**🔥 What is PySpark?**

**PySpark** is the **Python API for Apache Spark**, a powerful **distributed data processing engine**. It allows Python developers to harness the power of **Apache Spark** for big data processing, machine learning, and ETL.

⚙️ PySpark Architecture Overview

+----------------+

| PySpark Code | ← (Python Program using PySpark API)

+----------------+

|

v

+-------------------+

| PySpark Driver App| ← (Your Python application)

+-------------------+

|

v

+--------------------+

| Spark Driver (JVM) | ← (Runs on the Master Node)

+--------------------+

|

v

+----------------------+

| Cluster Manager | ← (YARN / Spark Standalone / Mesos / Kubernetes)

+----------------------+

|

v

+------------------+ +------------------+ +------------------+

| Executor 1 | | Executor 2 | | Executor 3 | ← (Run Tasks)

| (Worker Node) | | (Worker Node) | | (Worker Node) |

+------------------+ +------------------+ +------------------+

**🧩 Key Components**

**1. Driver Program**

* Your main PySpark application (Python file)
* Converts Python code to **RDDs/DataFrames**
* Controls the Spark job: task scheduling, result collection

**2. SparkSession**

* Entry point to use Spark functionality in PySpark:

from pyspark.sql import SparkSession

spark = SparkSession.builder.appName("MyApp").getOrCreate()

**3. Cluster Manager**

* Allocates resources across the cluster
* Types:
  + **Standalone** (built-in)
  + **YARN** (used in Hadoop)
  + **Kubernetes**
  + **Mesos**

**4. Executors**

* Run on **Worker Nodes**
* Perform the actual computation (map, reduce, filter, etc.)
* Store data in memory/disk during execution

**5. Tasks**

* A **job** is split into **stages**
* Each stage has multiple **tasks**
* Tasks are distributed to executors

**🔄 How PySpark Executes a Program (Step-by-Step)**

1. **Write Python Code** using PySpark APIs
2. **Driver Program** converts your code into a **DAG** (Directed Acyclic Graph)
3. Spark breaks DAG into **stages** and **tasks**
4. Tasks are sent to **executors**
5. **Executors** perform computation and return results
6. **Driver** collects results or stores them in HDFS/S3/DB

**💡 Memory Management**

* Spark stores data in **memory** (RAM) by default — much faster than disk.
* If memory is full, it **spills to disk**.
* You can **cache** or **persist** data to avoid recomputation:

df.cache() # keeps data in memory

📊 Data Representations

| Type | Description |

| ------------- | --------------------------------------------- |

| \*\*RDD\*\* | Resilient Distributed Dataset (low-level API) |

| \*\*DataFrame\*\* | Table-like structure (high-level, optimized) |

| \*\*Dataset\*\* | Type-safe, available in Scala/Java only |

PySpark primarily uses **DataFrames** for performance and ease.

**🧠 Optimizations Used by PySpark**

* **Catalyst Optimizer**: Optimizes queries on DataFrames
* **Tungsten Engine**: Enhances execution (e.g., memory management)
* **Lazy Evaluation**: Transformations are not executed until an action is called (like .collect() or .show())

✅ Real-World Example

from pyspark.sql import SparkSession

spark = SparkSession.builder.appName("SalesApp").getOrCreate()

df = spark.read.csv("sales.csv", header=True, inferSchema=True)

df\_filtered = df.filter(df["region"] == "South")

df\_grouped = df\_filtered.groupBy("product").sum("amount")

df\_grouped.show()

Load → Filter → Group → Action → Triggers execution on the cluster