READ MODE · ANSWER FIRST

How Apache Pinot segment commit protocol works

See how Apache Pinot elects one segment committer, finalizes metadata, synchronizes replicas, and recovers from failures.

CHEAT SHEET · 7 ESSENTIAL IDEAS

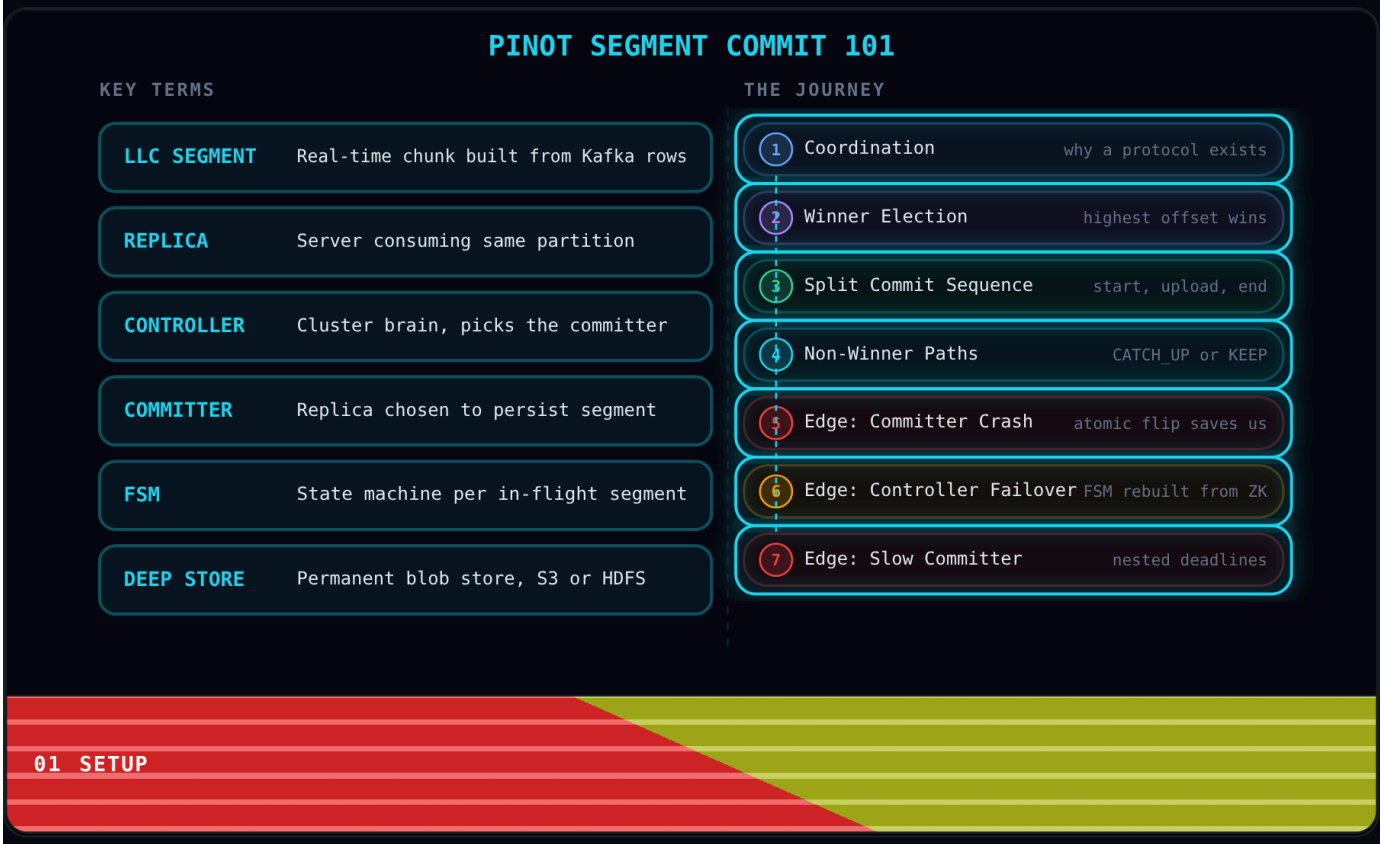
The whole story in 7 lines

Pinot turns several live replicas into one durable segment by electing one committer and treating ZooKeeper metadata as the final truth.

1. Replication supplies fault tolerance. Coordination supplies one durable answer.
2. The normal path compares reports and uses the highest offset so the finished segment covers the most consumed rows.
3. Split commit separates moving a large file from the small metadata decision that makes it official.
4. A lagging replica catches up, an exact match can rebuild locally, and an overshot copy is discarded for a safe download.
5. No DONE metadata means no official commit, so the FSM can abort and elect a surviving replica.
6. DONE is recoverable immediately, IN_PROGRESS can restart, and COMMITTING needs validation repair rather than an invented success.
7. A slow winner may request more time, but the whole transaction still has a hard thirty minute ceiling.

STAGE 1 OF 9

Setup



Pinot is a database for fast analytics on fresh events. We will follow one group of live rows as several servers turn it into one permanent segment.

A segment is a chunk of rows with its own indexes. A replica is one server consuming the same stream partition as its peers, so another copy can continue if one server fails.

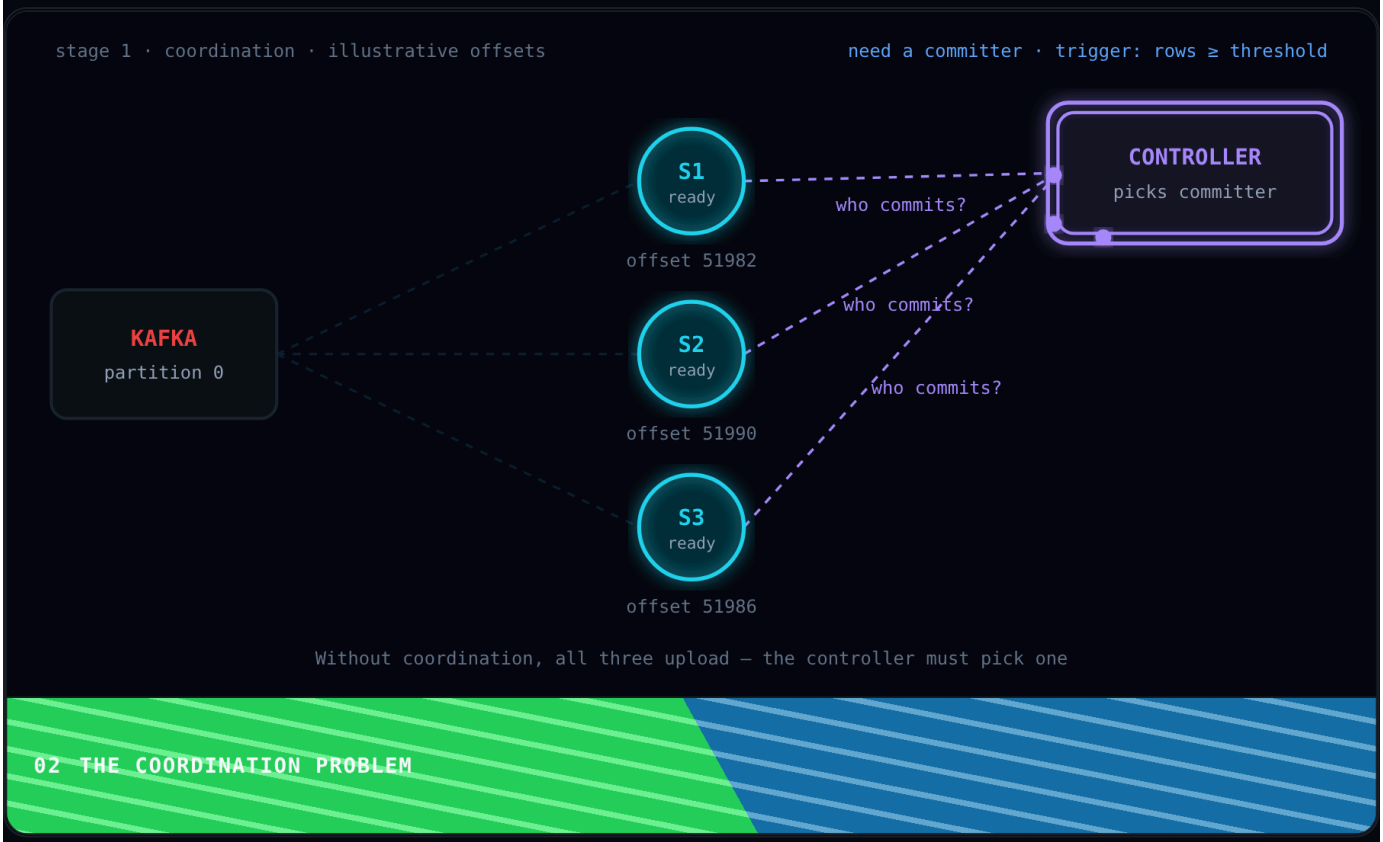
The controller coordinates the decision. The committer is the one replica chosen to persist the final segment file. The other replicas must line up with that result.

An FSM is a small state machine in controller memory. ZooKeeper stores durable segment metadata, while deep store holds the finished file in storage such as S3 or HDFS.

Our route is consume, elect, commit, synchronize, and recover. Now let us begin with the coordination problem created by several replicas finishing together.

STAGE 2 OF 9

The Coordination Problem



02 THE COORDINATION PROBLEM

The setup gave us the cast. One stream partition now feeds several replicas, and each server appends the same ordered events into its own segment in memory.

The illustrated offsets differ slightly because replicas poll independently. Use the Replicas control to see why more copies add safety and also create more reports to coordinate.

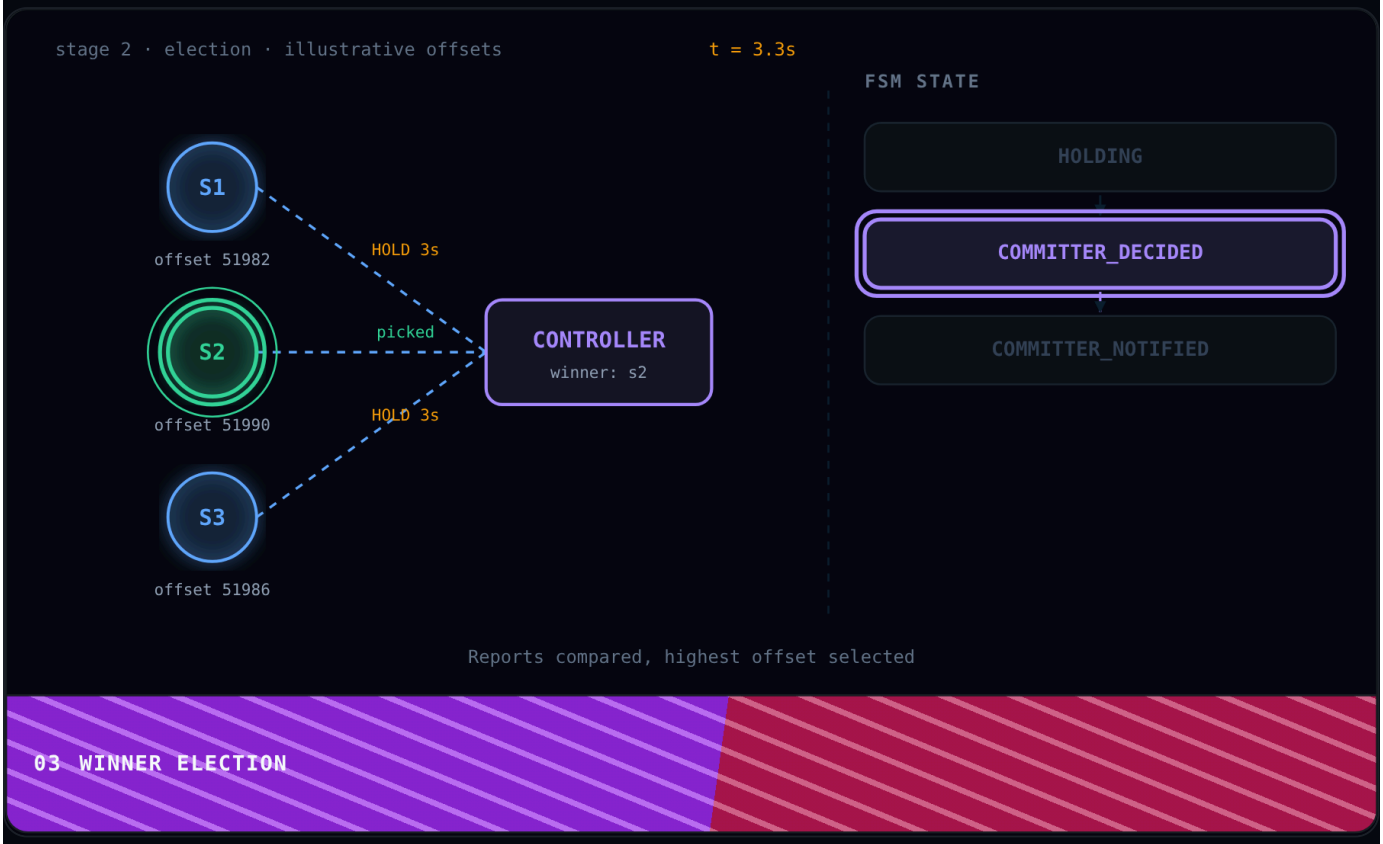
Use the Trigger control. A row limit, time limit, or partition end can stop consumption, and each choice visibly changes the condition attached to the segment.

The trigger fires. Every replica stops adding rows and holds a complete segment in memory. If all replicas are ready, who should upload?

All replicas report to the controller instead of uploading blindly. Coordination turns several valid copies into one authoritative commit. Next we will see how the controller picks it.

STAGE 3 OF 9

Winner Election



03 WINNER ELECTION

The replicas are holding their segments. The first segmentConsumed report creates an FSM in HOLDING, which is the controller saying that a decision has started but is not final.

The first report carries an illustrative stream offset. The controller records it and may answer HOLD so the same replica returns after a short wait.

A second report arrives. Use the Offset spread control to separate the values and make the future committed boundary easier to compare.

The remaining replica reports too. The controller can decide after all expected replicas answer or after the roughly 3.3 second pick window expires.

On the normal path the highest offset wins. Use the Selection control to reveal the shortcut: the first row limit or partition end report can be picked immediately when it is alone.

The winner receives COMMIT with a time budget. Other replicas remain available because the chosen server has not made anything durable yet.

Election fixes both a committer and a winning offset. That offset becomes the boundary every replica must share. Next we will follow the winner through the durable commit.

STAGE 4 OF 9

Split Commit Sequence

04 SPLIT COMMIT SEQUENCE

Winner election produced one committer and one boundary. The controller now sends COMMIT, and the winner begins building an immutable segment from its rows.

segmentCommitStart checks that the caller is still the winner at the winning offset. A valid request moves the FSM toward the uploading state.

The segment bytes travel to a controller or directly to segment storage, depending on deployment policy. Use the Segment size control to change the visible upload amount.

The upload fills a temporary location first. This keeps a partial file from looking like a finished segment if the network breaks.

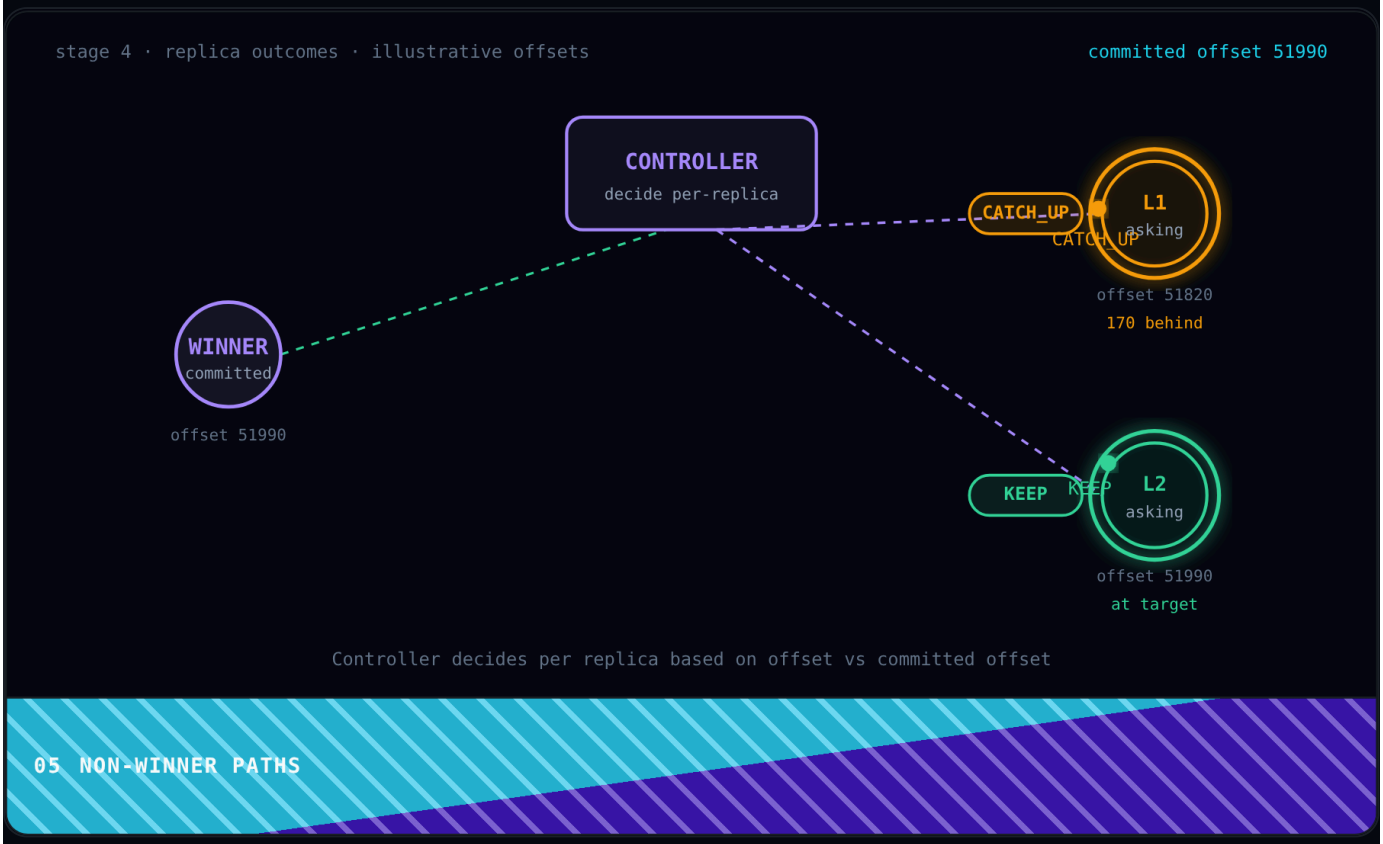
The complete file moves into its permanent location. Use the Commit mode control to compare a split transaction with the older combined RPC path.

segmentCommitEnd asks the leader to finalize metadata. The file exists, but the durable segment status has not reached DONE yet. What makes the commit official?

The controller commits metadata and ZooKeeper reaches DONE. That small durable flip is the official boundary. Next we will see how every losing replica lines up with it.

STAGE 5 OF 9

Non-Winner Paths



05 NON-WINNER PATHS

The winner made one boundary durable. Two non-winners still hold their own rows in memory, so each reports its offset again and asks what to do.

The controller compares each illustrated offset with the committed offset. Use the Loser offsets control to place both replicas behind, matched, ahead, or on different sides. What response should each get?

The lagging replica gets CATCH_UP while the exact match gets KEEP. The two responses appear together because the controller decides independently for each replica.

CATCH_UP consumes to the target. KEEP builds locally. An offset beyond the committed boundary gets DISCARD, which leads to downloading the authoritative segment.

Every replica finishes with the same immutable segment even though only one uploaded it. Next we will crash that committer before the metadata flip.

STAGE 6 OF 9

Edge: Committer Crash

06 EDGE: COMMITTER CRASH

The happy path ended with DONE. Now we stop the winner earlier, while ZooKeeper still says IN_PROGRESS and the FSM only knows that an upload is expected.

Use the Crash point control to move the failure around the tentative part of the commit. In every case, commitEnd and the durable DONE transition have not completed.

The winner disappears. The controller cannot assume failure means the upload is safe, so it keeps the FSM open until another request or the deadline supplies evidence.

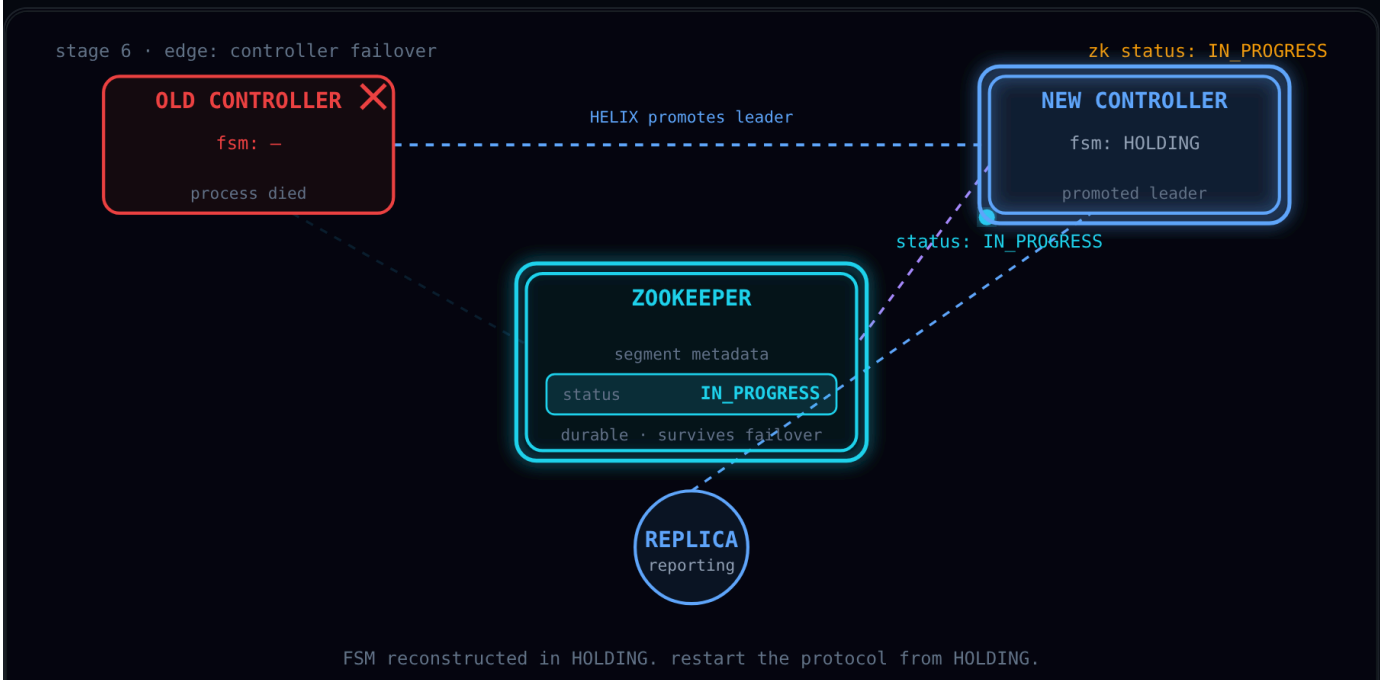
Time reaches the commit deadline while ZooKeeper remains IN_PROGRESS. The file is not authoritative, so what can the controller safely do?

The FSM becomes ABORTED and the abort metric increments. The durable status never claimed success, so no half-written segment can become official.

Surviving replicas report again. Their rows and offsets still exist, which lets the controller create a fresh FSM instead of repairing a partial commit.

A new winner is elected and the commit retries cleanly. Next we will fail the controller itself and ask which state survives that loss.

Edge: Controller Failover



07 EDGE: CONTROLLER FAILOVER

A replica has started the protocol and the current leader holds an FSM in memory. We now remove that controller before the segment becomes durable.

The old process dies and its FSM vanishes. Losing this state is acceptable because the FSM records coordination in progress, not a durable success.

Helix promotes another controller. The new leader starts with no FSM for this segment and waits for a replica retry.

A replica sends segmentConsumed to the new leader. That request is a clock tick for the protocol and gives the controller a reason to reconstruct state.

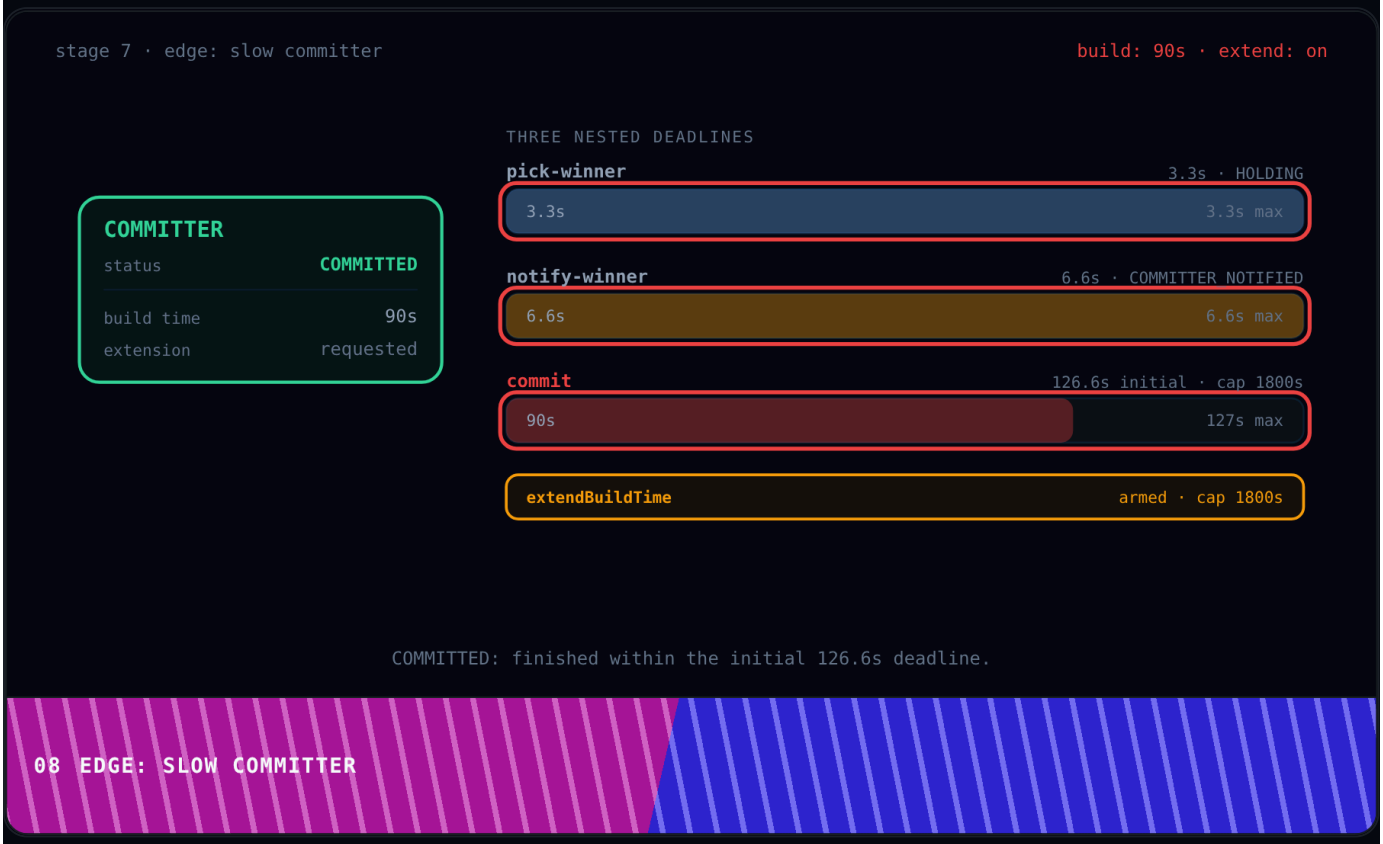
The controller reads ZooKeeper before answering. Use the ZK segment status control to expose IN_PROGRESS, COMMITTING, and DONE. What durable fact can the new leader trust?

IN_PROGRESS restarts in HOLDING and DONE reconstructs COMMITTED. COMMITTING is not guessed through; current Pinot lets validation repair that exceptional state.

The replacement leader responds from durable evidence, not lost memory. Next we will keep every process alive and test the protocol with a very slow build.

STAGE 8 OF 9

Edge: Slow Committer



08 EDGE: SLOW COMMITTER

Controller failover taught us which state is durable. Now every process stays alive, but the chosen committer takes a long time to build its segment.

The pick window is about 3.3 seconds. It prevents HOLDING from waiting forever when not every expected replica reports.

The notify window reaches about 6.6 seconds from FSM creation. It gives the chosen winner time to return and receive COMMIT.

The configured commit timeout is added to that notification budget. Use the Build time control to move the winner inside or outside the allowed window.

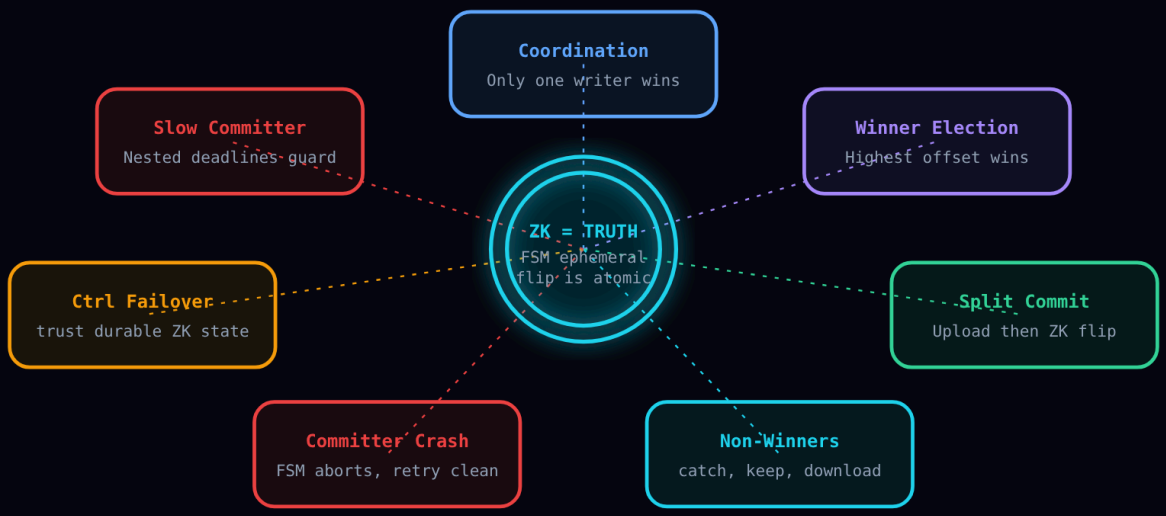
Use the extendBuildTime control. A working committer can request a later deadline before time expires, and the scene stretches the commit budget to match.

Extensions are capped at 1,800 seconds from FSM creation. This keeps a buggy or stalled build from holding one partition forever.

The timers converge on a verdict. A build inside its budget commits, while an expired or over-cap build aborts and increments the state machine abort metric.

Pinot gives real work room to finish and still guarantees eventual release. Now let us step back and connect all seven technical stages.

Recap



09 RECAP

We started with several replicas building the same live segment. The controller prevented several valid copies from becoming several competing commits.

Then segmentConsumed reports let the controller choose one committer and one winning offset for the shared boundary.

Split commit moved the file first and finalized metadata last. ZooKeeper reaching DONE made the segment official.

Non-winners compared themselves with that boundary. They caught up, kept a matching copy, or downloaded the committed file.

A crashed committer left no DONE record, so the controller safely aborted its tentative FSM and started another election.

A failed controller lost its FSM but not ZooKeeper. The replacement leader reconstructed only what durable metadata could prove.

A slow committer could request time within a hard cap. Layered deadlines kept every phase from blocking forever.

Together the pieces form one rule: coordinate in memory, commit through durable metadata, and retry from replicas whenever tentative work fails.