Technical Project Deep Dive

Yi Zhang

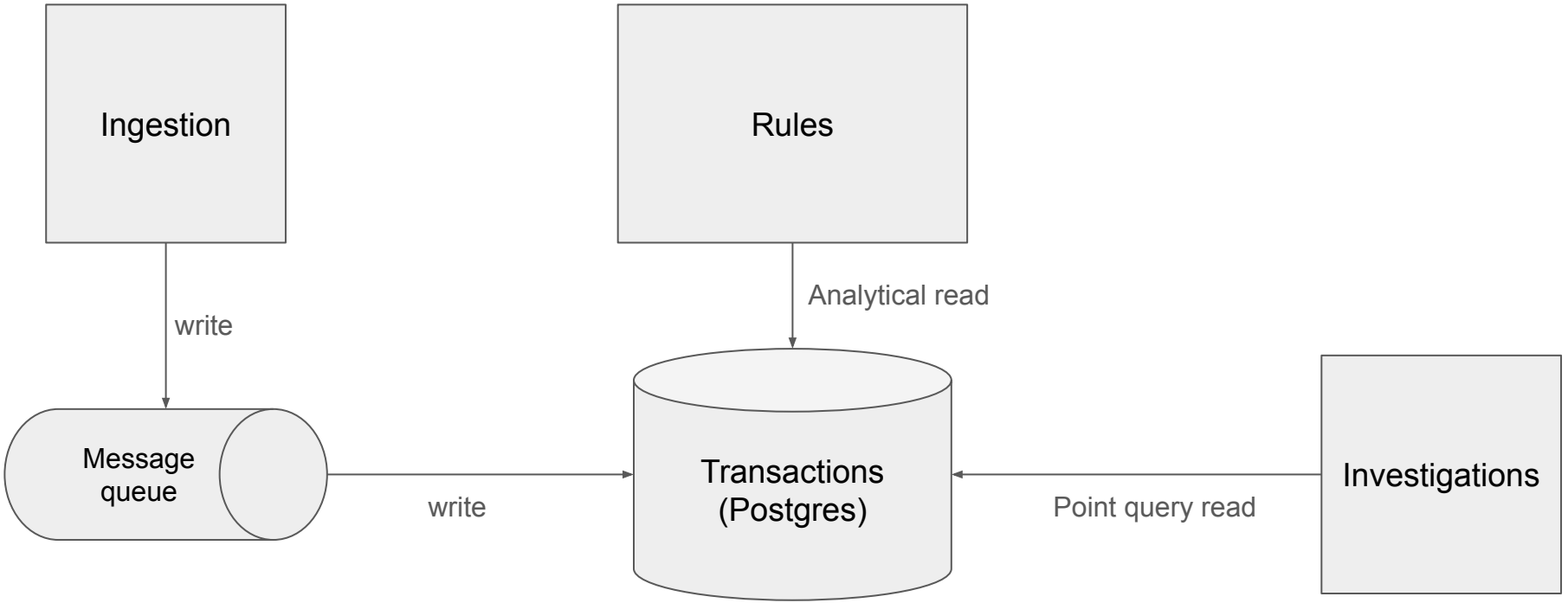
Overview

- Migrated from a single Postgres DB to a distributed backend involving 4 storage solutions, multiple streaming & batch pipelines, and 4+ microservices
- I was a contributor from day 1 and overall Tech Lead for ~1.5 years
- Project involved up to 5+ teams and 20+ engineers at its peak
- Raised peak supported ingestion volume from 5k tps to 150k tps, while cut infra spend per transaction by 50%

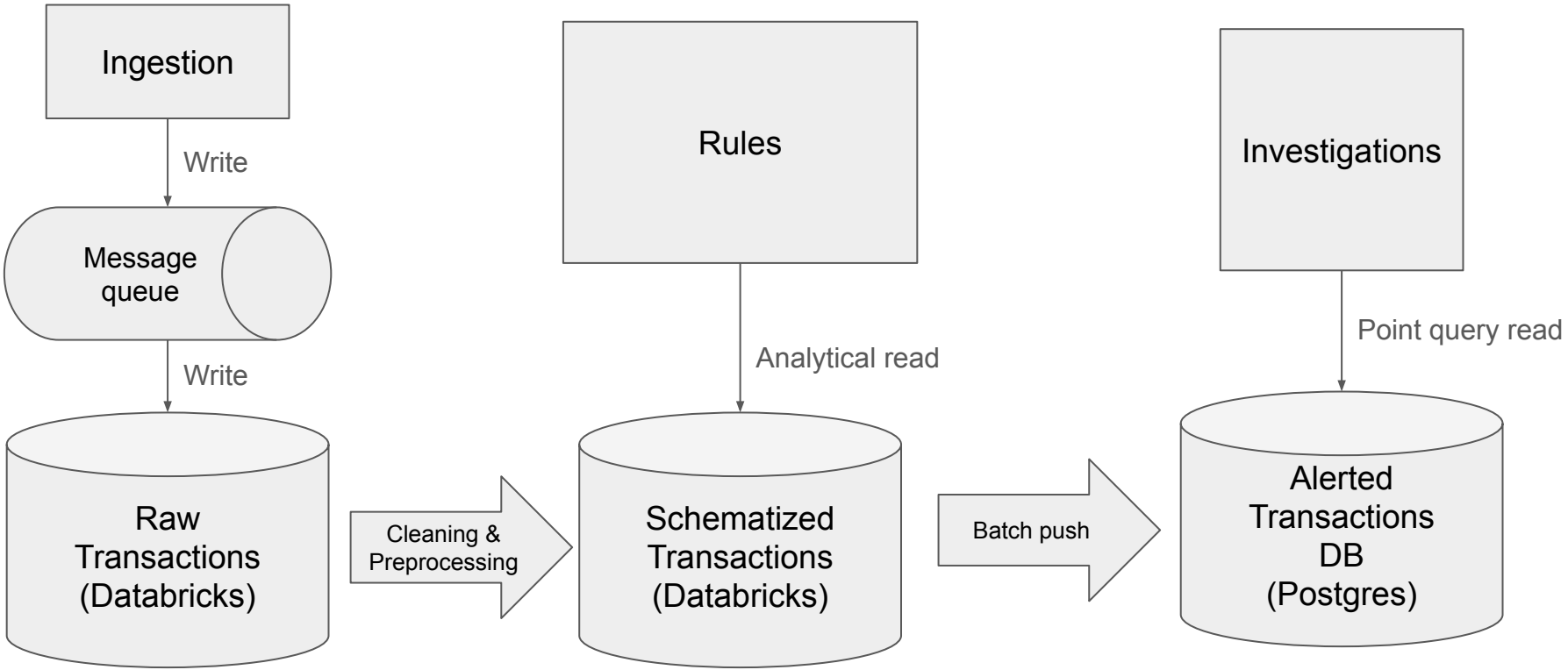
Background

- Unit21 makes vertically integrated software that performs transaction monitoring for compliance (e.g. anti-money laundering), and later on, fraud, use cases
- Customers ingest transactions into U21 via an API
- Customers manage a set of customizable Rules that are meant to detect suspicious activities
- Rules create alerts, which go into operational queues, to be investigated by operations specialists

System Architecture (initial state)

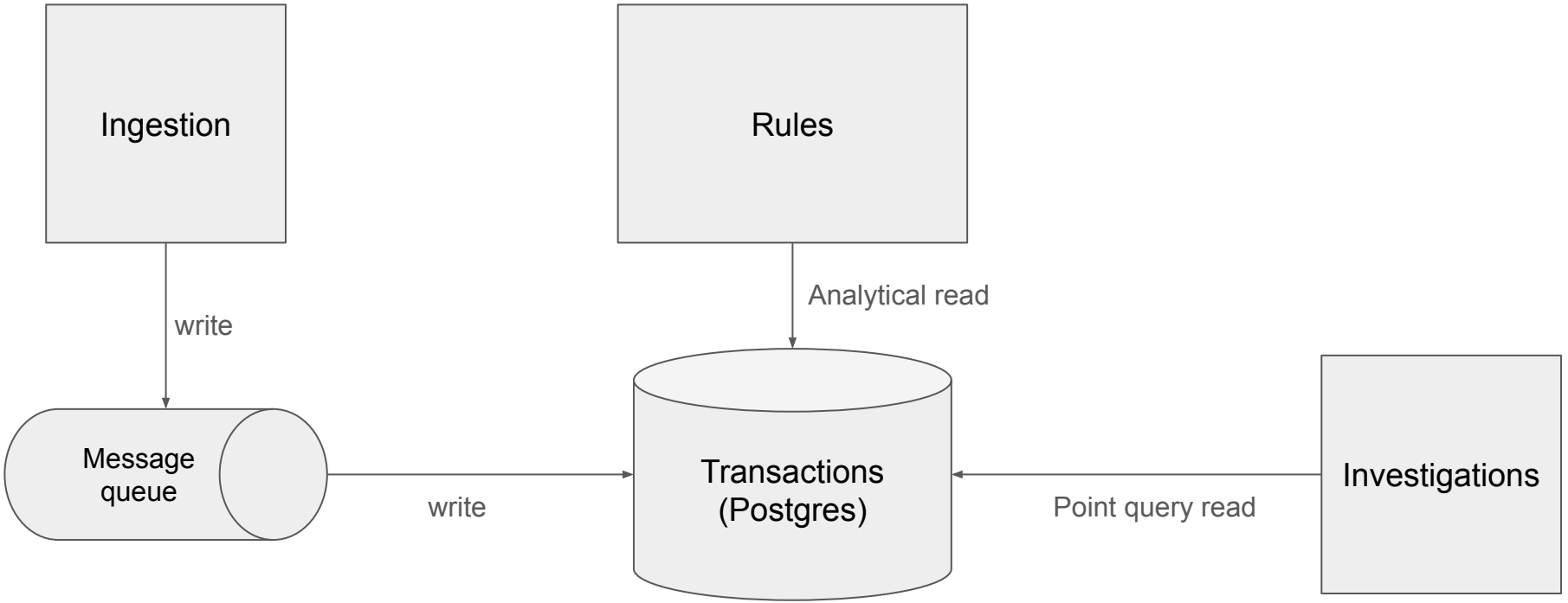


System Architecture (end state)



The Journey Begins

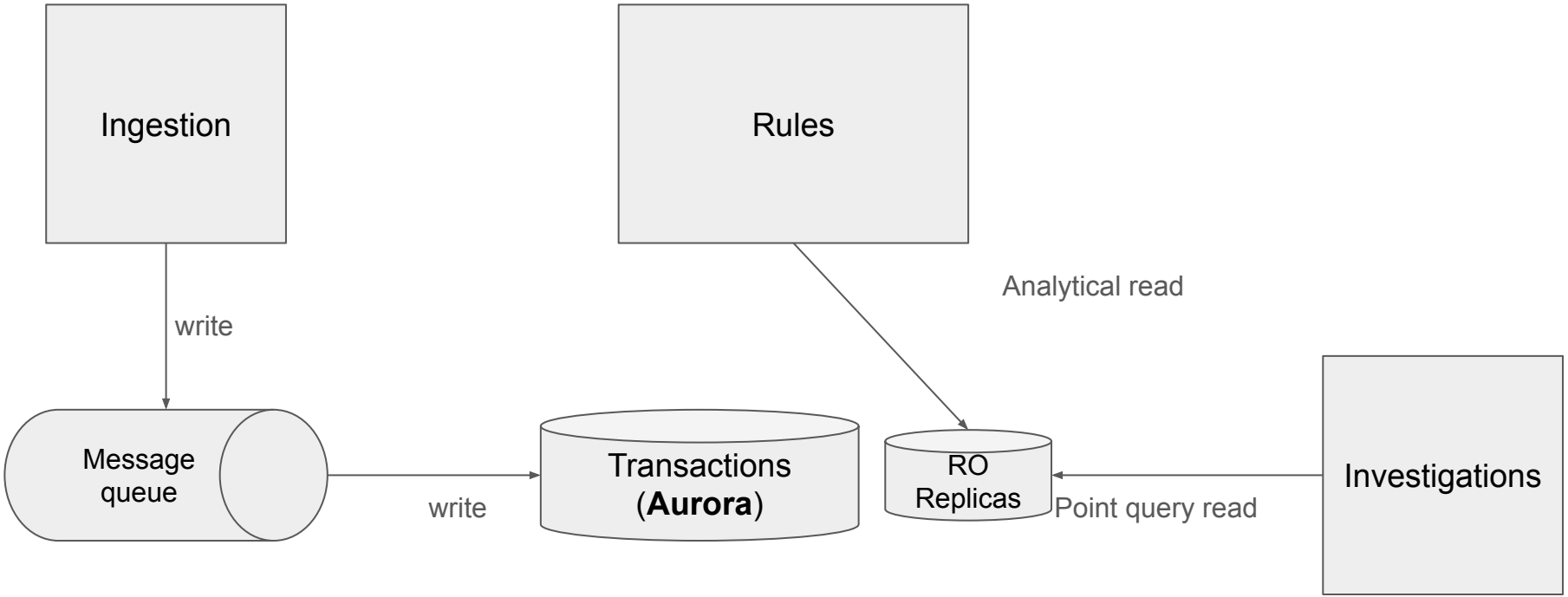
Stage 0


Ingestion
Rules
write
Analytical read
Message queue
write
Transactions (Postgres)
Point query read
Investigations

Stage 0 - Assessment

- As data volume grows, analytical queries starting to take a very long time (10min+)
- This causes frequent and unpredictable locking issue
- Ingestion delay starting to get out of hand
- Vanilla AWS RDS Postgres approaching its size limit
- AWS RDS IO Cost becoming large as well

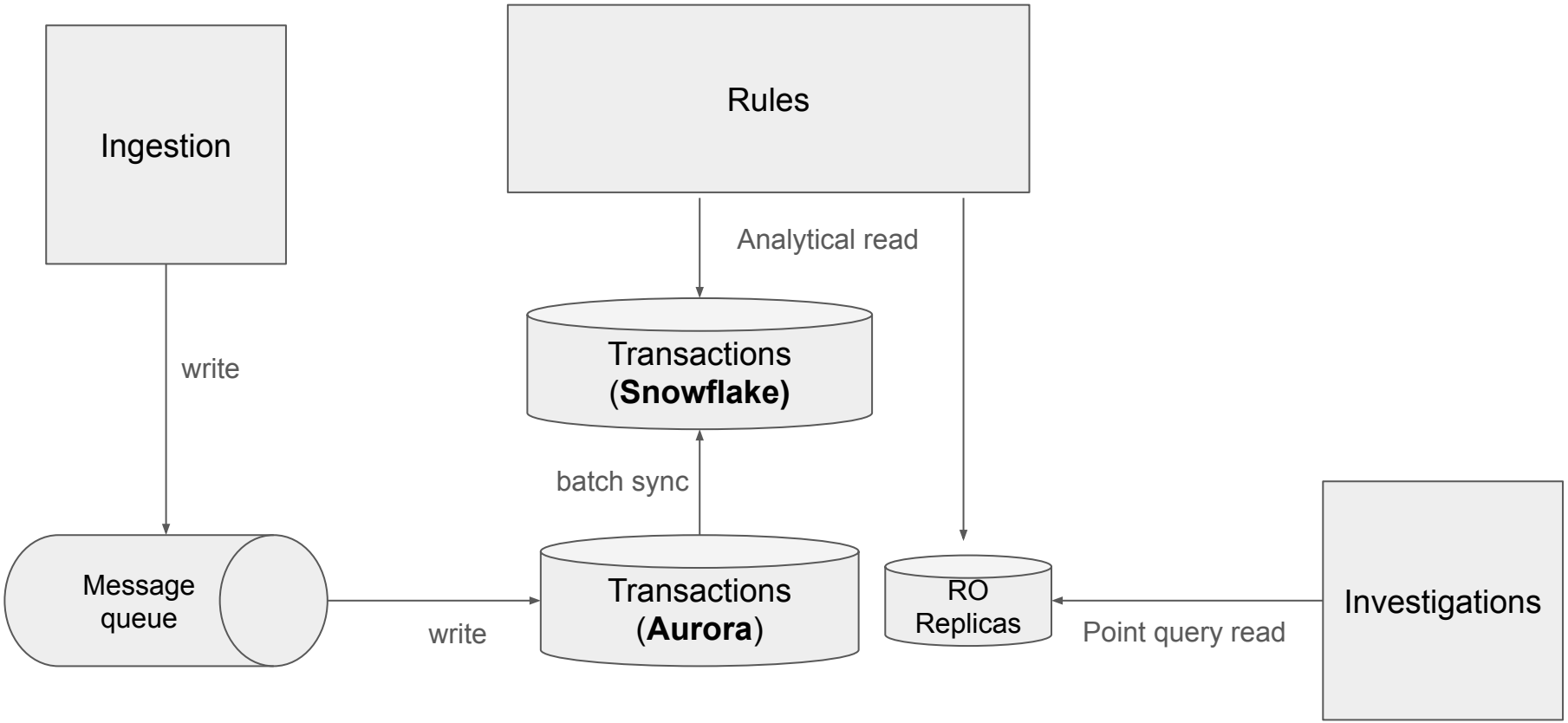
Stage 1


Ingestion
Rules
write
Analytical read
Message queue
write
Transactions (**Aurora**)
RO Replicas
Point query read
Investigations

Stage 1 - Assessment

- Switching over to Aurora IO Optimized instances cut cost by 15%
- Using read-only replicas helped with the locking issue
- However, we had incidents caused by read replicas & failover
- Data size still growing too quickly, we’ll run out of disk space soon
- Analytical reads can still take very long time

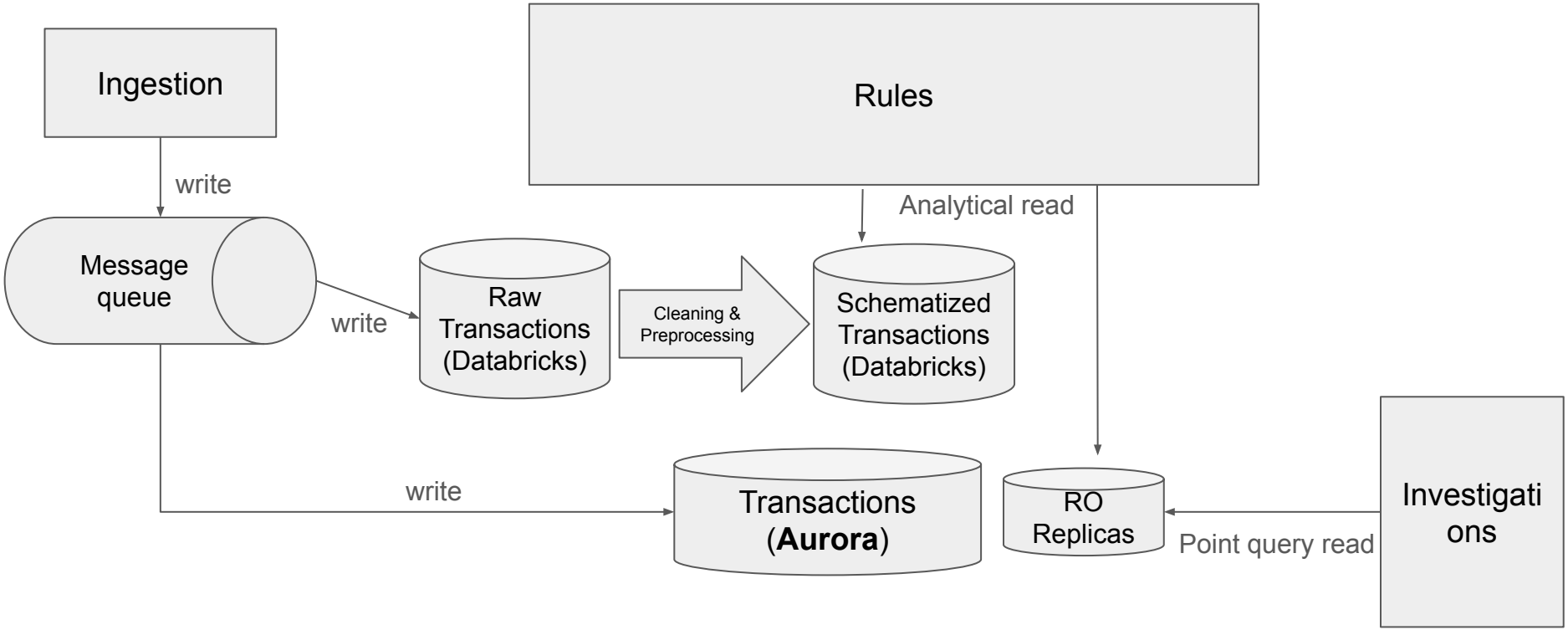
Stage 2


Ingestion
Rules
Analytical read
write
Transactions
(**Snowflake**)
batch sync
Message
queue
write
Transactions
(**Aurora**)
RO
Replicas
Point query read
Investigations

Stage 2 - Assessment

- Onboarded Snowflake as a Data Warehouse to support growing volume & complexity of analytical queries
- Now analytical reads have the flexibility to hit either data store, and performance has improved a lot
- However, this introduced more load (batch sync) on Aurora, while not able to reduce its size, we’re still at the risk of running out of disk space
- Overall cost still increased adding Snowflake

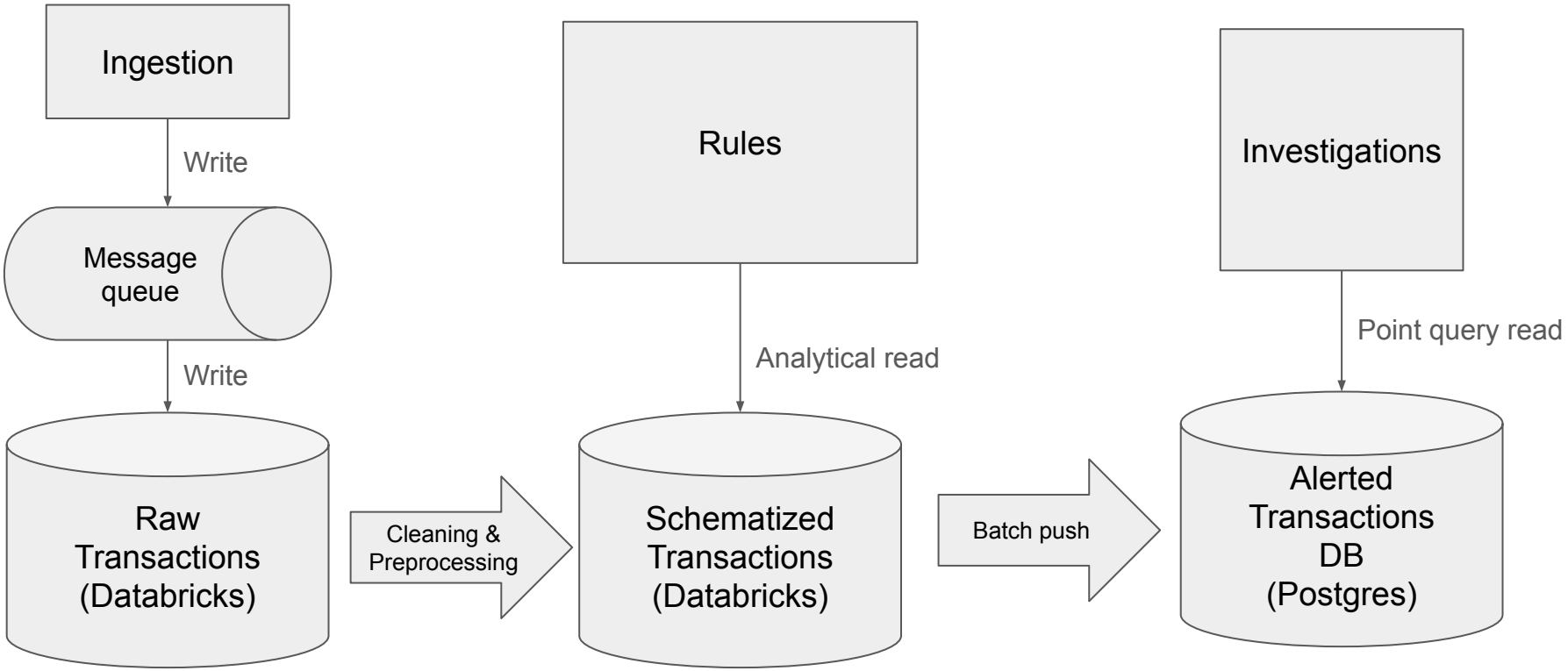
Stage 3


Ingestion
Rules
write
Analytical read
Message queue
write
Raw Transactions (Databricks)
Cleaning & Preprocessing
Schematized Transactions (Databricks)
write
Transactions (**Aurora**)
RO Replicas
Point query read
Investigations

Stage 3 - Assessment

- Instead of populating the Data Warehouse from Aurora, it’s much more efficient to ingest into it directly from message queue, so we’re working towards flipping the order of the data flow
- A Proof-of-Concept with Databricks showed major metric improvements, and it allowed deploying in customer’s cloud, which was very important for sales
- Following Databricks recommendations, we built Bronze -> Silver -> Gold pipelines to prepare for the analytical workloads
- The Databricks ingestion flow now has a much higher throughput, and data volume no longer a medium term concern

Stage 4


Ingestion
Write
Message queue
Write
Raw Transactions (Databricks)
Cleaning & Preprocessing
Rules
Analytical read
Schematized Transactions (Databricks)
Batch push
Investigations
Point query read
Alerted Transactions DB (Postgres)

Stage 4 - Assessment

- Gradually repurposed the legacy Aurora DB for Investigations use case
- Offboarded Snowflake entirely, and realized a major cost reduction
- There are more follow-ups to be had, but this state was already able to significantly raise the ceiling on data volume & vastly reduce cost per transaction

Major Challenges

Getting data into the Warehouse

- When onboarding Snowflake, batch syncing large parts of the table over introduced additional large load to the DB
- We attempted off-the-shelf solutions e.g. Fivetran, Logstash, but it look months and a few clever custom tricks for it to work
- Overall, incrementally syncing data turned out to be much more beneficial than letting large analytical queries hitting an OLTP DB

Sequential ID

- Many parts of the system had implicit dependency on the transactions having sequential IDs, which is no longer possible with ingestion into the Data Warehouse
- Had to meticulously remove this dependency across the entire system over months, including rewrite a large number of joins

Backward Compatibility & E2E validations

- Developed automated A/A testing for the new v.s. old pipelines to make sure ingestions are fully backward compatible
- Developed per customer/rule/execution knobs to allow granular control over the storage backend
- At one point, the system had to support Aurora + Snowflake + Databricks all at the same time, which was difficult to maintain
- Continuously monitoring new query patterns & make sure they are migration friendly

Learnings

Major Learnings

- Early prototyping & discovery is so important to flush out potential blockers
- Well-planned incremental steps with rigorous validation & options to switch back & forth is well worth it
- Building a strong alignment with the business on the complexity, magnitude, and the timeline of this project was absolutely essential
- Would have redirected the resources to be more focused on each self-contained component at a time with fewer parallel threads