Summary of Part 1

Use logs to record changes to your system.

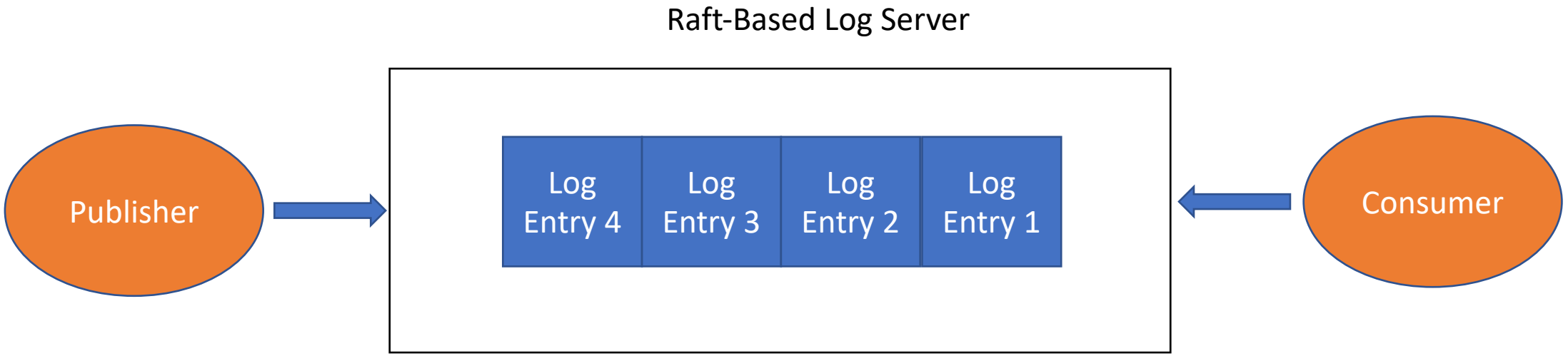
Centralized logs make it easy for the system have a universal, consistent, replayable record of how it evolved over time

Services that manage the central logs need to be correct, reliable, recoverable, and fault tolerant

The Raft algorithm is a popular way to provide these guarantees

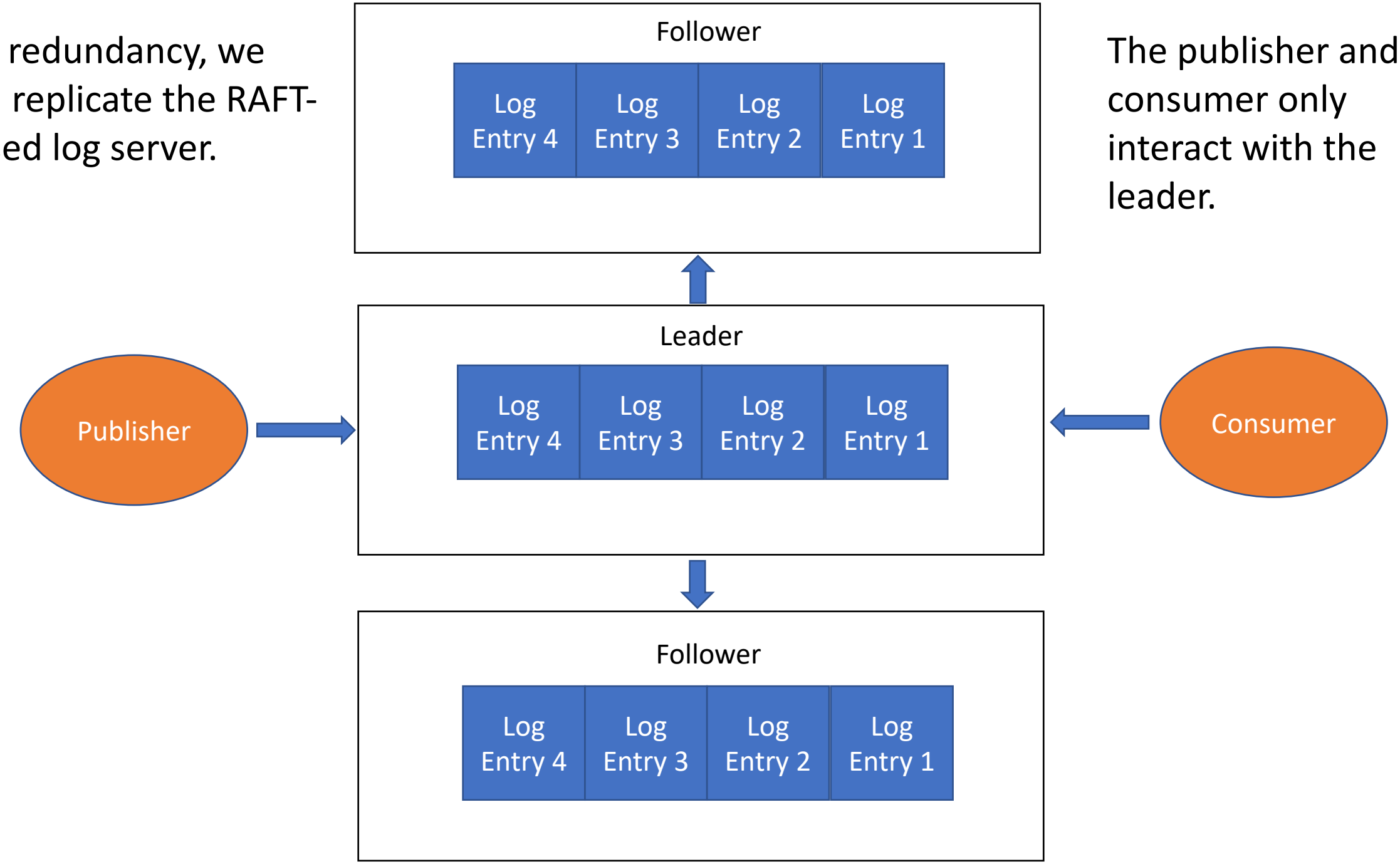
Part 2: Raft in Detail

A Simple Raft Scenario



- A publisher appends entries to the Raft-based log server
- Each log entry contains commands, data, etc
- The log server stores the entries in order that they are received
- A consumer reads entries from the server

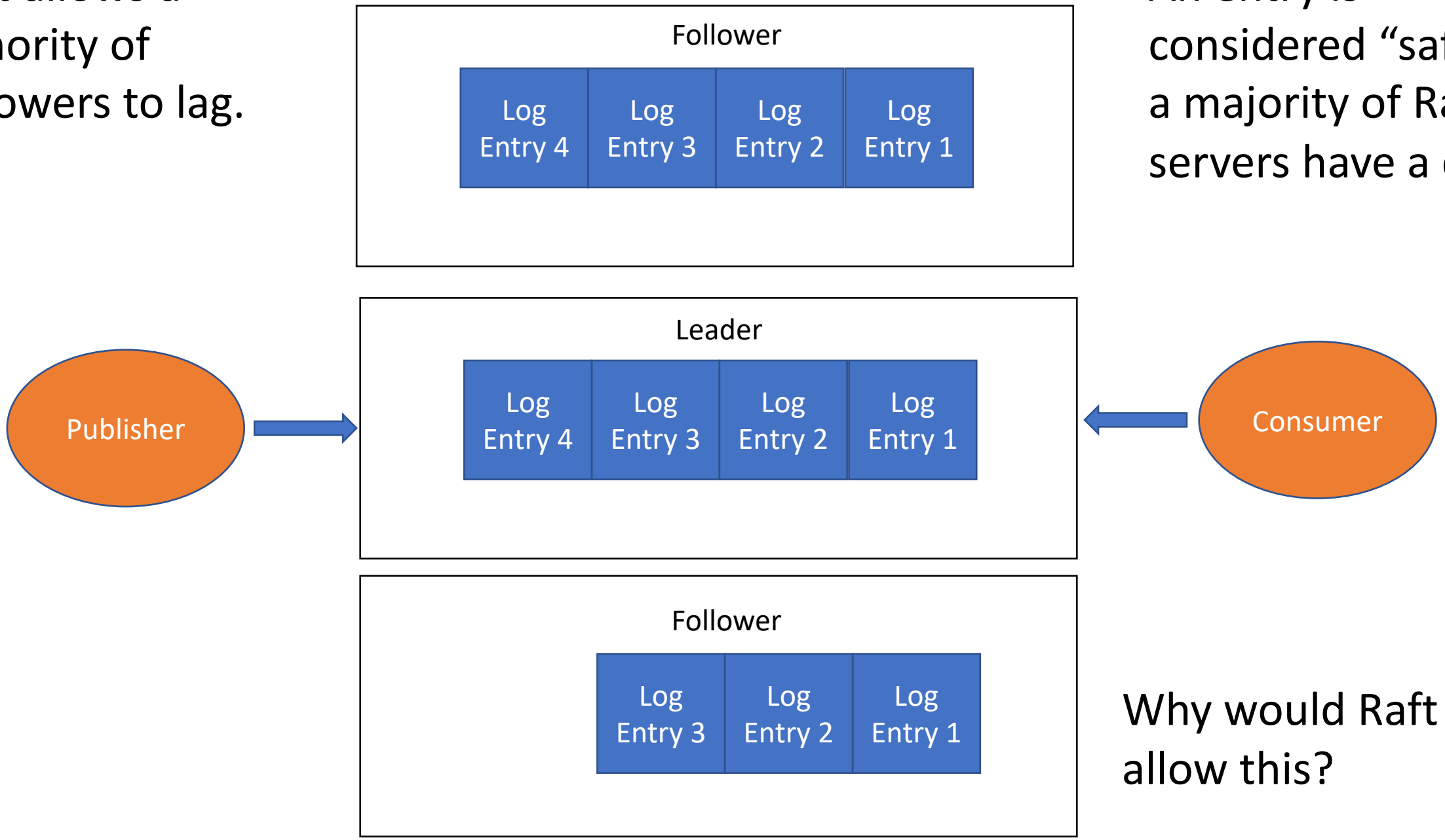
For redundancy, we can replicate the RAFT-based log server.



The publisher and consumer only interact with the leader.

Raft allows a minority of followers to lag.

An entry is considered “safe” if a majority of Raft servers have a copy



Why would Raft allow this?

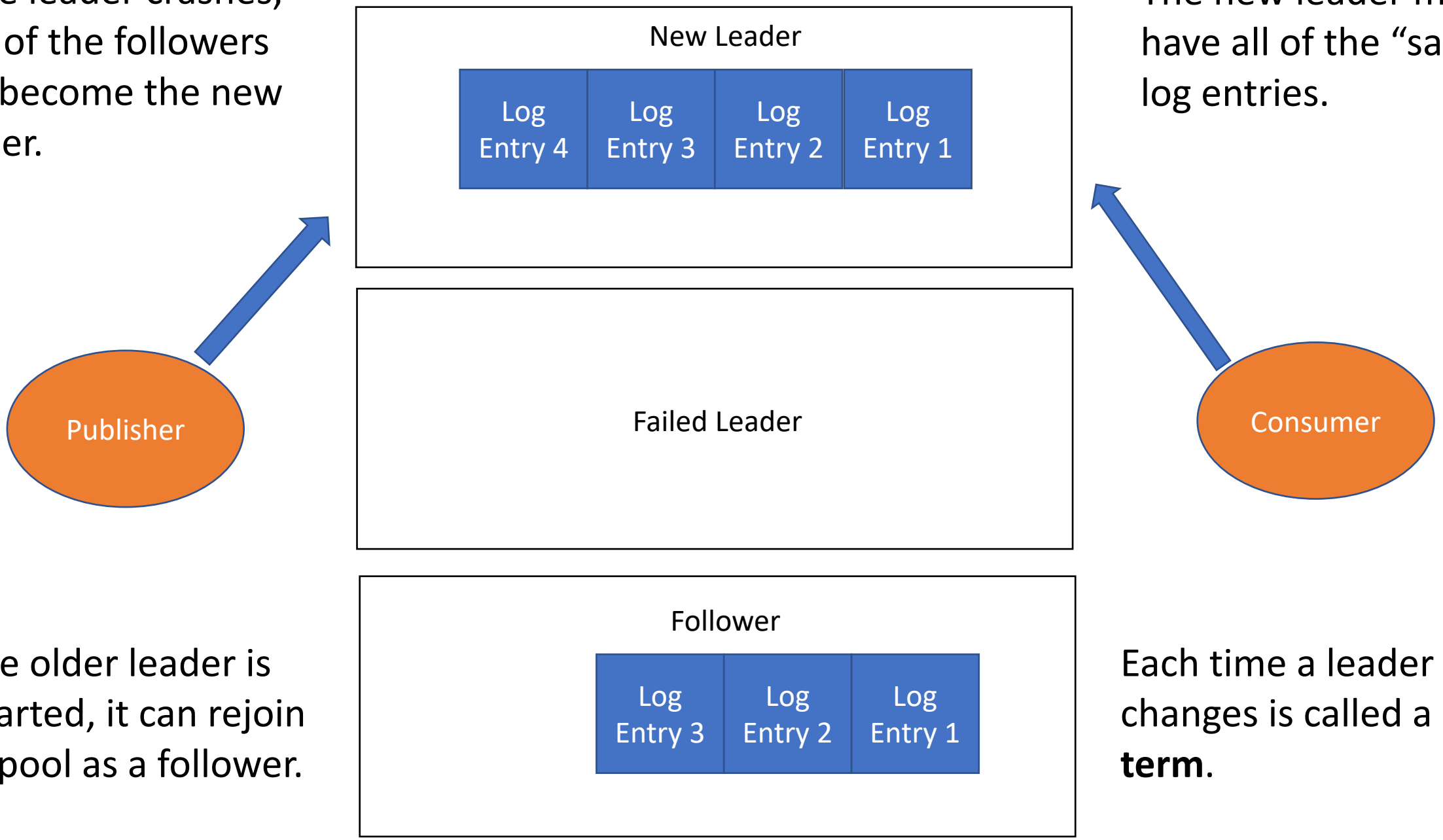
Why Allow Followers to Lag?

You can have more servers in the Raft cluster

Having more servers in the Raft cluster makes it more fault tolerant.

Publishers don’t have to wait for the log to be replicated on all followers, just most of them.

If the leader crashes, one of the followers can become the new leader.



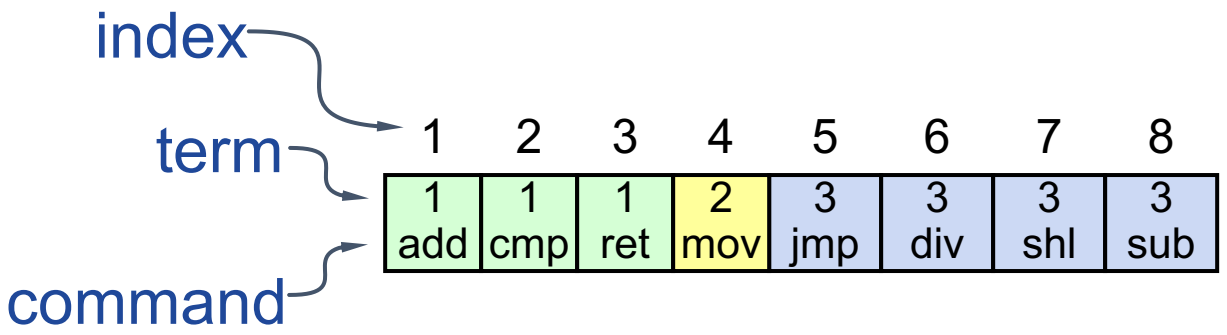
The new leader must have all of the “safe” log entries.

If the older leader is restarted, it can rejoin the pool as a follower.

Each time a leader changes is called a **term**.

This is called “consensus”

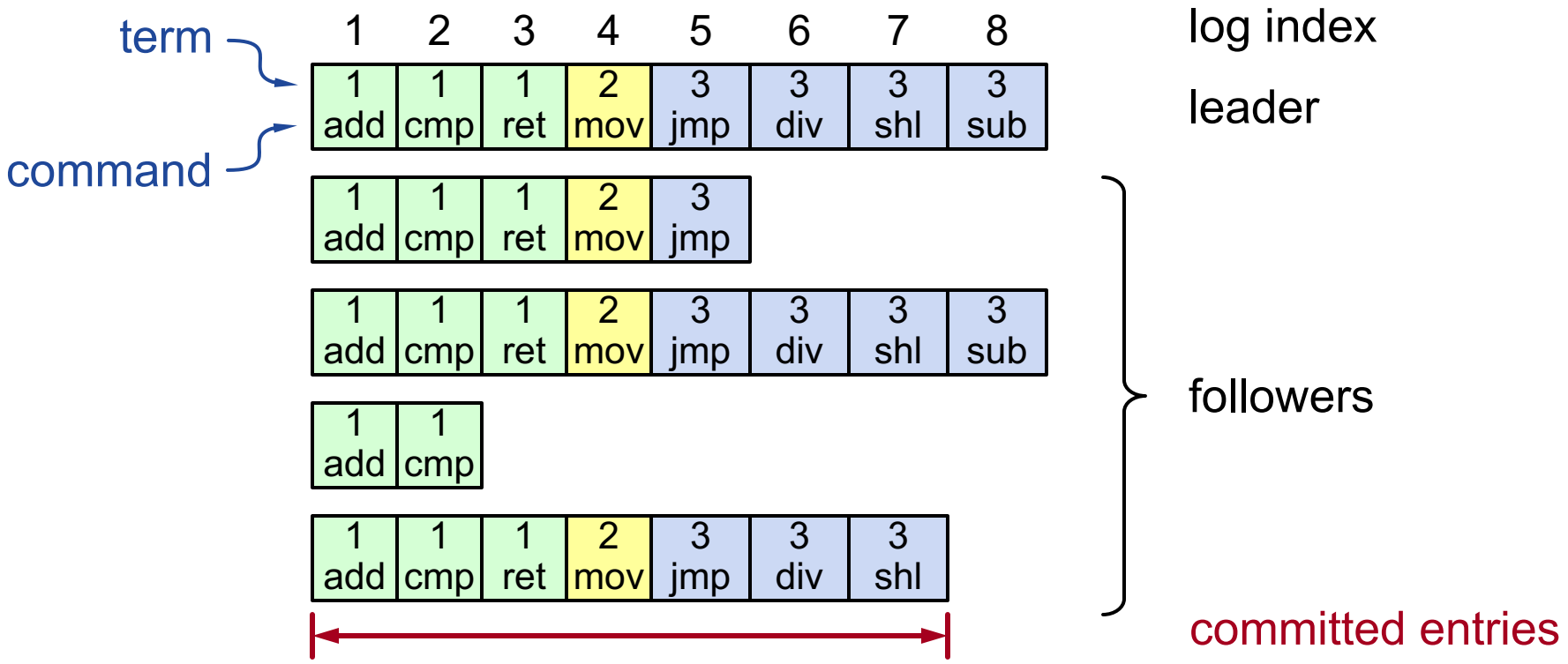
A RAFT Log Looks Like This



- Log entry = index, command, term
- Logs are stored on stable storage (disk).
- If the server crashes, re-read the log from disk

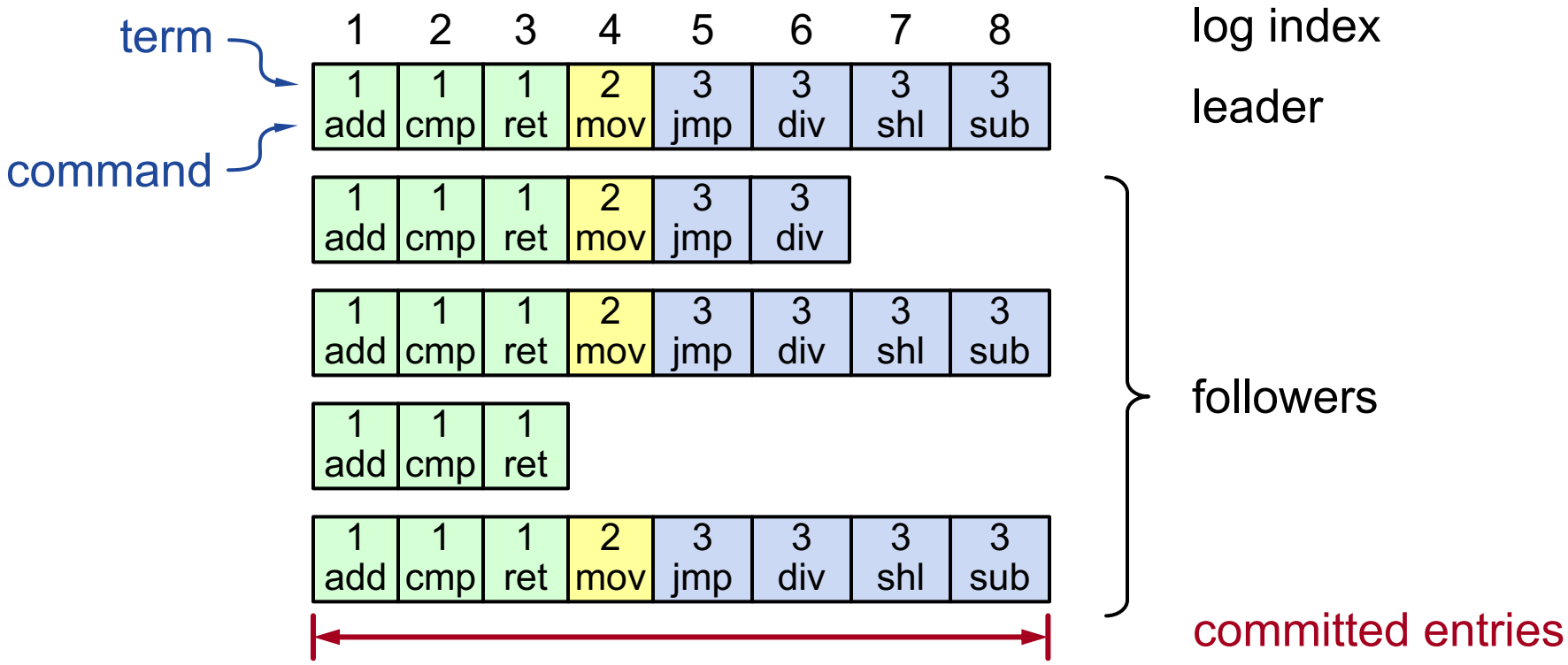
Let’s see what this looks like for five servers

Log Structure Snapshot



- **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

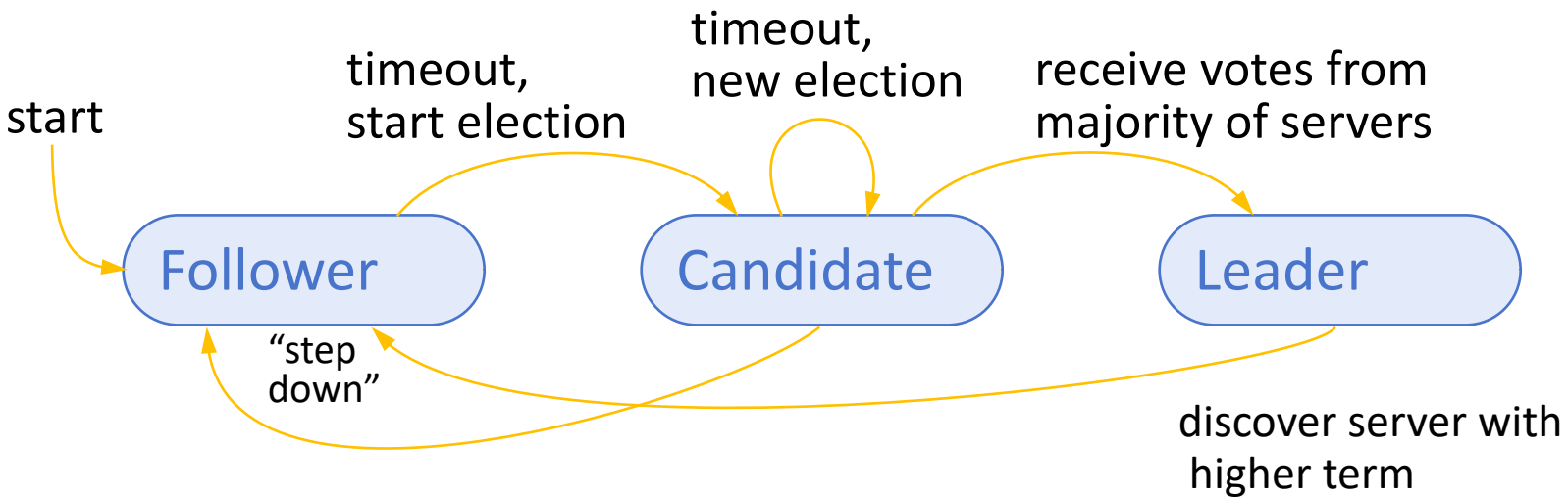
Log Structure Snapshot+1



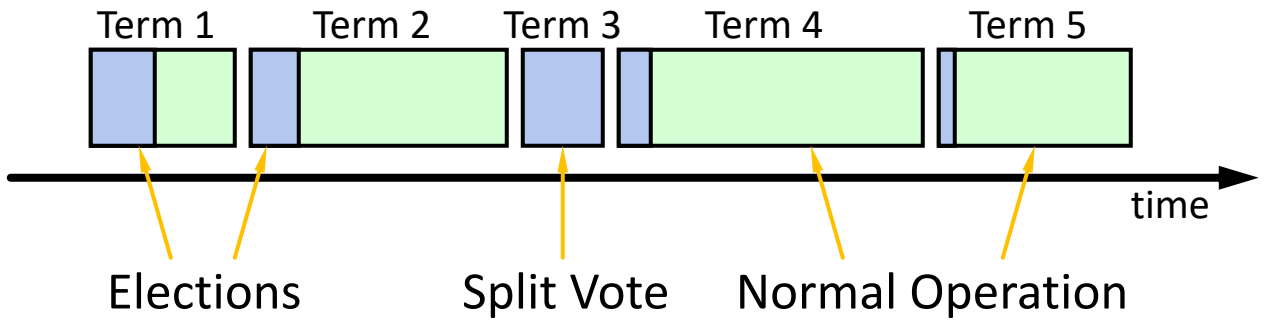
Leaders and Leadership Changes

- There is at most one leader at any time
- A system needs a new leader if the leader fails or becomes disconnected from a majority of followers: **heartbeat failures**
- New leaders are chosen by election from among the followers
  - Followers become candidates if they detect that a leader has failed
  - Only one candidate can win
- A **term** is the time period that a particular leader is in charge

Raft Server States:



Raft Time Evolution:



Key Raft Properties

- **Election Safety:** At most, only one leader can be elected
- **Leader Append-Only:** Leaders can only append entries to the logs. They never change committed (safe) entries.
- **Log Matching**: if two logs have the same index and same term, then the logs are identical in all entries up to that index
- **Leader Completeness:** If a log entry is committed in a given term, then the entry will appear in the logs of leaders of future terms
- **State Machine Safety:** if a server has committed an entry, no server will ever overwrite this entry

Desirable qualities, but how do we implement it?

Logs and Committed (Safe) Logs

- Servers can have log entries that are volatile
- Log entries are only **committed** after a majority of servers have accepted the log message.
- **Committed logs are guaranteed**
- External clients only get acknowledgements about committed log entries

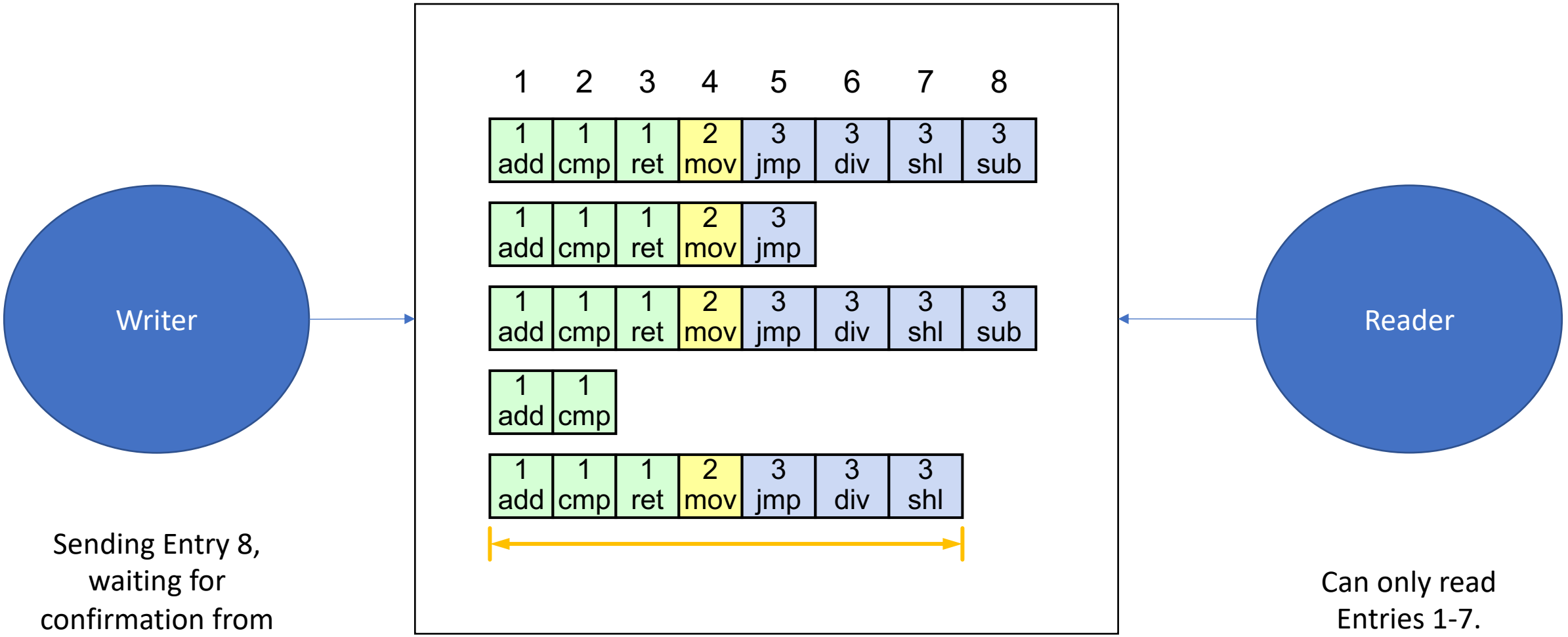
RAFT Has Only Two API Methods: AddEntries and RequestVote

AppendEntries: Leader sends this to each follower

RequestVote: a candidate sends this to the rest of the cluster

Both methods use term and log entry indices.

Based on the term and log index, the recipient either applies or rejects the request.

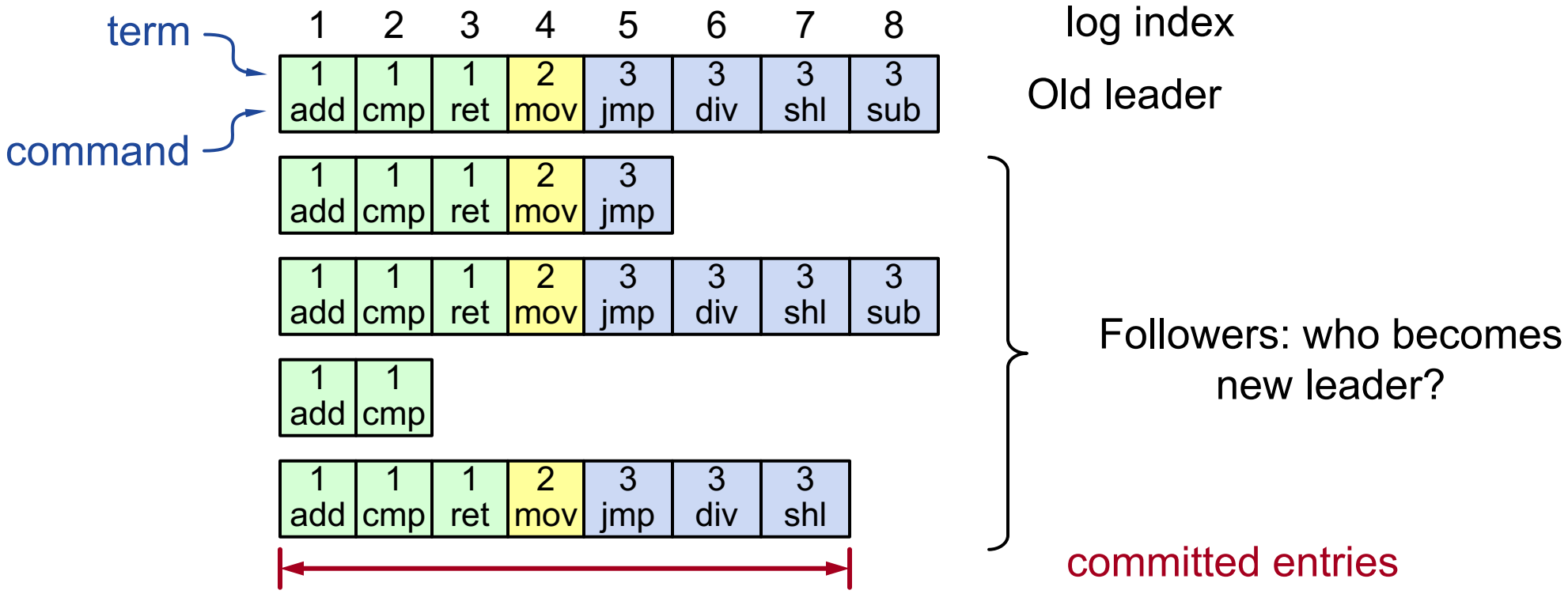


Sending Entry 8, waiting for confirmation from the leader.

Can only read Entries 1-7.

Only Entries 1-7 are committed when this snapshot was taken.

Run an Election

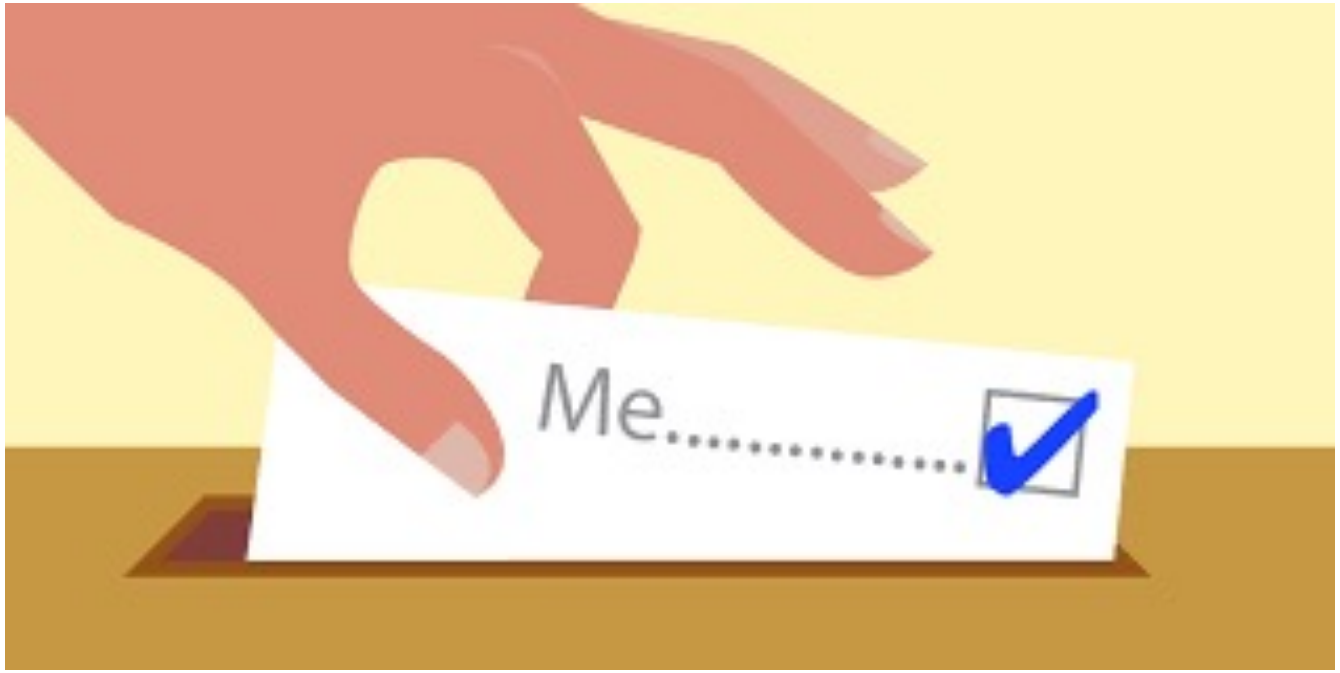


Raft Election Process (1/4)

- Leaders send heartbeat messages periodically
- If a **follower** doesn't receive a heartbeat during an "election timeout" period, it changes to a **candidate.**

Raft Election Process (2/4)

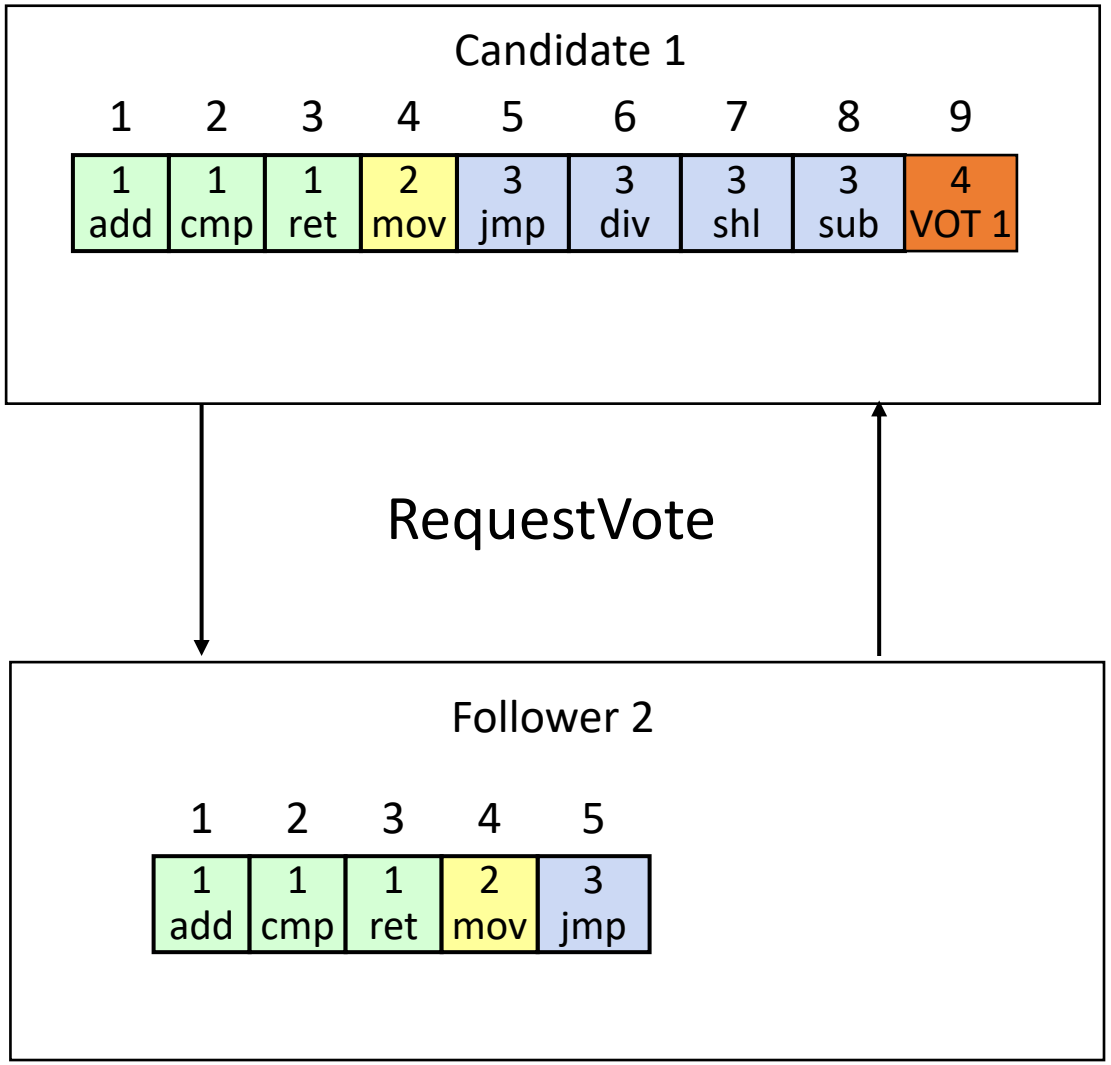
The candidate increments its term value and votes for itself



Raft Election Process (3/4)

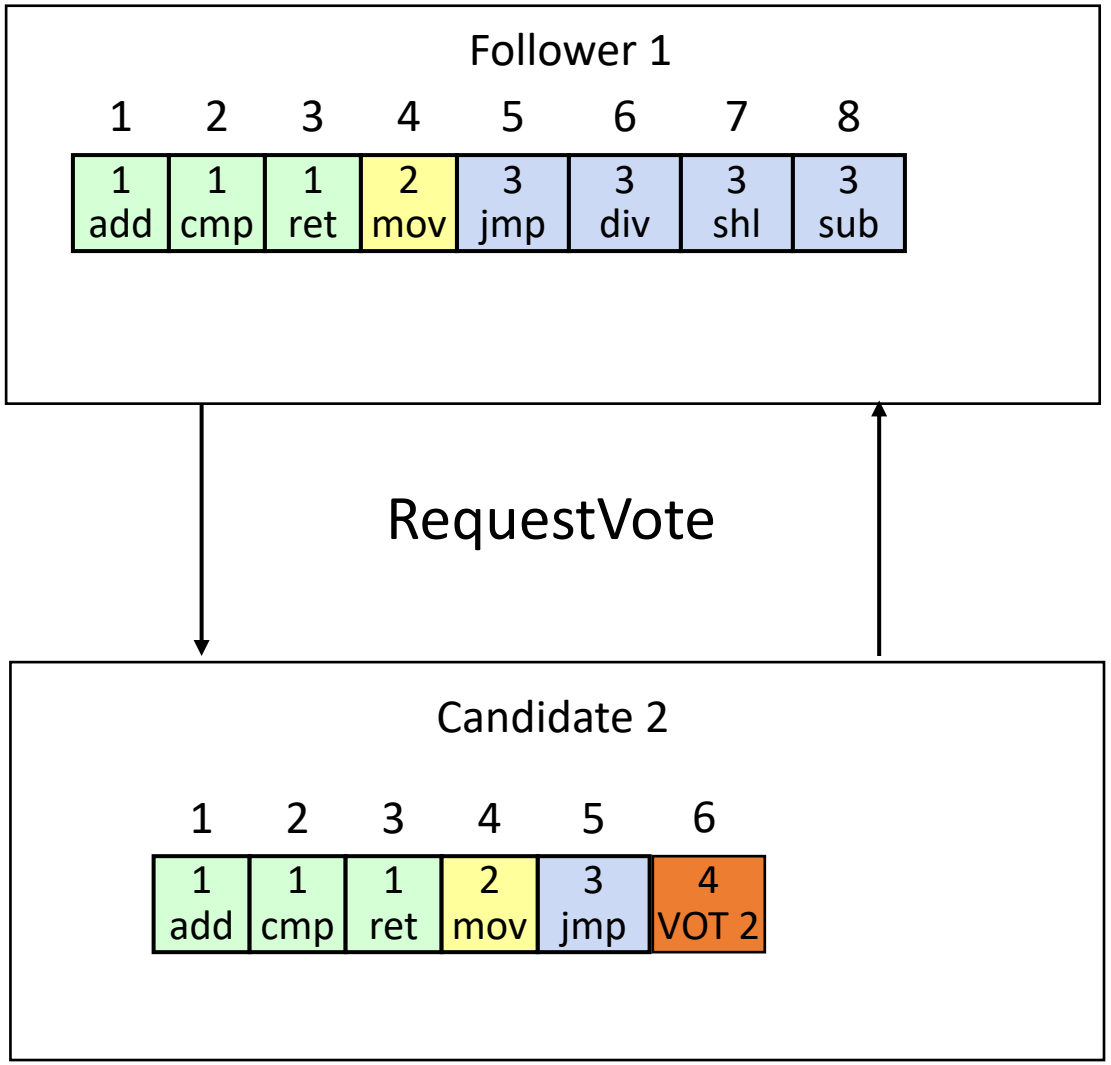
- Candidate sends its **term** and **index** to all the other servers in a RequestVote message
- The recipient server is either in “follower” state or “candidate” state
- They compare **term** and **index** values of last logs
- Highest wins
- A server only votes at most once per term

The RequestVote command can come from either candidates.



Candidate 1 will get Follower 2's vote.

The RequestVote command can come from either candidates.

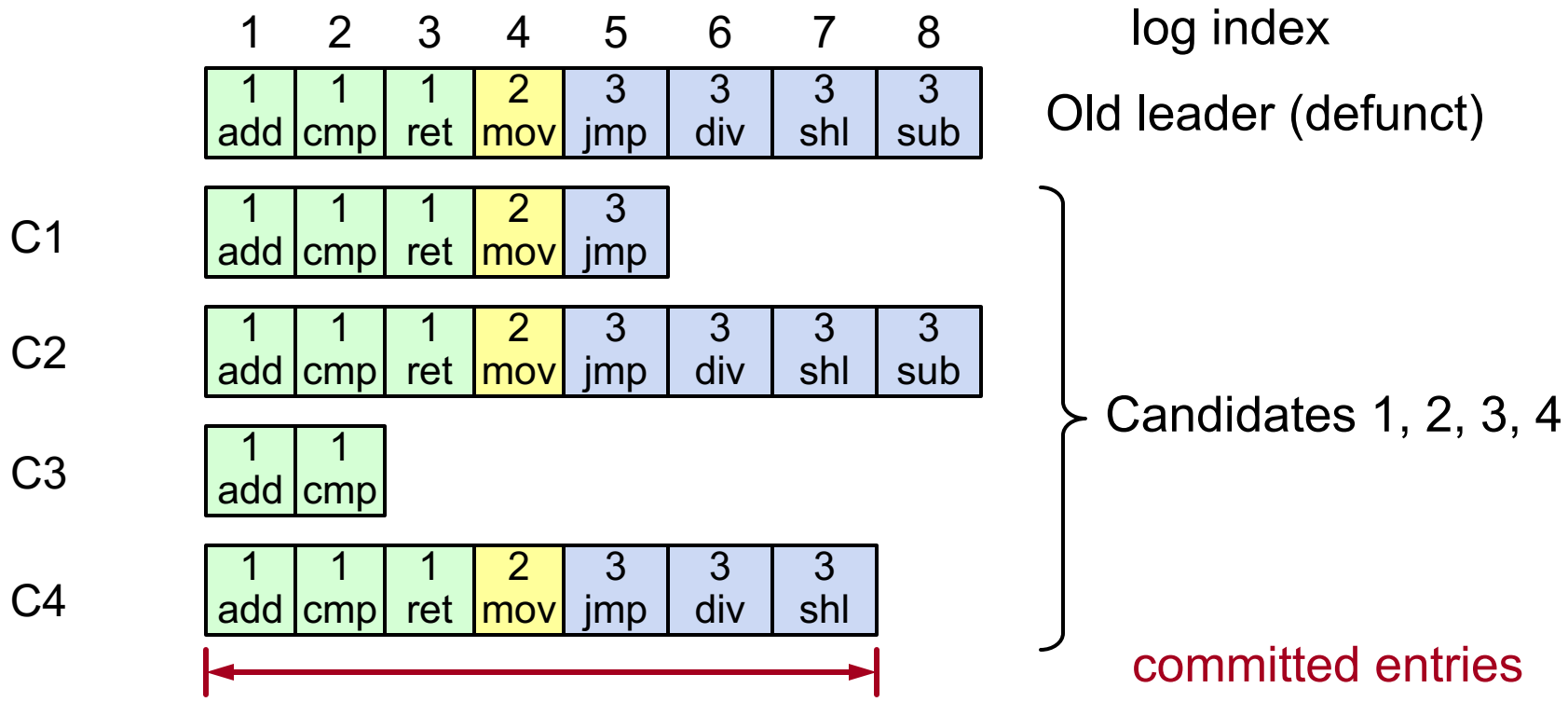


Follower 1 will not vote for Candidate 2.

Raft Election Process (4/4)

- When a candidate receives votes from a majority of servers, it wins the election and becomes the new leader
- It establishes this by sending an ”I Won!” log message to the clulster with its term.

Election



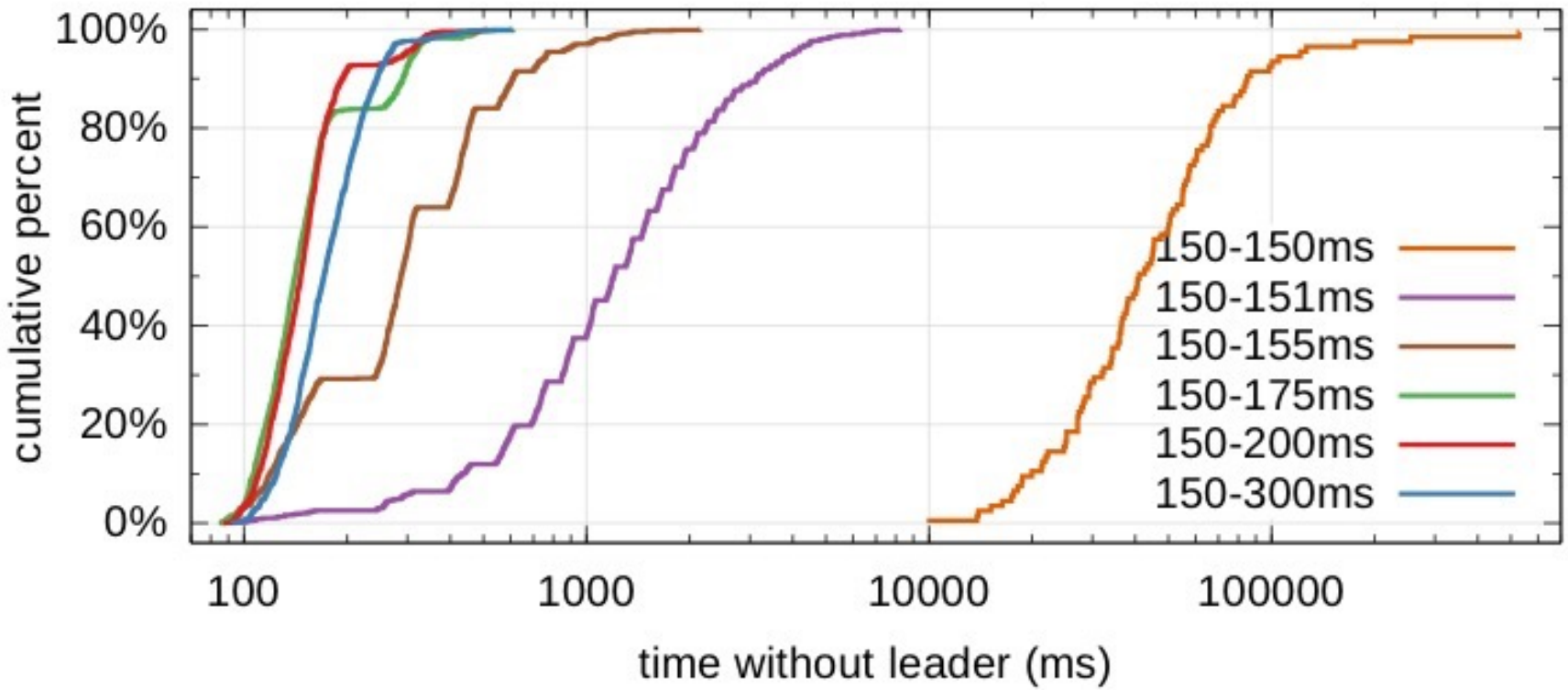
- **Candidates C2 and C4 can win the election with 3 votes**
  - Candidates C1 and C3 will vote for either
  - Candidate C4 can win if gets votes from C1 and C3 before contacting Candidate 2
- **Note a split election is possible: Follower 1 votes for Candidate 2, and Follower 3 votes for Candidate 4, for example**
- **Log entry 8 was not committed, but that's ok: the client (not shown) hasn't received an acknowledgement**

Split Elections

- If no candidate receives a majority of votes, the election fails and must be held again.
- Raft uses random election timeouts.
- If a leader hasn’t been elected within the election time out, a candidate increments its term and starts a new election.
- Each server has a different time out, chosen at random.
- This randomness is key for the system to find a new leader quickly

Randomized Timeouts

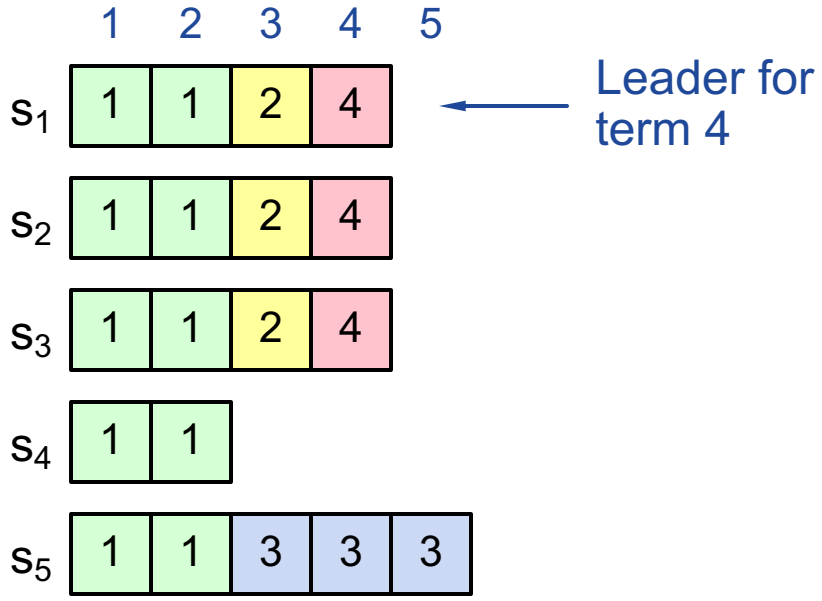
- How much randomization is needed to avoid split votes?



- Conservatively, use random range ~10x network latency

Log Commitment Rules

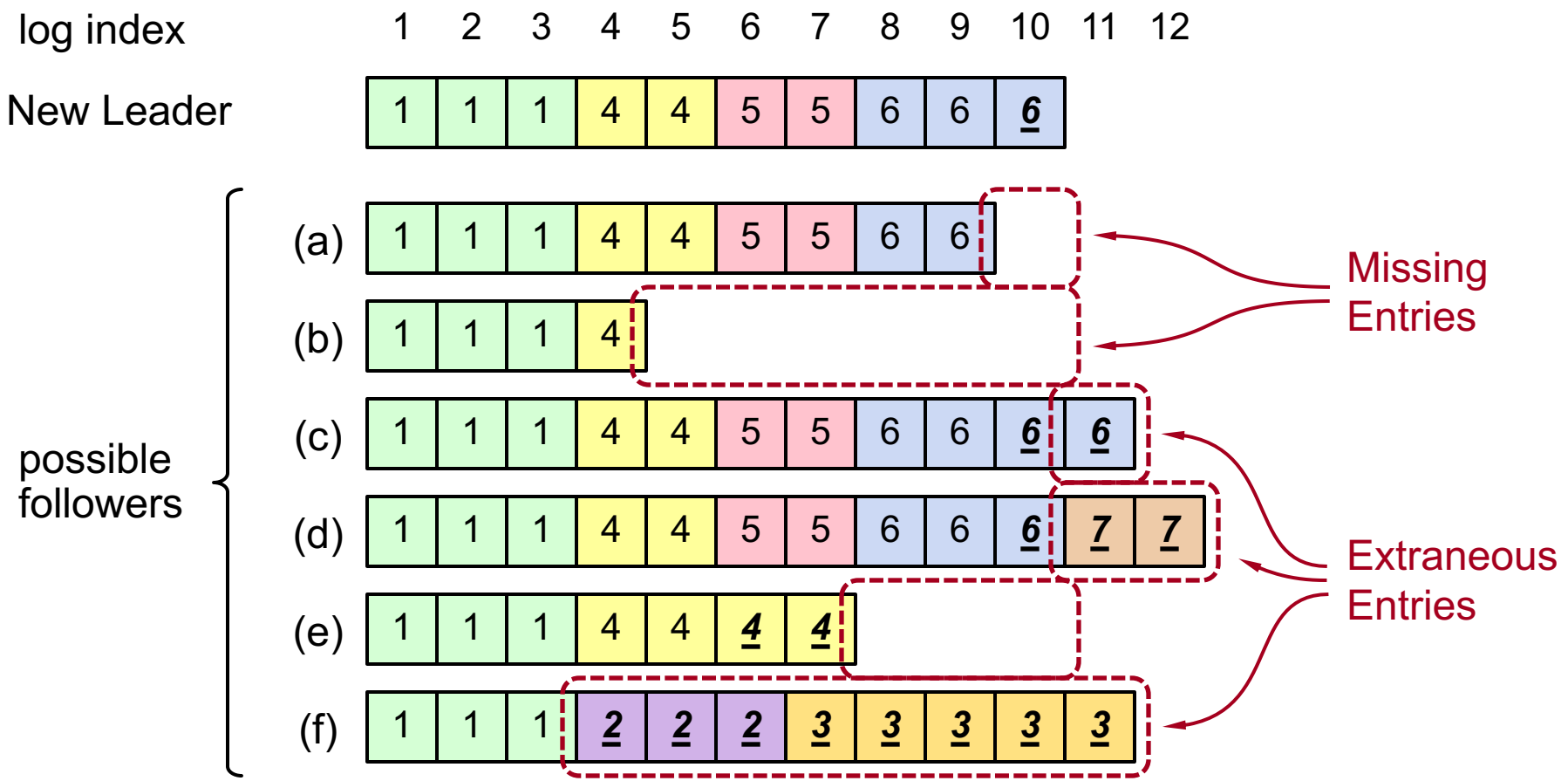
- **For a leader to decide an entry is committed:**
  - Must be stored on a majority of servers
  - **At least one new entry from leader's term must also be stored on majority of servers**
- **Once entry 4 committed:**
  - s_5 cannot be elected leader for term 5
  - Entries (2,3) and (4,4) both safe



Combination of election rules and commitment rules makes Raft safe

Log Inconsistencies

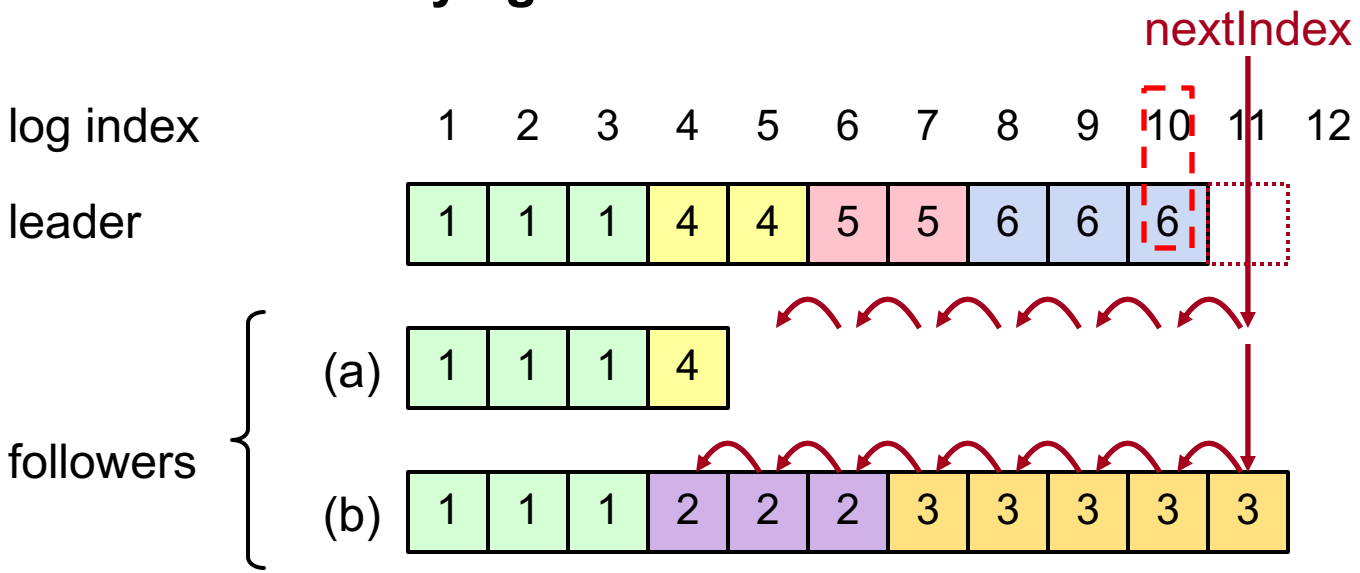
Leader changes can result in log inconsistencies:



Uncommitted entries: ***N***

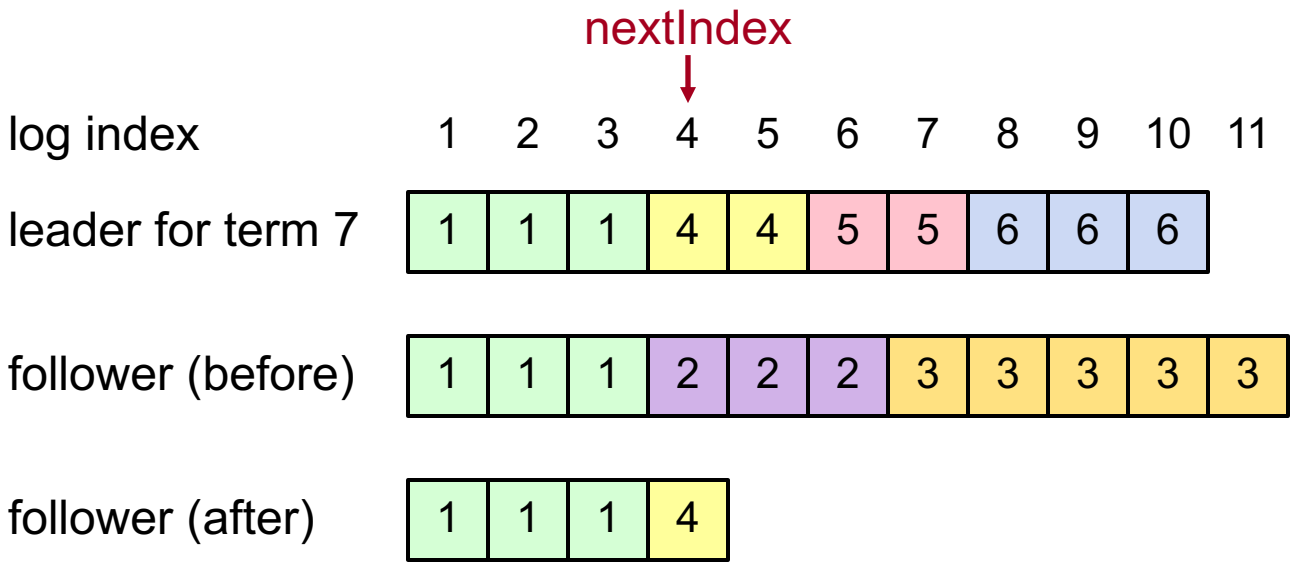
Repairing Follower Logs

- **New leader must make follower logs consistent with its own**
  - Fill in missing entries
  - Delete extraneous entries (these weren't committed)
- **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:**



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 (smaller), RPC is rejected, sender reverts to follower and updates its term
  - If receiver's term is older (smaller), it reverts to follower, updates its term, then processes RPC normally
- **An election updates the terms of majority of servers, by definition**
  - Deposed server cannot commit new log entries

Client Protocol

- **Send commands to leader**
  - If the client doesn't know the leader, contact any server in the cluster
  - If contacted server is not the leader, it will redirect to the leader
    - Even for READS
    - Client guaranteed to get latest committed log state
- **Leader does not send the response to the client until the 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

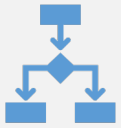
Some Previews of Upcoming Lectures

Changing a Raft Cluster

Imagine if you need to grow or shrink the Raft cluster

Or update the Raft server versions

Or move to new host servers



Can you do this without taking the system down?

Raft and Configuration Changes (Brief)

- Raft clusters can update themselves while continuously operating.
- They do this by requiring **joint consensus** between the old and new collections.
- This is an interesting approach for handling **Continuous Deployment** scenarios.

Some Limits of Raft (1/3)

All READ requests also go to the leader

The followers are just there as backups

When the Raft leader is stable, Raft provides strong consistency to its clients

But this can limit throughput

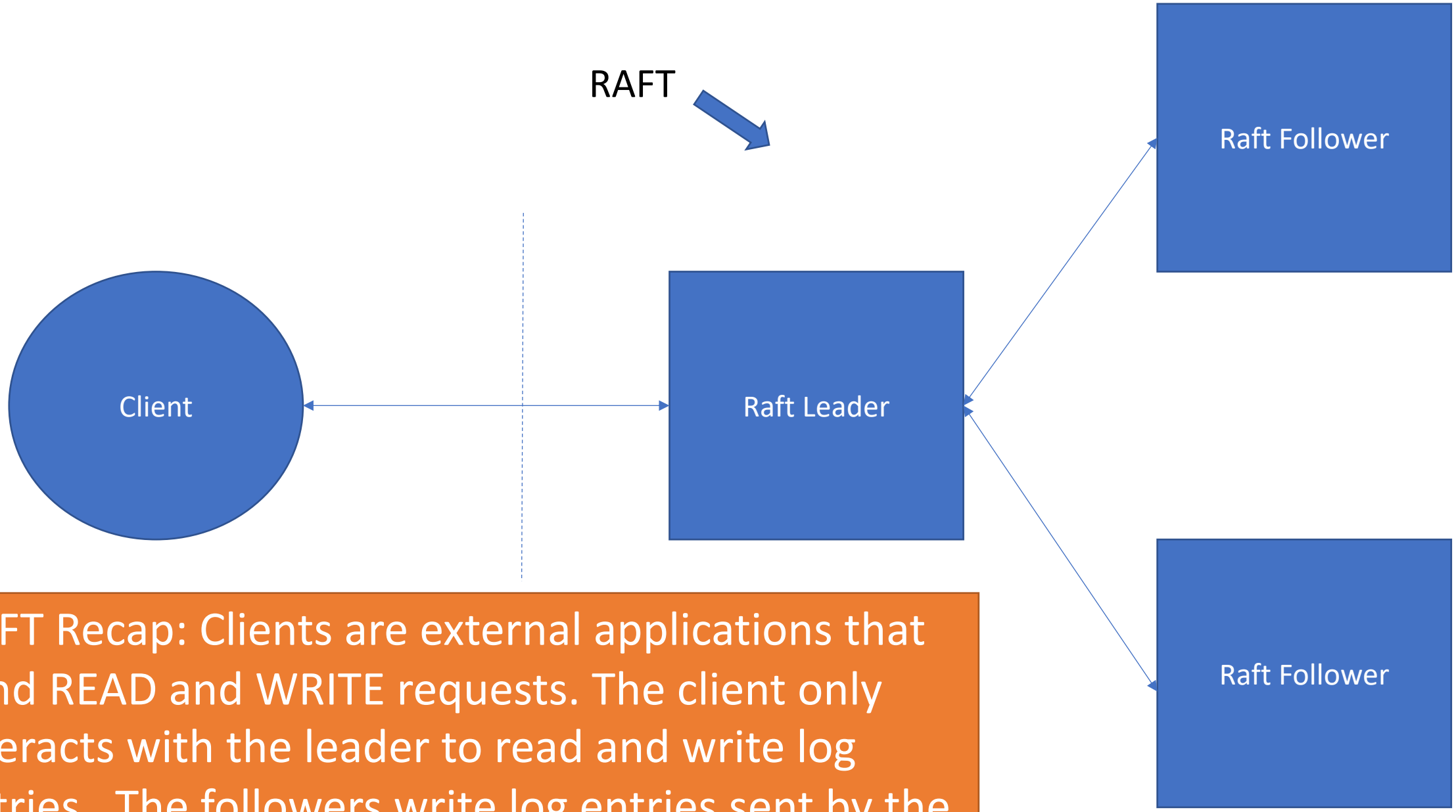
Zookeeper has weaker consistency but is more scalable for READs

Some Limits of Raft (2/3)

- Raft may not scale across geographically distant data centers or cloud regions
  - Consul supplements Raft with another protocol called SWIM
- The problem: network communication speed and network unreliability will make the system unwieldy
- Blockchain is a similar logging protocol that works at larger scales

Some Limits of Raft (3/3)

- Raft members communicate with each other using RPC calls.
- What if a malicious or faulty server is in the cluster?
  - It could tamper with logs
  - It could inappropriately share logs
  - It could disrupt the system by calling elections
- These are called Byzantine failures



RAFT Recap: Clients are external applications that send READ and WRITE requests. The client only interacts with the leader to read and write log entries. The followers write log entries sent by the leader. A follower can become a new leader.

What You Should Really Remember About Raft

Raft is a consensus algorithm for maintaining consistent state in distributed systems

It is the basis for Control Plane (information) services

Don’t try to invent something like this yourself. It is tricky.

It works best within a single data center with low latencies.