Real-time Data Pipelines with Structured Streaming in Apache Spark™

Tathagata "*TD*" Das
@tathadas

DataEngConf 2018
18th April, San Francisco

databricks

About Me

Started Spark Streaming project in AMPLab, UC Berkeley

Currently focused on building Structured Streaming

PMC Member of Apache Spark

Engineer on the StreamTeam @ databricks

"we make all your streams come true"

Applications

Java python R Scala

Streaming SQL ML Graph

APACHE Spark

unified processing engine

Environments

Apache MESOS

amazon web services EC2

kubernetes

openstack

Microsoft Azure

Google Cloud Platform

hadoop YARN

Data Sources

kafka

S3

MySQL

hadoop

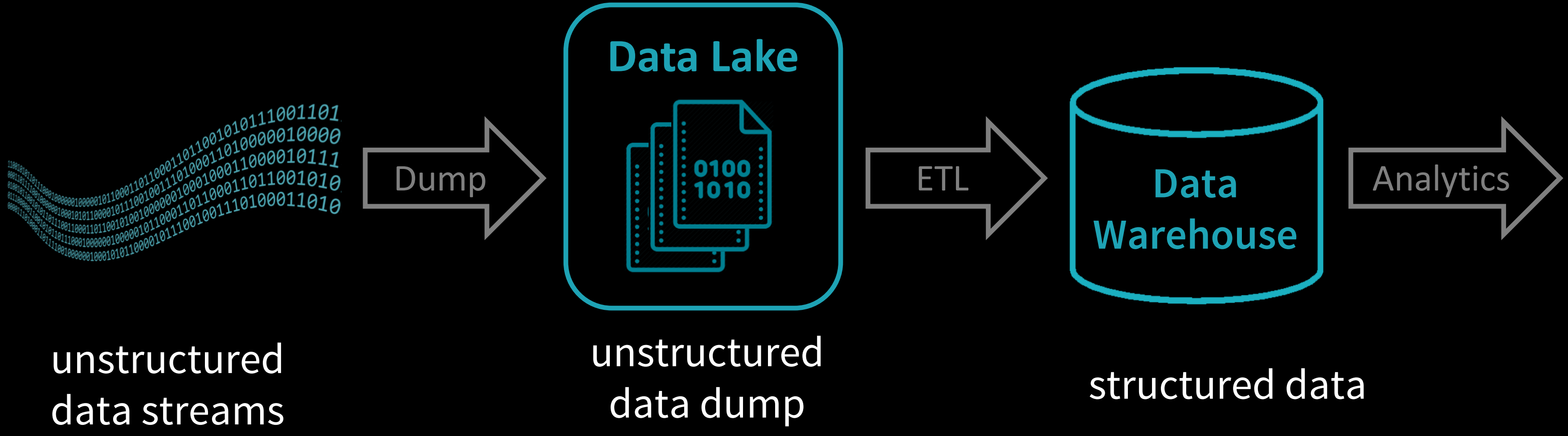
APACHE HBASE

AVRO

cassandra

Parquet

Data Pipelines – 10000ft view



Data Pipeline @ Fortune 100 Company

Trillions of Records

Security Infra
IDS/IPS, DLP, antivirus, load balancers, proxy servers

Cloud Infra & Apps
AWS, Azure, Google Cloud

Servers Infra
Linux, Unix, Windows

Network Infra
Routers, switches, WAPs, databases, LDAP

Dump

Messy data not ready for analytics

DATALAKE1

DATALAKE2

Complex ETL

Separate warehouses for each type of analytics

DW1 Incidence Response

DW2 Alerting

DW3 Reports

Hours of delay in accessing data

Very expensive to scale

Proprietary formats

No advanced analytics (ML)

New Pipeline @ Fortune 100 Company

Apache Spark STRUCTURED STREAMING → databricks DELTA → Apache Spark SQL, ML, STREAMING

Incidence Response

Alerting

Reports

Data usable in minutes/seconds

Easy to scale

Open formats

Enables advanced analytics

databricks

STRUCTURED STREAMING

you

should not have to

reason about streaming

you

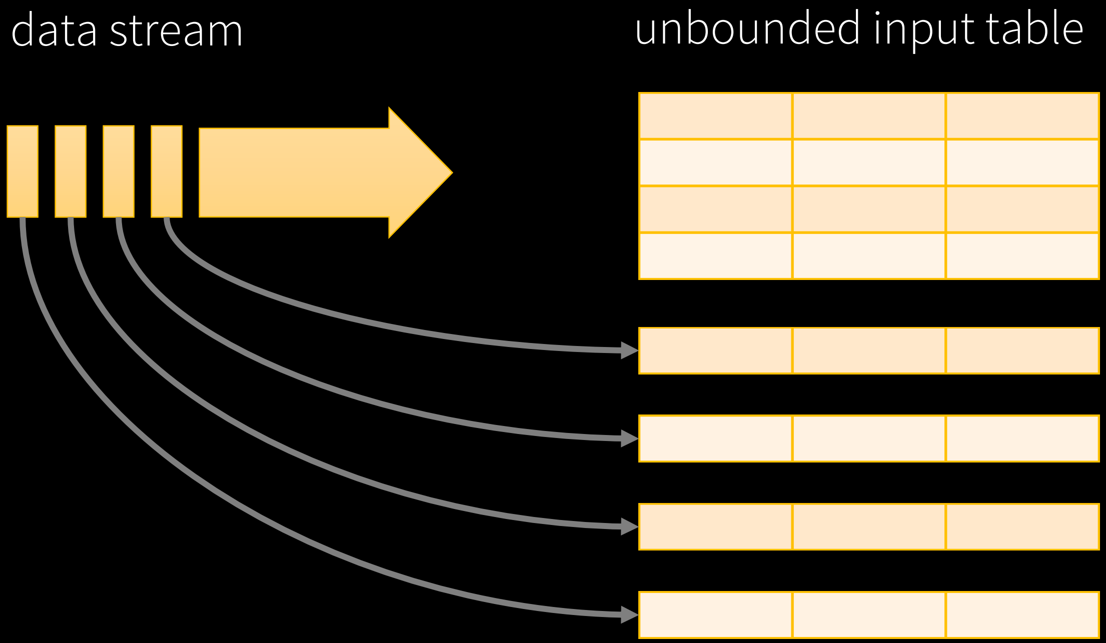
should write simple queries

&

Spark

should continuously update the answer

Treat Streams as Unbounded Tables



new data in the
data stream
=
new rows appended
to a unbounded table

databricks

Anatomy of a Streaming Query

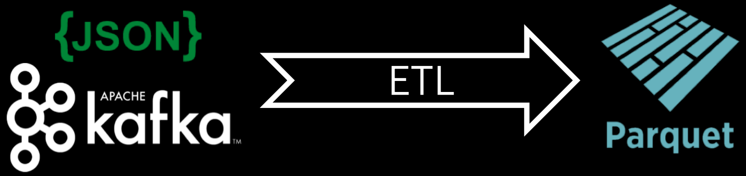
Example

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()
```

returns a DataFrame

Source

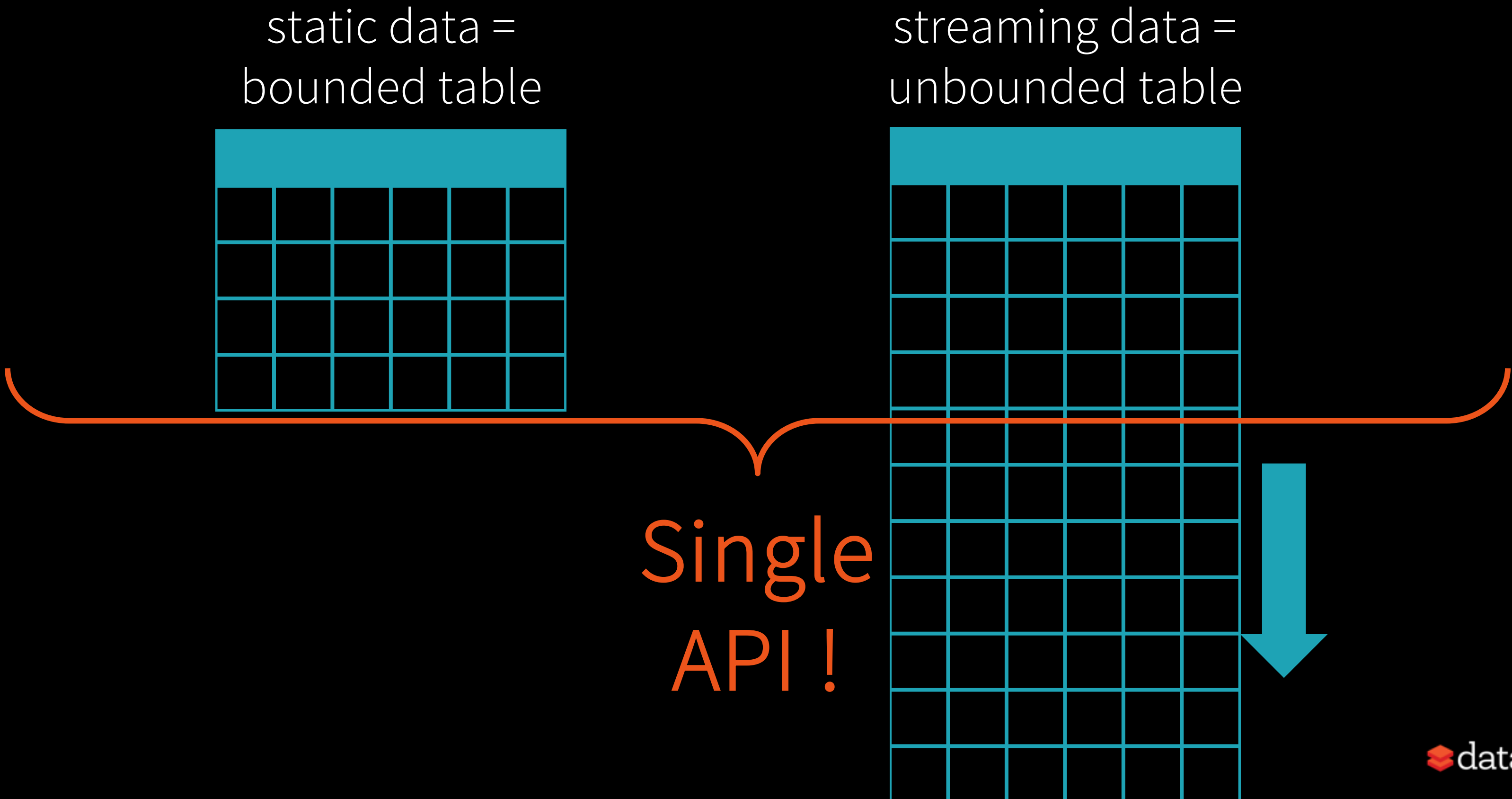
Specify where to read data from

Built-in support for Files / Kafka / Kinesis*

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

*Available only on [Databricks Runtime]

DataFrame ⇔ Table



DataFrame/Dataset

SQL

```
spark.sql("
  SELECT type, sum(signal)
  FROM devices
  GROUP BY type
")
```

Most familiar to BI Analysts

Supports SQL-2003, HiveQL

DataFrame

```
val df: DataFrame =
  spark.table("device-data")
    .groupBy("type")
    .sum("signal"))
```

Great for Data Scientists familiar with Pandas, R Dataframes

Dataset

```
val ds: Dataset[(String, Double)] =
  spark.table("device-data")
    .as[DeviceData]
    .groupByKey(_.type)
    .mapValues(_.signal)
    .reduceGroups(_ + _)
```

Great for Data Engineers who want compile-time type safety

Same semantics, same performance

Choose your hammer for whatever nail you have!

Anatomy of a Streaming Query

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

Kafka DataFrame

key	value	topic	partition	offset	timestamp
[binary]	*[binary]*	"topic"	0	345	1486087873
[binary]	*[binary]*	"topic"	3	2890	1486086721

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 records 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

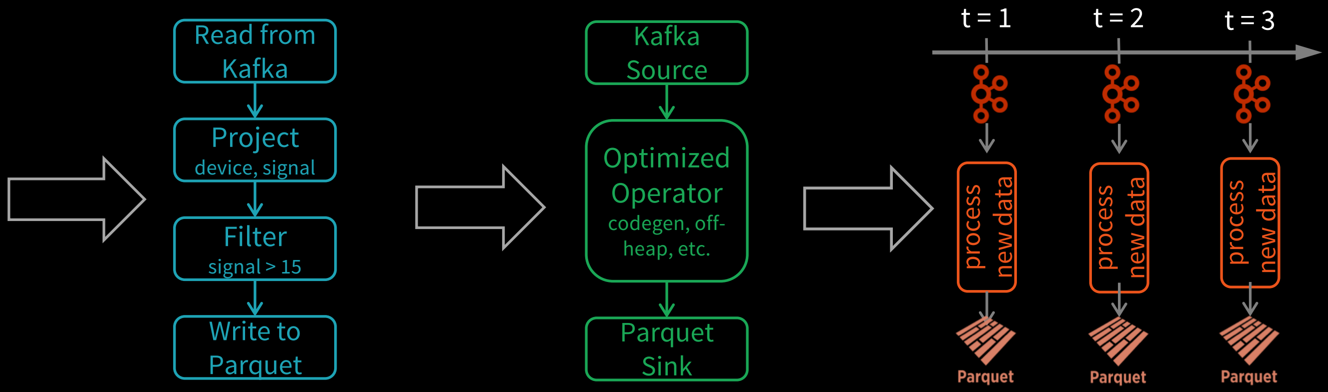
Trigger: when to process data

- 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

databricks

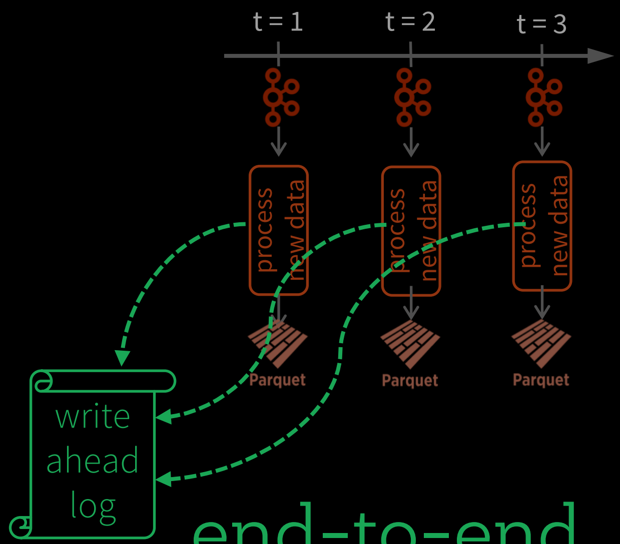
Fault-tolerance with Checkpointing

Checkpointing

Saves processed offset info to stable storage
  Saved as JSON for forward-compatibility

Allows recovery from any failure

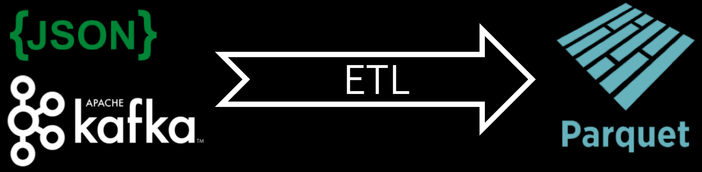
  Can resume after limited changes to your streaming transformations (e.g. adding new filters to drop corrupted data, etc.)



end-to-end exactly-once guarantees

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()
```

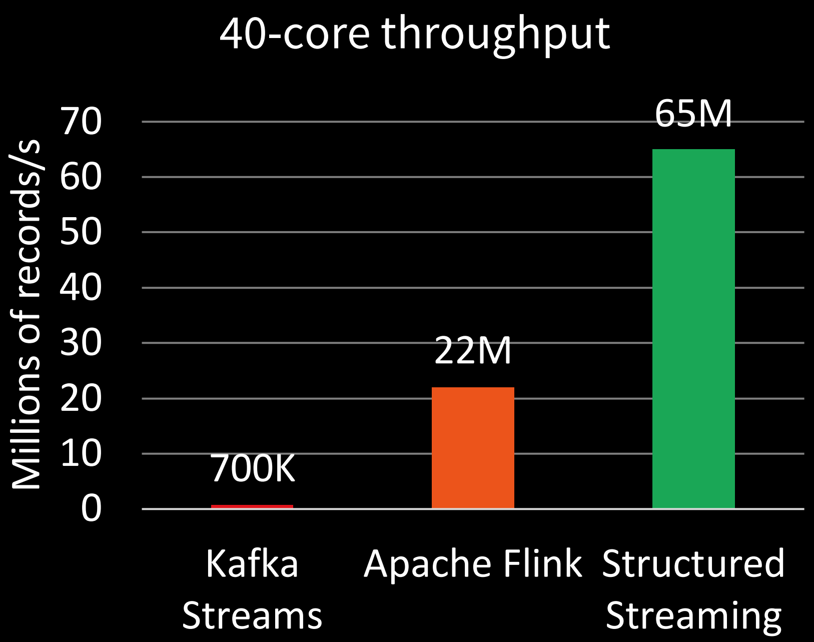


Raw data from Kafka available as structured data in seconds, ready for querying

Performance: YAHOO! Benchmark

Structured Streaming reuses the **Spark SQL Optimizer** and **Tungsten Engine**

3x cheaper



More details in our [blog post]

Business Logic independent of Execution Mode

```
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()
```

Business logic

Business Logic independent of Execution Mode

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

Business logic remains unchanged

Peripheral code decides whether it's a batch or a streaming query

databricks

Business Logic independent of Execution Mode

```
.selectExpr("cast (value as string) as json")
.select(from_json("json", schema).as("data"))
```

Batch

high latency
(hours/minutes)

execute on-demand

high throughput

Micro-batch
Streaming

low latency
(seconds)

efficient resource allocation

high throughput

Continuous**
Streaming

ultra-low latency
(milliseconds)

static resource allocation

**experimental release in Spark 2.3, read our blog

Event time Aggregations

Windowing is just another type of grouping in Struct. Streaming

number of records every hour

```
parsedData
      .groupBy(window("timestamp","1 hour"))
      .count()
```

avg signal strength of each device every 10 mins

```
parsedData
      .groupBy(
            "device",
            window("timestamp","10 mins"))
      .avg("signal")
```

Support UDAFs!

databricks

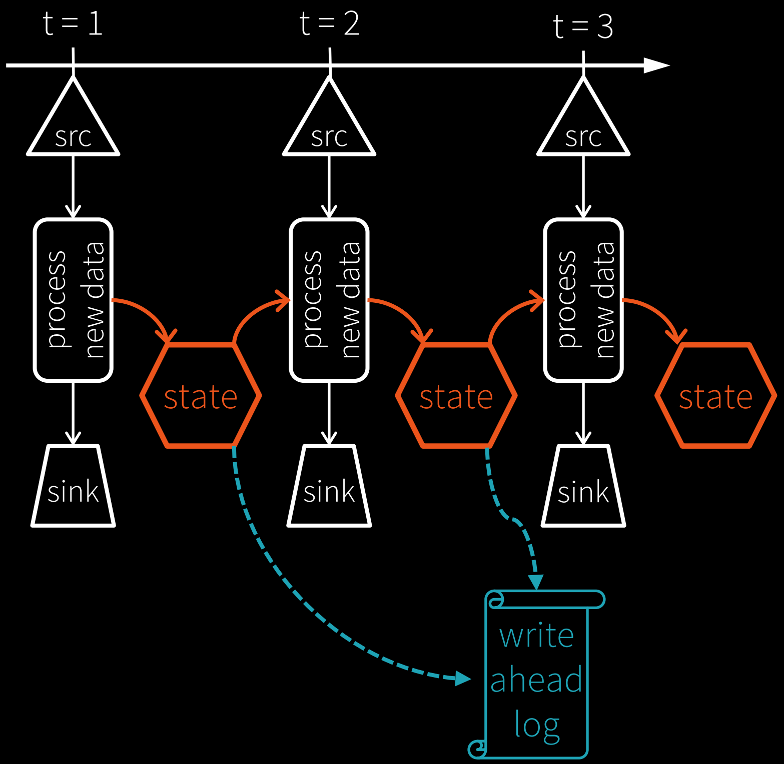
Stateful Processing for Aggregations

Aggregates has to be saved as distributed state between triggers

Each trigger reads previous state and writes updated state

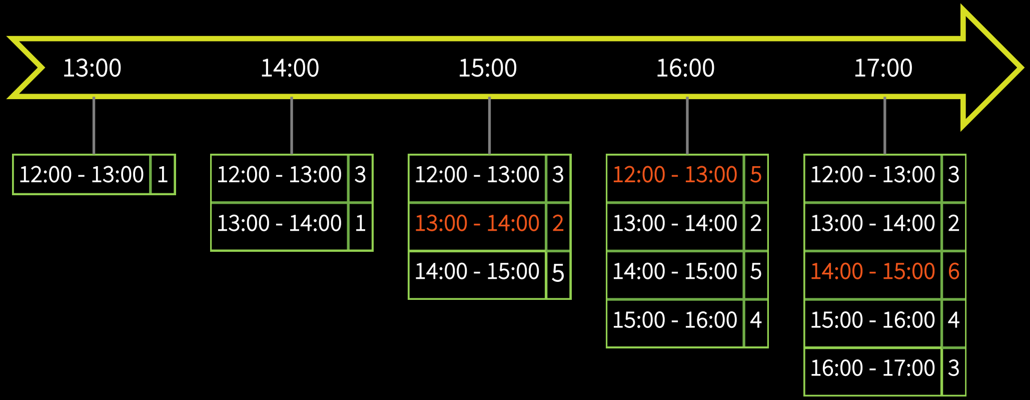
State stored in memory, backed by *write ahead log* in HDFS

Fault-tolerant, exactly-once guarantee!



Automatically handles Late Data

Keeping state allows late data to update counts of old windows



red = state updated with late data

But size of the state increases indefinitely if old windows are not dropped

databricks

Watermarking

Watermark - moving threshold of how late data is expected to be and when to drop old state

Trails behind max event time seen by the engine

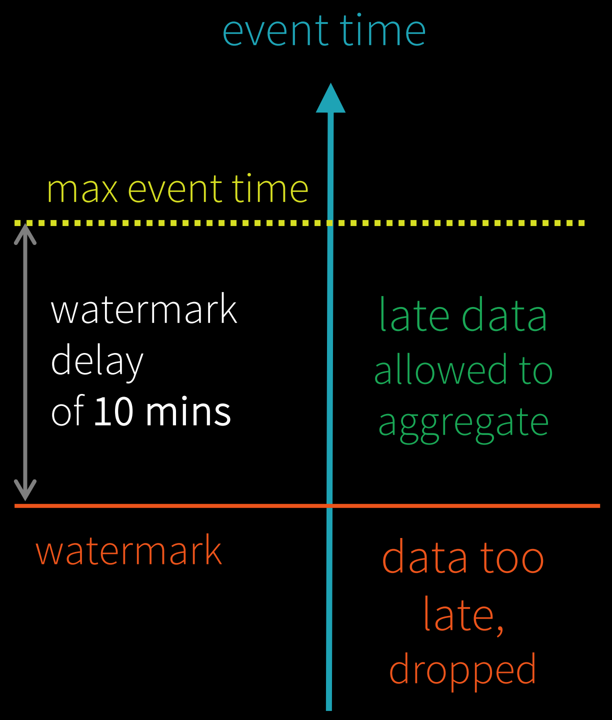
Watermark delay = trailing gap

Watermarking

Data newer than watermark may be late, but allowed to aggregate

Data older than watermark is "too late" and dropped

Windows older than watermark automatically deleted to limit the amount of intermediate state

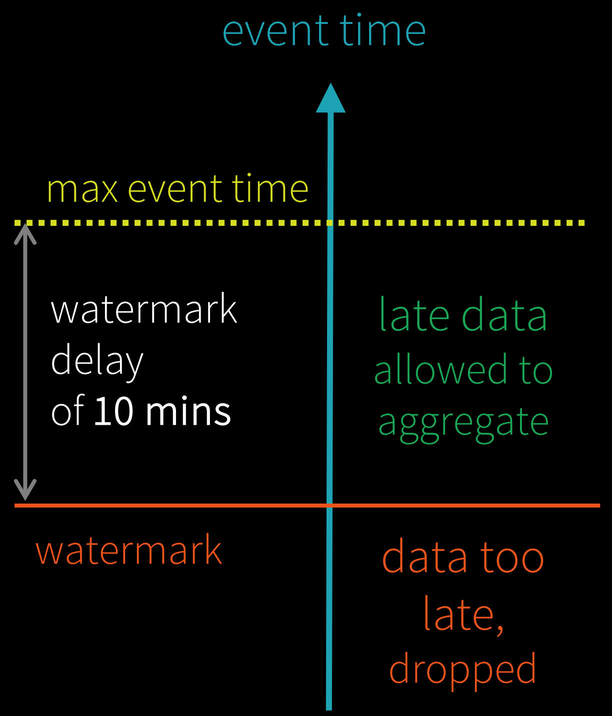


Watermarking

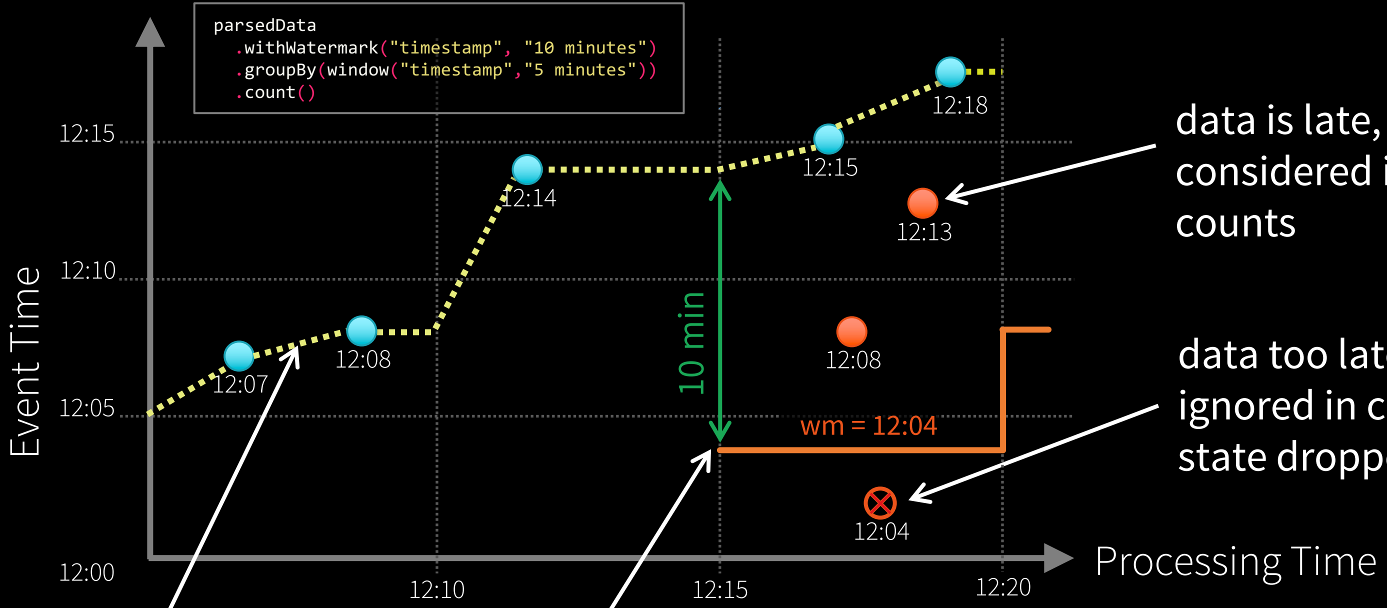
```
parsedData
   .withWatermark("timestamp", "10 minutes")
   .groupBy(window("timestamp","5 minutes"))
   .count()
```

Useful only in stateful operations

Ignored in non-stateful streaming queries and batch queries



Watermarking



data is late, but considered in counts

data too late, ignored in counts, state dropped

system tracks max observed event time

watermark updated to
12:14 - 10m = 12:04
for next trigger,
state < 12:04 deleted

More details in my blog post

Other Interesting Operations

Streaming Deduplication

```
parsedData.dropDuplicates("eventId")
```

Joins

- Stream-batch joins
- Stream-stream joins

```
stream1.join(stream2, "device")
```

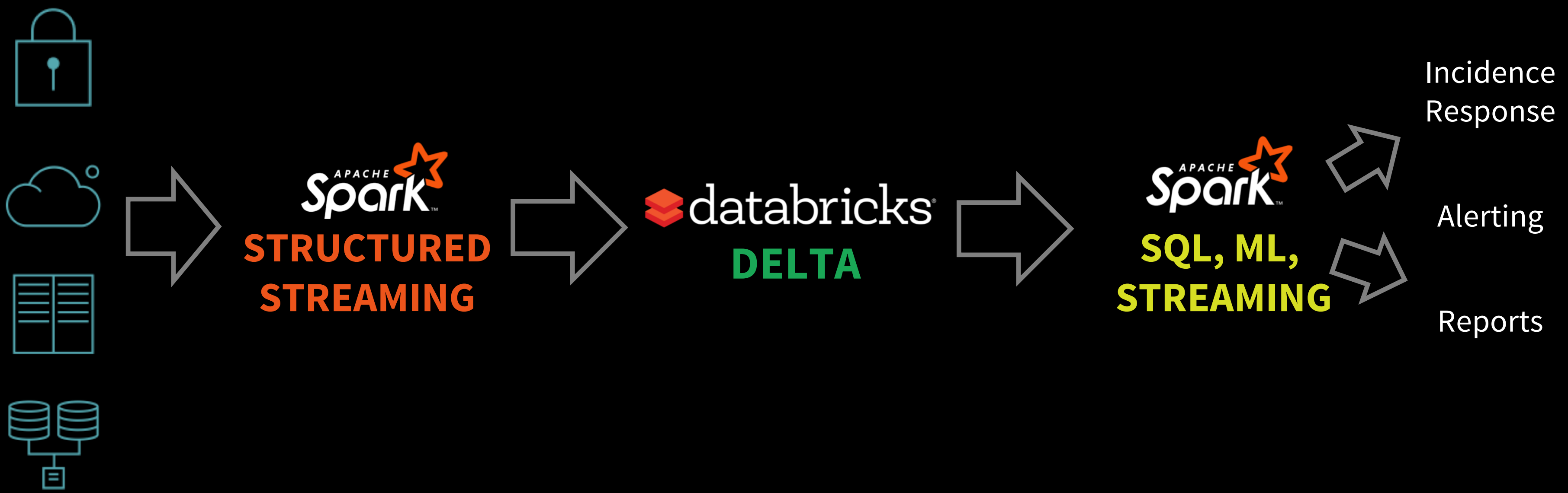
Arbitrary Stateful Processing

- [map|flatMap]GroupsWithState

```
ds.groupByKey(_.id)
  .mapGroupsWithState
    (timeoutConf)
    (mappingWithStateFunc)
```

See my previous Spark Summit talk and blog posts (here and here)

Data Pipeline with Apache Spark

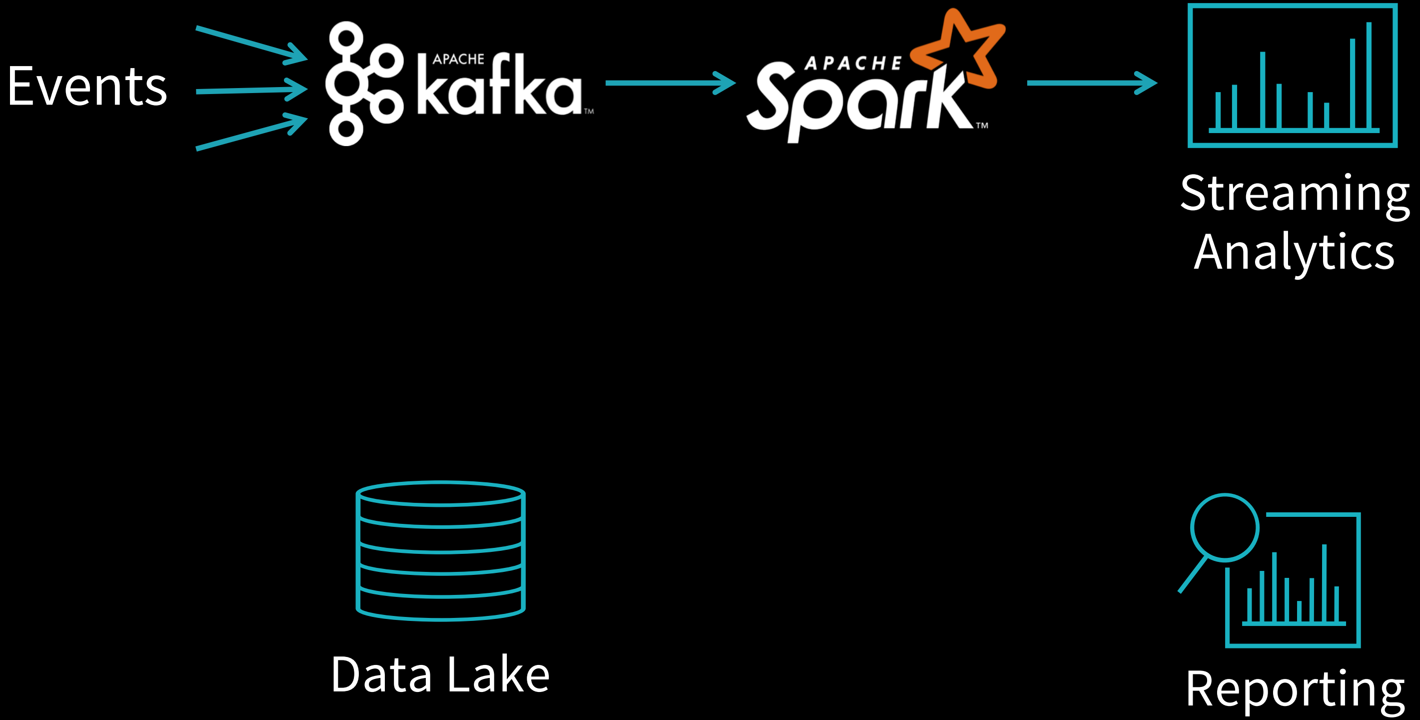


ETL @ databricks

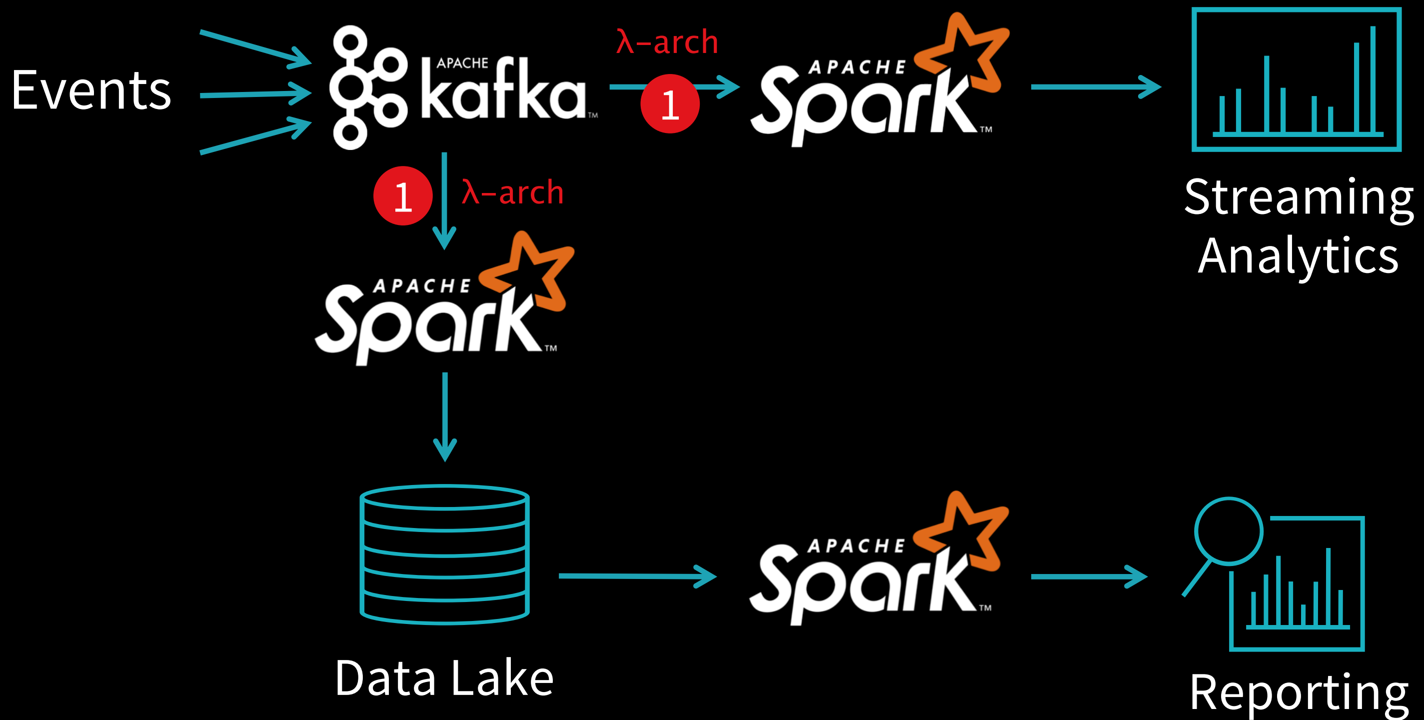
Evolution of a Cutting-Edge Data Pipeline

databricks

Evolution of a Cutting-Edge Data Pipeline

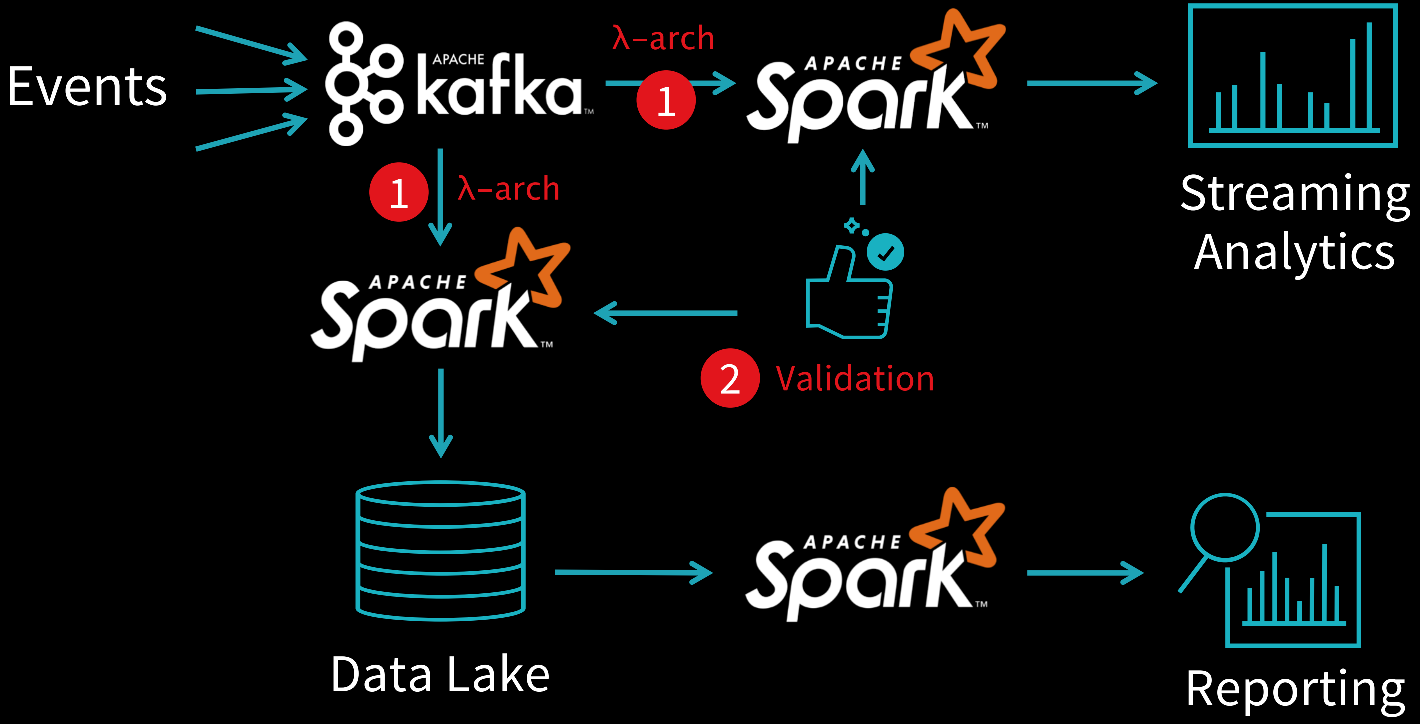


Challenge #1: Historical Queries?



1 λ-arch

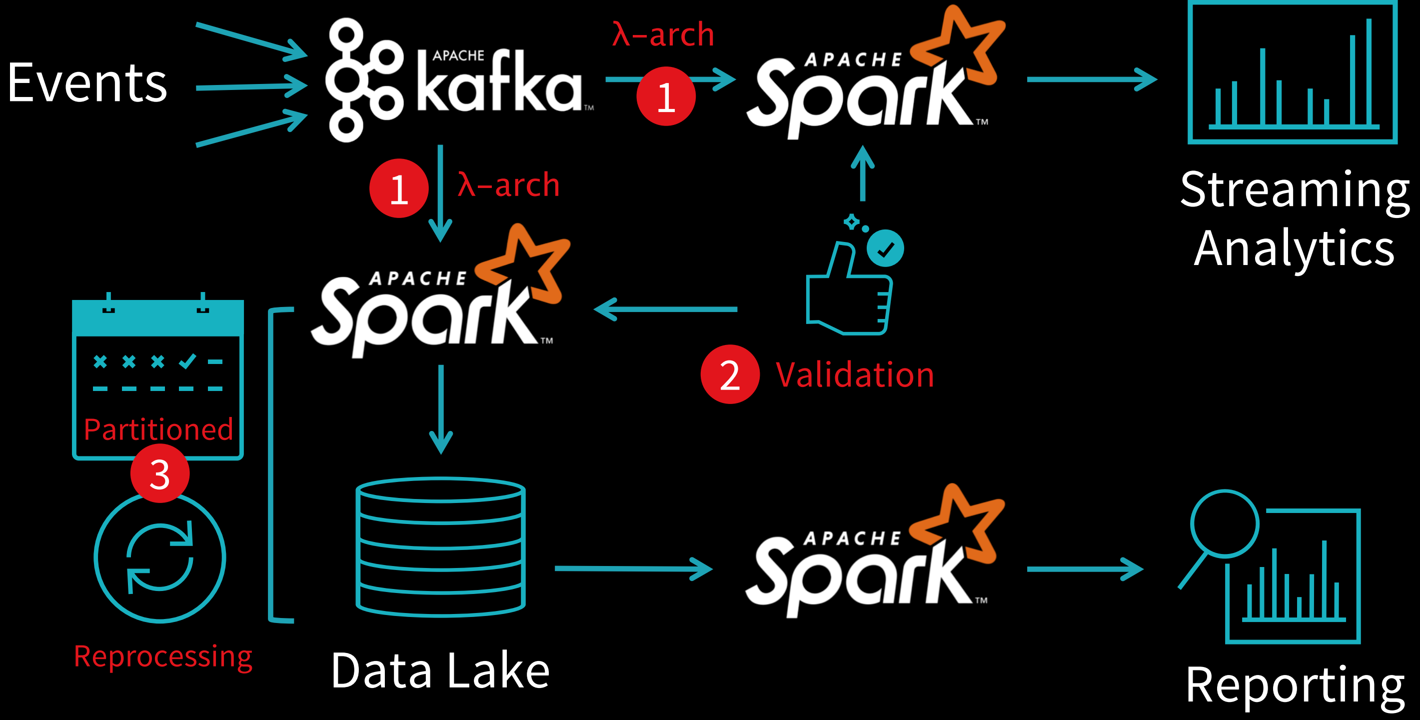
Challenge #2: Messy Data?



1 λ-arch

2 Validation

Challenge #3: Mistakes and Failures?

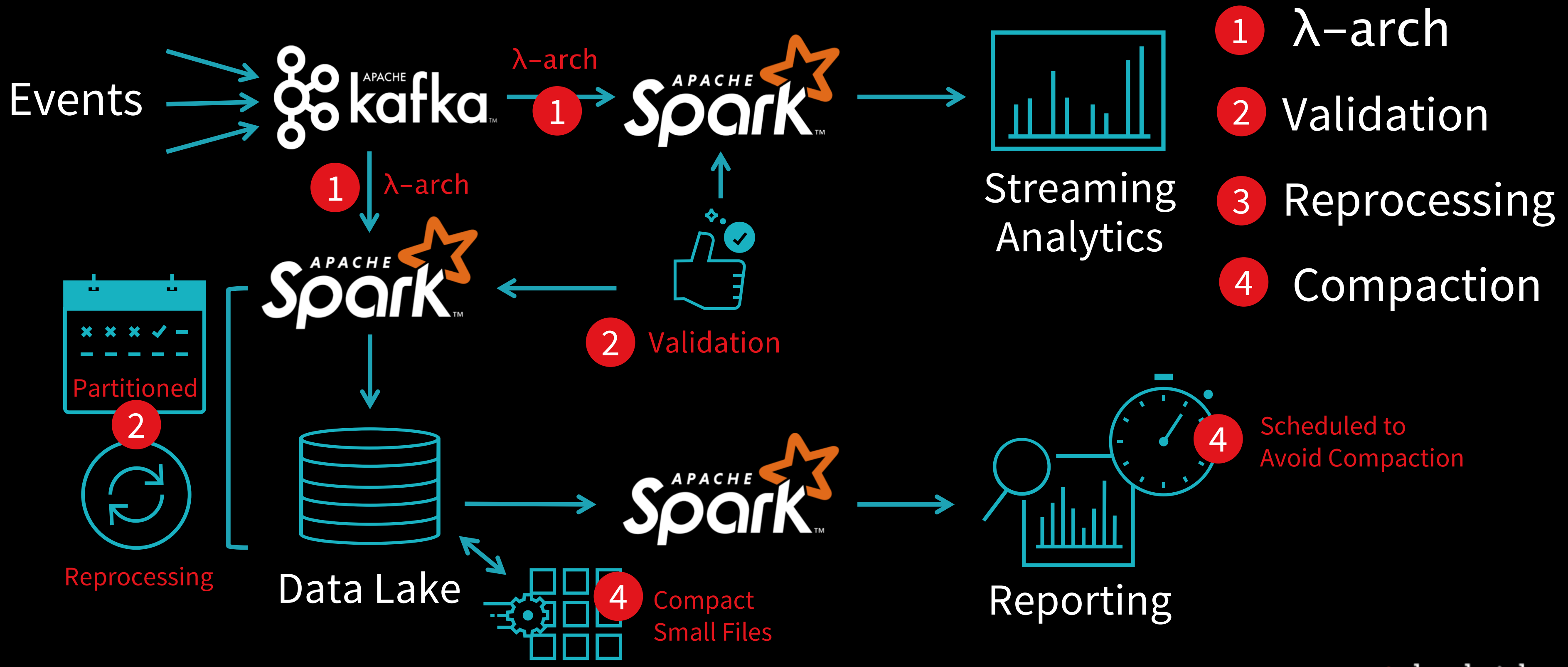


1 λ-arch

2 Validation

3 Reprocessing

Challenge #4: Query Performance?



Let's try it instead with

databricks

DELTA

databricks

DELTA

The
SCALE
of data lake

The
RELIABILITY & PERFORMANCE
of data warehouse

The
LOW-LATENCY
of streaming

THE GOOD OF DATA WAREHOUSES

- Pristine Data
- Transactional Reliability
- Fast Queries

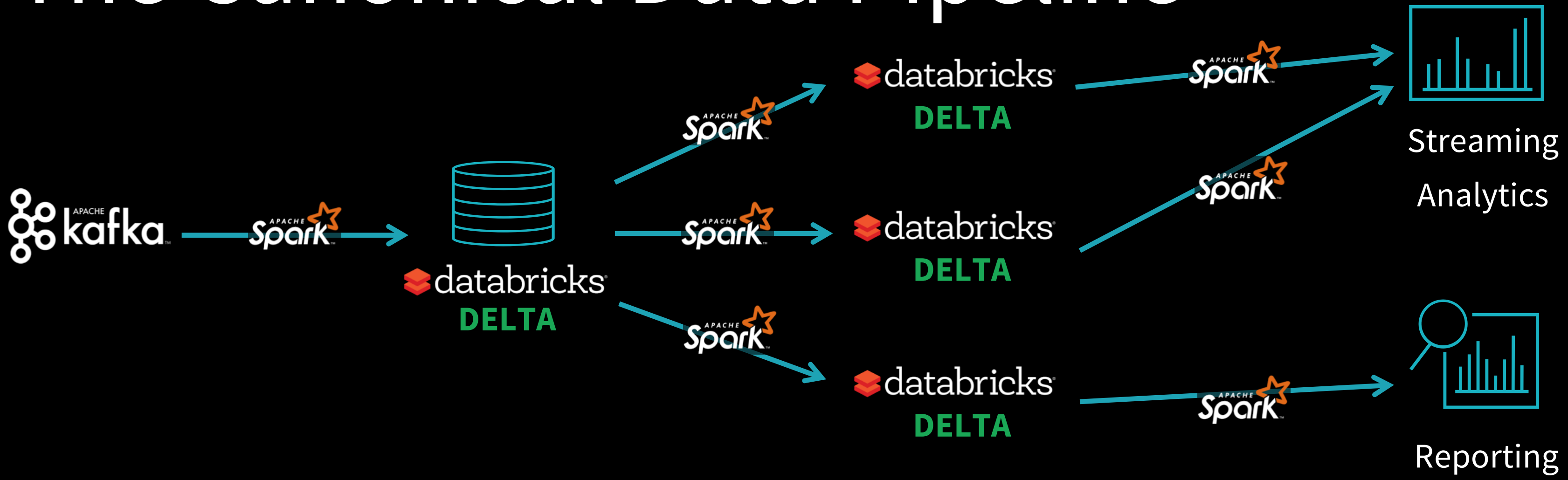
THE GOOD OF DATA LAKES

- Massive scale on cloud storage
- Open Formats (Parquet, ORC)
- Predictions (ML) & Streaming

Databricks Delta Combines the Best

MASSIVE SCALE	Decouple Compute & Storage
RELIABILITY	ACID Transactions & Data Validation
PERFORMANCE	Data Indexing & Caching (10-100x)
OPEN	Data stored as Parquet, ORC, etc.
LOW-LATENCY	Integrated with Structured Streaming

The Canonical Data Pipeline



1. ~~λ arch~~ → Not needed, Delta handles both short and long term data
2. Validation ✓ → Easy as data in short term and long term data in one location
3. Reprocessing ✓
4. Compaction ✓

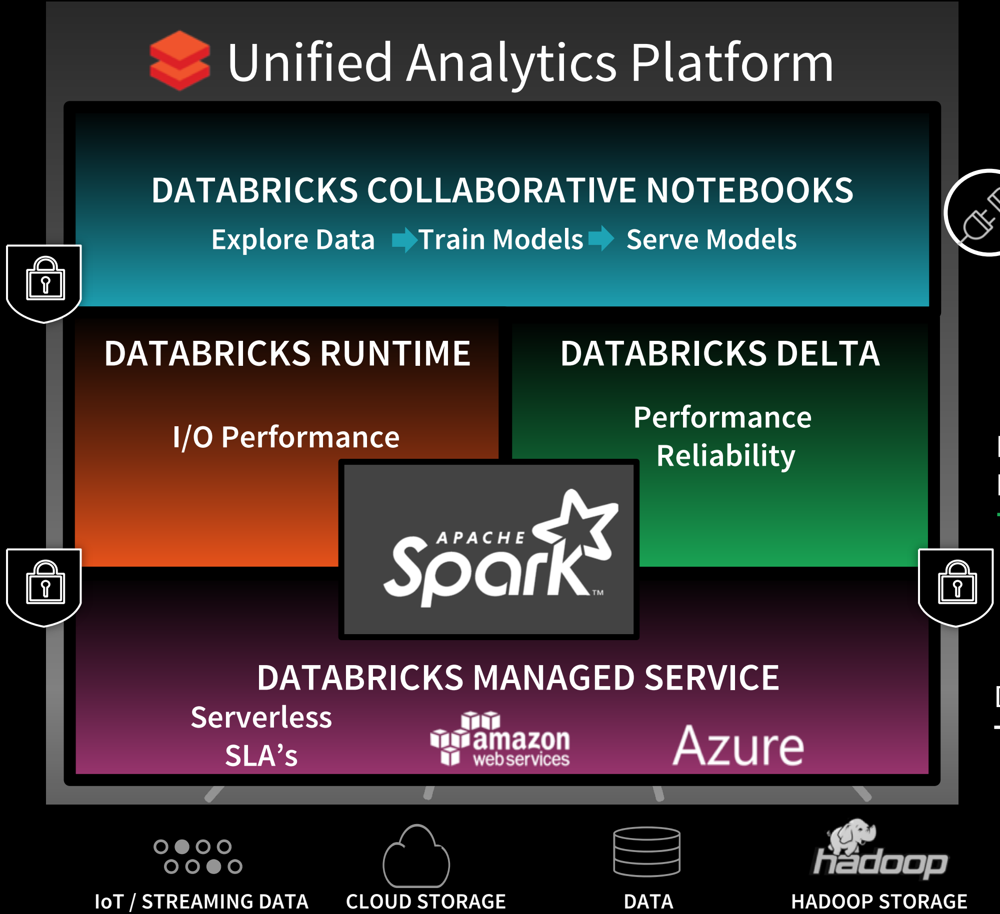
→ Easy and seamless with Detla's transactional guarantees

Accelerate Innovation with Databricks

Increases Data Science Productivity by 5x

Improves Performance by 10-20X over Apache Spark

Removes Devops & Infrastructure Complexity



Open Extensible API's

+ableau looker

Higher Performance & Reliability for your Data Lake

Databricks Enterprise Security

databricks

Data Pipelines with Spark and DELTA

fast, scalable, fault-tolerant stream processing with high-level, user-friendly APIs

Spark
STRUCTURED STREAMING

databricks
DELTA

data storage solution with the reliability of data warehouses and the scalability of data lakes

Spark
SQL, ML, STREAMING

Incidence Response

Alerting

Reports

More Info

Structured Streaming Programming Guide

http://spark.apache.org/docs/latest/structured-streaming-programming-guide.html

Databricks blog posts for more focused discussions on streaming

https://databricks.com/blog/category/engineering/streaming

Databricks Delta

https://databricks.com/product/databricks-delta

VIDEO ARCHIVE

SAIS 2018 | SCHEDULE | TRAINING | VENUE + HOTEL | SPONSORS

ORGANIZED BY databricks

REGISTER TODAY

SPARK+AI SUMMIT 2018

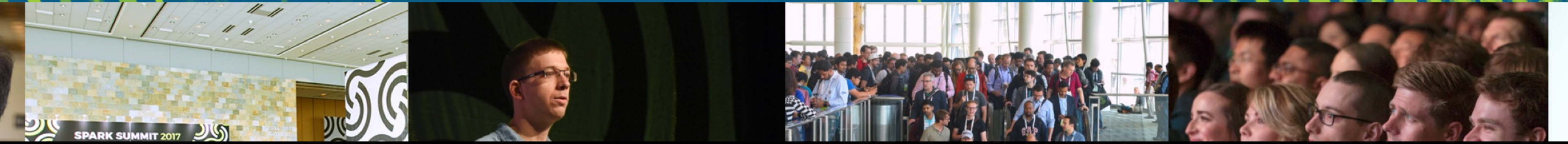
JUNE 4 – 6 | SAN FRANCISCO

ORGANIZED BY databricks

COUNTDOWN TO SUMMIT

47 DAYS | 07 HOURS | 26 MINUTES

REGISTER TODAY



Thank you!

@tathadas