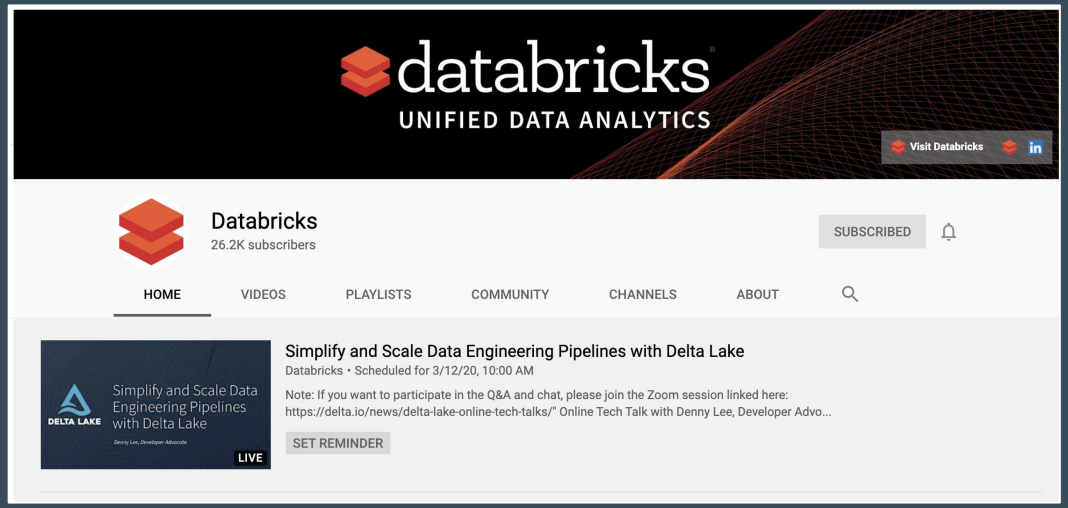

# Simplify and Scale Data Engineering Pipelines with Delta Lake

Denny Lee, Developer Advocate

# Subscribe Today!




https://dbricks.co/youtube

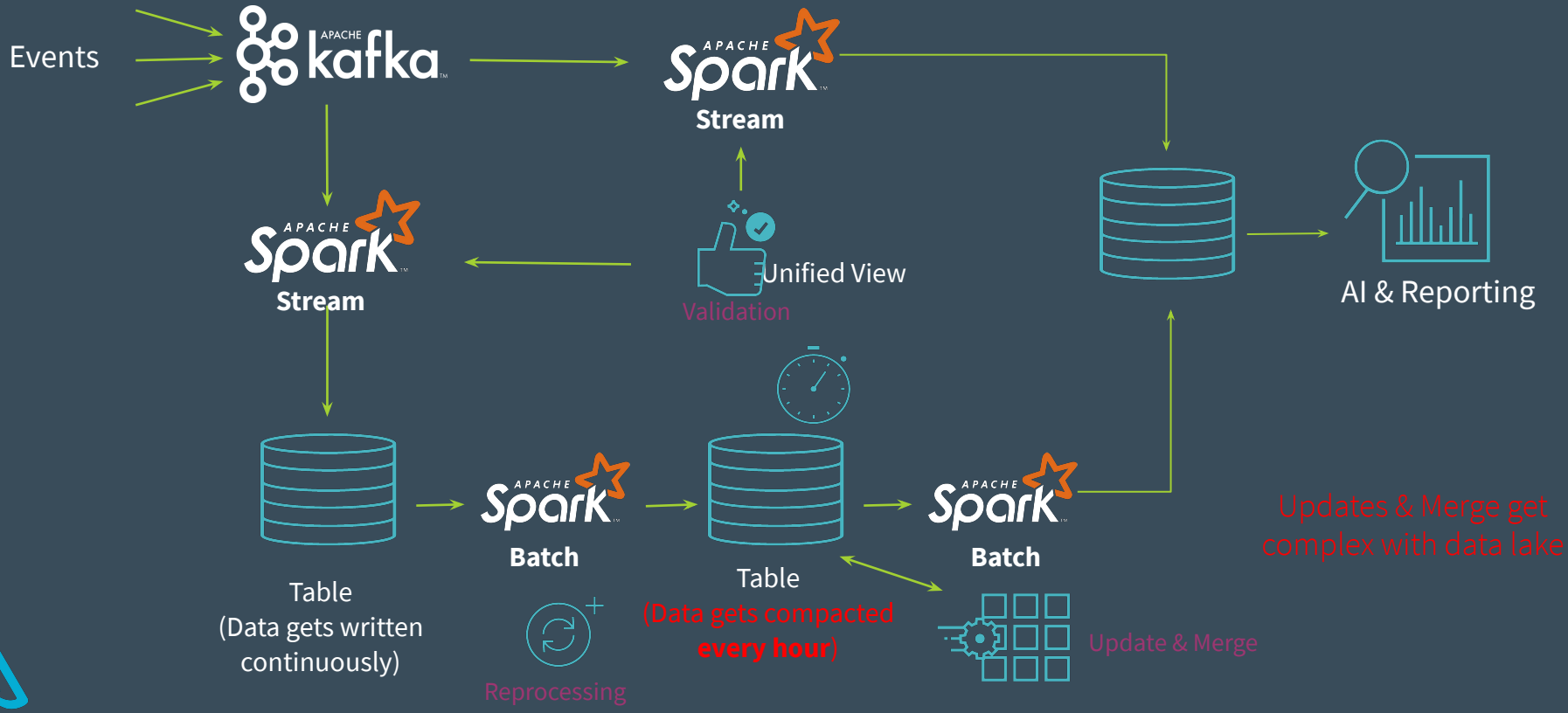
# Today's Speakers

Denny Lee is a Developer Advocate at Databricks. He is a hands-on distributed systems and data sciences engineer with extensive experience developing internet-scale infrastructure, data platforms, and predictive analytics systems for both on-premise and cloud environments.

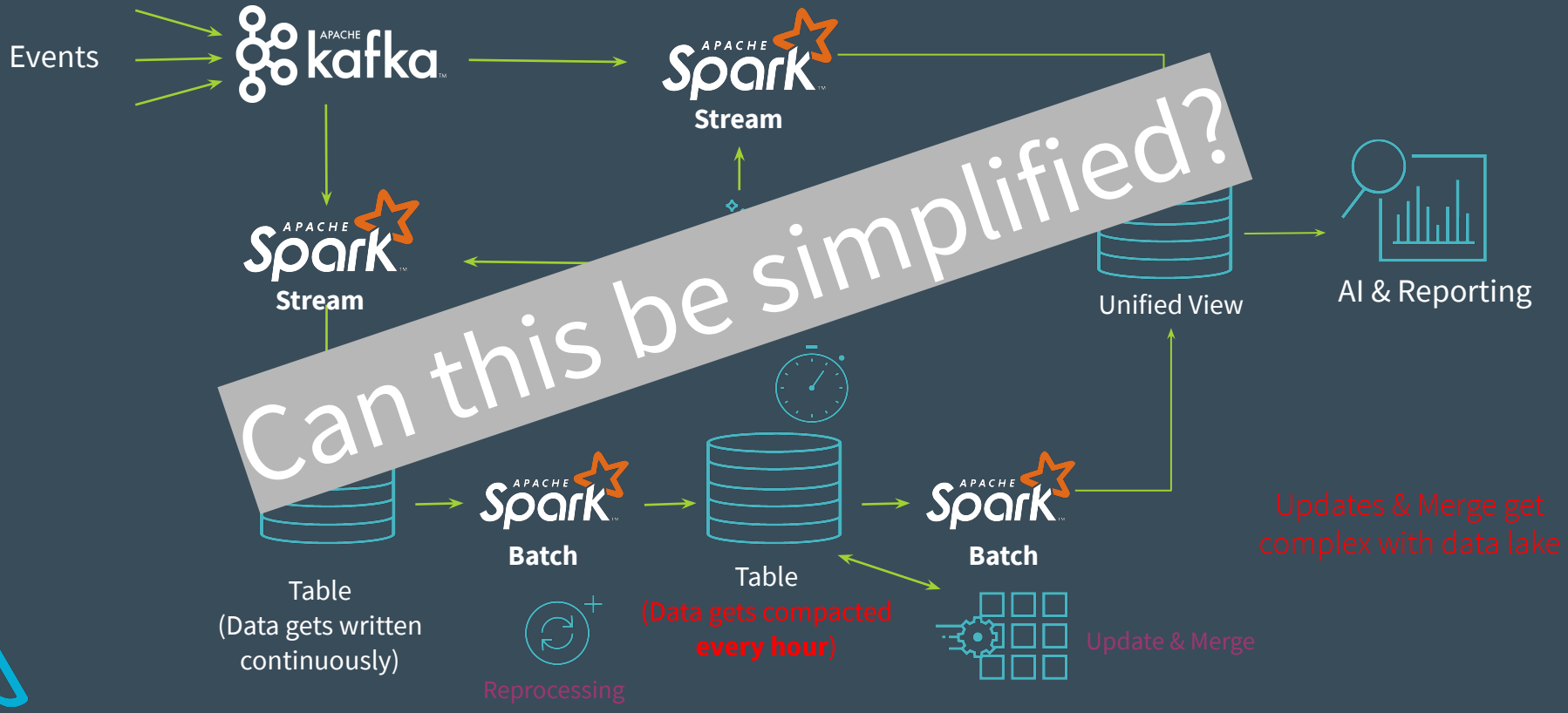
# Logistics

- Recording and slides will be available after this webinar
- Everyone is muted, put questions in the Q&A panel
- We will give you a link to follow up, for more information fill out the form

# The Data Engineer's Journey...



# The Data Engineer's Journey...



# A Data Engineer's Dream...

Process data continuously and incrementally as new data arrive in a cost efficient way without having to *choose* between batch or streaming


# What's missing?


1. Ability to read consistent data while data is being written
2. Ability to read incrementally from a large table with good throughput
3. Ability to rollback in case of bad writes
4. Ability to replay historical data along new data that arrived
5. Ability to handle late arriving data without having to delay downstream processing

# So… What is the answer?

**STRUCTURED STREAMING**

+

**DELTA LAKE**

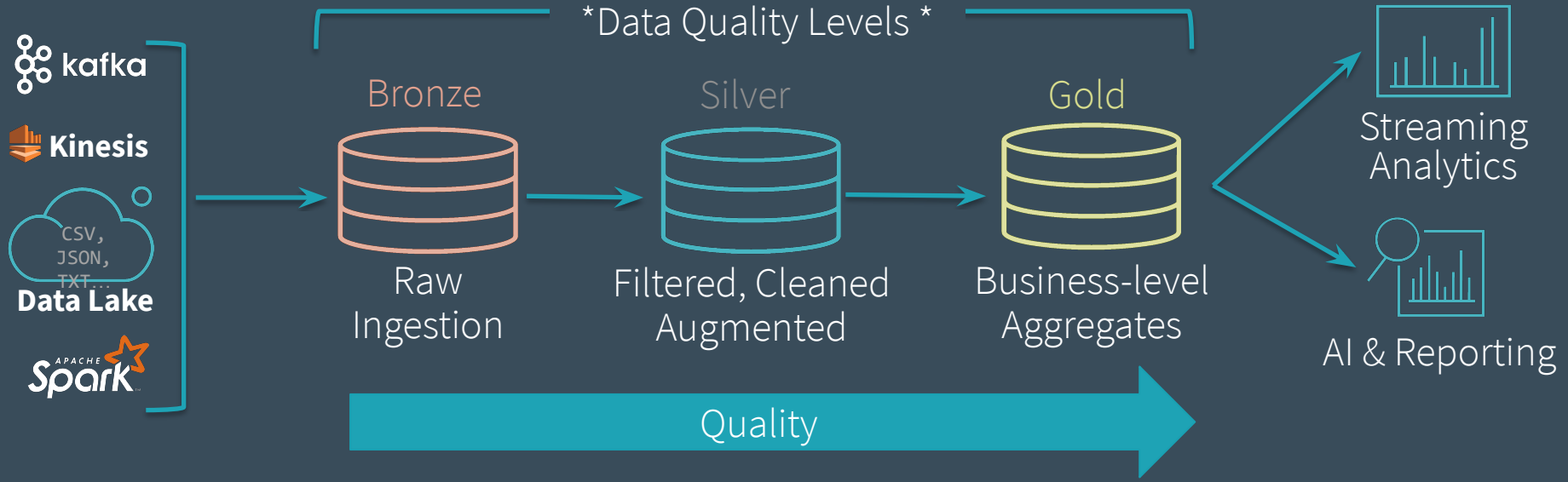
=

**The Delta Architecture**

1. Unify batch & streaming with a continuous data flow model
2. Infinite retention to replay/reprocess historical events as needed
3. Independent, elastic compute and storage to scale while balancing costs

# Let’s try it instead with

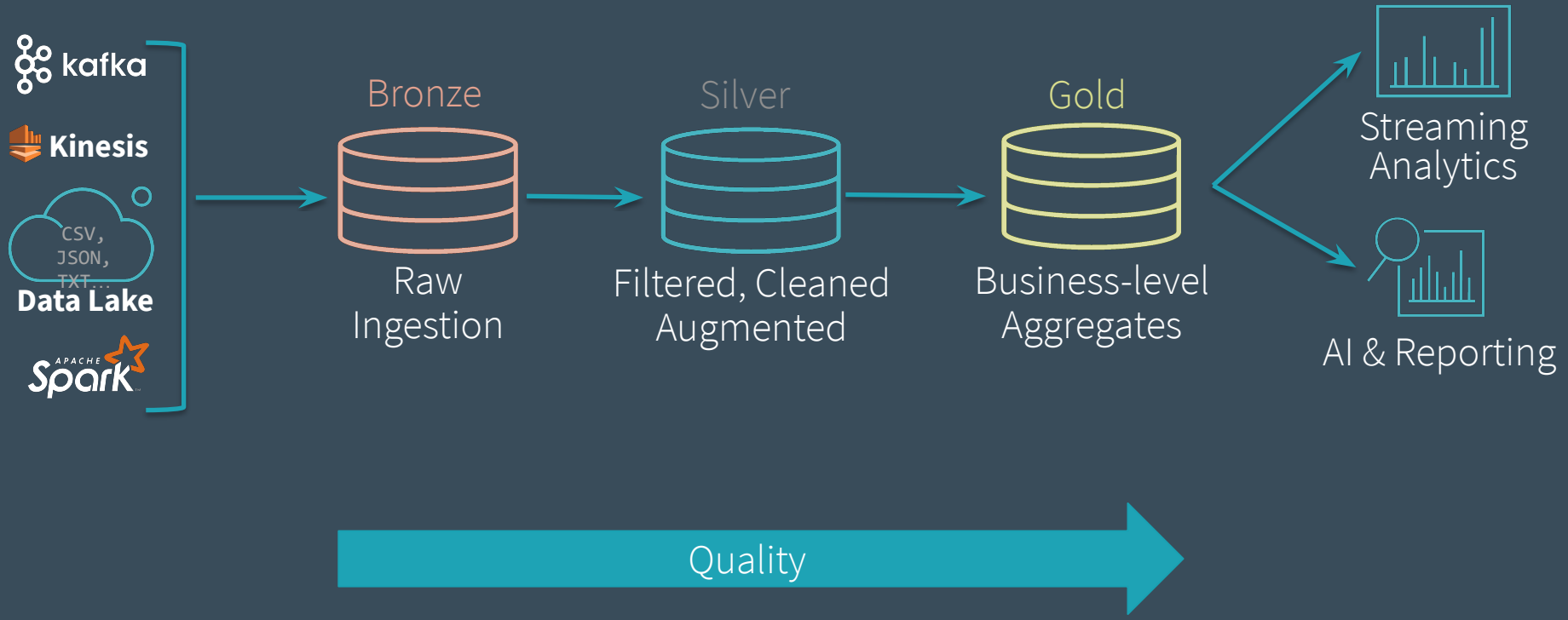

# The DELTA LAKE



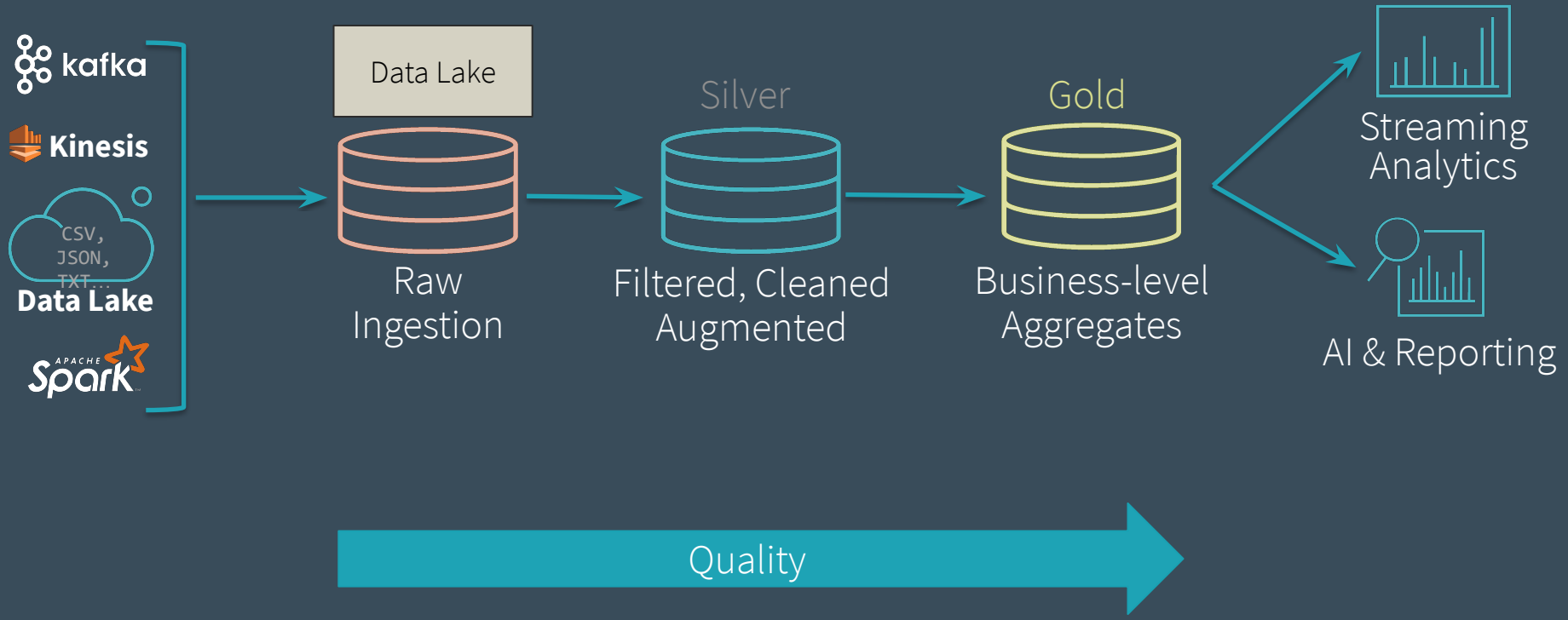

Delta Lake allows you to *incrementally* improve the quality of your data until it is ready for consumption.

# What does this remind you of?

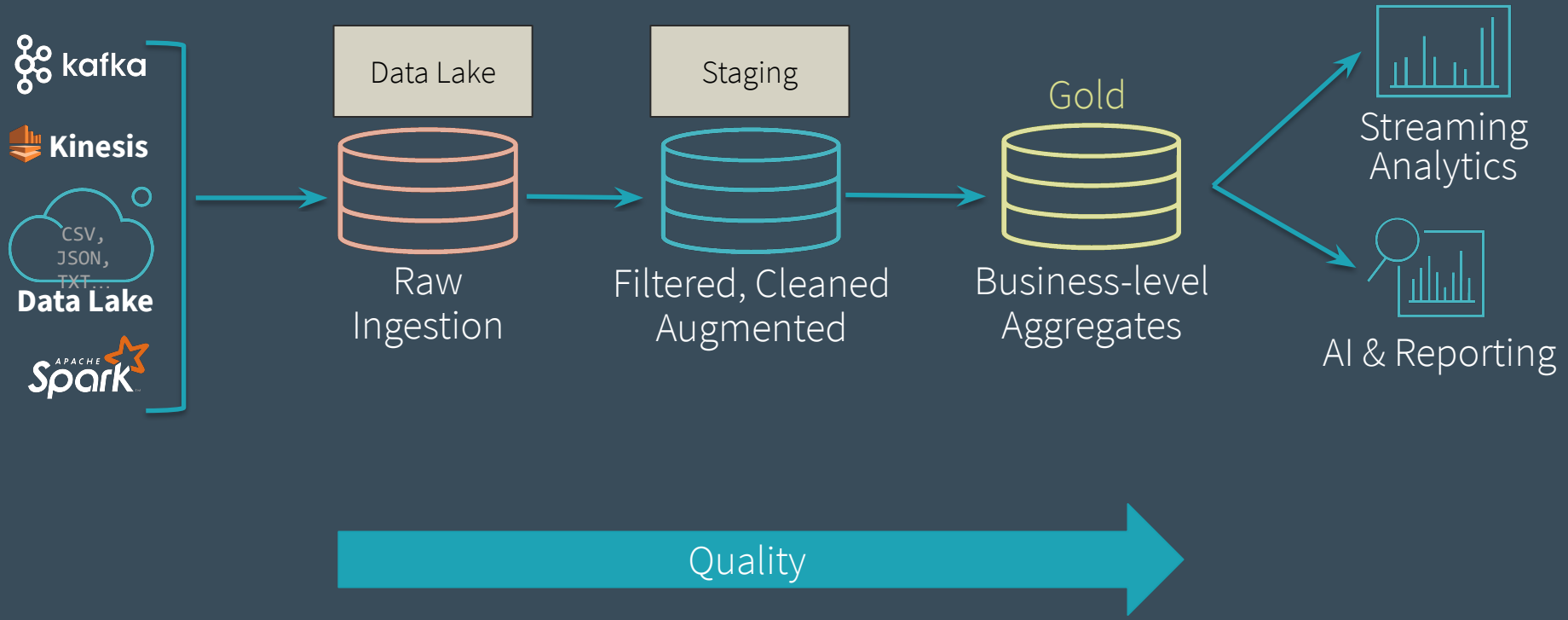
# The Data Lifecycle



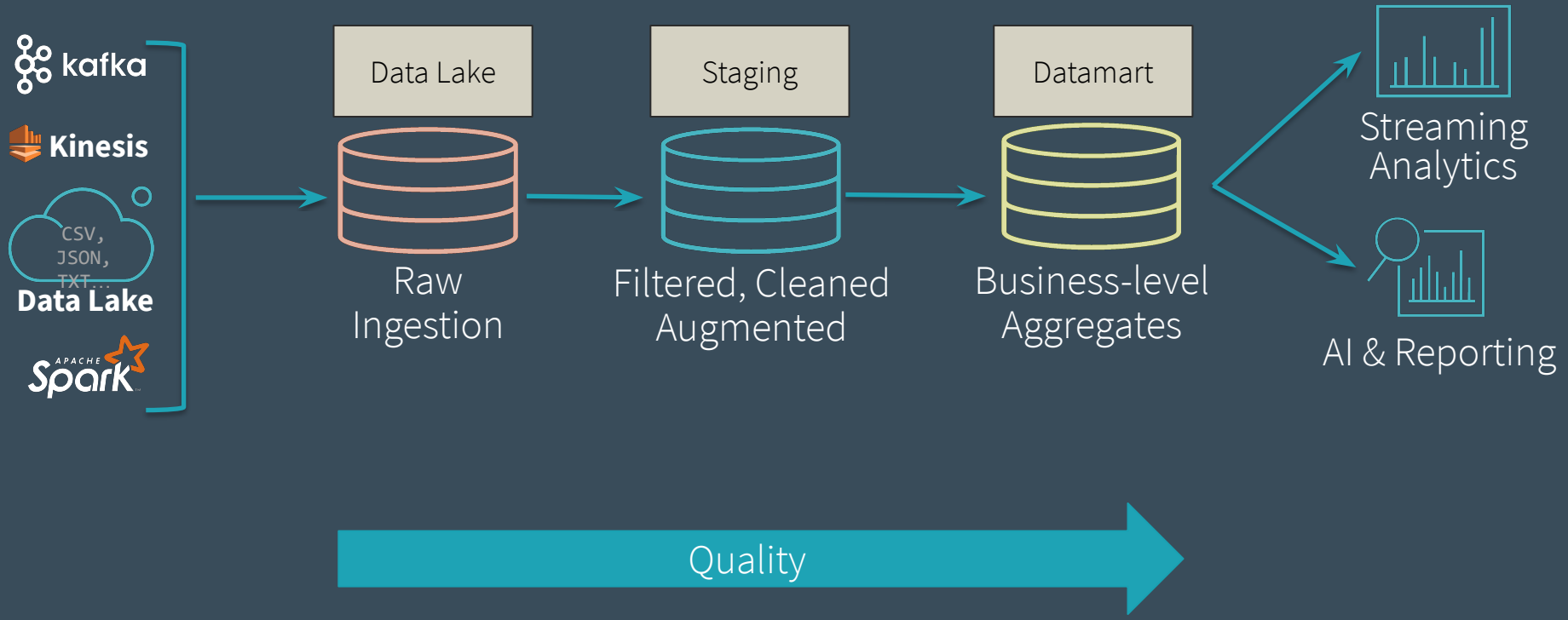
# The Data Lifecycle



# The Data Lifecycle

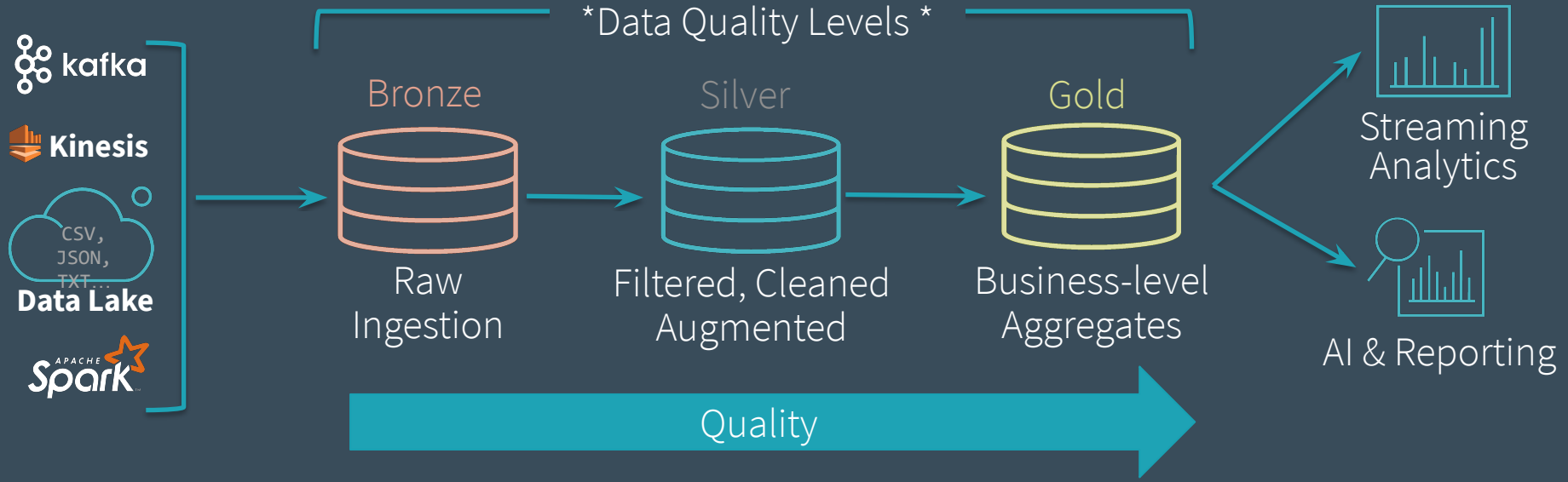


# The Data Lifecycle



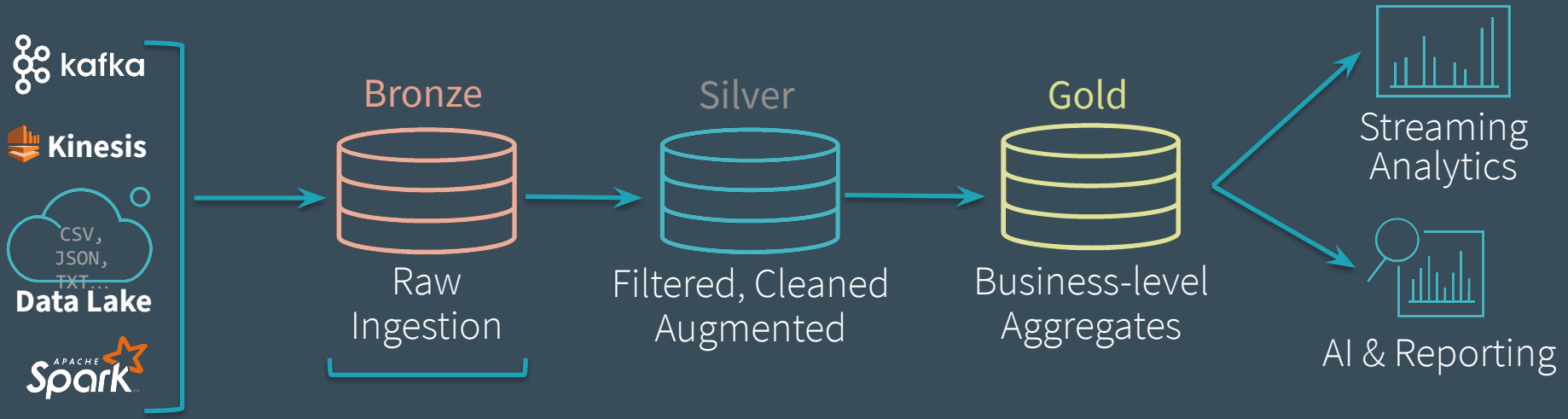
# Transitioning from the Data Lifecycle to the Delta Lake Lifecycle

# The DELTA LAKE



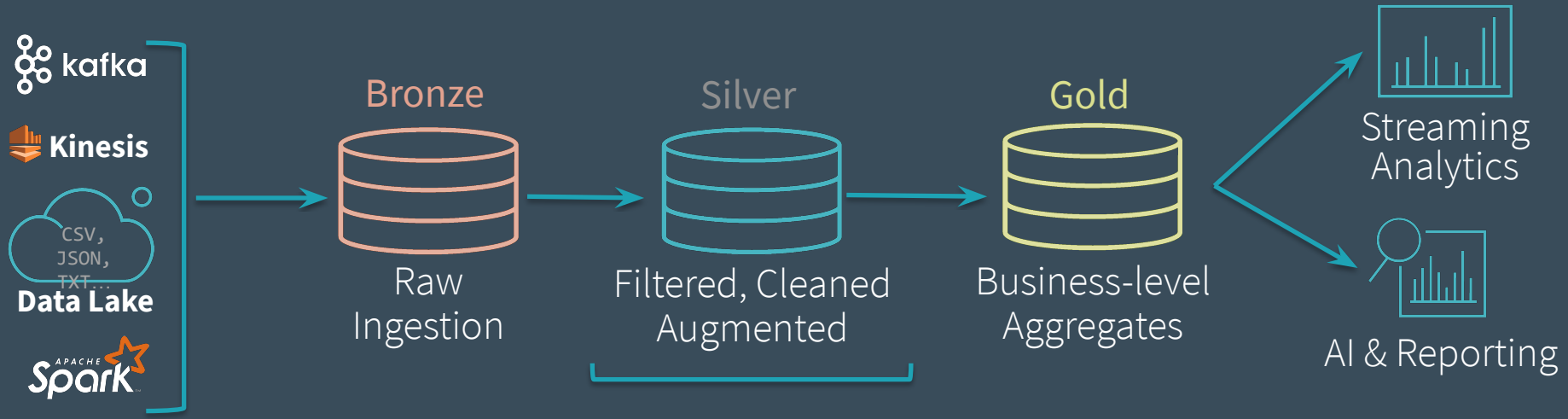

Delta Lake allows you to *incrementally* improve the quality of your data until it is ready for consumption.

# The DELTA LAKE




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

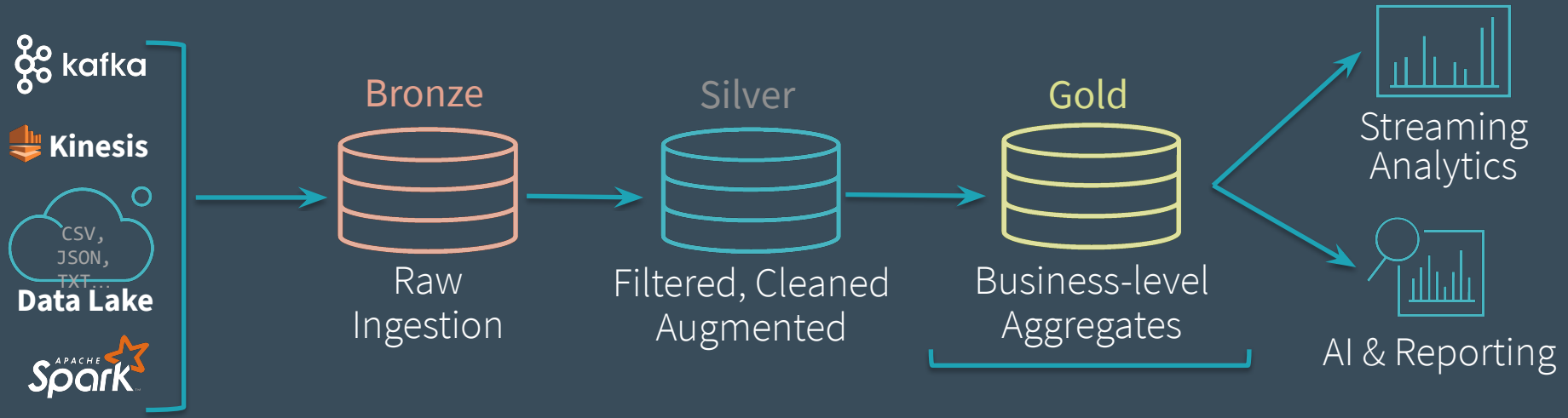
# The DELTA LAKE




Intermediate data with some cleanup applied.

Queryable for easy debugging!

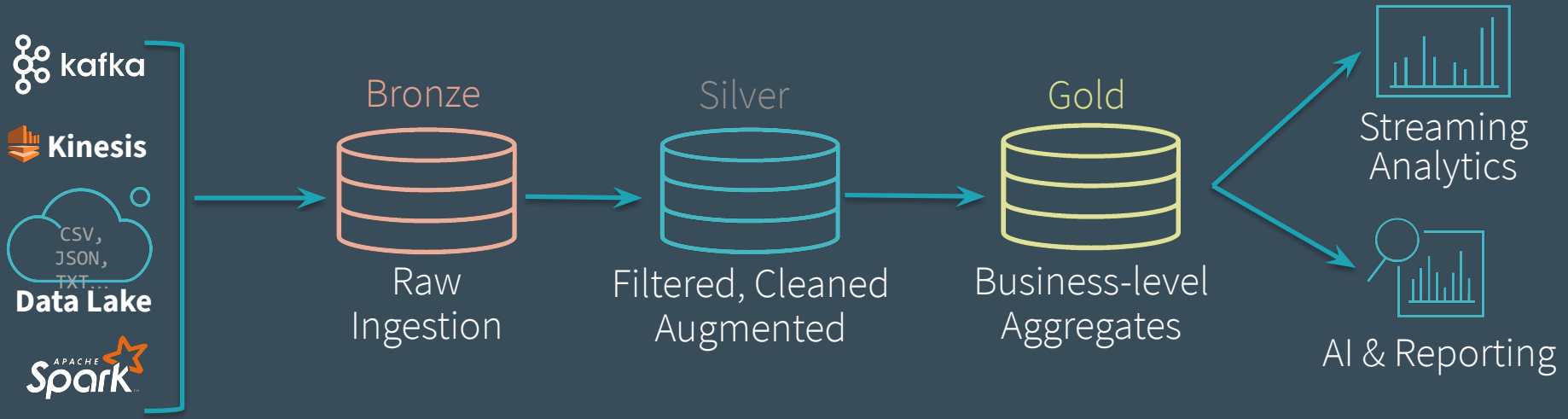
# The DELTA LAKE




Clean data, ready for consumption.

Read with Spark or Presto*

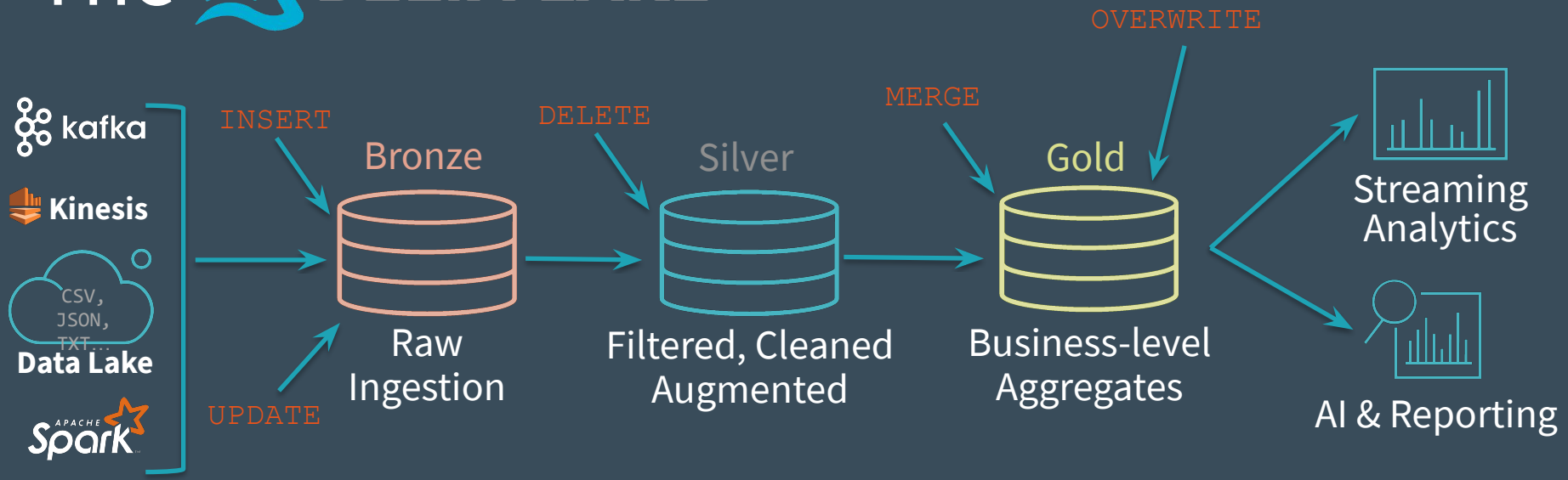
# The DELTA LAKE




Streams move data through the Delta Lake

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

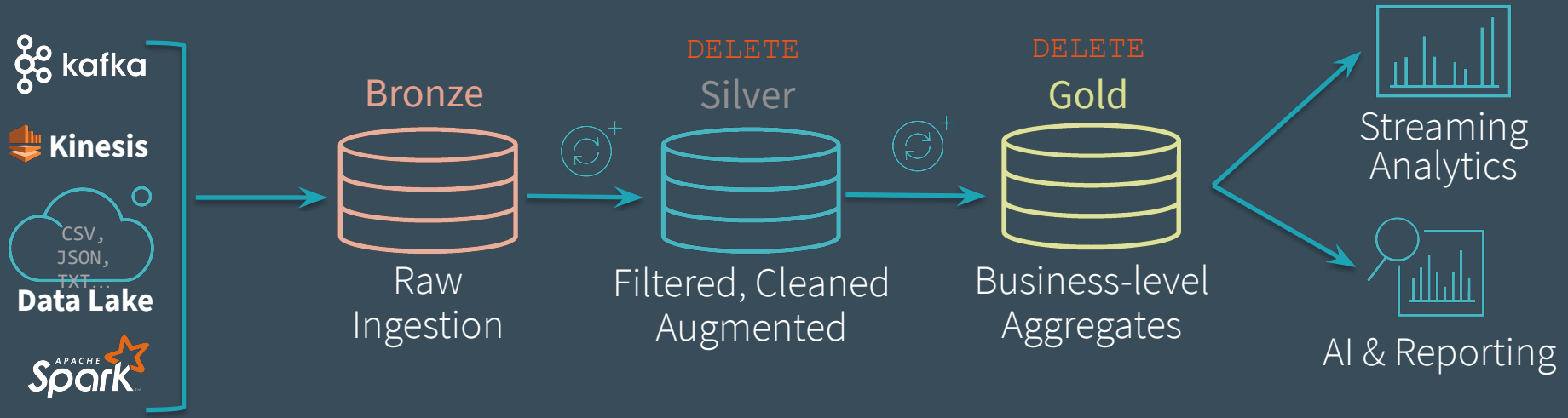
# The DELTA LAKE




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

Let's see  DELTA LAKE in action!

# The Delta Architecture

# Connecting the dots...


# Connecting the dots...


1. Ability to **read consistent data** while data is being written

Snapshot isolation between writers and readers

# Connecting the dots...


1. Ability to **read consistent data** while data is being written → Snapshot isolation between writers and readers
2. Ability to **read incrementally from a large table** with good throughput → Optimized file source with scalable metadata handling

# Connecting the dots...


1. Ability to **read consistent data** while data is being written → Snapshot isolation between writers and readers
2. Ability to **read incrementally from a large table** with good throughput → Optimized file source with scalable metadata handling
3. Ability to **rollback** in case of bad writes → Time travel

# Connecting the dots...


1. Ability to **read consistent data** while data is being written → Snapshot isolation between writers and readers
2. Ability to **read incrementally from a large table** with good throughput → Optimized file source with scalable metadata handling
3. Ability to **rollback** in case of bad writes → Time travel
4. Ability to **replay historical data** along new data that arrived → Stream the backfilled historical data through the same pipeline

# Connecting the dots...


1. Ability to **read consistent data** while data is being written → Snapshot isolation between writers and readers
2. Ability to **read incrementally from a large table** with good throughput → Optimized file source with scalable metadata handling
3. Ability to **rollback** in case of bad writes → Time travel
4. Ability to **replay historical data** along new data that arrived → Stream the backfilled historical data through the same pipeline
5. Ability to **handle late arriving data** without having to delay downstream processing → Stream any late arriving data added to the table as they get added

# Connecting the dots...

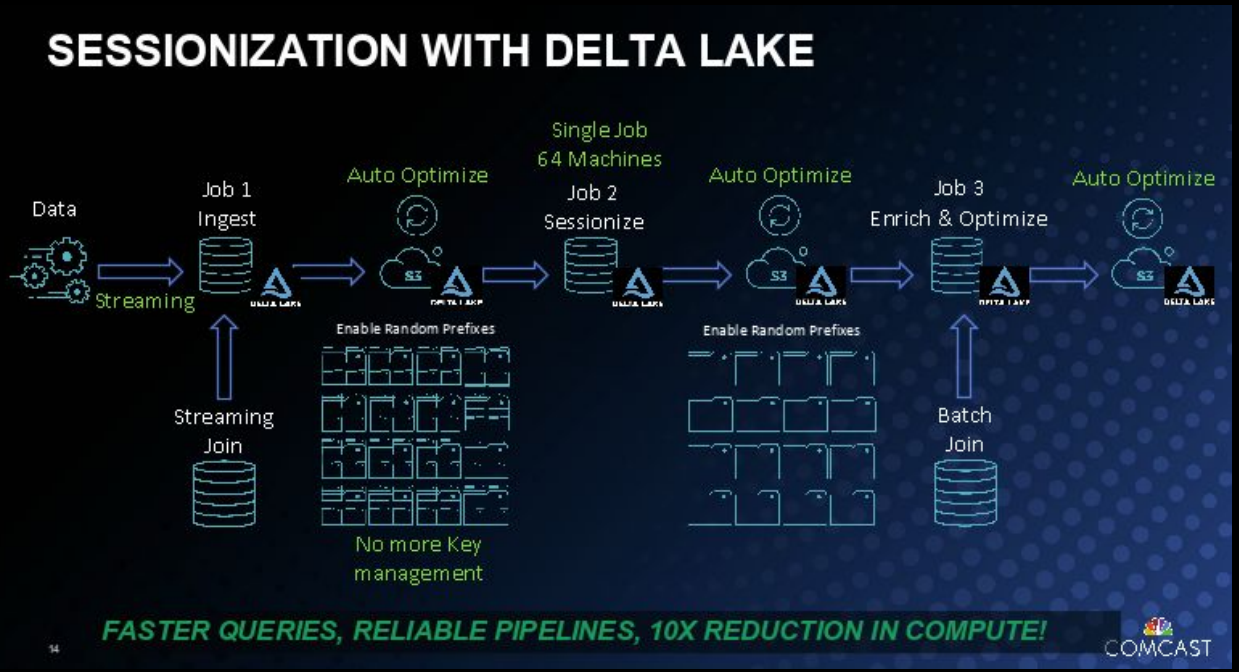

1. Ability to **read consistent data** while data is being written → Snapshot isolation between writers and readers
2. Ability to **read incrementally from a large table** with good throughput → Optimized file source with scalable metadata handling
3. Ability to **rollback** in case of bad writes → Time travel
4. Ability to **replay historical data** along new data that arrived → Stream the backfilled historical data through the same pipeline
5. Ability to **handle late arriving data** without having to delay downstream processing → Stream any late arriving data added to the table as they get added

# 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")
```

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

# Notebook from today's session

Try the notebook from today's webinar on Databricks Community Edition!

Download the notebook at https://dbricks.co/sais-eu19-delta

# Delta Lake Connectors

Standardize your big data storage with an open format accessible from various tools

# Delta Lake Partners and Providers

More and more partners and providers are jumping working with Delta Lake


Google Dataproc

Privacera



Informatica

Streamsets

WANDisco

Qlik

# Users of Delta Lake

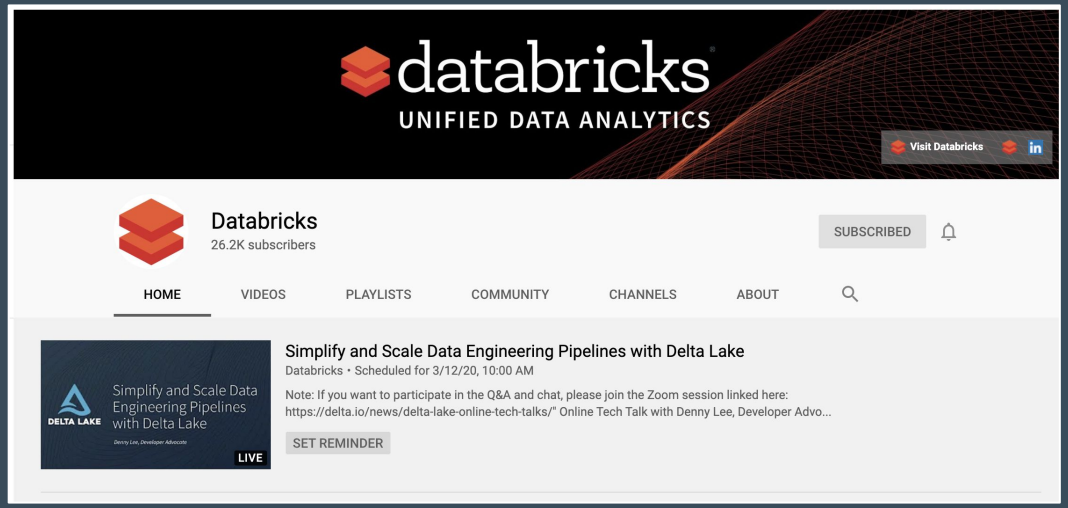
Tencent 腾讯 · Alibaba Group · COMCAST · viacom · ciena · JAM CITY

edmunds · databricks · healthdirect Australia · Booz | Allen | Hamilton · McAfee Together is power. · CONDÉ NAST

TILTING POINT · Mc Graw Hill Education · acorns · EVERQUOTE · Namely · usermind

ZEISS · split · upwork · DOLLAR SHAVE CLUB · wehkamp

# Subscribe Today!




https://dbricks.co/youtube

# Join us at Spark+AI Summit 2020


Get 20% off using the discount code: **DennySAI020**