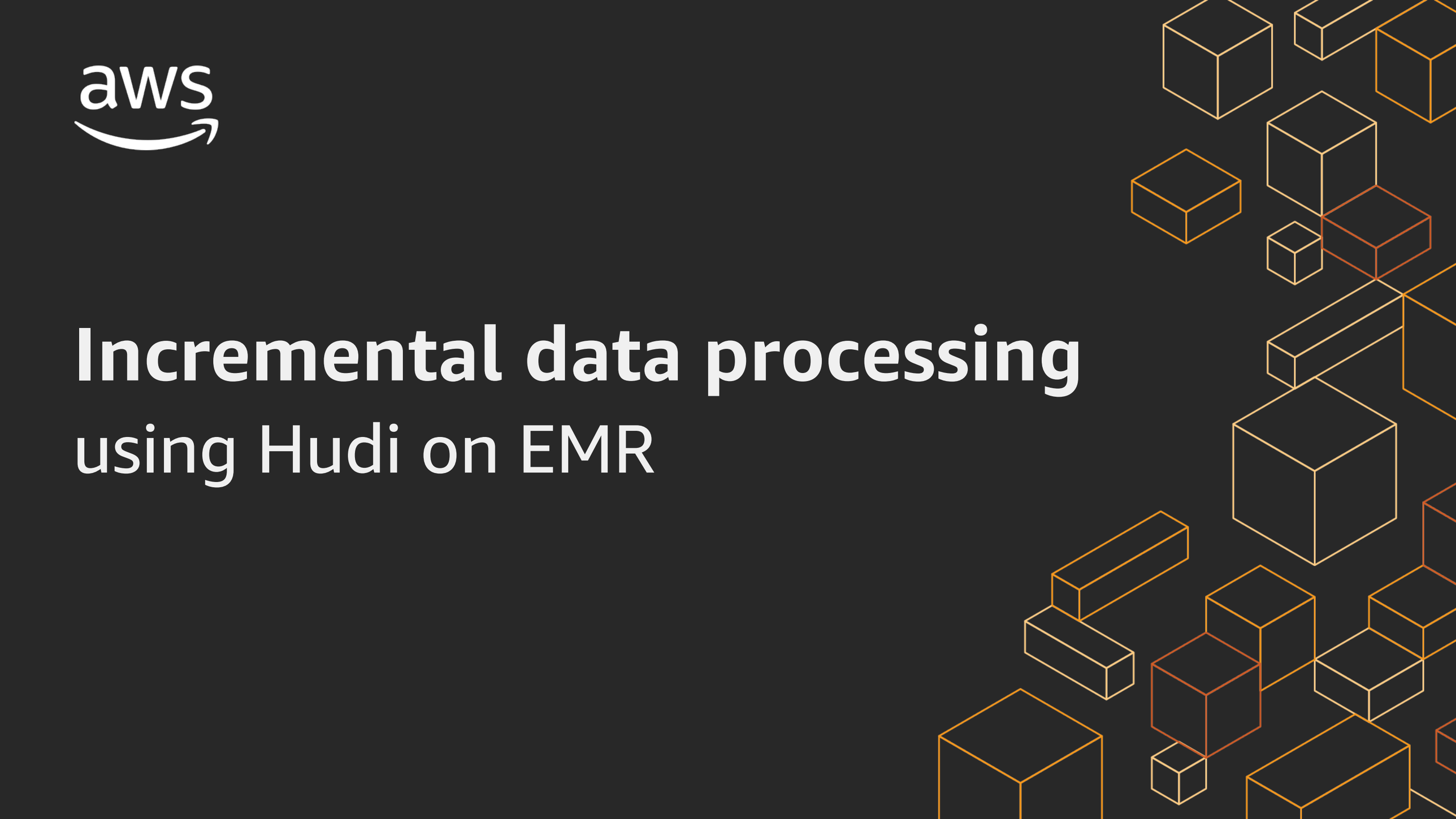
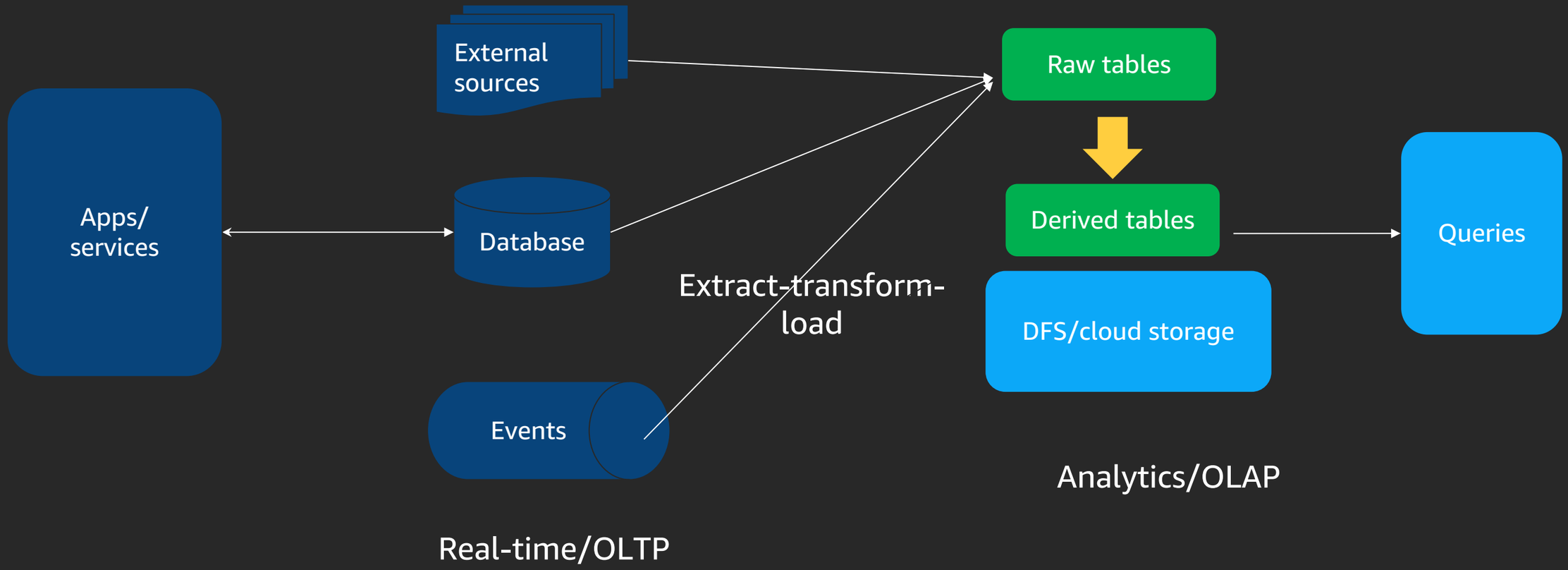


Incremental data processing

using Hudi on EMR

Typical lakes



The problem

1. Latest snapshot of data
2. Historical view

Source database

Data lake

Order ID	Quantity	Date
001	10	01/01/2019
001	15	01/02/2019
002	20	01/01/2019
~~002~~	~~20~~	~~01/02/2019~~

Action	Order ID	Quantity	Date
I	001	10	01/01/2019
U	001	15	01/02/2019
I	002	20	01/01/2019
D	002	20	01/02/2019

3. Incremental processing
4. Low end-to-end SLA

Why incremental processing in a data lake?

Motivating use cases

Change data capture (CDC)

GDPR (data erasure)

De-duplication

Enforce minimum file size on HDFS

Time travel

Apache Hudi

Data deletions: Privacy regulation

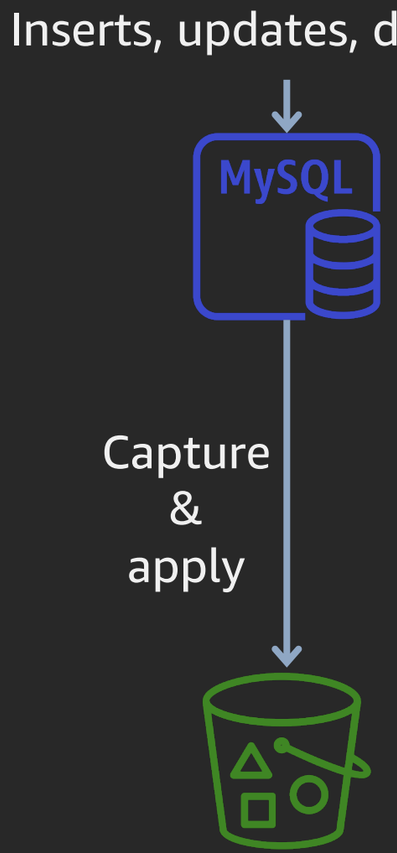
Enforcing data privacy laws

- Delete data within a specific time frame
- Delete data across all data sets
- Identifying files & partitions with specific data
- Lack of indexes on data lake storage

Change Data Capture & apply

Applying change streams to Amazon Simple Storage Service (Amazon S3) data

- Bulk loads don’t scale
- No support for upserts
- Data quality is a serious concern
- Need similar guarantees as a database



Users

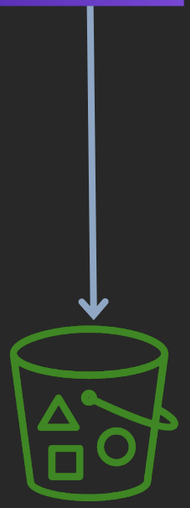
Column	Type
userID	int
country	string
last_modified	long
…	…

Streaming data ingestion

High-volume, time-ordered data

- Duplicate events mess up analytics
- Need fast ingestion to Amazon S3
- Schema management, checkpointing

Produce events in Apache Kafka

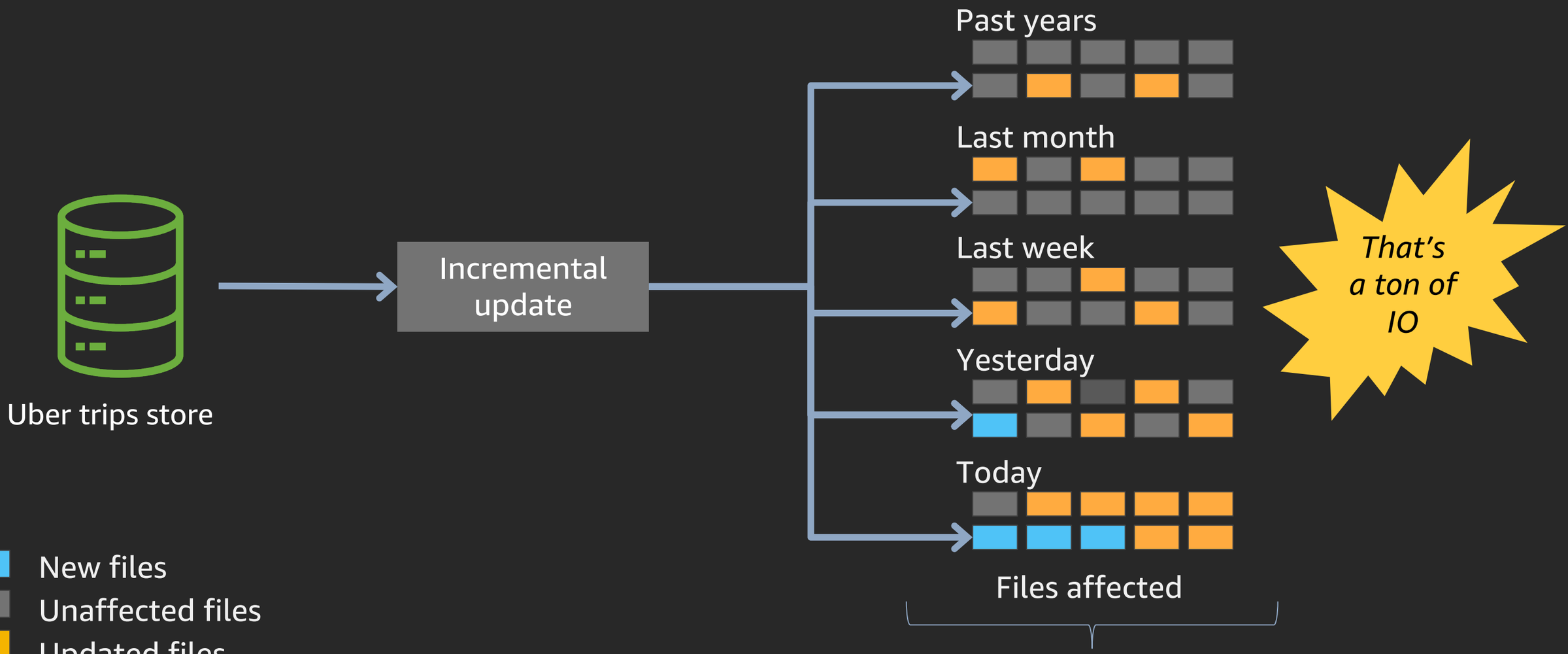


Amazon S3

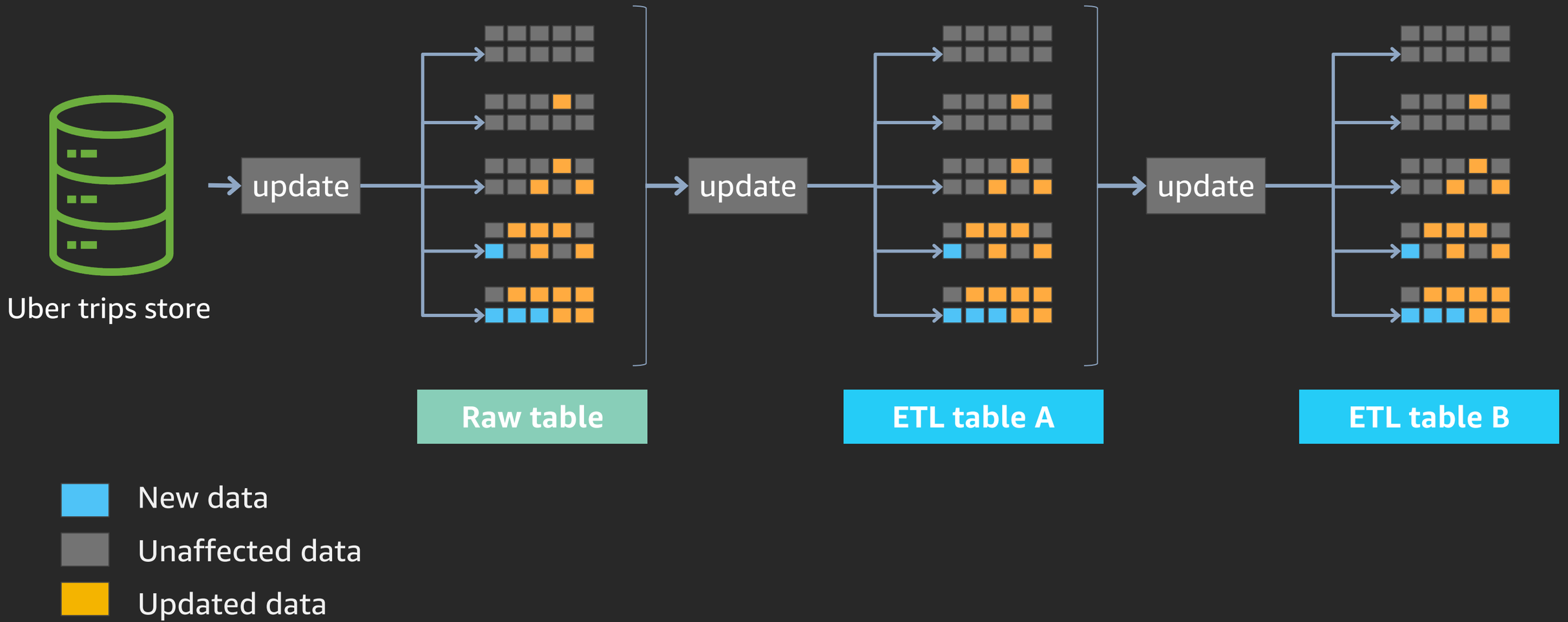
Impressions

Field Name	Type
event_id	string
datestr	string
time	long

Motivating use case



Cascading effects

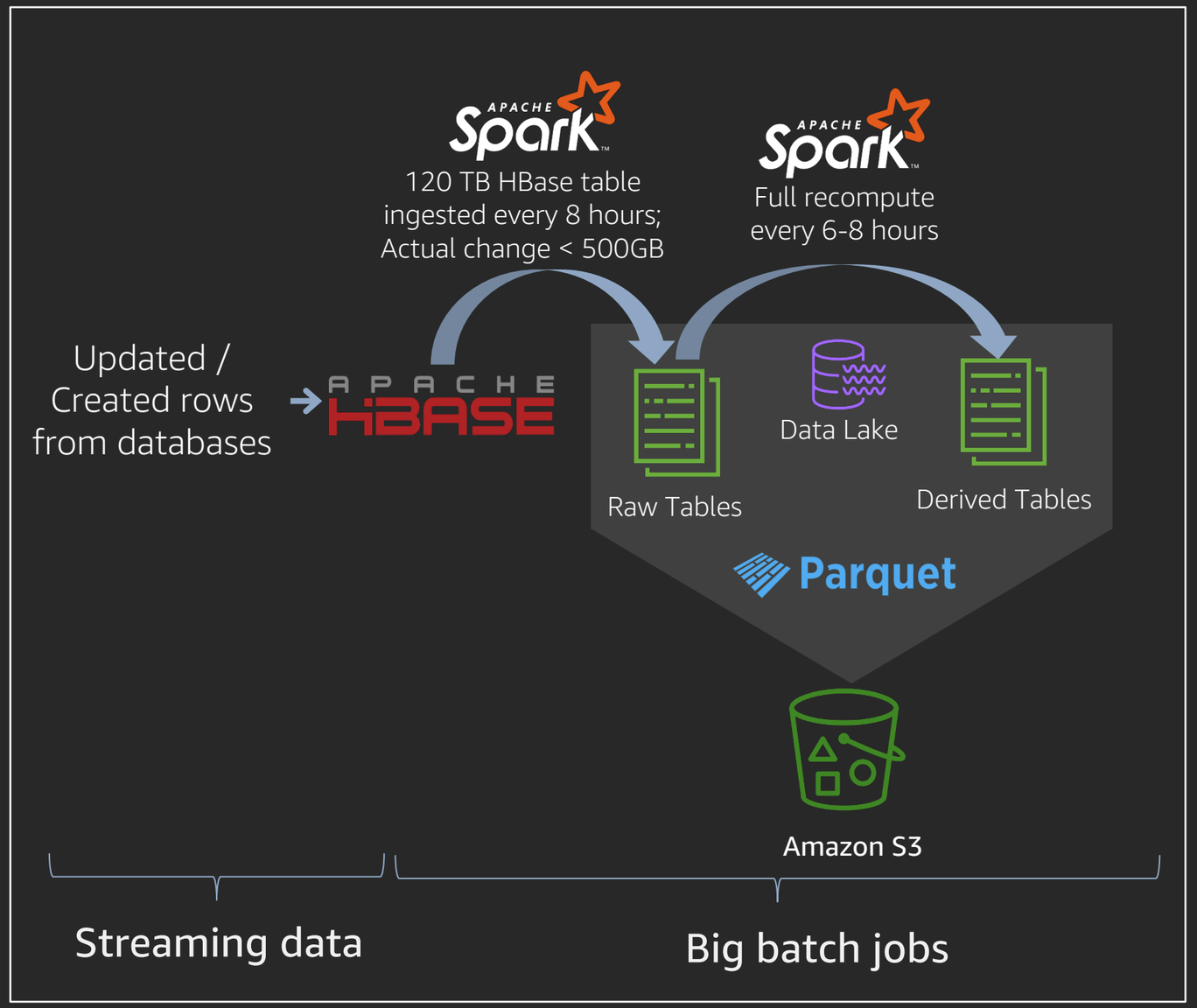


Slow data lake

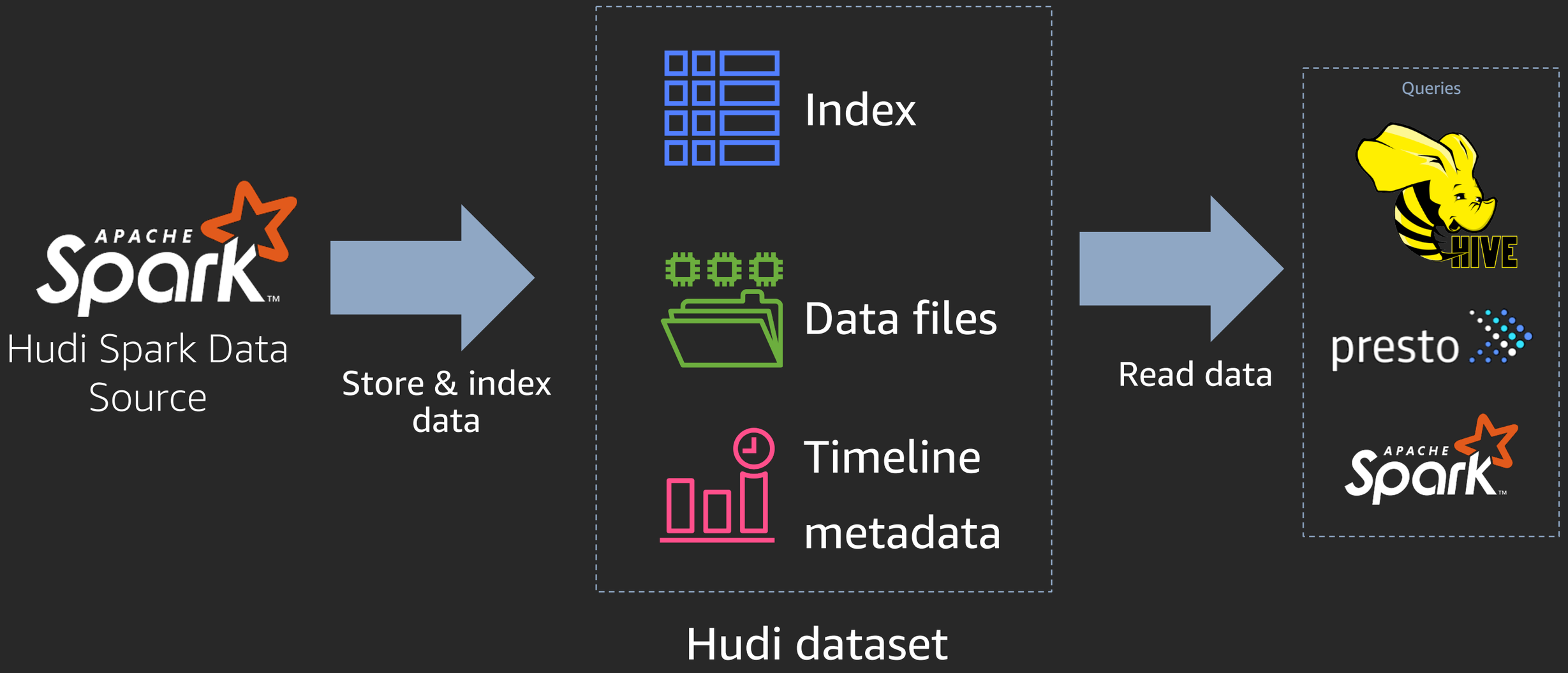
120 TB Apache Hbase table ingested every

8 hours to account for less than 500GB of

data



Introducing Apache Hudi (incubating)



Why You Should Care

- Near real-time data ingestion
- Batch jobs on steroids
- Unified, optimized analytical storage
- Row-level deletions to simplify data privacy
- Building block for great data lakes
- All open source, open formats

Primitives supported

- upsert() support with fast, pluggable indexing
- Atomic publish with rollback, save points
- Snapshot isolation between writer & queries
- Manages file sizes, layout using statistics
- Async compaction of row & columnar data
- Timeline metadata to track lineage

Storage types

Copy On Write

Read heavy scenarios

Hudi Dataset

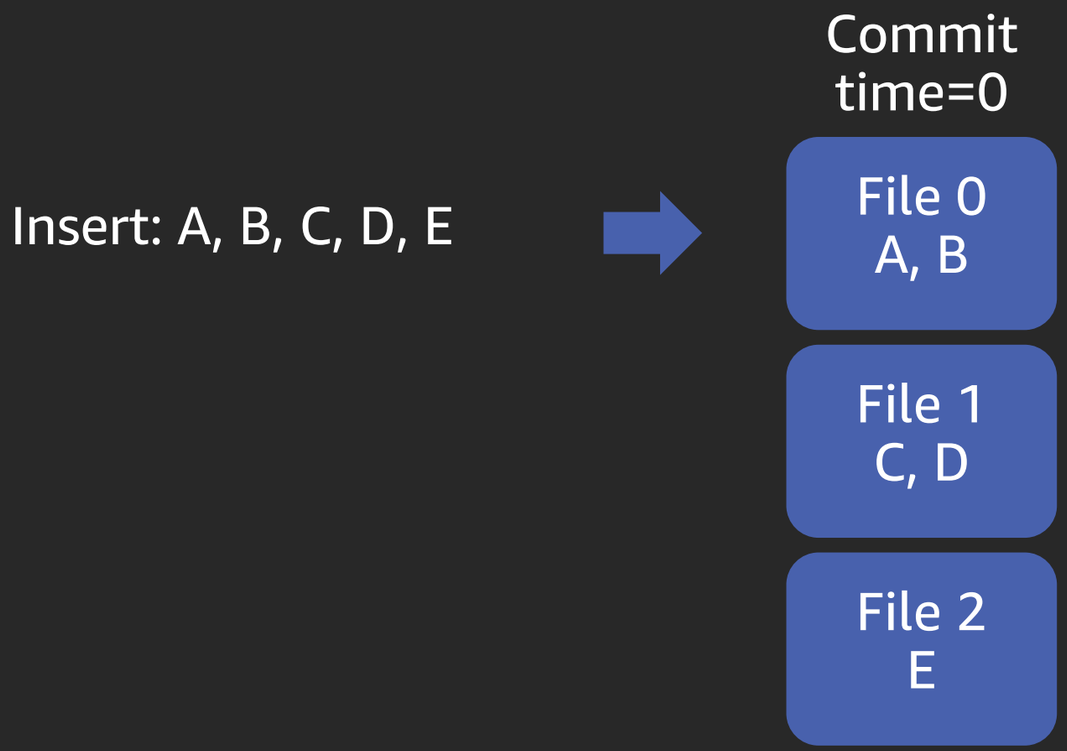
Merge On Read

Write heavy scenarios

Storage types & Views

Storage Type: Copy On Write

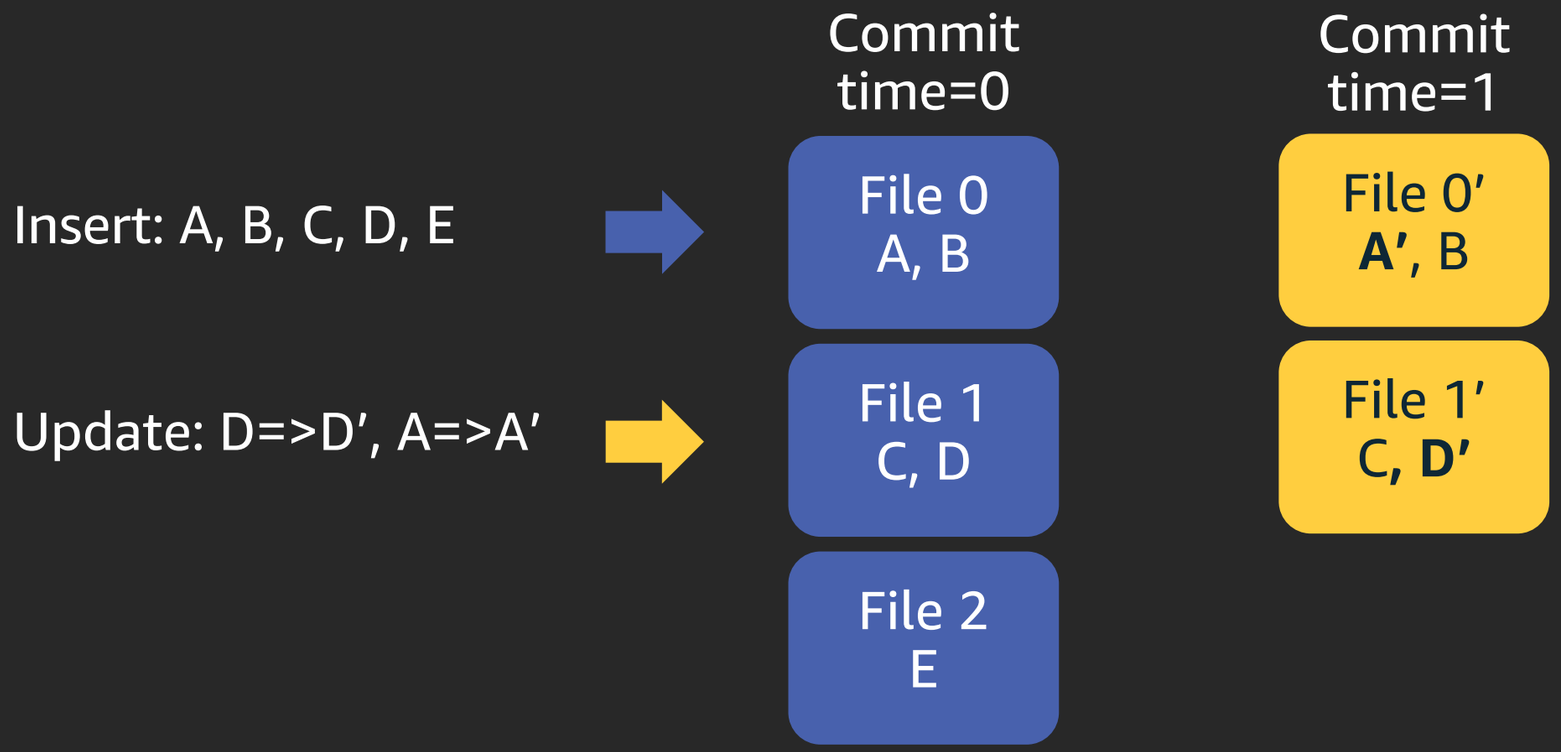
Views/Queries: Read-Optimized, Incremental



Storage types & Views

Storage Type: Copy On Write

Views/Queries: Read-Optimized, Incremental

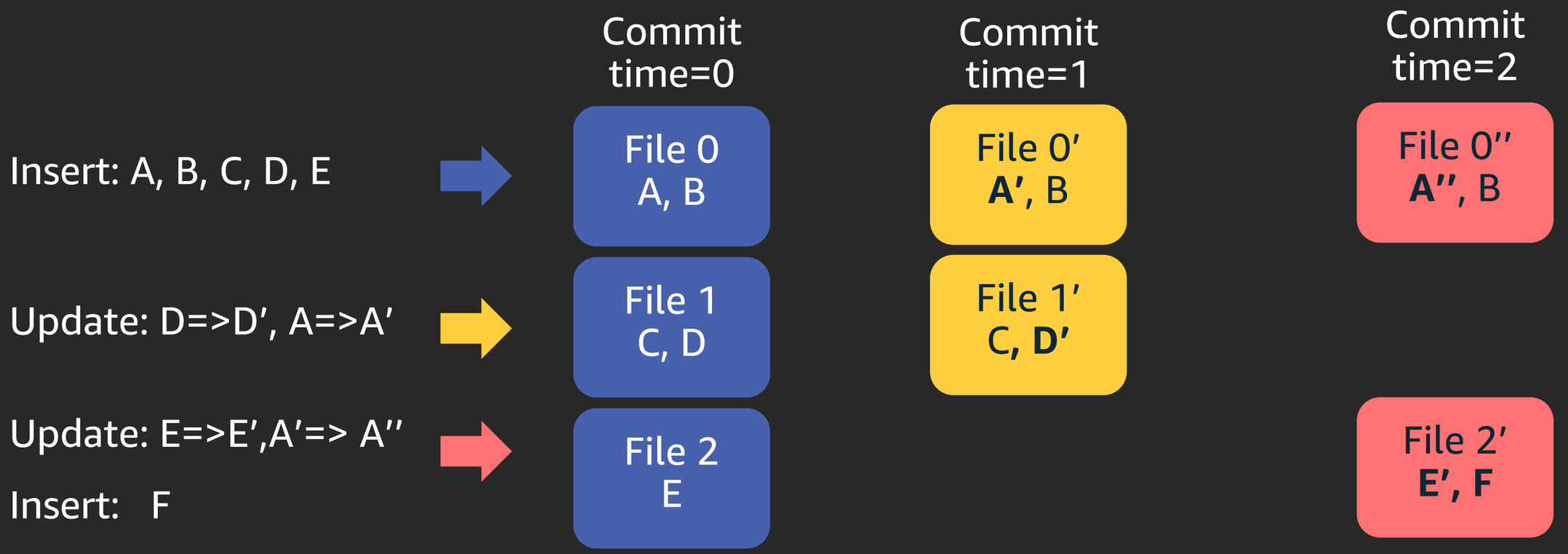


Read-Optimized	A,B,C,D,E	A',B,C,D',E
Incremental	A,B,C,D,E	A',D'

Storage types & Views

Storage Type: Copy On Write

Views/Queries: Read-Optimized, Incremental



	Commit time=0	Commit time=1	Commit time=2
Read-Optimized	A,B,C,D,E	A',B,C,D',E	A'',B,C,D',E',F
Incremental	A,B,C,D,E	A',D'	A'',E', F

Storage types & Views

Storage type: Copy On Write

Views: Read Optimized, Incremental

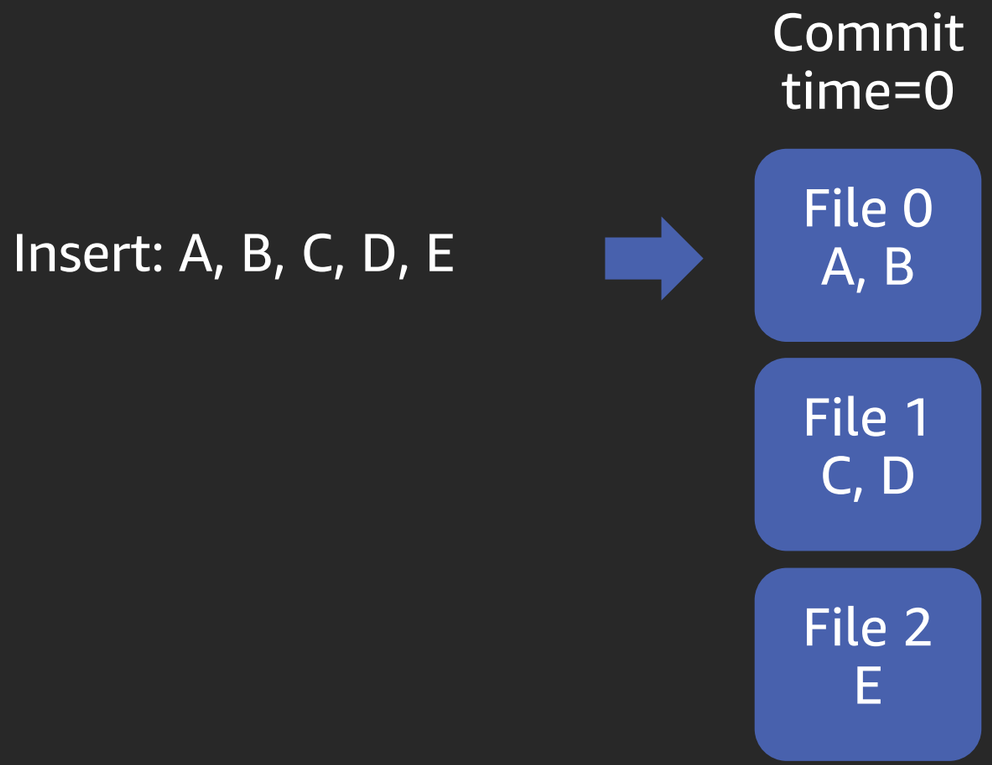
When to Use

- Your current job is rewriting entire table/partition to deal with updates
- Your workload is fairly well understood and does not have sudden bursts
- You’re already using Parquet files for your tables
- You want to keep things operationally simple

Storage types & Views

Storage type: Merge On Read

Views/Queries: Read Optimized, Incremental, Real Time

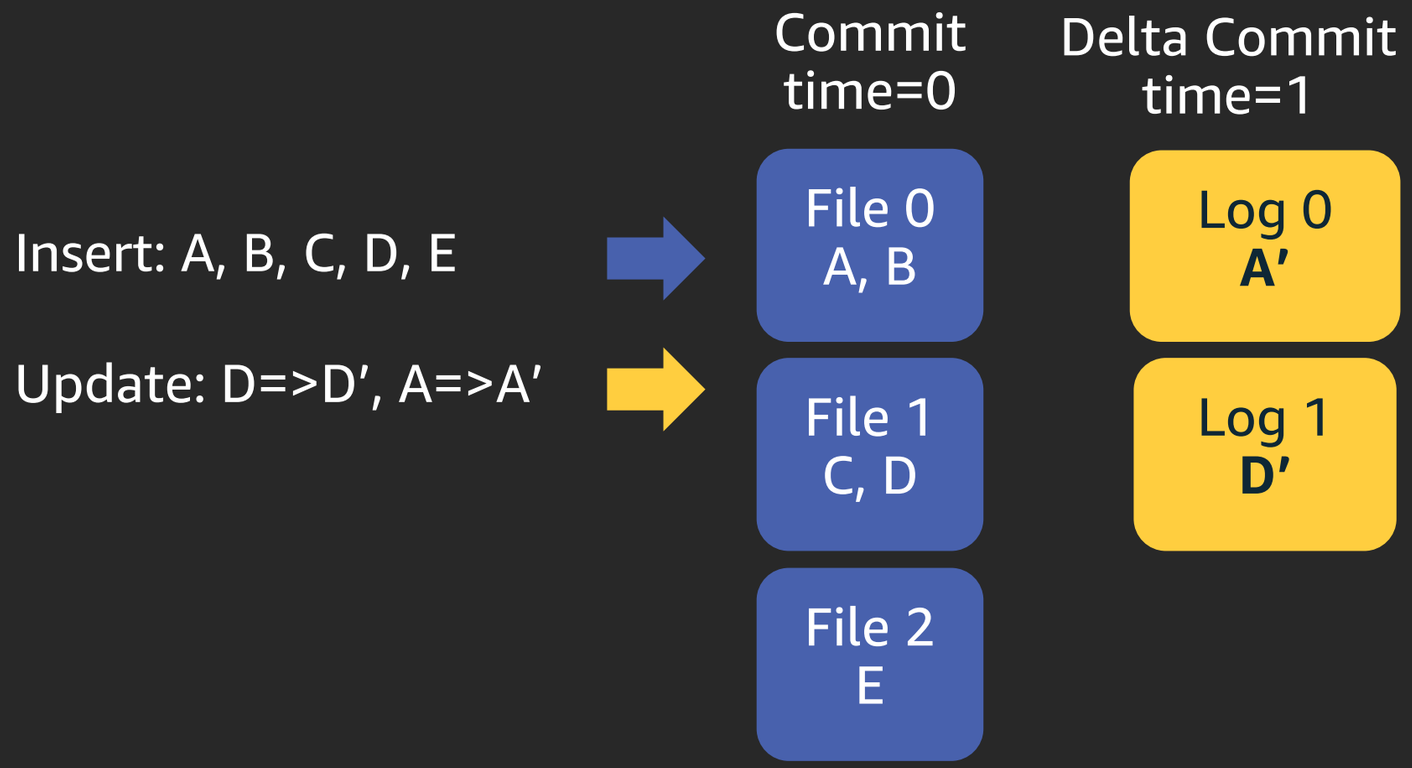


Real time	A,B,C,D,E
Incremental	A,B,C,D,E
Read-Optimized	A,B,C,D,E

Storage types & Views

Storage type: Merge On Read

Views/Queries: Read Optimized, Incremental, Real time

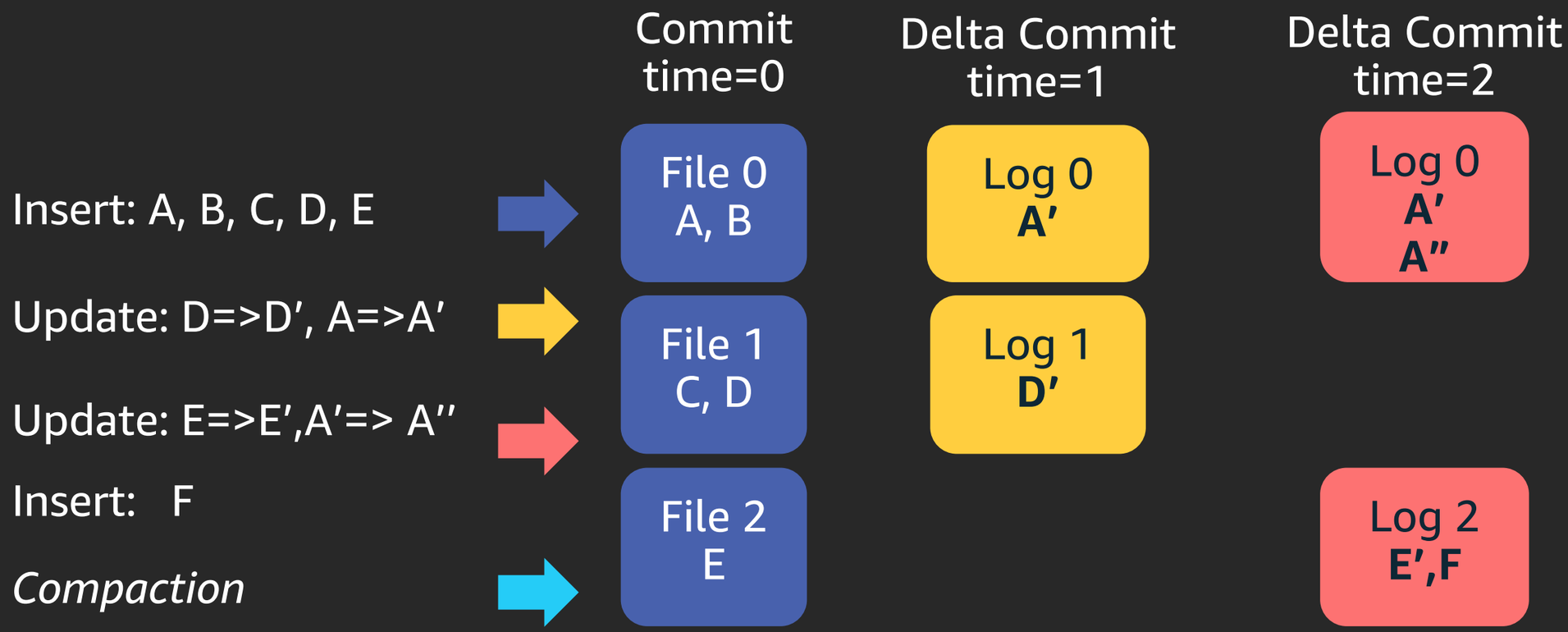


Real time	A,B,C,D,E	A',B,C,D',E
Incremental	A,B,C,D,E	A',D'
Read-Optimized	A,B,C,D,E	A,B,C,D,E

Storage types & Views

Storage type: Merge On Read

Views/Queries: Read Optimized, Incremental, Real time

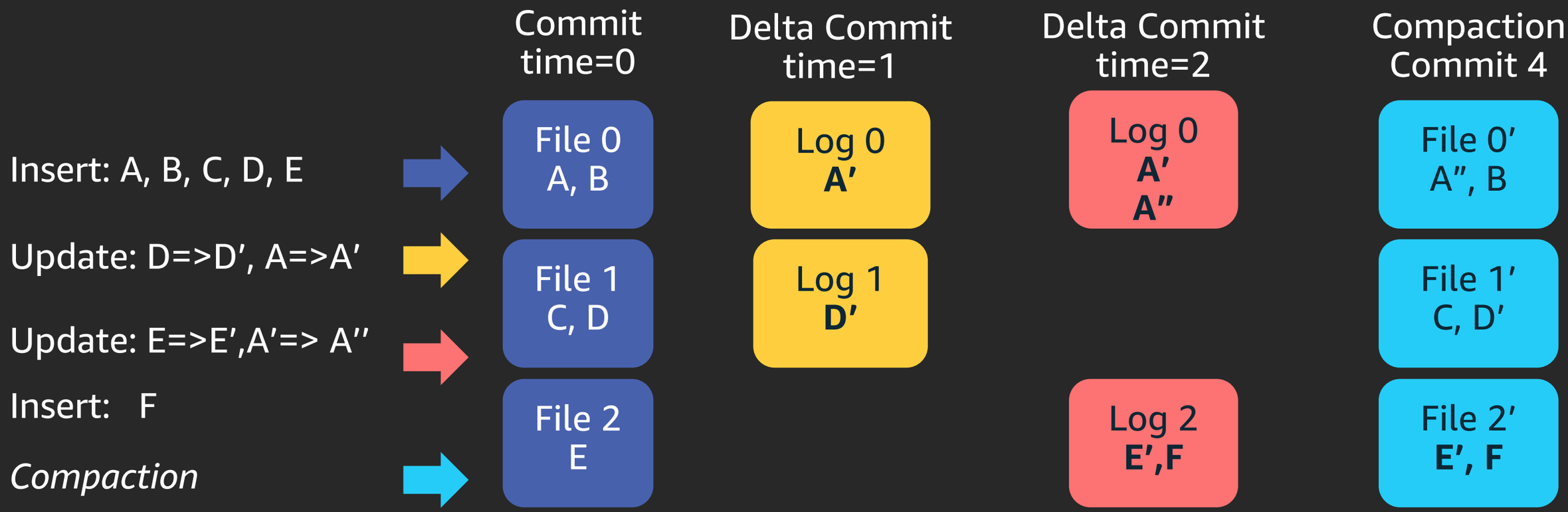


	Commit time=0	Delta Commit time=1	Delta Commit time=2
Real time	A,B,C,D,E	A',B,C,D',E	A'',B,C,D',E',F
Incremental	A,B,C,D,E	A',D'	A'',E', F
Read-Optimized	A,B,C,D,E	A,B,C,D,E	A,B,C,D,E

Storage types & Views

Storage type: Merge On Read

Views/Queries: Read Optimized, Incremental, Real time



	Commit time=0	Delta Commit time=1	Delta Commit time=2	Compaction Commit 4
Real time	A,B,C,D,E	A',B,C,D',E	A'',B,C,D',E',F	A'',B,C,D',E',F
Incremental	A,B,C,D,E	A',D'	A'',E', F	
Read-Optimized	A,B,C,D,E	A,B,C,D,E	A,B,C,D,E	A'',B,C,D',E',F

Storage types & Views

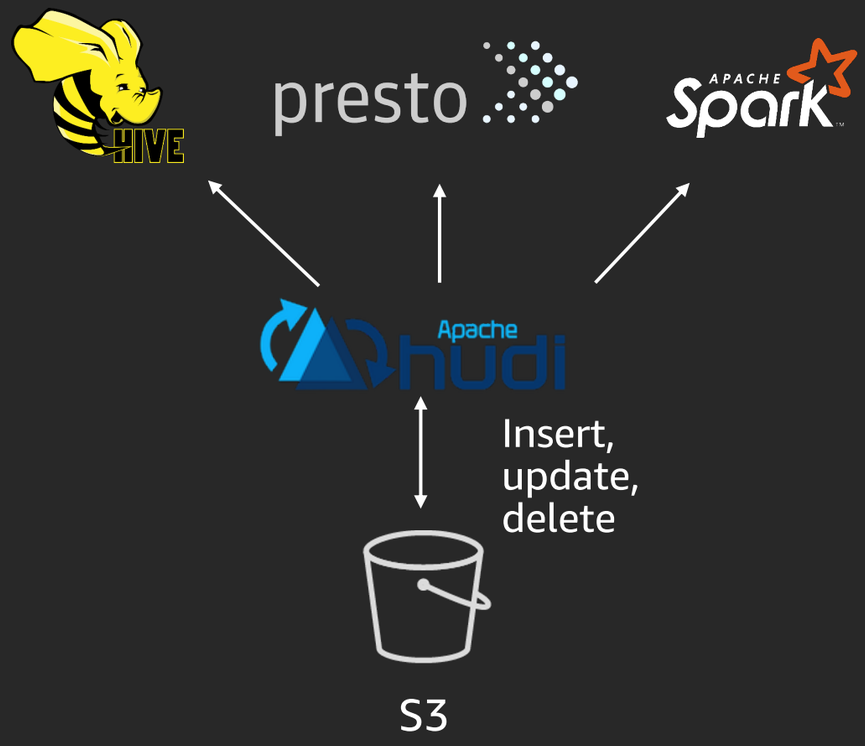
Storage type: Merge On Read

Views: Read Optimized, Incremental, Real time

When to Use

- You want ingested data available for query as fast as possible
- Your workload can have sudden spikes or changes in pattern
  - Example: Bulk updates to older transactions in upstream database cause updates to old partitions in Amazon S3

So why should you use Apache Hudi



Open source, open format, vendor neutral with Apache HUDI

Support for Spark, Hive, and Presto

Enables data lakes to

a) Comply with data privacy laws

b) Consume real-time streams and change data captures

c) Reinstate late arriving data

d) Track change history and rollback

Storage type: Trade-offs

Trade-off	Copy on write	Merge on read
Data latency	Higher	Lower
Update cost (I/O)	Higher (rewrite entire Parquet)	Lower (append to delta log)
Parquet file size	Smaller (high update cost)	Larger

Views

Read optimized view

- Latest snapshot of the dataset as of a given commit or compaction action
- Exposes only the base/columnar
- Guarantees the same columnar query performance compared to a non-Hudi columnar dataset

Incremental view

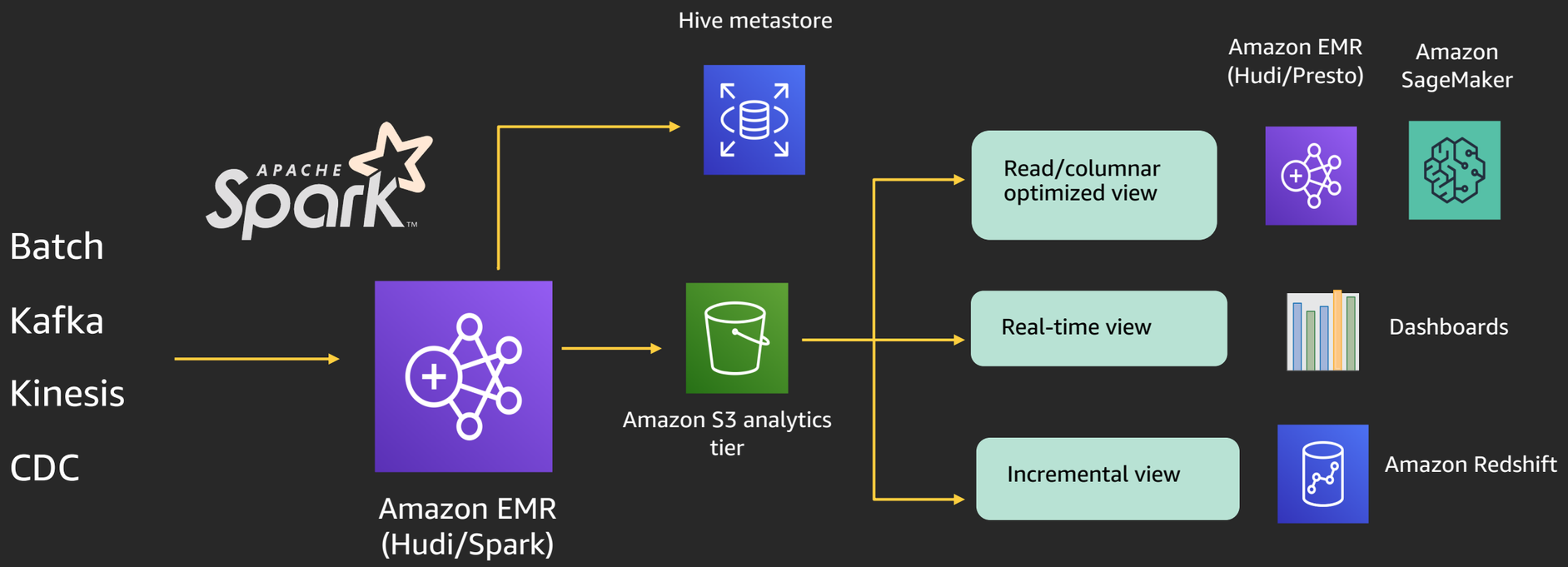
- New data written to the dataset, since a given commit/compaction
- Provides change streams to enable incremental data pipelines

Real-time view

- Latest snapshot of dataset as of a given delta commit action
- Provides near-real-time datasets (few minutes) by merging the base and delta files of the latest file slice on the fly

Two ways towrite Hudi datasets

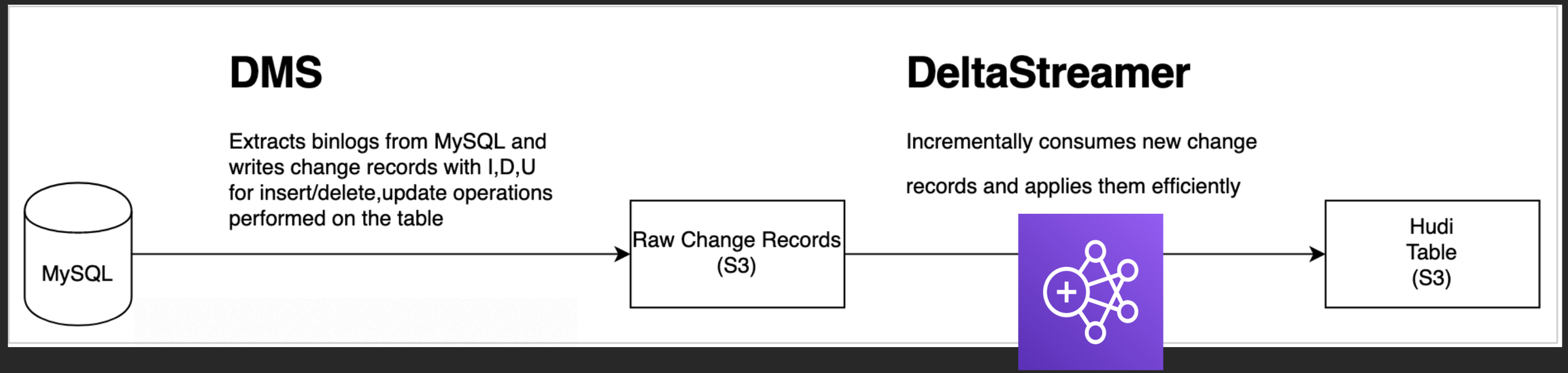
1. Hudi Data Source API




```
df1.write.format(HUDI_FORMAT)
    .option(PRECOMBINE_FIELD_OPT_KEY, config["sort_key"])
    .option(RECORDKEY_FIELD_OPT_KEY, config["primary_key"])……
```

```
df2=spark.read.format(HUDI_FORMAT).load(config["target"]+"/*")….
spark.sql("select count(*) from table_name").show(20,False)
```

2. Hudi Delta Streamer



```
spark-submit --class org.apache.hudi.utilities.deltastreamer.HoodieDeltaStreamer
--jars /usr/lib/spark/external/lib/spark-avro_2.11-2.4.5-amzn-0.jar
--master yarn --deploy-mode client
--executor-memory 10G --executor-cores 4
file:///usr/lib/hudi/hudi-utilities-bundle_2.11-0.5.2-incubating.jar
--table-type COPY_ON_WRITE
--source-ordering-field TIMESTAMP
--enable-hive-sync
--source-class org.apache.hudi.utilities.sources.ParquetDFSSource
--transformer-class org.apache.hudi.utilities.transform.AWSDmsTransformer
--target-base-path s3://mybucket/hudi_table --target-table table_test
--payload-class org.apache.hudi.payload.AWSDmsAvroPayload
--hoodie-conf hoodie.datasource.write.recordkey.field="Field1, Field2, Field3"
--hoodie-conf hoodie.datasource.write.keygenerator.class=org.apache.hudi.keygen.ComplexKeyGenerator
--hoodie-conf hoodie.datasource.hive_sync.database=default
--hoodie-conf hoodie.datasource.hive_sync.table=hudi_table
--hoodie-conf hoodie.datasource.hive_sync.partition_fields="datefield"
--hoodie-conf hoodie.deltastreamer.source.dfs.root=s3://mybucket/input
```

Action time!

Navigate to the following URL

http://bit.ly/2qhiVO4

- Select Workshop section from the panel on the left
- Expand the section and go to the AWS Workshop Portal
- Follow the instructions in the Workshop Portal to complete the lab activities

Thank you!

AWS re:Invent

aws

Please complete the session survey in the mobile app.