Distributed Systems

Spring Semester 2020

Lecture 5: Raft

John Liagouris
liagos@bu.edu

RAFT

- Fault Tolerance through Replicated State Machines (RSM)
- RAFT: Much more complete design than the PAXOS paper
- Fundamentally: Raft bakes in notion of a leader and log

RAFT

- Fault Tolerance through Replicated State Machines (RSM)
- RAFT: Much more complete design than the PAXOS paper
- Fundamentally: Raft bakes in notion of a leader and log

https://en.wikipedia.org/wiki/Paxi

Big Picture

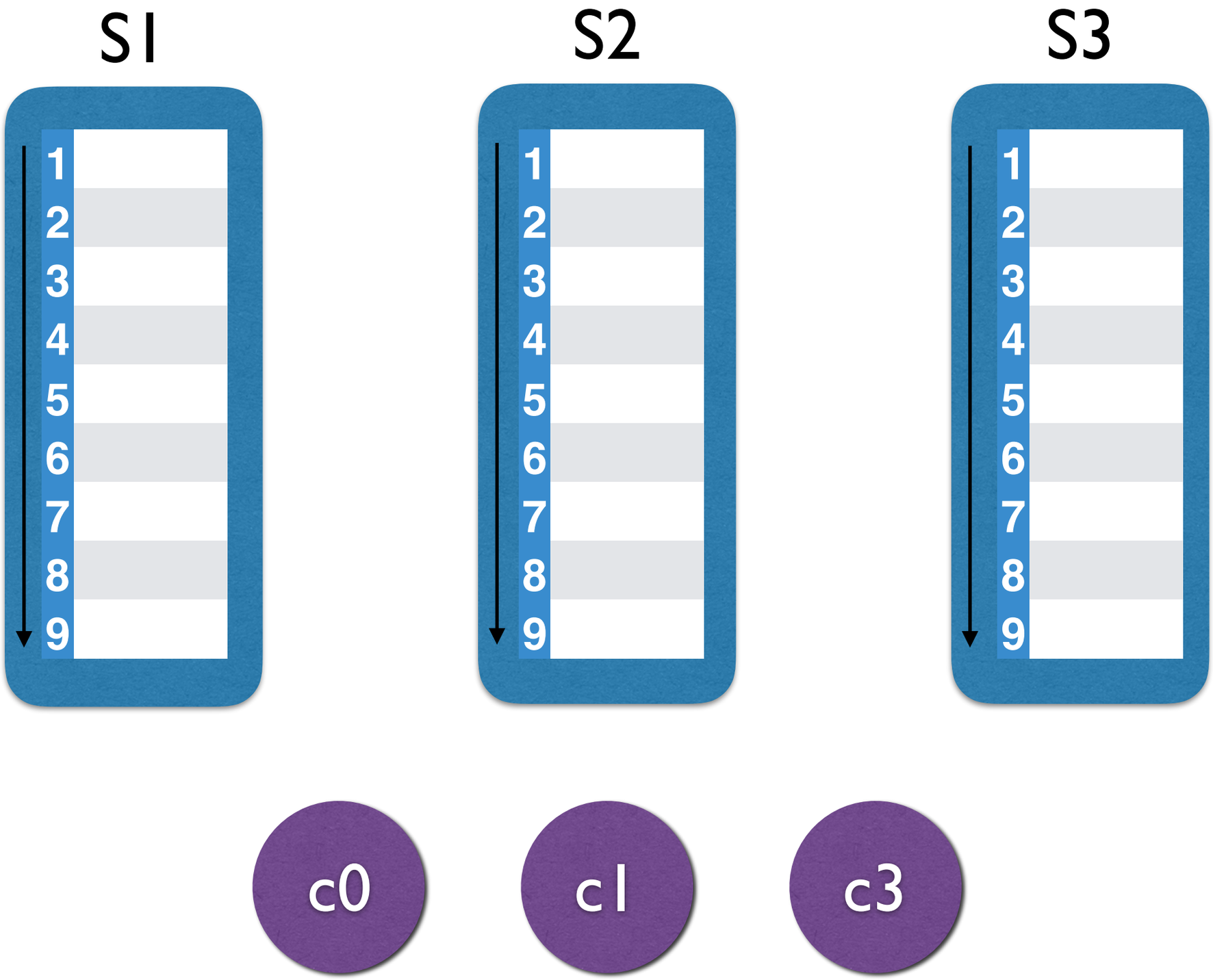
Bunch of Clients

S1

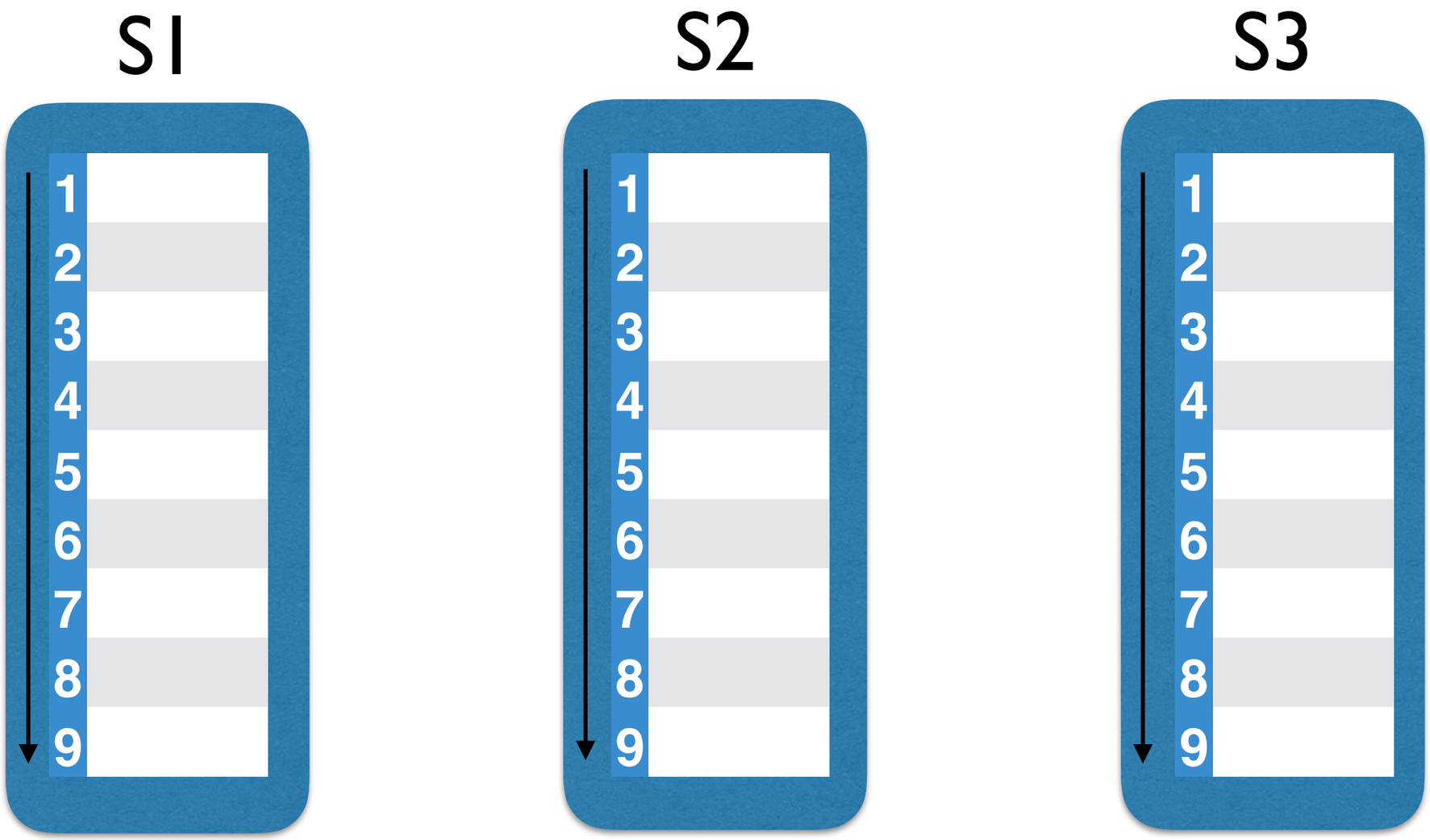
S2

S3

and a bunch of servers



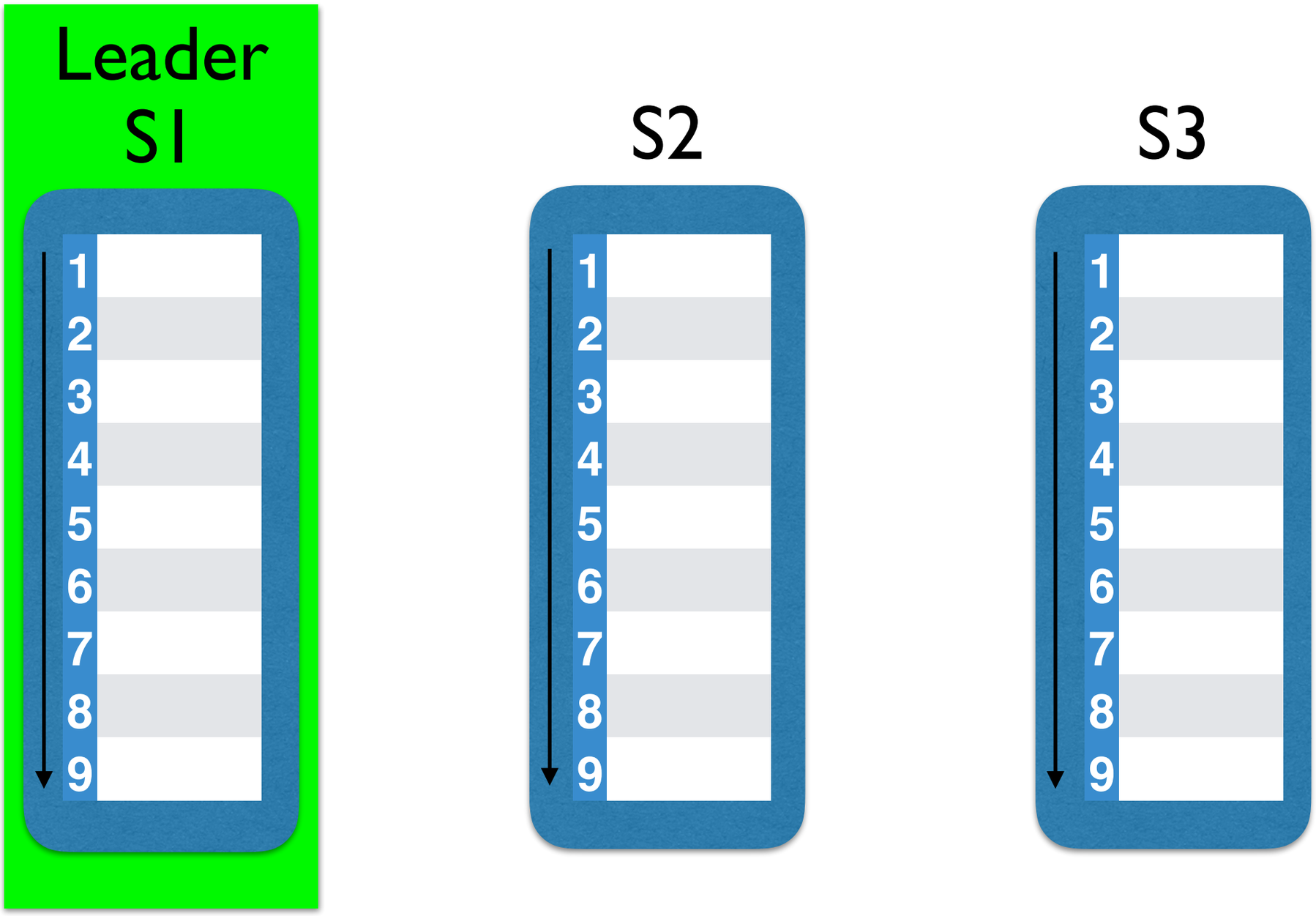
Servers are accumulating logs of client commands



Servers are accumulating log of client commands

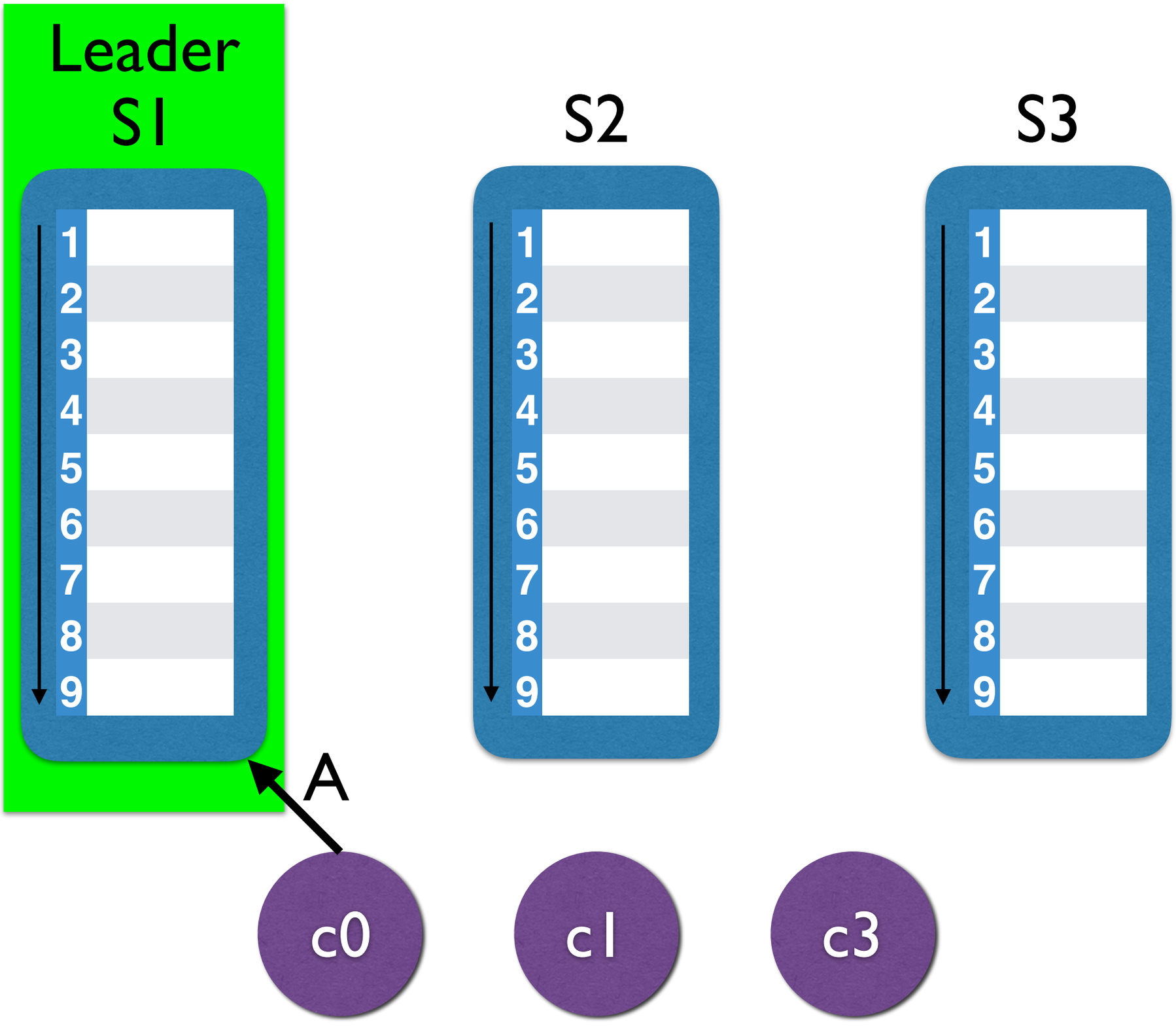
which again we hope to keep identical

Leader



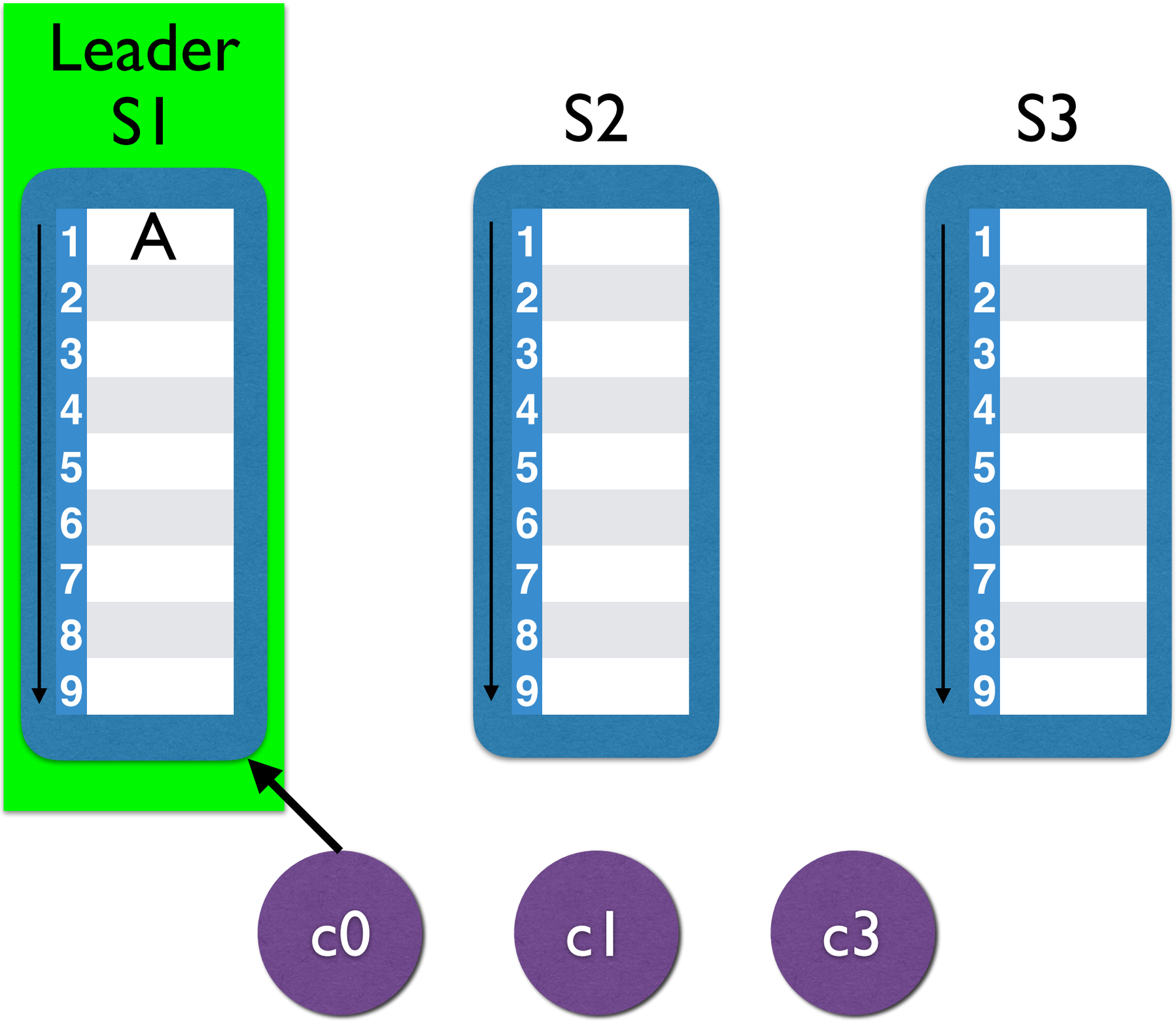
At any given time there is a Leader

Leader



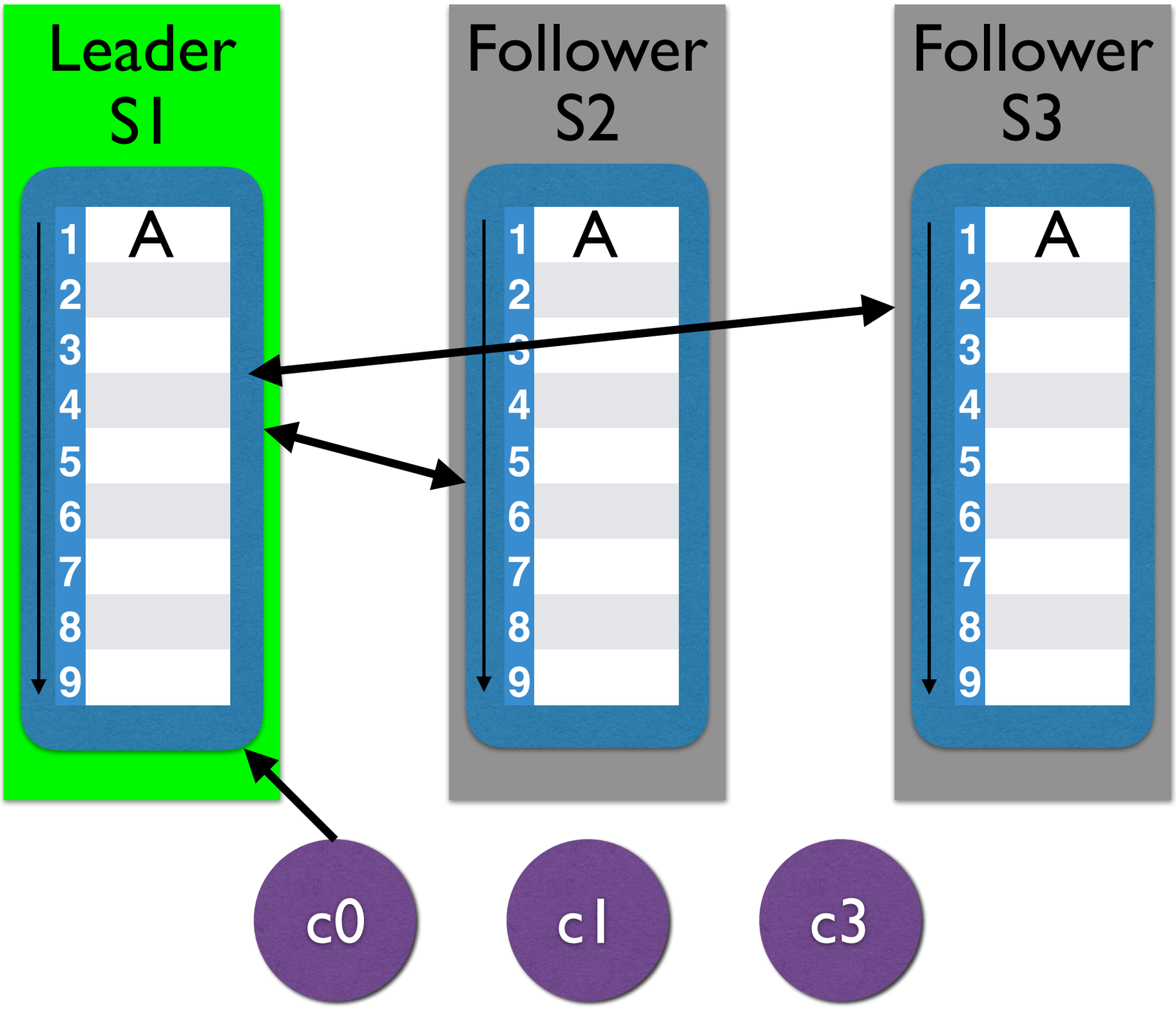
Clients are just supposed to send requests to the Leader

Leader



Leader puts request into its log

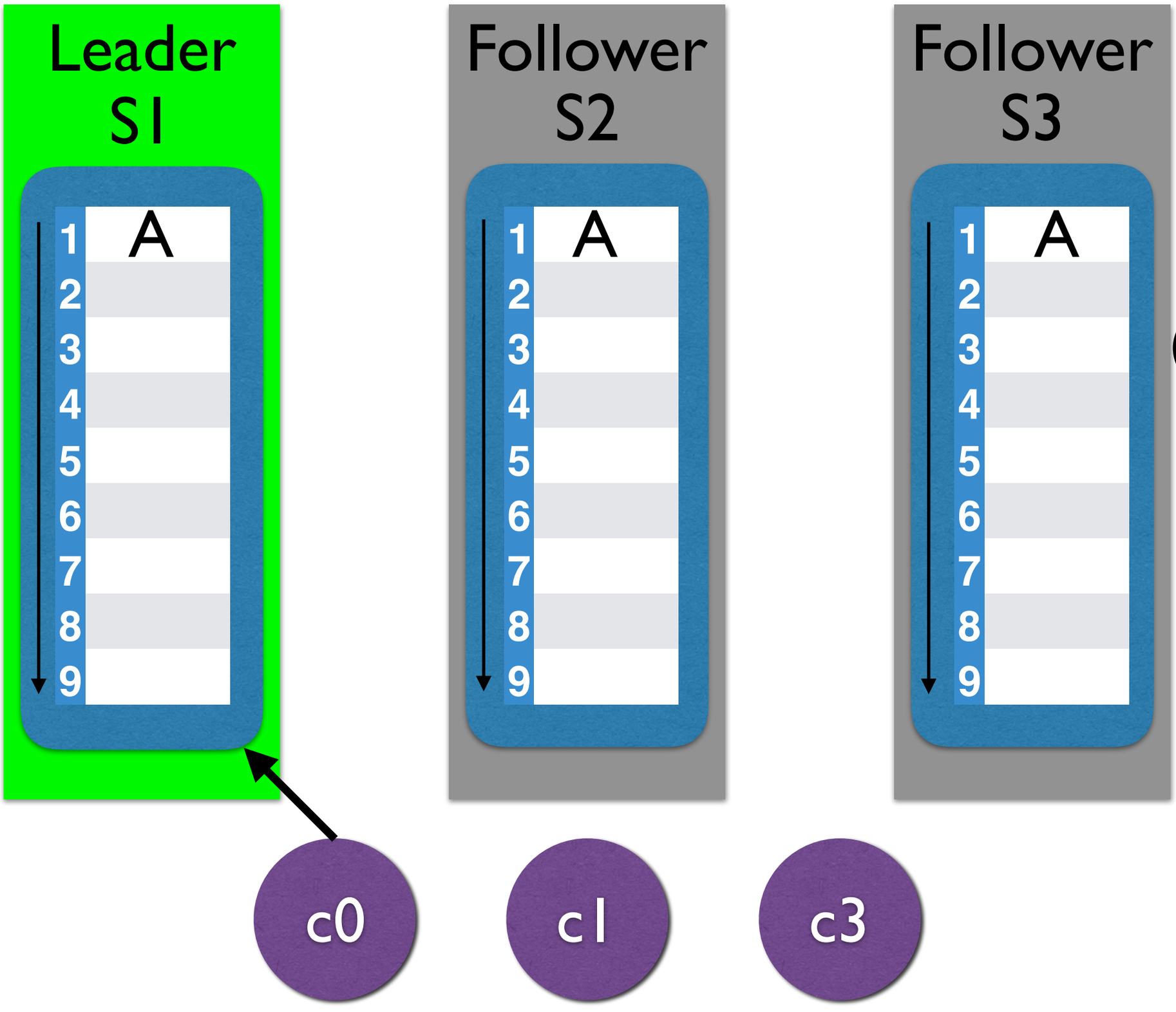
Leader



Other servers are Followers which leader sends command for replication

eg. please stick this value into your log at the same place

Leader



Once the leader believes majority of the followers (including itself) have command in there log it announces commitment and that command can be executed

What matters

- NOT: is it more or less understandable that PAXOS
- NOT: is it better, faster, more efficient
- RATHER: It is a relatively complete tutorial on how to build a RSM system
- Read it to learn how people build this stuff and basis for remains labs

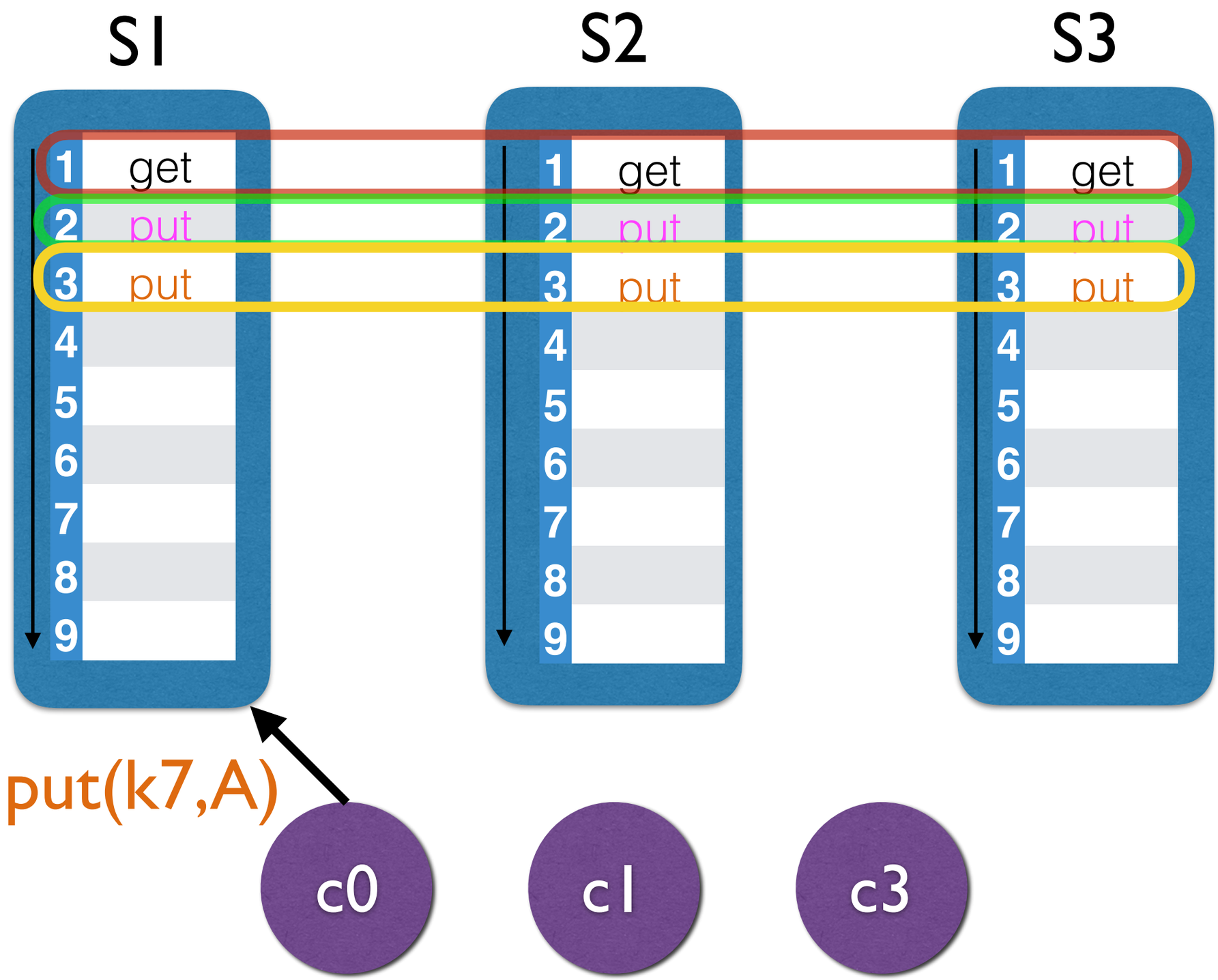
What matters

- NOT: is it more or less understandable that PAXOS
- NOT: is it better, faster, more efficient
- RATHER: It is a relatively complete tutorial on how to build a RSM system
- Read it to learn how people build this stuff and basis for remains labs

What matters

- NOT: is it more or less understandable that PAXOS
- NOT: is it better, faster, more efficient
- RATHER: It is a relatively complete tutorial on how to build a RSM system
- Read it to learn how people build this stuff and basis for remains labs

PAXOS — lots of messages



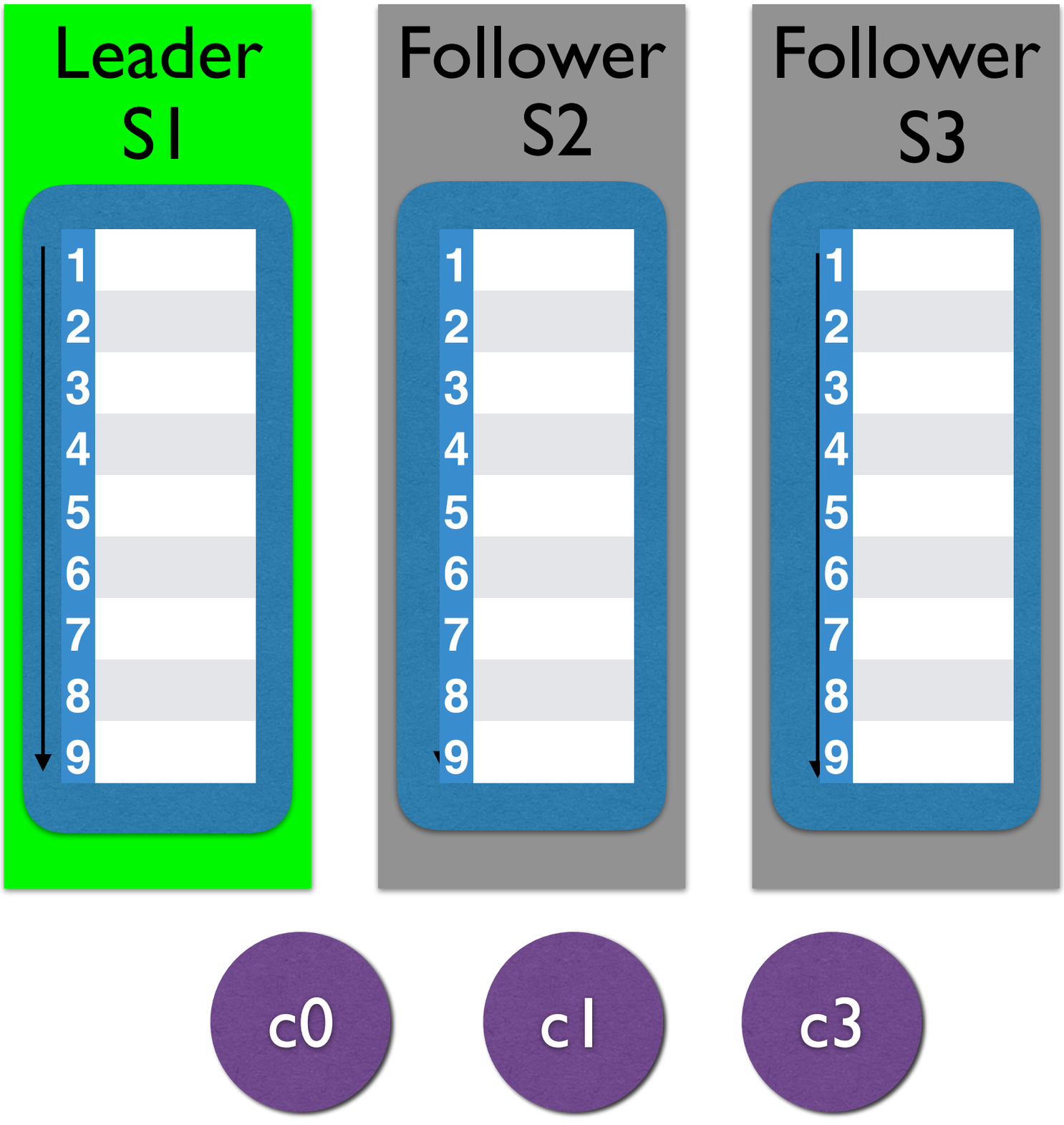
whole bunch of independent PAXOS instances — too slow & requires lots of messages

— In practice optimize so that common case does not need this

Some other differences

- Single leader reduces probability of dueling proposers
- Might happen in elections but expect this to be rare
- Note once a leader is elected — during term of leader — there really is not agreement
  - It just tells everyone what to do ;-)
- Much easier to know who has complete state
  - Logs don't have holes

No Leader Failure



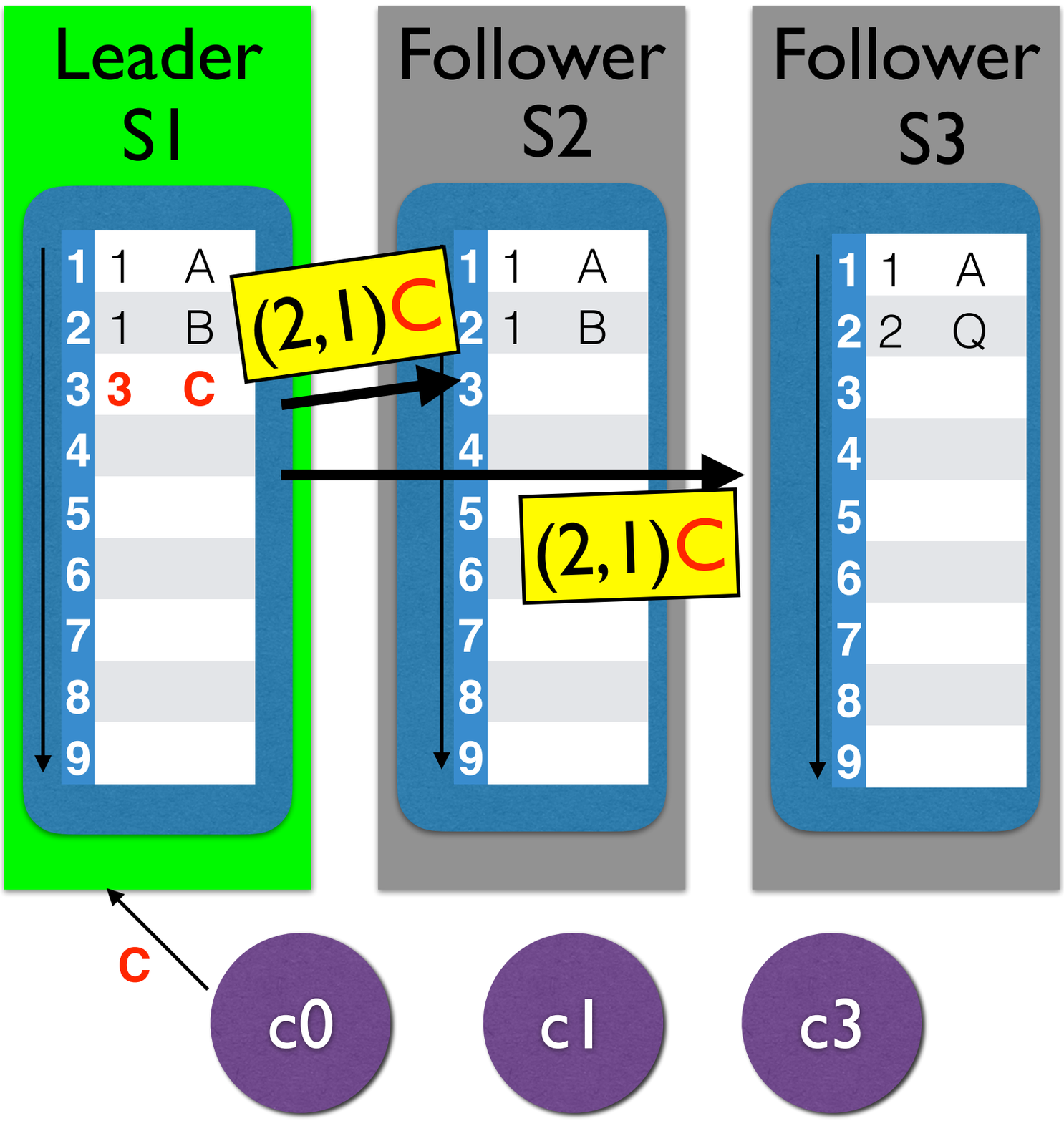
BUT:

- Follower's crash
- network looses messages

What are we hoping for?

1. Tolerate failure of a minority of followers
2. Convergence on one version of the log
3. Only execute client requests (and thus reply to client) when it is "committed"— can't go back and unexecuted or un-tell client a response

No Leader Failure



LOGS same by force:

Leader

1) Appends to its log.

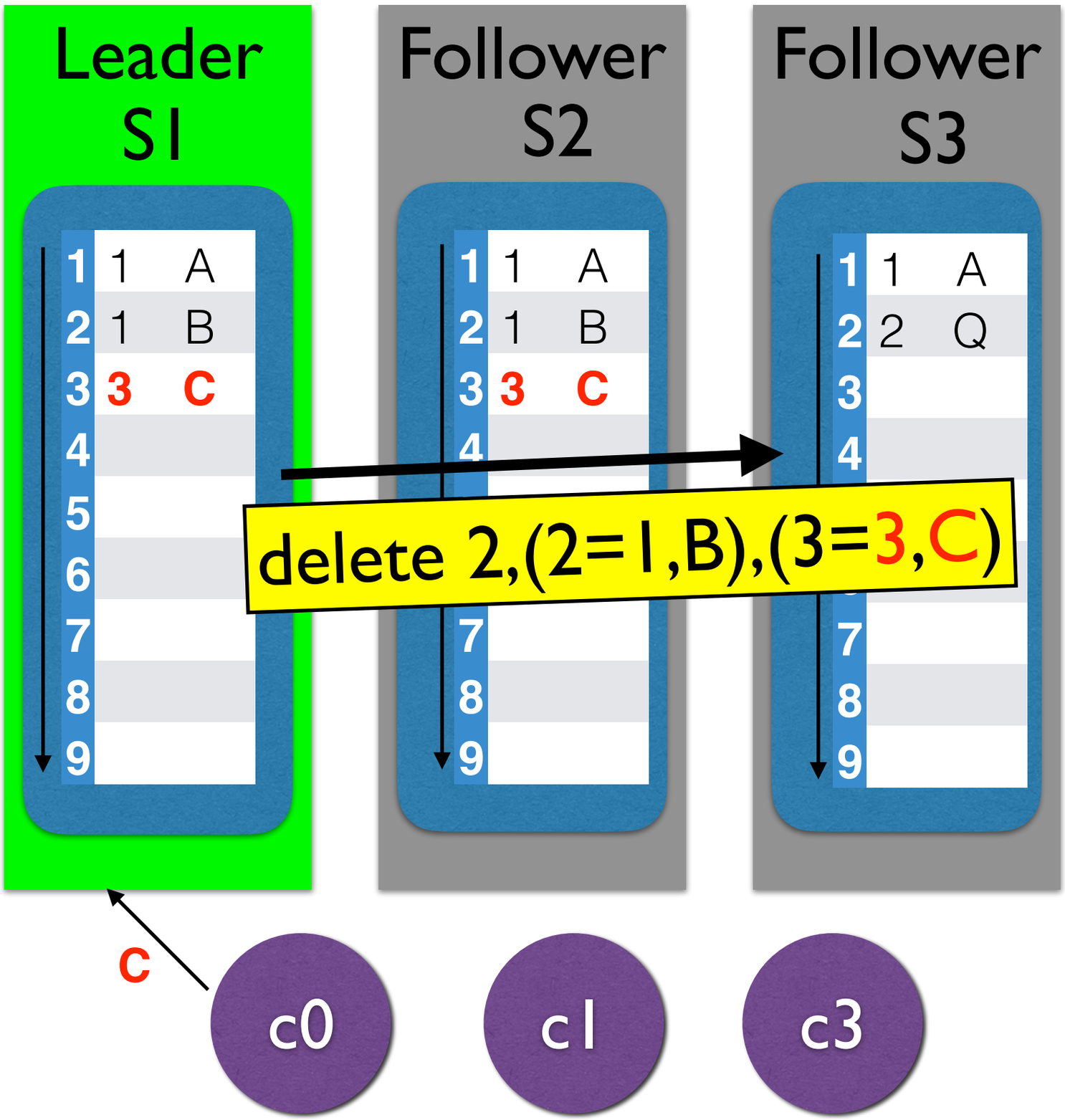
2) In parallel Sends index and "term" of prior entry as part of AppendEntries RPC

Followers

if prior index and term not in log REJECT

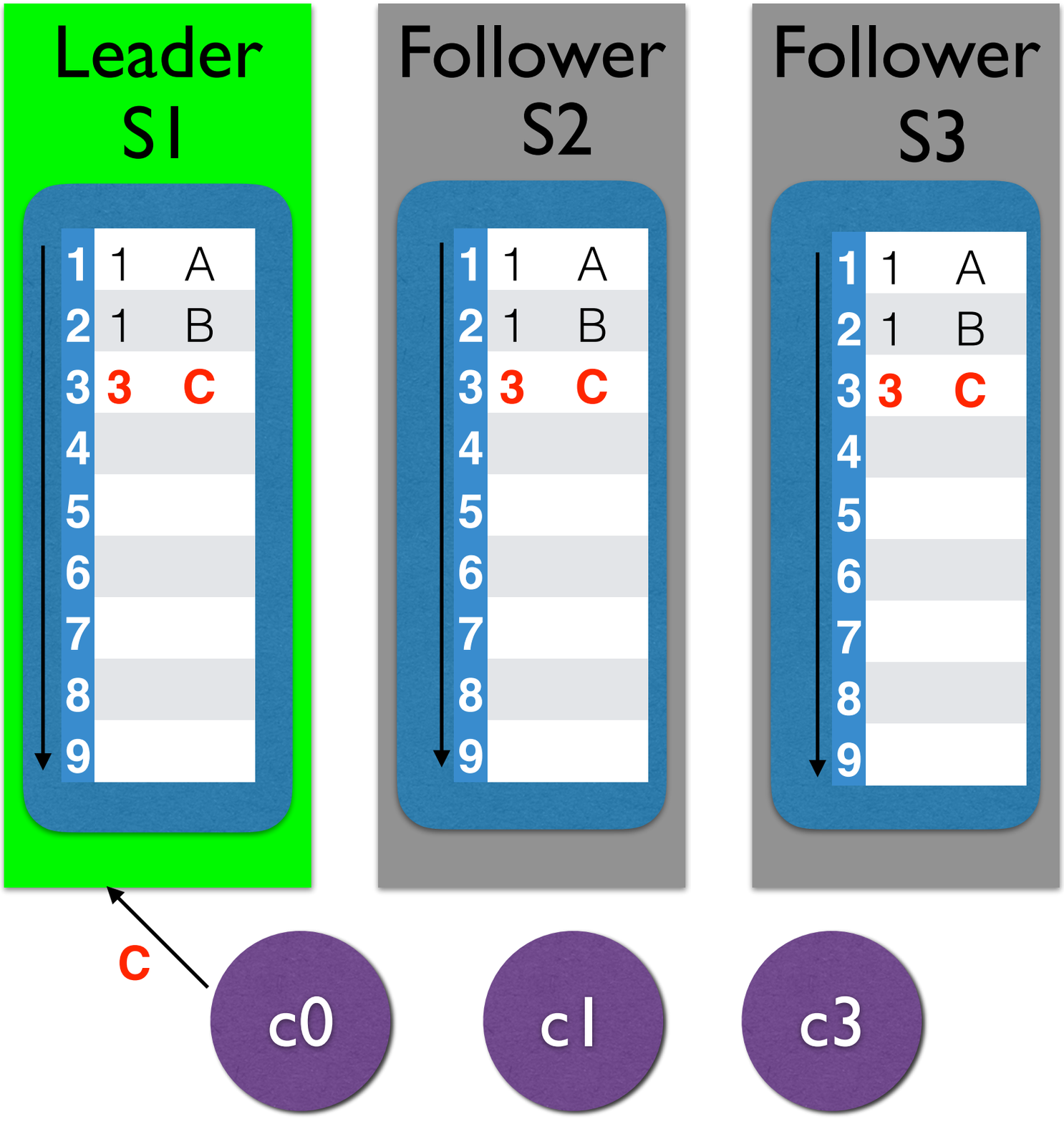
if match then log is consistent!

No Leader Failure



LOGS same by force: If a follower is found inconsistent then leader rolls back follower and then rolls it forward

No Leader Failure



Leader does not care what is in your log it forces its view on all followers!

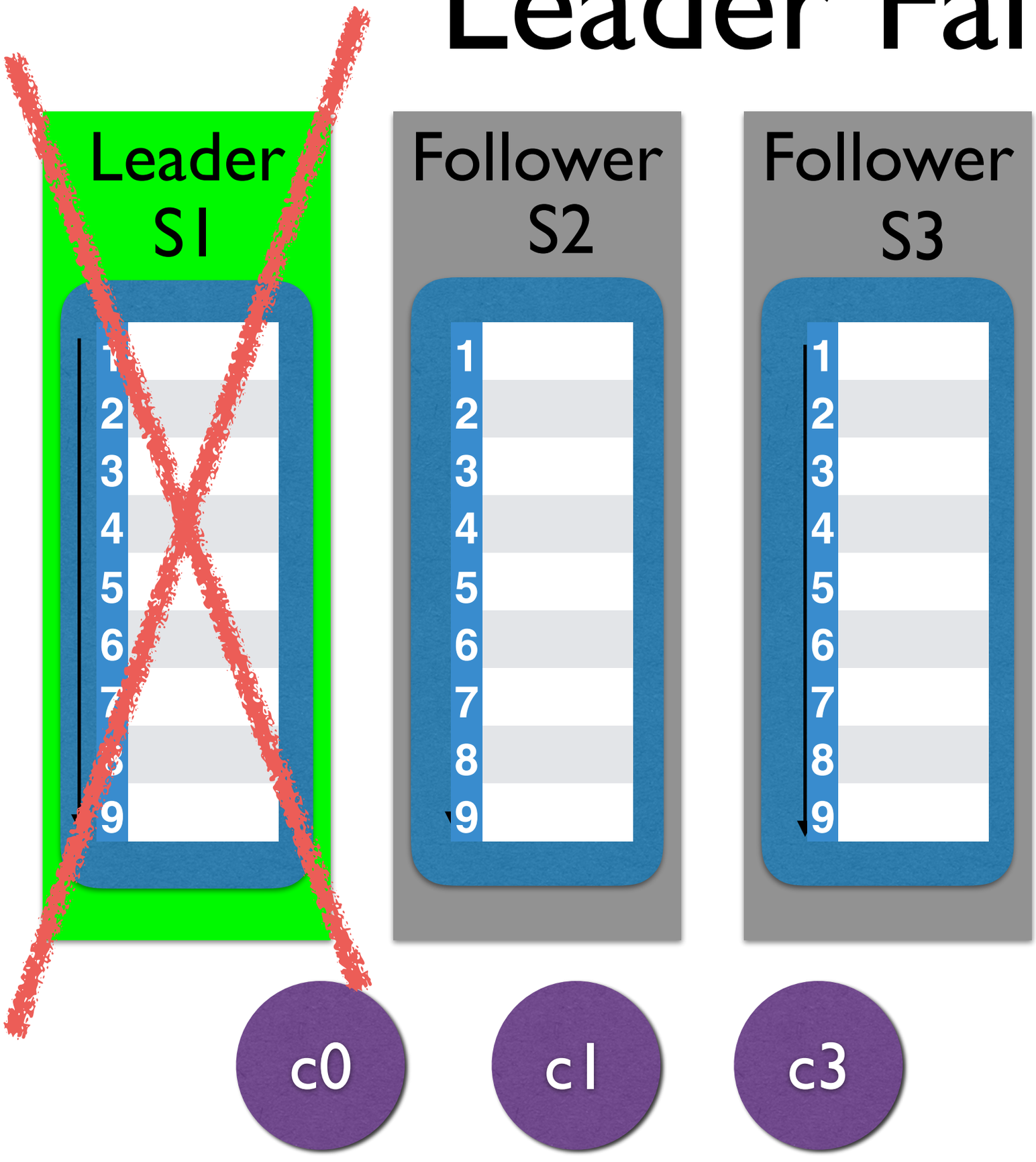
2 common divergence

- Follower crashed for a while (disconnected)
  - The leader simply rolls follower forward
  - A prior leader crashed while sending out some AppendEntries messages
    - Then leader both rolls follower back and forward

Consistency through leader driven reconciliation

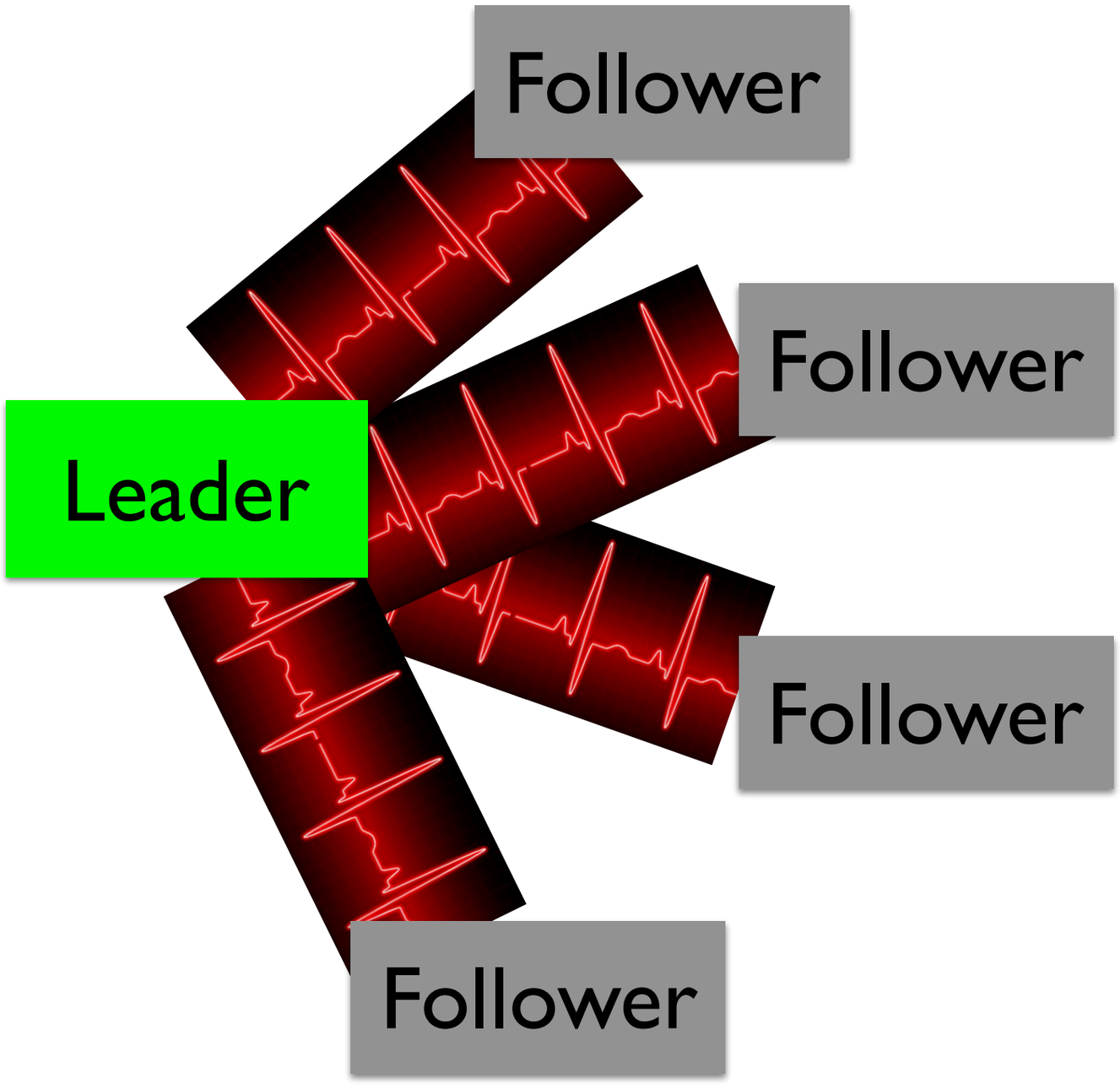
1. Tolerate failure of a minority of followers
    - various times states “wait for majority”, “when majority”,…
2. Convergence on one version of the log
    - by reconciliation to leader’s log
3. Only execute “committed” requests
    - Not a problem if leader does not crash and waits for a majority to respond that append is durable — then sends out a commit point (piggy backed or AppendEntries RPC)

Leader Failure



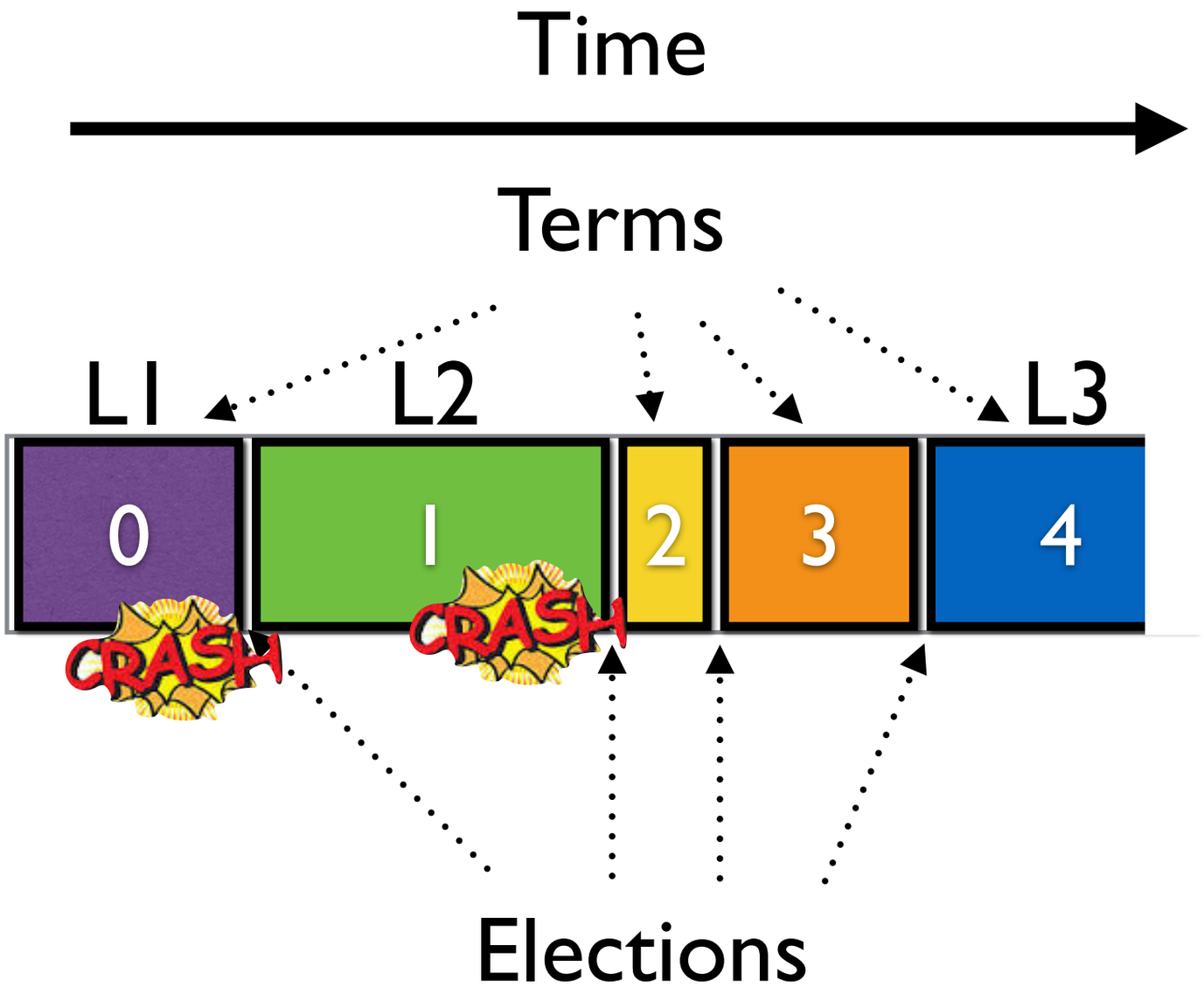
Now things get interesting

HeartBeat to Followers



- AppendEntries done repeatedly by leader to all followers (null if necessary)
- Followers initiate an "Election" if heartbeat timeout expires

Time broken into Terms



1. Detect that a leader has "crashed"
2. Start an election to initiate a new term with a new leader
3. New leader has to pick up the pieces

Remember a leader can crash in the middle of anything!

Leader Election

1. Fewer than 2 leaders any time (0 and 1 ok)
2. Ideally greater than 0

Raft has separate mechanisms for these

Fewer than 2 (at most one) leader elected per term

Figure 2 Rules

- When a server gets a vote request it is allowed to vote yes or no BUT it is only allowed ONE VOTE in any given term — eg it can only say yes to one candidate
- candidate must collect yes votes from a majority to become the leader

Therefore only one candidate become the leader for a term

Reason to use the Majority

- Only requiring majority (not everyone) means that you can tolerate failure of a minority!
  - Avoids split brain
  - Any two majorities must overlap with one server — prior majority must share a server with current majority — ensures continuity

New leader will be sure to know about all previous decisions (committed log entries)

Of course this kind of election could fail!

Right?

eg. Everyone detects failure and starts an election voting for themselves — can't vote for anyone else then and no-one can get a majority

Failed Elections

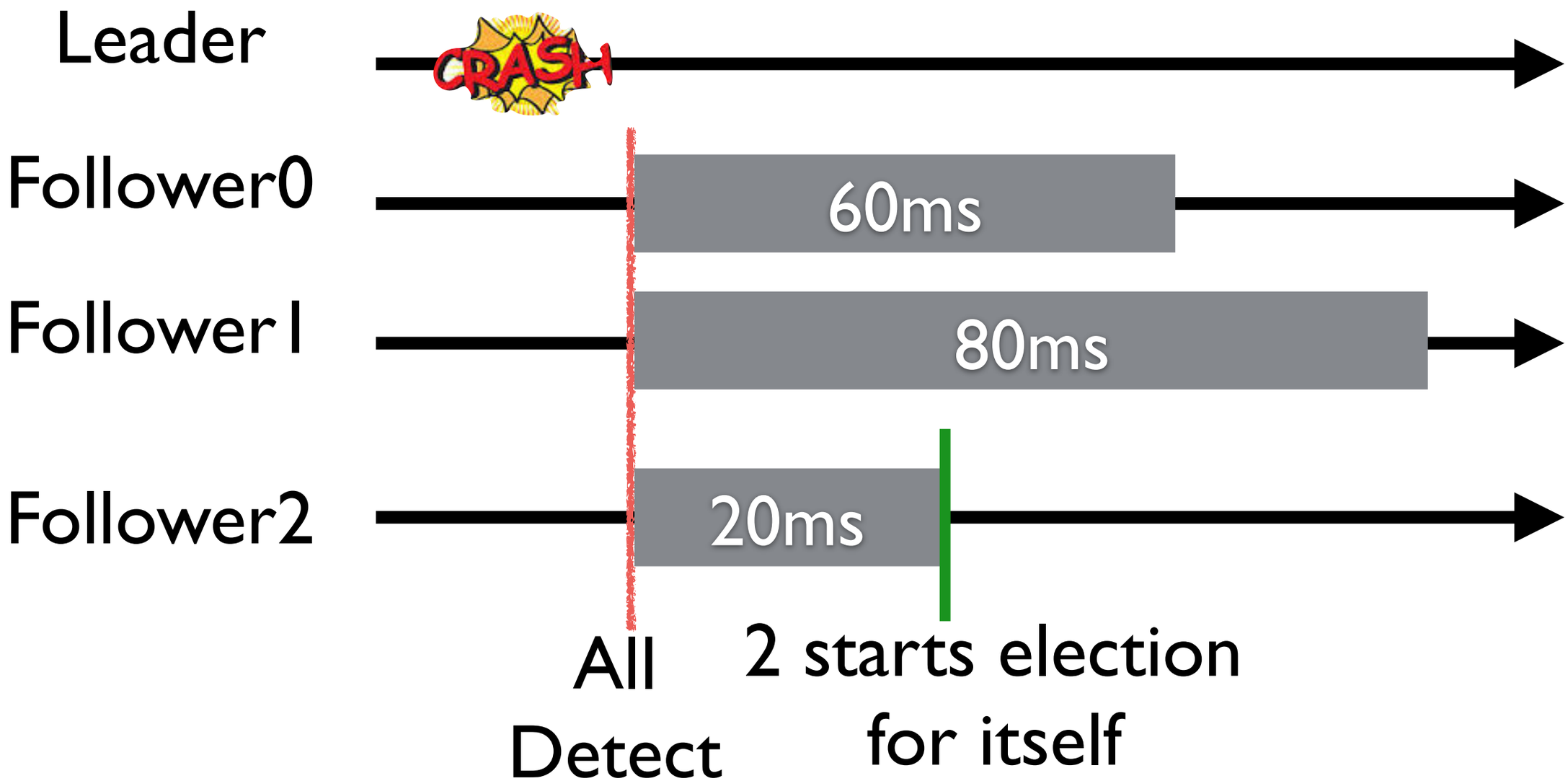
- Nothing is particularly wrong a new term will be initiated and an associated election
- But….
- So how do we ensure there is more than zero leader leaders ever elected

Probabilistic approach : can’t stop any particular election from failing but with high probability some election will succeed

Randomized Delays

- Split votes will unlikely happen repeatedly
  - When a server wants to become a candidate it picks a random number and wait
  - If no-one was elected in that amount of time then actually start the election for oneself and ask for votes

Intuition for Random Delays



If 2 can finish election in 40ms we for sure avoid a split vote - uncontested election (base delay matters — Tune as needed)

Majority and Random Delays are tricks that come up again and again in DS

Notice another pattern in the design

Breaks hard problem down and applies two different approaches that complement each other

1. Critical Safety property — at most one leader — addressed by a **hard** mechanism (must have a majority) that can fail leading to retry
2. Performance/Liveness — softer probabilistic scheme — retry and hopefully eventually succeed — no impact on correctness

A way of thinking: Separate safety from performance/ liveness