Spark Core Concepts

Pavitra Varshini Prajit Aditya Shajan Selvam

Introduction to Apache Spark

~Apache Spark is an open-source, distributed computing system designed for big data processing and analytics.

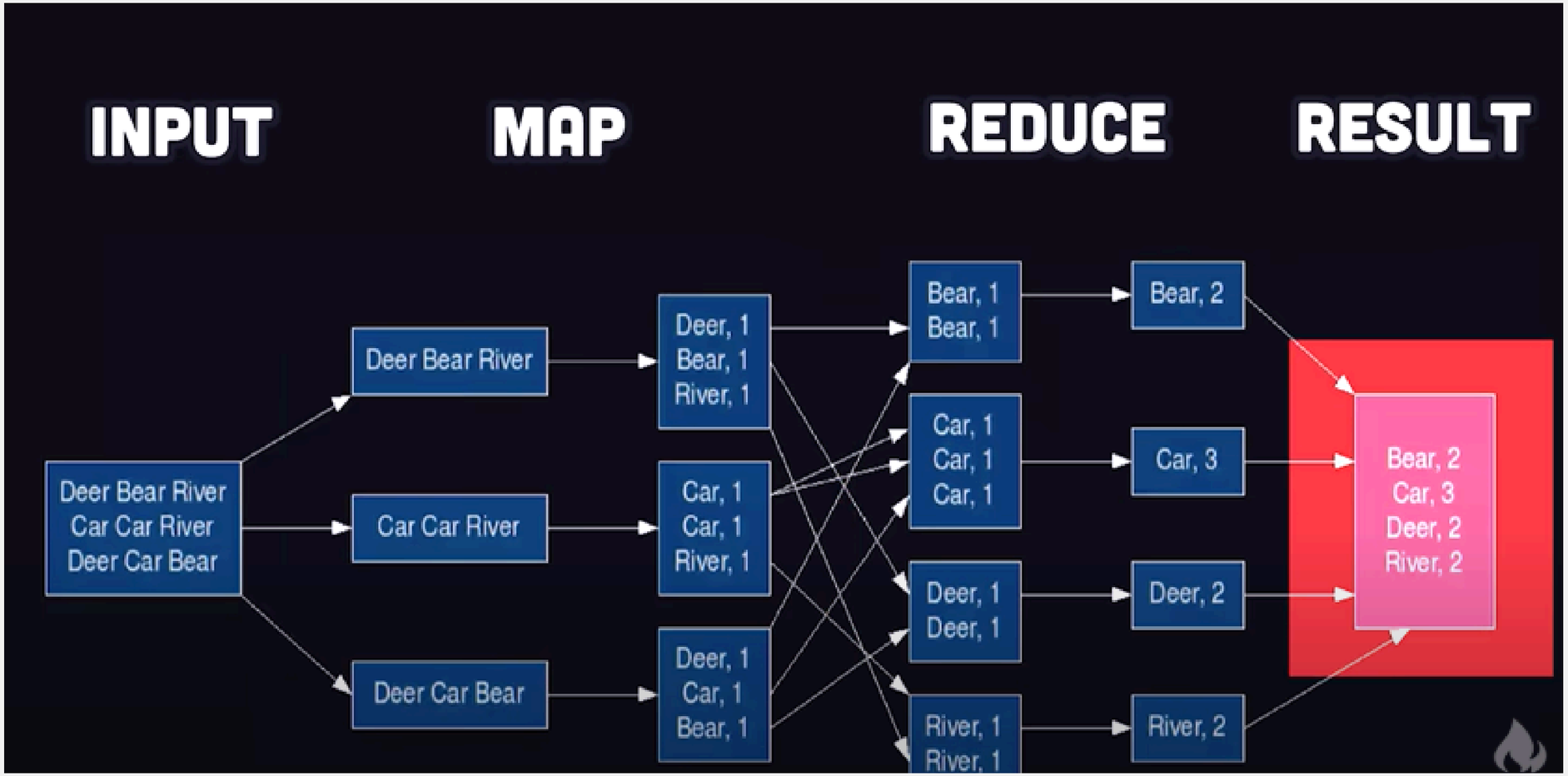
~It was developed at UC Berkeley and is now one of the most widely used frameworks for processing large-scale data efficiently

.

~Spark has become the go-to framework for handling big data, thanks to its speed, flexibility, and scalability.

~In this presentation, we'll explore its core concepts and architecture in more detail.

MapReduce:

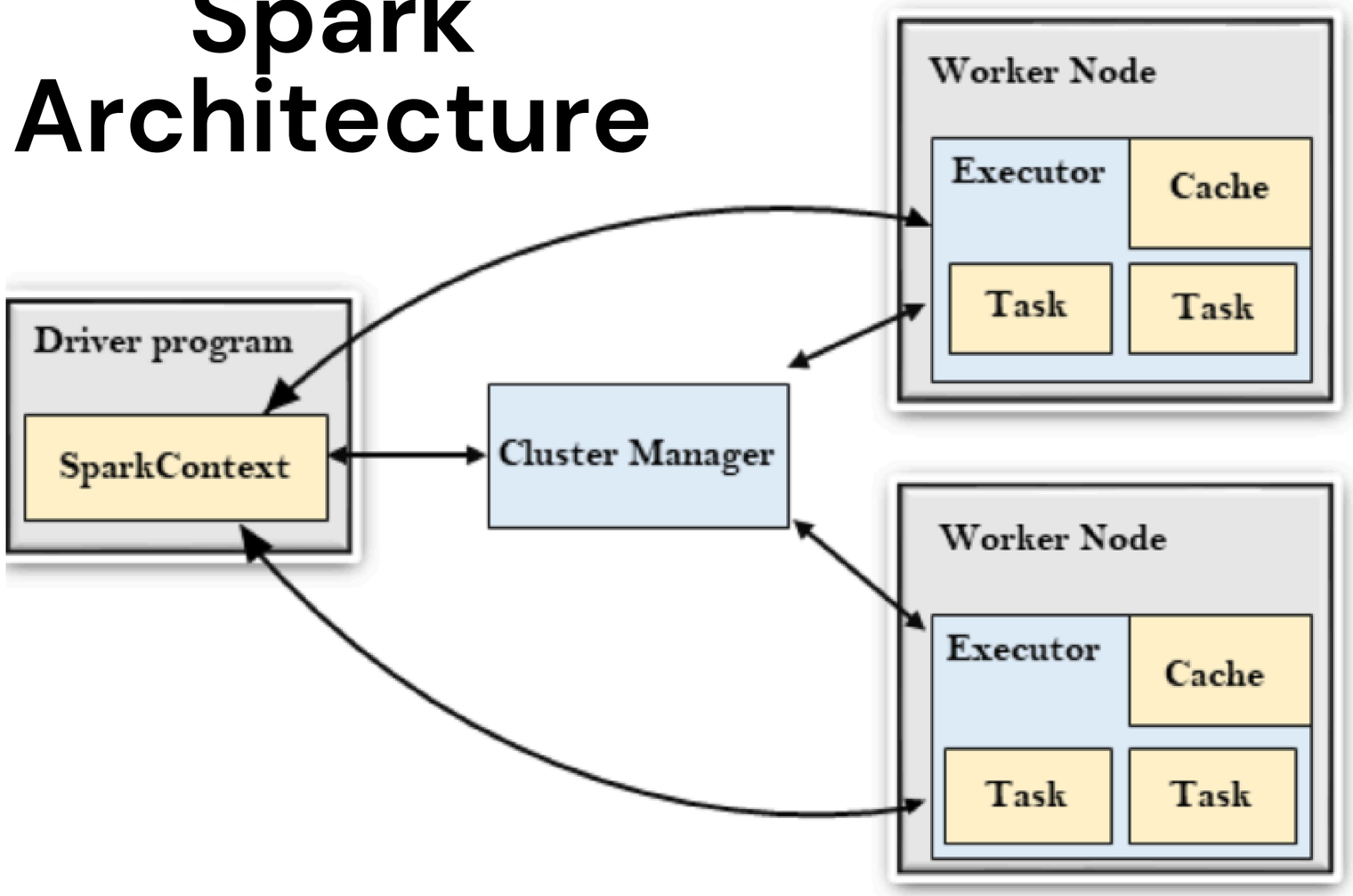


If Hadoop is like writing everything on paper before typing, Spark is like working directly on a computer.

Difference Between MapReduce and Spark

1. **Processing Approach**
   - MapReduce: Disk-based; writes intermediate results to disk after every step.
   - Spark: In-memory computation; keeps data in RAM, reducing disk I/O.
     Advantage: Spark is up to 100x faster than MapReduce for iterative tasks.
2. **Execution Speed**
   - MapReduce: Slow due to frequent disk reads/writes.
   - Spark: Lightning-fast because it minimizes disk access.
     Advantage: Spark processes large datasets much faster than MapReduce.
3. **Ease of Use**
   - MapReduce: Requires complex Java code (boilerplate-heavy).
   - Spark: Supports Python (PySpark), Scala, Java, and SQL, making it easier to write code.
     Advantage: Developers can write shorter and simpler code in Spark.
4. **Fault Tolerance**
   - MapReduce: Uses replication; if a node fails, it reprocesses data.
   - Spark: Uses RDD lineage, meaning it can recompute only the lost data instead of rerunning the entire job.
     Advantage: Spark is more efficient in handling failures.

Spark Architecture



Components

1.Driver Program
- The main process that runs the Spark application.
- Creates and manages RDDs, transformations, and actions.

2.Cluster Manager
- Allocates resources to Spark applications.
- Can be Standalone, YARN, or Mesos.

3.Executors
- Worker nodes that execute tasks assigned by the driver.
- Store data in memory for fast access.

4.Tasks
- The smallest unit of execution inside an executor.
- Multiple tasks run in parallel for efficiency.

Execution Flow:

- User submits a Spark application.
- Driver program creates an execution plan (DAG - Directed Acyclic Graph).
- Cluster Manager allocates resources to executors.
- Executors run tasks in parallel on worker nodes.
- Results are returned to the driver or stored in HDFS, databases, etc.

RDD (Resilient Distributed Dataset):

What is an RDD?

- Immutable, distributed collection of objects.
- Can be created from data sources (HDFS, local files, databases) or by applying transformations to existing RDDs.
- Provides fault tolerance through lineage (recomputes only lost data).

Characteristics

- Partitioned: Data is split across multiple nodes for parallel processing.
- Lazy Evaluation: Transformations are not executed immediately; they are only computed when an action is called.
- Fault-Tolerant: Uses lineage instead of replication.

Operations

- Transformations (Lazy, Return RDDs)
  - map(), filter(), flatMap(), groupBy(), reduceByKey()
- Actions (Trigger Execution, Return Values)
  - collect(), count(), reduce(), first(), take()

Transformation vs Actions

In Spark, transformations are operations that create a new RDD from an existing one but are lazy, meaning they do not execute immediately. Instead, they build a logical execution plan that Spark optimizes before running. Examples include map(), filter(), and reduceByKey(), which modify or restructure the data without triggering computation.

In contrast, actions are operations that trigger execution by instructing Spark to compute and return a result or save data externally. Actions like collect(), count(), and reduce() force Spark to execute all preceding transformations. This separation allows Spark to optimize execution, reducing unnecessary computations.

RDD vs. DataFrame vs. Dataset

RDD (Resilient Distributed Dataset)

- Low-level API with no schema.
- Works with structured and unstructured data.
- Optimized for functional transformations but lacks query optimization.

2. DataFrame

- Distributed collection of data with a schema (like SQL tables).
- More efficient than RDDs due to Catalyst optimizer and Tungsten execution engine.
- Supports SQL queries (.sql()), making it easier to use.

3. Dataset (Scala & Java Only)

- Like DataFrames but with type safety and compile-time checks.
- Uses encoders, making it more memory-efficient than RDDs.
- Best for structured data when type safety is needed.

- Use RDDs for low-level control and unstructured data.
- Use DataFrames for performance and ease of use.
- Use Datasets when type safety is required (Scala/Java).

Thank you very much!