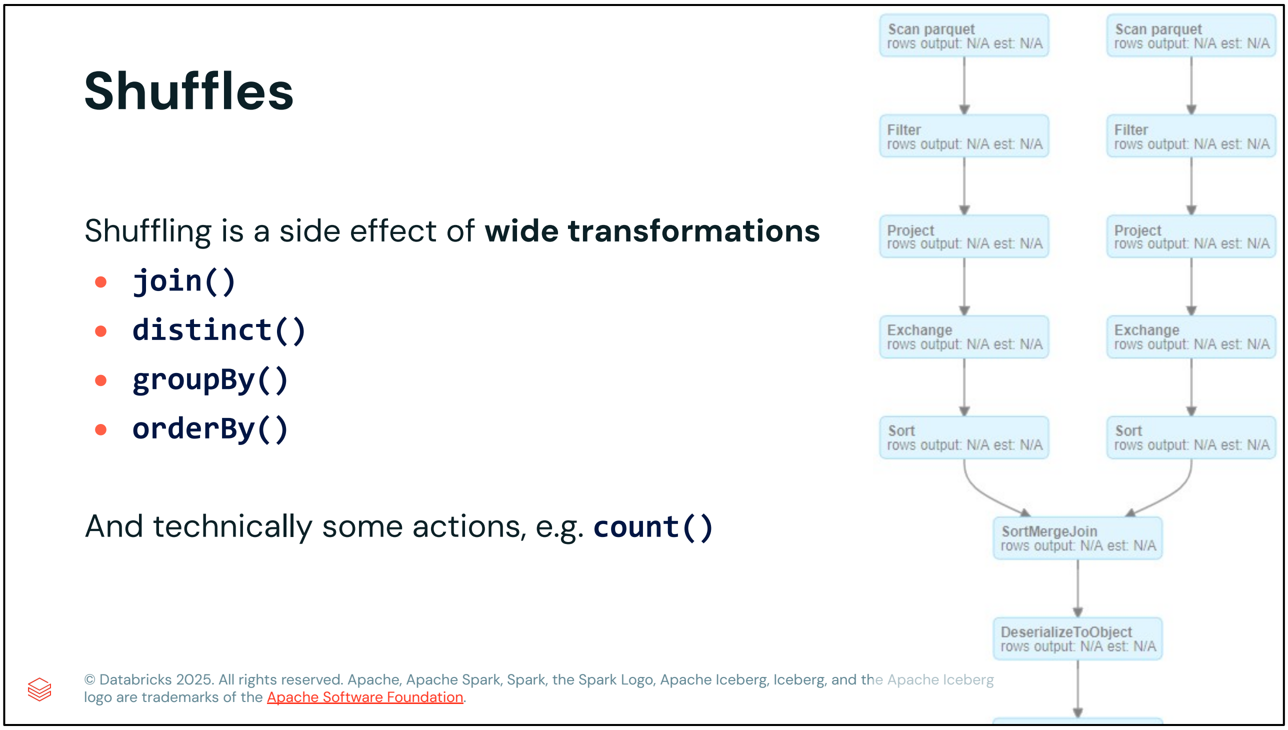
Code Optimization

LECTURE

Shuffles

This lecture explains what shuffles are in Spark, how they work, and strategies to mitigate their impact.



Shuffles

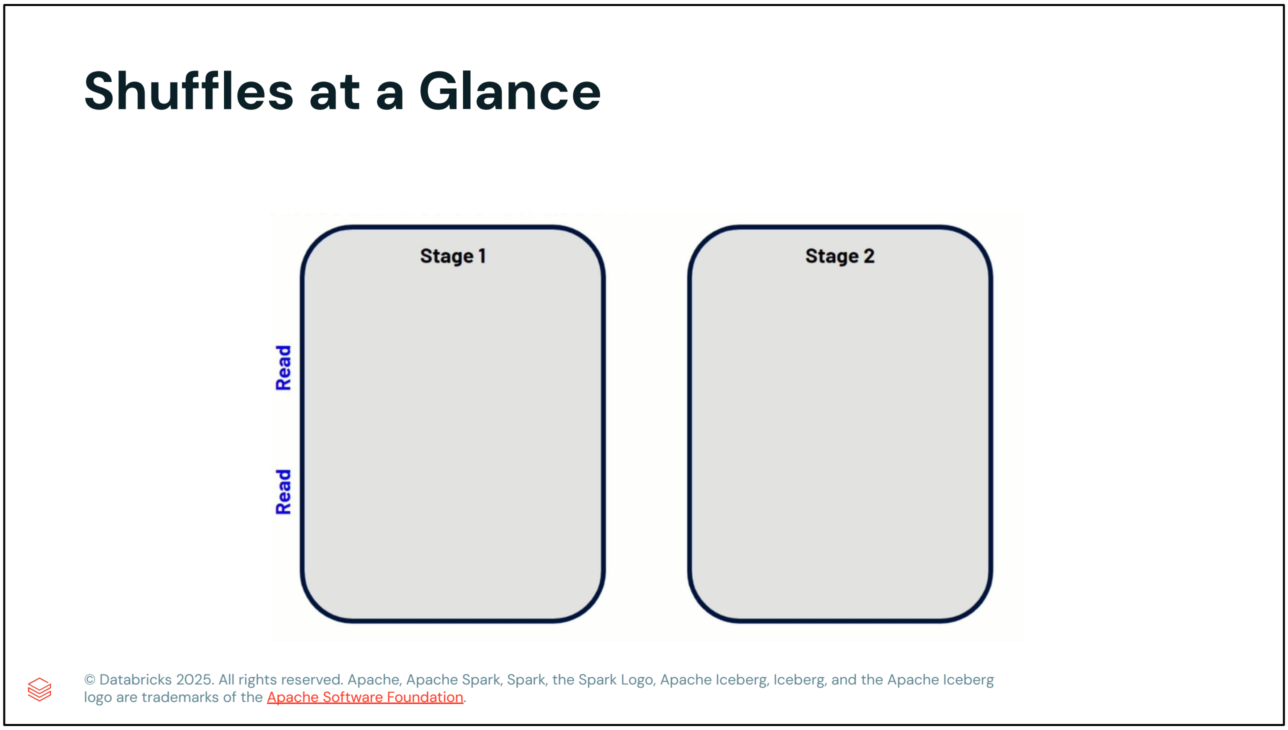
Shuffling is a side effect of **wide transformations**

- `join()`
- `distinct()`
- `groupBy()`
- `orderBy()`

And technically some actions, e.g. **`count()`**

Scan parquet
rows output: N/A est: N/A
Scan parquet
rows output: N/A est: N/A
Filter
rows output: N/A est: N/A
Filter
rows output: N/A est: N/A
Project
rows output: N/A est: N/A
Project
rows output: N/A est: N/A
Exchange
rows output: N/A est: N/A
Exchange
rows output: N/A est: N/A
Sort
rows output: N/A est: N/A
Sort
rows output: N/A est: N/A
SortMergeJoin
rows output: N/A est: N/A
DeserializeToObject
rows output: N/A est: N/A

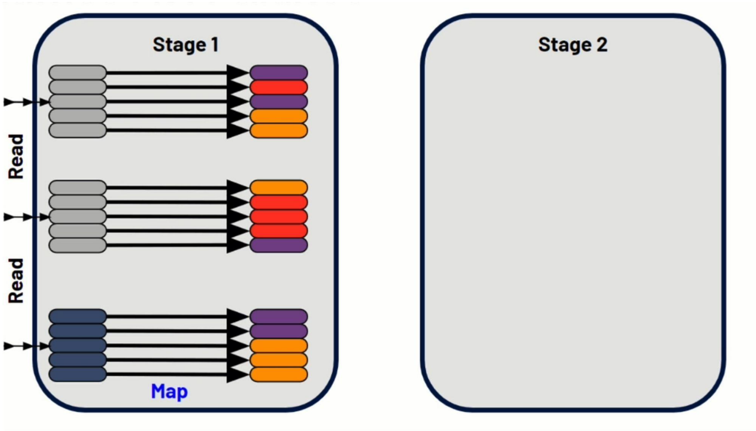
Job: A job in Apache Spark refers to the overall computation that needs to be executed on the data. It comprises one or more stages.
Stage: A stage is a collection of tasks that can be executed together. Stages are formed based on the transformations applied to the data, and they represent a unit of work.
Task: A task is the smallest unit of work in Spark. Each task performs an identical operation across a partition of the data.
Wide Transformation: A wide transformation is an operation that requires two stages to complete. It often involves shuffling, which is the process of redistributing data across partitions. Examples of wide transformations include **join(), distinct(), groupBy(), orderBy(),** and some actions like **count()**.
Narrow Transformation: A narrow transformation is an operation that requires only one stage to complete. Unlike wide transformations, narrow transformations do not involve shuffling.
Shuffle: Shuffling is the act of moving data from the output of one stage to the input of another. It is a side effect of wide transformations and is a critical operation that involves redistributing and reorganizing data.



Shuffles at a Glance

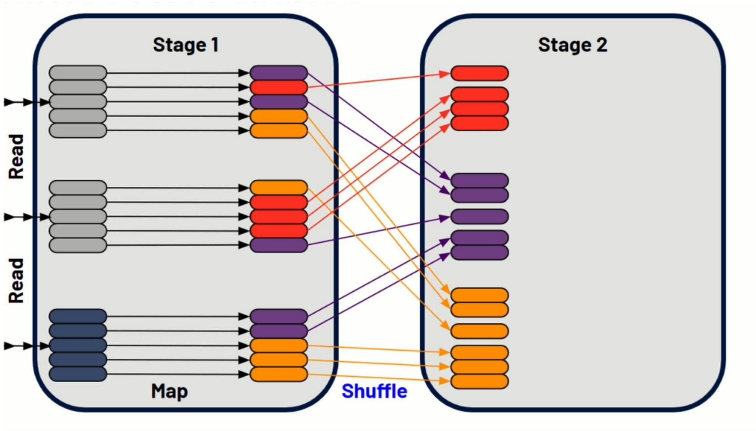
So, what is a shuffle? Here's a two-stage job that's required, where one stage needs to be completed and data generated at the end of the stage, with some transformation or something that's happening. Stage two will happen after stage one has completed. Now, we're going to look at MapReduce and how this can be applied through these two stages.

Shuffles at a Glance

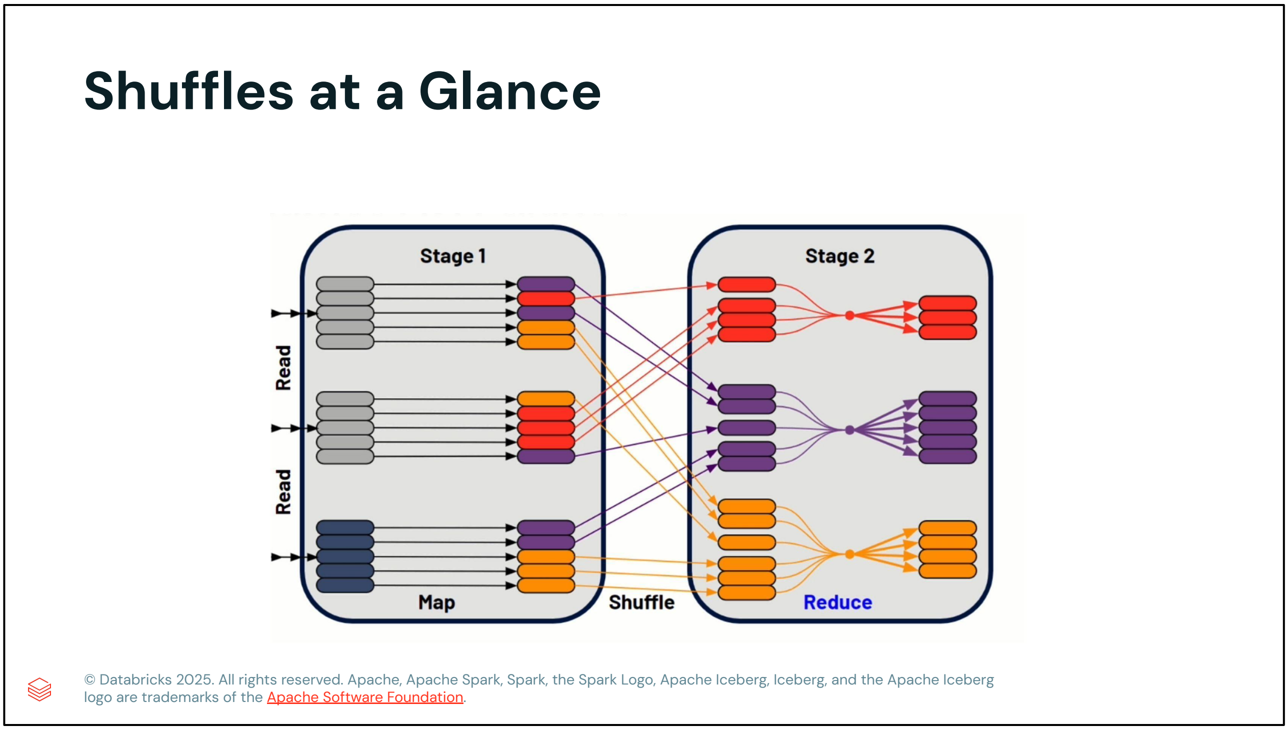


So first, the data is read into stage one. We map and perform the map transformation, and then we have data that needs to be moved into the reducer.

Shuffles at a Glance



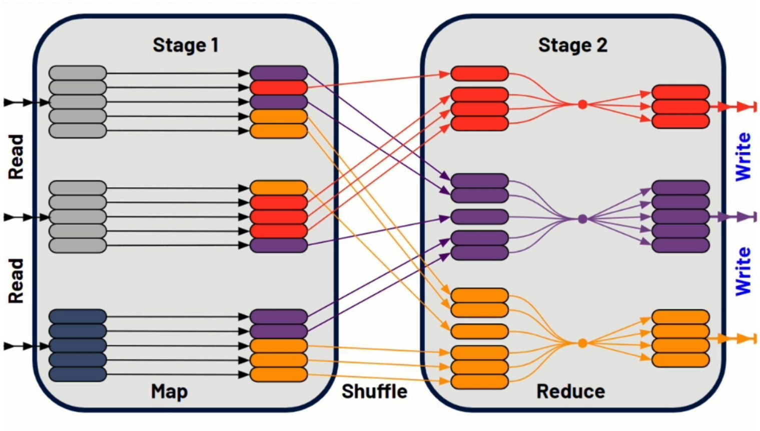
So now we shuffle because that map has caused us to have data that needs to be moved into stage two based on that specific mapping. This shuffles data all over the place and represents network movement from one worker to another. As these workers perform specific tasks, we have to move things around the network at the cloud service provider, and that's a shuffle.



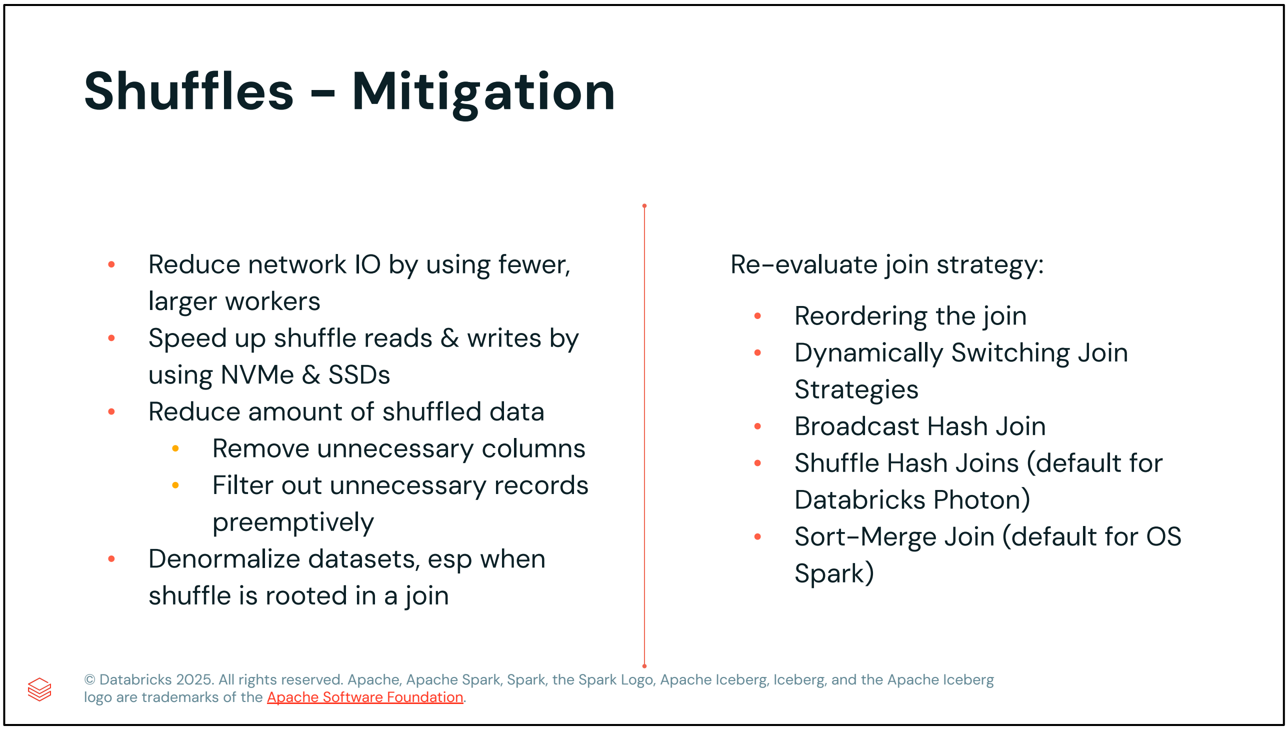
Shuffles at a Glance

Right, so then we have the reduce, and we end up with our outputted data.

Shuffles at a Glance



Our outputted data then needs to be written to some DataFrame or a table, whatever it might be. We have stage one and stage two that completes, and in between, we have the shuffle. This is a result of the aspect of data being assigned based on the map occurring in stage one and then being reorganized so that it can be pushed through the reducer function.



Shuffles – Mitigation

- Reduce network IO by using fewer, larger workers
- Speed up shuffle reads & writes by using NVMe & SSDs
- Reduce amount of shuffled data
  - Remove unnecessary columns
  - Filter out unnecessary records preemptively
- Denormalize datasets, esp when shuffle is rooted in a join

Re-evaluate join strategy:

- Reordering the join
- Dynamically Switching Join Strategies
- Broadcast Hash Join
- Shuffle Hash Joins (default for Databricks Photon)
- Sort-Merge Join (default for OS Spark)

By using fewer and larger VMs (e.g. more cores), we still pay the cost for disk io, but reduce network io

AQE & DPP should be making it unnecessary to denormalize datasets as Spark moves forward. But it is a viable strategy outside of Spark 3

Bucketing

- “If you are bucketing datasets, you are doing it wrong” - DT
- Bucketing is hard to get right and is an expensive operation to being with… especially if you are bucketing a periodically changing dataset
- Eliminates the sort in the Sort-Merge Join by pre-sorting partitions
- The cost is paid in production of the dataset on the assumption that savings will be made by frequent joins of both tables
- Not worth considering for datasets less than 1-5 TBs
- DT = Daniel Tomes, from one of his presentations are Spark Summit

Thank you for completing this lesson and continuing your journey to develop your skills with us.