The Isolation Paradox

How sharding a multi-tenant platform
revealed hidden failures — and created new ones.

Where we’re going

01 **Why we isolated**

A shared indexing stack and its failure domain

02 **Isolation as a lens**

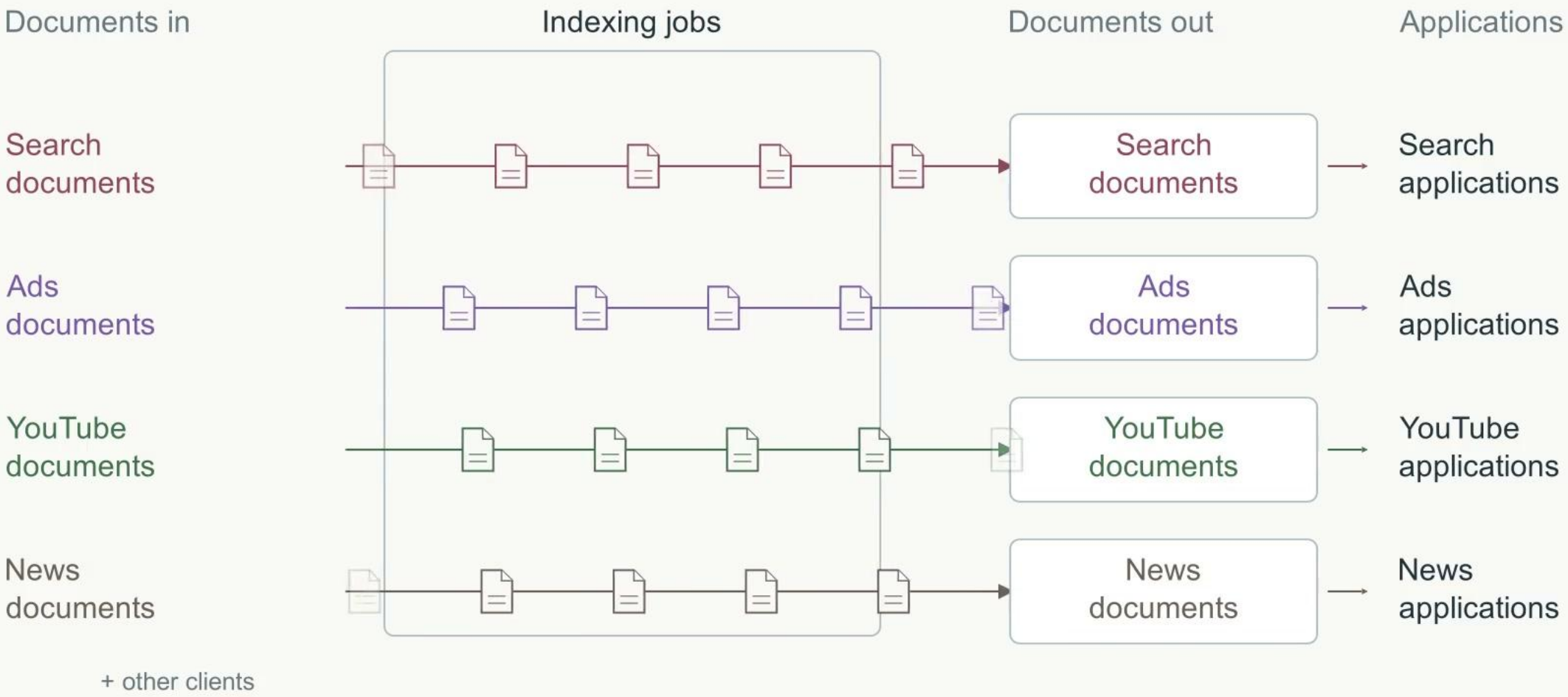
Hidden errors and overlooked client latency

03 **Isolation as a source of risk**

Independent recovery and dedicated capacity

Why we isolated, what isolation exposed, and what it changed.

One indexing stack, many clients



Many clients share the same indexing jobs.

Inside indexing jobs

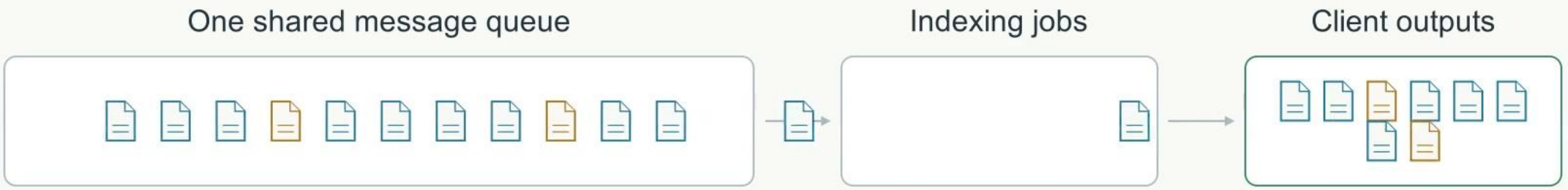


+ other clients

Shared services create a shared failure domain

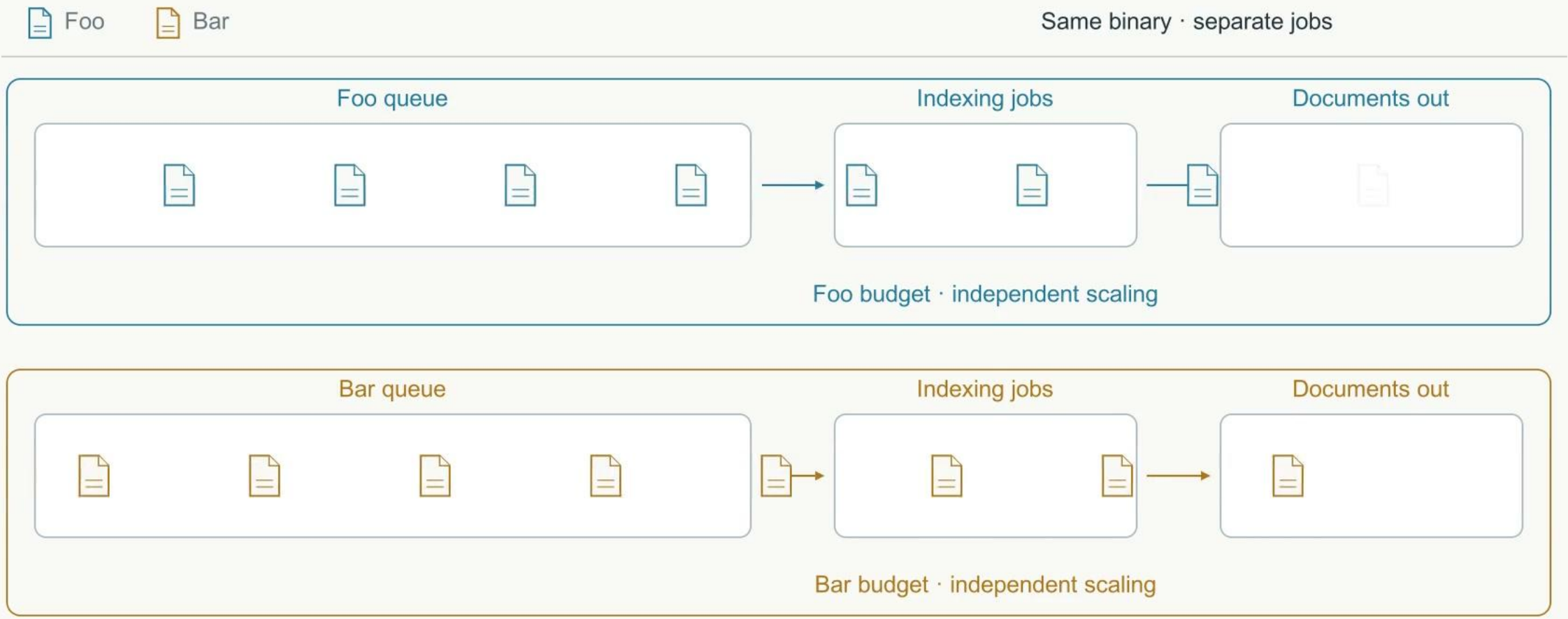
Foo Bar

Example 2



Bar makes progress, slowly. Most completions serve Foo.

Dedicated allocs isolate client traffic



Each alloc owns its backlog, jobs, resource budget, and scaling.

Dedicated service instances

Search	Ads	YouTube	News
Search.microservice1	Ads.microservice1	YouTube.microservice1	News.microservice1
Search.microservice2	Ads.microservice2	YouTube.microservice2	News.microservice2
Search.microservice3	Ads.microservice3	YouTube.microservice3	News.microservice3
Search.microservice4	Ads.microservice4	YouTube.microservice4	News.microservice4
Search.message_queue1	Ads.message_queue1	YouTube.message_queue1	News.message_queue1
Search.message_queue2	Ads.message_queue2	YouTube.message_queue2	News.message_queue2

+ other clients

After isolation, each client needs its own instances of these jobs.

Two kinds of post-migration surprises

Problems isolation revealed

Same behavior. Different visibility.

Problems isolation introduced

Different recovery and access to capacity.

Other problems arose because isolation changed the system.

Isolation as a lens

Problems isolation revealed

Same behavior. Different visibility.

Problems isolation introduced

Different recovery and access to capacity.

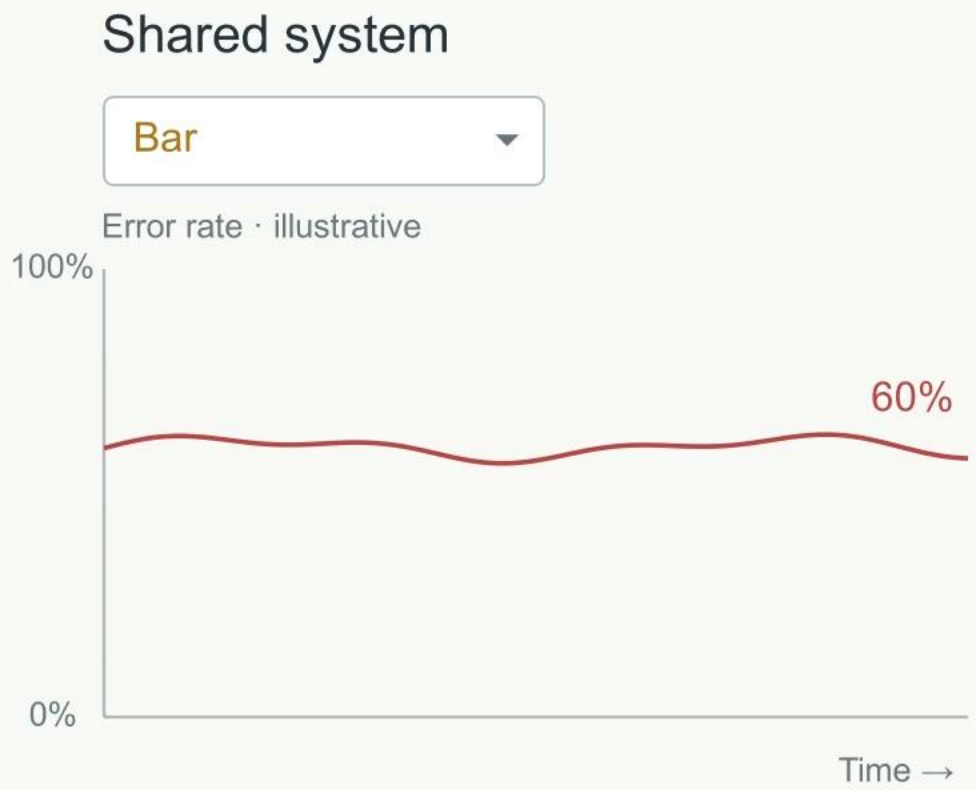
First: problems that already existed, but isolation made visible.

Migration makes the error rate visible



Inspect the errors reported by the dedicated instance.

Did the migration cause the errors?





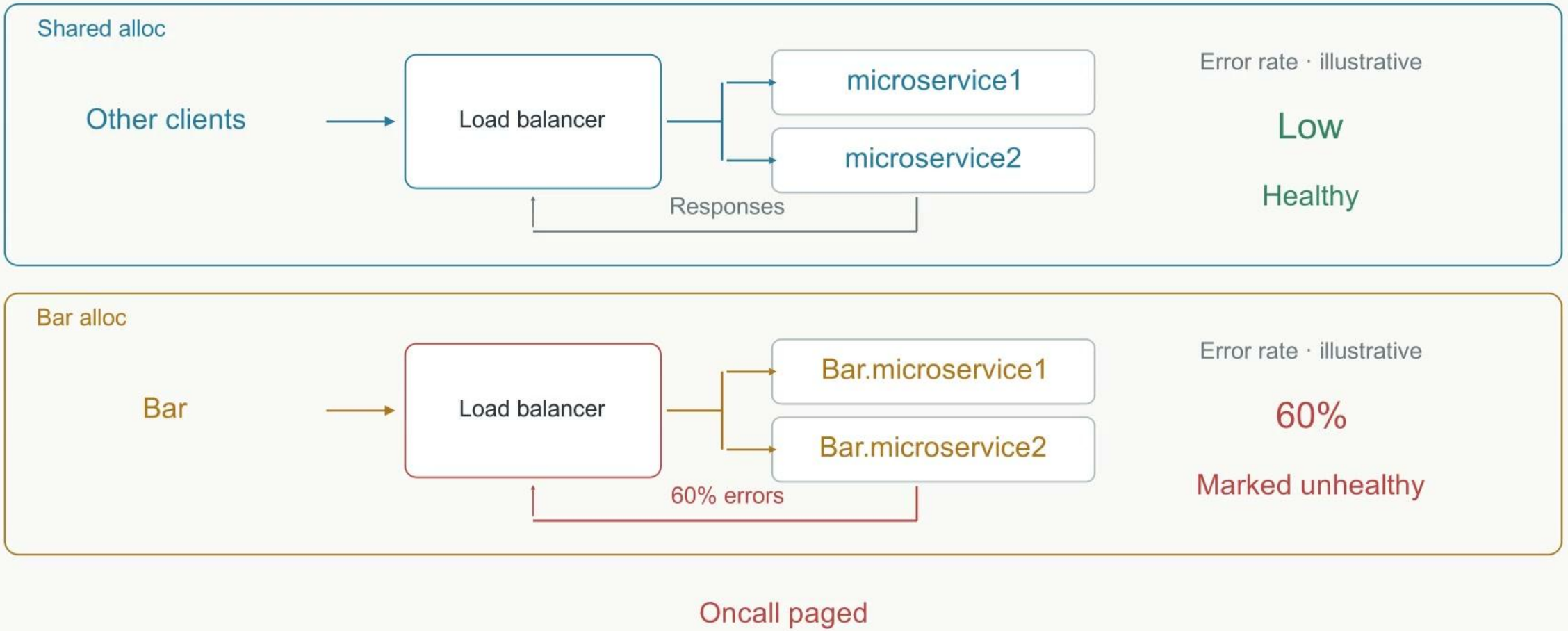
Isolation exposed the errors; migration prompted the check.

The client uses throttling as feedback



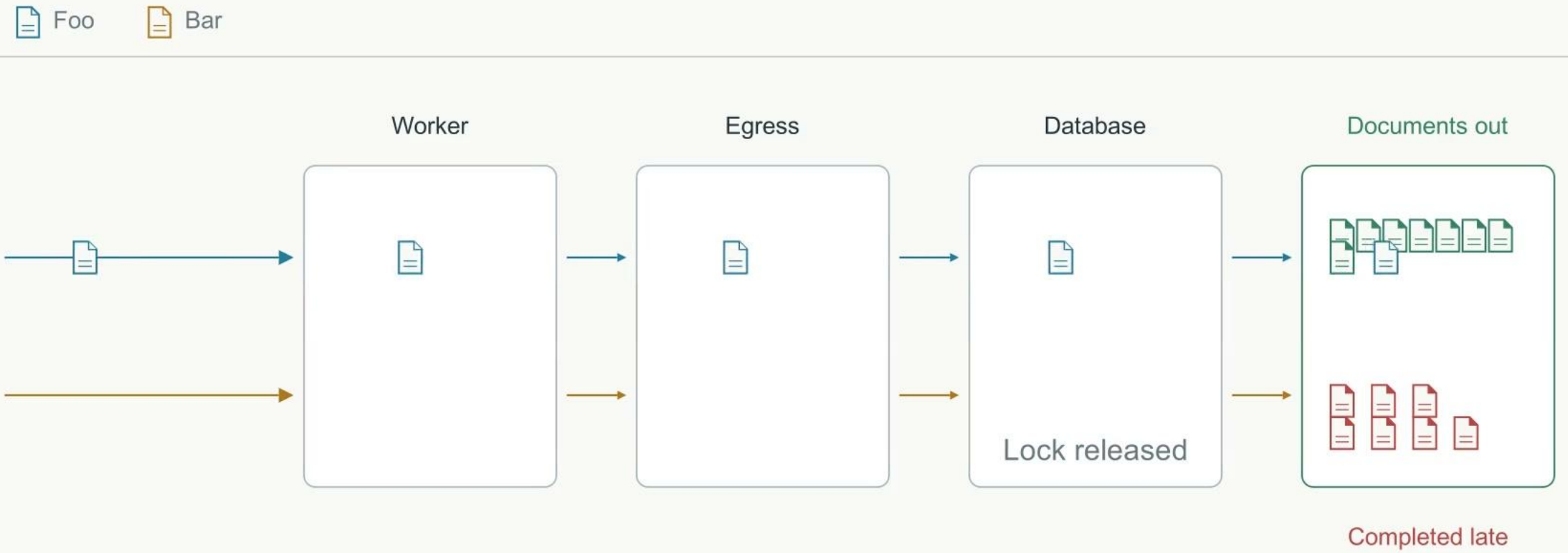
The client team confirmed: this behavior was intentional.

The same errors now trigger paging



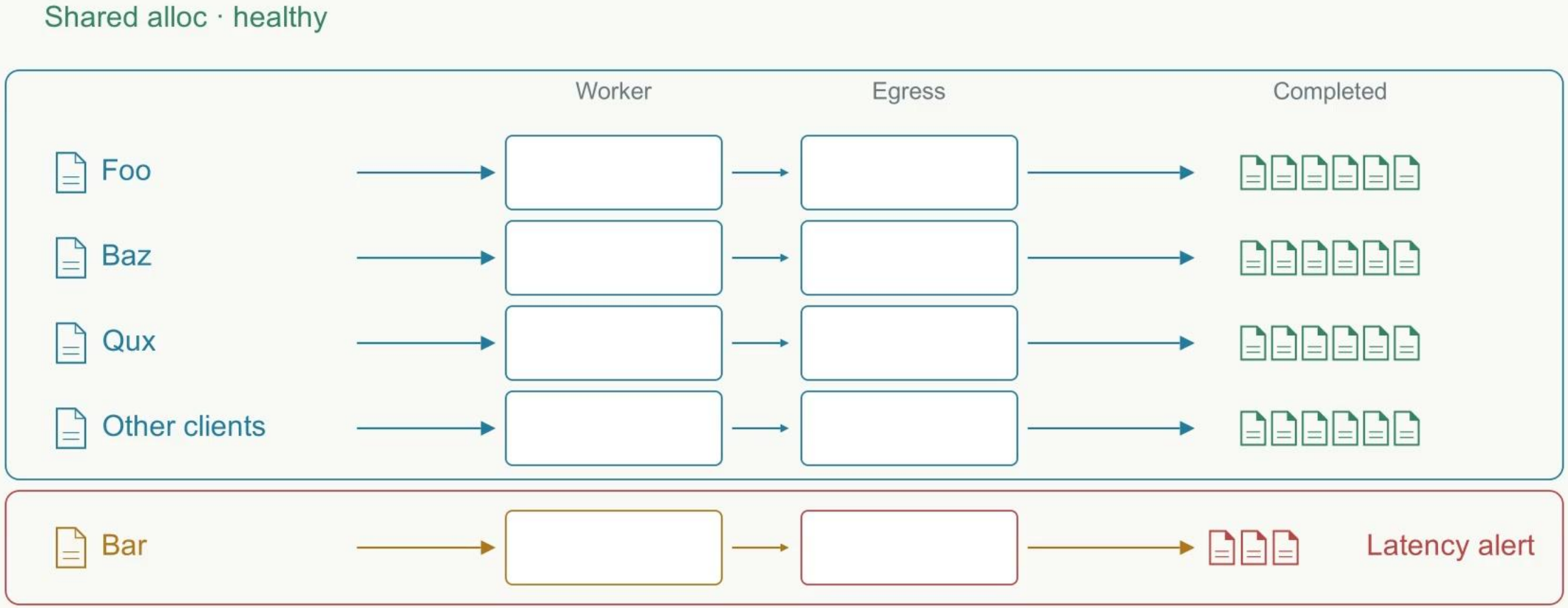
The isolated error rate triggers unhealthy status and paging.

After isolation, Bar triggers a latency alert



Bar completes late; Foo stays on time.

Did migration cause the slowness?



Isolation exposed the delay; migration prompted the investigation.

Isolation as a source of risk

Recovery coordination

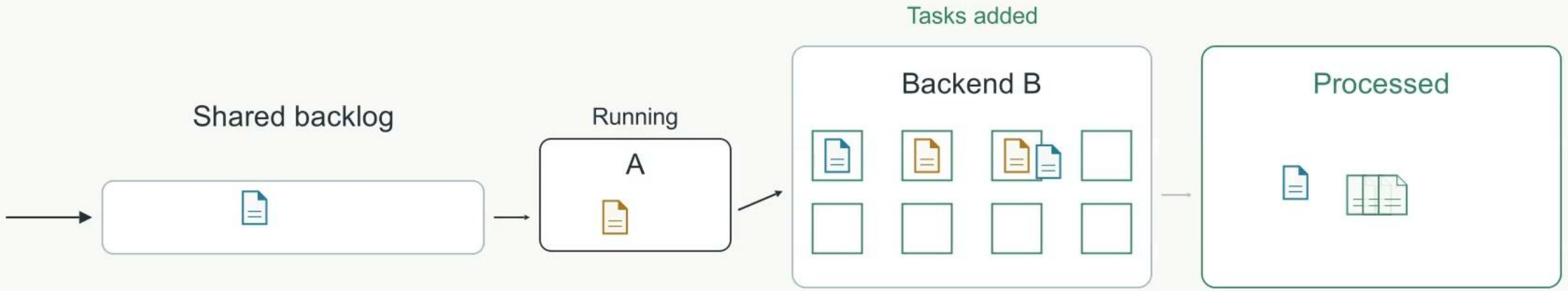
Separate controllers can send too much to a shared dependency.

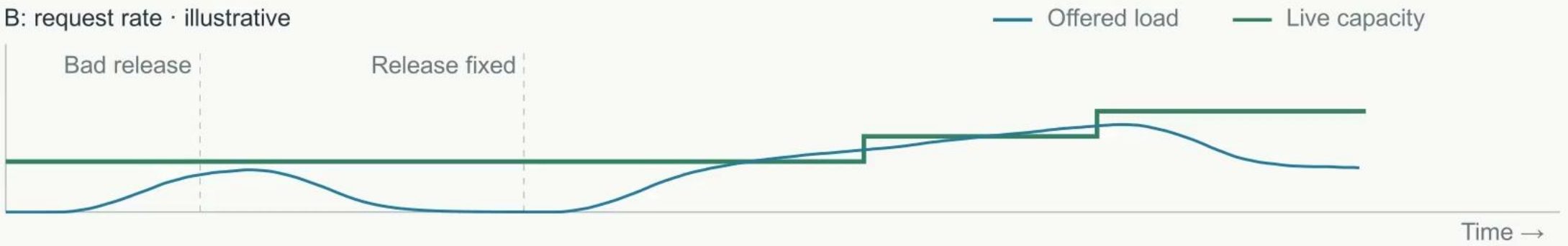
Pooled spare capacity

Idle resources can exist outside the alloc that needs them.

Dedicated resource budgets also remove access to other clients' idle capacity.

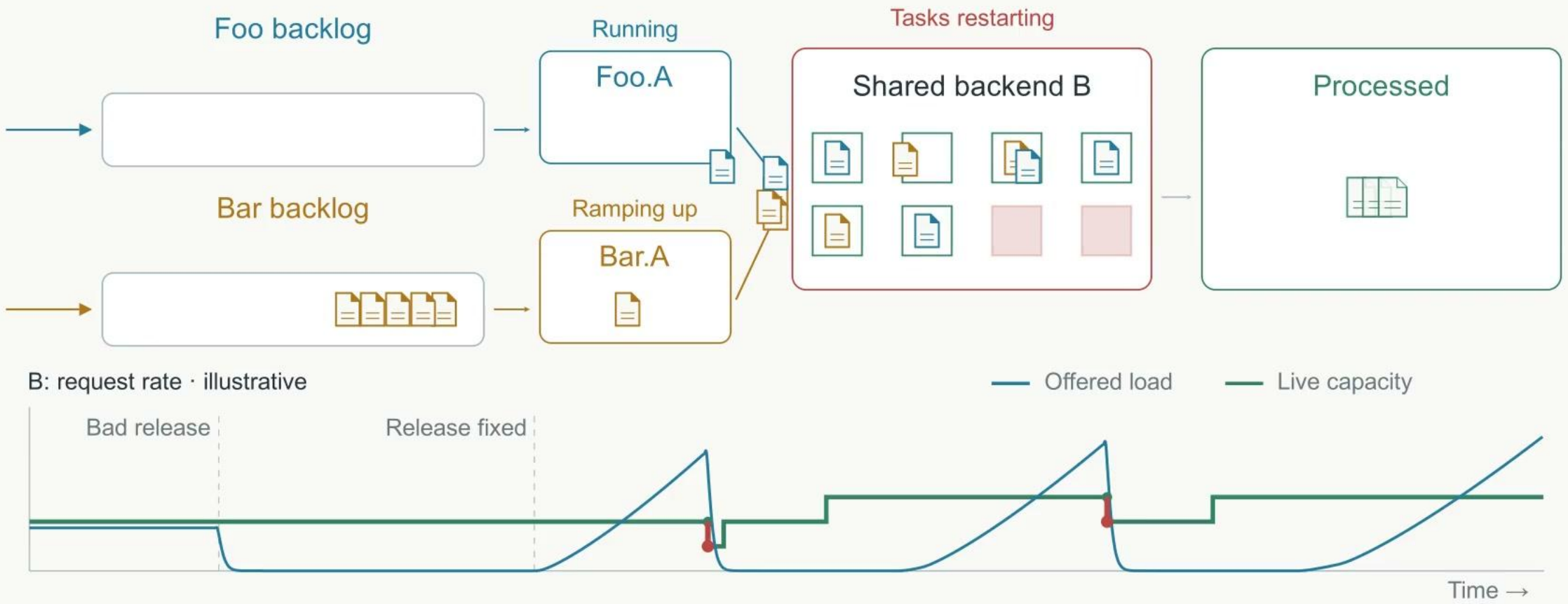
Recovery after a bad release of A





Completions overtake arrivals. The backlog drains.

After isolation: independent recovery



Separate retries surge again: more overload, more crashes, more backoff.

Dedicated budgets give clients control

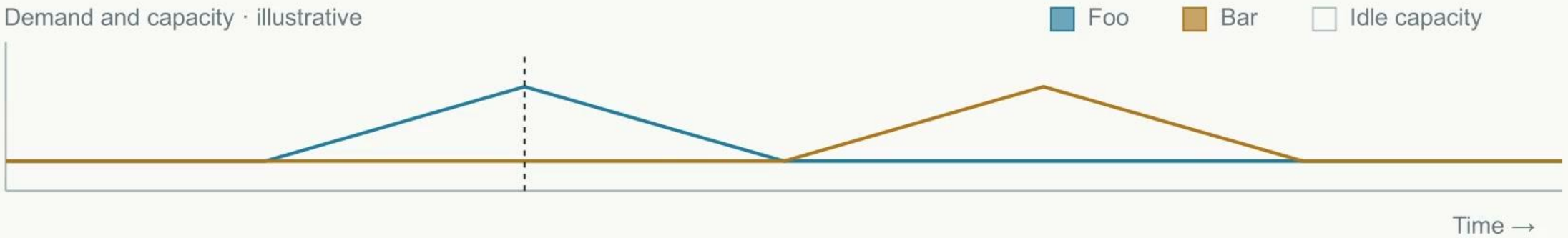
Capacity · illustrative



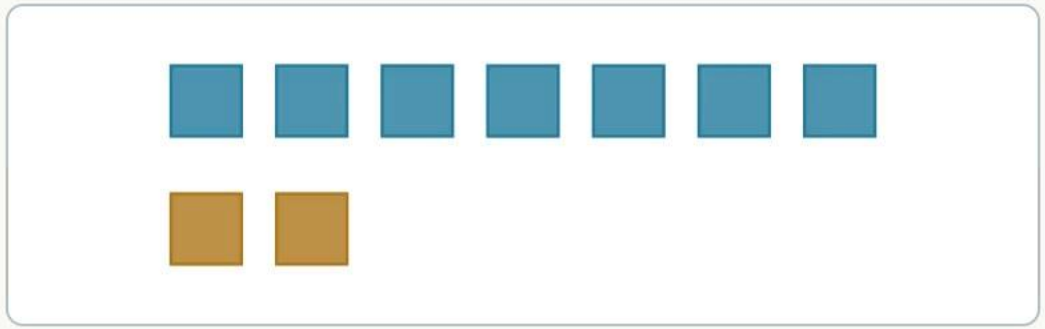
The old safety margin was shared idle capacity.

Both expected peaks are covered. But what happens above that peak?

Provisioning for each peak costs more

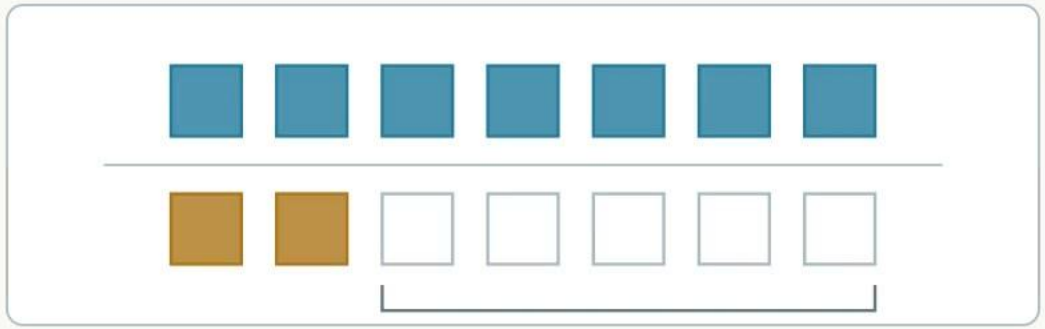


Shared pool



Same demand at the Foo peak

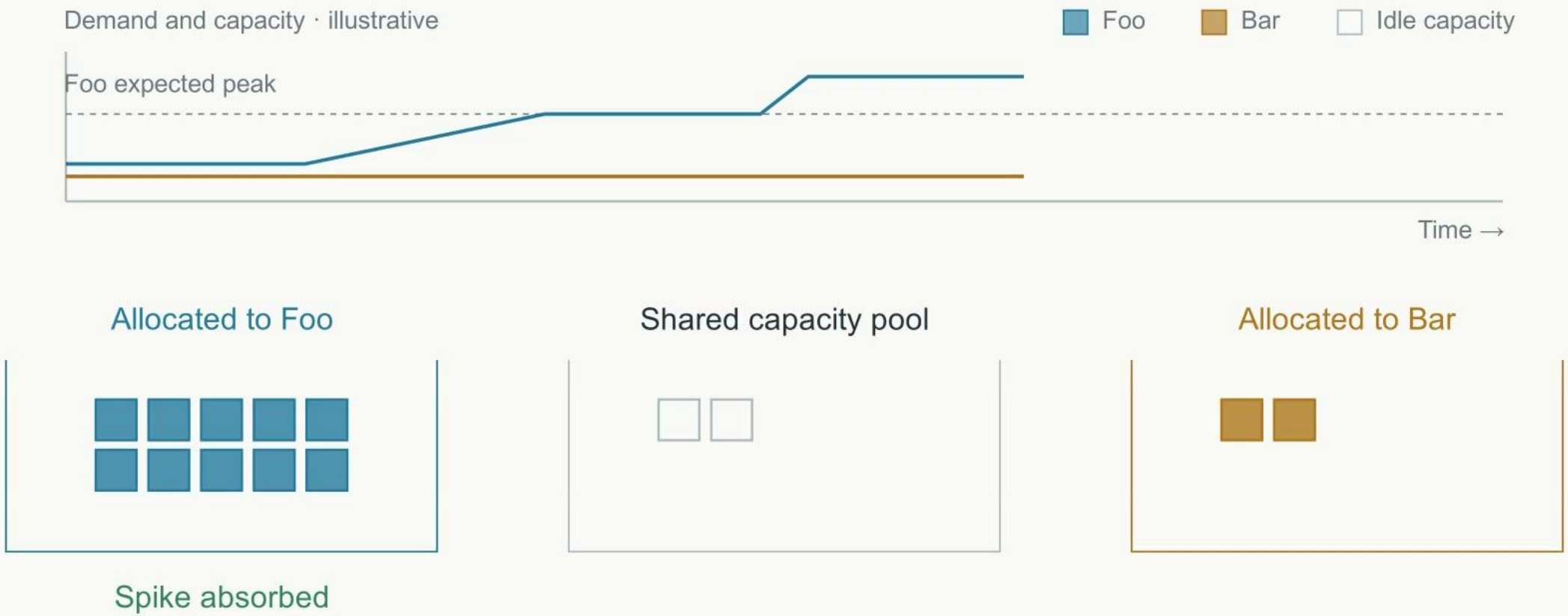
Dedicated allocs



Reserved for the Bar peak

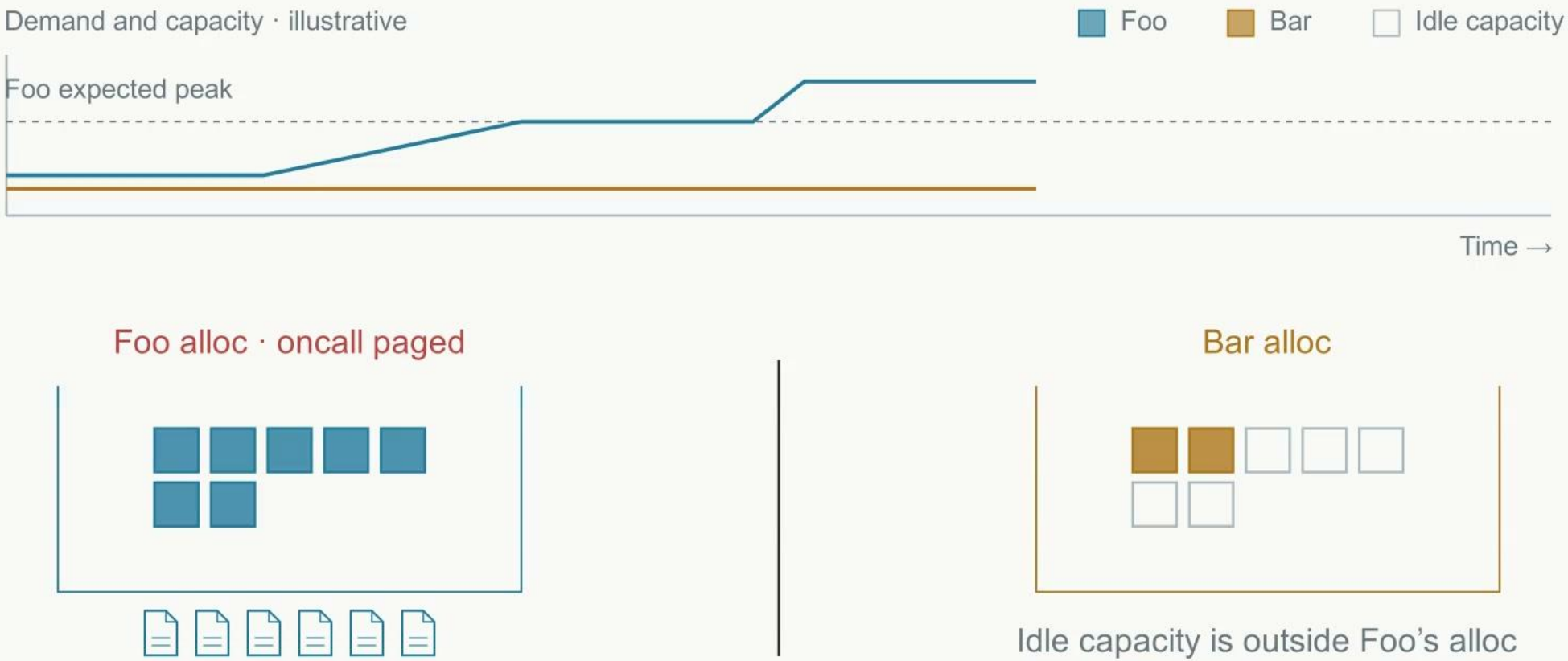
Each alloc must cover its own peak.

The shared pool absorbs the unexpected spike



The shared pool absorbs the spike.

The same spike is now an incident



The same spike now needs incident mitigation.

What to take back to your system

Whose experience do your SLOs describe?

What does the shared system give you implicitly?

What happens while the system is recovering?

Reason about outages and recovery, as well as steady-state behavior.