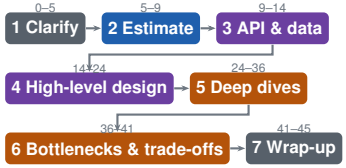
System Design: cheat sheet

The 16 classes on eight pages. Pages 1 to 3: the method, step by step, with its formulas and numbers, and the patterns that combine the components. Pages 4 to 6: the 49 components of the palette. Pages 7 and 8: class by class.

The method (a 45-minute interview, minutes in grey)



Say the steps out loud at the start. Go broad before deep; every number must end in a decision.

1 Clarify the requirements

class 1 · ≈ 5 min

- **Functional**: what it does (3–5 core features, and what you leave out). They decide the API and the data.
- **Non-functional**: how well. They decide the architecture. Each needs a **number, a percentile, a window and a load**: "hold p99 < 300 ms at 5 000 holds/s".
- Quality attributes: scalability, availability, latency, consistency, durability, cost, security, operability.
- **Scenario**: source → stimulus → environment → artifact → response → measure.
- Different data, different needs: seats and money strong; catalog 60 s stale; viewer counter approximate.
- If told "you decide", state an assumption: "10 M daily users, reads:writes 100:1".

BoxOffice targets: 2 M fans in the first minute for 60 000 seats; normal peak 1 500 catalog req/s. Catalog p99 < 200 ms, hold p99 < 300 ms, checkout p99 < 1.5 s; checkout 99.95%/30 days; 0 double bookings; RPO = 0 for paid orders; < R$ 0.10 per ticket; MTTR < 30 min.

Trap: adjectives ("fast", "scalable") instead of numbers; one target for everything; forgetting the peak.

2 Estimate

class 2 · ≈ 4 min

Decision → assumptions → powers of ten → sanity check → conclusion. Within 2–3× is enough.

2^{10}	$\approx 10^3$	KB: a short JSON; 10 seat rows
2^{20}	$\approx 10^6$	MB: a poster
2^{30}	$\approx 10^9$	GB: RAM of a small VM
2^{40}	$\approx 10^{12}$	TB: a large single-node database
2^{50}	$\approx 10^{15}$	PB: a data lake

1 day = 86 400 s ≈ 10^5 s; 1 month ≈ 2.5×10^6 s; 1 year ≈ 3×10^7 s.
1 M/day ≈ 12/s; 100 M/day ≈ 1 200/s; 1 B/day ≈ 12 000/s.
1 Gbit/s = 125 MB/s; 10 Gbit/s = 1.25 GB/s (bits ÷ 8).

$$\text{QPS}_{\text{avg}} = \frac{\text{DAU} \times \text{actions/day}}{86\,400}, \quad \text{peak} = 2\text{–}3\times \text{avg}$$

storage = items/day × size × retention × replicas (×1.5–2 indexes)
bandwidth = req/s × bytes per response

Latency numbers (2020s, ± 2–3×)

L1 cache / main memory reference	0.5 ns / 100 ns
read 1 MB sequentially from memory	10–50 µs
NVMe SSD random 4 KB read	10–100 µs
read 1 MB sequentially from NVMe	0.1–0.3 ms
round trip inside an availability zone (AZ)	0.1–0.5 ms
Redis GET over the network, same AZ	0.2–1 ms
round trip between AZs of a region	1–2 ms
indexed PostgreSQL lookup, end to end	1–5 ms
HDD seek	10 ms
light in fibre, 1 000 km one way	5 ms
São Paulo ↔ US East round trip	110–130 ms
São Paulo ↔ Western Europe	180–220 ms
TLS handshake on a new connection	+1–2 RTT
LLM time to first token; then	0.2–2 s; 30–150 tok/s

Memory is ≈ 1 000× faster than an SSD; a round trip in a data centre is ≈ 1 000× faster than one across an ocean. 50 sequential calls of 0.5 ms = 25 ms: batch them.

Throughput of one instance (orders of magnitude)

Nginx/Envoy, small responses	10^4–10^5 req/s
app server, CRUD with one DB call	10^2–10^4 req/s
PostgreSQL indexed reads / write tx	10^4–10^5 / 10^3–10^4
Redis (single instance)	$\approx 10^5$ ops/s
Kafka broker	10^5–10^6 msgs/s
WebSocket server	10^4–10^5 connections
DynamoDB partition	3 000 RCU / 1 000 WCU
S3 prefix	3 500 writes / 5 500 reads/s
Lambda per region (default)	1 000 concurrent

Sizes

UUID 16 B (36 as text) · timestamp 8 B · seat row ≈ 100 B · order or user row 0.5–1 KB · log line 0.2–2 KB · embedding 1 536 float32 ≈ 6 KB · web page 50–200 KB · photo 0.2–3 MB · 1 min of 1080p video ≈ 38 MB.

The on-sale on a napkin (BoxOffice)

2 M fans refresh every ≈ 15 s	≈ 130 000 req/s
behind a CDN with 99% hits	≈ 1 300 req/s at origin
edge bandwidth, 100 KB per view	≈ 13 GB/s ≈ 100 Gbit/s
hold attempts, 2 M in 60 s	≈ 33 000/s
DB: $c = 8$ slots, $S = 5$ ms	1 600/s; 1 100/s at 70%
waiting room admits	1 000 fans/s
checkout sessions (Little)	33/s × 5 min ≈ 10 000
payments in flight	17/s × 3 s ≈ 50
e-mails, 30 000 tickets in 30 min	≈ 17/s

Trap: designing for the average, false precision ("1 157.4 req/s"), numbers without a conclusion.

Load, latency and capacity

classes 2 and 15

Little $L = \lambda W$ **utilization** $\rho = \frac{\lambda S}{c}$ $X_{\max} = \frac{c}{S}$

L in the system, λ arrival rate, W time in the system, S service time, c parallel slots. 1 000/s × 40 ms = 40 in flight; 8/5 ms = 1 600/s.

M/M/1 $R = \frac{S}{1-\rho}$, p99 ≈ $4.6\,R$

$S = 5$ ms: ρ 50% → 10 ms; 70% → 17; 90% → 50 (p99 230); 95% → 100; 99% → 500 ms. The knee: 70–80% for one server; more servers move it right.

- Run latency-sensitive tiers at **50–70%** at peak (batch 80–90%): 4 replicas at 60% lose one → 80%; at 80% → 107%.
- **Closed** load test: $N = X(R + Z)$ (users, throughput, response, think time). 100 users, $R = 50$ ms → 2 000 req/s; at 500 ms only 200. Real fans are an **open** model: arrivals do not wait.
- **Amdahl**: speedup ≤ $1/\sigma$ (5% serial → 20×). **USL**: $C(N) = \frac{N}{1+\sigma(N-1)+\kappa N(N-1)}$ peaks at $N^* = \sqrt{(1-\sigma)/\kappa}$, then falls.

$$\text{replicas} = \left\lceil \frac{\text{peak}}{\text{capacity per replica} \times \rho_{\text{target}}} \right\rceil + \text{spares}$$

$4\,000/(500 \times 0.65) \to 13$, $+1 = 14$. Plan with the knee, not the maximum, and add zone-loss replicas. Check shared resources: 500 DB connections ÷ 20 per replica = 25 replicas at most.

Percentiles and the tail

classes 2 and 9

$P(\text{slow}) = 1 - (1-q)^N$ (N calls in fan-out, each slow with probability q)

$q = 1\%$: $N = 5 \to 4.9\%$; $20 \to 18\%$; $100 \to 63\%$; $500 \to 99\%$.

- Percentiles come from **histograms**; never average them (p99s of 100 and 1 000 ms are not 550). Sum the buckets, then compute.
- Tail tools: headroom, hedging after p95 (idempotent reads only), timeouts, partial results, warm pools.

Trap: an 80 ms average can hide 5% of fans waiting 4 s.

Availability, SLOs and error budgets

classes 2, 9 and 10

$A = \dfrac{\text{MTTF}}{\text{MTTF} + \text{MTTR}}$ series $\prod a_i$ parallel $1 - \prod(1 - a_i)$

3 at 99.9% in series → 99.7%; 2 replicas at 99% → 99.99% (if independent); 2-of-3 quorum $= a^3 + 3a^2(1-a)$. MTTF 720 h, MTTR 2 h → 99.72%; MTTR 2 min → 99.995%: faster repair beats fewer failures.

	year	month	28 days	day
99%	3.65 d	7.31 h	6.72 h	14.4 min
99.9%	8.77 h	43.8 min	40.3 min	1.44 min
99.95%	4.38 h	21.9 min	20.2 min	43.2 s
99.99%	52.6 min	4.38 min	4.03 min	8.64 s
99.999%	5.26 min	26.3 s	24.2 s	0.86 s

Year = 365.25 days, month = 30.44 days. Always say the window. Rule of the extra nine: in a chain of ≈ 5 links, each link needs one nine more than the target.

$\text{SLI} = \dfrac{\text{good events}}{\text{valid events}}$ error budget $= 1 - \text{SLO}$

burn rate $b = \dfrac{\text{observed error ratio}}{1 - \text{SLO}}$ budget spent $= \dfrac{b \cdot t}{\text{window}}$

99.9% over 28 days: 0.001 × 28 M requests = 28 000 failures, or 40.3 min. Burn 14.4 empties the 28-day budget in 1.9 days.

	burn	long	short	budget when it fires
page	14.4×	1 h	5 min	2%
page	6×	6 h	30 min	5%
ticket	1×	3 d	6 h	10%

Both windows must exceed the threshold: the short one makes the alert reset fast. Page on symptoms users feel, never on causes (CPU).

3 API and data model

classes 5 and 6 · ≈ 5 min

- 4–6 endpoints for the core features: `GET /events`, `GET /events/{id}/seats`, `POST /holds`, `POST /orders` (with `Idempotency-Key`), `POST /payments/webhook`, `GET /orders/{id}`.
- **REST** for public resources; **gRPC** between services (HTTP/2, schemas); **GraphQL** for client-shaped reads (BFF); **async** (events) when nobody waits for the answer. Version without breaking shipped clients.
- Entities with their keys and access patterns: event, seat (status, version), hold (expires in 10 min), order (idempotency key), ticket.
- **SQL by default** for invariants and joins; **NoSQL** for huge write rates with simple key access; object store for blobs; search engine, warehouse and vector DB as derived copies.

No double booking: `UPDATE seat SET status='HELD' WHERE id IN (...) AND status='FREE'`; rows updated ≠ requested ⇒ roll back, 409. A unique constraint is the last line of defence. A hot row caps throughput at $1/t$ (5 ms → 200/s).

Trap: check-then-act (`SELECT`, then `UPDATE`) under read committed: two fans get seat A-12.

Consistency, replication and partitioning

Quorum $R + W > N$ $(N = 3, W = 2, R = 2)$ **consensus** $2f + 1$ nodes

Read and write sets overlap, so a read sees the latest write. 3 nodes tolerate 1 failure, 5 tolerate 2; use odd N spread so no zone holds a majority.

- Models, strong to weak: **linearizable** (seats, money) → sequential → causal → **read-your-writes** (a fan's own orders) → **eventual** (catalog, counters).
- **CAP**: during a partition choose consistency or availability. **PACELC**: else, choose latency or consistency, on every write (≈ 1–2 ms across AZs, 150–200 ms across continents).
- **Replicas** scale reads, never writes, and lag (ms, seconds under load): read your own writes from the primary.
- **Partition** by a high-cardinality key read together (`event_id`, `section`). **Consistent hashing** with virtual nodes moves ≈ $1/(N+1)$ of the keys when a node joins; `hash mod N` moves almost all.
- Across services: **sagas** with compensations and the **outbox**, not two-phase commit. Order comes from one authority or a per-key log, never from wall clocks of different machines.

Trap: believing in exactly-once delivery: it is at-least-once + idempotent processing.

4 High-level design

classes 4 to 8 · ≈ 10 min

Draw the happy path first, left to right, with the palette: client → DNS → CDN → WAF → load balancer / API gateway → stateless app servers → cache → SQL; the slow side branches go through a queue or a stream to workers; external providers behind adapters; observability on every box.

Scale in steps: symptom → add → new problem

one box fails all	separate SQL	a network hop
CPU 85% at peak	LB + stateless replicas + health check	sessions leave the server
DB busy, 95% reads	read replicas	lag, stale reads
same 5 queries	cache	invalidation, stampede
1 GB/s of images	CDN + object store	edge invalidation
checkout 2–3 s; spike	queue + workers + waiting room	duplicates, backlog
5 teams, 1 codebase	services + gateway + events	network failures
billions of rows	partitions + search + warehouse	hot partitions
bots; region down	WAF, rate limits, multi-region	cross-region data

Add a component only for a measured symptom. Monolith (modular) first.

5 Deep dives: the math of the hard parts

≈ 12 min

Cache DB load $= (1-h)\,\lambda$ $\bar{t} = h\,t_{\text{hit}} + (1-h)\,t_{\text{miss}}$

50 000 reads/s: h 90% → 5 000 DB QPS; 99% → 500; each nine is 10× less. TTL caps $h \le 1 - 1/(rT+1)$ (r reads/s of a key, T TTL). Stampede = reads/s × recompute time (20 000 × 0.2 s = 4 000 queries). Size for the working set: 20 000 events × 4.1 KB ≈ 82 MB.

Queue backlog $= (\lambda - \mu)\,t$ wait $= \dfrac{\text{backlog}}{\mu}$ workers $= \lambda S$

6 000/s against 2 000/s for 3 min: 720 000 waiting, the last waits 360 s. Drain = backlog ÷ (consume − arrive). Divide by the drain rate, never the arrival rate.

Kafka partitions $\ge \max\left(\dfrac{T}{p}, \dfrac{T}{c}\right)$ consumers per group ≤ partitions

T target rate, p and c per-partition producer and consumer rates. 167 events/s, 20/s per consumer → ≥ 9 (choose 24). 6 partitions, 10 consumers → 4 idle. Order only per key.

Resilience in flight $= \lambda \times$ latency

100/s: 200 ms → 20 threads; 4 s → 400 > 200: cascade. Timeout ≈ p99.9 (1.2 s → 2 s); 1 retry, backoff $\min(b \cdot 2^n, \text{cap})$ with full jitter; breaker 50% of 20 calls, open 30 s; bulkhead 50. Retries at 3 layers: $3^3 = 27$ calls.

LLM cost $= \sum_i (C_i\,p_{\text{in}} + O_i\,p_{\text{out}})$, $C_{i+1} = C_i + O_i + R_i$

Context grows each step: input is ≈ quadratic in the steps. Latency = TTFT + TPOT $(N_{\text{out}} - 1)$: 0.6 s + 20 ms × 249 ≈ 5.6 s. Agent success p^n: $0.95^{10} \approx 60\%$. KV cache 128 KiB/token: 8 000 tokens ≈ 1 GiB.

Recovery RTO ≈ $t_{\text{detect}} + t_{\text{decide}} + t_{\text{promote}} + t_{\text{scale}} +$ TTL

$2 + 5 + 1 + 5 + 1 \approx 14$ min. RPO: data lost, in time. Replicas copy a `DELETE` in ms: only backups with point-in-time recovery undo it. 2 TB at 200 MB/s ≈ 2.8 h to restore.

6 Bottlenecks: symptom → cause → what to do ≈ 5 min

symptom	cause	what to do
p99 explodes, CPU idle, threads busy	slow dependency (Little)	timeout, breaker, bulkhead, async
latency rises long before errors	past the knee ($\rho > 0.7$)	capacity, shed load, waiting room
DB CPU high, mostly reads	no cache, low hit ratio	cache-aside, replicas, CDN
DB spikes when a hot key expires	stampede	coalescing, TTL jitter, stale-while-revalidate
one partition hot, the others idle	hot key	salt the key, cache it, admission control
my update is not there	read from a lagging replica	read-your-writes from the primary
consumer lag keeps growing	consumers < load or = partitions	more consumers, more partitions, faster handler
duplicate e-mails or charges	at-least-once, no idempotency	idempotency key, dedupe in the same tx
never recovers after a short outage	retry storm (metastable)	backoff + jitter, retry budget, breaker
errors for 30 s after a crash	no health check	readiness probes, short intervals
minutes of errors after failover	DNS TTL, detection	lower TTL, faster checks, rehearse
restarts everywhere on a DB blip	DB in the liveness probe	shallow liveness, local readiness
429 at a burst (serverless)	concurrency limit, cold starts	raise limit, provisioned instances
green dashboards, angry users	causes and averages	SLOs on percentiles, tracing
LLM slow, 429, expensive	too many calls, long context	fewer steps, prefix cache, fallback, quotas
invoice surprise	egress, NAT, logs	CDN, private endpoints, sampling

Trade-offs to say out loud

- Cache: −latency and −DB load, +staleness (price up to 60 s old).
- Replicas: +reads, +lag. Sync replication: +durability, +write latency.
- Queue: +smooth load, +eventual consistency, duplicates.
- Strong consistency: correctness, at the cost of latency and availability in a partition.
- Second region: survives a region, ≈ 2× the cost.
- Every quantified: "adds ≈ 1 ms per request, cuts DB load 10×".

7 Wrap-up ≈ 3 min

- Re-read the requirements and say how each is met, with its number.
- What you would measure: an SLO per journey, a burn-rate alert, one metric specific to the problem (hit ratio, consumer lag, replication lag).
- What fails first at 10×, and what you would do next.
- What you left out on purpose.

Trap: defending every decision as perfect: name the weakness before the interviewer does.

Security and network in one breath classes 4 and 12

- VPC per region; subnets in ≥ 2 AZs: public (LB, NAT), private (apps), isolated (data). /16 = 65 536 addresses, /24 = 256 (AWS reserves 5).
- **STRIDE** at every trust boundary: spoofing, tampering, repudiation, information disclosure, denial of service, elevation of privilege.
- OAuth 2.1: Authorization Code + PKCE; access JWT ≈ 15 min, validated locally (revocation window = lifetime).
- **Zero trust**: mTLS between services *and* an authorization policy; authorize in the owning service (BOLA); secrets in a manager, rotated; signed images, SBOM.

Trap: "the gateway validated the JWT, so we are safe": authentication is not authorization.

Delivery and cost classes 11 and 14

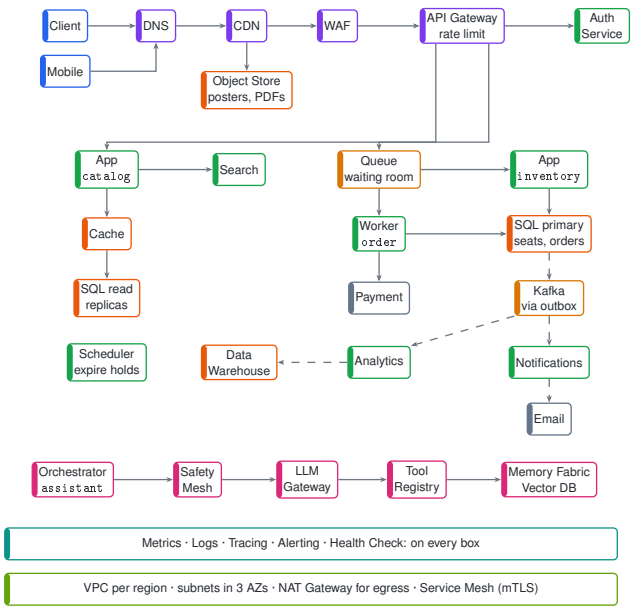
- Images small, immutable, by digest; probes local; requests sized; HPA: $\lceil \text{replicas} \times \text{current}/\text{target} \rceil$ ($\lceil 4 \times 90/60 \rceil = 6$). Autoscaling lags (pods 15 s steps, nodes minutes): pre-scale before 10:00.
- GitOps; canary 5% → 25% → 50% → 100% gated on error rate and p99 against the baseline; feature flags.
- Cost = capacity × time. Data services, egress, logs and fixed hourly items (NAT, LB) dominate, not compute.
- AI in the pipeline: AI proposes, rules and humans decide. Precision beats recall; budgets live in the harness; judge by dollars per merged fix.

Measure it class 15

- Profiles: smoke, load, stress, spike, soak, breakpoint. Open model for real traffic; watch `dropped_iterations` = 0.
- **Coordinated omission**: a closed test waits with the system and hides the queue (p99 1 ms vs 1 366 ms).
- Find the bottleneck: RED (rate, errors, duration) per service → USE (utilization, saturation, errors) per resource → profile. Change one thing at a time.

BoxOffice on one page the reference design

The on-sale design that the classes build, drawn with the palette. Solid arrows: the fan waits; dashed: asynchronous.



Read path: CDN, cache, replicas (eventual, seconds stale). Purchase: waiting room at the speed of the database and the provider; seats by conditional update in one primary (strong); payment with an idempotency key. Everything else is an event.

Before you trust a design: is every number tied to a decision? Is the peak (not the average) sized, with headroom for a lost node? Is every retry idempotent? Does every call have a timeout? Is strong consistency only where money and seats need it? What does the user see when each box fails?

Patterns that combine the components classes 7, 8, 10, 13

- **Caching** (7): cache-aside (miss: read DB, fill; write: commit, then *delete*; TTL + jitter); write-behind may lose data; L1 + Redis L2. Cache what you display, not what you decide on.
- **Messaging** (8): waiting room admitting at the bottleneck's rate; outbox, never dual writes; idempotent consumer (dedupe in the same tx, retries, DLQ); saga with compensations.
- **Resilience** (10): timeout → one jittered retry with an idempotency key → breaker on slow calls → bulkhead by Little → honest fallback; shed load by priority.
- **AI** (13): workflow first, capped agent; gateway with fallbacks and token quotas; RAG with hybrid search and rerank; live data through tools; least privilege, confirm irreversible actions.

The palette, component by component

The 49 components of the simulator in 10 categories: what, when, key numbers ("sim": the playground default per replica), how it fails, products.

Client

who sends the traffic? · classes 1 and 4

Client — client

Is	a browser, partner backend or load tester: you can neither patch nor trust it.
Use	as the workload; what the user feels is DNS + TLS + round trips.
Numbers	RTT São Paulo–US East ≈ 110–130 ms; nearby edge 5–20 ms; TLS 1.3 adds 1 RTT; "instant" ≈ 100 ms; 2 M fans in ≈ 1 min ≈ 33 k sessions/s.
Fails	retry storms (backoff + jitter, Retry-After); double submits (idempotency key).
Products	k6, Locust, Gatling; AWS/GCP/Azure SDKs; synthetic checks.

Mobile — mobile

Is	a native or mobile-web app: flaky network, battery, old versions, offline, push.
Use	push, offline survival (QR ticket at the gate), a second traffic mix.
Numbers	RTT 4G 30–100, 3G 100–500, 5G 10–30 ms; push in seconds, best effort.
Fails	drop mid-purchase (idempotency key + status endpoint); reconnect storms (jitter).
Products	Firebase/FCM, APNs, Amplify, Notification Hubs.

Traffic & edge

how does a request get in? · class 4

DNS — dns

Is	name → IP, cached for the TTL; managed DNS adds geo, weighted and failover routing; anycast.
Use	always; the decision is the policy: failover, latency/geo, weighted canary.
Numbers	lookup cached 1–10 ms, full 20–120 ms; TTL 60–300 s. Failover ≈ interval × threshold + TTL = $3 \times 10 + 60$ s ≈ 1.5 min. Sim TTL 60 s.
Fails	provider outage (second provider); slow failover from cached answers.
Products	Route 53 · Cloud DNS · Azure DNS · CoreDNS, BIND.

CDN — cdn

Is	edge caches (PoPs) near users; origin only on a miss; TLS and DDoS shield.
Use	static assets, media, video, public read-heavy pages (TTL of seconds).
Numbers	edge hit 5–30 ms vs 100+ ms to a far origin; static hit > 90%; origin = $(1-h)\times$ requests; egress 0.02–0.09 USD/GB. Sim $h = 0.9$.
Fails	cold cache, purge storm (versioned URLs, origin shield); caching private data.
Products	CloudFront · Cloud CDN · Front Door · Cloudflare, Fastly, Varnish.

Load Balancer — lb

Is	spreads requests over healthy replicas; L4 (TCP) or L7 (HTTP: paths, TLS).
Use	as soon as there are two replicas: scale-out and zero-downtime deploys.
Numbers	adds < 1 ms; health check every 5–30 s, 2–3 failures to remove. Sim: detection 30 s without a Health Check, 3 s with one.
Fails	traffic to a dead replica until detected; a health check that lies (split liveness/readiness).
Products	ALB/NLB · Cloud Load Balancing · Azure LB · HAProxy, Nginx, Envoy.

WAF — waf

Is	HTTP firewall at the edge: blocks SQLi, XSS, bots (managed + custom rules).
Use	in front of login, search and checkout; scalper bots; virtual patch; L7 DDoS.
Numbers	adds 1–3 ms; blocked requests never reach the origin. Sim blocks 95% of bots.
Fails	false positives on real purchases; bypass by calling the origin directly.
Products	AWS WAF · Cloud Armor · Azure WAF · ModSecurity + OWASP CRS.

API Gateway — apigw

Is	the API's front door: routing, JWT checks, rate limits, quotas; a BFF per client.
Use	several services behind one API; per-user limits (429 + Retry-After); partners.
Numbers	adds 1–10 ms; Redis rate-limit check 0.5–1 ms. 429 = you are throttled, 503 = I am overloaded. Sim 10 k req/s.
Fails	the gateway is the bottleneck; no per-route timeouts; limiter store down (fail open or closed?).
Products	API Gateway · Apigee · API Management · Spring Cloud Gateway, Kong.

Ingress — ingress

Is	Kubernetes rules for external HTTP(S) into Services, run by a controller; successor: Gateway API.
Use	host/path routing, TLS with cert-manager, canary splits, GitOps routes.
Numbers	adds ≤ 1 ms; ≥ 2 controller replicas across zones; certificates 90 days.
Fails	under-provisioned controller; expired certificate; a rule shadowing a route.
Products	AWS LB Controller · GKE Gateway · AGC · Traefik, Envoy Gateway.

Compute

where does the code run? · class 5

App Server — app

Is	stateless replicas running the business logic for requests a user waits on.
Use	every synchronous API; modular monolith first; LB + autoscale at 50–70% CPU.
Numbers	limit = threads ÷ time per request: $200 \div 0.1$ s = 2 000 req/s. Sim 500 req/s, 10 ms, 120 USD/mo.
Fails	threads exhausted by a slow dependency (timeouts, breakers, bulkheads); past the knee as $\rho \to 1$.
Products	EC2, ECS, EKS · Cloud Run, GKE · App Service, AKS · Spring Boot.

Worker — worker

Is	processes pulling jobs from a queue or stream; the producer returns at once.
Use	slow or unreliable work nobody waits for (e-mail, PDF, transcoding); spikes.
Numbers	workers = $\lambda\times$ job time: $500/\text{s} \times 0.2$ s = 100; drain = backlog ÷ (consume − produce): $600\text{k} \div 1\,000/\text{s} = 10$ min. Sim 100 jobs/s, 50 ms.
Fails	poison message (max receives + DLQ); duplicates (dedupe table); hidden lag (alert on it).
Products	Lambda + SQS · Cloud Run jobs · Functions + Service Bus · Celery, KEDA.

Serverless — serverless

Is	functions run per event, scale from zero, billed per call and per GB-second.
Use	spiky, short, event-driven work: webhooks, thumbnails, nightly jobs, glue.
Numbers	cold start 0.1–1 s (JVM: seconds); Lambda 15 min max, 1 000 concurrent per region (default). Instances = $\lambda \times W$. Sim cold start 600 ms.
Fails	throttling at the concurrency limit (429); DB connection storm (proxy).
Products	Lambda · Cloud Run functions · Azure Functions · Knative, OpenFaaS.

Auth Service — auth

Is	authenticates users; issues short-lived JWTs and refresh tokens (OIDC).
Use	any system with accounts; validate at the gateway, authorize in the services.
Numbers	local JWT check ≈ µs; password hash ≥ 100 ms; access token 5–15 min.
Fails	auth down = logged-in features down; login storm amplified by slow hashing.
Products	Cognito · Identity Platform · Entra ID · Keycloak, ZITADEL.

Search — search

Is	ranked full-text, fuzzy and faceted queries over an inverted index; a secondary copy of the DB.
Use	event search, autocomplete, facets; hybrid keyword + vector.
Numbers	query 10–100 ms; refresh 1 s (near real time); autocomplete < 100 ms.
Fails	index out of sync (CDC/outbox + reconciliation); expensive queries; hot shards.
Products	OpenSearch Service · Vertex AI Search · Azure AI Search · OpenSearch.

Scheduler — scheduler

Is	triggers work by time (cron, delays) with one run guaranteed by a lock or leader.
Use	release expired holds, reminders, opening the sale at 10:00, batch pipelines.
Numbers	cron granularity 1 min; lock lease a few × job time. Sim: all jobs at once = thundering herd.
Fails	double run (fencing token, idempotent jobs); missed run (dead-man's switch); herd at round hours (jitter).
Products	EventBridge Scheduler · Cloud Scheduler · timer triggers · K8s CronJob.

Notifications — notifications

Is	turns "tell user X" into push, e-mail or SMS: preferences, templates, dedupe.
Use	many services reaching users; isolation from provider quirks.
Numbers	push in seconds; e-mail seconds to minutes; provider quotas per second/day.
Fails	provider throttling (queue + backoff, second provider); duplicates; storms (kill switch).
Products	SNS/SES · FCM · Notification Hubs · Novu.

Analytics — analytics

Is	events → stream → real-time counters and a warehouse for reports and funnels.
Use	business metrics, A/B tests, live sale counters.
Numbers	2 M fans × 20 events ≈ 40 M events in the sale hour; streaming seconds, batch hours; HyperLogLog 1–2% error.
Fails	backlog delays dashboards; double counting under at-least-once.
Products	Firehose, Redshift · BigQuery, Dataflow · Event Hubs · Kafka + Flink, ClickHouse.

External

services we don't run · class 5

3rd Party API

thirdparty

Is	someone else's API: a contract, a rate limit, a price, no access to their metrics.
Use	buy, don't build, what is not your differentiator.
Numbers	50–500 ms with a long tail; timeout a bit above p99 (p99 800 ms ⇒ 1 s).
Fails	slow calls exhaust threads (timeout, breaker, bulkhead); 429 (backoff with Retry-After).
Products	Resilience4j, Toxiproxy, WireMock; API Management as a proxy.

Payment

payment

Is	a PSP: synchronous create/confirm (maybe "pending") + a signed asynchronous webhook.
Use	always for real money, behind an adapter (authorize, capture, refund, webhook).
Numbers	authorization 0.5–3 s; webhooks late, twice or out of order; idempotency keys kept $\geq$ 24 h; seat hold 10 min.
Fails	double charge (one idempotency key per attempt); lost webhook (reconciliation).
Products	Stripe, Adyen, Mercado Pago, PagBank, Pix PSPs; Hyperswitch.

Email

email

Is	delivery provider with reputation-managed infrastructure and bounce webhooks.
Use	transactional mail (ticket, receipt, reset), always via a queue.
Numbers	60 000 orders in 15 min $\approx$ 67 e-mails/s; $\approx$ 0.10 USD per 1 000; bounces $<$ 2–5%, complaints $<$ 0.1%.
Fails	throttling (queue absorbs it); duplicates (key: order id + template); spam folder.
Products	SES · Communication Services · SendGrid, Postmark · Mailpit (dev).

Containers

how is it packaged and placed? · class 11

Microservice

microservice

Is	one bounded context, deployed and scaled on its own, with its own database.
Use	when teams or scaling diverge, after a modular monolith shows the seams.
Numbers	a hop 0.5–2 ms; in a sync chain latencies add and availabilities multiply ($0.995^5 \approx 97.5\%$). Sim 400 req/s, 8 ms, warm-up 20 s.
Fails	distributed monolith (shared DB); cascades along sync chains; chatty calls.
Products	EKS, ECS · Cloud Run, GKE · Container Apps, AKS · Spring Boot, gRPC, Dapr.

Container

container

Is	a process from an OCI image, isolated by namespaces and limited by cgroups; no guest kernel.
Use	every service: the same image from laptop to production; replaced in seconds.
Numbers	start 0.1–1 s with the image cached, pull 1–10 s, Spring Boot ready 5–30 s. Capacity $\times$ CPU limit: 0.5 ⇒ half the req/s, 2$\times$ slower. Sim 300 req/s, start-up 20 s, limit 1.
Fails	CPU throttling (p99 up, no errors); OOMKilled loop; big images scale slowly.
Products	ECR, Fargate · Artifact Registry · ACR, ACI · Docker, containerd, Podman.

Kubernetes Cluster

k8s

Is	nodes running pods; the HPA adds pods, the cluster autoscaler adds nodes.
Use	many services on one platform: self-healing, rolling updates, autoscaling, GitOps.
Numbers	room = nodes $\times$ pods per node, the rest Pending; node NotReady after 40 s; new pod 5–30 s, new node 1–3 min. Sim 3 nodes $\times$ 10 pods, node start 90 s, 140 USD/node-month.
Fails	Pending pods (no room); a lost node takes its pods (N+1 nodes); slow node scale-up.
Products	EKS · GKE · AKS · Kubernetes, Karpenter, k3s, kind.

Service Registry

registry

Is	service discovery: instances register and heartbeat; callers get the healthy ones.
Use	services calling each other while replicas come and go.
Numbers	heartbeat/TTL 1–10 s; CoreDNS TTL 5 s; 3 or 5 servers (Raft). Sim: callers drop a dead replica in 3 s (not 30 s, or 40 s on a lost node).
Fails	stale entries (DNS caching); registry outage (replicate, cache the list).
Products	Cloud Map · Service Directory · K8s Services + CoreDNS, Consul, Eureka, etcd.

Storage

where does the data live? · classes 6 and 7

SQL Database

sql

Is	tables, SQL and ACID transactions: the default system of record.
Use	relations and invariants ("a seat is sold at most once"), joins, ad-hoc queries.
Numbers	point read 0.1–1 ms (+ RTT); 10^3–10^4 write tx/s; `max_connections` 100; failover 30 s–min. Sim 3 000 req/s.
Fails	connection exhaustion (PgBouncer); hot row in a flash sale (short tx, `SKIP LOCKED`, waiting room).
Products	RDS, Aurora · Cloud SQL, AlloyDB, Spanner · Azure SQL · PostgreSQL, MySQL.

DB Cluster

dbcluster

Is	shards, each a primary plus followers copying its log; failover promotes one.
Use	a system of record that must survive a node or an AZ; shards for writes.
Numbers	async: lost writes $\approx$ write rate $\times$ lag (RPO); sync/quorum: RPO 0, +1–2 ms per write; failover $\approx$ 30 s (RTO). Sim 1 shard, 2 followers, async, write = 3 reads, 450 USD/node.
Fails	lost writes on failover; stale reads; hot shard; split brain.
Products	Aurora · Spanner, AlloyDB · Cosmos DB · Patroni, Vitess, CockroachDB.

NoSQL DB

nosql

Is	key-value, document, wide-column or graph; partitioned by key and replicated.
Use	very high write rates with simple key access: locations, chat, carts, timelines.
Numbers	single-key op 1–10 ms; scales with partitions; DynamoDB 400 KB items, 3 000 RCU/1 000 WCU per partition; RF 3 ($W = R = 2$).
Fails	hot key (salt the key, cache it); stale reads under eventual consistency.
Products	DynamoDB · Firestore, Bigtable · Cosmos DB · Cassandra, MongoDB.

Cache

cache

Is	in-memory copy of hot data, usually cache-aside with a TTL.
Use	read-heavy, slow-changing data (catalog); expensive results; sessions and counters.
Numbers	Redis GET 0.2–1 ms; $\approx 10^5$ ops/s per node; DB load = $(1-h)\times$ reads (90% → 99% = 10$\times$ fewer); $\bar{t} = 0.95 \times 1 + 0.05 \times 20 \approx 2$ ms.
Fails	stampede on a hot key (coalescing, jittered TTL, stale-while-revalidate); cold cache after a flush.
Products	ElastiCache, MemoryDB · Memorystore · Azure Managed Redis · Redis, Valkey, Memcached.

Object Store

objstore

Is	buckets of whole objects by key over HTTP: cheap, unlimited, very durable.
Use	posters, PDF tickets, uploads, video, backups, data lake.
Numbers	11 nines durability; first byte 10–100 ms; S3 3 500 writes/5 500 reads per s per prefix; $\approx$ 0.02 USD/GB-month.
Fails	public exposure (block public access); egress and tiny-request costs (CDN, lifecycle).
Products	S3 · Cloud Storage · Blob Storage · MinIO, Ceph.

Data Warehouse

dwh

Is	columnar OLAP store for scans and aggregations over history, fed by ETL/ELT or CDC.
Use	dashboards over months, joins of many sources, ML features.
Numbers	seconds to minutes per query; compression 5–10$\times$; BigQuery $\approx$ 6 USD/TiB scanned.
Fails	analytics on the production DB; late pipelines (freshness alerts); runaway query cost.
Products	Redshift, Athena · BigQuery · Fabric, Databricks · ClickHouse, Trino.

Vector DB

vectordb

Is	stores embeddings and answers approximate nearest-neighbour queries (HNSW, IVF).
Use	RAG retrieval, semantic search, semantic cache, agent memory.
Numbers	384–3 072 dims; size = dims $\times$ 4 B (1 536 → 6 KB; 1 M → 6 GB); ANN 1–10 ms; recall@10 0.9–0.99.
Fails	right chunk not in top-k (hybrid search, rerank); cross-tenant leakage (mandatory filters).
Products	OpenSearch, pgvector · Vertex Vector Search · Azure AI Search · Qdrant, Milvus.

Messaging

how do services talk without waiting? · class 8

Message Queue

queue

Is	holds messages until one of several competing consumers acks them; failures go to a DLQ.
Use	background jobs, spike absorption, load levelling (waiting room).
Numbers	enqueue ms; drain = backlog ÷ (consume − arrive): 120k ÷ 2k/s = 60 s; SQS visibility 30 s, retention 4 days (max 14).
Fails	poison message (max receives → DLQ); duplicates after a crash before the ack.
Products	SQS · Cloud Tasks · Service Bus · RabbitMQ, BullMQ.

Pub/Sub — pubsub

Is	publish once to a topic; every subscription gets a copy (each acts as its own queue).
Use	fan-out of events (EventCancelled → refunds, notifications, search); chat delivery; cache invalidation.
Numbers	publish-to-deliver tens of ms; 1 publish → N deliveries; GCP retention 7 days.
Fails	slow subscriber builds its own backlog; fire-and-forget loses messages.
Products	SNS, EventBridge · Pub/Sub · Service Bus topics · NATS, Redis Pub/Sub.

Event Stream — stream

Is	append-only, partitioned, retained log; each consumer group reads at its own offset and can replay.
Use	many consumers of the same facts, CDC, event sourcing, ingestion.
Numbers	Kinesis shard 1 MB/s or 1 000 rec/s in; retention 24 h; parallelism per group = partitions; watch consumer lag.
Fails	lag grows (more consumers, up to the partitions); lag beyond retention loses events.
Products	Kinesis · Pub/Sub ordering keys · Event Hubs · Pulsar, NATS JetStream.

Kafka — kafka

Is	topics of replicated partitions, consumer groups committing offsets, retention or compaction (KRaft).
Use	event backbone, outbox + CDC (Debezium), high-volume pipelines with replay.
Numbers	retention 7 days; RF 3, min ISR 2, `acks=all`; 6 partitions, 10 consumers ⇒ 4 idle.
Fails	rebalance storms; poison message blocks a partition (retry + DLQ topics); duplicates before commit.
Products	MSK · Managed Kafka · Event Hubs · Kafka, Strimzi, Confluent.

Observability

how do we know? (side channels) · class 9

Metrics — metrics

Is	numeric time series with labels: counters, gauges, histograms. RED, USE, golden signals.
Use	always: SLIs, SLOs, dashboards, alerts, autoscaling.
Numbers	scrape 15–60 s; series = metrics × label combos × instances ($50 \times 200 \times 20 = 200$ k); 1–2 B/sample.
Fails	cardinality explosion; monitoring that dies with the system.
Products	CloudWatch · Cloud Monitoring · Azure Monitor · Prometheus, Grafana.

Logs — logs

Is	timestamped event records, structured JSON with a `trace_id`, in a central store.
Use	investigating one request or user, errors with context, audit.
Numbers	2 000 req/s × 5 lines × 500 B = 5 MB/s ≈ 430 GB/day; hot retention 7–30 days.
Fails	log flood in an incident (sample); no correlation id; secrets leaked.
Products	CloudWatch Logs · Cloud Logging · Log Analytics · Loki, OpenSearch.

Tracing — tracing

Is	one request across services as a tree of spans under a trace id (W3C `traceparent`).
Use	where latency or errors come from; retries, fan-out, agent runs.
Numbers	tens to hundreds of spans per trace; head sampling 1–10%; µs per span.
Fails	broken traces; sampling hides the rare slow request (tail sampling).
Products	X-Ray · Cloud Trace · App Insights · OpenTelemetry, Jaeger, Tempo.

Alerting — alerting

Is	rules that page or ticket a human when users are affected: multi-window SLO burn rates.
Use	every user-facing SLO; leading indicators (disk, certificates, lag near retention).
Numbers	99.9%/30 d ≈ 43 min; page at burn 14.4 over 1 h (2% of the budget); ≤ 2 incidents per shift.
Fails	alert fatigue; silent metric (absent-data alert, dead man's switch).
Products	CloudWatch alarms · Cloud Monitoring · Azure Monitor · Alertmanager, Grafana.

Health Check — healthcheck

Is	probes deciding if an instance gets traffic (readiness) or is restarted (liveness); startup probe.
Use	every replica behind an LB or Service; warm-up and shutdown; DNS failover.
Numbers	detection = interval × threshold (K8s $10 \times 3 \approx 30$ s); lost requests ≈ share × detection.
Fails	restart storm; zombie returns 200 while requests fail (outlier detection).
Products	ELB/Route 53 checks · GCLB checks · Azure probes · K8s probes, Actuator.

Network

who can talk to whom? · class 12

VPC — vpc

Is	your isolated private network in one region; nothing in or out without gateways and routes.
Use	always; one per environment; one per region for multi-region.
Numbers	/16 = 65 536 IPs; RFC 1918: 10/8, 172.16/12, 192.168/16; RTT in an AZ 0.1–0.5 ms, between AZs 1–2 ms. Sim cross-region +70 ms.
Fails	overlapping CIDRs; everything in one AZ; open security group on the DB.
Products	Amazon VPC · VPC network · VNet · Docker networks, kind.

Subnet — subnet

Is	an address range in one AZ; public or private by its route table.
Use	three tiers per AZ (public LB/NAT, private apps, isolated data), in ≥ 2 AZs.
Numbers	/24 = 256 addresses (AWS reserves 5 ⇒ 251); cross-AZ ≈ 0.01 USD/GB each way; SGs stateful, NACLs stateless.
Fails	AZ outage; wrong route table ("private" subnet with an IGW route).
Products	VPC subnets + SGs · firewall rules · NSGs · NetworkPolicy (Cilium).

NAT Gateway — nat

Is	outbound internet for private machines, unreachable from outside; lives in a public subnet.
Use	private services calling payment, e-mail, LLM providers; fixed egress IPs for allow-lists.
Numbers	≈ 0.045 USD/h + 0.045 USD/GB; up to 100 Gbps; sub-ms.
Fails	port exhaustion (keep-alive); zonal failure; per-GB cost surprise.
Products	NAT Gateway · Cloud NAT · Azure NAT · iptables masquerade.

VPN — vpn

Is	encrypted tunnel between networks (IPsec site-to-site, client VPN); interconnect for bandwidth.
Use	hybrid systems (stadium DC → cloud), admin access.
Numbers	≤ 1.25 Gbps per tunnel; adds 5–20 ms; 2 tunnels per connection; interconnect 1–100 Gbps.
Fails	tunnel down (two tunnels + BGP); MTU (MSS clamping); over-trust.
Products	Site-to-Site VPN · HA VPN · VPN Gateway · WireGuard.

Service Mesh — mesh

Is	a proxy per service (sidecar or ambient) + control plane: mTLS, retries, splits, telemetry.
Use	tens of services, zero trust in the cluster, canaries.
Numbers	≤ 1 ms per hop at p50; sidecar tens of MB per pod; certs live hours.
Fails	retry amplification; control plane down blocks new pods.
Products	VPC Lattice · Cloud Service Mesh · AKS Istio · Istio, Linkerd, Cilium.

AI & agents

an LLM in production, safely · class 13

LLM Gateway — llmgw

Is	one proxy to every model provider: routing, fallbacks, quotas, budgets, caches, cost metrics.
Use	several teams calling LLMs; outages and 429s; cost control.
Numbers	TTFT 0.2–2 s; 30–150 tokens/s; semantic hit tens of ms. Sim quota 50 req/s, 1 500 tokens, 0.003 USD/1k tokens.
Fails	provider outage or 429 (fallback, breaker); runaway cost (hard budgets).
Products	Bedrock · Vertex AI · Azure AI Foundry · LiteLLM, Envoy AI Gateway.

Orchestrator — orchestrator

Is	runs the agent loop: model call → tool calls → repeat until an answer or a limit.
Use	tasks needing data or actions the model lacks; paths that depend on results.
Numbers	latency ≈ steps × per step (6×1.7 s ≈ 10 s); cost ≈ steps × tokens × price; in flight = $20/\text{s} \times 10\text{ s} = 200$.
Fails	loops (step and token caps); partial execution (durable state, idempotency keys).
Products	Bedrock Agents · Agent Engine · Foundry Agents · LangGraph, Temporal.

Tool Registry — toolreg

Is	the tools an agent may call: name, description, JSON schema, endpoint, permissions (MCP).
Use	many tools, several agents, per-user authorization of calls.
Numbers	5–20 tools per request; 50–300 tokens per definition; call 10–500 ms, timeout 1–5 s.
Fails	over-privileged tools (authorize with the user's identity); a slow tool stalls the agent.
Products	AgentCore Gateway · Agent Builder · APIM as MCP · MCP servers.

Memory Fabric — memory

Is	context the model was not trained on: RAG (chunk, embed, retrieve, rerank), short- and long-term memory.
Use	answers from your own changing data (refund policy), citations, personalization.
Numbers	chunks 200–1 000 tokens; top-k 3–10; ANN 1–20 ms; rerank 50–300 ms.
Fails	retrieval miss; stale index; cross-tenant leak; memory poisoning.
Products	Bedrock KB · RAG Engine · Azure AI Search · pgvector, LlamaIndex.

Safety Mesh — safety

Is	guardrails on input (injection, PII), output (moderation, grounding, schema) and actions.
Use	any user-facing LLM feature; untrusted content; agents that act.
Numbers	rules < 1 ms; classifier 10–200 ms; LLM judge 0.5–2 s; eval set 50–200 examples. Sim catches 98%.
Fails	indirect prompt injection; PII leak; hallucination; guardrail outage (fail open or closed?).
Products	Bedrock Guardrails · Model Armor · Content Safety · NeMo Guardrails, Llama Guard.

Class by class

Each class's question, what to take away and its most common trap. The pages: insper.github.io/platform/2027.1/classes/.

1 · What is System Design

intro · clarify

2 M fans hit 60 000 seats at 10:00: which decisions above the code decide whether BoxOffice answers in time and sells each seat once?

- Requirements drive design: functional decide API and data; non-functional (number, percentile, window, load) decide the architecture.
- Architectures evolve: add a component for a measured symptom and expect the new problem it brings.
- Seven time-boxed steps: clarify, estimate, API and data, high level, deep dives, bottlenecks, wrap-up.

Numbers: hold p99 $<$ 300 ms, checkout p99 $<$ 1.5 s, 99.95%/28 days, catalog 60 s stale; interview minutes $5 \cdot 4 \cdot 5 \cdot 10 \cdot 12 \cdot 5 \cdot 3$.
Say: *"Before I draw, let me agree on the requirements and do a quick estimate."*
Trap: jumping to technology names before requirements and numbers.

2 · Estimation & Performance Math

estimation · estimate

Can one PostgreSQL take the on-sale? Requests per second, time per hold, safe utilization, the slow call behind a page.

- Estimate to pick a side of a threshold: 33 000 holds/s against $8/5$ ms $= 1\,600$/s $\Rightarrow$ waiting room at 1 000/s.
- $L = \lambda W$, $X_{\max} = c/S$, $R = S/(1-\rho)$: run at 50–70%.
- Users feel percentiles and chains: $1-(1-q)^N$, $\prod a_i$.

Numbers: 1 day $\approx 10^5$ s; 1 M/day $\approx$ 12/s; AZ round trip 0.5 ms; São Paulo–Virginia 120 ms; at 90% $R = 10\,S$.
Say: *"About 400 reads/s, 1 200 at peak: one server and a cache."*
Trap: sizing for the average (load, latency, availability of one box) instead of peak, tail and chain.

3 · Distributed Systems

distributed · design

A "Hold seat" times out after 5 s and the fan presses again: is she holding two seats? Will she pay twice?

- A timeout cannot tell slow from dead: every retried operation must be idempotent (conditional write or idempotency key).
- Consistency per data: linearizable (majority, $R + W > N$) for seats and money; read-your-writes for a fan's data; eventual for the catalog.
- Majorities overlap: odd N across zones; leases need fencing tokens; sagas and outbox across services, not 2PC.

Numbers: $R + W > N$ (3, 2, 2); $2f + 1$ nodes; a write pays 1–2 ms across AZs, 150–200 ms across continents.
Say: *"Seats are linearizable through one leader; the catalog is eventually consistent."*
Trap: believing in exactly-once delivery, or ordering events by the wall clocks of different machines.

4 · Traffic & Edge

edge · design

2 M fans press refresh at 10:00:00: which hops absorb, filter and route the traffic so only a fair, healthy stream reaches `inventory`?

- Each hop removes load from the next: origin $=$ traffic $\times$ miss rate (200 k req/s at 99% $\rightarrow$ 2 k).
- Token bucket: capacity $=$ burst, rate $=$ sustained; shared atomic counters across replicas; 429 $+$ Retry-After.
- Latency is round trips $\times$ distance: reuse connections, terminate near the user; consistent hashing moves $1/(N+1)$ of keys.

Numbers: DNS failover $\approx 3 \times 10 + 60$ s; new connection: TCP $+$ TLS 2–3 RTT, QUIC 1; fan limit 10 req/s, burst 20.
Say: *"Everything identical for every fan comes from the CDN; the origin sees only the misses."*
Trap: thinking in hit ratio, not miss rate (95% $\rightarrow$ 99% is 5$\times$ less origin load); believing DNS failover is fast.

5 · Compute & Services

compute · design, estimate

Millions of page views, seats held without double booking, cards charged, 60 000 tickets: and a few hundred req/s at 03:00. Where does each piece of work run?

- Stateless tier: $\lceil \text{peak}/(\text{cap} \times \rho) \rceil +$ spare $(4\,000/(500 \times 0.65) \rightarrow 13 + 1)$.
- Little everywhere: workers $= \lambda \times$ job time; Lambda concurrency $=$ rate $\times$ duration.
- Integrations: idempotency keys from the logical operation, timeout $=$ unknown, signed deduplicated webhooks, reconciliation.

Numbers: 200 threads $\div$ 0.1 s $=$ 2 000 req/s; cold start 0.1–1 s; 500 DB connections $\div$ 20 $=$ 25 replicas; hold 10 min.
Say: *"The purchase answers 202; the e-mail and the PDF follow from a queue."*
Trap: assuming elasticity and duplication are free: autoscaling takes minutes; cron and retries run N times.

6 · Databases & Data Systems

databases · design

One PostgreSQL holds everything; at 10:02 CPU is 100%, a report locks for 40 s and seat A-12 is sold twice. What goes where?

- Choose by access pattern, volume and consistency: relational for invariants; 60 000 seats is tiny, contention is the hard part.
- Put the check inside the write (conditional `UPDATE`, rowcount 1, unique constraint); a hot row caps at $1/t$.
- Replicas scale reads and lag; partition by a high-cardinality key with consistent hashing; expect the hot partition.

Numbers: hot row 5 ms $\Rightarrow$ 200/s; consistent hashing moves $\approx 1/N$ of keys, `mod N` $\approx$ 90%; B-tree fan-out $\approx$ 500.
Say: *"Seats live in PostgreSQL behind a conditional update; I shard only when writes force me to."*
Trap: check-then-act under read committed; reading your own write from an async replica.

7 · Caching

caching · design, deep dive

Tens of thousands of `GET /events/42` per second run the same three queries on data unchanged for a week: how do we serve a copy without acting on a stale one?

- Think in miss ratio: DB QPS $= (1-h)\lambda$; each nine is 10$\times$ less load.
- Cache-aside: write DB, commit, then delete the key; TTL with jitter; cache what you display, not what you decide on.
- Once the DB is sized for misses, the cache is load-bearing: stampede $=$ reads/s $\times$ recompute time; L1 for hot keys.

Numbers: 50 000 reads/s: h 90% $\rightarrow$ 5 000 DB QPS, 99% $\rightarrow$ 500; Redis 0.2–1 ms, $\approx 10^5$ ops/s; stampede $20\,000 \times 0.2 = 4\,000$.
Say: *"Cache-aside, TTL 5 min with jitter, delete after commit, 2-second L1."*
Trap: treating 95% $\rightarrow$ 99% as "4 points" (it is 5$\times$ fewer misses); deleting the key before the commit.

8 · Messaging & Event-Driven

messaging · design, deep dive

The e-mail provider goes from 80 to 800 ms and purchases fail: which steps must the fan wait for, and how do we hand off the rest safely?

- Go async for work nobody waits for: the chain availability becomes order $\times$ broker (99.65% $\rightarrow$ 99.90%).
- Queue math: backlog $= (\lambda - \mu)t$, wait $=$ backlog$/\mu$, partitions $\geq T/c$, consumers $\leq$ partitions.
- At-least-once: idempotent consumers, bounded retries $+$ DLQ, outbox instead of dual writes, sagas with a payment pivot.

Numbers: 6 000 vs 2 000/s for 3 min $\rightarrow$ 720 000 waiting, 360 s; RF 3, `acks=all`, min ISR 2; retention 7 days.
Say: *"Order events keyed by order id, idempotent consumers, and I alert on consumer lag."*
Trap: dividing the backlog by the arrival rate; trusting exactly-once; dual writes.

9 · Observability

observability · deep dive

Every pod is green, CPU 45%, errors 0.3%, yet the median fan waits 9 s to pay. What would have found the slow hop in minutes?

- Measure what users feel per journey: percentiles and SLIs; multiwindow burn-rate alerts (14.4$\times$ 1 h/5 min).
- Shared context (`trace_id`, exemplars, `traceparent`) turns signals into one investigation: 7.9 s in `pool.acquire`.
- Telemetry costs: bound cardinality, tail-sample traces (7.8% kept, 100% of errors). Mitigate first; blameless postmortems.

Numbers: series $=$ metrics $\times$ label values $\times$ instances (204 000; $\times 10^6$ with `customer_id`); traces $500 \times 8 \times 500$ B $=$ 2 MB/s.
Say: *"Metrics to detect, traces to locate, logs to explain."*
Trap: green dashboards of causes and averages; averaging per-pod p99s (450 vs 756 ms).

10 · Reliability & Resilience

reliability · deep dive

The payment provider goes from 200 ms to 4 s and nothing is broken: how do we stop it taking down `order`, the gateway and browsing, and how much failure is acceptable?

- Availability is arithmetic: lower MTTR beats higher MTTF; series multiplies; the unreplicated part dominates.
- SLO → error budget → burn rate turns "reliable enough?" into decisions: ship or freeze, page or ticket.
- Slow is worse than dead: timeout, one jittered retry, breaker on slow calls, bulkhead by Little, honest fallback.

Numbers: 720 h / 2 h → 99.72%; burn 14.4 over 1 h ≈ 2% of the budget; breaker 50% of 20 calls, open 30 s; bulkhead 50.
Say: *"Payment gets a 2 s timeout, one jittered retry with an idempotency key, a breaker and a bulkhead."*
Trap: retries at every layer ($27\times$) and replicas as backups (they copy the `DELETE`).

11 · Cloud-Native Platforms

cloud-native · design, estimate

From one `docker compose up` to 20× traffic for 30 minutes, a fix at 09:55 without downtime, and the cost known before the invoice.

- Stateless, disposable apps in small immutable images; graceful shutdown.
- Kubernetes reconciles desired state: right requests, local probes, HPA $\lceil 4 \times 90/60 \rceil = 6$; pre-scale before 10:00.
- GitOps and canaries gated on error rate and p99; cost = capacity × time (CDN, data, egress dominate).

Numbers: HPA $\lceil 4 \times 90/60 \rceil = 6$; canary 5 → 25 → 50%, gate errors < 0.5%, p99 < 300 ms; ≈ 0.013 USD per ticket.
Say: *"We pre-scale at 09:45: the autoscaler reacts in minutes, the on-sale in seconds."*
Trap: the database in readiness or liveness probes; "the HPA will save the on-sale".

12 · Network & Security

network-security · design

Who can talk to whom, and how does each call prove who it is and what it may do, from the fan's phone to `payment`?

- Security is a design decision: STRIDE at every trust boundary, independent layers.
- Isolation from routes, security groups and NetworkPolicy; a NAT per AZ; endpoints for cloud services; plan CIDRs.
- Identity everywhere, short-lived: Authorization Code + PKCE, 15-min JWTs, mTLS + authorization policy, rotated secrets, verified signatures.

Numbers: /16 = 65 536, /24 = 256 (5 reserved); access token 15 min; NAT ≈ 0.045 USD/h + 0.045 USD/GB.
Say: *"Only the load balancer is public; the database subnet has no route to the internet."*
Trap: authentication taken for authorization (BOLA); STRICT mTLS without an authorization policy.

13 · AI Systems & Agents

ai-systems · design, estimate

"Find two seats under R$ 300 and hold them": answered in seconds, within rate limits and budget, without invented holds or obeying text hidden in a venue page?

- The LLM is a capacity-bound dependency: cost and latency ≈ calls × tokens; context grows each step (4 → 10 calls: cost $3.6\times$).
- The model proposes tools; the orchestrator decides (as the user, budgets, idempotency, risk tiers). Knowledge via RAG, live data via tools.
- Safety is architectural; quality needs evals (golden set, pass^k) and GenAI traces.

Numbers: 4 calls ≈ 17 000 input tokens; $0.95^{10} \approx 60\%$; latency = TTFT + TPOT$(N-1)$; KV cache 128 KiB per token.
Say: *"Five components, not one LLM box: gateway, orchestrator, tools, memory, safety."*
Trap: prompt caching (safe) confused with semantic caching (false hits); reporting pass@k.

14 · AI-Assisted Delivery

ai-delivery · deep dive

12 engineers, 60 PRs a week: where should AI steps sit in the pipeline to cut review latency without noise, deleted tests or surprise invoices?

- Each AI step has a contract: trigger, untrusted context, validated output, least-privilege action, budget, fail direction.
- Tokens are cents; attention is the cost: precision beats recall (41% at 3 comments per PR is ignored).
- Judge deterministically: mutation score, external verification, statistical canary verdicts; agent cost is quadratic in turns.

Numbers: review ≈ 0.02–0.09 USD; 40% precision × 4 comments × 5 000 reviews/week ≈ 12 000 useless; 10 → 40 turns: 1.33 → 11.32 USD.
Say: *"AI proposes; rules and humans dispose."*
Trap: letting the model decide (approve, merge, quarantine, roll back) or trusting instructions in the prompt.

15 · Performance Engineering

performance · estimate, deep dive

Last year's on-sale fell at 10:02 after an "all green" test of 200 virtual users. How do you test so it does not happen again?

- $R = S+$ waiting; $N = XR$ and $U = XS$ tie the numbers; latency explodes near saturation.
- Plan with the knee, not the maximum: replicas = $\lceil$peak$/($knee $\times (1 -$ headroom$))\rceil$ + zone loss.
- Open model, realistic data, per-endpoint thresholds; RED → USE → profile; one change at a time.

Numbers: closed model backs off 182 → 67 req/s; peak 4 500 holds/s, 275 per pod at the knee → 22 pods, 33 with a zone loss.
Say: *"We test with an open model and require zero dropped iterations."*
Trap: a closed-model test against open-world traffic (coordinated omission: p99 1 ms vs 1 366 ms).

16 · Case Studies

case-studies · all stages

URL shortener, rate limiter, feed, chat, rides, video, notifications, payments, autocomplete, ticketing, AI assistant, cache: how do you solve any of them in 45 minutes?

- Follow the framework in order and end every estimate in a decision ("hot set 33 GB fits in memory ⇒ cache").
- Twelve cases, eight patterns; Little's Law and the hit-ratio formula appear in almost every estimate.
- Correctness lives in idempotency and atomic guards; strong consistency only for seats, money and assignment.

Numbers: 7 base62 characters = 3.5 trillion codes; in flight = rate × latency; DB load = $(1-h)\lambda$.
Say: *"Let me clarify, estimate and then draw; every number ends in a decision."*
Trap: rate limiting or autoscaling a resource that cannot scale (hot row, primary, PSP limit): it needs admission control.

The 28 playground scenarios

`monolith`	the first box to saturate sets the breaking point
`scale-out`	N+1 replicas; health checks detect the dead one
`read-heavy-cache`	DB load = $(1-h)\times$ reads; flush → stampede
`cdn-static`	static bytes stay at the edge
`db-replicas`	replicas add reads, not writes, and lag
`sharding`	partitions scale until one key is hot
`async-queue`	a queue turns a spike into backlog
`event-fanout`	each consumer group reads at its own pace
`cascading-failure`	slow dependency → threads exhausted (Little)
`retry-storm`	retries without backoff never let it recover
`rate-limiter`	reject the excess fast (429)
`multi-region`	failover = detection + DNS TTL
`serverless-burst`	cold starts and concurrency limits
`flash-sale`	CDN, cache and a waiting room for the peak
`url-shortener`	key lookup, read-heavy
`news-feed`	fan-out on write vs on read
`chat`	store partitioned, deliver by pub/sub
`ride-sharing`	a write-heavy geo index
`video-streaming`	CDN + object store + transcoding workers
`notification-system`	queue in front of provider quotas
`payment-system`	retries need idempotency keys
`search-autocomplete`	a latency budget met by a cache
`ai-assistant`	LLM calls set latency and cost
`observability-stack`	you can't fix what you don't measure
`microservices`	the HPA adds pods; only nodes add room
`k8s-node-failure`	N+1 nodes; discovery drops dead pods
`db-failover`	RTO = failover; RPO = writes in the lag
`blank`	an empty canvas for your own design

insper.github.io/platform/playground/scenarios/#<id>

Design adapted from the instructor's Machine Learning cheat sheet. Colours: grey clarify, blue estimate, purple design, orange deep dive, green answer, red trap; chips: simulator categories.