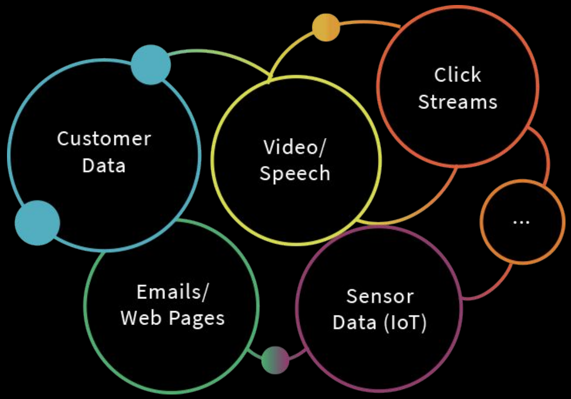
Making Apache Spark™ Better with Delta Lake

Steven Yu, Senior Solutions Architect

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

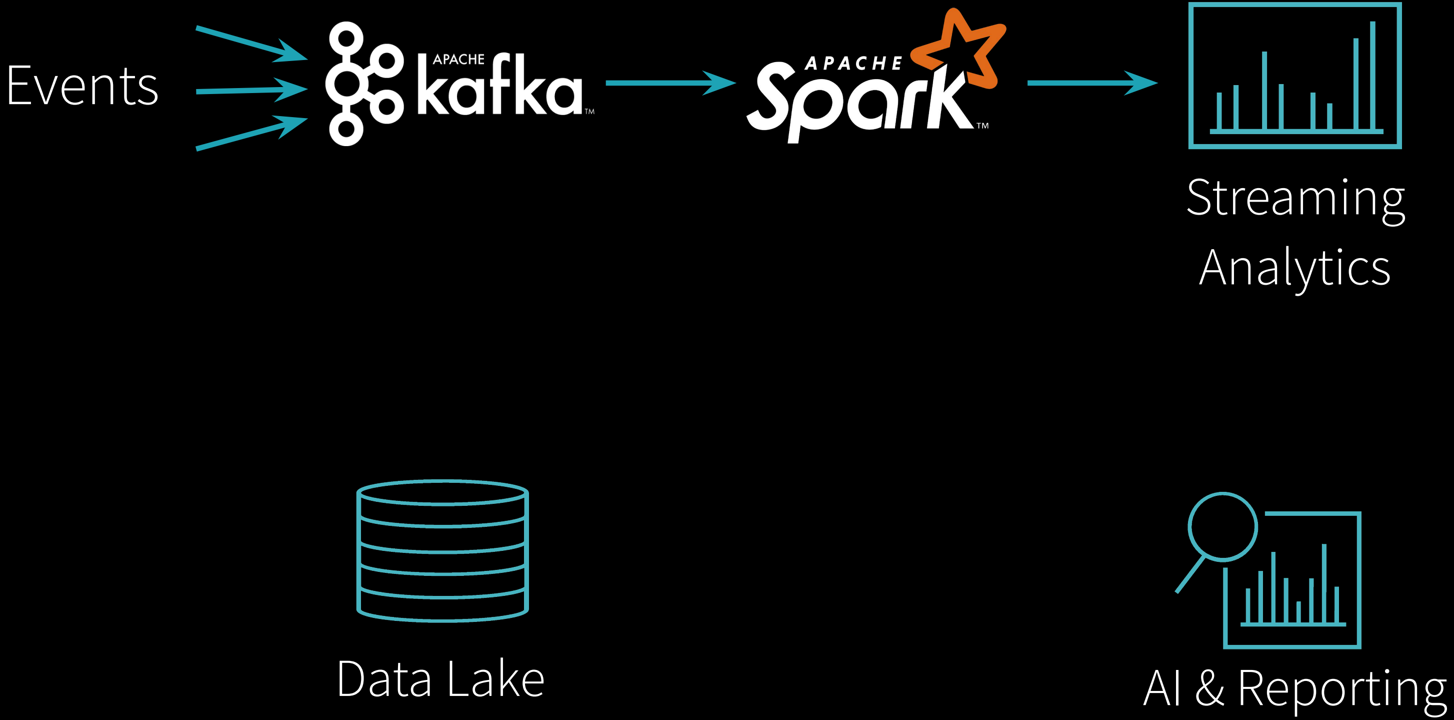
Garbage Out

What does a typical

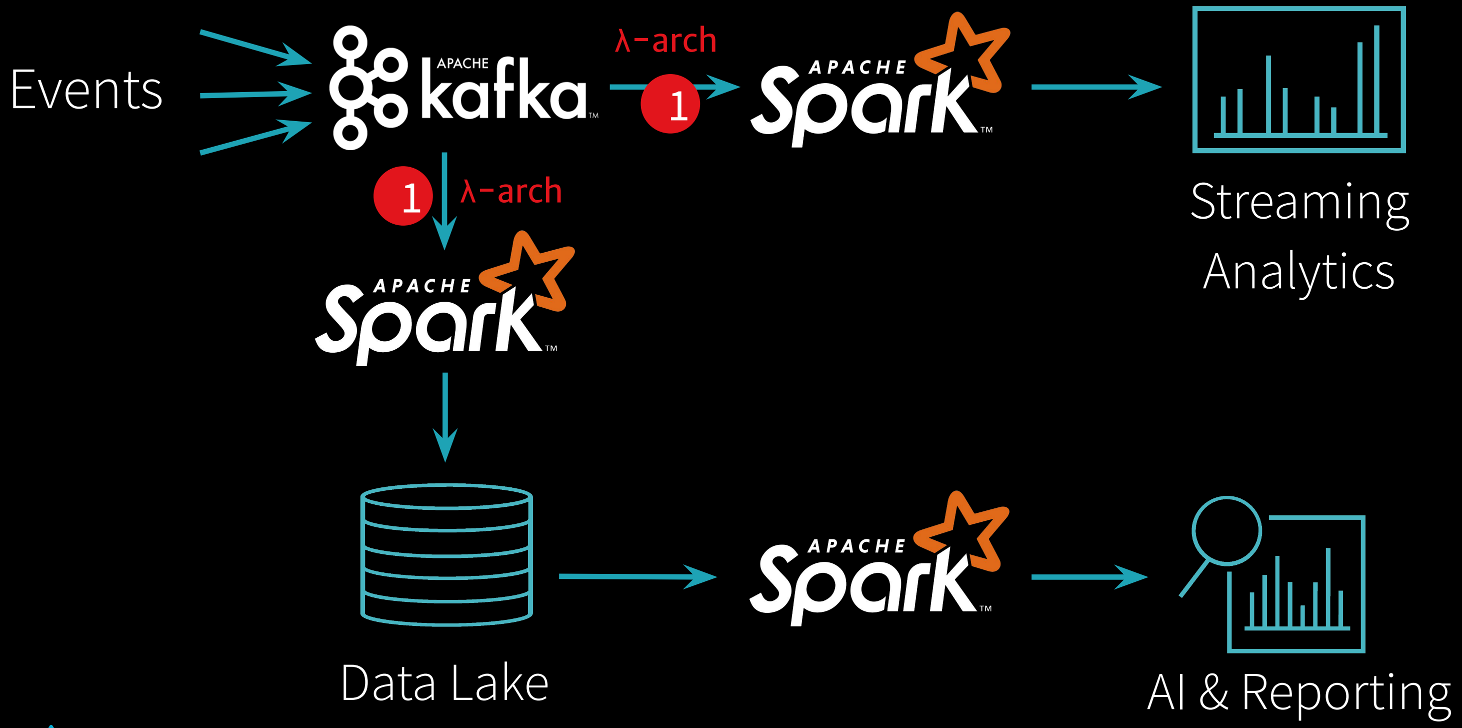
data lake project look like?

Evolution of a Cutting-Edge Data Lake

Evolution of a Cutting-Edge Data Lake

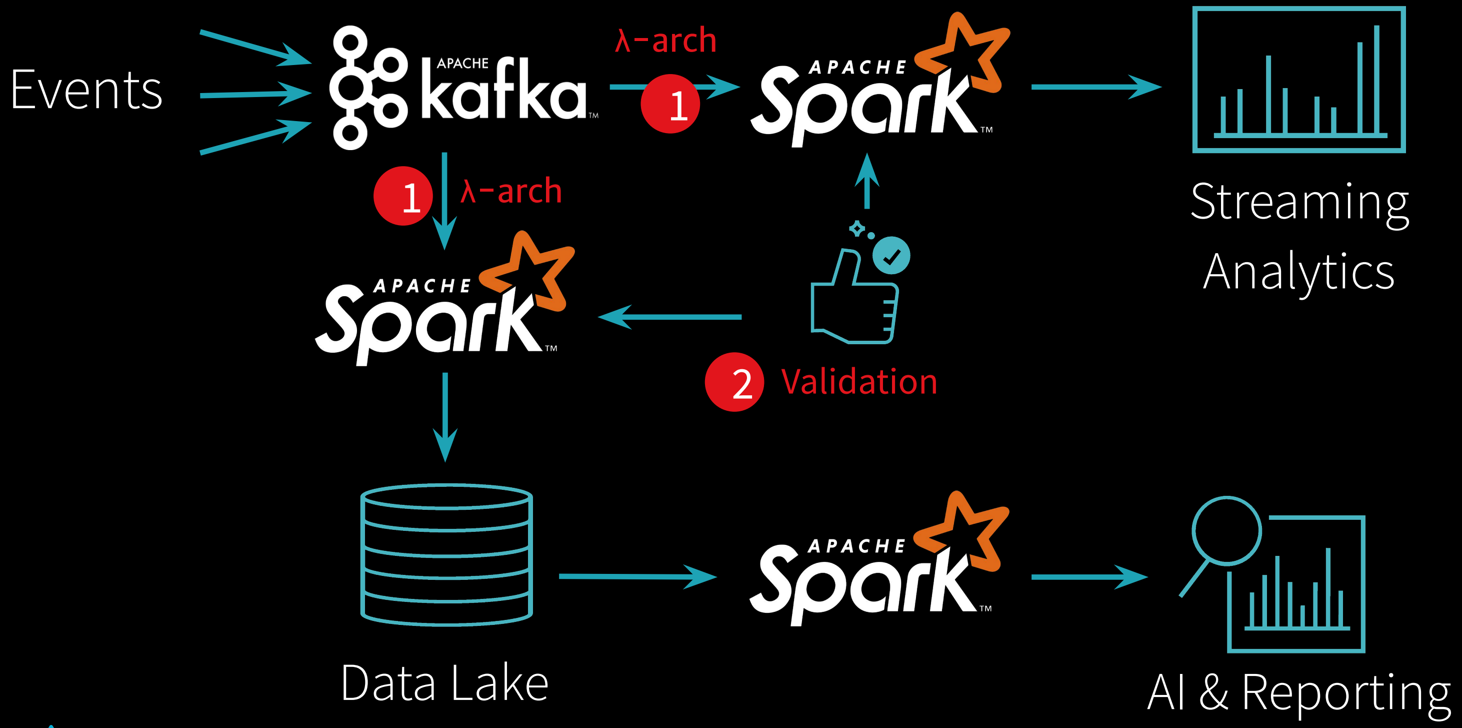


Challenge #1: Historical Queries?



1 λ-arch

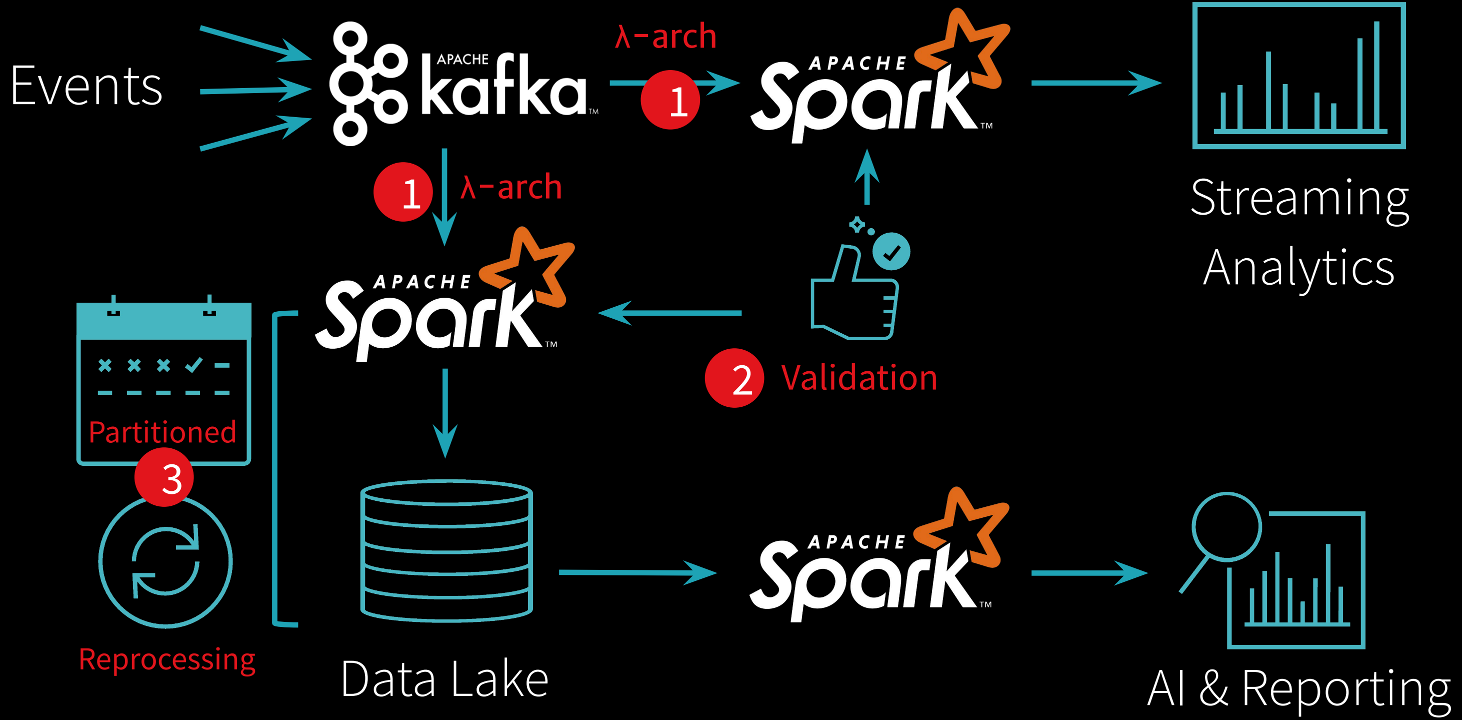
Challenge #2: Messy Data?



1 λ-arch

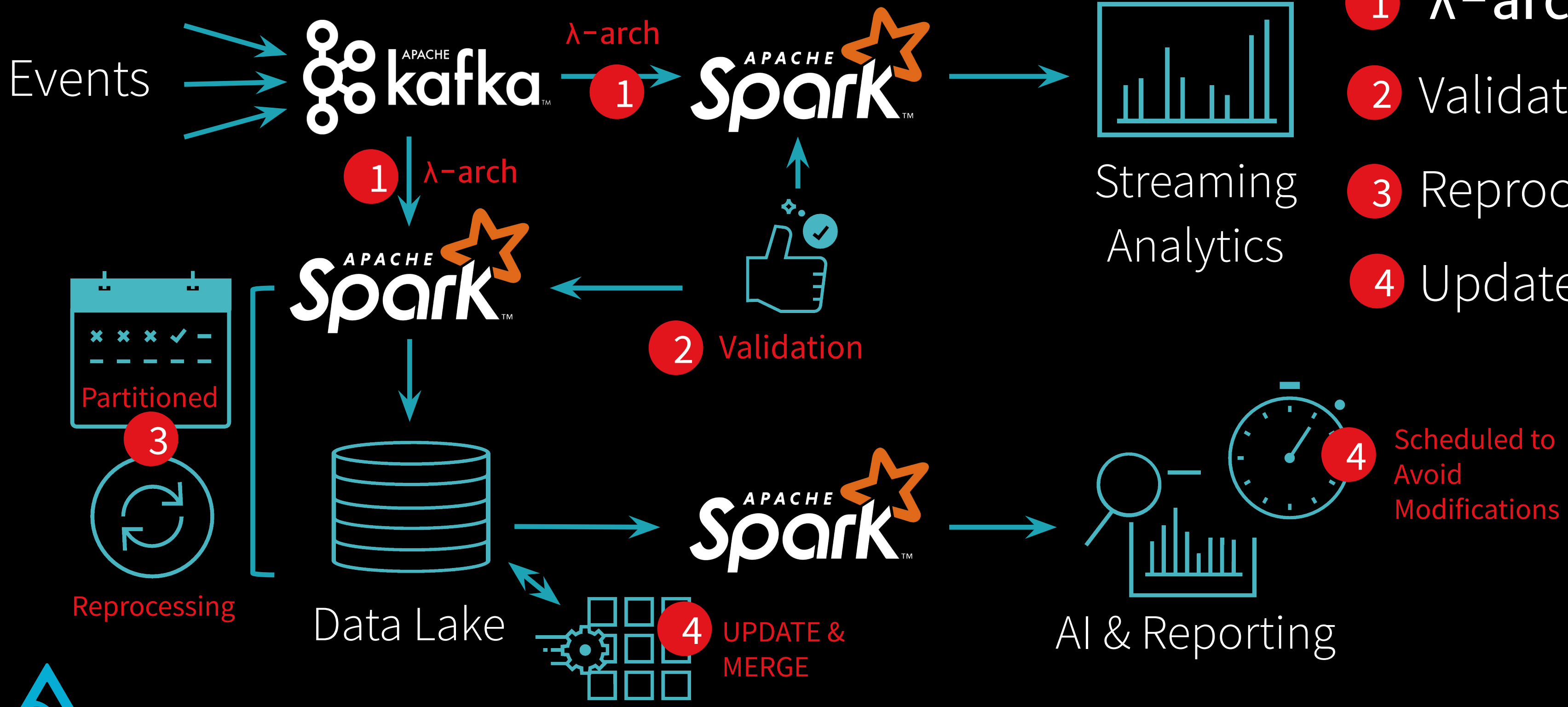
2 Validation

Challenge #3: Mistakes and Failures?



1 λ-arch
2 Validation
3 Reprocessing

Challenge #4: Updates?



1. λ-arch
2. Validation
3. Reprocessing
4. Updates

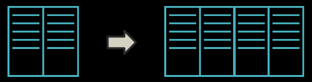
Wasting Time & Money

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



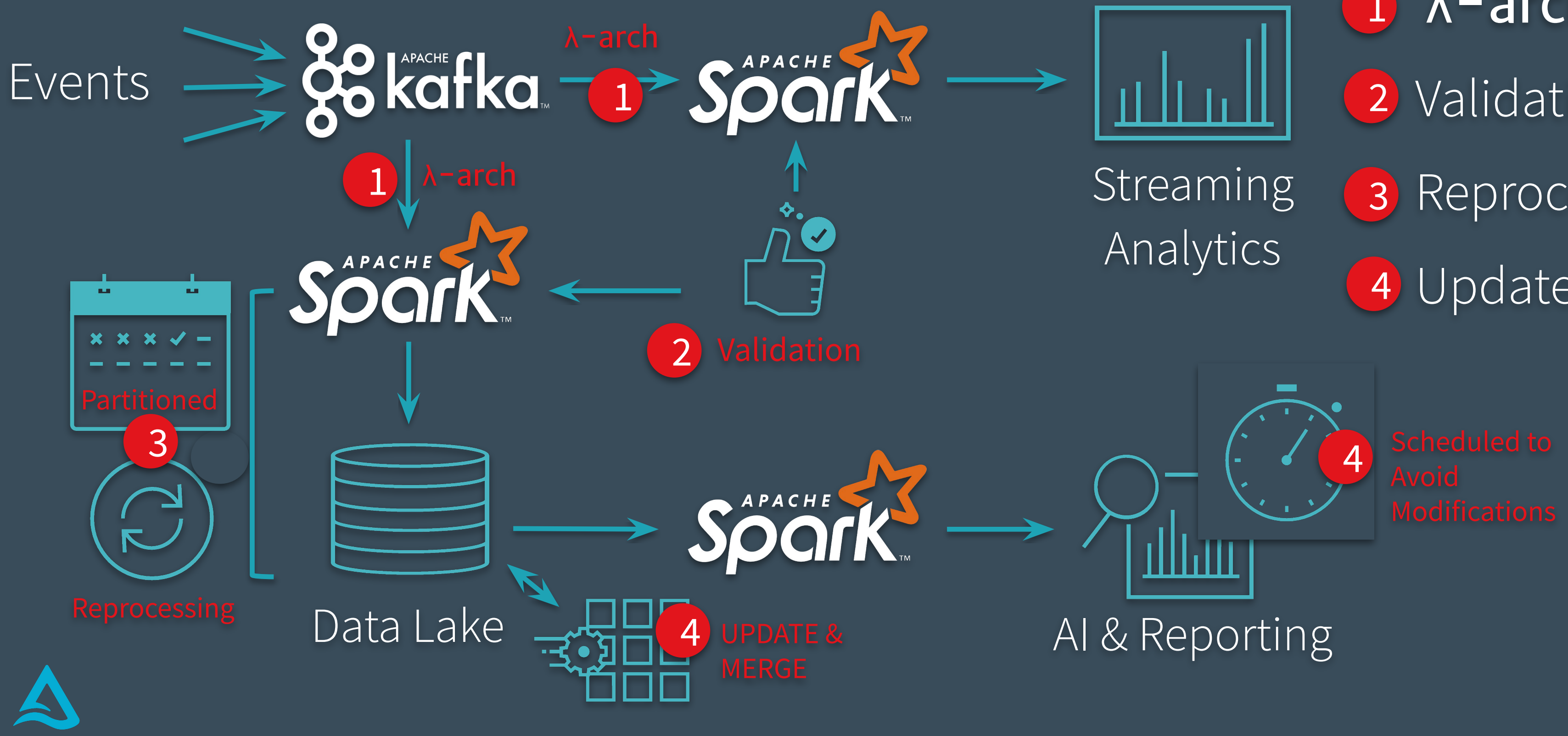
No quality enforcement creates inconsistent and unusable data

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

Let’s try it instead with

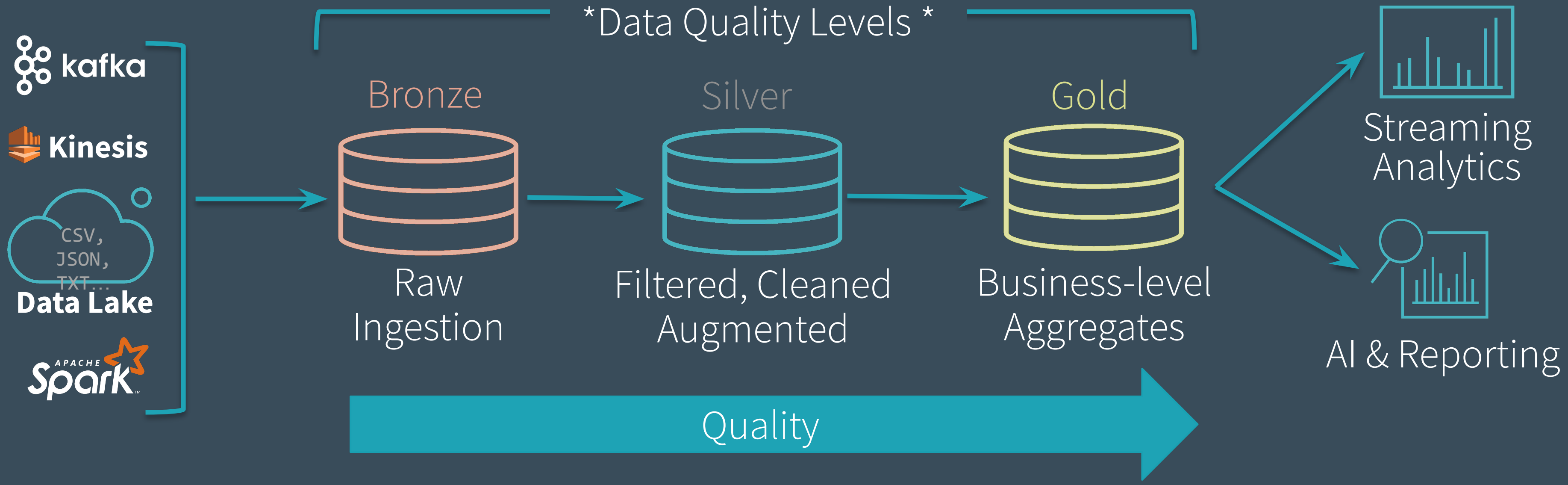
DELTA LAKE

Challenges of the Data Lake



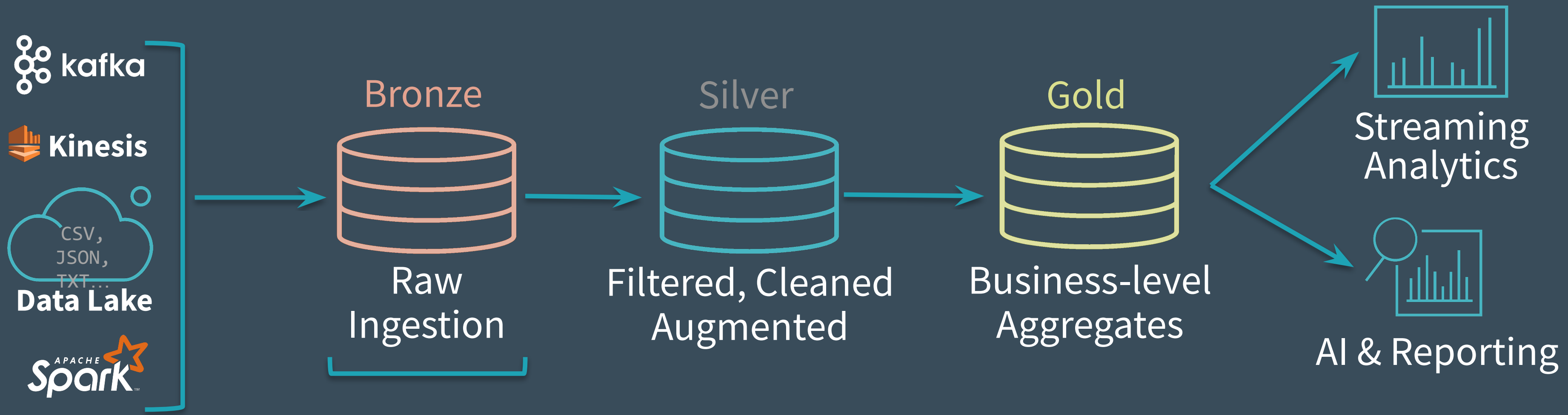
1 λ-arch
2 Validation
3 Reprocessing
4 Updates

The DELTA LAKE



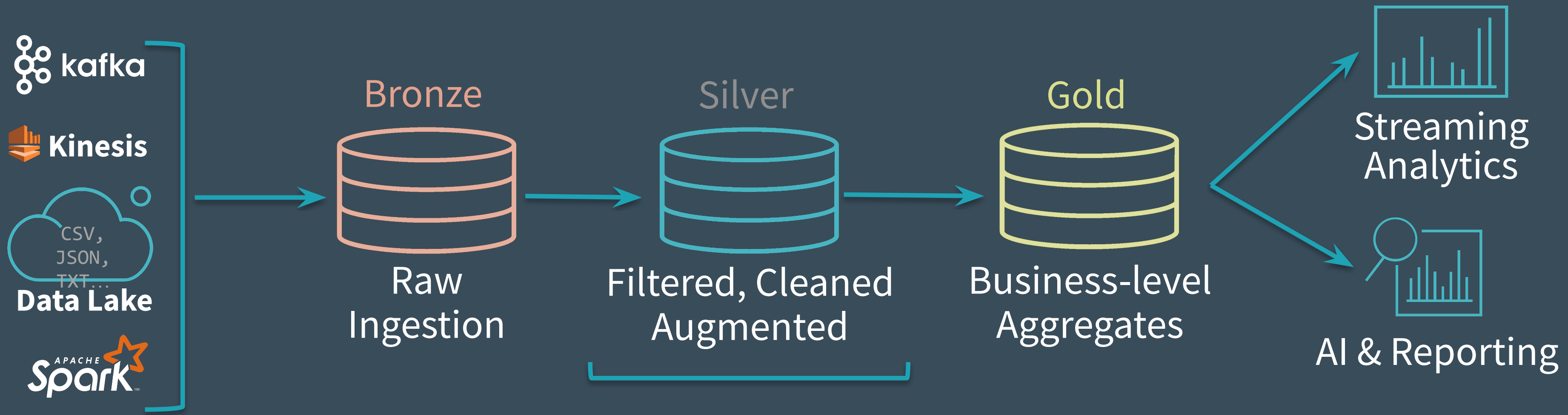
Delta Lake allows you to *incrementally* improve the quality of your data until it is ready for consumption.

The DELTA LAKE



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

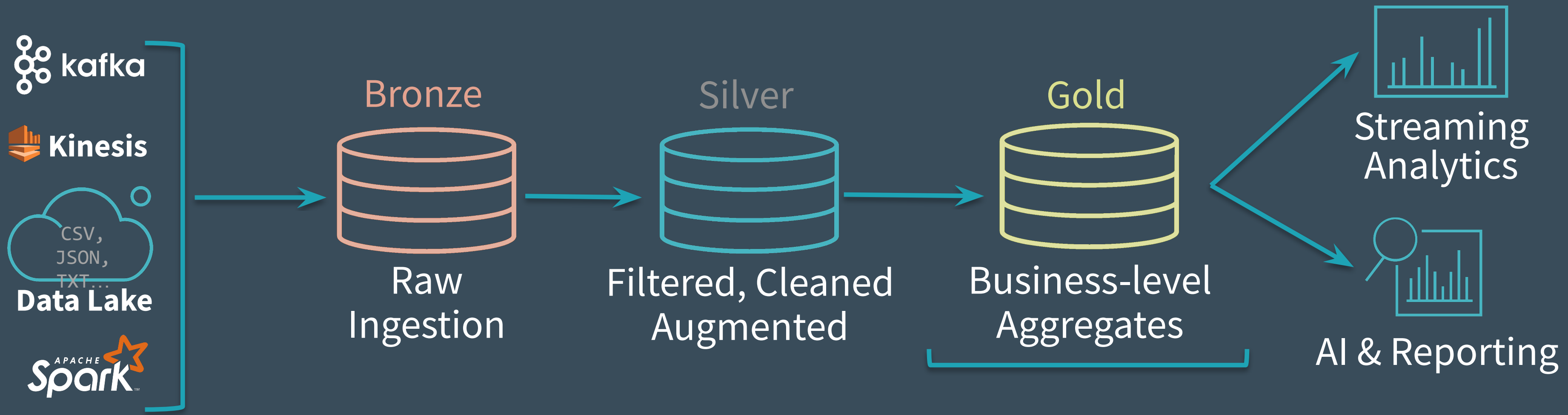
The DELTA LAKE



Intermediate data with some cleanup applied.

Queryable for easy debugging!

The DELTA LAKE

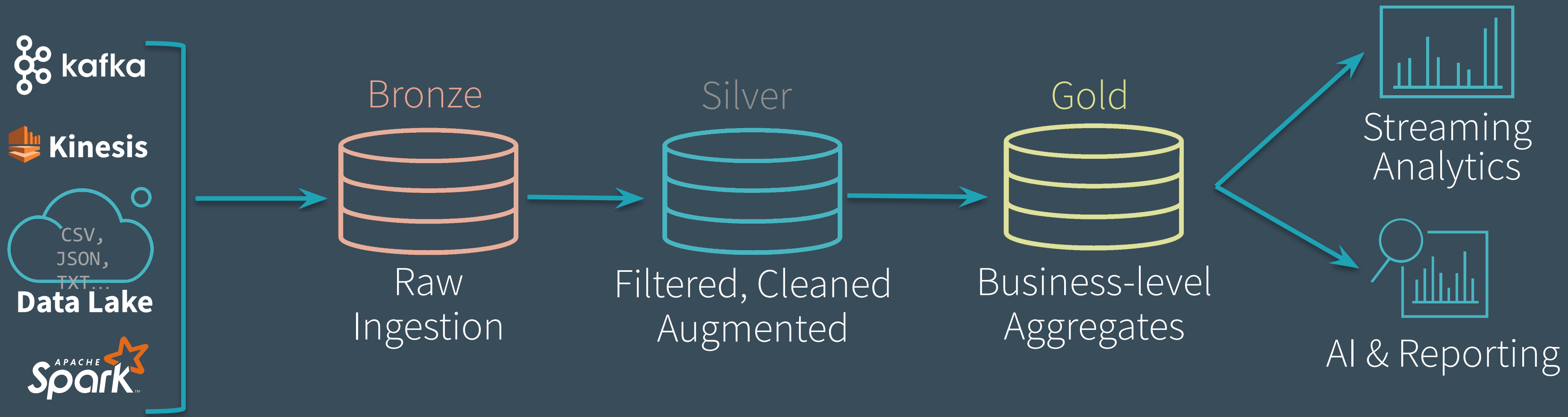


Clean data, ready for consumption.

Read with Spark or Presto*

*Coming Soon

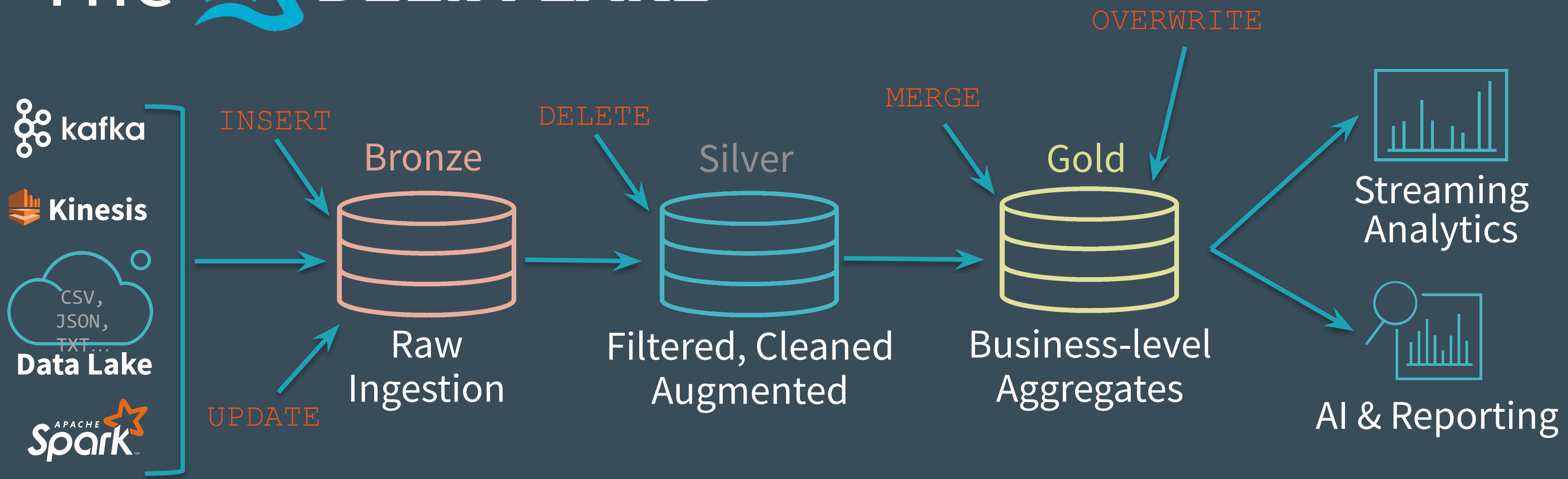
The DELTA LAKE



Streams move data through the Delta Lake

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

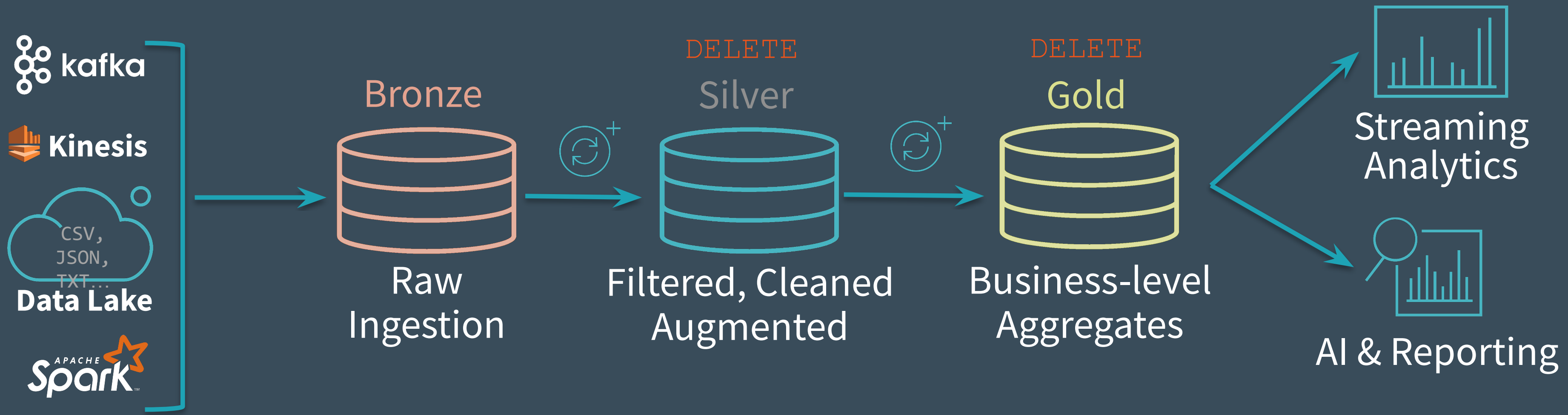
The DELTA LAKE



Delta Lake also supports batch jobs and standard DML

- Retention
- Corrections
- GDPR

The DELTA LAKE



Easy to recompute when business logic changes:

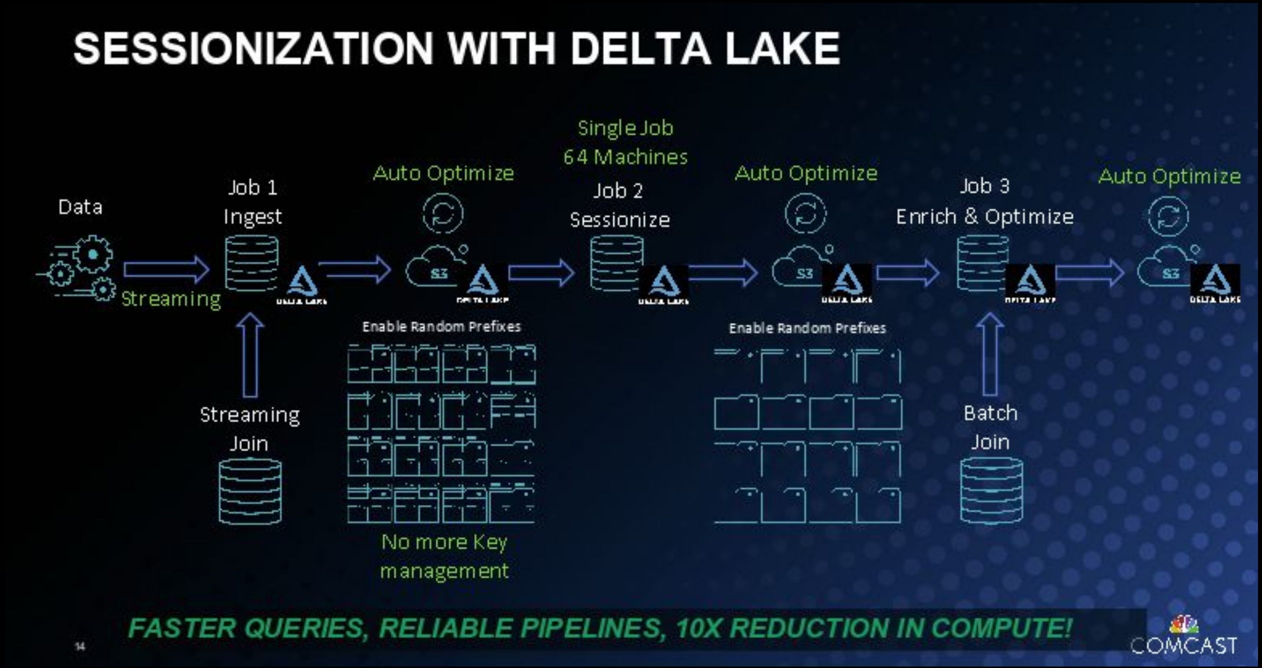
- Clear tables
- Restart streams

Who is using DELTA LAKE?

Used by 1000s of organizations world wide

> 1 exabyte processed last month alone

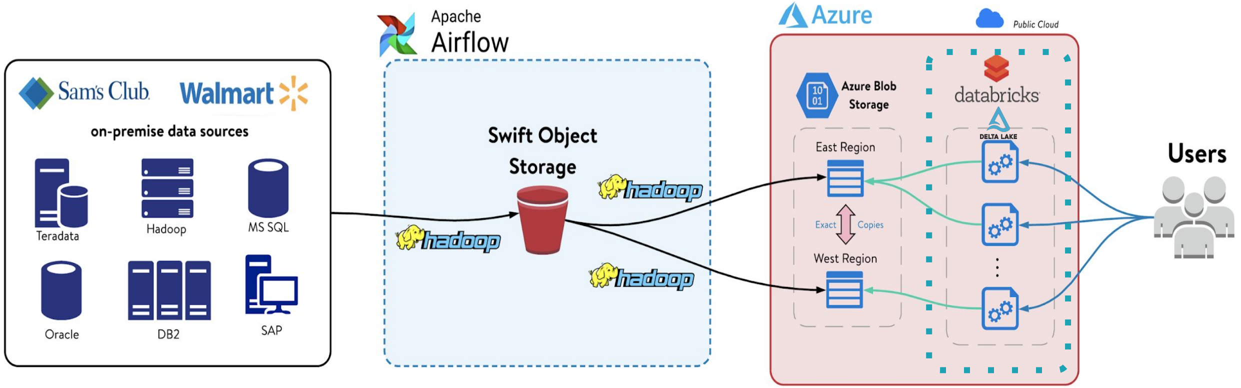
COMCAST



Improved reliability:
Petabyte-scale jobs

10x lower compute:
640 instances to 64!

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



Easier transactional updates:
No downtime or consistency issues!

Simple CDC:
Easy with MERGE

Improved performance:
Queries run faster
>1 hr → < 6 sec

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")
```

How does DELTA LAKE work?

Delta On Disk

	my_table/
Transaction Log	_delta_log/
Table Versions	00000.json
	00001.json
(Optional) Partition Directories	date=2019-01-01/
Data Files	file-1.parquet

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

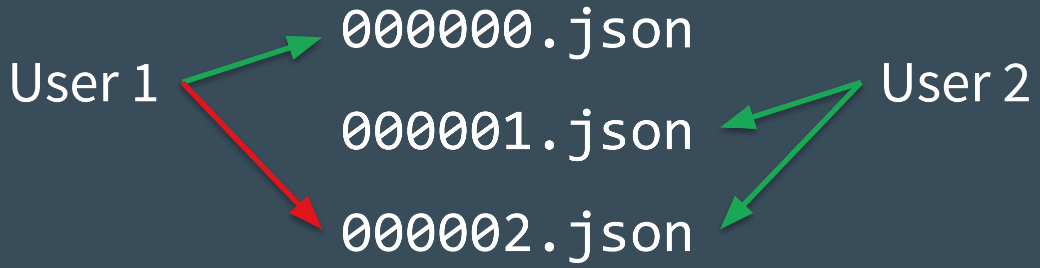
```
000000.json
000001.json
...
```

```
Add 1.parquet
Add 2.parquet
```

```
Remove 1.parquet
Remove 2.parquet
Add 3.parquet
```

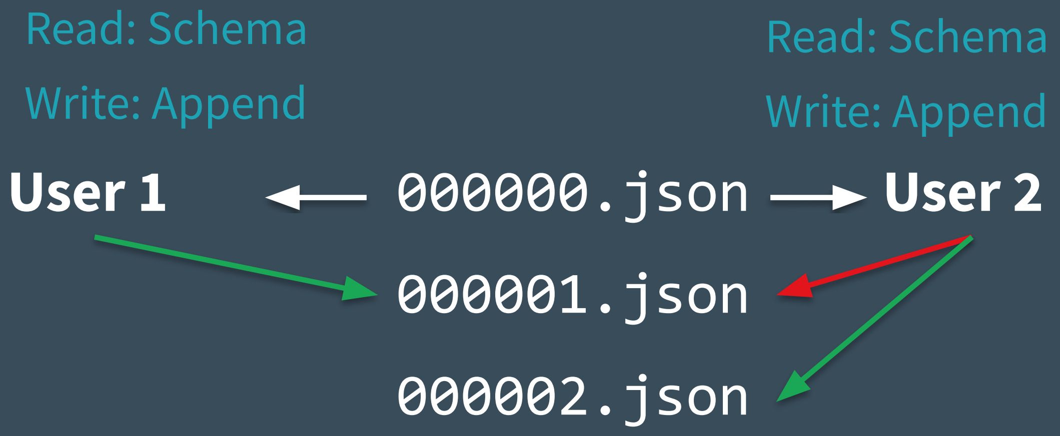
Ensuring Serializablity

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!

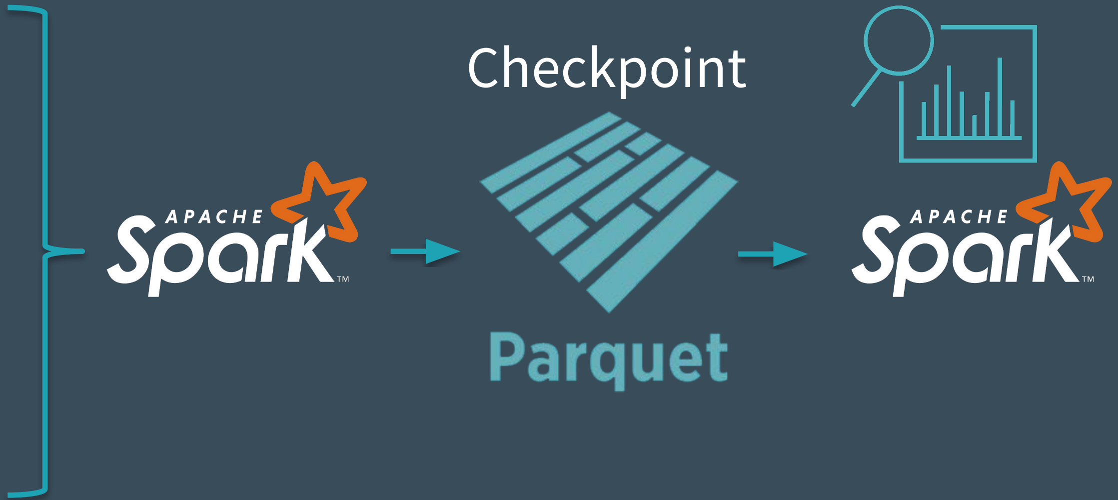
```
Add 1.parquet

Add 2.parquet

Remove 1.parquet

Remove 2.parquet

Add 3.parquet
```



Road Map

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

Build your own Delta Lake

at **https://delta.io**