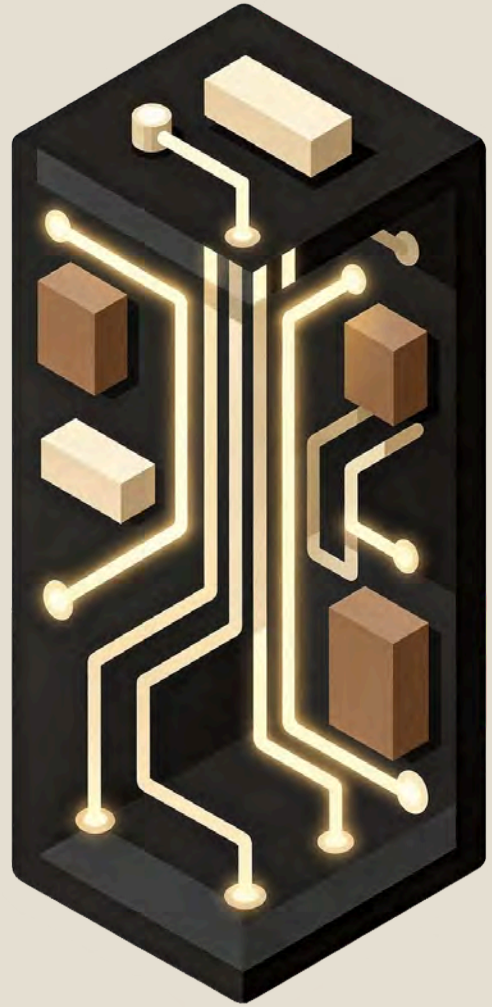
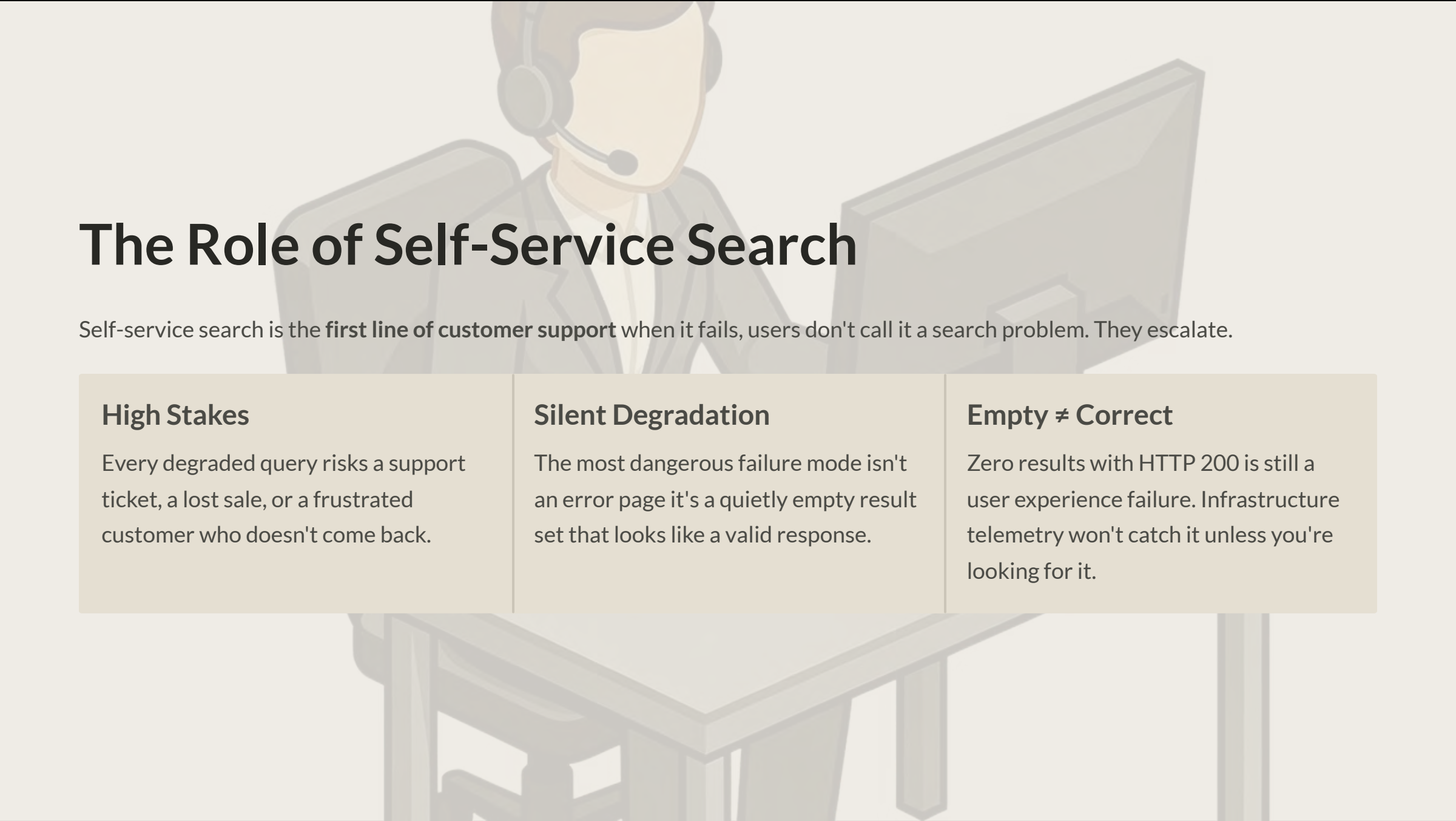
Designing Resilient Self-Service Search in Go From Failure Modes to Intelligent Fallback Architectures

Conf 42 Golang 2026

Hima Bindu Yanala · eBay Inc.





The Role of Self-Service Search

Self-service search is the **first line of customer support** when it fails, users don't call it a search problem. They escalate.

High Stakes

Every degraded query risks a support ticket, a lost sale, or a frustrated customer who doesn't come back.

Silent Degradation

The most dangerous failure mode isn't an error page it's a quietly empty result set that looks like a valid response.

Empty ≠ Correct

Zero results with HTTP 200 is still a user experience failure. Infrastructure telemetry won't catch it unless you're looking for it.

What Can Go Wrong

Traffic Spikes

Sudden surge overwhelms shards

Partial Shard Failures

Some nodes return degraded or no results

Cascading Fan-Out Effects

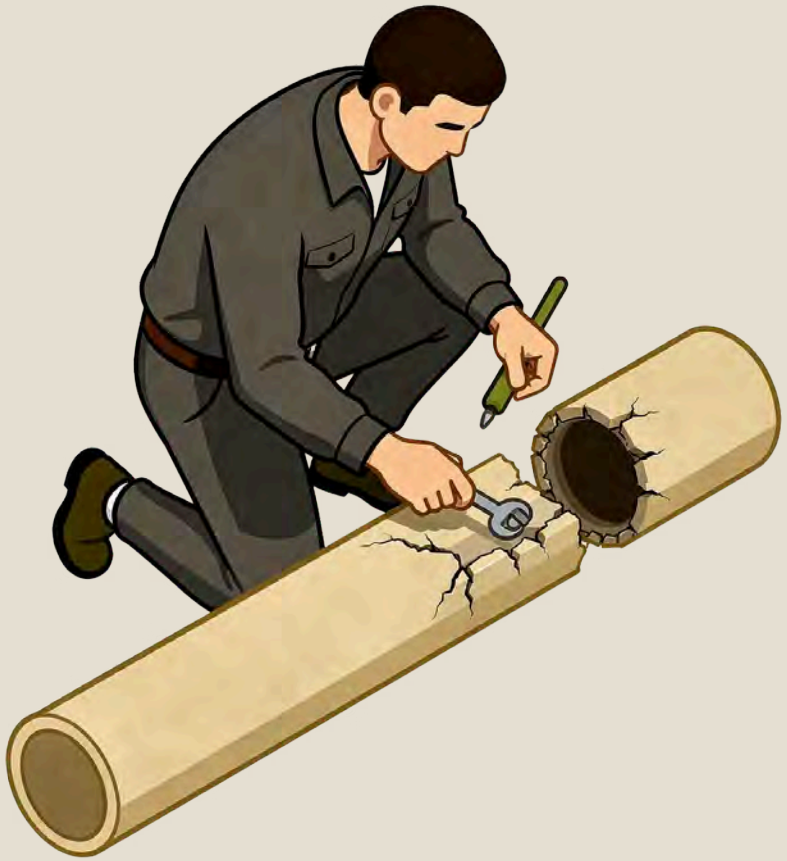
One slow node delays all

Silent Degradation

Empty results returned without an error signal

These failure modes share a dangerous property: **they rarely produce explicit errors**.

The system appears healthy while users experience silent degradation.



The Problem with Reactive Fallback

In most systems, fallback is an afterthought bolted on after an incident rather than engineered from the start.

1

Reactive Design

Fallback added post-incident, covering only known failure patterns

2

Gap at Scale

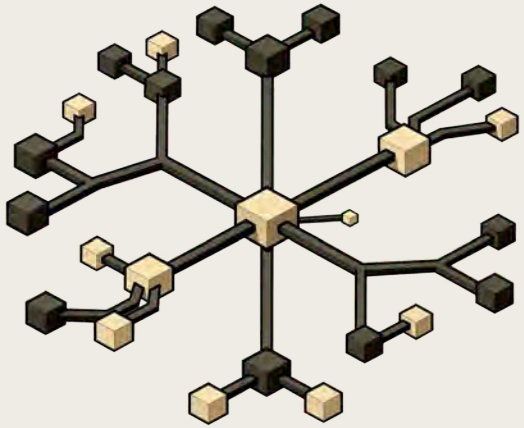
High-traffic distributed systems surface failure modes faster than reactive patches can address

3

First-Class Reliability

Fallback must be a **core architectural concern**, designed in not added on

Fan-Out Amplification: A Hidden Threat



A distributed search query fans out across dozens of shards simultaneously. A single slow node doesn't just add latency it **holds the entire response hostage**.

- Rare outliers (p99 nodes) become systemic bottlenecks
- Compounding timeouts cascade across dependent shards
- Go's goroutine model amplifies both throughput *and* blast radius

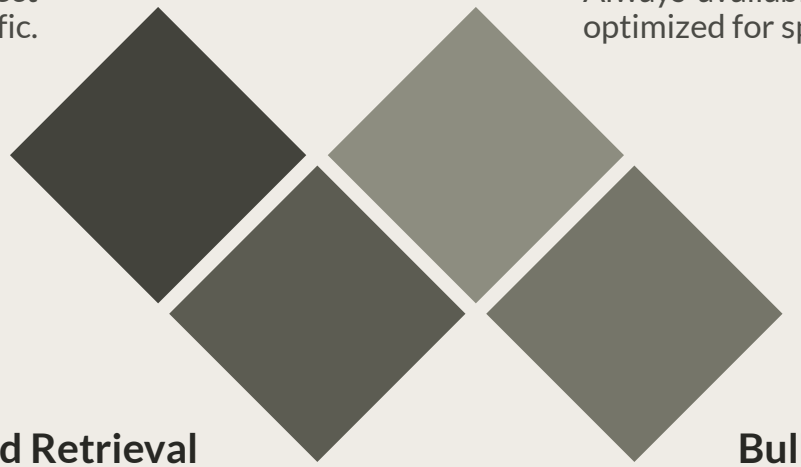
Architecture Overview

Circuit Breaker Routing

Monitor health and redirect traffic.

Lightweight Fallback

Always-available index optimized for speed.



Distributed Retrieval

Fan-out queries to primary shards.

Bulkhead Isolation

Protect fallback capacity from failures.

Each layer is independently designed for failure: the retrieval layer handles fan-out, circuit breakers gate traffic, the fallback index absorbs load, and bulkheads ensure fallback capacity is never exhausted by primary system failures.

The Fallback Index

A **purpose-built, always-available index** designed exclusively for failure scenarios not a stripped-down copy of the primary index.

Lightweight

Smaller footprint, reduced I/O, minimal dependencies

Fast

Optimized for sub-millisecond response under load

Always Available

Isolated from primary system degradation paths

Circuit Breaker–Driven Routing

1. **Closed**

   Normal traffic flows to the primary index. Failures are counted but don't interrupt service.

2. **Open**

   Threshold exceeded. Traffic routed immediately to the fallback index. Primary is isolated to recover.

3. **Half-Open**

   A probe request tests primary recovery. On success, traffic is gradually restored. On failure, circuit reopens.

This enables **graceful degradation** users receive meaningful results from the fallback index rather than errors or timeouts during primary recovery.

Bulkhead-Based Resource Isolation

Borrowed from ship design, the bulkhead pattern helps you **compartmentalize failure domains** so a breach in one section doesn't flood the entire vessel.

Goroutine Pools

Separate worker pools per subsystem prevent one path from exhausting all execution capacity.

Connection Limits

Independent connection pools keep primary and fallback traffic isolated.

Semaphore Caps

Go semaphores enforce hard concurrency boundaries so overload cannot spread.

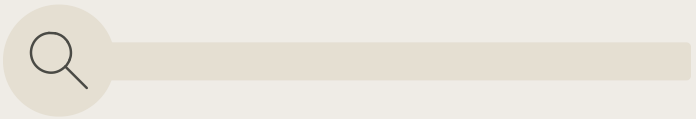
Fallback Protection

Fallback capacity is **never consumed** by primary system overload.

- Separate goroutine pools and connection limits per subsystem
- Semaphore-based concurrency caps in Go enforce hard boundaries
- Fallback capacity is **never consumed** by primary system overload
- Context cancellation propagates cleanly across pool boundaries, preventing goroutine leaks
- Backpressure signals from the fallback pool never bleed into primary request handling
- Resource exhaustion in one bulkhead triggers circuit breaker escalation, not global degradation

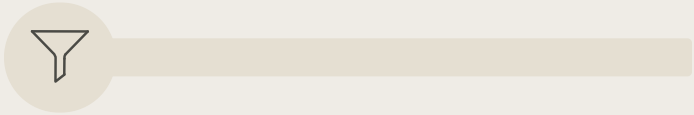
Avoiding Zero-Result Queries

Zero results is the **worst possible search outcome** it signals complete failure while returning HTTP 200. Preventing it requires layered defense.



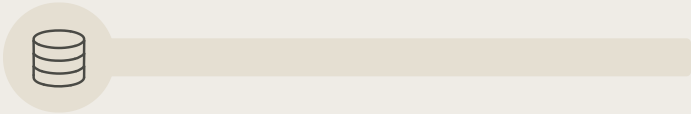
Primary Index

Full fidelity search across all shards with standard ranking



Query Relaxation

Drop low-signal constraints (filters, strict phrases) to broaden recall before escalating



Fallback Index

Lightweight, always-available index surfaces meaningful results even under primary failure

Query-Log-Driven Synonym Optimization

Static synonym dictionaries decay. Query logs are a **living feedback signal** that surfaces what users actually search for and where the index fails them.

- **Recall**: Synonyms expand coverage, surfacing results users expect but exact-match misses
- **Precision**: Over-broad synonyms introduce noise tuning is iterative, not set-and-forget
- High zero-result queries in logs → prime candidates for synonym injection
- Reformulation patterns reveal semantic gaps in the index

Connecting Infrastructure Signals to User Impact

Infrastructure Signals

- Tail latency (p99 / p999)
- Circuit breaker trip frequency
- Shard timeout rates

User-Centric Metrics

- Query reformulation rate
- Zero-result frequency
- Support escalation rate

Infrastructure metrics measure system health. User metrics measure **whether that health translates to a good experience**. Both are required neither alone tells the full story.

A system with p50 latency within SLA can still be silently degrading user experience at the p99 tail. Bridging these signal layers is what separates operationally mature teams from reactive ones.

Early Detection Before Users Feel It

Escalation rates are **lagging indicators** by the time they spike, thousands of users have already experienced failure.

Lagging Indicators

Support escalations, NPS drops, session abandonment visible only after damage is done

Leading Indicators

Tail latency increases, circuit breaker trips, rising zero-result rates detectable before users notice

Proactive Response

Alert on leading signals → trigger fallback routing → prevent user-facing degradation entirely

Key Design Principles

Fallback Is First-Class

Design fallback in from day one not after the first major incident

Design Against Fan-Out

Explicitly bound goroutine concurrency and timeout budgets at every fan-out boundary

Defense in Depth

Circuit breakers gate traffic; bulkheads protect fallback capacity layer both

Measure User Outcomes

System uptime is necessary but insufficient track reformulation rates and zero-result frequency

Query Logs as Feedback

Treat query logs as a continuous signal for synonym tuning and gap detection not a debugging tool

Thank You!

Hima Bindu Yanala · eBay Inc.

Questions & Discussion