How Apache Spark™ 3.0 and Delta Lake Enhances Data Lake Reliability

Easily Build your Data Lakehouse with Apache Spark 3.0 and Delta Lake

Denny Lee dennyglee

August 2020

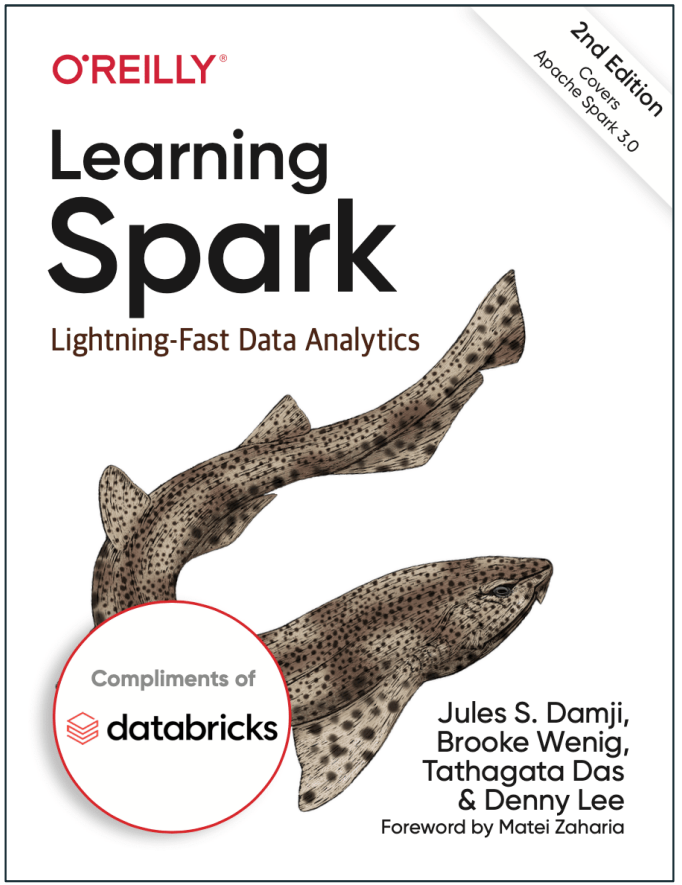
databricks

About the Speaker

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.

O'Reilly Learning Spark 2nd Edition for free

http://dbricks.co/get-ebook



databricks

Go to delta.io to Read the VLDB Paper!

Delta Lake: High-Performance ACID Table Storage over Cloud Object Stores

Michael Armbrust, Tathagata Das, Liwen Sun, Burak Yavuz, Shixiong Zhu, Mukul Murthy, Joseph Torres, Herman van Hovell, Adrian Ionescu, Alicja Łuszczak, Michał Świtakowski, Michał Szafrański, Xiao Li, Takuya Ueshin, Mostafa Mokhtar, Peter Boncz[1], Ali Ghodsi[2], Sameer Paranjpye, Pieter Senster, Reynold Xin, Matei Zaharia[3]
Databricks, [1]CWI, [2]UC Berkeley, [3]Stanford University
delta-paper-authors@databricks.com

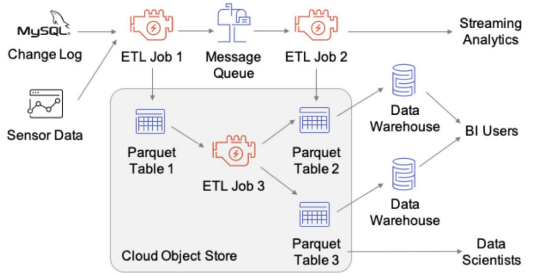
ABSTRACT

Cloud object stores such as Amazon S3 are some of the largest and most cost-effective storage systems on the planet, making them an attractive target to store large data warehouses and data lakes. Unfortunately, their implementation as key-value stores makes it difficult to achieve ACID transactions and high performance: metadata operations such as listing objects are expensive, and consistency guarantees are limited. In this paper, we present Delta Lake, an open source ACID table storage layer over cloud object stores initially developed at Databricks. Delta Lake uses a transaction log that is compacted into Apache Parquet format to provide ACID properties, time travel, and significantly faster metadata operations for large tabular datasets (e.g., the ability to quickly search billions of table partitions for those relevant to a query). It also leverages this design to provide high-level features such as automatic data layout optimization, upserts, caching, and audit logs. Delta Lake tables can be accessed from Apache Spark, Hive, Presto, Redshift and other systems. Delta Lake is deployed at thousands of Databricks customers that process exabytes of data per day, with the largest instances managing exabyte-scale datasets and billions of objects.

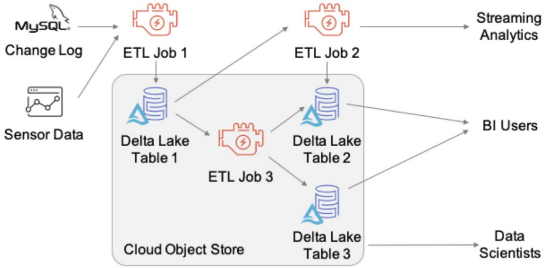
The major open source "big data" systems, including Apache Spark, Hive and Presto [45, 52, 42], support reading and writing to cloud object stores using file formats such as Apache Parquet and ORC [13, 12]. Commercial services including AWS Athena, Google BigQuery and Redshift Spectrum [1, 29, 39] can also query directly against these systems and these open file formats.

Unfortunately, although many systems support reading and writing to cloud object stores, achieving *performant* and *mutable* table storage over these systems is challenging, making it difficult to implement data warehousing capabilities over them. Unlike distributed filesystems such as HDFS [5], or custom storage engines in a DBMS, most cloud object stores are merely key-value stores, with no cross-key consistency guarantees. Their performance characteristics also differ greatly from distributed filesystems and require special care.

The most common way to store relational datasets in cloud object stores is using columnar file formats such as Parquet and ORC, where each table is stored as a set of objects (Parquet or ORC "files"), possibly clustered into "partitions" by some fields (e.g., a separate set of objects for each date) [45]. This approach can offer acceptable performance for scan workloads as long as the object



(a) Pipeline using separate storage systems.



(b) Using Delta Lake for both stream and table storage.

databricks

databricks

Unified data analytics platform for accelerating innovation across data science, data engineering, and business analytics

Global company with 5,000 customers and 450+ partners

Original creators of popular data and machine learning open source projects

Deep Dive into the New Features of Apache Spark 3.0

Xiao Li (Github: gatorsmile)

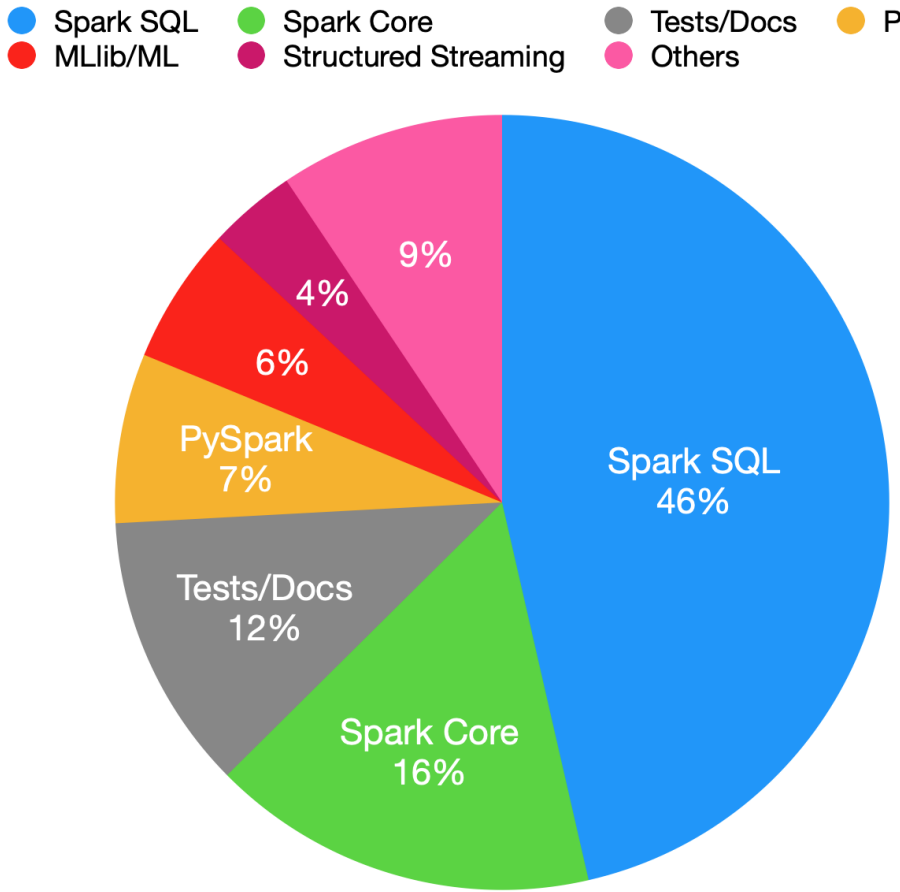
Wenchen Fan (Github: cloud-fan)

Delta Lake 0.7.0 + Spark 3.0 AMA

Burak Yavuz

Tathagata Das

Denny Lee



APACHE Spark™

3400+ Resolved JIRAs in Spark 3.0 rc2

Performance

Adaptive Query Execution

Dynamic Partition Pruning

Query Compilation Speedup

Join Hints

Built-in Data Sources

Parquet/ORC Nested Column Pruning

CSV Filter Pushdown

Parquet: Nested Column Filter Pushdown

New Binary Data Source

Richer APIs

Accelerator-aware Scheduler

Built-in Functions

pandas UDF Enhancements

DELETE/UPDATE/ MERGE in Catalyst

SQL Compatibility

Overflow Checking

ANSI Store Assignment

Proleptic Gregorian Calendar

Reserved Keywords

Extensibility and Ecosystem

Data Source V2 API + Catalog Support

Hadoop 3 Support

Hive 3.x Metastore Hive 2.3 Execution

Java 11 Support

Monitoring and Debuggability

Structured Streaming UI

DDL/DML Enhancements

Observable Metrics

Event Log Rollover

Performance

Adaptive Query Execution

Dynamic Partition Pruning

Query Compilation Speedup

Join Hints

Built-in Data Sources

Parquet/ORC Nested Column Pruning

CSV Filter Pushdown

Parquet: Nested Column Filter Pushdown

New Binary Data Source

Richer APIs

Accelerator-aware Scheduler

Built-in Functions

pandas UDF Enhancements

DELETE/UPDATE/ MERGE in Catalyst

SQL Compatibility

Overflow Checking

ANSI Store Assignment

Proleptic Gregorian Calendar

Reserved Keywords

Extensibility and Ecosystem

Data Source V2 API + Catalog Support

Hadoop 3 Support

Hive 3.x Metastore Hive 2.3 Execution

Java 11 Support

Monitoring and Debuggability

Structured Streaming UI

DDL/DML Enhancements

Observable Metrics

Event Log Rollover

Performance

Adaptive Query Execution

Dynamic Partition Pruning

Query Compilation Speedup

Join Hints

Achieve high performance for interactive, batch, streaming and ML workloads

Performance

Adaptive Query Execution

Dynamic Partition Pruning

Query Compilation Speedup

Join Hints

Achieve high performance for interactive, batch, streaming and ML workloads

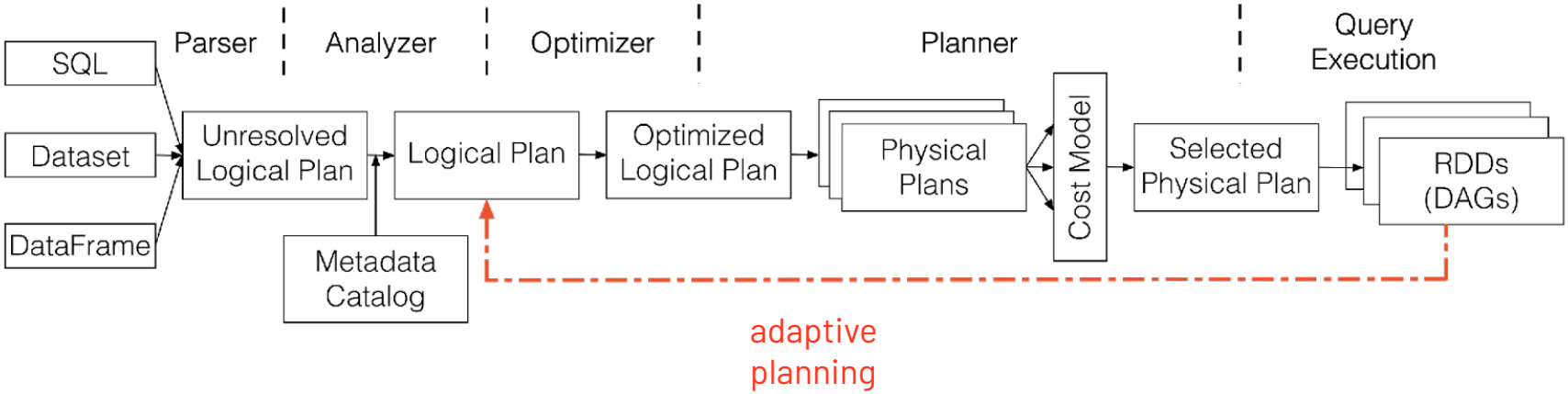
Spark Catalyst Optimizer

Spark 1.x, Rule

Spark 2.x, Rule + Cost

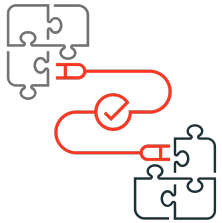
Spark 3.0, Rule + Cost + Runtime

Adaptive Query Execution [AQE]

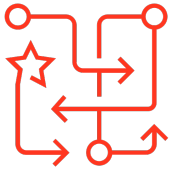


Based on statistics of the finished plan nodes, re-optimize the execution plan of the remaining queries

Apache Spark™ 3.0 AQE Fundamentals



Dynamically switching join strategies

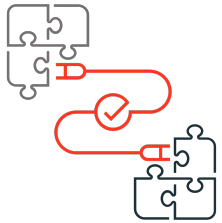


Dynamically coalescing shuffle partitions

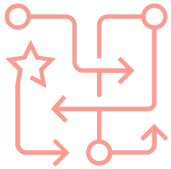
Dynamically optimizing skew joins

Reference: Adaptive Query Execution: Speeding Up Spark SQL at Runtime

Apache Spark™ 3.0 AQE Fundamentals



Dynamically switching join strategies



Dynamically coalescing shuffle partitions

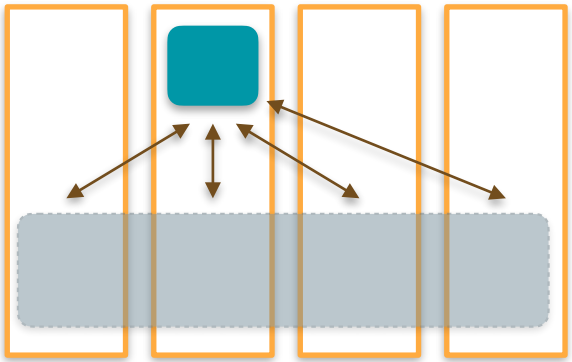
Dynamically optimizing skew joins

Reference: [Adaptive Query Execution: Speeding Up Spark SQL at Runtime]

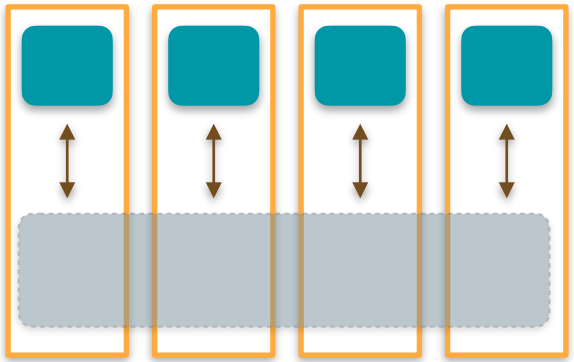
Starting with Broadcast Hash Joins

Apache Spark™ 3.0 AQE Fundamentals

Joins and Shuffling



Broadcast Hash Joins



`spark.sql.autoBroadcastJoinThreshold`

Why not always broadcast join?

Apache Spark™ 3.0 AQE Fundamentals

Missing or Incomplete Statistics

Compressed Files

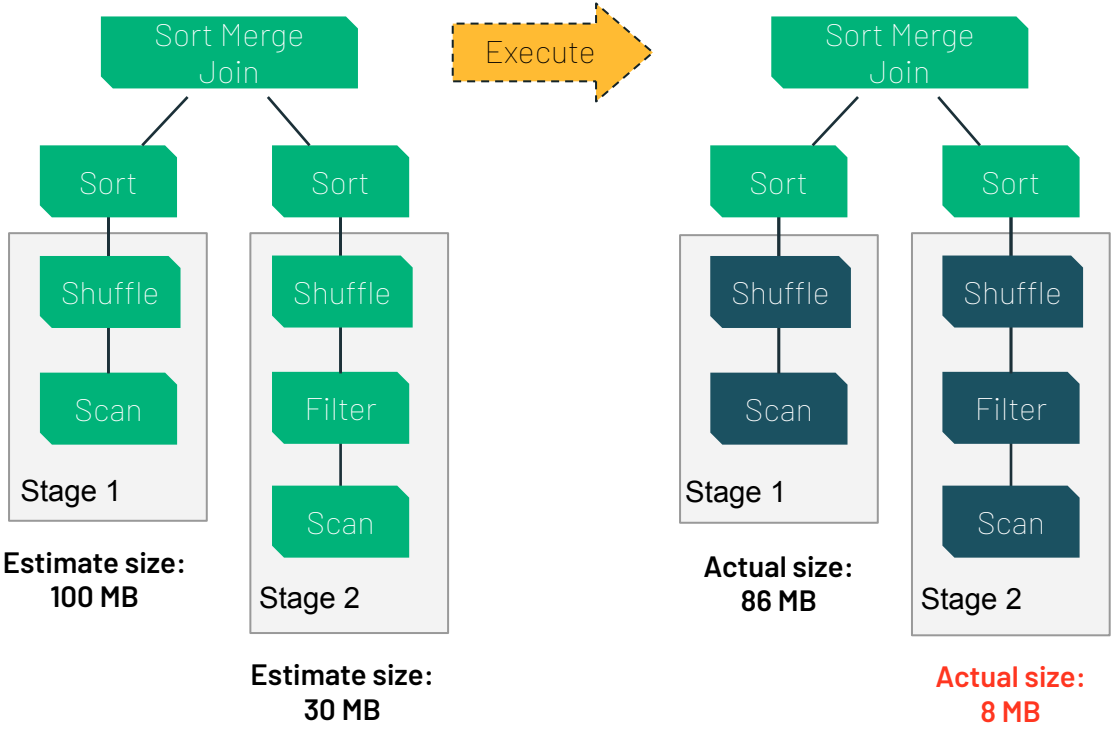
Column Store

Complex Filtering / UDFs

Complex Query Fragments

Dynamically Switching Join Strategies

Apache Spark™ 3.0 AQE Fundamentals



Estimate size: 100 MB

Estimate size: 30 MB

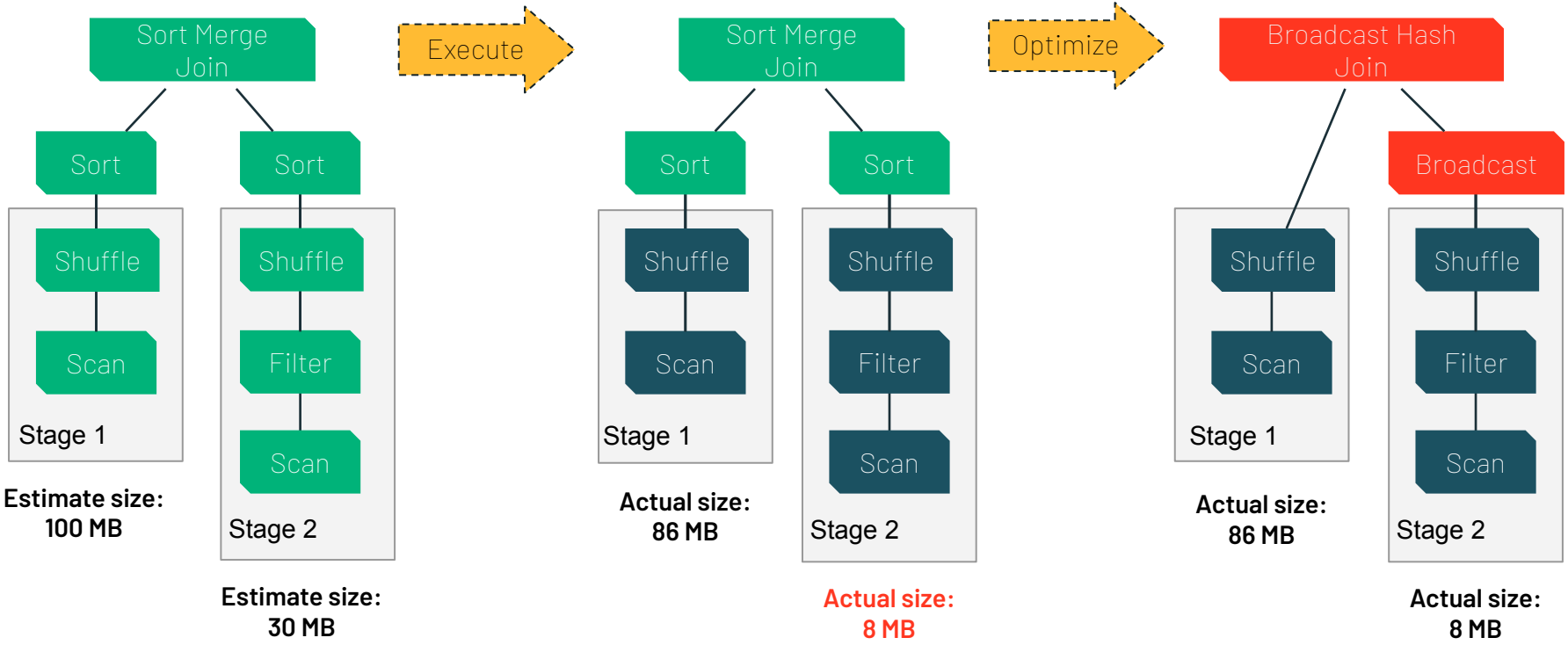
Actual size: 86 MB

Actual size: 8 MB

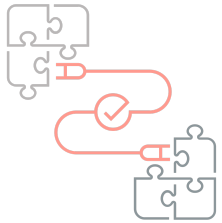
databricks

Dynamically Switching Join Strategies

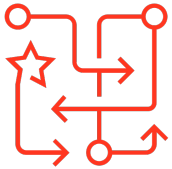
Apache Spark™ 3.0 AQE Fundamentals



Apache Spark™ 3.0 AQE Fundamentals



Dynamically switching join strategies



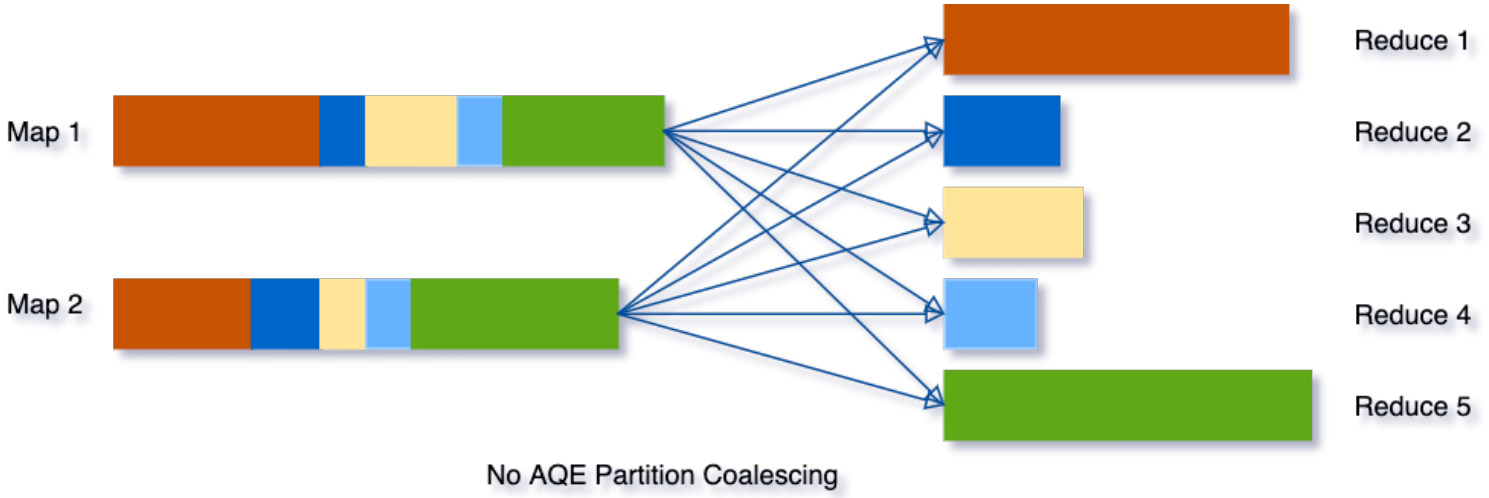
Dynamically coalescing shuffle partitions

Dynamically optimizing skew joins

Reference: [Adaptive Query Execution: Speeding Up Spark SQL at Runtime]

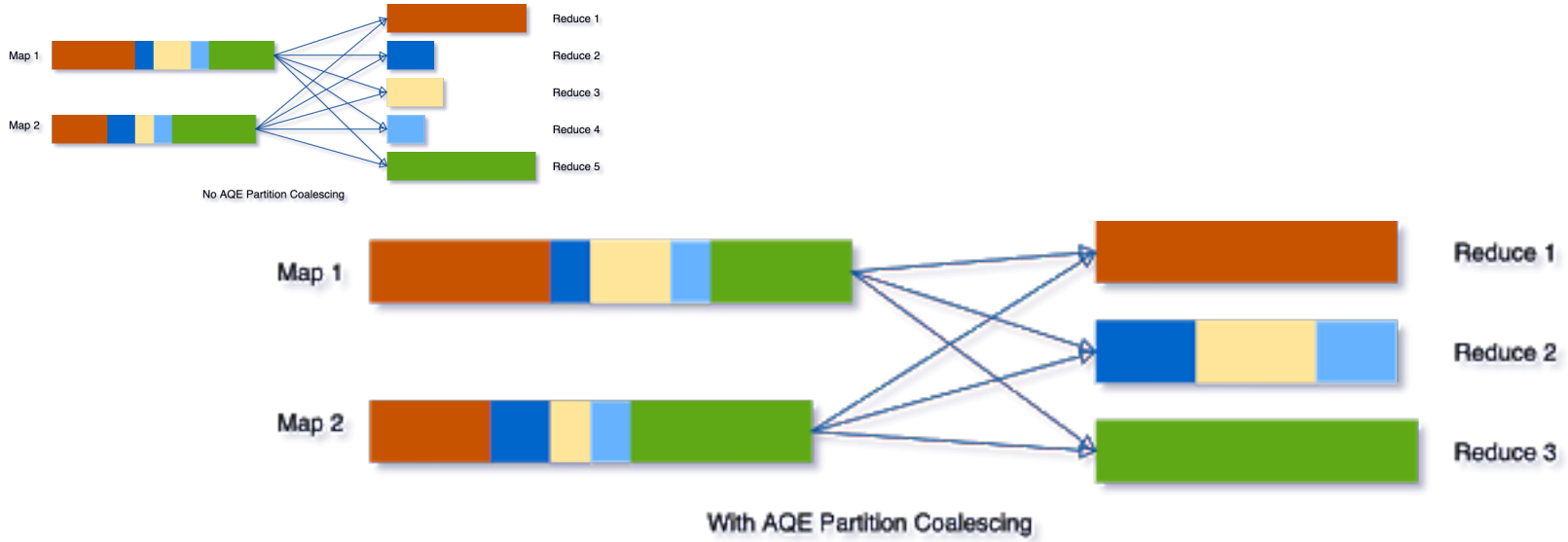
Dynamically Coalescing Shuffle Partitions

Apache Spark™ 3.0 AQE Fundamentals

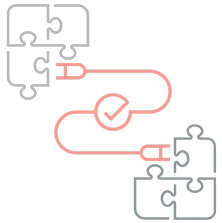


Dynamically Coalescing Shuffle Partitions

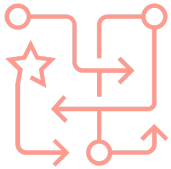
Apache Spark™ 3.0 AQE Fundamentals



Apache Spark™ 3.0 AQE Fundamentals



Dynamically switching join strategies



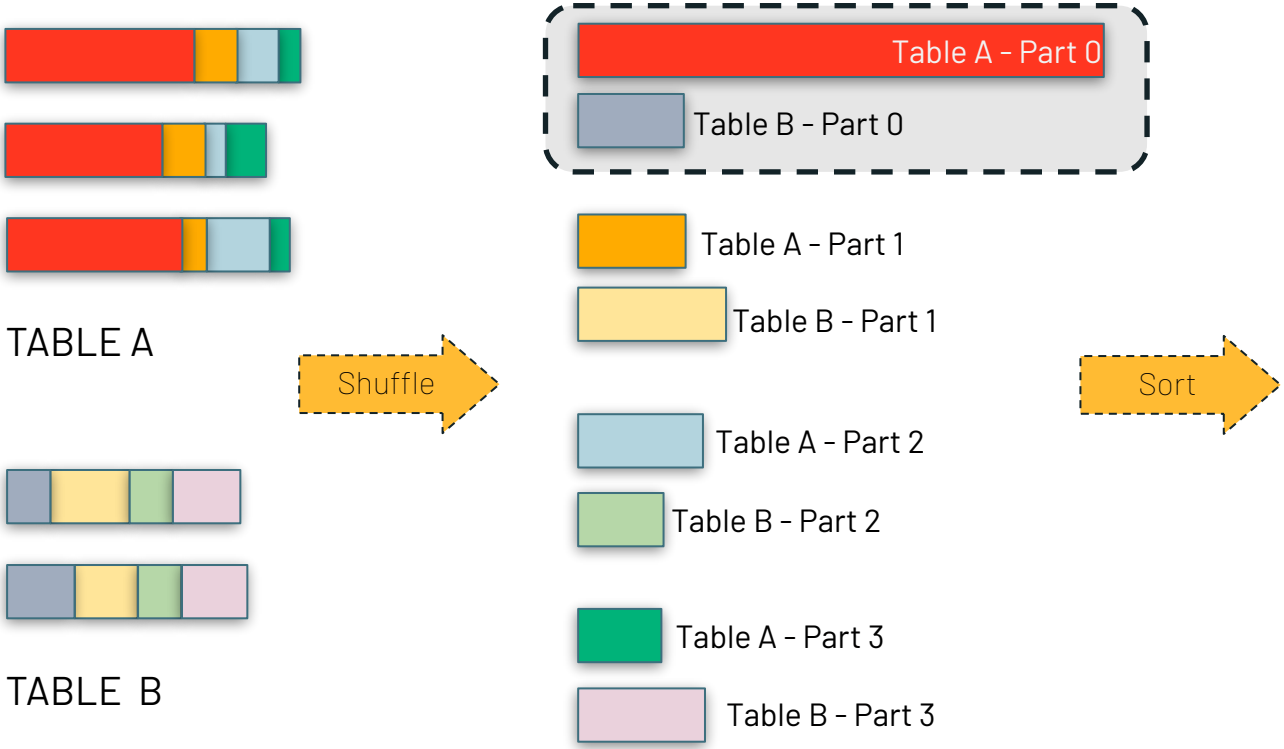
Dynamically coalescing shuffle partitions

Dynamically optimizing skew joins

Reference: [Adaptive Query Execution: Speeding Up Spark SQL at Runtime]

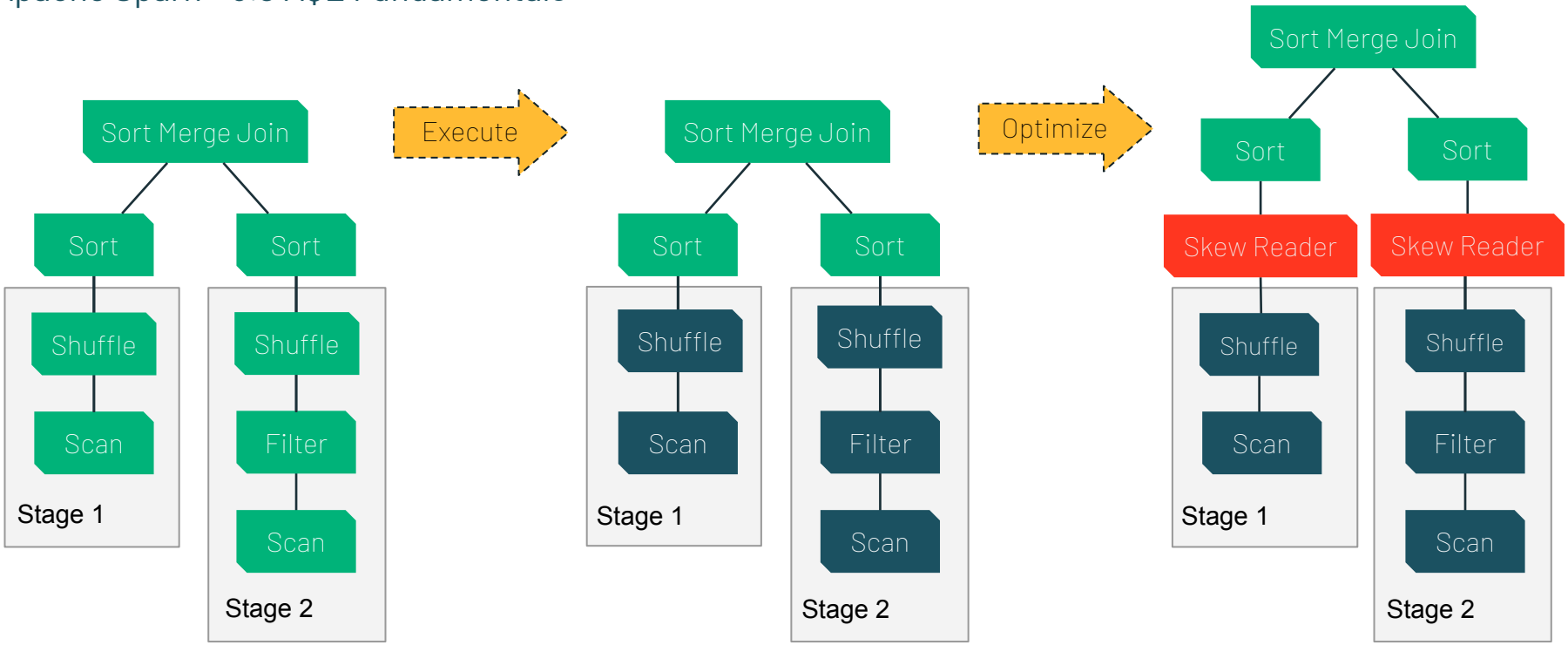
Dynamically Optimizing Skew Joins

Apache Spark™ 3.0 AQE Fundamentals



Dynamically Optimize Skew Joins

Apache Spark™ 3.0 AQE Fundamentals



Dynamically Optimize Skew Joins

Apache Spark™ 3.0 AQE Fundamentals

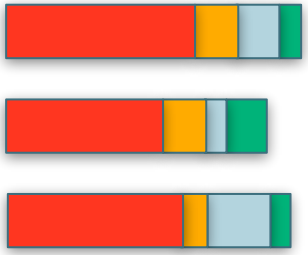


TABLE A

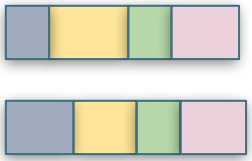


TABLE B

Shuffle

Table A - Part 0

Table B - Part 0

Table A - Part 1

Table B - Part 1

Table A - Part 2

Table B - Part 2

Table A - Part 3

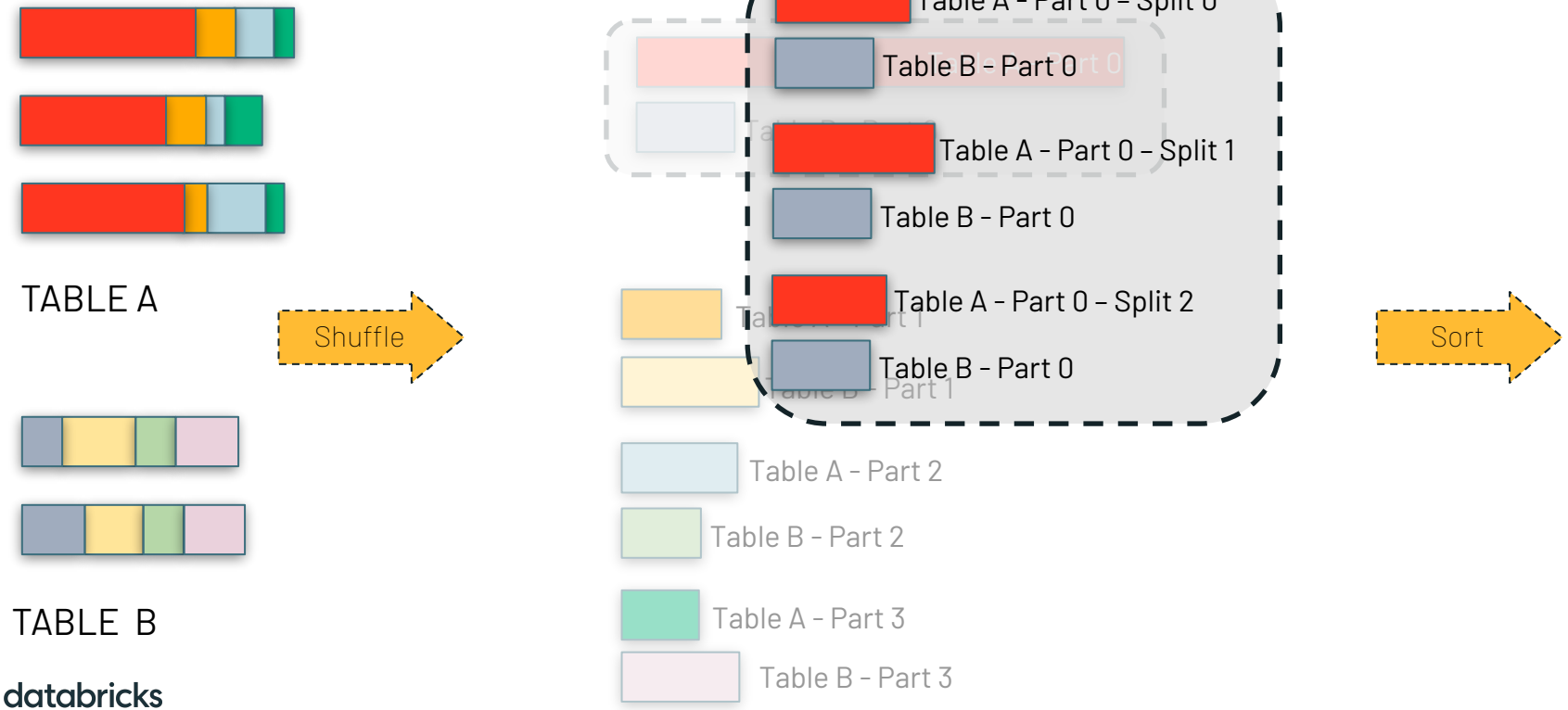
Table B - Part 3

Sort

databricks

Dynamically Optimize Skew Joins

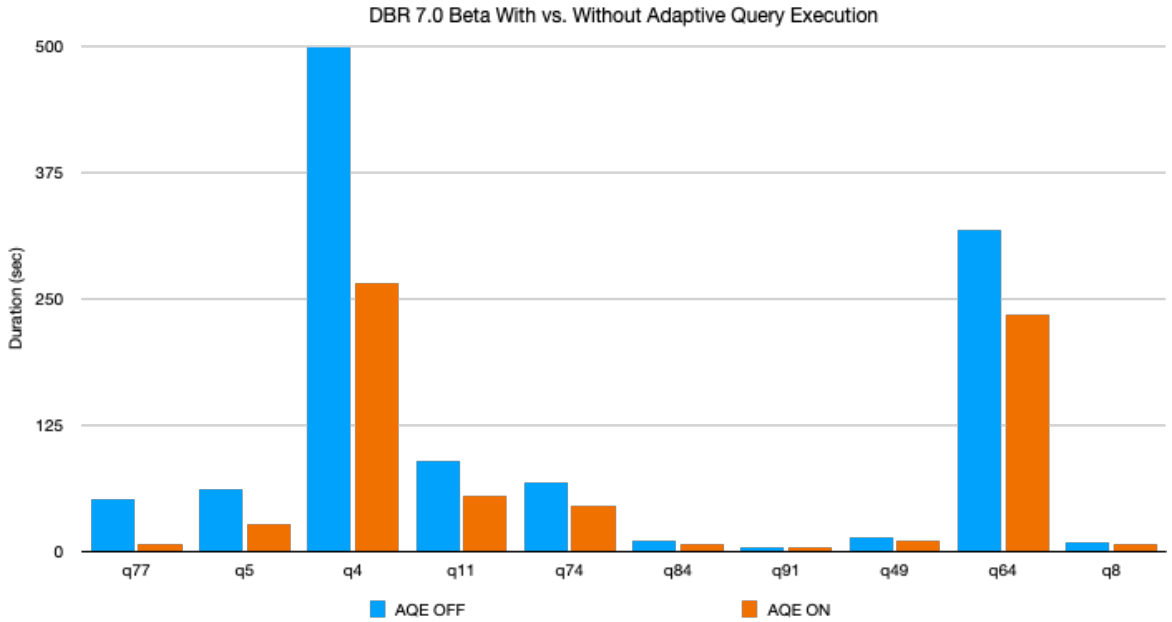
Apache Spark™ 3.0 AQE Fundamentals



databricks

TPC-DS performance gains from AQE

Apache Spark™ 3.0 AQE Fundamentals



Performance

Adaptive Query Execution

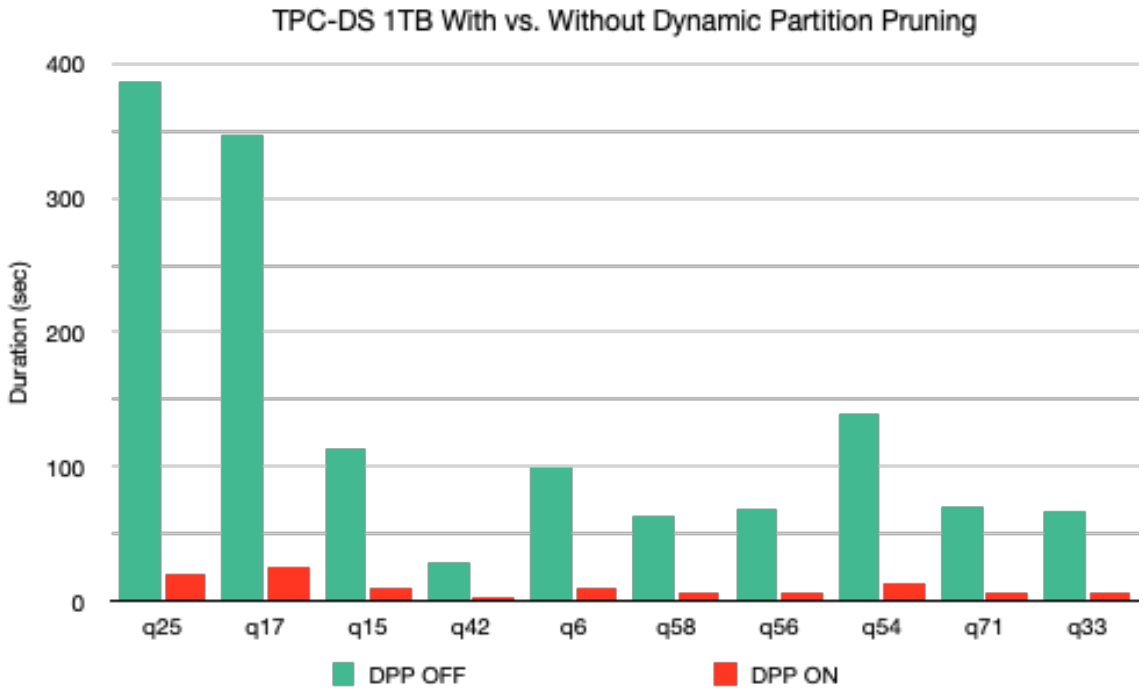
Dynamic Partition Pruning

Query Compilation Speedup

Join Hints

Achieve high performance for interactive, batch, streaming and ML workloads

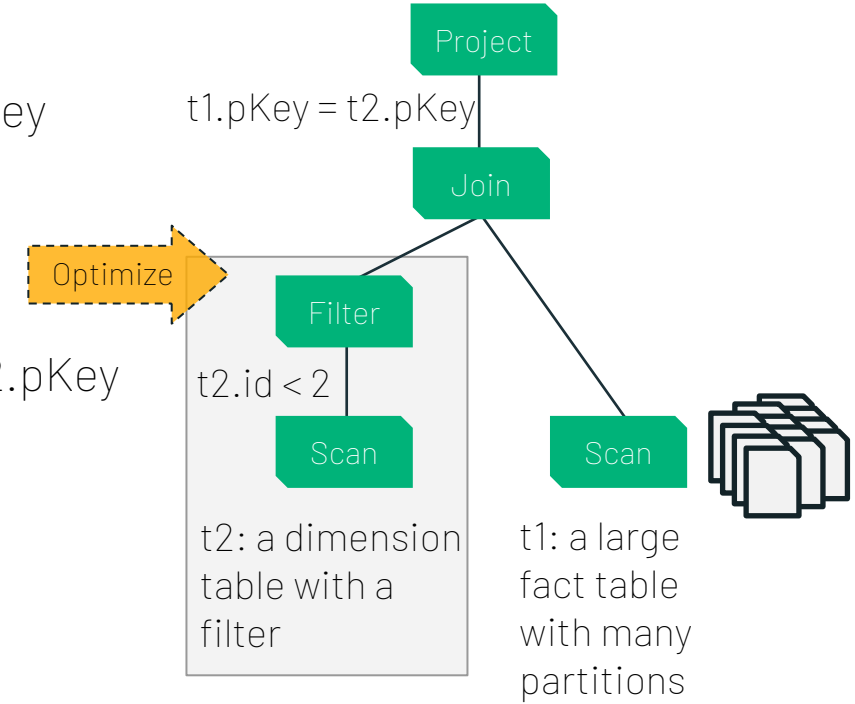
Dynamic Partition Pruning



60 / 102 TPC-DS queries: a speedup between 2x and 18x

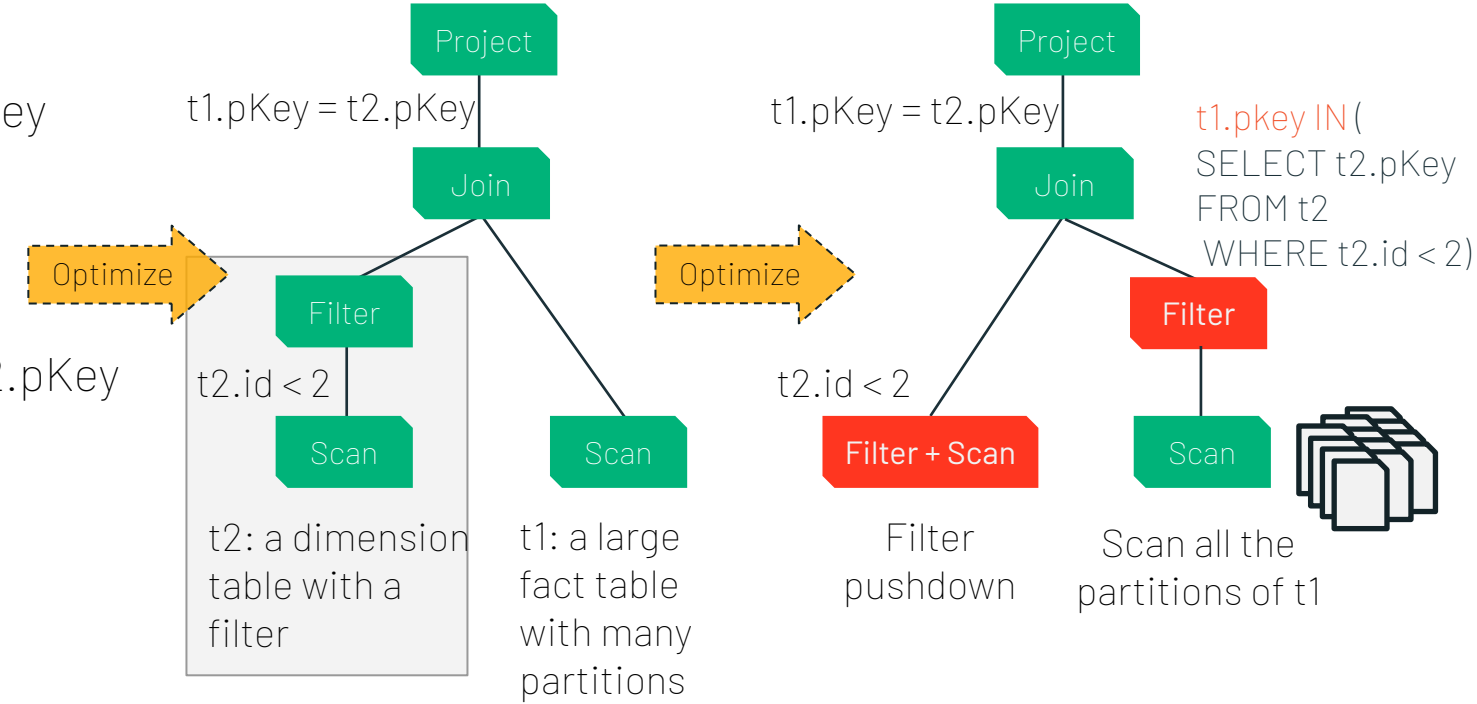
Dynamic Partition Pruning: Before Optimization

```
SELECT t1.id, t2.pKey
  FROM t1
  JOIN t2
    ON t1.pKey = t2.pKey
  AND t2.id < 2
```

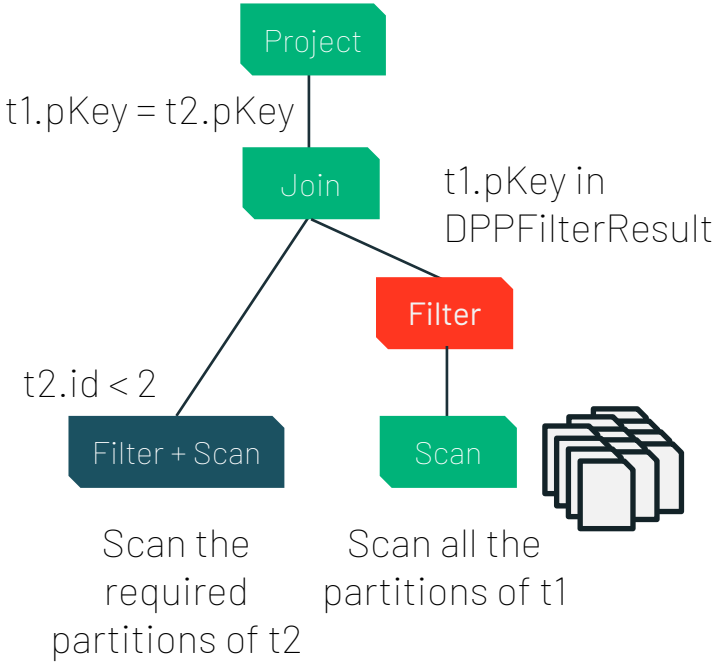


Dynamic Partition Pruning

```
SELECT t1.id, t2.pKey
  FROM t1
  JOIN t2
    ON t1.pKey = t2.pKey
  AND t2.id < 2
```



Dynamic Partition Pruning

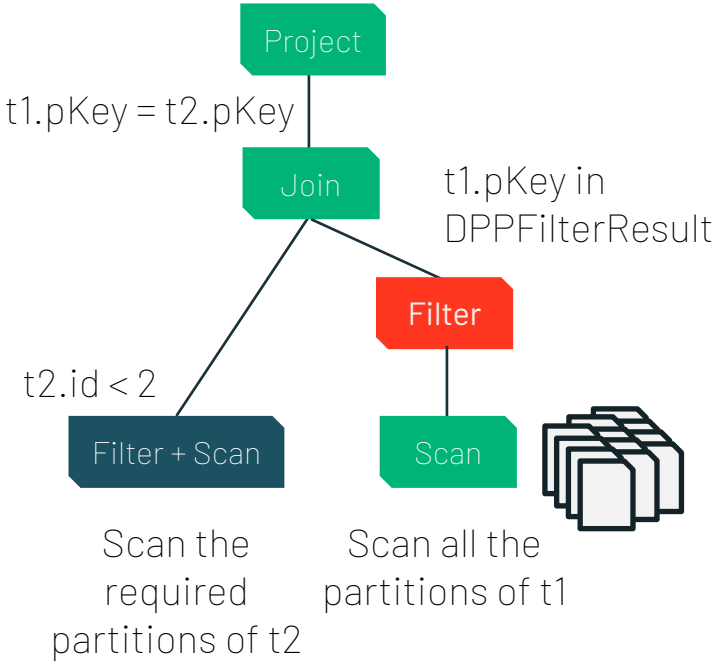


Dynamic Partition Pruning

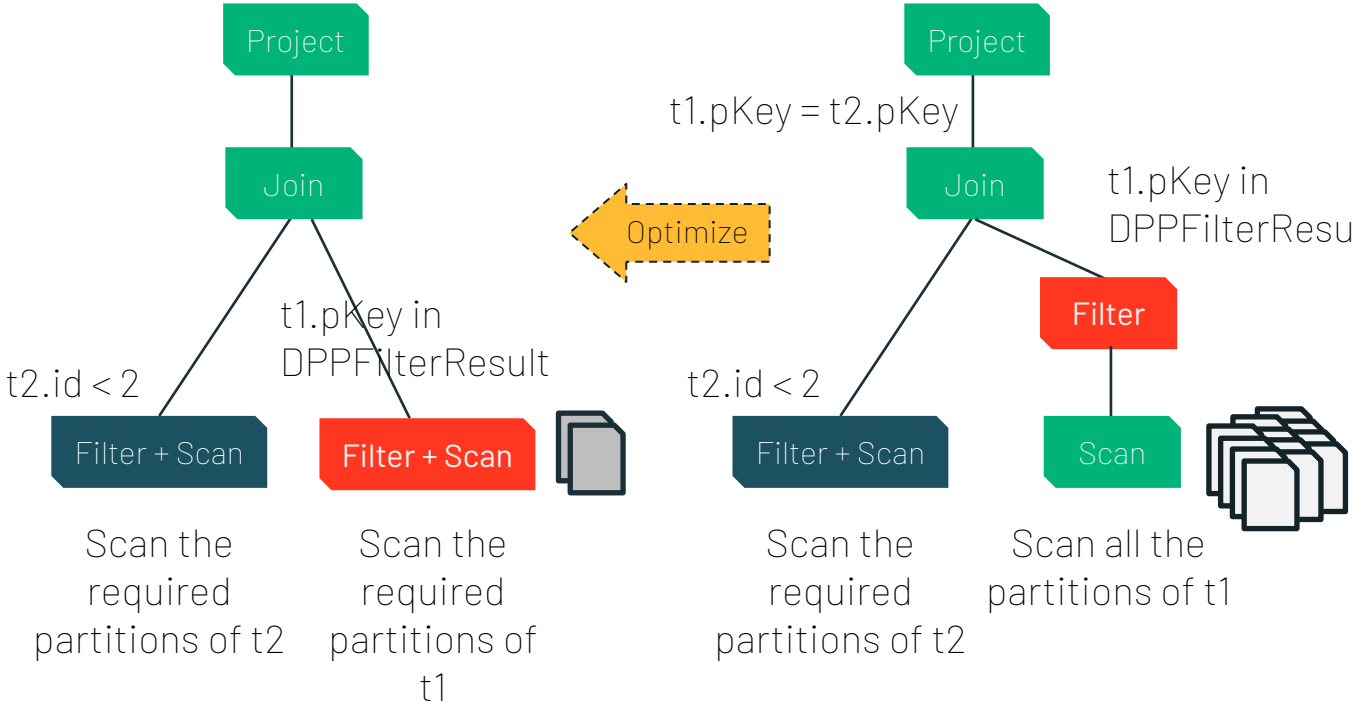
The basic mechanism for DPP inserts a duplicated subquery with the filter from the other side, for the following conditions:

1. the table to prune is partitioned by the JOIN key

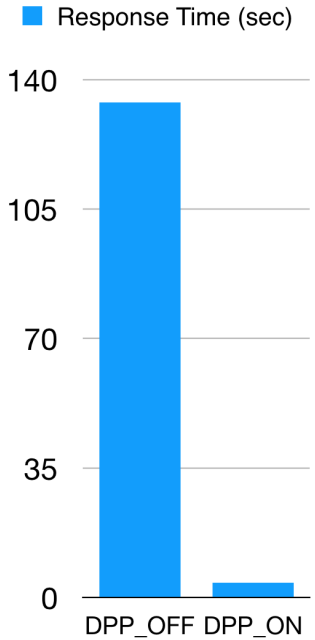
2. the join operation is one of the following types:
   - INNER
   - LEFT SEMI (partitioned on left)
   - LEFT OUTER (partitioned on right)
   - RIGHT OUTER (partitioned on left)



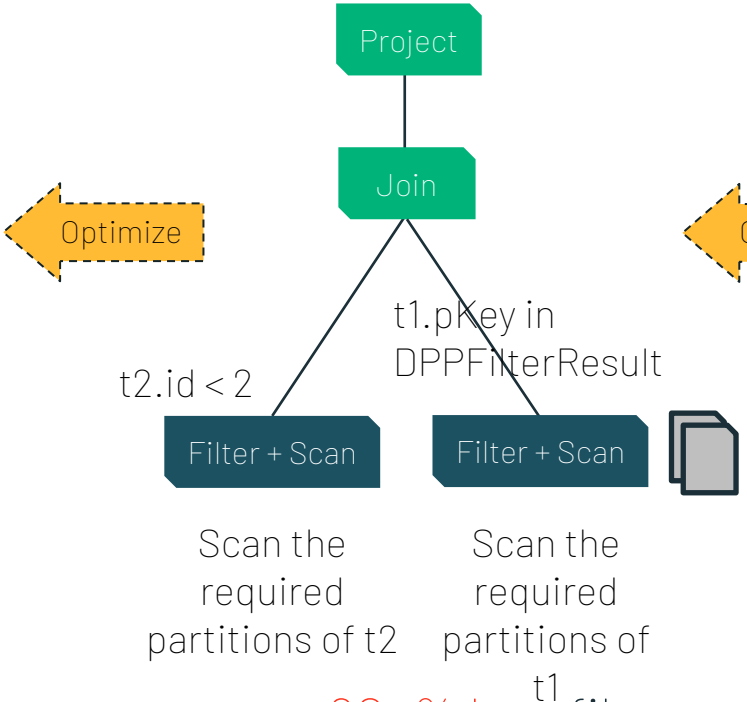
Dynamic Partition Pruning

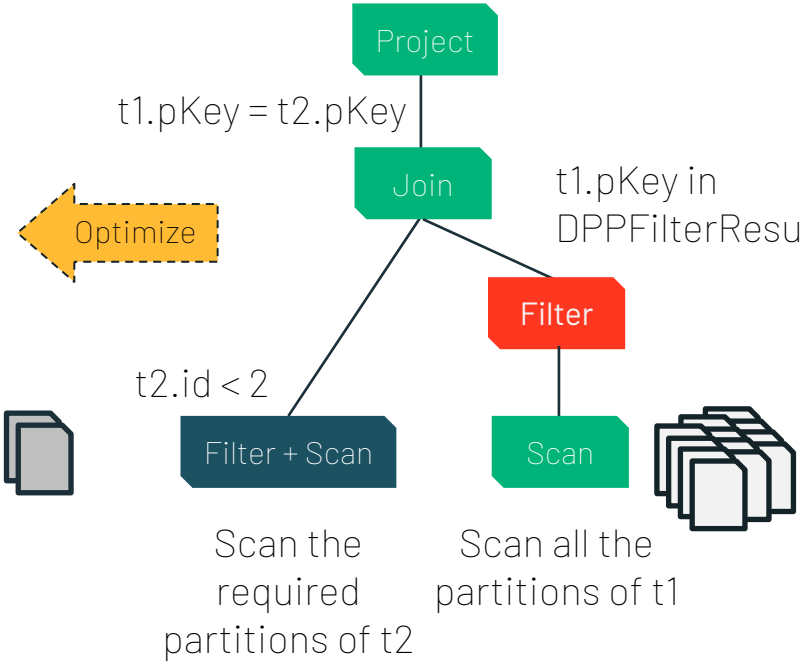


Dynamic Partition Pruning



33 X faster





90+% less file scan

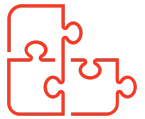
Performance

Adaptive Query Execution

Dynamic Partition Pruning

Query Compilation Speedup

Join Hints



Achieve high performance for interactive, batch, streaming and ML workloads

How to Use Join Hints?

- Broadcast Hash Join

  SELECT /*+ BROADCAST(a) */ id FROM a JOIN b ON a.key = b.key

  Requires one side to be small. No shuffle, no sort, very fast.

- Sort-Merge Join

  SELECT /*+ MERGE(a, b) */ id FROM a JOIN b ON a.key = b.key

  Robust. Can handle any data size. Needs to shuffle and sort data, slower in most cases when the table size is small.

- Shuffle Hash Join

  SELECT /*+ SHUFFLE_HASH(a, b) */ id FROM a JOIN b ON a.key = b.key

  Needs to shuffle data but no sort. Can handle large tables, but will OOM too if data is skewed.

- Shuffle Nested Loop Join

  SELECT /*+ SHUFFLE_REPLICATE_NL(a, b) */ id FROM a JOIN b

  Does not require join keys as it is a cartesian product of the tables

Extensibility and Ecosystem

Data Source V2 API
+ Catalog Support

Java 11
Support

Hadoop 3
Support

Hive 3.x Metastore
Hive 2.3 Execution

Improve the plug-in interface and extend the deployment environments

Extensibility and Ecosystem

Data Source V2 API
+ Catalog Support

Java 11
Support

Hadoop 3
Support

Hive 3.x Metastore
Hive 2.3 Execution

Improve the plug-in interface and extend the deployment environments

Data Source V2

Data Source V2 API

Many motivations for improving Data Source API:

1. The data source API compatibility depends on the upper level (DataFrames) API and cannot leverage the DataSet API.
2. The physical storage information (e.g., partitioning and sorting) is not propagated from the data sources, and thus, not used in the Spark optimizer.
3. Extensibility is not good and operator push-down capabilities are limited.
4. Lacking columnar read interface for high performance
5. The write interface is so general that it does not support transactions.

Data Source V2

Data Source V2 API

Many motivations for improving Data Source API:

1. The data source API compatibility depends on the upper level (DataFrames) API and cannot leverage the DataSet API.
2. The physical storage information (e.g., partitioning and sorting) is not propagated from the data sources, and thus, not used in the Spark optimizer.
3. Extensibility is not good and operator push-down capabilities are limited.
4. Lacking columnar read interface for high performance
5. **The write interface is so general that it does not support transactions.**

Catalog plugin API

Users can register customized catalogs and use Spark to access/ manipulate table metadata directly.

```
-- Create Table
CREATE TABLE events (
 date DATE,
 eventId STRING,
 eventType STRING,
)
```

```
-- Alter Table Schema
ALTER TABLE events
  ADD COLUMNS (
    eventCategory STRING
  )
```

Delta Lake 0.7.0 is the first release on Apache Spark 3.0 and adds support for metastore-defined tables and SQL DDLs

Support for defining tables in the Hive metastore

With Spark 3.0 metastore support, you can define Delta tables in the Hive metastore and use the table name in all SQL operations

```
-- Create Table
CREATE TABLE events (
 date DATE,
 eventId STRING,
 eventType STRING,
)
```

```
-- Alter Table Schema
ALTER TABLE events
  ADD COLUMNS (
    eventCategory STRING
  )
```

Support for SQL Delete, Update and Merge

SQL - not just for inserts any more!

```
-- Create Table
CREATE TABLE events (
 date DATE,
 eventId STRING,
 eventType STRING,
)
```

```
-- Delete events
DELETE FROM events WHERE
date < '2017-01-01'
```

```
-- Alter Table Schema
ALTER TABLE events
  ADD COLUMNS (
    eventCategory STRING
  )
```

```
-- Update events
UPDATE events SET
eventType = 'click' WHERE
eventType = 'click'
```

```
-- Insert into table
INSERT INTO events
SELECT * FROM newEvents
```

```
-- Upsert data to a target Delta
-- table using merge
MERGE INTO events
USING updates
ON events.eventId = updates.eventId
WHEN MATCHED THEN UPDATE
     SET events.data = updates.data
WHEN NOT MATCHED THEN INSERT
     (date, eventId, data)
     VALUES (date, eventId, data)
```

But what happens with DMLs under the covers?

What really happens to the file system when you run delete, update, and merge?

```
DELETE FROM events ...
UPDATE events ...
MERGE INTO events ...
```

But what happens with DML under the covers?

What really happens to the file system when you run delete, update, and merge?

```
DELETE FROM events ...
UPDATE events ...
MERGE INTO events ...
```

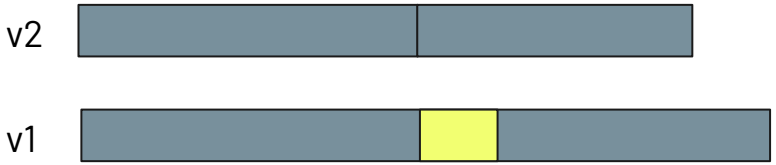
v1

But what happens with DML under the covers?

What really happens to the file system when you run delete, update, and merge?

```
DELETE FROM events ...
UPDATE events ...
MERGE INTO events ...
```

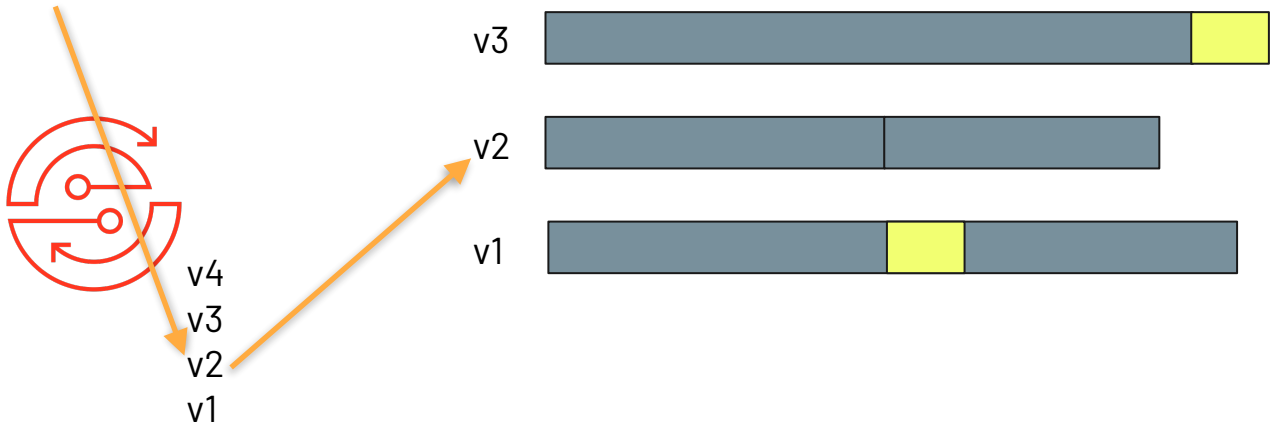
v2



Time Travel

The transaction log and additive files = data versioning

```
SELECT * FROM events
VERSION AS OF 2
```



Control Table History Retention

- `delta.logRetentionDuration`: controls how long the history for a table is kept.
- `delta.deletedFileRetentionDuration`: controls how long ago a file must have been deleted before being a candidate for VACUUM.

```
ALTER TABLE delta.`pathToDeltaTable`
SET TBLPROPERTIES(
   delta.logRetentionDuration = "interval <interval>"
   delta.deletedFileRetentionDuration = "interval <interval>"
)
```

Enable DataSourceV2 and Catalog API Integration

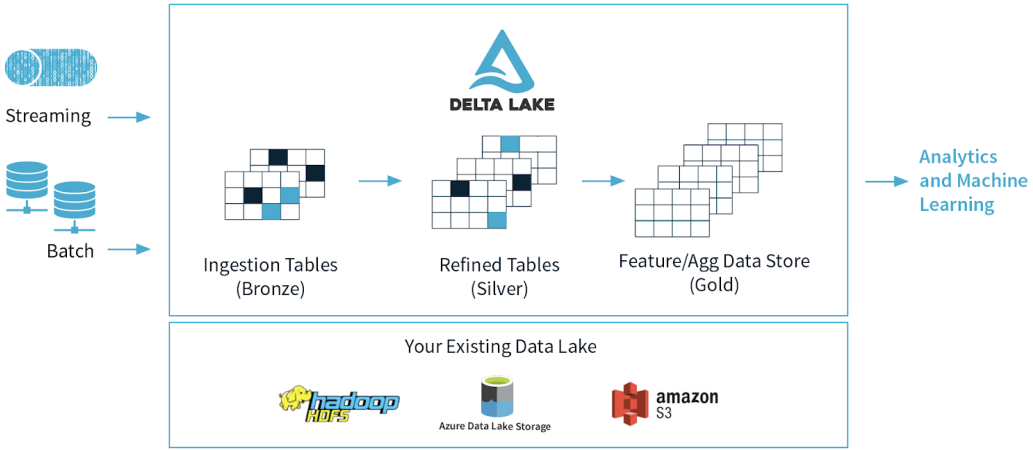
To use the aforementioned features, you must enable the integration with Apache Spark DataSourceV2 and Catalog APIs (since 3.0) by setting the following configurations when creating a new SparkSession.

```
from pyspark.sql import SparkSession

spark = SparkSession \
  .builder \
  .appName("...") \
  .master("...") \
  .config("spark.sql.extensions", "io.delta.sql.DeltaSparkSessionExtension") \
  .config(“spark.sql.catalog.spark_catalog", "org.apache.spark.sql.delta.catalog.DeltaCatalog") \
  .getOrCreate()
```

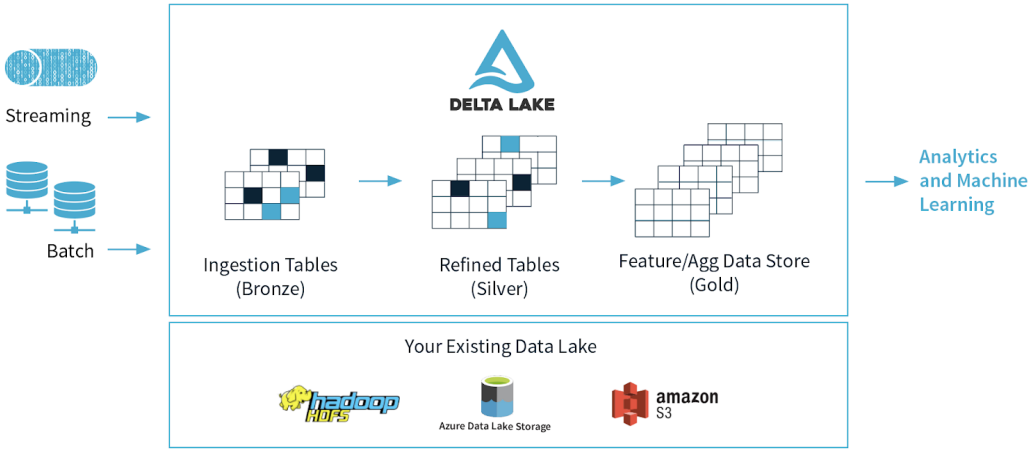
Data Quality Framework

Improved SQL DDL and DMLs and ACID Transactions are just the start



Lakehouse Paradigm

Improved Performance, DW-like capabilities, on low cost cloud object stores



Adaptive Query Execution

Dynamic Partition Pruning

Query Compilation Speedup

Join Hints

Data Source V2 API + Catalog Support

DDL/DML Enhancements

DELETE/UPDATE/ MERGE in Catalyst

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

Try out Spark 3.0 + Delta Lake now!

- Try out Spark 3.0 and Delta Lake now using Databricks Community Edition at databricks.com/try
- Try out the notebooks available at https://github.com/databricks/tech-talks
- Learn more by joining us at the Data + AI Online meetup: https://www.meetup.com/data-ai-online/
- Get the free O'Reilly Learning Spark 2nd Edition eBook at http://dbricks.co/get-ebook (incl Spark 3.0 and Delta Lake)
- Read the VLDB paper: Delta Lake: High-Performance ACID Table Storage over Cloud Object Stores at delta.io