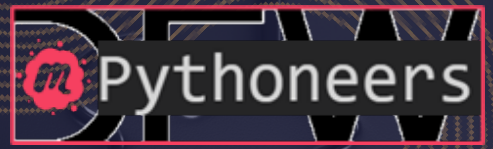


Real-Time, Data-Driven Decision-Making with Spark Streaming

May 4, 2023

AGENDA

01 From Big Data to Databricks

02 Streaming Analytics Basics

03 Demo, Demo, Demo!!!

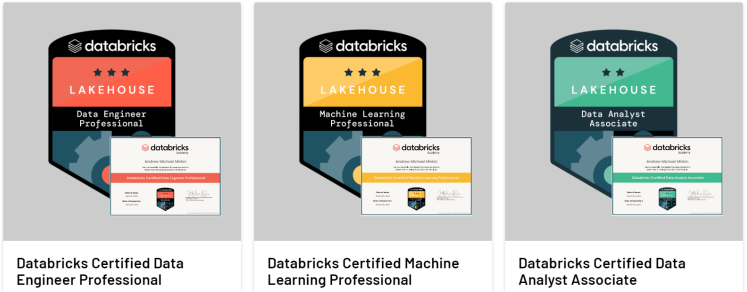
SPEAKER INFO

Previous Incarnations

- SQL Support/Consulting 2000-06
- Data Scientist 2010 – Now
- 5 Startups 2007-19
- Analytics Architect/Dev
- Entrepreneur

Current Incarnation

- Advisory Data/AI Architect



Databricks Certified Data Engineer Professional

Databricks Certified Machine Learning Professional

Databricks Certified Data Analyst Associate

From Big Data to Databricks

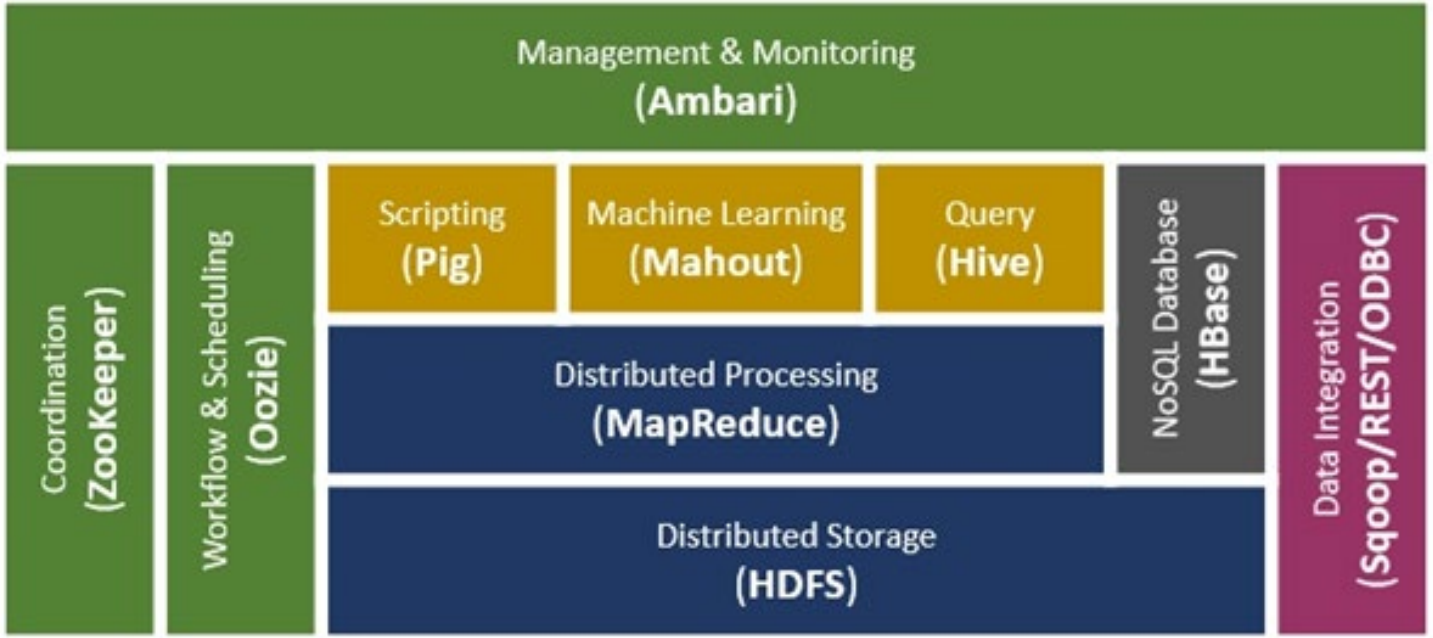
Why Big Data?

THIN THE LINES
BETWEEN
OPERATIONAL AND
ANALYTICAL

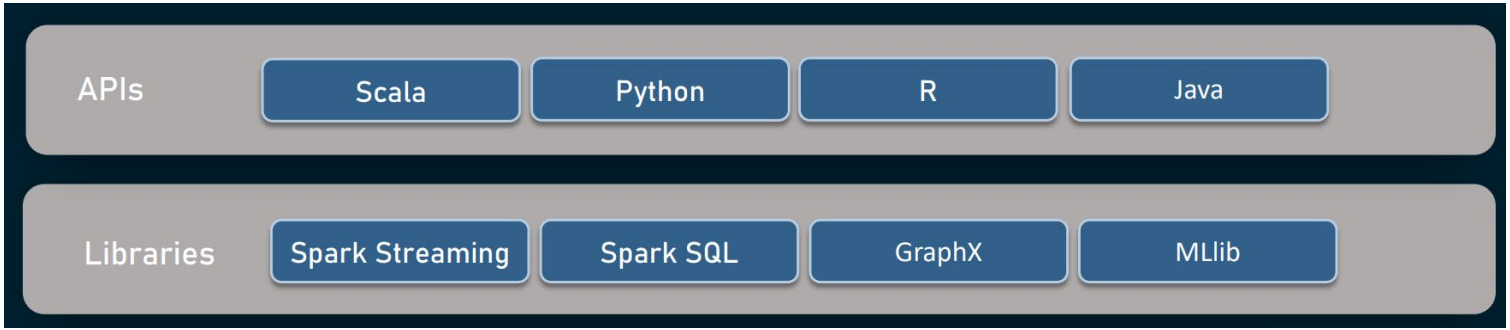
DECENTRALIZED
WORKSPACES
CENTRALIZED STORAGE

COMPUTE IS
PAY AS YOU GO

Hadoop Architecture (Pre-Spark)



Spark APIs



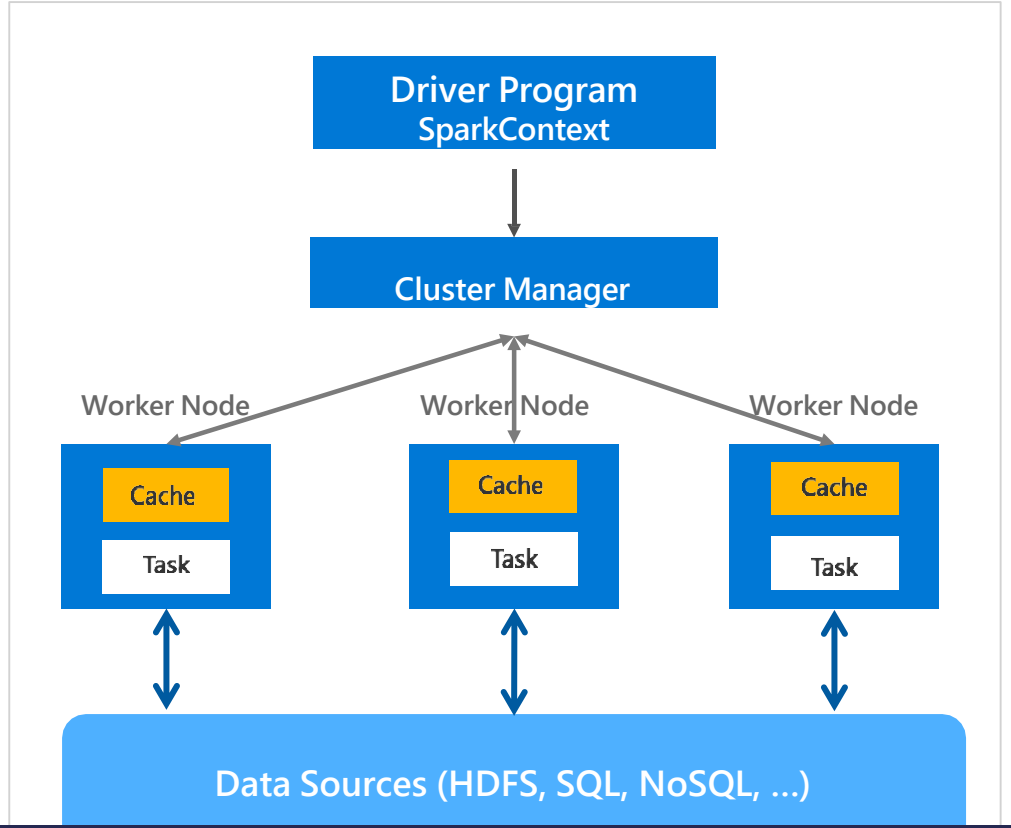
SEPARATION OF COMPUTE AND STORAGE

NATIVE JVM WITH SCALA DIALECT

SPARKR AND PYSPARK THUNKY INTEGRATIONS

Spark Architecture

- 'Driver' runs the user's 'main' function and executes the various parallel operations on the worker nodes.
- The results of the operations are collected by the driver
- The worker nodes read and write data from/to Data Sources including HDFS.
- Worker node also cache transformed data in memory as RDDs (Resilient Data Sets).
- Worker nodes and the Driver Node execute as VMs in public clouds (AWS, Google and Azure).



GraphX

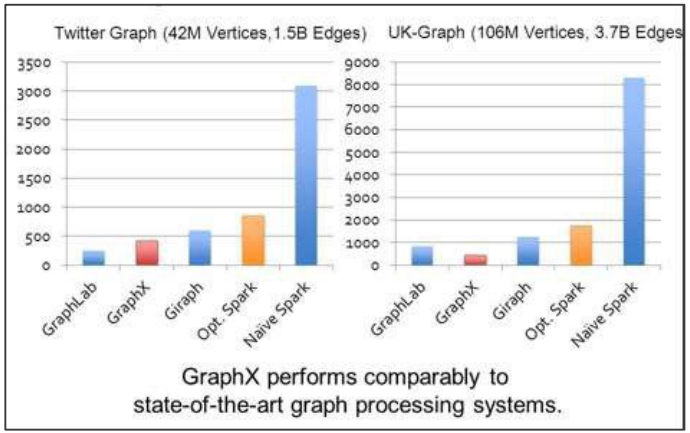
A set of APIs for graph and graph-parallel computation.

- Unifies ETL, exploratory analysis, and iterative graph computation within a single system.
- Developers can:
  - view the same data as both graphs and collections,
  - transform and join graphs with RDDs, and
  - write custom iterative graph algorithms using the Pregel API.
- Currently only supports using the Scala and RDD APIs.

Algorithms

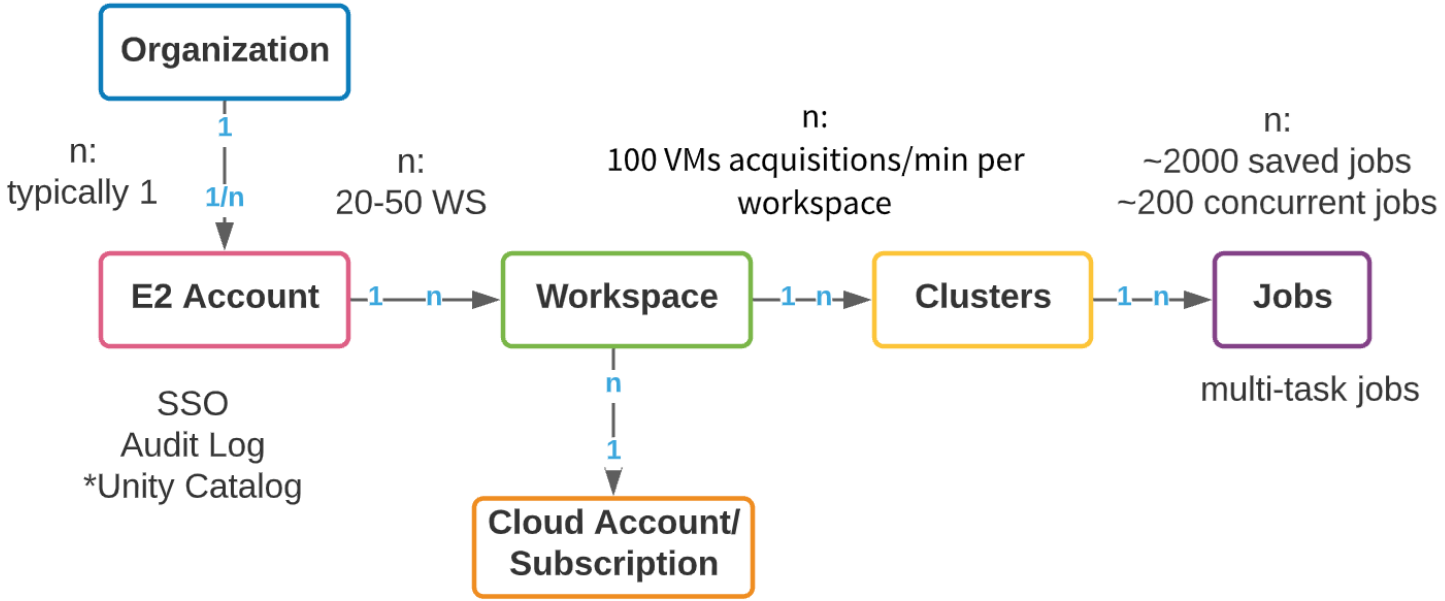
- PageRank
- Connected components
- Label propagation
- SVD++
- Strongly connected components
- Triangle count

PageRank Benchmark



Source: AMPLab

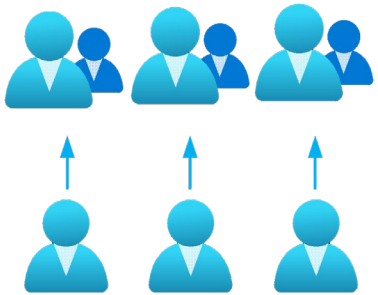
Databricks High Level Object Model



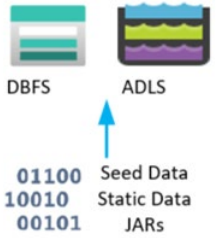
1 cloud account can be used for multiple workspace

Databricks Configuration

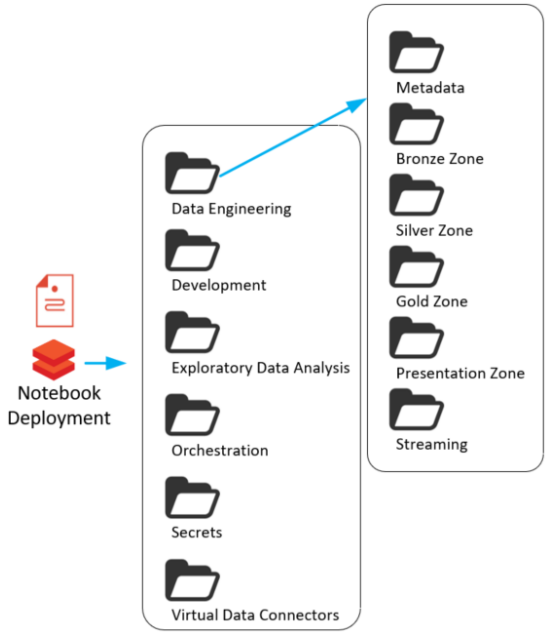
Databricks Security & IAM Roles



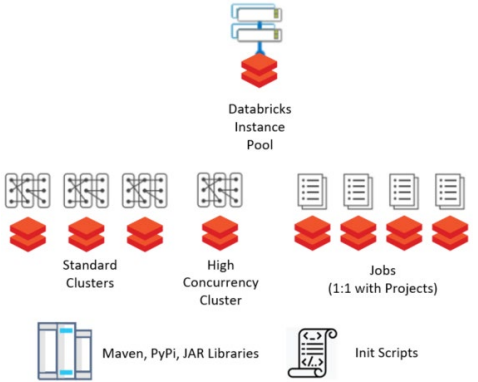
Upload Prerequisites



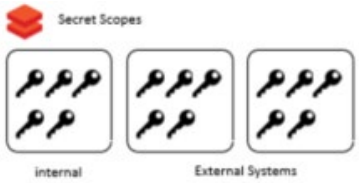
Databricks Workspace/Notebooks



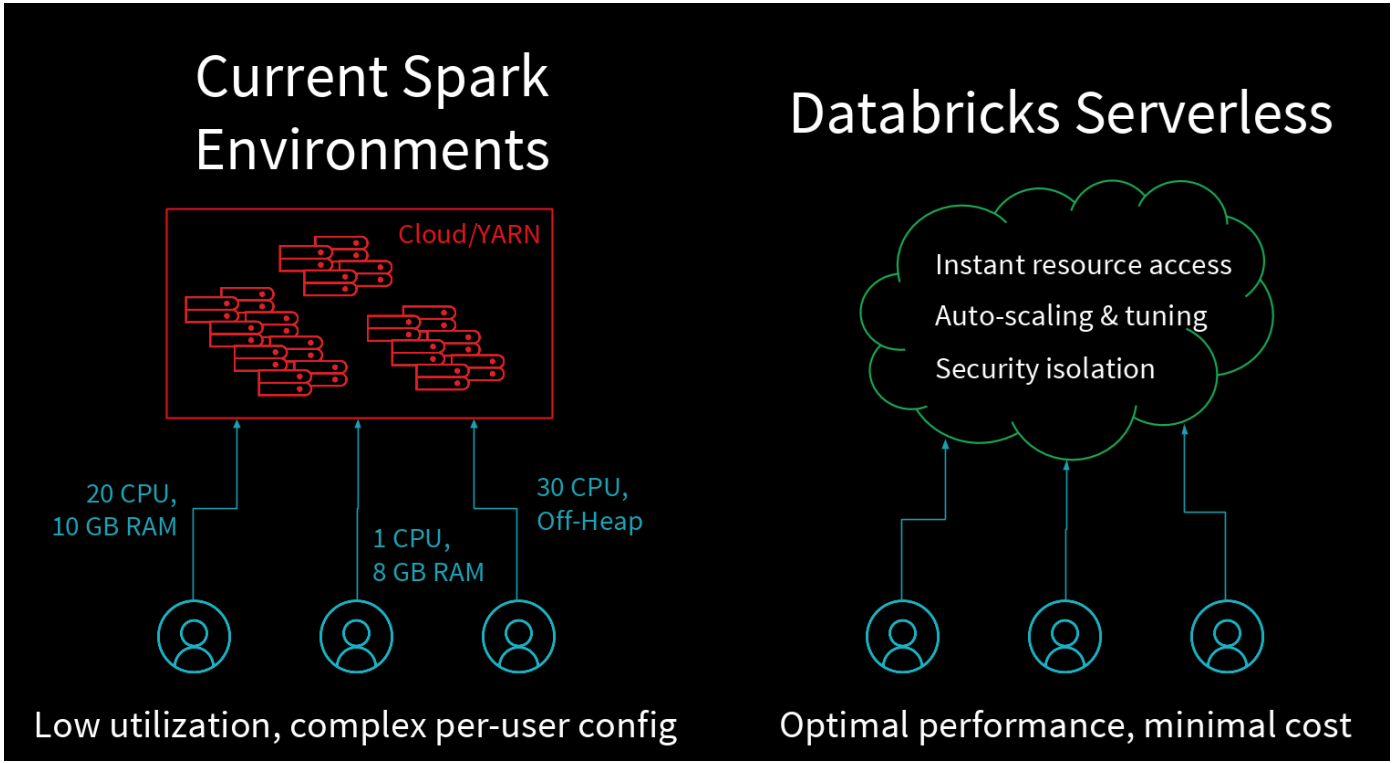
Pools, Clusters, Jobs, Libraries, Init Scripts



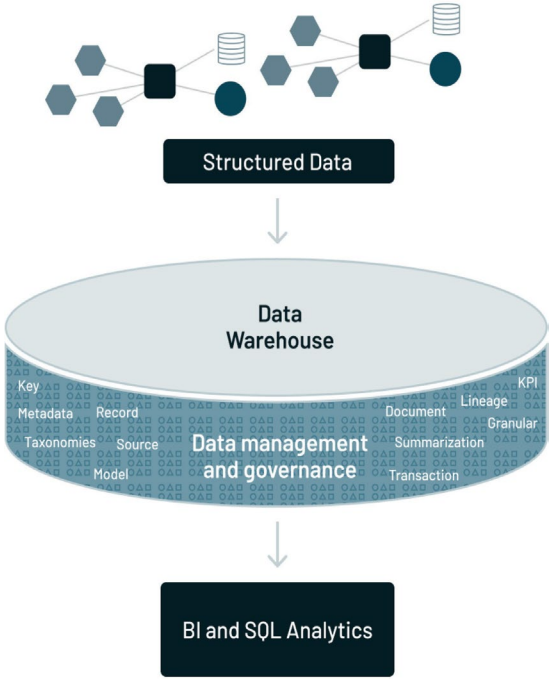
Secrets

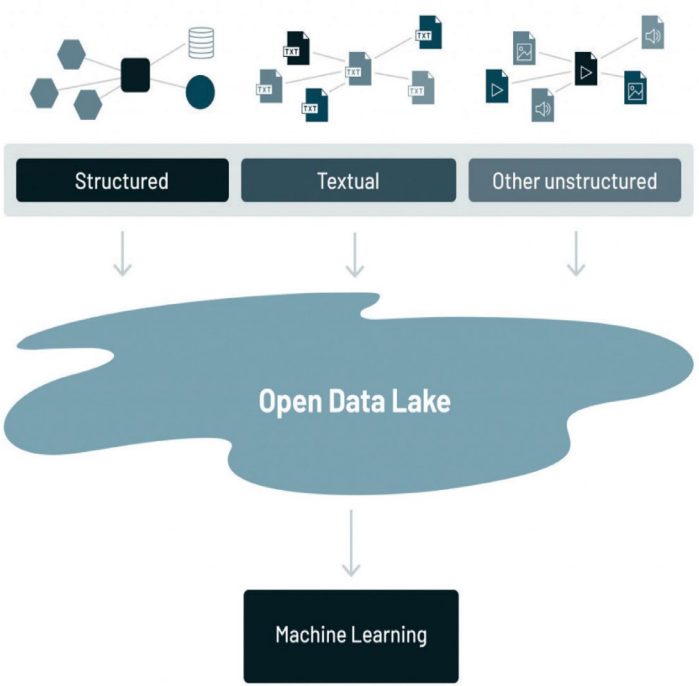


Databricks Serverless

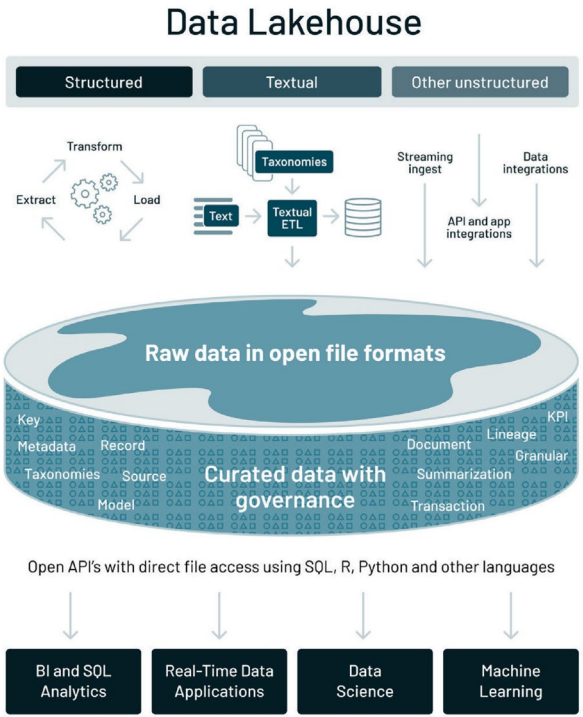


Clash of Architectures

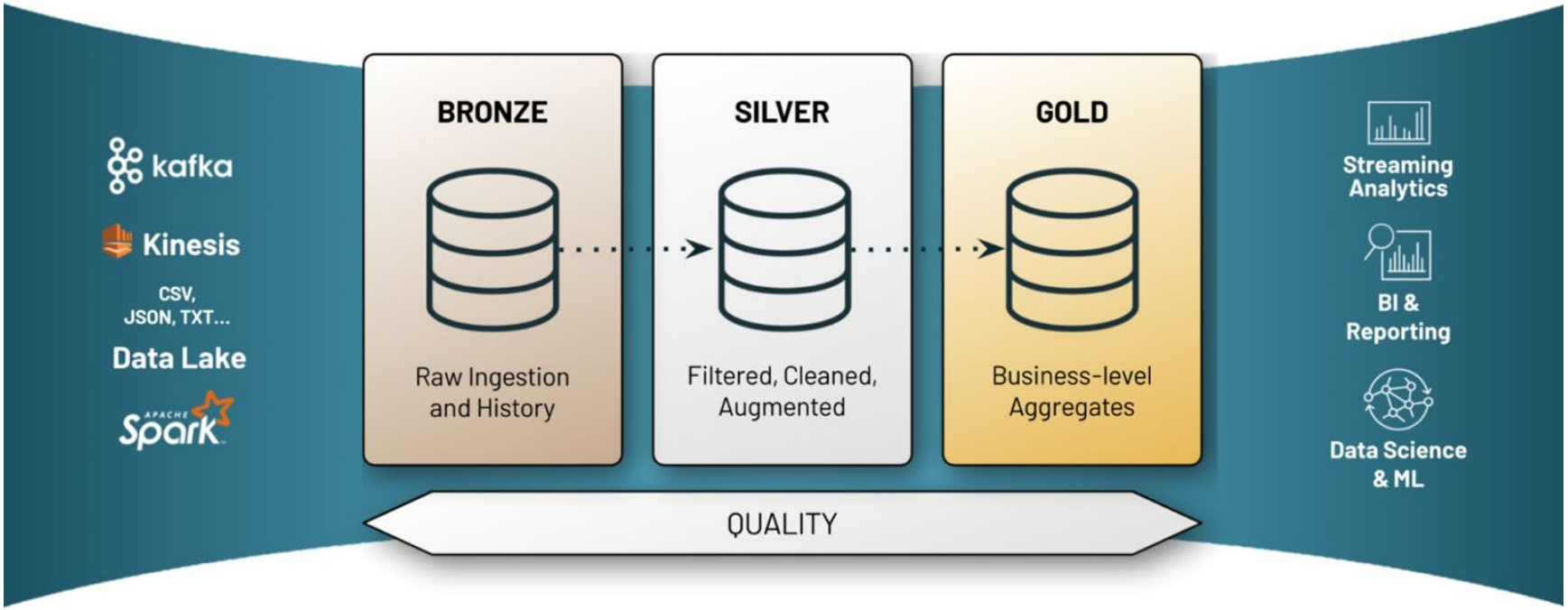




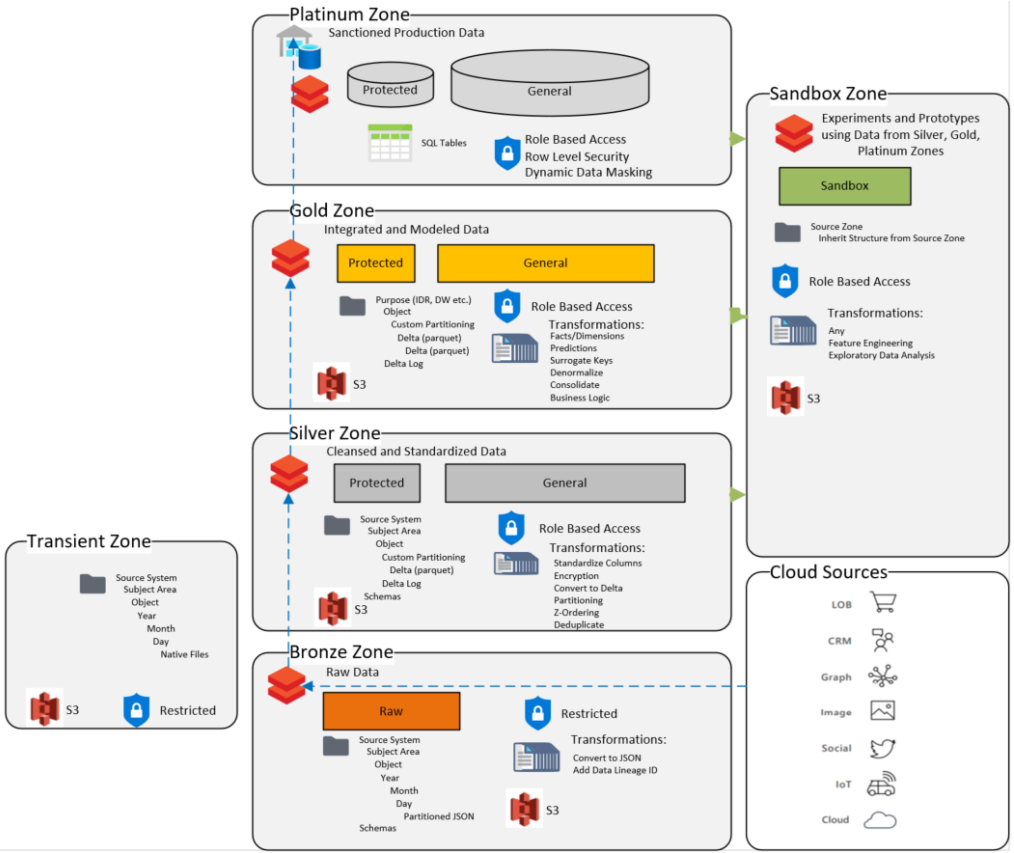
An Evolution of Architectures



Delta Live Tables



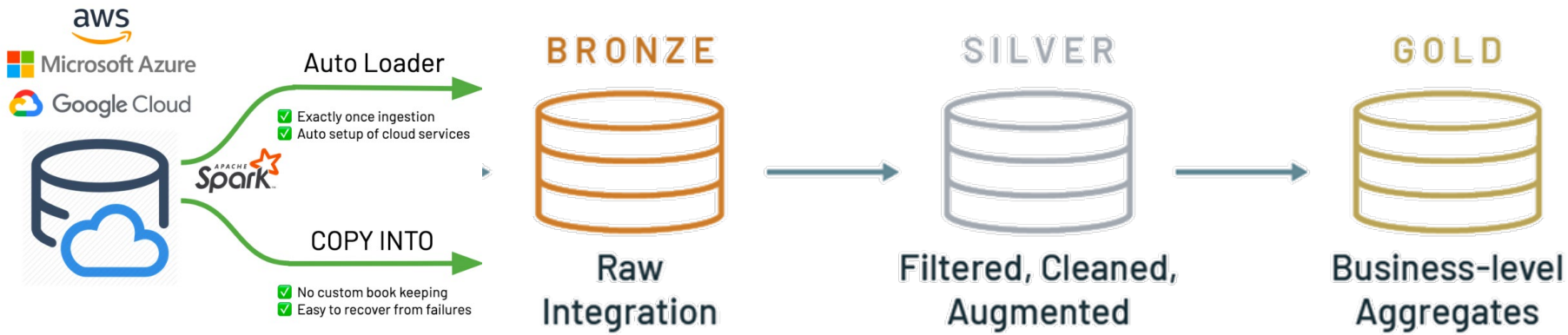
Data Lake Zones



Data Lake Data Sources

Structured Streaming

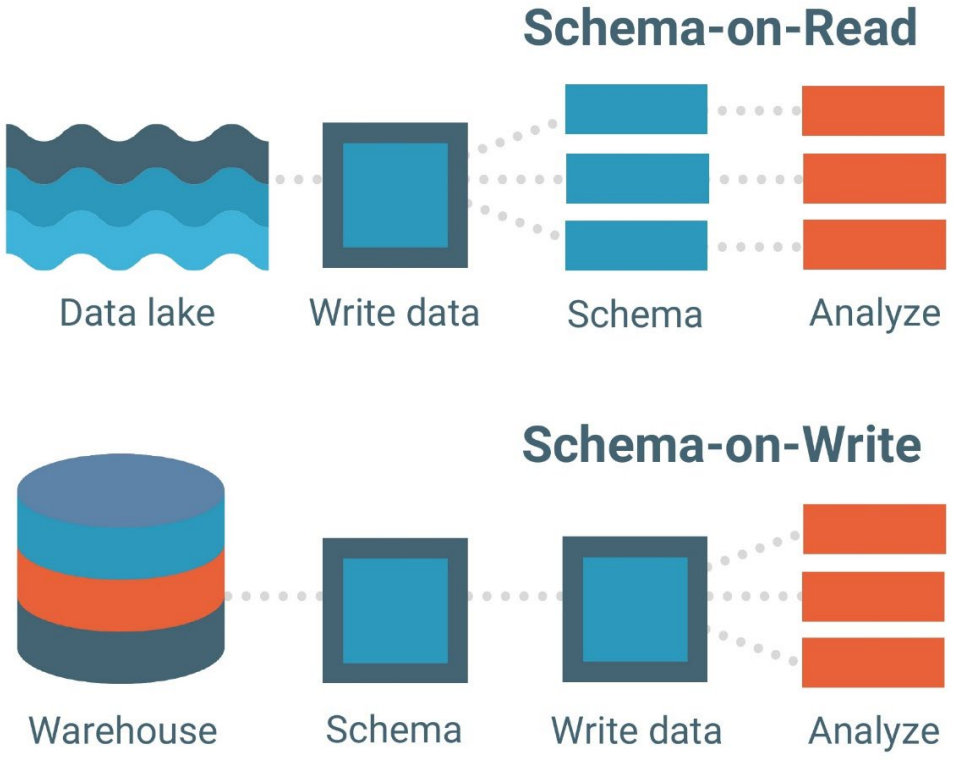
- Transform Avro payload
- Protocol buffers
- Apache Kafka
- AWS Kinesis
- Google Cloud DataFlow
- Apache Flink
- Azure Event Hubs
- Azure Synapse with Structured Streaming
- Databricks Auto-loader
- Azure Stream Analytics



Data Lake Enrichment - Schema Management

Common Schema Management Scenarios

- Schema Validation
- Schema Exception Routing
- Schema Evolution
- JSON Schema



Azure Deep Learning Platforms - Databricks

AZURE DATABRICKS

- Multiple Cloud Support
- Deeper Spark IP
- Cross-cloud support
- Single node support
- Maturer autoscale
- Tighter MLFlow integration
- Native SQL
- Delta Live Tables
- Azure Data Factory integration
- Notebook as Dashboard
- SQL Dashboards/ Queries/Alerts
- Unity Catalog
- Cloudfile/Auto-Loader
- Notebook based AutoML

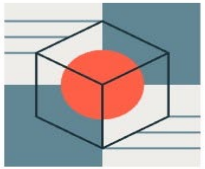
Hive Support
Java/Scala/Python
Spark Scaleout
GPU Support
SparkML Support

- SynapseML support
- Serverless or Dedicated SQL Pool
- Data warehousing
- Data Lake Tables
- Tighter Power BI integration
- .NET language support
- Big Data Scale Model Scoring with Zero Data Movement
- Transact-SQL compatibility
- Azure Purview Integration
- Azure Data Factory workspace integration

AZURE SYNAPSE

Unity Catalog

Unity Catalog is a unified governance solution for all data and AI assets including files, tables and machine learning models in your lakeho



Centralized governance for data and AI

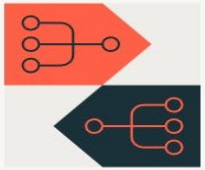
With a common governance model based on ANSI SQL, centrally govern files, tables, dashboards and ML models on any cloud.

Built-in data search and discovery

Quickly find, understand and reference data from across your data estate, boosting productivity.

Performance and scale

Benefit from enhanced query performance with low-latency metadata serving and auto-tuning of the tables.



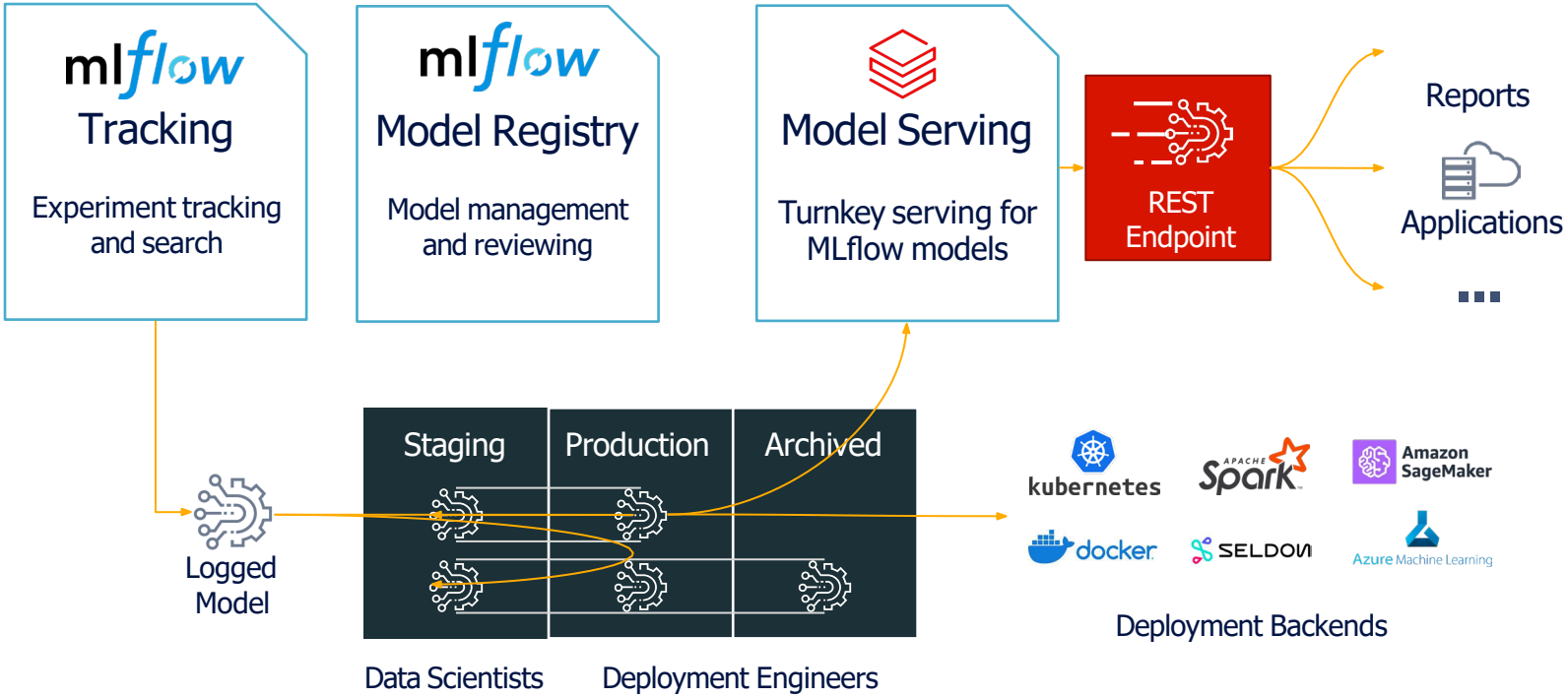
Automated lineage for all workloads

Create a unified, transparent view of your entire data ecosystem with automated and granular lineage for all workloads in SQL, R, Python, Scala and across all asset types — tables, notebooks, workflows and dashboards.

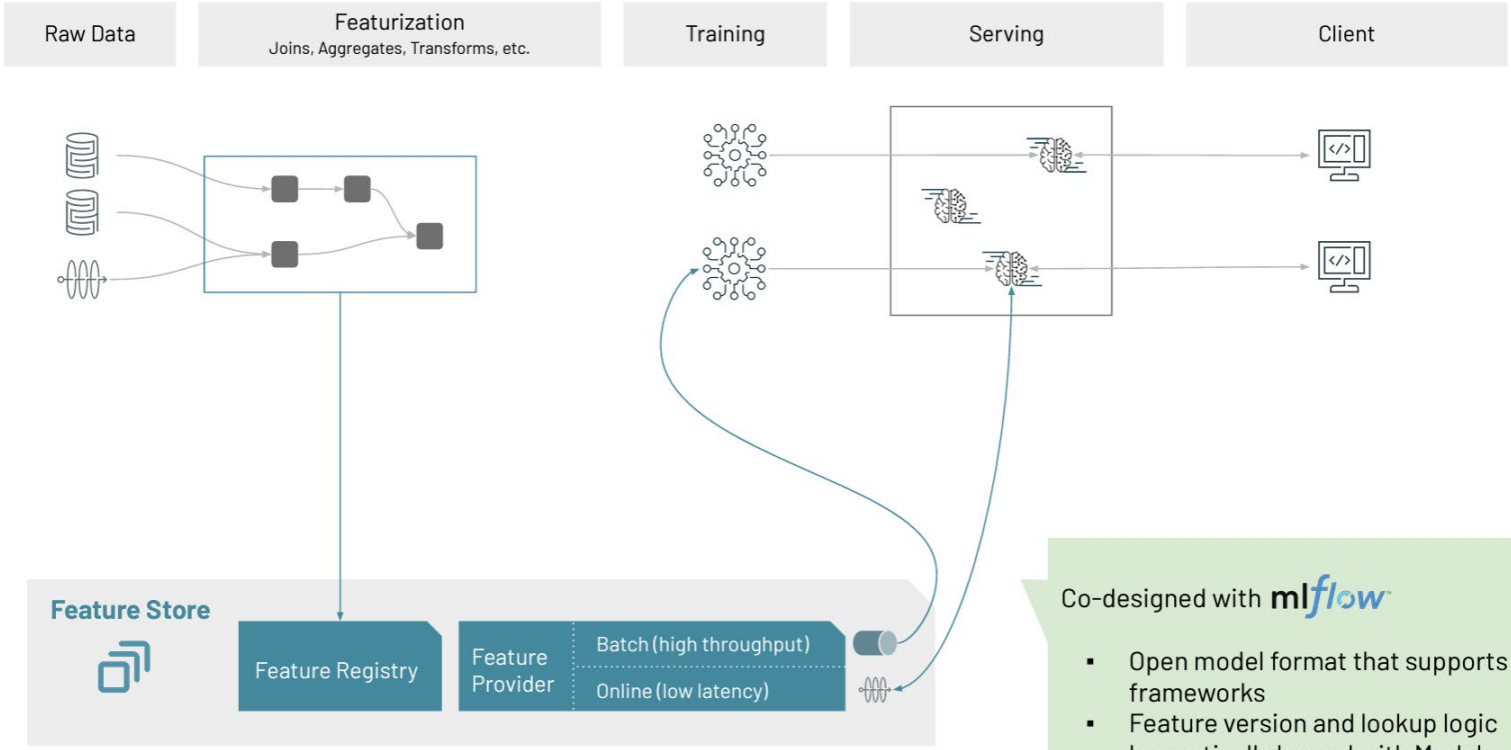
Integrated with your existing tools

Future-proof your data and AI governance with the flexibility to leverage your existing data catalogs and governance solutions.

Open Source MLOps - MLFlow



Data Lake Enrichment – AI/ML Modeling



Co-designed with mlflow

- Open model format that supports all ML frameworks
- Feature version and lookup logic hermetically logged with Model

Streaming Analytics Basics

Batch & Streaming - Overview

Batch vs. Streaming

Bucket

Batch processing

- High-latency, high delay time
- Large batches of data over time
- Hours to days

Faucet

Stream processing

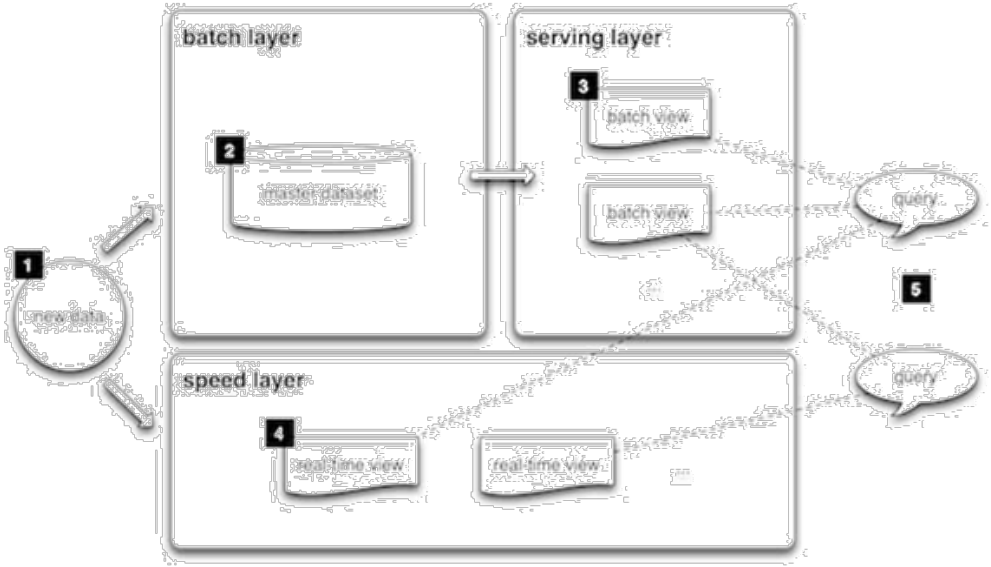
- Low-latency, low delay time
- Micro batches of data at once
- Milliseconds to seconds

https://www.g2.com/articles/what-is-a-data-lake

Batch & Streaming - Overview

Lambda Architecture

1. New data arrives to the data lake via batch or streaming
2. Batch layer processes data on disk
3. Serving layer transforms and caches
4. Speed layer processes real time data in memory
5. Clients query for a combination of real time streaming and batch data



Batch & Streaming – Native Execution

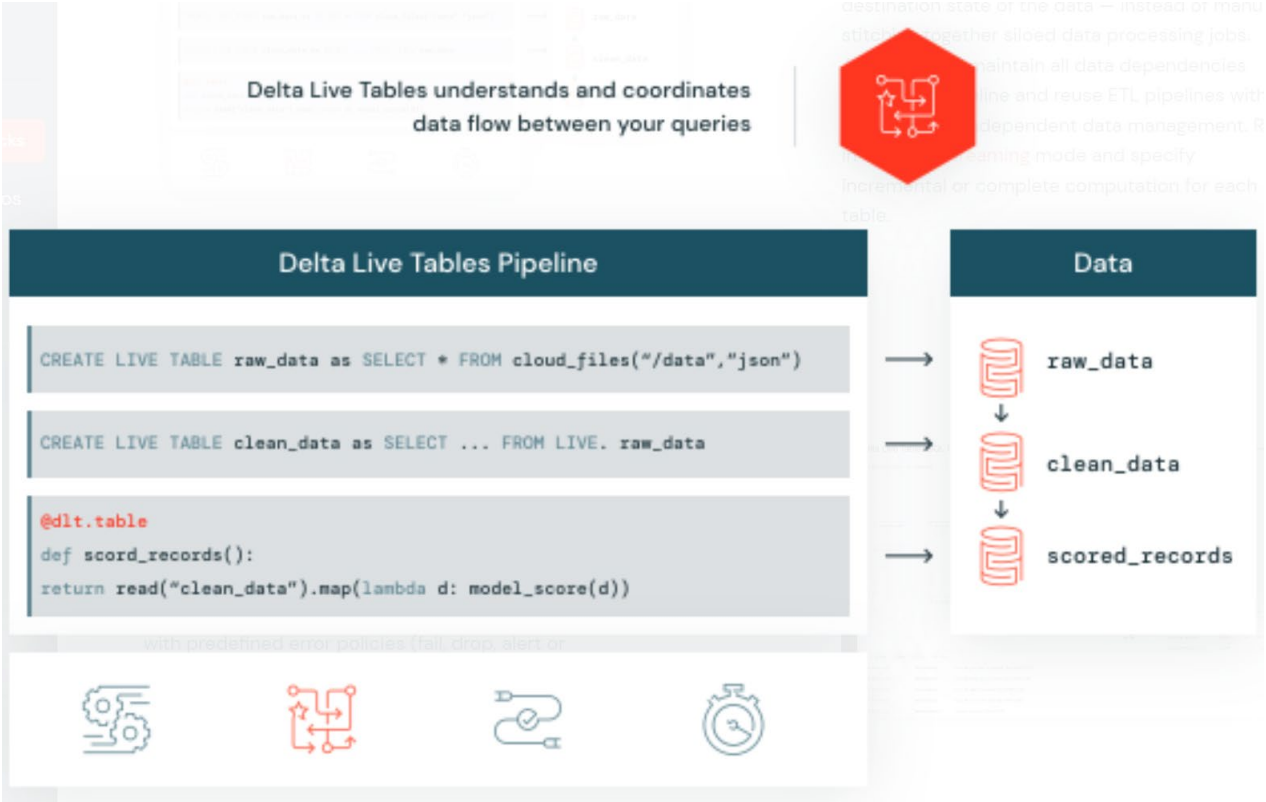
Option 1

- The combination of using both Batch and Streaming via native Spark gives you:
  - Native performance in unified platform
  - Least amount of storage silos and conversions of data
  - Common skillset of Spark API across Scala, Java, R & Python
  - Lowest cost
  - Simplest set of dependencies

Batch & Streaming – Databricks Managed

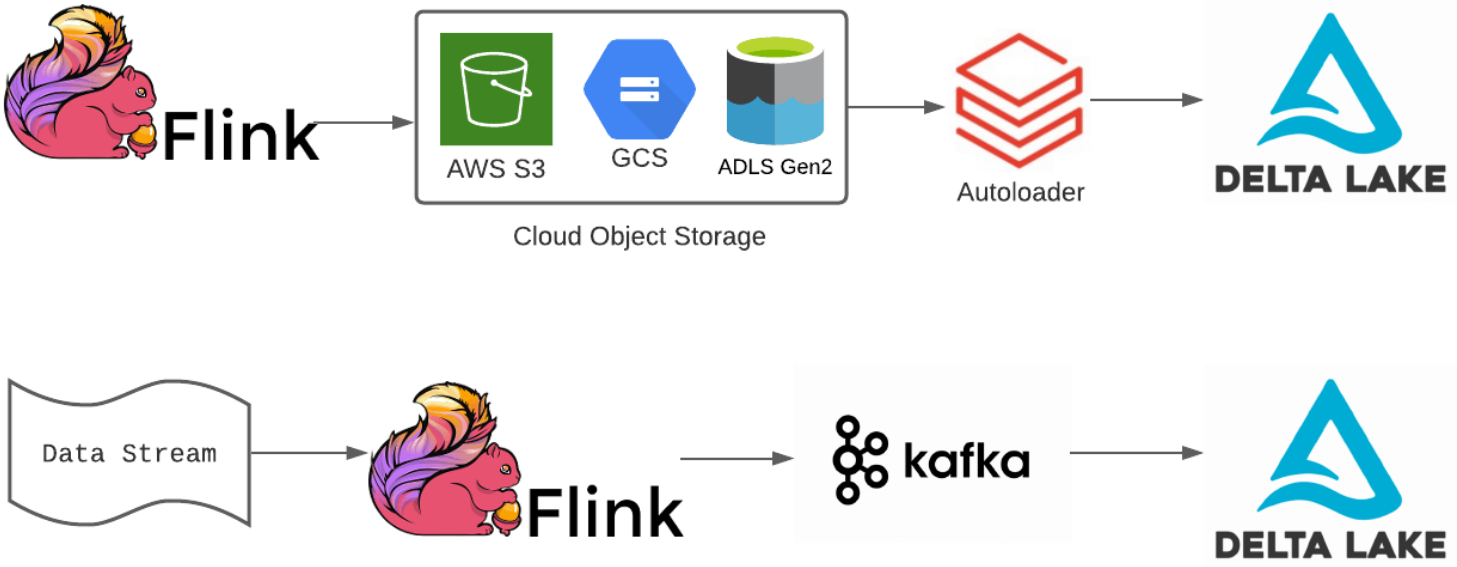
Option 2

- Delta Live Tables
  - Databricks SQL
  - Streaming and Batch seamlessly



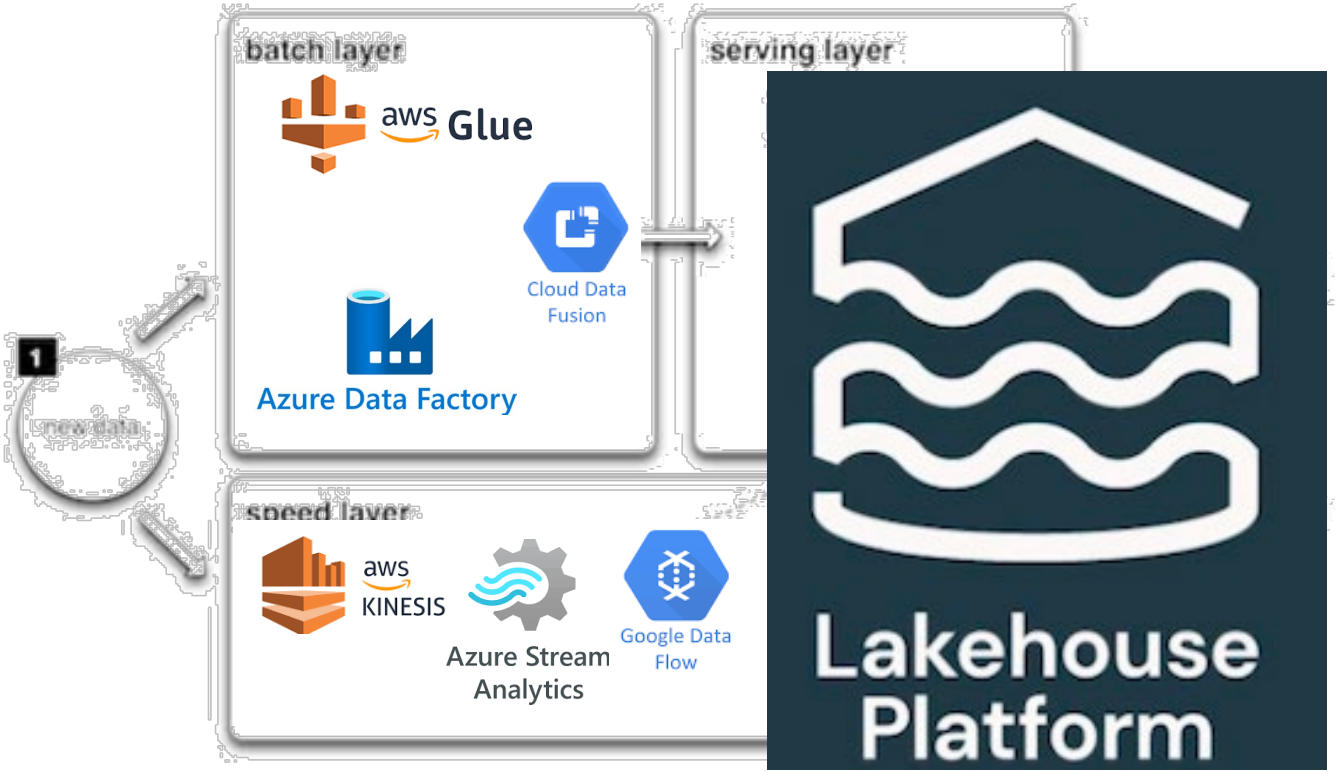
Batch & Streaming – Open Source

Option 3



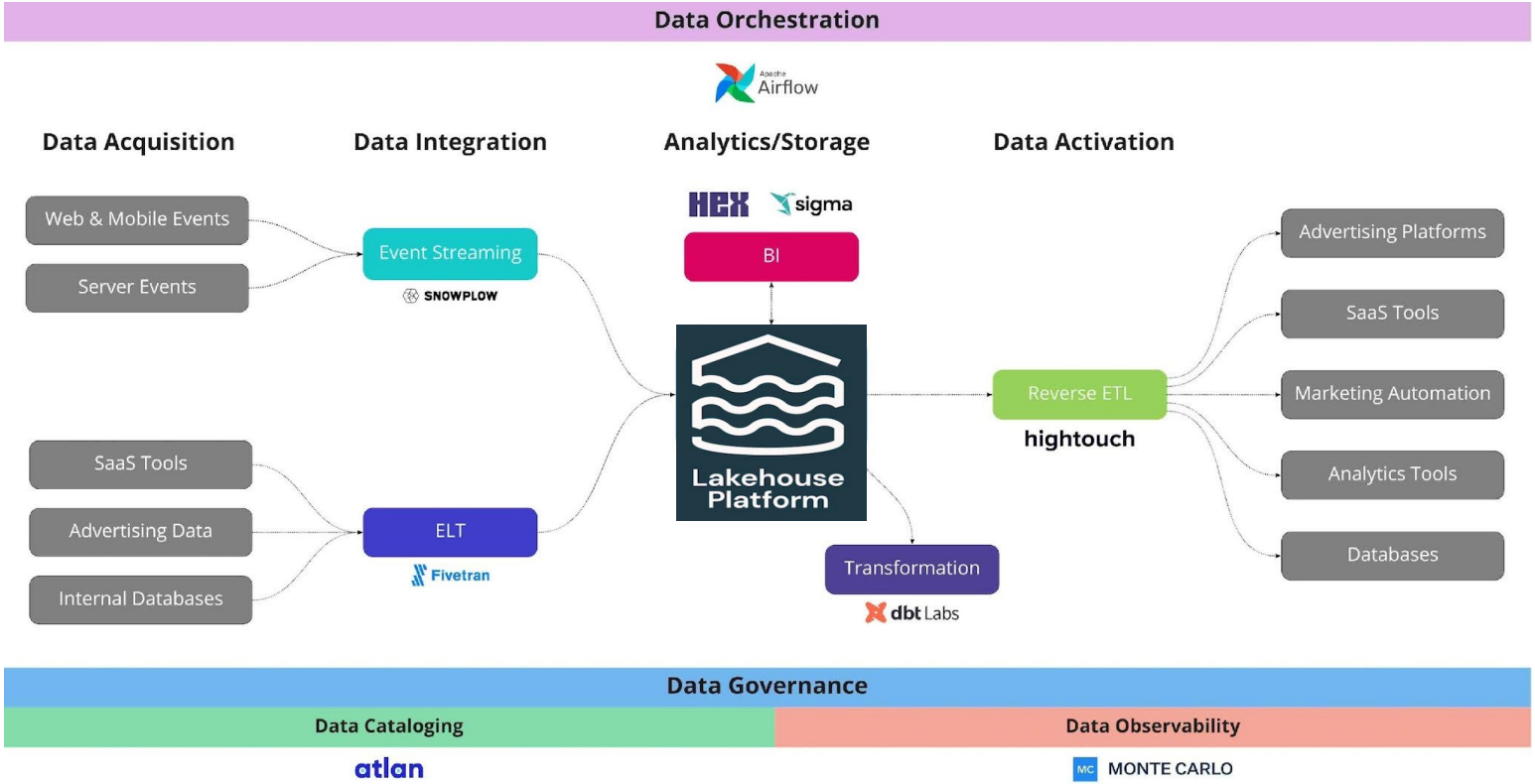
Batch & Streaming – Cloud Native

Option 4



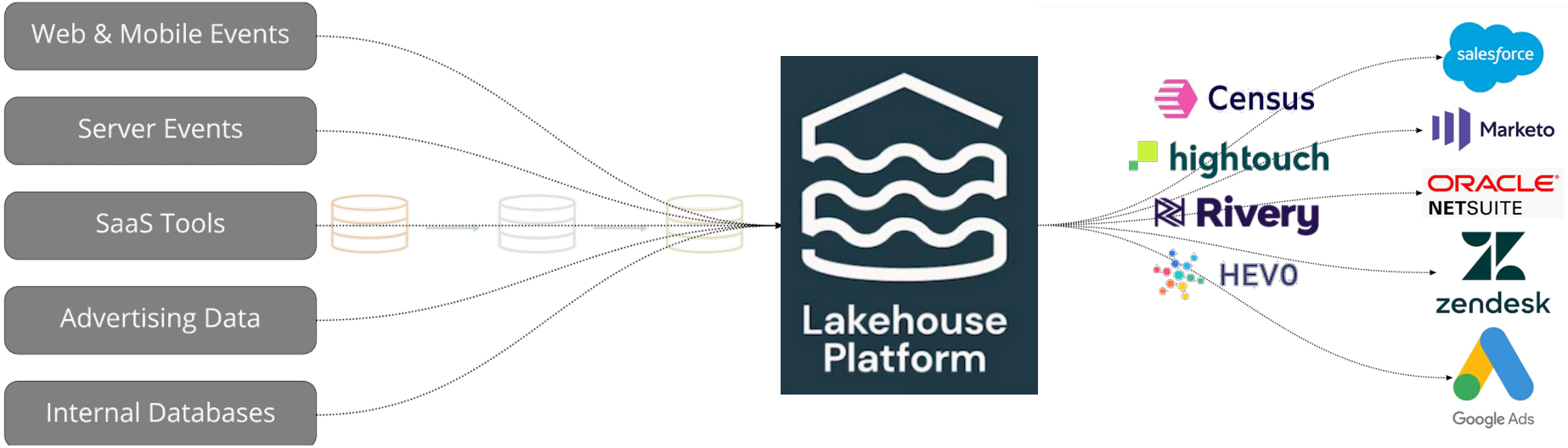
Batch & Streaming – Partner Tools

Option 5



Batch & Streaming – Partner Tools

Data Acquisition **Data Integration** **Analytics/Storage** **Data Activation**



The process of copying data from your central data warehouse to your operational tools, including but not limited to SaaS tools used for growth, marketing, sales, and support.

Batch & Streaming – ETL Evolution

Traditional ETL

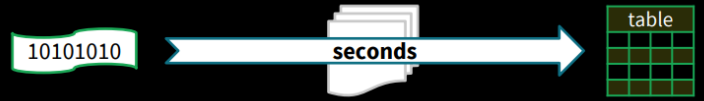
Raw, dirty, un/semi-structured is data dumped as files

Periodic jobs run every few hours to convert raw data to structured data ready for further analytics

Hours of delay before taking decisions on latest data

Unacceptable when time is of essence
[intrusion detection, anomaly detection, etc.]

Streaming ETL w/ Structured Streaming



Structured Streaming enables raw data to be available as structured data as soon as possible

Complexities in stream processing

Complex Data

Diverse data formats
(json, avro, binary, …)

Data can be dirty,
late, out-of-order

Complex Workloads

Event time processing

Combining streaming
with interactive queries,
machine learning

Complex Systems

Diverse storage
systems and formats
(SQL, NoSQL, parquet, …)

System failures

Structured Streaming

stream processing on Spark SQL engine
fast, scalable, fault-tolerant

rich, unified, high level APIs
deal with *complex data* and *complex workloads*

rich ecosystem of data sources
integrate with many *storage systems*

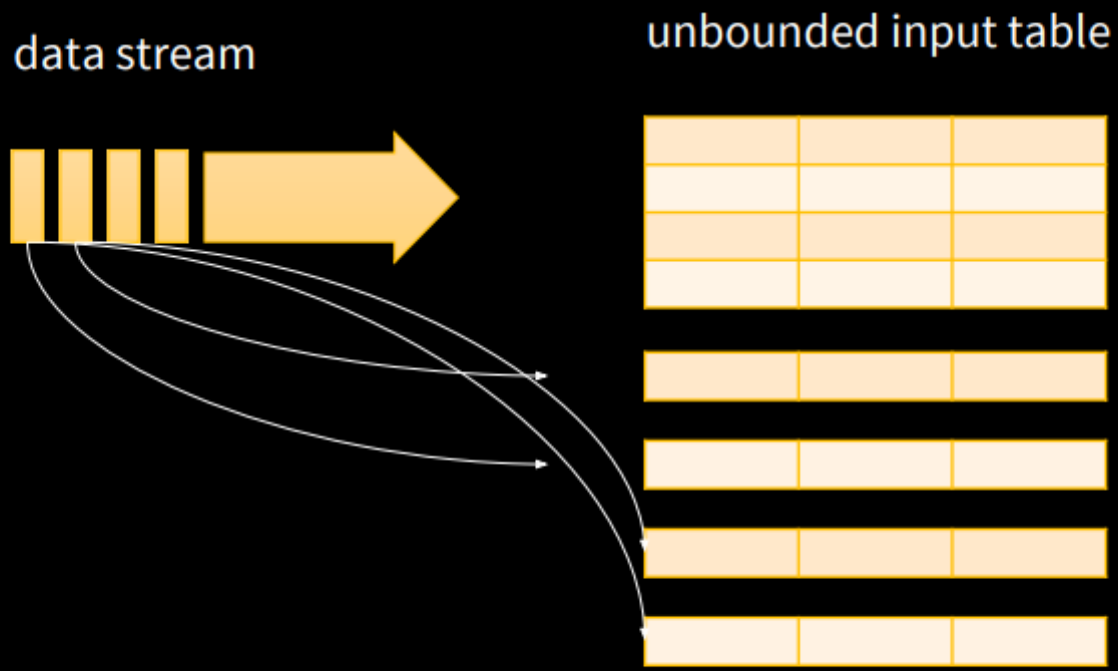
you
should not have to
reason about streaming

databricks

you
should write simple batch queries
&
Spark
should automatically *streamify*
them

databricks

Treat Streams as Unbounded Tables



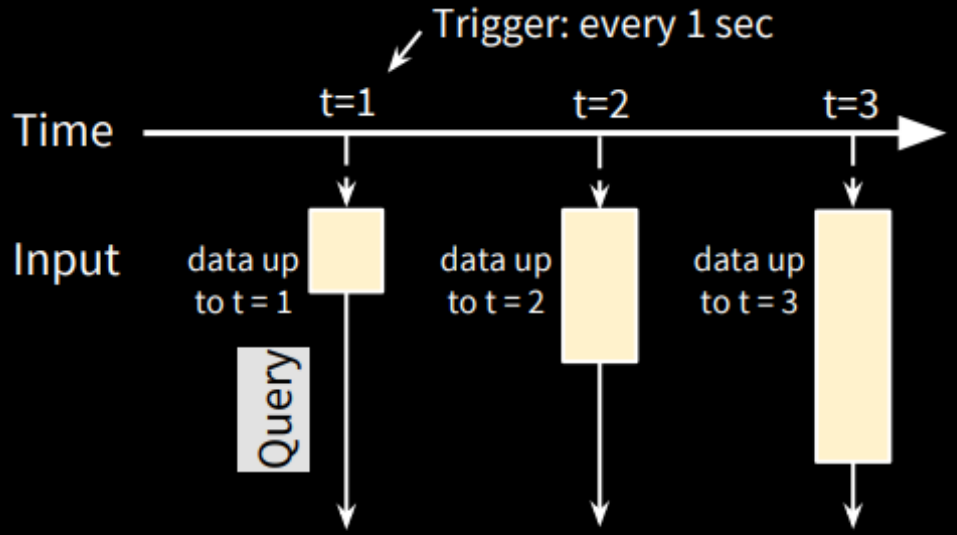
new data in the
data stream
=
new rows appended
to a unbounded table

New Model

Input: data from source as an append-only table

Trigger: how frequently to check input for new data

Query: operations on input
usual map/filter/reduce
new window, session ops



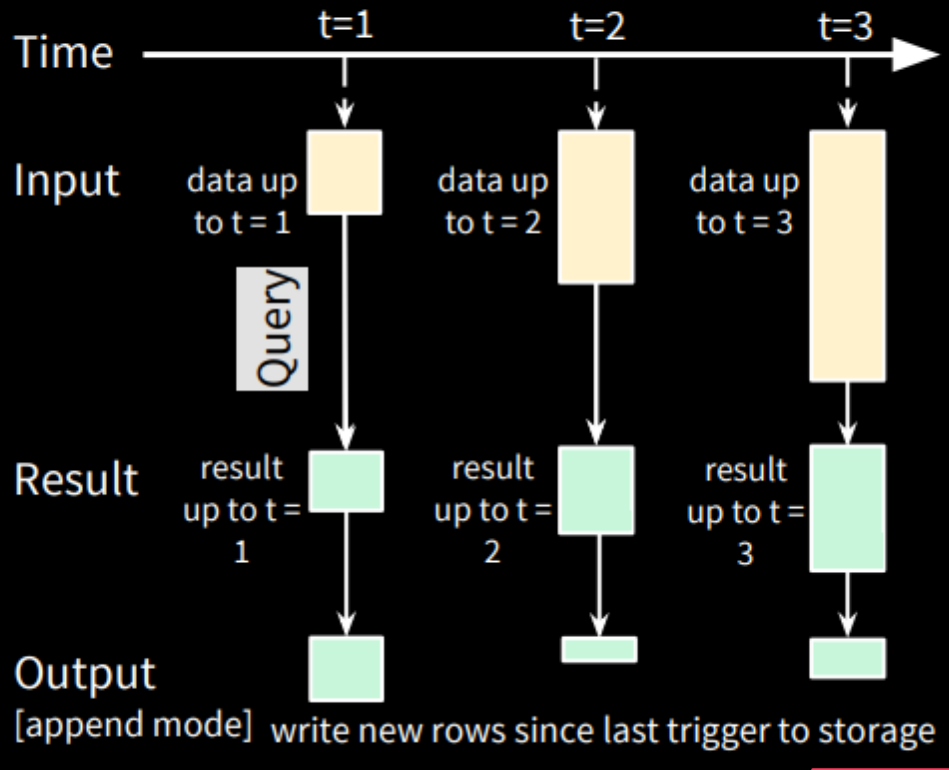
New Model

Result: final operated table
updated after every trigger

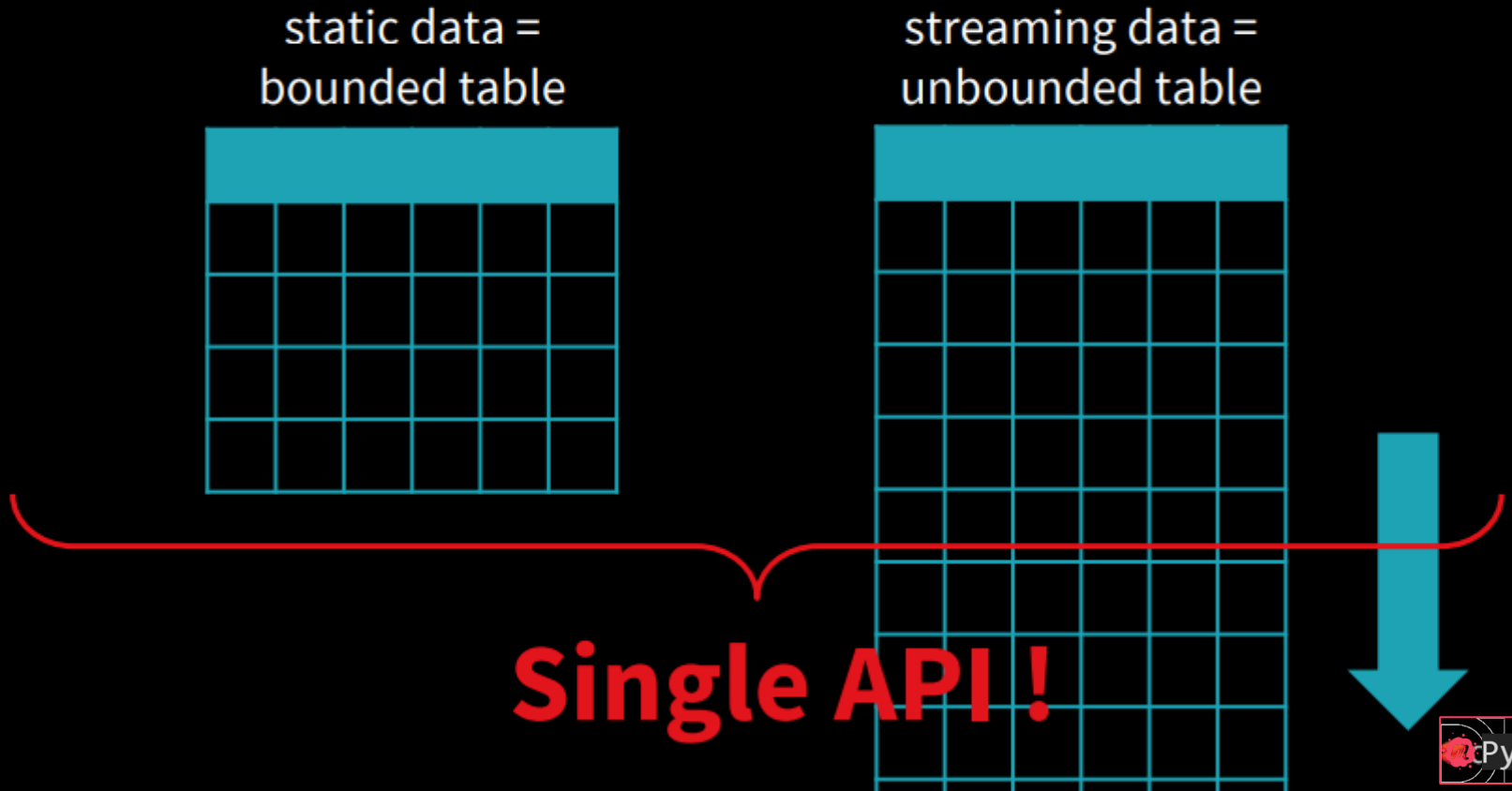
Output: what part of result to write
to storage after every trigger

Complete output: write full result table every time

Append output: write only new rows that got added to result table since previous batch



API - *Dataset/DataFrame*



Batch Queries with DataFrames

```
input = spark.read
    .format("json")
    .load("source-path")
```

Read from Json file

```
result = input
    .select("device", "signal")
    .where("signal > 15")
```

Select some devices

```
result.write
    .format("parquet")
    .save("dest-path")
```

Write to parquet file

databricks

Streaming Queries with DataFrames

```
input = spark.readStream
    .format("json")
    .load("source-path")
```

Read from Json file stream
Replace **read** with **readStream**

```
result = input
    .select("device", "signal")
    .where("signal > 15")
```

Select some devices
Code does not change

```
result.writeStream
    .format("parquet")
    .start("dest-path")
```

Write to Parquet file stream
Replace **save()** with **start()**

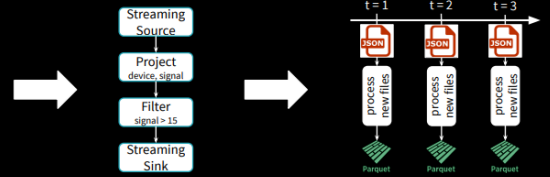
databricks

Spark automatically streamifies!

```
input = spark.readStream
    .format("json")
    .load("source-path")

result = input
    .select("device", "signal")
    .where("signal > 15")

result.writeStream
    .format("parquet")
    .start("dest-path")
```



DataFrames, Datasets, SQL

Logical Plan

Series of Incremental

Spark SQL converts batch-like query to a series of incremental execution plans operating on new batches of data

databricks

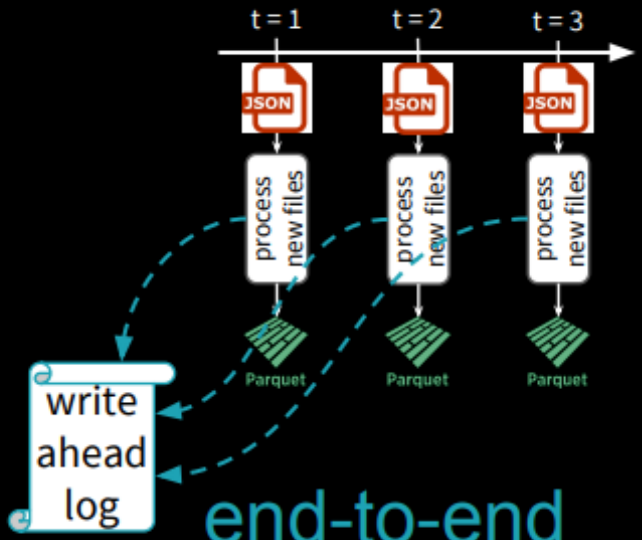
Fault-tolerance with Checkpointing

Checkpointing - metadata (e.g. offsets) of current batch stored in a *write ahead log* in HDFS/S3

Query can be restarted from the log

Streaming sources can replay the exact data range in case of failure

Streaming sinks can dedup reprocessed data when writing, idempotent by design



end-to-end exactly-once guarantees

Streaming ETL w/ Structured Streaming

Example

- Json data being received in Kafka
- Parse nested json and flatten it
- Store in structured Parquet table
- Get end-to-end failure guarantees

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

val parsedData = rawData
    .selectExpr("cast (value as string) as json"))
    .select(from_json("json").as("data"))
    .select("data.*")

val query = parsedData.writeStream
    .option("checkpointLocation", "/checkpoint")
    .partitionBy("date")
    .format("parquet")
```

Reading from Kafka [Spark 2.1]

Support Kafka 0.10.0.1
Specify options to configure

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

How?

```
kafka.boostrap.servers => broker1
```

What?

```
subscribe        =>  topic1,topic2,topic3   // fixed list of topics
subscribePattern =>  topic*         // dynamic list of topics
assign           =>  {"topicA":[0,1] }      // specific partitions
```

Where?

```
startingOffsets => latest(default) / earliest / {"topicA":{"0":23,"1":345} }
```

Reading from Kafka

rawData dataframe has the following columns

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

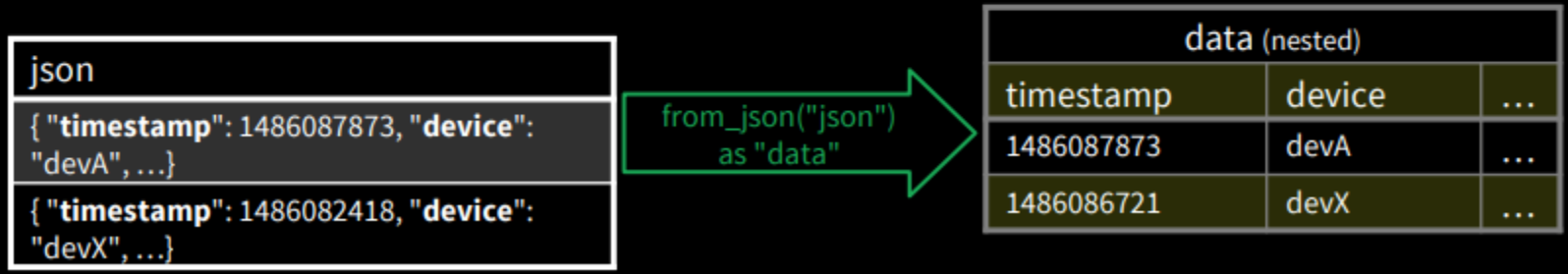
key	value	topic	partition	offset	timestamp
[binary]	*[binary]*	"topicA"	0	345	1486087873
[binary]	*[binary]*	"topicB"	3	2890	1486086721

Transforming Data

Cast binary *value* to string
Name it column *json*

```
val parsedData = rawData
    .selectExpr("cast (value as string) as json")
    .select(from_json("json").as("data"))
    .select("data.*")
```

Parse *json* string and expand into nested columns, name it *data*



Transforming Data

Cast binary *value* to string
Name it column *json*

Parse *json* string and expand into nested columns, name it data

Flatten the nested columns

```
val parsedData = rawData
    .selectExpr("cast (value as string) as json")
    .select(from_json("json").as("data"))
    .select("data.*")
```

powerful built-in APIs to perform complex data transformations

from_json, to_json, explode, ...
100s of functions

(see our blog post)

Writing to Parquet table

Save parsed data as Parquet table in the given path

Partition files by date so that future queries on time slices of data is fast

- e.g. query on last 48 hours of data

```
val query = parsedData.writeStream
    .option("checkpointLocation", ...)
    .partitionBy("date")
    .format("parquet")
    .start("/parquetTable")
```

Checkpointing

Enable checkpointing by setting the checkpoint location to save offset logs

`start` actually starts a continuous running StreamingQuery in the Spark cluster

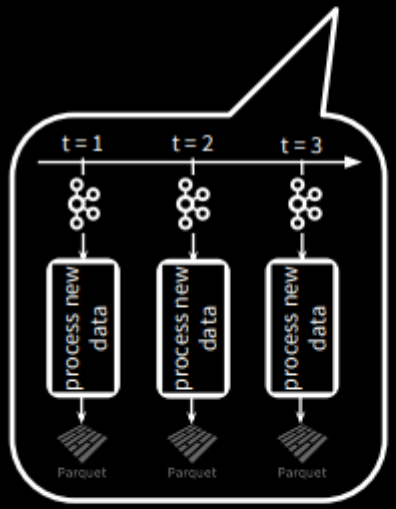
```
val query = parsedData.writeStream
    .option("checkpointLocation", ...)
    .format("parquet")
    .partitionBy("date")
    .start("/parquetTable/")
```

Streaming Query

kafka

StreamingQuery

Parquet




```
val query = parsedData.writeStream
  .option("checkpointLocation", ...)
  .format("parquet")
  .partitionBy("date")
  .start("/parquetTable/")
```

query is a handle to the continuously running StreamingQuery

Used to monitor and manage the execution

Data Consistency on Ad-hoc Queries

Data available for complex, ad-hoc analytics within seconds

Parquet table is updated atomically, ensures *prefix integrity*

Even if distributed, ad-hoc queries will see either all updates from streaming query or none, read more in our blog
https://databricks.com/blog/2016/07/28/structured-streaming-in-apache-spark.html

Demo, Demo, Demo!!!

THANK YOU

Code & PDF @ https://github.com/hekaplex/RTDDDMSS