CHAPTER 2

The Sequential Bottleneck

The root-cause engine's One-Truth Pipeline, revisited.

THEN

From Flood to Cause, Chapter 4, runs the root-cause engine as a strict sequence rather than a vote: Graph, then Telemetry, then Database, then the language model with RAG, then a confirmed and audited conclusion. Each stage holds sole authority over a different question and checks or enriches the result it receives, rather than casting a ballot alongside the others — the volume is explicit that diagnostic ensemble voting, blending the verdicts of unlike methods into one score, is a category error dressed up as arithmetic.

THE MECHANISM, IN DETAIL

Only two of the four stages can actually stop the flow, and the difference between them is the entire mechanism worth understanding. The graph stage halts outright — "no path → STOP" — before telemetry, the database, or the model are ever consulted. Telemetry, running second, has no equivalent authority: a timing conflict downgrades a hypothesis to a flag, never a stop. The database stage can downgrade a candidate whose component turns out to be healthy. Only the model's retrieval gate can halt again, and only for a different reason — no grounding documents, not no causal path. The architecture, in full:

```
Graph          Telemetry      Database       LLM + RAG      Conclusion
  |               |              |               |              |
no path ->    conflict ->   healthy ->     no docs ->
  STOP            FLAG          DOWNGRADE       WARN

  (halts)        (qualifies)    (qualifies)     (halts)
```

The asymmetry is the point and also the risk: of the four stages, exactly one sits first in line with the unilateral power to stop everything behind it before any other evidence is even gathered. Whatever the graph does not yet know, the rest of the pipeline never gets a chance to compensate for, because it never runs.

WHY THIS SEEMED SUFFICIENT AT THE TIME

Putting the graph first is not an oversight; it is the correct expression of the volume's central principle, that the engine decides and the model only explains. A causal claim has to be causally possible before anything downstream is allowed to elaborate on it, and checking causal possibility first is also the cheapest place to fail closed — an incident the graph cannot connect at all is stopped before telemetry queries, retrieval, or a model call are ever spent on it. The ordering is an efficiency decision as much as a safety one, and on both counts it is the right one. The gap is not in what runs first. It is in what happens to the evidence gathered by everything that would have run next.

LEARNED — confirmed boundary

Get the mechanism right before assigning it a moral, because it is easy to misremember which stage is doing the stopping. The instinct is to picture noisy telemetry jamming the pipeline mid-flight — a plausible-sounding failure that is not, in fact, how the sequence is built, as the previous section shows directly. The single stage that can stop the whole pipeline before it starts is the graph, first in line, and that placement is the real finding, not an invented

one: the volume's own Honest Boundary for this chapter already names it — a pipeline that fails closed concentrates all of its risk on its first stage, and if the graph is incomplete, the engine abstains even when telemetry and the component registry, taken together, already held the answer.

That is a sharper problem than a stall. A stall is visible — someone notices the system hung and goes looking. A false abstention is not: the engine returns exactly the answer it is supposed to return when it genuinely knows nothing, and there is no signal anywhere in that output to distinguish *nothing caused this* from *the cause exists but the graph has never been told the edge is there*. Both look identical from the operator's side of the screen. Only one of them is fixable by better data reaching the engine before the next incident.

NOW

Do not weaken the halt. The graph's authority to stop the pipeline is the load-bearing safety property of the whole design — failing closed on genuinely absent evidence is exactly right, and nothing here argues otherwise. What is missing is a second, distinguishable outcome between *confirmed cause* and *abstained*: a **provisional candidate**, produced only when telemetry corroboration and the component registry agree on a hypothesis the graph does not yet connect. A provisional candidate is not promoted to a conclusion — it is logged, audited, and shown to the operator under a visibly different treatment than an engine-decided cause, with its own confidence band and a mandatory acknowledgement before action, precisely so it can never be mistaken for the thing Chapters 1 through 3 spent their whole argument earning: a cause the engine actually decided.

Concretely, this means the graph stage's halt gets a third exit alongside pass and stop: `PROVISIONAL → requires telemetry+registry agreement, tagged, escalation only`. It reuses the same `Candidate` and `DiagnosisRun` tables Chapter 8's schema already defines, adding one status value rather than a parallel structure — the same discipline as Chapter 1's fix, extending an existing audited path instead of building a second one beside it.

WHAT THE FIX ITSELF RISKS

A provisional-candidate path is a second, weaker channel next to the one the entire book is built to trust, and every weaker channel is a place a tired operator can learn to stop reading the label — the same automation-bias risk the original volume already names in its eleventh chapter, now given a second, more tempting door to walk through. The mitigation has to be procedural, not just visual: a provisional candidate must cost the operator an extra deliberate action to act on, never the same click as a confirmed cause, or the distinction exists in the schema and nowhere else that matters.

There is a second risk worth naming precisely because it looks like an improvement: a provisional-candidate channel that works well will quietly reduce the pressure to actually fix the graph. If telemetry and the registry can usually get an operator most of the way there without a graph edge, the incentive to invest in the tedious, unglamorous work of graph curation weakens exactly when it is needed most. The honest complement to this fix is operational, not architectural: a provisional candidate seen twice for the same missing edge should become a mandatory graph-review ticket, not a permanent, comfortable second-class citizen of the pipeline.

RETROSPECTIVE BOX

Would I do it again?	Yes, on the halt itself — fail-closed on the graph stage is the right call and stays unchanged. No, on stopping there: shipping only one silence for two different failures was the incomplete part.
What was the single worst mistake here?	Not distinguishing “nothing caused this” from “the cause exists but the graph was never told” — the pipeline abstains identically in both cases, and only the second one is fixable by better data reaching the engine.
What would I tell my younger self?	A fail-closed system needs two different silences, not one, or your most careful operators will eventually stop trusting the difference between them — and build the escape hatch so it makes the underlying gap more visible, not less urgent.