Big Data Systems

Djellel Difallah

Spring 2023

Lecture 13 (cont.) – Cluster Coordination

The Raft Consensus Algorithm

Outline

- Introduction
  - Consensus and the Byzantine Fault
- The Raft Algorithm
  - Replicated logs
  - Leader election
  - Normal operations
  - Safety and consistency

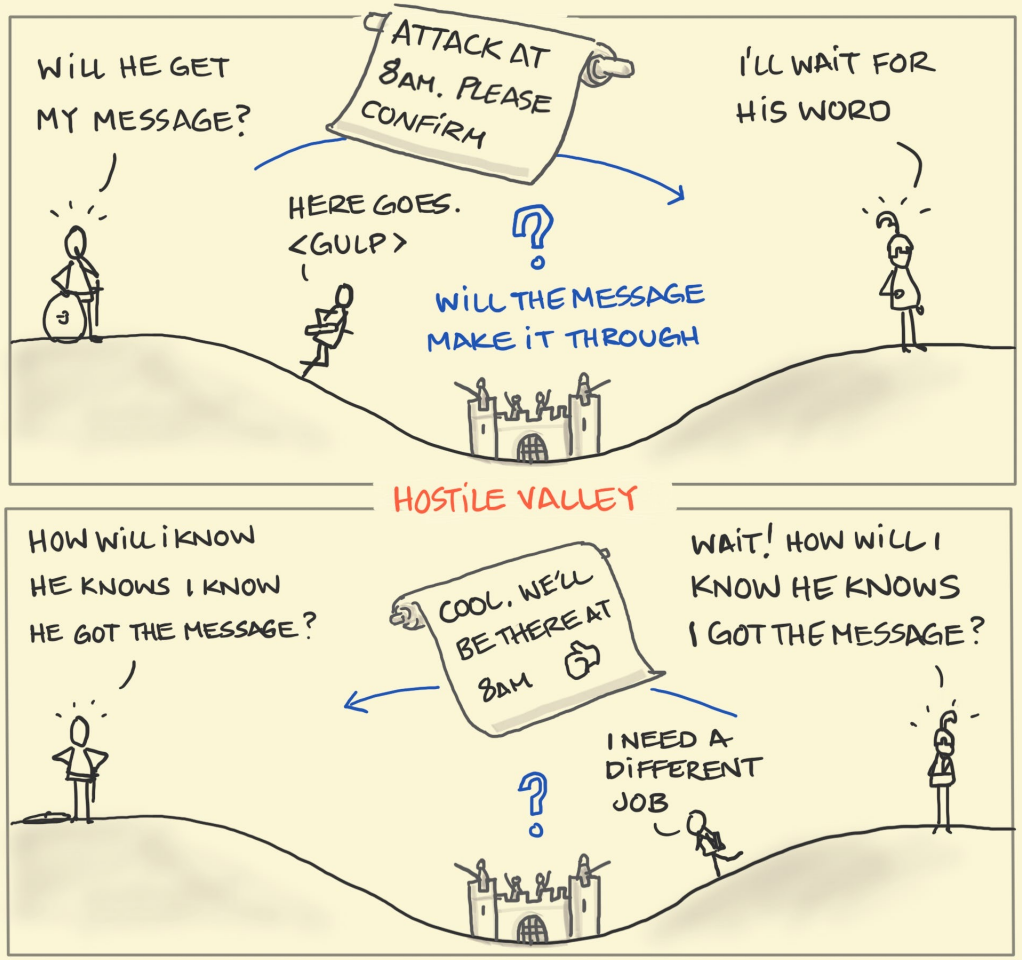
Consensus

- Agreement on shared state
- Recovery from server failures
  - If a minority of servers fail: no problem
  - Majority fail: lose availability, retain consistency
- This concept is key to building consistent storage systems
  - Specifically: replicated logs
  - Example: Write Ahead Log (WAL), Journals, Transaction Logs, etc.

THE TWO GENERALS PROBLEM

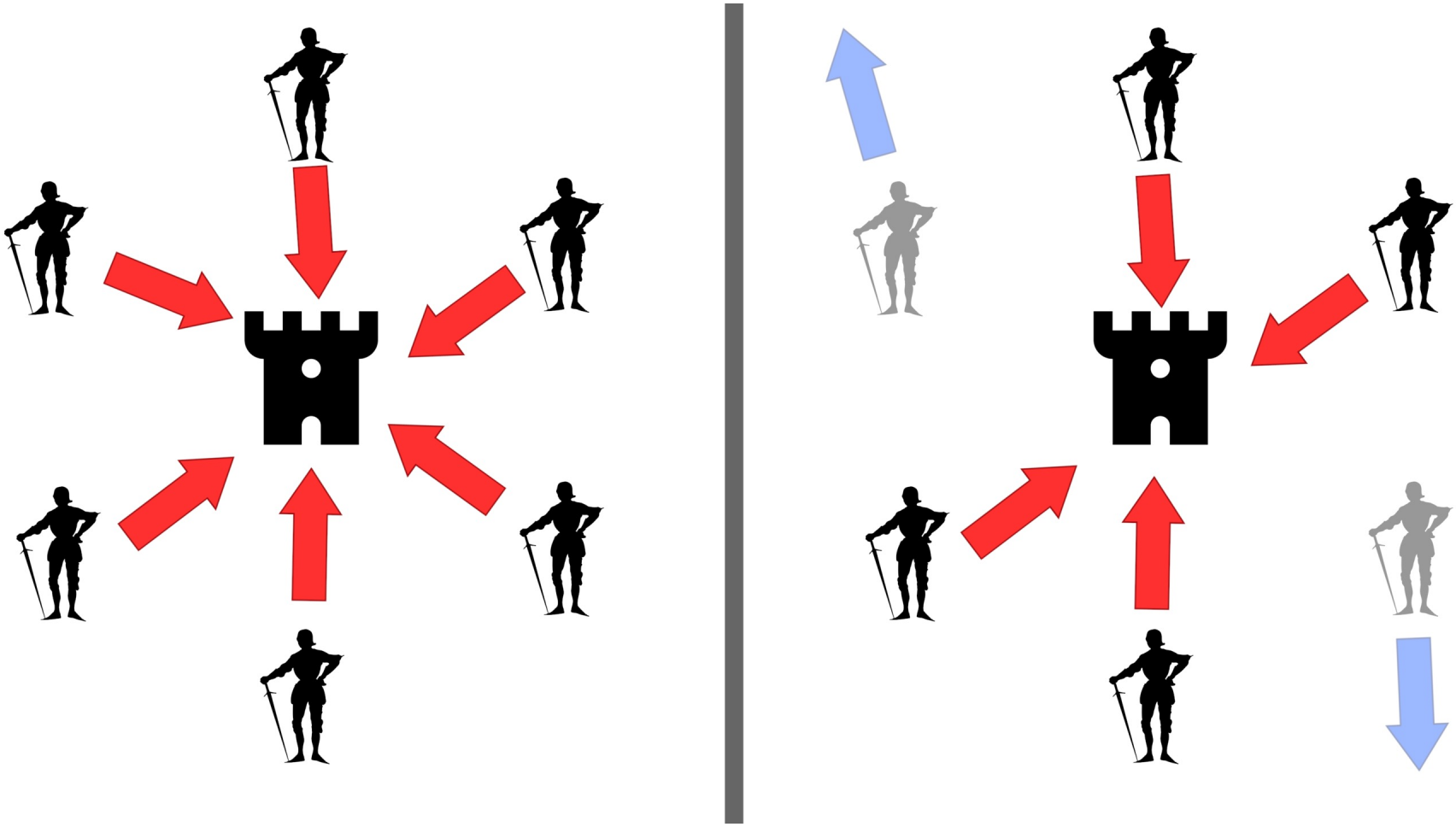
CHALLENGES OF COMMUNICATING ON AN UNRELIABLE CHANNEL

TWO ARMIES SEPARATED BY A HOSTILE VALLEY NEED TO COORDINATE THEIR ATTACK TO WIN. BUT THEIR MESSENGERS MAY NOT MAKE IT THROUGH WITH THEIR MESSAGES.



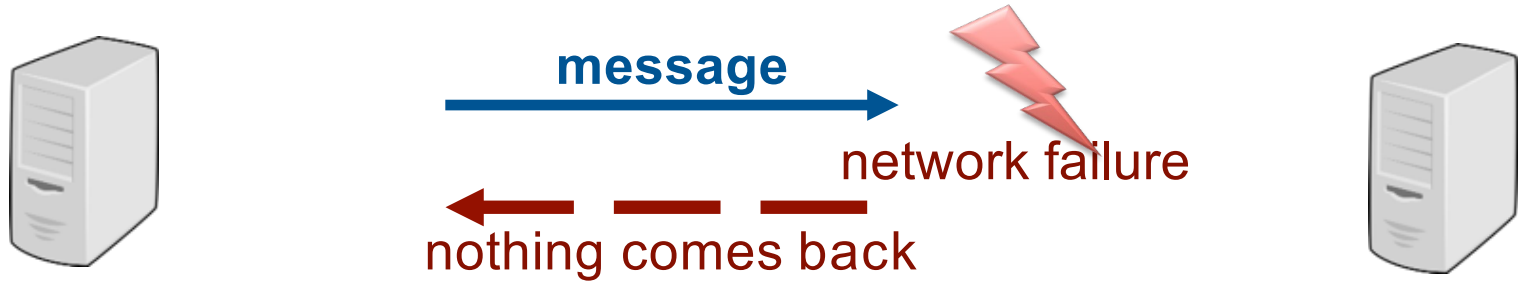
sketchplanations

The Byzantines Generals Problem



(Flash back) Difficulties

Partial failures make application writing difficult



Sender does not know:

- whether the message was received
- whether the receiver's process died before/after processing the message
- The message was corrupted

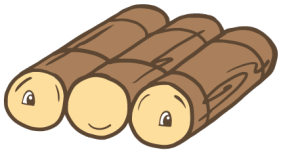
Types of Issues

- **Concurrency:** Handling simultaneous requests and ensuring that they are consistently ordered across all nodes.
- **Network partitions:** Nodes may become unreachable due to network issues, making consensus harder to achieve.
- **Fault tolerance:** The system must tolerate a certain number of node failures (crashes, restarts) while maintaining consensus.
- **Byzantine faults:** Malicious or arbitrary node behavior can disrupt consensus.

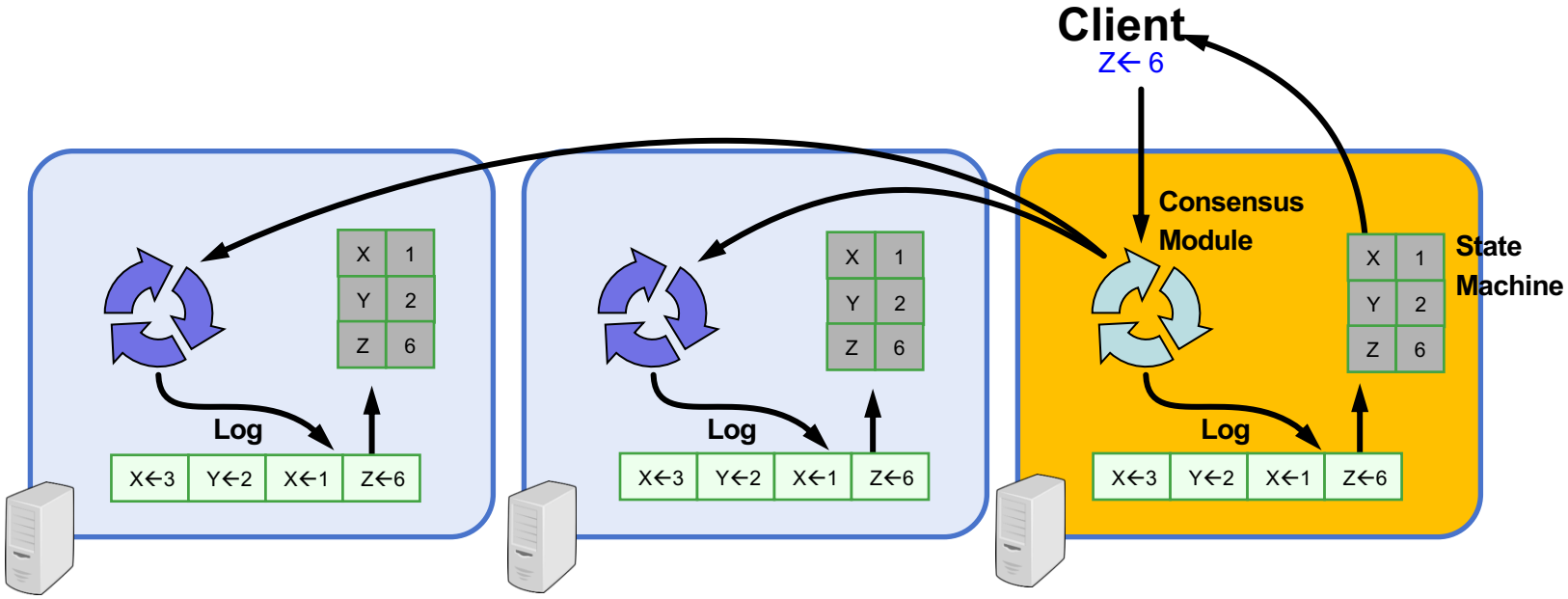
[Paper] Diego Ongaro, and John Ousterhout. "In search of an understandable consensus algorithm." *2014 USENIX*

Adapted material from: https://raft.github.io/

RAFT



Replicated Logs



- Replicated log => replicated state machine
  - All servers execute same commands in same order
- Consensus module ensures proper log replication
- System makes progress as long as any majority of servers are up
- Failure model: fail-stop servers, delayed/lost messages, not Byzantine

Important Raft Steps

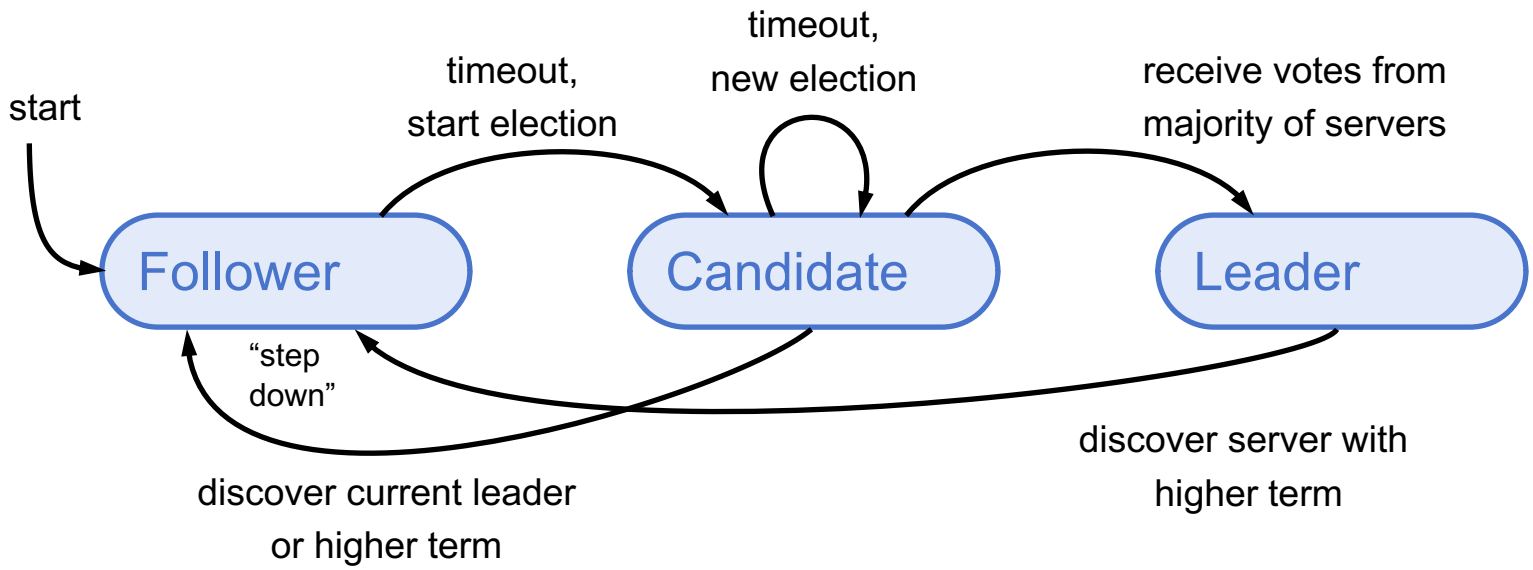
1. Leader Election
   - Select one of the servers to act as leader
   - Detect crashes, choose new leader
2. Log Replication
   - Leader accepting commands from clients, and appends them to its log
   - Leader initiate log replication to other servers (overwrite inconsistencies)
3. Safety
   - Keeping logs consistent
   - Only servers with up-to-date logs can become leaders

Server States

- At any given time, each server is either:
  - Leader: handles all client interactions, log replication
    - At most 1 viable leader at a time
  - Follower: completely passive (issues no RPCs*, responds to incoming RPCs)
  - Candidate: used to elect a new leader
- Normal operation: 1 leader, N-1 followers

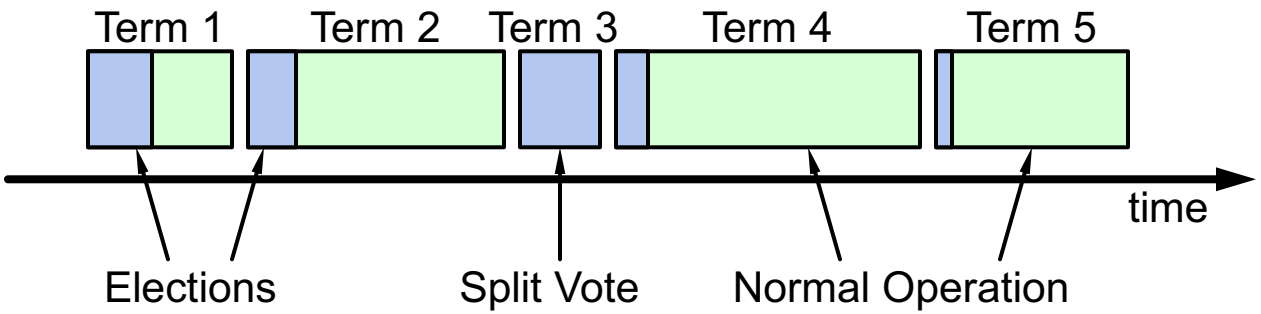
**RPC: Remote Procedure Call*

Server State Workflow



Terms

- Time divided into terms:
  - Election
  - Normal operation under a single leader
- At most 1 leader per term
- Some terms have no leader (failed election)
- Each server maintains current term value (no global view)
- Key role of terms: identify obsolete information
- Term begins when a new election initiated, which can happen due to:
  - leader failure, a network partition, or an election timeout.
- If a stable leader is present and no election timeouts occur, the term can continue indefinitely.

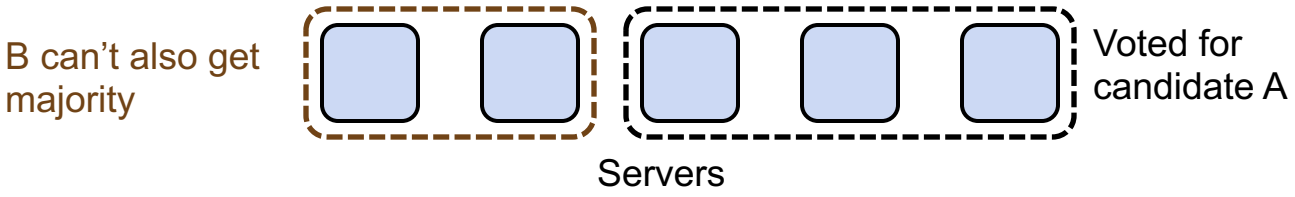


Election

- Increment current term
- Change to Candidate state
- Vote for self
- Send RequestVote RPCs to all other servers, retry until either:
  1. Receive votes from majority of servers:
     - Become leader
     - Send AppendEntries RPC heartbeats to all other servers
  2. Receive RPC from valid leader:
     - Return to follower state
  3. No-one wins election (election timeout elapses, split vote):
     - Increment term, start new election

Election Properties

- **Safety:** allow at most one winner per term
  - Each server gives out only one vote per term (persist on disk)
  - Two different candidates can't accumulate majorities in same term

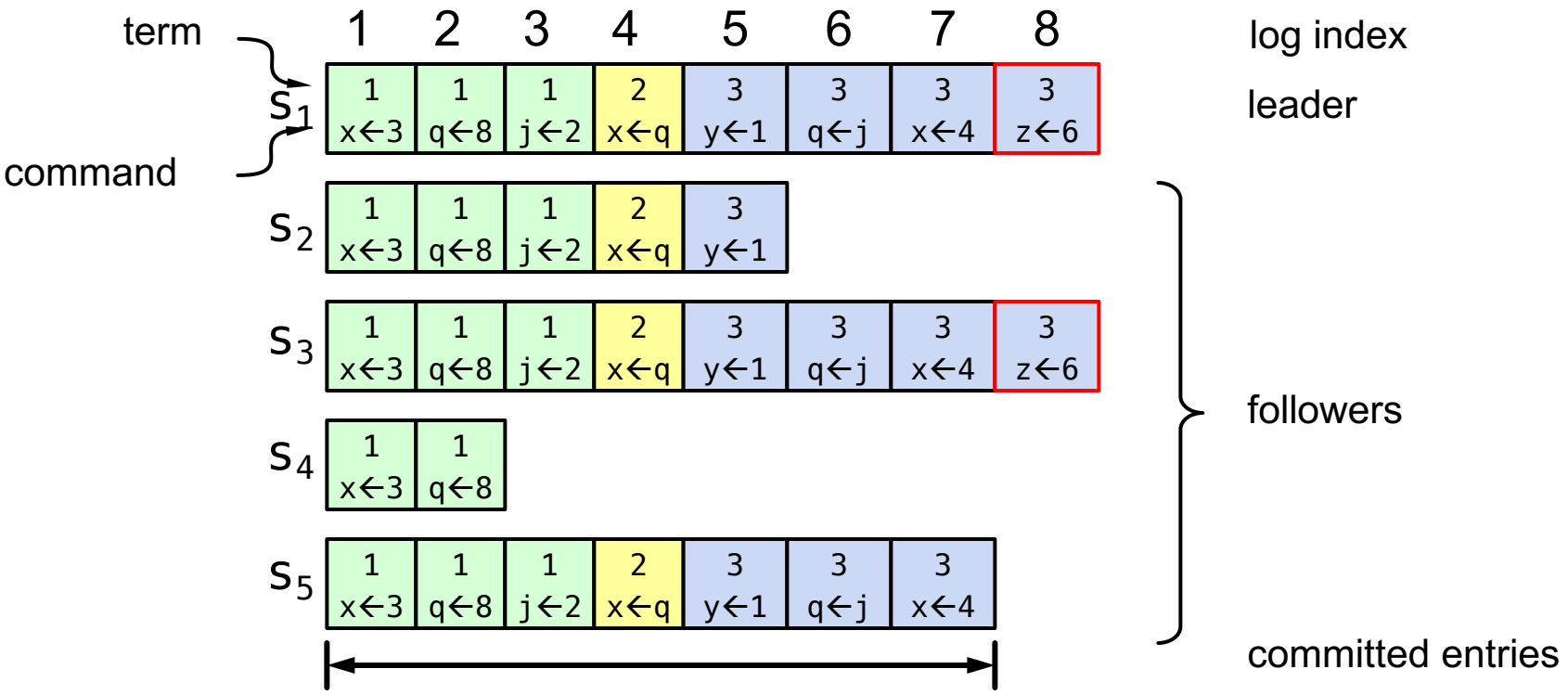


- **Liveness:** some candidate must eventually win
  - Choose election timeouts randomly in [T, 2T]
  - One server usually times out and wins election before others wake up
  - Works well if T >> Network Broadcast Time

Log Structure

- Log entry = index, term, command
- Log stored on stable storage (disk); survives crashes
- Entry committed if known to be stored on majority of servers
- Will eventually be executed by state machines
  - Example, checkpoint consolidation in HDFS
    - previous checkpoint + log

Log Structure

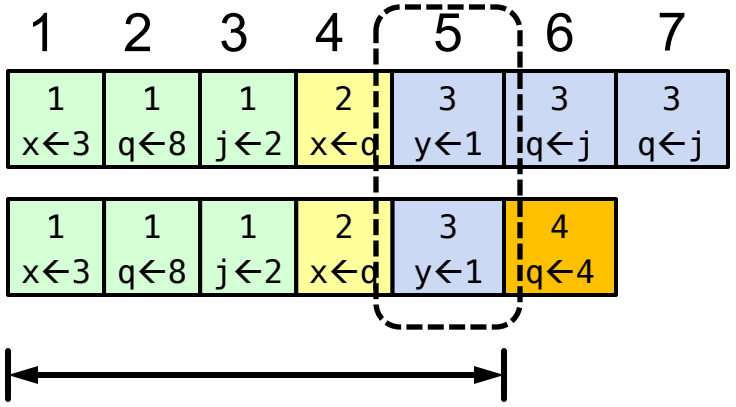


Normal Operations (during term)

- Client sends command to leader
- Leader appends command to its log
- Leader sends *AppendEntries* RPCs to followers
- Entry is committed if:
  - Replicated on majority of machines
- Once new entry committed:
  - Leader passes command to its state machine, returns result to client
  - Leader notifies followers of committed entries in subsequent AppendEntries RPCs
  - Followers pass committed commands to their state machines

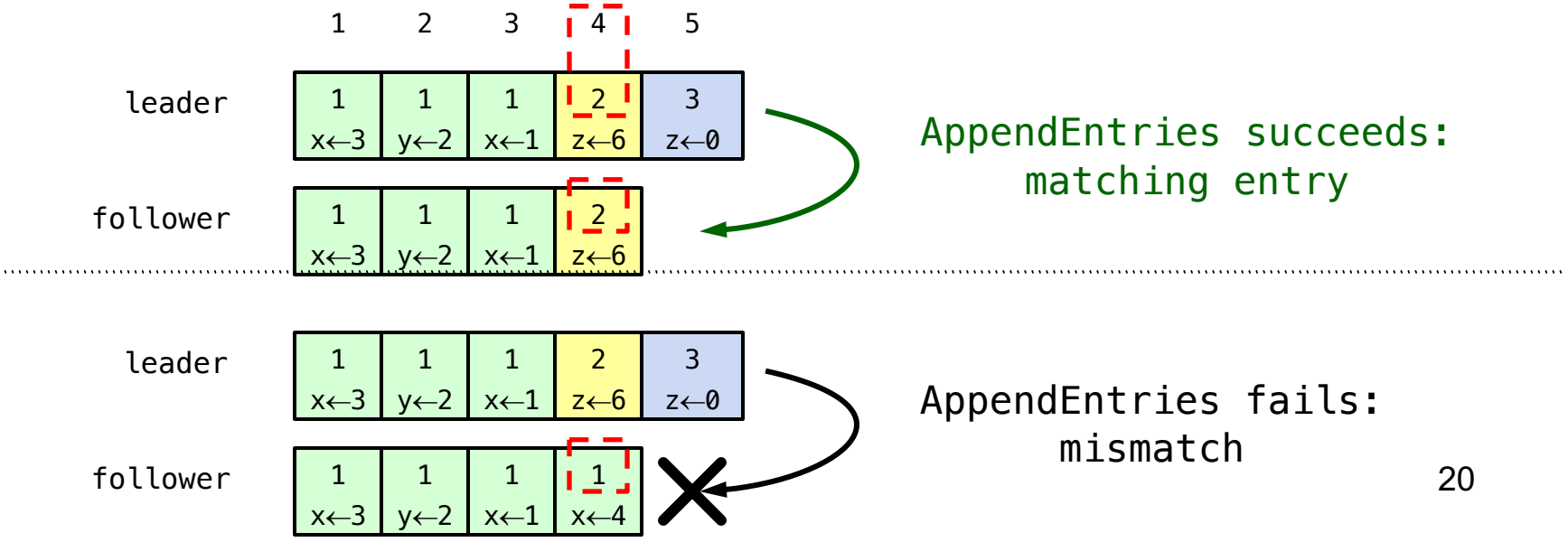
Log Matching Property

- If log entries on different servers have the same index and term
  - They store the same command sequence
  - The log are identical in all preceeding entries
- If an entry is committed, all preceeding entries are commited



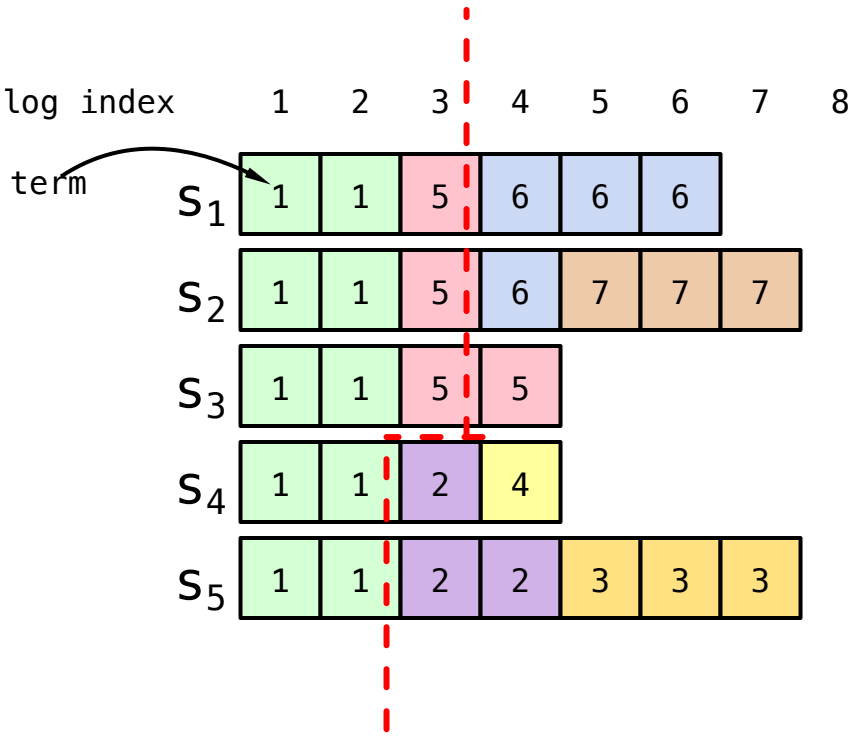
AppendEntries Consistency Check

- Each *AppendEntries* RPC contains index, term of entry **preceeding** new ones
- Follower must contain matching entry; otherwise it rejects request
- Implements an **induction step**, ensures coherency



Leader Changes

- Logs may be inconsistent after leader change
- At beginning of **new leader's** term:
  - Old leader may have left entries partially replicated
  - No special steps by new leader: just start normal operation
  - **Leader's log is "the truth"**
  - Will **eventually** make follower's logs identical to leader's
  - Multiple crashes can leave many **extraneous** log entries (see figure)

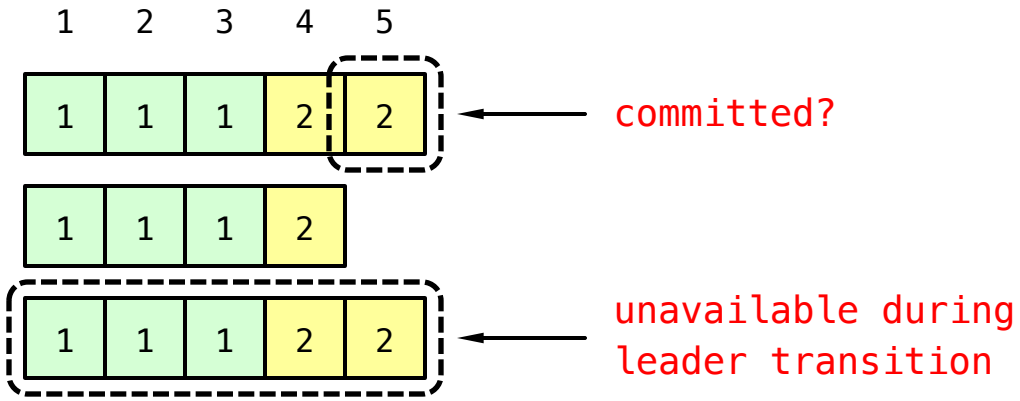


Safety Requirement

Once a log entry has been applied to a state machine, no other state machine must apply a different value for that log entry

- Raft safety property:
  - If a leader has decided that a log entry is committed, that entry will be present in the logs of all future leaders
- This guarantees the safety requirement
  - Leaders never overwrite entries in their logs
  - Only entries in the leader's log can be committed
  - Entries must be committed before applying to state machine

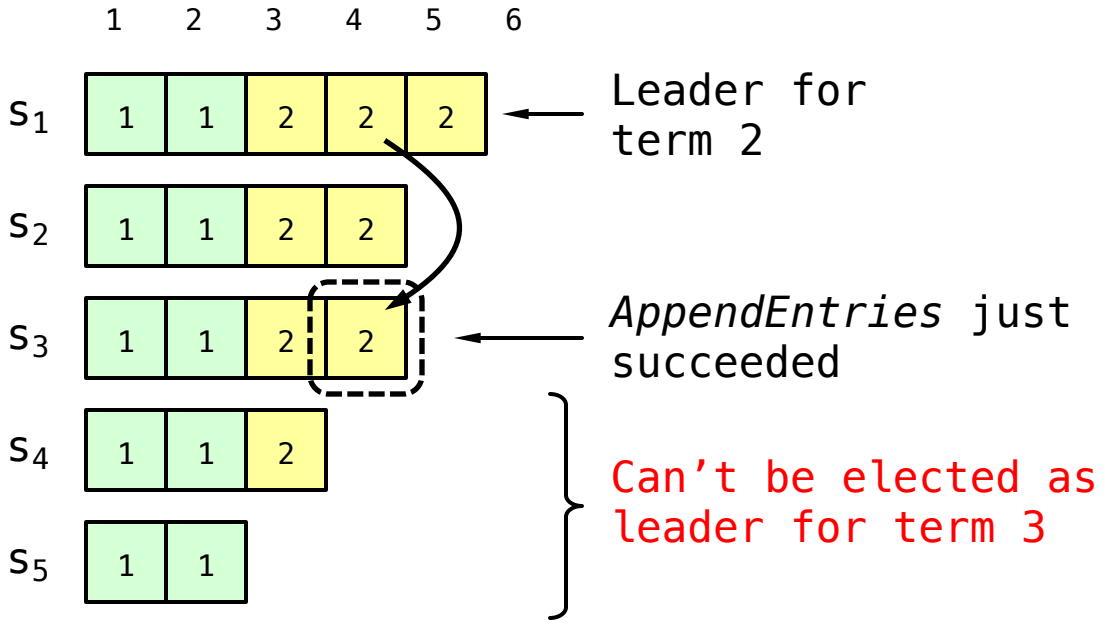
→ New Election Rule
Picking the **Best Leader**



- During elections, choose candidate with log most likely to contain all committed entries
  - Candidates include log info in RequestVote RPCs (index & term of last log entry)
  - Voting server V denies vote if its log is “more complete”:
    - ($lastTerm_V > lastTerm_C$) OR
    - ($lastTerm_V == lastTerm_C$) AND ($lastIndex_V > lastIndex_C$)
  - Leader will have “most complete” log among electing majority

Committing Entry from **Current Term**

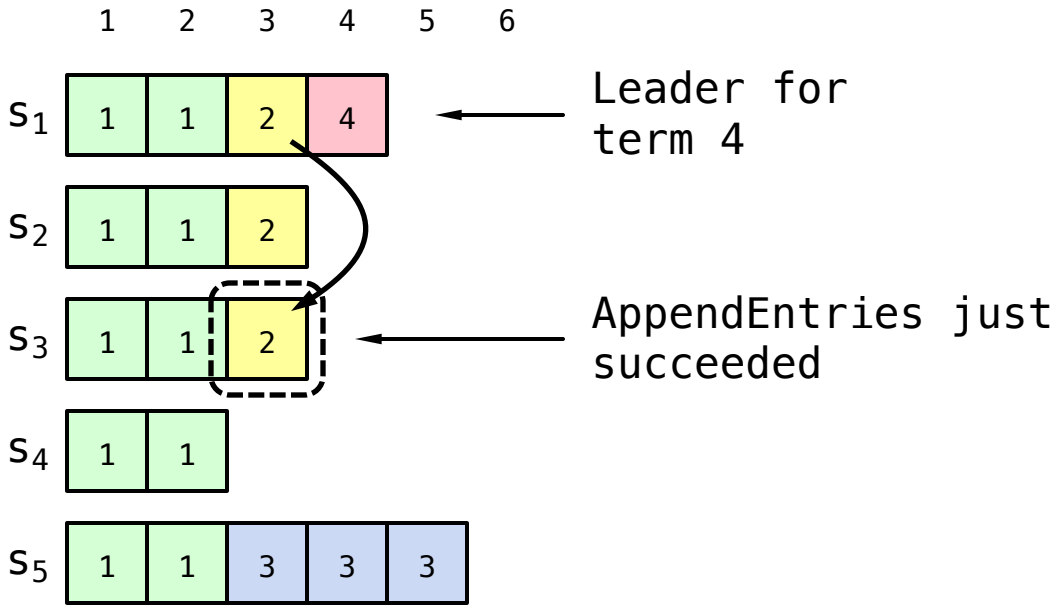
- Case 1: leader decides entry in current term is committed



- Safe: Leader for term 3 must contain entry ***idx 4***

Committing Entry from **Earlier Term**

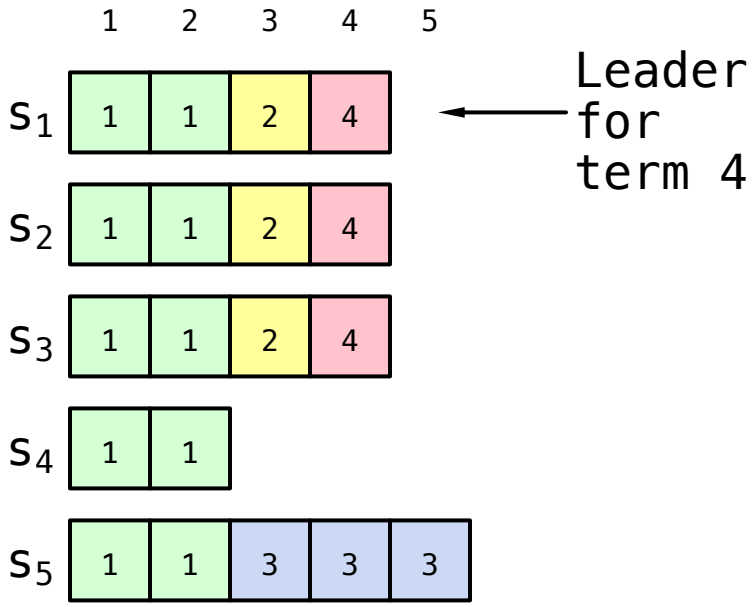
- Case 2: Leader is trying to finish committing entry from an earlier term



- Entry 3 not safely committed:
  - S_5 can be elected as leader for term 5
  - If elected, it will overwrite entry 3 on s_1, s_2, and s_3!

→ New Commitment Rules

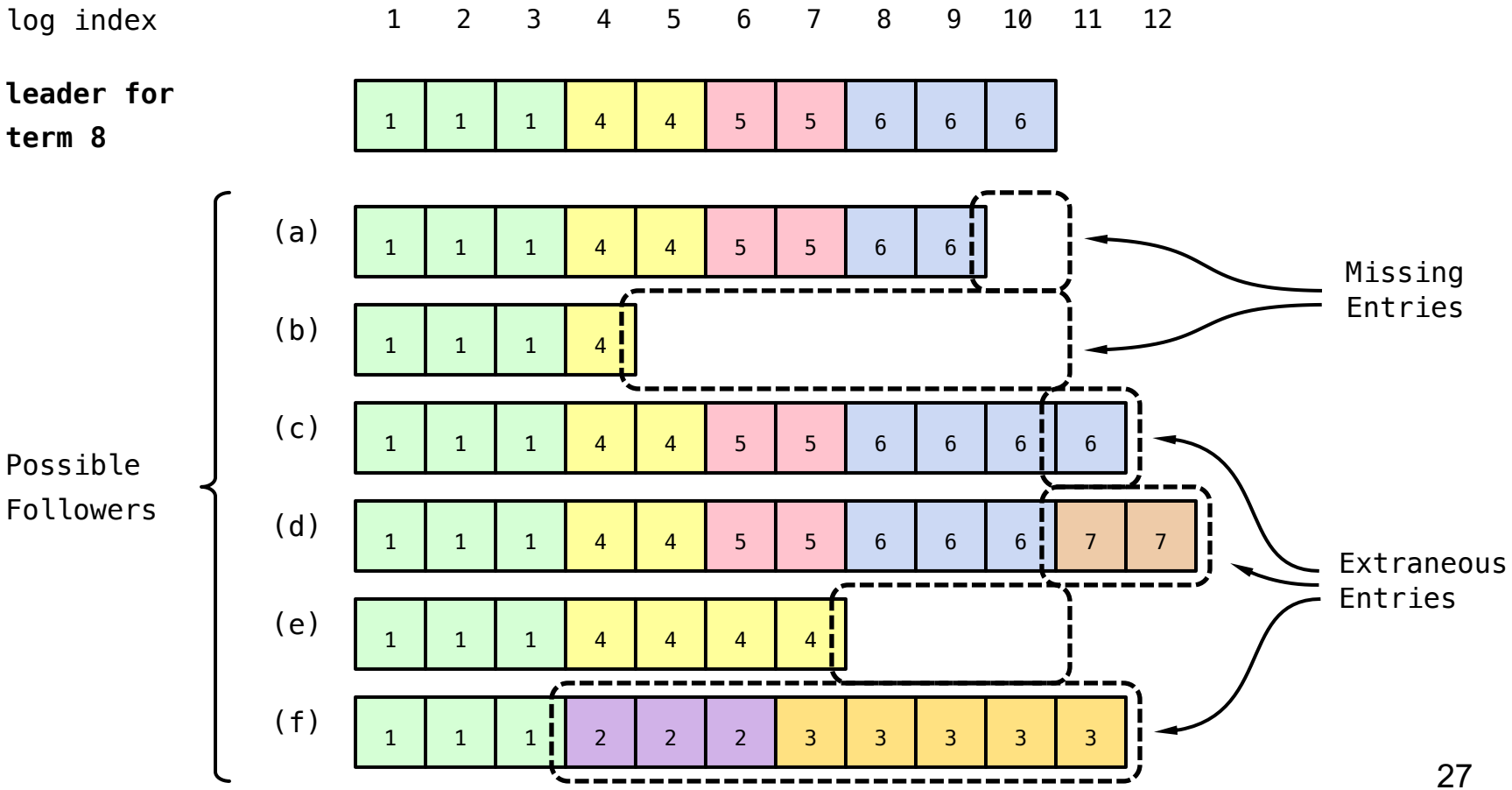
- For a leader to decide an entry is committed:
  - Entry must be stored on a majority of servers
  - At least one new entry from leader's term must also be stored on majority of servers
- Safe: Once entry 4 committed:
  - S_5 cannot be elected leader for term 5
  - Entries 3 and 4 both safe



Combination of election rules and commitment rules makes Raft safe

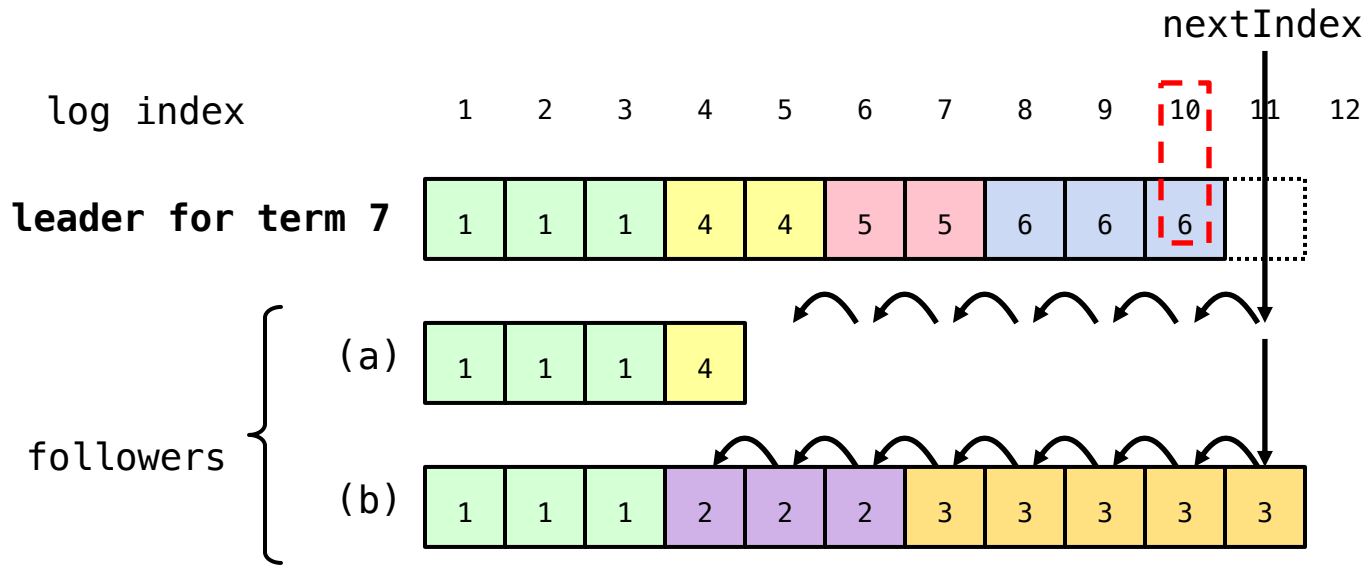
Log Inconsistencies

- Leader changes can result in log inconsistencies



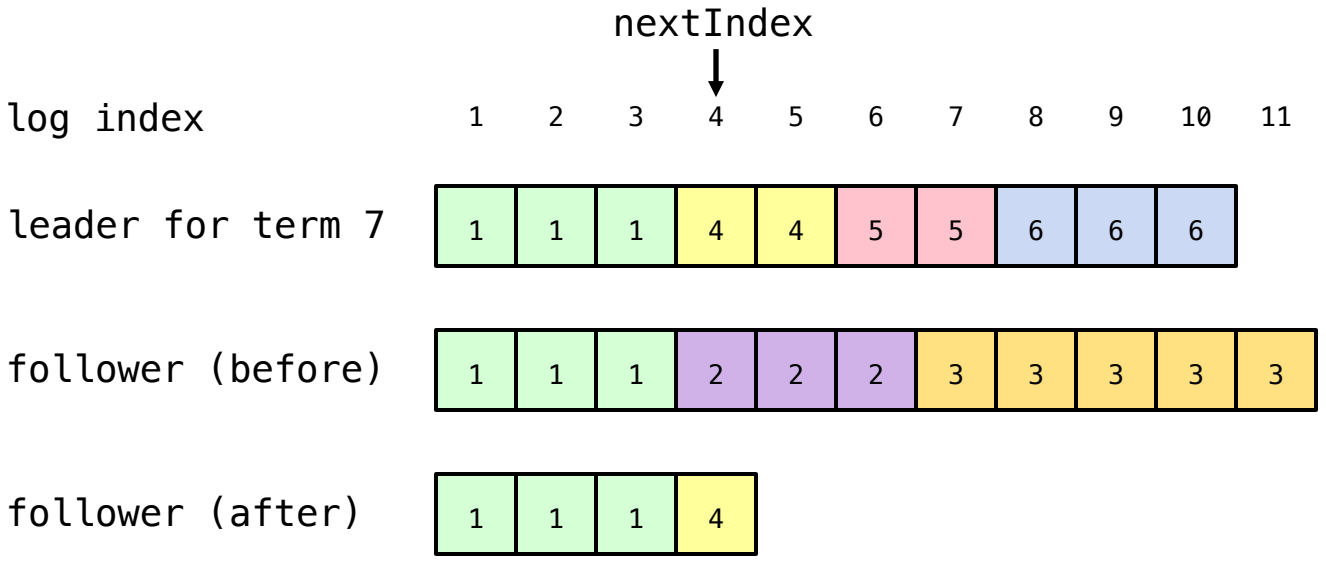
Repairing Follower Logs

- New leader must make follower logs consistent with its own
  - Delete extraneous entries
  - Fill in missing entries
- Leader keeps nextIndex for each follower:
  - Index of next log entry to send to that follower
  - Initialized to (1 + leader's last index)
- When AppendEntries consistency check fails, decrement nextIndex and try again:



Repairing Follower Logs

- When follower overwrites inconsistent entry, it deletes all subsequent entries:



Neutralizing Old Leaders

- Deposed leader may not be dead:
  - Temporarily disconnected from network
  - Other servers elect a new leader
  - Old leader becomes reconnected, attempts to commit log entries
- Terms used to detect stale leaders (and candidates)
  - Every RPC contains term of sender
  - If sender's term is older, RPC is rejected, sender reverts to follower and updates its term
  - If receiver's term is older, it reverts to follower, updates its term, then processes RPC normally
- Election updates terms of majority of servers
  - Deposed server cannot commit new log entries

Client Protocol

- Send commands to leader
  - If leader unknown, contact any server
  - If contacted server not leader, it will redirect to leader
- Leader does not respond until command has been logged, committed, and executed by leader's state machine
- If request times out (e.g., leader crash):
  - Client reissues command to some other server
  - Eventually redirected to new leader
  - Retry request with new leader

Client Protocol (cont.)

- What if leader crashes after executing command, but before responding?
  - Must not execute command twice
- Solution: client embeds a unique id in each command
  - Server includes id in log entry
  - Before accepting command, leader checks its log for entry with that id
  - If id found in log, ignore new command, return response from old command
- Result: exactly-once semantics as long as client doesn't crash

Configuration Changes

- System configuration:
  - ID, address for each server
  - Determines what constitutes a majority
- Consensus mechanism must support changes in the configuration:
  - Replace failed machine
  - Change degree of replication

Demo Time

- Raft Scope
  - https://raft.github.io/raftscope/index.html

- Didactic tutorial:
  - http://thesecretlivesofdata.com/raft/