LECTURE-10
LECTURE 9 - BIG DATA IN CLOUDS (APACHE HADOOP & APACHE SPARK) PART 2

CSCI 5408:

Data Management, Warehousing, and Analytics

Prepared By: Suhaib Qaiser (suhaibqaiser@dal.ca)

Big Data Overview

Recap from last lecture …

Q1. Describe three Vs of Big Data?

Q2. What is Introspection?

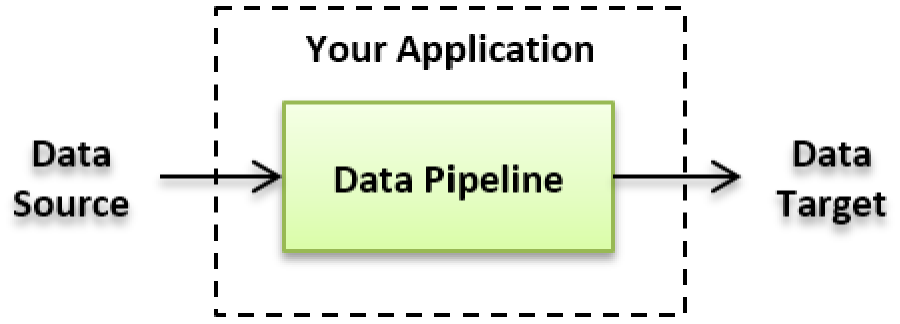
Q3. What is Longevity?

Q4. Tell me a real example of Predictive Marketing in Big Data?

Data Pipelines

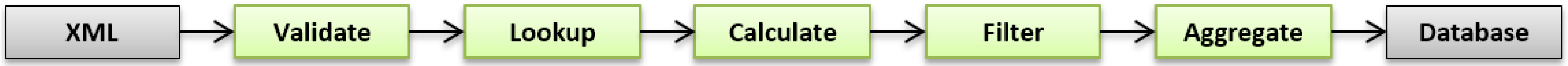
What is Data Pipeline?

Data Pipeline is an embedded data processing engine for the several computing nodes connected together in an order. The engine runs inside your applications, APIs, and jobs to filter, transform, and migrate data on-the-fly.



Here are a few things you can do with Data Pipeline.

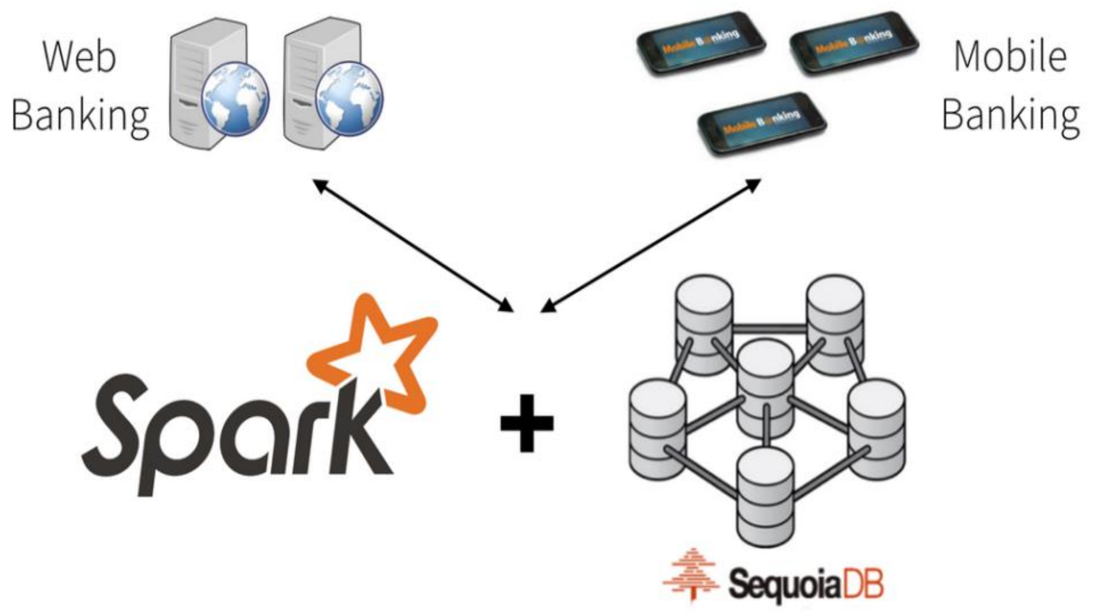
1. Convert incoming data to a common format.
2. Prepare data for analysis and visualization.
3. Migrate between databases.
4. Share data processing logic across web apps, batch jobs, and APIs.
5. Power your data ingestion and integration tools.
6. Consume large XML, CSV, and fixed-width files.
7. Replace batch jobs with real-time data.



Source: https://northconcepts.com/docs/what-is-it

Apache Sparks

Current Architecture



All 15 Years of data quickly accessible on 1PB of disk space

APACHE SPARKS

What is Apache Sparks

Apache Sparks is a cluster computing platform designed to be fast and general purpose

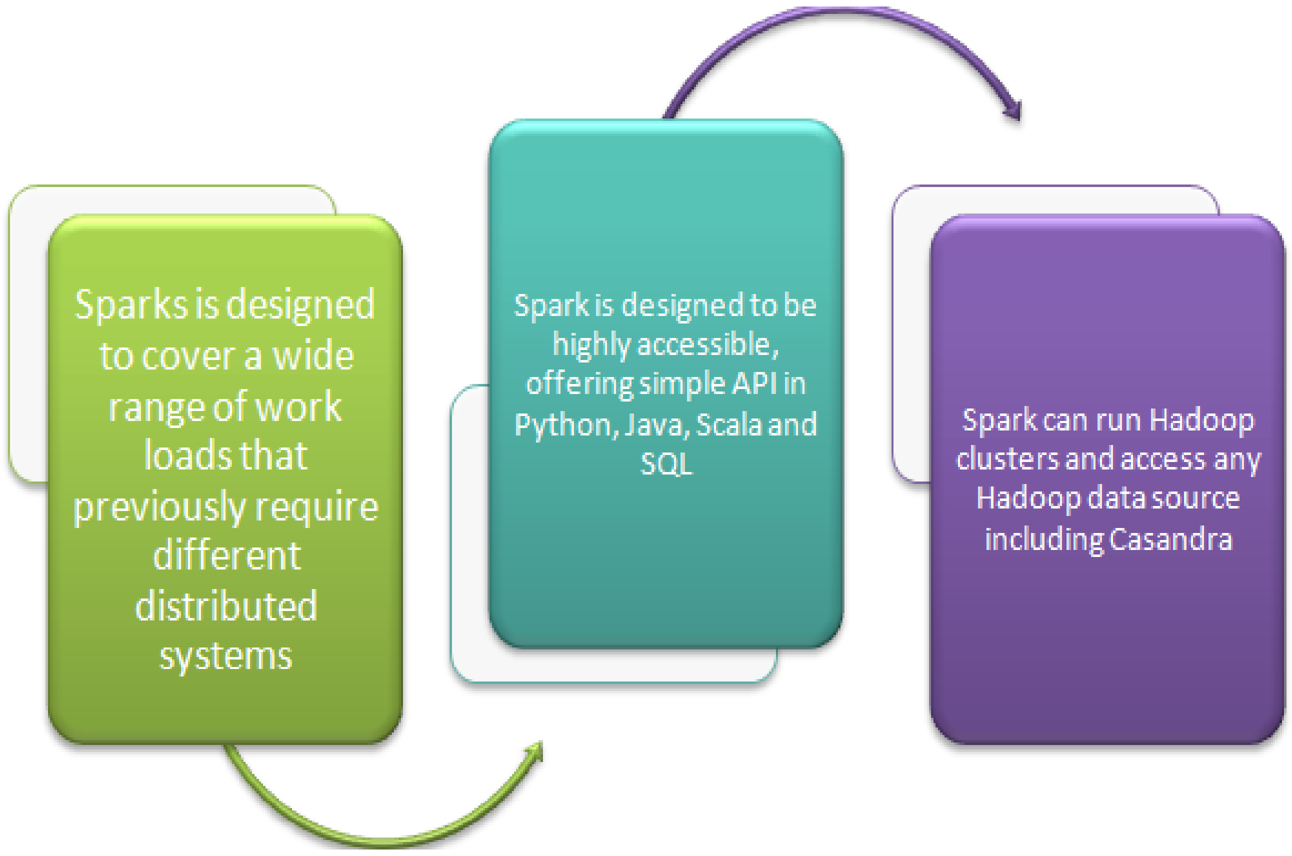
Sparks extends the popular MapReduce model to efficiently support more types of computations, including interactive quries and stream processing

Sparks has the ability to run computations in memory

Sparks can perform complex computations more efficiently on disk

Apache Sparks

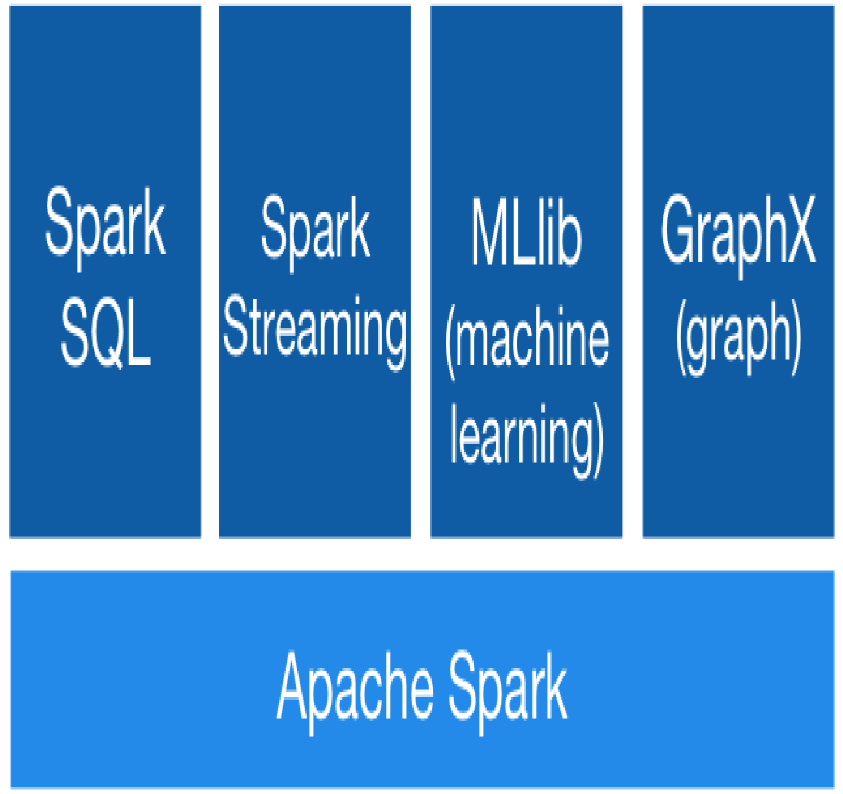
Features



Sparks is designed to cover a wide range of work loads that previously require different distributed systems

Spark is designed to be highly accessible, offering simple API in Python, Java, Scala and SQL

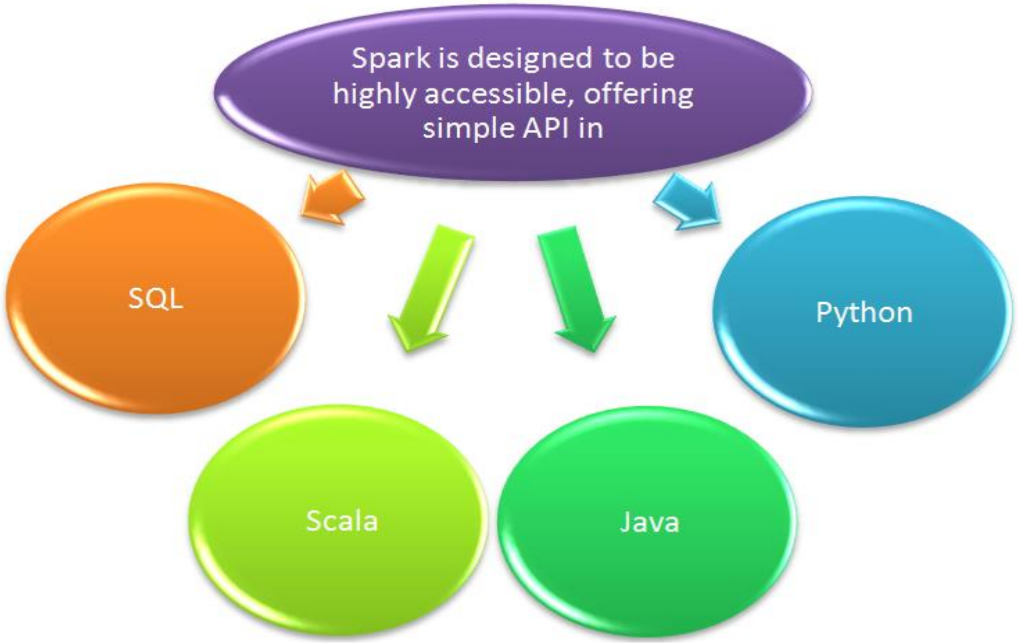
Spark can run Hadoop clusters and access any Hadoop data source including Casandra

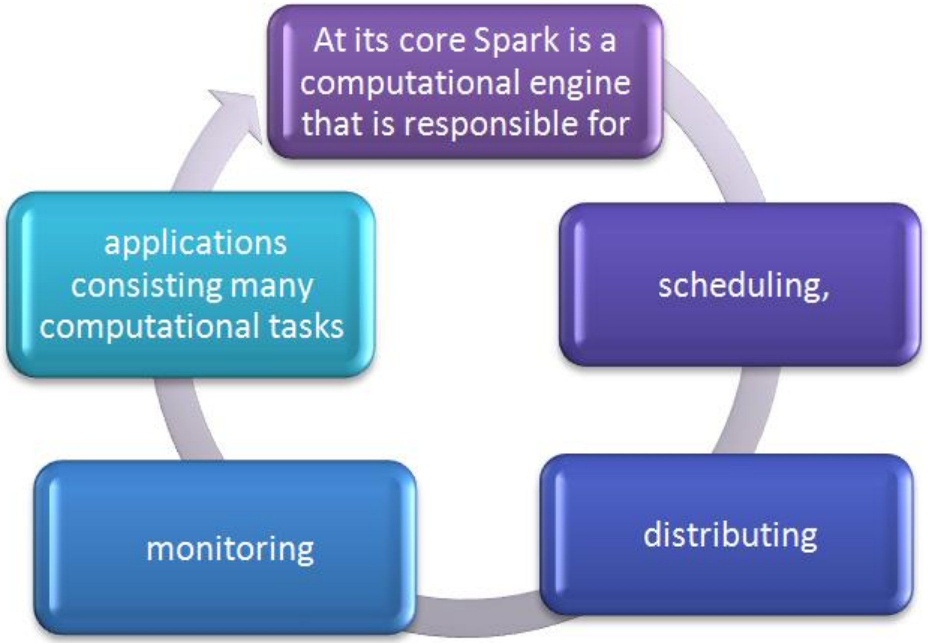


APACHE SPARKS

What is Apache Spark

Spark project contains multiple closely integrated components


Spark is designed to be highly accessible, offering simple API in
SQL
Scala
Java
Python

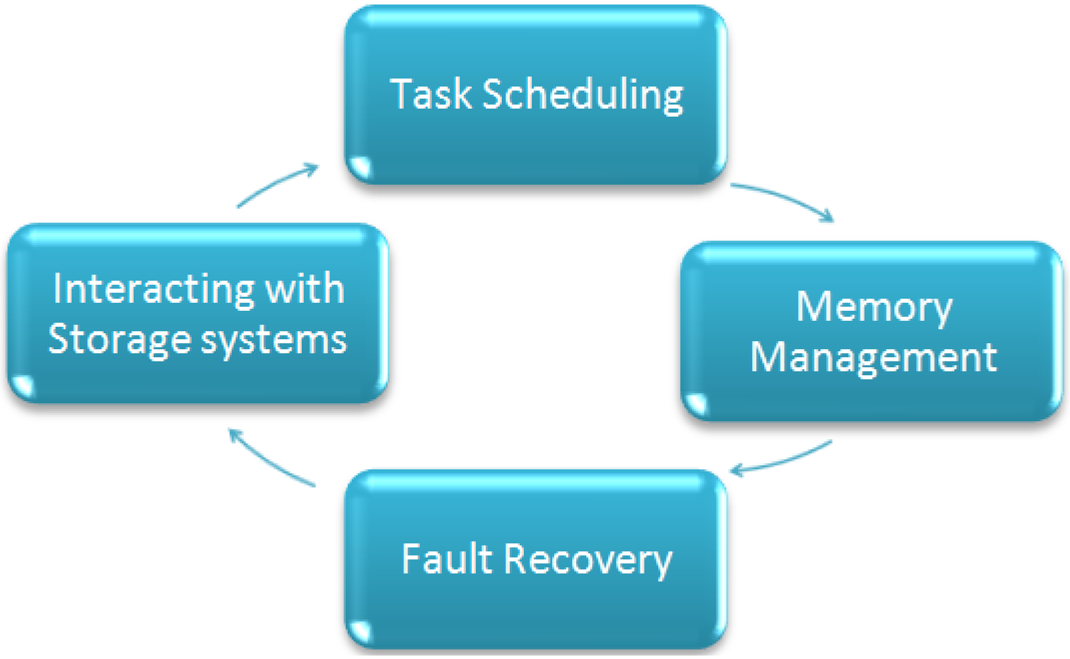

At its core Spark is a computational engine that is responsible for
scheduling,
distributing
monitoring
applications consisting many computational tasks

Spark can run Hadoop clusters and access any Hadoop data source including Casandra

One of the largest advantages of tight integration is the ability to build applications that can seamlessly combine different programming models

Spark Core

Spark core contains the basic functionality of Spark



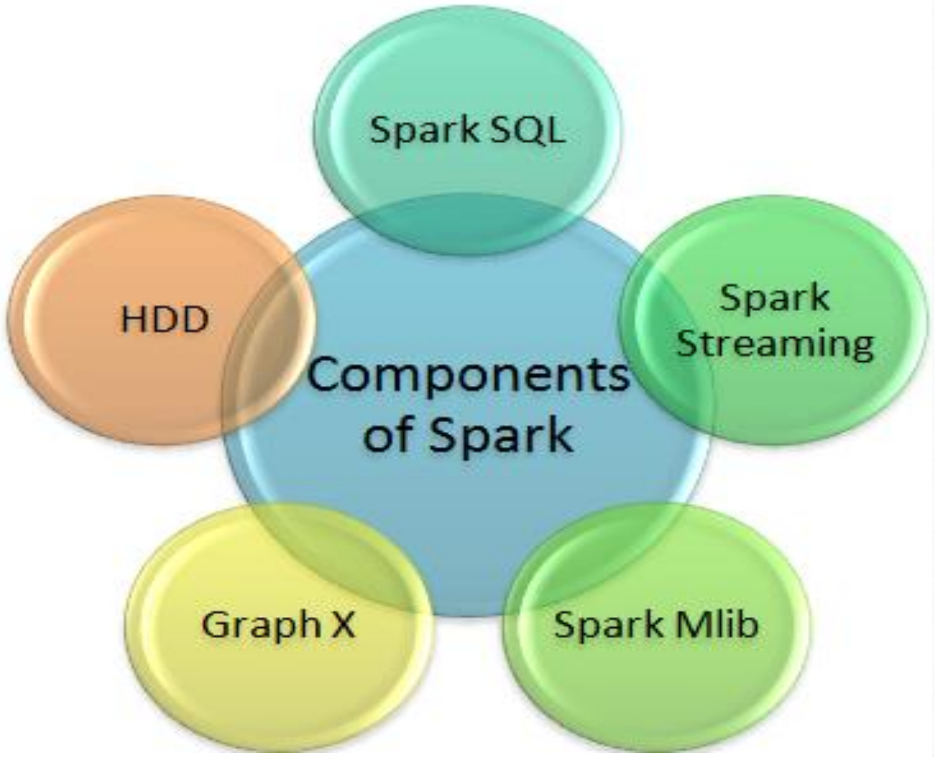
RDD represents a collection of items distributed across many computation nodes

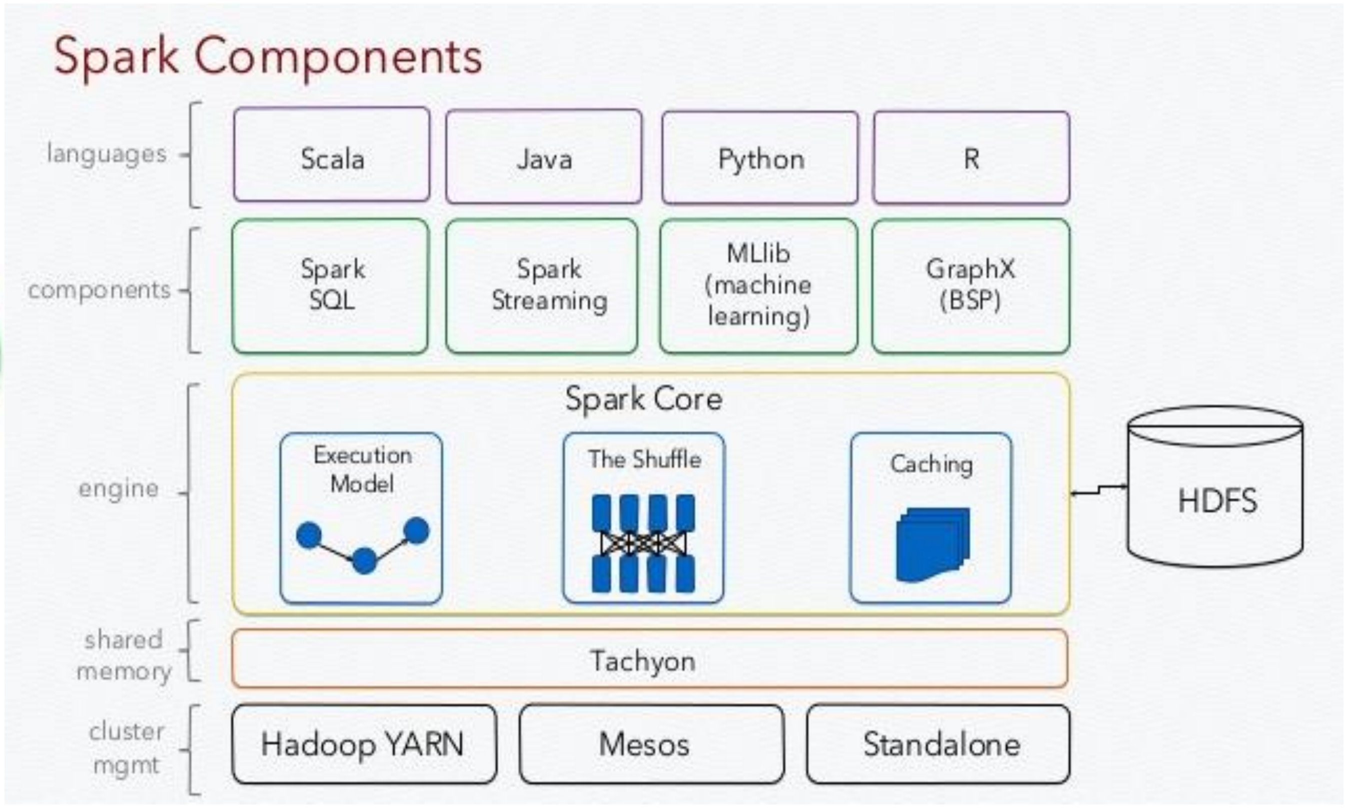
Spark provides many APIs for building and manipulating these collections

It is also the home to the API that defines “resilient distributed data”

Apache Sparks

Sparks Components



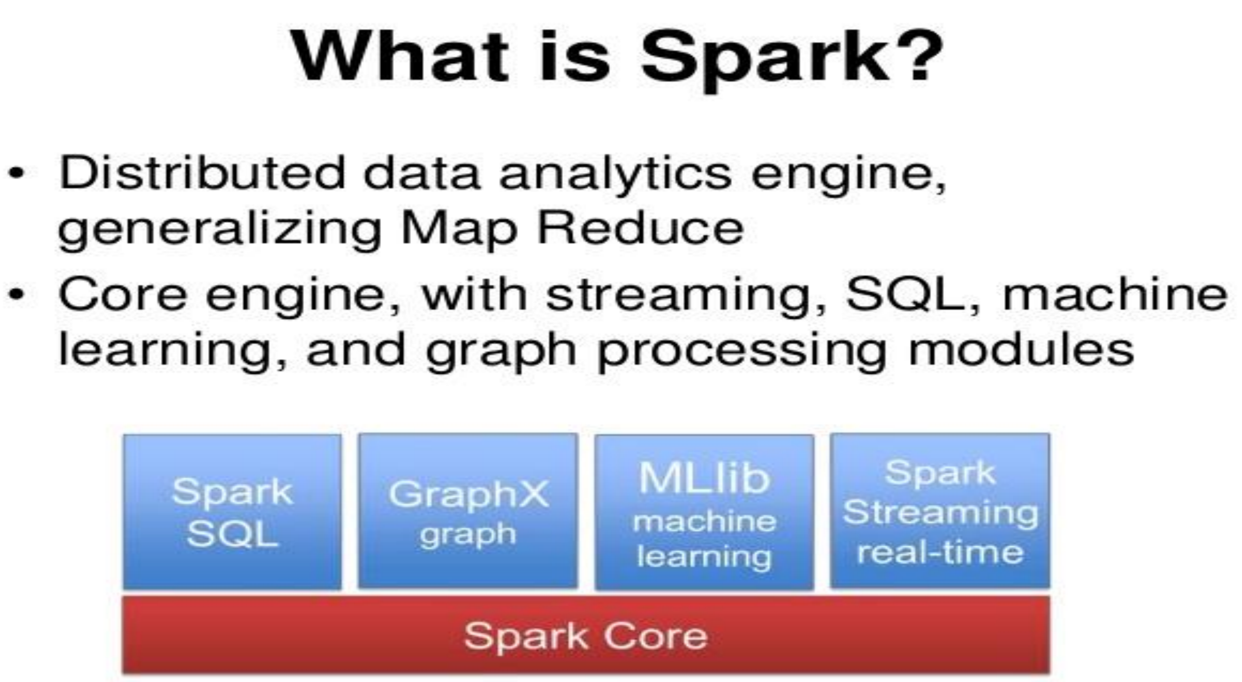


Spark SQL

Spark SQL is Sparks package for working with structured data

It allows querying data via SQL as well as Apache Hive (variant of SQL called Hive Query Language HQL)

Spark SQL allows developers to intermix SQL queries with programmatic data manipulations supported by RDDs in Python, Java and Scala



What is Spark?

- Distributed data analytics engine, generalizing Map Reduce
- Core engine, with streaming, SQL, machine learning, and graph processing modules

Spark SQL
GraphX graph
MLlib machine learning
Spark Streaming real-time
Spark Core

Spark SQL

Spark SQL is Sparks package for working with structured data

Spark SQL is Sparks package for working with structured data

It allows querying data via SQL as well as Apache Hive (variant of SQL called Hive Query Language HQL)

Spark SQL allows developers to intermix SQL queries with programmatic data manipulations supported by RDDs in Python, Java and Scala

Mix any query with Python, Java and Scala

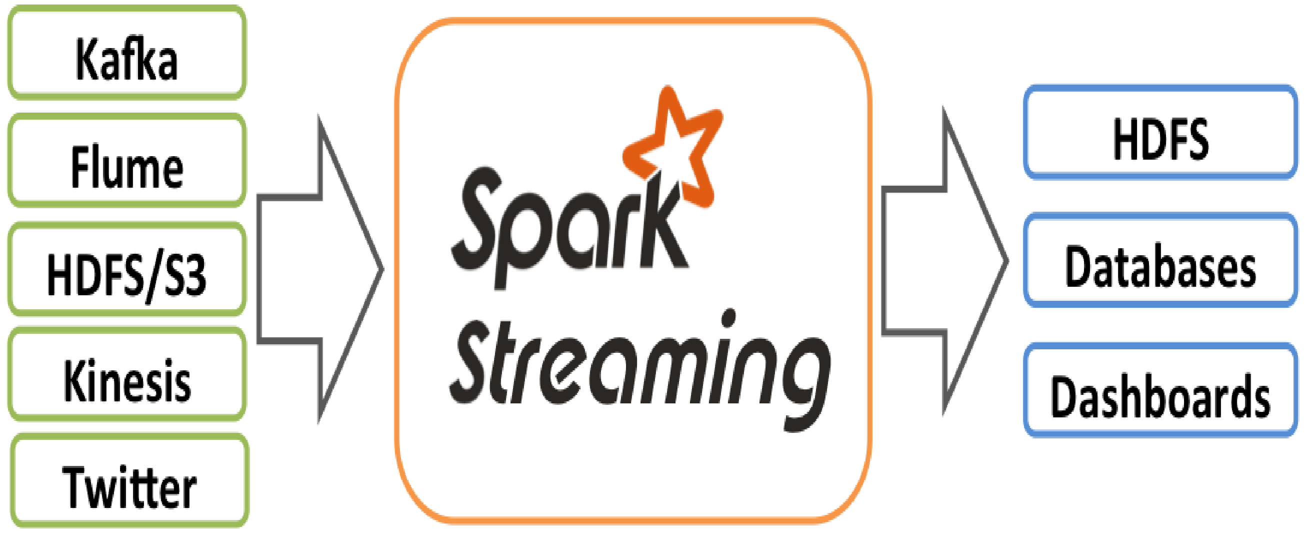
It has unified data access

Spark Streaming

Spark Streaming is a Spark component that enables processing of live streams of data

Examples of data streams includes log files generated by production web servers or queues of messages containing status updates posted by user of a web service

Spark Streaming provides an API for manipulating data streams



RDD Basics

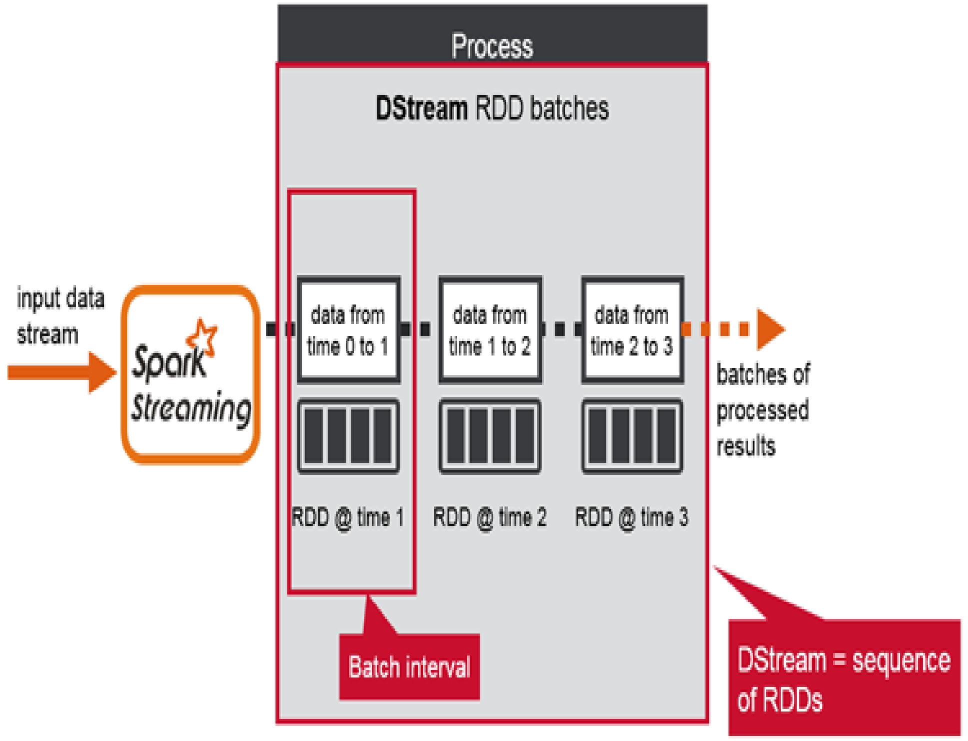
Key Points

An RDD in Spark is simply an immutable distributed collection of objects

Each RDD is split into multiple partitions which may be computed on different nodes on cluster

RDDs can contain any type of Python, Java or Scala objects including user defined classes

Data stream divided into batches of X milliseconds = DStreams



RDD Basics

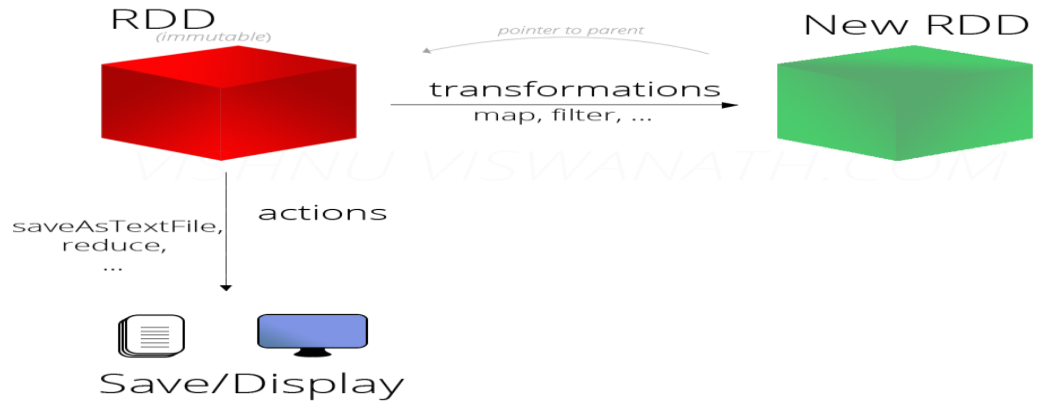
Key Points

User create RDDs in two ways:

- Loading an external dataset
- Distributing a collection of objects

Simplest way to create RDD is through loading existing collection and pass it to SparkContext's parallelize method

Outside of prototyping, this is not widely used since it requires that you have entire dataset in memory

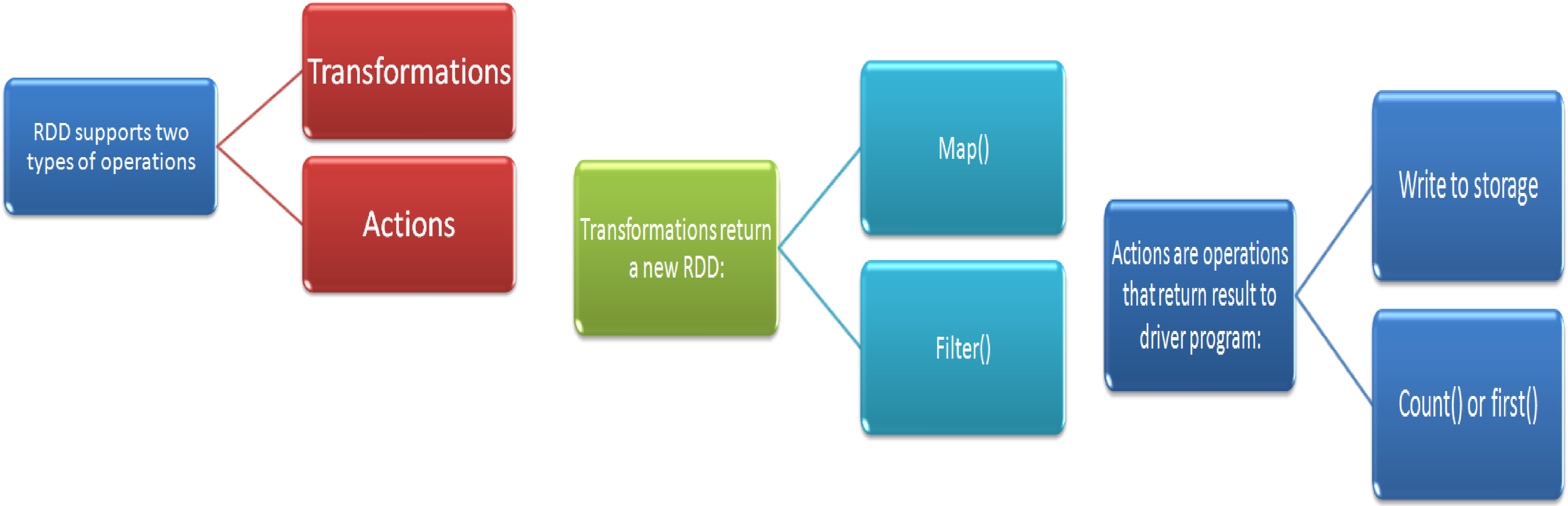


RDD Basics

- Loading a text file as an RDD of strings using SparkContext.textFile()
  - *Creating an RDD of strings with* textFile() *in Python*
    - `>>> lines = sc.textFile("README.md")`
- RDDs offer two types of operations: *transformations* and *actions*
  - *Transformations* construct a new RDD from a previous one
    - `>>> pythonLines = lines.filter(lambda line: "Python" in line)`
  - *Actions*, compute a result based on an RDD
    - `>>> pythonLines.first()`

RDD Operations

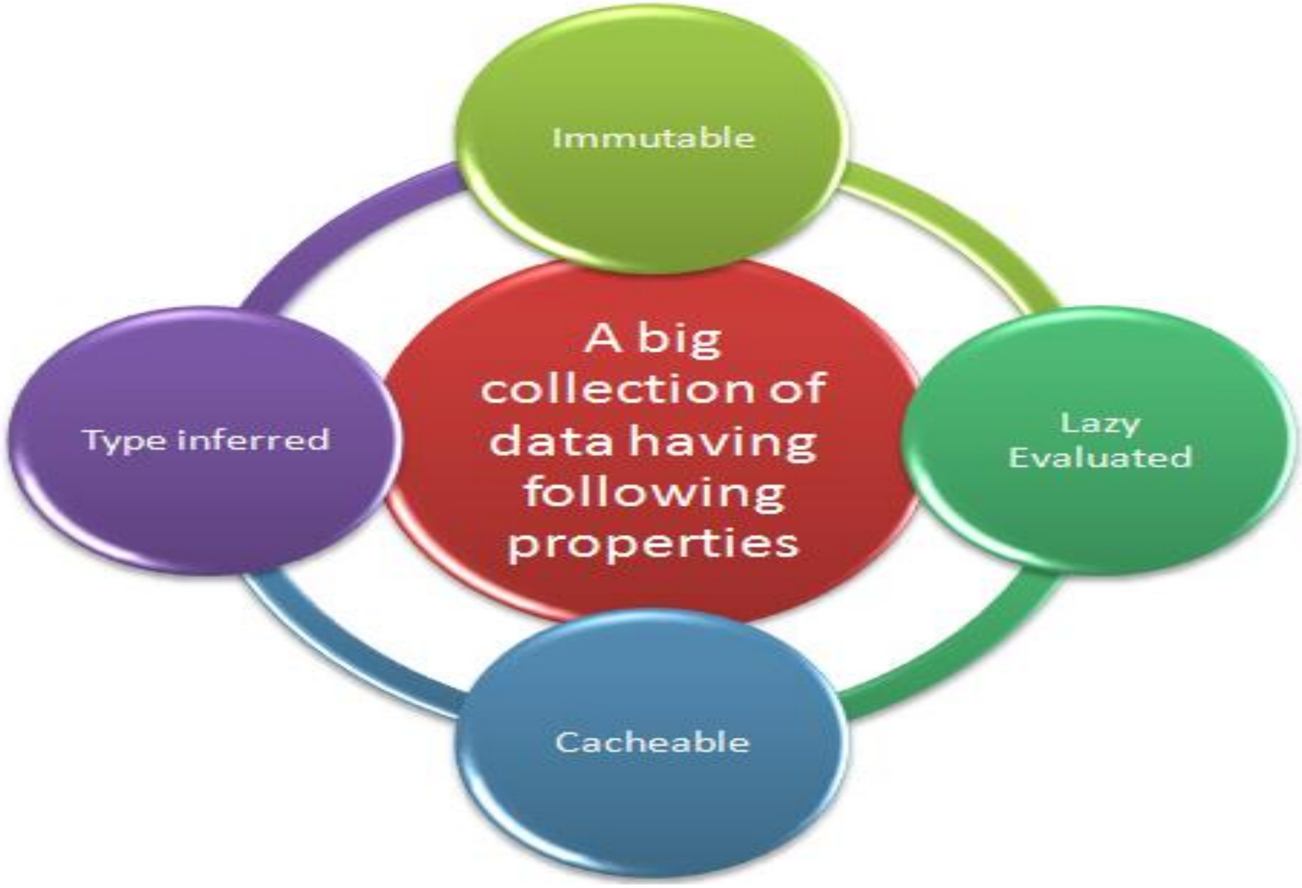
Features

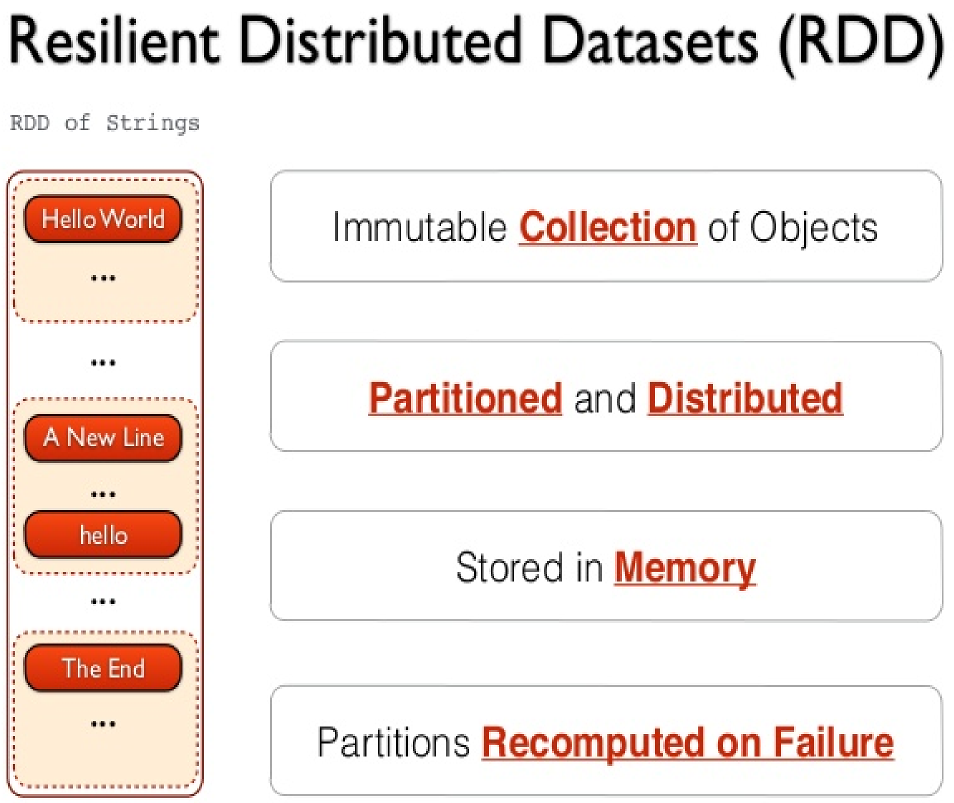

RDD supports two types of operations
Transformations
Actions
Transformations return a new RDD:
Map()
Filter()
Actions are operations that return result to driver program:
Write to storage
Count() or first()

http://www.devx.com/opensource/managing-large-volumes-of-data-with-apache-cassandra-nosql.html

Resilient Distributed Dataset

Features




Resilient Distributed Datasets (RDD)
RDD of Strings
Hello World
...
...
A New Line
...
hello
...
The End
...
Immutable Collection of Objects
Partitioned and Distributed
Stored in Memory
Partitions Recomputed on Failure

Immutability

Once created it would never changes

Reference can change but value cannot change

Immutability helps in parallel computing and caching

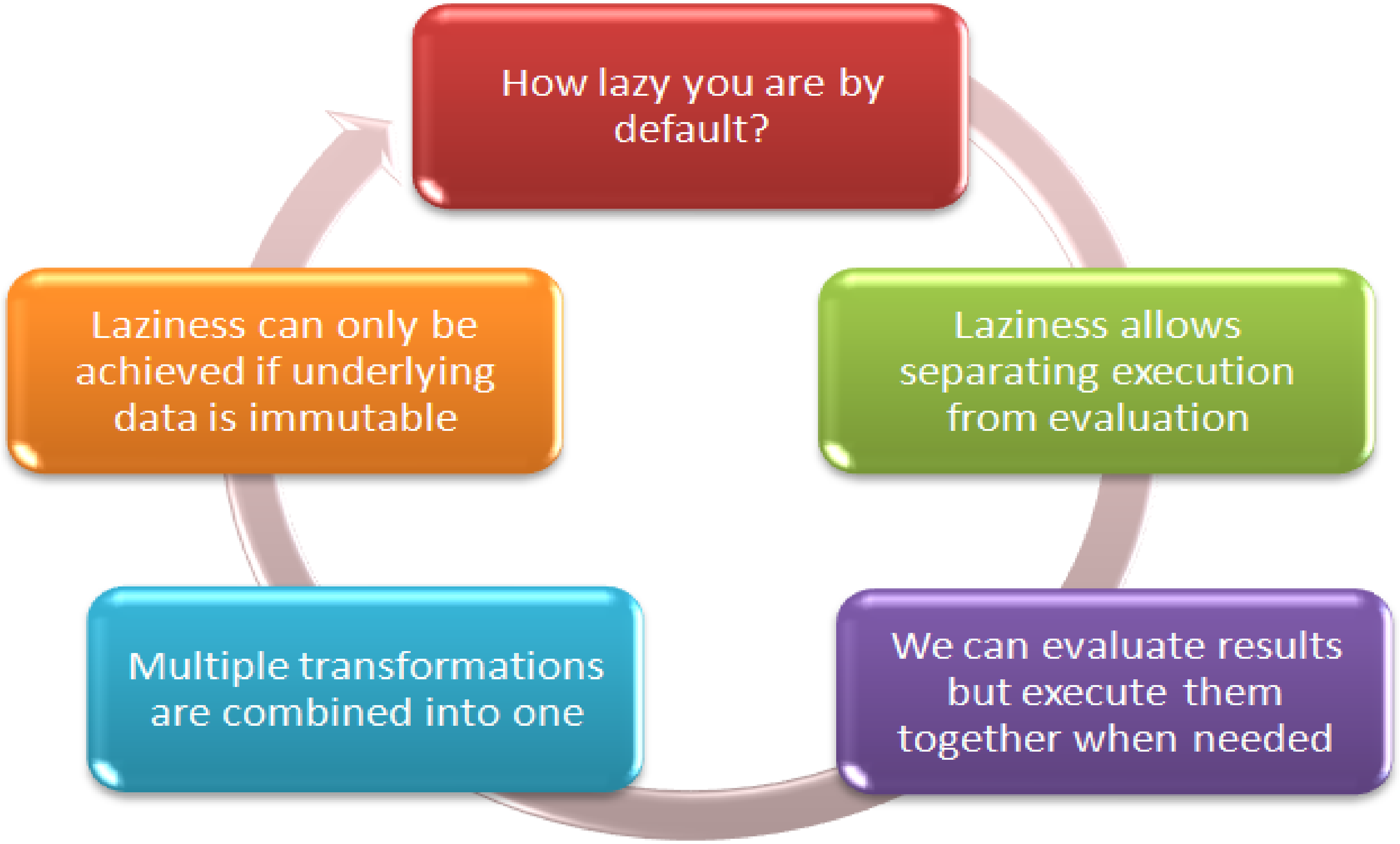
If it is immutable we do not need to worry about updates on sets of data

- This gives freedom of caching values for longer period of time
- This helps is distributing data and perform parallel computing on it
- This help in lazy evaluation of types of data

Immutability is good for parallelism but not good for space

In Big data created multiple copies of huge chunks of data will create performance bottlenecks

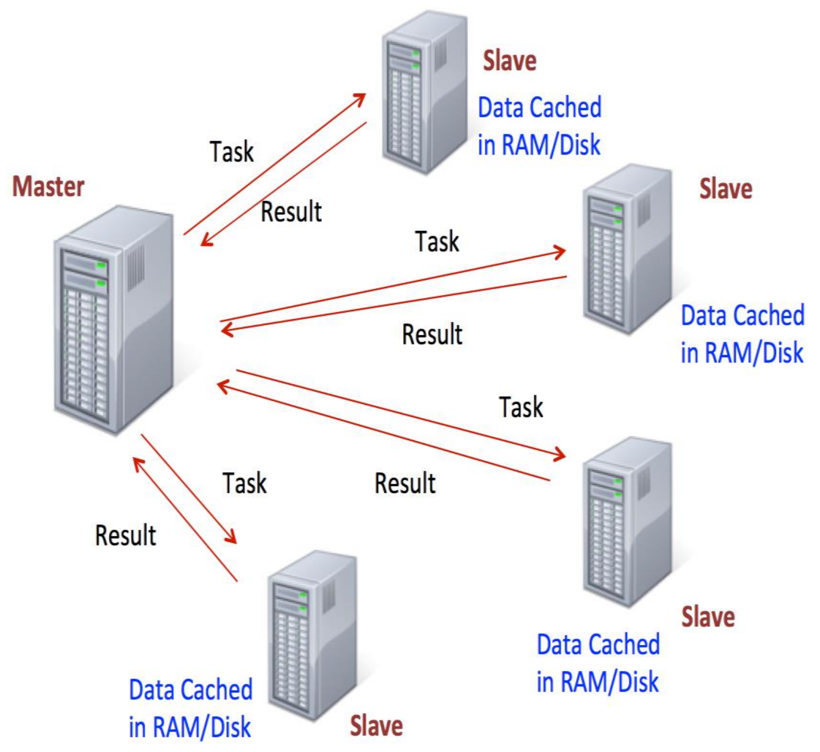
Lazy Evaluation of Transformations


How lazy you are by default?
Laziness allows separating execution from evaluation
We can evaluate results but execute them together when needed
Multiple transformations are combined into one
Laziness can only be achieved if underlying data is immutable

Caching

- Immutable allows you to cache data for long time
- Lazy transformations allows recreate data on failures
- Transformations can saved as well
- Caching data improves execution engine

How does Spark execute a job



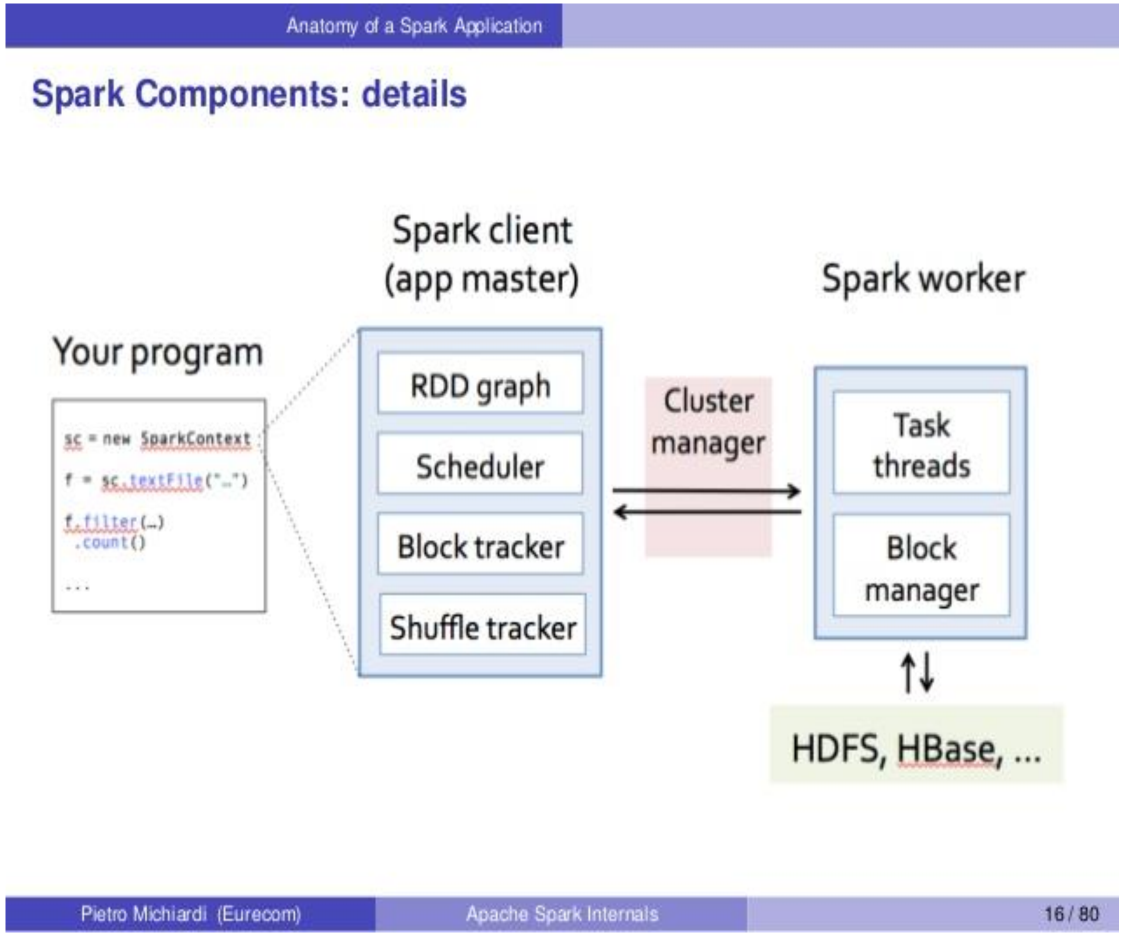
Sparks Architecture

Sparks uses a master slave architecture with one central coordinator and many distributer nodes

The central coordinator is called a driver

Distributed workers are called executors

A driver and executor together termed as Spark Application



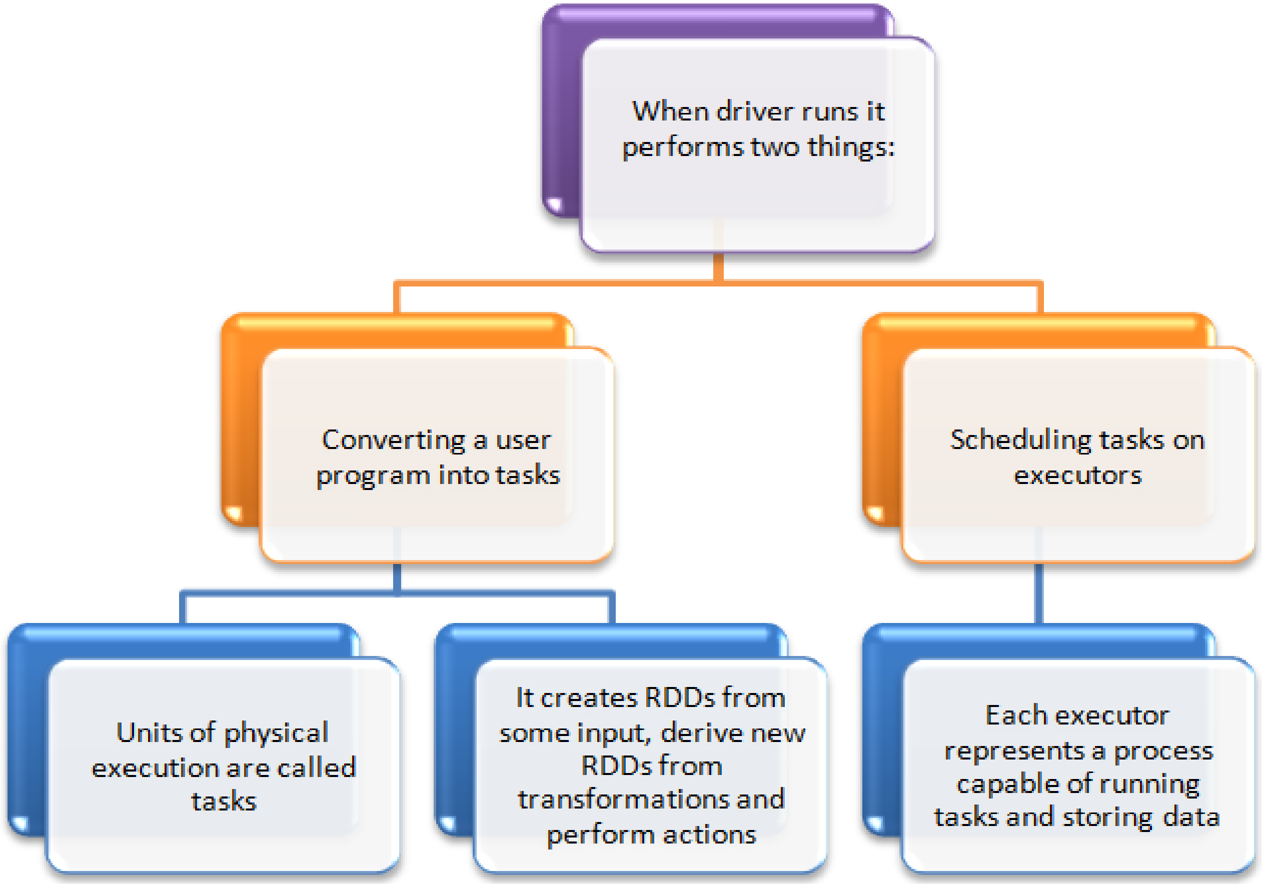
Sparks Architecture

The Driver

The driver is the process where main() method of your program runs

It is the process running the user code that creates a Sparks Context

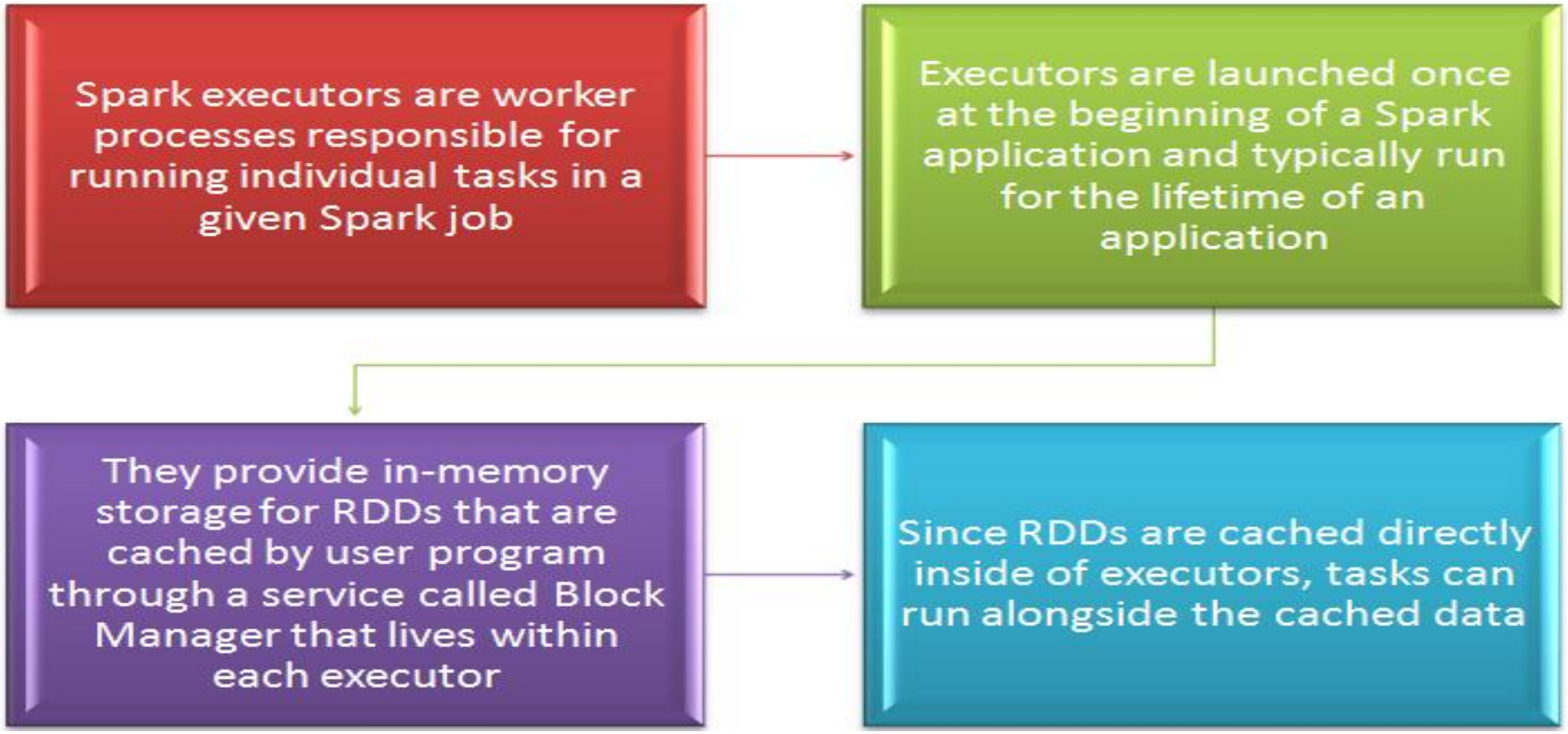
Spark Context creates RDDs and perform transformations and actions



When driver runs it performs two things:
Converting a user program into tasks
Scheduling tasks on executors
Units of physical execution are called tasks
It creates RDDs from some input, derive new RDDs from transformations and perform actions
Each executor represents a process capable of running tasks and storing data

The executors

Features



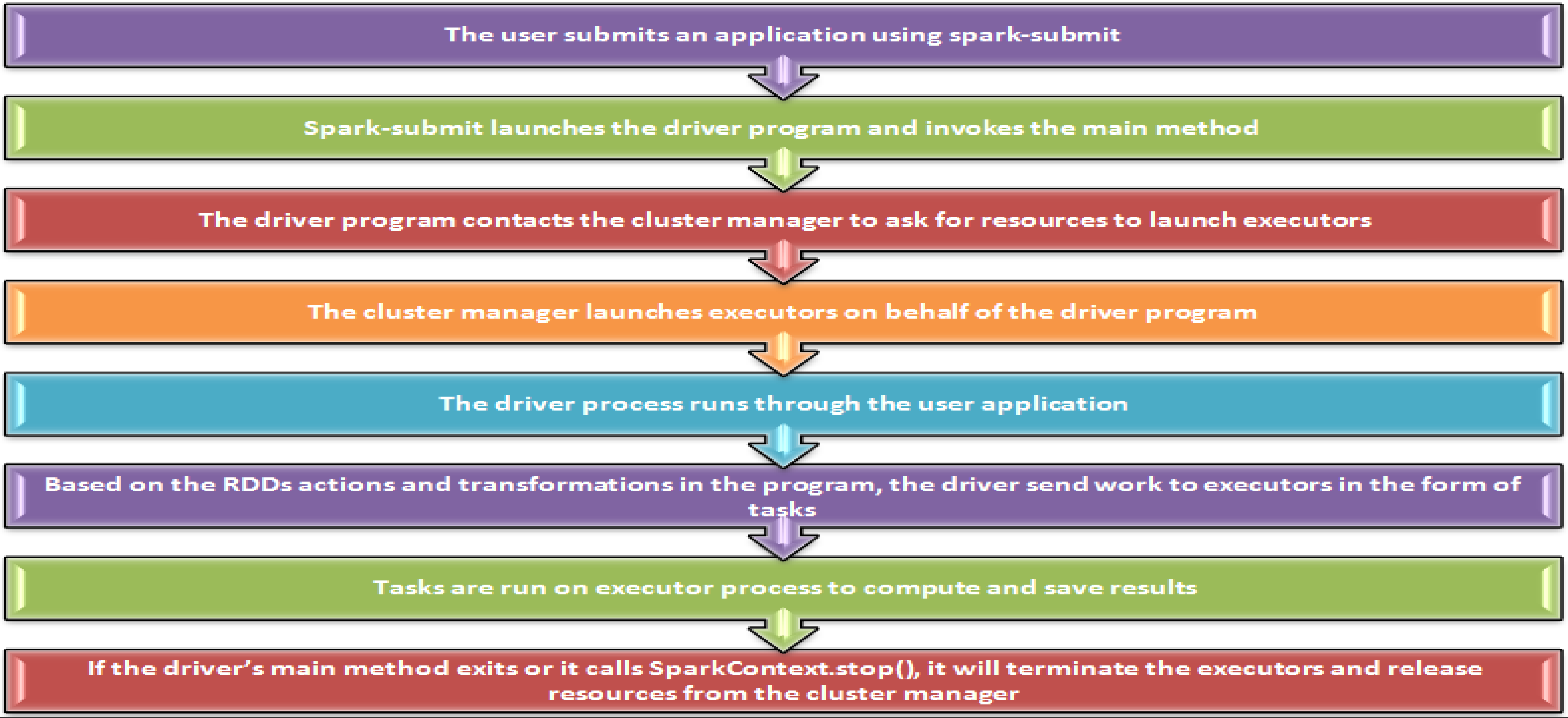
Spark executors are worker processes responsible for running individual tasks in a given Spark job

Executors are launched once at the beginning of a Spark application and typically run for the lifetime of an application

They provide in-memory storage for RDDs that are cached by user program through a service called Block Manager that lives within each executor

Since RDDs are cached directly inside of executors, tasks can run alongside the cached data

Launching a Program



The user submits an application using spark-submit

Spark-submit launches the driver program and invokes the main method

The driver program contacts the cluster manager to ask for resources to launch executors

The cluster manager launches executors on behalf of the driver program

The driver process runs through the user application

Based on the RDDs actions and transformations in the program, the driver send work to executors in the form of tasks

Tasks are run on executor process to compute and save results

If the driver's main method exits or it calls SparkContext.stop(), it will terminate the executors and release resources from the cluster manager

QUIZ

Q1. Why is Apache Sparks faster than Hadoop?

Q2. How can Apache Sparks use Data Pipelines to implement scheduling?

Q3. How can we use Sparks Streaming in real world?

Q4. What happens if one Executor fails to perform its job?

Q5. What is meant by Immutable? Can we delete a reference of immutable object?

Reading Material

Tutorial: https://www.youtube.com/watch?v=o8Jy7ii4Uks

Tutorial: https://www.youtube.com/watch?v=65aV15uDKgA

Tutorial: https://www.youtube.com/watch?v=mL5dQ_1gkiA

Book: Learning Spark by O’Reilly (https://www.pdf-archive.com/2016/04/21/learningspark-o-reilly-2015/preview/page/1/)

ANY QUESTIONS ?