1) What is Apache Spark and how it is Different from Hadoop?

Ans)

Spark: Spark is an open-source distribution computing engine. We use it for processing and analyzing a large amount of data. Like Hadoop, spark also works in distributed nature but differs in-memory processing.

Hadoop: Hadoop is a distributed computing framework that allows you to process data on a cluster of computers. It consists of two main components - **Hadoop Distributed File System (HDFS).** HDFS is a distributed file system that stores data on a cluster of computers.

➔ **How Spark is Different from Hadoop**

i. Compared to Hadoop or traditional systems, 100 times faster in memory and 10 times faster in disc. Its lightning speed is from in-memory and parallel processing.
ii. Spark is a general-purpose cluster computing system while Hadoop is a distributed file system.
iii. **Spark uses in-memory processing while Hadoop uses disk-based processing.** It means - Spark can process data much faster than Hadoop.
iv. **Spark is more fault-tolerant than Hadoop.** It Means - Spark can automatically recover from failures, while Hadoop requires manual intervention.
v. **Spark can be scaled to much larger clusters than Hadoop.**
vi. Spark uses a resilient distributed dataset (RDD) abstraction, while Hadoop uses a Hadoop Distributed File System (HDFS).
vii. Spark supports a wider range of programming languages, including Scala, Python, and Java, while Hadoop only supports Java.

2) What is the difference between DataFrame and RDD in apache Spark?

Ans)

DataFrames:

➔ DataFrames are a distributed collection of data organized in a tabular format.
➔ They are similar to relational tables in a database
➔ DataFrames are more structured than RDDs and can be manipulated using SQL-like syntax.
➔ Computation is lazy in DataFrames

RDDs:

➔ RDDs means - Resilient Distributed Datasets.
➔ RDDs are a distributed collection of data that is immutable and partitioned across multiple nodes in a cluster.
➔ RDDs are more flexible than DataFrames and can be used to represent a wider variety of data.
➔ Computation is Fast in RDDs

3) What is BUCKETING in Spark?

Ans)

- ➔ Bucketing is a technique for partitioning data into a fixed number of buckets.
- ➔ The number of buckets is typically determined by the number of partitions in the Spark cluster.
- ➔ Ensures that similar data is stored in the same bucket, so that it does not have to transfer data between nodes in the cluster.
- ➔ Bucketing can be used to improve the performance of certain operations, like **JOINS and Aggregations**.

Example of Bucketing:

```
import pyspark

# Create a SparkContext
sc = pyspark.SparkContext()

# Create a list of data
data = [(1, "Alice"), (2, "Bob"), (3, "Carol")]

# Create a DataFrame from the data
df = spark.createDataFrame(data, ["id", "name"])

# Bucket the DataFrame by the `id` column
df_bucketed = df.bucketBy(3, "id")

# Print the bucketed DataFrame
df_bucketed.show()
```

Syntax - df.bucketBy(Number_of_Bucketing,Column_name)

4) What is difference between RDD and Bucketing in spark?

RDDs:

- ➔ RDDs means - Resilient Distributed Datasets.
- ➔ It is a fundamental data structure in Spark. Once an RDD is created, it cannot be changed.
- ➔ RDDs are a distributed collection of data that is immutable and partitioned across multiple nodes in a cluster.
- ➔ Partitioning is done Automatic.

Bucketing:

- ➔ Bucketing is a technique for partitioning data into a fixed number of buckets.
- ➔ Ensures that similar data is stored in the same bucket, so that it does not have to transfer data between nodes in the cluster.
- ➔ This can be useful for improving the performance of certain operations, such as joins and aggregations.
- ➔ Partitioning done in Manual way

Key differences between RDDs and bucketing:

i. **RDDs are immutable, while bucketing is not**. This means that once an RDD is created, it cannot be changed. Bucketing, on the other hand, is a dynamic process. The buckets can be changed as needed.
ii. **RDDs are partitioned automatically, while bucketing is partitioned manually.** This means that with RDDs, Spark decides how to partition the data. With bucketing, you need to specify how the data should be partitioned.
iii. **RDDs are lazy, while bucketing is eager.** This means that RDDs are not computed until they are needed. Bucketing, on the other hand, is computed as soon as it is created.

5) What is Parquet file and why it is preferred in Databricks and BigData Processing?

Ans)

Reasons why Parquet is a popular choice for data storage in Databricks:

➔ **Parquet is a columnar storage format** (data is stored in columns, rather than rows, which is more efficient for queries that only need to access a small subset of the data.
➔ Parquet files can be read and queried much faster than other file formats, such as JSON or CSV because **Parquet is a columnar storage format.**
➔ **Efficiency:** Parquet is a very efficient format for storing and querying large datasets. It is designed to be read and written quickly, even for large datasets.
➔ **Compression**: Parquet uses compression to reduce the size of the data files, which can save storage space and improve performance.
➔ **Schema validation:** Parquet files support schema validation, which can help to prevent errors and ensure that the data is consistent.
➔ **Interoperability**: Parquet is a widely supported format, which means that it can be read and written by many different applications. This makes it a good choice for storing data that needs to be shared with other systems.

6) Explain these File formats – JSON, Parquet, and AVRO?

Ans)

JSON: JSON stands for JavaScript Object Notation. It is a lightweight data-interchange format. JSON is text-based and human-readable, making it easy to understand and manipulate.

Parquet: Parquet is a columnar storage format. This means that data is stored in columns, rather than rows. This can make it more efficient to read and write data, especially for queries that only need to access a small subset of the data.

Avro: Avro is a binary data serialization format. This means that data is stored in a binary format, which can make it more efficient to read and write data. Avro also supports schema, which means that the data can be validated when it is read or written.

7) Why JSON format is not efficient as Parquet or Avro for big data processing.

Ans)

- ➔ **JSON is a text-based format:** JSON is a human-readable format, which means that it is not as efficient as binary formats like Parquet or Avro.
- ➔ Since JSON is a text-based formats, it require more space to store the data and are slower to read and write.
- ➔ **JSON does not support compression** which will not help in decreasing the storage space.
- ➔ **JSON does not have a schema**, which means that the data structure must be inferred when the data is read. This can lead to errors in the data.

8) How is Parquet file format much efficient than AVRO file format?

Ans)

- ➔ **Columnar storage: Parquet stores data in a columnar format** which means that data is stored by columns instead of rows. This makes it more efficient to read, write and access data a small subset of the data. **Avro stores data in a row-based format**, which is less efficient for queries that only need to access a small subset of the data.
- ➔ **Compression:** Parquet and Avro both supports compression, but Parquet's compression algorithms are more efficient.
- ➔ **Schema evolution:** Parquet supports schema evolution, which means that the schema of the data can be changed without breaking compatibility with existing data. Avro does not support schema evolution, which can be a problem if you need to change the schema of your data.
- ➔ **Interoperability:** Parquet is a more widely supported format than Avro. This means that it can be read and written by more applications.

9) What is PySpark?

Ans)

- ➔ PySpark is a Python API for Apache Spark
- ➔ PySpark can be used for a variety of data processing tasks, including batch processing, streaming processing, machine learning, and graph processing.

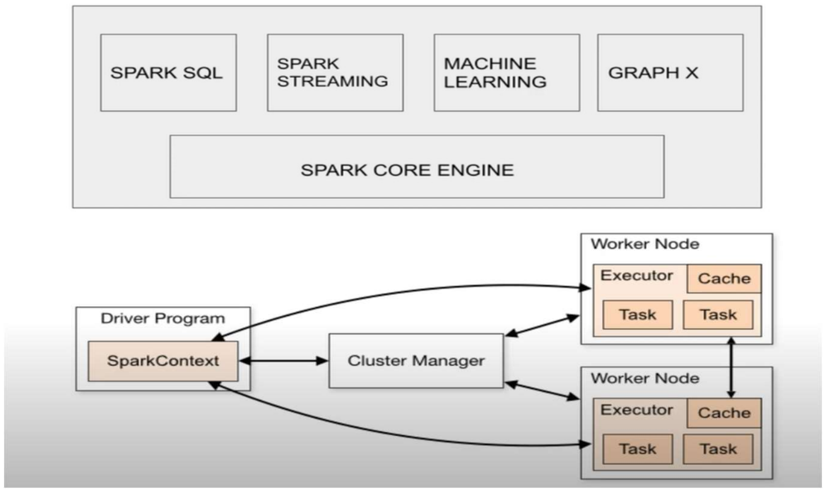
Spark Architecture

1) **What are the main components of Spark Architecture?**

Ans) here are the main components of Spark Architecture in simple terms:

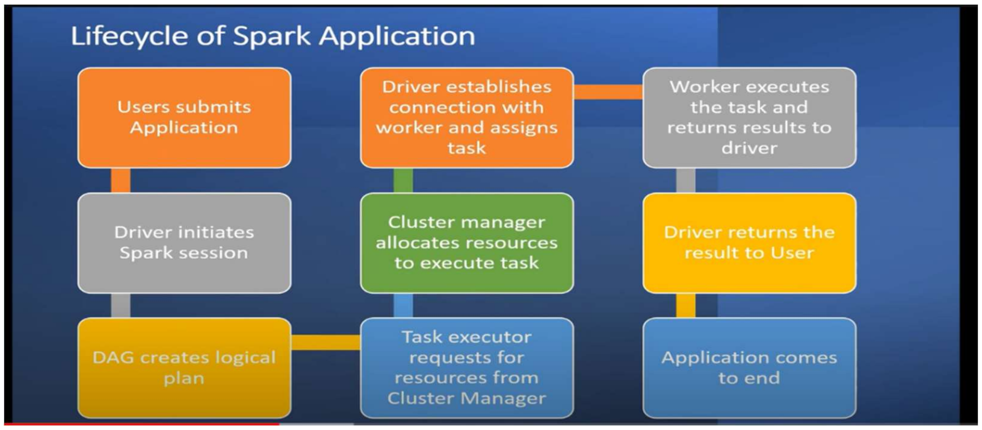
- **Spark Core:** Spark Core is the foundation of Spark. It provides the basic execution engine, data structures, and APIs for programming large-scale data processing applications.
- **Spark SQL:** Spark SQL is a module that provides SQL and structured data processing capabilities to Spark. It allows users to run SQL queries on both structured and unstructured data.
- **Spark MLlib:** Spark MLlib is a module that provides machine learning libraries for Spark. It includes algorithms for classification, regression, clustering, and dimensionality reduction.
- **Spark Streaming:** Spark Streaming is a module that provides real-time streaming data processing capabilities to Spark. It allows users to process streaming data as it arrives.
- **Spark GraphX:** Spark GraphX is a module that provides graph processing capabilities to Spark. It allows users to perform graph operations such as PageRank, connected components, and shortest paths.

Spark Architecture Diagram



- ➔ **Driver and worker process:** These are nothing but JVM process. Within one worker node, there could be multiple executors. Each executor runs its own JVM process.
- ➔ **Application:** It could be single command or combination of multiple notebooks with complex logic. When code is submitted to spark for execution, application starts.
- ➔ **Jobs:** When an application is submitted to spark for execution, application starts.

- ➔ **Stage:** Jobs are divided into stages. If the application code demands shuffling the data across nodes, a new stage is created. The number of stages are determined by the number of shuffling operations. Join is an example of shuffling operation.
- ➔ **Tasks:** Stages are further divided into multiple tasks. All tasks would execute the same logic. Each task would process one task at a time.
- ➔ **Transformations:** It is a kind of operation which will transform DataFrame from one form to another form.
  It transforms the input RDD and creates new RDD until actions are called and transformations are evaluated lazily.
  Ex: Filter, Union etc.
- ➔ **DAG:** Directed Acyclic graph keeps record of all the transformations. For each transformations logical plan is created and lineage graph is maintained by the graph.
- ➔ **Action:** When the data output is needed for developer or for storage purpose action is called. Action would be executed based on the DAG and processes the actual data.
  Ex: Count, Collect and Save etc.
- ➔ **Executor:** Each worker node can consist of many executors. It can be configured by spark setti
- ➔ **Core:** Each executor can consist of multiple cores. This is configurable by spark settings.


Lifecycle of Spark Application
Users submits Application
Driver initiates Spark session
DAG creates logical plan
Task executor requests for resources from Cluster Manager
Cluster manager allocates resources to execute task
Driver establishes connection with worker and assigns task
Worker executes the task and returns results to driver
Driver returns the result to User
Application comes to end

Explanation of Architecture:

- ➔ Apache Spark works on the concept of Divide-and-conquer that is – Master Node and Worker Node.
- ➔ While installing Spark, the **default cluster manager** is installed.
- ➔ Apache Spark supports 4 main **Open Source Cluster Managers** :
- YARN – If we install Hadoop cluster ,then we find **YARN Cluster Manager**
- Mesos
- Standalone
- Kubernetes

In this Architecture we have 3 elements:

1. Master Node (Driver Program)
2. Cluster Manager
3. Worker Node

1. <u>Master Node (Driver Program) Role:</u>

➔ The Master Node is the central control point of a Spark cluster. It is responsible for scheduling tasks, managing resources, and monitoring the cluster.

➔ The Driver Program is the main entry point for a Spark application. It is responsible for creating the **SparkContext**, submitting jobs, and retrieving results.

2. <u>Cluster Manager Role</u> –

Cluster Manager allocates resources to tasks, monitors the health of the cluster, and restarts tasks that fail.

➔ Apache Spark has 4 different **Open Source Cluster Managers** :

1. YARN – If we install Hadoop cluster ,then we find **YARN Cluster Manager**
2. Mesos
3. Standalone
4. Kubernetes

<u>Workflow of Cluster manager in below steps</u>

Step 1: Cluster Manager identifies the **Data processing Task**

↓

Step 2: Divide the Main task into Sub Task

↓

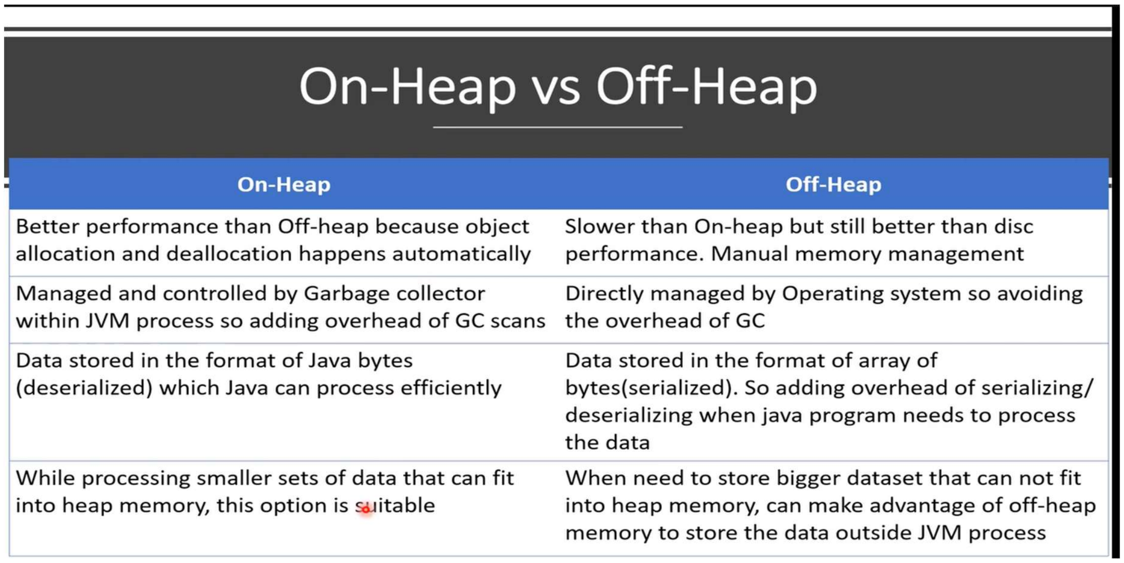
Step 3: Assign or Distribute **Sub Task to available Worker Node** in the Cluster for parallel execution

3. <u>Worker Node:</u>

➔ Worker nodes are the nodes that do the actual work of executing Spark tasks.

➔ They are responsible for fetching data from Cluster Manager, executing tasks, and returning results to the Driver Program.

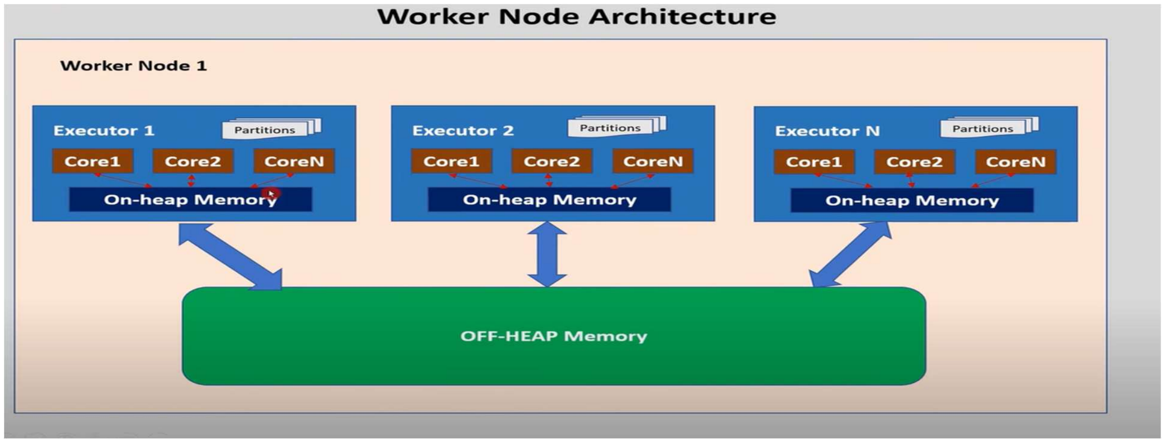
➔ Worker nodes can be physical machines or virtual machines.



On-Heap vs Off-Heap

On-Heap	Off-Heap
Better performance than Off-heap because object allocation and deallocation happens automatically	Slower than On-heap but still better than disc performance. Manual memory management
Managed and controlled by Garbage collector within JVM process so adding overhead of GC scans	Directly managed by Operating system so avoiding the overhead of GC
Data stored in the format of Java bytes (deserialized) which Java can process efficiently	Data stored in the format of array of bytes(serialized). So adding overhead of serializing/ deserializing when java program needs to process the data
While processing smaller sets of data that can fit into heap memory, this option is suitable	When need to store bigger dataset that can not fit into heap memory, can make advantage of off-heap memory to store the data outside JVM process

Worker Node Architecture



- **On-Heap-Memory:** The executor memory that lies within the JVM process managed JVM.
- **Off -Heap-Memory**: The executor memory that lies outside the JVM process managed by the OS.
- Each executor within worker node has access to off-heap memory.
- Off-Heap memory can be used by the spark explicit storing its data.
- The amount of off-heap memory used by the spark to store actual data frames is governed by spark.memory.offHeap.size.
- To enable off-heap memory set spark.memory.offHeap.use to true.
- Accessing off-heap is slightly slower than accessing the on-heap storage but still faster than reading/writing from disk.
- GC (Garbage collector) scan can be avoided by using off-heap memory.

Advantages of Spark Architecture

- ➔ **Speed and efficiency:** Spark is very fast and efficient for processing large datasets. It can process data much faster than traditional MapReduce frameworks.
- ➔ **General-purpose:** Spark is a general-purpose framework, which means that it can be used for a variety of data processing tasks. This includes batch processing, streaming processing, machine learning, and graph processing.
- ➔ **Fault tolerance:** Spark is fault-tolerant, which means that it can recover from failures without losing data or state.
- ➔ **Spark is a distributed framework**, which means that it can run on a **multiple machines**. This makes it possible to process large datasets that would not be possible to process on a single machine.

Disadvantages of Spark Architecture

- ➔ **Memory requirements:** Spark can use a lot of memory, especially when processing large datasets. This can be a problem for machines with low memory.
- ➔ **Complexity:** Spark can be complex to use, especially for developers who are not familiar with distributed computing frameworks.
- ➔ **Cost:** Spark can be expensive to run, especially when processing large datasets. This is because Spark requires a cluster of machines to run.
- ➔ **Not for real-time processing**: It is not as well-suited for real-time processing. It is best suited for batch processing and streaming processing.

Q) How Apache Spark is different from MapReduce?

Ans)

- ➔ **Data processing model:** Spark uses RDDs to process data in parallel, while MapReduce uses key-value pairs.
- ➔ **Programming model:** Spark uses **functional programming** as its programming model and provides a high-level API whereas MapReduce uses **imperative programming** as its programming model and requires more code to be written.
- ➔ **Storage:** Spark stores data **in-memory**. This makes it faster to process data than MapReduce, which **stores data on disk**.
- ➔ **Speed:** Spark is faster than MapReduce. This is because Spark stores data in-memory and uses a more efficient data processing model.
- ➔ Fault tolerance: Spark is more fault-tolerant than MapReduce. This is because Spark can recover from failures without losing data or state. MapReduce is less fault-tolerant because it does not checkpoint data.