Distributed System : Concensus

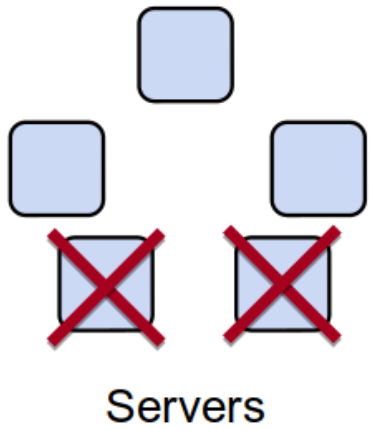
Houmin Wei

Electronics Engineering & Computer Science
Peking University

November 26, 2017

What is Consensus?

- Agreememt on shared state(single system image)
- Recovers from server failure autonomously
  - Minority of servers fail: no problem
  - Majority fail: lose availability, retain consistency

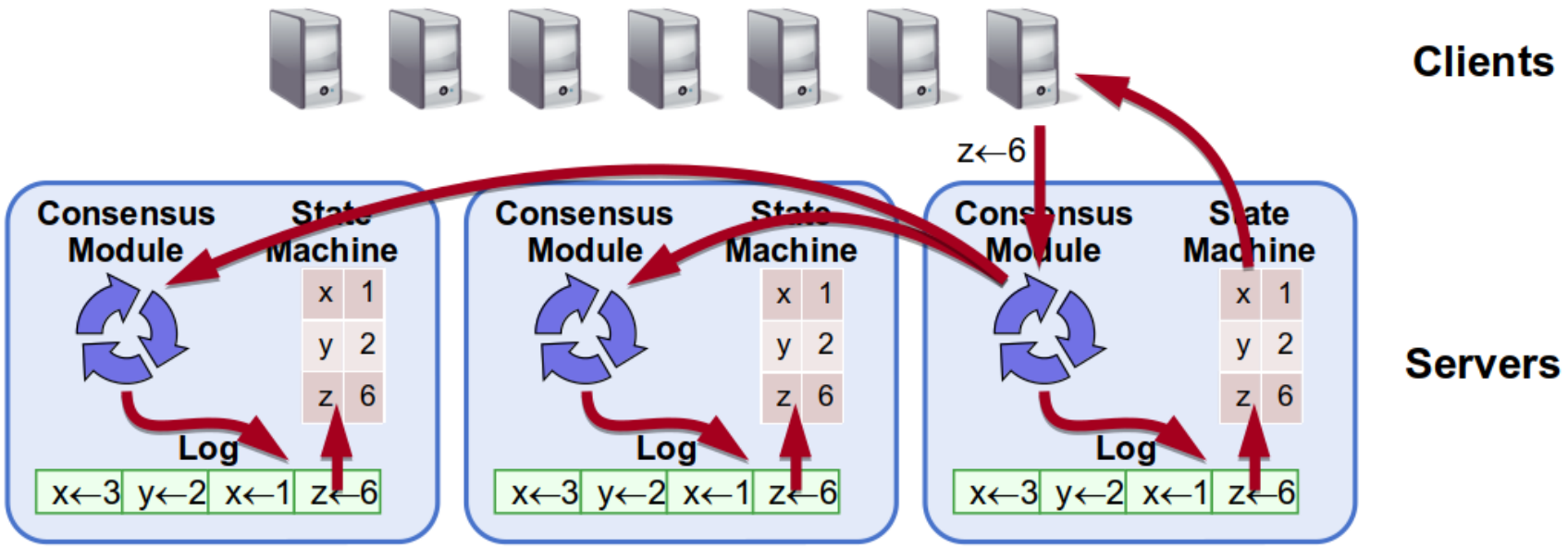


- Key to building consistent storage systems.

To eliminate single point of failure: Replication

- Network partition or server down
- Consensus
  - Allows collection of machines to work as coherent group
  - Continuous service, even if some machines fail
- A concensus algorithm(built-in or library)
  - Paxos(1990) has dominated discussion for 25 years, hard for engineer.
  - Raft(2013) is easier to understand.
  - ...
- A concensus service
  - Google Chubby
  - Apache ZooKeeper
  - ...

Replicated State Machine(RSM)



- Replicated log => 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(not Byzantine), delayed/lost messages.

Raft Overview

- Leader election
  - Select one of servers to act as cluster leader
  - Detect crashes, choose new leader
- Log replication
  - Leader takes commands from clients, appends them to its log.
  - Leader replicates its log to other servers(overwriting inconsistencies)
- Safety: Only a server with an up-to-date log can become leader.

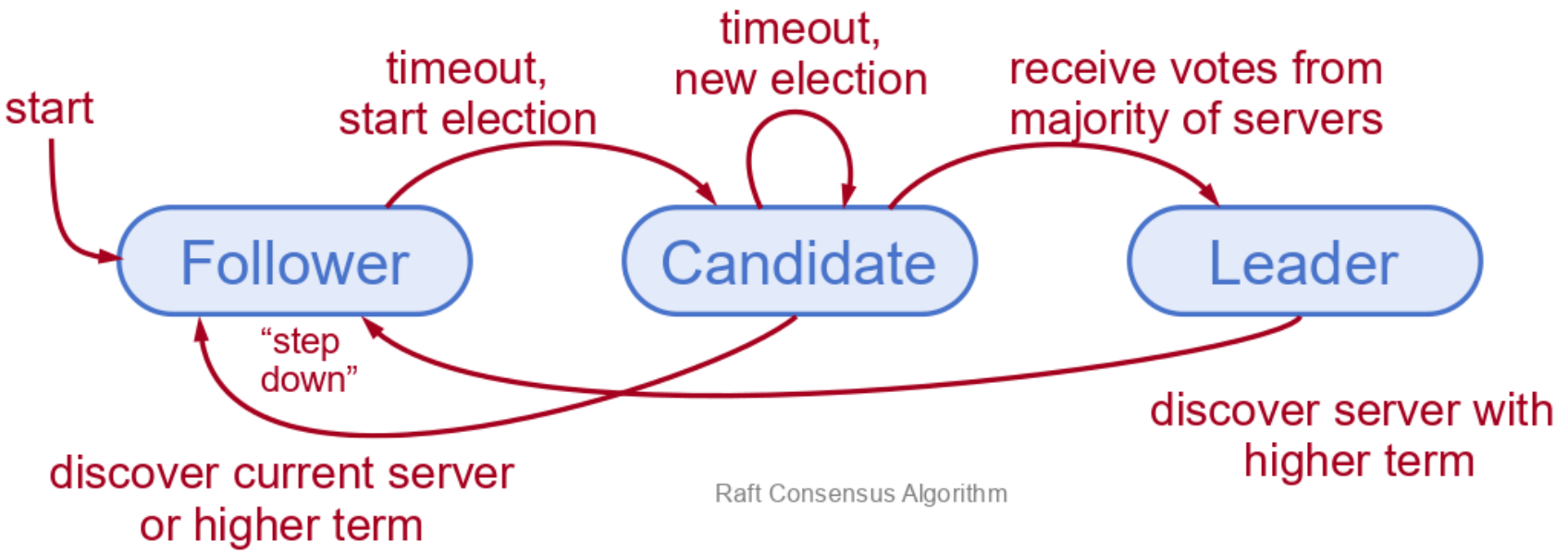
Raft Visualization

Core Raft Overview

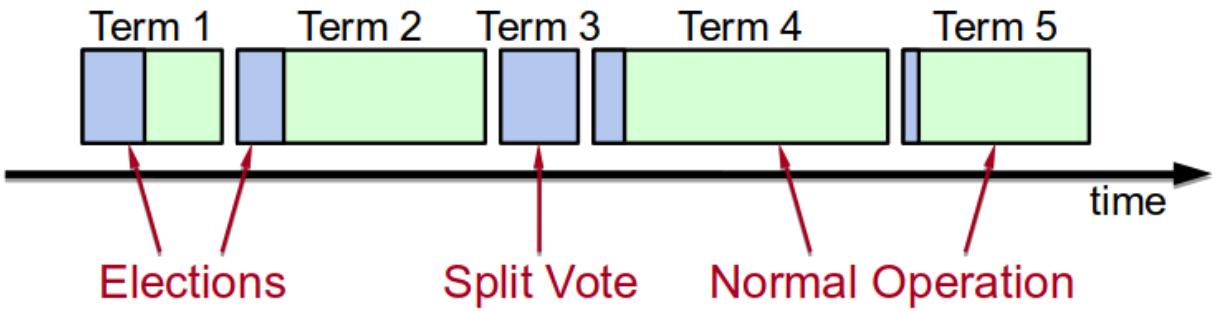
- Leader election
  - Heartbeats and timeouts to detect crashes
  - Randomized timeouts to avoid split votes
  - Majority voting to guarantee at most one leader per term
- Log replication
  - Leader takes commands from clients, appends them to its log.
  - Leader replicates its log to other servers(overwriting inconsistencies)
- Safety
  - Only elect leaders with all committed entries in their logs.
  - New leader defers committing entries from prior terms.

Server States

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



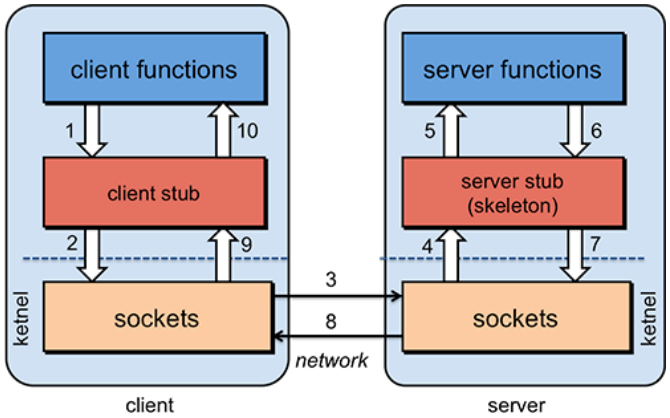
Terms



- **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**
- **Key role of terms: identify obsolete information**

Raft Remote Procedure Calls(RPCs)

- **Raft servers using RPCs to communicate**
  - RPC is a key piece of distribute system machinery
  - RPC ideally make net communication look just like function call



- **RequestVote RPC**
  - Invoked by candidate to gather votes
- **AppendEntries RPC**
  - Invoked by leader to replicate log entries
  - Also used as heartbeats

RequestVote RPC

RequestVote RPC

Invoked by candidates to gather votes (§5.2).

Arguments:

term	candidate's term
candidateId	candidate requesting vote
lastLogIndex	index of candidate's last log entry (§5.4)
lastLogTerm	term of candidate's last log entry (§5.4)

Results:

term	currentTerm, for candidate to update itself
voteGranted	true means candidate received vote

Receiver implementation:

1. Reply false if term < currentTerm (§5.1)
2. If votedFor is null or candidateId, and candidate's log is at least as up-to-date as receiver's log, grant vote (§5.2, §5.4)

AppendEntries RPC

AppendEntries RPC

Invoked by leader to replicate log entries (§5.3); also used as heartbeat (§5.2).

Arguments:

term	leader's term
leaderId	so follower can redirect clients
prevLogIndex	index of log entry immediately preceding new ones
prevLogTerm	term of prevLogIndex entry
entries[]	log entries to store (empty for heartbeat; may send more than one for efficiency)
leaderCommit	leader's commitIndex

Results:

term	currentTerm, for leader to update itself
success	true if follower contained entry matching prevLogIndex and prevLogTerm

Receiver implementation:

1. Reply false if term < currentTerm (§5.1)
2. Reply false if log doesn't contain an entry at prevLogIndex whose term matches prevLogTerm (§5.3)
3. If an existing entry conflicts with a new one (same index but different terms), delete the existing entry and all that follow it (§5.3)
4. Append any new entries not already in the log
5. If leaderCommit > commitIndex, set commitIndex = min(leaderCommit, index of last new entry)

Heartbeats and Timeouts

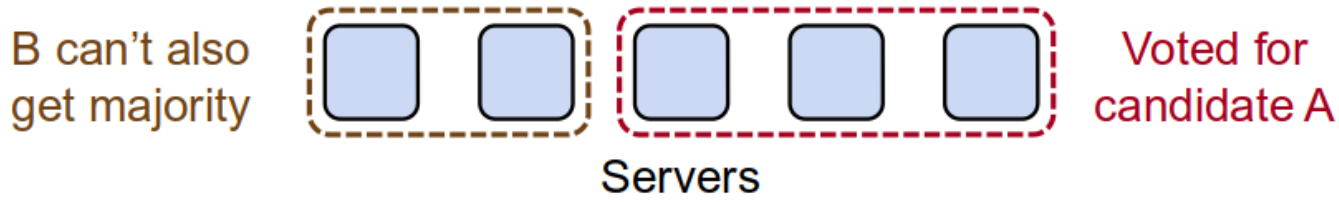
- **Servers start up as followers**
- **Followers expect to receive RPCs from leaders or candidates**
- **Leaders must send heartbeats(empty AppendEntries RPCs) to maintains authority**
- **If election Timeout elapses with no RPCs:**
  - Follower assumes leader has crashed
  - Follower starts new election
  - Timeouts typically 100-500 ms

Election Basics

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

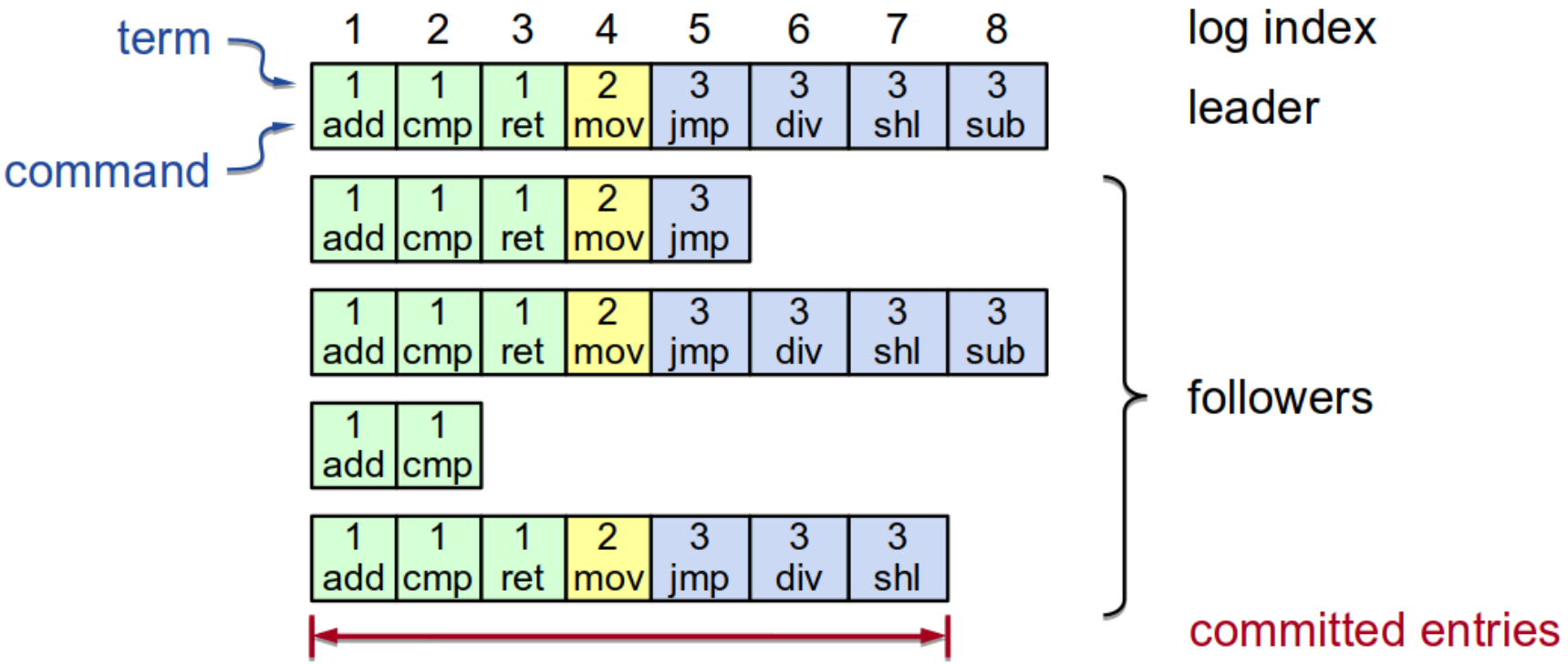
Elections, cont'd

- **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 sever usually times out and wins election before others wake up
  - Works well if T $\gg$ 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
  - Durable, will eventually be executed by state machines

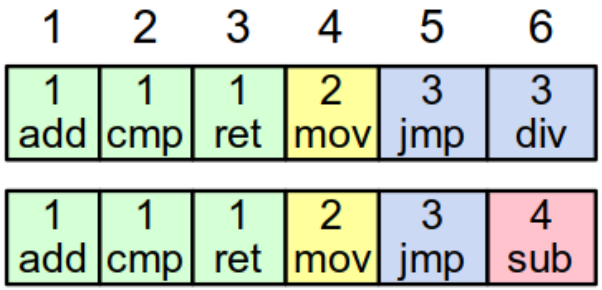
Normal Operation

- **Client sends command to leader**
- **leader appends command to its log**
- **Leader sends AppendEntries RPCs to followers**
- **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
- **Crashed/slow followers?**
  - Leader retries RPCs until they succeed
- **Performance is optimal in common case:**
  - One successful RPC to any majority of servers

Log Consistency

High level of conherency between logs:

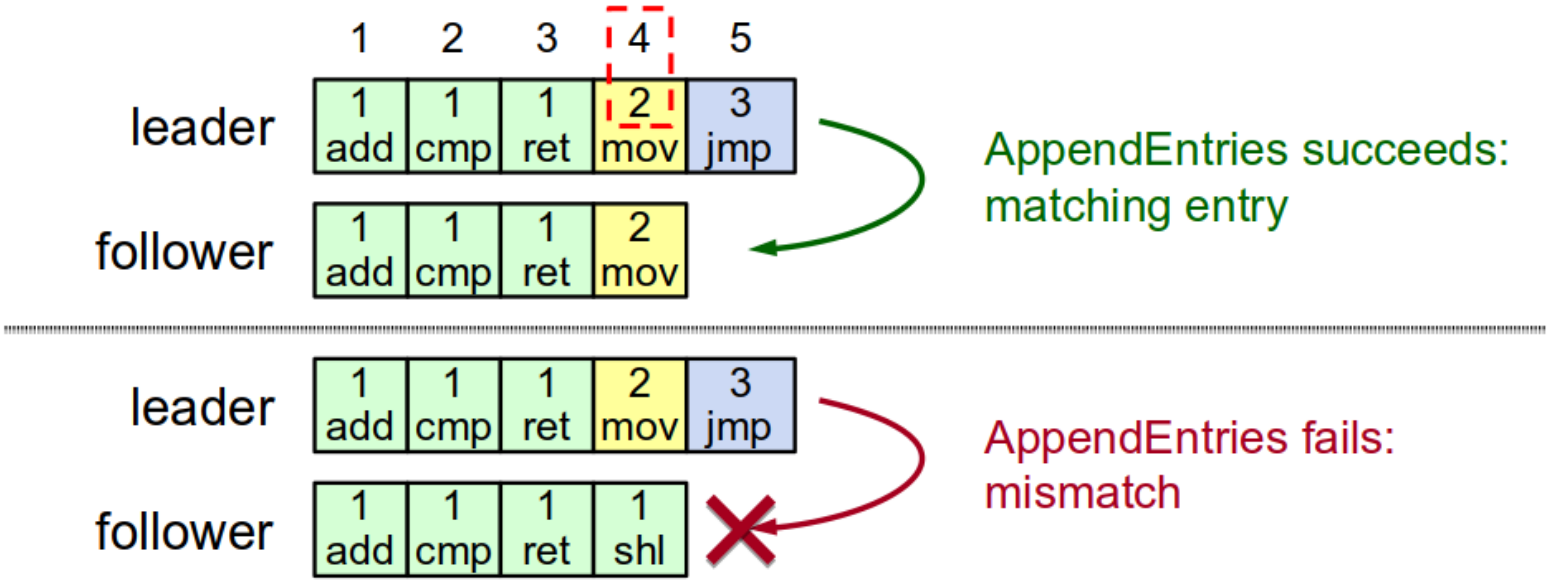
- If log entries on different servers have same index and term:
  - They store the same command
  - The logs are identitcal in all preceding entries



- If a given entry is committed, all preceding entries are also committed

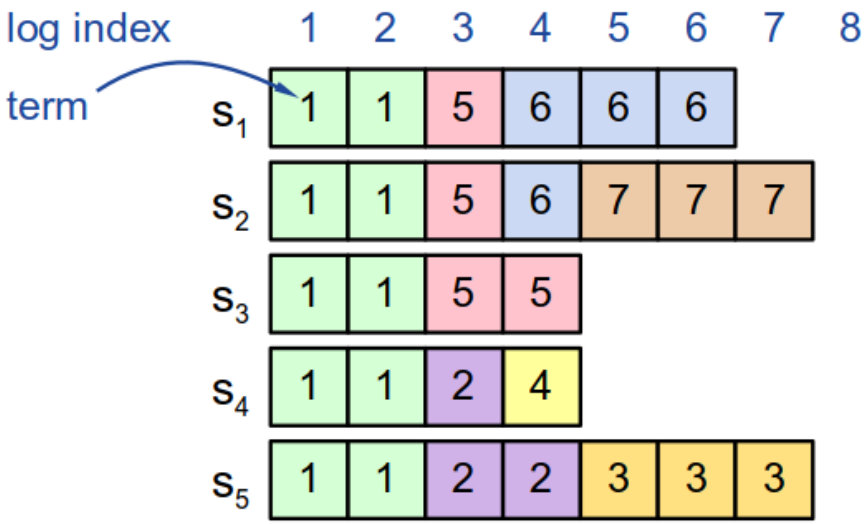
AppendEntries Consistency Check

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



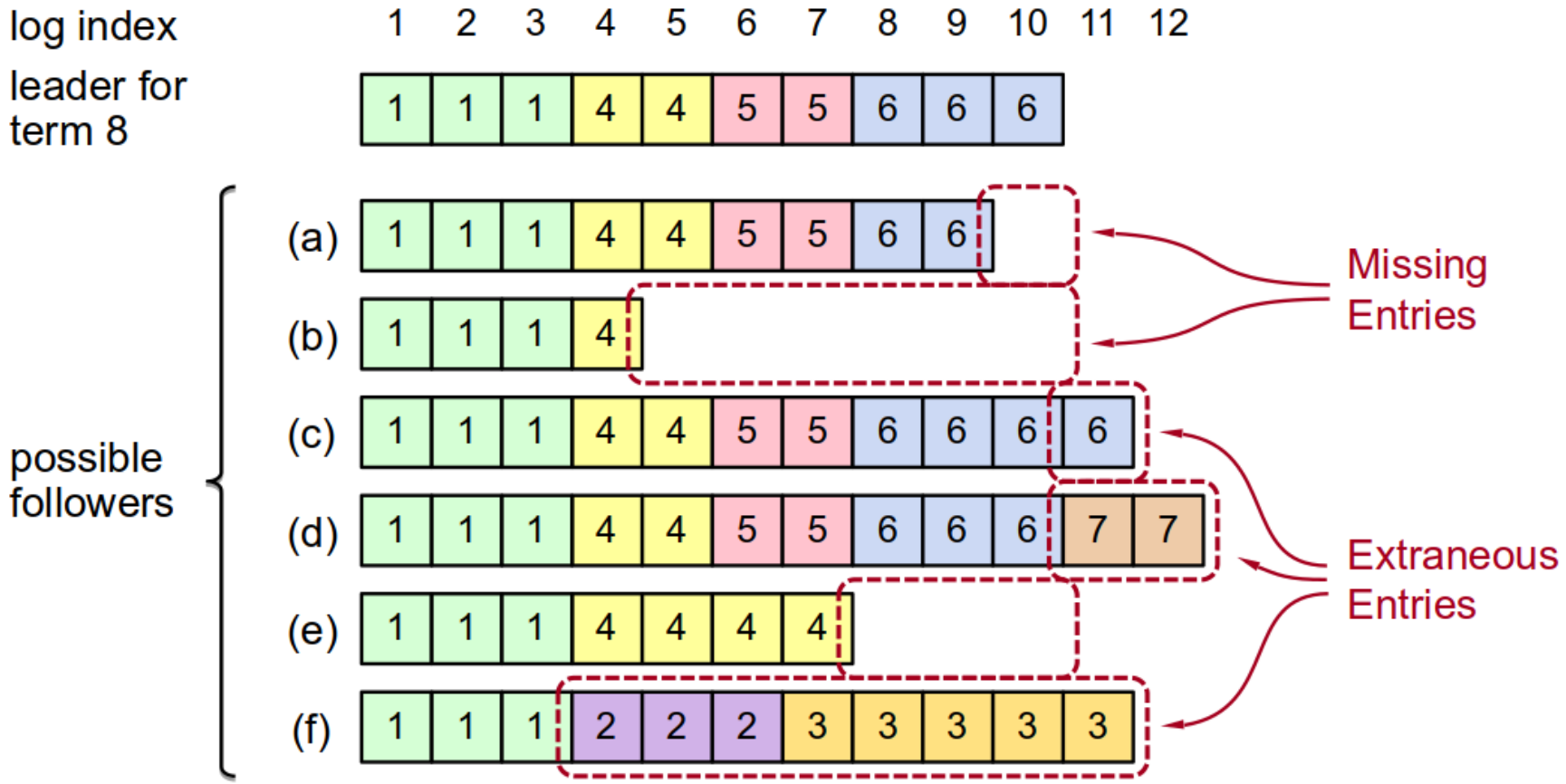
Leader Changes

- At begining 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:



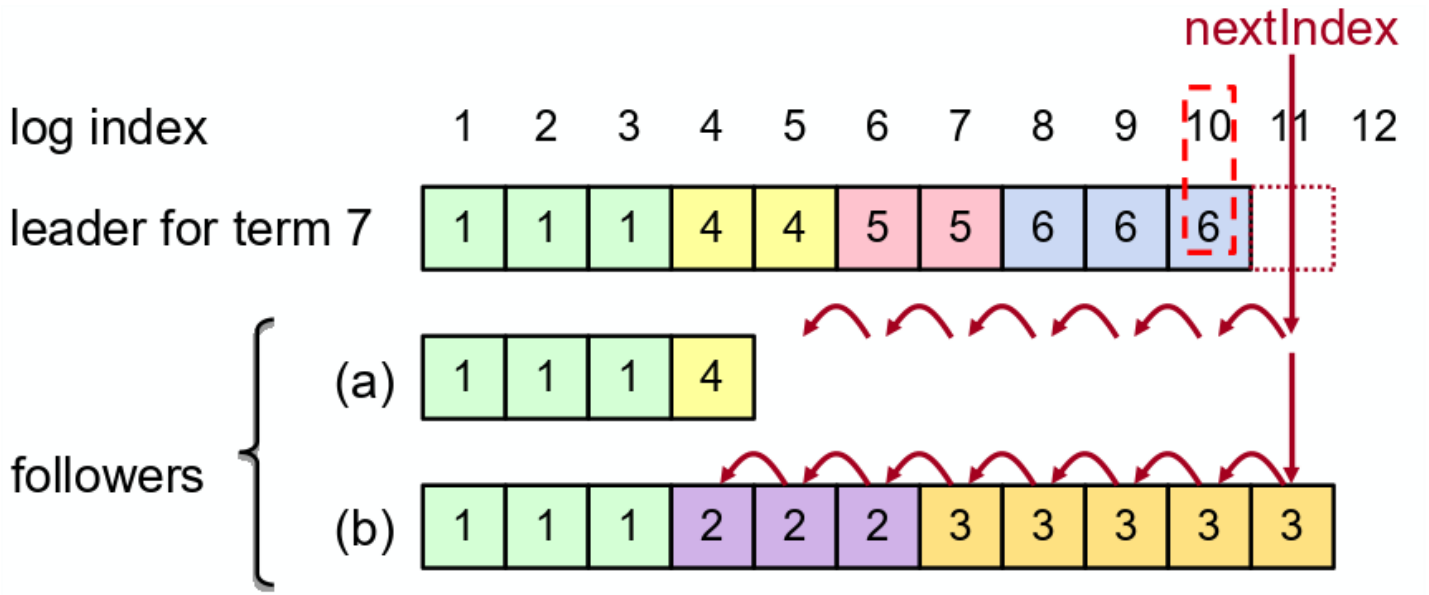
Log Inconsistencies

Leader changes can result in log inconsistencies:



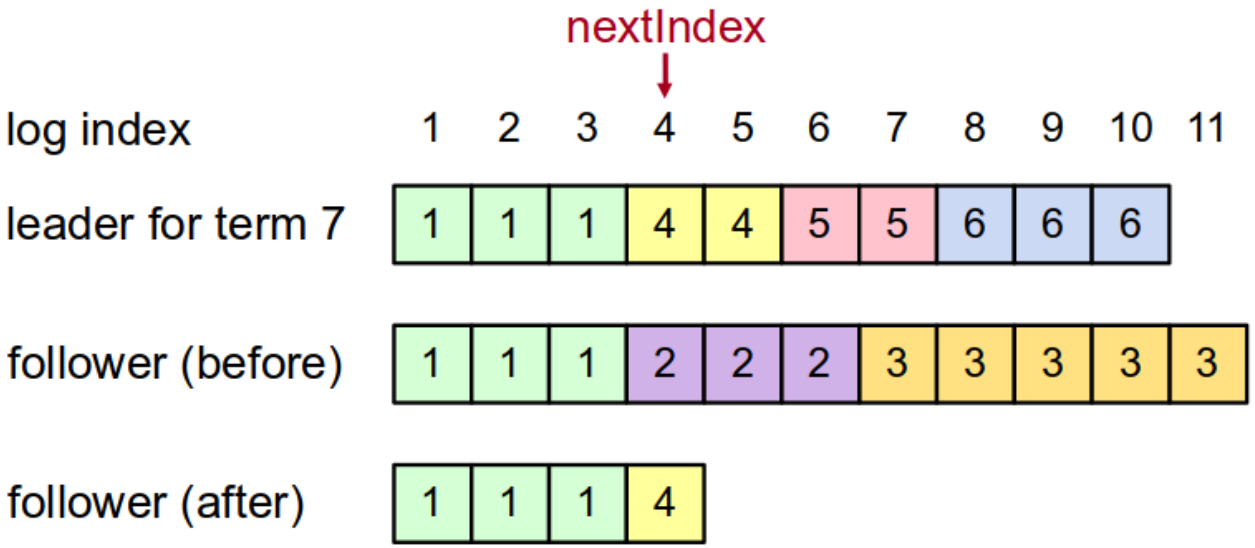
Repairing Follower Logs

- **New leader must make follower logs consistent with its own**
  - Delelte 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 Logs, cont'd

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



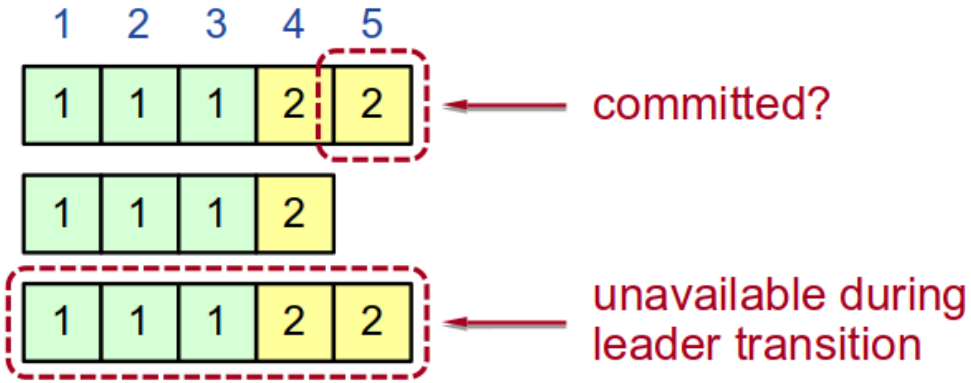
Safety Requirement

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

Picking the Best Leader

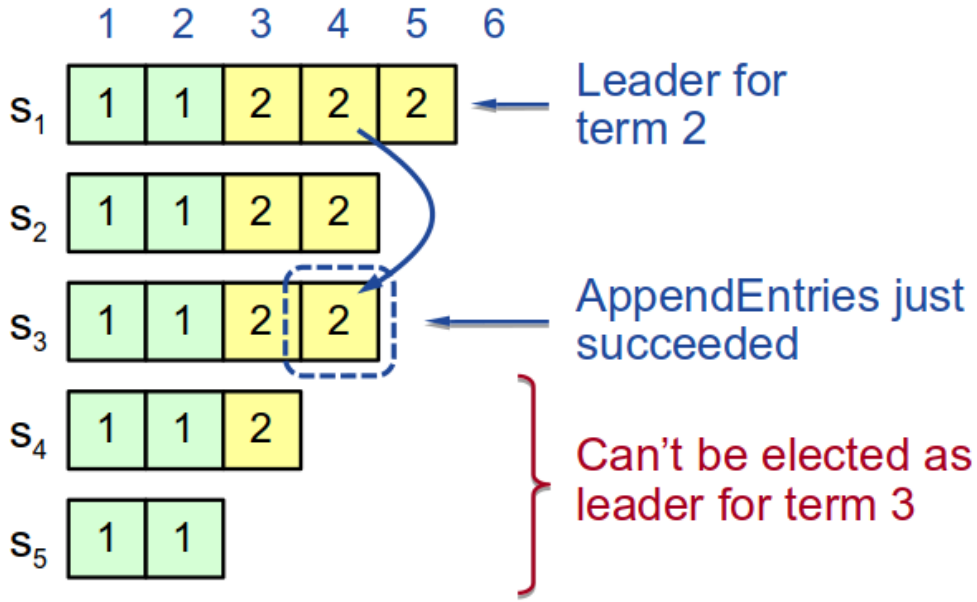
- **Can't tell which entries are committed!**



- **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':
    $(lastTerm_V > lastTerm_C) \|$
    $(lastTerm_V == lastTerm_C) \&\& (lastIndex_V > lastIndex_C)$
  - Leader will have 'most complete' log among electing majority

Committing Entry from Current Term

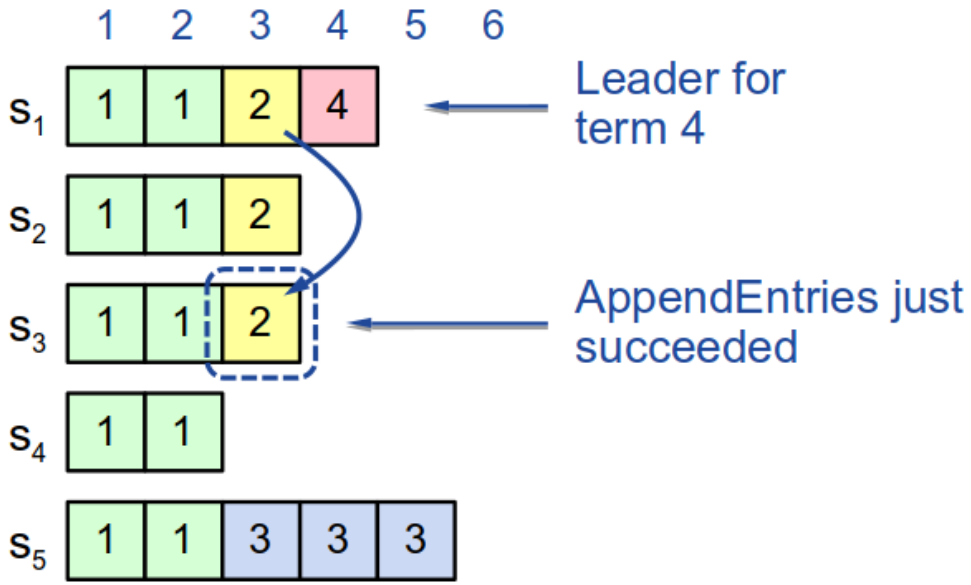
- **Case 1/2: Leader decides entry in current term is committed**



- **Safe: leader for term 3 must contain entry 4**

Committing Entry from Earlier Term

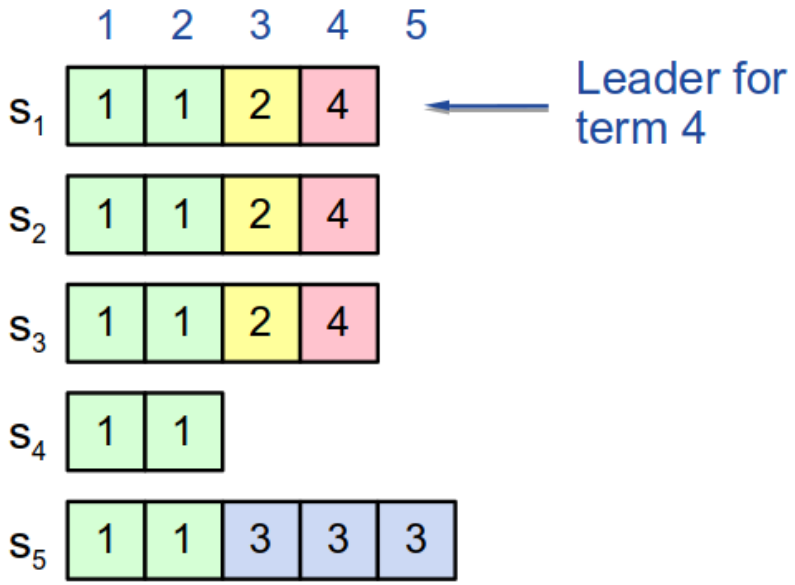
- **Case 2/2: Leader is trying to finish committing entry from an earlier term**



- **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:**
  - 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
- **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

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 redirected 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'd

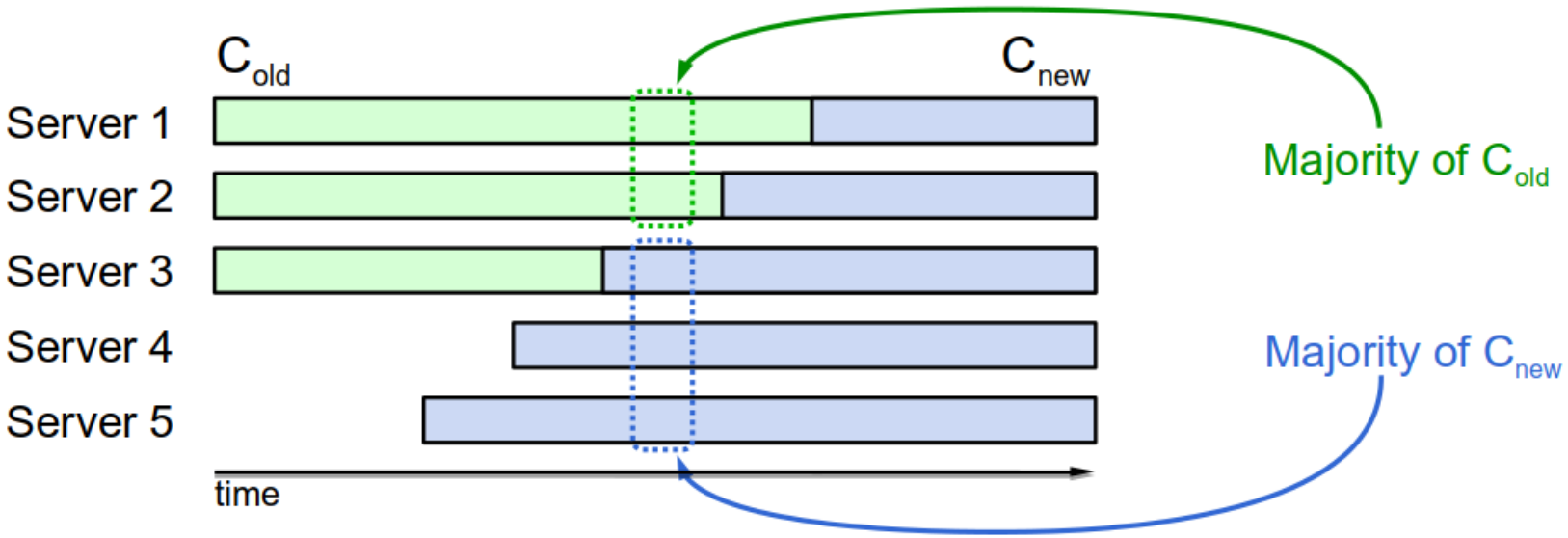
- **What if leader crashes after executing command, but before responding?**
  - Must not execute command twice
- **Solution: client embeds a unique id in each command**
  - Server includes id in log entry
  - Before accepting command, leader checks its log for entry with that id
  - if id found in log, 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

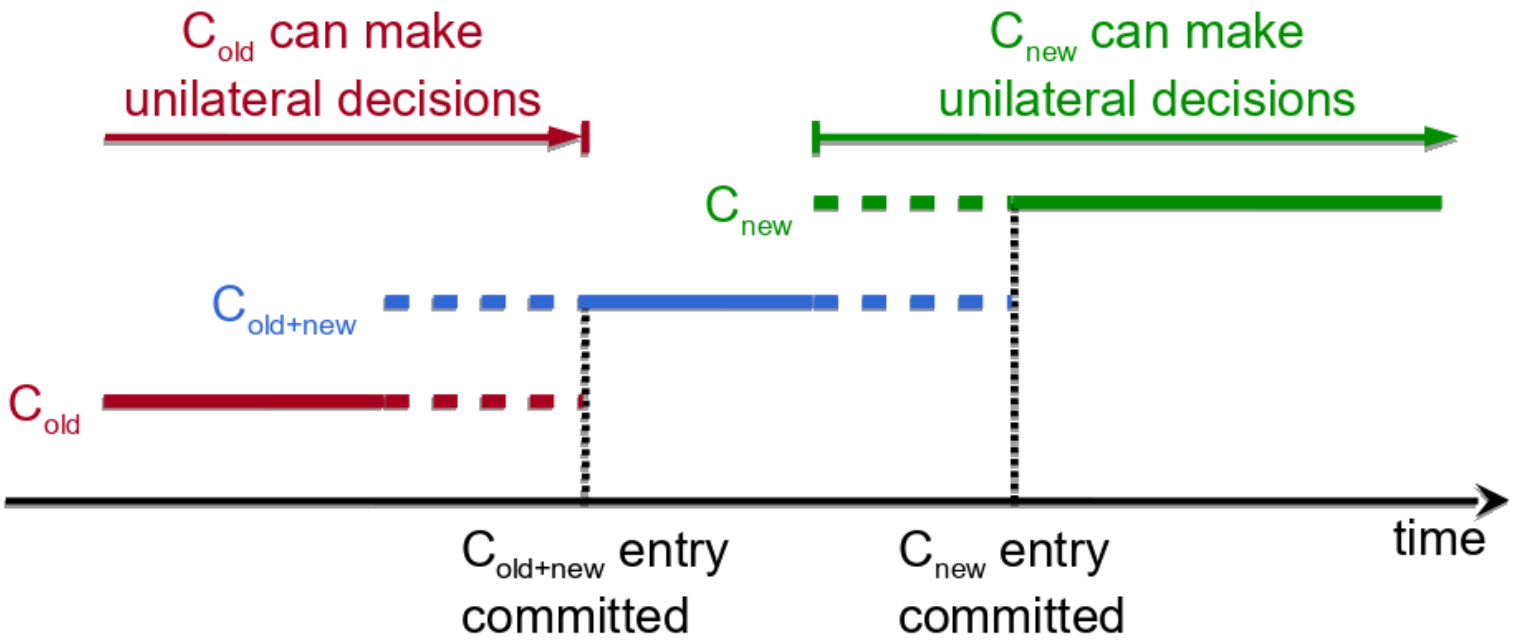
Configuration Changes, cont'd

Cannot switch directly from one configuration to anther: conflicting majorities could rise



Joint Consensus

- **Raft uses a 2-phase approach**
  - Intermediate phase uses joint consensus(need majority of both old and new configurations for elections, Commitment)
  - Configuration changes is just a log entry; applied immediately on receipt(committed or not)
  - Once joint consensus is committed, begin replicating log entry for final configuration



Joint Consensus, cont'd

- **Additional details:**
  - Any server from either configuration can serve as leader
  - If current leader is not in C_{new}, must step down once C_{new} is committed