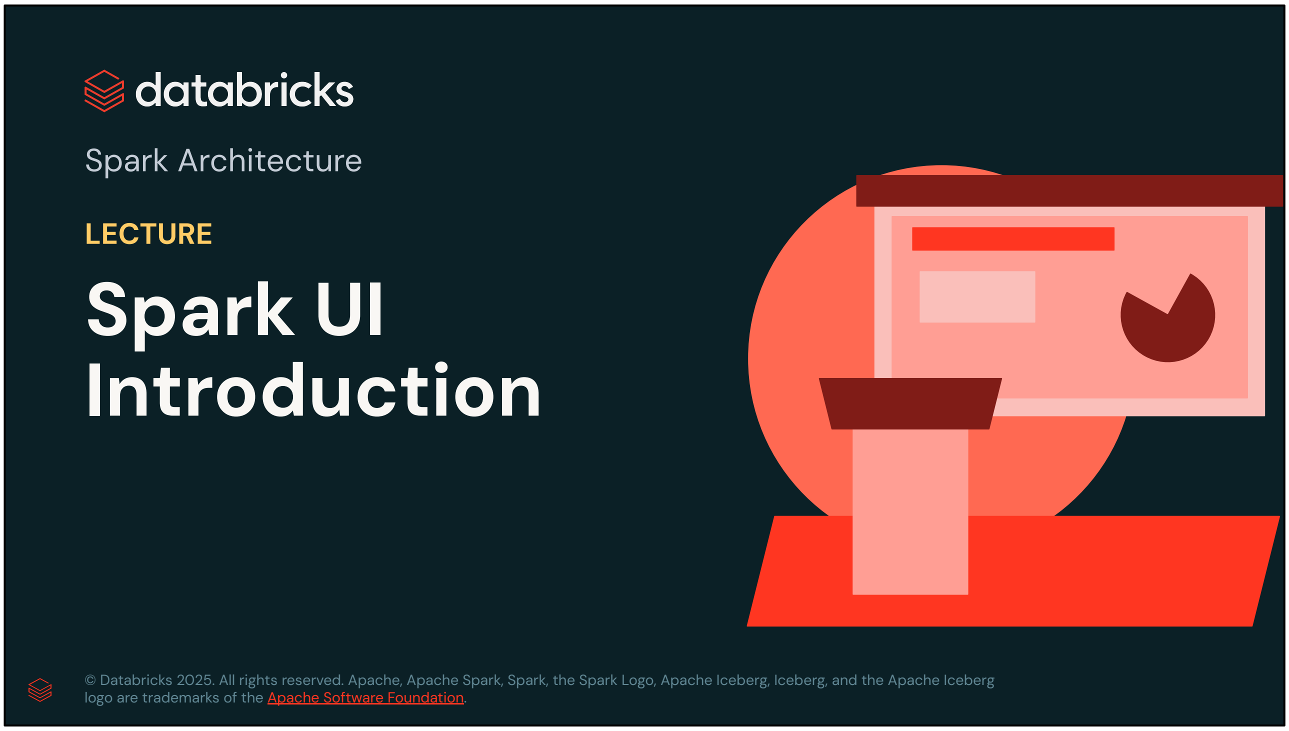


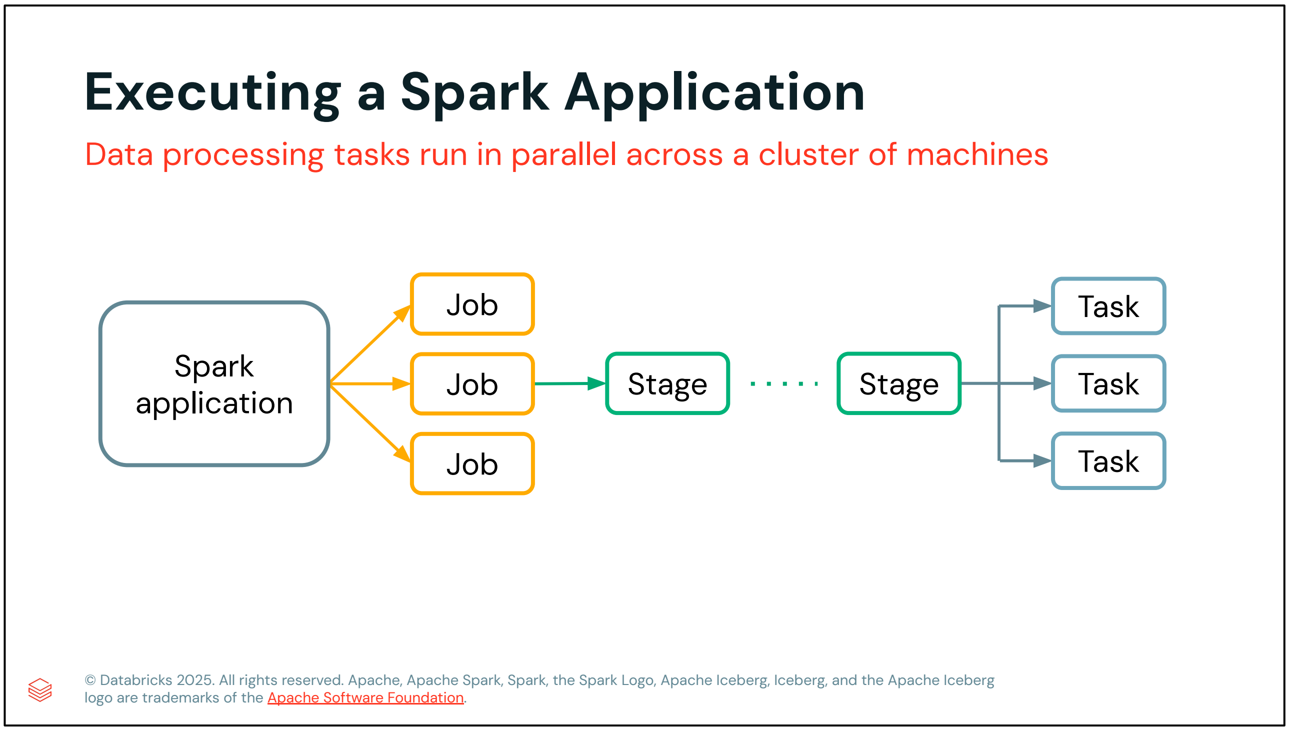
databricks

Spark Architecture

LECTURE

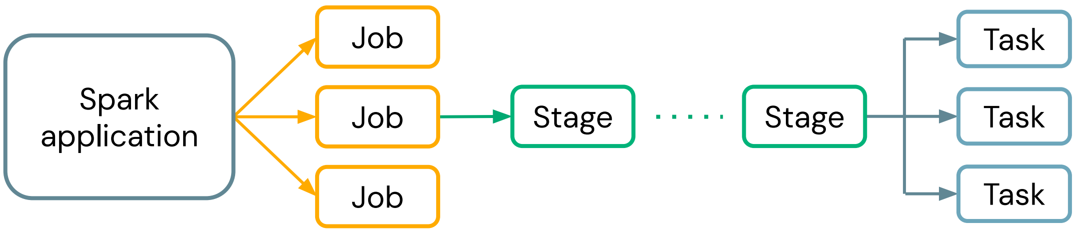
Spark UI Introduction

In this lecture, we will explore the Spark UI, its architecture, and optimization techniques to enhance query and code performance.



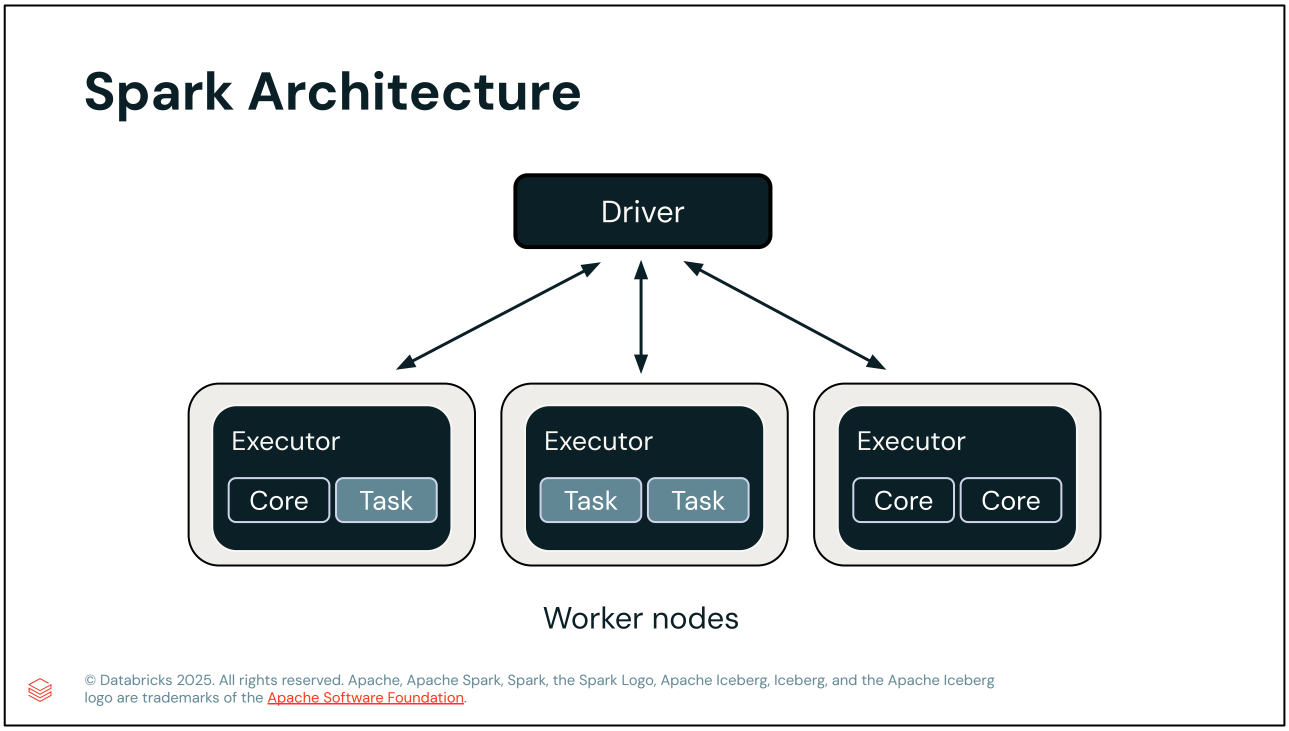
Executing a Spark Application

Data processing tasks run in parallel across a cluster of machines



Spark uses clusters of machines to process big data by breaking a large task into smaller ones and distributing the work among several machines.
Let's look at how spark executes a spark application.

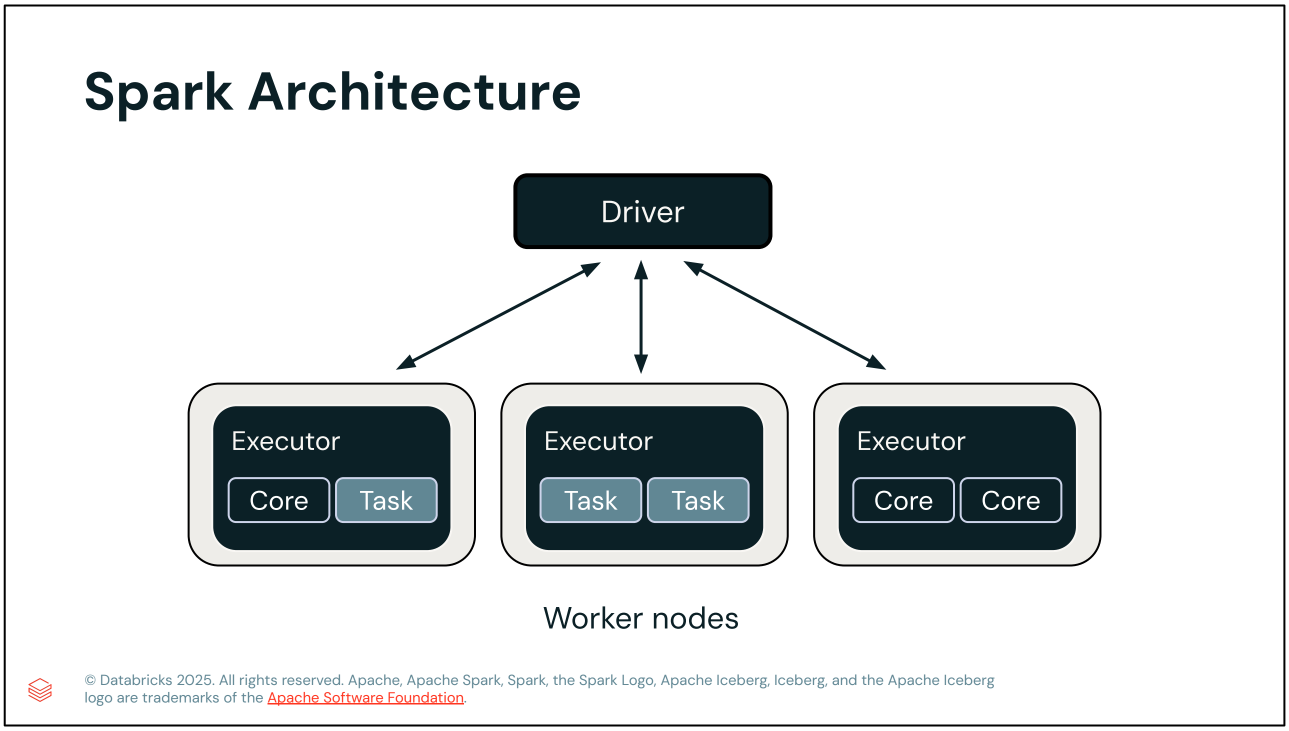
1. Jobs
   - The secret to Spark's performances is parallelism. Each parallelized action is referred to as a job. Each job is broken down into stages.
2. Stages
   - Each job is broken down into stages, which is a set of ordered steps that, together, accomplish a job.
3. Tasks
   - Tasks are created by the driver and assigned a partition of data to process. These are the smallest unit of work.



Spark Architecture

Let's go over each of the items in the diagram below to identify the components that Spark uses to coordinate work across a cluster of computers.
At a high level, the workflow goes something like: client runs query. Driver builds marching orders for bite-size Memory Partitions that Executor's Cores then creates and runs in Parallel as Tasks (units of work)

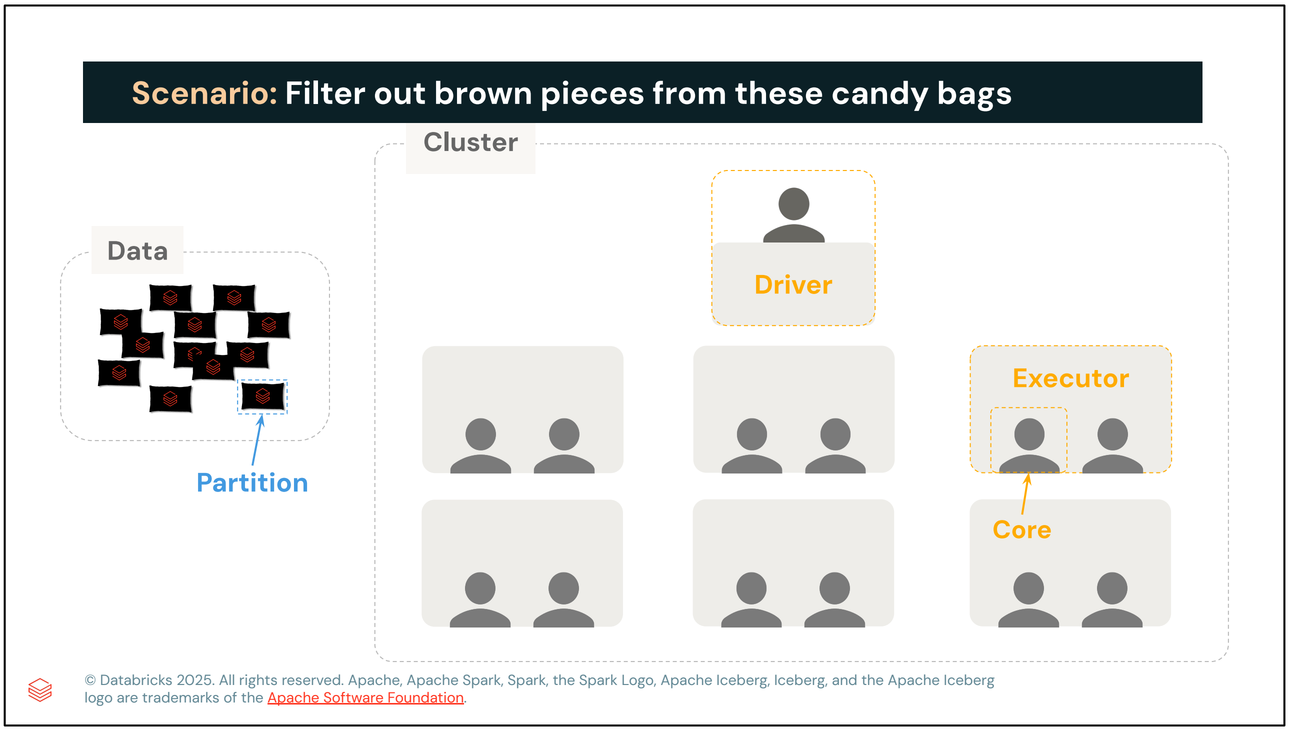
- Driver
  - The driver is the machine in which the application runs. It is responsible for three main things
    - maintaining information about the Spark Application
    - responding to the user's program
    - analyzing, distributing, and scheduling work across the executors.
  - In a single Databricks cluster, there will only be one driver, regardless of the number of executors.
- Worker node
  - A worker node hosts the Executor process. It has a fixed number of Executors allocated at any point in time.
- Executors
  - Each executor will hold a chunk of the data to be processed. This chunk is called a Spark partition. It is a collection of rows that sits on one physical machine in the cluster.
    - Note: This is completely separate from hard disk partitions, which have to do with the storage space on a hard drive.



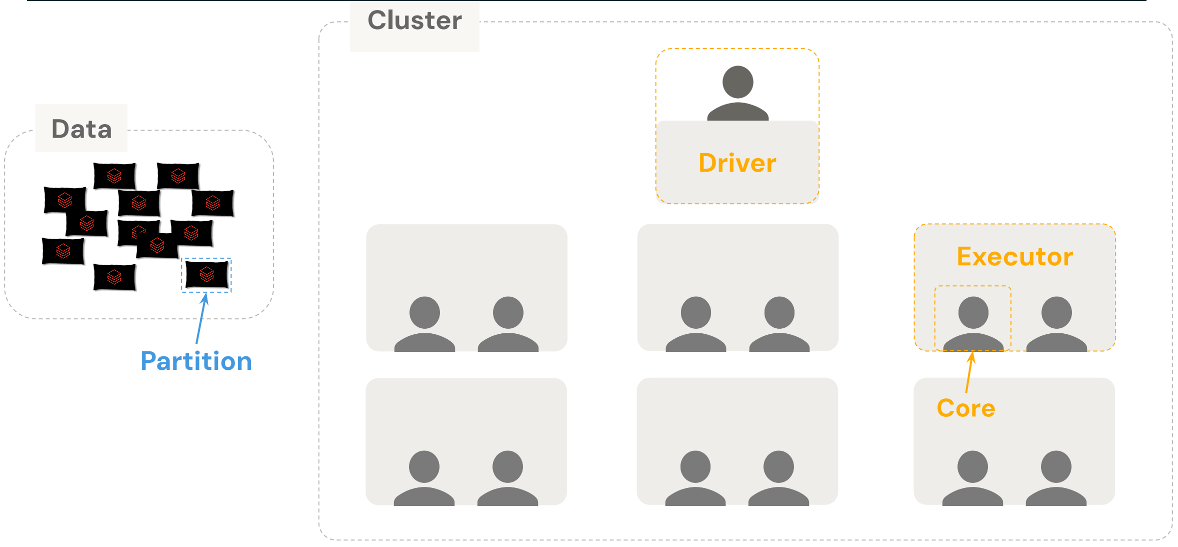
Spark Architecture

    - Executors are responsible for carrying out the work assigned by the driver. Each executor is responsible for two things:
        - Execute code assigned by the driver
        - Report the state of the computation back to the driver
- Cores (also known as slots or threads)
    - Spark parallelizes at two levels. One is splitting the work among executors. The other is the slot. Each executor has a number of slots. Each slot can be assigned a task.
- In this diagram, some slots have been filled by tasks and some slots are ope

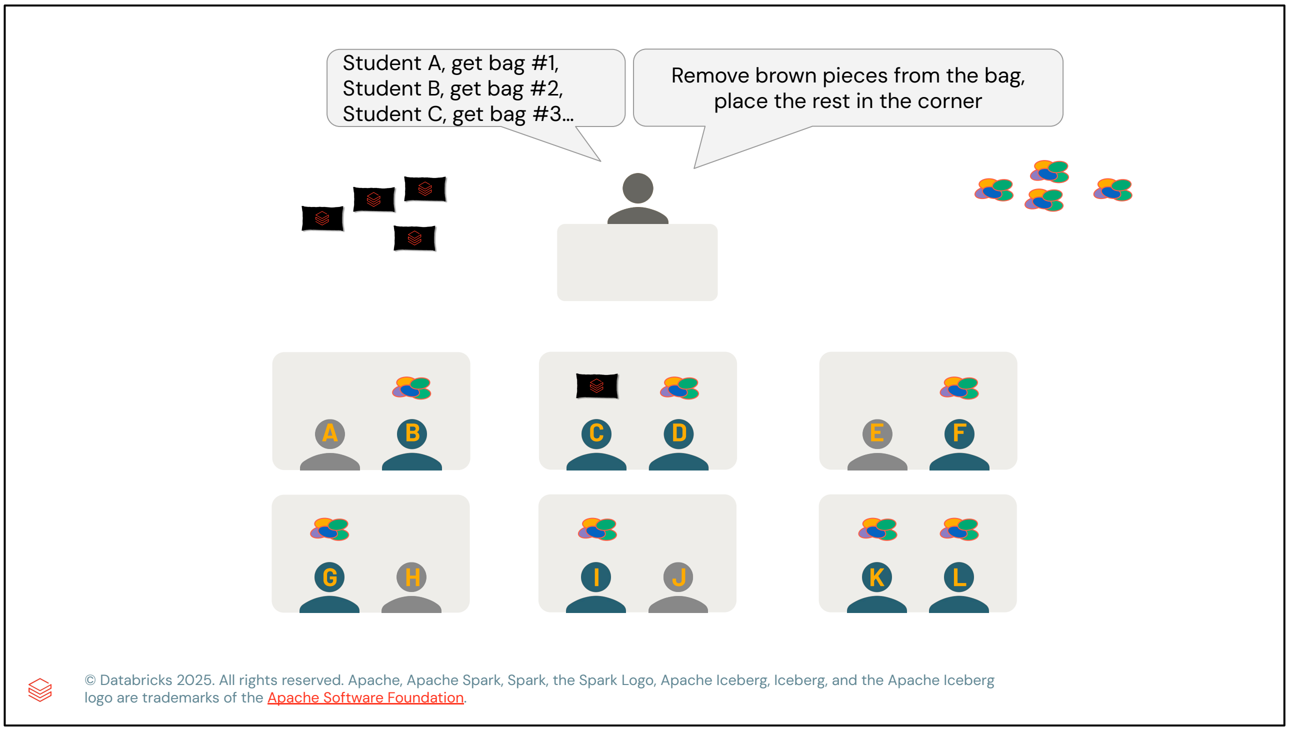
Clusters consist of a set of one or more virtual machine instances, over which computation workloads are distributed. In the typical case, a cluster has a driver node alongside one or more worker nodes, although Databricks provides a single-node model as well (typically limited to development or testing with small workloads). Workloads are distributed across available worker nodes by the driver.
While the abstractions and interfaces are simple, managing clusters of computers and ensuring production-level stability is not. Databricks makes big data simple by providing Apache Spark as a hosted solution, where much of your Spark setup will be managed for you. However, understanding basic Spark architecture and execution concepts is essential to Spark development, as you will see later in the course.



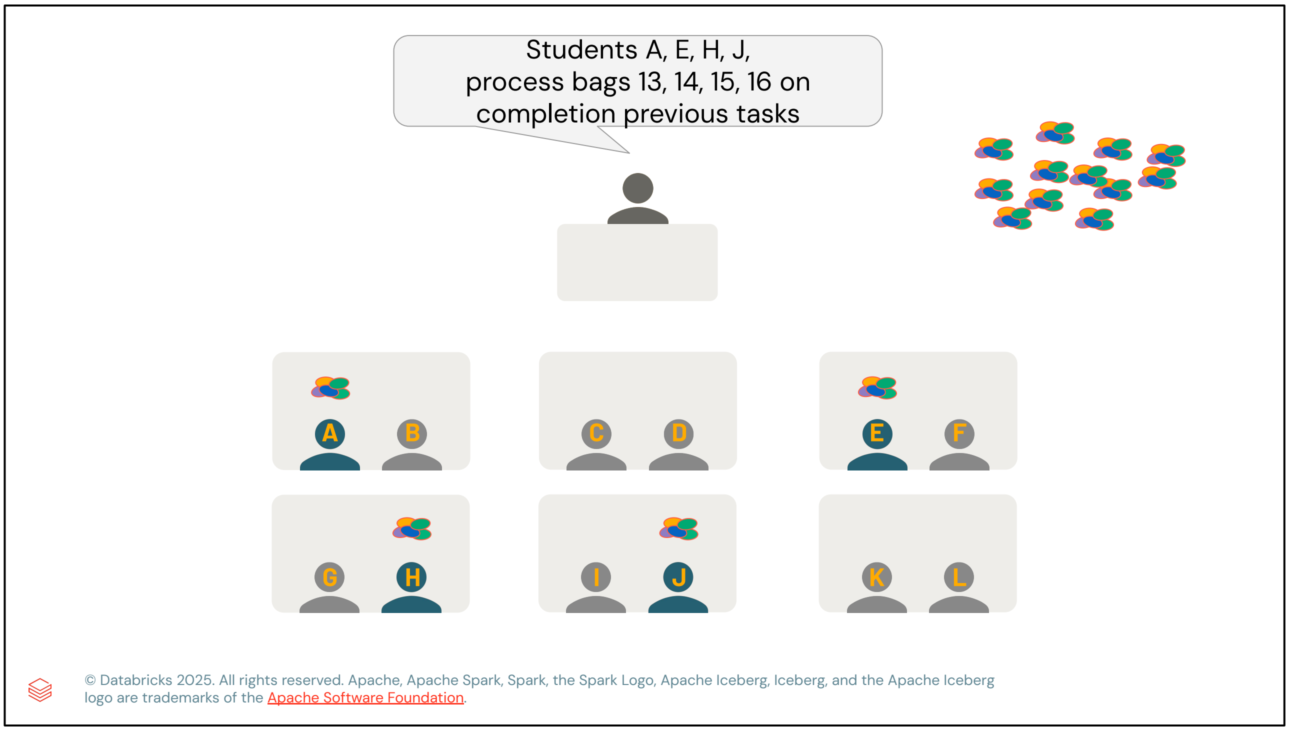
Scenario: Filter out brown pieces from these candy bags



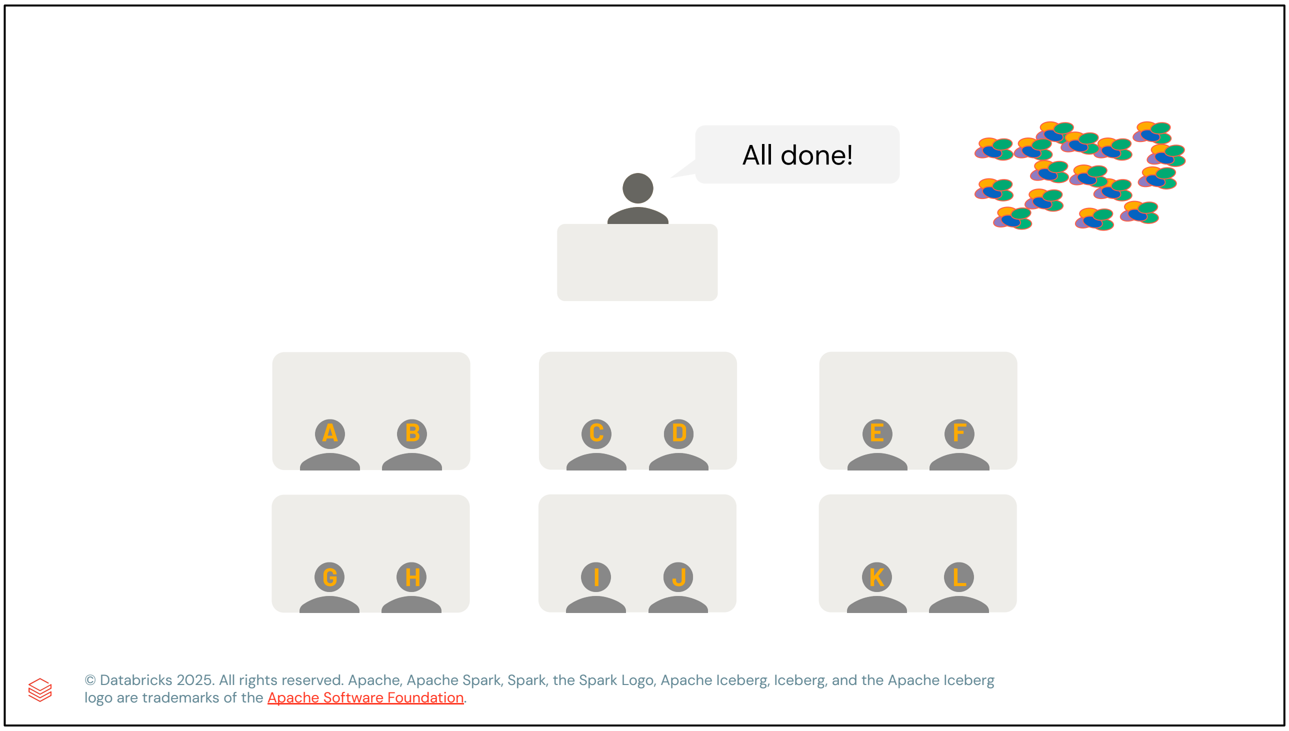
To describe Spark architecture concepts, we are going to use a classroom analogy. A class is given a pile candy bags, and is asked to filter out brown candy pieces from the bags. The class, including the teacher and students, is the cluster. The teacher is the driver that provides instructions and collects results. Each table of students is an executor and each student is a core. Students at the same table share resources like paper and pencils, and each student is given an individual task.
In this scenario, the pile of candy bags is our dataset, and each candy bag is a partition of data. Each candy piece is a record in our dataset.



Think of the classroom as a cluster, where the teacher represents the driver. The teacher has a big list of assignments (tasks) that need to be done. Instead of doing all the work themselves, the teacher decides what needs to get done and then distributes these assignments to different groups of students (executors). Each student within a group is like a core—an individual worker who will complete a part of the assignment. The groups of students receive the instructions from the teacher, and then each student in the group works on their allotted task. This way, the workload is divided and completed efficiently, with the teacher coordinating and the students executing the actual work.

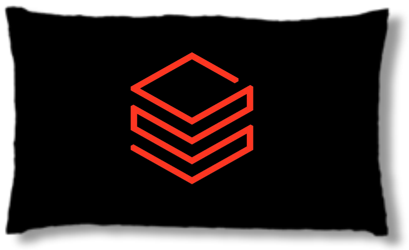


Each group of students (executors) uses their members (cores) to filter the brown candies. In the end, all brown candies are separated out.

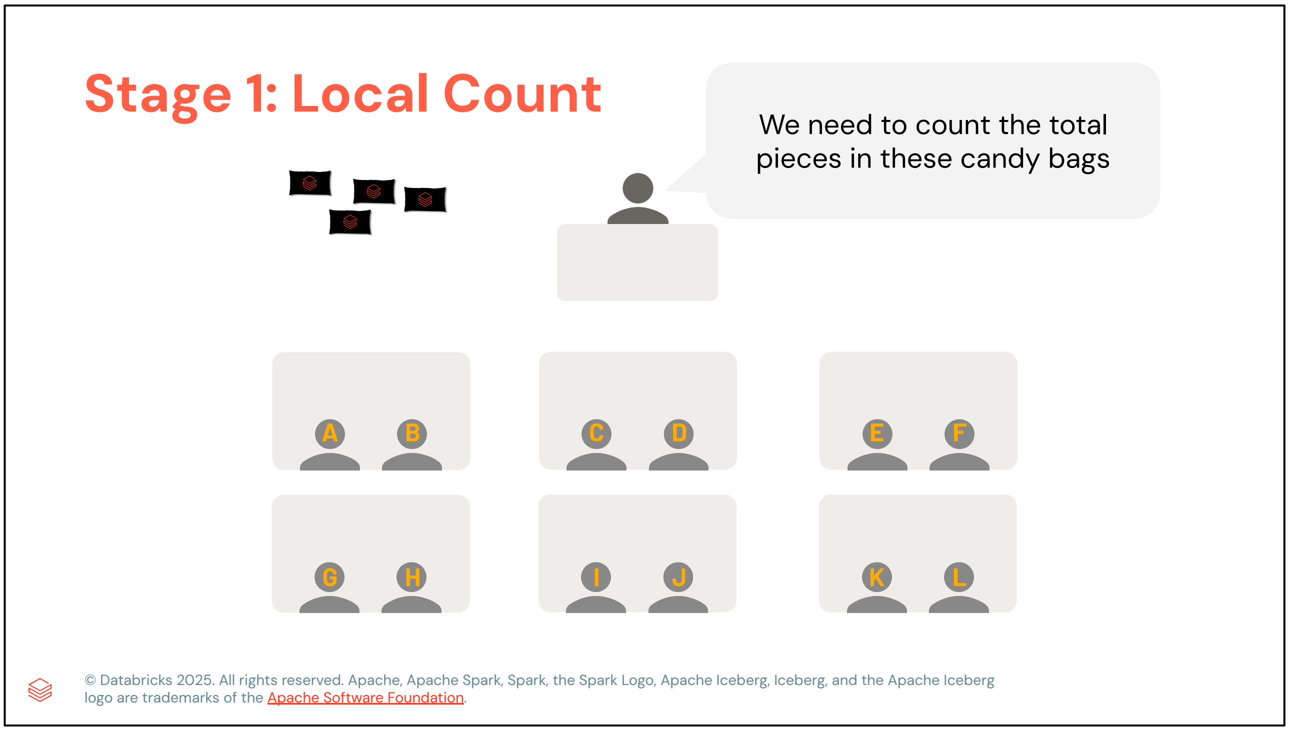


Yes, this was done in parallel—each student (core) worked on their own packet of candy at the same time. When finished, they reported back with the candies that were left after filtering out the brown ones.

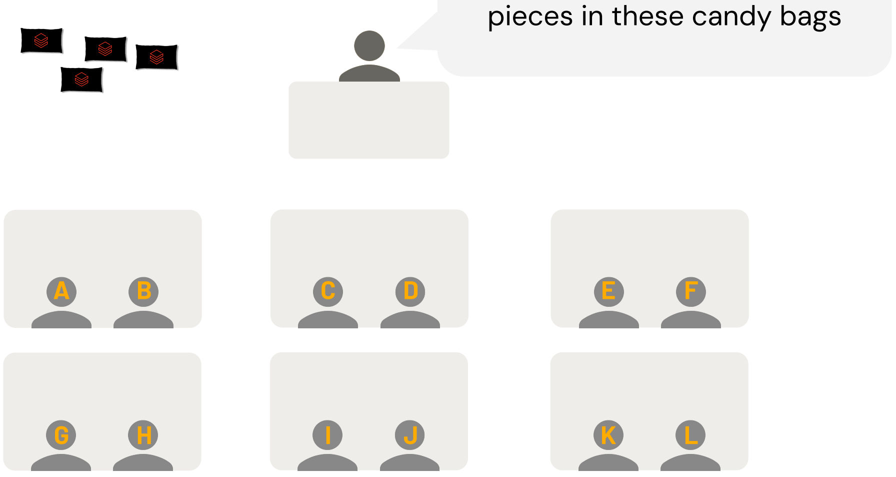
Scenario 2: Count Total Pieces in Candy Bags Introducing Stages



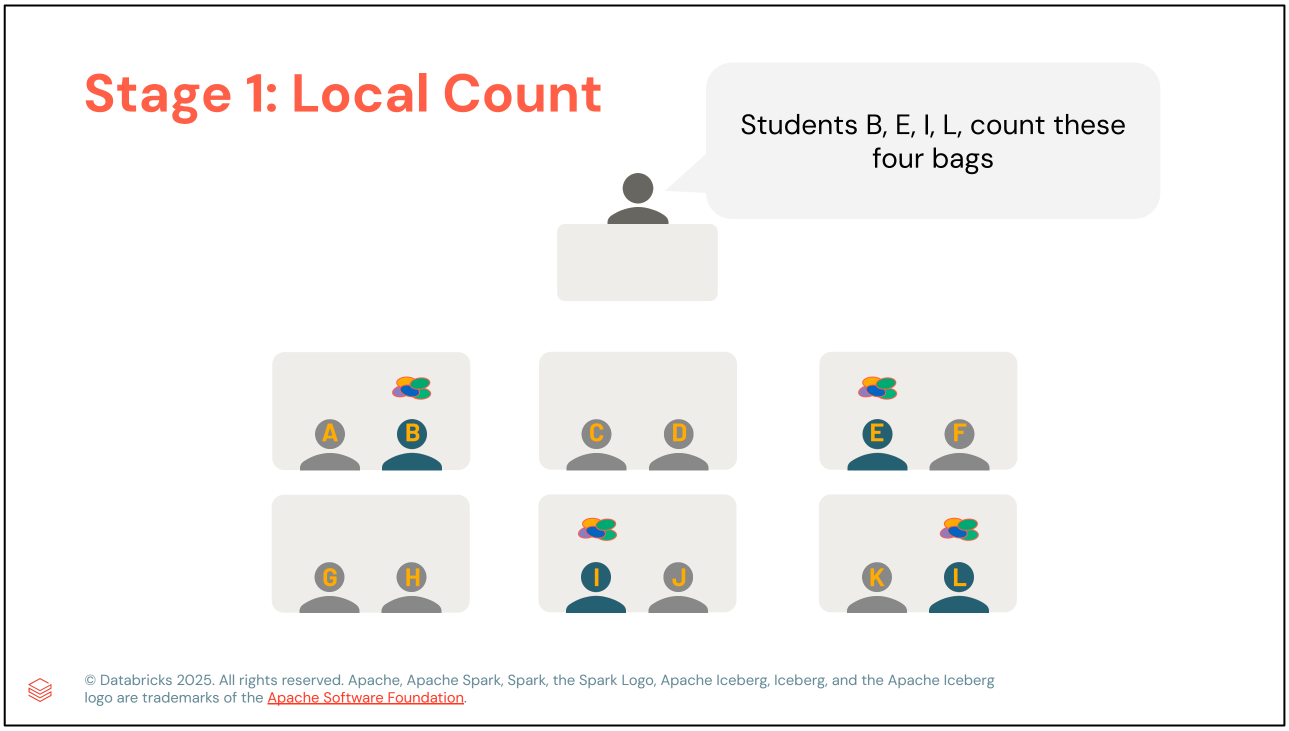
If we want to count the total number of candies in all the bags, we need an order to the steps. First, each student (core) counts their candies, and then those results are combined. This process happens in stages, where one task must finish before the next can begin. That’s what a stage means in Spark.



Stage 1: Local Count

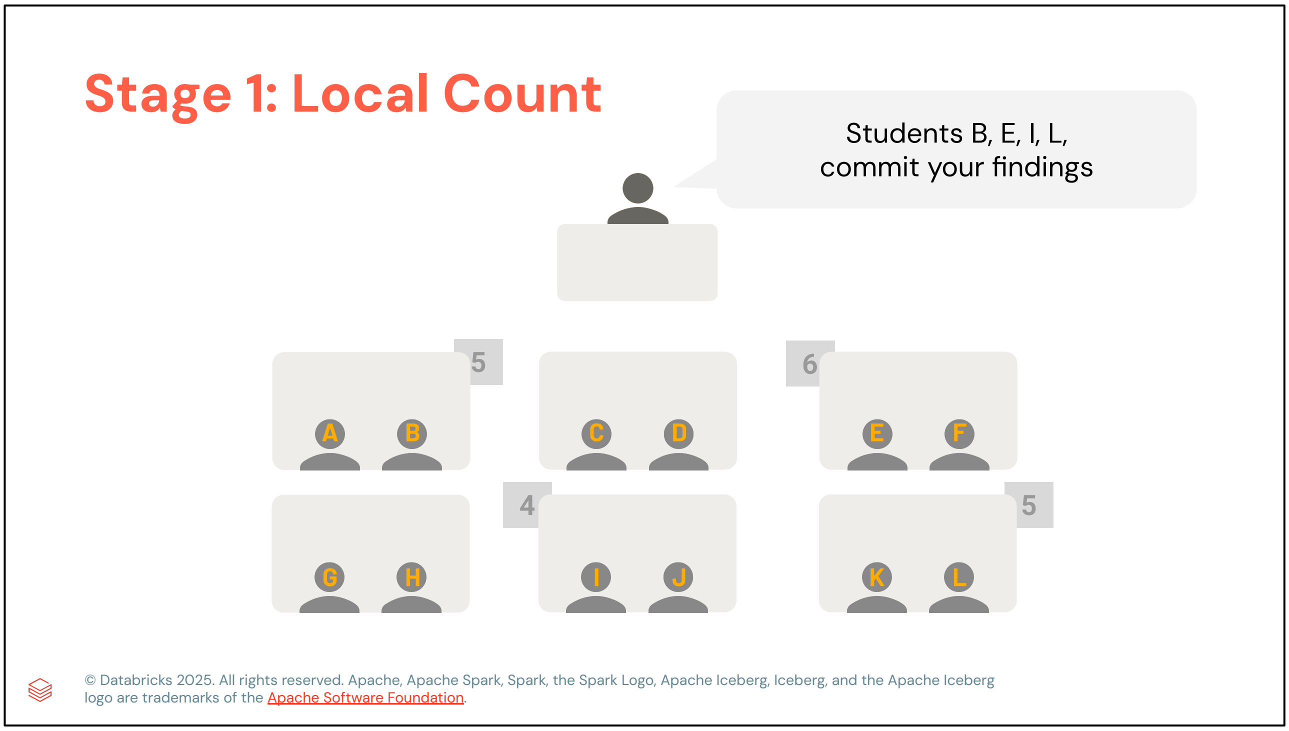


Now, each group of students (executors) counts their own candies and reports the total to the teacher (driver), who adds them all together to get the final count for the entire class.



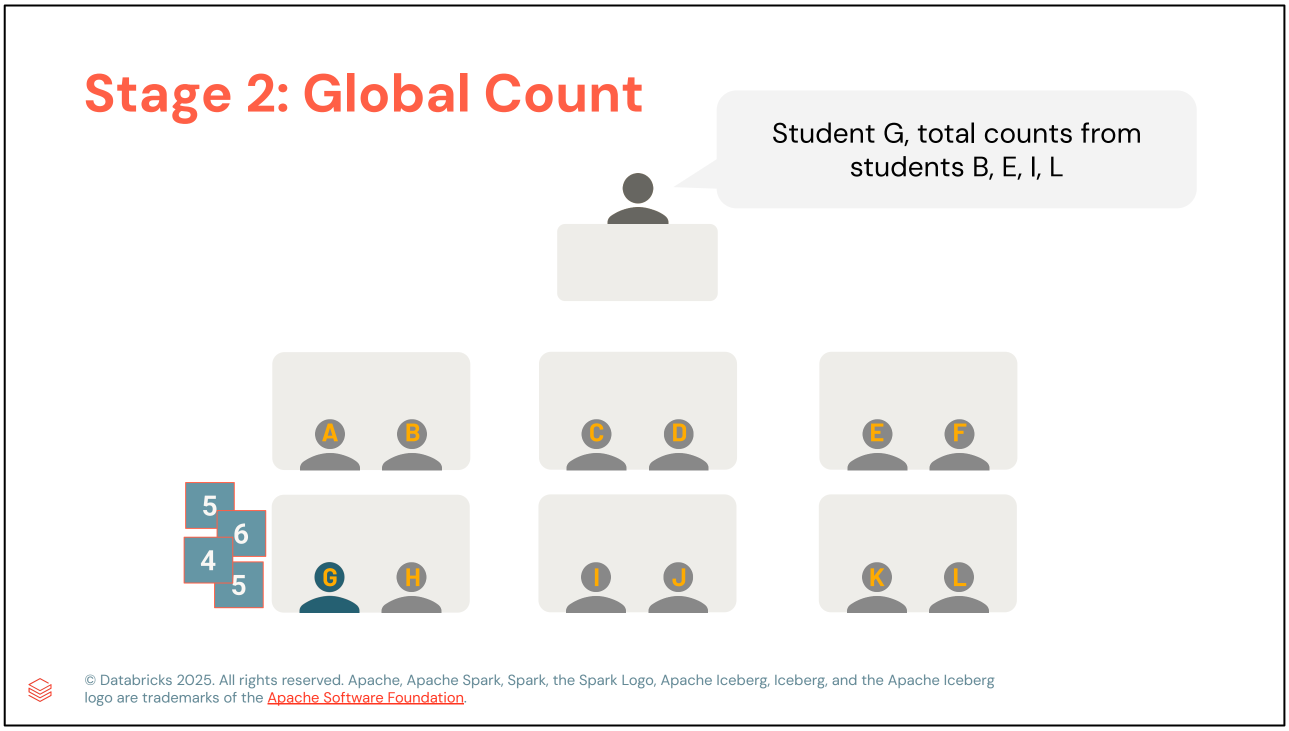
Stage 1: Local Count

The teacher (driver) gives out candy bags and asks students (cores) to count.



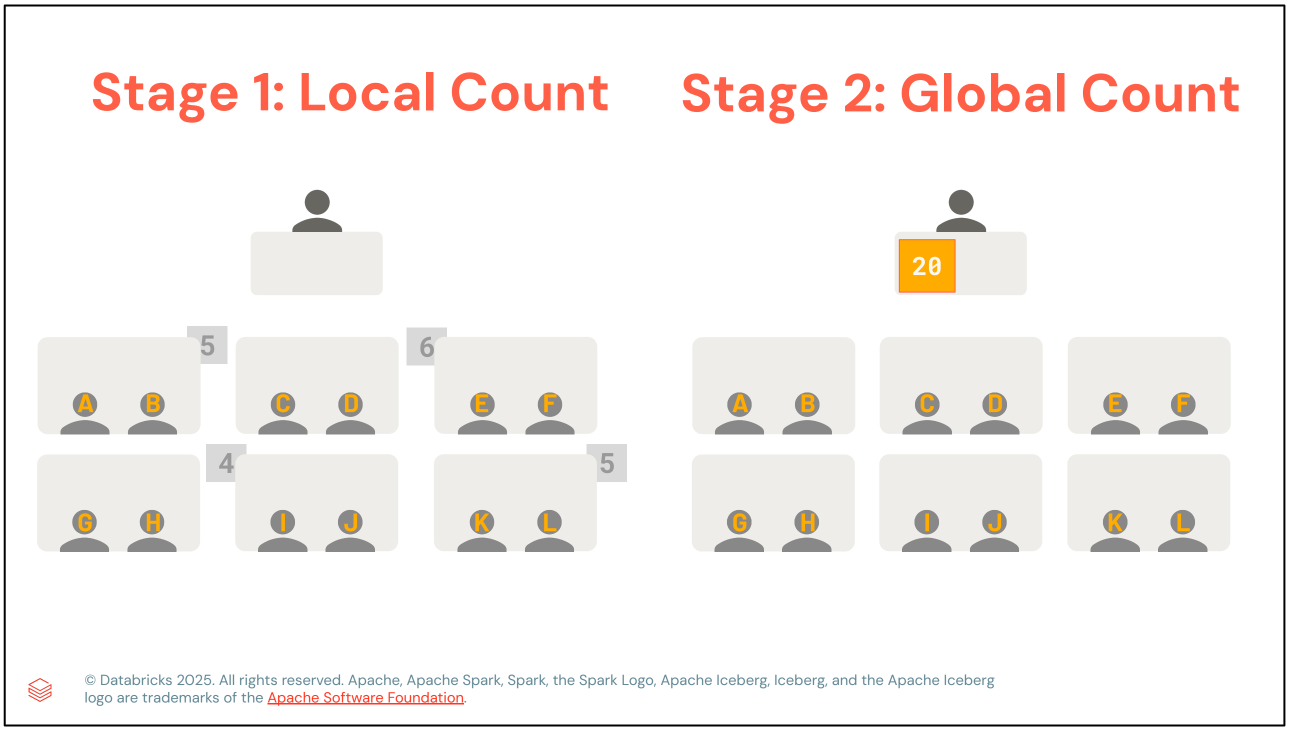
Stage 1: Local Count

This exchange of data between the executors is commonly referred to as a “shuffle” in Spark. In this example, the only data shuffled are the partial counts, which are only a few bytes each. On the other hand, a transformation like distinct() or orderBy() can result in the shuffling of the entire data set, which is substantially more data to transmit.



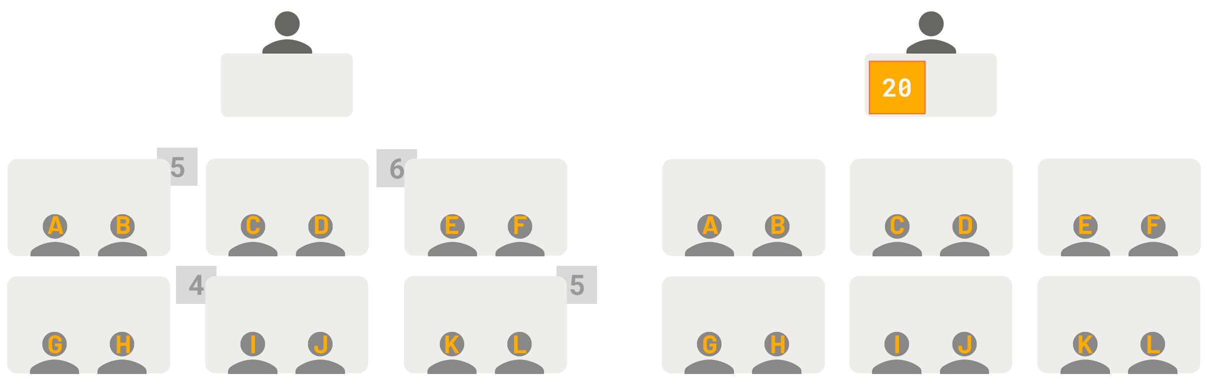
Stage 2: Global Count

Each group (executor) first reports their local candy count. Then, in the next stage, the teacher (driver) adds up all the local counts to get the total number of candies. The global count can only be done after all the local counts are complete.



Stage 1: Local Count

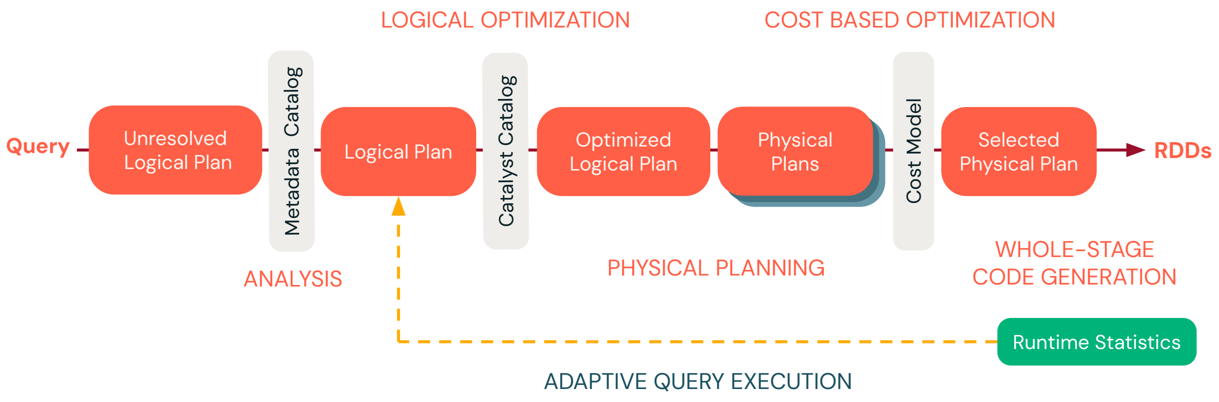
Stage 2: Global Count



Stage one is where each group counts their own candies (local count). Stage two is where the teacher adds all those numbers for the total count (global count). These stages happen one after the other—stage two only starts after stage one is done.

Query Optimization

Enabled by default as of Spark 3.2



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

- Dynamically switch join strategies
- Dynamically coalesce shuffle partitions
- Dynamically optimize skew joins

New in Spark 3.0, enabled by default as of Spark 3.2

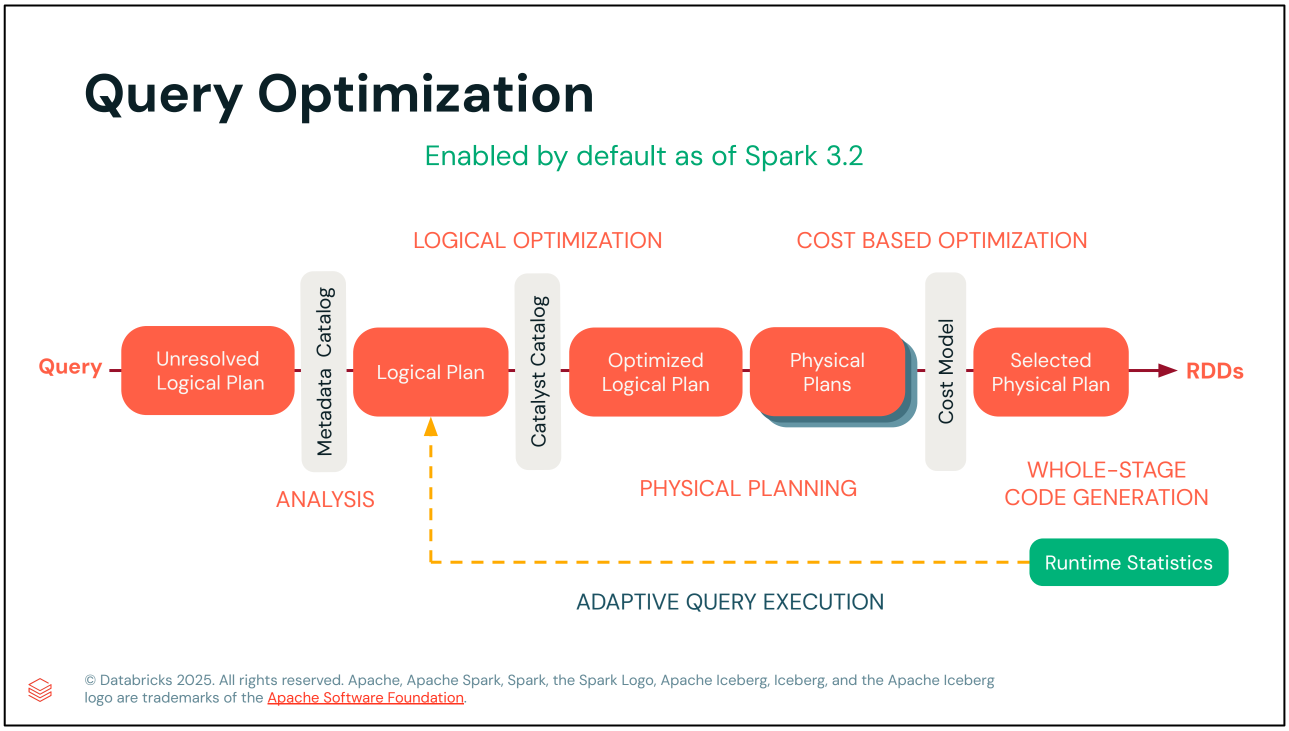
Job Optimization with AQE (Adaptive Query Execution):

Enhanced Optimization with AQE:

- AQE in Spark introduces runtime adaptability to optimize job execution based on actual data and execution statistics.

Key Components of AQE in Job Optimization:

- Dynamic Adaptations: AQE dynamically adjusts the execution plan during runtime based on observed data statistics.
- Runtime Optimizations: Dynamic partition pruning, join reordering, and other optimizations are performed as the job executes.



Query Optimization

Enabled by default as of Spark 3.2

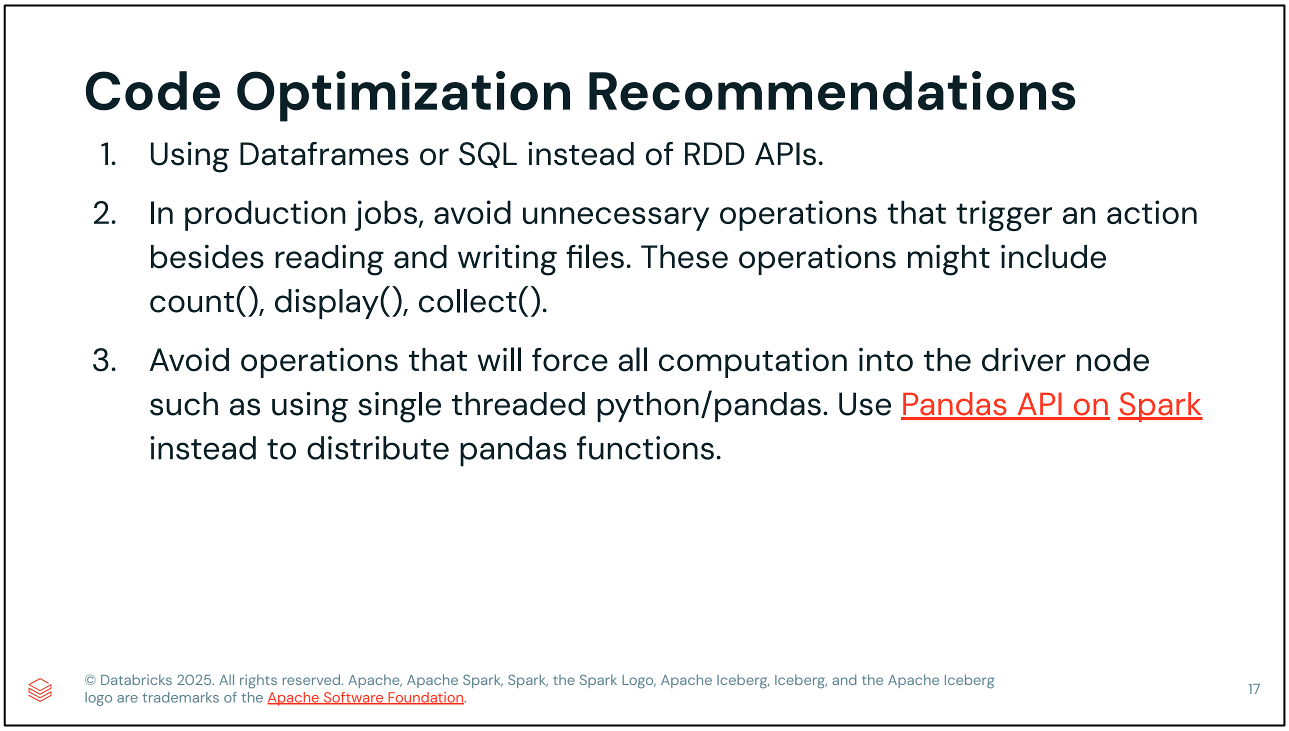
Job Optimization without AQE:

DataFrame Operations and Job Optimization:

- DataFrame operations in Spark involve a sequence of transformations and actions performed on distributed data.
- Job optimization in this context involves optimizing the sequence of DataFrame operations within a Spark job to minimize unnecessary shuffling, leverage parallelism, and optimize resource utilization.

Optimization Steps in Spark without AQE:

- Logical Plan: DataFrame operations are represented as a logical plan in Spark, defining the sequence of operations to be performed.
- Catalyst Optimization: Catalyst, Spark's optimizer, translates the logical plan into a physical plan, considering optimization rules and cost models.
- Physical Plan: The physical plan specifies how the computations will be executed on the Spark cluster, converting DataFrame operations into RDD transformations and actions.



Code Optimization Recommendations

1. Using Dataframes or SQL instead of RDD APIs.
2. In production jobs, avoid unnecessary operations that trigger an action besides reading and writing files. These operations might include count(), display(), collect().
3. Avoid operations that will force all computation into the driver node such as using single threaded python/pandas. Use Pandas API on Spark instead to distribute pandas functions.

- Encourage students to use the new and improved methods (DataFrames or Datasets) instead of older, less efficient methods (RDDs), which don't get the benefit of the teacher's (Spark's) best planning and optimization.
- Avoid having students do unnecessary work just to see the results (avoid triggering actions like displaying or collecting data unless really needed). Save those actions for when work actually needs to happen.
- Don't let a single student (the driver) do all the work on their own using single-threaded or basic pandas functions. Instead, use tools (like Pandas API on Spark or SQL) that let the whole class (all cores) help, spreading out the workload. This way, counting candies goes much faster and uses everyone in the class efficiently.

databricks

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