programming models

[OSDI'20,ATC'18,SoCC'17]

popular applications

and more

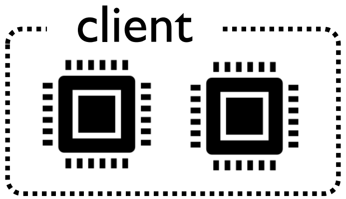
[FAST22, OSDI'18, ATC'15]

However, protection for remote memory is not explored

Necessity: Protection for Remote Memory

Unprotected RM fails to avoid application anomalies

Example 1 **buggy/malicious clients**
(access illegal address)



(out-of-bound)

`write(0x00-0xff)`

corrupt data!

(unregulated)

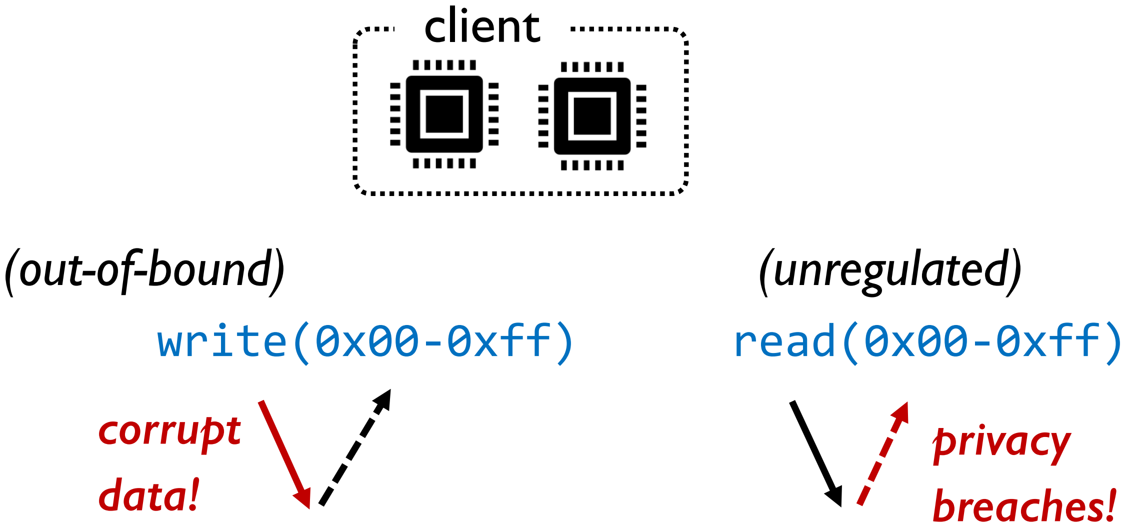
`read(0x00-0xff)`

privacy breaches!

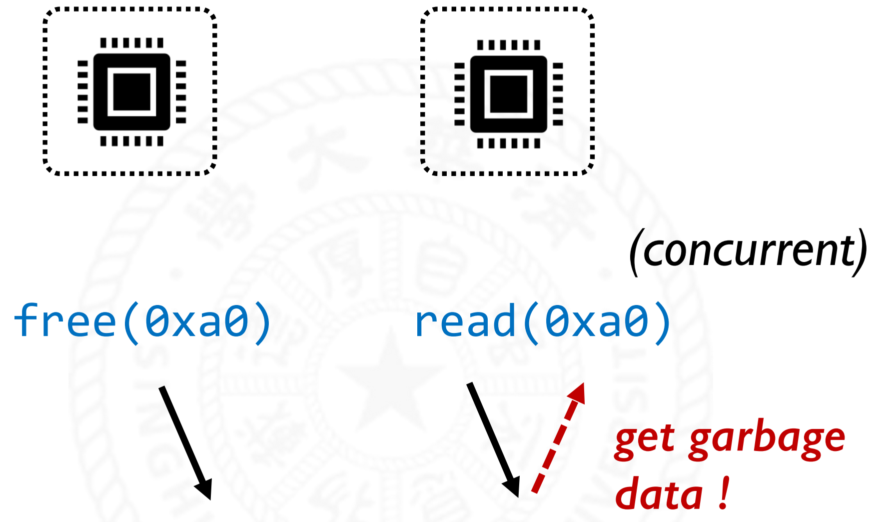
Necessity: Protection for Remote Memory

Unprotected RM fails to avoid application anomalies

Example I buggy/malicious clients
(access illegal address)



Example II memory management race
(access at illegal time)



Necessity: Protection for Remote Memory

Unprotected RM fails to avoid application anomalies

Example I buggy/malicious clients
(access illegal address)

client

(out-of-bound)
`write(0x00-0xff)`
corrupt data!

(unregulated)
`read(0x00-0xff)`
privacy breaches!

Example II memory management race
(access at illegal time)

`free(0xa0)`

(concurrent)
`read(0xa0)`
get garbage data !

RM protection is necessary
especially for workloads with shared access patterns

Difficulty: Protection + Performance (I)

It is difficult to achieve high-performance protection on the common path

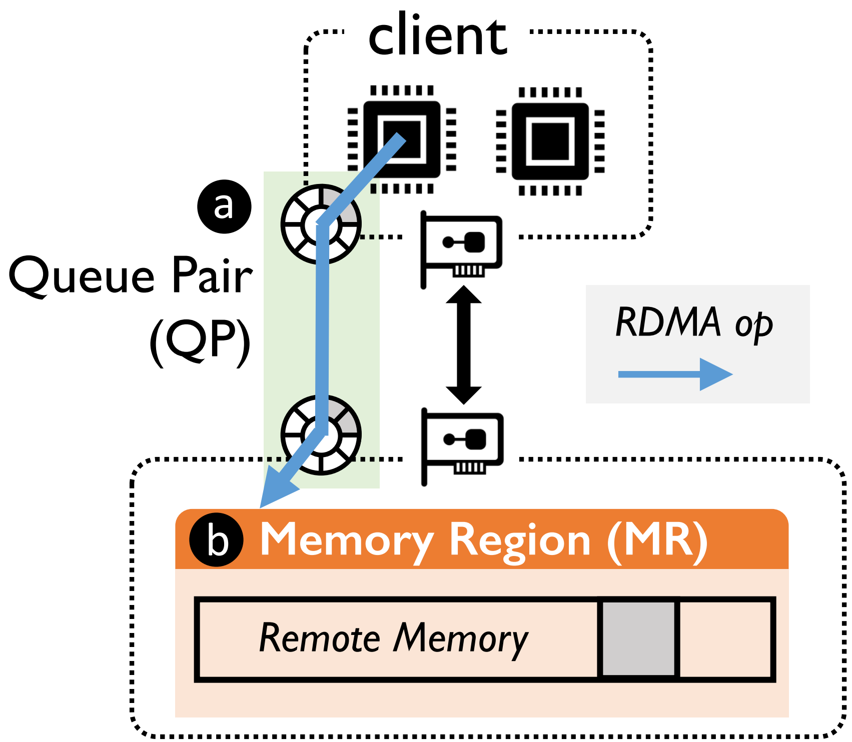
- Reason 1: CPUs are weak on memory nodes
- Reason 2: Existing protection mechanisms are expensive

Difficulty: Protection + Performance (I)

It is difficult to achieve high-performance protection on the common path

Reason 1: CPUs are weak on memory nodes

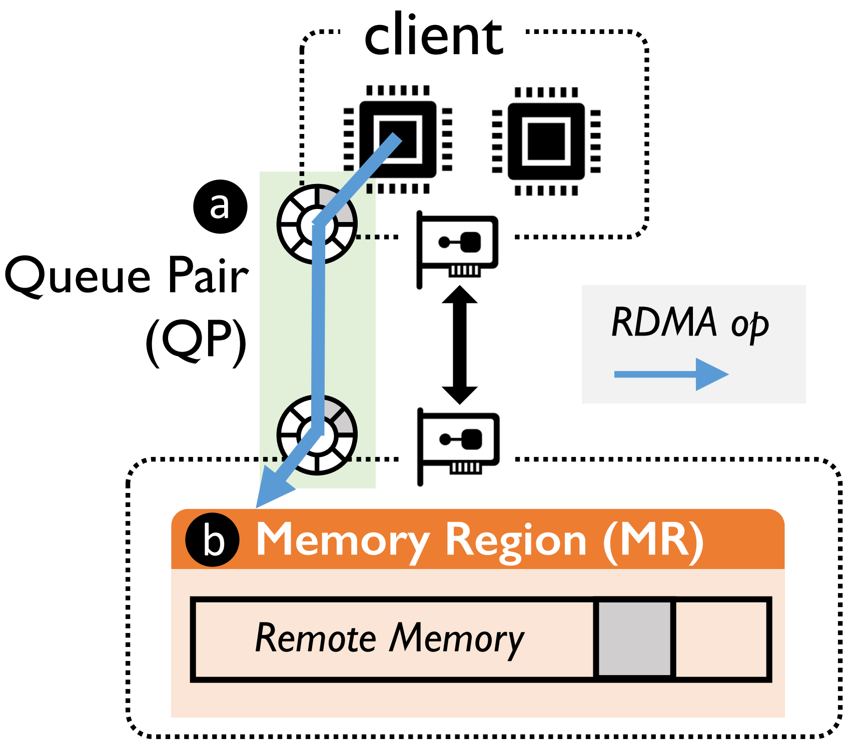
Reason 2: Existing protection mechanisms are expensive

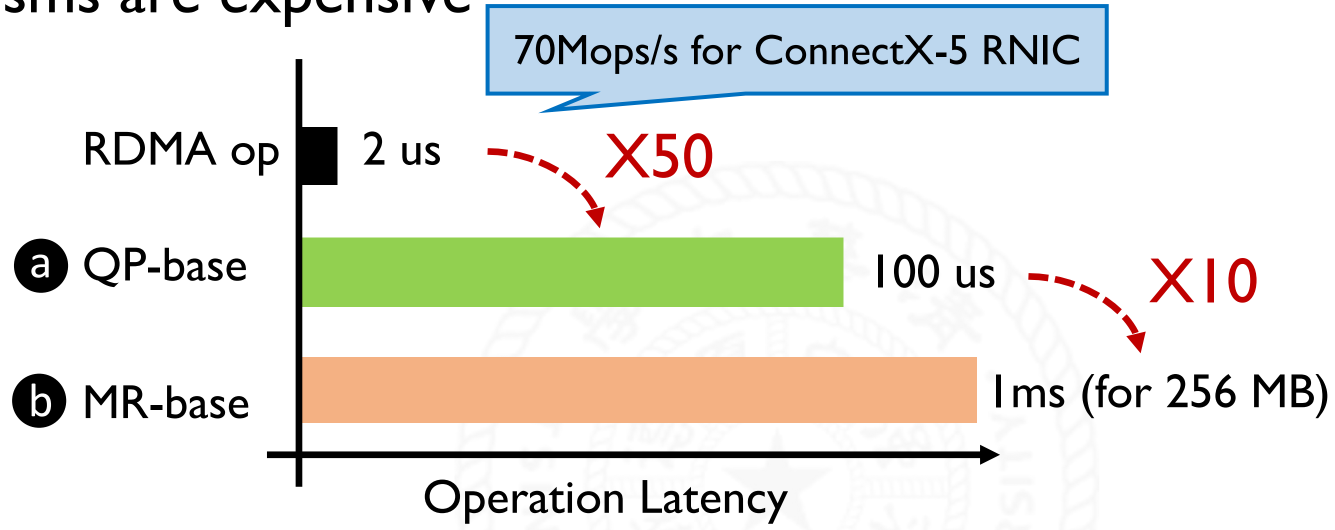


Difficulty: Protection + Performance (I)

It is difficult to achieve high-performance protection on the common path

- Reason 1: CPUs are weak on memory nodes
- Reason 2: Existing protection mechanisms are expensive



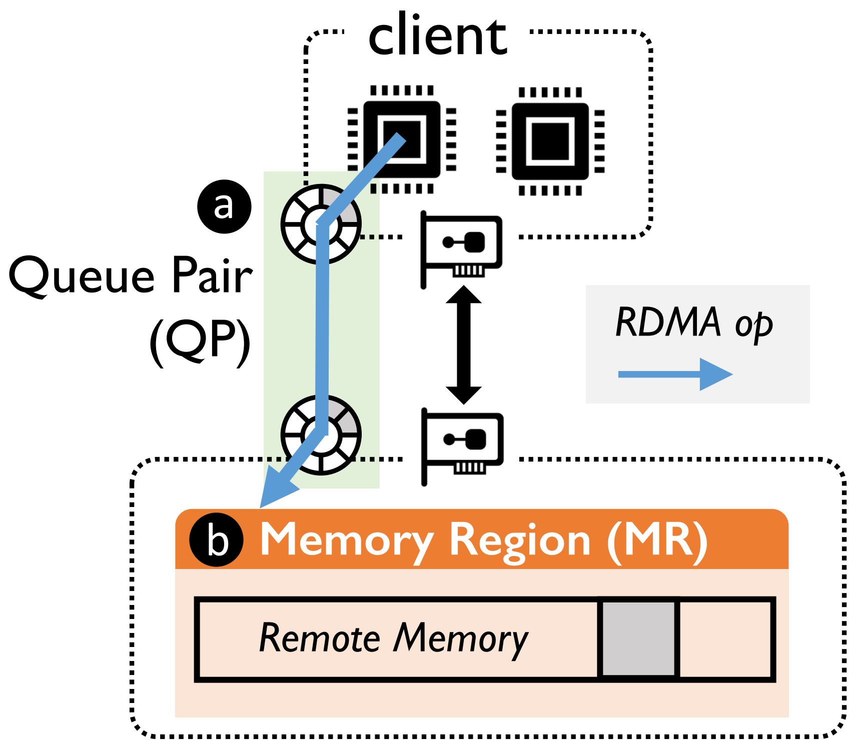


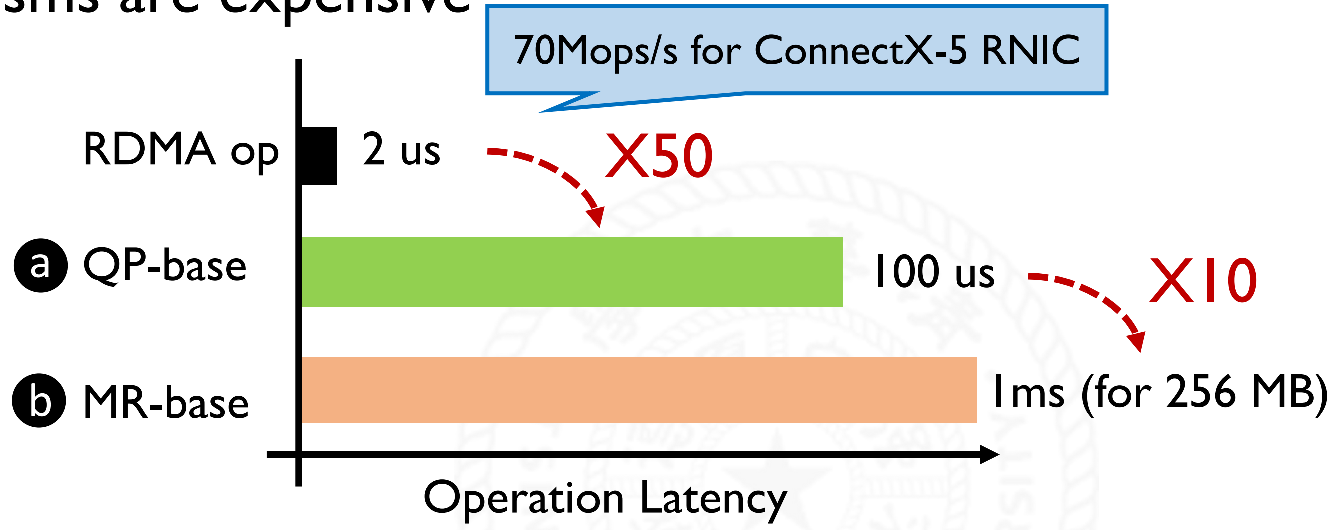
Difficulty: Protection + Performance (I)

It is difficult to achieve high-performance protection on the common path

Reason 1: CPUs are weak on memory nodes

Reason 2: Existing protection mechanisms are expensive





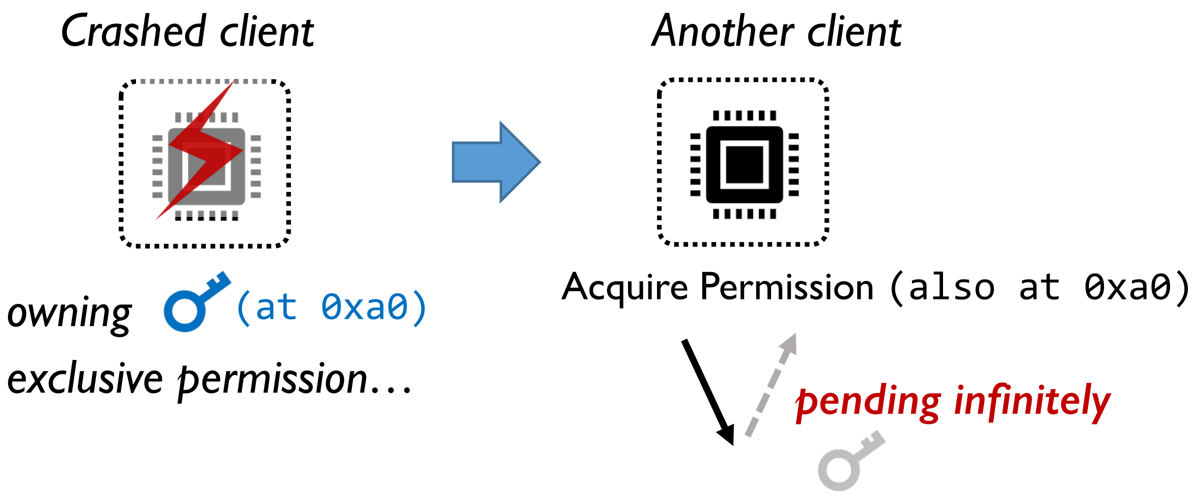
Existing mechanisms are <u>50X-500X slower</u> than RDMA data path.

EXAMPLE Existing applications (FaRM [NSDI'14], Octopus [ATC'17]) have to use 2GB coarse-grained MR, leaving RM no protection.

Difficulty: Protection + Robustness (II)

It is difficult to remain robust on the failure path:

Exception I Client failures

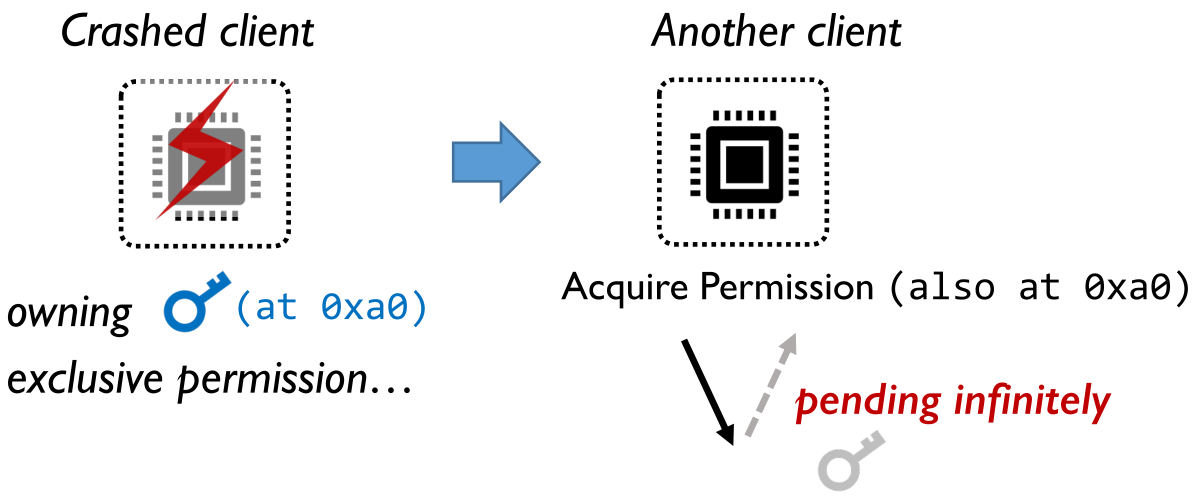


Client failures impact system progress

Difficulty: Protection + Robustness (II)

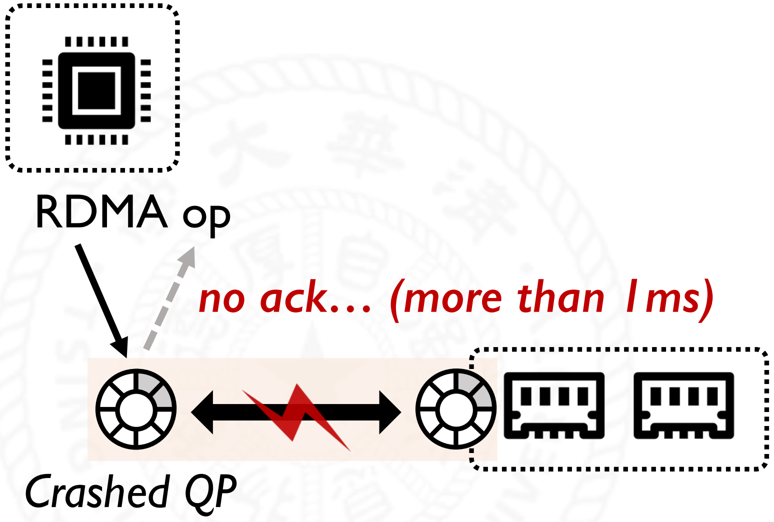
It is difficult to remain **robust** on the **failure** path:

Exception I Client failures



Client failures impact system progress

Exception II QP failures



QP failures interrupt application execution

Goal - Protective System

- RM systems are performance-critical → Fast protection management on par with data path
- Client failures impact system progress → React fast to client failures
- QP failures interrupt application execution → Retain performance under QP failures

Goal - Protective System

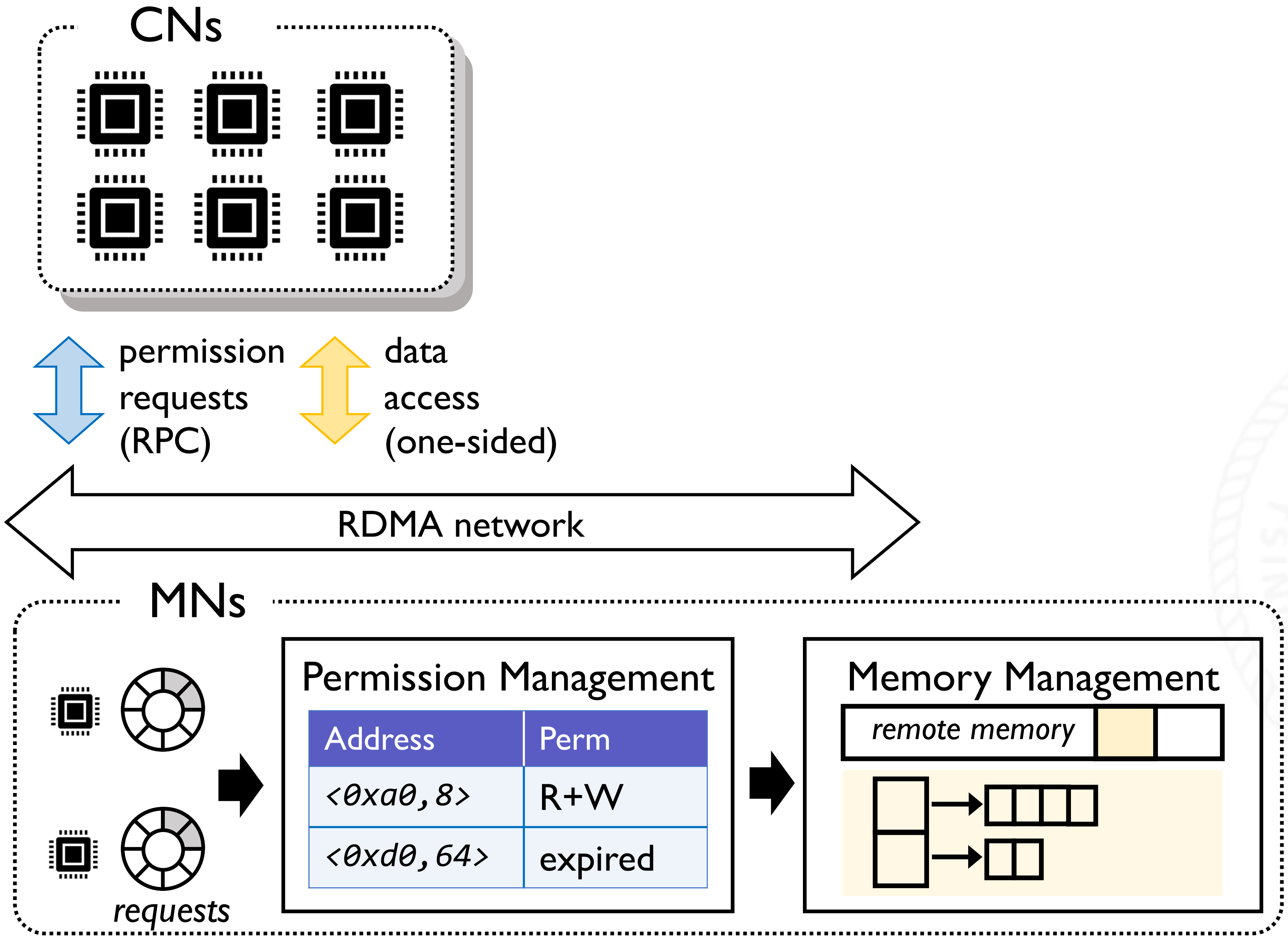
- RM systems are performance-critical → Fast protection management on par with data path
- Client failures impact system progress → React fast to client failures
- QP failures interrupt application execution → Retain performance under QP failures

Patronus: a protective RM system that is
high-performance and robust under all situations

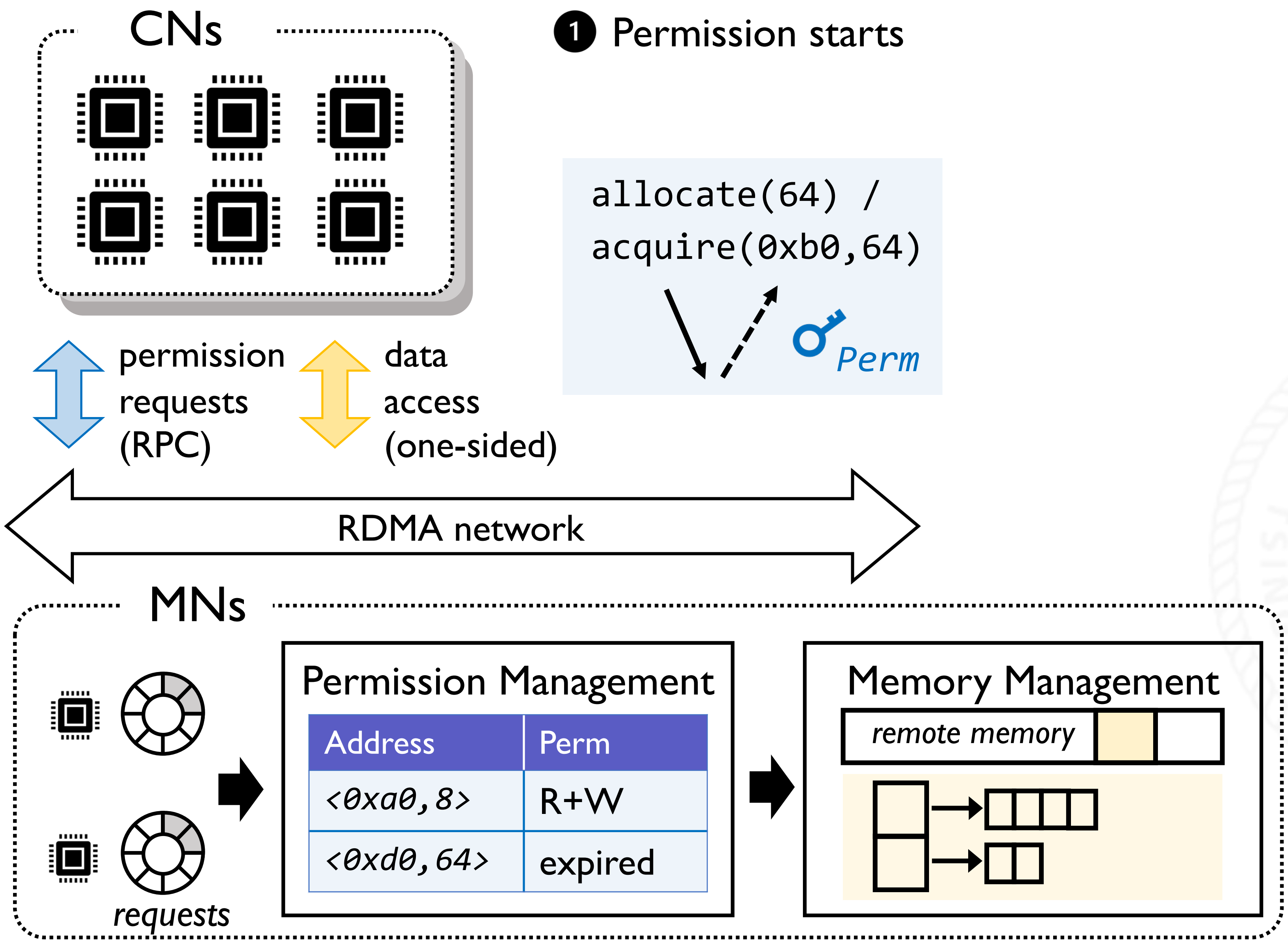
Outline

- ❖ Background & Motivation
- ❖ Patronus – High-Performance Protective Remote Memory
- ❖ Results
- ❖ Conclusion

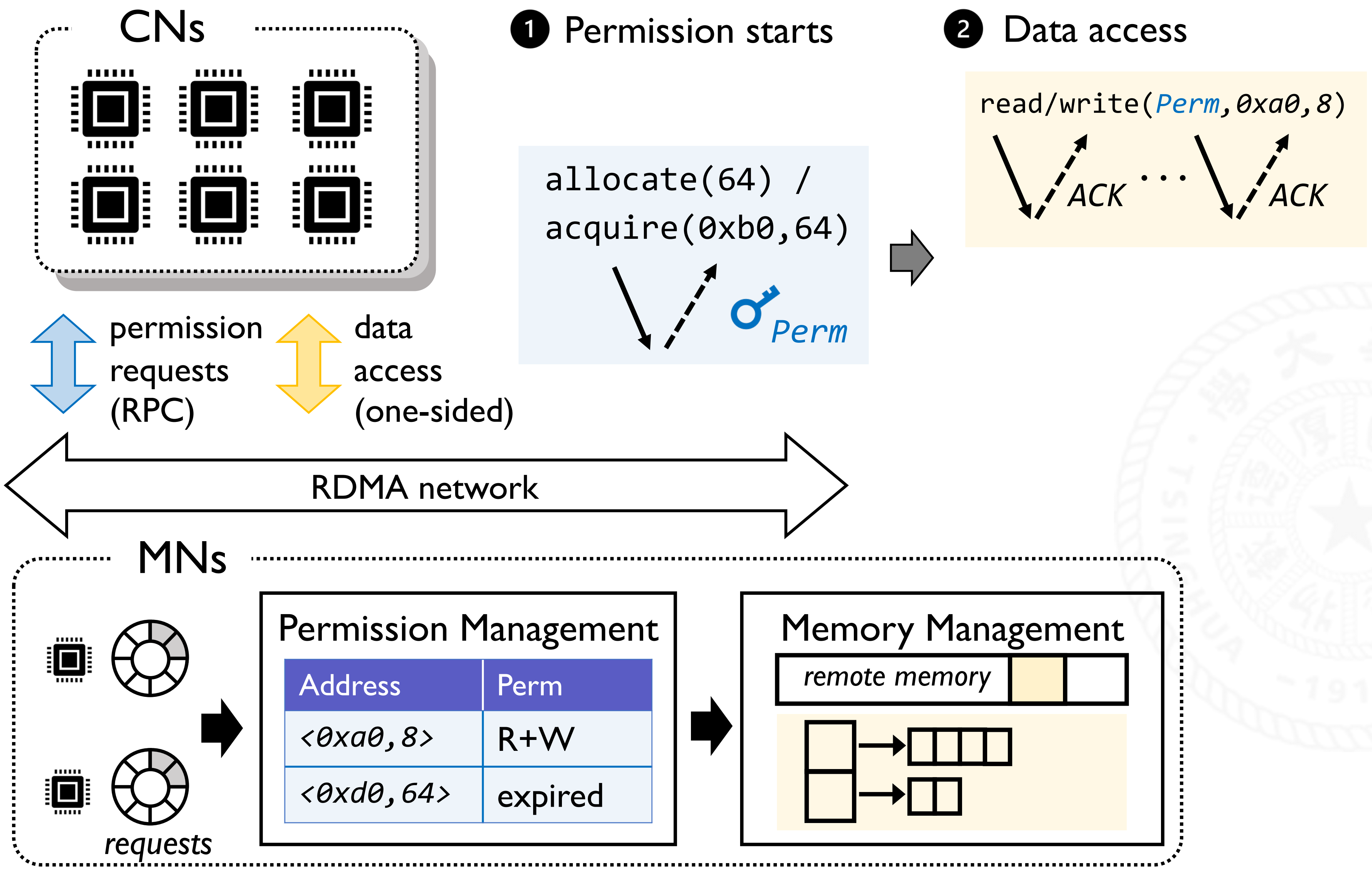
Patronus Overview



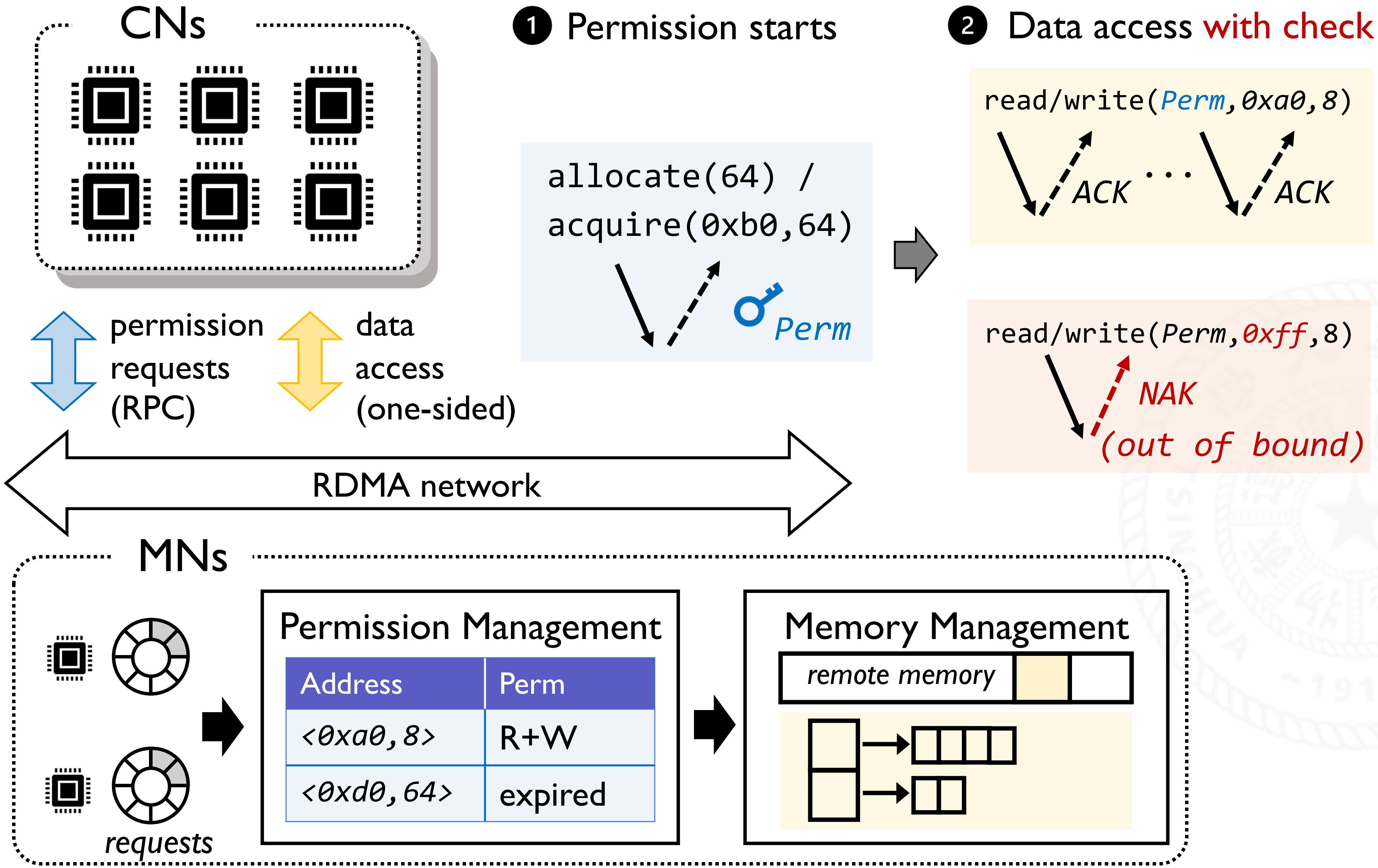
Patronus Overview



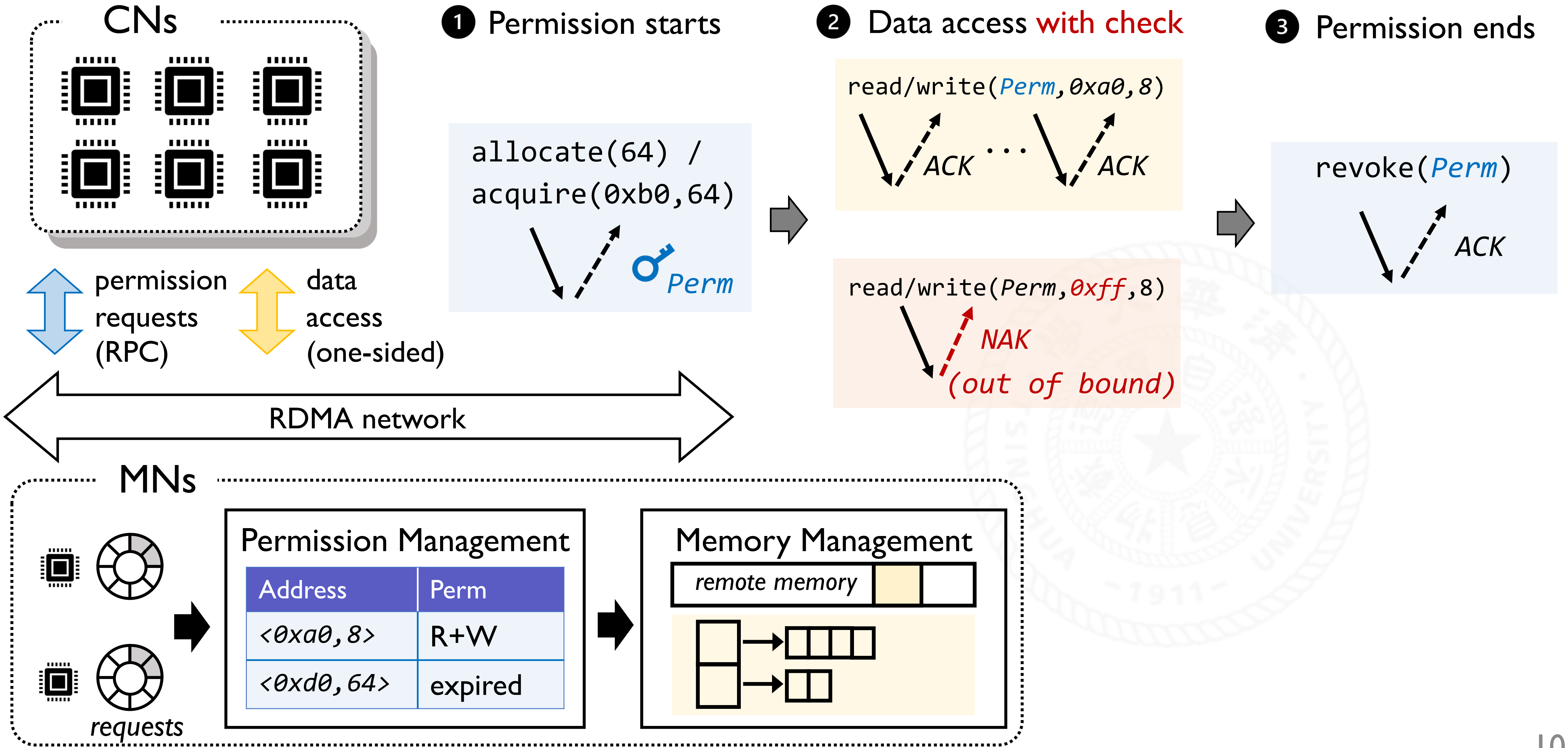
Patronus Overview



Patronus Overview



Patronus Overview



Opportunity – Memory Window (MW)

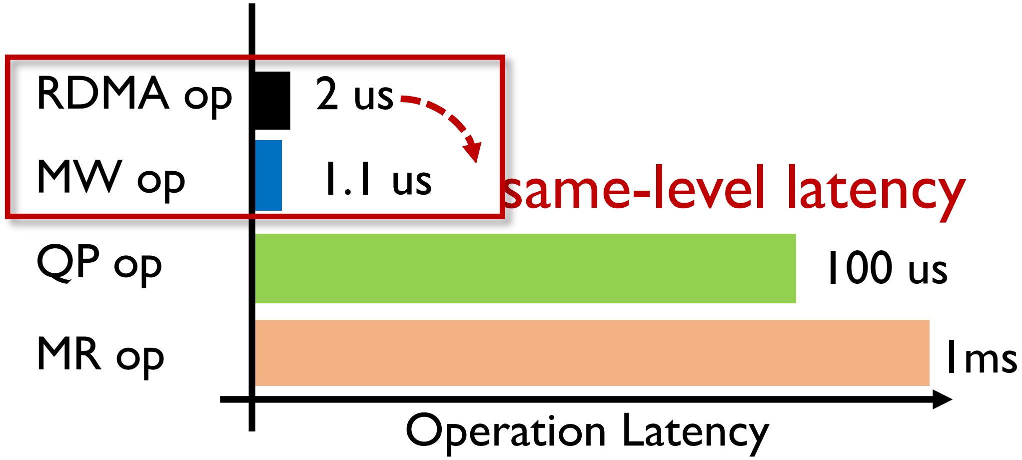
Memory Window: an advanced & light-weight protection mechanism

- Patronus leverages MWs for fast permission management.

Opportunity – Memory Window (MW)

Memory Window: an advanced & light-weight protection mechanism

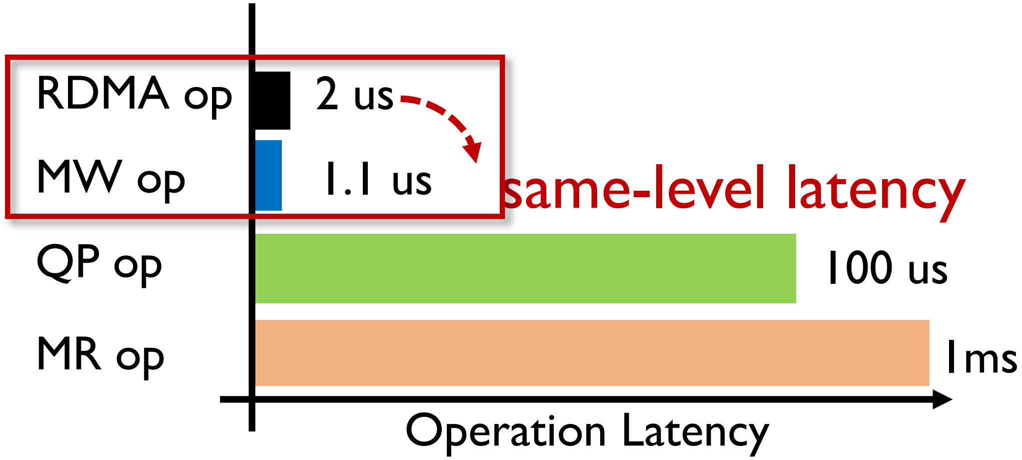
❖ Patronus leverages MWs for fast permission management.



Opportunity – Memory Window (MW)

Memory Window: an advanced & light-weight protection mechanism

- Patronus leverages MWs for fast permission management.

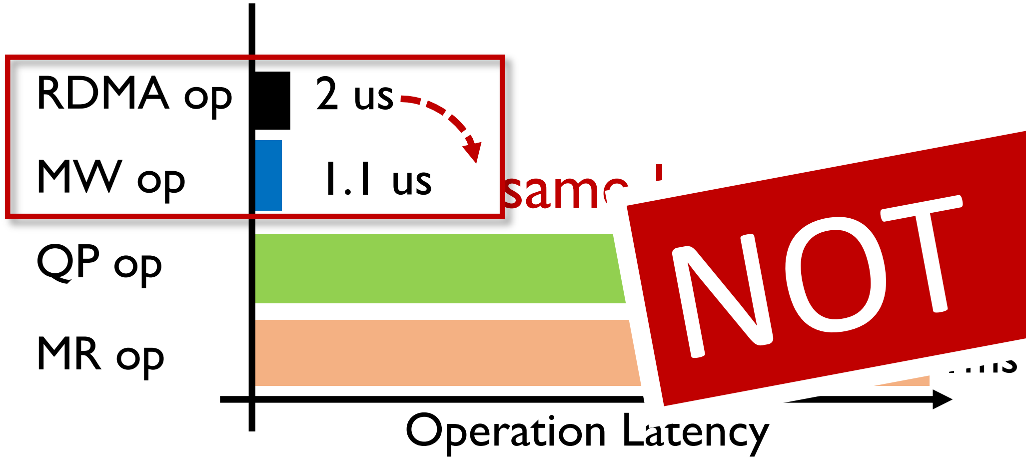


Action	MW	Effect
Permission starts	`bind`	-> 🔑
Permission ends	`unbind`	✗🔑
(rebind = unbind + bind)		

Opportunity – Memory Window (MW)

Memory Window: an advanced & light-weight protection mechanism

- Patronus leverages MWs for fast permission management.



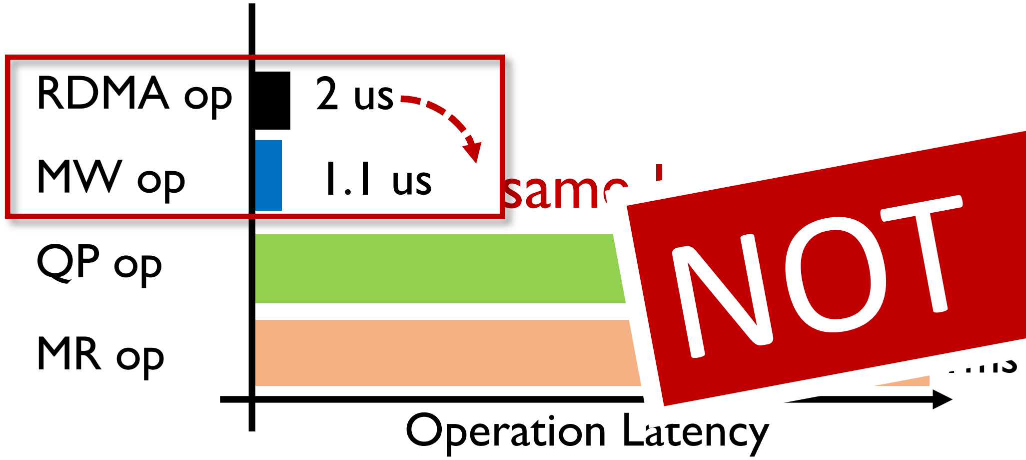
NOT ENOUGH

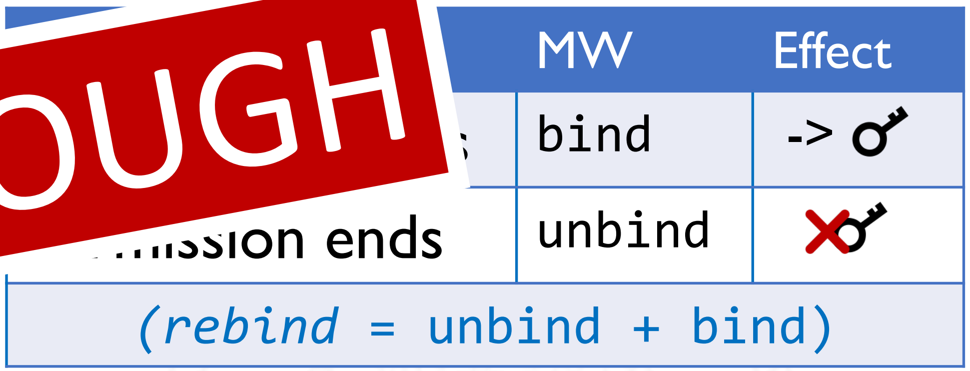
	MW	Effect
s	bind	->
ission ends	unbind	
(rebind = unbind + bind)		

Opportunity – Memory Window (MW)

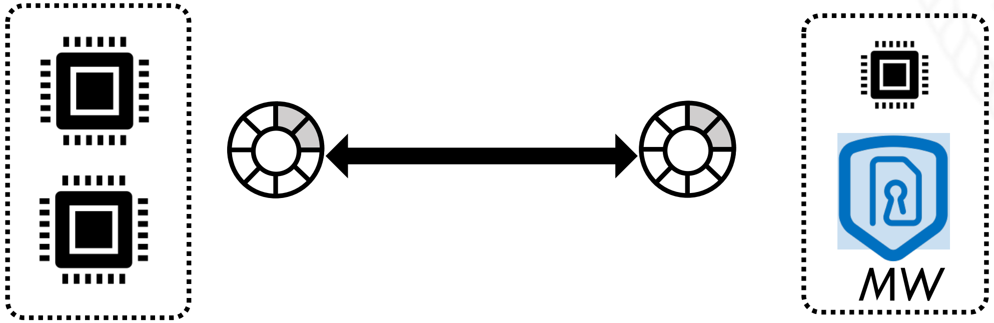
Memory Window: an advanced & light-weight protection mechanism

- Patronus leverages MWs for fast permission management.





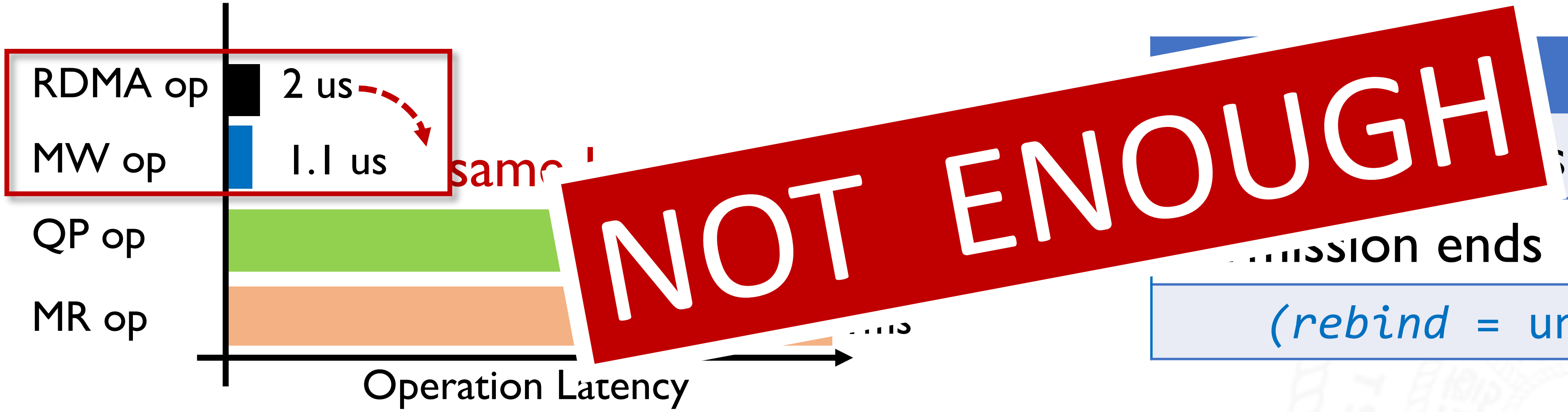
NOT ENOUGH



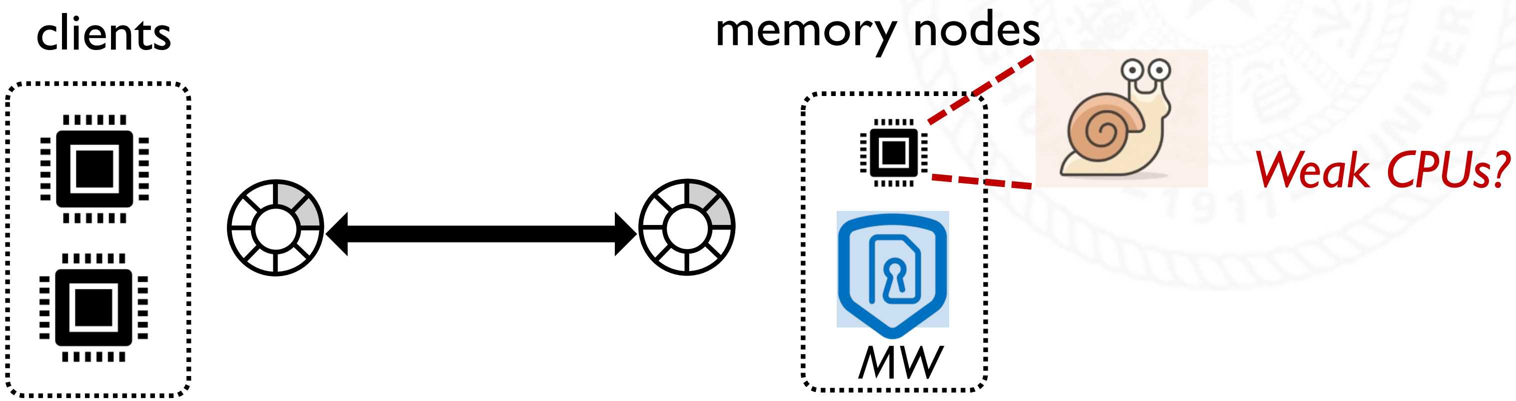
Opportunity – Memory Window (MW)

Memory Window: an advanced & light-weight protection mechanism

- Patronus leverages MWs for fast permission management.



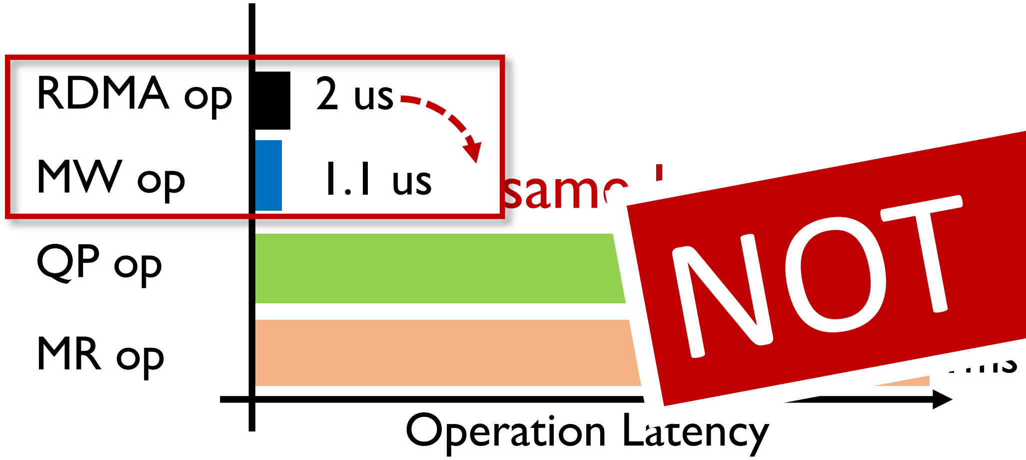
	MW	Effect
	bind	->
...ission ends	unbind	
(rebind = unbind + bind)		



Opportunity – Memory Window (MW)

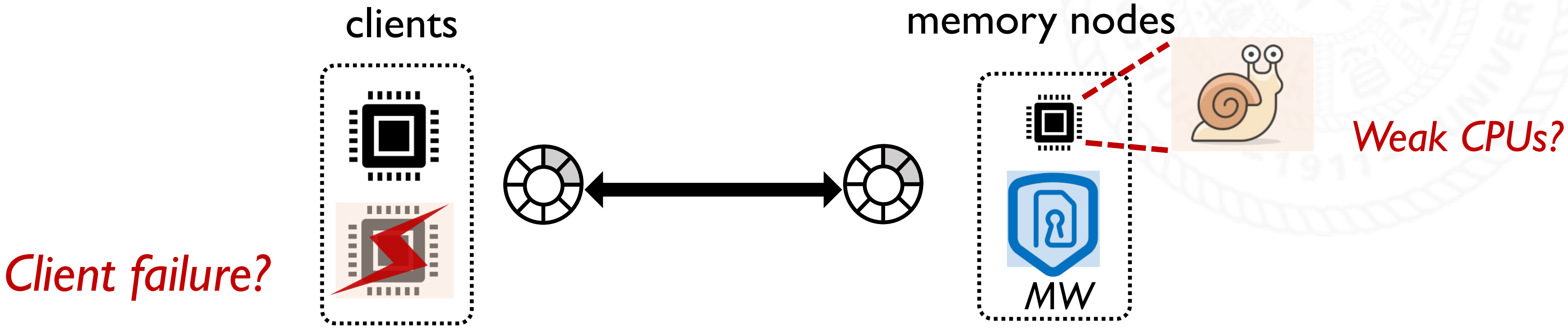
Memory Window: an advanced & light-weight protection mechanism

- Patronus leverages MWs for fast permission management.



NOT ENOUGH

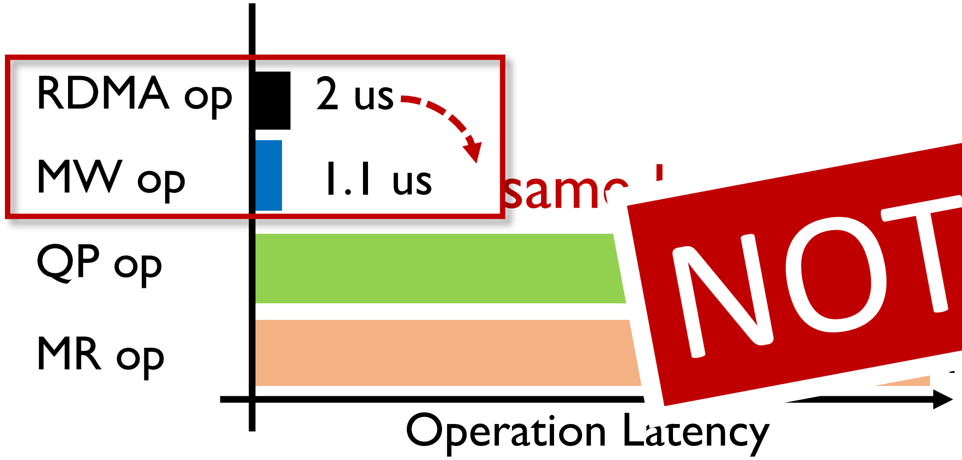
	MW	Effect
	bind	->
…ission ends	unbind	
(rebind = unbind + bind)		



Opportunity – Memory Window (MW)

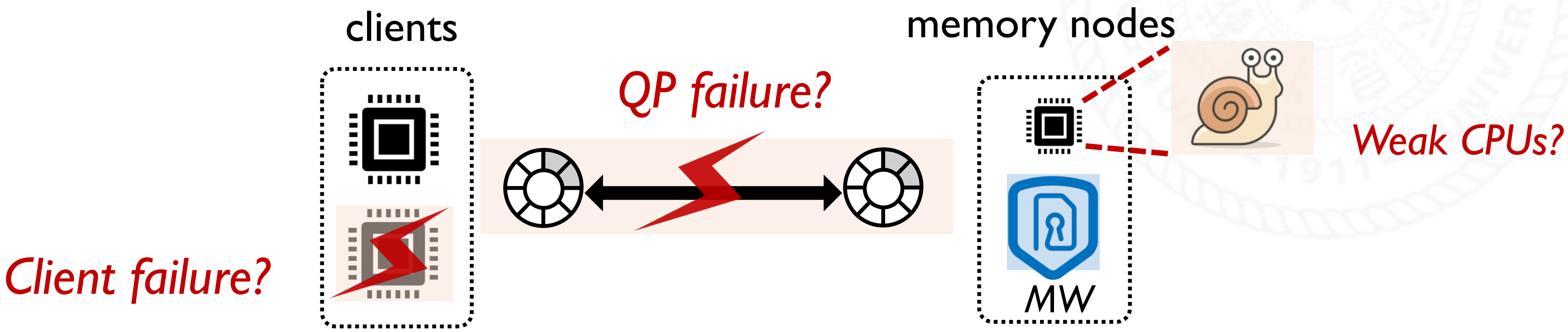
Memory Window: an advanced & light-weight protection mechanism

❖ Patronus leverages MWs for fast permission management.



NOT ENOUGH

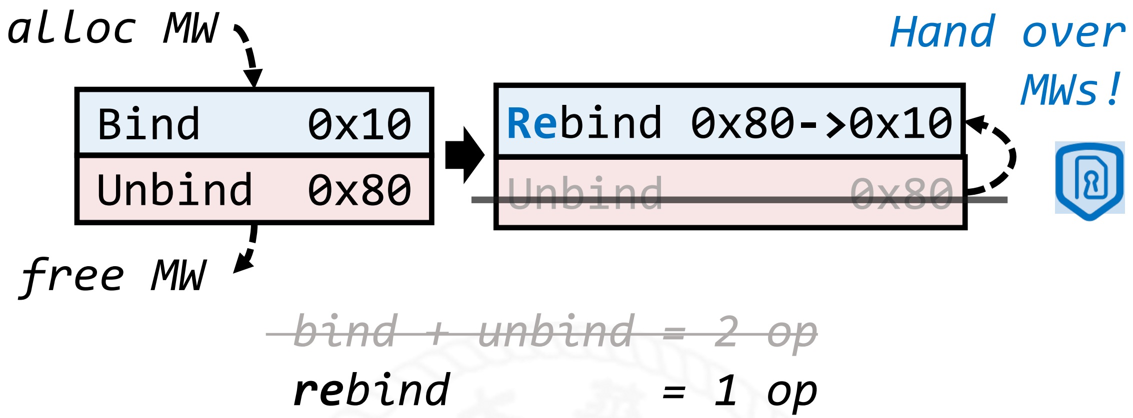
	MW	Effect
	bind	->
…mission ends	unbind	
(rebind = unbind + bind)		



Technique (1) – MW Operation Reduction

❖ MW handover

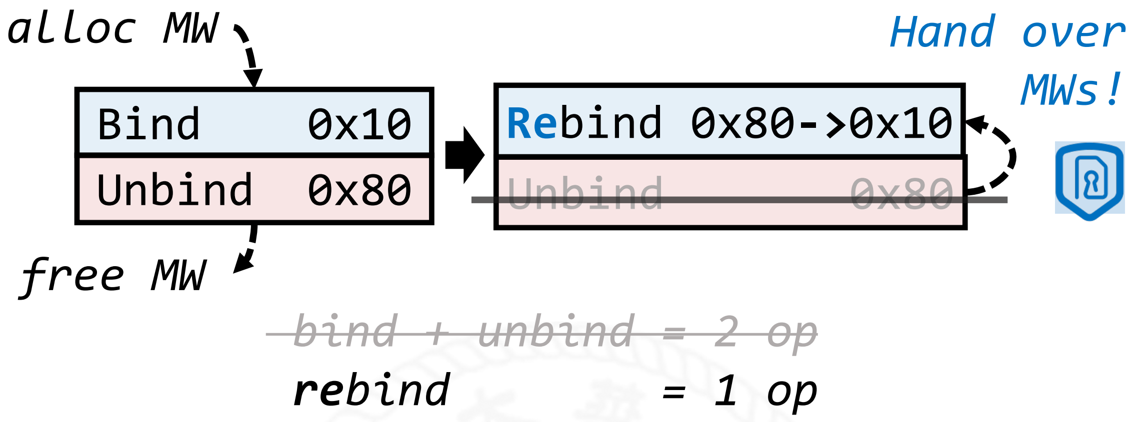
- Observation: binding and unbinding ops co-exist
- *Bind* op + *unbind* op => *rebind* op
- Hand over MWs between requests
- Result: reduce *half* of MW ops



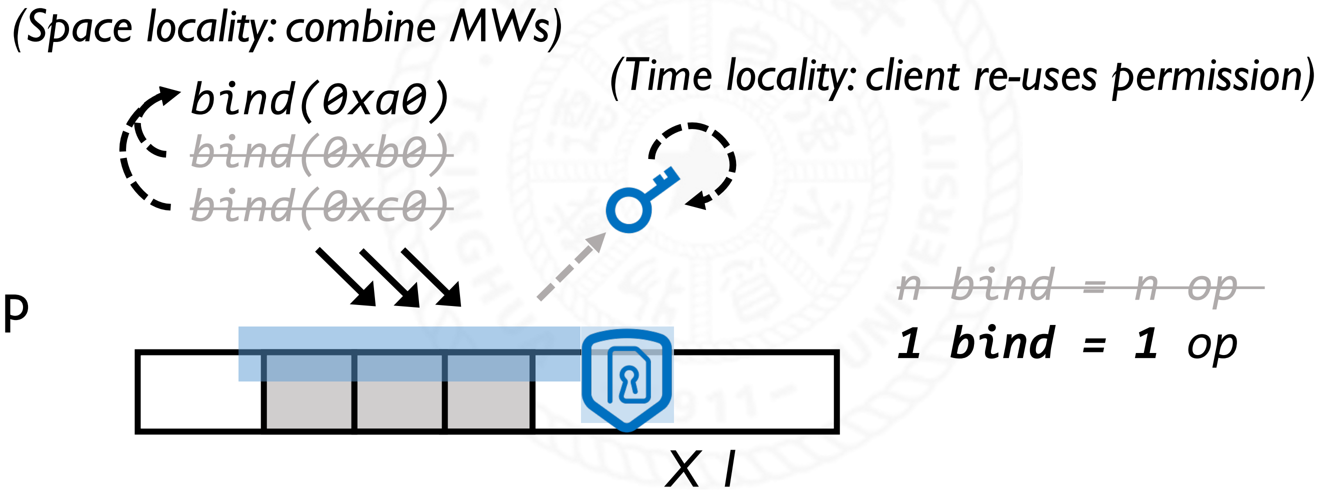
Same semantics can be achieved with fewer operations

Technique (I) – MW Operation Reduction

- MW handover
  - Observation: binding and unbinding ops co-exist
  - *Bind* op + *unbind* op => *rebind* op
  - Hand over MWs between requests
  - Result: reduce *half* of MW ops



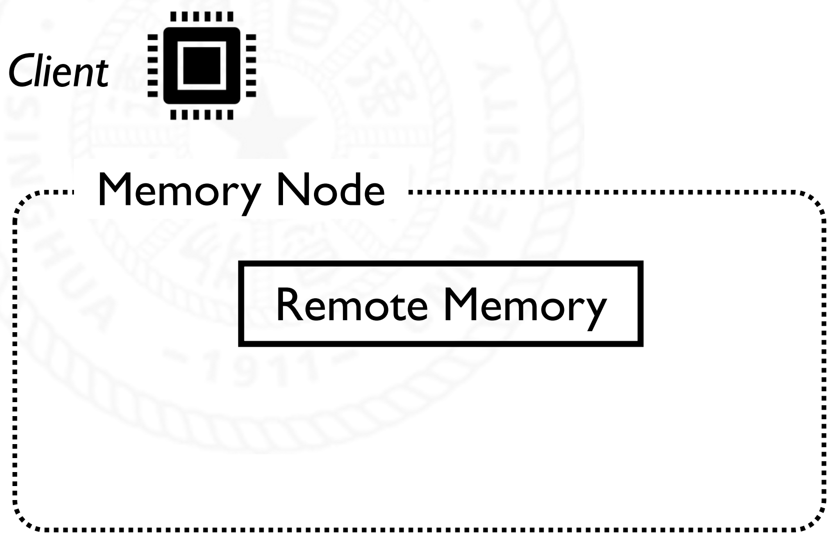
- Exploit locality
  - Observation: space and time locality
  - Multiple *bind* ops (w/ locality) => one *bind* op
  - Result: reduce binding MW ops



Same semantics can be achieved with fewer operations

Technique (2) – Lease for Client Failures

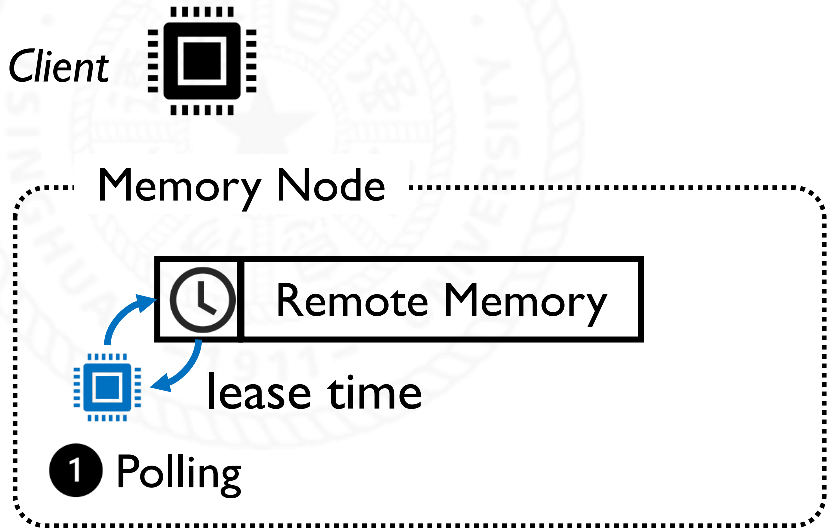
Use *leases* to handle client failures



Technique (2) – Lease for Client Failures

Use *leases* to handle client failures

- Equip MWs with automatic reclamation
- Memory nodes poll for expiration periodically (❶)

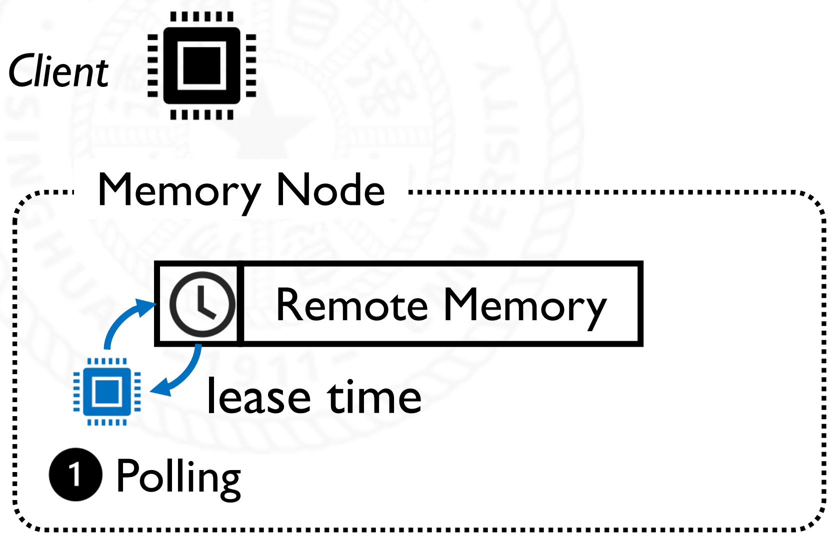


Technique (2) – Lease for Client Failures

Use *leases* to handle client failures

- ❖ Equip MWs with automatic reclamation
- ❖ Memory nodes poll for expiration periodically (❶)

Offloads lease extension overhead to compute nodes



Technique (2) – Lease for Client Failures

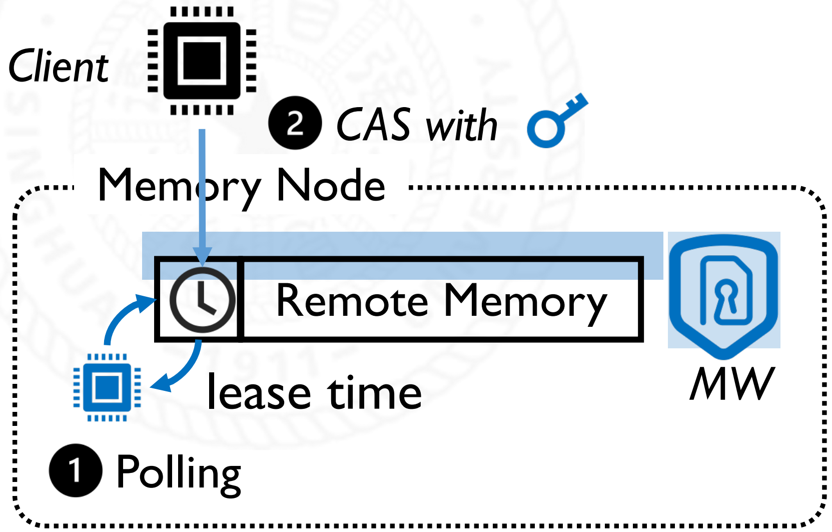
Use *leases* to handle client failures

- Equip MWs with automatic reclamation
- Memory nodes poll for expiration periodically (1)

Offloads lease extension overhead to compute nodes

Client-collaborated lease extension

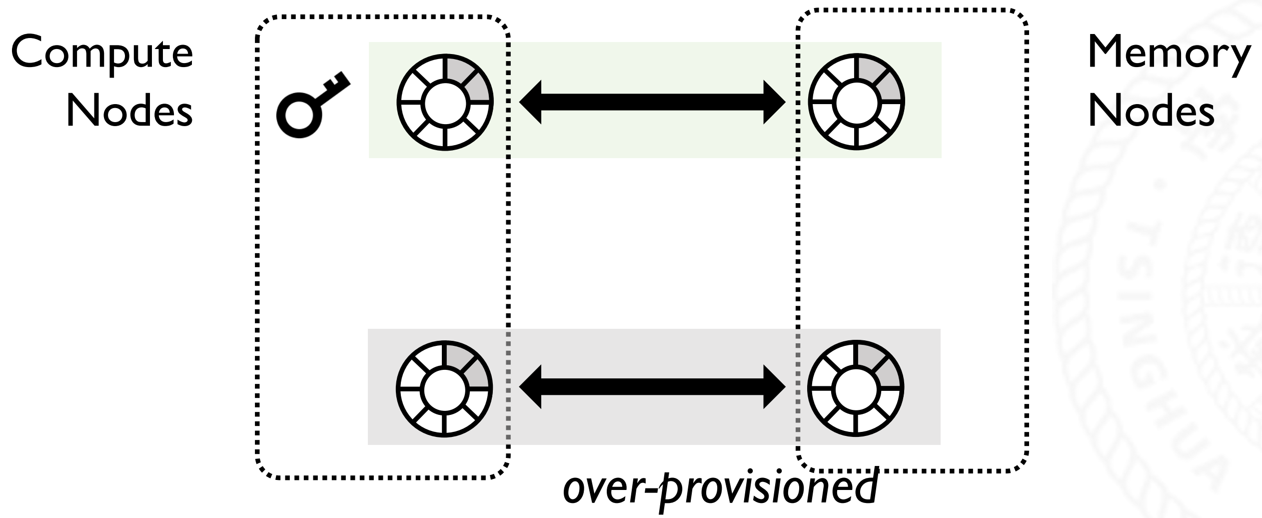
- Enabler: MWs are byte-granularity to expose variables
- Expose the `lease_time` variable ( ) to clients
- Clients extend permission via RDMA_CAS (2)
- Result: Extension only costs one in-bound RDMA op



Technique (3) – Over-Provisioning for QP failures

Over-provision QPs to hide interruption from QP failures

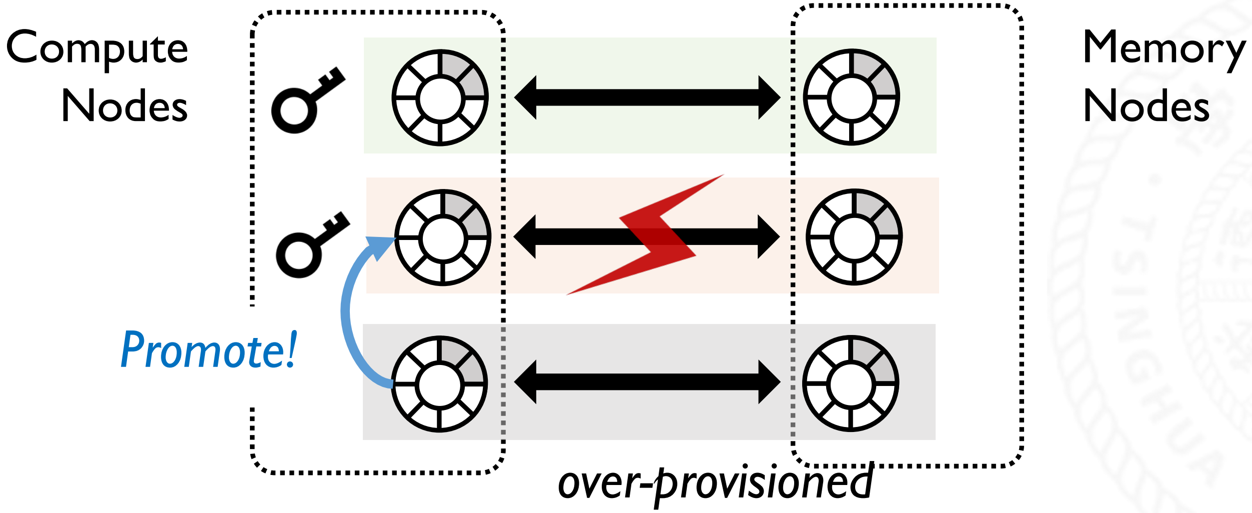
❖ On QP failures: promote a healthy QP as substitution



Technique (3) – Over-Provisioning for QP failures

Over-provision QPs to hide interruption from QP failures

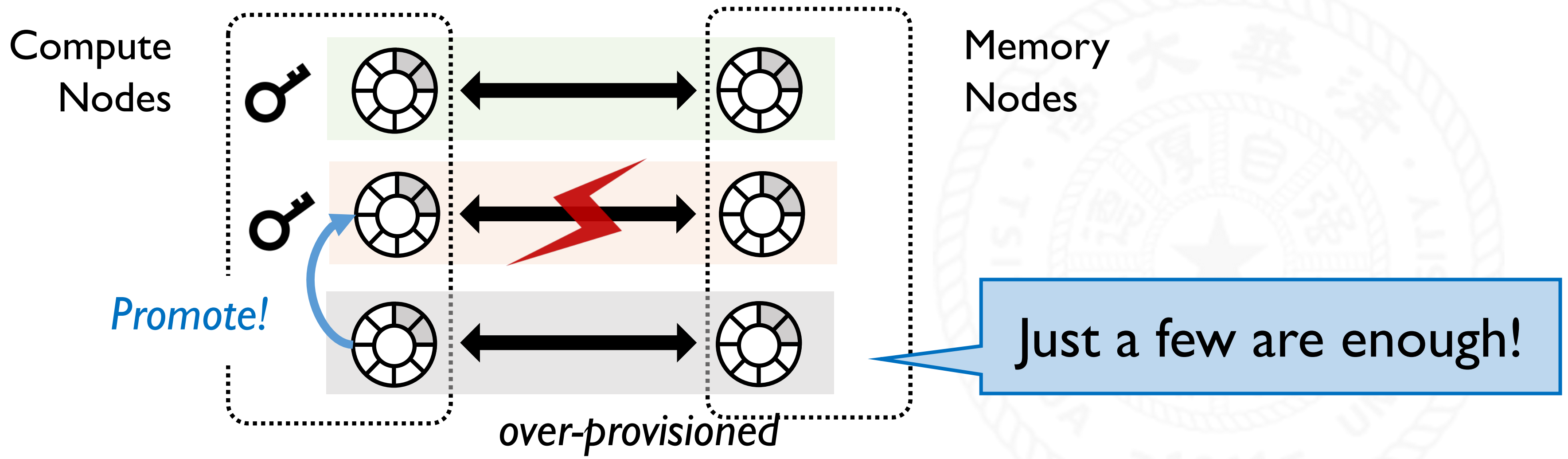
- On QP failures: promote a healthy QP as substitution



Technique (3) – Over-Provisioning for QP failures

Over-provision QPs to hide interruption from QP failures

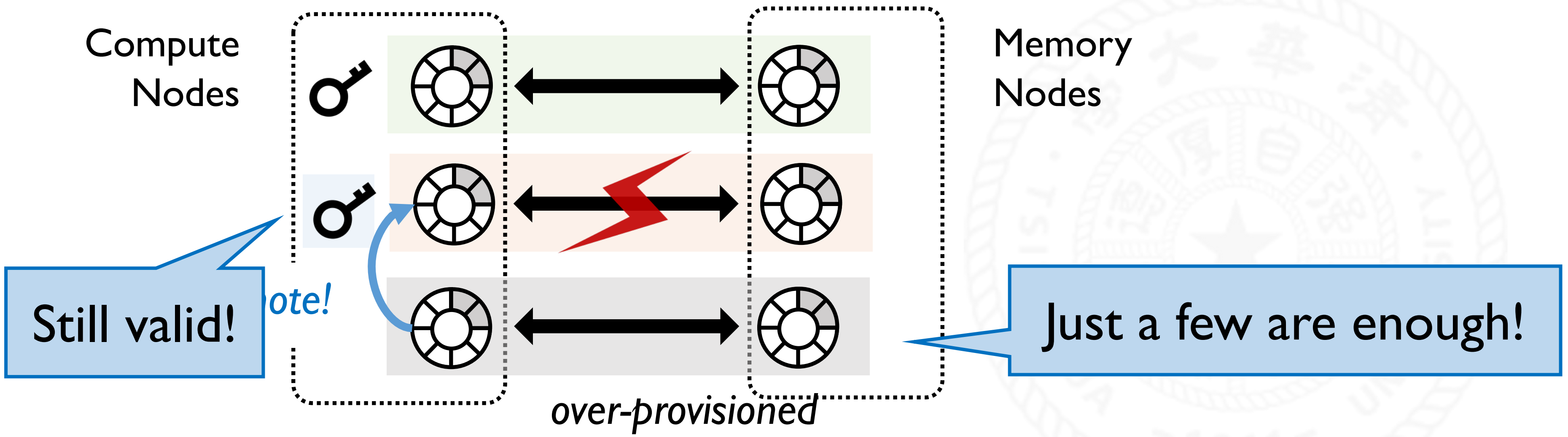
- On QP failures: promote a healthy QP as substitution



Technique (3) – Over-Provisioning for QP failures

Over-provision QPs to hide interruption from QP failures

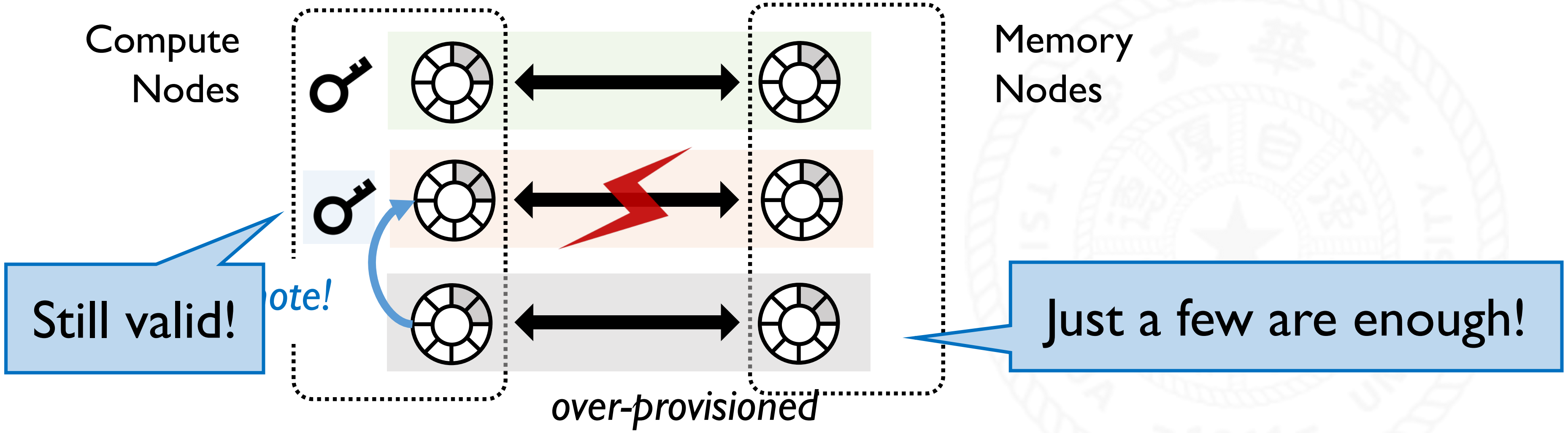
- On QP failures: promote a healthy QP as substitution
- Enabler: MWs can remain valid across QPs => previous permission still works



Technique (3) – Over-Provisioning for QP failures

Over-provision QPs to hide interruption from QP failures

- ❖ On QP failures: promote a healthy QP as substitution
- ❖ Enabler: MWs can remain valid across QPs => previous permission still works
- ❖ Result: low downtime under QP failures



Over-provisioning improves robustness

Outline

- Background & Motivation
- Patronus – High-Performance Protective Remote Memory
- Results
- Conclusion

Experimental Setup

Hardware Platform

CPU	Xeon Gold 6240M CPUs, **32** cores per node
DRAM	186GB DDR4
NIC	Mellanox MT27800 ConnectX-5 Family

Cluster

- 3 Compute node (no cache)
- 1 Memory node (<= 4 CPU cores)

Experimental Setup

Hardware Platform

CPU	Xeon Gold 6240M CPUs, **32** cores per node
DRAM	186GB DDR4
NIC	Mellanox MT27800 ConnectX-5 Family

Cluster

- 3 Compute node (no cache)
- 1 Memory node (<= 4 CPU cores)

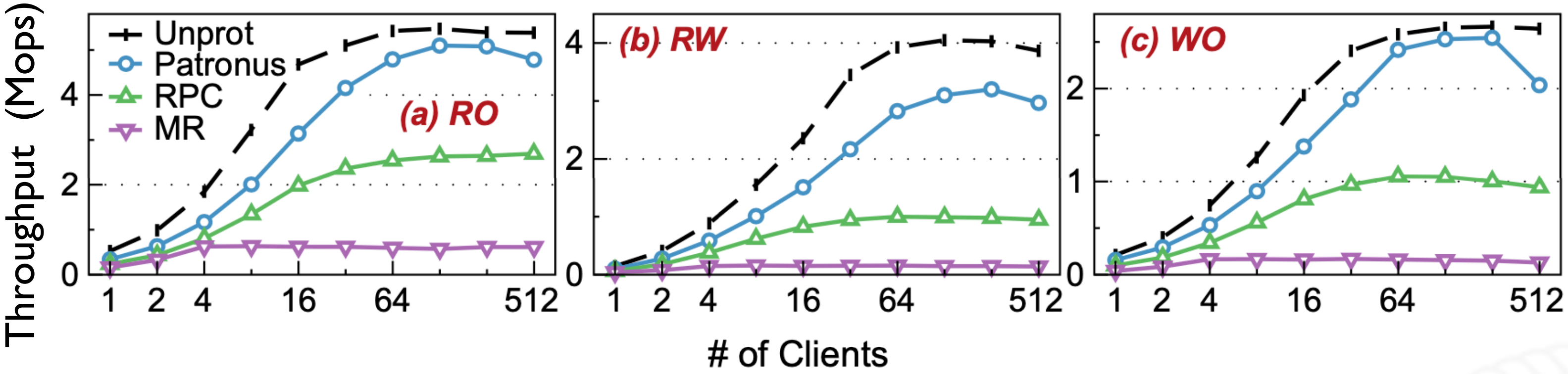
Evaluated Cases

- Common path: 2 one-sided data structures and 2 function-as-a-service workloads.
- Exception path: client failures and QP failures.

Evaluation (1): Overall Performance

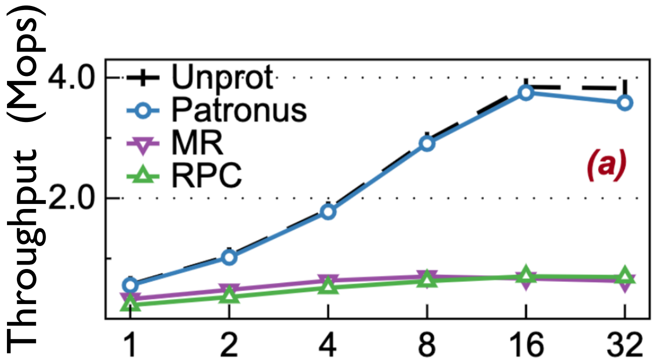
RACE Hashing

Zipfian 0.99
KV = 4KB

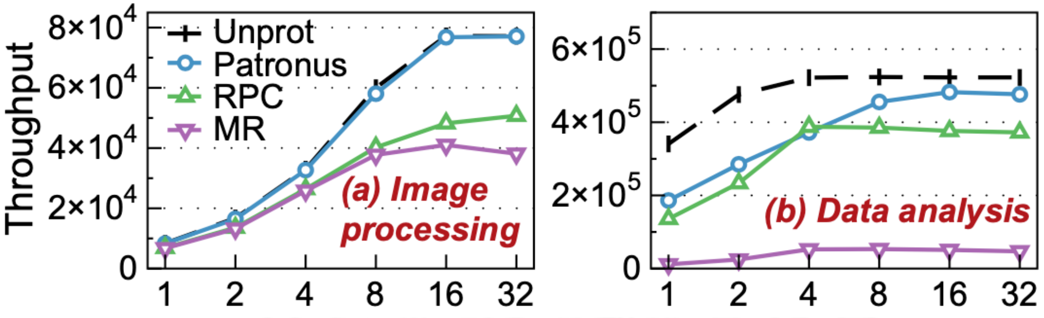


- Vanilla unprotected impl
- Use Patronus for permission
- Use MR for permission
- Use RPC in data path

Concurrent Queue



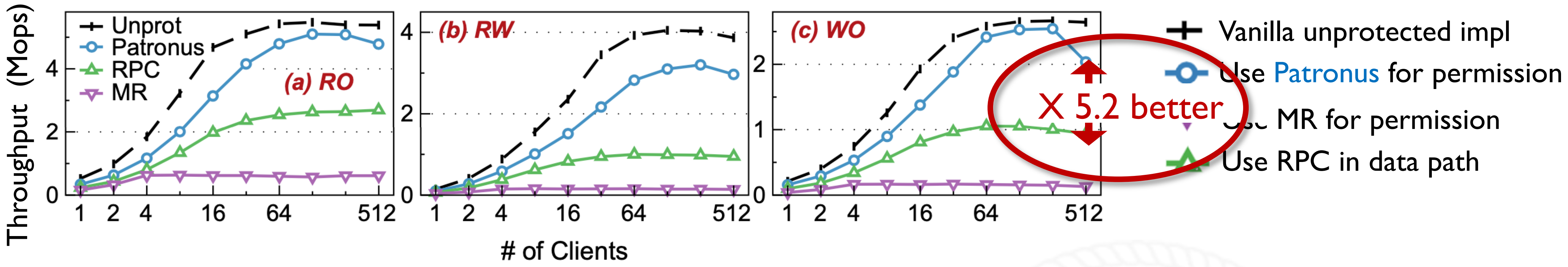
Function as a Service



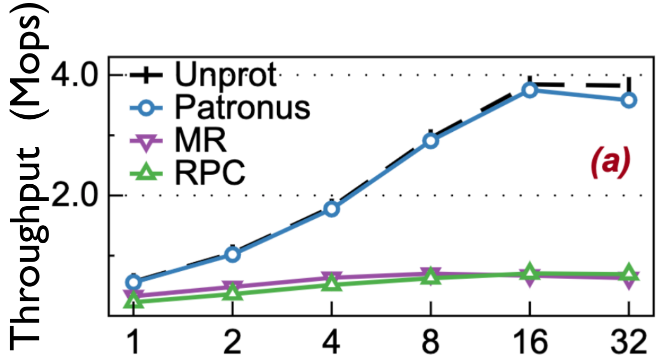
Evaluation (1): Overall Performance

RACE Hashing

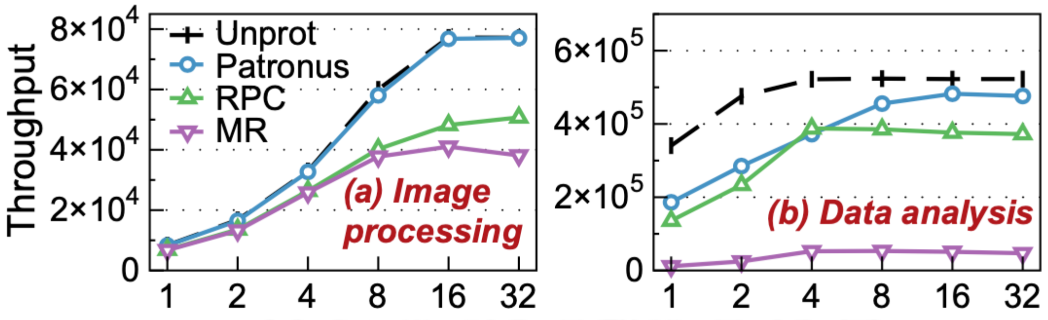
Zipfian 0.99
KV = 4KB



Concurrent Queue



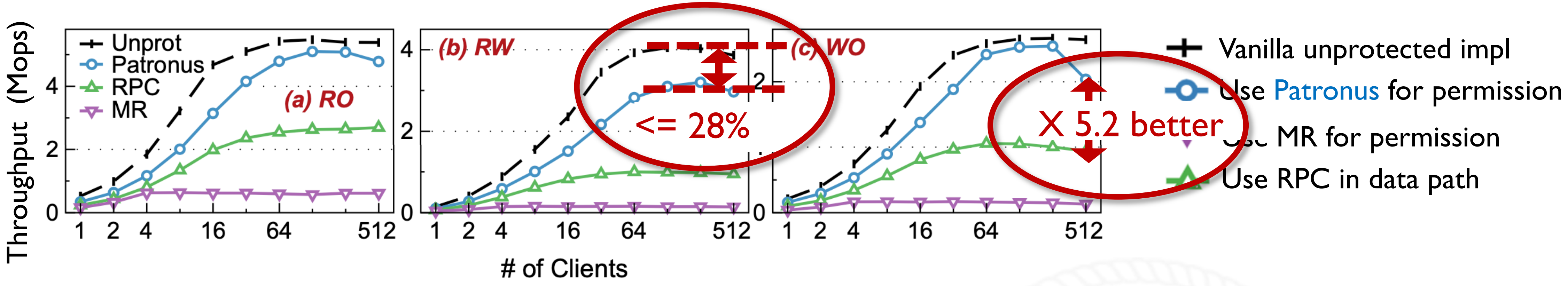
Function as a Service



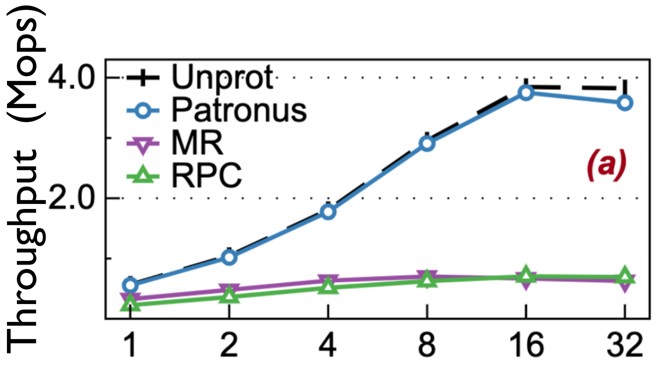
Evaluation (1): Overall Performance

RACE Hashing

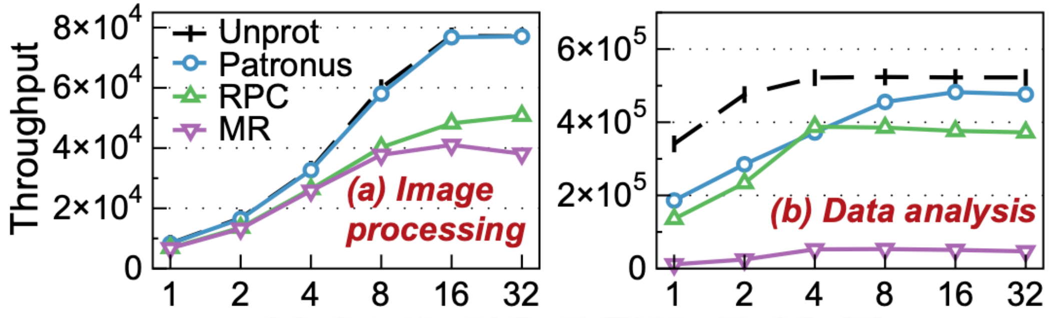
Zipfian 0.99
KV = 4KB



Concurrent Queue



Function as a Service

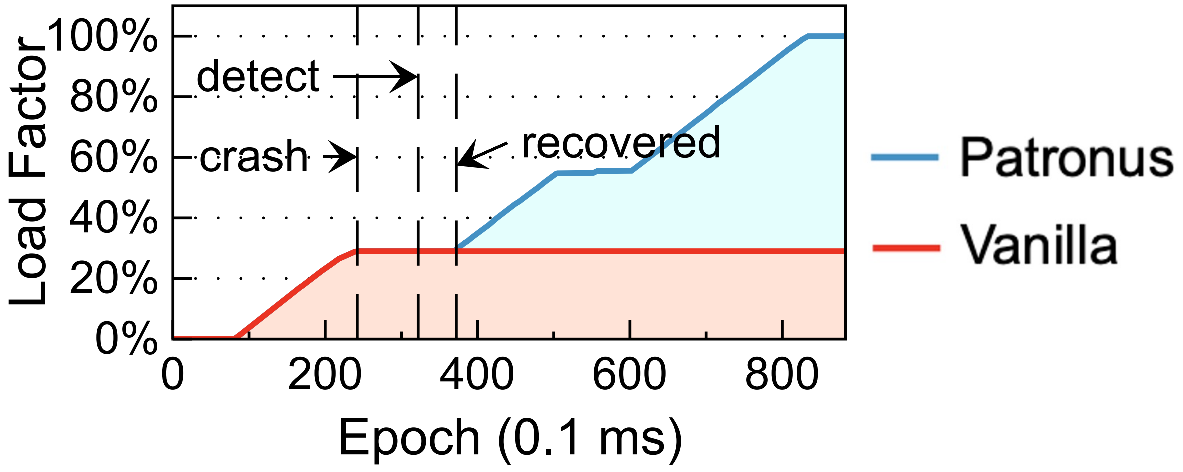


Patronus performs up to X5.2 better than the competitors and has <= 28% overhead than vanilla implementation

Evaluation (2): Failure Handling

Handling client failures

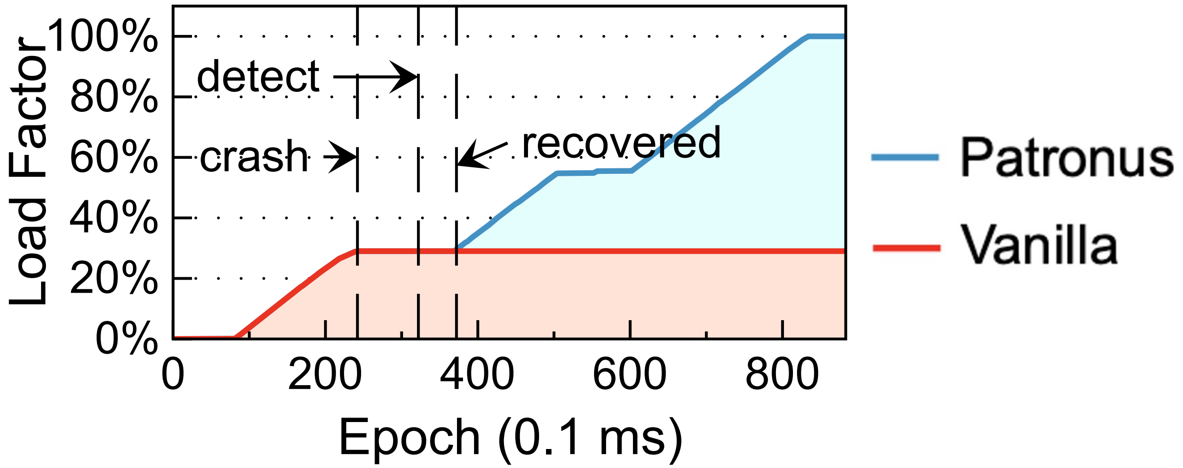
- Vanilla: not handling client failures
- Patronus: resumes progress after 80 epochs.



Evaluation (2): Failure Handling

Handling client failures

- Vanilla: not handling client failures
- Patronus: resumes progress after 80 epochs.



Handling QP failures

- Trigger out-of-bound access manually

Category	Vanilla	Patronus
Promote QP	-	78 us
Notify QP Failure	8 us	-
Recover QP	1004 us	-
Summary	1012 us	78 us (8%)

Patronus is robust to handle client failures and QP failures quickly

Conclusion

- We propose Patronus, a high-performance protective remote memory system for RM protection.
- Three techniques for performance & robustness: MW operations reduction, client-collaborated lease, and QP over-provisioning.
- Patronus takes less than 28% overhead and performs at most x5.2 than the competitors in various real-world workloads.
- More analysis and evaluation results in the paper.

Thanks
Q&A

Patronus – High-Performance and Protective Remote Memory

Bin Yan, Youyou Lu, Qing Wang, Minhui Xie, Jiwu Shu
yanb20@mails.tsinghua.edu.cn