EXPERT SOFT
software development

Speed Through the Mess: How to Build Scalable Retrieval in Complex Stacks

Real fixes for data slowdowns
in multi-system environments

When systems scale, speed tends to suffer. Not all at once, but in the small delays, the mismatched data, the dashboards that stall when you need them most.

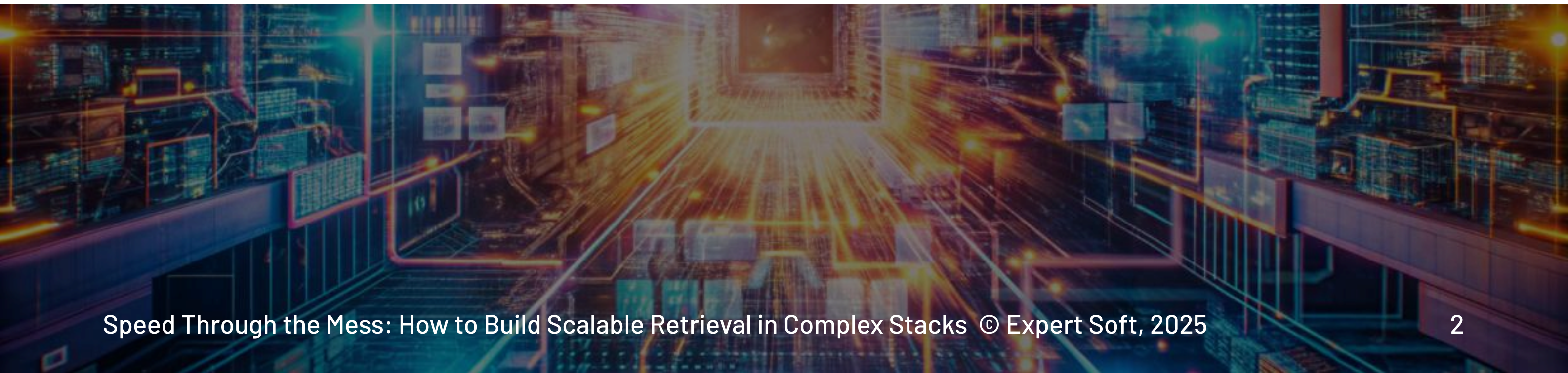
The problem isn't always the code or the hardware. It's how data moves — or doesn't — through the maze of platforms, APIs, and back-end logic holding everything together. It's especially tricky when that maze is a mix of old and new: multiple ERPs, real-time APIs, custom portals, and stacks inherited post-acquisition.

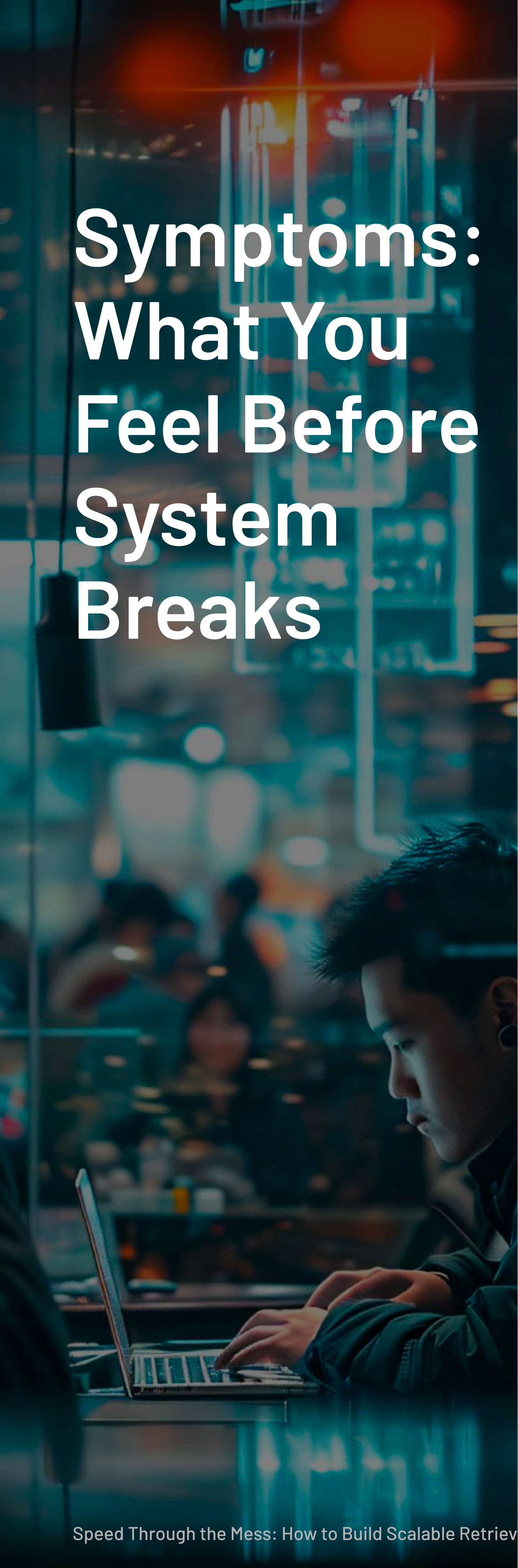
This whitepaper dives into how to make data retrieval fast, stable, and scalable across all that complexity. Not by ripping everything out, but by making the architecture smarter where it counts.

You'll find practical strategies for cutting latency, fixing fragmentation, and keeping your system responsive even when user load and business complexity keep growing. Because when internal systems perform with speed and consistency, your teams can deliver with the same reliability your customers count on.

Every recommendation inside comes from systems already doing the work. No theory, no guesswork, just real patterns that hold under pressure, fine-tuned in the middle of scaling, and built to support companies where performance and trust go hand in hand.

Table of Contents





Symptoms: What You Feel Before System Breaks

Unlike small platforms where data moves between two or three systems without much drama, enterprise setups — well, they swear under their breath and deal with a maze. Data flows in from all sides: ERPs, warehouse systems, finance tools, custom apps, and whatever's still hanging on after the last few acquisitions.

> *What really matters here is how that data gets pulled in. If retrieval is slow or patchy, performance drops and everything connected to it slips.*

At some point, every big system starts to crack. Sometimes it's loud — full outages, major slowdowns. But more often, it's quiet: a wrong search result, a delay no one can explain, another ticket to support that didn't need to happen.

That's why early signals matter. They tell you the system's under stress before it breaks. And from what we've seen across complex architectures, these signals tend to show up in the same places.

SPOT THE SIGNALS EARLY

PRODUCT PAGES GET SLOWER BY THE WEEK

Heavy joins, nested queries, or fragmented cache layers can quietly pile on latency. Multiply that by variants, custom pricing, and regional logic and response times start to climb.

DASHBOARDS STALL OR FREEZE UNDER LOAD

When analytics pipelines aren't built for concurrency or real-time pulls, even basic dashboards choke. Especially if data is stitched together on the fly from multiple systems.

PRICING OR AVAILABILITY DOESN'T MATCH ACROSS CHANNELS

Usually caused by delayed syncs, inconsistent caching, or APIs hitting stale or incomplete records.

SYSTEM SLOWS DOWN DURING ACQUISITIONS OR MIGRATIONS

New brands or systems get plugged in, and suddenly nothing feels stable. Behind the scenes: conflicting data models, bloated imports, or legacy systems that can't keep up with the integration layer.

SEASONAL TRAFFIC OR PROMOTIONS SLOW DOWN THE SYSTEM

Sudden spikes expose the weak spots, such as cache invalidation issues, memory leaks, or services that scale vertically instead of horizontally.

These symptoms don't just slow things down — they signal that your architecture isn't keeping up with the weight it's carrying. Whether it's growth, agility, or just delivering the basics without friction, fragile data retrieval stands in the way.

The specifics may vary, but the core issue is the same: if the foundation can't scale, neither can everything built on top of it.

Architectural Landmines: What’s Really Behind the Lag

The symptoms you’ve seen in the previous section aren’t random. They’re surface-level signals of deeper architectural strain. And as your setup grows — across units, regions, and or plugging in new platforms post-acquisition — those weak spots only get louder.

Through audits and implementations across retail, pharma, and industrial platforms, we’ve seen the same root issues come up again and again. Different stacks, same bottlenecks.

Here’s what’s usually behind the slowdown.

EXTERNAL DEPENDENCIES

Modern platforms rely on third-party APIs for everything from personalization to geolocation and payment validation. But each external dependency is a performance wildcard: it can slow down page rendering, introduce inconsistencies, or fail silently, making troubleshooting harder.

> *The more real-time your experience needs to be, the more these delays start to hurt.*

EXAMPLE FROM PRACTICE

A high-traffic ecommerce brand used third-party APIs for geolocation and store locator functionality. During peak sales periods, these external services became a bottleneck, introducing random delays, occasional timeouts, and making storefront performance unpredictable. The team also faced rising costs tied to request volume.

We replaced the external calls with lean internal services that replicated the core features. This cut latency across pages, eliminated failure points during traffic spikes, and gave the team full control over response times and scaling.

INSUFFICIENT OBSERVABILITY

In complex, distributed systems, problems rarely show up where they start. Without proper tracing, slow-query logging, or cross-service correlation, bottlenecks hide in the noise. Performance issues go unnoticed until they hit users, and even then, it's hard to pinpoint what actually went wrong.

> *Lack of visibility makes troubleshooting reactive and time-consuming, especially when data flows across multiple platforms and asynchronous services.*

EXAMPLE FROM PRACTICE

A large ecommerce platform struggled with delays in third-party data syncs that occasionally disrupted storefront content, but the root cause was never clear. Log retention was short, and there were no consistent trace IDs to follow events across services.

We extended log retention and added trace identifiers to async messages. This gave the team end-to-end visibility, making it easier to trace issues in real time and resolve them before they reached the user.

ETL AND BI OVERLOADS

Reporting and integration jobs often share infrastructure with live systems.

> *When ETL pipelines or BI tools query production databases directly, especially during business hours, they add load where performance matters most.*

As complexity grows, so does the risk of collisions between back-end processes and customer-facing operations. Without clear scheduling, data isolation, or job optimization, these background tasks can quietly degrade performance.

EXAMPLE FROM PRACTICE

A European luxury cosmetics brand ran a daily pricing job that ballooned from minutes to over three hours after new calculation logic was introduced. As the job expanded, it began to interfere with storefront operations and slow down user interactions.

A parallelized reimplementation reduced runtime to under 20 minutes, enabling daily execution again.

DATA BLOAT AND LACK OF TTL

As systems grow, so does the data they store — session logs, old configurations, abandoned carts, audit records. Without clear data lifecycle rules or TTL (time-to-live) policies, this accumulation clogs up databases and slows down operations. Over time, even simple queries have to sift through irrelevant history, putting strain on indexing and storage.

EXAMPLE FROM PRACTICE

A leading jewelry retailer noticed that order lookups and reporting operations were slowing down, especially during high-traffic periods. Years of historical data, including completed orders, returns, and stale session records, were stored in the same tables used for live operations. With no archival strategy in place, even routine queries became heavier over time.

We introduced a smart cleanup flow that moved aged records into an archive layer outside transactional storage. Query speed improved immediately, and the system became more predictable under load.

UNCOORDINATED CACHING LAYERS

Caching is one of the fastest ways to improve performance, but only when it's consistent. In many enterprise setups, caches are added reactively at different layers: database, back-end, API gateway, CDN, front-end. Without a clear strategy, they fall out of sync, leading to stale data, invalidation delays, or even conflicting results across channels.

EXAMPLE FROM PRACTICE

An omnichannel retailer faced product availability mismatches during flash sales due to uncoordinated caching across five layers (database, back-end service, API gateway, CDN, and front-end). Updates weren't properly propagated, leading to stale data and user confusion.

We introduced centralized cache invalidation triggers tied to specific data change events, along with smarter TTL settings based on traffic patterns. This brought consistency across channels and kept performance stable under pressure.

INEFFICIENT DATA MODELS AND QUERIES

Legacy schemas, over-normalization, and missing indexes are classic culprits. As platforms evolve, data models often become bloated with excessive joins and complex relationships, slowing down even the simplest queries under load.

> “*That lag creeps into the user experience: carts take longer to load, dashboards hang, and time-sensitive operations get stuck waiting on data that should be instant.*”

EXAMPLE FROM PRACTICE

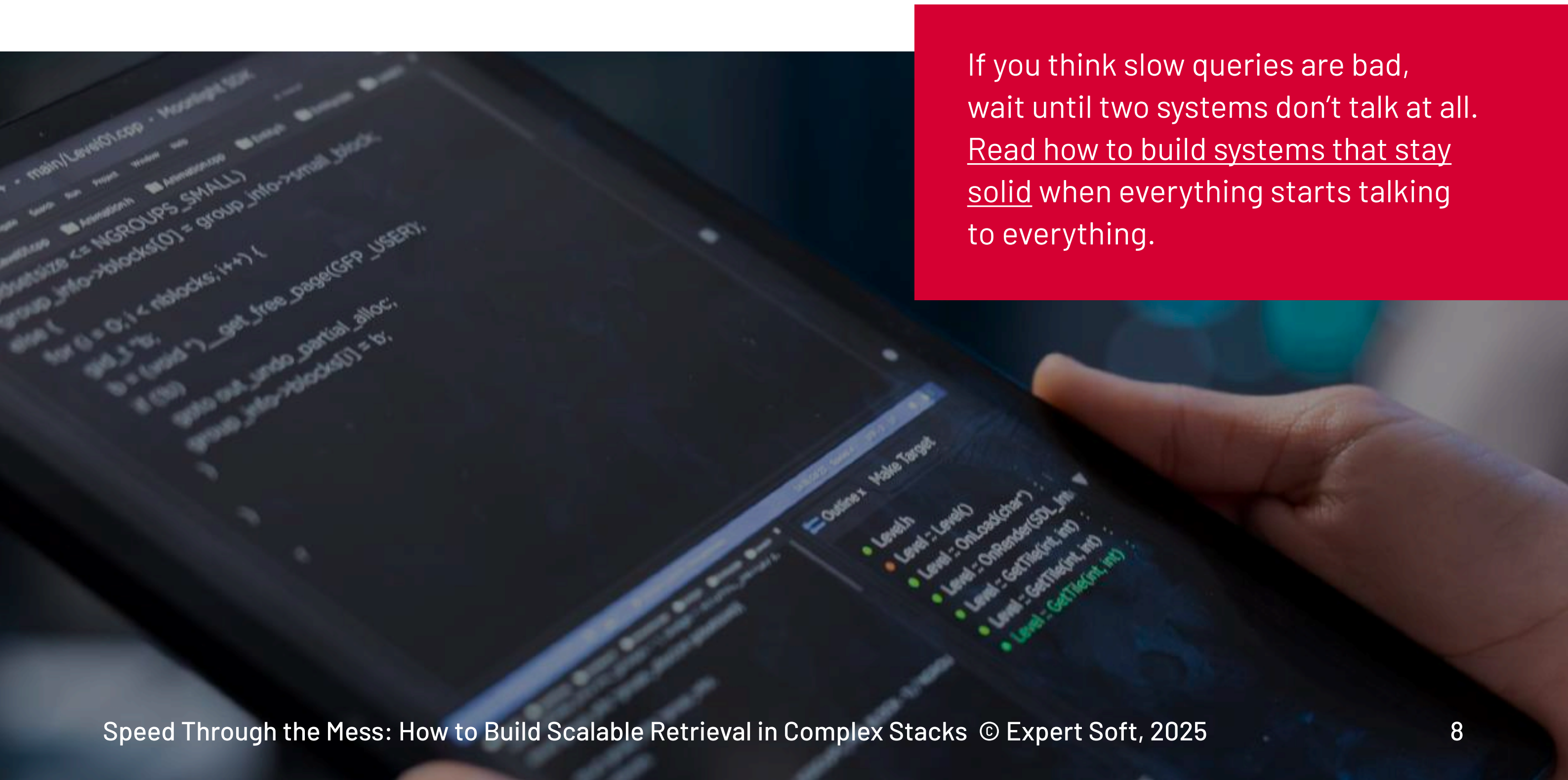
A renowned beauty retailer faced delays in checkout and mini-cart views. The cart data was stored in a highly normalized structure, requiring multiple joins and subqueries to reconstruct each session. As the load increased, these queries became a bottleneck.

We redesigned the model to store cart data as compact JSON documents, minimizing joins and making the structure cache-friendly. This change improved both performance and system resilience, stabilizing performance under load and enhancing the overall user experience during peak activity.

These issues don't always show up early, but at scale, they become impossible to ignore. What held up fine at 10,000 users can buckle at a million if the architecture isn't ready for it.

The good news is that most of these pitfalls can be fixed without tearing everything down — it just takes the right changes in the right places.

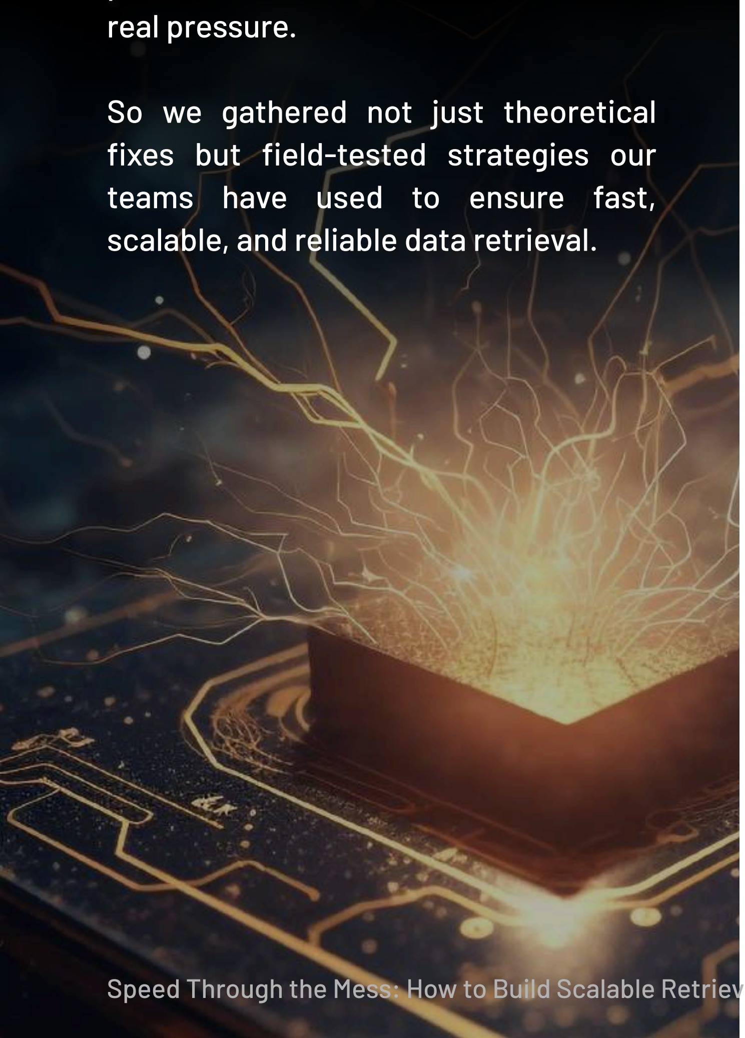
If you think slow queries are bad, wait until two systems don't talk at all. Read how to build systems that stay solid when everything starts talking to everything.



How To Survive When the Load Hits Hard

Once you know where the cracks form, the next step is knowing how to reinforce them on a real scale, not just short-term fixes. We've seen the same problems play out across high-load platforms, and we've solved them under real pressure.

So we gathered not just theoretical fixes but field-tested strategies our teams have used to ensure fast, scalable, and reliable data retrieval.



REFACTOR WHAT YOUR QUERIES DEPEND ON

Scaling starts with removing what doesn't serve speed. Instead of adding horsepower, strong systems streamline what their queries actually rely on — reducing clutter, reshaping documents, and aligning indexing to how data is used in real-world scenarios.

MOVES TO TAKE

1. **REMOVE COMPLEXITY THAT DOESN'T SERVE THE USER**

Audit where localization, references, or deep joins are used for internal logic and simplify when no user-facing need exists.

EXAMPLE

Challenge: Performance issues due to stored non-user-facing attributes, leading to hundreds of unnecessary joins.

What we did: Removed unnecessary localization logic on internal attributes by switching to non-localized fields where appropriate.

Impact: Query load dropped by over 100× and memory usage fell under peak traffic.

2. STRUCTURE DATA FOR THE WAY IT'S QUERIED, NOT FOR THE WAY IT WAS MODELED

Split bloated documents and shift from generic queries to targeted, purpose-built ones that return only what the user needs.

EXAMPLE

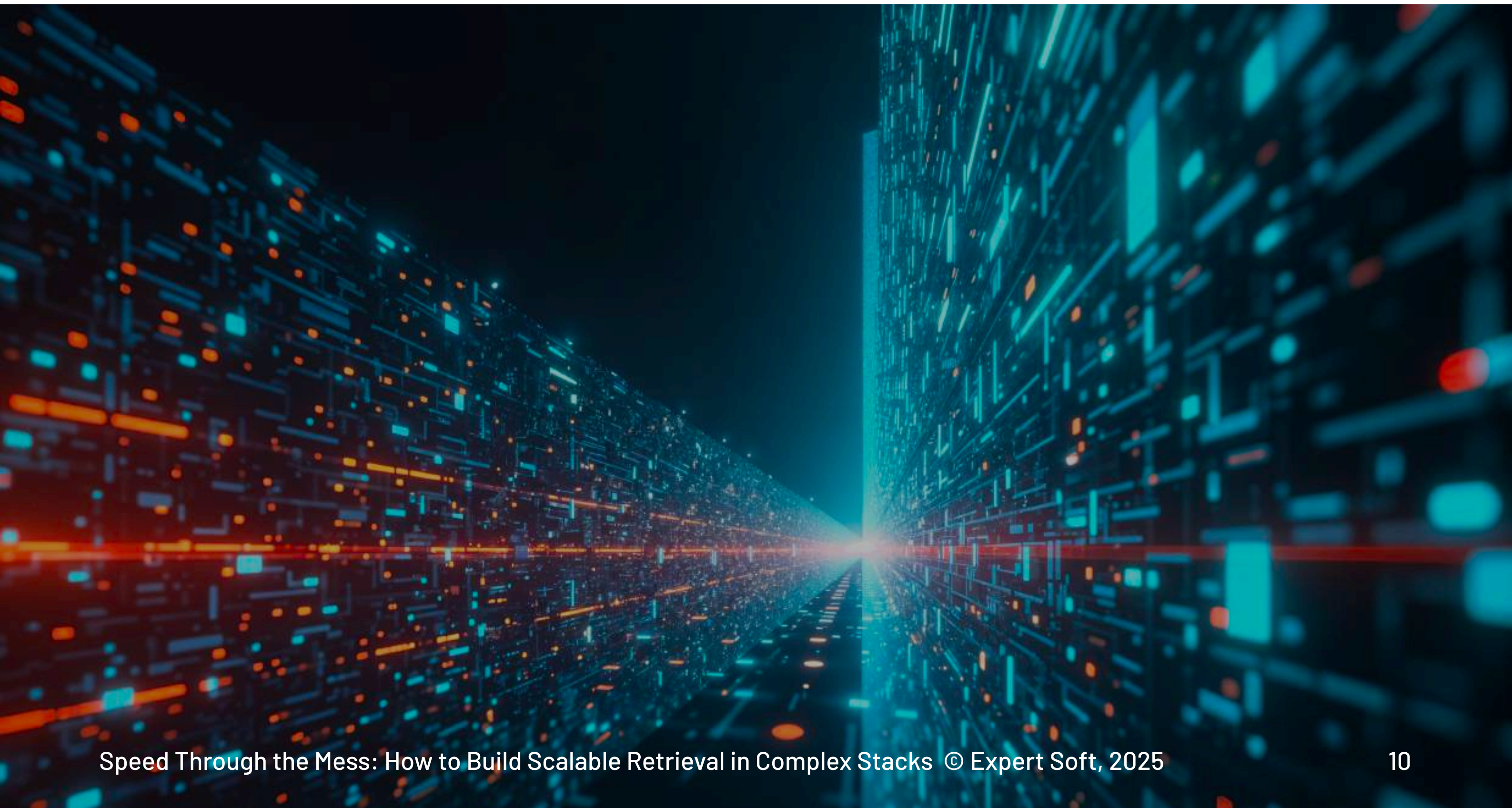
- **Challenge:** Product detail and category pages were slow to render, especially under traffic bursts.
- **What we did:** Combining Flexible Search query optimization with targeted use of Solr, we split monolithic product documents into smaller segments and optimized Flexible Search queries to target only the needed data.
- **Impact:** PDP response times improved significantly, making performance stable even during seasonal spikes.

3. ALIGN YOUR INDEXING WITH REAL-WORLD ACCESS PATTERNS

Run audits to detect which queries dominate system load, then update indexes to match actual, not theoretical, usage.

EXAMPLE

- **Challenge:** Frequent query lags and high CPU usage.
- **What we did:** Performed a deep database index audit to analyze production traffic, based on which we rebuilt indexes to match real-world access patterns.
- **Impact:** Query speed improved and CPU overhead dropped across all core operations.



COORDINATE YOUR CACHING AT EVERY LAYER

Caching is powerful, until it's fragmented. The fastest systems don't cache aggressively, but intelligently and in sync. That means aligning caching behavior across front-end, back-end, and infrastructure to reduce duplication, stale data, and unnecessary load.

MOVES TO TAKE

1. MAKE INVALIDATION PREDICTABLE, NOT REACTIVE

Create shared rules and batch logic between back-end and front-end layers to ensure that updates reflect quickly and reliably.

EXAMPLE

- **Challenge:** Updated product and content data took too long to appear on the storefront due to uncoordinated cache invalidation across layers.
- **What we did:** Implemented clear invalidation rules across SAP back-office and Spartacus front-end layers that triggered cache clears at both ends based on specific business events, and scheduled batches to avoid collisions.
- **Impact:** Content propagation delays were cut by 80%.

2. USE IN-MEMORY CACHING FOR LOW-CHURN LOGIC

Offload frequently accessed but rarely changed data, such as configurations, into lightweight, memory-first caches to prevent unnecessary back-end hits.

EXAMPLE

- **Challenge:** Back-end responsiveness dropped as the system reevaluated thousands of config entries on every call.
- **What we did:** Moved key configuration logic into in-memory caching layers, decoupling it from main transactional flows.
- **Impact:** Back-end load decreased and response times improved noticeably under concurrent usage.

However, not all cache is a win.
We've seen teams go fast...
straight into chaos.
Learn how to spot the traps

ENSURE DATA GETS WHERE IT NEEDS TO GO INSTANTLY

When systems grow, syncing data in real time becomes non-negotiable. Efficient architectures shift from bulk jobs and blind retries to event-based flows, delta tracking, and reliable messaging built to handle real-world edge cases.

MOVES TO TAKE

1. STREAM CHANGES AS EVENTS, NOT IN BATCHES

Replace batch-driven sync jobs with change-event pipelines that reflect updates across systems almost instantly — ideal for dashboards, reporting, and user-facing tools.

EXAMPLE

Challenge: Customer-facing dashboards showed stale data due to periodic batch jobs that introduced minutes-long delays.

What we did: Implemented CDC using AWS DMS and Kafka to stream updates from MySQL to MongoDB, replacing the batch ETL sync.

Impact: Dashboards now reflect updates within one minute, boosting reliability and user confidence.

2. MATCH YOUR MESSAGE BROKER TO THE CRITICALITY OF THE DATA

When data loss isn't an option — like with orders or payments — pick messaging systems with built-in retries, persistence, and delivery tracking rather than just raw throughput.

EXAMPLE

Challenge: A platform experienced occasional order loss during network disruptions, with no way to trace failed deliveries.

What we did: Switched from Kafka to ActiveMQ to improve delivery reliability in volatile network environments, leveraging ActiveMQ's built-in persistence and message acknowledgment features.

Impact: Order flow became more reliable under unstable network conditions.

3. SWITCH FULL IMPORTS TO DELTA-BASED UPDATES

For high-volume domains, importing only changed records, instead of refreshing everything, saves time, bandwidth, and CPU.

EXAMPLE

Challenge: Importing 60,000+ promotions caused slowdowns and prolonged downtime during catalog updates.

What we did: Implemented delta sync logic using checksums and timestamps to update only what changed.

Impact: Sync time dropped from several minutes to just seconds.

4. CENTRALIZE AND ORCHESTRATE YOUR BACKGROUND SYNC TASKS

Unify scattered background tasks under one scheduler to avoid collisions, improve timing precision, and gain visibility across your sync operations.

EXAMPLE

Challenge: Background jobs were scattered across services, leading to overlapping syncs and unstable load behavior.

What we did: Consolidated fragmented sync tasks into a single centralized scheduler to improve visibility and scheduling precision.

Impact: Sync reliability increased and maintenance became easier across the board.

CLEAR OUT WHAT DOESN'T NEED TO STICK AROUND

When session data, temporary carts, and irrelevant records hang around too long, even simple queries get bogged down. Scalable platforms bake in rules to retire what's no longer useful, keeping core operations lean.

MOVES TO TAKE

1. SET TTL POLICIES FOR SESSION-BASED DATA

Apply automatic expiration rules to clear out data like "recently viewed" or user history after it's no longer relevant to business logic or user experience.

EXAMPLE

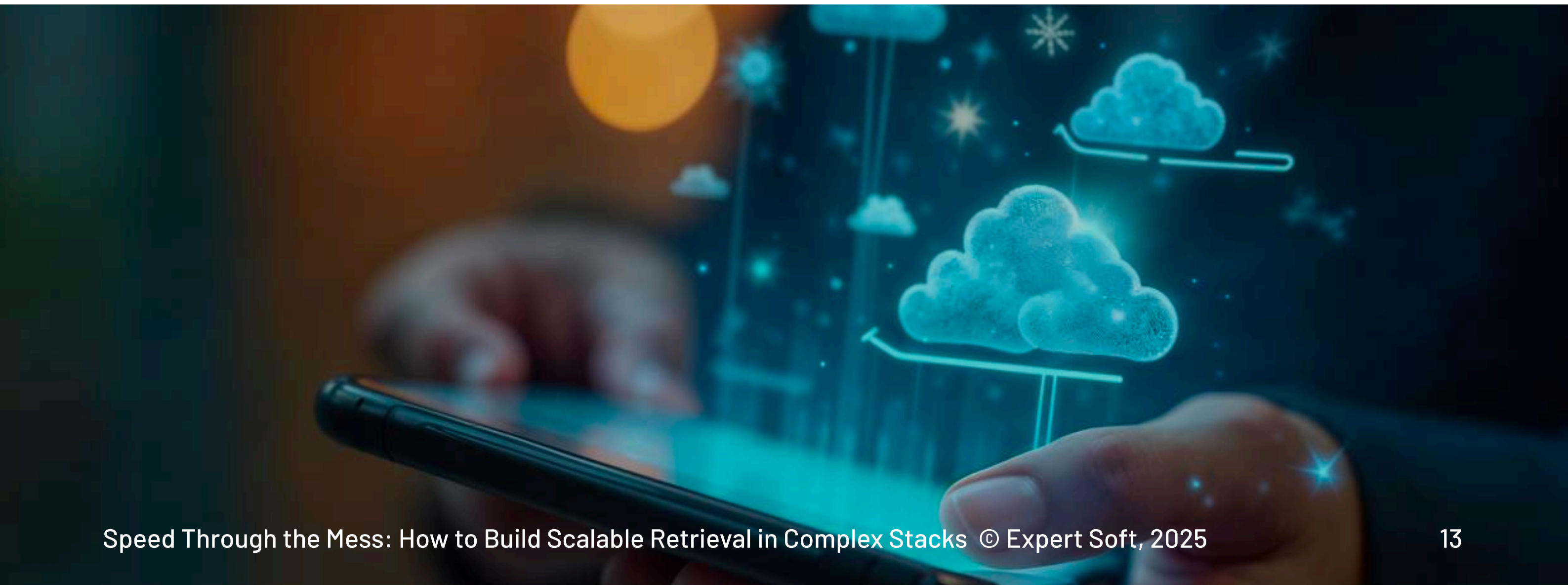
- **Challenge:** The "recently viewed products" table kept growing with stale records, putting pressure on database performance.
- **What we did:** Introduced TTL logic via timestamp-based cleanup jobs, ensuring expired data was regularly purged from the storage.
- **Impact:** Session bloat was eliminated and query speed improved across related operations.

2. AUTO-CLEAN TEMPORARY CARTS AND ABANDONED ACTIVITY

Define automated logic to regularly remove carts and transient data that no longer lead to conversion or interaction, keeping your transactional tables focused and efficient.

EXAMPLE

- **Challenge:** A growing number of empty and abandoned carts triggered performance issues.
- **What we did:** Introduced an auto-deletion rule for inactive and empty carts, reducing table volume.
- **Impact:** Cart-related queries became faster and more consistent during traffic spikes.



KEEP DELIVERY FAST EVEN WHEN THE CONTENT ISN'T

The more media, integrations, and redirects your system handles, the heavier each request gets. Scalable systems don't try to carry everything upfront — they shift non-critical loads downstream, outsource static tasks to the edge, and simplify what the application has to process.

MOVES TO TAKE

1. DEFER MEDIA LOADING UNTIL IT'S NEEDED

Use lazy loading and viewport-based logic to delay non-essential content, like images or embedded assets, so core layout and logic load first.

EXAMPLE

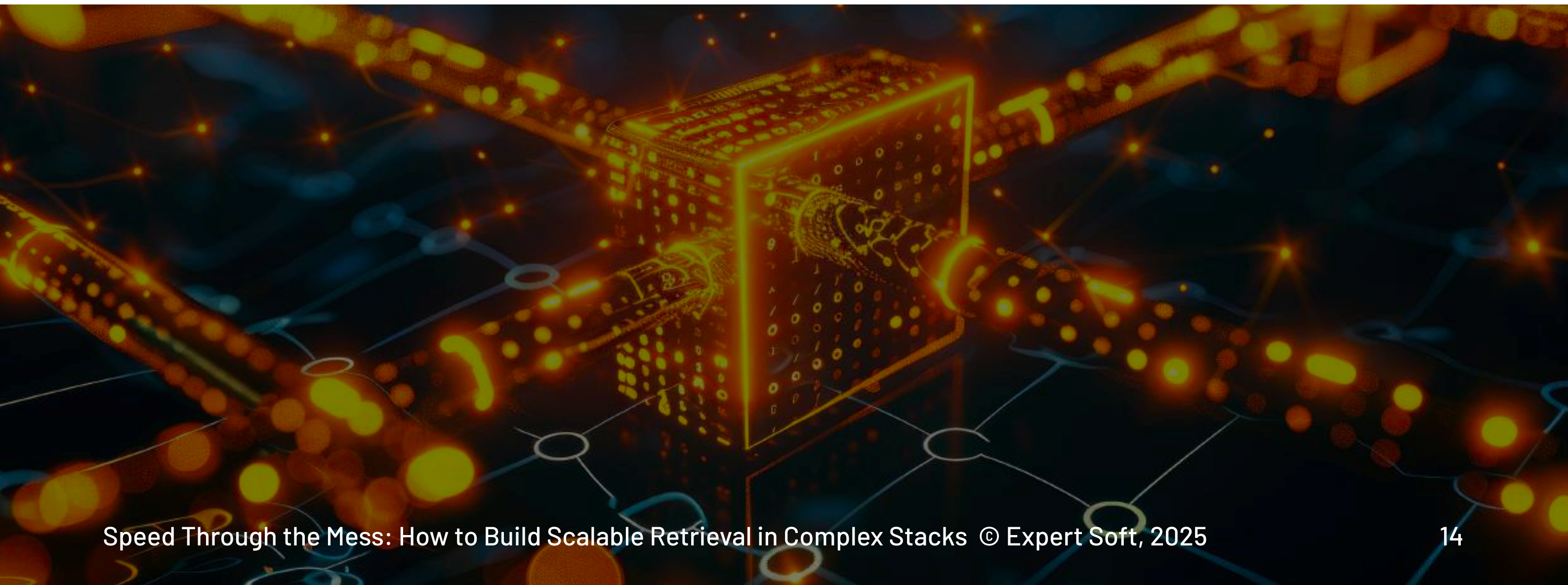
- **Challenge:** Category and campaign pages loaded slowly due to dozens of media assets rendering upfront.
- **What we did:** Implemented lazy loading for image assets across key templates to prioritize visible content and reduce blocking.
- **Impact:** Initial page load times improved by 5–12%, especially under mobile and high-traffic scenarios.

2. PUSH LIGHTWEIGHT LOGIC TO THE EDGE

Offload routing, redirects, and simple rules to the web server or CDN layer, freeing your app layer to focus on core business logic.

EXAMPLE

- **Challenge:** Thousands of redirects were processed through the application layer, creating unnecessary load and slowing down SEO-critical paths.
- **What we did:** Migrated redirect logic from SAP Commerce Cloud into Apache config rules, shifting it outside the app's request flow.
- **Impact:** Response times for redirected URLs improved, and app-level performance became more predictable.



MAKE IT EASY TO SEE WHAT'S SLOWING YOU DOWN

Fast systems don't just run well — they make it obvious when something isn't. Teams that scale successfully invest in tooling that gives them full visibility: across services, layers, and devices. With strong observability in place, they can catch issues early, track regressions before they spread, and debug without the guesswork.

MOVES TO TAKE

1. IMPLEMENT PLATFORM-WIDE TRACING AND CODE QUALITY SCANNING

Use integrated observability and security tools to catch performance regressions, bad deploys, or risky code changes before they cause downstream impact.

EXAMPLE

- **Challenge:** The team lacked a unified view of system health and code-level risks, delaying root cause detection.
- **What we did:** Deployed Datadog for system monitoring and added SonarQube, Veracode, and X-Ray Scan to tighten quality control across environments.
- **Impact:** Regressions and security risks were caught earlier in the cycle, reducing deployment delays and improving system stability.

2. UNCOVER FRONT-END ISSUES BEFORE USERS REPORT THEM

Use client-side analytics and visual replay tools to identify slowdowns, rendering glitches, or broken flows that aren't visible through back-end logs.

EXAMPLE

- **Challenge:** Bugs and layout issues appeared inconsistently across browsers, making them hard to reproduce and fix..
- **What we did:** Introduced Content Square to centralize front-end logs and replay user behavior across devices.
- **Impact:** Errors became easier to trace and resolve, shortening response time and reducing support load.

You've built the muscle —
now protect the core. Here's how to keep your SAP data sharp, synced, and trusted.

From Insight to Action — Scalability Checkpoint

Scalable data retrieval has nothing to do with one silver bullet, as it's about how everything connects.

Clean models, smart caching, synced updates, background jobs that don't clash, and visibility when things go sideways — that's what keeps systems fast when complexity kicks in.

You've seen the symptoms. You've seen how they're solved. If you're ready to take a closer look at your own setup, this checklist offers a quick-hit way to surface the areas that matter most.

Use it as a gut check or a starting point.

DATA SYNC & PROCESSING

- □ Do you use event-based processing (CDC, MQs) instead of daily ETLs?
- □ Are heavy cron jobs parallelized or migrated to stream processing?

DATA ARCHITECTURE

- □ Are your database schemas optimized for read-heavy access patterns?
- □ Are non-essential joins replaced with denormalized fields or document models?
- □ Do you partition or shard large datasets to reduce query load?

CACHING

- □ Are all major cache layers (CDN, back-end, DB) coordinated?
- □ Is invalidation logic based on clear TTL rules, event triggers, or both?
- □ Do you review and test caching behavior regularly to avoid silent failures?

OBSERVABILITY & MONITORING

- □ Do you have structured logs, distributed tracing, and proactive alerting in place?
- □ Are slow queries and system hotspots tracked and regularly analyzed?

DATA RETENTION

- □ Do you enforce archiving for historical or low-use records?
- □ Are stale sessions, logs, and unused records cleaned up automatically?

If most of these boxes are checked, you're likely in a solid place. If not, the good news is that improvements are achievable. You can make incremental yet impactful changes to deliver fast, scalable data retrieval under enterprise load.

At Expert Soft, we work with complex systems every day, helping teams untangle what's slowing them down and put scalable foundations in place. If you ever need a second set of eyes, we're around.

About Expert Soft

Expert Soft is a targeted ecommerce software delivery company, partnering with Fortune 500 companies and global corporations across the US and EU. With SAP Commerce Cloud and Java as our backbone, we know how to ensure scalable and high-performing solutions that can handle 1 mln requests per second, delivering a smooth customer experience.

Developing a payment engine that saved our client about $100 million in operational expenses, ensuring multi-country platform support, adapting solutions for new market entry with tailored enhancements — these are just a few of the challenges our specialists tackle.

We aim to deliver more than a software system. We aim to deliver tailored solutions that maximize profitability within available resources. Our success is driven by:

TEAM STRENGTHS

- All our engineers have a university background
- Specialists excel their skills in our training LABs
- Perfect English skills
- Ready to help 24/7

CLIENTS

We work with corporations around the world with revenue of over $20 billion and 150K+ employees.

APPROVALS BY AUDITS

Our ongoing work with corporations is consistently validated through rigorous audits, both by internal teams and Big 4 consulting firms.

HIGH-LEVEL SECURITY

Approved by assessments from global companies, who are leaders in their respective industries.

BUDGET EFFICIENCY

By carefully aligning technology investments with your business goals, we ensure optimal value and cost-effectiveness.

PROFESSIONAL TEAM

No offshore outsourcing and our team's average tenure of 4+ years means you get seasoned problem-solvers, not just coders.

EXPERT SOFT EXCELS IN

- PAYMENT ENGINE
- MICROSERVICES ARCHITECTURE
- CONTENT MANAGEMENT
- REDESIGN
- E-COMMERCE PLATFORM
- HEADLESS COMMERCE
- MICRO UI FRONT-END
- MIGRATION&INTEGRATION

OUR TECH CORE

FRONT-END

HTML, CSS, JavaScript (Angular, React, Vue, Next, TypeScript, Jquery), Spartacus

BACK-END

Java EE, Spring, SAP Commerce (Cloud), Node.JS.

DEVOPS

Docker, Kubernetes, CI/CD

UX/UI DESIGN

UX Research, UI Design, Figma, Adobe, Sketch

QUALITY ASSURANCE

Manual Testing, Test Automation

TARGETED DOMAINS

RETAIL

TELECOM

HEALTHCARE

FINTECH

WHOLESALE

MANUFACTURING

SHARED PATHS, LASTING ECOM VICTORIES

LET'S TALK SOLUTIONS!

EKATERINA LAPCHANKA
Chief Operating Officer
kate.lapsenco@expert-soft.com

+1 585 4997879 ✆
+371 25 893 015 ✆

PAVEL TSARYKAU
CEO & Founder
of Expert Soft

Let’s connect in

expert-soft.com

LinkedIn in