# Designing an Efficient Replicated Log Store with Consensus Protocol

Jung-Sang Ahn (junahn@ebay.com),
Woon-Hak Kang, Kun Ren, Guogen Zhang, and Sami Ben-Romdhane

Platform Architecture & Data Infrastructure, eBay Inc.

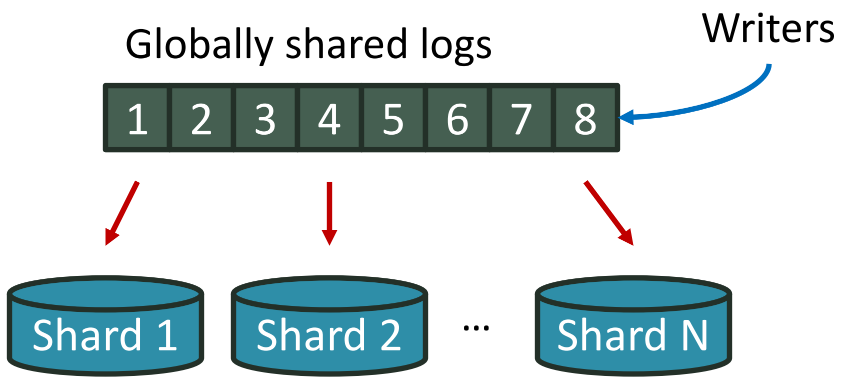
USENIX HotCloud 2019

# Outline

- Purpose & Requirement
- Challenging Issue?
- Our Approach
- Brief Evaluation
- What’s Next?

# Log Store

- Strict ordering
  - Determining global order of execution
  - Useful for distributed transaction
    - GRIT (ICDE 2019, our work)
    - Tango (SOSP 2013), CORFU (NSDI 2012)
    - Calvin (SIGMOD 2012), FaunaDB
    - ...
- Should be
  - Highly available: multiple replicas
  - Highly efficient: lots of subscribers




Same order of execution,
without any coordination.

# Requirements

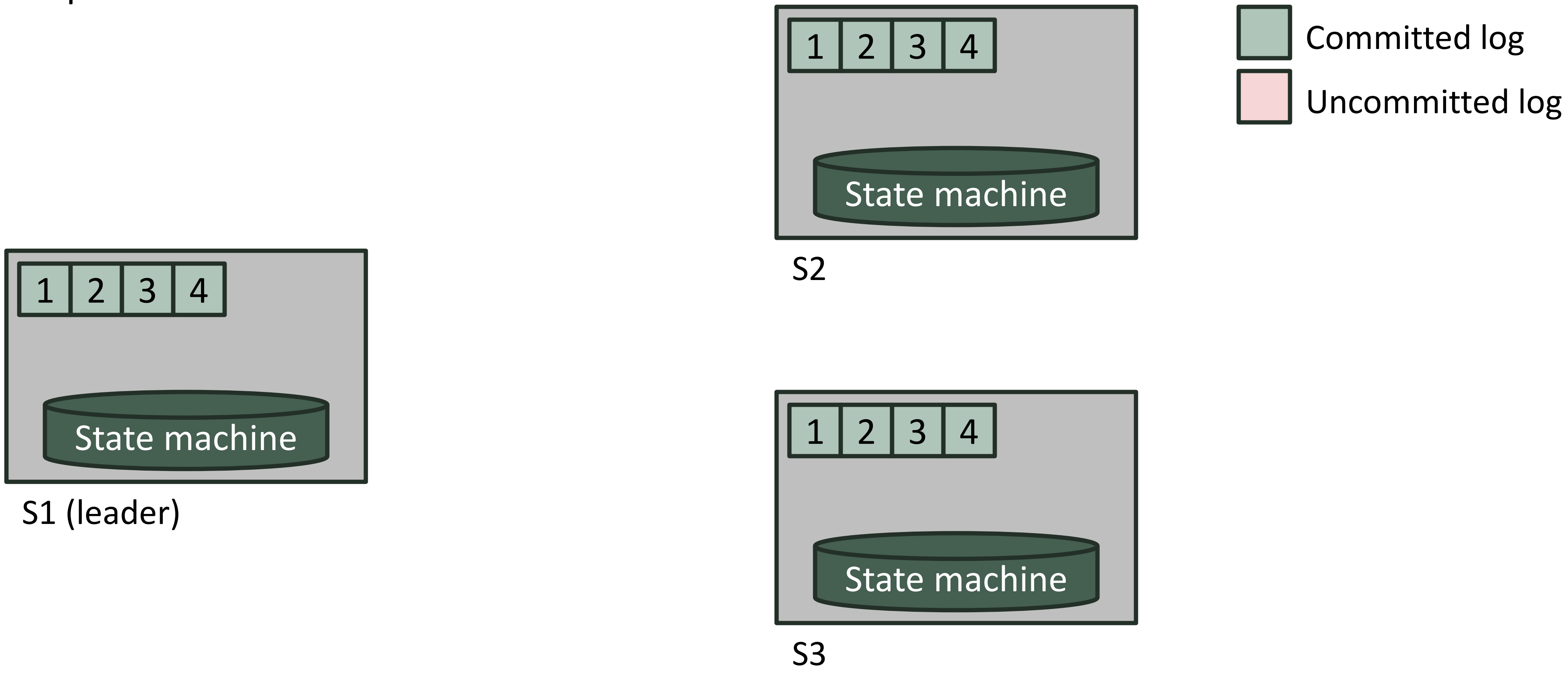
- Each user payload → unique log sequence number (LSN)
  - Non-zero positive number
- All replicas: same data + same LSN order
  - Even after failure and recovery
- No empty LSN in the middle
  - All LSNs should be continuous
- Clients can send payloads in batch
  - Client batch (a set of LSNs) should be committed atomically
  - Partial commit is not allowed

# Problems

- Apache Kafka
  - Can be used as a log store (G. Wang et al. VLDB 2015)
  - No group commit for a single topic
  - Multiple clients: one is blocked by previous replication from other client
- What if we use consensus protocol?
  - Multi-Paxos (L. Lamport 2001), Raft (ATC 2014), etc.
  - Raft guarantees aforementioned requirements

# Log Store with Raft

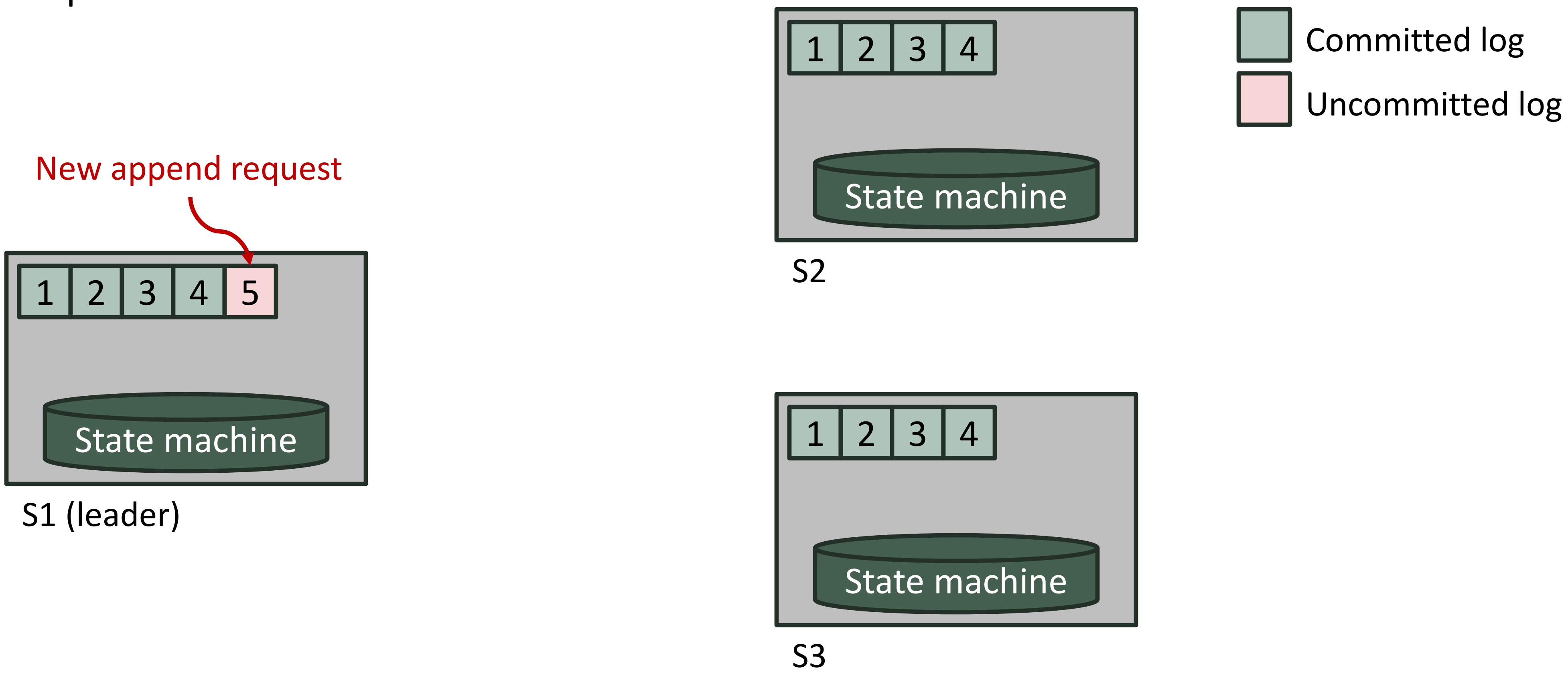
- Raft protocol overview




- Log: mutate operation
- State machine: back-end DB

# Log Store with Raft

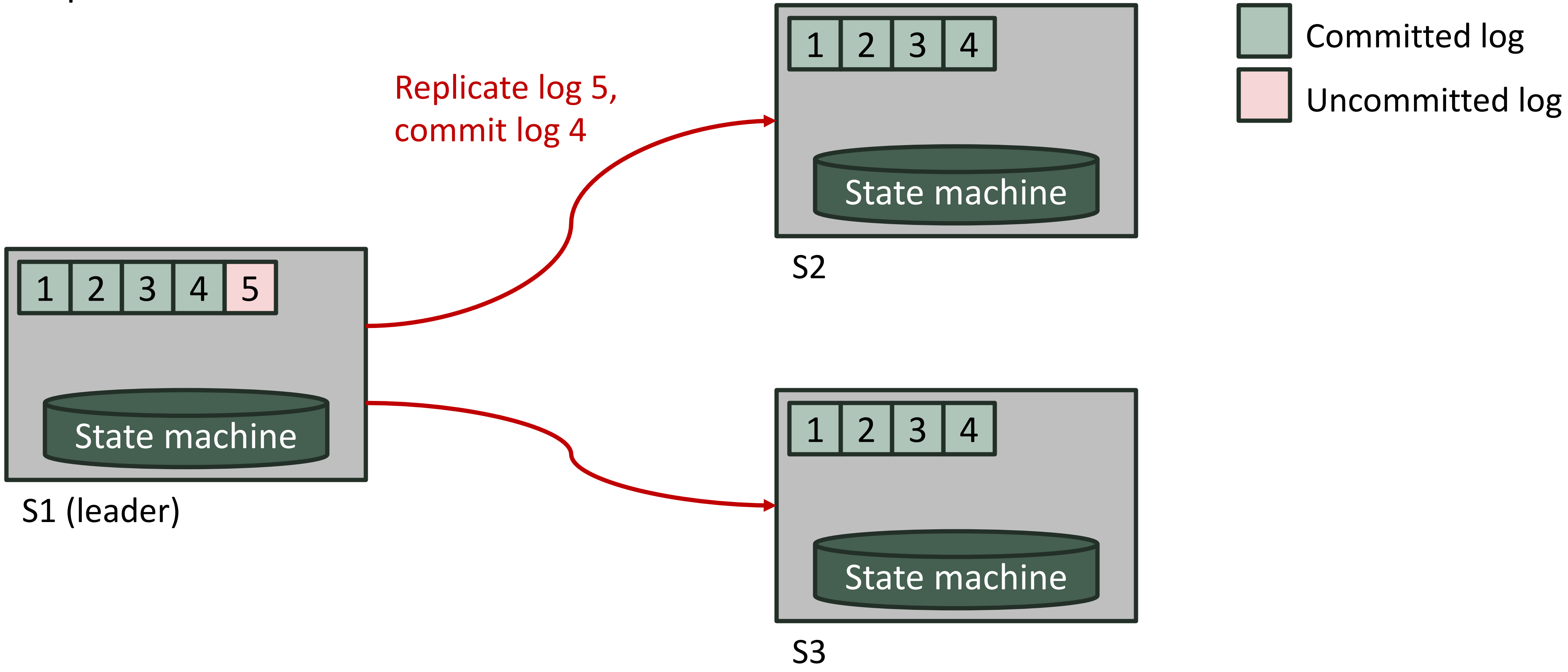
- Raft protocol overview




- Log: mutate operation
- State machine: back-end DB

# Log Store with Raft

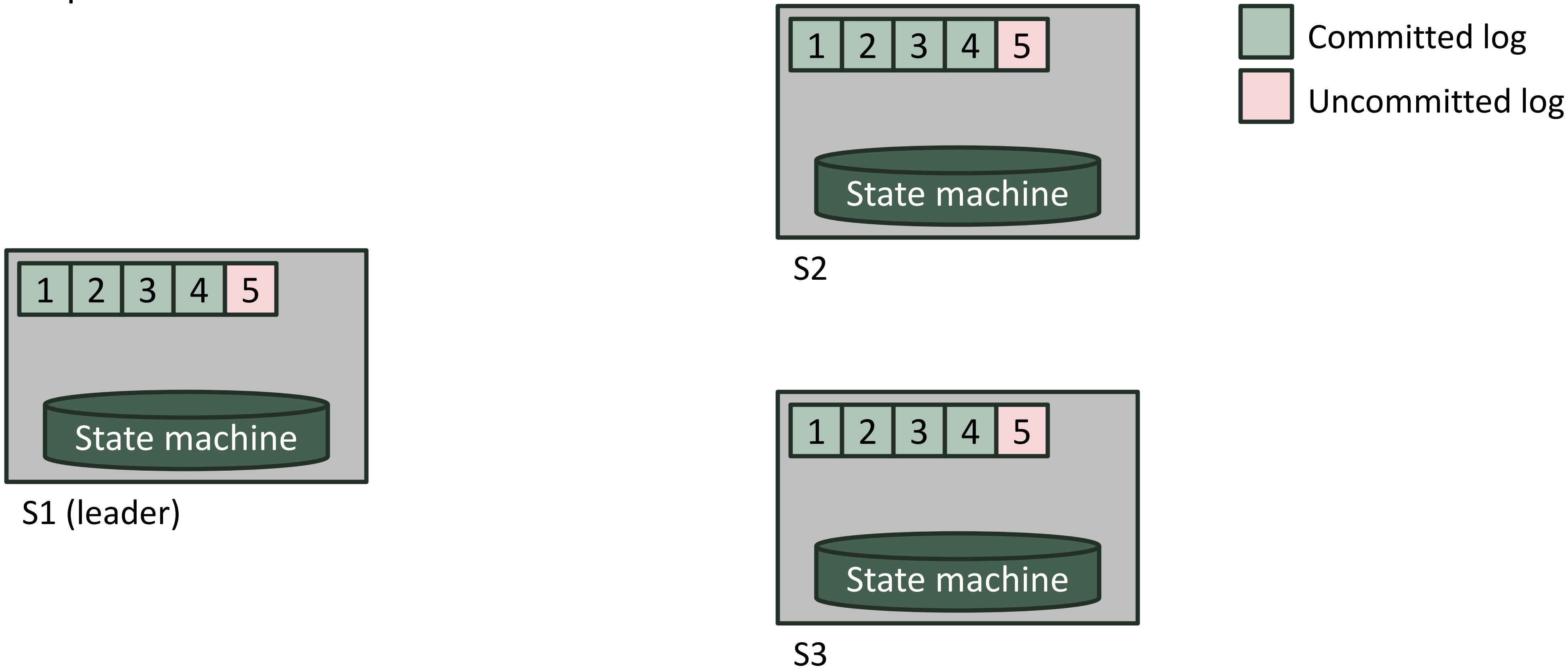
- Raft protocol overview




- Log: mutate operation
- State machine: back-end DB

# Log Store with Raft

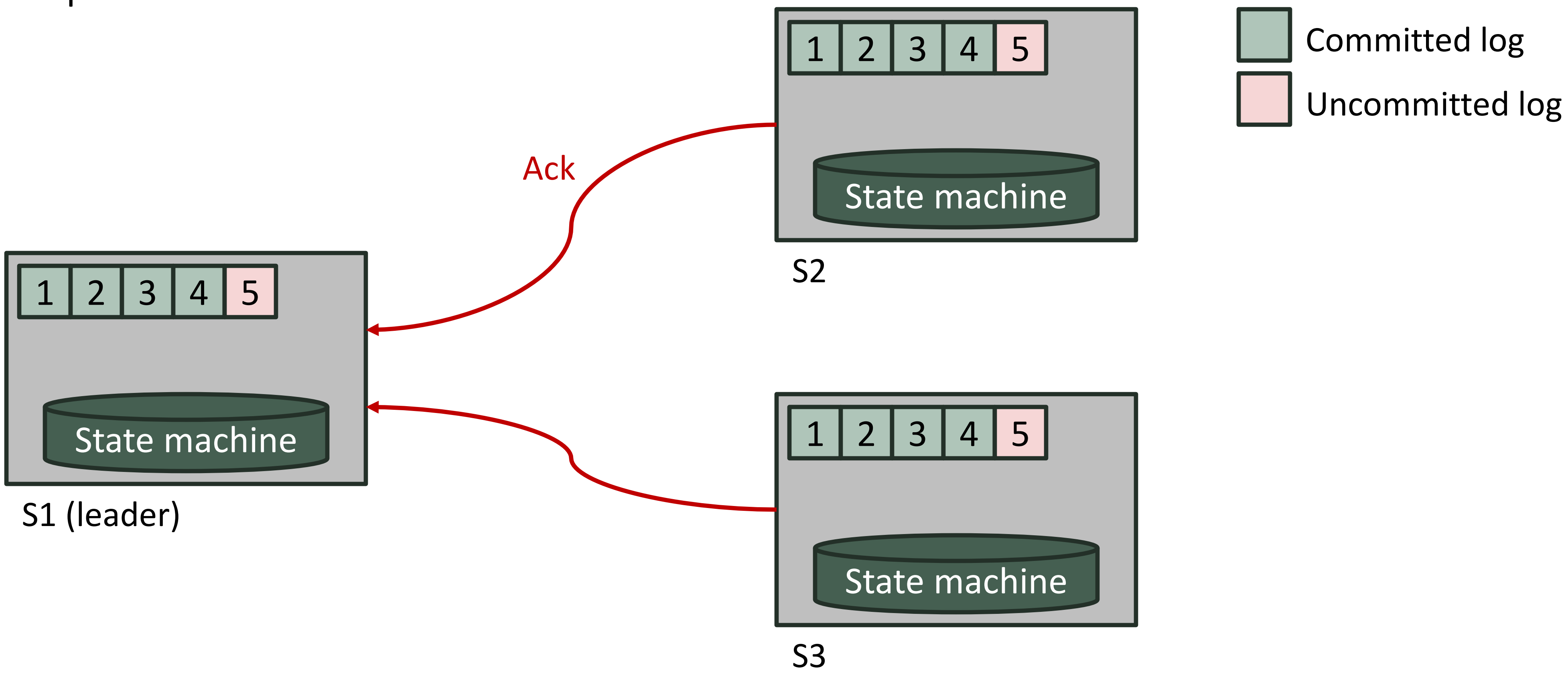
- Raft protocol overview




- Log: mutate operation
- State machine: back-end DB

# Log Store with Raft

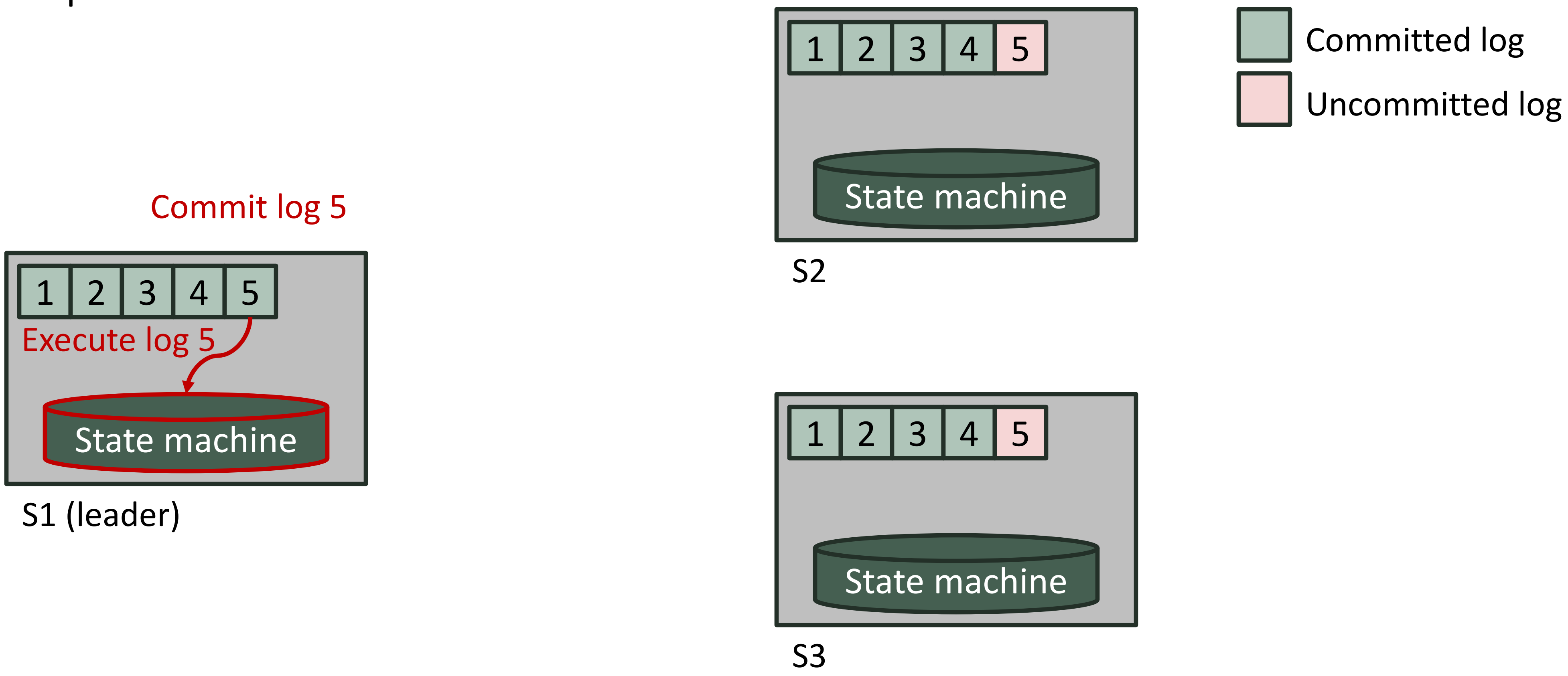
- Raft protocol overview




- Log: mutate operation
- State machine: back-end DB

# Log Store with Raft

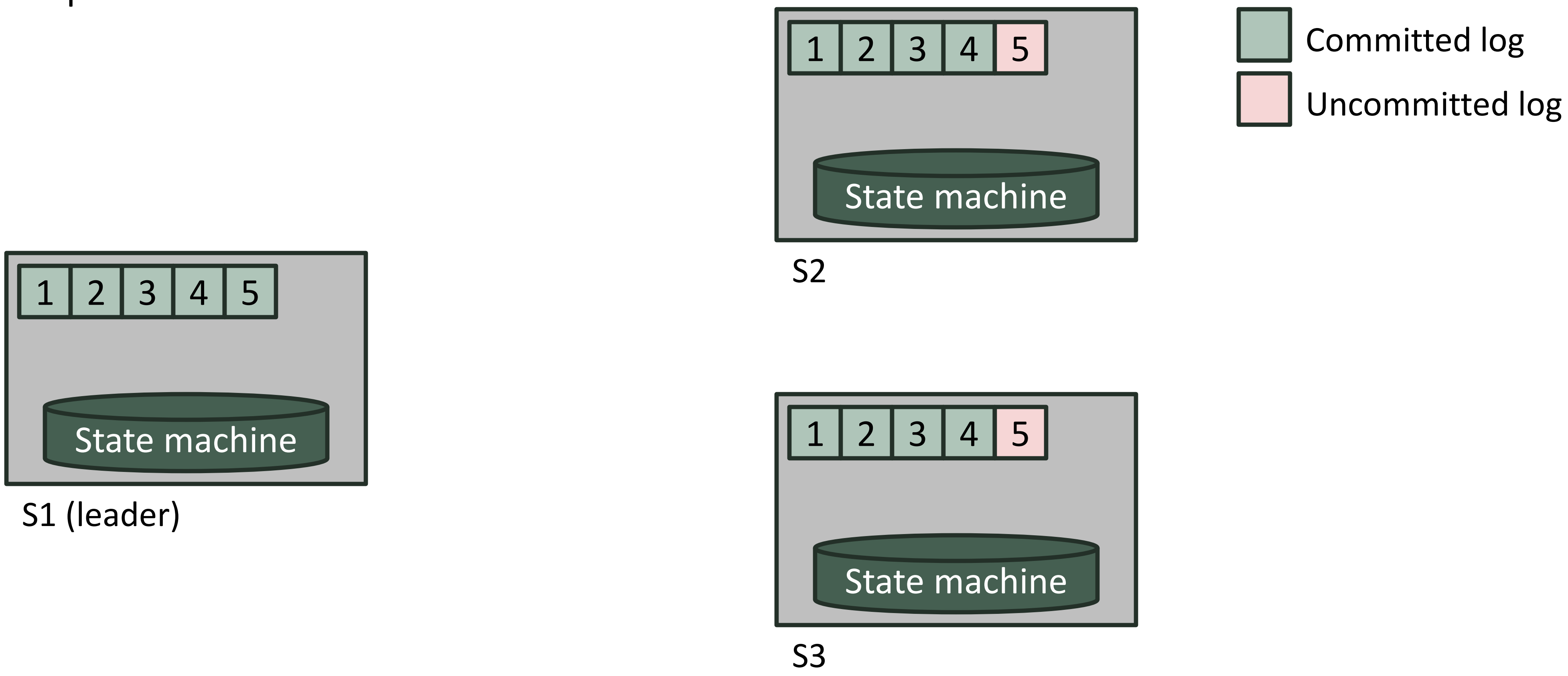
- Raft protocol overview




- Log: mutate operation
- State machine: back-end DB

# Log Store with Raft

- Raft protocol overview

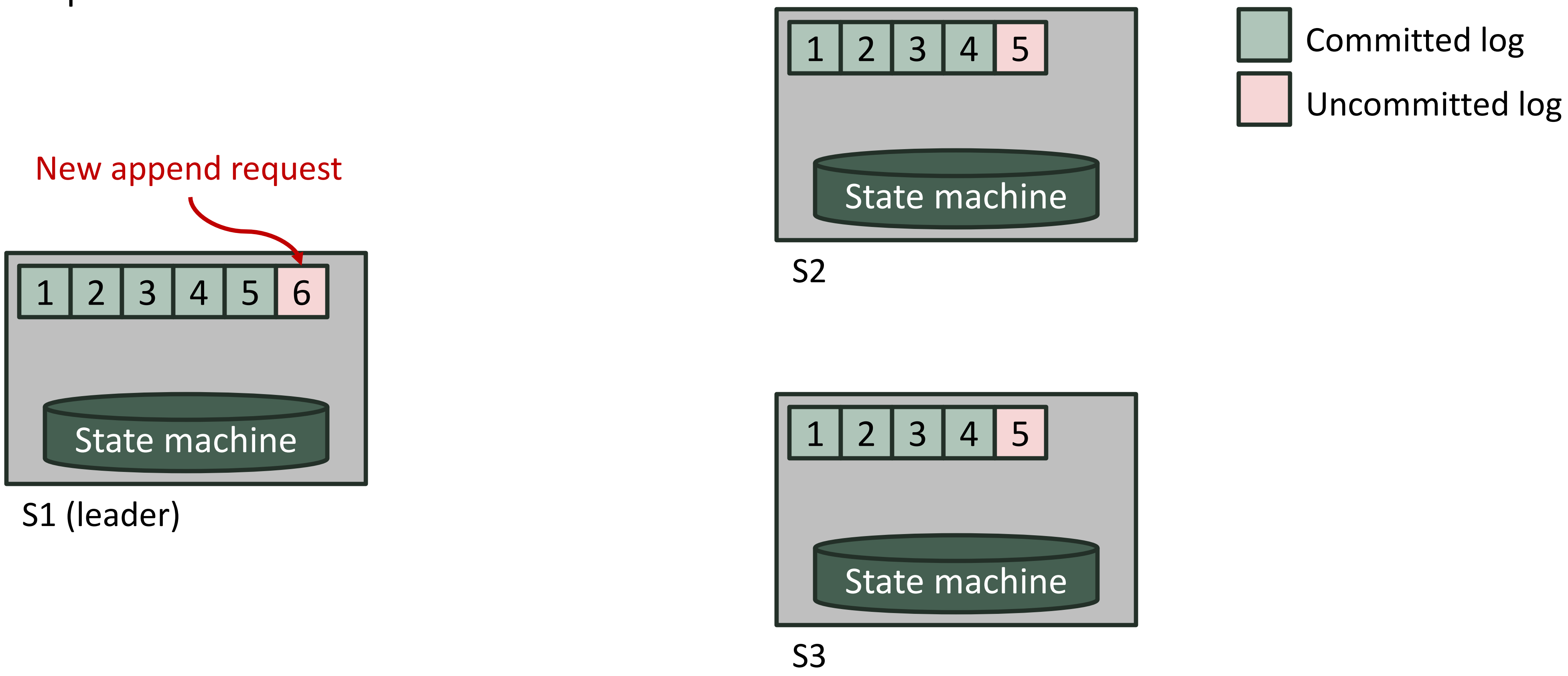



Commit of followers is done by
- Next replication (pipelining)
- Heartbeat (if no replication meanwhile)

- Log: mutate operation
- State machine: back-end DB

# Log Store with Raft

- Raft protocol overview

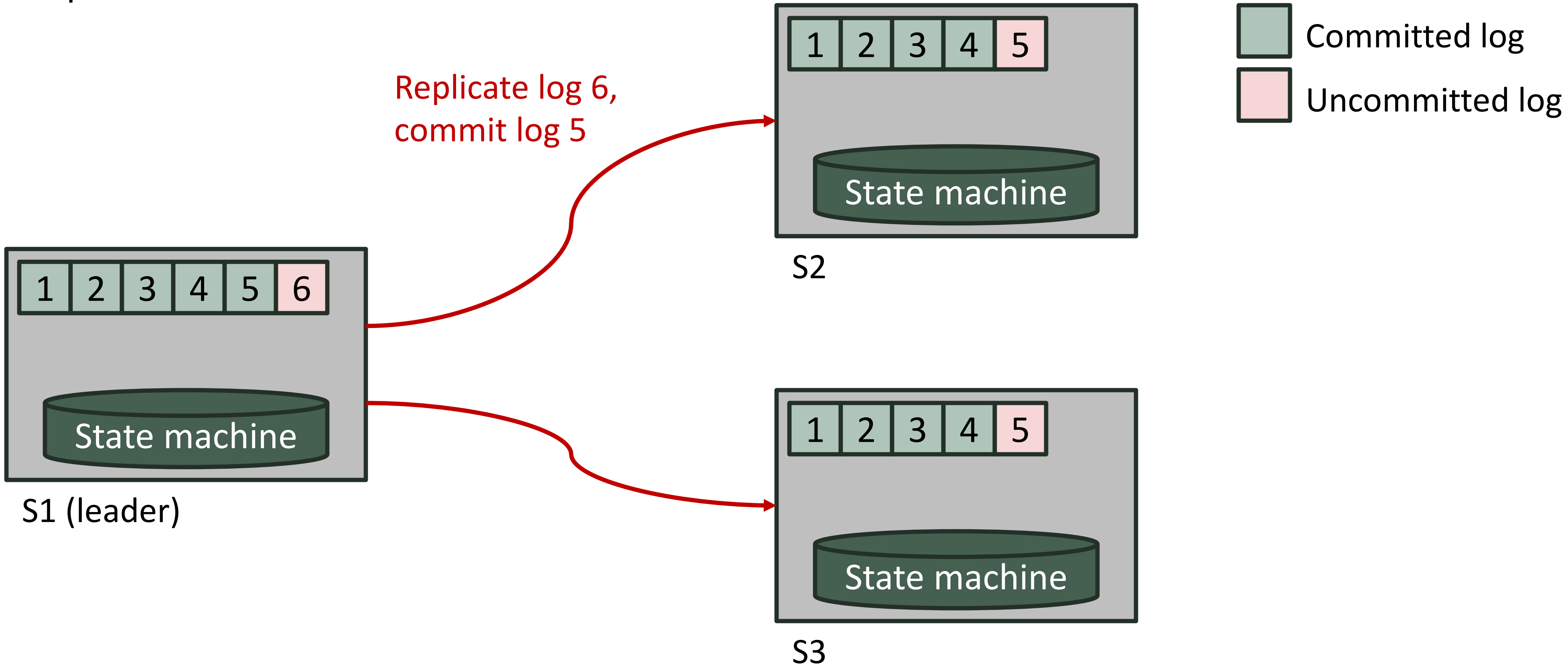



Commit of followers is done by
- Next replication (pipelining)
- Heartbeat (if no replication meanwhile)

- Log: mutate operation
- State machine: back-end DB

# Log Store with Raft

- Raft protocol overview



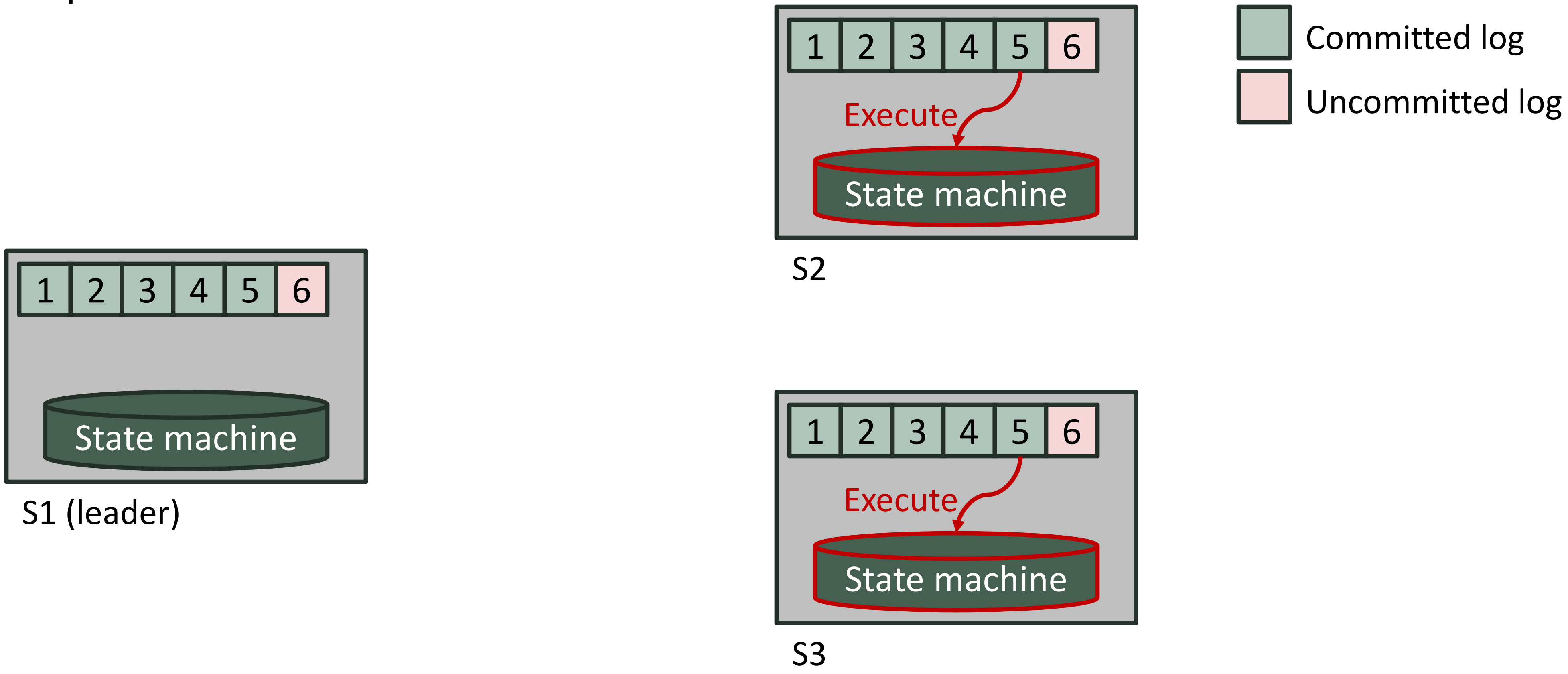

Commit of followers is done by

- Next replication (pipelining)
- Heartbeat (if no replication meanwhile)

- Log: mutate operation
- State machine: back-end DB

# Log Store with Raft

- Raft protocol overview



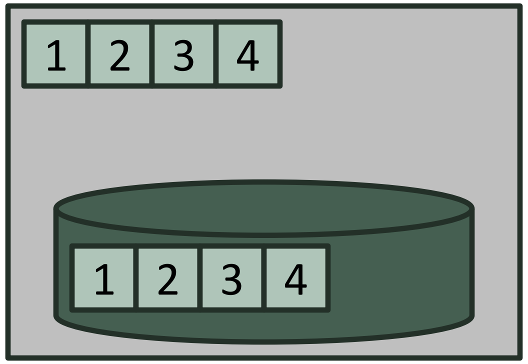

Commit of followers is done by

- Next replication (pipelining)
- Heartbeat (if no replication meanwhile)

- Log: mutate operation
- State machine: back-end DB

# Log Store with Raft

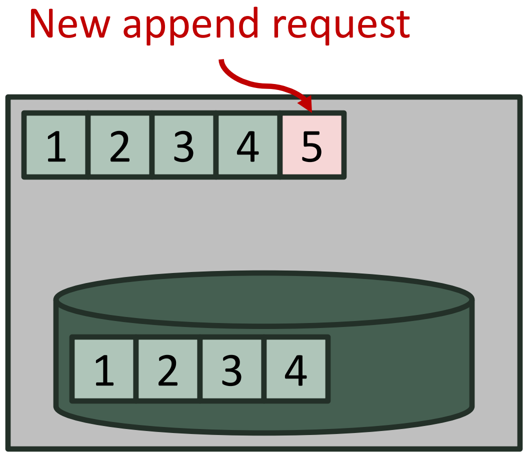
- Option 1: log store based on Raft
  - State machine: another append-only logs




S1 (leader)

# Log Store with Raft

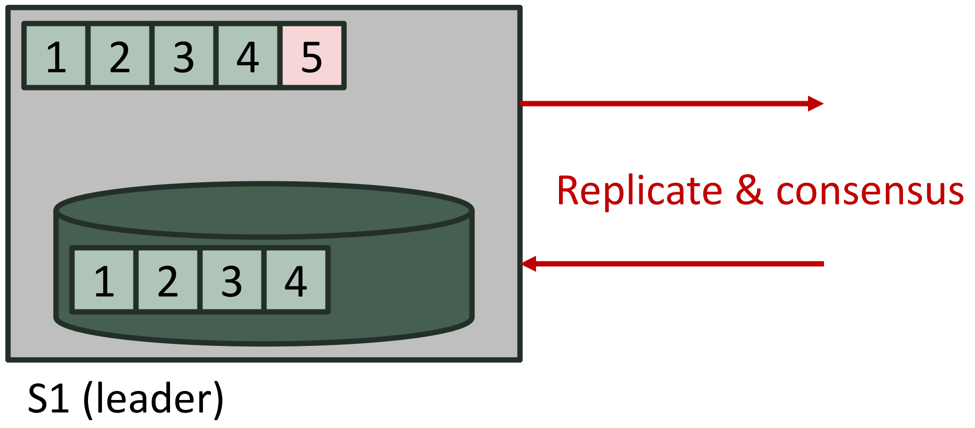
- Option 1: log store based on Raft
  - State machine: another append-only logs




S1 (leader)

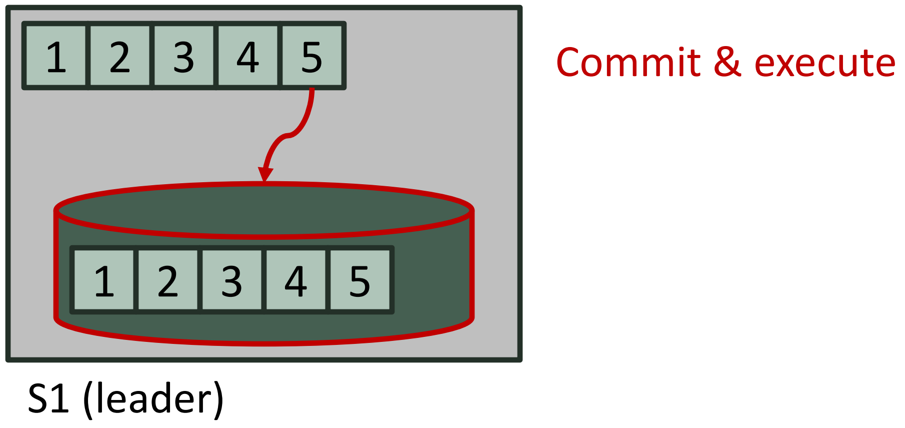
# Log Store with Raft

- Option 1: log store based on Raft
  - State machine: another append-only logs



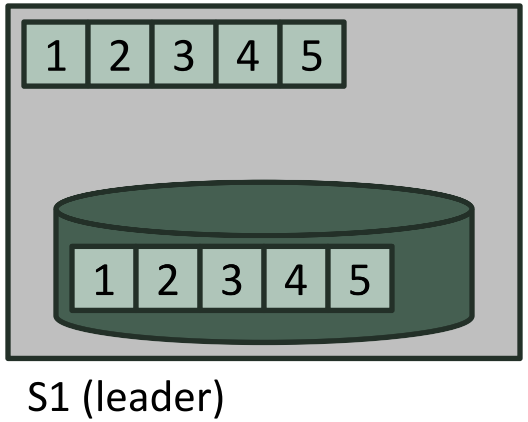
# Log Store with Raft

- Option 1: log store based on Raft
  - State machine: another append-only logs



# Log Store with Raft

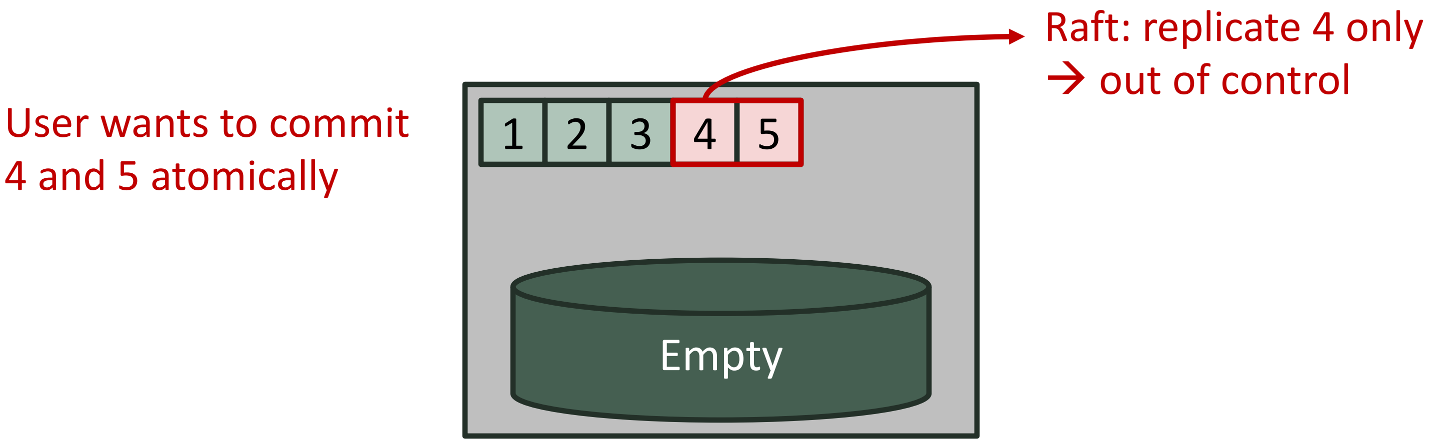
- Option 1: log store based on Raft
  - State machine: another append-only logs




Exactly duplicate logs!
- Double the space
- Double disk writes

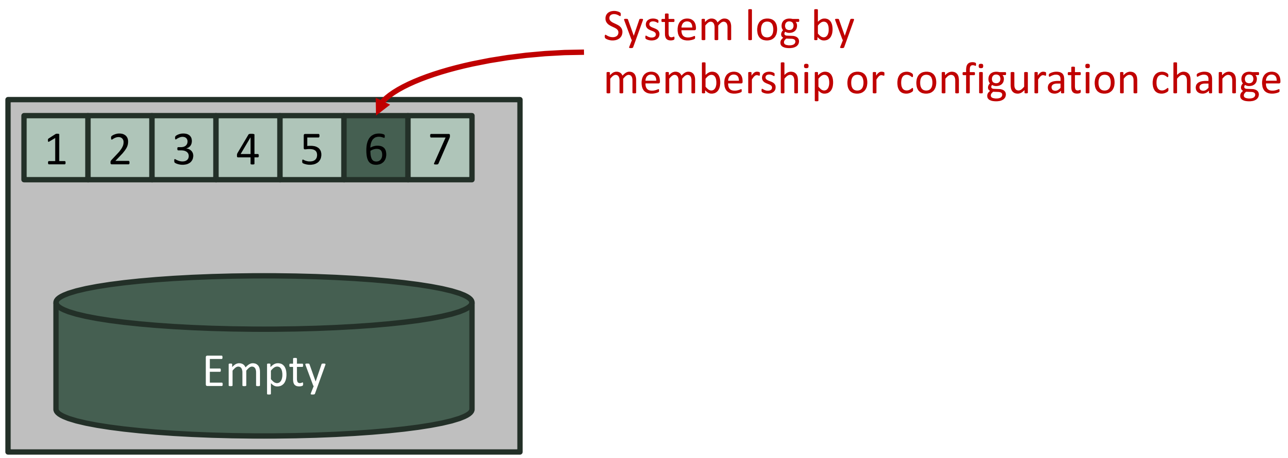
# Log Store with Raft

- Option 1: log store based on Raft
  - State machine: another append-only logs
- Option 2: directly using Raft logs as actual logs
  - LSN == Raft log number
  - Granularity difference
    - Cannot guarantee atomic commit of a batch
    - Basic unit of replication, consensus, recovery, and commit: a Raft log



# Log Store with Raft

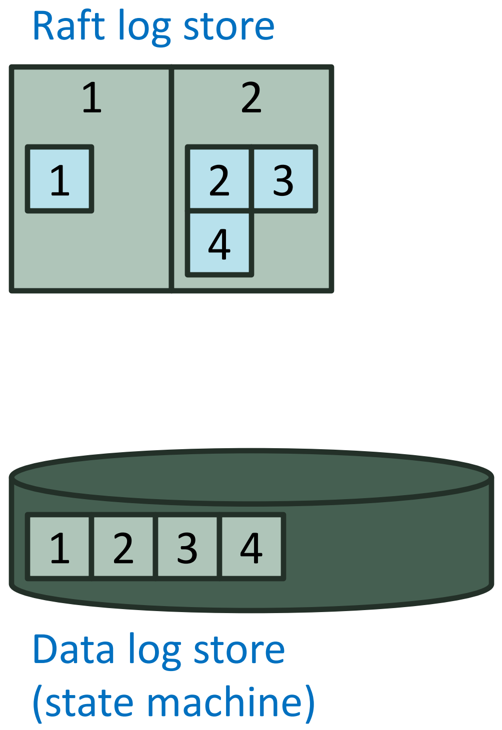
- Option 1: log store based on Raft
  - State machine: another append-only logs
- Option 2: directly using Raft logs as actual logs
  - LSN == Raft log number
  - Granularity difference
    - Cannot guarantee atomic commit of a batch
    - Basic unit of replication, consensus, recovery, and commit: a Raft log
  - System logs in the middle
    - Not continuous LSN



## Our Approach:
# Log Sharing Scheme

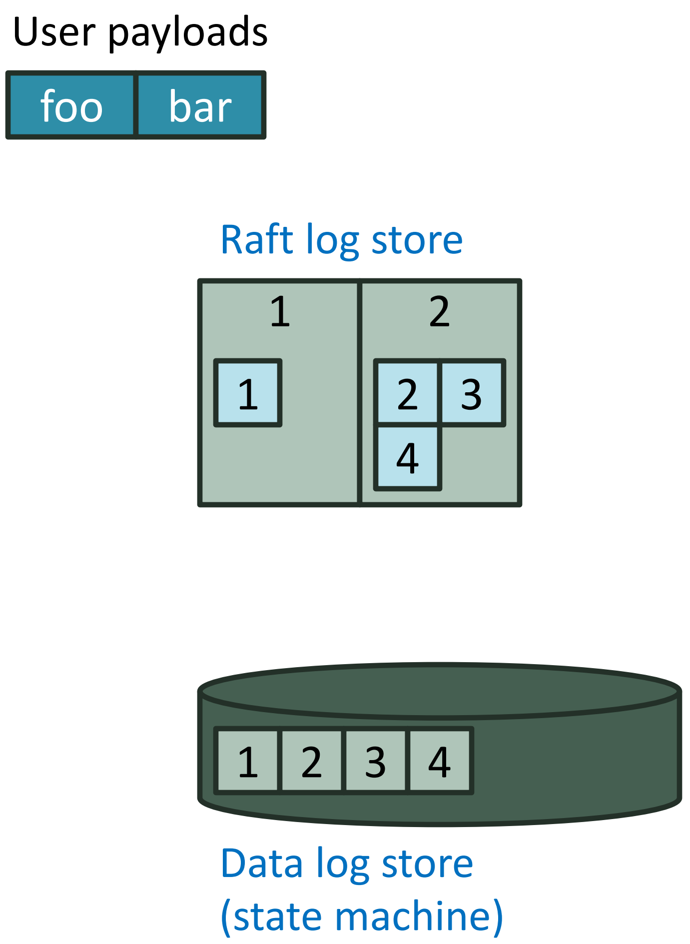
# Log Sharing Scheme

- Data log store (state machine)
  - Stores user payloads
  - Assigns LSN to each payload
- Raft log store
  - Stores Raft logs
    - References to data log store
- Each payload is written to disk only once
- Granularity difference
  - Raft log: multiple references to data log



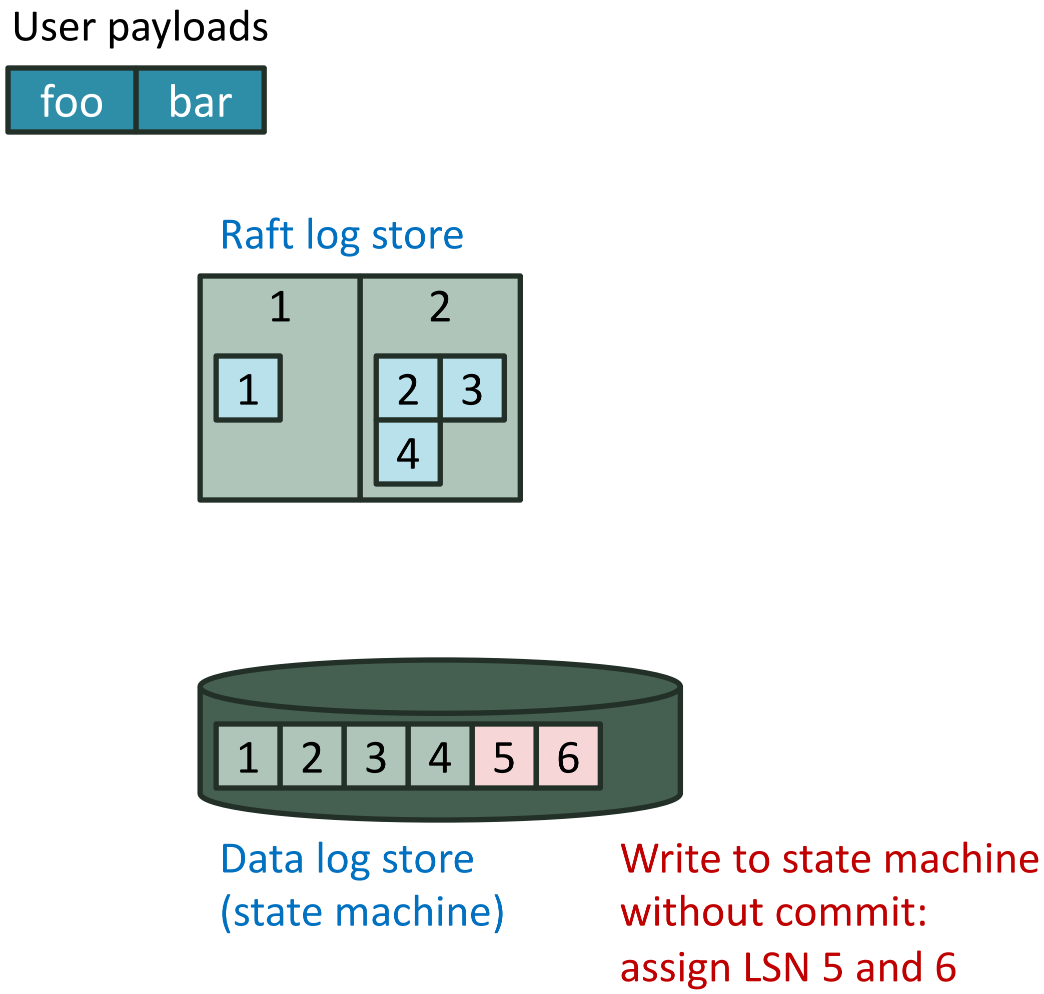
# Log Sharing Scheme

- Data log store (state machine)
  - Stores user payloads
  - Assigns LSN to each payload
- Raft log store
  - Stores Raft logs
    - References to data log store
- Each payload is written to disk only once
- Granularity difference
  - Raft log: multiple references to data log



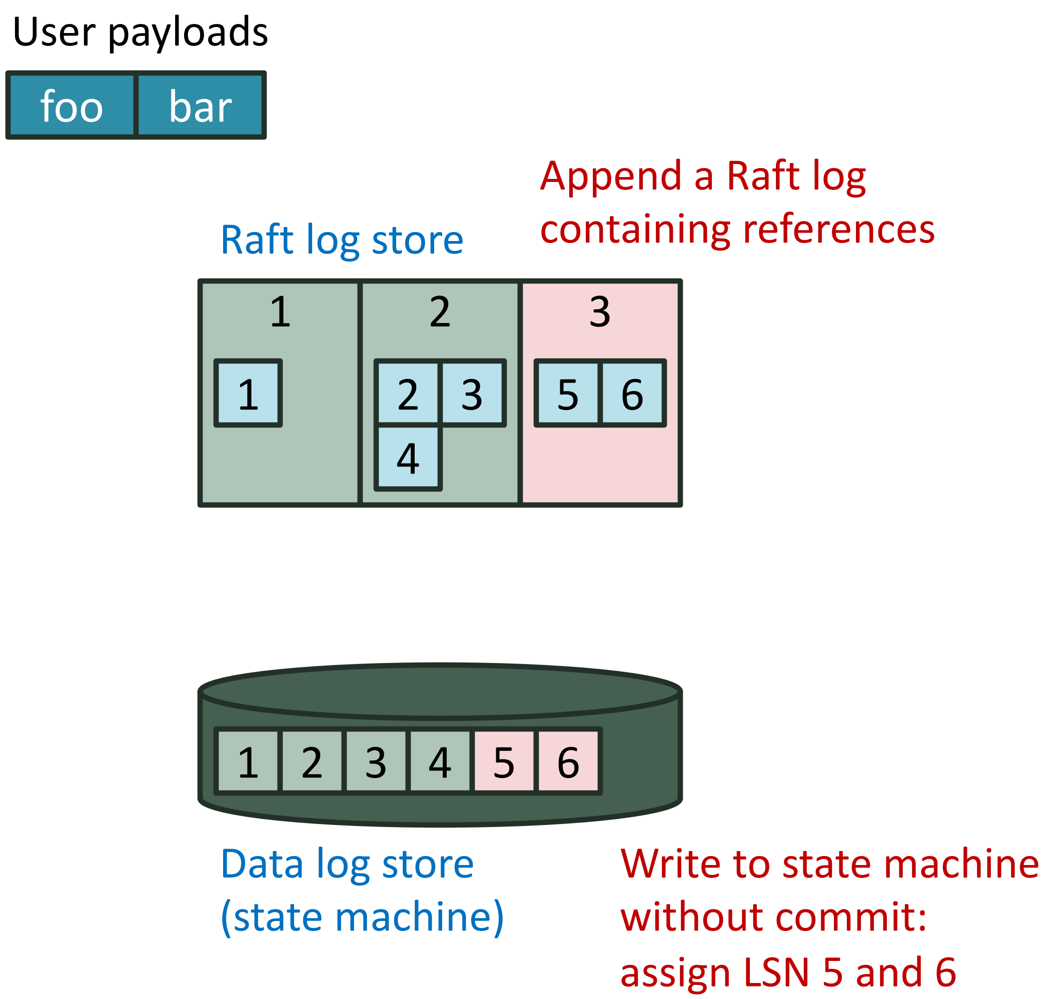
# Log Sharing Scheme

- Data log store (state machine)
  - Stores user payloads
  - Assigns LSN to each payload
- Raft log store
  - Stores Raft logs
    - References to data log store
- Each payload is written to disk only once
- Granularity difference
  - Raft log: multiple references to data log



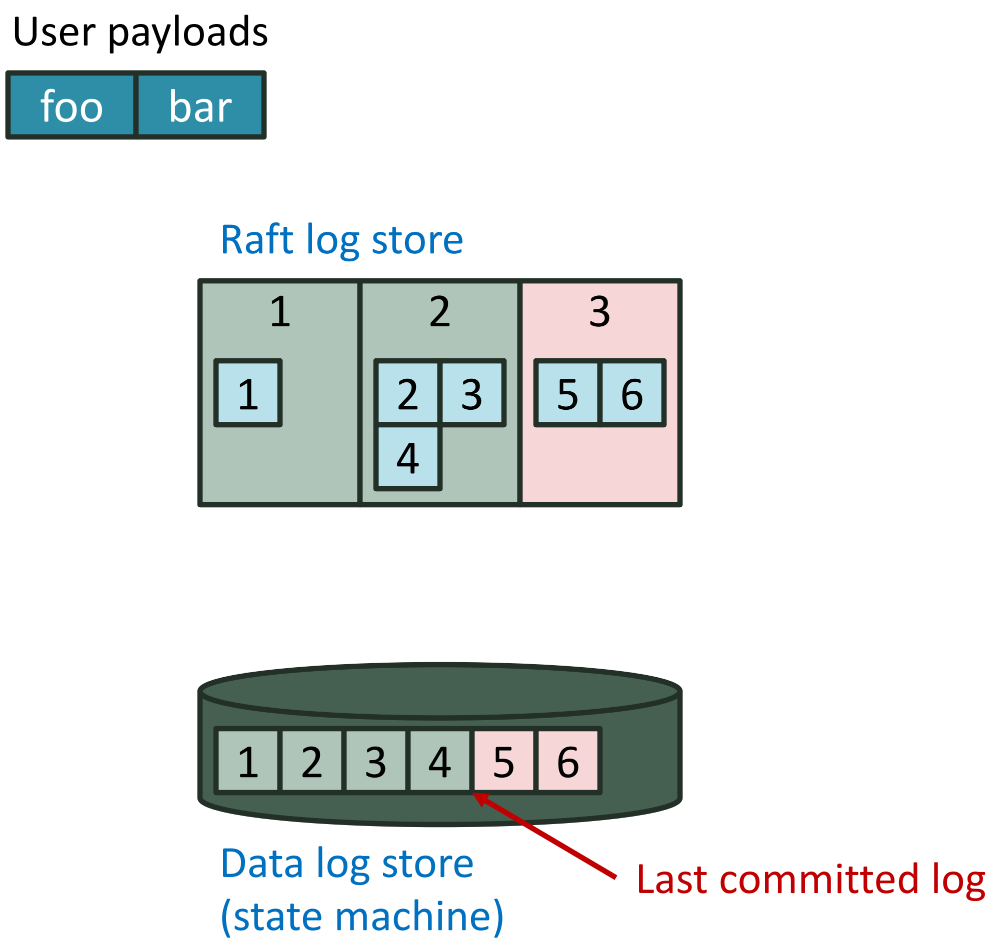
# Log Sharing Scheme

- Data log store (state machine)
  - Stores user payloads
  - Assigns LSN to each payload
- Raft log store
  - Stores Raft logs
    - References to data log store
- Each payload is written to disk only once
- Granularity difference
  - Raft log: multiple references to data log



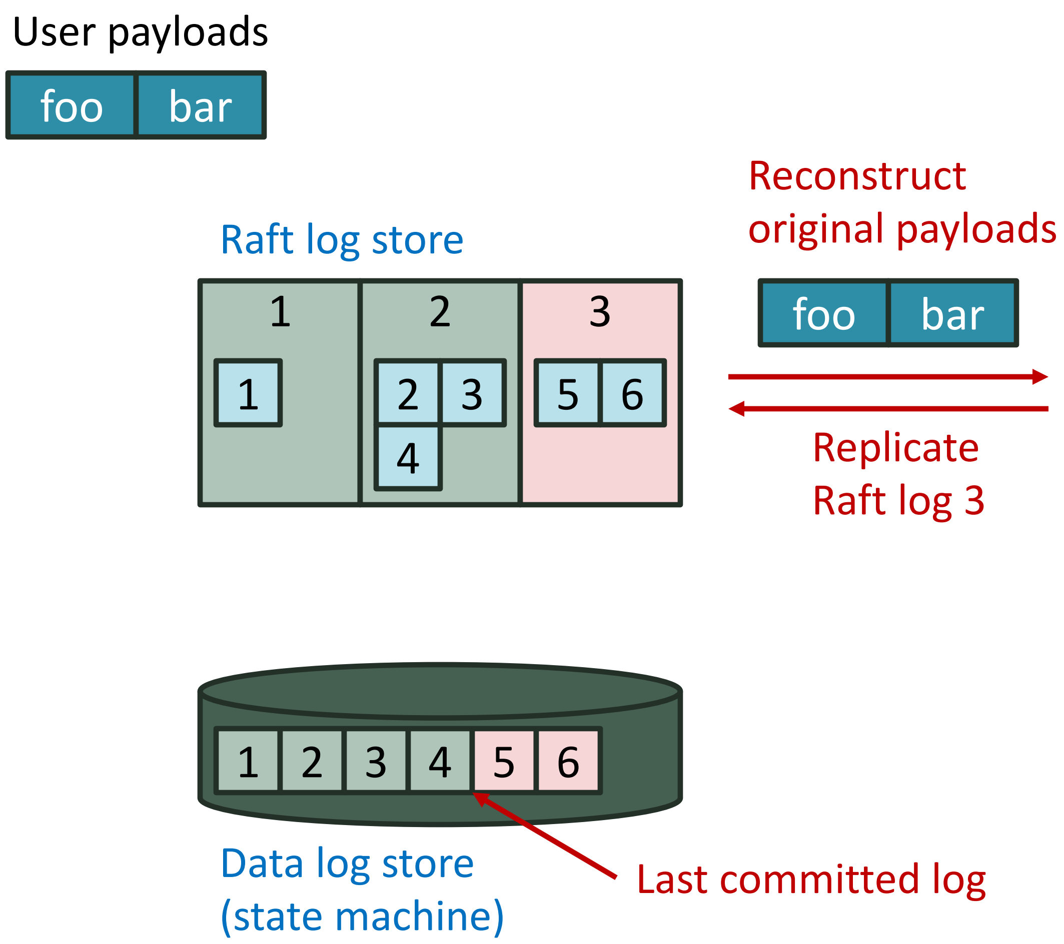
# Log Sharing Scheme

- Data log store (state machine)
  - Stores user payloads
  - Assigns LSN to each payload
- Raft log store
  - Stores Raft logs
    - References to data log store
- Each payload is written to disk only once
- Granularity difference
  - Raft log: multiple references to data log



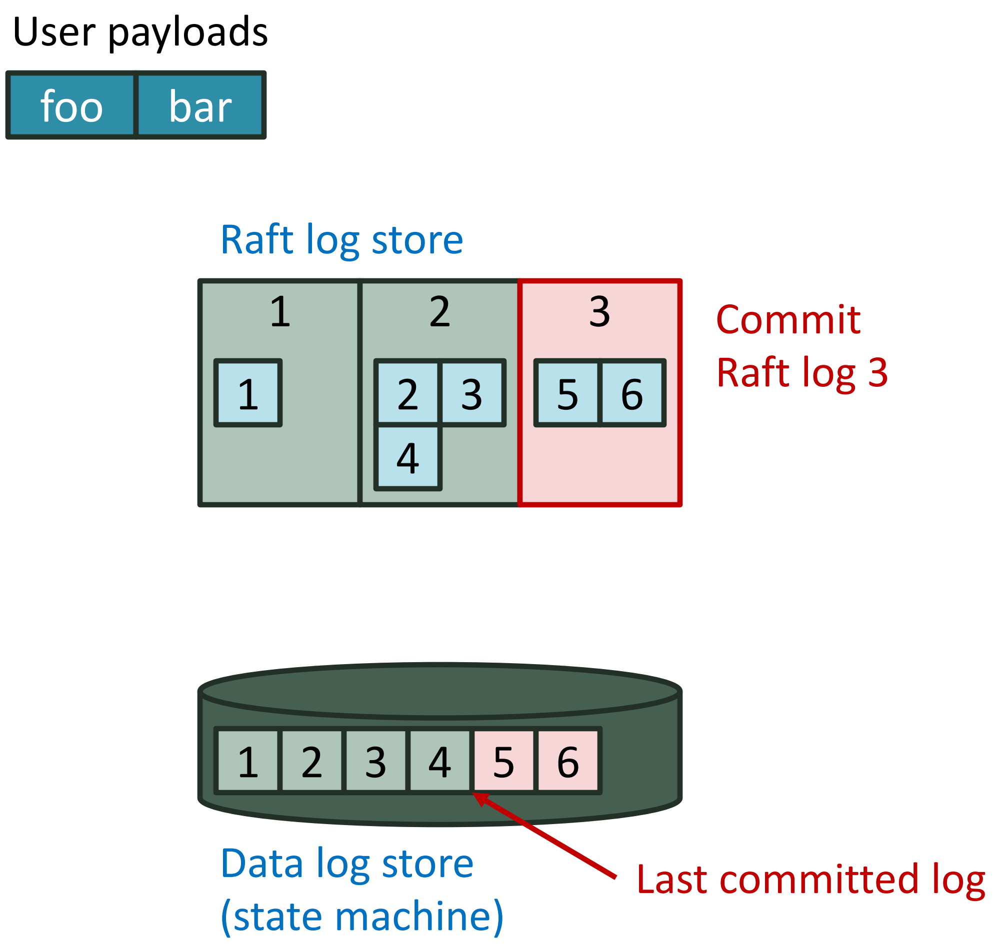
# Log Sharing Scheme

- Data log store (state machine)
  - Stores user payloads
  - Assigns LSN to each payload
- Raft log store
  - Stores Raft logs
    - References to data log store
- Each payload is written to disk only once
- Granularity difference
  - Raft log: multiple references to data log



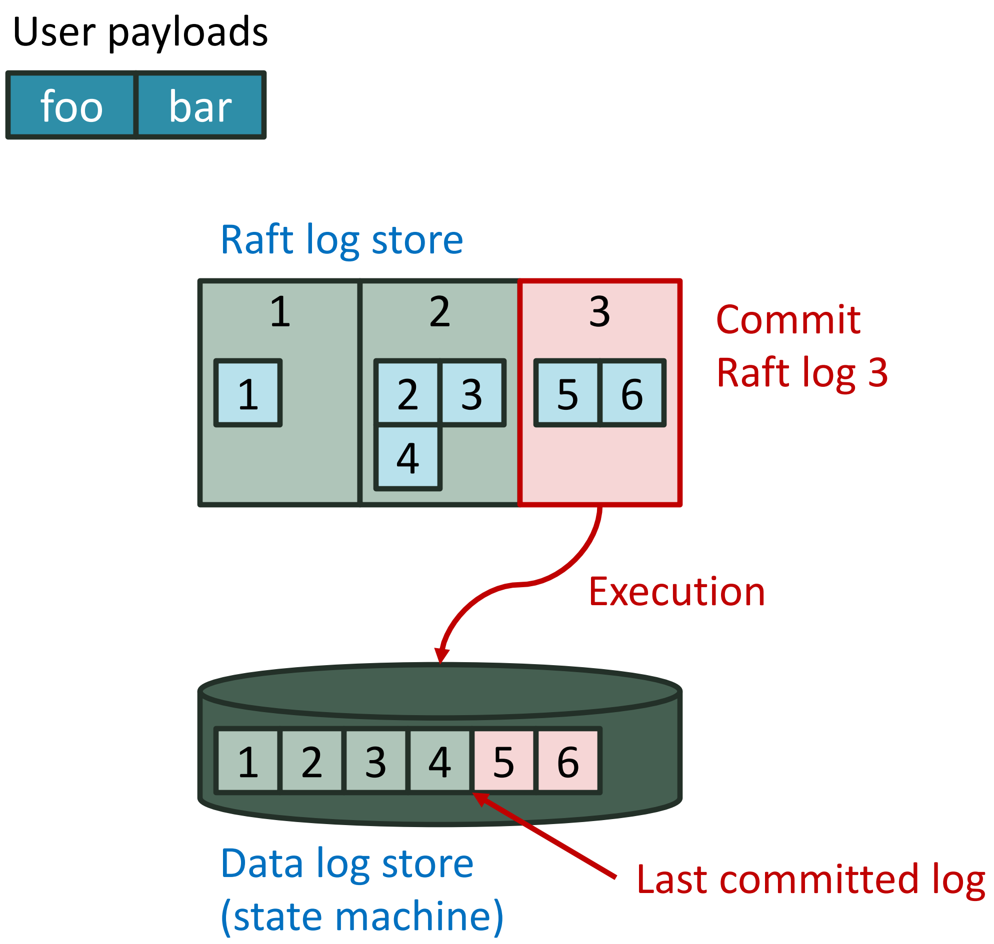
# Log Sharing Scheme

- Data log store (state machine)
  - Stores user payloads
  - Assigns LSN to each payload
- Raft log store
  - Stores Raft logs
    - References to data log store
- Each payload is written to disk only once
- Granularity difference
  - Raft log: multiple references to data log



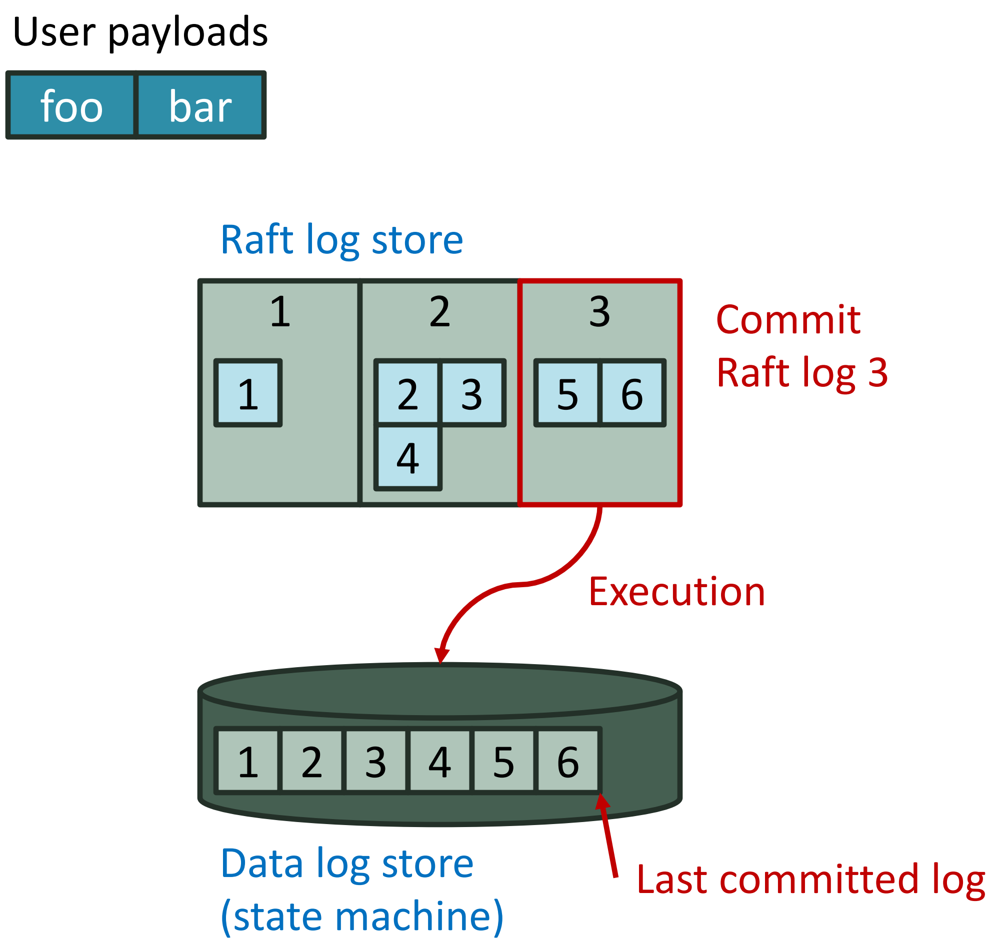
# Log Sharing Scheme

- Data log store (state machine)
  - Stores user payloads
  - Assigns LSN to each payload
- Raft log store
  - Stores Raft logs
    - References to data log store
- Each payload is written to disk only once
- Granularity difference
  - Raft log: multiple references to data log



# Log Sharing Scheme

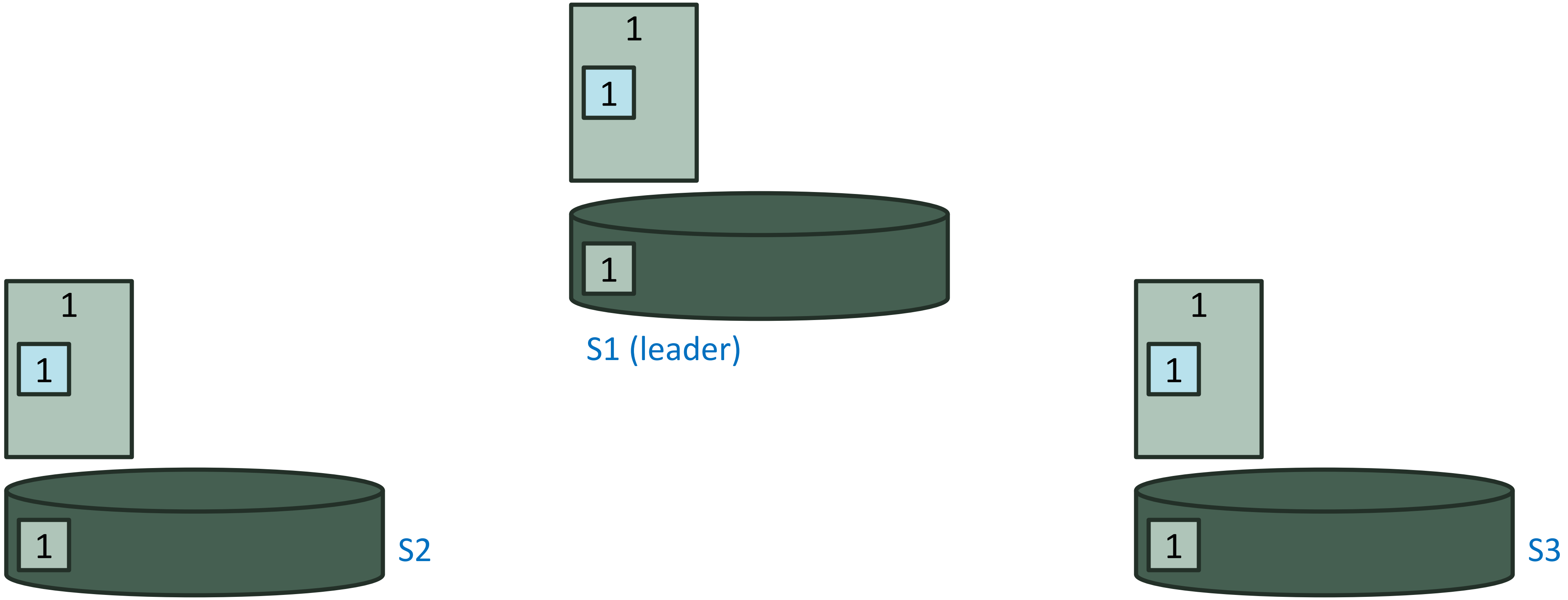
- Data log store (state machine)
  - Stores user payloads
  - Assigns LSN to each payload
- Raft log store
  - Stores Raft logs
    - References to data log store
- Each payload is written to disk only once
- Granularity difference
  - Raft log: multiple references to data log




Commit process should be atomic

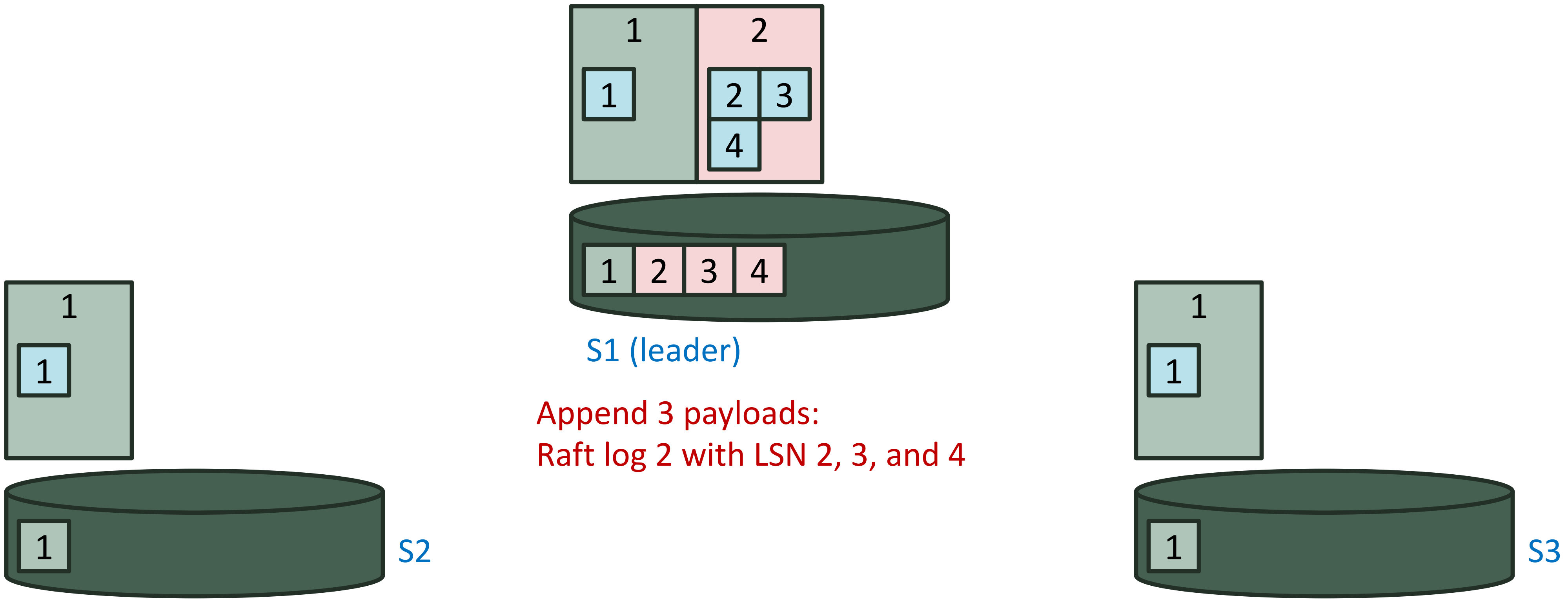
# Log Sharing Scheme

- Payloads are written to state machine before commit
  - Inconsistency?



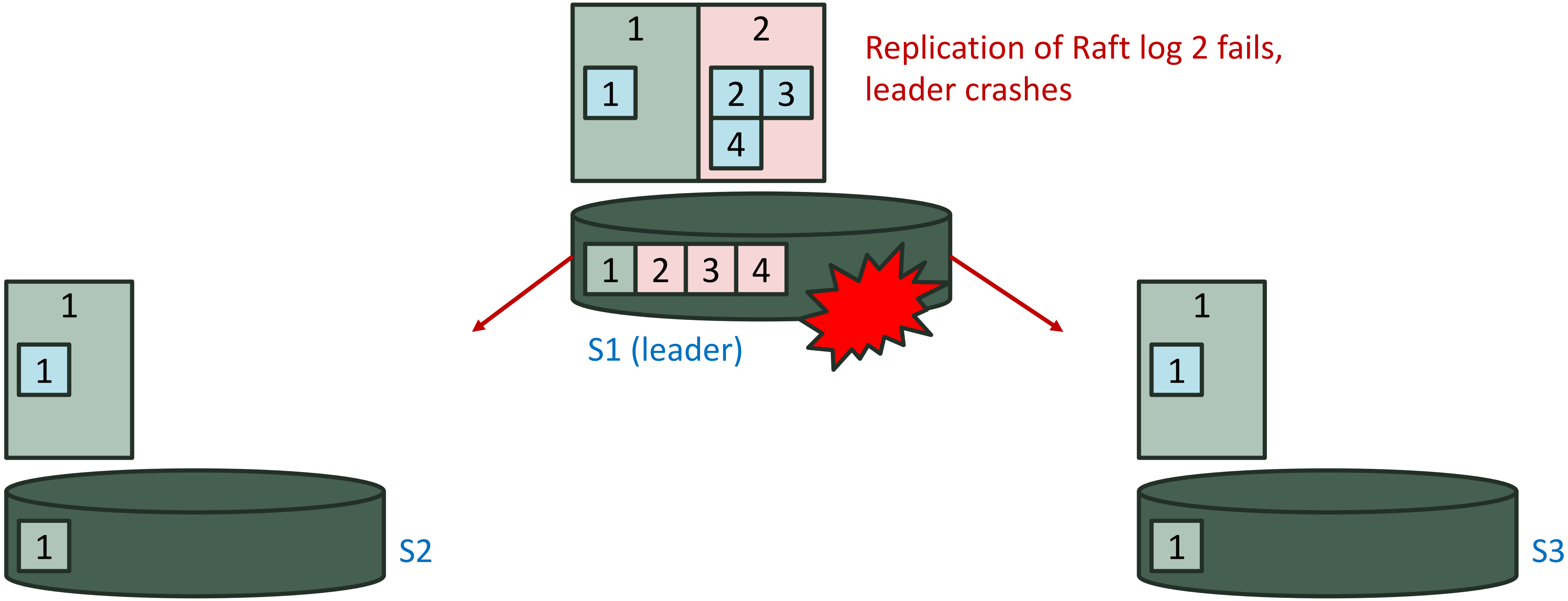
# Log Sharing Scheme

- Payloads are written to state machine before commit
  - Inconsistency?



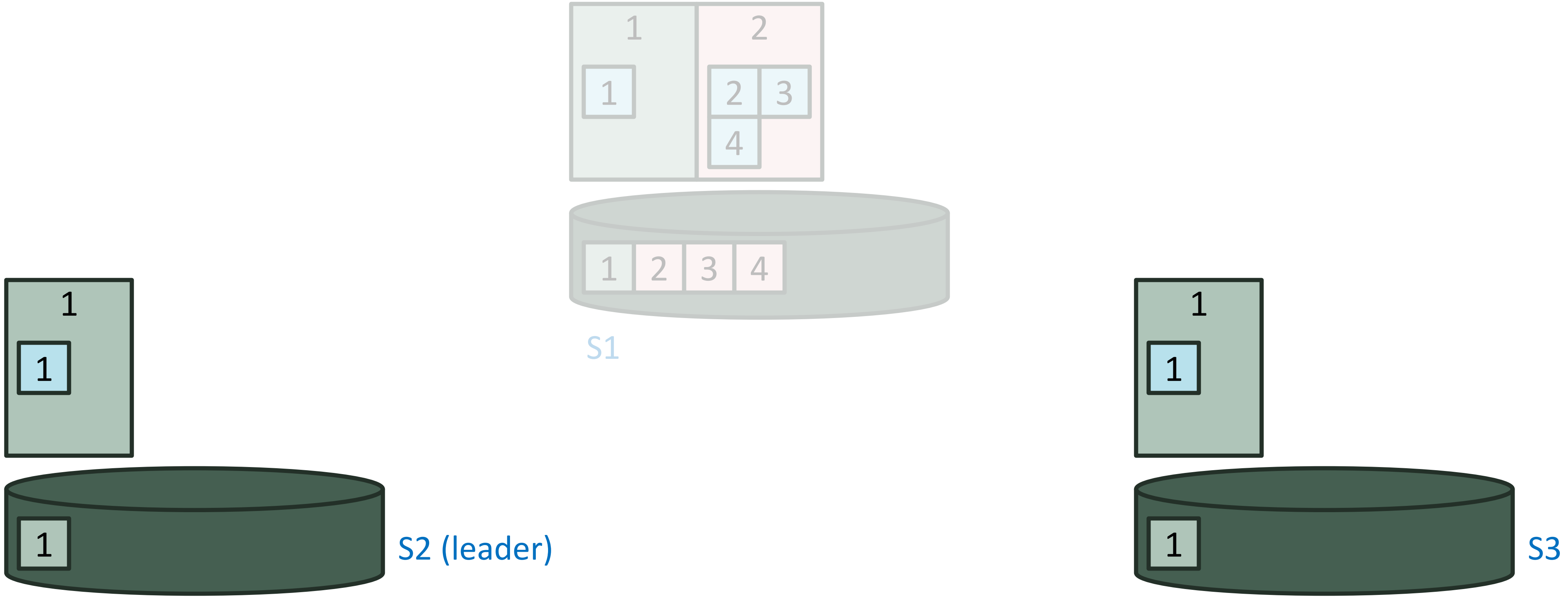
# Log Sharing Scheme

- Payloads are written to state machine before commit
  - Inconsistency?



# Log Sharing Scheme

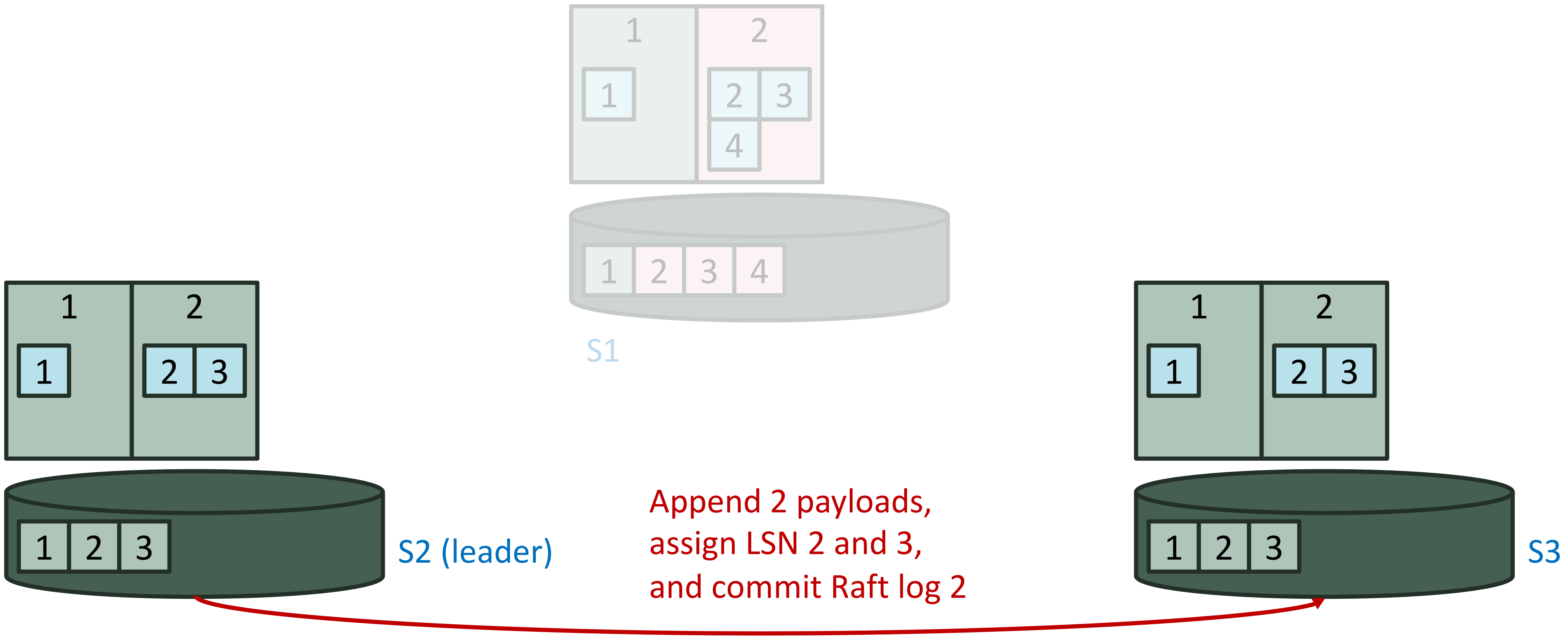
- Payloads are written to state machine before commit
  - Inconsistency?




S2 is elected as a new leader

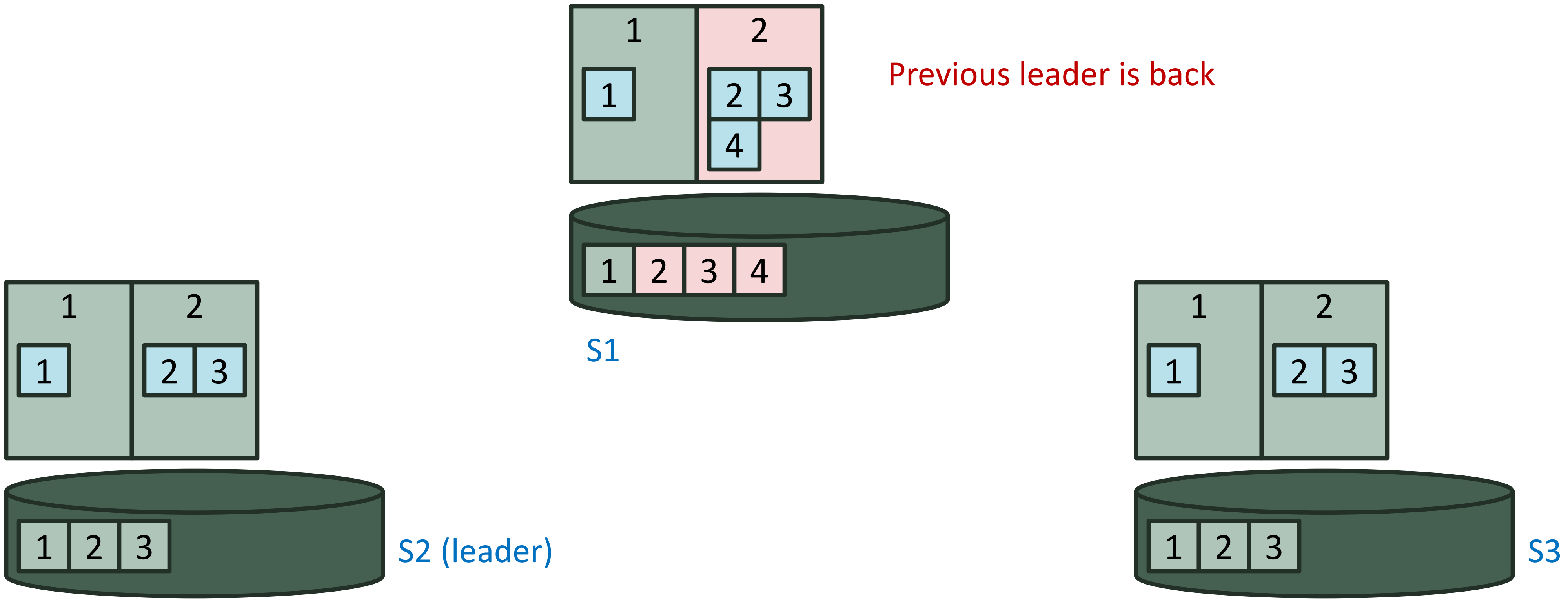
# Log Sharing Scheme

- Payloads are written to state machine before commit
  - Inconsistency?



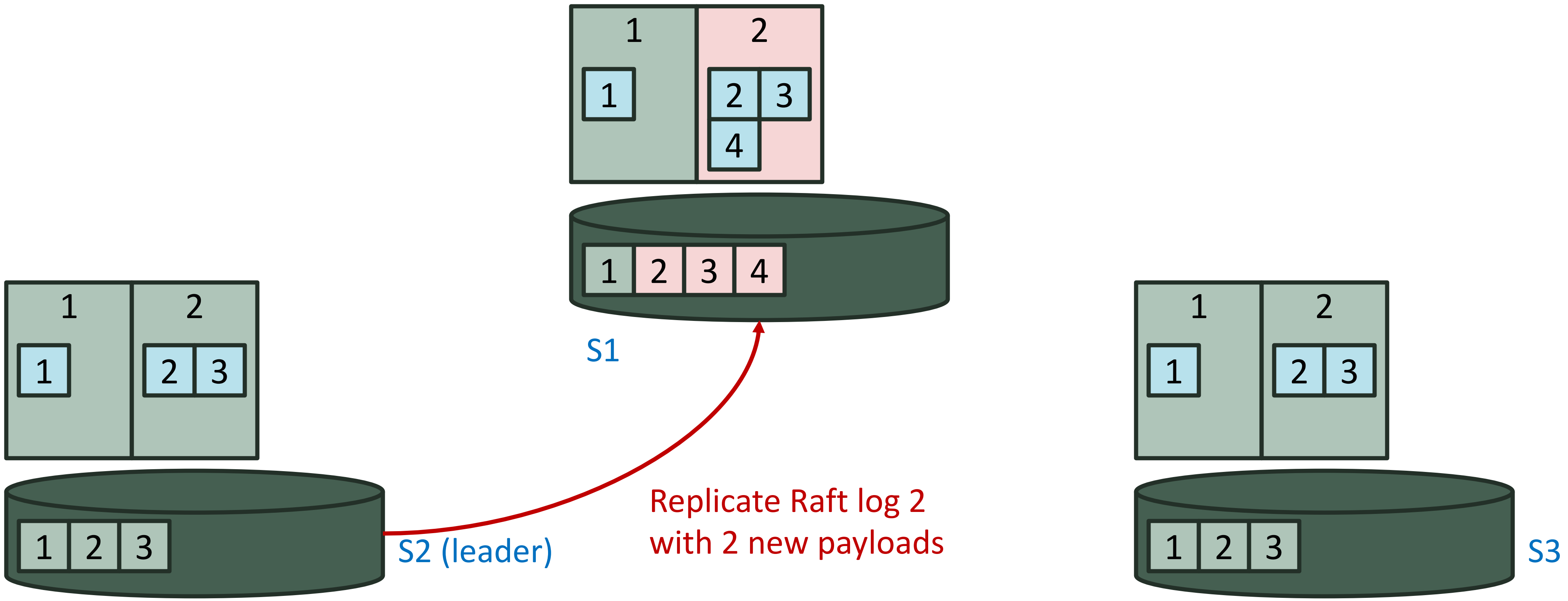
# Log Sharing Scheme

- Payloads are written to state machine before commit
  - Inconsistency?



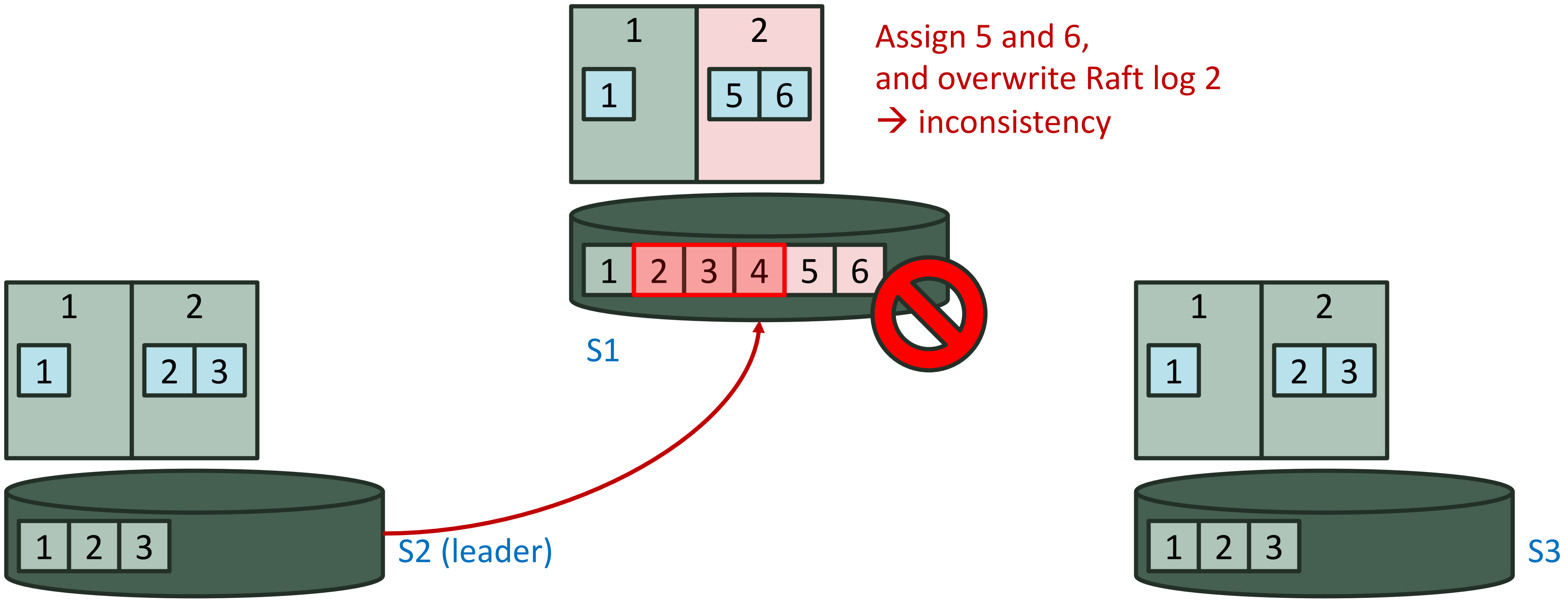
# Log Sharing Scheme

- Payloads are written to state machine before commit
  - Inconsistency?



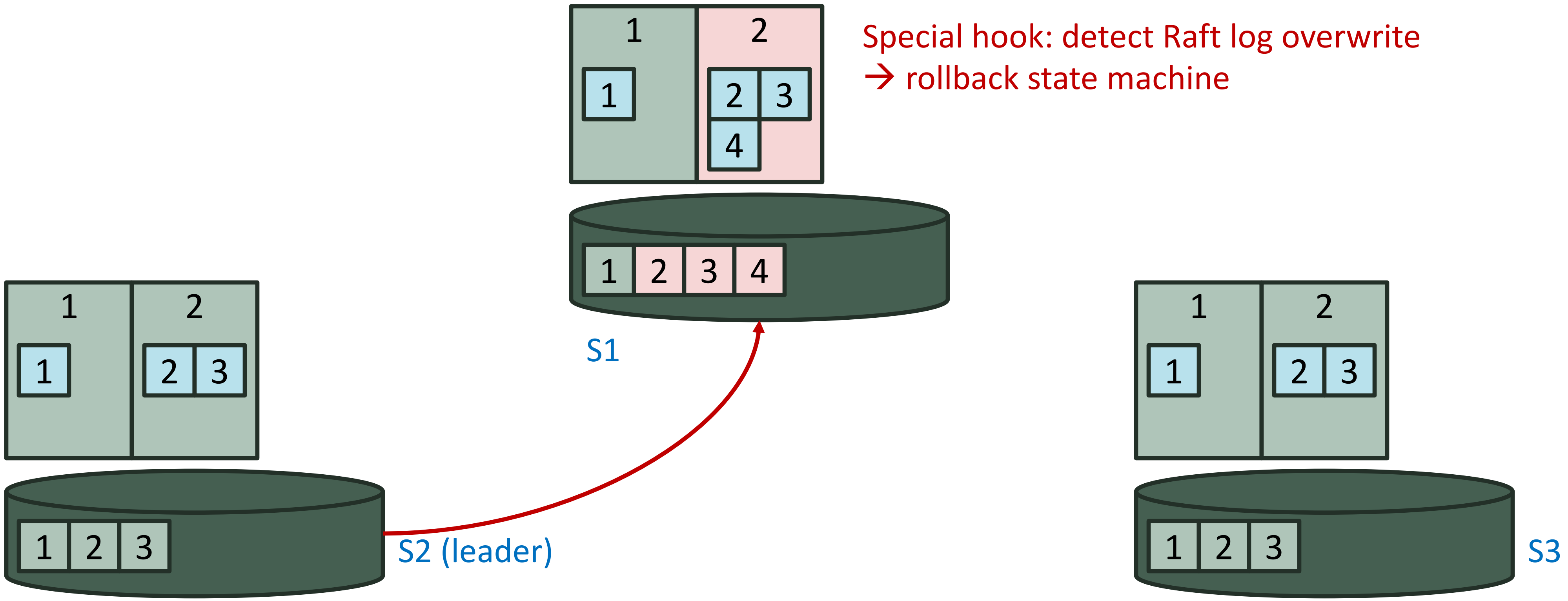
# Log Sharing Scheme

- Payloads are written to state machine before commit
  - Inconsistency?



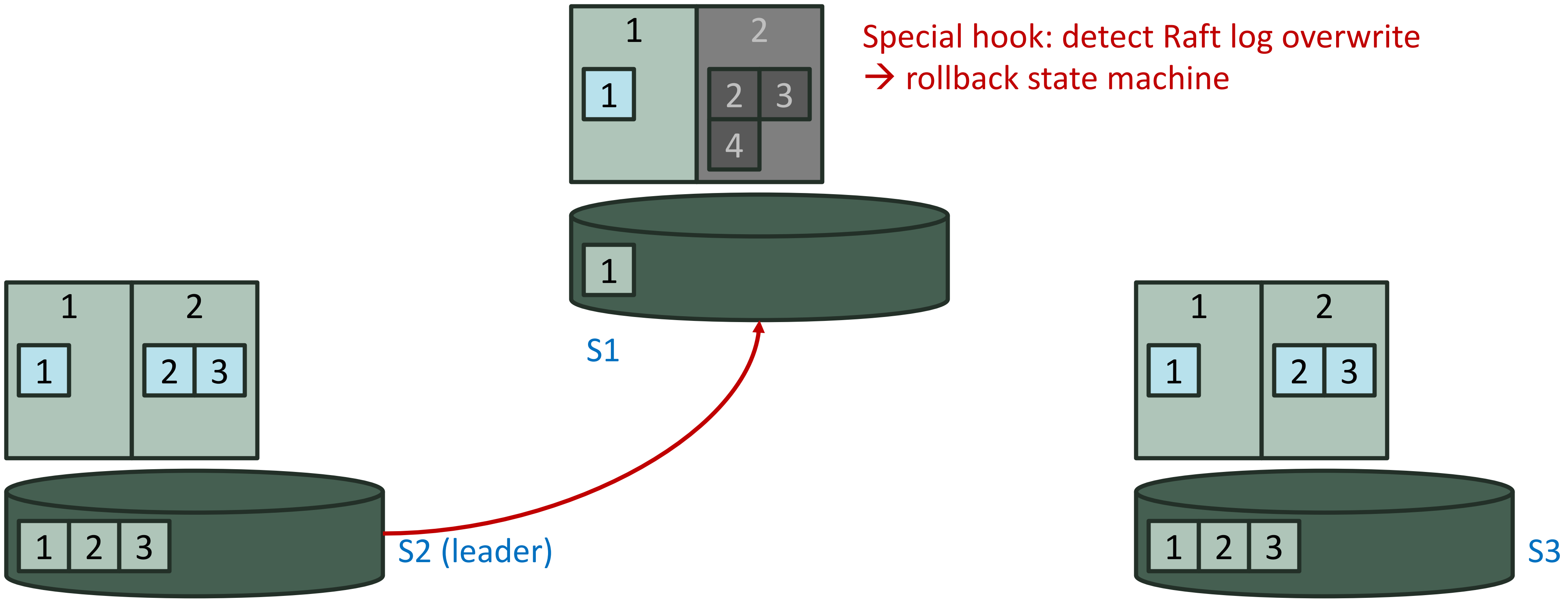
# Log Sharing Scheme

- Payloads are written to state machine before commit
  - Inconsistency?



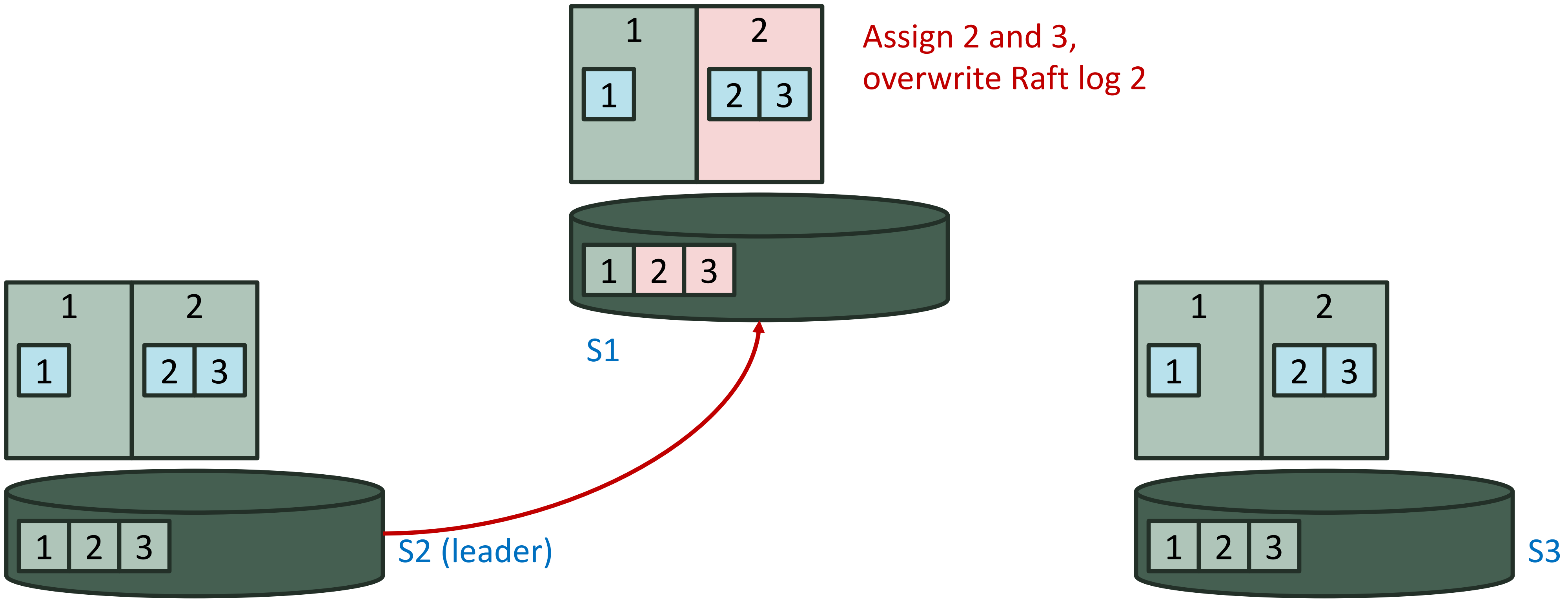
# Log Sharing Scheme

- Payloads are written to state machine before commit
  - Inconsistency?



# Log Sharing Scheme

- Payloads are written to state machine before commit
  - Inconsistency?

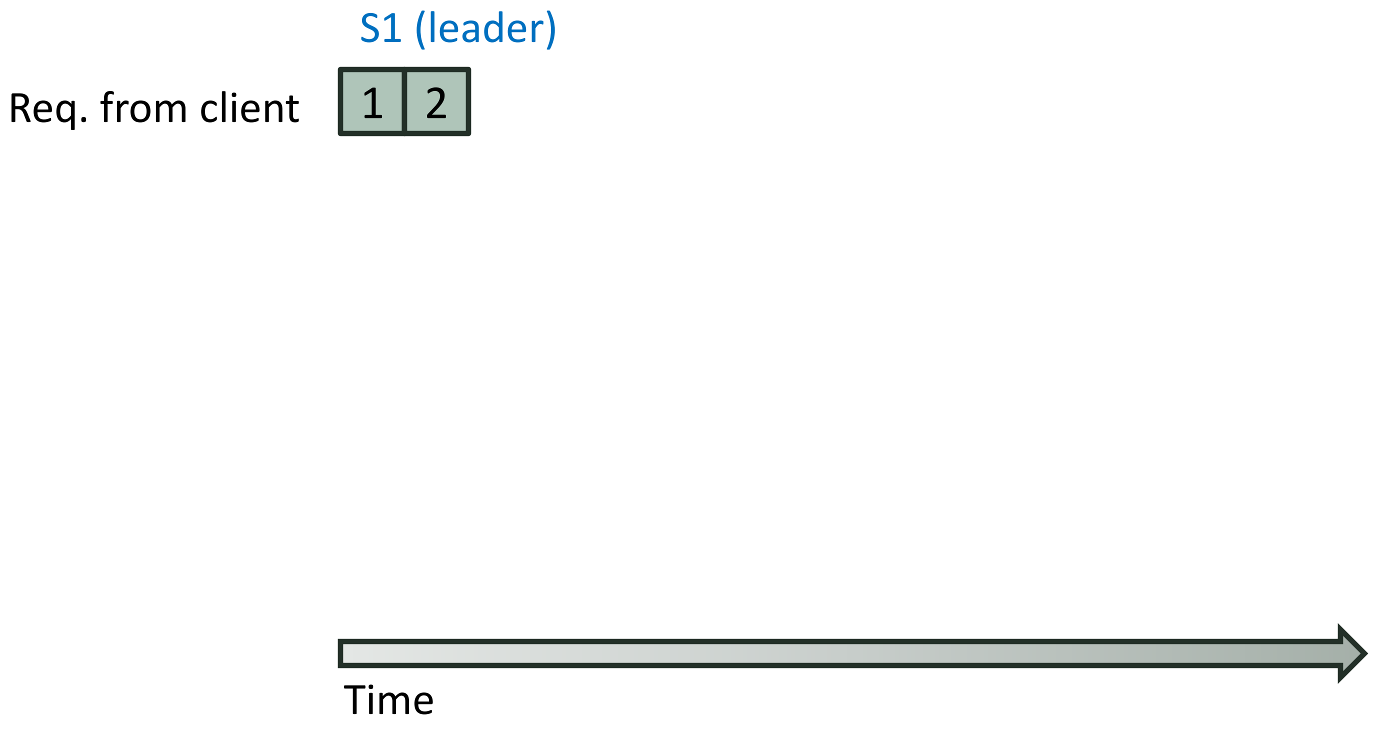


# Log Sharing Scheme

- Payloads are written to state machine before commit
  - Inconsistency?
- Rollback is easily doable
  - Log store's state machine: log-structured format
- What if state machine is general database or key-value store?
  - Un-do logs for rollback? → cancel the benefit of log sharing
  - Not easy

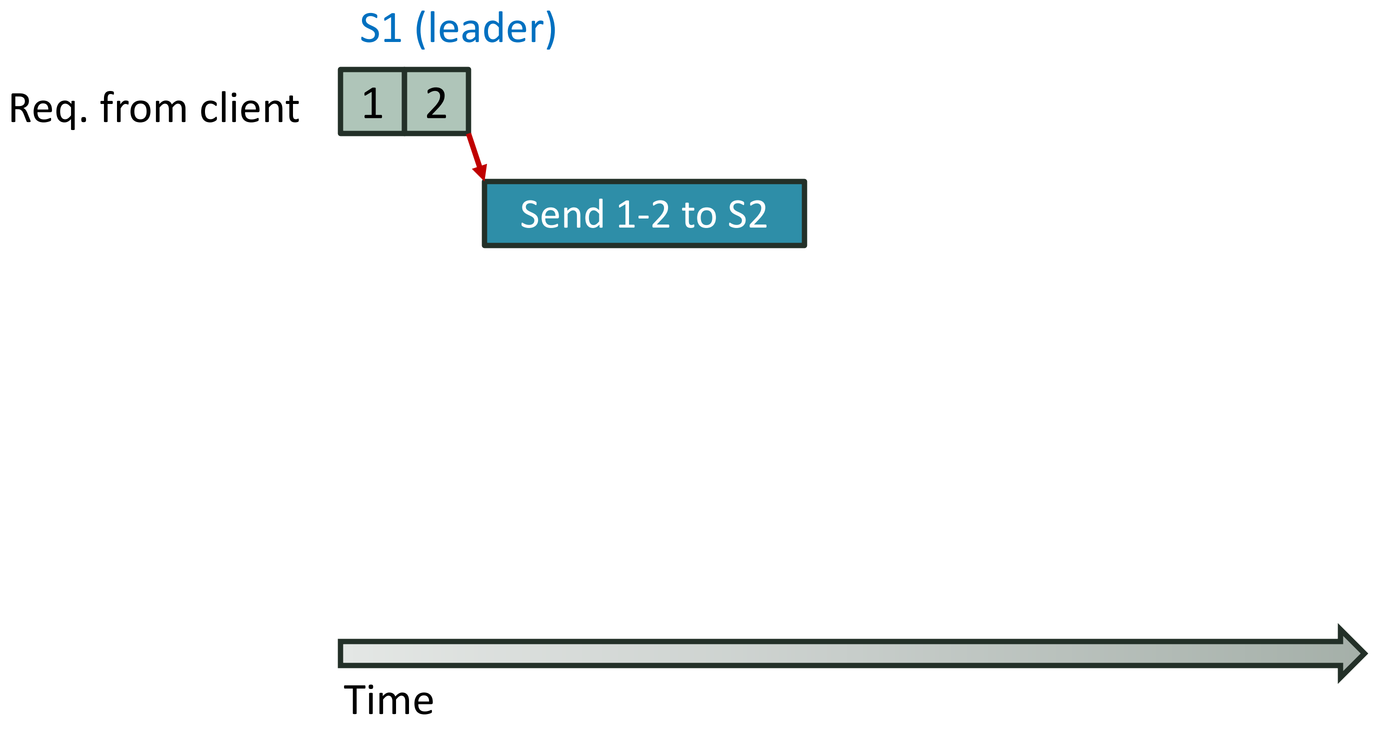
# Group Commit & Pipelining

- To maximize throughput
- Accept new payloads while previous replication is in flight
- Commit multiple user batches at once



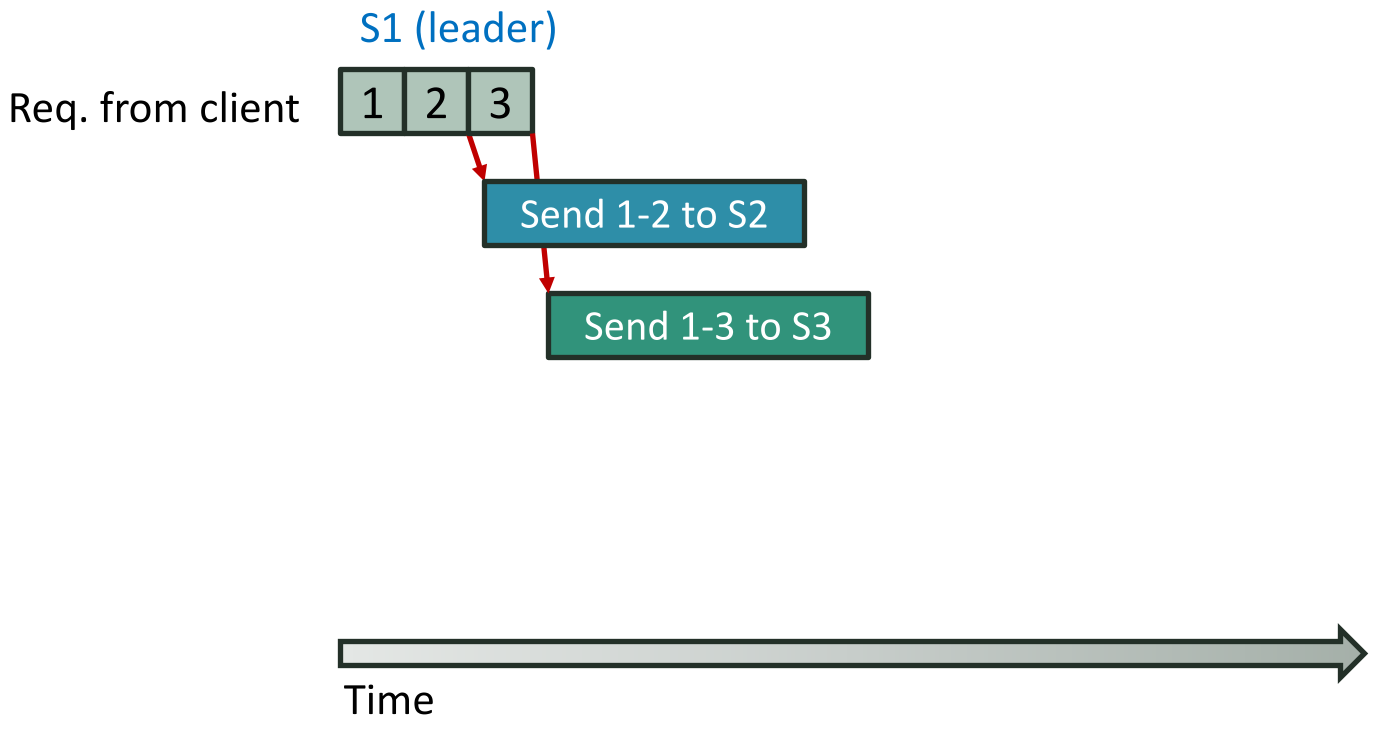
# Group Commit & Pipelining

- To maximize throughput
- Accept new payloads while previous replication is in flight
- Commit multiple user batches at once



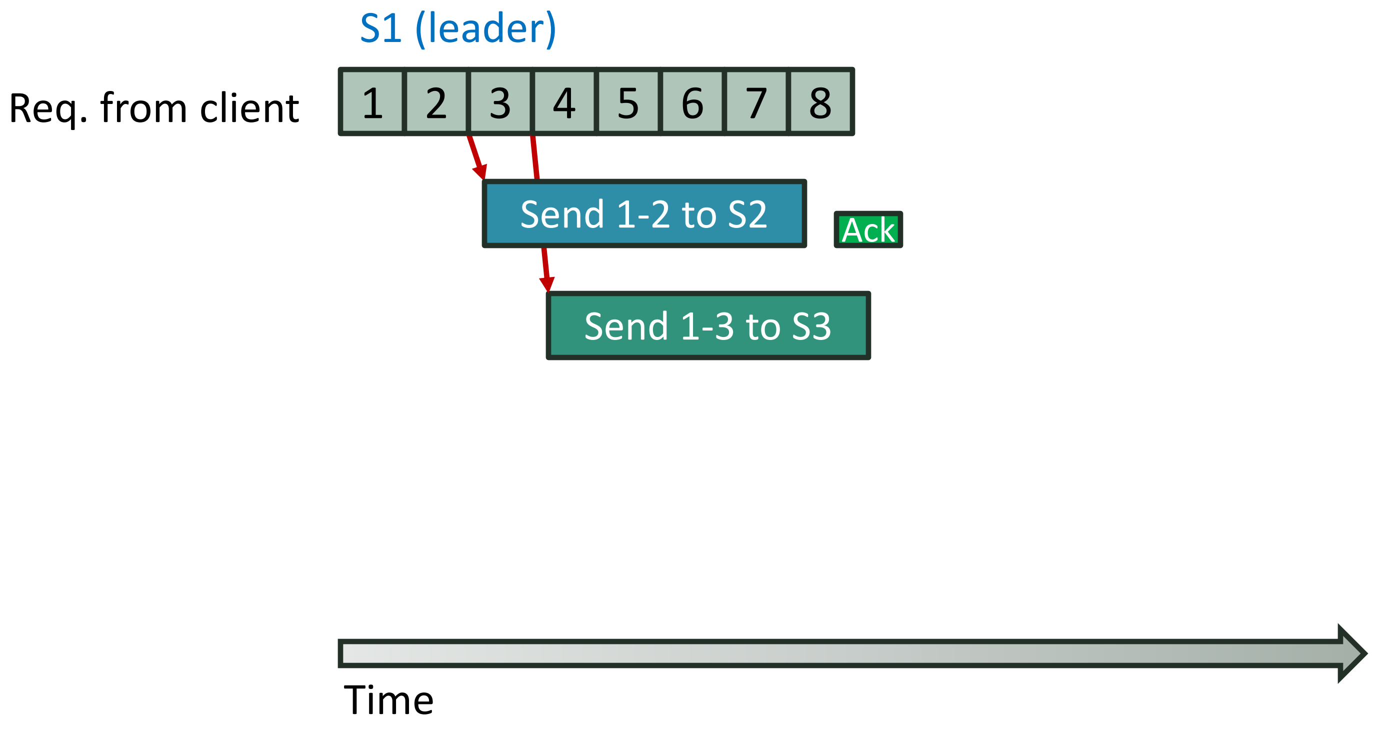
# Group Commit & Pipelining

- To maximize throughput
- Accept new payloads while previous replication is in flight
- Commit multiple user batches at once



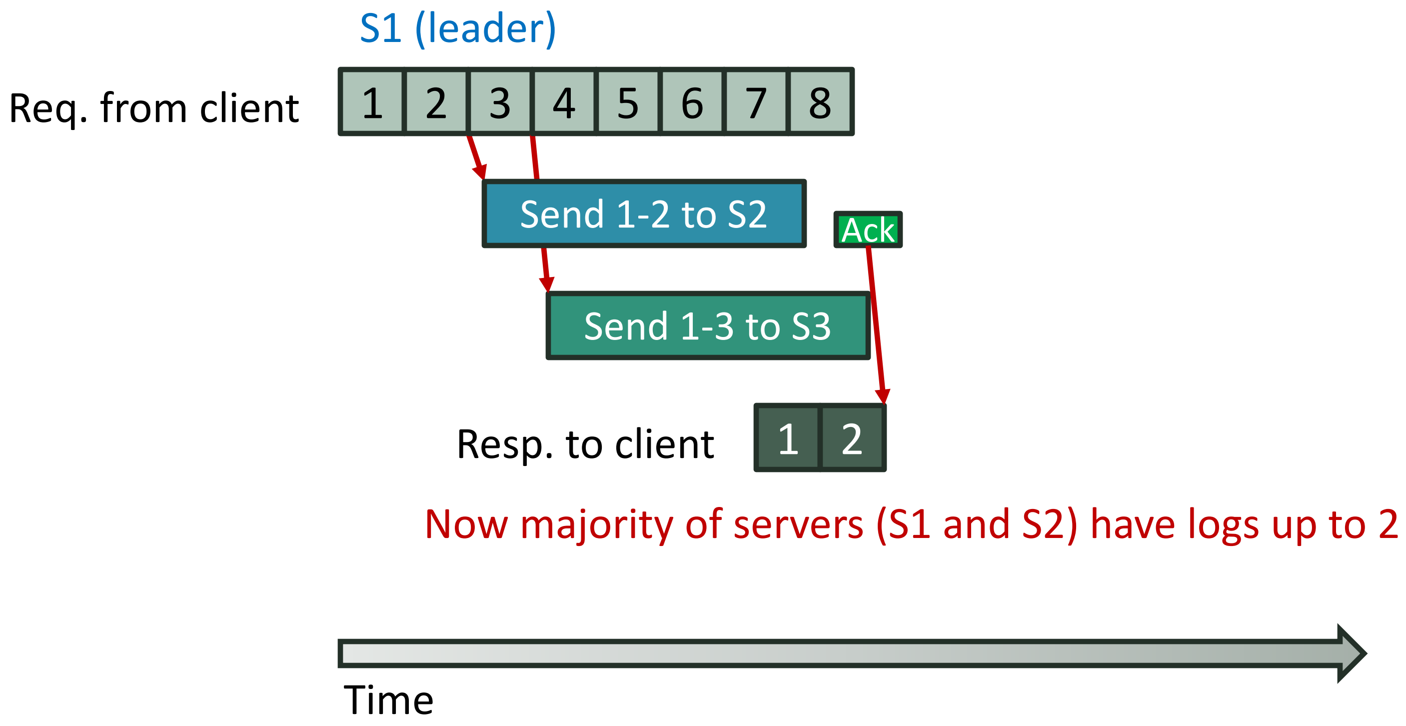
# Group Commit & Pipelining

- To maximize throughput
- Accept new payloads while previous replication is in flight
- Commit multiple user batches at once



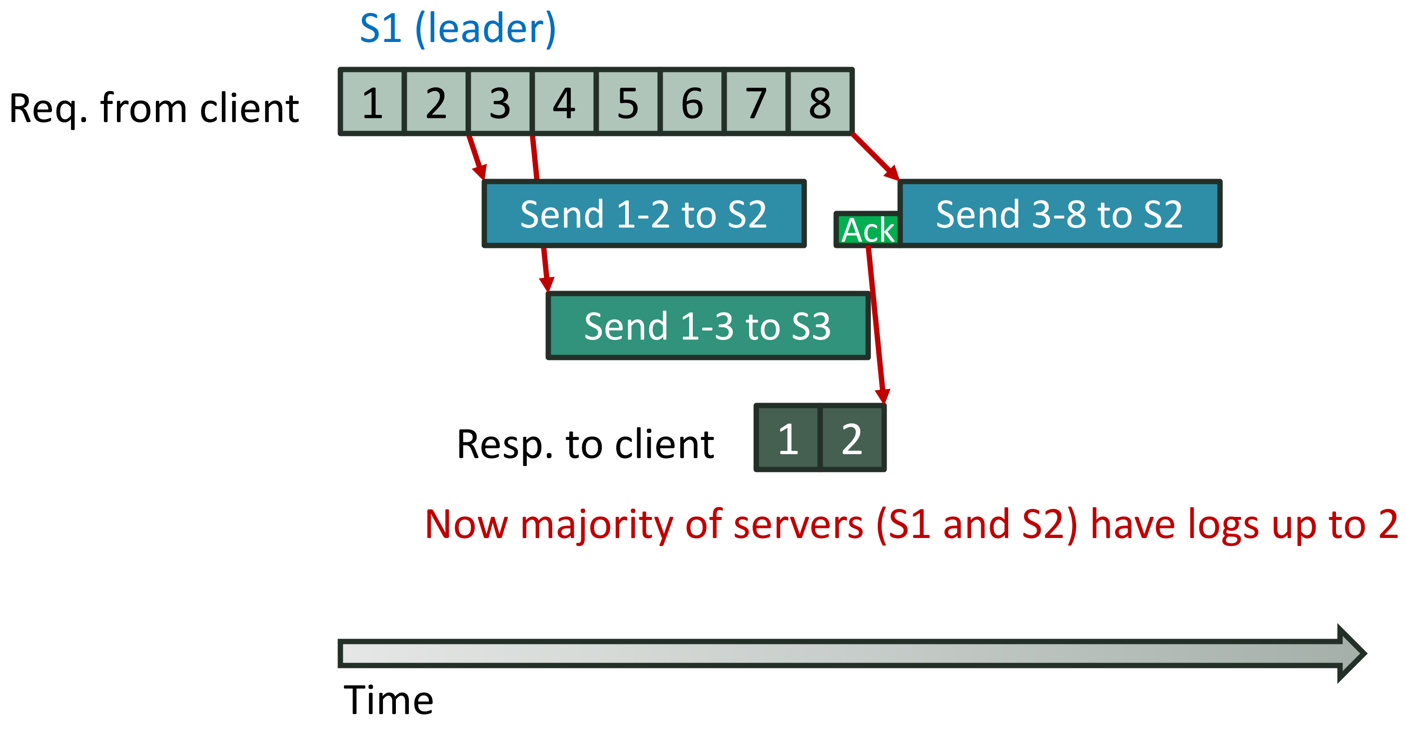
# Group Commit & Pipelining

- To maximize throughput
- Accept new payloads while previous replication is in flight
- Commit multiple user batches at once



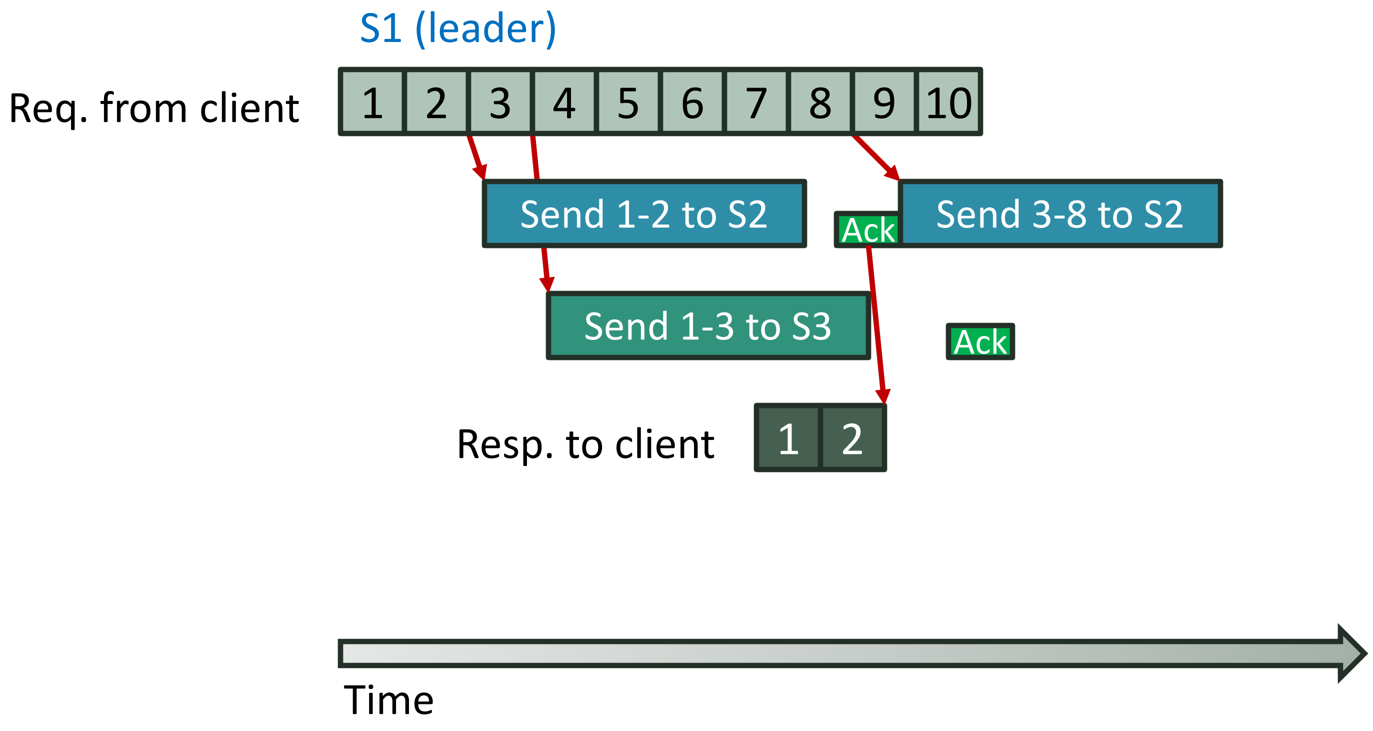
# Group Commit & Pipelining

- To maximize throughput
- Accept new payloads while previous replication is in flight
- Commit multiple user batches at once



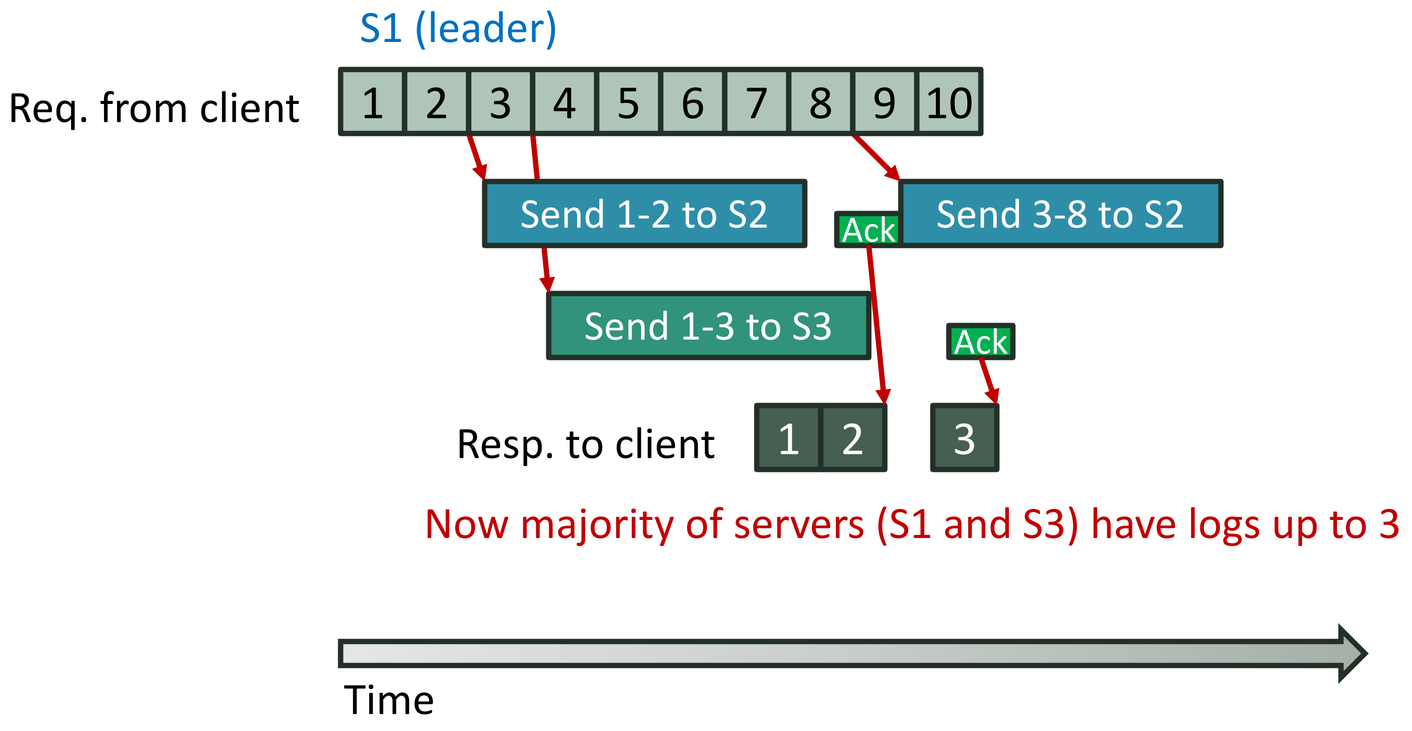
# Group Commit & Pipelining

- To maximize throughput
- Accept new payloads while previous replication is in flight
- Commit multiple user batches at once



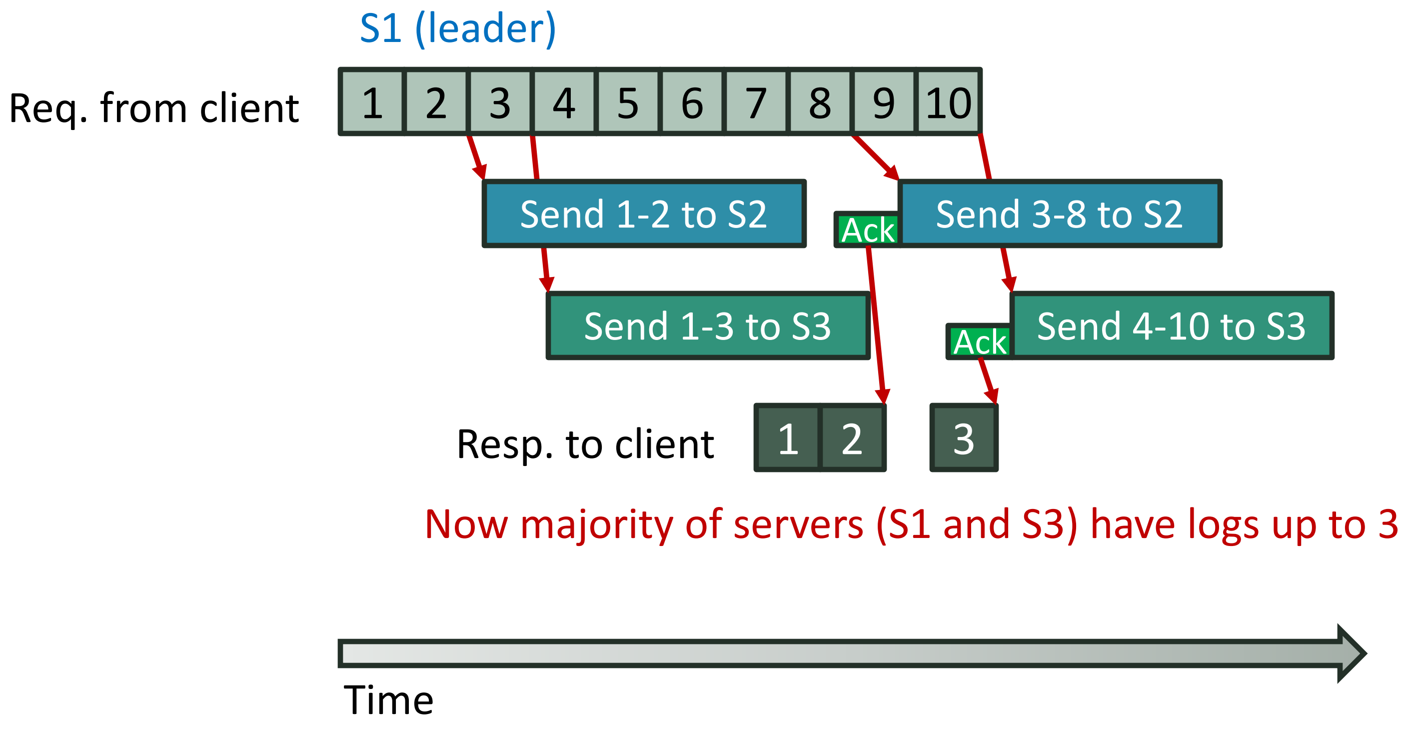
# Group Commit & Pipelining

- To maximize throughput
- Accept new payloads while previous replication is in flight
- Commit multiple user batches at once



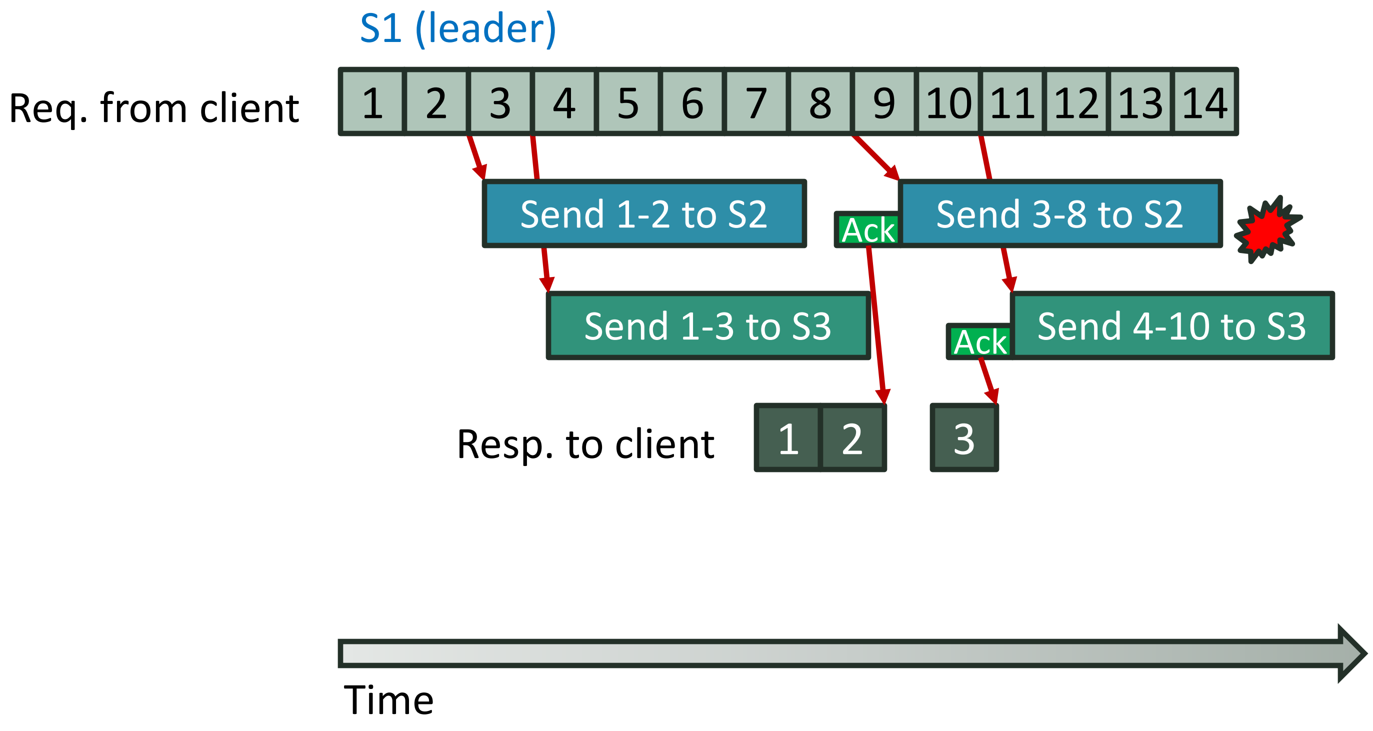
# Group Commit & Pipelining

- To maximize throughput
- Accept new payloads while previous replication is in flight
- Commit multiple user batches at once



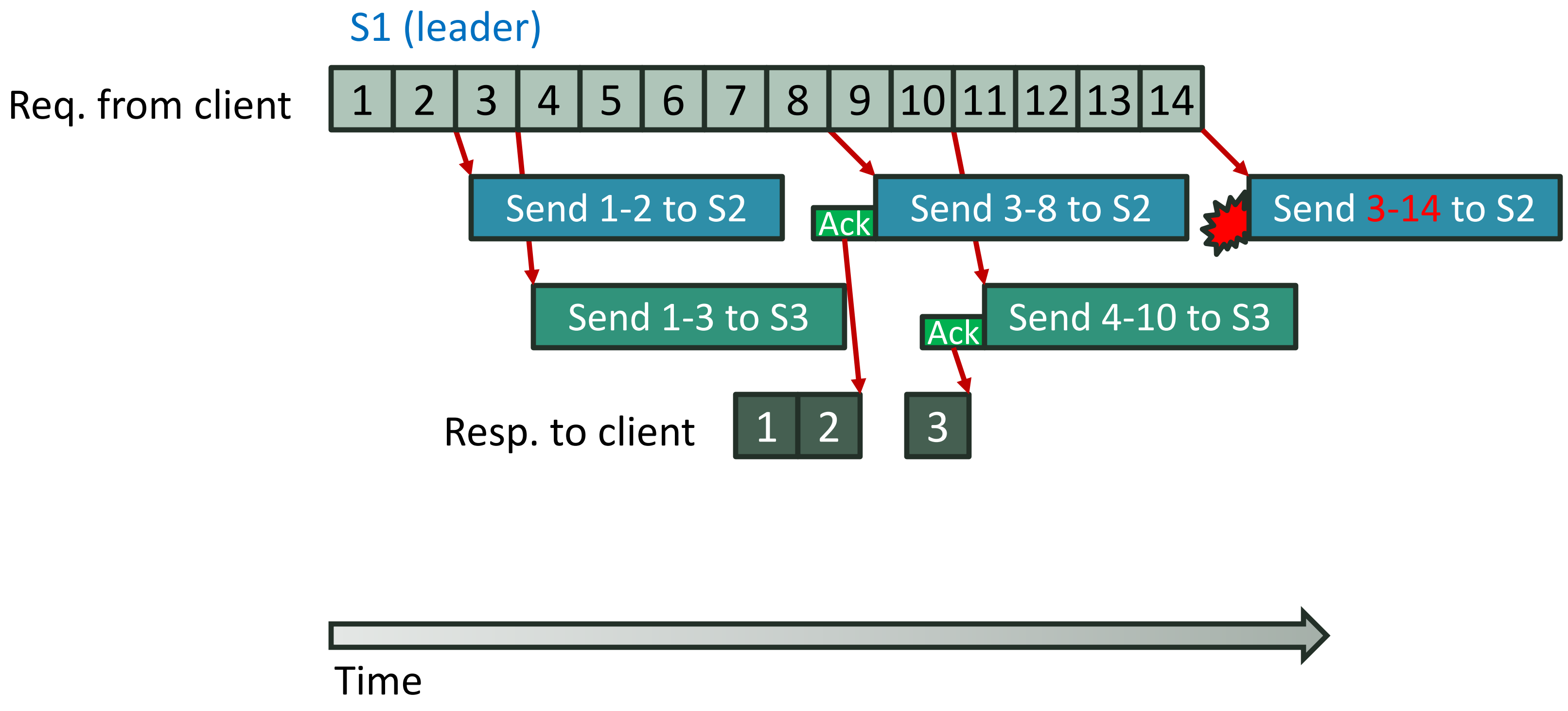
# Group Commit & Pipelining

- To maximize throughput
- Accept new payloads while previous replication is in flight
- Commit multiple user batches at once



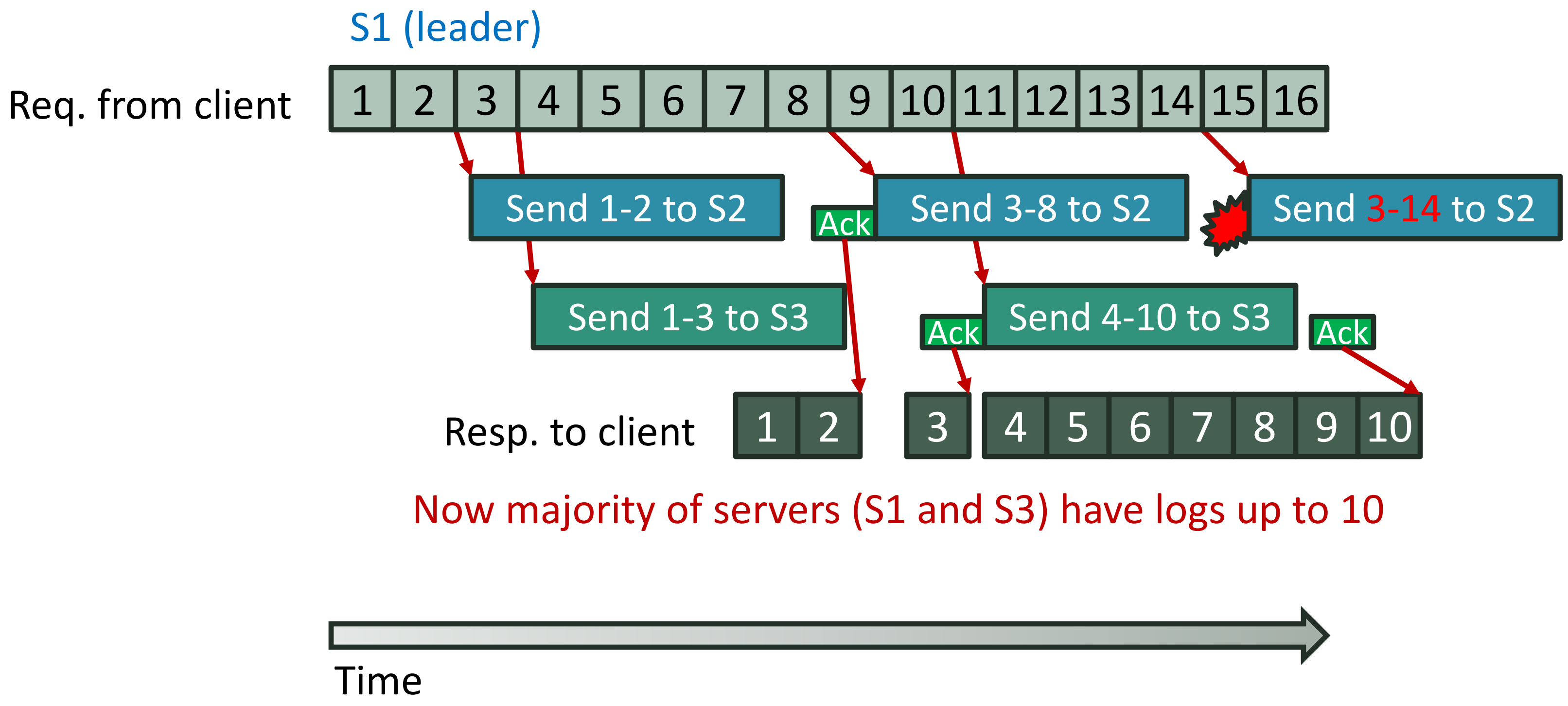
# Group Commit & Pipelining

- To maximize throughput
- Accept new payloads while previous replication is in flight
- Commit multiple user batches at once



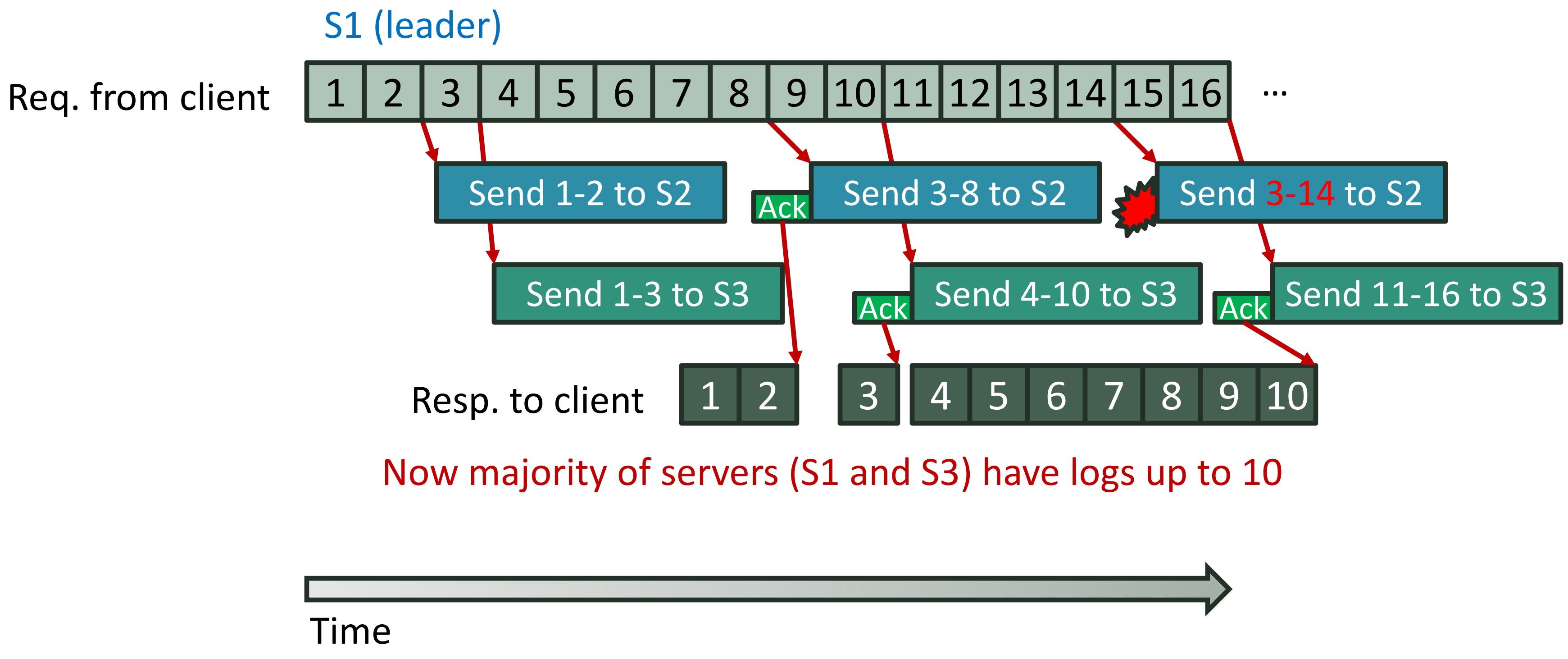
# Group Commit & Pipelining

- To maximize throughput
- Accept new payloads while previous replication is in flight
- Commit multiple user batches at once



# Group Commit & Pipelining

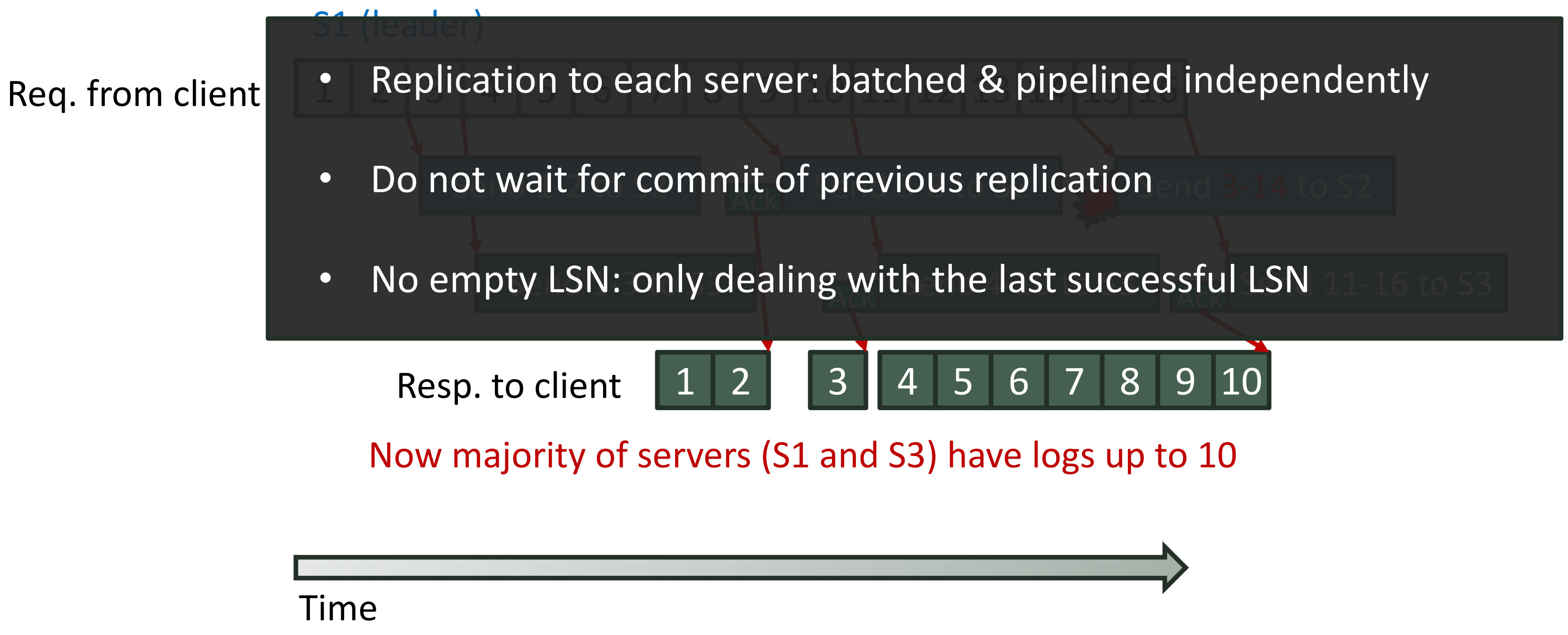
- To maximize throughput
- Accept new payloads while previous replication is in flight
- Commit multiple user batches at once



# Group Commit & Pipelining

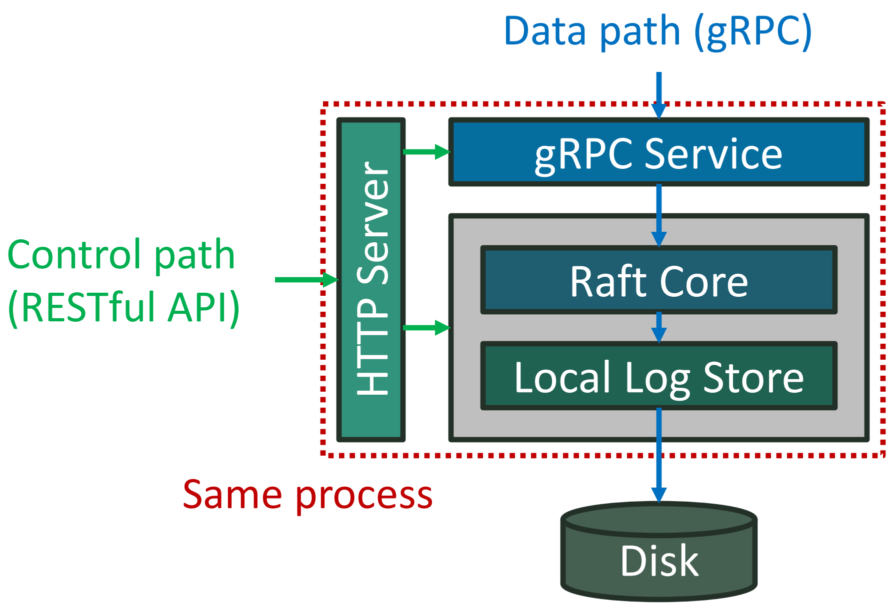
- To maximize throughput
- Accept new payloads while previous replication is in flight
- Commit multiple user batches at once

- Replication to each server: batched & pipelined independently
- Do not wait for commit of previous replication
- No empty LSN: only dealing with the last successful LSN

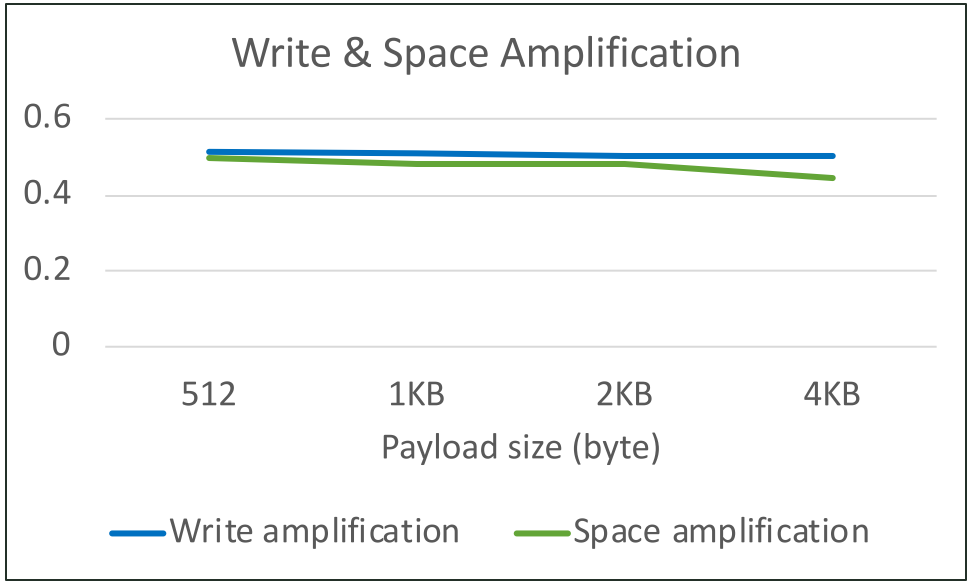


# Implementation

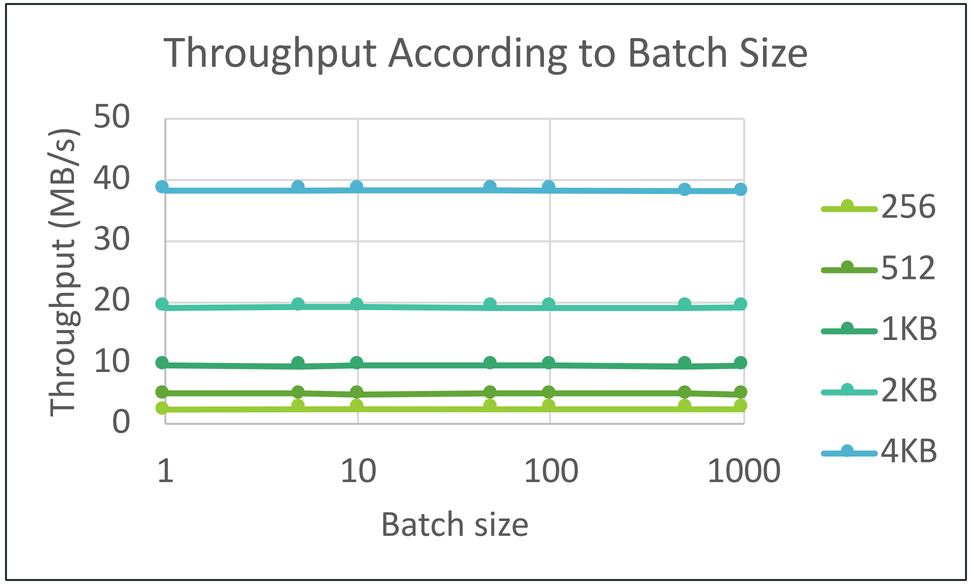
- Written in C++
  - Raft: 5,578 lines of code
  - Core logic: 20,196 lines of code
  - gRPC service: 7,494 lines of code
- Deployed as a service

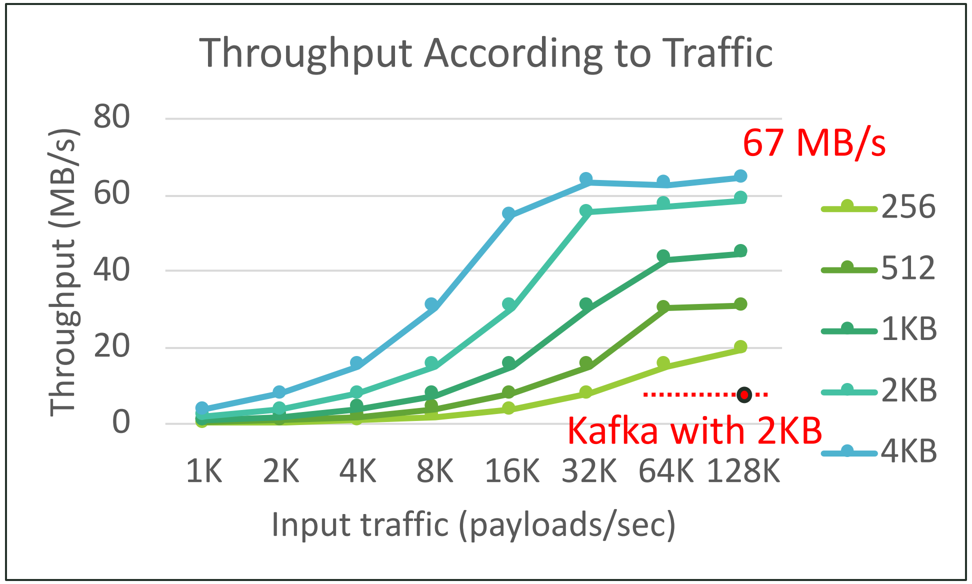


# Brief Evaluation




- 3 replicas in the same DC
  - Not powerful: 2.5 cores each
- Client: different node in the same DC
  - Send traffic using gRPC stream
- Max network throughput (per stream): 85-90 MB/s



# Summary

- State machine-based replication for log store
  - End up with duplicate log issue
- Log sharing scheme
  - Using characteristics of log-structured state machine
  - Group commit + pipelining on top of it
- What's next?
  - Distributed (sharded) log store
    - Scaling out over multiple shards (partitions, share nothing)
    - How can we guarantee global ordering?
  - Generalization: based on other consensus protocol, not Raft?
    - What if we allow empty LSN in the middle?
  - Extending log sharing scheme for general databases
    - Rollback of indexes without un-do logs?

# Thank You