# Faster Spark SQL: Adaptive Query Execution in Databricks

Maryann Xue <maryann.xue@databricks.com>
Allison Wang <allison.wang@databricks.com>

databricks

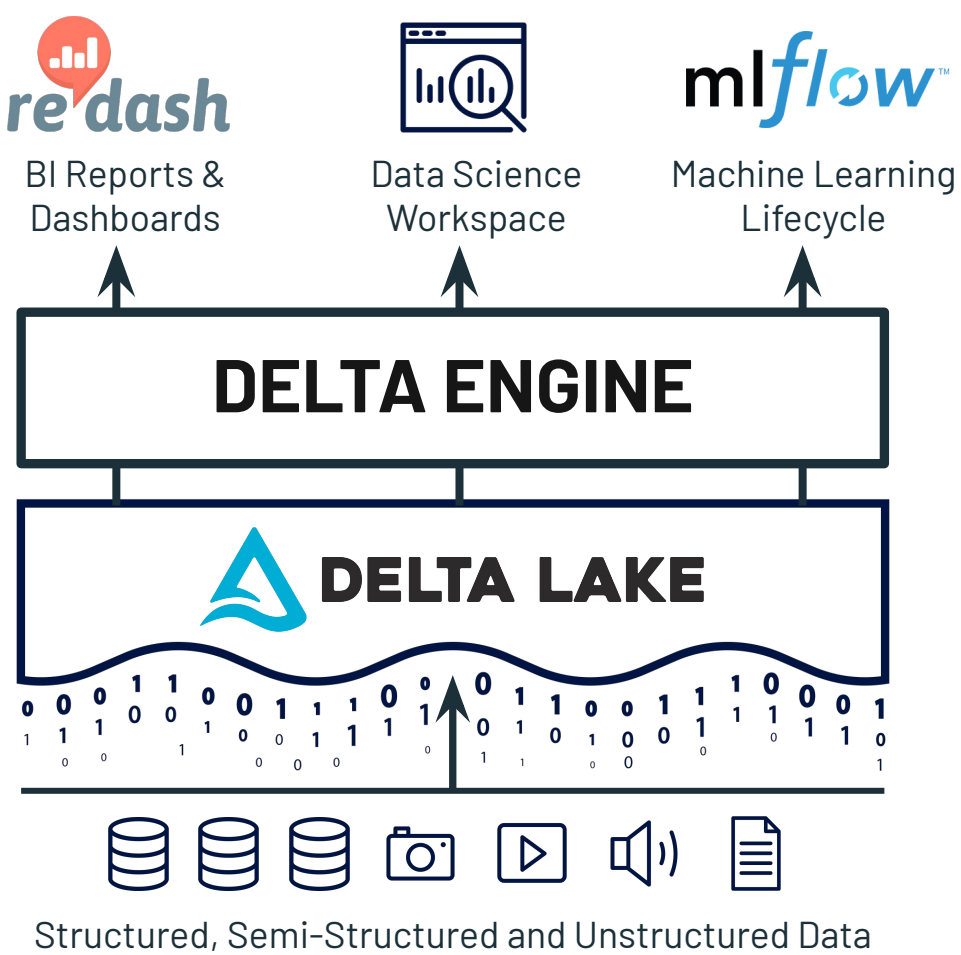
# databricks

A unified data analytics platform for accelerating innovation across data engineering, data science, and analytics

- Global company with over 5,000 customers and 450+ partners
- Original creators of popular data and machine learning open source projects




# Databricks Unified Data Analytics Platform




One platform for every use case

High performance query engine

Structured transactional layer

Data Lake for all your data

databricks

# About us

## Maryann Xue

- Staff Engineer at Databricks
- PMC Member of Apache Calcite & Apache Phoenix

## Allison Wang

- Software Engineer at Databricks

# Cost-Based Optimization (CBO)

Optimize query plans based on cost inferred from **stats**:

- Join reordering
- Join strategy selection
- Shuffle push-down/combination

To enable CBO, need to run **ANALYZE TABLE** command to collect stats.

# CBO Limitations

Stats collection overhead:

- Stats collection can be costly (e.g., column histograms)
- Maintenance effort: running command, keeping it up to date

CBO based on static stats does **NOT** work well when:

- Stale or missing statistics lead to inaccurate estimates
- Complex query plan structure
- Predicates contain UDFs
- Hints do not work for rapidly evolving data

# Adaptive Query Execution (AQE)

**Re-optimize** query plans based on cost inferred from **runtime stats**.

AQE can work with:

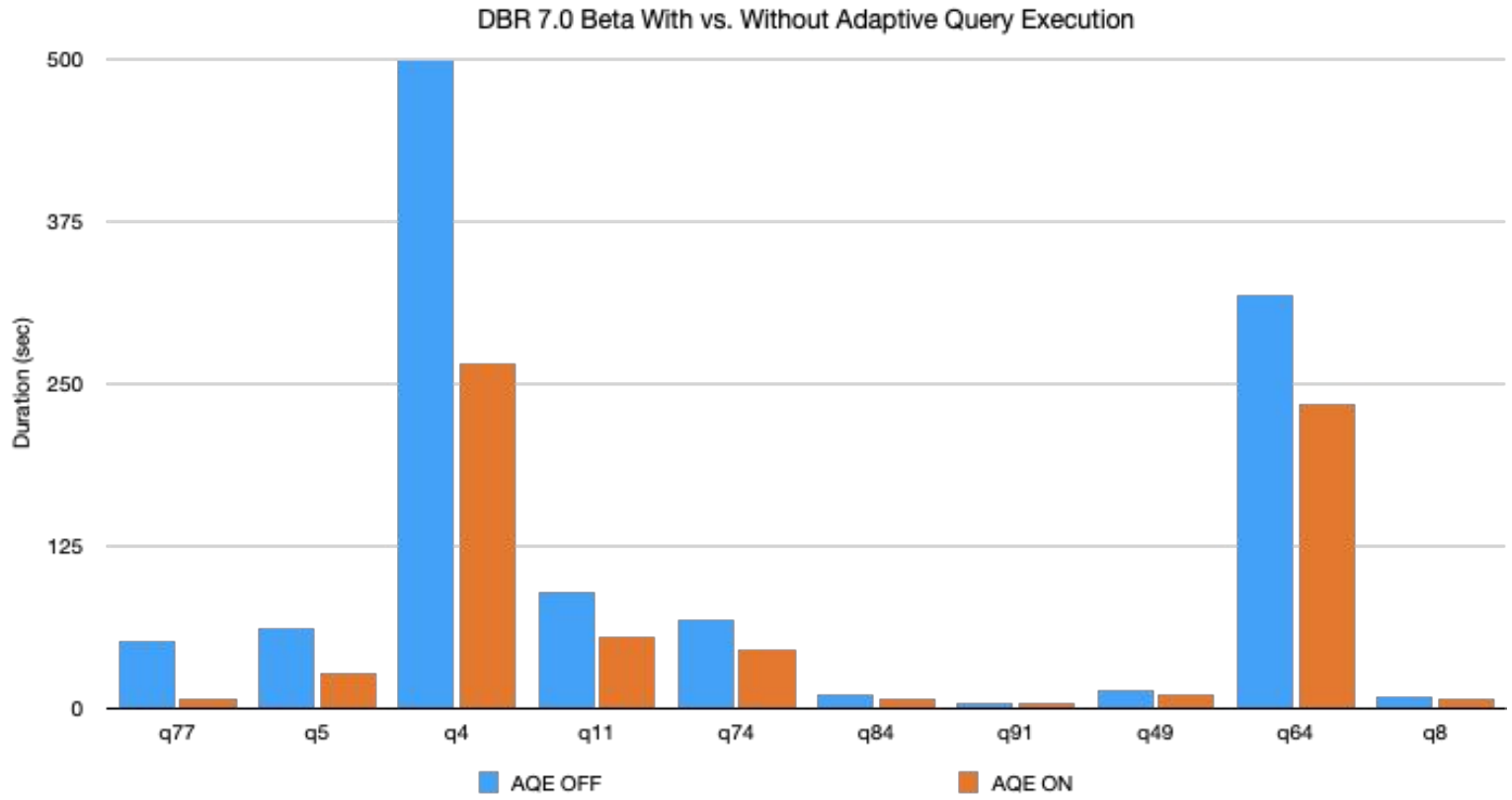
- Tables with no static stats
- Tables with stale stats
- Hard-to-estimate predicates
- Queries with no hints

# What Can AQE Do?

- Join strategy adjustment
- Skew join detection and optimization
- Shuffle partition number adjustment
- Other Optimizations:
  - Empty relation propagation
  - Join reordering (future work)
  - Other logical optimizations that depend on stats (future work)

# Benchmarking AQE on TPC-DS

- 32 queries had more than 1.1x speedup
- Q77 had an 8x speedup



# Join Strategy Dilema

Spark chooses **Broadcast Hash Join** if either child of the join can fit well in memory.

Problem: estimates can go **wrong** and the opportunity of doing BHJ can be **missed**:

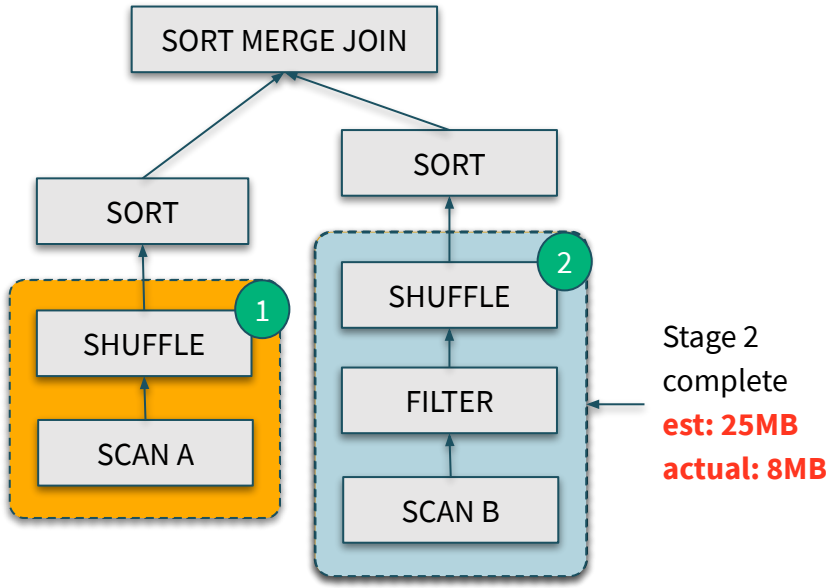
- Stats not qualified for accurate cardinality or selectivity estimate
- Child relation being a complex subtree of operators
- Blackbox predicates, e.g., UDFs

Solution by AQE: replan joins with **runtime** data sizes.

# AQE Join Strategy Adjustment

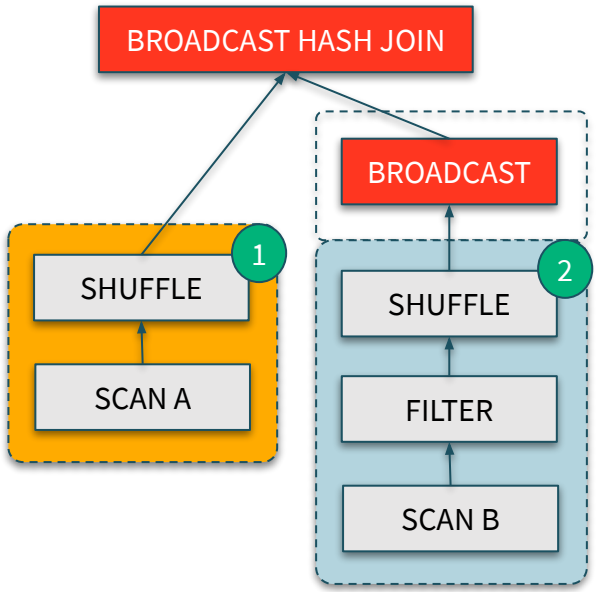
**SELECT** * **FROM** a **JOIN** b **ON** a.key = b.key **WHERE** b.value **LIKE** '%xyz%'

1. Run leaf stages

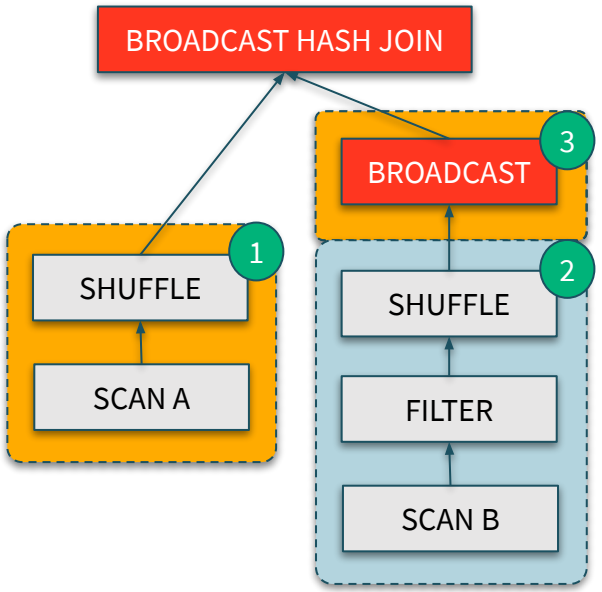



Stage 2 complete
**est: 25MB**
**actual: 8MB**

2. Optimize




3. Run more stages



# Skew Join

Problem: data skew can lead to significant performance **downgrade**

- Individual long running tasks slow down the entire stage
- Especially large partitions lead to more slowdown with disk spilling.

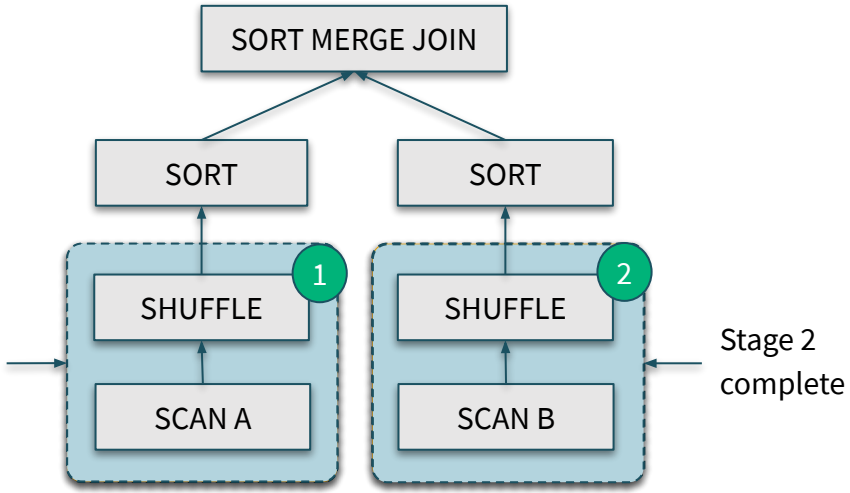
Solution by AQE: handle skew join **automatically** using runtime statistics

- **Detect** skew from partition sizes
- **Split** skew partitions into smaller subpartitions

# AQE Skew Join Optimization

**SELECT** * **FROM** a **JOIN** b **ON** a.col = b.col

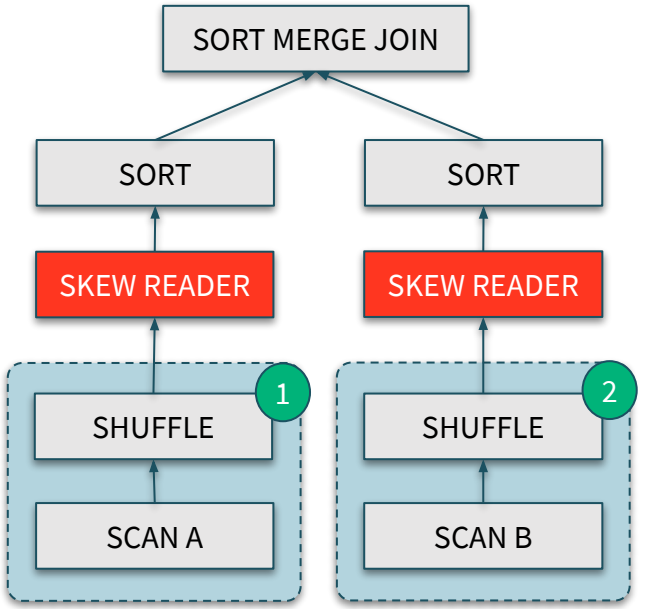
1. Run leaf stages




Stage 1 complete
**med: 55MB**
**min: 40MB**
**max: 250MB**

Stage 2 complete

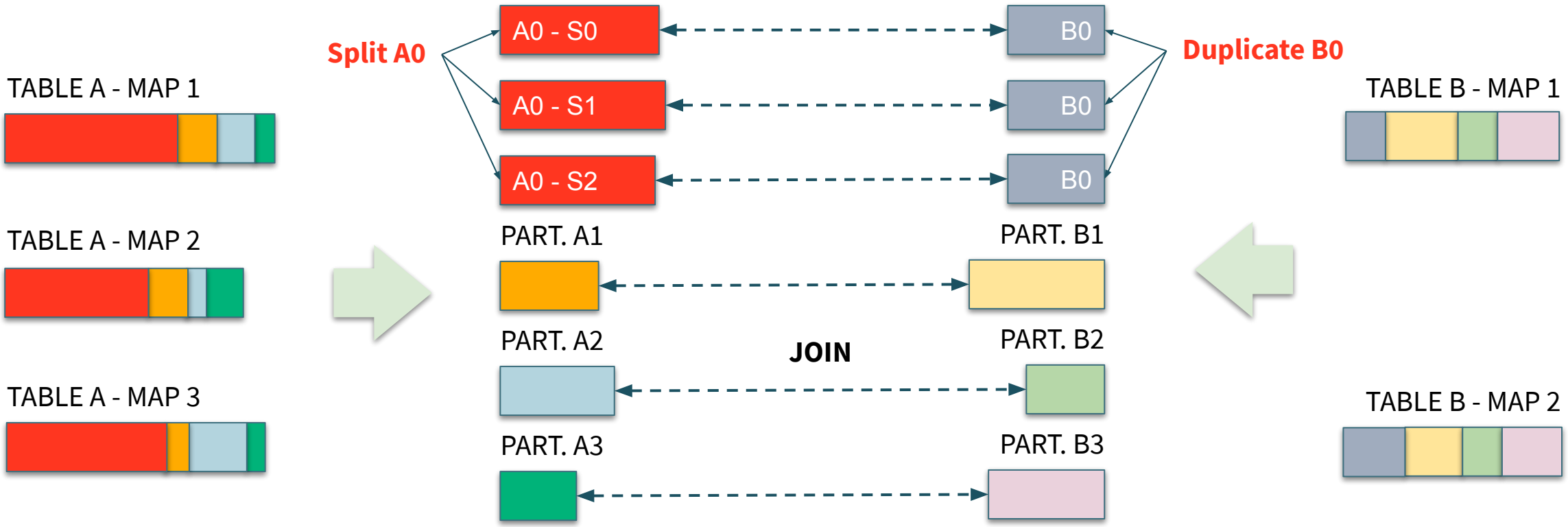
2. Optimize




databricks

# AQE Skew Join Optimization — Details

Skew-optimized sort merge join — with skew shuffle reader:



# Shuffle Partition Number Dilema

Shuffle partition number and sizes **crucial** to query performance

Partition too small

Inefficient I/O

Scheduler overhead

Task setup overhead

Partition too large

GC pressure

Disk spilling

# Shuffle Partition Number Dilema

Problem:

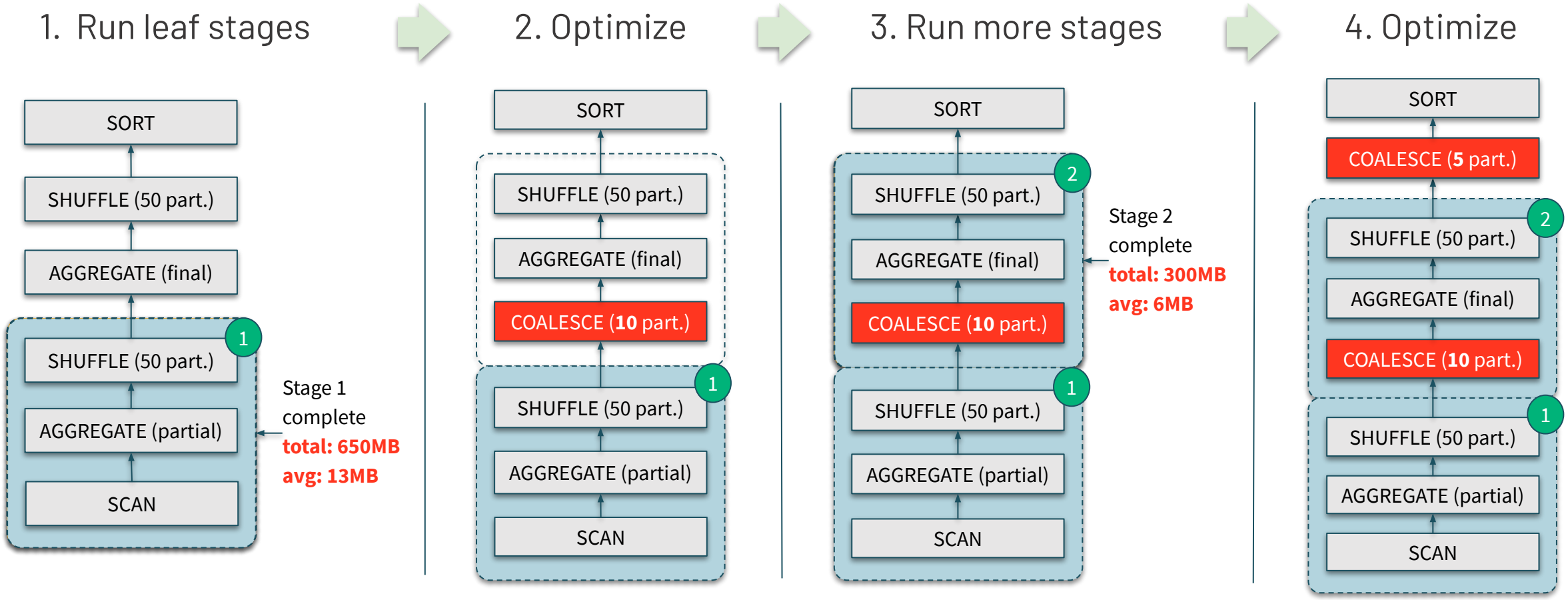
- One **universal** partition number throughout the entire query execution
- Data size **changes** at different times of query execution

Solution by AQE:

- Set the initial partition number high to accommodate the largest data size of the entire query execution
- Automatically **coalesce** partitions if needed after each query stage

# AQE Shuffle Partition Number Adjustment

**SELECT** x, avg(y) **FROM** t **GROUP BY** x **ORDER BY** avg(y)




databricks

# Databricks Auto Optimized Shuffle (AOS)

Problem:

- AQE does NOT increase shuffle partition number
- Need to **manually** pick the right initial partition number

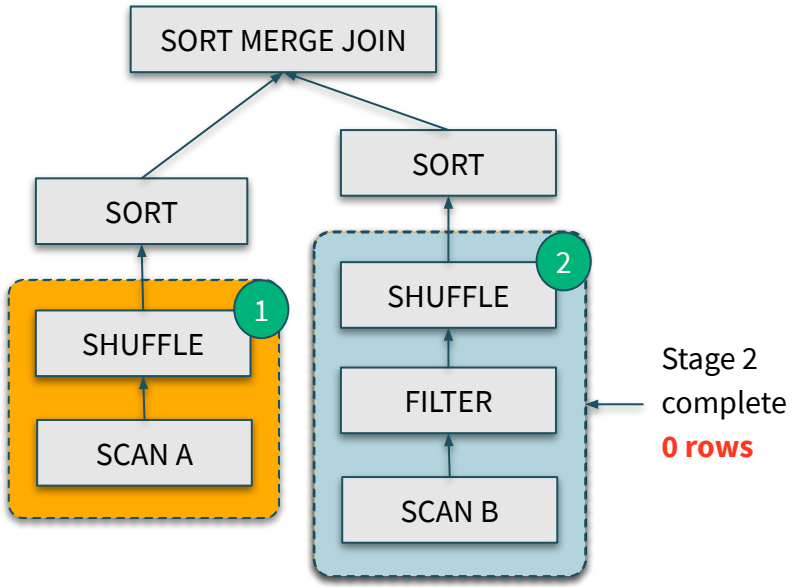
Solution by AOS:

- **Automatically** set the initial shuffle partition number
- Combined with AQE, the shuffle partition number can **expand** and **shrink** based on the data size of each query stage

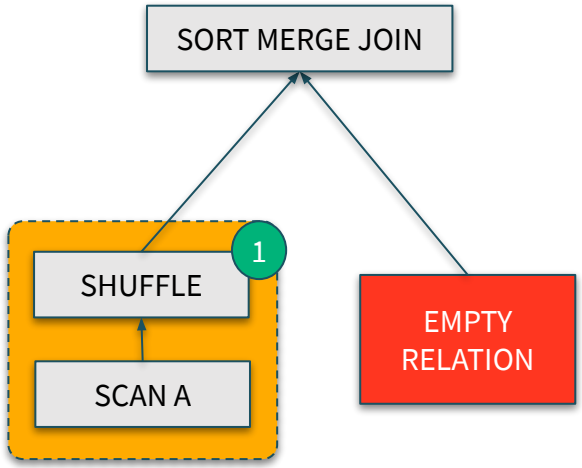
# AQE Empty Relation Propagation

**SELECT** * **FROM** a **JOIN** b **ON** a.key = b.key **WHERE** b.value < 10

1. Run leaf stages




2. Convert empty relation




3. Optimize