What's New in Apache Spark 2.3?

Xiao Li

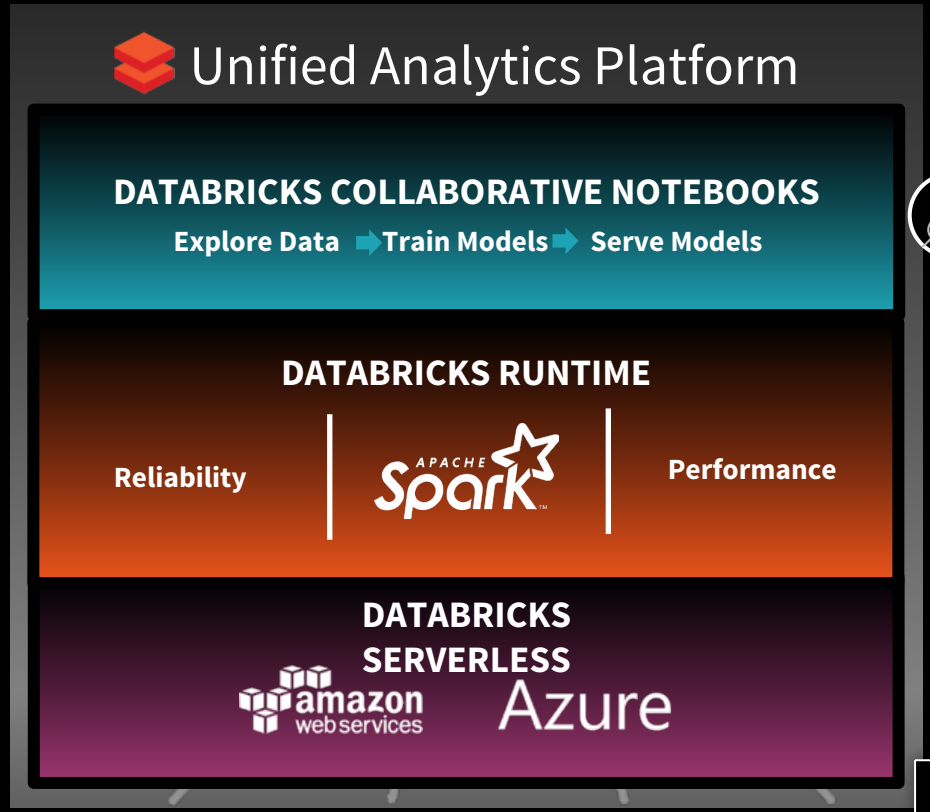
Feb 8, 2018

databricks

About Me

- Software Engineer at Databricks
- Apache Spark Committer and PMC Member
- Previously, IBM Master Inventor
- Spark SQL, Database Replication, Information Integration
- Ph.D. in University of Florida
- Github: gatorsmile

Simplify Big Data and AI with Databricks



Increases Data Science Productivity by 5x

Eliminates Disparate Tools with Optimized Spark

Removes Devops & Infrastructure Complexity

Open Extensible API's

+ableau looker

Accelerates & Simplifies Data Prep for Analytics

Databricks Enterprise Security

databricks

IoT / STREAMING DATA

CLOUD STORAGE

DATA WAREHOUSES

HADOOP STORAGE

Databricks Customers Across Industries

Financial Services

Healthcare & Pharma

Media & Entertainment

Data & Analytics Services

Technology

Public Sector

Retail & CPG

Consumer Services

Marketing & AdTech

Energy & Industrial IoT

This Talk

Databricks Runtime 4.0

Continuous Processing

Streaming ML + Image Reader

PySpark Performance

Spark on Kubernetes

Databricks Delta

OSS Apache Spark 2.3

Major Features on Spark 2.3

Continuous Processing

Data Source API V2

Spark on Kubernetes

PySpark Performance

ML on Streaming

History Server V2

Stream-stream Join

UDF Enhancements

Image Reader

Native ORC Support

Stable Codegen

Various SQL Features

Over 1300 issues resolved!

This Talk

Continuous Processing

Streaming ML + Image Reader

PySpark Performance

Spark on Kubernetes

Databricks Delta

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*

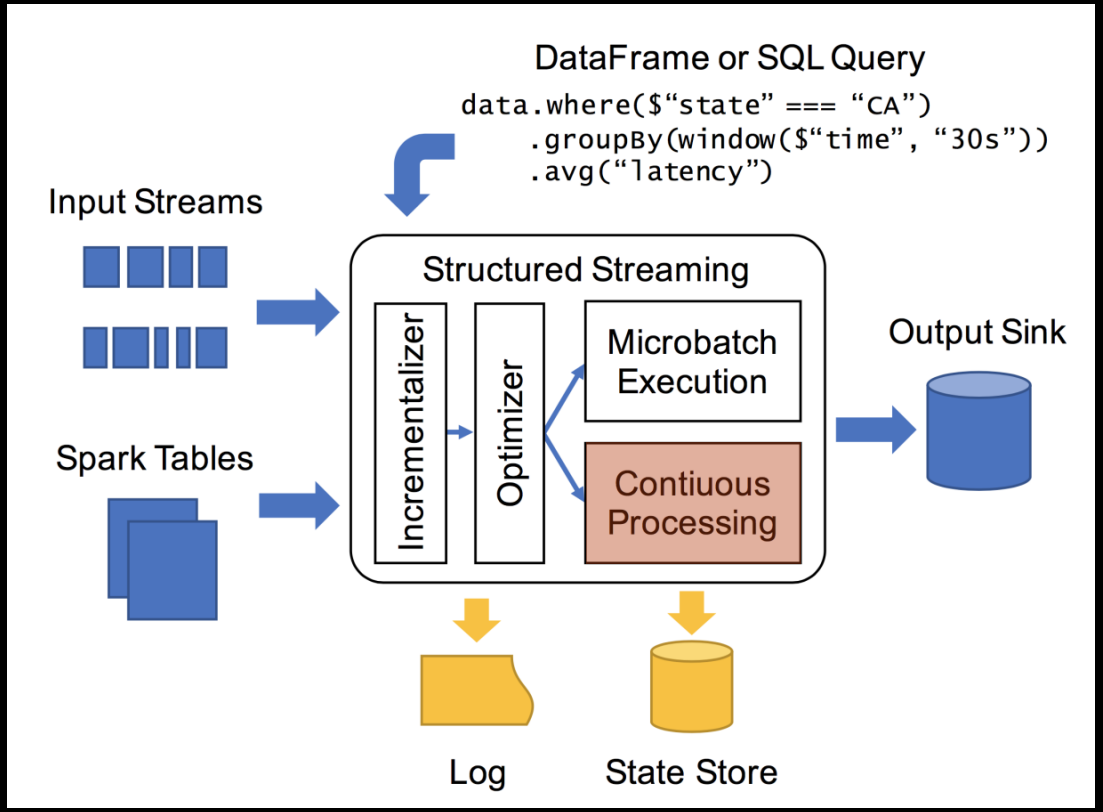
Structured Streaming

Introduced in Spark 2.0

Among Databricks customers:

- 10X more usage than DStream
- Processed 100+ trillion records in production

Structured Streaming



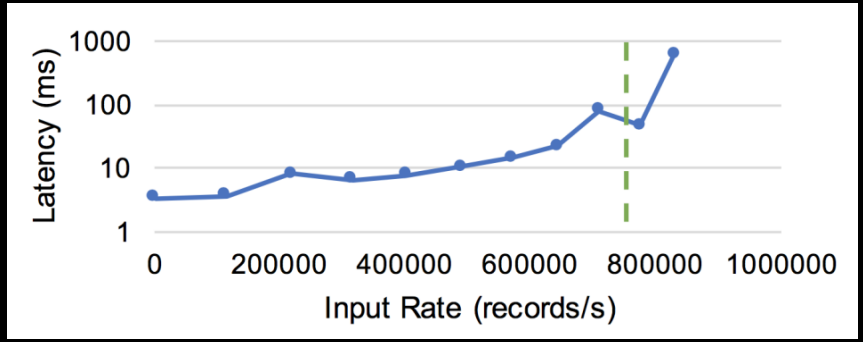
Continuous Processing Execution Mode

Micro-batch Processing (since 2.0 release)

- Lower end-to-end latencies of ~100ms
- Exactly-once fault-tolerance guarantees

Continuous Processing (since 2.3 release) [SPARK-20928]

- A new streaming execution mode (experimental)
- Low (~1 ms) end-to-end latency
- At-least-once guarantees



Continuous Processing

```
import org.apache.spark.sql.streaming.Trigger

spark
  .readStream
  .format("rate")
  .option("rowsPerSecond", "10")
  .option("")

spark
  .readStream
  .format("kafka")
  .option("kafka.bootstrap.servers", "host1:port1,host2:port2")
  .option("subscribe", "topic1")
  .load()
  .selectExpr("CAST(key AS STRING)", "CAST(value AS STRING)")
  .writeStream
  .format("kafka")
  .option("kafka.bootstrap.servers", "host1:port1,host2:port2")
  .option("topic", "topic1")
  .trigger(Trigger.Continuous("1 second"))
  .start()
```

The only change you need!

Continuous Processing

Continuous Processing

Supported Operations:

- Map-like Dataset operations
  - projections
  - selections
- All SQL functions
  - Except current_timestamp(), current_date() and aggregation functions

Supported Sources:

- Kafka source
- Rate source

Supported Sinks:

- Kafka sink
- Memory sink
- Console sink

This Talk

Continuous Processing

Streaming ML
+
Image Reader

PySpark Performance

Spark on Kubernetes

Databricks Delta

ML on Streaming

ML on Streaming

Model transformation/prediction on batch and streaming data with unified API.

After fitting a model or Pipeline, you can deploy it in a streaming job.

```
val streamOutput = transformer.transform(streamDF)
```

Demo

Notebook:

https://databricks-prod-cloudfront.cloud.databricks.com/public/4027ec902e239c93eaaa8714f173bcfc/1212468499849361/6491801434571 89/7151759854724023/latest.html

Image Support in Spark

Image Reader

Spark Image data source SPARK-21866 :

- Defined a standard API in Spark for reading images into DataFrames
- Deep learning frameworks can rely on this.

```
val df = ImageSchema.readImages("/data/images")
```

This Talk

Continuous Processing

Streaming ML + Image Reader

PySpark Performance

Spark on Kubernetes

Databricks Delta

PySpark

Introduced in Spark 0.7 (~2013); became first class citizen in the DataFrame API in Spark 1.3 (~2015)

Much slower than Scala/Java with user-defined functions (UDF), due to serialization & Python interpreter

Note: Most PyData tooling, e.g. Pandas, numpy, are written in C++

PySpark Performance

Fast data serialization and execution using vectorized formats [SPARK-22216] [SPARK-21187]

- Conversion from/to Pandas

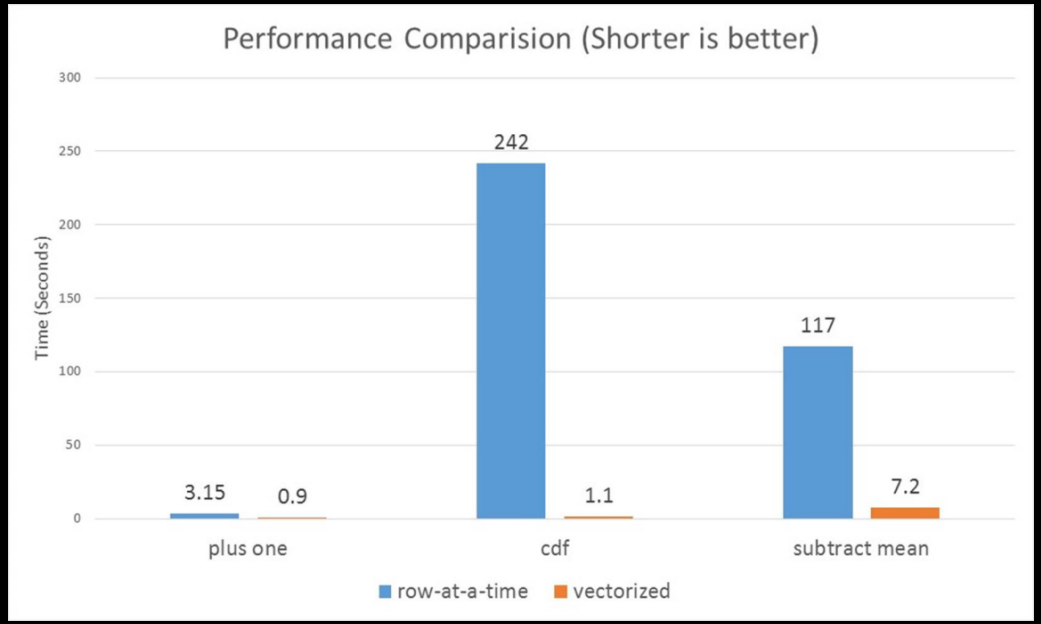
  df.*toPandas*()

  createDataFrame(pandas_df)

- Pandas/Vectorized UDFs: UDF using Pandas to process data
  - Scalar Pandas UDFs
  - Grouped Map Pandas UDFs

PySpark Performance

PySpark Performance



Blog "Introducing Vectorized UDFs for PySpark" http://dbricks.co/2rMwmW0

Pandas UDF

Scalar Pandas UDFs:

- Used with functions such as *select* and *withColumn*.
- The Python function should take *pandas.Series* as inputs and return a *pandas.Series* of the same length.

```
import pandas as pd
from pyspark.sql.functions import col, pandas_udf
from pyspark.sql.types import LongType

x = pd.Series([1, 2, 3])
# Create a Spark DataFrame, 'spark' is an existing SparkSession
df = spark.createDataFrame(pd.DataFrame(x, columns=["x"]))

# Declare the function and create the UDF
def multiply_func(a, b):
    return a * b
multiply = pandas_udf(multiply_func, returnType=LongType())

df.select(multiply(col("x"), col("x"))).show()
```

Pandas UDF

Grouped Map Pandas UDFs:

- **Split-apply-combine**
- A Python function that defines the computation for each group.
- The output schema

```
from pyspark.sql.functions import pandas_udf, PandasUDFType

df = spark.createDataFrame(
    [(1, 1.0), (1, 2.0), (2, 3.0), (2, 5.0), (2, 10.0)],
    ("id", "v"))

@pandas_udf("id long, v double", PandasUDFType.GROUPED_MAP)
def substract_mean(pdf):
    # pdf is a pandas.DataFrame
    v = pdf.v
    return pdf.assign(v=v - v.mean())

df.groupby("id").apply(substract_mean).show()
```

Demo

Notebook:

https://databricks-prod-cloudfront.cloud.databricks.com/public/4027ec902e239c93eaaa8714f173bcfc/1212468499849361/4219142393863 10/7151759854724023/latest.html

This Talk

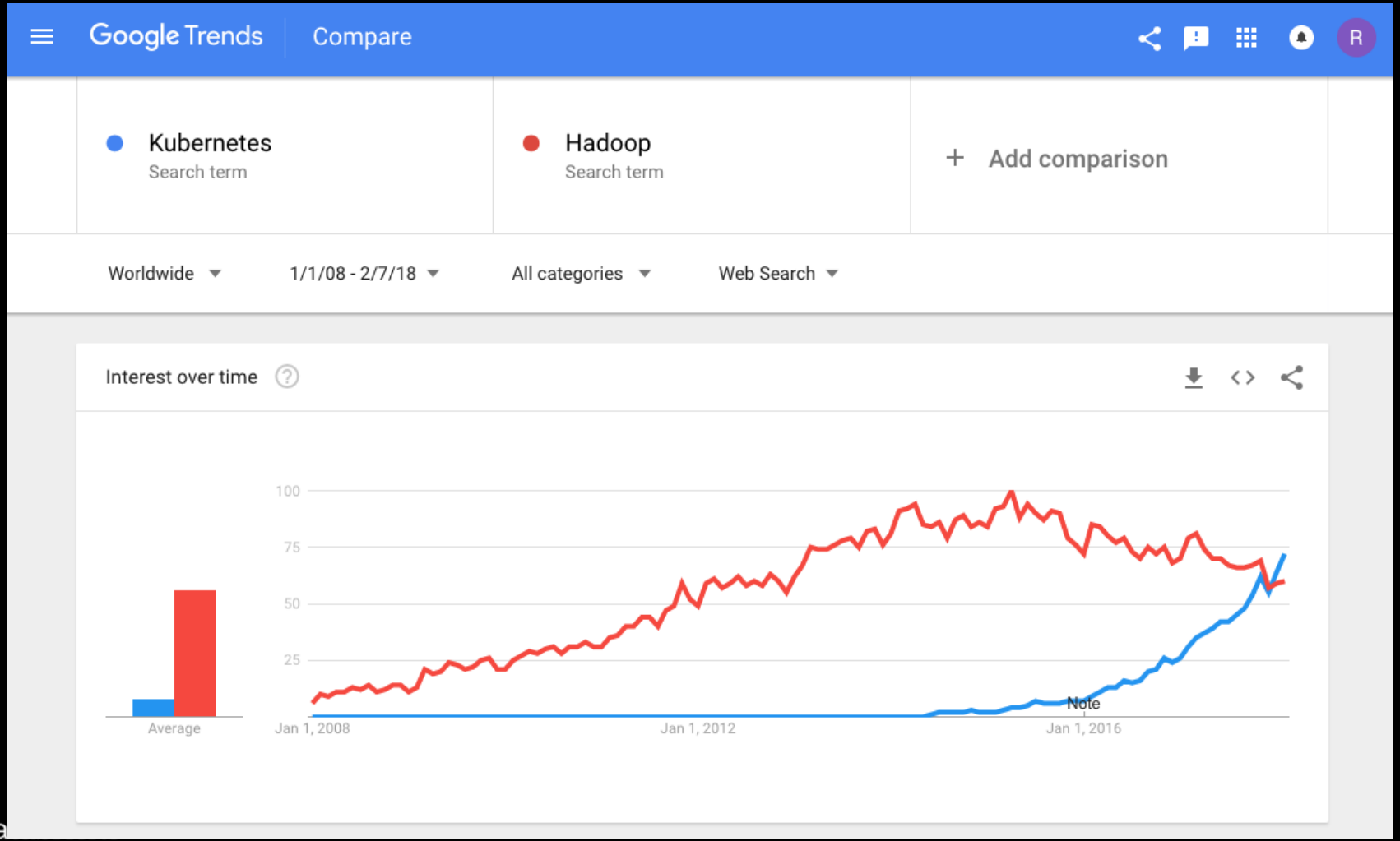
Continuous Processing

Streaming ML + Image Reader

PySpark Performance

Spark on Kubernetes

Databricks Delta


Google Trends
Compare
R
Kubernetes
Search term
Hadoop
Search term
+ Add comparison
Worldwide
1/1/08 - 2/7/18
All categories
Web Search
Interest over time
100
75
50
25
Average
Jan 1, 2008
Jan 1, 2012
Jan 1, 2016
Note

Native Spark App in K8S

- New Spark scheduler backend
- Driver runs in a Kubernetes pod created by the submission client and creates pods that run the executors in response to requests from the Spark scheduler. [K8S-34377] [SPARK-18278]
- Make direct use of Kubernetes clusters for multi-tenancy and sharing through Namespaces and Quotas, as well as administrative features such as Pluggable Authorization, and Logging.

Spark on Kubernetes

Supported:

- Supports Kubernetes 1.6 and up
- Supports cluster mode only
- Static resource allocation only
- Supports Java and Scala applications
- Can use container-local and remote dependencies that are downloadable

In roadmap (2.4):

- Client mode
- Dynamic resource allocation + external shuffle service
- Python and R support
- Submission client local dependencies + Resource staging server (RSS)
- Non-secured and Kerberized HDFS access (injection of Hadoop configuration)

This Talk

Continuous Processing

Streaming ML
+
Image Reader

PySpark Performance

Spark on Kubernetes

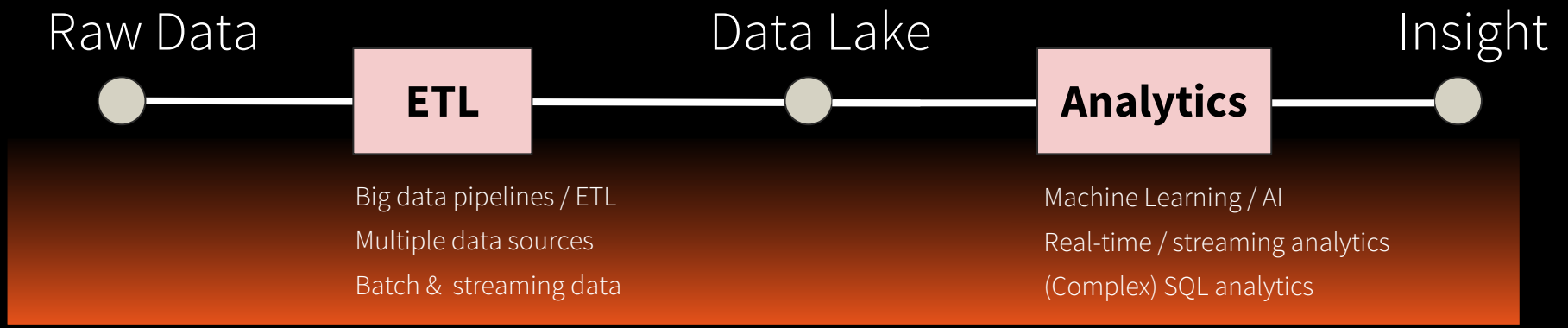
Databricks Delta

What is Databricks Delta?

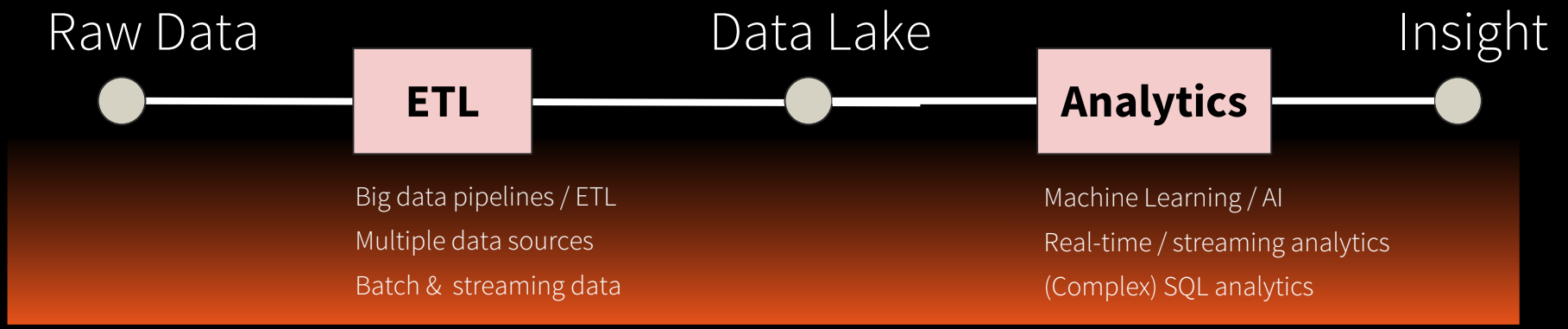
Delta is a data management capability that brings data reliability and performance optimizations to the cloud data lake.

Key Capabilities - Transactions & Indexing

Standard data pipelines with Spark



Yet challenges still remain



Reliability and Complexity

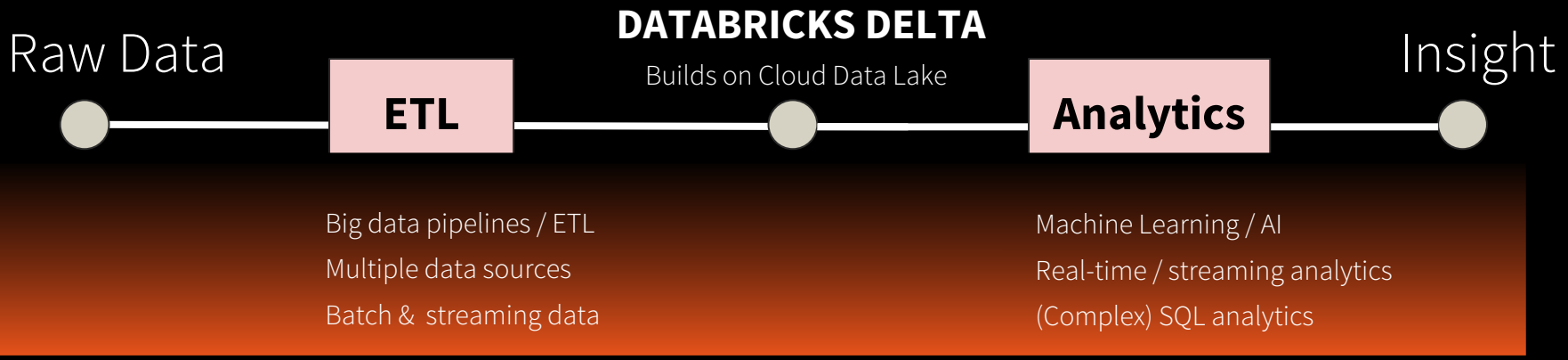
- Data corruption issues and broken pipelines
- Complex workarounds - tedious scheduling and multiple jobs/staging tables
- Many use cases require updates to existing data - not supported by Spark / Data lakes

Reliability & Performance Problems

- Performance degradation at scale for advanced analytics
- Stale and unreliable data slows analytic decisions

Streaming magnifies these challenges

Databricks Delta address these challenges



Reliability & Automation

Transactions guarantees eliminates complexity

Schema enforcement to ensure clean data

Upserts/Updates/Deletes to manage data changes

Seamlessly support streaming and batch

Performance & Reliability

Automatic indexing & caching

Fresh data for advanced analytics

Automated performance tuning

Databricks Delta Under the Hood

SCALE

- Use cloud object storage (e.g. AWS S3)
- Decouple compute & storage

RELIABILITY

- ACID compliant & data upserts
- Schema evolution

PERFORMANCE

- Intelligent partitioning & compaction
- Data skipping & indexing

LOW LATENCY

- Real-time streaming ingest
- Structured streaming

Continuous Processing

Streaming ML + Image Reader

PySpark Performance

Spark on Kubernetes

Databricks Delta

Try these on Databricks Runtime 4.0 beta today!

Community edition:
https://community.cloud.databricks.com/

Thank you

Major Features on Spark 2.3

Continuous Processing

Data Source API V2

Spark on Kubernetes

PySpark Performance

ML on Streaming

History Server V2

Stream-stream Join

UDF Enhancements

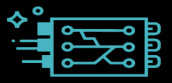
Image Reader

Native ORC Support

Stable Codegen

Various SQL Features

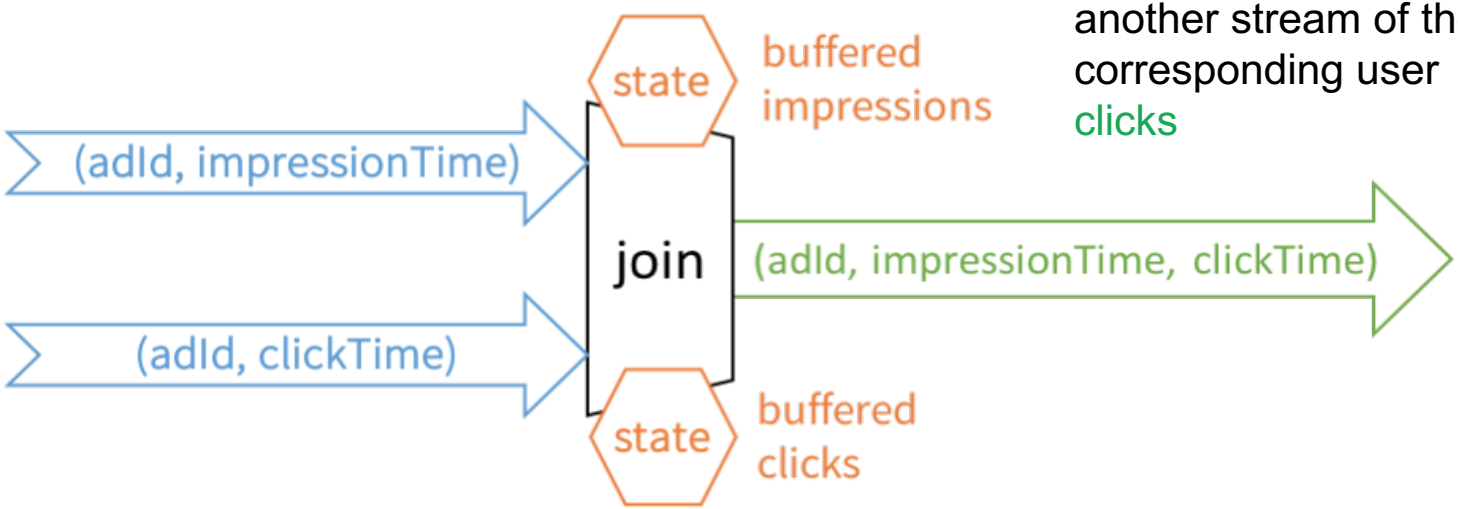
Stream-stream Joins



Stream-stream Join

Example: Ad Monetization

Join stream of ad impressions with another stream of their corresponding user clicks



Inner Join + Time constraints + Watermarks

Stream-stream Join

Time constraints

- Impressions can be 2 hours late
- Clicks can be 3 hours late
- A click can occur within 1 hour after the corresponding impression

```
val impressionsWithWatermark = impressions
      .withWatermark("impressionTime", "2 hours")

val clicksWithWatermark = clicks
      .withWatermark("clickTime", "3 hours")
```

Range Join

```
impressionsWithWatermark.join(
  clicksWithWatermark,
  expr("""
    clickAdId = impressionAdId AND
    clickTime >= impressionTime AND
    clickTime <= impressionTime + interval 1 hour
    """
  ))
```

Stream-stream Joins

Stream-stream Join

Inner	Supported, optionally specify watermark on both sides + time constraints for state cleanup
Left	Conditionally supported, must specify watermark on right + time constraints for correct results, optionally specify watermark on left for all state cleanup
Right	Conditionally supported, must specify watermark on left + time constraints for correct results, optionally specify watermark on right for all state cleanup
Full	Not supported.

Major Features on Spark 2.3

Continuous Processing

Data Source API V2

Spark on Kubernetes

PySpark Performance

ML on Streaming

History Server V2

Stream-stream Join

UDF Enhancements

Image Reader

Native ORC Support

Stable Codegen

Various SQL Features

What's Wrong With V1?

- **Leak** upper level API in the data source (DataFrame/SQLContext)
- **Hard** to extend the Data Source API for more optimizations
- **Zero** transaction guarantee in the write APIs

Design Goals

- Java friendly (written in Java).
- No dependency on upper level APIs (DataFrame/RDD/…).
- Easy to extend, can add new optimizations while keeping backward compatibility.
- Can report physical information like size, partition, etc.
- Streaming source/sink support.
- A flexible and powerful, transactional write API.
- No change to end users.

Features in Spark 2.3

Data Source API V2

- Support both row-based scan and columnar scan.
- Column pruning and filter pushdown.
- Can report basic statistics and data partitioning.
- Transactional write API.
- Streaming source and sink support for both micro-batch and continuous mode.
- Spark 2.4 + : more enhancements

Major Features on Spark 2.3

Continuous Processing

Data Source API V2

Spark on Kubernetes

PySpark Performance

ML on Streaming

History Server V2

Stream-stream Join

UDF Enhancements

Image Reader

Native ORC Support

Stable Codegen

Various SQL Features

UDF Enhancement

UDF Enhancements

- [SPARK-19285] Implement UDF0 (SQL UDF that has 0 arguments)
- [SPARK-22945] Add java UDF APIs in the functions object
- [SPARK-21499] Support creating SQL function for Spark UDAF(UserDefinedAggregateFunction)
- [SPARK-20586][SPARK-20416][SPARK-20668] Annotate UDF with Name, Nullability and Determinism

```
val foo = udf(() => Math.random()).asNondeterministic()
testData.select(foo())
```

Java UDF and UDAF

UDF
Enhancements

- Register Java UDF and UDAF as a SQL function and use them in PySpark.

```
# class MyDoubleAvg extends UserDefinedAggregateFunction
spark.udf.registerJavaUDAF("javaUDAF", "test.org.apache.spark.sql.MyDoubleAvg")
df = spark.createDataFrame([(1, "a"),(2, "b"), (3, "a")],["id", "name"])
df.registerTempTable("df")
spark.sql("SELECT name, javaUDAF(id) as avg from df group by name").collect()

# class JavaStringLength implements UDF1<String, Integer>
spark.udf.registerJavaFunction(
    "javaStringLength", "test.org.apache.spark.sql.JavaStringLength", "integer")
spark.sql("SELECT javaStringLength('test')").collect()
```

Major Features on Spark 2.3

Continuous Processing

Data Source API V2

Spark on Kubernetes

PySpark Performance

ML on Streaming

History Server V2

Stream-stream Join

UDF Enhancements

Image Reader

Native ORC Support

Stable Codegen

Various SQL Features

Stable Codegen

Stable Codegen

- [SPARK-22510] [SPARK-22692] Stabilize the codegen framework to avoid hitting the 64KB JVM bytecode limit on the Java method and Java compiler constant pool limit.
- [SPARK-21871] Turn off whole-stage codegen when the bytecode of the generated Java function is larger than **spark.sql.codegen.hugeMethodLimit**. The limit of method bytecode for JIT optimization on HotSpot is 8K.

Major Features on Spark 2.3

Continuous Processing

Data Source API V2

Spark on Kubernetes

PySpark Performance

ML on Streaming

History Server V2

Stream-stream Join

UDF Enhancements

Image Reader

Native ORC Support

Stable Codegen

Various SQL Features

Vectorized ORC Reader

- [SPARK-20682] Add new ORCFileFormat based on ORC 1.4.1. **spark.sql.orc.impl** = native (default) / hive (Hive 1.2.1)
- [SPARK-16060] Vectorized ORC reader **spark.sql.orc.enableVectorizedReader** = true (default) / false
- Enable filter pushdown for ORC files by default

Major Features on Spark 2.3

Continuous Processing

Data Source API V2

Spark on Kubernetes

PySpark Performance

ML on Streaming

History Server V2

Stream-stream Join

UDF Enhancements

Image Reader

Native ORC Support

Stable Codegen

Various SQL Features

Performance

- [SPARK-21975] Histogram support in cost-based optimizer
- Enhancements in rule-based optimizer and planner
  [SPARK-22489] [SPARK-22916] [SPARK-22895] [SPARK-20758] [SPARK-22266] [SPARK-19122] [SPARK-22662] [SPARK-21652] (e.g., constant propagation)
- [SPARK-20331] Broaden support for Hive partition pruning predicate pushdown. (e.g. date = 20161011 or date = 20161014)
- [SPARK-20822] Vectorized reader for table cache

API

- Improved ANSI SQL compliance and Dataset/DataFrame APIs
- More built-in functions [SPARK-20746]
- Better Hive compatibility
  - [SPARK-20236] Support Dynamic Partition Overwrite
  - [SPARK-17729] Enable Creating Hive Bucketed Tables
  - [SPARK-4131] Support INSERT OVERWRITE DIRECTORY

Major Features on Spark 2.3

Continuous Processing

Data Source API V2

Spark on Kubernetes

PySpark Performance

ML on Streaming

History Server V2

Stream-stream Join

UDF Enhancements

Image Reader

Native ORC Support

Stable Codegen

Various SQL Features

History Server Using K-V Store

Stateless and non-scalable History Server V1:

- Requires parsing the event logs (that means, so slow!)
- Requires storing app lists and UI in the memory (and then OOM!)

[SPARK-18085] **K-V store-based History Server V2:**

- Store app lists and UI in a persistent K-V store (LevelDB)
  - **spark.history.store.path** – once specified, LevelDB is being used; otherwise, in-memory KV-store (still stateless like V1)
  - V2 still can read the event log wrote by V1

Thank you