Creating an Extensible Big Data Platform

Reza Shiftehfar
Hadoop Platform team
reza@uber.com

September 2018

Who am I

Reza Shiftehfar

- PhD in Computer Science from University of Illinois @Urbana-Champaign
- Working at Uber since 2014
- Founding engineer of the data platform team at Uber
- Currently managing the Hadoop Platform team at Uber
- Helped scale Uber's data from a few TB to 100+ PB
- Helped lower data latency from 24+ hrs to minutes

Agenda

1. **Intro to Data @ Uber**
2. **Data Platform - Past**
    - Traditional Big Data Platform
3. **Data Platform - Present**
    - Redesigned extensible Big Data Platform
4. **Data Platform - Future**
    - What’s coming next?
5. **Lessons learned**

Intro to Data @ Uber:

Uber's Mission

“Transportation as reliable as running water, everywhere, for everyone”

700+ Cities

75+ Countries

2M+ Driver Partners

And Growing...

The Impact of Data @ Uber

1. **City Operators (~1000s)**
    - On the ground team who run and scale uber's transportation network in each city

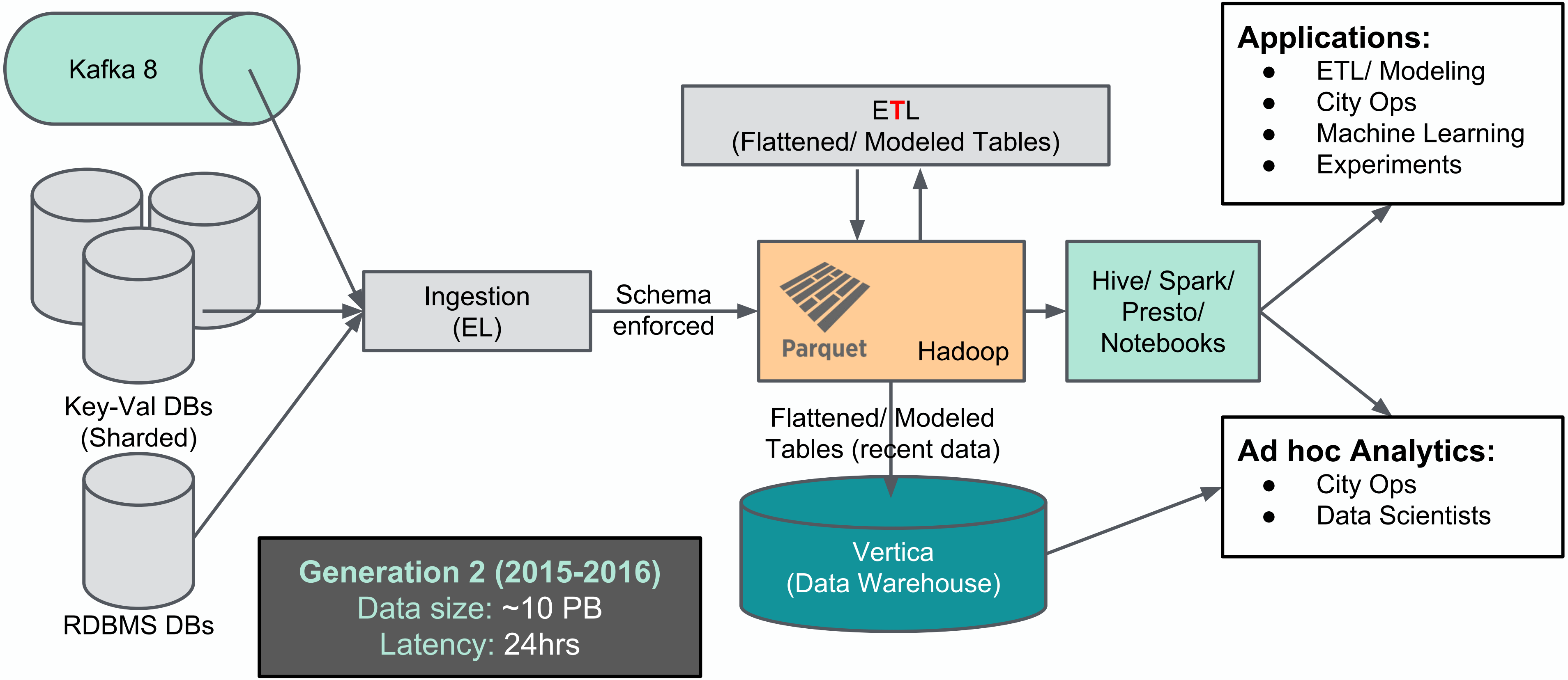
2. **Data Scientists and Analysts (~100s)**
    - Spread across various functional groups (e.g. Marketing Spend, Forecasting demand)

3. **Engineering Teams (~100s)**
    - Focused on building automated data applications (Fraud Detection, Incentive Payments, Driver onboarding,...)

UBER

Past:
Traditional Big Data Platform
(2016)

Past: Traditional Big Data Platform (2015-2016)



Past: Traditional Big Data Platform (2015-2016)

Highlights:

- All raw data is stored in Hadoop Data Lake
- Data stored as Columnar Parquet format
    - More efficient storage
    - More efficient queries
- All ETL/Modeling happens in Hadoop
- Subset of data transferred to warehouse
    - Only flattened selected recent dates
- Presto added as interactive query engine
- Spark notebooks added to encourage data scientists to use Hadoop

Past: Traditional Big Data Platform (2015-2016)

Big Wins:

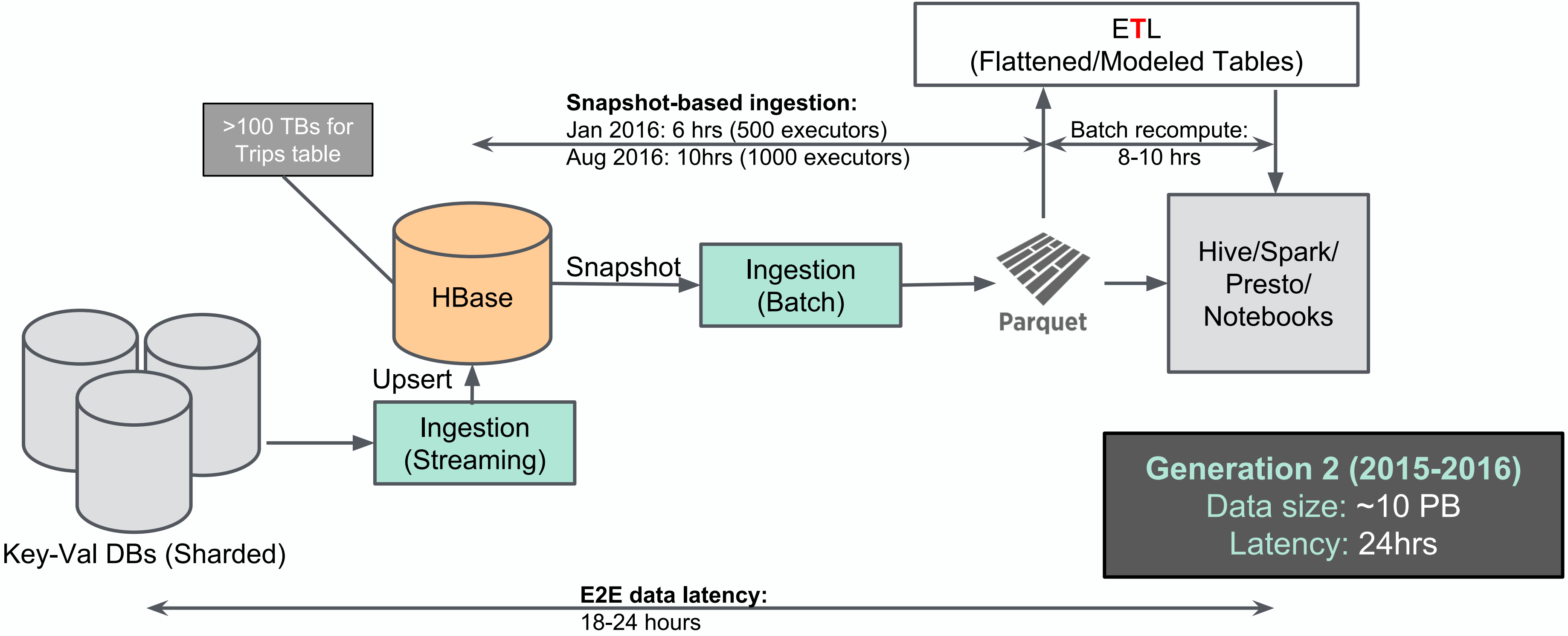
- Hadoop became the source-of-truth for all data
  - 100% of All analytical data in one place
- Hadoop powered critical Business Operations
  - Partner Incentive Payments, Fraud
- Unlocked the real power of data
- Gave us time to stabilize the infrastructure (Kafka,....) & think long-term

Some Numbers (early 2016):

- **~10 PB in HDFS**
- **~10 TB/day new data**
- **~10k vcores**
- **~100k daily batch jobs**
- **And growing...**

Past: Traditional Big Data Platform (2015-2016)

Why does data latency remain at 24 hours?



Past: Traditional Big Data Platform (2015-2016)

Problems/ Limitations:

Pain Point #1: Scalability:

- Too many small files in HDFS (required async stitcher)
- Source-specific data ingestion pipelines increased maintenance cost

Pain Point #2: Data Latency too high:

- snapshot based ingestion results in 24hrs data latency

Pain Point #3: Updates became a big problem:

- Updates/late-arriving-data are natural part of our data

Pain Point #4: ETL/ Modeling became the bottleneck:

- ETL/Modeling was snapshot based (running daily off raw tables)

UBER

Present:
Redesigned extensible Big Data Platform

(2017-present)

Present: Redesigned extensible Big Data Platform (2017-present)

Some Numbers (early 2017):

- ~100+ PB in HDFS data
- ~100k vcores
- ~100k Presto queries/day
- ~1000+ Spark apps/day
- ~20k Hive queries/day
- And still growing...

Present: Redesigned extensible Big Data Platform (2017-present)

Motivation for rebuilding:

- Interactive Query engines -> Hadoop data extremely popular
- No more fire-fighting -> allowed study of our real needs
- **Let's build for long-term** (Generation 3 of our Big Data Platform)

Problems to solve:

- **Pain Point #1:** HDFS Scalability
  - Namenode will always be the bottleneck
  - Small files are the killer
  - Benefit from ViewFS and Federation to scale
    - Controlling small files and moving part of data to a separate cluster (e.g. HBase, Yarn app logs) can let you get to 100+ PB
    - See our recent [Engineering Blog post]() on this

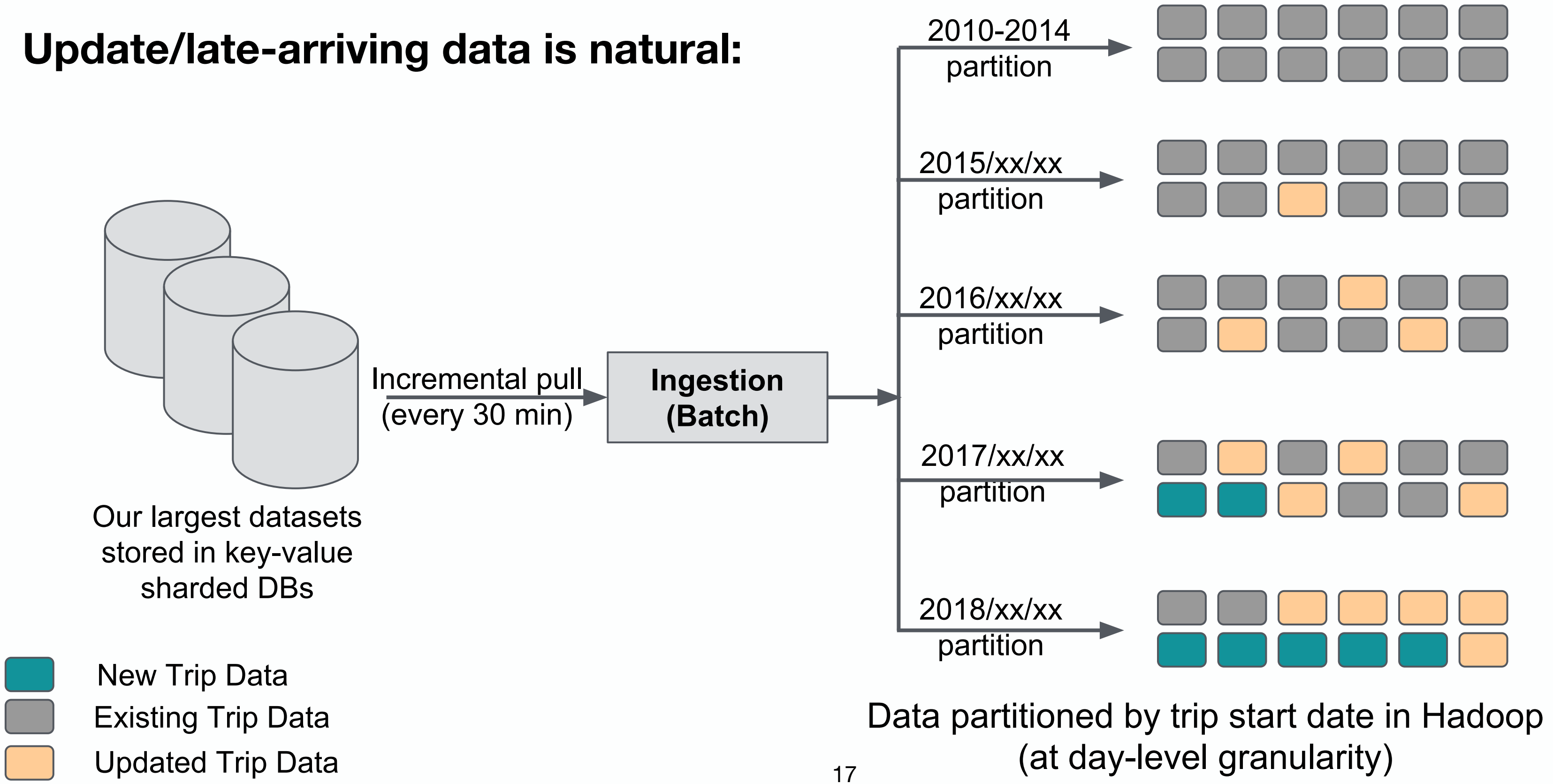
Present: Redesigned extensible Big Data Platform (2017-present)

Problems to solve:

- **Pain Point #2:** Faster data in Hadoop
    - Need fully incremental ingestion of data
- **Pain Point #3:** Support for Updates/Deletes in Hadoop/Parquet
    - Need to support Update/Deletion during ingestion of incremental changelogs
        - Our data has large number of columns with nested data support -> Parquet stays
- **Pain Point #4:** Faster ETL/ Modeling
    - ETL has to become incremental too
    - Need to allow users to pull out only changes incrementally
    - Have to support all different query engines (Hive, Presto, Spark,...)

Present: Redesigned extensible Big Data Platform (2017-present)

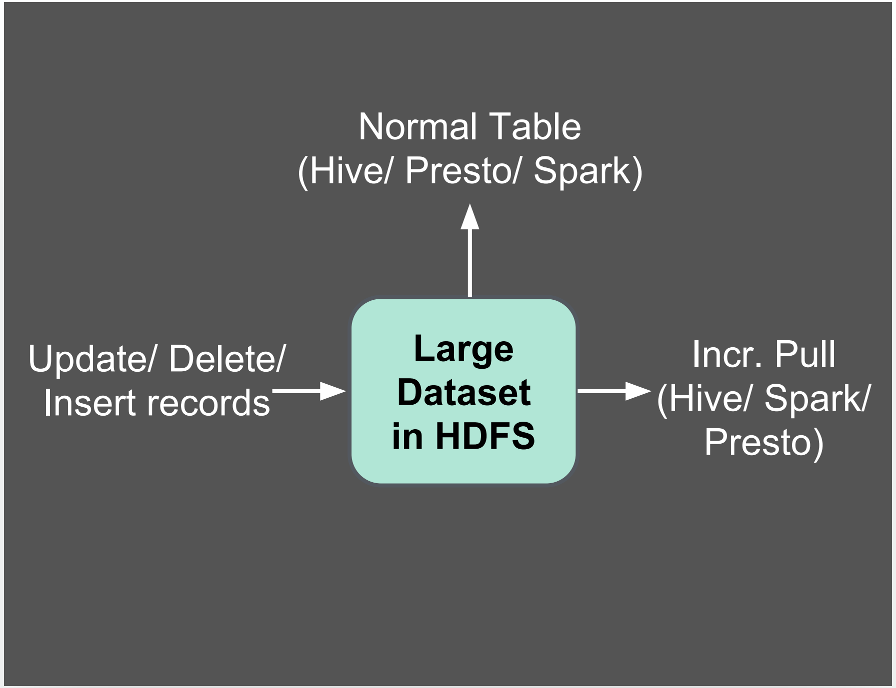
Update/late-arriving data is natural:



Present: Redesigned extensible Big Data Platform (2017-present)

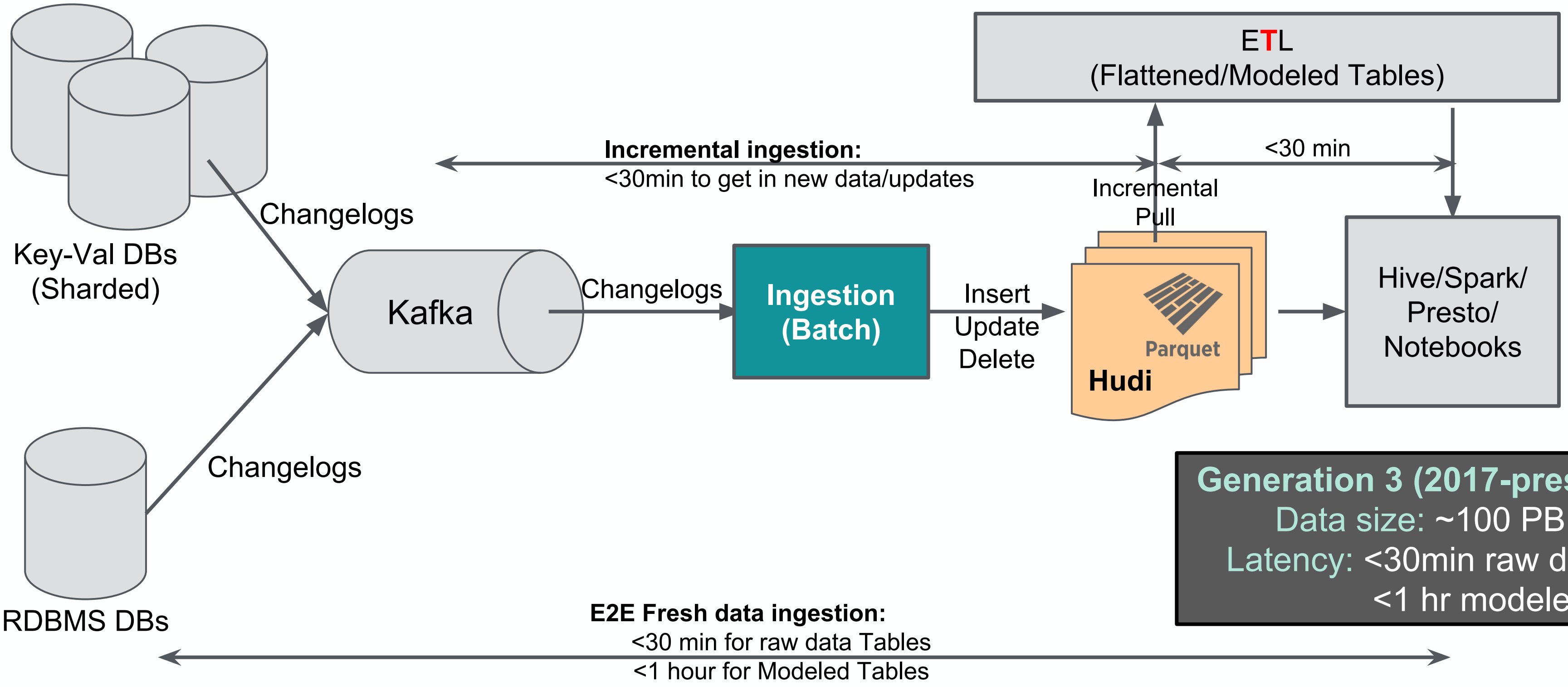
What did we build to address these needs?

- Built **Hudi**: **H**adoop **U**pserts an**D** **I**ncremental
- Storage abstraction to:
  - Apply upsert/delete on existing Parquet data in Hadoop
  - Pull out changed data incrementally
- Spark based library:
  - Scales horizontally like any Spark job
  - Only relies on HDFS
- It is open-sourced (Hudi on Github) (Hudi Eng Blog)



Present: Redesigned extensible Big Data Platform (2017-present)

Incremental ingestion:

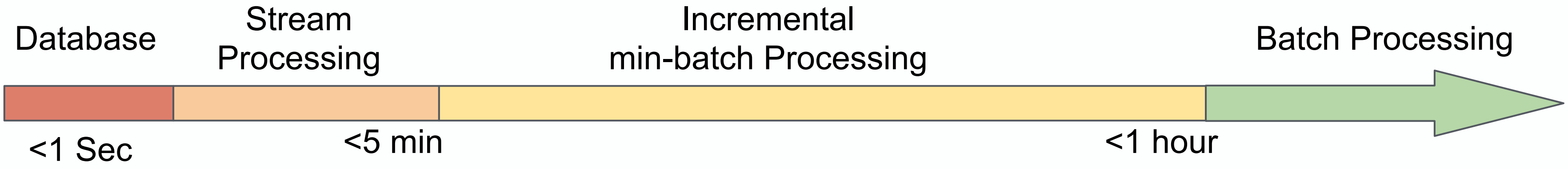


Generation 3 (2017-present)
Data size: ~100 PB
Latency: <30min raw data
<1 hr modeled

Present: Redesigned extensible Big Data Platform (2017-present)

What is Incremental Processing:

- Traditional λ architecture provides: Streaming vs Batch solutions
    - That assumes append-only immutable data
    - Processing based on timestamp (usually skips late-arriving data)
- Incremental Processing is mini-batch jobs that pulls out only changed data
    - This gets you all the recently appended data as well as old changed/updated records
    - Provides high completeness (compared to streaming mode)
    - Processing no longer limited by updates/deletes or late-arriving data
    - Is a batch job and supports full batch functionality (e.g. joins,....)

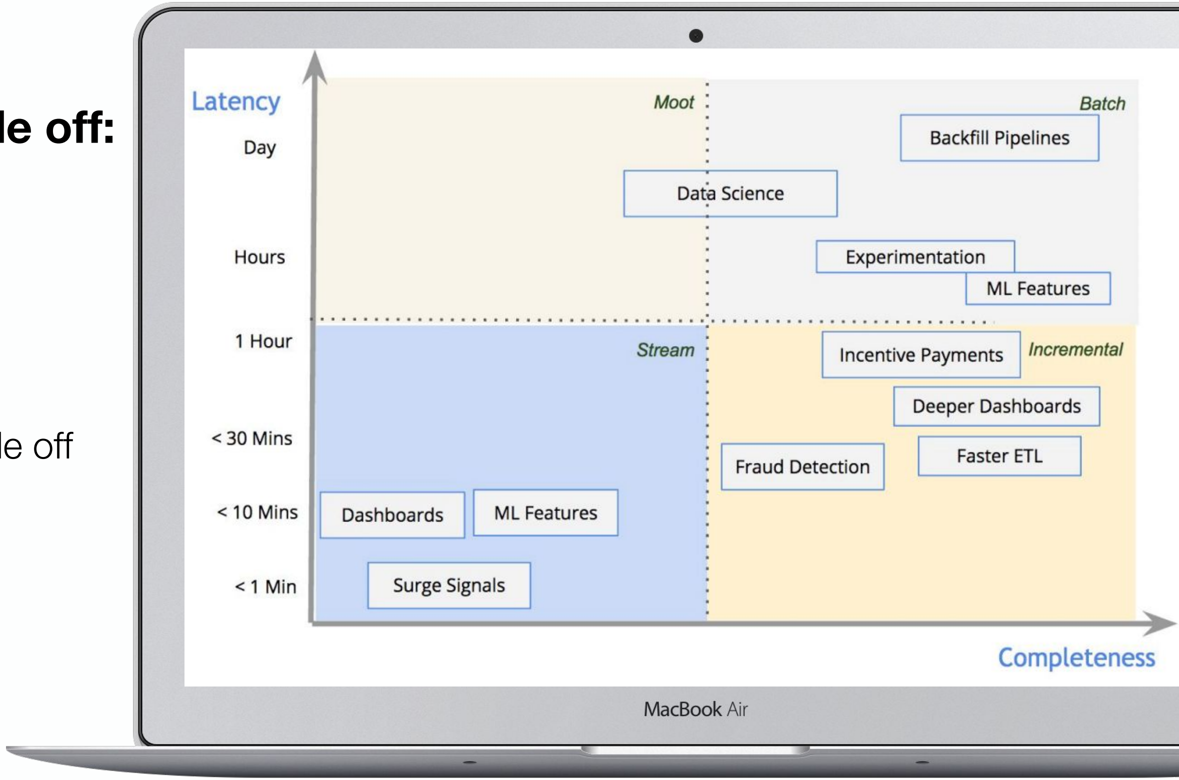


Present: Redesigned extensible Big Data Platform (2017-present)

Stream/Batch processing Trade off:

- Latency
- Completeness
- Cost (Throughput/efficiency)

Study your use case based on these trade off

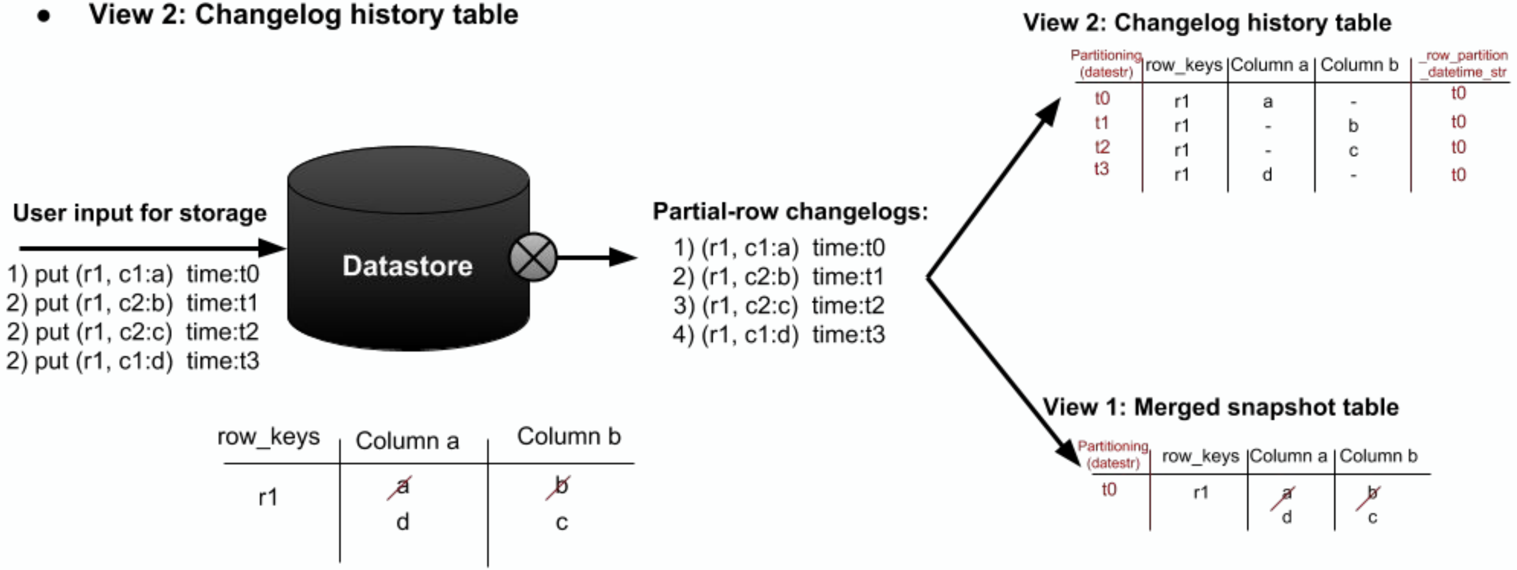


Present: Redesigned extensible Big Data Platform (2017-present)

Other Improvements: Standardized data model

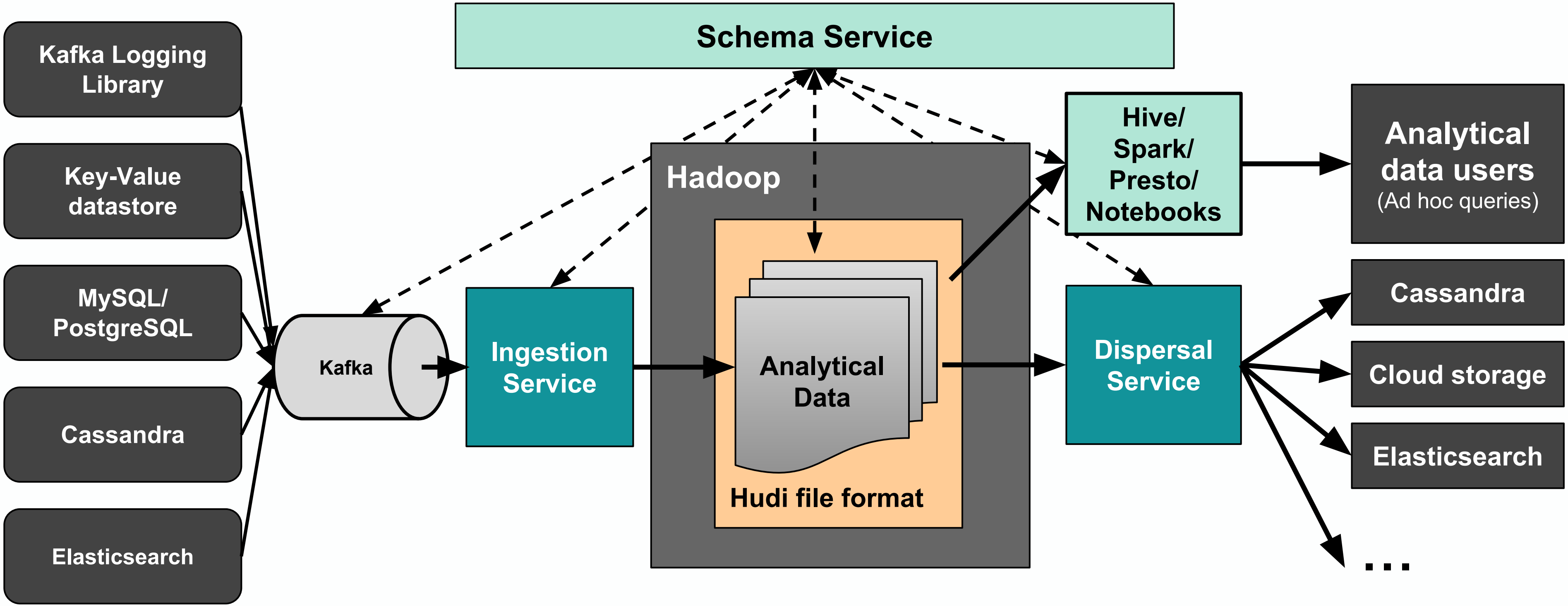
Standardized Hive raw data model:

- **View 1: Merged snapshot table**
- **View 2: Changelog history table**



Present: Redesigned extensible Big Data Platform (2017-present)

Other Improvements: Generic Any-to-Any Data platform



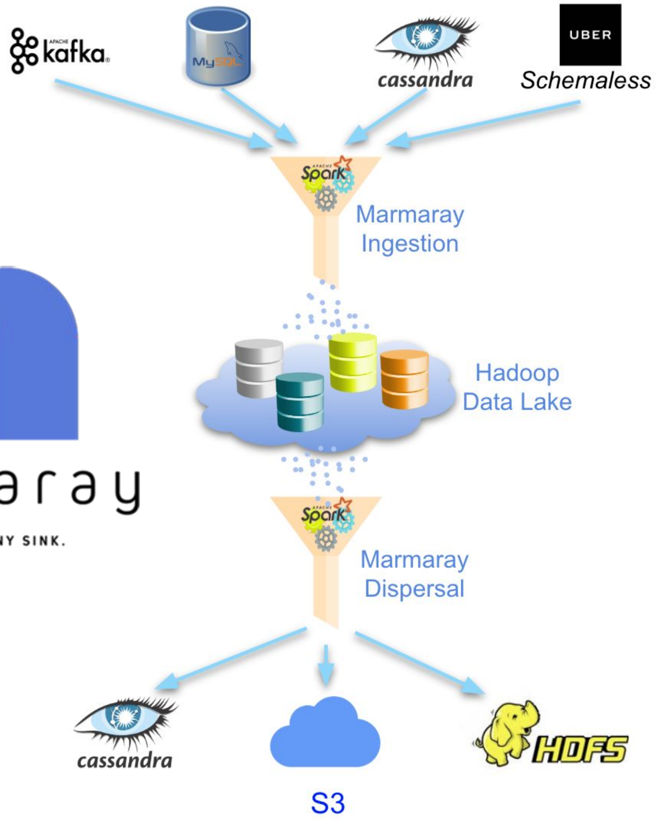
Present: Redesigned extensible Big Data Platform (2017-present)

Other Improvements: Generic Any-to-Any Data platform

- Built **Marmaray**:
    - Both Ingestion & Dispersal Framework/Library
    - Generic Any Source to Any Sink
- Spark based:
    - Scales horizontally like any Spark job
    - Sources and Sinks can easily be extended
- It is open-sourced ([Marmaray on Github]) ([Marmaray Eng Blog])

marmaray

ANY SOURCE, ANY SINK.



Future:
What’s coming next?

(Ongoing effort)

What’s coming next - Generation 4 (Ongoing effort)

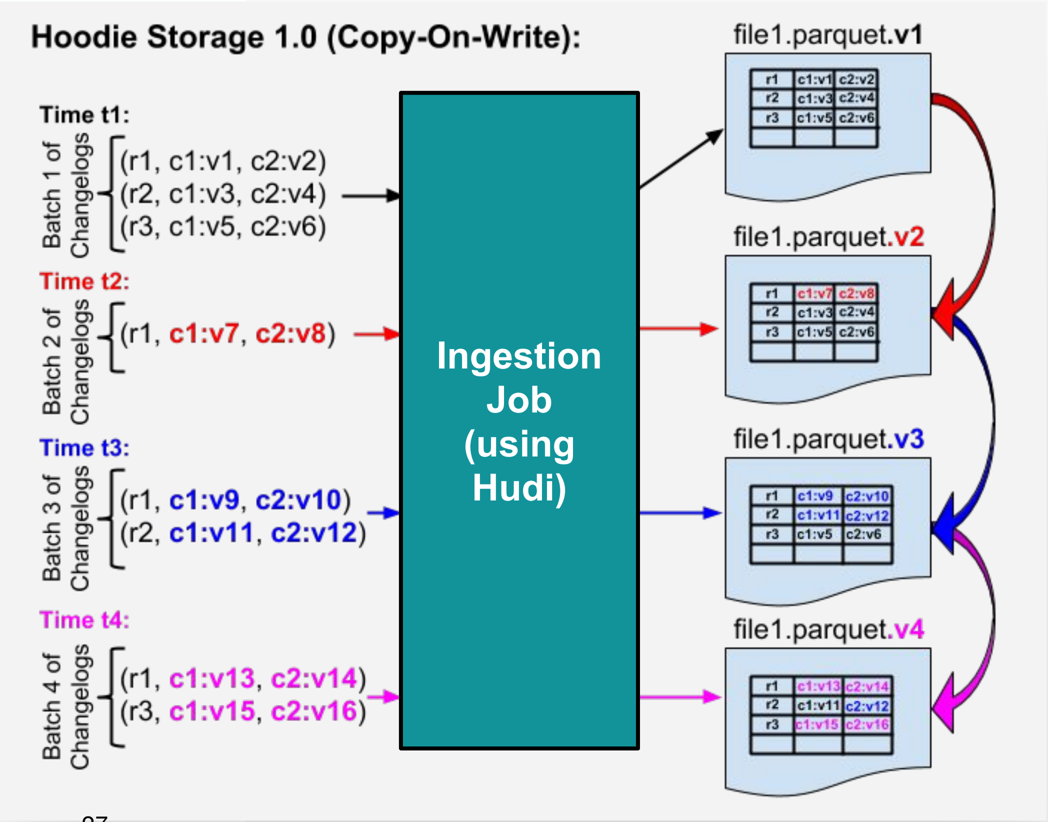
Are we done? Any remaining items?

1. **Data Quality is still a concern:**
   - Further unification of Hadoop Ingestion with strict contract with Storage team
   - Expand schema-service beyond type/structural check and into semantic checks
   - Unify RPC vs Analytical worlds (especially on data schema side)
2. **Still Need faster data access**
   - ~5-10 min Hadoop data for mini-batching to compete with Streaming
3. **Efficiency is the next big monster**
   - Don’t limit yourself to Hadoop. Go for the entire compute resources
   - Unified resource scheduler for Hadoop and beyond (Mesos, Yarn and now Peloton)
   - See our presentation at “Hadoop Infrastructure@Uber Past , Present and Future”
4. **Hudi is still actively being developed**
   - Get rid of sensitivity with respect to the ratio of update/delete vs insert
   - Provide large Parquet file (1+ GB) with data latency of 5-10min

What's coming next - Generation 4 (Ongoing effort)

Hudi Storage 1.0:

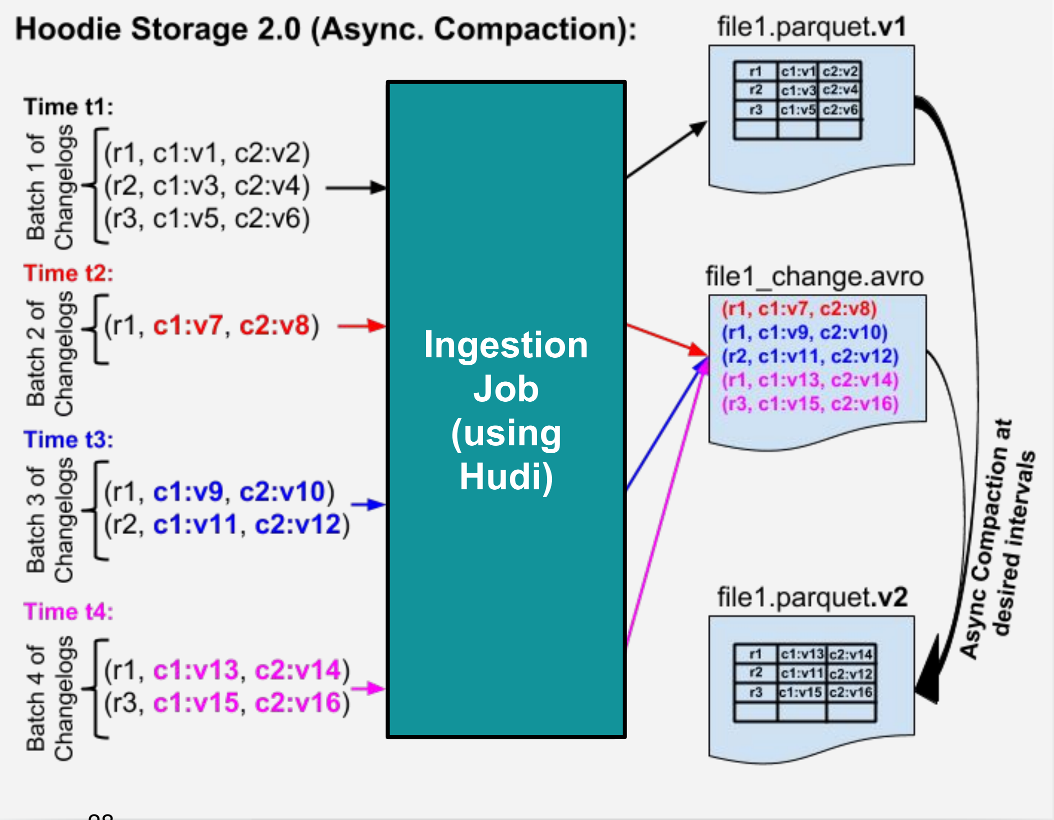
- Copy-on-write solution
- Rewriting Parquet files on updates/deletes
    - 1GB file very expensive
- Output Partition + Row_Key are required
    - Supports per partition index
    - Can we get rid of output partition?



What's coming next - Generation 4 (Ongoing effort)

Hudi Storage 2.0:

- Merge-on-Read solution
- Have row-based delta file + Parquet file
    - Merge only when the cost of rewrite is amortized
- Merge on Query side
    - Provides 5-10min hadoop data
- Add Global Index

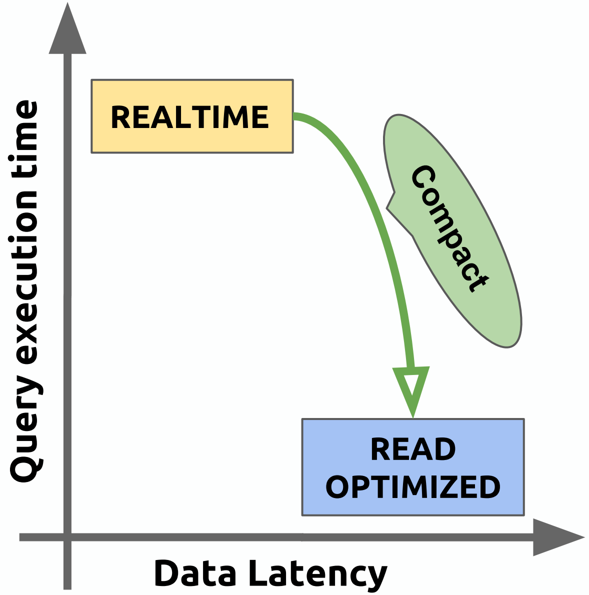


What's coming next - Generation 4 (Ongoing effort)

Be flexible with users:

- Hudi's supported different Storage Types and Views

Storage Type	Supported Views
Storage 1.0 (Copy On Write)	Read Optimized, ChangeLog View
Storage 2.0 (Merge On Read)	Read Optimized, RealTime, ChangeLog View



UBER

Creating an Extensible Big Data Platform: Lessons learned

Creating an Extensible Big Data Platform: Lessons Learned

1. **Investigating your data/use cases and <u>finding the required primitives</u> pays back huge**
    - With <u>GDPR requirement</u>, Having Update/Delete on the entire Hadoop dataset is life-saving
2. **<u>Data Quality</u> will be an ongoing effort**
    - This is the key distinction between a <u>data swamp</u> and an <u>effective data lake</u>
    - <u>Enforce schema</u> (mandatory and pre-defined) as early as possible
    - Move beyond type checking and into <u>semantic checking</u>
    - Use "<u>enumerated values"</u> instead of Strings as much as possible
    - Enforce <u>mandatory documentation</u> for all fields
    - <u>Standardize schema name</u>, <u>field names</u> as well as define your <u>core entities as types</u>
3. **<u>Standardize</u> everything as soon as possible**
    - Don't make <u>exceptions</u> (it always comes back at you)
    - This is the key to having reliable Big data that can scale while being efficient
    - This is the key to have happy data users and to be able to educate them on how to use your data

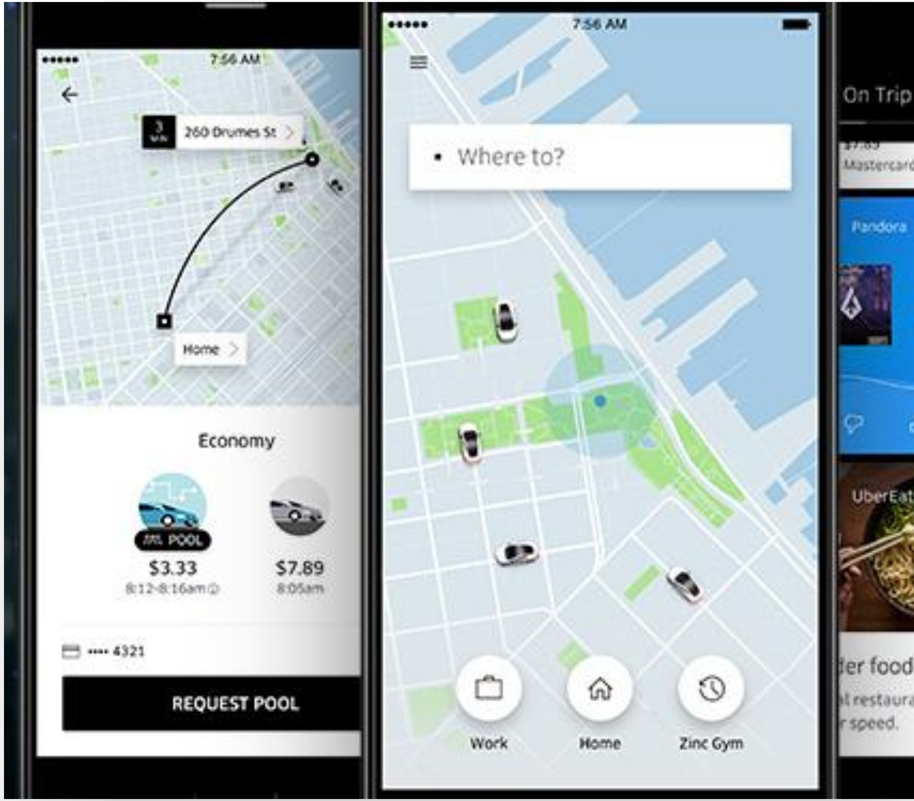
Creating an Extensible Big Data Platform: Lessons Learned

4. **Ensure you have a solid data <u>retention policy</u> as well as a standard <u>data model</u> as early as possible**
    - Retention from beginning saves you $ on wasted space and educates users on not wasting
5. **Track all related data <u>metadata</u>**
    - Who owns what data, data lineage, data content, data access,...
6. **Invest in a good <u>data pipeline monitoring</u>**
    - Define your terminology and stick to it (<u>Freshness</u>, <u>Latency</u>, <u>Completeness</u>, Late-arriving-data,...)
    - Detects many corner cases and lets you solve the issue before it affects your users
7. **Minimize your platform dependency on <u>user-defined values</u>**
    - User-defined values always break your Big data platform
    - Replace them by <u>system-defined values</u> as much as possible (e.g. user define ts vs system ts)
8. **Pay attention to the <u>notion of time in your data</u> and educate users on those**

Hadoop Platform @ Uber

Want to be part of our future effort?

- **Come talk to me**
    - Office Hours: 11:45am - 12:30 pm
- **Positions in both SF & Palo Alto**
    - email me: reza@uber.com



Creating an Extensible Big Data Platform

UBER

reza@uber.com

Further references

1. “Hadoop Data Journey @ Uber”, Reza Shiftehfar, Data Eng Conference in San Francisco, 2018
2. Open-Source Hudi Project on Github
3. “Hudi: Uber Engineering’s Incremental Processing Framework on Hadoop”, Prasanna Rajaperumal, Vinoth Chandar, Uber Eng blog, 2017
4. Open-Source Marmaray on Github
5. Open-Source Marmaray Project on Github
6. “Marmaray: An Open Source Generic Data Ingestion and Dispersal Framework and Library for Apache Hadoop”, Danny Chen, Omkar Joshi, Uber Eng blog, 2018
7. “Uber, your Hadoop has arrived: Powering Intelligence for Uber’s Real-time marketplace”, Vinoth Chandar, Strata + Hadoop, 2016.
8. “Case For Incremental Processing on Hadoop”, Vinoth Chandar, O’Reily article, 2016
9. “Hudi: Incremental processing on Hadoop at Uber”, Vinoth Chandar, Prasanna Rajaperumal, Strata + Hadoop World, 2017.

Further references

9. "Hudi: An Open Source Incremental Processing Framework From Uber", Vinoth Chandar, DataEngConf, 2017.
10. "Incremental Processing on Large Analytical Datasets", Prasanna Rajaperumal, Spark Summit, 2017.
11. "Scaling Uber's Hadoop Distributed File System for Growth", Ang Zhang, Wei Yan, Uber Eng blog, 2018
12. "Hadoop Infrastructure @Uber Past, Present and Future", Mayank Bansal, Apache Big Data Europe , 2016.
13. "Even Faster: When Presto Meets Parquet @ Uber", Zhenxiao Luo, Apache: Big Data North America, 2017.

UBER

Extra slides

Let’s rebuild for long term - Generation 3 (2017-present)

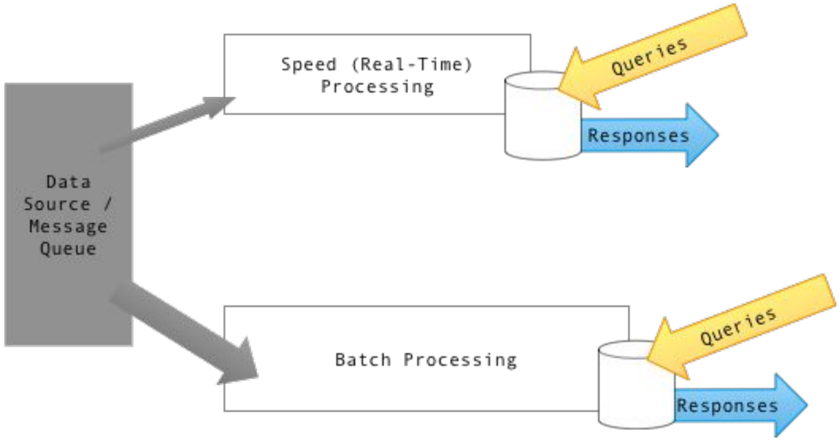
Any work-around for snapshot-based ingestion?

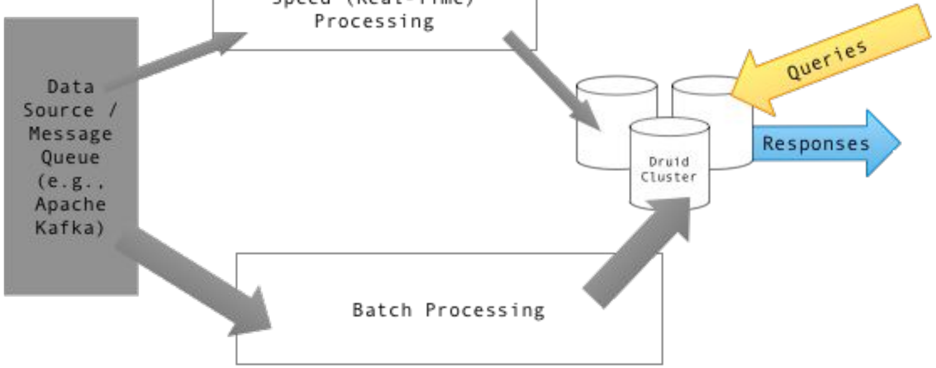
1. **Directly Query HBase**
    - Range scan will make it a bad fit
    - Lack of support for nested data
    - Significant operational overhead for 100 PB
2. **Don’t support Snapshot view and only provide logs**
    - Users need the merged view and will have to do it in their queries which makes it inefficient
    - Merging can be done inconsistency resulting in data correctness
3. **Use specialized analytical DBs**
    - Can’t bypass HDFS since we still need to join with other data in HDFS
    - Not all data fits into memory and many queries will fail
    - Leads to lambda architecture issue and multiple copies of the same data

Data @ Uber: Generation 3

What does Incremental Processing mean:

Lambda architecture:





Data @ Uber: Generation 3

Stream/Batch processing Trade off:

- Latency
- Completeness
- Cost (Throughput/efficiency)

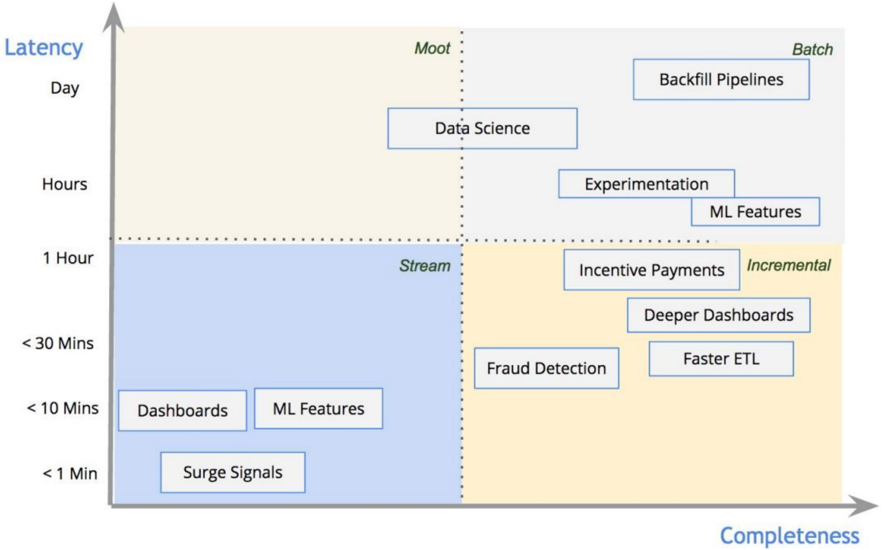
Operation challenges in Streaming & Batch:

- Projections (Streaming:Easy   Batch:Easy)
- Filtering (Streaming:Easy   Batch:Easy)
- Aggregations (Streaming:Tricky   Batch:Easy)
- Window (Streaming:Tricky   Batch:Easy)
- Joins (Streaming:HARD   Batch:Easy)

Data @ Uber: Generation 3

Do we need Streaming, Batch or Incremental?

- Need to investigate your use cases (based on latency vs Completeness)

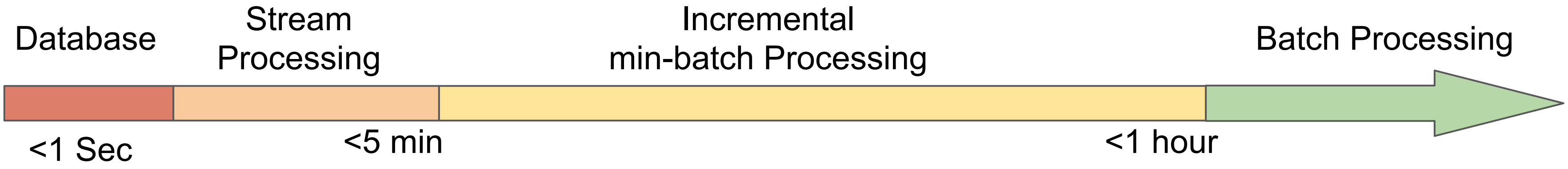


- Very distinct uses cases for Streaming
- Very distinct use cases for Batch
- A lot of use cases that can benefit from incremental mode

Data @ Uber: Generation 3: Provide Incremental processing

What exactly is Incremental mode?

- Mini-batch jobs that pulls out only changed data
- Provides high completeness (compared to streaming mode)
- Supports all hard operations as any other batch job (like multi-table joins,....)



Data @ Uber: Generation 3: Provide Incremental processing

How does Incremental mode help efficiency?

- Read only what you need by using Columnar file formats
- Simple solution for all types of queries (joins, …)
- Consolidation of Compute & Storage for all use case (exploratory, interactive,....)