RubbleDB: CPU-Efficient Replication with NVMe-oF

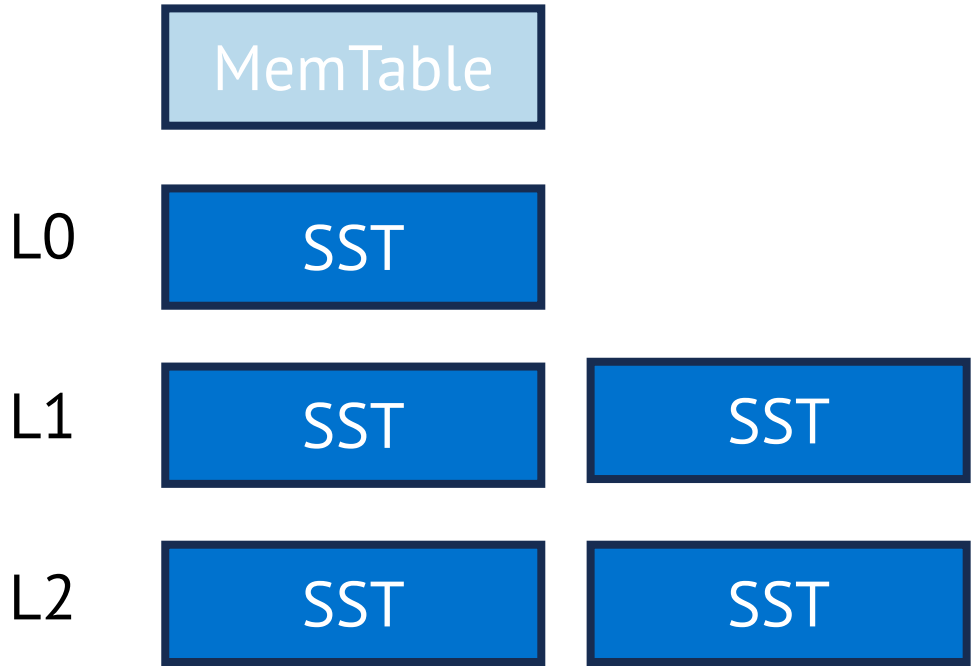
Haoyu Li[1], Sheng Jiang[1], Chen Chen[1], Ashwini Raina[2], Xingyu Zhu[1], Changxu Luo[1], Asaf Cidon[1]

[1]*Columbia University,* [2]*Princeton University*

USENIX ATC '23

CPU: the bottleneck in key-value stores

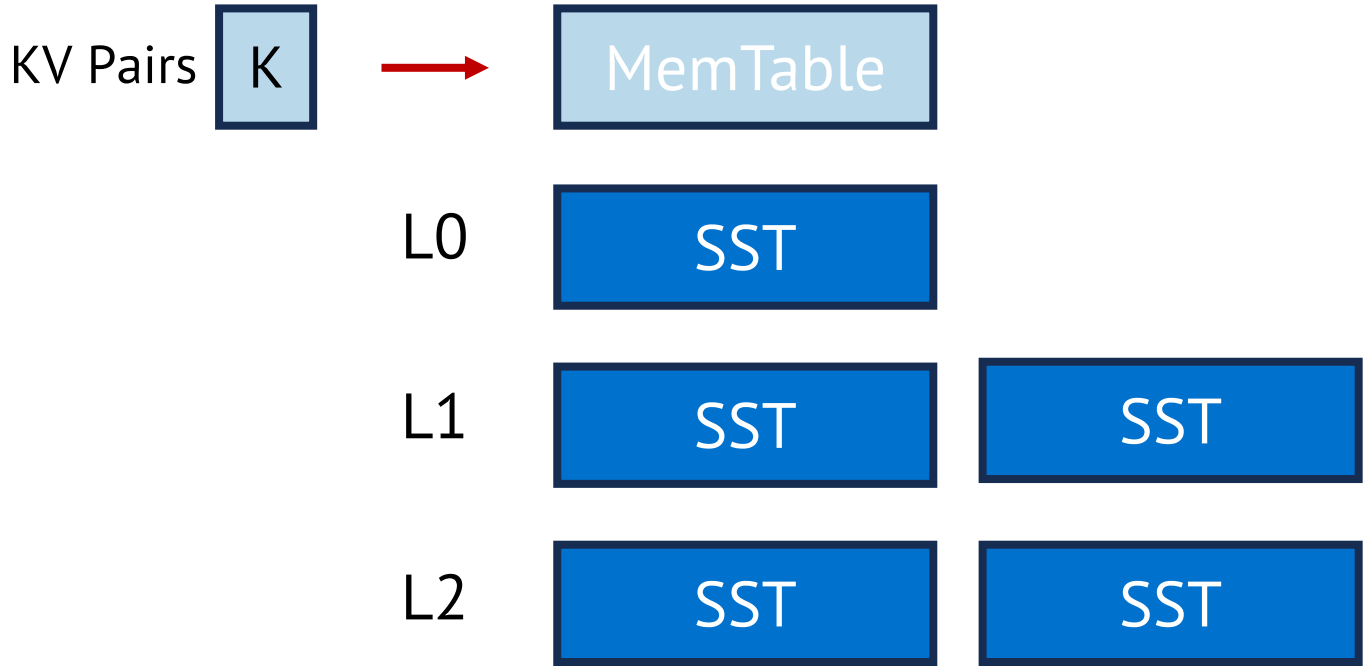
Compactions in log-structured merge trees (LSM) is CPU expensive



*Workload: YCSB 100% insert, 10GB DB size

CPU: the bottleneck in key-value stores

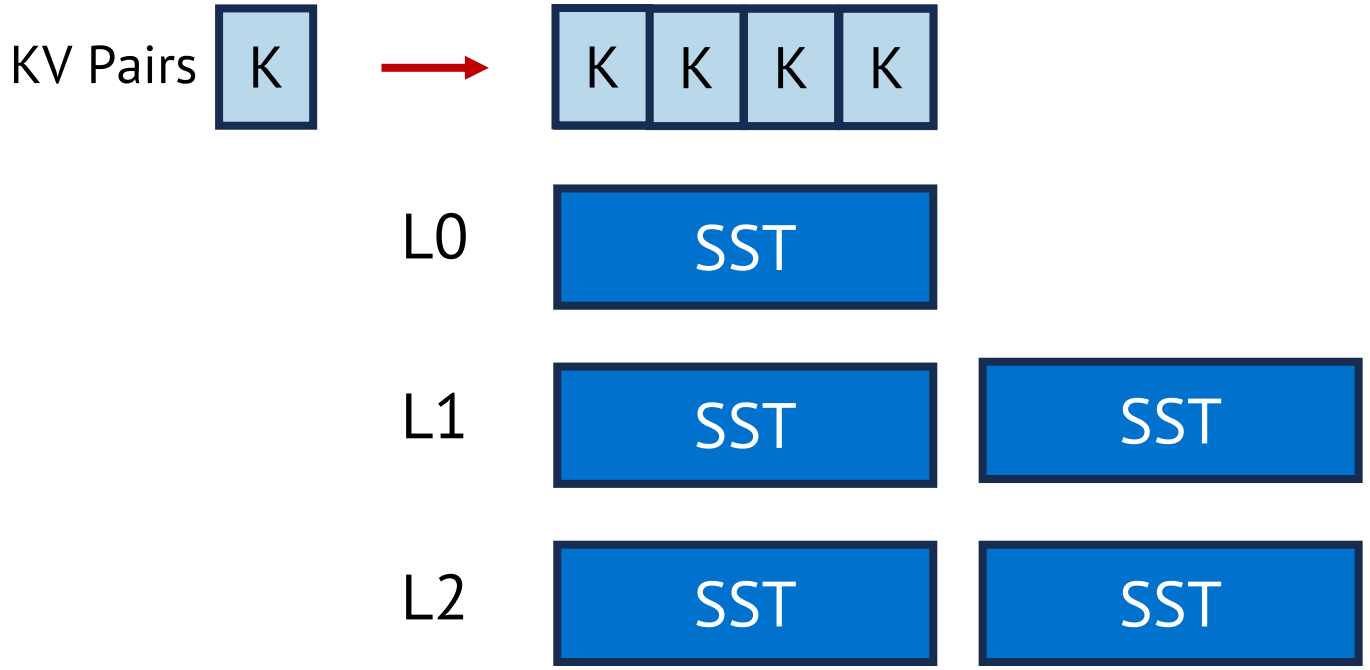
Compactions in log-structured merge trees (LSM) is CPU expensive



*Workload: YCSB 100% insert, 10GB DB size

CPU: the bottleneck in key-value stores

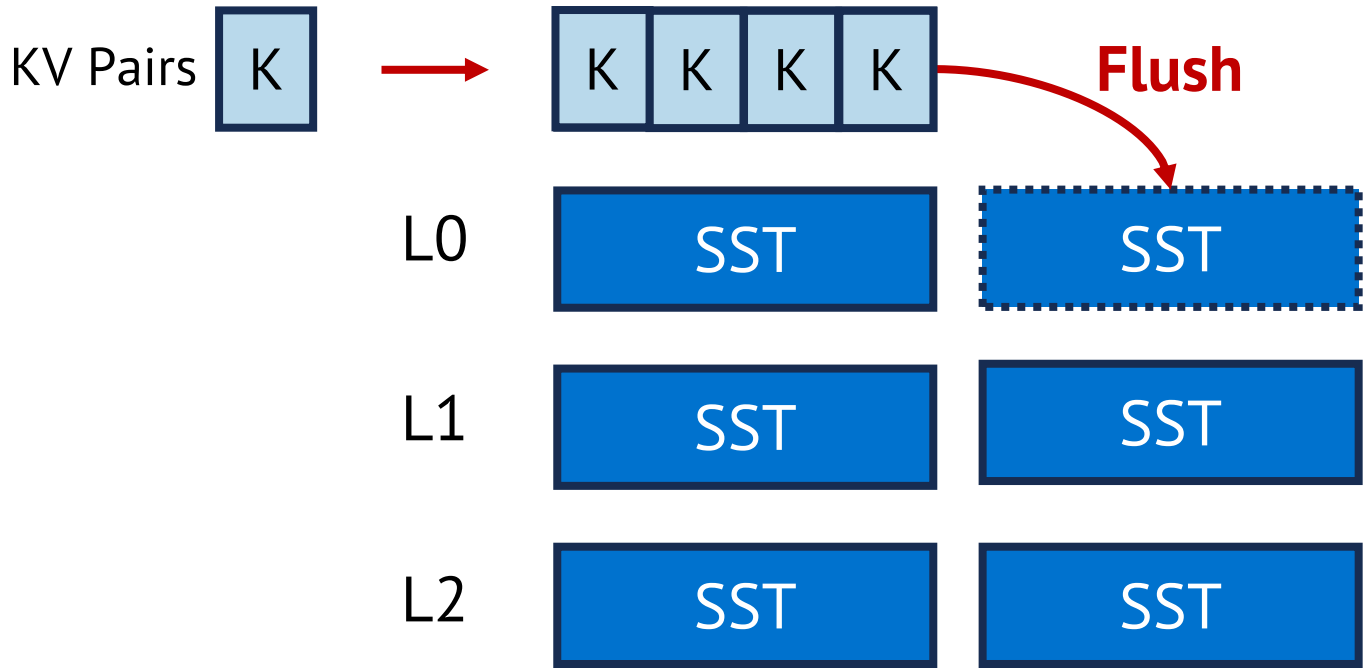
Compactions in log-structured merge trees (LSM) is CPU expensive



*Workload: YCSB 100% insert, 10GB DB size

CPU: the bottleneck in key-value stores

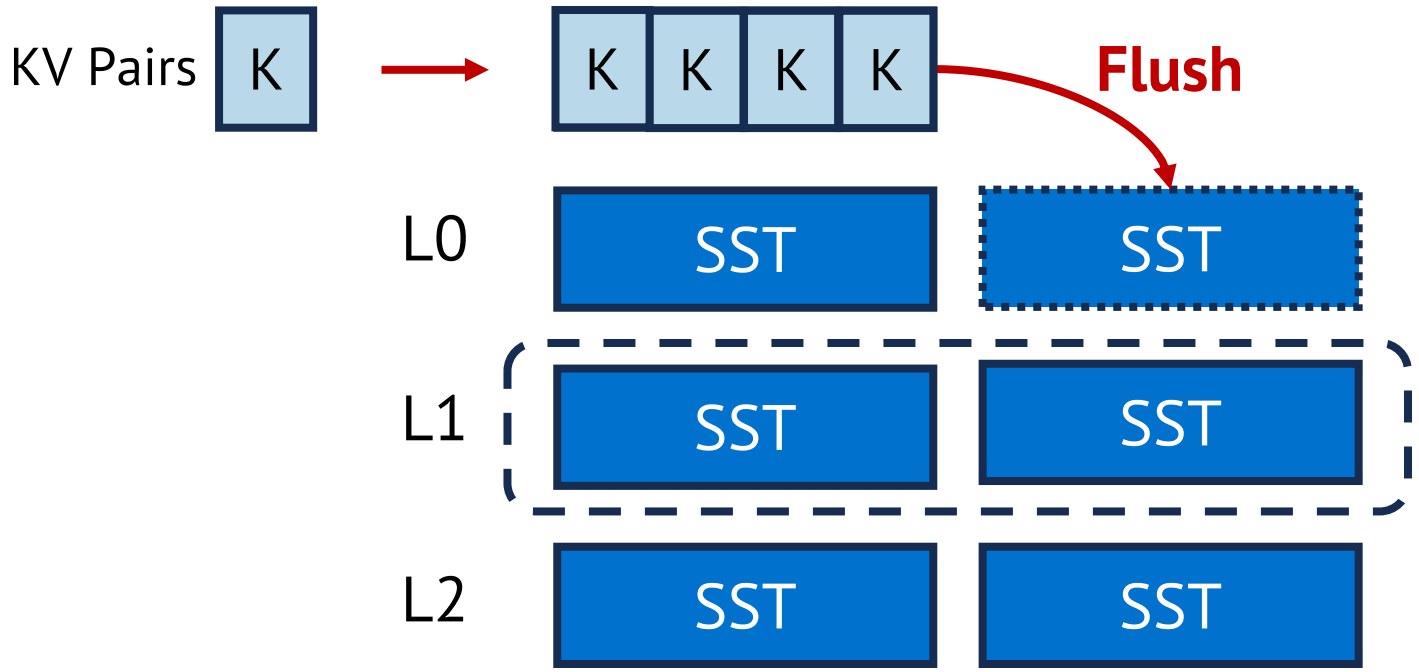
Compactions in log-structured merge trees (LSM) is CPU expensive



*Workload: YCSB 100% insert, 10GB DB size

CPU: the bottleneck in key-value stores

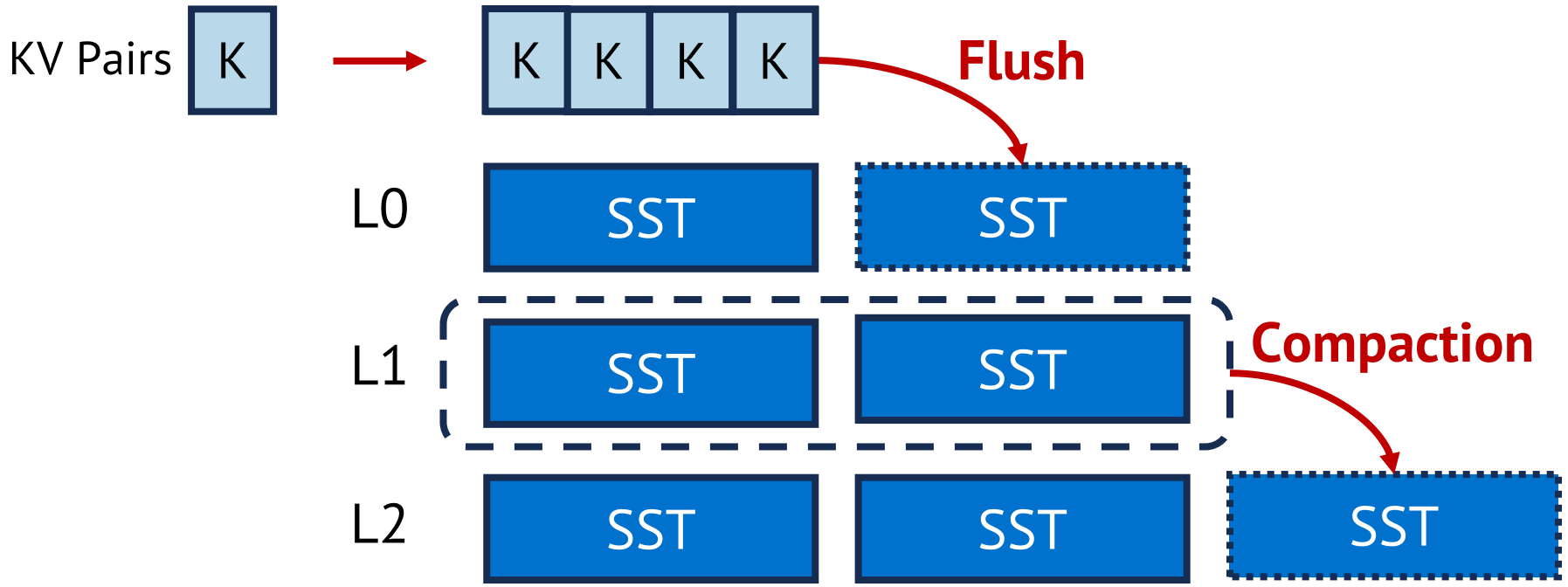
Compactions in log-structured merge trees (LSM) is CPU expensive



*Workload: YCSB 100% insert, 10GB DB size

CPU: the bottleneck in key-value stores

Compactions in log-structured merge trees (LSM) is CPU expensive

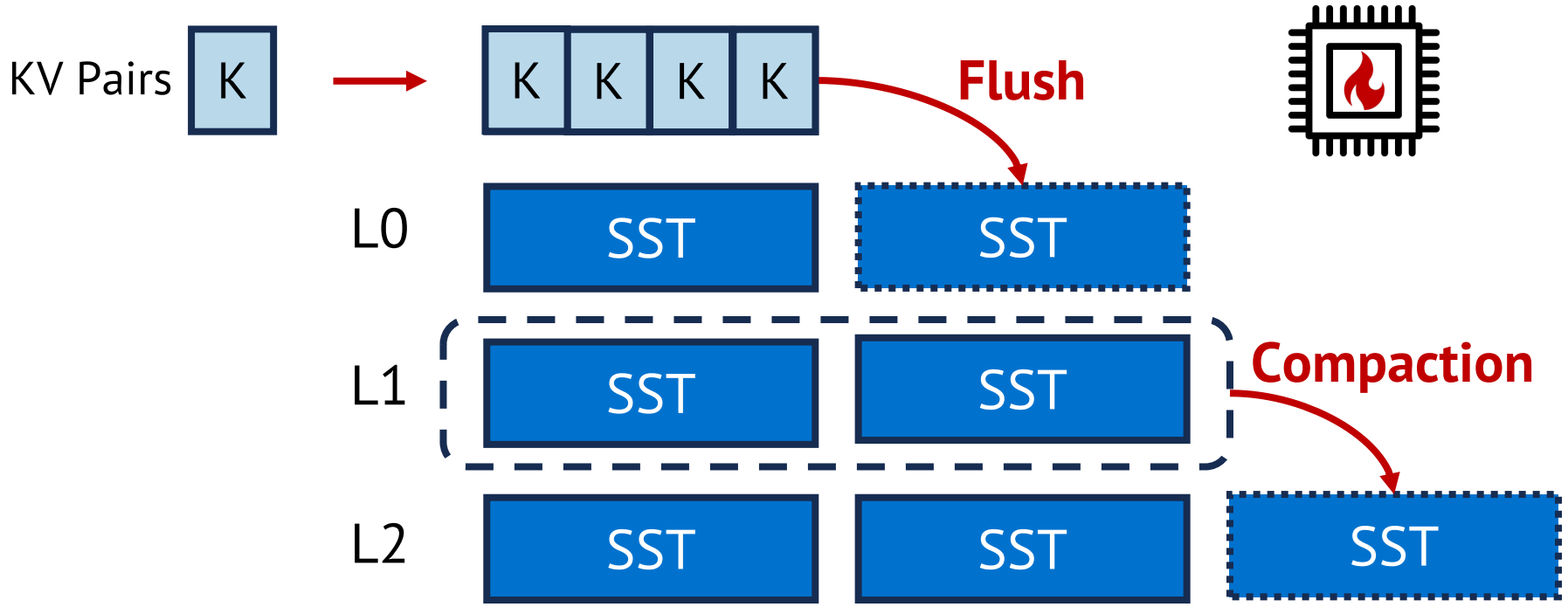


*Workload: YCSB 100% insert, 10GB DB size

CPU: the bottleneck in key-value stores

Compactions in log-structured merge trees (LSM) is CPU expensive

- Up to 72% of the total CPU time*!

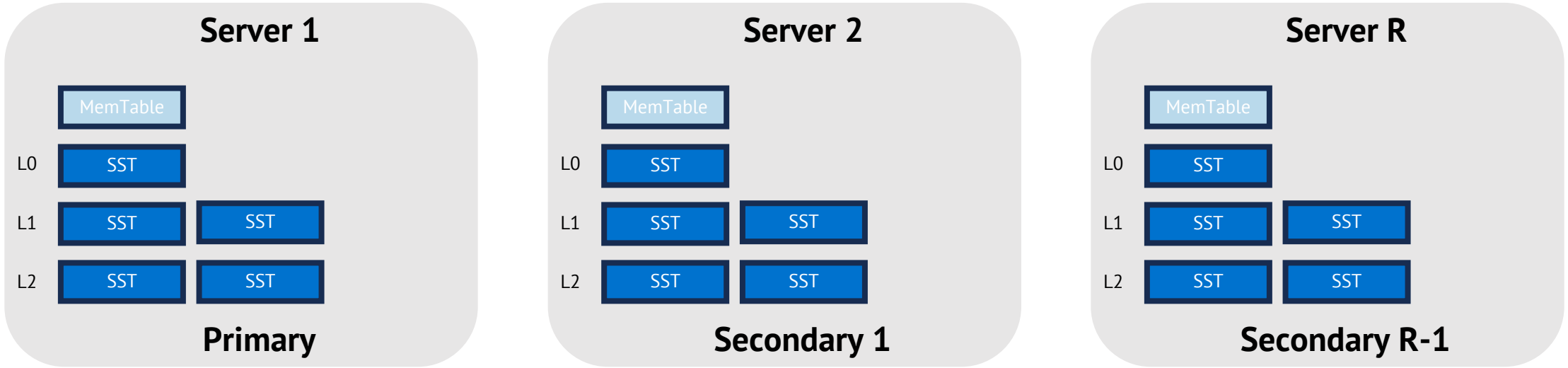


*Workload: YCSB 100% insert, 10GB DB size

Compactions in replicated LSM trees

Redundant compactions happen in each replica

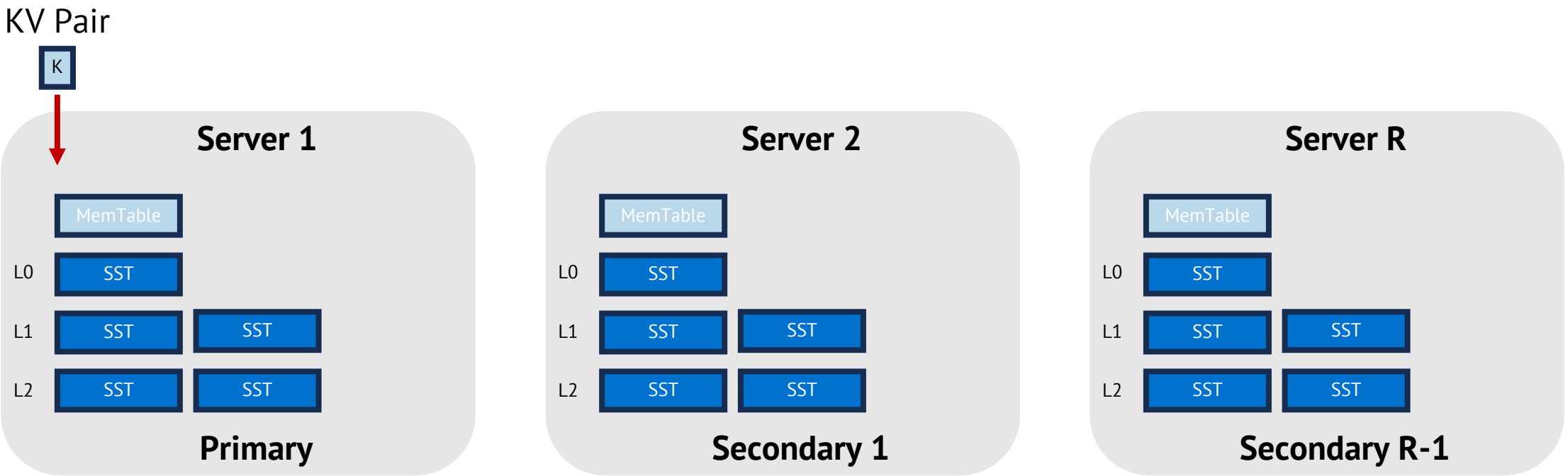
- e.g., CockroachDB, ZippyDB, Cassandra, and etc.



Compactions in replicated LSM trees

Redundant compactions happen in each replica

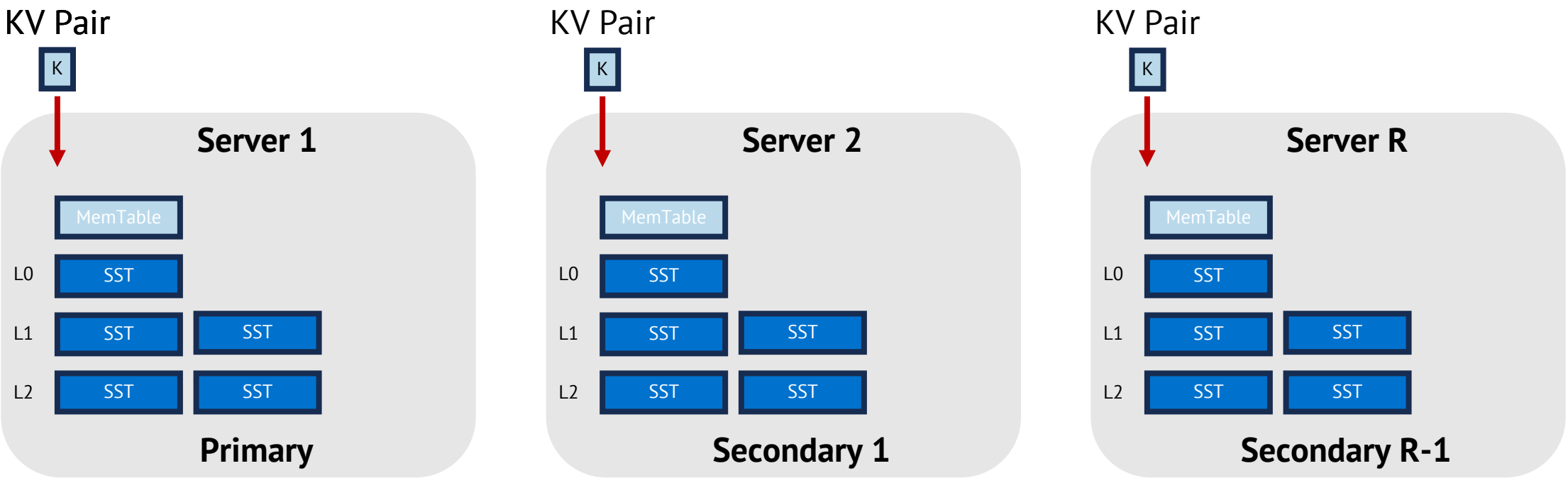
- e.g., CockroachDB, ZippyDB, Cassandra, and etc.



Compactions in replicated LSM trees

Redundant compactions happen in each replica

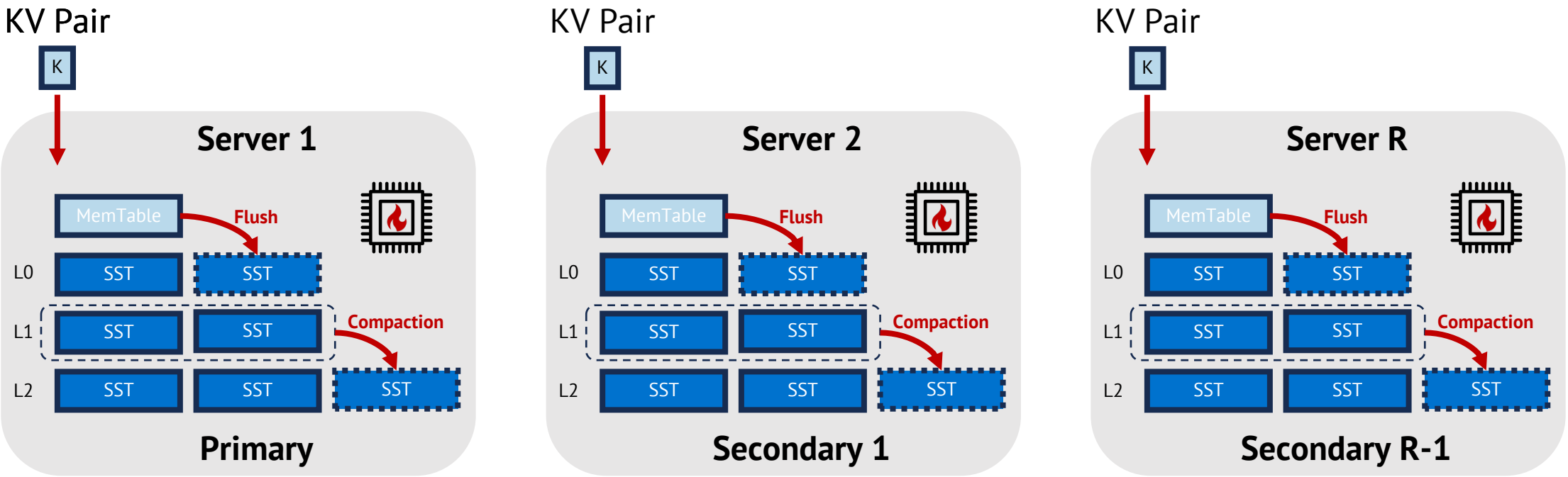
- e.g., CockroachDB, ZippyDB, Cassandra, and etc.



Compactions in replicated LSM trees

Redundant compactions happen in each replica

- e.g., CockroachDB, ZippyDB, Cassandra, and etc.

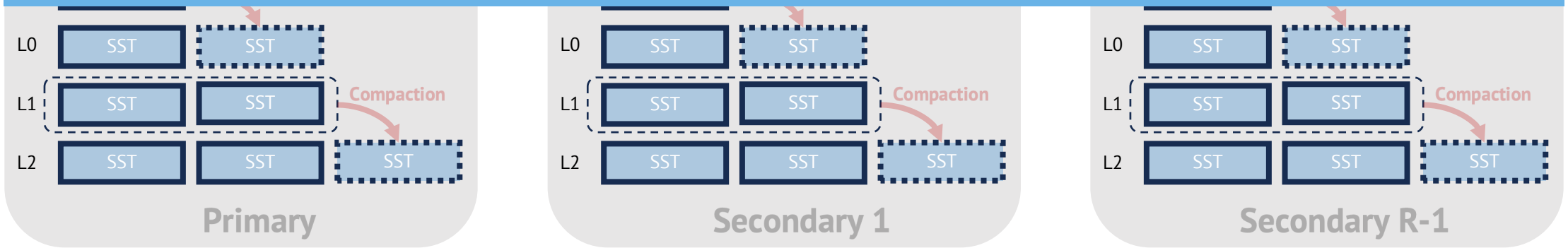


Compactions in replicated LSM trees

Redundant compactions happen in each replica

- e.g., CockroachDB, ZippyDB, Cassandra, and etc.

Can we remove redundant compactions?

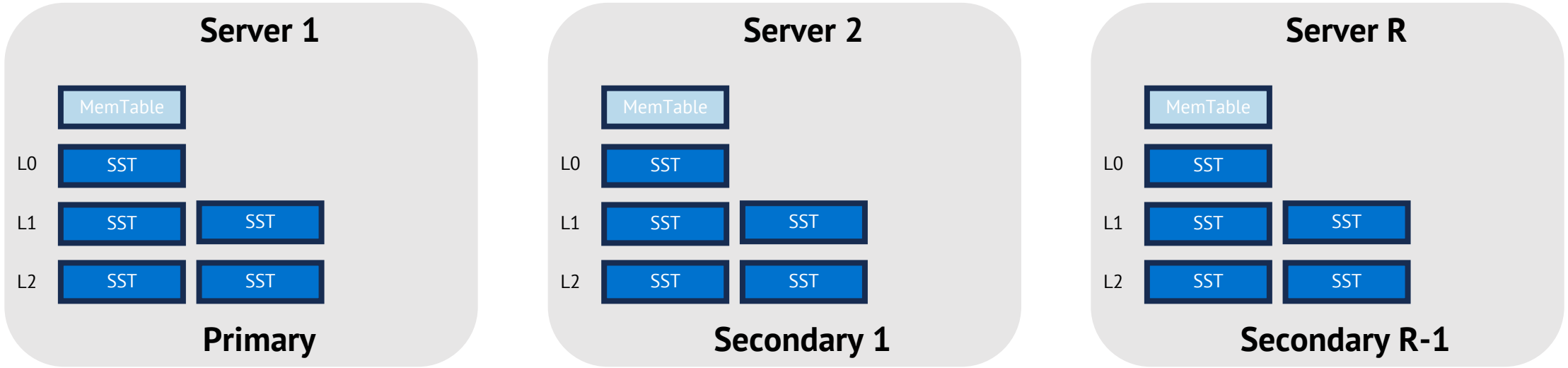


This Talk

Redundant compactions can be eliminated

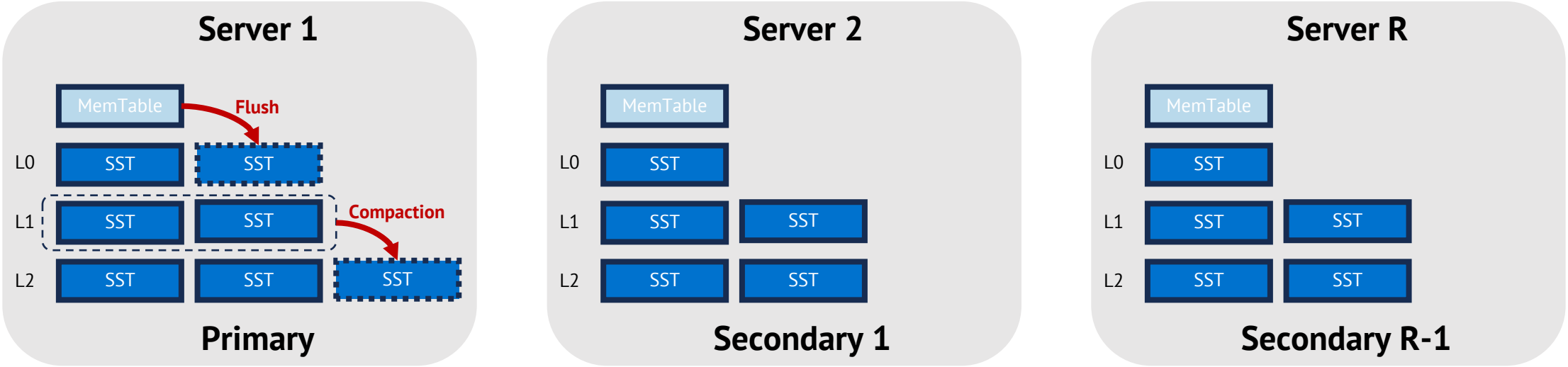
RubbleDB makes it practical to share compaction results with NVMe-oF

How to remove redundant compactions?



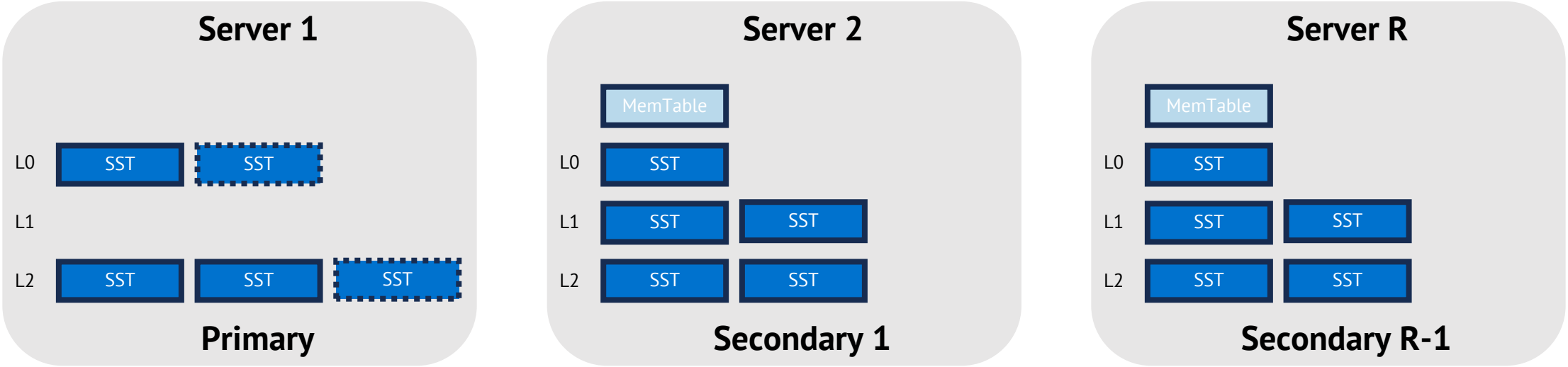
How to remove redundant compactions?

- Only perform compactions in the primary



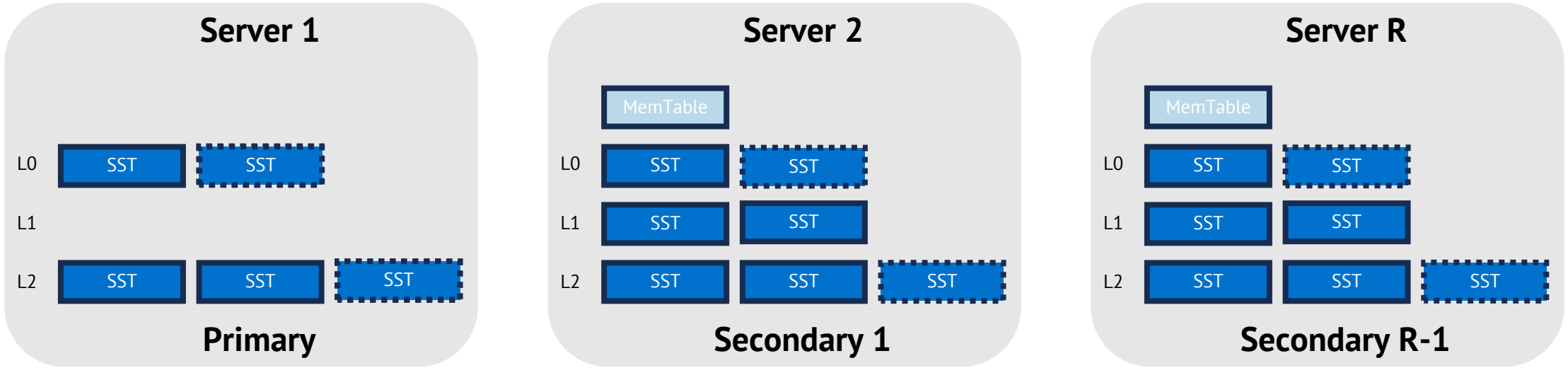
How to remove redundant compactions?

- Only perform compactions in the primary



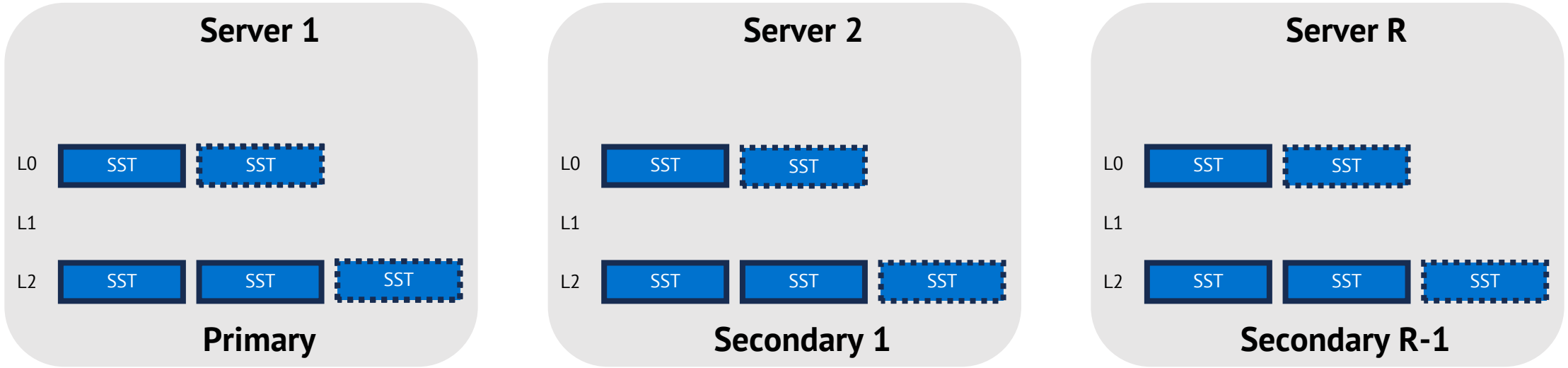
How to remove redundant compactions?

- Only perform compactions in the primary
- Ship compacted SST files to each secondary



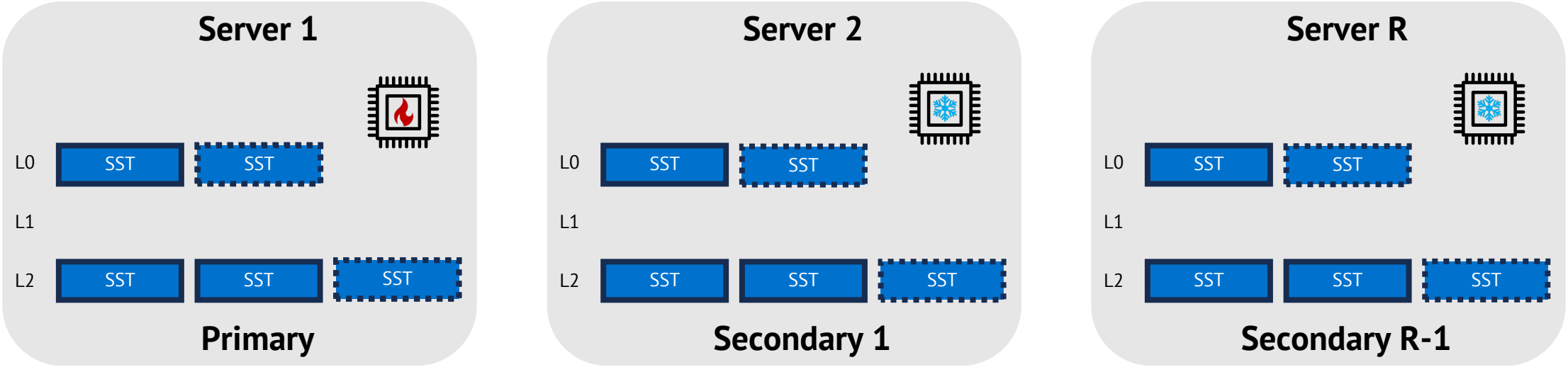
How to remove redundant compactions?

- Only perform compactions in the primary
- Ship compacted SST files to each secondary
- Delete input files in secondaries

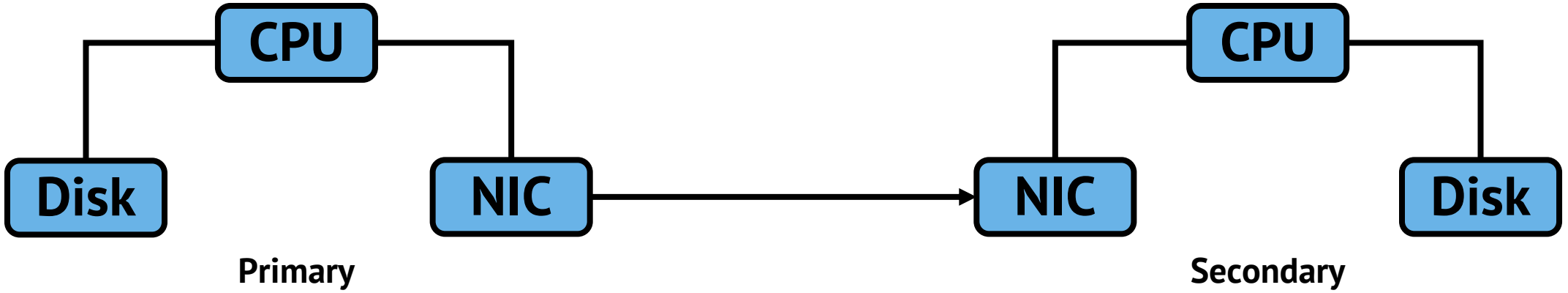


How to remove redundant compactions?

- Only perform compactions in the primary
- Ship compacted SST files to each secondary
- Delete input files in secondaries

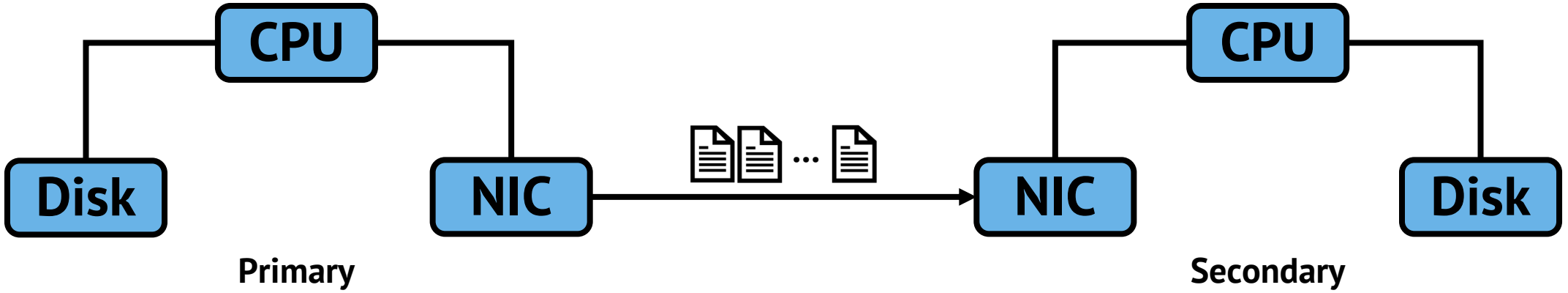


Challenges of sharing SST files



Challenges of sharing SST files

- Heavy network traffic
  - Luckily datacenter network is often underutilized[1][2]



[1] Alibaba Cluster Trace 2018.
[2] Building an elastic query engine on disaggregated storage. (NSDI 20)

Challenges of sharing SST files

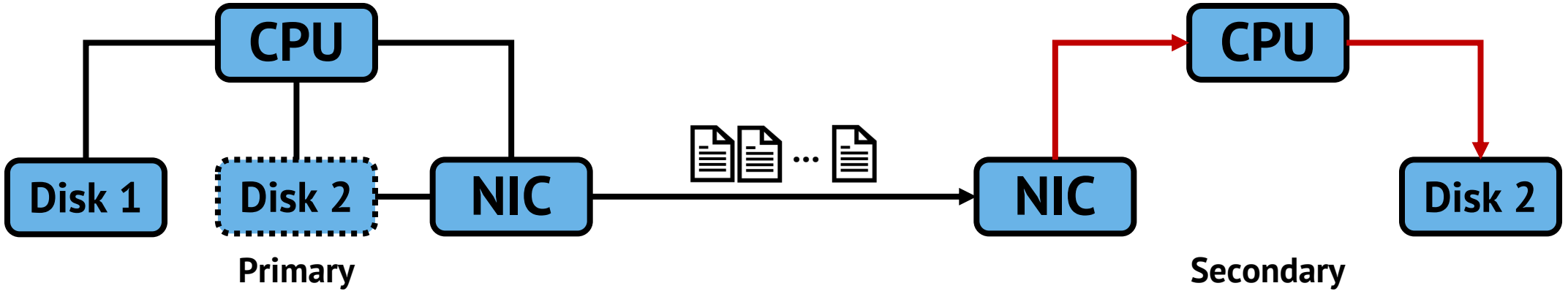
- Heavy network traffic
  - Luckily datacenter network is often underutilized[1][2]

- CPU involvement on the secondary
  - After receiving the data, the secondary writes it to the local disk

[1] Alibaba Cluster Trace 2018.
[2] Building an elastic query engine on disaggregated storage. (NSDI 20)

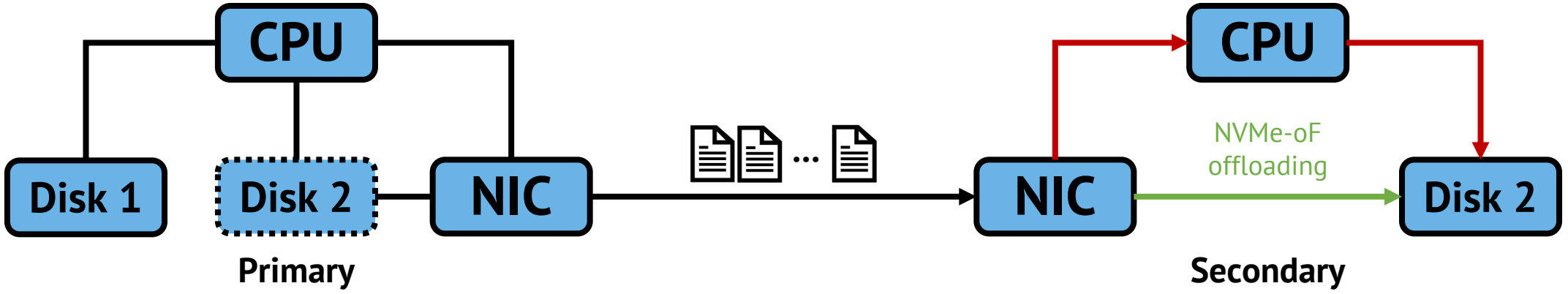
An attractive opportunity: NVMe-oF

- Non-Volatile Memory Express over Fabric
- Mount a remote disk as a local file system over RDMA or TCP



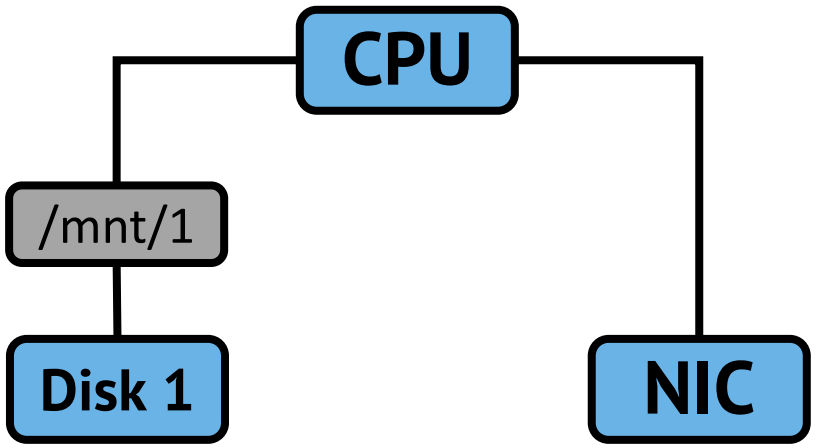
An attractive opportunity: NVMe-oF

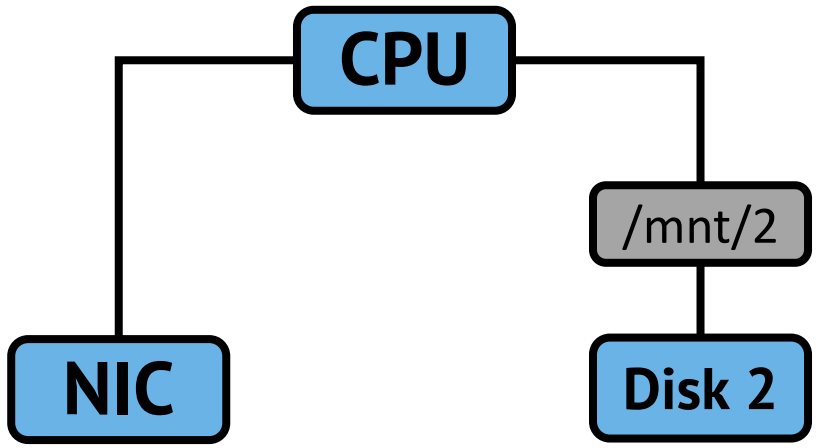
- Non-Volatile Memory Express over Fabric
- Mount a remote disk as a local file system over RDMA or TCP
- Zero CPU involvement on remote target
  - Commodity NICs support NVMe-oF target offloading



The challenge: NVMe-oF bypasses the remote filesystem

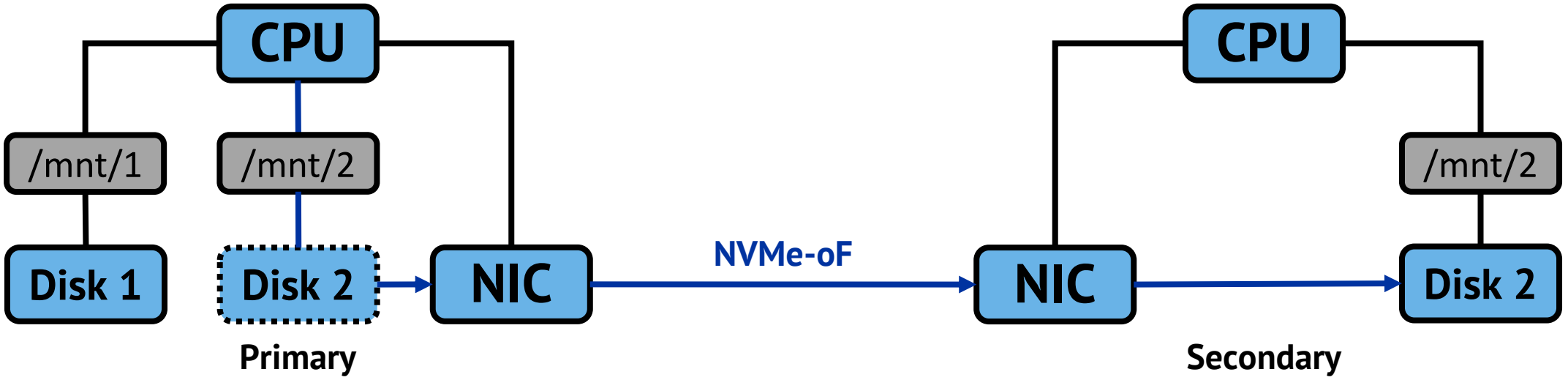
Secondaries cannot see incoming SST files





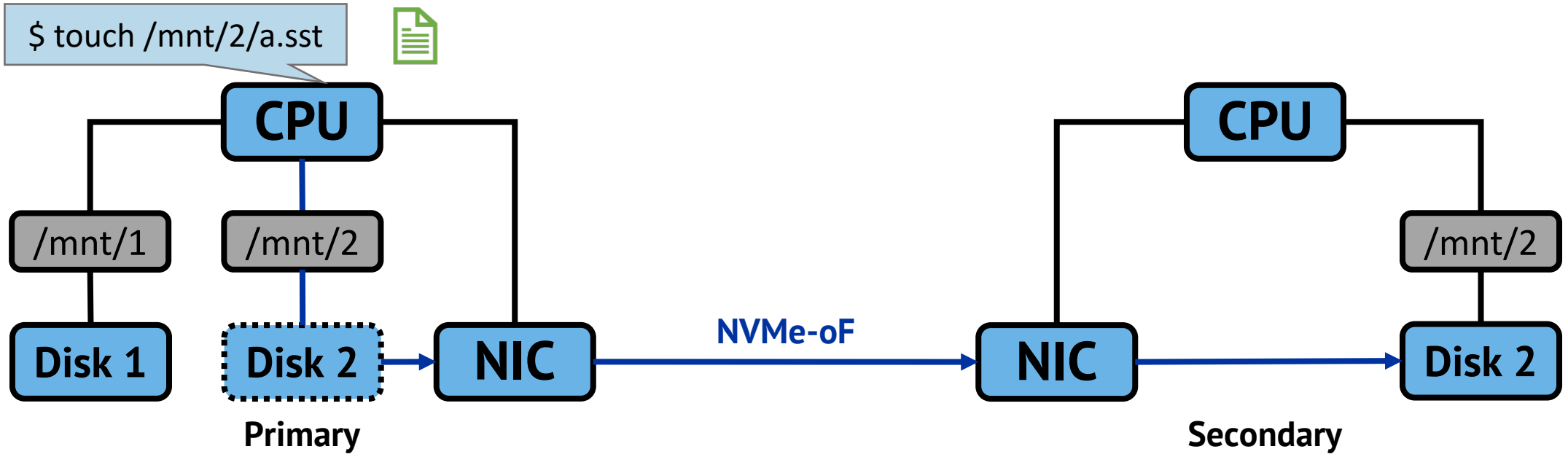
The challenge: NVMe-oF bypasses the remote filesystem

Secondaries cannot see incoming SST files



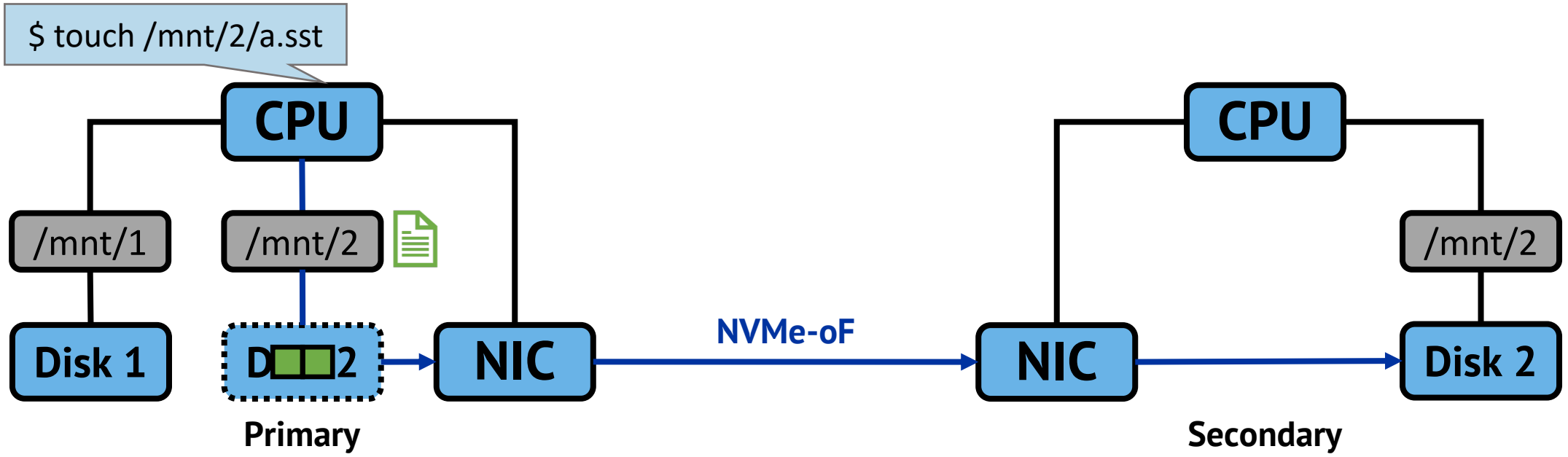
The challenge: NVMe-oF bypasses the remote filesystem

Secondaries cannot see incoming SST files



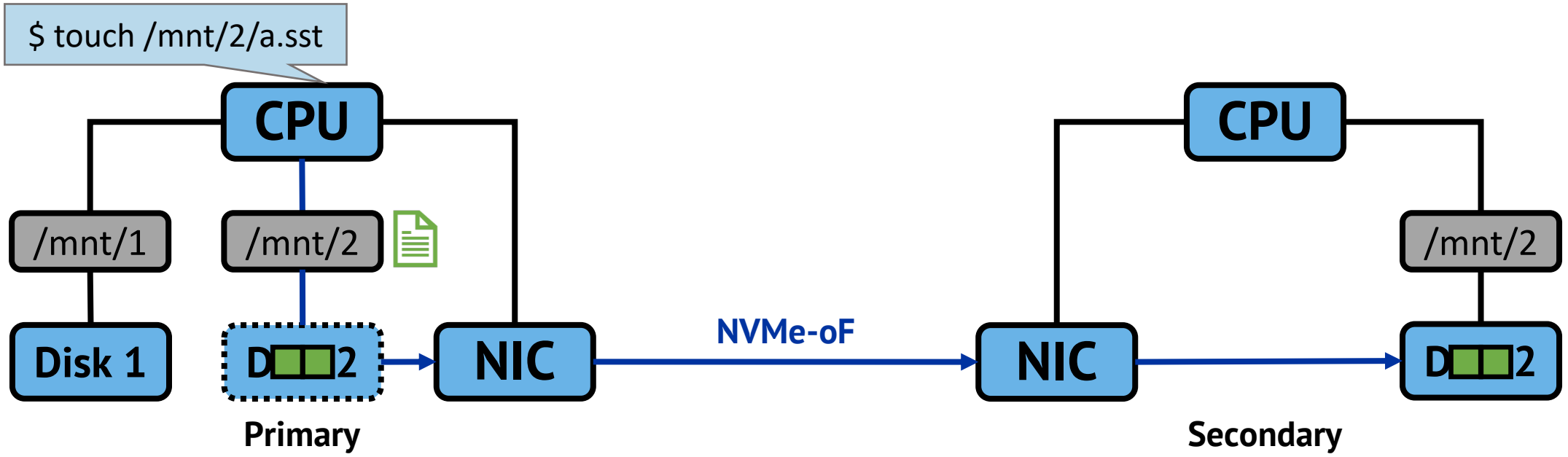
The challenge: NVMe-oF bypasses the remote filesystem

Secondaries cannot see incoming SST files



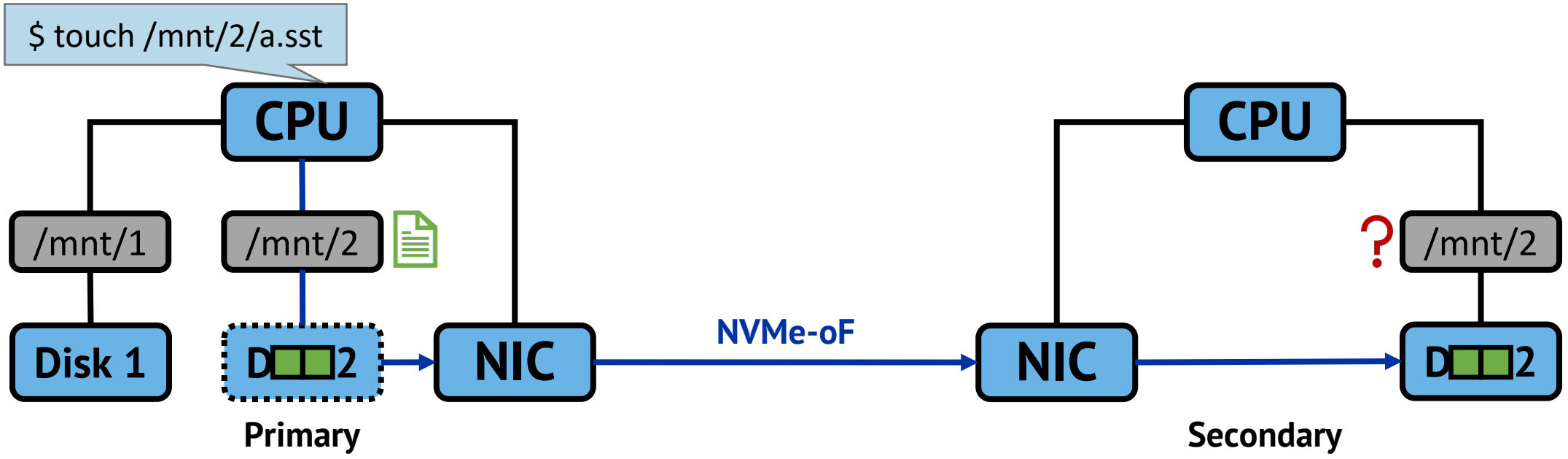
The challenge: NVMe-oF bypasses the remote filesystem

Secondaries cannot see incoming SST files



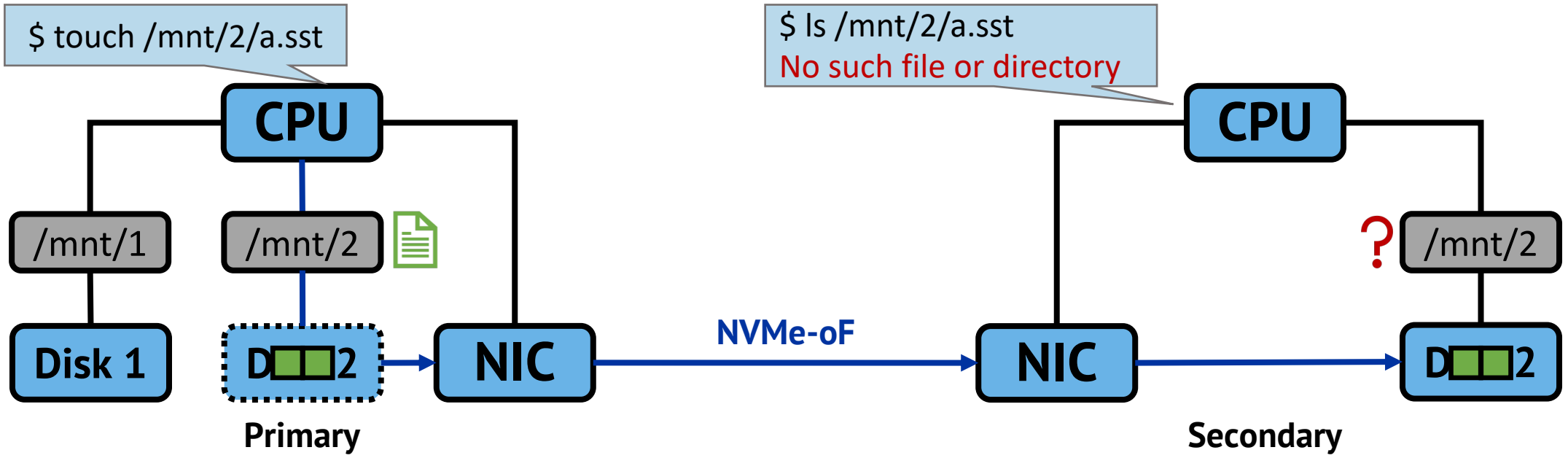
The challenge: NVMe-oF bypasses the remote filesystem

Secondaries cannot see incoming SST files



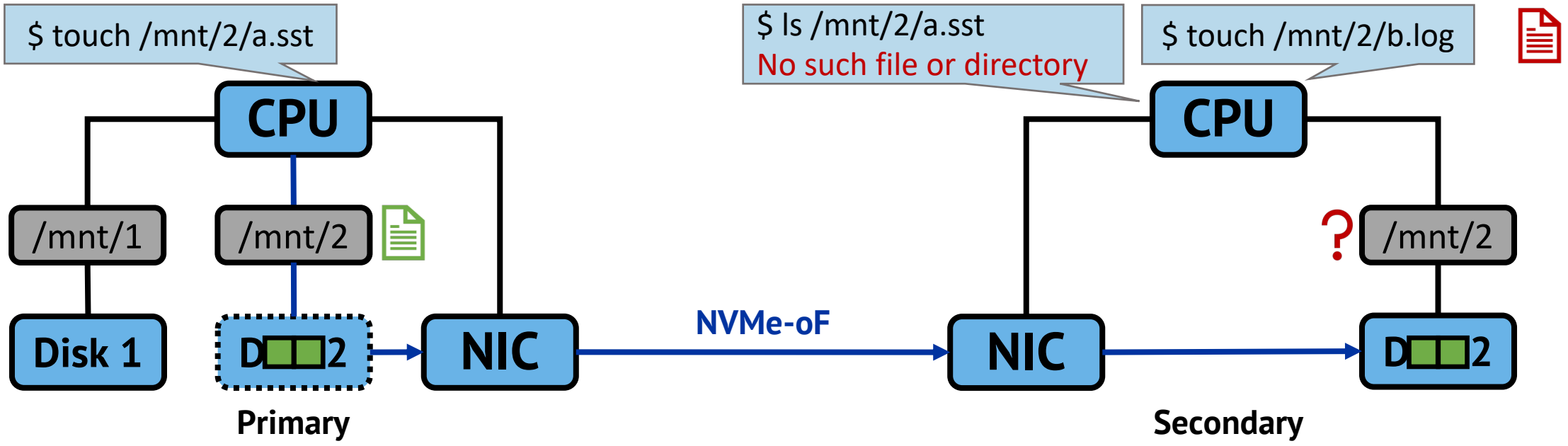
The challenge: NVMe-oF bypasses the remote filesystem

Secondaries cannot see incoming SST files



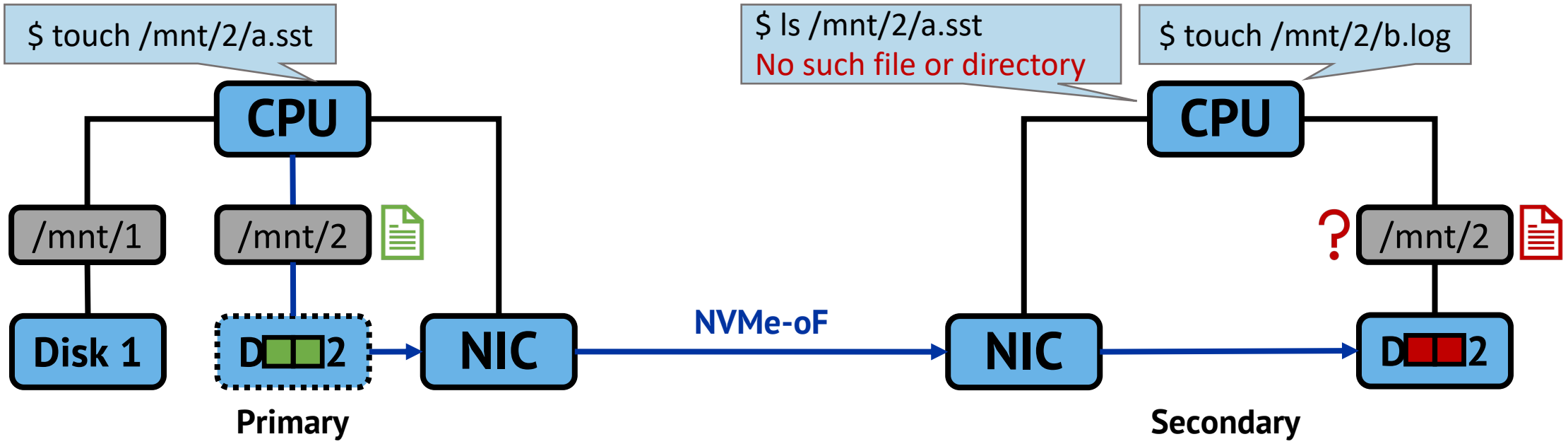
The challenge: NVMe-oF bypasses the remote filesystem

Secondaries cannot see incoming SST files and may **overwrite** them!

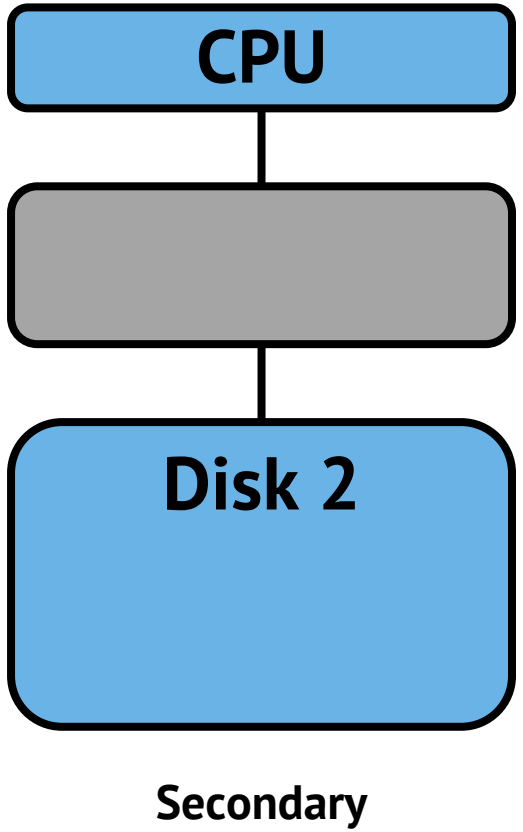


The challenge: NVMe-oF bypasses the remote filesystem

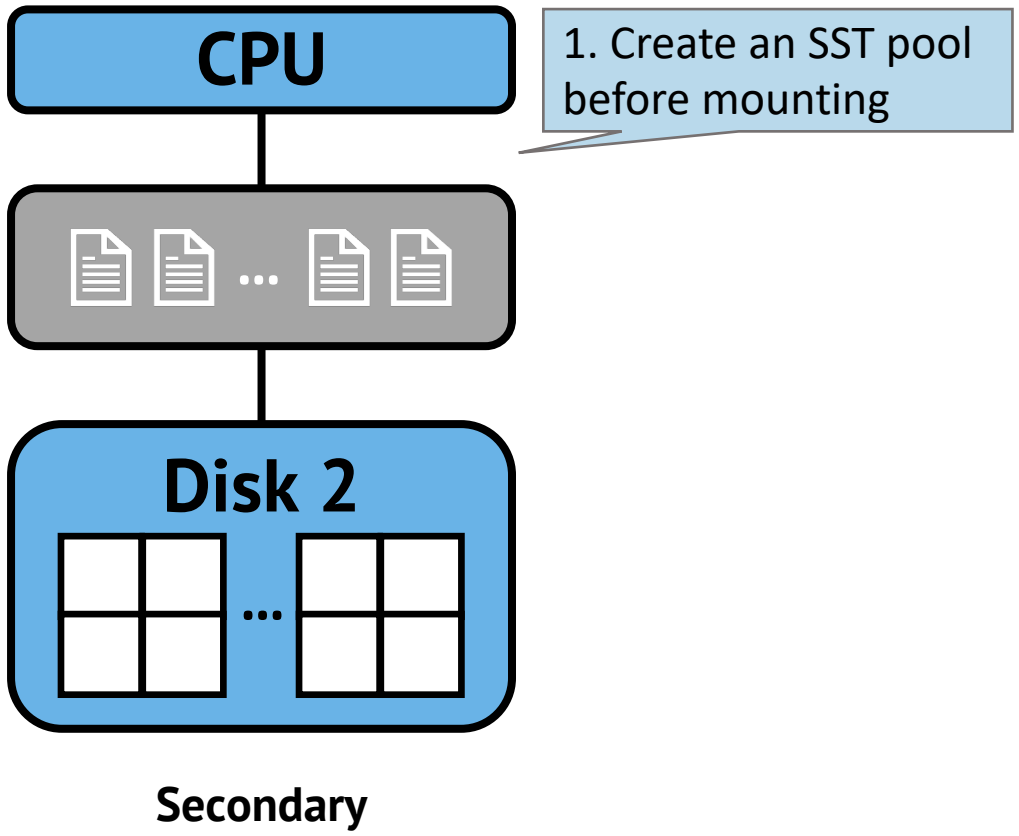
Secondaries cannot see incoming SST files and may **overwrite** them!



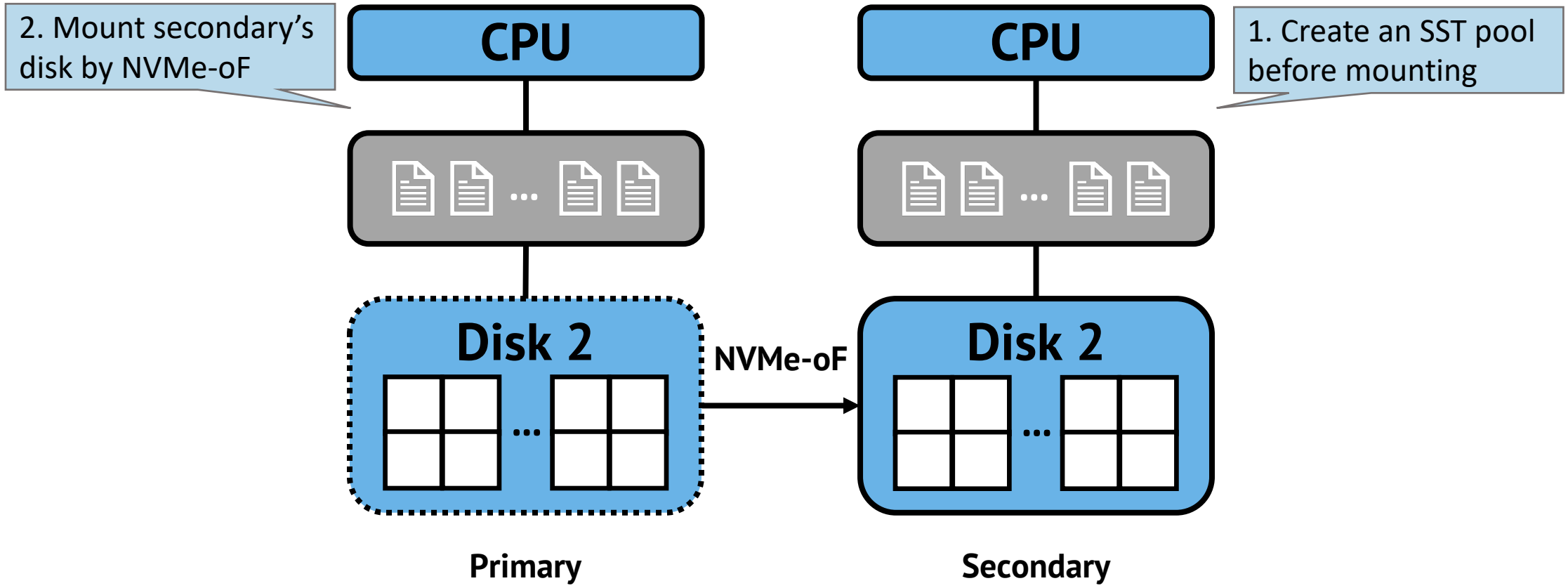
RubbleDB's approach: SST pre-allocation



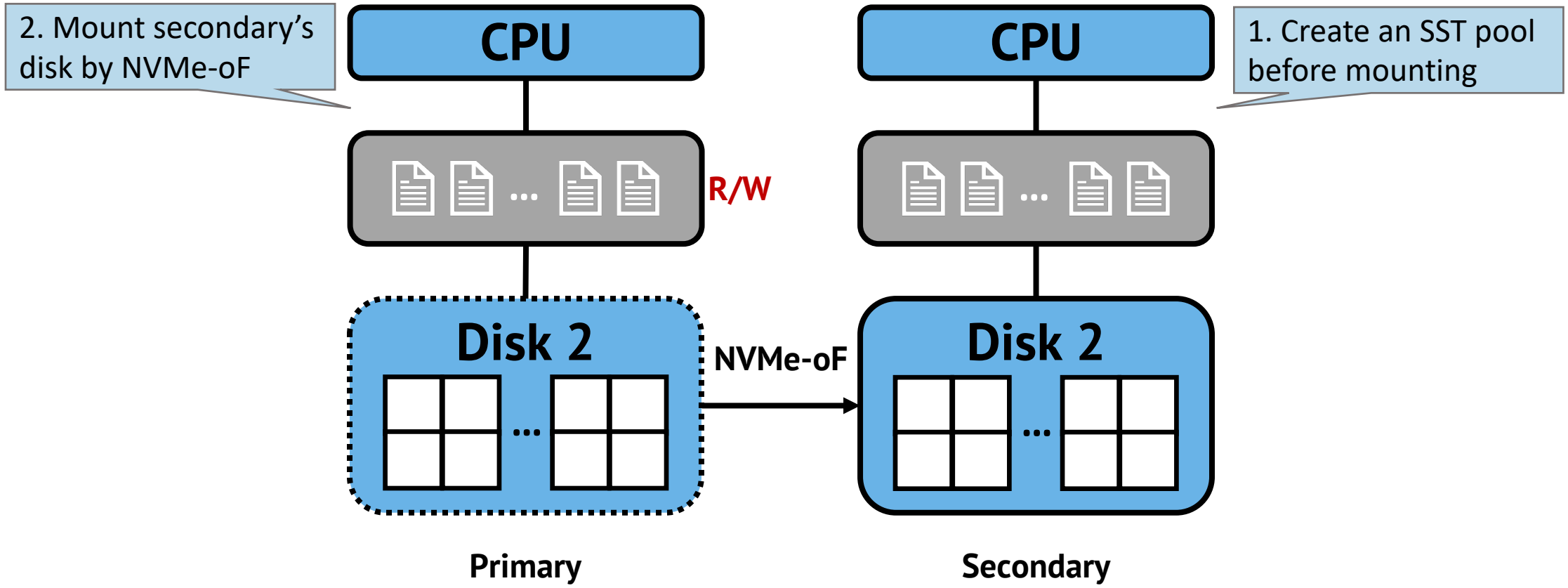
RubbleDB's approach: SST pre-allocation



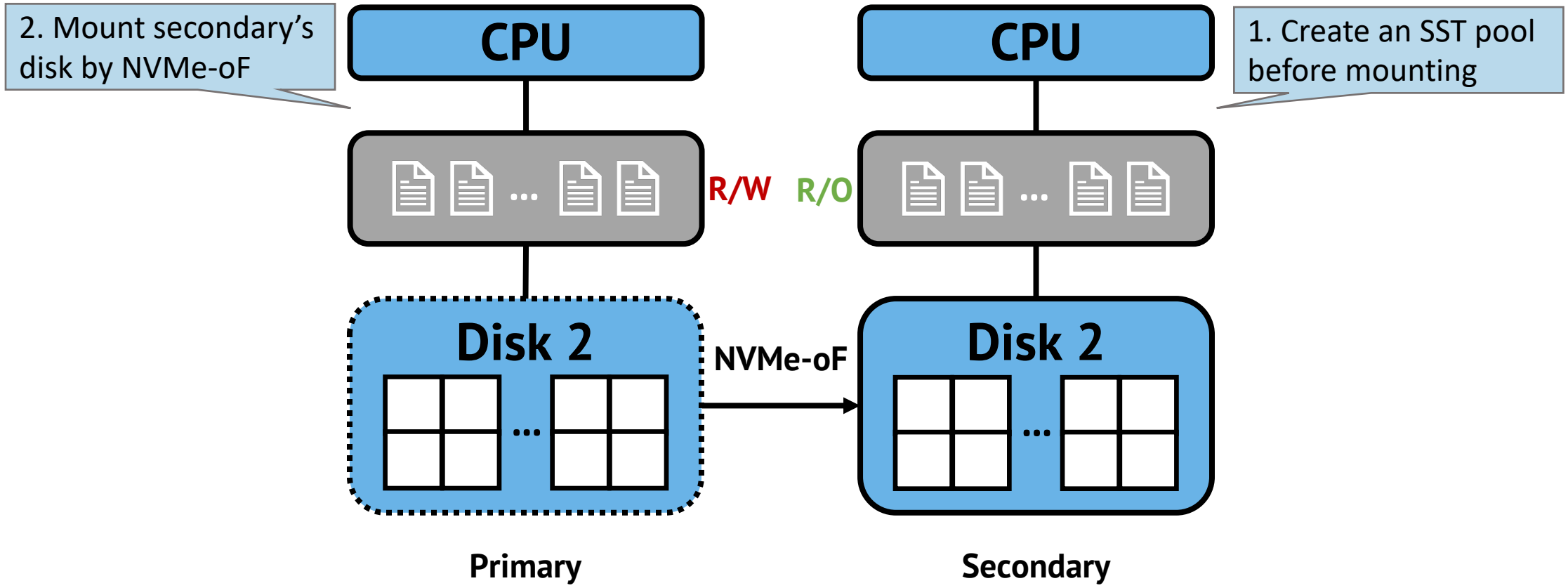
RubbleDB's approach: SST pre-allocation



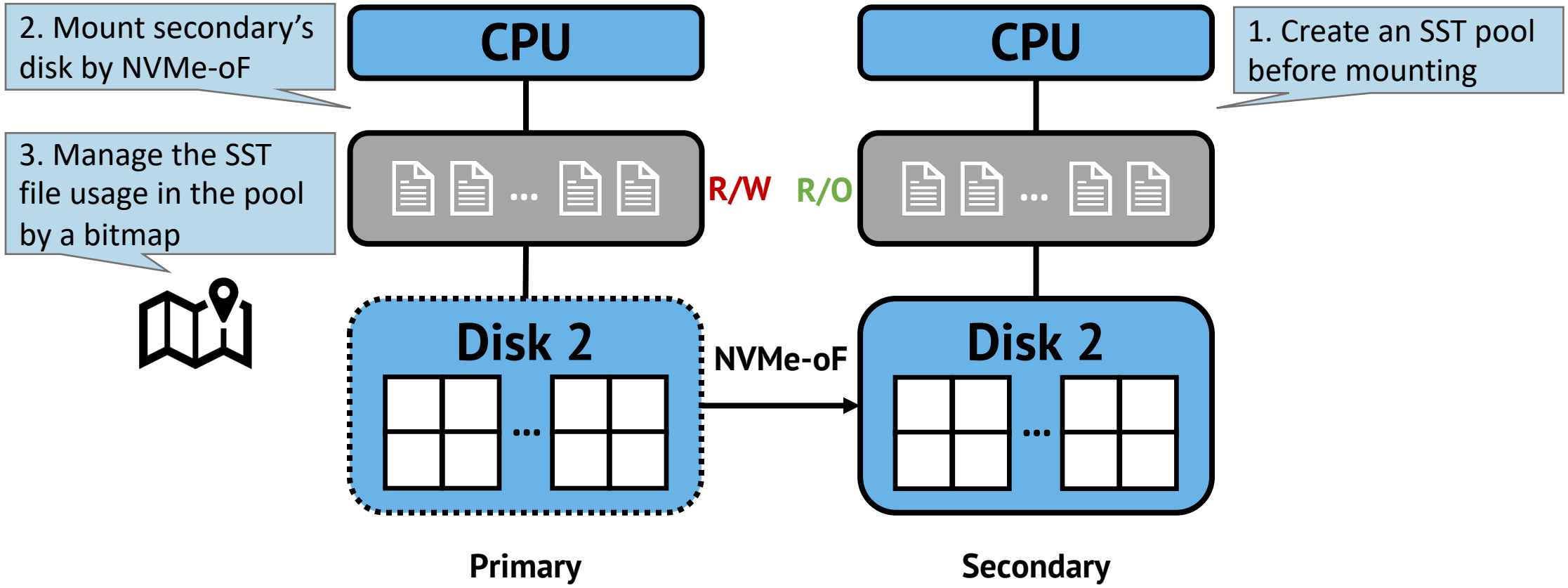
RubbleDB's approach: SST pre-allocation



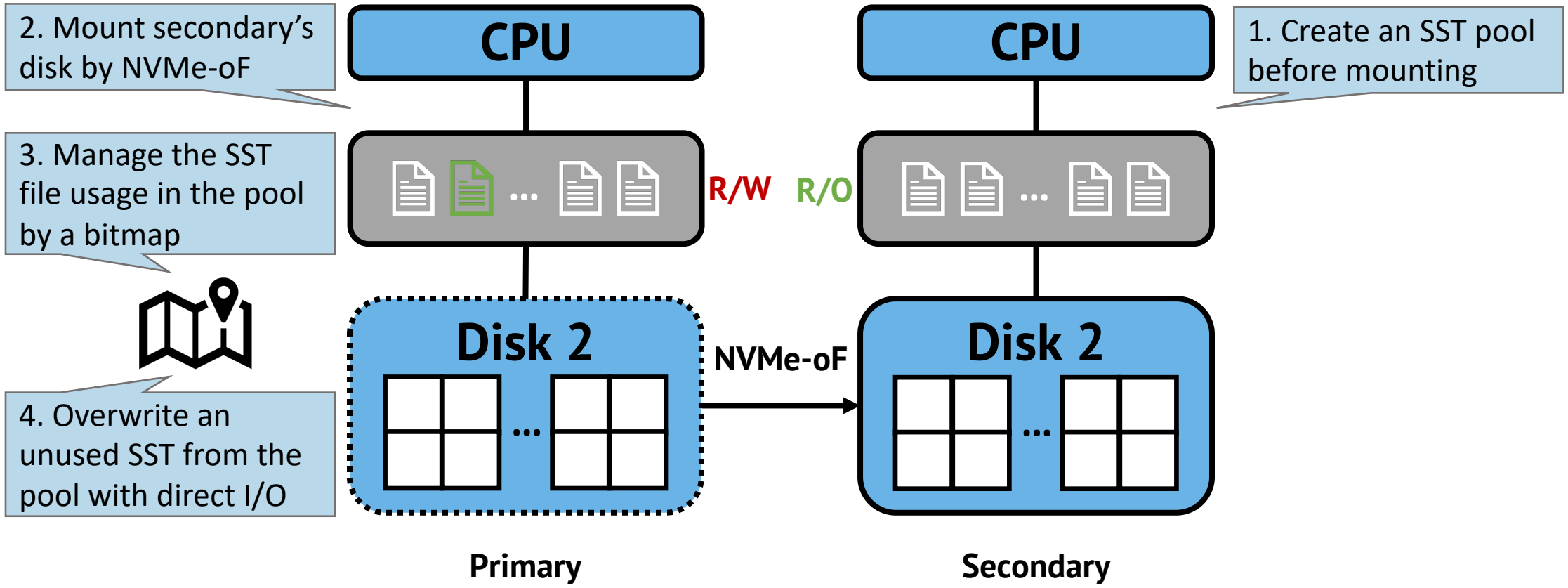
RubbleDB's approach: SST pre-allocation



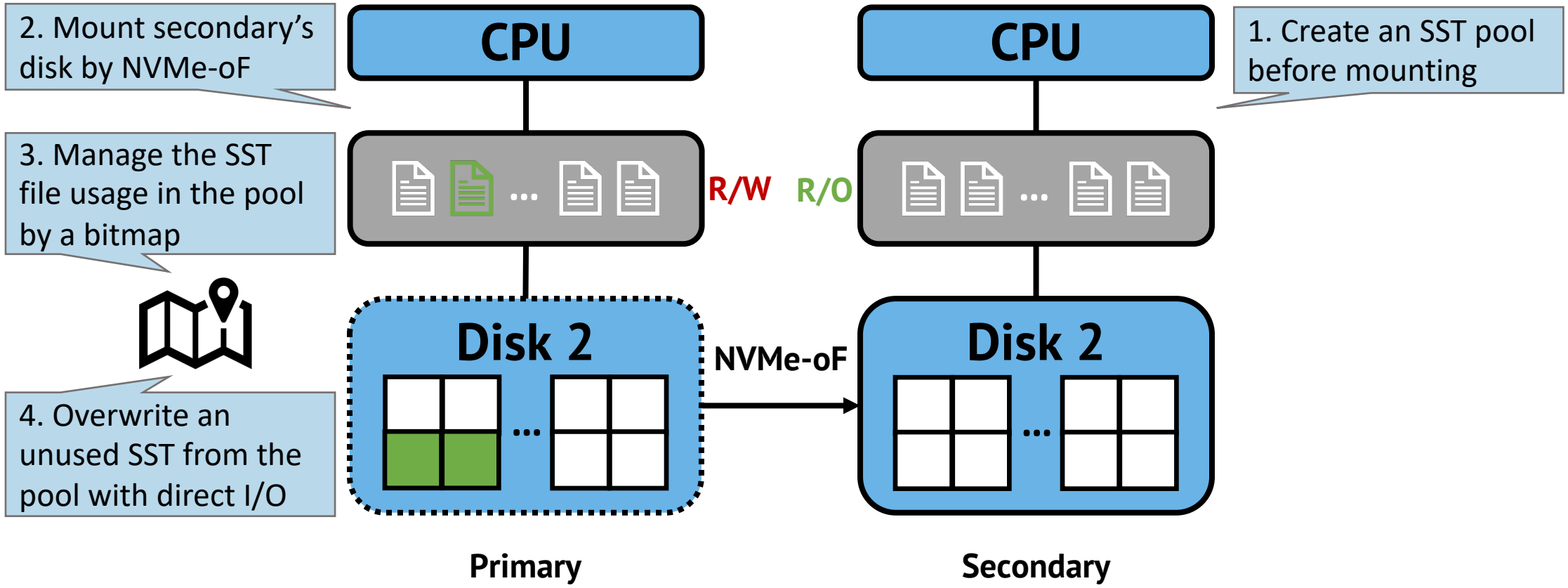
RubbleDB's approach: SST pre-allocation



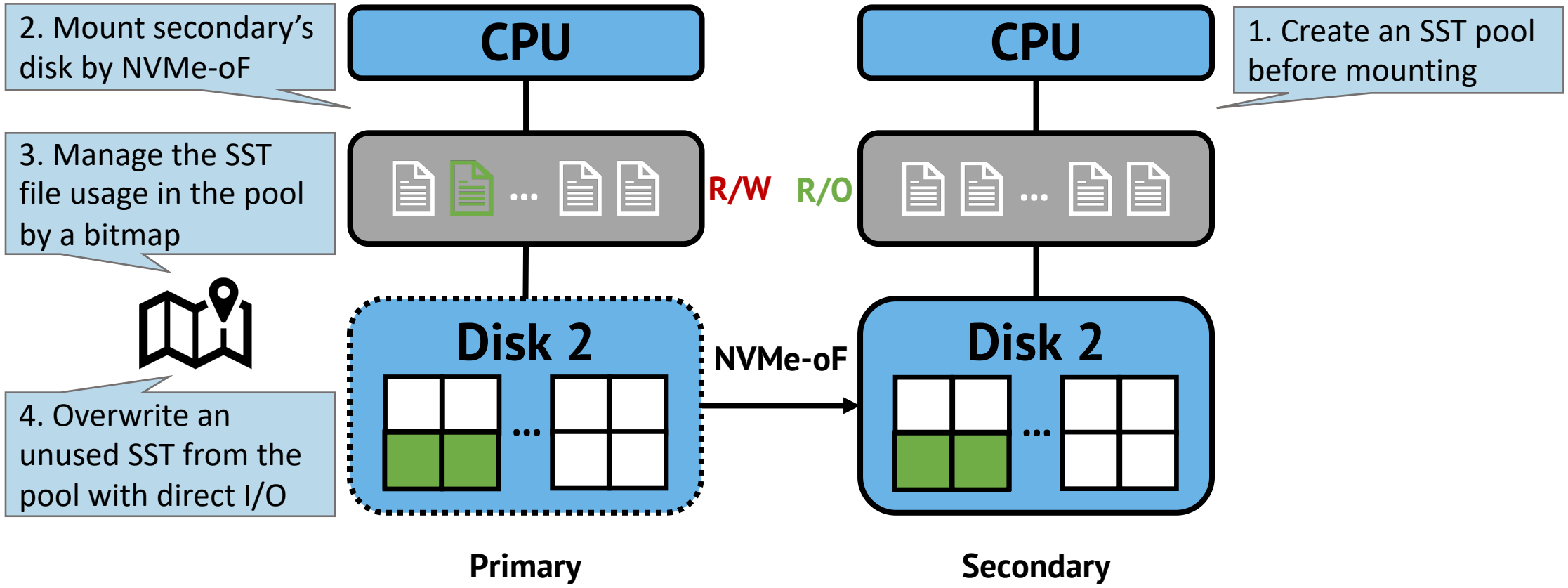
RubbleDB's approach: SST pre-allocation



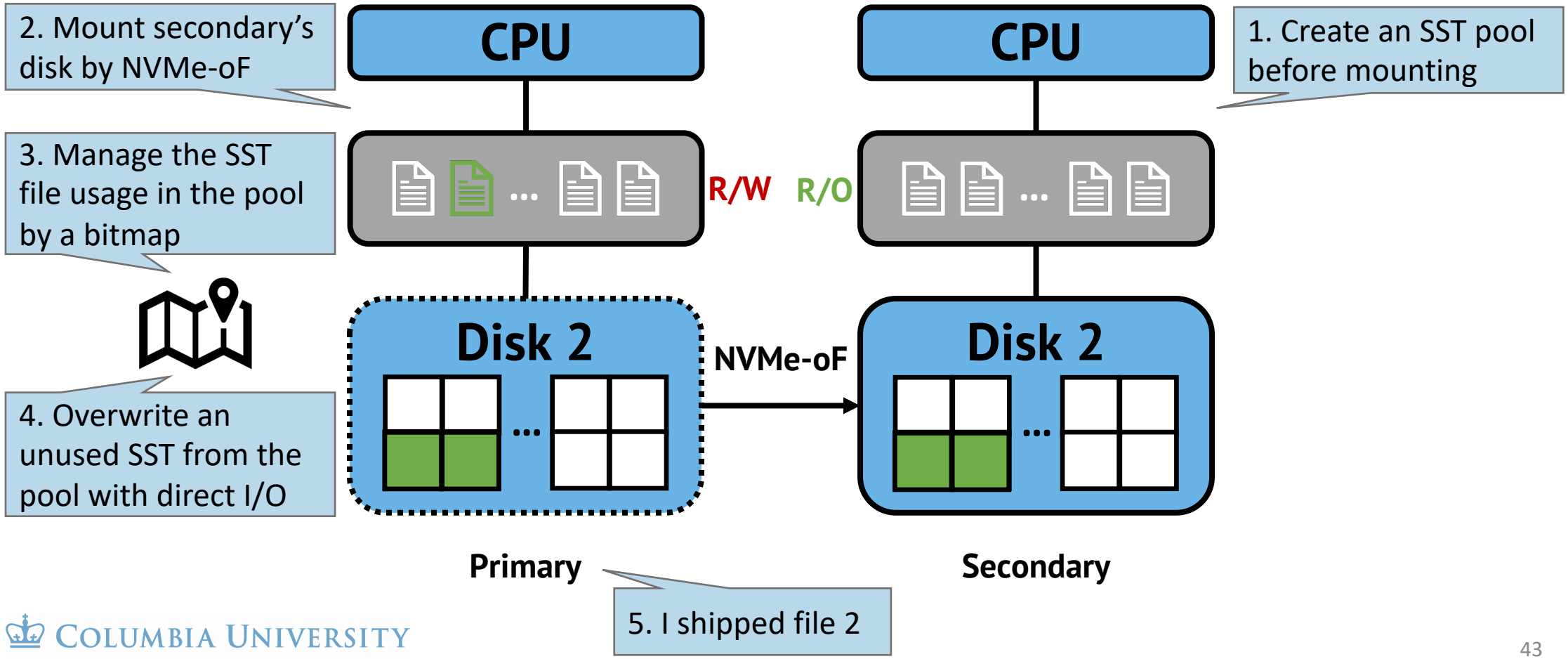
RubbleDB's approach: SST pre-allocation



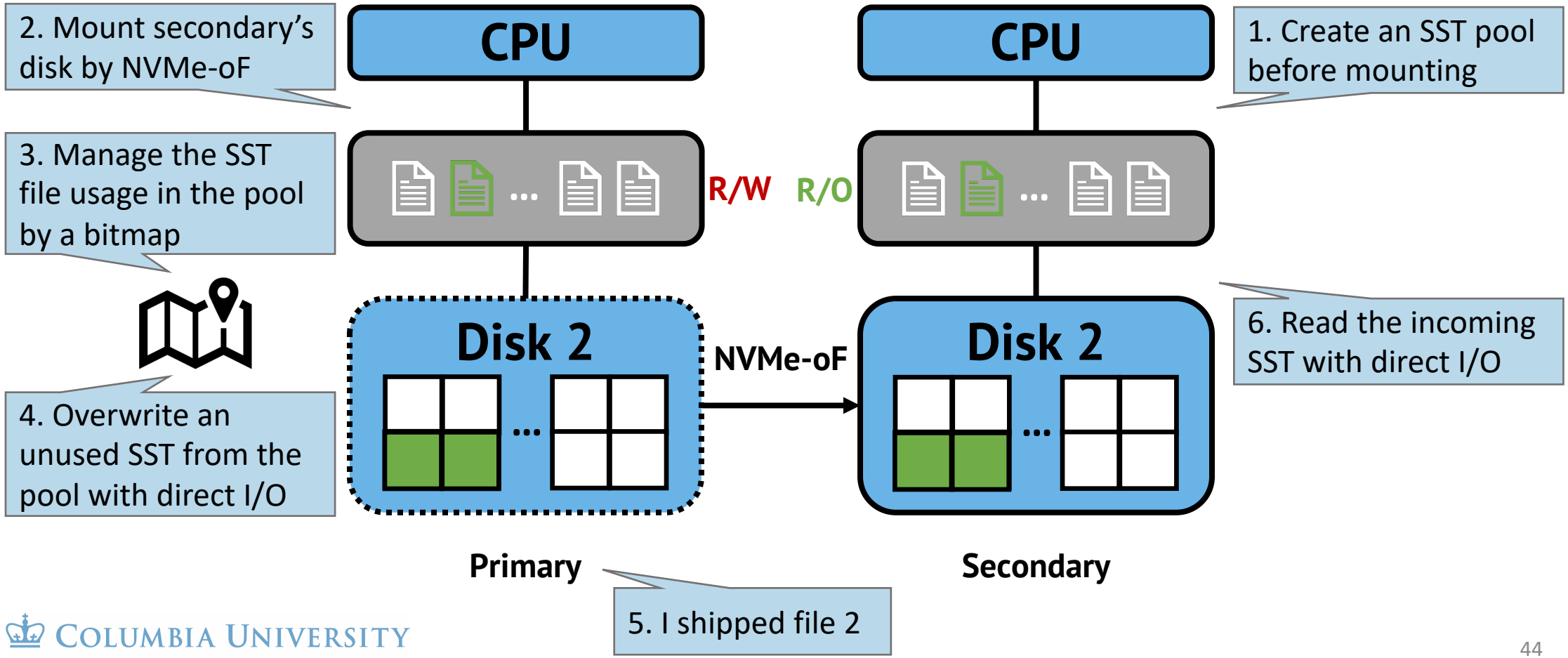
RubbleDB's approach: SST pre-allocation



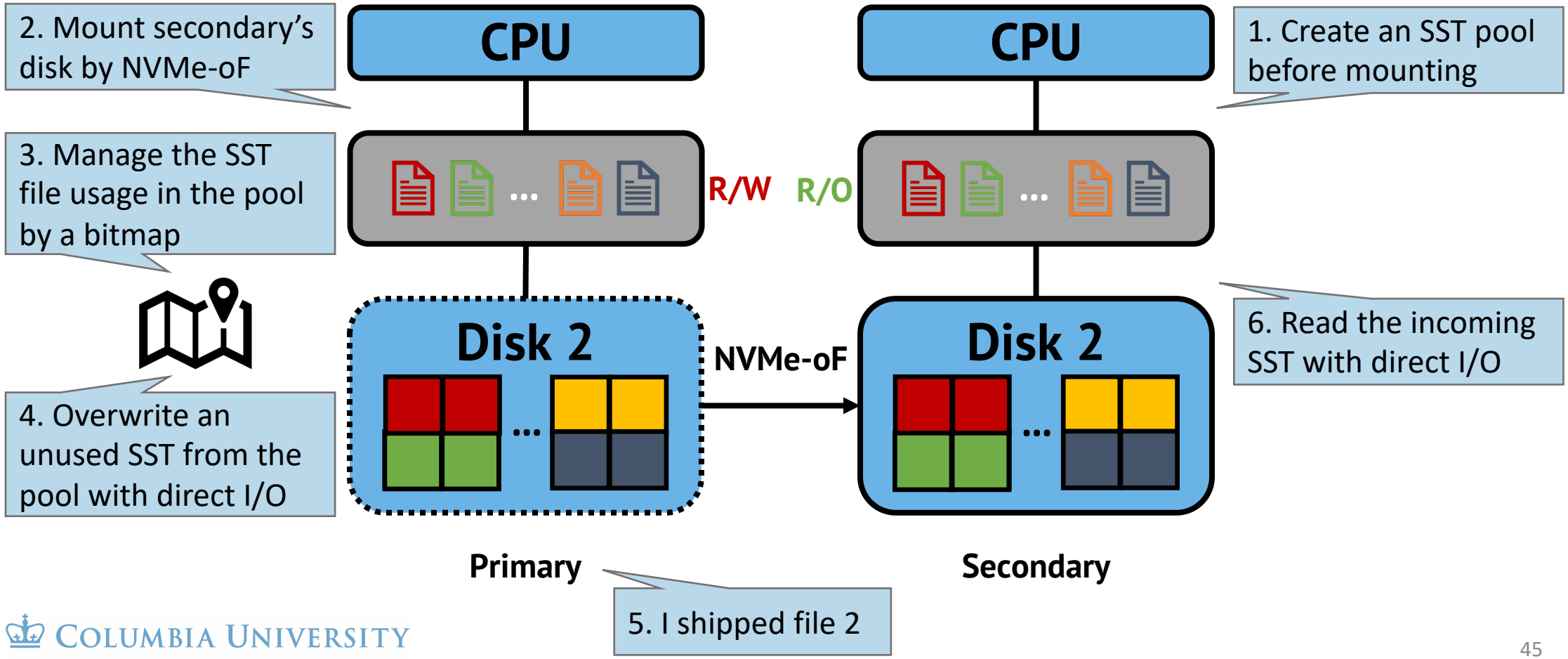
RubbleDB's approach: SST pre-allocation



RubbleDB's approach: SST pre-allocation



RubbleDB's approach: SST pre-allocation



Shared SSTs causes replica inconsistency

Primary' and secondaries' internal states are actually different

Shared SSTs causes replica inconsistency

Primary' and secondaries' internal states are actually different

MemTable 1

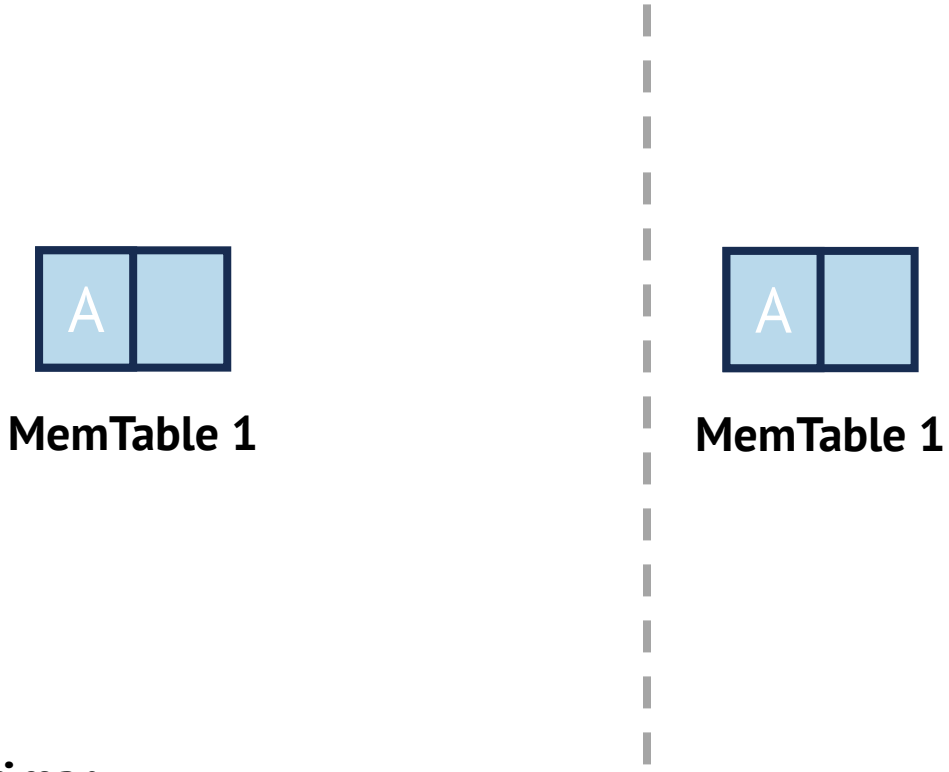
MemTable 1

Primary

Secondary

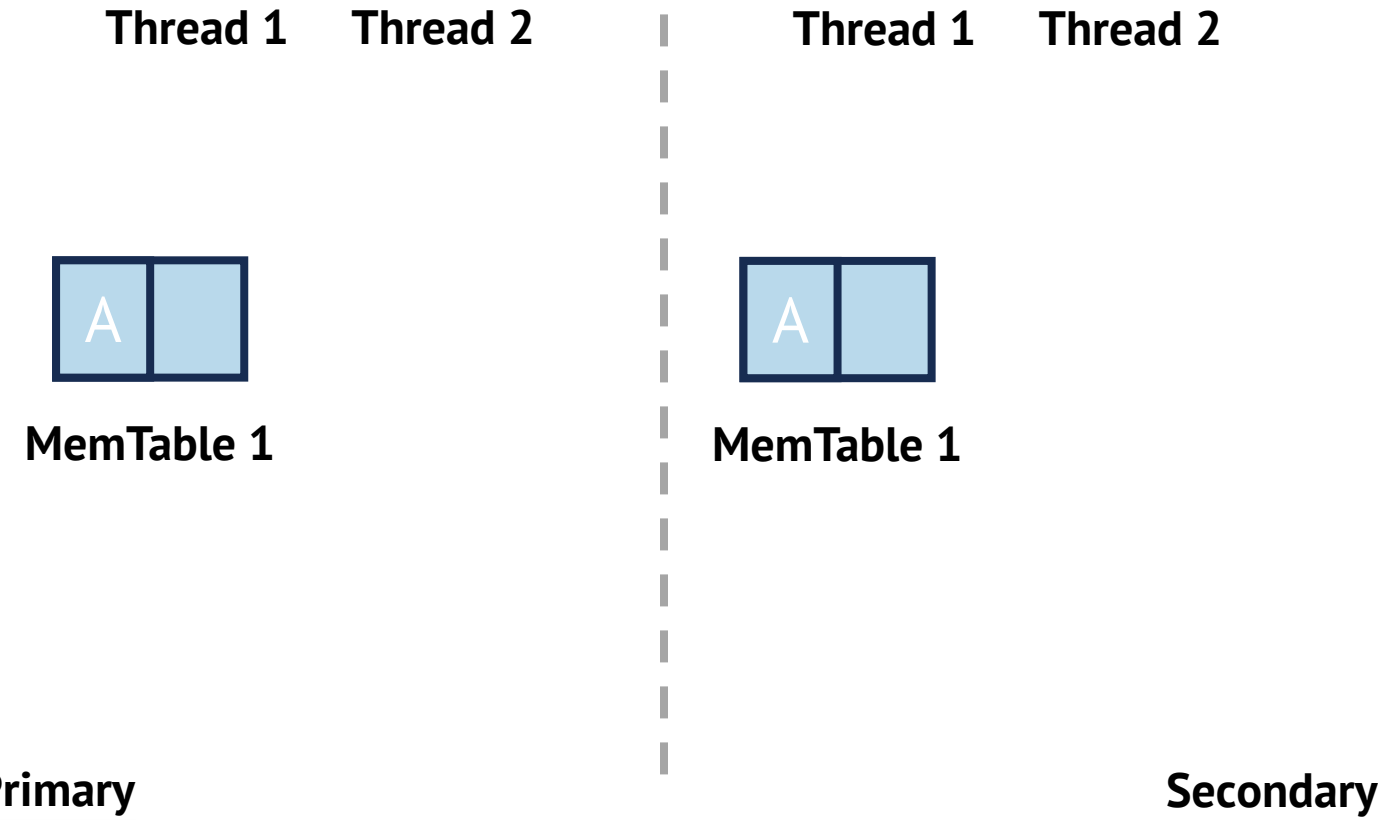
Shared SSTs causes replica inconsistency

Primary' and secondaries' internal states are actually different



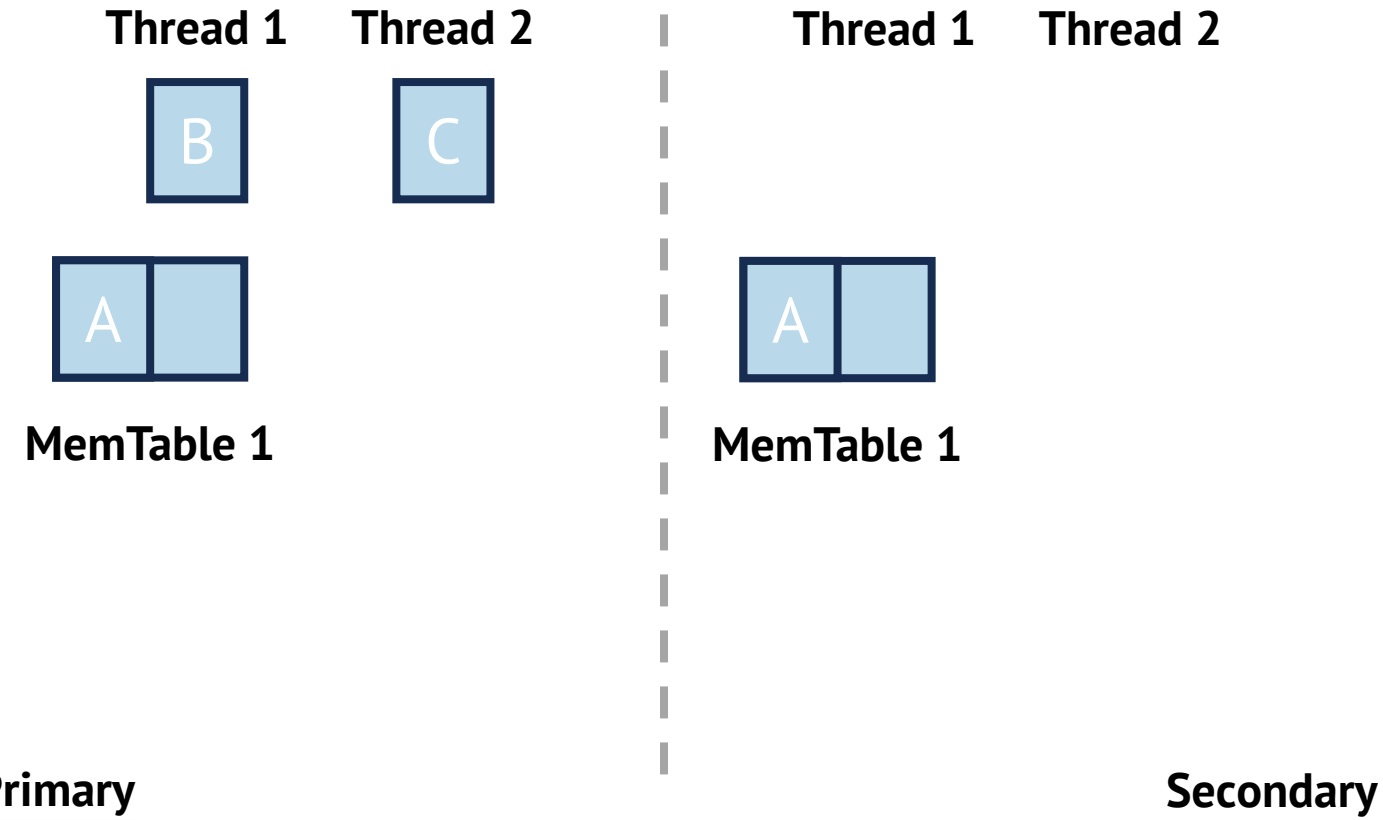
Shared SSTs causes replica inconsistency

Primary' and secondaries' internal states are actually different



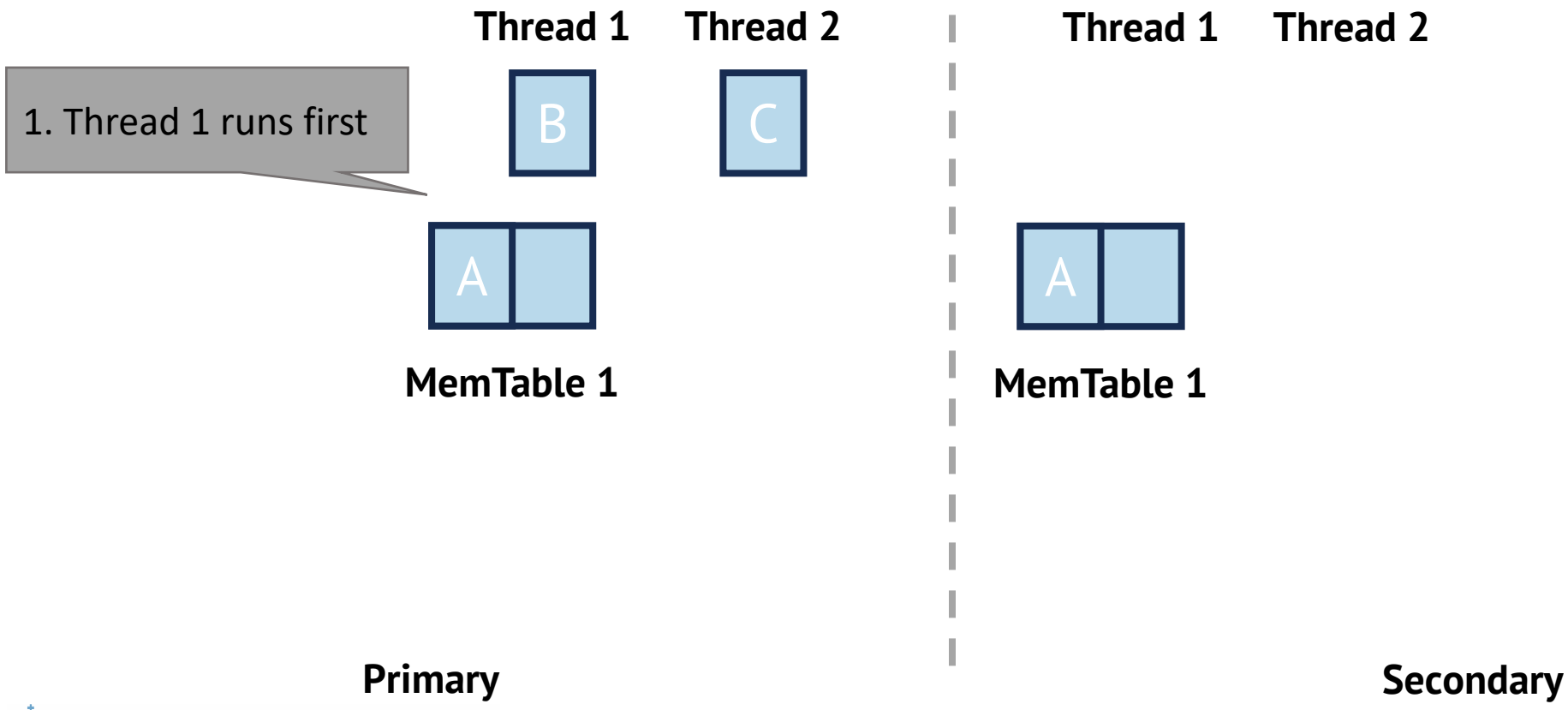
Shared SSTs causes replica inconsistency

Primary' and secondaries' internal states are actually different



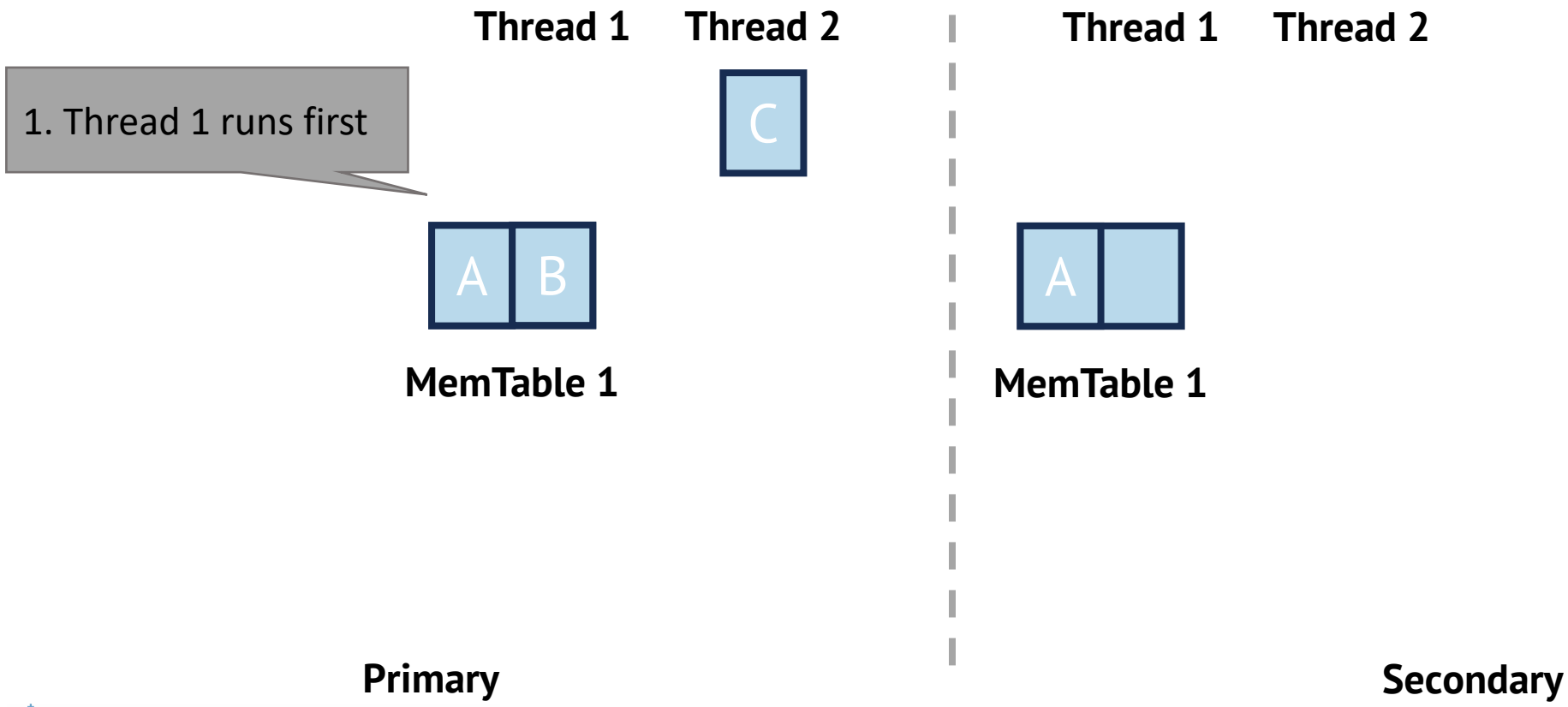
Shared SSTs causes replica inconsistency

Primary' and secondaries' internal states are actually different



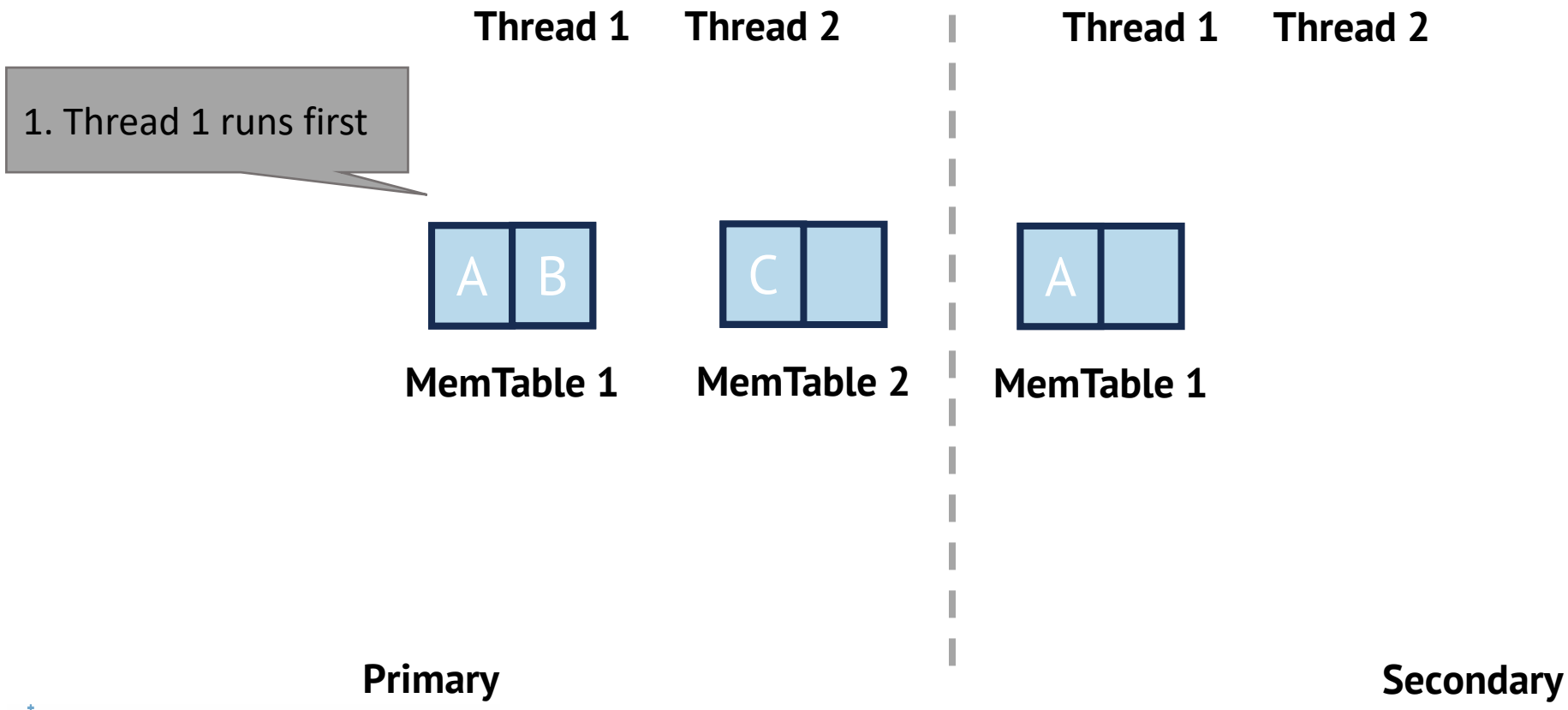
Shared SSTs causes replica inconsistency

Primary' and secondaries' internal states are actually different



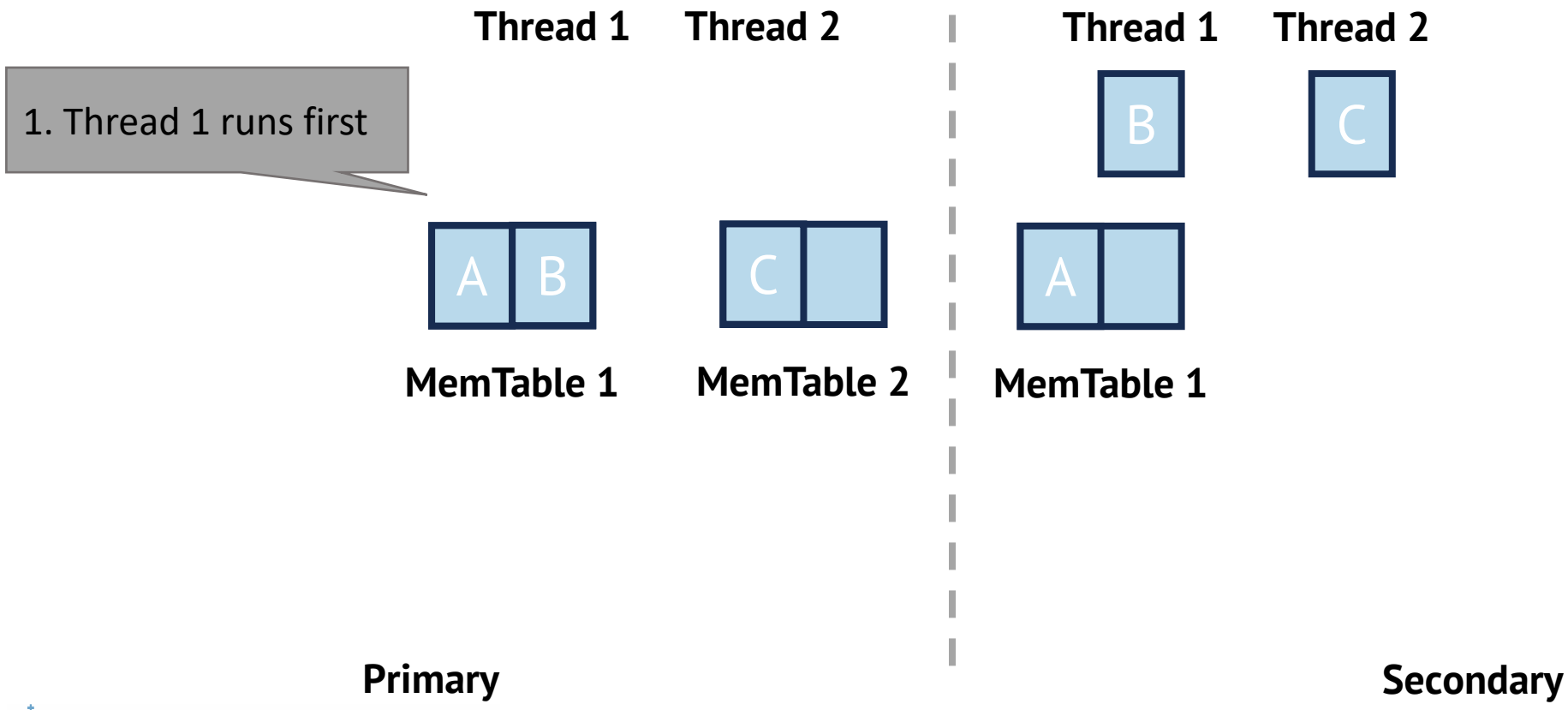
Shared SSTs causes replica inconsistency

Primary' and secondaries' internal states are actually different



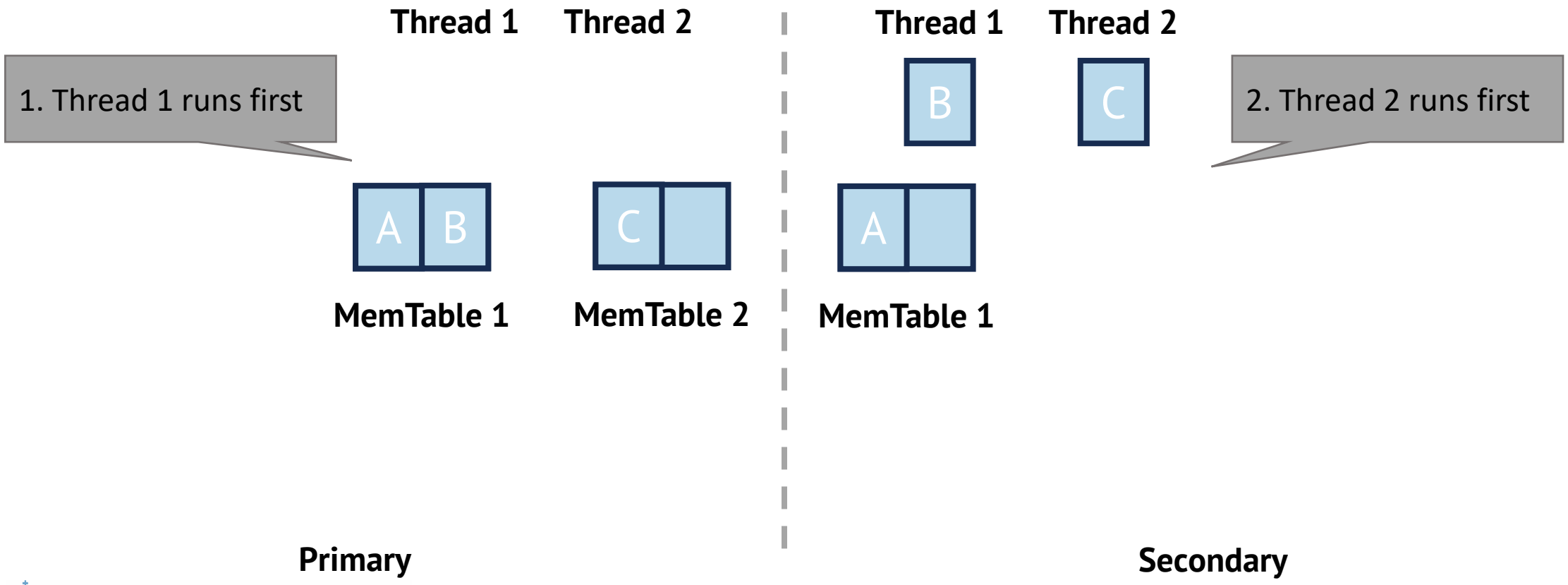
Shared SSTs causes replica inconsistency

Primary' and secondaries' internal states are actually different



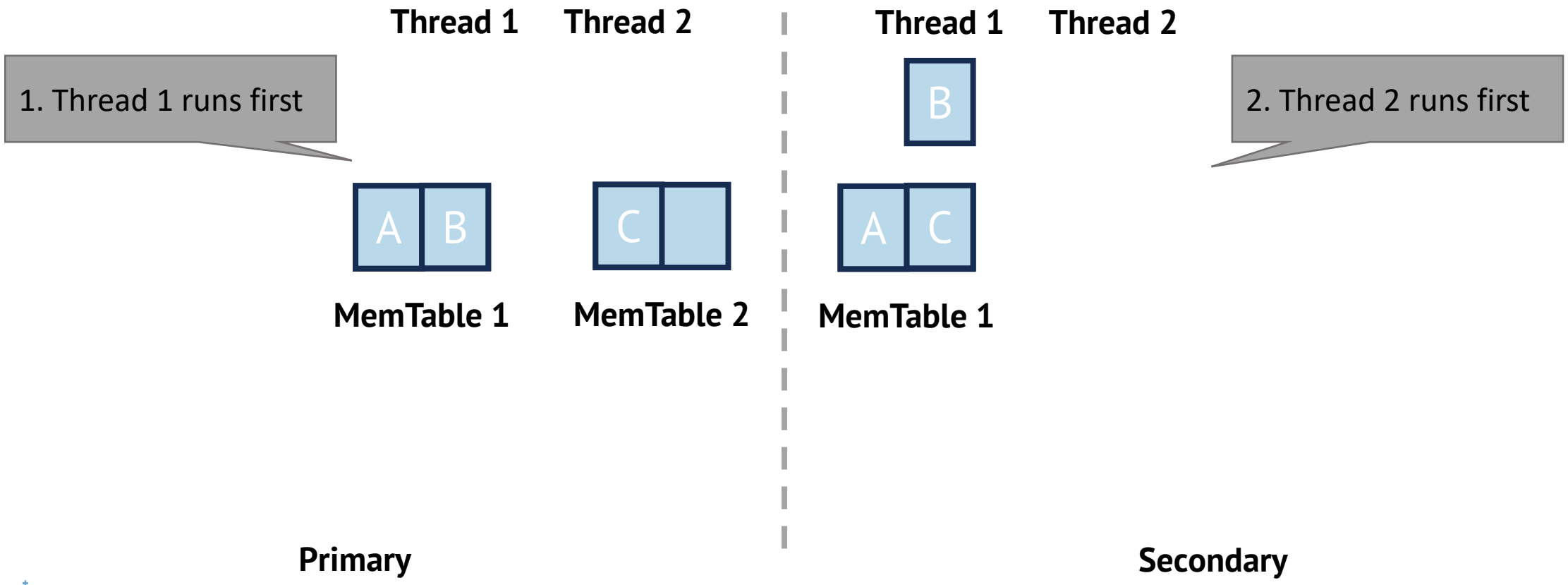
Shared SSTs causes replica inconsistency

Primary' and secondaries' internal states are actually different



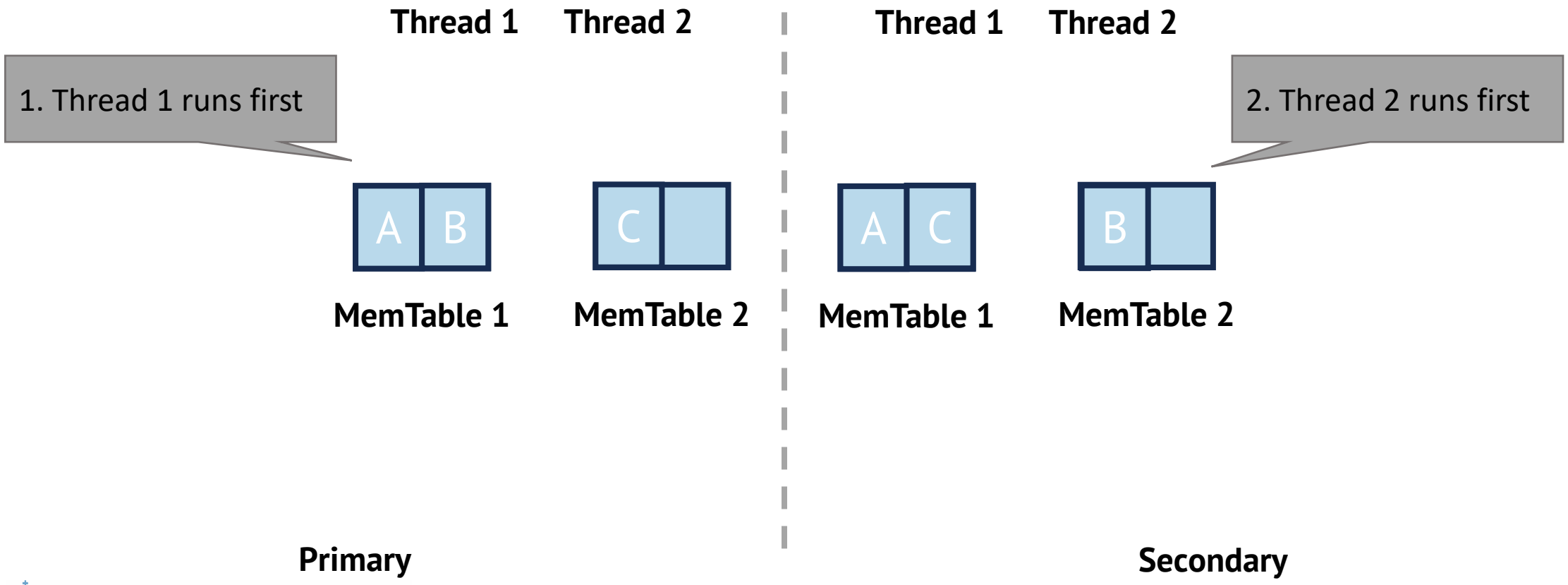
Shared SSTs causes replica inconsistency

Primary' and secondaries' internal states are actually different



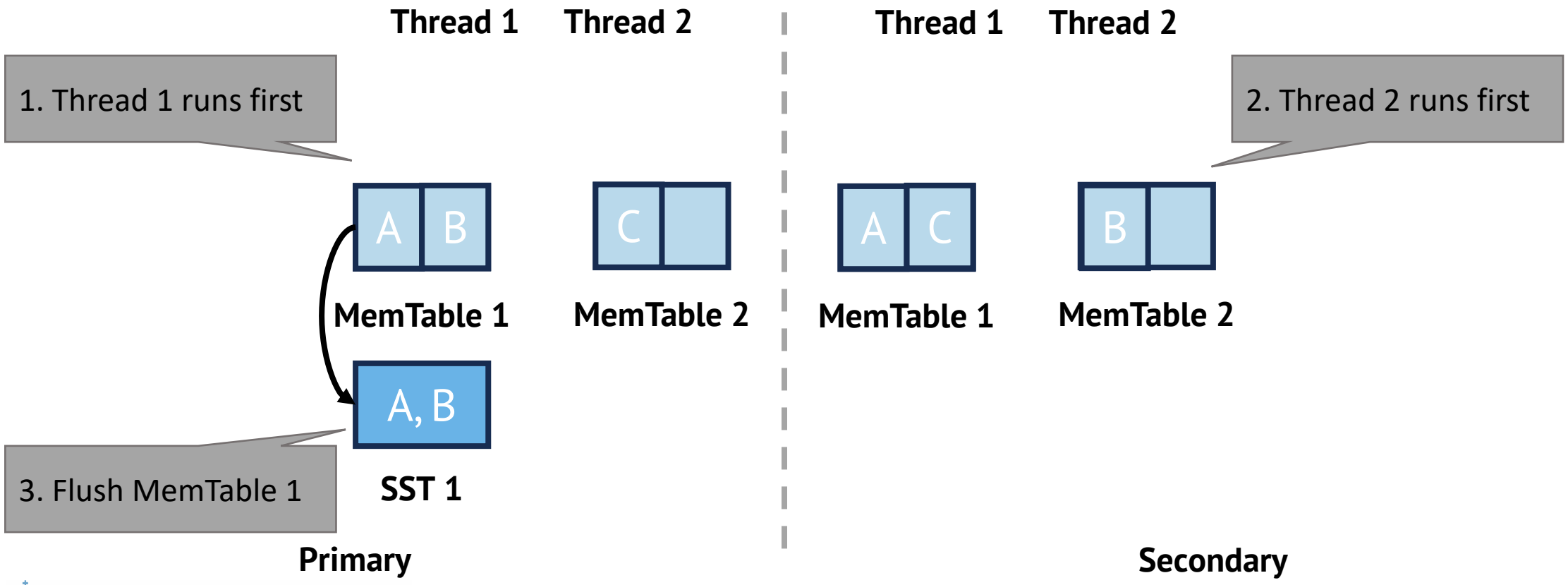
Shared SSTs causes replica inconsistency

Primary' and secondaries' internal states are actually different



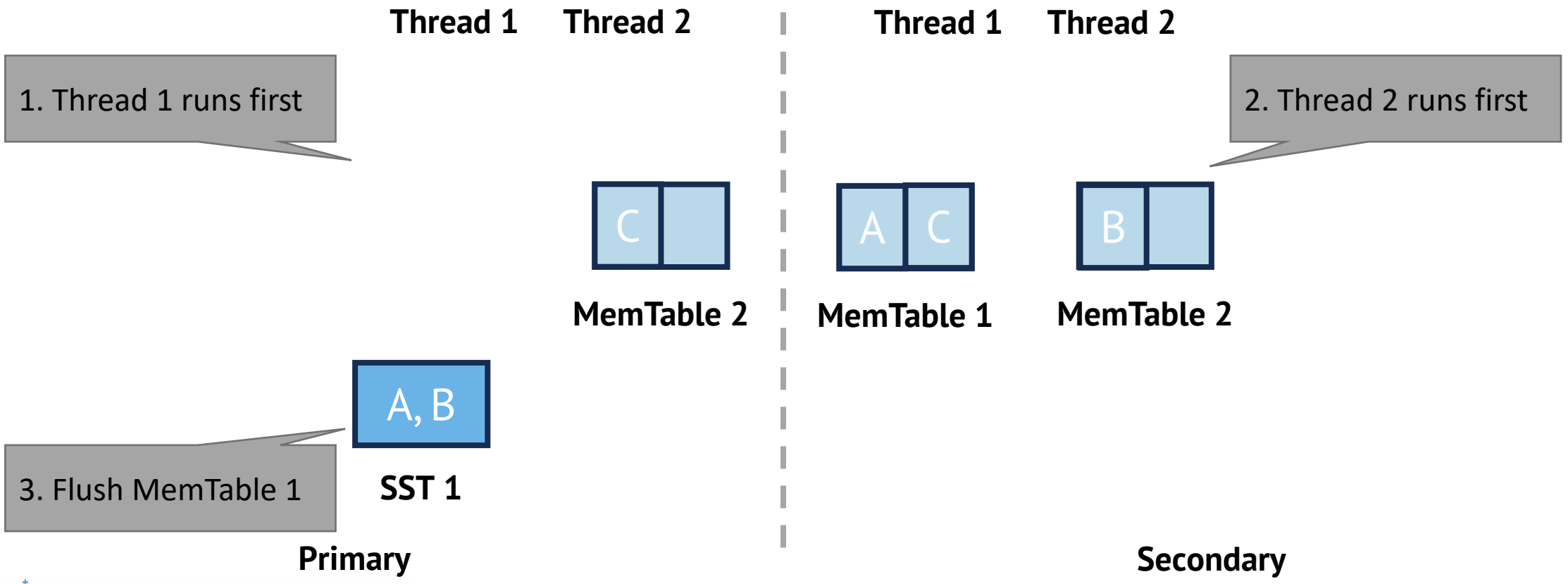
Shared SSTs causes replica inconsistency

Primary' and secondaries' internal states are actually different



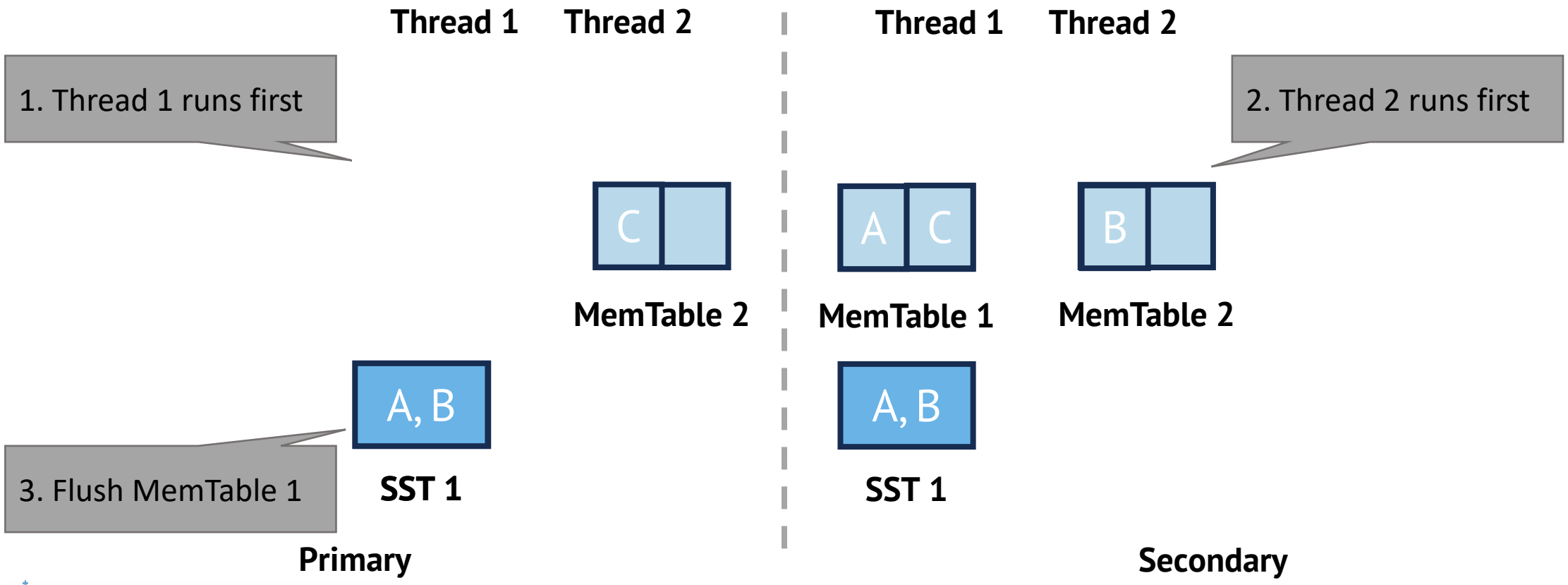
Shared SSTs causes replica inconsistency

Primary' and secondaries' internal states are actually different



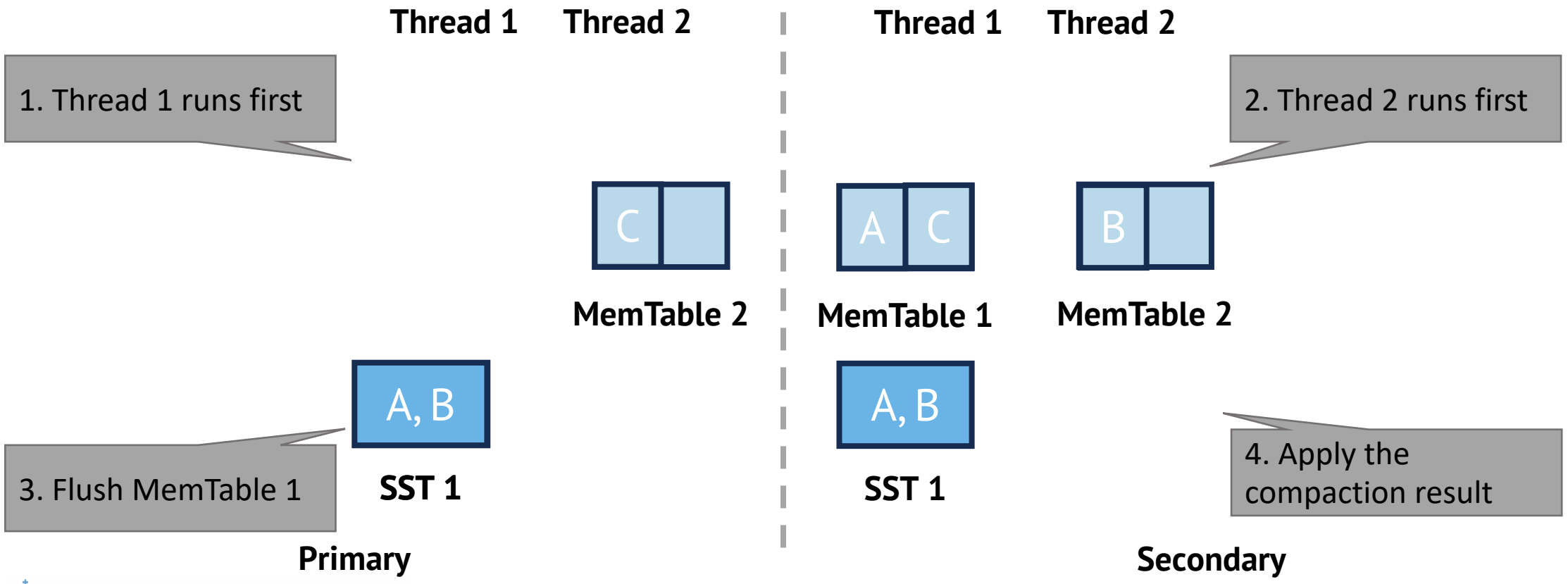
Shared SSTs causes replica inconsistency

Primary' and secondaries' internal states are actually different



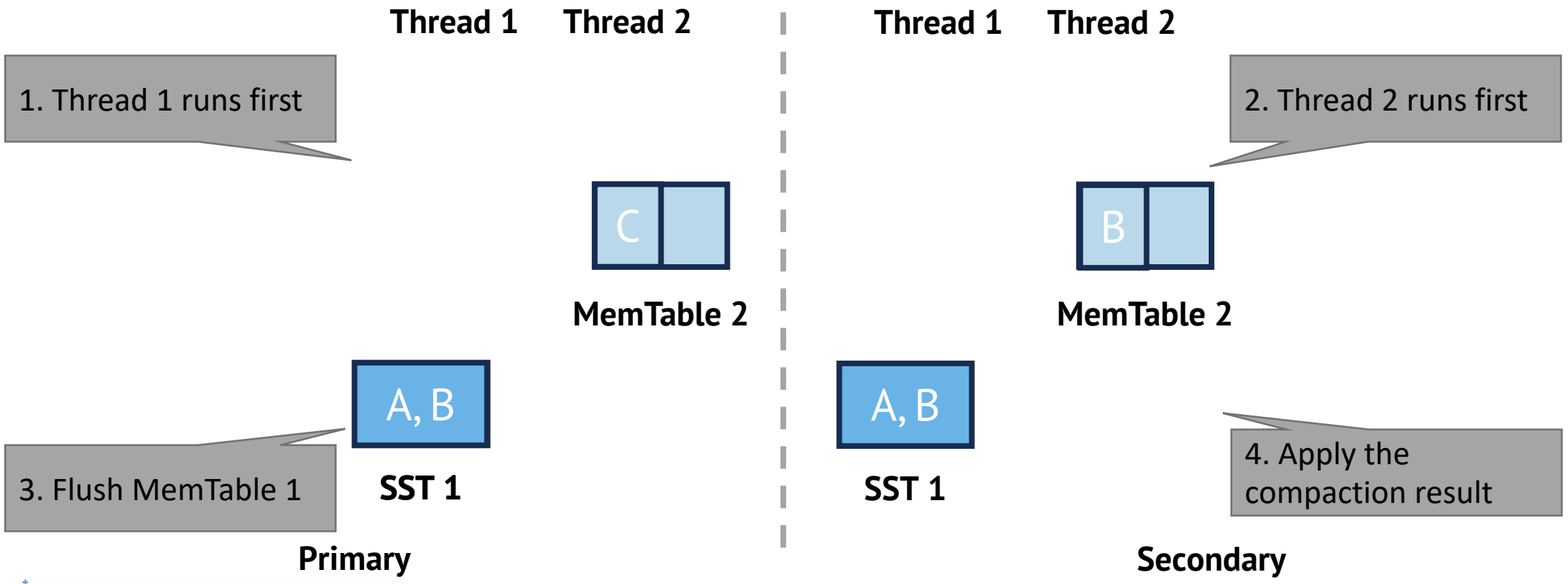
Shared SSTs causes replica inconsistency

Primary' and secondaries' internal states are actually different



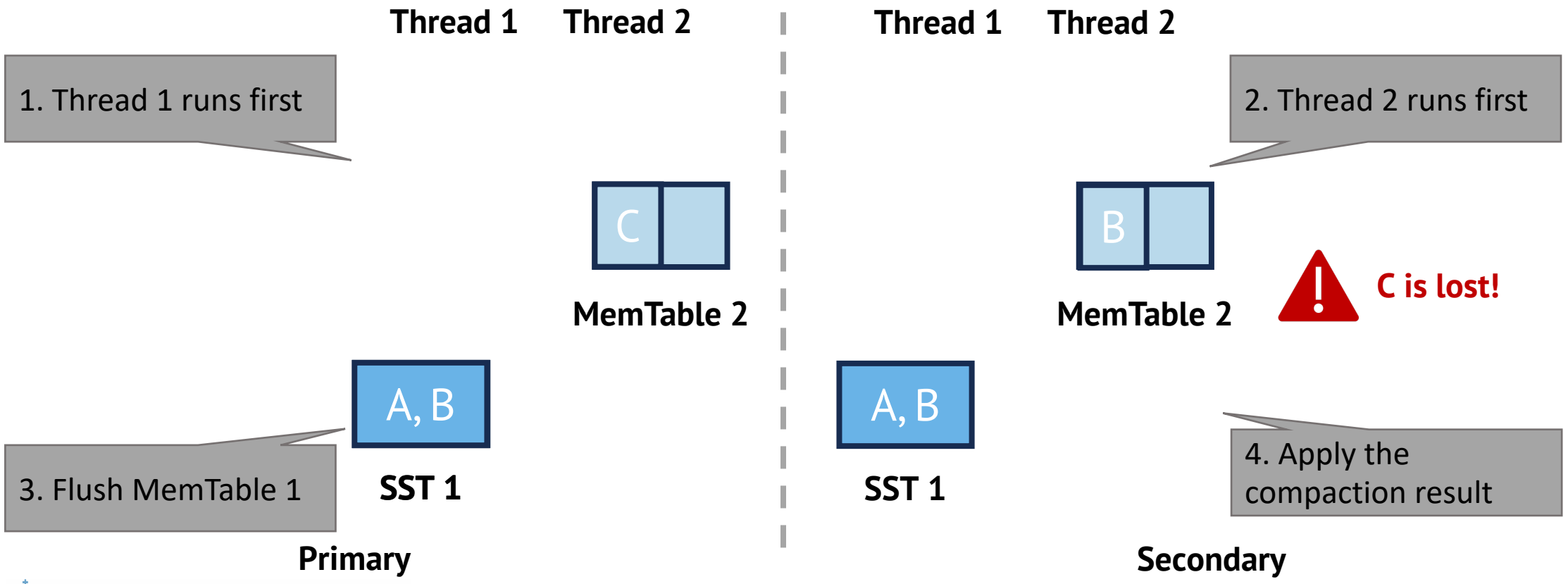
Shared SSTs causes replica inconsistency

Primary' and secondaries' internal states are actually different



Shared SSTs causes replica inconsistency

Primary' and secondaries' internal states are actually different

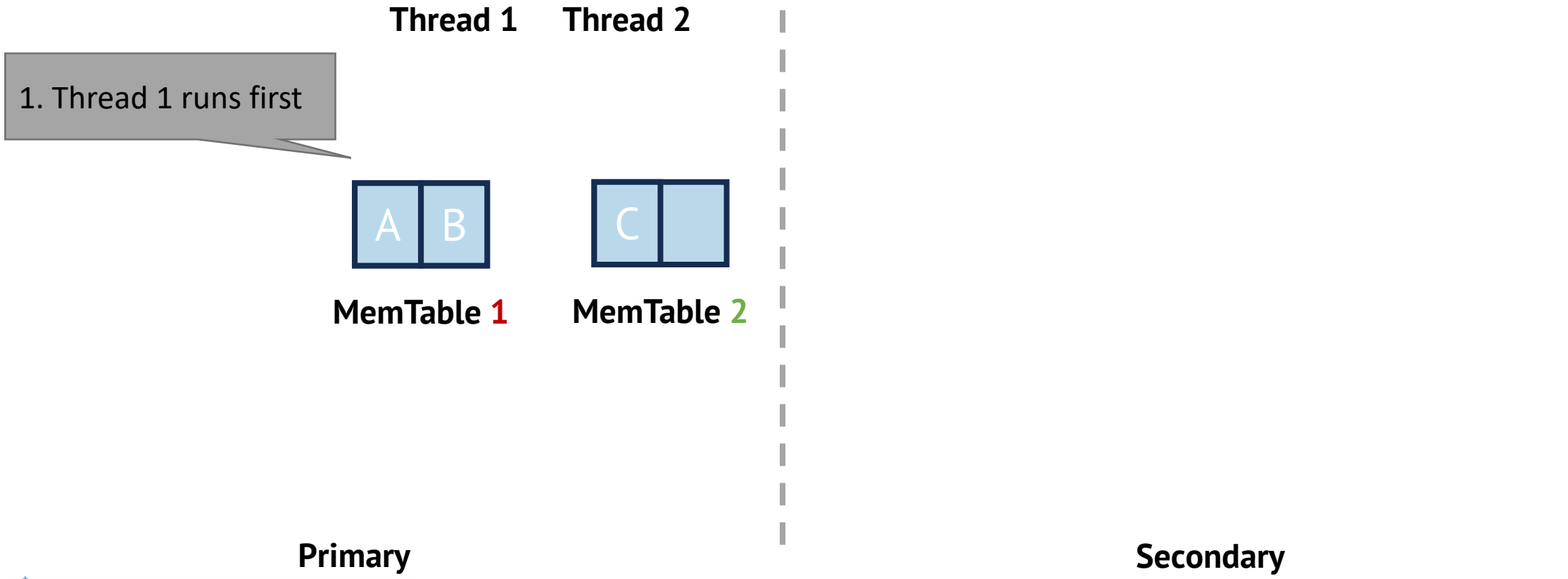


Partial order of write requests in RubbleDB

Goal: MemTables with a same ID should store the same data

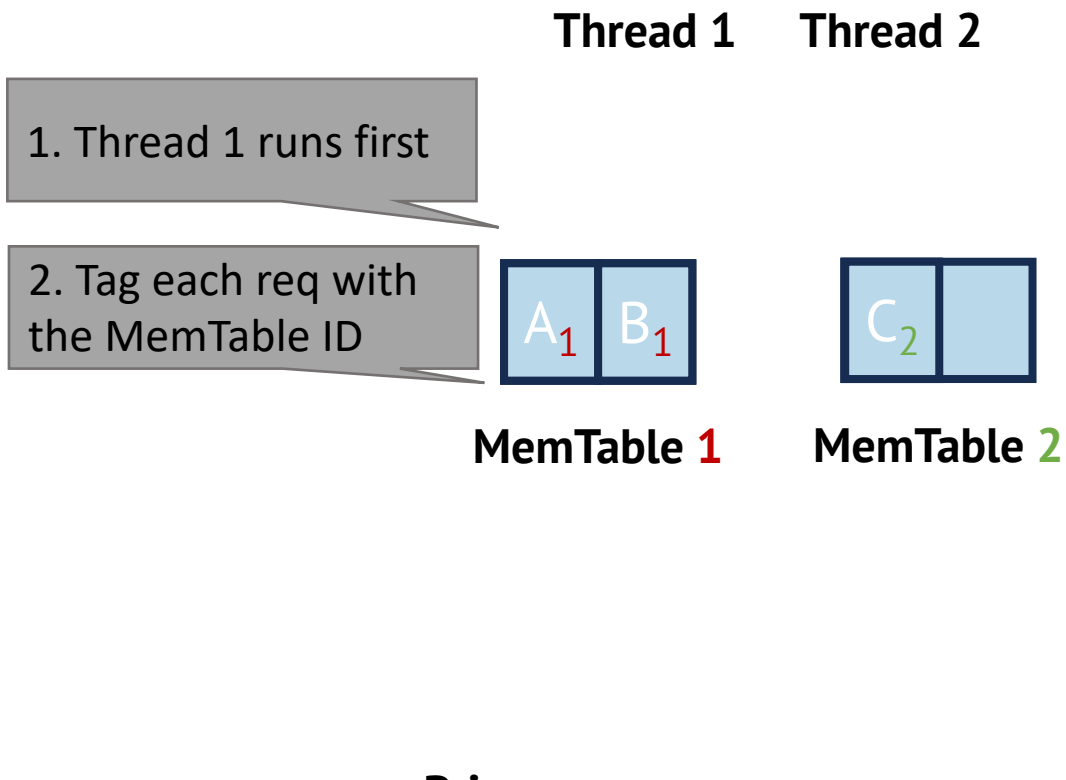
Partial order of write requests in RubbleDB

Goal: MemTables with a same ID should store the same data



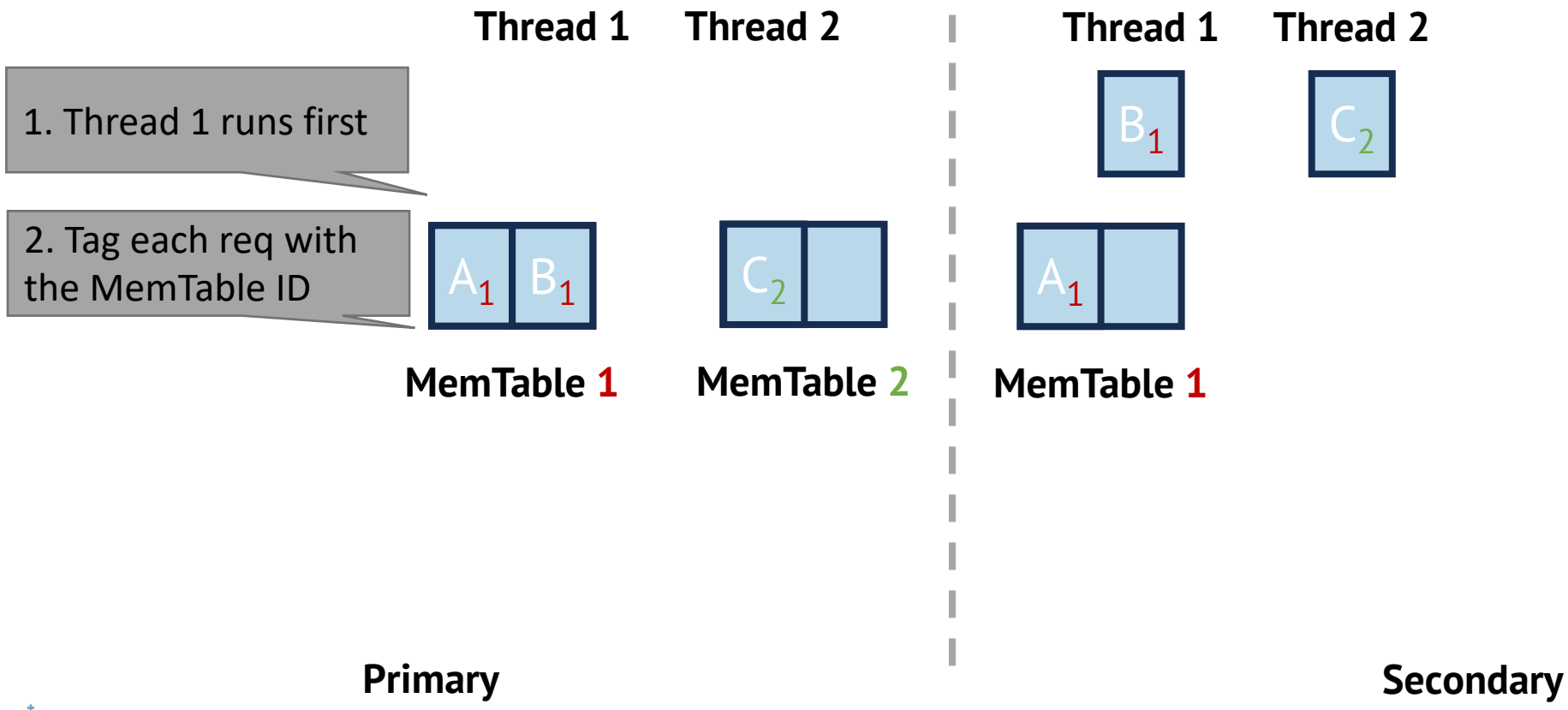
Partial order of write requests in RubbleDB

Goal: MemTables with a same ID should store the same data



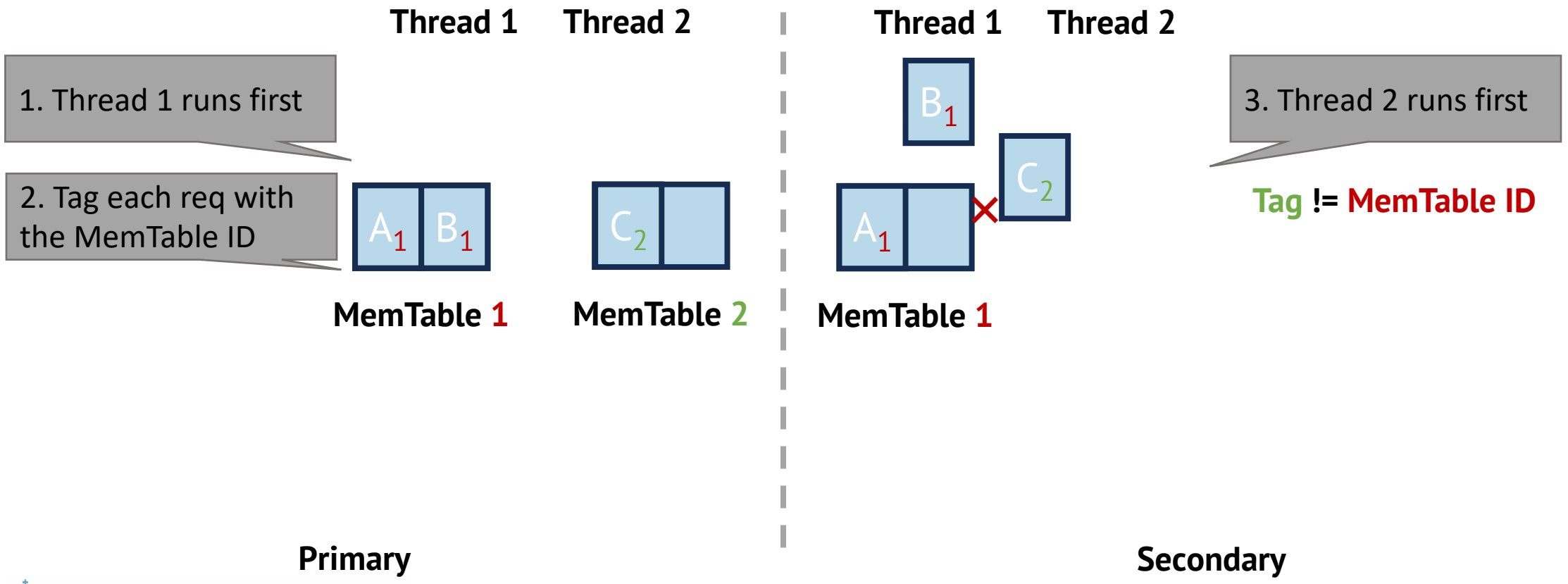
Partial order of write requests in RubbleDB

Goal: MemTables with a same ID should store the same data



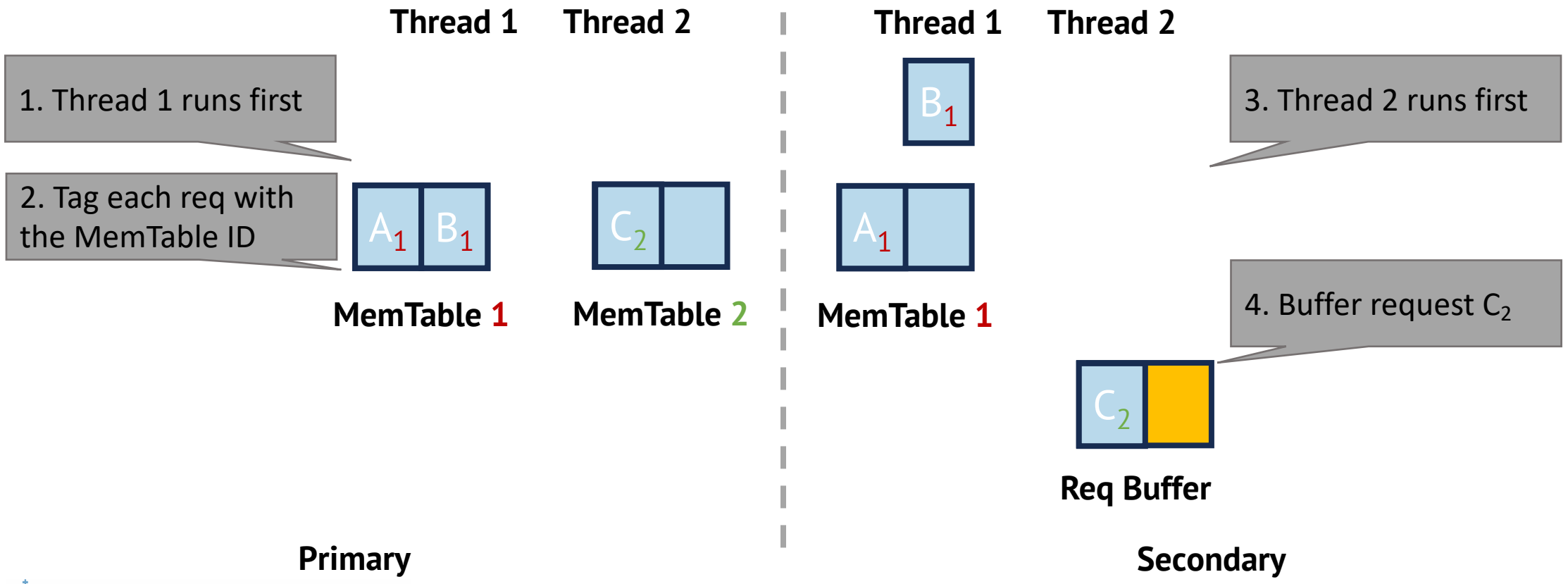
Partial order of write requests in RubbleDB

Goal: MemTables with a same ID should store the same data



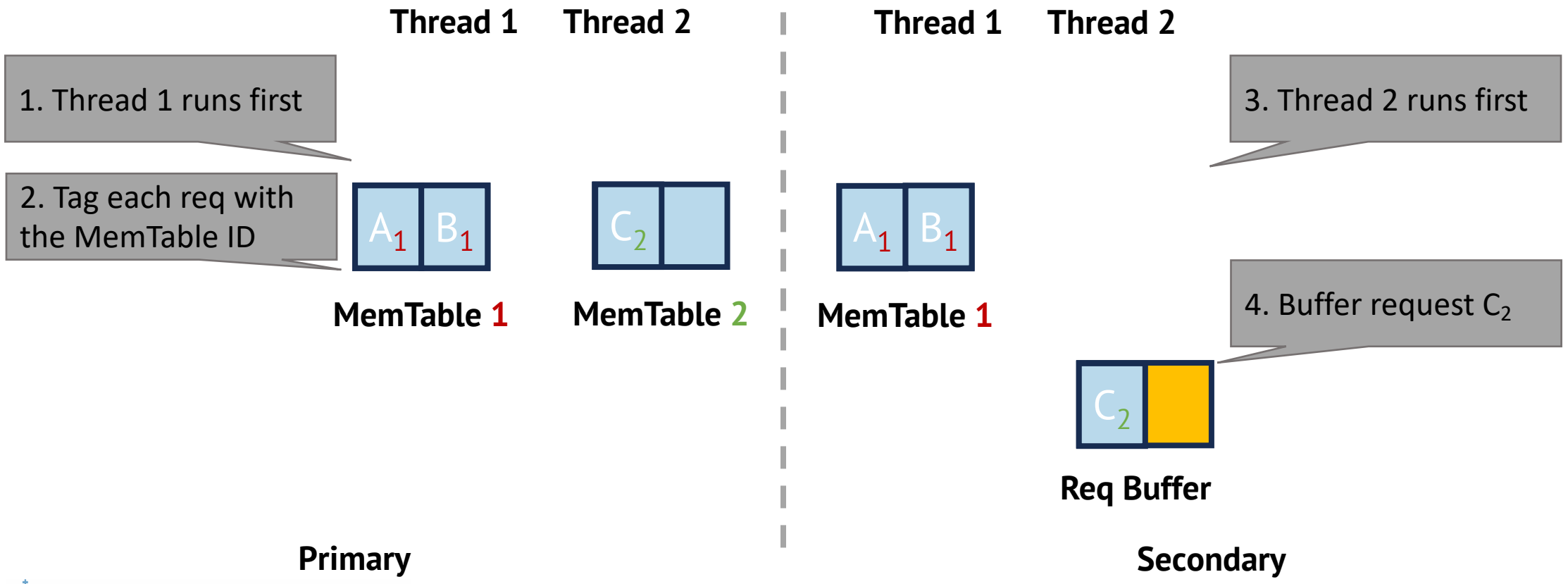
Partial order of write requests in RubbleDB

Goal: MemTables with a same ID should store the same data



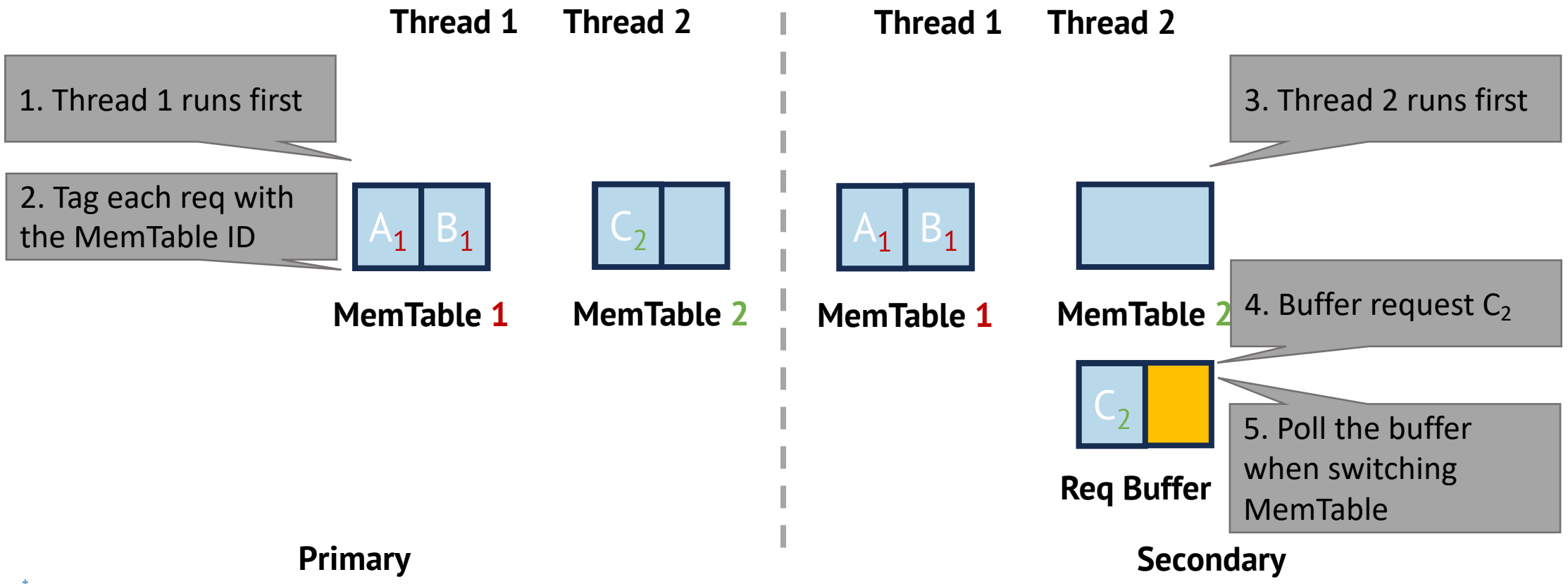
Partial order of write requests in RubbleDB

Goal: MemTables with a same ID should store the same data



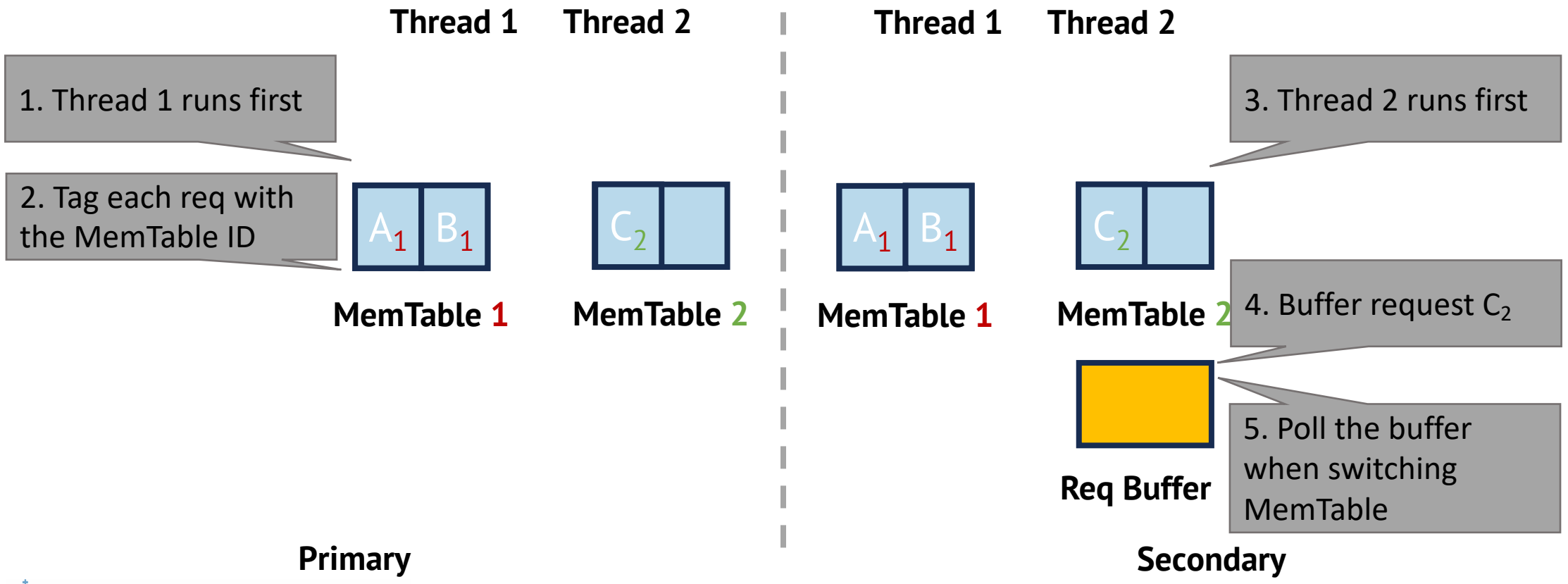
Partial order of write requests in RubbleDB

Goal: MemTables with a same ID should store the same data



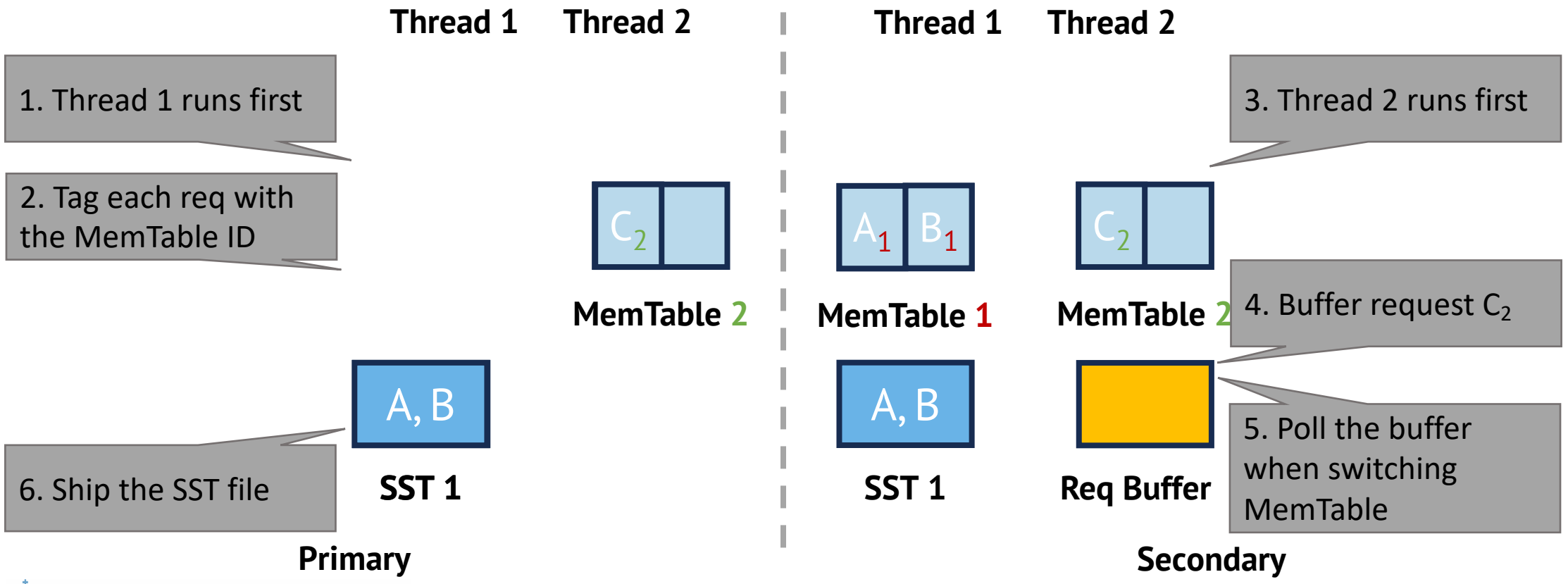
Partial order of write requests in RubbleDB

Goal: MemTables with a same ID should store the same data



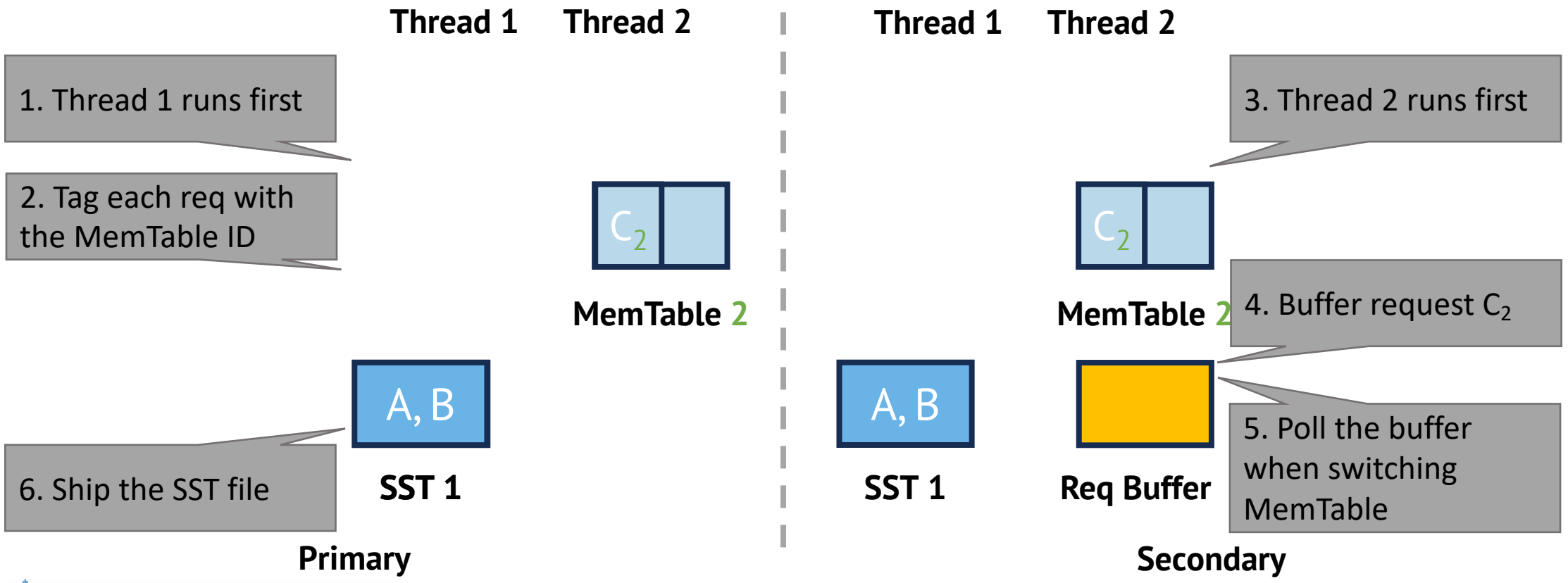
Partial order of write requests in RubbleDB

Goal: MemTables with a same ID should store the same data



Partial order of write requests in RubbleDB

Goal: MemTables with a same ID should store the same data



Evaluation

- How much can RubbleDB improve the end-to-end performance?
- What is the trade-off behind the improvement?
- How do different storage types affect RubbleDB?
- How fast can RubbleDB recover from failures?

} In the paper

Evaluation setup

Testbed: CloudLab r6525

- CPU: Two 32-core AMD 7543 at 2.8GHz
- Disk: One 1.6TB NVMe SSD
- NIC: Dual-port Mellanox ConnectX-6 100Gb

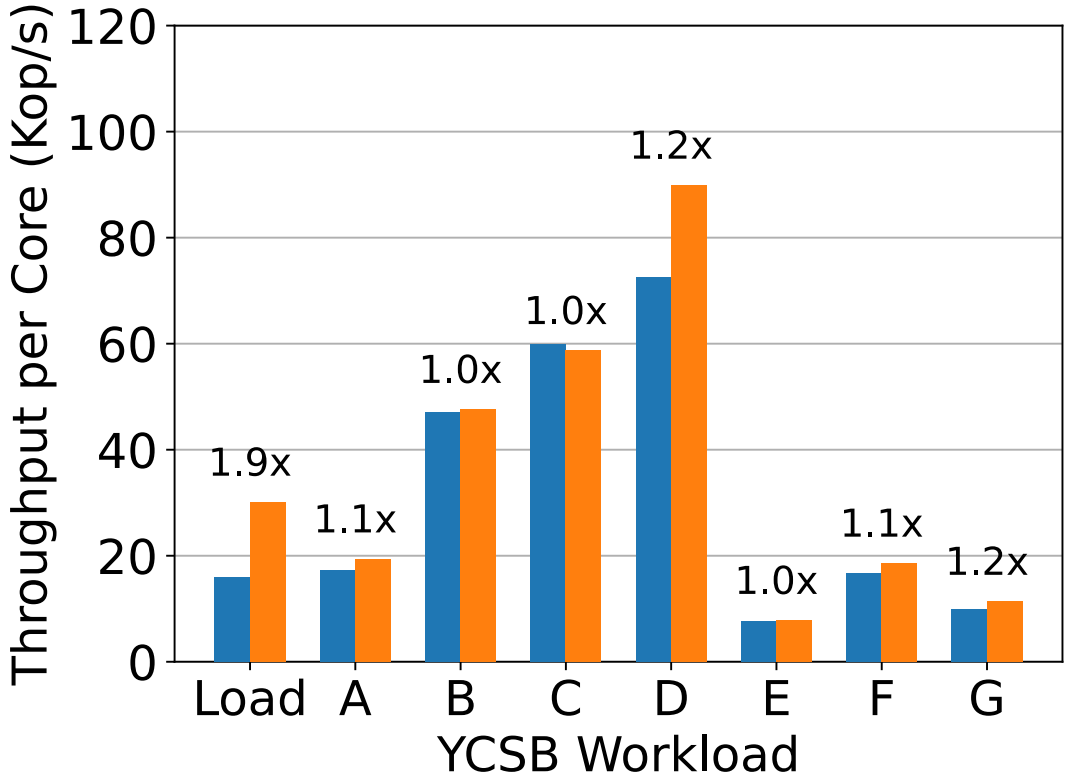
Benchmark:

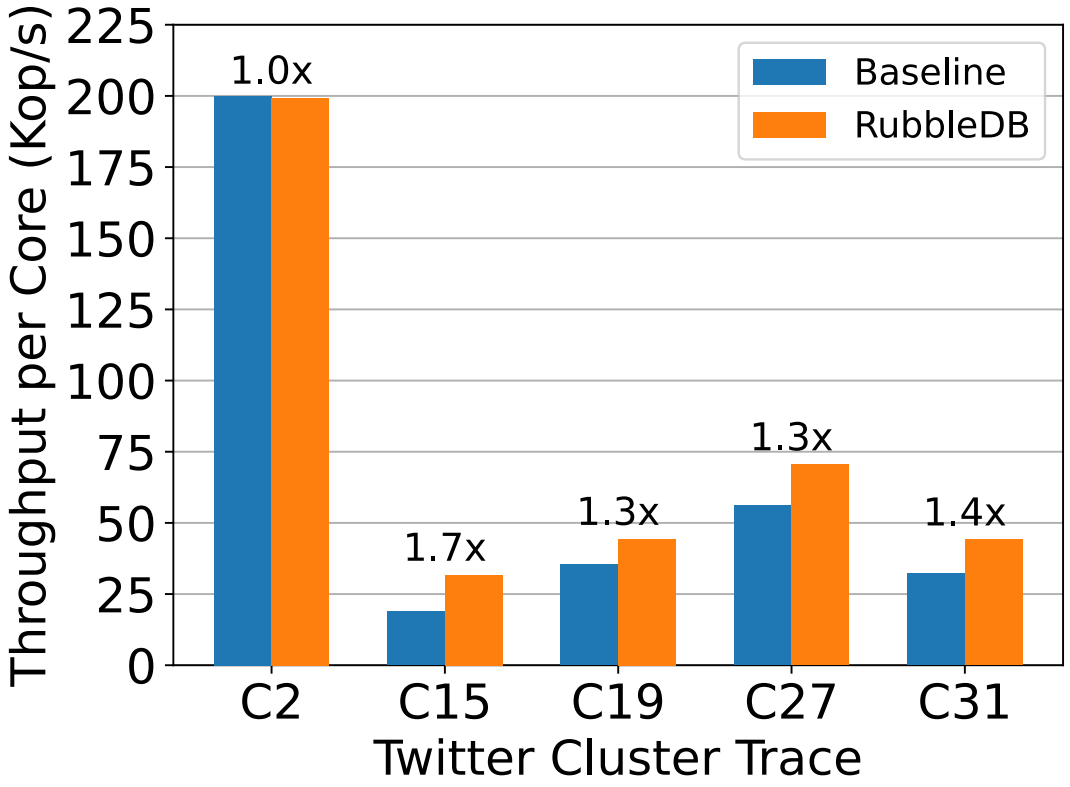
- YCSB load and A-G workloads
- Five Twitter cluster traces

Baseline:

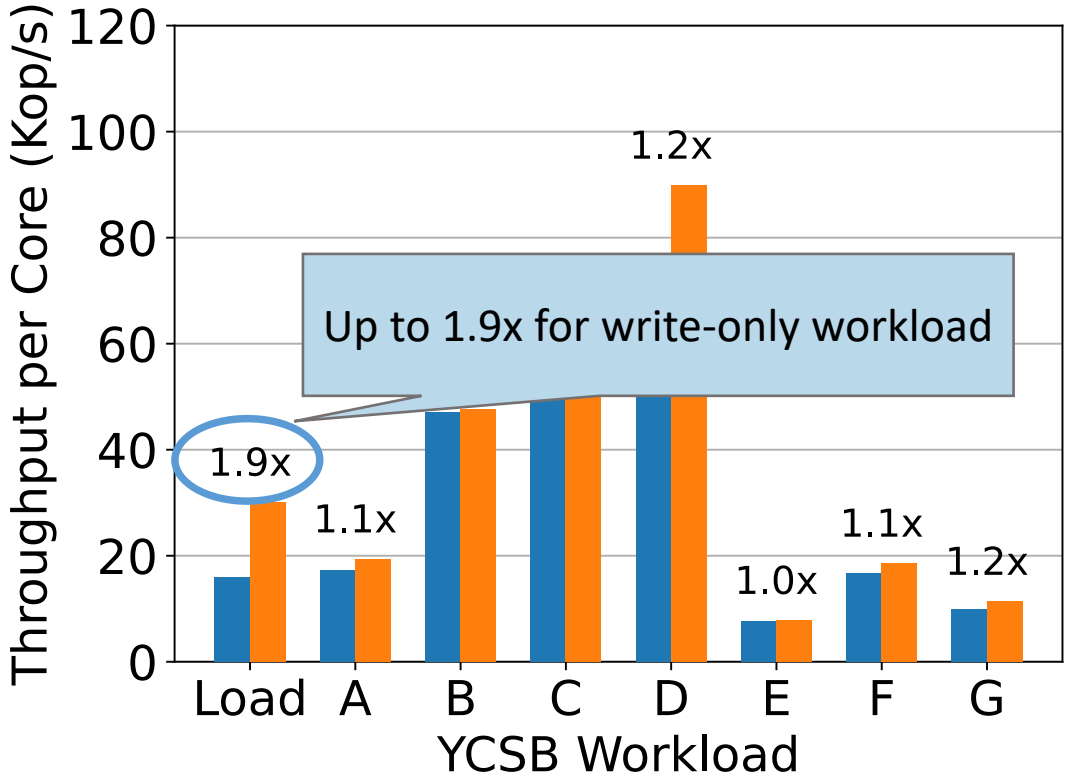
- Replicated RocksDB with compactions in secondaries

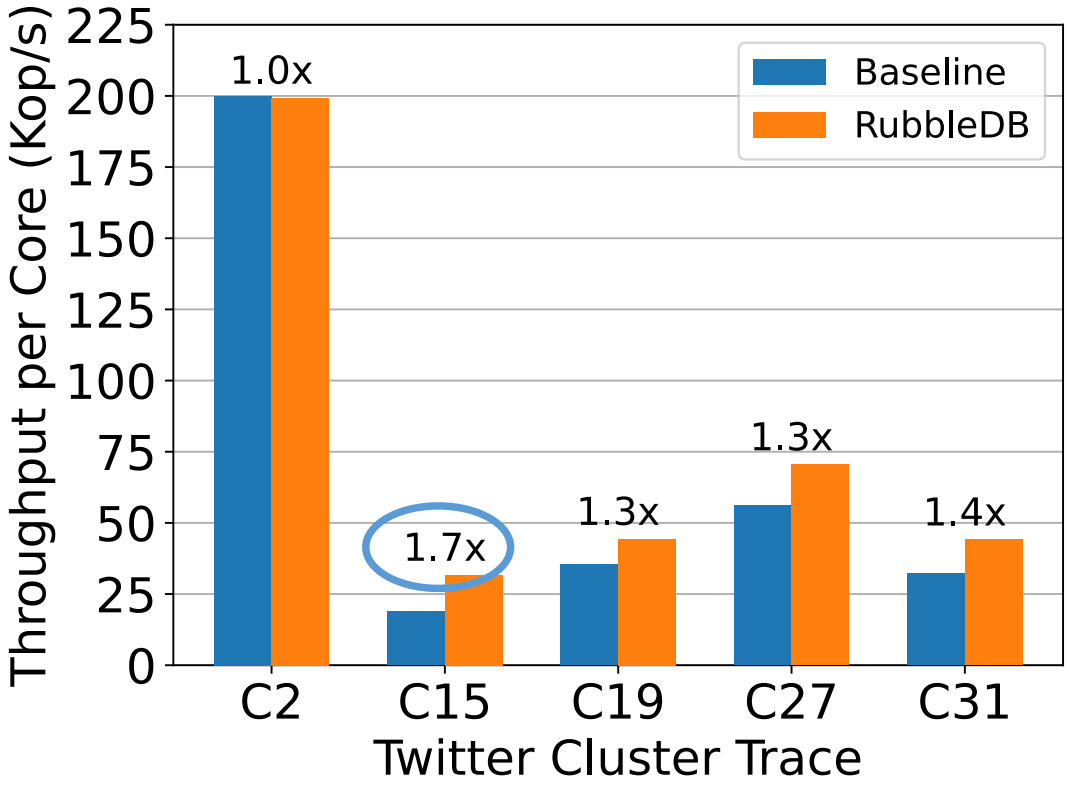
Throughput under different workloads



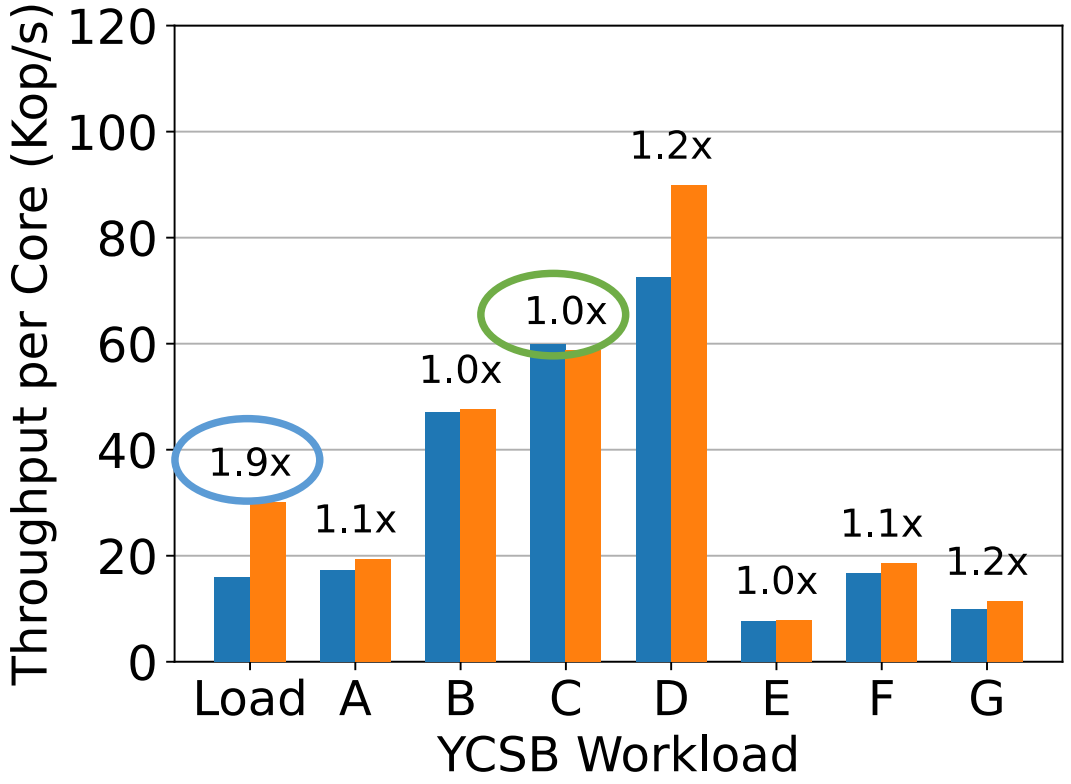


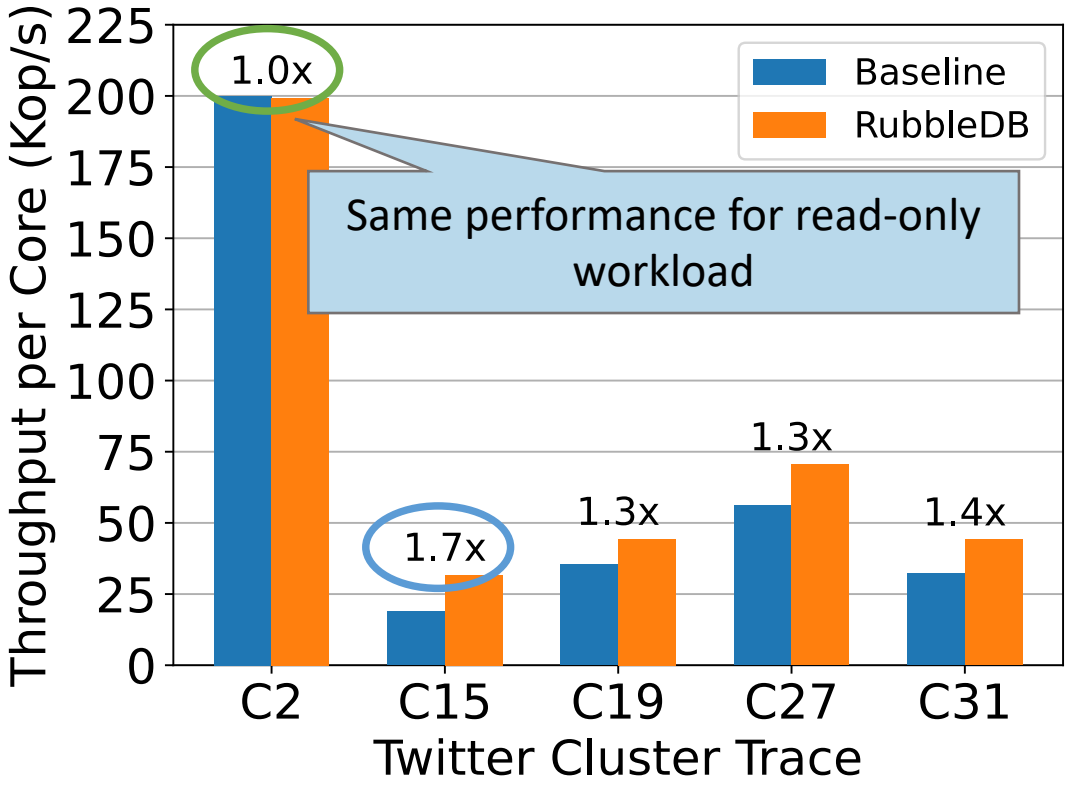
Throughput under different workloads



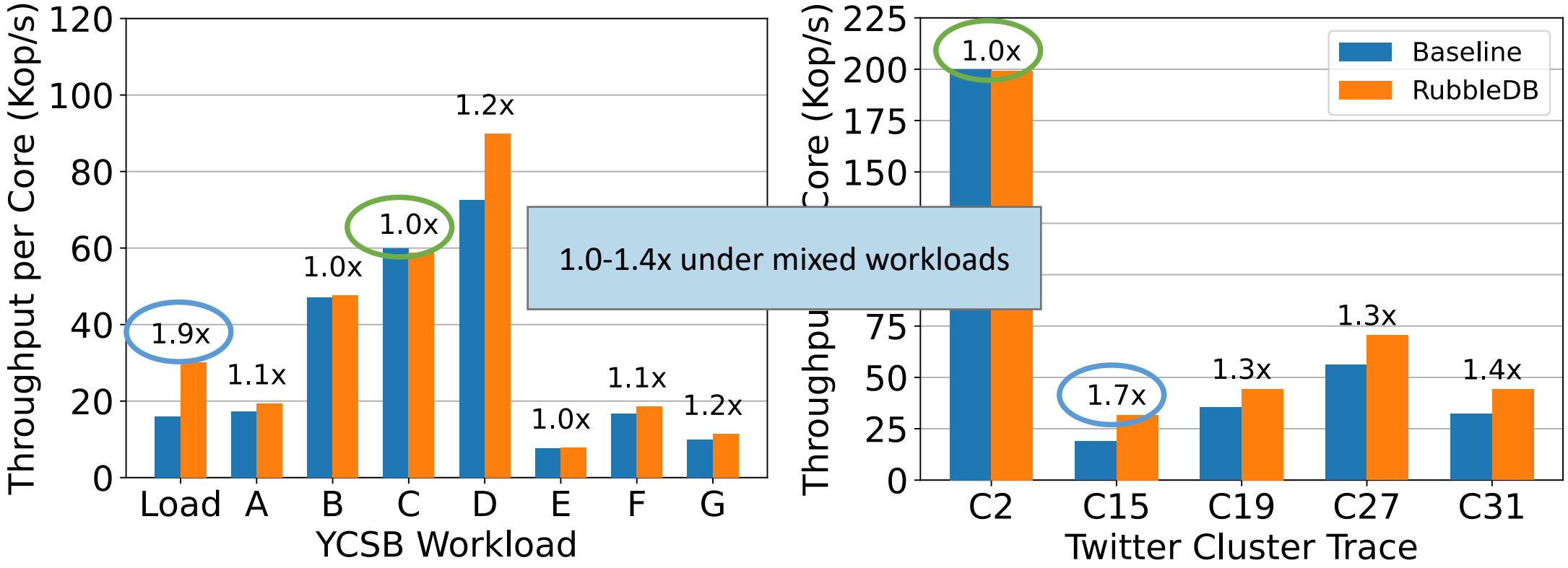


Throughput under different workloads

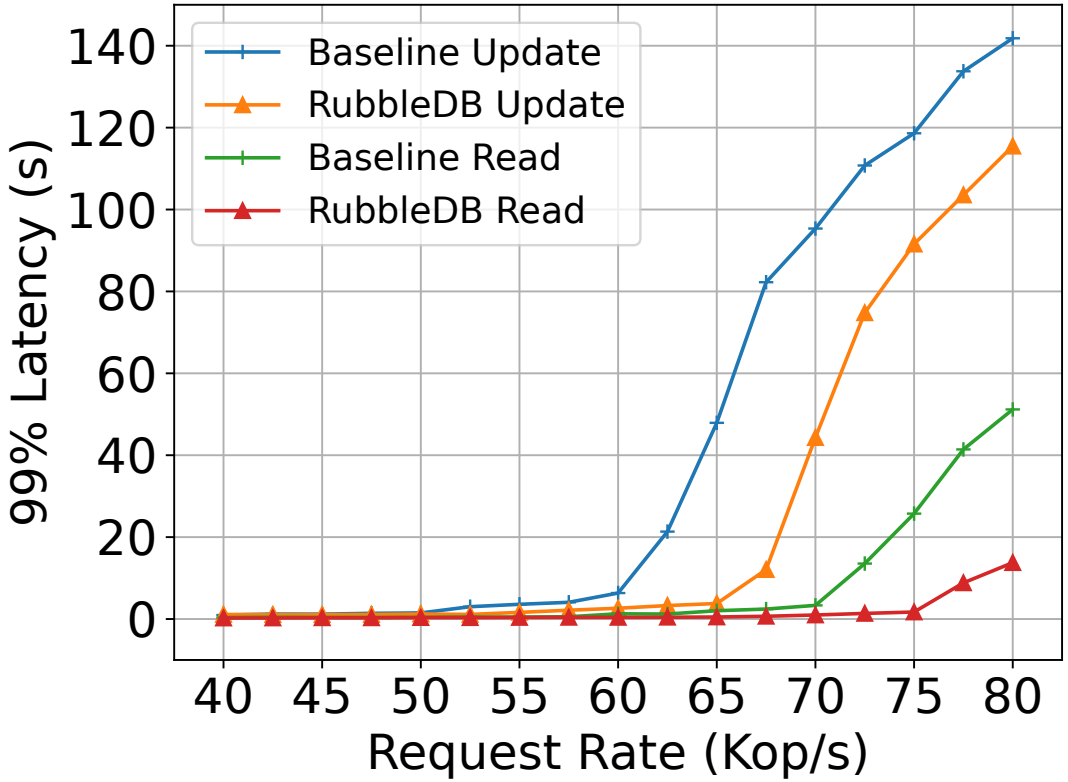




Throughput under different workloads



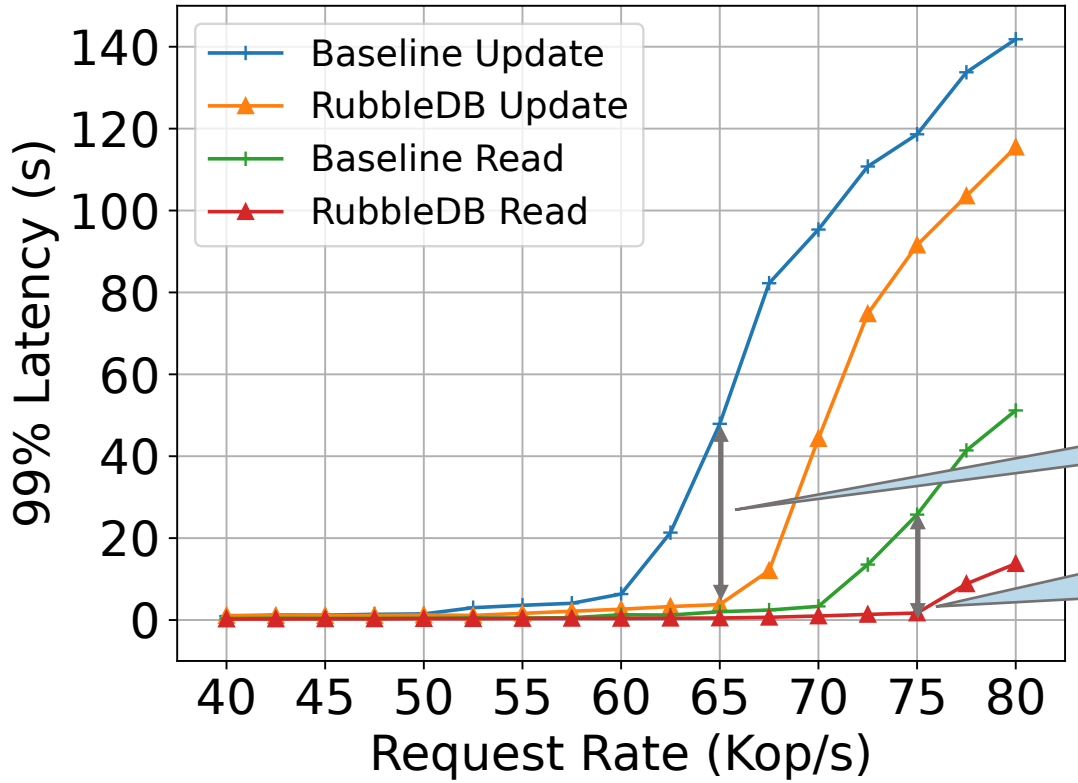
Tail latency comparison



Fewer compactions lead to fewer write stalls in RubbleDB

YCSB Workload A (50% read and 50% update), 30GB DB size, and three-way replication

Tail latency comparison



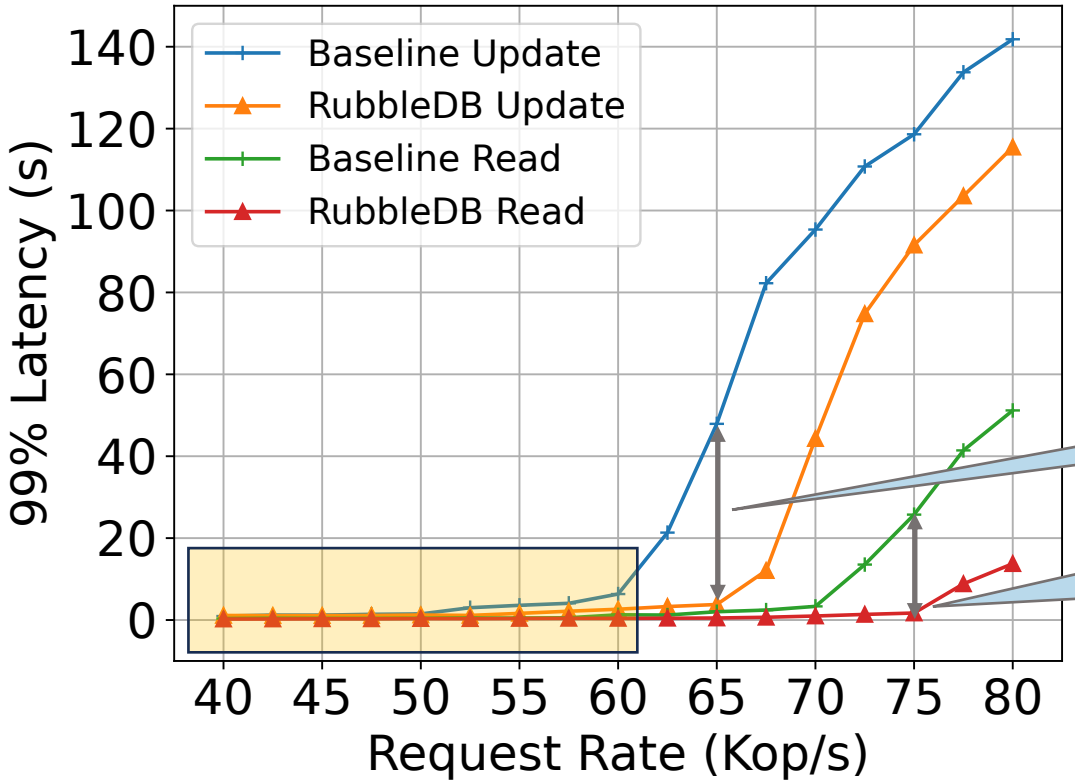
Fewer compactions lead to fewer write stalls in RubbleDB

Up to 92.1% lower update tail latency

Up to 93.4% lower read tail latency

YCSB Workload A (50% read and 50% update), 30GB DB size, and three-way replication

Tail latency comparison



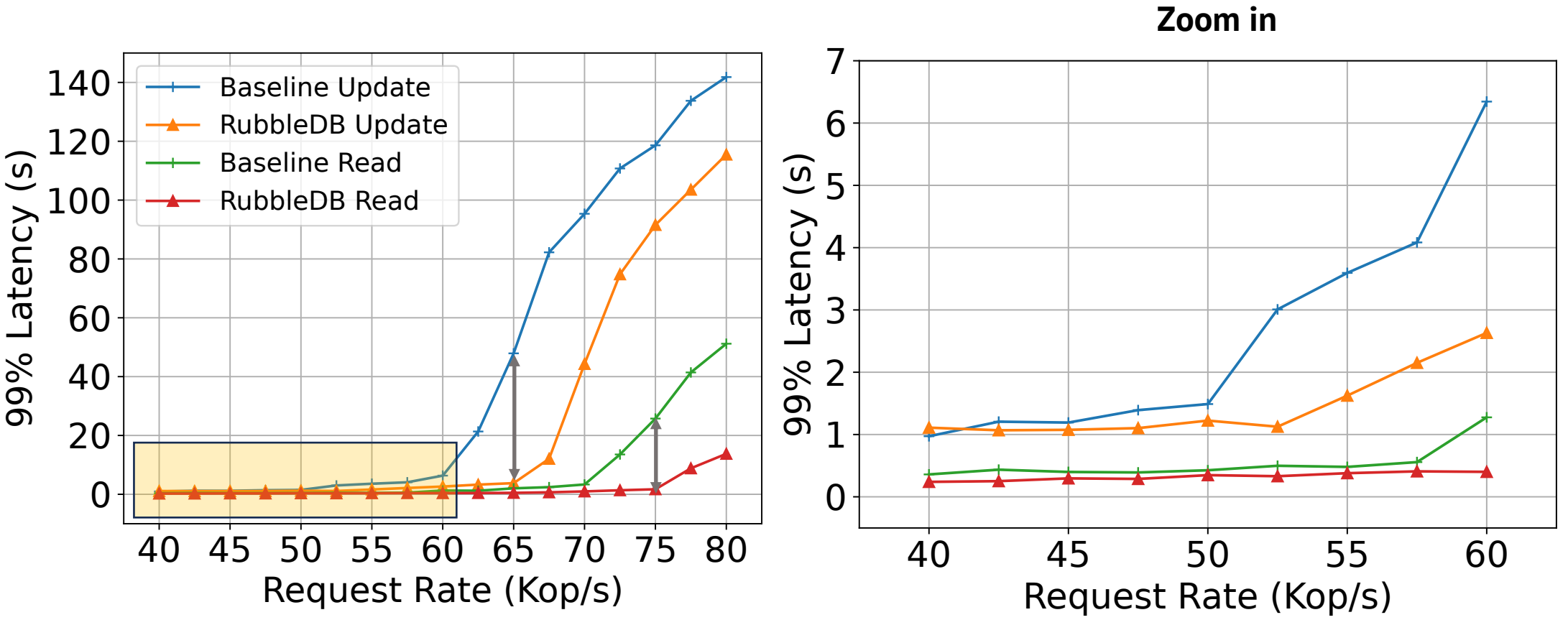
Fewer compactions lead to fewer write stalls in RubbleDB

Up to 92.1% lower update tail latency

Up to 93.4% lower read tail latency

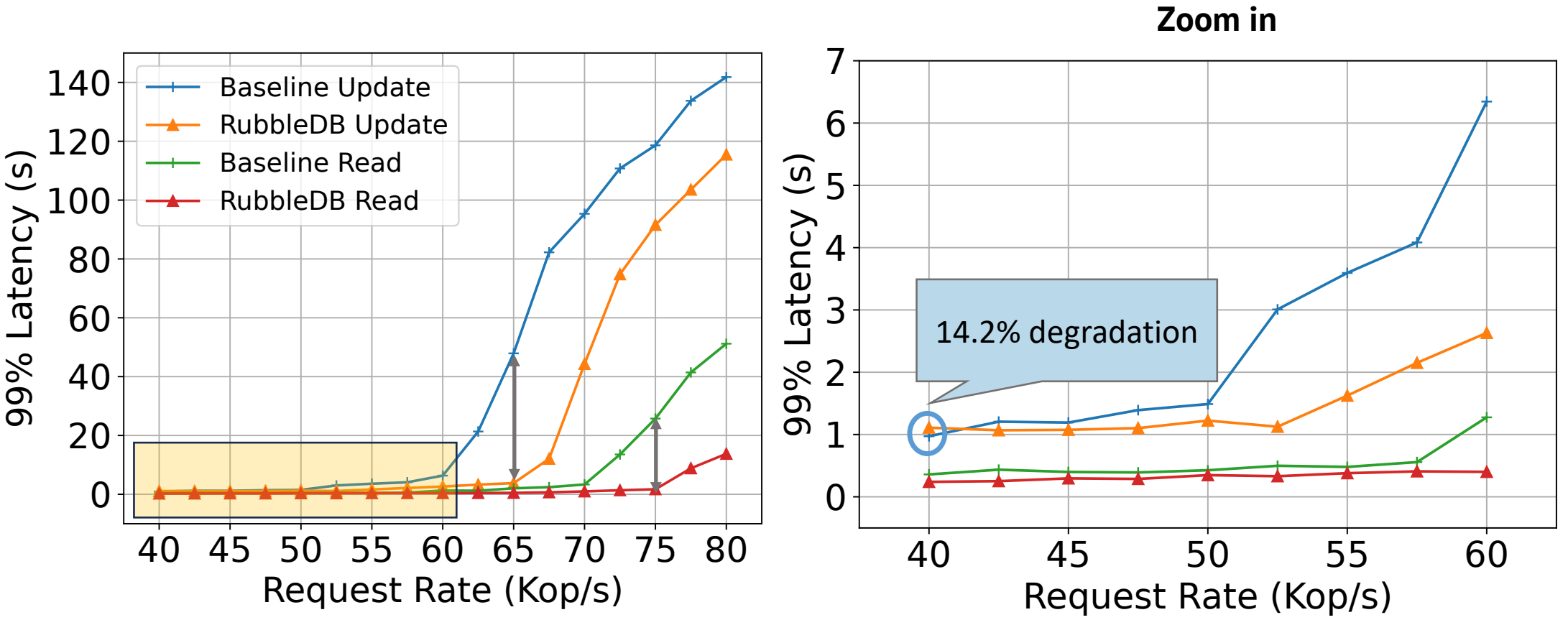
YCSB Workload A (50% read and 50% update), 30GB DB size, and three-way replication

Tail latency comparison



YCSB Workload A (50% read and 50% update), 30GB DB size, and three-way replication

Tail latency comparison



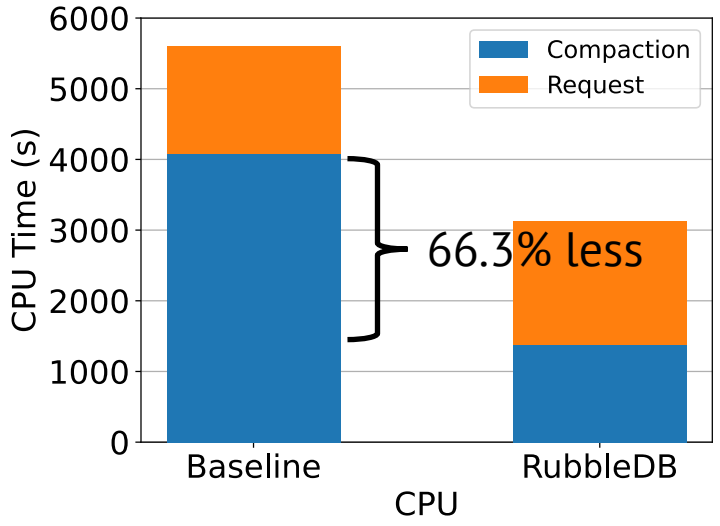
YCSB Workload A (50% read and 50% update), 30GB DB size, and three-way replication

Cluster-wide CPU, disk, and network stats

- RubbleDB trades network for CPU and disk
  - New network traffic for shipping SST files
  - No compaction CPU and read I/O on secondaries

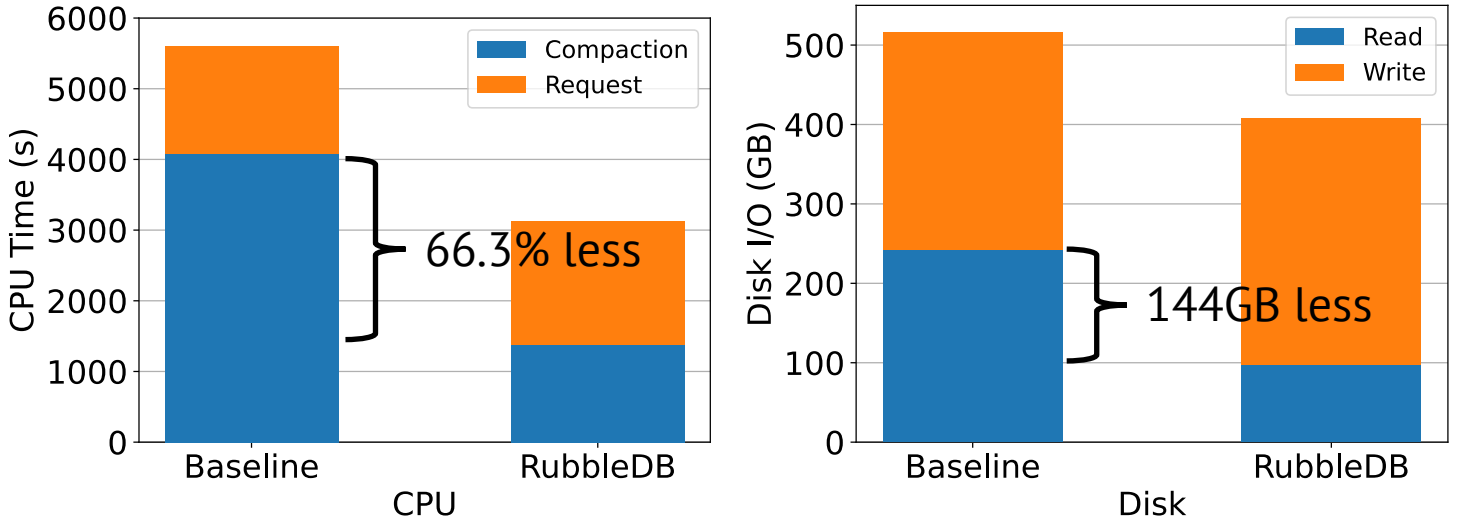
Cluster-wide CPU, disk, and network stats

- RubbleDB trades network for CPU and disk
  - New network traffic for shipping SST files
  - No compaction CPU and read I/O on secondaries



Cluster-wide CPU, disk, and network stats

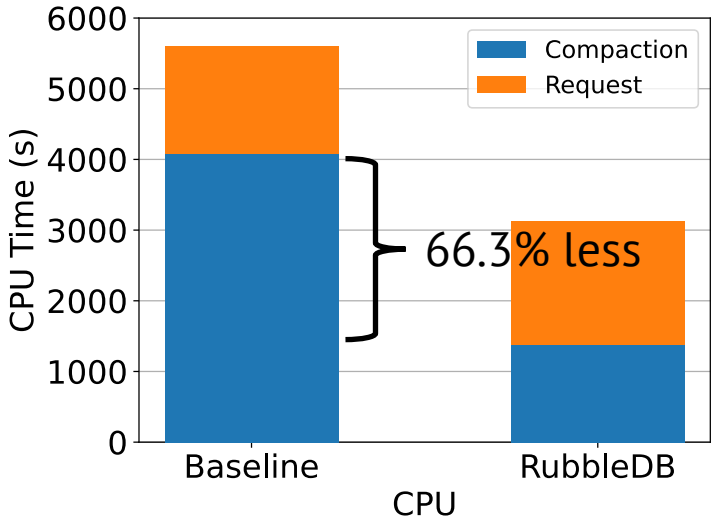
- RubbleDB trades network for CPU and disk
  - New network traffic for shipping SST files
  - No compaction CPU and read I/O on secondaries

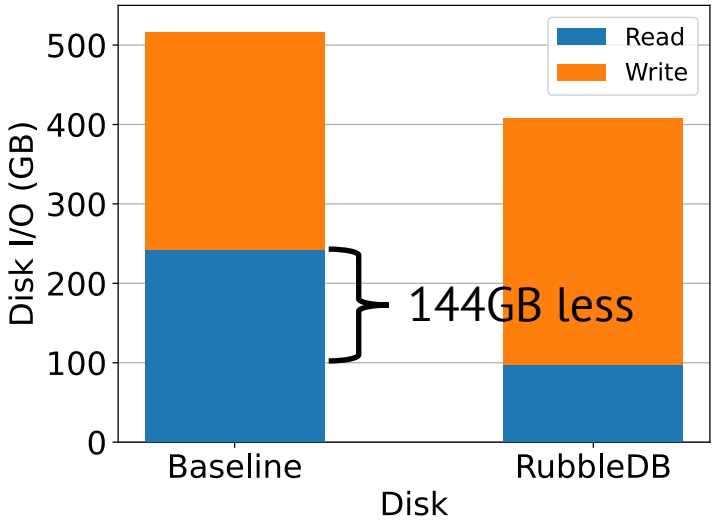


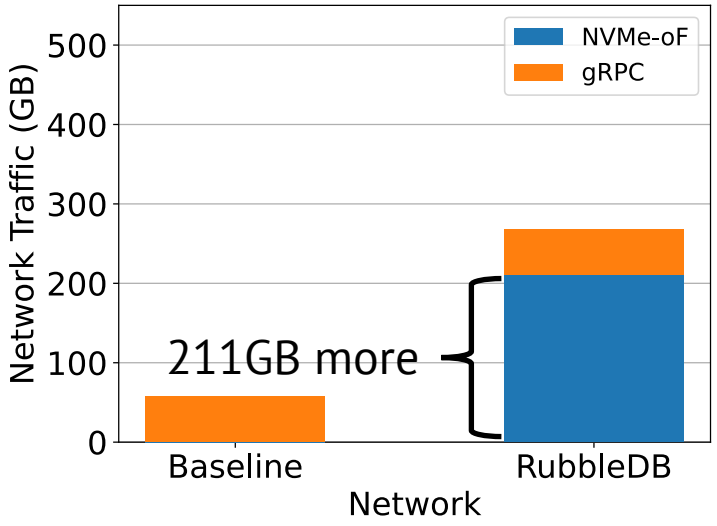
YCSB load, 30GB DB size, and three-way replication

Cluster-wide CPU, disk, and network stats

- RubbleDB trades network for CPU and disk
  - New network traffic for shipping SST files
  - No compaction CPU and read I/O on secondaries







YCSB load, 30GB DB size, and three-way replication

Conclusions

- NVMe-oF is an attractive opportunity for replicated storage systems
- RubbleDB trades network for CPU and disk read I/O by shipping compactions results to secondaries
- Try RubbleDB at https://github.com/lei-houjyu/RubbleDB

Thank you!

haoyu.li@columbia.edu

Backup Slides

Cluster topology

- K replication groups spread on R servers
- Saving compactions in secondaries gives the primary more CPU