SCALE WITHOUT SURPRISE

Visual Companion

Understanding Cost, Capacity, and Capability

Timothy O'Brien

A shorter route into the book

The main ideas, selected explanations, practical decisions, and chapter diagrams. Follow the links when you need the full argument, worked examples, and qualifications.

Draft edition. Preliminary chapters remain preliminary here.

https://withoutsurprise.com/briefs

Copyright © 2026 Timothy O'Brien. All rights reserved.

Contents

What Computing Costs 3

Capacity Is a Decision 11

The Players and the Vocabulary 14

The Simplest Practice 17

Knowing Your Workloads 20

Built-In Tools 23

AI Services and Token Capacity 26

Measuring AI Workloads 34

Orchestration and Agentic Topologies 41

Specialized Hardware 44

Exotic Hardware: Quantum and Emerging Capacity 48

Databases and Data-Bound Technologies 59

Measuring Value 62

CHAPTER 1 / 4 MINUTE BRIEF

What Computing Costs

Read the full chapter → / Read this brief online →

The main idea

Go back to the case from the introduction: a demonstration has turned into a launch, the service can absorb the new work only while nothing fails, and the additional capacity won't arrive in time. One of Finance's first questions is why the resources we already pay for can't do more of the work. It's a fair question. Before we can answer it, or ask anyone to approve a purchase, we need to say what the organization pays for today, what it has already promised to keep paying, and what would actually change if we did something different.

Explore this section in the full chapter →

Knowing what you consume before you change it

Sometimes an application will stop using resources the organization has already committed to paying for. Releasing them may let another team use them or avoid a purchase later, but it does not cancel the payment. Every proposed change has to be followed through to the obligation before anyone calls it a reduction.

Explore this section in the full chapter →

Several dimensions of the same dollar

Fixed, variable, committed, metered: these are useful words, but trouble starts when we treat them as four separate boxes and assume that placing an expense in one tells us how to manage it. A single cloud product might charge one component as fixed overhead and another by usage. Consumption can be metered whether it is covered by a commitment or billed at an on-demand rate. A service whose usage varies can still carry a minimum payment. The label alone does not tell us how the charge will respond to a change in use.

Explore this section in the full chapter →

Then there is **attribution**, which connects an expense to the workload or decision that caused it. The easy version is to tag resources to the teams that use them and charge those teams, but many costs are shared. Some organizations spread them evenly across the company and call it a *peanut butter spread*; management accountants know it as peanut-butter costing and warn that it overcharges light users and undercharges heavy ones.[1] Other shared costs go to the groups that use a particular service, and some are simply overhead. Attribution also gets political, because who pays is a question about who is answerable. It tells us who is consuming resources and where to start a conversation, not what the enterprise would save if that consumer went away, and shifting a cost between teams is surprisingly easy to mistake for reducing it.

Explore this section in the full chapter →

[1] Charles T. Horngren, Srikant M. Datar, and Madhav V. Rajan, *Cost Accounting: A Managerial Emphasis*, 15th ed. (Pearson, 2015), chapter 5, "Activity-Based Costing and Activity-Based Management": broad averaging, or peanut-butter costing, spreads resource costs uniformly across products or services that use those resources non-uniformly, overcosting some and undercosting others. The application to shared cloud costs is the author's.

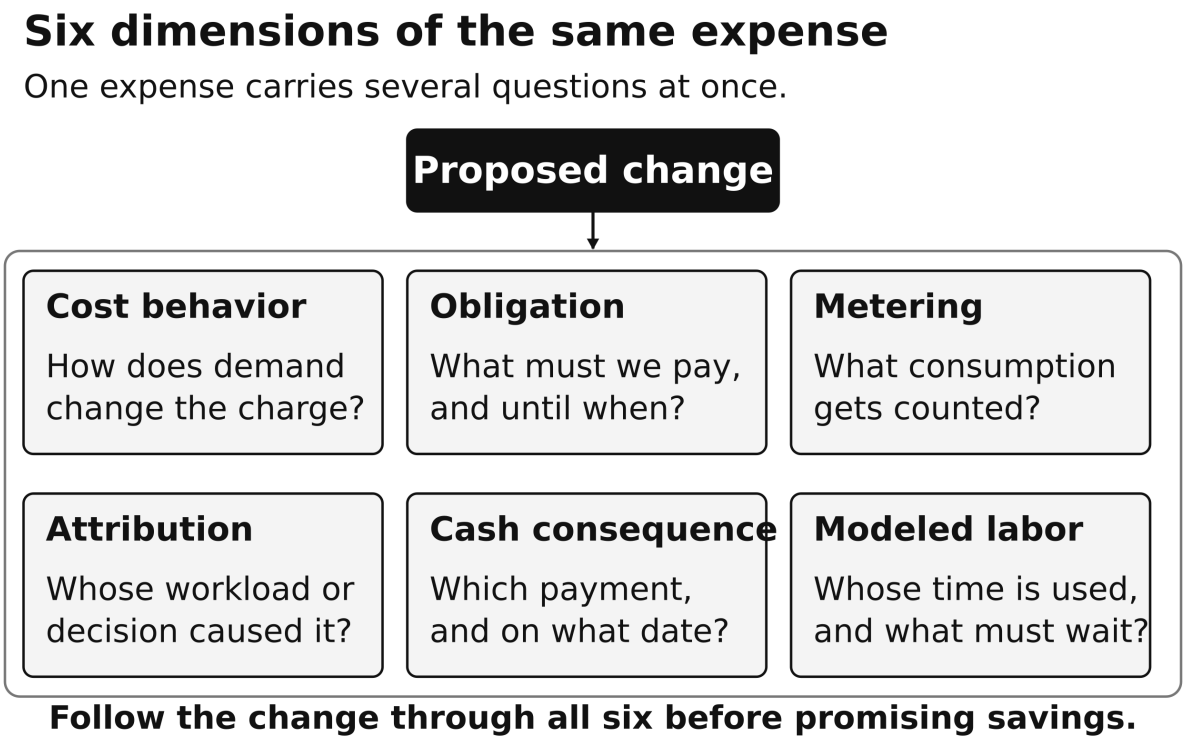


Diagram 1.1. Every expense carries overlapping dimensions: how its charge behaves, what is owed, what is counted, who caused it, which payment changes, and whose time is consumed. These questions must be answered together.

Explore this section in the full chapter →

A small number of teams dominate cost

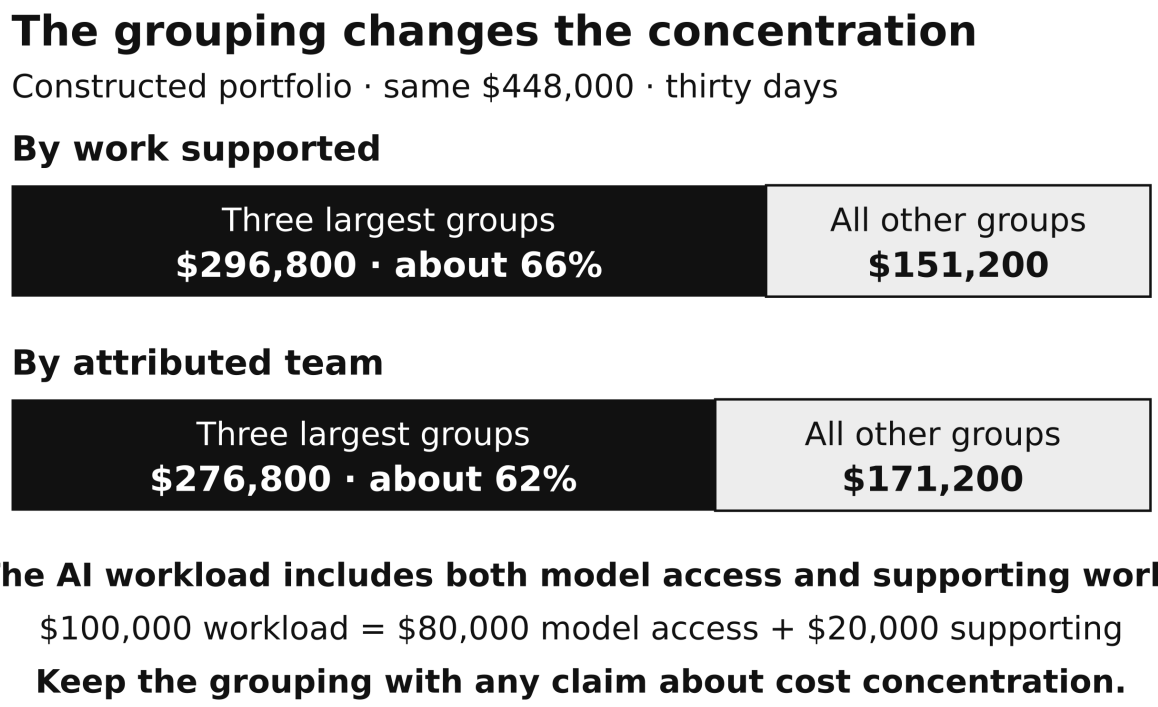


Diagram 1.2. The three largest workloads account for $296,800 of the $448,000 bill, about 66 percent; the three largest attributed teams account for $276,800, about 62 percent. The difference is the AI workload's supporting work, which belongs to other teams. Keep the grouping with the concentration percentage.

Explore this section in the full chapter →

How the charges behave

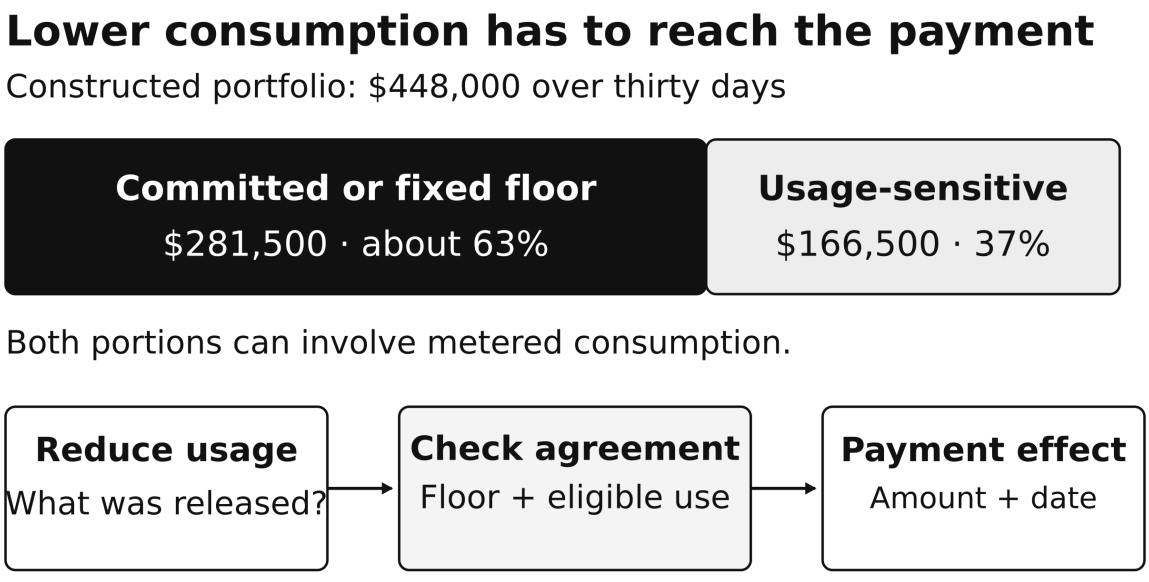


Diagram 1.3. In the constructed thirty-day portfolio, $281,500 is a committed or fixed floor and $166,500 is usage-sensitive. Reducing consumption must be followed through the agreement before it becomes a payment reduction; seasonality is a separate property of demand.

Explore this section in the full chapter →

Follow a change across the boundary

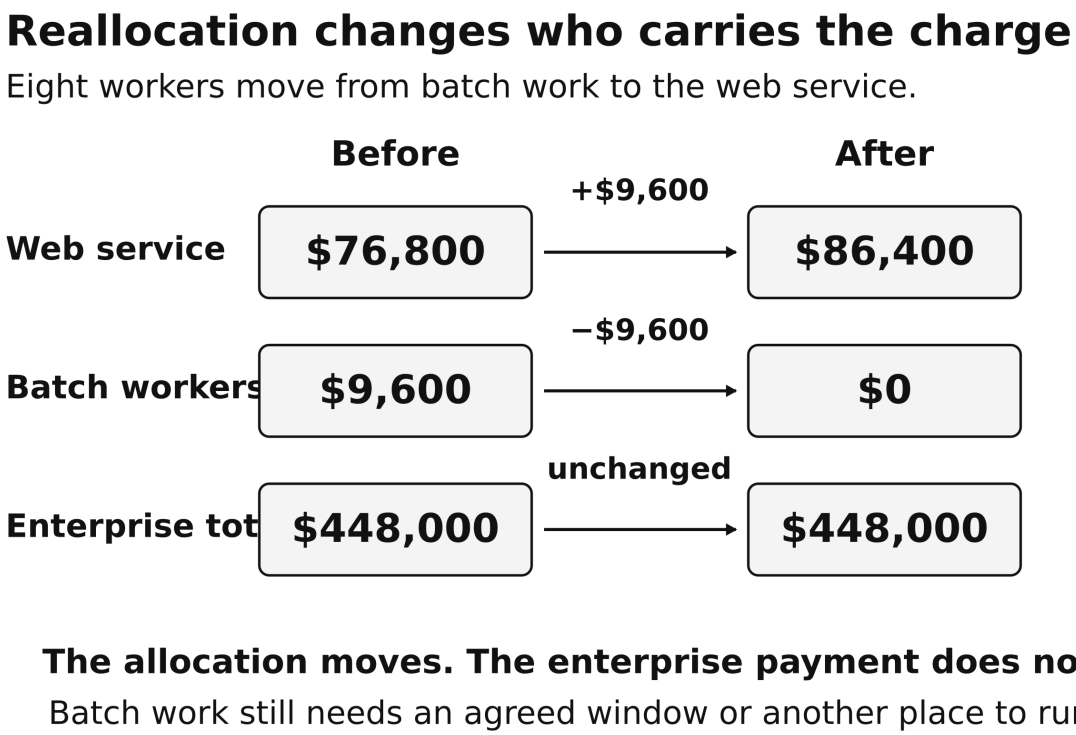


Diagram 1.4. Moving eight batch workers changes attribution: the web service rises by $9,600, the batch allocation falls by $9,600, and the enterprise total stays $448,000. The batch work and any funding transfer still need separate agreement.

Explore this section in the full chapter →

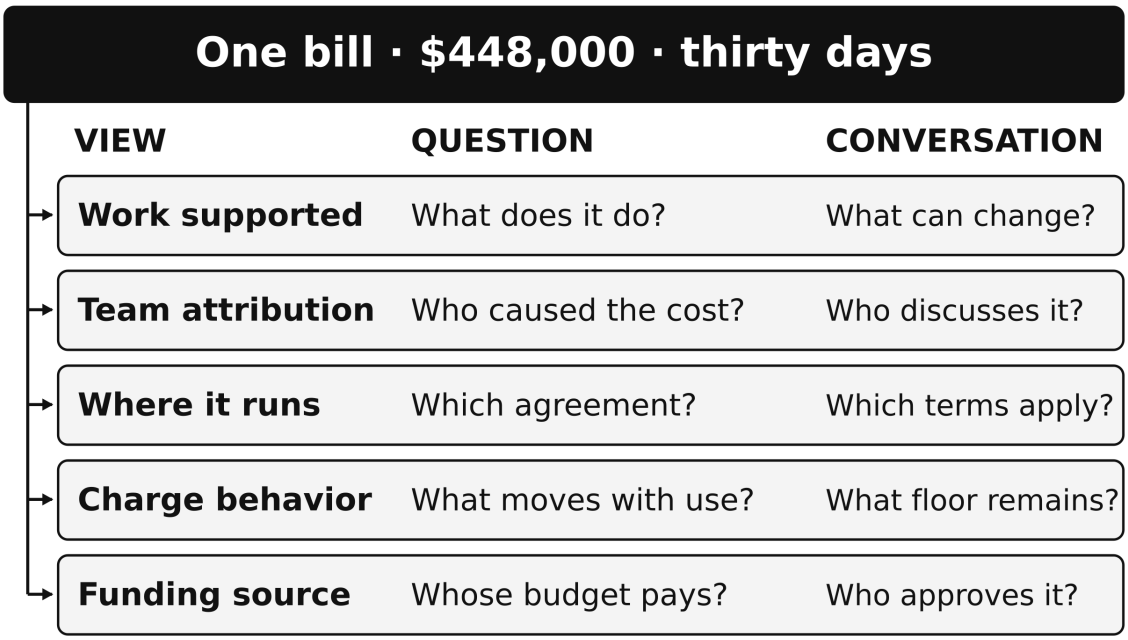
The same bill, five ways

The same $448,000 has now appeared as five tables, each because a different conversation needs it. The work view says what could change and what the change would disturb; the attribution view, whom to talk to; the location view, which agreement governs the change; the charge view, how much of the bill is floor; and the funding view, whose budget moves and whether its owner has agreed.

Explore this section in the full chapter →

One bill, five conversations

Each view groups the same money; never add these totals.



A proposal needs all five conversations about the same expense.

Diagram 1.5. The same $448,000 over thirty days grouped five ways: by work supported, team attribution, where it runs, charge type, and funding source. Each view answers a different question about the same money.

Explore this section in the full chapter →

Enough information to have the next conversation

I would begin the cost record with the work being supported, its owner, the period, the amount, the decision under consideration, and whether each charge is direct, shared, or still unresolved, then follow the change through every dimension. For this chapter's two changes, it would start like this:

Dimension	Eight borrowed batch workers	Sixteen new workers
Amount	$9,600 per thirty days moves to the web service; total unchanged	$19,200 per thirty days added, before any commercial offset
Cost behavior	Usage-sensitive; the charge continues because the workers stay in use	Usage-sensitive unless bought under a commitment
Obligation	None new; the agreement with the batch owner sets the return date	None until bought; a one- or three-year commitment would outlast the launch
Metering	Worker-days, unchanged	Worker-days from provisioning, which precedes qualified service
Attribution and funding	Web service; funding moves only if both budget owners agree	Web service; website product line, if it agrees
Cash	No additional payment	Set by the agreement, effective date, and invoice terms
Modeled labor	Qualifying the moved workers; the batch catch-up	Qualifying the new supply
Still open	The cost of the batch delay: unpriced, not zero	Lead time to qualified service

Keep the underlying records available, because the first version will contain questions that somebody else can answer better than we can.

Explore this section in the full chapter →

CHAPTER 2 / 3 MINUTE BRIEF

Capacity Is a Decision

Read the full chapter → / Read this brief online →

The main idea

Anytime I hear “yeah, we’ve got enough capacity,” I want to ask what they mean. Enough for which operating mode? A complex system runs under steady load, through a seasonal swing, and at the peak that arrives once a year. When someone who runs one breezily tells me everything is fine, no problems, I start wondering whether they have ever experienced a failure, because the honest answer to “do you have enough capacity” is a complex one. Something like: for the work in front of us right now, yes; here is the evidence for that; here is the traffic we expect over the next 30 to 90 days; and here is what we are watching so we can prepare before it becomes a problem. No measurement can give that answer by itself, because it depends on what the organization has decided to promise, who is entitled to rely on the promise, and who answers for keeping it.

Explore this section in the full chapter →

Headroom has a purpose

Headroom is often described as the difference between a measured limit and expected demand, but that leaves out the question of which limit we mean. In our original configuration, normal capacity of 16,000 minus expected demand of 10,000 leaves 6,000 requests per second. Under one-group loss, the calculation is 12,000 minus 10,000, leaving 2,000. These describe different states of the same service, not two reserves we can add together.

Some of that room is carrying more than one responsibility in the plan: growth while everything works, and service preserved when something fails. The healthy service's 6,000 looks like room for the rollout's 4,000 and also for the 4,000 that a lost group takes with it, but it cannot be both, because together they need 8,000. Test the combined requirement in the condition you intend to promise. Counting each responsibility separately and then celebrating the apparent total can make us feel considerably better prepared than we are.

Explore this section in the full chapter →

Autoscaling still needs someone to supply the capacity

I like automation that responds to changing demand. What worries me is the point at which the word autoscaling starts doing work that belongs in a supply plan. An autoscaler can ask for resources when a signal changes; it cannot, by that request alone, establish that the right resources will be available, ready, and connected to their dependencies in time. Azure's quota guidance makes the distinction explicit: quota is a subscription's permission to deploy, capacity is the compute actually available in a region or zone, and a deployment with enough quota can still fail for lack of capacity.[2] That leaves two questions to settle before the autoscaler needs an answer — will we be allowed, and will the supply be there — with options such as qualifying another size or region, or securing an eligible capacity reservation where the workload requires it.

Explore this section in the full chapter →

Make the promise understandable

The person approving a launch should be able to explain what the customer will receive and what has to remain true for us to deliver it. If that explanation requires reconstructing a month of technical discussions, we have more work to do. I would give them a short account of the useful work, quality, deadline, operating condition, and usable supply, with named owners and the underlying evidence available to whoever needs to challenge it.

[2]Microsoft Learn, "Check vCPU quotas", checked October 3, 2026. Defines quota as a subscription's permission to deploy and capacity as the compute available in a region or zone, says a deployment with sufficient quota can still fail for lack of capacity, and suggests a different size, zone, or region, or an on-demand capacity reservation.

For our web service, the account might run like this. At the tested request mix, the service carries the expected 10,000-request peak with 2,000 to spare after losing one of its four sixteen-worker groups. The full rollout, at 14,000, would leave it 2,000 short in that condition. A staged rollout reaches 12,000 and uses all of the margin, so it also needs eight batch workers borrowed for thirty days, if the batch owner accepts the delay and the workers pass qualification. Sixteen new workers would carry the full rollout once funded, but they are estimated to be six weeks from qualified readiness, and the borrowed workers are due back before then.

Explore this section in the full chapter →

CHAPTER 3 / 4 MINUTE BRIEF

The Players and the Vocabulary

Read the full chapter → / Read this brief online →

The main idea

As an engineer, I can learn the difference between a capital expense and an operating expense, and I can put together a comparison that explains why changing one into the other looks attractive. Then I talk to someone in Finance and discover how much of the question I have left out. The timing matters, the planning process matters, the person who can approve the commitment matters. "It depends" begins to sound less like an obstacle and more like the answer I should have expected.[3]

Explore this section in the full chapter →

The nouns that change a meeting

A **forecast** estimates future demand under stated assumptions. Give it an as-of date, because the information available at the time is part of what makes the estimate interpretable. Revising it after a customer changes plans may be exactly the right thing to do; the revision is a new forecast, not evidence that the old one was accurate. Check the unit, too: in a Finance meeting "the forecast" usually means expected spending for the year, and in a capacity meeting, demand in requests, cores, or tokens.

A **target** is something the organization intends to achieve. A sales target may be deliberately ambitious to encourage people to pursue an opportunity, while the most likely estimate of realized demand is lower. Capacity needs to examine the target as a scenario without silently converting ambition into a prediction.

A **budget** is an approved plan for spending: an amount, for a period, held by someone accountable for it, usually set months before the demand it funds. It doesn't approve a particular purchase. Who can approve that kind of spending, and whether the amount

[3] Tim O'Brien, "It's Not That Simple", Medium, June 21, 2026. The opening passage adapts the author's description of recurring conversations with Finance.

crosses a threshold that sends it through a different process, are separate questions, and often what Finance means by "it depends." Whether the money will support the service's obligations is an engineering and business question, and a discrepancy between the two needs a conversation rather than a revised label.

A **request** says what someone wants and starts an investigation or a negotiation. It doesn't establish that the resources exist, that funding has been approved, or that another customer will give up an allocation, which sounds obvious until a requested amount starts appearing in other people's plans as confirmed supply. Requests also lose their reasons on the way: the SRE book's account of Auxon, a Google planning tool, describes requests arriving as "X cores in cluster Y," their degrees of freedom "long lost."[4] In our launch, the degrees of freedom were most of the answer. A quota increase says no more about supply: it raises the limit on what we may request without putting any machines behind it.

A **reservation** needs to be understood in its particular context. It might be a provider product, an internal hold, or a position in a planning queue, and a provider's reservation may buy a discount, a hold on capacity, or both. I want to know what has been reserved, for whom, where, for which dates, and with what release rule before I use it to support a customer promise.

A **commitment** is a promise with defined parties and conditions. In Volume II, Chapter 20 we will separate a promise to spend from a promise to supply usable capacity, because each can exist without the other. An optimistic forecast in a presentation establishes neither.

One more noun is not a planning word but a product word, and meetings trip over it as often as the others: the **SKU**. The term is stock-keeping unit, borrowed from retail, where it names one distinct item a store sells — one product, in one configuration, that someone can order by a code. The cloud providers borrowed it for their catalogs, and in this book a SKU is one sellable configuration of a service: one size, one feature set, one price. Every line on the bill in Chapter 1 names a SKU.

[4] Beyer et al., *Site Reliability Engineering*, Chapter 18, "Software Engineering in SRE", quoted from the Auxon case study. Auxon was an internal Google tool; the connection to this chapter's launch is the author's.

The name is worth reading slowly, because providers encode the machine into it. In Microsoft's convention, `Standard_D4s_v5` decodes as a general-purpose family (D), four vCPUs, premium-storage capable (the s), fifth version (v5), and the encoded extras run to local disks, network bandwidth, memory ratios, processors, and, for the accelerated families, the accelerator model itself: the GPU size Chapter 10 works with, `Standard_ND96isr_H200_v5`, announces the H200 in its own name.[5] Other providers publish naming conventions of their own, and the skill of reading them transfers even where the grammar differs.

Explore this section in the full chapter →

Who can change the promise?

For each condition the plan relies on, I want to know who supplied it, who can check it, who can change it, and who else needs to hear when it changes. For our constructed launch, a first version might look like this:

Condition	Supplied by	Checked by	Can change it	Must also hear
First stage adds 2,000 requests/s	Product lead	Engineering and operations	Product and the business owner	Operations, Finance
Batch work can wait while eight workers move	Batch owner	Batch owner, against a catch-up plan	Batch owner	Product, operations
Moved workers ready after two days' testing	Operations	Operations, by testing	Operations	Product
Sixteen new workers in about six weeks	Supplier, via Procurement	Operations, at qualification	Supplier and operations	Product, batch owner
Money for the new workers	Budget owner	Finance	Budget owner, within a delegated limit	Product, Procurement
Service survives loss of one failure group	Business owner	SRE, by testing	Responsible business executive	Everyone relying on it

Explore this section in the full chapter →

[5] Microsoft Learn, "Azure virtual machine sizes naming conventions", checked October 6, 2026. Field order (family, sub-family, vCPUs, additive features, accelerator, version), the additive-feature letters, and the rule that only specialized SKUs launched from the third quarter of 2020 carry the accelerator in the name. The `Standard_D4s_v5` reading follows the page's conventions and examples.

CHAPTER 4 / 3 MINUTE BRIEF

The Simplest Practice

Read the full chapter → / Read this brief online →

The main idea

If you're starting a capacity and cost practice, I would resist the temptation to begin by designing the process for the entire organization. You'll need to understand that organization eventually, and it will need more than one approach. But there's a more useful place to begin: find a decision somebody needs to make and stay involved long enough to see what it takes to make it well.

You might start with a service that's about to grow. The engineering team can explain what it runs today, somebody in the business has a plan that will change the demand, and Finance wants to understand the cost of supporting that plan. Getting those people into the same discussion sounds straightforward until you discover that the growth estimate assumes a launch date, the launch date assumes available capacity, and the capacity depends on a purchasing decision that hasn't been made. Now you have something worth working on.

Explore this section in the full chapter →

A portfolio small enough to follow

For the first decision, we'll concentrate on the web service and the measurement Chapter 2 gave it. Its operators have tested a particular request mix at the configurations we're considering: one worker can complete 250 requests per second while holding ninety-fifth-percentile (p95) latency to 200 milliseconds and errors to no more than 0.1 percent of offered requests. That evidence exists because we've assumed it. In a real discussion, I'd want the operators to explain how they qualified the rate and what change in request mix, configuration, or downstream behavior would make it unreliable.

The service has 64 workers arranged in four independent failure groups of 16, providing a tested 16,000 requests per second when everything is available. The business has

also agreed that the service must continue after losing any one group. That requirement changes the capacity we can promise:

$$(64 - 16) \times 250 = 12{,}000 \text{ requests/second.}$$

Twelve thousand is the limit I'd use for this decision; as Chapter 2 showed, sixteen thousand answers a question about normal operation. Even the lower number covers only the loss of one independent group; we haven't established what happens when an entire region fails.

Explore this section in the full chapter →

Borrowed capacity comes with a return date

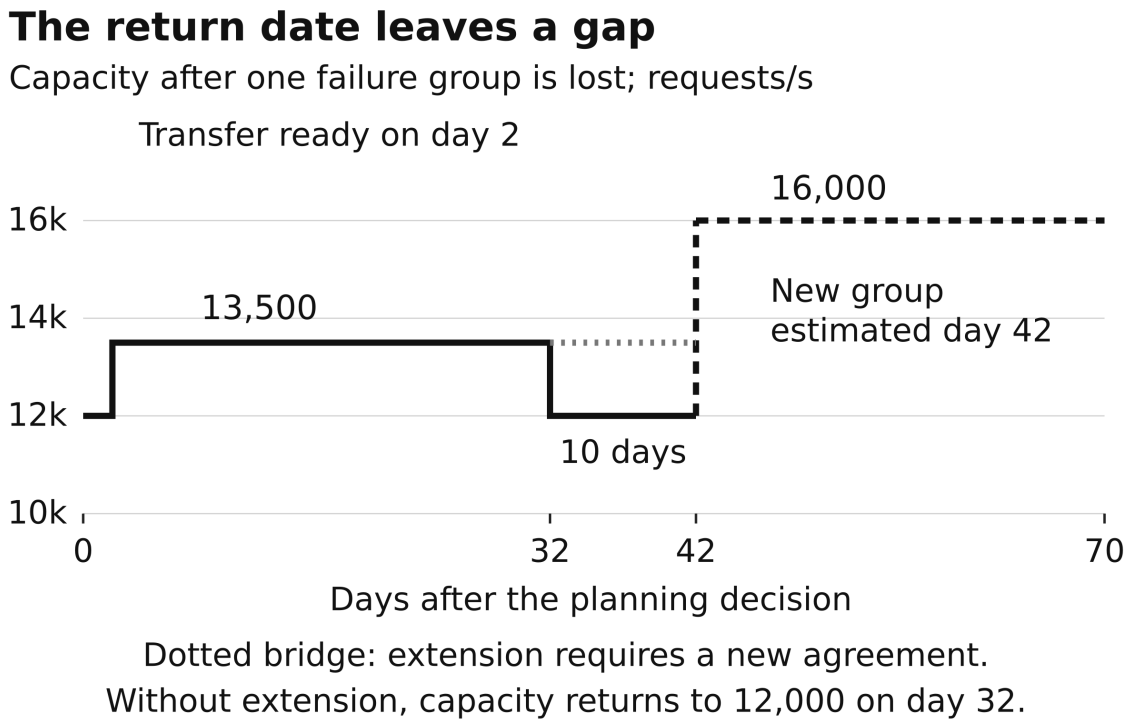


Diagram 4.1. The temporary transfer expires on day 32; the new group's qualified readiness is only estimated for day 42. The dotted bridge requires a further batch agreement.

Explore this section in the full chapter →

Make the decision legible

For this proposal to proceed, five of the people in Chapter 3's authority map have to commit to something:

Who	Commitment	Needed before
Product lead	Stage the rollout at the estimated 2,000 additional requests/s, and keep the ability to stop expansion	the rollout
Batch owner	Accept a bounded delay with a catch-up plan	the transfer
Web operations lead	Qualify the transferred workers, including their behavior when a group fails	the rollout
Budget owner	Authorize the $640-a-day purchase within a delegated limit, contingent on supply and the actual terms	the purchase
Responsible business executive	Accept the remaining exposure, or change the launch scope or date	the rollout

Several may be settled in one conversation. They're still five commitments, and the decision record should make a missing one obvious. If batch refuses the delay, we no longer have this option. If the business refuses staging, 14,000 still exceeds 13,500, and changing the failure requirement to make the arithmetic pass would mean offering a different service. We may need to discuss that possibility, but with the person entitled to change the obligation and with its consequence understood.

The agreement also needs a way to reopen the discussion before the next scheduled review. Demand running above the staged estimate, a changed request mix, a larger launch, an infeasible catch-up plan, or delayed delivery can each remove a condition we relied on. Waiting for the monthly meeting would only be sensible if we could afford to wait. Whoever learns about the change should know whom to contact and what decision it may affect.

Explore this section in the full chapter →

CHAPTER 5 / 3 MINUTE BRIEF

Knowing Your Workloads

Read the full chapter → / Read this brief online →

The main idea

Give me a report showing that a machine uses a fifth of its CPU and I can make a plausible argument for shrinking it. Tell me that the machine runs a critical database and I can make an equally plausible argument for leaving it alone. Neither argument requires much knowledge of the database, which is precisely what should make us uncomfortable.

The interesting work begins when we try to explain why that fifth of the CPU is all the system uses. Perhaps nobody needs the machine anymore. Perhaps the application is waiting on a lock, or the memory is keeping an expensive set of reads away from storage. Perhaps the person who called it critical knows something about next month's work that has never appeared in our measurements. I want to understand those possibilities before we turn either the low utilization or the word *critical* into an answer.

Explore this section in the full chapter →

Look at utilization, waiting, and errors

Gregg's USE method—utilization, saturation, and errors—gives resource investigation a disciplined start: for every resource, check all three. He defines utilization as the average time a resource was busy servicing work, saturation as the degree to which it has extra work it can't service, often queued, and errors as the count of error events.[6] The averaging matters. Gregg describes a customer whose CPUs were saturated although monitoring, reporting five-minute averages, never showed them above 80 percent; they were hitting 100 percent for seconds at a time. Relating busy, queued, and failing work tells us far more than sorting a spreadsheet of CPU percentages from smallest to largest.

Explore this section in the full chapter →

Test the arrivals you expect to receive

The workload generator deserves the same skepticism as the system under test. A reassuring latency result means little if the test quietly stopped sending work whenever the service became slow, and many generators do exactly that by default: each simulated client waits for a response before sending its next request.

Gil Tene calls the resulting measurement error coordinated omission: the generator stops sending the requests that would have suffered most.[7] In his `wrk2` documentation, a 30-second run aimed at 2,000 requests per second over 100 connections included one 1.4-second server pause. Measured from when each request should have been sent, the 99th percentile was 1.27 seconds; measured from when each was actually sent, 6.04 milliseconds. About 2,800 requests were due during the pause, 7 percent of the 39,500 recorded, but 100 waiting connections could record at most 100 slow responses, too few to reach the 99th percentile.

Explore this section in the full chapter →

[6] Brendan Gregg, *The USE Method*, https://www.brendangregg.com/usemethod.html, accessed October 4, 2026. Gregg also notes a capacity-based definition of utilization; the method sets no universal utilization target.

[7] Gil Tene, "How Not to Measure Latency," QCon San Francisco 2012, https://qconsf.com/sf2012/dl/qcon-sanfran-2012/slides/GilTene_HowNotToMeasureLatency.pdf, accessed October 3, 2026; `wrk2` README, https://github.com/giltene/wrk2, accessed October 4, 2026. The percentiles and the 39,500 requests recorded after a 10-second calibration period are from the README's second example; the other request counts are this chapter's arithmetic. `wrk2` is an HTTP tool.

A record someone can challenge

By now we can keep three kinds of knowledge apart instead of blending them into one confidence score: measurements from a known interval and configuration, experimental claims bounded by what was tested, and business judgments about demand, deadlines, and acceptable consequences. For the web service, a short evidence record might read:

Claim	Evidence	What would change it
250 requests/second per worker at p95 ≤ 200 ms, errors ≤ 0.1%	Experiment: tested mix, each configuration used	A different mix; a generator that could not hold its rate
Next peak of 10,000 requests/second	Measurement and forecast: 16 weeks of daily one-minute peaks of offered requests	Peak demand rejected or queued; sub-minute bursts
Staging adds 2,000 requests/second	Business judgment: product lead, dated, staged scope only	Changed scope or customers; the first stage's peak
13,500 requests/second after losing a group	Arithmetic: 54 × 250, linear within tested configurations	Per-request demand on the limit rising over 12.5%
The shared database is not the limit at 13,500	None yet	A database test at the new mix, before launch

A missing metric must not become zero during an export, and a successful experiment on one workload must not silently become permission to shrink every machine of the same type. Contrary runs and failed tests belong in the record too. It is working if a specialist or customer can point to the line they think we have misunderstood.

Explore this section in the full chapter →

CHAPTER 6 / 3 MINUTE BRIEF

Built-In Tools

Read the full chapter → / Read this brief online →

The main idea

It is possible to open three perfectly useful tools and receive what look like contradictory instructions. A machine-sizing report recommends a smaller instance, a budget alert says the project is spending too much, and the autoscaler wants to create more instances. Before we schedule a meeting to determine which tool has failed, it would help to know what each one was asked.

The sizing report summarizes at least a week of one machine's use, the alert compares charges that typically arrive within a day with a monthly threshold, and the autoscaler responds to the last few minutes of CPU. All three could be right. The difficult part is deciding whether today's load is temporary, whether the proposed smaller machine can handle the work we have promised, and whether somebody has already agreed to pay for it.

Explore this section in the full chapter →

Keep inventory, use, and cost distinguishable

When I look at a service, I want to move among three views without losing track of what each tells me. Inventory shows what exists and where; telemetry shows what it did during an interval; billing shows what was charged under the commercial arrangement. A resource that disappeared yesterday may be absent from today's inventory and still account for a substantial part of this month's bill. A workload that stopped running may have left billed resources behind, while an apparently unused resource may still be needed for recovery.

The distinction seems obvious until someone joins the three views, drops the records that do not match, and presents the result as a complete account. We need the unmatched records precisely because they may explain what we have failed to understand.

Explore this section in the full chapter →

A recommendation is the beginning of an experiment

Before I accepted or dismissed one, I would want to connect the recommendation and its evidence date to the workload's next important event. Has the application changed? Is a migration about to end? Does the proposed smaller configuration affect a recovery obligation? I would also ask Finance what the reduction means under the commercial arrangement. Freeing capacity inside an existing commitment can be worthwhile even when it does not lower this month's cash expenditure.

For a stable stateless service, a representative subset may give us a straightforward experiment: agree on completion and latency criteria, make the change through the operating team's process, compare with the baseline, and preserve a feasible route back. A data service may need more patience, because ordinary traffic does not exercise the maintenance or recovery work that justified the allocation.

Suppose a smaller configuration serves requests successfully but cannot complete a required rebuild. We have discovered that the proposed saving changes the service being supplied. The business may be willing to accept that change, but I would not present it as an equivalent service for less money.

This is also why I want a useful explanation when a recommendation is deferred. "Awaiting a representative peak" gives us a next step, as does "retained for this recovery obligation" or "superseded by the planned migration." "Engineering refused" mostly tells me that we have run out of patience with each other. The capacity team's insistence deserves no more authority than Engineering's refusal unless somebody can explain the work.

Explore this section in the full chapter →

Alerts, caps, and the person who receives them

An Azure budget notifies people when actual or forecast cost crosses a threshold; in Microsoft's words, "Resources aren't affected, and your consumption isn't stopped." At subscription and resource-group scopes, an action group can pass the notification to automa-

tion such as a function or webhook, which needs its own design: which work may stop, who authorizes it, and how the service resumes.[8]

Cost data typically arrives 8 to 24 hours after use, budgets are evaluated every 24 hours, and email follows within about an hour, so an overrun can be nearly two days old when someone reads about it. An application that needs an immediate limit has to enforce it in its own request or admission path, reconcile it with billing, and decide what happens when the limit is reached.

Explore this section in the full chapter →

[8] Microsoft Learn, “Tutorial - Create and manage budgets”, checked October 4, 2026. Notification without stopping consumption; data and evaluation timing; actual cost including purchases, with usage-only filters; action groups at subscription and resource-group scopes.

CHAPTER 7 / 4 MINUTE BRIEF

AI Services and Token Capacity

Read the full chapter → / Read this brief online →

Before any theory: the four ways to buy tokens

There is more than one way to buy the same token, and the differences matter to a capacity plan.

The most direct path is the provider's own API, usually priced by consumption: so much per million input tokens, so much per million output tokens, with separate treatment for caching and other features. Payment may be prepaid, invoiced, or governed by a contract; a usage-based price does not establish when the money changes hands.[9][10]

The second path is through a platform. Microsoft Foundry, Google's Gemini Enterprise Agent Platform (formerly Vertex AI) and OpenRouter provide model access under their own deployment, routing and billing arrangements.[11][12][13] Before comparing rates for a

[9] OpenRouter, FAQ, https://openrouter.ai/docs/faq, and "API Credit & Rate Limits," https://openrouter.ai/docs/api_reference/limits, checked October 4, 2026; Stripe, "Stripe agrees to acquire OpenRouter to help businesses optimize token routing and usage," August 19, 2026, https://stripe.com/newsroom/news/stripe-agrees-to-acquire-openrouter. Product terms and transaction announcement; no transaction price is inferred.

[10] OpenAI, "GPT-6 Astra: A new generation of intelligence," September 3, 2026, https://openai.com/index/gpt-6-astra/. Launch Standard rates $10/$50 per million input/output tokens; Fast mode twice the Standard price and advertised at up to twice the speed. On OpenRouter, which reports weekly route traffic, Astra was among the most-used routes through early October 2026 and processed about 98.6 billion tokens on October 9, 2026, moving down the popularity list.

[11] Microsoft, "Understanding deployment types in Microsoft Foundry Models," https://learn.microsoft.com/en-us/azure/foundry/foundry-models/concepts/deployment-types, checked October 3, 2026. Deployment categories, supported processing boundaries and model-specific availability; managed-compute deployments are a separate option.

[12] Google Cloud, "Use Provisioned Throughput," https://docs.cloud.google.com/gemini-enterprise-agent-platform/models/provisioned-throughput/use-provisioned-throughput, checked October 4, 2026. Default spillover, the dedicated-only 429, enforcement windows and spillover hidden by one-minute averages. Renamed from Vertex AI on April 22, 2026, per the release notes: https://docs.cloud.google.com/gemini-enterprise-agent-platform/release-notes.

[13] OpenRouter, FAQ, https://openrouter.ai/docs/faq, and "API Credit & Rate Limits," https://openrouter.ai/docs/api_reference/limits, checked October 4, 2026; Stripe, "Stripe agrees to acquire OpenRouter to help businesses optimize token routing and usage," August 19, 2026,

named model, establish which version and features each offering supports, who supplies the inference, where processing may occur and which agreement pays for it. An existing cloud agreement can make this path operationally convenient; the workload still needs its own comparison.

The third path is a negotiated agreement. Large consumers may negotiate discounts, usage minimums, and model-specific terms. Negotiation does not guarantee a discount or held supply, and the price of the same work can depend on which agreement it flows through.

The fourth path is provisioned capacity, because a workload can hit a throughput limit while its monthly spending is modest. Providers sell arrangements that reserve a defined level of processing capability, subject to the model, region, workload and service terms, and the charge continues while that capacity is idle. A spending commitment alone reserves no capacity.[14][15]

Explore this section in the full chapter →

https://stripe.com/newsroom/news/stripe-agrees-to-acquire-openrouter. Product terms and transaction announcement; no transaction price is inferred.

[14]Microsoft, "Provisioned throughput billing and cost management," https://learn.microsoft.com/en-us/azure/foundry/openai/concepts/provisioned-throughput-billing, checked October 4, 2026. Deployed throughput incurs hourly charges whether or not it is used; a reservation is a discount on those charges and does not guarantee capacity.

[15]Google Cloud, "Use Provisioned Throughput," https://docs.cloud.google.com/gemini-enterprise-agent-platform/models/provisioned-throughput/use-provisioned-throughput, checked October 4, 2026. Default spillover, the dedicated-only 429, enforcement windows and spillover hidden by one-minute averages. Renamed from Vertex AI on April 22, 2026, per the release notes: https://docs.cloud.google.com/gemini-enterprise-agent-platform/release-notes.

Token capacity is a different kind of capacity

What limits an AI service is a combination of three things that no single graph shows:

- **Scarcity.** Rate limits cap requests, tokens or concurrency within a window, and their scope varies by model and account. On shared service a limit is permission to send rather than capacity held for the sender, so a request within it can still be refused when the provider is busy. Queuing and retries are application decisions with latency and cost consequences.[16]
- **Pricing.** Per-token charges, subscriptions and provisioned capacity respond differently to demand. Compare the full cost of the service, including unused commitments, before drawing a conclusion from its token rate.
- **Budgets.** A prepaid balance can run out; a commitment can charge after demand disappears. Alerts, hard caps and advisory limits behave differently, so the plan needs to say which one is in force and what happens when it is reached.

Explore this section in the full chapter →

[16] OpenAI, "Rate limits," https://developers.openai.com/api/docs/guides/rate-limits, and "Error codes," https://developers.openai.com/api/docs/guides/error-codes; Google Cloud, "Standard PayGo," https://docs.cloud.google.com/gemini-enterprise-agent-platform/models/deploy/standard-paygo, and "Priority PayGo," https://docs.cloud.google.com/gemini-enterprise-agent-platform/models/deploy/priority-paygo; Microsoft, "Enable priority processing for Microsoft Foundry Models," https://learn.microsoft.com/en-us/azure/foundry/openai/concepts/priority-processing; AWS, "Service tiers for optimizing performance and cost," https://docs.aws.amazon.com/bedrock/latest/userguide/service-tiers-inference.html; checked October 3, 2026. Documented ramp, retry, spend-limit and downgrade behavior; shared pool sizes are not published.

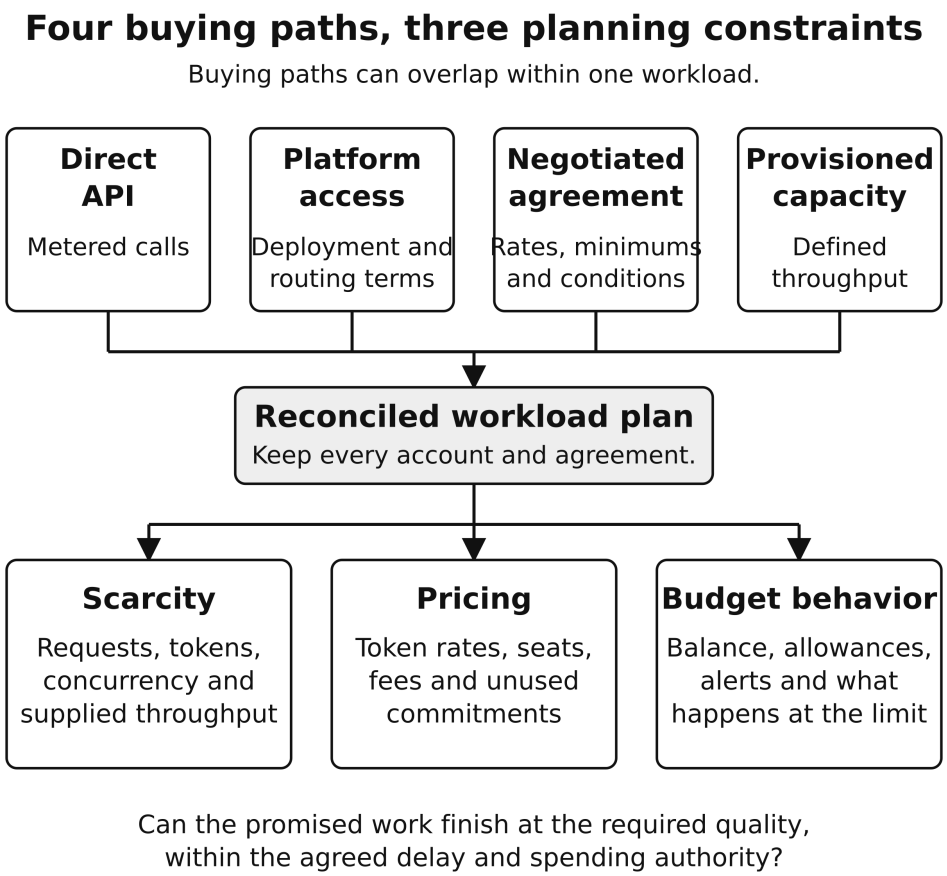


Diagram 7.1. Four buying paths feed one workload plan. Scarcity, pricing and budget behavior remain separate checks even when the organization buys through several paths.

Explore this section in the full chapter →

The model product list and how to read it

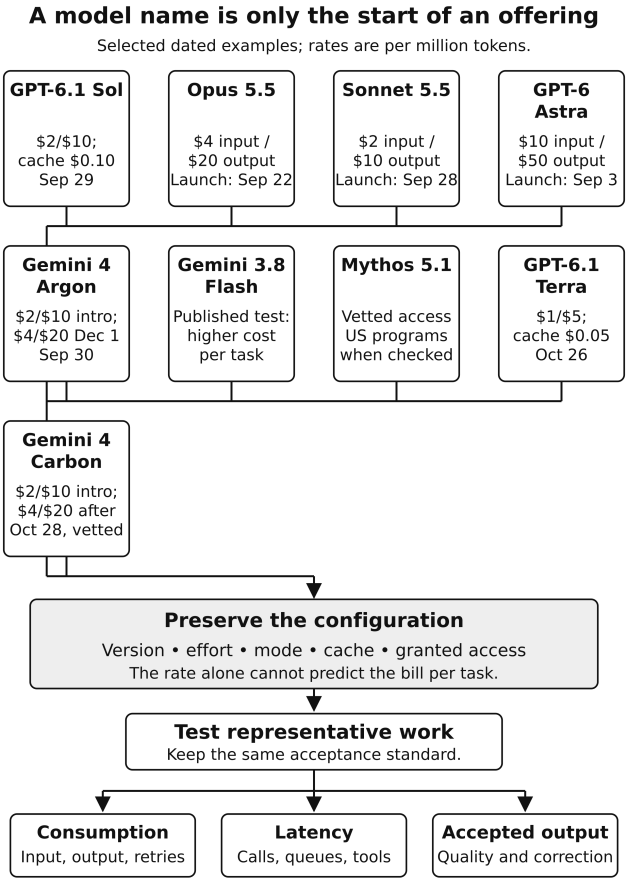


Diagram 7.2. A dated model offering feeds a configuration record, then a test on representative work. The selected examples illustrate different prices, consumption and access conditions; they are not a performance ranking.

Explore this section in the full chapter →

Sizing supply for the busy interval

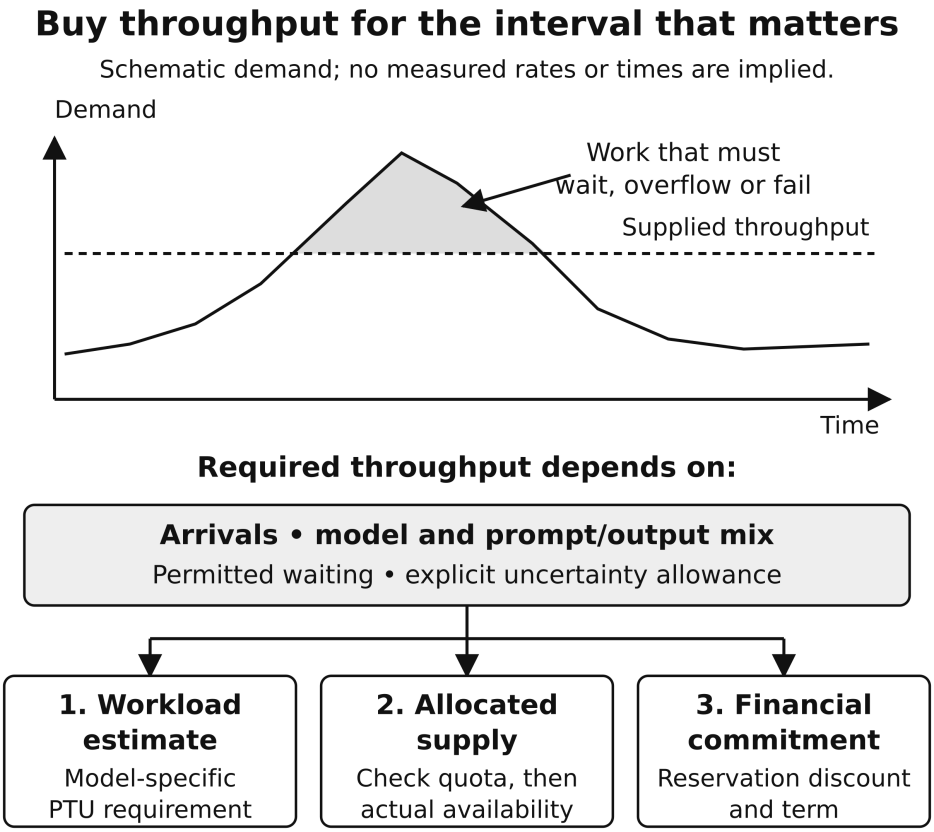


Diagram 7.3. A schematic busy interval exceeds supplied throughput. The amount to buy depends on arrivals, the model and prompt/output mix, permitted waiting and uncertainty; the workload estimate, allocated supply and financial commitment remain separate records.

Explore this section in the full chapter →

Measure the efficiency of the models, not just the spend

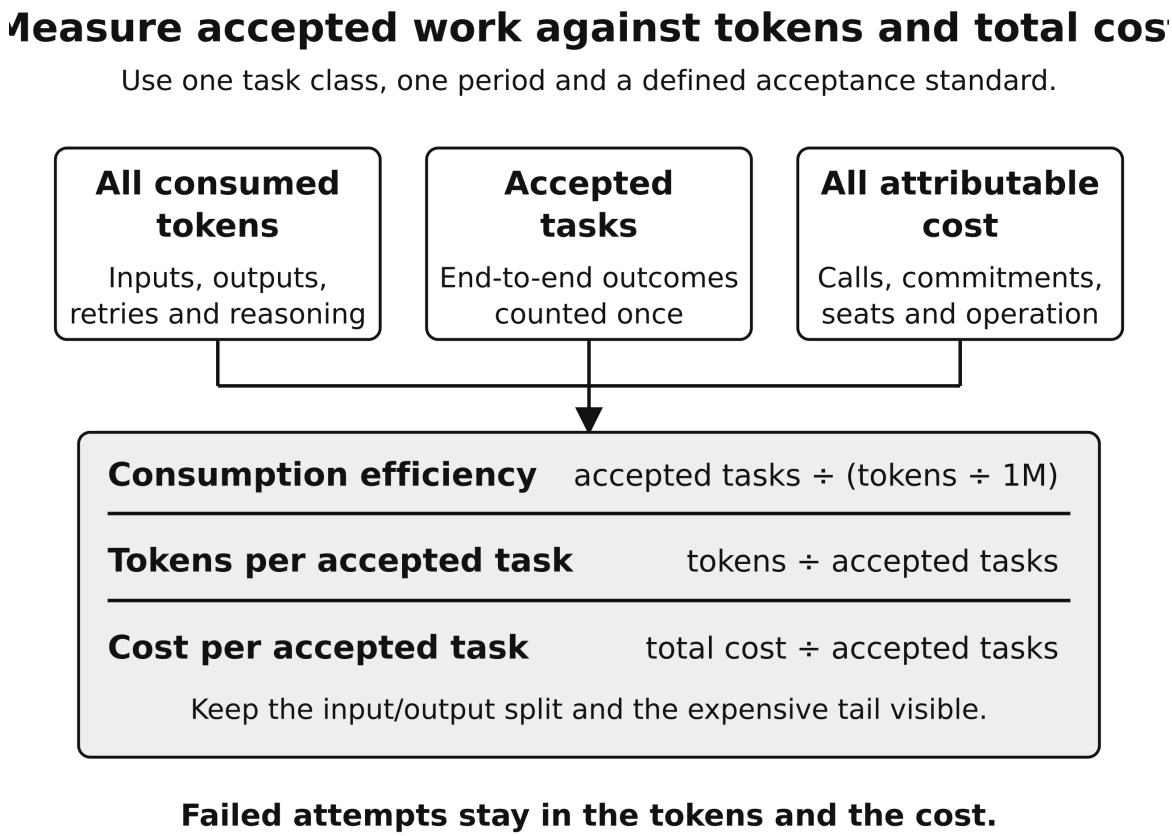


Diagram 7.4. Three measures share the same task class, time window and acceptance standard. Accepted tasks per million tokens and tokens per accepted task are reciprocals, read as efficiency and as intensity; total cost per accepted task adds price and fixed charges. Failed attempts remain in consumption and cost.

Explore this section in the full chapter →

Consumption efficiency counts accepted tasks per million tokens:

$$\frac{\text{accepted tasks}}{\text{tokens consumed}/1{,}000{,}000}.$$

Record it per model and per task class, measured monthly.

Accepted ties the count to the business's outcome standard; preserve whether the result needed correction and what that correction cost. *Per task class* keeps unlike work from being averaged together. Model-level comparisons are useful when a model performs the complete task. For a workflow spanning several models, report the end-to-end outcome once and use step-level costs and acceptance checks to explain each model's contribution. Crediting every model with the same outcome and then summing would count the work repeatedly.

Explore this section in the full chapter →

Recommended practices

Size supply for the work that must finish on time. Use measured demand over the relevant interval, the permitted delay and an explicit uncertainty allowance. Decide which work may queue, which may use overflow and which needs reserved throughput, and test any reservation on price as well as supply.

Run the efficiency measurement every month, with an explanation. Report accepted tasks and token consumption by model and task class, alongside total cost and retries. Investigate changes rather than assuming they represent improvement or waste.

Explore this section in the full chapter →

Keep the budget policy usable across models and providers. State who may spend, what is counted and what happens at the limit separately from the feature implementing it. A provider change may require different controls: a balance, gateway quota, spend cap and commitment make different promises. Re-test their behavior rather than transferring the old configuration by name.

Explore this section in the full chapter →

CHAPTER 8 / 3 MINUTE BRIEF

Measuring AI Workloads

Read the full chapter → / Read this brief online →

The main idea

Ten million tokens sent through a highly rated model do not establish a known amount or quality of completed work. A smaller model may be sufficient for the task; a larger one may succeed where it cannot. Only the application's acceptance results can settle the comparison.

This chapter also proposes **benchmark-weighted consumption**: an unvalidated, descriptive index that joins token counts to model benchmark scores. It can show a change in model mix that a raw token total hides. It cannot turn tokens into a measured quantity of intelligence or substitute for counting accepted work.

Explore this section in the full chapter →

What the workload actually consumes

For an AI workload, the unit of consumption is not the user request. It is the **model call**. One task — a support question answered, a report drafted — typically triggers several calls: a cheap model to interpret the question, a capable model to produce the answer, another model to check the result, occasionally a retry when something failed. Each call consumes input tokens and output tokens on a particular model.

Explore this section in the full chapter →

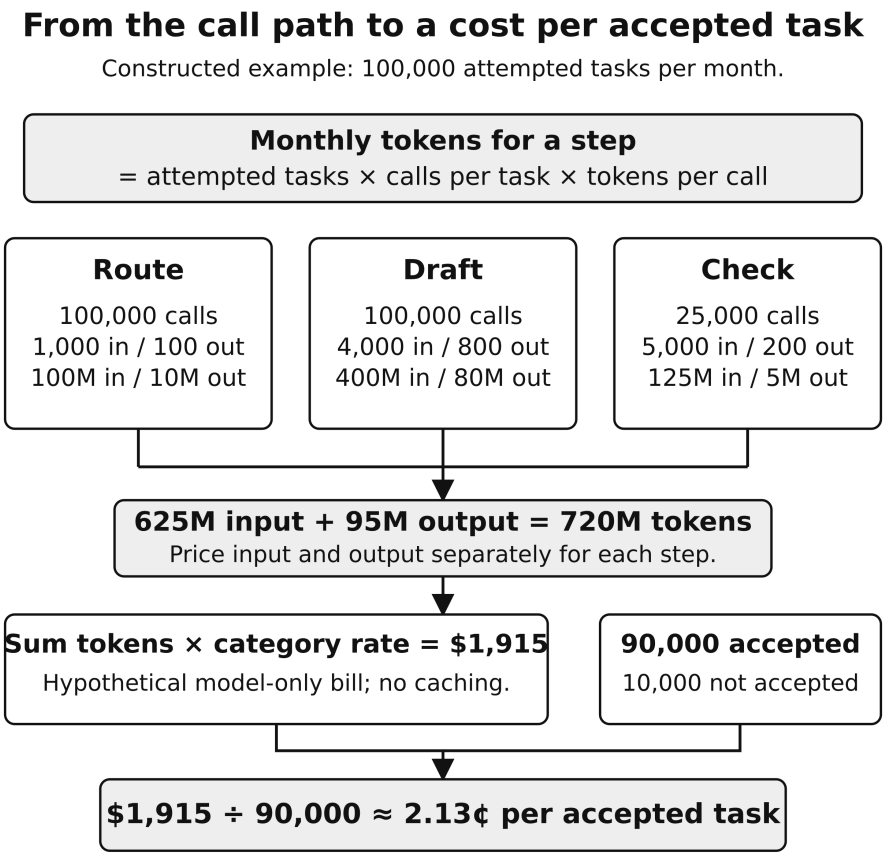


Diagram 8.1. The constructed call path expands attempted tasks into route, draft and check calls, then into token totals and the model-only bill. Dividing by accepted tasks retains the cost of rejected attempts.

Explore this section in the full chapter →

Describing the model mix

Start with a single benchmark version run under comparable conditions. For illustration, suppose two models score 0.85 and 0.45, where the score is a fraction of test cases passed. These are hypothetical results on one test, not universal capability ratings.

For each model, multiply its token count by that score, then add the products. If model k consumes T_k tokens and scores b_k, the index is:

$$B = \sum_{k} T_k \times b_k.$$

One million tokens at 0.85 contribute 850,000 benchmark-weighted tokens; one million at 0.45 contribute 450,000. This is an accounting convention proposed here, not an established measure of effective capability purchased.

An increase from ten million to twelve million weighted tokens is $(12 - 10) / 10 = \mathbf{20}$ **percent** growth in the index, not evidence of 20 percent more useful work. Repeating a prompt unnecessarily would raise the index too. A change in tokenization, benchmark version or task mix can also break the comparison. Publish those conditions alongside it.

Suppose routing moves work to a model with a lower benchmark score. The index falls for the same token count, making that model-mix change visible. It does not establish lost capability on this workload: the cheaper model may pass every task that matters here. Compare acceptance rates, correction effort and total cost before calling the routing change a saving.

Explore this section in the full chapter →

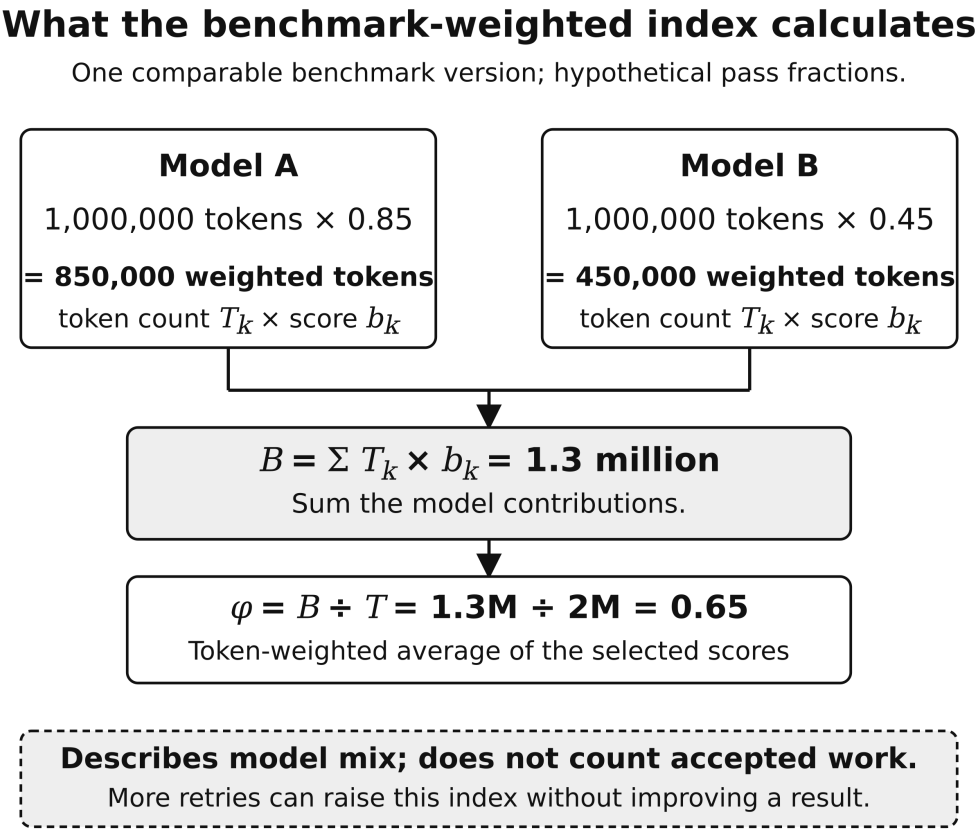


Diagram 8.2. The proposed benchmark-weighted index multiplies each model's tokens by a comparable benchmark score, sums the contributions and divides by total tokens to describe the model mix. All scores here are hypothetical; the calculation does not measure accepted work.

Explore this section in the full chapter →

Choosing the benchmarks

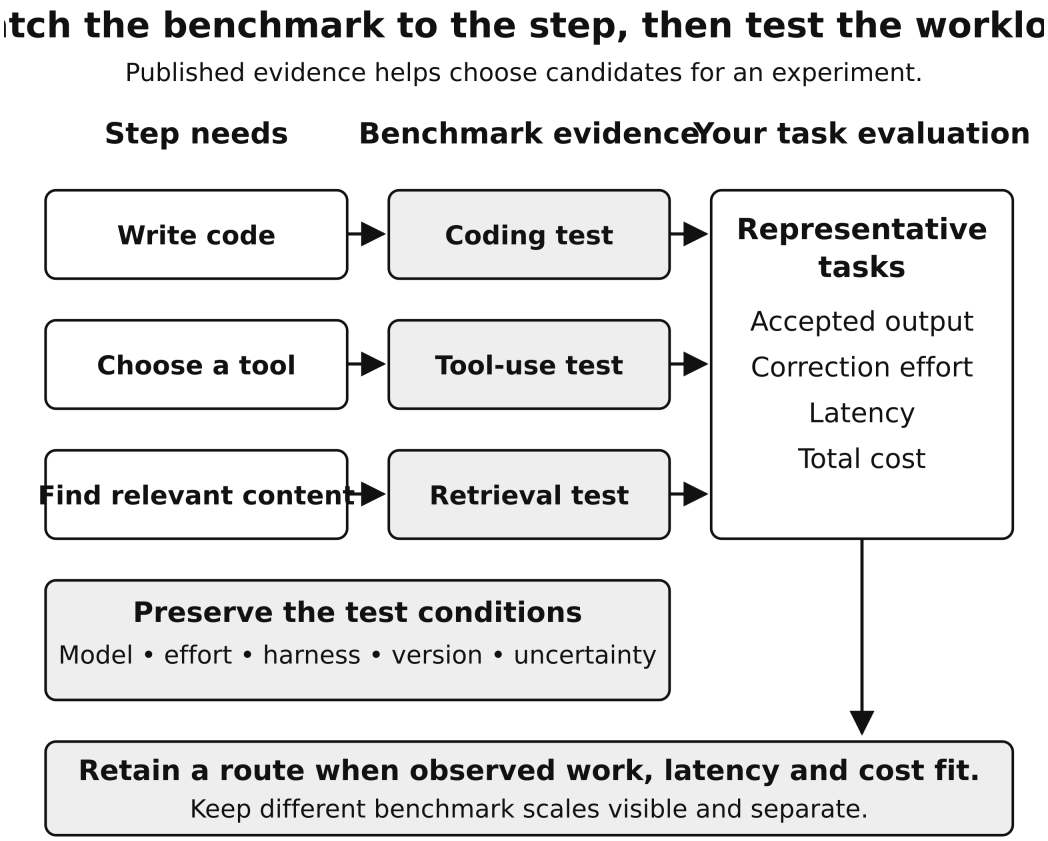


Diagram 8.3. Match the evidence to the skill a step needs, preserve its test conditions and evaluate the candidate on representative work. The routing decision depends on observed acceptance, correction, latency and total cost.

Explore this section in the full chapter →

The purpose question: what no consumption measure can say

So the measurement design needs a second mechanism: an audit question set, asked per task class, by people who can judge the answers:

- Does this step need a model at all? Formatting, validation and deterministic transformations may be better served by ordinary code, whose compute and maintenance still cost money.
- Does it need this class of model? Compare a smaller model on representative examples, including difficult cases and changes in the label set.
- Does it duplicate another workload? Two teams may have good reasons for separate systems, but the reasons should survive being asked.
- Is the architecture appropriate? Repeated retrieval, excessive context and unnecessary agent loops are testable design choices.

Explore this section in the full chapter →

Right-sizing: the cheaper ways to do the work

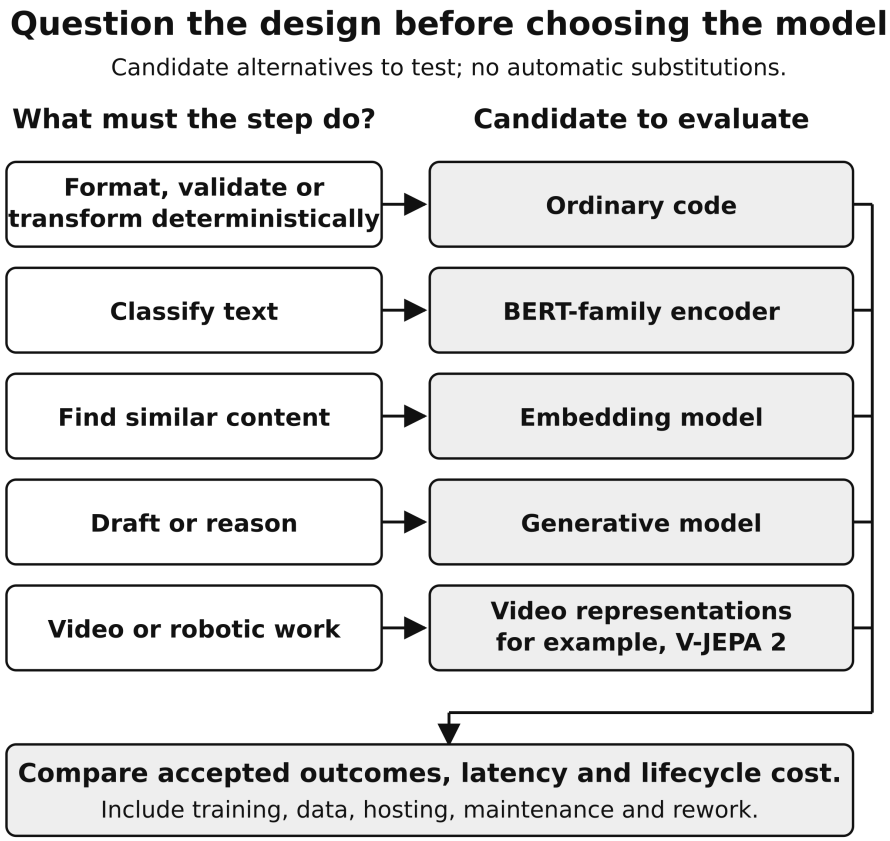


Diagram 8.4. Task requirements suggest candidates to evaluate, from ordinary code and encoders to embeddings, generative models and video representations. Compare accepted outcomes, latency and lifecycle costs before changing the design.

Explore this section in the full chapter →

CHAPTER 9 / 3 MINUTE BRIEF

Orchestration and Agentic Topologies

Read the full chapter → / Read this brief online →

The main idea

A forecast built on model tokens cannot explain an agent's cost: the memory, tools and runtime around each call carry their own bills and limits, and the agent's loops change the shape of consumption itself. Consider a hypothetical support assistant that looks up a customer's account, drafts a reply, checks a refund rule, updates its memory and writes a trace. The customer sees one answer, but several systems did work.

A memory store, a tool server and a gateway may begin as small additions, then acquire their own bills and scaling constraints. A loop rereads its growing history every turn, a lead agent fans work out to parallel workers, retries stack across layers, and a task waiting for a person can hold a sandbox for hours. Some failures interrupt the task; others leave it running with incomplete context or stale data. The capacity team needs to follow both the cost and the behavior back to the work these components support, then ask whether that work was worth what it cost.

Explore this section in the full chapter →

Topologies

An agent may work alone with tools, pass work through a pipeline, or delegate to parallel workers. Each arrangement changes context reuse, latency and coordination. None is reliably the cheapest: isolating a subtask can avoid replaying irrelevant context, while unnecessary delegation can duplicate it. Even a single agent can accumulate substantial input consumption by repeatedly sending its growing history.

Explore this section in the full chapter →

The operating inventory

This is the inventory I would keep for the layer; check your own stack against it for lines nobody owns.

Component	What it is	Consumption scales with	Bill shape
Model gateway	Routing, retries and fallbacks	Traffic × retries at the gateway	Gateway tier, request charges or self-run compute; model charges remain separate
Prompt cache	Reuse of identical prompt prefixes	Repeated prefixes within the cache lifetime	Premium on writes, discount on reads; reads may not count toward rate limits[17]
Session store	Conversation state with time-to-live expiry	Concurrent sessions × retention	Stored state and infrastructure; expiry is a policy choice
Memory service	Extracted facts, episodes and instructions	Writes × model calls per write; retained history	Extraction and embedding calls, retrieval queries, storage under the retention policy
Vector database	Similarity search over embeddings	Vectors × dimensions × replicas, plus query rate	RAM-heavy nodes or hosted tiers
Graph database	Entities and relationships, sometimes with validity intervals	Entities × traversal rate; superseded facts if kept	Licensing varies by engine; self-hosting is an operations obligation
Tool/MCP servers	Structured access to external systems	Agent activity × tools per step × retries	Per-seat or per-call hosted, or self-run processes; downstream charges separate
Sandbox runners	Isolated execution for generated code	Arrival rate × holding time, including waits	Compute time or session charges; startup and expiry affect cost
Observability	Traces of agent steps and tool calls	Agent activity × sampling rate	Ingestion plus storage; sampling is a cost decision

Explore this section in the full chapter →

[17] Anthropic, “Prompt caching” and “Rate limits”, checked October 4, 2026. Five-minute writes at 1.25 times the base input price, one-hour writes at 2 times, reads at 0.1 times (less on some newer models); hits need identical prefixes; on most models, cached reads do not count toward input-tokens-per-minute limits. Provider-specific and subject to change.

What the review inherits

Agent spend has the shape of Volume II, Chapter 15's long tail — thousands of small autonomous decisions, no single owner watching each one — inside the shape of Volume II, Chapter 17's relationships, because the workflow owners are the people who know what the agents should be doing. I would have the capacity team build an attribution record first, tying each event on each system to an agent, workflow, customer and task. With it, the monthly review in Volume II, Chapter 22 can ask what the agents completed and what that work cost across every layer, and the workflow owner can say what it was worth.

Questions I would bring to that review:

- Which events, on which systems, carry no agent, workflow or customer?
- What did an accepted task cost across all layers, counting failed and abandoned attempts, and how wide is the spread?
- Which resources are held rather than used, and what releases them?
- How many attempts can one failure produce, and where is the retry budget enforced?
- What does the memory store keep, and does deletion reach its indexes, history and backups?

Explore this section in the full chapter →

CHAPTER 10 / 4 MINUTE BRIEF

Specialized Hardware

Read the full chapter → / Read this brief online →

The main idea

A hosted model is not the only way to serve an AI workload. Data-placement requirements, latency or sustained volume may justify running a model on rented or owned GPUs, but none of them by itself requires buying hardware: a permitted regional service may solve the placement problem, and a negotiated API arrangement may solve the price problem. Compare those options before taking on a device fleet.

GPU capacity can be metered by instance time, reserved under contract or owned, and each choice changes lead times and operating obligations. The plan needs to know where the devices can run, how much power and cooling they need, which failures remove a whole group at once, and the supplier's current lead time; "measured in quarters" is not a property of every purchase.

Explore this section in the full chapter →

The arithmetic a planner owns

The raw weight payload is parameters times bytes per parameter. A seventy-billion-parameter model therefore needs about 140 GB at 16 bits, 70 GB at 8 bits or 35 GB at 4 bits, using decimal gigabytes. Actual allocation also includes scales, metadata and runtime state. Quantization can reduce memory and alter quality or speed; evaluate the chosen method rather than assuming a fixed quality loss.[18]

For a conventional transformer attention cache, the bytes per retained token are:

$$\begin{aligned}\text{cache bytes per token} = 2 \times \text{layers} \times \text{key-value heads} \\ \times \text{head dimension} \times \text{bytes per value}.\end{aligned}$$

[18] vLLM, "Quantization", checked October 3, 2026. Methods and supported hardware differ by release. Raw weight arithmetic excludes scales, metadata and runtime allocations.

The factor of two counts keys and values. Grouped-query attention can use fewer key-value heads than query heads, and other attention designs need different calculations. PagedAttention addresses cache allocation and sharing; it does not imply that every seventy-billion-parameter model has the same cache size.[19] The memory-only concurrency bound is:

$$\text{memory-only concurrency bound} = \frac{\text{device memory} - \text{weights} - \text{reserve}}{\text{KV cache per request}}.$$

Take a hypothetical configuration with 80 layers, 8 key-value heads, head dimension 128 and two bytes per cached value. The cache per token is $2 \times 80 \times 8 \times 128 \times 2 =$ **327,680 bytes**. At 8,000 retained tokens, a request needs 2,621,440,000 bytes, or about **2.62 GB**, before allocation overhead. Suppose 70 GB of weights fit in 141 GB and we reserve 14.2 GB for other use: $141 - 70 - 14.2 =$ **56.8 GB** remains. The memory bound is $\left\lfloor \frac{56.8}{2.62144} \right\rfloor =$ **21 requests**. With 64 key-value heads instead of eight, each cache would be eight times larger and the bound would fall to two. These invented configurations show why head count belongs in the calculation, not whether either workload meets its latency target.

Explore this section in the full chapter →

What must be measured

There is no universal factor that turns a month's token count into the number of GPUs to buy. The conversion requires a measured configuration, task distribution, latency requirement, availability arrangement and demand pattern. A serving team may know that long contexts change the bottleneck; a platform team may know that the required devices cannot be placed together by launch. Establish those conditions before negotiating a price.

Explore this section in the full chapter →

[19] Woosuk Kwon and colleagues, "Efficient Memory Management for Large Language Model Serving with PagedAttention," SOSP 2023, https://arxiv.org/abs/2309.06180. Describes the cache memory lost to reservation and fragmentation under contiguous allocation, and paged allocation as the remedy. Its OPT-13B example applies the same per-token formula: $2 \times 5{,}120 \times 40$ layers $\times$ 2 bytes, about 800 KB.

An experiment on shared hardware, before the fleet

The chapter opens on the arithmetic a purchased fleet needs, and that arithmetic starts with a serving experiment. The experiment itself does not need purchased or reserved hardware. Three shared services cover the step: an interpretability service that runs intervention code on models already deployed on shared GPUs, and two rented options — a GPU pod billed by the second, and a function platform that runs custom code on a GPU it selects. Both rented options charge for the time the work uses.

Explore this section in the full chapter →

The account and the meter teach the same lesson Chapter 2 teaches about elasticity: per-second or per-hour capacity keeps an experiment's cost proportional to the experiment. The fleet decision still needs everything this chapter sizes — goodput, failure groups, usable fraction — but the evidence arrives first, on someone else's capacity.

Explore this section in the full chapter →

The sizing process

Measure, model, decide, and come back afterward: the loop Chapter 4 ran for web workers. The serving experiment establishes real rates at the real configuration. The model connects demand to devices: concurrency, cache budget, latency targets, growth. The decision compares purchasing arrangements, failure conditions and lead times before committing to supply.

Explore this section in the full chapter →

The process should leave a record that someone else can check and rerun:

- **Workload:** peak arrival rate, prompt and output lengths, TTFT and TPOT objectives and the share that must meet them.
- **Configuration:** model, precision, engine version, device, VM size, devices per replica.
- **Evidence:** benchmark command and date, goodput per device, memory bound, bandwidth floor.
- **Failure and supply:** purchase unit, failure allowance, maintenance plan, supply arrangement, lead time, fallback.
- **Revisit when:** the model, precision or prompt lengths change, goodput falls, or a contract date nears.

Explore this section in the full chapter →

CHAPTER 11 / 5 MINUTE BRIEF

Exotic Hardware: Quantum and Emerging Capacity

Read the full chapter → / Read this brief online →

The main idea

"Do we have a plan for quantum?" is an awkward question to receive in a meeting about next quarter's capacity. It may mean that a research team needs access to a particular machine, that Security needs help with a cryptographic migration, or that somebody read an impressive announcement. Those are three different requests. Answering all of them with "we've got it taken care of" would be efficient only until somebody asked what we meant.

I want a capacity team to take the question seriously without making every promising technology a purchasing requirement. The useful response names what exists, what the organization might do with it, the evidence that would justify a commitment, and the work we should already be doing on ordinary infrastructure: preparing for post-quantum cryptography does not require a quantum computer.

Explore this section in the full chapter →

Quantum, honestly assessed

Quantum error correction is making measurable progress. In Google's Willow experiment, logical error rates fell as the surface code grew from distance three to five to seven: adding physical qubits made the encoded memory more reliable.[20] IBM's published roadmap targets a fault-tolerant Starling system in 2029 with 200 logical qubits and a circuit capacity

[20] Google Quantum AI and collaborators, "Quantum error correction below the surface code threshold," *Nature*, 2024, https://www.nature.com/articles/s41586-024-08449-y. Accessible author account: https://blog.google/innovation-and-ai/technology/research/google-willow-quantum-chip/. The result concerns the demonstrated error-correction experiment, not enterprise workload economics.

of 100 million gates.[21] The first is an experimental result and the second a supplier's target; both matter, in different columns of the evidence record.

Two results deepened the experimental column in October 2026. An error-corrected machine runs some operations cheaply and consumes a prepared state, called a **magic state**, to perform the rest; Google's post notes estimates that producing such states can occupy up to 95 percent of a fault-tolerant machine's capacity. On the same Willow processor, a filtering method called magic state cultivation — repeated error checks that keep and grow only the cleanest attempts — cut errors in the prepared states roughly fortyfold at a reported 99.99 percent fidelity, with the authors' own note that demonstration at larger scale remains to be done.[22] Quantinuum, whose systems are available through Azure Quantum, reported a 99.9995 percent logical fidelity on its commercial Helios machine — five errors in a million operations, ten times better than its own result the month before — and ran two error-correction codes on the same hardware at different physical-qubit costs, ten versus up to thirty-seven per logical qubit.[23]

[21] IBM, "How IBM will build the world's first large-scale, fault-tolerant quantum computer," June 10, 2025, https://www.ibm.com/quantum/blog/large-scale-ftqc. Supplier roadmap and explanation of logical versus physical qubits; dates and delivered capabilities must be checked when procuring.

[22] Google Quantum AI, "Bringing the magic: Google breaks open a bottleneck in quantum computing," October 15, 2026, https://blog.google/innovation-and-ai/models-and-research/quantum-computing/magic-state-cultivation-quantum-computing/; the paper, "Magic state cultivation on a superconducting quantum processor," is in *Science*, https://www.science.org/doi/10.1126/science.aee7961. First hardware demonstration of the technique: about a fortyfold error reduction and a reported 99.99 percent T-state logical fidelity on the Willow processor. The post notes estimates that up to 95 percent of a machine's capacity can go to producing magic states and states that demonstration at larger scale remains. A vendor's account of its own result.

[23] Quantinuum, "Quantinuum Breaks Error-Correction World Record by 10x, Validating its Leading Path to Fault-Tolerant Quantum Computing by 2029," October 19, 2026, https://www.quantinuum.com/press-releases/quantinuum-breaks-error-correction-world-record-by-10x-validating-its-leading-path-to-fault-tolerant-quantum-computing-by-2029. Vendor announcement: a 99.9995 percent logical fidelity on the commercial Helios system using a color code, ten times the company's own September 2026 result on the same hardware; the announcement states its Helix code used 10 physical qubits per logical qubit against up to 37 for the color code, and links the result to its planned Apollo system. Quantinuum appears among Azure Quantum's priced providers; the record and the roadmap claims are the company's own.

A **physical qubit** is part of the hardware. A **logical qubit** is encoded information protected using a particular error-correction scheme. The number of physical qubits needed for a useful logical computation depends on error rates, code, connectivity and the operations being performed. Dividing one headline by another, such as a million-qubit resource estimate by the latest logical-qubit count, therefore does not measure the remaining distance.[24]

Explore this section in the full chapter →

For an enterprise capacity decision, qubit count is only a beginning. I would ask which algorithm the workload needs, what accuracy and confidence the result requires, how many repetitions it takes, and what classical preprocessing and verification remain. Then compare the complete run with the best suitable classical approach. A benchmark designed to expose a quantum advantage can be an important scientific result without establishing a cheaper way to complete the company's work.

That is a reason to keep qualified people engaged, not to dismiss the field or to buy a machine on a roadmap's date.

Explore this section in the full chapter →

[24] IBM, "How IBM will build the world's first large-scale, fault-tolerant quantum computer," June 10, 2025, https://www.ibm.com/quantum/blog/large-scale-ftqc. Supplier roadmap and explanation of logical versus physical qubits; dates and delivered capabilities must be checked when procuring.

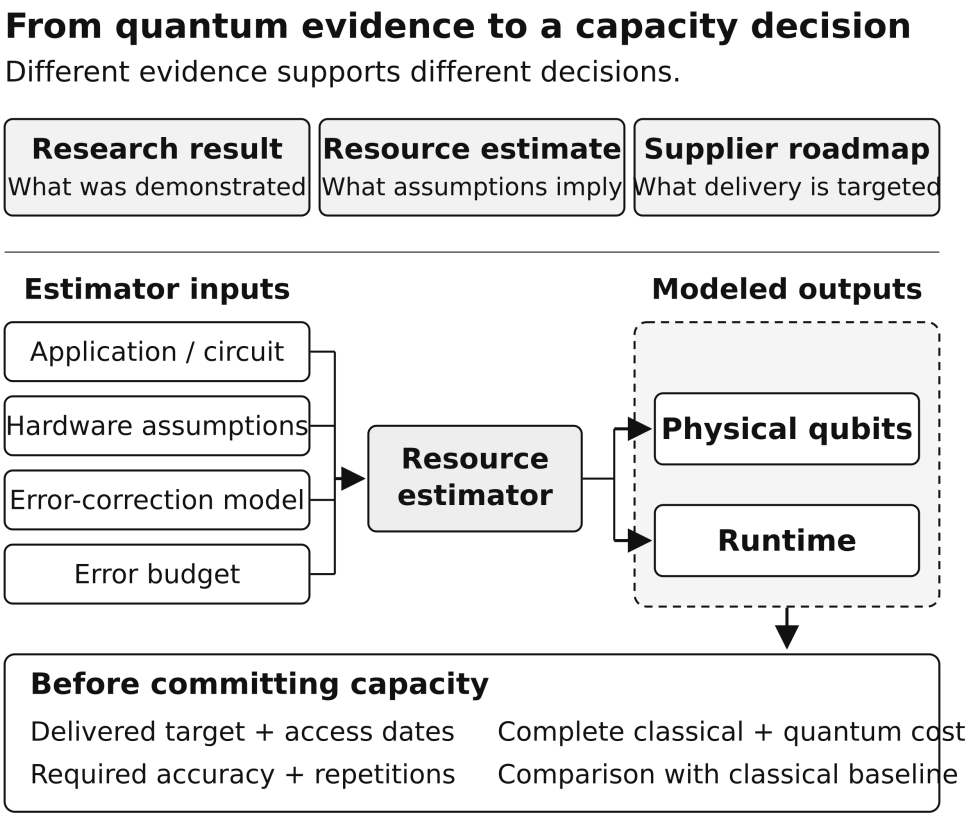


Diagram 11.1. A demonstrated result, a resource estimate and a supplier roadmap support different decisions. The estimator maps application and hardware assumptions, error correction and error budget to physical qubits and runtime. Access, complete cost and comparison with a classical baseline remain separate capacity checks.

Explore this section in the full chapter →

Post-quantum cryptography is a different project

Post-quantum cryptography replaces vulnerable cryptographic mechanisms with ones designed to resist quantum attacks. It includes key establishment and digital signatures, not just encryption. NIST finalized ML-KEM, ML-DSA and SLH-DSA in August 2024; these algorithms run on classical computers.[25] An organization can therefore have a substantial quantum-related migration while purchasing no quantum hardware.

Explore this section in the full chapter →

[25] NIST, FIPS 203, https://csrc.nist.gov/pubs/fips/203/final; FIPS 204, https://csrc.nist.gov/pubs/fips/204/final; FIPS 205, https://csrc.nist.gov/pubs/fips/205/final. Finalized August 13, 2024; key encapsulation and signature standards for classical implementation.

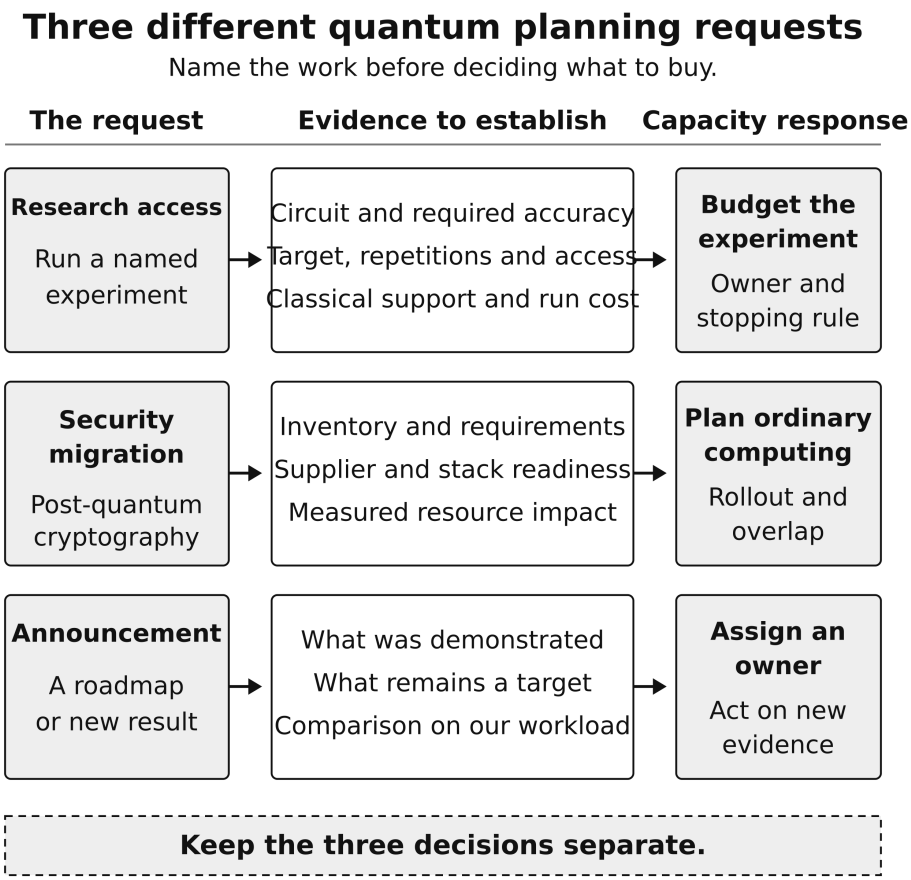


Diagram 11.2. Research access, cryptographic migration and a supplier announcement ask for different evidence and capacity responses. An experiment needs a budget and stopping rule; post-quantum migration uses ordinary computing; a roadmap needs an owner who can recognize when the evidence changes.

Explore this section in the full chapter →

The evidence standard for emerging architectures

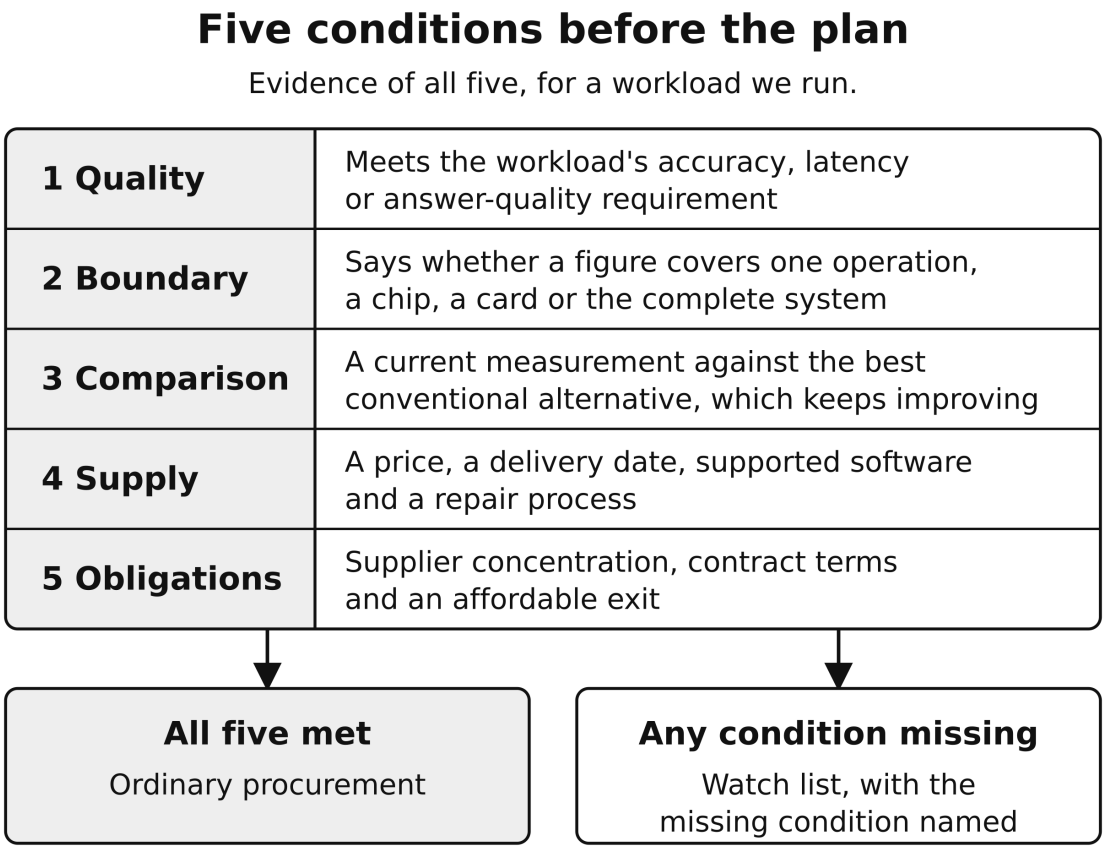


Diagram 11.3. An architecture enters the plan with evidence of all five conditions for a workload we run: quality, boundary, comparison, supply and obligations. A product that meets them all is ordinary procurement; anything short of that goes on the watch list, with the missing condition named.

Explore this section in the full chapter →

Photonic

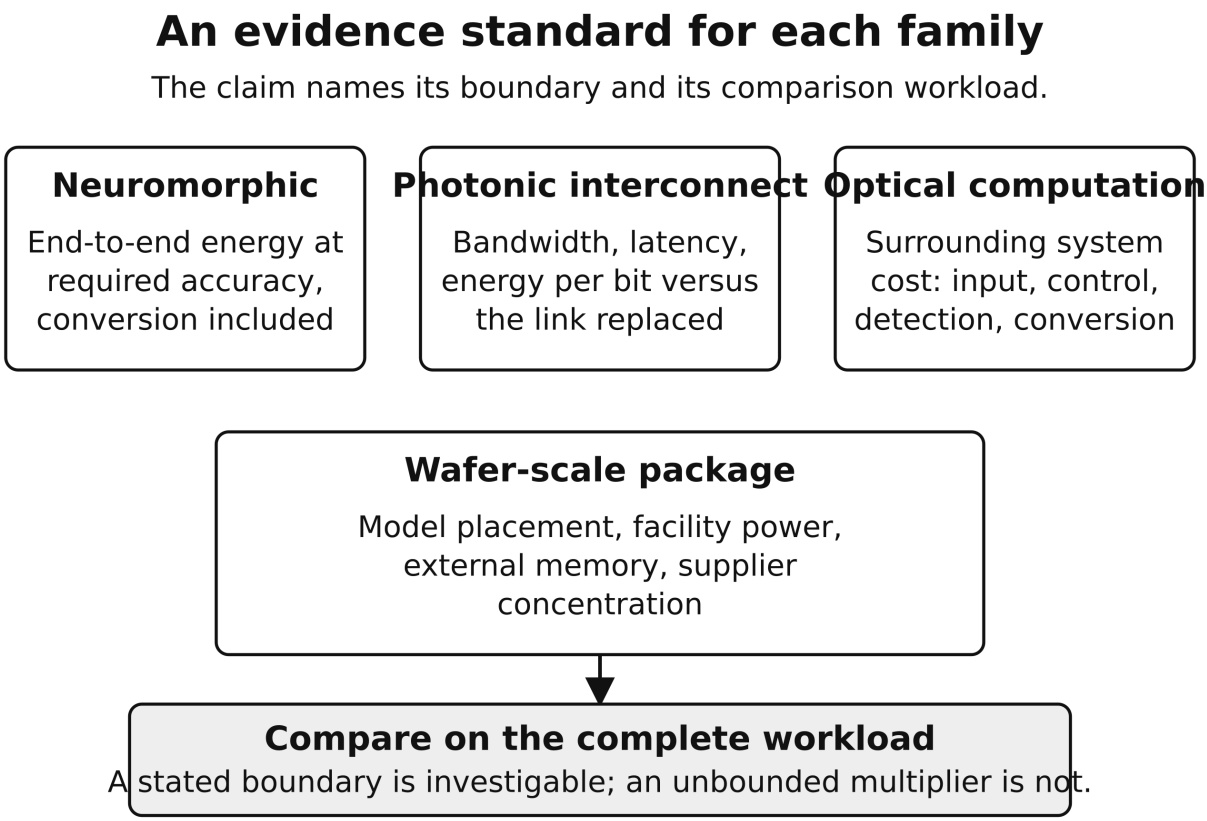


Diagram 11.4. One rule applies to every family: a claim names its boundary and its comparison workload. That means energy at the required accuracy for neuromorphic chips, energy per bit against the replaced link for optical interconnect, the surrounding system for optical computation, and placement, facility power and supplier concentration for wafer-scale systems.

Explore this section in the full chapter →

Quantum annealing

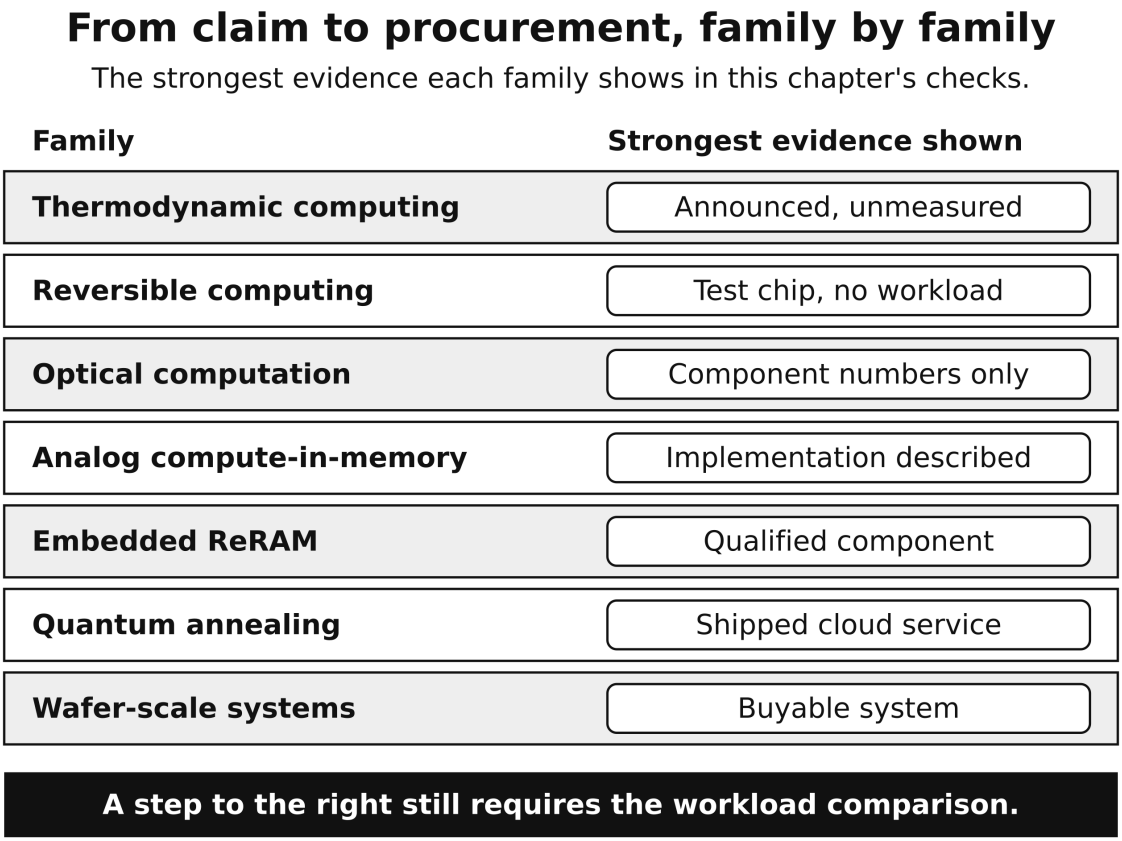


Diagram 11.5. Each hardware family in this section sits at a different evidence stage: announced designs, measured components, a qualified component you can source, or systems you can buy today. Every stage still faces the same workload comparison before it enters a capacity plan.

Explore this section in the full chapter →

Wafer-scale

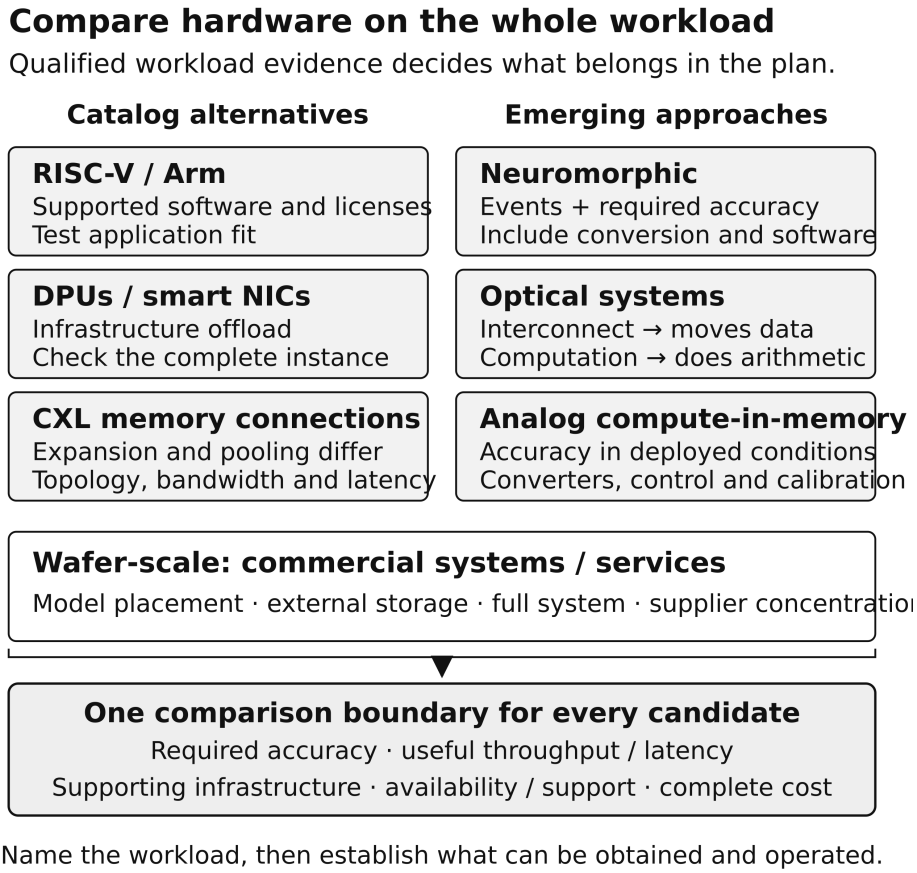


Diagram 11.6. Catalog alternatives, emerging approaches and commercial wafer-scale systems change different parts of a computing system. Every candidate is compared on the complete workload, including support, surrounding infrastructure and cost.

Explore this section in the full chapter →

The watch list

The watch list should say what would change a decision and who will recognize the change. “Quantum is interesting” is too vague to review. “Run a funded comparison when a supplier can execute this circuit at this accuracy within this experimental budget” gives the specialist something to evaluate.

Entry and owner	Evidence to keep	Decision it could change
Quantum experiment (research lead)	Circuit, accuracy, repetitions, classical baseline, access and complete run cost	Whether to fund the next research stage
Emerging accelerator or priced specialist service (platform owner)	Quality and latency on a representative workload, full-system power, price and terms, supported software	Whether to pay for a comparison or procure for a named workload
New memory or interconnect (hardware specialists)	Supported configuration, measured bottleneck, placement and failure behavior	Whether the next pool should use it
Cryptographic migration (Security, with capacity)	Required algorithm and stack, interoperability results, measured resource impact, deadline	When to fund rollout capacity

The last entry belongs to an active migration program when requirements are established; it should not be left waiting for a quantum breakthrough. The others can remain conditional without becoming forgotten.

Explore this section in the full chapter →

CHAPTER 12 / 3 MINUTE BRIEF

Databases and Data-Bound Technologies

Read the full chapter → / Read this brief online →

Plan the work as well as the space

I find it helpful to follow four questions through an investigation. What has to exist, including physical representations, copies, and retained history? What has to happen to it—reads and writes, maintenance, transfer, or recovery? Where can that work happen, given the nodes, partitions, volumes, and failure domains? Finally, when must it finish, and what service has to remain available while it does?

The questions resist being collapsed into one metric because an answer to one may tell us almost nothing about another. Knowing the retained footprint does not establish recovery time; a good aggregate throughput result may conceal the busiest partition. Free disk space cannot be spent as network throughput, however attractively a report combines their percentages.

Explore this section in the full chapter →

Cassandra: the data model is part of the capacity plan

Cassandra's partition key determines how data is grouped into partitions. Rows sharing that key stay together on their replica nodes, so adding nodes does not subdivide the partition. The project's modeling guidance starts with the queries the application needs to support.[26] That makes the distribution of partition sizes and activity much more interesting to me than the result of dividing total bytes by node count.

Explore this section in the full chapter →

[26] Apache Cassandra documentation, "RDBMS design" and "Data Definition", checked October 3, 2026. Query-first modeling and partitions as undivided units of storage; the customer example is hypothetical.

Kafka: retention is a promise to a future reader

A Kafka consumer tracks its position in a partition's log and can return to earlier positions to read retained records again. Consuming a record does not instruct Kafka to delete it. Under the delete policy, retention limits govern deletion, while log compaction preserves the latest value for a key rather than a complete event history.[27] Before estimating storage, I want to know which of those things the customer expects to have available when they return.

Explore this section in the full chapter →

A deletion does not settle the storage question

PostgreSQL updates and deletes leave old row versions for later reclamation. Routine vacuuming makes that space reusable, and it also freezes old transaction IDs; if freezing falls far enough behind, PostgreSQL stops assigning new ones and accepts only read-only transactions until vacuum catches up. Ordinary `VACUUM` generally does not return space to the operating system. `VACUUM FULL` rewrites the table, so it needs room for a complete new copy and an exclusive lock that blocks all other use of the table until it finishes.[28]

Explore this section in the full chapter →

Build a data-capacity decision record

We need a way to carry this work into the forecast without asking every reviewer to become an expert in the technology. The following record preserves the technical differences while exposing the obligations and choices that other teams need to understand.

[27] Apache Kafka 4.3 documentation, "Design", checked October 3, 2026; the downloads page still listed 4.3.1 (June 25, 2026) as the newest release when rechecked October 4, 2026. Replay of retained records; consumption versus deletion and key compaction.

[28] PostgreSQL 18 documentation, "Routine Vacuuming", checked October 3, 2026. Reuse versus release, the steady-state goal, `VACUUM FULL`'s copy and `ACCESS EXCLUSIVE` lock, and read-only operation near wraparound.

Record	What the decision needs
Business obligation	Customer, useful outcome, retention purpose, latency requirement, recovery deadline, and who may change each commitment
Data footprint	Logical versus physical measurements, indexes and logs included, replica scope, growth and deletion assumptions, and placement distribution
Work and limits	Foreground demand, tested bottleneck, maintenance backlog and its deadline, effective transfer or replay rate, and competing work
Operating scenarios	Normal service, maintenance, expansion, a specified failure, recovery, and migration overlap; identify which coincide
Evidence boundary	Engine version, configuration, schema/query assumptions, measurement window, test conditions, and unresolved questions
Decision	Action, cost, resource location, lead time, accountable owner, review trigger, and the consequence if the assumption is wrong

I would not turn the record into a universal utilization score. Cassandra repair, Kafka replay, and PostgreSQL vacuuming need different measurements; what we can share is a way to explain how their results affect a decision. The specialists retain their methods, and the business can see which commitments a proposed change preserves or puts at risk.

Explore this section in the full chapter →

CHAPTER 13 / 3 MINUTE BRIEF

Measuring Value

Read the full chapter → / Read this brief online →

The main idea

One recurring part of this job is being shown an attractive savings number and then making myself less popular by asking how it was calculated. I usually start with staffing, licensing, and the prices used for the alternative: were they list prices or our negotiated rates, and would the supplier recognize them? Those questions are often where an enthusiastic conversation becomes a little more restrained.[29]

I understand the frustration. Someone has found a worthwhile improvement and wants to proceed, and my questions can easily become a referendum on whether I support the team, when I only want to know whether we can rely on the estimate. Often the staffing is covered and the alternative really is more expensive, and saying so strengthens the proposal. The capacity team's own proposals get the same questions.

A savings number is a claim about two states of an organization, and it is only as good as what it counts: a unit that means something, the cost of the route as well as the destination, the kind of benefit and the date it reaches the bill, and the people who will run the result. The comparisons that go most wrong usually leave out the people.

Explore this section in the full chapter →

Choose a unit that means something

The FinOps Foundation describes unit economics as bringing together what an organization spends on technology and the value that spending creates, and it separates resource-efficiency metrics, such as cost per gigabyte stored or per token, from business metrics such

[29] Tim O'Brien, "What's the Secret to FinOps? I'll Tell You Mine—Now Share Yours!", May 11, 2025. The source of the opening's questions about staffing, licensing, and list versus negotiated prices. It describes a recurring experience, not a particular proposal or realized saving.

as cost per transaction or cost to serve.[30] That connection is worth building. The difficulty comes when a view of the whole system becomes the only view permitted in a decision about a particular customer, migration, or recovery obligation.

We need to say when a unit of work is complete and what quality it must meet. A completed customer order, a gigabyte of data kept recoverable for a month, and an accepted AI task are possible starting points, each needing its own definition. Requests, retries, copied records, and generated tokens explain activity within those services, but more activity doesn't mean the business received more value.

Explore this section in the full chapter →

Name the benefit that actually occurred

I keep the names of benefits specific, as the FinOps Foundation does in asking practitioners to separate avoidance, savings, and cost reduction.[31] **Reduced expenditure** means an actual obligation or charge fell within the stated boundary; it is the only one of these that lowers a bill already in the baseline, so it is the only one I would call a saving without qualification. An **avoided purchase**, often reported as cost avoidance, needs an identifiable plan and a credible reason the purchase would otherwise have happened; it changes a forecast rather than a bill. **Released capacity** is available for another use, **shifted cost** is carried by another workload or organization, and **improved service** is a useful change in quality, capability, or timing even where no bill fell.

One action may produce several of these effects; they are not separate savings to add together. Reducing an allocation inside an unchanged commitment can create headroom without lowering expenditure.

Explore this section in the full chapter →

A number worth carrying forward

A useful value estimate states the expected outcome, the costs included, today's spend, and the proposed system's cost during and after the transition. It names who supplied

[30] FinOps Foundation, "Unit Economics", FinOps Framework capability, checked October 4, 2026. Relates technology spending to the value it creates and divides unit metrics into resource-efficiency and business categories, with the examples quoted here. The website and migration examples are hypothetical.

[31] FinOps Foundation, "Unit Economics", checked October 4, 2026. Lists "Separate avoidance, savings, and cost reduction measurements" among the FinOps practitioner's activities. The five benefit names and their definitions here are the author's.

the inputs and keeps uncertain conditions, especially future volume, visible enough to test. For the website migration:

Item	Website migration (hypothetical)
Outcome and unit	Completed transactions, counted once: 25 million a month
Spend today	$2.5 million a month for computing, amortized view
Proposed steady state	$2 million a month for the same outcomes and obligations
Cost of getting there	(4 × $2 million) + $1 million = $9 million in cash; Finance confirms any capitalized share
Benefit	Reduced expenditure of $0.5 million a month at run rate from cutover, realized as old charges and commitments end
Payback	18 months after cutover, 22 from the start; each month of slip moves breakeven five months
Assumptions to test	Constant demand, the replacement's cost at production volume, and the length of the overlap
Owners and reviews	Application, platform, data, and Finance owners; review at cutover and six months later

Explore this section in the full chapter →