Raft Protocol

Prof. Christof Fetzer, Ph.D.
TU Dresden

Motivation

“Shared nothing distributed architecture”

Motivation

- Minimize access latency
  - ensure that we exact copy of all data in several data centers
  - clients are at most 25 ms round-trip time to next data center
    - i.e., can read data within 25ms
- **Problem**:
  - **update of data**: we need to replicate across all data centers

Synchronous Replication

- **Geo-Replication:**
  - round-trip time between data centers could be more than 65 ms
- **Paxos**:
  - we would need >= 90 ms for updates:
    - 45ms for **prepare** message
    - 45ms for **accept** message
- **Question**:
  - would single round-trip, i.e., 45 ms be sufficient?

Synchronous Replication

- **Client triggers update**
  - connect to closest data center for updates?
- **Round-Trip Delay**
  - 25 ms client to data center US-West
  - 45 ms client to US-Central
- **Questions**:
  - what if we have single round trip consensus from one data center?
    - 45ms + 45ms = 90ms?
    - 25ms +45ms + 45ms = 115ms?
    - **ideally**: 25ms+45ms = 70ms?

Raft

“In Search of an Understandable Consensus Algorithm”

Practical Fault Tolerance

- Service fails (e.g., because of a program crash):
  - automatic restart on same host (e.g., using systemd)
- A **coordination service / orchestrator** helps to
  - detect if a service is unavailable (e.g., host is unavailable), and
  - to restart on same (host is available) or separate host (if unavailable)
- **Notes**:
  - we only need one replica of a **stateless service** (for fault tolerance)!
  - coordination service needs to be fault-tolerant itself

Consensus Use Case

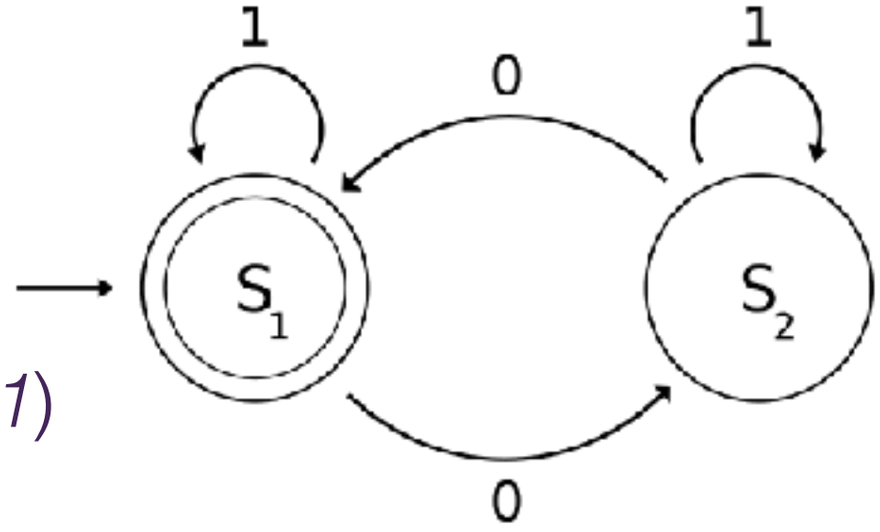
- **Coordination services:**
  - **Raft**: consensus in Consul coordination service
  - **Paxos**: used by Chubby (see SE2)
  - **Zab:** for ZooKeeper

- **Fault-tolerant services**
  - maintain state across multiple replicas
  - depending on the type of failures and service (stateless vs stateful), we need
    - **1**, **F+1**, **2F+1** or **3F+1** replicas
  - to tolerate **F** failures

State Machine Replication
vs
Primary/Backup

State Machine

- state machine:
  - processes sequence of requests (*input*)
  - initial state: start state (S1)
  - modifies internal *state*
    - transitions between states
  - produces reply (*output, e.g., 0 or 1*)



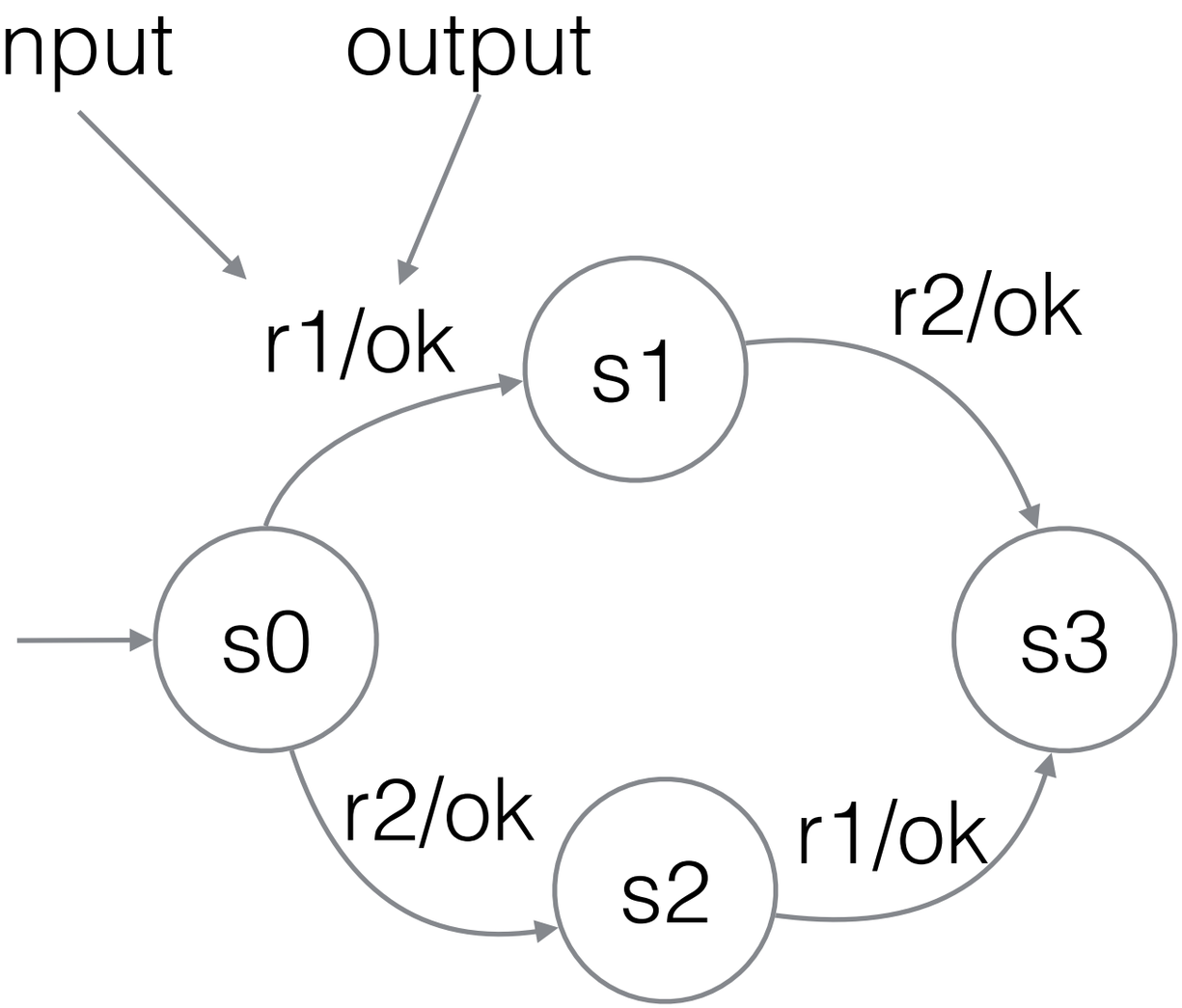
- deterministic
  - same input sequence leads to same modifications and replies

State Machine Replication (SMR)

- SMR: used for stateful, fault-tolerant client-server systems
  - replication of servers
  - coordination of client interactions with replicas

- every replica executes the same set of **client requests**
  - execute requests in an **equivalent order**
    - to ensure that replicas produce the same outputs

Example



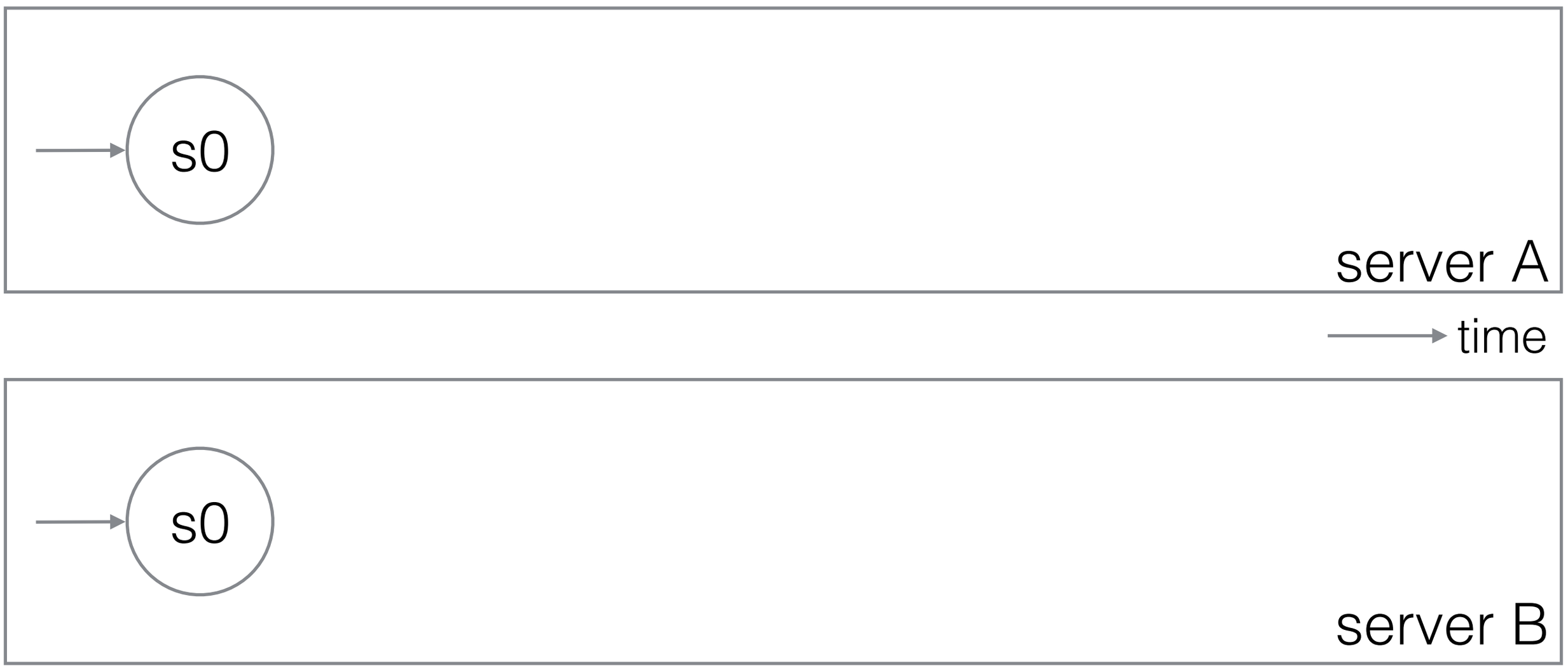
requests **r1** and **r2**
are commutative

states:
s0: v = 0
s1: v = 1
s2: v = 2
s3: v = 3

requests:
r1: „v += 1“; output **ok**;
r2: „v += 2“; output **ok**;

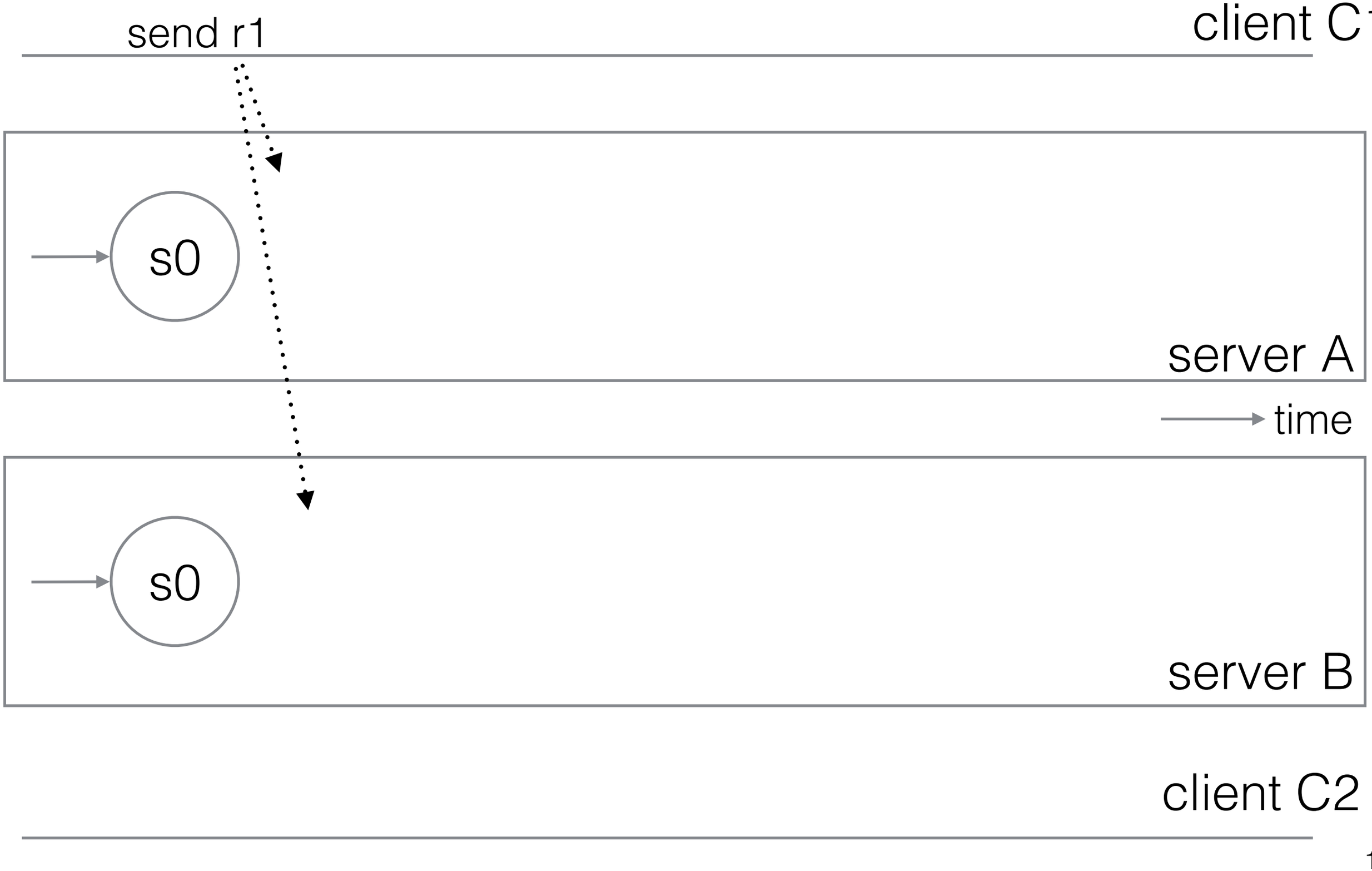
SMR: 2 Replicas

client C1

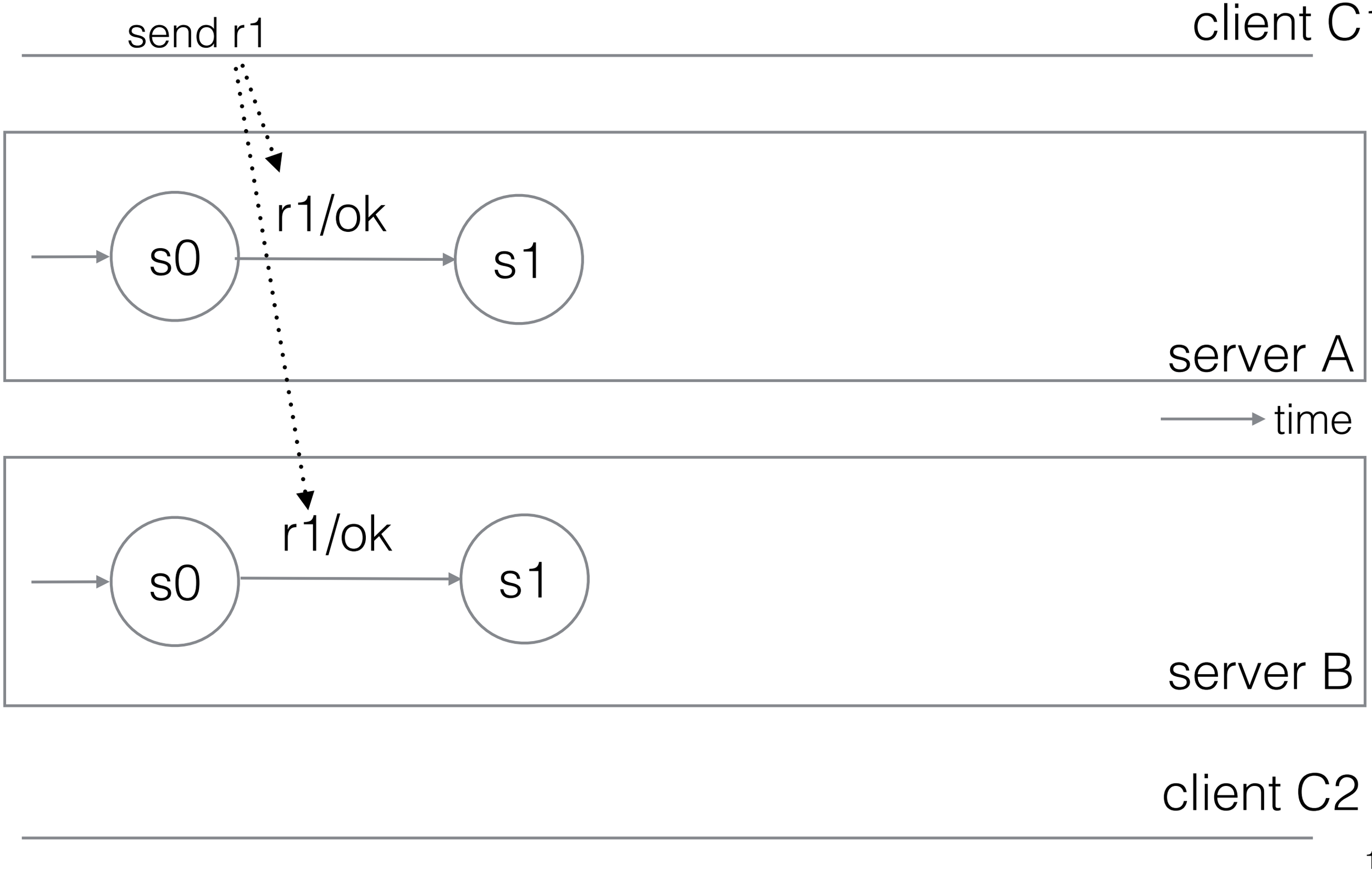


client C2

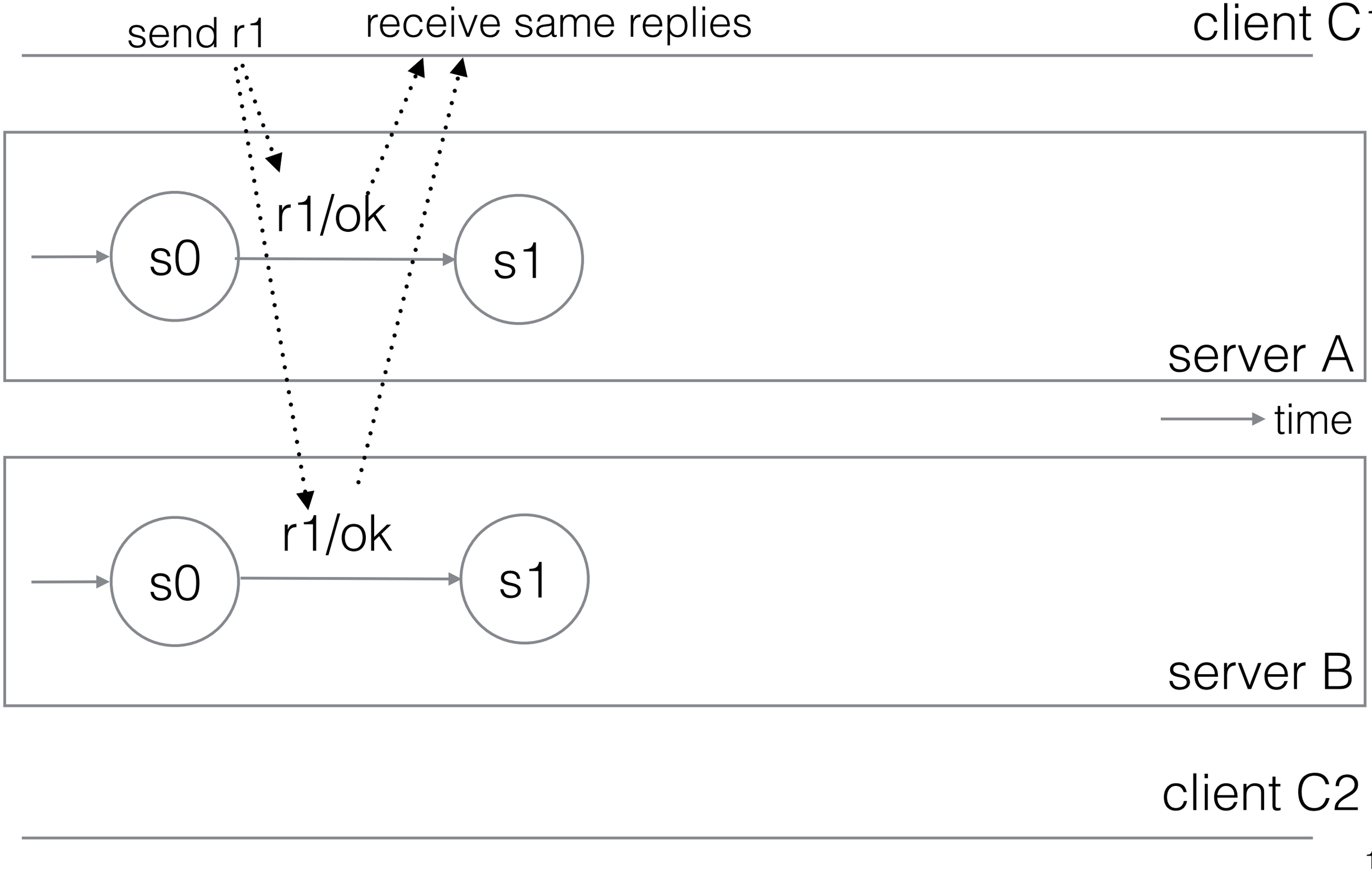
SMR: 2 Replicas



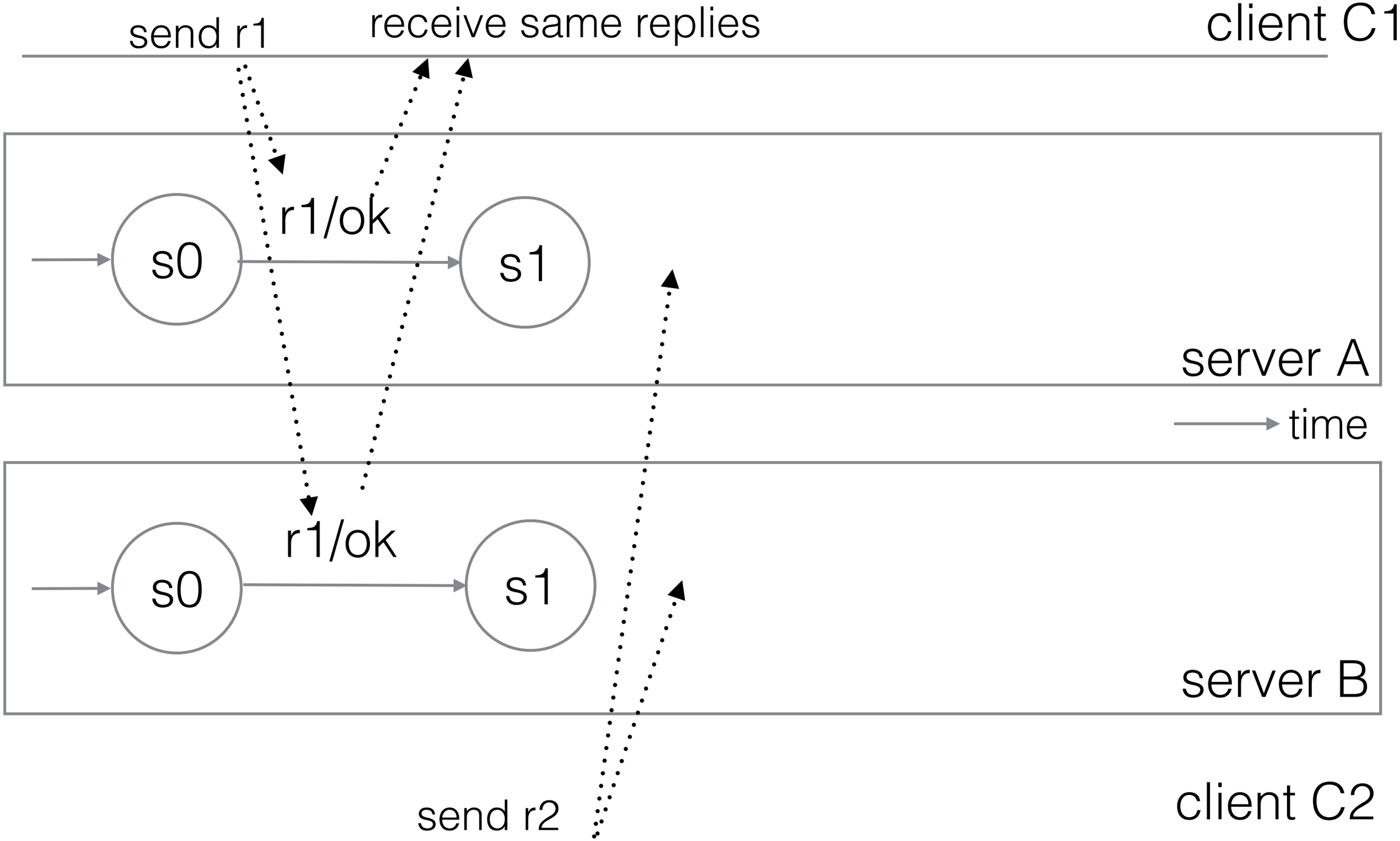
SMR: 2 Replicas



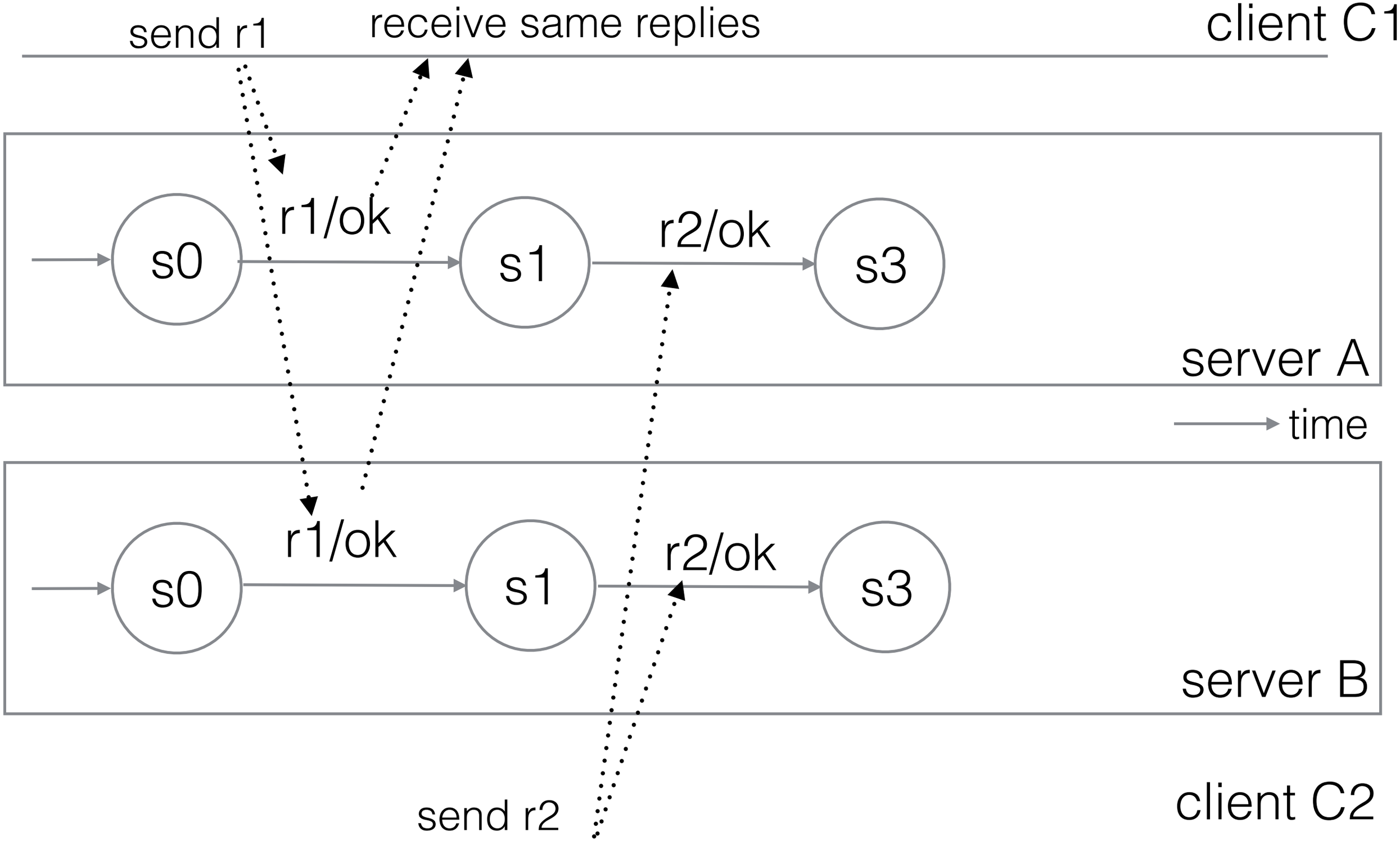
SMR: 2 Replicas



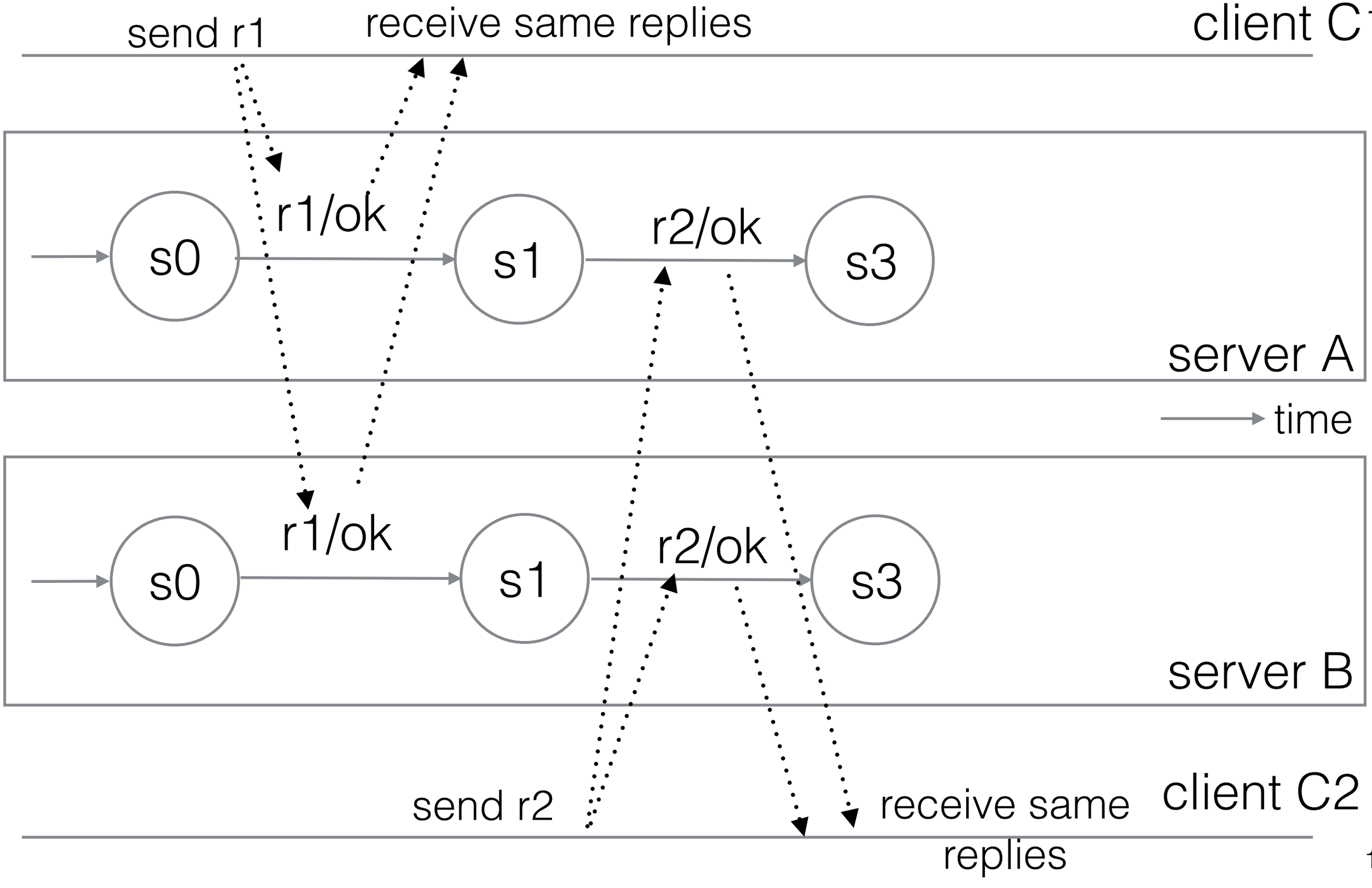
SMR: 2 Replicas



SMR: 2 Replicas

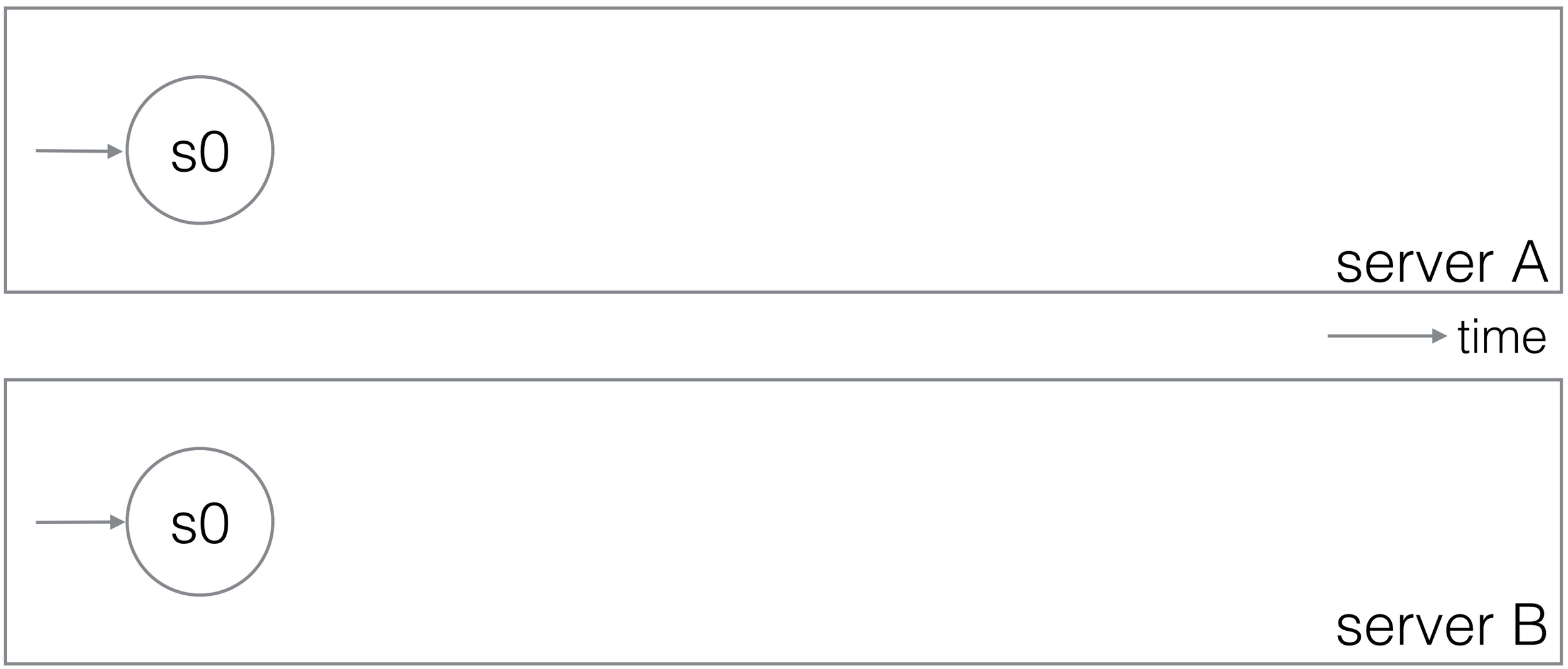


SMR: 2 Replicas



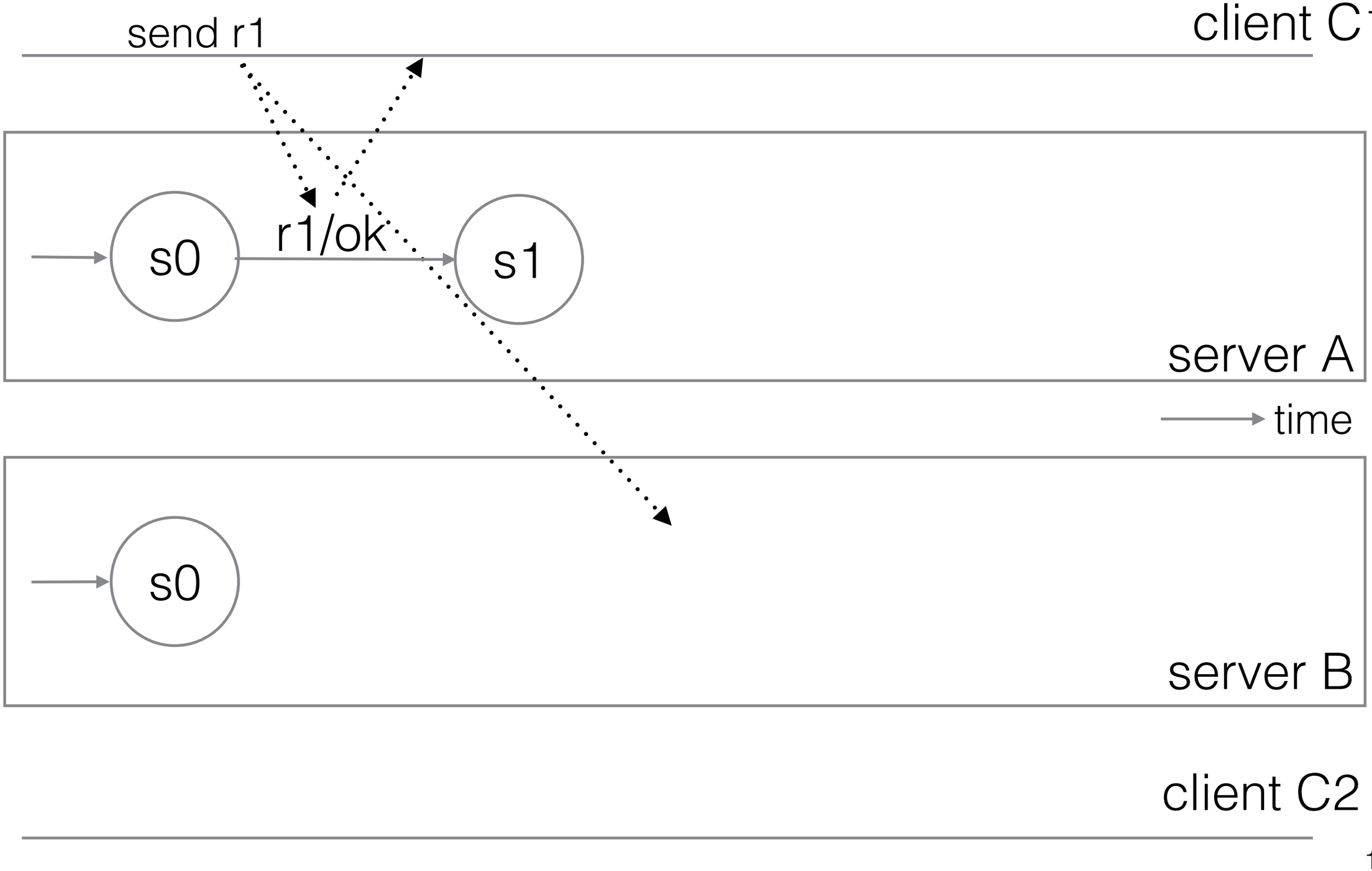
Out Of Order Execution

client C1

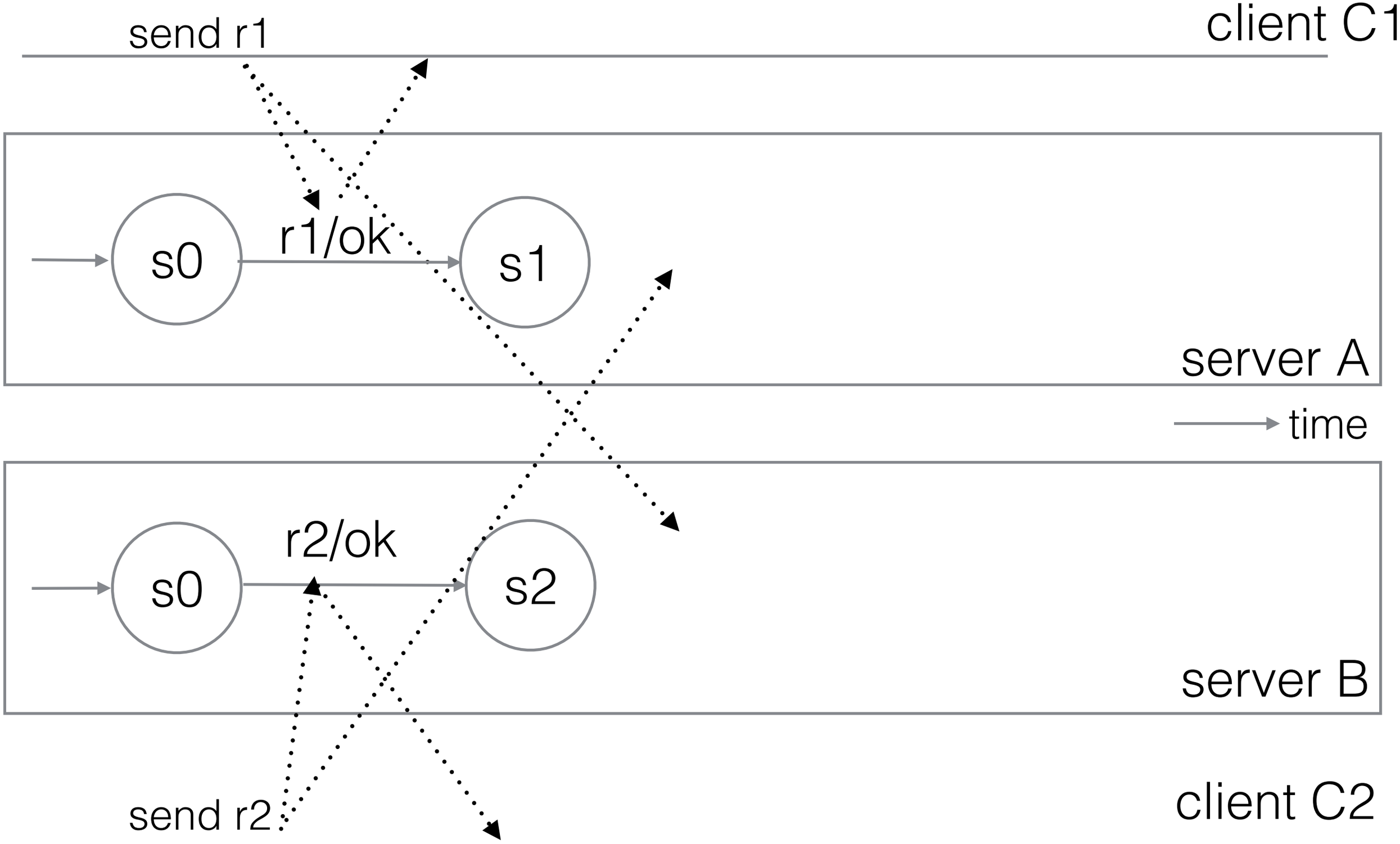


client C2

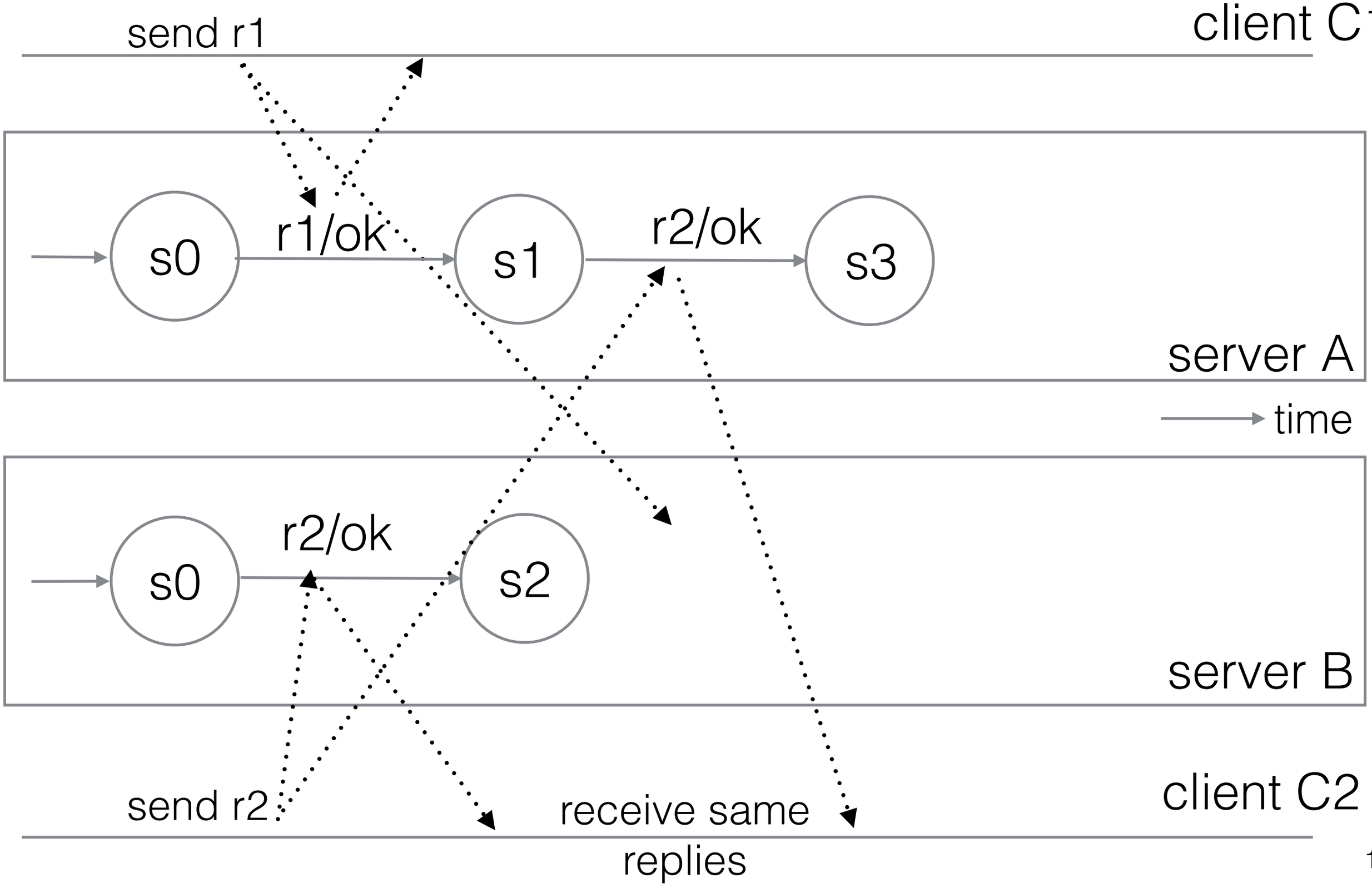
Out Of Order Execution



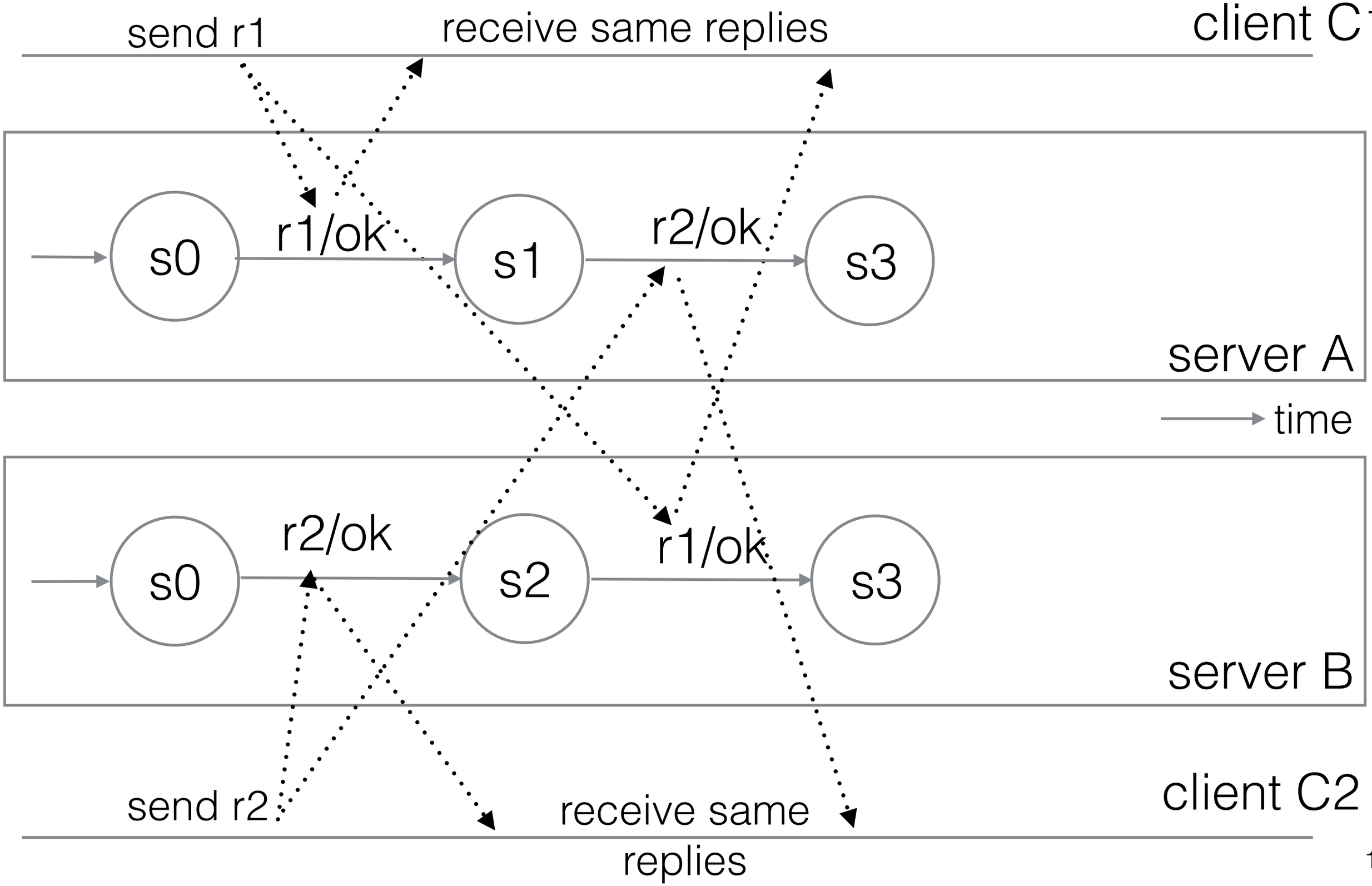
Out Of Order Execution



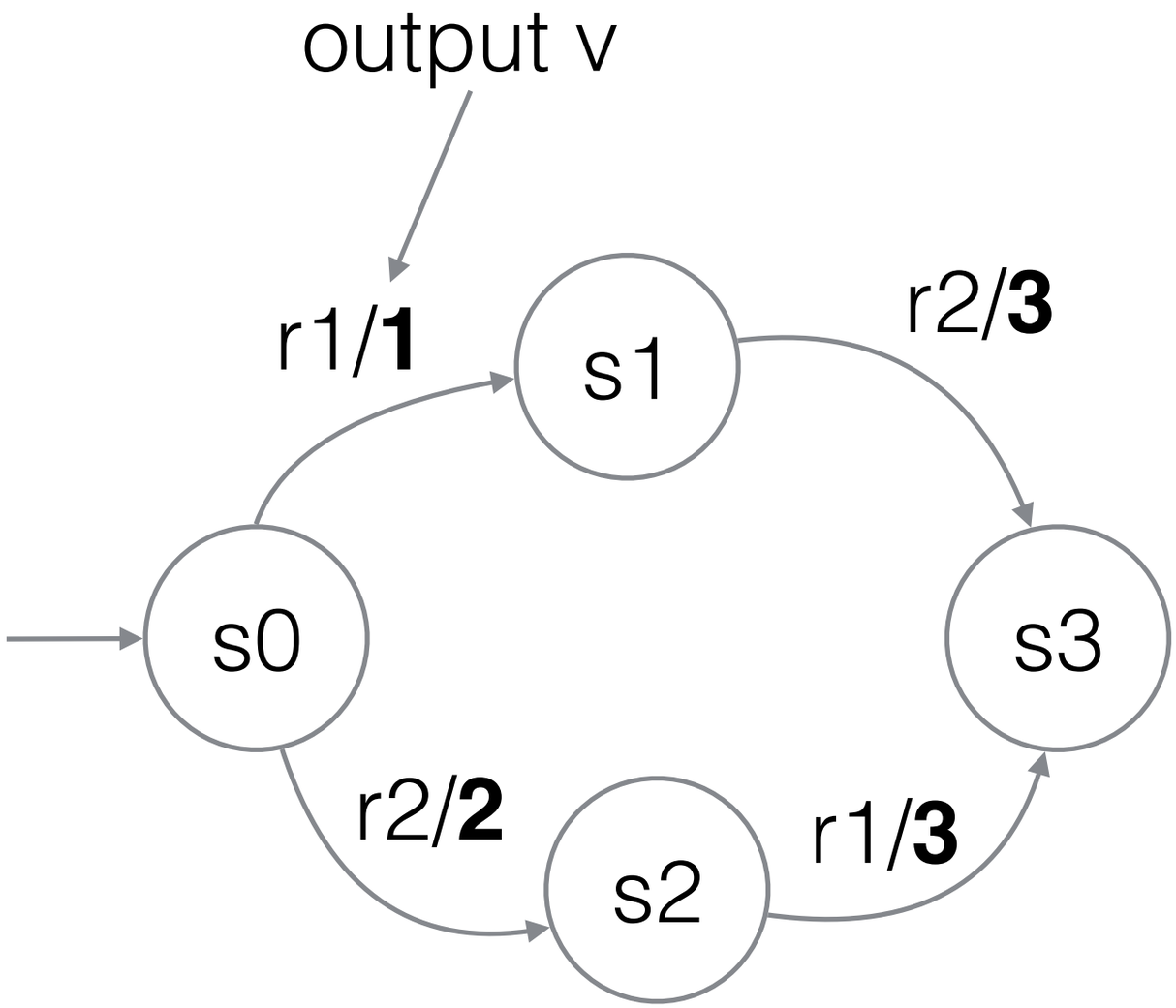
Out Of Order Execution



Out Of Order Execution



Output Value V



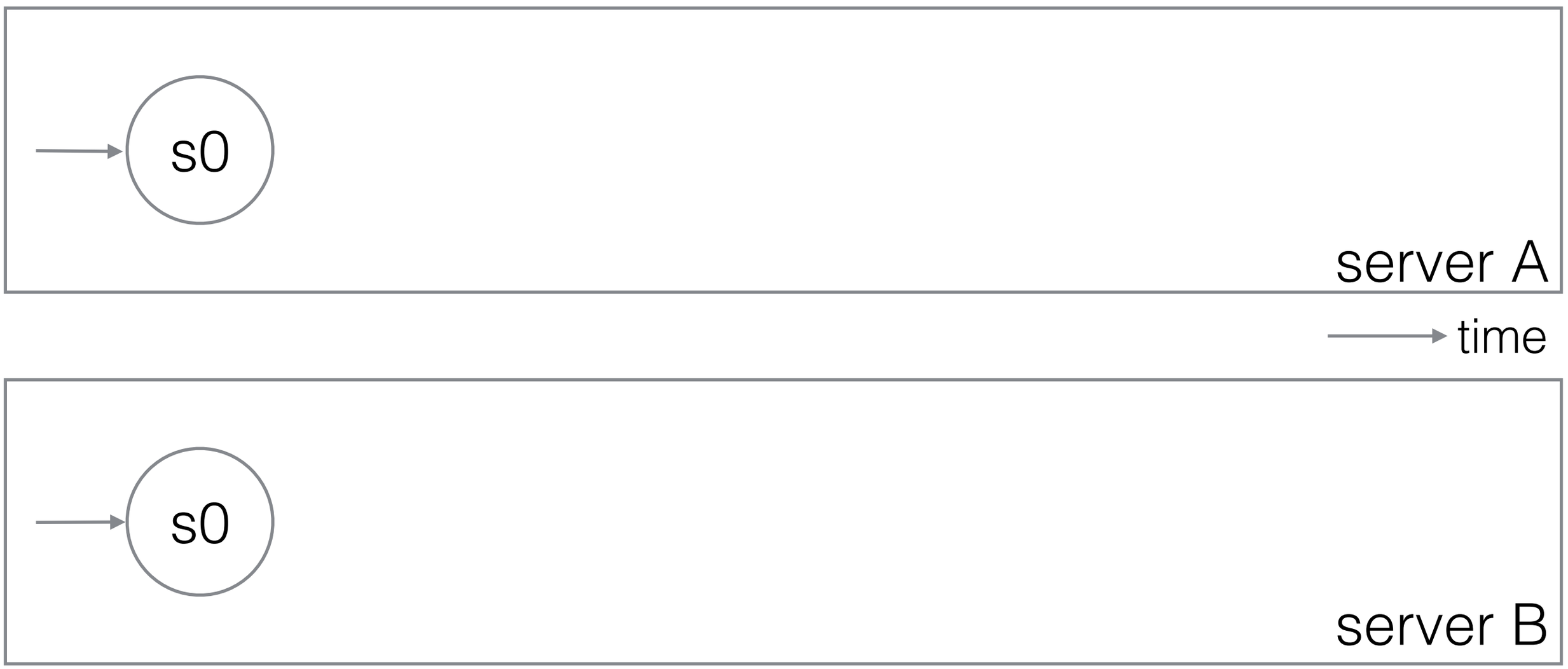
requests **r1** and **r2**
are not commutative
anymore!

states:
s0: v = 0
s1: v = 1
s2: v = 2
s3: v = 3

requests:
r1: „v += 1“; output **v**;
r2: „v += 2“; output **v**;

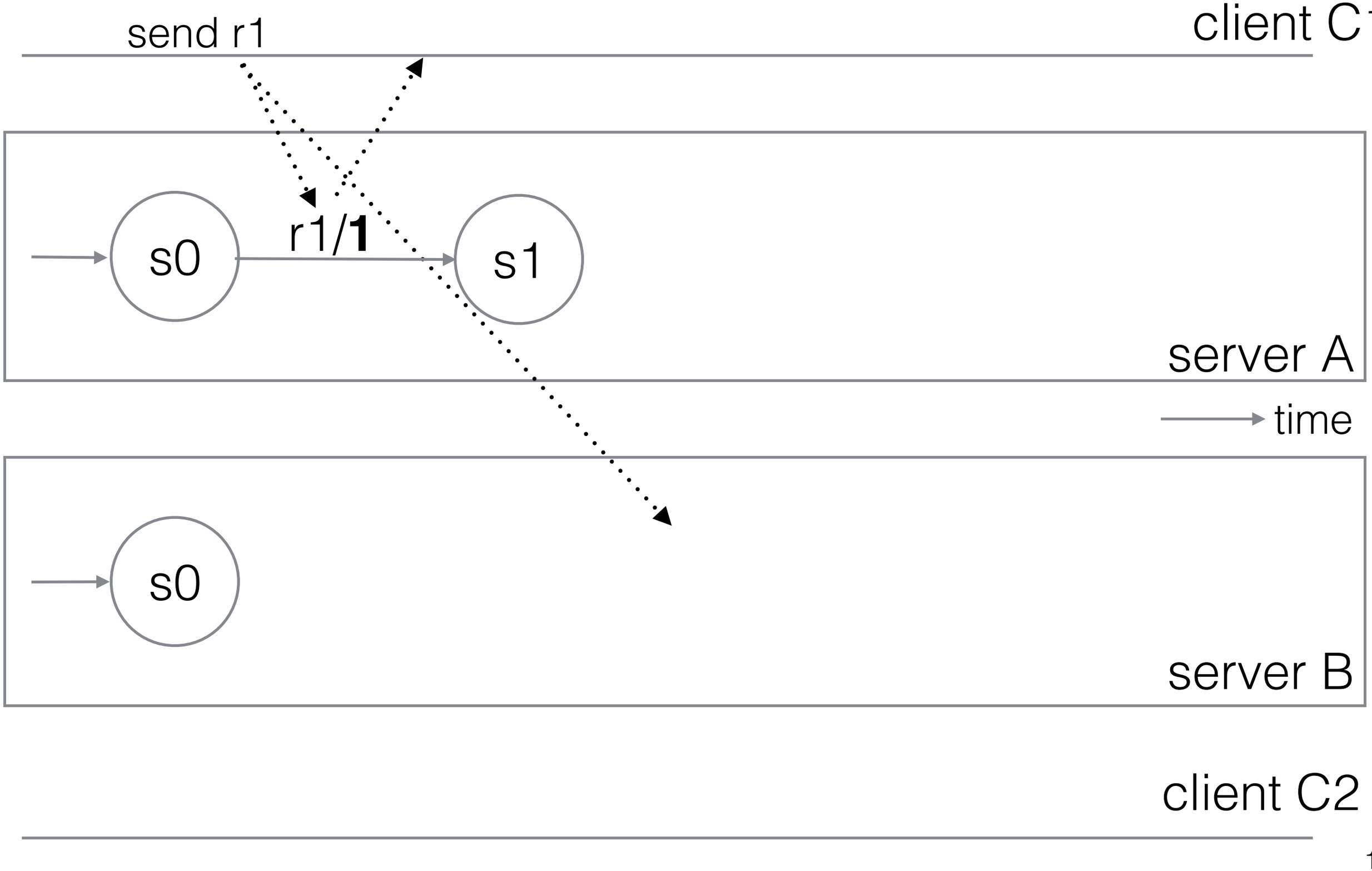
Out Of Order Execution

client C1

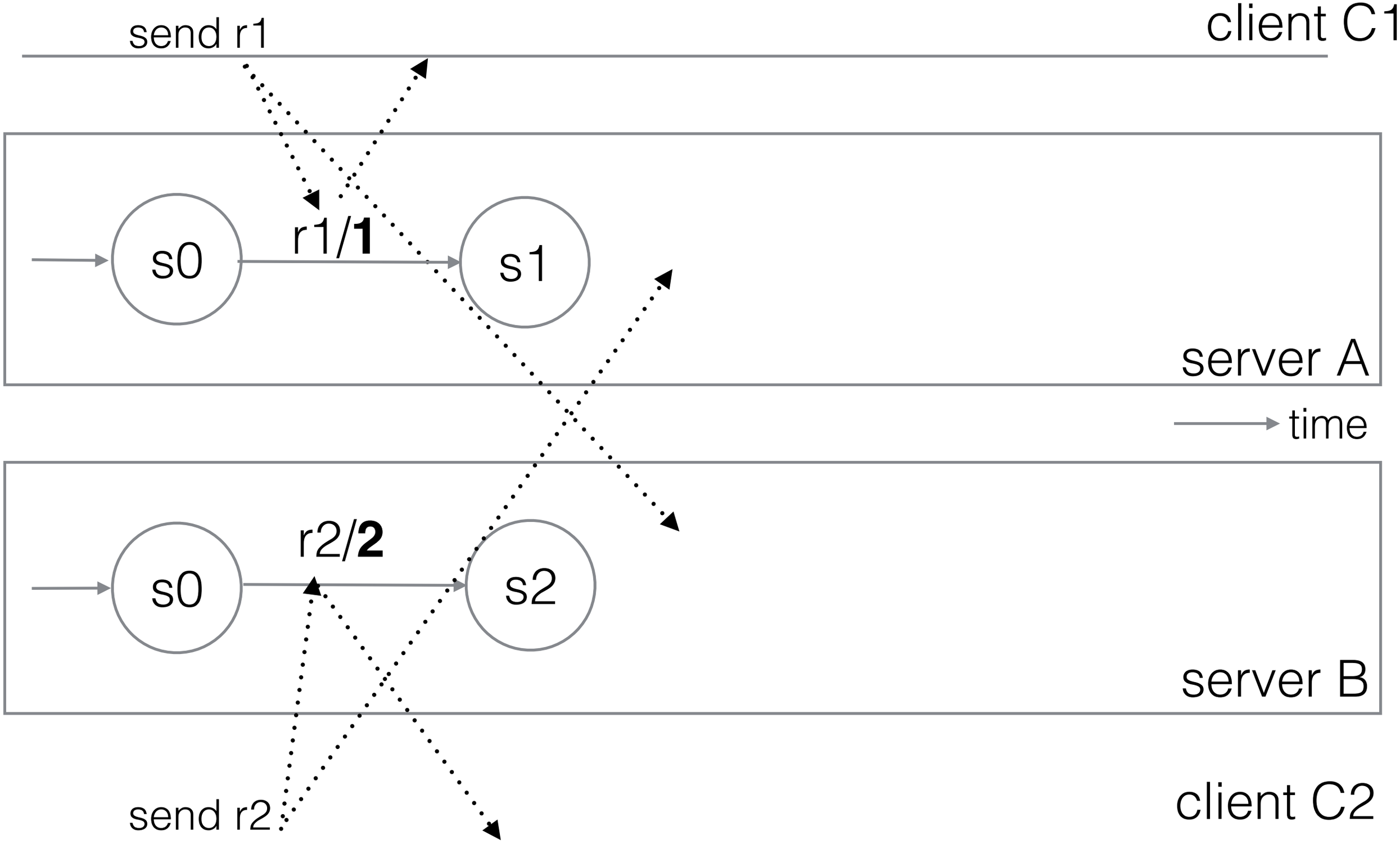


client C2

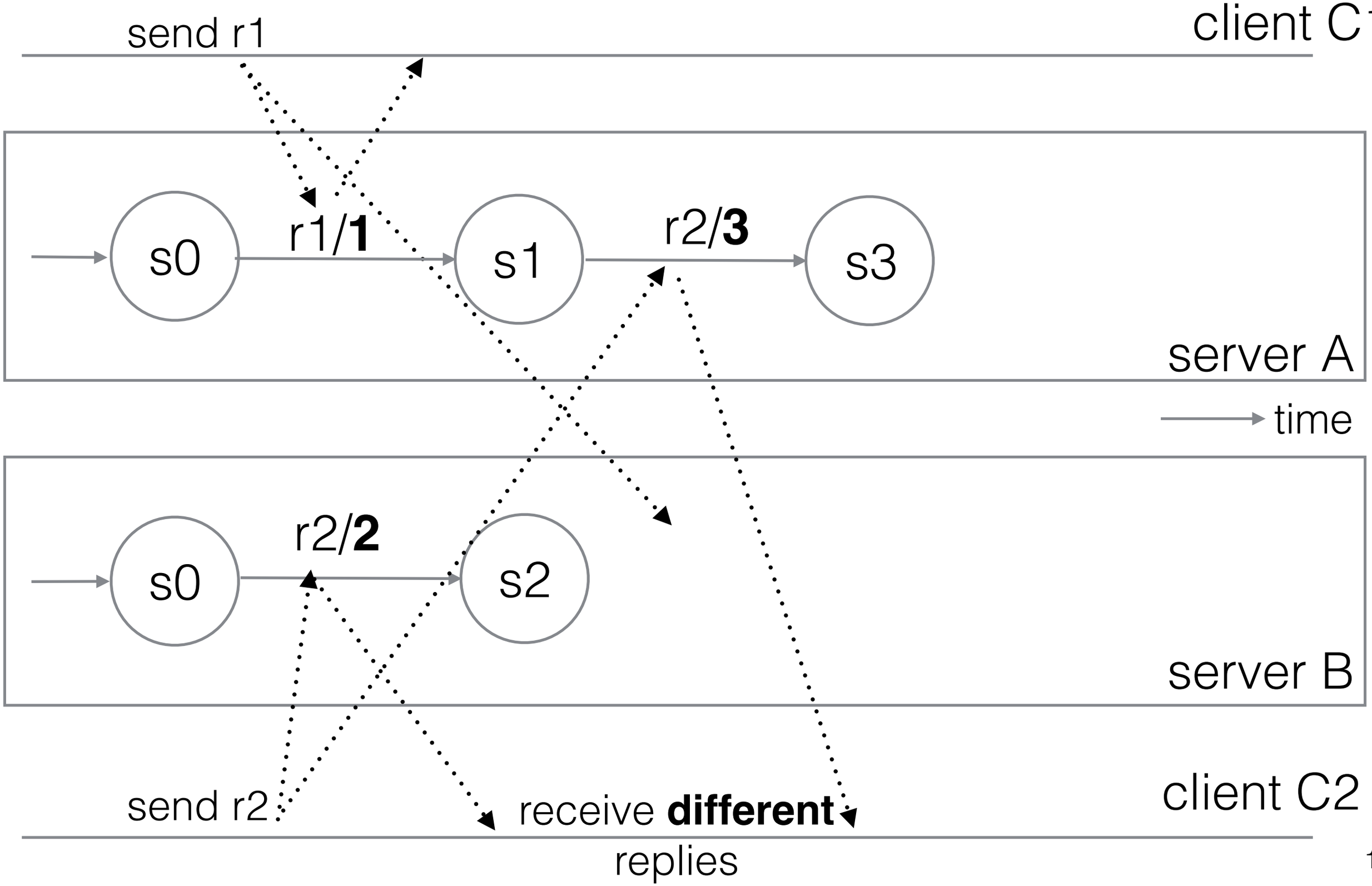
Out Of Order Execution



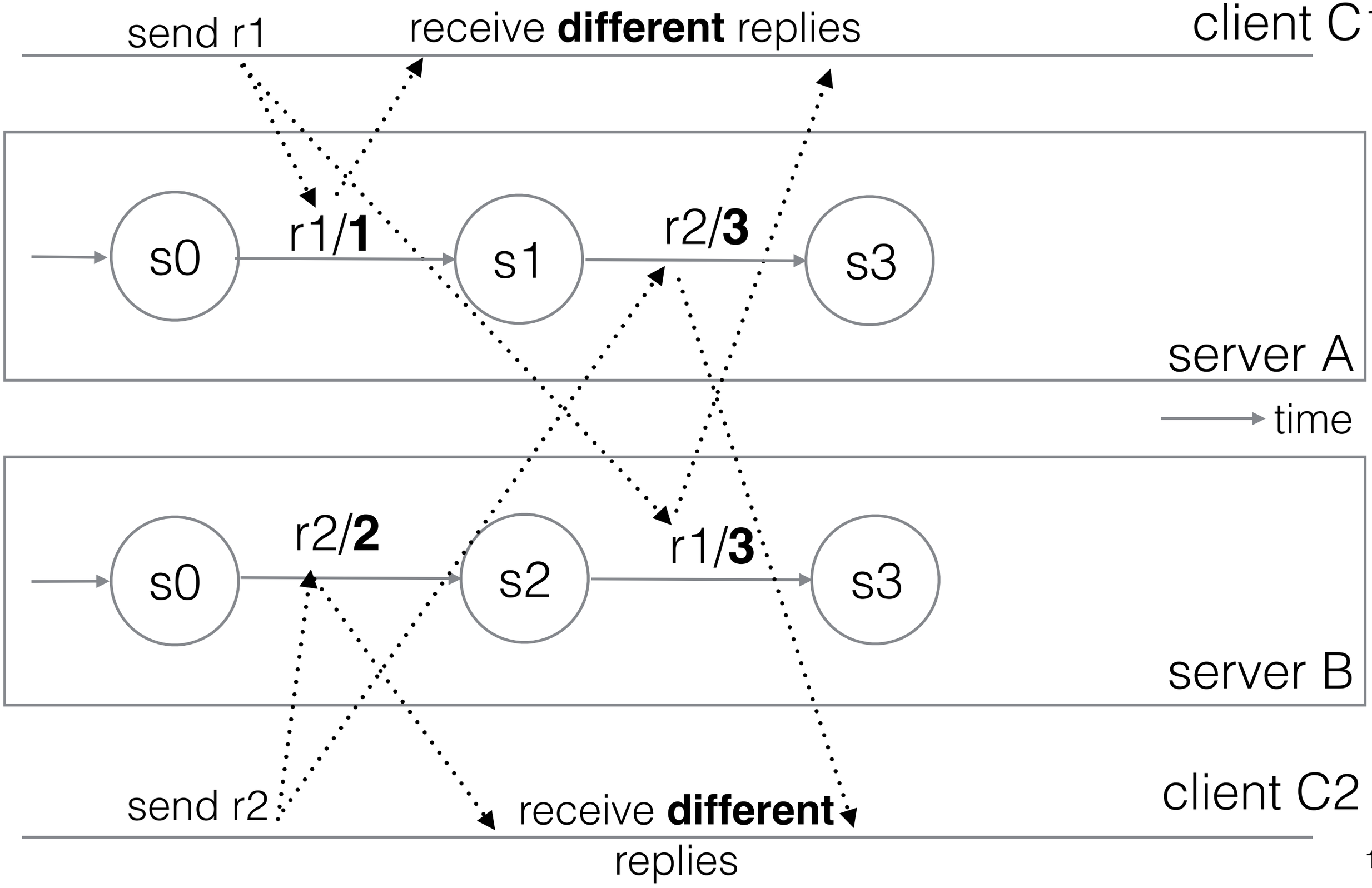
Out Of Order Execution



Out Of Order Execution



Out Of Order Execution



State Machine Replication

- **To ensure that replicas agree on the outputs and the internal state**
  - the state machine is deterministic
  - every replica executes the same sequence of client requests

Primary/Backup

- only primary executes requests
  - sends **state updates** to backup servers
  - can deal with non-deterministic state updates!

- usually (incremental) **delta updates**
  - must be applied by all replicas in same order as generated by leader
  - based on same initial state

Why not original Paxos?

- **Primary/Backup**:
  - a more specific solution than general consensus possible
    - „atomic broadcast“ (e.g., Zab)
- **Paxos**:
  - difficult to understand
  - difficult to tune
  - RAFT: a more understandable alternative?

RAFT

- used for state machine replication -

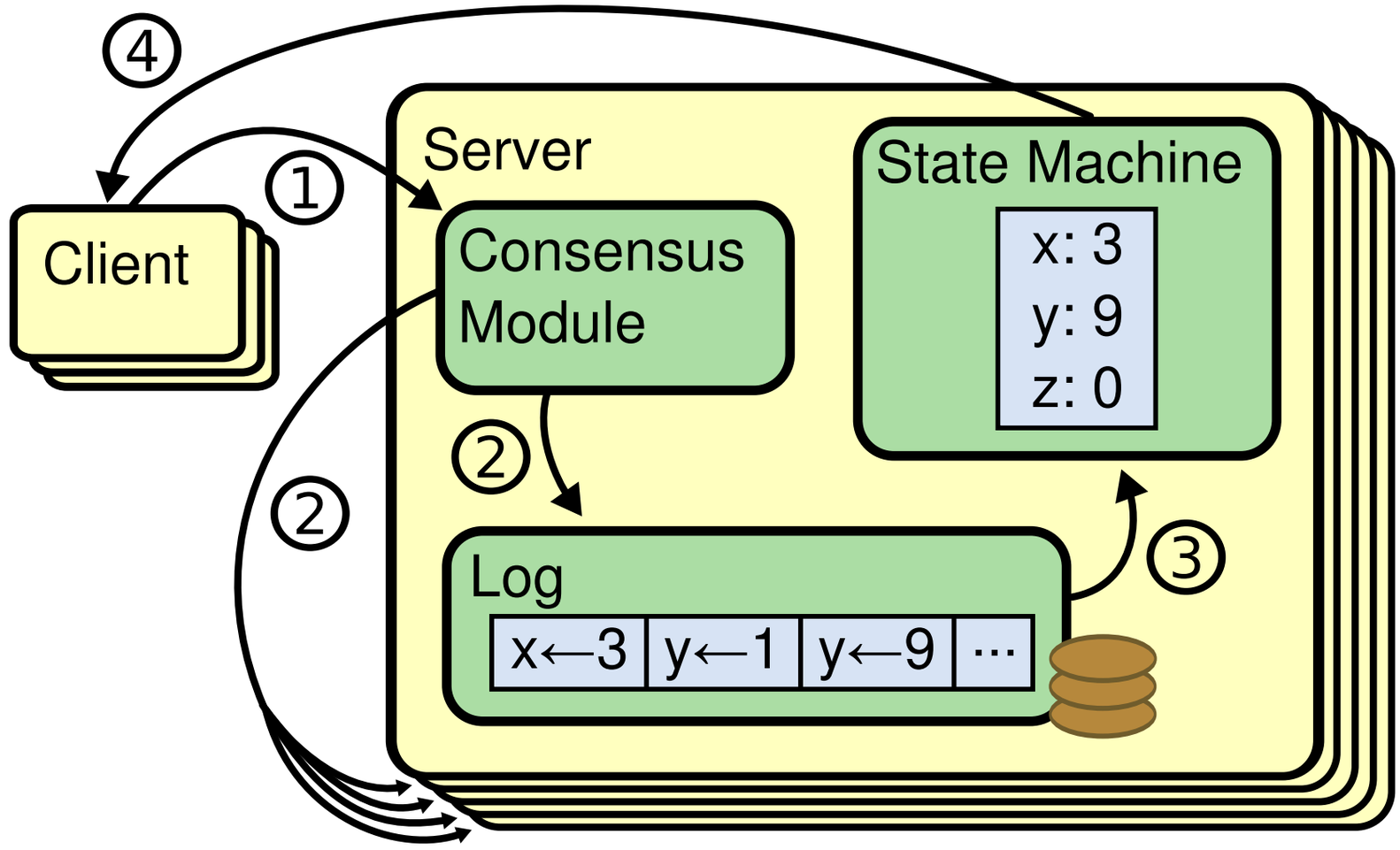
Raft

- Used in, e.g., Consul, etcd
- Leader serves **all** requests
- New leader starts new **term**
- Non-Byzantine fail-stop failure model

Motivation

- **Simpler Abstraction:**
  - Raft abstraction: **leader-based consensus** in the context of **state machine replication**
  - popular with engineers - easier to implement
- **Simpler Protocol:**
  - in order decision on log entries
  - Paxos requires extra protocol (not presented)
- **Leader election**:
  - Simple election but
  - Ensures safety that decisions are not reversed

Architecture



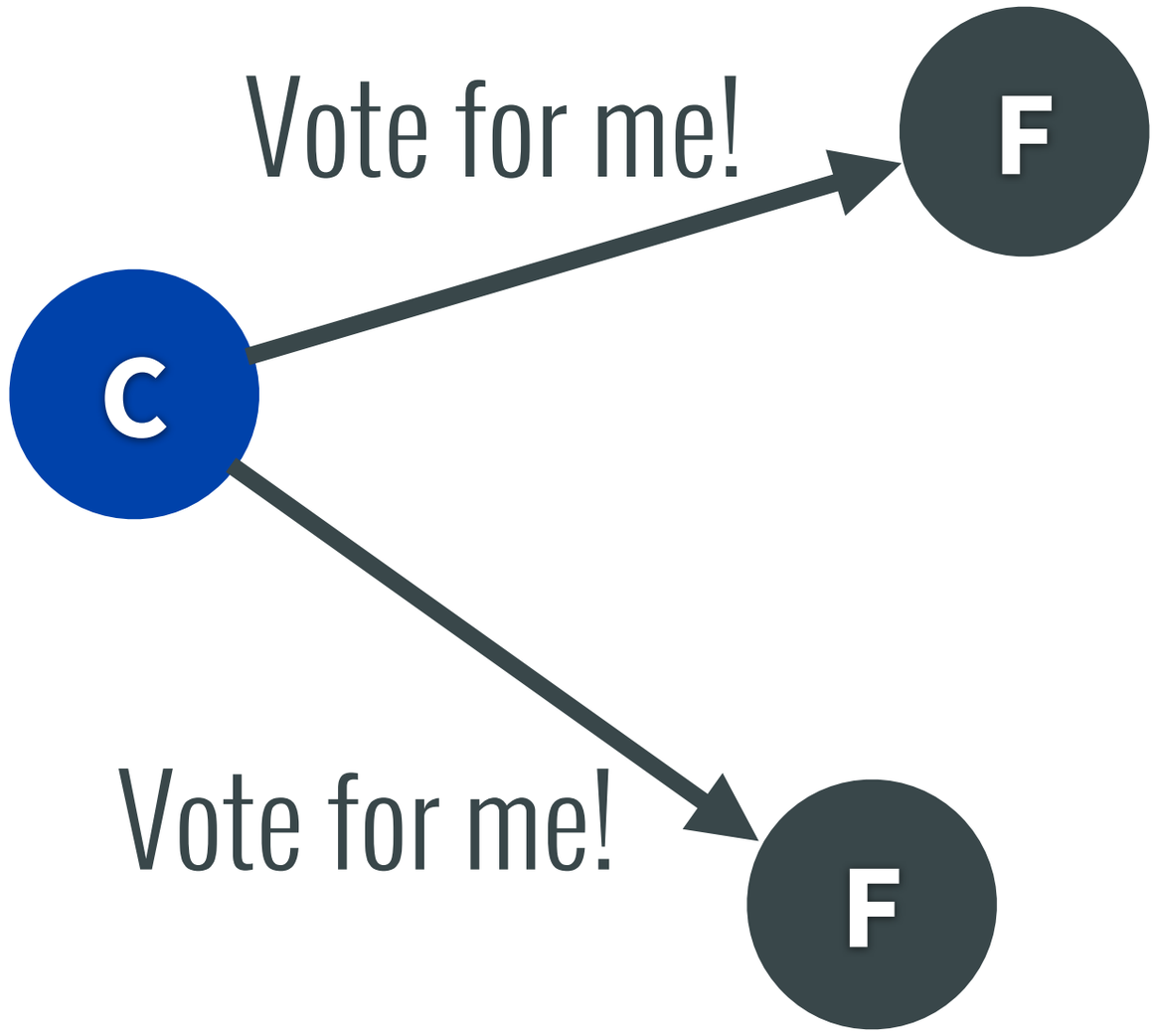
ONGARO, D., AND OUSTERHOUT, J. In search of an understandable consensus algorithm. In Proc ATC'14, USENIX Annual Technical Conference (2014), USENIX, May 2014

Three Roles

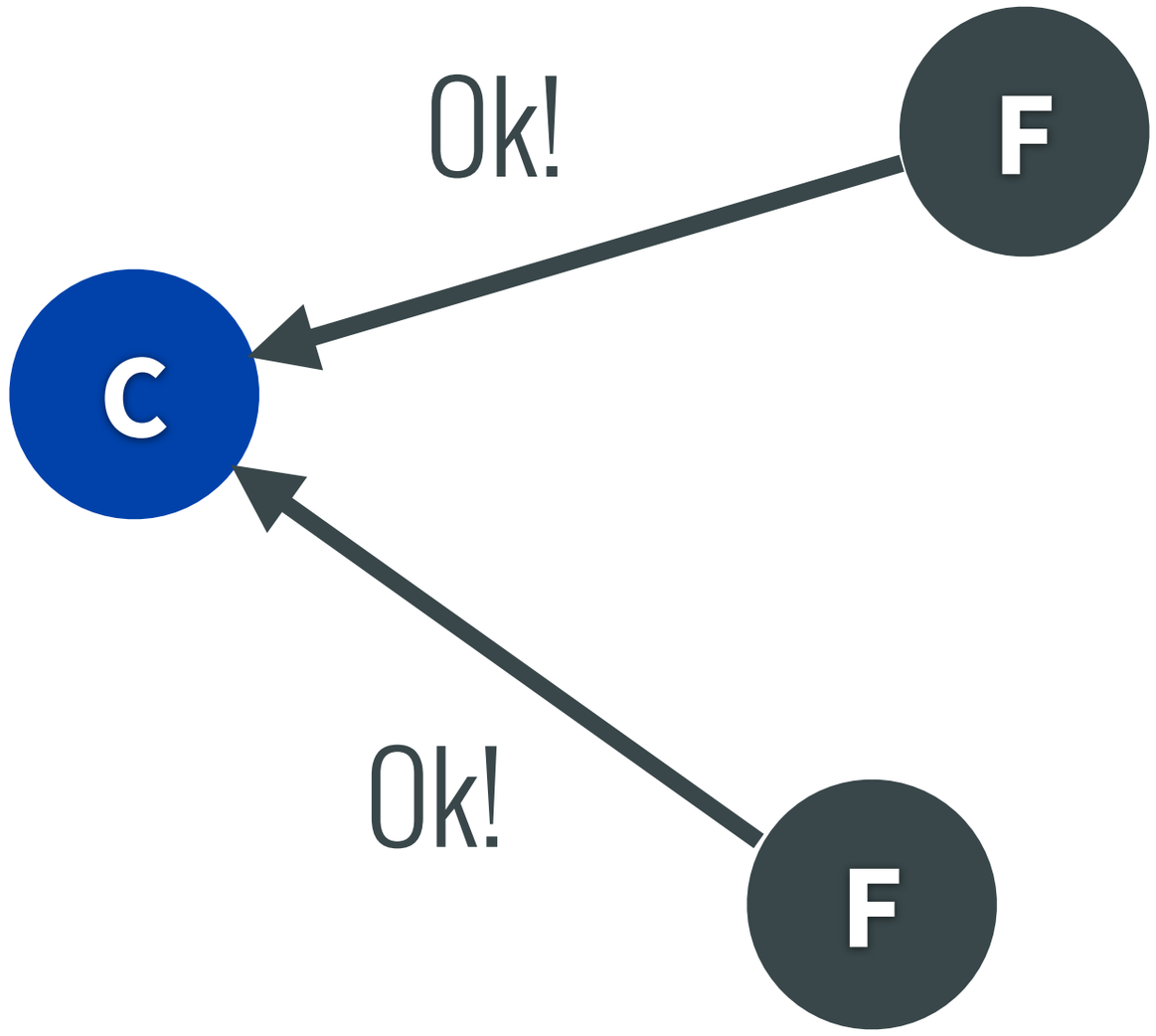
- Leader
- Follower
- Candidate

ONGARO, D., AND OUSTERHOUT, J. In search of an understandable consensus algorithm. In Proc ATC'14, USENIX Annual Technical Conference (2014), USENIX, May 2014

ONGARO, D., AND OUSTERHOUT, J. In search of an understandable consensus algorithm. In Proc ATC'14, USENIX Annual Technical Conference (2014), USENIX, May 2014

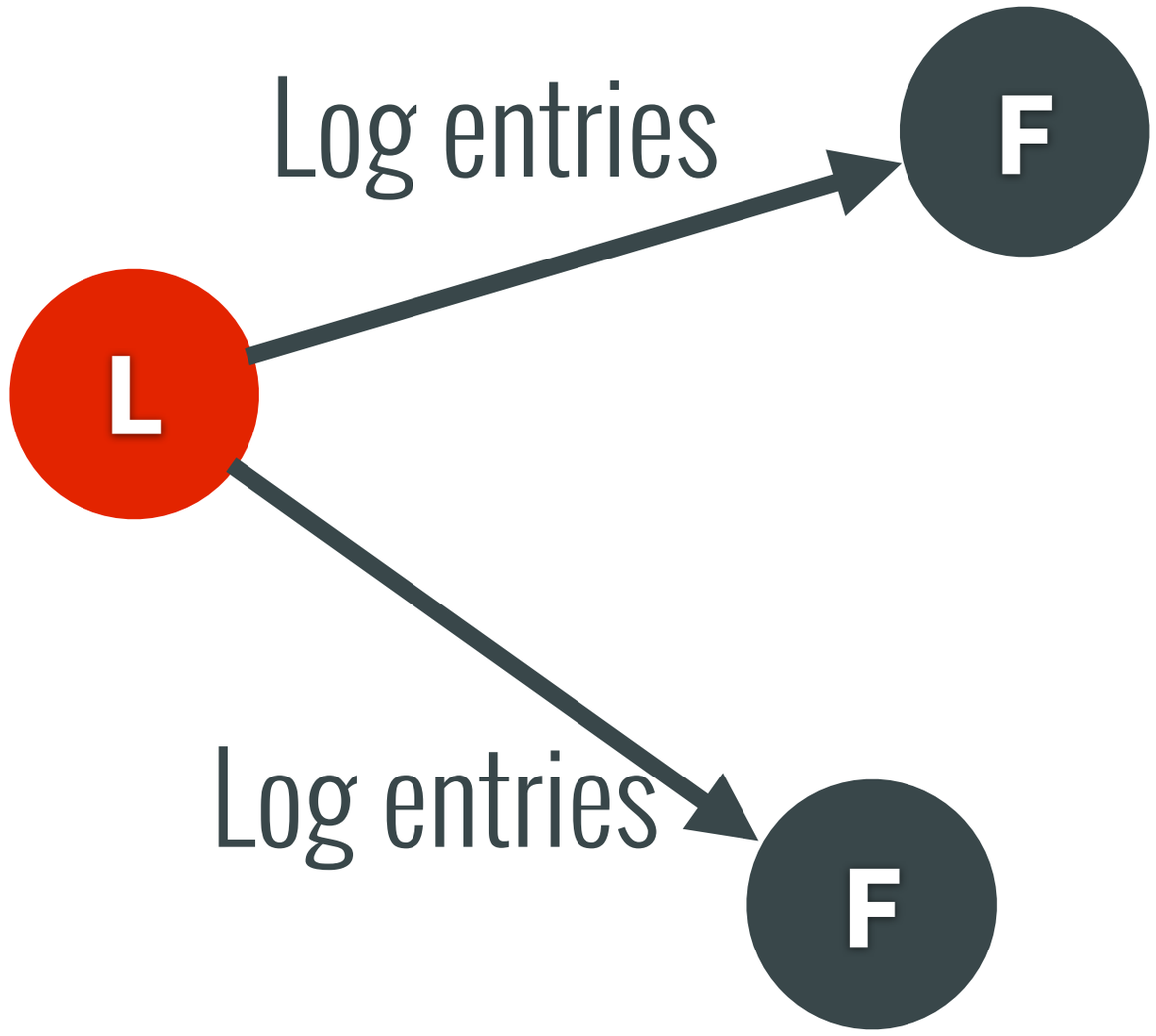


ONGARO, D., AND OUSTERHOUT, J. In search of an understandable consensus algorithm. In Proc ATC'14, USENIX Annual Technical Conference (2014), USENIX, May 2014

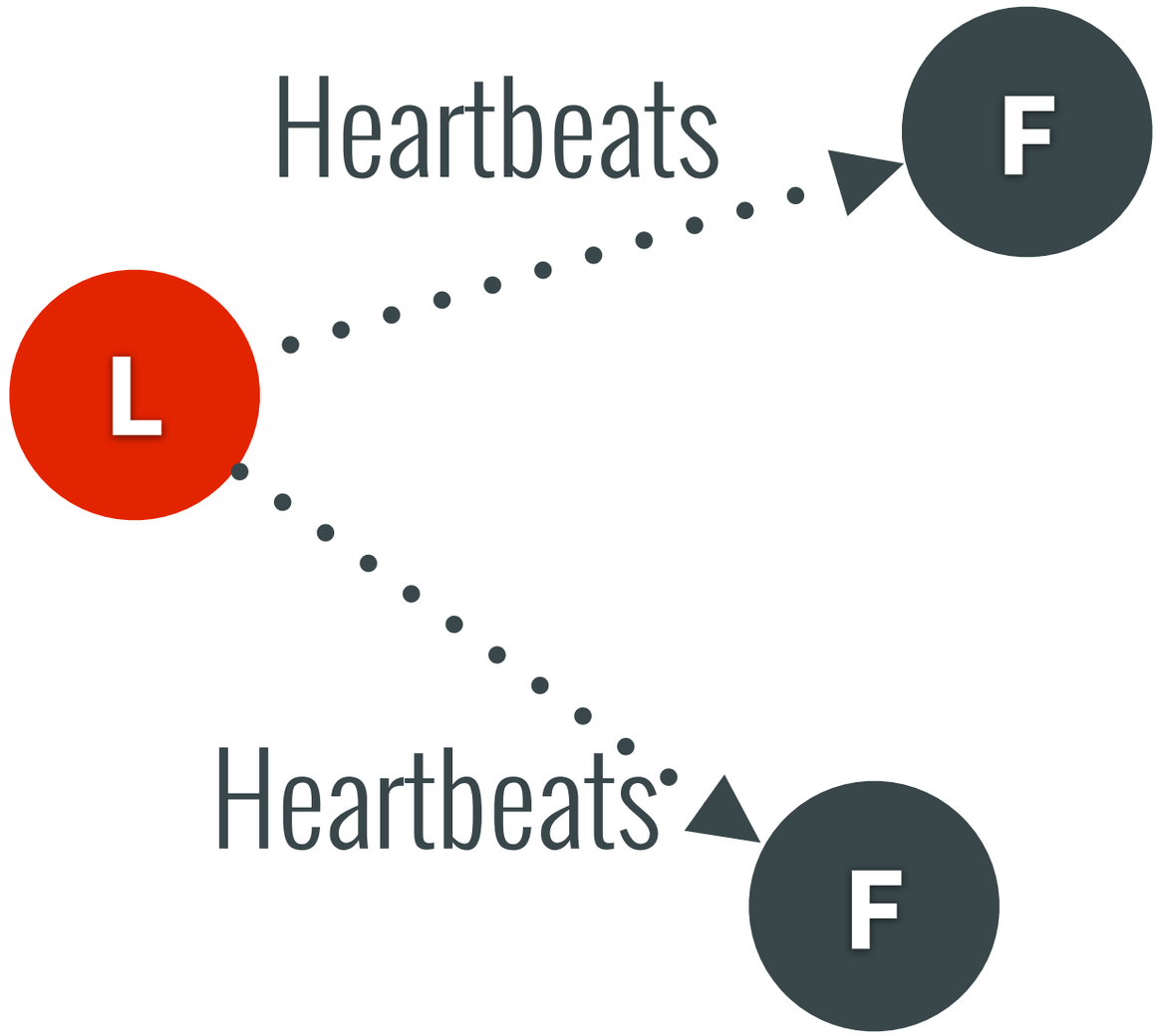


ONGARO, D., AND OUSTERHOUT, J. In search of an understandable consensus algorithm. In Proc ATC'14, USENIX Annual Technical Conference (2014), USENIX, May 2014

ONGARO, D., AND OUSTERHOUT, J. In search of an understandable consensus algorithm. In Proc ATC'14, USENIX Annual Technical Conference (2014), USENIX, May 2014



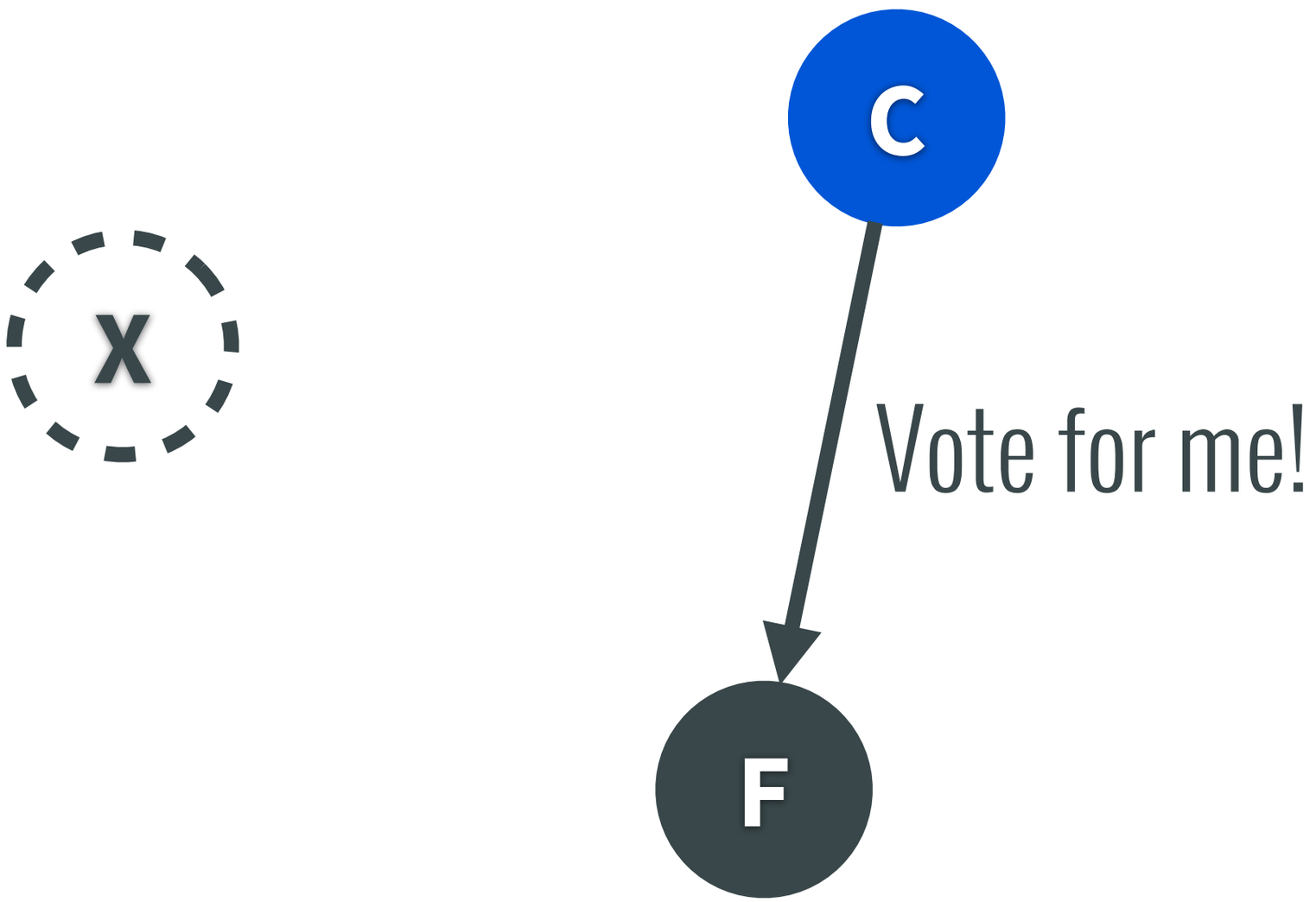
ONGARO, D., AND OUSTERHOUT, J. In search of an understandable consensus algorithm. In Proc ATC'14, USENIX Annual Technical Conference (2014), USENIX, May 2014



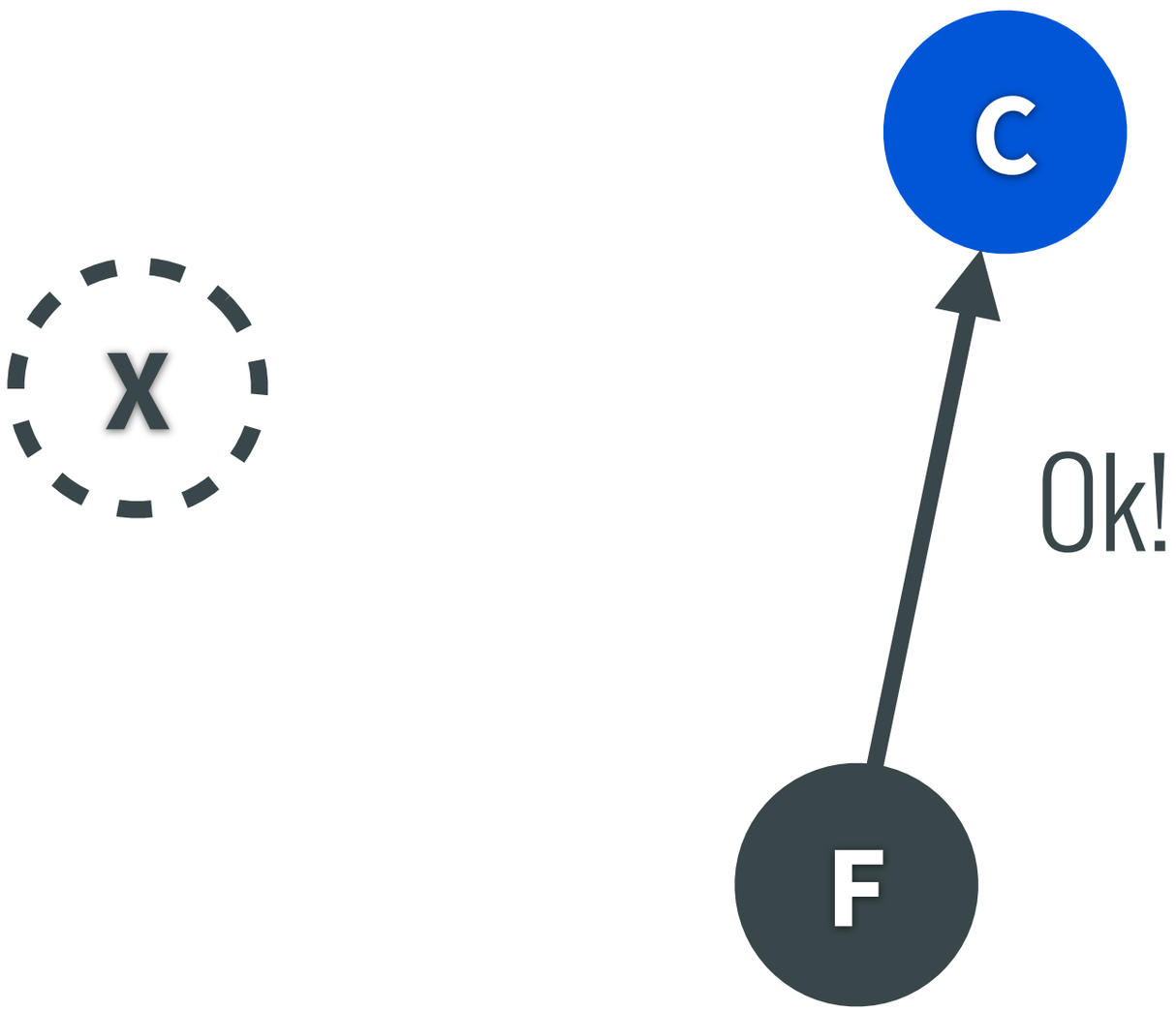
ONGARO, D., AND OUSTERHOUT, J. In search of an understandable consensus algorithm. In Proc ATC'14, USENIX Annual Technical Conference (2014), USENIX, May 2014

ONGARO, D., AND OUSTERHOUT, J. In search of an understandable consensus algorithm. In Proc ATC'14, USENIX Annual Technical Conference (2014), USENIX, May 2014

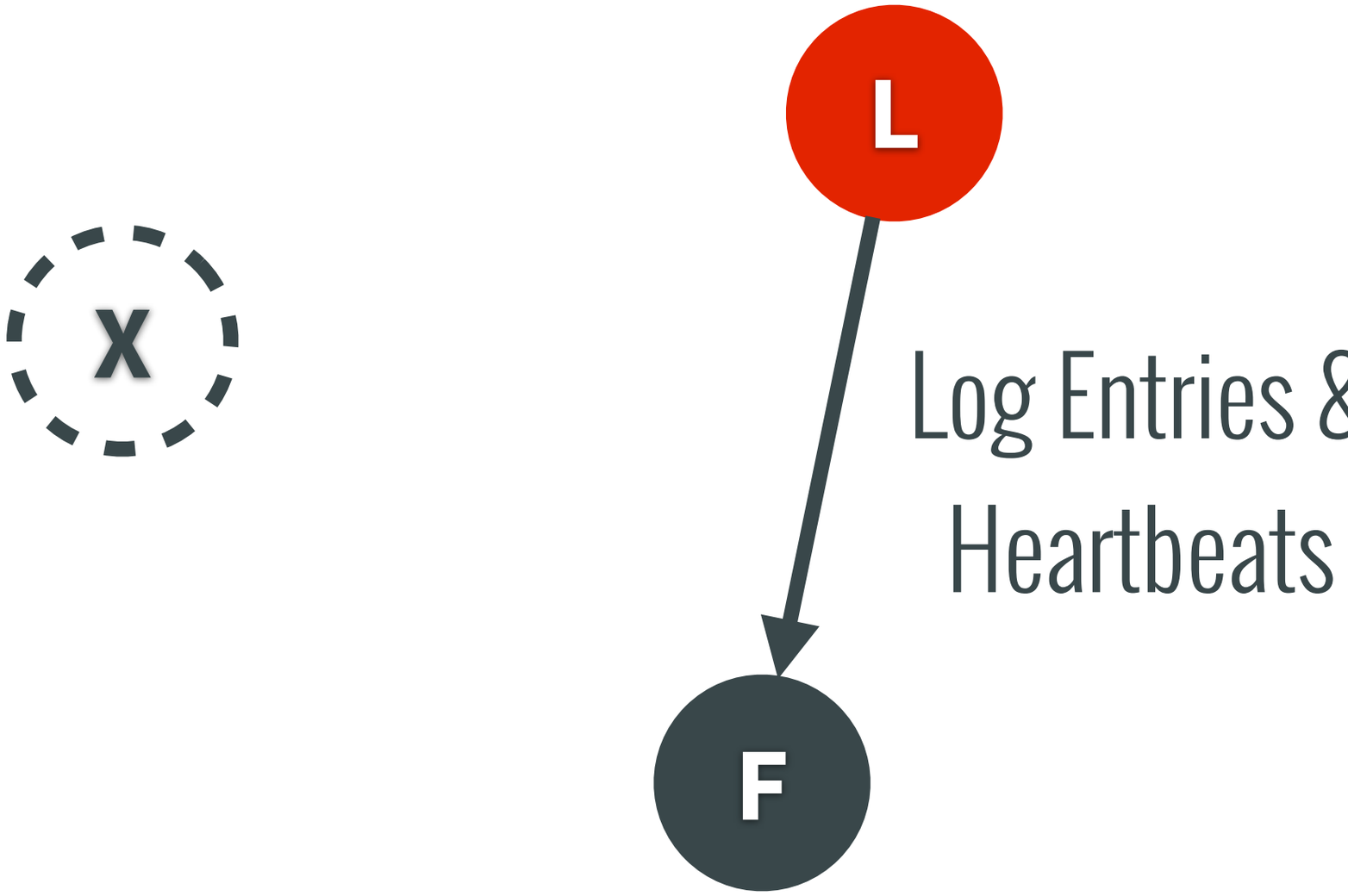
ONGARO, D., AND OUSTERHOUT, J. In search of an understandable consensus algorithm. In Proc ATC'14, USENIX Annual Technical Conference (2014), USENIX, May 2014



ONGARO, D., AND OUSTERHOUT, J. In search of an understandable consensus algorithm. In Proc ATC'14, USENIX Annual Technical Conference (2014), USENIX, May 2014



ONGARO, D., AND OUSTERHOUT, J. In search of an understandable consensus algorithm. In Proc ATC'14, USENIX Annual Technical Conference (2014), USENIX, May 2014

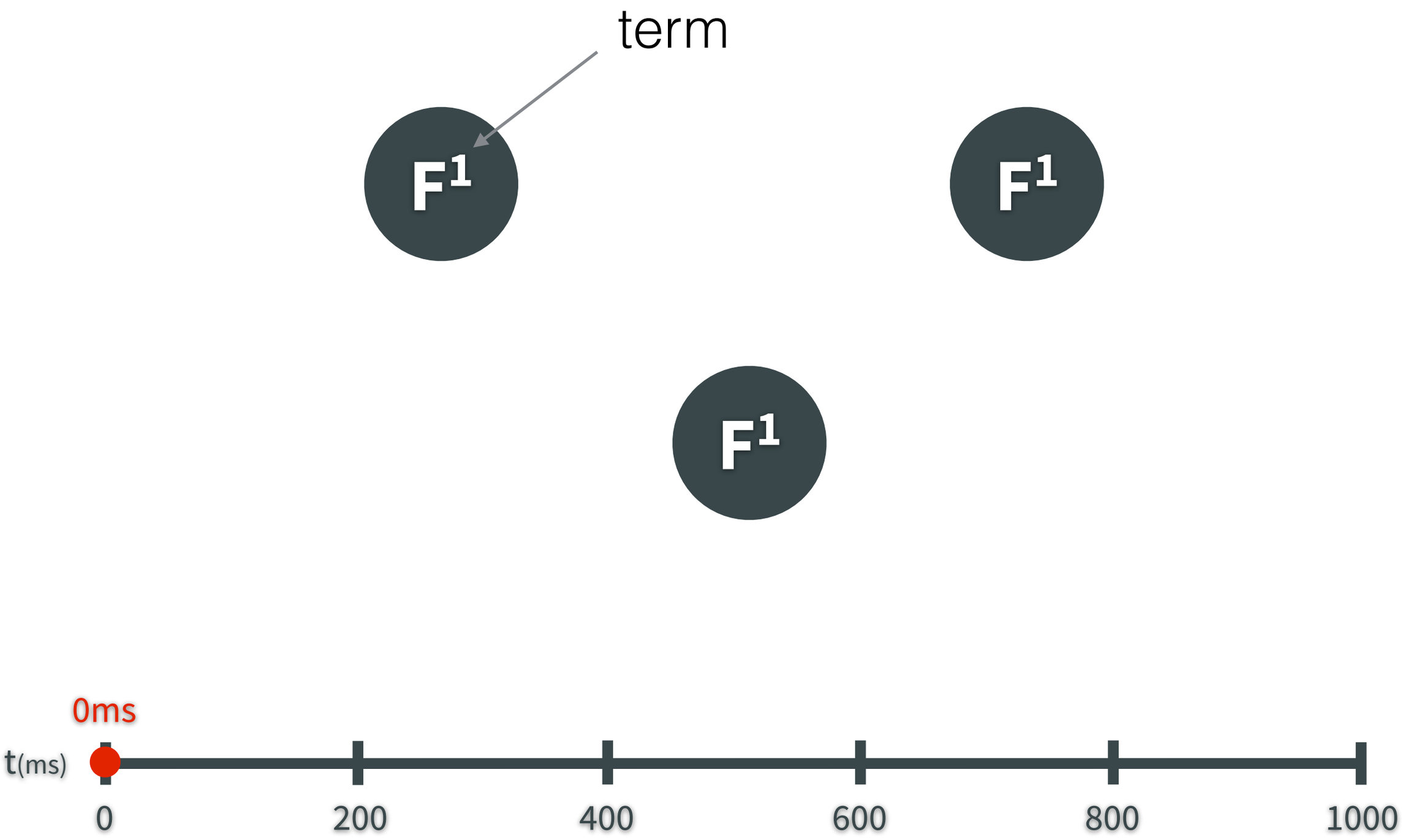


ONGARO, D., AND OUSTERHOUT, J. In search of an understandable consensus algorithm. In Proc ATC'14, USENIX Annual Technical Conference (2014), USENIX, May 2014

Leader Election

ONGARO, D., AND OUSTERHOUT, J. In search of an understandable consensus algorithm. In Proc ATC'14, USENIX Annual Technical Conference (2014), USENIX, May 2014

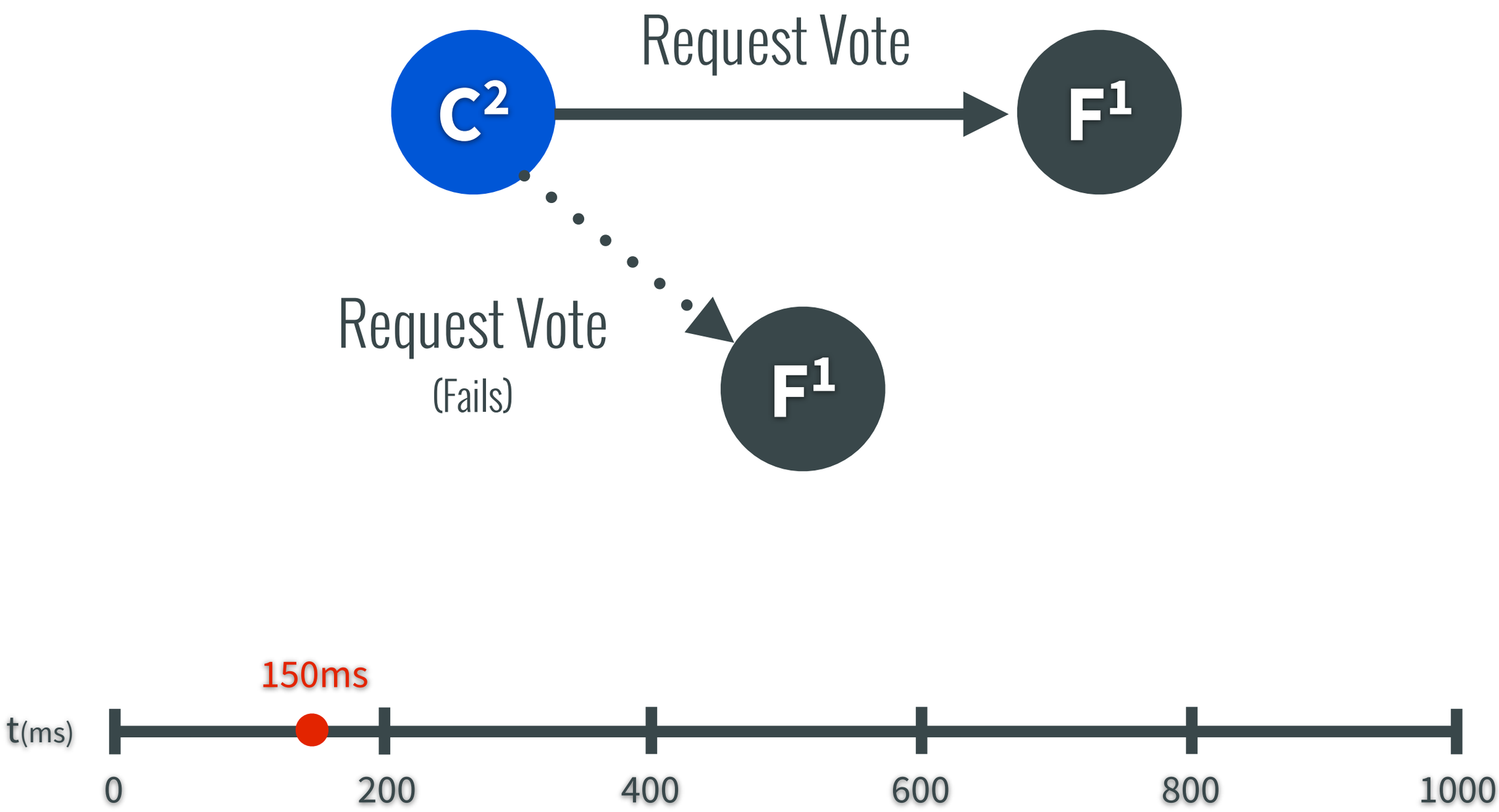
Leader Election



ONGARO, D., AND OUSTERHOUT, J. In search of an understandable consensus algorithm. In Proc ATC'14, USENIX Annual Technical Conference (2014), USENIX, May 2014

Leader Election

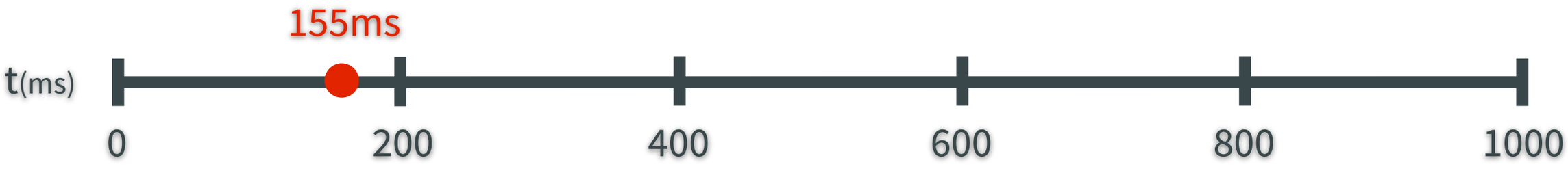
One follower becomes a candidate after an election timeout and requests votes



ONGARO, D., AND OUSTERHOUT, J. In search of an understandable consensus algorithm. In Proc ATC'14, USENIX Annual Technical Conference (2014), USENIX, May 2014

Leader Election

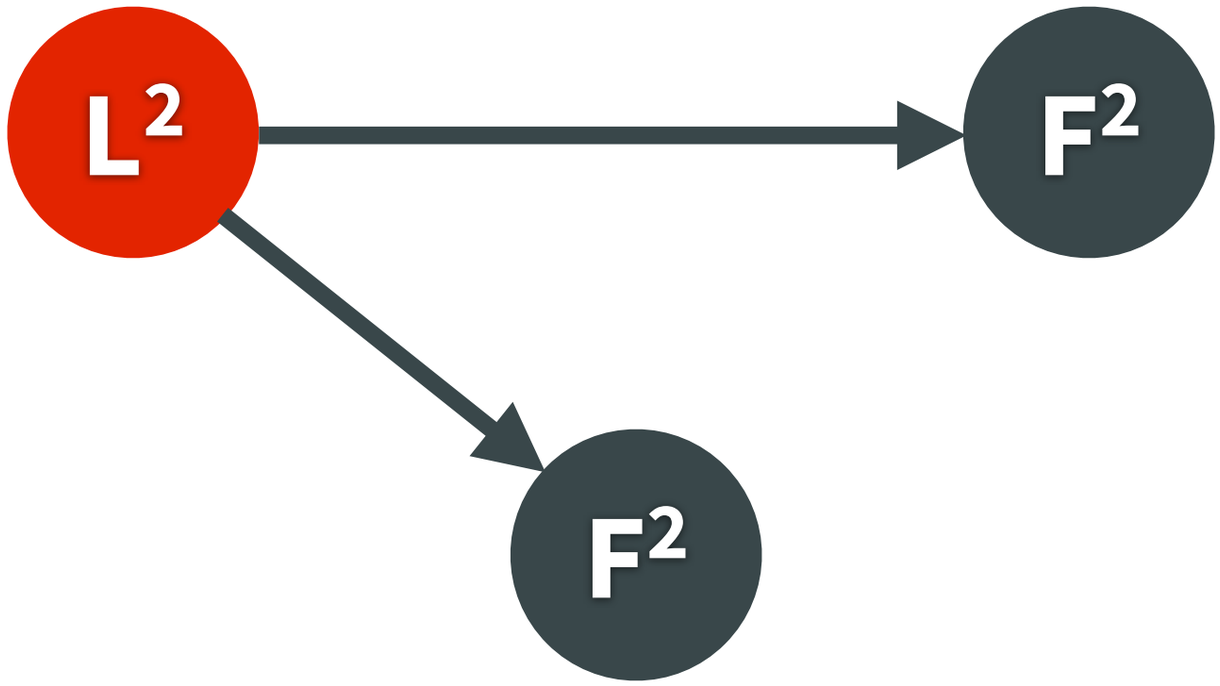
Candidate receives one vote from a peer and one vote from self

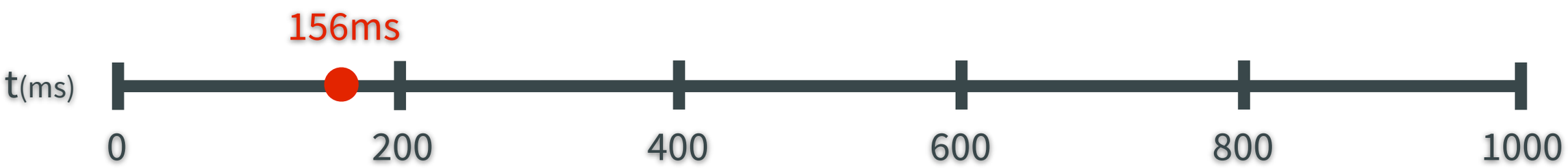


ONGARO, D., AND OUSTERHOUT, J. In search of an understandable consensus algorithm. In Proc ATC'14, USENIX Annual Technical Conference (2014), USENIX, May 2014

Leader Election

Two votes is a majority so candidate becomes leader



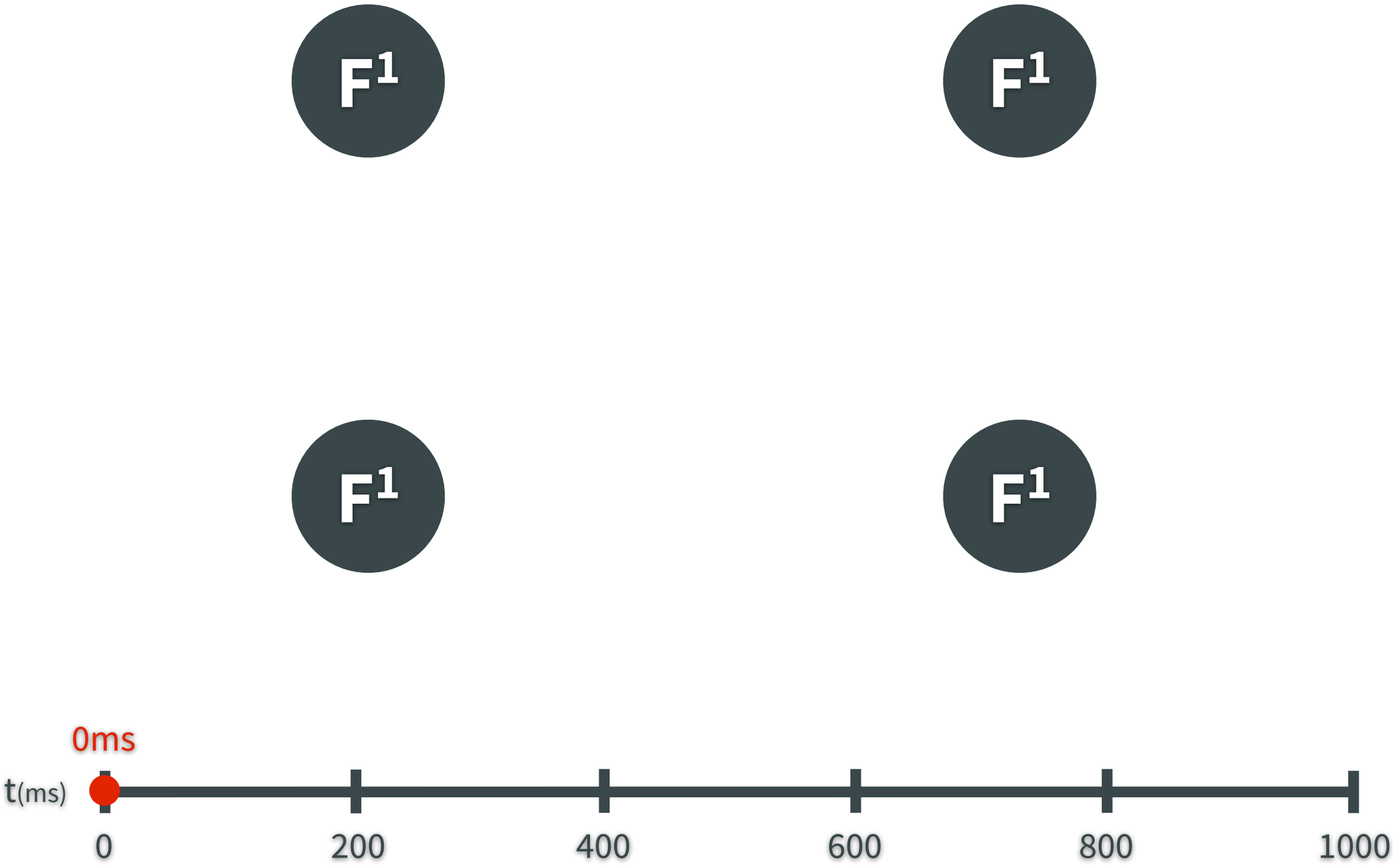


ONGARO, D., AND OUSTERHOUT, J. In search of an understandable consensus algorithm. In Proc ATC'14, USENIX Annual Technical Conference (2014), USENIX, May 2014

Leader Election (Split Vote)

ONGARO, D., AND OUSTERHOUT, J. In search of an understandable consensus algorithm. In Proc ATC'14, USENIX Annual Technical Conference (2014), USENIX, May 2014

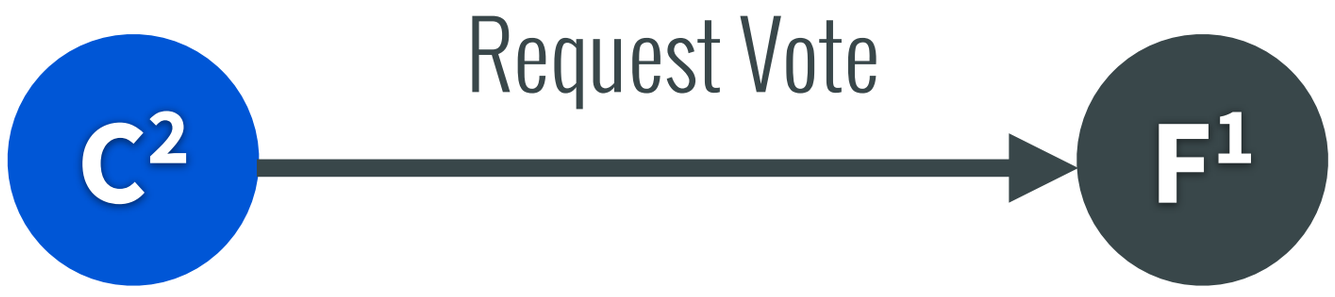
Leader Election

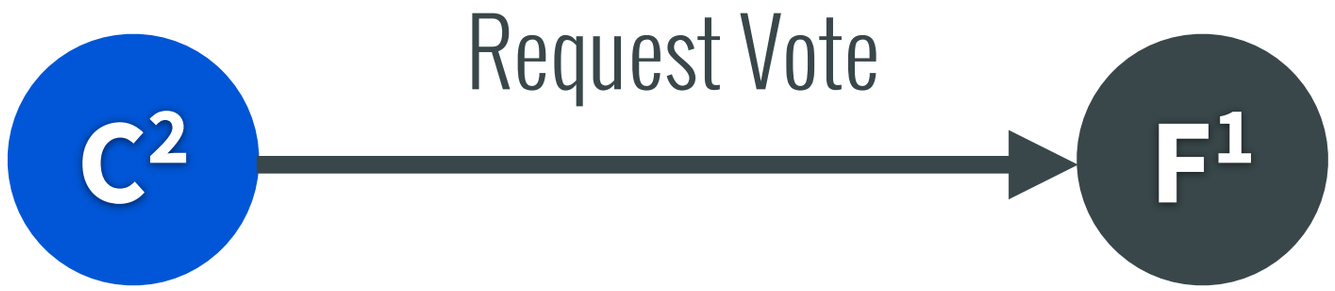


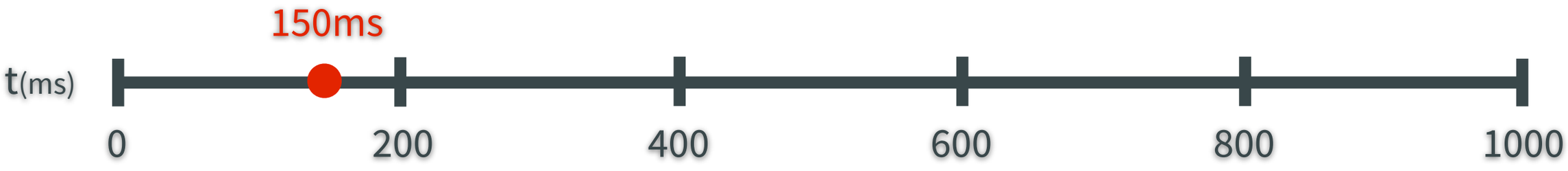
ONGARO, D., AND OUSTERHOUT, J. In search of an understandable consensus algorithm. In Proc ATC'14, USENIX Annual Technical Conference (2014), USENIX, May 2014

Leader Election

Two followers become candidates simultaneously and begin requesting votes



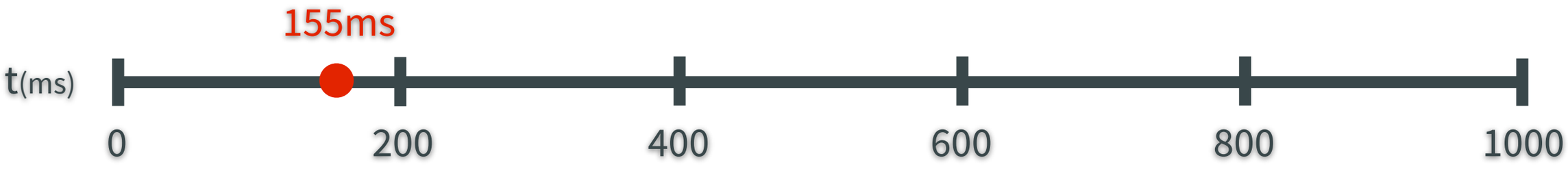




ONGARO, D., AND OUSTERHOUT, J. In search of an understandable consensus algorithm. In Proc ATC'14, USENIX Annual Technical Conference (2014), USENIX, May 2014

Leader Election

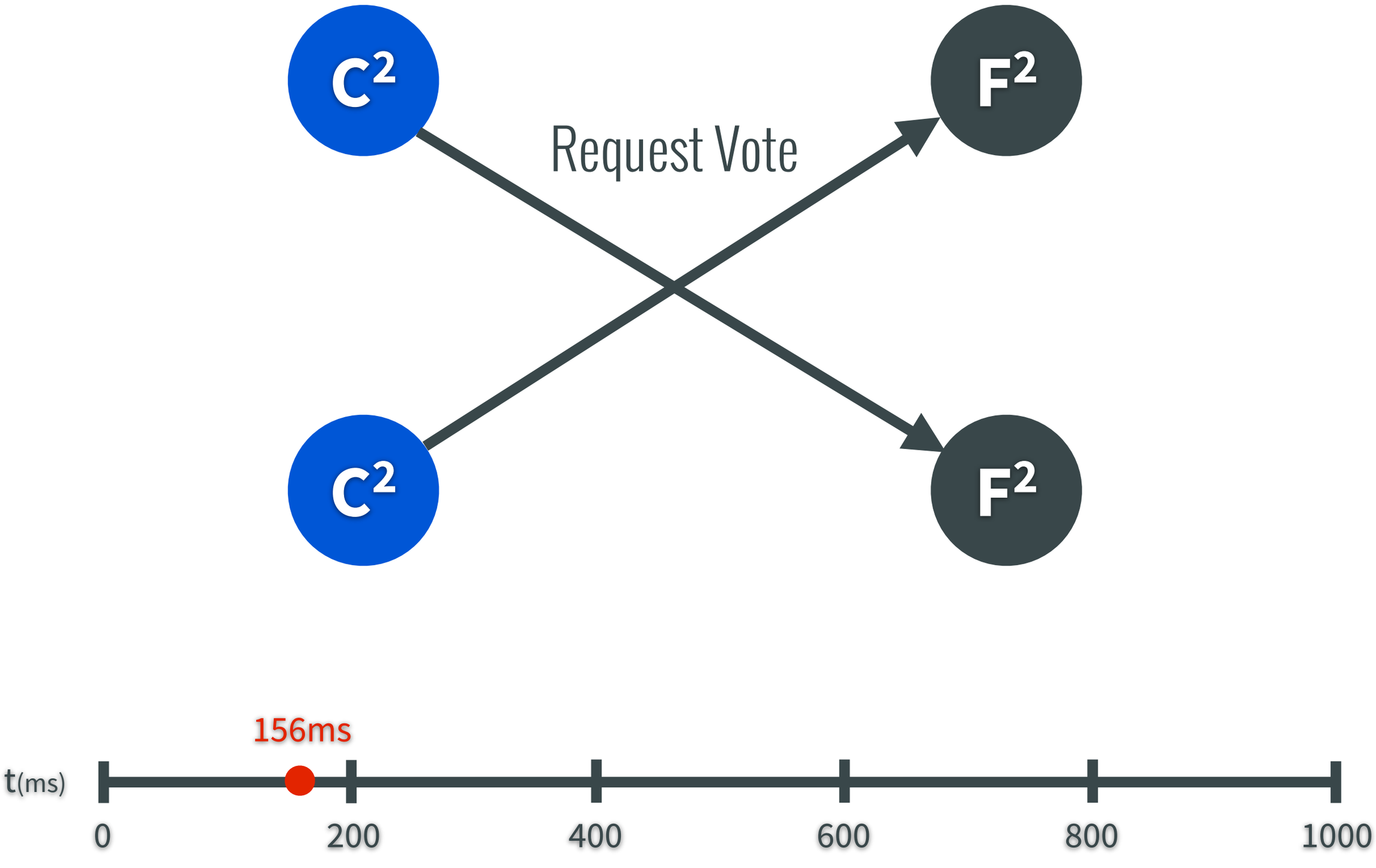
Each candidate receives a vote from themselves and from one peer



ONGARO, D., AND OUSTERHOUT, J. In search of an understandable consensus algorithm. In Proc ATC'14, USENIX Annual Technical Conference (2014), USENIX, May 2014

Leader Election

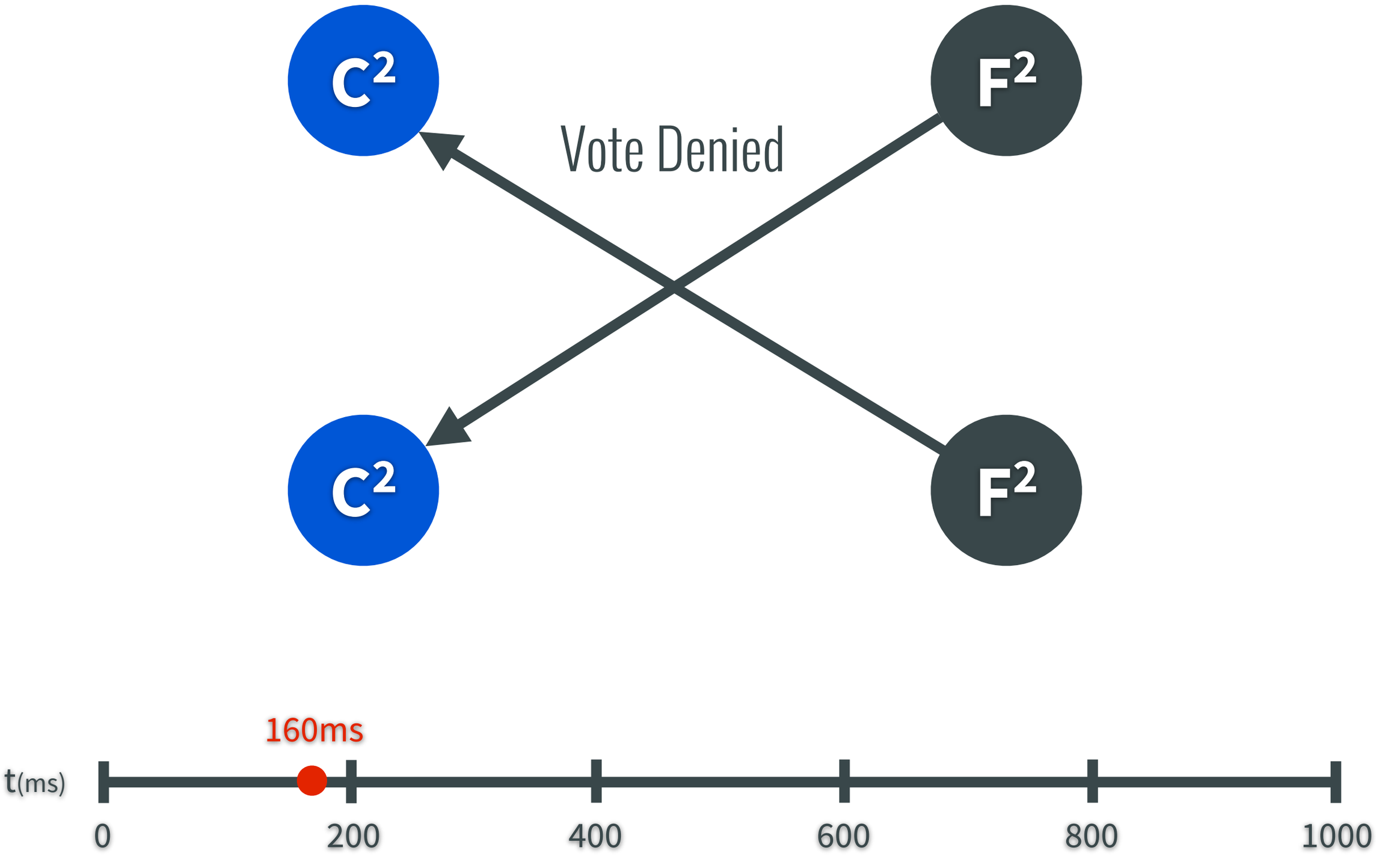
Each candidate requests a vote from a peer who has already voted



ONGARO, D., AND OUSTERHOUT, J. In search of an understandable consensus algorithm. In Proc ATC'14, USENIX Annual Technical Conference (2014), USENIX, May 2014

Leader Election

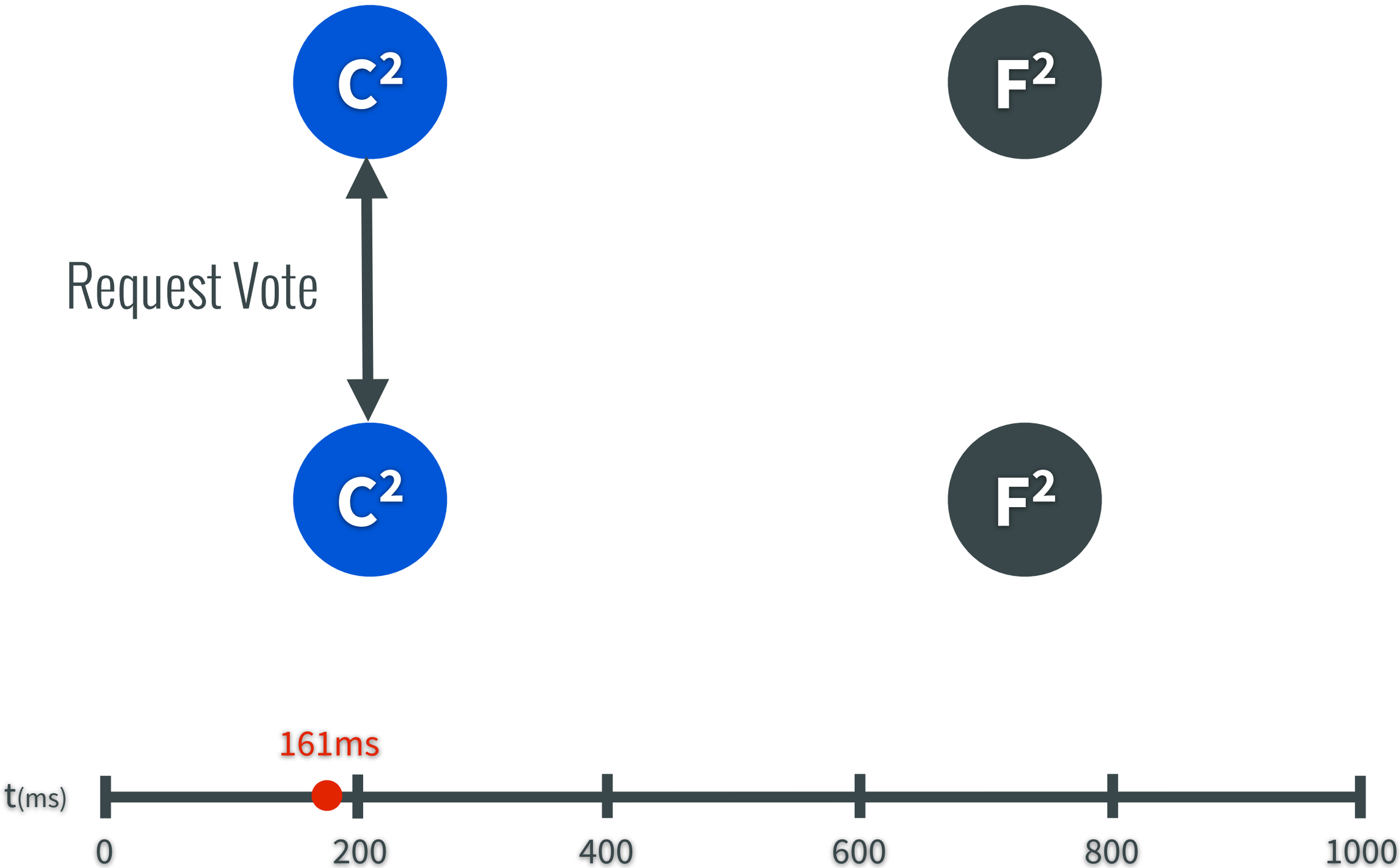
Vote requests are denied because the follower has already voted



ONGARO, D., AND OUSTERHOUT, J. In search of an understandable consensus algorithm. In Proc ATC'14, USENIX Annual Technical Conference (2014), USENIX, May 2014

Leader Election

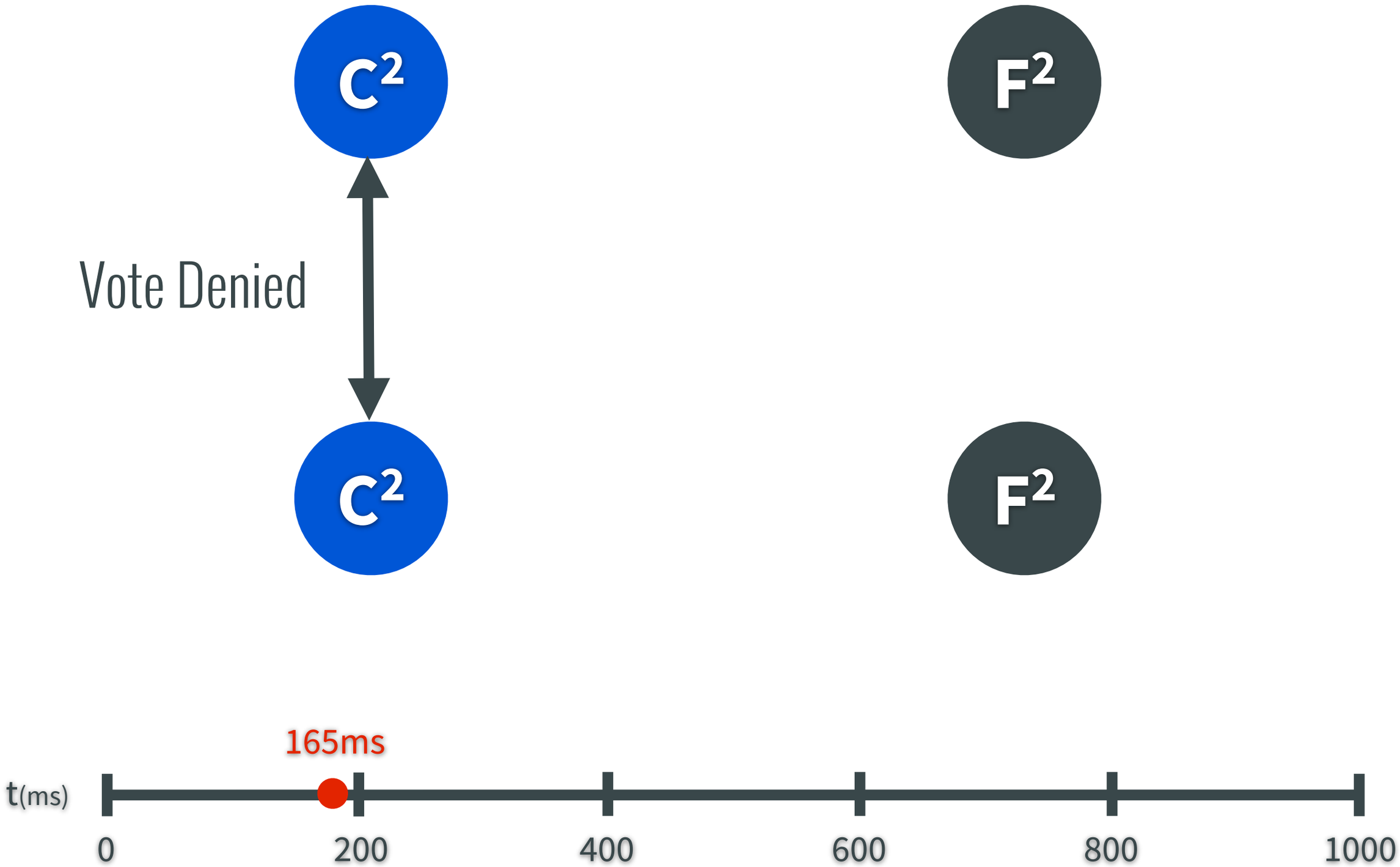
Candidates try to request votes from each other



ONGARO, D., AND OUSTERHOUT, J. In search of an understandable consensus algorithm. In Proc ATC'14, USENIX Annual Technical Conference (2014), USENIX, May 2014

Leader Election

Vote requests are denied because candidates voted for themselves



ONGARO, D., AND OUSTERHOUT, J. In search of an understandable consensus algorithm. In Proc ATC'14, USENIX Annual Technical Conference (2014), USENIX, May 2014

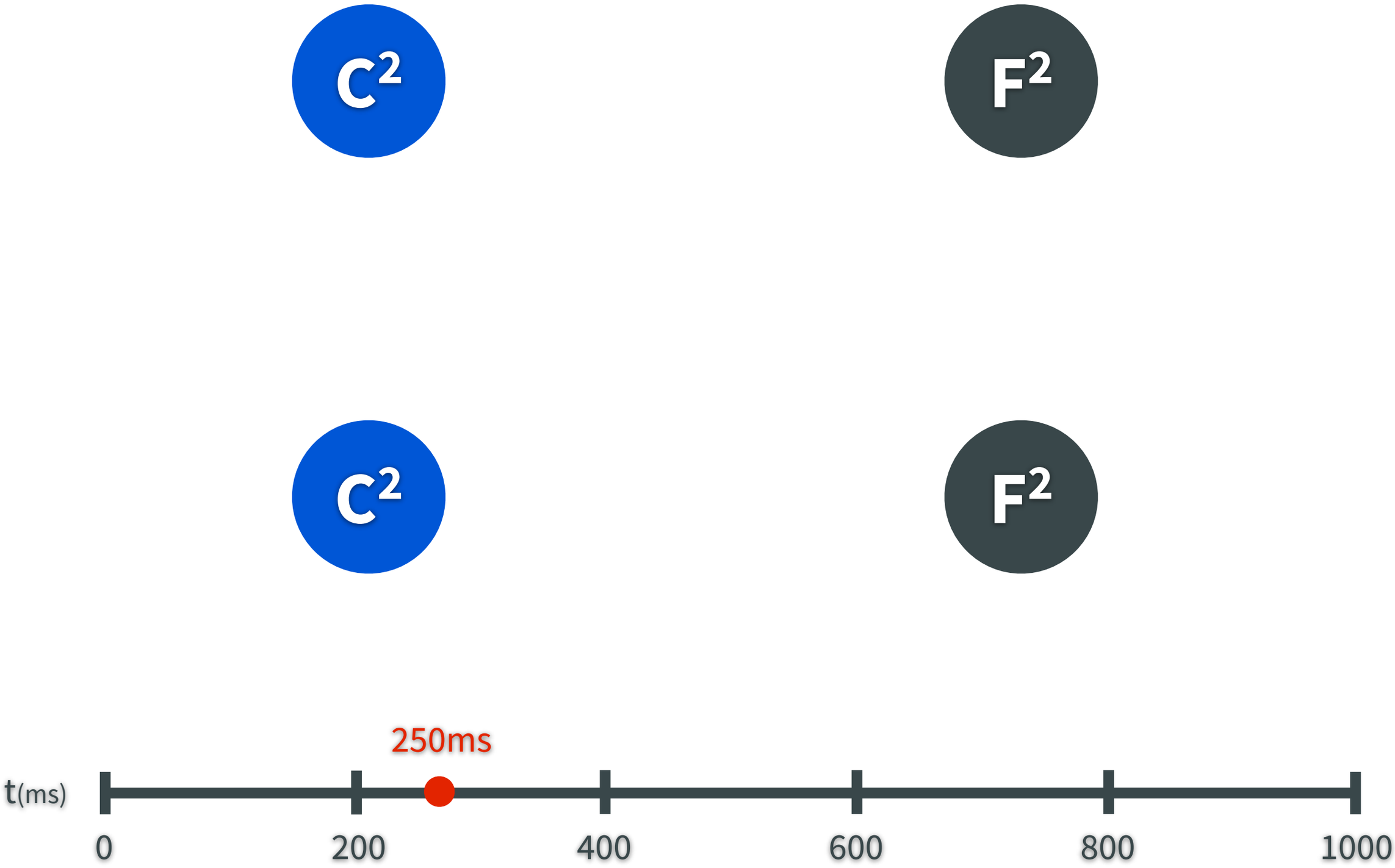
Leader Election

Candidates wait for a randomized election timeout to occur (150ms - 300ms)

ONGARO, D., AND OUSTERHOUT, J. In search of an understandable consensus algorithm. In Proc ATC'14, USENIX Annual Technical Conference (2014), USENIX, May 2014

Leader Election

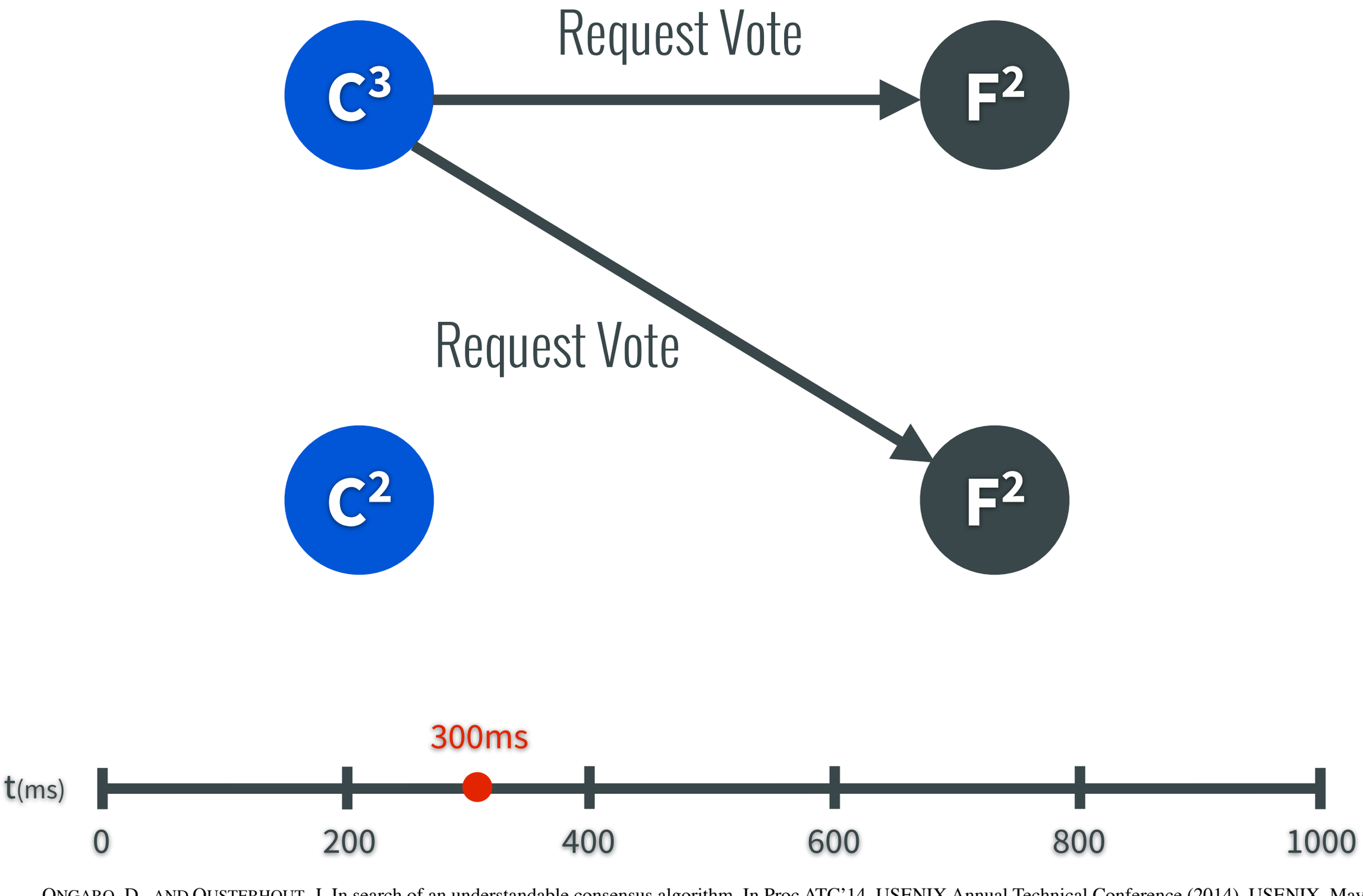
Still waiting...



ONGARO, D., AND OUSTERHOUT, J. In search of an understandable consensus algorithm. In Proc ATC'14, USENIX Annual Technical Conference (2014), USENIX, May 2014

Leader Election

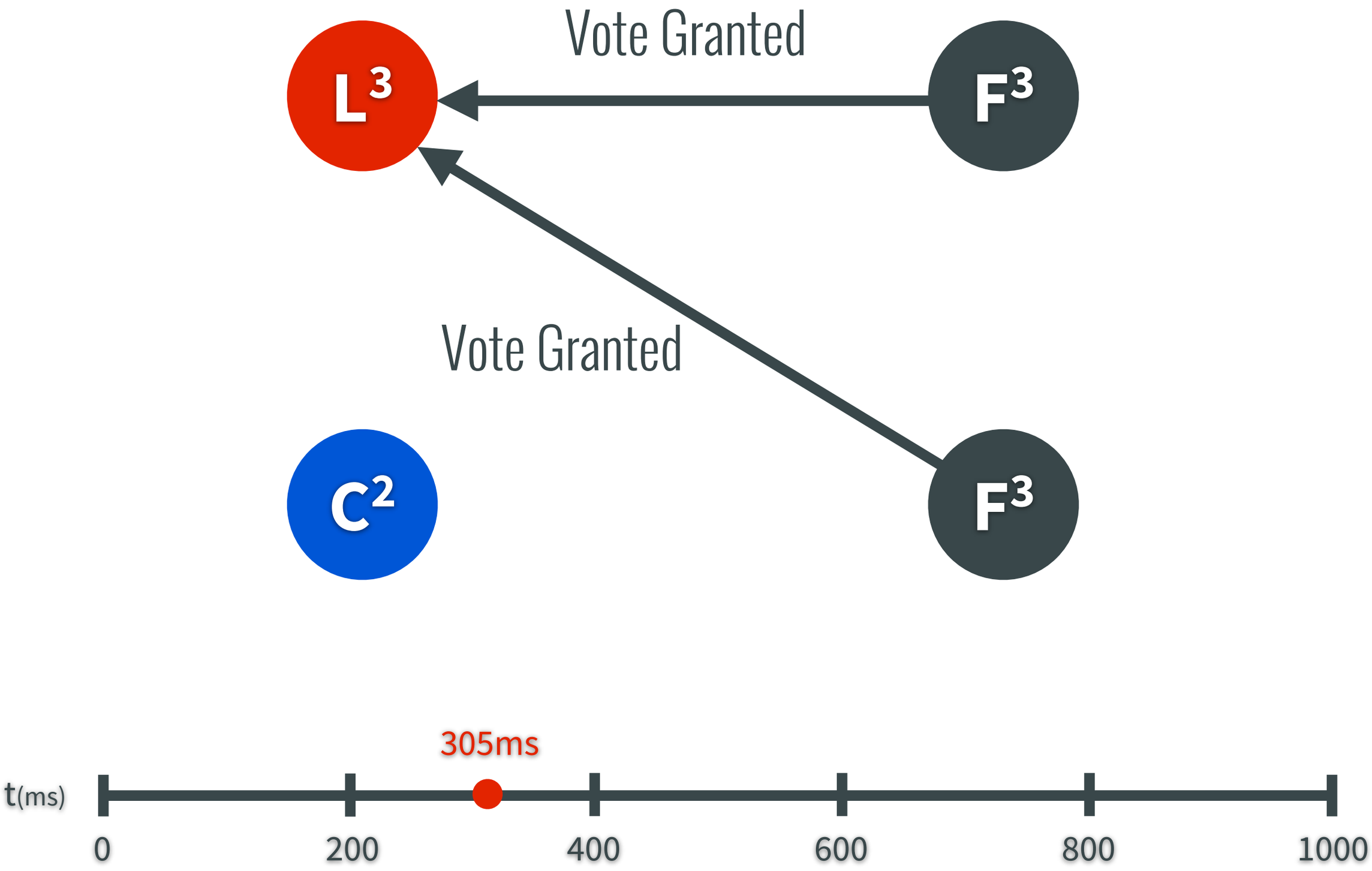
One candidate begins election term #3



ONGARO, D., AND OUSTERHOUT, J. In search of an understandable consensus algorithm. In Proc ATC'14, USENIX Annual Technical Conference (2014), USENIX, May 2014

Leader Election

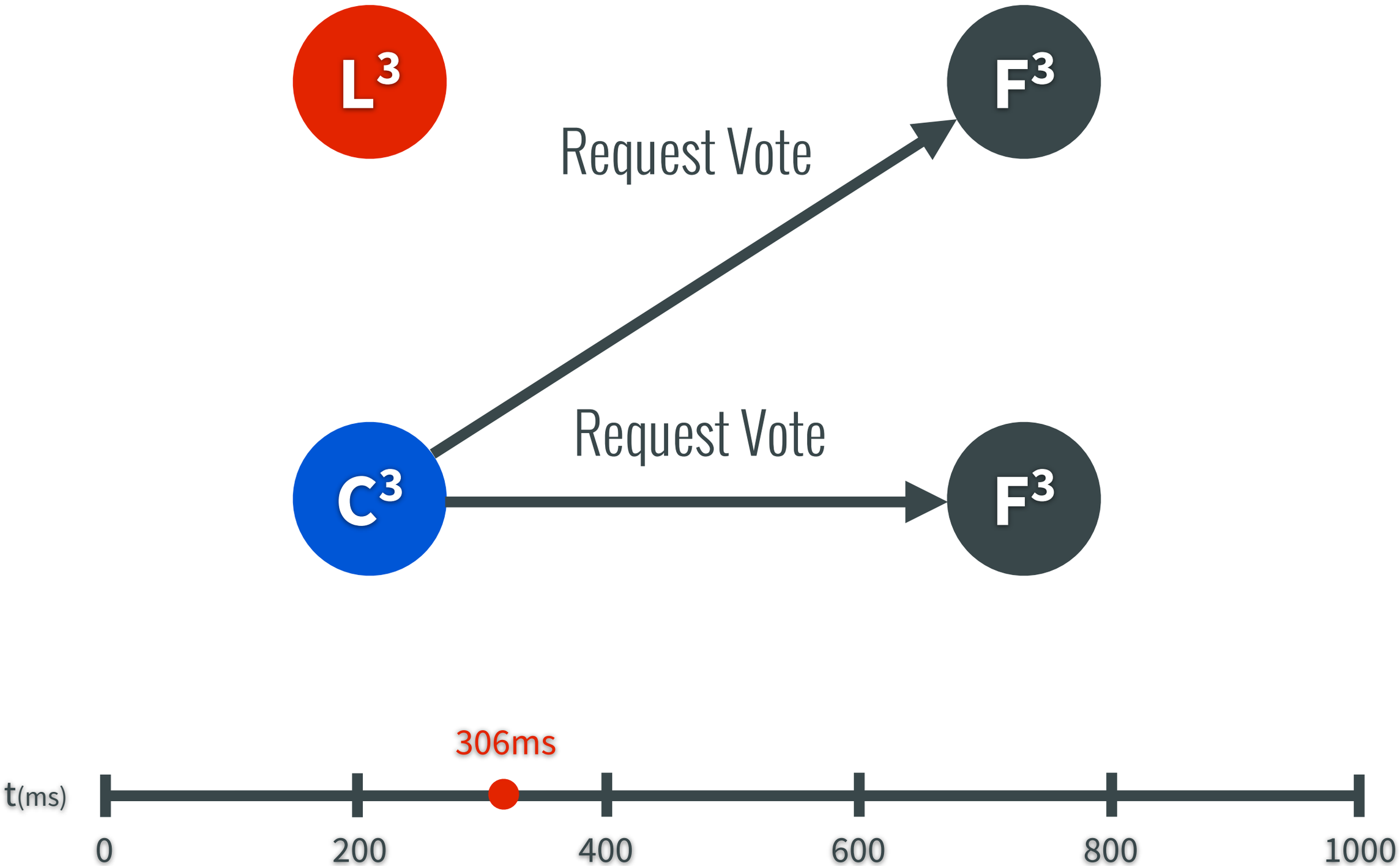
Candidate receives vote from itself and two peer votes so it becomes leader for election term #3



ONGARO, D., AND OUSTERHOUT, J. In search of an understandable consensus algorithm. In Proc ATC'14, USENIX Annual Technical Conference (2014), USENIX, May 2014

Leader Election

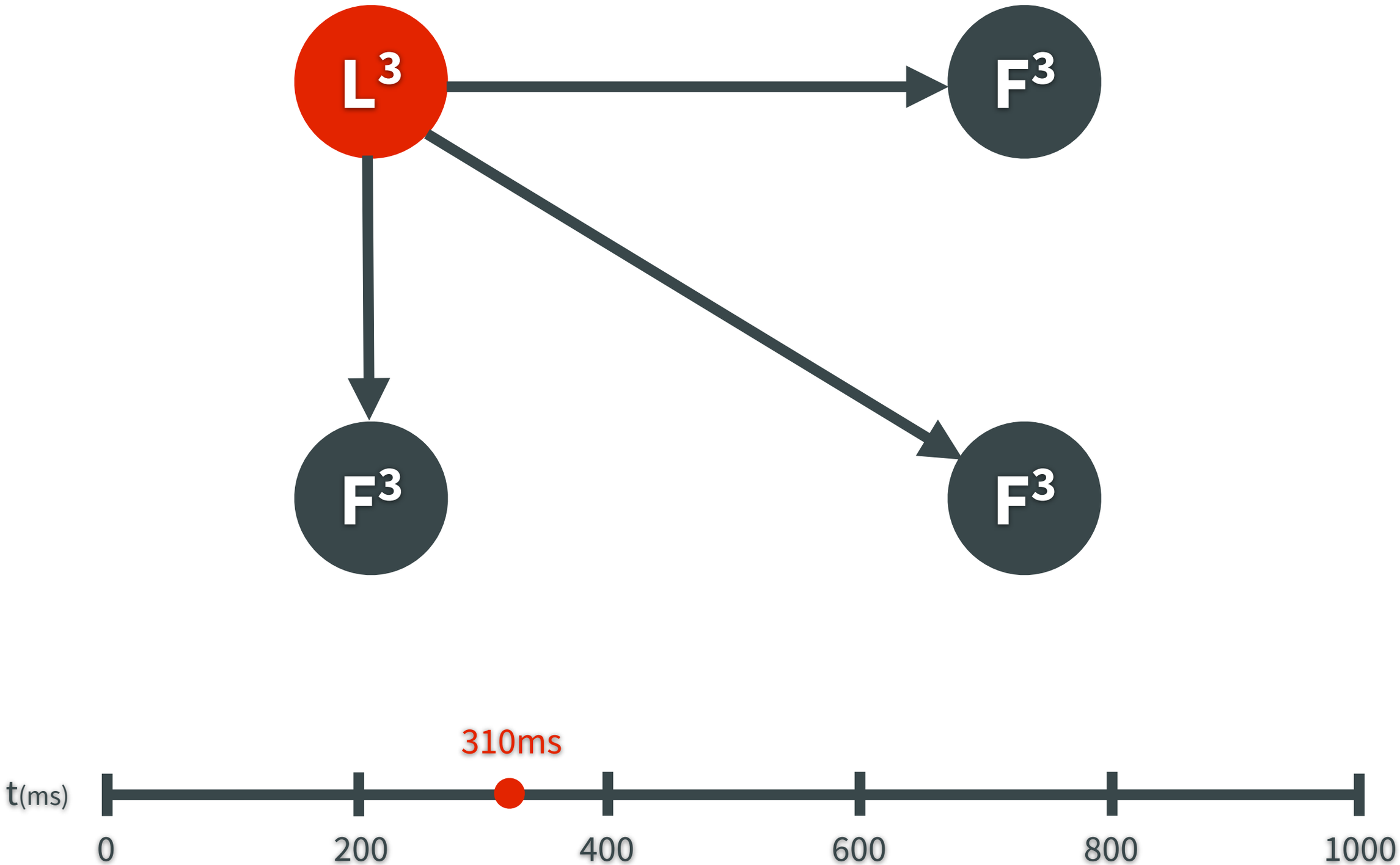
Second candidate doesn't know first candidate won the term and begins requesting votes



ONGARO, D., AND OUSTERHOUT, J. In search of an understandable consensus algorithm. In Proc ATC'14, USENIX Annual Technical Conference (2014), USENIX, May 2014

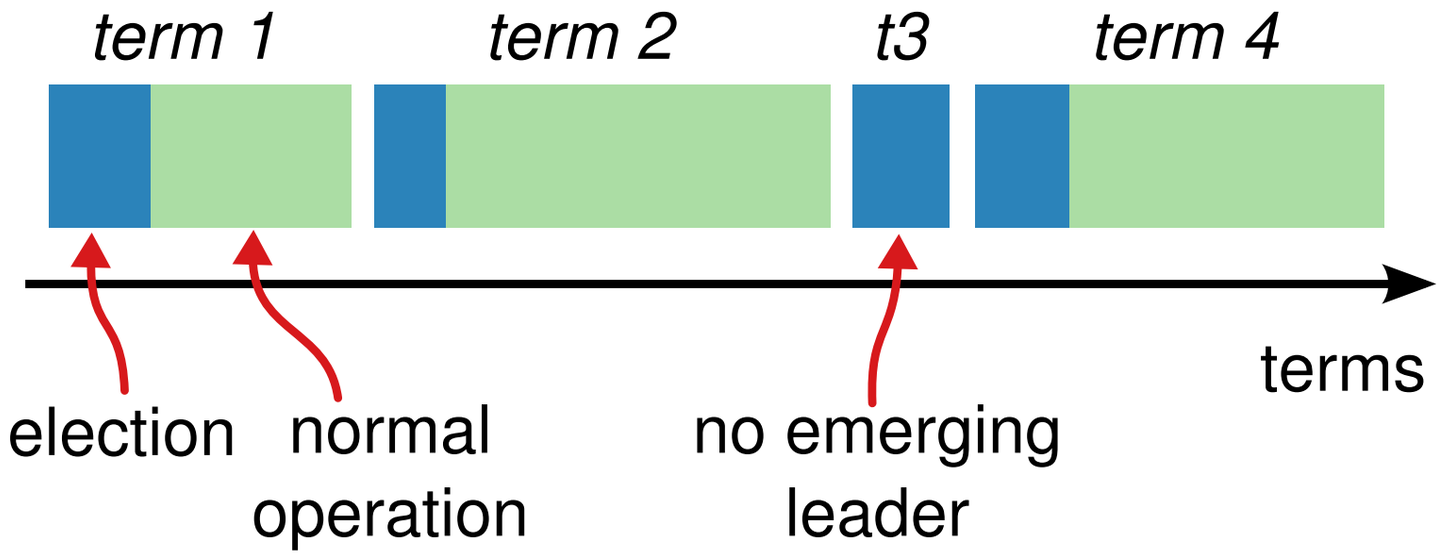
Leader Election

Leader notifies peers of election and other candidate steps down



ONGARO, D., AND OUSTERHOUT, J. In search of an understandable consensus algorithm. In Proc ATC'14, USENIX Annual Technical Conference (2014), USENIX, May 2014

Term

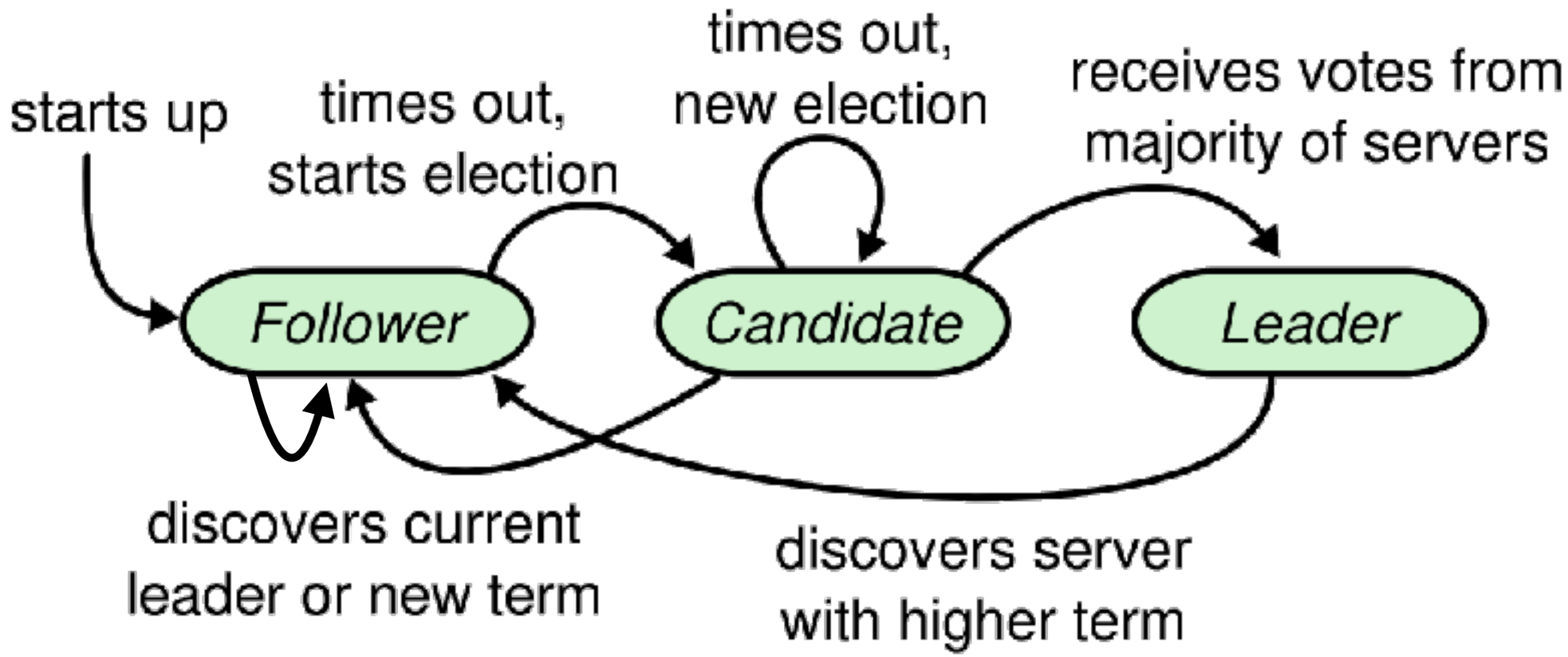


- Time is divided into terms
- Terms are strictly monotonic!

A follower can become candidate in any term.
A follower will vote for only one candidate per term!

ONGARO, D., AND OUSTERHOUT, J. In search of an understandable consensus algorithm. In Proc ATC'14, USENIX Annual Technical Conference (2014), USENIX, May 2014

Raft: Leader Election

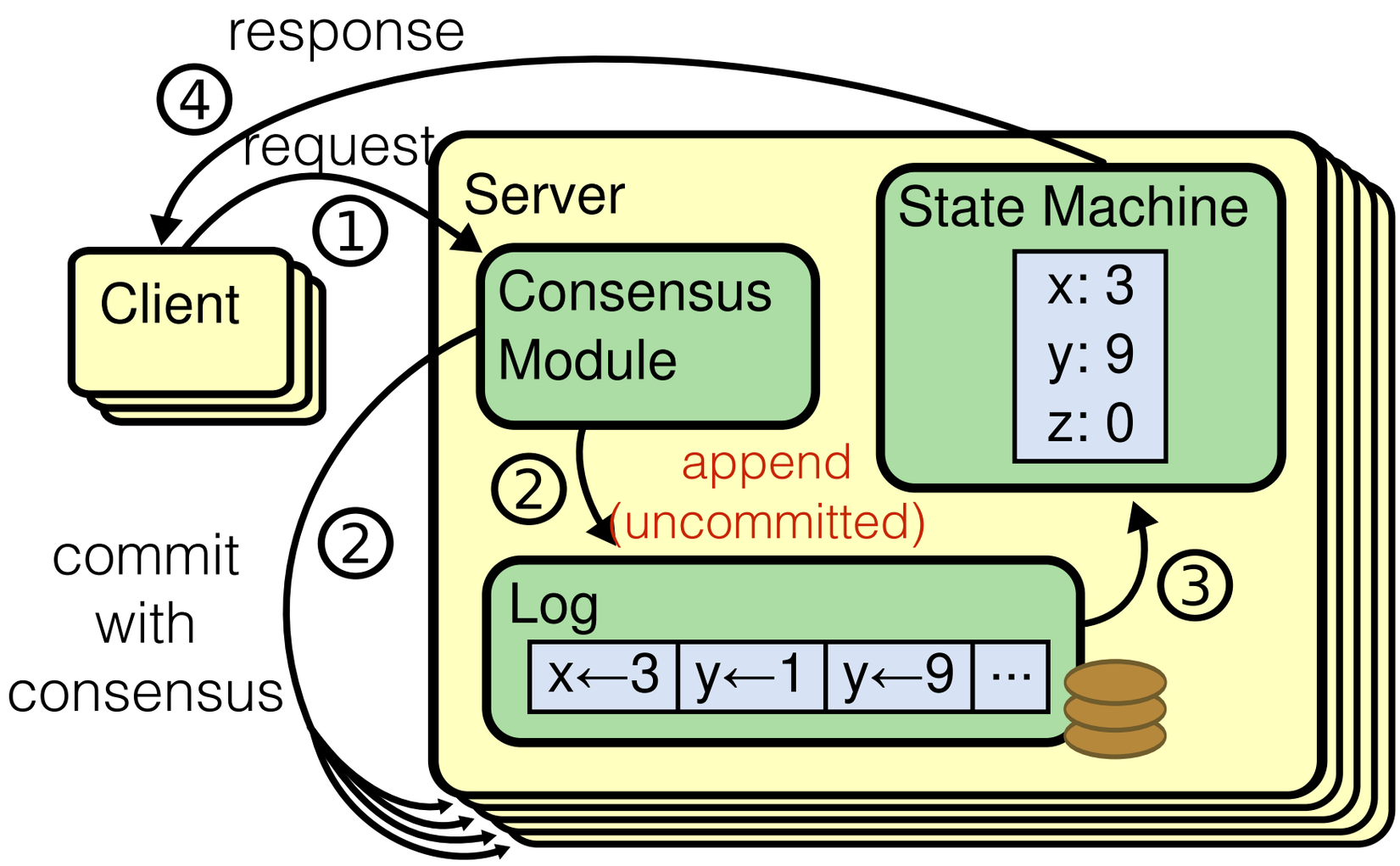


ONGARO, D., AND OUSTERHOUT, J. In search of an understandable consensus algorithm. In Proc ATC'14, USENIX Annual Technical Conference (2014), USENIX, May 2014

Log Replication

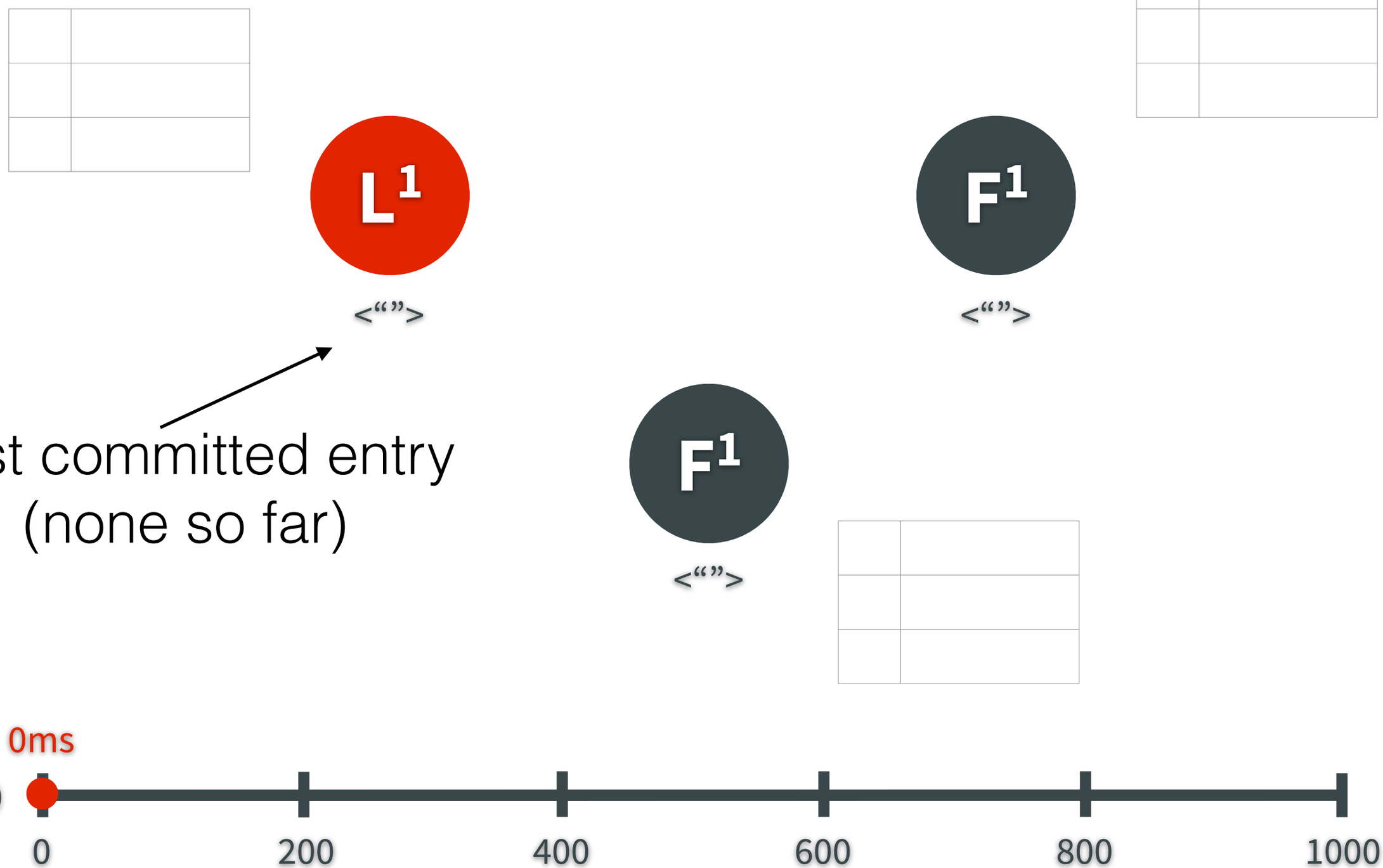
ONGARO, D., AND OUSTERHOUT, J. In search of an understandable consensus algorithm. In Proc ATC'14, USENIX Annual Technical Conference (2014), USENIX, May 2014

Architecture



ONGARO, D., AND OUSTERHOUT, J. In search of an understandable consensus algorithm. In Proc ATC'14, USENIX Annual Technical Conference (2014), USENIX, May 2014

Log Replication



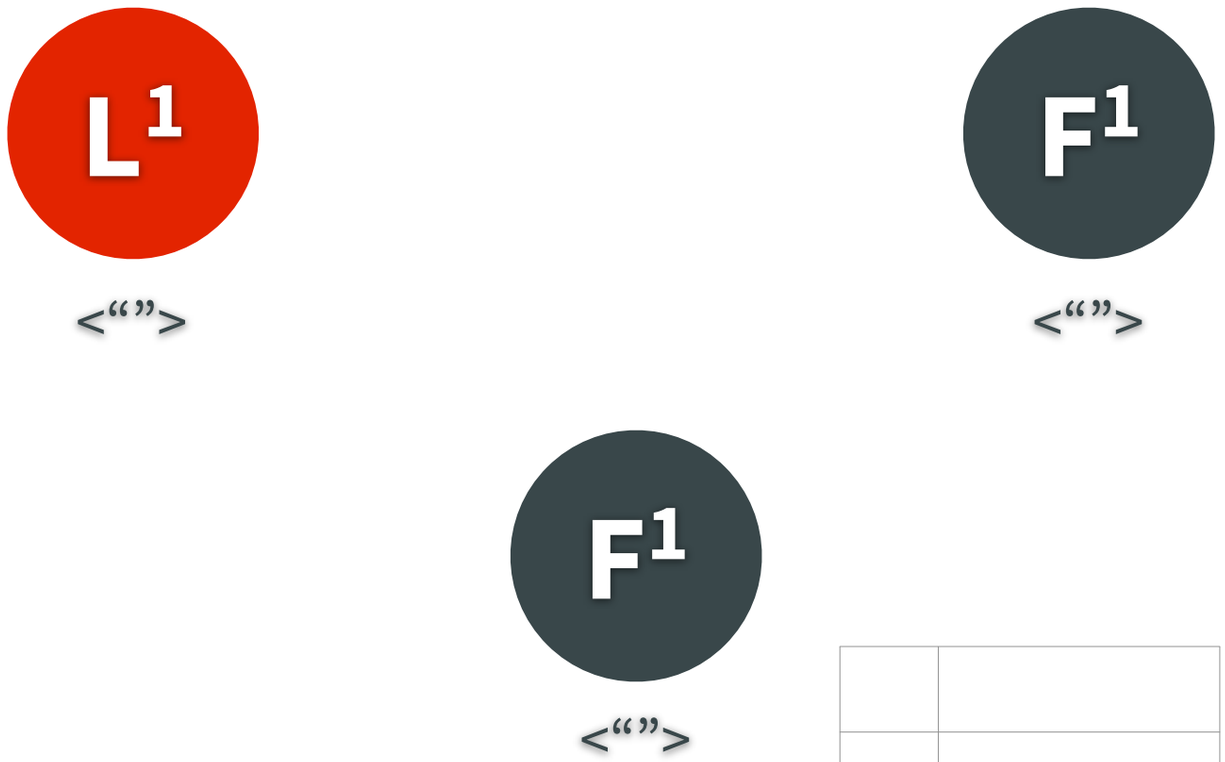
ONGARO, D., AND OUSTERHOUT, J. In search of an understandable consensus algorithm. In Proc ATC’14, USENIX Annual Technical Conference (2014), USENIX, May 2014

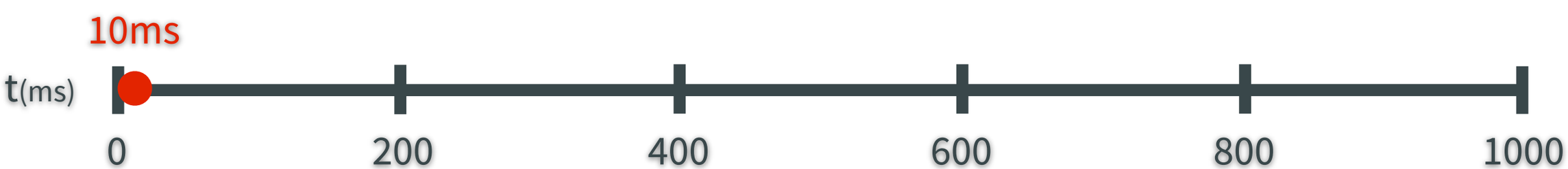
Log Replication

A new uncommitted log entry is added to the leader

1	“sally”

red: not yet committed





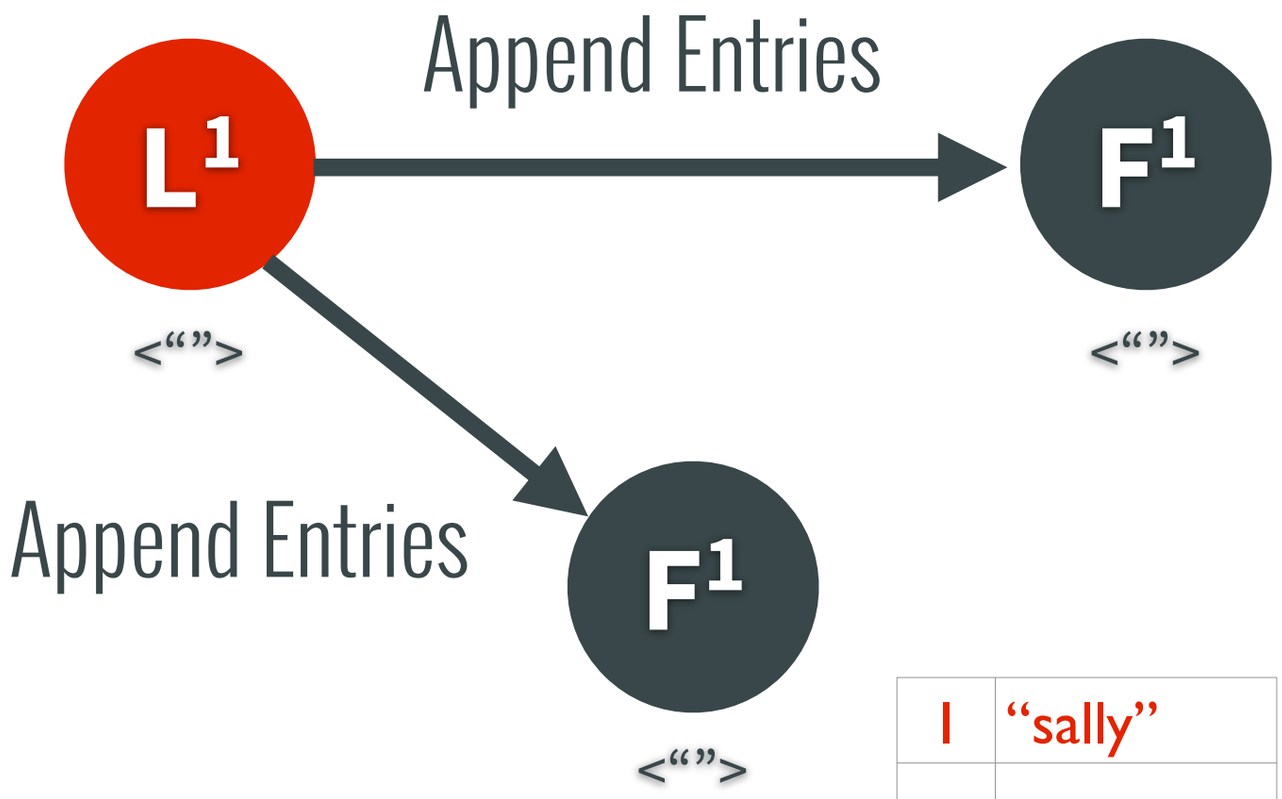
ONGARO, D., AND OUSTERHOUT, J. In search of an understandable consensus algorithm. In Proc ATC’14, USENIX Annual Technical Conference (2014), USENIX, May 2014

Log Replication

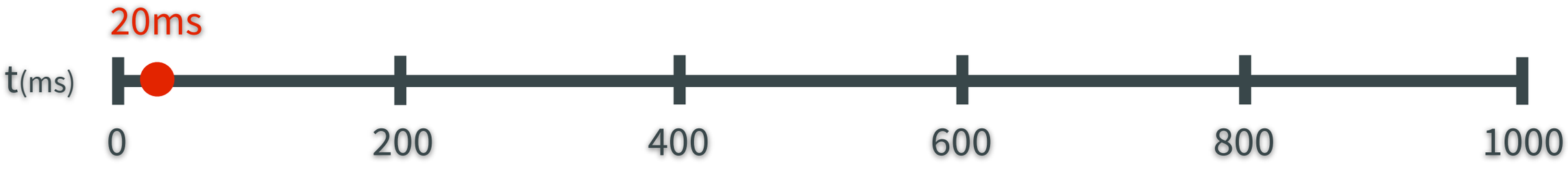
At the next heartbeat, the log entry is replicated to followers

1	“sally”

1	“sally”



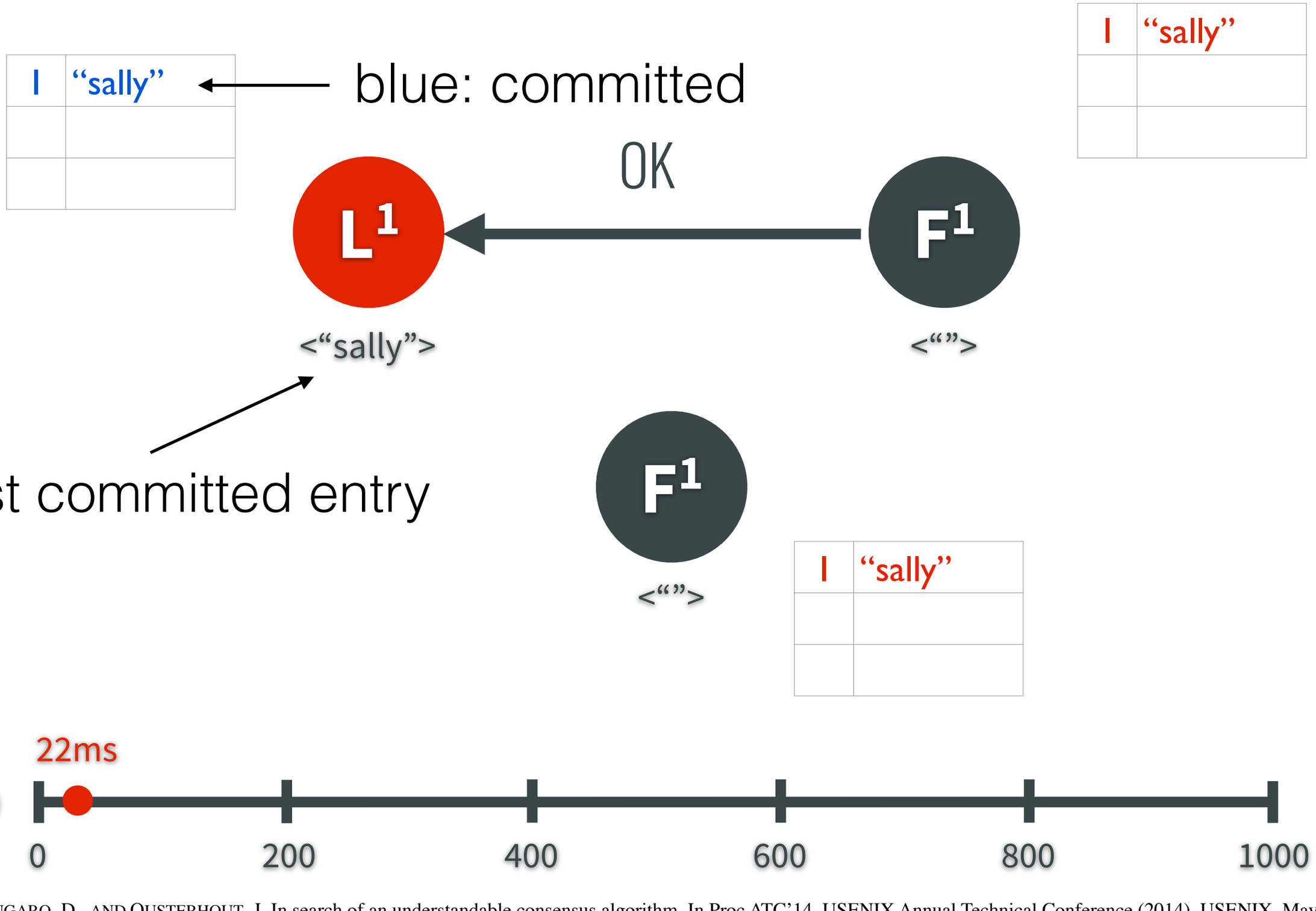
1	“sally”



ONGARO, D., AND OUSTERHOUT, J. In search of an understandable consensus algorithm. In Proc ATC’14, USENIX Annual Technical Conference (2014), USENIX, May 2014

Log Replication

a majority of nodes have ack'ed that they written log entry to disk

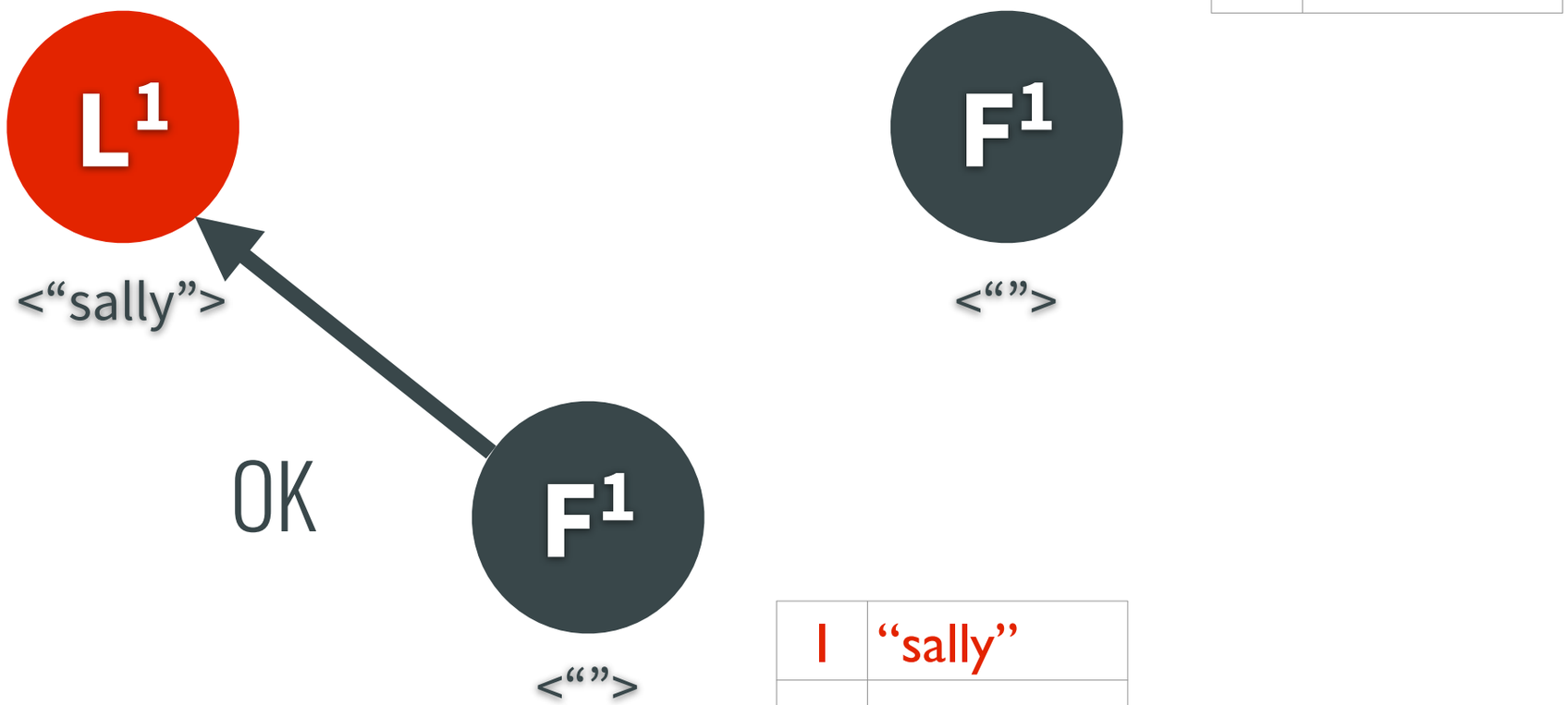


ONGARO, D., AND OUSTERHOUT, J. In search of an understandable consensus algorithm. In Proc ATC'14, USENIX Annual Technical Conference (2014), USENIX, May 2014

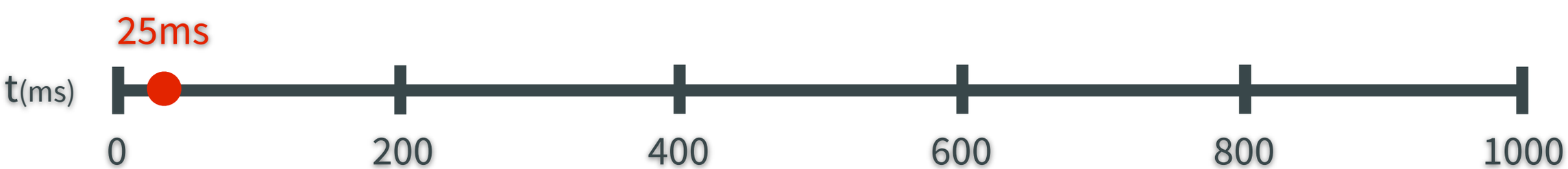
Log Replication

I	“sally”

I	“sally”



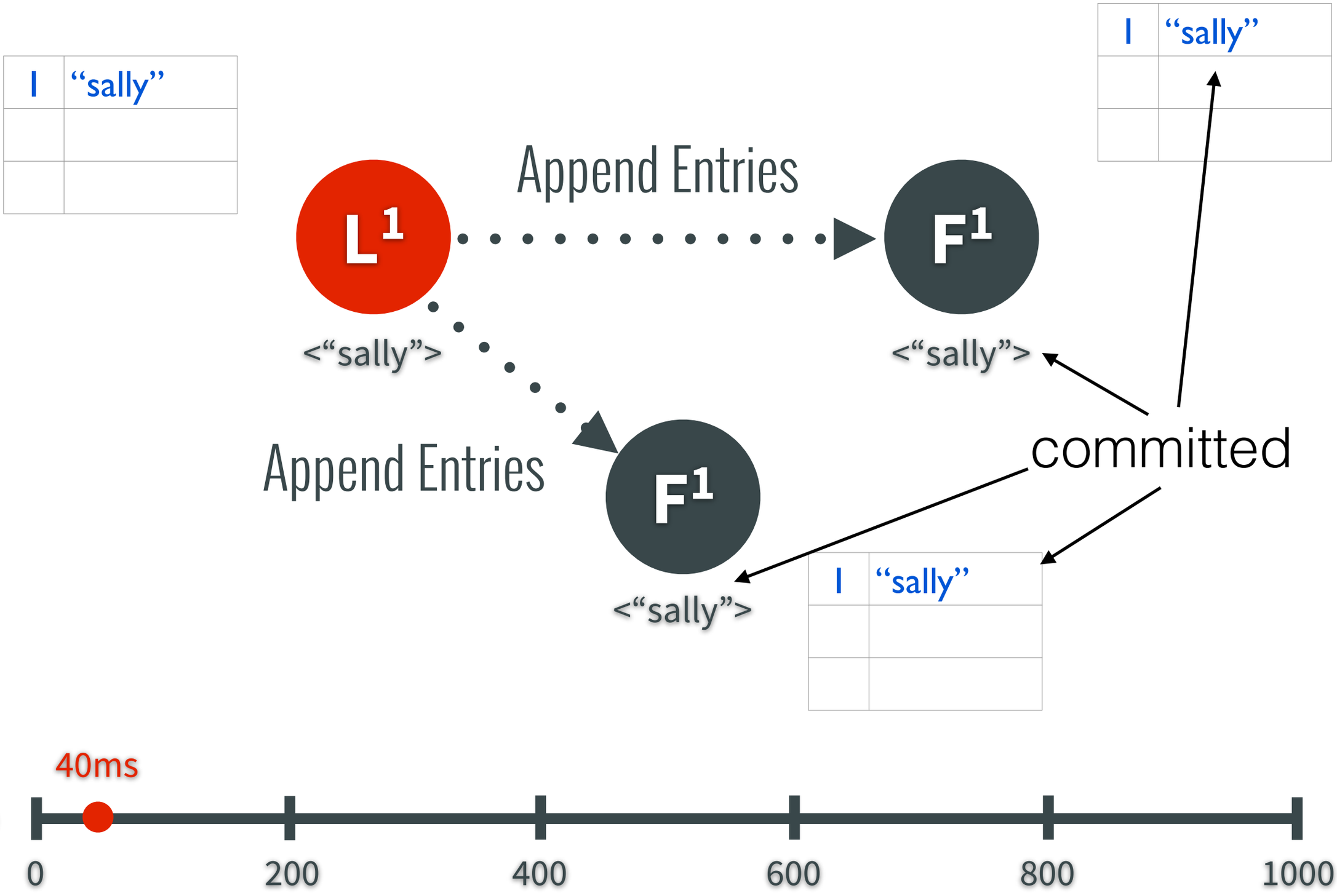
I	“sally”



ONGARO, D., AND OUSTERHOUT, J. In search of an understandable consensus algorithm. In Proc ATC’14, USENIX Annual Technical Conference (2014), USENIX, May 2014

Log Replication

At the next heartbeat, the leader notifies followers of updated committed entries



ONGARO, D., AND OUSTERHOUT, J. In search of an understandable consensus algorithm. In Proc ATC’14, USENIX Annual Technical Conference (2014), USENIX, May 2014

Log Replication

I	“sally”

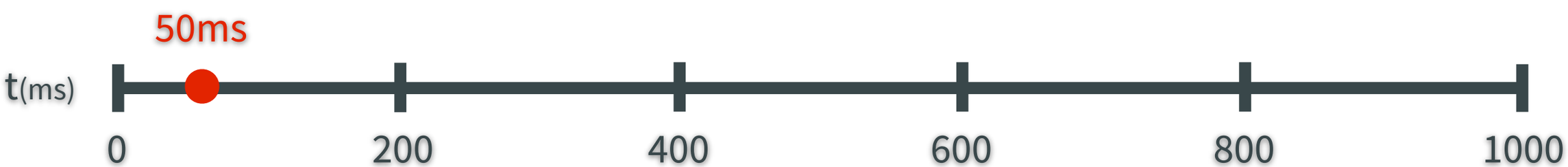
<“sally”>

<“sally”>

I	“sally”

<“sally”>

I	“sally”



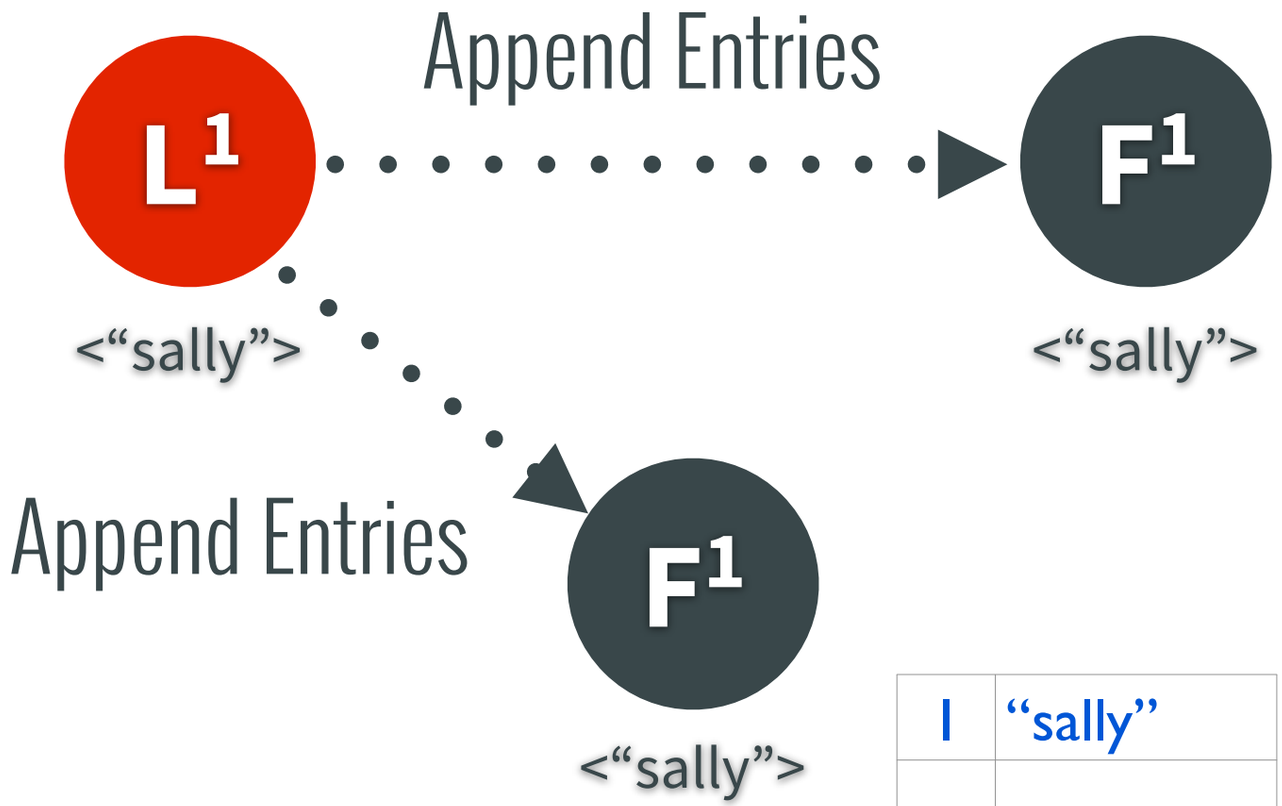
ONGARO, D., AND OUSTERHOUT, J. In search of an understandable consensus algorithm. In Proc ATC’14, USENIX Annual Technical Conference (2014), USENIX, May 2014

Log Replication

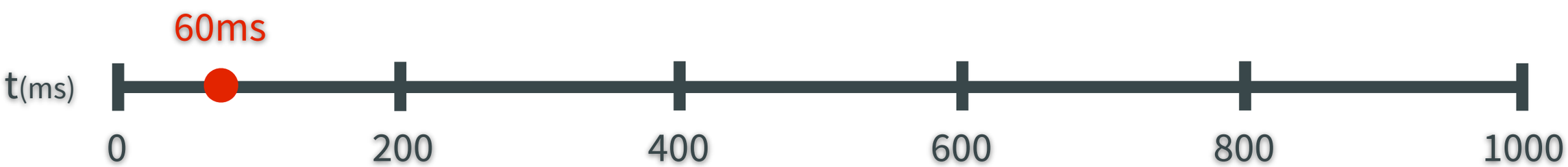
At the next heartbeat, no new log information is sent

I	“sally”

I	“sally”



I	“sally”



ONGARO, D., AND OUSTERHOUT, J. In search of an understandable consensus algorithm. In Proc ATC’14, USENIX Annual Technical Conference (2014), USENIX, May 2014

Log Replication

I	“sally”

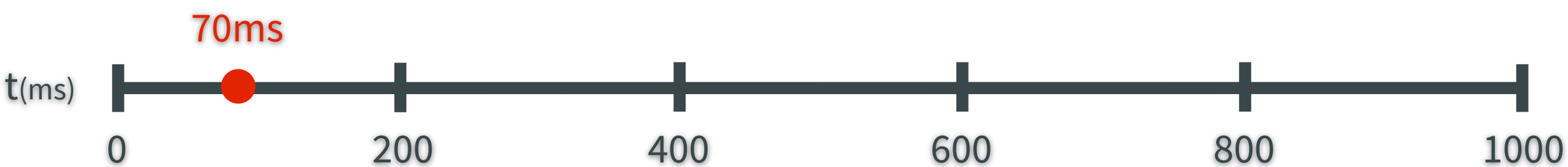
I	“sally”

<“sally”>

<“sally”>

<“sally”>

I	“sally”



ONGARO, D., AND OUSTERHOUT, J. In search of an understandable consensus algorithm. In Proc ATC’14, USENIX Annual Technical Conference (2014), USENIX, May 2014

Log Replication

A new uncommitted log entry is added to the leader

1	“sally”
2	“bob”

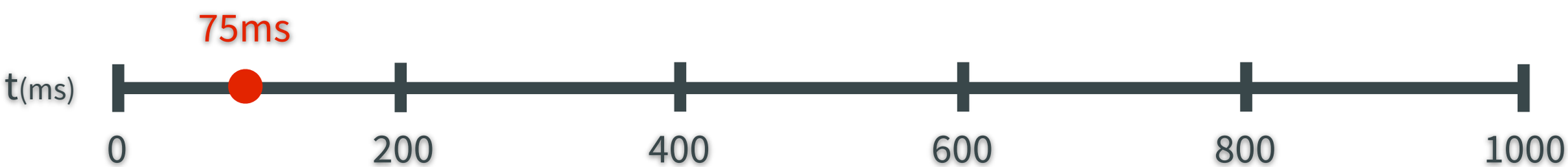
<“sally”>

1	“sally”

<“sally”>

<“sally”>

1	“sally”

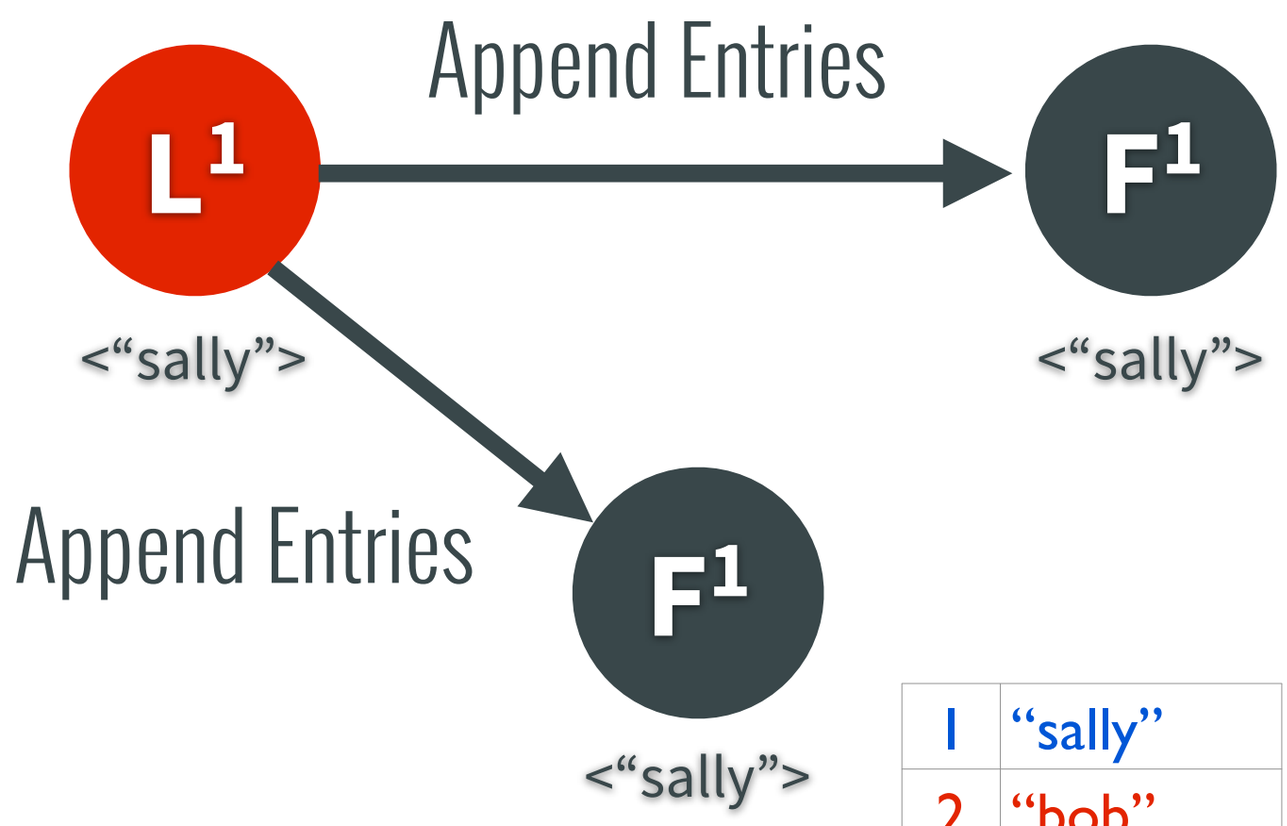


ONGARO, D., AND OUSTERHOUT, J. In search of an understandable consensus algorithm. In Proc ATC’14, USENIX Annual Technical Conference (2014), USENIX, May 2014

Log Replication

At the next heartbeat, the entry is replicated to the followers

1	“sally”
2	“bob”



1	“sally”
2	“bob”

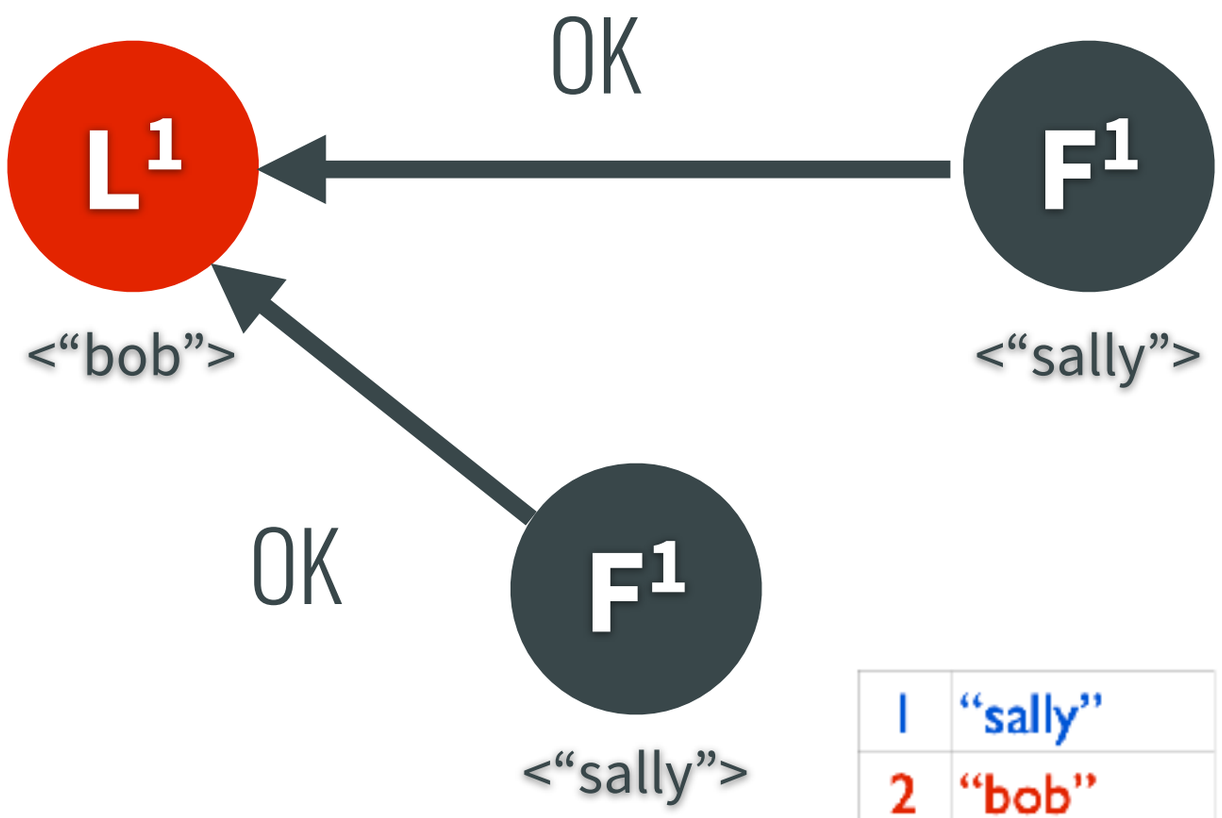
1	“sally”
2	“bob”

ONGARO, D., AND OUSTERHOUT, J. In search of an understandable consensus algorithm. In Proc ATC’14, USENIX Annual Technical Conference (2014), USENIX, May 2014

Log Replication

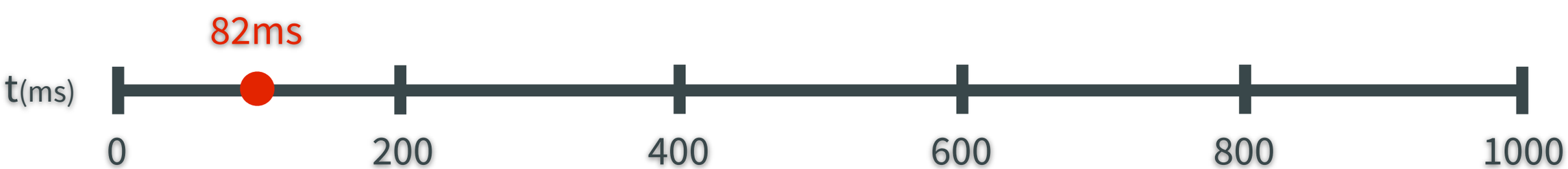
The entry is committed once the followers acknowledge the request

1	“sally”
2	“bob”



1	“sally”
2	“bob”

1	“sally”
2	“bob”

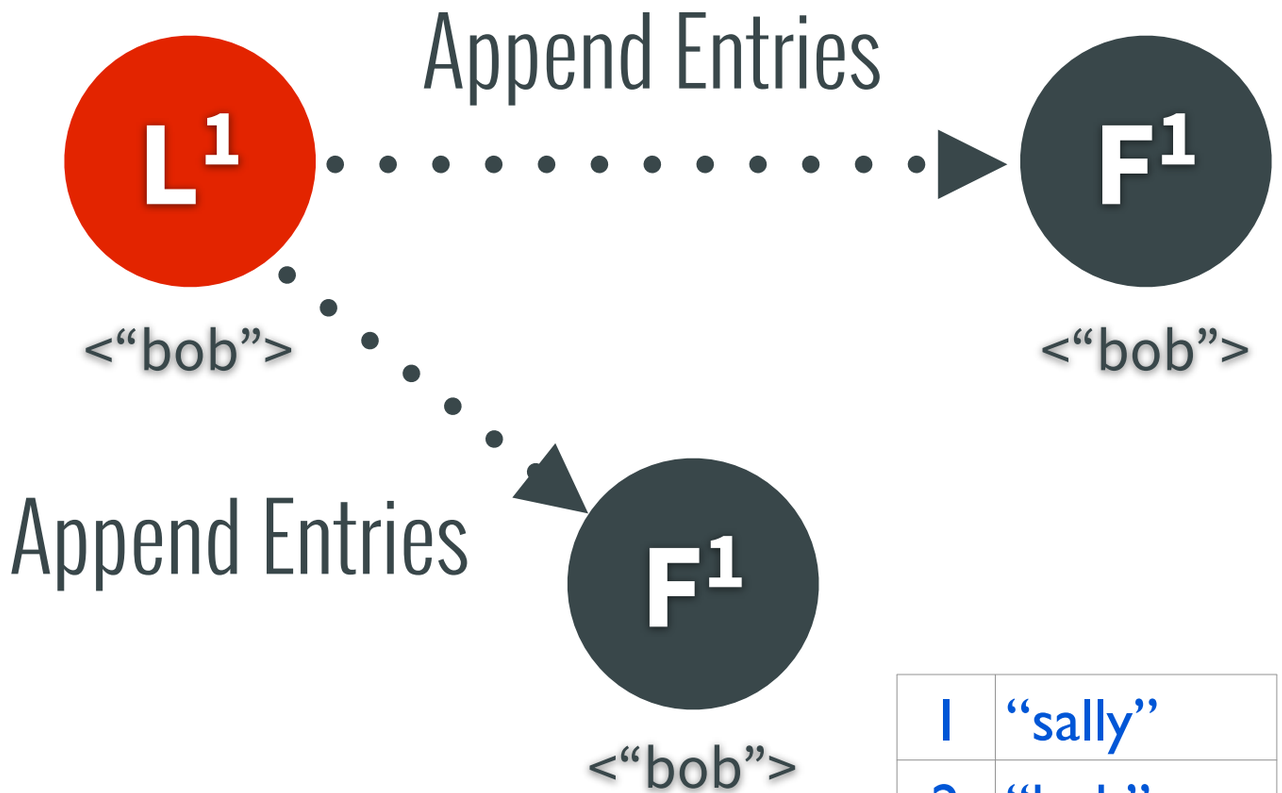


ONGARO, D., AND OUSTERHOUT, J. In search of an understandable consensus algorithm. In Proc ATC’14, USENIX Annual Technical Conference (2014), USENIX, May 2014

Log Replication

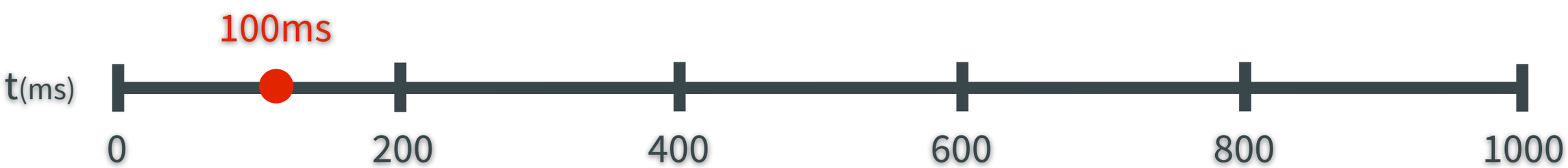
At the next heartbeat, the leader notifies the followers of the new committed entry

1	“sally”
2	“bob”



1	“sally”
2	“bob”

1	“sally”
2	“bob”

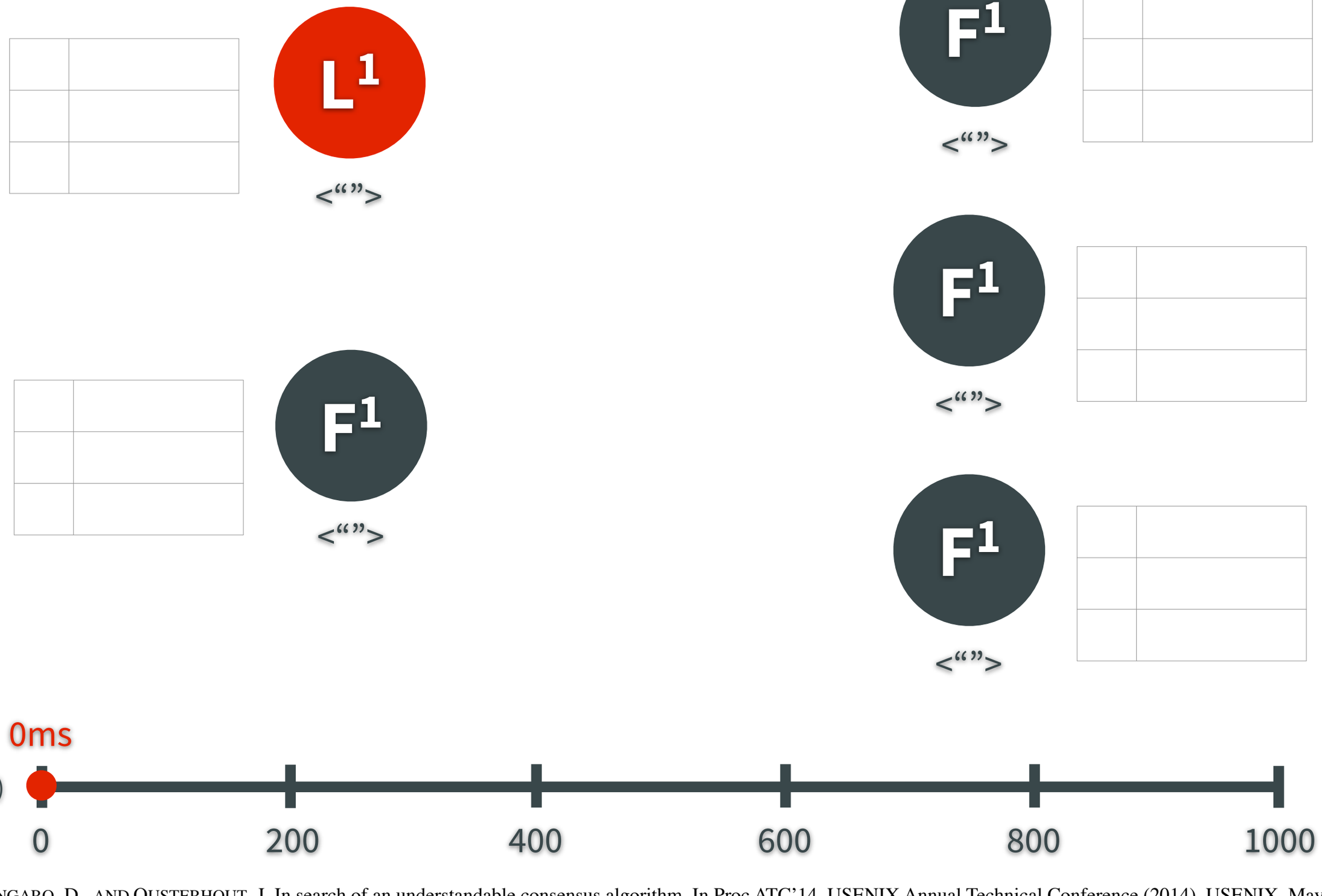


ONGARO, D., AND OUSTERHOUT, J. In search of an understandable consensus algorithm. In Proc ATC’14, USENIX Annual Technical Conference (2014), USENIX, May 2014

Log Replication
(with Network Partitions)

ONGARO, D., AND OUSTERHOUT, J. In search of an understandable consensus algorithm. In Proc ATC'14, USENIX Annual Technical Conference (2014), USENIX, May 2014

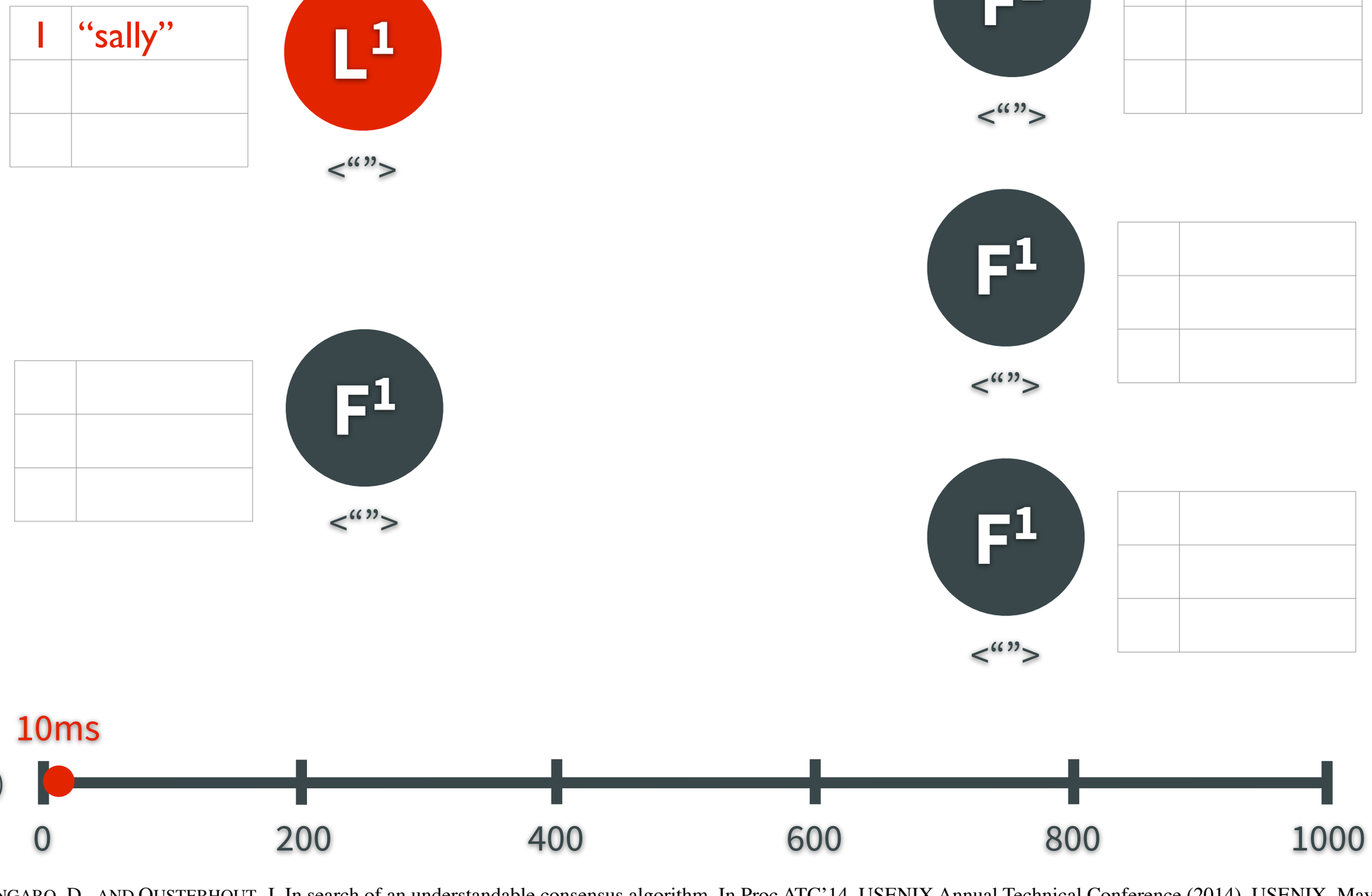
Log Replication



ONGARO, D., AND OUSTERHOUT, J. In search of an understandable consensus algorithm. In Proc ATC’14, USENIX Annual Technical Conference (2014), USENIX, May 2014

Log Replication

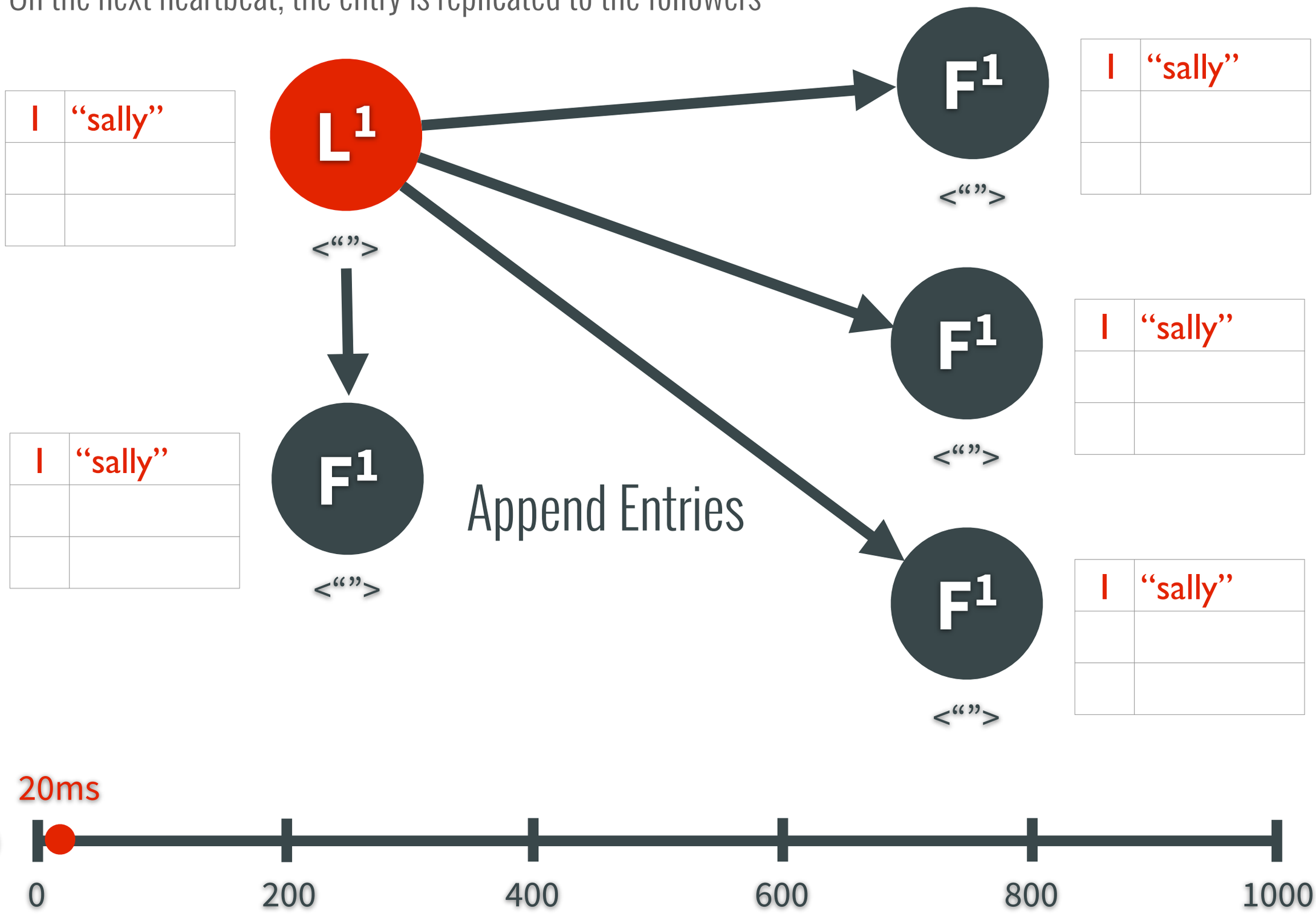
A new uncommitted log entry is added to the leader



ONGARO, D., AND OUSTERHOUT, J. In search of an understandable consensus algorithm. In Proc ATC’14, USENIX Annual Technical Conference (2014), USENIX, May 2014

Log Replication

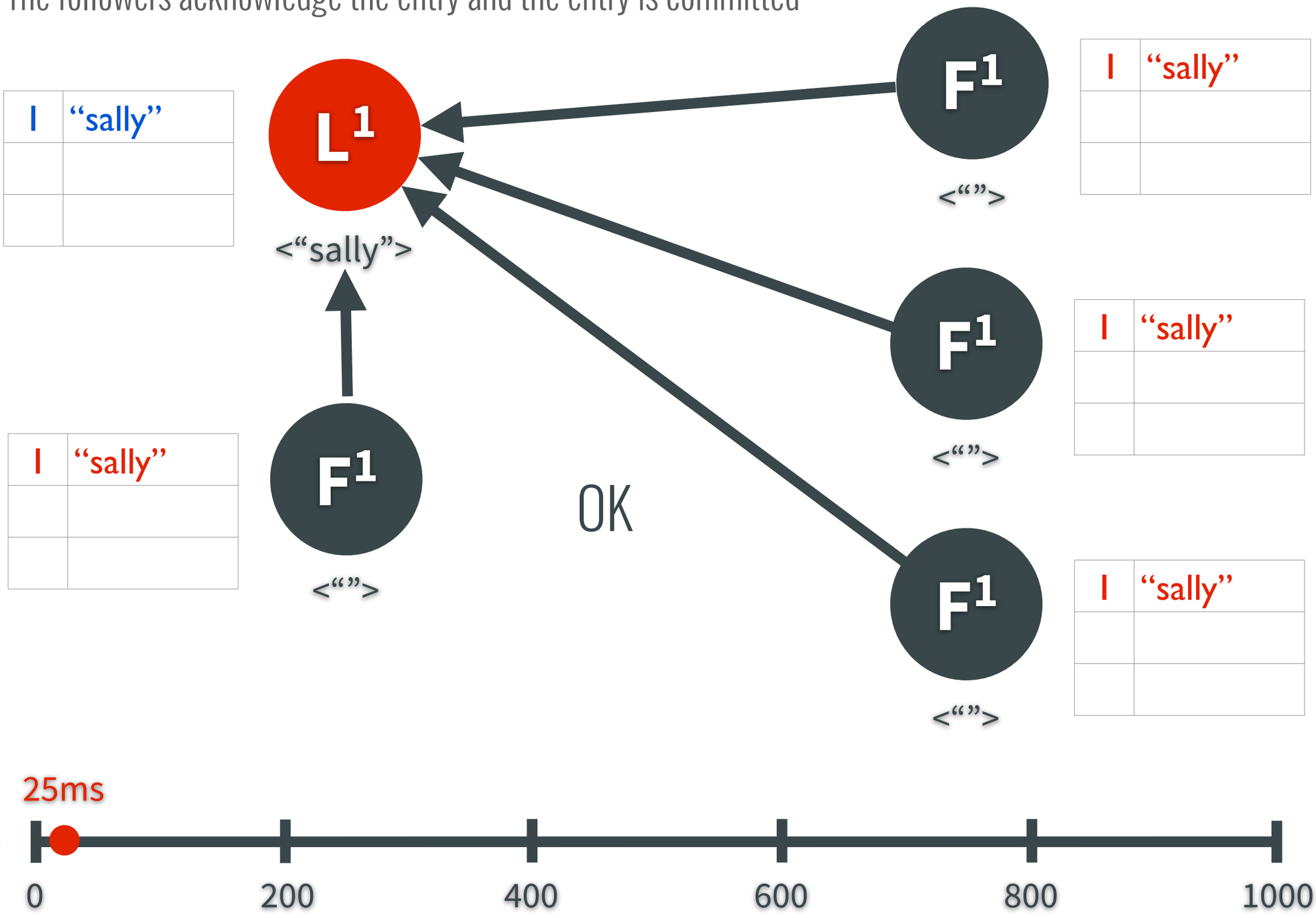
On the next heartbeat, the entry is replicated to the followers



ONGARO, D., AND OUSTERHOUT, J. In search of an understandable consensus algorithm. In Proc ATC’14, USENIX Annual Technical Conference (2014), USENIX, May 2014

Log Replication

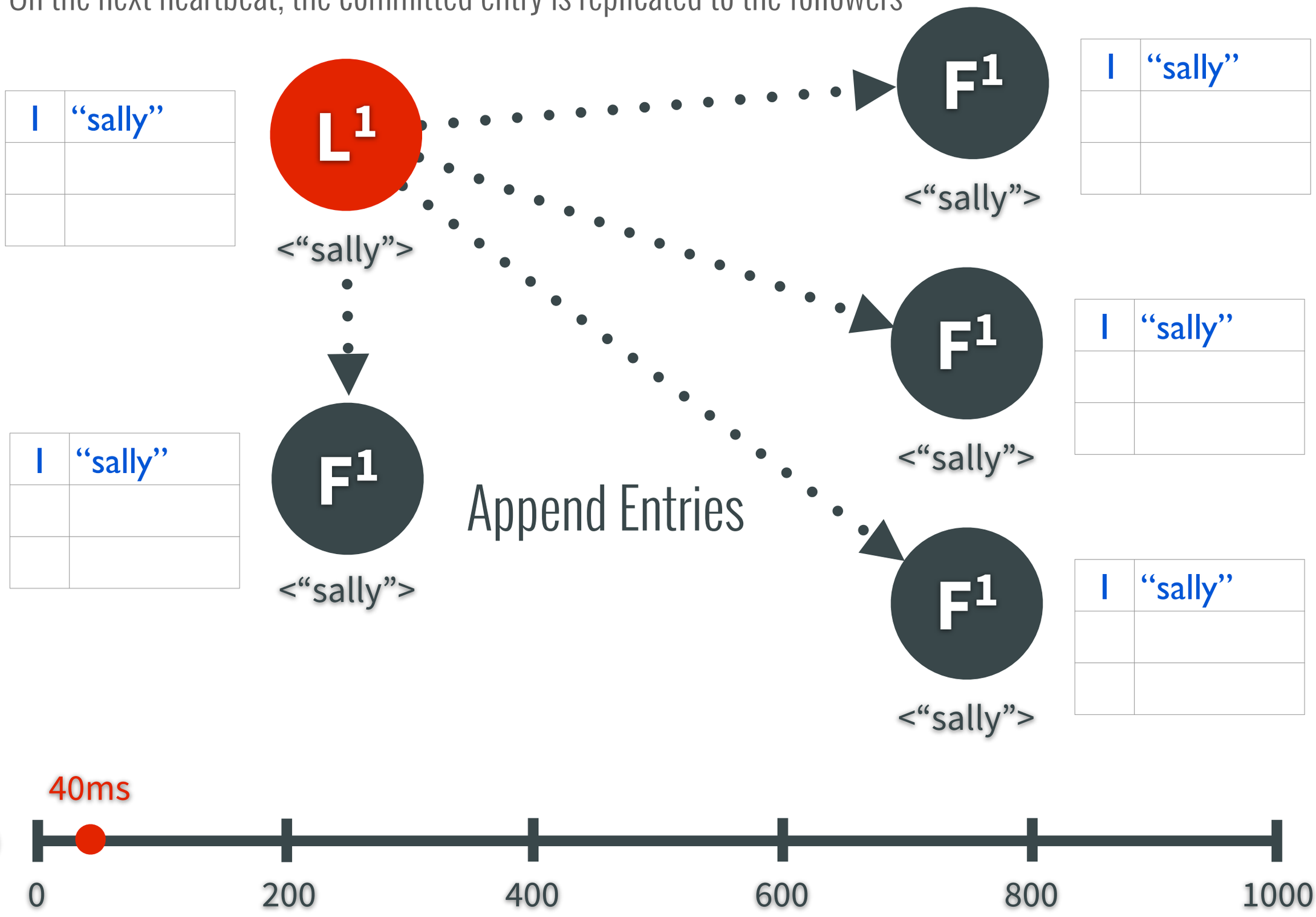
The followers acknowledge the entry and the entry is committed



ONGARO, D., AND OUSTERHOUT, J. In search of an understandable consensus algorithm. In Proc ATC'14, USENIX Annual Technical Conference (2014), USENIX, May 2014

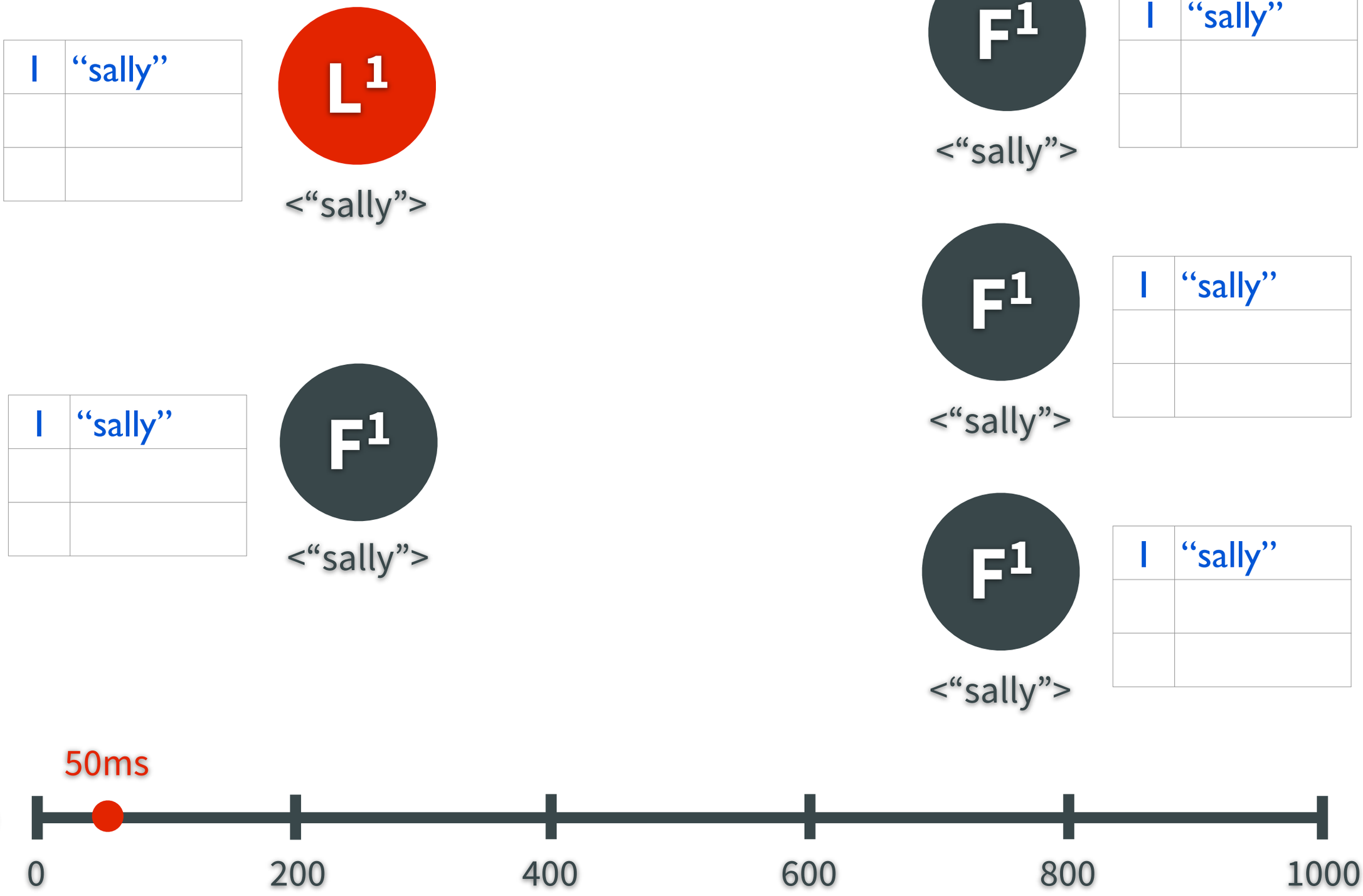
Log Replication

On the next heartbeat, the committed entry is replicated to the followers



ONGARO, D., AND OUSTERHOUT, J. In search of an understandable consensus algorithm. In Proc ATC’14, USENIX Annual Technical Conference (2014), USENIX, May 2014

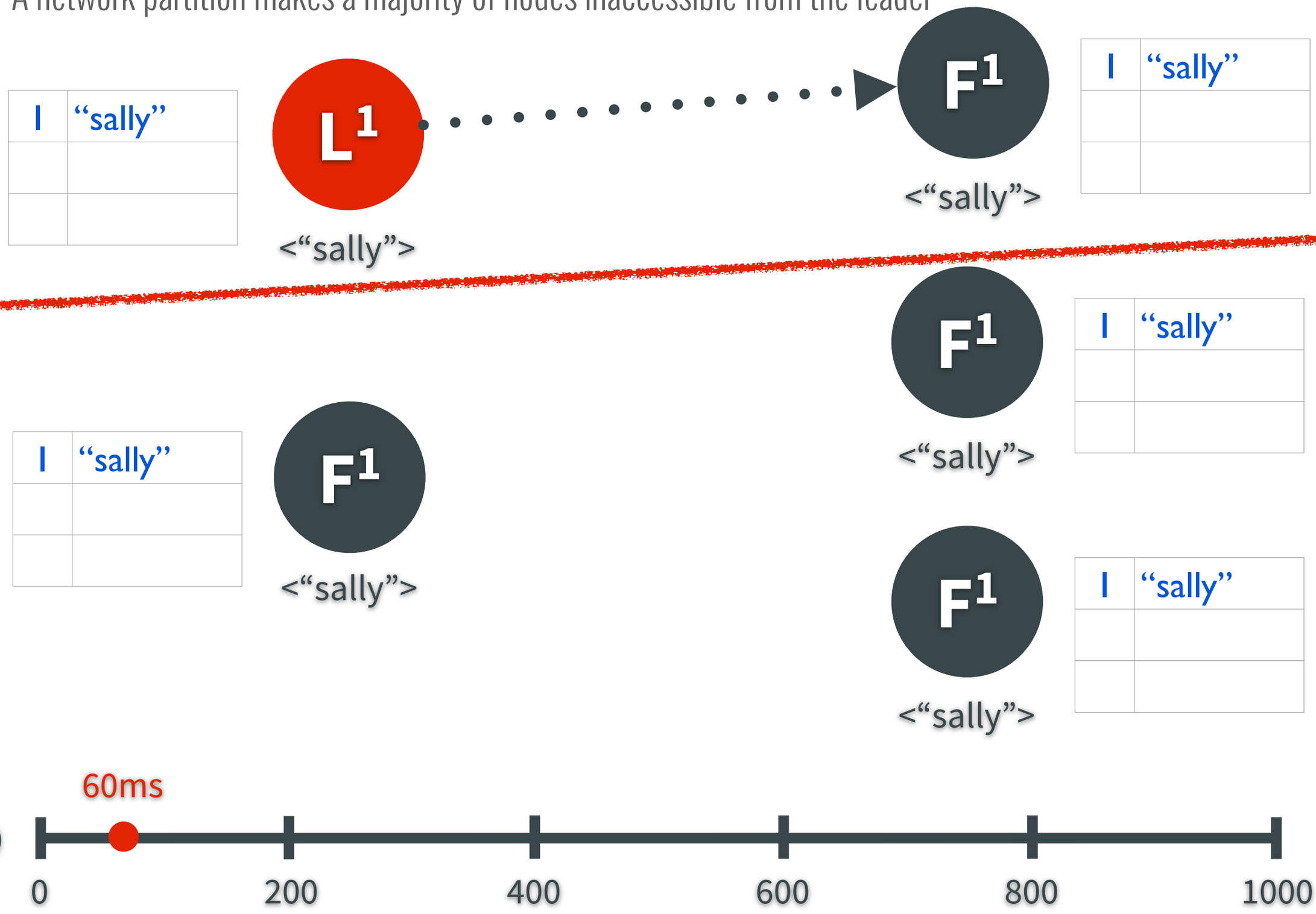
Log Replication



ONGARO, D., AND OUSTERHOUT, J. In search of an understandable consensus algorithm. In Proc ATC'14, USENIX Annual Technical Conference (2014), USENIX, May 2014

Log Replication

A network partition makes a majority of nodes inaccessible from the leader



ONGARO, D., AND OUSTERHOUT, J. In search of an understandable consensus algorithm. In Proc ATC'14, USENIX Annual Technical Conference (2014), USENIX, May 2014

Log Replication

A new log entry is added to the leader

1	“sally”
2	“bob”

L1

<“sally”>

F1

<“sally”>

1	“sally”

1	“sally”

F1

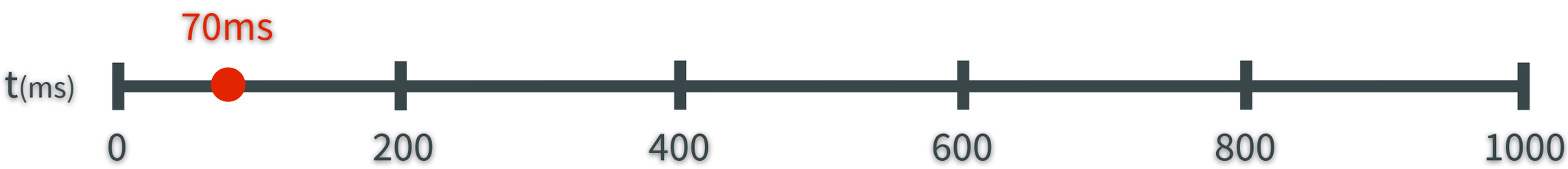
<“sally”>

1	“sally”

F1

<“sally”>

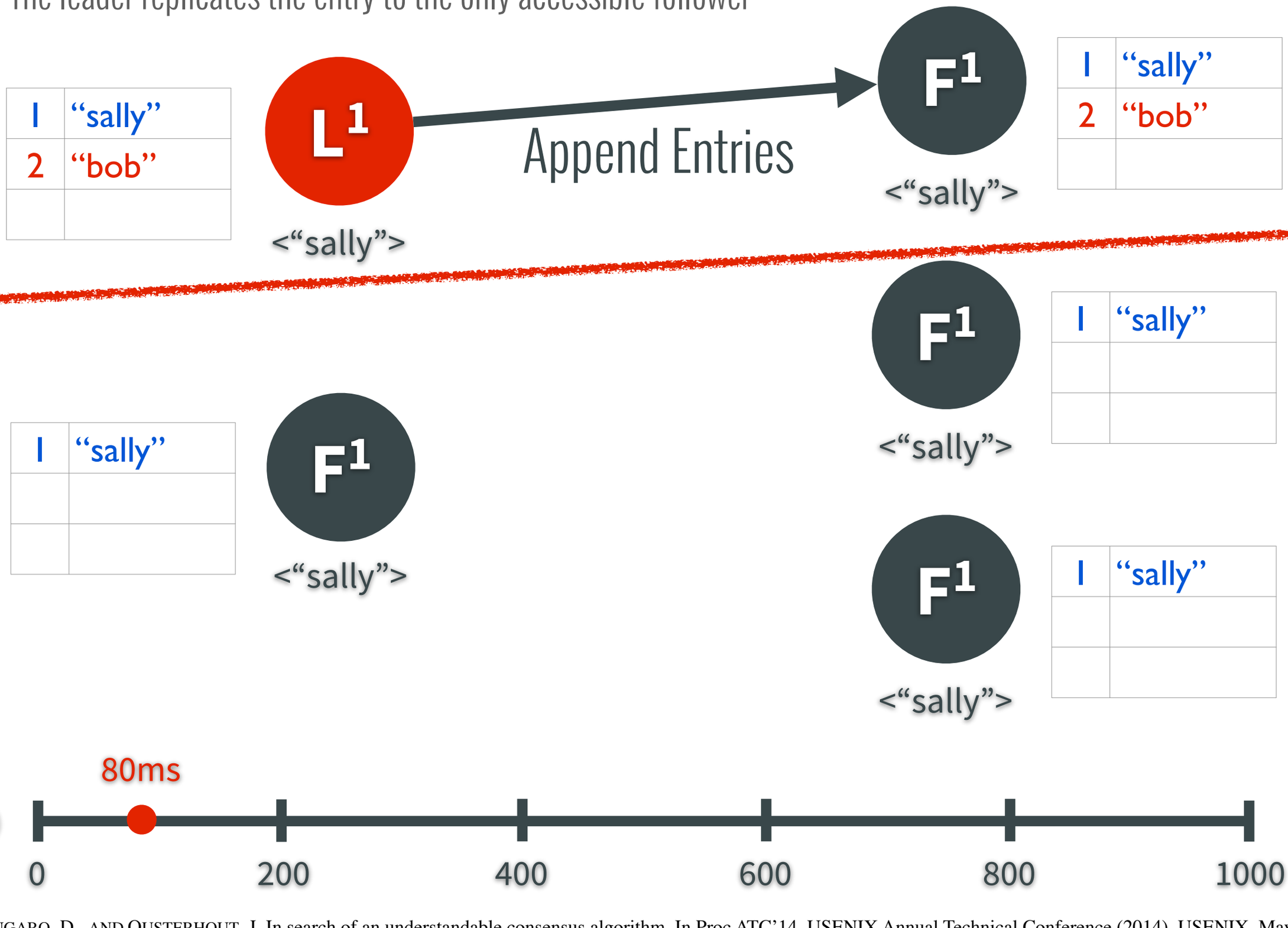
1	“sally”



ONGARO, D., AND OUSTERHOUT, J. In search of an understandable consensus algorithm. In Proc ATC’14, USENIX Annual Technical Conference (2014), USENIX, May 2014

Log Replication

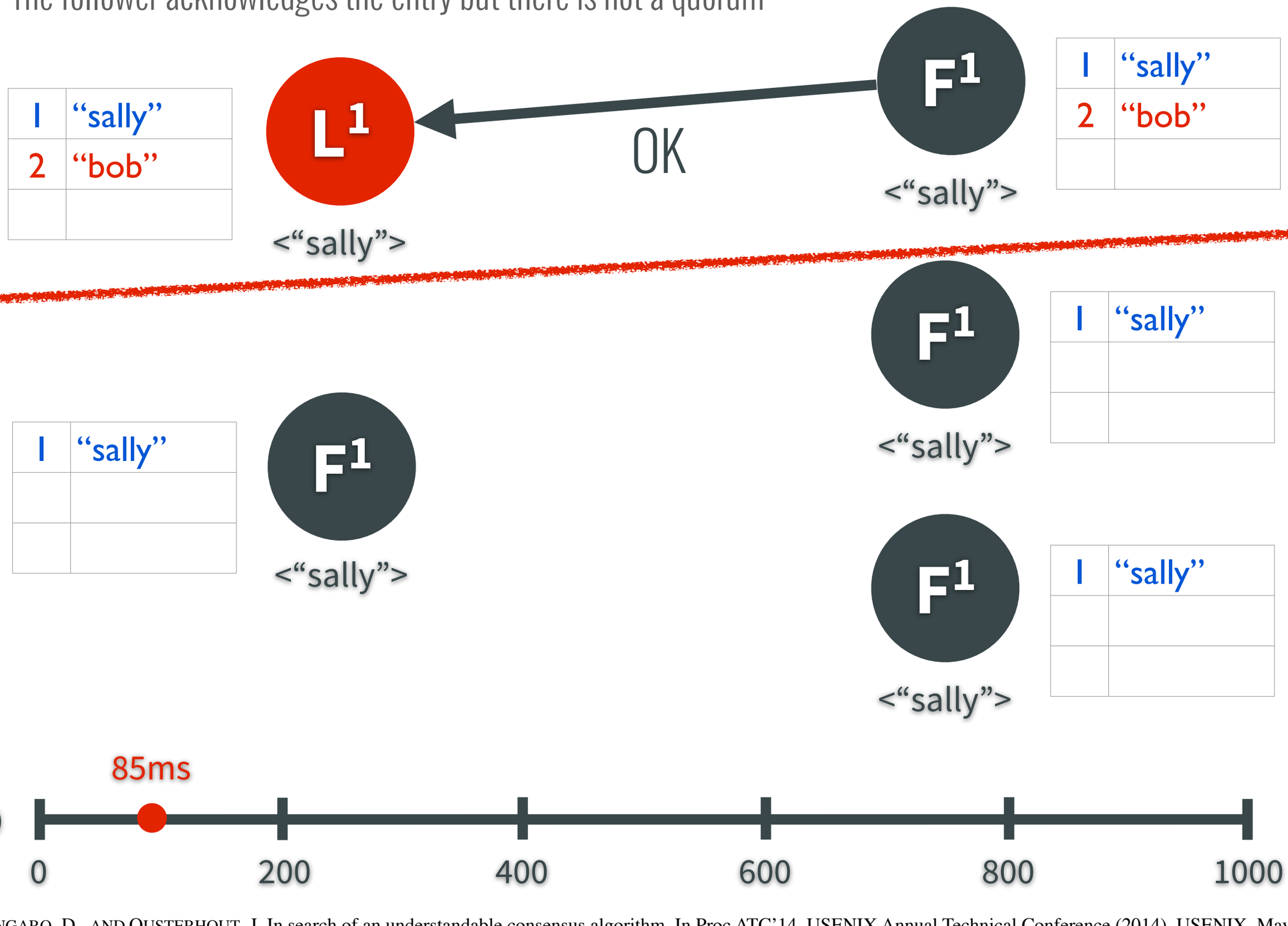
The leader replicates the entry to the only accessible follower



ONGARO, D., AND OUSTERHOUT, J. In search of an understandable consensus algorithm. In Proc ATC’14, USENIX Annual Technical Conference (2014), USENIX, May 2014

Log Replication

The follower acknowledges the entry but there is not a quorum



ONGARO, D., AND OUSTERHOUT, J. In search of an understandable consensus algorithm. In Proc ATC’14, USENIX Annual Technical Conference (2014), USENIX, May 2014

Log Replication

1	“sally”
2	“bob”

L1

<“sally”>

F1

<“sally”>

1	“sally”
2	“bob”

1	“sally”

C

<“sally”>

Request Vote

F1

<“sally”>

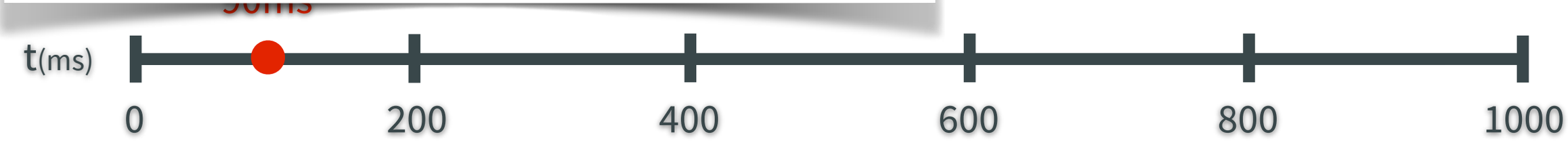
1	“sally”

F1

<“sally”>

1	“sally”

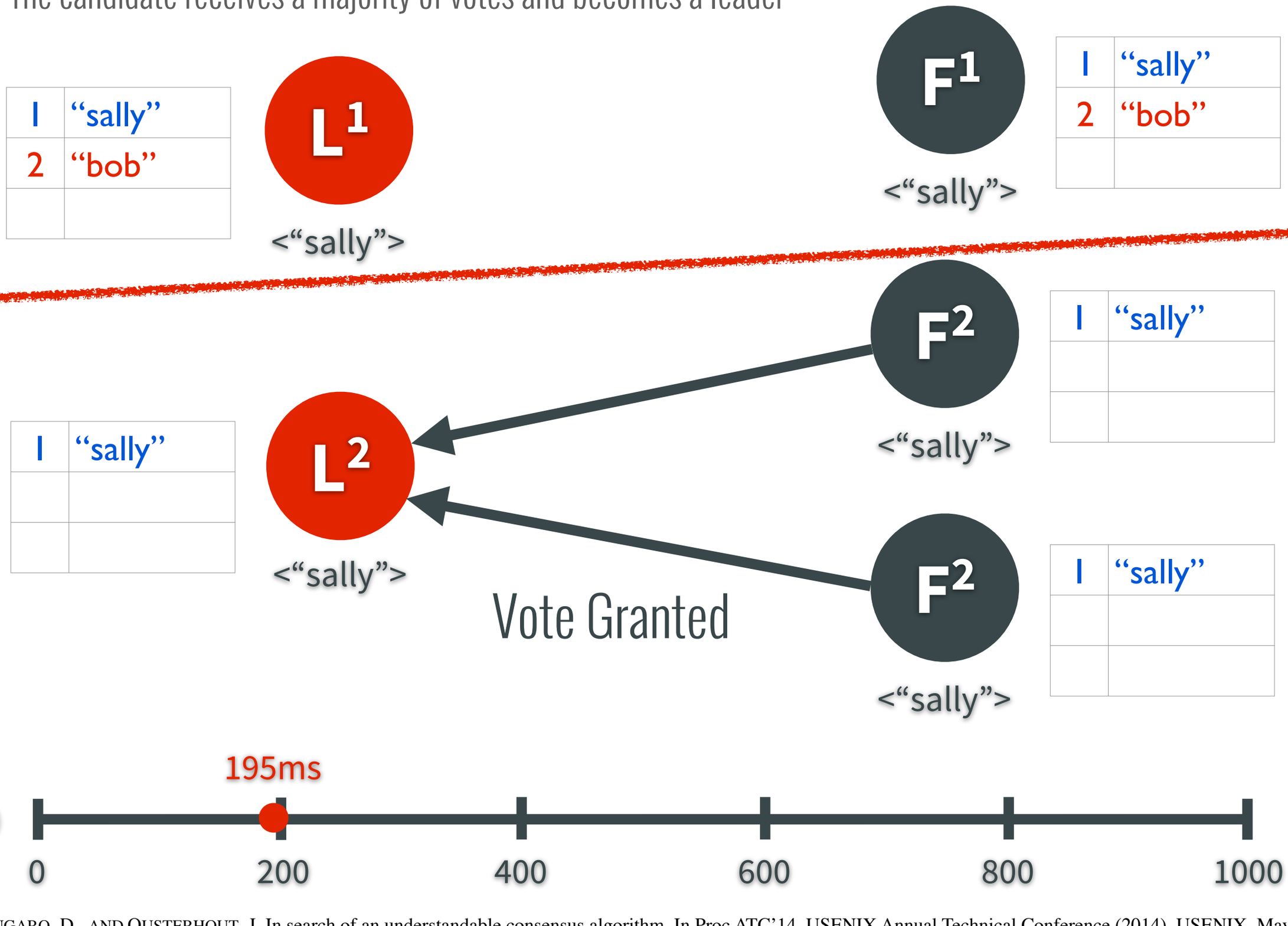
F only supports election if C’s log is at least as up-to-date



ONGARO, D., AND OUSTERHOUT, J. In search of an understandable consensus algorithm. In Proc ATC’14, USENIX Annual Technical Conference (2014), USENIX, May 2014

Log Replication

The candidate receives a majority of votes and becomes a leader



ONGARO, D., AND OUSTERHOUT, J. In search of an understandable consensus algorithm. In Proc ATC'14, USENIX Annual Technical Conference (2014), USENIX, May 2014

Log Replication

1	“sally”
2	“bob”

L¹

<“sally”>

F¹

<“sally”>

1	“sally”
2	“bob”

1	“sally”

L²

<“sally”>

F²

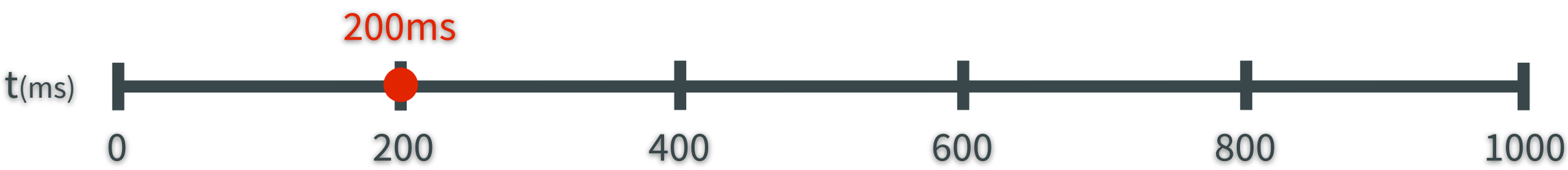
<“sally”>

1	“sally”

F²

<“sally”>

1	“sally”



ONGARO, D., AND OUSTERHOUT, J. In search of an understandable consensus algorithm. In Proc ATC’14, USENIX Annual Technical Conference (2014), USENIX, May 2014

Log Replication

A log entry is added to the new leader

1	“sally”
2	“bob”

L¹

<“sally”>

F¹

<“sally”>

1	“sally”
2	“bob”

1	“sally”
2	“tom”

L²

<“sally”>

F²

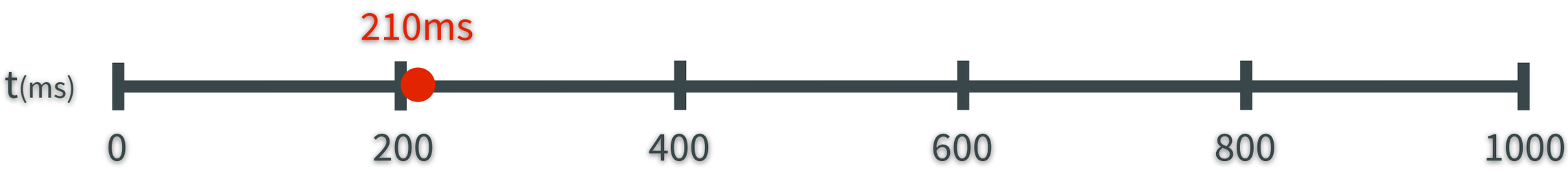
<“sally”>

1	“sally”

F²

<“sally”>

1	“sally”



ONGARO, D., AND OUSTERHOUT, J. In search of an understandable consensus algorithm. In Proc ATC’14, USENIX Annual Technical Conference (2014), USENIX, May 2014

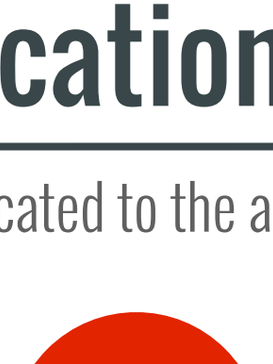
Log Replication

The log entry is replicated to the accessible followers

1	“sally”
2	“bob”

1	“sally”
2	“bob”

1	“sally”
2	“tom”



1	“sally”
2	“tom”

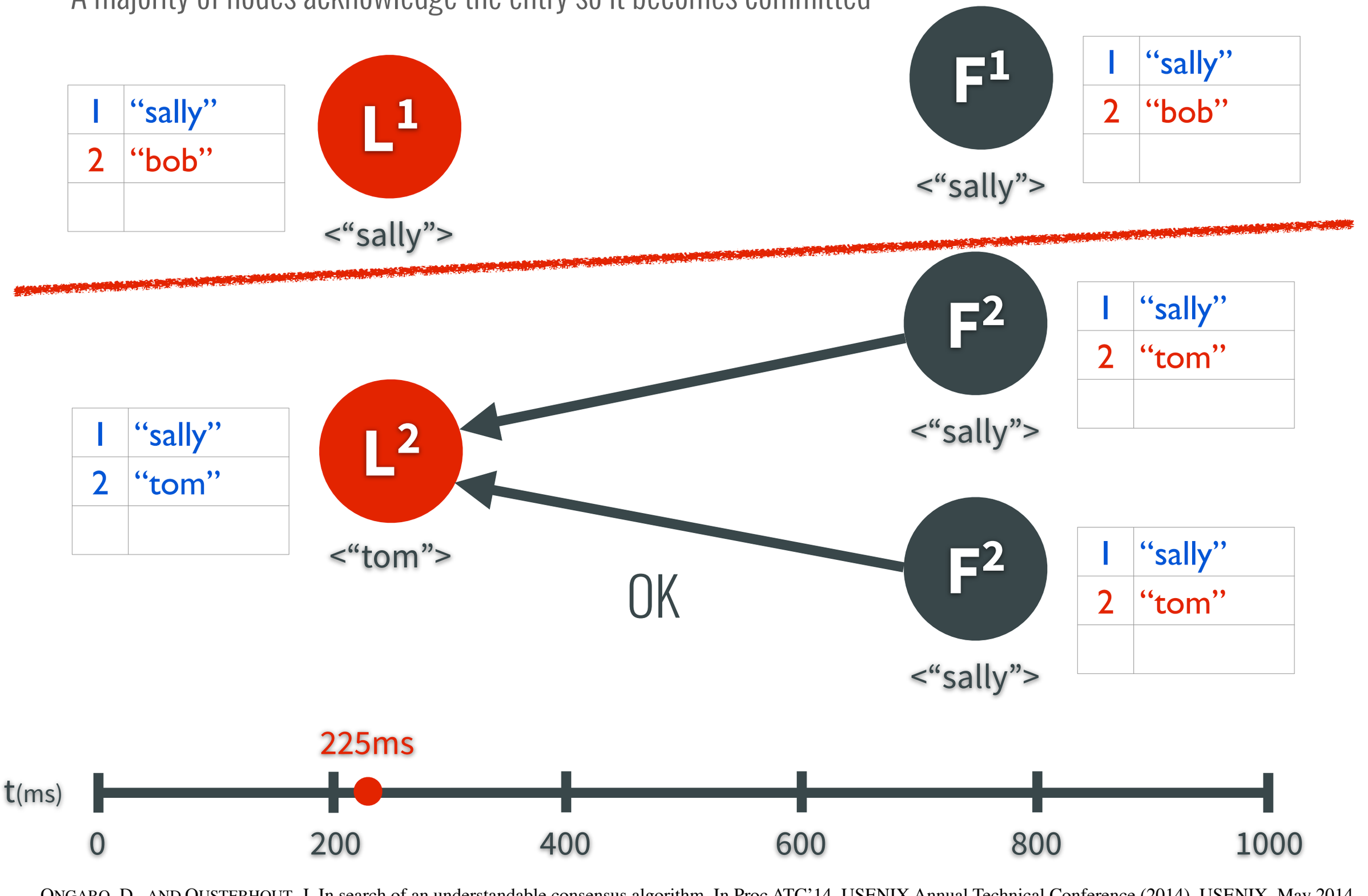
1	“sally”
2	“tom”

Append Entries

ONGARO, D., AND OUSTERHOUT, J. In search of an understandable consensus algorithm. In Proc ATC’14, USENIX Annual Technical Conference (2014), USENIX, May 2014

Log Replication

A majority of nodes acknowledge the entry so it becomes committed



ONGARO, D., AND OUSTERHOUT, J. In search of an understandable consensus algorithm. In Proc ATC'14, USENIX Annual Technical Conference (2014), USENIX, May 2014

Questions

- What if L2 crashes now?
  - how do we know if „tom“ was committed or not?
- What if L1 also crashes?
  - how do we decide if „bob“ or „tom“ was committed?

Log Replication

A majority of nodes acknowledge the entry so it becomes committed

1	“sally”
2	“bob”

CRASHED

ally”>

F1

<“sally”>

1	“sally”
2	“bob” ?

1	“sally”
2	“tom”

CRASHED

n”>

F2

<“sally”>

1	“sally”
2	“tom” ?

OK

F2

<“sally”>

1	“sally”
2	“tom” ?

ONGARO, D., AND OUSTERHOUT, J. In search of an understandable consensus algorithm. In Proc ATC’14, USENIX Annual Technical Conference (2014), USENIX, May 2014

Log Replication

A majority of nodes acknowledge the entry so it becomes committed

1	“sally”	
2	“bob”	1

CRASHED

ally”>

F1

<“sally”>

1	“sally”	
2	“bob”	1

term

1	“sally”	
2	“tom”	2

CRASHED

n”>

F2

<“sally”>

1	“sally”	
2	“tom”	2

term

OK

F2

<“sally”>

1	“sally”	
2	“tom”	2

term

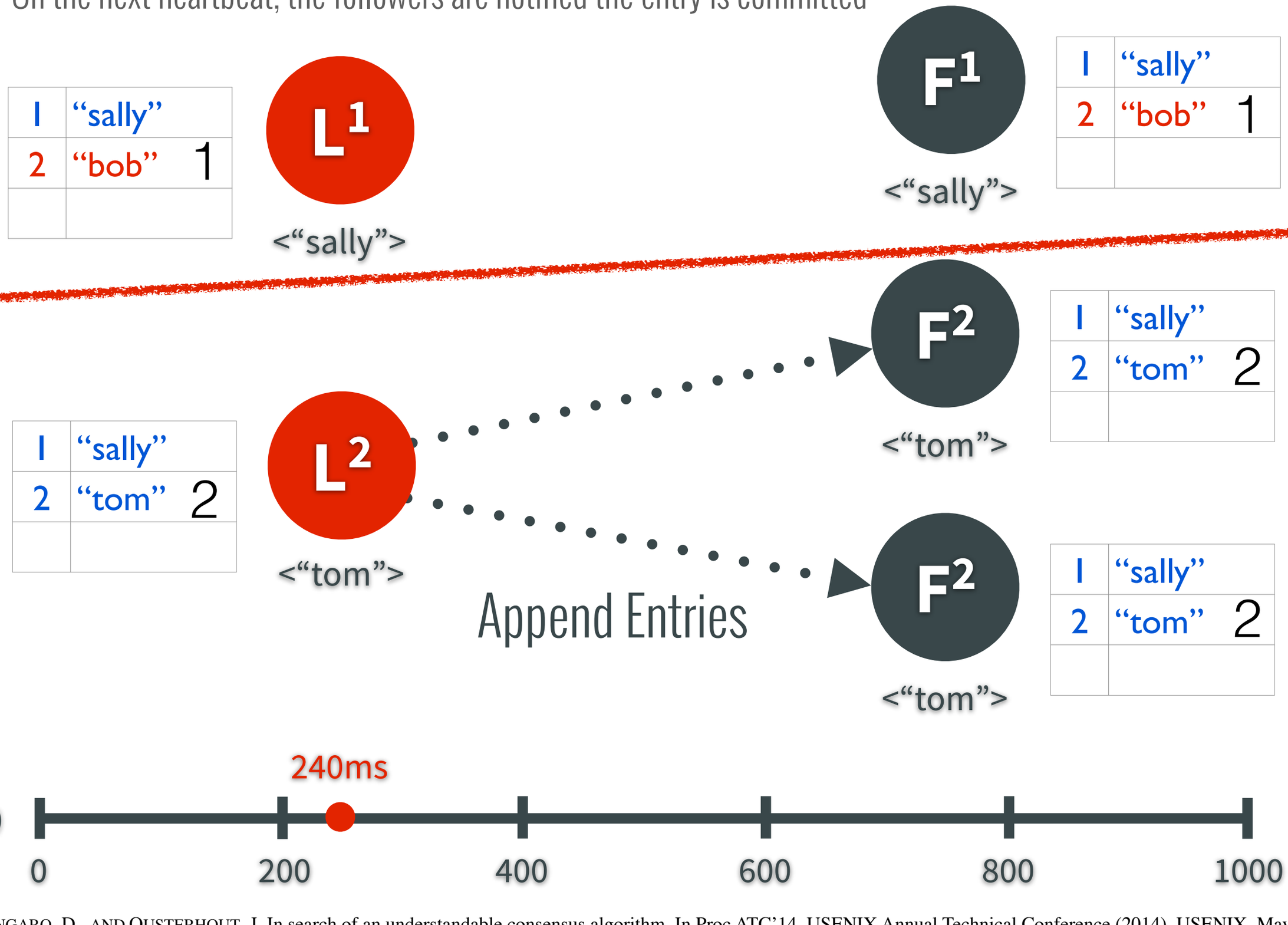
new leader has to use value(s) with highest term number (i.e., „tom“ instead“ of „bob“)

600 800 1000

ONGARO, D., AND OUSTERHOUT, J. In search of an understandable consensus algorithm. In Proc ATC’14, USENIX Annual Technical Conference (2014), USENIX, May 2014

Log Replication

On the next heartbeat, the followers are notified the entry is committed



ONGARO, D., AND OUSTERHOUT, J. In search of an understandable consensus algorithm. In Proc ATC'14, USENIX Annual Technical Conference (2014), USENIX, May 2014

Log Replication

1	“sally”
2	“bob” 1

<“sally”>

<“sally”>

1	“sally”
2	“bob” 1

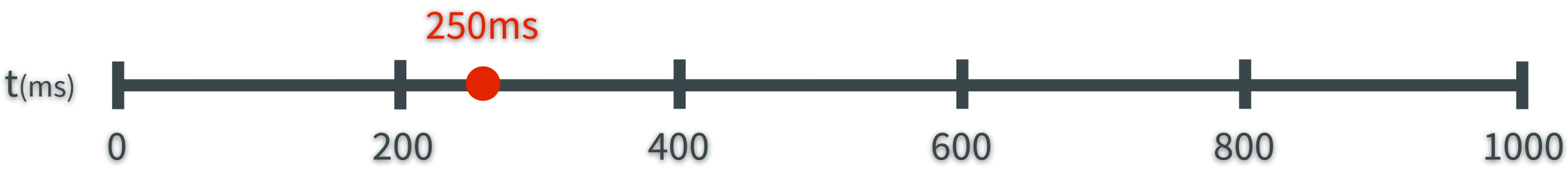
1	“sally”
2	“tom” 2

<“tom”>

<“tom”>

<“tom”>

1	“sally”
2	“tom” 2



ONGARO, D., AND OUSTERHOUT, J. In search of an understandable consensus algorithm. In Proc ATC’14, USENIX Annual Technical Conference (2014), USENIX, May 2014

Log Replication

The network recovers and there is no longer a partition

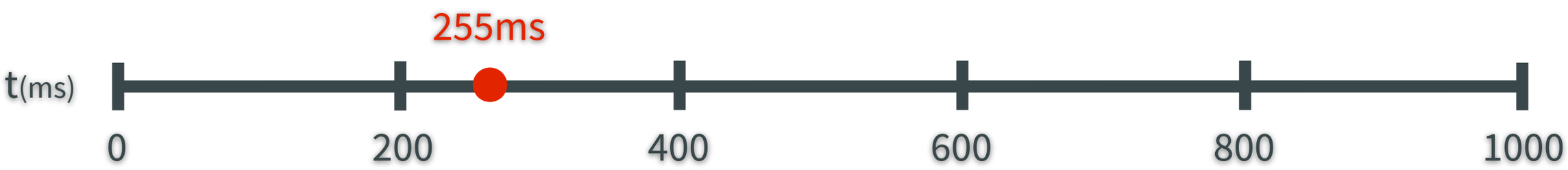
1	“sally”	
2	“bob”	1

1	“sally”	
2	“bob”	1

1	“sally”	
2	“tom”	2

1	“sally”	
2	“tom”	2

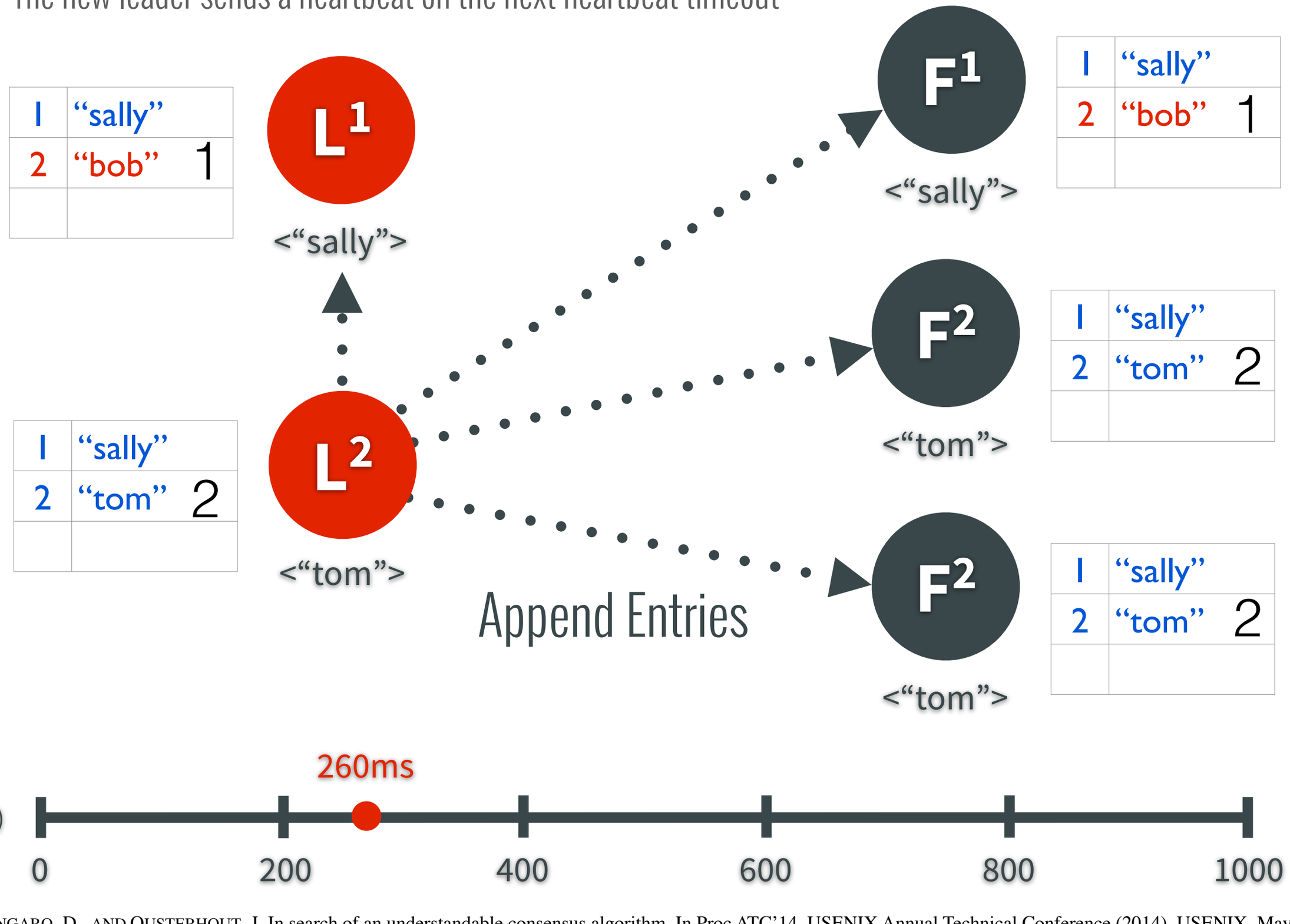
1	“sally”	
2	“tom”	2



ONGARO, D., AND OUSTERHOUT, J. In search of an understandable consensus algorithm. In Proc ATC’14, USENIX Annual Technical Conference (2014), USENIX, May 2014

Log Replication

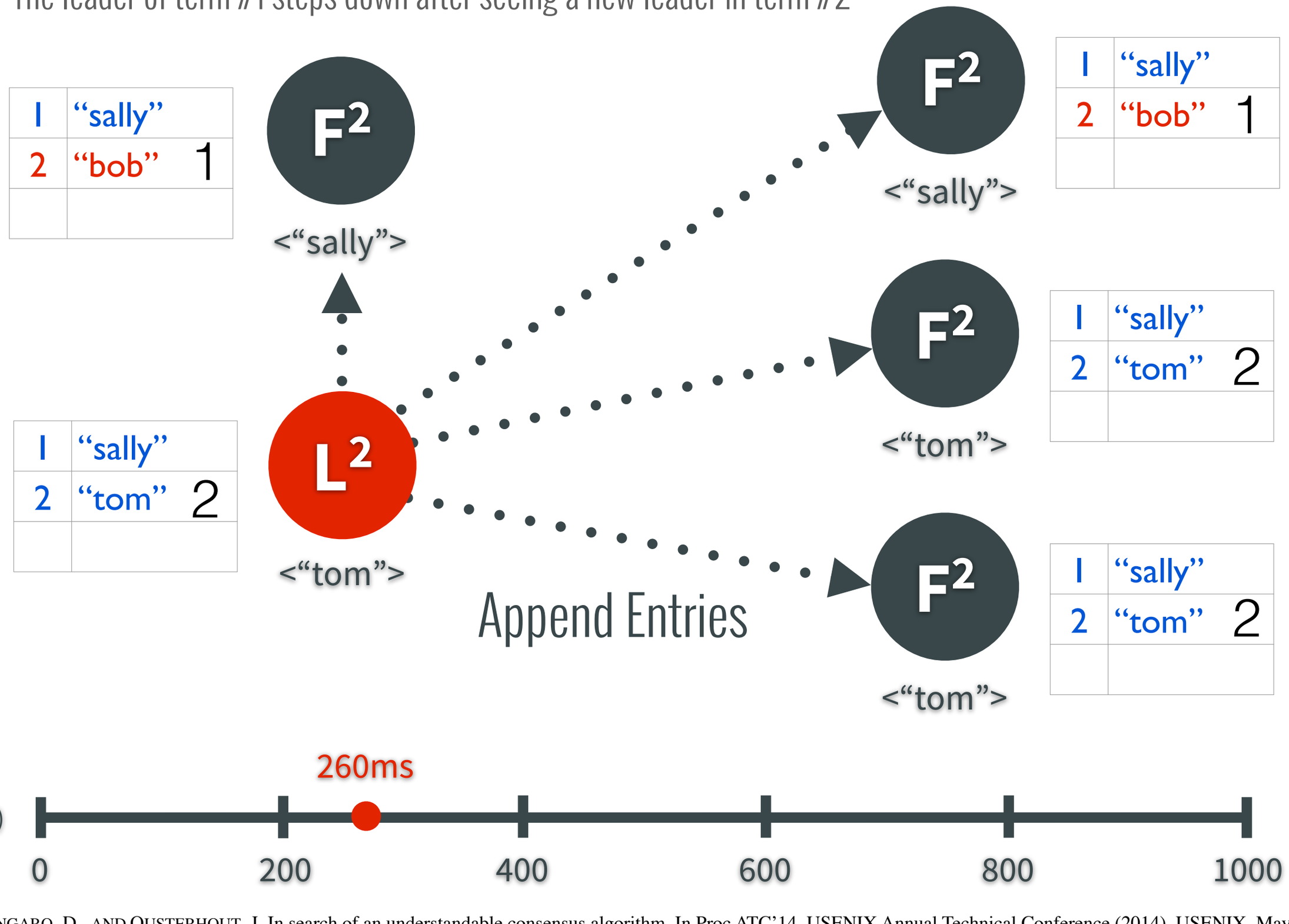
The new leader sends a heartbeat on the next heartbeat timeout



ONGARO, D., AND OUSTERHOUT, J. In search of an understandable consensus algorithm. In Proc ATC'14, USENIX Annual Technical Conference (2014), USENIX, May 2014

Log Replication

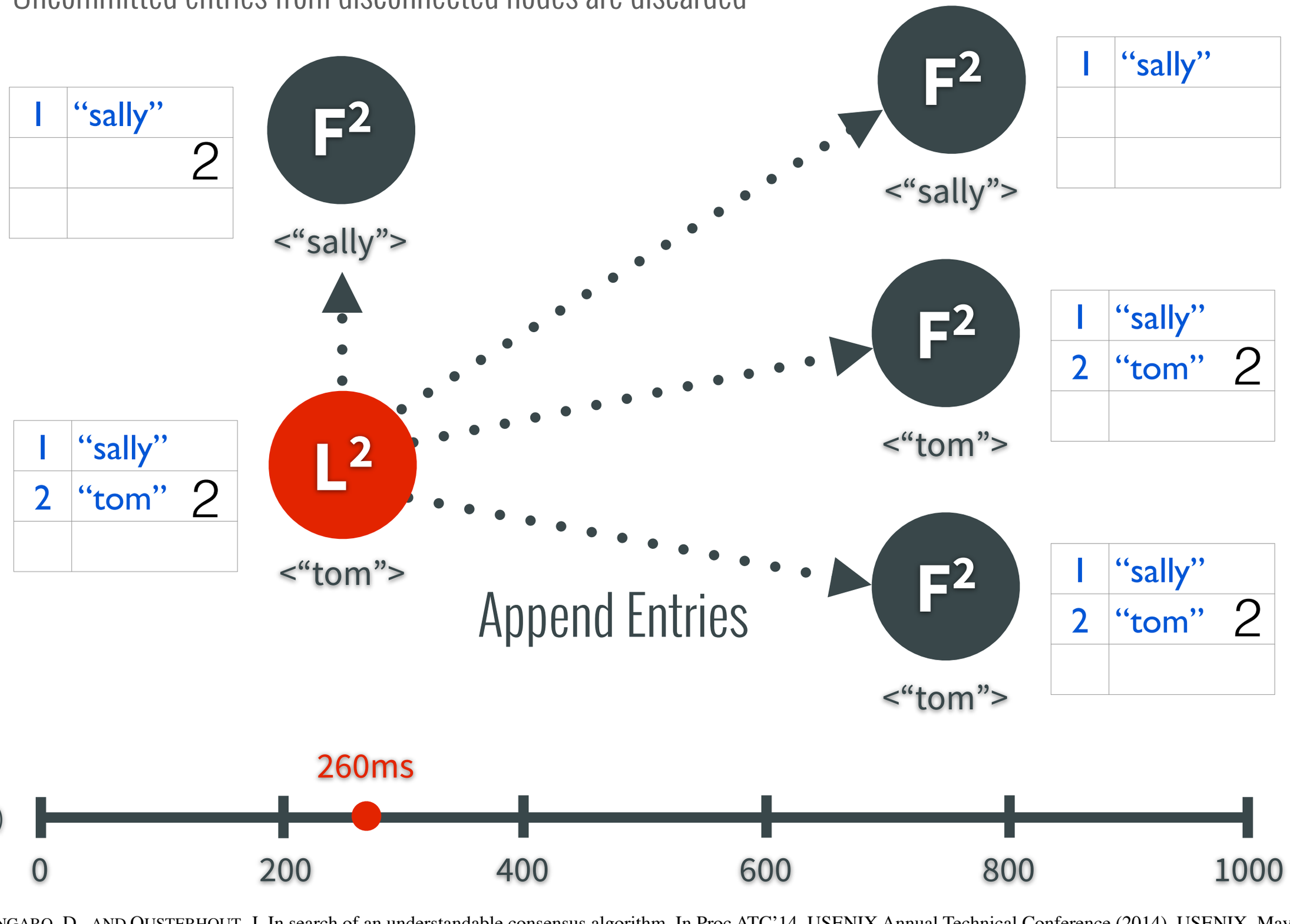
The leader of term #1 steps down after seeing a new leader in term #2



ONGARO, D., AND OUSTERHOUT, J. In search of an understandable consensus algorithm. In Proc ATC'14, USENIX Annual Technical Conference (2014), USENIX, May 2014

Log Replication

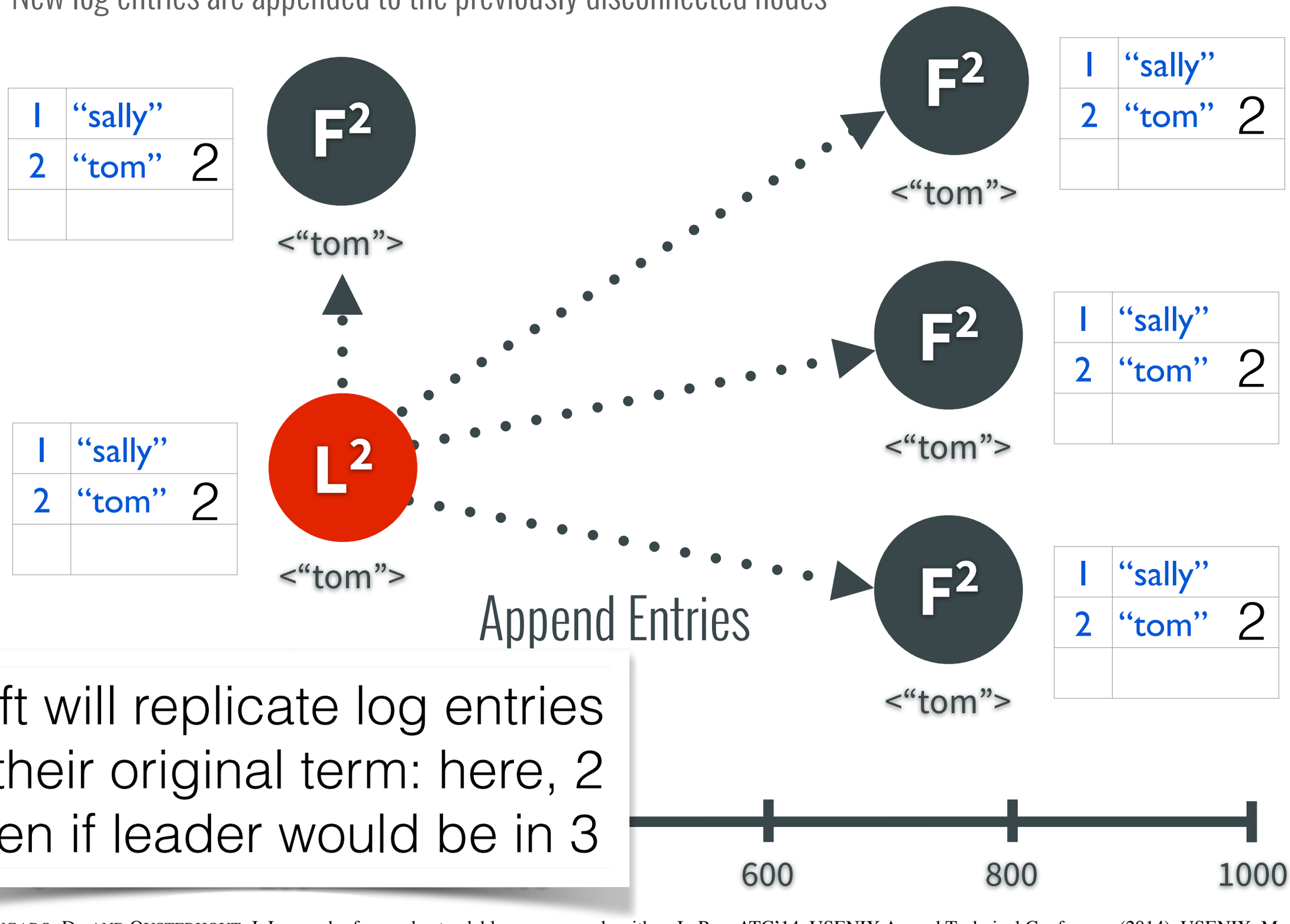
Uncommitted entries from disconnected nodes are discarded



ONGARO, D., AND OUSTERHOUT, J. In search of an understandable consensus algorithm. In Proc ATC'14, USENIX Annual Technical Conference (2014), USENIX, May 2014

Log Replication

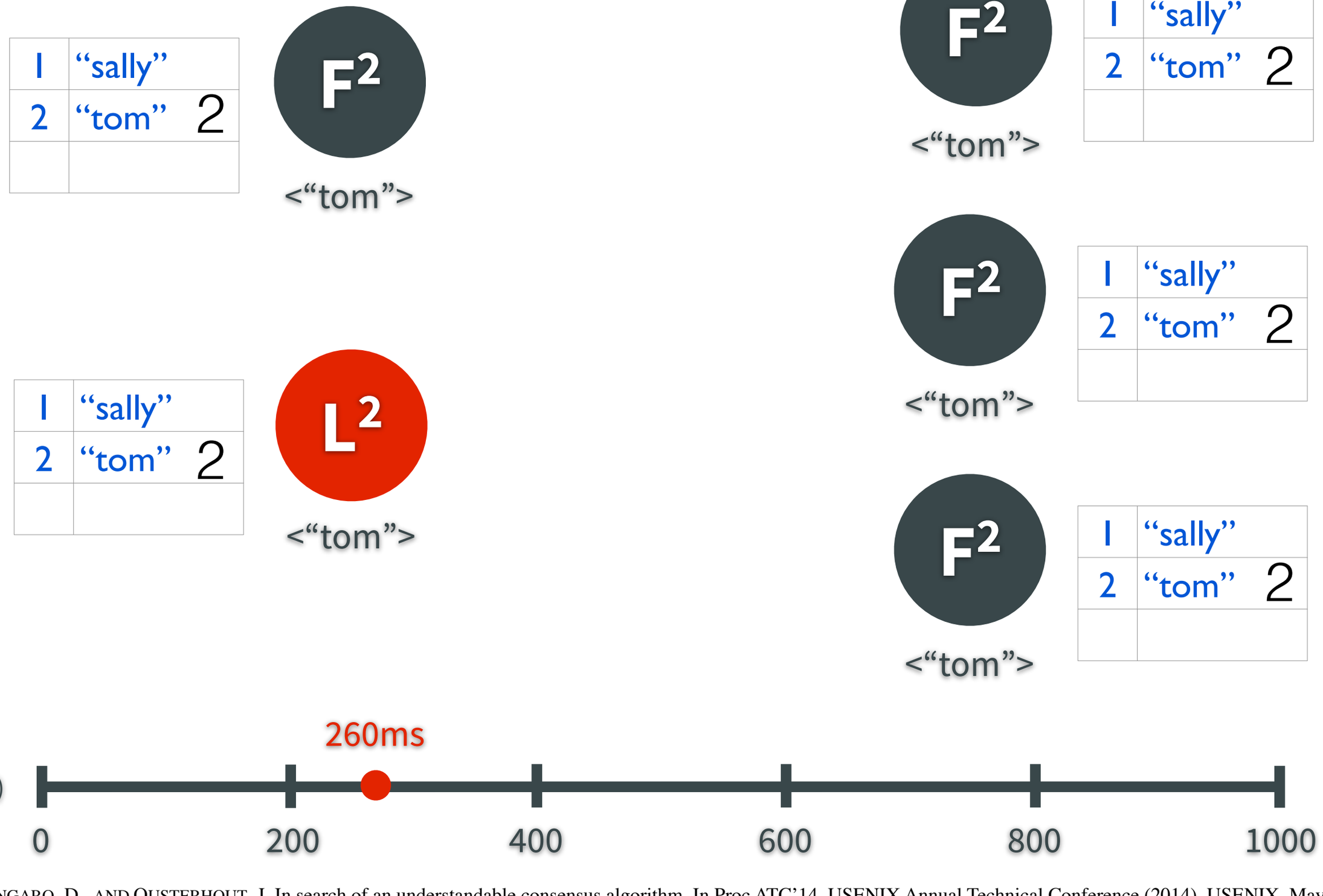
New log entries are appended to the previously disconnected nodes



Raft will replicate log entries in their original term: here, 2 even if leader would be in 3

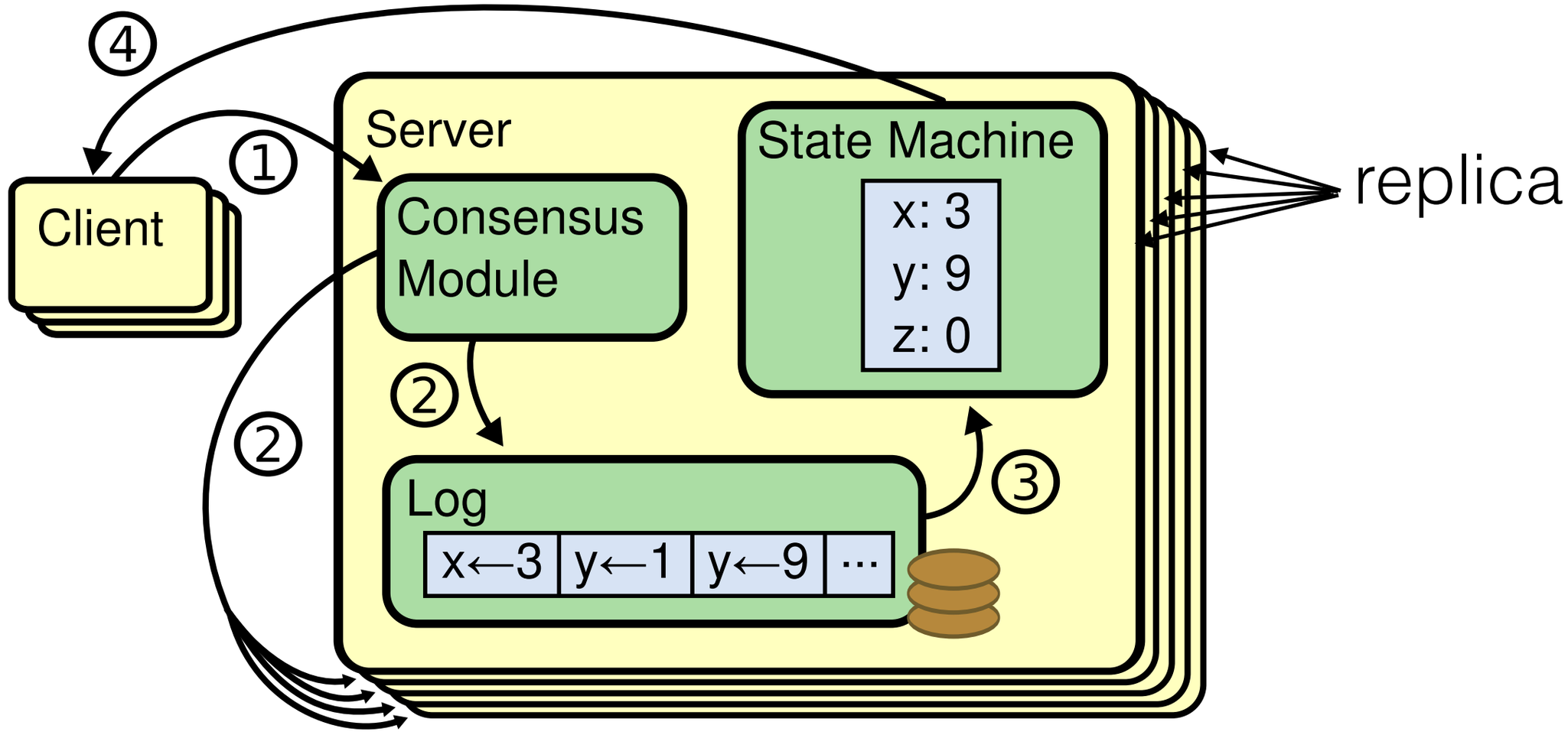
ONGARO, D., AND OUSTERHOUT, J. In search of an understandable consensus algorithm. In Proc ATC’14, USENIX Annual Technical Conference (2014), USENIX, May 2014

Log Replication



ONGARO, D., AND OUSTERHOUT, J. In search of an understandable consensus algorithm. In Proc ATC'14, USENIX Annual Technical Conference (2014), USENIX, May 2014

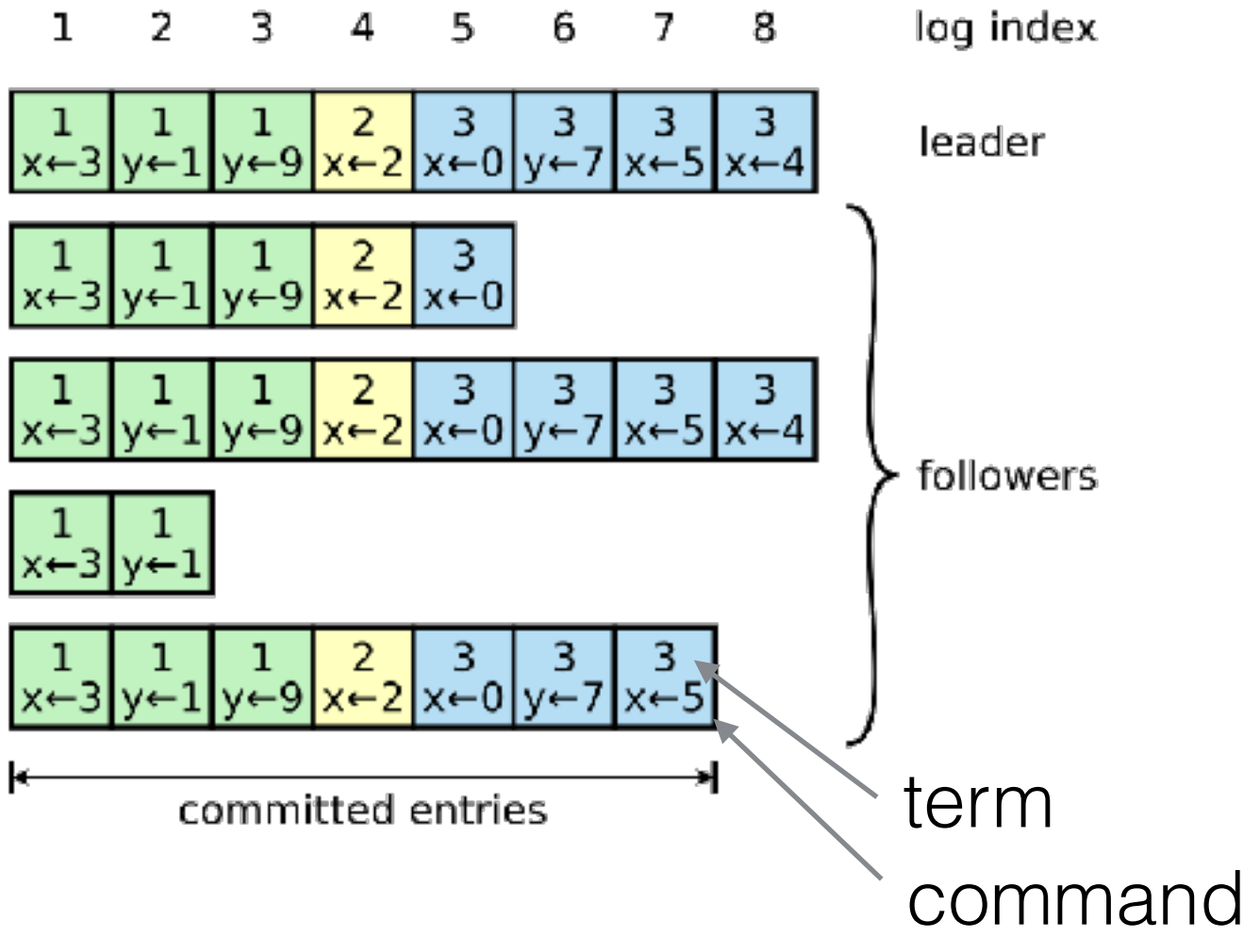
Architecture



ONGARO, D., AND OUSTERHOUT, J. In search of an understandable consensus algorithm. In Proc ATC'14, USENIX Annual Technical Conference (2014), USENIX, May 2014

Raft: Transaction Log

- **replicated log** guarantees immutable total order (FIFO)
  - append-only (committed entries)
  - leader is single writer
  - leader forces own log on followers



ONGARO, D., AND OUSTERHOUT, J. In search of an understandable consensus algorithm. In Proc ATC'14, USENIX Annual Technical Conference (2014), USENIX, May 2014

Log Compaction

ONGARO, D., AND OUSTERHOUT, J. In search of an understandable consensus algorithm. In Proc ATC'14, USENIX Annual Technical Conference (2014), USENIX, May 2014

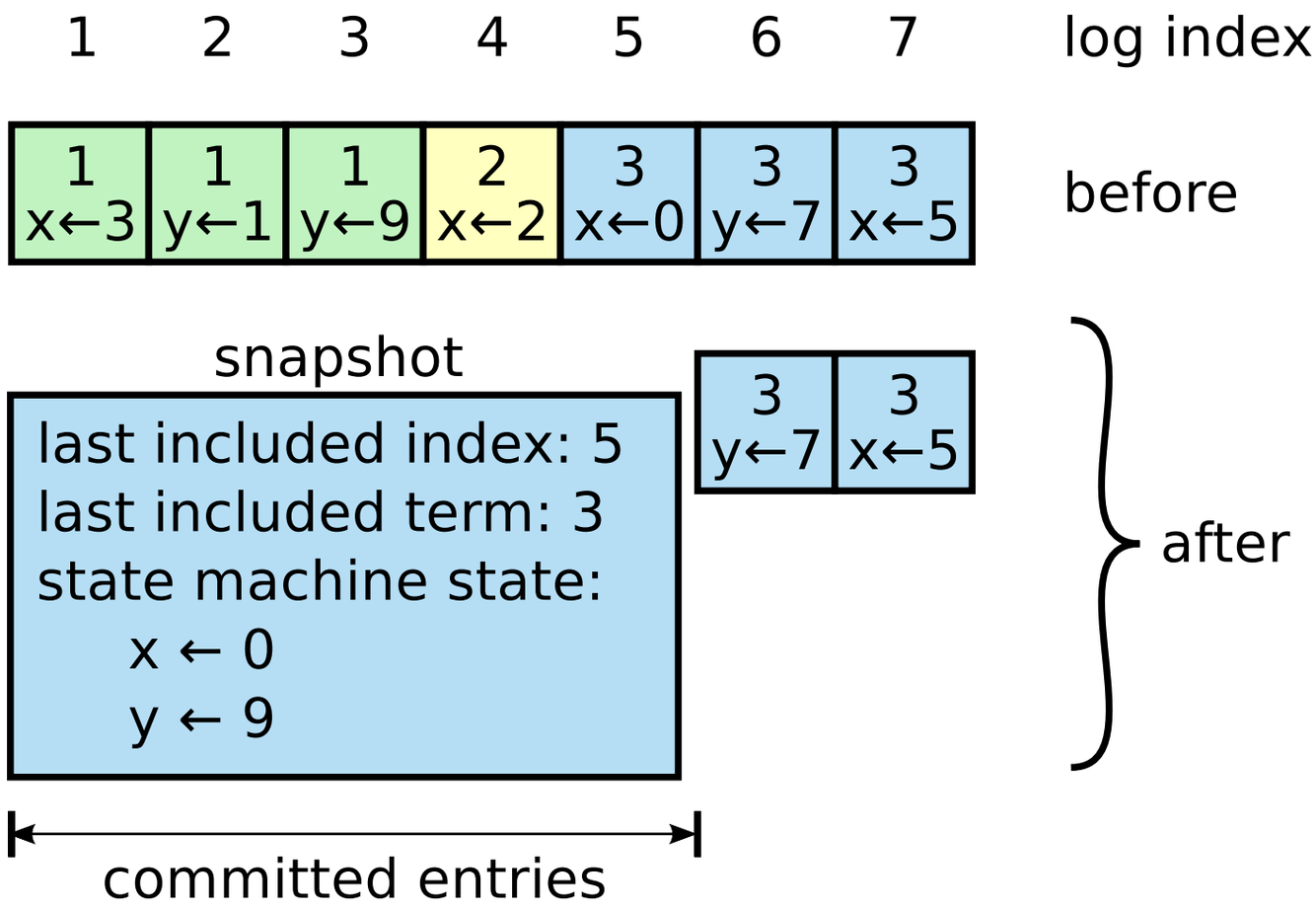
Unbounded log can grow until there’s no more disk

ONGARO, D., AND OUSTERHOUT, J. In search of an understandable consensus algorithm. In Proc ATC’14, USENIX Annual Technical Conference (2014), USENIX, May 2014

Recovery time increases as log length increases

ONGARO, D., AND OUSTERHOUT, J. In search of an understandable consensus algorithm. In Proc ATC'14, USENIX Annual Technical Conference (2014), USENIX, May 2014

Leader-Initiated Snapshot



ONGARO, D., AND OUSTERHOUT, J. In search of an understandable consensus algorithm. In Proc ATC'14, USENIX Annual Technical Conference (2014), USENIX, May 2014

Summary

- RAFT
  - a popular alternative to Paxos
  - several implementations in several programming languages available
  - not sure if this is really easier to understand?
    - It might just be the way Paxos and Raft are presented

Question?

- **Client triggers update**
  - connect to closest of three data centers for updates?
- **Round-Trip Delay**
  - 25 ms client to data center US-West
  - 45 ms client to US-Central
- **Questions**:
  - can we achieve 25ms+45ms = 70ms? (in optimal case)
  - if yes, how? if no, why not?

References

- Junqueira, F., Reed, B., Serafini, M. “Zab: High-performance broadcast for primary-backup systems”. *41st International Conference on Dependable Systems & Networks*, 2011
- Ongaro, D. and Ousterhout, J. „In Search of an Understandable Consensus Algorithm“. *USENIX Annual Technical Conference*, 2014
- https://ramcloud.stanford.edu/~ongaro/cs244b.pdf
- http://slidedeck.io/trinary/raft-talk