Booking.com

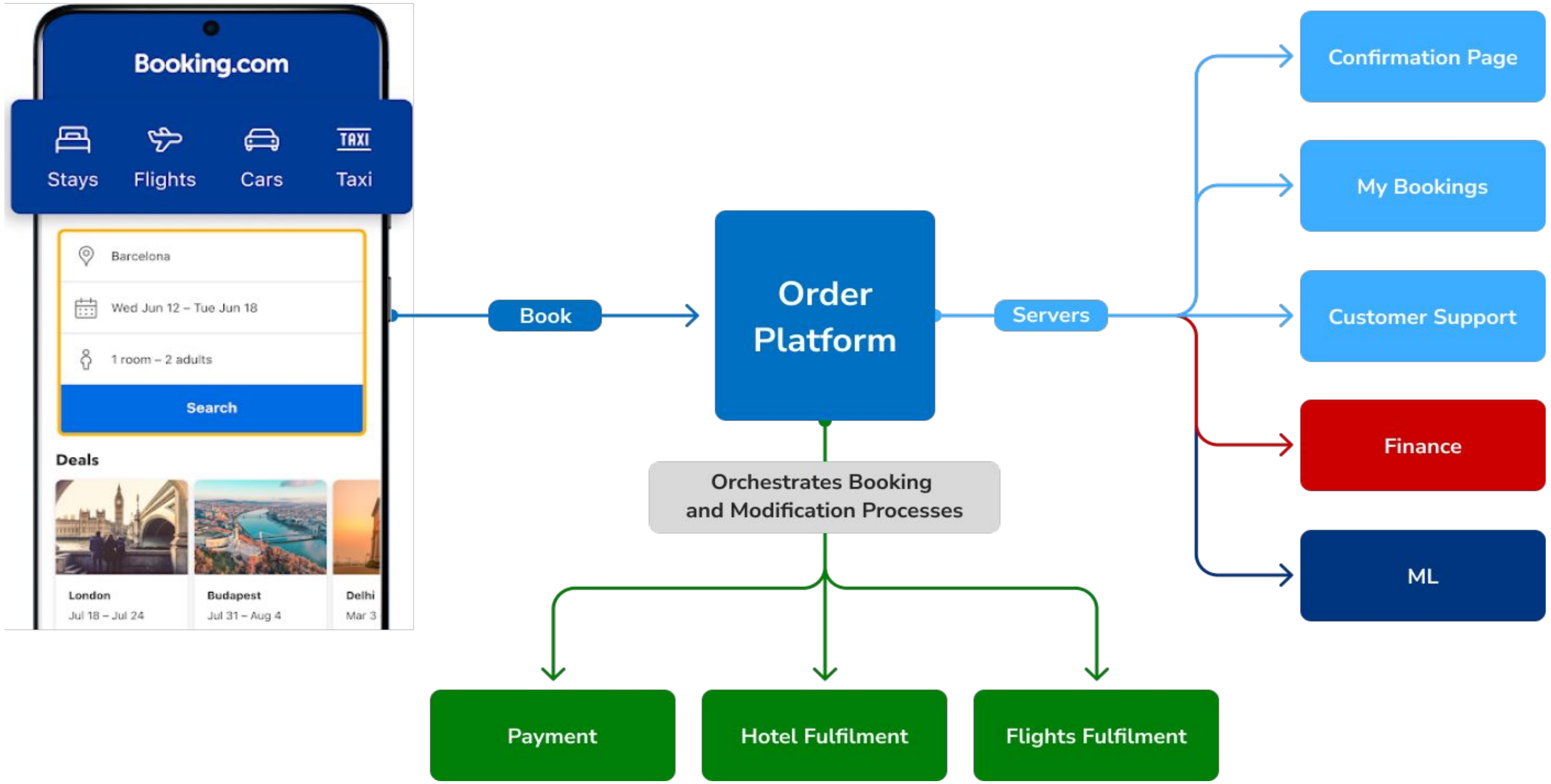
Booking.com's Modernization and Simplification Journey with CockroachDB

Mahmoud Nagib
Principal Software Engineer

RoachFest25

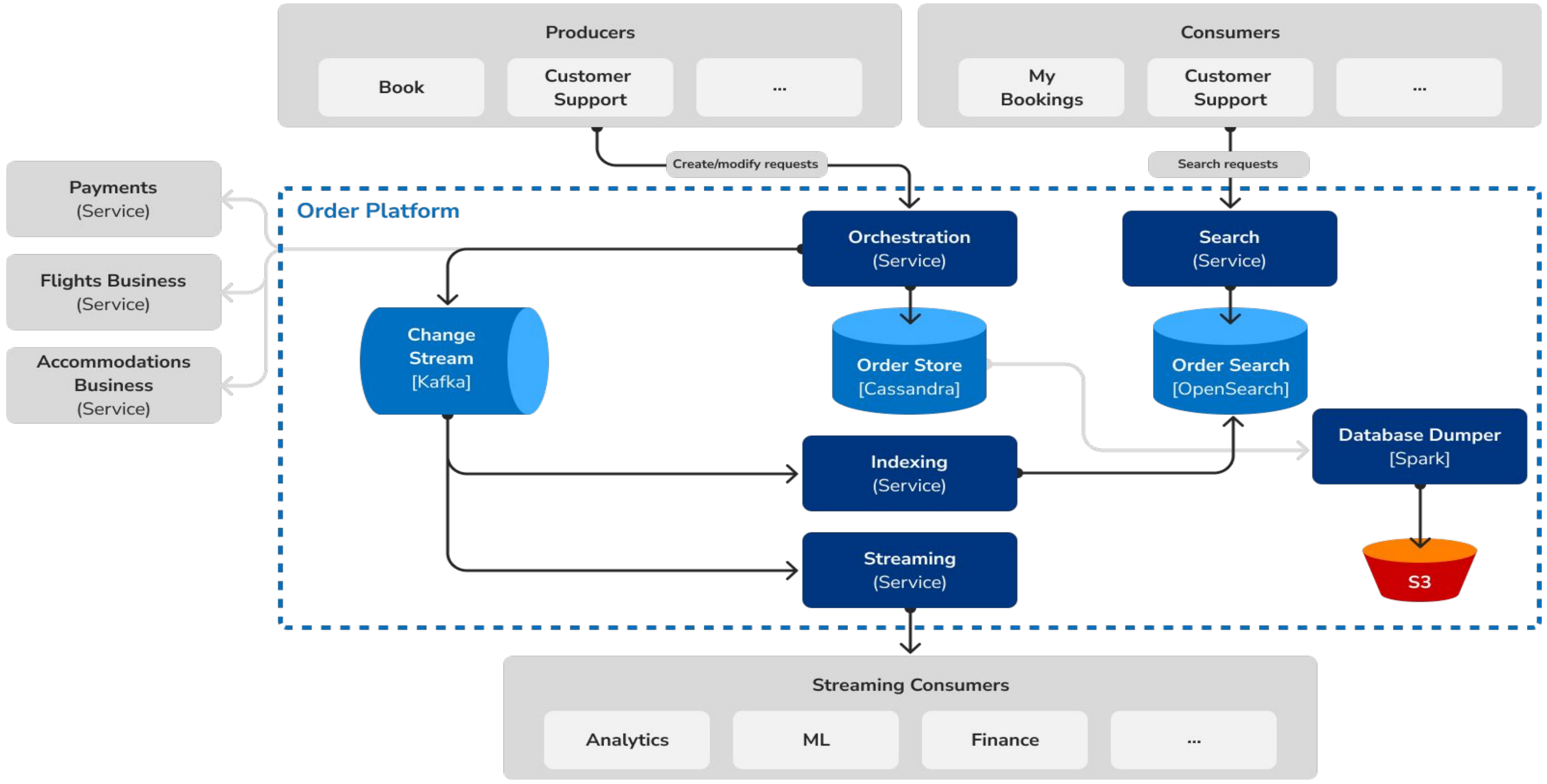
What do we do in Order Platform?

Overview

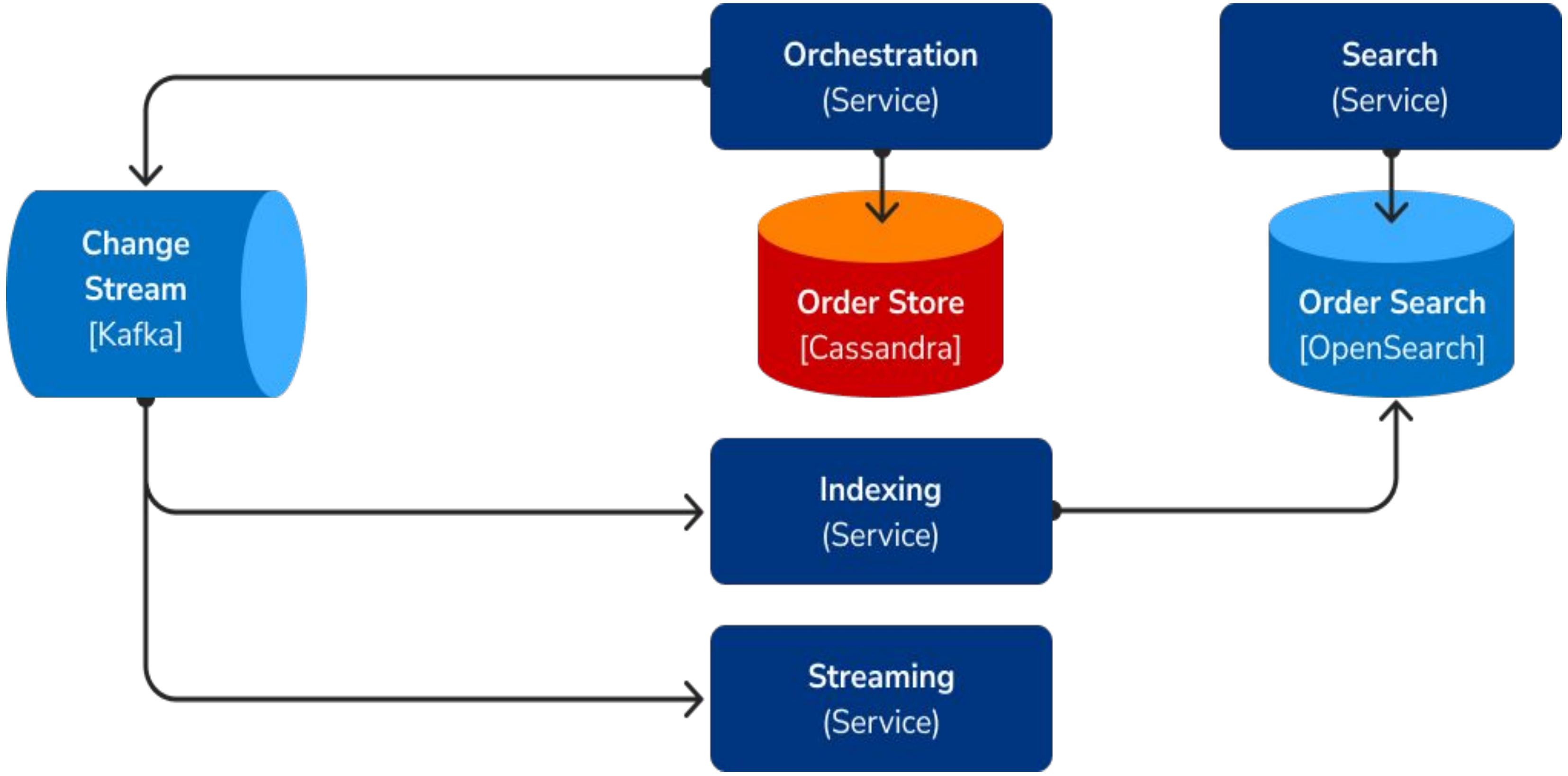

Booking.com
Stays
Flights
Cars
Taxi
Barcelona
Wed Jun 12 – Tue Jun 18
1 room – 2 adults
Search
Deals
London
Jul 18 – Jul 24
Budapest
Jul 31 – Aug 4
Delhi
Mar 3
Book
Order Platform
Servers
Orchestrates Booking and Modification Processes
Confirmation Page
My Bookings
Customer Support
Finance
ML
Payment
Hotel Fulfilment
Flights Fulfilment

The Call to Change: Why we are migrating?

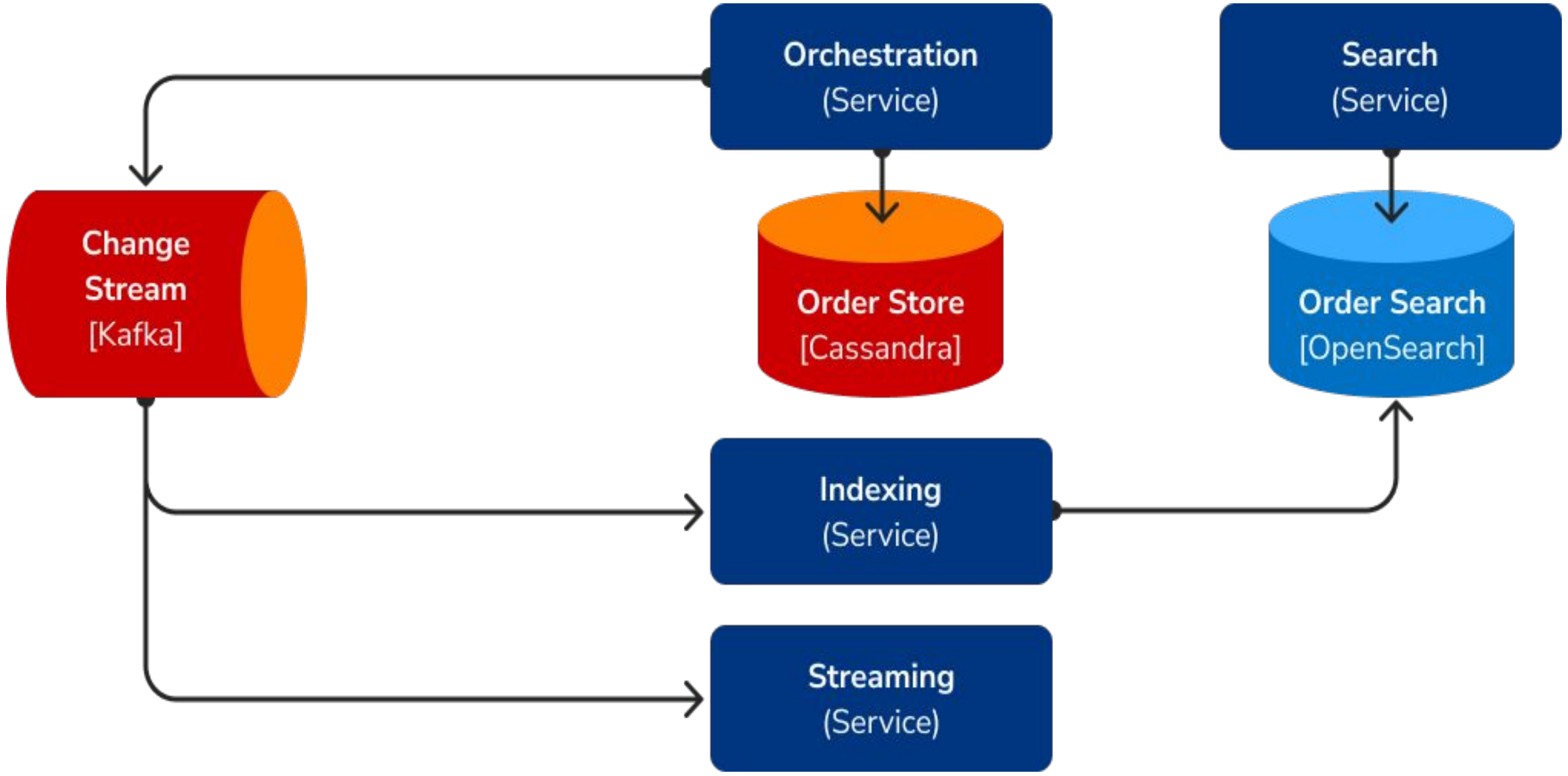
High Level Architecture



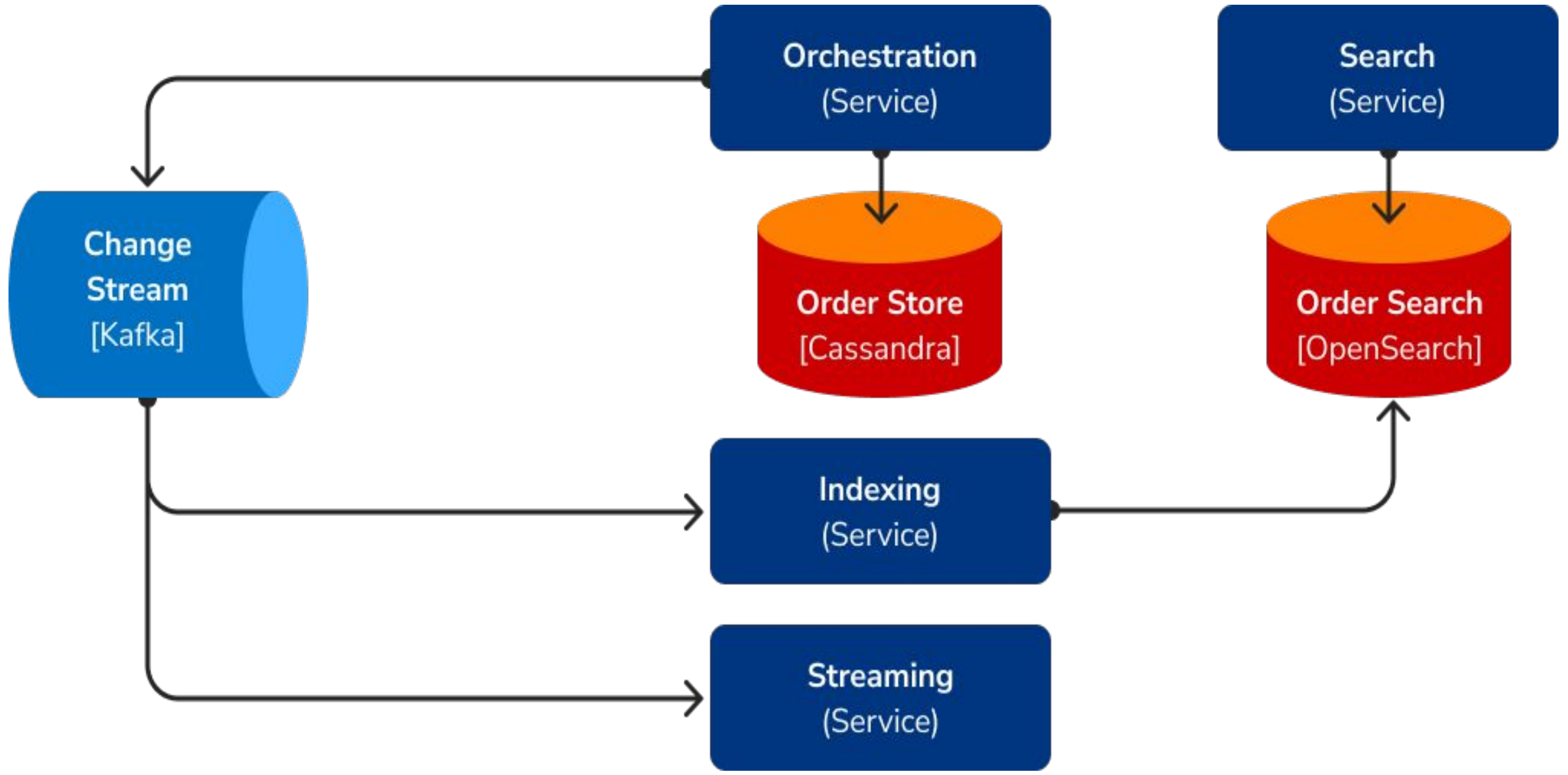
Lack of ACID Transactions



Consistency Across Multiple Stores



Limited Retrieval Patterns



Summary of Pain Points

Cost

Replicating massive data sets across multiple databases is QUITE EXPENSIVE

Time

Building and maintain additional systems.

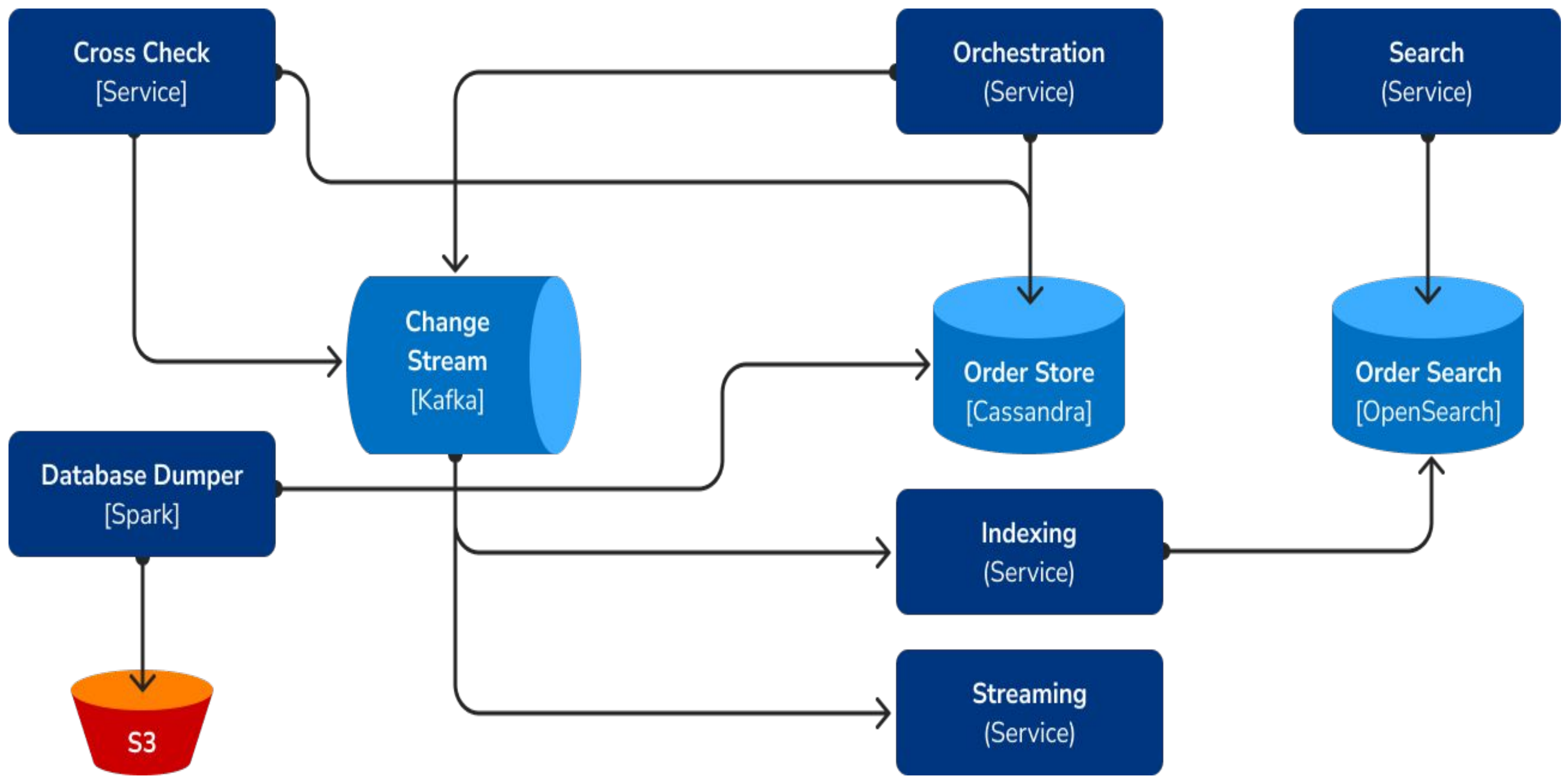
Troubleshooting becomes harder and more time consuming.

Cognitive Load

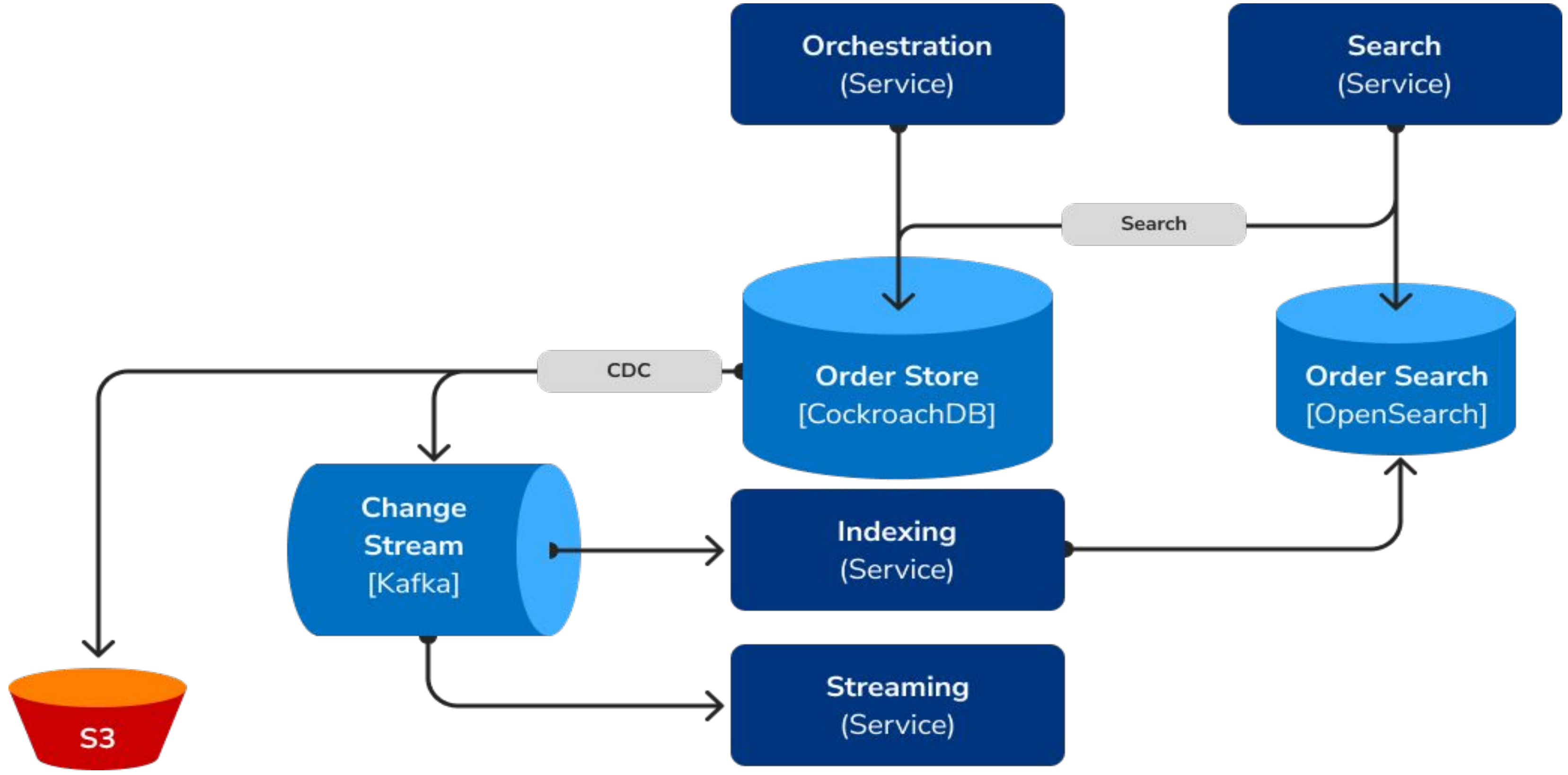
Excessively complex systems take a lot of mental power to understand, onboard new team members and to change

Simplifying the Reference Architecture

Current Architecture

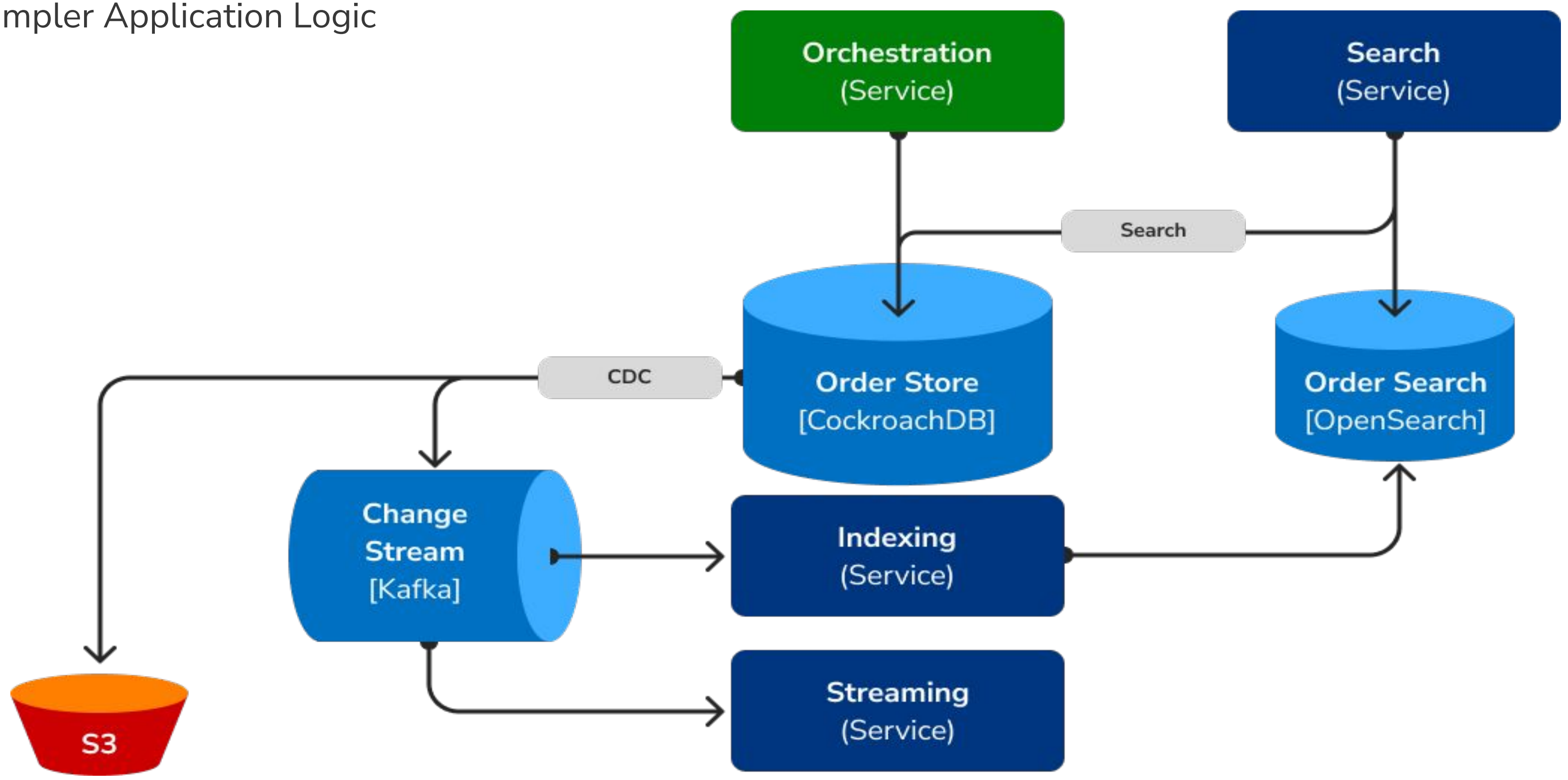


New Architecture



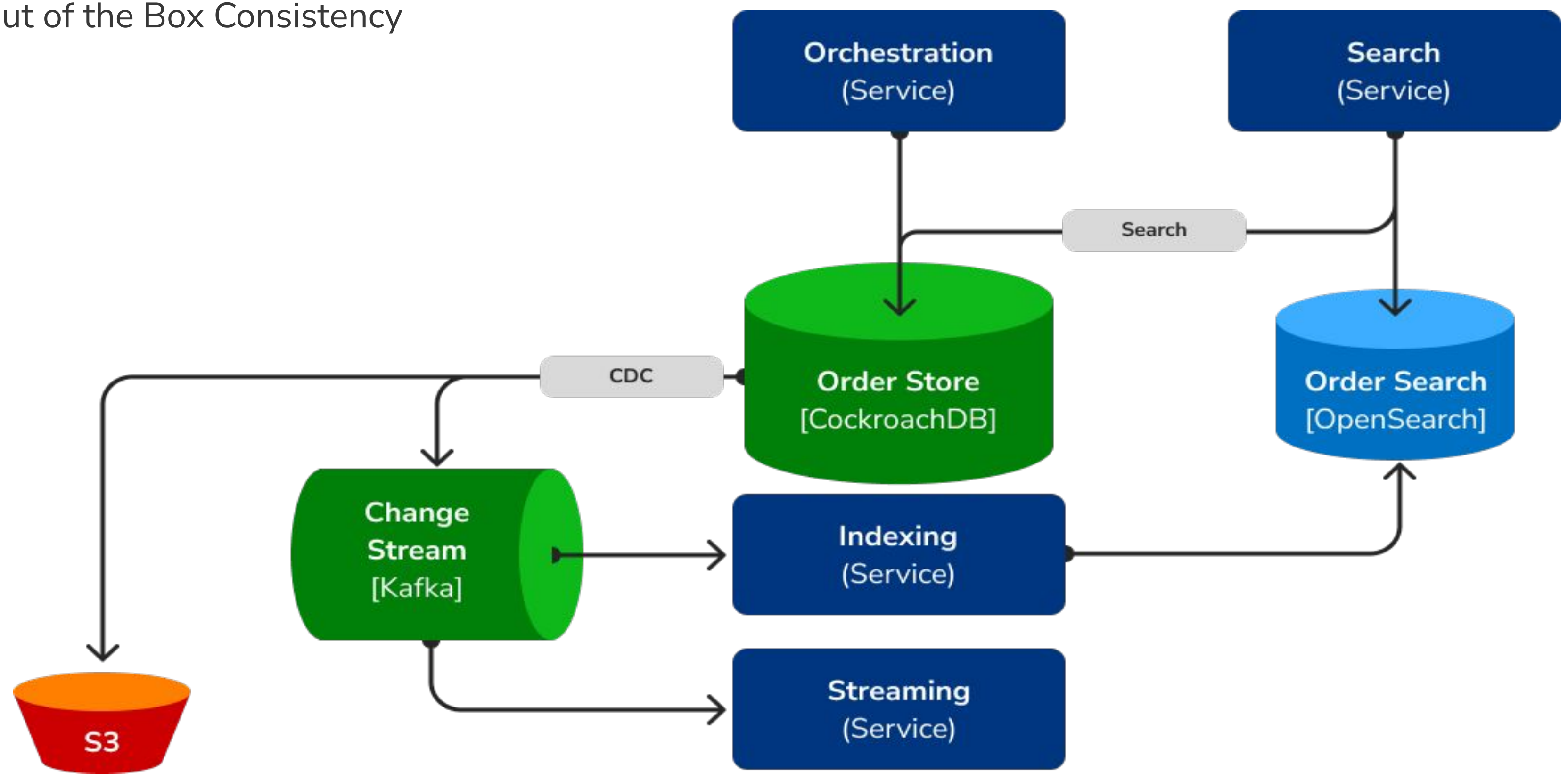
New Architecture

Simpler Application Logic



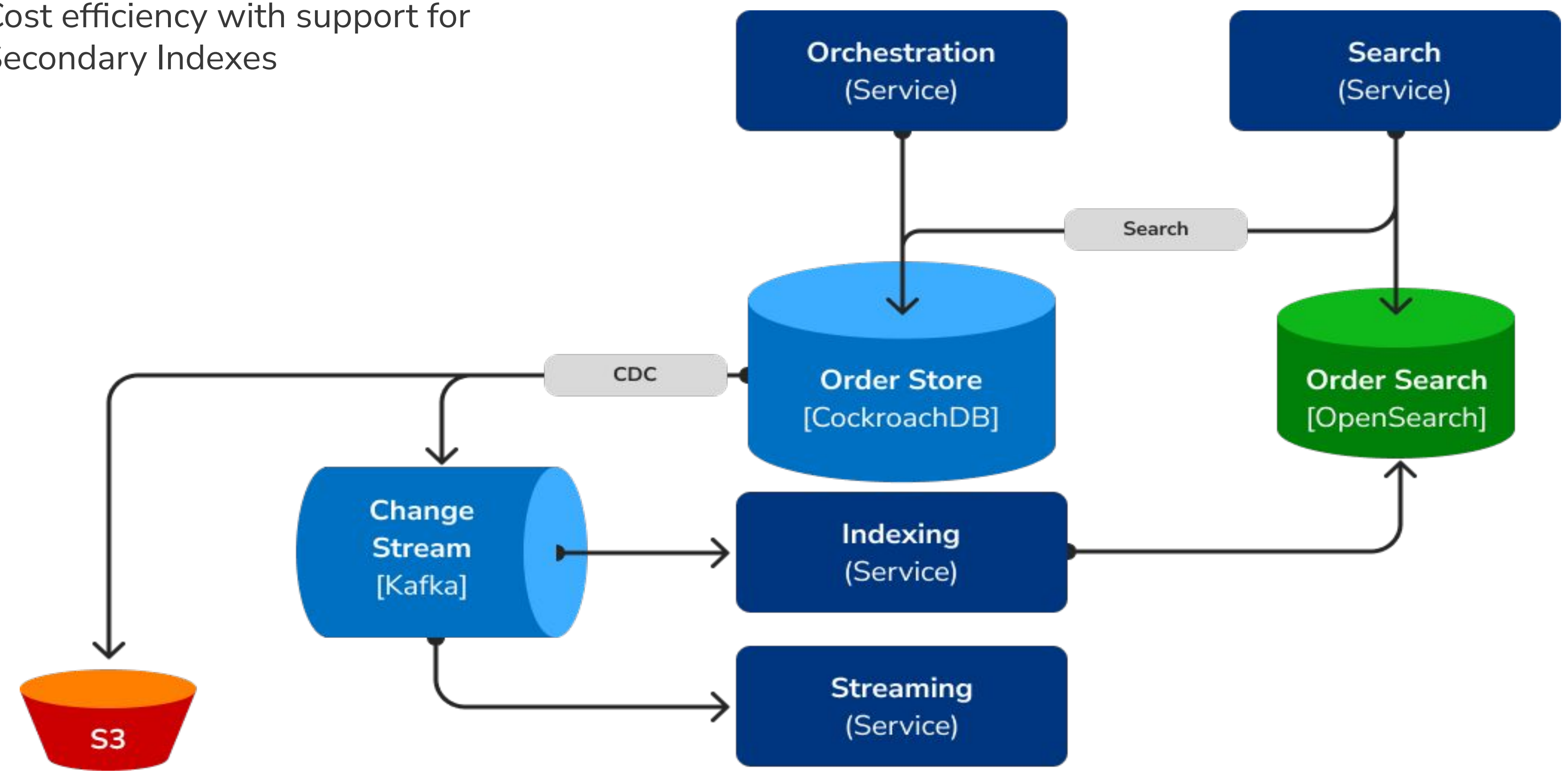
New Architecture

Out of the Box Consistency



New Architecture

Cost efficiency with support for Secondary Indexes



The Hard Truths: It is all about Tradeoffs

Booking.com

Trading A(vailability) for C(onsistency)

Expectations had to be adjusted since the architecture and promises between Cassandra and CockroachDB are not the same.

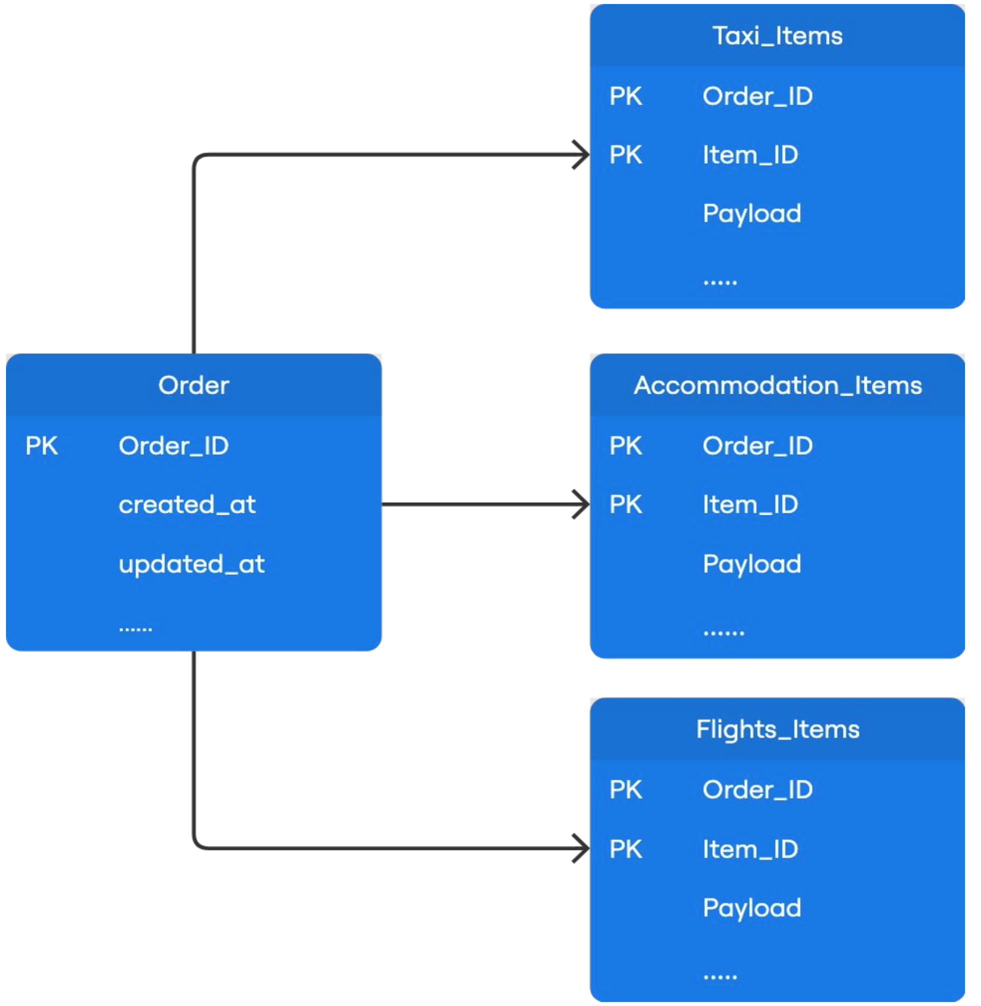
99.99999955% → **99.99998%**

SLI for Cassandra writes

SLI for CockroachDB writes

To Normalize or Not to Normalize?

The SQL interface was tempting to start normalizing our schema at the beginning of the migration rather than migrating with the same Cassandra schema and refactoring later.

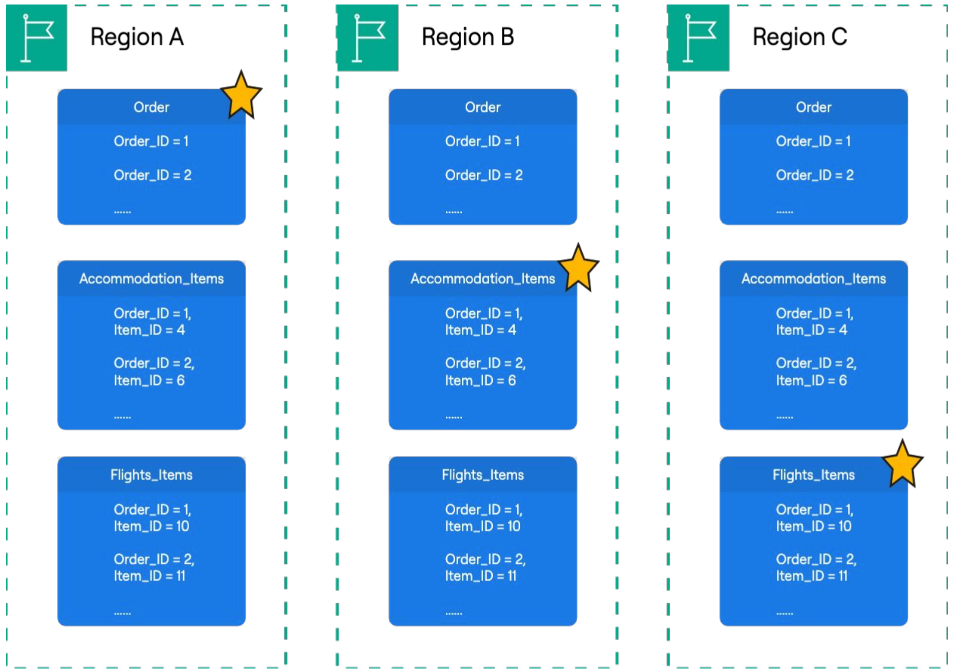


To Normalize or Not to Normalize?

Constructing an order becomes a join across multiple regions.

Queries can be executed in parallel but this means 1 request will result in N queries but the performance is bound to the slowest query.

More queries means more resources needed.



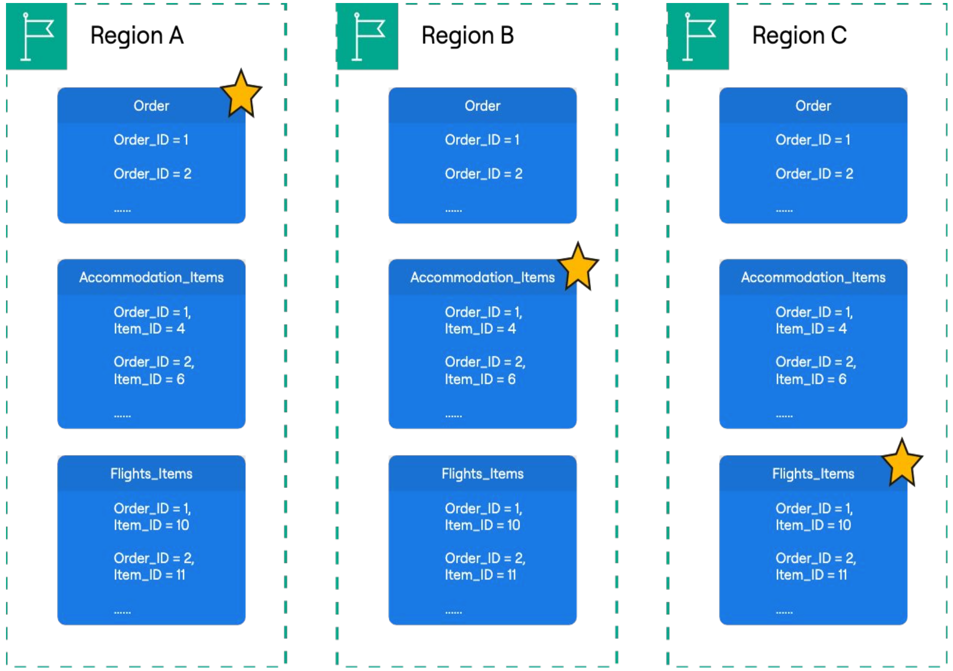
Region A
Order
Order_ID = 1
Order_ID = 2
......
Accommodation_Items
Order_ID = 1, Item_ID = 4
Order_ID = 2, Item_ID = 6
......
Flights_Items
Order_ID = 1, Item_ID = 10
Order_ID = 2, Item_ID = 11
......
Region B
Order
Order_ID = 1
Order_ID = 2
......
Accommodation_Items
Order_ID = 1, Item_ID = 4
Order_ID = 2, Item_ID = 6
......
Flights_Items
Order_ID = 1, Item_ID = 10
Order_ID = 2, Item_ID = 11
......
Region C
Order
Order_ID = 1
Order_ID = 2
......
Accommodation_Items
Order_ID = 1, Item_ID = 4
Order_ID = 2, Item_ID = 6
......
Flights_Items
Order_ID = 1, Item_ID = 10
Order_ID = 2, Item_ID = 11
......

To Normalize or Not to Normalize?

Experimented with partitioning to co-locate the data in the same region but *it didn't show a significant gain*

De-normalization to bring the data showed significant improvements on *latencies*

Continuing to experiment to find the sweet spot between performance and data models cleanliness.



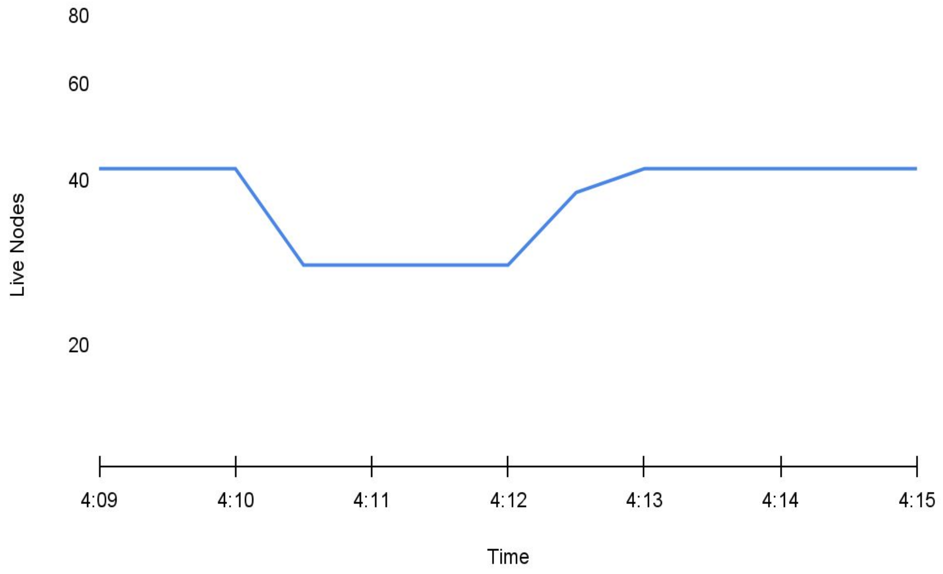
A Wake-up Call

The question is not ‘Will it fail?’ but ‘How will it fail — and when?’

Chaos Engineering

Big Loss of Availability

A big drop in the live nodes count across all regions leading to a severe degradation of services.

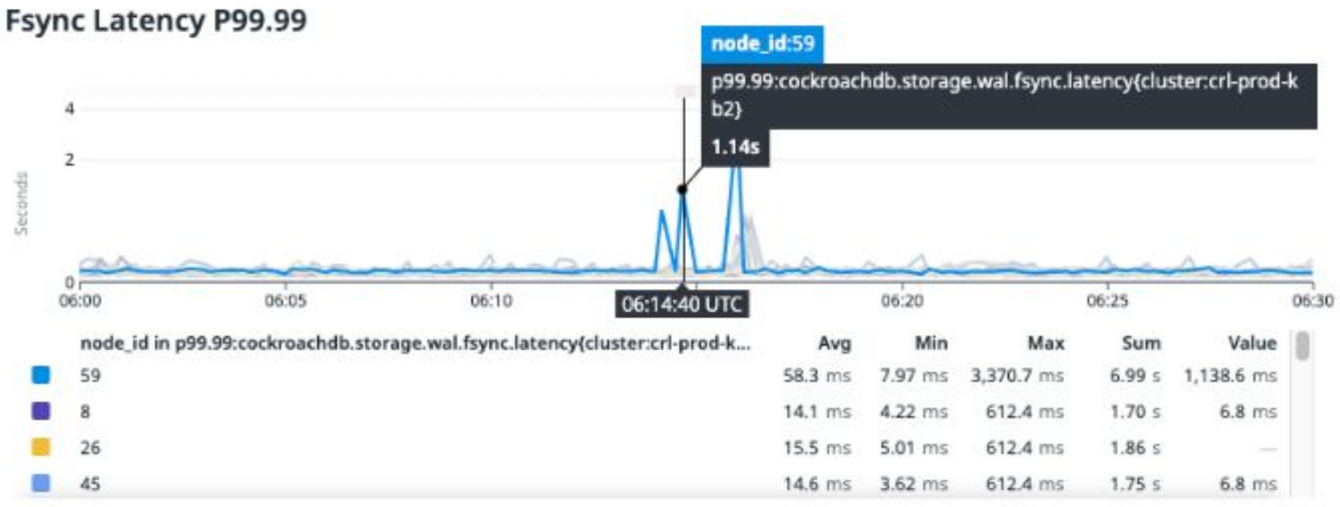


Root Cause

"*One node had a slow disk* but was able to retain the (expiration-based) lease on the node liveness range.
Given that it was able to retain the lease and that there were no network partitions preventing others from contacting it.
This is exactly the kind of single-point-of-failure event that we're working to solve now with the move away from epoch-based leases towards decentralized leader leases. With those changes, no single node's disk slowness will be able to impact the liveness of every other node in the cluster"

Disk Stalls Everywhere

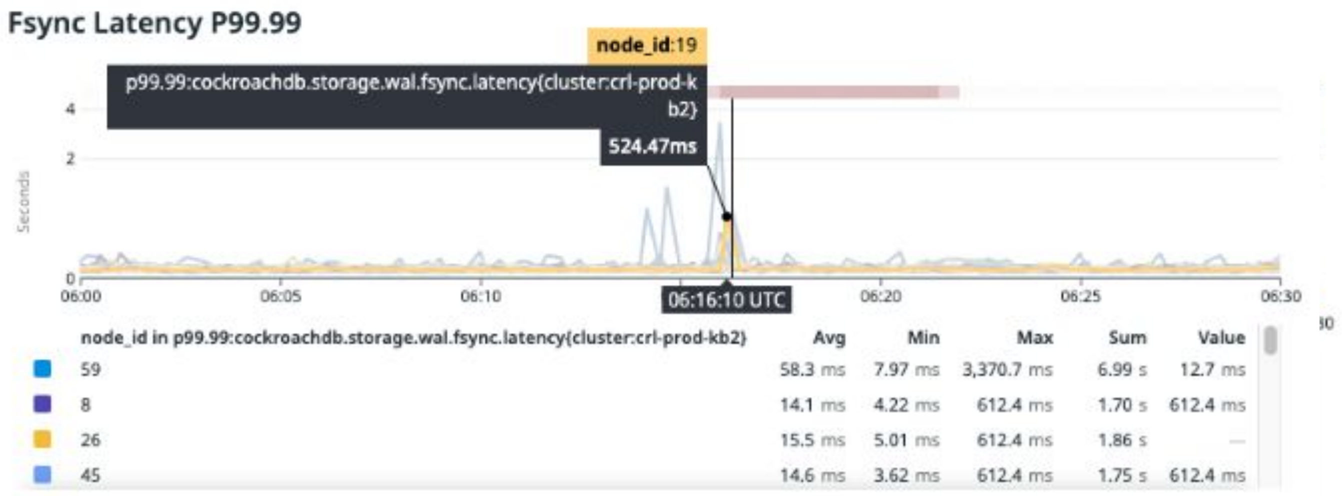
Disk stalls at node 59 when it was holding the liveness range leaseholder



	node_id in p99.99:cockroachdb.storage.wal.fsync.latency{cluster:crl-prod-k...	Avg	Min	Max	Sum	Value
🟦	59	58.3 ms	7.97 ms	3,370.7 ms	6.99 s	1,138.6 ms
🟪	8	14.1 ms	4.22 ms	612.4 ms	1.70 s	6.8 ms
🟨	26	15.5 ms	5.01 ms	612.4 ms	1.86 s	—
🟦	45	14.6 ms	3.62 ms	612.4 ms	1.75 s	6.8 ms

Disk Stalls Everywhere

Node Disk stalls at node 19 right when it became the leaseholder of the liveness range



	node_id in p99.99:cockroachdb.storage.wal.fsync.latency{cluster:crl-prod-kb2}	Avg	Min	Max	Sum	Value
🟦	59	58.3 ms	7.97 ms	3,370.7 ms	6.99 s	12.7 ms
🟪	8	14.1 ms	4.22 ms	612.4 ms	1.70 s	612.4 ms
🟨	26	15.5 ms	5.01 ms	612.4 ms	1.86 s	—
🟦	45	14.6 ms	3.62 ms	612.4 ms	1.75 s	612.4 ms

Remediation & Follow-ups

The incident was resolved by CRL implementing ***WAL failover*** to address transient disk stalls—brief periods when disk operations become unresponsive, often due to IOPS throttling or hardware issues. These stalls can significantly impact write latency, especially in cloud environments like AWS, Azure, and GCP.

Improve incident response
Identified gaps in our incident response procedures.

Fix comms with CRL
The issue kept recurring for months. However, the business impact wasn't made clear enough to CRL. After it was made clear, CRL escalated and provided an interim fix which is the WAL failover.

Run joint drills
Run drills with CRL to simulate failures of CRDB to test our systems' response and the incident response procedure.

Final Words

Next Challenges

Scale to > 20B Reads/Day

Close a yet another massive migration project that would lead to more than 20B Reads/Day

Achieve economies of scale

Our focus is to increase the API requests/vCPU

Mature adoption of CRDB

As more teams start to use CRDB we will have to find more effective ways of collaboration internally and with CRL

Key Takeaways

Understanding database complexities prevents future complications and costs

Databases are quite complicated and whenever there is a gap between what the database can offer and the requirements for the system then the compensation for that will be complicated as well. This added complexity usually manifests as some **critical logic** that has to be well maintained or **additional systems** that costs a lot of money and effort to keep running.

Embrace Tradeoffs

Database systems each have unique designs and cannot be simply substituted for one another.

Anything will fail so be always prepared

Resilience isn't about never falling — it's about getting back up fast, smarter, and with alerts that make sense

❝ Every outage is a dress rehearsal for the next one. Learn your lines. ❞
-Charity Majors

Thank you!

Mahmoud Nagib
mahmoud.nagib@booking.com

Booking.com