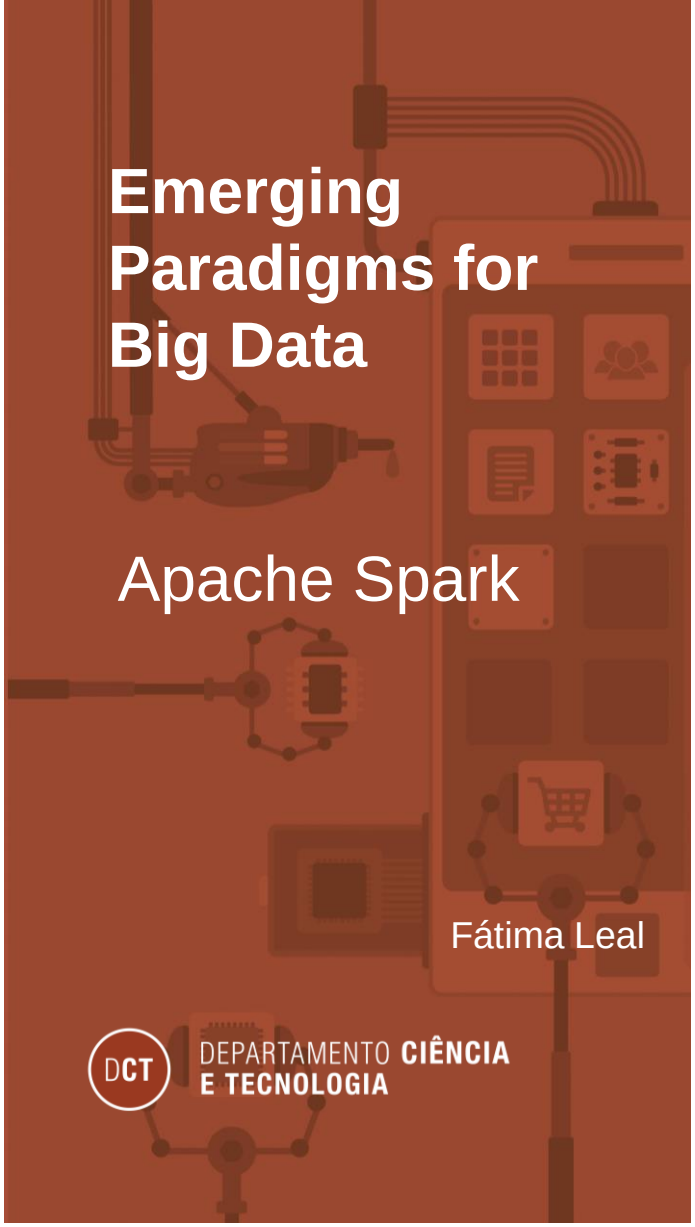


UNIVERSIDADE PORTUCALENSE

Content

- Introduction to Spark
- Spark Core Concepts and Architecture
- Spark Unified Stack
- Apache Spark Applications
- Kafka + Cassandra + Spark
- Example/Exercises

Overview

- **Spark** is a general **distributed data processing engine** built for speed, ease of use, and flexibility.

- The Apache Spark website claims it can run a certain data processing job up to 100 times faster than Hadoop MapReduce.

- It offers more than 80 high-level, commonly needed data processing operators to make it easy for developers, data scientists, and data analysts to build all kinds of interesting data applications.

Overview

- In terms of flexibility, Spark offers a single unified **data processing stack** that can be used to solve multiple types of data processing workloads, including batch **processing**, **interactive queries**, **iterative processing** needed by **machine learning algorithms**, and **real-time streaming processing** to extract actionable insights at near real-time.

- A big data ecosystem consists of many pieces of technology including a distributed storage engine called HDFS, a cluster management system to efficiently manage a cluster of machines, and different file formats to store a large amount of data efficiently in binary and columnar format.

History

- Spark started out as a research project at Berkeley AMPLab in 2009. The researchers observed the inefficiencies of the Hadoop MapReduce framework in handling interactive and iterative data processing use cases, so they came up with ways to overcome those inefficiencies by introducing ideas such as in-memory computation and an efficient way of dealing with fault recovery.

- Once this research project proved to be a viable solution that outperformed MapReduce, it was open sourced in 2010 and became the Apache top-level project in 2013

History

- A group of researchers working on this research project got together and founded a company called **Databricks**.
- The creators of Spark selected the **Scala programming language** for their project because of the combination of Scala's conciseness and static typing.
- Now Spark is considered to be one of the largest applications written in Scala, and its popularity certainly has helped Scala to become a mainstream programming language.

Spark Core and Architecture

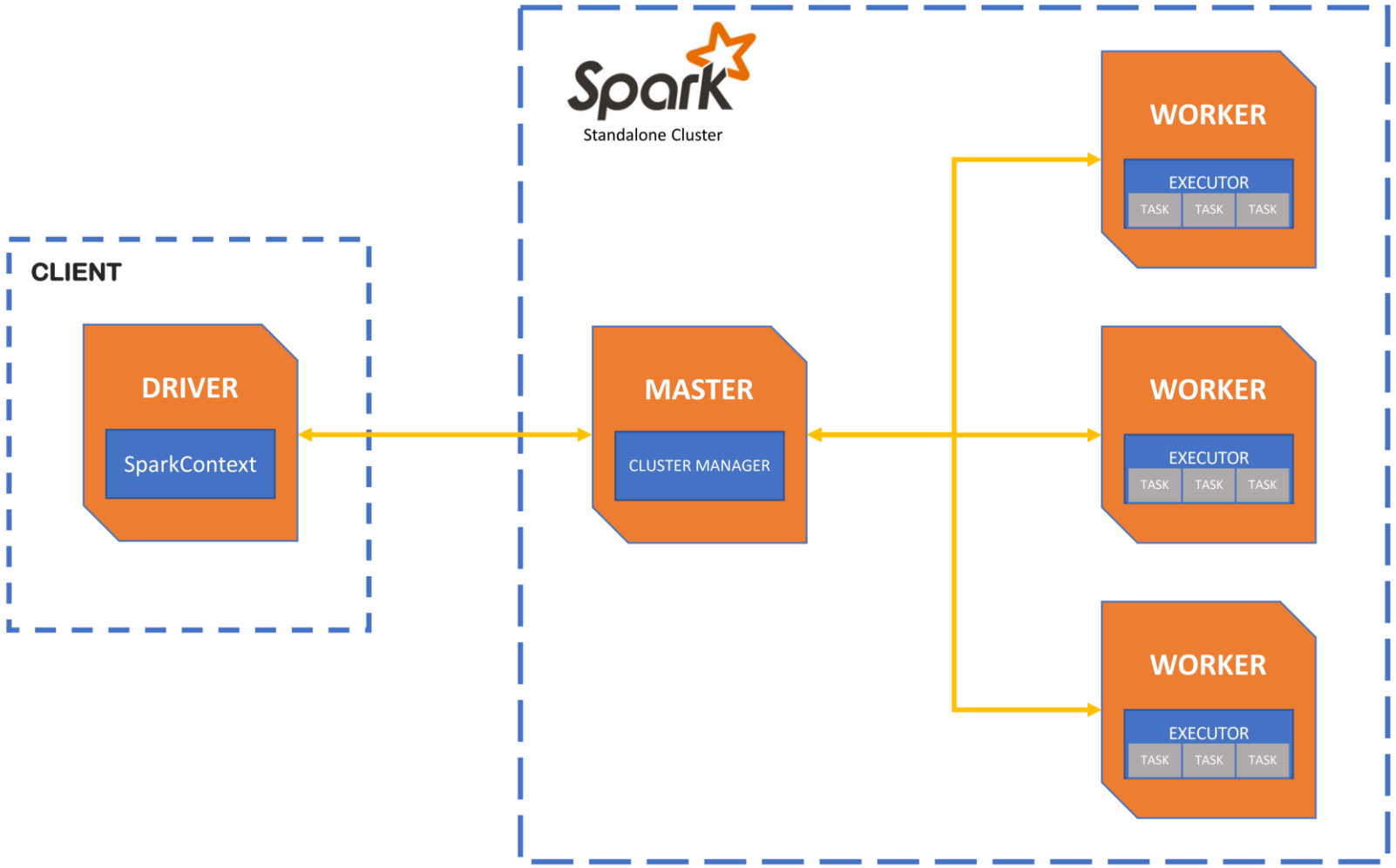
- It is important to have a high-level understanding of the core concepts and the various core components in Spark.

- The main components are **Spark clusters**, the **resource management system**, **Spark applications**, **Spark drivers** and **Spark executors**.

Spark Clusters

- Spark is essentially a distributed system that was designed to process a large volume of data efficiently and quickly.
- This distributed system is typically deployed onto a **collection of machines**, which is known as a **Spark cluster**.
- A cluster size can be as small as a few machines or as large as thousands of machines.
- The largest publicly announced Spark cluster in the world has more than 8000 machines.

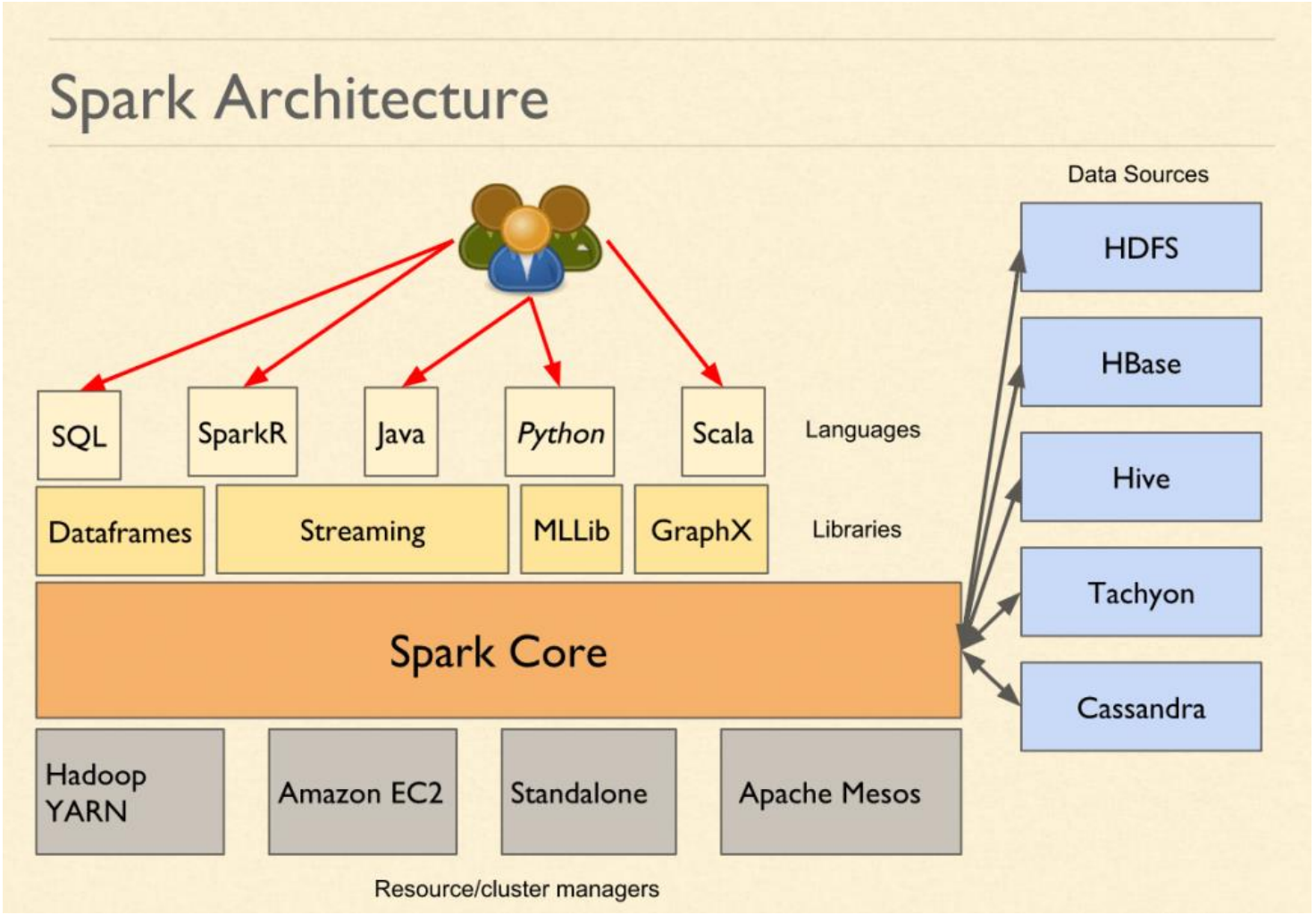
Spark Clusters



Resource Management System

- To efficiently and intelligently manage a collection of machines, companies rely on a resource management system such as Apache YARN or Apache Mesos.
- The two main components in a typical resource management system are the **cluster manager and the worker**.
- Each worker offers resources (memory, CPU, etc.) to the cluster manager and performs the assigned work.
- The cluster manager knows where the workers are located, how much memory they have, and the number of CPU cores each one has.
- One of the main **responsibilities of the cluster manager is to orchestrate the work by assigning it to each worker**.

Resource Management System



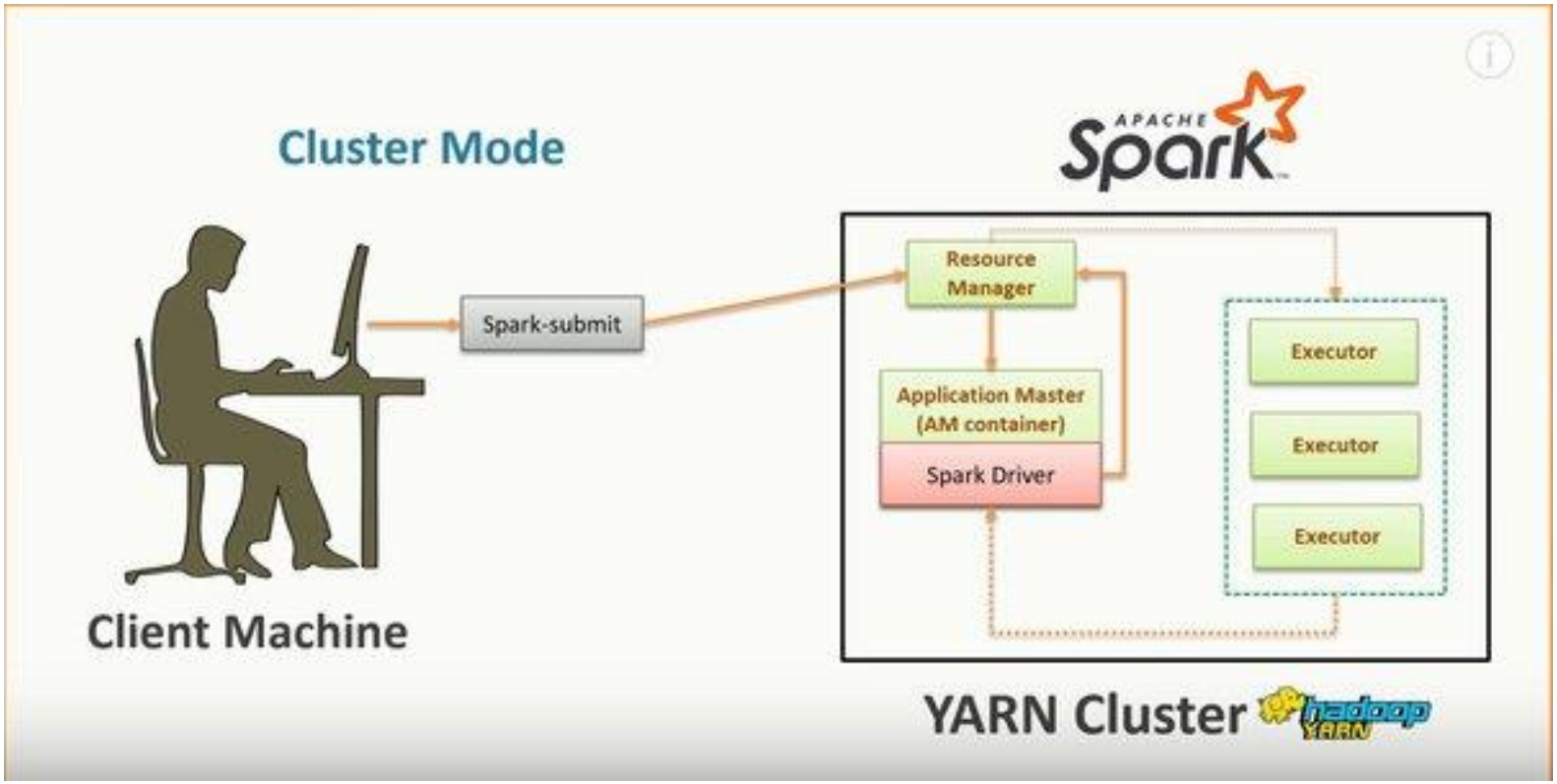
Spark Application

- A Spark application consists of two parts. The first is the application data processing logic expressed using Spark APIs, and the other is the Spark driver.

- The **application data processing** logic can be as simple as a few lines of code to perform a few **data processing** operations or can be as complex as training a large **machine learning model** that requires many iterations and could run for many hours to complete.

- The **Spark driver** is the central coordinator of a **Spark application,** and it interacts with a cluster manager to figure out which machines to run the data processing logic on.

Spark Application



Spark Application

- For each one of those machines, the **Spark driver requests** that the **cluster manager launch** a process called **the Spark executor**.
- Another important job of the Spark driver is to manage and distribute Spark tasks onto **each executor** on behalf of the application.
- If the data processing logic requires the **Spark driver to display the computed results to a user**, then it will coordinate with each Spark executor to collect the computed result and merge them together.

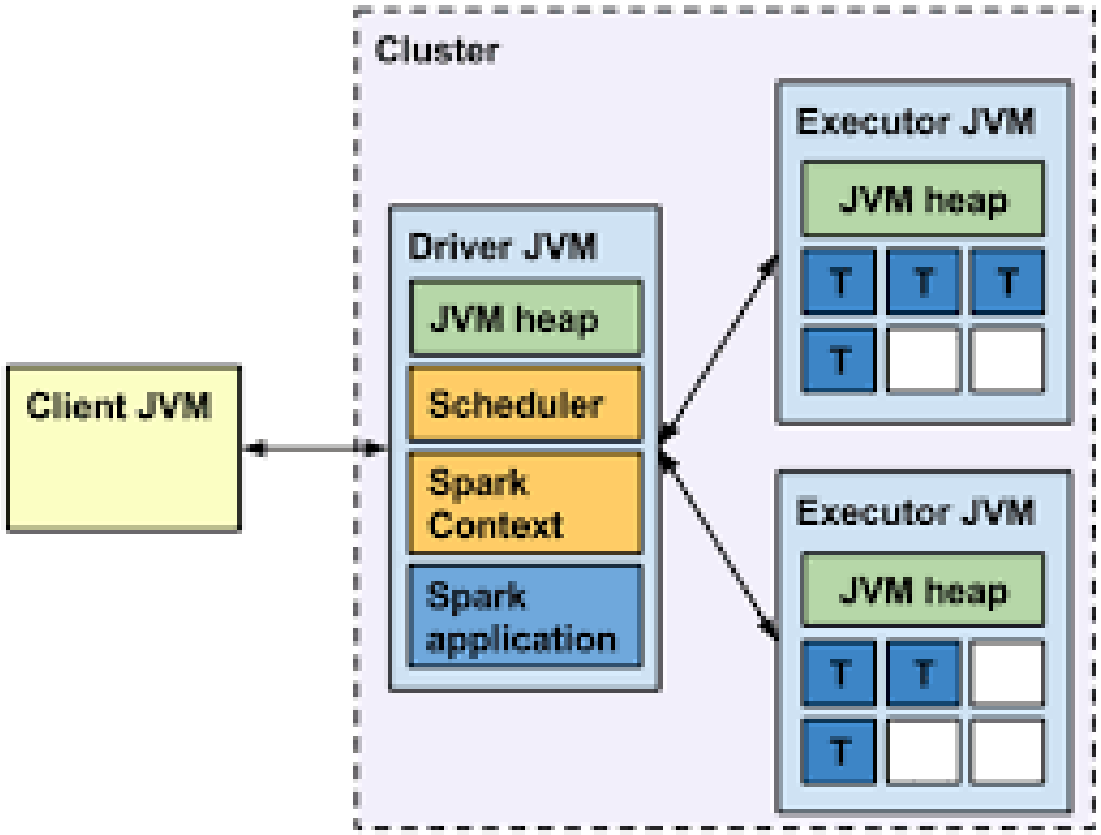
Spark driver and executor

- **Each Spark executor** is a JVM process and is exclusively allocated to a specific **Spark application**.
- This was a conscious design decision to avoid sharing a Spark executor between multiple Spark applications in order **to isolate them from each other so one badly behaving Spark application** wouldn't affect other Spark applications.
- **The lifetime of a Spark executor is the duration of a Spark application**, which could run for a few minutes or for a few days.

Spark driver and executor

- Since Spark applications are running in separate Spark executors, sharing data between them will require writing the data to an external storage system like HDFS.
- Spark employs a **master-slave architecture**, where the Spark driver is the **master and the Spark executor is the slave.**
- Each of these components runs as an independent process on a Spark cluster.

Spark driver and executor



Spark driver and executor

- A Spark application consists of one and only **one Spark driver** and **one or more Spark executors.**
- Playing the slave role, each Spark executor does what it is told, which is to execute the data processing logic in the form of tasks.
- **Each task** is executed on **a separate CPU core**. This is how Spark can speed up the processing of a large amount of data by processing it in parallel.
- In addition to executing assigned tasks, each Spark executor has the responsibility of caching a portion of the data in memory and/or on disk when it is told to do so by the application logic.

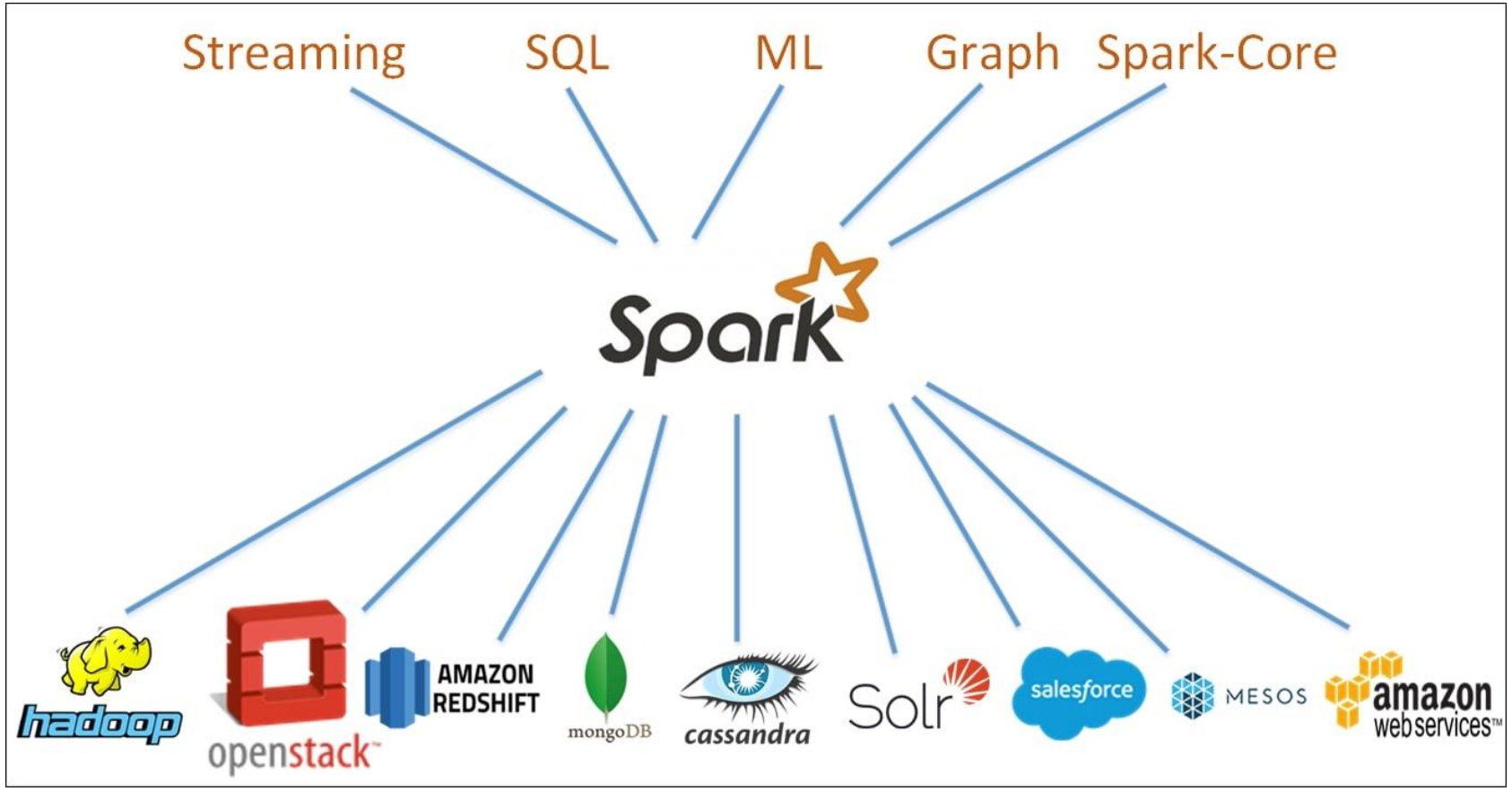
Spark Unified Stack

- Spark provides a unified **data processing engine** known as the **Spark stack**.
- Similar to other well-designed systems, this stack is built on top of a strong foundation called Spark Core, which provides all the necessary **functionalities to manage and run distributed applications** such as scheduling, coordination, and fault tolerance.
- In addition, it provides a powerful and generic programming abstraction for data processing called **resilient distributed datasets** (RDDs).

Spark Unified Stack

- Spark SQL is for batch as well as interactive data processing.
- Spark Streaming is for real-time stream data processing.
- Spark GraphX is for graph processing.
- Spark MLlib is for machine learning.
- Spark R is for running machine learning tasks using the R shell.

Spark Unified Stack



Spark Core

- Spark Core is the bedrock of the Spark distributed **data processing engine**. It consists of two parts: the **distributed computing infrastructure** and the **RDD programming abstraction.**

- The distributed computing infrastructure is responsible for the distribution, coordination, and scheduling of computing tasks across many machines in the cluster.

- This enables the ability to perform **parallel data processing** of a large volume of data efficiently and quickly on a large cluster of machines.

Spark Core

- Two other important responsibilities of the distributed computing infrastructure are **handling of computing task failures** and **efficiently moving data across machines**, which is known as data shuffling.

- The key programming abstraction in Spark is called RDD, and it is something every Spark developer should have some knowledge of, especially its APIs and main concepts.

- The **technical definition of an RDD** is that it is **an immutable** and **fault-tolerant collection** of objects partitioned across a cluster that **can be manipulated in parallel.**

Spark SQL

- Spark SQL is a component built on top of Spark Core, and it is designed for **structured data processing at scale.**

- Its popularity has skyrocketed since its inception because it brings a new level of flexibility, ease of use, and performance.

- Structured Query Language (SQL) has been the lingua franca for data processing because it is fairly easy for users to express their intent, and the execution engine then performs the necessary intelligent optimizations.

Spark SQL

- Spark users now can issue SQL queries to perform data processing or use the high-level abstraction exposed through the DataFrames APIs.

- Behind the scenes, Spark SQL leverages the Catalyst optimizer to perform the kinds of the optimizations that are commonly done in many analytical database engines.

- Another feature **Spark SQL provides is the ability to read data from and write data to various structured formats and storage systems**, such as JavaScript Object Notation (JSON), comma-separated value (CSV) files, Parquet or ORC files, relational databases, Hive, and others.

Spark Structured Streaming

- **Data in motion** has equal or greater value than historical data.
- The **Spark Streaming module enables the ability to process real-time streaming data** from various data sources in a high-throughput and fault-tolerant manner.
- Data can be ingested from sources such as **Kafka**, Flume, Kinesis, Twitter, HDFS, or TCP sockets.
- The main abstraction in the first generation of the Spark Streaming processing engine is called **discretized stream (DStream),** which implements an **incremental stream processing model** by splitting the input data into small batches.

Spark MLlib

- Spark MLlib library provides abstractions for managing and simplifying many of the machine learning model building tasks, such as **featurization, pipeline for constructing, evaluating and tuning model, and persistence of models** to help with moving the model from development to production.

- Machine learning algorithms are iterative in nature, meaning they run through many iterations until a desired objective is achieved. Spark makes it extremely easy to implement those algorithms and run them in a scalable manner through a cluster of machines.

- Commonly used machine learning algorithms such as **classification, regression, clustering, and collaborative filtering** are available out of the box for data scientists and engineers to use.

Spark Graphx

- Graph processing operates on data structures **consisting of vertices and edges connecting them.**
- A graph data structure is often used to represent **real-life networks** of **interconnected entities**, including **professional social networks** on LinkedIn, networks of connected web pages on the Internet, and so on.
- Spark GraphX is a component that enables **graph-parallel computations** by providing an abstraction of a directed multigraph with properties attached to each vertex and edge.
- The GraphX component includes a collection of common **graph processing algorithms** including page ranks, connected components, shortest paths, and others.

SparkR

- SparkR is an R package that provides a light-weight front end to use Apache Spark.
- R is a popular statistical programming language that supports data processing and machine learning tasks.
- However, R was not designed to handle large datasets that can't fit on a single machine.
- SparkR leverages Spark's distributed computing engine to enable large-scale data analysis using the familiar R shell and popular APIs that many data scientists love.

Apache Spark Applications

- Spark is a versatile, fast, and scalable data processing engine.
- It was designed to be a general engine since the beginning days and has proven that it can be used to solve various use cases.
- The following is a small list of applications that were developed using Spark: Customer intelligence applications, **Data warehouse solutions, Real-time streaming solutions, Recommendation engines, Log processing, User-facing services, and Fraud detection**

Spark Examples

Boston house Predictions

`spark-submit pathtopythonfile`

```
spark = SparkSession.builder.appName('spark').getOrCreate()
data = spark.read.csv('boston_housing.csv', header=True, inferSchema=True)
data.show()

feature_columns = data.columns[:-1] # here we omit the final column
assembler = VectorAssembler(inputCols=feature_columns,outputCol="features")
data_2 = assembler.transform(data)

train, test = data_2.randomSplit([0.7, 0.3])
algo = LinearRegression(featuresCol="features", labelCol="medv")
model = algo.fit(train)
```

Cancer Classification

- Run the breast cancer script
- Try with Random Forest classification
- Create a cassandra table to store the information
- Implement a Kafka producer to send information to cassandra
- Configure a cassandra conector to store the information
- Read information from Cassandra (pip install cassandra-driver==3.26 in pyspark_env)
- Submit the spark application

```
create keyspace breastCancer with replication = { 'class':
'SimpleStrategy', 'replication_factor' :1};
use breastCancer;

create table dataCancer(
codeNumber int primary key,
ClumpThickness int,
UniformityCellSize int,
UniformityCellShape int,
MarginalAdhesion int,
SingleEpithelialCellSize int,
BareNuclei int,
BlandChromatin int,
NormalNucleoli int,
Mitoses int,
Classes int );
```

```
# Kafka configuration
bootstrap_servers = 'localhost:9092'
topic = 'breastcancer'

# Load the breast cancer dataset
data = pd.read_csv(‘breast-cancer-wisconsin.data', header=None)

# Create Kafka producer
producer = KafkaProducer (bootstrap_servers =['localhost:9092'],
                          key_serializer = str.encode, value_serializer =
                          lambda v : json.dumps(v).encode('utf-8'))

# Define the producer function
def send_messages():
    for _, row in data.iterrows():
        # Convert each row of the DataFrame to a comma-separated string
        message = ','.join(str(x) for x in row.values)
        message = message.split(",")
        messageFinal = {
            #'codenumber': message[0],
            'clumpthickness': message[1],
            'uniformitycellsize': message[2],
            'uniformitycellshape': message[3],
            'marginaladhesion': message[4],
            'singleepithelialcellsize': message[5],
            'barenuclei': message[6],
            'blandchromatin': message[7],
            'normalnucleoli': message[8],
            'mitoses': message[9],
            'classes': message[10]
        }
        print(messageFinal)
        sleep(2)
        producer.send(topic, key = message[0], value = messageFinal)
        producer.flush()

# Send messages to Kafka
send_messages()

# Close the producer
producer.close()
```

Cassandra Connector

```
topic.breastcancer.breastcancer.datacancer.mapping=codenumber=key,
barenuclei=value.barenuclei, blandchromatin=value.blandchromatin,
classes=value.classes, clumpthickness=value.clumpthickness,
marginaladhesion=value.marginaladhesion, mitoses=value.mitoses,
normalnucleoli=value.normalnucleoli,
singleepithelialcellsize=value.singleepithelialcellsize,
uniformitycellshape= value.uniformitycellshape,
uniformitycellsize=value.uniformitycellsize
```

Additional exercises

- Create two producers that send in parallel random numbers with different time intervals. The connector should send the data to the datastore.
- Create a producer that send the information of the Boston dataset. The connector should send the data to the datastore. Additionally, the Consumer should read the data from Cassandra and apply a regression model using the scikit-learn.
- Create a producer that send the information of the Wine dataset. The Consumer should read the data from Cassandra and apply a regression model using the scikit-learn.
- Create a producer that send the information of the Breast Cancer dataset. The connector should send the data to the datastore. Additionally, the Consumer should read the data from Cassandra and apply a classification model using the apache spark.

Work to be done to the next class

- Create a table to store important information from kafka producers that you have created for your dataset
- Configure the Cassandra connector to store the information sent by producer
- In the paper:
  - Start the description of the proposed method

References

- Estrada, R., & Ruiz, I. (2016). Big data smack. Apress, Berkeley, CA.
- Kumar, M., & Singh, C. (2017). Building Data Streaming Applications with Apache Kafka. Packt Publishing Ltd

UNIVERSIDADE
PORTUCALENSE

Do conhecimento à prática.