LoLKV: The Logless, Linearizable, RDMA-based Key-Value Storage System

Ahmed Alquraan, Sreeharsha Udayashankar, Virendra Marathe,
Bernard Wong, Samer Al-Kiswany

University of Waterloo

Oracle Labs

Production-quality Key-Value Stores

- Leader-based consensus protocols → Strong consistency

Log Cabin

CockroachDB

TiKV

ArangoDB

AEROSPIKE

FoundationDB

etcd

Raft

Paxos

ZAB

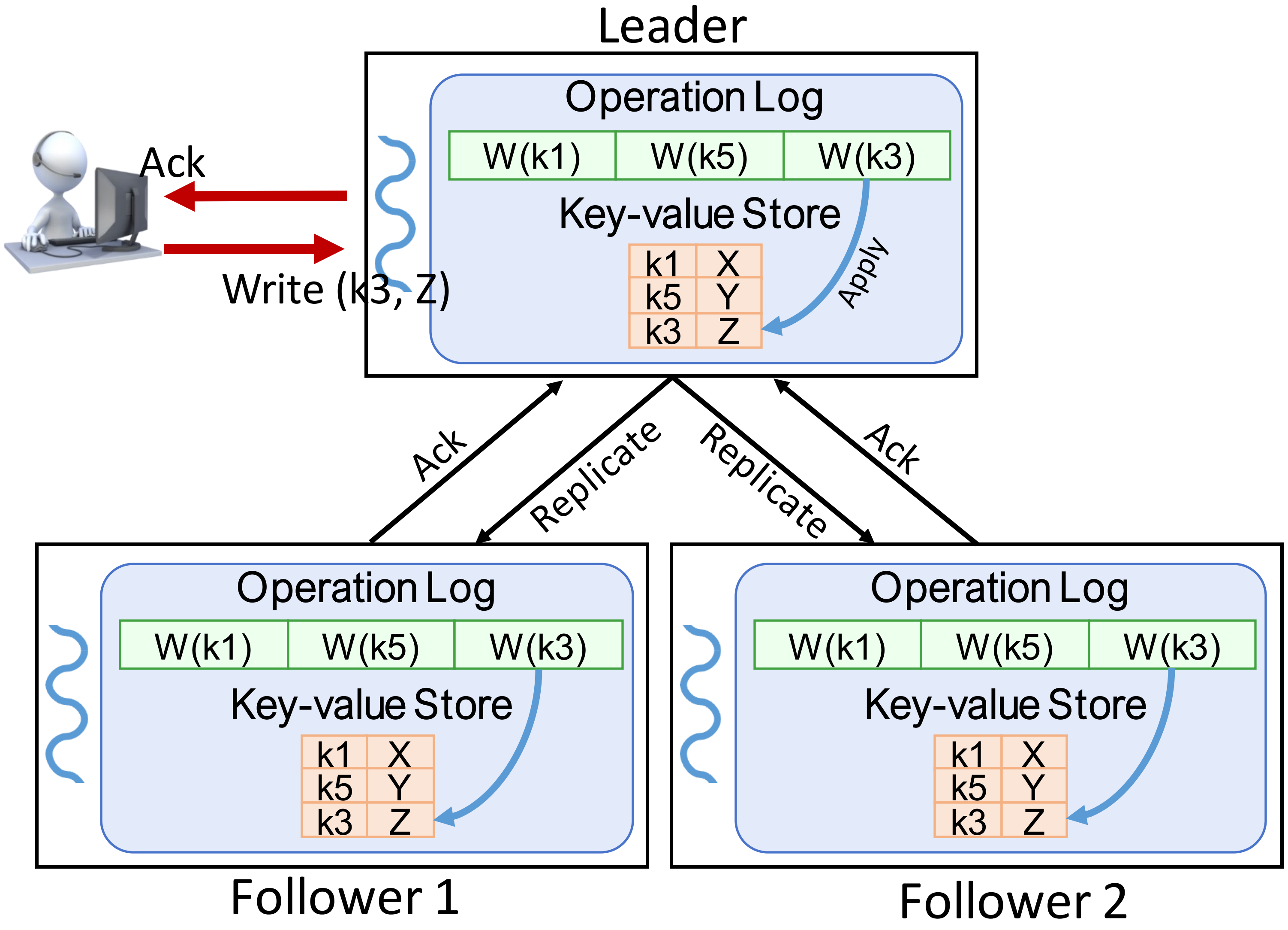
Mu

APUS

DARE

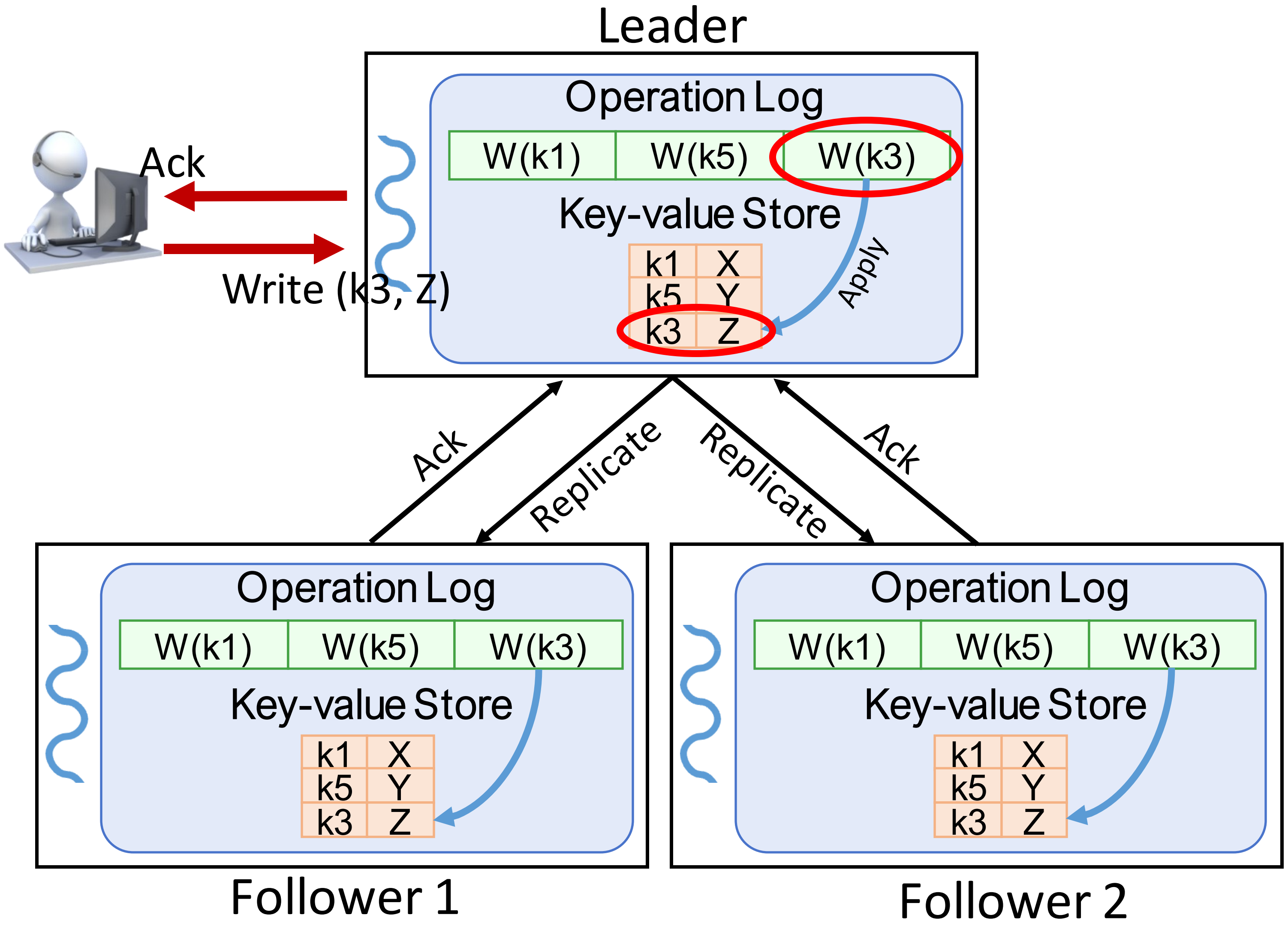
Viewstamped Replication

Shortcomings of Current Systems



Shortcomings of Current Systems

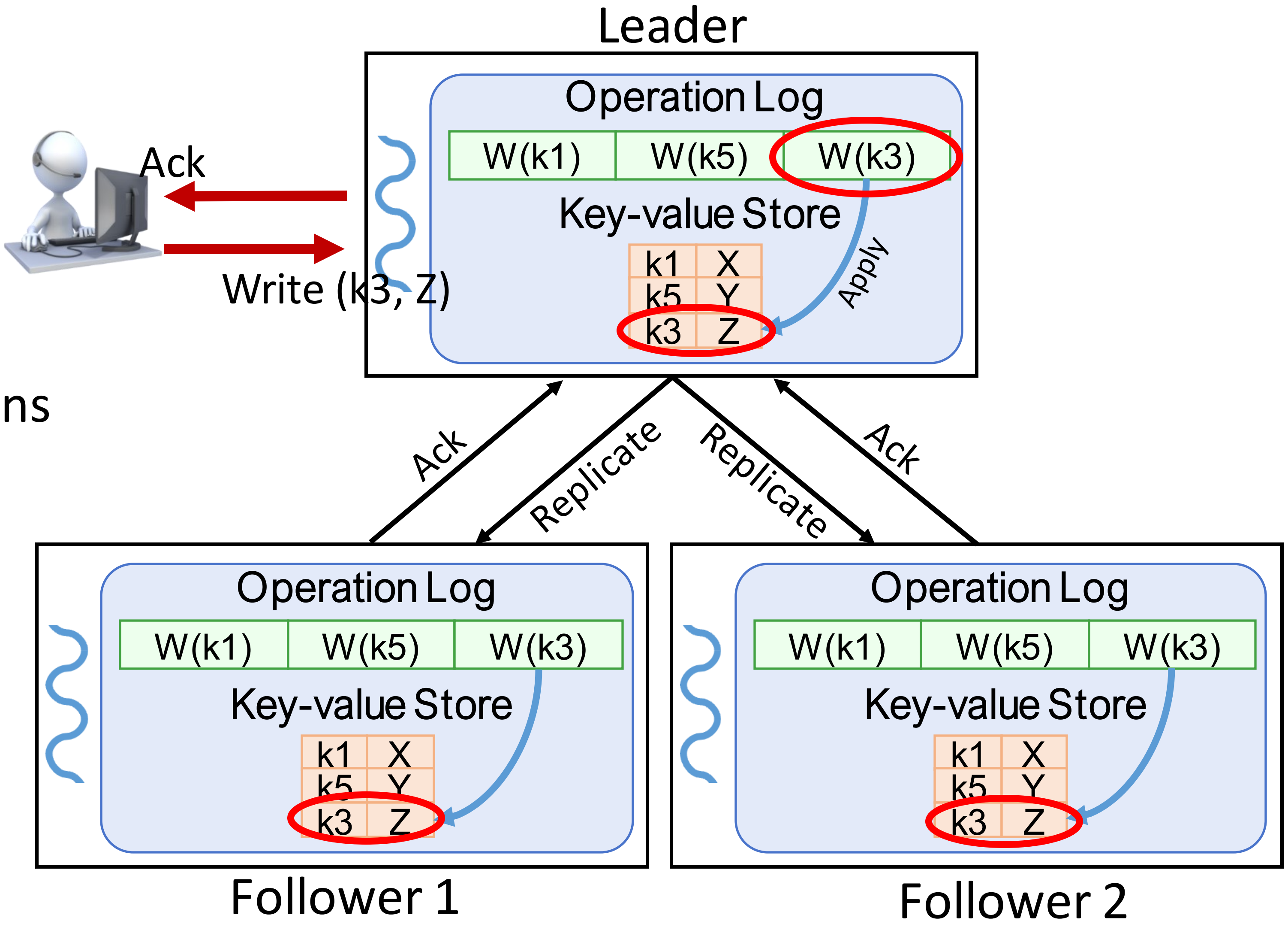
- Log-based replication
  - Log is a serialization point
  - Unnecessary data copying



Shortcomings of Current Systems

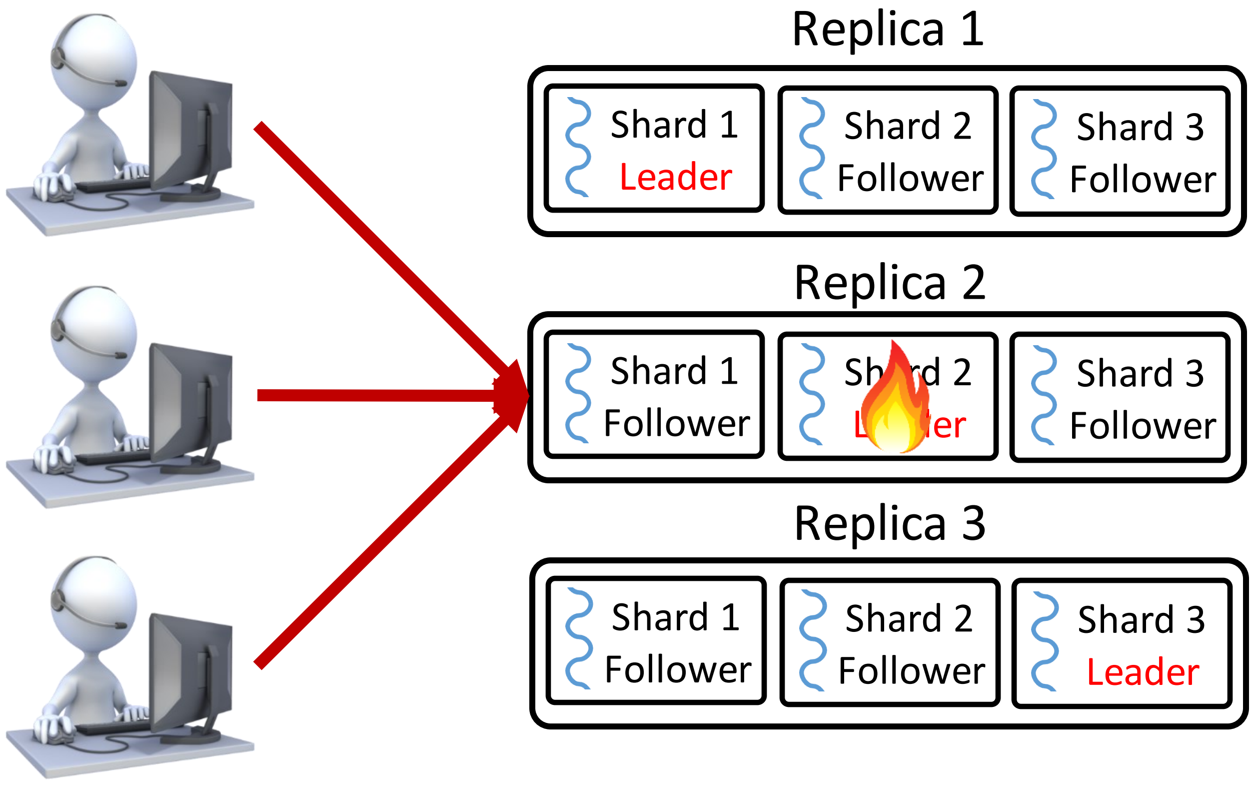
- Log-based replication
  - Log is a serialization point
  - Unnecessary data copying

- All replicas apply committed operations
  - Work repetition on all replicas



Shortcomings of Current Systems

- Log-based replication
  - Log is a serialization point
  - Unnecessary data copying
- All replicas apply committed operations
  - Work repetition on all replicas
- Multiple single-threaded shards
  - Inefficient for skewed workloads
  - Resource fragmentation: separate memory regions per shard



Shortcomings of Current Systems

- Log-based replication
  - ~~Log is a serialization point~~
  - ~~Unnecessary data copying~~
- All replicas apply committed operations
  - ~~Work repetition on all replicas~~
- Multiple single-threaded shards
  - ~~Inefficient for skewed workloads~~
  - ~~Resource fragmentation: separate memory regions per shard~~

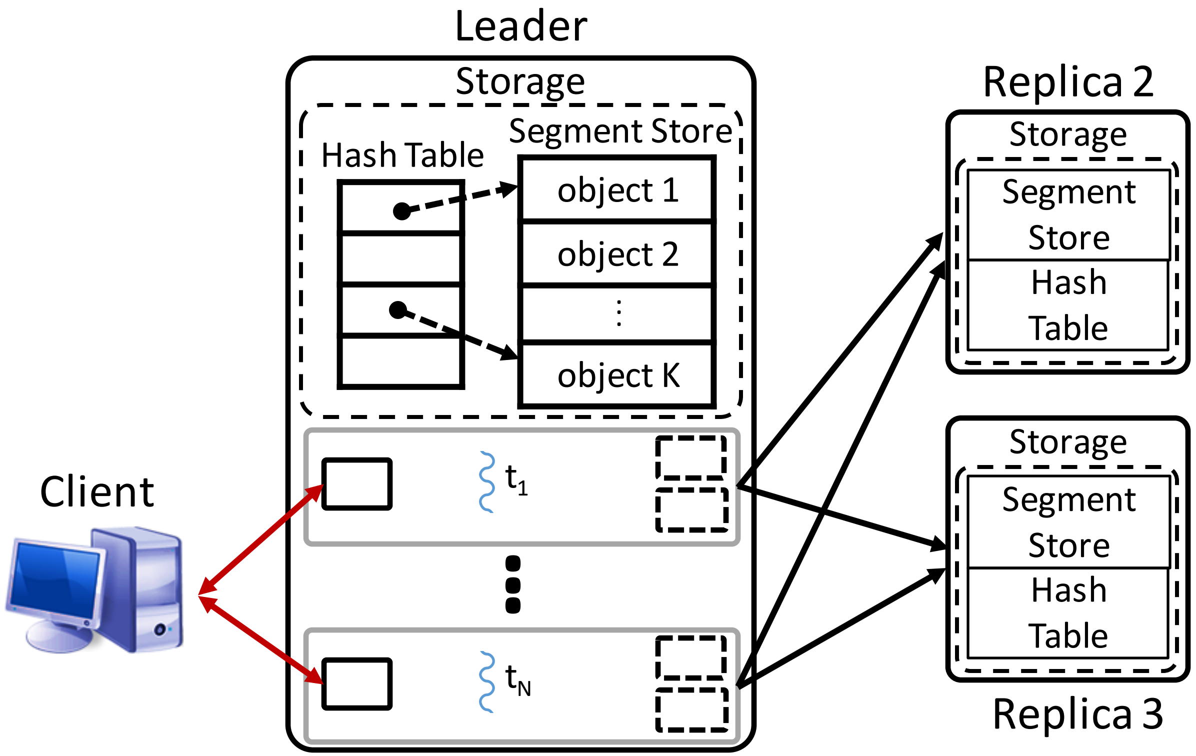
LoLKV

A new linearizable RDMA-based KV Store

- A novel logless replication
- Combines replication and apply
- Passive followers
- A novel multi-threaded shard design

Design

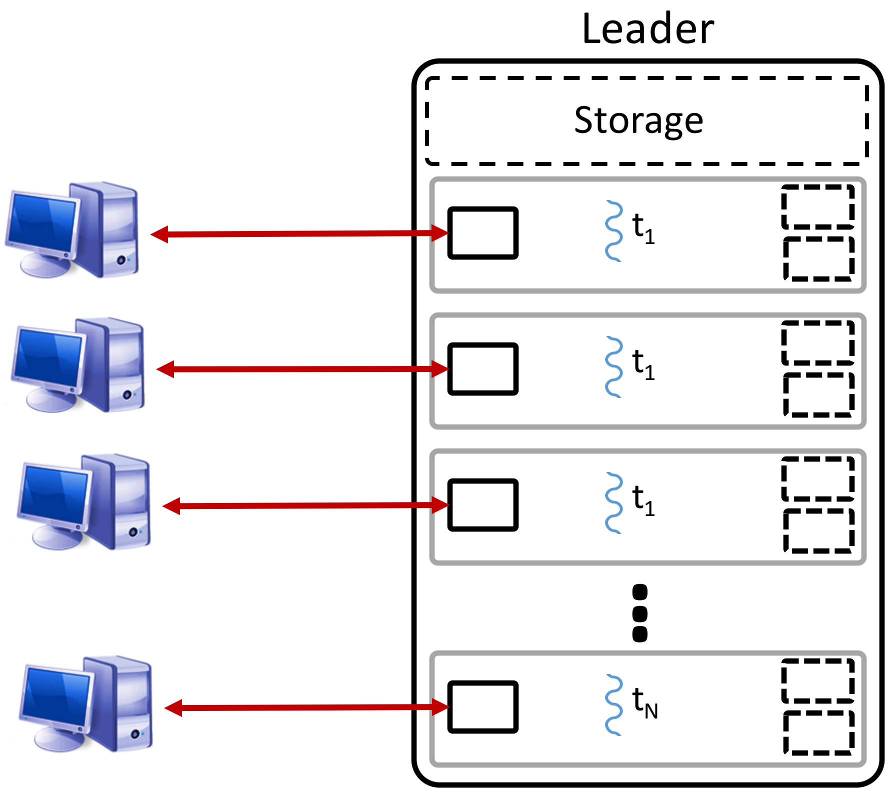
- Leader-based system
- Two main components
  - Storage
  - Worker threads
- RDMA-based system
  - UD for communication with clients
  - RC for communication between replicas



Worker Threads

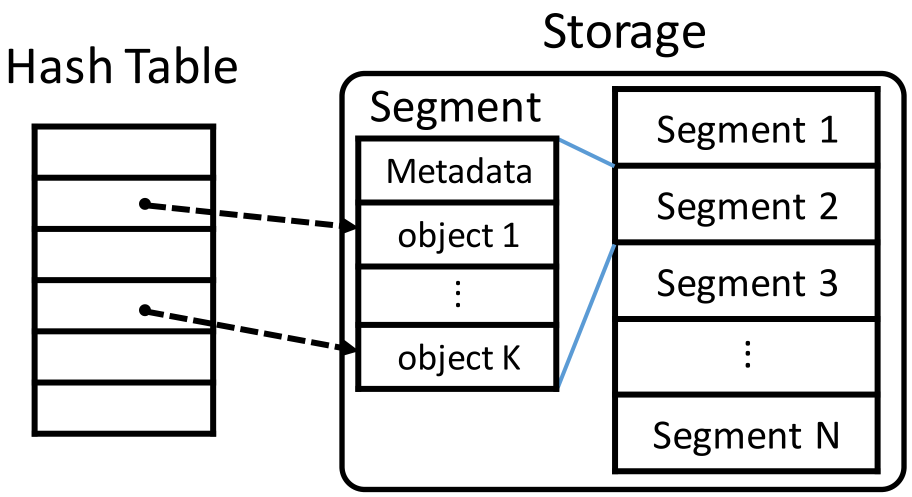
- Design Goals
  - Highly-concurrent design
  - Avoid sharding the key space among threads

- Employs multiple worker threads
- Each thread has its own RDMA resources
- Each thread serves requests for any key
  - Run the consensus protocol
  - Update the storage



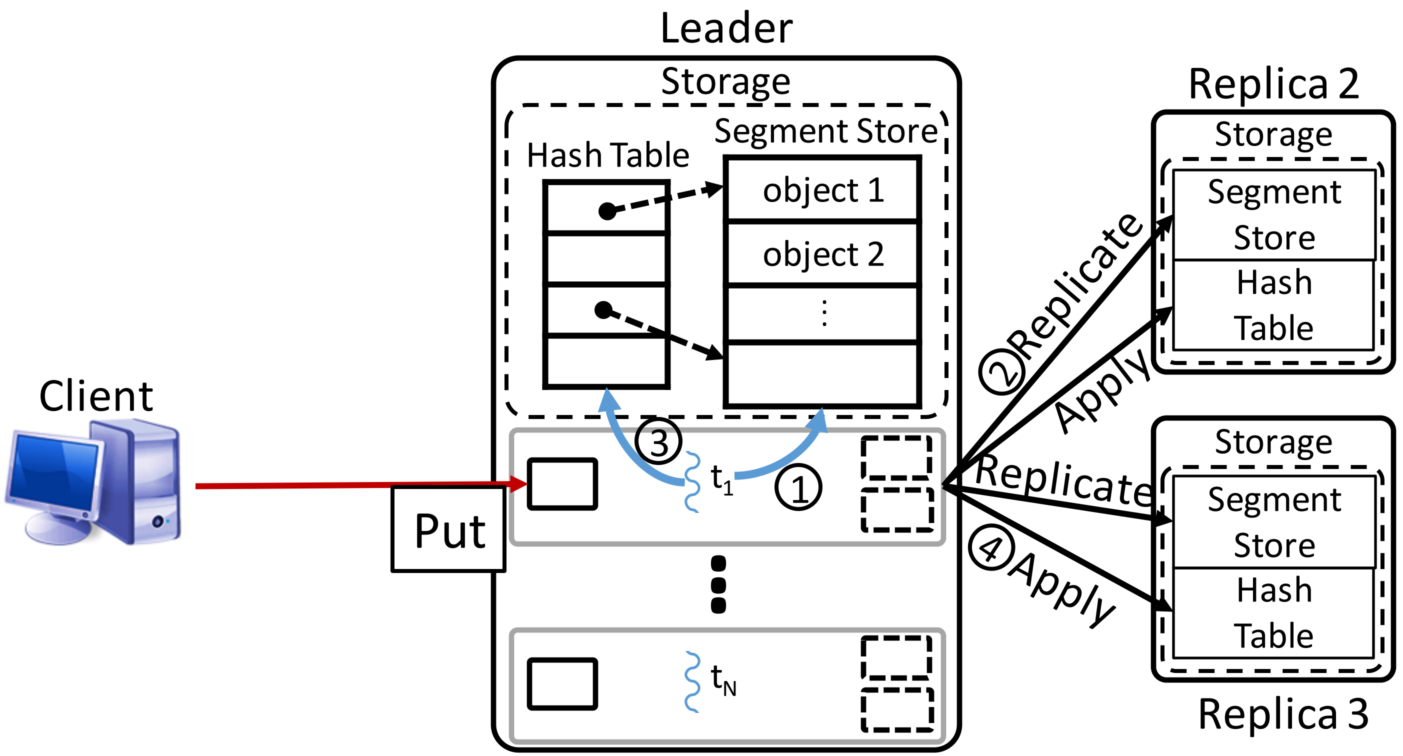
Storage Design

- Design Goals
  - Minimize RDMA communication
  - Minimize contention between threads
- Storage
  - Memory divided into segments
  - Each segment stores a set of objects
  - A segment is owned by one thread at a time
  - One RDMA Write to commit an operation
- Hash Table
  - Stores pointers to objects in the storage
  - A lock-free linear probing hash table
  - Shared between all threads
  - One RDMA Write to apply an operation



LoLKV Write Request Path

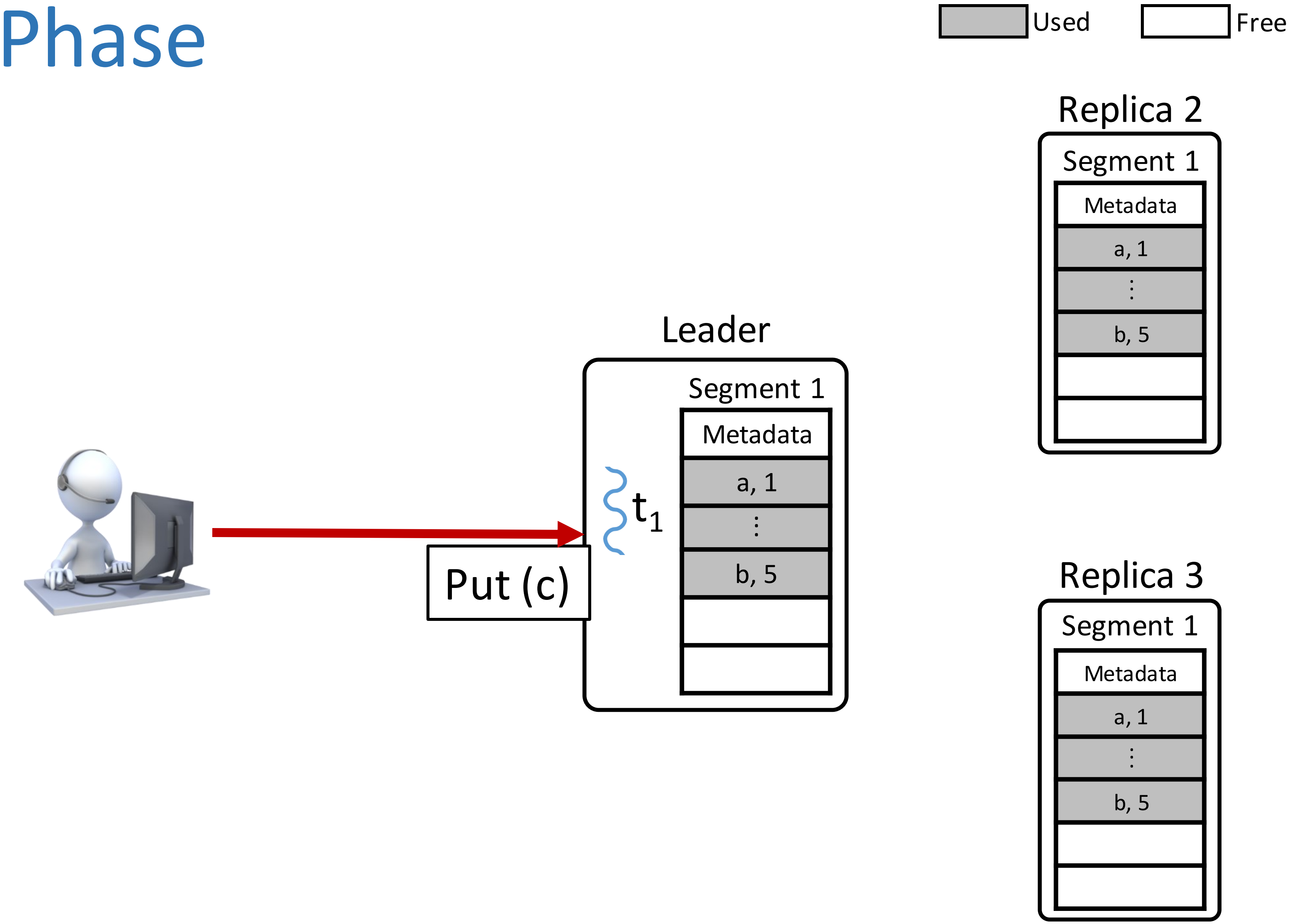
- Put operations processing
  - Replication phase ①②
  - Local apply phase ③
  - Remote apply phase ④



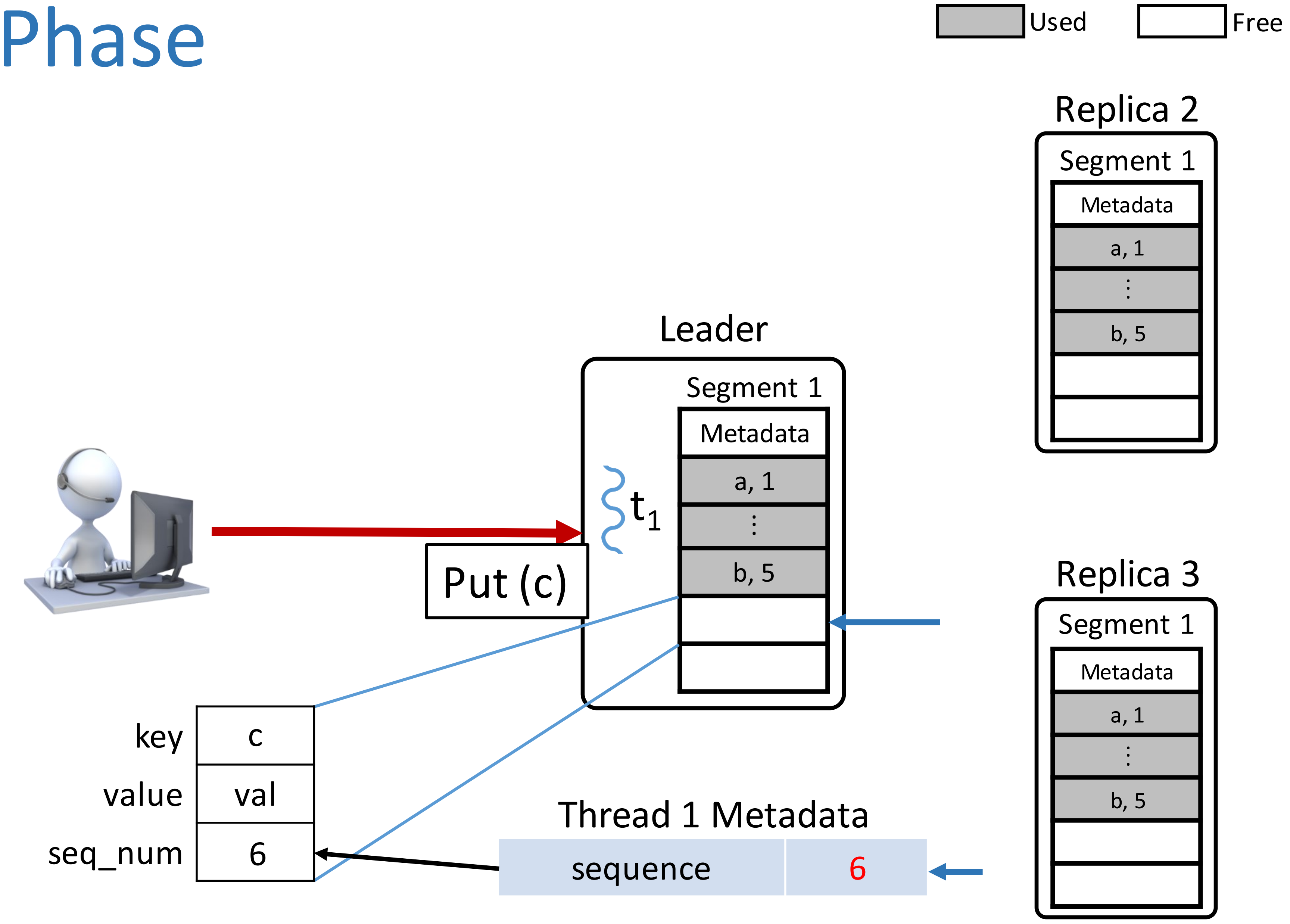
One RDMA Write for replication

One async RDMA Write for apply

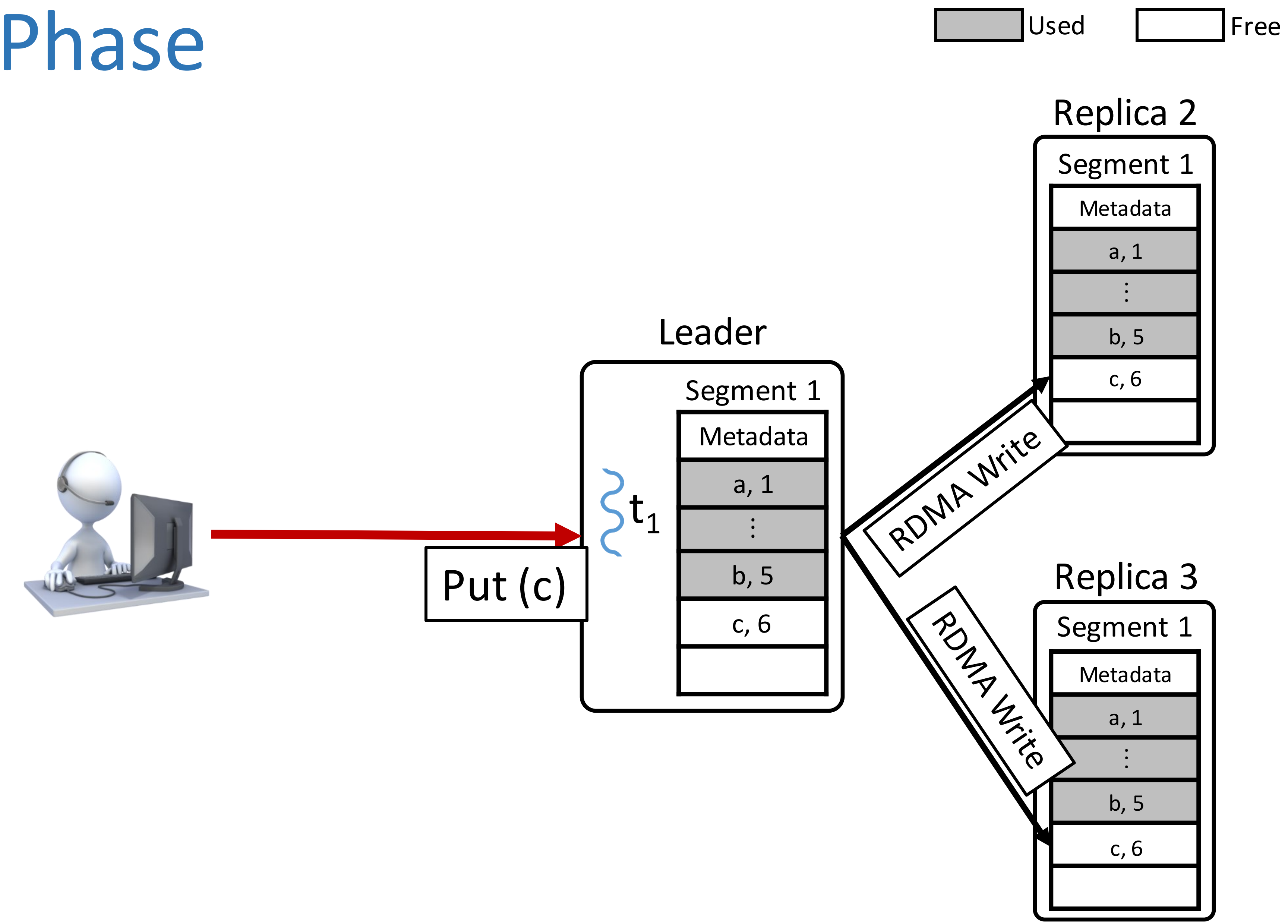
Replication Phase



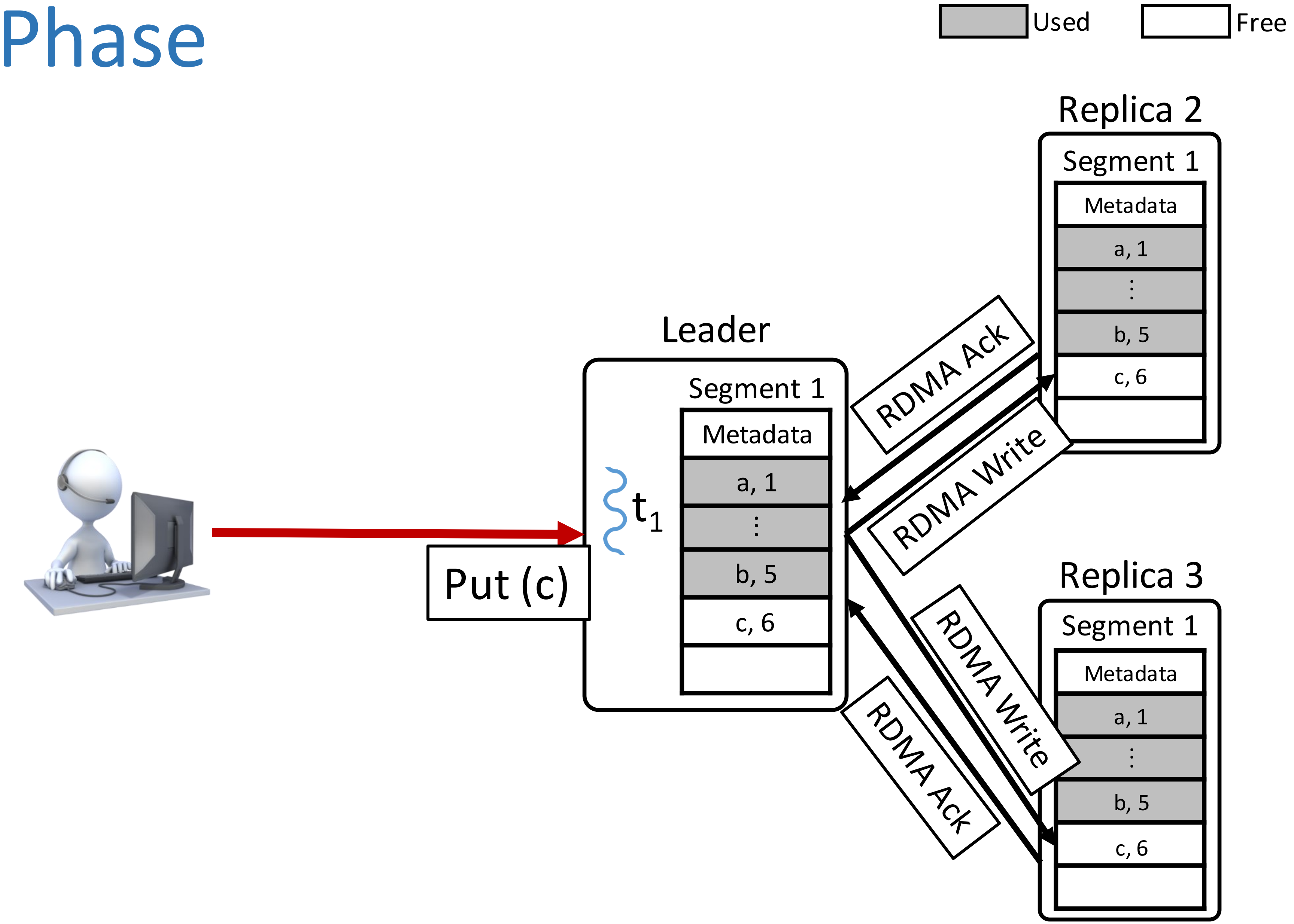
Replication Phase



Replication Phase

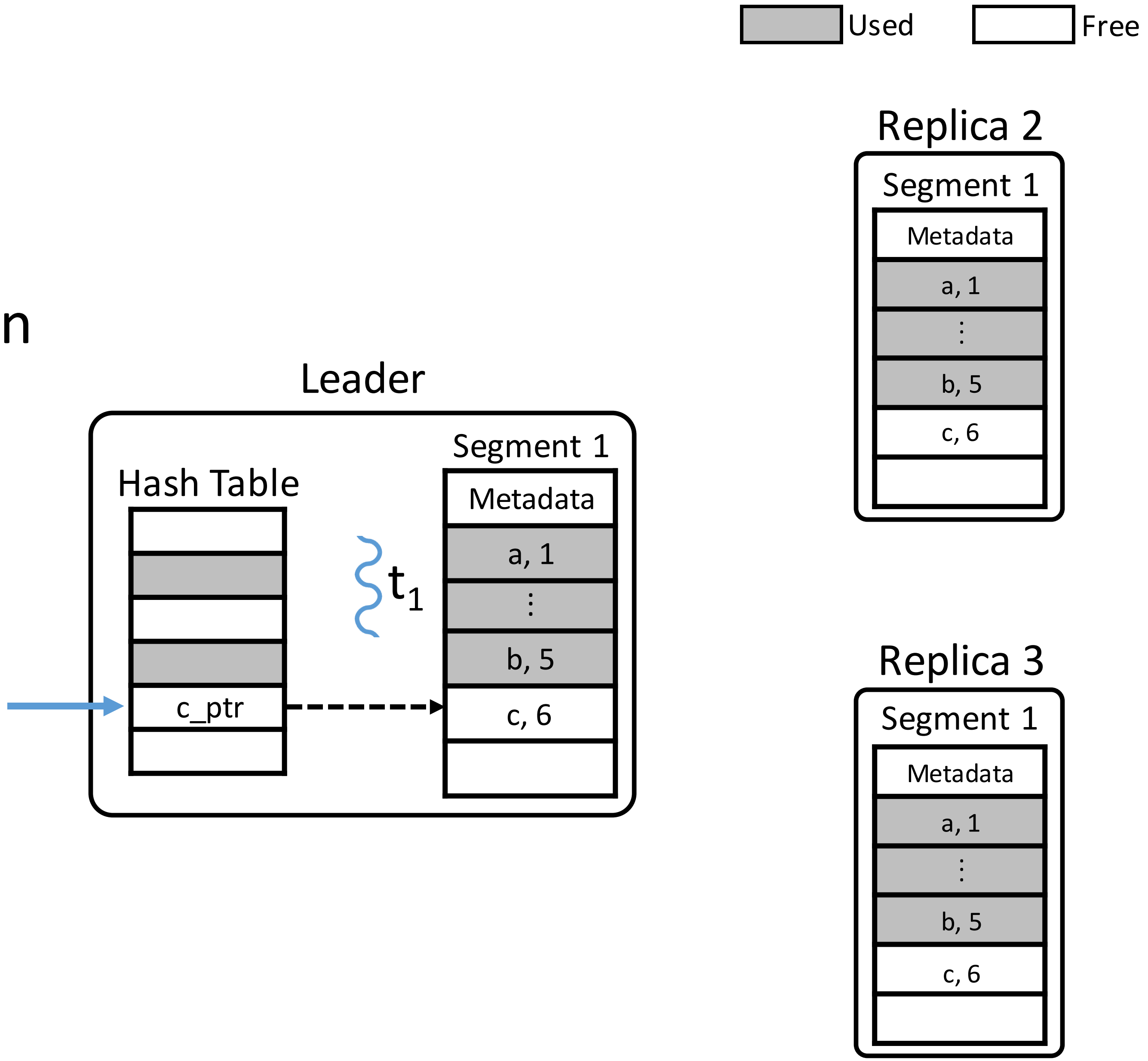


Replication Phase



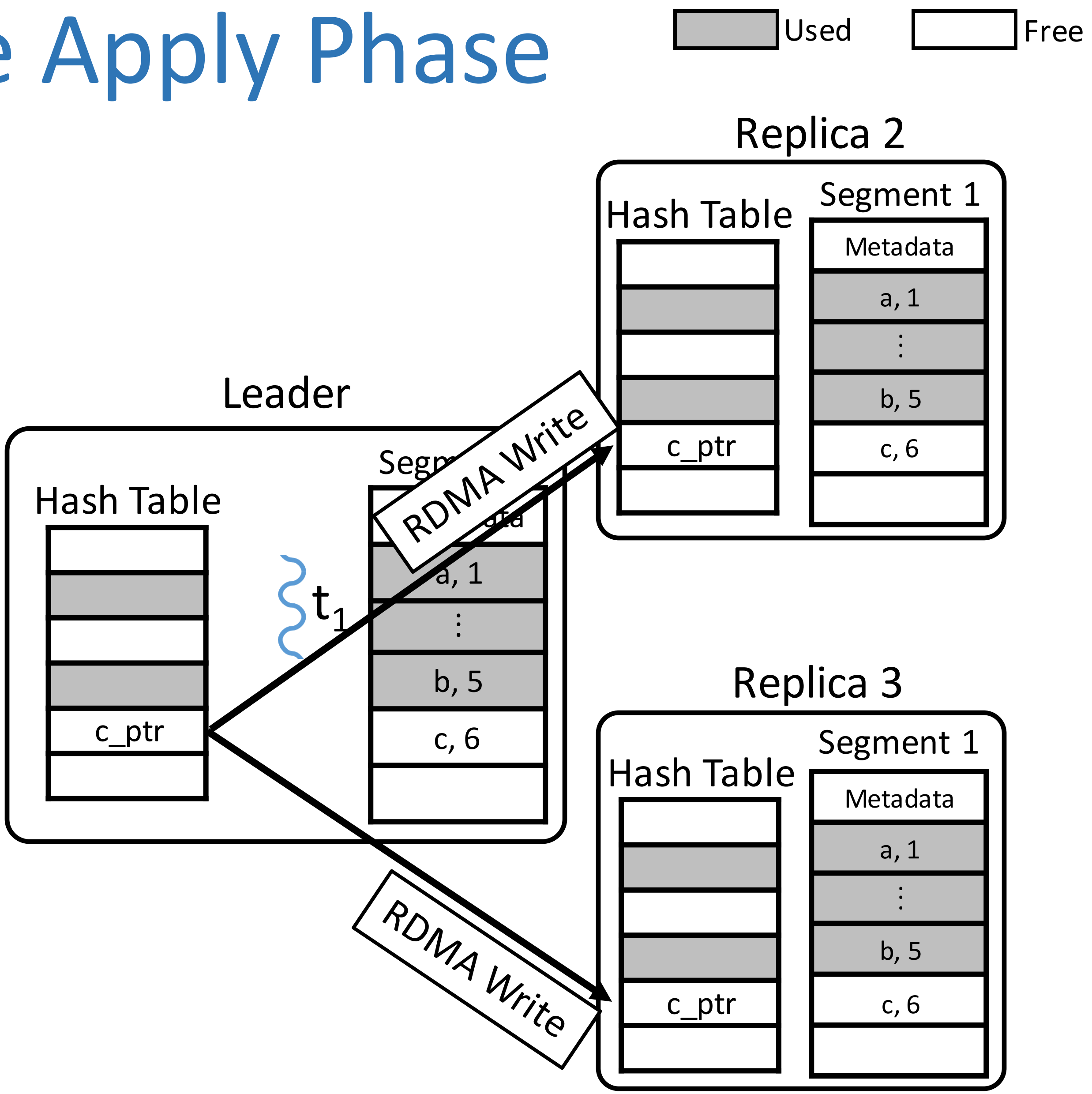
Local Apply Phase

- The leader applies the operation to its hash table
- Hashes the key to find the hash table entry
- Terminates probing if
  - Finds an empty entry
  - Finds an entry pointing to the same key



Asynchronous Remote Apply Phase

- The leader updates followers hash tables lazily
  - Using RDMA Write

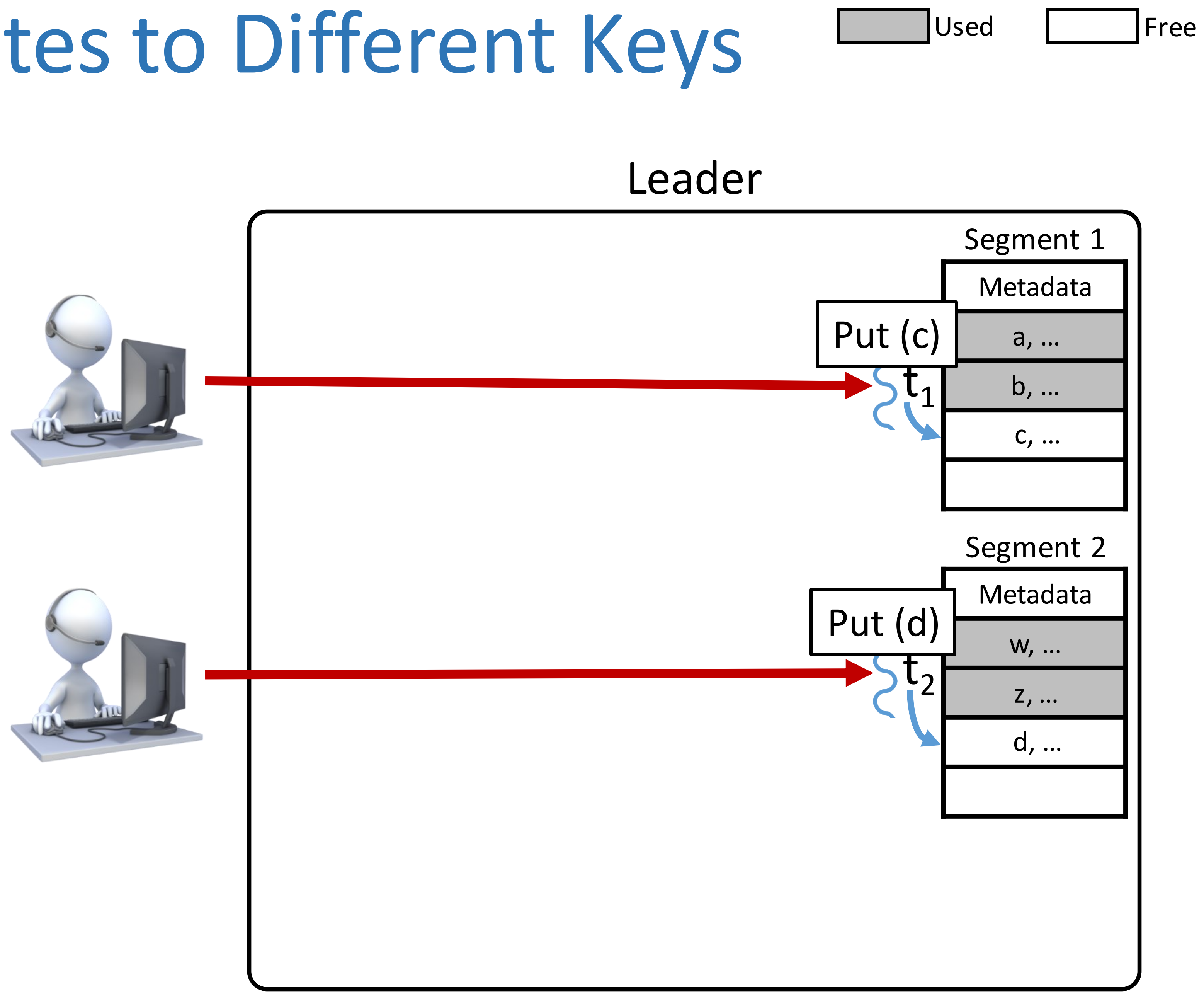


LoLKV is a Complete System

- Concurrent writes
- Fault tolerance
  - Follower failure
  - Leader failure
  - Torn writes
- Leader election protocol
- Garbage collection protocol
- Proof of correctness
  - Proof sketch
  - TLA+ model checking

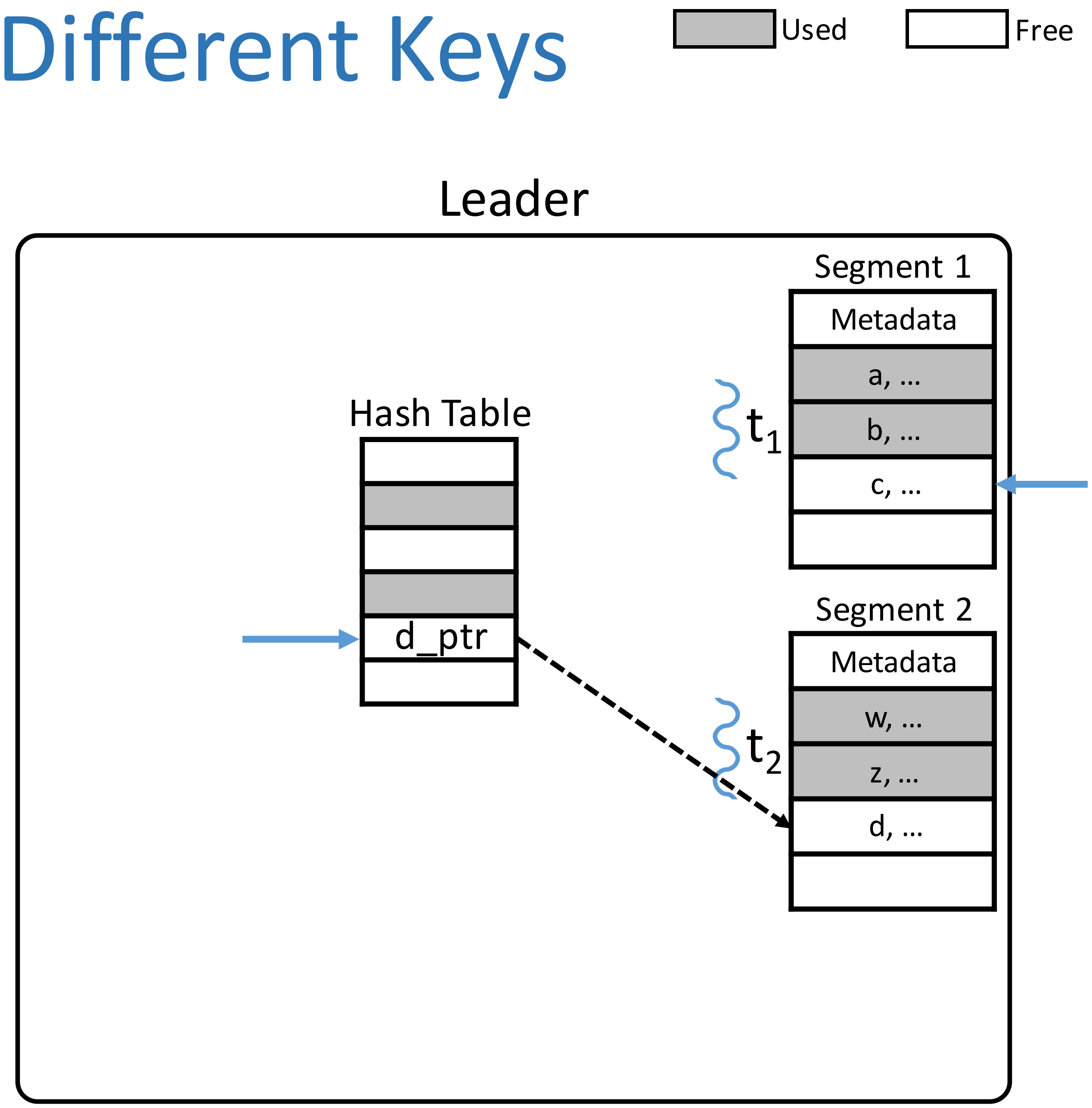
Concurrent Writes to Different Keys

- Objects are committed in parallel
- Objects are applied in parallel



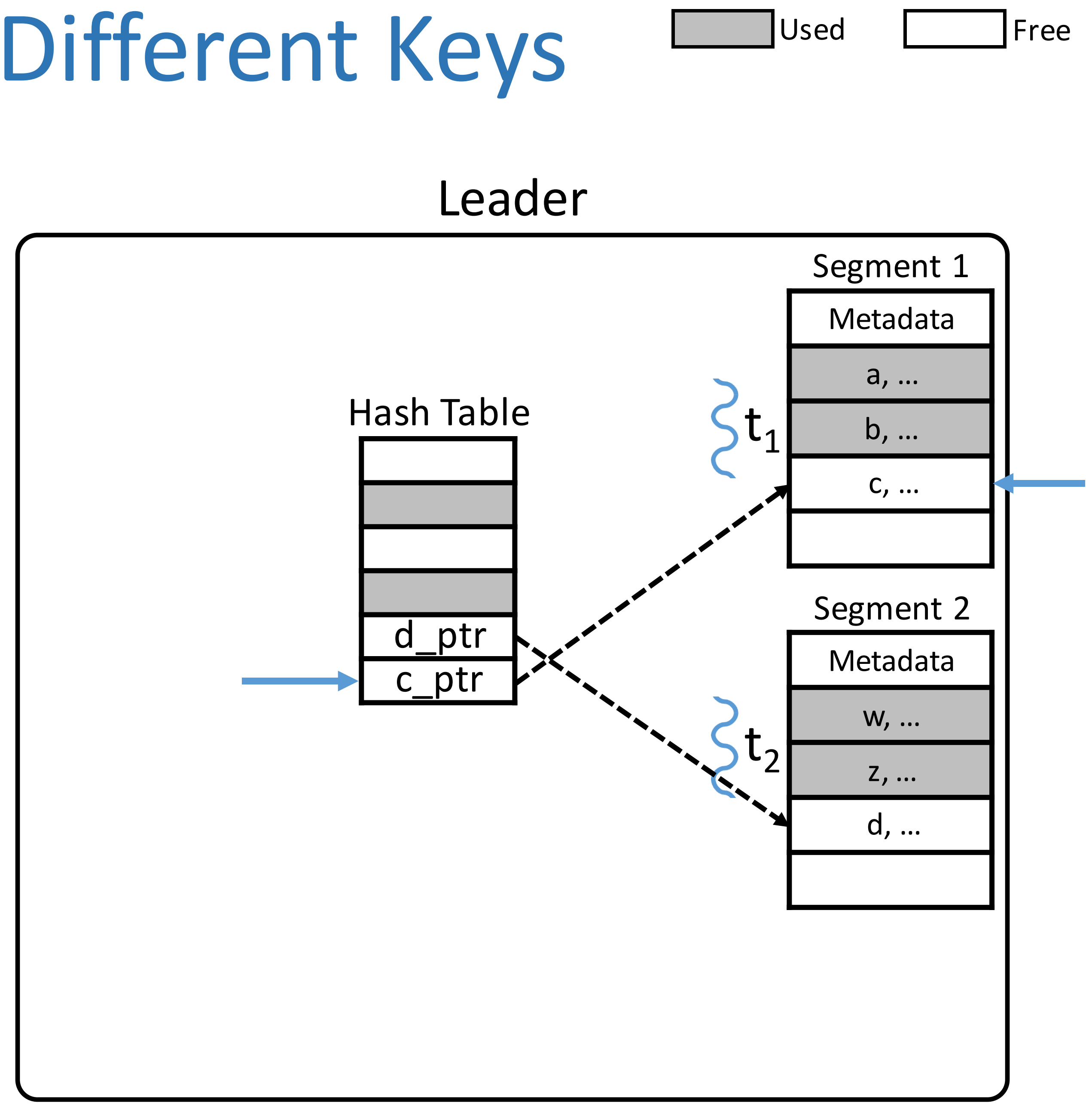
Concurrent Writes to Different Keys

- Objects are committed in parallel
- Objects are applied in parallel
- Hash table is updated using CAS
  - Handles concurrent access



Concurrent Writes to Different Keys

- Objects are committed in parallel
- Objects are applied in parallel
- Hash table is updated using CAS
  - Handles concurrent access
- If CAS fails, repeat linear probing

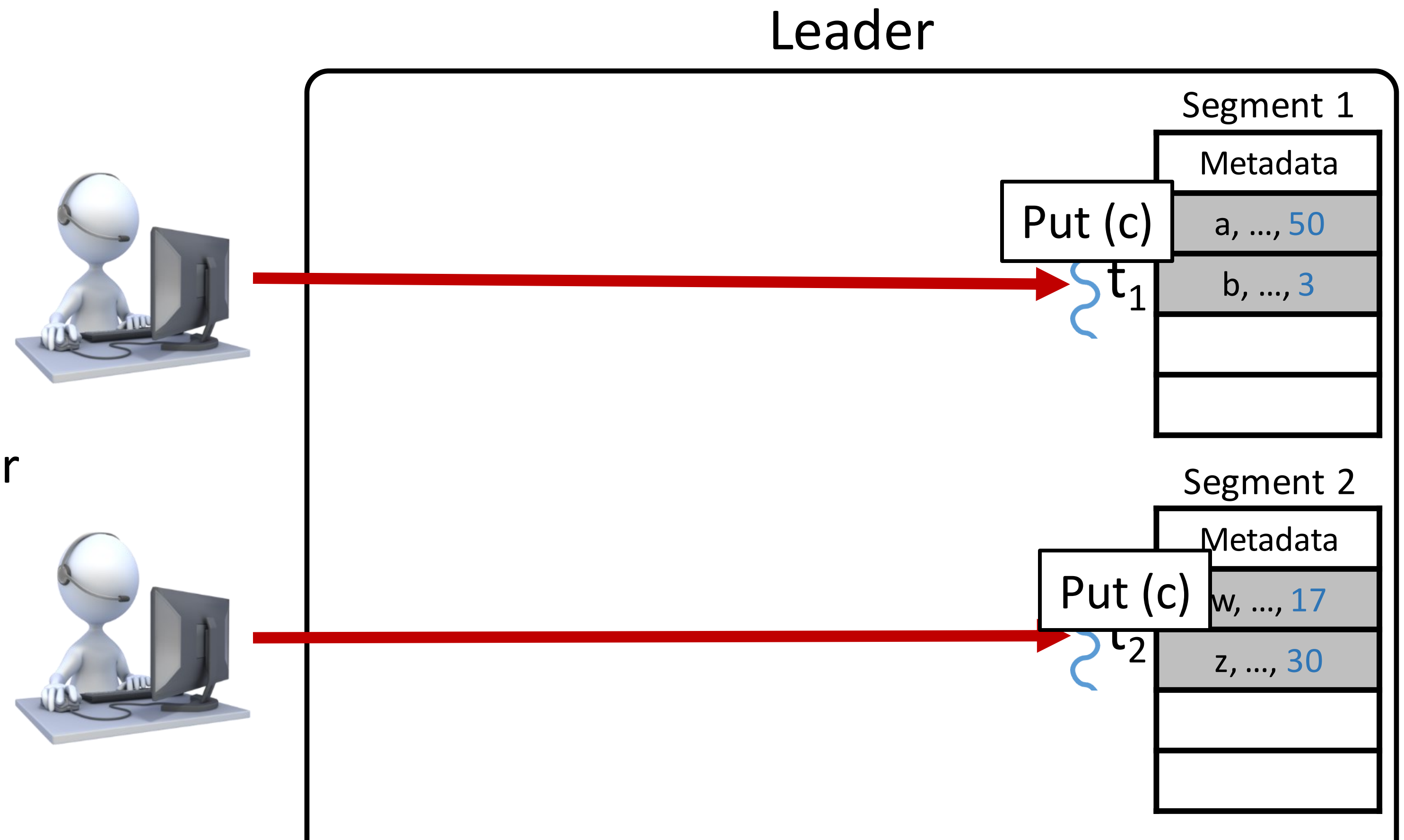


Concurrent Writes to the Same Key

Used Free

- Incarnation Array
  - Array of atomic counters

- Each Put has an incarnation number

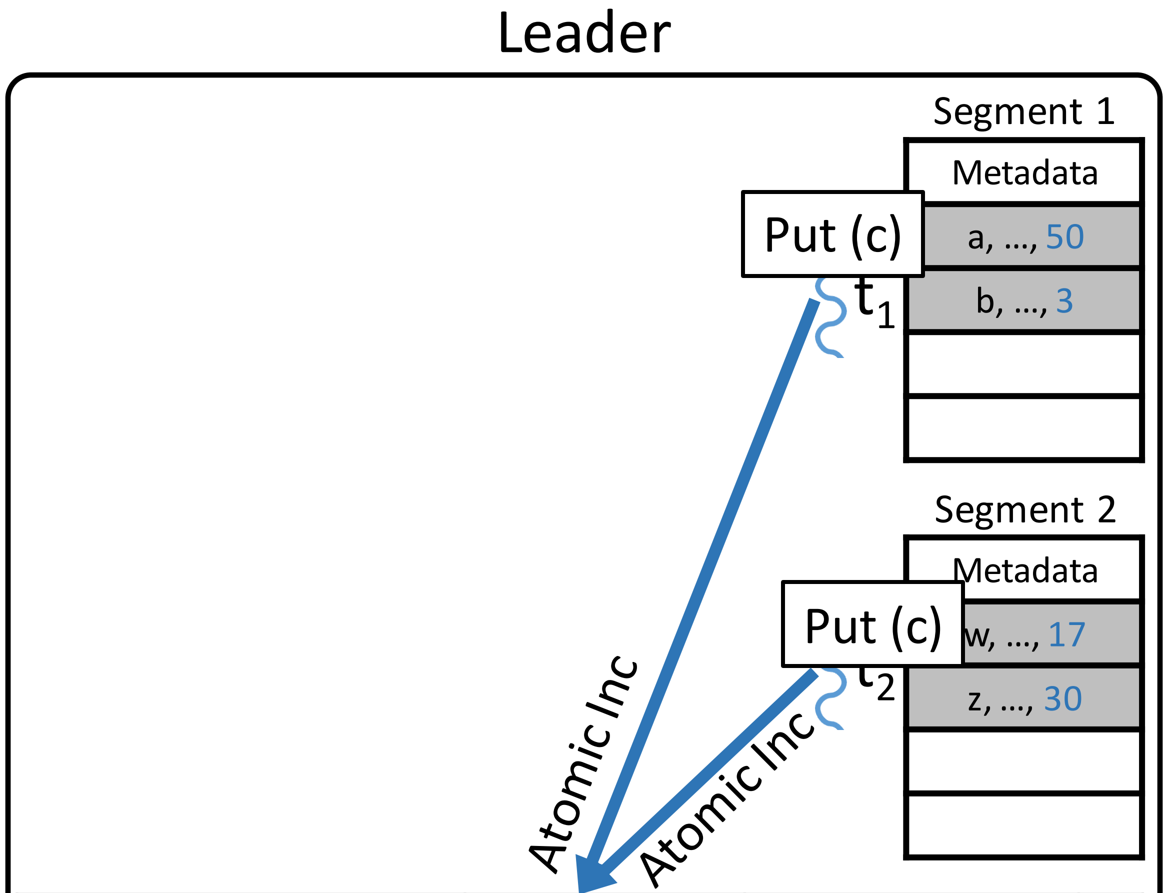


Key range	[0, 4096)	[4096, 8192)	…
Incarnation Number	100	115	…

Concurrent Writes to the Same Key

Used Free

- Incarnation Array
  - Array of atomic counters
- Each Put has an incarnation number

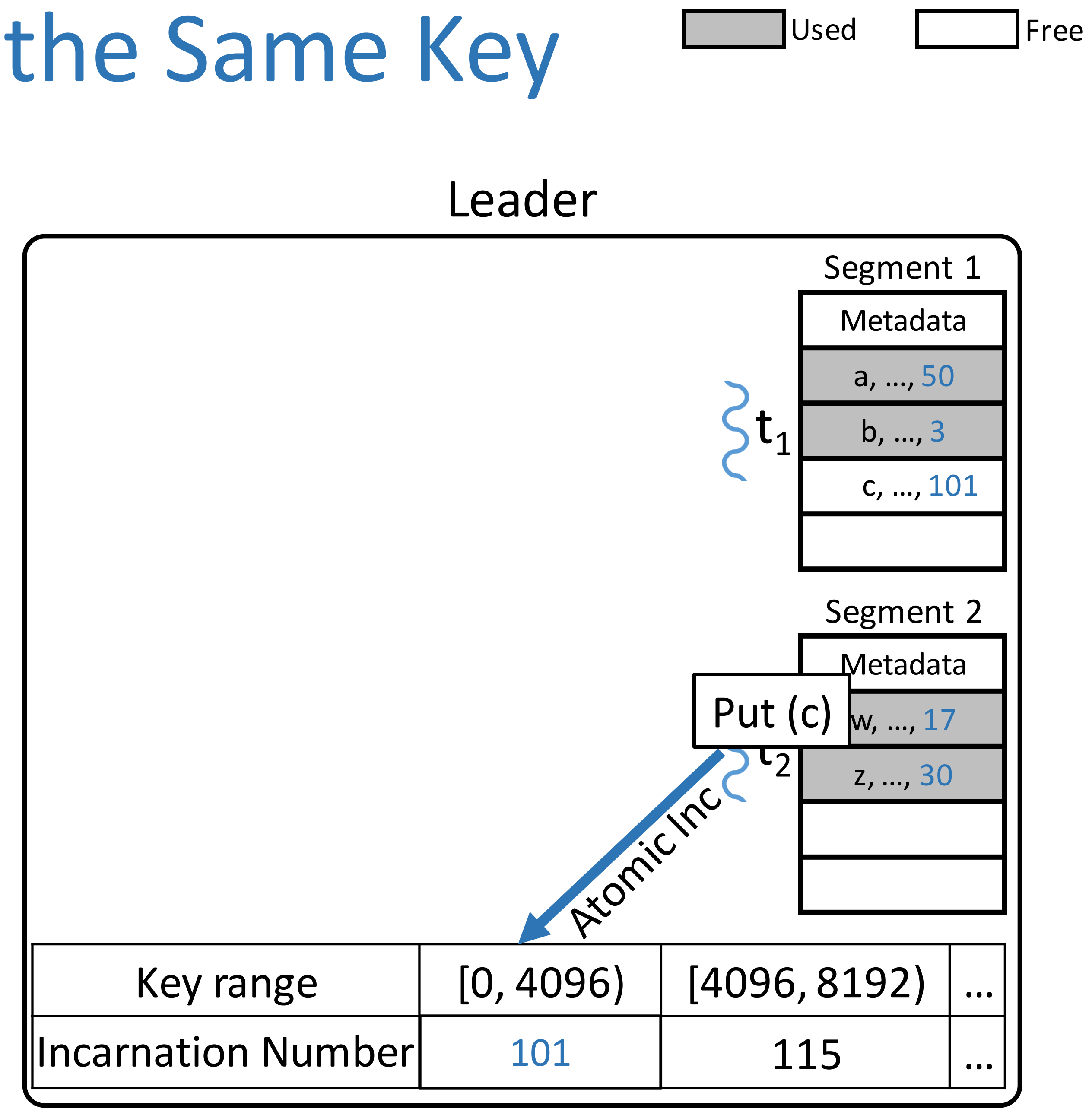


Key range	[0, 4096)	[4096, 8192)	…
Incarnation Number	100	115	…

Concurrent Writes to the Same Key

- Incarnation Array
  - Array of atomic counters

- Each Put has an incarnation number

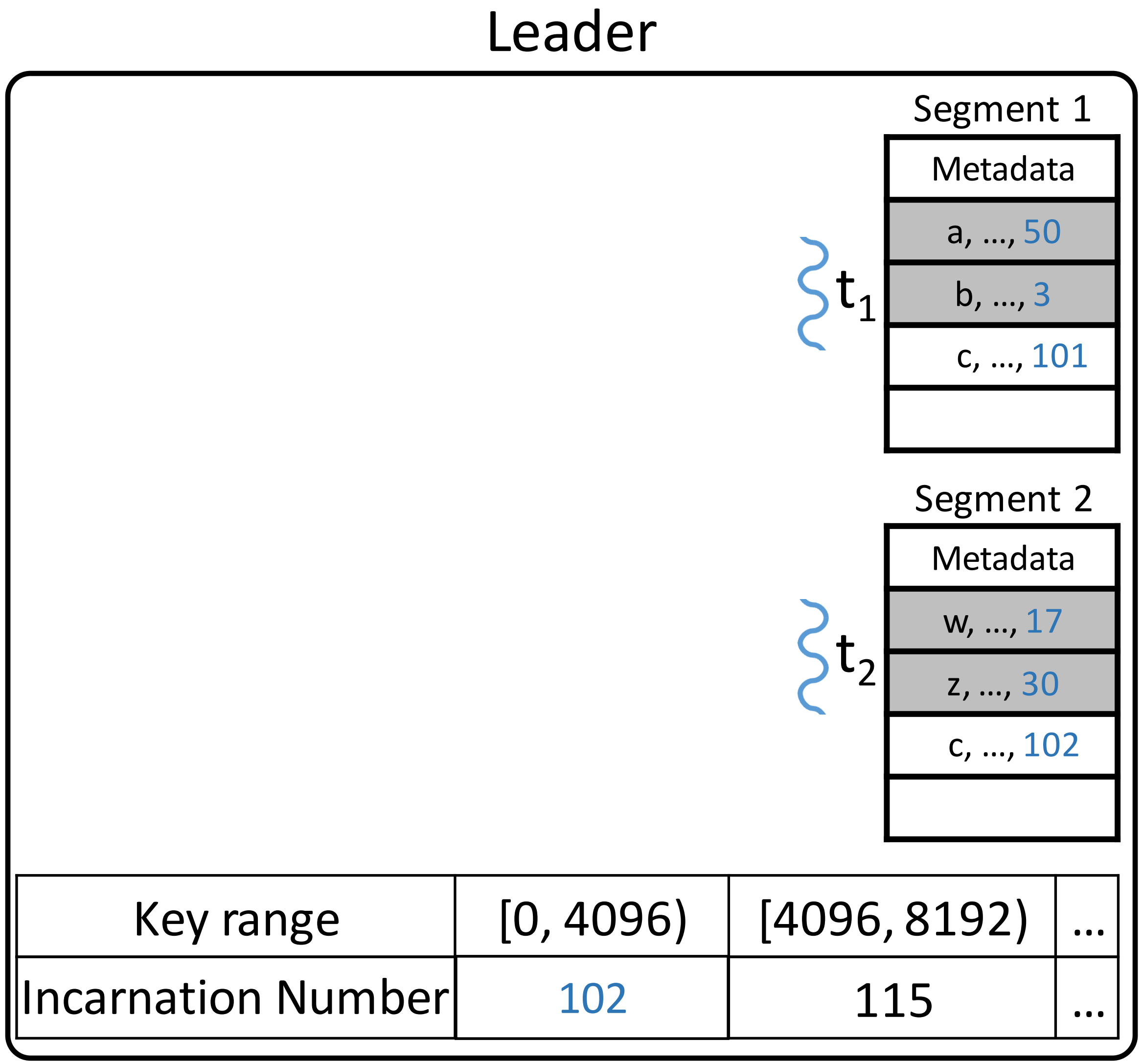


Key range	[0, 4096)	[4096, 8192)	…
Incarnation Number	101	115	…

Concurrent Writes to the Same Key

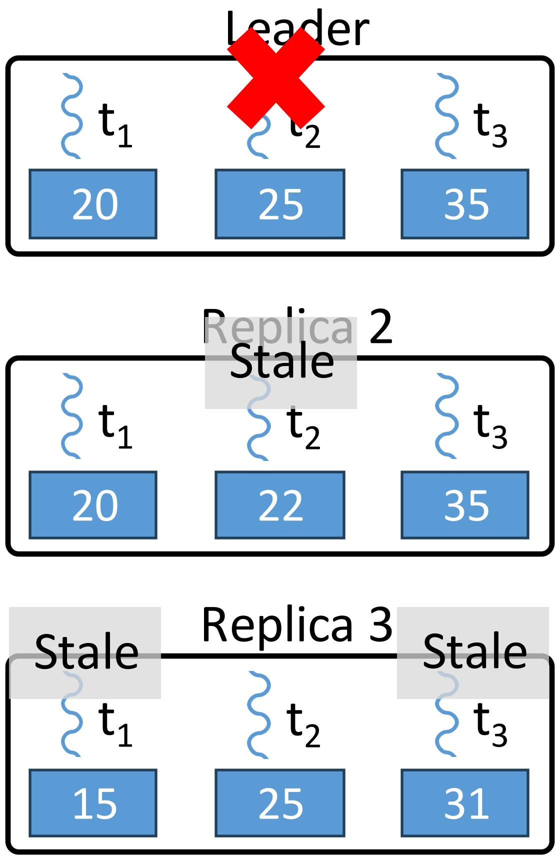
Used Free

- Incarnation Array
  - Array of atomic counters
- Each Put has an incarnation number
- Orders Puts for the same key



Leader Election

- Any replica can become a leader
- The new leader might be stale for some threads
  - Different threads replicate operations on different majorities
- State synchronization brings the new leader up-to-date



Evaluation

Alternatives (best configuration per system)
- DARE (8 shards)
- APUS (7 shards)
- Mu (4 shards)
- uKharon (4 shards)

Workloads
- YCSB benchmark
- Different workload skewness
- Different read-to-write ratios

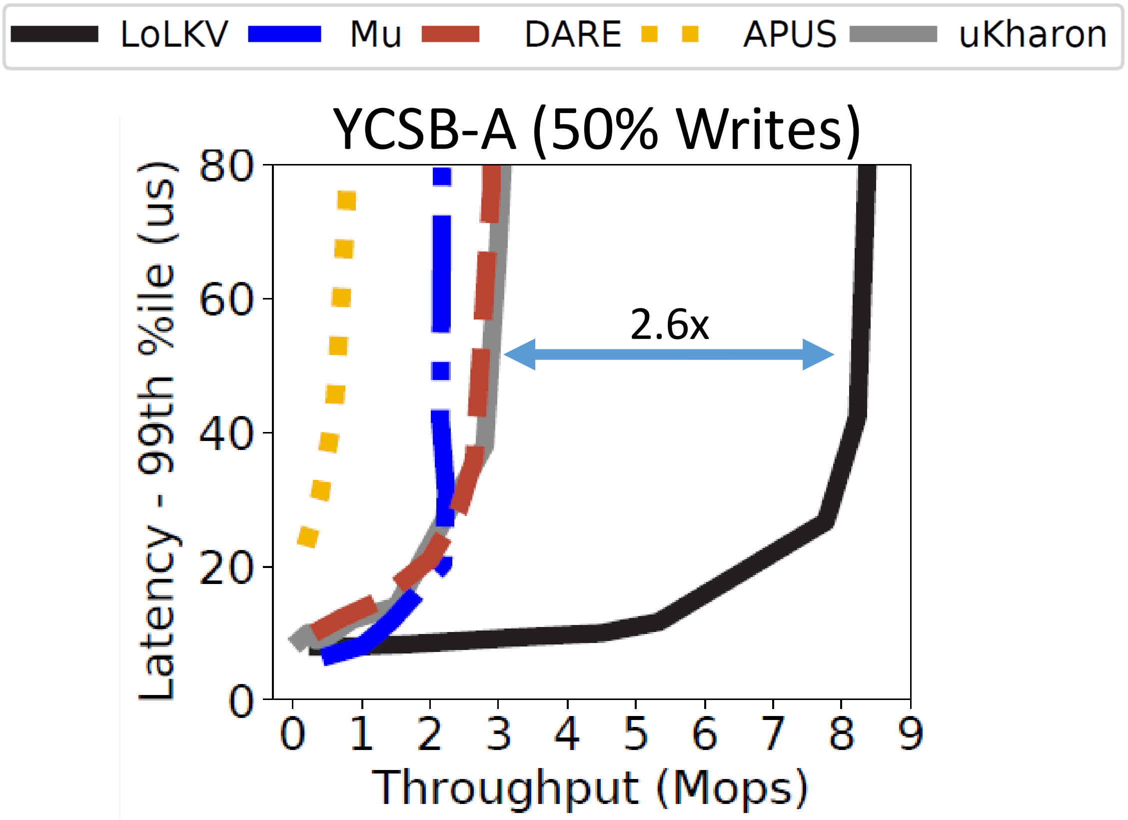
Metrics
- Throughput
- Latency
- Scalability

Testbed

12 machines in CloudLab
  - 8-core CPU (2.1 Ghz)
  - 16 GB of RAM
  - Infiniband network (56 Gbps)
  - Mellanox CX3

Uniform Workload

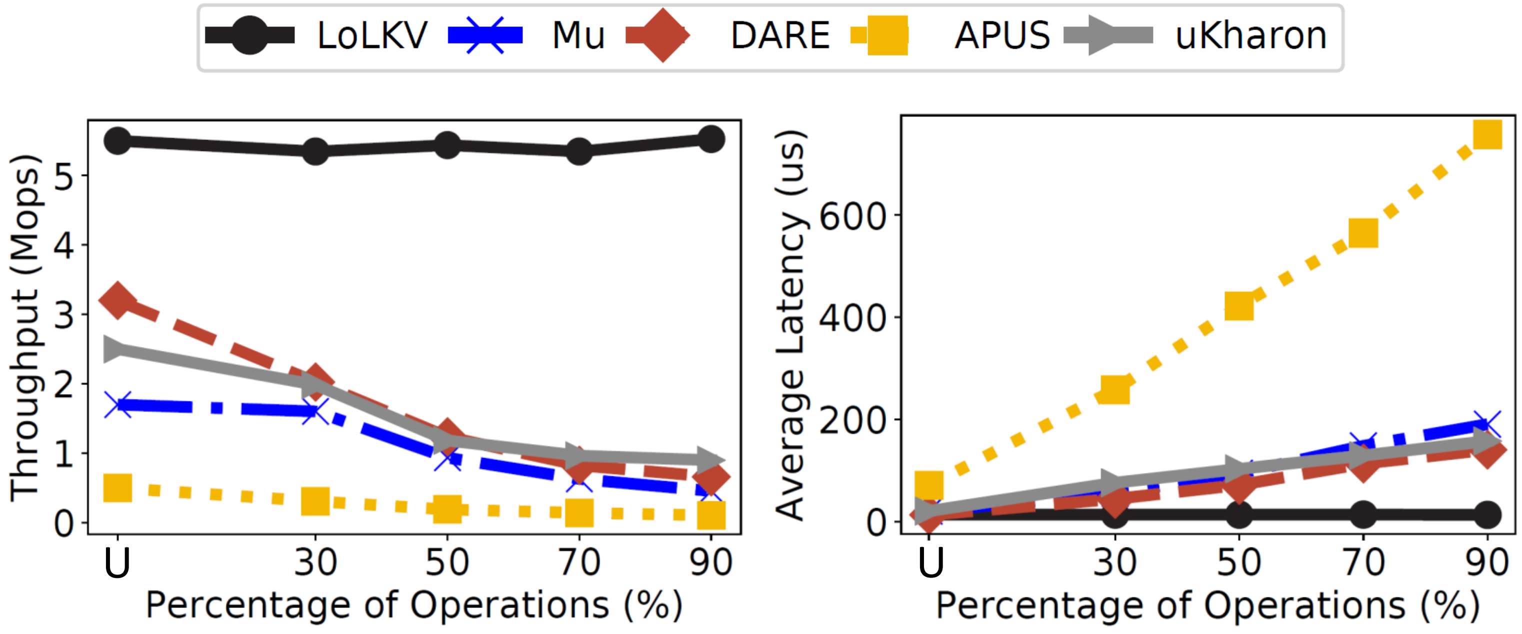
- APUS requires two RDMA Writes
- Mu and uKharon require one RDMA Write
- DARE requires two RDMA Writes



LoLKV outperforms other systems in terms of throughput and latency

Skewed Workload

- Uniform write-only workload
- One popular shard
- Control the percentage of operations served by that shard



Other systems performance decreases with skewness

- Popular shard is overwhelmed

LoLKV efficiently handles skewed workloads

Conclusion

- LoLKV is a low-latency, highly-concurrent, and linearizable object store
- LoLKV adopts a novel logless design
  - Eliminates the serialization point
  - Eliminates unnecessary memory copy operations
- LoLKV adopts a novel multi-threaded shard design
  - Efficient for both uniform and skewed workloads
  - Eliminates resource fragmentation
- LoLKV outperforms state-of-the-art systems
  - At least 1.7× higher throughput
  - At least 20% lower latency
  - Better scalability