

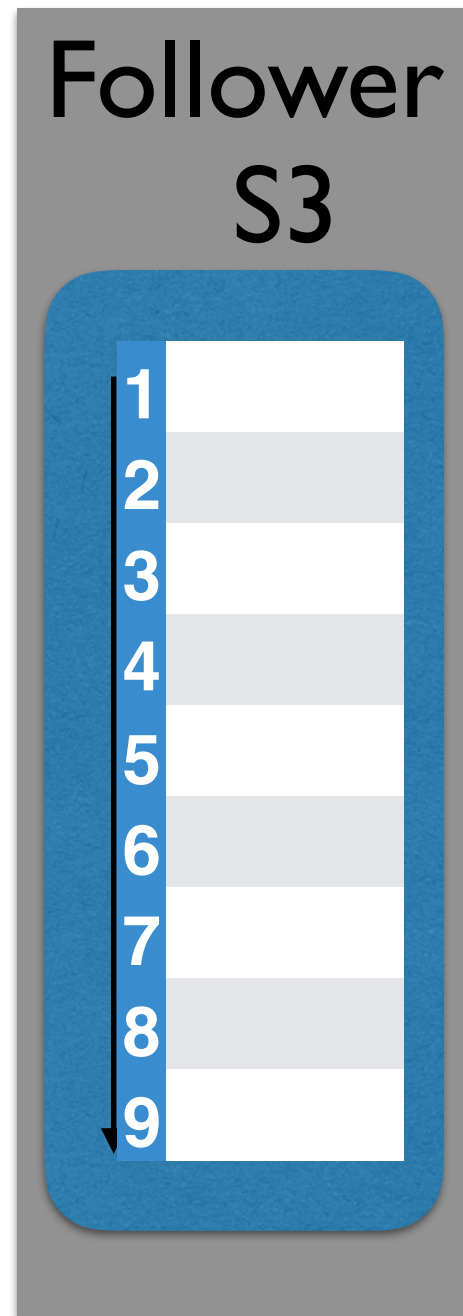
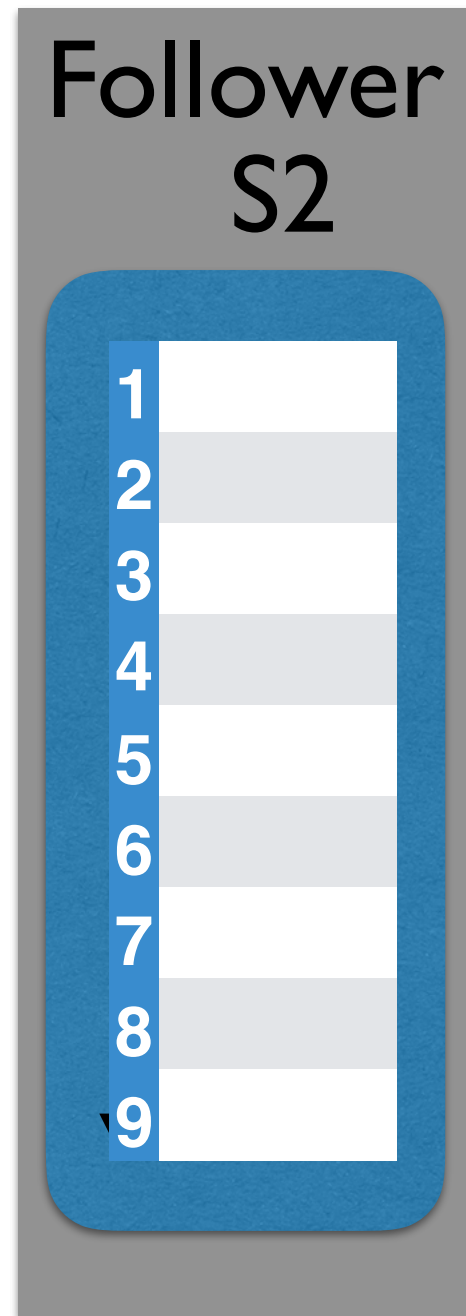
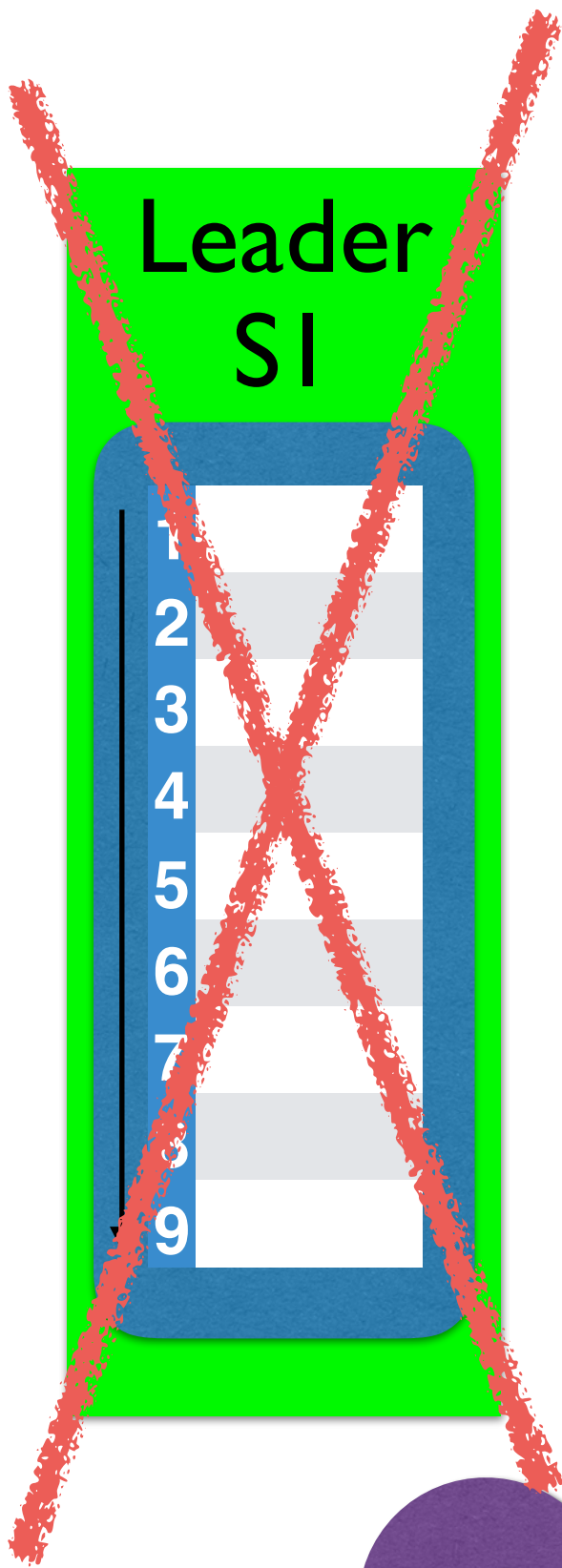
# Distributed Systems

**Spring Semester 2020**

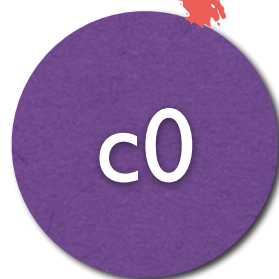
Lecture 6: Raft

John Liagouris  
[liagos@bu.edu](mailto:liagos@bu.edu)

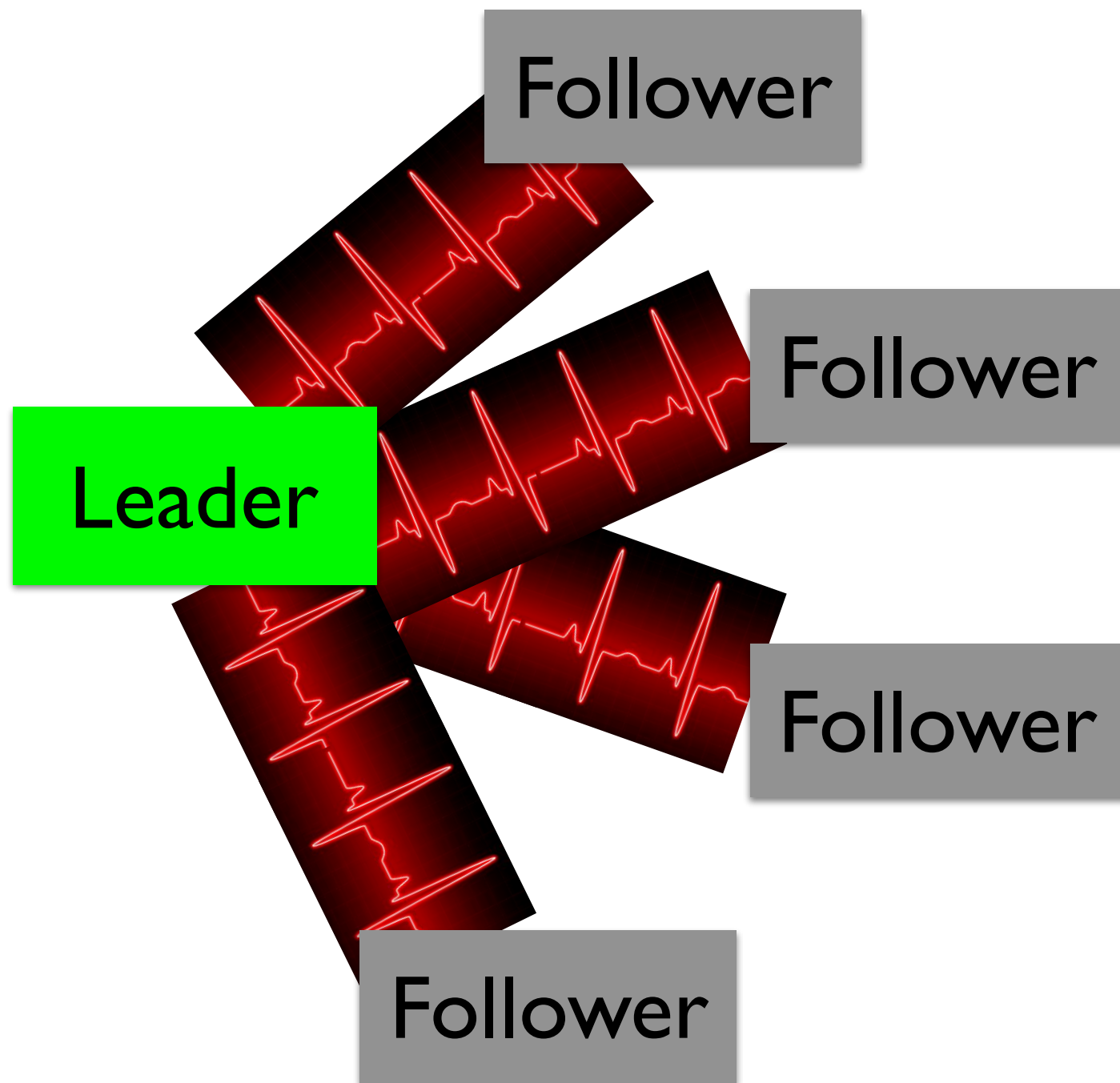
# Leader Failure



Now things get interesting

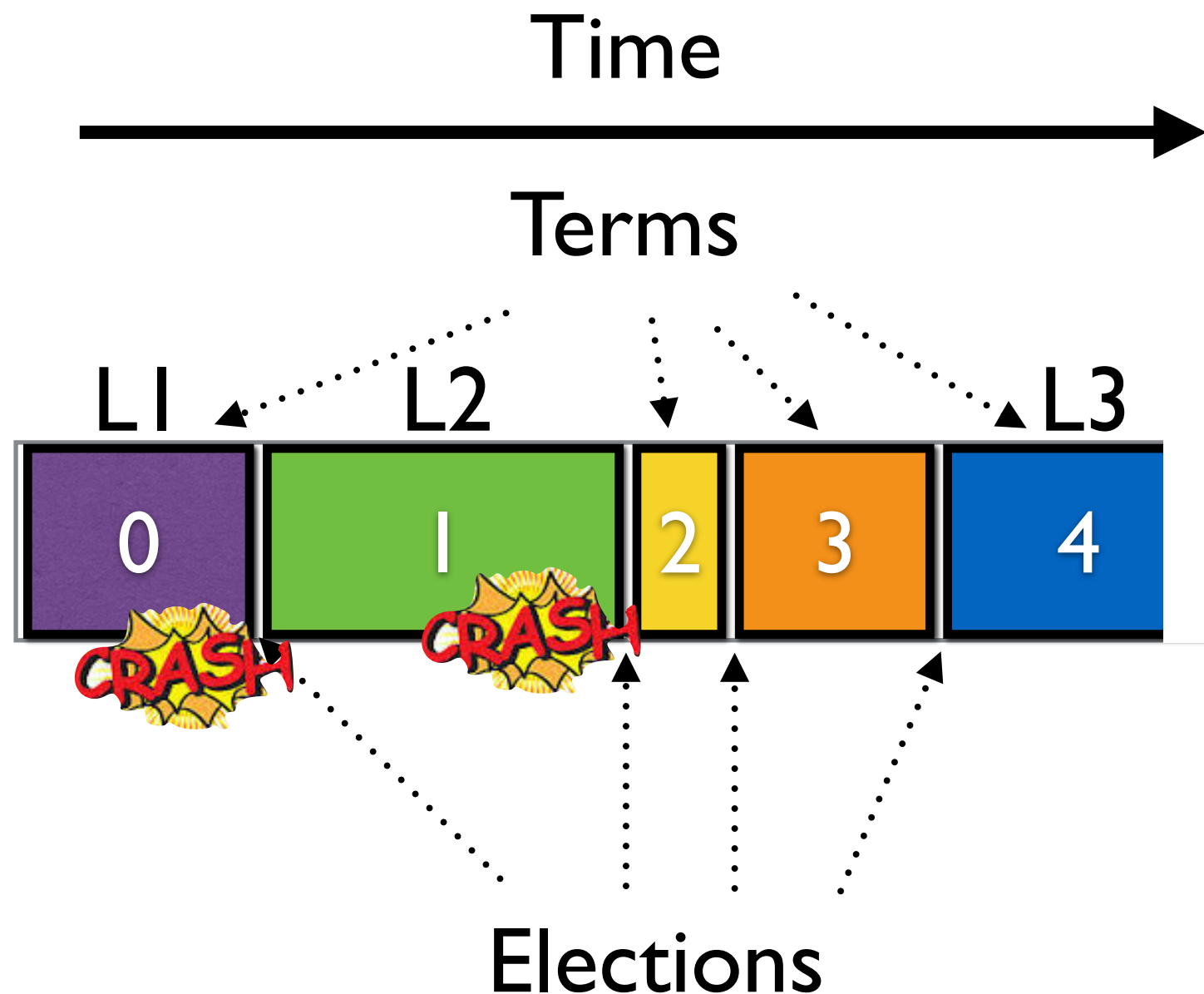


# HeartBeat to Followers



- 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 ONEVOTE 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

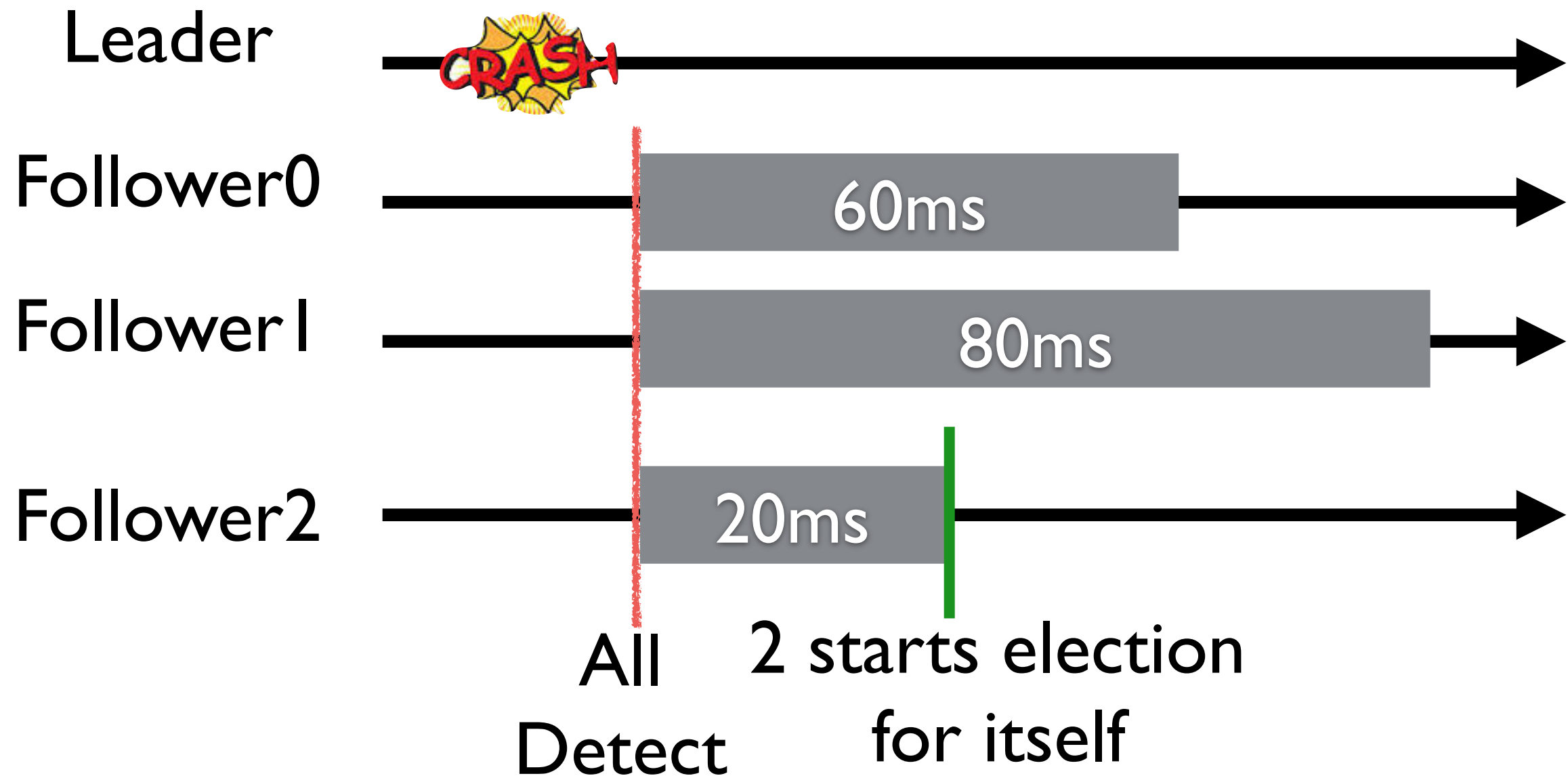
- Nothing is particularly wrong a new term will be initiated and an associated election
- But....
- So how do we ensure there is more than zero leader leaders ever elected

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

# Back to Raft

- What does the newly elected leader do —  
transition from one leader to the next —  
Data Handling



# Data Handling

L SI

F S2

F S3

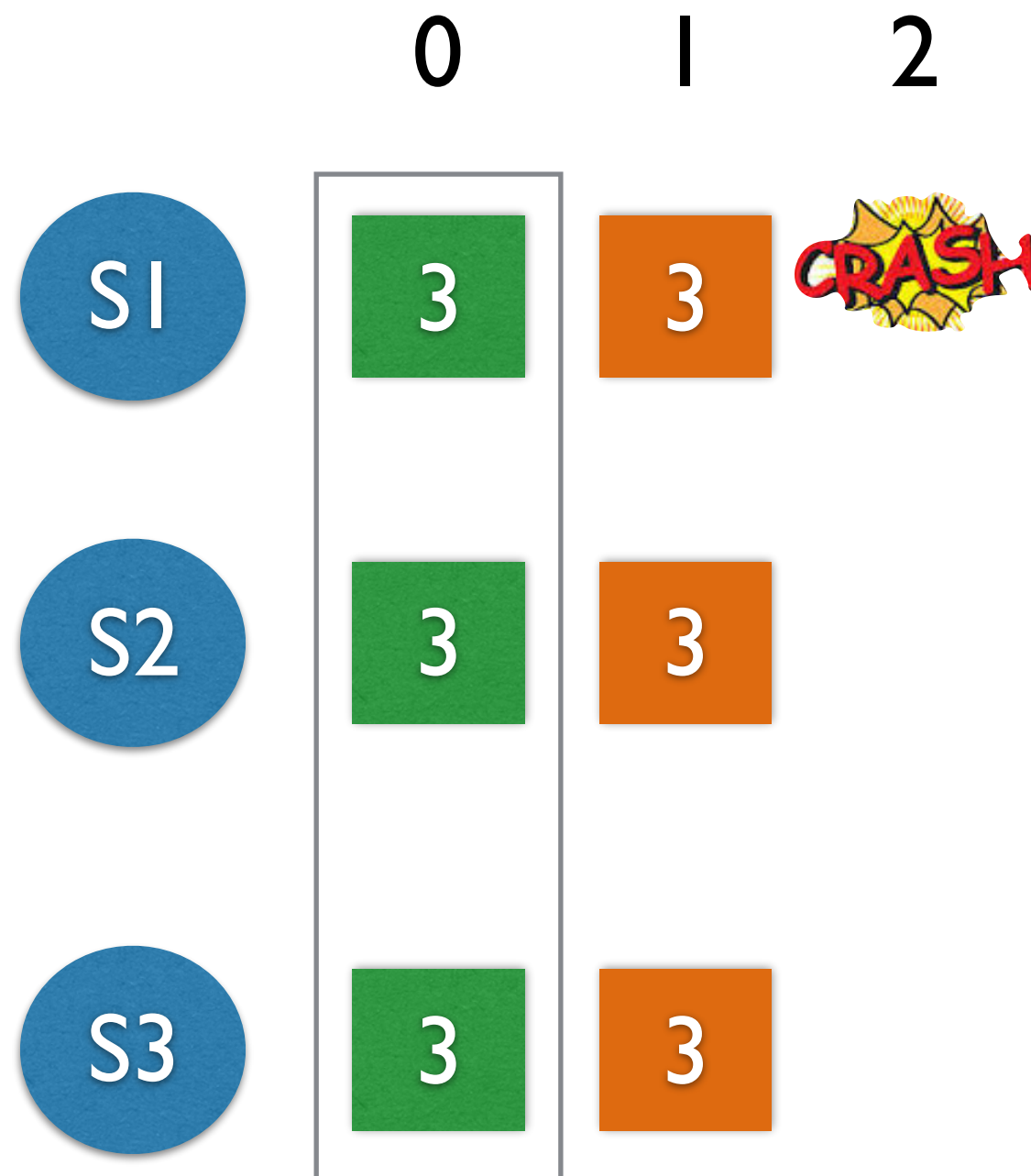
# Data Handling

0 1 2 3



Assume Term is 3 and S1 sends an append  
that is logged by everyone

# Data Handling

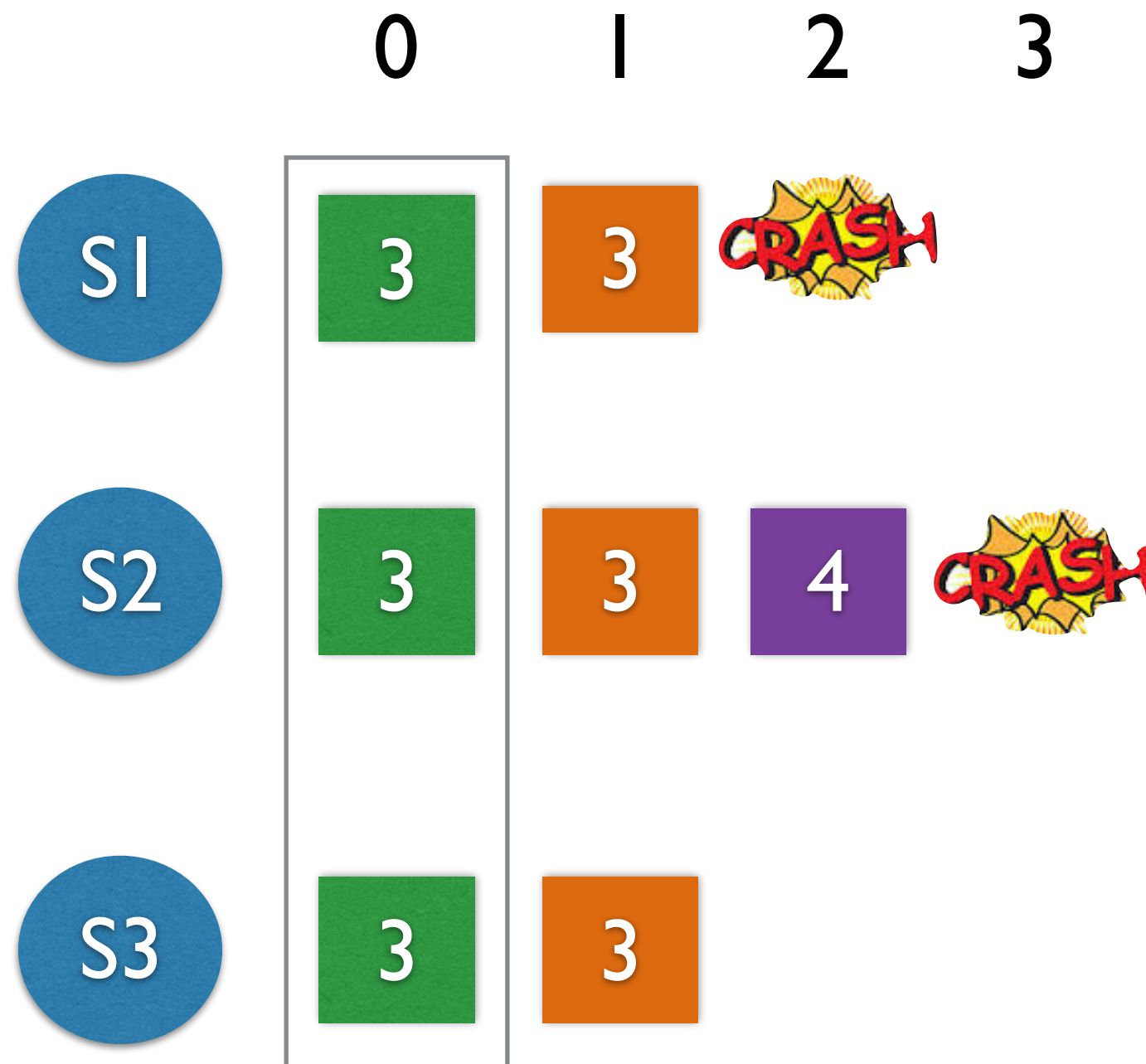


S1 then initiates another append that is received by S2 and S3 but S1 crashes prior to finishing

Any log request that might have been committed on a majority of servers before a leader died must be preserved no matter what scheme we come up with — new leader can not throw it away!

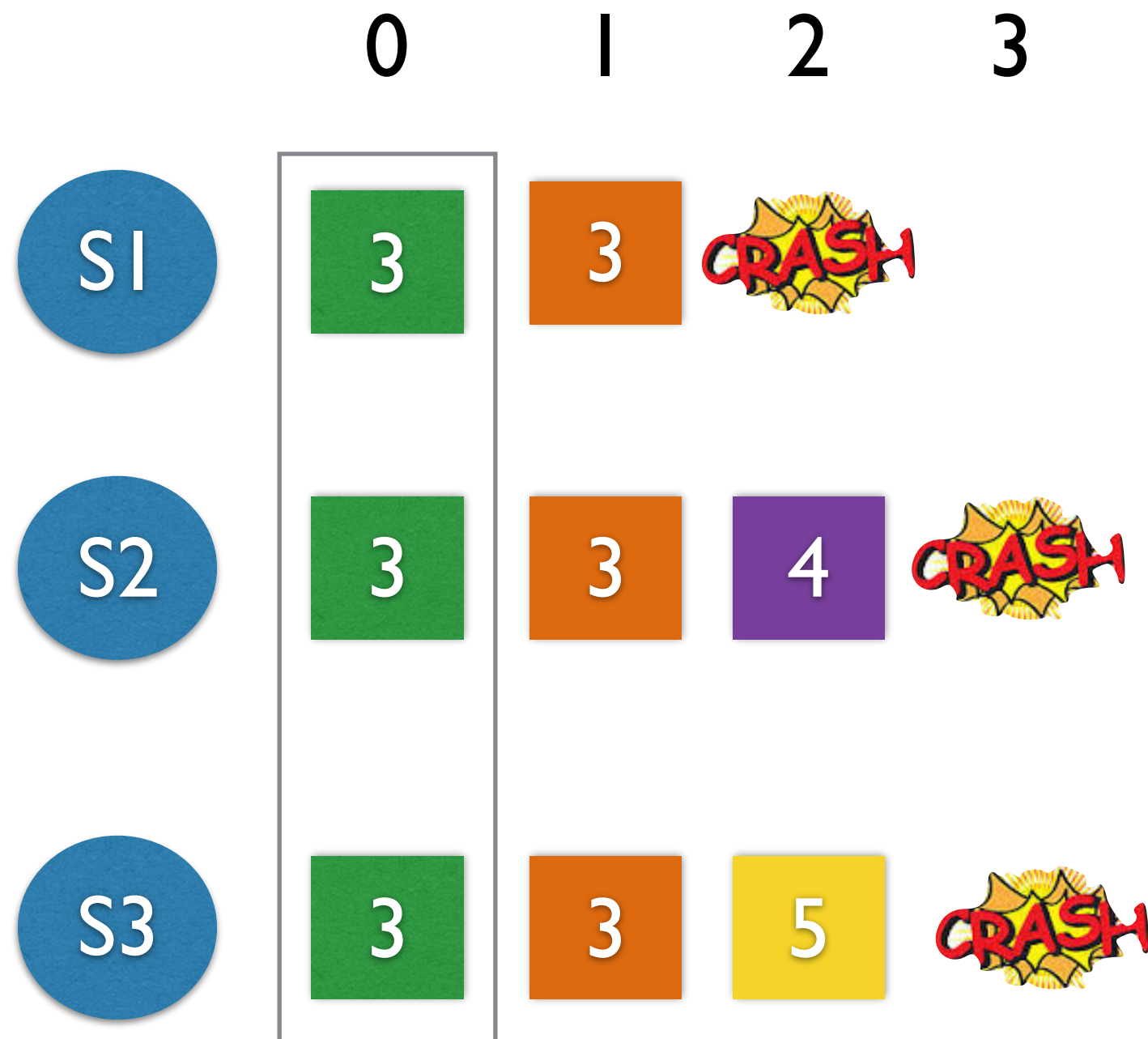
However if not committed they maybe  
overwritten even if on a majority of servers (later  
example)

# Data Handling



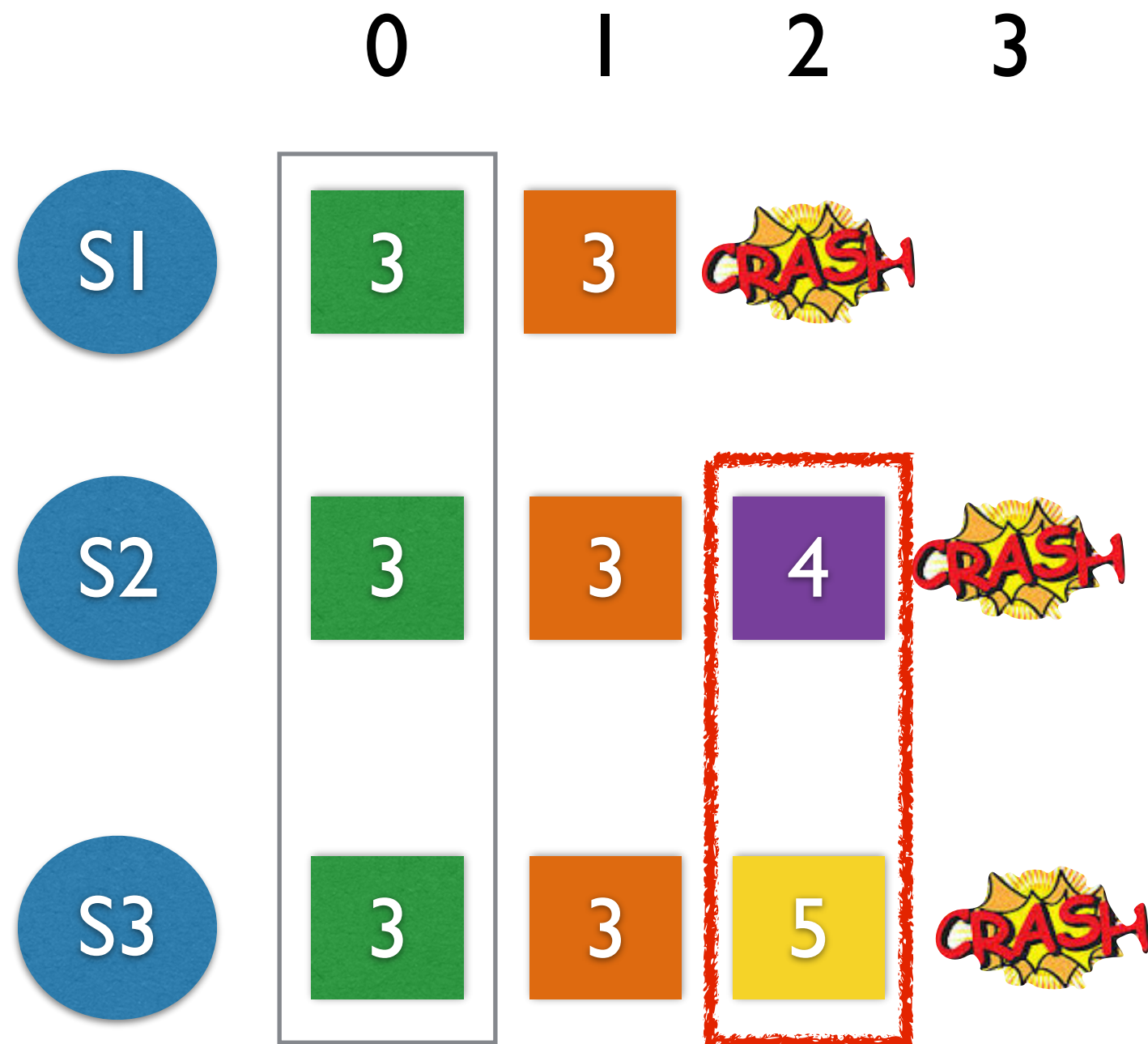
The new leader S2 dies right after updating itself and but before getting messages out

# Data Handling



Now S3 becomes the leader and updates itself and also crashes

# Data Handling



Note this log entry has different values this can't be allowed to persist into next term

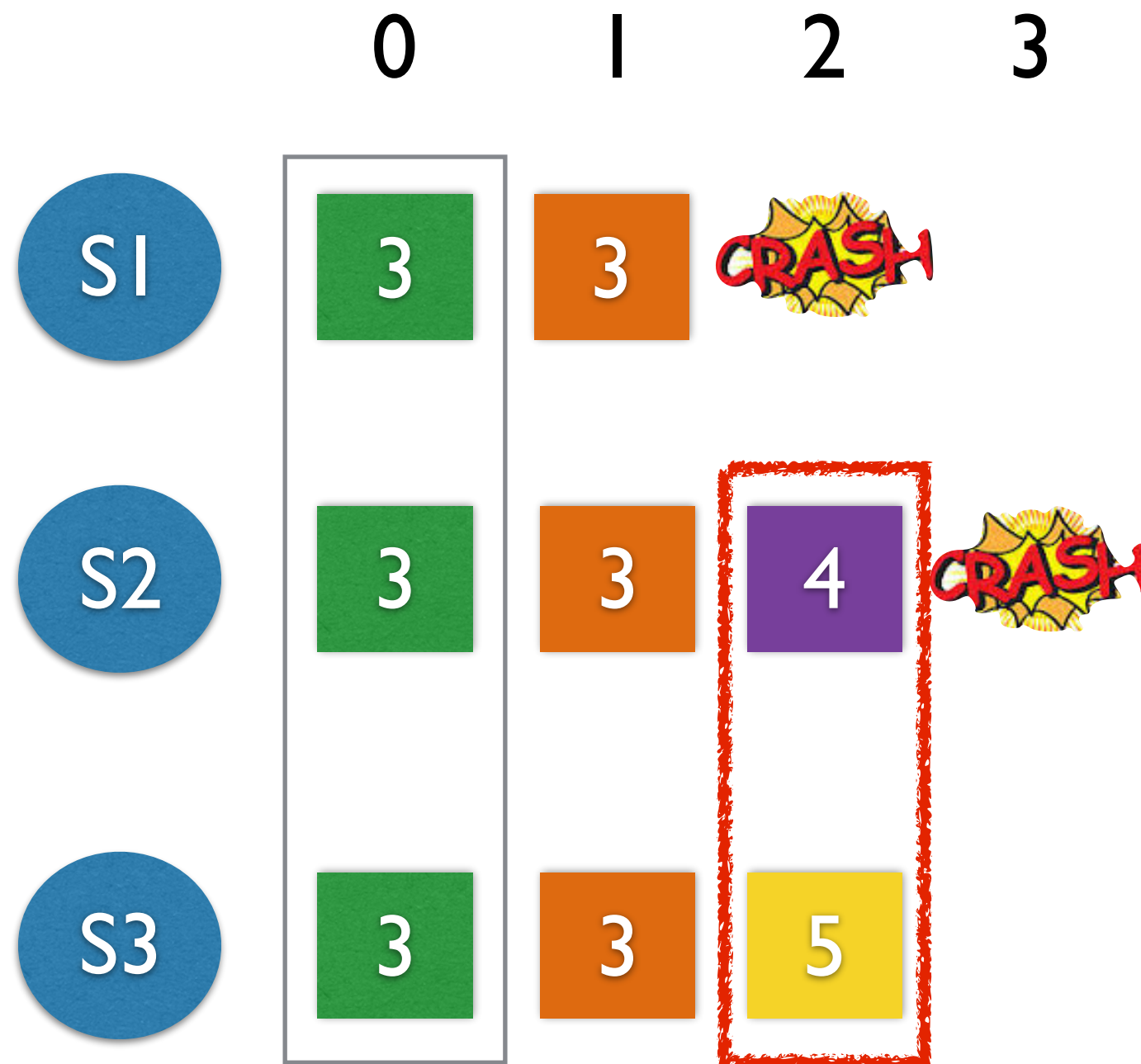


The new leader must take care of this we can't  
let the logs diverge

# Data Handling

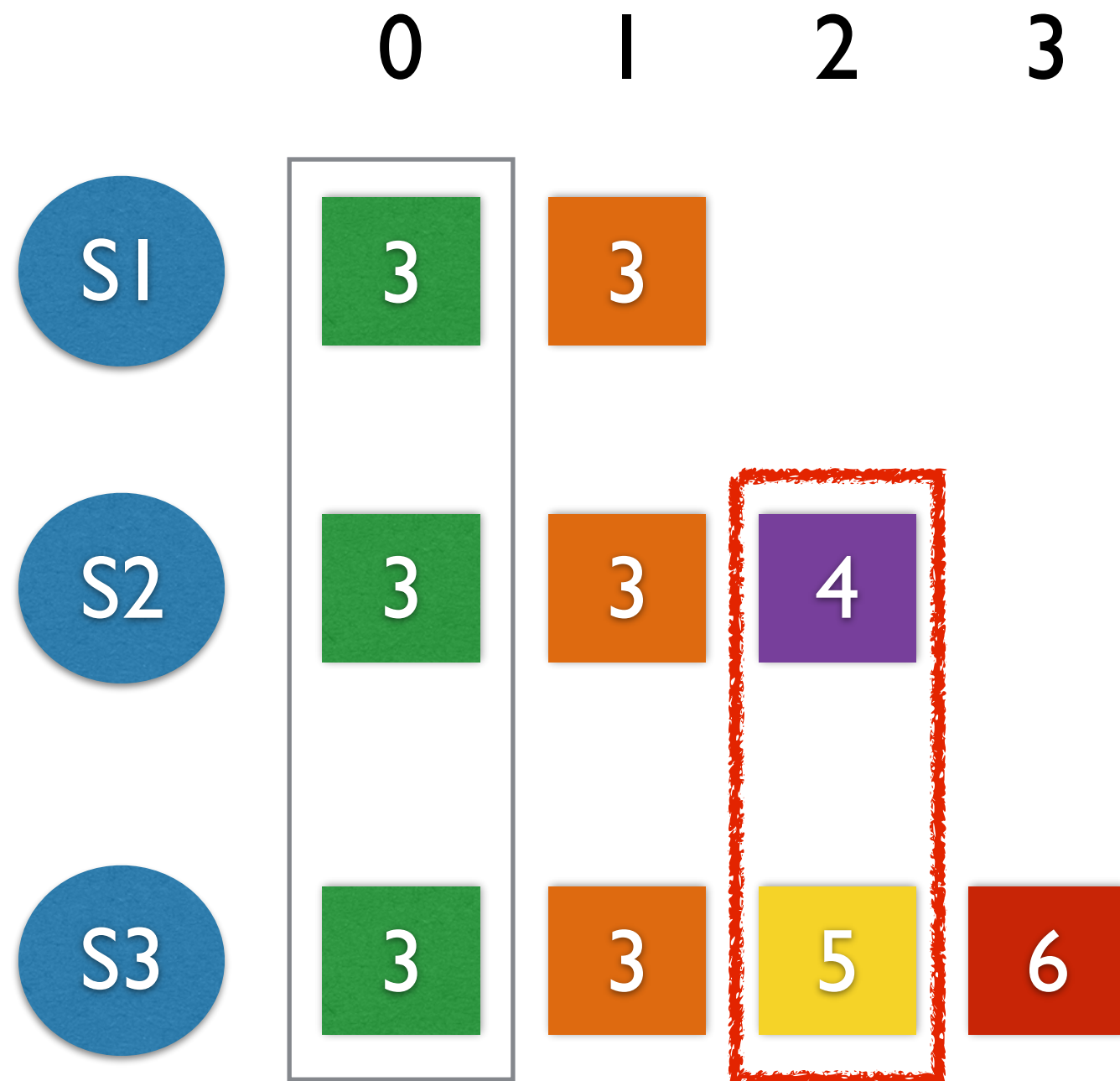
- Ok so how do these scenarios get dealt with
- Remember new leader will force log of the followers to look like its own (they will be rolled back and forward as necessary)

# Data Handling



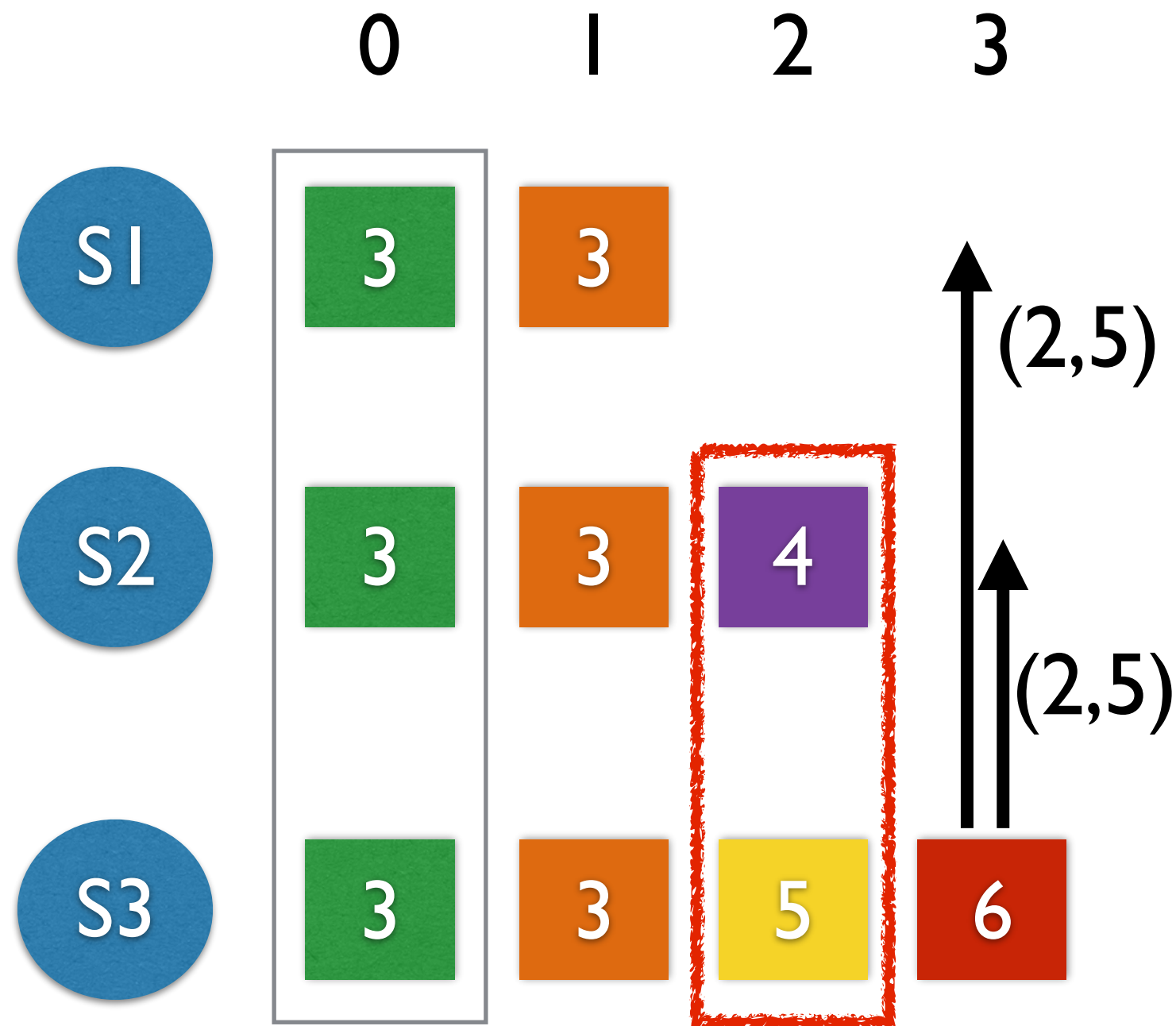
eg. suppose S3 comes back and is chosen as the new leader for term 6

# Data Handling



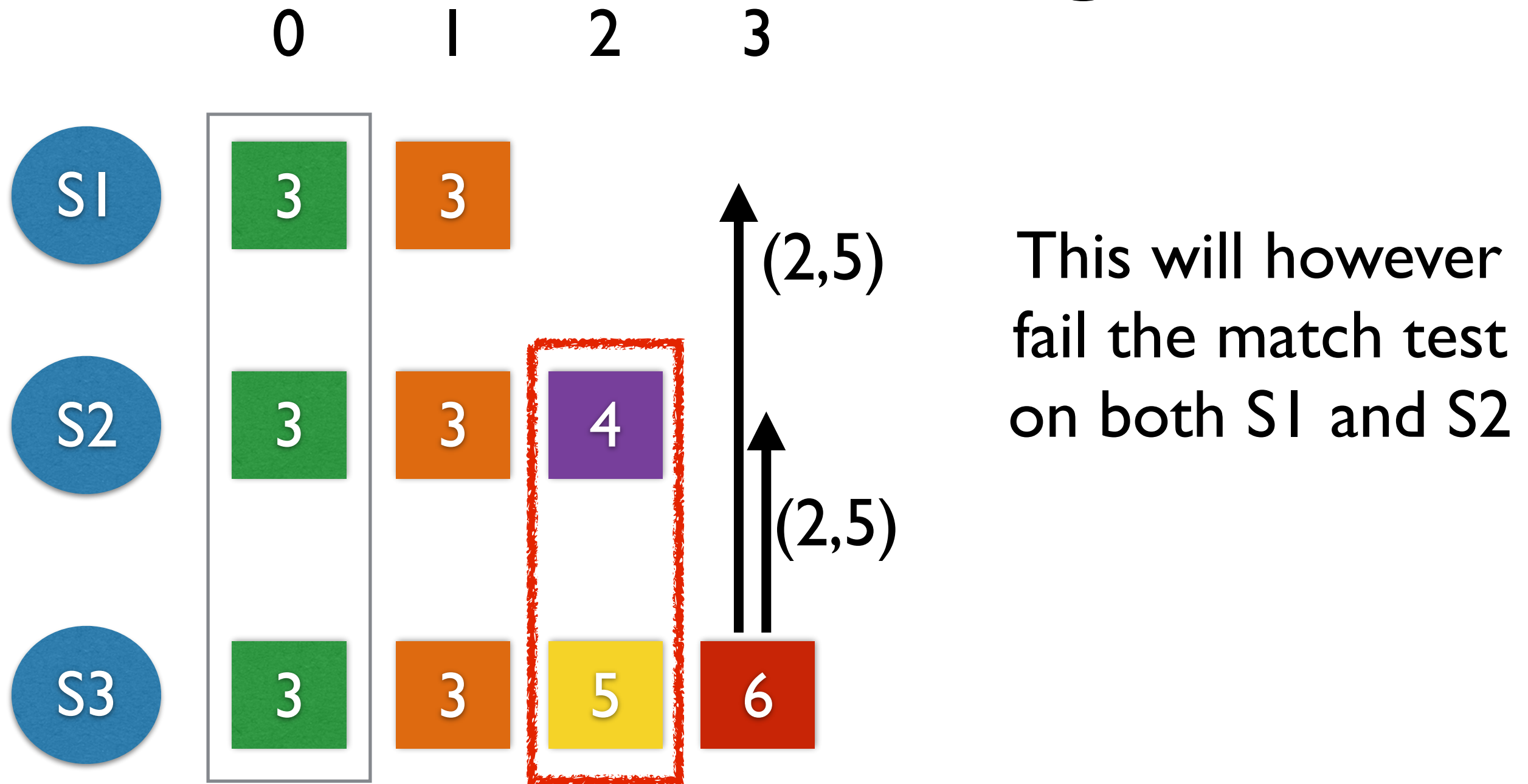
Now on the first append message in term 6 it sends out — we will start to resolve things

# Data Handling



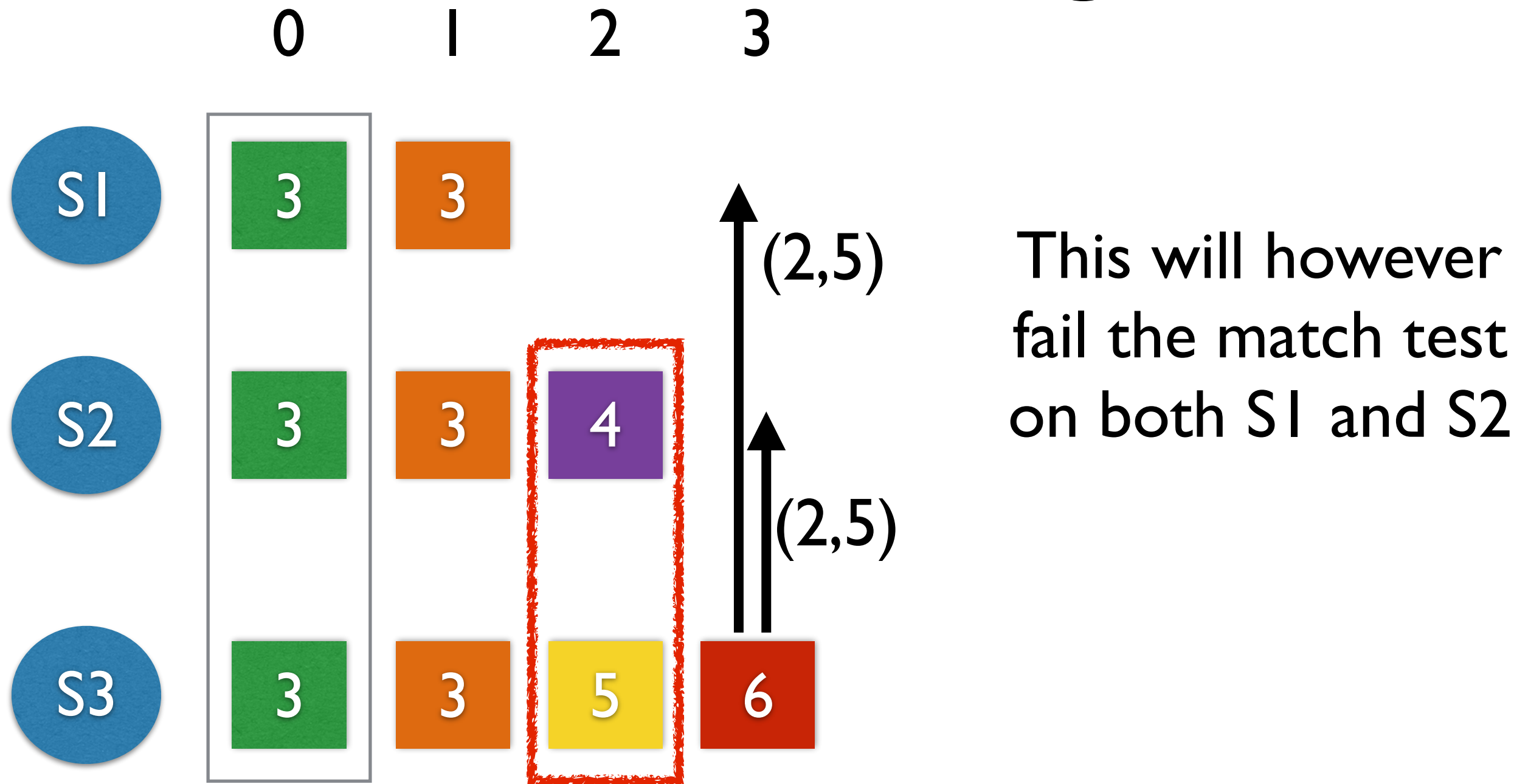
When S3 sends an append for slot 3 in term 6 it will include that its slot 2 is from term 5

# Data Handling


















When S3 sends an append for lot 3 in term 6 it will include that its slot 2 is from term 5

# Data Handling



When S3 sends an append for lot 3 in term 6 it will include that its slot 2 is from term 5

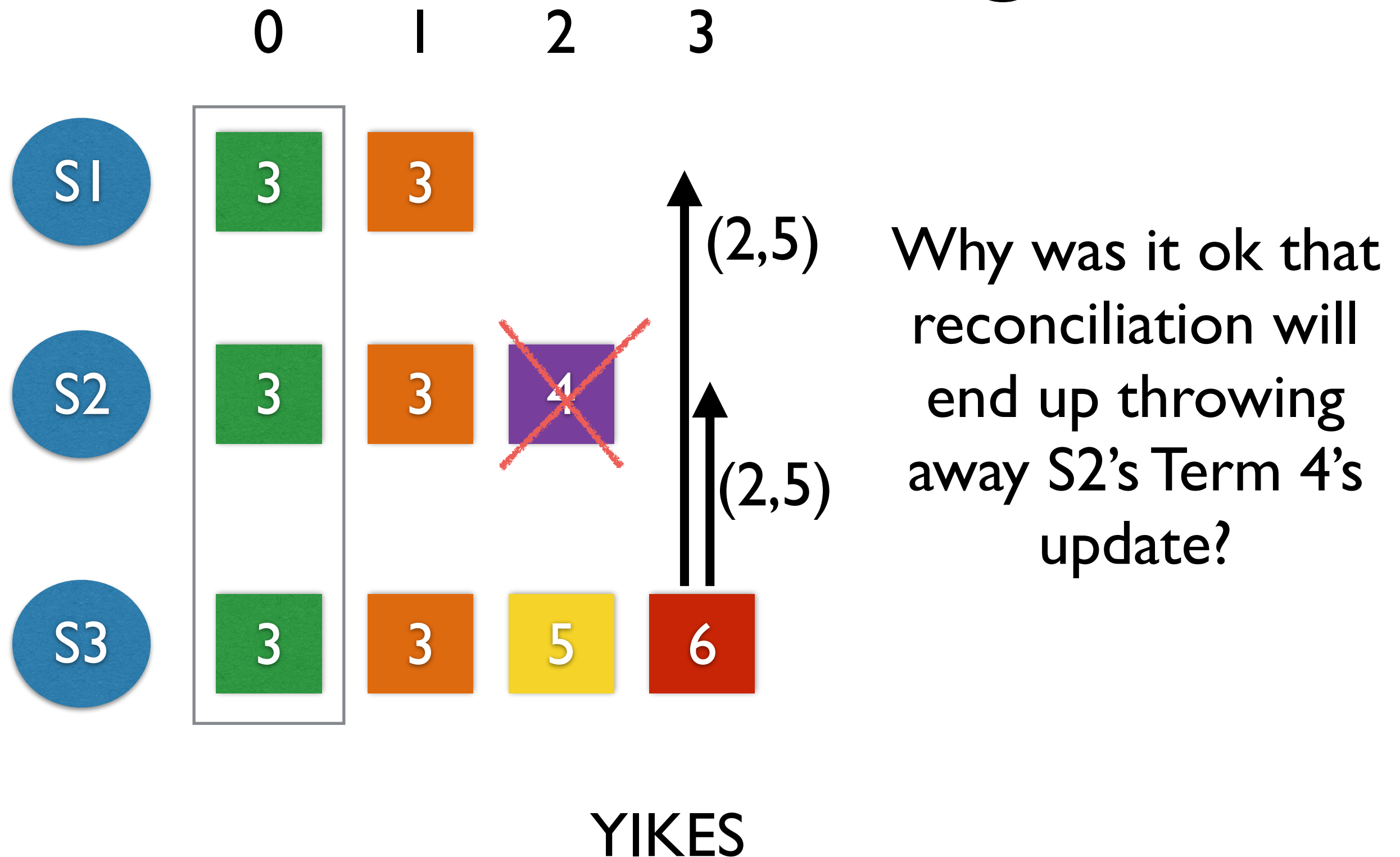
# Data Handling

	0	1	2	3	
					Rolling S1 Forward
					Rolling S2 Backwards and Forwards
					

Commits from prior terms happen after first entry in  
6 commits



# Data Handling



- Remember majority reply was required to be observed before we could have committed it and replied back to the client!
- So this operation has not really happened yet! — Not committed
- The client will eventually time out and resend to what ever the current leader will be at that time

# Another question

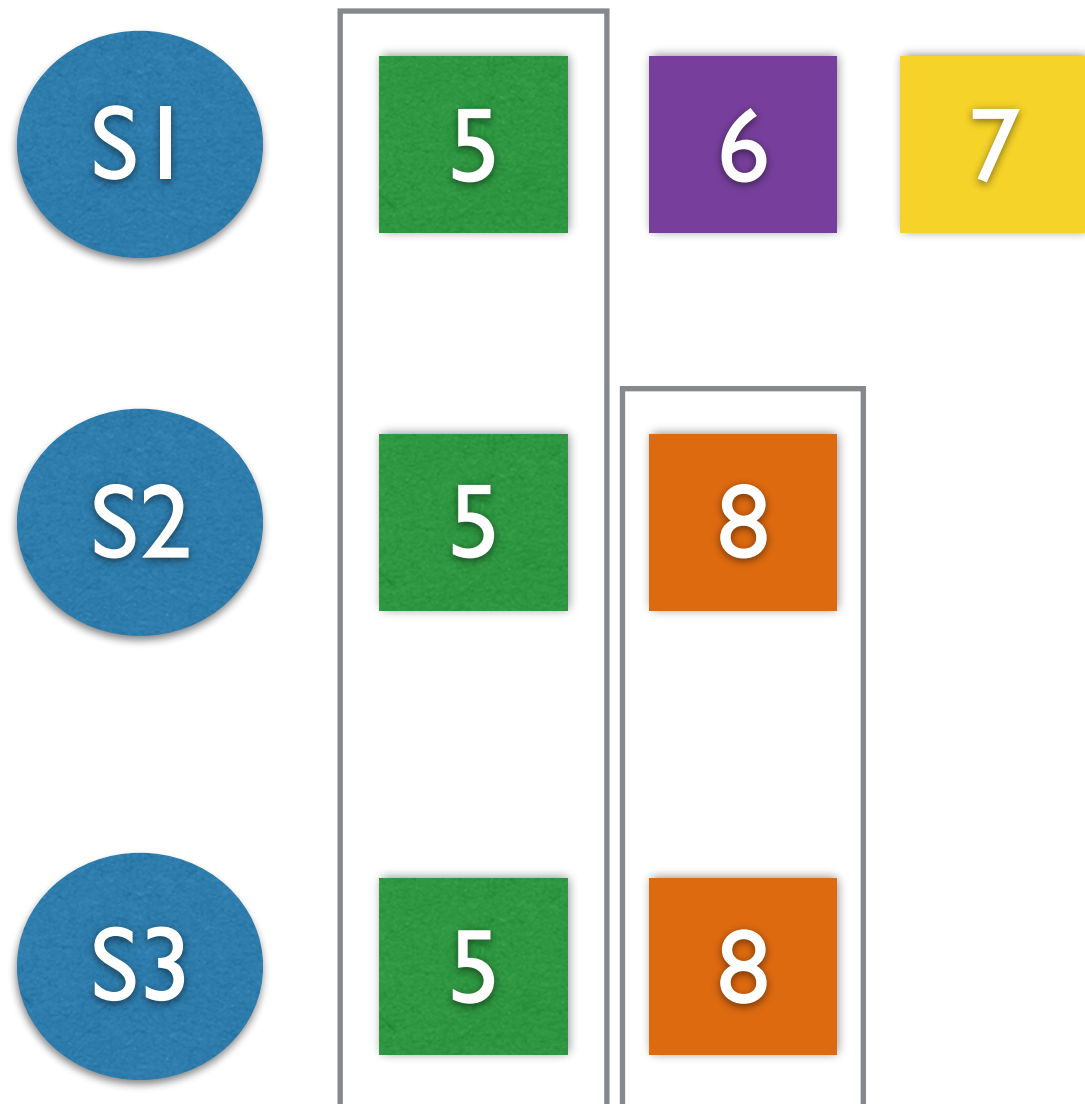
- How can we be sure a new leader will never cause followers to delete entries that made it to a majority but there was a crash before anyone new it?

# Another question

Remember that the new leader got a majority vote and all those followers have confirmed that the new leader's log is at least as up to date as their own — so it must have all entries that made it to the majority in its log!

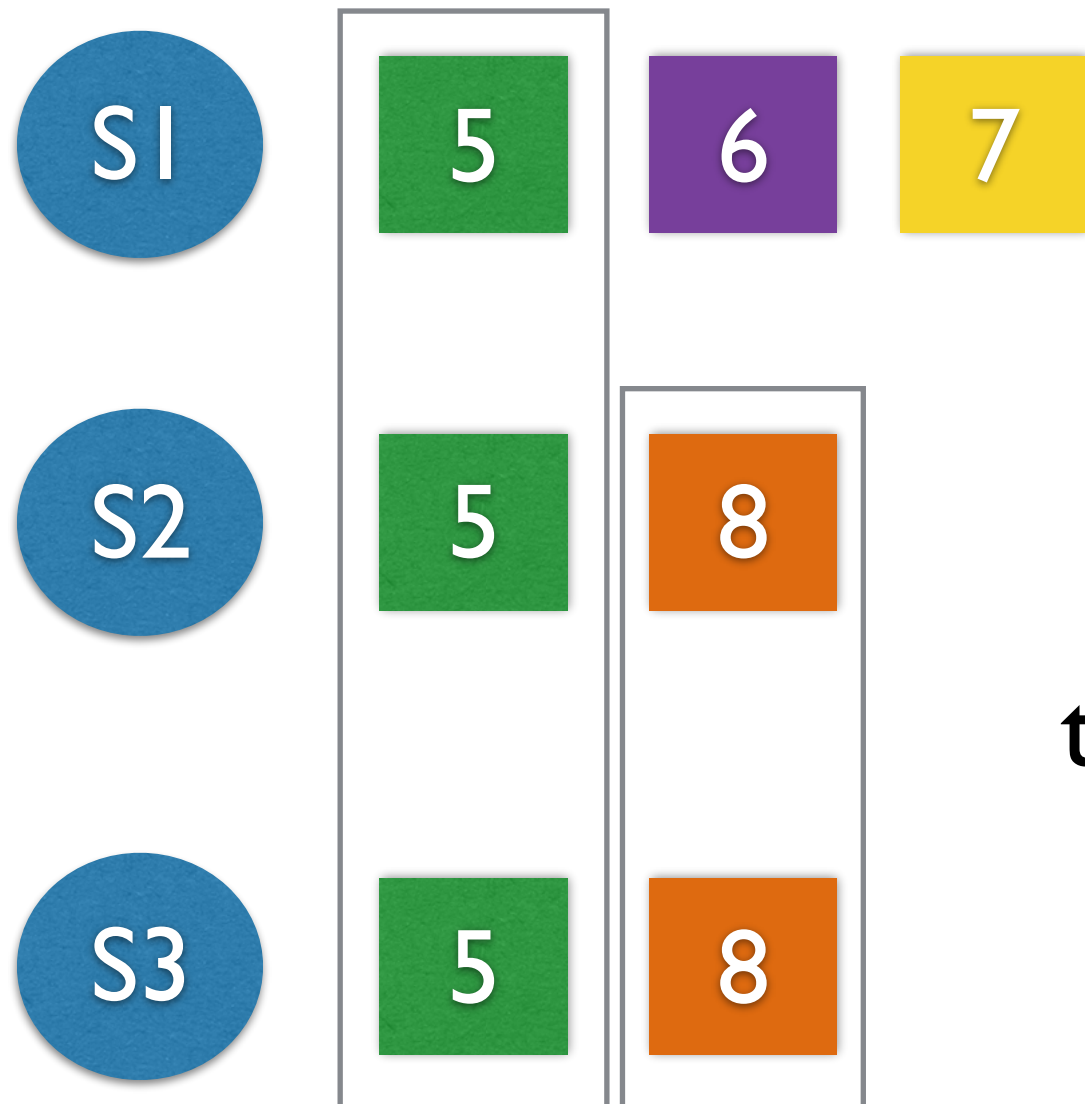
Thus the leader election rules are critical to this:  
Only elect a leader that has every majority log entry in it!

# Example



Let's assume we got to this situation and we now have to elect a new leader

# Example



The election rules would not let S2 and S3 vote for S1 even though it has the longest log — it is not the most “up-to-date”

# KEEP IN MIND

Critical statements:

Figure 2: “and candidate’s log is at least as up-to-date as receiver’s log, grant vote”

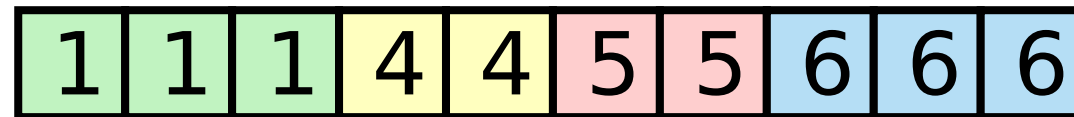
Critical statements:

end of 5.4.1: “Raft determines which of two logs is more up-to-date by comparing the index and term of the last entries in the logs. If the logs have last entries with differing terms the log with the later term is more up-to-date. If the logs end with the same term, then whichever logs is longer is more up-to-date.

# a/d/f Leader?

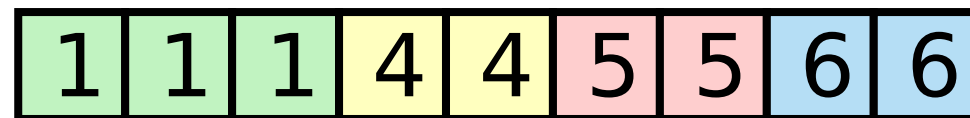
1 2 3 4 5 6 7 8 9 10 11 12

log index

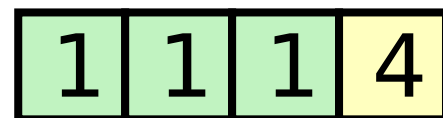


leader for  
term 8

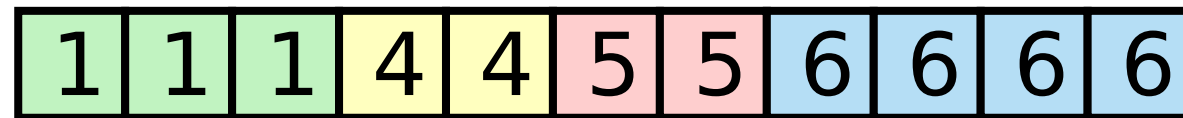
(a)



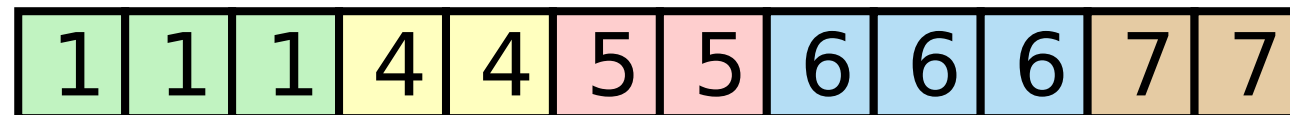
(b)



(c)



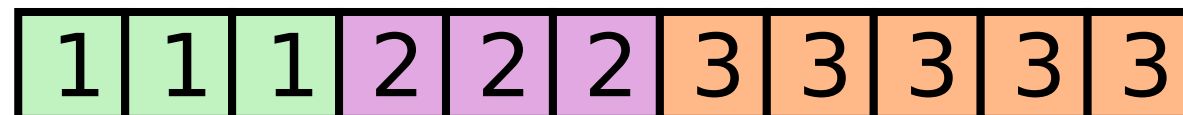
(d)



(e)



(f)

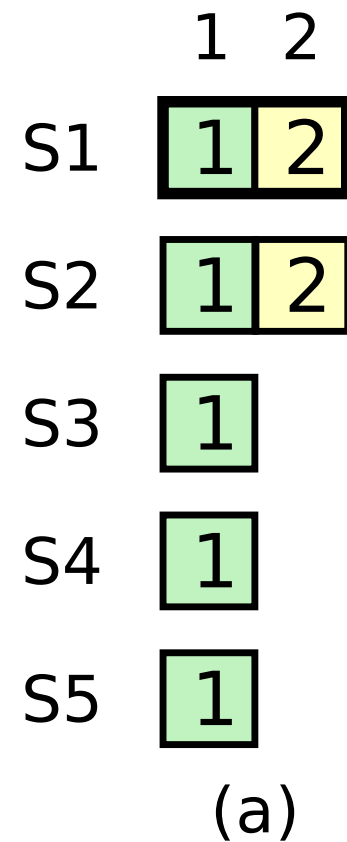


possible  
followers



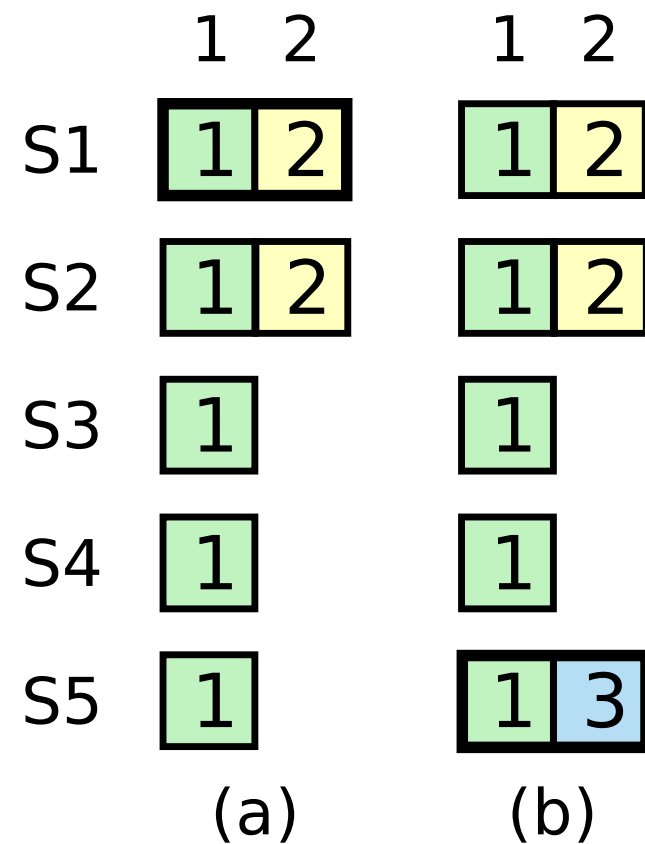
“committed” is not as  
easy as “made it to a  
Majority”

A time sequence showing why a leader cannot determine commitment using log entries from older terms.



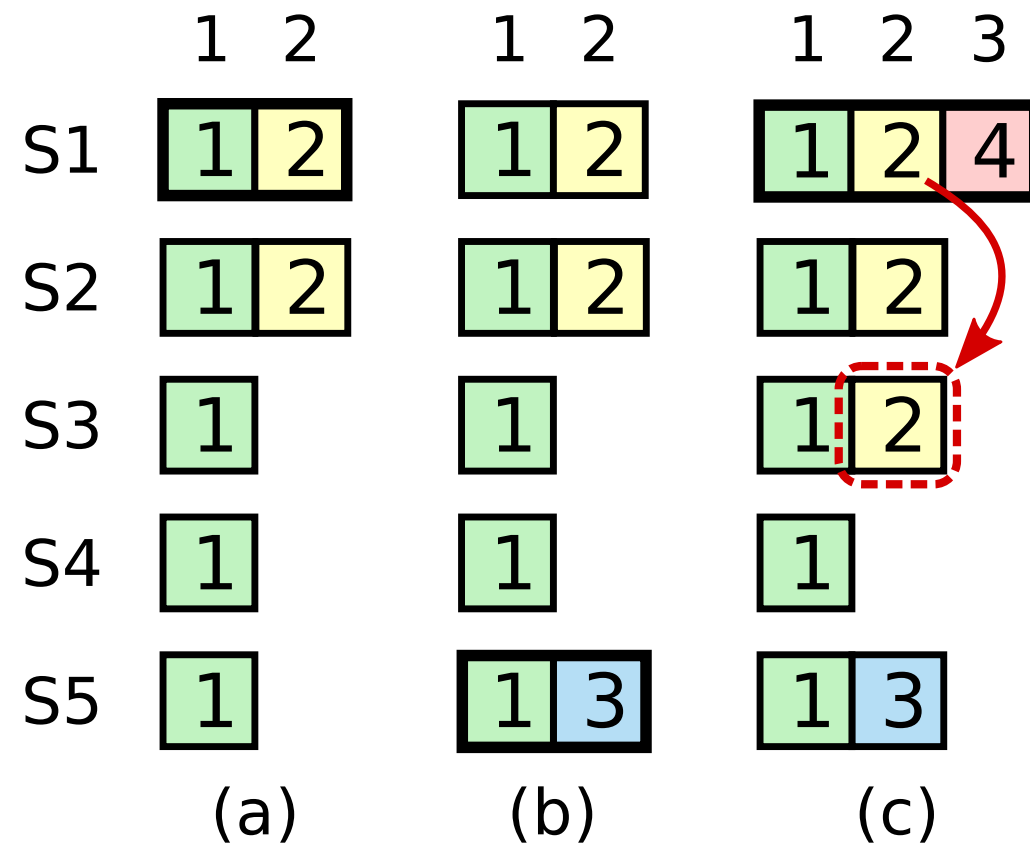
S1 is leader and partially replicates the log entry at index 2.

A time sequence showing why a leader cannot determine commitment using log entries from older terms.



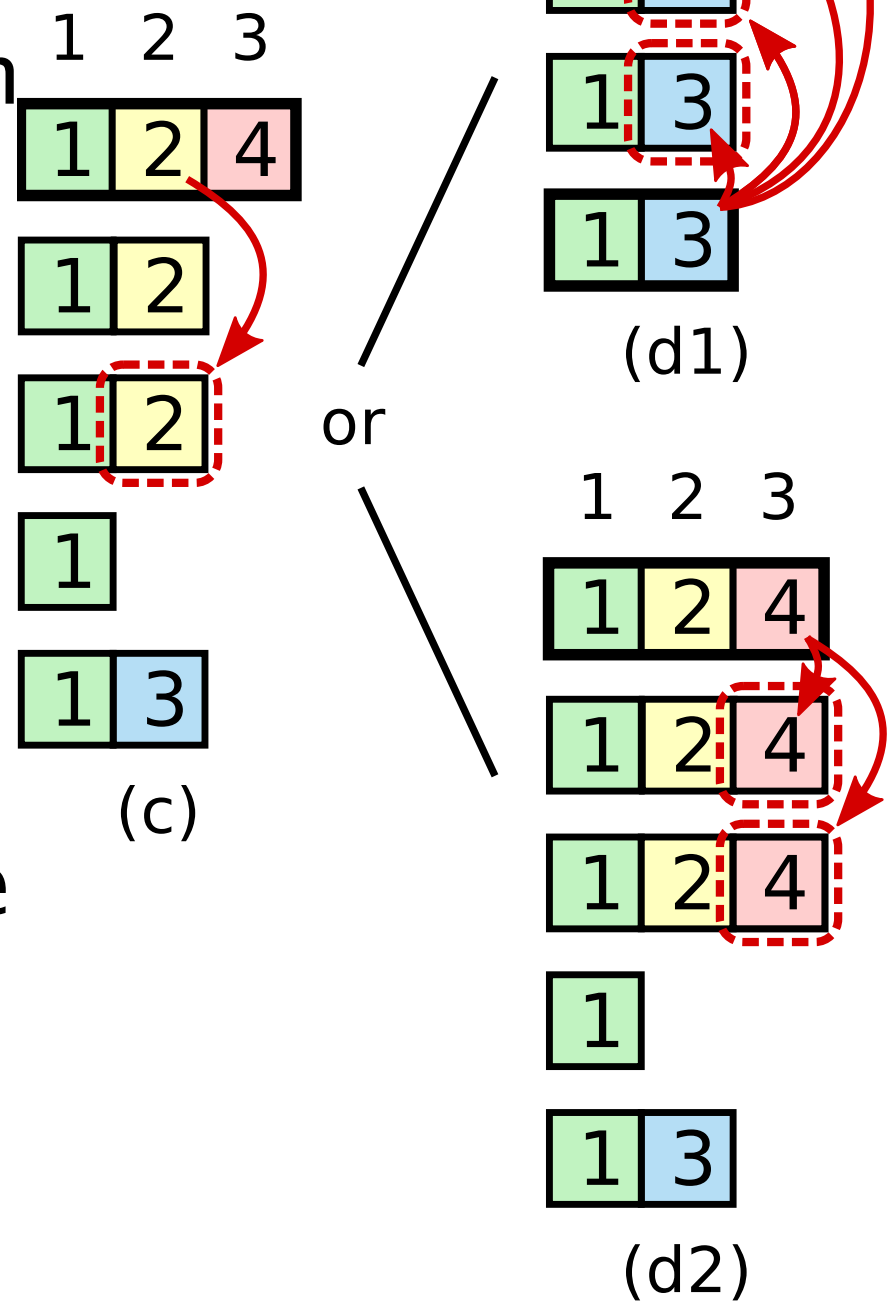
S1 crashes; S5 is elected leader for term 3 with votes from S3, S4, and itself, and accepts a different entry at log index 2.

A time sequence showing why a leader cannot determine commitment using log entries from older terms.



S5 crashes; S1 restarts, is elected leader, and continues replication. At this point, the log entry from term 2 has been replicated on a majority of the servers, but it is not committed.

If S1 crashes as in (d1), S5 could be elected leader (with votes from S2, S3, and S4) and overwrite the entry with its own entry from term 3. However, if S1 replicates an entry from its current term on a majority of the servers before crashing, as in (d2), then this entry is committed (S5 cannot win an election). At this point all preceding entries in the log are committed as well.



committed means that the  
entry:

1) reached a majority in  
the term it was initially  
sent out, or

2) a subsequent log entry  
is committed

# Assignment Note

- Quick Rollback: End of section 5.3
- Given aggressive nature of failure testing you will likely need this:
- “When rejecting an AppendEntries request follower can include the term of the conflicting entry and the first index it stores for that term. ... the leader can decrement nextIndex to bypass all of the conflicting, rather than one RPC per entry”