Chapter 1: Introduction to Spark

Analytics Tensor

Mahesh KC

mahesh.kc@analyticstensor.com

https://analyticstensor.com

Introduction

- Big Data and Distributed Computing
- Apache Spark
- History of Spark
- Spark Architecture
- Spark Distributed Components
- Spark API
- Spark Users
- Download Spark
- Wrap-up

Big Data and Distributed Computing

- Big Data evolution
- Google Distribute Filesystem
- Big Table
- HDFS (Hadoop Distributed File System)
- MapReduce

In 2006, Yahoo donated Hadoop to Apache Software Foundation. Several components such as Hadoop Common, MapReduce, HDFS and Hadoop YARN has been developed rapidly.

Big Data and Distributed Computing (cont.)

Problem:

- Manage and Administer: It was hard to manage and monitor the cluster.
- MapReduce API: Lot of Java code has to be written for performing simple task.
- Disk I/O: MapReduce heavily uses disk to store intermediate data during mapper and reducer process. The MR job might take from hours to days.
- Only Batch Processing: It only support batch processing. It doesn't have streaming and interactive functionality to support real-time analytics.

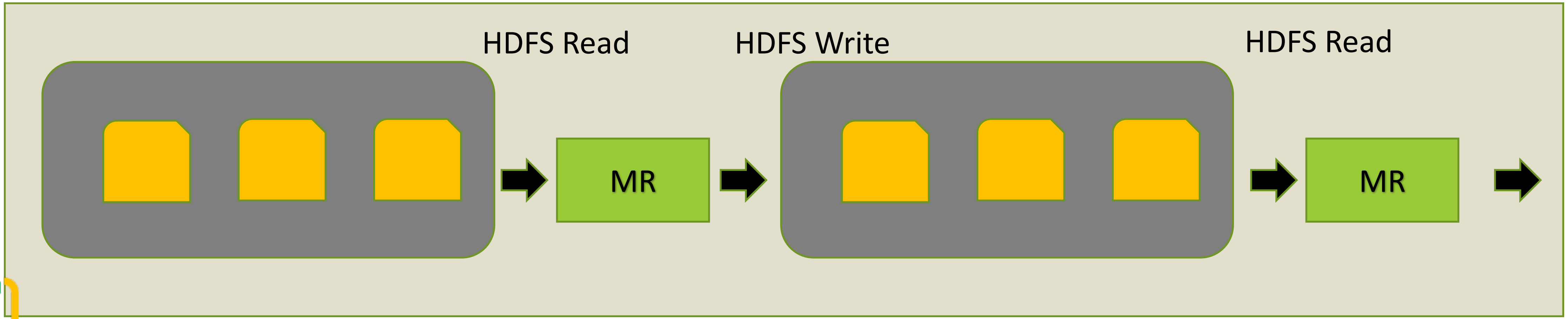


Figure: MapReduce interim state for processing data

Apache Spark

Apache Spark is an unified computing engine that has rich set of libraries and APIs designed for large-scale distribute data processing in a cluster of machines. It supports various programming languages such as Python, Java, Scala and R.

Spark uses in-memory storage for intermediate computation which overcome Hadoop for fast processing.

It has variety of APIs such as:

- Spark SQL: SQL for interactive queries
- Structure Streaming: Stream Processing for real-time data
- MLlib: Machine Learning
- GraphX: Graph Processing

Apache Spark (cont.)

The design philosophy of Spark is based on four characteristics:

1. Speed
2. Ease of Use
3. Modularity
4. Extensibility

Speed: Spark uses Directed Acyclic Graph (DAG) to builds query computations. DAG scheduler and Catalyst query optimizer are used to construct computational graph and execute graph in parallel on multiple workers. Tungsten, the physical execution engine, is used to generate compact code for execution. Eliminating usage of disk I/O and using in-memory help to increase the speed of processing.

Ease of Use: The logical data structure of Spark is build on Resilient Distributed Datasets (RDDs). It provides set of transformations and actions for all other higher-level data abstractions. With these simple programming model, Spark is easy to use and build parallel processing applications.

Apache Spark (cont.)

Modularity: Spark has unified libraries for Spark SQL, Structured Streaming, Machine Learning (MLlib) and graph analytics (GraphX). These modules has well-documented APIs which supports various programming language such as Scala, Java, Python, SQL, and R. We can build Spark application running on one engine using these modules. It helps to build application on single platforms.

Extensibility: Spark decouples its storage and computing. It enables reading data into in-memory with logical data structure (RDD) and DataFrames from many sources. It can read data stored in Hadoop, Cassandra, HBase, MongoDB, Hive, RDBMS etc. Spark *DataReader* and *DataWriters* can be extended to read data from Kafka, Kinesis, Azure Storage, Amazon S3 etc. into its logical data abstraction.

History of Spark

Spark was a research project started in UC Berkeley AMPlab in 2009. Matei Zaharia et.al published the paper entitled “Spark: Cluster Computing with Working Sets” in 2010. During that era, Hadoop MapReduce was the major parallel programming engine for clusters. Matei worked with Hadoop users at UC Berkeley to understand the needs of distributed computing platform. The team designed an API based on functional programming in a new engine that perform efficiently and uses in-memory computation. The first version of Spark only supported batch applications. They implemented Scala interpreter into Spark. They build Shark which was released in 2011 that runs SQL queries over Spark by enabling interactive queries. Several AMPlab teams started MLlib, Spark Streaming, and GraphX. The AMPlab contributed Spark to Apache Software Foundation. The AMPlab team launched Databricks company. Apache Spark community released Spark 1.0 and Spark 2.0 in 2014 and 2016 respectively.

Spark Architecture

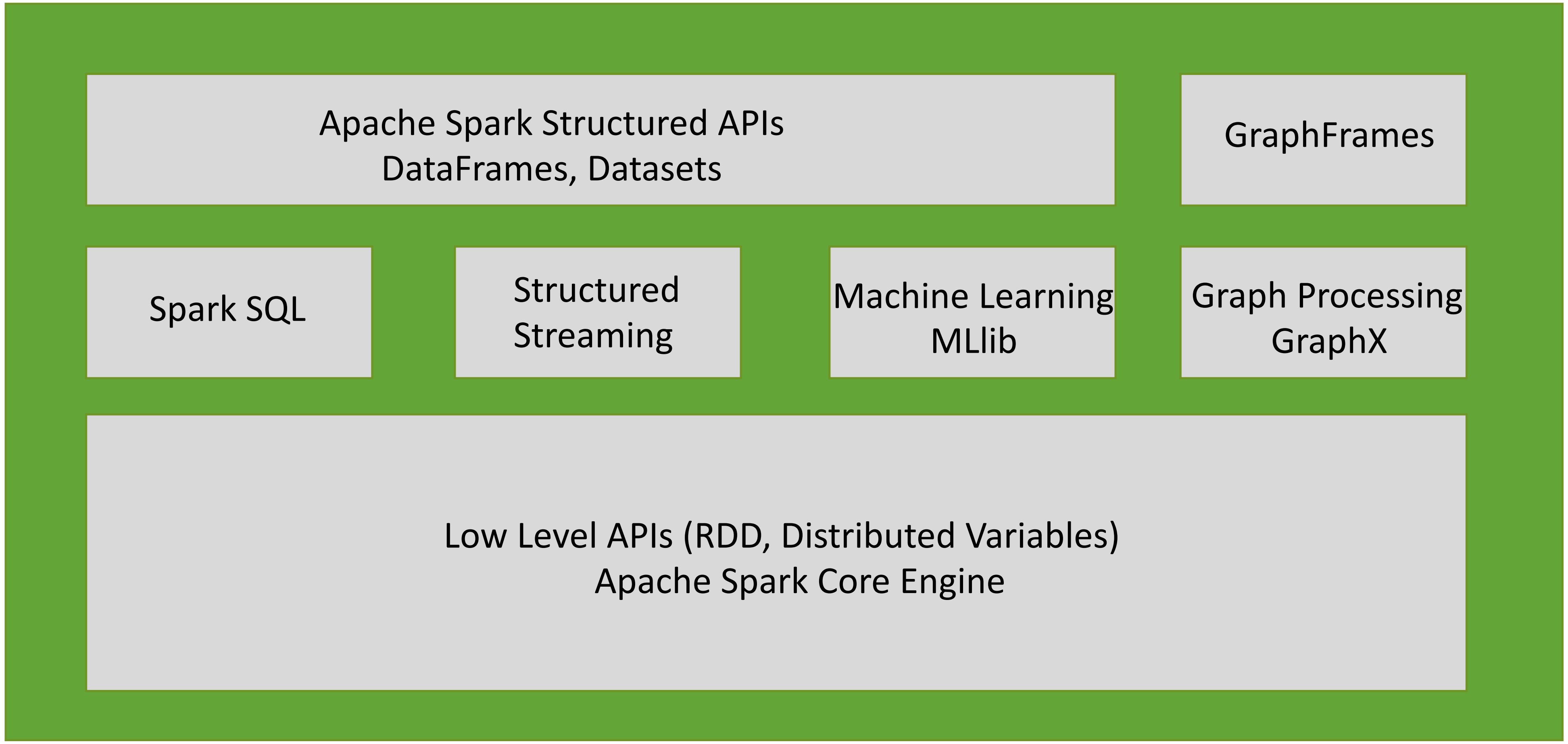


Figure: Apache Spark components and APIs stack

Spark Architecture (cont.)

Cluster is a group of computers that pools the resources from multiple machines. Spark manages and coordinates the execution of tasks on data across cluster of computers. The cluster manager like Spark's standalone cluster manager, YARN, or Mesos are used to manager the resources in cluster. The Spark application is submitted to cluster managers which provide resources and executed the jobs.

Spark Applications: Spark applications consists of a **driver** process and a set of **executor** process.

Driver: Driver process runs the main() function and sits on a node in the cluster. The main task of driver are:

- Maintain information about the Spark Application.
- Respond to user's program or input.
- Analyze, distribute, and schedule work across the executors.

Driver process maintains all information during the lifetime of the application and it is the heart of Spark application.

Spark Architecture (cont.)

Executors: Executors are responsible for performing the actual works assigned by driver. The main task of executors are:

- Execute code assigned by driver.
- Report the state of computation to the driver node.

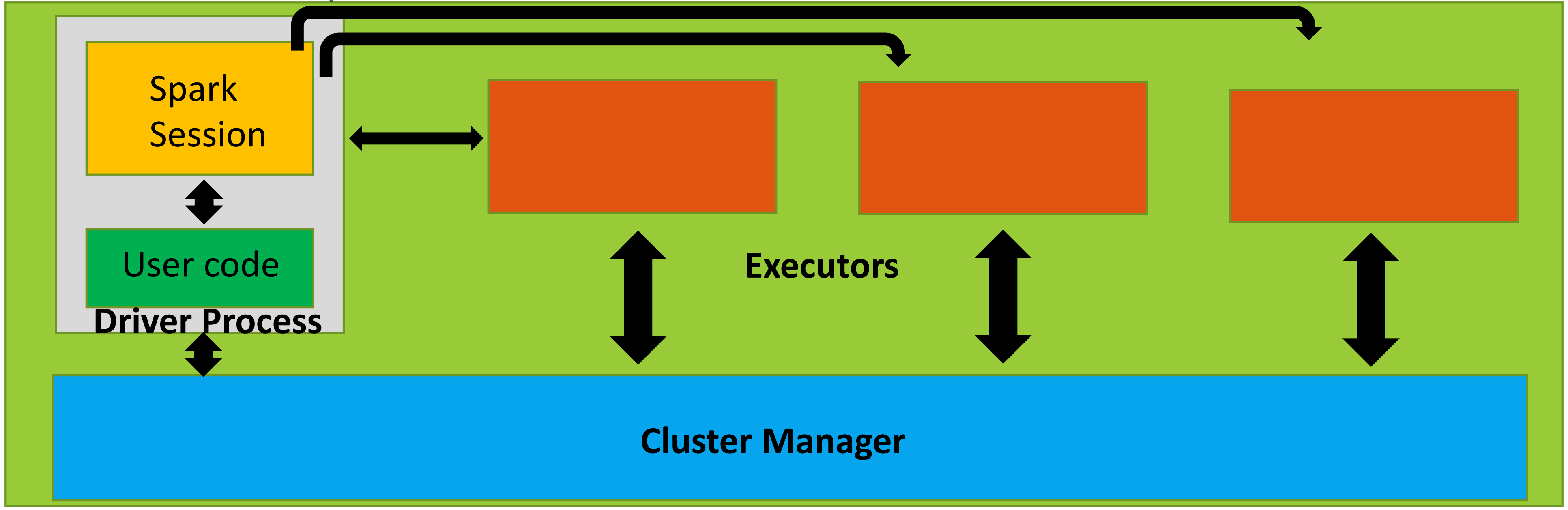


Figure: Spark Application architecture

Spark Distributed Components

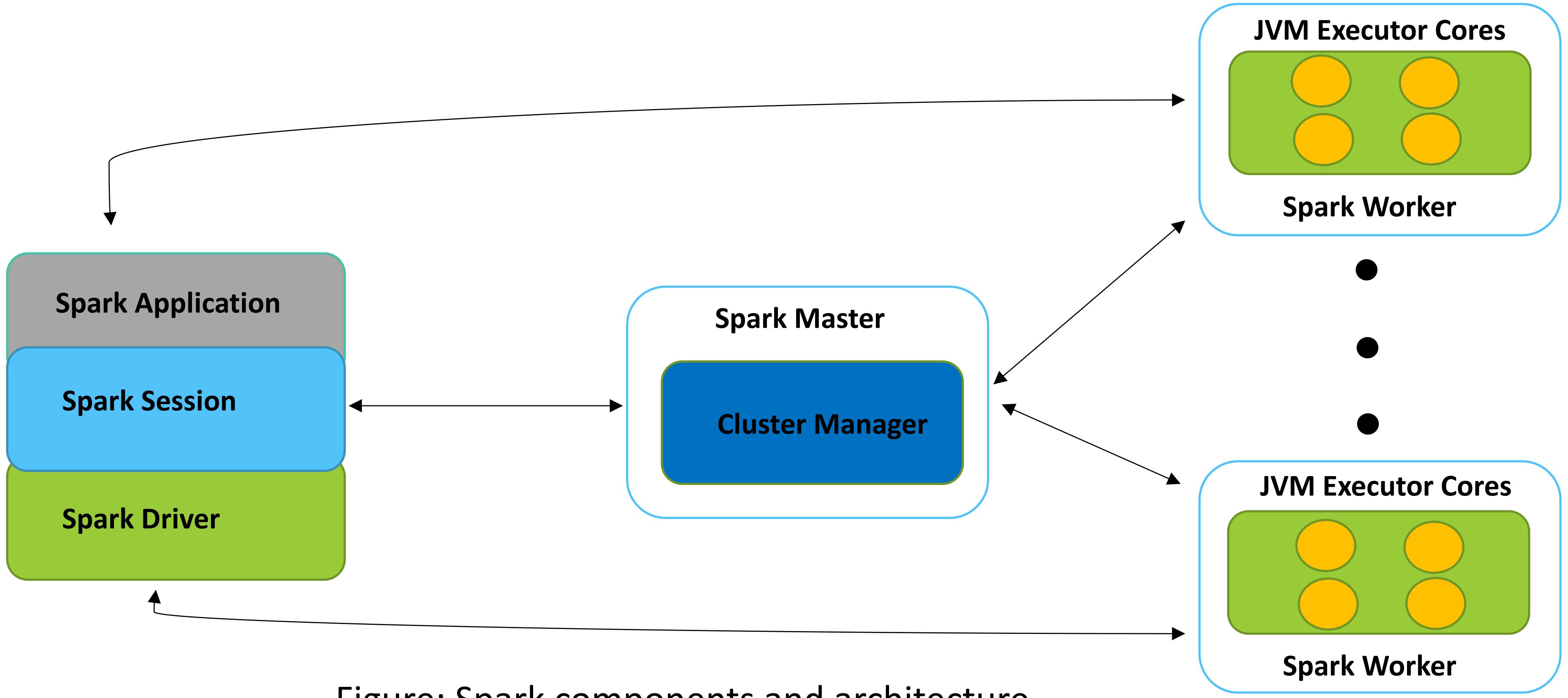


Figure: Spark components and architecture

Spark Distributed Components (cont.)

- Spark Session: Spark Session is an unified channel for all Spark operations and data. It is an entry points to Spark like SparkContext, SQLContext, HiveContext, SparkConf and StreamingContext. With Spark session also allows to:
  - Create JVM runtime parameters
  - Create DataFrames and Datasets
  - Read from data sources
  - Access Catalog Metadata
  - Issue Spark SQL queries

  In Spark application, we can create single SparkSession per JVM and used to do various Spark operations. But in Spark 1.x we have to create individual contexts using multiple boiler code.

Spark Distributed Components (cont.)

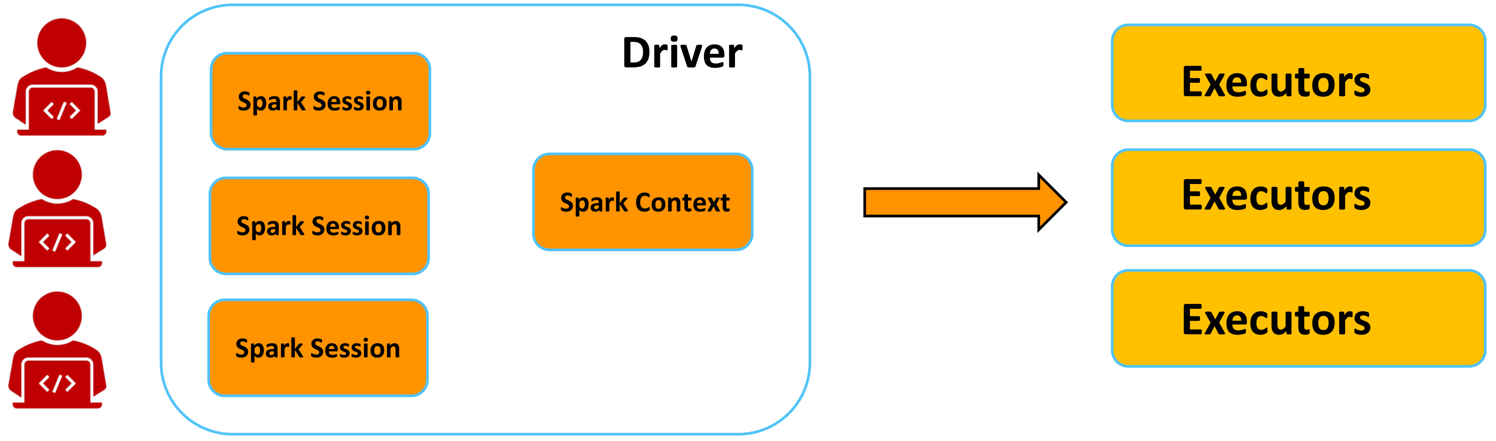


Figure: Spark session and context

Spark Distributed Components (cont.)

- Spark Driver: Spark Driver instantiate a SparkSession for Spark Application. The functions of Spark Driver are:
  - It communicates with Spark Master where the cluster manager runs. It gets Spark workers resources like JVMs, CPU, memory, disk etc. from Spark Master.
  - It requests from Spark Master resources and Spark Executors JVMs.
  - It transforms all the Spark operations into DAG computations.
  - It schedules all the task.
  - It distributes the executions across all the Spark workers.
- Spark Master: Spark Master is a physical node where Cluster Manager runs as a daemon.
- Cluster Manager: Cluster Manager maintains a cluster of machines. It’s daemon communicates with worker daemons on Spark worker nodes.
- Spark Worker: Spark Worker is a physical nodes (machines) in the cluster. It’s main task is to launch Executors where Spark’s tasks runs.

Spark API

- **RDDs:** Resilient Distributed Datasets is a distributed memory abstraction that helps in-memory computation on large clusters with fault-tolerant. It is a parallel data structures where user can control the partitioning of data with the help of its rich set of operators.
- **DataFrames**: DataFrame is the commons structured API. It is a representation data in rows and columns. It is similar to Python DataFrames but the exists on multiple machines by providing more resources. Pythons Pandas Dataframe can be easily converted to Spark Dataframe. The schema defines the columns and its data types.
  - **Partitions**: Partitions is the chunks of data. It is a collection of rows that reside in physical machine in the cluster.
    - If there is single partitions, the parallelism will be only one even though we have multiple executors.
    - If there are many partitions but single executor, the parallelism will be only one since there is single computation resource (i.e. executor).

The physical data is distributed across Spark Cluster. Partitions are stored in HDFS or cloud storage. The default partition size is 64 MB. Although the data is distributed as partition across physical cluster, Spark used them as high-level logical data abstraction (RDD or DataFrame) in memory. Spark worker's Executor reads the partition nearest to its rack.

Spark API (cont.)

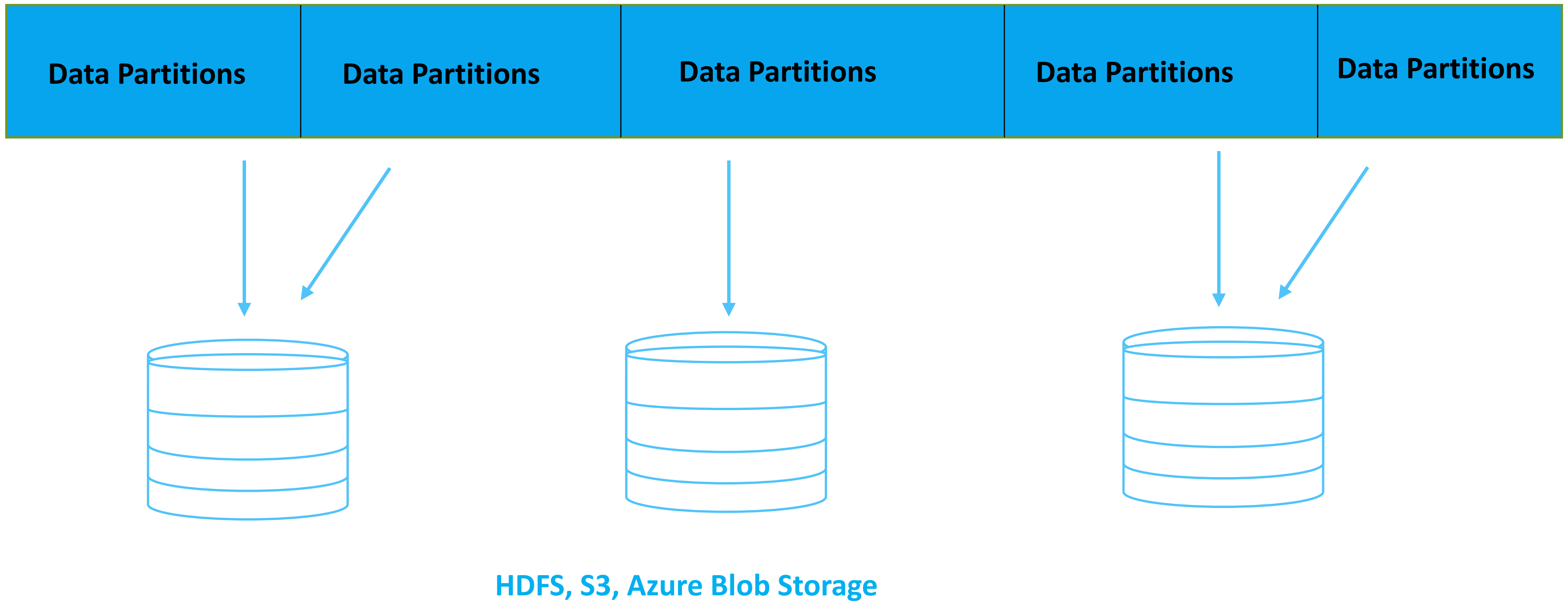


Figure: Distribution of data across physical machines as partitions

Spark API (cont.)

Partitioning is used for efficient parallelism. When the data are broken into chunks of partitions, Spark Executors process the data close to its location to reduce network bandwidth. Each executor’s core has its own data partition.

For example, the code snippet that will break the physical data stored across clusters into 10 partitions, each executor will get partitions to read into its memory.

```
sales_rdd = spark.textFile(“/tmp/sales/sales_2018.txt”, minPartitions = 10)
sales_df = spark.createDataFrame(sales_rdd, schema)
```

Spark API (cont.)

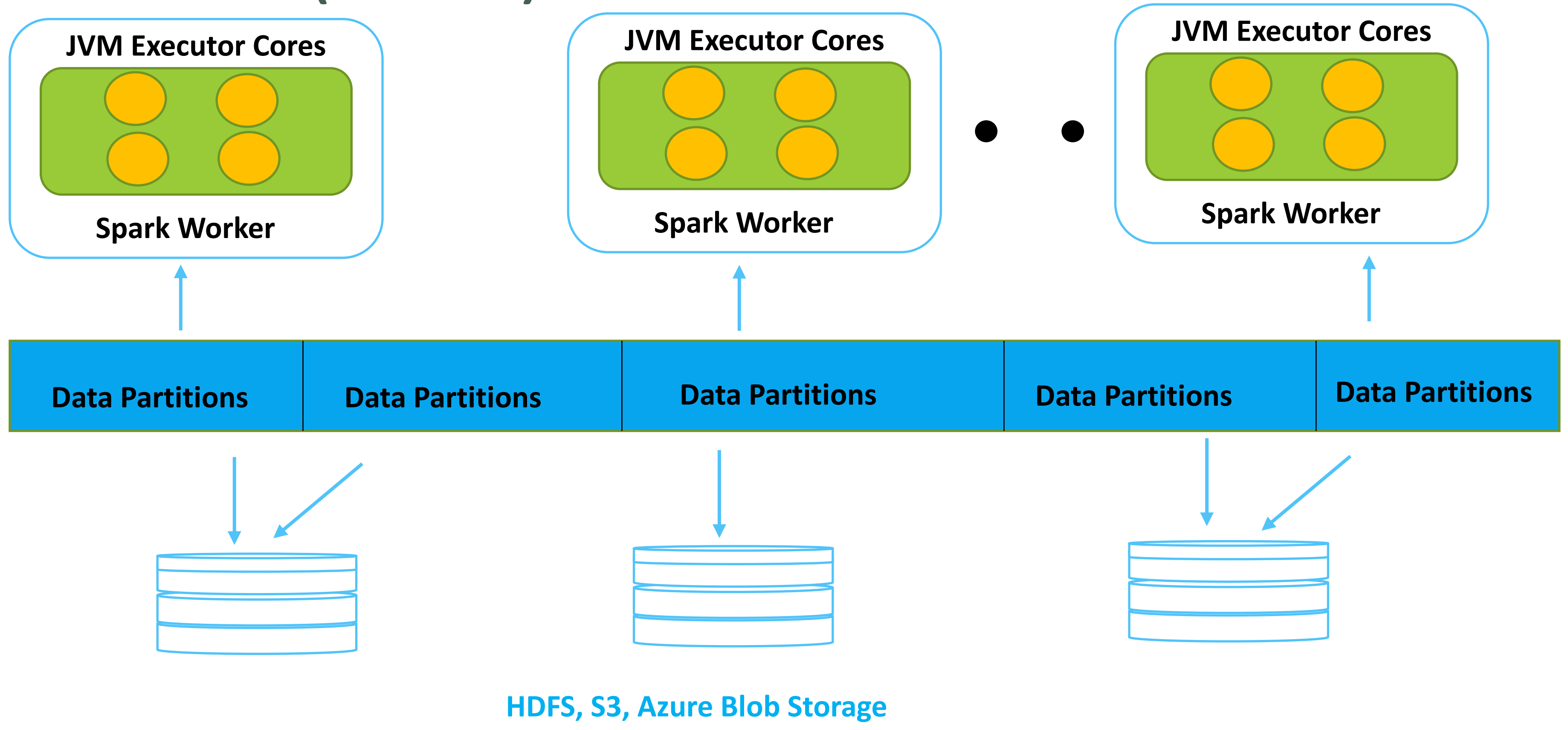


Figure: Specify partition of data for each Executors in cluster

Spark API (cont.)

- **Transformations**: The data structures in Spark are immutable, i.e. they cannot be changed after created. Transformation is used to transform one DataFrame into new DataFrame without changing original dataset. It maintains the data lineage. Types of transformations:
  - Narrow Dependencies: When the input partition contributes to only one output partition then these types of transformation are known as narrow dependencies.
  - Wide Dependencies: When the input partition contributes to many output partitions then these types of transformation are known as wide dependencies.

  The common types of Spark transformation are:- map(), filter(), flatmap(), mapPartitions(), sample(), union(), join() etc.

- **Actions**: An actions instructs Spark to compute result from a series of transformations. Transformation builds the logical transformation plan. An example of actions is *count*, to get total number of records in the DataFrame. Types of actions includes:
  - View data in the console
  - Collect data to native objects in the respective language
  - Write to output data sources.

  The common types of Spark action are:- reduce(), collect(), count(), first(), take(), saveAsTextFile(), countByKey(), foreach() etc.

- **Lazy Evaluation**: Spark waits for the last moment to execute the graph of computation instructions. Instead of executing the code, Spark build the transformation plan. Spark compiles the plan for raw DataFrame to streamline physical plan. This helps to optimize the data flow from end to end. For e.g. if the Spark job has filter at the end of the code then using the plan helps to just filter the data at the beginning instead at end. Spark will automatically optimize by pushing the filter down.

Spark API (cont.)

```
# Read sensor dataset in Spark with Python
sensor_data = spark.read.option("inferSchema", "true").option("header", "true").csv("/data/sensor/dataset.csv")
spark.conf.set("spark.sql.shuffle.partitions", "4")  # set partition size to 4
sensor_data.sort("count").take(2) # return first 2 elements
sensor_data.sort("count").explain()  # print explanation plan
```

Outputs:

```
Array([2019-11-05 12:00,33.75,45,N], [2019-11-05 12:05,35.00,48,Y])
```

Spark API (cont.)

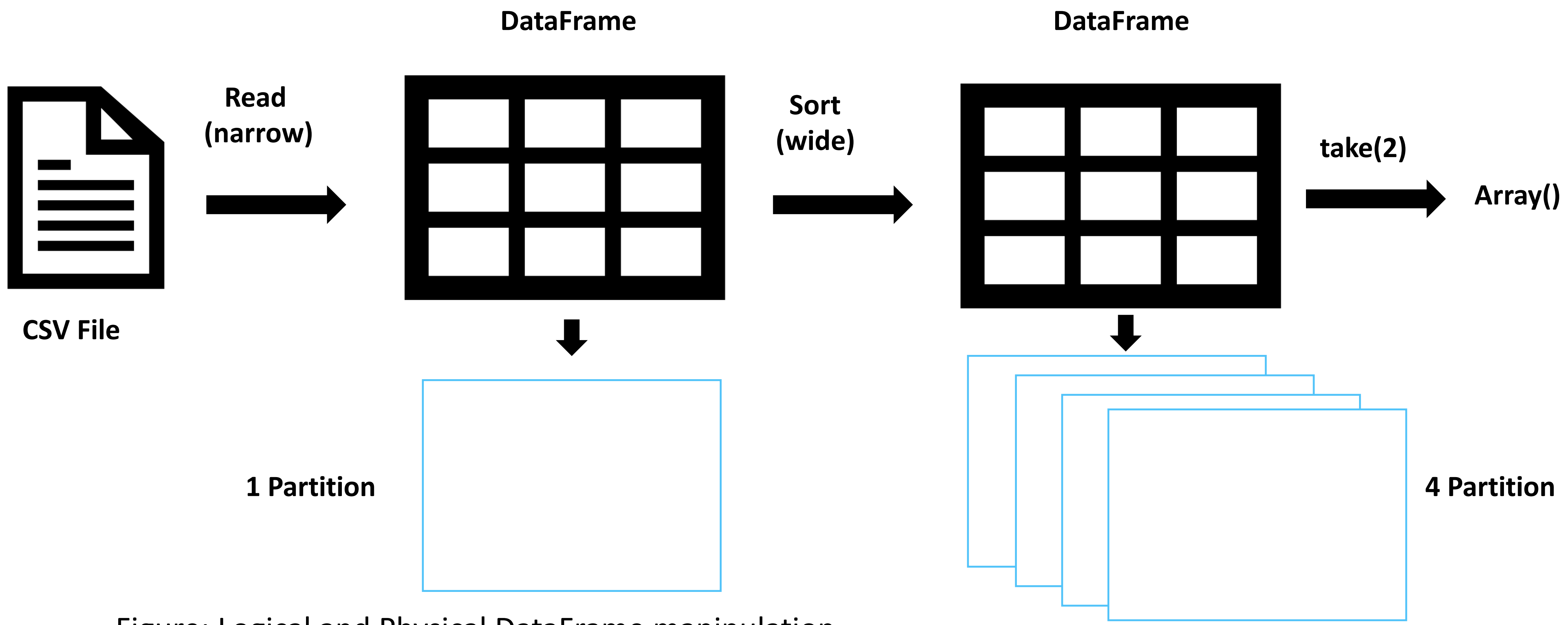


Figure: Logical and Physical DataFrame manipulation

Spark Users

Spark is widely used by data engineers, data scientists, machine learning engineers and deep learning engineers. Spark is used for:

- Process large datasets across a cluster.
- Perform ad hoc and interactive queries.
- Build, train, and evaluated machine learning models.
- Design data pipeline from myriad streams of data.
- Analyze graph datasets.

Download Spark

1. Click the link http://spark.apache.org/downloads.html
2. Choose a Spark release: 3.0.0-preview
3. Choose a package type: Pre-built for Apache Hadoop 3.2 and later
4. Download Spark: spark-3.0.0-preview-bin-hadoop3.2.tgz
5. Move the file to /usr/local/spark. And unzip the tarball

This link will download the spark tarball. It contains all the Hadoop's binaries needed to run Spark in the local mode.

Install PySpark:

conda install pyspark or pip install pyspark

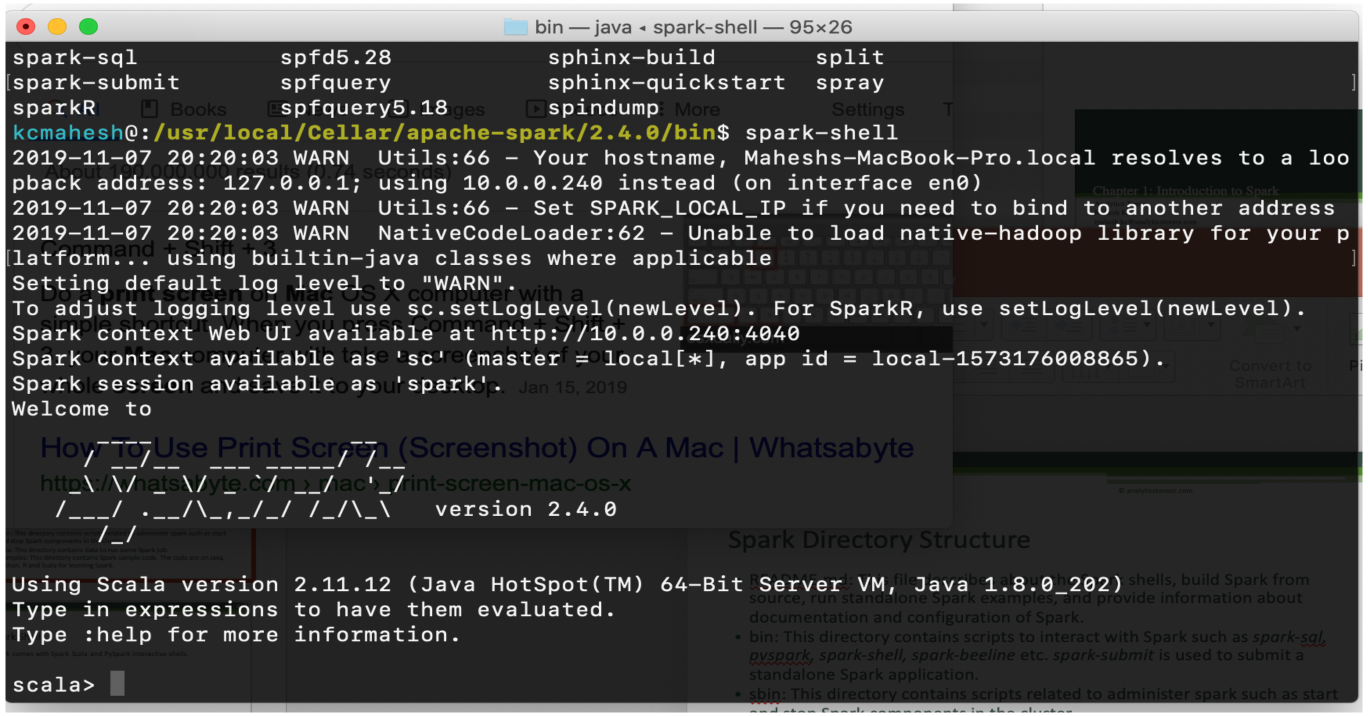
Spark Directory Structure

- README.md: This file describes about the Spark shells, build Spark from source, run standalone Spark examples, and provide information about documentation and configuration of Spark.
- bin: This directory contains scripts to interact with Spark such as *spark-sql, pyspark, spark-shell, spark-beeline* etc. *spark-submit* is used to submit a standalone Spark application.
- sbin: This directory contains scripts related to administer spark such as start and stop Spark components in the cluster.
- data: This directory contains data to run some Spark job.
- examples: This directory contains Spark sample code. The code are on Java, Python, R and Scala for learning Spark.

Spark Shell

Spark comes with default Spark Scala interactive shells. *For all of the tutorial, we'll run Spark in local mode not cluster mode*.

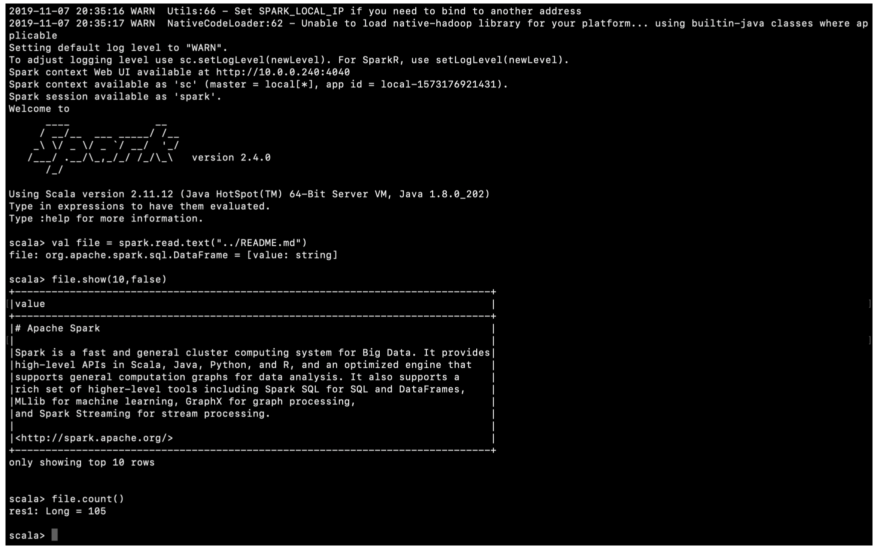
You can also start other shell such as pyspark, spark-sql, spark-beeline etc. located in bin. To exit from shell Control+C, Ctrl+D in mac and window resp.



```
spark-sql              spfd5.28               sphinx-build           split
spark-submit           spfquery               sphinx-quickstart      spray
sparkR                 spfquery5.18           spindump
kcmahesh@:/usr/local/Cellar/apache-spark/2.4.0/bin$ spark-shell
2019-11-07 20:20:03 WARN  Utils:66 - Your hostname, Maheshs-MacBook-Pro.local resolves to a loo
pback address: 127.0.0.1; using 10.0.0.240 instead (on interface en0)
2019-11-07 20:20:03 WARN  Utils:66 - Set SPARK_LOCAL_IP if you need to bind to another address
2019-11-07 20:20:03 WARN  NativeCodeLoader:62 - Unable to load native-hadoop library for your p
latform... using builtin-java classes where applicable
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLevel(newLevel).
Spark context Web UI available at http://10.0.0.240:4040
Spark context available as 'sc' (master = local[*], app id = local-1573176008865).
Spark session available as 'spark'.
Welcome to
      ____              __
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ `/ __/  '_/
   /___/ .__/\_,_/_/ /_/\_\   version 2.4.0
      /_/

Using Scala version 2.11.12 (Java HotSpot(TM) 64-Bit Server VM, Java 1.8.0_202)
Type in expressions to have them evaluated.
Type :help for more information.

scala> 
```

Figure: Spark Scala Interactive Shell

Spark Example using Scala and PySpark Shell



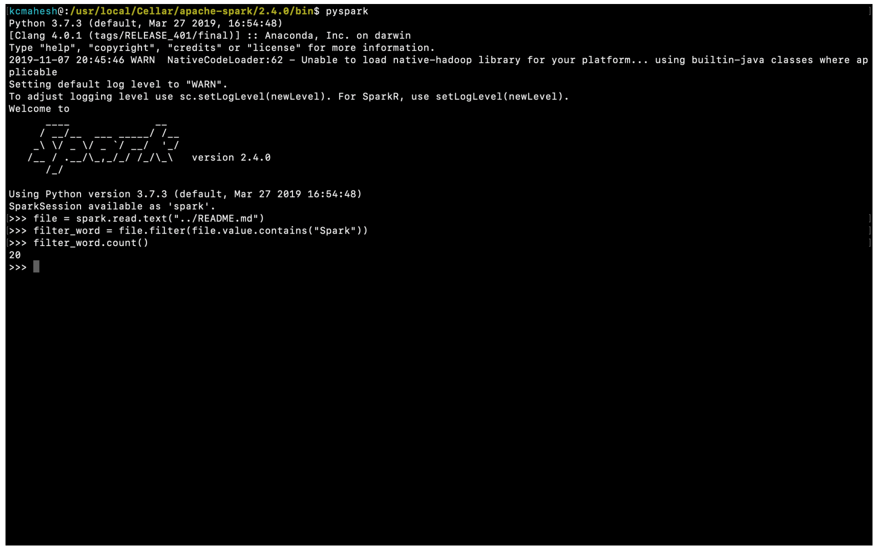
```
2019-11-07 20:35:16 WARN  Utils:66 - Set SPARK_LOCAL_IP if you need to bind to another address
2019-11-07 20:35:17 WARN  NativeCodeLoader:62 - Unable to load native-hadoop library for your platform... using builtin-java classes where ap
plicable
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLevel(newLevel).
Spark context Web UI available at http://10.0.0.240:4040
Spark context available as 'sc' (master = local[*], app id = local-1573176921431).
Spark session available as 'spark'.
Welcome to
      ____              __
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ `/ __/  '_/
   /___/ .__/\_,_/_/ /_/\_\   version 2.4.0
      /_/

Using Scala version 2.11.12 (Java HotSpot(TM) 64-Bit Server VM, Java 1.8.0_202)
Type in expressions to have them evaluated.
Type :help for more information.

scala> val file = spark.read.text("../README.md")
file: org.apache.spark.sql.DataFrame = [value: string]

scala> file.show(10,false)
+--------------------------------------------------------------------------------+
|value                                                                           |
+--------------------------------------------------------------------------------+
|# Apache Spark                                                                  |
|                                                                                |
|Spark is a fast and general cluster computing system for Big Data. It provides|
|high-level APIs in Scala, Java, Python, and R, and an optimized engine that   |
|supports general computation graphs for data analysis. It also supports a     |
|rich set of higher-level tools including Spark SQL for SQL and DataFrames,    |
|MLlib for machine learning, GraphX for graph processing,                      |
|and Spark Streaming for stream processing.                                    |
|                                                                                |
|<http://spark.apache.org/>                                                      |
+--------------------------------------------------------------------------------+
only showing top 10 rows


scala> file.count()
res1: Long = 105

scala>
```

Scala code to count all line from file.

scala> val file = spark.read.text("../README.md")

scala> file.show(10,false)

scala> file.count()



```
kcmahesh@:/usr/local/Cellar/apache-spark/2.4.0/bin$ pyspark
Python 3.7.3 (default, Mar 27 2019, 16:54:48)
[Clang 4.0.1 (tags/RELEASE_401/final)] :: Anaconda, Inc. on darwin
Type "help", "copyright", "credits" or "license" for more information.
2019-11-07 20:45:46 WARN  NativeCodeLoader:62 - Unable to load native-hadoop library for your platform... using builtin-java classes where ap
plicable
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLevel(newLevel).
Welcome to
      ____              __
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ `/ __/  '_/
   /__ / .__/\_,_/_/ /_/\_\   version 2.4.0
      /_/

Using Python version 3.7.3 (default, Mar 27 2019 16:54:48)
SparkSession available as 'spark'.
>>> file = spark.read.text("../README.md")
>>> filter_word = file.filter(file.value.contains("Spark"))
>>> filter_word.count()
20
>>>
```

Python code to count all line that only contains word “Spark”

>>> file = spark.read.text("../README.md")

>>> filter_word = file.filter(file.value.contains("Spark"))

>>> filter_word.count()

Overview of Spark Application

So far, we have downloaded Spark and installed in standalone mode in our pc. We have launched Spark shells and executed Spark Job. We'll discuss on overview of Spark Application for the earlier program.

- Spark Session
- Spark Jobs
- Spark Stages
- Spark Tasks
- Spark UI
- Transformation, Actions, and Lazy Evaluation (*Reading Assignment*)

Spark Session

Spark Session: The core of Spark Application is a Spark driver program. When the application is launched it create Spark Session. In our previous example, Spark shell is a Spark Application and the driver. It means that the driver is a part of shell and it creates the Spark Session. In our local machine the code execute into single JVM but it can be executed parallelly in cluster mode. After Spark Session is created, we can program Spark using its APIs to perform Spark operations.

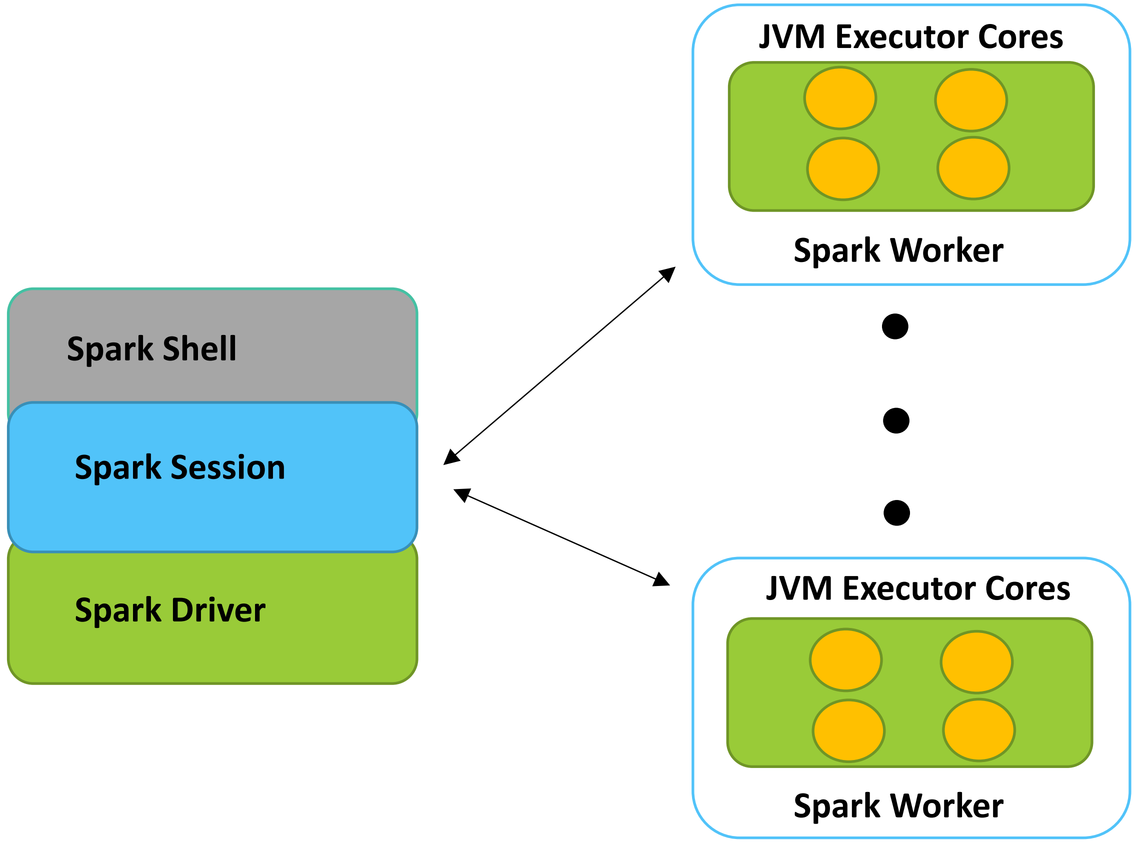


Figure: Spark components communication through Spark shell in cluster mode

Spark Jobs

The Driver converts Spark applications into a single or multiple batch of Spark jobs during interactive session in Spark shells. Each job is then converted into DAG. It is basically, the Spark execution plan. Each node in DAG are either single or multiple Spark stages.

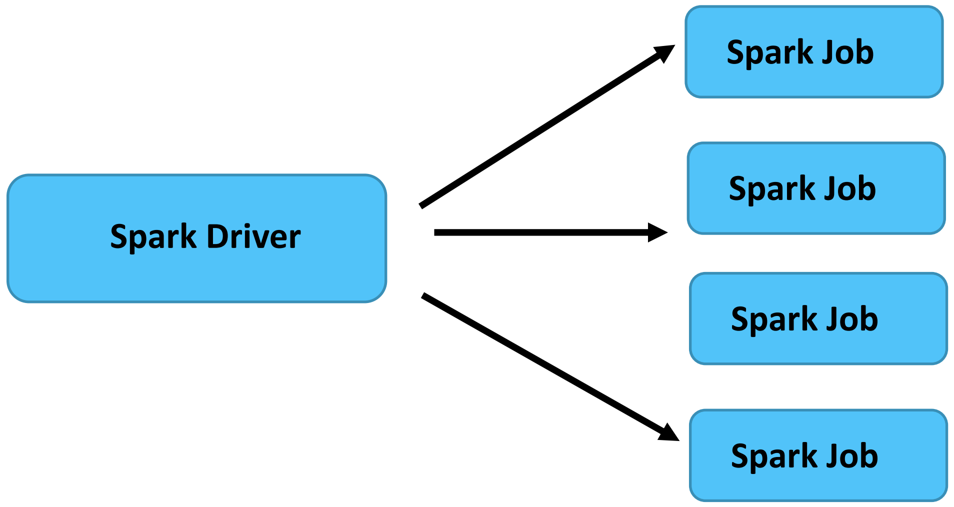


Figure: Spark jobs created by Spark Driver

Spark Stages

During creation on DAG nodes, stages are created based on the operations which can be performed serially or parallelly. All the Spark operation cannot be created on single stage. So, they are divided into multiple stages. Stages are described on the operator’s computation boundaries where data need to be exchange or shuffled across Spark workers.

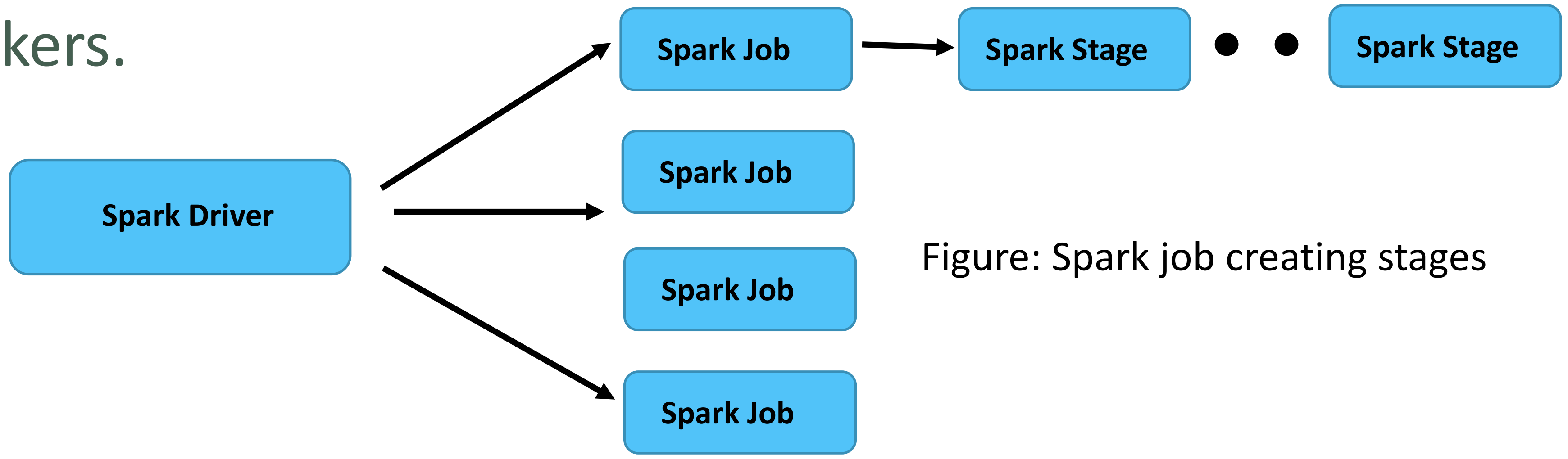


Figure: Spark job creating stages

Spark Tasks

Spark Stage is defined by Spark tasks. Task is the unit of execution. Each tasks is federated across each Spark workers Executor. Each task maps to a single core and works on a single partition of data.

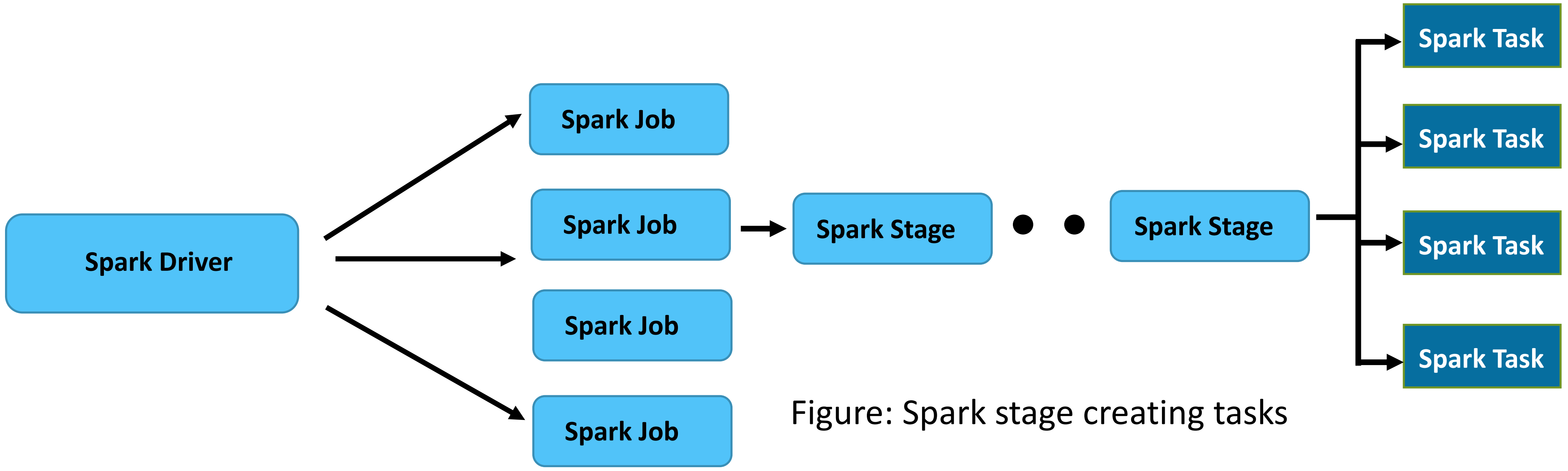


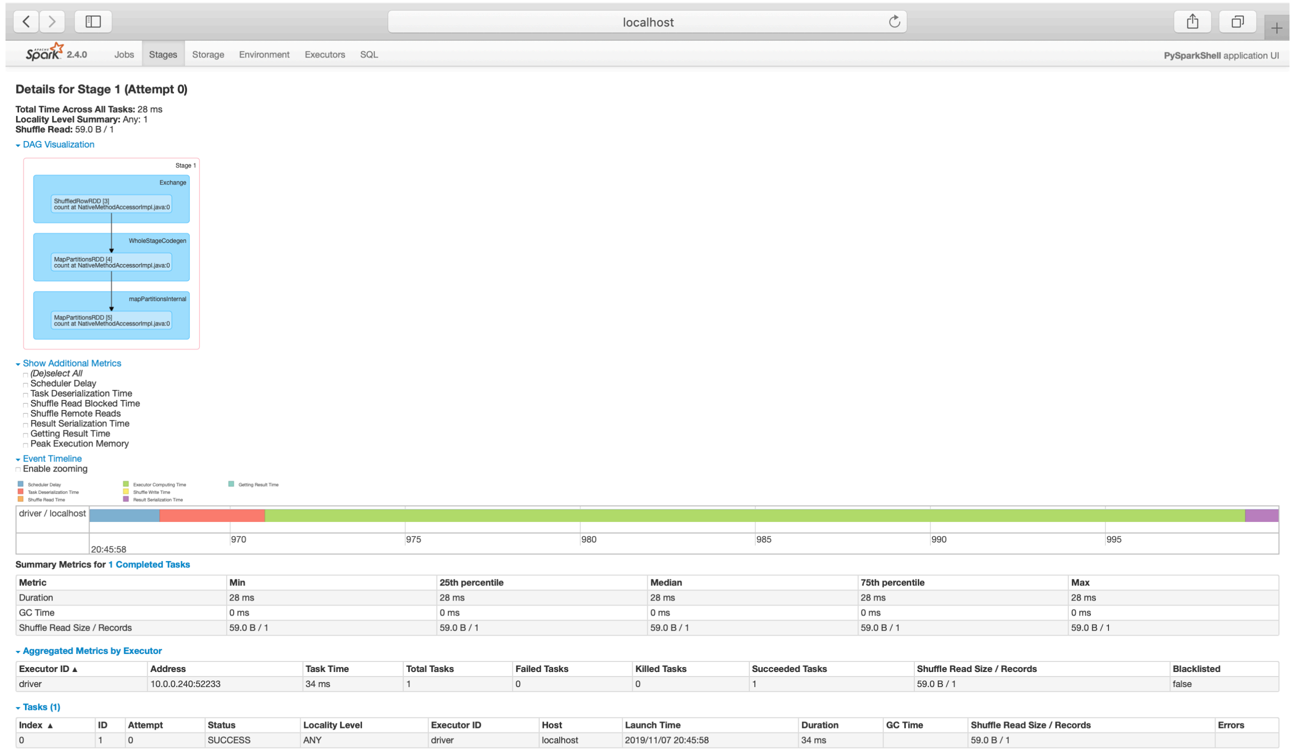
Figure: Spark stage creating tasks

Spark UI

Spark provides a graphical user interface to monitor and debug Spark Applications. The driver launches web UI on localhost at default port 4040 in web browser. Spark UI provides various details such as:

- Environment Information
- List of scheduler stages and tasks
- Summary of RDD and memory usage
- Running Executors
- Spark SQL Queries

Spark UI



Details for Stage 1 (Attempt 0)

Total Time Across All Tasks: 28 ms
Locality Level Summary: Any: 1
Shuffle Read: 59.0 B / 1

DAG Visualization

Show Additional Metrics
- (De)select All
- Scheduler Delay
- Task Deserialization Time
- Shuffle Read Blocked Time
- Shuffle Remote Reads
- Result Serialization Time
- Getting Result Time
- Peak Execution Memory

Event Timeline
- Enable zooming

Summary Metrics for 1 Completed Tasks

Metric	Min	25th percentile	Median	75th percentile	Max
Duration	28 ms	28 ms	28 ms	28 ms	28 ms
GC Time	0 ms	0 ms	0 ms	0 ms	0 ms
Shuffle Read Size / Records	59.0 B / 1	59.0 B / 1	59.0 B / 1	59.0 B / 1	59.0 B / 1

Aggregated Metrics by Executor

Executor ID ▲	Address	Task Time	Total Tasks	Failed Tasks	Killed Tasks	Succeeded Tasks	Shuffle Read Size / Records	Blacklisted
driver	10.0.0.240:52233	34 ms	1	0	0	1	59.0 B / 1	false

Tasks (1)

Index ▲	ID	Attempt	Status	Locality Level	Executor ID	Host	Launch Time	Duration	GC Time	Shuffle Read Size / Records	Errors
0	1	0	SUCCESS	ANY	driver	localhost	2019/11/07 20:45:58	34 ms		59.0 B / 1	

Click http://localhost:4040/ in your browser to view Spark UI.

Try Spark on Databricks. https://databricks.com/try-databricks

Wrap-up

- Important points.
- Q & A.