Beyond Lambda: Introducing Delta Architecture

Denny Lee, Developer Advocate

Who Am I?

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.

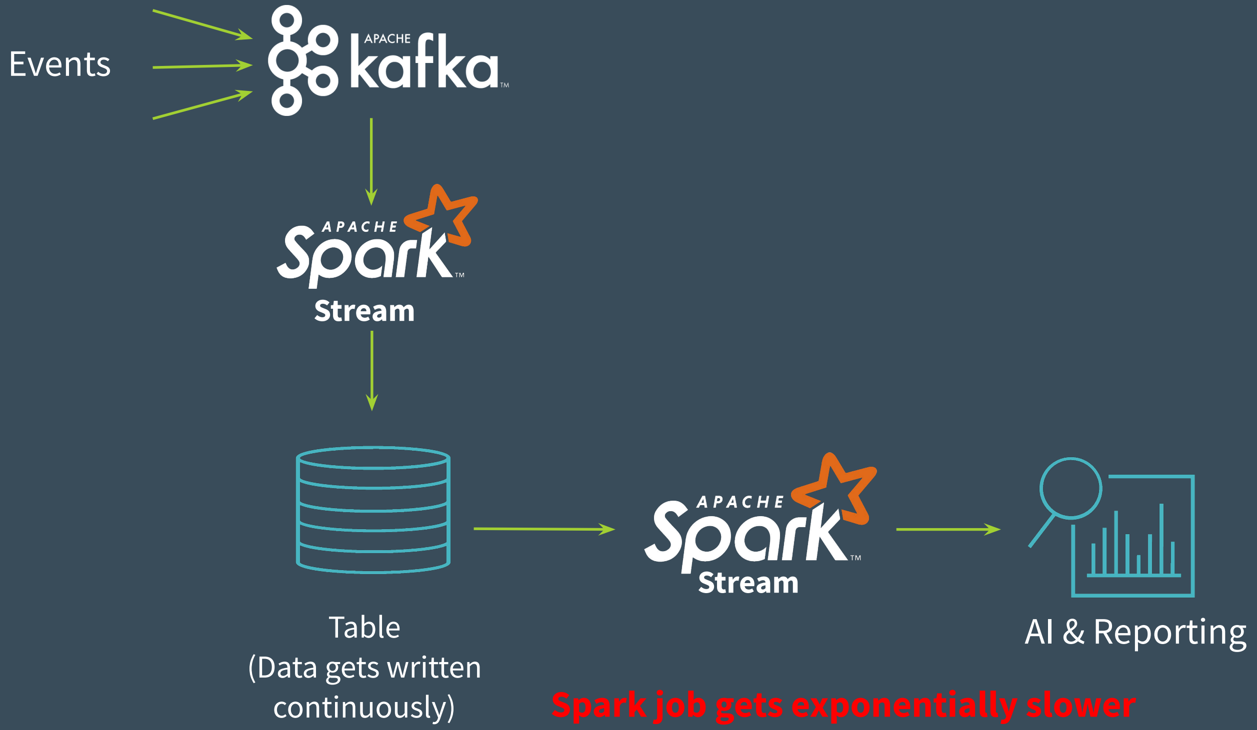
Agenda

1. What are the complexities to build a simple data pipeline?
2. Why these complexities exist?
3. Delta Lake and How it works?
4. Delta Architecture: The pattern Databricks' customers follow to build continuous pipelines with Delta Lake
5. The key characteristics & benefits of the Delta Architecture.

A Data Engineer's Dream...

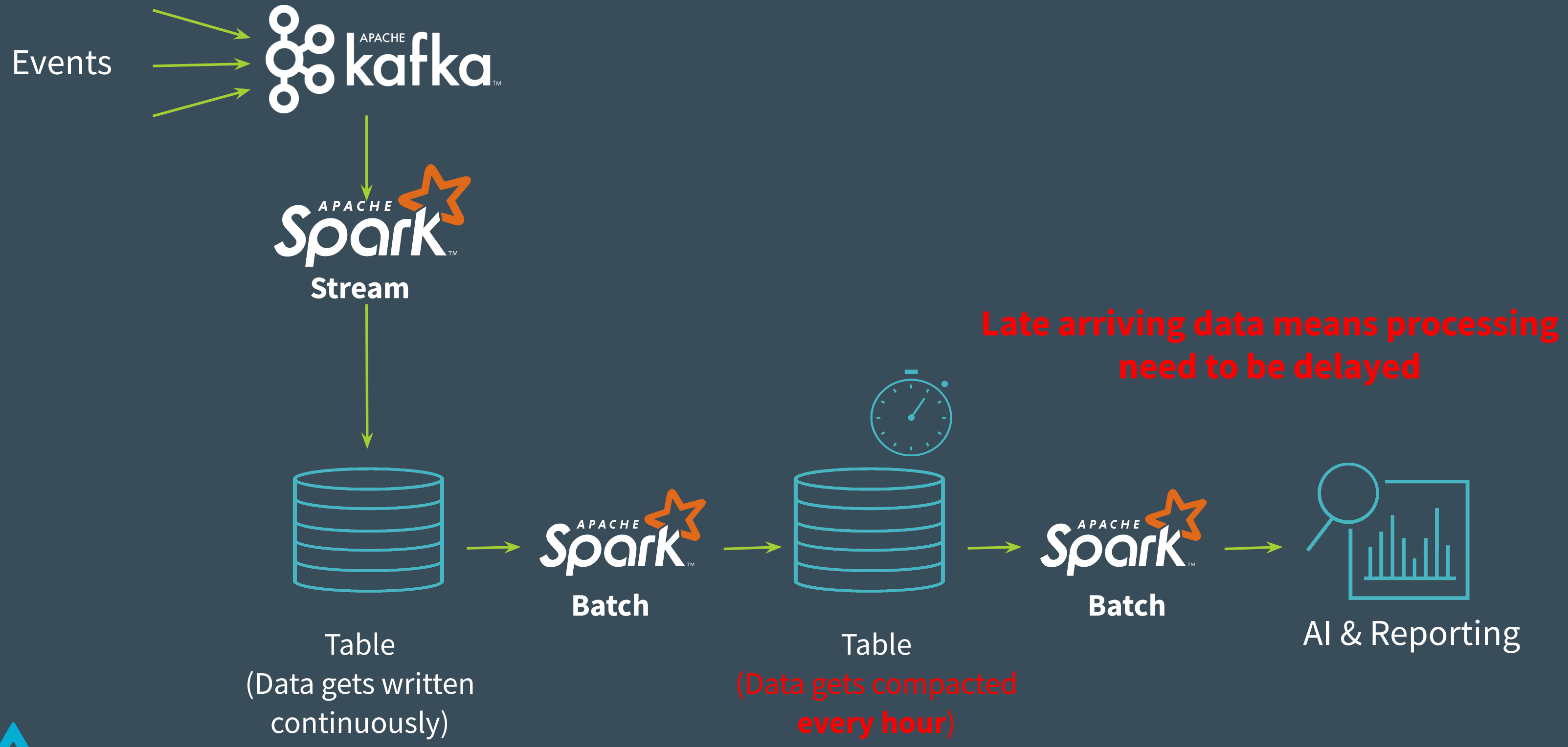
Process data **continuously** and **incrementally** as new data arrive in a **cost efficient way** without having to *choose* between batch or streaming

The Data Engineer’s Journey…

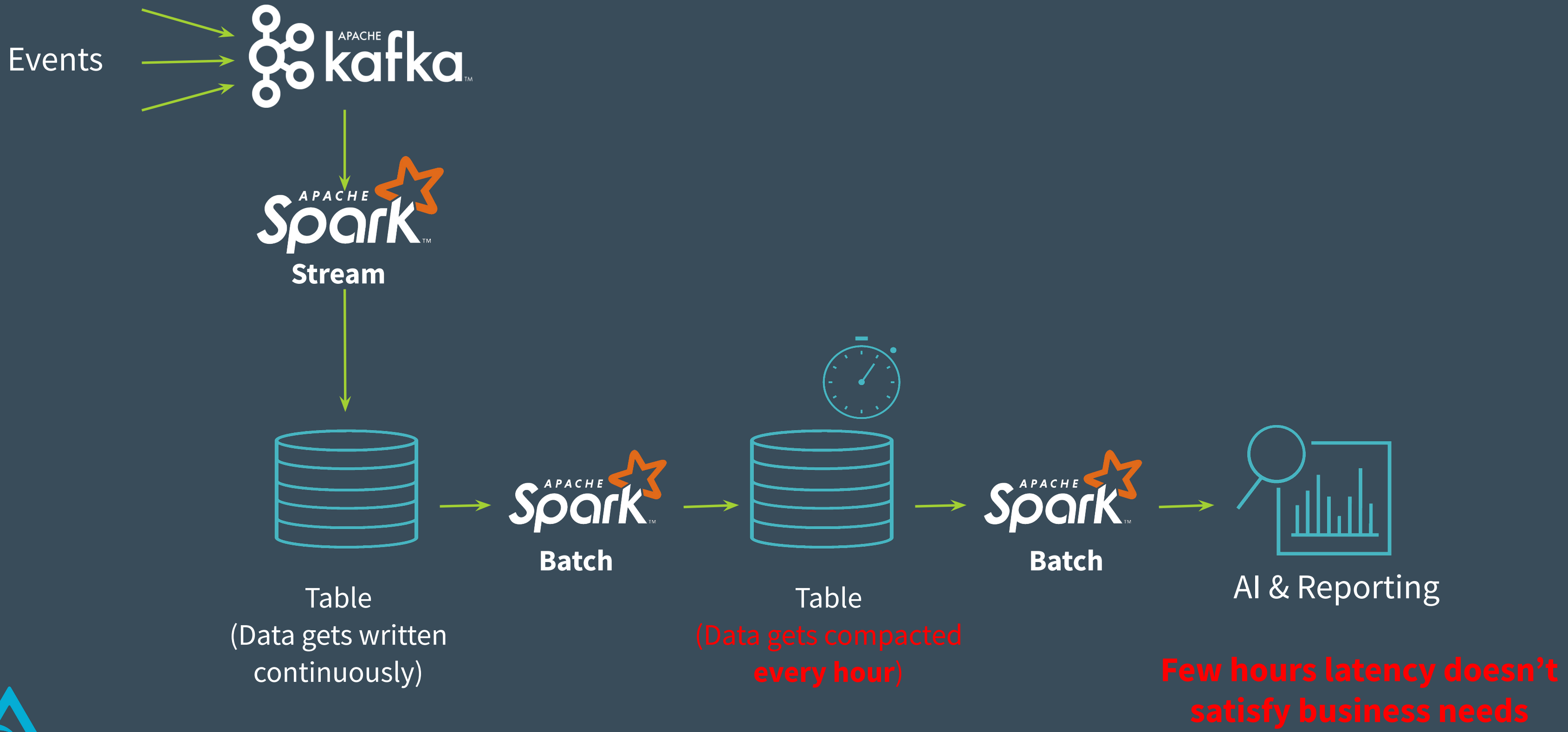


Spark job gets exponentially slower with time due to small files.

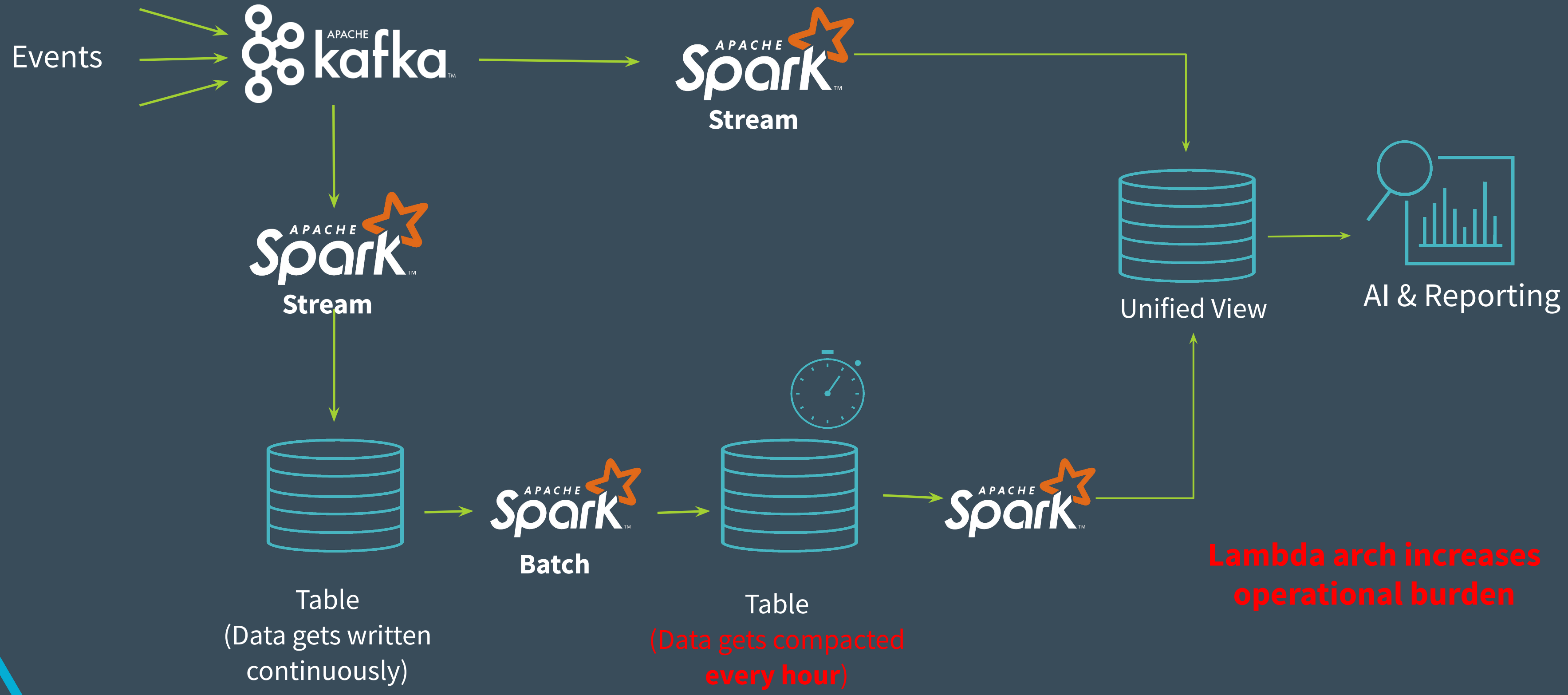
The Data Engineer's Journey...



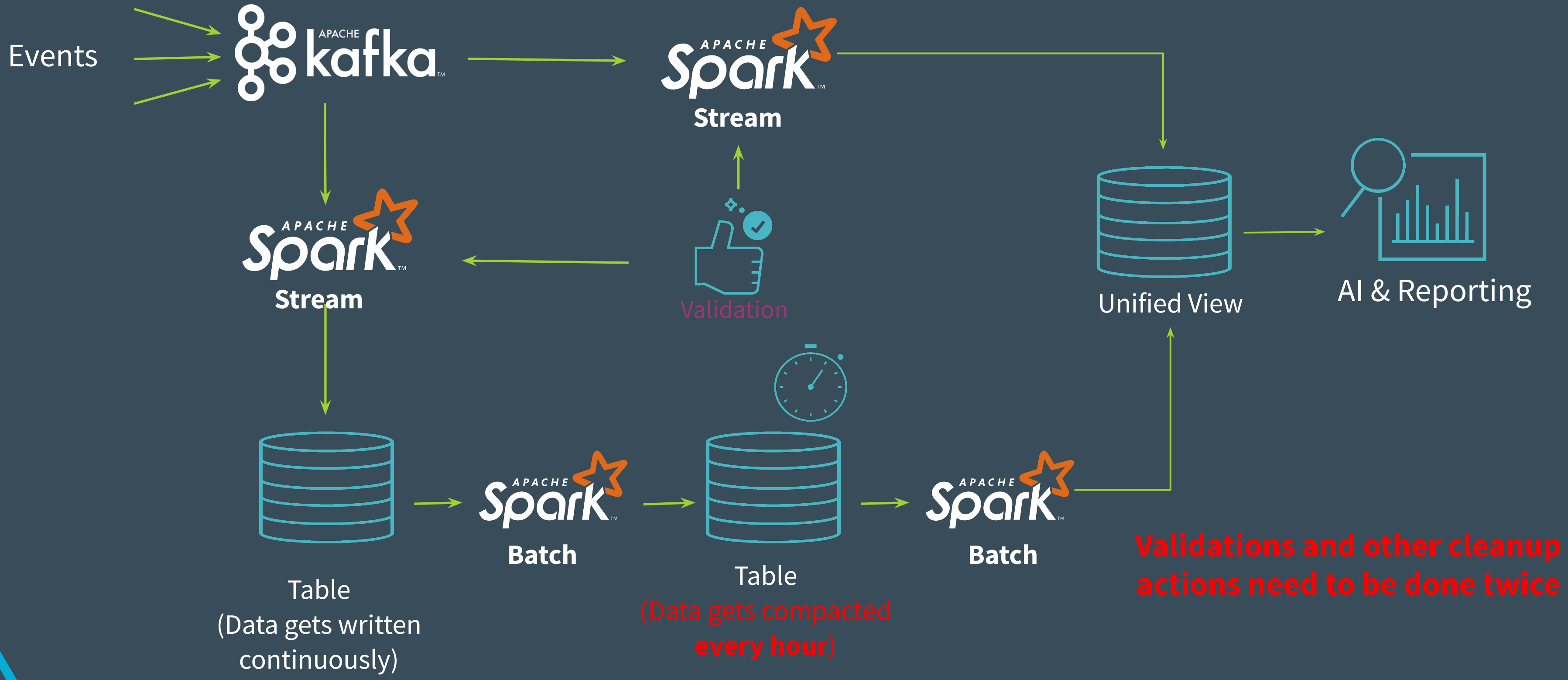
The Data Engineer's Journey...



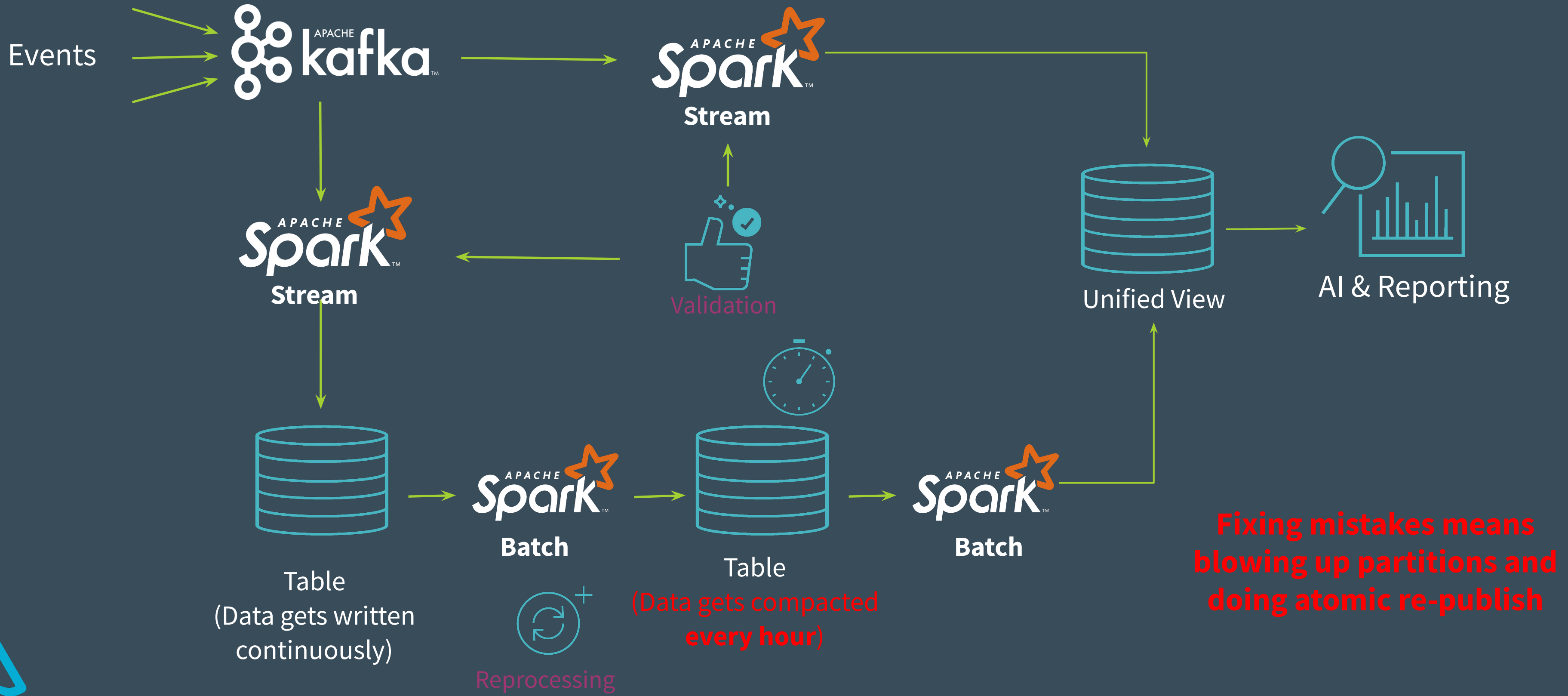
The Data Engineer's Journey...



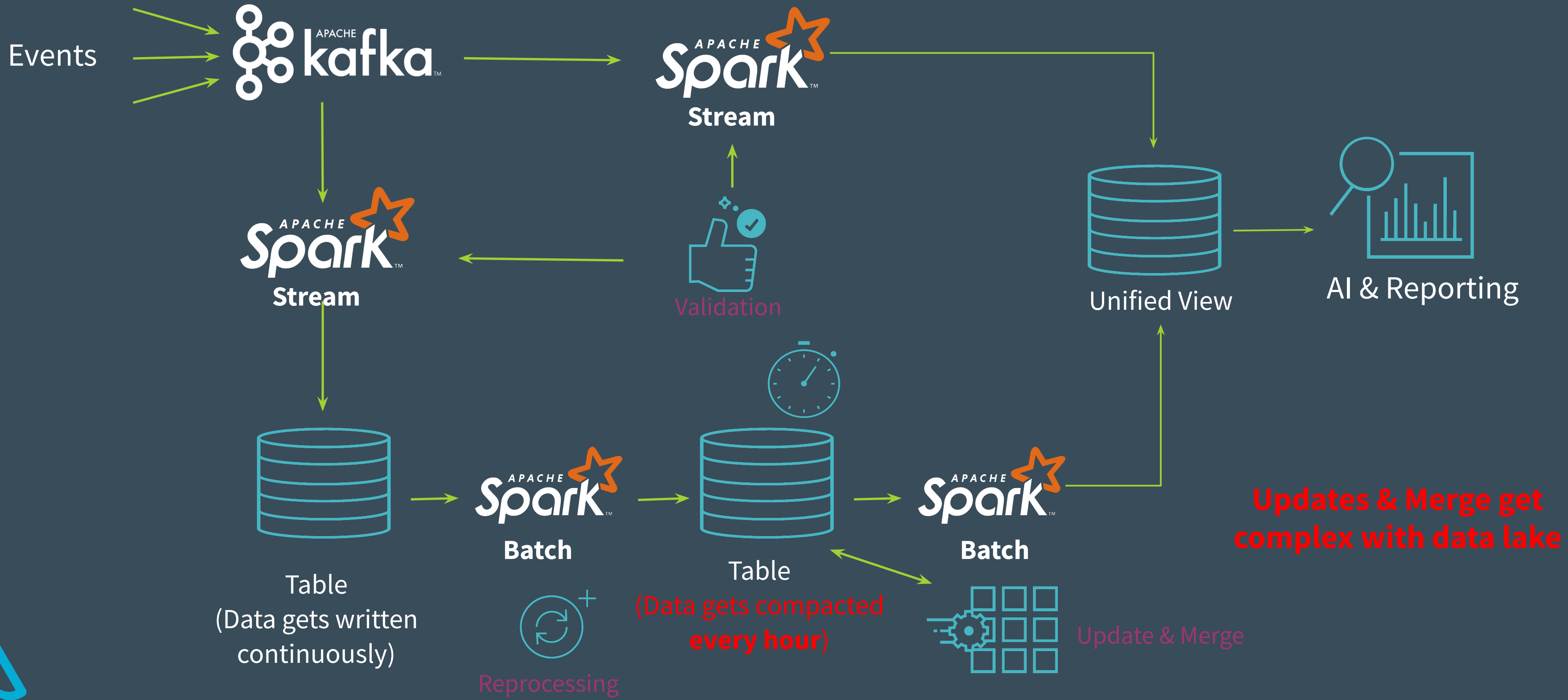
The Data Engineer's Journey...



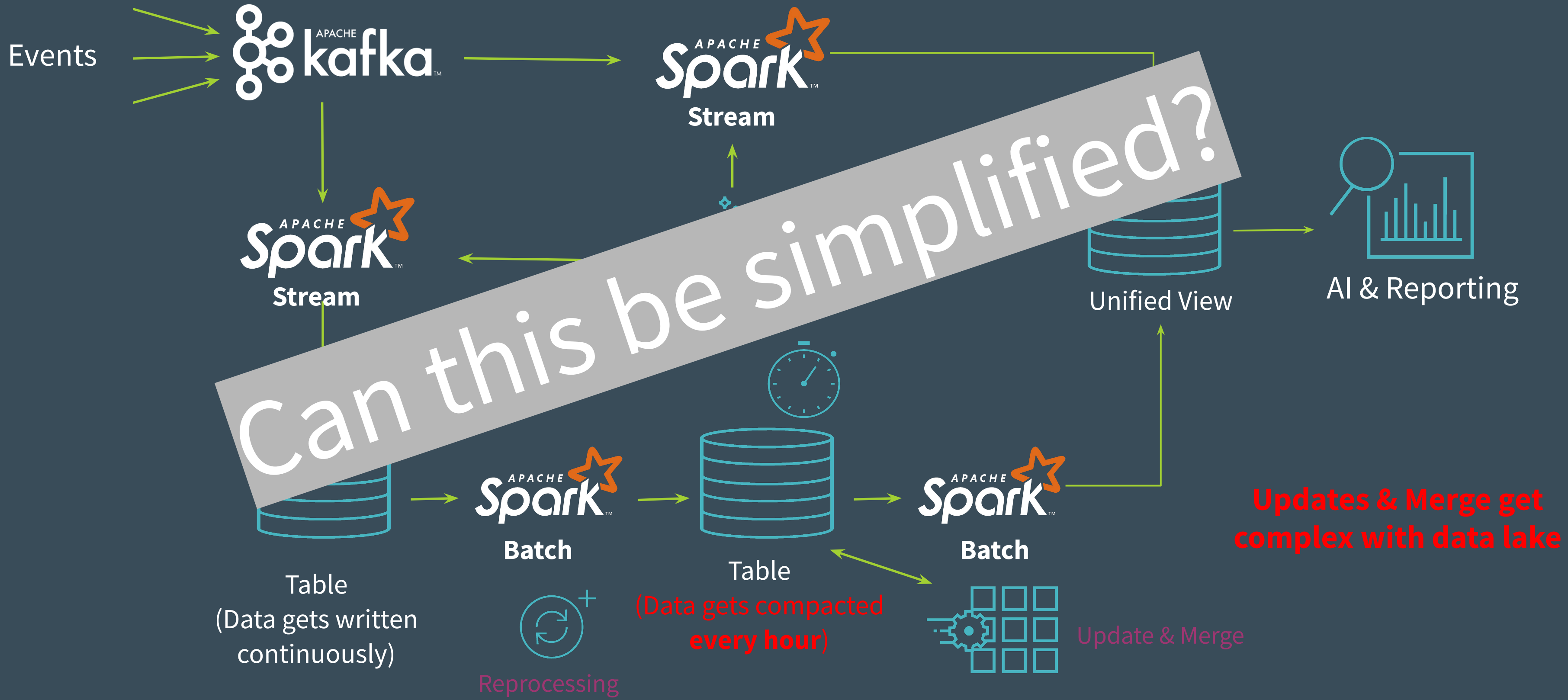
The Data Engineer's Journey...



The Data Engineer's Journey...



The Data Engineer's Journey...



What was 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

How Delta Lake Works?

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

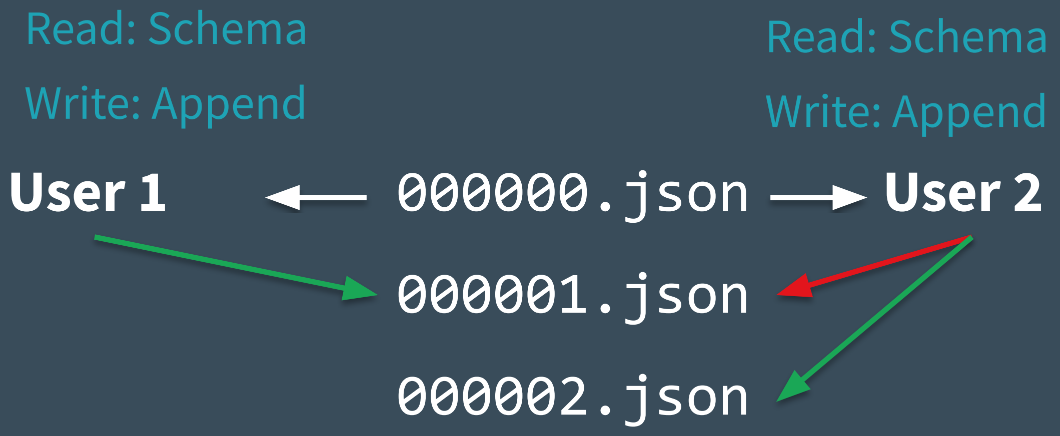
```
000000.json
000001.json
...
```

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

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

Solving Conflicts Optimistically

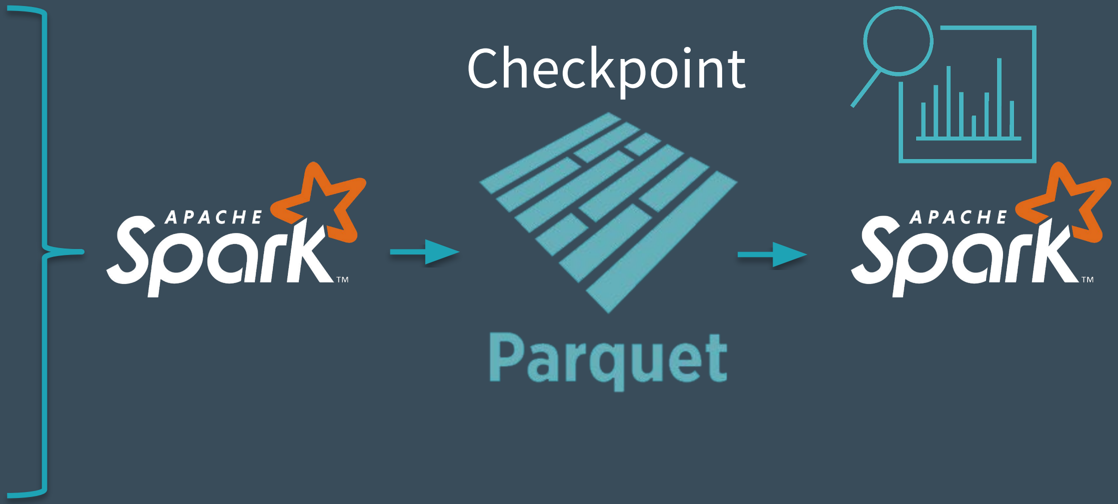
1. Record start version
2. Record reads/writes
3. Attempt commit
4. If someone else wins, check if anything you read has changed.
5. Try again.



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
```



The Delta Architecture

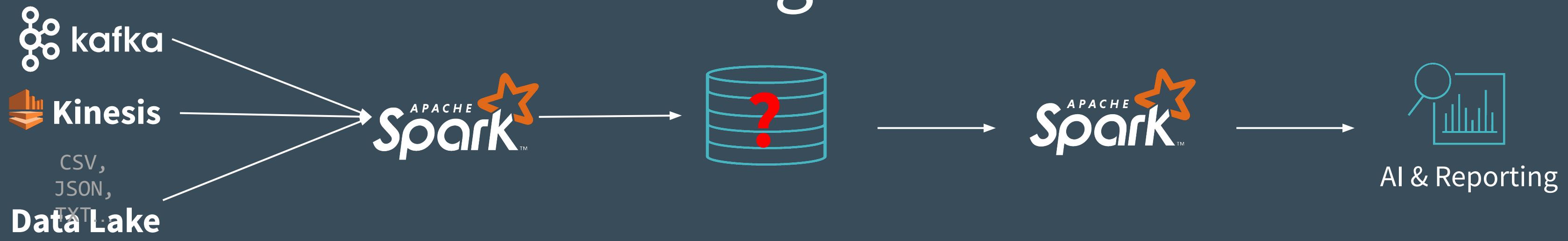
Connecting the dots...

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...

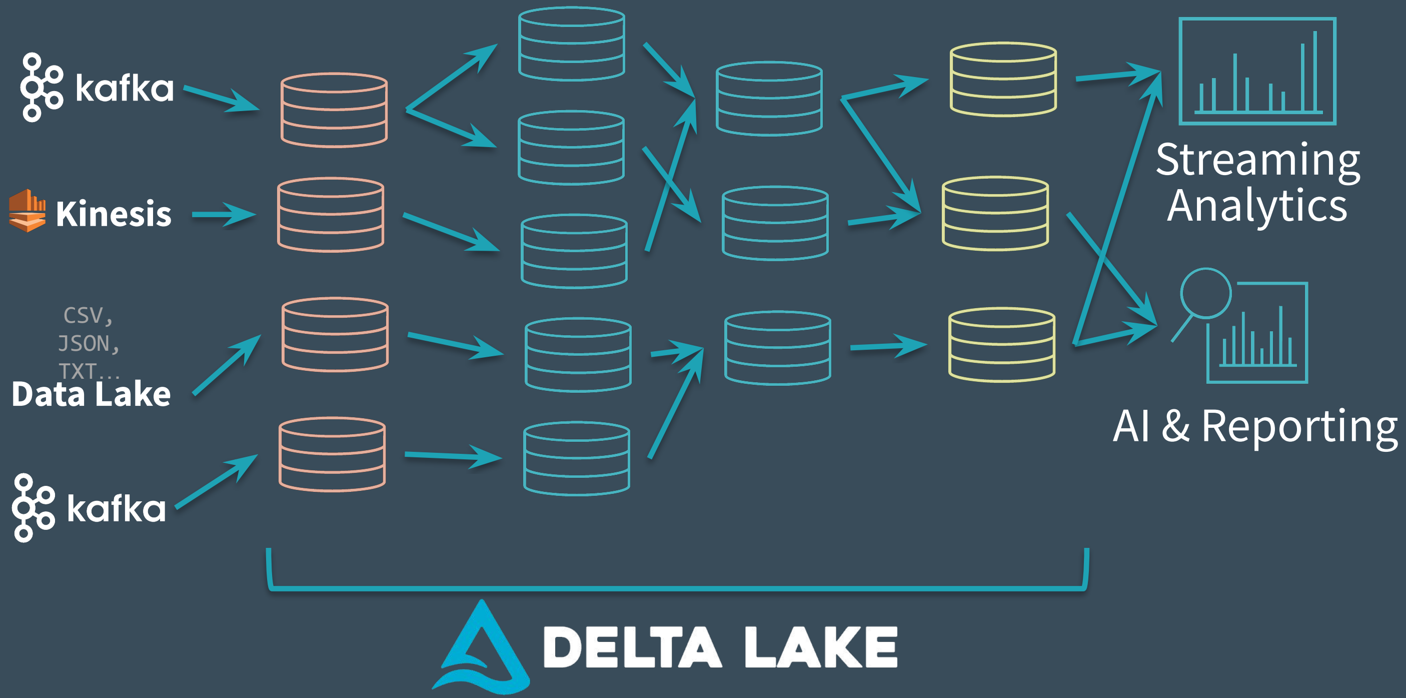
1. Ability to **read consistent data** while data is being written → Snapshot isolation between writers and readers
2. Ability to **read incrementally from a large table** with good throughput → Optimized file source with scalable metadata handling
3. Ability to **rollback** in case of bad writes → Time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

The Delta Architecture

A continuous data flow model to unify batch & streaming



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

Characteristics of the Delta Architecture

#1. Adopt continuous data flow model

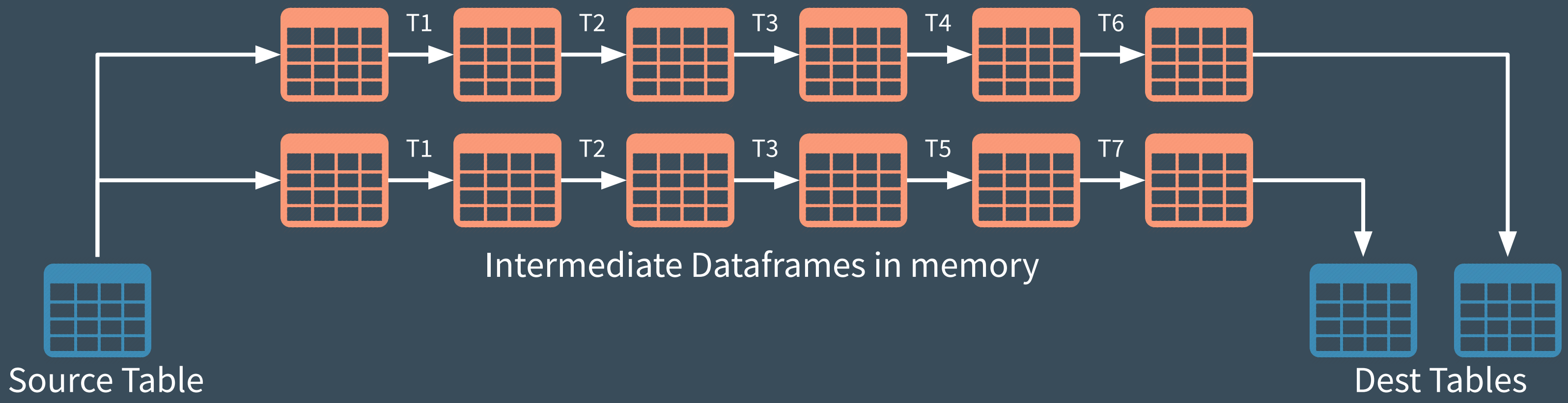
Stream to and from a Delta Lake table whenever possible.

- **Unify batch and streaming.** Same engine. Same APIs. Same user code. No need to reason about system complexities separately.
- **Incrementally load the new data efficiently.** No need to do state management on what are the new files added.
- **Process the data quickly as it arrives** without any delays.

#2. Use Intermediate Hops

Materialize DataFrames wherever applicable; especially when large number of transformations are involved. Materialization could be for:

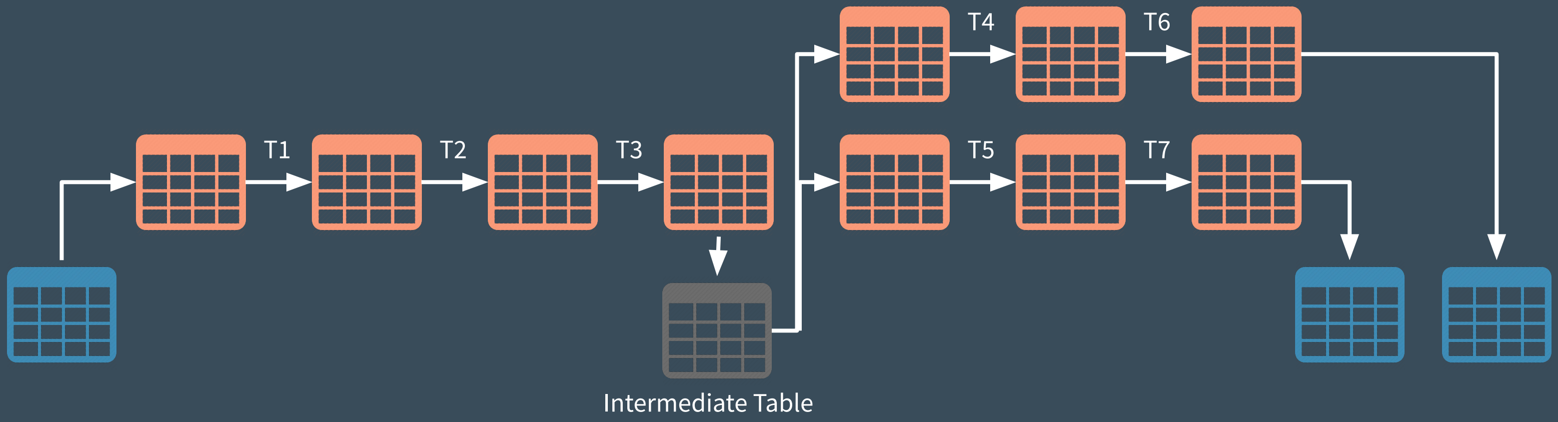
- Fault recovery
- Easy troubleshooting
- Multiple consumers expected



#2. Use Intermediate Hops

Materialize DataFrames wherever applicable; especially when large number of transformations are involved. Materialization could be for:

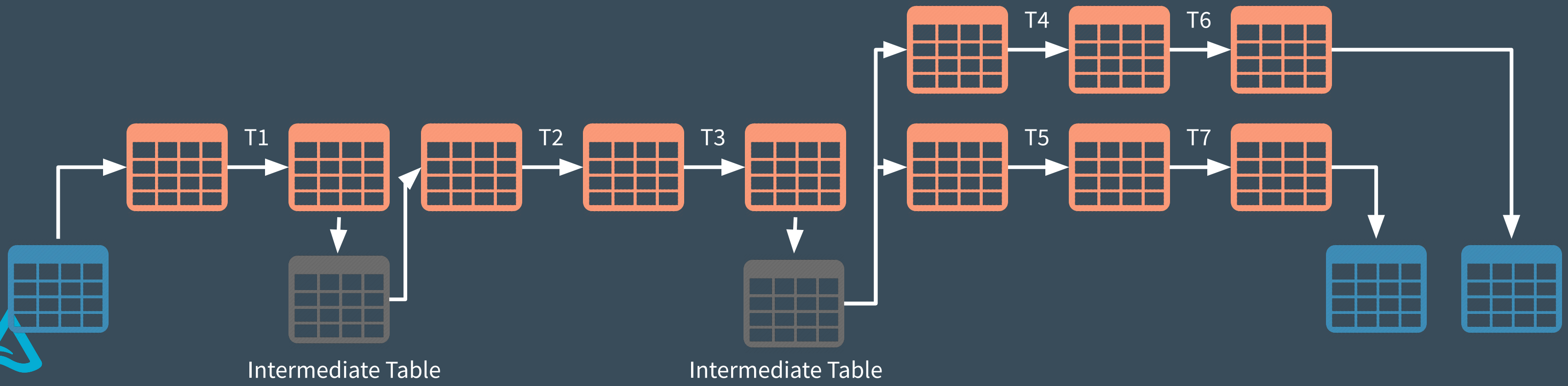
- Fault recovery
- Easy troubleshooting
- Multiple consumers expected



#2. Use Intermediate Hops

Materialize Dataframes wherever applicable; especially when large number of transformations are involved. Materialization will help with:

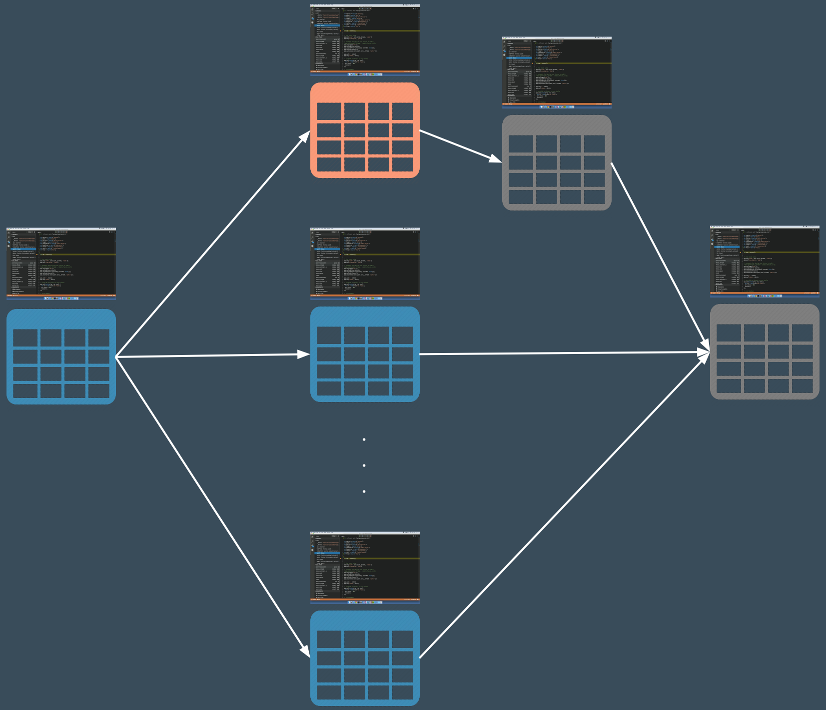
- Fault recovery
- Easy troubleshooting
- Multiple consumers expected



#3. Cost vs Latency Trade Off

1. **Streams; data arriving continuously:** Have an always-on cluster continuously processing data.
2. **Frequent batches; data arriving every few minutes (say 30 mins):** Use a warm pool of machines. Turn off the cluster when idle. Start the cluster when data needs to be processed. Use streaming `Trigger.Once` mode.
3. **Infrequent batches; data arriving every few hours or days:** Turn off the cluster when idle. Start the cluster when data needs to be processed. Use streaming `Trigger.Once` mode.

#4. Reprocessing

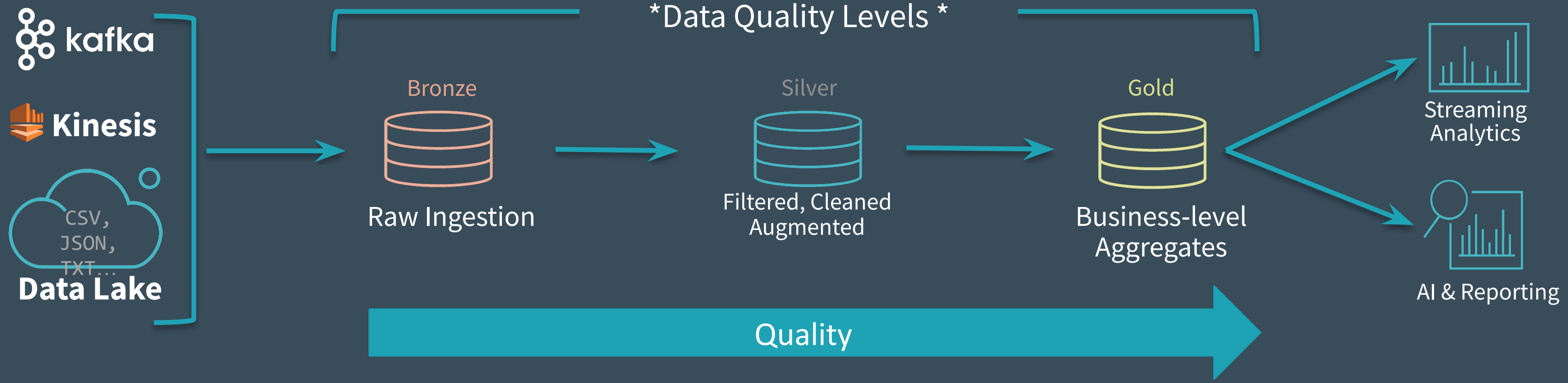


Infinite retention of raw data + stream = trival recomputation

- Simply clear out the result table and restart the stream
- Leverage cloud elasticity to quickly process initial backfill

#5. Tune Data Quality

- **Merge schemas automatically for raw ingestion tables:** Make sure you capture all the raw events without ignoring any data.
- **Enforce Schema on write for high quality analytics tables:** Make sure the data is clean and ready for analytics by enforcing schema restrictions (and data expectations in future)



Summary of the key characteristics

1. **Adopt a continuous data flow model** to unify batch and streaming
2. **Use intermediate hops** to improve reliability and troubleshooting
3. **Make the cost vs latency trade off** based on your use cases and business needs
4. **Optimize the storage layout** based on the access patterns
5. **Reprocess the historical data as needed** by simply clearing the result table and restarting the stream
6. **Incrementally improve the quality of your data** until it is ready for consumption with schema management options and data expectations.

Benefits of the Delta Architecture

1. Reduce end-to-end pipeline SLA.
   a. Organizations reduced pipeline SLAs from days and hours to minutes.
2. Reduce pipeline maintenance burden.
   a. Eliminate lambda architectures for minute-latency use cases.
3. Handle updates and deletes easily.
   a. Change data capture, GDPR, Sessionization, Deduplication use cases simplified.
4. Lower infrastructure costs with elastic, independent compute & storage
   a. Organizations reduce infrastructure costs by up to 10x

The Delta Architecture

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

STRUCTURED STREAMING + **DELTA LAKE**

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

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

ATTUNITY

tableau

talend

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

Thank you

DELTA LAKE

https://delta.io

Join us at Spark+AI Summit 2020

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