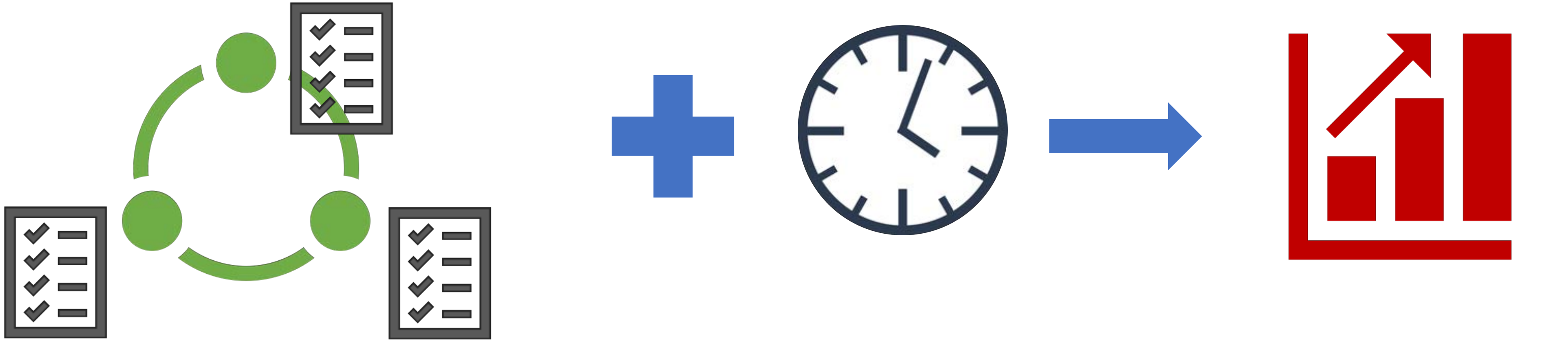
# CRaft: Using Accurate Clocks to Build a Multi-Leader Version of Raft

Feiran Wang*, Balaji Prabhakar*, Mendel Rosenblum*, Gene Zhang†
*Stanford University, †eBay Inc.

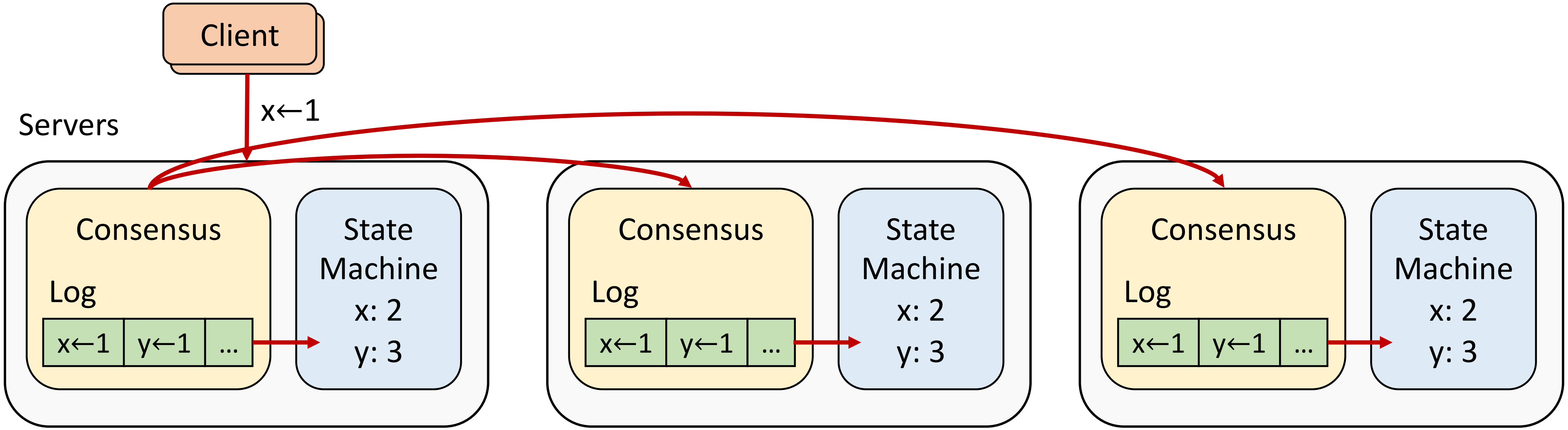
PLATFORMLAB

# Overview

- Limitation with leader-based consensus protocols
- CRaft: a multi-leader extension to Raft enabled by accurate clocks
  - 2-2.5x throughput improvement



# Replicated State Machines

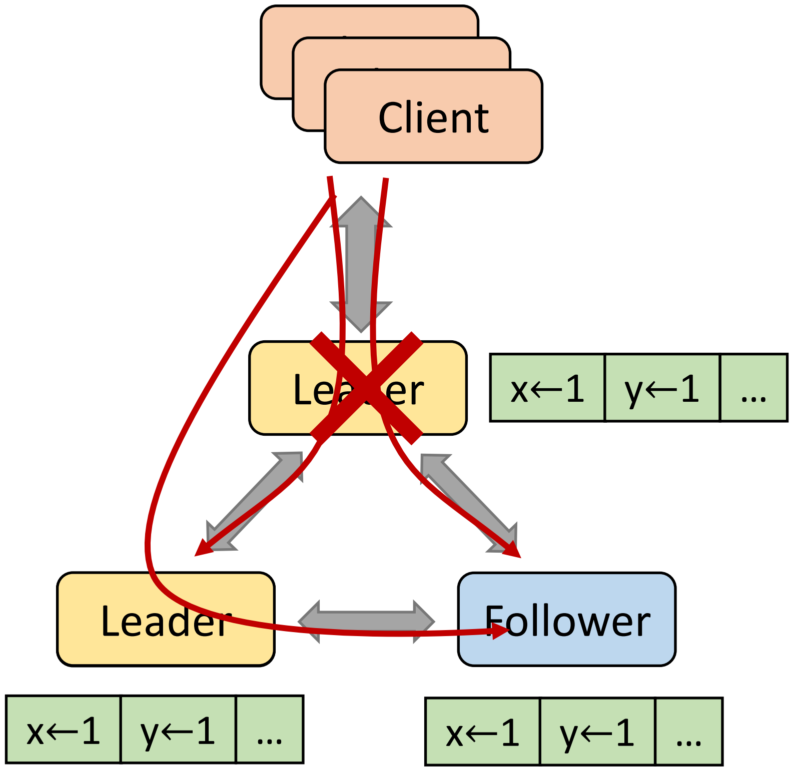



- State machines execute same commands in same order
- Consensus: ensures all servers agree on the same log
- Tolerate f failures with 2f + 1 replicas
- Failure model: fail-stop

Diego Ongaro and John Ousterhout.
The Raft consensus algorithm. https://raft.github.io

# The Raft Consensus Protocol

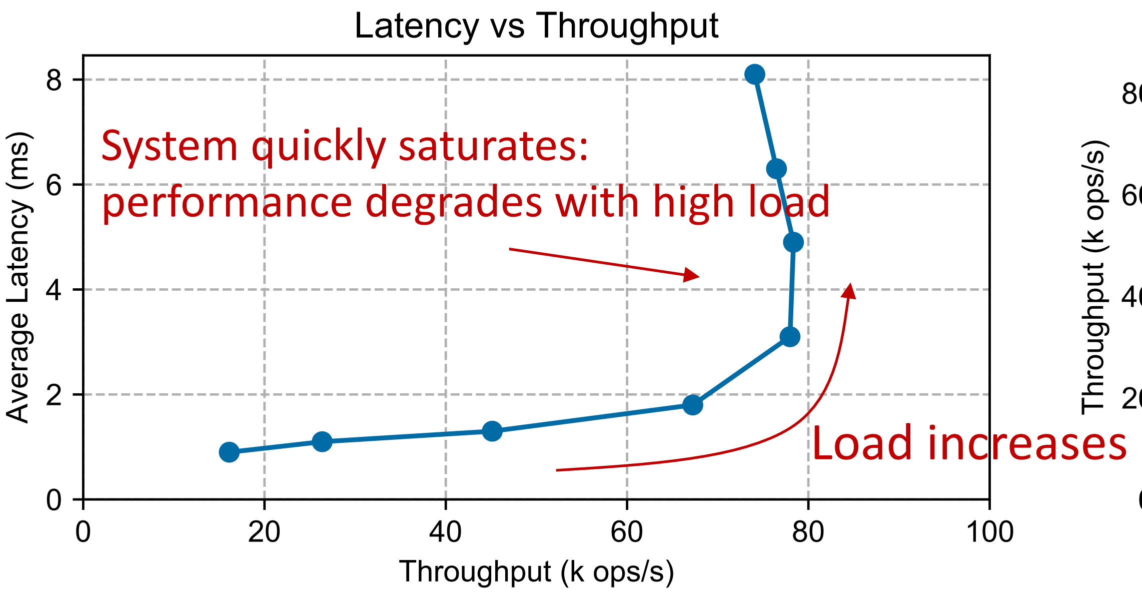
- A widely used consensus protocol
- Leader-based: one server selected as leader
  - Leader accepts commands from clients and replicate to other servers
- Benefits: simple and efficient
- Limitation: leader is the bottleneck for throughput and scalability

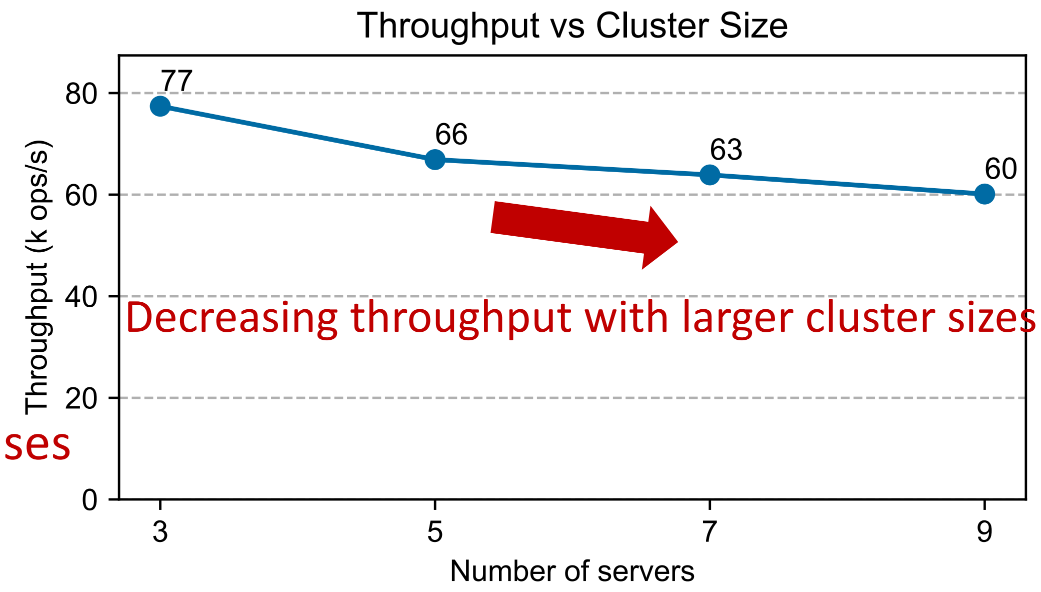



Diego Ongaro and John Ousterhout. 2014. In search of an understandable consensus algorithm. In USENIX Annual Technical Conference. 305–319.

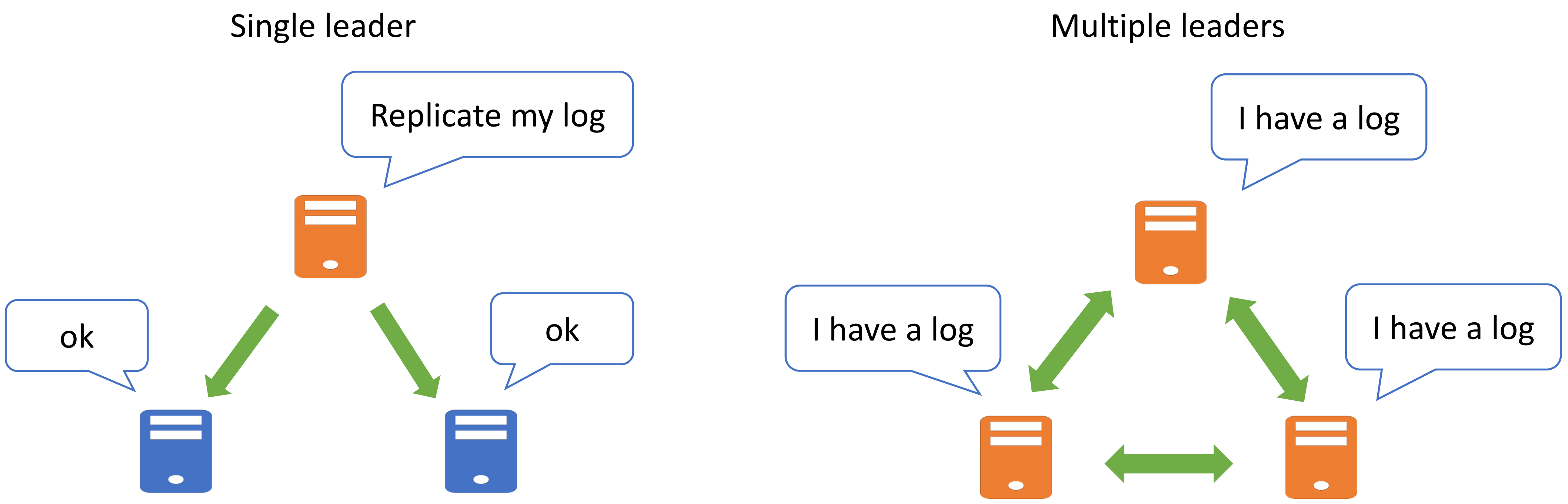
# Limitations with Single Leader

- Single leader limits throughput and scalability
- Measurements taken from HashiCorp Raft in CloudLab, using a key-value store



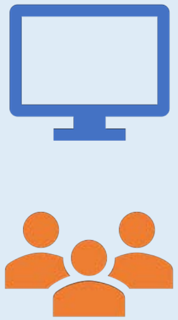



# Challenge in a Multi-Leader Protocol

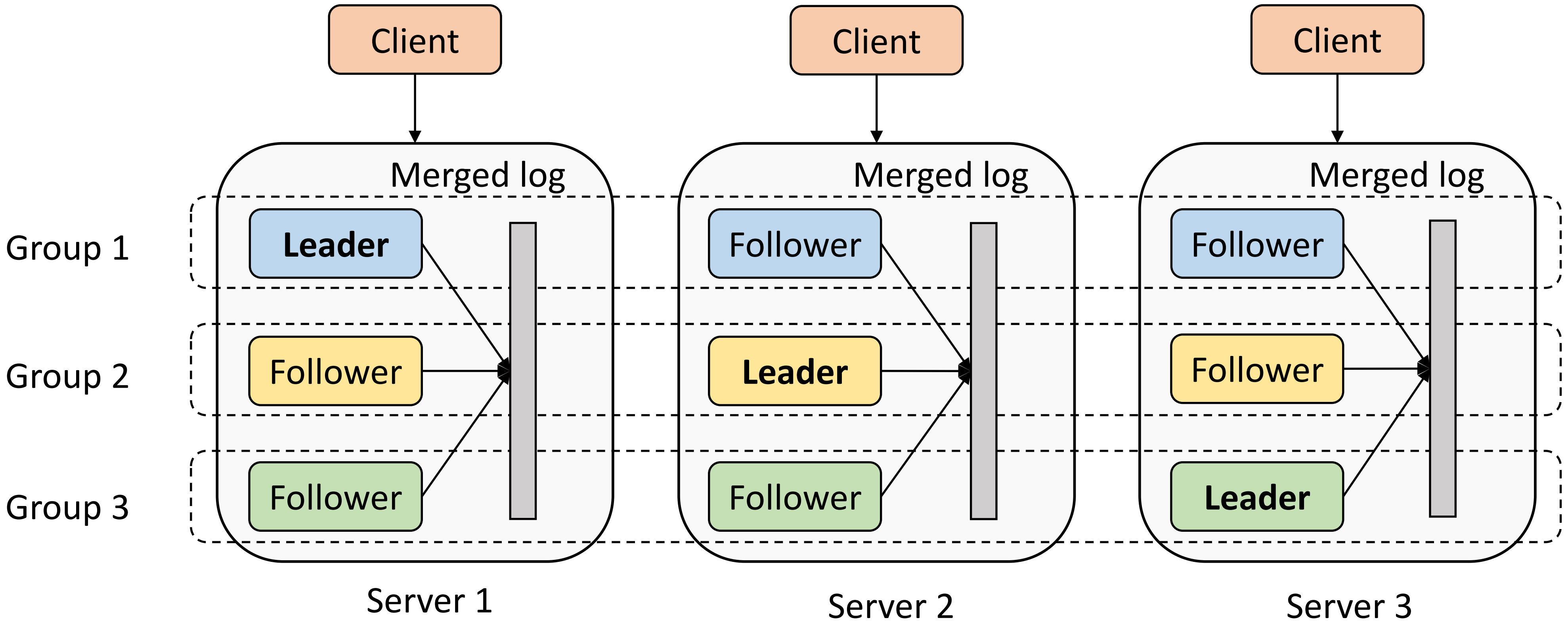



- Challenge: how to coordinate leaders?
- Solution: ordering with accurate clocks

# Our Approach: CRaft

| | Raft | CRaft (Clocks + Raft) |
|---|---|---|
| Scalability |  |  |

# CRaft Overview




- Run multiple concurrent Raft groups
- Merge logs entries across groups using timestamps from synchronized clocks

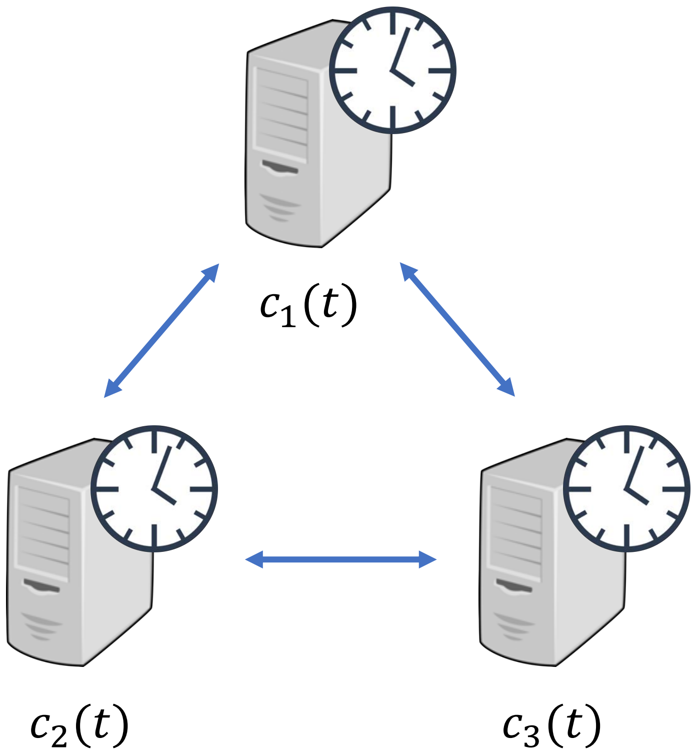
# Clock Synchronization

# Clock Synchronization

- $c_i(t)$ – system clock of machine *i* at real time *t*
- Clock synchronization:

$$\forall t, \forall i, j: |c_i(t) - c_j(t)| \leq \epsilon$$

- $\epsilon$: precision




Lamport, Leslie. Time, clocks, and the ordering of events in a distributed system. Communications of the ACM 21.7 (1978): 558-565.

# Huygens Clock Synchronization

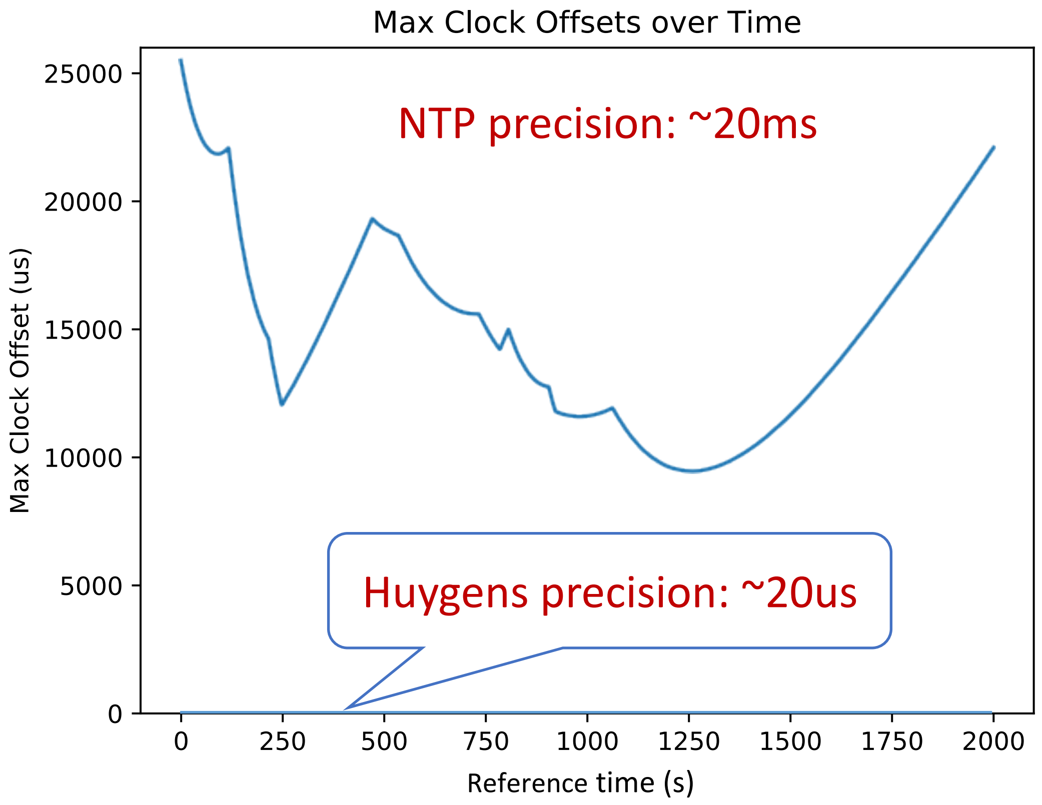
- Huygens: a software clock synchronization system
- Measurements taken from CloudLab, 20 machines in a data center (with software timestamping)

Distribution of clock offsets between servers

| Percentile | 90th | 99th | 99.9th | max |
|---|---|---|---|---|
| Clock offset | 7us | 11us | 15us | 26us |

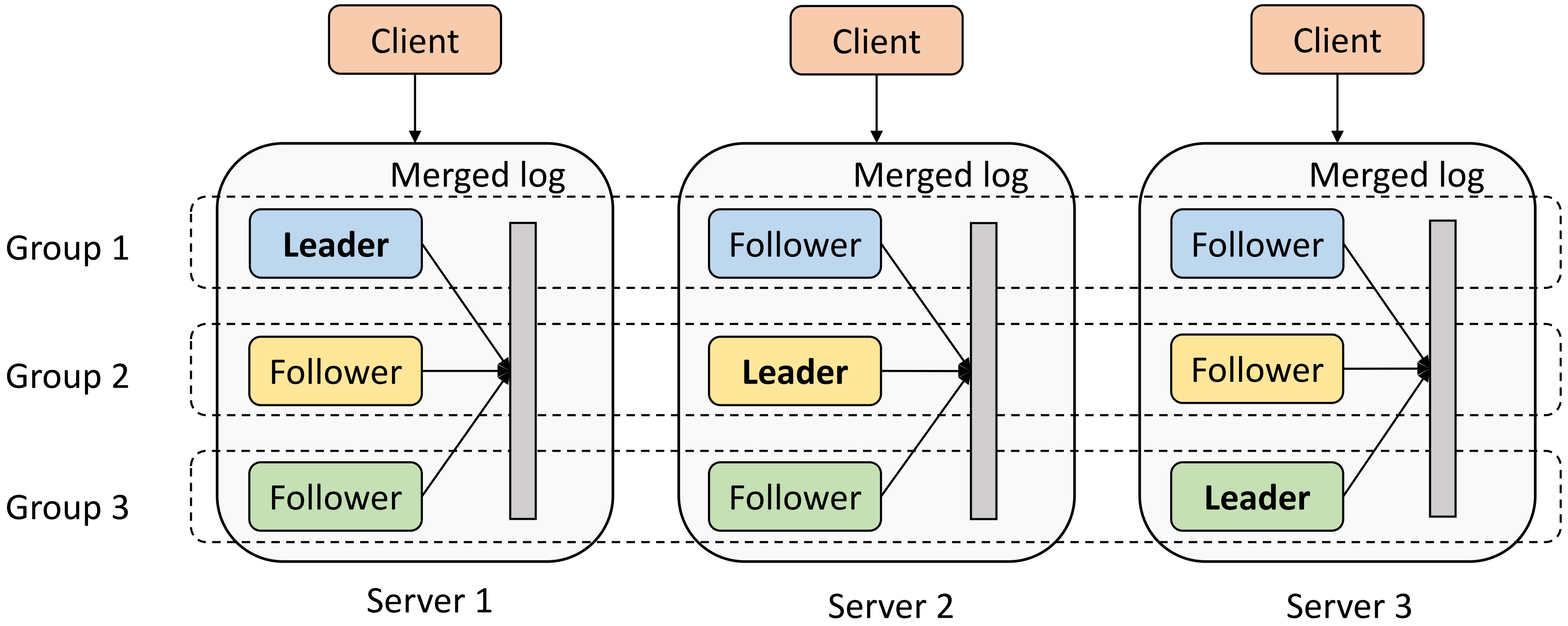
Yilong Geng, Shiyu Liu, Zi Yin, Ashish Naik, Balaji Prabhakar, Mendel Rosenblum, and Amin Vahdat. Exploiting a natural network effect for scalable, fine-grained clock synchronization. In NSDI 2018. 81–94.

# Network Time Protocol (NTP) vs Huygens



# The CRaft Consensus Protocol

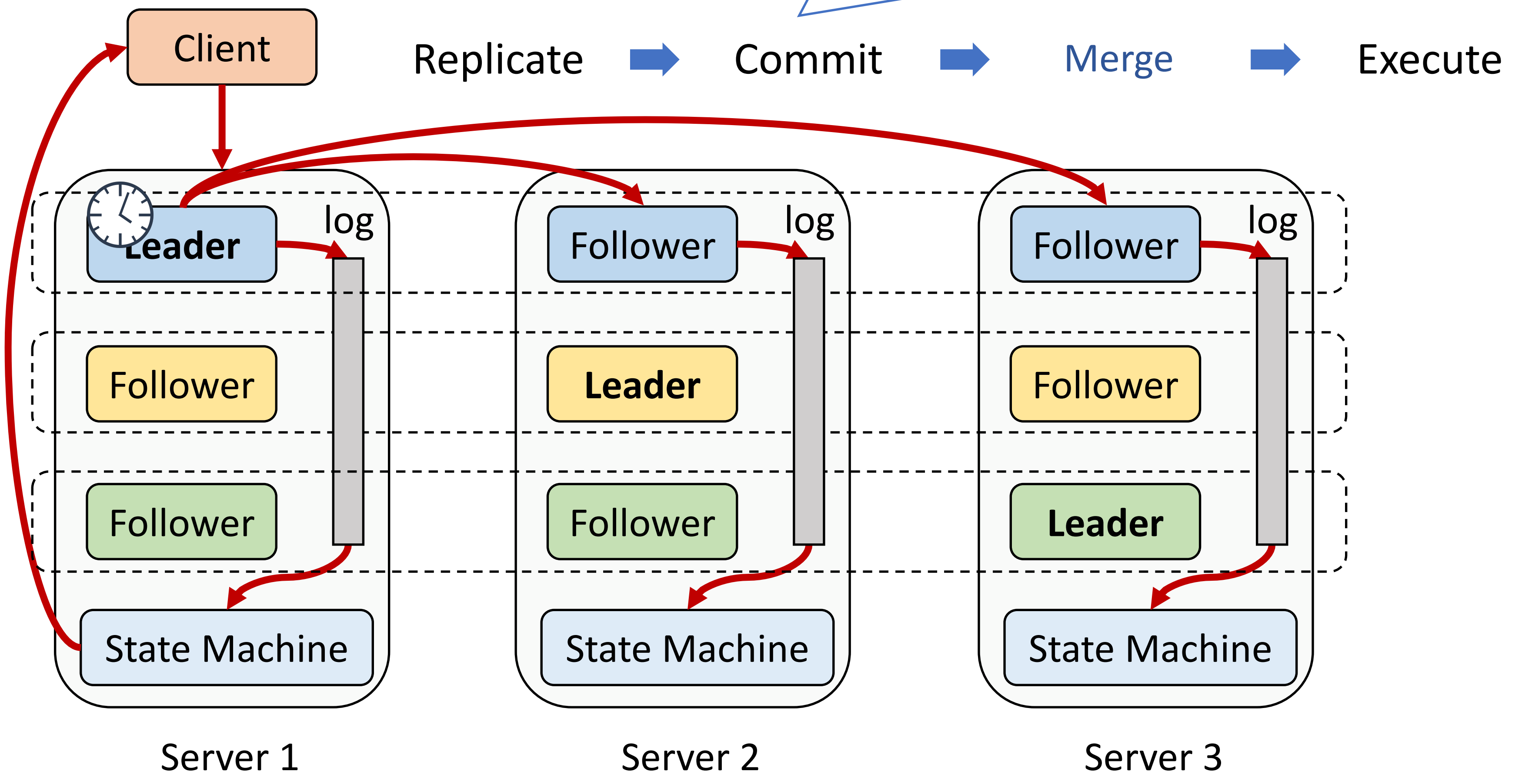
# CRaft Overview



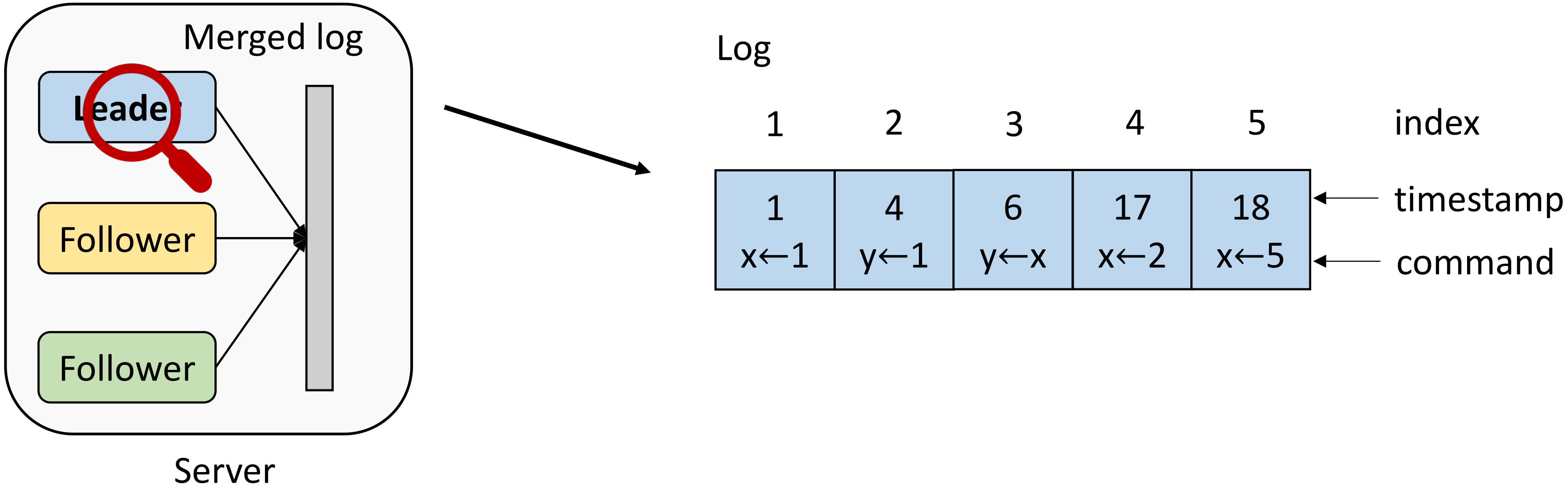

- Run multiple concurrent Raft groups
- Merge logs entries across groups using timestamps from synchronized clocks

# Life of a Request

- Replicated on a majority of servers
- Safe and durable

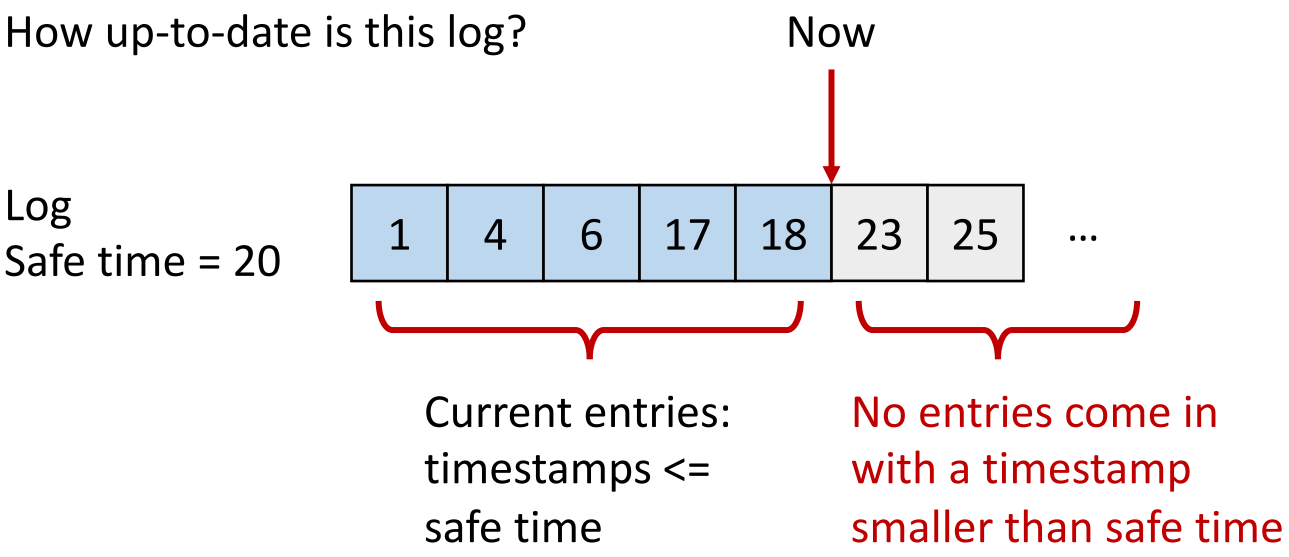


# Entry Timestamps



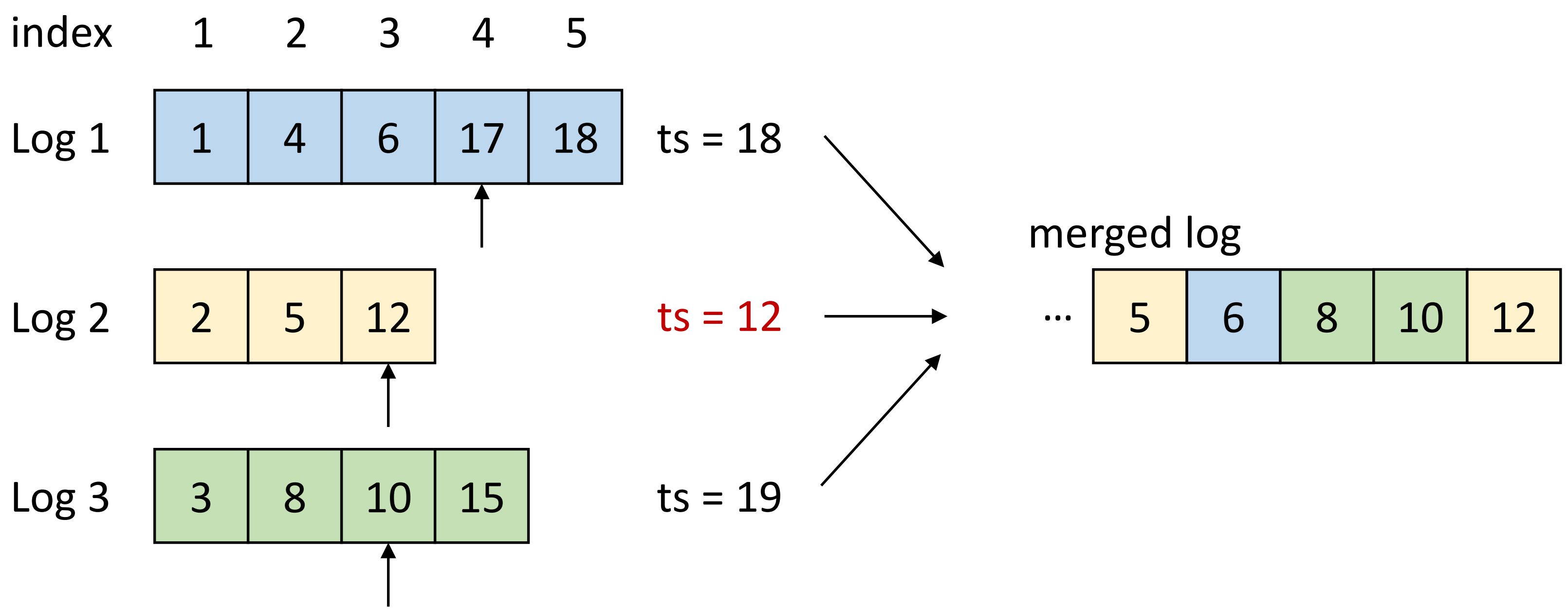

- Each entry has a timestamp assigned by its leader
- CRaft guarantees monotonically increasing timestamps in each log

# Safe Times




- Safe to process log entries up to its safe time
- Safe times are updated when replicating entries

# Merging

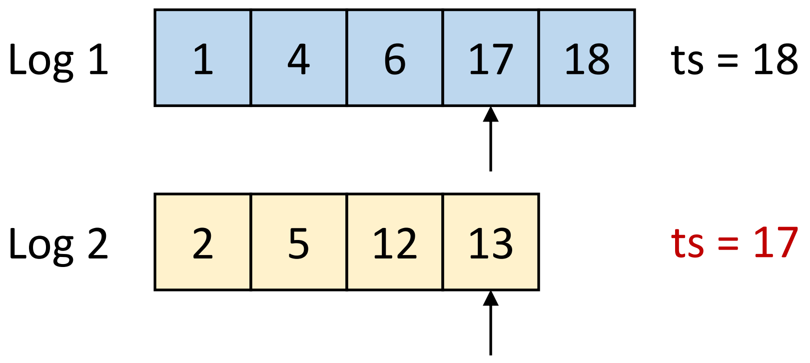



- Log entries are merged on each server, in increasing timestamp order
- Can merge up to the smallest safe time
- CRaft ensures merged log in monotonically increasing timestamp order

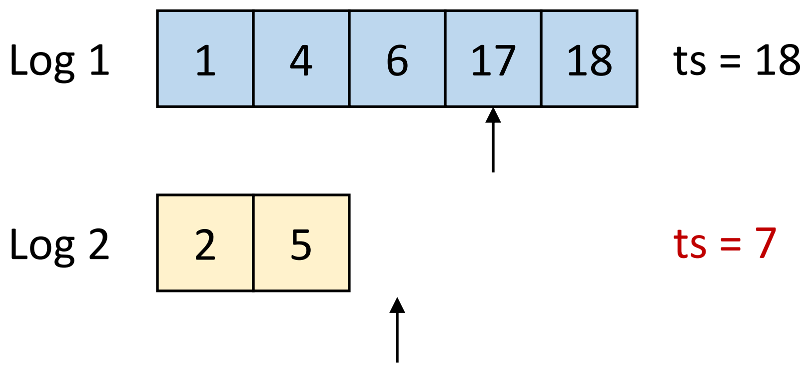
# Maintaining Safe Times

- Goals: keeping safe times correct and up-to-date

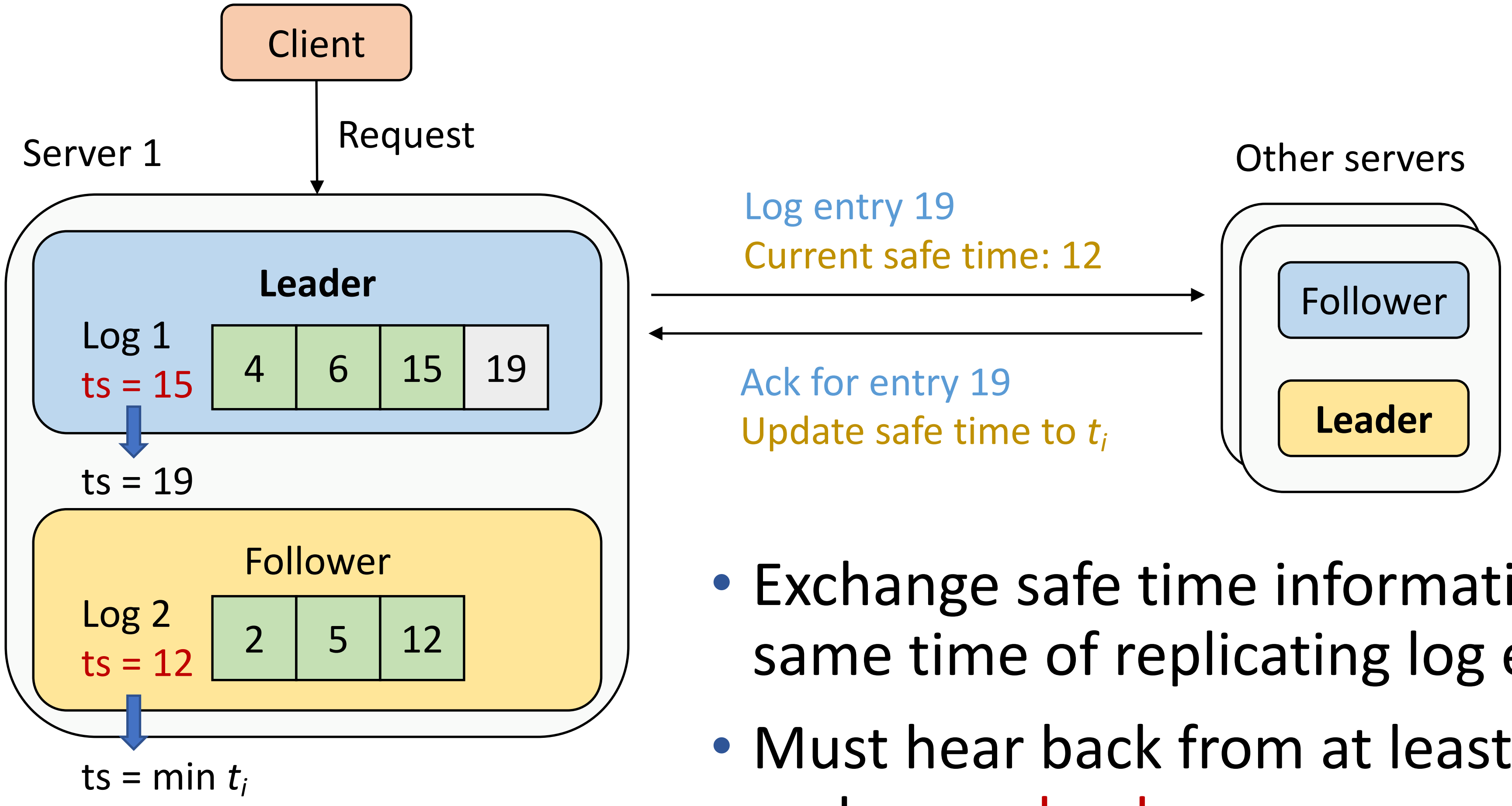
Incorrect safe times cause inconsistencies




Small safe times block merging



# Maintaining Safe Times




- Exchange safe time information at the same time of replicating log entries
- Must hear back from at least a majority and group leaders

# Optimization: Fast Path

Replicate → Commit → Merge → Execute

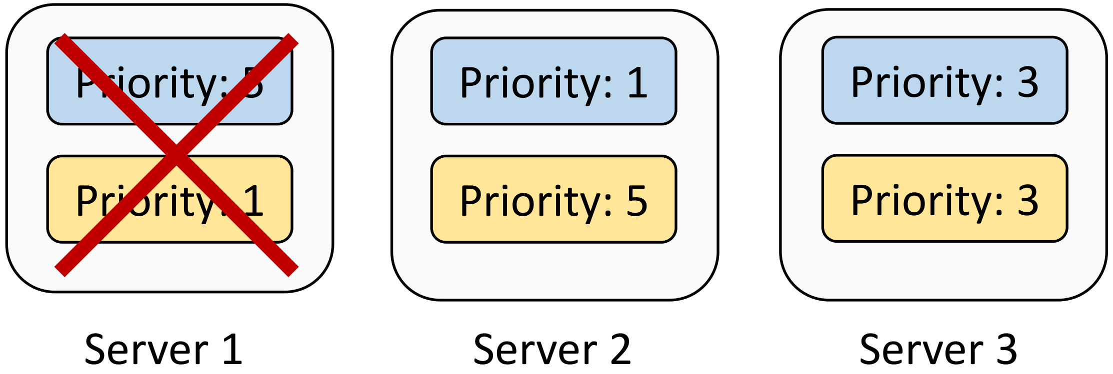
Fast path: respond before execution

Normal path: respond after execution

- For a write operation, confirmation is sufficient, as long as later reads can see the write result

# Leader Election Policy

- Each group performs leader election independently
- Elect leaders according to priorities – distribute leaders to different servers
- Leaders can be auto re-balanced



# Protocol Guarantees

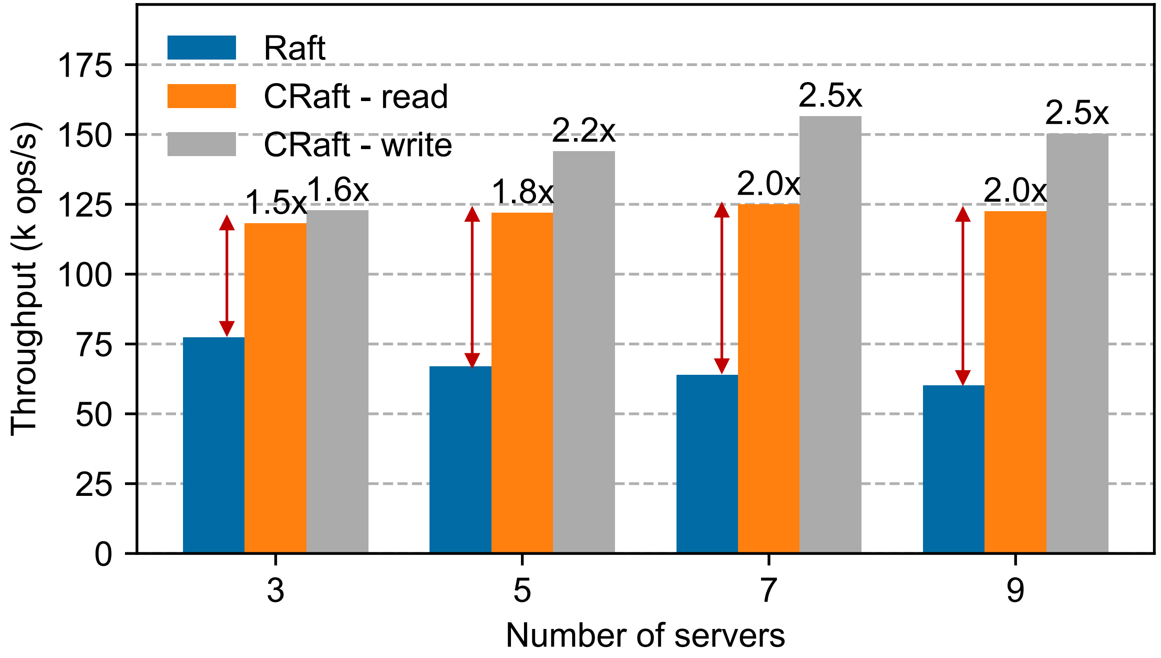
- State machine safety: all servers will apply the same set of log entries to their state machines, in the same order.
- Linearizability: if operation A is responded before B starts, A will be executed before B
  - Example: a read following a write will return the newly written value

# Evaluation

# Experiment Setup

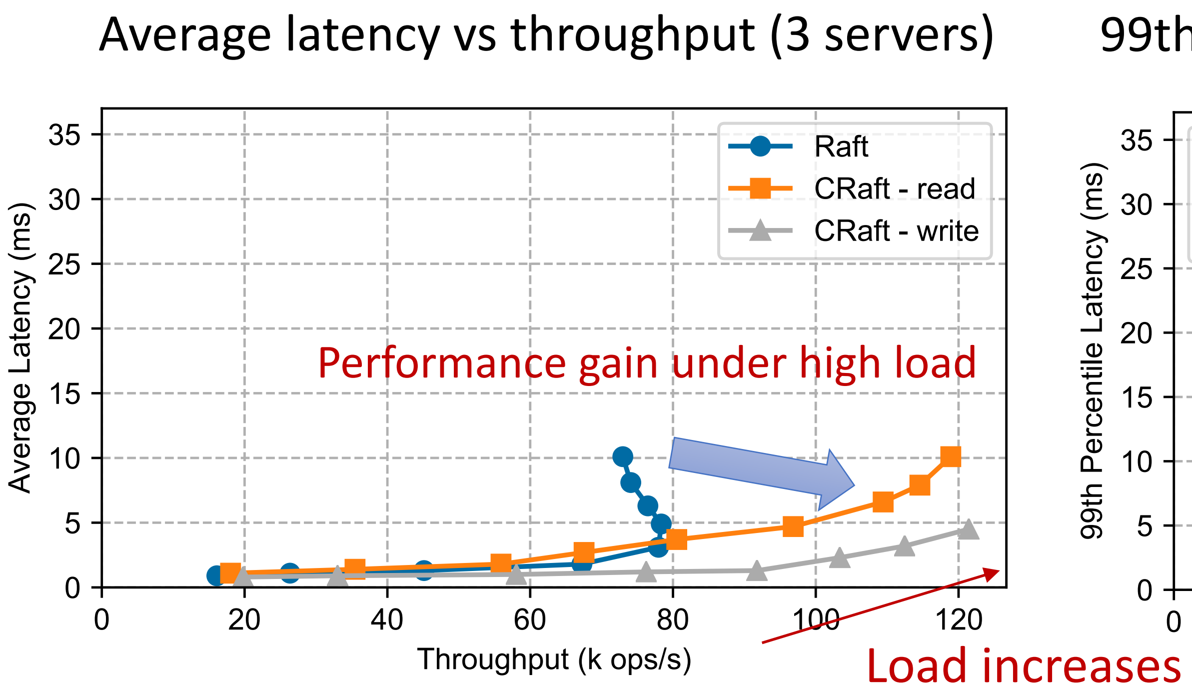
- Implementation
  - Based on HashiCorp Raft – a popular and well-optimized implementation
- Environment
  - CloudLab, single data center
- Workload
  - In-memory key-value store
  - Multiple clients send get or set requests concurrently

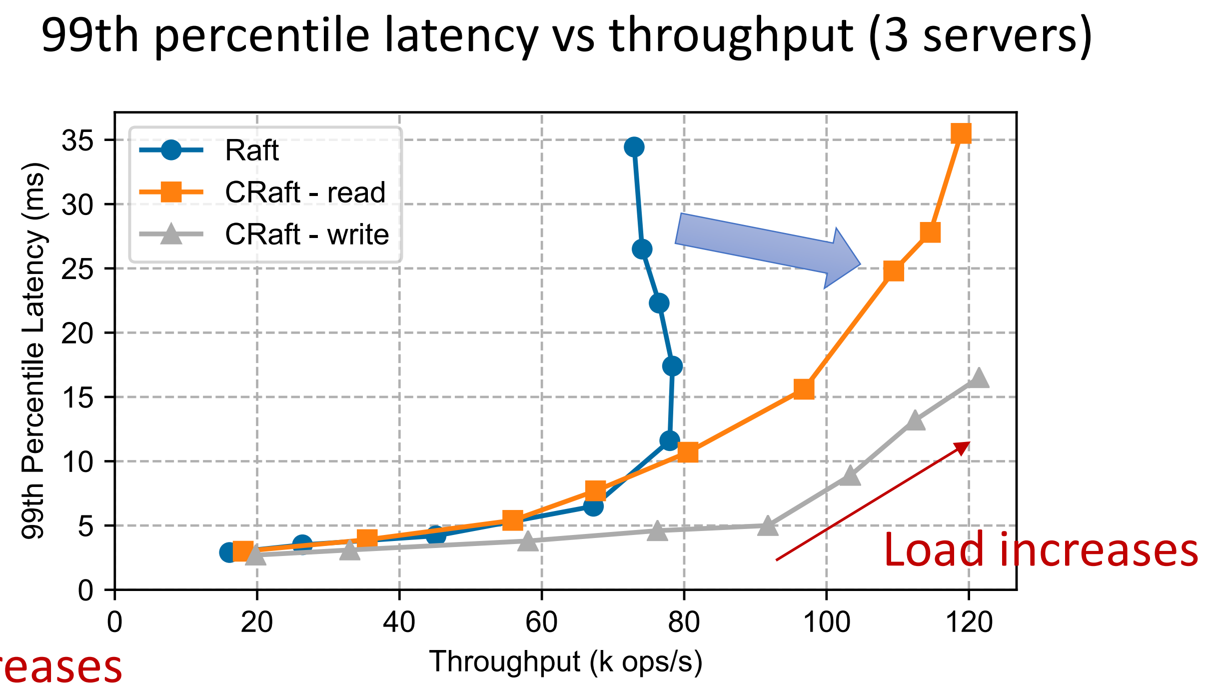
# Throughput vs Cluster Size




- Up to ~2x read and ~2.5x write throughput compared to Raft

# Latency vs Throughput



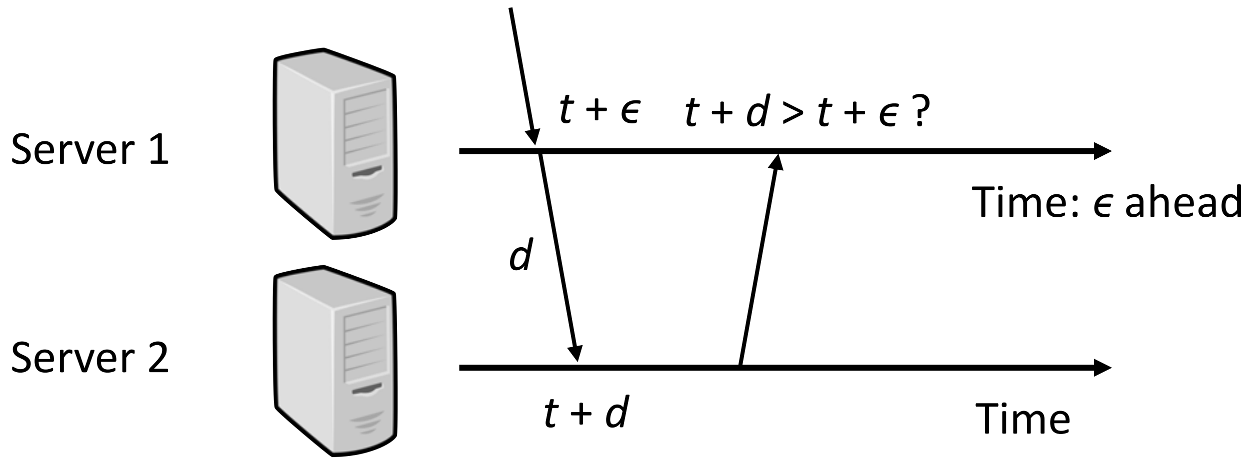

- CRaft improves throughput and latency under high load

# Timing Requirement

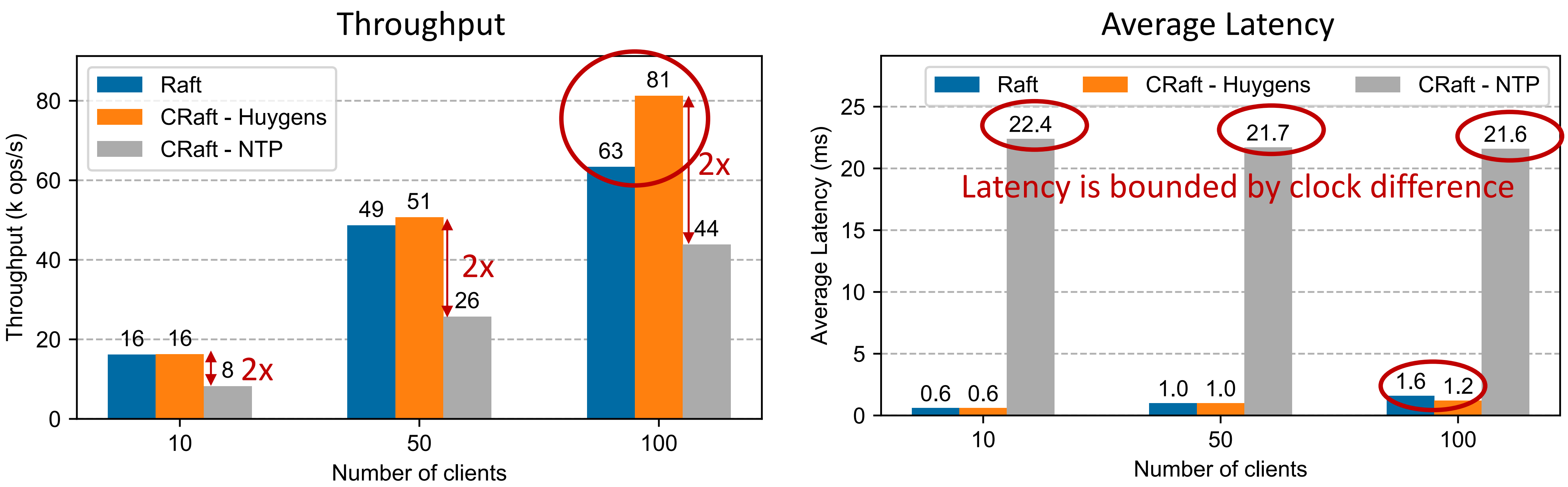
- Timing requirement for high performance:

$$\epsilon < \text{one-way message delay}$$

- Intuition from a server's perspective: to execute a command, the other servers' times need to be larger than mine

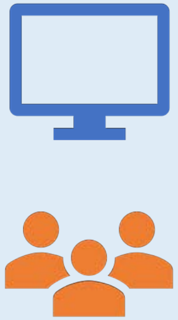


# Performance vs Number of Clients




- NTP precision: ~20ms, Huygens: ~20us

# Conclusion: CRaft

| | **Raft** | **CRaft (Clocks + Raft)** |
|---|---|---|
| Scalability |  |  |
| Output | A replicate log | A replicated log |
| Safety & Consistency | ✓ | ✓ Same guarantee as Raft |
| Practicability | ✓ | ✓ A simple add-on to Raft; easy to implement |