Designing Structured Streaming Pipelines

How to Architect Things Right

Tathagata "*TD*" Das

@tathadas

#UnifiedAnalytics #SparkAISummit

Apache Spark Structured Streaming

Distributed stream processing built on SQL engine

- High throughput, second-scale latencies
- Fault-tolerant, exactly-once
- Great set of connectors

Philosophy: Data streams are unbounded tables

- Users write batch-like code on a table
- Spark will automatically run code incrementally on streams

Structured Streaming @ databricks

1000s of customer streaming apps
in production on Databricks

1000+ trillions of rows processed
in production

Structured Streaming

Example

Read JSON data from Kafka

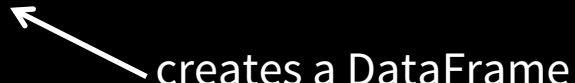
Parse nested JSON

Store in structured Parquet table

Get end-to-end failure guarantees

Anatomy of a Streaming Query

```
spark.readStream.format("kafka")
  .option("kafka.boostrap.servers",...)
  .option("subscribe", "topic")
  .load()
```



key	value	topic	partition	offset	timestamp
[binary]	[binary]	topicA	0	345	1486087873
[binary]	[binary]	topicB	3	2890	1486086721

Source

Specify where to read data from

Built-in support for Files / Kafka / Kinesis*

Can include multiple sources of different types using `join()` / `union()`

*Available only on Databricks Runtime

Anatomy of a Streaming Query

```
spark.readStream.format("kafka")
  .option("kafka.boostrap.servers",...)
  .option("subscribe", "topic")
  .load()
  .selectExpr("cast (value as string) as json")
  .select(from_json("json", schema).as("data"))
```

Transformations

Cast bytes from Kafka to a string, parse it as a json, and generate nested columns

100s of built-in, optimized SQL functions like `from_json`

user-defined functions, lambdas, function literals with map, flatMap…

Anatomy of a Streaming Query

```
spark.readStream.format("kafka")
  .option("kafka.boostrap.servers",...)
  .option("subscribe", "topic")
  .load()
  .selectExpr("cast (value as string) as json")
  .select(from_json("json", schema).as("data"))
  .writeStream
  .format("parquet")
  .option("path", "/parquetTable/")
```

Sink

Write transformed output to external storage systems

Built-in support for Files / Kafka

Use `foreach` to execute arbitrary code with the output data

Some sinks are transactional and exactly once (e.g. files)

Anatomy of a Streaming Query

```
spark.readStream.format("kafka")
  .option("kafka.boostrap.servers",...)
  .option("subscribe", "topic")
  .load()
  .selectExpr("cast (value as string) as json")
  .select(from_json("json", schema).as("data"))
  .writeStream
  .format("parquet")
  .option("path", "/parquetTable/")
  .trigger("1 minute")
  .option("checkpointLocation", "…")
  .start()
```

Processing Details

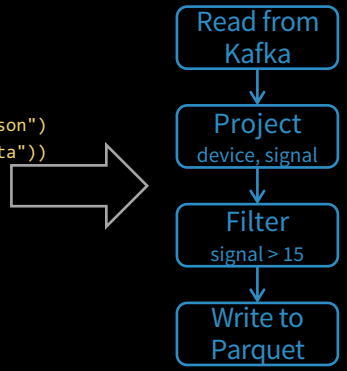
Trigger: when to process data

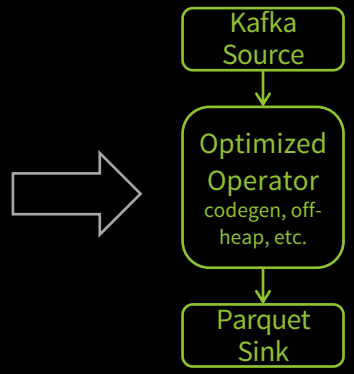
- Fixed interval micro-batches
- As fast as possible micro-batches
- Continuously (new in Spark 2.3)

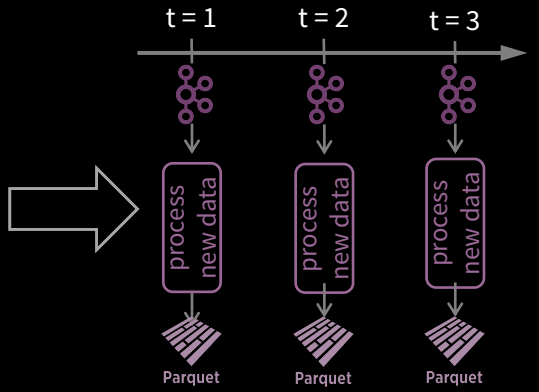
Checkpoint location: for tracking the progress of the query

Spark automatically streamifies!

```
spark.readStream.format("kafka")
  .option("kafka.boostrap.servers",...)
  .option("subscribe", "topic")
  .load()
  .selectExpr("cast (value as string) as json")
  .select(from_json("json", schema).as("data"))
  .writeStream
  .format("parquet")
  .option("path", "/parquetTable/")
  .trigger("1 minute")
  .option("checkpointLocation", "…")
  .start()
```







DataFrames, Datasets, SQL

Logical Plan

Optimized Plan

Series of Incremental Execution Plans

Spark SQL converts batch-like query to a series of incremental execution plans operating on new batches of data

Anatomy of a Streaming Query

```
spark.readStream.format("kafka")
  .option("kafka.boostrap.servers",...)
  .option("subscribe", "topic")
  .load()
  .selectExpr("cast (value as string) as json")
  .select(from_json("json", schema).as("data"))
  .writeStream
  .format("parquet")
  .option("path", "/parquetTable/")
  .trigger("1 minute")
  .option("checkpointLocation", "…")
  .start()
```

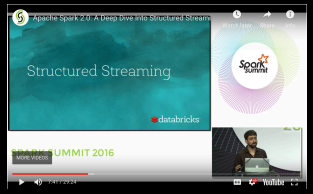
Raw data from Kafka available as structured data in seconds, ready for querying

My past talks: Deep dives

APACHE Spark™ STRUCTURED STREAMING

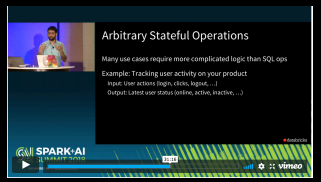
[A Deep Dive Into Structured Streaming]

Spark Summit 2016



[A Deep Dive into Stateful Stream Processing in Structured Streaming]

Spark + AI Summit 2018



This talk: Streaming Design Patterns

Zoomed out view of your data pipeline

Another streaming design pattern talk?

Most talks

Focus on a pure streaming engine

Explain one way of achieving the end goal

This talk

Spark is more than a streaming engine

Spark has multiple ways of achieving the end goal with tunable perf, cost and quality

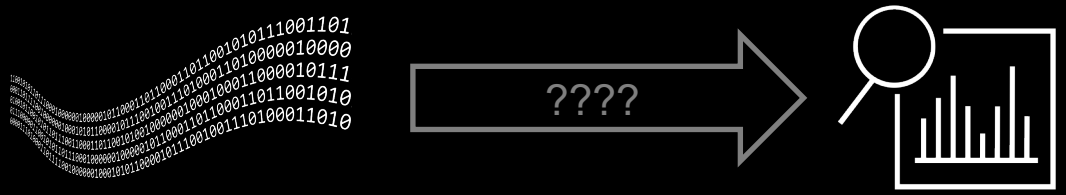
This talk

How to think about design

Common design patterns

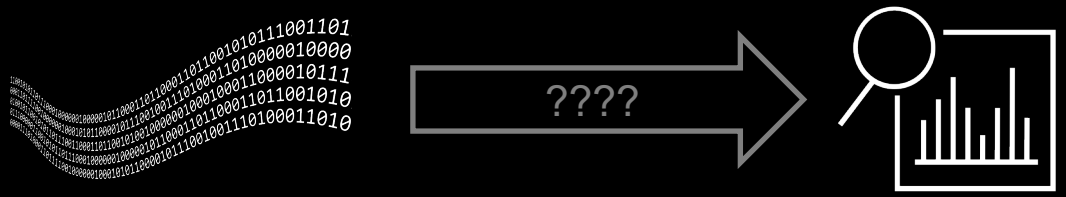
How we are making this easier

Streaming Pipeline Design



Data streams

Insights



What?

Why?

How?

What?

What is your input?

What is your data?

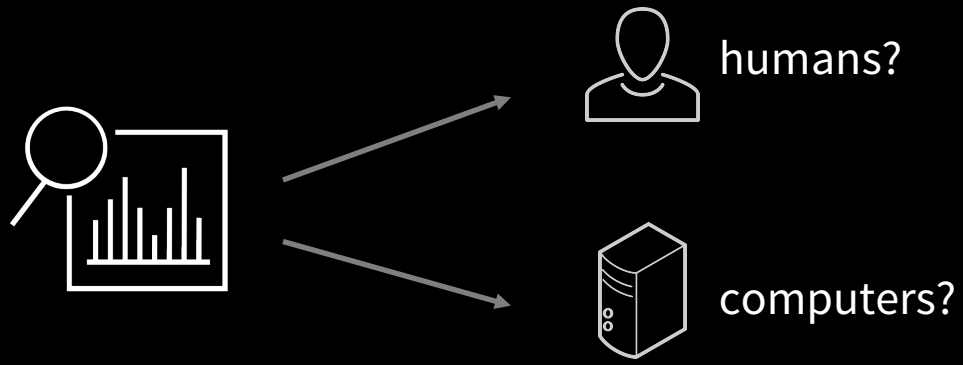
What format and system is your data in?

What is your output?

What results do you need?

What throughput and latency do you need?

Why?



Why do you want this output in this way?

Who is going to take actions based on it?

When and how are they going to consume it?

Why? Common mistakes!

#1

"I want my dashboard with counts to be updated every second"

No point of updating every second if humans are going to take actions in minutes or hours

#2

"I want to generate automatic alerts with up-to-the-last second count"

(but my input data is often delayed)

No point taking fast actions on low quality data and results

Why? Common mistakes!

#3

"I want to train machine learning models on the results"

(but my results are in a key-value store)

Key-value stores are not great for large, repeated data scans which machine learning workloads perform

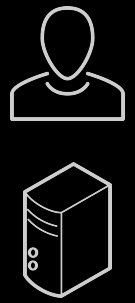
How?

How to process the data?

????

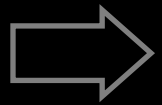
How to store the results?

????



Streaming Design Patterns

What?
Why?



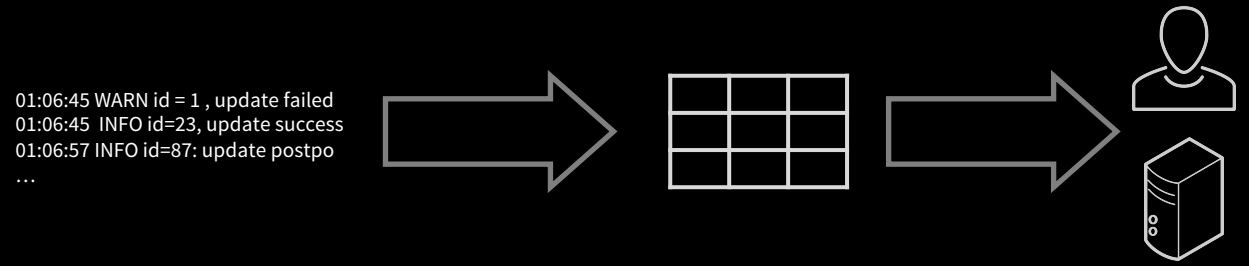
How?

Pattern 1: ETL

What?

Input: unstructured input stream from files, Kafka, etc.

Output: structured tabular data



Why? Query latest structured data interactively or with periodic jobs

P1: ETL

What?

Convert unstructured input to structured tabular data

Latency: few minutes

Why?

Query latest structured data interactively or with periodic jobs

How?

Process: Use Structured Streaming query to transform unstructured, dirty data

Run 24/7 on a cluster with default trigger

Store: Save to structured scalable storage that supports data skipping, etc.

E.g.: Parquet, ORC, or even better, Delta Lake

P1: ETL

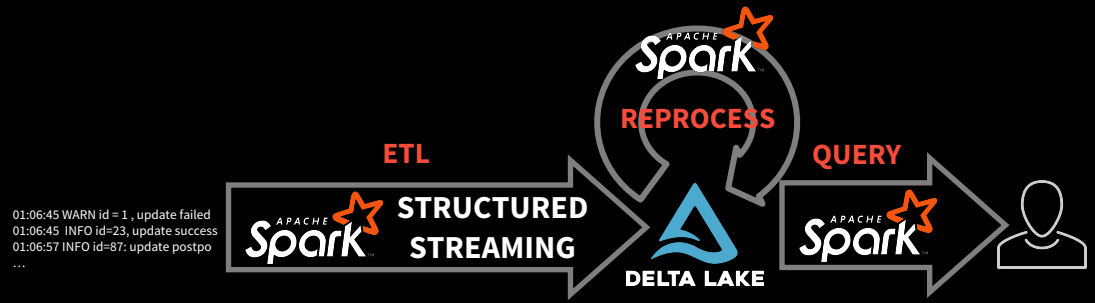
How?

Store: Save to DELTA LAKE

Read with snapshot guarantees while writes are in progress

Concurrently reprocess data with full ACID guarantees

- Coalesce small files into larger files
- Fix mistakes in existing data



P1: Cheaper ETL

What?

Convert unstructured input to structured tabular data

Latency: ~~few minutes~~ hours

Not have clusters up 24/7

Why?

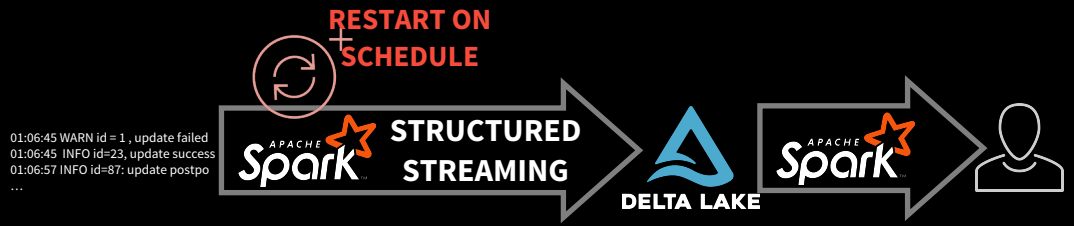
Query latest data ~~interactively~~ ~~or~~ with periodic jobs

Cheaper solution

How?

Process: Still use Structured Streaming query!

- Run streaming query with "trigger.once" for processing all available data since last batch
- Set up external schedule (every few hours?) to periodically start a cluster and run one batch



P1: Query faster than ETL!

What?

Latency: ~~hours~~ seconds

How?

Query data in Kafka directly using Spark SQL

Can process up to the last records received by Kafka when the query was started

Why?

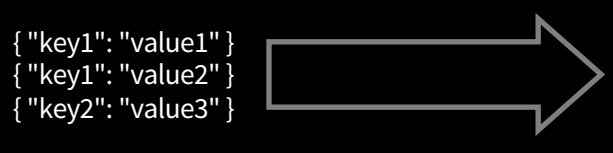
Query latest up-to-the last second data interactively

Pattern 2: Key-value output

What?

Input: new data for each key

Output: updated values for each key



KEY	LATEST VALUE
key1	value2
key2	value3

Aggregations (sum, count, …)

Sessionizations

Why?

Lookup latest value for key (dashboards, websites, etc.)

OR

Summary tables for querying interactively or with periodic jobs

P2.1: Key-value output for lookup

What?

Generate updated values for keys

Latency: seconds/minutes

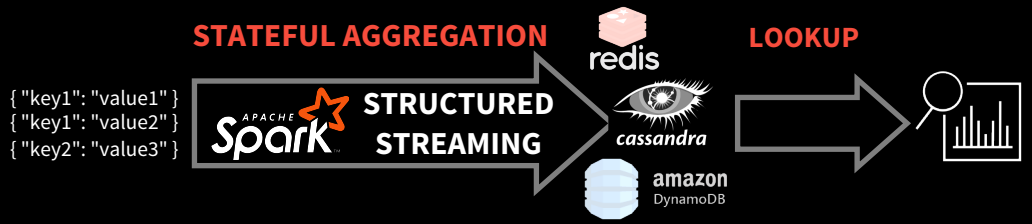
How?

Process: Use Structured Streaming with stateful operations for aggregation

Store: Save in key-values stores optimized for single key lookups

Why?

Lookup latest value for key



P2.2: Key-value output for analytics

What?

Generate updated values for keys

Latency: ~~seconds/~~minutes

How?

Process: Use Structured Streaming with stateful operations for aggregation

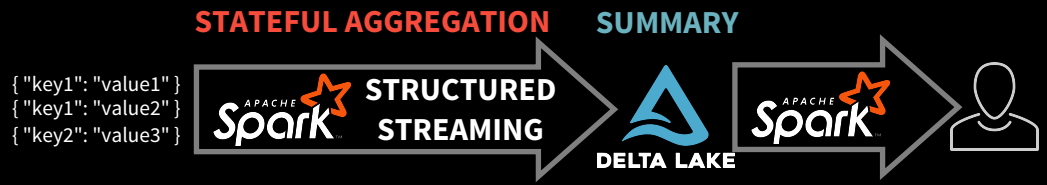
Store: Save in *Delta Lake*!

Managed Delta Lake supports upserts using SQL MERGE

Why?

~~Lookup latest value for key~~

Summary tables for analytics



P2.2: Key-value output for analytics

How?

stateful operations for aggregation

Managed Delta Lake supports upserts using SQL MERGE

```
streamingDataFrame.foreachBatch { batchOutput =>
  spark.sql("""
      MERGE INTO deltaTable USING batchOutput
      WHEN MATCHED THEN UPDATE ...
      WHEN NOT MATCHED THEN INSERT ...
      """)
}.start()
```

Coming to OSS Delta Lake soon!

P2.2: Key-value output for analytics

How?

Stateful operations for aggregation

Managed Delta Lake supports upserts using SQL MERGE

Stateful aggregation requires setting watermark to drop very late data

Dropping some data leads some inaccuracies

P2.3: Key-value aggregations for analytics

What?

Generate aggregated values for keys

Latency: hours/days

Do not drop any late data

Why?

Summary tables for analytics

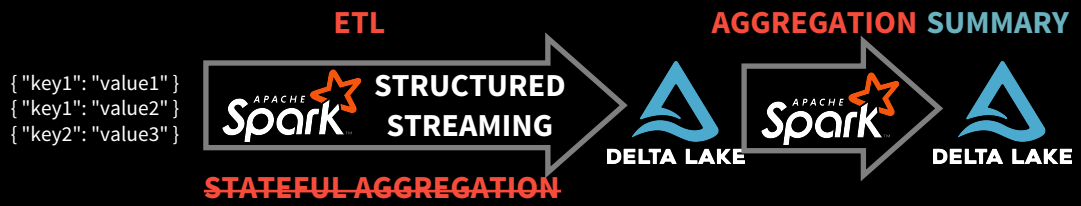
Correct

How?

Process: ETL to structured table (no stateful aggregation)

Store: Save in Delta Lake

Post-process: Aggregate after all delayed data received

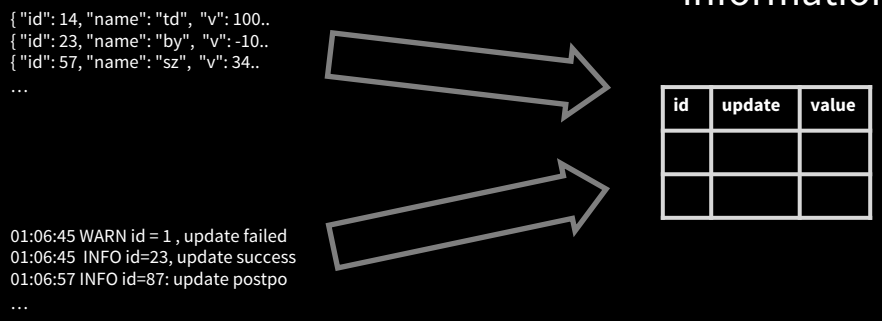


Pattern 3: Joining multiple inputs

What?

Input: Multiple data streams based on common key

Output: Combined information



P3.1: Joining fast and slow data

What + Why?

Input: One fast stream of facts and one slow stream of dimension changes

Output: Fast stream enriched by data from slow stream

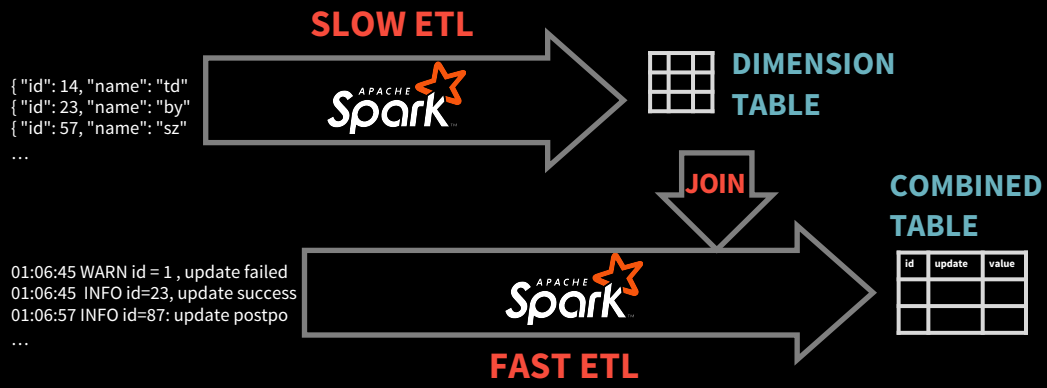
Example:

product sales info enriched by more production info

How?

ETL slow stream to a dimension table

Join fast stream with snapshots of the dimension table

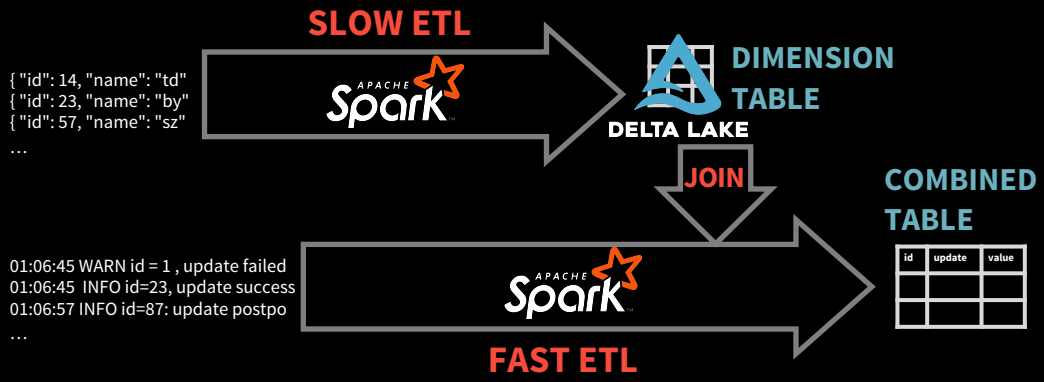


P3.1: Joining fast and slow data

How? - Caveats

Structured Streaming by default does reload dimension table snapshot

Changes by slow ETL wont be seen until restart



Better Solution

Store dimension table in Delta Lake

Delta Lake's versioning allows changes to be detected and the snapshot automatically reloaded without restart**

*** available only in Managed Delta Lake in Databricks Runtime*

P3.1: Joining fast and slow data

How? - Caveats

Delays in updates to dimension table can cause joining with stale dimension data

E.g. sale of a product received even before product table has any info on the product

Better Solution

Treat it as a "joining fast and fast data"

P3.2: Joining fast and fast data

What + Why?

Input: Two fast streams where either stream maybe delayed

Output: Combined info even if one is delayed over another

Example:

ad impressions and ad clicks

Use stream-stream joins in Structured Streaming

- Data will be buffered as state
- Watermarks define how long to buffer before giving up on matching

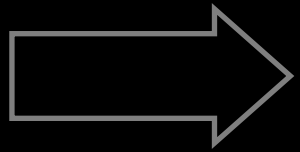
See my previous deep dive talk for more info

Pattern 4: Change data capture

What? Input: Change data based on a primary key

Output: Final table after changes

```
INSERT a, 1
INSERT b, 2
UPDATE a, 3
DELETE b
INSERT b, 4
```



key	value
a	3
b	4

Why? End-to-end replication of transactional tables into analytical tables

P4: Change data capture

How?

Use Structured Streaming and Delta Lake!

- In each batch, apply changes to the Delta table using MERGE

MERGE in Managed Data Lake supports UPDATE, INSERT and DELETE

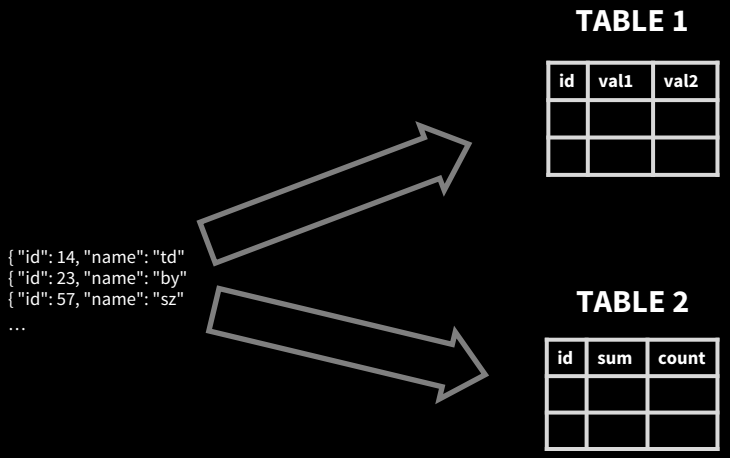
- Coming soon to OSS Delta Lake!

```
streamingDataFrame.foreachBatch { batchOutput =>
   spark.sql("""
      MERGE INTO deltaTable USING batchOutput
      WHEN MATCHED ... THEN UPDATE ...
      WHEN MATCHED ... THEN INSERT ...
      WHEN NOT MATCHED THEN INSERT ...
      """)
}.start()
```

```
INSERT a, 1
INSERT b, 2
UPDATE a, 3
DELETE b
INSERT b, 4
```

See Databricks docs for more info

Pattern 5: Writing to multiple outputs



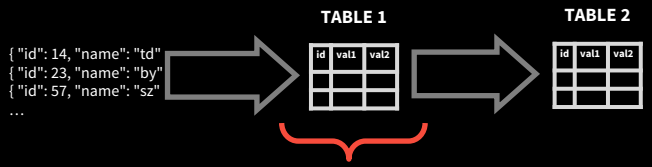
What?

RAW LOGS	TABLE CHANGE LOG	SUMMARY FOR ANALYTICS
+	+	+
SUMMARIES	UPDATED TABLE	SUMMARY FOR LOOKUP

P5: Serial or Parallel?

How?

Serial

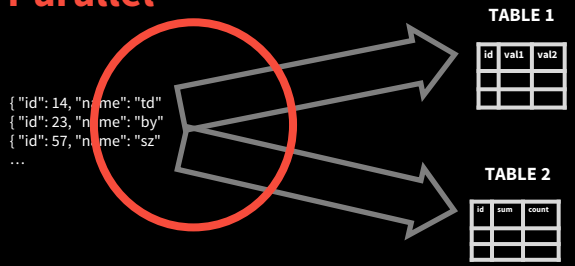


Writes table 1 and reads it again

Cheap or expensive depends on the size and format of table 1

Higher latency

Parallel



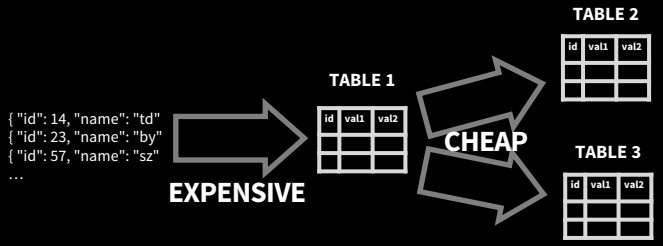
Reads input twice, may have to parse the data twice

Cheap or expensive depends on size of raw input + parsing cost

P5: Combination!

How?

Combo 1: Multiple streaming queries



Do expensive parsing once, write to table 1

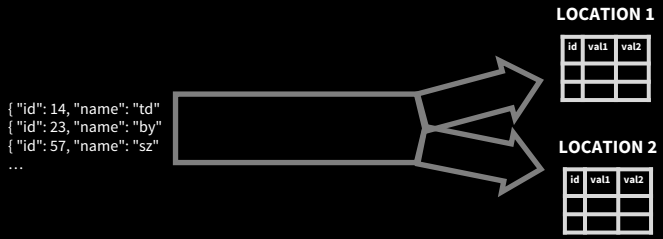
Do cheaper follow up processing from table 1

Good for

- ETL + multiple levels of summaries
- Change log + updated table

Still writing + reading table1

Combo 2: Single query + foreachBatch

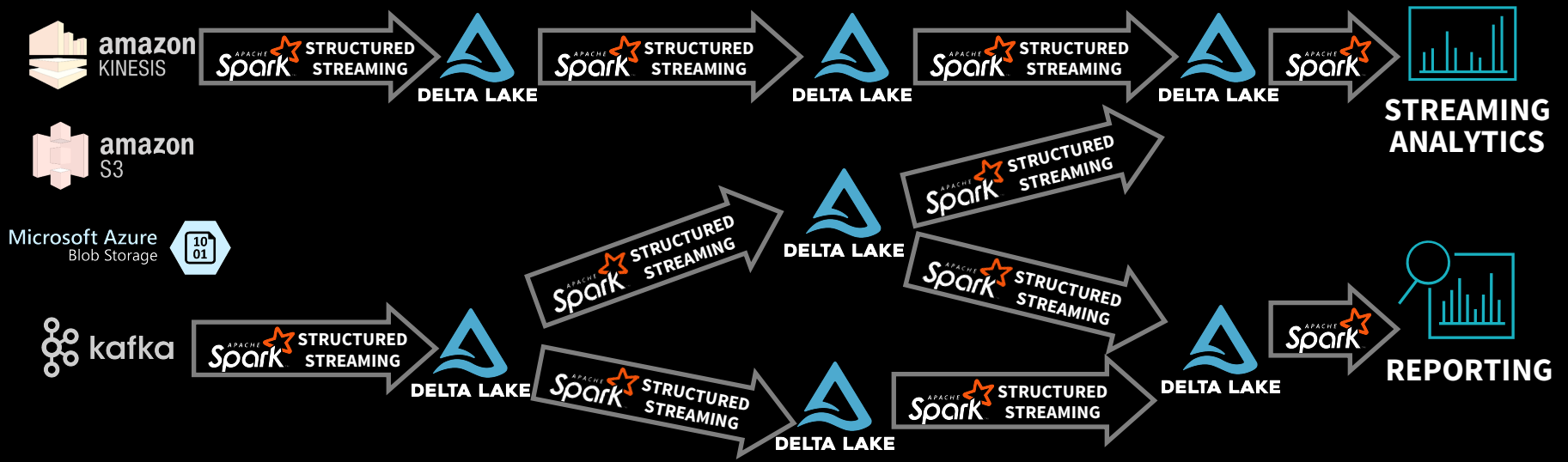


Compute once, write multiple times

```
streamingDataFrame.foreachBatch { batchSummaryData =>
  // write summary to Delta Lake
  // write summary to key value store
}.start()
```

Cheaper, but looses exactly-once guarantee

Production Pipelines @ databricks



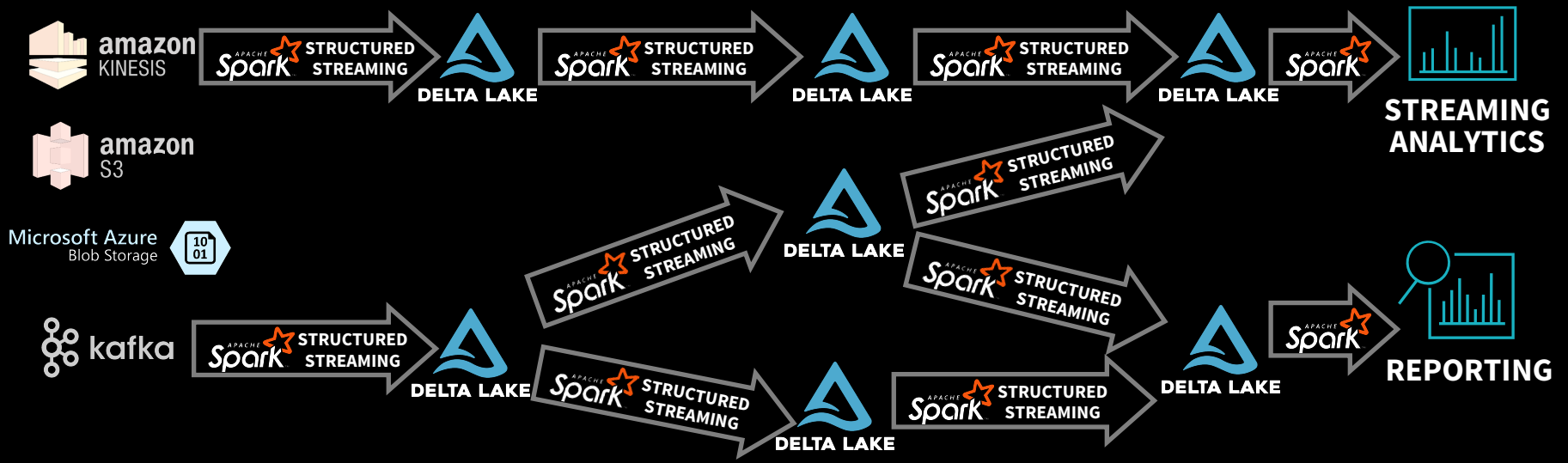
Production Pipelines @ databricks

Productionizing a single pipeline requires the following

- Unit testing
- Integration testing
- Deploying
- Monitoring
- Upgrading

See Burak's talk **Productizing Structured Streaming Jobs**

Production Pipelines @ databricks



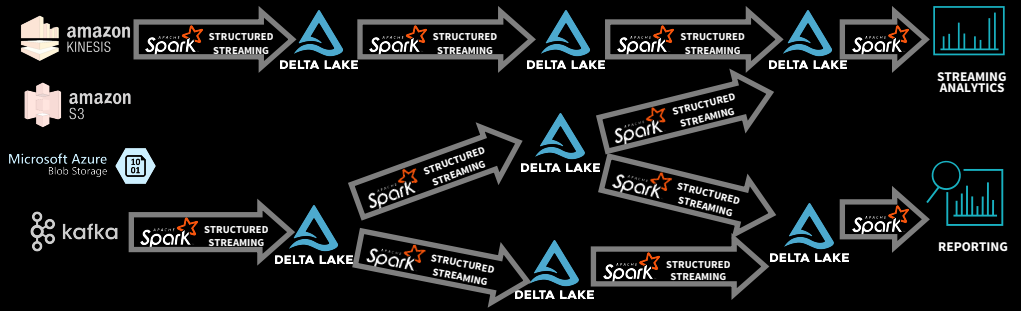
But we really need to productionize the entire DAG of pipelines!

Future direction

Declarative APIs to specify and manage entire DAGs of pipelines

Delta Pipelines

A small library on top of Apache Spark that lets users express their whole pipeline as a dataflow graph



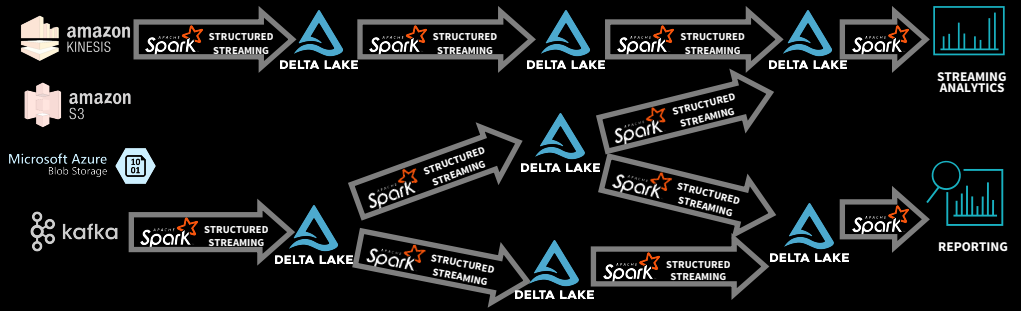

```
source(
  name = "raw",
  df = spark.readStream…)
```

```
sink(
  name = "warehouse",
  (df) => df.writeStream…
```

```
flow(
  name = "ingest",
  df = input("raw")
         .withColumn(…),
  dest = "warehouse",
  latency = "5 minutes")
```

Delta Pipelines

A small library on top of Apache Spark that lets users express their whole pipeline as a dataflow graph



Specify expectations with Delta tables

Unit test DAG with sample input data

Integration test DAG with production data

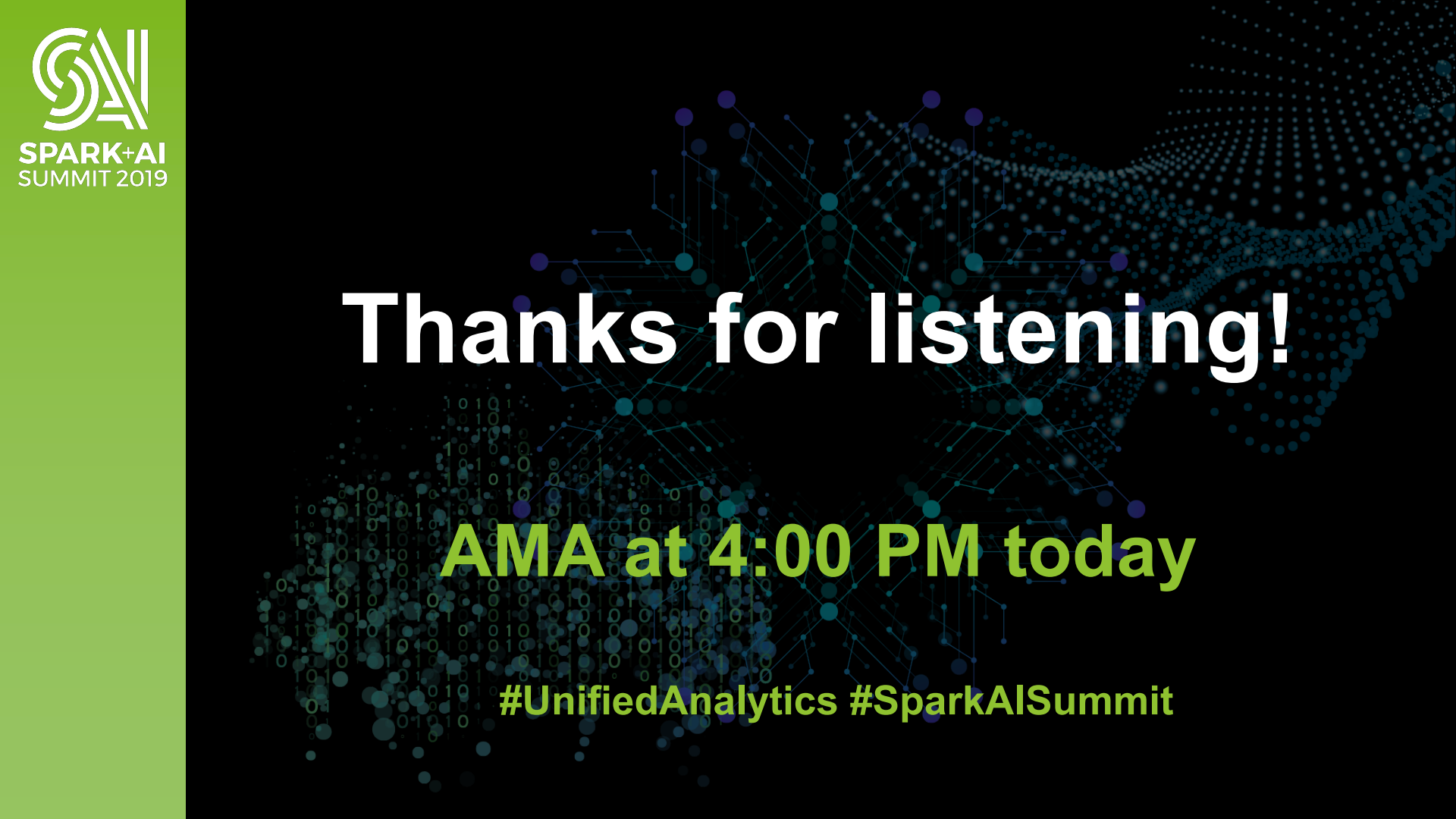
Deploy/upgrade new code of the entire DAG

Rollback code and/or data when needed

Monitor entire DAG in one place

STAY TUNED!


SPARK+AI
SUMMIT EUROPE
15 - 17 OCTOBER 2019 | AMSTERDAM | ORGANIZED BY databricks


SPARK+AI
SUMMIT 2019
Thanks for listening!
AMA at 4:00 PM today
#UnifiedAnalytics #SparkAISummit