# Raft for Consistent Replicated Log


Seif Haridi

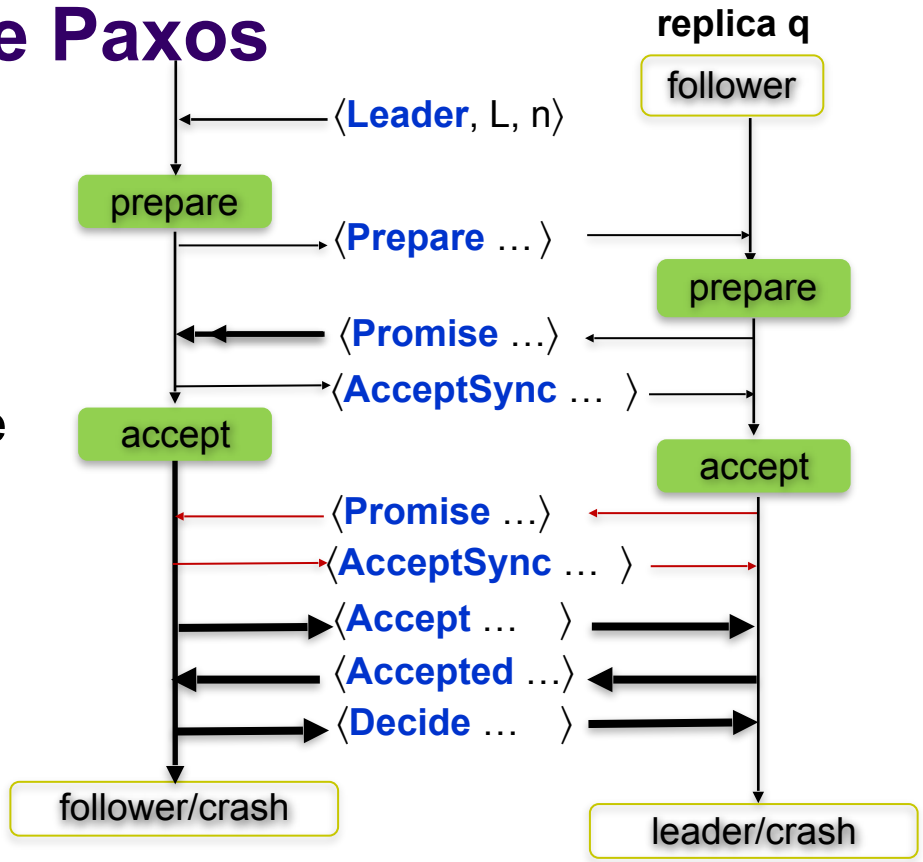
# Outline

- Sequence Paxos
  - Fail-Recovery Model
  - Session-based FIFO links
- The Raft algorithm
  - A functional restructuring of leader-based Sequence Paxos with some innovations
  - Does not assume FIFO links
  - Tolerates arbitrary message losses

# Sequence Paxos Fail Recovery Model

# Leader Based Sequence Paxos

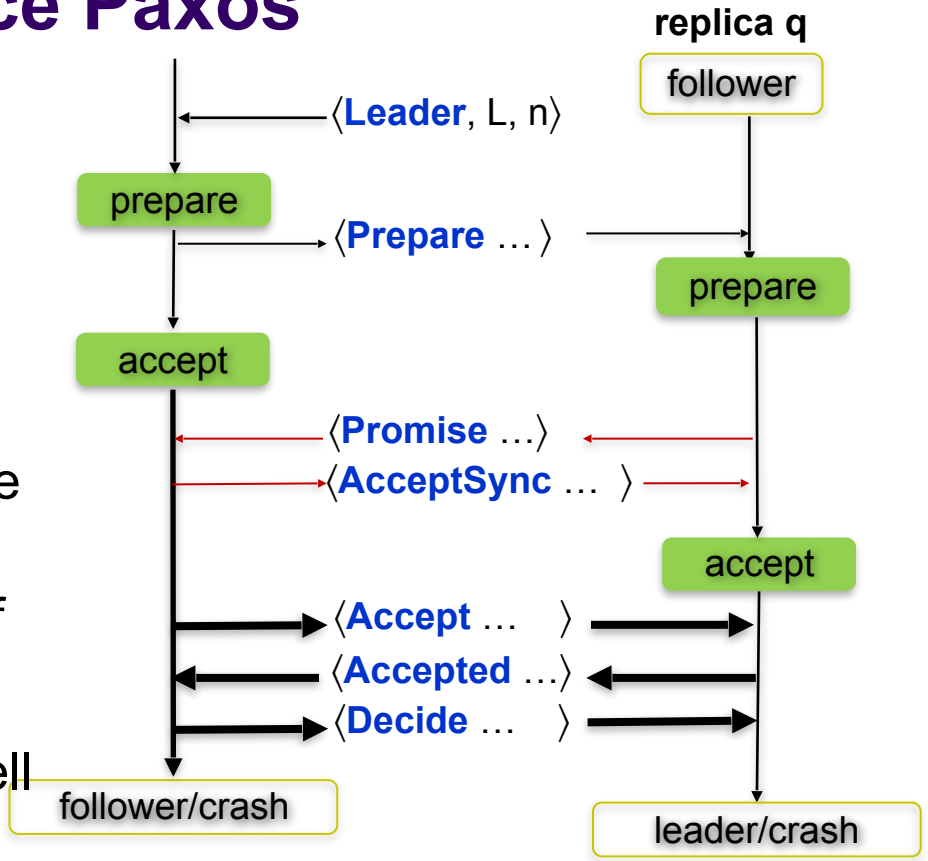
- Once leader L is elected
- Sends prepare to collect a *majority* of promises and forms its accepted sequence $v_a$
- $v_{a\ at\ L}$ has the longest chosen sequence at a prefix
- AcceptSync synchronizes $v_{a\ at\ q}$ for a majority of follower replicas
- The leader and those followers move to the accept phase
- $v_a$ is extended incrementally as well as the decided sequence prefix($v_a$, $l_d$)



# Leader Based Sequence Paxos

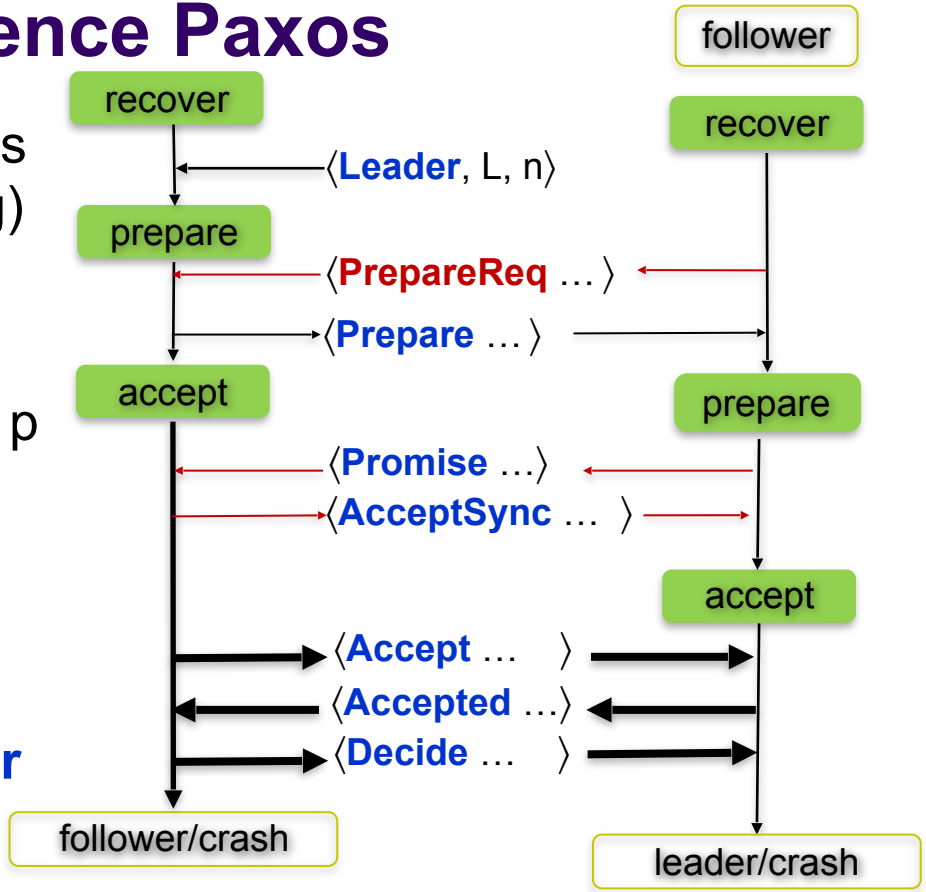
- Once leader L is elected
- Sends prepare to collect a *majority* of promises and forms its accepted sequence $v_a$
- $v_{a\ at\ L}$ has the longest chosen sequence at a prefix

- Late replicas q sends its promise while leader is at the accept phase
- AcceptSync synchronizes the state of $v_{a\ at\ q}$ for the replicas q
- $v_{a\ at\ q}$ is extended incrementally as well as the decided sequence prefix($v_a$, $l_d$)



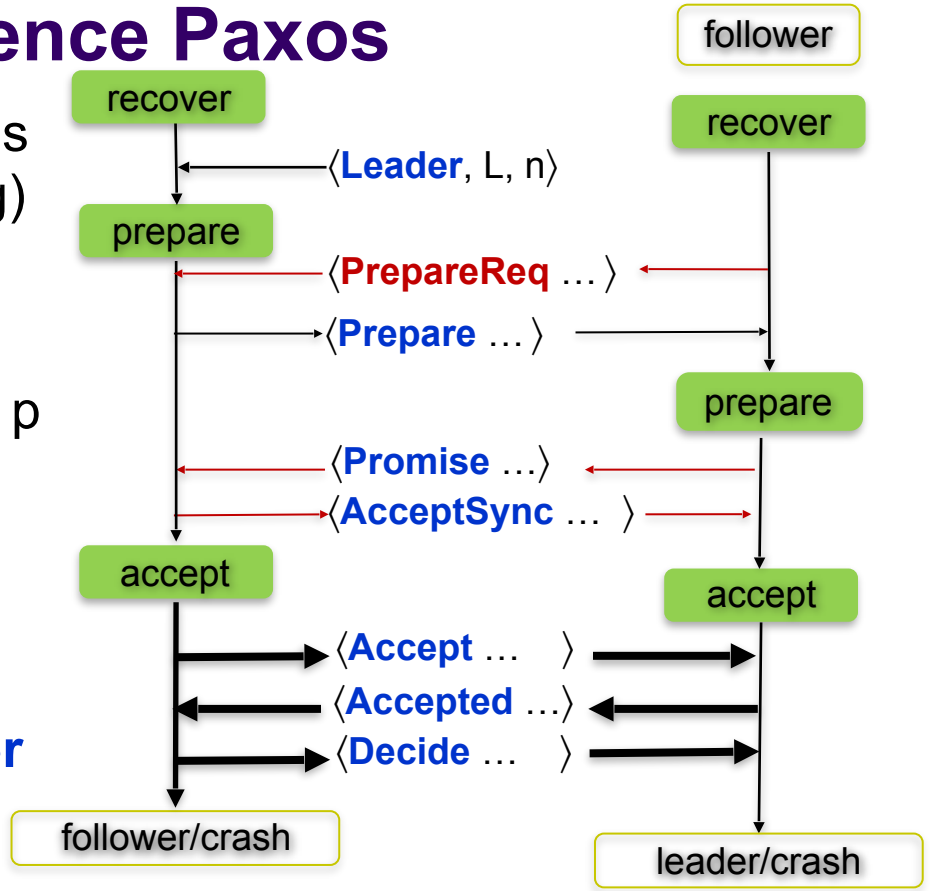
# Fail-Recovery in Sequence Paxos

- In the fail-recovery model a process is correct as long as it fails (by crashing) and recovers finite number of times
- By crashing and restarting a process p loses any arbitrary suffixes of most recent messages in each FIFO link
- Once a process restart: it joins the leader-election algorithm in a **recover** state



# Fail-Recovery in Sequence Paxos

- In the fail-recovery model a process is correct as long as it fails (by crashing) and recovers finite number of times
- By crashing and restarting a process p loses any arbitrary suffixes of messages in each FIFO link
- Once a process restart it joins the leader-election algorithm in a **recover** state



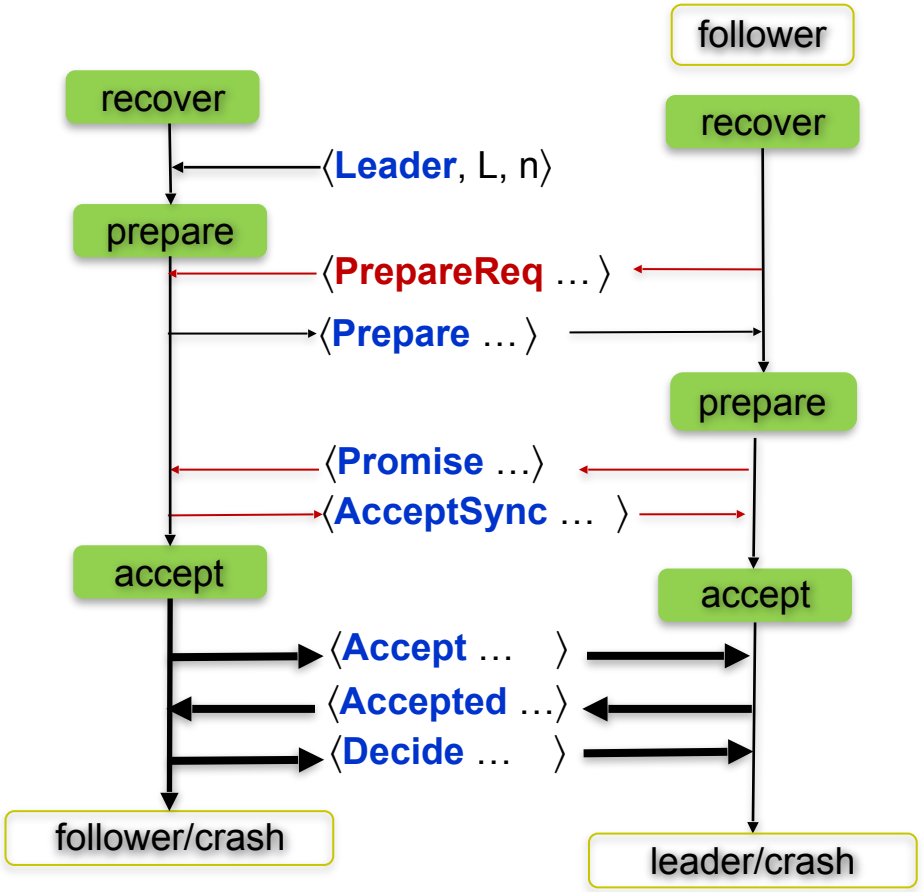
# Fail Recovery  persistent variables

- The algorithm needs to store the following variables in a persistent store for each process
  - $n_{prom}$ Promise not to accept in lower rounds
  - $n_a$ Round number in which last command is accepted
  - $v_a$ Accepted sequence
  - $l_d$ Length of decided sequence

- A recovered process resets its $ballot_{max}$ to $n_{prom}$ in BLE
- The leader election guarantees that a leader with higher ballot is elected if the leader crashed and recovered

# Fail Recovery

- A recovered process **p** starts in the state (**follower**, **recovered**)
  - Restores the persistent variables $n_{prom}$, $n_a$, $v_a$ , $l_d$
  - Waits for leader event ⟨**Leader**, L, n⟩
- **p = L:** p is the leader
  - Moves to state (**leader**, **prepare**)
  - Runs normal prepare phase



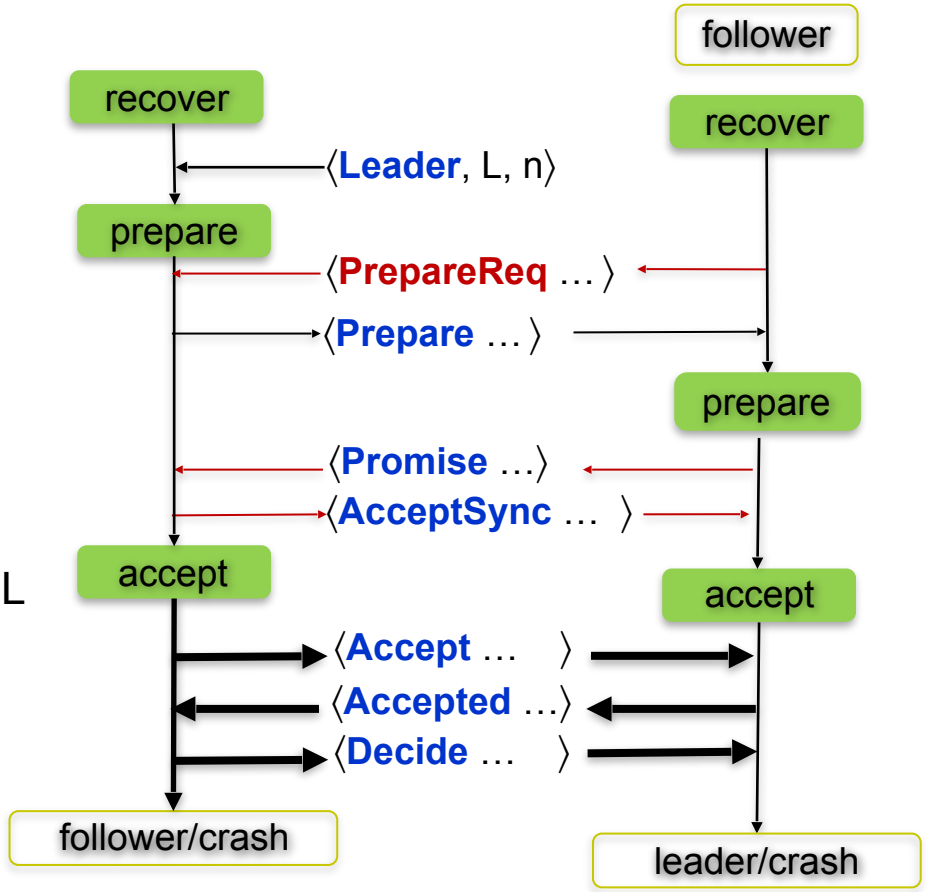
# Fail Recovery

A recovered process **p** starts in the state (**follower**, **recovered**)

- Restores the persistent variables $n_{prom}$, $n_a$, $v_a$ , $l_d$
- Waits for leader event ⟨**Leader**, L, n⟩

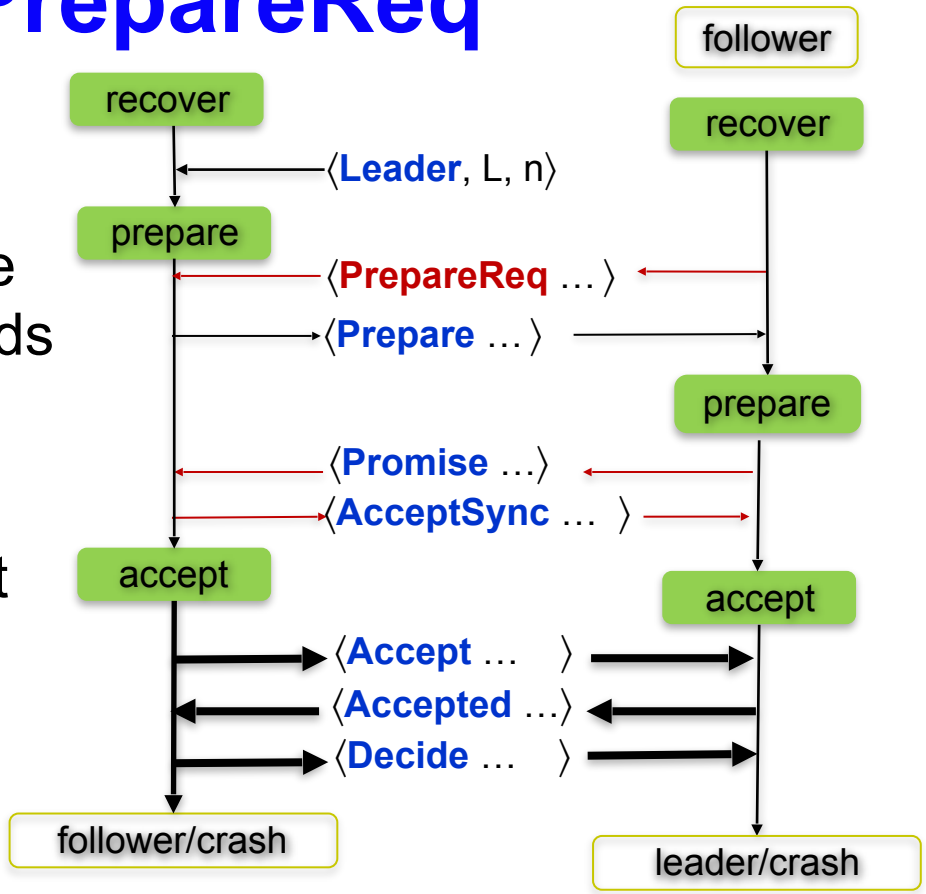
**p ≠ L:** p is a follower

- Request a prepare message for the leader L
- **send** ⟨**PrepareReq**⟩ **to** L
- When it received a prepare message it moves to (**follower**, **prepare**)
- Runs as normal



# Why the need for PrepareReq

- If the leader L is still in the prepare phase the recovered process needs to know the length of decided sequence $l_d$ at L
- Necessary to compute the longest chosen sequence at the leader



# Session based FIFO links

- Dropping a session between processes p1 and p2 means the links between the two processes are broken and an arbitrary suffix of messages are lost. Restarting a connection means new links are established between p1 and p2
- Session failure is normally due to process crashes or network partition
- In our algorithm if a session is dropped
  - If a follower p1 drops the session, it tries to reconnect in recovery state
  - If a leader p1 drops a session it just ignore it until a new connection request from the follower. Leader continues as normal

# Raft

## An algorithm for Replicated Log

# Raft Consensus Algorithm

- Based on a presentation by the designers of Raft:
  "Designing for Understandability: the Raft Consensus Algorithm"

  - **Diego Ongaro and  John Ousterhout**
  - **Some slides are borrowed from this presentation**

- We relate to Sequence Paxos

| | | |
|---|---|---|
| • **Sequence Paxos** | | • **Raft** |
| • $v_a$ The accepted sequence | ⟷ | • The Log |
| • The Decided sequence | ⟷ | • The committed prefix of Log |
| • Round/ballot number | ⟷ | • Term |
| • Process | ⟷ | • Server |
| • $n_{prom}$ , $n_L$ | ⟷ | • Highest Term |
| • Element in a sequence | ⟷ | • Entry |

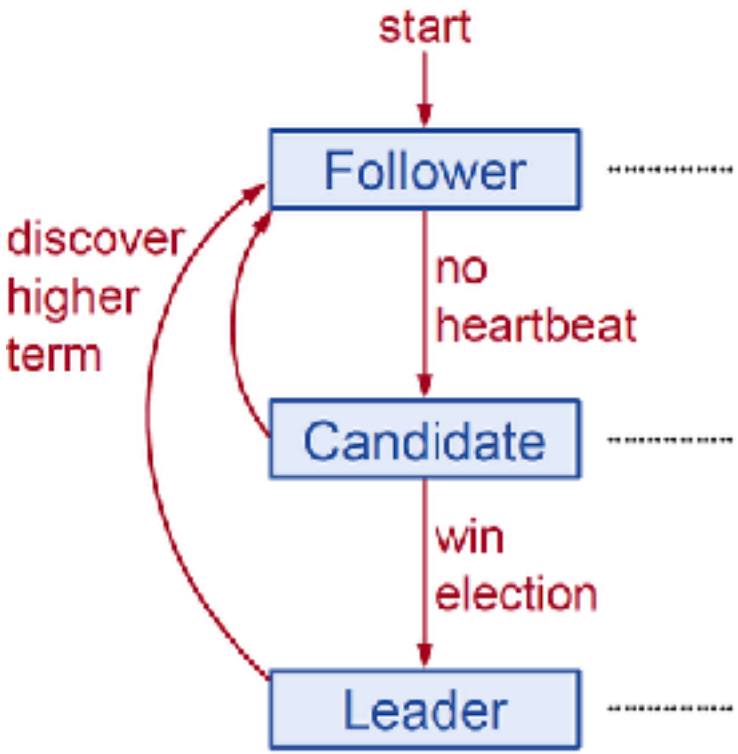
# Raft Decomposition

- **Leader election**
  - Select one server to act as leader (BLE)
  - Detect crashes, choose new leader (BLE)
  - Only servers with up-to-date logs can become leader
    - The leader election and Raft consensus are fused in one component
    - Incorporates the prepare phase in the leader-election algorithm
    - In election a leader with highest term (round number) and highest entry index (longest sequence) is elected

# Raft Decomposition

- **Log replication (normal operation)**
  - Leader accepts commands from clients, appends to its log
  - **Leader replicates its log to other servers (overwrites inconsistencies)**
  - **Keep logs consistent**
  - Consistent replication is done differently from sequence Paxos by some form of *log reconciliation*

# Server States and RPCs



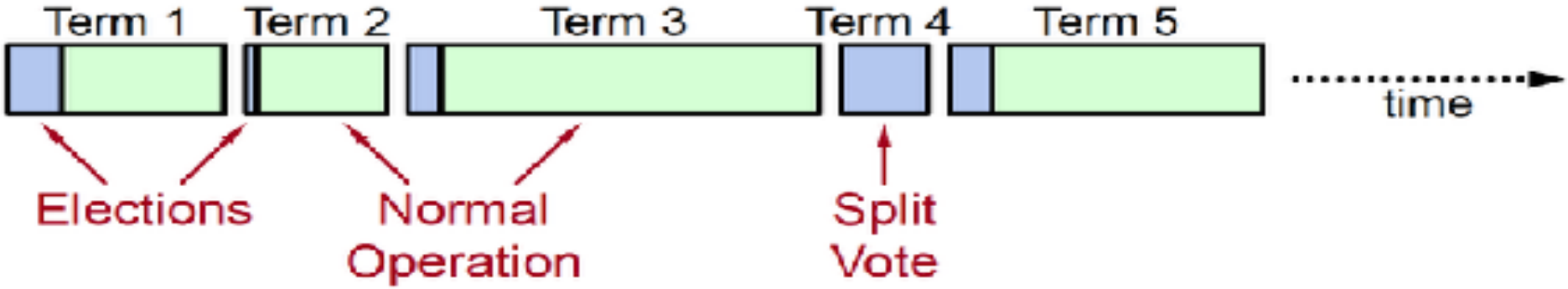

- Raft uses a request/reply pattern for sending messages  RPCs

Passive (but expects regular heartbeats)

Issues **RequestVote** RPCs to get elected as leader

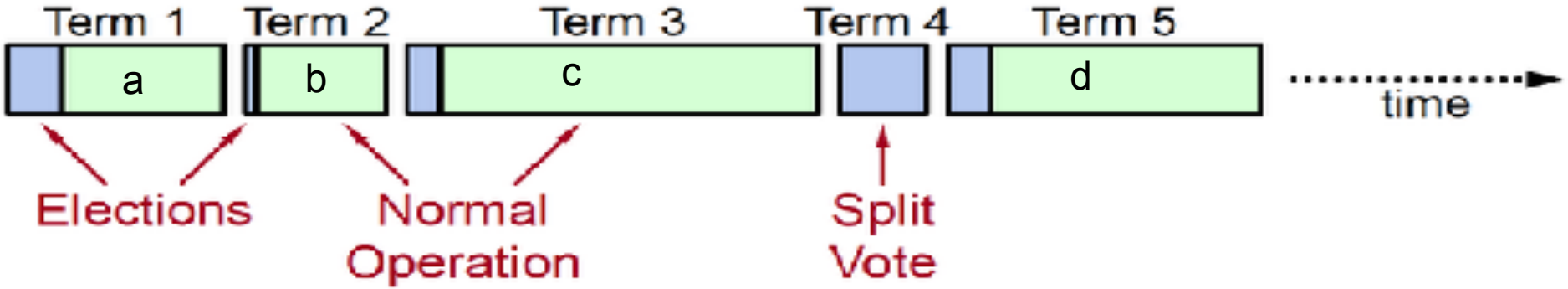
Issues **AppendEntries** RPCs:
- Replicate its log
- Heartbeats to maintain leadership

# Terms (rounds)




- **At most 1 leader per term (**some terms might fail to elect a leader)
- **Each server maintains current term value** (maintaining $n_{prom}$)
  - Exchanged in every RPC
  - Server has higher term? Update term, leader revert to follower
  - Incoming RPC has lower term? Reply with error

# Terms (rounds) vs. Ballot Array



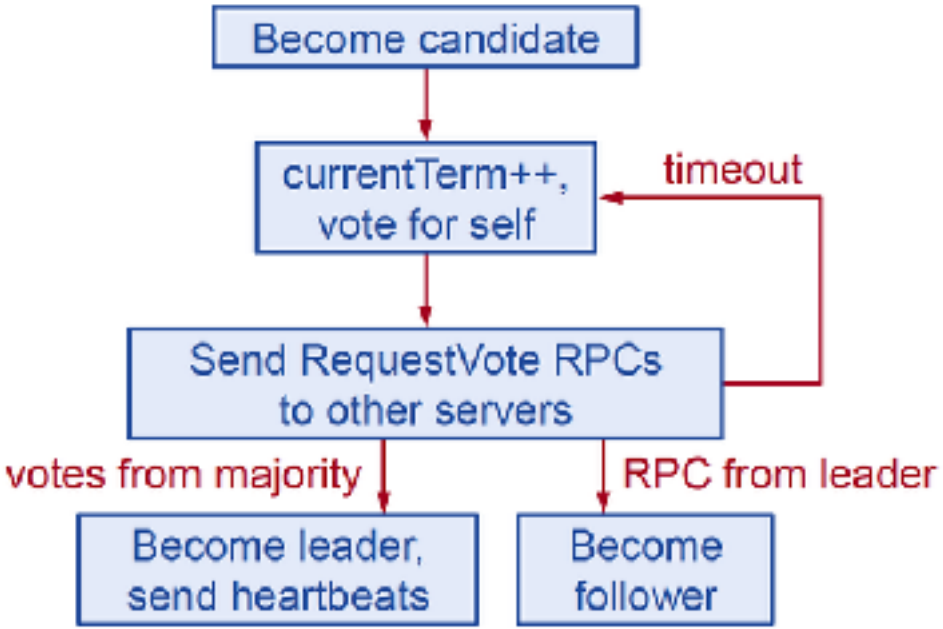

| Round | Accepted by $p_1$ | Accepted by $p_2$ | Accepted by $p_3$ |
|---|---|---|---|
| Term 5 | **a⊕b⊕c⊕d** | | |
| **Term 4** | | | |
| Term 3 | **a⊕b⊕c** | **a⊕b⊕c** | |
| Term 2 | **a** | **a⊕b** | **a⊕b** |
| Term 1 | **a** | **a** | ⟨⟩ |

# The Election

# Leader Election

- Randomized starts
- Each server gives only one vote per term
- Majority required to win election

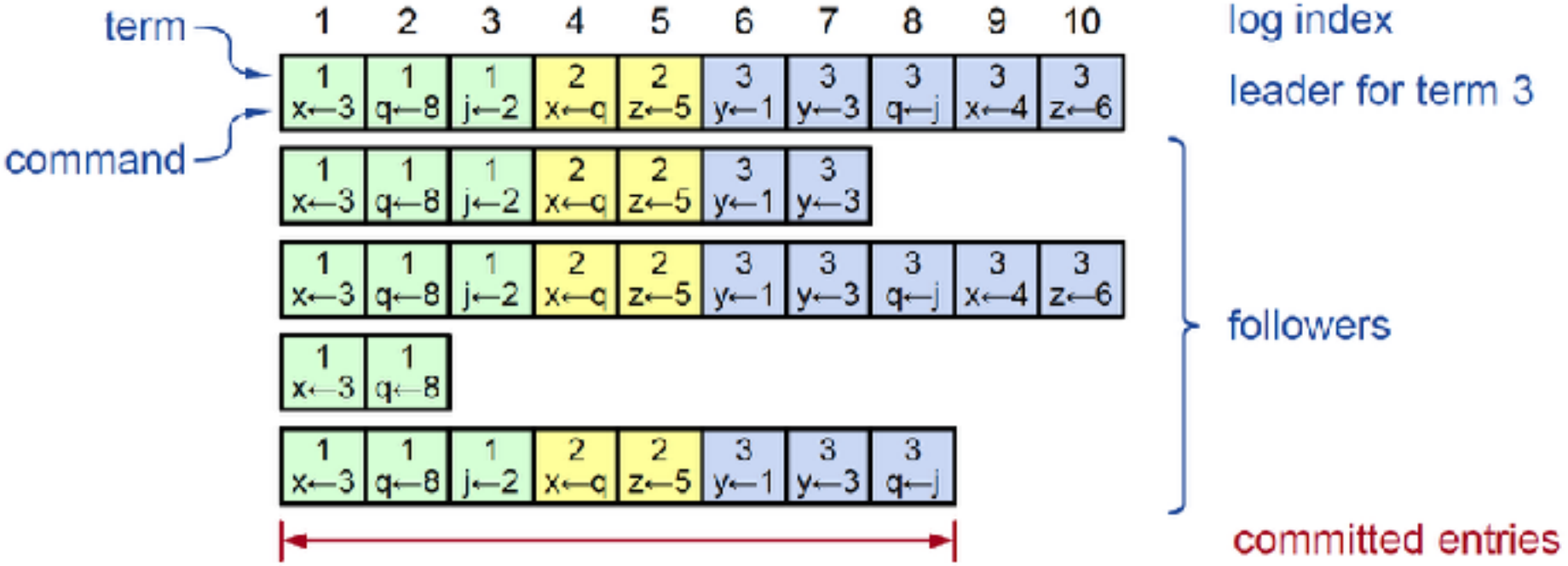
- **Server p rejects candidate q**
  - If highest log entry of q has a lower term or same term but lower index



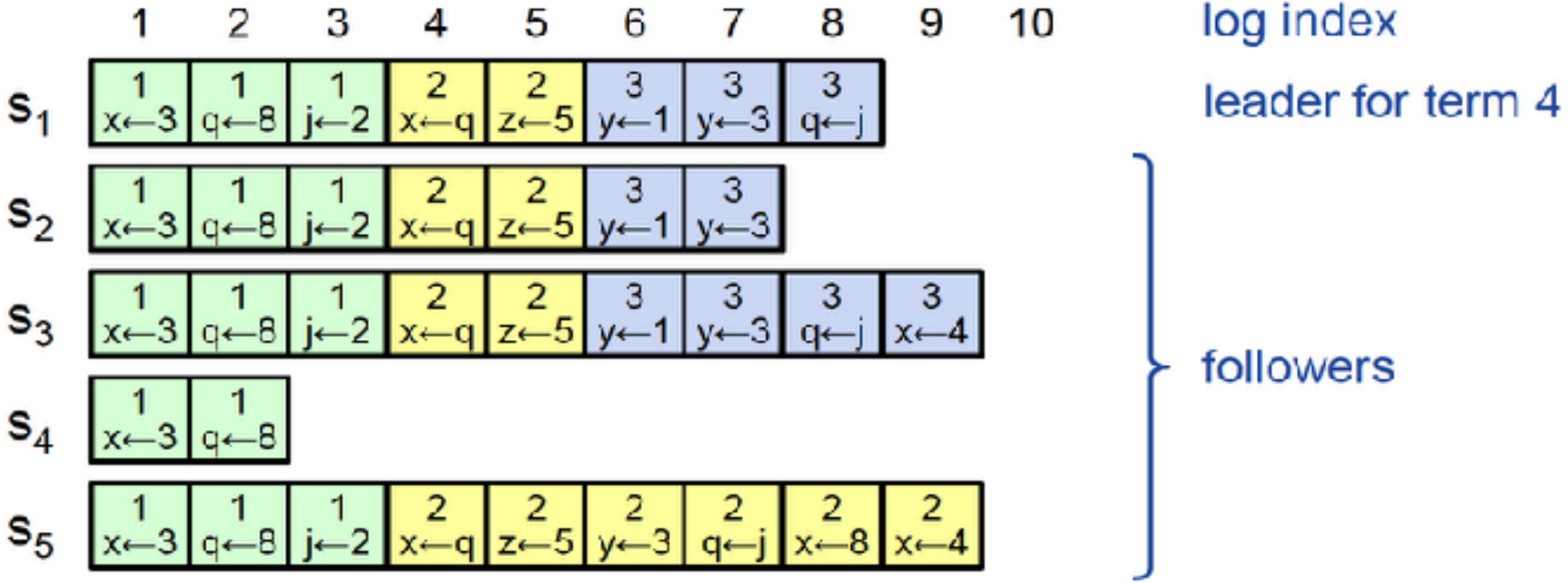
# Normal Operation

- Client sends command to leader
- Leader appends command to its log
- Leader sends **AppendEntries** RPCs to all followers (similar to **accept** messages in Sequence Paxos)
- **Entry is committed if**
  - Replicated on majority of servers by leader of its term
  - Once committed Leader executes command in its state machine, returns result to client
  - Notifies followers in subsequent **AppendEntries**(similar to **decide** messages)

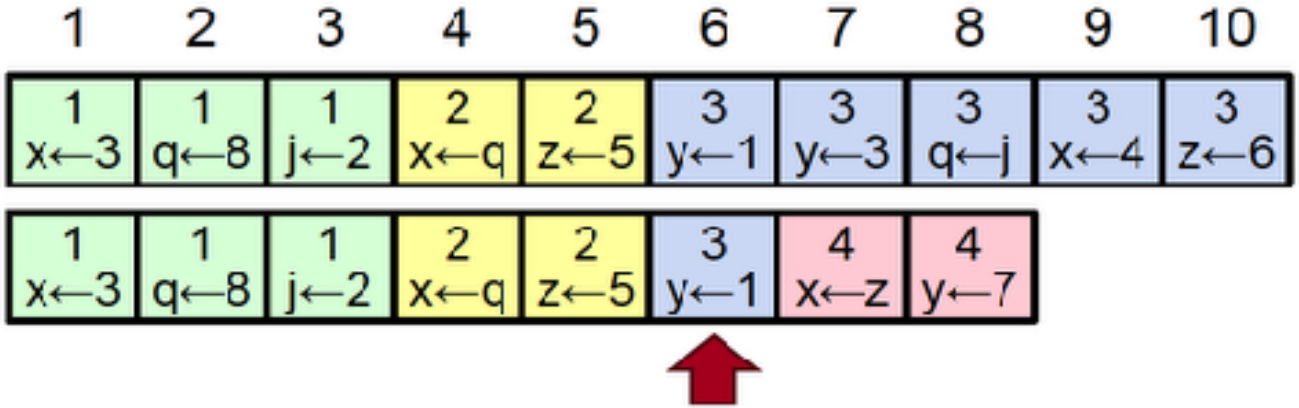
# Log structure and Log reconciliation


term
command
1 2 3 4 5 6 7 8 9 10
log index
1 x←3 | 1 q←8 | 1 j←2 | 2 x←q | 2 z←5 | 3 y←1 | 3 y←3 | 3 q←j | 3 x←4 | 3 z←6
leader for term 3
1 x←3 | 1 q←8 | 1 j←2 | 2 x←q | 2 z←5 | 3 y←1 | 3 y←3
1 x←3 | 1 q←8 | 1 j←2 | 2 x←q | 2 z←5 | 3 y←1 | 3 y←3 | 3 q←j | 3 x←4 | 3 z←6
1 x←3 | 1 q←8
1 x←3 | 1 q←8 | 1 j←2 | 2 x←q | 2 z←5 | 3 y←1 | 3 y←3 | 3 q←j
followers
committed entries

- Crashes and network partitions my results in inconsistent logs

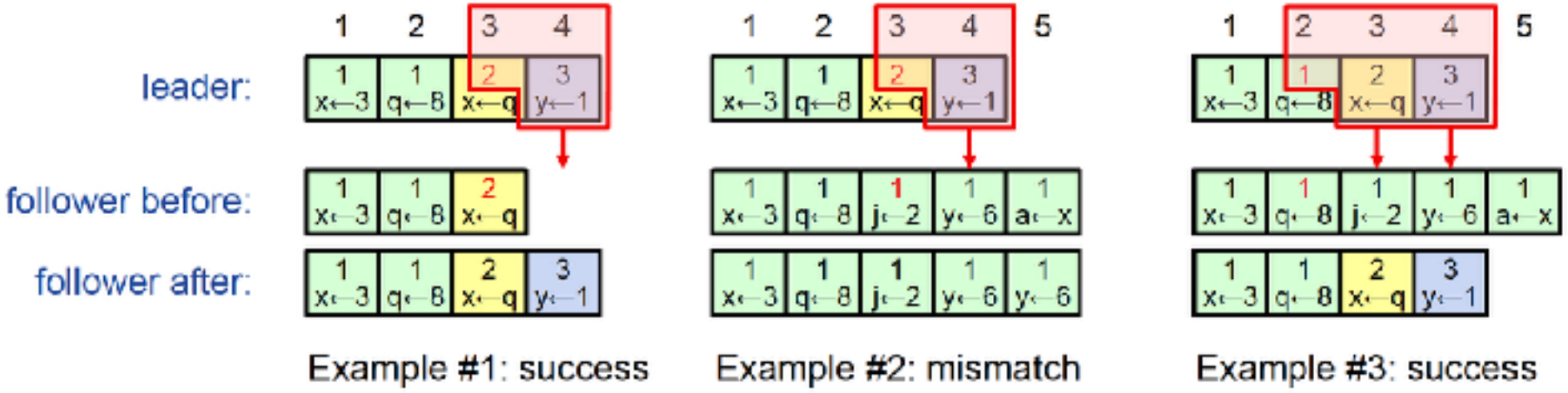


- **If log entries on different servers have same index and term**
  - They store the same command
  - The logs are identical in all preceding entries
- **If a given entry is committed, all preceding entries are also committed**



# Log reconciliation

- AppendEntries RPCs include <index, term> of entry preceding new one(s)
- Follower must contain matching entry; otherwise it rejects request
  - Leader retries with lower log index



# Summary

- Raft as Sequence Paxos have the same basic Paxos idea
  - The longest chosen sequence is the decided (committed) sequence
  - Leaders must have a higher round (term) number
- Raft differs from Sequence Paxos on
  - Leader election algorithm
  - Incorporating the prepare phase as part of electing a leader
  - Log (Accepted Sequence) reconciliation between leaders and followers