Uber
Business Case Study On Application of Big Data Technologies

- Business Case Study Presentation by:
- Divya Vemula
- Rahul Chauhan
- Mani Krishna Tippani
- Shaheryar Nadeem

Introduction to Uber and Big Data

- Overview of Uber
  - Founded in 2009, Uber operates in ride-sharing, food delivery (Uber Eats), and freight logistics.
  - Uber operates in more than 60 countries and has millions of riders and drivers worldwide, making it one of the largest transportation networks globally.
  - Uber generates revenue from ride-sharing, food delivery, freight logistics, and advertising. It operates using a commission-based model, taking a percentage from both drivers and businesses.
- Why Big Data for Uber?
  - Uber generates 15 TB of data daily, processes it at a real-time rate of millions of events per second, and stores over 100 petabytes of data.
  - Massive scale operations demand efficient data handling.
  - Real-time decision-making crucial for rider-driver matching, dynamic pricing, and operational optimization.

Challenges at Uber and Need for Hadoop

Data Infrastructure challenges:

- Scalability became a challenge as Uber's operations expanded globally. As Uber handles petabytes of data daily, total incoming data volume is growing at an exponential rate Replication factor & several geo regions copy data.
- the infrastructure must scale seamlessly without re-architecture, while keeping costs low due to Uber's low-margin business model
- Data must be processed within seconds of generation to support real-time demand-supply adjustments and dynamic features like surge pricing.
- Uber relied on relational databases like MySQL and PostgreSQL. Uber needs to offer both **SQL** and **programmatic interfaces** to accommodate users with varying technical skills and support diverse business needs.
- Data silos created difficulties in generating a unified, global operational view.

Operational Bottlenecks:

- Data processing latency exceeded 24 hours, hindering timely decision-making.
- The inability to handle rapidly growing data volumes constrained operational efficiency and innovation.

Uber’s Data Challenges Before 2014:

- Before 2014, Uber used a few traditional OLTP databases (MySQL, PostgreSQL) to manage data., with no global view of the data
- As Uber expanded, data volume and complexity increased, requiring a centralized data warehouse to support analysis.

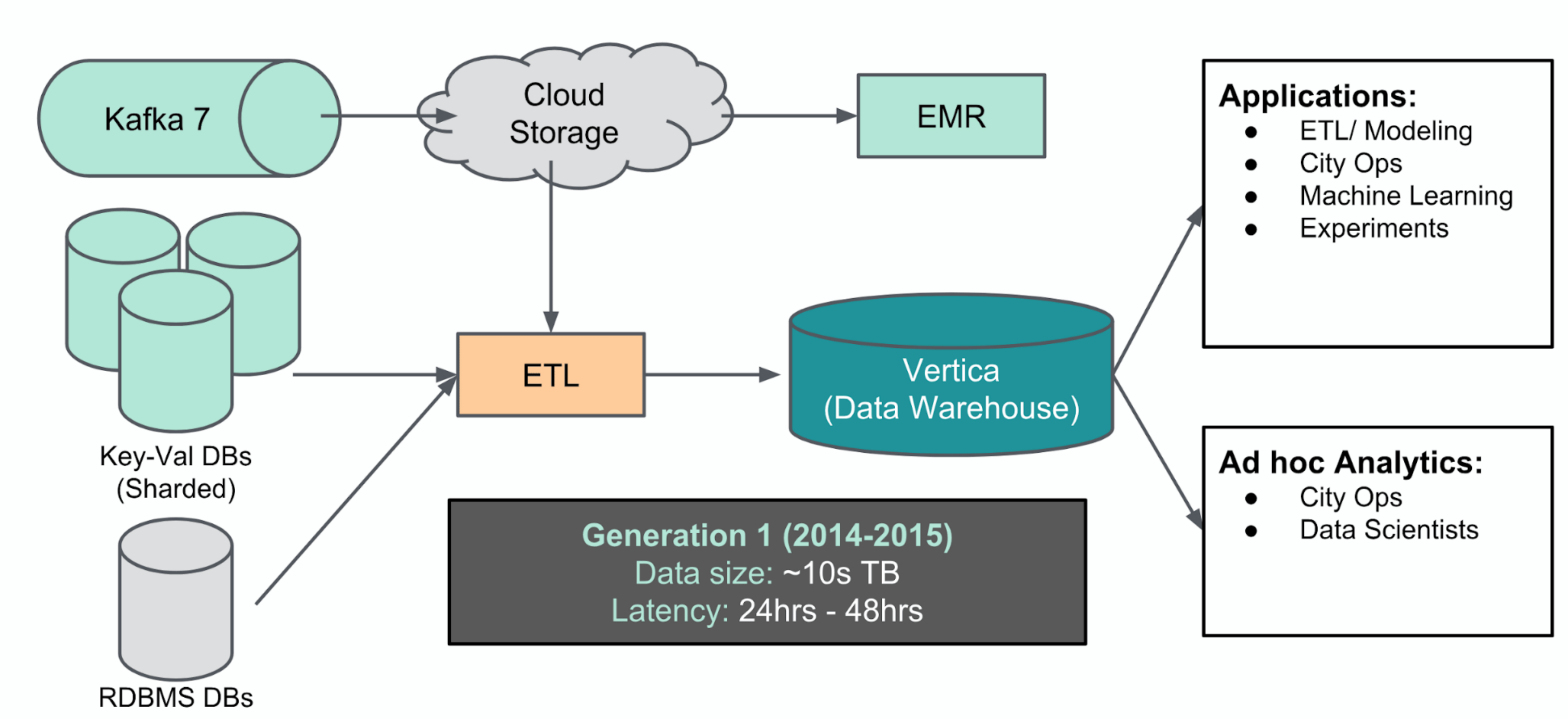
Beginning of Big Data at Uber 2014-15

- The Beginning of Uber’s Big Data platform allowed to aggregate all of Uber’s data in one place i.e., Vertica DW and provide standard SQL interface for users to access data.

Limitations

- ETL jobs lacked schema enforcement and formal contracts between data producers and consumers, which led to duplicate data ingestion.
- Most of data was in JSON format, and ingestion jobs were not resilient to changes in the producer code.

- Ad hoc ETL processes, caused data reliability concerns, inefficiencies and difficulty in integrate new data types.
- This led to pressure on upstream data sources and increased storage, operational costs.



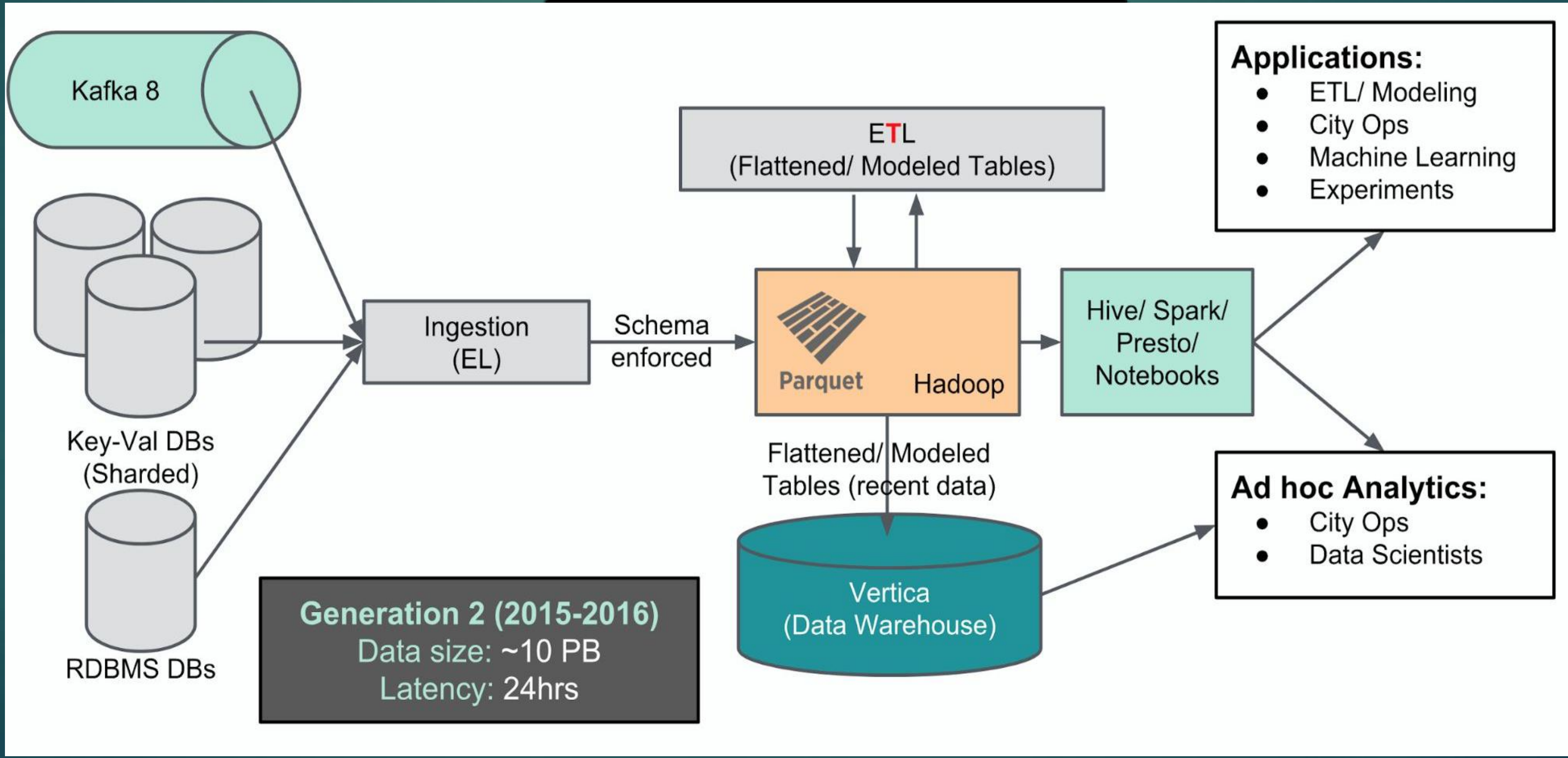
Arrival of Hadoop in Uber (2015-16)

Uber transitioned into a Hadoop data lake for raw data ingestion, avoiding transformations during ingestion to reduce pressure on online datastores.

Data access data in Hadoop:

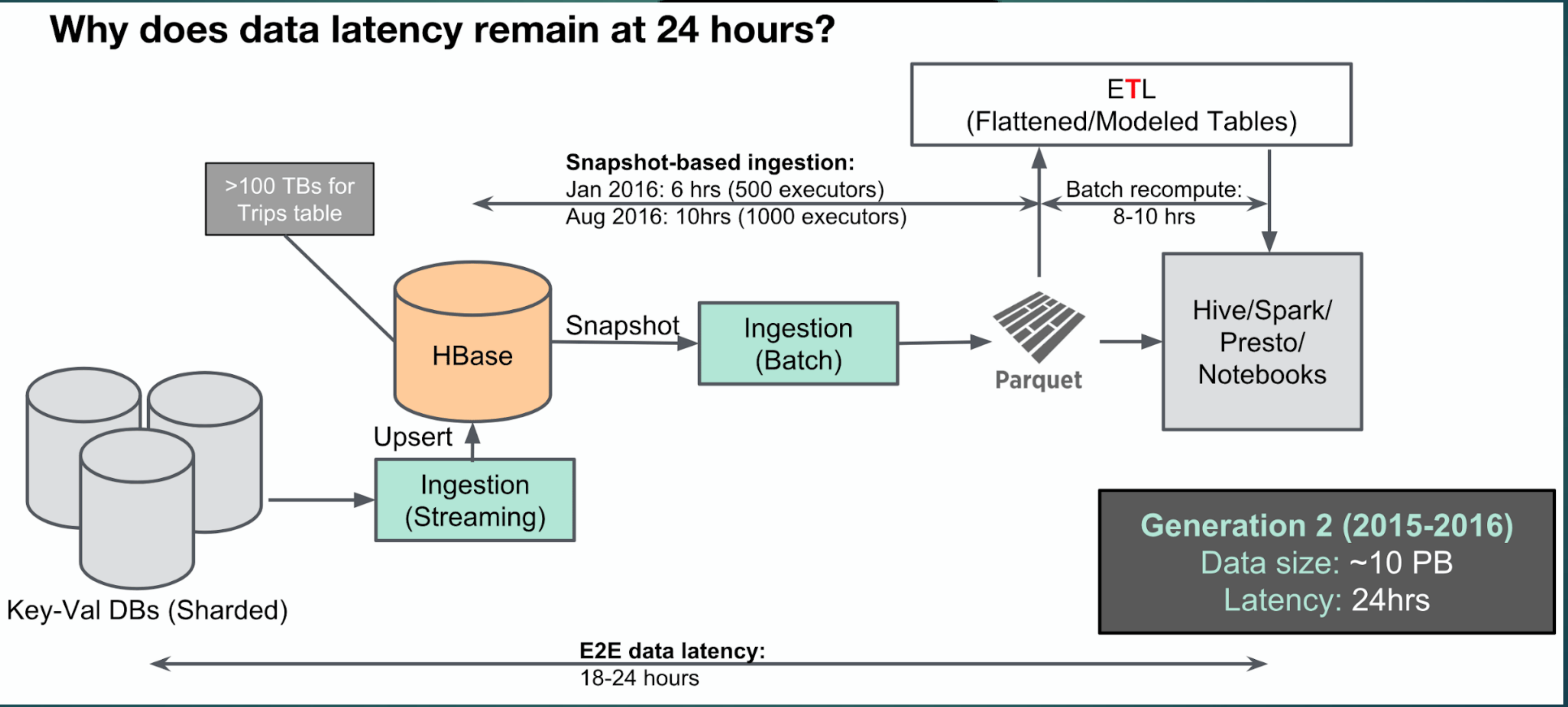
- **Presto -** enable interactive ad hoc user queries,
- **Apache Spark -** facilitate programmatic access to raw data
- **Apache Hive -** workhorse for extremely large queries.

- Data modeling and transformations only in Hadoop, enabling fast backfilling & recovery when issues arose.
- Transition from JSON to **Apache Parquet**, to store both schema and data together, ensuring data consistency and preventing issues from upstream format changes.



Limitations

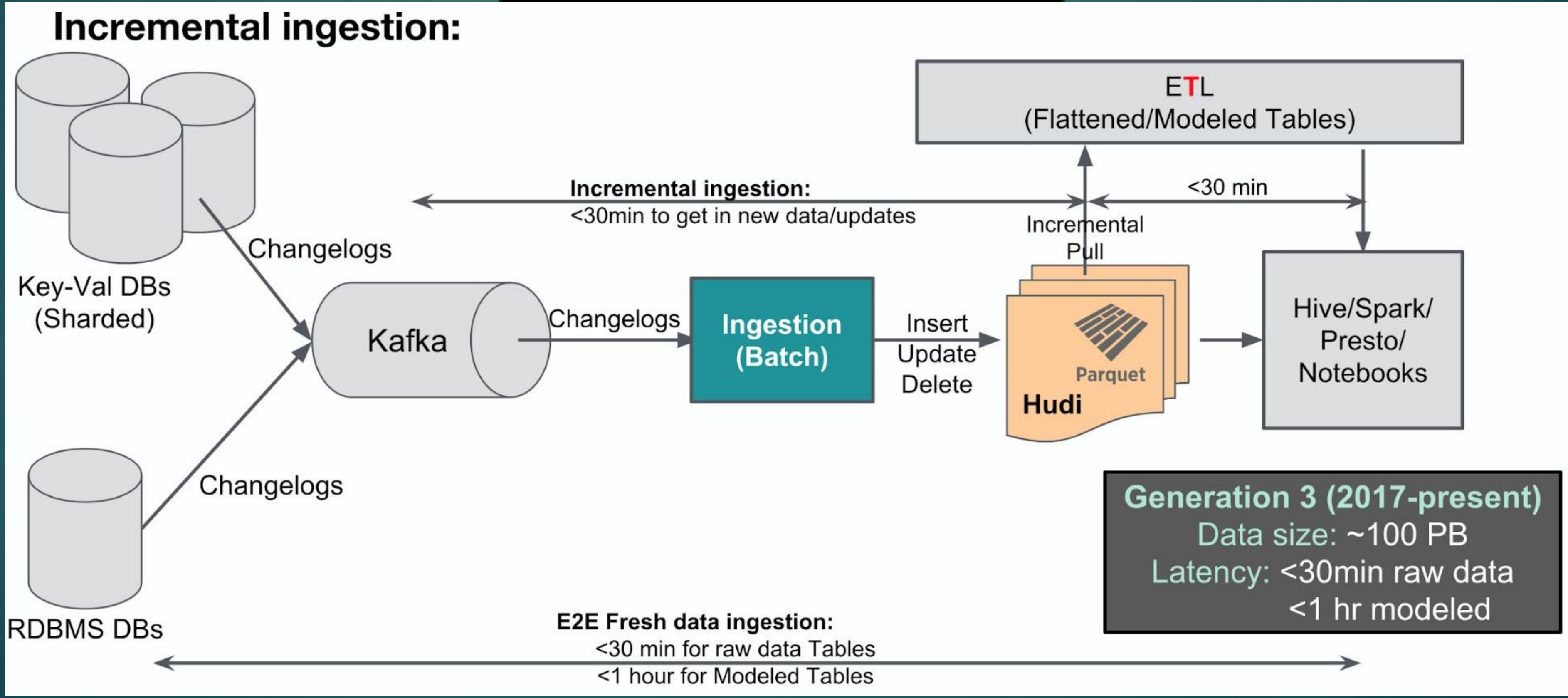
- **Data latency** remained a challenge, with new data becoming available only **every 24 hours**, hindering real-time decisions.
- ETL and modeling processes were still bottlenecks due to the need to **recreate entire datasets** during each run.
- **Snapshot-based ingestion** was required for data updates, which led to **over 1,000 Spark executors** running jobs that could take **20+ hours**.



The latency for new data was still over one day, a lag due to the snapshot-based ingestion of large, upstream source tables that take several hours to process.

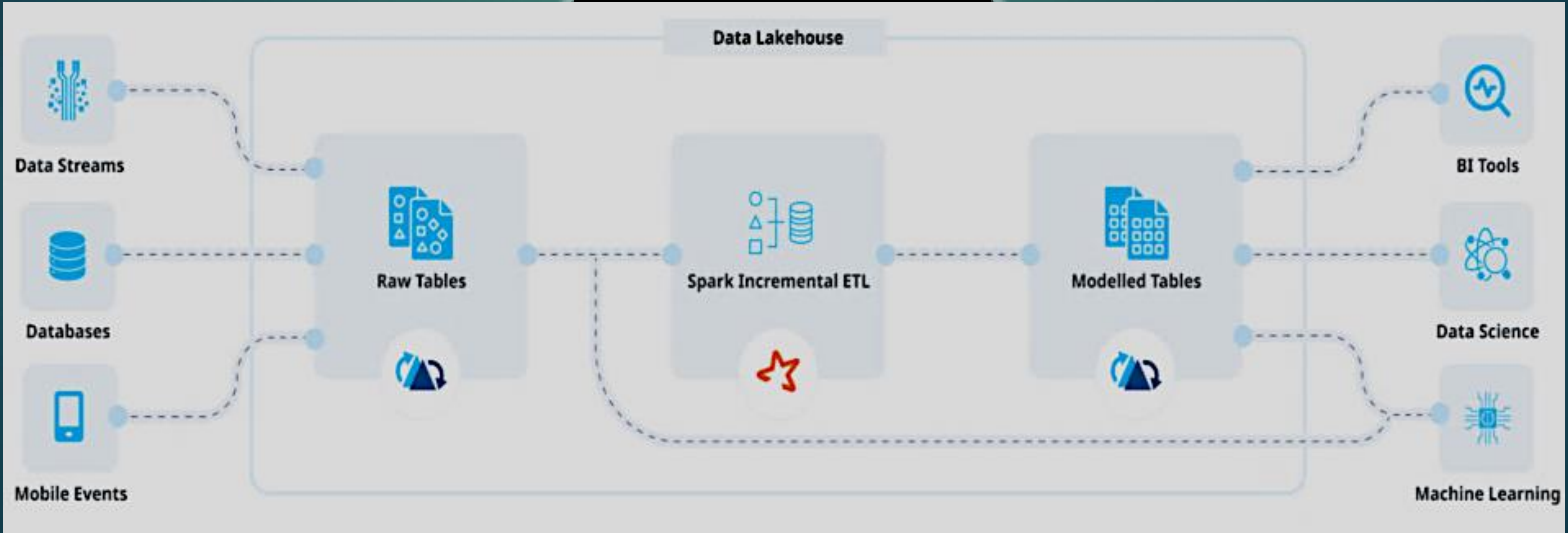
Rebuilding Big Data platform of Uber (2017- Present)

- **Hadoop and Parquet** lacked support for **updates and deletes**, requiring a solution for efficient data modifications.
- To address above issue, **Hudi (**Hadoop Upserts and Incremental) **is developed by Uber**, an open-source **Spark library** for **incremental ingestion**, updates, and deletes is used.
- With **Hudi**, data latency reduced from 24 hours to **less than one hour**, improving **ETL jobs** and enabling real-time analytics.
- The **Marmaray** platform ingests these events in mini-batches, applying changes to Hadoop using **Hudi** and also avoids inefficient transformations during raw data ingestion, using an **EL** (Extract-Load) model instead of traditional ETL.



- Uber uses incremental data processing to ensure data freshness and cost-efficiency at scale.
- **Apache Hudi**, supports incremental updates by handling late-arriving data seamlessly, Combines stream processing real-time data with the efficiency of batch processing.
- Incremental processing reduces the computational costs and duration of batch jobs, allowing for more frequent updates.

For example, Late-arriving updates, such as rider tips added after trips, are processed incrementally, avoiding recalculation of entire partitions.



- Incremental processing introduces mutable, database-like features in data lakes, enhancing performance and reliability.
- Data from various sources (e.g., Kafka, databases) is continuously ingested into raw datasets.
- Ingestion uses upserts to add or update data in raw datasets incrementally without overwriting entire partitions.

Incremental Pull and Joins:

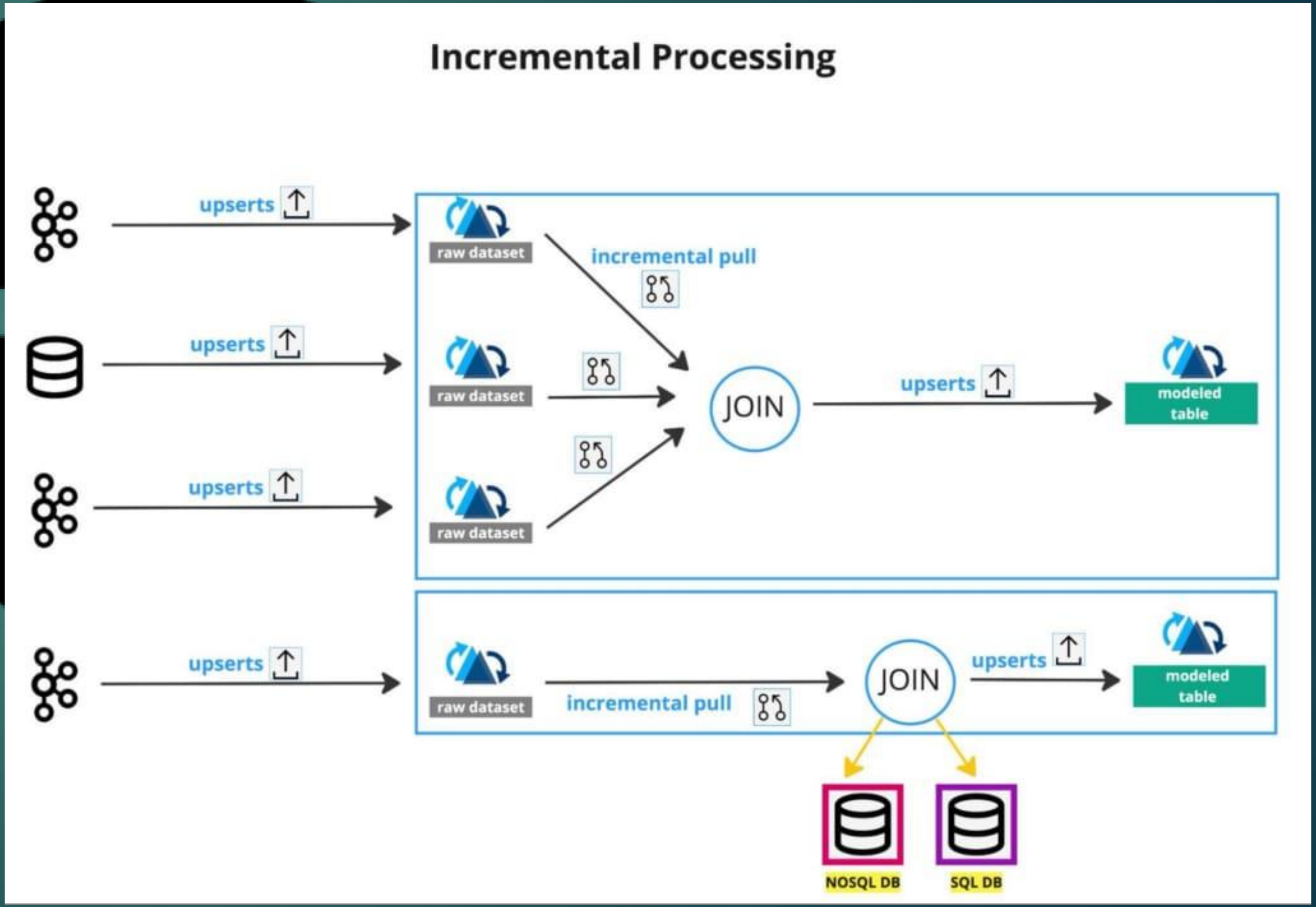
- Incremental pull operations fetch only the newly ingested or updated data from raw datasets.
- These incremental updates are combined using joins to integrate data from multiple raw datasets.

Modeled Table:

- Results are written to modeled tables using upserts.
- Modeled tables have processed and aggregated datasets for downstream applications.

Database Integration:

- Incremental data is also joined with external databases to enhance the modeled table with additional information in required workflows.



Performance and Cost Savings

Transitioning to Apache Hudi enabled strongly consistent replication by moving only incrementally changed files using Hudi metadata, avoiding inconsistencies with plain parquet tables.

- Incremental ETL pipelines reduced **pipeline run time by up to 82%**, achieving a **50% average reduction** compared to batch ETL.
- Significant cost reductions, with **up to 78.6% savings** on processing costs for dimensional driver tables.
- Memory and CPU usage improved by **73% and 59%, respectively** for key pipelines.

Pipeline	vcore_seconds	memory_seconds	Cost	Run Time (mins)
Batch ETL of Dimensional Driver Table	3,129,130	23,815,200	$11.39	220
Incremental ETL of Dimensional Driver Table	1,280,928	6,427,500	$2.44	39
Difference	1,848,202	17,387,700	$8.95	181
% Improvement	**59.06%**	**73.01%**	**78.57%**	**82.27%**
Batch ETL of Driver Status Fact Table	2,162,362	5,658,785	$3.30	94
Incremental ETL of Driver Status Fact Table	1,640,438	3,862,490	$2.45	48
Difference	521,924	1,796	$0.85	46
% Improvement	24.13%	31.74%	25.75%	48.93%

Big Data Prospects and Developments in Uber

- Uber is migrating its batch data analytics and ML training stack to Google Cloud Platform (GCP) to enhance productivity, cost efficiency, and data governance.
- The strategy involves initially replicating the on-prem stack on GCP's IaaS, with plans to adopt PaaS offerings like Dataproc and BigQuery for scalability and performance.
- The migration strategy leverages cloud's object storage with HDFS compatibility and translating HDFS paths to object store paths via a "Path Translation Service."
- Uber's existing container and deployment systems, built to work across on-prem and cloud, support the expansion of batch data ecosystem microservices to GCP IaaS without major modifications.
- Extending HiveSync to support active-active bi-directional replication and ongoing updates to the cloud-based data lake.

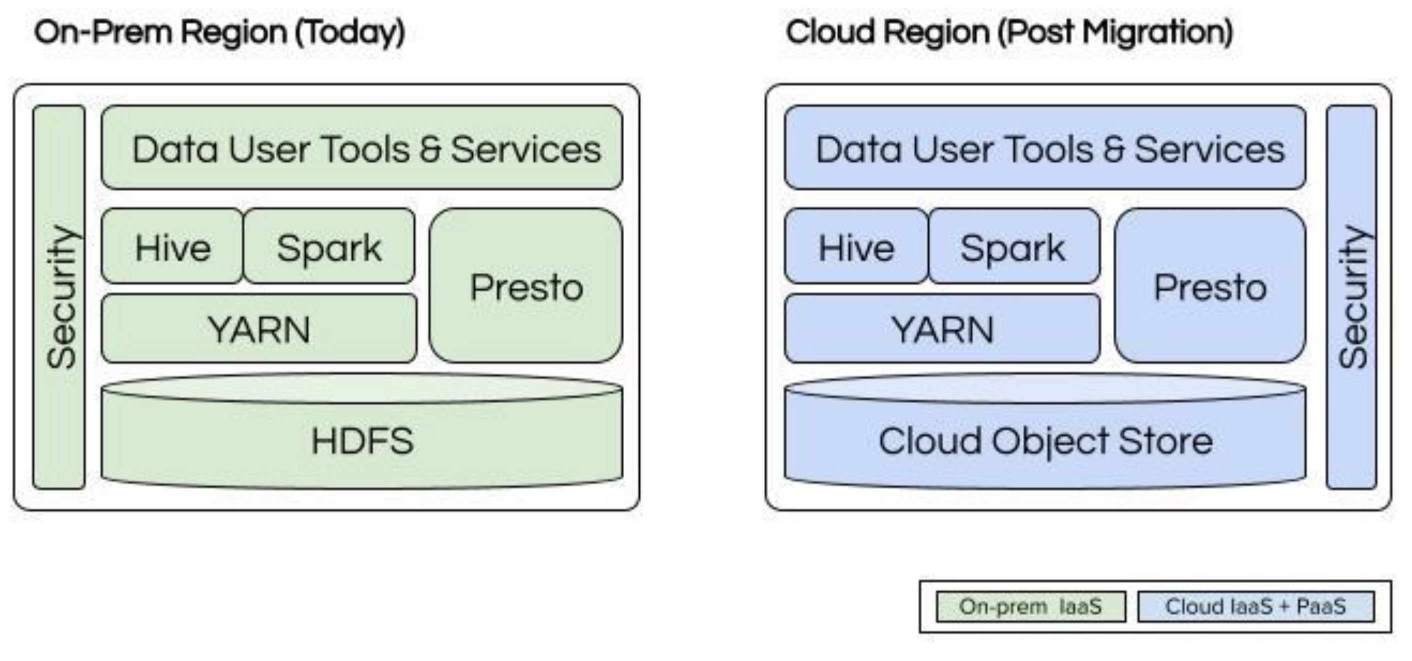


Figure 2: Uber Batch Data Stack: Pre and Post Migration

References

- Uber's Big Data Platform: 100+ Petabytes with Minute Latency

(https://www.uber.com/blog/uber-big-data-platform/)

- Cost-Efficient Open Source Big Data Platform at Uber

(https://www.uber.com/blog/cost-efficient-big-data-platform/)

- Setting Uber's Transactional Data Lake in Motion with Incremental ETL Using Apache Hudi

(https://www.uber.com/blog/ubers-lakehouse-architecture/)

- Modernizing Uber's Batch Data Infrastructure with Google Cloud Platform

(https://www.uber.com/blog/modernizing-ubers-data-infrastructure-with-gcp/#:~:text=Uber%20runs%20one%20of%20the%20largest%20Hadoop%20installations%20in%20the%20world.&text=We%20will%20leverage%20open%20standards%20such%20as,the%20teams%20within%20the%20data%20platform%20organization.)

- DataMesh: How Uber laid the foundations for the data lake cloud migration

(https://www.uber.com/blog/datamesh/)

- DataCentral: Uber's Big Data Observability and Chargeback Platform

(https://www.uber.com/blog/datacentral-ubers-observability-and-chargeback-platform/?uclick_id=bb4c38b1-fe21-44a3-be95-f97578d04874)