BIG DATA ANALYTICS

Parallel Programming with Spark

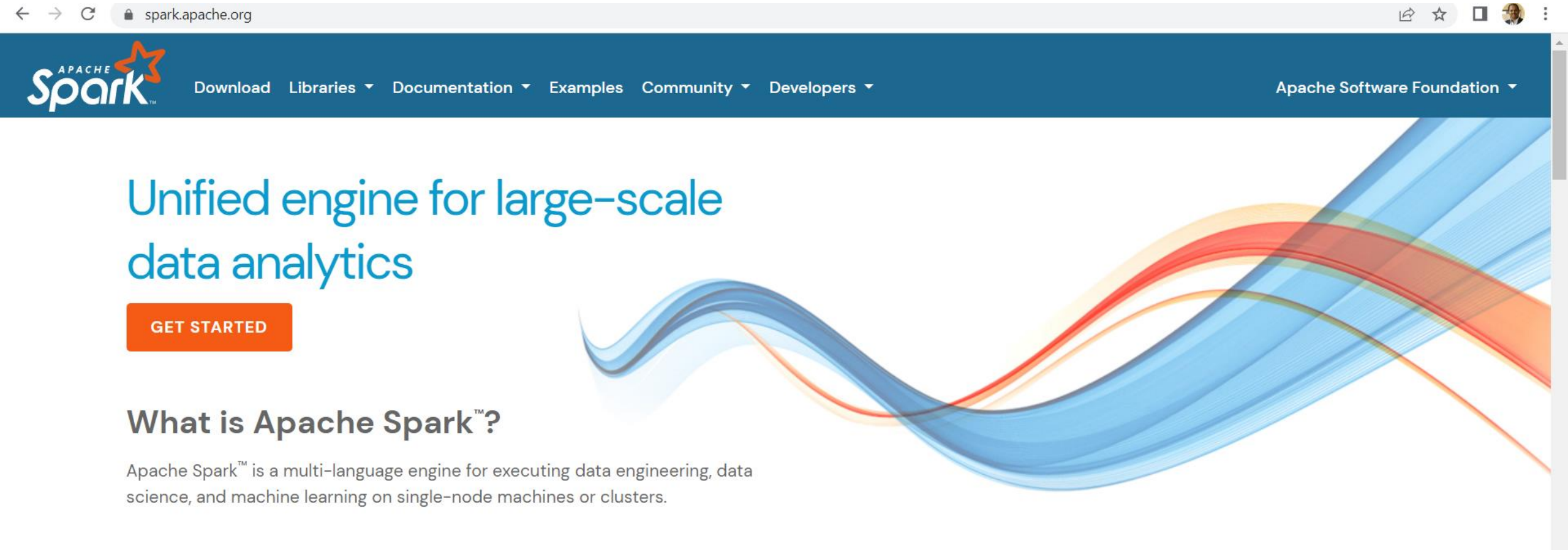
Overview of Spark

Parallel Programming with Spark: *Outline*

- Overview of Spark 
- Fundamentals of Scala and Functional Programming
- Spark Concepts
  - Resilient Distributed Datasets (RDD)
  - Creating RDDs
  - Basic Transformations
  - Basic Actions
  - Word Count Example
- Spark operations
- Job execution
- Spark Applications : Cluster computing with working sets

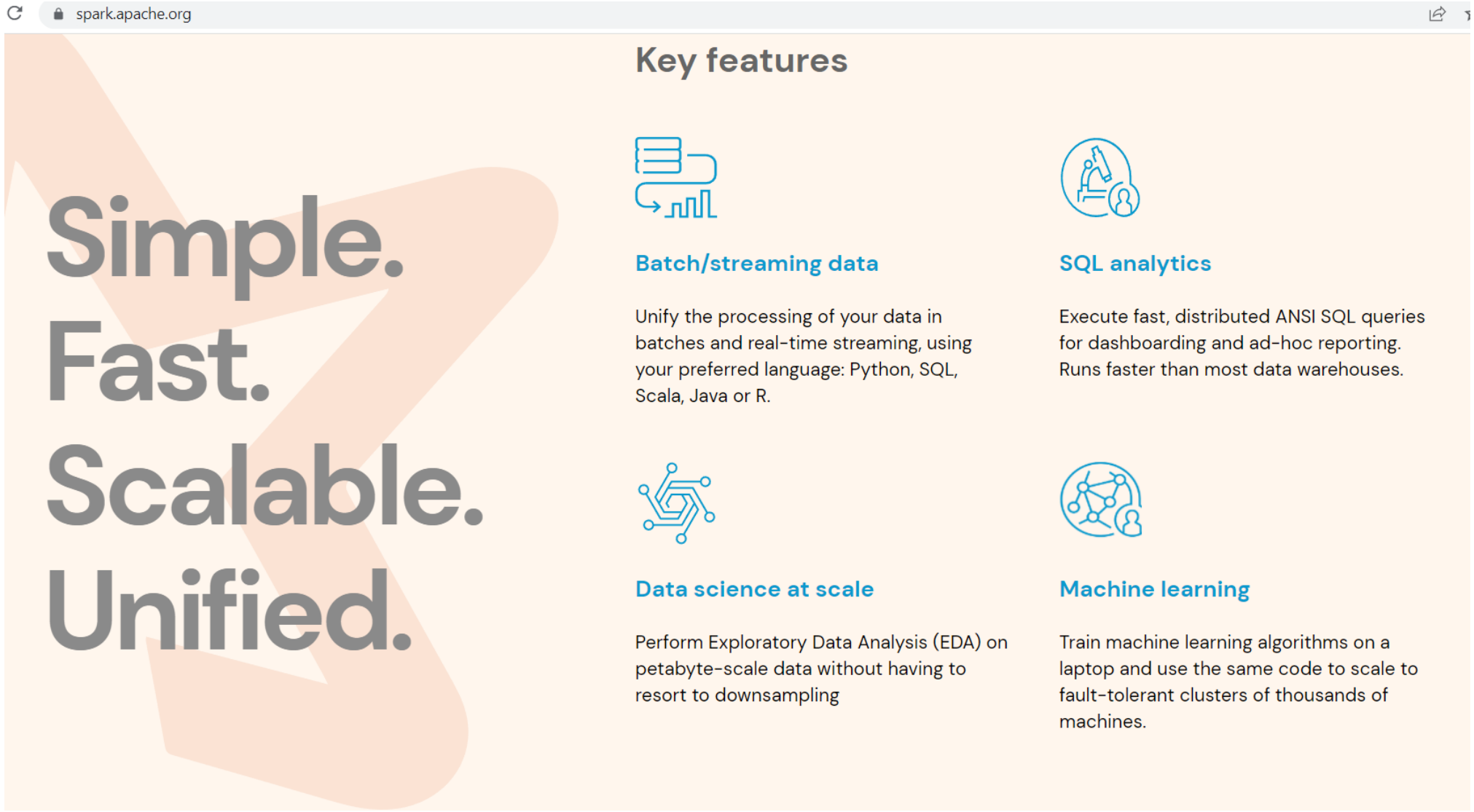
Overview of Spark

From Official Website: https://spark.apache.org/



Overview of Spark

From Official Website: https://spark.apache.org/



Overview of Spark

From Official Website: https://spark.apache.org/

Python | SQL | Scala | Java | R

Run now

```
$ docker run -it --rm apache/spark /opt/spark/bin/spark-shell
scala>
```

```
val df = spark.read.json("logs.json")
df.where("age > 21")
  .select("name.first").show()
```

Overview of Spark

From Official Website: https://spark.apache.org/

The most widely-used engine for scalable computing

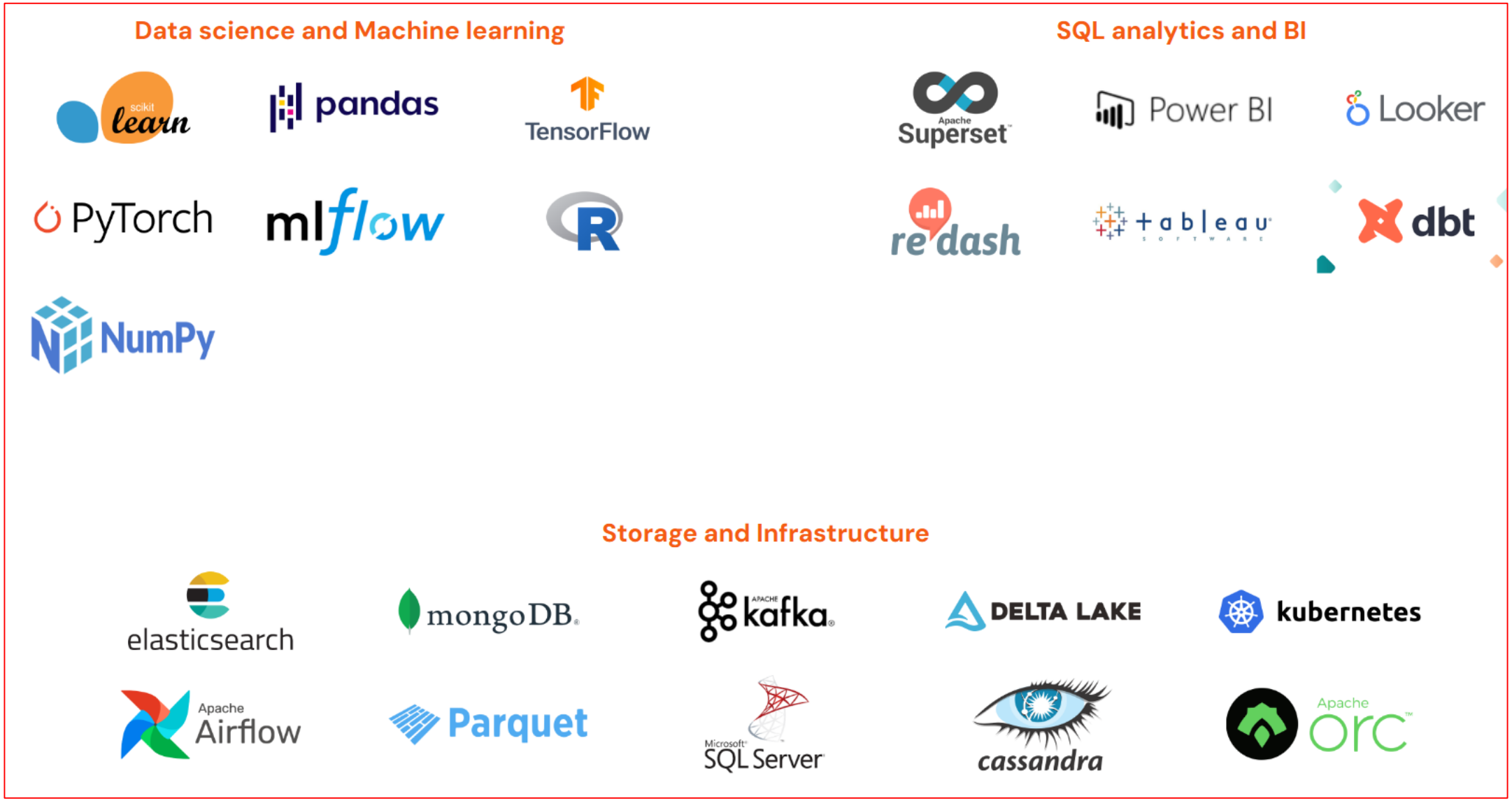
Thousands of companies, including 80% of the Fortune 500, use Apache Spark™.
Over 2,000 contributors to the open source project from industry and academia.

Ecosystem

Apache Spark™ integrates with your favorite frameworks,
helping to scale them to thousands of machines.

Overview of Spark

From Official Website: https://spark.apache.org/



Overview of Spark

From Official Website: https://spark.apache.org/

Spark SQL engine: under the hood

Apache Spark™ is built on an advanced distributed SQL engine for large-scale data

Adaptive Query Execution

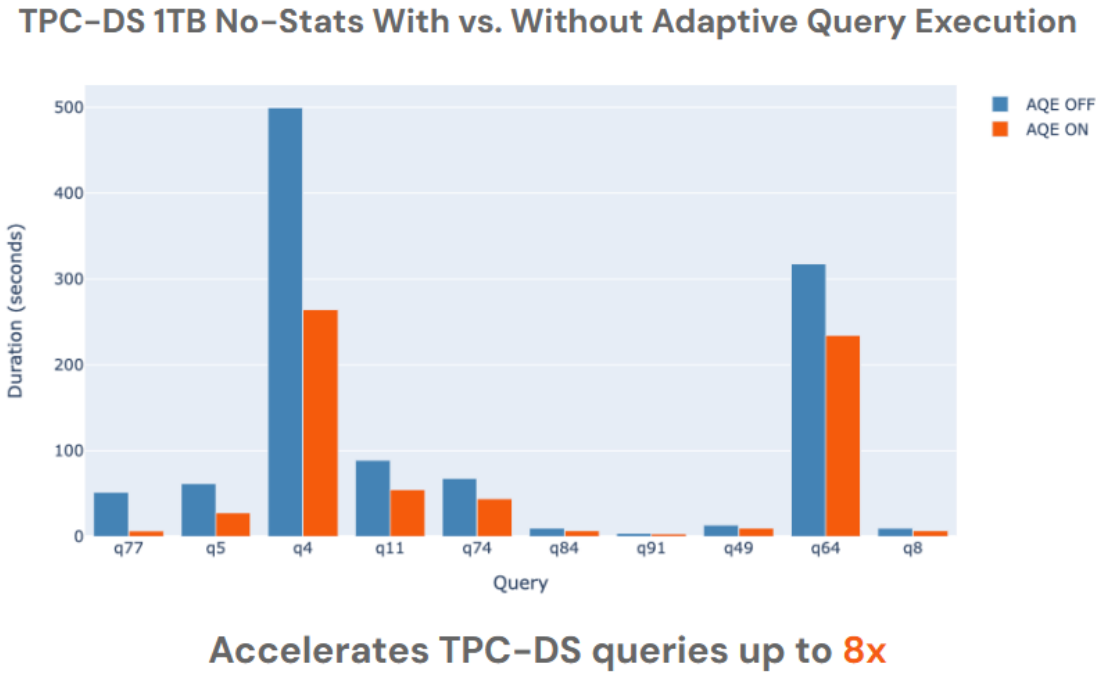
Spark SQL adapts the execution plan at runtime, such as automatically setting the number of reducers and join algorithms.

Support for ANSI SQL

Use the same SQL you're already comfortable with.

Structured and unstructured data

Spark SQL works on structured tables and unstructured data such as JSON or images.



Spark: *Introduction*

- Apache Spark is a unified computing engine and a set of libraries for parallel data processing on computer clusters.
- Spark is the most actively developed open-source engine for this task, making it a standard tool for any developer or data scientist interested in big data.
- Spark supports multiple widely used programming languages (Python, Java, Scala, and R), includes libraries for diverse tasks ranging from SQL to streaming and machine learning, and runs anywhere from a laptop to a cluster of thousands of servers. This makes it an easy system to start with and scale-up to big data processing or incredibly large scale.

Spark: *Introduction*

- Spark has many parallels with MapReduce, in terms of both API and runtime.
- Spark is closely integrated with Hadoop: it can run on YARN and works with Hadoop file formats and storage backends like HDFS.
- Spark is best known for its ability to keep large working datasets in memory between *jobs*.
- This capability allows Spark to outperform the equivalent MapReduce workflow (by an order of magnitude or more in some cases), where datasets are always loaded from disk.

Spark: *Introduction*

- Two styles of application that benefit greatly from Spark's processing model are iterative algorithms (where a function is applied to a dataset repeatedly until an exit condition is met) and interactive analysis (where a user issues a series of ad hoc exploratory queries on a dataset).
- Even if we don't need in-memory caching, Spark is very attractive for a couple of other reasons: its DAG engine and its user experience. Unlike MapReduce, Spark's DAG engine can process arbitrary pipelines of operators and translate them into a single job for the user.
- Spark's user experience is also second to none, with a rich set of APIs for performing many common data processing tasks, such as joins.
- At the time of writing, Spark provides APIs in three languages: Scala, Java, and Python.

Spark: *Introduction*

- Spark also comes with a REPL (read-eval-print loop) for both Scala and Python, which makes it quick and easy to explore datasets.
- Spark is proving to be a good platform on which to build analytics tools, too, and the Apache Spark project includes modules for machine learning (MLlib), graph processing (GraphX), stream processing (Spark Streaming), and SQL (Spark SQL) etc.

Spark: *Introduction*

Key Features

- Speed: Up to 10x faster on disk, 100x faster in memory.
- Ease of Use: Less code compared to traditional methods.
- Versatility: Used for machine learning, stream processing, and graph processing.

Real-Time Application

- Processing large-scale log files for error detection in web services.

Spark: *Introduction*

Comparison with Hadoop

Feature	Hadoop MapReduce	Spark
Execution Model	Disk-based	In-memory
Speed	Slower	Faster (100x memory, 10x disk)
Iterative Processing	Inefficient	Efficient
Real-time Processing	Not suitable	Supported

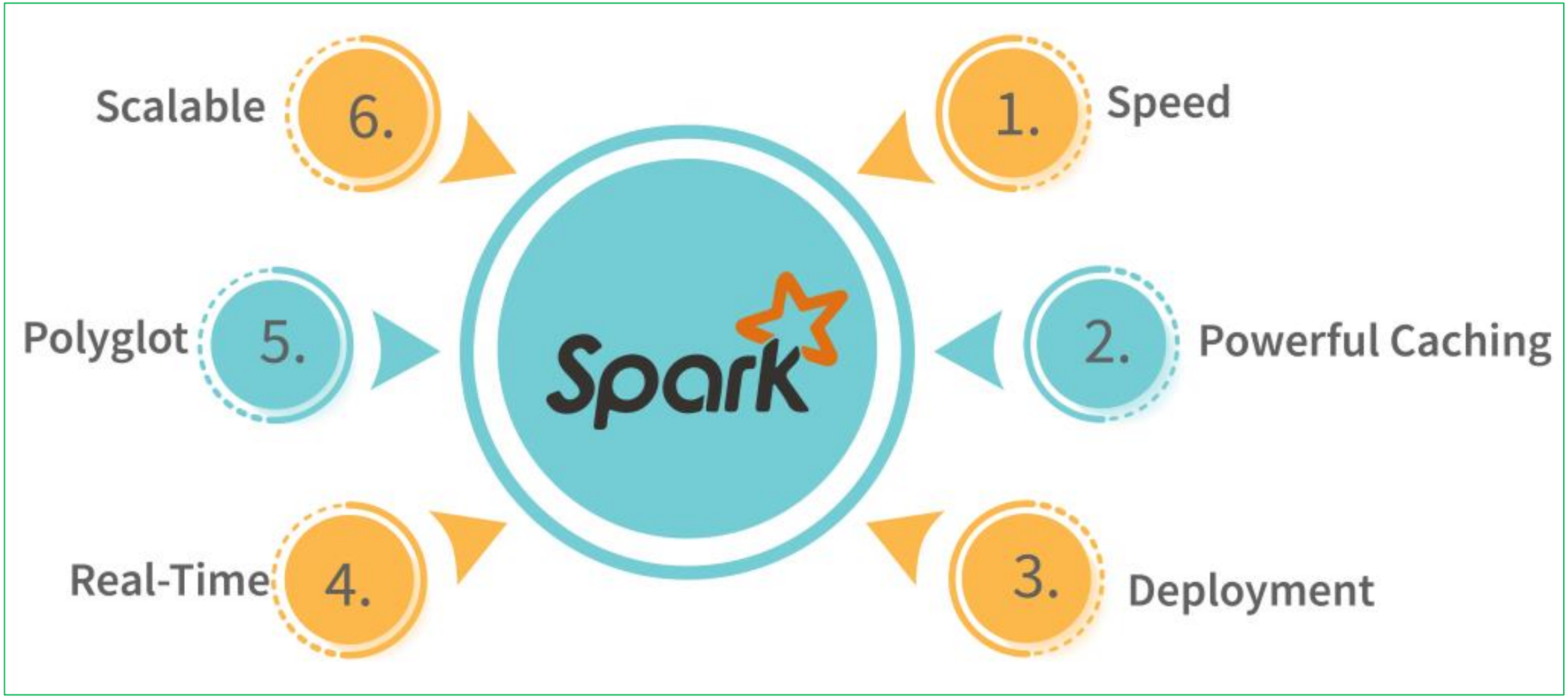
Importance of Spark Framework

Apache Spark Features:

- Apache Spark, a popular cluster computing framework, was created in order to accelerate data processing applications.
- Spark, which enables applications to run faster by utilising in-memory cluster computing, is a popular open-source framework.
- A cluster is a collection of nodes that communicate with each other and share data.
- Because of implicit data parallelism and fault tolerance, Spark may be applied to a wide range of sequential and interactive processing demands.

Importance of Spark Framework

Apache Spark Features:



Importance of Spark Framework

Apache Spark Features:

- **Speed:** Spark performs up to 100 times faster than MapReduce for processing large amounts of data. It is also able to divide the data into chunks in a controlled way.
- **Powerful Caching:** Powerful caching and disk persistence capabilities are offered by a simple programming layer.
- **Deployment:** Mesos, Hadoop via YARN, or Spark's own cluster manager can all be used to deploy it.
- **Real-Time:** Because of its in-memory processing, it offers real-time computation and low latency.
- **Polyglot:** In addition to Java, Scala, Python, and R, Spark also supports all four of these languages. We can write Spark code in any one of these languages. Spark also provides a command-line interface in Scala and Python.

Components of the Spark unified stack

Components and libraries Spark offers to end-users:

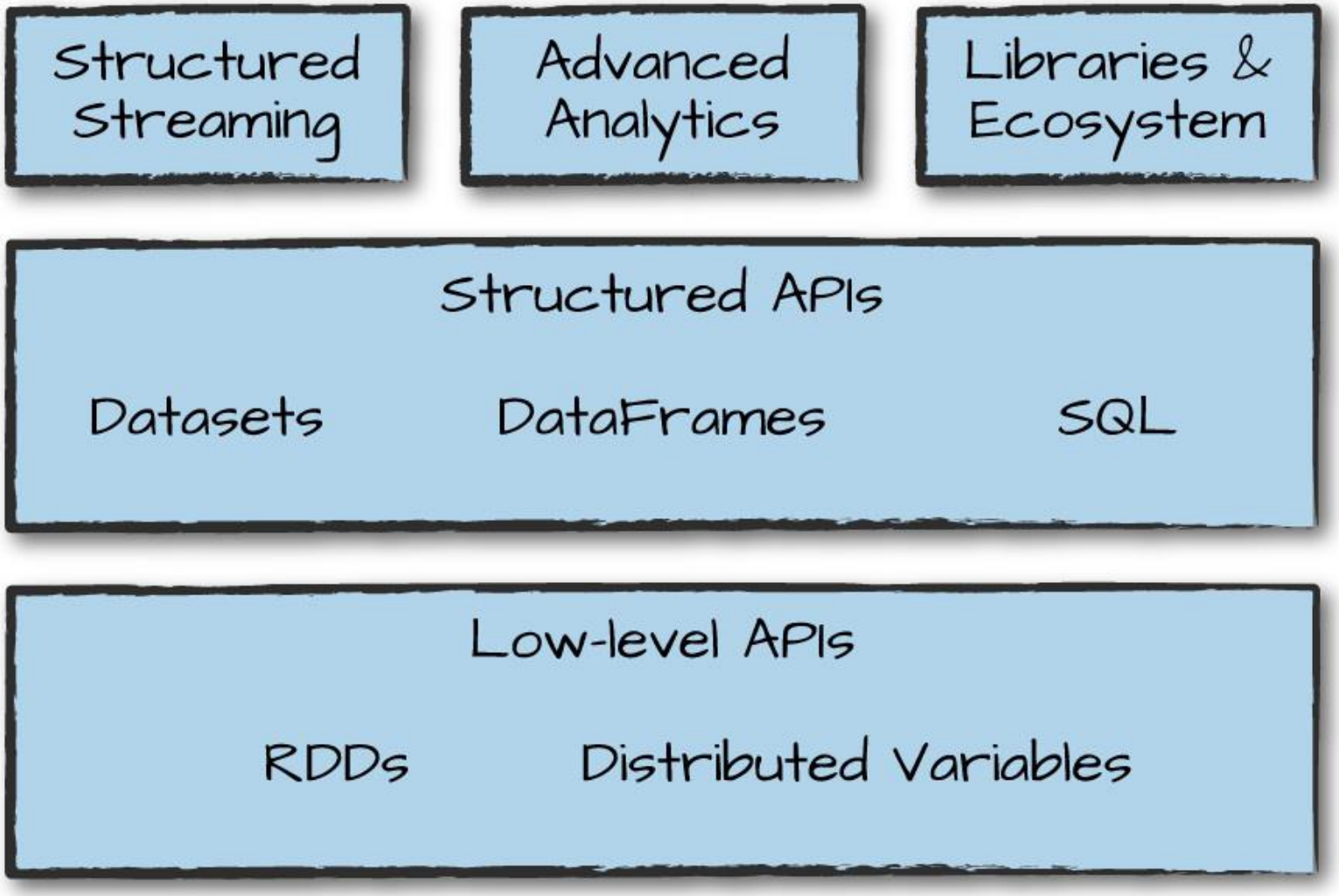
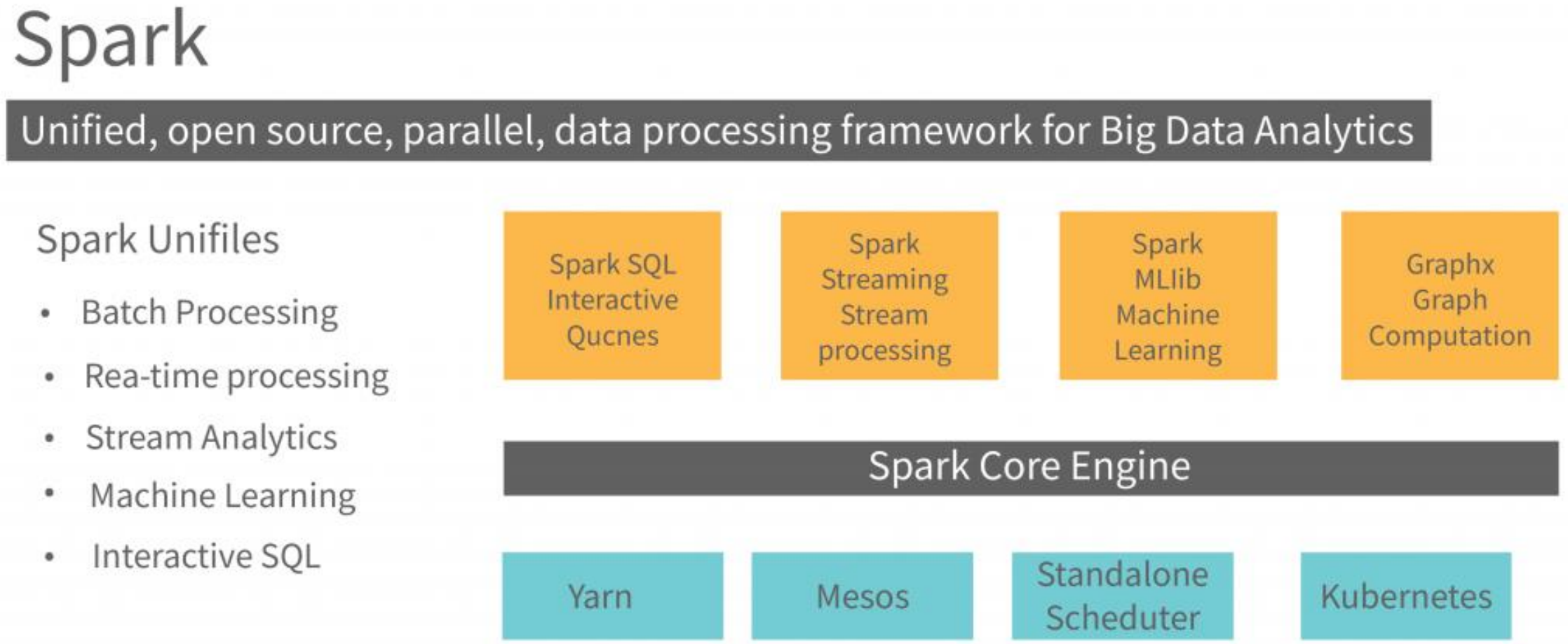
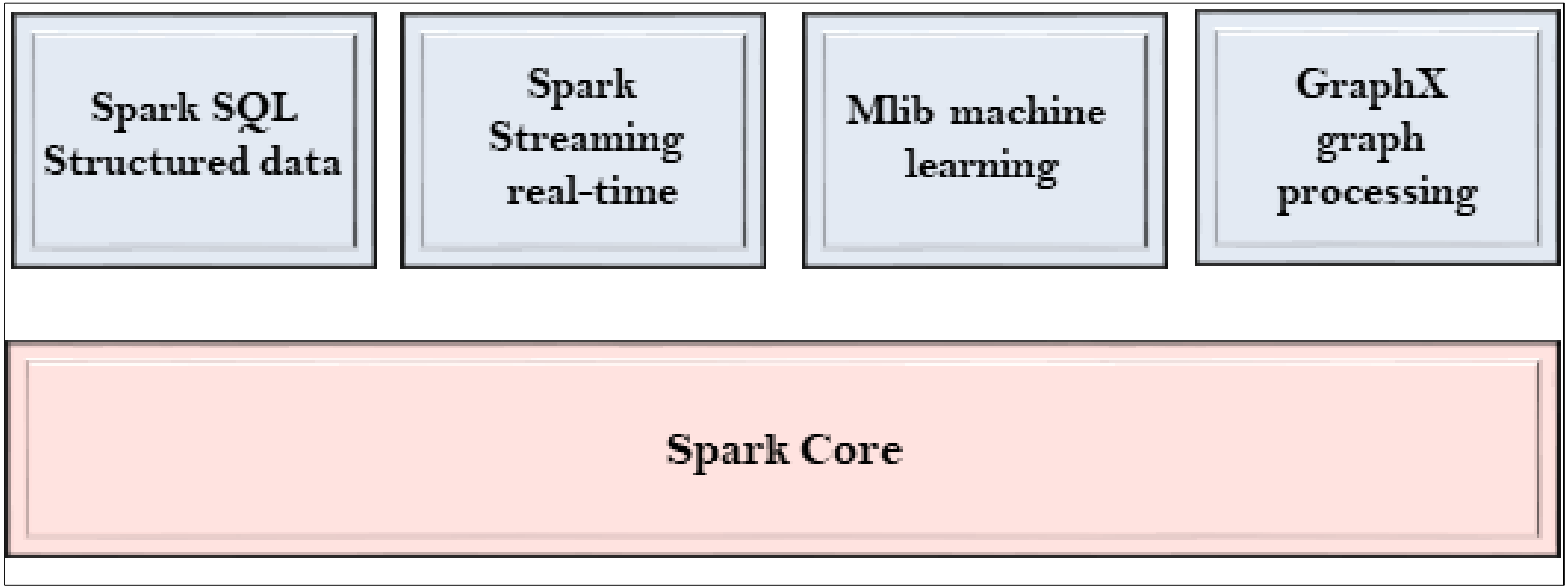


Fig: Spark's Toolkit

Components of the Spark unified stack



Components of the Spark unified stack



Components of the Spark unified stack

Spark Core:

- The Spark Core is the heart of Spark and performs the core functionality.
- It holds the components for task scheduling, fault recovery, interacting with storage systems and memory management.

Components of the Spark unified stack

Spark SQL:

- The Spark SQL is built on the top of Spark Core. It provides support for structured data.
- It allows to query the data via SQL (Structured Query Language) as well as the Apache Hive variant of SQL?called the HQL (Hive Query Language).
- It supports JDBC and ODBC connections that establish a relation between Java objects and existing databases, data warehouses and business intelligence tools.
- It also supports various sources of data like Hive tables, Parquet, and JSON.

Components of the Spark unified stack

Spark Streaming:

- Spark Streaming is a Spark component that supports scalable and fault-tolerant processing of streaming data.
- It uses Spark Core's fast scheduling capability to perform streaming analytics.
- It accepts data in mini-batches and performs RDD transformations on that data.
- Its design ensures that the applications written for streaming data can be reused to analyze batches of historical data with little modification.
- The log files generated by web servers can be considered as a real-time example of a data stream.

Components of the Spark unified stack

MLlib:

- The MLlib is a Machine Learning library that contains various machine learning algorithms.
- These include correlations and hypothesis testing, classification and regression, clustering, and principal component analysis.
- It is nine times faster than the disk-based implementation used by Apache Mahout.

Components of the Spark unified stack

GraphX:

- The GraphX is a library that is used to manipulate graphs and perform graph-parallel computations.
- It facilitates to create a directed graph with arbitrary properties attached to each vertex and edge.
- To manipulate graph, it supports various fundamental operators like subgraph, join Vertices, and aggregate Messages.

Spark Architecture

Two Main Abstractions of Apache Spark:

The Apache Spark architecture consists of two main abstraction layers:

Resilient Distributed Datasets (RDD):

- It is a key tool for data computation.
- It enables you to recheck data in the event of a failure, and it acts as an interface for immutable data.
- It helps in recomputing data in case of failures, and it is a data structure.
- There are two methods for modifying RDDs: transformations and actions.

Spark Architecture

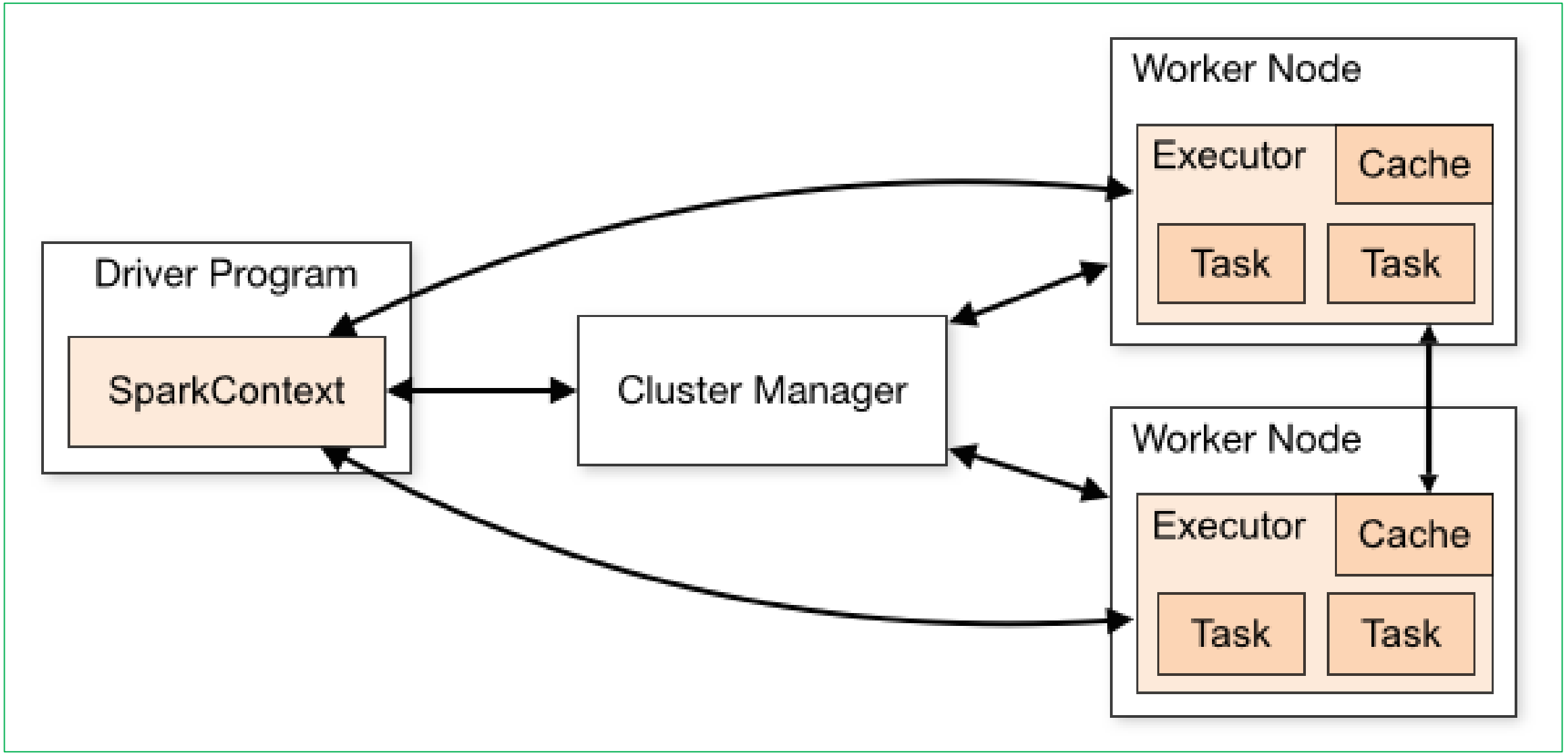
Two Main Abstractions of Apache Spark:

The Apache Spark architecture consists of two main abstraction layers:

Directed Acyclic Graph (DAG):

- The driver converts the program into a DAG for each job.
- The Apache Spark Eco-system includes various components such as the API core, Spark SQL, Streaming and real-time processing, MLIB, and Graph X.
- A sequence of connection between nodes is referred to as a driver.
- As a result, you can read volumes of data using the Spark shell.
- You can also use the Spark context -cancel, run a job, task (work), and job (computation) to stop a job.

Spark Architecture



https://spark.apache.org/docs/latest/cluster-overview.html

Spark Architecture

Components:

- Spark applications run as independent sets of processes on a cluster, coordinated by the SparkContext object in our main program (called the ***driver program***).
- Specifically, to run on a cluster, the SparkContext can connect to several types of *cluster managers* (either Spark's own standalone cluster manager, Mesos, YARN or Kubernetes), which allocate resources across applications. Once connected, Spark acquires executors on nodes in the cluster, which are processes that run computations and store data for our application. Next, it sends our application code (defined by JAR or Python files passed to SparkContext) to the executors. Finally, SparkContext sends tasks to the executors to run.

Spark Architecture

There are several useful things to note about this architecture:

1. Each application gets its own executor processes, which stay up for the duration of the whole application and run tasks in multiple threads. This has the benefit of isolating applications from each other, on both the scheduling side (each driver schedules its own tasks) and executor side (tasks from different applications run in different JVMs). However, it also means that data cannot be shared across different Spark applications (instances of SparkContext) without writing it to an external storage system.

2. Spark is computing to the underlying cluster manager. As long as it can acquire executor processes, and these communicate with each other, it is relatively easy to run it even on a cluster manager that also supports other applications (e.g. Mesos/YARN/Kubernetes).

Spark Architecture

There are several useful things to note about this architecture:

3. The driver program must listen for and accept incoming connections from its executors throughout its lifetime. As such, the driver program must be network addressable from the worker nodes.

4. Because the driver schedules tasks on the cluster, it should be run close to the worker nodes, preferably on the same local area network. If you'd like to send requests to the cluster remotely, it's better to open an RPC to the driver and have it submit operations from nearby than to run a driver far away from the worker nodes.

Spark Architecture Applications

A high-level view of the architecture of the Apache Spark application is as follows:

The Spark driver:

- The master node (process) in a driver process coordinates workers and oversees the tasks.
- Spark is split into jobs and scheduled to be executed on executors in clusters.
- Spark contexts (gateways) are created by the driver to monitor the job working in a specific cluster and to connect to a Spark cluster.
- In the diagram, the driver programs call the main application and create a spark context (acts as a gateway) that jointly monitors the job working in the cluster and connects to a Spark cluster.
- Everything is executed using the spark context.

Spark Architecture Applications

The Spark driver:

- Each Spark session has an entry in the Spark context.
- Spark drivers include more components to execute jobs in clusters, as well as cluster managers.
- Context acquires worker nodes to execute and store data as Spark clusters are connected to different types of cluster managers.
- When a process is executed in the cluster, the job is divided into stages with gain stages into scheduled tasks.

Spark Architecture Applications

The Spark executors:

- An executor is responsible for executing a job and storing data in a cache at the outset.
- Executors first register with the driver programme at the beginning.
- These executors have a number of time slots to run the application concurrently.
- The executor runs the task when it has loaded data and they are removed in idle mode.
- The executor runs in the Java process when data is loaded and removed during the execution of the tasks.
- The executors are allocated dynamically and constantly added and removed during the execution of the tasks.
- A driver program monitors the executors during their performance. Users' tasks are executed in the Java process.

Spark Architecture Applications

Cluster Manager:

- A driver program controls the execution of jobs and stores data in a cache.
- At the outset, executors register with the drivers.
- This executor has a number of time slots to run the application concurrently.
- Executors read and write external data in addition to servicing client requests.
- A job is executed when the executor has loaded data and they have been removed in the idle state.
- The executor is dynamically allocated, and it is constantly added and deleted depending on the duration of its use.
- A driver program monitors executors as they perform users' tasks.
- Code is executed in the Java process when an executor executes a user's task.

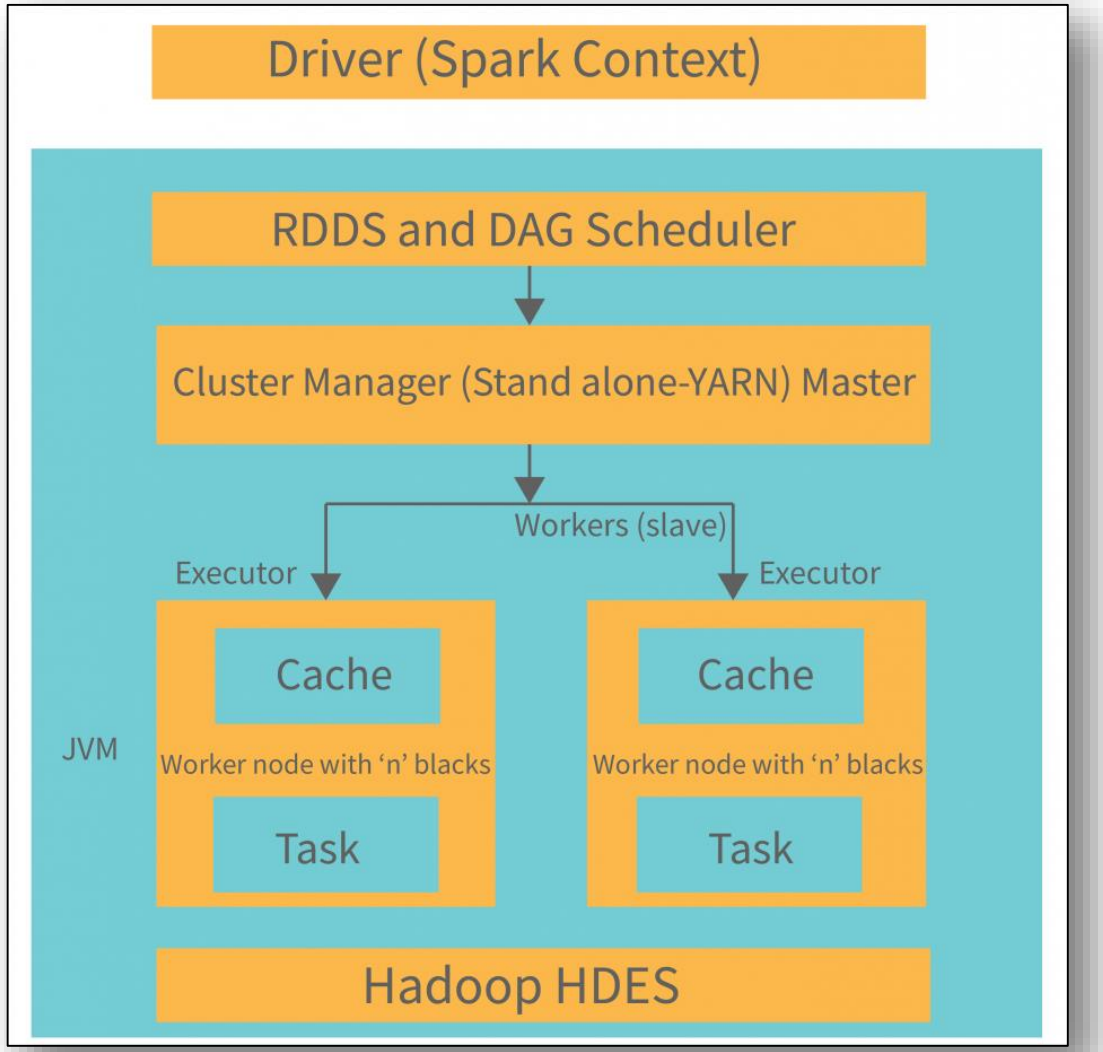
Spark Architecture Applications

Worker Nodes:

- The slave nodes function as executors, processing tasks, and returning the results back to the spark context.
- The master node issues tasks to the Spark context and the worker nodes execute them.
- They make the process simpler by boosting the worker nodes (1 to n) to handle as many jobs as possible in parallel by dividing the job up into sub-jobs on multiple machines.
- A Spark worker monitors worker nodes to ensure that the computation is performed simply.
- Each worker node handles one Spark task.
- In Spark, a partition is a unit of work and is assigned to one executor for each one.

Spark Architecture Applications

Worker Nodes:



Spark Architecture Applications

Modes of Execution:

- We can choose from *three different execution modes*: local, shared, and dedicated. These determine where our app's resources are physically located when we run our app. We can decide where to store resources locally, in a shared location, or in a dedicated location.

1. Cluster mode
2. Client mode
3. Local mode

Spark Architecture Applications

Cluster mode:

- Cluster mode is the most frequent way of running Spark Applications.
- In cluster mode, a user delivers a pre-compiled JAR, Python script, or R script to a cluster manager.
- Once the cluster manager receives the pre-compiled JAR, Python script, or R script, the driver process is launched on a worker node inside the cluster, in addition to the executor processes.
- This means that the cluster manager is in charge of all Spark application-related processes.

Spark Architecture Applications

Client mode:

- In contrast to cluster mode, where the Spark driver remains on the client machine that submitted the application, the Spark driver is removed in client mode and is therefore responsible for maintaining the Spark driver process on the client machine.
- These machines, usually referred to as gateway machines or edge nodes, are maintained on the client machine.

Spark Architecture Applications

Local mode:

- Local mode runs the entire Spark Application on a single machine, as opposed to the previous two modes, which parallelized the Spark Application through threads on that machine.
- As a result, the local mode uses threads instead of parallelized threads.
- This is a common way to experiment with Spark, try out your applications, or experiment iteratively without having to make any changes on Spark's end.
- *In practice, we do not recommend using local mode for running production applications.*

Spark Architecture Applications

Cluster Manager Types:

There are several cluster managers supported by the system:

1. Standalone:

- A Spark cluster manager is included with the software package to make setting up a cluster easy.
- The Resource Manager and Worker are the only Spark Standalone Cluster components that are independent.
- There is only one executor that runs tasks on each worker node in Standalone Cluster mode.
- When a client establishes a connection with the Standalone Master, requests resources, and begins the execution process, a Standalone Clustered master starts the execution process.

Spark Architecture Applications

Cluster Manager Types:

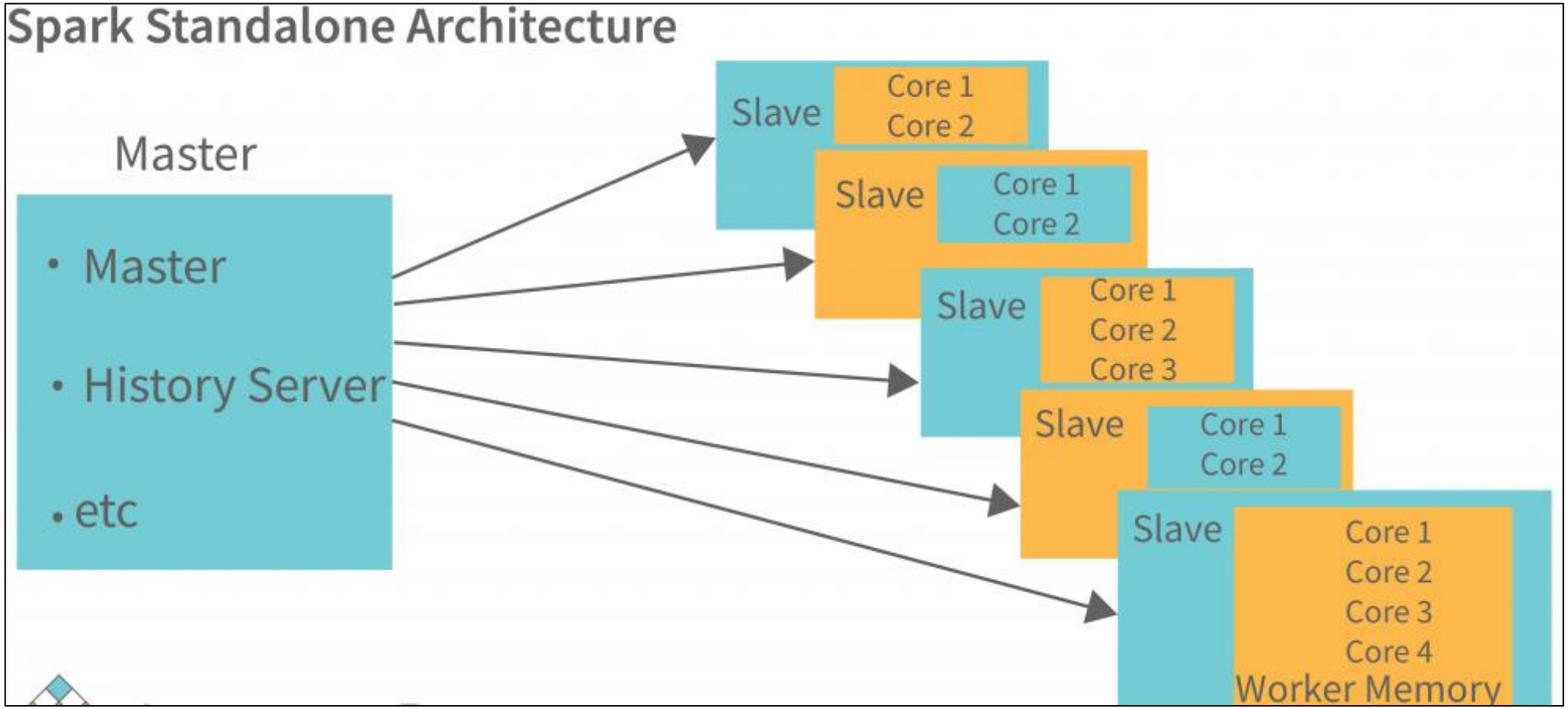
1. Standalone:

- The client here is the application master, and it wants the resources from the resource manager.
- We have a Web UI to view all clusters and job statistics in the Cluster Manager.

Spark Architecture Applications

Cluster Manager Types:

1. Standalone:



Spark Architecture Applications

Cluster Manager Types:

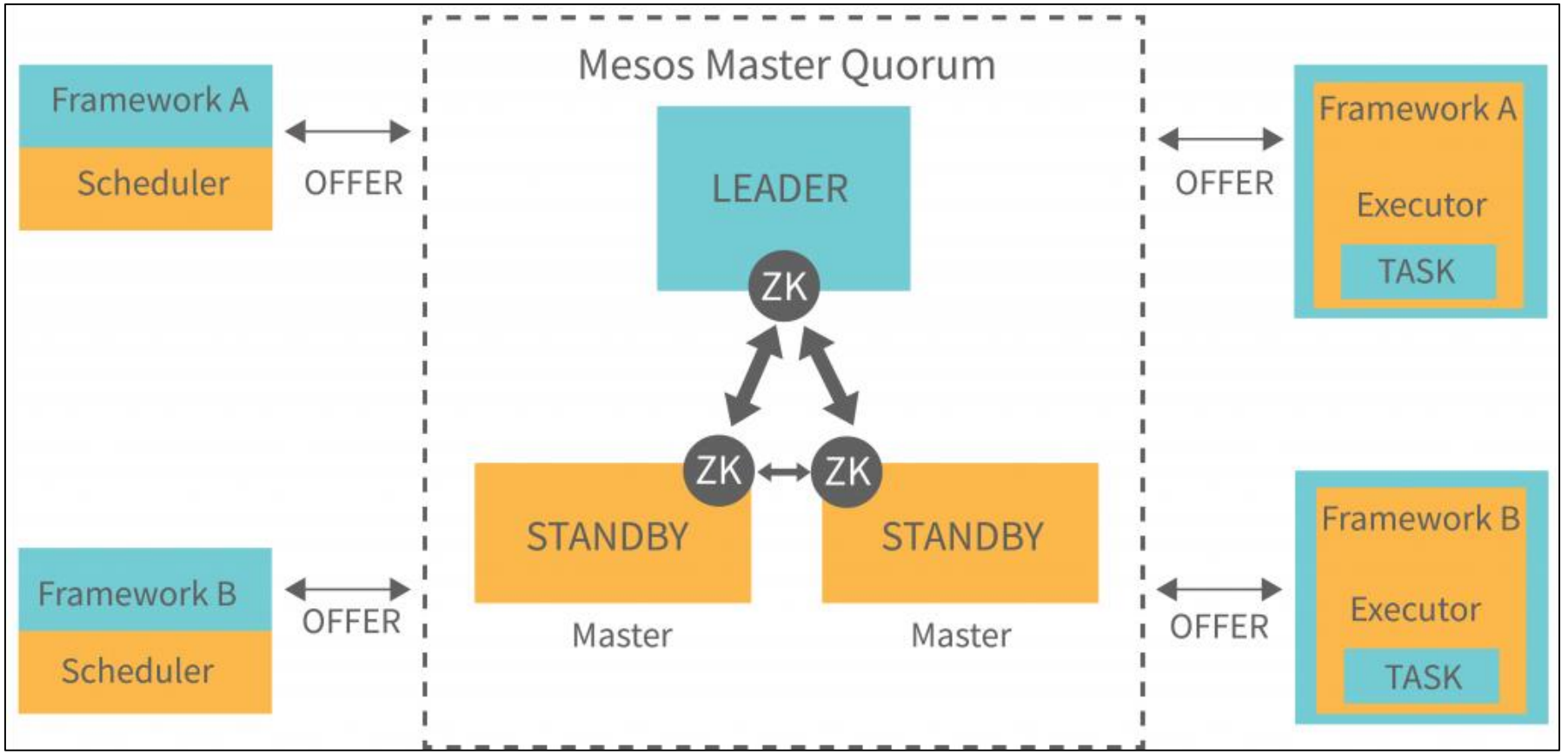
2. Apache Mesos :

- It can run Hadoop MapReduce and service apps as well as be a general cluster manager.
- Apache Mesos contributes to the development and management of application clusters by using dynamic resource sharing and isolation.
- It enables the deployment and administration of applications in large-scale cluster environments.

Spark Architecture Applications

Cluster Manager Types:

2. Apache Mesos:



Spark Architecture Applications

Cluster Manager Types:

2. Apache Mesos :

The Mesos framework includes three components:

- **Mesos Master:** A Mesos Master cluster provides fault tolerance (the capability to operate and recover from loss when a failure occurs). Because of the Mesos Master design, a cluster contains many Mesos Masters.
- **Mesos Slave:** A Mesos Slave is an instance that delivers resources to the cluster. When a Mesos Master assigns a task, Mesos Slave does not assign resources.
- **Mesos Frameworks:** Applications can request resources from the cluster so that the application can perform the tasks. Mesos Frameworks allow for this.

Spark Architecture Applications

Cluster Manager Types:

3. Hadoop YARN:

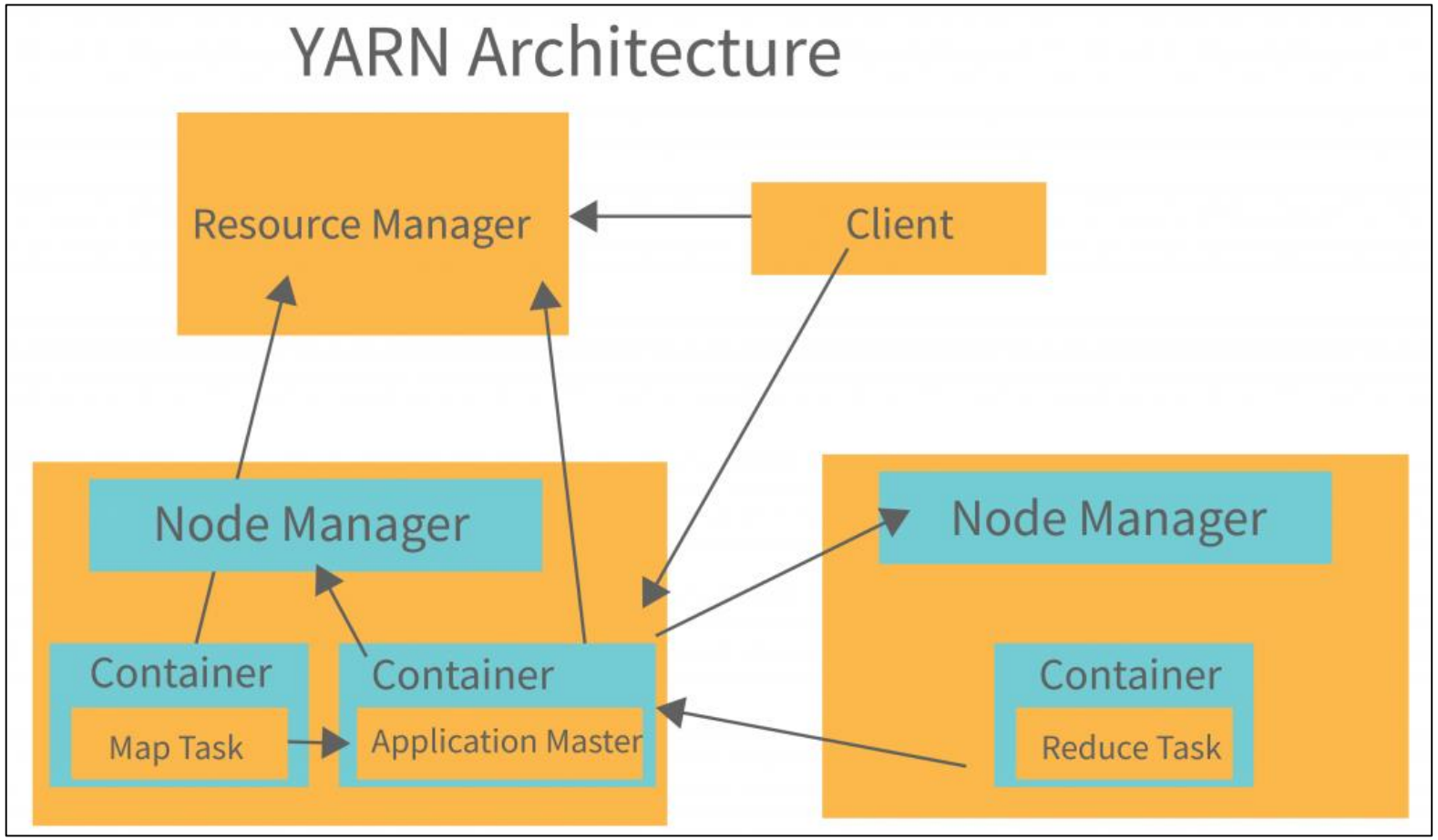
A key feature of Hadoop 2.0 is the improved resource manager. The Hadoop ecosystem relies on YARN to handle resources. It consists of the following two components:

- **Resource Manager:** It controls the allocation of system resources on all applications. A Scheduler and an Application Manager are included. Applications receive resources from the Scheduler.
- **Node Manager:** Each job or application needs one or more containers, and the Node Manager monitors these containers and their usage. Node Manager consists of an Application Manager and a Container Manager. Each task in the MapReduce framework runs in a container. The Node Manager monitors the containers and resource usage, and this is reported to the Resource Manager.

Spark Architecture Applications

Cluster Manager Types:

3. Hadoop YARN:



Spark Architecture Applications

Cluster Manager Types:

4. Kubernetes:

- A framework for deploying, scaling, and managing containerized applications that are open source.
- There is a third-party project (not supported by the Spark project) that adds support for Nomad (Nomad is a distributed, highly available, data center-aware cluster manage) as a cluster manager.

Spark Architecture Applications

The following table summarizes terms we'll see used to refer to cluster concepts:

Term	Meaning
Application	User program built on Spark. Consists of a *driver program* and *executors* on the cluster.
Application jar	A jar containing the user's Spark application. In some cases users will want to create an "uber jar" containing their application along with its dependencies. The user's jar should never include Hadoop or Spark libraries, however, these will be added at runtime.
Driver program	The process running the main() function of the application and creating the SparkContext
Cluster manager	An external service for acquiring resources on the cluster (e.g. standalone manager, Mesos, YARN, Kubernetes)
Deploy mode	Distinguishes where the driver process runs. In "cluster" mode, the framework launches the driver inside of the cluster. In "client" mode, the submitter launches the driver outside of the cluster.
Worker node	Any node that can run application code in the cluster
Executor	A process launched for an application on a worker node, that runs tasks and keeps data in memory or disk storage across them. Each application has its own executors.
Task	A unit of work that will be sent to one executor
Job	A parallel computation consisting of multiple tasks that gets spawned in response to a Spark action (e.g. `save`, `collect`); you'll see this term used in the driver's logs.
Stage	Each job gets divided into smaller sets of tasks called *stages* that depend on each other (similar to the map and reduce stages in MapReduce); you'll see this term used in the driver's logs.