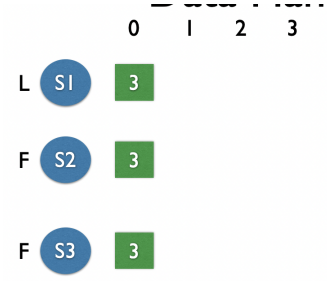
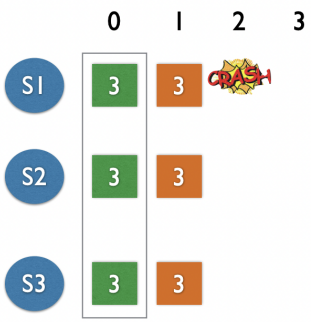
CAS CS 350
Lec 13

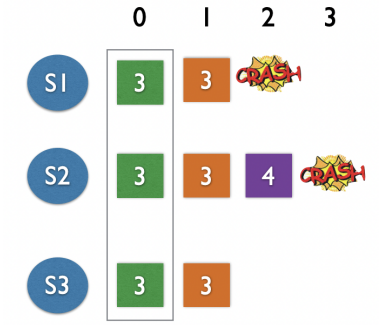
Raft

1. Data Handling
    a. What does newly elected leader do - transition from one leader to the next - Data Handling
    b. 

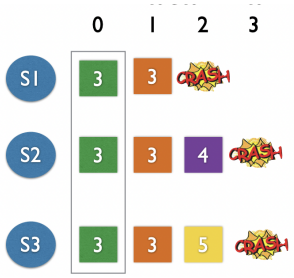
    c. S1 is the leader
    d. Assume term is 3 and S1 sends an append that is logged by everyone
    e. Term increases everytime an election happens
        i. Term
            1. servers use the term numbers to check whether they are up to date
            2. sequence number that increases (if there is a failure, this number increases
            3. Check whether an entry is consistent or not
            4. Term represents phase in the raft code (new phase begins when there is a failure) → term keeps track of these phases
    f. 

    g. S1 then initiates another append that is received by S2 and S3 but S1 crashes prior to finishing → new election happens
    h. 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 → because we might have replied to the client (it must be included in the log)

i. However if not committed they maybe overwritten even if on a majority of servers

j. 

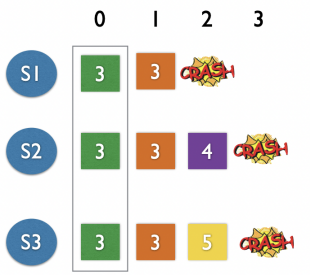
k. The new leader S2 dies right after updating itself and before getting messages out to the followers → elect new leader

l. 

m. S3 becomes the leader and updates itself and also crashes (S3 receives client request in term 5 and appends it to its log and fails immediately before sending request to followers)

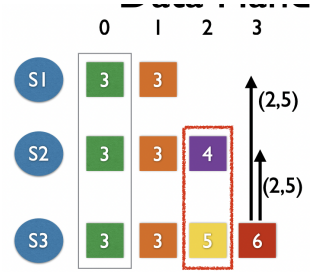
n. It doesn't guarantee the amount of time that the servers are unavailable

o. Note this log entry has different values this can't be allowed to persist into next term → new leader forces the servers to have the same identical log (they will be rolled back and forward as necessary)

p. 

q. S3 comes back to the leader and is chosen as the new leader for term 6

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

s. 

t. When S3 sends an append for slot 3 in term 6 it will include that its slot 2 is from term 5 → log stays inconsistent until there is a client request

u. This will however fail the match test on both S1 and S2 (it needs to check whether the previous entry matches) → in order to match, the previous entry must match with the log of leader

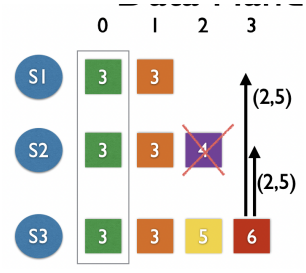
v. Leader receives replies from follower S1 and S2 → leader will backtrack and go to slot2 and send append request on term5 (rolling back) to check whether the previous entries match (and continues to do so until they find the identical entry match)

w. 

x. Rolling S1 Forward and Rolling S2 Backwards and forwards

y. Commits from prior terms happen after first entry in 6 commits

z. At some point, the protocol makes sure that logs of servers are identical and matches

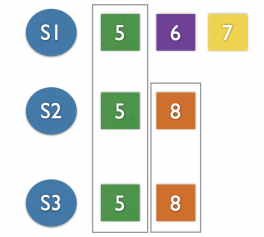
aa. 

bb. Why was it ok that reconciliation will end up throwing away S2's Term 4's update?

   i. it would not have been committed (it appears in the minority)

cc. 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 knew it?

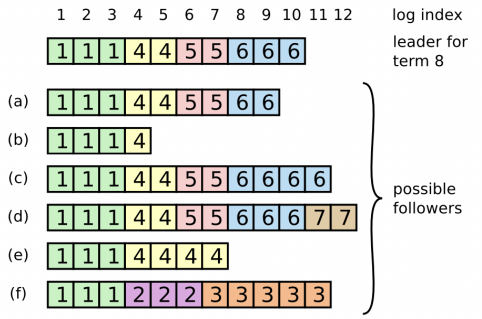
i. Remember that the new leader got a majority vote and all those followers have confirmed that the new leader's log is at least as up to date as their own – so it must have all entries that made it to the majority in its log
ii. Leaders that got elected must be up to date and needs to have all committed entries to become the leader
iii. The leader election rules are critical to this: Only elect a leader that has every majority log entry in it

dd. 

ee. Let's assume we got to this situation and we now have to elect a new leader: S2 and S3 can be the leader

ff. 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" (check the term and if they are same, go to index → check only two numbers)
i. log needs to have the latest term
ii. Candidate's log is at least as up-to-date as receiver's log, grant vote
iii. 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 log end with the same term, then whichever logs is longer is more up-to-date

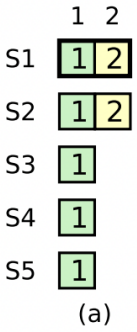
2. A/D/F leader?

a. 

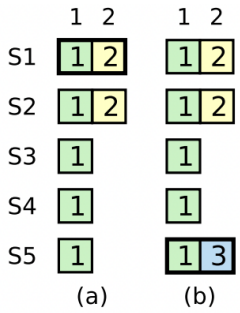
b. Can server a, d, or f be leader?
i. Server A can become leader → a, b, e, f can vote
ii. Server D can become leader → a, b, c, d, e, f can vote
iii. Server F cannot become leader → only F would vote for it

c. When A becomes the leader, it will force other servers to have the same log as itself once it receives client request

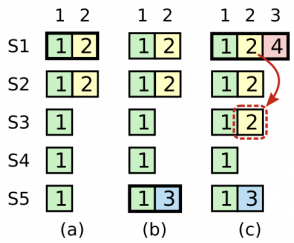
d. “Committed” is not as easy as “made it to a Majority”
e. A time sequence showing why a leader cannot determine commitment using log entries from older terms



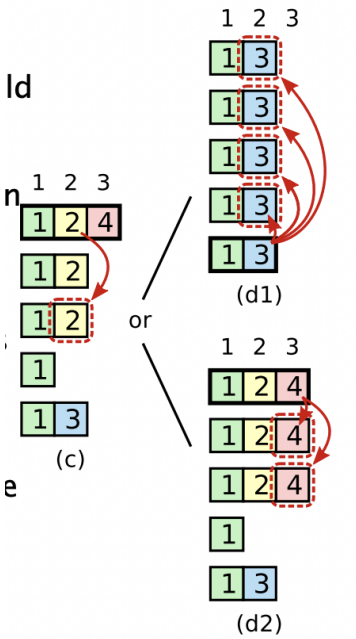
f.
g. S1 is a leader and partially replicates the log entry at index 2
h. S2 receives the entry and appends, but the other servers do not receive it due to failures



i.
j. S1 crashes; S5 becomes the new leader for term 3 with votes from S3, S4, and itself, and accepts a different entry at log index 2 (S1 and S2 cannot vote for S5 because their log is more up-to-date)



k.
l. S5 crashes; S1 restarts, is elected leader, and continues replication. At this point, the log entry from 2 has been replicated on a majority of the servers. Can it be committed?
    i. No
    ii. If there exists an N such that N > commitIndex, a majority of matchIndex[i] >= N, and log[N].term == currentTerm: set commitIndex = N(5.3, 5.4)



m.

n. 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

o. Committed means that the entry:
    i. Reached a majority in the term it was initially sent out
    ii. A subsequent log entry is committed