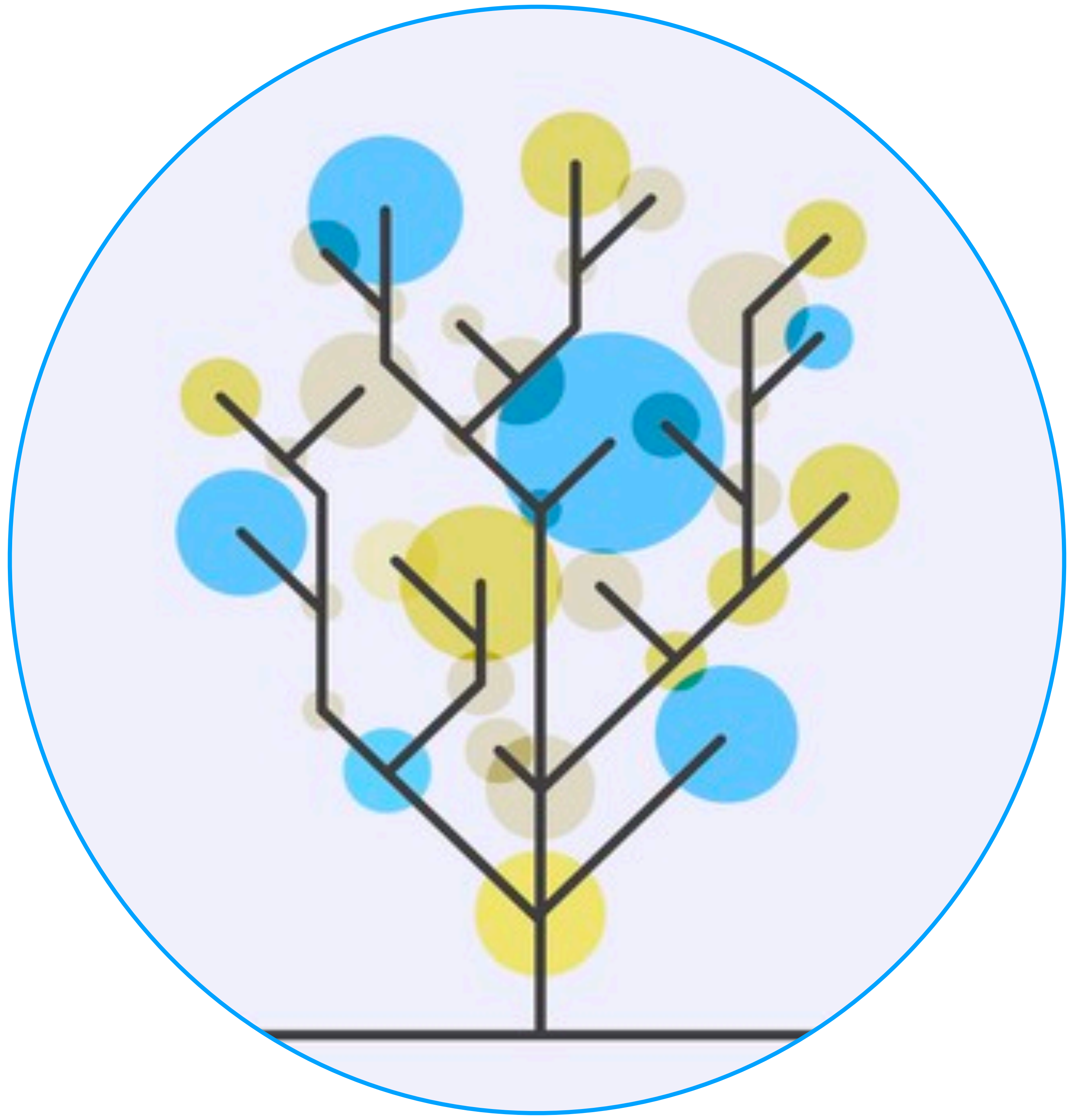
Algorithms for Practical Distributed Agreement

Naama Ben-David
VMware Research

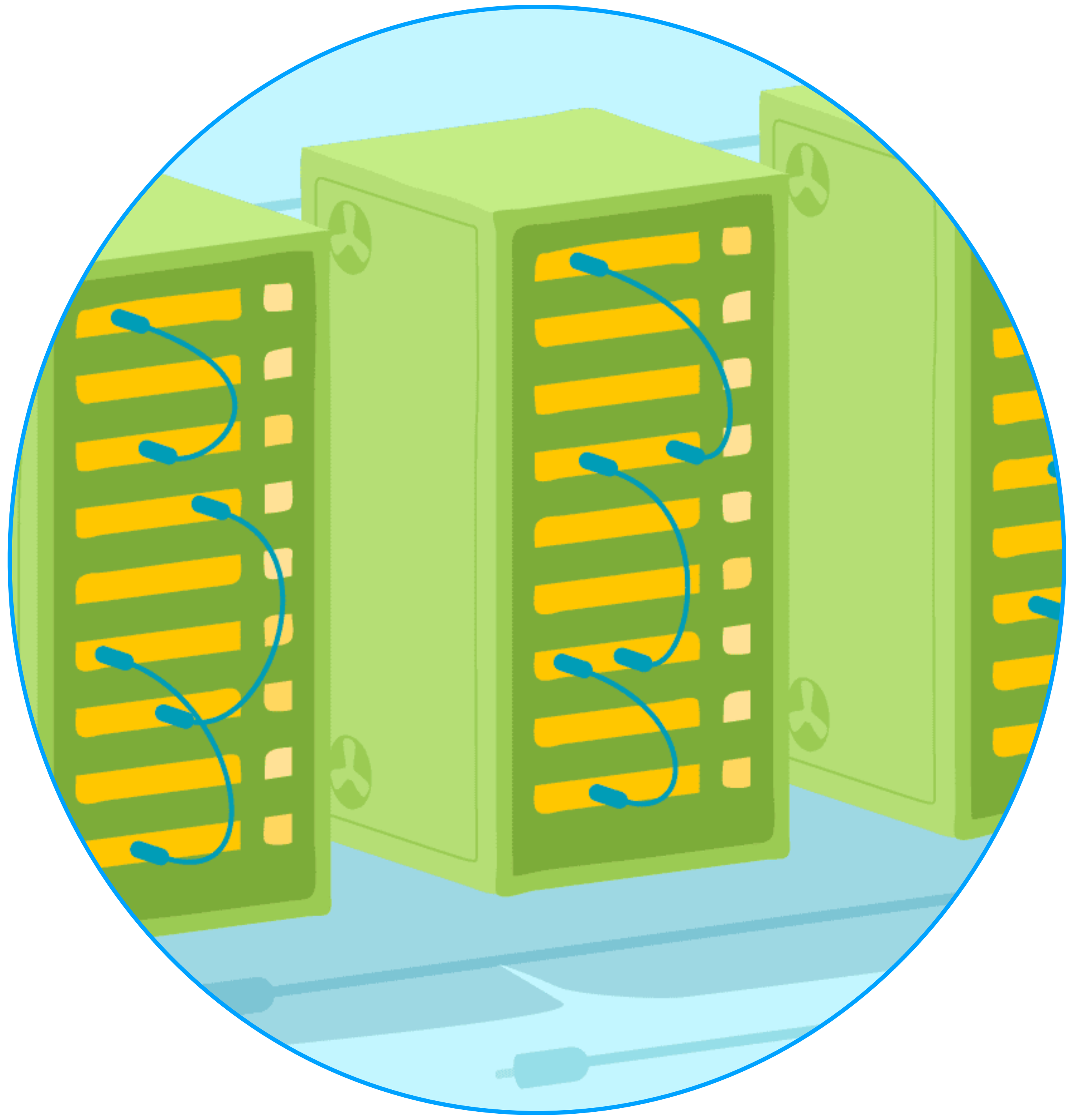
Agreement

Many processes agree on a single value or order of events

Blockchains



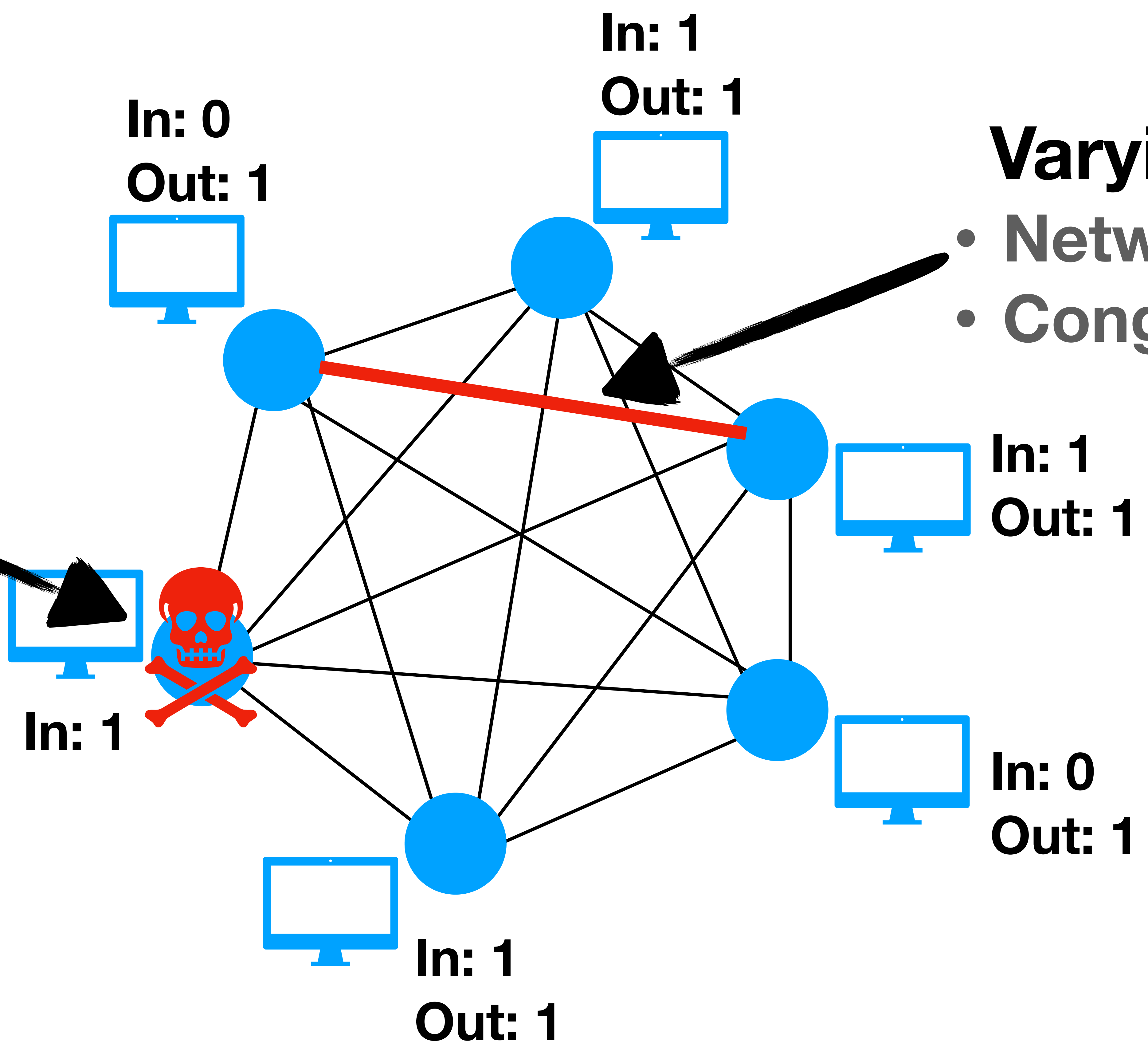
Data structures



Replication

Agreement Challenges

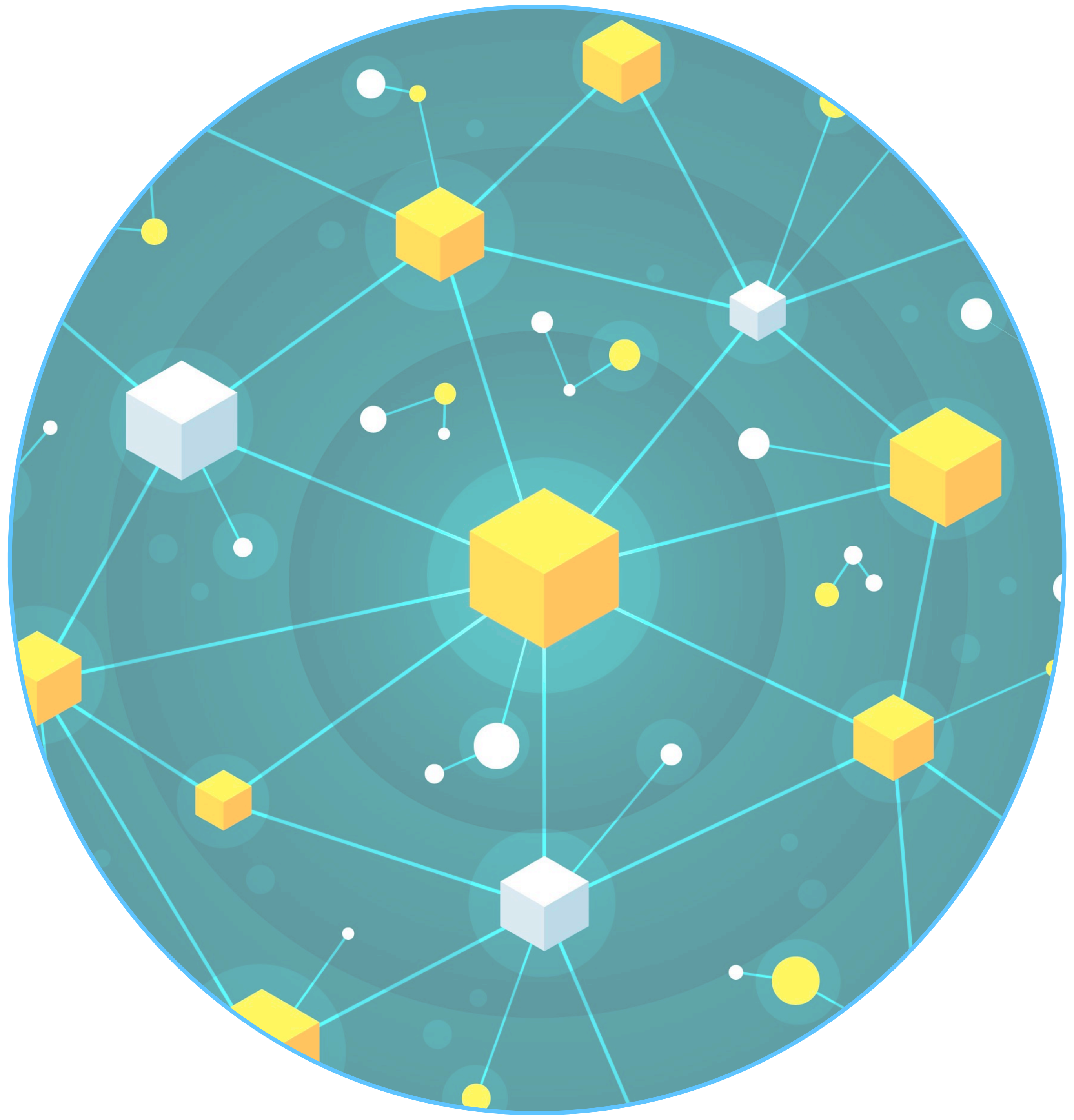
Process failures
- Software bugs
- Overheating
- Hackers



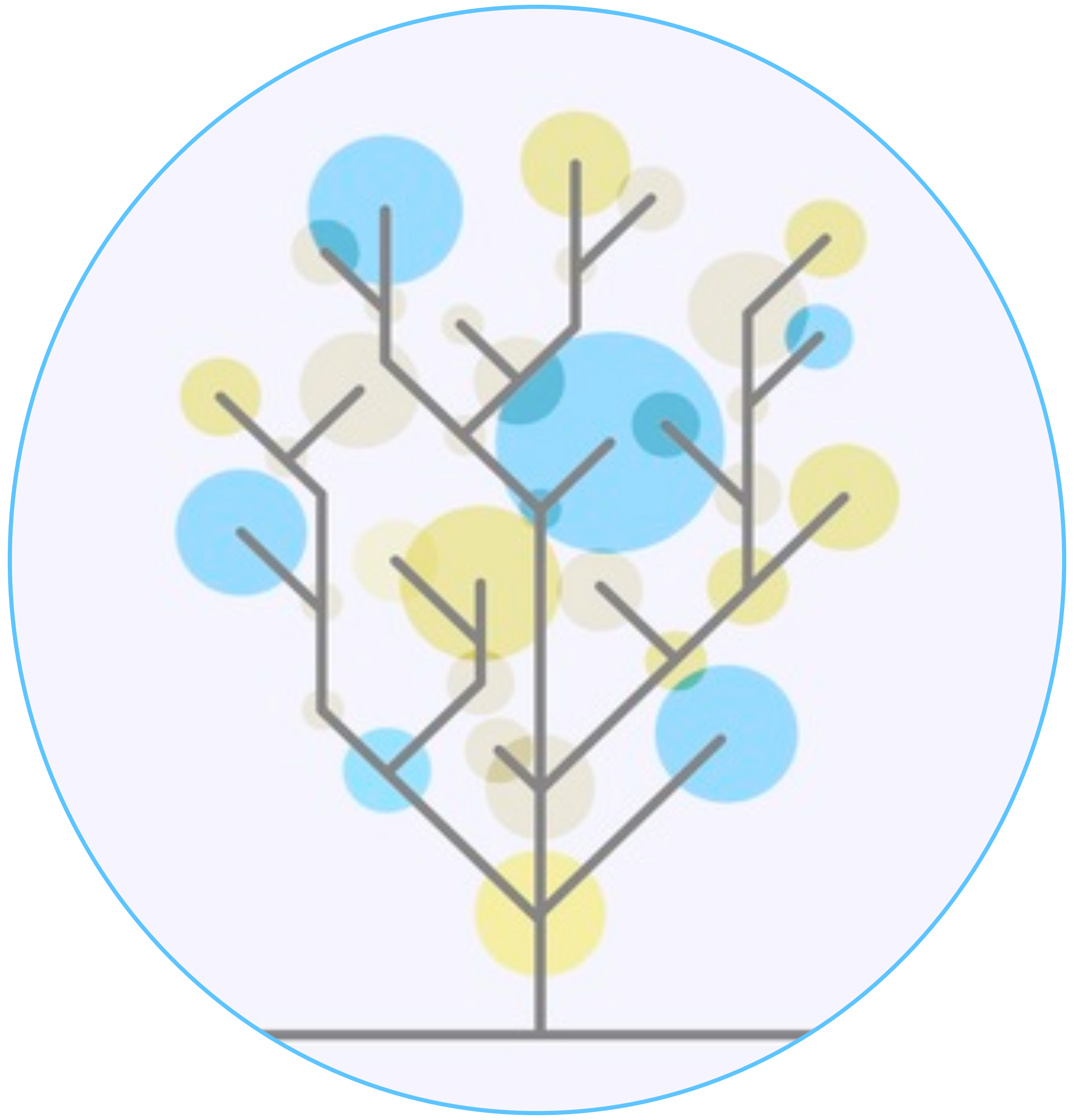
Varying speeds
- Network topology
- Congestion

Agreement

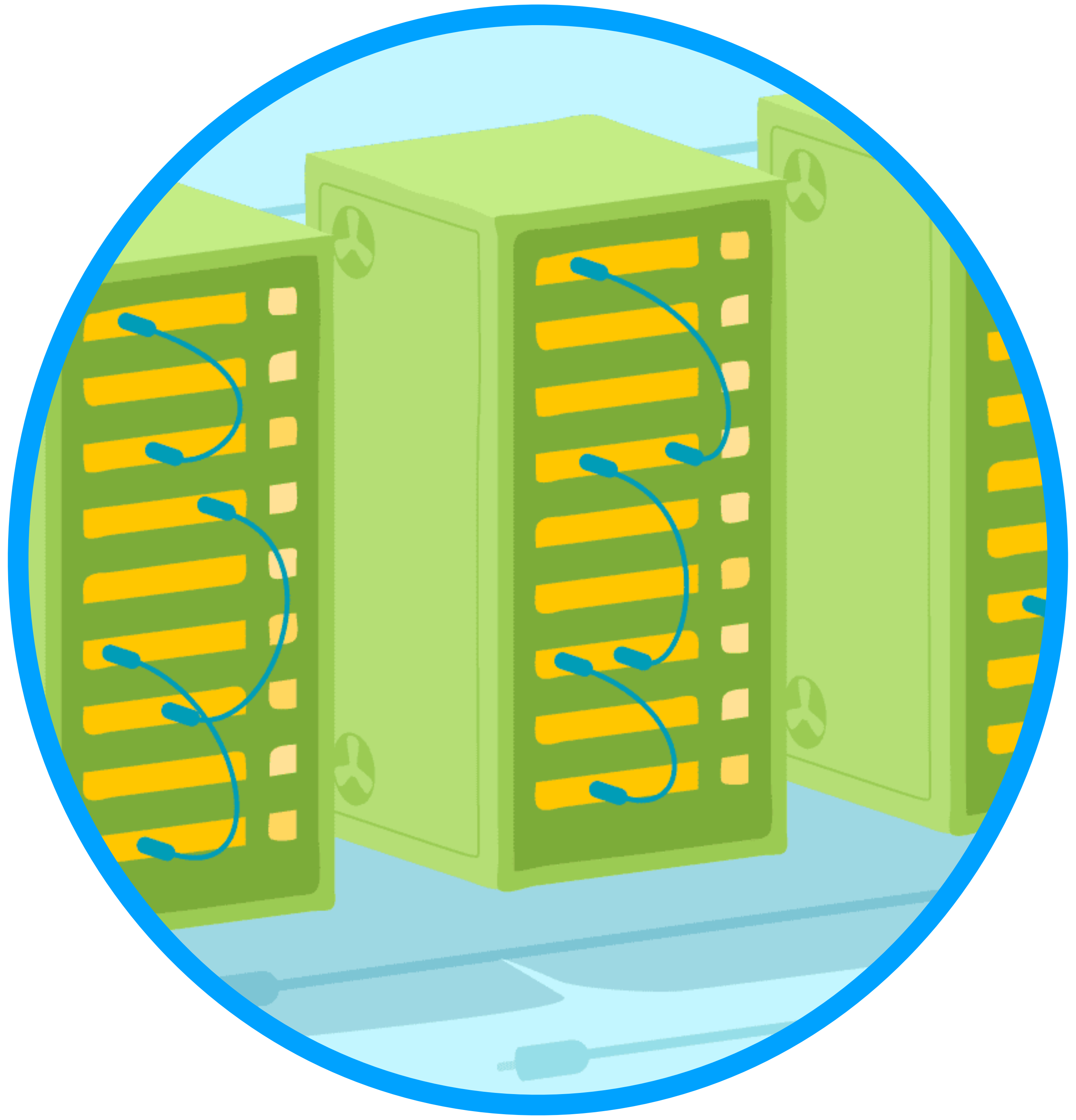
Many processes agree on a single value or order of events



Blockchains

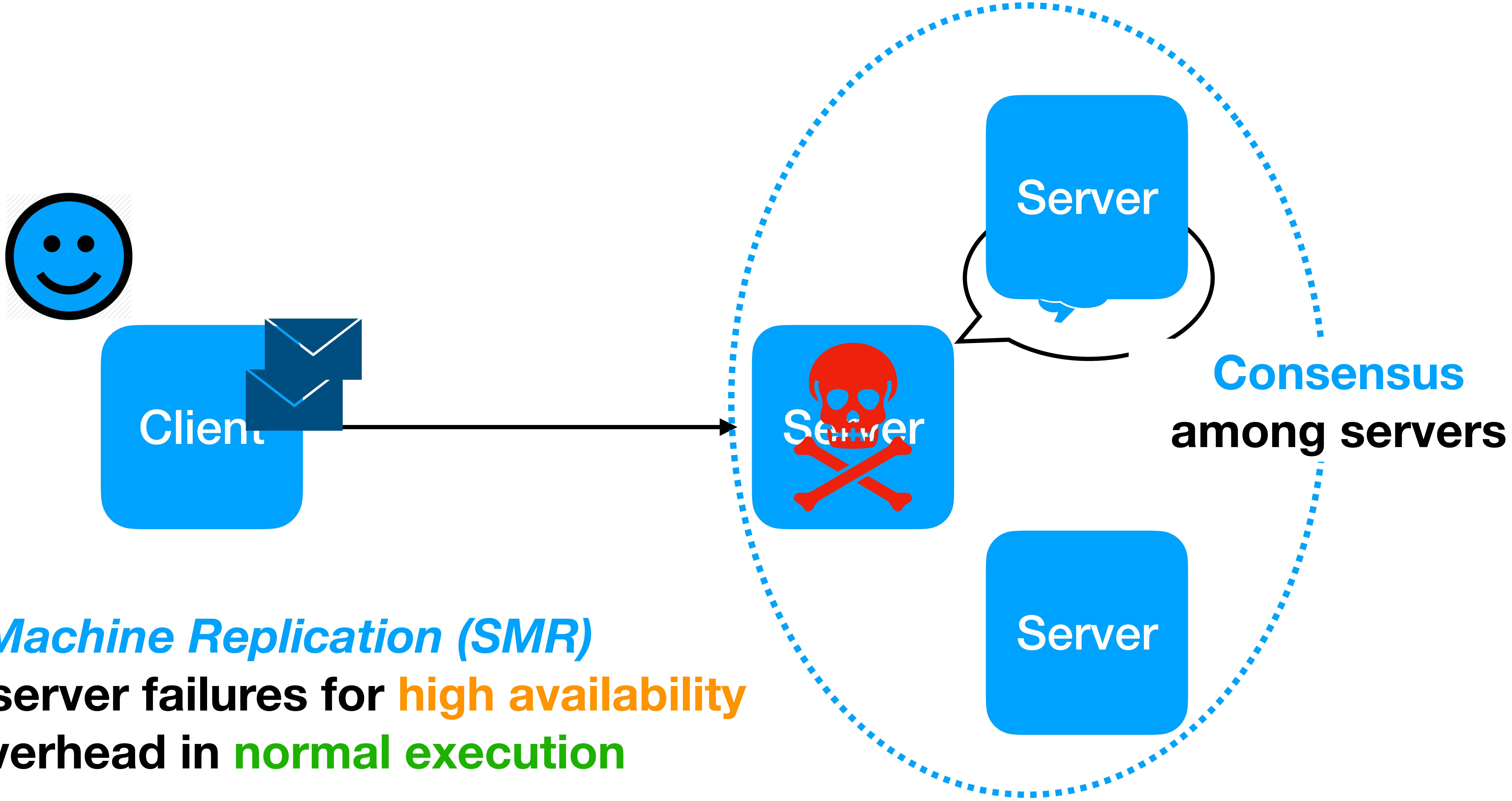


Data structures



Replication

Consensus for Availability



State Machine Replication (SMR)

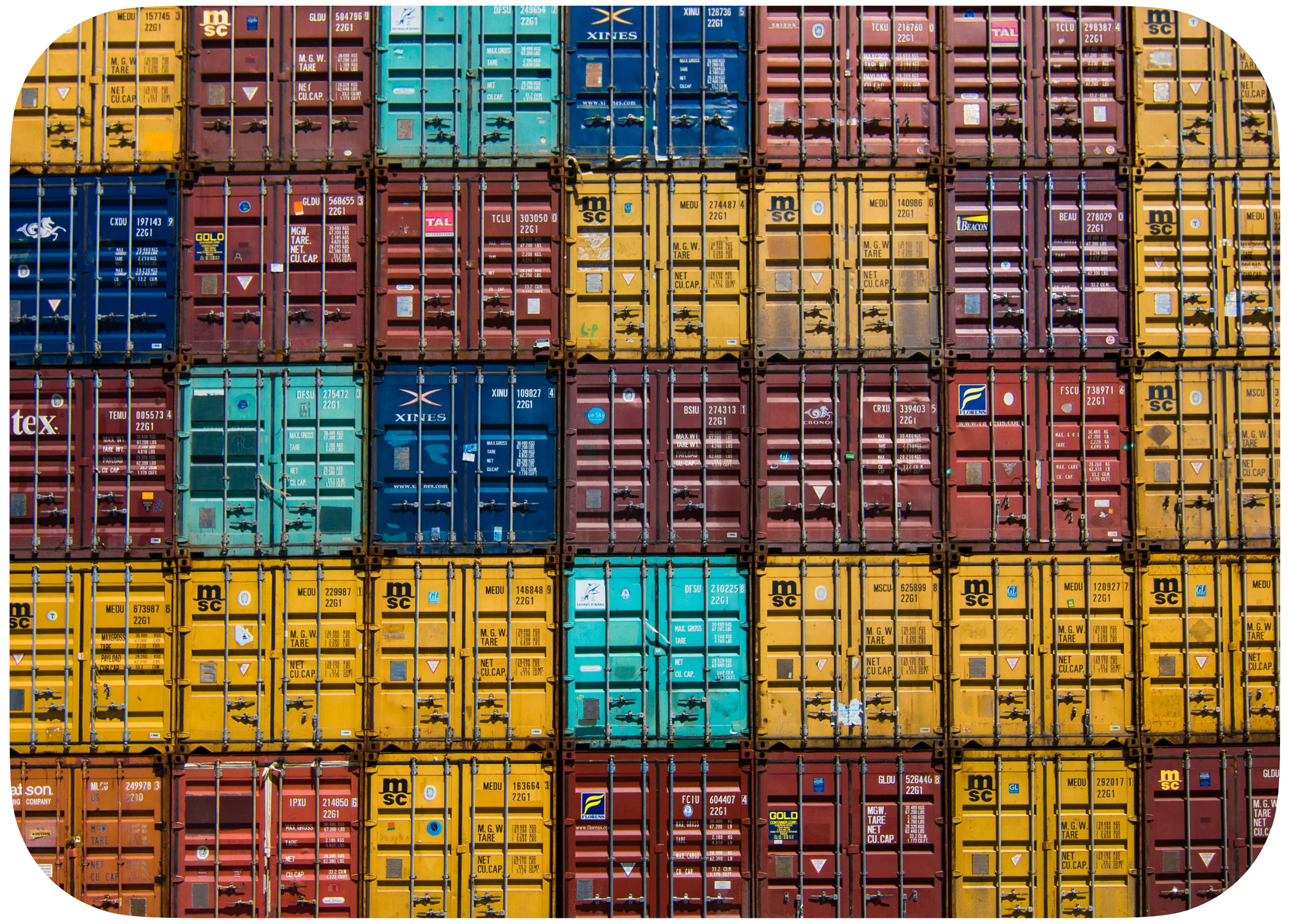
Goal: Masks server failures for high availability

Cost: Overhead in normal execution

Faster and faster applications

Finance
(e.g., high-frequency trading)

Embedded systems
(e.g., industrial robots)



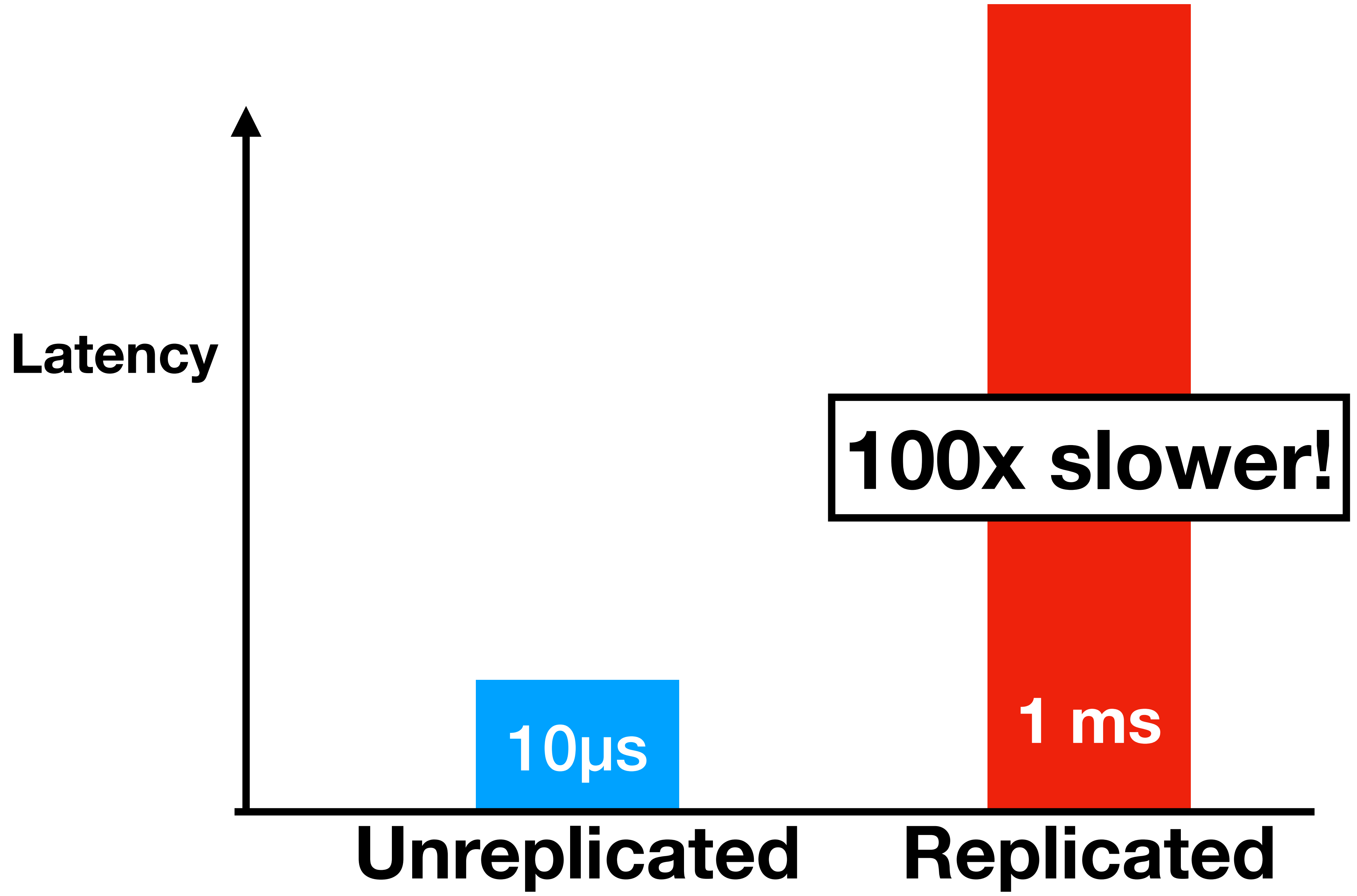
Microservices
(e.g., key-value stores)

Requests expected to be processed within ~10μs

Availability is critical

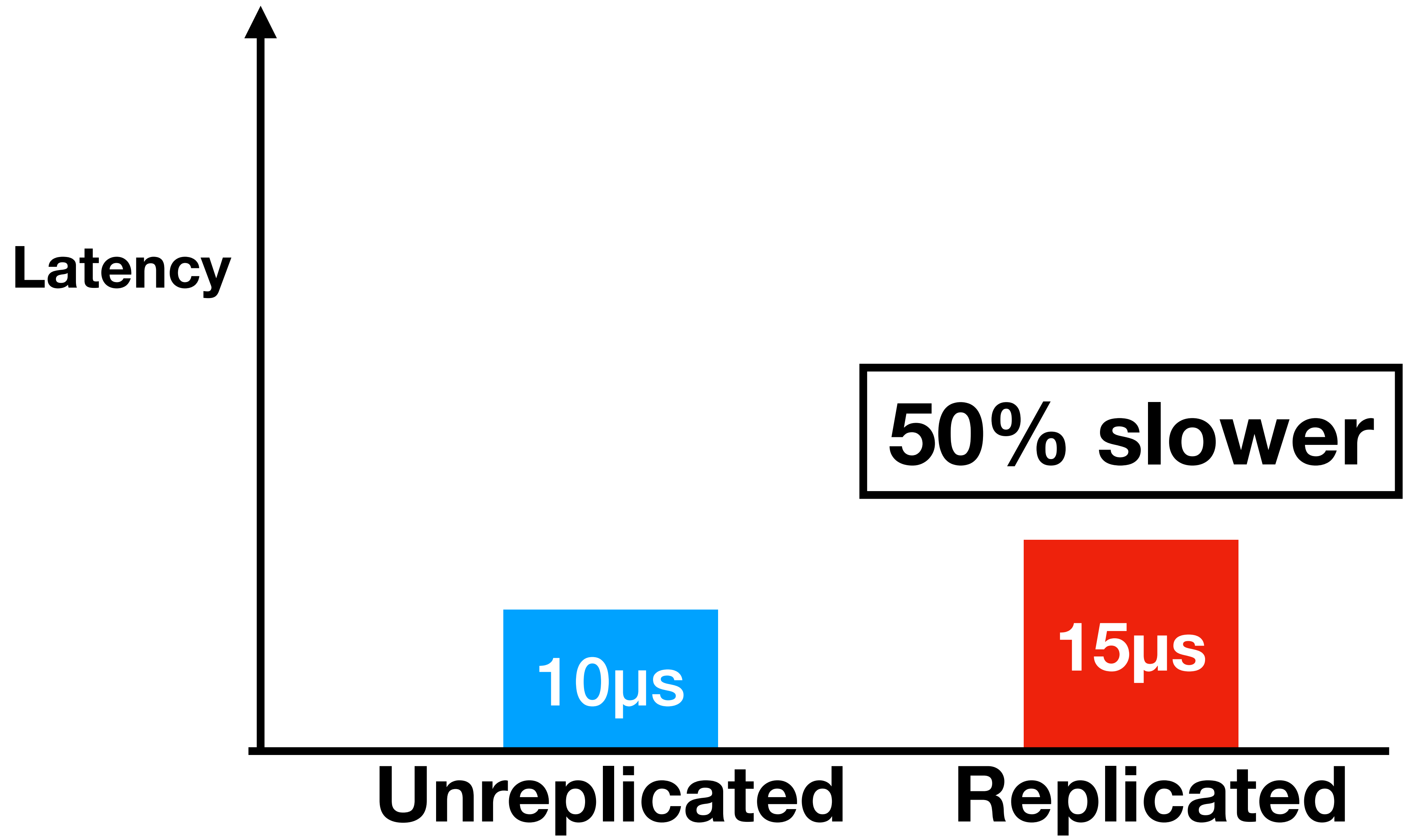
How Efficiently Can We Replicate?

Classic replication (e.g., zookeeper)
~1ms overhead, **~200ms recovery**



Zookeeper: [HuntKonarJunqueiraReed’10]

Faster solutions (e.g. APUS, DARE)
~5µs overhead, **~30ms recovery**



DARE: [PokeHoefler’15]
APUS: [WangJiangChenYiCui’17]

Common-Case Analysis

Want algorithms to withstand worst-case conditions

But usually, conditions are much better

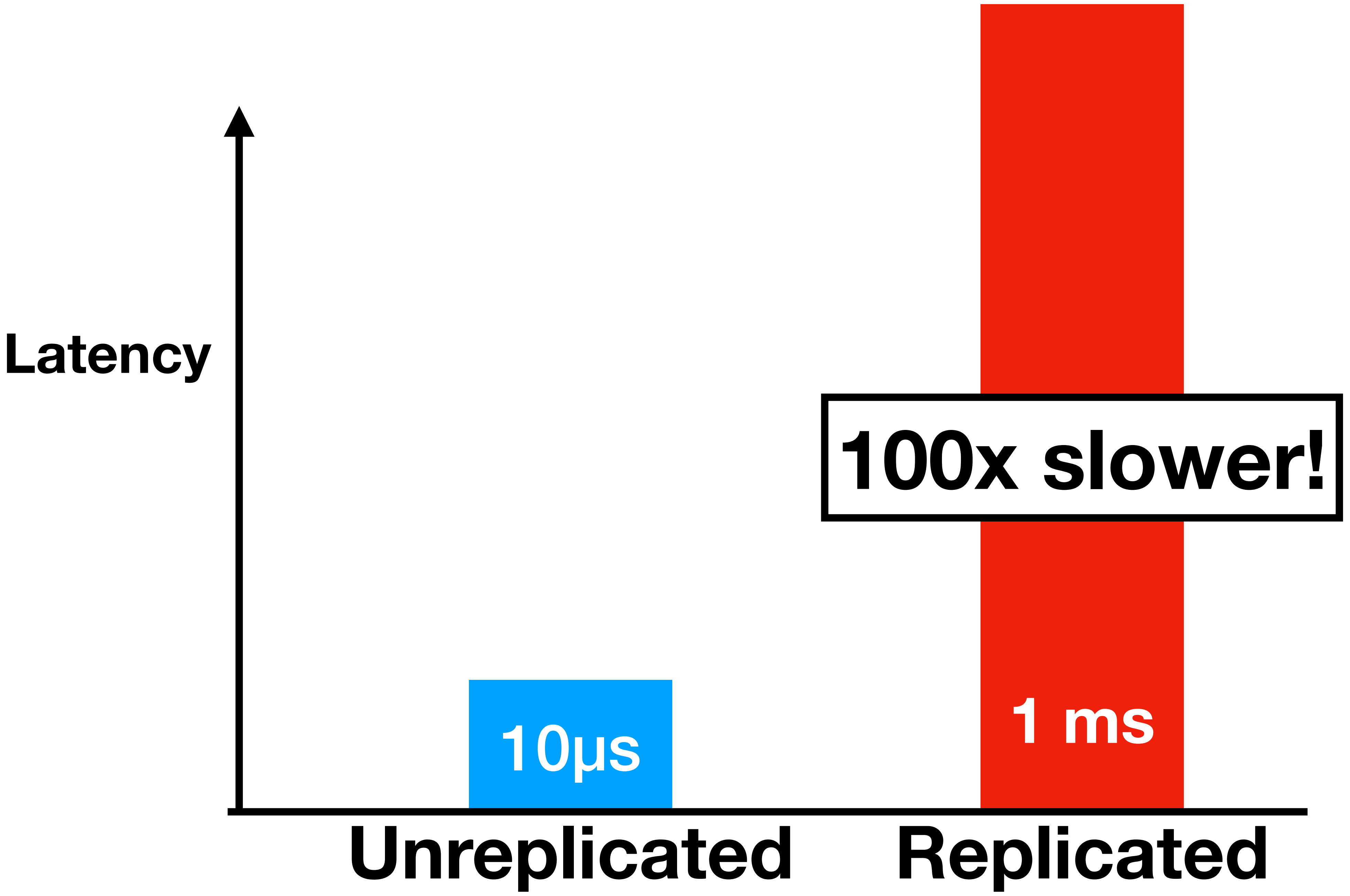
Common-case running time for agreement [KR'01, BGMR'01, Lamport'06, SR'08]**:**

Perfect network conditions:
Synchrony, No Failures

How Efficiently Can We Replicate?

Classic replication (e.g., zookeeper)

~1ms overhead, **~200ms recovery**

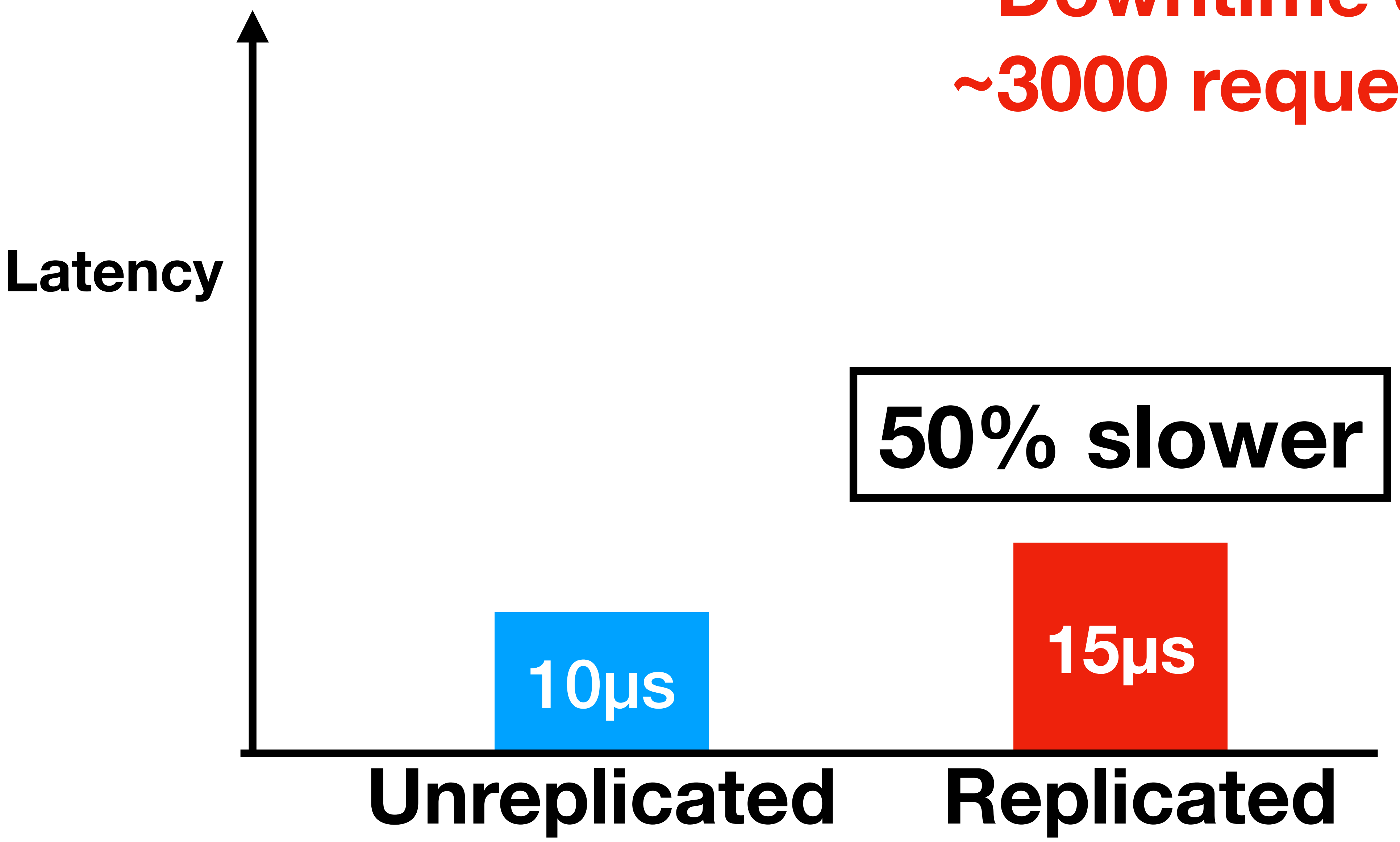


Zookeeper: [HuntKonarJunqueiraReed'10]

Faster solutions (e.g. APUS, DARE)

~5μs overhead, **~30ms recovery**

Downtime of ~3000 requests



DARE: [PokeHoefler'15]
APUS: [WangJiangChenYiCui'17]

Common-Case Analysis: Drawbacks

Making Byzantine Fault Tolerant Systems Tolerate Byzantine Faults

Allen Clement, Edmund Wong, Lorenzo Alvisi, Mike Dahlin
The University of Texas at Austin

Mirco Marchetti

Many systems stall completely when failures happen
Others sacrifice throughput in common case,
for only minor drop from failures

ble of rendering PBFT, Q/U, HQ, and Zyzzyva virtually unusable. In this paper, we (1) demonstrate that exist-

BFT is not an oxymoron—it has led to protocols whose complexity undermines robustness in two ways: (1) the

This Talk

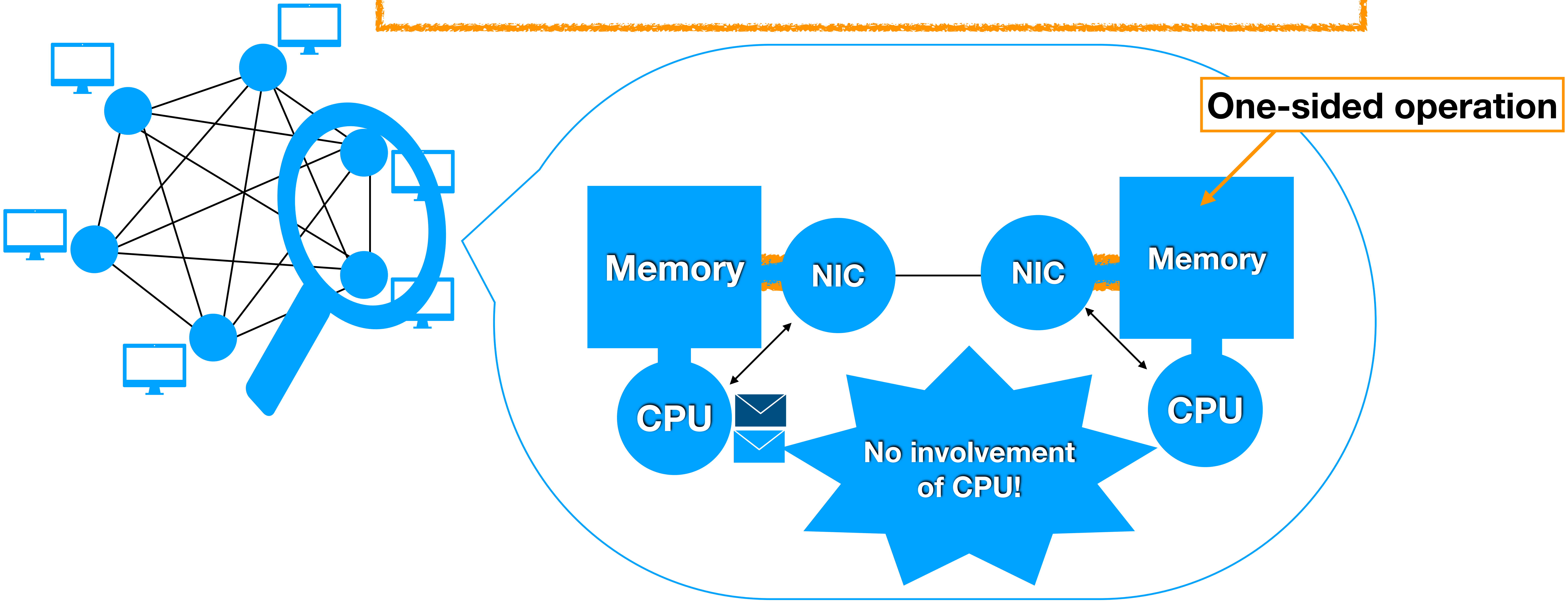
Create algorithms that improve on both best-case and worst-case performance

Roadmap

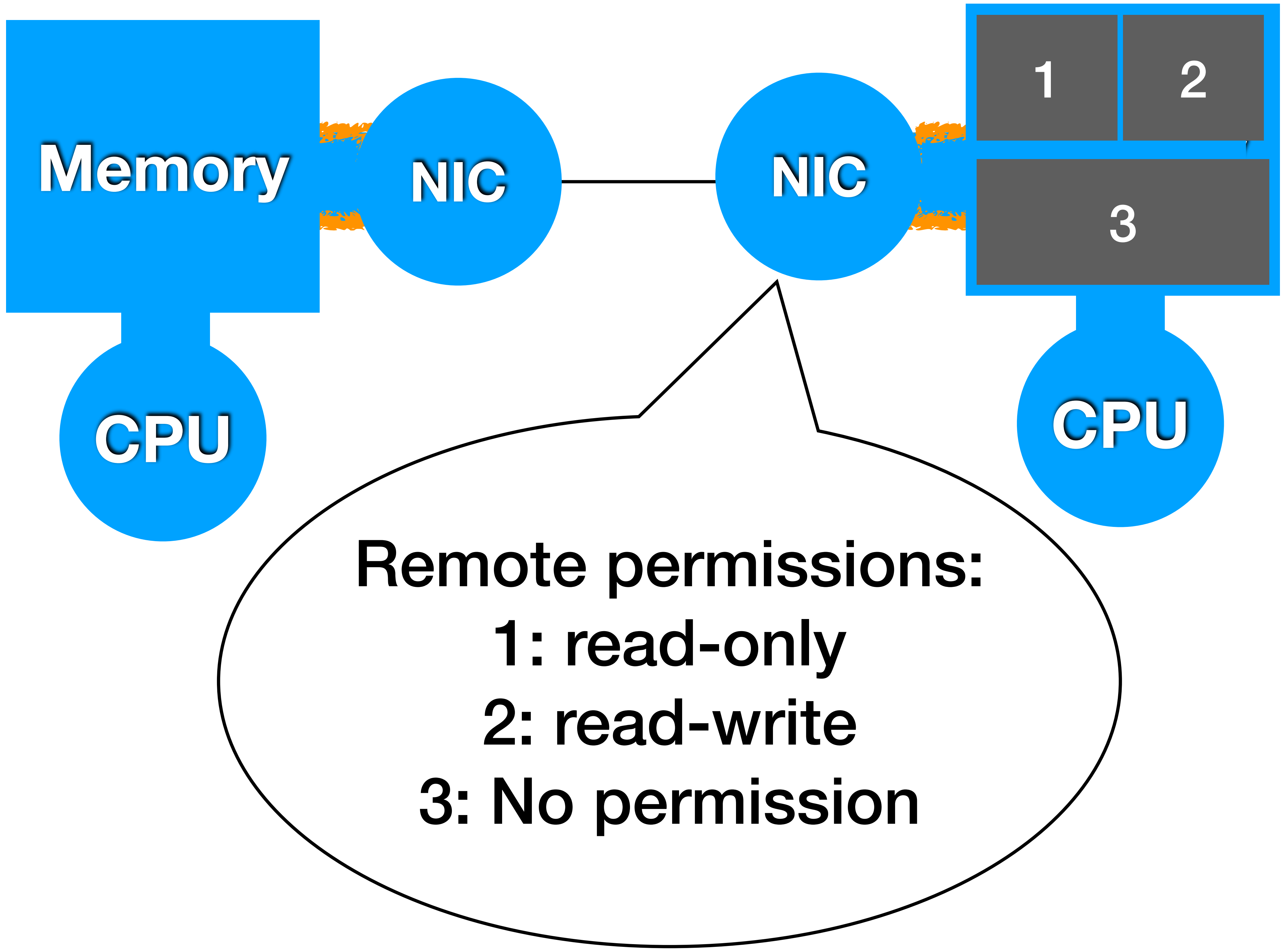
- Case Study: Mu μ
  - Background: SMR based on **RDMA**
  - **Best Case**: **~1.3µs replication** overhead
    ~3x improvement over state of the art
  - **Worst Case**: **<1ms recovery**
    ~12x improvement over state of the art
  - Experimental Evaluation
- Other **best case**/**worst case** improvements

Data Center Technology: RDMA

Remote Direct Memory Access (RDMA)



RDMA: More Details

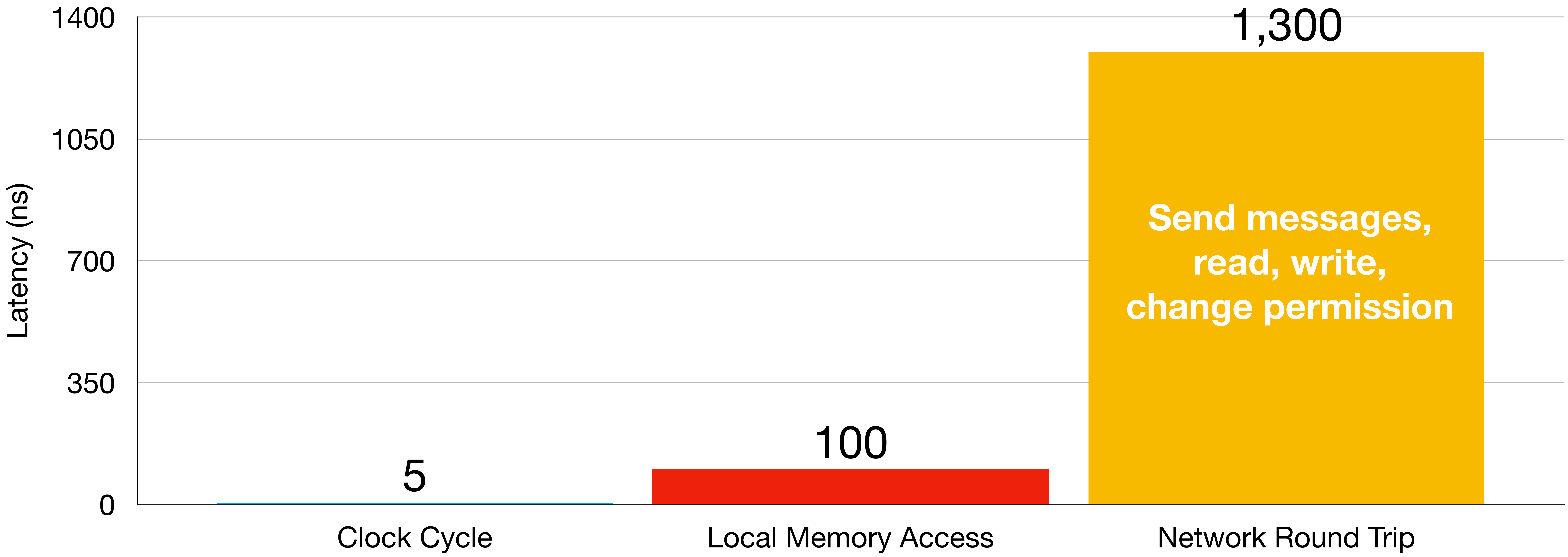


Fast communication:
~1μs latency, ~100Gbps bandwidth

RDMA can specify **access permissions** at a **fine granularity**

These permissions can be **dynamically changed**

Network Communication Cost

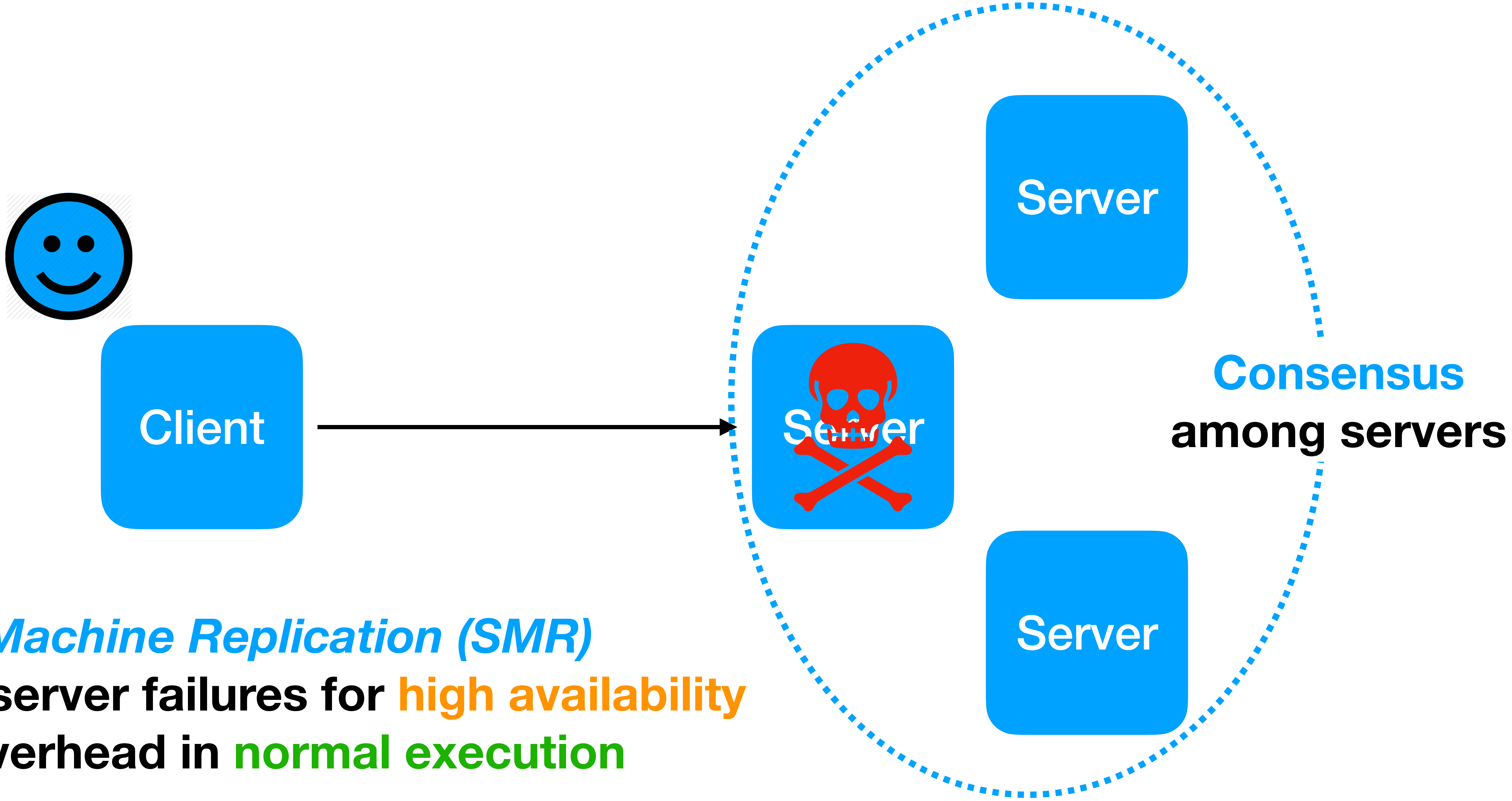


Mu system goals

In terms of round trips

RDMA-based SMR system with
optimal common-case performance and
improved performance under failures

Replication for Availability



State Machine Replication (SMR)

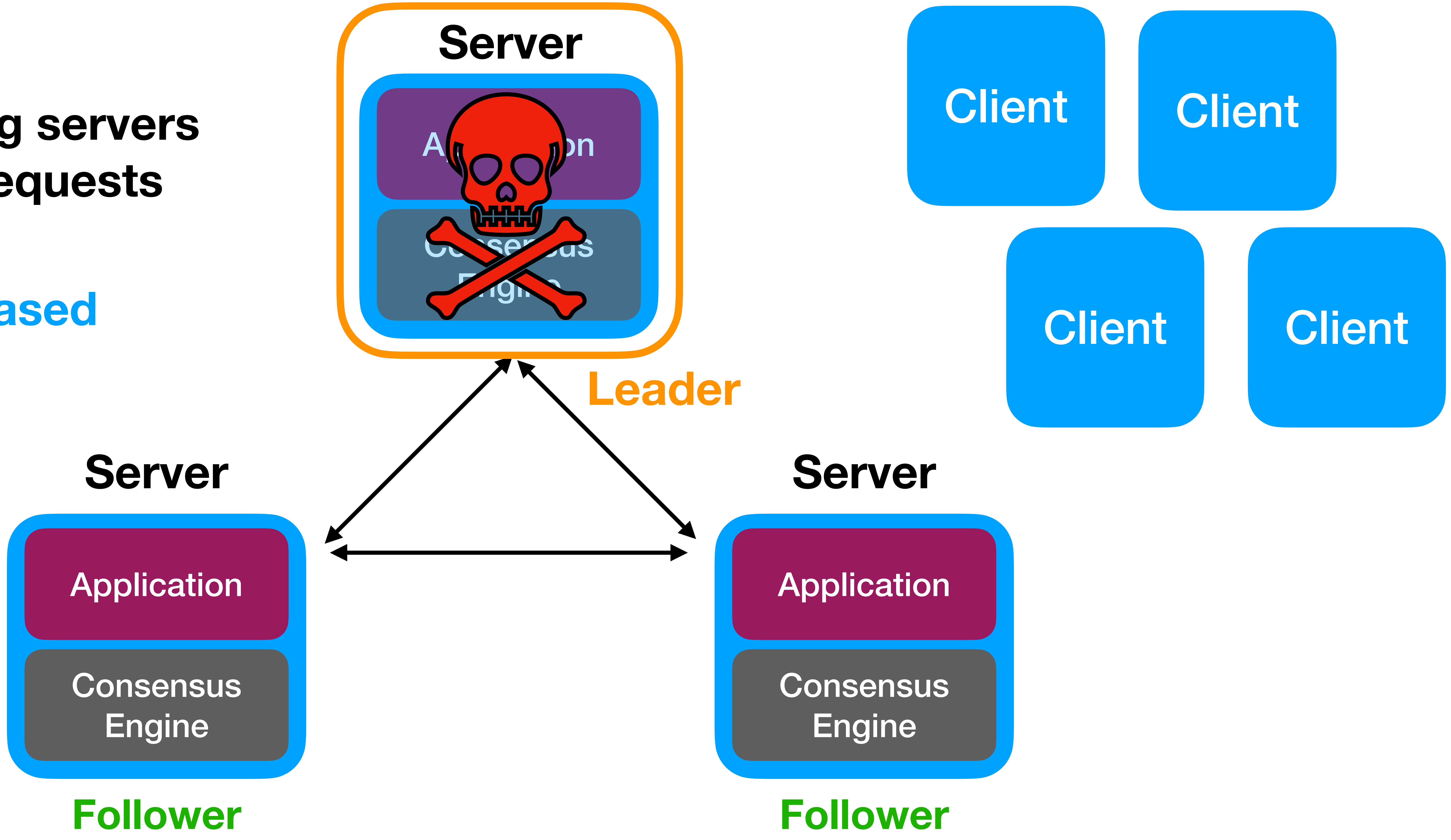
Goal: Masks server failures for high availability

Cost: Overhead in normal execution

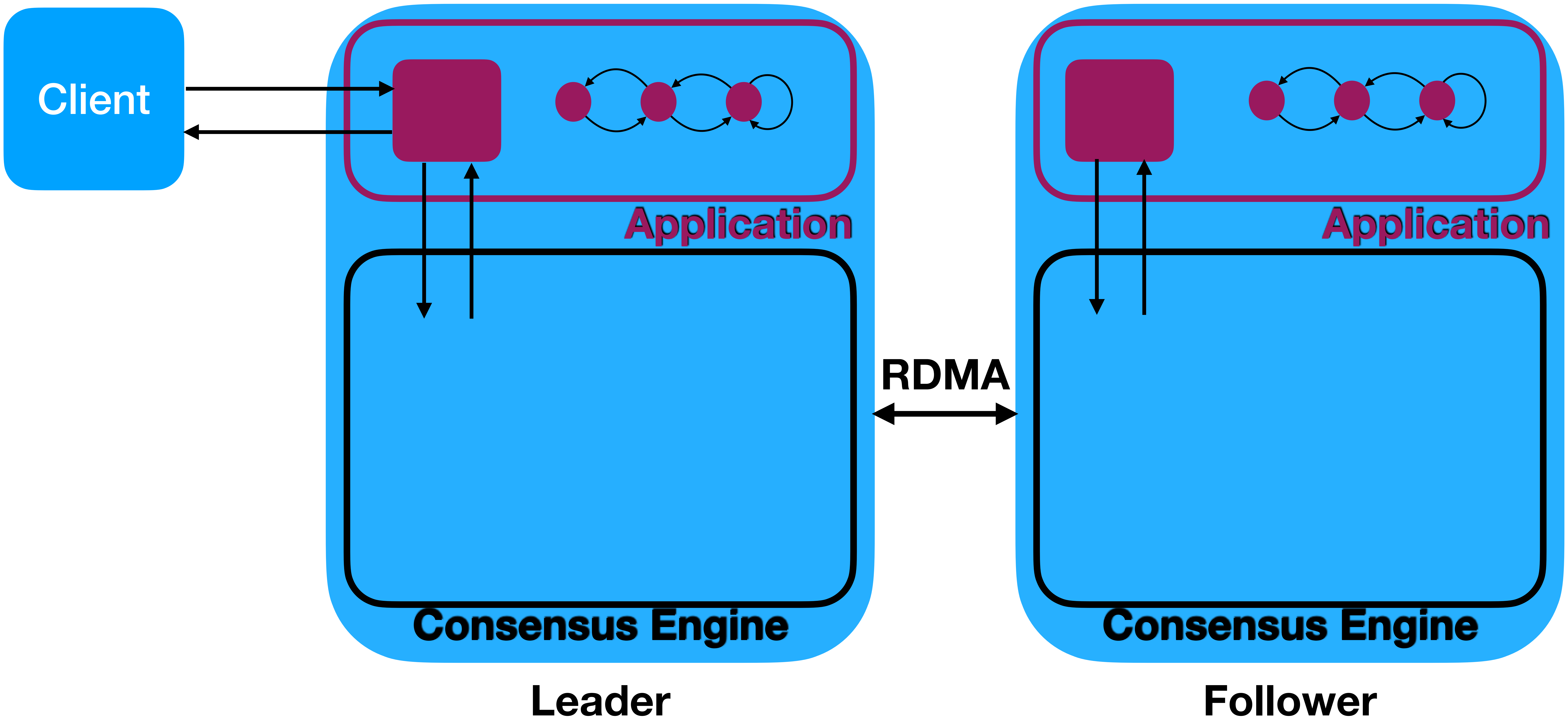
State Machine Replication

Consensus among servers
to order client requests

Often **leader-based**



Basic Mu Architecture



Roadmap

- Case Study: Mu μ
  - Background: SMR based on **RDMA**
  - **Best Case**: **~1.3μs replication** overhead
  - **Worst Case**: **<1ms recovery**
  - Experimental Evaluation
- Other **best case**/**worst case** improvements

Common Case Execution

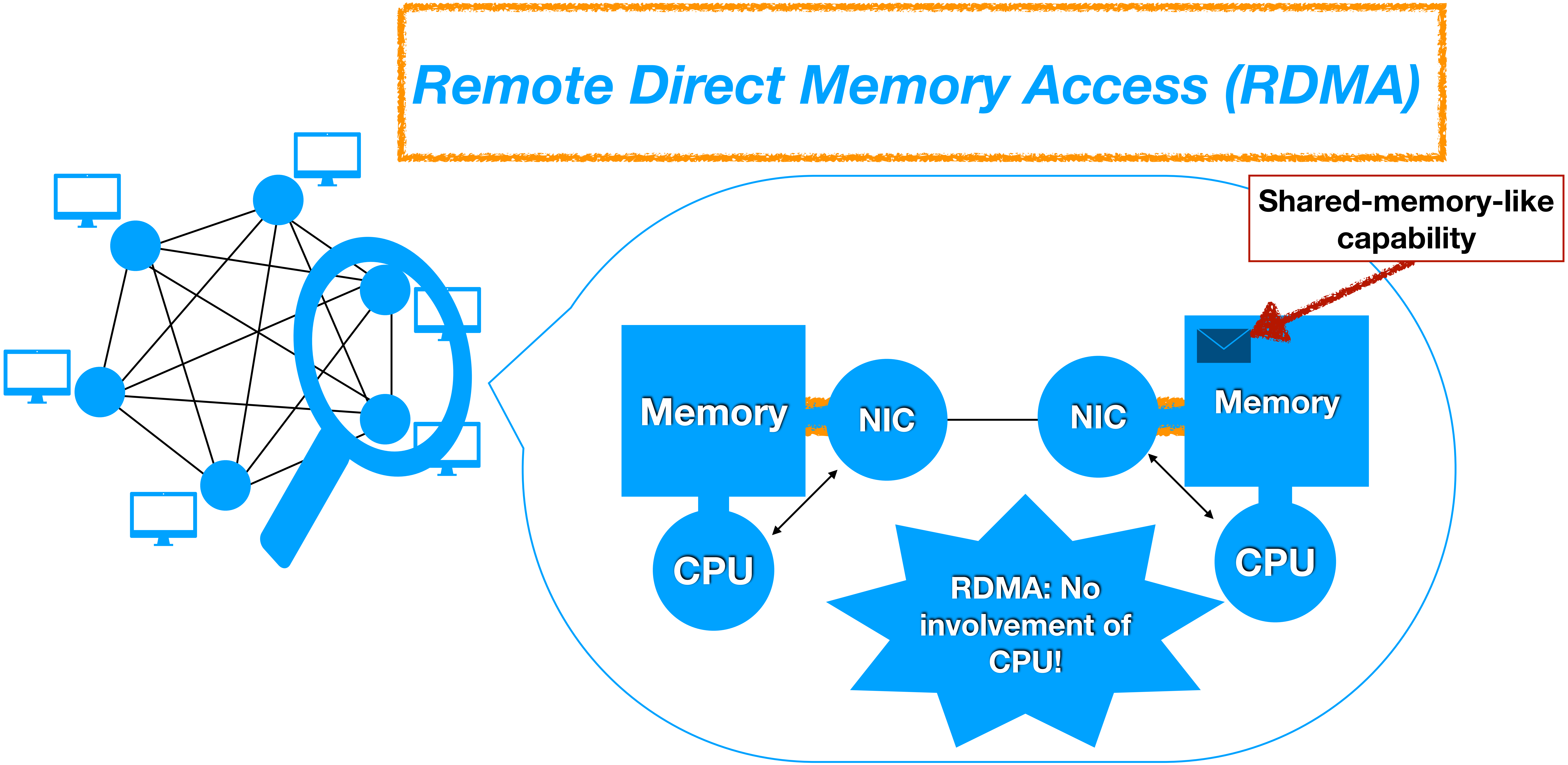
Bypass remote CPU for improved performance

Replicate requests in **a single one-sided RDMA round trip** when there is **synchrony and no failures**

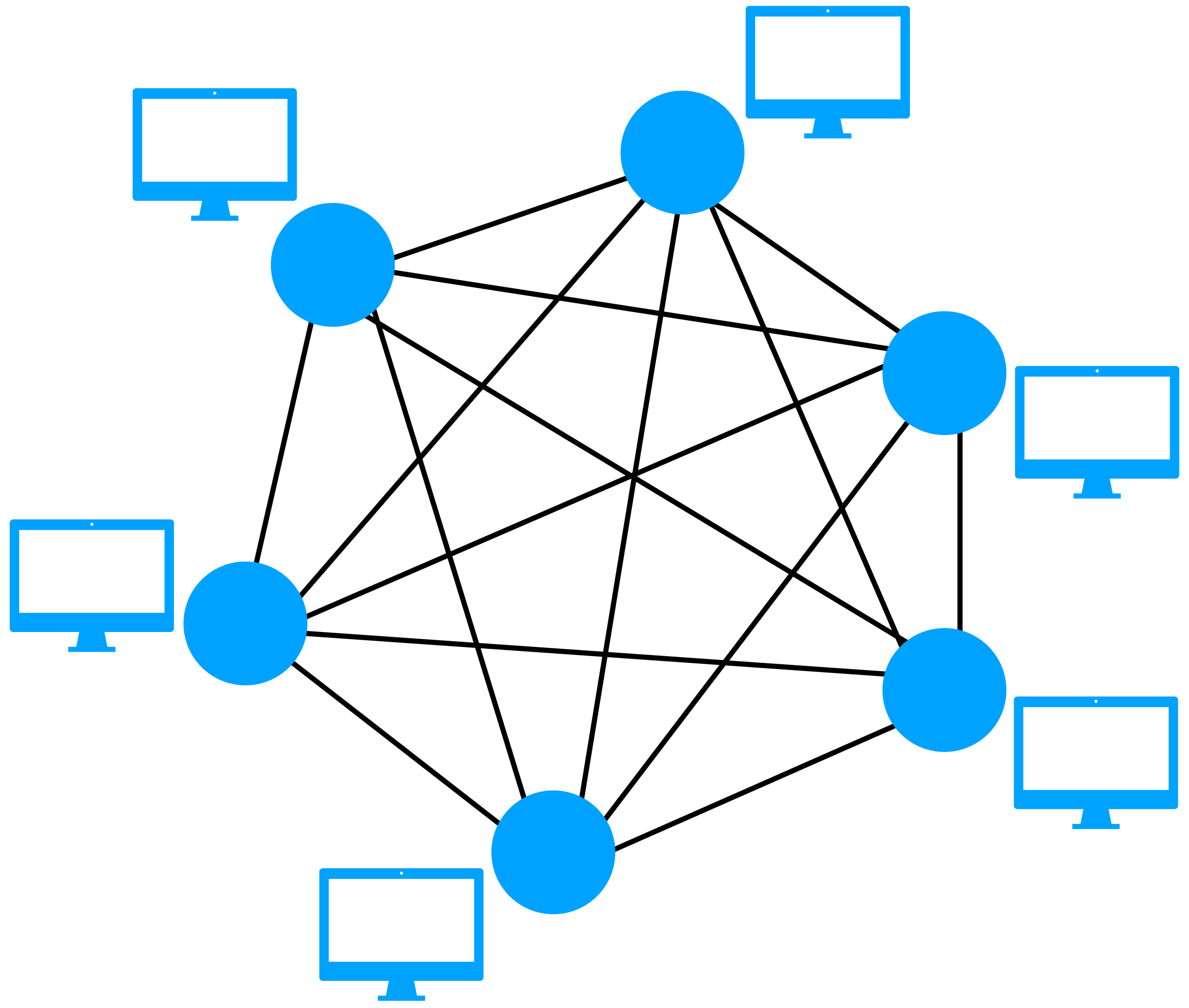
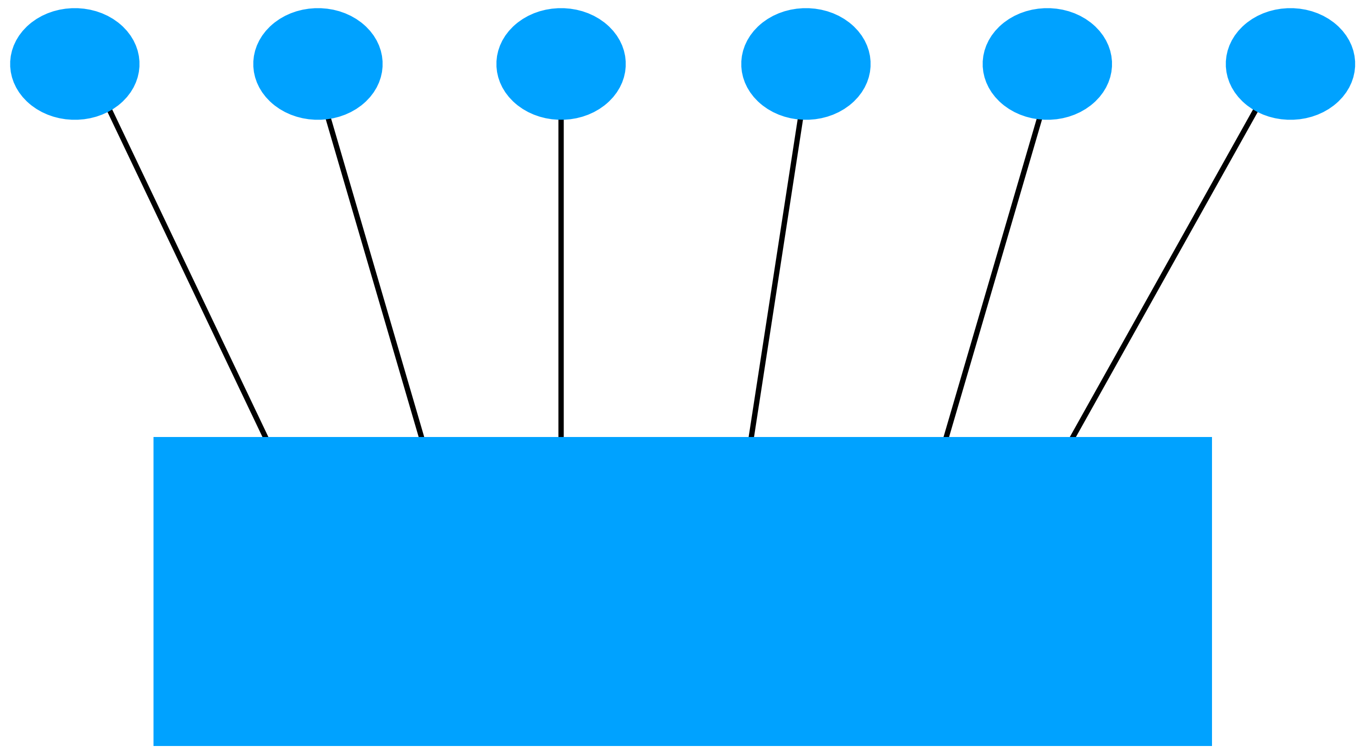
~1.3μs
Where our latency comes from

Theoretical model to reason about algorithmic possibilities

Data Center Technology: RDMA

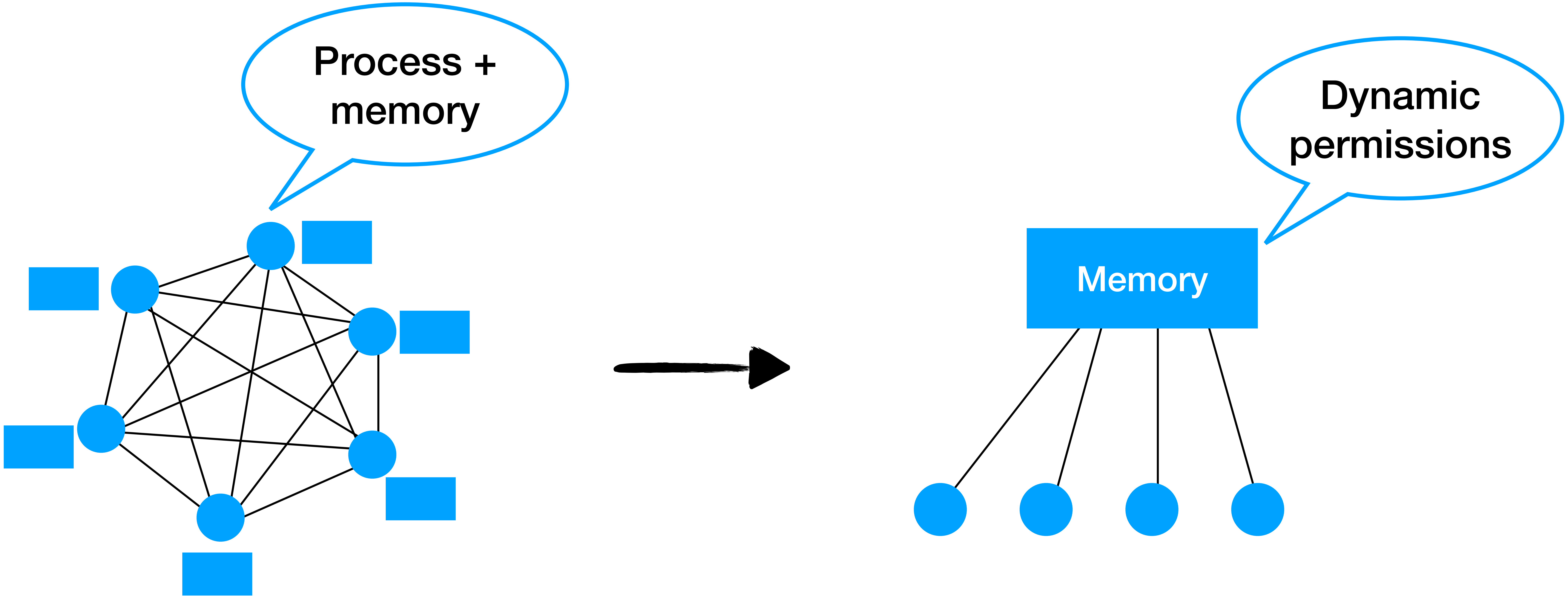


Crash-Tolerant Consensus in the Literature

	Message Passing	Shared Memory
		
Fault tolerance	**f < n/2**	**f < n**
Common-Case Performance	**1 Round trip**	**?**

Can RDMA achieve best of both worlds?

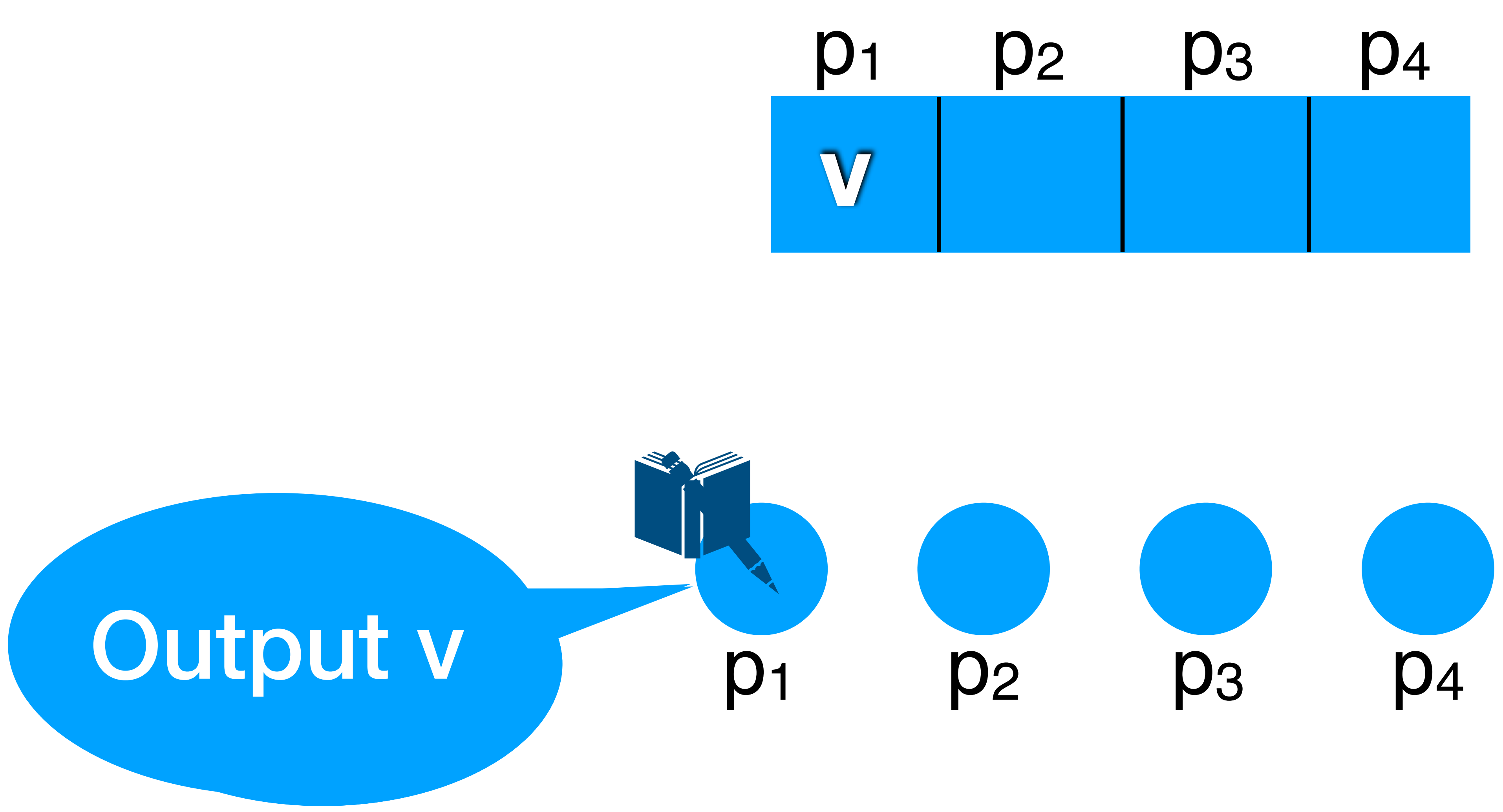
Single-Memory Abstraction



Consensus Algorithm: Disk Paxos

Algorithm: For a designated **leader** process p1
- **Write** your value in your slot
- **Read** disk; If no one else wrote anything
    - Done :-)
- Else
    - More complicated…

Fault tolerance f < n



2 rounds trips for common-case execution

[GafniLamport'02]

Adding Permissions to Disk Paxos

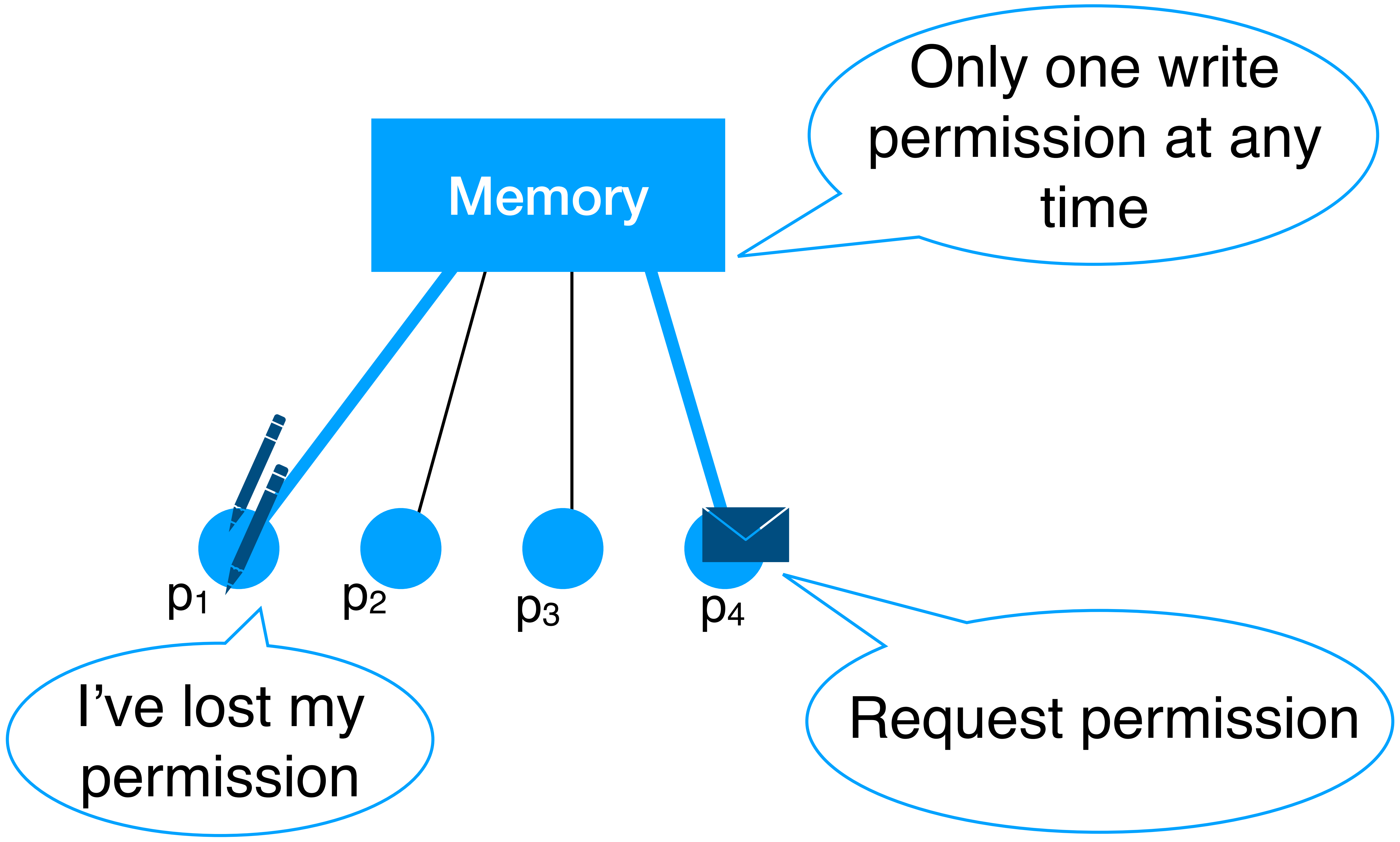
Idea: leverage RDMA dynamic permissions to get rid of check.

Lost permission iff contention

If wrote successfully, no need to read!

2 round trips —> 1 round trip

Fault tolerance f < n



ABGMZ [PODC'19]

Theoretical Results

Theorem [ABGMZ’19]**:**
There exists a **1 round trip** consensus algorithm using **reads**, **writes**, and **dynamic permissions** tolerating **f < n process crash** failures.

In **shared memory:**

Disk Paxos: tolerates **f < n crashes**, but requires **2 round trips**

Theorem [ABGMZ’19]**:**
No consensus algorithm using only **reads** and **writes**, tolerating **f = 1 crash failures**, can terminate in 1 round trip.

Implies result for more failures

Common Case Execution

Bypass remote CPU for improved performance

Replicate requests in **a single one-sided RDMA round trip** when there is **synchrony and no failures**

Theoretical model to reason about algorithmic possibilities

Practical SMR algorithm

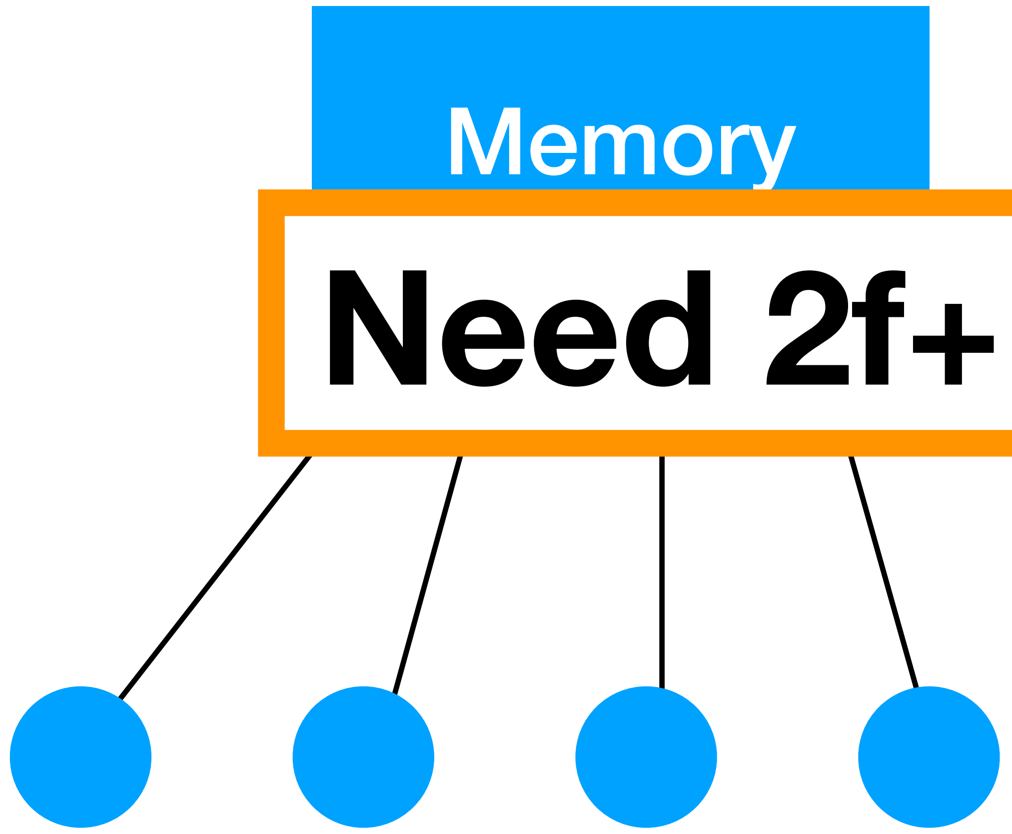
Translating Theory to Practice

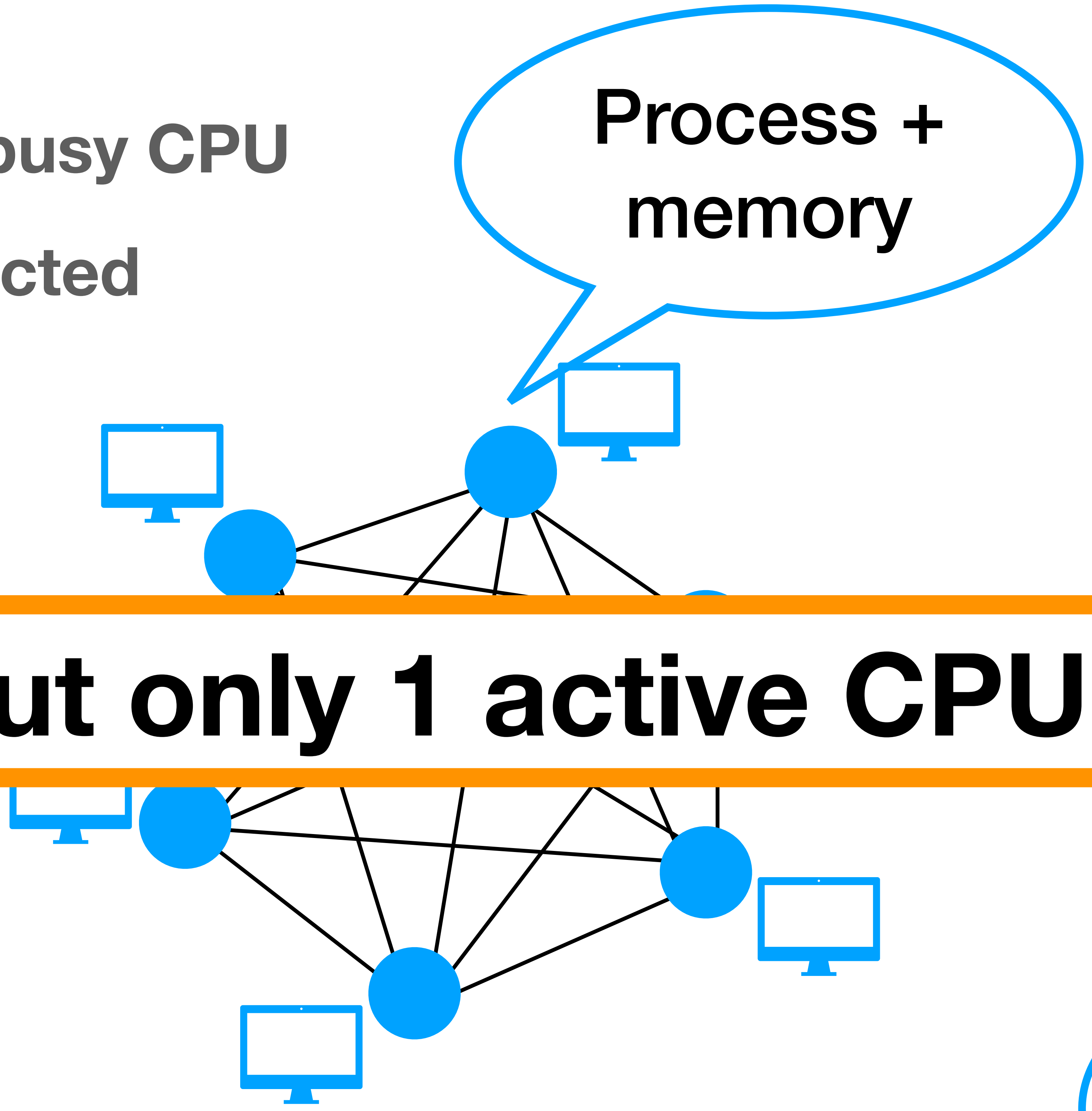
- f+1 fault tolerance to **process failures**; what about **other failures**?
- **Single-shot** consensus to **long-lived** SMR

Practical Fault Tolerance

Process-only failure: software bug, slow/busy CPU

Total failure: NIC failure, machine disconnected



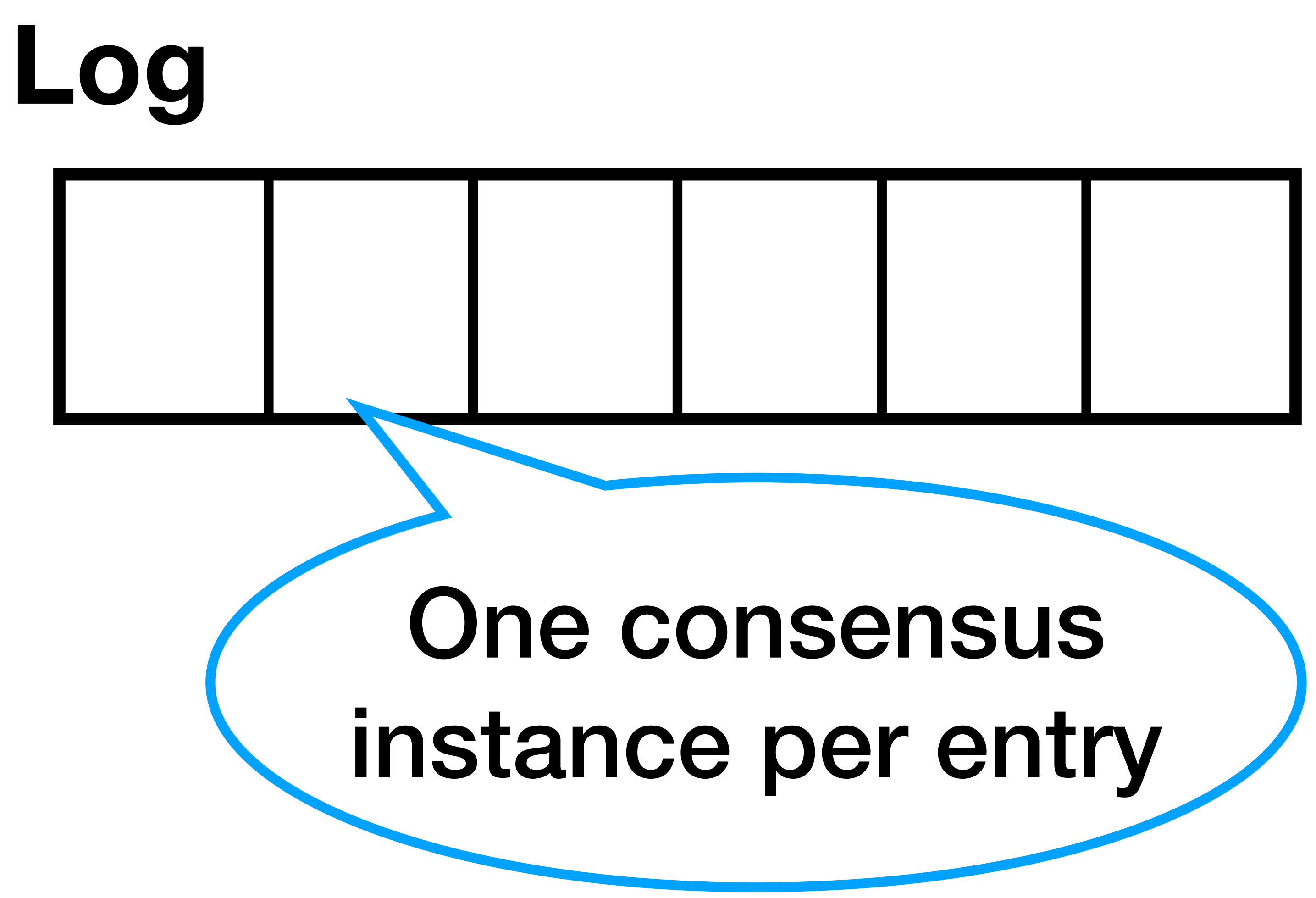


Need 2f+1 memories, but only 1 active CPU

Need 2f+1
memories

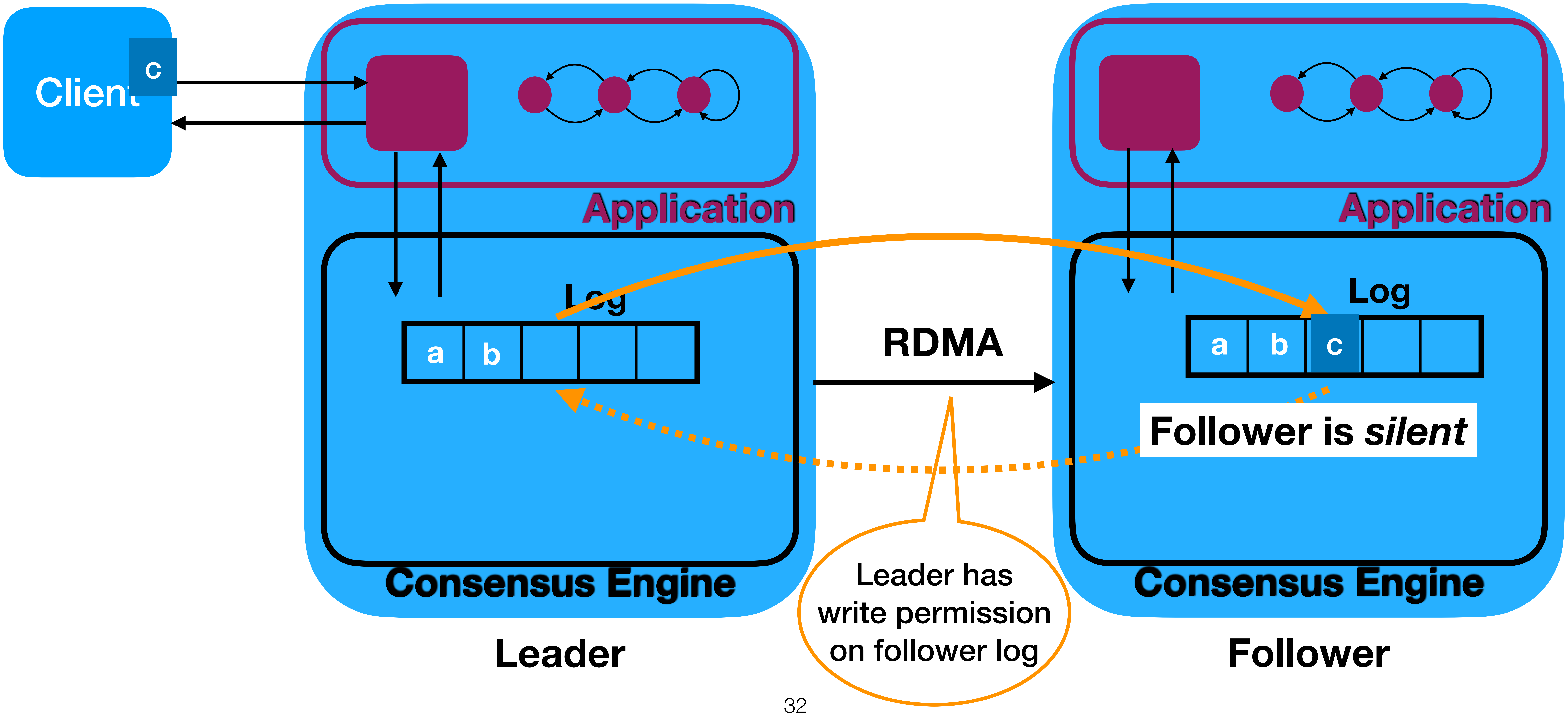
Single memory abstraction is achieved by **replication technique**:
To access global memory, access all memories and **wait for majority**

Multi-Instance Consensus



Leader and other metadata are shared among multiple slots

Basic Mu Architecture



Common Case Execution

Bypass remote CPU for improved performance

Replicate requests in **a single one-sided RDMA round trip** when there is **synchrony and no failures**

~1.3μs
Where our latency comes from

Theoretical model to reason about algorithmic possibilities

Practical SMR algorithm

Roadmap

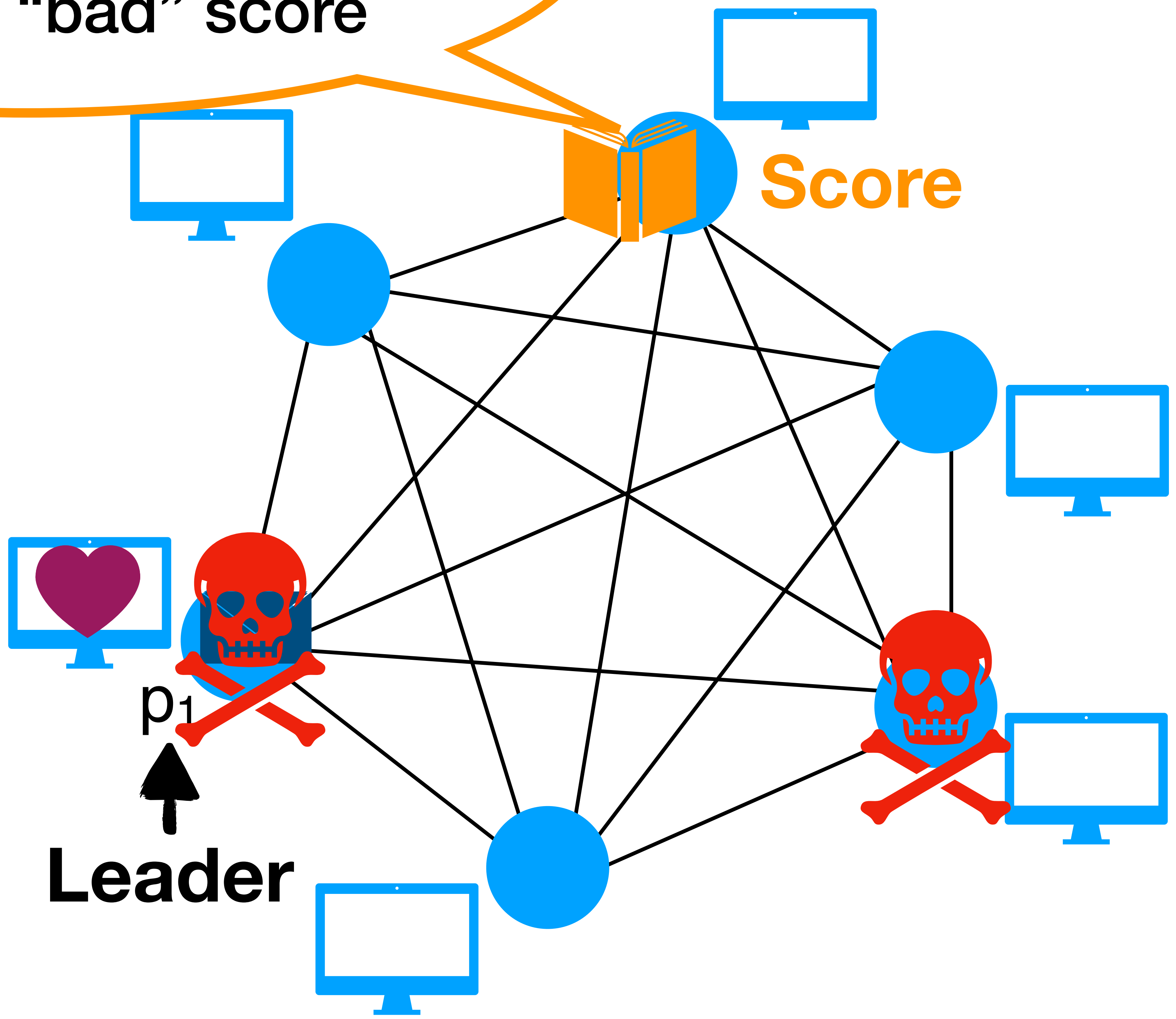
- Case Study: Mu μ
  - ✓ Background: SMR based on **RDMA**
  - ✓ **Best Case**: **~1.3μs replication** overhead
  - **Worst Case**: **<1ms recovery**
  - Experimental Evaluation
- Other **best case**/**worst case** improvements

Failure Recovery

Detect failures more reliably using RDMA and change leaders quickly by **changing RDMA permissions**

Handling Failures

New heartbeat?
Yes - decrease score
No - Raise “bad” score



Non-leader failure doesn’t affect performance

On leader crash:

1. ***Detect leader failure***
2. **Initialize new leader**

Change permission

Push Mechanism

False positive could occur because of slow network

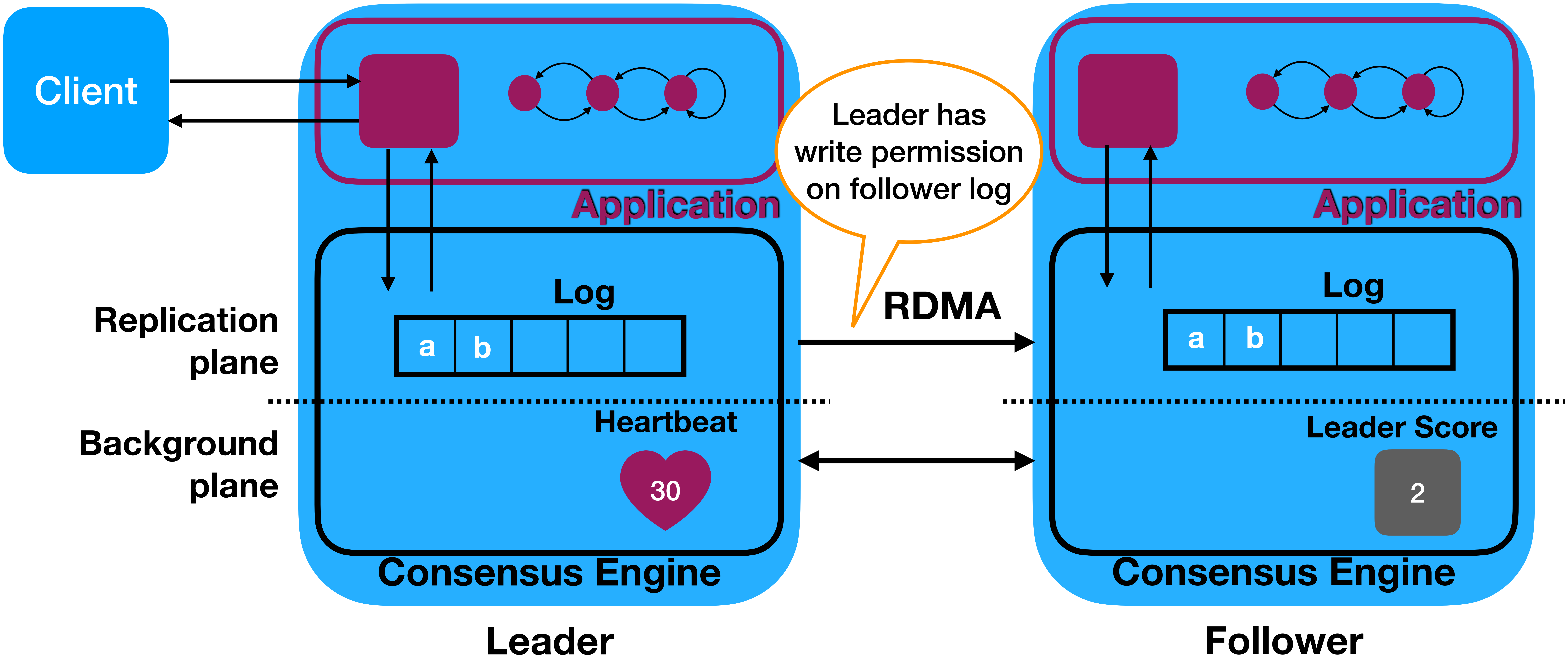
Need conservative timeouts

Pull-Score Mechanism

Slow reads don’t affect heartbeat score

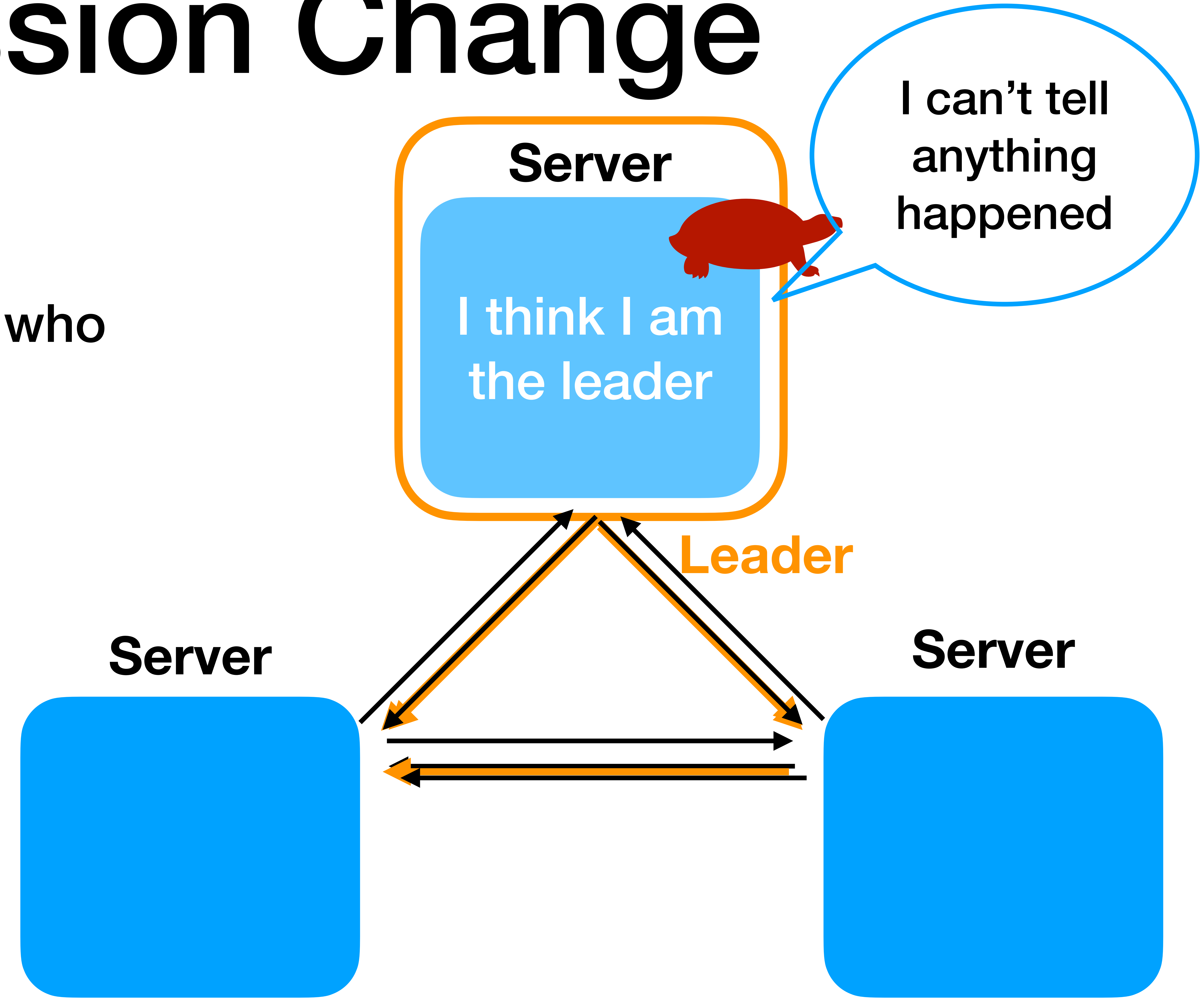
Can have aggressively low score threshold

Basic Mu Architecture



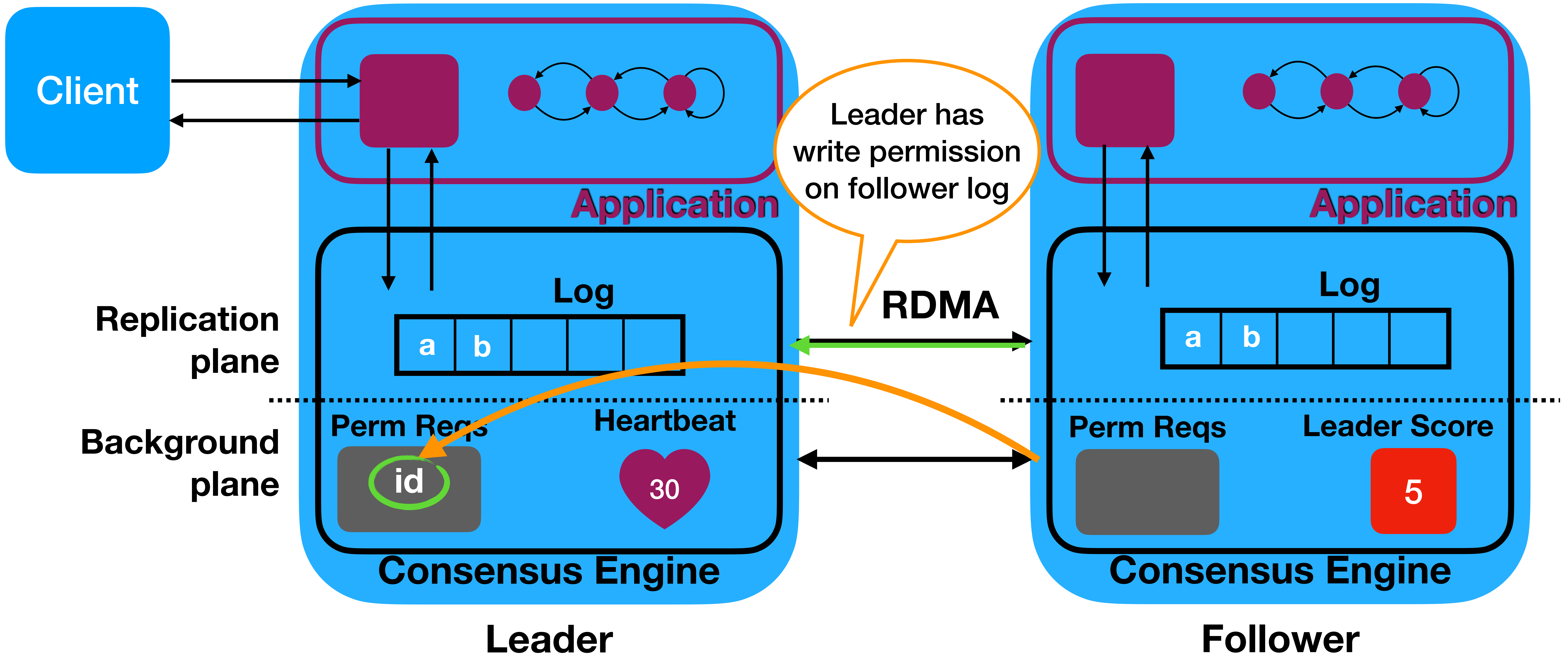
Permission Change

Must request permission:
giving permission to a new leader who didn't request it is dangerous



Permission Request Mechanism

Basic Mu Architecture



Mu System Goals

1 round trip with
silent followers

RDMA-based SMR system with
optimal common-case performance and
improved performance under failures

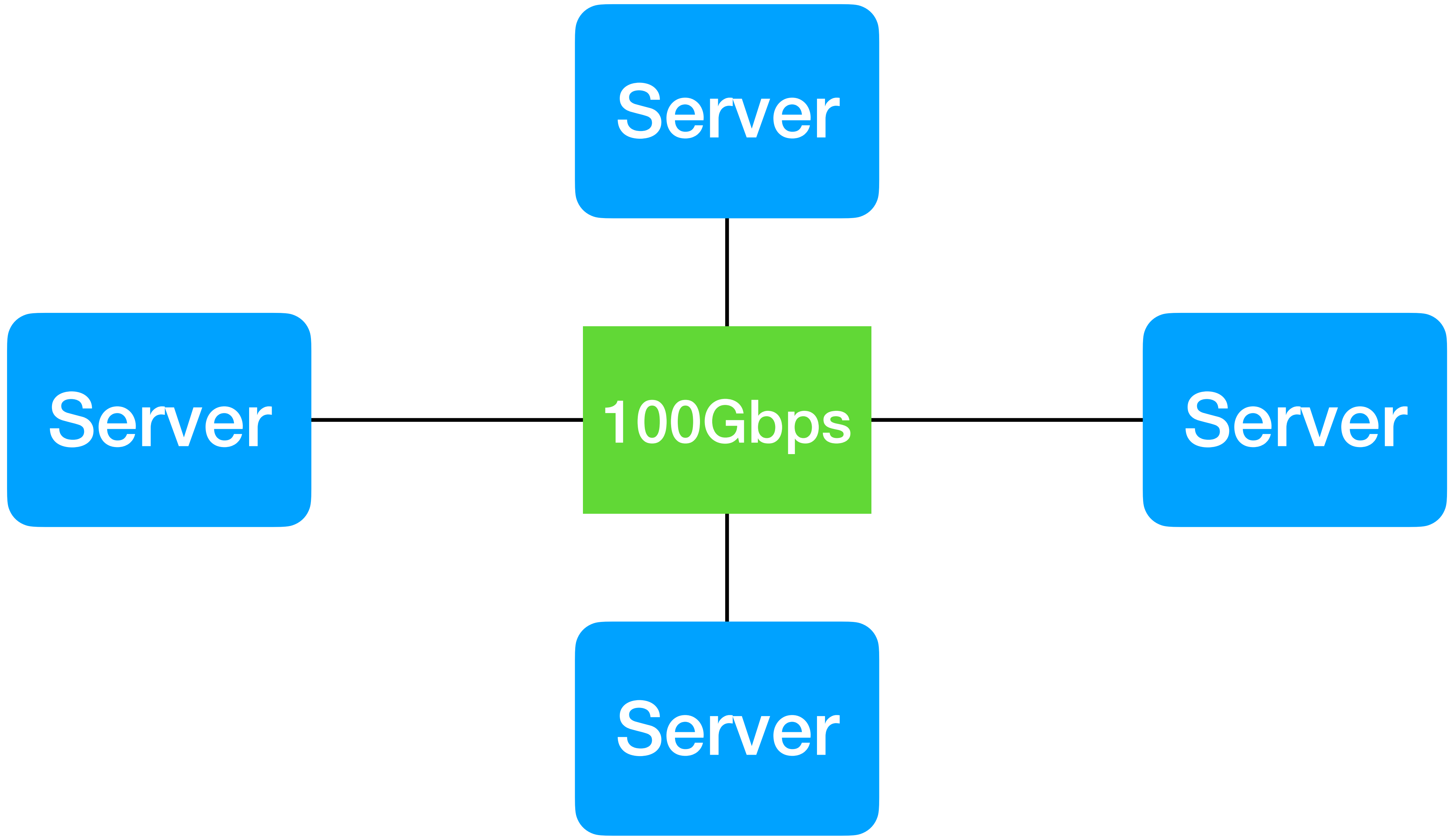
Local heartbeats for
leader election

Roadmap

- Case Study: Mu μ
  - Background: SMR based on **RDMA**
  - **Best Case**: **~1.3µs replication** overhead
  - **Worst Case**: **<1ms recovery**
  - Experimental Evaluation
- Other **best case**/**worst case** improvements

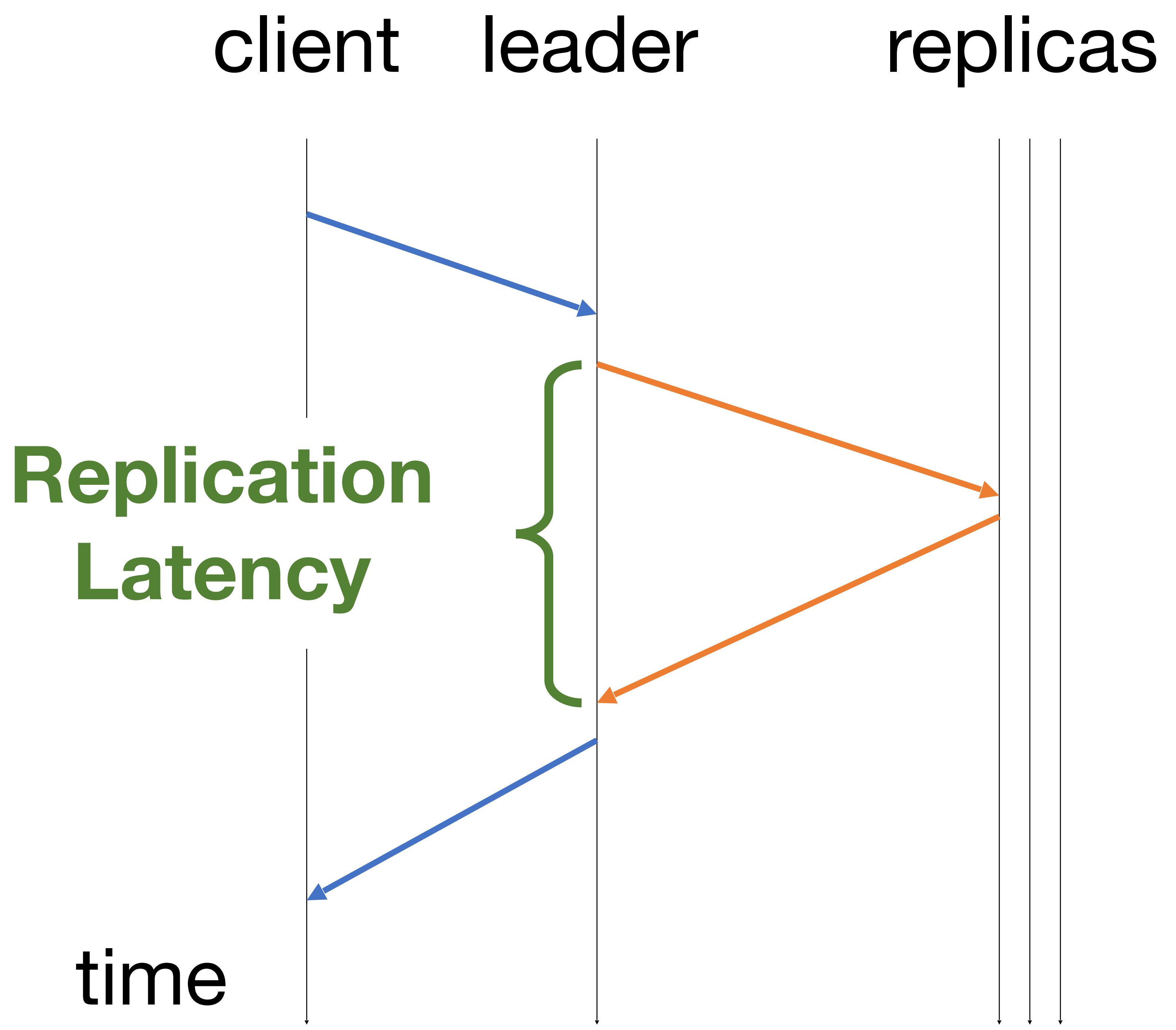
Evaluation: Setup

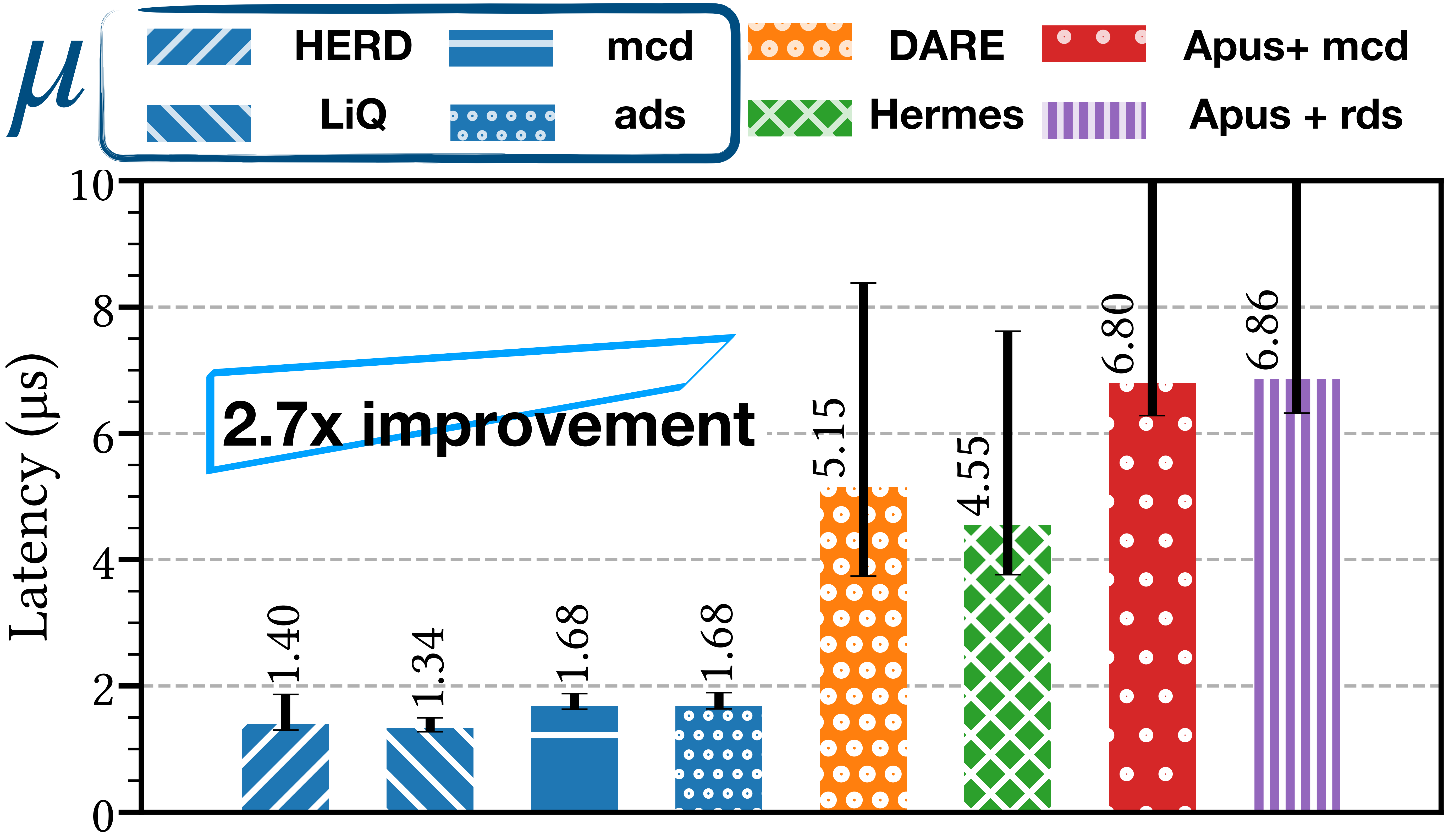
- Metrics
  - Latency, Throughput, Fail-over time
- Applications:
  - RDMA-based: HERD
  - Financial: Liquibook
  - TCP/IP-based: Redis, Memcached
- Competition:
  - DARE [PokeHoefler'15]
  - APUS [WangJiangChenYiCui'17]
  - Hermes [KatsarakisGavrielatosKatebzadeh JoshiDragojevicGrotNagarajan'20]



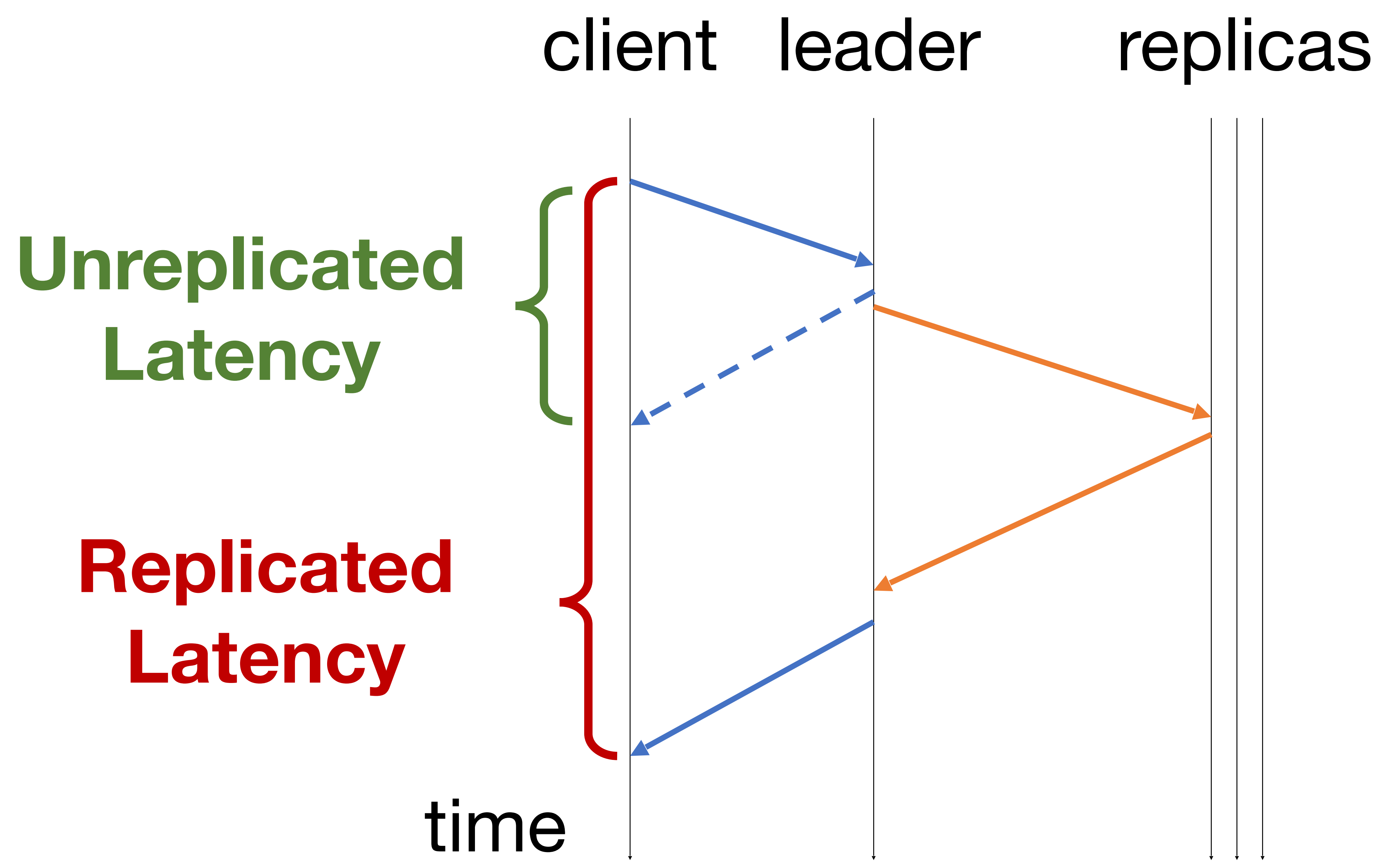
	Mu	DARE	Hermes	APUS
Liquibook	✓	✗	✗	✗
HERD	✓	✗	✗	✗
Memcached & Redis	✓	✗	✗	✓

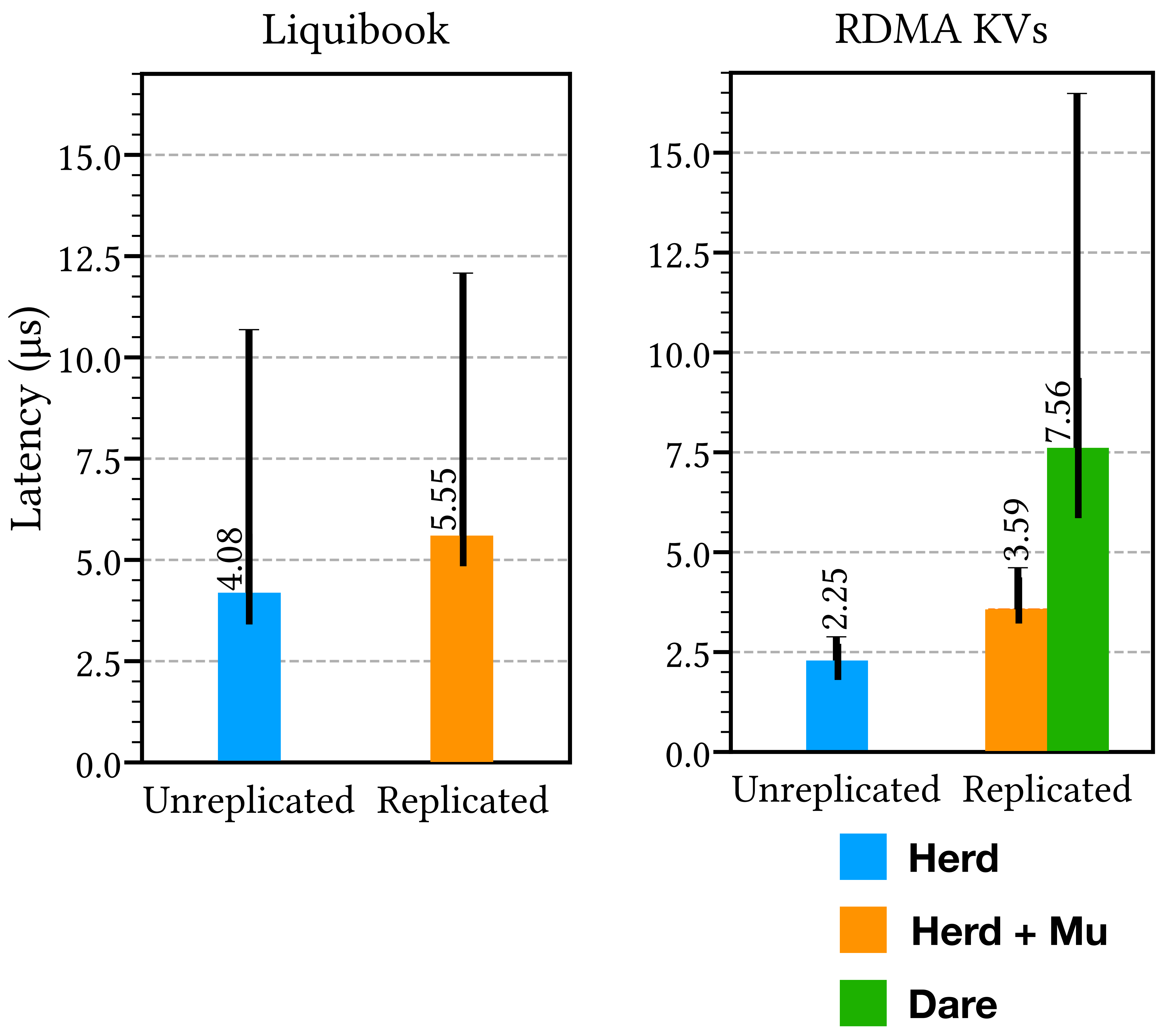
Replication Latency





End-to-End Latency





Throughput

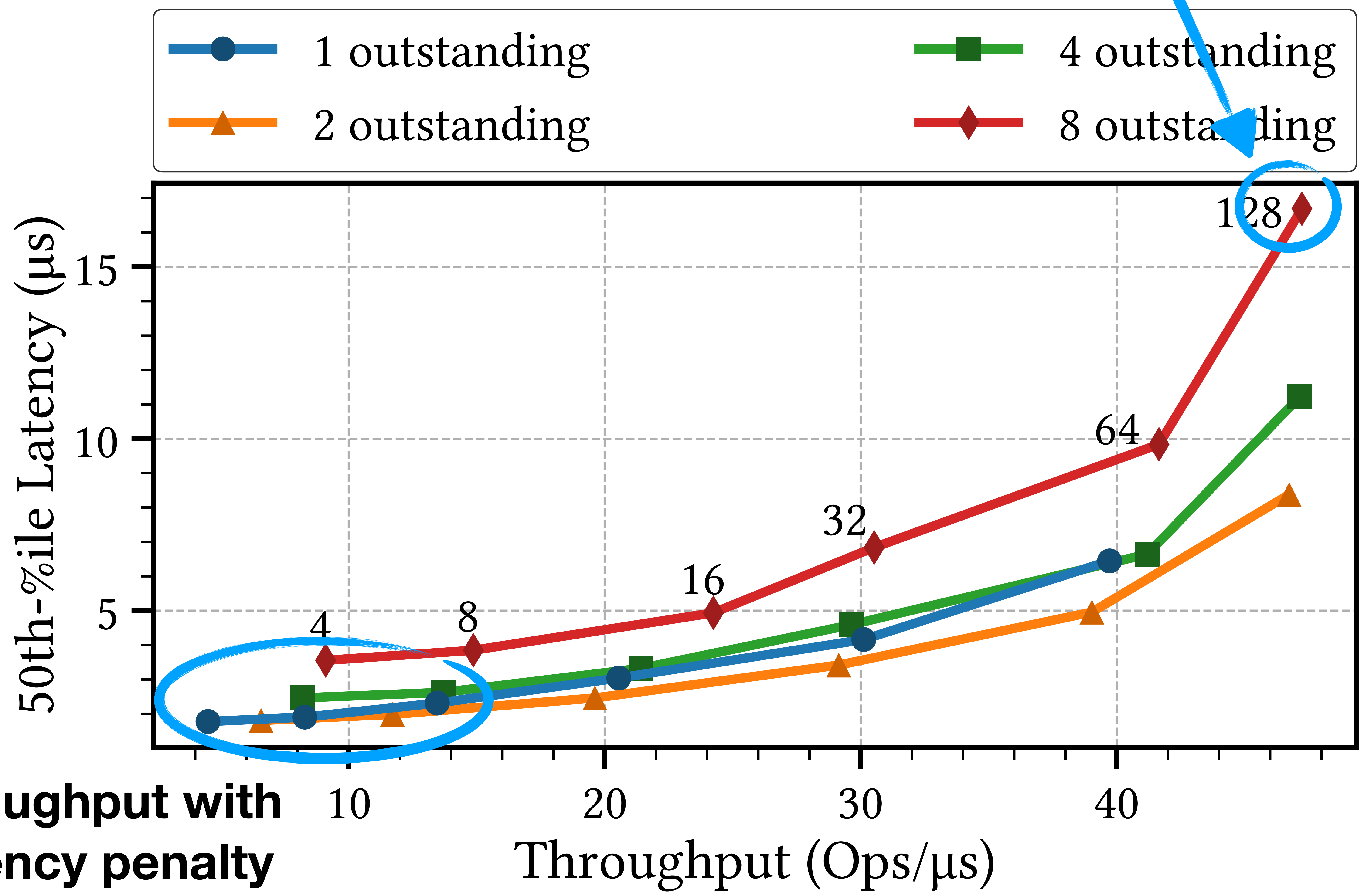
Points on lines

Two throughput optimizations:
- Batching
- Outstanding requests

Different lines

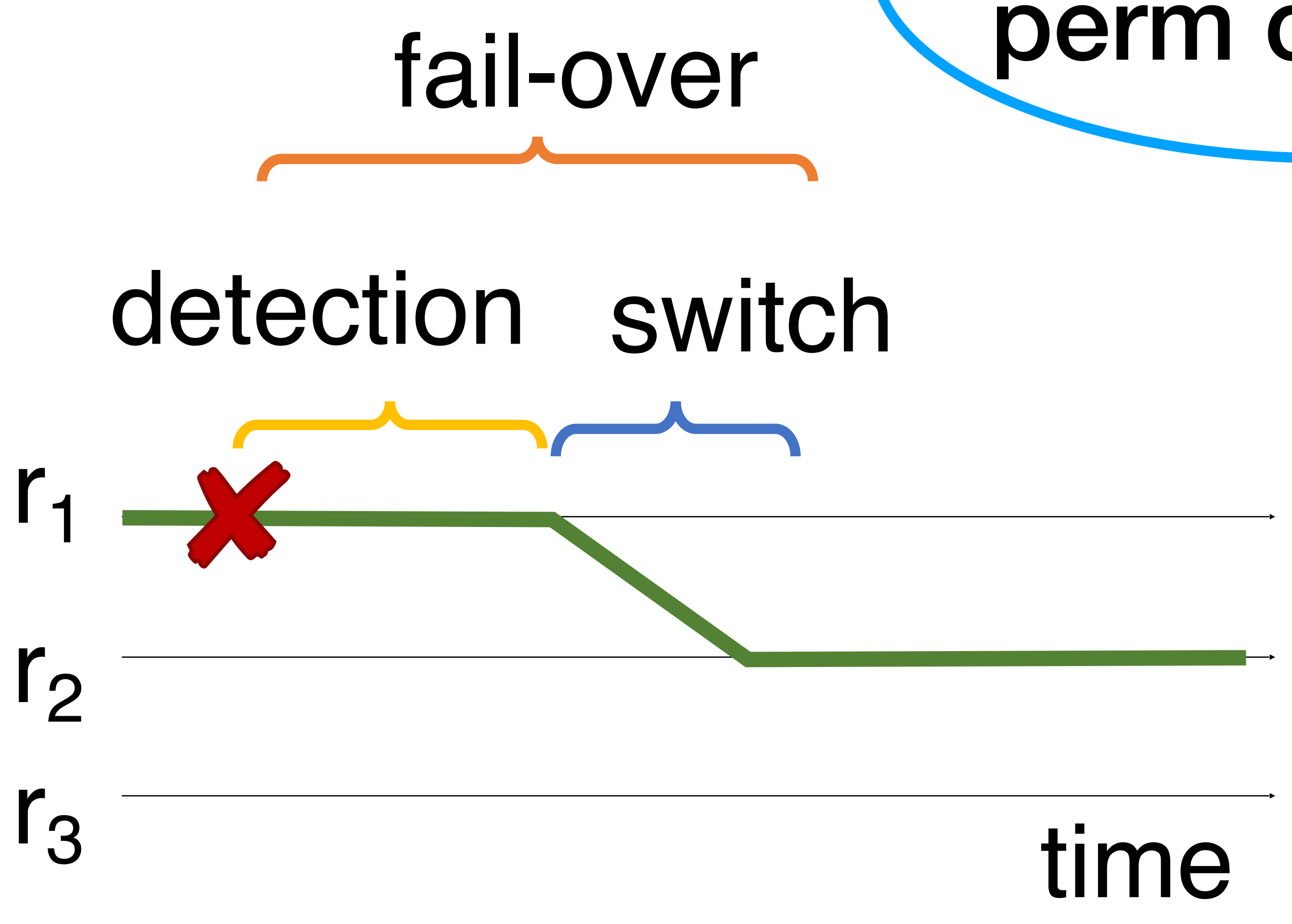
Setup:
Fixed number of clients
64-byte payload

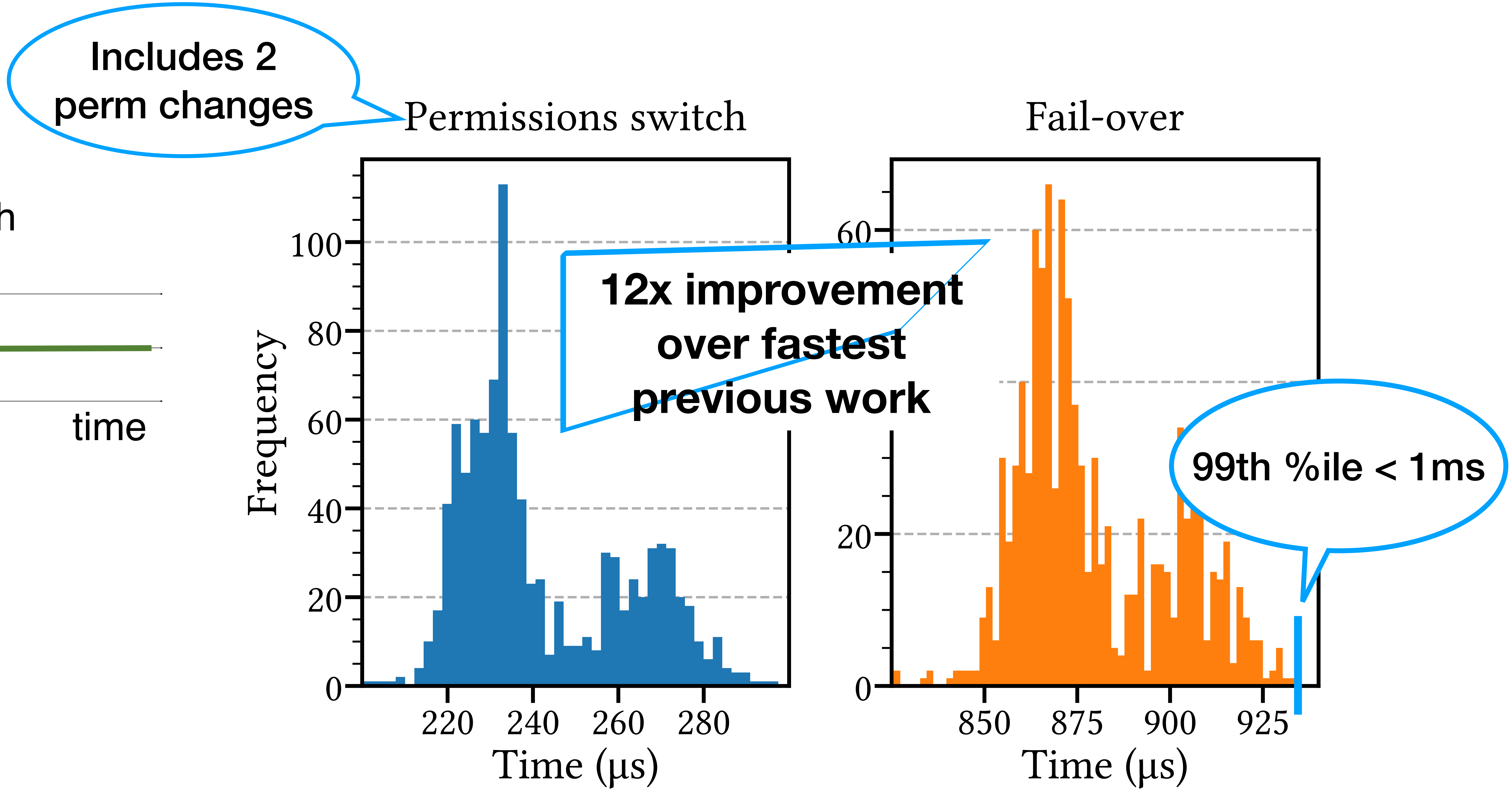
~0.5 of network bandwidth



Increased throughput with almost no latency penalty

Failover time





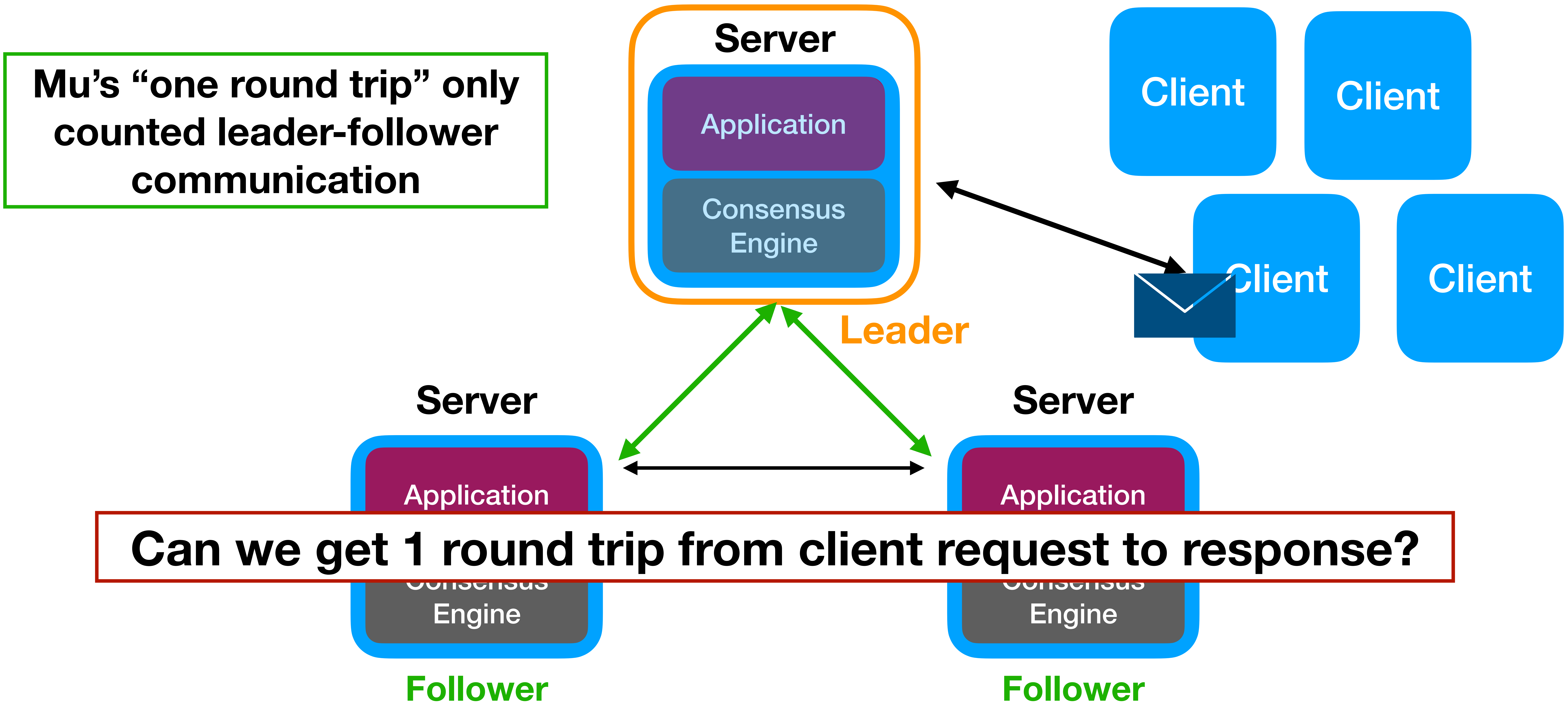
Roadmap

- Case Study: Mu μ
  - Background: SMR based on **RDMA**
  - **Best Case**: **~1.3μs replication** overhead
  - **Worst Case**: **<1ms recovery**
  - Experimental evaluation
- Other **best case**/**worst case** improvements

Other Approaches

- Leaderless consensus - cutting out the middle man
- Byzantine Fault Tolerance - Decreasing Signatures

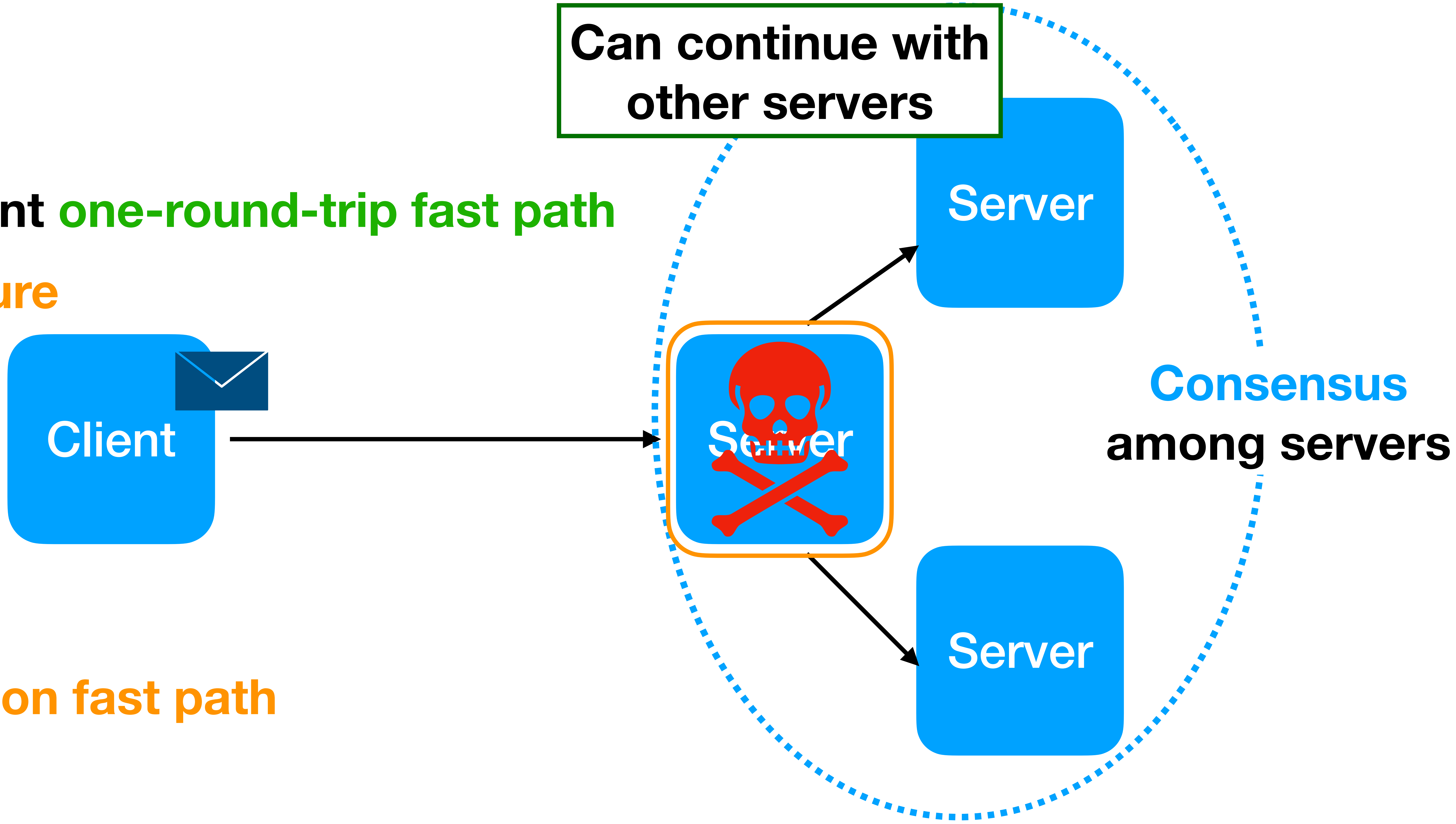
State Machine Replication



Leaderless Replication

Advantages:

- **Possible client-to-client one-round-trip fast path**
- **No single point of failure**



Disadvantages:

- **More complicated**
- **Lower fault tolerance on fast path**

Other Approaches

- ✓ Leaderless consensus - cutting out the middle man
- Byzantine Fault Tolerance - Decreasing Signatures

Signatures

Each process p can:

- sign(v) — outputs $\sigma_{v,p}$
- verify(v, σ, q) — outputs bool indicating whether σ is q's signature of v

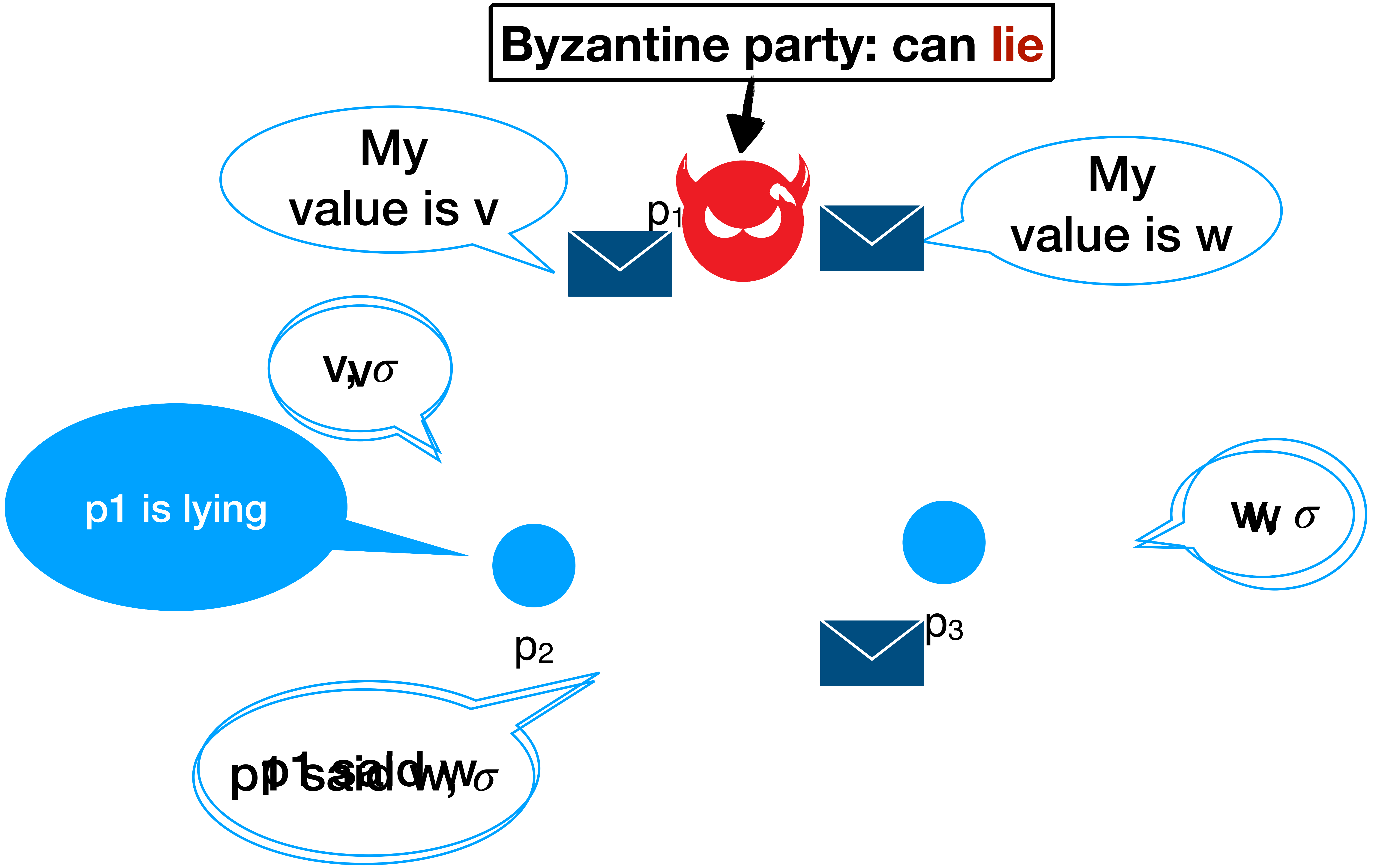
Value specific

Unforgeable

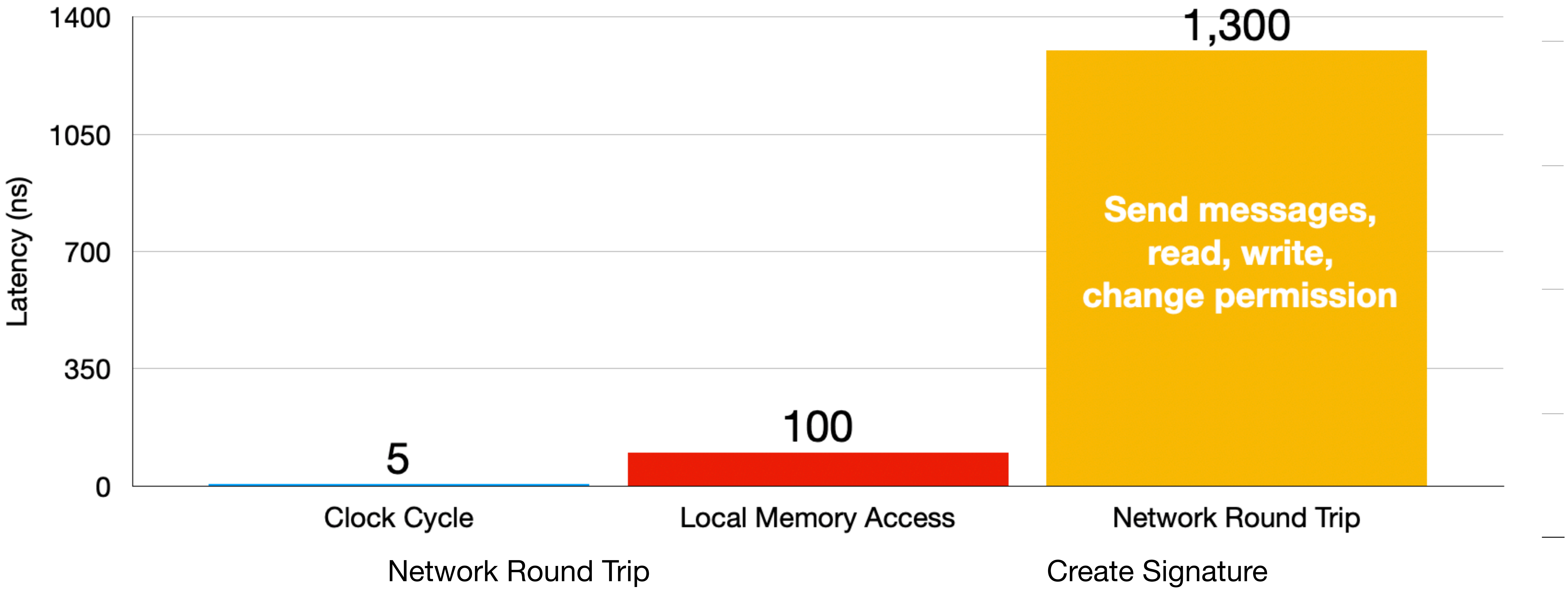
Transferable

Why do we need signatures?

Signatures are **unforgeable** and **transferable**



Signature Cost



Byzantine Fault Tolerance

Helpful for SMR, but needs more work

Broadcast algorithm with no signatures in the common case and optimal number of signatures in the worst case

Lower bound on signatures

Other Approaches

- ✓ Leaderless consensus - cutting out the middle man
- ✓ Byzantine Fault Tolerance - Decreasing Signatures

Roadmap

- Case Study: Mu μ
  - Background: SMR based on **RDMA**
  - **Best Case**: **~1.3μs replication** overhead
  - **Worst Case**: **<1ms recovery**
  - Experimental evaluation
- Other **best case**/**worst case** improvements

Summary

Create algorithms that improve on both best-case and worst-case performance

μ **RDMA-based SMR with 1-round-trip commitment in the common case and better leader election mechanism**

Can leaderless approaches deliver similar or better performance?

Byzantine-tolerant SMR with good best and worst case performance?

Using new hardware to help: NVRAM for durability?

How can we reliably compare approaches?

Thank you!