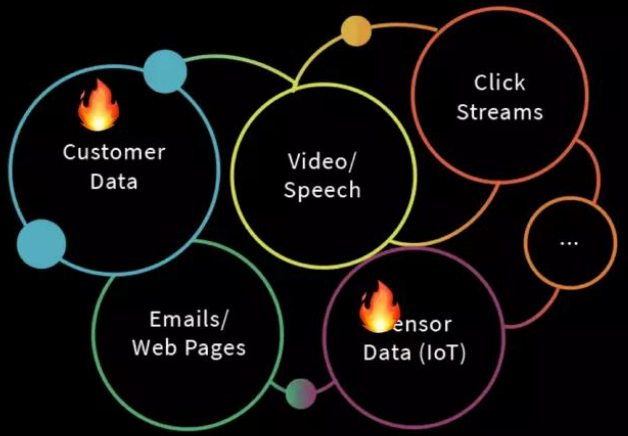
Making Apache Spark™ Better with Delta Lake

Michael Armbrust
@michaelarmbrust

The Promise of the Data Lake

1. Collect Everything



Garbage In

2. Store it all in the Data Lake

Garbage Stored

3. Data Science & Machine Learning

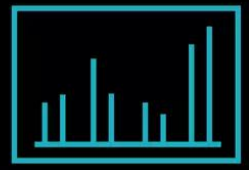
- Recommendation Engines
- Risk, Fraud Detection
- IoT & Predictive Maintenance
- Genomics & DNA Sequencing

Garbage Out

What does a typical
data lake project look like?

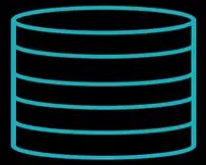
Evolution of a Cutting-Edge Data Lake

Events



Streaming Analytics

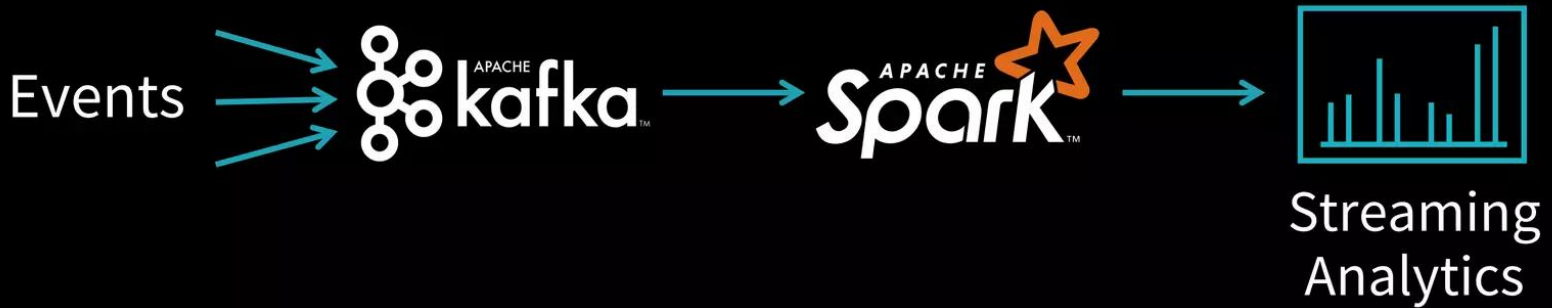
?



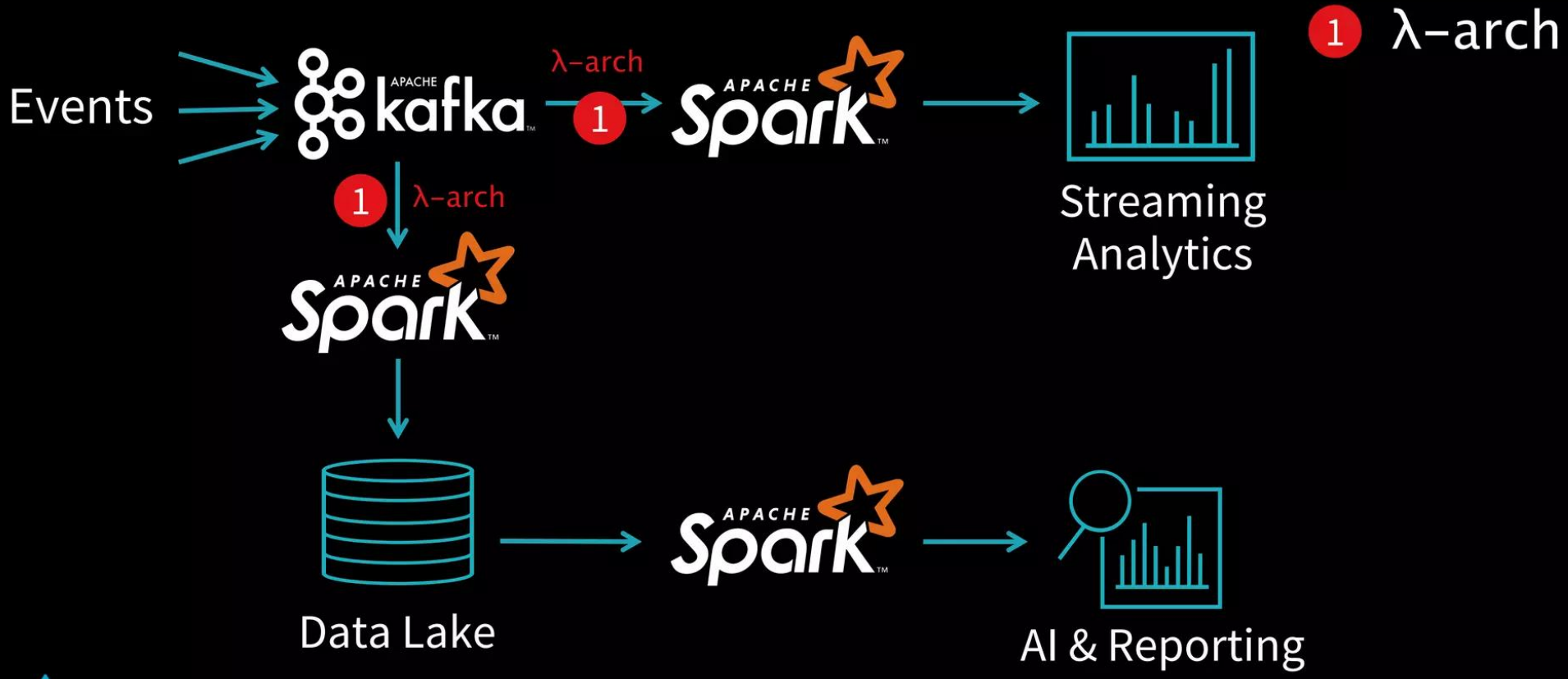
Data Lake

AI & Reporting

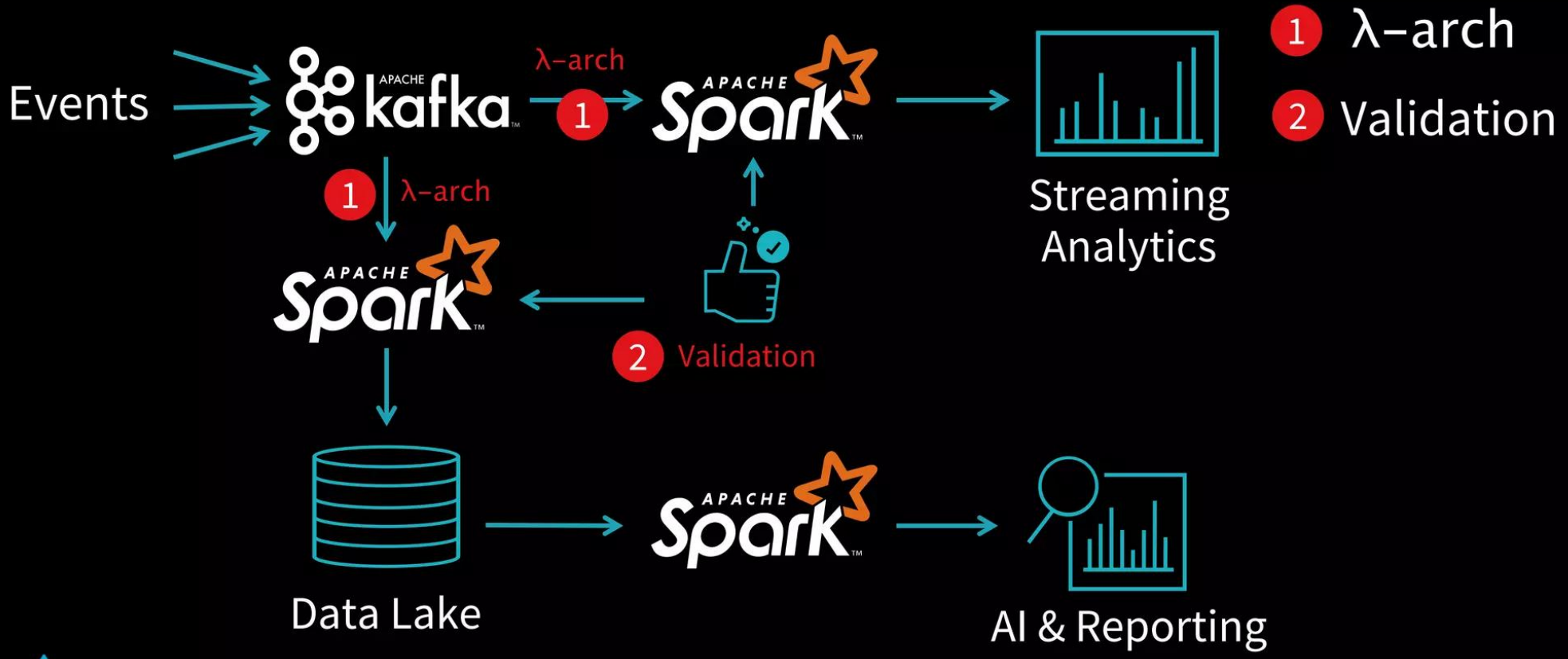
Evolution of a Cutting-Edge Data Lake



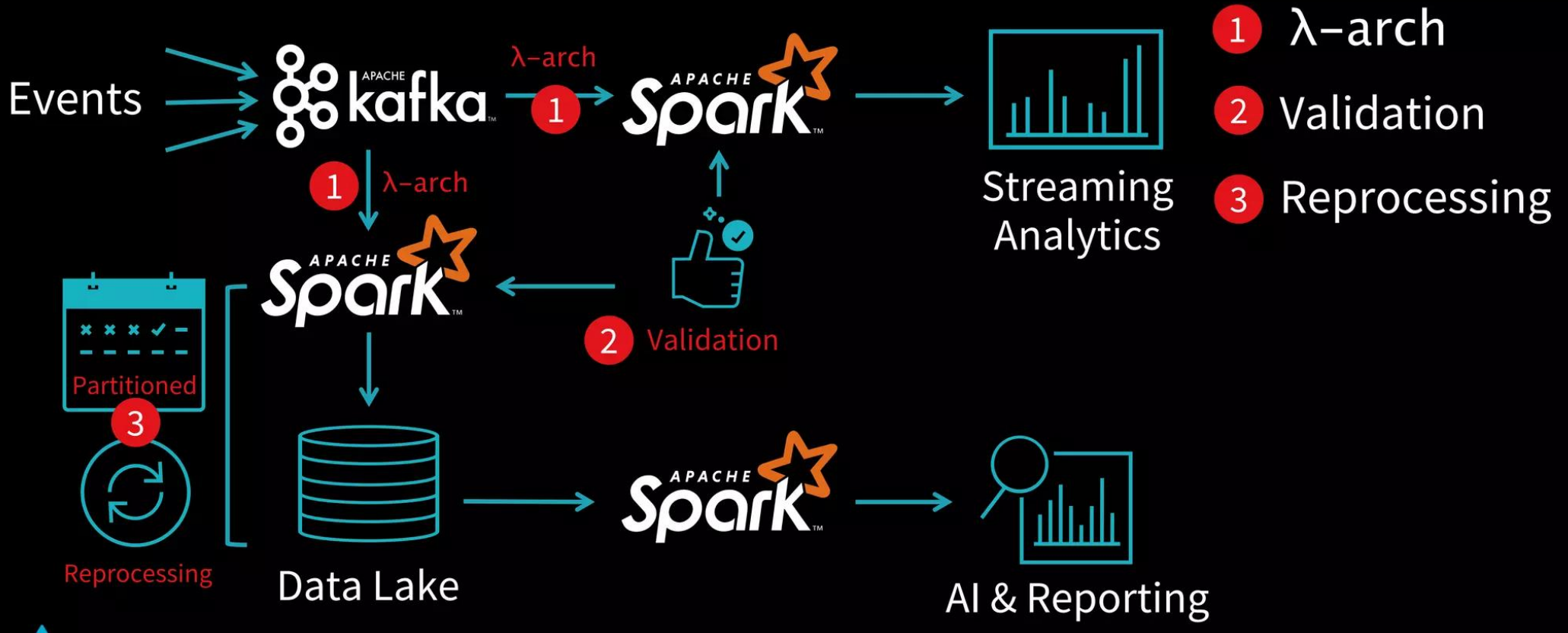
Challenge #1: Historical Queries?



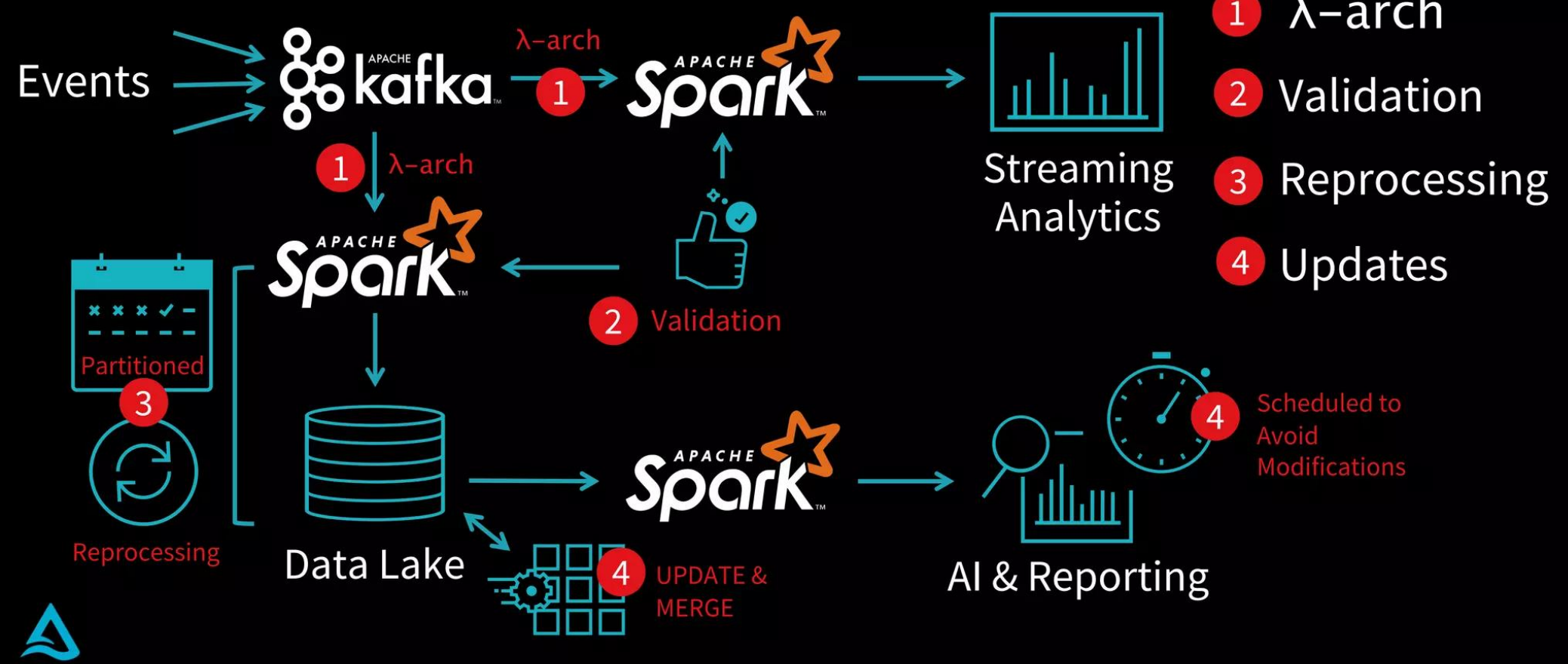
Challenge #2: Messy Data?



Challenge #3: Mistakes and Failures?



Challenge #4: Updates?



Wasting Time & Money

Solving Systems Problems

Instead of Extracting Value From Data

Data Lake Distractions

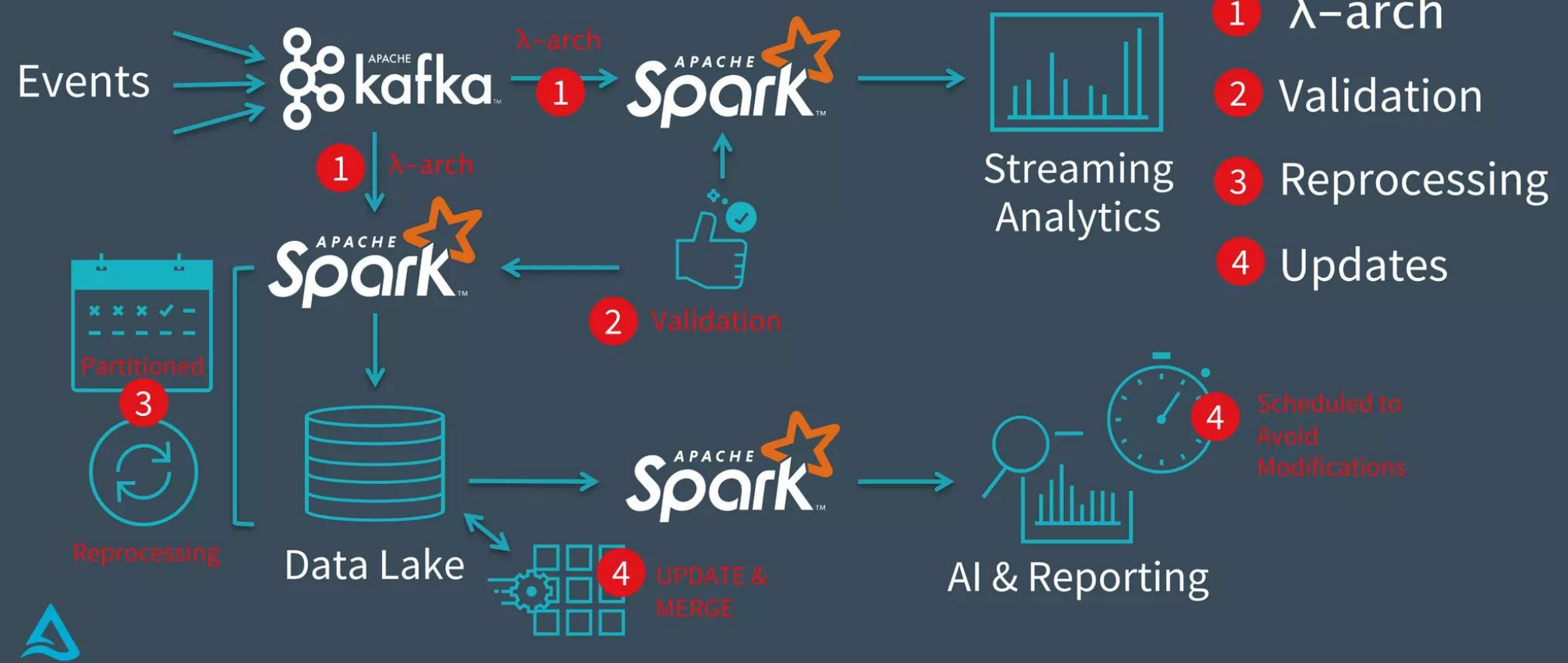
No atomicity means failed production jobs leave data in corrupt state requiring tedious recovery

No quality enforcement creates inconsistent and unusable data

No consistency / isolation makes it almost impossible to mix appends and reads, batch and streaming

Let’s try it instead with

Challenges of the Data Lake



The DELTA LAKE Architecture

The DELTA LAKE Architecture

Full ACID Transaction

Focus on your data flow, instead of worrying about failures.

The DELTA LAKE Architecture

Open Standards, Open Source (Apache License)

Store petabytes of data without worries of lock-in. Growing community including Presto, Spark and more.

The DELTA LAKE Architecture

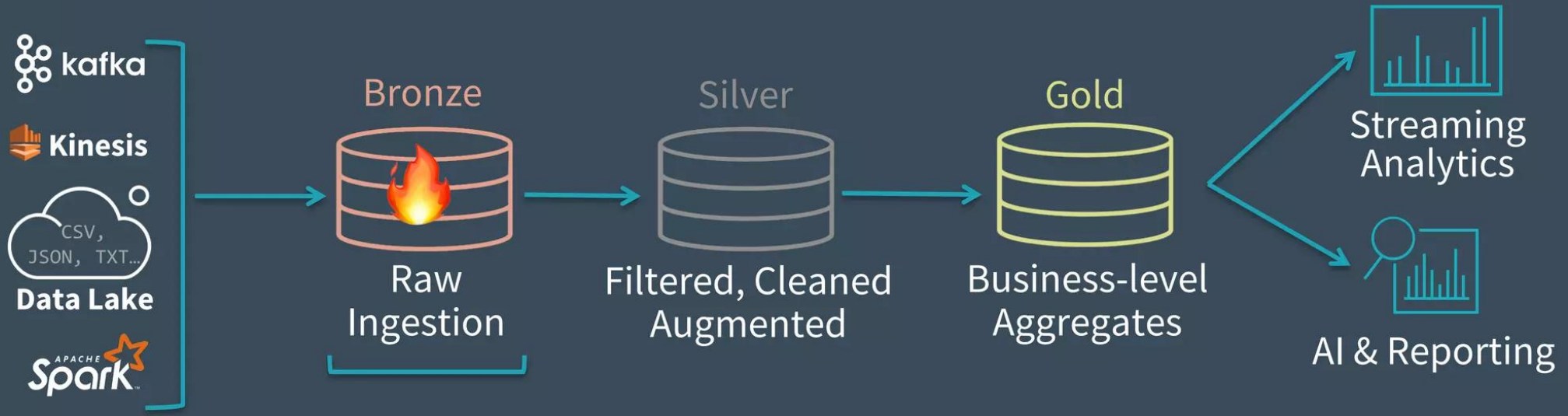
Powered by APACHE Spark™

Unifies Streaming / Batch. Convert existing jobs with minimal modifications.

The DELTA LAKE

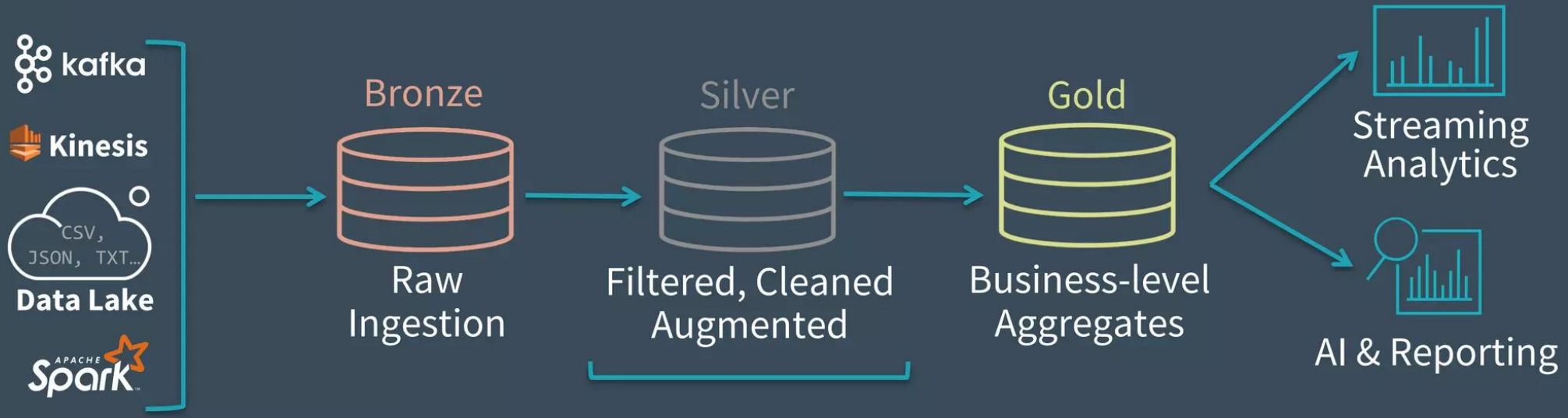
Delta Lake allows you to *incrementally* improve the quality of your data until it is ready for consumption.

The DELTA LAKE



- Dumping ground for raw data
- Often with long retention (years)
- Avoid error-prone parsing

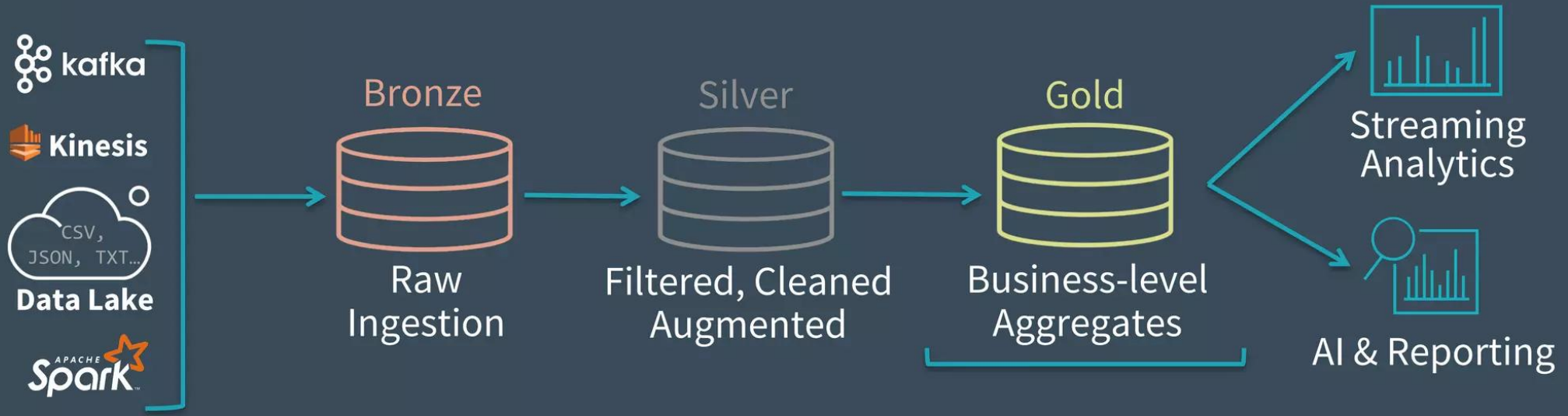
The DELTA LAKE



Intermediate data with some cleanup applied.

Queryable for easy debugging!

The DELTA LAKE



Clean data, ready for consumption.

Read with Spark or Presto*

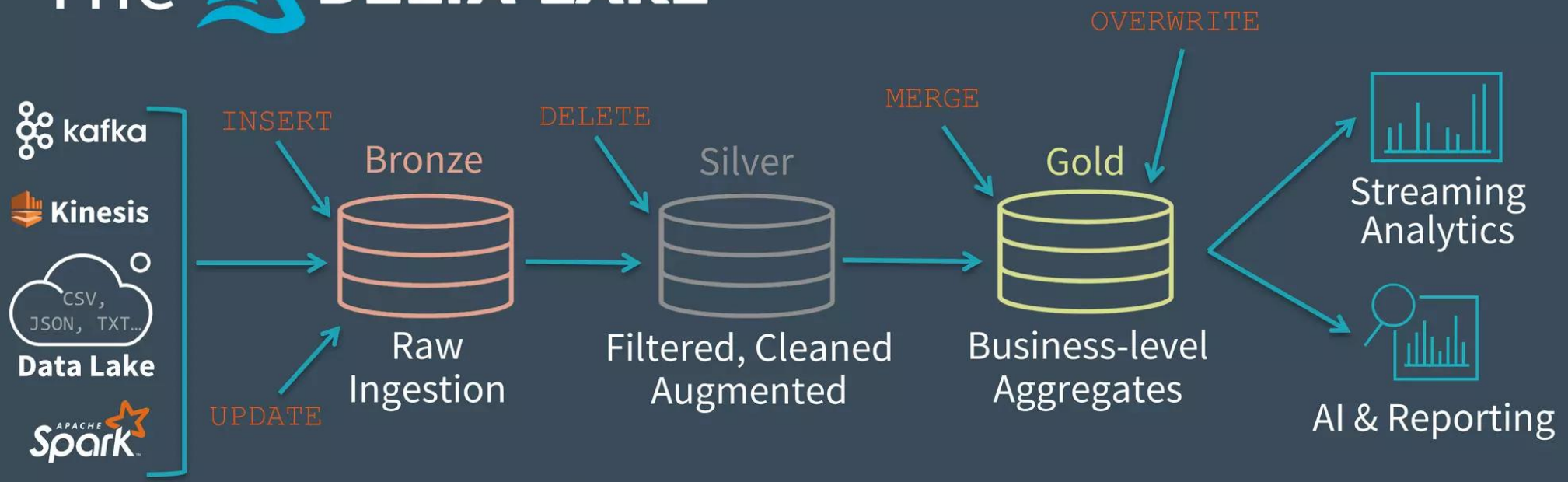
*Coming Soon

The DELTA LAKE

Streams move data through the Delta Lake

- Low-latency or manually triggered
- Eliminates management of schedules and jobs

The DELTA LAKE

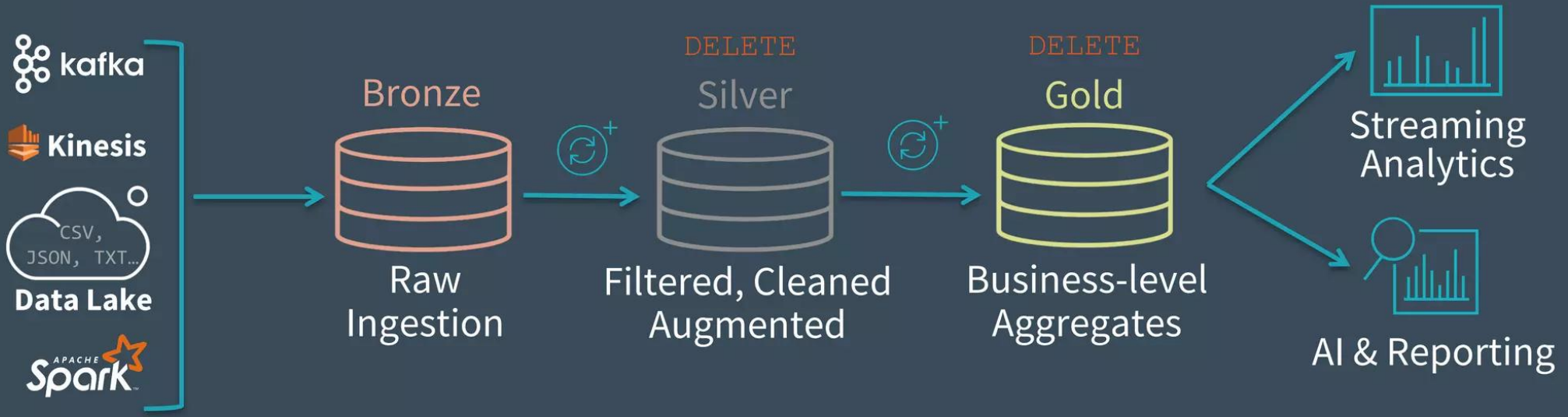


Delta Lake also supports batch jobs and standard DML

- Retention
- Corrections
- GDPR
- UPSERTS

*DML Coming in 0.3.0

The DELTA LAKE



Easy to recompute when business logic changes:

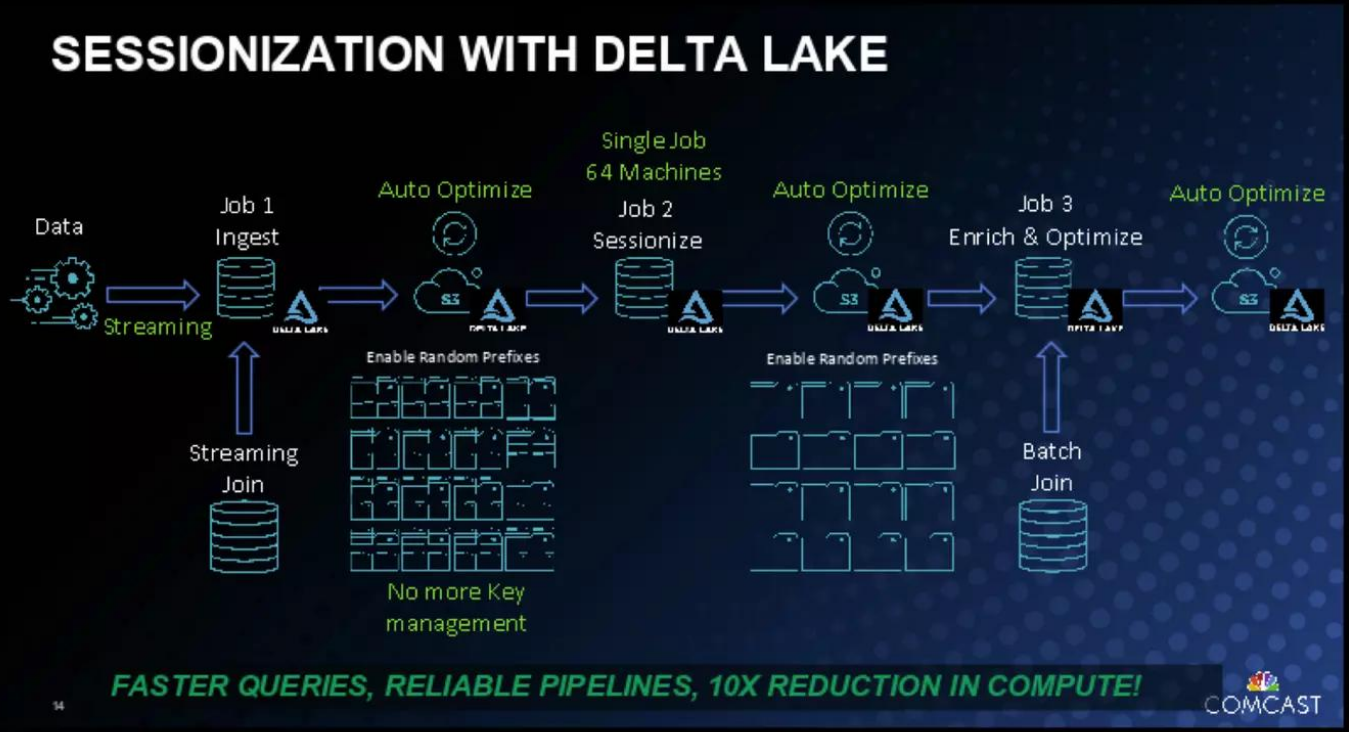
- Clear tables
- Restart streams

Who is using  DELTA LAKE?

Used by 1000s of organizations world wide

> 1 exabyte processed last month alone

COMCAST



Improved reliability:
Petabyte-scale jobs

10x lower compute:
640 instances to 64!

Simpler, faster ETL:
84 jobs → 3 jobs
halved data latency

How do I use  DELTA LAKE ?

Get Started with Delta using Spark APIs

Add Spark Package

```
pyspark --packages io.delta:delta-core_2.12:0.1.0

bin/spark-shell --packages io.delta:delta-core_2.12:0.1.0
```

Maven

```
<dependency>
  <groupId>io.delta</groupId>
  <artifactId>delta-core_2.12</artifactId>
  <version>0.1.0</version>
</dependency>
```

Instead of `parquet`...

```
dataframe
    .write
    .format("parquet")
    .save("/data")
```

... simply say `delta`

```
dataframe
    .write
    .format("delta")
    .save("/data")
```

Data Quality

Enforce metadata, storage, and quality declaratively.

```
table("warehouse")
  .location(…)                          // Location on DBFS
  .schema(…)                            // Optional strict schema checking
  .metastoreName(…)                     // Registration in Hive Metastore
  .description(…)                       // Human readable description for users
  .expect("validTimestamp",             // Expectations on data quality
    "timestamp > 2012-01-01 AND …",
    "fail / alert / quarantine")
```

*Coming Soon

Data Quality

Enforce metadata, storage, and quality declaratively.

```
table("warehouse")
  .location(…)                          // Location on DBFS
  .schema(…)                            // Optional strict schema checking
  .metastoreName(…)                     // Registration in Hive Metastore
  .description(…)                       // Human readable description for users
  .expect("validTimestamp",             // Expectations on data quality
    "timestamp > 2012-01-01 AND …",
    "fail / alert / quarantine")
```

*Coming Soon

How does  DELTA LAKE work?

Delta On Disk

Transaction Log

Table Versions

(Optional) Partition Directories

Data Files

Table = result of a set of actions

Change Metadata – name, schema, partitioning, etc

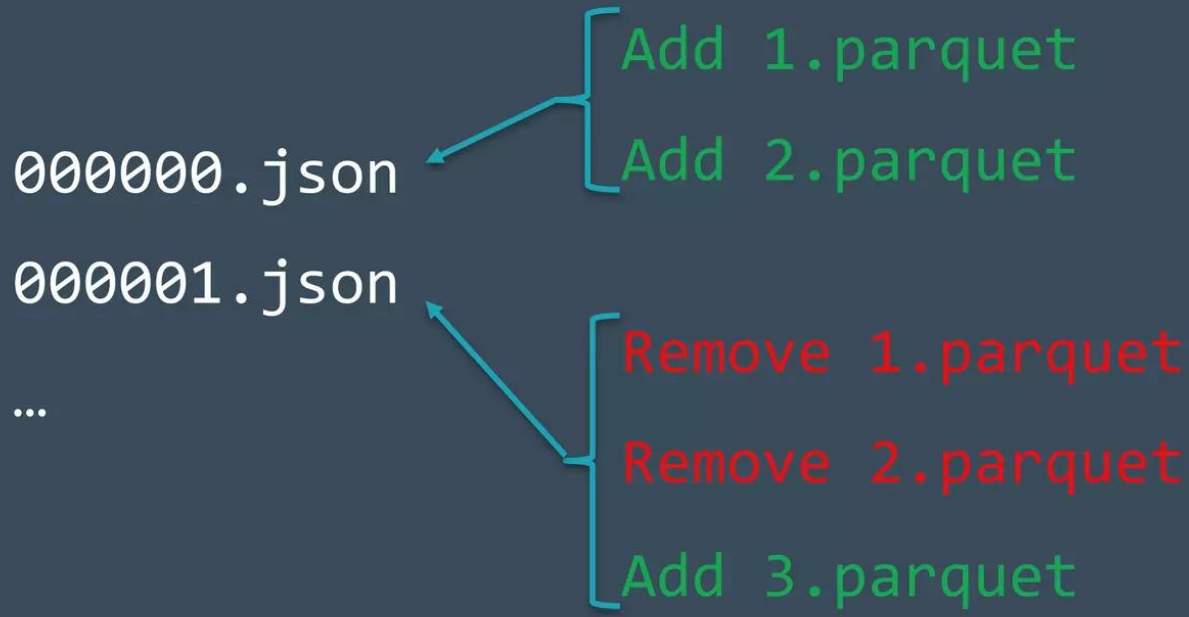
Add File – adds a file (with optional statistics)

Remove File – removes a file

Result: Current Metadata, List of Files, List of Txns, Version

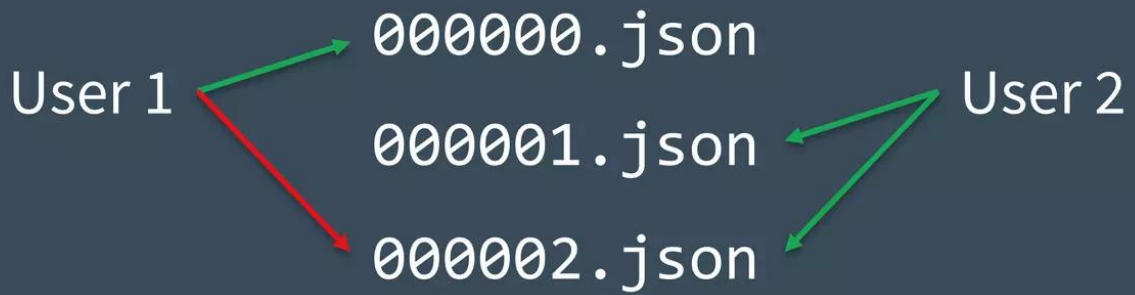
Implementing Atomicity

Changes to the table are stored as *ordered, atomic* units called commits



Ensuring Serializablity

Need to agree on the order of changes, even when there are multiple writers.



Solving Conflicts Optimistically

1. Record start version
2. Record reads/writes
3. Attempt commit
4. If someone else wins, check if anything you read has changed.
5. Try again.

Handling Massive Metadata

Large tables can have millions of files in them! How do we scale the metadata? Use Spark for scaling!

Road Map

- 0.2.0 – Released!
  - S3 Support
  - Azure Blob Store and ADLS Support
- 0.3.0 (~July)
  - UPDATE (Scala)
  - DELETE (Scala)
  - MERGE (Scala)
  - VACUUM (Scala)
- Rest of Q3
  - DDL Support / Hive Metastore
  - SQL DML Support

Build your own Delta Lake at **https://delta.io**