
Spark

MapReduce revolutionized Big Data...

but it has limitations for:

- Iterative algorithms
- Complex multi-stage pipelines
- Real-time or near real-time processing.

Spark to the Rescue ..!

What is Apache Spark?

Introducing Spark: A Unified Big Data Engine

- In-memory distributed computing framework designed for speed and scale.

What is Apache Spark?

Writing to disk between stages :(

Introducing Spark: A Unified Big Data Engine

- **In-memory** distributed computing framework designed for speed and scale.

What is Apache Spark?

Introducing Spark: A Unified Big Data Engine

- In-memory distributed computing framework designed for speed and scale.
- Part of the Apache ecosystem, often used with Hadoop

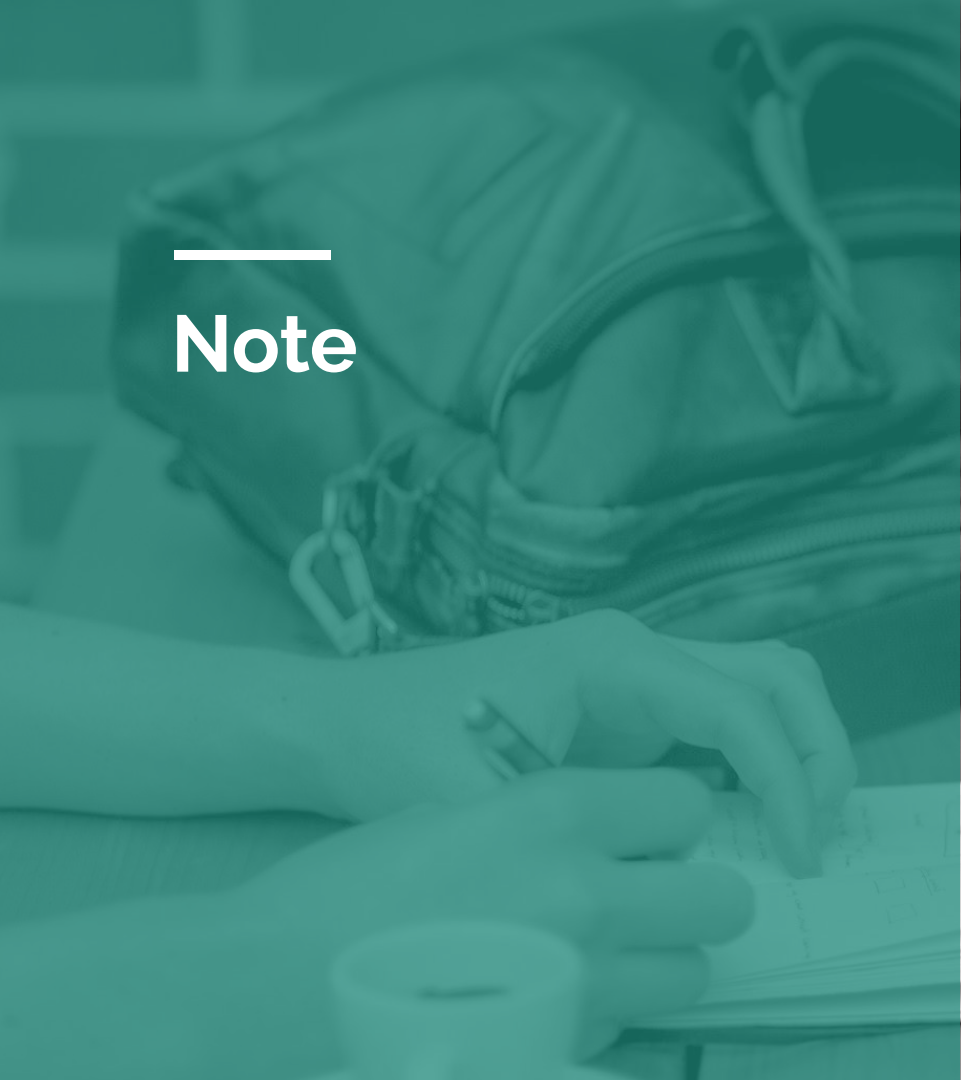
Welcome to the Zoo!

What is Apache Spark?

Introducing Spark: A Unified Big Data Engine

- In-memory distributed computing framework designed for speed and scale.
- Part of the Apache ecosystem, often used with Hadoop
- Offers a rich set of APIs in multiple languages (Python, Java, Scala, R)

Spark's RDDs (Resilient Distributed Datasets)



Note

Spark isn't a direct replacement for MapReduce.

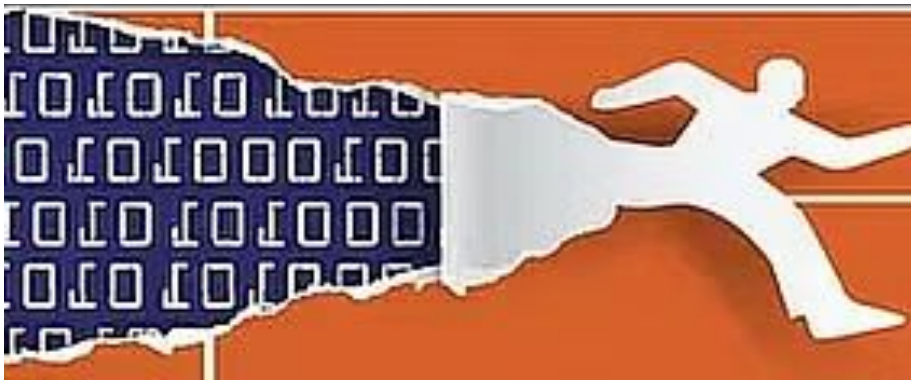
Apache Spark is an open source, wide range data processing engine with revealing development API's, that qualify data workers to accomplish streaming in spark, machine learning or SQL workloads which demand repeated access to data sets.

Apache Spark is an open source, wide range data processing engine with revealing development API's, that qualify data workers to accomplish streaming in spark, machine learning or SQL workloads which demand repeated access to data sets.

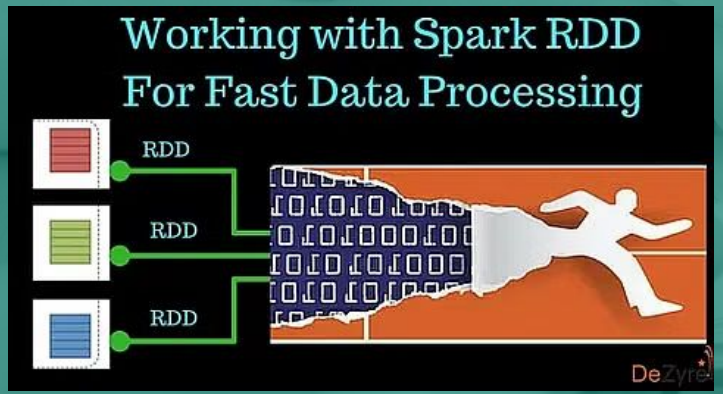
Spark Features

Blazing Speed with In-Memory Processing

How fast can it be with in-memory computations?



Blazing Speed with In-Memory Processing



Up to 100x faster than MapReduce in specific scenarios (especially iterative ones).

Blazing Speed with In-Memory Processing

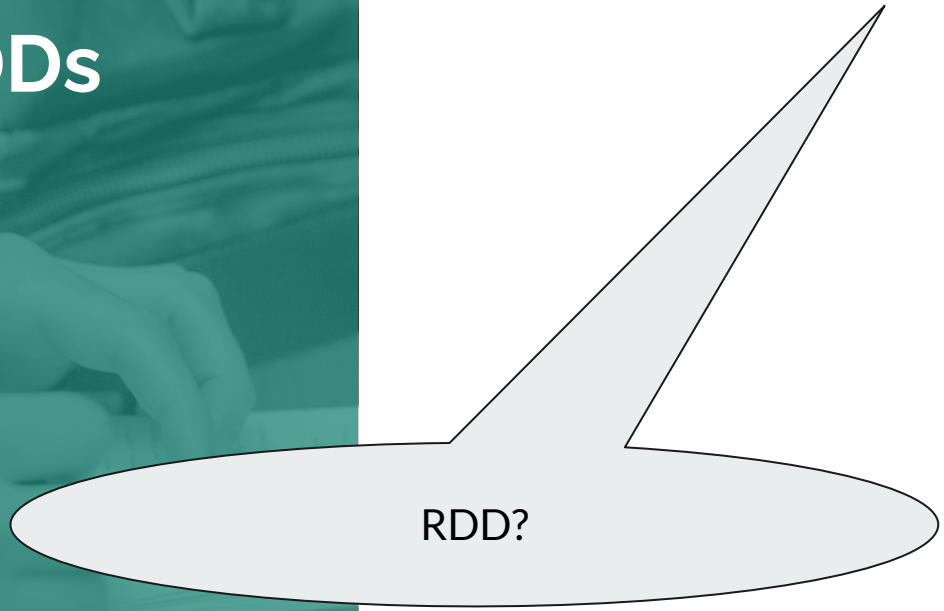
In-Memory for Lightning-Fast Analysis

- Spark prioritizes storing data within RAM (memory) of a cluster
- Significantly faster than disk-based processing (like MapReduce)
- Ideal for iterative algorithms and interactive analysis

Fault Tolerance through RDDs

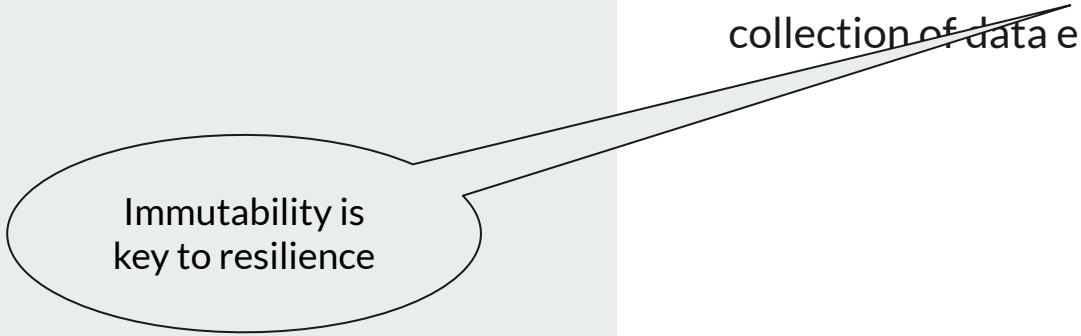
Resilient in the Face of Failure

- RDDs track their lineage – how they were created from other RDDs.

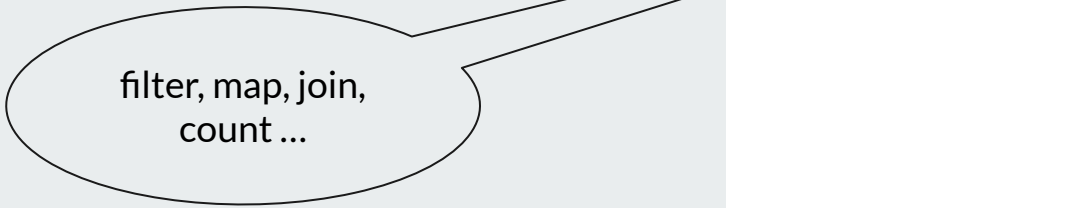


What is RDD?

- **RDD** stands for **Resilient Distributed Dataset.**
- **Fundamental data structure in Spark.** This is how Spark works with your data.
- An RDD is an immutable, distributed collection of data elements.



Understanding Immutability in Spark



Immutability: The Key to Spark's Resilience

- RDDs themselves cannot be changed after they're created.
- So, how does Spark allow us to manipulate and work with data?
- **Transformations & Actions**: Manipulating Data without Changing the Original.

Transformations: Creating New RDDs

Manipulating Data without Changing the Original

- Operations like filter, map, join don't modify the original RDD.
- Instead, each transformation produces a new RDD.
- **The new RDD reflects the changes you want.**

Actions: Extracting value

Manipulating Data without Changing the Original

- Return value or export data
- Actions include count, collect.

Fault Tolerance through RDDs

Resilient in the Face of Failure

- RDDs track their lineage – how they were created from other RDDs.
- If a machine goes down, Spark can recompute lost data partitions.
- Ensures long-running Big Data computations complete successfully.

Advanced Analytics with Machine Learning

Spark Fuels Smarter Data Analysis

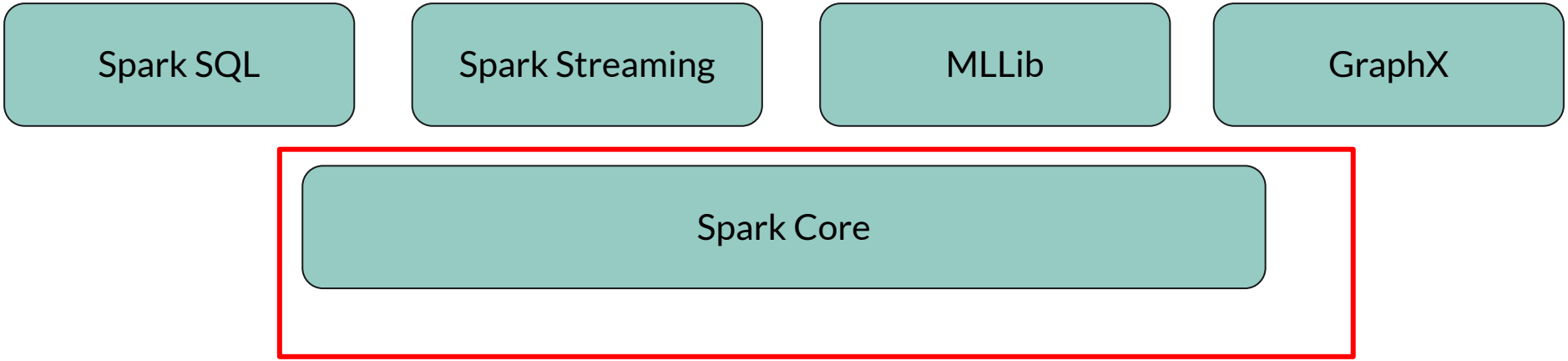
- **MLlib** library provides scalable implementations of common algorithms
  - Classification, regression, clustering, etc.
- In-memory nature makes iterative ML computations faster.

Recap: Spark Features

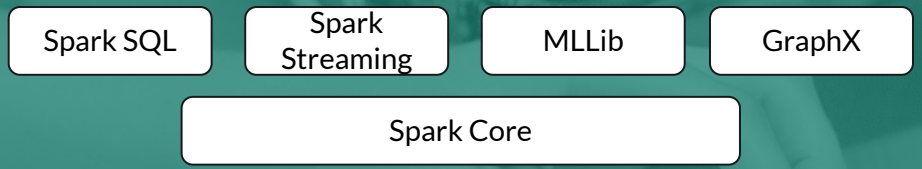
- Fault Tolerance through RDDs
- Advanced Analytics with Machine Learning
- Blazing Speed with In-Memory Processing

Spark Components

Components of Spark



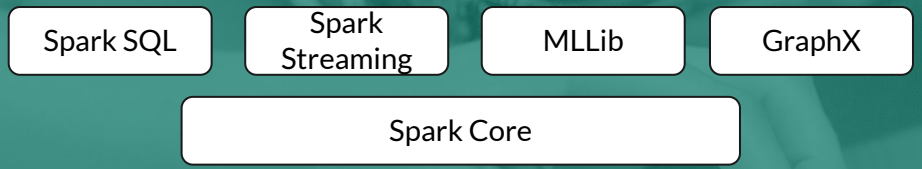
Spark Core



The Foundation of Distributed Computing

- Base engine for distributed data processing.
- Foundation of the RDD concept.
- Provides APIs for transformations, actions, and task scheduling.

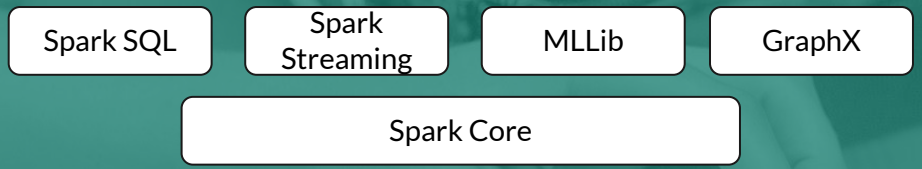
Spark SQL



SQL for Big Data

- Lets you query structured data within Spark using SQL syntax or DataFrame APIs.
- Integrates relational processing with Spark's functional model.

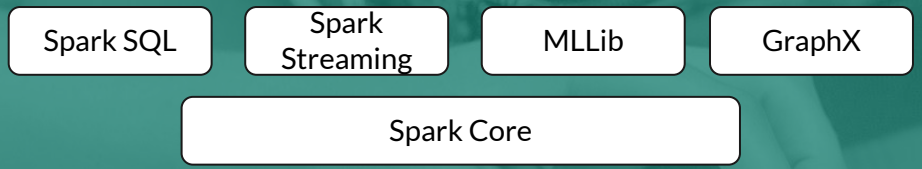
Spark Streaming



Handling Real-Time Data

- Extends Spark for continuous data streams (think IoT, logs, etc.)
- Example - Tracking Sentiment on Social Media with Spark Streaming.

Spark MLLib



Scalable Machine Learning

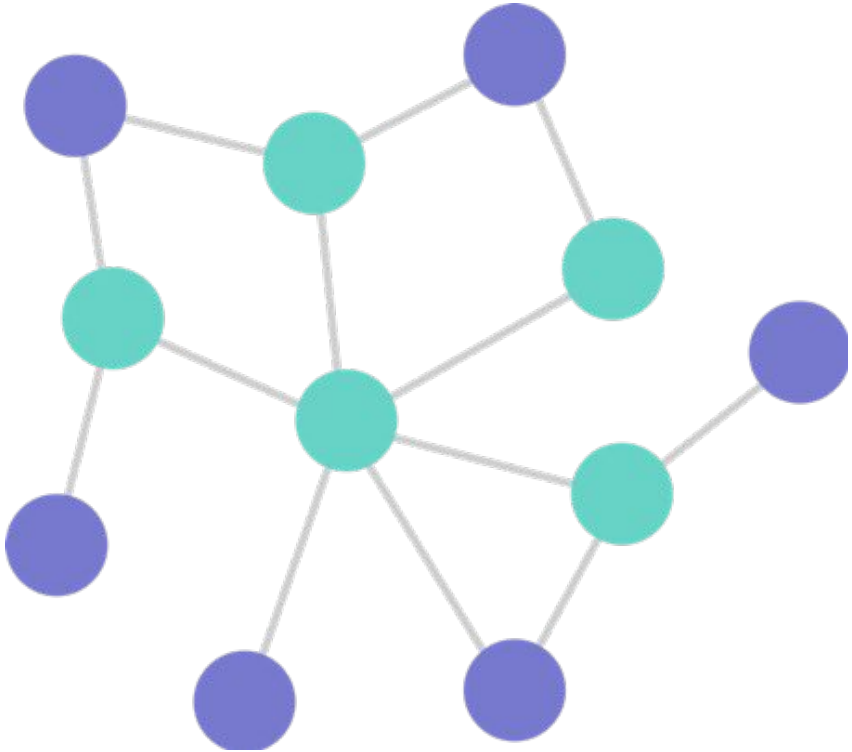
- Library of common machine learning algorithms.
  - Classification, Regression, Clustering, etc.
- Optimized for distributed computation.
- Tools for feature engineering, pipelines, and model evaluation

Spark GraphX

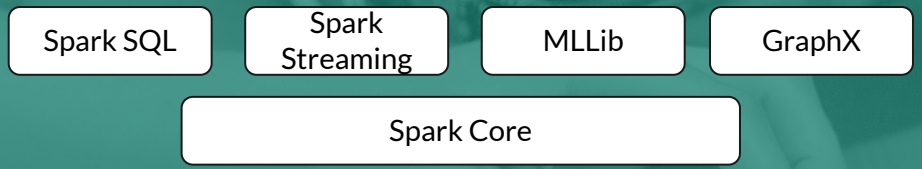
Spark SQL | Spark Streaming | MLLib | GraphX

Spark Core

GraphX: Demystifying Relationships with Spark



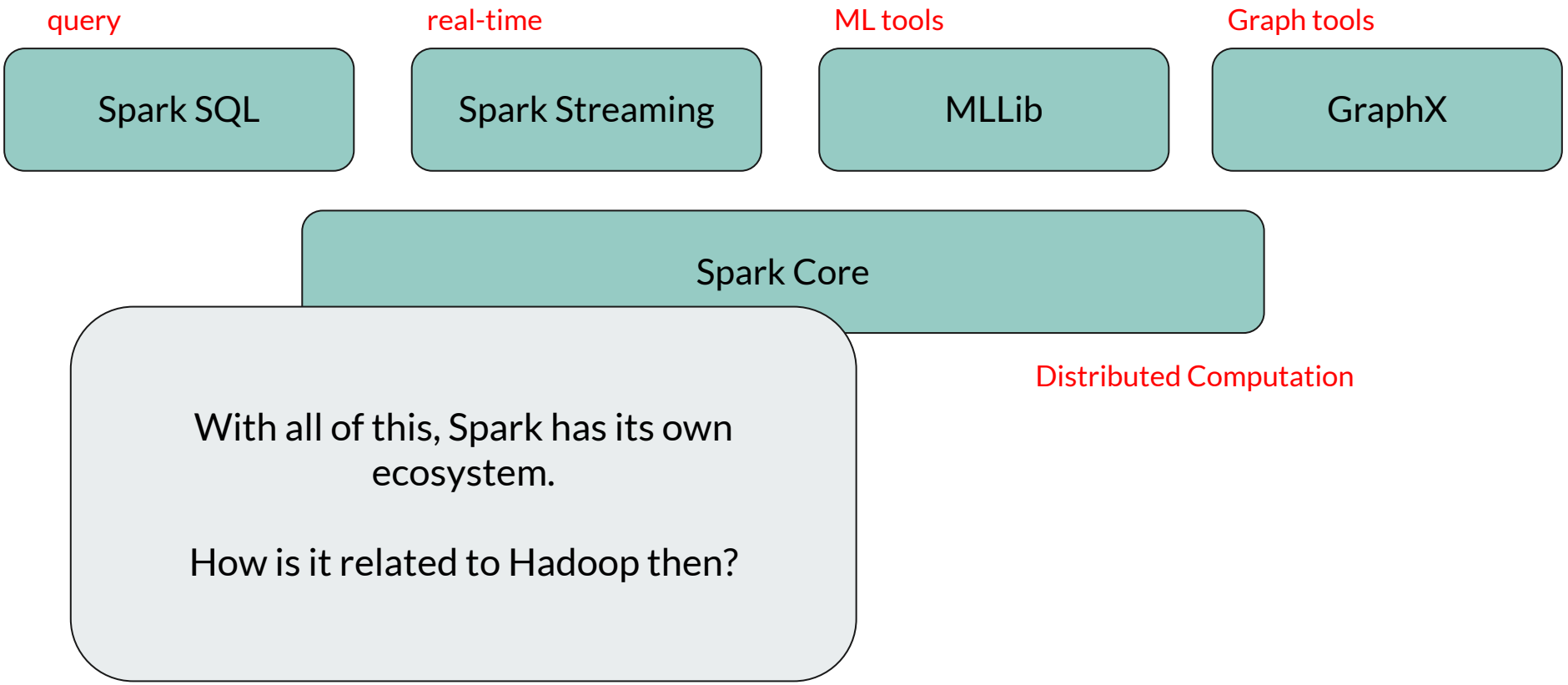
Spark GraphX



GraphX: Demystifying Relationships with Spark

- A Spark component designed for graph processing and analysis.
- Graphs: Data structures where nodes represent entities, and edges represent relationships between them.
- Provides APIs for graph creation, transformations, and common algorithms.

Components of Spark

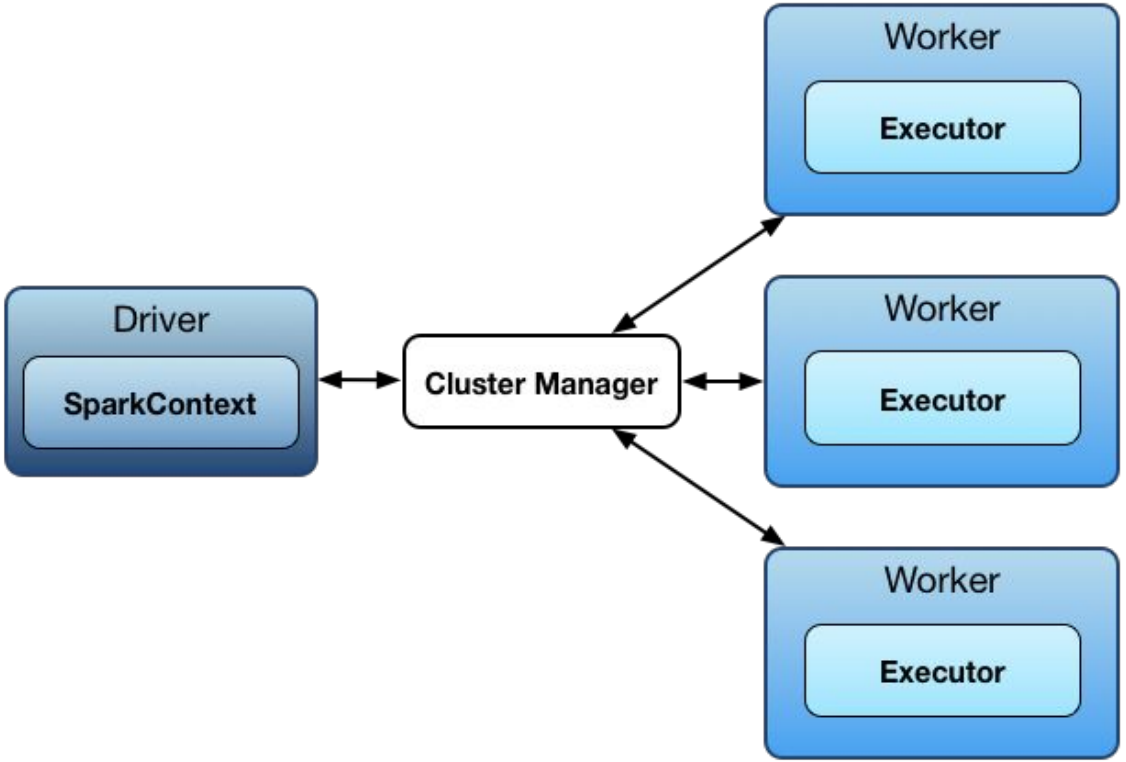


Welcome to the Zoo!

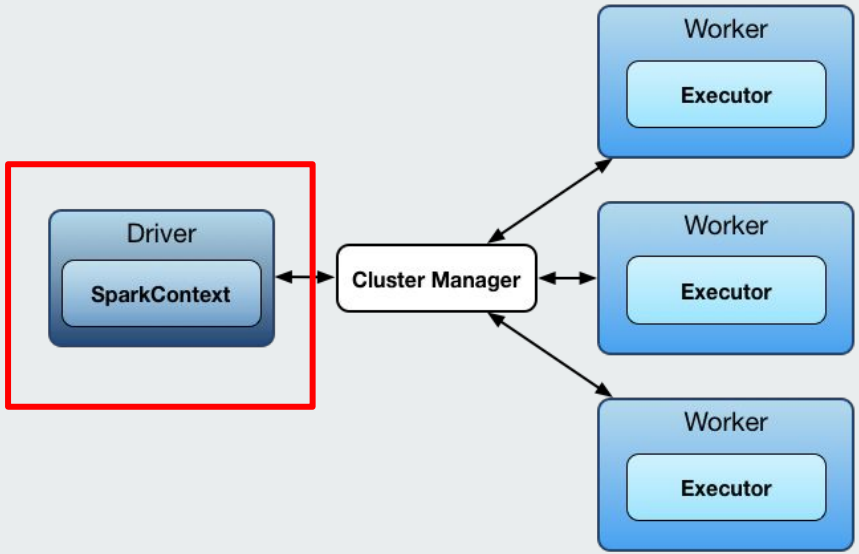
Spark is its own standalone distributed computing framework. It does not require Hadoop to function. Spark CAN work with Hadoop for storage.

Spark Architecture

Spark Architecture

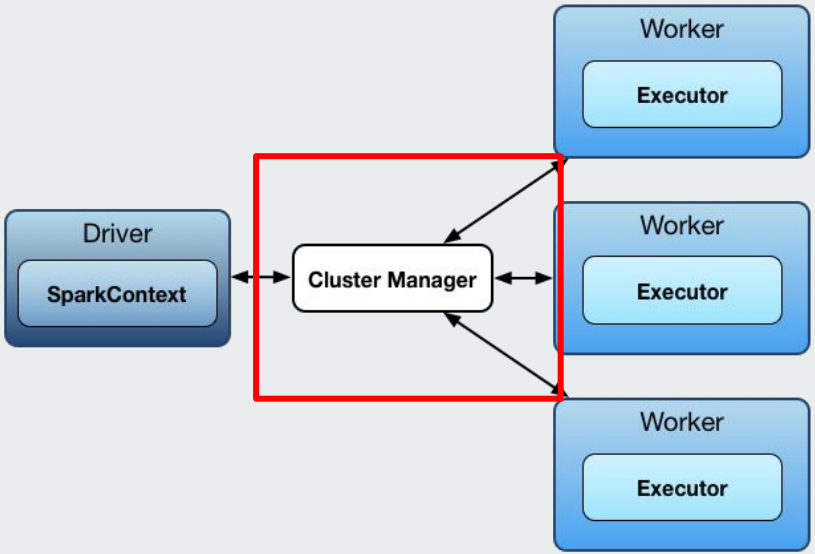


The Driver Program



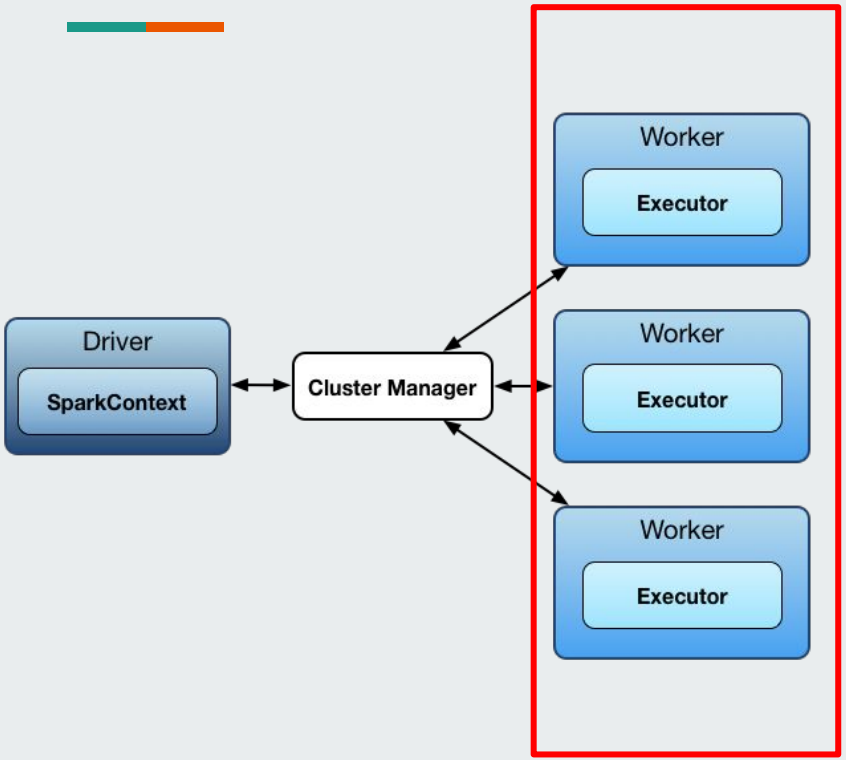
- The Brain of Your Spark Application - Your main entry point as a developer.
- Contains the **SparkContext**, which coordinates with the cluster.
- Defines your RDDs and the operations you wish to perform.

Cluster Manager



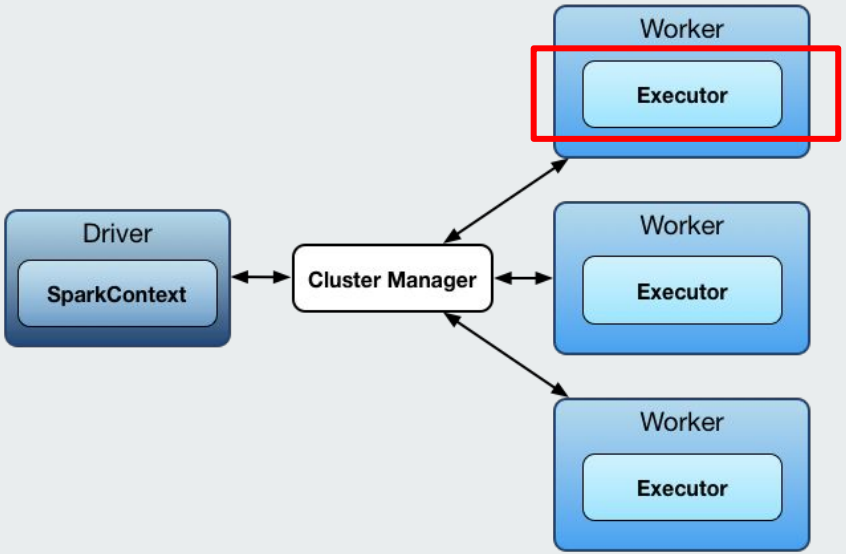
- Responsible for allocating resources (CPU, memory) across the cluster.
- Spark supports several cluster managers:
  - Standalone (built-in)
  - YARN (common in Hadoop setups)
  - Mesos

Worker Nodes



- Machines that actually perform the computations.
- Store partitions of your RDDs in memory.
- Each worker has Executors.

Executors



- Live within worker nodes.
- Responsible for executing tasks assigned by the driver program.
- Process data and report the results back.

Example Code

```
import pyspark

# 1. Create a SparkContext
sc = pyspark.SparkContext(appName="WordCount")
```

SparkContext: sc is our gateway to Spark.

The **appName** helps identify your job.

```
import pyspark

# 1. Create a SparkContext
sc = pyspark.SparkContext(appName="WordCount")

# 2. Load a Text File
text_file = sc.textFile("your_input_file.txt")
```

RDD Creation: We load a text file.

textFile creates an RDD where each line is an element.

```
import pyspark

# 1. Create a SparkContext
sc = pyspark.SparkContext(appName="WordCount")

# 2. Load a Text File
text_file = sc.textFile("your_input_file.txt")

# 3. Transformations
counts = text_file.flatMap(lambda line: line.split("
                    ")) \
                  .map(lambda word: (word, 1)) \
                  .reduceByKey(lambda a, b: a + b)
```

Transformations:

flatMap splits lines into words.

map creates (word, 1) pairs for each word.

reduceByKey adds counts for the same word, resulting in a "word count" RDD.

```
import pyspark

# 1. Create a SparkContext
sc = pyspark.SparkContext(appName="WordCount")

# 2. Load a Text File
text_file = sc.textFile("your_input_file.txt")

# 3. Transformations
counts = text_file.flatMap(lambda line: line.split("
                    ")) \
                .map(lambda word: (word, 1)) \
                .reduceByKey(lambda a, b: a + b)

# 4. Action
counts.saveAsTextFile("output_directory")

# 5. Stop Spark Context
sc.stop()
```

Action:
saveAsTextFile triggers computation and saves output. Files in the output directory will contain (word, count) pairs.

Context Cleanup:
Good practice to stop the **SparkContext** when done.

```
import pyspark

# 1. Create a SparkContext
sc = pyspark.SparkContext(appName="WordCount")

# 2. Load a Text File
text_file = sc.textFile("your_input_file.txt")

# 3. Transformations
counts = text_file.flatMap(lambda line: line.split("
                    ")) \
                  .map(lambda word: (word, 1)) \
                  .reduceByKey(lambda a, b: a + b)

# 4. Action
counts.saveAsTextFile("output_directory")

# 5. Stop Spark Context
sc.stop()
```

How Spark Works Behind the Scenes

- The **Driver** (your script) splits the file and distributes lines of text to Worker Nodes.
- **Executors** on each **Worker** perform the flatMap and map.
- Data is shuffled for reduceByKey – words are grouped for counting.
- **Executors** do the final counting.
- Results are combined by the **Driver** and written to the output directory.

```
import pyspark

# 1. Create a SparkContext
sc = pyspark.SparkContext(appName="WordCount")

# 2. Load a Text File
text_file = sc.textFile("your_input_file.txt")

# 3. Transformations
counts = text_file.flatMap(lambda line: line.split("
                    ")) \
                .map(lambda word: (word, 1)) \
                .reduceByKey(lambda a, b: a + b)

# 4. Action
counts.saveAsTextFile("output_directory")

# 5. Stop Spark Context
sc.stop()
```

Now replaced with
“SparkSession”

Feature	SparkContext	SparkSession
Introduced	Earlier Spark versions	Spark 2.0 onwards
Entry Point	Primary entry point	Unified entry point for all Spark functionalities
Main Data Structure	RDDs	DataFrames and Datasets

Feature	SparkContext	SparkSession
Introduced	Earlier Spark versions	Spark 2.0 onwards
Entry Point	Primary entry point	Unified entry point for all Spark functionalities
Main Data Structure	RDDs	DataFrames and Datasets

SparkContext functionality

SQLContext (for working with structured data)

StreamingContext (for real-time data processing)

HiveContext (for Hive integration)

```
import pyspark

# 1. Create a SparkContext
sc = pyspark.SparkContext(appName="WordCount")

# 2. Load a Text File
text_file = sc.textFile("your_input_file.txt")

# 3. Transformations
counts = text_file.flatMap(lambda line:
          line.split(" ")) \
             .map(lambda word: (word, 1)) \
             .reduceByKey(lambda a, b: a +
                  b)

# 4. Action
counts.saveAsTextFile("output_directory")

# 5. Stop Spark Context
sc.stop()
```

```
import pyspark.sql.functions as F
from pyspark.sql import SparkSession

# 1. Create a SparkSession
spark =
SparkSession.builder.appName("WordCount").getOr
Create()

# 2. Load a Text File (as a DataFrame)
text_df =
spark.read.text("your_input_file.txt")

# 3. Transformations
counts_df =
text_df.select(F.explode(F.split(F.col("value")
, " ")).alias("word")) \
                .groupBy("word") \
                     .count()

# 4. Action
counts_df.write.text("output_directory")

# 5. (Optional) Stop SparkSession - often not
needed with interactive sessions
spark.stop()
```

F.explode : Transform a column containing arrays (lists) or maps into multiple rows.

Originally -

```
# +---+---------+
# | id|   fruits|
# +---+---------+
# |  1|[apple,..]|
# |  2| [orange]|
# +---+---------+
```

F.explode("fruits") -

```
# +------+
# | fruit|
# +------+
# | apple|
# |banana|
# |orange|
# +------+
```

Fun Facts

- Spark set a record in 2014, sorting 100 TB of data on a cluster in only 23 minutes.
- The movie "The Martian" used Spark to analyze NASA data for visually simulating Mars landscapes! Talk about out-of-this-world applications.

Fun Facts

- Spark set a record in 2014, sorting 100 TB of data on a cluster in only 23 minutes.
- The movie "The Martian" used Spark to analyze NASA data for visually simulating Mars landscapes! Talk about out-of-this-world applications.
- "Spark" was initially a coded reference to how it aims to be faster than Hadoop's MapReduce..