Microsecond Consensus for Microsecond Applications

Marcos K. Aguilera, Naama Ben-David, Rachid Guerraoui,
Virendra J. Marathe, Athanasios Xygkis, **Igor Zablotchi**

vmware
EPFL
ORACLE Labs

Does Consensus Have to Be Slow?

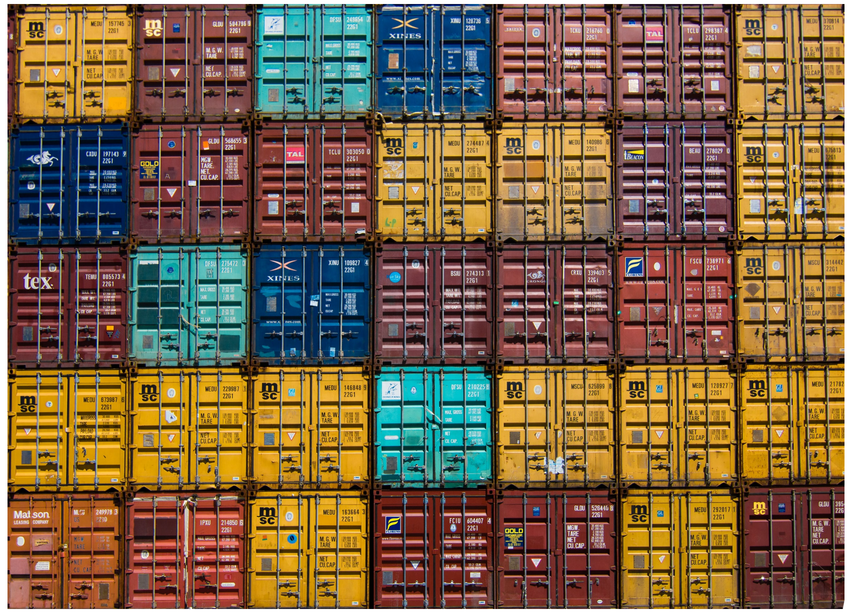
- With the right technology and the right algorithms, no
  - Near-microsecond consensus is possible

Why Care About Microsecond Consensus?

- Microsecond-scale computing is here

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

Embedded systems
(e.g., industrial robots)



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

Why Care About Microsecond Consensus?

- Sometimes desirable to replicate
- Replication should also be at microsecond level
- Existing solutions are too costly
  - TCP/IP adds overhead of >100 us
  - RDMA solutions exist, but we can do better

Need algorithmic solutions!

Our Contribution: Mu μ

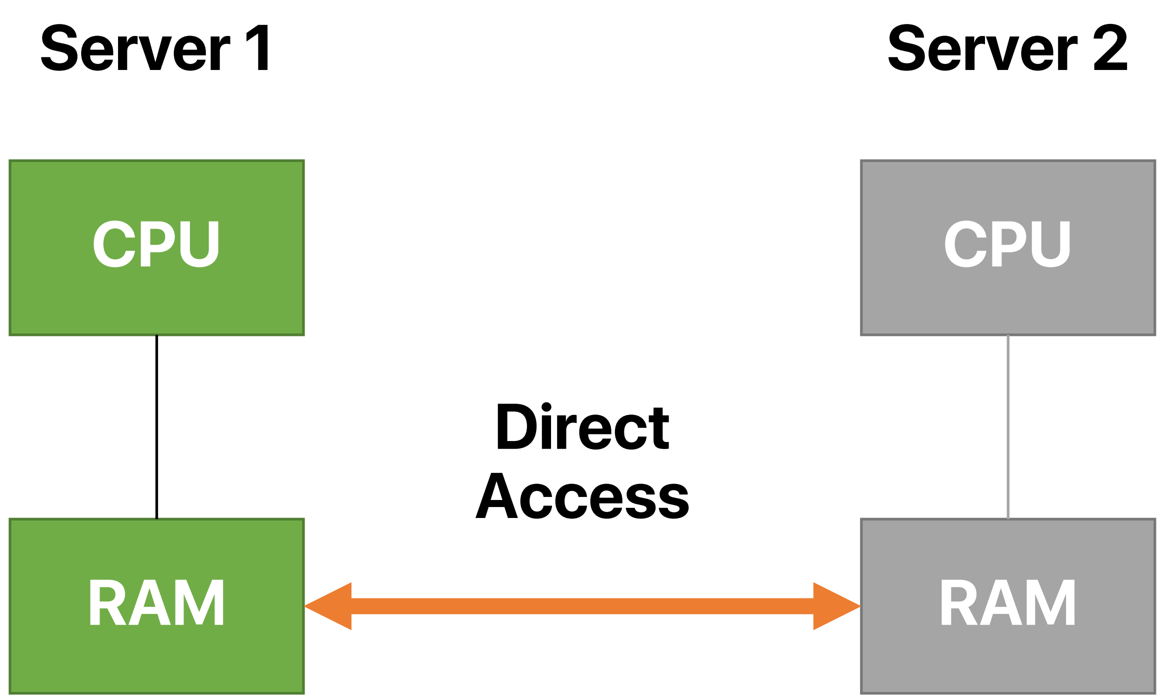
- SMR system for fast networks (datacenter setting)
- Common case: no failures, no asynchrony
  - Replication time is just **1.3 us** (small requests)
  - **61% better** than state-of-the-art
- If leader fails
  - Fail-over to new leader in under **1 ms**
  - **90% better** than state-of-the-art

Outline

- Background: RDMA and SMR
- How does Mu achieve 1.3 us replication latency?
- How does Mu achieve <1 ms fail-over time?
- Evaluation

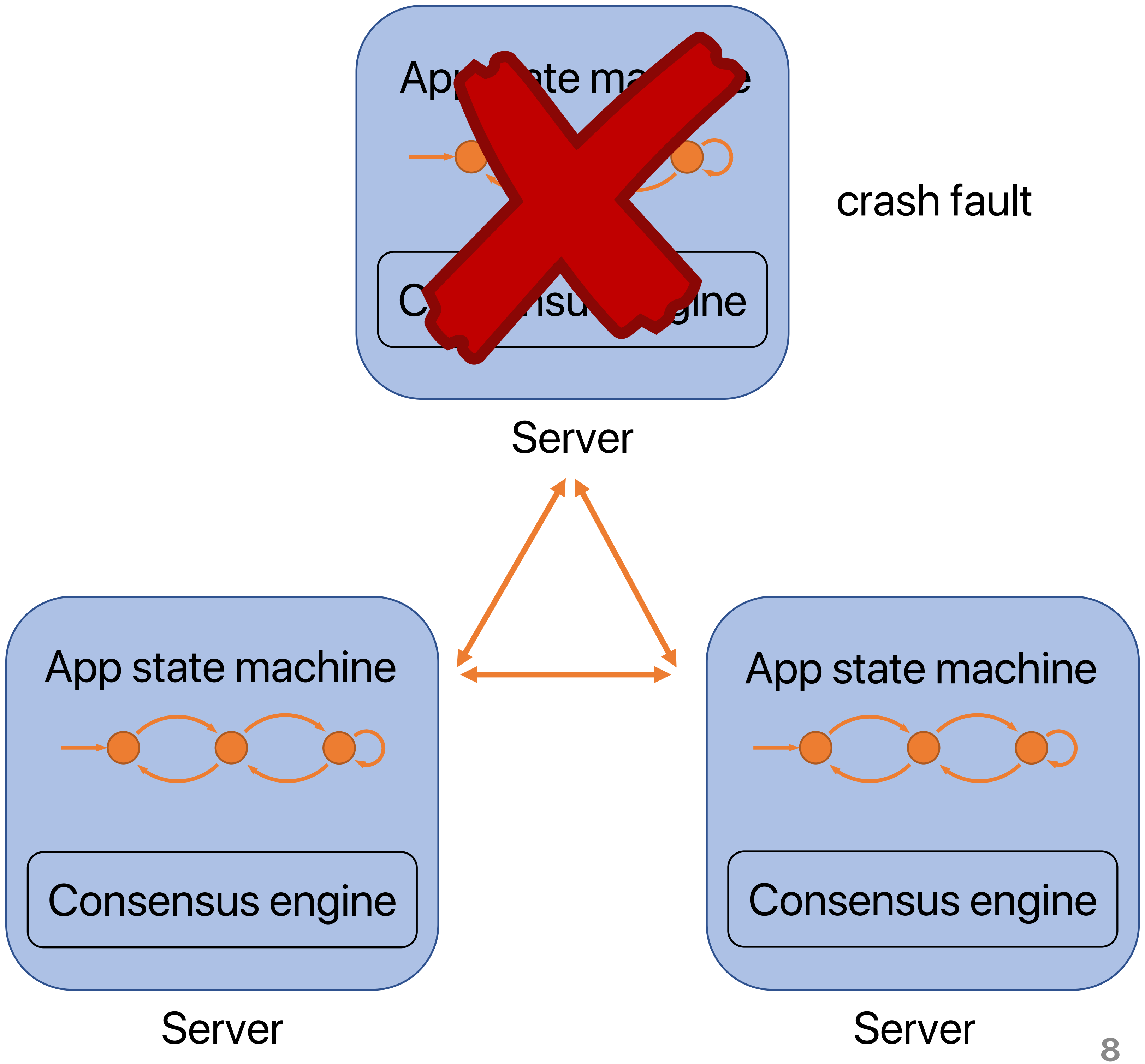
Background: RDMA

- Networking hardware feature
- Direct access to remote memory
  - No CPU at remote side
  - No OS at either side
- Good performance
  - ~1us latency
  - ~100Gbps bandwidth
- Configurable access permissions



Background: State Machine Replication

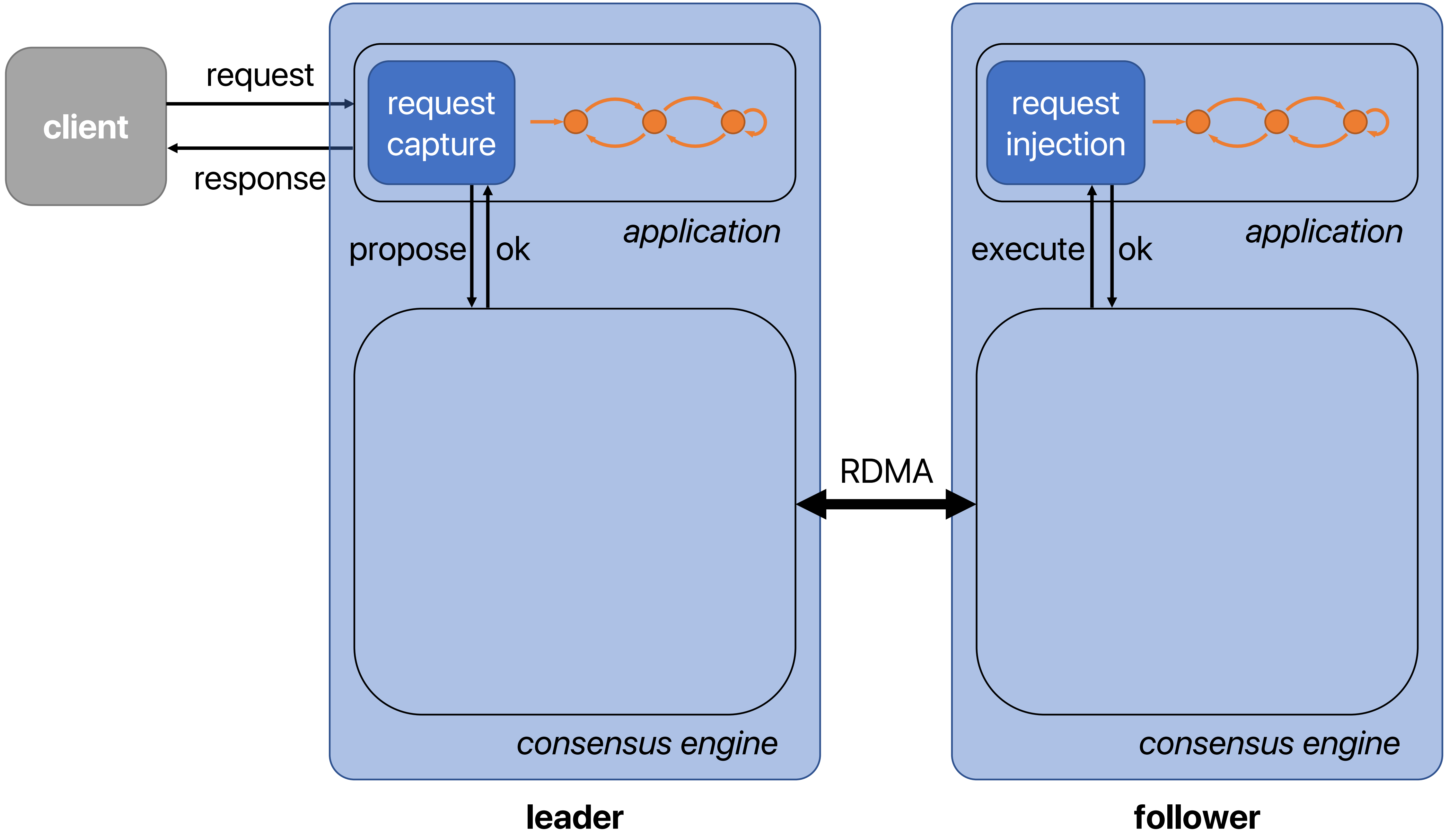
- Replicates a service across several machines = replicas
- Availability, consistency despite replica failures
- Strong consistency: linearizability



Outline

- Background: RDMA and SMR
- How does Mu achieve 1.3 us replication latency?
- How does Mu achieve <1 ms fail-over time?
- Evaluation

Mu Roles



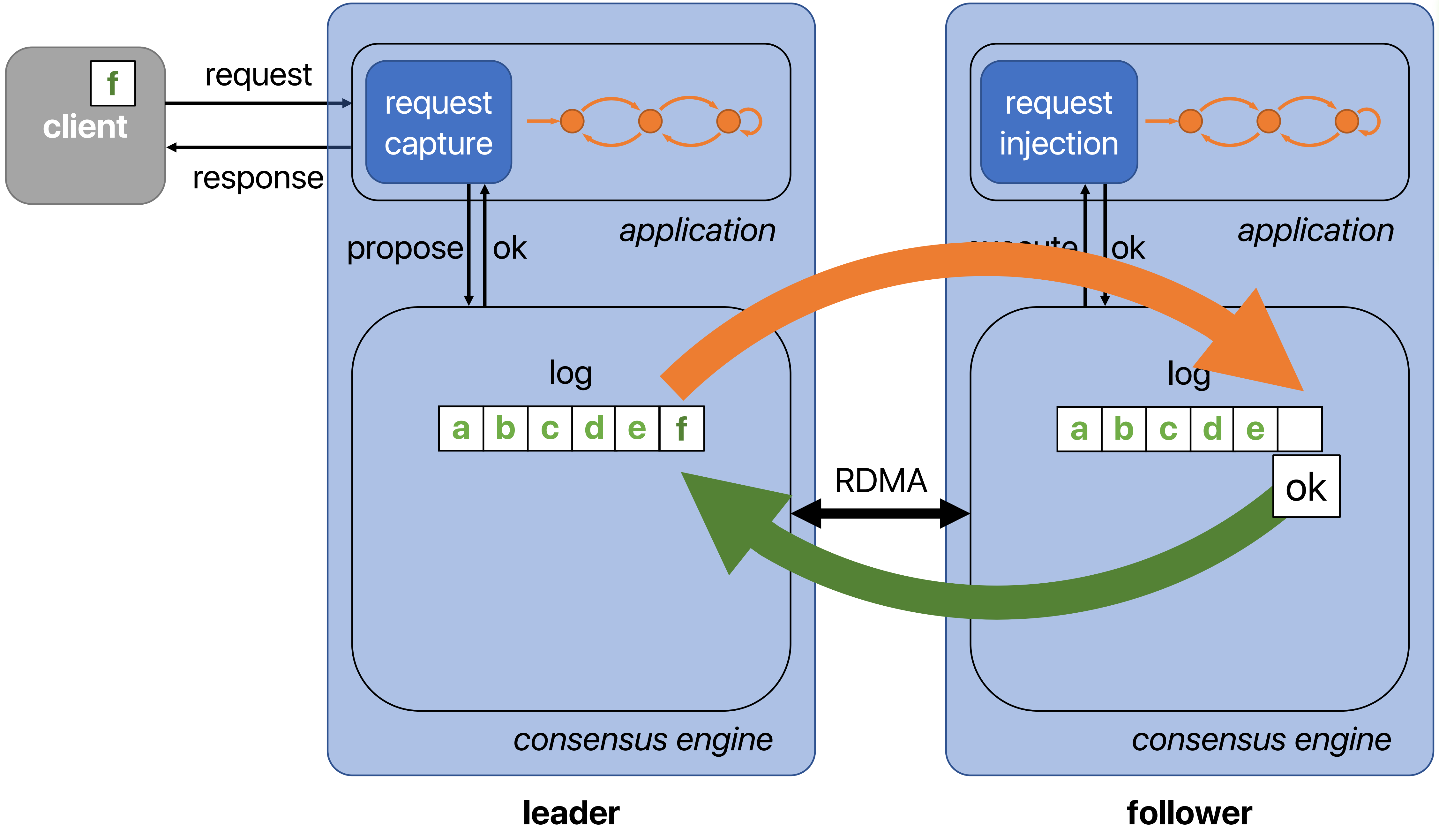
Leader(s):

- Accept requests from client(s)
- Replicate to followers
- Apply to local app copy
- Respond to client(s)

Followers:

- Passively wait for replicated requests
- Apply to local app copy
- Monitor leader health; if leader is slow, switch to new leader

Mu Common Case Replication



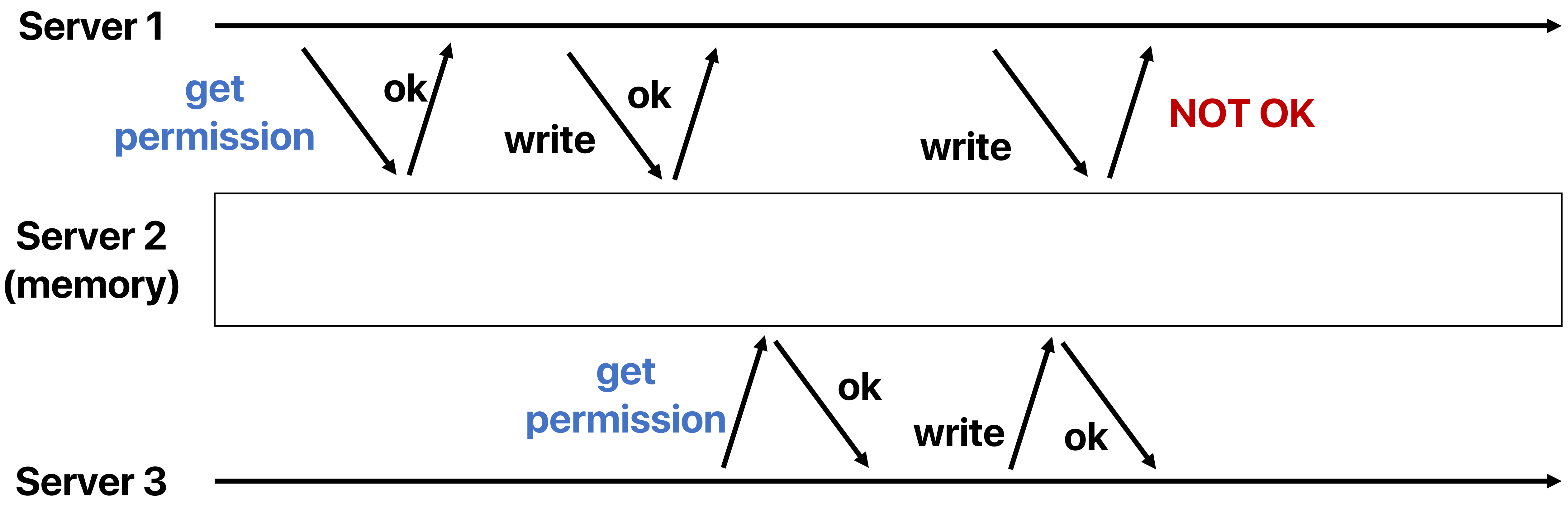
Q: How does Mu achieve ~1 us replication?

1. issue write

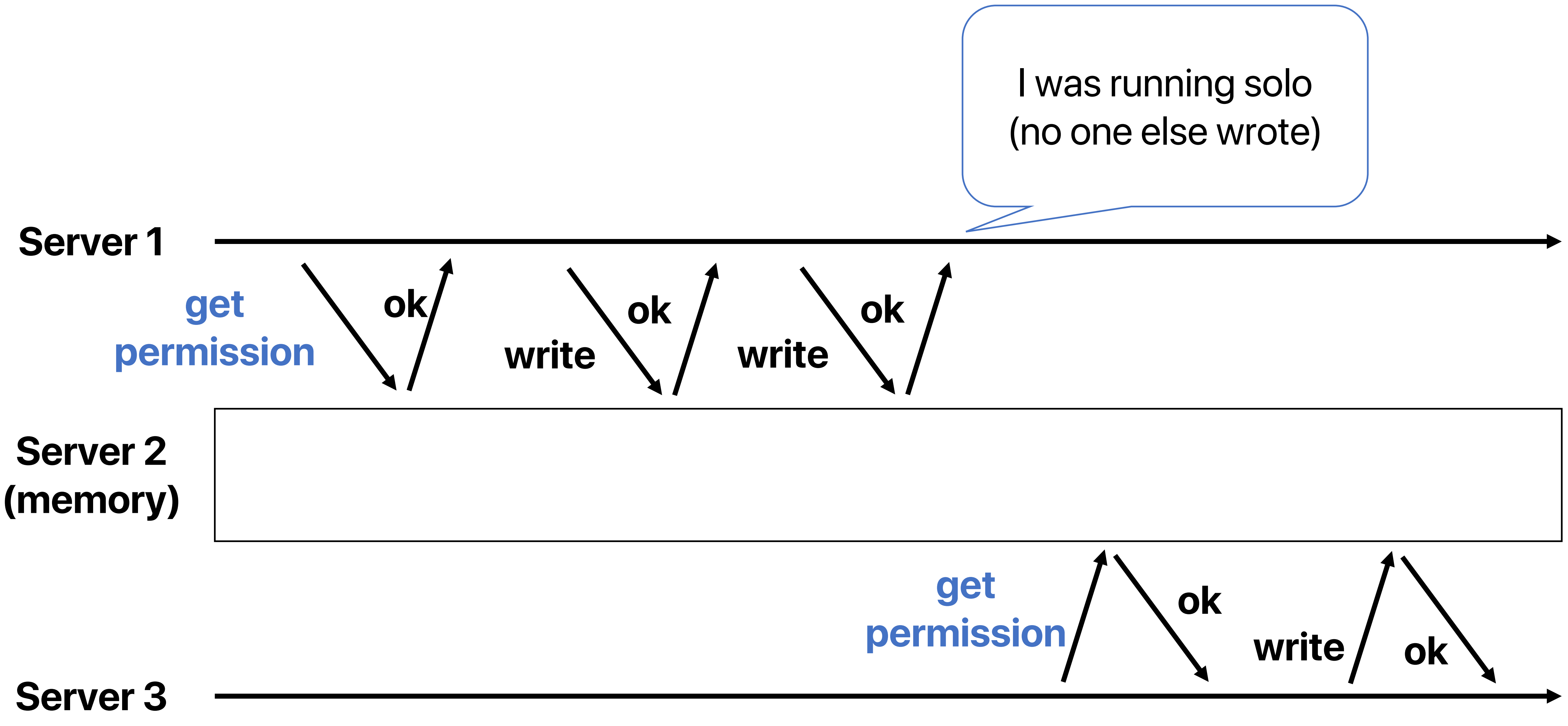
2. wait for completion

Common Case Replication: Intuition

Invariant: only 1 server has write permission on a given memory



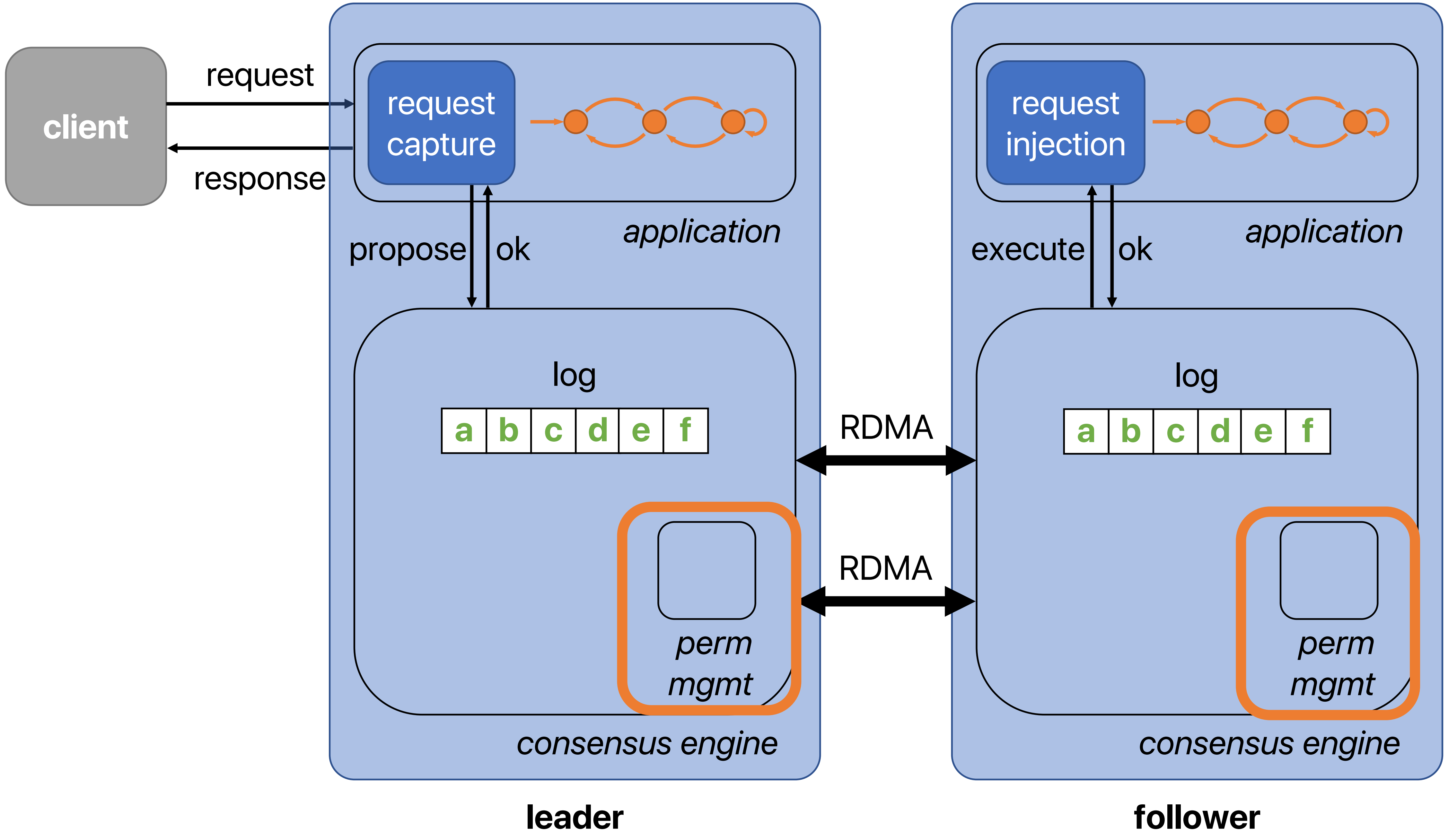
Common Case Replication: Intuition



Mu Background Plane

Q: How does Mu achieve ~1 us replication?

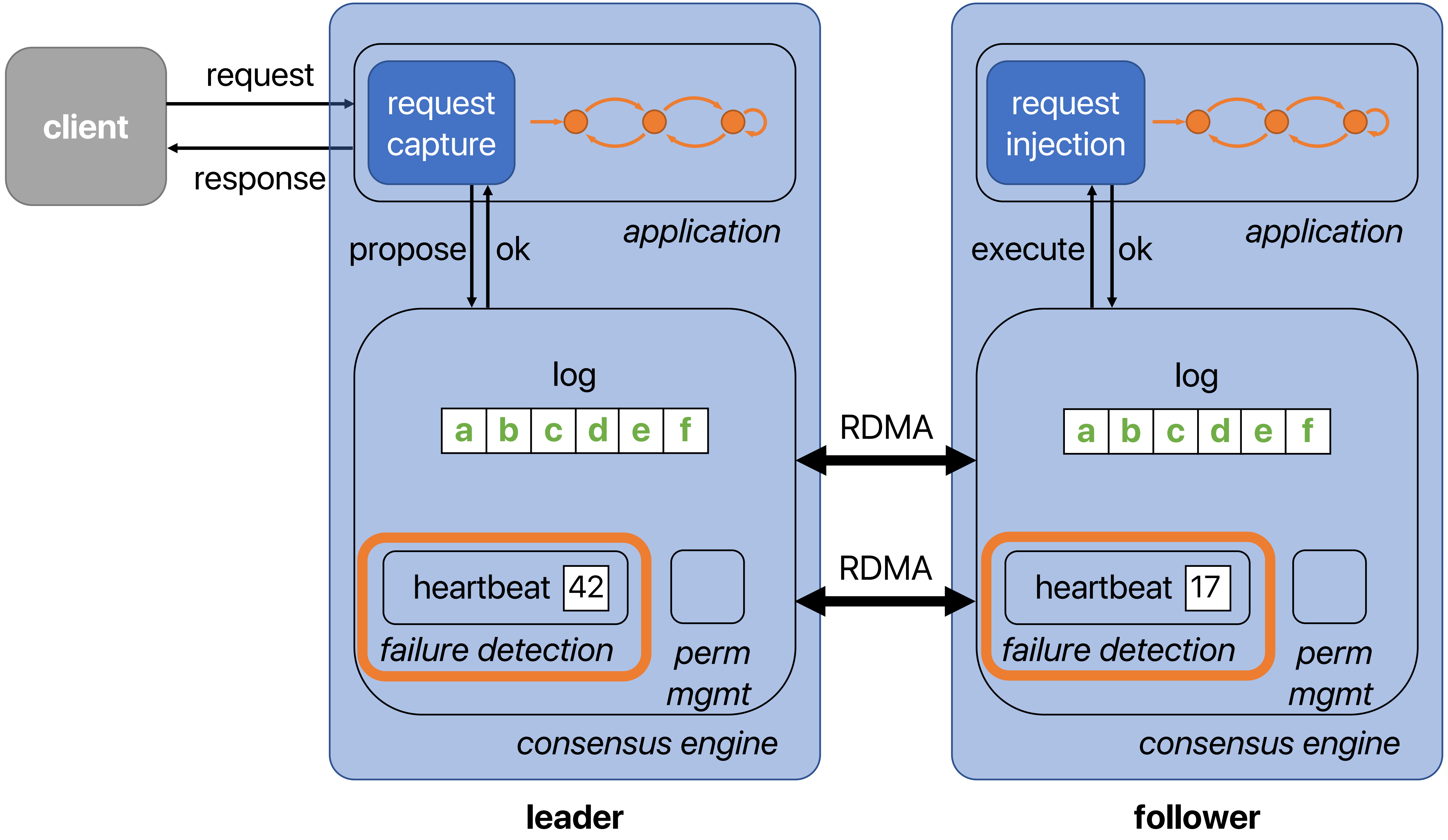
A: Replication = single round trip.
Leader simply writes on followers, relies on permissions to ensure safety.



Outline

- Background: RDMA and SMR
- How does Mu achieve 1.3 us replication latency?
- How does Mu achieve <1 ms fail-over time?
- Evaluation

Mu Failure Detection



Pull-score failure detection

- Increment local heartbeat
- Read remote heartbeats

- Assign badness score to other servers
- Heartbeat stays the same: score ↑
- Heartbeat increases: score ↓

Score not affected by slow reads!

Q: How does Mu achieve <1 ms fail-over?

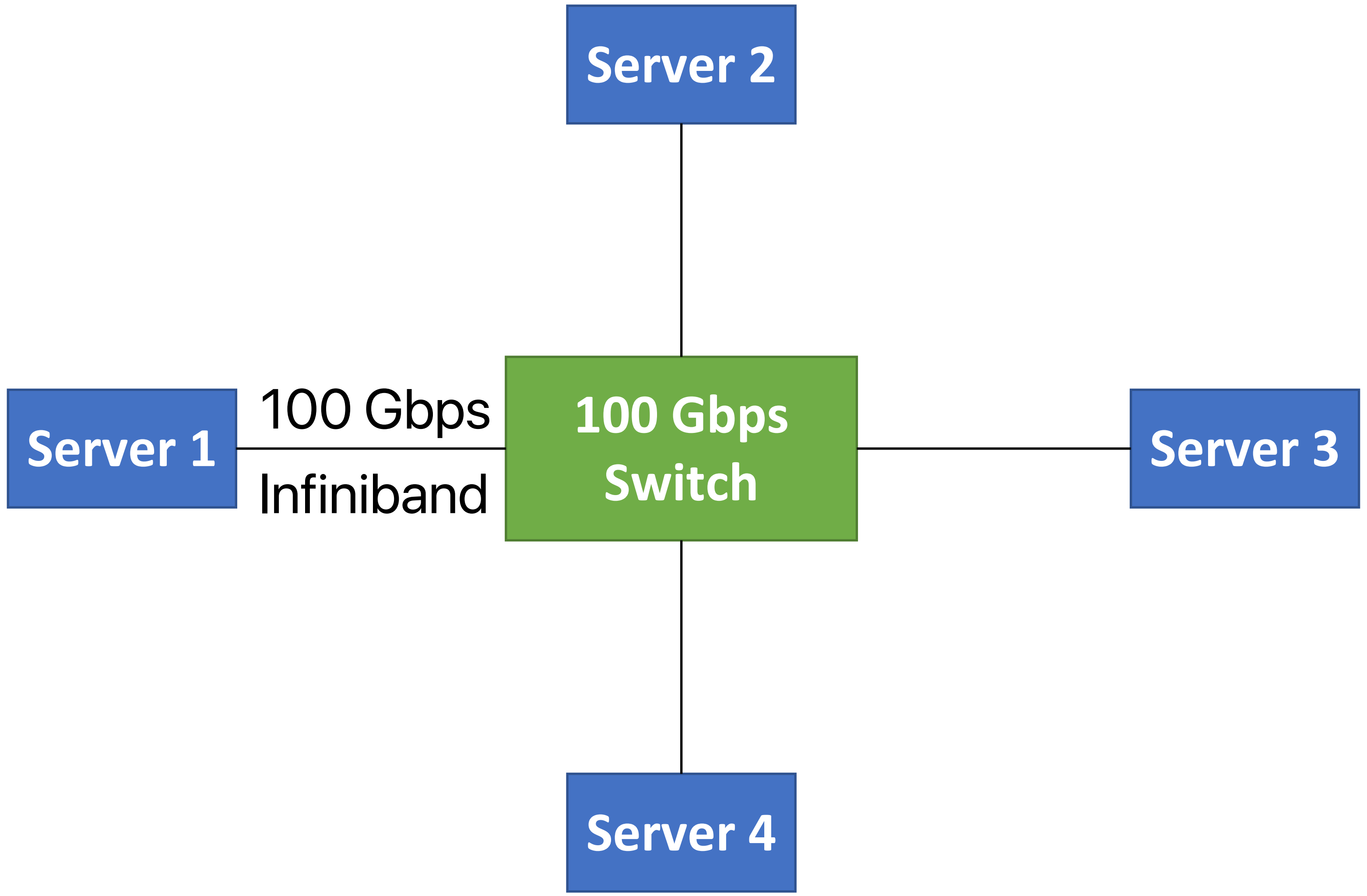
A: Aggressive timeout enabled by pull-score mechanism.

Outline

- Background: RDMA and SMR
- How does Mu achieve 1.3 us replication latency?
- How does Mu achieve <1 ms fail-over time?
- Evaluation

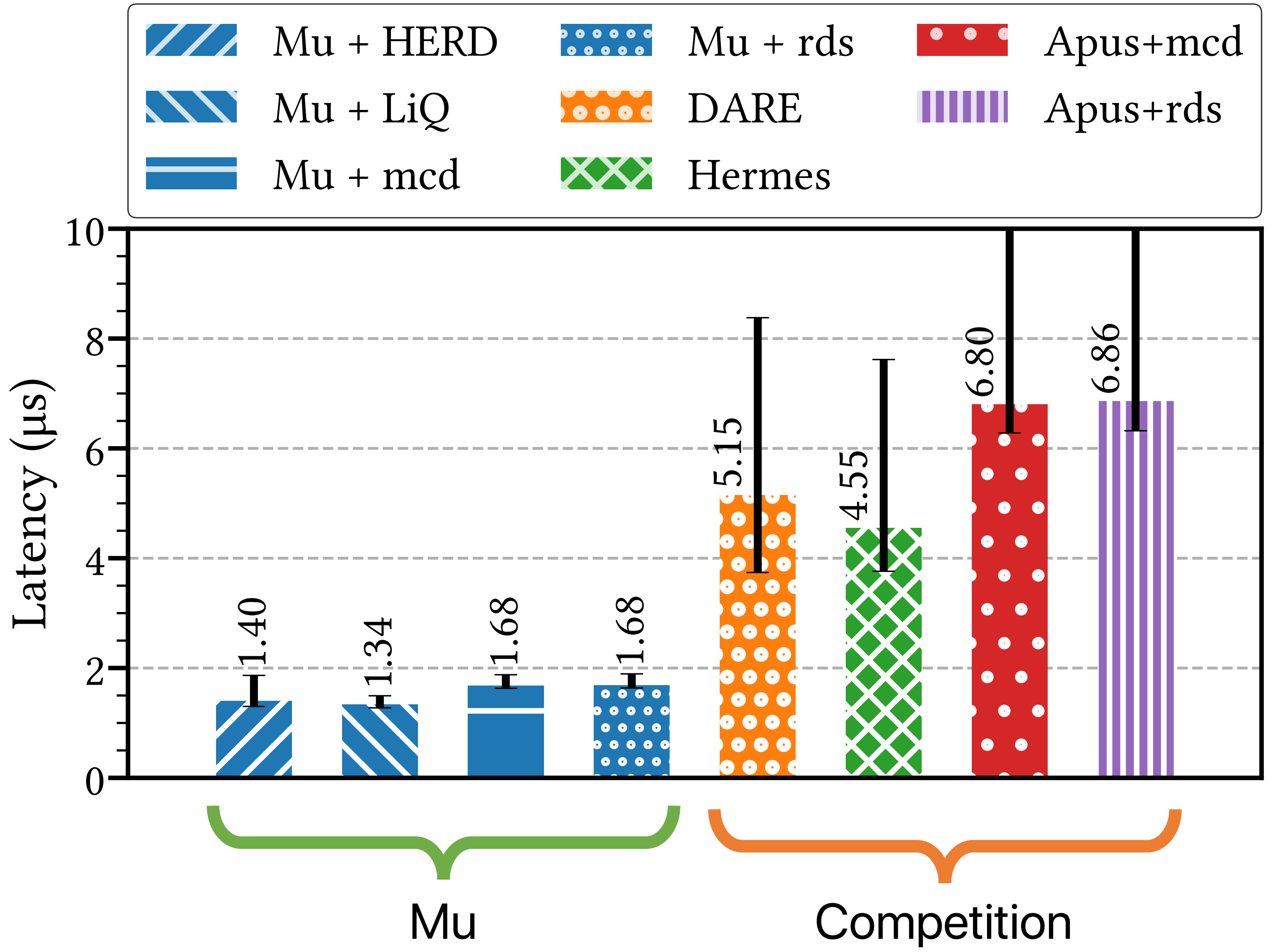
Evaluation: Setup

- Metrics
  - Latency
  - Fail-over time
  - Throughput
- Applications:
  - RDMA-based: HERD
  - Financial: Liquibook
  - TCP/IP-based: Redis, Memcached
- Competition:
  - DARE, APUS, Hermes



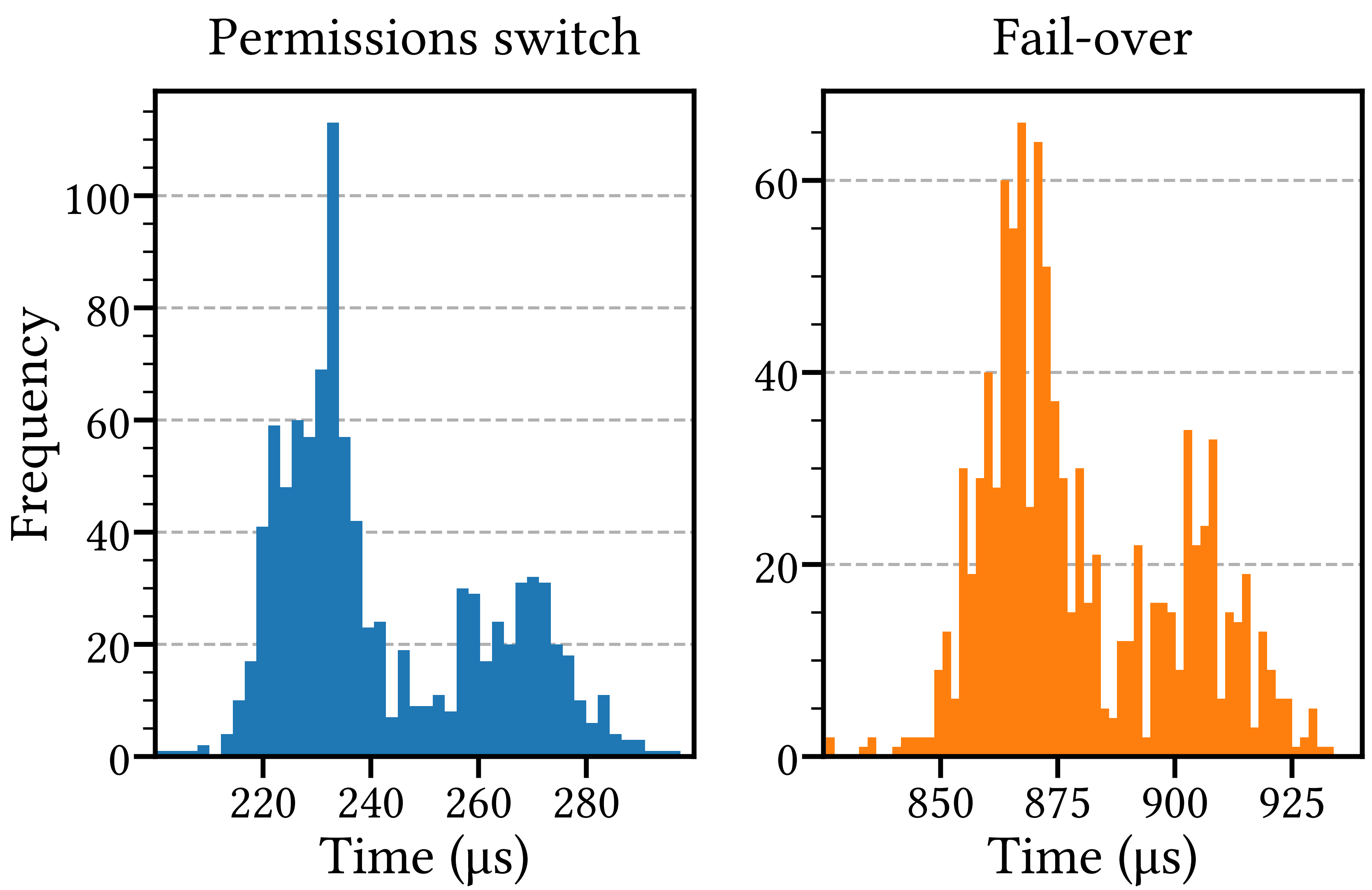
	Mu	DARE	Hermes	APUS
Liquibook	✓	✗	✗	✗
HERD	✓	✗	✗	✗
Memcached & Redis	✓	✗	✗	✓

Evaluation: Replication Latency



Evaluation: Fail-over time

Failure detection + Permission switch time ≈ Fail-over time



Conclusion

- Near-microsecond consensus is possible!
- Important for microsecond apps
- Mu: an RDMA-based SMR system
  - Single round-trip replication → 1.3 *us* replication time
  - Pull-score failure detection → <1 *ms* fail-over time

ARTIFACT EVALUATED usenix ASSOCIATION AVAILABLE

ARTIFACT EVALUATED usenix ASSOCIATION FUNCTIONAL

https://github.com/LPD-EPFL/mu

Check out paper for more!