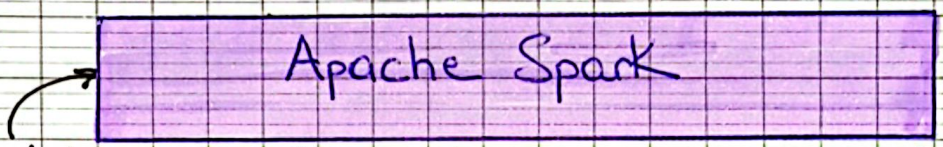
Chapter 6: Apache Spark

→ What is Spark?

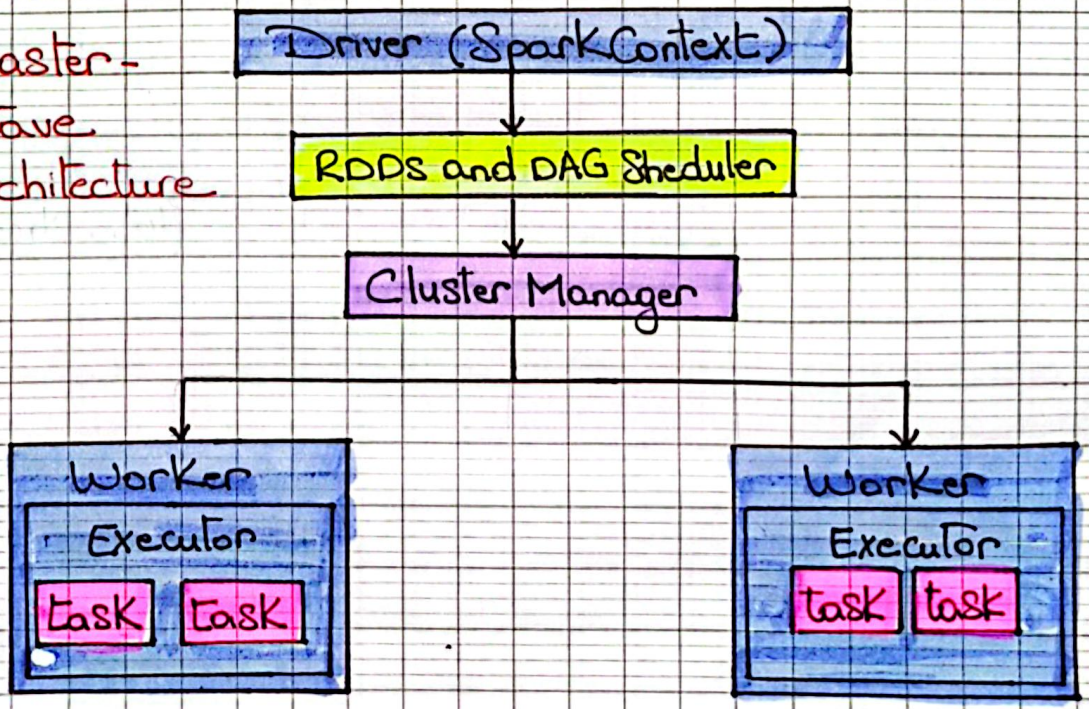
- Distributed computing system designed for big data processing and analytics
- It's known for its speed and efficiency
- Built to solve limitation of systems like Hadoop MapReduce
- Unified platform supporting multiple processing types



Foundation of Spark, handling memory management, task scheduling and interaction with storage systems

→ Spark Architecture

Master-Slave Architecture



. Components
↳ Driver : Contains SparkContext, coordinate execution
↳ Cluster Manager : Manages resources across cluster (Supports Yarn, Mesos, Kubernetes, standalone mode)
↳ Workers / Executors : Execute tasks on data partitions
↳ Task : smallest unit of work in Spark

→ Spark RDDs (Resilient Distributed Datasets)
. Definition : core data structure in Spark. They represent distributed collection of data (records) processed in parallel across the cluster

. Features :
↳ Resilient : Fault-tolerance through lineage tracking (no replication)
↳ Distributed : Data spread across multiple machines
↳ Dataset : Collection of partitioned data
↳ Immutable : Once created, cannot be modified
↳ Lazy Evaluation : Computations delayed until action is called

. Storage : In memory or disk

→ Creating RDDs - Three Methods
1) Loading from External Dataset (File)
. most common method
. Data can be located in HDFS, HBase, Amazon S3, Cassandra, etc.
. Ex: lines = spark.textFile("hdfs://...")
File path

2) Parallelizing centralized collection
- Convert local data structure to distributed RDD
- Ex:

```
val data = Array (1, 2, 3, 4, 5, 100, 8, 7, ...)
val distData = sc. parallelize (data)
val distData = sc. parallelize (data, 10) # create 10 partitions
```

3) Creating new RDD from existing RDD
- Create new RDDs by transforming existing ones using transformations operations
- Ex: lines. Filter (_. startsWith ("ERROR"))

existing RDD ↙ ↳ operation example can be any other operation

→ RDD operations
- Two types of operations
  1) Transformation (Similar to map-side of Hadoop)
  2) Action (Similar to reduce-side of Hadoop)

1) Transformations
- Function that produces new RDD from existing one.
- Takes RDD a input and produces one or more RDD as output
- Input RDD left intact (immutability)
- Lazy evaluation: they get execute when we call an action and not immediatly

Transformations Types

1) Narrow Transformation
   - Each partition of the input RDD contributes to only one partition of the output RDD.
   - No data is shuffled across partitions

   - Examples:
     → map():
     ```
     val rdd = sc.parallelize (List (1,2,3))
     val result= rdd.map ( x => x + 1) // output : 2,3,4
     ```

     → FlatMap()
     ```
     val rdd = sc.parallelize (List ("hello world"))
     val result= rdd.flatMap (line => line.split (" "))
     // output : "hello", "world"
     ```

     → Filter()
     ```
     val rdd = sc.parallelize (List (1,2,3,4))
     val result = rdd. filter ( x => x %2 == 0) // Output: 2,4
     ```

2) Wide Transformation
   - Each partition of the input RDD may contribute to multiple partitions of the output RDD
   - Data is shuffled

   - Examples
     → groupByKey()
     ```
     val rdd = sc.parallelize (Seq (("A", 1), ("B", 2), ("A", 3)))
     val result = rdd.groupByKey() // Output (A , (1,3))
     result. collect().foreach (println)          (B , 2)
     ```

↳ join()

```
val rdd1 = sc.parallelize (Seq(("A",1), ("B",2)))
val rdd2 = sc.parallelize (Seq(("A",3), ("B",4)))
val result = rdd1.join(rdd2)
result.collect().foreach(println) // Output: (A,(1,3))
                                             (B,(2,4))
```

2) Actions

- Perform compulation on existing RDDS producing a result
- Resut oplions
  - ↳ returned to Driver Program
  - ↳ Slored in File system (like HDFS)
- Examples: count(), collect(), reduce(), save()

↳ A Complete Example

I)

```
// 1. Create RDD From external source (no execution yet)
val lines = sc.textFile("data.txt")
// 2. Apply length function to each line (still no execution)
val lineLengths = line.map(s => s.length)
// 3. Action: reduce operation
val totalLength = lineLengths.reduce((a,b) => a+b)
```

↳ Now everything executes, Spark will:

1) read "data.txt" From disk
2) calculates length of each line
3) Sums all length together

II) // 1. Create RDD from external text file
```
val lineRDD = sc.textFile("data.txt")
// 2. Split each line into words and flattens result
val wordRDD = lineRDD.FlatMap(_.split(" "))
// 3. Keeps only elements matching condition (words = "the")
val filteredWordRDD = wordRDD.filter(_.equalsIgnoreCase("the"))
// 4. Action : collect (return all RDD elements to driver program)
filteredWordRDD.collect
```

↳ Result : Array of strings containing all occurrences of "the"

→ Spark Programming Model
- Primary Language : Scala
- Other : Java, Python, R
- Scala Features
  - ↳ Combines OOP and Functional Programming
  - ↳ Compile to Java bytecode
  - ↳ Runs on JVM