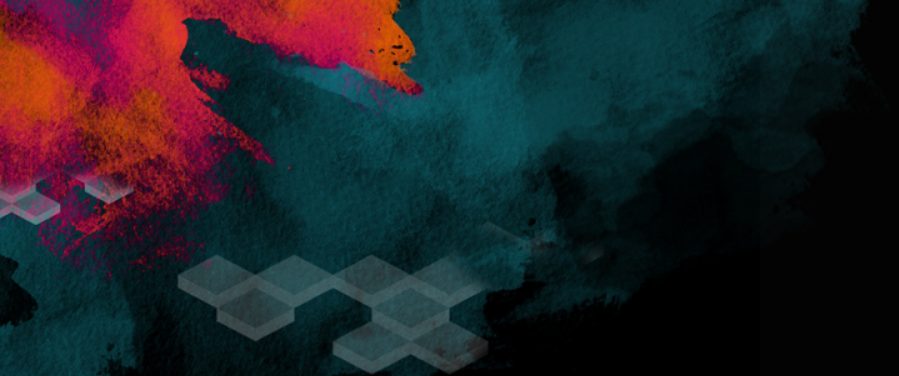
Unifying Big Data Workloads in Apache Spark

Hossein Falaki
@mhfalaki

Outline

- What’s Apache Spark
- Why Unification
- Evolution of Unification
- Apache Spark + Databricks
- Q & A

What’s Apache Spark



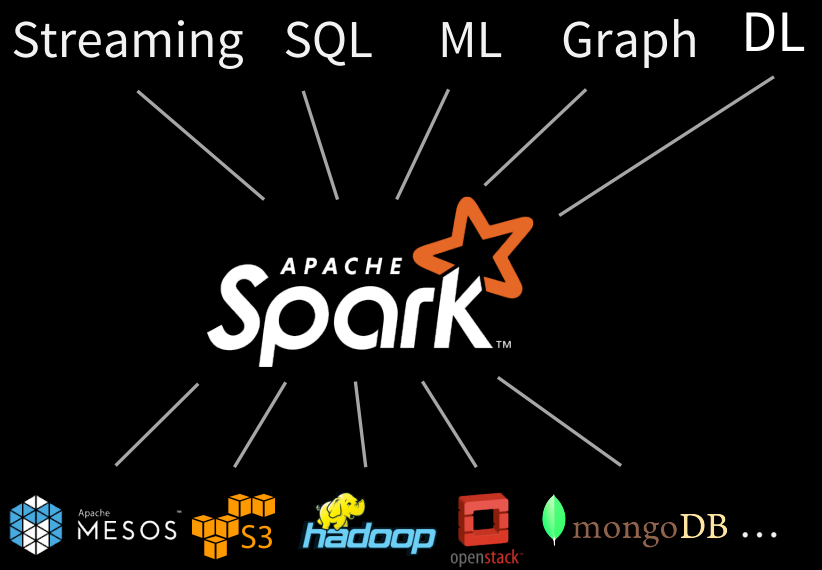
databricks

What is Apache Spark?

General cluster computing engine that extends MapReduce

Rich set of APIs and libraries

Large community: 1000+ orgs, clusters up to 8000 nodes



Why Unification?

Unique Thing about Spark

Unification: same engine and same API for diverse use cases

- Streaming, batch, or interactive
- ETL, SQL, machine learning, or graph
- Scala, Python, R, and SQL

This talk: Why unification? And how has it evolved in Spark?

Why Unification?

MapReduce: Simplified Data Processing on Large Clusters

Jeffrey Dean and Sanjay Ghemawat

jeff@google.com, sanjay@google.com

Google, Inc.

Abstract

MapReduce is a programming model and an associated implementation for processing and generating large data sets. Users specify a *map* function that processes a key/value pair to generate a set of intermediate key/value pairs, and a *reduce* function that merges all intermediate values associated with the same intermediate key. Many

given day, etc. Most such computations are conceptually straightforward. However, the input data is usually large and the computations have to be distributed across hundreds or thousands of machines in order to finish in a reasonable amount of time. The issues of how to parallelize the computation, distribute the data, and handle failures conspire to obscure the original simple computation with large amounts of complex code to deal with

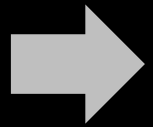
Why Unification?

MapReduce: a general engine for batch processing

> We wrote the first version of the MapReduce library in February of 2003, and made significant enhancements to it in August of 2003, including the locality optimization, dynamic load balancing of task execution across worker machines, etc. Since that time, we have been pleasantly surprised at how broadly applicable the MapReduce library has been for the kinds of problems we work on. It has been used across a wide range of domains within Google, including:

Big Data Systems Today

MapReduce



Pregel Giraph

Dremel Millwheel

Storm Impala

Drill S4 . . .

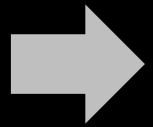
General batch processing

Specialized systems for new workloads

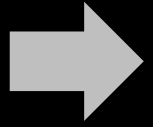
Hard to *combine* in pipelines

Big Data Systems Today

MapReduce



Pregel Giraph
Dremel Millwheel
Storm Impala
Drill S4 . . .



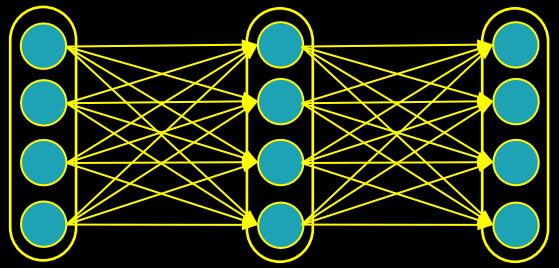
General batch processing

Specialized systems for new workloads

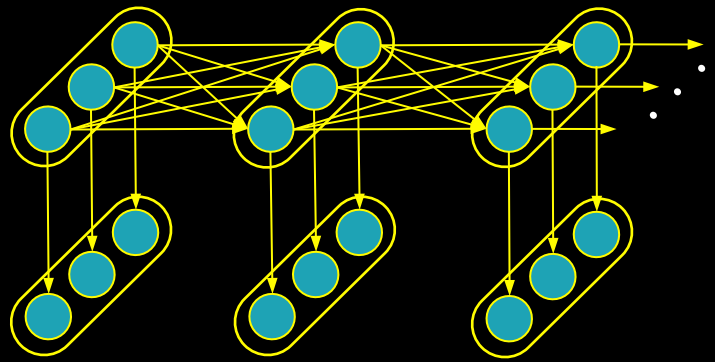
Unified engine

Key Idea

MapReduce + data sharing (RDDs) captures most distributed apps



Iterative



Streaming

Benefits of Unification

1. Simpler to use and operate

2. Code reuse: e.g. only write monitoring, FT, etc once

3. New apps that span processing types: e.g. interactive queries on a stream, online machine learning

An Analogy

Specialized devices

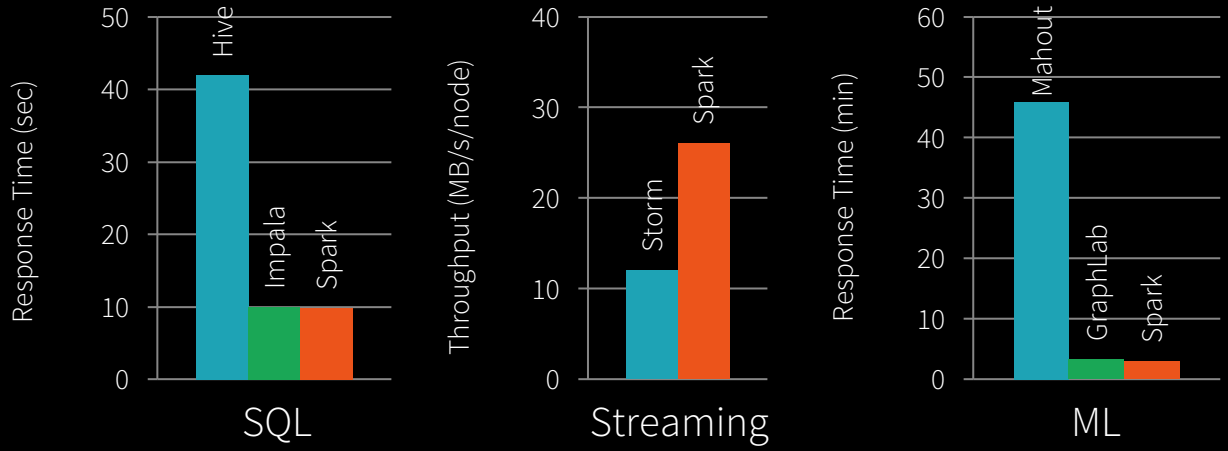
New applications

Unified device

databricks

How Well Does It Work?

- 75% of users use multiple Spark components
- Competitive performance on diverse workloads



Evolution of Unification

databricks

Challenges

Spark's original "unifying abstraction" was RDDs: DAG of map/reduce steps given as Java functions on Java objects

Two challenges:

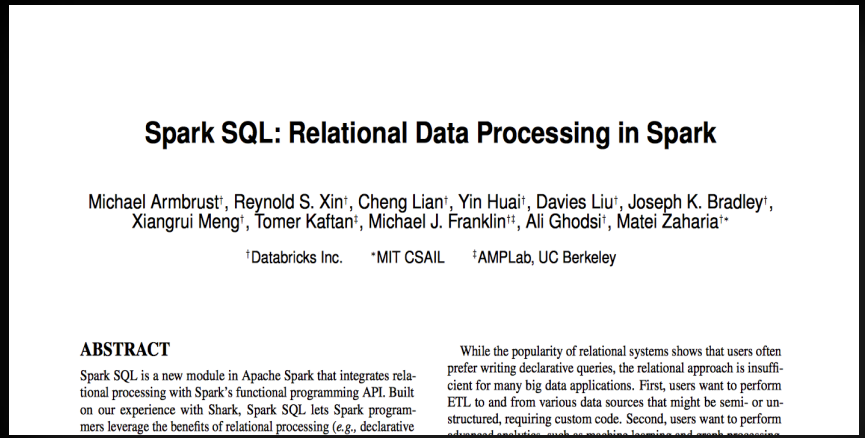
1. Richness of optimizations: hard to understand the computation
2. Changing hardware limits: I/O-bound → compute-bound

Solution: Structured APIs

New APIs that act on structured data (known schema) and make semantics of computation more visible (relational ops)

- Spark SQL engine + DataFrames + Datasets

Enable rich optimizations similar to databases and compilers



Spark SQL: Relational Data Processing in Spark

Michael Armbrust†, Reynold S. Xin†, Cheng Lian†, Yin Huai†, Davies Liu†, Joseph K. Bradley†, Xiangrui Meng†, Tomer Kaftan‡, Michael J. Franklin†‡, Ali Ghodsi†, Matei Zaharia†*

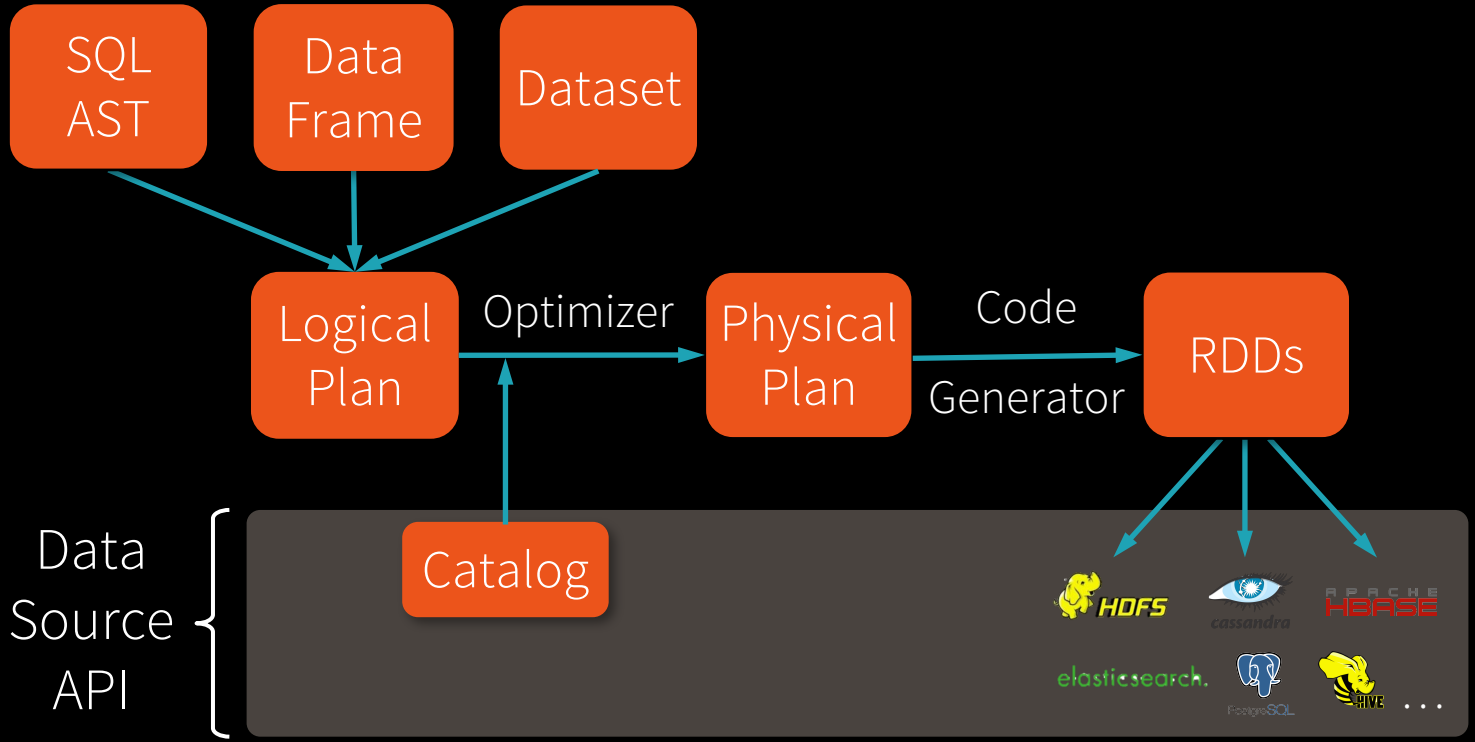
†Databricks Inc. *MIT CSAIL ‡AMPLab, UC Berkeley

ABSTRACT

Spark SQL is a new module in Apache Spark that integrates relational processing with Spark's functional programming API. Built on our experience with Shark, Spark SQL lets Spark programmers leverage the benefits of relational processing (*e.g.*, declarative

While the popularity of relational systems shows that users often prefer writing declarative queries, the relational approach is insufficient for many big data applications. First, users want to perform ETL to and from various data sources that might be semi- or unstructured, requiring custom code. Second, users want to perform

Execution Steps



Example: DataFrame API

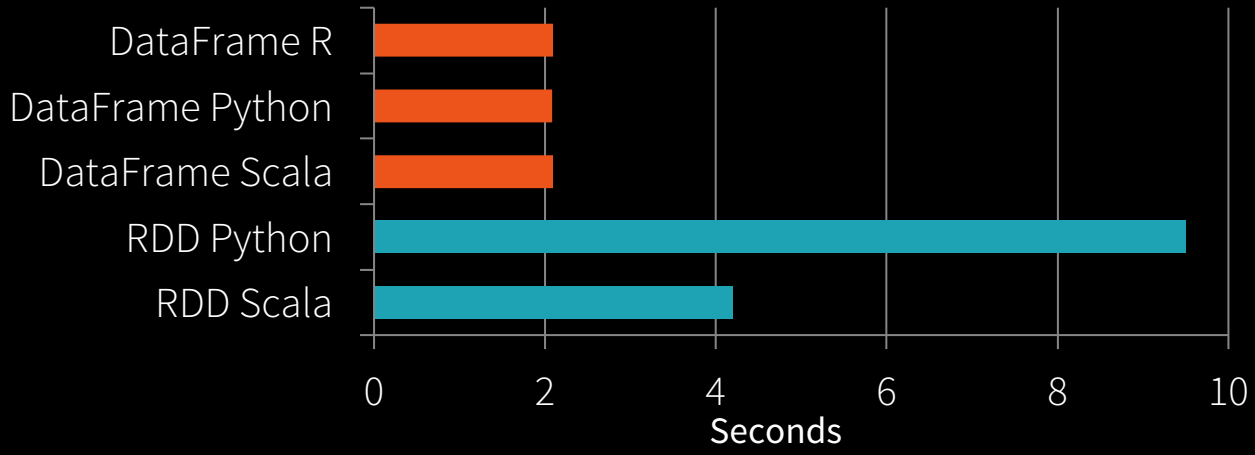
DataFrames hold rows with a known schema and offer relational operations on them through a DSL

```
> sparkR.session()

> users <- sql(“select * from users”)

> ca.users <- users[users$state == “CA”]

> nrow(ca.users)

> ca.users %>% groupBy(“name”) %>% avg(“age”)

> dapply(ca.users, function(df) { df$age + rnorm(nrow(df)) }, ...)
```

Expression AST

What Structured APIs Enable

1. Compact binary representation
2. Optimization across operators (join order, pushdown, etc)
3. Runtime code generation



New in 2.2 & 2.3: Structured Streaming

High-level streaming API built over DataFrames

- Event time, windowing, sessions, transactional sinks
- Arbitrary or Customized Stateful Aggregation
- Stream-to-stream joins

Supports interactive & batch queries

- Ad-hoc queries via JDBC, joins with static data

Works by just incrementalizing static DataFrame queries!

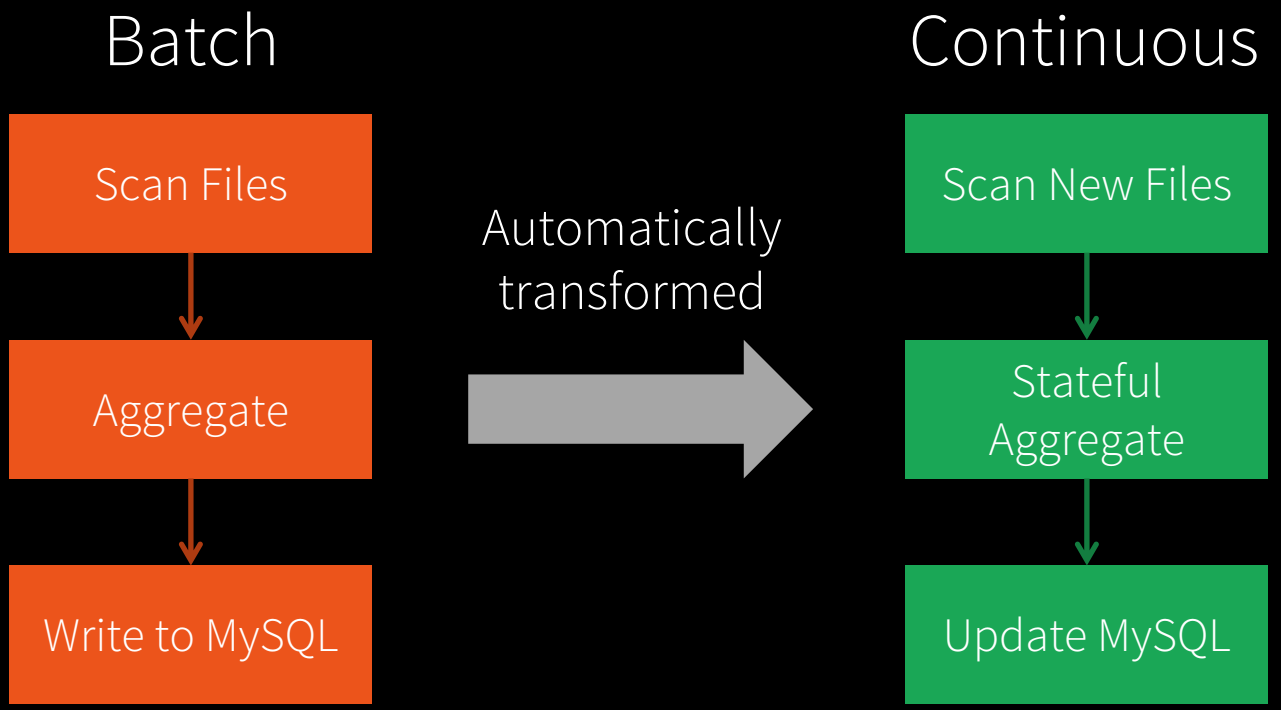
Example: Batch Aggregation

```
> logs <- read.df(source = "json”, path = "s3://logs")

> logs %>%
    groupBy(“userid”, “hour”) %>%
    avg(“latency”) %>%
    write.df(source = "jdbc”, path = "jdbc:mysql//...")
```

Example: Continuous Aggregation

```
> logs <- read.stream(source = "json”, path = "s3://logs")

> logs %>%
    groupBy(“userid”, “hour”) %>%
    avg(“latency”) %>%
    write.stream(source = "jdbc”, path = "jdbc:mysql//...")
```

Incremental Execution



Benefits of Structured Streaming

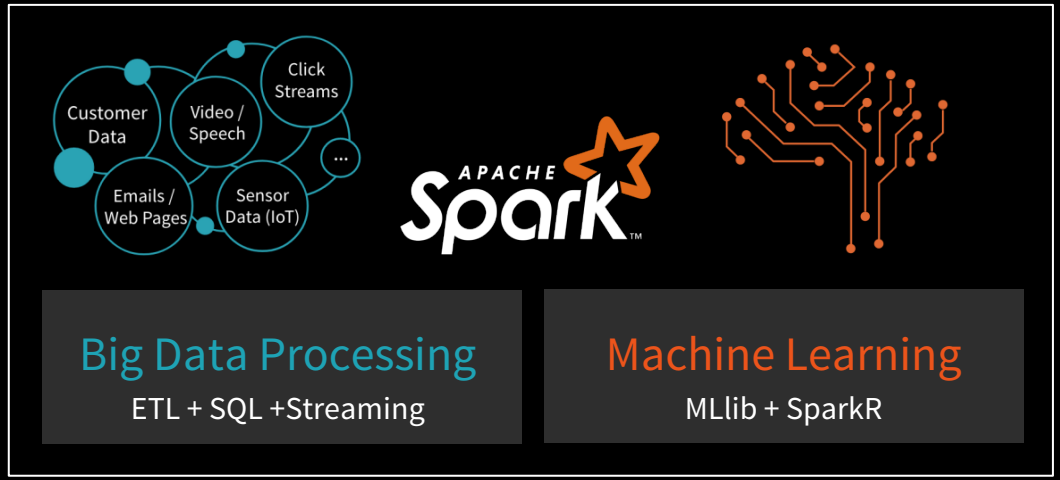
1. Results always consistent with a batch job on a prefix of data

2. Same code and libraries run in both modes

3. Ad-hoc queries on consistent snapshots of state

Apache Spark: The First Unified Analytics Engine

Uniquely combines Data & AI technologies



databricks


databricks

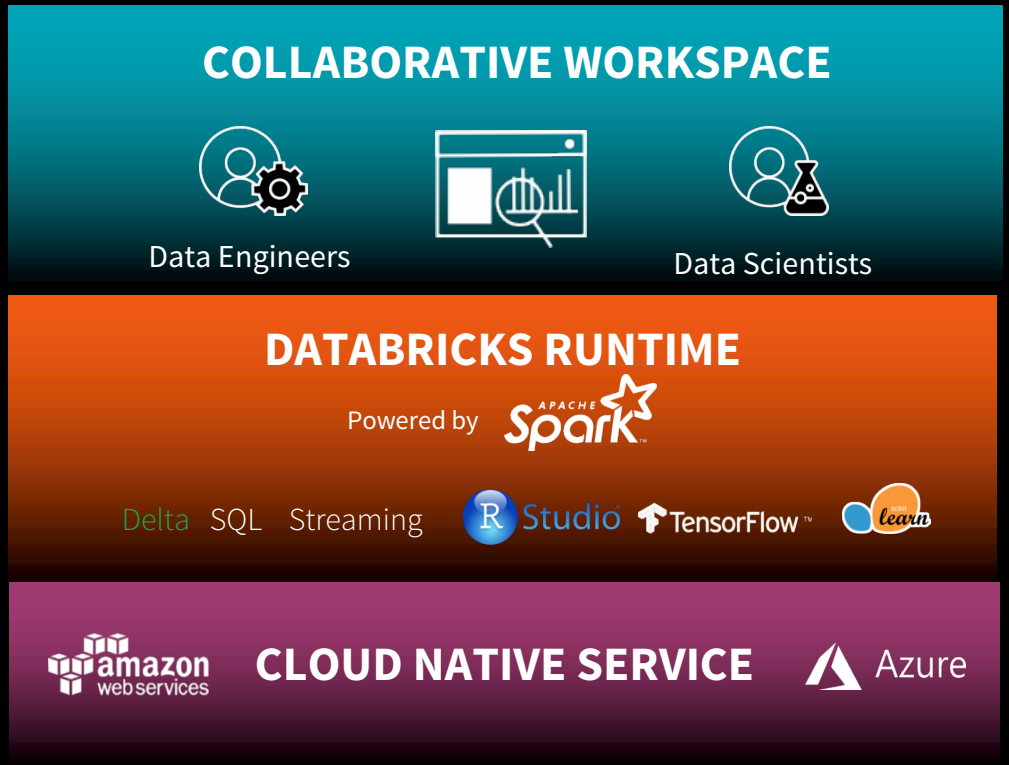

APACHE
Spark

Databricks' Unified Analytics Platform

Unifies Data Engineers and Data Scientists

Unifies Data and AI Technologies

Eliminates infrastructure complexity



Conclusion

Unified computing models are essential for usability & performance

- Same results across batch, interactive & streaming
- Powerful cross-library optimizations
- Same functionality across programming languages

In most areas of computing, we converged to unified systems

- UNIX OS, general-purpose languages, SQL, etc
- Why not in cluster computing?
- And that's Unified Analytics Platform
  - With Apache Spark at the core of Databricks Runtime

+

databricks

Thank You