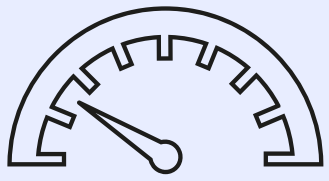


Tech Jargon

Part 10

for

Product Managers

The -ilities

Scalability | Availability | Latency | SLAs

From an Engineer-Turned-PM

The Reliability & Performance Mental Model

A way of framing **system behavior under real-world load** — scalability, availability, latency, throughput, and what happens when users actually show up.

It helps PMs understand why reliability work competes with feature work — and why it should. Get this wrong, and you learn about scalability limits from your angriest customers. Get it right, and your product performs under pressure.

When you understand this mental model, you can:
• Speak the language of SLAs, uptime nines, and latency percentiles — not just "it should be fast"
• Make the case for load testing, redundancy, and failover even when there's feature pressure
• Participate in incidents and postmortems with real context, not just "when will it be fixed?"
• Understand why reliability is a product decision, not just an engineering one

This guide covers the 10 essential terms that bring this mental model to life.

Scalability

Can your system handle more?

Scalability is your system's ability to handle growth — more users, more data, more requests — without falling over. There are two kinds: vertical (a bigger server — more CPU, more memory) and horizontal (more servers sharing the load). Vertical has a ceiling. Horizontal is how modern systems grow.

A restaurant chain. Vertical scaling is building a bigger kitchen with more stoves. Horizontal scaling is opening more locations. The single kitchen has a ceiling. More locations is how chains grow.

We can't vertically scale anymore — we need to shard the database and scale horizontally.

For PMs, this means when your team says "it won't scale," the architecture has a ceiling. Scaling affects cost, timeline, and what you can promise customers — a product conversation, not just a backend one.

Availability

Is your system up when users need it?

Availability is the percentage of time your system is up. It's measured in "nines" — each one exponentially harder and more expensive.

99.9% → 8.76 hours/year downtime
99.99% → 52.6 minutes/year downtime
99.999% → 5.26 minutes/year downtime

A 24-hour diner advertising "always open." Three nines means closed almost 9 hours a year. Five nines means closed for 5 minutes.

We're at four nines right now. Going to five nines would mean rearchitecting the failover — that's a quarter of work.

For PMs, this means availability targets shape architecture, cost, and team investment. Promising 99.99% uptime means engineering is on the hook for less than 53 minutes of downtime per year. Know what you actually deliver before putting a number in a sales deck.

Latency

How long users wait for a response?

Latency is the time between a user's request and the system's response. The number that matters isn't the average — it's the percentiles. P50 is the median. P95 covers all but the slowest 5%. P99 catches the worst 1%. Great P50 with terrible P99 feels fast most of the time — until it doesn't.

A coffee shop's wait times. The average wait is 2 minutes — but the slowest 1% of customers wait 10. If you happen to be in that 1% every morning, the average means nothing to you.

P50 looks fine at 120ms, but P99 is spiking to 3 seconds — something's choking under load.

For PMs, this means When users say "the app feels slow," ask for percentiles, not averages. Averages hide the worst experiences. A small percentage of users hitting 3-second loads can destroy trust — even if most users are fine.

Throughput

How much work the system handles at once

Throughput is the volume of work your system processes per unit time — usually requests per second (RPS) or transactions per second (TPS). High throughput doesn't automatically mean low latency. Under load, the two often trade off.

A highway. Throughput is how many cars pass per hour. Latency is how long any one car takes to get through. A jammed highway can have high throughput and miserable latency at the same time.

We're handling 2,000 requests per second, but above 3,000 latency starts climbing — we need to optimize before the product launch.

For PMs, this means throughput tells you capacity. Before a big launch, a marketing push, or a new enterprise customer, ask the expected load — and whether the system can handle it. Have this conversation before the launch, not during the incident.

Load testing

Testing the system before users do

Load testing simulates real-world traffic to verify your system handles expected volume. It's different from stress testing (pushing past limits to see what breaks) and soak testing (sustained load over hours to surface memory leaks and slow degradation).

A fire drill before a real fire. You don't wait until the building is burning to find out the alarms work. You test under controlled conditions so you trust the system under real ones.

We ran a load test at 5x current traffic — the database connection pool maxed out at 3x. We need to fix that before Black Friday.

For PMs, this means before a launch, a migration, or onboarding a large customer, ask if the team has load-tested for the expected volume. "It worked fine in staging" isn't the same as "we tested it at 10x current load.

Redundancy

Backup systems for when things break

Redundancy means duplicate components so there's no single point of failure. Active-active: all copies handle traffic; if one fails, the others absorb it. Active-passive: one handles traffic while the backup waits to take over. More redundancy means higher availability — and higher cost.

A plane with two engines. If one fails mid-flight, the other keeps you flying. It costs more to build and fuel — but you don't board a plane that bets your life on a single engine.

The database is active-passive — if the primary goes down, the replica promotes automatically, but there's about 30 seconds of read-only mode.

For PMs, this means redundancy is why your product stays up when things fail — and things always fail eventually. When your team asks for budget here, they're buying insurance. The question isn't "will we need it?" but "can we afford the downtime without it?

Failover

Automatic switching to a backup

Failover is when a system detects a failure and automatically switches to a backup. Hot: the backup is running, ready in seconds. Warm: the backup exists but needs startup — minutes. Cold: starting from scratch — longer. The goal is to recover so fast users barely notice.

A backup generator at a hospital. The moment the power flickers, it kicks on — surgery doesn't pause, monitors don't blink. Hot failover is the generator already running. Cold failover is calling the electrician.

Failover kicked in during the outage — users saw about 15 seconds of degraded performance, but no data loss.

For PMs, this means failover separates "the system went down for 2 hours" from "it hiccupped for 15 seconds." When reviewing architecture or SLAs, ask about the strategy. Hot costs more but recovers faster — a product tradeoff, not just a technical one.

SLA

Service Level Agreement — the promise you make

An SLA is a formal commitment to customers about service quality — usually uptime, response time, and support responsiveness. Behind it: SLOs (Service Level Objectives, internal targets, usually stricter than the SLA) and SLIs (Service Level Indicators, the actual measured metrics).

SLI → what you measure
SLO → what you aim for
SLA → what you promise

A pizza place's "30 minutes or it's free." The receipt promise is the SLA. The internal goal of 25 minutes is the SLO. The stopwatch on every order is the SLI.

Our SLO is 99.95% but the SLA says 99.9% — that gives us a buffer before we're in breach.

For PMs, this means SLAs aren't just legal documents — they're product commitments backed by engineering investment. If sales promises 99.99% but engineering targets 99.9%, someone has a bad quarter. Know the gap between SLO and SLA — that's your margin of safety.

Observability

Seeing what's happening inside the system

Observability is your ability to see inside your system from its outputs. Three pillars: logs (timestamped events), metrics (error rates, latency, CPU), and traces (a request's journey across services).

Logs → what happened
Metrics → how much
Traces → the journey of a request

A car's dashboard plus its diagnostic port. Metrics are the gauges (speed, fuel, temperature). Logs are the maintenance journal. Traces are plugging in to see exactly what every system did during the last drive.

We can see the error in the logs, but without traces we can't tell which service in the chain is causing the timeout.

For PMs, this means “we need better observability" means asking for visibility into problems before users report them. It's the difference between "users are complaining" and "we detected a 2am latency spike and fixed it before anyone noticed." Observability moves teams from reactive to proactive.

Incident response

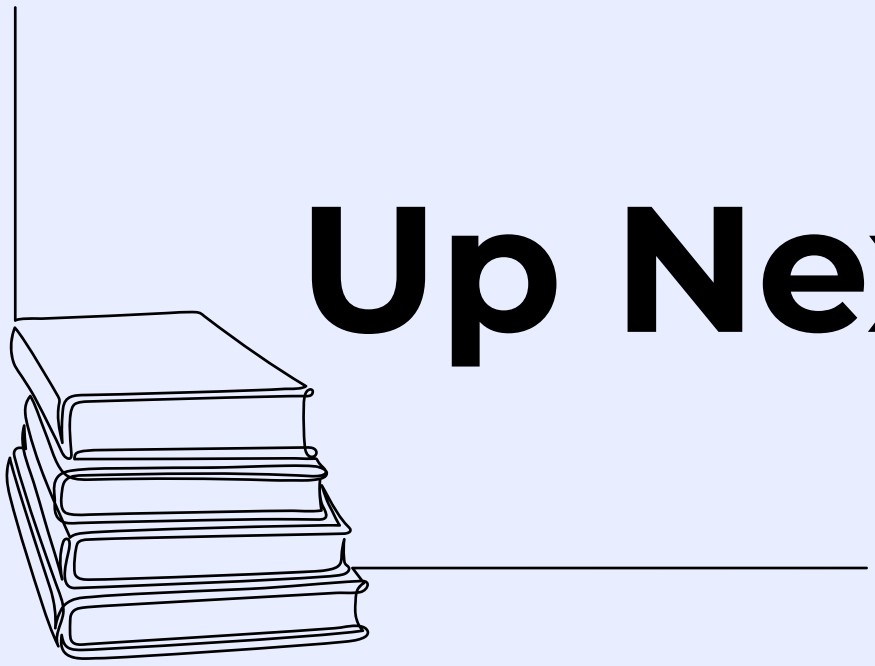
What happens when things go wrong

Incident response is the structured process for detecting, responding to, and recovering from failures. Severity levels (Sev 1: down, Sev 2: degraded, Sev 3–4: limited) set urgency. An on-call engineer gets paged. A blameless postmortem captures what to change.

A hospital ER. Sev levels are triage. The on-call engineer is the doctor on duty. The postmortem is the case review.

This is a Sev 1 — payments are down. Let's get the incident channel open and page the database on-call.

For PMs, this means incidents are inevitable. Your job isn't to prevent every outage — it's to know the process, communicate to stakeholders, and join the postmortem. Don't ask "when will it be fixed?" Ask "what's the customer impact?"



Stay Tuned

Up Next

Your system is live. Your metrics are flowing. But where does all that data actually go? Next up: a plain-English guide to data warehouses, pipelines, schemas, and ETL — the infrastructure that turns raw events into the numbers your dashboards depend on.