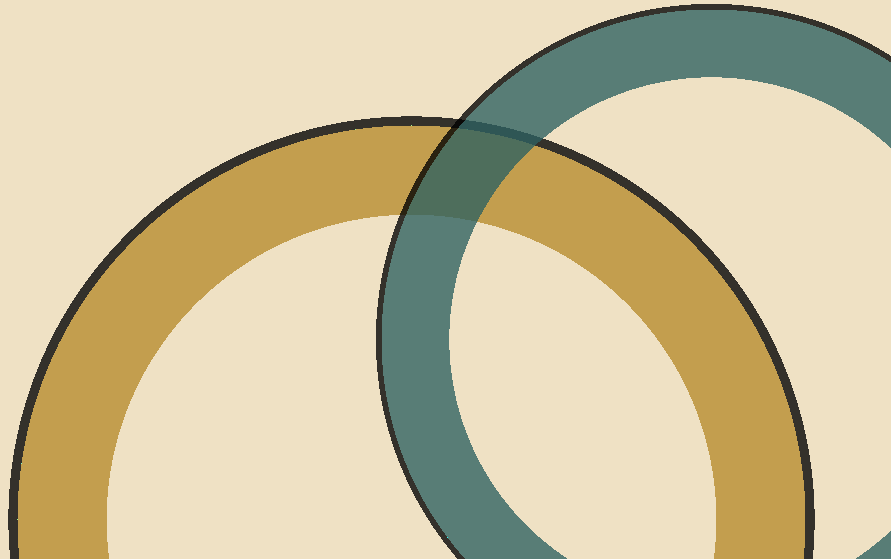
Engineering AI Agents · Week 11

Cost, latency, deployment, scaling

The invoice · the clock · serving strangers · shipping a change



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

An agent’s standing prompt is about 3,000 tokens; the task adds 200. Each step, the model writes about 300 tokens and the tool returns about 700.

A run takes ten steps. Against one chat call with the same prompt, how many input tokens does the run cost?

Chapter 19, “Where Tokens (and Money) Go” · the chapter’s napkin; every number an illustration, chosen round

By the end of today you can

Four objectives

1. Derive the **staircase** from a trace and name the five **levers**
2. Tell **first token** from **completion**, and pick the fix for a complaint
3. Design the **async job**: states, keyed submission, polling, **backpressure**
4. Write a **rollout plan**: shadow, canary, automatic rollback, one artifact

The invoice

Explainer, about four minutes: Why agent cost compounds

The model holds nothing between calls, so every step re-reads the run so far

The staircase

Step k reads $\mathbf{F + g \cdot (k-1)}$

A run of n steps reads $\mathbf{F \cdot n + g \cdot n(n-1)/2}$

Napkin, F = 3,200, g = 1,000	Input
Naive: F × 10	32,000
10 steps	77,000
30 steps	531,000
History beats prefix, per step	step 5
History over half, per run	8 steps

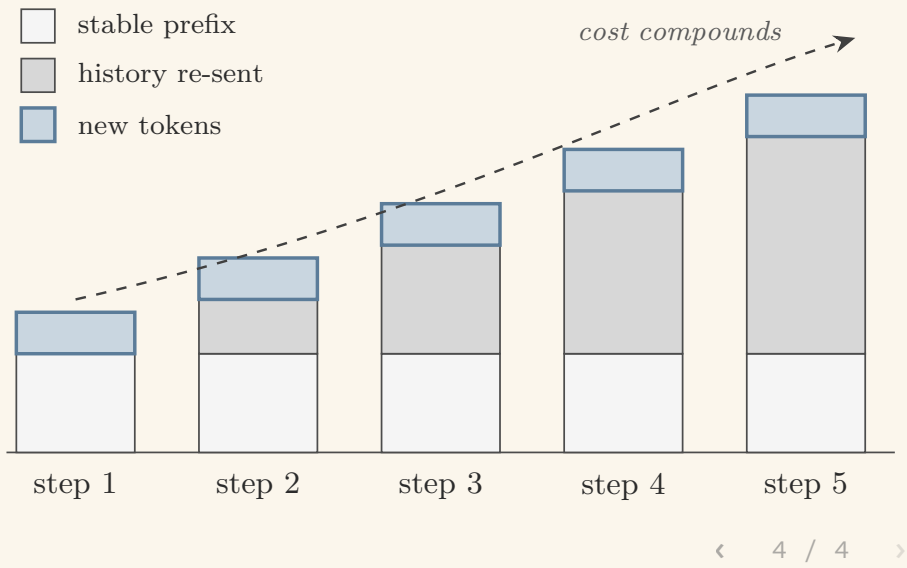


Figure from Chapter 19

Chapter 19 · “every step re-sends the entire transcript so far, because the model cannot remember what it cannot re-read.” · 531,000: the cost estimator on the napkin · the crossovers: the formula

What the napkin leaves out

Item	What it does to the bill
Output price	billed separately, usually several times the input rate; the napkin wrote 3,000 and read 77,000
Hidden reasoning	billed as output whether or not anyone sees it
Retries	full-price tokens with no visible result; one write-up's range: 10–20% where nobody instruments them
Subagents	each worker is a whole agent: its own prefix, its own staircase, plus a brief and a merge

Chapter 19 · "the agent's economics are mostly input economics" · the retry range is a cost-governance write-up's, labeled illustrative

Tokens are what capability is bought with

Yield, not abstinence

> *“token usage by itself explains 80% of the variance”*
>
> A production write-up on a research agent, quoted in Chapter 19 · one team, one workload

- Ask what share of the tokens **did work**; the waste is in retries, discarded drafts, context never read again
- **Estimate before you build**: steps, context size, rates per tier, times retries and workers
- Then let the **traces** retire the spreadsheet

Chapter 19 · “the most expensive way to learn arithmetic” is a first month’s invoice · no prices, ever: they move too fast to print

Each lever acts on one term of the bill, and each can make the agent worse

Five levers

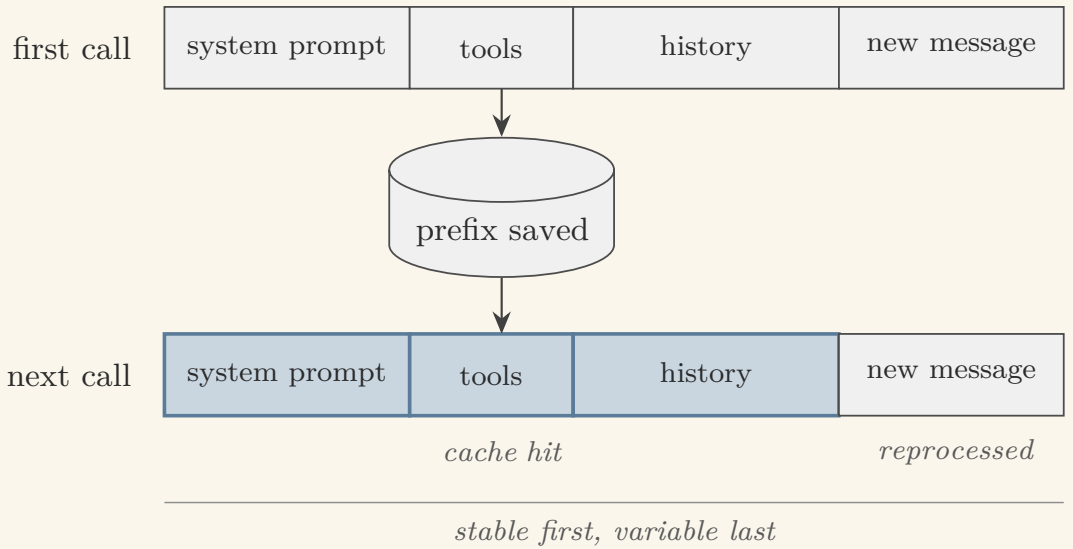
Lever	Acts on	Watch for
1 · Which model does which work	the rate paid on most steps	the “occasional” strong call on every turn; a judge dearer than the work
2 · Context discipline	*g*, paid again every later step	trimming what the task needed
3 · Prompt caching	the price of the re-read front	a changing token near the front
4 · Caps (steps, tokens, time, fan-out)	*n* and the worst case	a cap below what real tasks need
5 · Bulk work in code	data that never enters the transcript	a script nobody checked

Chapter 19, “Cost Levers” · “The pair of numbers is the deliverable; either alone is a story.”

Lever 3, the one an agent loop was built for

Prompt caching

- A call that opens with the **same bytes** as an earlier one resumes reading where the match ends
- The rule: **stable first, variable last**
- Classic misses: a timestamp in the system prompt, a user id above the tools, tools in a shifting order
- Napkin, at a 10% factor: **84%** cached → 18,680 billed. Timestamp first: **0%**



So put stable content first and variable content last.

‹ 4 / 4 ›

Figure from Chapter 19

Chapter 19 · “Cheap clutter is still clutter.” · 84% and 18,680 token-equivalents: the caching calculator on the napkin, its illustrative factor

Part 2 · Chapter 19 (full book)

The clock, and the triangle

Predict before the next slides · one minute in pairs

A. A support chat: the screen stays blank for 9 seconds, then the whole answer appears at once.

B. The first words appear in under a second; then six paragraphs crawl out over forty seconds.

C. A nightly report agent takes 50 minutes. Its trace: twenty independent lookups, two minutes each, one after another.

For each: streaming, caching, parallelism, a smaller model, or nothing?

Scenarios invented for the lecture · after Chapter 19, “Latency”

One model call, two phases with different physics

First token, last token

Prefill: reading

Everything sent is read in one pass before a word comes out. Sets the **time to first token**; grows with what you **send**.

Decode: writing

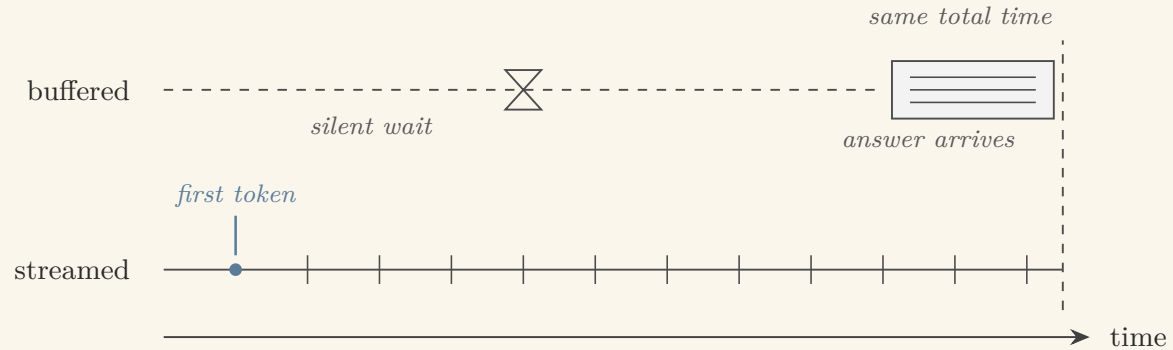
One token at a time, each after the last. Sets the rest of **completion time**; grows with what the model **writes**.

Long blank pause, then words	an input problem: send less, cache the front
Prompt start, then a crawl	an output problem: ask for less, or a faster model

Chapter 19, “Latency” · “Same complaint from the user, opposite fixes.”

Is a human watching?

- **Yes**: stream first, then spend on **time to first token**
- **No** (a pipeline, a nightly job): optimize completion and throughput; the batch discount fits here
- Streaming makes no step faster, and it can **hide** a slow backend
- A stall in mid-stream reads as a crash mid-sentence



‹ 4 / 4 ›

Figure from Chapter 19

Chapter 19, “Latency” · “So the first question of any latency effort is: is a human watching?”

Measure first: split the wall clock, then pick the lever

Where the seconds went

Most of the time is	Lever
tool calls, one after another	**run independent calls together**
prefill	cut and cache the input
decode	shorter output, smaller model
waiting in a queue	capacity: Part 3

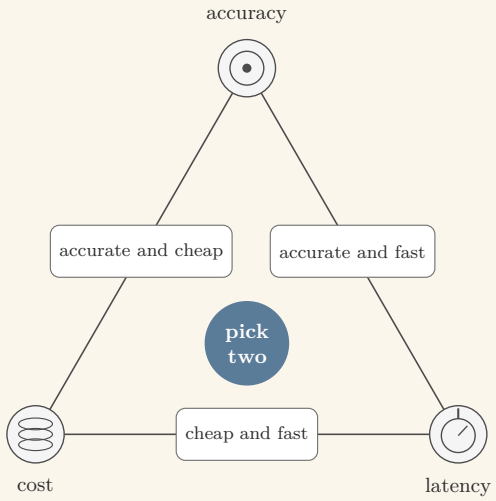
- The **false linear chain**: calls in a row that depend on nothing
- Fewer, larger steps; reads **prefetched**, never writes
- Read the **percentiles**; the user who leaves is in the tail

Chapter 19 · “parallelism buys wall-clock speed and breadth, never efficiency.”

Quality is bought with tokens and time, the two things the other corners minimize

Pick two

- **Accurate and fast**, paying money: strong model, parallel candidates
- **Accurate and cheap**, paying time: cascades, batch, overnight loops
- **Cheap and fast**, paying accuracy: one cheap call with a *measured* error rate
- Before the trade: remove the **waste**, which improves all three



‹ 4 / 4 ›

Figure from Chapter 19

Chapter 19 · “accuracy, cost, and latency form a triangle, and you usually get to pick two” · “The yield work of this chapter is that free lunch.”

Set the trade where it belongs: per use case, not per company

Per feature, with numbers, on purpose

Which corner this feature sacrifices · how far it may slide · which reading on which meter triggers a rethink

- “Two points more accurate and three times slower” is a **position**: an upgrade overnight, a regression mid-call
- Cost, latency and steps tracked on the **same eval bank** as accuracy
- Prices fall; **watch consumption**, never just the bill

Chapter 19, “The Accuracy-Cost-Latency Triangle” · the card is the chapter’s three parts of a stated budget

Part 3 · Chapter 20 (full book)

Serving strangers

Predict before the next slide · one minute in pairs

An HTTP endpoint runs the agent loop and returns the answer. A user asks it to audit twenty supplier contracts: four minutes of work. A load balancer closes any connection that is quiet for about thirty seconds.

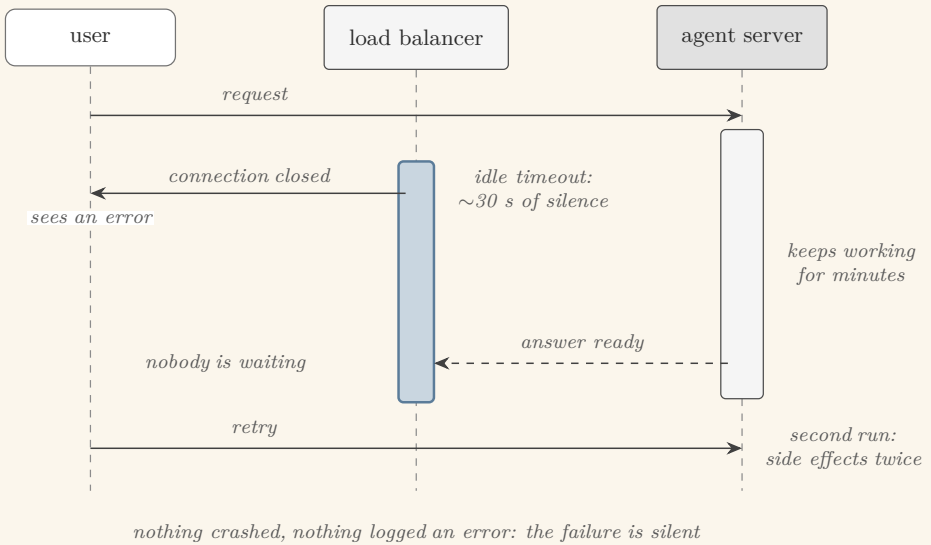
What does the user see? What does the agent do? What happens when the user clicks retry?

Chapter 20, “Serving Shapes” · the chapter’s own three questions

Nothing crashed, nothing logged an error,
every part did as designed

The thirty-second wall

- The user sees a **failure**
- The agent **keeps working**: four minutes of tokens for an answer with nowhere to go
- The retry starts a **second agent**; side effects happen twice
- Raising timeouts only **moves** the wall



A retry starts a second run; nothing crashed or logged an error, so the failure is silent.

‹ 5 / 5 ›

Figure from Chapter 20

Chapter 20, “Serving Shapes” · “The exact number varies by stack; the wall does not.”

Take a number

- Submit → **202** and a **task id** at once; a worker does the job
- **Status**: pending → working → completed | failed | cancelled; terminal never changes
- **Progress** and **cancel**
- An **idempotency key** on submission: a repeat returns the same ticket
- **Poll** for truth; a webhook only saves time
- Under ~10 s sync · to ~1 h async · beyond, or a person mid-run: durable

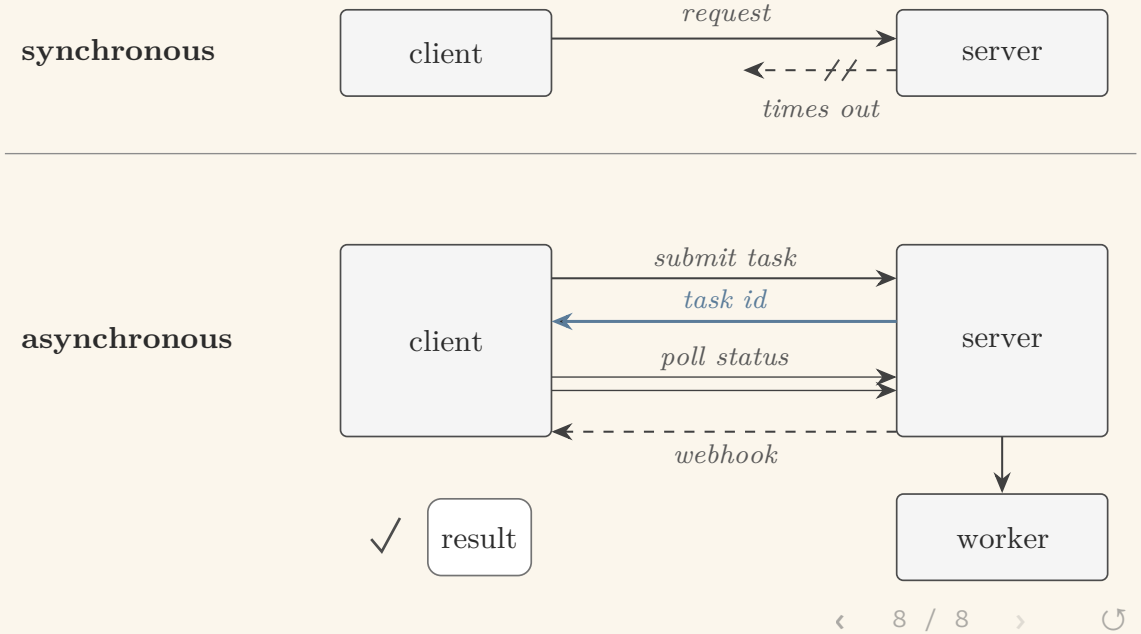
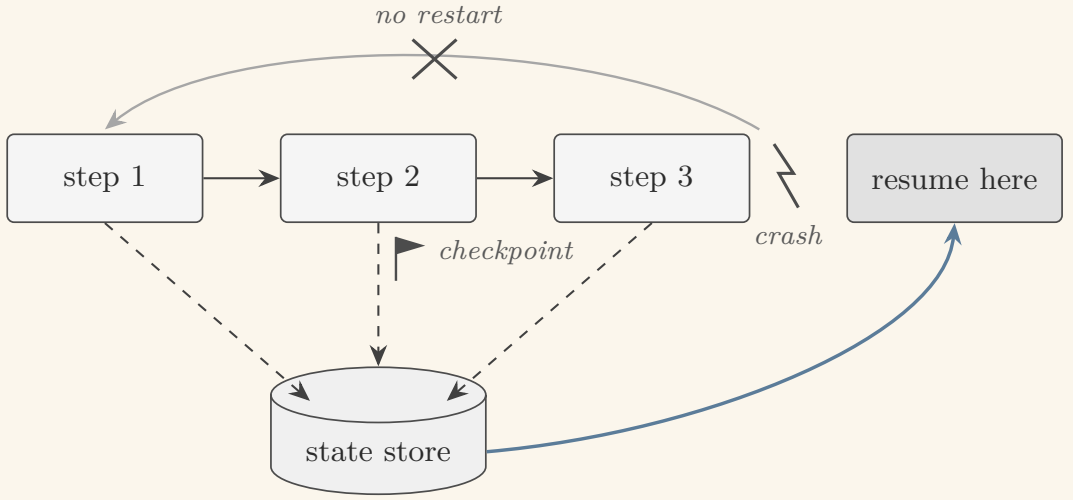


Figure from Chapter 20

Chapter 20 · “treat polling as the source of truth and webhooks as an optimization.” · the duration thresholds are the chapter’s illustrations

A row, not a process

- Deploys, autoscalers and cheap machines kill workers **by design**
- In-memory run, dead worker: the client polls a task that no longer exists
- A tasks table: **id · key · status · checkpoint · result**
- Session state is run state
- Engines: replay without **re-buying inference**; a parked run holds **no compute**



Instead the run reloads the last checkpoint and continues where it stopped.

‹ 5 / 5 ›

Figure from Chapter 20 (not on the syllabus list)

Chapter 20 · “Oversight scales only if waiting is free.” · “The moment you catch yourself building the machine is the right moment to buy it.”

One sandbox is a security decision; ten thousand a day are an infrastructure product

Rooms by the thousand

- An **orchestrator** and separate **worker pools**, scaled on their own
- A **warm pool**: rooms booted before demand
- A fresh room per session, **torn down** after; hard caps inside
- Isolation strength is boot time against wall thickness
- Strangers side by side: one room per **session and tenant**
- Egress **denied** by default; credentials per task

Chapter 20, “Sandboxed Execution at Scale” · “anything that survives teardown is a message from one session to the next.”

Predict before the next slides · one minute in pairs

Launch week. Traffic triples. The provider starts refusing calls: 429 Too Many Requests.

Every call in your stack retries transient failures, as week 10 taught. What happens next? What should your own front door do?

Chapter 20, “Rate Limits, Queuing, and Backpressure” · the chapter’s own question

At one run the limits are invisible; at a thousand they are the main failure

Two axes, one division

The ceiling, in runs per minute

tokens per minute ÷ tokens per run

770,000 ÷ 77,000 = **10**

- Limits on **requests** and on **tokens** per minute; agents hit tokens first
- Retries on a 429 rejoin the overload, **together**
- Jitter spreads them; only **admitting less** cures it
- Downstream tool limits stay **inside the tool**

Chapter 20 · “The machinery that made a single run resilient is now amplifying a fleet’s failure” · 770,000 and 77,000: the queue simulator’s made-up round numbers

When downstream saturates, push back upstream

The door

- A **queue** absorbs bursts; a **cap** meters work in flight
- **Admission policy**: past a chosen depth, refuse politely, with a retry-later signal
- **Lanes**: interactive first, background in the slack, the patient bulk to batch
- Caps per lane, tenant and agent
- Queue depth is the **alarm**

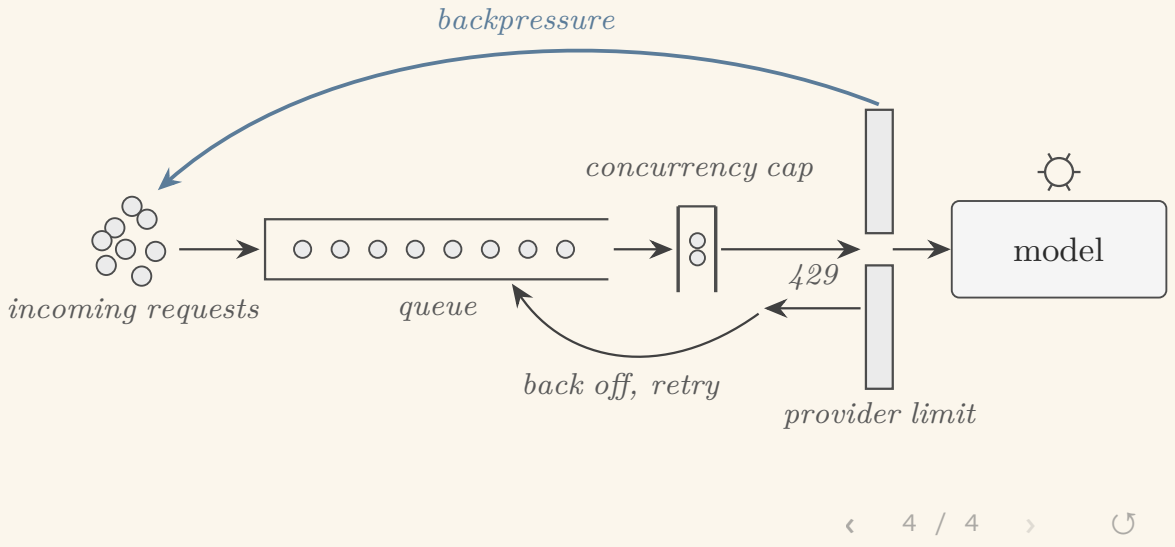


Figure from Chapter 20

Chapter 20 · “Refused at the door costs the caller a retry. Accepted and abandoned costs them their result, their patience, and, because the ticket eventually said failed with nothing to show, their trust.”

Part 4 · Chapter 20 (full book)

The rollout ladder

Optional reading in place of the explainer: Shadow Deploy an AI Agent, Then Canary

A composite of the field’s incident reports

Tuesday’s edit

`Tue: “be more concise”` → `suite green, merged` → `an hour of calm` →

`Thu: tickets up a third`

Behavior fails	so merge, deploy and watch misses it
Diffusely	bad answers, not crashes: nothing pages
Late	a ticket days later, never an error at deploy
Variably	the same input differs run to run; one green run proves little

Chapter 20, “The Rollout Ladder” · the chapter says it composed the scene from many reports

Each rung buys confidence with a larger slice of reality

Five rungs, a way down from each

- **Eval gate**: the suite decides the merge
- **Shadow**: the candidate runs on real traffic; users see only the incumbent; a judge compares
- **Canary**: a small slice of real users
- **Ramp**: widen while the numbers hold
- **All traffic**
- Shadow costs **double**: use it for changes with reach

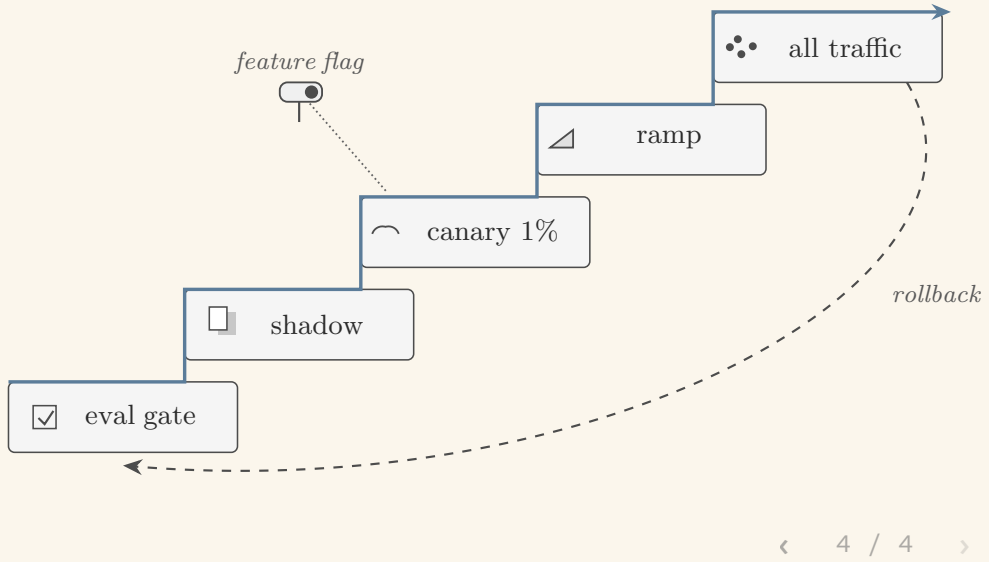


Figure from Chapter 20

Chapter 20 · “the gate tests the failures you already imagined. Every rung above it exists because reality imagines better than you do.”

Two disciplines separate a canary from a small gamble

The canary

Assign per session

A user who lands on the new version stays there for the session. Per request, one conversation mixes two behaviors.

Roll back automatically

Thresholds chosen in advance, per cohort; a breach flips traffic back with **no person in the path**.

- Errors and refusals · latency percentiles · cost per request · output length at both ends · thumbs, regenerations, abandonment

Chapter 20 · “rollout incidents surface at night, and nobody makes good judgment calls at 2 a.m.” · slices of a percent, or a tenth of one: the chapter’s illustrations

A flag and one artifact

Feature flag

Which version serves which slice is **configuration**: promotion and rollback are flips, the ramp is edits to a number.

Immutable artifact

One version over **code · prompt · model and settings · tool schemas**. Change = a new version; nothing edited in place.

- An incident names **one** version; a behavior change is a **diff**; rollback points at the previous one

Chapter 20 · “version the compound as one immutable artifact, and change version one by shipping version one-point-one, never by editing in place.”

In class · teams of two or three · 26 minutes · four browser tools

Price it, cache it, admit it, ship it

1. **A · the bill (8 min).** The cost estimator on the order desk, then on your project's trace
2. **B · the cache (5 min).** Prompt caching savings: move the timestamp; a proposal to trim the tools
3. **C · the door (6 min).** The queue simulator: the ceiling, a rush, two policies, lanes
4. **D · the ladder (5 min).** The rollout plan generator for a model swap

Every link on the lecture page opens prefilled. Prices and limits are illustrations. Debrief, 2 minutes

Tools after Chapters 19 and 20 · each tool's model and readings are its own, labeled on the tool · the order desk is invented for the exercise

Homework, and reading

The meters · no lab this week

- Cost per run, **latency percentiles**, step counts, from your traces
- F and g measured; the step where history overtakes the prompt
- A **step budget** from the distribution, plus a margin
- The **triangle statement**, per feature

Reading for week 12

- Ch. 21, Coding Agents `full book`
- Ch. 22, The Coding Workflow in Practice `full book`
- Ch. 26, Agents as Classifiers and Scorers `full book`

This week’s reading: Ch. 19 and Ch. 20 (full book) · glossary: App. B (free) · no paid API needed

Five things to keep

- Input is **F·n + g·n(n−1)/2**; know where your history overtakes your prompt
- Stable first, variable last; cheap clutter is **still clutter**
- Split the clock: **first token**, decode, tools, queue; then pick the lever
- Take a number: a **ticket**, a key, a row, and a door that says no
- Climb the ladder; roll back by **flag**, to an artifact nobody edited

Engineering AI Agents · Week 11

Trust the shape, never the figure.

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/

