RAFT & Its TLA Spec

Consensus Algorithm for a Replicated Log

Jin Li
Microsoft

Outline

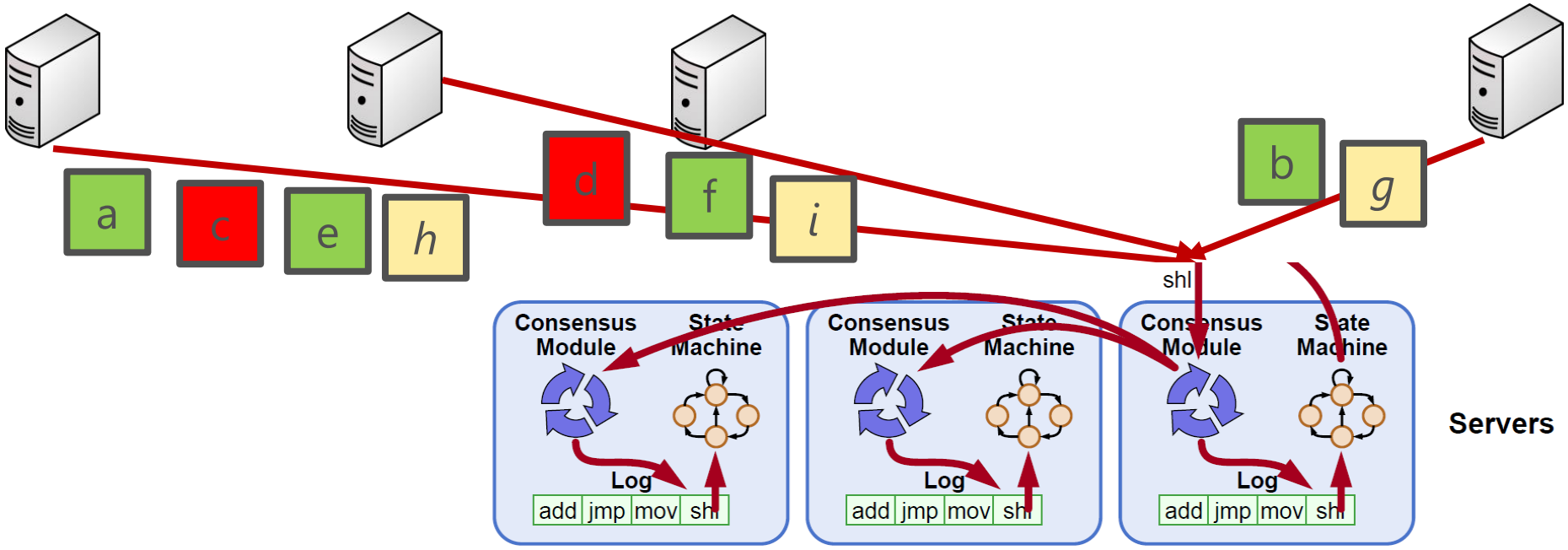
- Raft: Replicated Log for replicated state machines
  - 71 implementations on Raft Web, with Bloom, C, C#, C++, Clojure, Elixir, Erlang, F#, Go, Java, Javascript, Ocaml, PHP, Python, Ruby, Scala, Shell
  - Accessible and easily understandable
- Term & Leader election
- Safety and consistency:
- Configuration changes

Basic

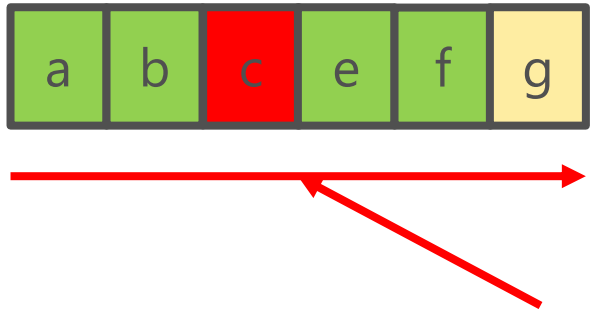
RAFT vs Paxos

	RAFT	Paxos
Concept	Replicated Log for Replicated State Machine	Consensus Algorithm
Leader Election	Key (First elected leader, then normal operation)	Optional (A form of operation)
Log maintenance	Leader's Log is replicated to all Followers	Node may log consensus operation, but it may have holes
TLA+ Spec	Close to developer's implementation & usage	More abstract
State Space	Large	Small

Raft: Functionality, Safety & Availability



Raft Cluster

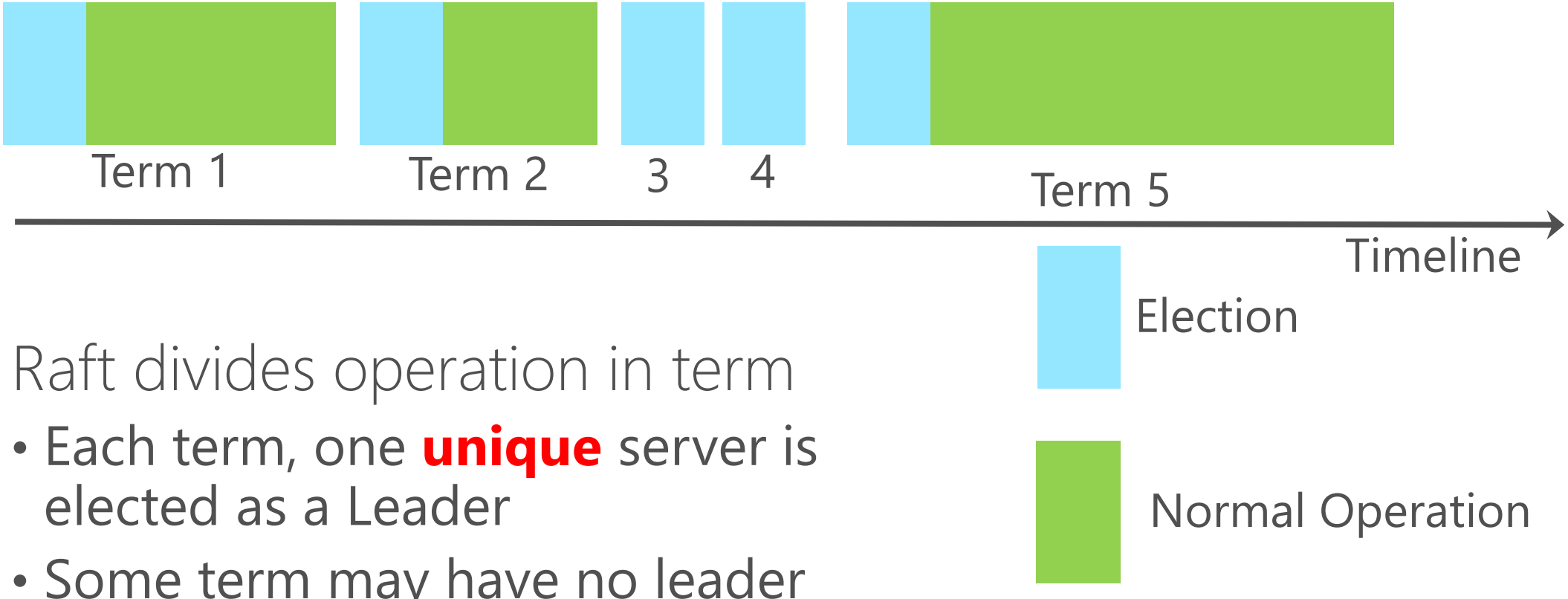


Committed Command

Uncommitted Command

Unacked Command

Raft: Operation Timeline



- Raft divides operation in term
  - Each term, one **unique** server is elected as a Leader
  - Some term may have no leader
  - Once a Leader is elected, it will enter normal operation mode

Raft: Operation

- Initialization

S_1: 1	Follower
S_2: 1	Follower
S_3: 1	Follower

Raft server can be in one of three state: Follower, Leader, Candidate

Raft: Operation

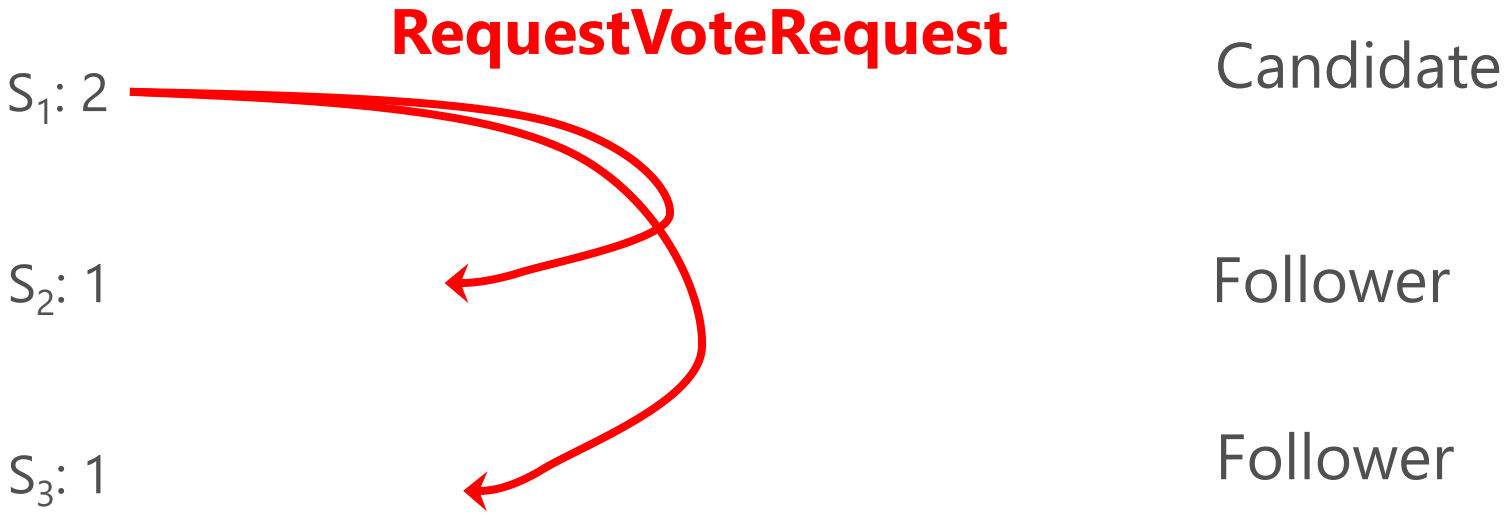
- S_1 timeout, start an election

S_1: 2	Candidate
S_2: 1	Follower
S_3: 1	Follower

- Timeout period of a server is chosen randomly between [T,2T], with T >> broadcast time of network. , eventually, one server will win election

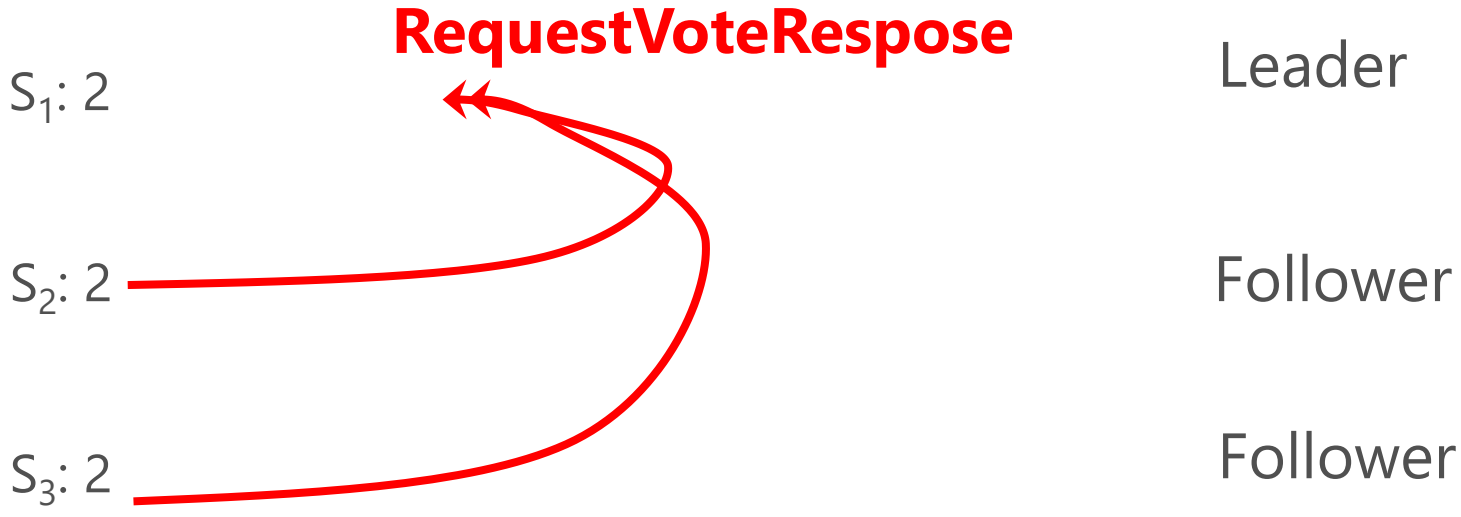
Raft: Operation

- S_1 timeout
  - Start an election



Raft: Operation

- S_1 become Leader
  - S_2 and S_3 grants the request of S_1

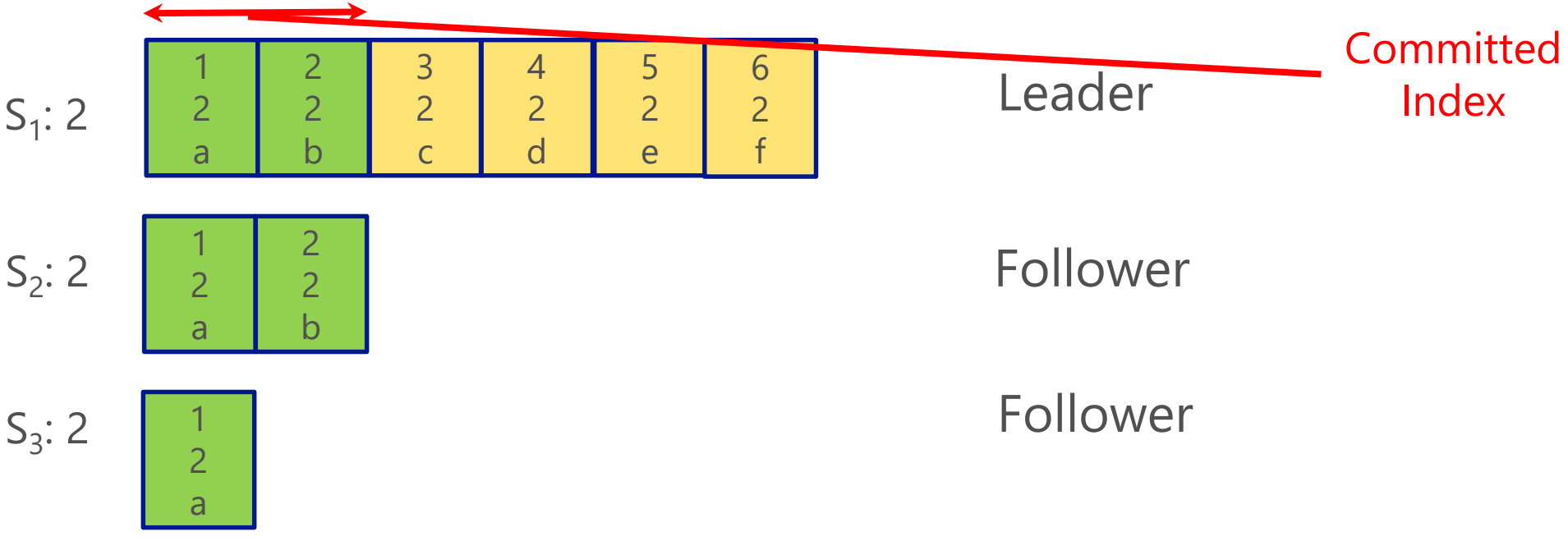


Raft: Operation

- Leader S_1 starts to receive client request
  - Leader write the received request in its log
  - If S_2 and S_3 gets client request, they will refer to Leader S_1
  - Asynchronously replicate the log to S_2 and S_3
  - Committed index grow once the log is committed and persisted to a majority of servers

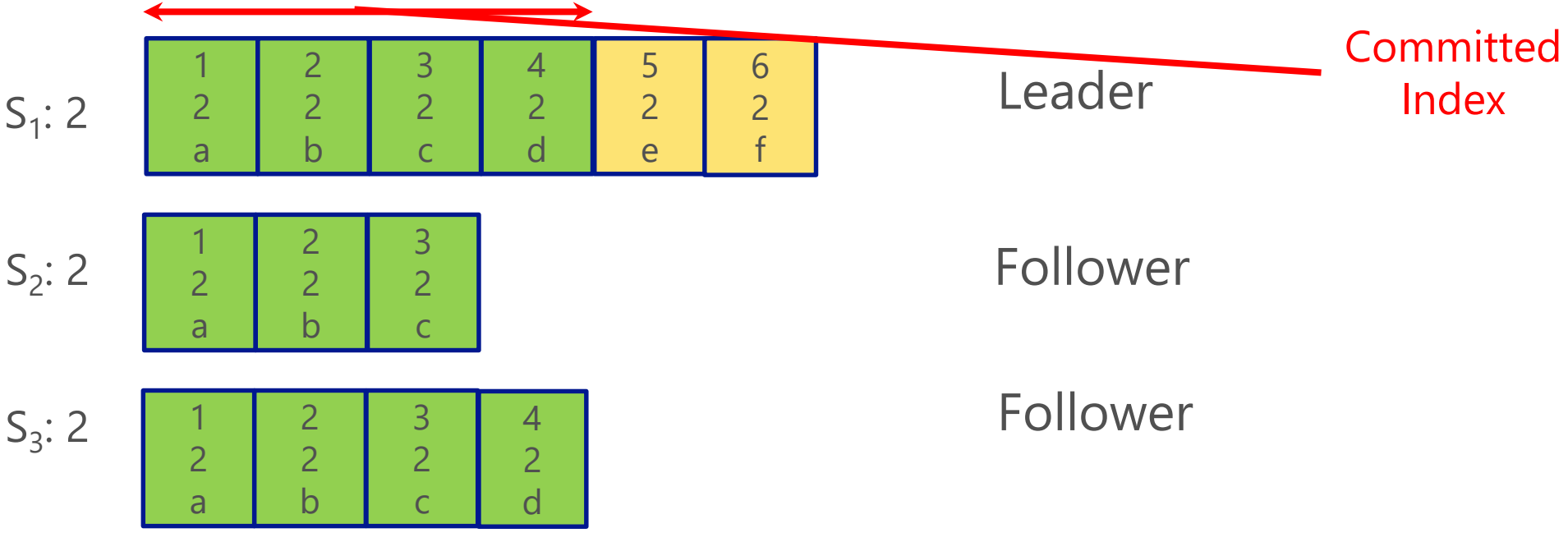
Raft: Operation

- Leader S_1 starts to receive client request
  - Leader write the received request in its log
  - If S_2 and S_3 gets client request, they will refer to Leader S_1
  - Asynchronously replicate the log to S_2 and S_3
  - Committed index grow once the log is committed and persisted to a majority of servers



Raft: Operation

- Leader S_1 starts to receive client request
  - Leader write the received request in its log
  - If S_2 and S_3 gets client request, they will refer to Leader S_1
  - Asynchronously replicate the log to S_2 and S_3
  - Committed index grow once the log is committed and persisted to a majority of servers



Raft Safety: Election, Log Replication, etc..

Safety

- Under complex failure scenario in real world
  - Server can fail and restart
  - Network packet can be delayed, replicated and reordered
- How to ensure safety of the replicated log?
  - Election safety: at most **one server per term**
  - Leader completeness: if a log entry is committed, the entry will present in the log of leaders for all higher-numbered term
  - Log matching: if two logs contains an entry with the same index and term, then the logs are identical up through that entry
  - Leader append only: Leader only appends new entries, never overwrites or delete its entries
  - Sate machine safety: if a server has applied a log entry at a given index, all other servers will apply the same entries for the same index

Election & Unique Leader

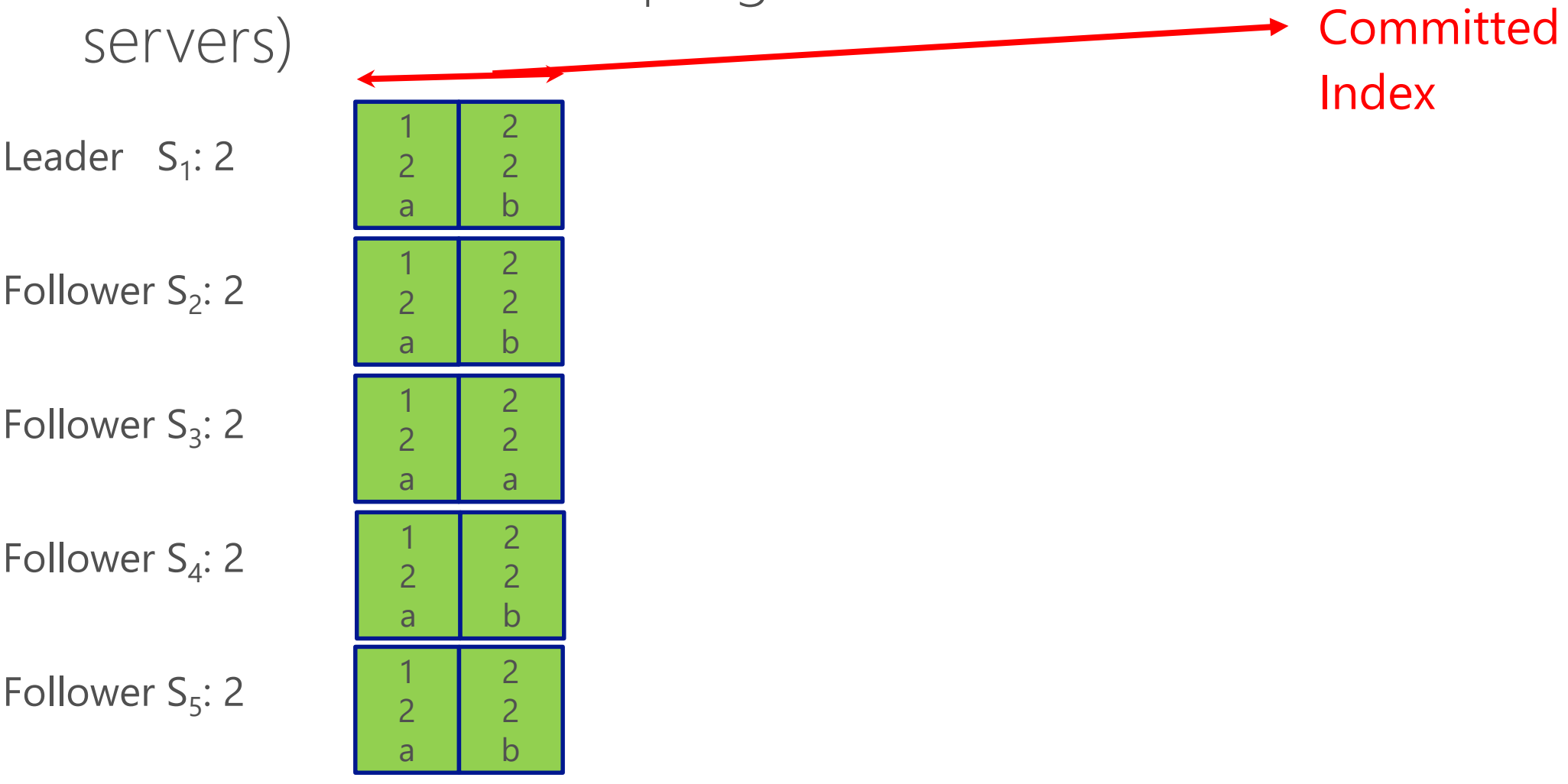
- Server
  - currentTerm & voteFor: **persisted** (save & flush to disk before complete the operation, survive crash/reboot)
- Election:
  - Increases currentTerm, changes to Candidate state.
  - Votes for itself, and requests vote from all other servers
  - Server only grants vote if the received term is larger than or equal to its own term
  - Server only grants vote for one Leader (including itself) per term
  - Safety: only one server per term can accumulate vote from a majority
- Every RPC contains term of sender's term
  - If sender's term is older, RPC is rejected, the reply of receiver will update sender's term and change it to Follower
  - If receiver's term is older, it updates its term, and change itself to Follower

Heartbeats and Timeouts

- Leaders must send heartbeats
  - If there is client request and/or logs to be replicated, the heartbeats is just the normal **AppendEntries** RPC
  - Otherwise, send empty **AppendEntries** RPC
- Followers expect to receive RPCs from Leaders or Candidates
- If timeout
  - Follower starts new election (became candidate), each with different timeout
  - As long as election timeout is chosen randomly between [T,2T], with T >> broadcast time, eventually, one server will win election
- Servers start up as Follows

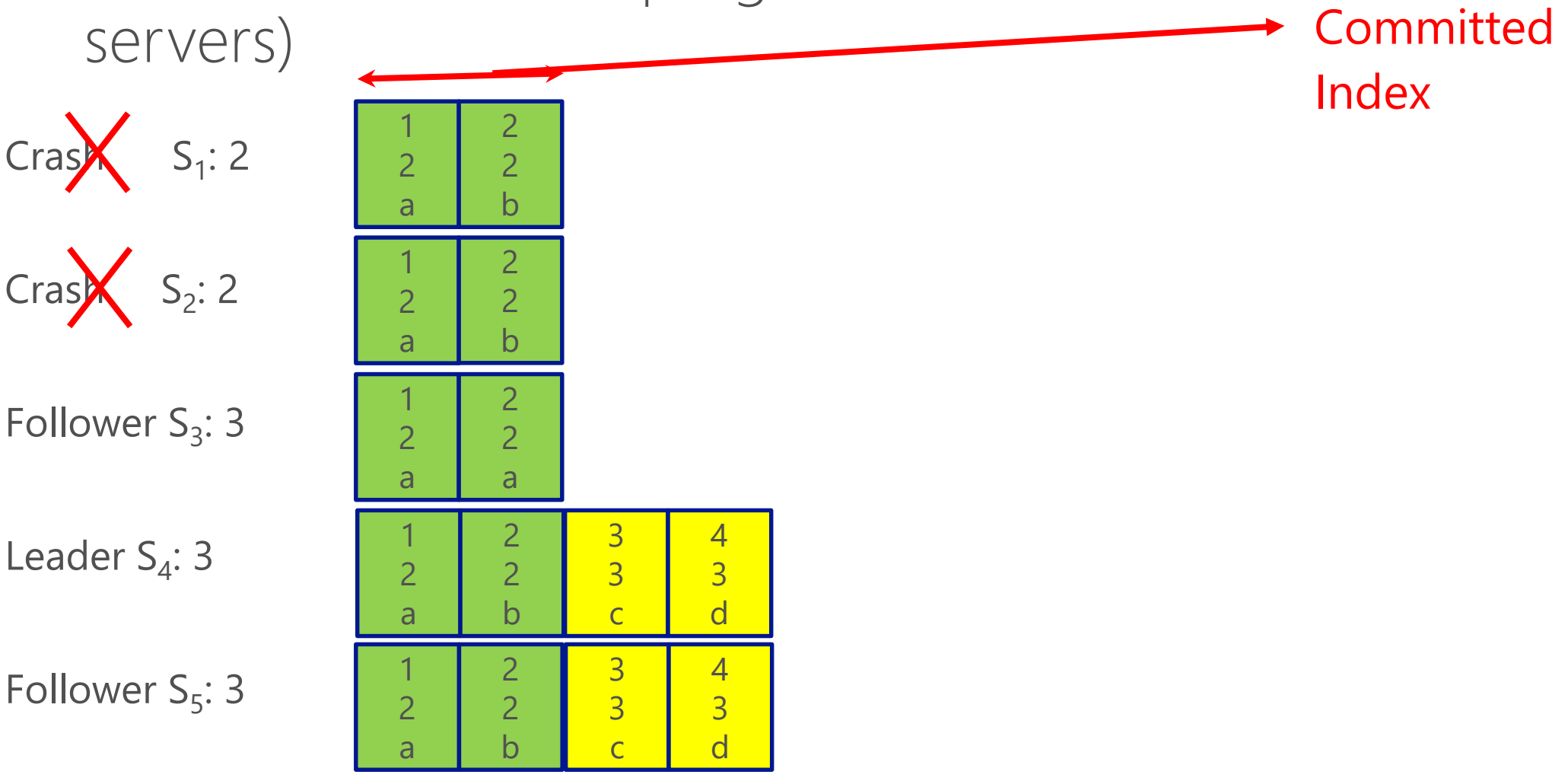
Safety: A Complicated Case

- Let's use a complicated case to examine Raft (lots of server crash/restart, progress made with bare minimum servers)



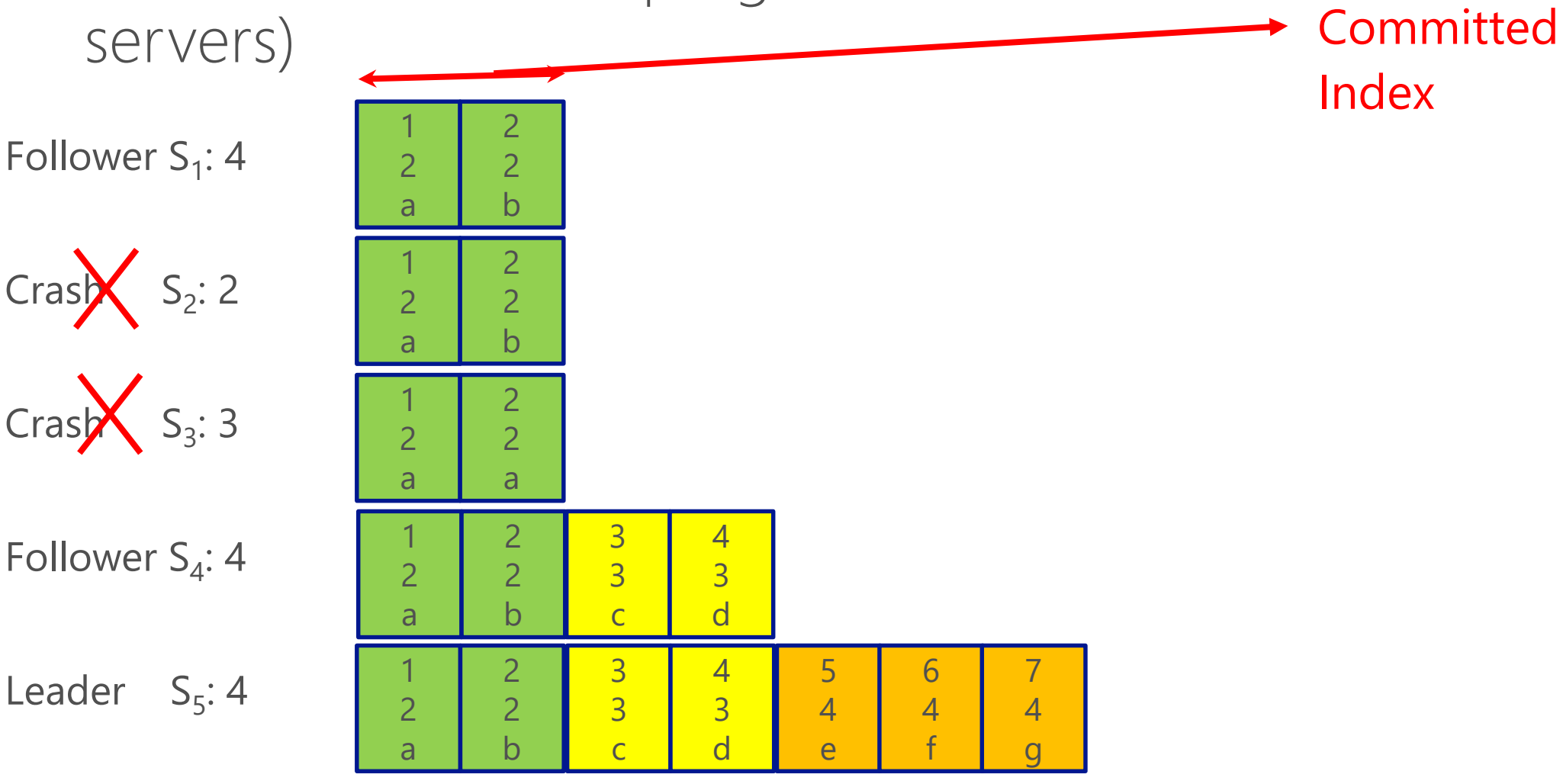
Safety: A Complicated Case

- Let's use a complicated case to examine Raft (lots of server crash/restart, progress made with bare minimum servers)



Safety: A Complicated Case

- Let's use a complicated case to examine Raft (lots of server crash/restart, progress made with bare minimum servers)



Safety: A Complicated Case

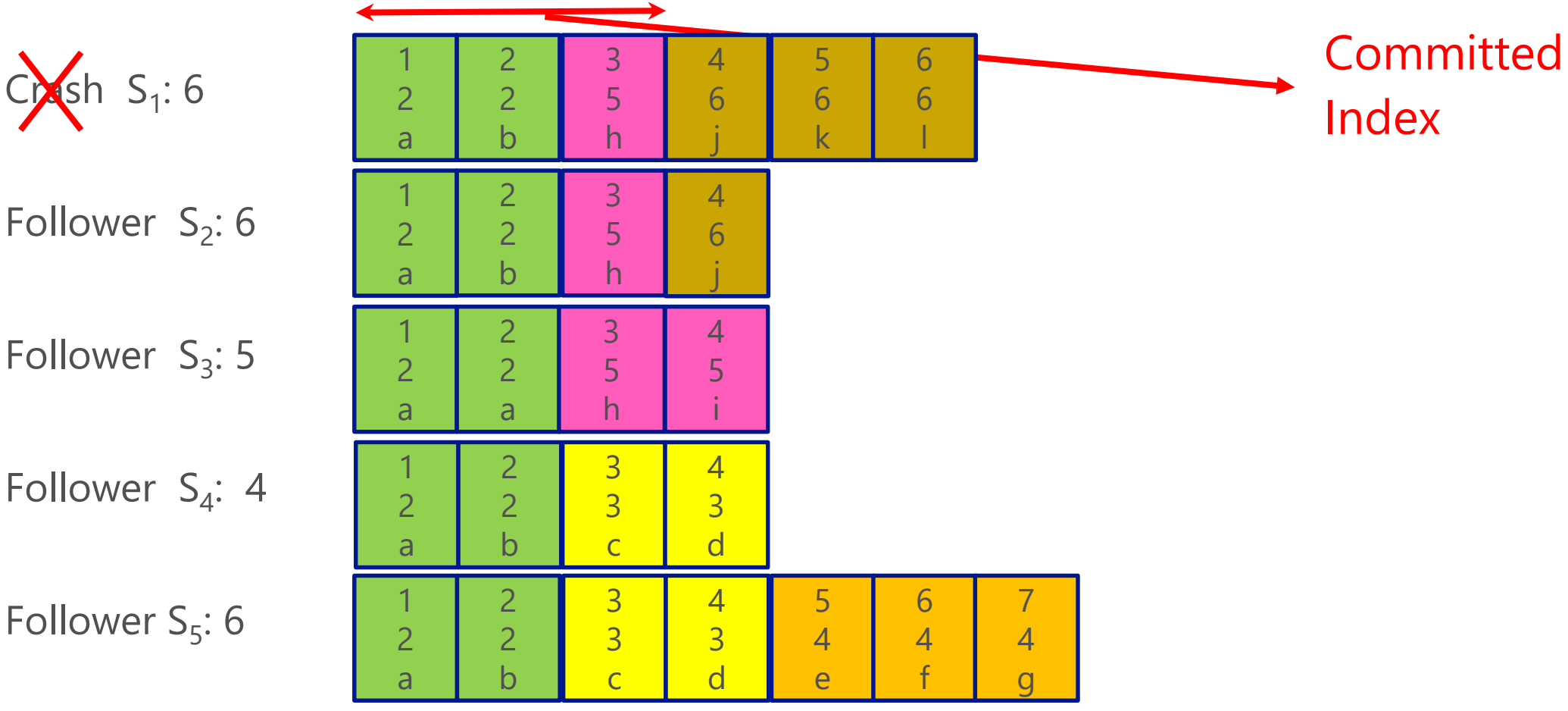
- Let's use a complicated case to examine Raft (lots of server crash/restart, progress made with bare minimum servers)

Safety: A Complicated Case

- Let's use a complicated case to examine Raft (lots of server crash/restart, progress made with bare minimum servers)

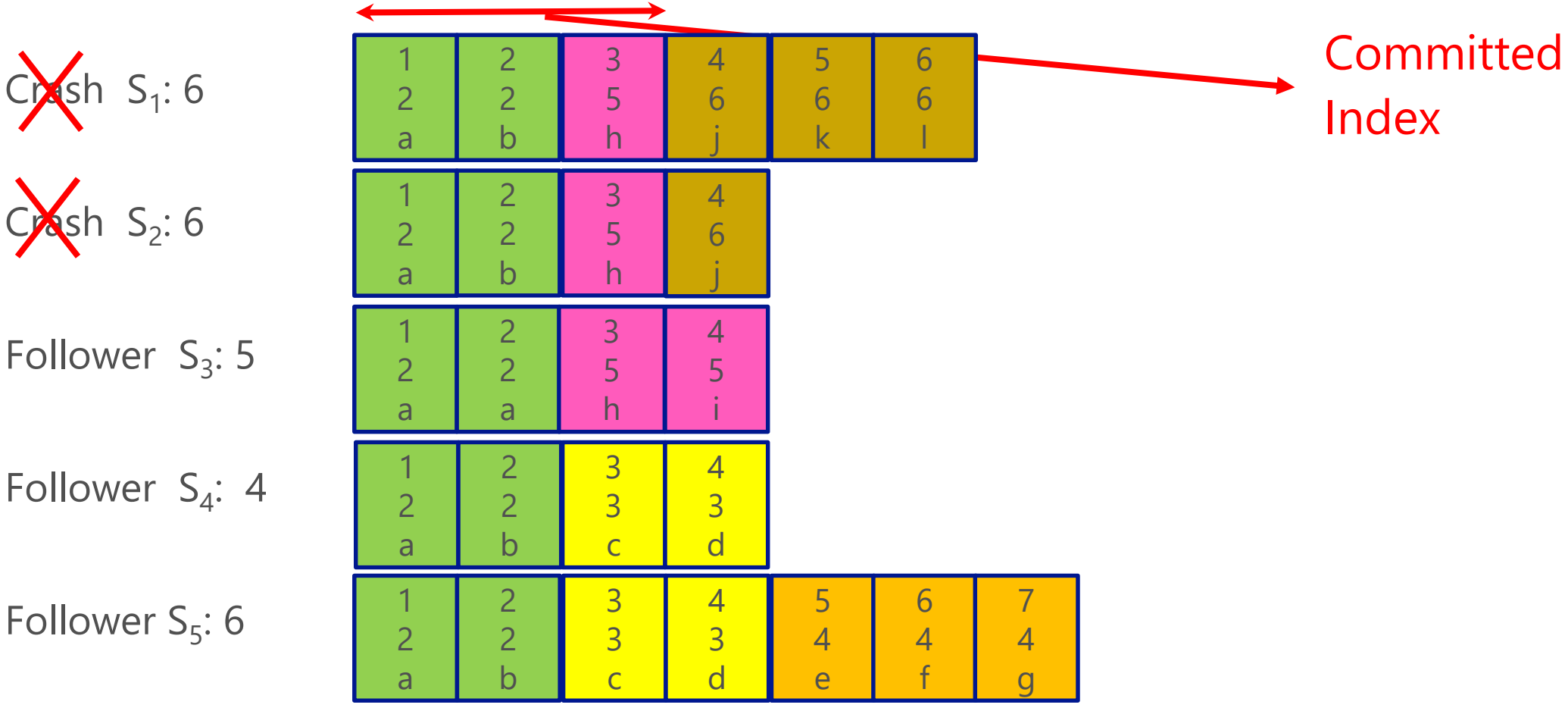
Who can be Leader at Term 7?

- S_2 may be new leader
- S_3, S_4 can't (term is 4 & 5, can't convince S_2 and S_5)
- S_5 can't (last log term is 4, S_2 and S_3 has log with higher term)



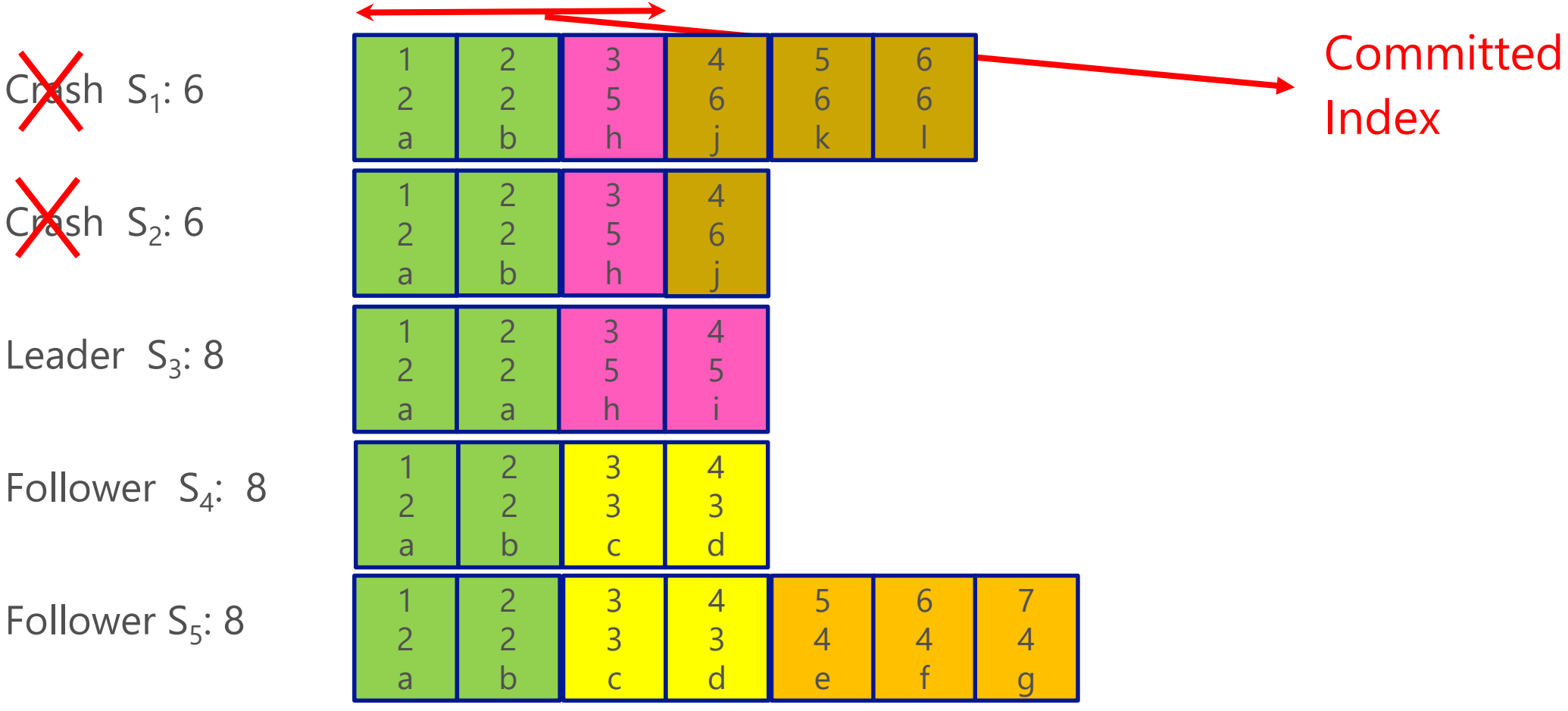
Who can be Leader at Term 7?

- S_3, S_4 can't (term is 4 & 5, can't convince S_5)
  - But S_3 and S_4 will learn new term 7
- S_5 can't (last log term is 4, S_2 and S_3 has log with higher term)



Who can be Leader at Term 8?

- S_3 can become Leader at term 8

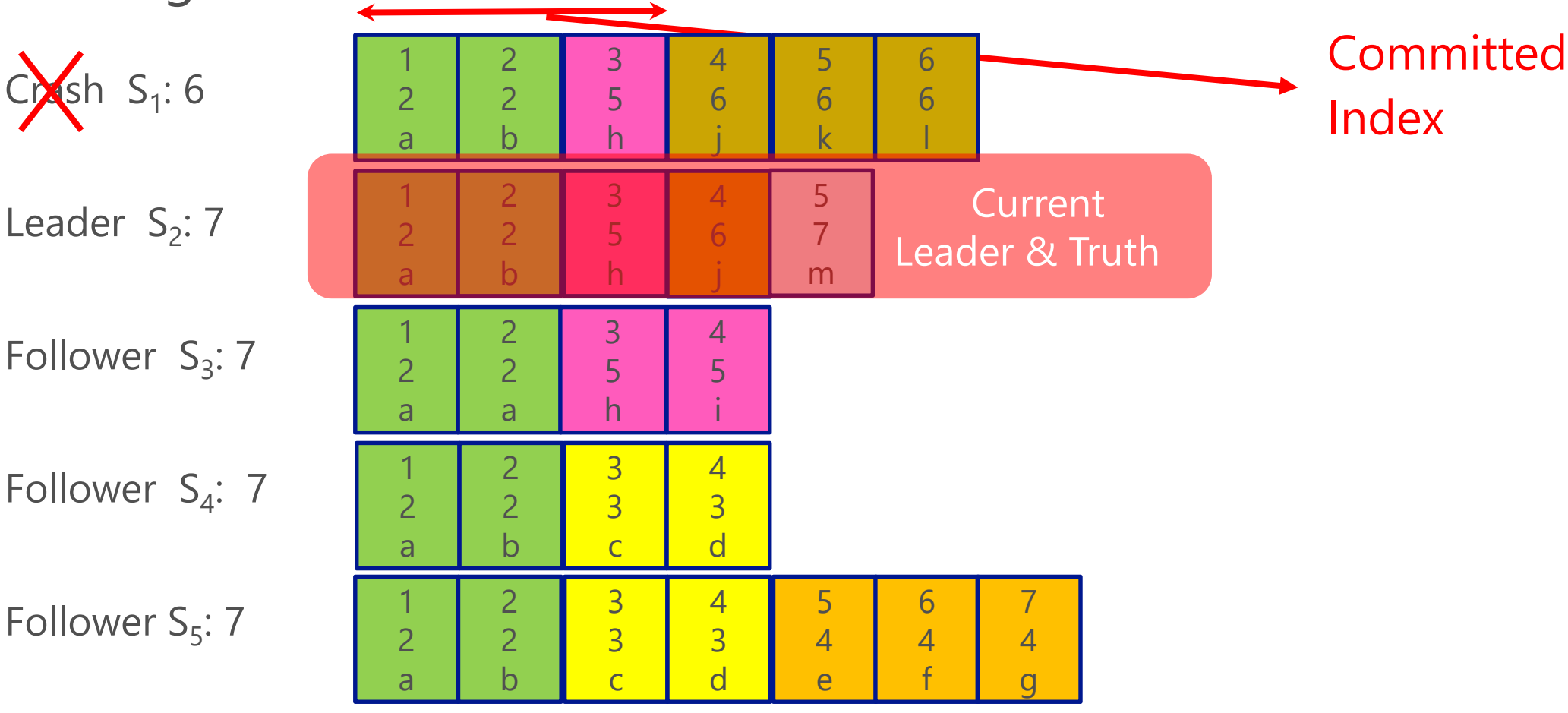


Safety

- Leader Selection Rule
- During election, a candidate is rejected if its log is outdated
- $(\text{lastTerm}_V > \text{lastTerm}_C) \,||\, (\text{lastTerm}_V == \text{lastTerm}_C) \,\&\&\, (\text{lastIndex}_V > \text{lastIndex}_C)$
- Result:
- If a Leader has committed a log entry, it will present in the logs of **all future** Leaders
  - **Otherwise, those nodes will not be able to get vote and become Leader in the first place**
- Leader will never overwrite entries in their logs (its log is the new truth)
- Server can only apply entries to state machine after they are committed

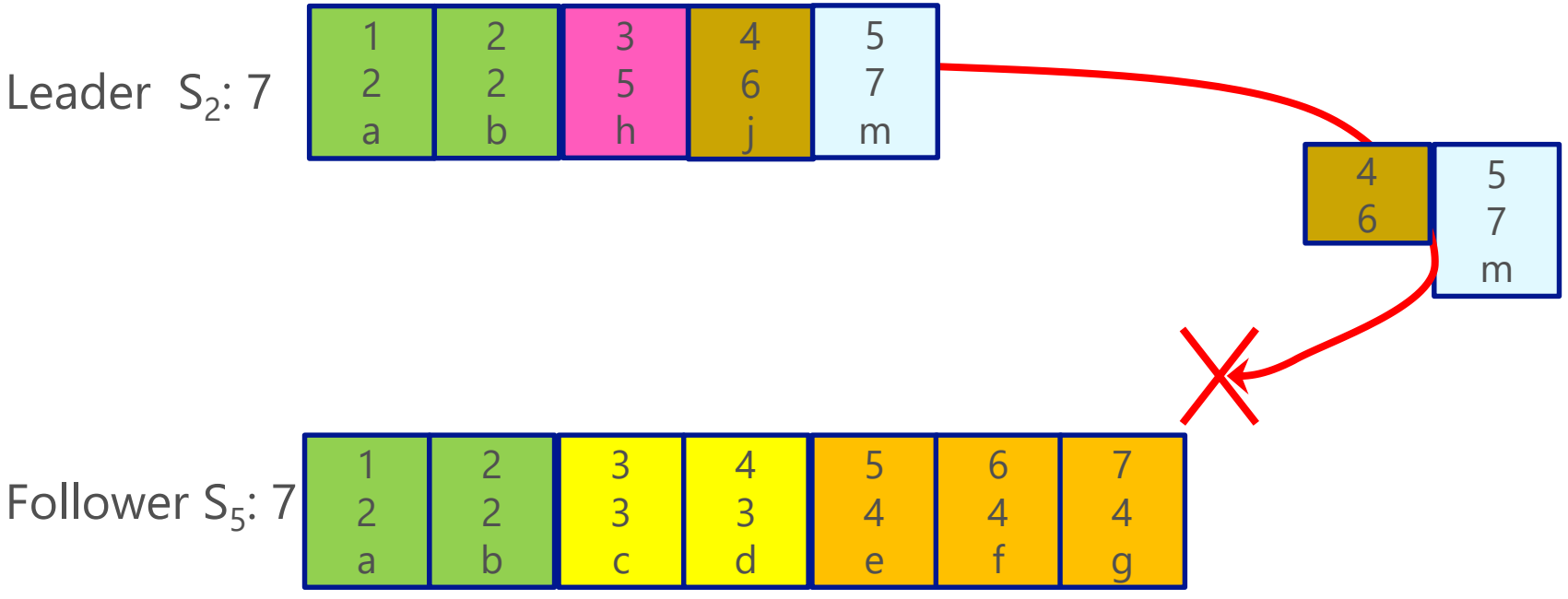
Repairing Follower's Log

- S_2 become the new Leader in term 7
  - Its log is the new truth, it will attempt to repair all its Follow's log



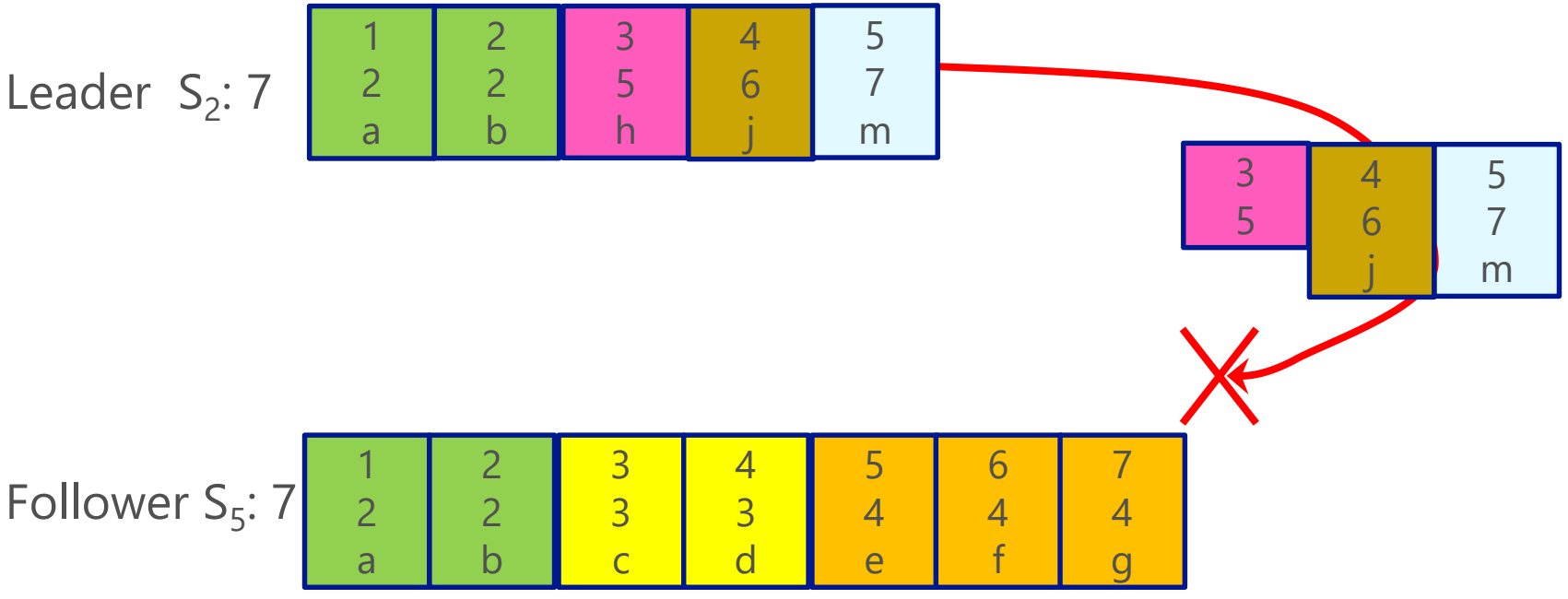
Repairing Logs

- S_2 Send appended entries at index 5, and the index & term of the entry before (at 4)
- S_2 find that at index 4, its term is 3, so request is rejected



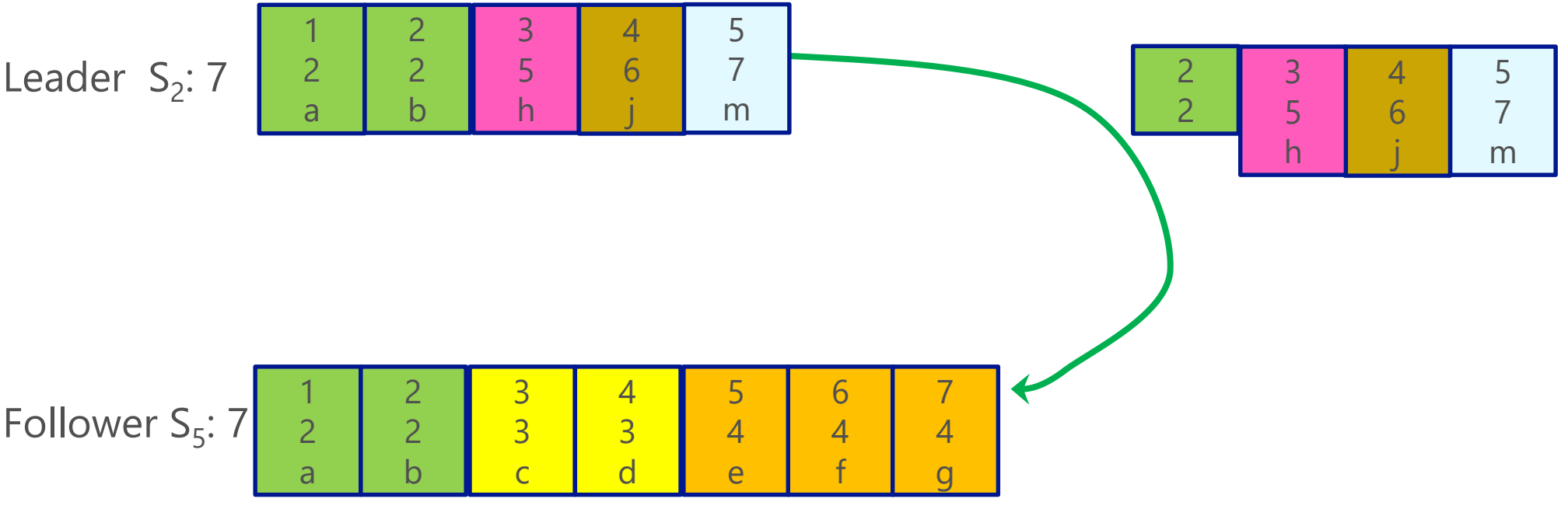
Repairing Logs

- S_2 Send appended entries at index 4, 5, and the index & term of the entry before (at 3)
- S_2 find that at index 3, its term is 3, so request is rejected



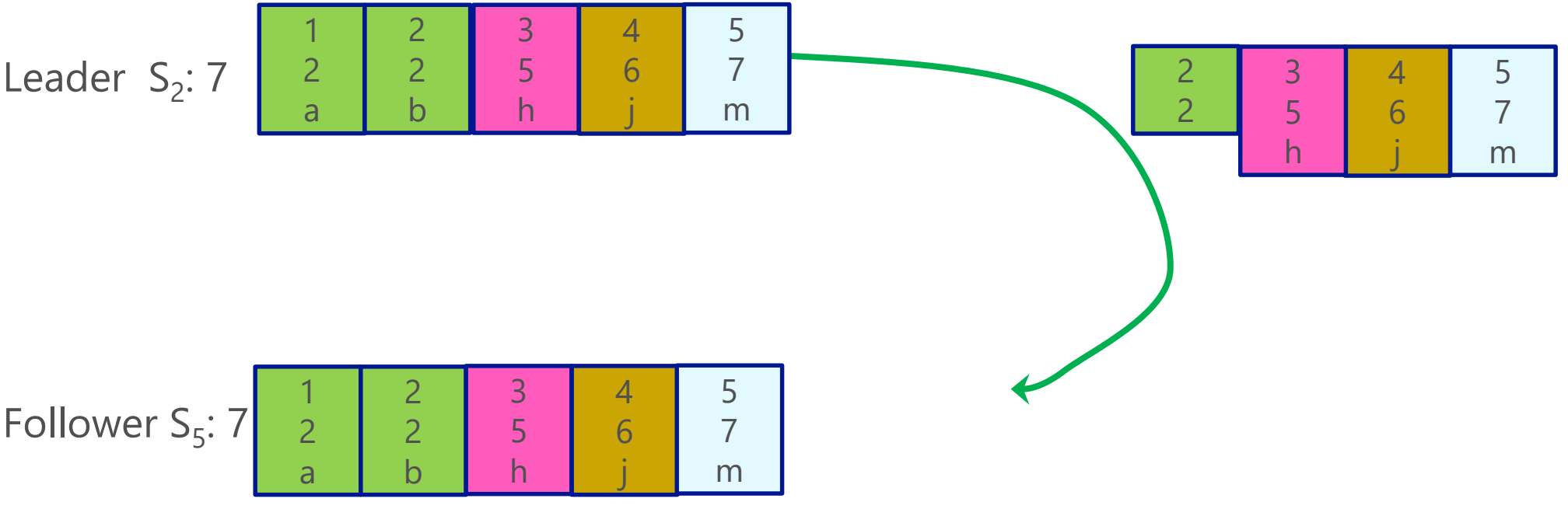
Repairing Logs

- S_2 Send appended entries at index 3, 4, 5, and the index & term of the entry before (at 2)
- S_2 find that at index 2, its term is 2, so request is accepted
- When follower overwrites inconsistent entry, it deletes all subsequent entries



Repairing Logs

- S_2 Send appended entries at index 3, 4, 5, and the index & term of the entry before (at 2)
- S_2 find that at index 2, its term is 2, so request is accepted
- When follower overwrites inconsistent entry, it deletes all subsequent entries



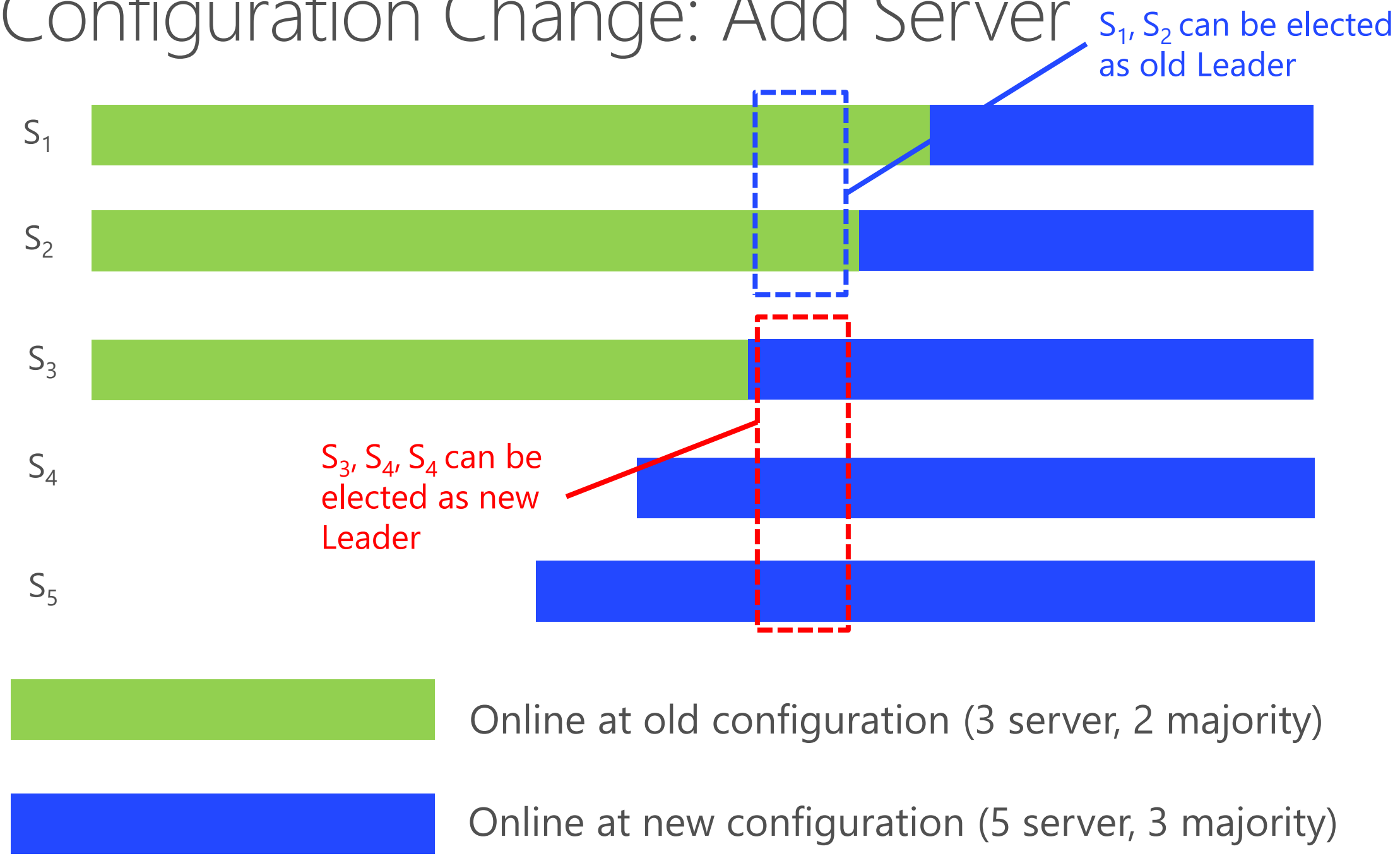
Client Protocol

- Send commands to Leader
  - If Leader is unknown, it may contact any server. If non Leader, it will redirect command to Leader
  - Leader does not ack until command has been logged, committed and executed by leader's state machine
- If request times out (e.g., Leader crashes):
  - Client reissues command to some other server, and retry request with new Leader
  - It is possible for unacked command to be executed/committed in Leader's Log (with Leader failing before ack)
  - To prevent a command to be executed multiple times, client need to embeds **unique id** in each command
    - Before accepting command, Leader can check its log for duplication

Configuration Change (Add/Remove Machine)

- Raft Configuration
  - Each server: name (ID) & address,
  - Quorem (what constitutes a majority)
- Configuration changes
  - Remove failed machine
  - Add new machine
  - Change degree of replication
- Safety condition
  - Only one leader per term

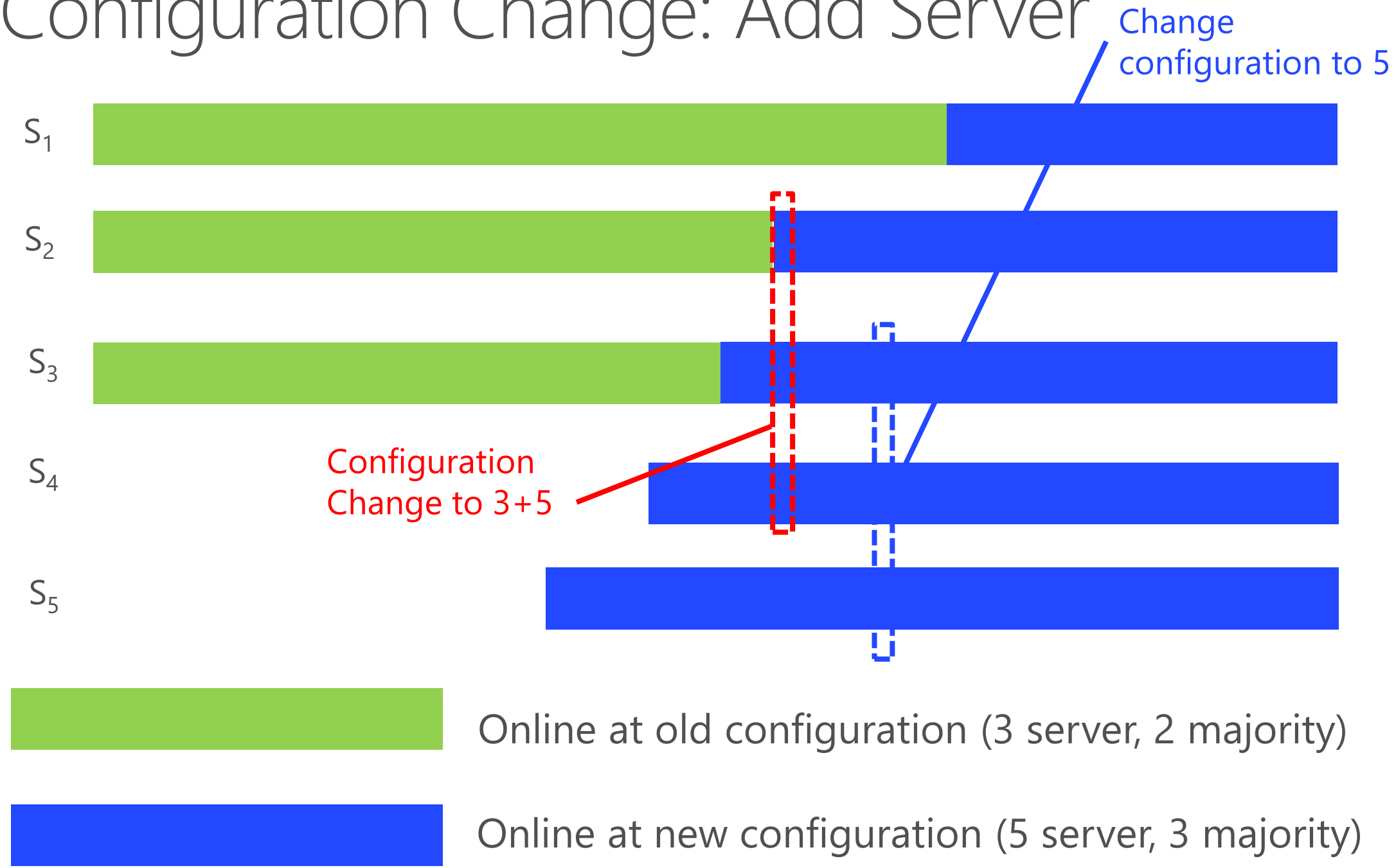
Configuration Change: Add Server



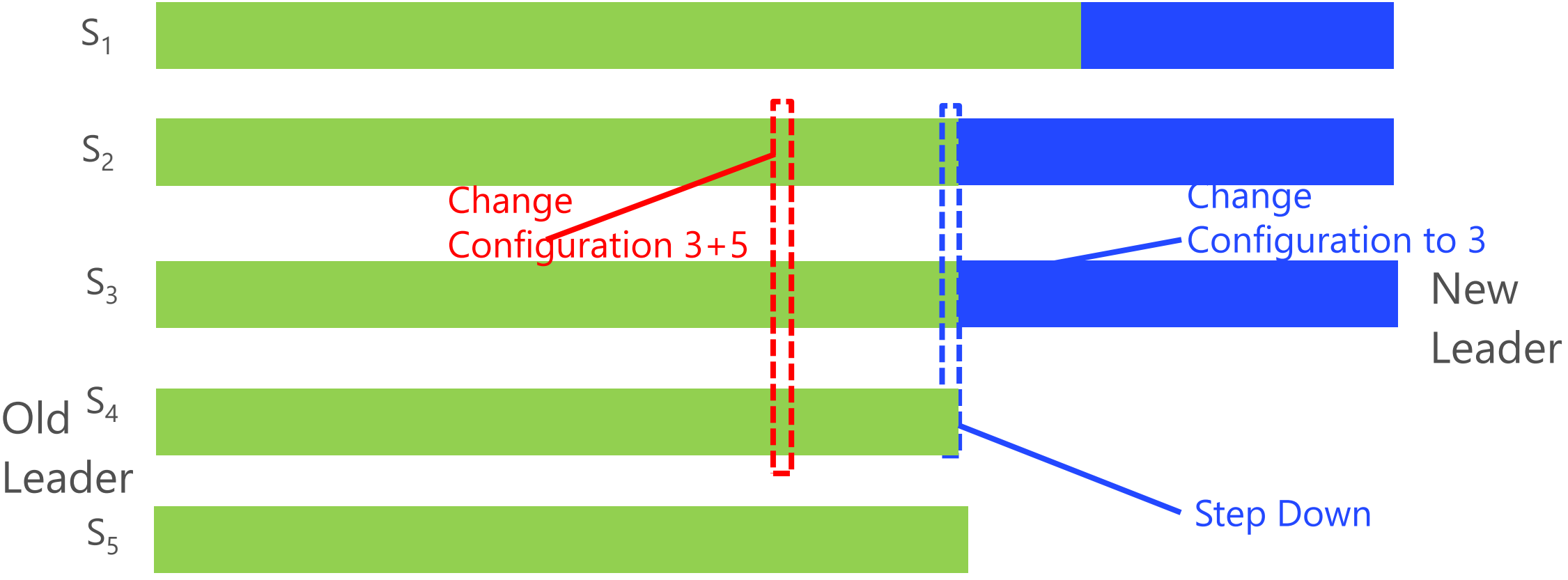
Configuration Change: Joint Consensus

- Leader receives change of configuration from C_{old} to C_{new}
- Leader creates a command to change to configuration $C_{old, new}$ and commits the command to cluster with majority that covers a majority of both C_{old} and C_{new}
  - **A majority of nodes cover both C_{old} and C_{new} need to be live for the algorithm to move forward**
- It then creates a command to change to configuration C_{new} and commits to a majority of both C_{old} and C_{new}
- If Leader is not in C_{new}, it will steps down and allow a new election
- From that point onward, command only need to be committed to C_{new}

Configuration Change: Add Server



Configuration Change: Remove Servers



Online at old configuration (5 server, 3 majority)

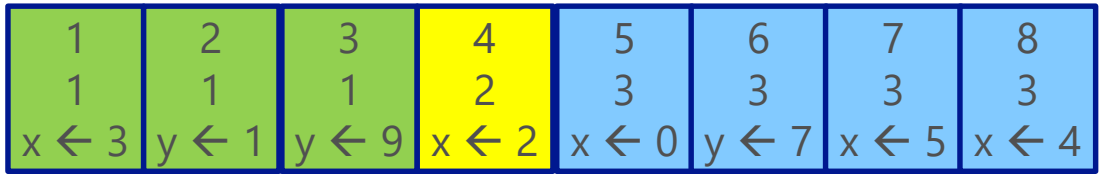
Online at new configuration (3 server, 2 majority)

TLA+ Implementation & Model Checking

TLA+ Variables

- **Server Variable**

currentTerm: **(persisted)**
state: { Follower, Leader, Candidate}
votedFor: **(persisted)**
log: **(persisted)**



commitIndex

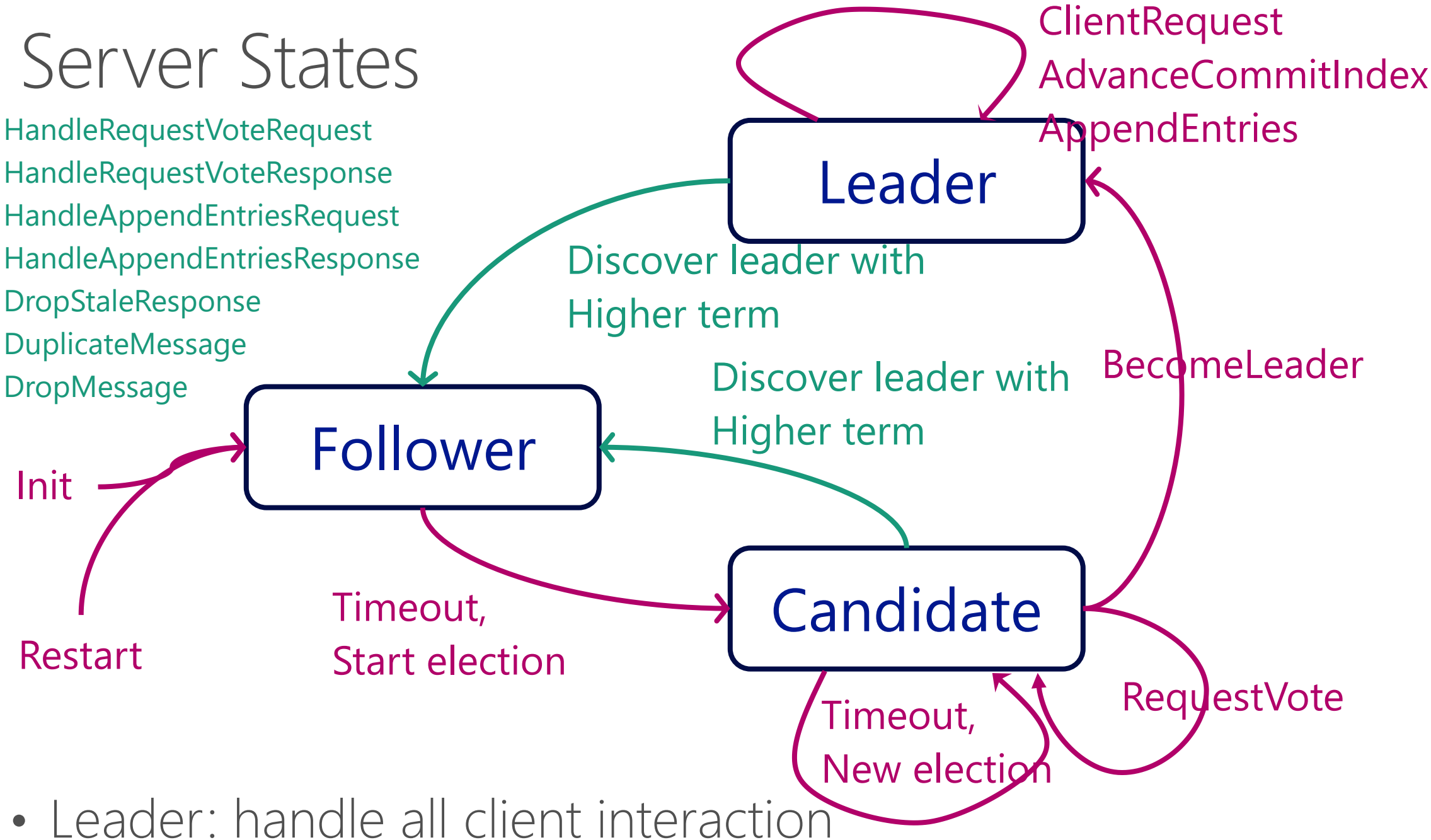
- **Candidate Variable**

votesResponded
votesGranted

- **Leader Variable**

nextIndex
matchIndex

Server States



- Leader: handle all client interaction
- Follower: completely passive

A example

- TLC Model checking
  - Find error if currentTerm is not persisted through crash
  - Find error if voteFor is not persisted through crash
- Tricks on TLA+
  - Reduce state space to be explored by using state restriction

Conclusion

Conclusion

- Raft: Replicated Log for replicated state machines
  - **E.g., etcd: reliable distributed key-value store maintained by CoreOS**
- Term & Leader election
- Safety and consistency:
- Configuration changes