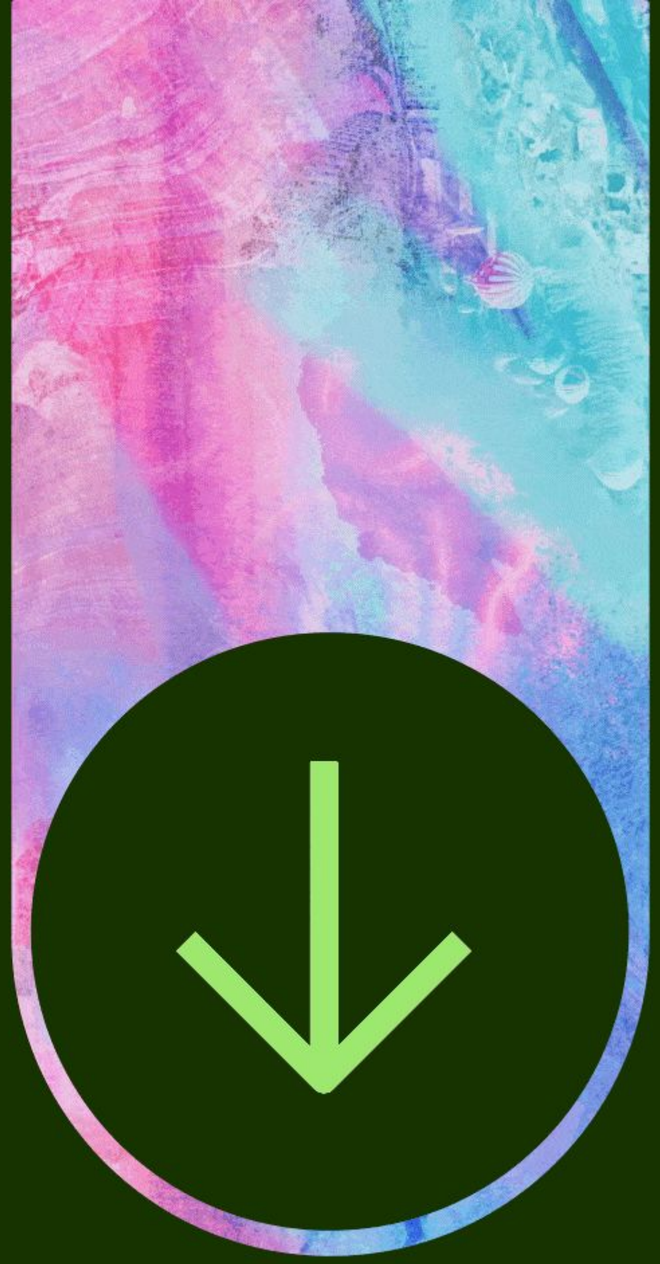
From 60 Million to 25 Billion. 8 Years of Scaling a Core System with PostgreSQL in Fintech



18 March 2025 / Nordic PGDay 2025

Dmytro Hnatiuk
Principal Software Engineer @ Wise

Wise in numbers:

12.8m

active customers worldwide

£118.5bn

across borders in the last financial year

6000+

people work at Wise

SCALING JOURNEY

Role of Finance database

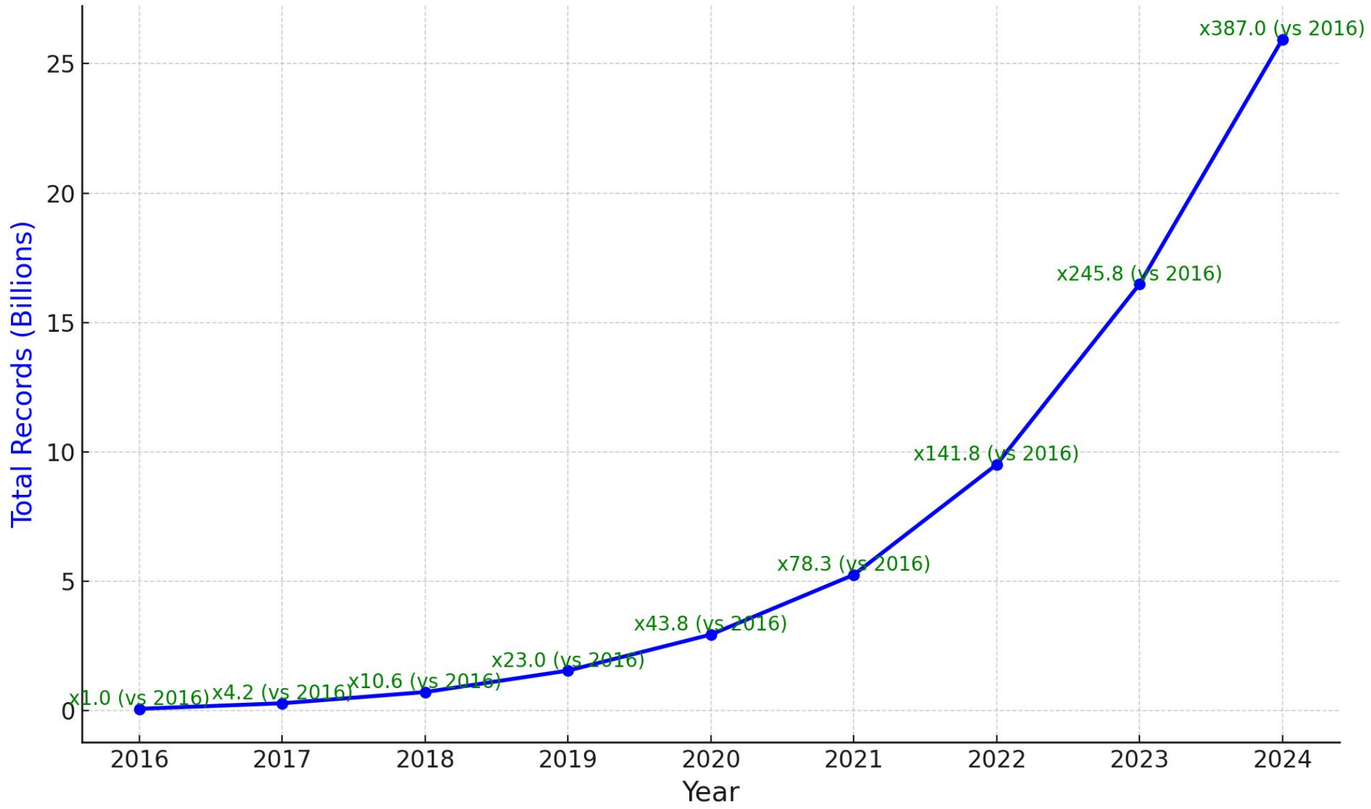
- **The Golden Source**: Serves as the single source of truth for external regulatory, market, and financial reporting.
- **Operational Backbone**: Powers internal operational controls, reconciliations, and ensures smooth financial workflows.
- **Critical for Safeguarding**: Used as the primary data source for safeguarding customer funds and ensuring compliance with financial regulations.

This means...

Data must be **always available, accurate, and reliable**.
Any compromise here risks financial stability and trust.

Scale numbers

Database growth



25B

Rows in a main table

37Tb

Total size

Scale numbers

Data volumes

~ 80k

Processed events per minute

~ 20k

Database transactions per second

> 100

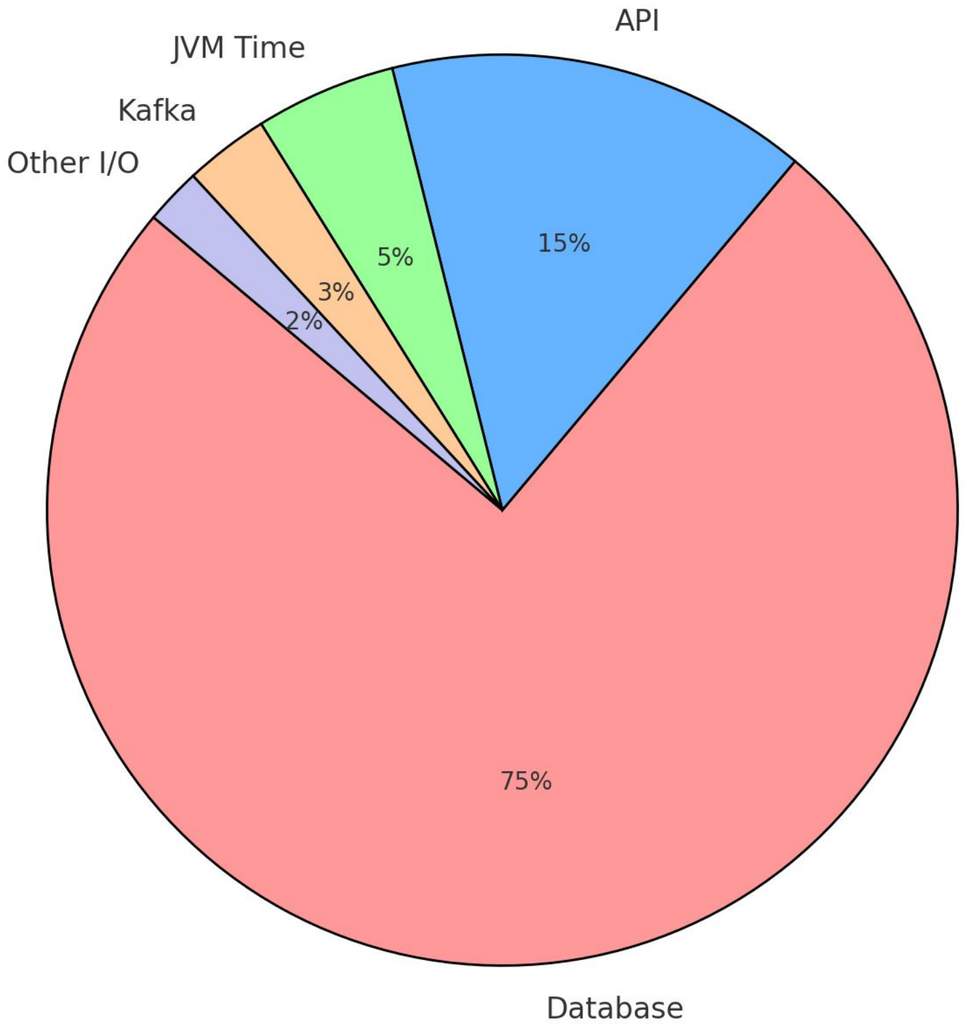
Kafka topics consumed by application

I will focus on relational databases.

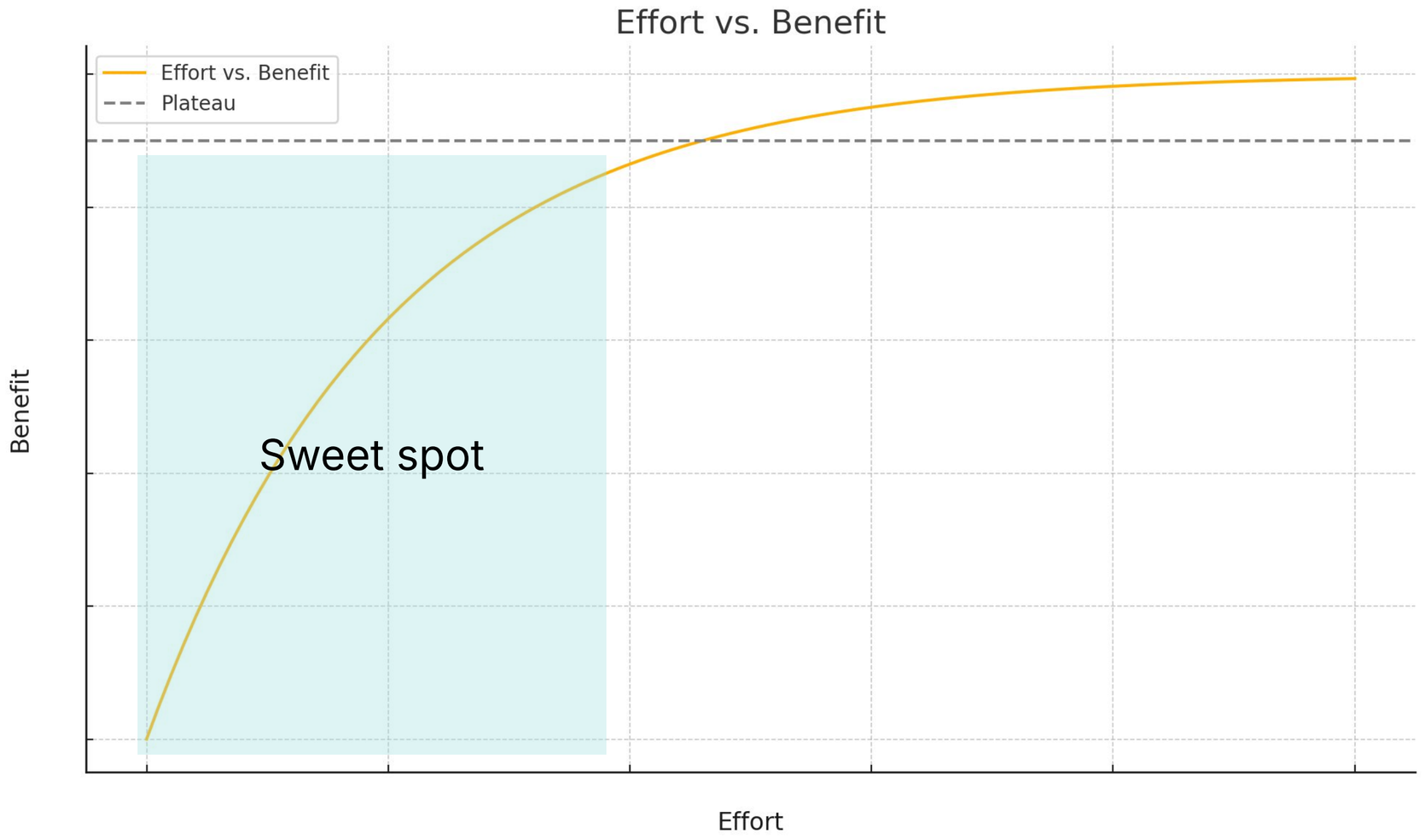
We can scale services horizontally with ease these days (adding more instances, etc).

Relational databases don't scale the same way.

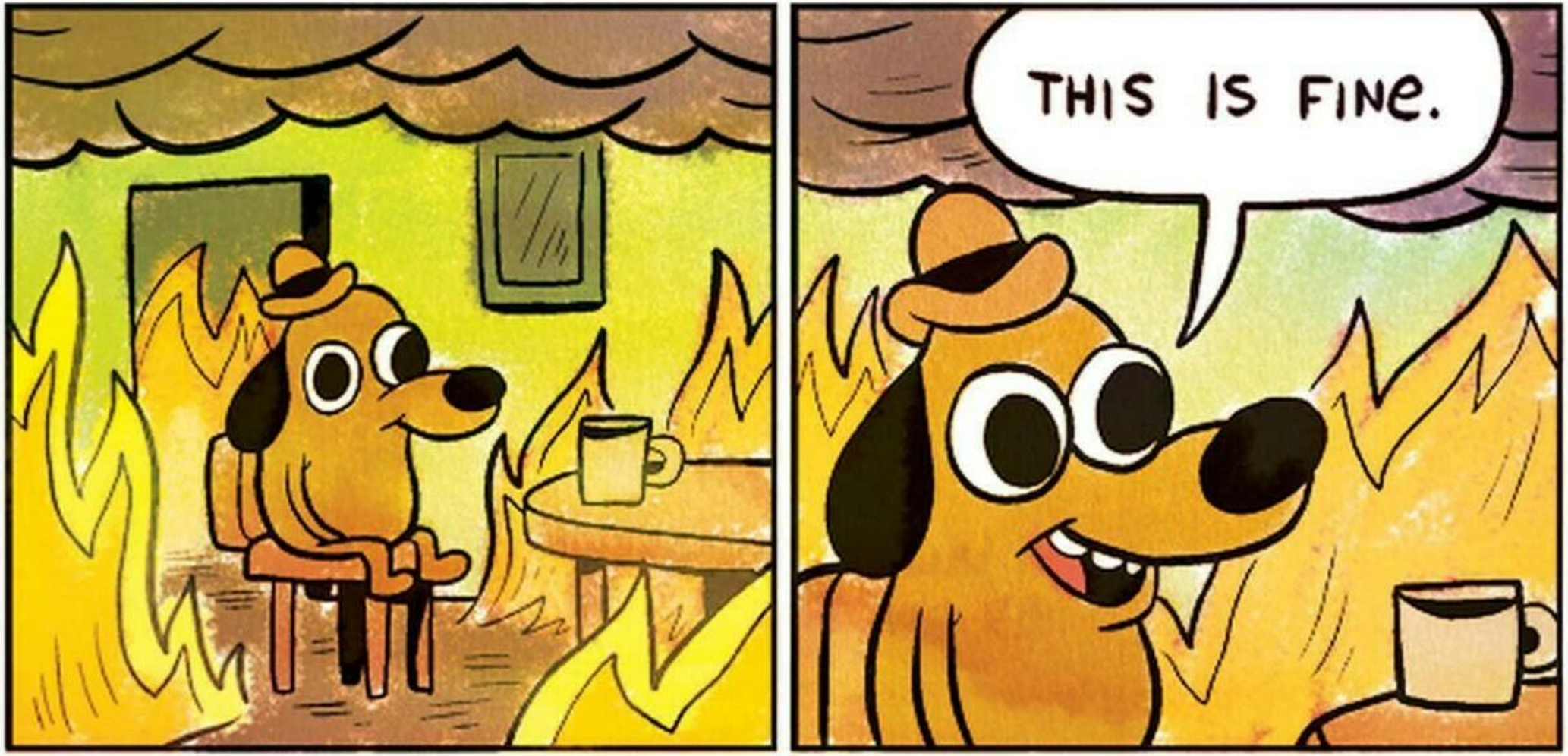
Database will be your bottleneck



Small efforts = Big gains



Why my application is slow?



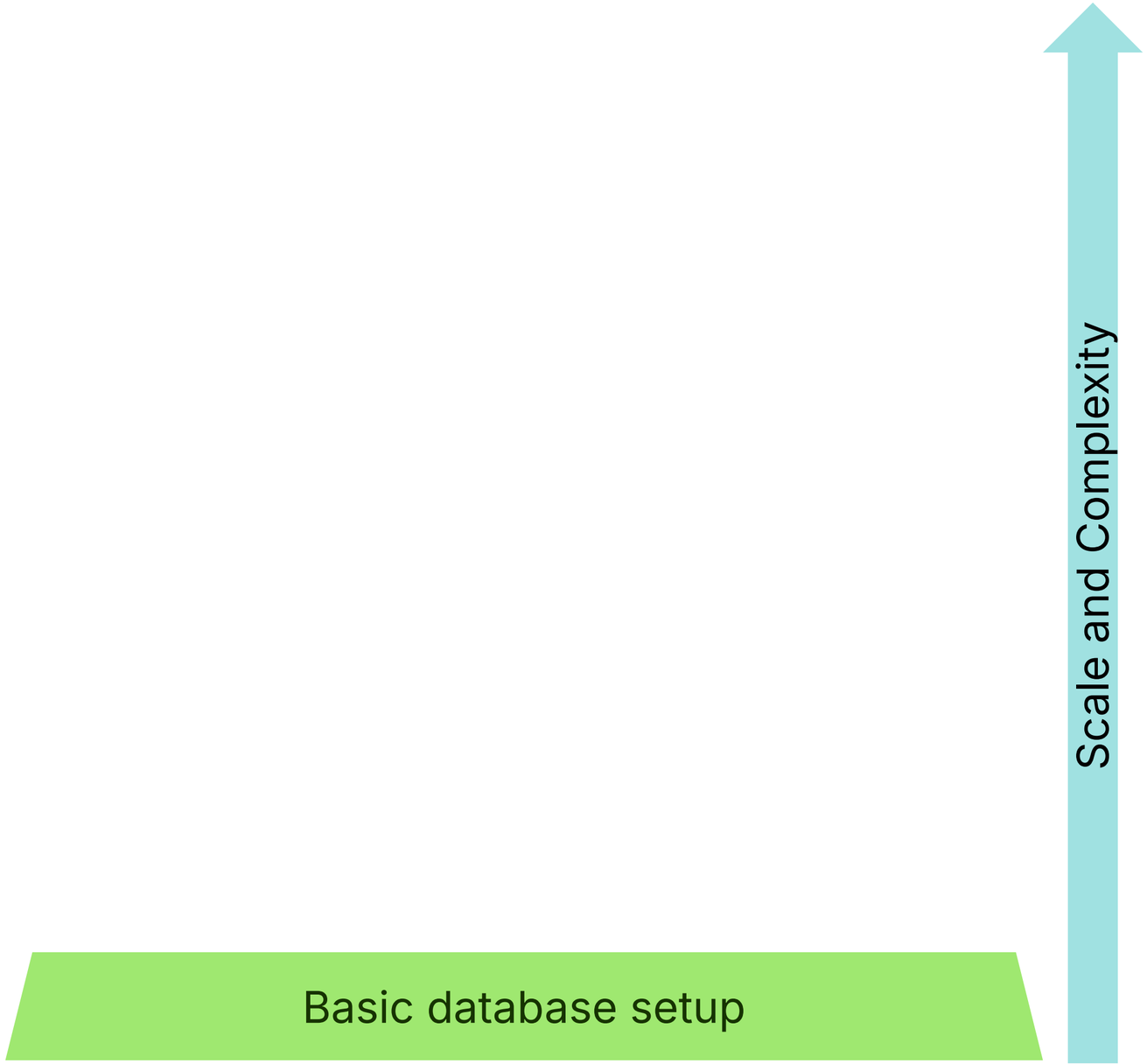
PYRAMID OF DATABASE SCALABILITY ©

The pyramid.

Basic database setup

What It Involves: Adjust basic parameters like memory allocation, disk I/O, buffer sizes, and query cache settings. These are the "quick wins" in database performance optimization.

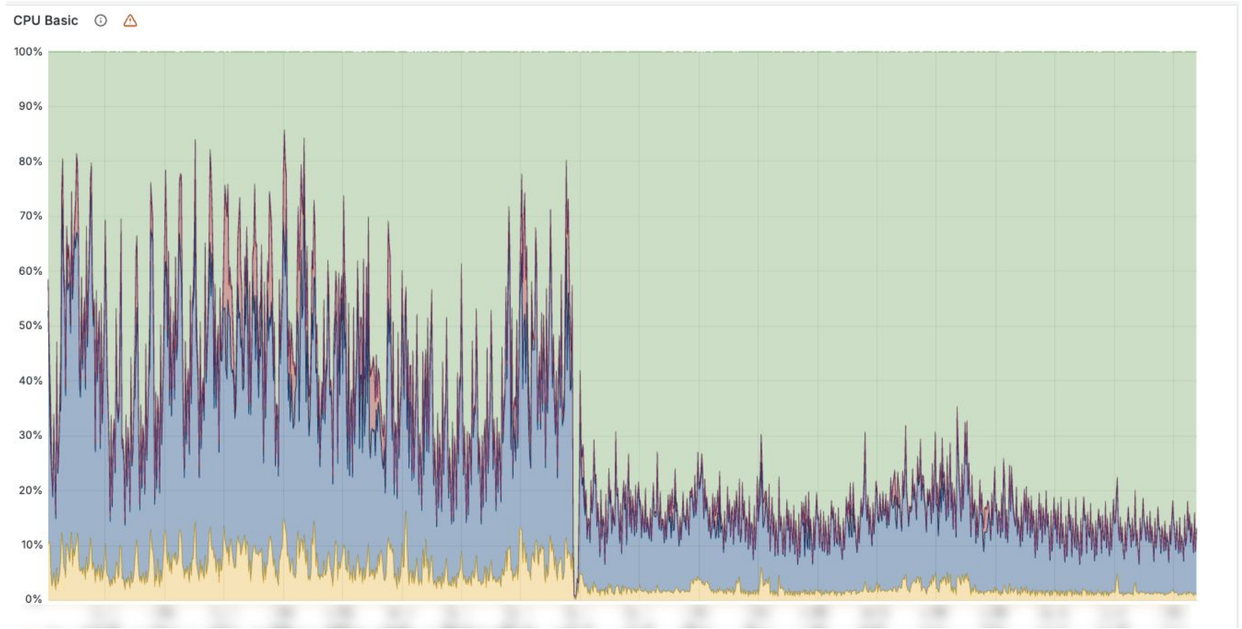
Why It Matters: This foundational layer ensures that the database is functioning efficiently out-of-the-box.

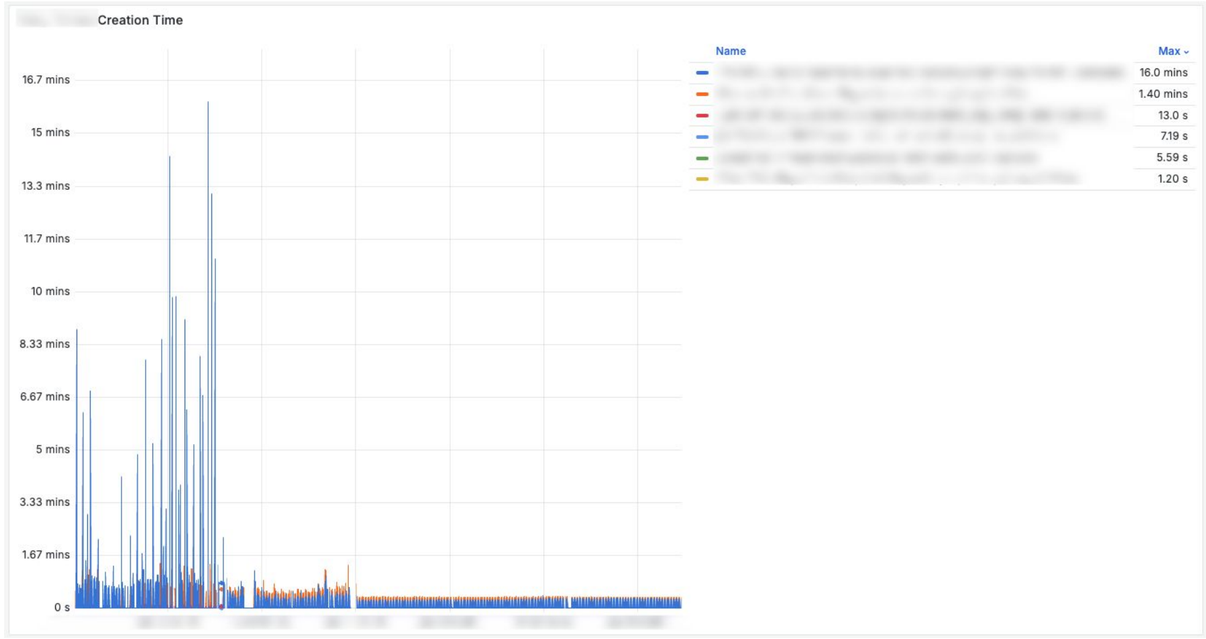


The pyramid. Real world example.

Basic database setup

Problem: Scheduled processes experienced performance spikes, increased execution time and unstable execution durations.





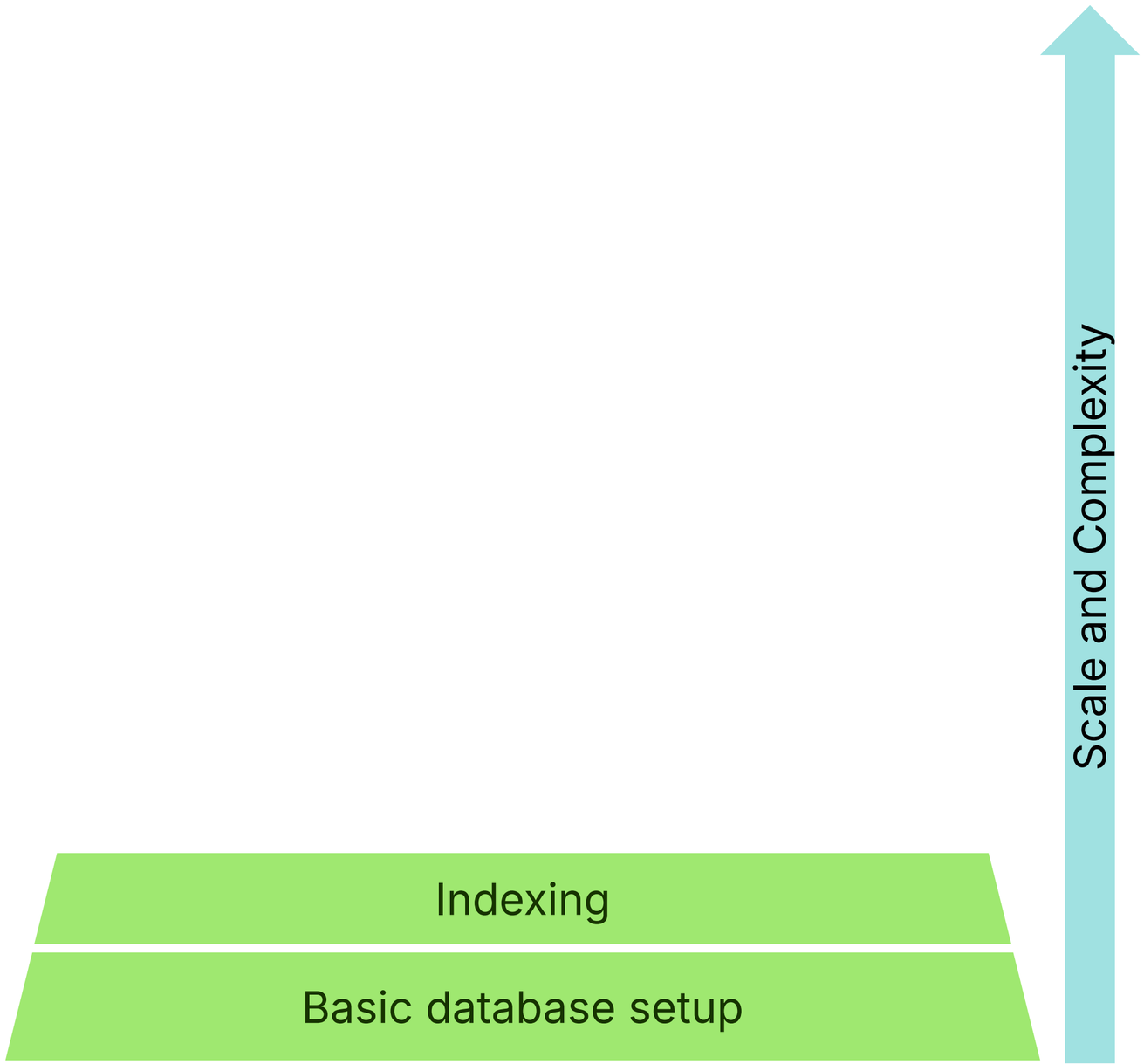
Solution: Database was vertically scaled at relatively low cost of **$540/month for 3 instances**. Performance drastically improved and processes start running stable.
AWS EC2 ***r5.2xlarge*** *to* ***r5.4xlarge*****.**

The pyramid.

Indexing

What It Involves: Adding indexes, limiting full table scans, and reviewing slow-running queries

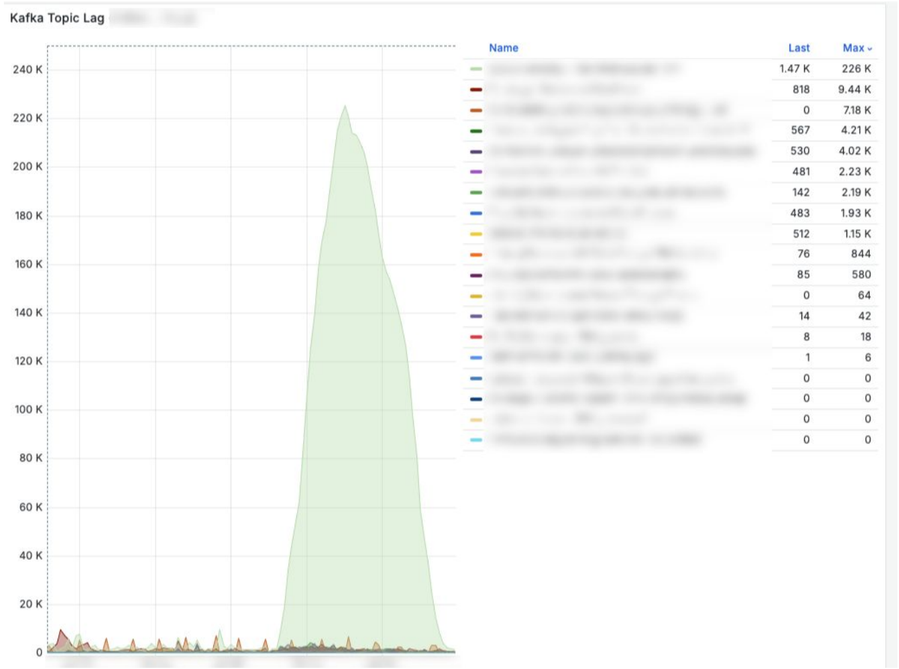
Why It Matters: Indexes can dramatically reduce query time, often with minimal effort and little need for database deep dives.



The pyramid. Real world example.

Indexing

Problem: After release of new query type, latency of processes degraded significantly. Backlogs start to grow. Manual execution of **query plan confirm all indexes in place.**




```
Append  (cost=0.12..22485439.59 rows=21886841 width=824)
(actual time=0.047..0.047 rows=0 loops=1)
…
  ->  Index Scan using idx___id_closed on _____
(cost= rows=7984054 width=827) (actual time=0.036..0.036
rows=0 loops=1)
        Index Cond: (____id = 1278931734)
        Buffers: shared hit=5
```

```
select * from ____

where ____._____id::text='1278931734'
```

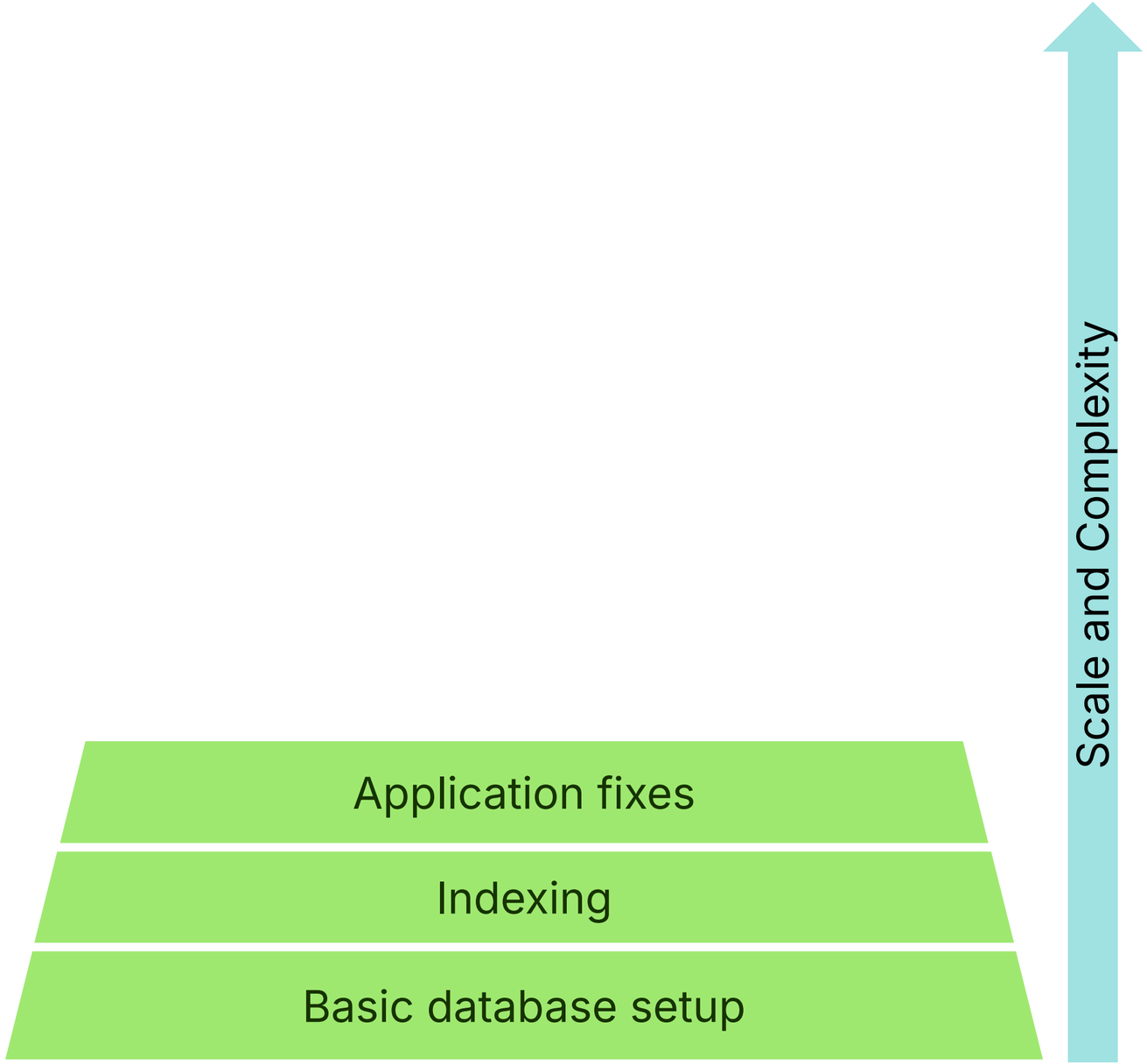
Solution: We had a **wrong index.** ORM (hibernate) was doing type conversion to **text**, instead of **long,** causing database planner to fallback to **full table scan**.

The pyramid.

Application Fixes

What It Involves: Optimizing the application logic to reduce the number of database calls. Implementing caching strategies to prevent repeated database hits for the same data, as well as refactoring inefficient queries and ensuring the application interacts with the database in a streamlined way.

Why It Matters: A well-optimized application can handle more load without placing undue stress on the database, postponing the need for more complex database optimizations.

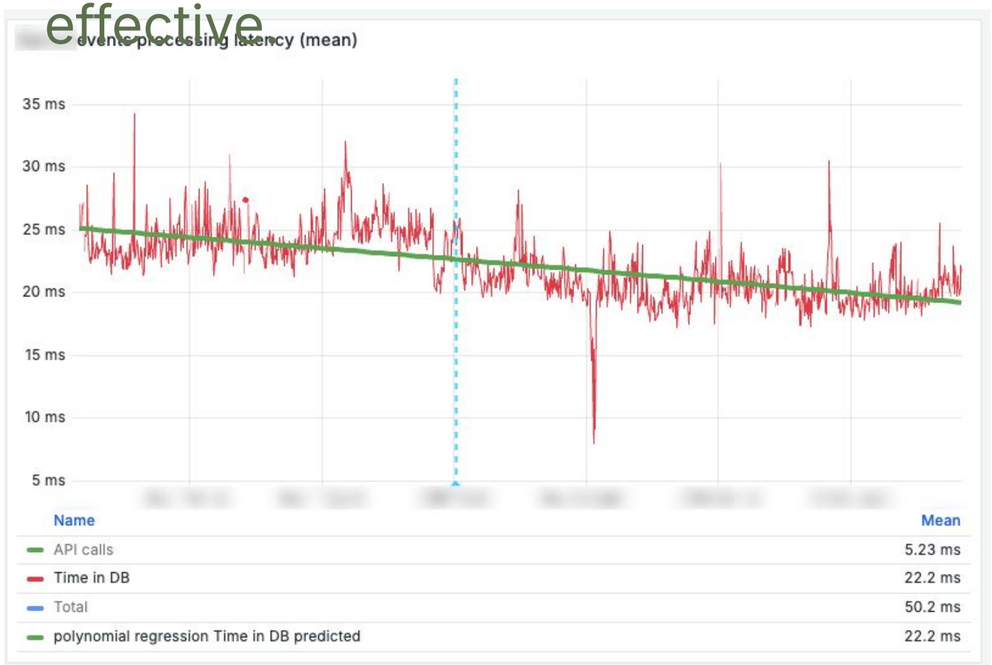


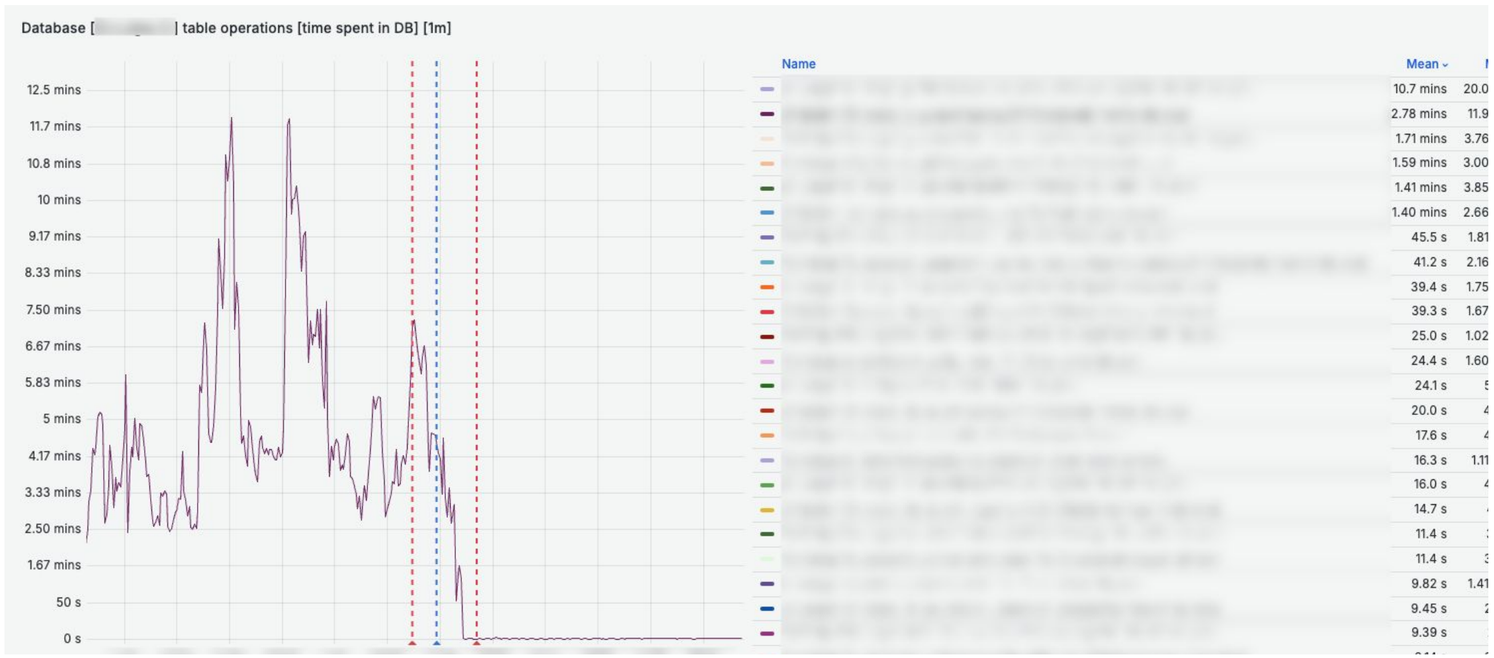
Isn’t it a talk about databases?

The pyramid. Real world example.

Application Fixes

Problem: Latency of events processing was higher than expected. After investigating we realized that some tables queried more than they should. Caching was not effective.



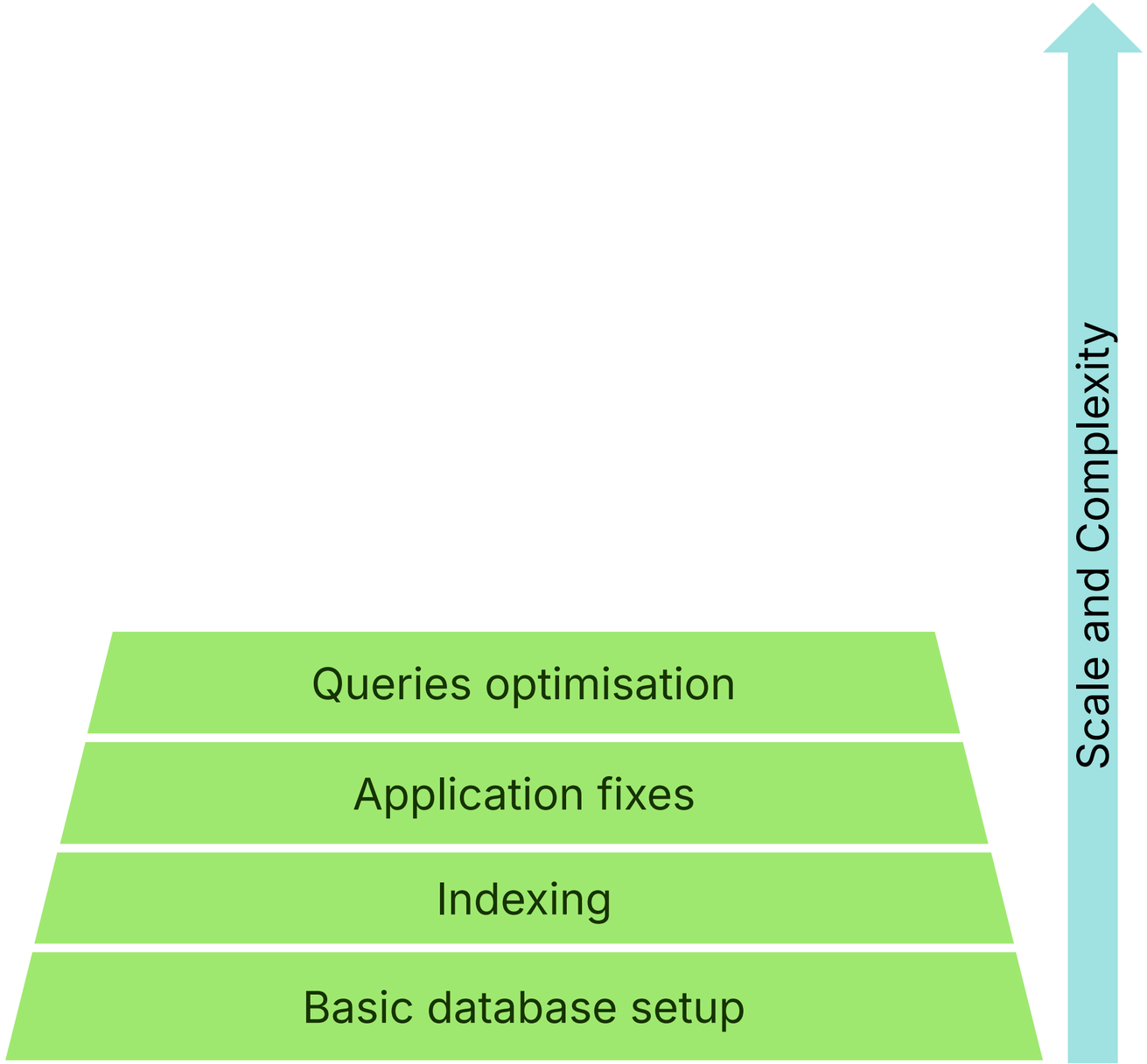


Solution: We had to rewrite number of queries to use application-level cache. In resulted in ~4ms **(15%) latency improvement**.

The pyramid.

Queries optimisation

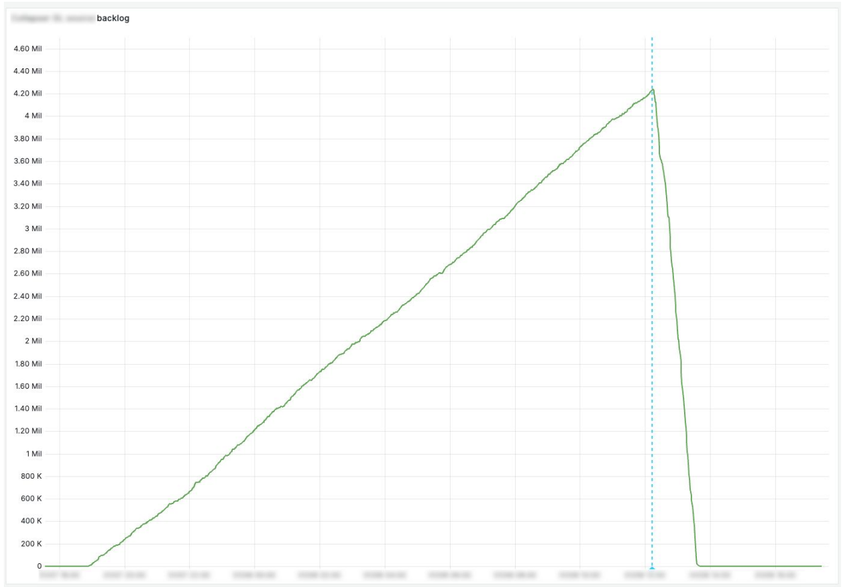
What It Involves: Tuning SQL queries to improve performance, such as optimizing JOIN operations, reducing subqueries, and using appropriate data types. Leveraging query analysis tools to identify bottlenecks.
Why It Matters: Poorly written queries can bottleneck performance. Query optimization ensures you’re getting the most out of your database without overburdening it.



The pyramid. Real world example.

Queries optimisation

Problem: Batch process had a complex SQL composed from 5 large sub-queries. At one day query performance significantly degraded.



FROM

```
1  WITH sales AS (SELECT *
2          FROM transactions),
3     churn AS (SELECT *
4               FROM sales),
5     arpu AS (SELECT *
6              FROM sales s
7                   JOIN churn c
8           ON s.customer_id = c.customer_id),
9     final_data AS (SELECT *
10             FROM ARPUData
11             WHERE arpu > 100)
12 SELECT *
13 FROM final_data;
14
```

TO

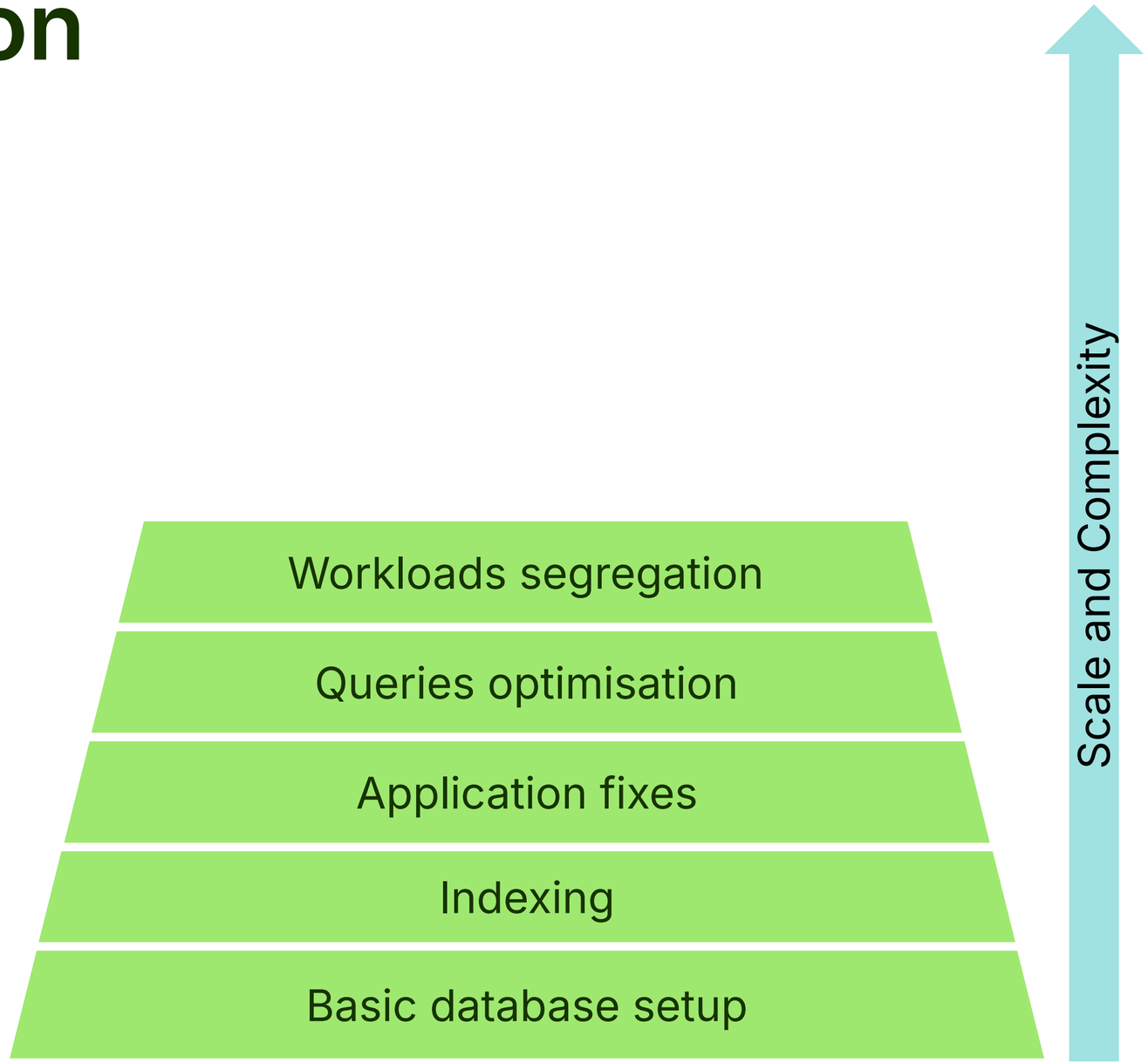
```
1  create temporary table sales
2     as (select * from transactions);
3  analyze sales;
4  create temporary table churn
5     as (select * from sales);
6  analyze churn;
7  create temporary table arpu
8     as (select * from sales s
9        join churn c on s.customer_id = c.customer_id);
10 analyze arpu;
11 create temporary table final_data
12    as (select * from arpu where arpu > 100);
13 analyze final_data;
14 select * from final_data;
```

Solution: We broke down **complex, 5-queries CTE** into **smaller queries** to help database in choosing a right query plan.
P.S. Avoid batch processes!

The pyramid.

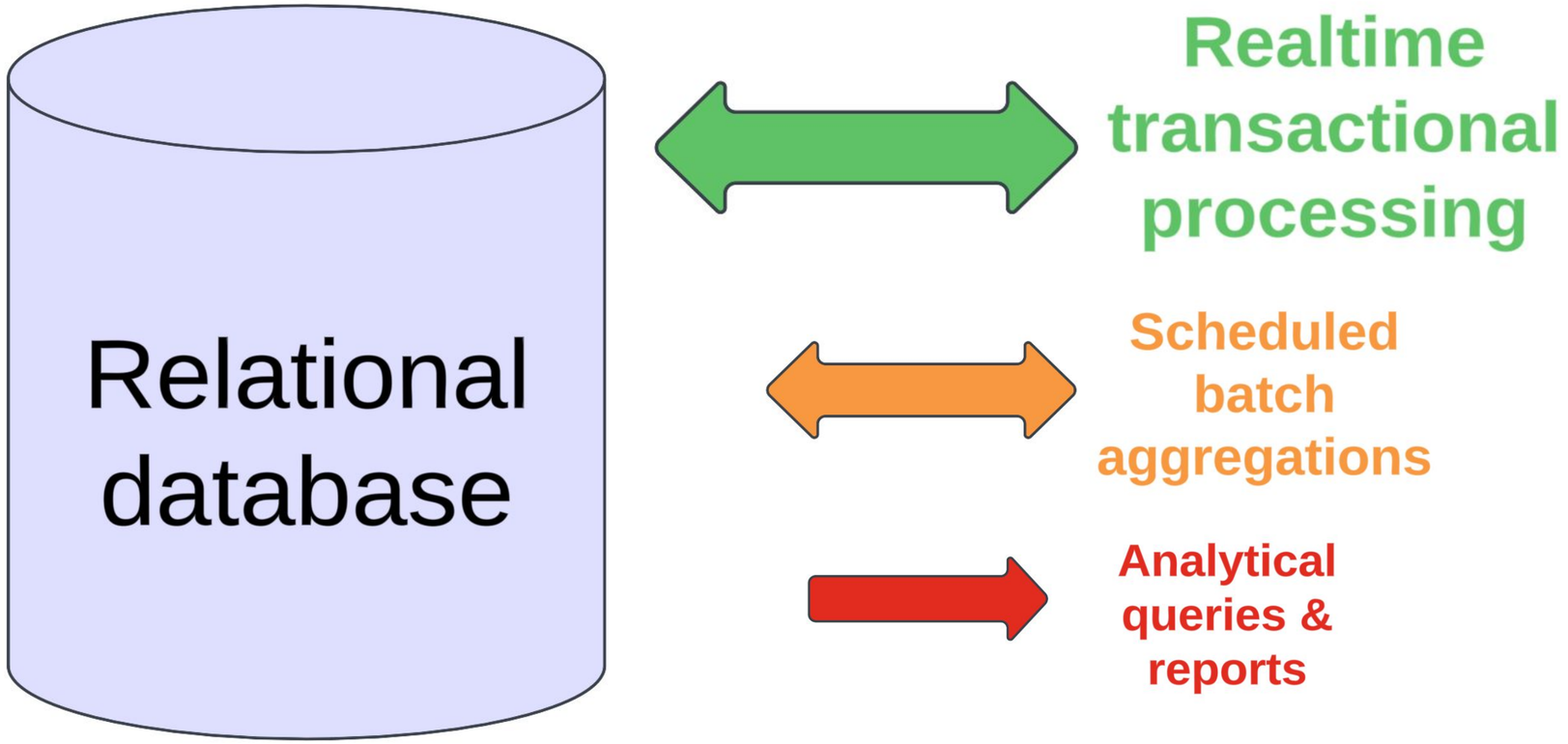
Workloads segregation

What It Involves: Splitting read and write workloads across different systems or using read replicas for read-heavy queries. You should start thinking about moving some of your heavy aggregations workloads to Data Warehouse or Data Lake.
Why It Matters: By segregating workloads, you ensure that one set of operations doesn’t affect the performance of another. For instance, long-running analytical queries won’t slow down your transactional database.



The pyramid. Real world example.

Workloads segregation



Problem: Mixing workload profiles means database cannot be optimised of neither. Long running **batch processes** often **negatively impact real time processing latency.**

The pyramid. Real world example.

Workloads segregation

Rapid business growth.

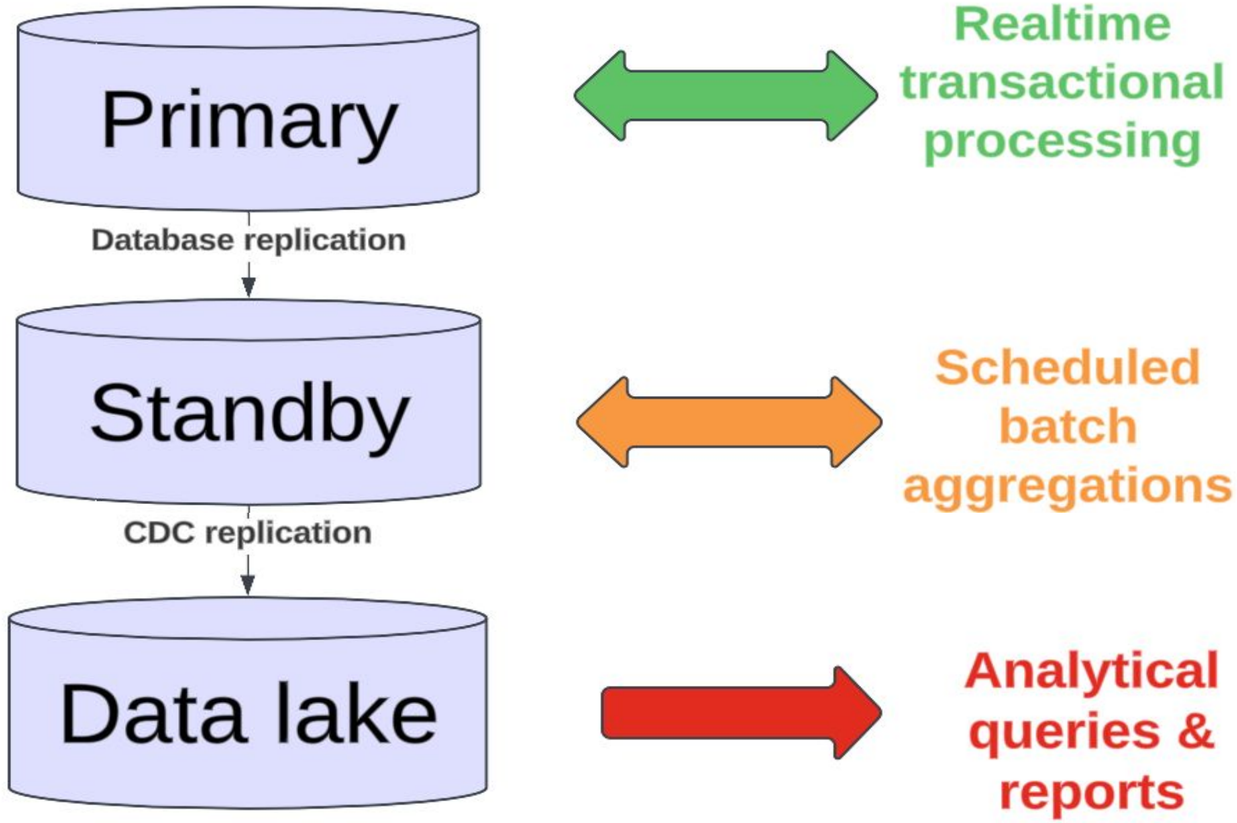
=

Pressure for a database.

The pyramid. Real world example.

Workloads segregation

Solution: Workload types segregation to different environments.

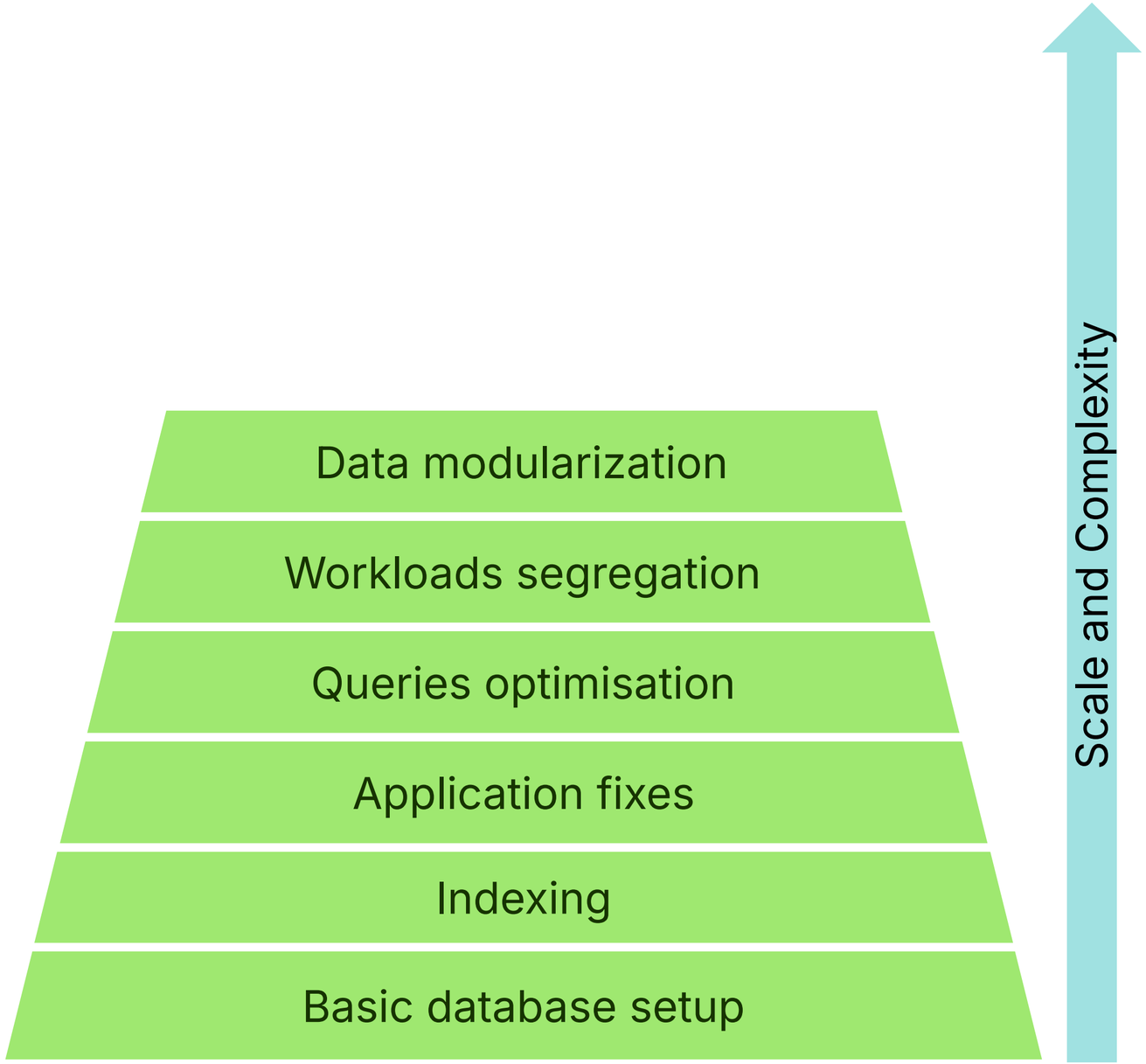


The pyramid.

Data modularization

What It Involves: Breaking the database into logical modules that correspond to different areas of the business (e.g., user data, transaction data, logs). This should involve splitting a monolithic database into smaller, modular databases for different functionalities.

Why It Matters: Modularizing your database reduces complexity and makes it easier to maintain and scale individual components. Each module can be tuned and scaled according to its specific needs, which enhances overall performance.



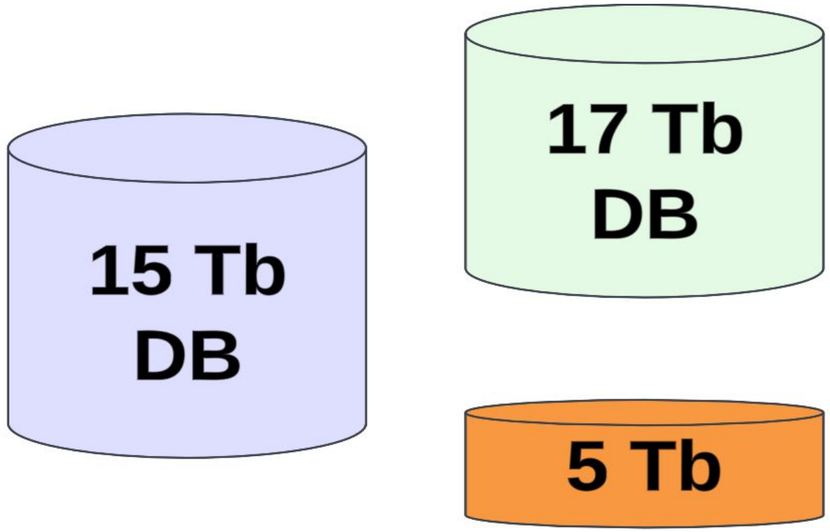
The pyramid. Real world example.

Data modularization

Problem: Single multi-tenant database was serving multiple usage profiles under one roof, leading to **performance bottlenecks, maintenance complexity and Resource contention.**

From...

To...

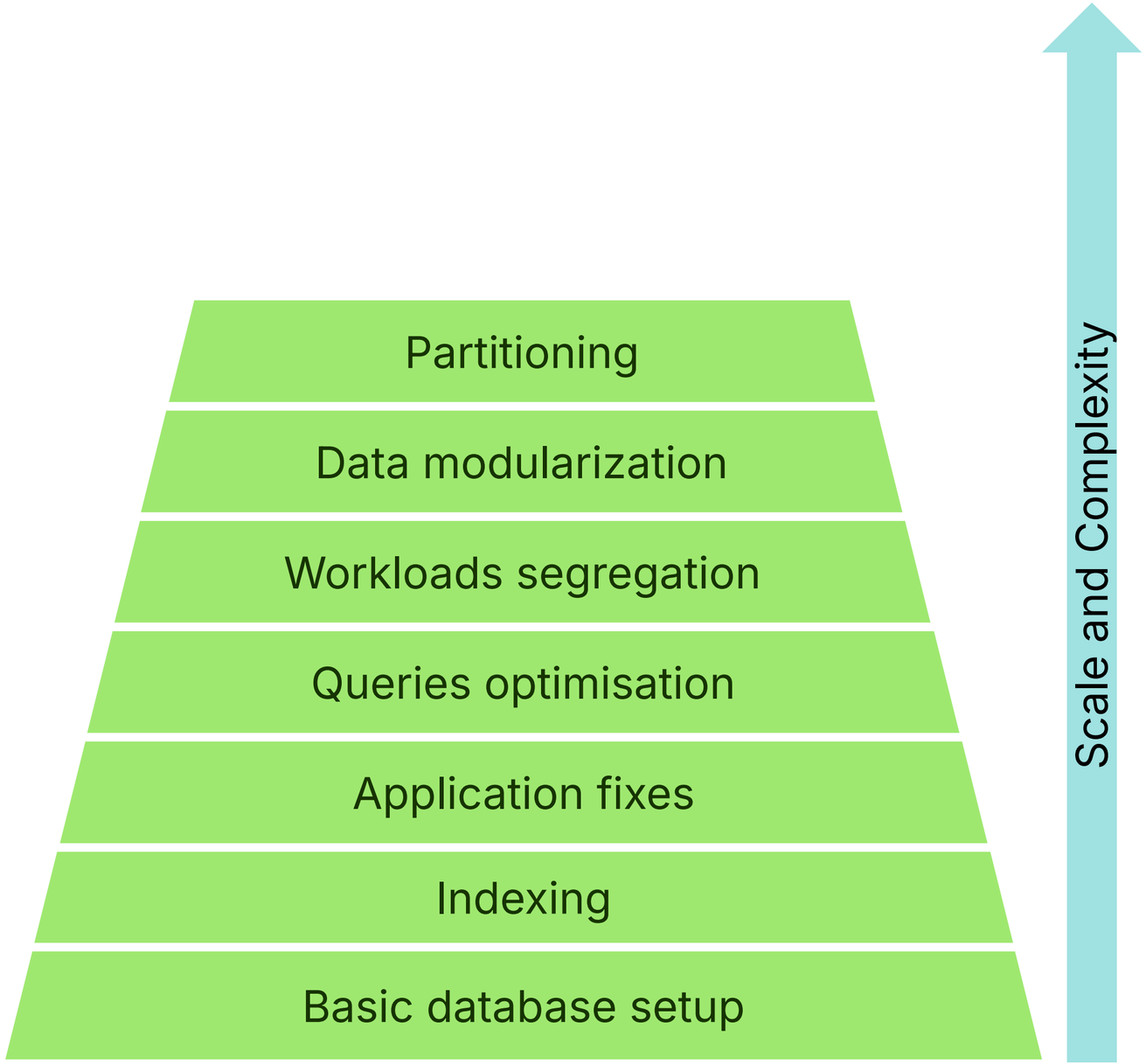


Solution: We split a single monolith to three smaller in size databases.

The pyramid.

Partitioning

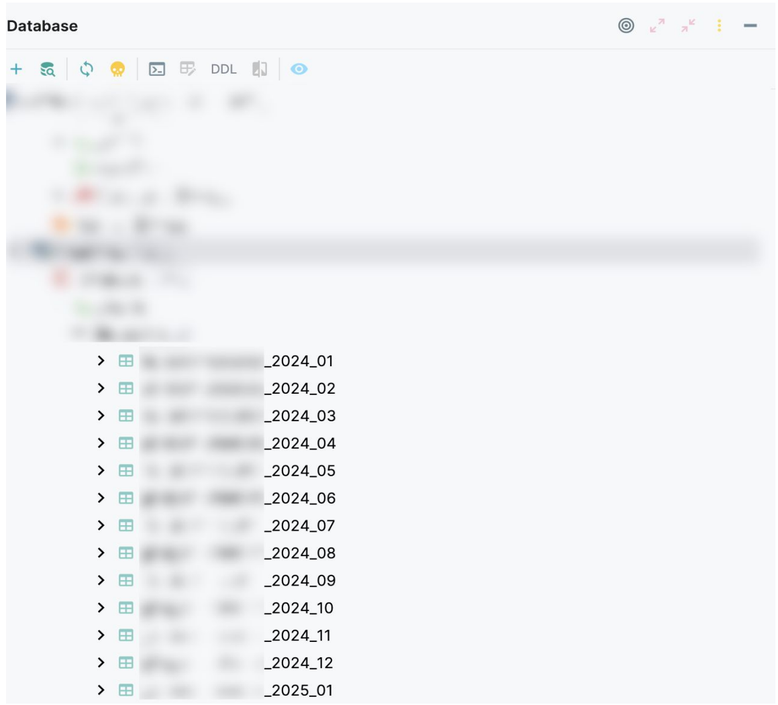
What It Involves: Splitting large tables into chunks (often by time) to reduce the amount of data processed in queries.
Why It Matters: Partitioning reduces the amount of data scanned during queries, making them faster. This would also guide you into a future archival - forcing to think about you data access patterns.



The pyramid. Real world example.

Partitioning

Problem: Performance of large tables started degrading.



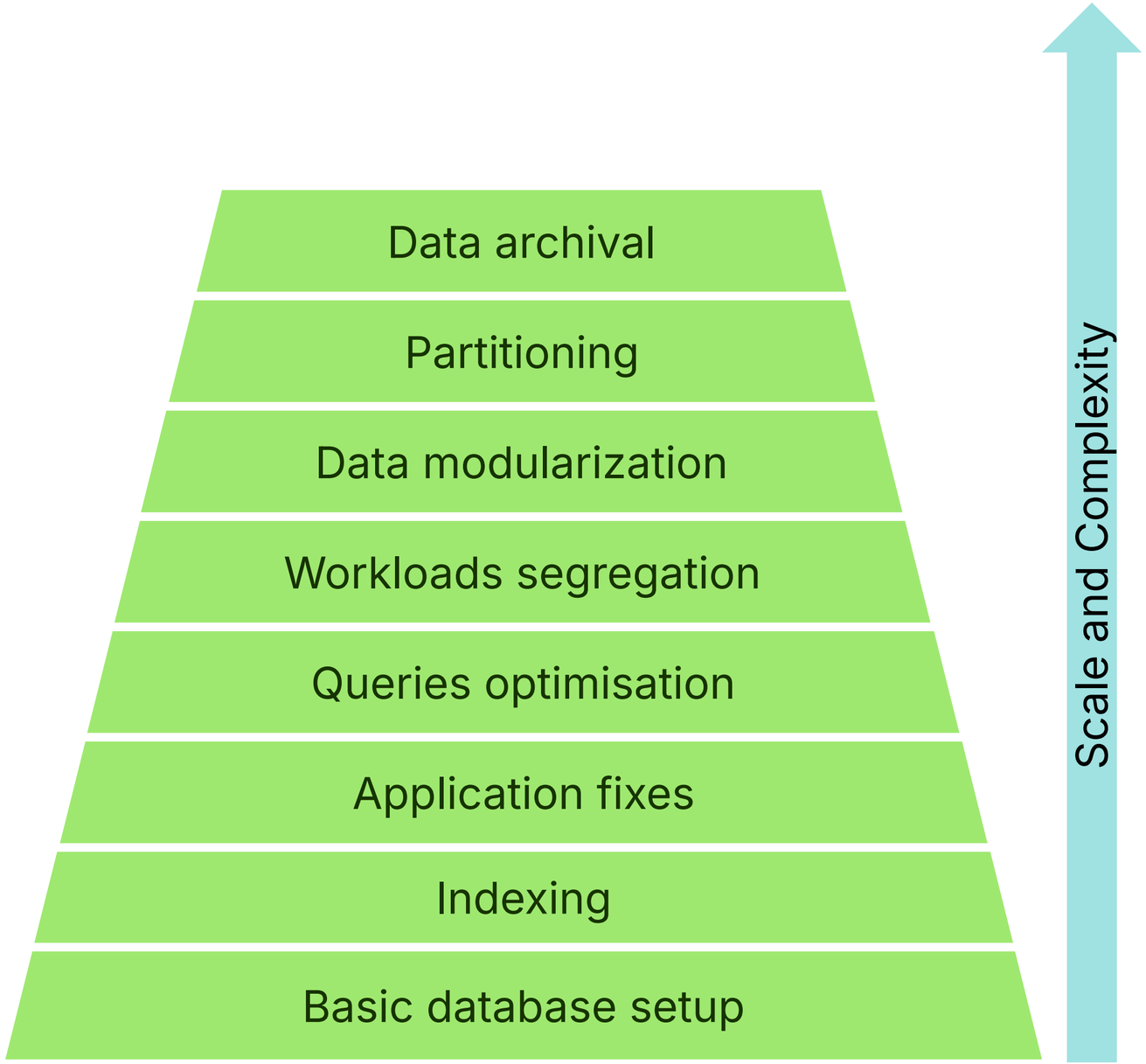
Solution: We partitioned large tables by **monthly chunks**, easing job for database management tasks and optimise data access. Duration of **maintenance tasks reduced more than 10 times.**

The pyramid.

Data archival

What It Involves: Moving outdated or rarely accessed data to cold storage solutions or secondary databases to reduce the load on your primary database. By regularly archiving old transactional or log data you will be maintaining only a lean and efficient active dataset.

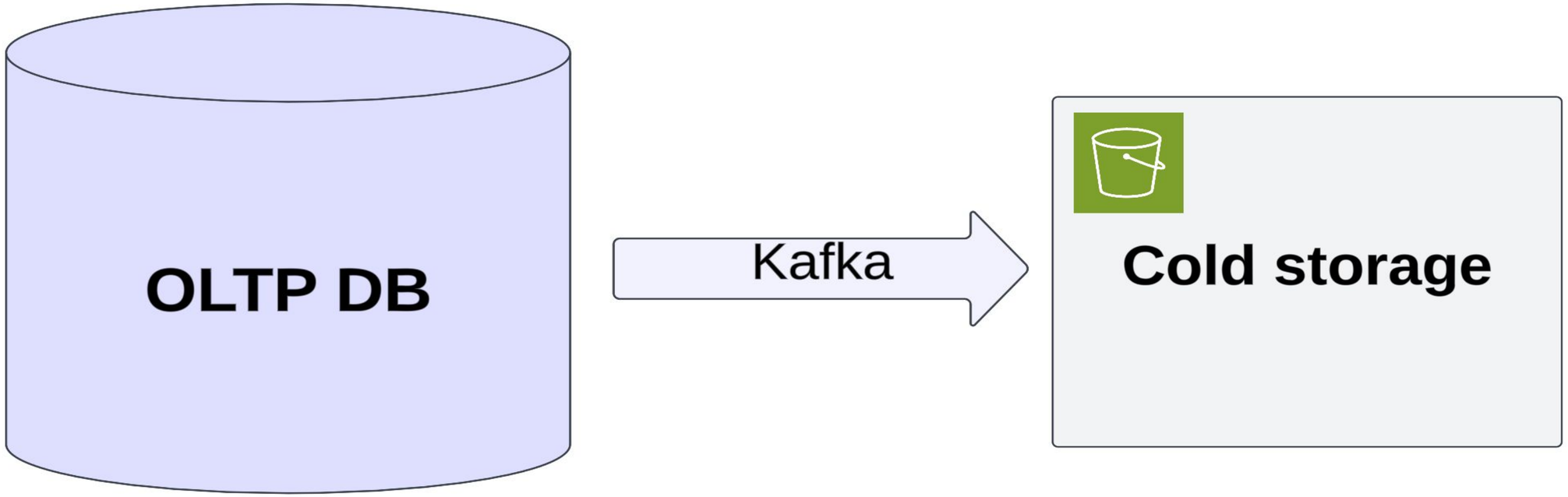
Why It Matters: Removing old data from your primary database keeps it running efficiently and improves query performance. Also it makes cold storage cheaper and reduces the cost of maintaining high-performance storage for inactive data.



The pyramid. Real world example.

Data archival

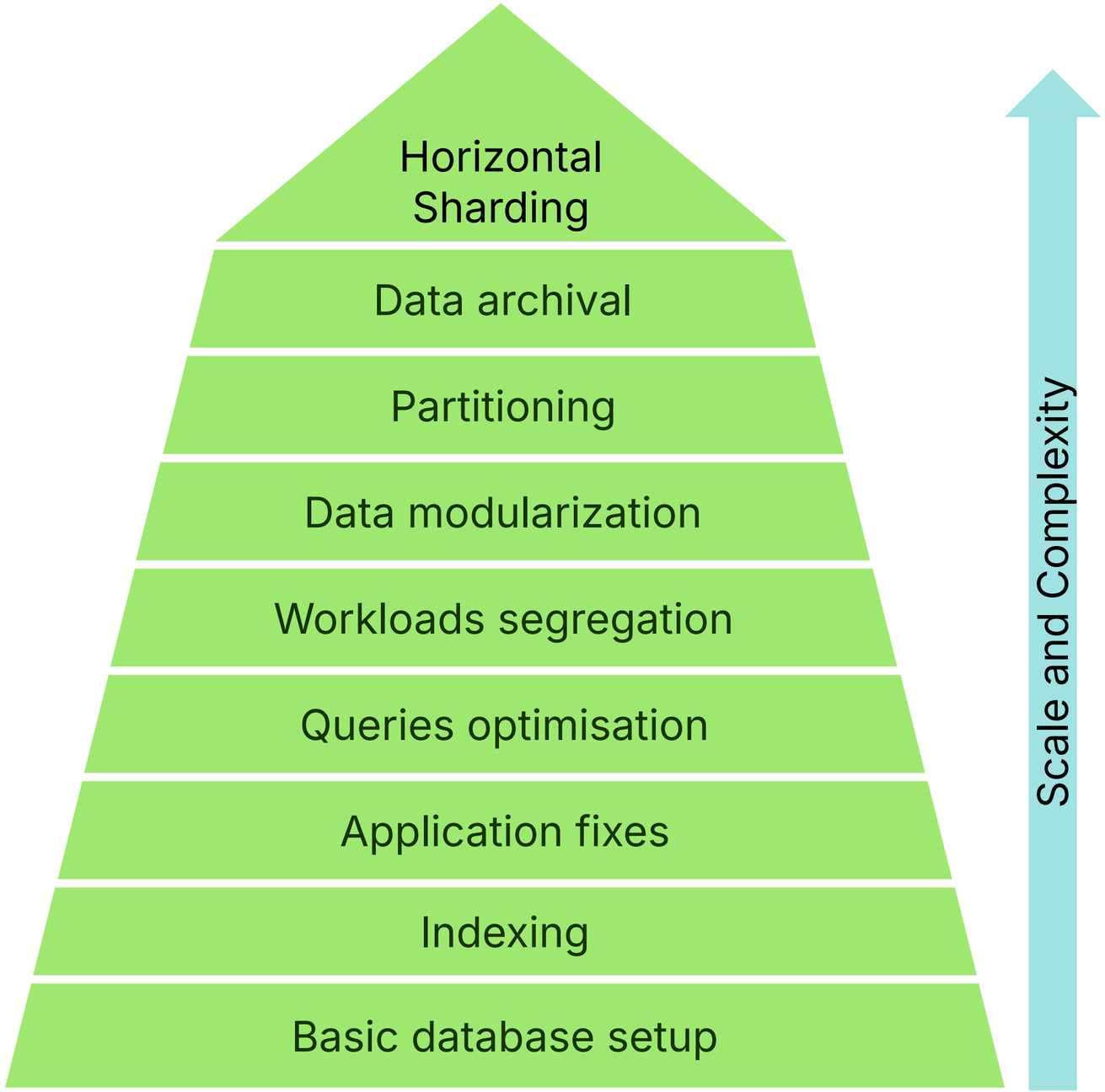
Problem: Database performance and cost degrades with growing volumes.



Solution: Cold data moved to a long-term storage with a slow access.

The pyramid.

Horizontal Sharding



What It Involves: Splitting data across multiple databases (shards) to distribute the load and allow the database to scale horizontally.

Why It Matters: Sharding allows your database to scale beyond the limits of a single machine, making it capable of handling large datasets and high traffic.

The pyramid. Real world example.

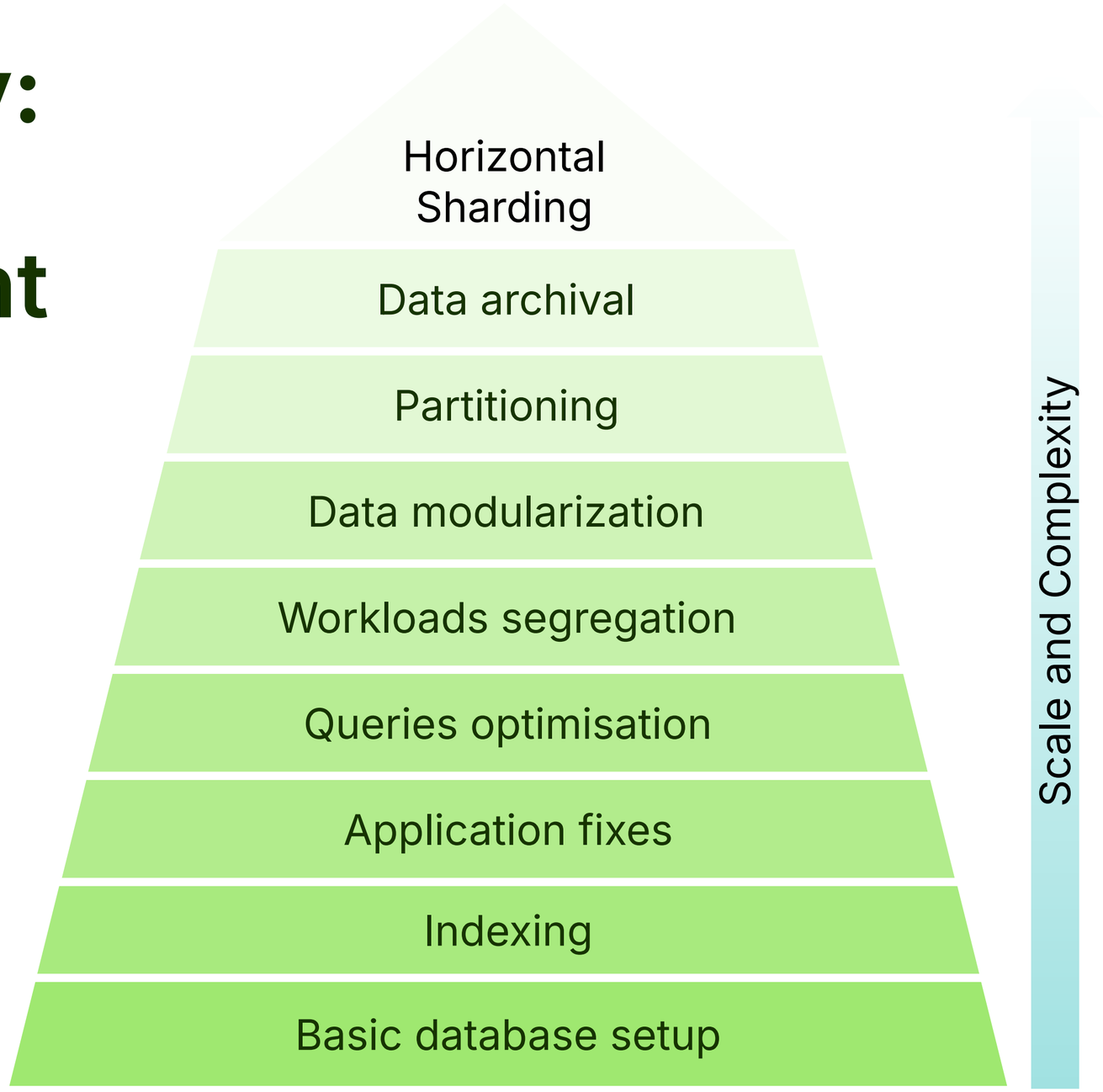
Horizontal Sharding

A step we haven’t needed to take... yet.

TAKEAWAYS

The pyramid.

Scaling 400x smartly: Applying just enough complexity at the right time.



Our scaling journey wasn’t smooth sailing, but we made it happen.

So can you.

WISE

Stay in Touch

Dmytro Hnatiuk

www.linkedin.com/in/dhnatiuk