BACKEND INTERVIEW CHEAT SHEET

Surviving overload: retry storms and cascading failures

Renekin AI

Inspired by this week's Outlook and Teams outage (Nov 12, 2026). Microsoft hasn't published a root cause, so this sheet covers the general patterns interviewers ask about, not what happened inside Microsoft.

Why systems collapse

> A **cascading failure** is "a failure that grows over time as a result of positive feedback" (Google SRE book). Typical loop: some servers fail or slow down, the rest take their load, requests time out, clients retry, load goes up again, more servers fail.



If each of 3 layers tries up to 4 times, one click can become 4 x 4 x 4 = 64 database attempts (Google SRE example).

The 8 tools, in the order to mention them

1 Timeouts

- Every network call gets a timeout. Without one, threads and connections pile up behind a slow dependency.
- Set it from measured latency (a bit above your slow-but-normal requests, such as p99), not a guess.

2 Retries with backoff and jitter

- At most **3 attempts** per request, then return the error.
- Wait longer each time (exponential backoff) with a cap, and add **random jitter** so clients don't retry in sync.
- Only retry operations that are safe to repeat (use idempotency keys for writes).

```
for attempt in range(3):
    try: return call(timeout=t)
    except Retryable:
        wait = min(cap, base * 2**attempt)
        sleep(random(0, wait))
raise  # "full jitter"
```

3 Retry budget

- Retry at **one layer only**: the one right above the failing service.
- Cap retries per client: Google retries only while retries are **under 10%** of requests.
- When overloaded, answer "overloaded, don't retry" so callers above stop too.

4 Circuit breaker

State	Behaviour
Closed	Calls go through; count failures.
Open	Too many failures: fail fast, don't call the dependency at all.
Half-open	After a cool-down, let a few test calls through. Success closes it, failure opens it again.

5 Load shedding

- Reject extra work early and cheaply (for example, return 429 or 503) instead of letting every request get slow.
- Tag requests by priority and drop the least important first. Google uses CRITICAL_PLUS, CRITICAL, SHEDDABLE_PLUS and SHEDDABLE.

6 Graceful degradation

- Do less work per request under stress: cached or older data, fewer results, turn off non-essential features.
- A slightly worse answer beats a timeout.

7 Deadlines, passed down

- Pass the request's deadline to every downstream call, so all calls in the tree share the same absolute deadline.
- Drop work whose caller has already given up: it can no longer help anyone.

8 Small, bounded queues

- Long queues just add latency. Google suggests queues of about 50% or less of the thread pool size.
- Under overload, serve newest first (LIFO) or use CoDel, so fresh requests still succeed.

Bonus: client-side throttling

> Clients can also protect a struggling backend. Each client tracks how many requests it sent and how many the backend accepted. Once requests reach **K times** the accepted ones, the client starts rejecting new requests locally, with a probability that grows as acceptance drops. Google generally uses **K = 2**.

Answer it in an interview (60 seconds)

Step	What to say
1. Name it	"This is a cascading failure: overload causes timeouts, timeouts cause retries, and retries add more load."
2. Stop the amplification	"Timeouts on every call, at most 3 attempts with exponential backoff and jitter, retries at one layer, and a retry budget of about 10%."
3. Fail fast	"A circuit breaker on each dependency, so we stop hammering something that's already down."
4. Protect the core	"Shed low-priority traffic first, degrade gracefully, keep queues short and pass deadlines down."
5. Recover safely	"Bring capacity back gradually. Servers that come back can be overwhelmed again by the queued-up traffic."

One line: "Timeouts, capped retries with backoff and jitter, a retry budget, circuit breakers and load shedding, so one failure can't snowball into a full outage."

Follow-up questions to expect

- **"Why is jitter needed?"** Without it, every client that failed at the same moment retries at the same moment, so the spikes repeat.
- **"Which errors should you retry?"** Timeouts and temporary errors (such as 503). Not bad requests (400), auth errors (401/403) or "overloaded, don't retry".
- **"What about writes, like payments?"** Only retry them with an idempotency key, so a retry can't charge twice.
- **"How do you know it's happening?"** Watch retry rate, error rate, latency percentiles and queue length per dependency, and alert when retries climb.
- **"Circuit breaker vs retry?"** Retries handle short blips. A circuit breaker stops calls when a dependency is clearly down.

Sources

- Google SRE book: "Handling Overload" (retry limits, 10% budget, criticality, client-side throttling) and "Addressing Cascading Failures" (64-attempt example, queues, deadlines): sre.google/sre-book
- Outage context: The Conversation and The Next Web, Nov 13, 2026. Microsoft had not confirmed a root cause at the time of writing.