PART 3 OF 3 · DURABLE EXECUTION

Why Most Agents Die in Production

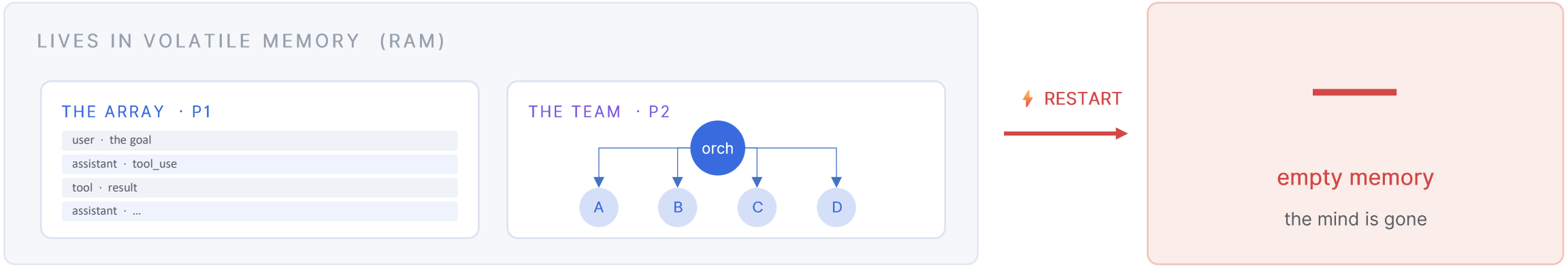
The structural failure teams overlook — and the one idea that fixes it. No magic: where state lives, why a deploy erases it, and how an agent comes back from the dead instead of starting over.

Why agents die · Externalizing state · Resume, not restart · What it unlocks

Clausey

We taught them to work together. We never taught them to survive.

Part 1 gave us the array. Part 2 gave us the team. But all of it — every message, every subagent, the record of who is doing what — lives in **one running process's memory**.

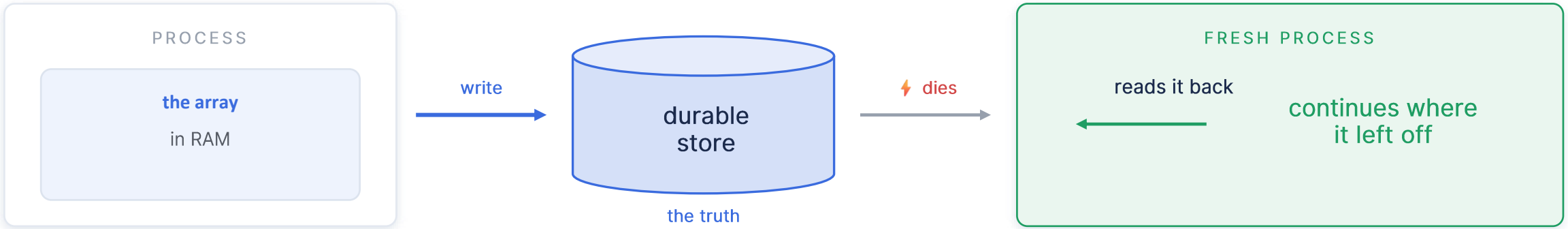


One restart and the whole mind is gone — array, team, and all.

Clausey

Write the array down.

Part 1's lesson was **"memory lives in your code."** But your code lives in a mortal process — so memory has to live **deeper than code**: in a durable store the process can't take with it.



If the truth lives outside the process, the process becomes disposable.

Clausey

SECTION 01 / 04

Why agents die.

01

The failure mode that never shows up in a demo — and why it's the baseline, not the edge case.

1 · Why agents die

2 · Durable execution

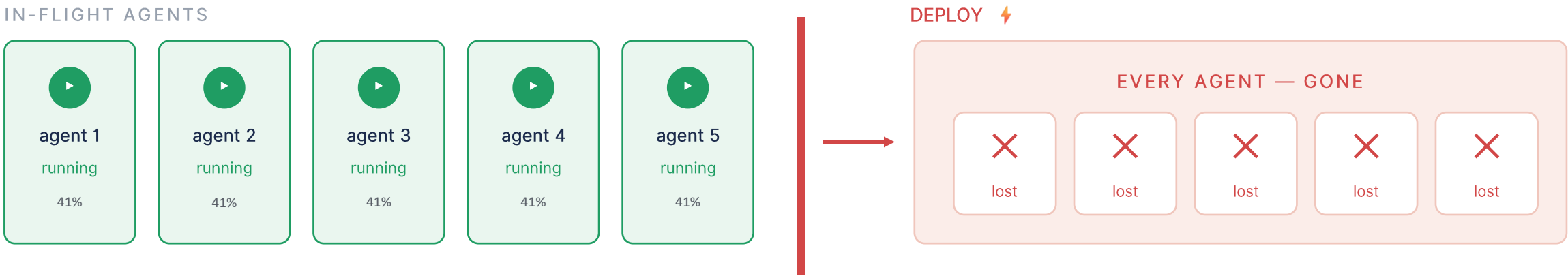
3 · Resume, not restart

4 · What it unlocks

Clausey

A routine deploy kills every agent.

Crash, timeout, out-of-memory, a reclaimed container — those happen too. But the one that gets everyone is the most ordinary: a **normal Tuesday deploy** restarts the box, and every in-flight agent silently dies mid-run.



This isn't the rare failure. It's Tuesday.

Clausey

There is no resume.

When the process dies, the array dies with it. There is no checkpoint to fall back to — the next run starts cold, from the very first message. All the work so far is simply gone.

40 min of work

15 iterations

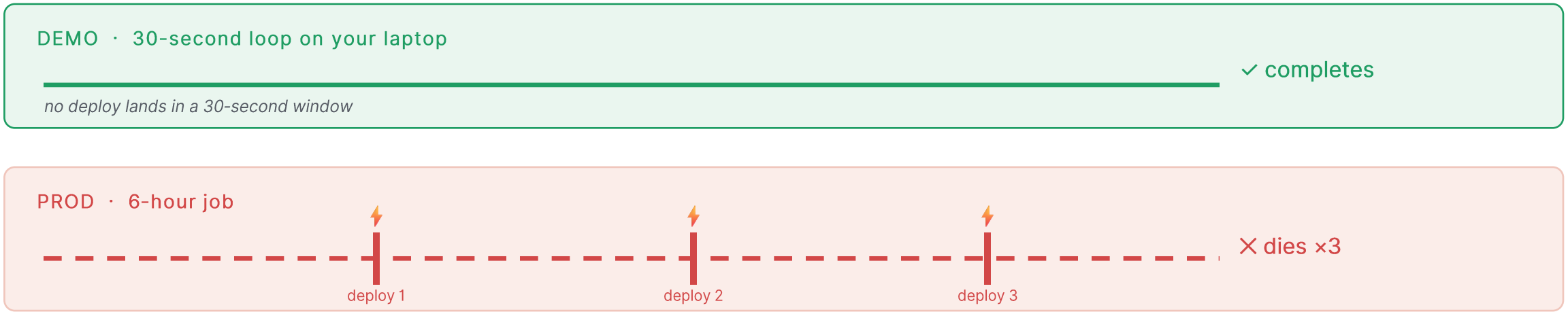
59 tool calls

Fifty-eight steps of work, one death, nothing saved.

Clausey

You can’t test your way out of this.

The demo gap *is* the prod gap. A 30-second loop on your laptop never meets a deploy. A 6-hour production job meets three. It’s **architectural, not bad luck** — no amount of testing the happy path removes it.



The gap isn’t reliability. It’s runtime that outlives a process.

SECTION 02 / 04

What durable execution actually is.

02

One definition, four moving parts. Two you set up in advance — two you'll watch run during a failure.

1 · Why agents die
2 · Durable execution
3 · Resume, not restart
4 · What it unlocks

Clausey

A computation that survives the death of its process.

That’s the whole idea. It rests on four ingredients — grouped not by when you write them, but by when they act: two you decide up front, two that kick in at the moment of recovery.

1

Externalize state

The truth leaves the process for a durable store.

SET UP FRONT

2

Idempotent retries

A step run twice does no extra damage.

SET UP FRONT

3

Checkpoint

Each completed step is recorded as it finishes.

AT RUNTIME → *next section*

4

Automatic recovery

A fresh process picks the work back up on its own.

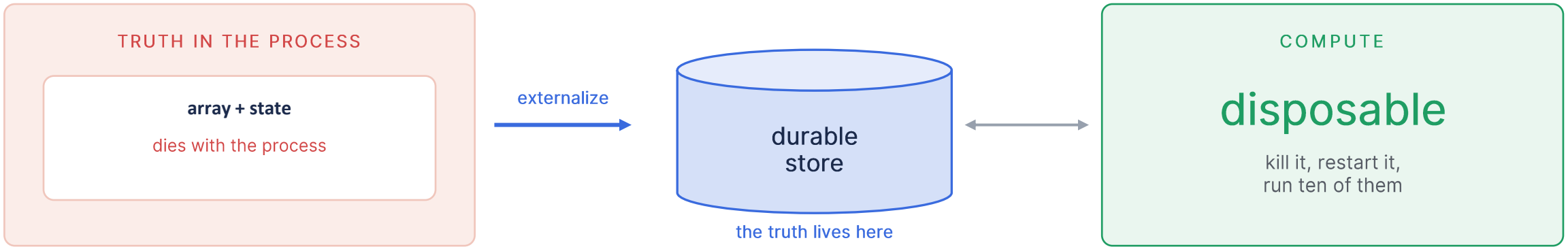
AT RUNTIME → *next section*

Two you design. Two you watch work.

Clausey

Decide where the truth lives.

This is the load-bearing one. Move the source of truth **out of process memory and into a durable store** — then the compute that reads and writes it becomes disposable. Lose the process; the truth stays put.

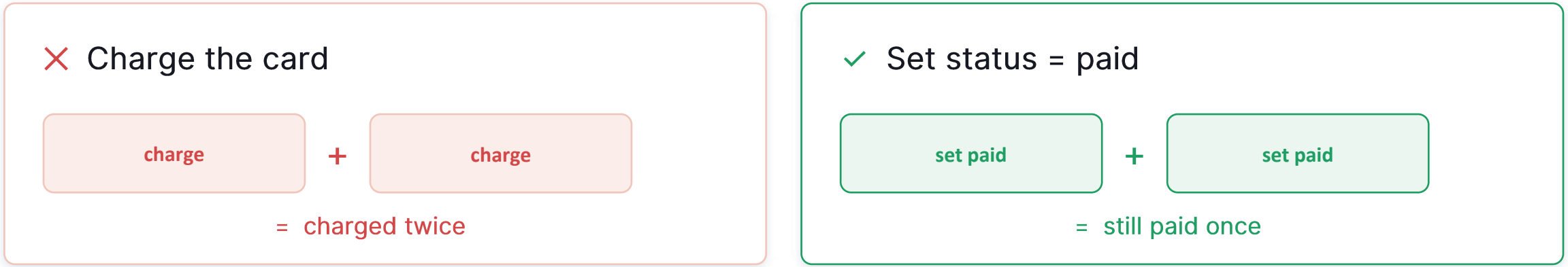


Externalize the state, and the process stops being precious.

Clausey

A step run twice can’t double-charge.

Recovery works by **re-running** — so any step that might run a second time has to be safe to. “Charge the card” twice hurts; “set status = paid” twice changes nothing.



If recovery re-runs a step, the step has to not care.

Clausey

SECTION 03 / 04

Coming back to life.

03

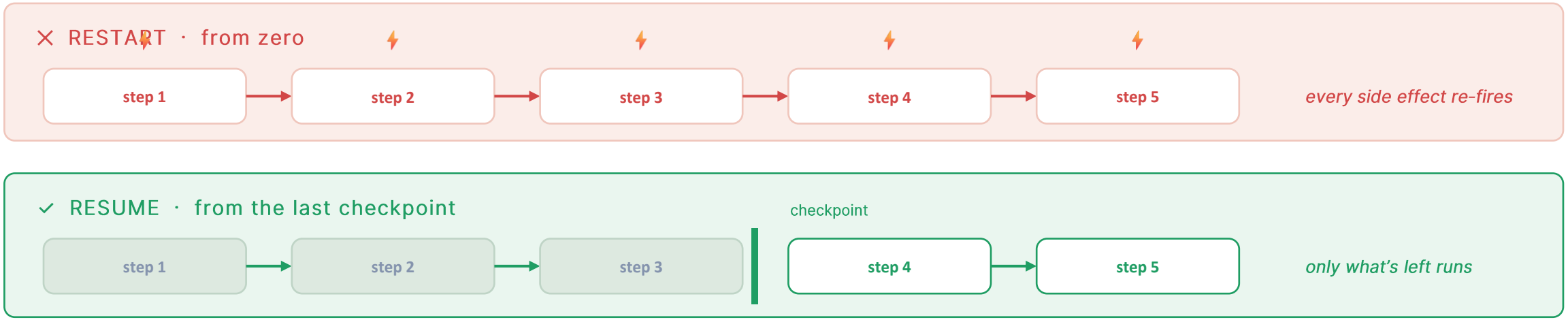
Now the two runtime ingredients, shown instead of defined: resume at the last checkpoint, not restart from zero.

1 · Why agents die
2 · Durable execution
3 · Resume, not restart
4 · What it unlocks

Clausey

Restarting re-runs everything. Resuming doesn’t.

Here are the two ingredients we set aside. A naive restart goes back to step one and re-fires every side effect along the way — dangerous. Real recovery reads the last checkpoint and continues from there.

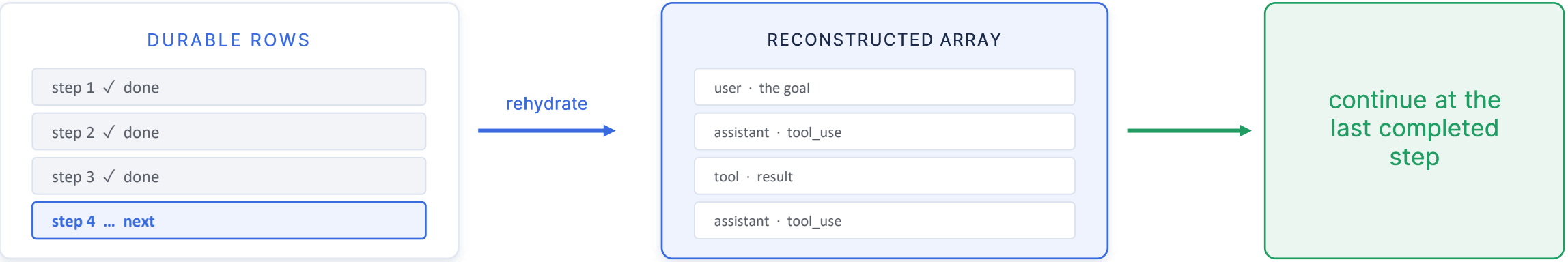


Recovery you can trust starts from the last checkpoint, not from zero.

Clausey

Rebuild the array from durable rows.

On recovery, the fresh process reads the durable record, **reconstructs the message array**, and continues at the last completed step. Same array as Part 1 — except it comes back from the database, not from RAM.



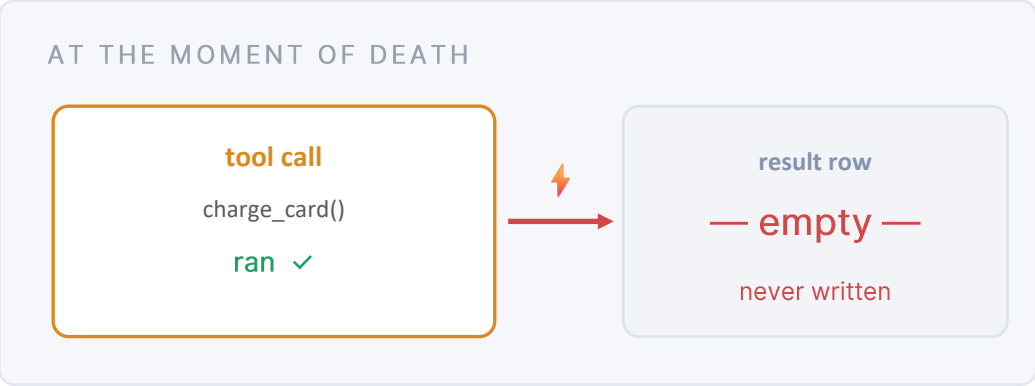
Precise: resume is at the **step grain** — not per individual tool call.

The array comes back from the database — not from the dead process.

Clausey

The tool ran. The result was never written.

The process died *after* a side-effecting call but *before* its result was saved. You can't tell if it happened — so don't silently re-run it. Flag it: interrupted, verify before retry.



✕ Silently re-run → maybe charge the card twice

the danger you must not walk into

✓ Mark interrupted → verify, then decide

check whether it landed before retrying

When you can't tell if it ran, idempotency is the only safe answer.

SECTION 04 / 04

What durability unlocks.

One mechanism, one big payoff: an agent that survives long enough to do real, autonomous work.

Clausey

Autonomy is just survival.

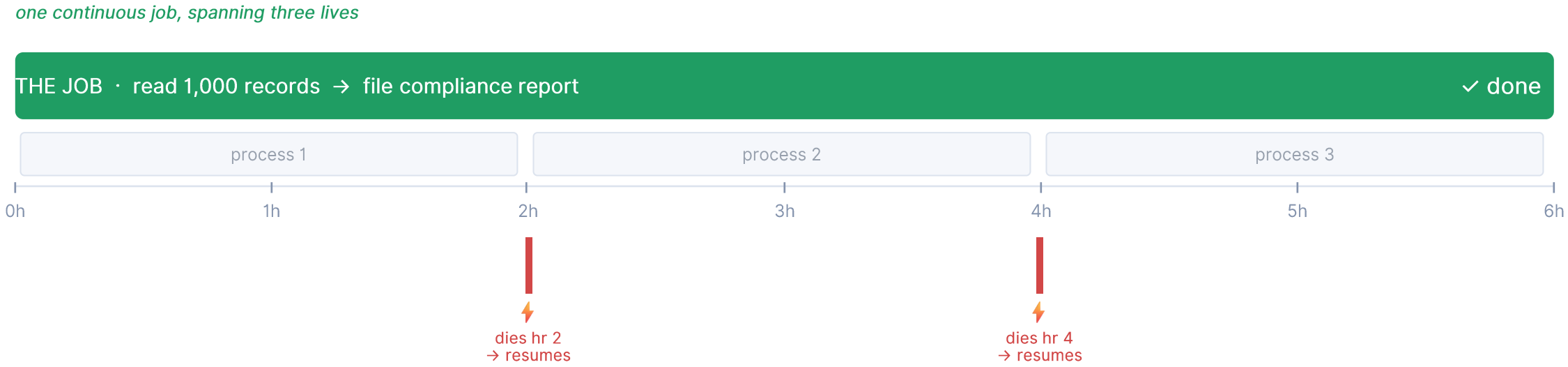
Before durability, “an agent” is a demo that runs until something kills it. *Autonomous* means it survives long enough to finish real work. Externalizing state buys exactly one big thing — longevity — and the rest of this section proves it.

The thing that survives a deploy is the thing that can work unattended.

Clausey

One job, three process lifetimes, one report.

The real task: **“read all 1,000 student records and file a compliance report.”** Hours long — and impossible the moment a mid-run deploy erased an hour’s work. With durability, it just keeps going.

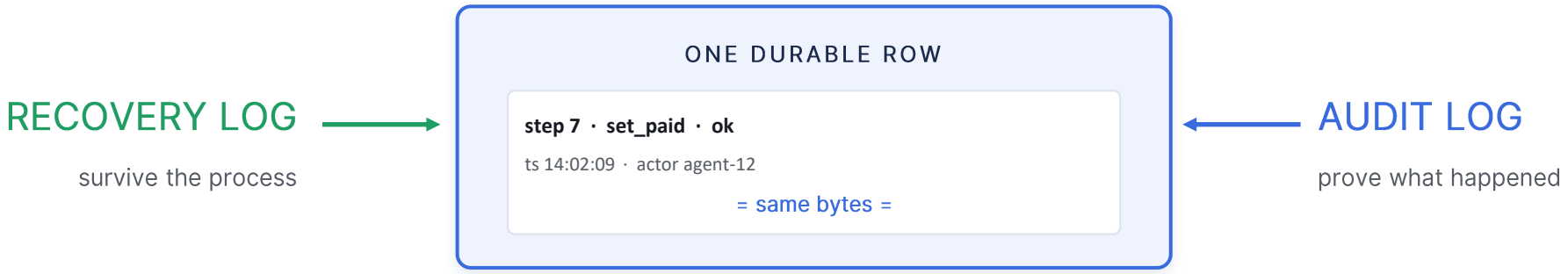


Durability is what turns *“an agent”* into an autonomous one.

Clausey

The recovery log and the audit log are the same rows.

You didn't build two things. The durable record that lets the agent survive is the very same record that makes it auditable — every state change, already persisted. Same bytes, two uses.



Verify replay any step

Control pause / gate / kill

Audit every change persisted

→ See our blog on harnesses to learn more

You built it to survive. You got governance for free.

Clausey

The durable agent, end to end.

Every piece from the trilogy, now wired to survive: the array persisted as it grows, leases that hand work to a fresh process, rehydration on recovery, and one database holding the only truth.

1

Persisted array

every step written as it happens

2

Leases + reclaim

dead work is detected and reassigned

3

Rehydration

a fresh process rebuilds and resumes

4

One database

the single source of truth

array persisted per step · leases + reclaim · rehydration on recovery · the database as the one truth

---

Same agent from Part 1 — its memory just lives somewhere a deploy can’t reach.

Durability isn’t free.

Every checkpoint is a write, and two rows that should agree can drift. So don’t pay for it when you don’t need it — a one-shot chat doesn’t, a 3-second tool call doesn’t. Reach for it when the work is too long, or too important, to lose.

THE COST

+1 write per step

drift between rows that should agree

DON’T PAY FOR IT

- ✓ one-shot chat
- ✓ a 3-second tool call
- ✓ anything you can cheaply redo

PAY FOR IT

- ★ hours-long jobs
- ★ money or records on the line
- ★ work too costly to repeat

Pay for durability only when the work is too long — or too important — to lose.

Clausey

Everyone builds the same four ingredients.

The industry doesn't disagree on *what* durable execution needs — only on **where the truth lives**. That's the whole spectrum: a dedicated engine, or your own database.

DEDICATED ENGINE

Temporal · AWS Step Functions

a separate system owns the truth

YOURS

YOUR OWN DATABASE

PostgreSQL · DBOS

the truth lives in tables you already run

SAME LINEAGE: workflow engines · the saga pattern · event sourcing

no magic — FedEx tracks a package across handoffs the same way

You may not need a dedicated engine. You always need state that survives the process dying.

THE SERIES

Clausey

That’s the trilogy.

One agent, a team, and a machine that survives. If you can draw the durable agent from memory — array externalized, leased, rehydrated, with the database as the one truth — you understand this better than most of the feed.

01 ✓ done

What Exactly Is an AI Agent?

The anatomy of one agent — down to the API call.

02 ✓ done

How AI Agents Work Together

Subagents as tools, an orchestrator, one owner per resource.

03 HERE

Why Most Agents Die in Production

Durable execution — state that outlives the process.

One agent, a team, and a machine that survives. That’s the whole stack.