Exploiting Commutativity For Practical Fast Replication

Seo Jin Park and John Ousterhout

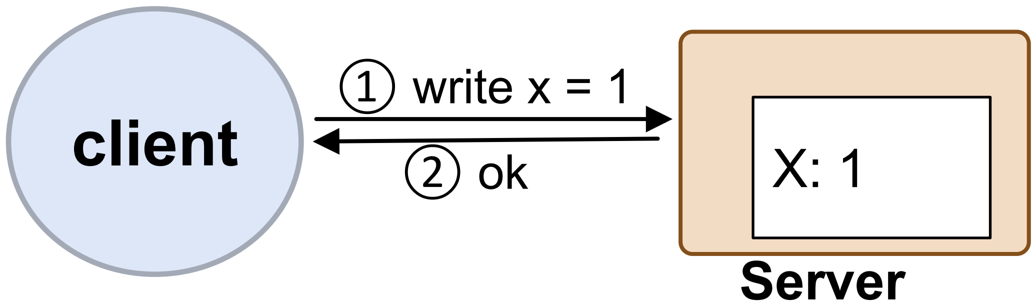
PLATFORMLAB

Stanford University

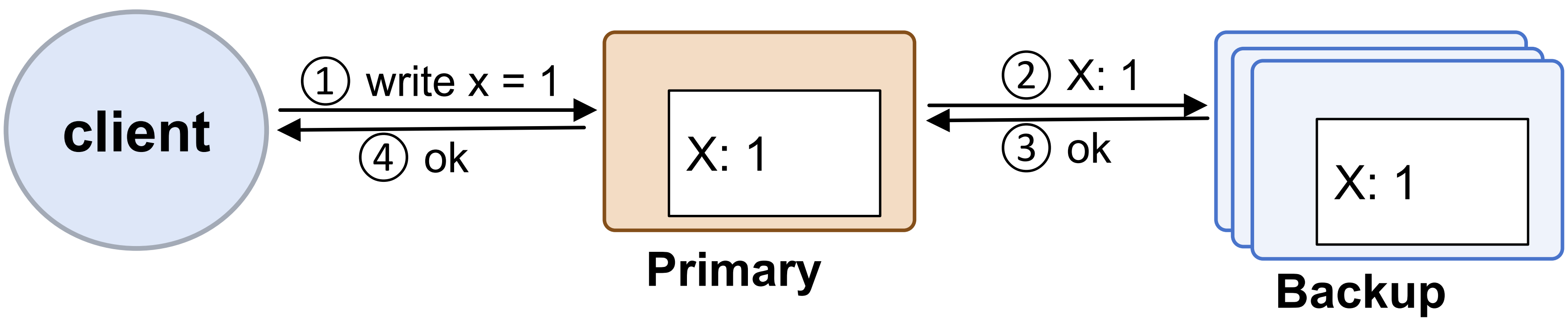
Overview

- **Problem: replication adds latency and throughput overheads**
- **CURP: Consistent Unordered Replication Protocol**
  - Replicate before ordering client requests.
    - Commutative operations don't need to be ordered for consistency.
  - Clients directly replicate to witnesses in 1 RTT.
- **Result**
  - RAMCloud's performance improvements
    - Latency: 13.8 μs → 7.3 μs
    - Throughput: 187 kops/sec → 710 kops/s (~3.8x)
  - Redis cache is now fault-tolerant with small cost (~12%)

Replication Doubles Latencies



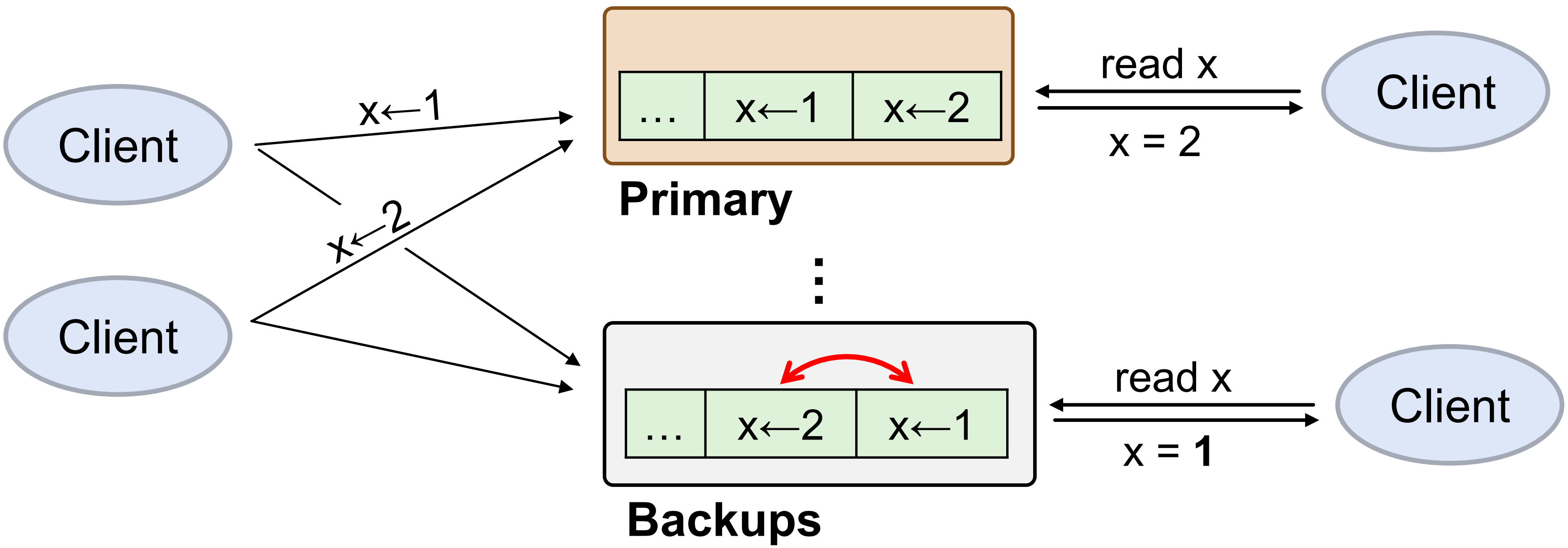
- **Unreplicated Systems: 1 RTT for operation**



- **Replicated Systems: 2 RTTs for operations**

Why can't clients directly replicate to backups?

Strawman 1 RTT Replication

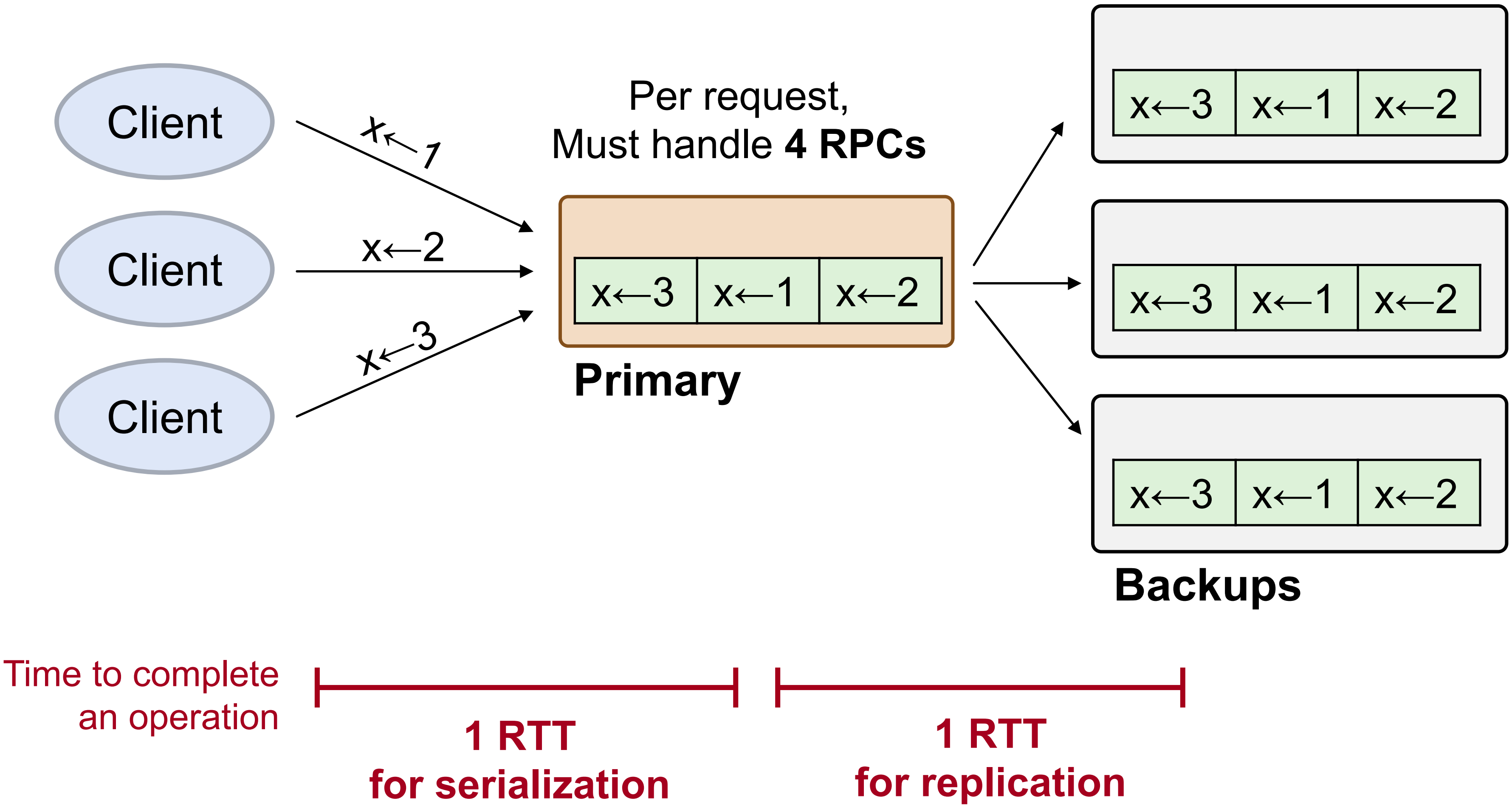


Strong consistency is broken!

Requirements for Consistent Replication

- **Replication protocols with concurrent clients must solve two problems:**
  - **Consistent Ordering**: all replicas should appear to execute operations in the same order
  - **Durability**: once completed, an operation must survive crashes.
- **Previous protocols combined the two requirements**
  - A server first order client requests
  - Then replicate the totally ordered requests to backups

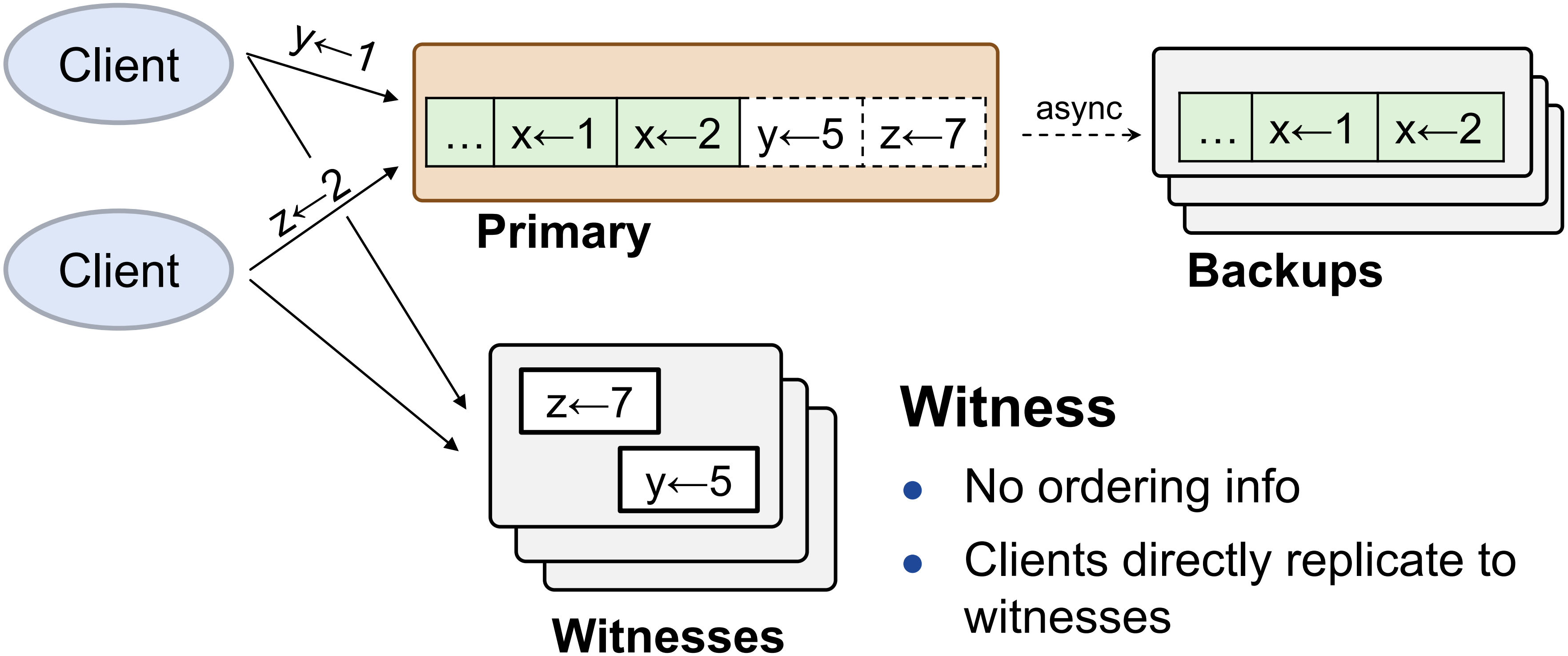
Ordering Before Replication



Exploiting Commutativity to Avoid Ordering

- **For performance:** cannot do totally ordered replication in 2 RTTs
- **There are cases where ordering doesn't matter**
  - When operations are *commutative*
- **CURP's key idea:**
  - When operations commute, replicate before ordering
  - When not, fall back to 2 RTT protocol

Unordered Replication for Durability



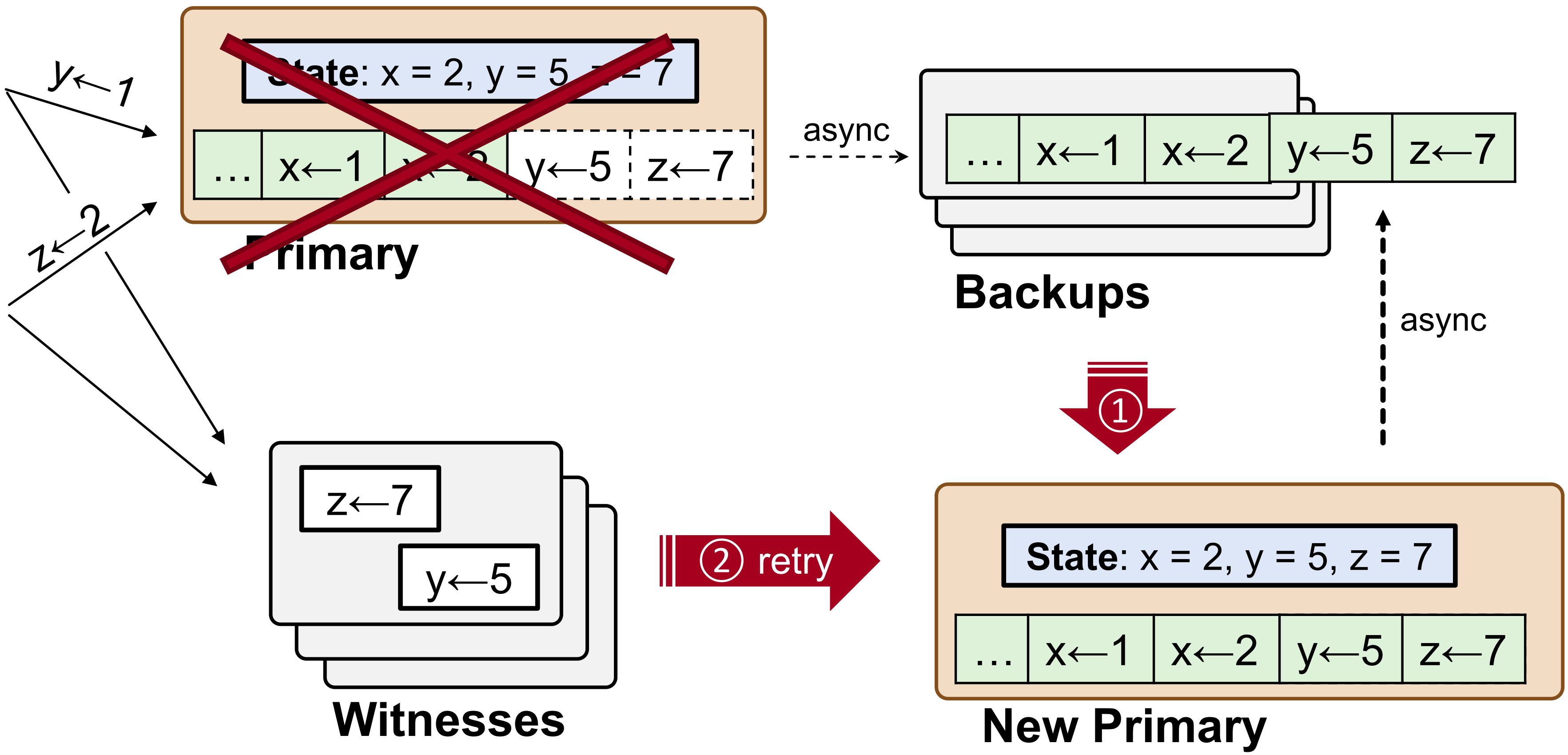
Time to complete an operation

1 RTT

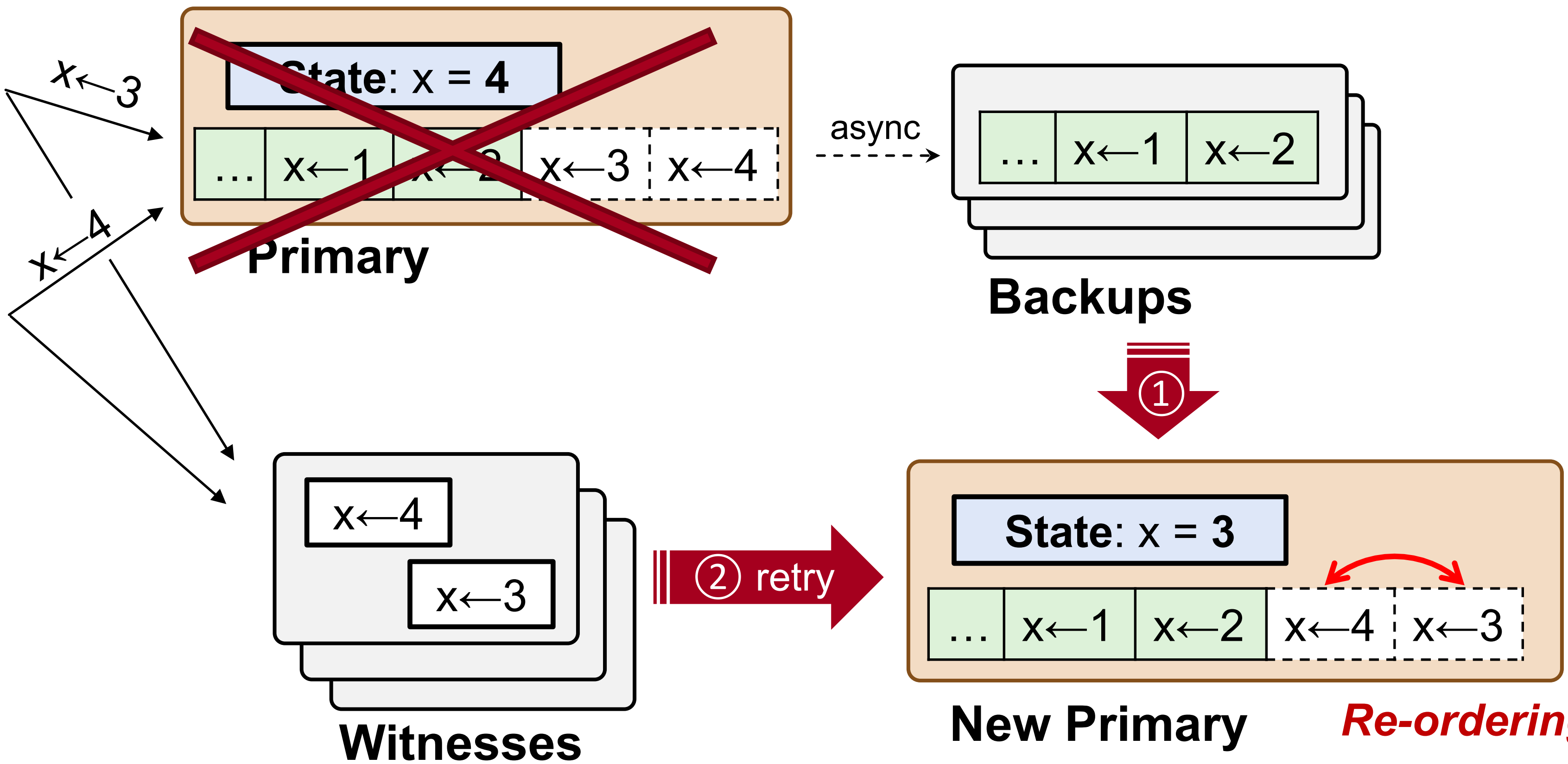
Witness

- No ordering info
- Clients directly replicate to witnesses
- Temporary until primary replicates to backups
- Witness data are replayed during recovery

Simple Crash Recovery



Potential Inconsistency Example



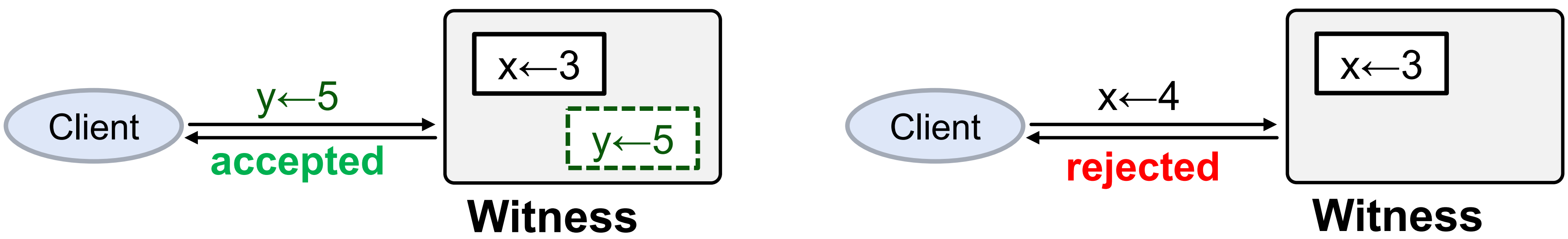
Re-ordering of non-commutative operation causes inconsistencies

3 Problems for Strong Consistency

1. **Witnesses have to detect non-commutative opreations**
2. **Primaries have to detect read of unsynced data**
3. **Must deal with duplicate replays**

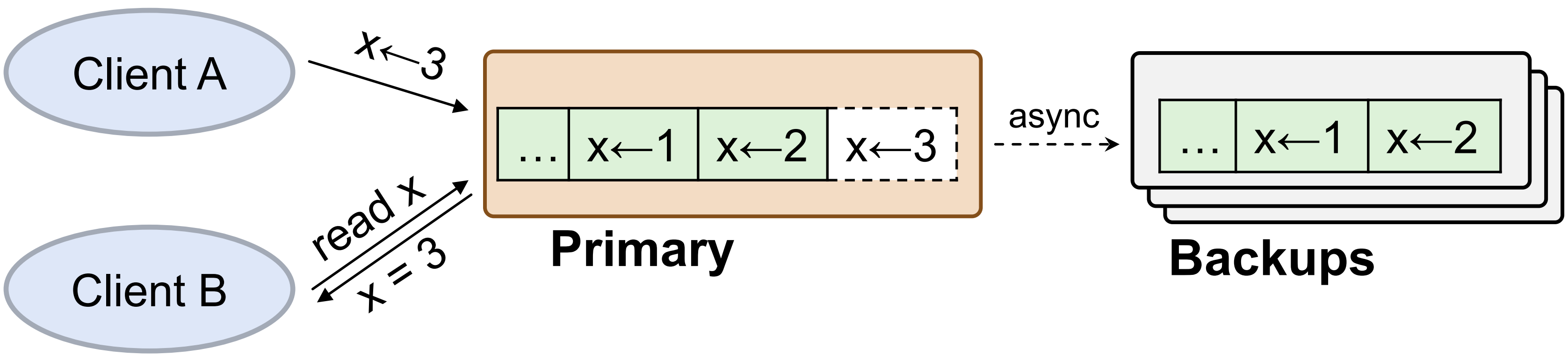
P1. Witnesses Keep Commutativity

- **Witness has no way to know operation order determined by primary**
- **Instead, make it okay to replay in any order**
  - Witness detects non-commutative operations and rejects them
  - Then, client needs to issue explicit sync request to master



P2. Primary Must Not Read Unsynced Data

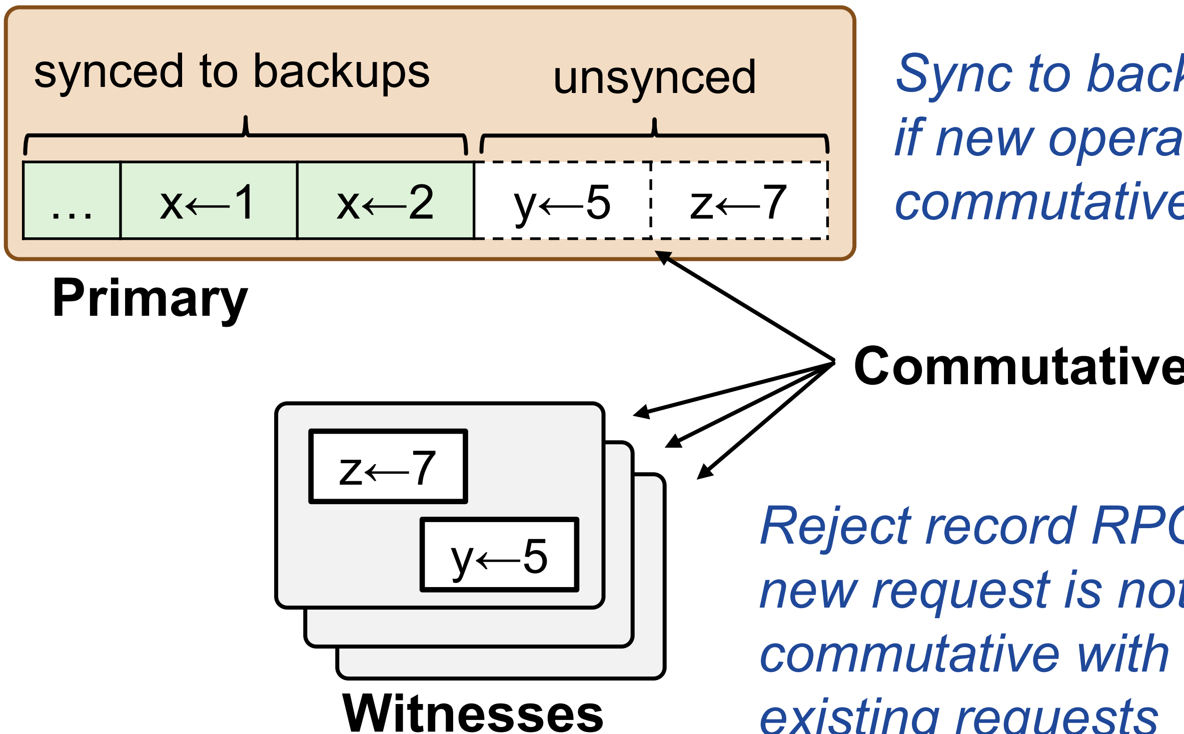
- Primary has no idea whether an operation is made durable in witnesses
- If primary externalize the data that are not yet synced to backups, consistency is in danger



- Primary must check commutativity with existing unsynced operations

P1 & P2. Avoiding Inconsistent Ordering

- **CURP exploits commutativity of operations to defer explicit ordering of client requests**

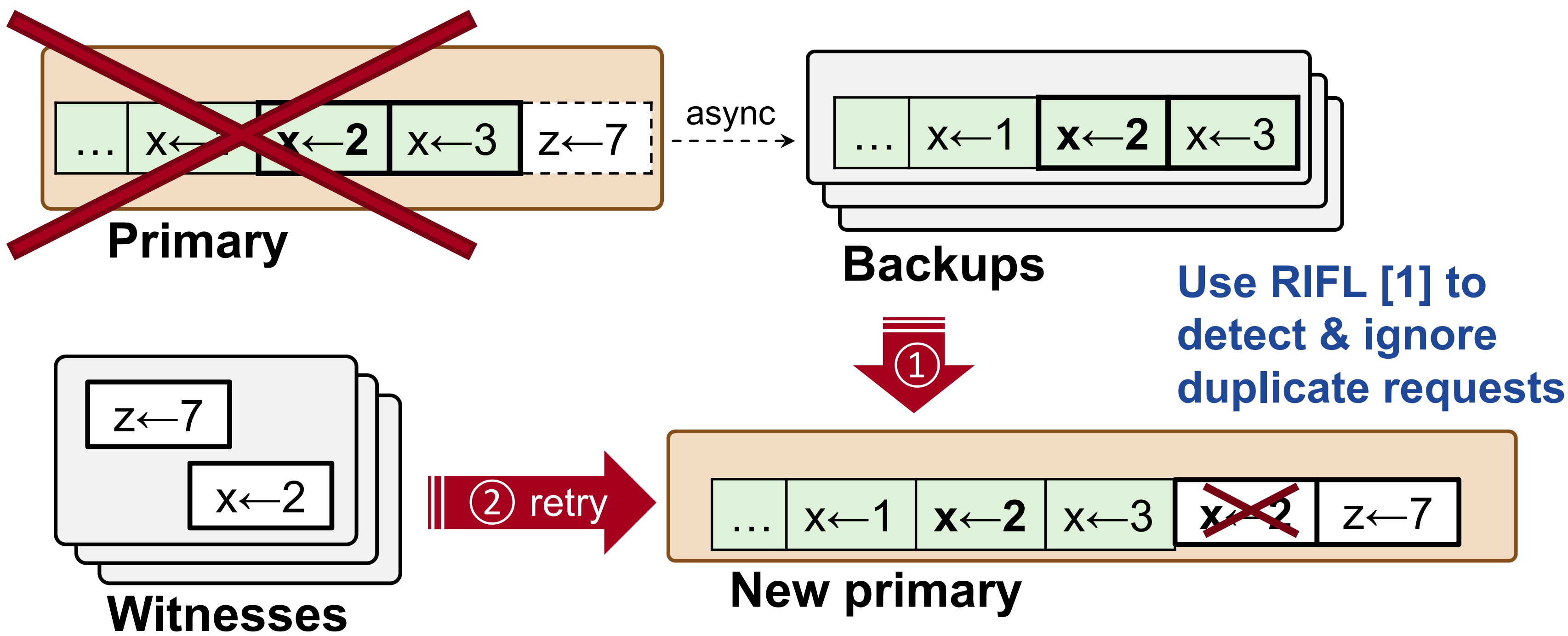


Sync to backups if new operation is not commutative

Reject record RPC if new request is not commutative with the existing requests

P3. Avoiding Duplicate Retries

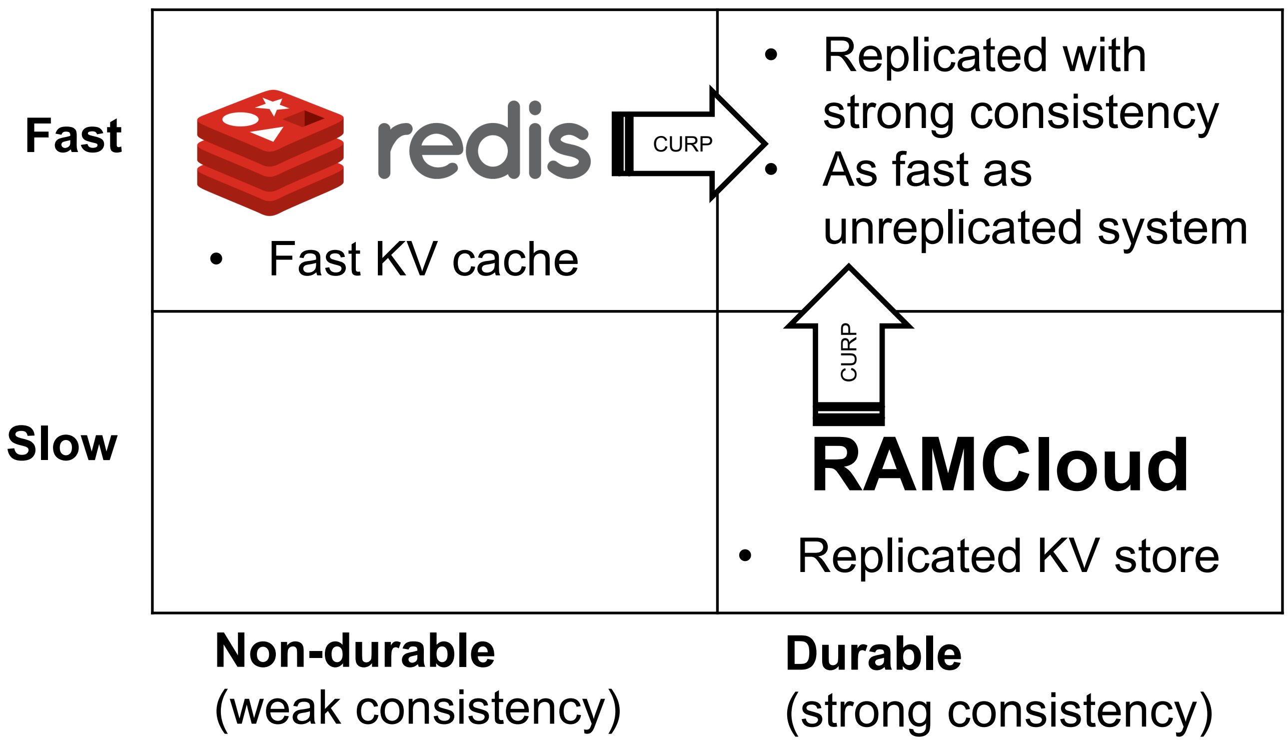
- **Replaying operations recovered by backups endangers consistency [1]**



[1] Implementing Linearizability at Large Scale and Low Latency, Collin Lee*, Seo Jin Park*, Ankita Kejriwal, Satoshi Matsushita, and John Ousterhout, SOSP 15

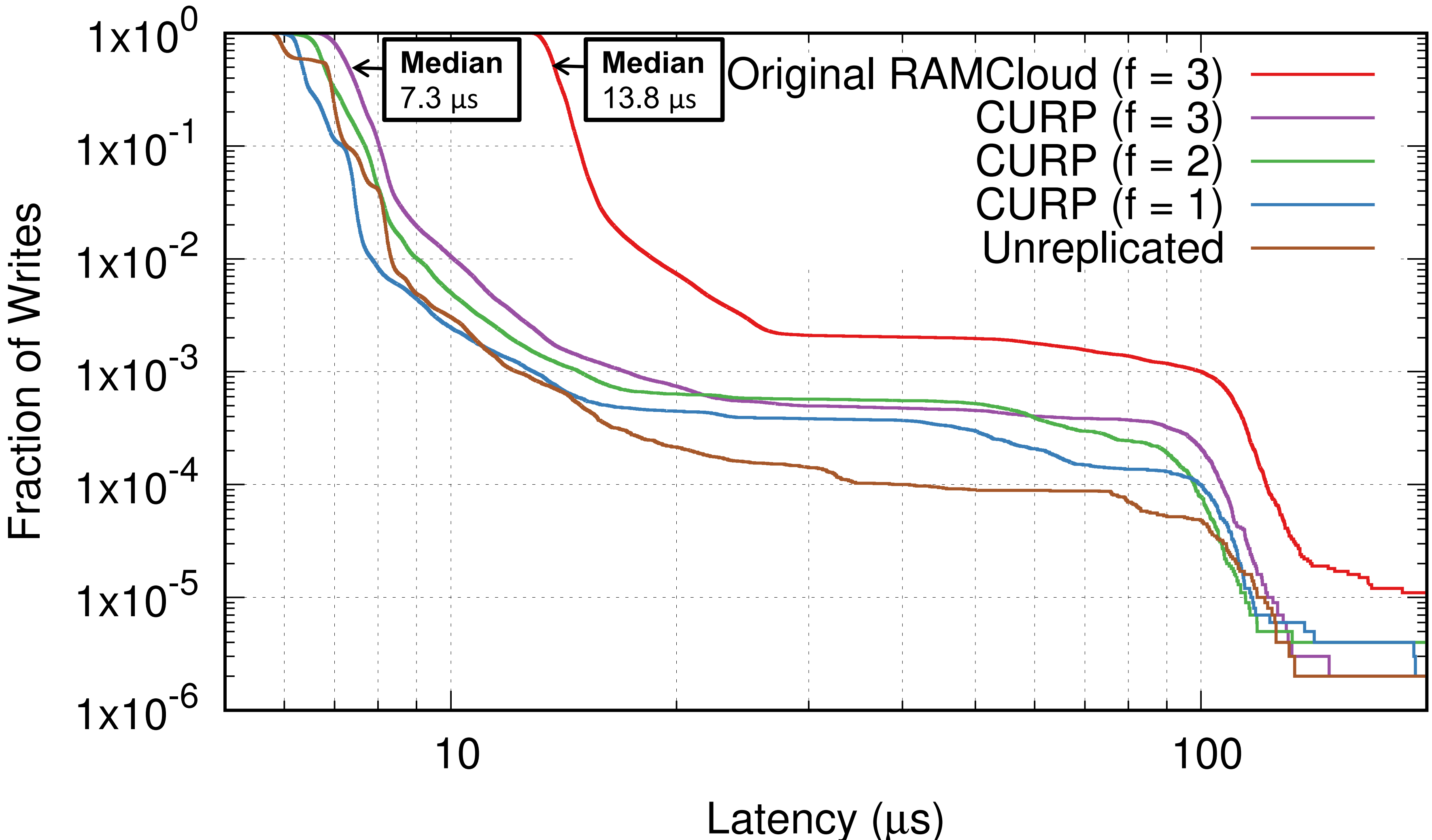
Performance Evaluation of CURP

- Implemented in Redis and RAMCloud



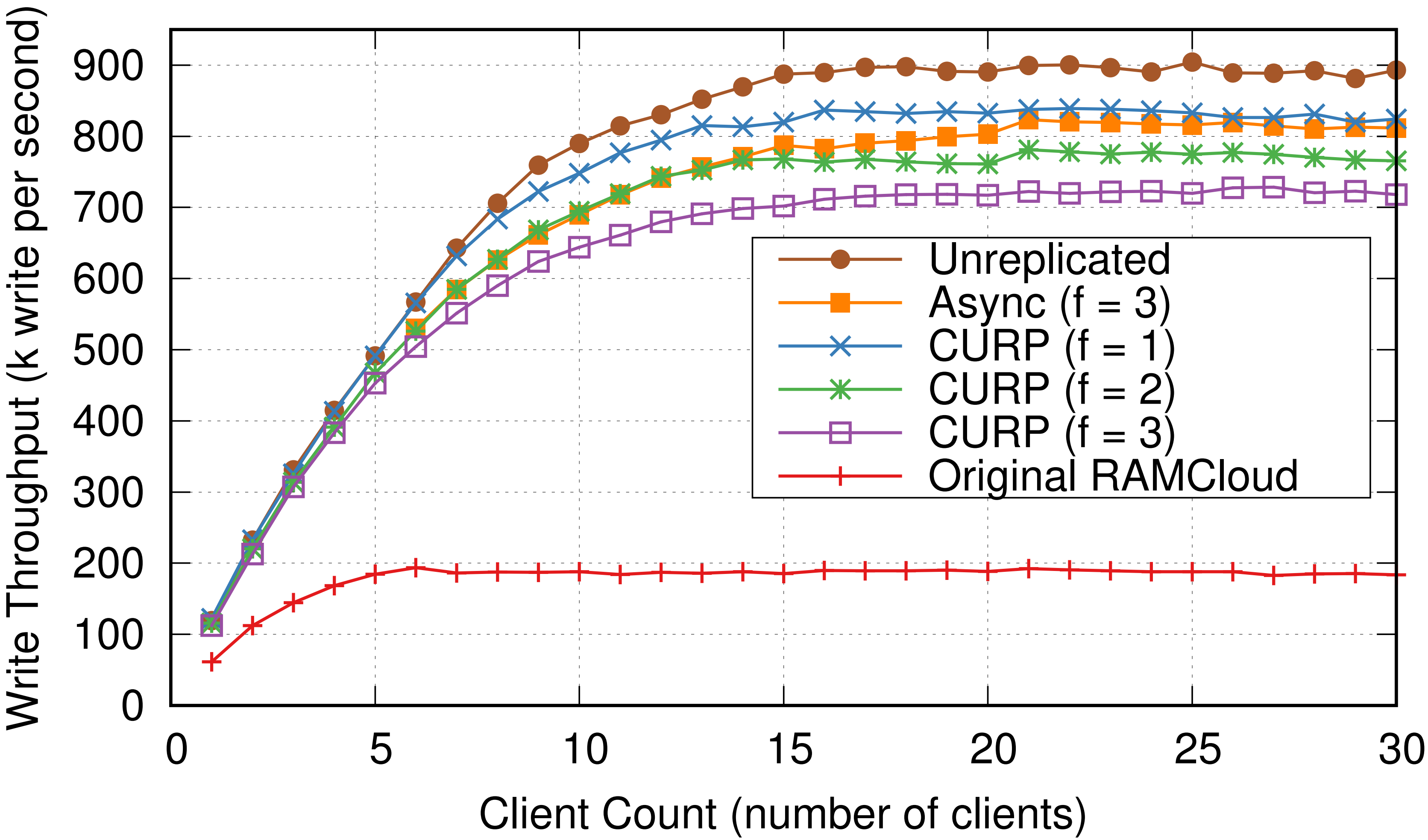
RAMCloud's Latency after CURP

- **Writes are issued sequentially by a client to a master**



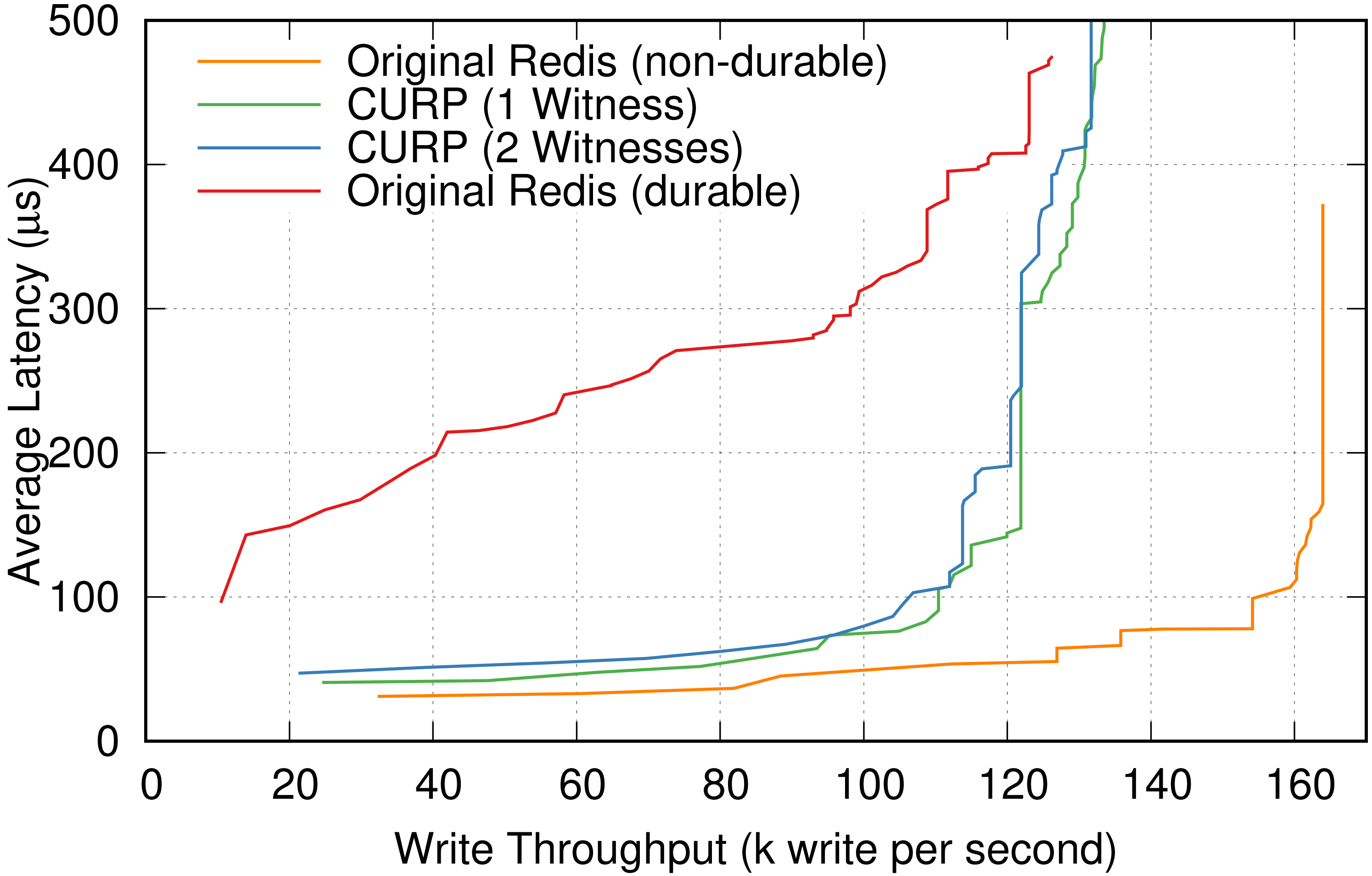
RAMCloud's Throughput after CGAR

- **Batching replication improved throughput**



Making Redis Consistent with Small Cost

- **Multiple clients sequentially issue SET operations**



Conclusion

- **Totally Ordering client requests are not requirement for consistent replication**
  - Potential of concurrency
- **CURP clients replicate without ordering in parallel with sending requests to execution servers**
- **CURP is applicable to**
  - Primary-backup replication
  - Consensus protocols with strong leader (e.g. Raft, Multi-Paxos, Viewstamped Replication, etc)
- **Improves latency and throughput**