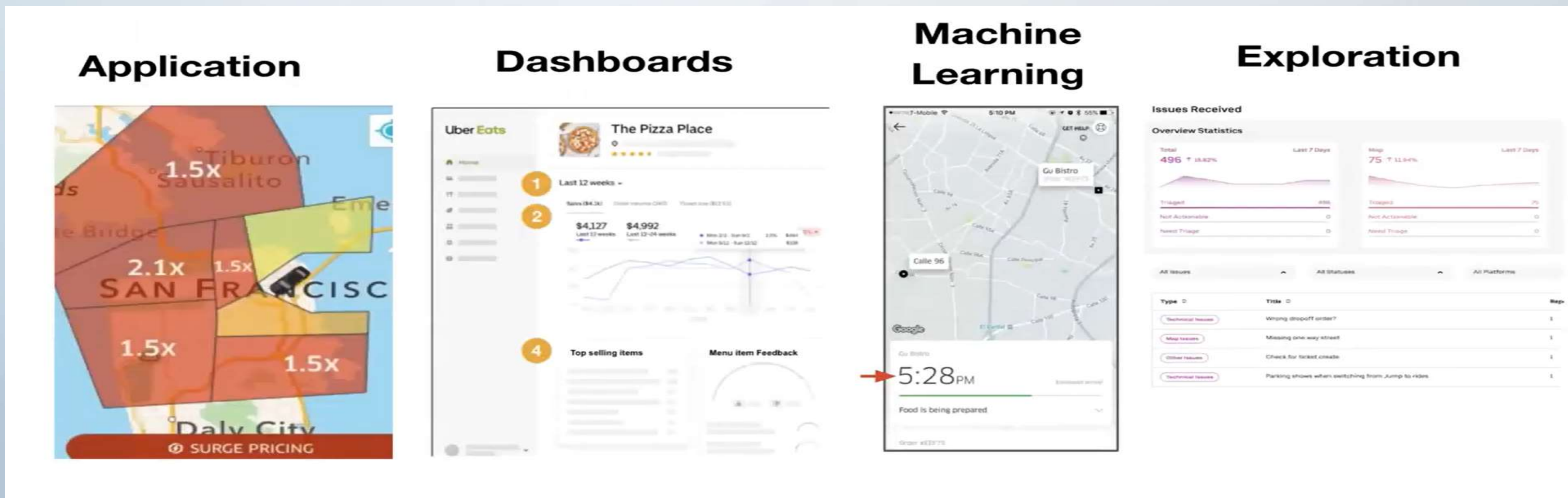
Real-time Data Infrastructure at Uber

Yupeng fu, Chinmay Soman
Uber, Inc

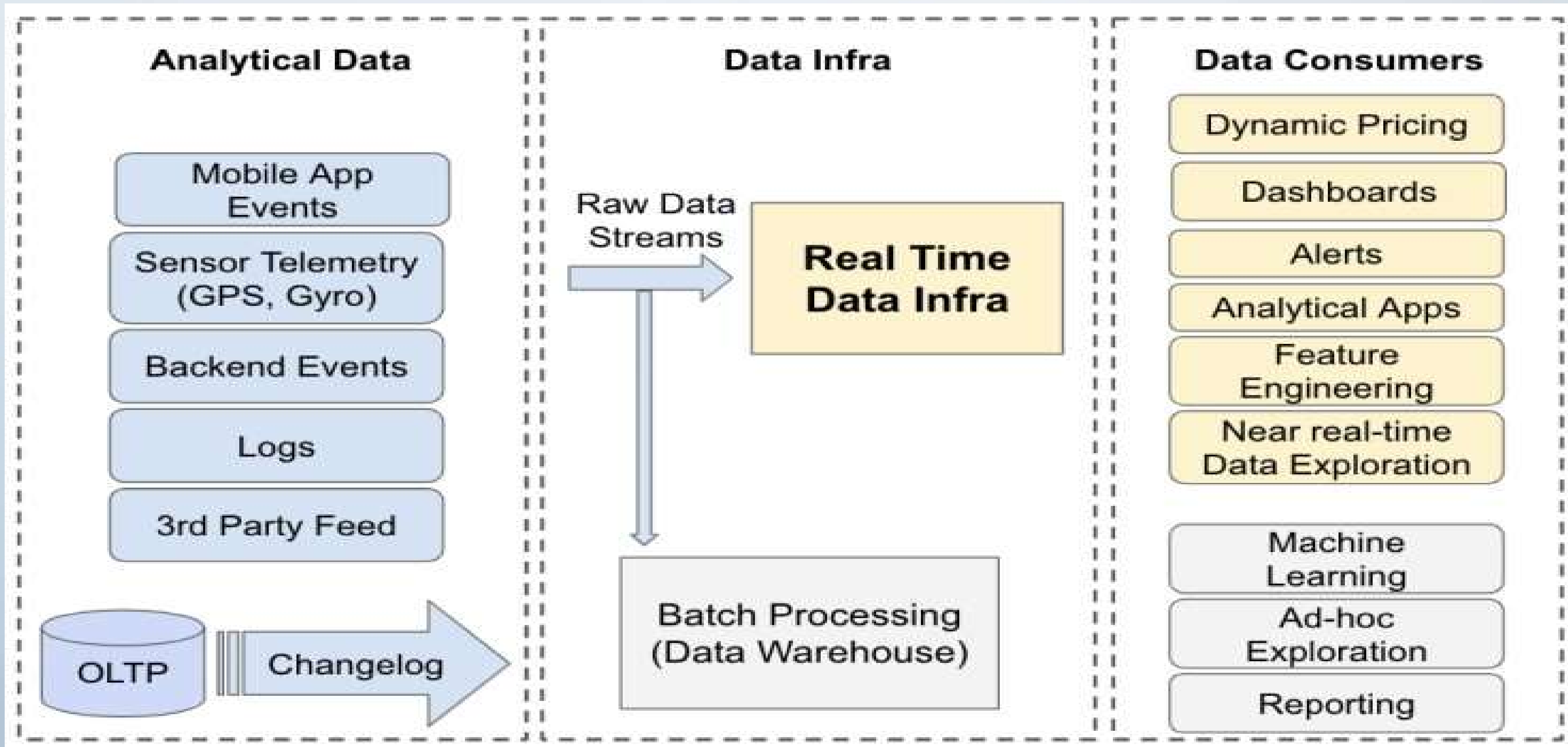
<u>PRESENTED BY</u>
BHARGHAV
SHIVAM

Data Generation @ Uber

- ✓ PBs of data is collected from the end users such as Uber drivers, riders, restaurants, eaters etc.
- ✓ For customer incentives, fraud detection, machine learning model prediction, data is useful to make decisions.
- ✓ Also, this data is useful for engineers, data scientists, executives & operations.



High-level data flow at uber infrastructure



Real time data processing needs & Scaling challenges

3 Main Areas to focus

- ✓ Messaging Platform
- ✓ Stream Processing
- ✓ OnLine Analytical Processing (OLAP)

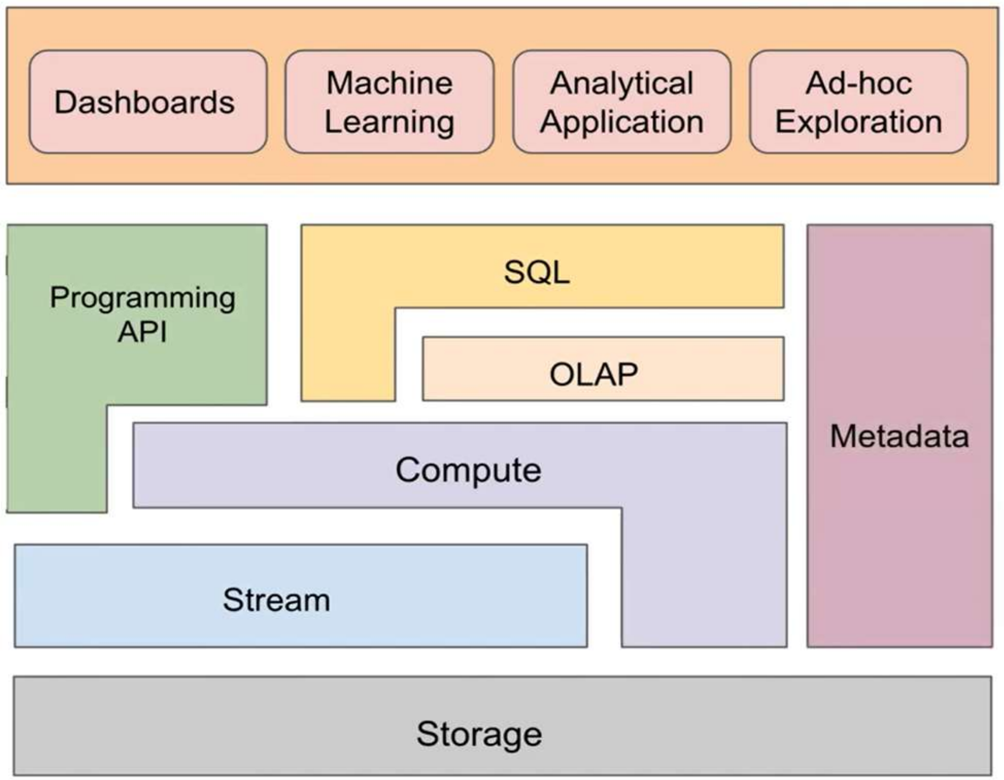
Each area must deal with 3 fundamental Scaling Challenges

- ✓ Scaling data
- ✓ Scaling use cases
- ✓ Scaling users

Use cases – Requirements pertaining to real time data infrastructure

- ✓ Consistency : Data to be consistent across all regions
- ✓ Availability : Infrastructure stack must be highly available
- ✓ Data Freshness : Events should be available quickly after their creation
- ✓ Query Latency : Require the P99th query latency to be <= 1 second
- ✓ Scalability : Ability to scale up with data without issues
- ✓ Cost : Ensure the cost of data processing & serving to be low
- ✓ Flexibility : Need programmatic & declarative interface

High level abstraction of real time data infrastructure – Overview of components



- For advanced users
- Full SQL Query layer
- Closer to ANSI SQL
- Stream + Batch
- At least once semantics
- Pub-sub interface
- At least once semantics
- Blob storage interface
- Read after write consistency
- Versioning
- Backwards compatibility checks
- At least once + Exactly once
- Real time in terms of ingestion and query latency

Apache Kafka for streaming storage

✓ An open-source distributed event streaming system.

✓ At Uber, Kafka is used for different workflows

- Propagating event data from rider and driver apps
- Enables a streaming analytics platform
- Ingesting all sorts of data into Apache's Hadoop data lake

To meet Uber's requirements, they customized Kafka and added enhancements.

Apache Kafka Enhancements

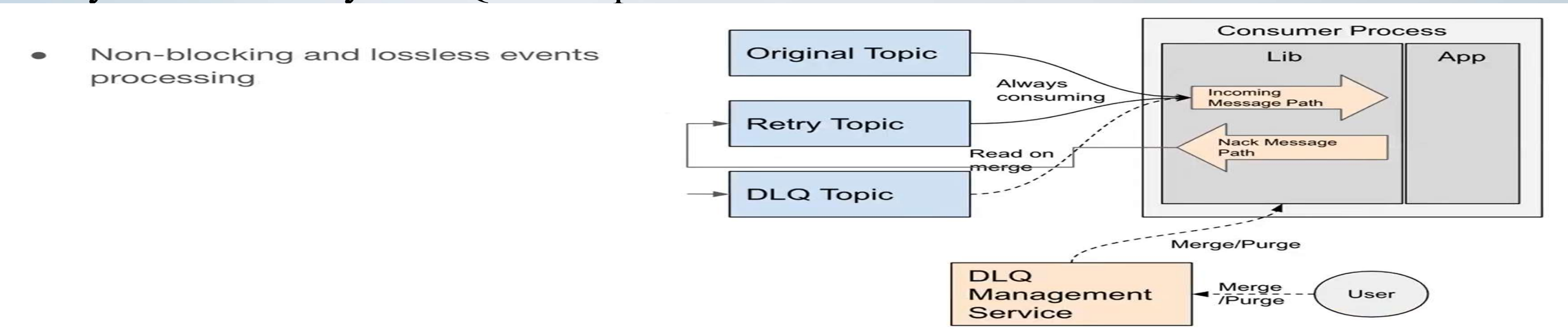
✓ Cluster Federation:

Used to improve reliability and scalability.

With federation, the Kafka service can scale horizontally by adding more clusters when a cluster is full

✓ Dead Letter Queue:

During failure of message processing, Kafka either drop those messages or retry indefinitely. DLQ on top of Kafka works as below

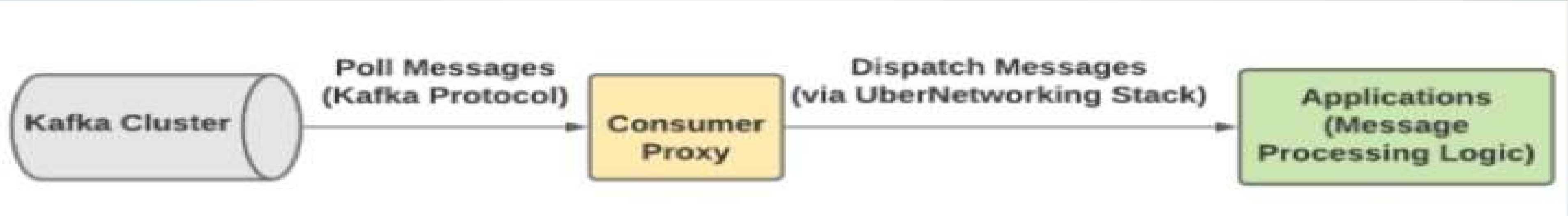


Apache Kafka Enhancements

✓ <u>Consumer Proxy:</u>

To support users in troubleshooting and debugging.

It is useful to retry the failure message and to send them through DLQ.



✓ <u>Cross-cluster Replication:</u>

uReplicator: In built rebalancing algorithm to minimize number of affected partitions during rebalancing.

Chaperone: No data loss during replication. Collects number of unique messages in a tumbling time window at every replication.

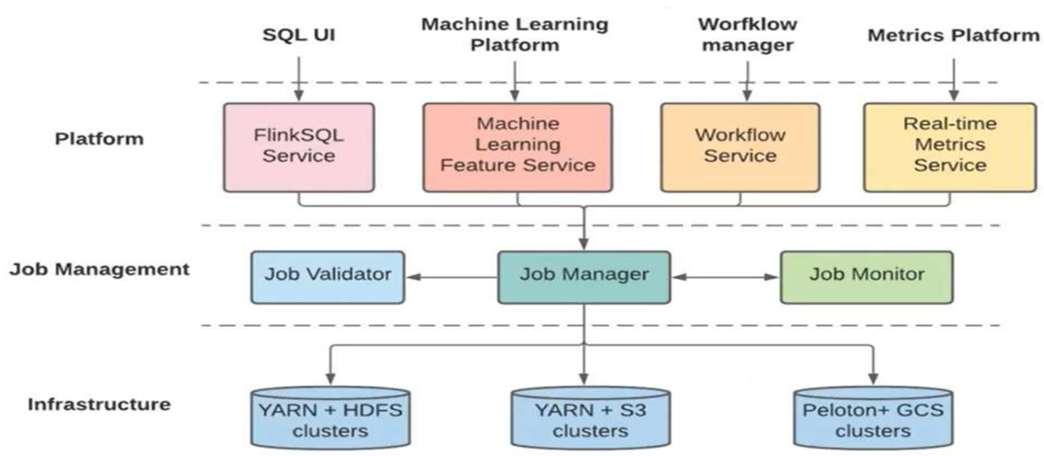
Apache Flink for stream processing -Enhancements

✓ Building streaming analytical applications with SQL:

Flink SQL: converts the input SQL query into a logical plan, runs it through the query optimizer and creates a physical plan which can be translated into a Flink job using the corresponding Flink API

✓ Unified architecture for deployment, management and operation:

- Heterogeneous Systems
  - Job Type (SQL, FaaS, DSL)
  - Flink Version
  - Cluster Management Systen
- High Availability
  - Deployment Flow
  - Dependent Systems
- Deployment Efficiency
  - Minimum Downtime during upgrade
  - Disaster recovery
- Reliability of Ecosystem

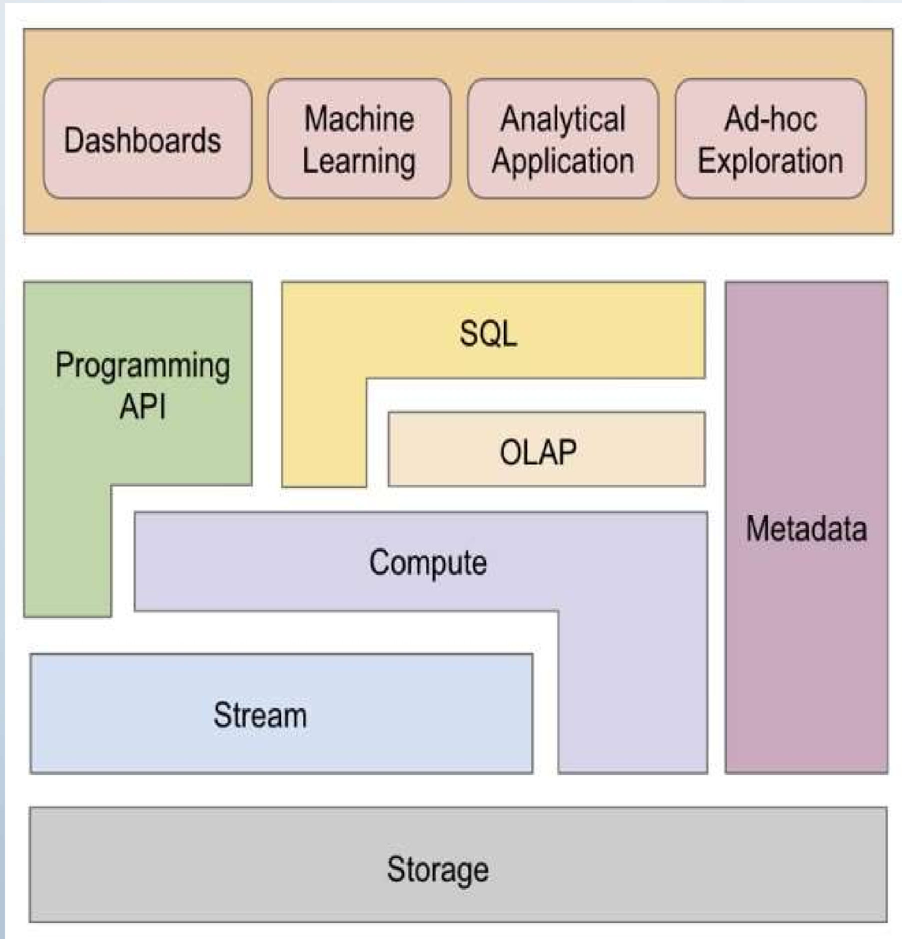


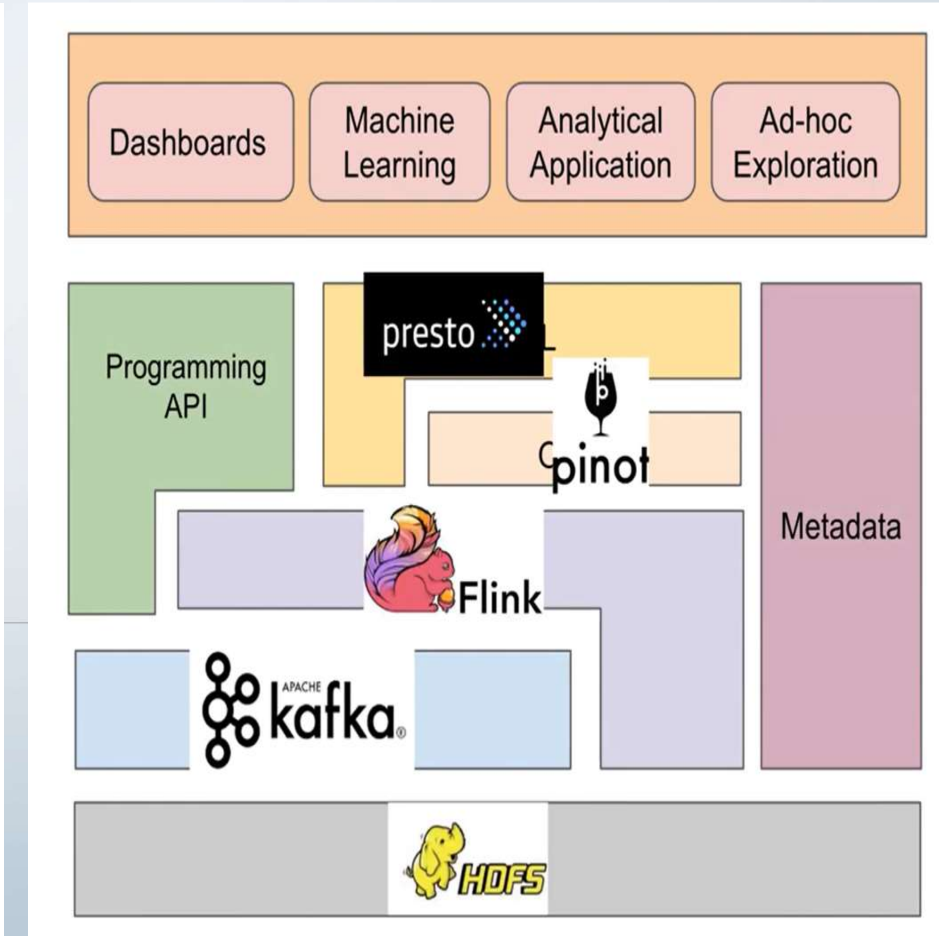
Apache Pinot for OLAP -Enhancements

- ✓ Upsert support: Scalable upsert solution on Pinot implemented to update the records during the real-time ingestion into the OLAP store.
- ✓ Full SQL support: Pinot lacks SQL features like subqueries and joins, so uber integrated Pinot with Presto for enabling standard PrestoSQL queries on Pinot tables
- ✓ Integration with the rest of Data ecosystem: Pinot integrates with FlinkSQL as a data sink, so customers can simply build a SQL transformation query and the output messages can be "pushed" to Pinot.
- ✓ Peer to peer segment recovery: Segment store failures caused all data ingestion to come to a halt. An asynchronous solution is implemented where server replicas can serve the archived segments in case of failures

HDFS for archival store and Presto for Interactive query

- ✓ HDFS: Used for long term storage of all data. Apache Flink uses HDFS for maintaining the job check points. Apache Pinot uses HDFS for long term segment archival.
- ✓ Presto: To perform exploration on real-time data, Uber leveraged Presto’s connector model and built a Pinot connector to deeply integrate with Apache Pinot so that we can execute standard Presto SQL queries on fresh data.


Dashboards
Machine Learning
Analytical Application
Ad-hoc Exploration
Programming API
SQL
OLAP
Metadata
Compute
Stream
Storage


Dashboards
Machine Learning
Analytical Application
Ad-hoc Exploration
Programming API
presto
pinot
Metadata
Flink
APACHE kafka
HDFS

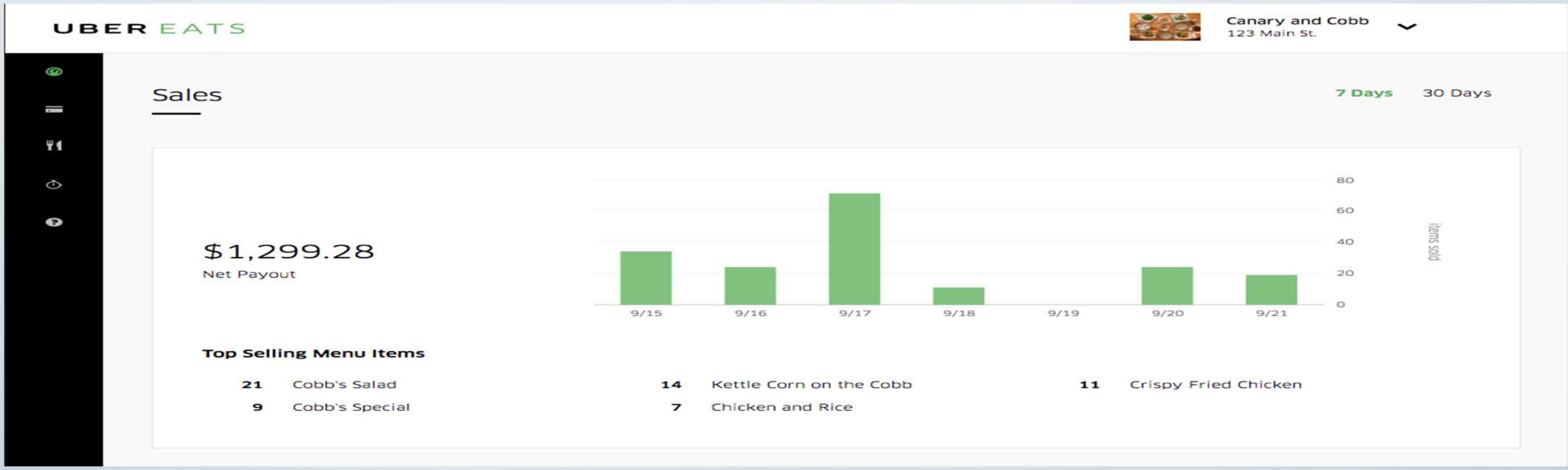
Use Cases Analysis

1. Analytical Application : Surge Pricing

- ✓ What is surge pricing?
- ✓ Key components – API, compute, stream
- ✓ Key requirements – data freshness and availability
- ✓ How it works
  - Ingest streaming data from Kafka
  - Run a machine learning based algorithm in Flink
  - Store the result in a key-value sink

Use Cases Analysis

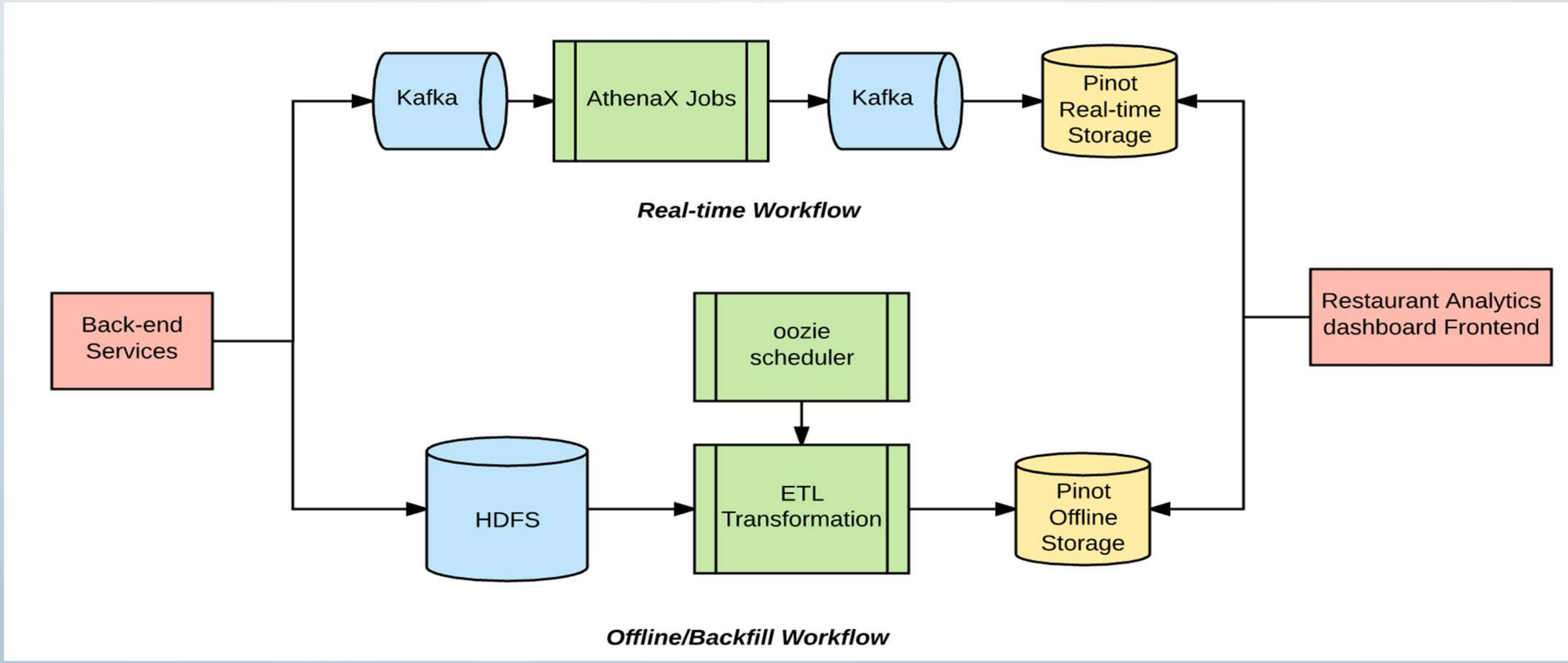
2. Dashboards: Ubereats restaurant manager



- ✓ Key components – SQL, OLAP, compute, stream, storage
- ✓ Key requirements – data freshness and low query latency

Use Cases Analysis

2. Dashboards: Ubereats restaurant manager (contd...)



Use Cases Analysis

3. Machine Learning: real-time prediction monitoring

- ✓ Machine Learning has been critical at uber
- ✓ Key components – API, SQL, OLAP, Compute, Stream, Storage
- ✓ Key requirements – Scalability
- ✓ A real-time prediction monitoring pipeline joins the predictions to the observed outcomes generated by the data pipeline
- ✓ Ongoing, live measurements of model accuracy are created

Use Cases Analysis

4. Ad-hoc exploration: Ubereats ops automation

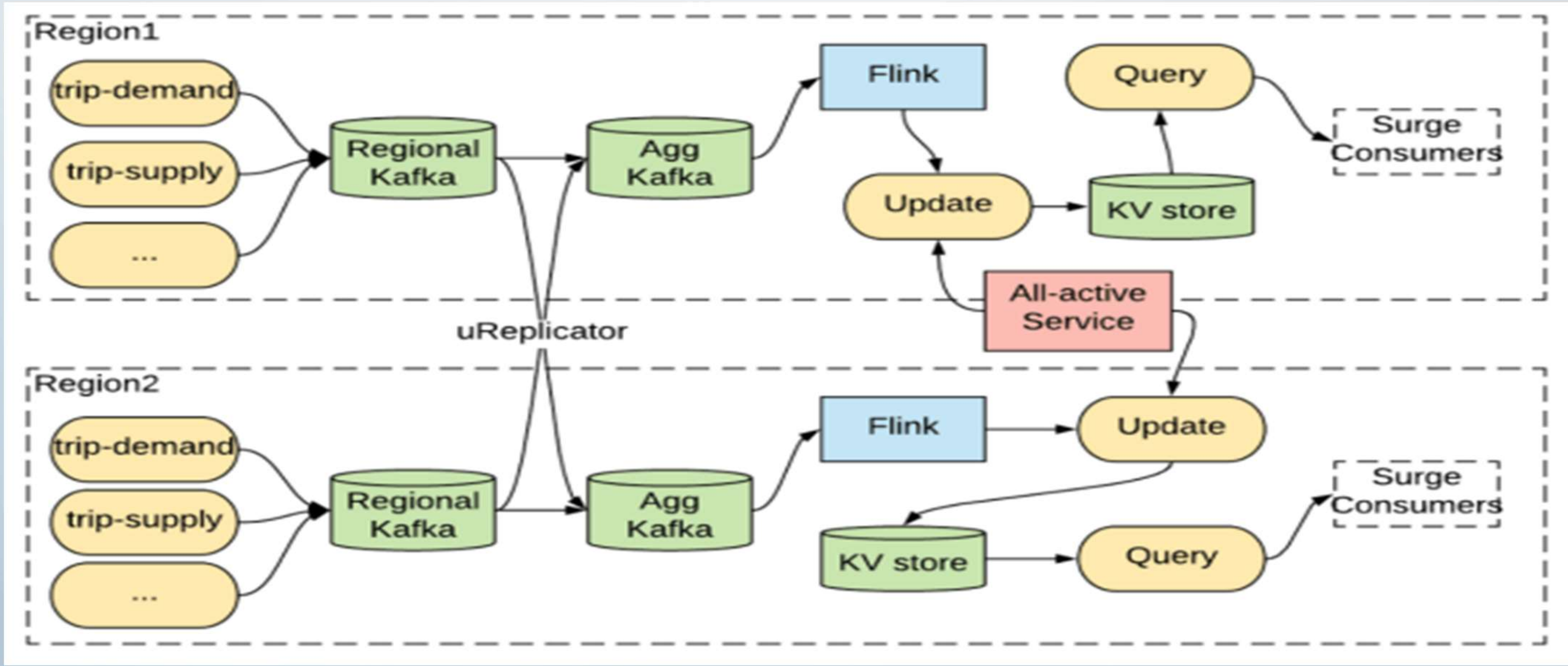
- ✓ To execute "ad hoc analytical queries" on real time data generated by users of uber
- ✓ Key components – SQL, OLAP, Compute, Stream
- ✓ Key requirements – Reliability and Scalability
- ✓ Uber ops team used it to combat covid 19 and keep restaurants open in different geographical locations
- ✓ Pinot, Presto and Flink were scaled easily with the organic data growth and performed reliably during peak hours

All Active Strategy

- ✓ Providing business resilience and continuity is a top priority for Uber
- ✓ Disaster recovery plans to minimize the business impact from natural and man-made disasters
- ✓ Ex- power outages, catastrophic software failures and network outages
- ✓ Multi-region strategy
- ✓ 2 types of setups
- ✓ Active-active
- ✓ Active-passive

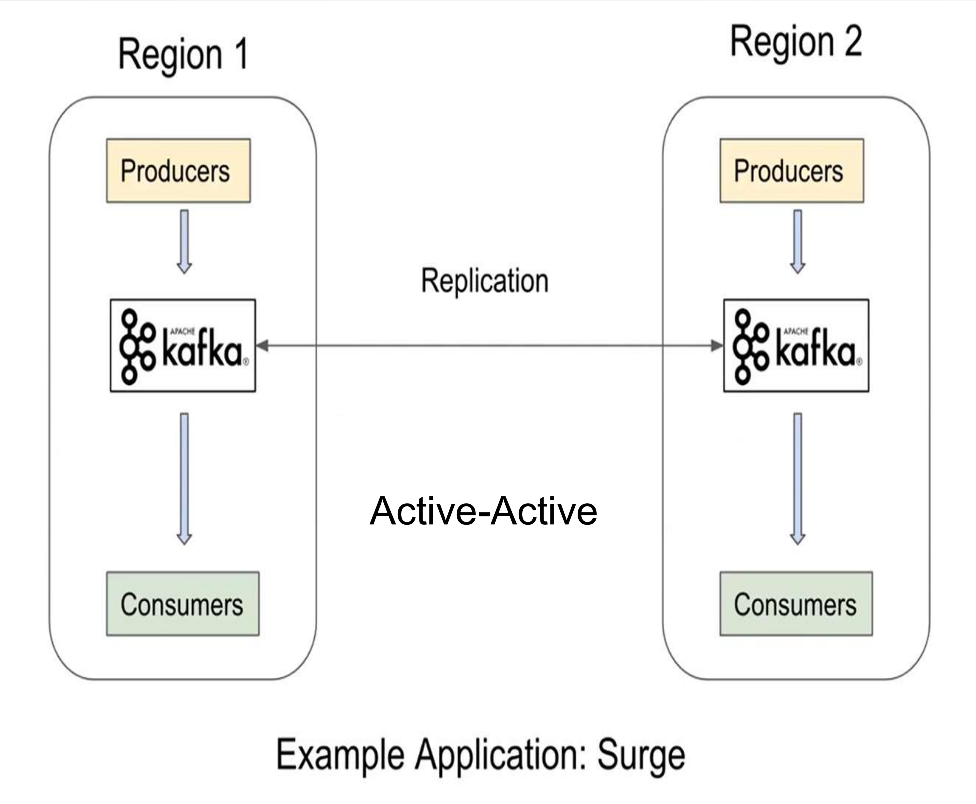
All Active Strategy

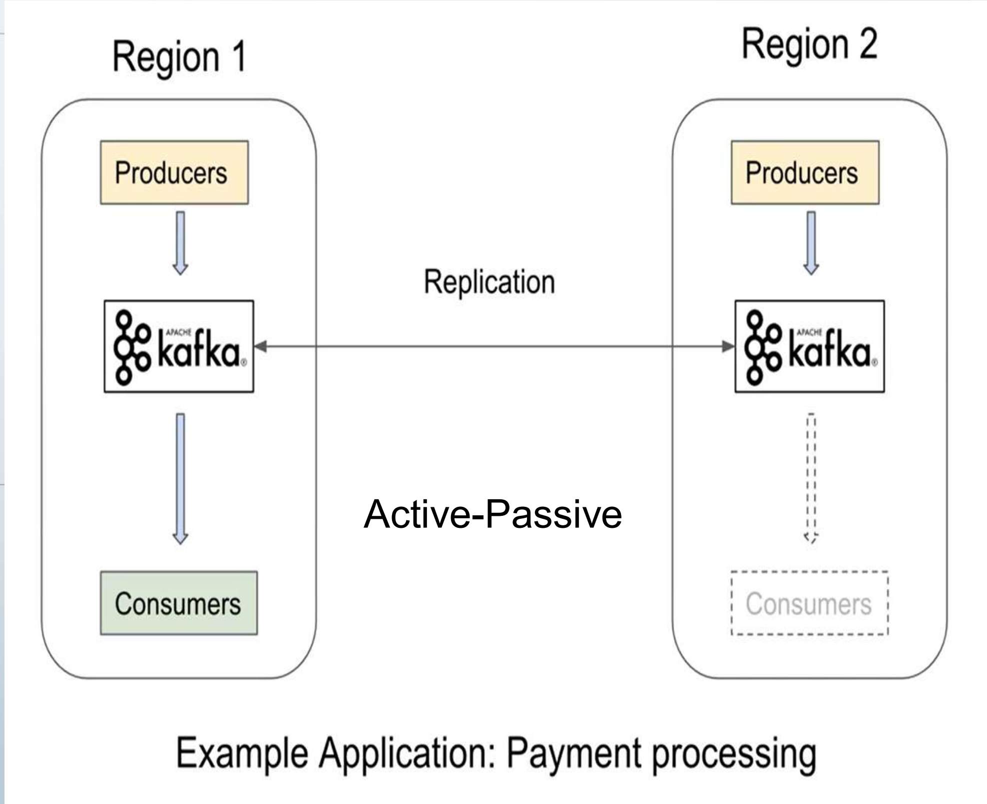
1. Active-Active Strategy



All Active Strategy

2. Active-Passive Strategy





Backfill

- ✓ Wherever there is real-time big data processing, there is backfill
- ✓ Reasons for reprocessing data:
  - New data process pipelines
  - New machine learning model
  - Bug discovered in a real-time application
  - Change in stream processing logic
- ✓ Solutions proposed:
  - Lambda architecture
  - Kappa architecture

Backfill

- ✓ Both Lambda and Kappa architectures have limitations
- ✓ Uber uses two solutions based on Flink
  - SQL-based
    - Same SQL query executed on both real-time and offline data
  - API-based
    - Reuse stream processing logic just like kappa architecture
    - Also has the ability to read archived data from offline datasets like hive
    - Same code can be executed on both streaming or batch data sources

Lessons Learnt

- ✓ Open Source adoption
  - Pros: Reduce time to market
  - Cons: Lots of executions
- ✓ Rapid system development and evolution
  - Enforce standardization (ex- kafka)
  - Monorepo, ci/cd
- ✓ Ease of operation and monitoring
  - Automation around cluster setup, dashboards, alerts
- ✓ Ease of user onboarding and debugging
  - Data discovery, Data audit, Self service onboarding

Future Work

- ✓ Streaming and Batch processing unification
- ✓ Multi-region and multi-zone deployments
- ✓ On-prem and cloud agnostic
- ✓ Tiered storage