READ MODE · ANSWER FIRST

How Raft Handles Leader Election, Log Replication, and Failover

An interactive, visual explanation of How Raft Handles Leader Election, Log Replication, and Failover, from setup through the complete system.

CHEAT SHEET · 5 ESSENTIAL IDEAS

The whole story in 5 lines

How Raft Handles Leader Election, Log Replication, and Failover becomes easier to reason about when every stage is connected as one system.

1. Teach the baseline Raft cluster before failure. The key visual is simultaneous heartbeats resetting follower timers across a...
2. Show randomized follower timers diverging until one node crosses the threshold first and starts an election.
3. Teach self-vote, log freshness checks, grant or deny replies, and the split-vote branch.
4. Show the leader log, follower logs, AppendEntries fan-out, and majority-based commit without waiting for every replica.
5. Show crash or partition, replacement election, higher-term step-down, and log suffix repair on rejoin.

STAGE 1 OF 7

Setup

Welcome to Raft Consensus. Raft is a protocol that lets a group of computers agree on shared data, even when some of them crash. Think of it like a classroom where students vote on answers and the majority wins, so one wrong guess does not ruin the result.

Consensus means all the nodes in a cluster agreeing on the same ordered list of commands. A leader is the single node that accepts new writes and tells everyone else what to store.

Followers are the other nodes. They do not accept writes directly. Instead they copy whatever the leader sends them, like students copying notes from the board.

A term is a numbered era. Every time a new election starts, the term number goes up by one. It works like a school year number so everyone knows which leader belongs to which era.

The log is the ordered list of commands each node keeps locally. A quorum is a strict majority, for example three out of five nodes. Raft never commits data unless a quorum agrees.

Over the next five stages we will walk through the full lifecycle: how the cluster stays calm, how silence triggers an election, how votes decide a leader, how data gets copied safely, and how a crashed leader gets replaced. Let us start with the peaceful baseline.

STAGE 2 OF 7

Stable Term

02 STABLE TERM

In Setup we learned that Raft picks one leader and copies data to followers. Here you can see that leader already in charge of term 12, with every follower waiting calmly.

The leader sends a heartbeat AppendEntries message to all followers. Think of it like a teacher calling "still here!" so the class knows the lesson is still going.

Each follower receives the heartbeat and resets its own election timer back to zero. Try switching the Cadence control to Slow and watch the timers climb higher before the next heartbeat arrives.

Between heartbeats, every follower counts time on its own local clock. The timers tick upward independently because no two machines share the same clock.

Another heartbeat arrives just in time, pushing all timers back down. Use the Nodes stepper to add or remove followers and notice that the leader still sends to everyone in the cluster.

The takeaway: as long as heartbeats keep winning the race against election timers, the cluster stays stable and no election is needed. But what happens when heartbeats stop? That is exactly what we explore in the next stage, Timeout Trigger.

STAGE 3 OF 7

Timeout Trigger

03 TIMEOUT TRIGGER

We just saw how heartbeats keep the cluster calm. Now the leader has gone silent. Without those heartbeats, every follower just keeps counting time on its own clock.

Each follower picked a random election timeout at startup. Because the timeouts differ, the timers drift apart instead of all expiring at the same instant. Toggle the Skew control to Tight and watch how close calls almost cause a pile up.

One follower crosses its timeout first. That node bumps the term number from 12 to 13, because a new election always means a new term.

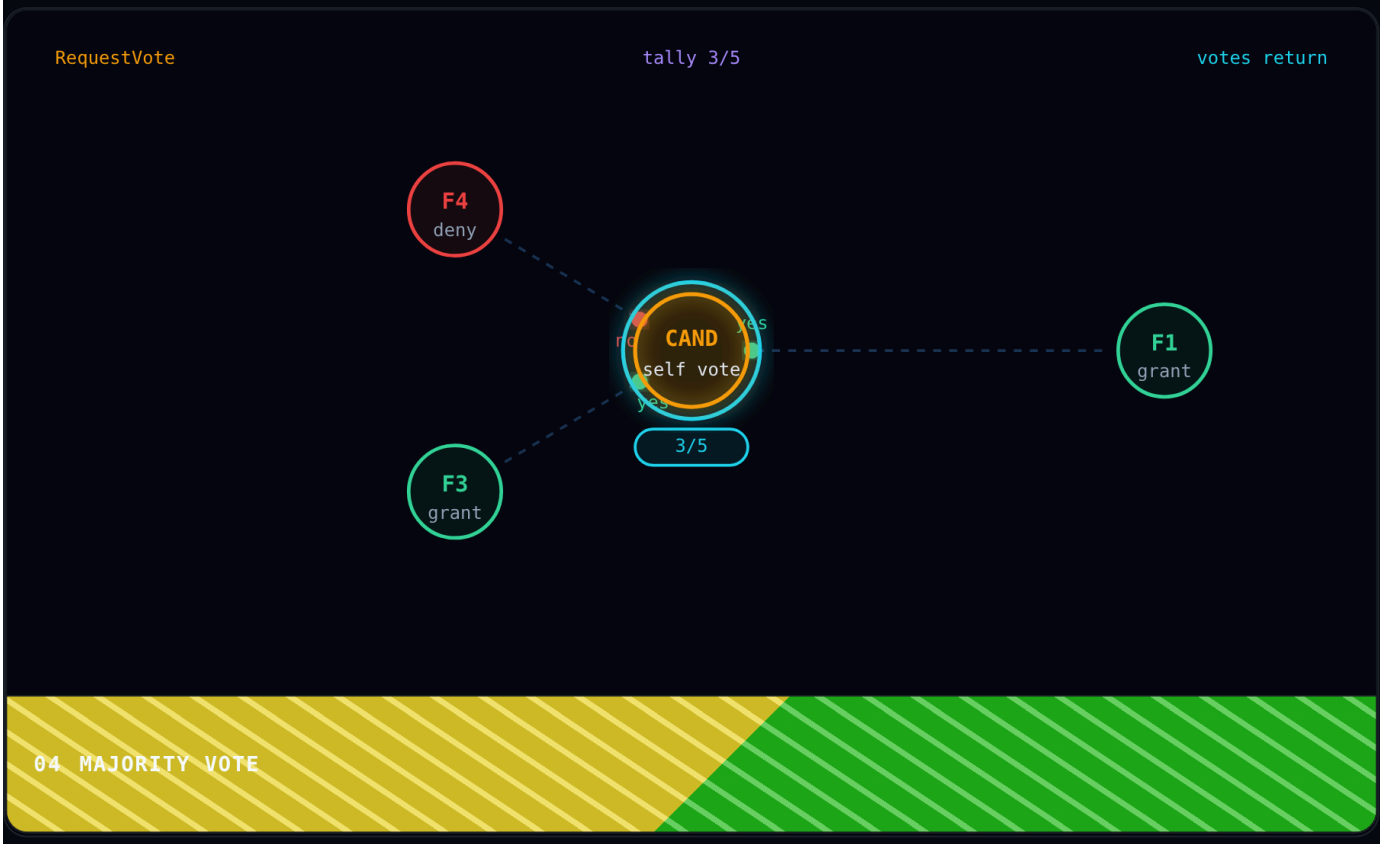
The moment it crosses the threshold, that follower becomes a candidate and records its own self vote. You cannot ask others to vote for you if you would not vote for yourself.

The candidate sends RequestVote messages to every other node in the cluster. This is where the election actually starts.

Raft has now turned silence into a single election round instead of chaos. The randomized timers are the key trick. Next, in Majority Vote, we will see whether the cluster actually grants enough votes to pick a winner.

STAGE 4 OF 7

Majority Vote



In the last stage a candidate emerged from the timeout. Now it needs actual votes. The candidate begins term 13 with exactly one vote, its own self vote from the moment it became a candidate.

RequestVote messages fan out to every reachable peer. Think of it like a student running for class president and asking each classmate for their vote.

Each peer checks two things before answering: is this term at least as high as mine, and is the candidate log at least as fresh as mine? Only if both pass does the peer grant its vote.

Vote replies flow back and the tally moves toward quorum. Remember, quorum means a strict majority, so in a five node cluster that is three votes. Switch the Outcome control to Split vote to see what happens when the tally falls short.

If the candidate collects a majority, it becomes the new leader right away. If votes split evenly, nobody wins and the term ends without a leader. Use the Nodes stepper to see how cluster size changes the quorum threshold.

When a split vote happens, randomized timeouts break the tie by letting a different follower try next. Raft never allows two leaders in the same term. Now that we have a leader, the next stage, Replicate and Commit, shows how that leader actually copies data safely.

STAGE 5 OF 7

Replicate + Commit

We now have a leader elected by majority vote. The first real job of that leader is accepting writes from clients. Here a client sends a new command to the leader.

The leader appends that command to its own log immediately. But the entry is still uncommitted because no follower has confirmed it yet. Use the Entries stepper to send more than one command at a time.

The leader fans out AppendEntries messages carrying the new log slot to every follower. This is the same AppendEntries message type we saw as heartbeats in Stable Term, but now it carries real data.

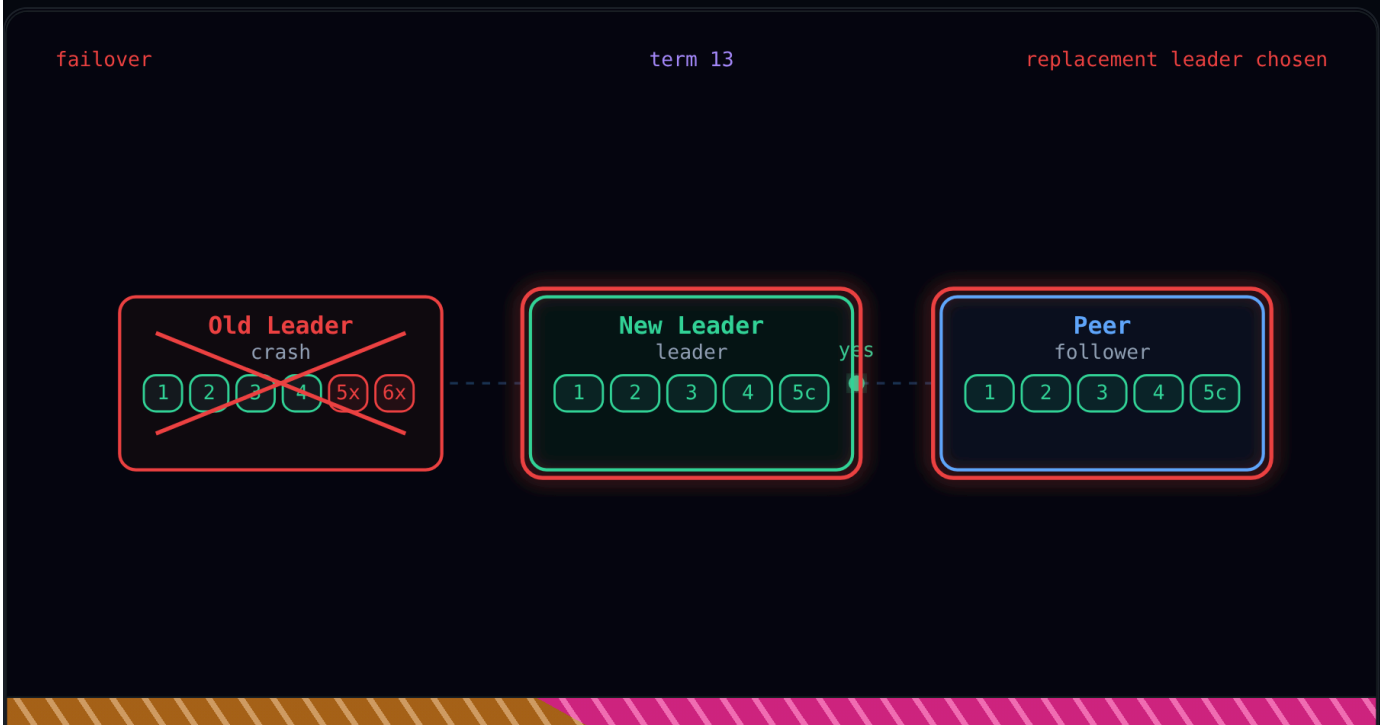
Followers that successfully store the entry reply with an acknowledgment. Toggle the Lag control to "F2 lags" and notice that one slow follower does not block the rest.

As soon as a quorum of nodes has the entry, the leader advances its commitIndex. The write is now durable. Quorum, not unanimity, is what matters here.

The leader applies the committed command, and any lagging follower catches up later through normal AppendEntries. The takeaway: Raft can lose a minority of nodes and still keep your data safe. Next, in Failover and Rejoin, we will see what happens when the leader itself crashes.

STAGE 6 OF 7

Failover + Rejoin



06 FAILOVER + REJOIN

In the last stage we saw how a healthy leader commits data through quorum. But what if that leader goes down? Here the old leader still holds a few uncommitted tail entries when failure strikes. Use the Tail stepper to control how many stale entries it has.

Switch the Failure control between Crash and Partition to see two flavors of the same problem. Either way, the old leader drops out of quorum communication and the remaining nodes stop hearing heartbeats.

This looks exactly like the Timeout Trigger stage. A surviving follower times out, starts a higher term, and wins a fresh majority vote. The whole election cycle we learned before repeats here automatically.

The replacement leader begins accepting new writes. Clients do not need to know the old leader crashed. The cluster keeps working because quorum still holds among the survivors.

When the old leader comes back online, it discovers a higher term number. The rule is simple: a node that sees a higher term must step down immediately and become a follower.

The new leader overwrites the stale uncommitted tail entries on the returning node so every log matches. This is the safety net that keeps Raft consistent. In the Recap we will connect all five stages into one self healing loop.

STAGE 7 OF 7

Recap

07 RECAP

We started with Stable Term, where regular heartbeats keep followers from starting elections. Remember, as long as the leader keeps talking, the cluster stays calm.

When heartbeats stop, Timeout Trigger kicks in. Randomized timers make sure only one follower crosses the threshold first, avoiding a pile-up of candidates. This is the same randomization trick we saw preventing chaos in stage two.

That candidate then runs a Majority Vote. Peers check the term number and log freshness before granting their vote. A strict quorum is needed, not unanimity, which connects back to the quorum idea from setup.

Once elected, the new leader uses Replicate and Commit to copy every client command to a majority of followers before marking it durable. This is why Raft can lose a minority of nodes and still keep your data safe.

When the leader crashes or gets partitioned, Failover and Rejoin handles the recovery. The returning node sees a higher term, steps down, and repairs its log to match the new leader.

Together these five pieces form a self-healing loop. Raft turns unreliable individual machines into one reliable group by always having a plan for who is in charge, how data is copied, and what happens when things go wrong.