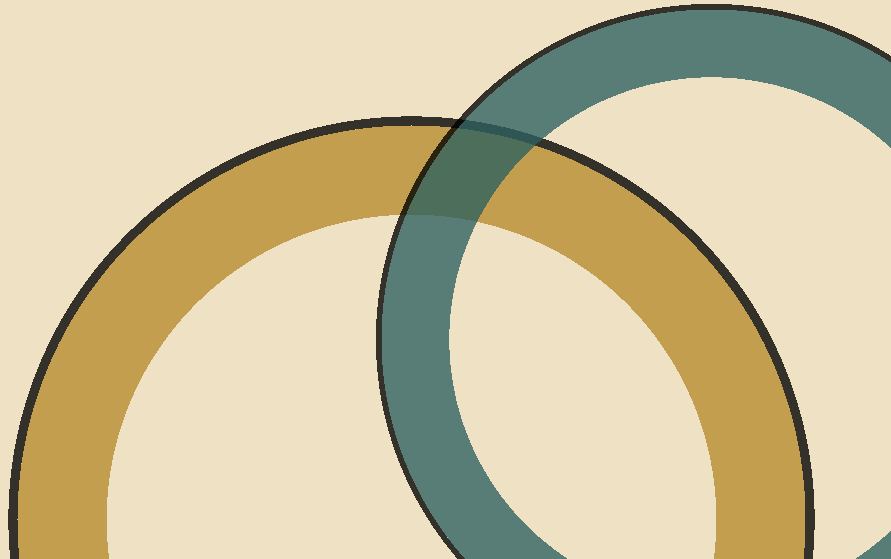
Engineering AI Agents · Week 10

Reliability, state, and the harness

Failing well · surviving a crash · degrading honestly · how much machinery to build



Slides CC BY 4.0 · from the teaching kit of *AI Agents, Engineered* by Enrique Gutiérrez · aiagentsengineered.com/teach/

Your agent is twenty steps into a research task. It queries the database; the network hiccups; the connection is refused.

In the agent you have built, what does the tool wrapper do with that exception? Write down what the model sees next.

Chapter 18, “Surface Errors Back to the Model” · the chapter’s own question

By the end of today you can

Four objectives

1. **Classify** a failure and route it to one primitive
2. Build retries with **jitter**, a shared **budget**, deadlines and a **breaker**
3. Make a run survive a **kill**: checkpoints, keys, receipts, a ladder
4. Fill the **harness grid** and decide how much harness to build

Part 1 · Chapter 18 (full book)

Failing well

Show the model the failure

- **Crash the run**: twenty steps lost to one packet
- **A bare apology**, “Error: failed”: nothing to reason with
- **An empty result**: the model fills the hole, fluently
- Instead: a **structured observation**: a stable *code*, *retryable* or not, a *hint*

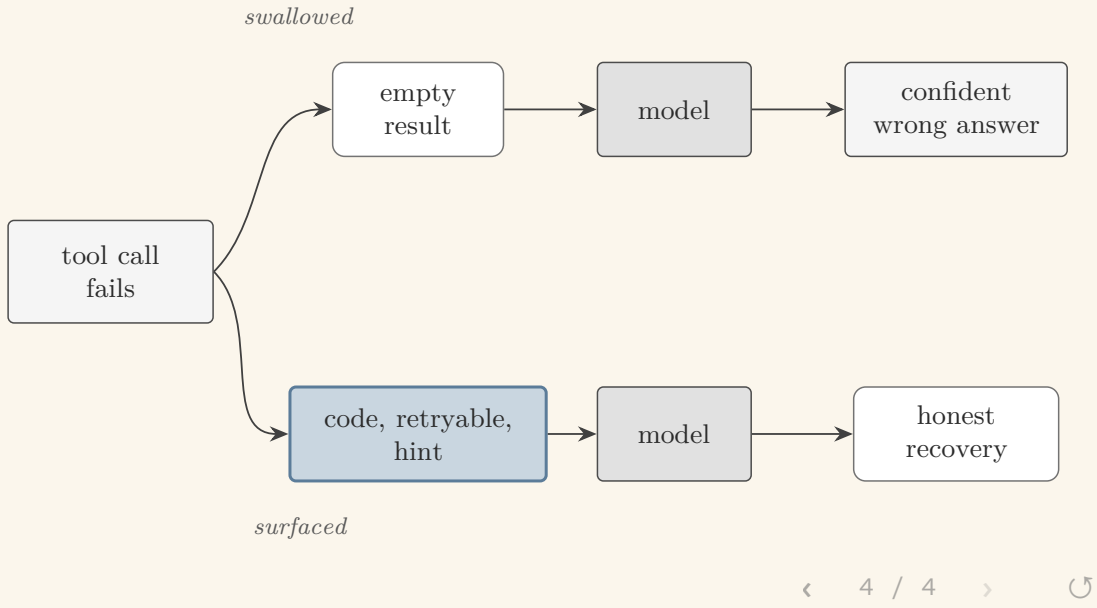


Figure from Chapter 18

Chapter 18, “Surface Errors Back to the Model” · “The failure has been laundered into an answer.”

Sort by what the failure wants done about it

Five classes, one primitive each

Class	For example	It wants
Transient	rate limit, overload, a read that timed out	a pause, another attempt
Model-recoverable	wrong tool, malformed argument, output that will not parse	back to the model, never a blind retry
Permanent	bad credential, missing record	fail fast
Policy	a guardrail trip	a loud halt
Ambiguous	a *write* that timed out	retry only if a repeat is safe

Chapter 18 · “Classify first, then route each class to its one primitive (retry, replan, fail fast, halt), and resist the beginner’s instinct to stack every mechanism on every step.”

Predict before the next slide · one minute in pairs

A service is down for ten seconds. A hundred clients fail at once; each retries at once, five tries in all. What does the service see?

Four layers of one agent each try three times. How many calls can one failure cause?

Chapter 18, “Retries, Timeouts, and Budgets” · three tries at each of four layers is the simulator’s reading

Backoff, jitter, one budget

Policy, 100 clients	Peak retries in 0.1 s	Got through
Retry at once	400	0
Back off	100	100
Back off with jitter	25	100

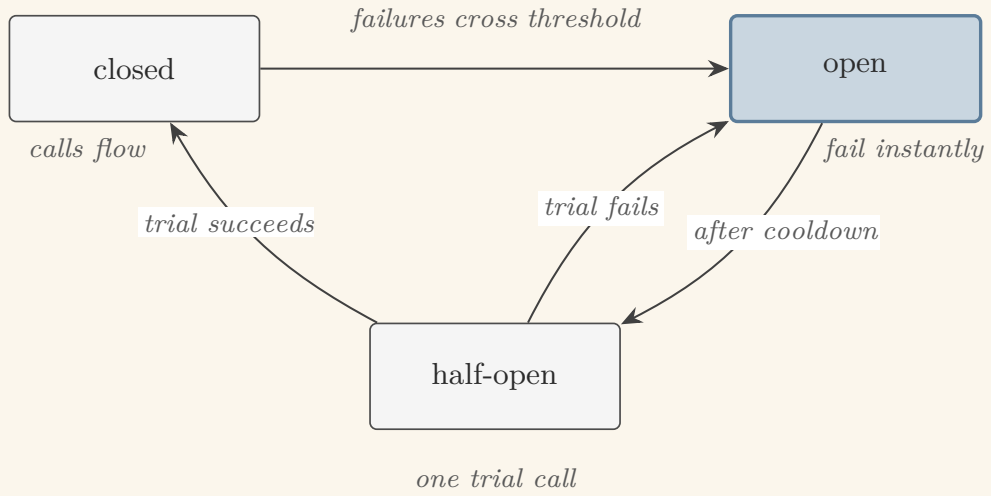
- **Backoff**: wait longer each time
- **Jitter**: a random offset, so the crowd stops arriving together
- Nested layers: 3 × 3 × 3 × 3 = **81** calls
- A **global budget** per run, across all layers

Chapter 18 · “The clients trying to recover from the outage have become the outage” · numbers: the circuit breaker simulator’s starting example (the chapter’s illustrative policy; a fixed 10 s outage, the tool’s model)

For the call that never answers, and the dependency that keeps failing

Deadlines and the breaker

- Every call gets a **deadline**; crossing it is a failure, routed by class
- A timeout on a *read*: transient. On a *write*: ambiguous
- **Closed**: calls flow, failures counted. **Open**: refuse at once. **Half-open**: one trial
- While open, route to the **fallback**



The trial alone decides: a failure re-opens the circuit, a success closes it.

‹ 4 / 4 ›

Figure from Chapter 18

Chapter 18 · “a breaker should trip on failures, and a merely disappointing result is an answer.” · the pattern comes from the distributed-systems literature

Every mechanism here costs latency, tokens and code to maintain

Price it by what it protects

A read-only lookup

Fails cheap. A retry, a deadline, maybe a breaker. Gold-plating buys nothing.

The path that charges a card

The full apparatus: classes, keys, receipts, a loud halt, a person.

Chapter 18, “Retries, Timeouts, and Budgets” · “Reliability engineering, like insurance, is priced per what it protects.”

Part 2 · Chapter 18 (full book)

State and recovery

Right now, in the agent you know best: where is the run's progress?

What a run carries

Part	What it is	Lose it and
Transcript	the message history: all the agent knows of this run	the agent forgets the run
Scratchpad	the plan, partial results, files already edited	it redoes or skips work
Ephemeral	a cache, a re-derivable plan, the loop index	recompute it, cheaply
Durable	irreversible actions done, approvals granted, the artifact	you cannot get it back

Chapter 18, "State and Recovery" · memory across runs is Chapter 9's subject; this is one run surviving its own lifetime

A saved state, at a deliberate boundary, under a stable run id

Two ways to checkpoint

Snapshot

Serialize the working state (messages, scratchpad, pending calls) every few steps; reload it on restart.

Journal and replay

Record the result of every model and tool call. On recovery, run the code from the top; journaled steps return their recorded result.

- The second is the core of **durable execution**; engines in that category supply it
- Replay needs determinism and a world that held still: record the **versions** each step used

Chapter 18, “State and Recovery” · “The code replays; the world does not.”

Predict before the next slide · one minute in pairs

Step 7 sends a welcome email. The email service accepts it. The process dies a moment later, before the checkpoint for step 7 is written.

You restore from step 6. What does the resumed run do? Would a checkpoint after every tool call fix it?

Chapter 18, “State and Recovery” · the chapter’s example

A checkpoint records position, never what the world did

Idempotency keys and receipts

- **Idempotent**: doing it twice has the effect of doing it once
- The **key** names the action: run, step, tool, arguments
- The service returns the first result on a repeat
- **Intent** before, **receipt** after; check the receipt first

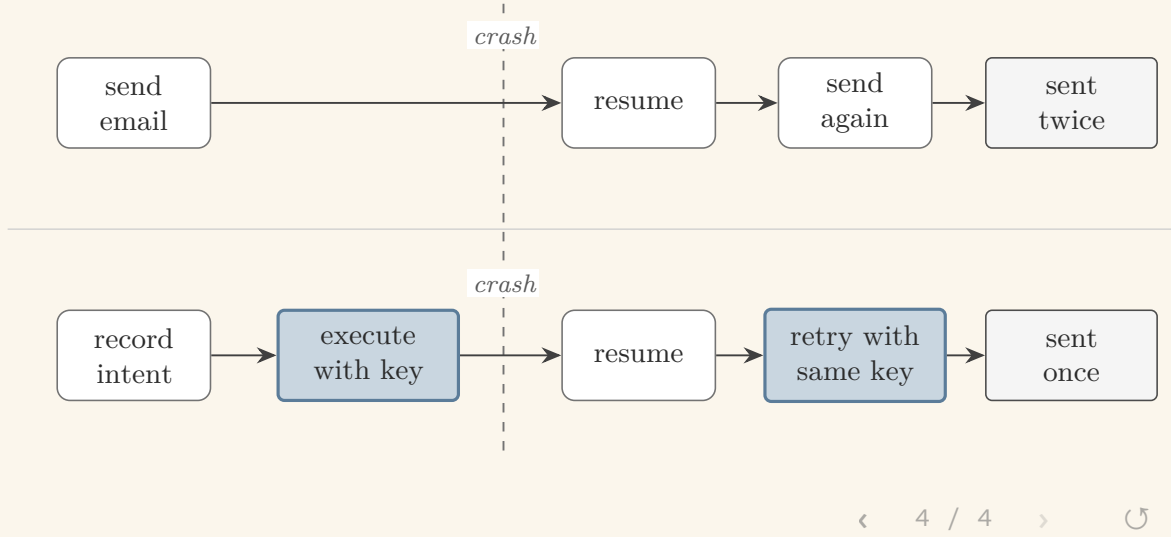


Figure from Chapter 18

Chapter 18 · “The receipts are the system of record; the transcript is the agent’s memory of writing them.”

When one action is several, and when a person sits in the middle

Sequences, approvals, the loop's shape

- **Saga**: reserve, charge, issue; each step paired with a **compensation**; on permanent failure, undo in reverse
- A human approval is **durable state**: who, when, and a fingerprint of what they saw
- A free loop, or a **state machine**: planning, gathering, acting, awaiting approval, done
- Parallel branches: append-only updates, one writer per key, merge at a join

Chapter 18, “State and Recovery” · compensations leave the world “consistent if not victorious”

The habit that separates recovery that works from recovery that is assumed

Crash on purpose

`kill after a model call` · `between effect and receipt` · `restore`

- An untested restore is a hope; the worst moments are the ones it gets wrong
- Two questions size everything: **what here is irreversible, and how long does the run live?**

Chapter 18, “State and Recovery” · “a checkpoint you have never restored from is a guess.”

Part 3 · Chapter 18 (full book)

Degrading honestly

For each critical dependency, a worse but available substitute, chosen in advance

The fallback ladder

`the primary` → `a substitute` → `a cache, marked as stale` → `a person`

- Each rung trades quality for availability, **on purpose and before the outage**
- **Graceful degradation**: narrow, keep reads alive, stop writes, and **say so**

Chapter 18 · “A degraded system that advertises its degradation keeps its trustworthiness; one that fakes full service converts an infrastructure incident into a credibility incident.”

“A worse answer beats a crash” is half true

Words degrade; deeds do not

Reads and advice

A stale cache with a warning label beats an error page. Degrade freely, and say so.

Charge, send, delete, merge

No acceptable worse version exists. Surface the failure, halt, escalate.

Chapter 18, “Fallbacks, Degradation, and Partial Failure” · “Fall back on words; fail loud on deeds.”

The bottom rung of every ladder is a person:
make the hand-off usable

An escalation a person can act on

The goal	one line
What was attempted	the action and its arguments
What failed	the code and message, in the structured shape
Already tried	so nobody tries it again
A recommended next step	the agent's best judgment
A link to the trace	week 7's record of the run

Chapter 18 · done well, escalation "converts a wedged run into a fifteen-second human decision"

Part 4 · Chapter 18 (full book)

The harness

One word, three nested senses

Which harness is yours?

Sense	What it covers	Who owns it
The builder's	the loop, tool dispatch, context assembly	whoever wrote the loop
The user's	convention files, checks, hooks wrapped around a finished product	the team operating it
This book's	everything in the system except the model	all of the above

Chapter 18, "Harness Engineering" · the builder-user split follows one practitioner's framework

Two axes: before or after the act; code or a model

The harness grid

- **Guides** steer before the act: conventions, a template, a skill
- **Sensors** check what it did and feed it **back into the loop**
- **Computational**: code, cheap, the same every time
- **Inferential**: a model; an instrument to calibrate

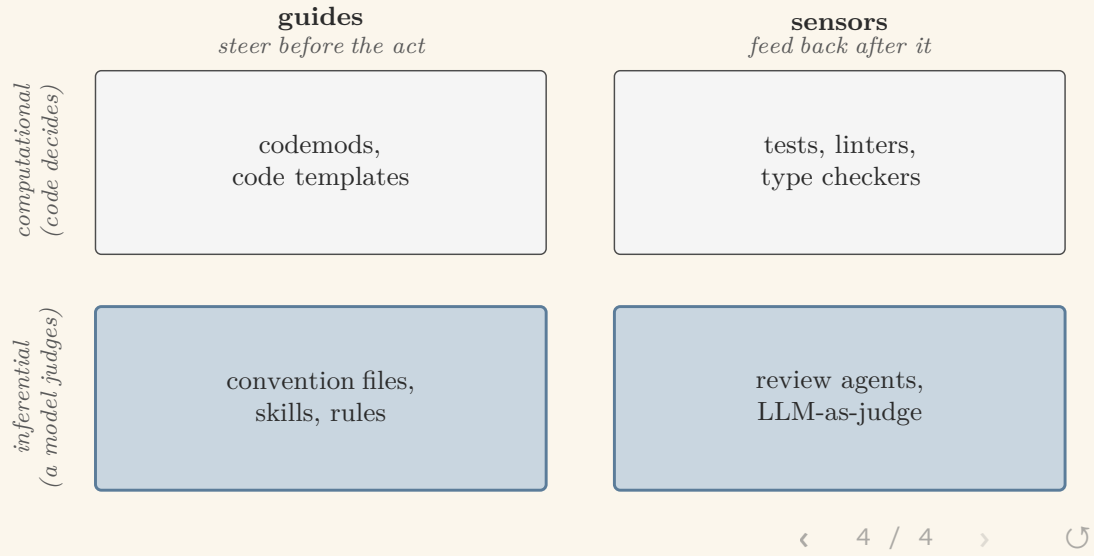


Figure from Chapter 18

Chapter 18 · “a check whose output lands on a dashboard for humans is monitoring; it becomes a sensor when its output lands back in the agent’s context.”

A sensor’s output lands in the context: write it for the model

Sensors that speak, and the loop that grows them

- Failure messages that explain the rule and offer a bounded way out steer the next attempt
- One overlooked rule kept its stock message; there, and only there, the agent learned to **raise the threshold** instead of fixing the code
- An issue that arrives **twice** leaves behind a guide or a sensor
- Place by cost: cheap on **every change**, expensive **after integration**, drift on a **schedule**

Chapter 18 · the framework calls such messages “a positive kind of prompt injection” · the threshold episode: one practitioner’s experiment, an indicator rather than a controlled result

What a harness cannot see

Green, checking nothing

100% statement coverage, no unit tests; mutation testing found 13 planted bugs the suite missed.

Intent

Sensors measure internal quality. No sensor catches a misunderstood requirement.

The legacy paradox

Most needed where it is hardest to build: no types, no tests, nothing to hang a sensor on.

Chapter 18 · “A sensor can glow green while verifying nothing” · coverage and mutation figures: the same single-codebase experiment

Predict before the next slide · one minute in pairs

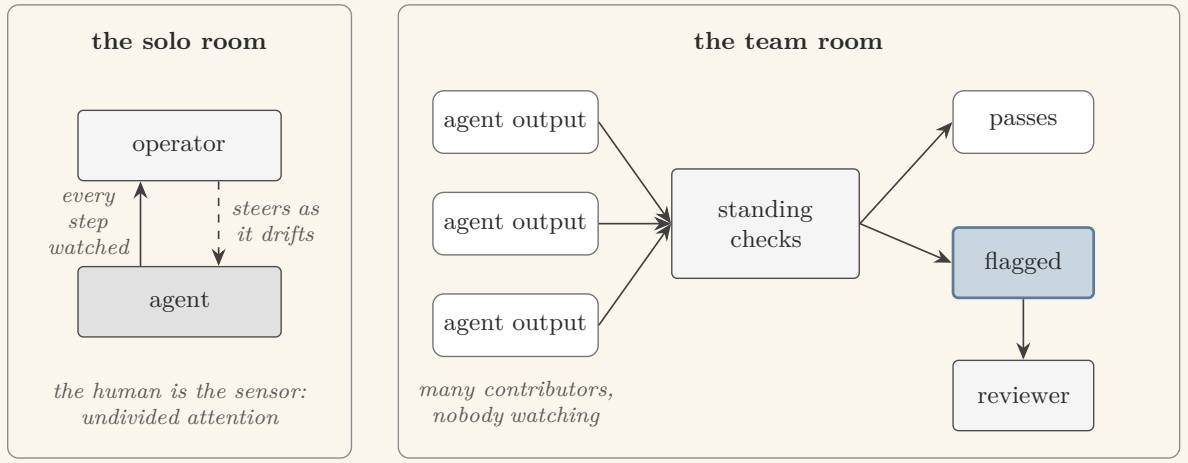
A solo developer with heavy mileage says: skip the subagents and the elaborate planning, “Just talk to it.”

A team lead whose agents work overnight wants the full grid. Who is right? What would you need to know to decide?

Chapter 18, “How Much Harness to Build” · both positions are practitioners’ the chapter quotes

A scaling law, not a verdict

- Harness scales with how long the code must **live**, how many must **trust** the output, how **unattended** the agent runs
- Unattended is the steepest of the three
- **Depreciation audit**: on every model change, reread the standing instructions and delete what is now redundant



not two truths but two rooms: the same scarce judgment, concentrated two ways

Checks save the one flagged item for a human: the same scarce judgment, concentrated another way.

‹ 4 / 4 ›

Figure from Chapter 18

Chapter 18 · “harness investment should scale with how long the codebase must live, how many people must trust the agent’s output, and how unattended the agent runs.”

In class · teams of two or three · 25 minutes · two browser tools

Break it on purpose, then map the harness

1. **A · retries (8 min).** The circuit breaker simulator: three policies, fewer tries, which layer keeps its retries
2. **B · the breaker (7 min).** Four cooldowns against two outages; the “no such record” mistake
3. **C · the harness (8 min).** The harness grid builder on a team that runs agents overnight, then on your project

Every link on the lecture page opens prefilled. Debrief, 2 minutes

Tools after Chapter 18 · each tool’s model and readings are its own, labeled on the tool · scenarios invented for the exercise

Lab 8, and reading

Lab 8 · crash on purpose

- Classes, backoff with jitter, **one budget**, deadlines
- A **breaker** and a ladder with a **stale** rung
- **Checkpoints** by run id; **keys and receipts**
- **Kill** it between effect and receipt; nothing twice

Reading for week 11

- Ch. 19, Cost, Latency, and Performance `full book`
- Ch. 20, Deploying and Scaling `full book`

This week's reading: Ch. 18 (full book) · glossary: App. B (free) · no paid API needed

Five things to keep

- Show the model the failure; **classify**, then route
- Retry the transient class only, with **jitter**, under **one budget**
- A breaker trips on **failures**, never on answers
- Checkpoints record position; **keys and receipts** record the world
- Degrade words, never deeds; scale the harness to its **room**

Engineering AI Agents · Week 10

Fall back on words; fail loud on deeds.

Slides from the teaching kit of *AI Agents, Engineered* by Enrique Gutiérrez, CC BY 4.0 · aiagentsengineered.com/teach/

Reuse, adapt and translate with attribution · creativecommons.org/licenses/by/4.0/

