Introduction

What is Spark?

- Apache Spark is a distributed general-purpose cluster computing platform for large-scale data processing and it is designed to be fast

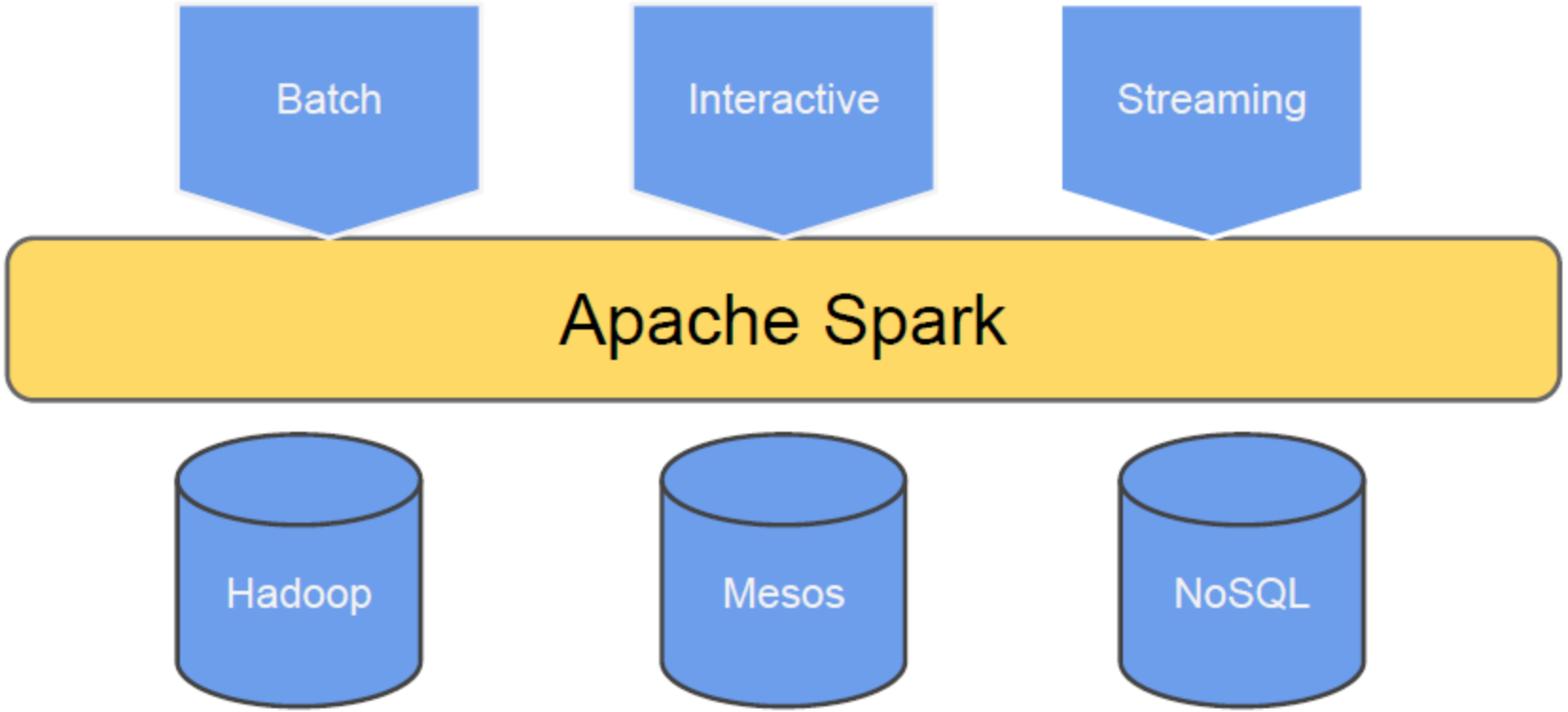
- Developed by researchers at AMP Lab at University of Berkeley who later founded Databricks Inc.
- Open sourced in 2010 under Apache
- It is written in Scala

- Focused on
  - Interactive querying
  - Iterative programs
  - Unifying real time and batch processing

Salient Features / Benefits of Spark

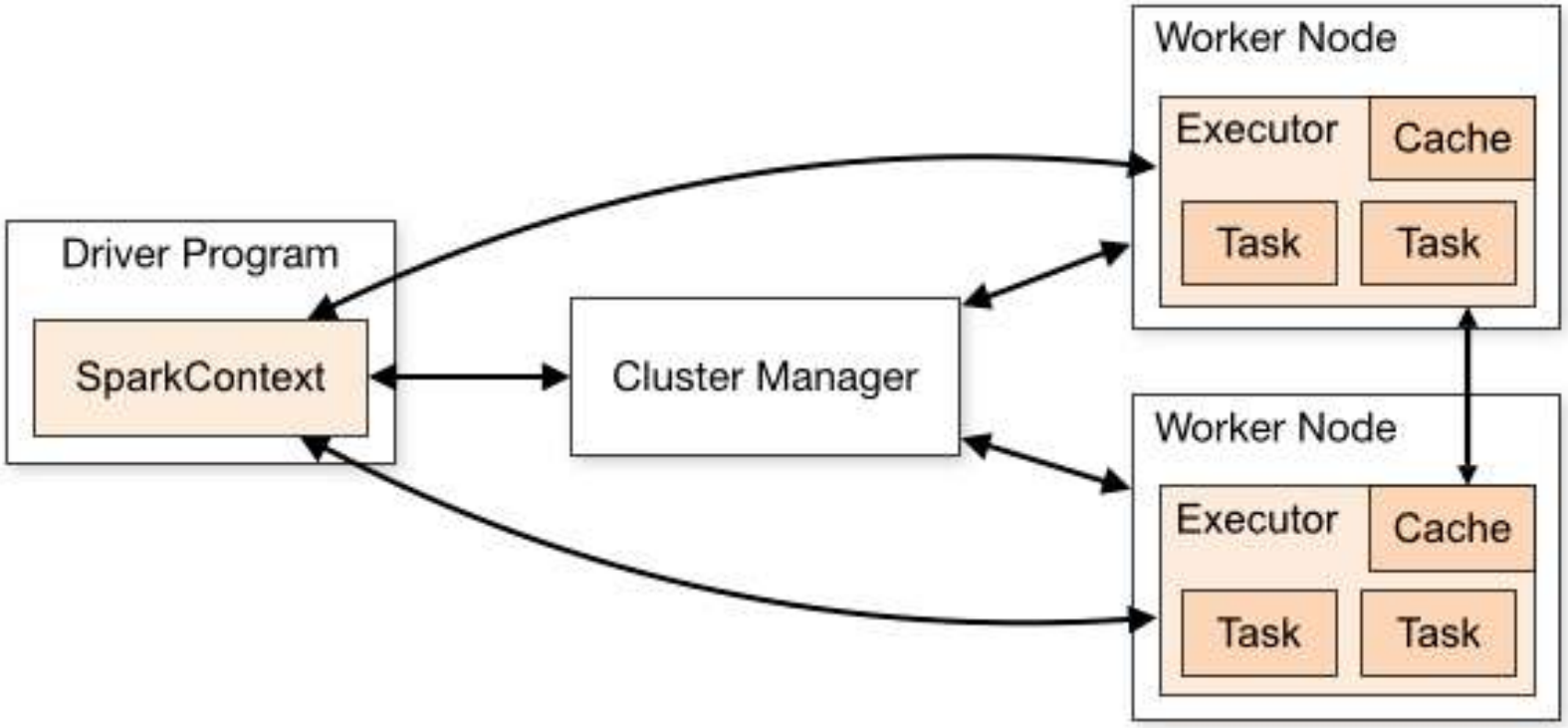
- Speed
  - Spark extends the popular MapReduce model to efficiently support more stages of computations
  - Spark runs most computations in memory and so the system is way faster than MapReduce
- Runs everywhere
  - Spark runs on a standalone cluster, Hadoop cluster, Mesos or in the cloud.
  - It can access diverse data sources including HDFS, Cassandra, HBase and AmazonS3
  - It is designed to be highly accessible, offering simple APIs in Scala, Python, Java, R with rich built-in libraries and supports SQL
- Unified platform
  - Combines SQL, streaming, and complex analytics.
  - Spark powers a stack of libraries including SQL and DataFrames, Spark Streaming and MLlib for machine learning algorithms.

Unified Platform



Architecture

- Spark uses master/slave architecture.
- Spark applications run as independent sets of processes on a cluster, coordinated by the SparkContext object in your main program (called the driver program).
- SparkContext connects to the cluster manager which allocates resources across the cluster.
- Once connected, Spark acquires executors on nodes in the cluster.
- It then sends the application code and data to the executors.
- Finally, SparkContext sends the tasks which need to be run to the executors, which are run in parallel on the data.



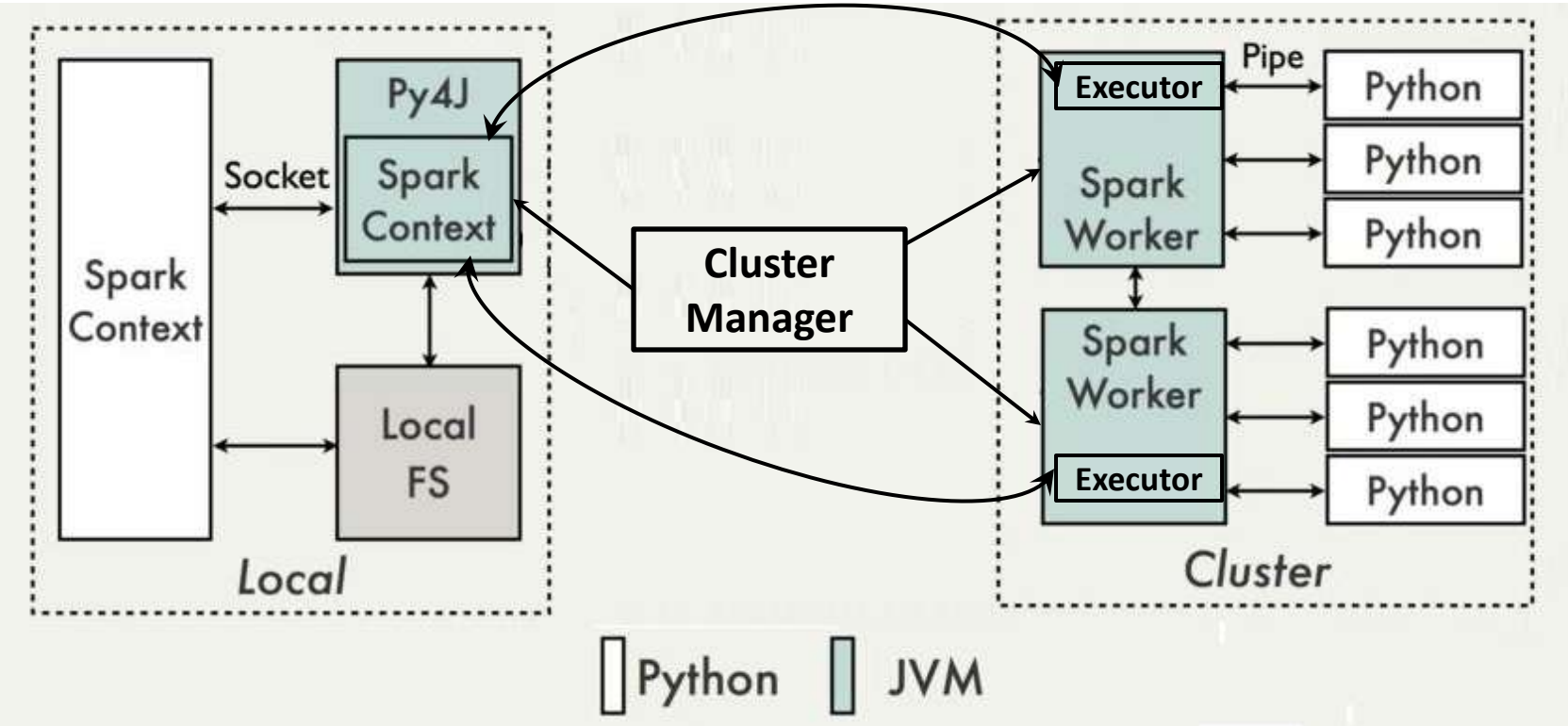
- So, as mentioned SparkContext of the Driver program talks to a coordinator called cluster manager and starts executors that run the give tasks and store the data for the application.

- Driver, which is a JVM (Java Virtual Machine) process, hosts SparkContext for the Spark application and schedules tasks on the cluster.
- Each application gets its own set of executor processes, which stay up for the duration of the whole application and run tasks in multiple threads.
  - So the applications are isolated from each other, on both the driver side (each driver schedules its own tasks) and executor side (tasks from different applications run in different JVMs).
  - And data cannot be shared across different Spark applications.
- The driver is accessible over network to the worker nodes. It listens for requests and accepts incoming connections from its executors throughout its lifetime.
- Executor is a distributed agent that is responsible for executing tasks.
- Executors report heartbeat and task progress to the driver.

- PySpark is built on top of Spark's Java API.
- In the Python driver program, SparkContext uses Py4J to launch a JVM process and creates a Java SparkContext.

- Py4J is a library written in Python and Java. It enables Python programs to dynamically access Java objects of a JVM process.

- Py4J is only used on the driver for local communication between the Python and Java SparkContext objects.
- On the worker machines, Python objects launch Python sub processes, communicate with them using pipes sending the code & the data to be processed.



- In the case of PySpark the flow can be pictured as above.

Running a Spark Application

- We can write an application in Python and run it on Spark using spark-submit command/script which is provided with Spark installation.
- The command syntax is as below:

```
spark-submit --master <master-url> <application.py>
  [application-arguments]
```

- The argument `--master <master-url>` specifies the cluster manager to be used depending on the cluster on which Spark is deployed.
- Also, we can use the argument `--py-files` to add .py, .zip or .egg files, if have multiple Python files in the application, by packaging them into a .zip or .egg file.

- Spark can be deployed on different types of cluster environments as mentioned earlier. For example:
  - Standalone cluster - This refers to an independent cluster that does not use any distributed software framework like Hadoop or Mesos. It uses a simple cluster manager that is included with Spark.
  - Hadoop cluster - A cluster on which Hadoop is running. So HDFS (Hadoop Distributed File System) is used for storage of data in this case. Yarn will be the cluster manager for allocating resources on the cluster nodes to run the application.
  - Also we can run Spark in non-distributed or local mode as well.

- Spark-submit can use all the supported cluster managers depending on the deployment mode of Spark.
- Let us see how Spark is deployed in different modes and how we can run an application on each mode.

Standalone cluster:

- We can run the application with the command:

```
spark-submit --master spark://ubuntu:7077
  myapp.py
```

- The argument --master specifies the host name and port on which Spark is deployed in Standalone mode as spark-submit needs to use the built-in cluster manager.

We can deploy Spark in standalone cluster mode as follows:

- Cluster launch scripts are usually available in the directory: $SPARK_HOME/sbin. In our VM $SPARK_HOME is /usr/local/spark, the path of the scripts is: /usr/local/spark/sbin.
- We can give the command below to start Spark in standalone cluster mode:

```
$SPARK_HOME/sbin/start-all.sh
```

- The command runs Spark in standalone cluster mode on our virtual machine which acts as the master and also as the slave.
- We can check that Spark is running in cluster mode using the command below:

```
$ jps
3120 Master
3243 Worker
3319 Jps
```

- We can now run the application with the spark-submit command as mentioned above.

Hadoop cluster:

- On a Hadoop cluster we can run the application with the command:

```
spark-submit --master yarn myapp.py
```

- The argument –master specifies that spark-submit needs to use yarn as the cluster manager as Spark is deployed on a Hadoop cluster.
- Several other arguments are also available for the command like `--driver-memory, --executor-memory` etc.

Deploying spark in Hadoop mode:

- You first need to start hadoop i.e. all the hadoop daemons (background processes) using the commands:

```
$ $HADOOP_HOME/sbin/start-dfs.sh
$ $HADOOP_HOME/sbin/start-yarn.sh
```

- We can verify that Hadoop daemons are initiated by giving jps command.

```
$ jps
3492 NameNode
3613 DataNode
3878 SecondaryNameNode
4110 ResourceManager
4236 NodeManager
4442 Jps
```

- Another prerequisite is that the environment variable `HADOOP_CONF_DIR` needs to be set and should be pointing to the configuration directory of our hadoop installation. You can set the variable as follows:

```
$ export HADOOP_CONF_DIR=$HADOOP_HOME/etc/hadoop
```

- The above specifies where spark-submit can find out Hadoop configuration settings such as the hostname and port number.
- Since the spark application looks for the input files on HDFS, we need to make sure that any input files are available on HDFS.
- Once these are done, we can run the application using spark-submit as mentioned above.

- Also we can run Spark application with Spark being in a non-cluster mode or *local* mode as follows:

```
spark-submit –master local[2] myapp.py
```

- This command runs spark as a single JVM process in non-cluster mode locally. Here `[2]` refers to the number of worker threads.
- We can use `[*]` to specify as many threads as available on the local system.
- If we check with jps command we will a spark-submit job running as a single process.
- For most of the testing we can run the application in this mode.

PySpark Shell

- *pyspark* shell allows us to use Spark with Python interactively.
- Like a typical REPL shell, in *pyspark* we can write Python code and run it interactively
- Most of the development work can be done in pyspark shell
- PySpark basically is a standalone Spark application that runs in a JVM process named SparkSubmit just as any application
- In the PySpark shell, SparkContext is already created and is made available in the variable named sc.
- Creating our own SparkContext object is not required and does not work.
- We can set master (cluster manager) using the --master argument and SparkContext will connect to it. For example:

```
pyspark --master local[4]
pyspark --master spark://ubuntu:7077
pyspark --master yarn
```

PySpark with Jupyter Notebook:

- We can use pyspark through Jupyter Notebook also.
- Since it s a popular and convenient tool for development purposes it is installed and made available on the Virtual Machine set up.
- In jupyter notebook we need to run the following commands to get SparkContext in the variable sc:

```
import findspark
findspark.init("<spark install directory>")
import pyspark
sc = pyspark.SparkContext(appName='myApp')
```

- Once the above lines are run we can go ahead with Python code of the application.
- PySpark shell or Jupyter Notebook are convenient and effective tools for interactive applications and for developing & testing Spark applications in Python