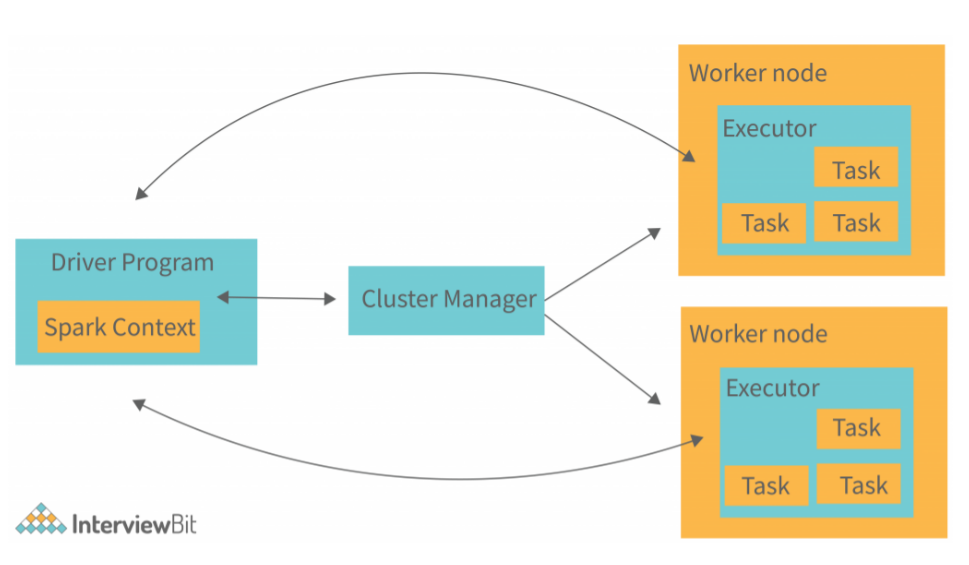
**APACHE SPARK**

# Spark architecture



**Driver program:** The Driver is the central component that runs the Spark application. It creates Spark Context, builds the execution plan (DAG), and schedules tasks on worker nodes. The Driver communicates with the Cluster Manager to request resources and collects results from Executors

**Cluster manager**: The Cluster Manager handles resource allocation for Spark applications. It assigns executors to jobs and ensures efficient resource utilization. Spark supports different cluster managers like YARN (Hadoop), Kubernetes, Mesos, and Standalone mode

**Worker node:** Executors are worker processes that run on different nodes in a cluster. They execute tasks in parallel, store intermediate results (if caching is enabled), and return processed data to the Driver. Each Executor has a fixed number of task slots, determining how many tasks it can run at once

**What happens when we submit a job to spark?**

* **Resource Request**: When the Driver Program is launched, it communicates with the Cluster Manager to request resources necessary for running the application, including the number of executors, memory per executor, and CPU cores. Driver program waits until response from Cluster manager
* Once the Executors are up and running, the Driver schedules tasks to be executed on these Executors. It uses the **DAG Scheduler** to divide the job into stages and tasks, sending the tasks to the appropriate Executors based on data locality and available resources.
* **Task Execution**: The Executors start executing the assigned tasks. They process data partitions, perform the required computations, and store intermediate results as needed. If a task requires data from other partitions, it may involve data shuffling
* **Result Collection**: As tasks complete execution, the Executors return results back to the Driver. The Driver collects and aggregates these results for further processing or presentation to the user

# Directed Acyclic Graph (DAG)

A **DAG (Directed Acyclic Graph)** in PySpark represents the sequence of computations performed on data. It shows how transformations and actions are executed in a **logical flow**, ensuring efficiency and fault tolerance.

**DAG Execution in PySpark**

When you write a PySpark program, it follows these two main steps:

1. **Transformations** (Lazy Evaluation)
   * These are operations that don’t execute immediately but define a logical execution plan.
   * Examples: .filter(), .select(), .join(), .groupBy().
   * They **create a DAG but don’t execute anything yet**.
2. **Actions** (Trigger Execution)
   * When you call an action like .count(), .show(), .collect(), Spark **executes** the DAG.
   * Spark breaks the DAG into **stages** and **tasks**, optimizing the execution.

# Spark Context vs Spark Session

**SparkContext (Old - Spark 1.x)**

1️. Manages low-level Spark operations but **only supports RDDs** (No built-in DataFrames or SQL)  
2️. Requires extra contexts like SQLContext and HiveContext for SQL and DataFrame operations  
3️. Less efficient and not recommended for new projects

**SparkSession (New - Spark 2.x & Later)**

1️. A **single-entry point** that supports RDDs, DataFrames, SQL, Streaming, and ML  
2️. Replaces SparkContext, SQLContext, and HiveContext—simplifying Spark usage  
3️. Recommended for all new applications due to better optimization and usability