Apache Spark

- Industries are using Hadoop extensively to analyze their data sets.
- The reason is that Hadoop framework is based on a simple programming model (MapReduce) and it enables a computing solution that is scalable, flexible, fault-tolerant and cost effective.
- The main concern is to maintain speed in processing large datasets in terms of waiting time between queries and waiting time to run the program.

Cont…

- Spark was introduced by Apache Software Foundation for speeding up the Hadoop computational computing software process.
- **Spark is not a modified version of Hadoop** and is not, really, dependent on Hadoop because it has its own cluster management.
- Hadoop is just one of the ways to implement Spark.
- Spark uses Hadoop in two ways – one is **storage** and second is **processing.** Since Spark has its own cluster management computation, it uses Hadoop for storage purpose only.

Introduction

- Apache Spark is a lightning-fast cluster computing technology, designed for fast computation.

- It is based on Hadoop MapReduce and it extends the MapReduce model to efficiently use it for more types of computations, which includes interactive queries and stream processing.

- The main feature of Spark is its **in-memory cluster computing** that increases the processing speed of an application.

Cont…

- Spark is designed to cover a wide range of workloads such as batch applications, iterative algorithms, interactive queries and streaming.
- It also reduces the management burden of maintaining separate tools.
- Spark is one of Hadoop's sub project developed in 2009 in UC Berkeley's AMPLab by Matei Zaharia.
- It was Open Sourced in 2010 under a BSD license.
- It was donated to Apache software foundation in 2013, and now Apache Spark has become a top level Apache project from Feb-2014.

Features of SPARK

Speed:

Spark helps to run an application in Hadoop cluster, up to 100 times faster in memory, and 10 times faster when running on disk. This is possible by reducing number of read/write operations to disk. It stores the intermediate processing data in memory.

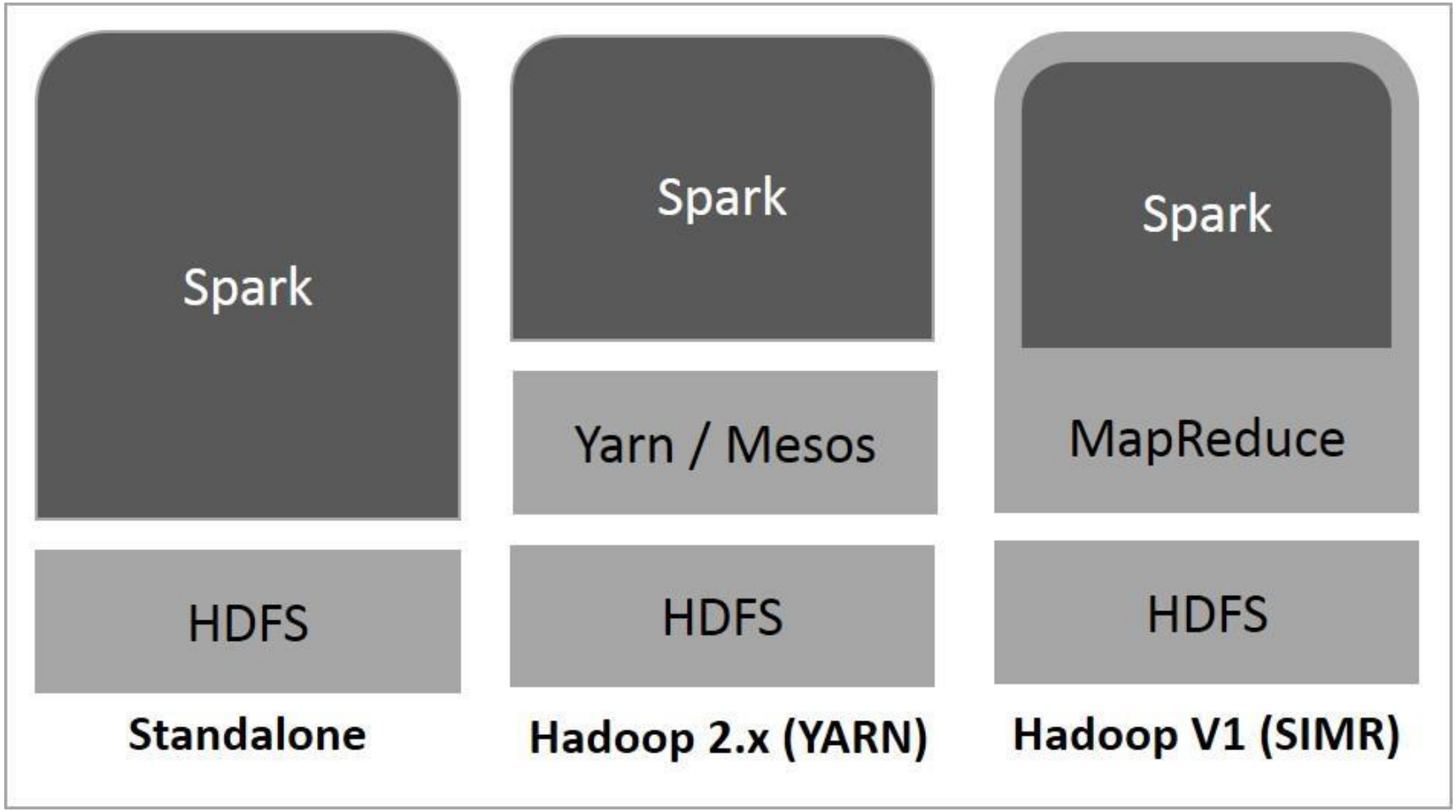
Supports multiple languages:

Spark provides built-in APIs in Java, Scala, or Python. Therefore, you can write applications in different languages. Spark comes up with 80 high-level operators for interactive querying.

Advanced Analytics:

Spark not only supports 'Map' and 'reduce'. It also supports SQL queries, Streaming data, Machine learning (ML), and Graph algorithms.

Spark Built on Hadoop



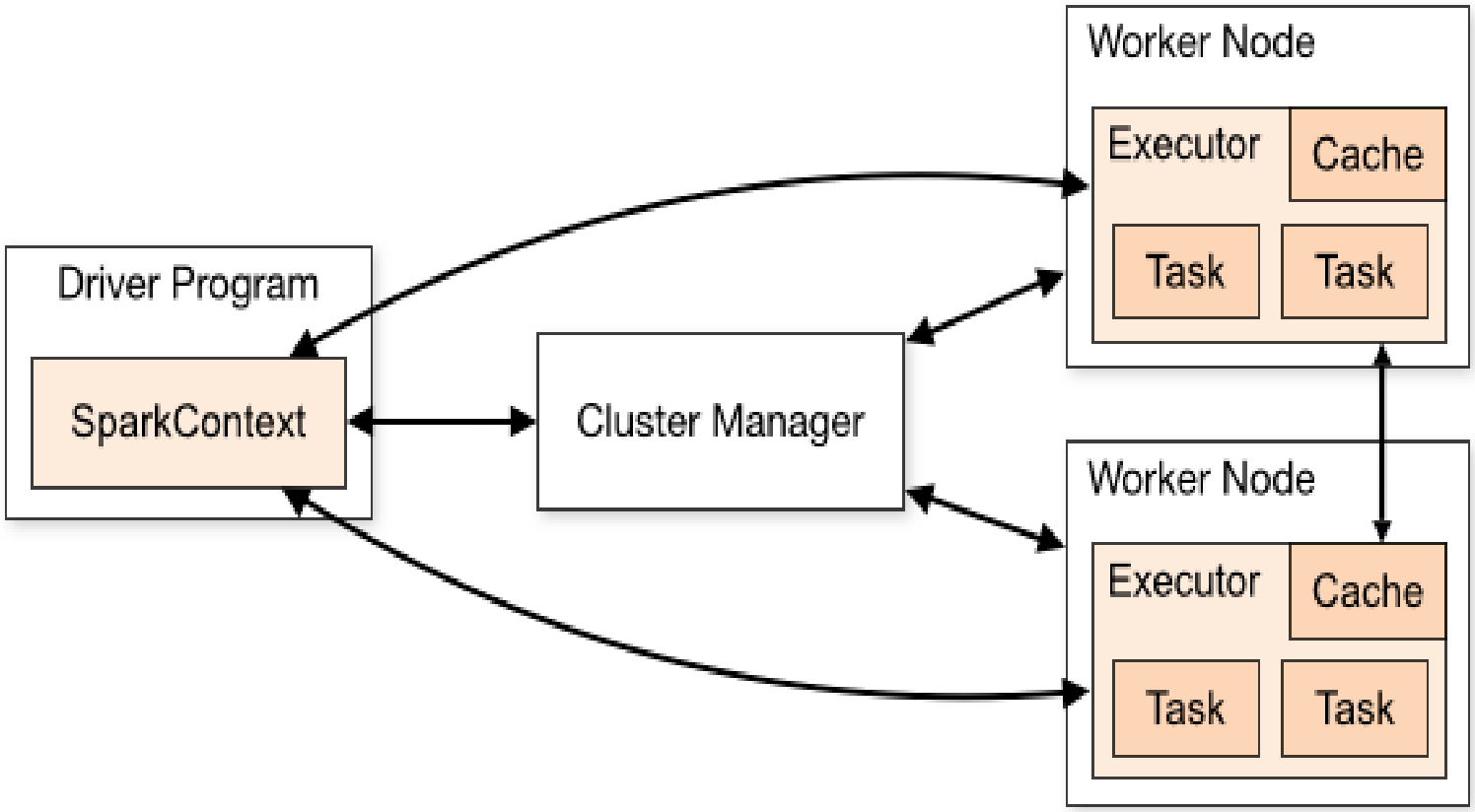
Cont…

- **Standalone:** Spark Standalone deployment means Spark occupies the place on top of HDFS(Hadoop Distributed File System) and space is allocated for HDFS, explicitly. Here, Spark and MapReduce will run side by side to cover all spark jobs on cluster.

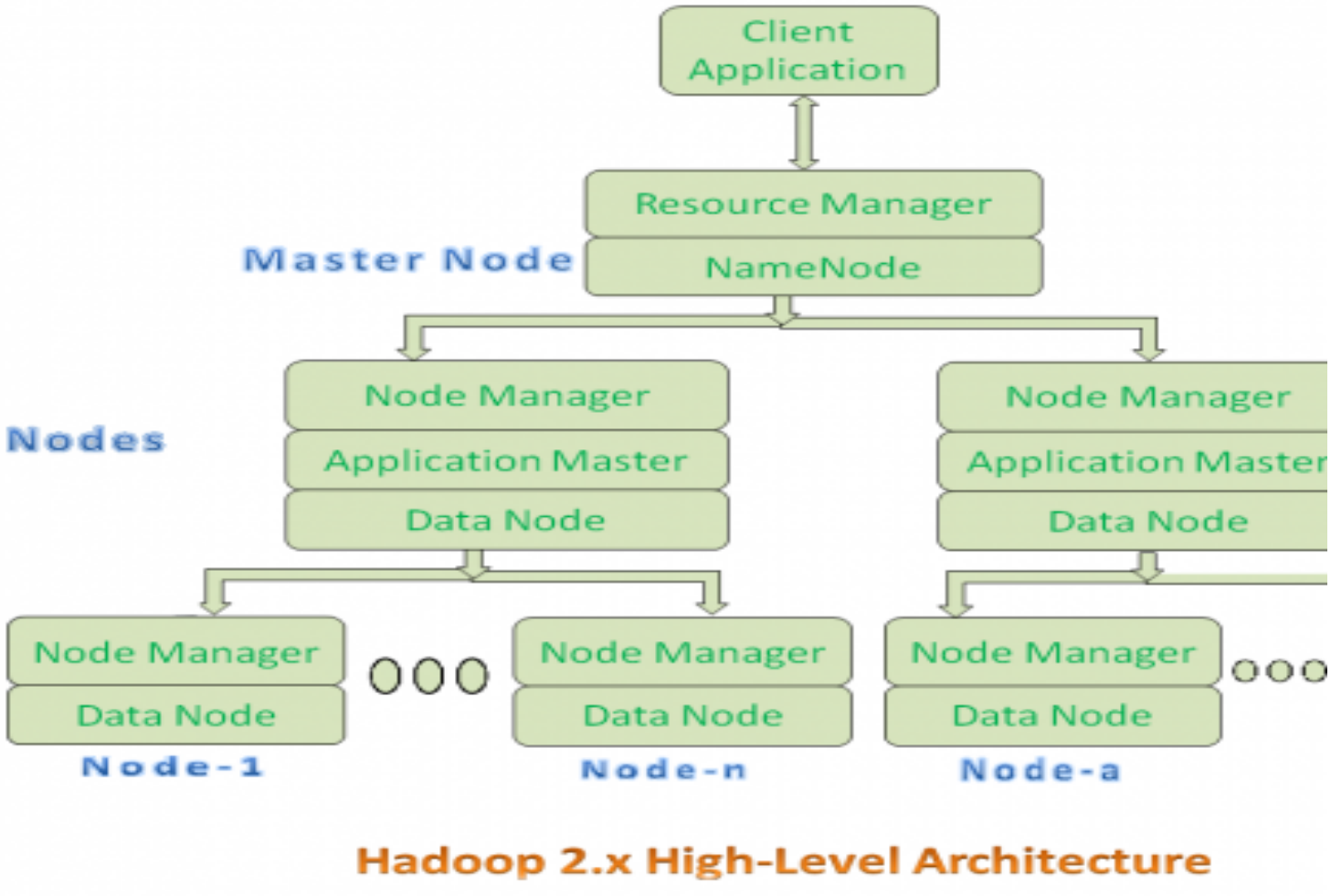
- **Hadoop Yarn:** Hadoop Yarn deployment means, simply, spark runs on Yarn without any pre-installation or root access required. It helps to integrate Spark into Hadoop ecosystem or Hadoop stack. It allows other components to run on top of stack.

- **Spark in MapReduce (SIMR):** Spark in MapReduce is used to launch spark job in addition to standalone deployment. With SIMR, user can start Spark and uses its shell without any administrative access.

SETUP SPARK CLUSTER ON MULTIPLE MACHINES



Hadoop 2.x Architecture



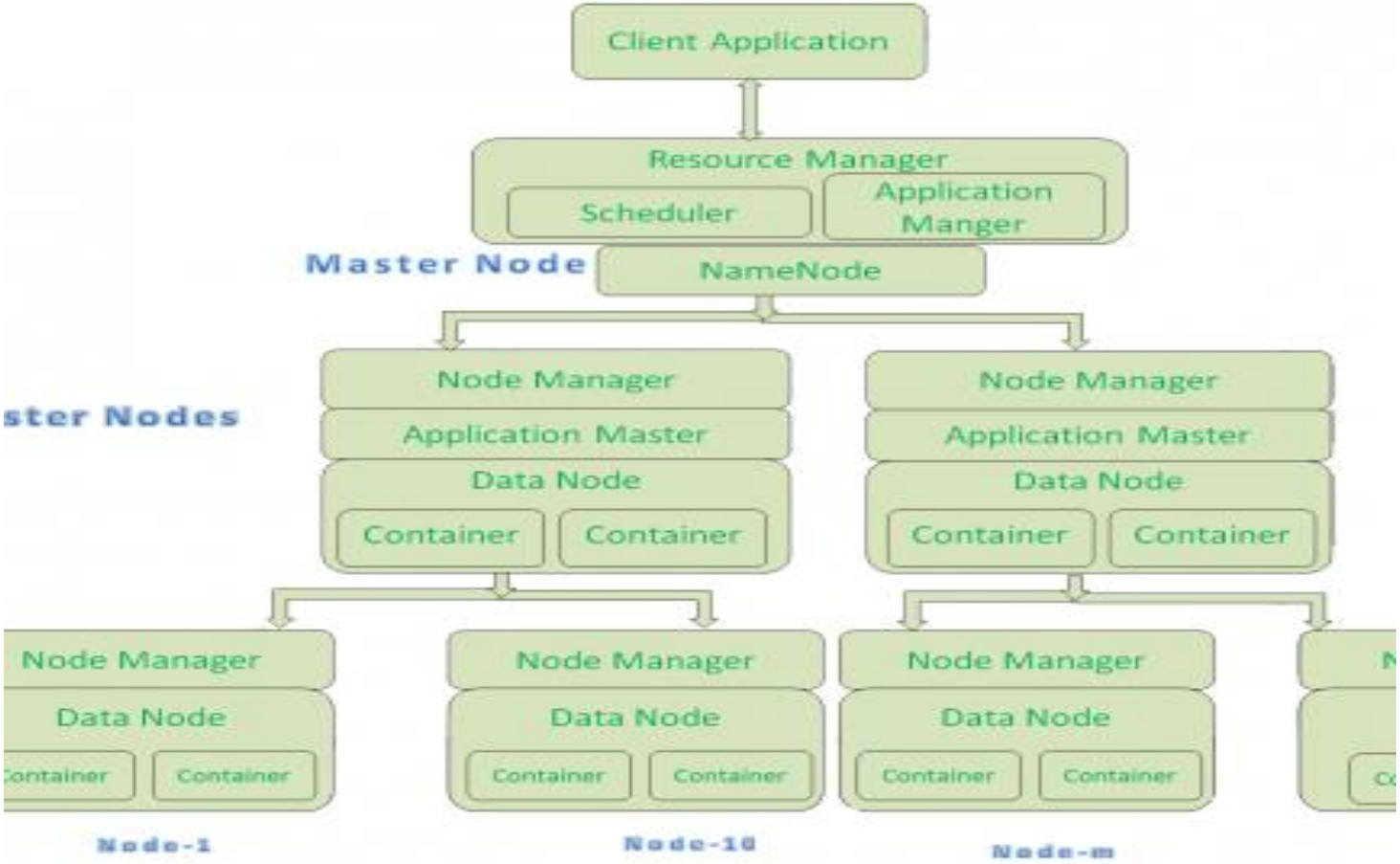
Cont…

- All Master Nodes and Slave Nodes contains both MapReduce and HDFS Components.
- One Master Node has two components:
  - Resource Manager(YARN or MapReduce v2)
  - HDFS
- It's HDFS component is also knows as NameNode. It's NameNode is used to store Meta Data.
- In Hadoop 2.x, some more Nodes acts as Master Nodes as shown in the above diagram. Each this 2nd level Master Node has 3 components:

Cont…

  - Node Manager
  - Application Master
  - Data Node
- Each this 2nd level Master Node again contains one or more Slave Nodes as shown in the above diagram.
- These Slave Nodes have two components:
  - Node Manager
  - HDFS
- It's HDFS component is also knows as Data Node. It's Data Node component is used to store actual our application Big Data. These nodes does not contain Application Master component.

Hadoop Indetail Architecture



Hadoop 2.x In-Detail Architecture

Hadoop 2.x Architecture Description

- **Resource Manager:**
- Resource Manager is a Per-Cluster Level Component.
- Resource Manager is again divided into two components:
  - Scheduler
  - Application Manager
- Resource Manager's Scheduler is :
  - Responsible to scheduler required resources to Applications (that is Per-Application Master).
  - It does only scheduling.
  - It does care about monitoring or tracking of those Applications.

Cont…

Container:

- Each Master Node or Slave Node contains set of Containers. In this diagram, Main Node's Name Node is not showing the Containers. However, it also contains a set of Containers.
- Container is a portion of Memory in HDFS (Either Name Node or Data Node).

Node Manager:

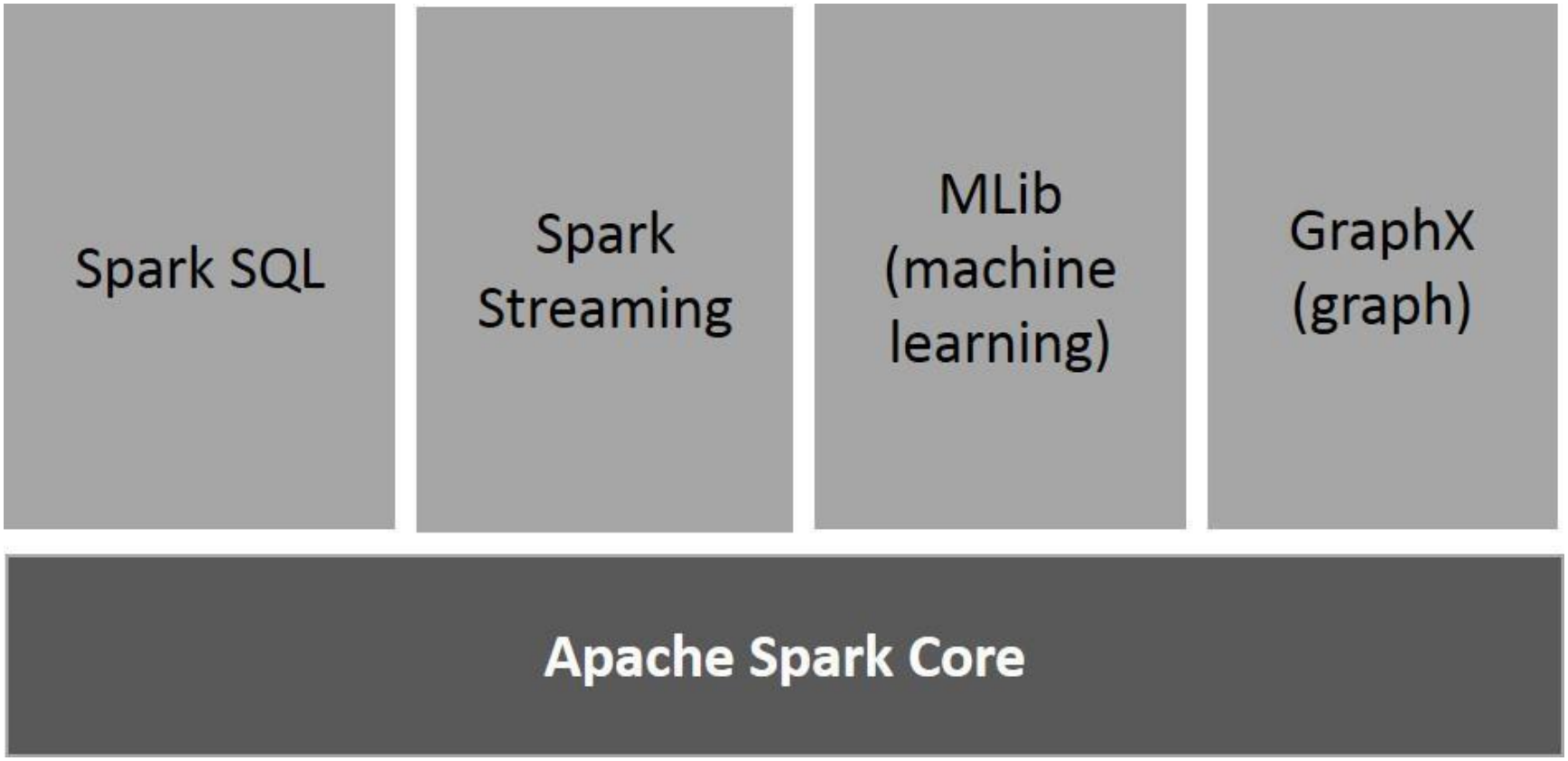
- Node Manager is a Per-Node Level component.
- It is responsible for:
  - Managing the life-cycle of the Container.
  - Monitoring each Container's Resources utilization.

Cont…

Application Master:

- Application Master is a per-application level component. It is responsible for:
  - Managing assigned Application Life cycle.
  - It interacts with both Resource Manager's Scheduler and Node Manager
  - It interacts with Scheduler to acquire required resources.
  - It interacts with Node Manager to execute assigned tasks and monitor those task's status.

Components of Spark



Cont...

Apache Spark Core

Spark Core is the underlying general execution engine for spark platform that all other functionality is built upon. It provides In-Memory computing and referencing datasets in external storage systems.

Spark SQL

Spark SQL is a component on top of Spark Core that introduces a new data abstraction called Schema RDD, which provides support for structured and semi-structured data.

Spark Streaming

Spark Streaming leverages Spark Core's fast scheduling capability to perform streaming analytics. It ingests data in mini-batches and performs RDD (Resilient Distributed Datasets) transformations on those mini-batches of data.

Cont…

MLlib (Machine Learning Library)

MLlib is a distributed machine learning framework above Spark because of the distributed memory-based Spark architecture. It is, according to benchmarks, done by the MLlib developers against the Alternating Least Squares (ALS) implementations. Spark MLlib is nine times as fast as the Hadoop disk-based version of **Apache Mahout** (before Mahout gained a Spark interface).

GraphX

GraphX is a distributed graph-processing framework on top of Spark. It provides an API for expressing graph computation that can model the user-defined graphs by using Pregel abstraction API. It also provides an optimized runtime for this abstraction.

SPARK-RDD

- Resilient Distributed Datasets (RDD) is a fundamental data structure of Spark.
- It is an immutable distributed collection of objects.
- Each dataset in RDD is divided into logical partitions, which may be computed on different nodes of the cluster.
- RDDs can contain any type of Python, Java, or Scala objects, including user-defined classes.

Cont…

- RDD is a read-only, partitioned collection of records.
- RDDs can be created through deterministic operations on either data on stable storage or other RDDs.
- RDD is a fault-tolerant collection of elements that can be operated on in parallel.
- There are two ways to create RDDs:
  - **Parallelizing** an existing collection in your driver program,
  - **Referencing a dataset** in an external storage system, such as a shared file system, HDFS, HBase, or any data source offering a Hadoop Input Format.

The features of RDDs (decomposing the name):

- Resilient, i.e. fault-tolerant with the help of RDD Lineage graph and so able to recompute missing or damaged partitions due to node failures.

- Distributed with data residing on multiple nodes in a cluster.

- Dataset is a collection of partitioned data with primitive values or values of values, e.g. tuples or other objects (that represent records of the data you work with).

Additional Traits

- **In-Memory**, i.e. data inside RDD is stored in memory as much (size) and long (time) as possible.
- **Immutable** or **Read-Only**, i.e. it does not change once created and can only be transformed using transformations to new RDDs.
- **Lazy evaluated**, i.e. the data inside RDD is not available or transformed until an action is executed that triggers the execution.
- **Cacheable**, i.e. you can hold all the data in a persistent "storage" like memory (default and the most preferred) or disk (the least preferred due to access speed).
- **Parallel**, i.e. process data in parallel.
- **Typed** — RDD records have types, e.g. Long in RDD[Long] or (Int, String) in RDD[(Int, String)].
- **Partitioned** — records are partitioned (split into logical partitions) and distributed across nodes in a cluster.

Map Reduce Operations

- **Data Sharing is Slow in Map Reduce**

Map Reduce is widely adopted for processing and generating large datasets with a parallel, distributed algorithm on a cluster.

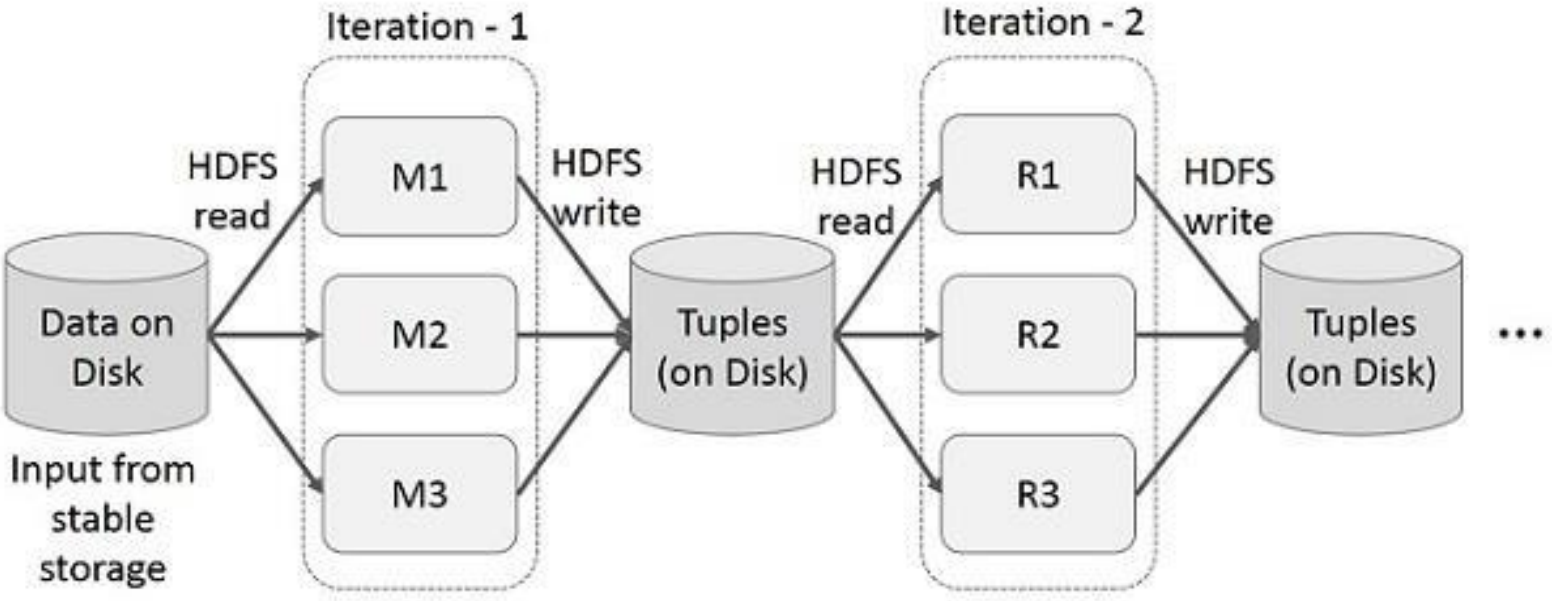
It allows users to write parallel computations, using a set of high-level operators, without having to worry about work distribution and fault tolerance.

In most current frameworks, the only way to reuse data between computations (Ex: between two Map Reduce jobs) is to write it to an external stable storage system (Ex: HDFS).

Both **Iterative** and **Interactive** applications require faster data sharing across parallel jobs. Data sharing is slow in MapReduce due to **replication, serialization, and disk IO.**

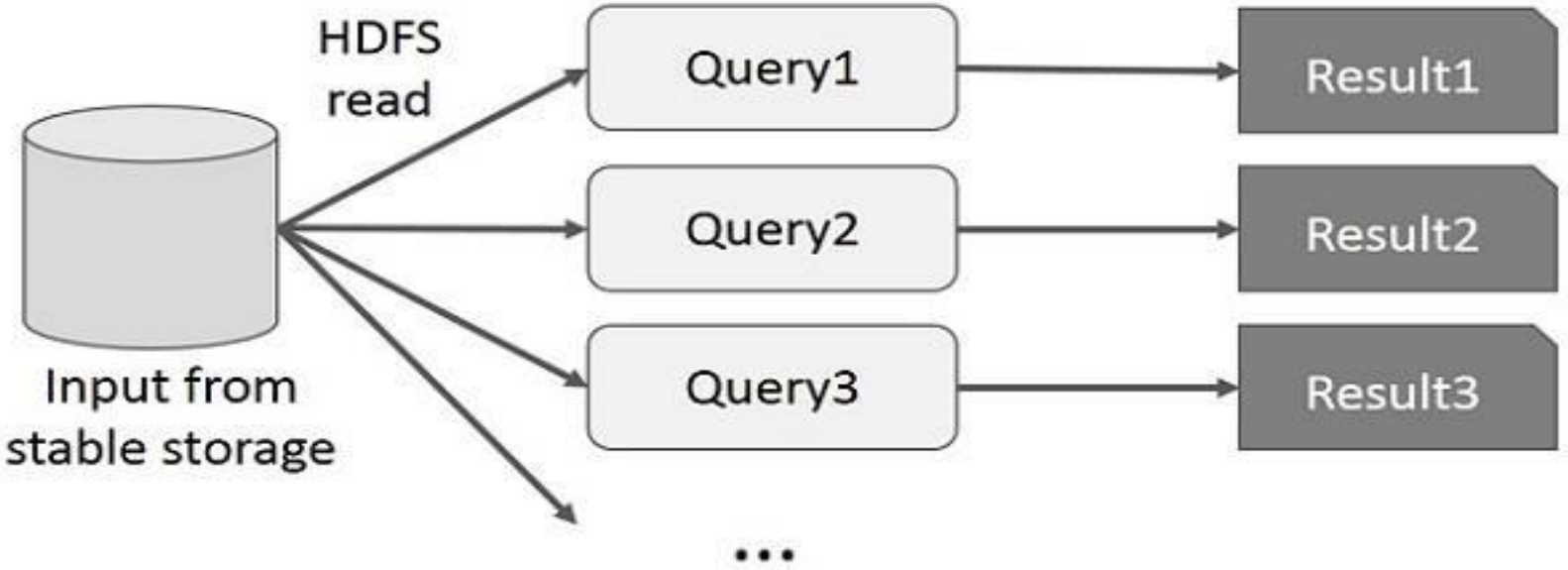
Iterative Operations on MapReduce

- Reuse intermediate results across multiple computations in multi-stage applications. The following illustration explains how the current framework works, while doing the iterative operations on MapReduce. This incurs substantial overheads due to data replication, disk I/O, and serialization, which makes the system slow.



Interactive Operations on MapReduce

- User runs ad-hoc queries on the same subset of data. Each query will do the disk I/O on the stable storage, which can dominate application execution time.
- The following illustration explains how the current framework works while doing the interactive queries on MapReduce.

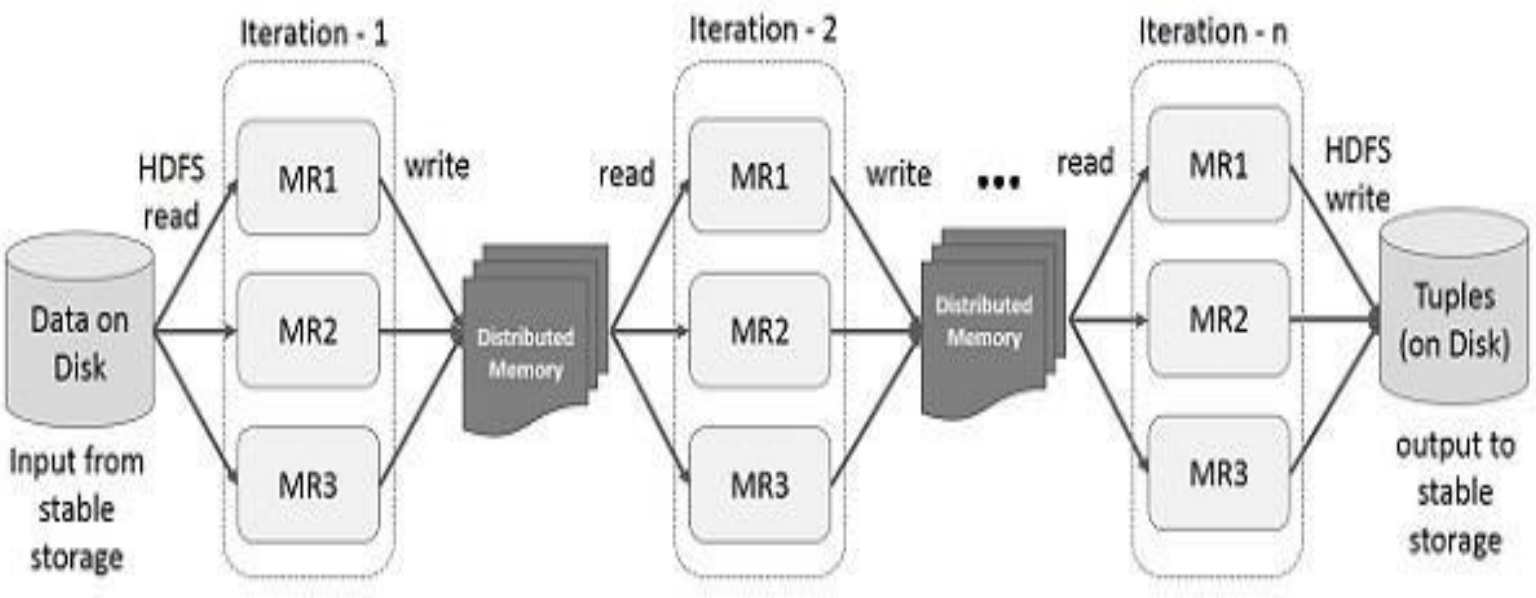


Data Sharing using Spark RDD

- Data sharing is slow in MapReduce due to **replication**, **serialization**, and **disk IO**.
- Recognizing this problem, researchers developed a specialized framework called Apache Spark.
- The key idea of spark is Resilient Distributed Datasets (RDD); it supports in-memory processing computation.
- This means, it stores the state of memory as an object across the jobs and the object is sharable between those jobs.
- Data sharing in memory is 10 to 100 times faster than network and Disk.

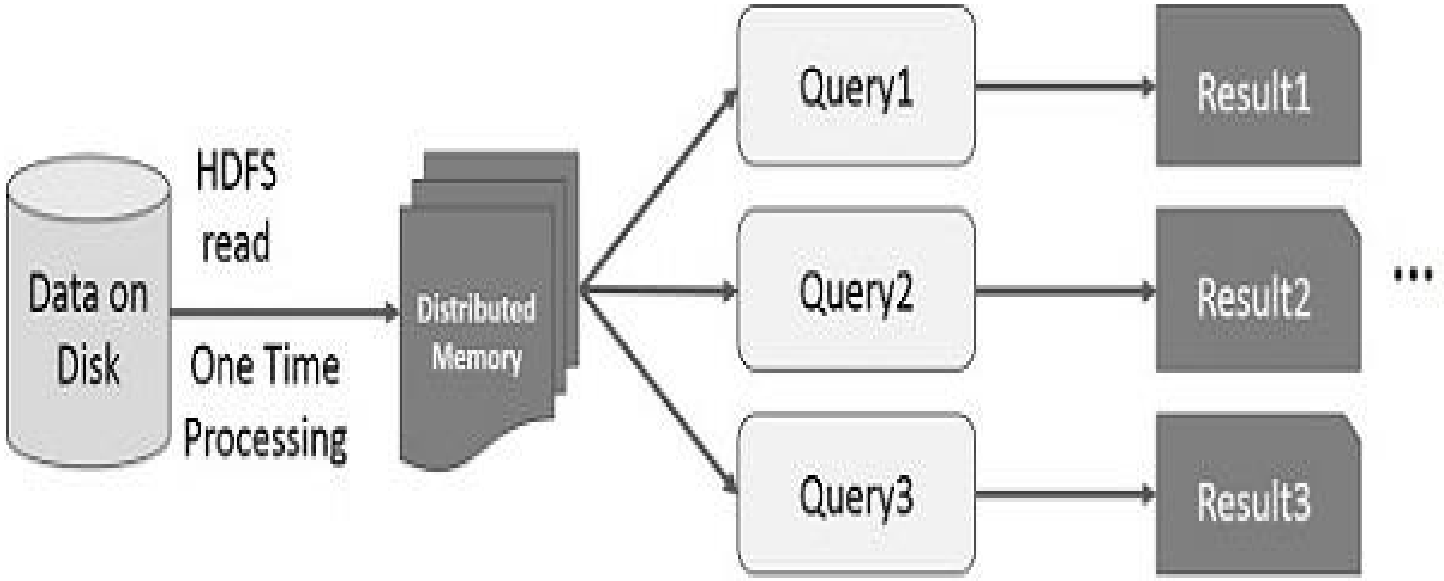
Iterative Operations on Spark RDD

- The illustration given below shows the iterative operations on Spark RDD. It will store intermediate results in a distributed memory instead of Stable storage (Disk) and make the system faster.



Interactive Operations on Spark RDD

- This illustration shows interactive operations on Spark RDD. If different queries are run on the same set of data repeatedly, this particular data can be kept in memory for better execution times.



Apache Spark Installation

Spark is Hadoop's sub-project. Therefore, it is better to install Spark into a Linux based system.

Verify Java Installation

Verify Scala Installation

Download Latest Scala

Installing Scala

Extract the Scala tar file

Downloading Apache Spark

Installing Spark

Verifying Spark Installation

CORE Programming

- Spark Core is the base of the whole project.
- It provides distributed task dispatching, scheduling, and basic I/O functionalities. Spark uses a specialized fundamental data structure known as RDD (Resilient Distributed Datasets) that is a logical collection of data partitioned across machines.
- RDDs can be created in two ways; one is by referencing datasets in external storage systems and second is by applying transformations (e.g. map, filter, reducer, join) on existing RDDs.

Spark Shell

- Spark provides an interactive shell: a powerful tool to analyze data interactively.
- It is available in either Scala or Python language.
- Spark's primary abstraction is a distributed collection of items called a Resilient Distributed Dataset (RDD).
- RDDs can be created from Hadoop Input Formats (such as HDFS files) or by transforming other RDDs

RDD Transformations

- RDD transformations returns pointer to new RDD and allows you to create dependencies between RDDs. Each RDD in dependency chain (String of Dependencies) has a function for calculating its data and has a pointer (dependency) to its parent RDD.

- Spark is lazy, so nothing will be executed unless you call some transformation or action that will trigger job creation and execution.

Sr.No	Transformations & Meaning
1	**map(func)** Returns a new distributed dataset, formed by passing each element of the source through a function func.
2	**filter(func)** Returns a new dataset formed by selecting those elements of the source on which func returns true.
3	**flatMap(func)** Similar to map, but each input item can be mapped to 0 or more output items (so func should return a Seq rather than a single item).
4	**intersection(otherDataset)** Returns a new RDD that contains the intersection of elements in the source dataset and the argument.
5	**union(otherDataset)** Returns a new dataset that contains the union of the elements in the source dataset and the argument.

Sr.No	Actions
1	**collect()** Returns all the elements of the dataset as an array at the driver program. This is usually useful after a filter or other operation that returns a sufficiently small subset of the data.
2	**count()** Returns the number of elements in the dataset.
3	**first()** Returns the first element of the dataset(similar to take (1)).
4	**take(n)** Returns an array with the first n elements of the dataset.
5	**saveAsTextFile(path)** Writes the elements of the dataset as a text file (or set of text files) in a given directory in the local filesystem, HDFS or any other Hadoop-supported file system. Spark calls toString on each element to convert it to a line of text in the file.