Spark Introduction

Core Concepts

SPARK INTRODUCTION

Agenda

In this module, we are going to look at the following topics:

- ✓ What is Spark
- ✓ Why Spark?
- ✓ Spark Unified Stack
- ✓ Spark Architecture
- ✓ Spark APIs
- ✓ SparkSession
- ✓ DataFrames
- ✓ Transformations, Actions & Lazy Evaluation
- ✓ Spark UI

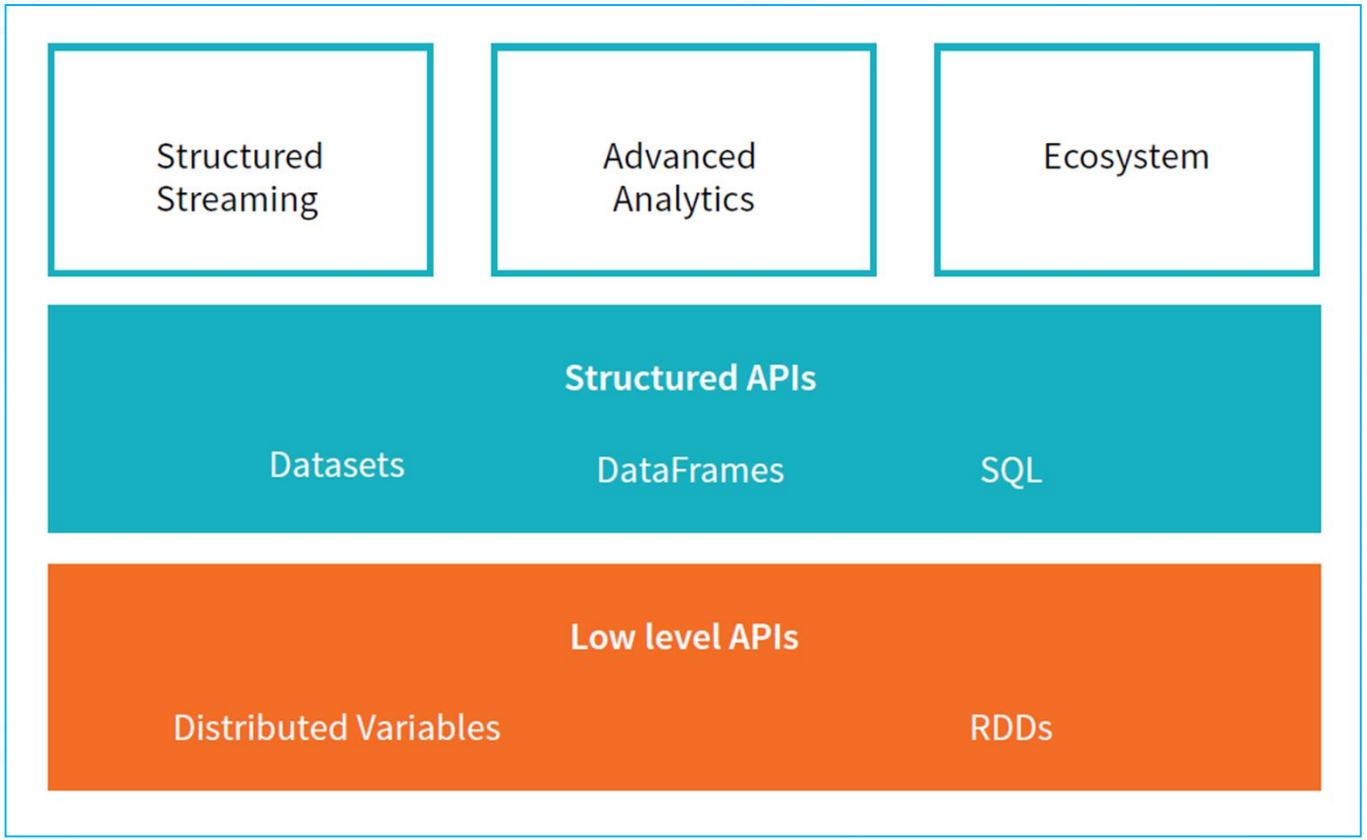
What is Spark ?

- Unified computing engine
- Set of libraries for big data

What is Spark ?

- Apache Spark is a ***unified computing engine***.
- Provides a set of libraries for parallel data processing on computer clusters.
- Spark supports multiple programming languages
  - Python
  - Java
  - Scala
  - R
- Includes libraries for diverse tasks
  - SQL / Structured Data Processing
  - Streaming
  - Machine Learning
  - Graph Parallel Computations

What Spark Offers?



Spark – Unified Platform

- Spark's key driving goal is to offer a **unified platform** for writing big data applications.
- Spark is designed to support a wide range of data analytics tasks over the same computing engine and with a consistent set of APIs.
- Before Spark, no open source systems tried to provide this type of unified engine for parallel data processing, meaning that users had to stitch together an application out of multiple APIs and systems. Thus, Spark quickly became the standard for this type of development.

Spark – Computing Engine

- Spark carefully limits its scope to a computing engine
- Spark only handles loading data from storage systems and performing computation on it, not permanent storage as the end itself.
- Spark can be used with a wide variety of persistent storage systems:
  - cloud storage systems such as Azure and Amazon S3
  - distributed file systems such as Apache Hadoop,
  - key-value stores such as Apache Cassandra,
  - message buses such as Apache Kafka.
- However, Spark neither stores data long-term itself, nor favors one of these.

Spark - Data Sources

Spark - Libraries

- Spark's main component is its libraries, which build on its design as a unified engine to provide a unified API for common data analysis tasks.

- Spark supports both standard libraries that ship with the engine, and a wide array of external libraries published as third-party packages by the open source communities.

Spark - Libraries

- In addition to it's core engine, Spark includes libraries for:
  - SQL & Structured data (Spark SQL)
  - Machine Learning (MLlib)
  - Stream Processing (Spark Streaming and Structured Streaming)
  - Graph Analytics (GraphX)
- Beyond these libraries, there hundreds of open source external libraries ranging from connectors for various storage systems to ML algorithms.
- An index of external libraries is available at spark-packages.org.

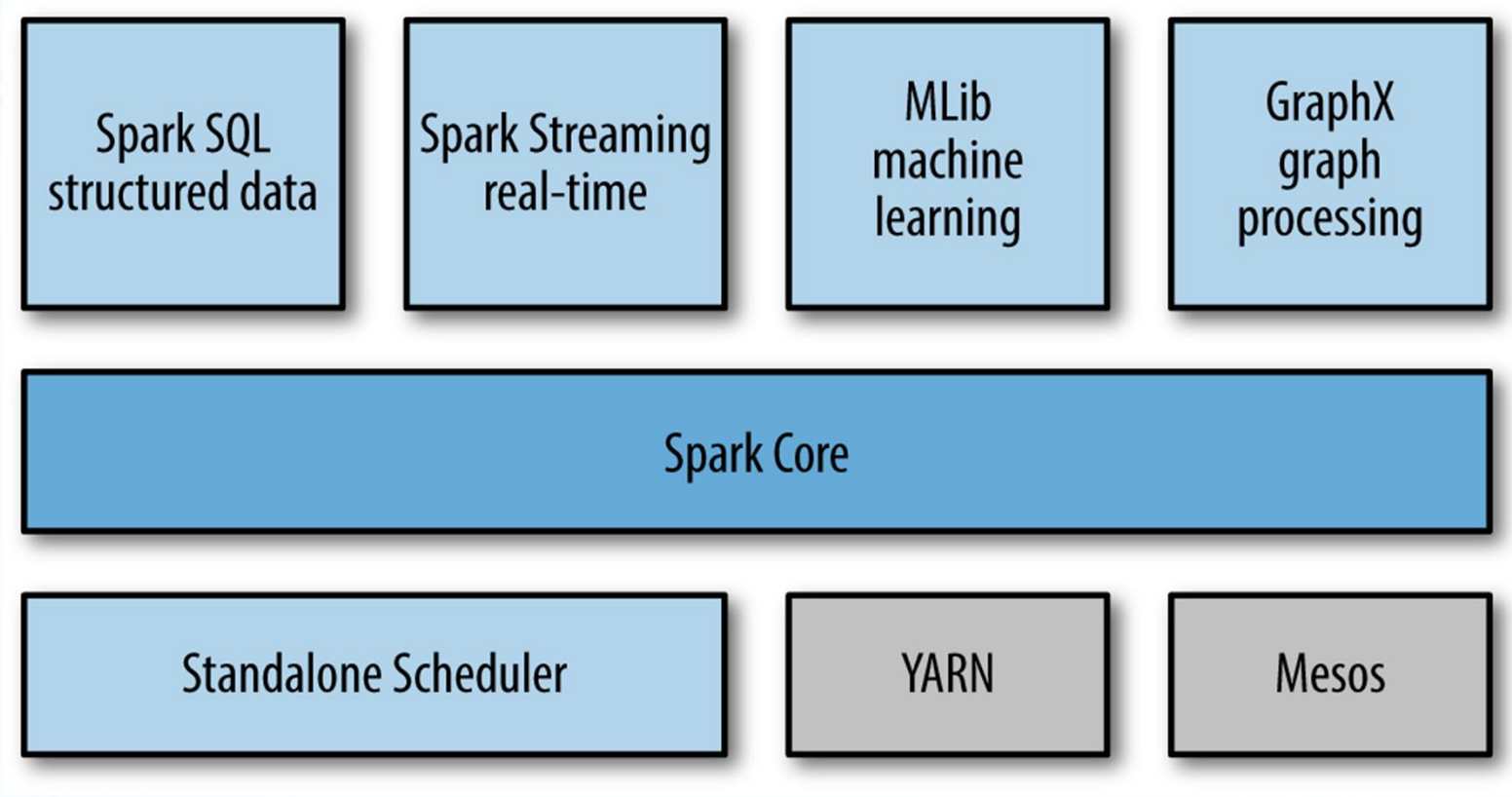
The Big Data Problem

- Why do we need a new engine and programming model for data analytics in the first place?
  - Processor speeds stopped growing since 2005 due to hard limits on hardware.
  - Technologies for storing and collecting data did not slow down appreciably in 2005, when processor speeds did.
  - The cost of storage drops by roughly 2x every 14 months.
  - Many of the technologies for collecting data (sensors, cameras, public datasets) continue to drop in cost and improve in resolution.

Why Spark?

- Collecting data is extremely inexpensive but processing it requires large, parallel computations often on clusters of machines.
- Software developed in the past 50 years cannot automatically scale up, and neither can the traditional programming models for data processing applications.
- This situation created the need for new programming models. Apache Spark was created to cater to this need.

Spark Ecosystem



Spark Unified Stack

Spark Core

- Contains all the basic functionality of Spark
- Contains APIs for RDD transformations and actions
- Contains APIs for task scheduling, memory management, fault recovery etc.
- Dependency: spark-core_xx (@ MvnRepository)

Spark Unified Stack

Spark SQL

- Allows querying data using SQL and HiveQL
- Supports various sources of data such as JSON, Hive, JDBC, NoSQL, Parquet, ORC, Delimited files like CSV etc.
- Allows programmers to intermix SQL with programmatic data manipulations supported by RDDs within a single application, thus combining SQL with complex analytics.

Spark Unified Stack

Spark Streaming

- Allows processing of real time streaming data
- Provides an API for manipulating data streams that closely match the Spark Core's RDD API

Spark Unified Stack

Spark MLlib

- Supports multiple types of machine learning algorithms including clustering, classification, regression and collaborative filtering.
- Has tools for model evaluation, feature extraction, feature transformation and data import.
- Provides some lower level ML primitives, including a generic gradient descent optimization algorithm.

Spark Unified Stack

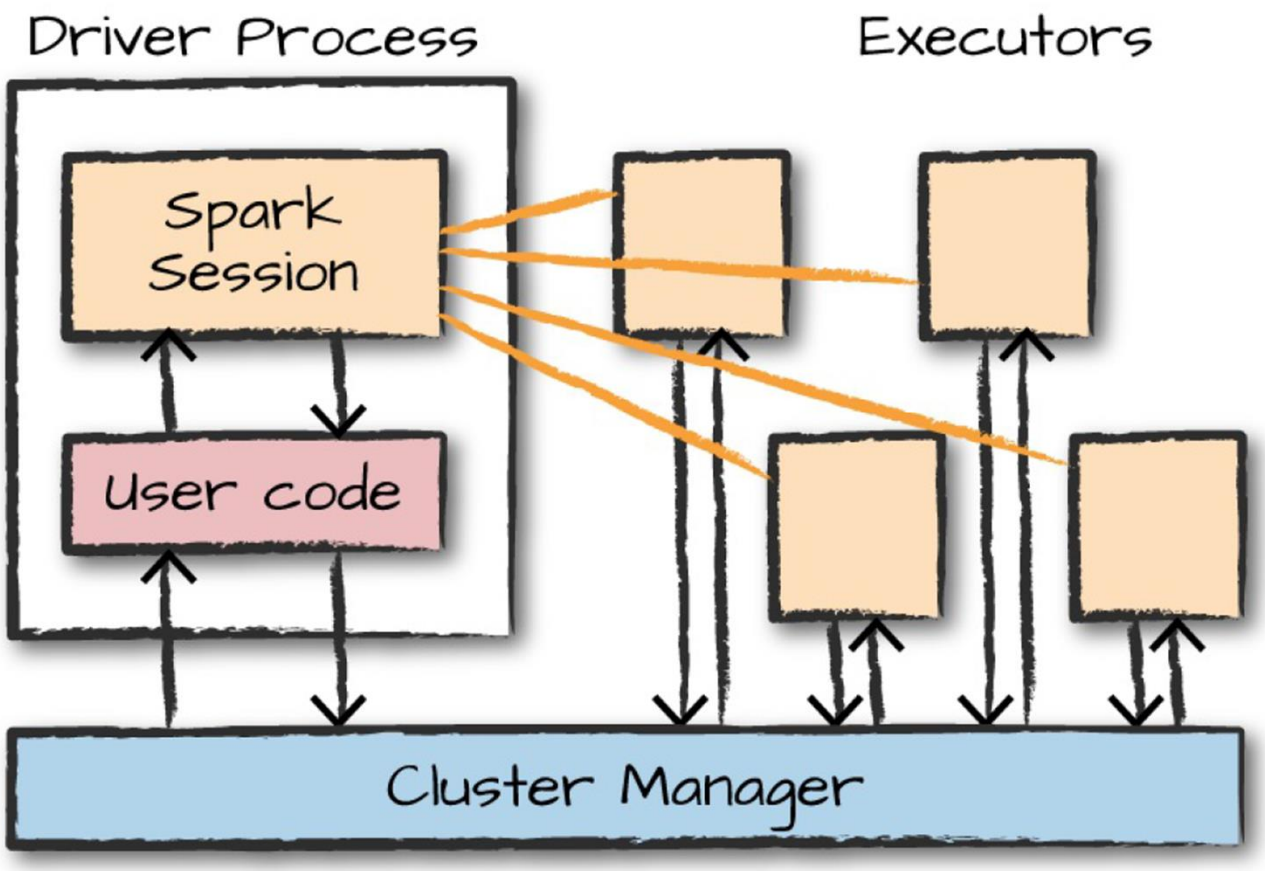
Spark GraphX

- Library for manipulating graphs and performing graph parallel computations.
- Provides various operators for manipulating graphs (ex: subgraphs, map vertices) and a library of common graph algorithms (ex; PageRank, Triangular Counting etc)

Spark Architecture

- Single machines do not have enough power and resources to perform computations on huge amounts of information
- A ***cluster*** pools the resources of many machines together allowing us to use all the cumulative resources as if they were one.
- Spark is a tool for managing and coordinating the execution of tasks on data across a cluster of computers.
- The cluster of machines that Spark will leverage to execute tasks will be managed by a ***cluster manager***.
- We then submit Spark Applications to these cluster managers which will grant resources to our application so that we can complete our work.

Spark Architecture



Spark Architecture - Driver

- Spark Applications consist of a ***driver*** process and a set of ***executor*** processes.

- The driver process runs your ***main()*** function, sits on a node in the cluster, and is responsible for three things:

  - maintaining information about the Spark Application
  - responding to a user's program or input
  - analyzing, distributing, and scheduling work across the executors.

- The driver process is the heart of a Spark Application and maintains all relevant information during the lifetime of the application.

Spark Architecture - Executors

- The executors are responsible for actually executing the work that the driver assigns them.
- This means, each executor is responsible for only two things:
  - executing code assigned to it by the driver
  - reporting the state of the computation back to the driver node.
- The user can specify how many executors should fall on each node through configurations.

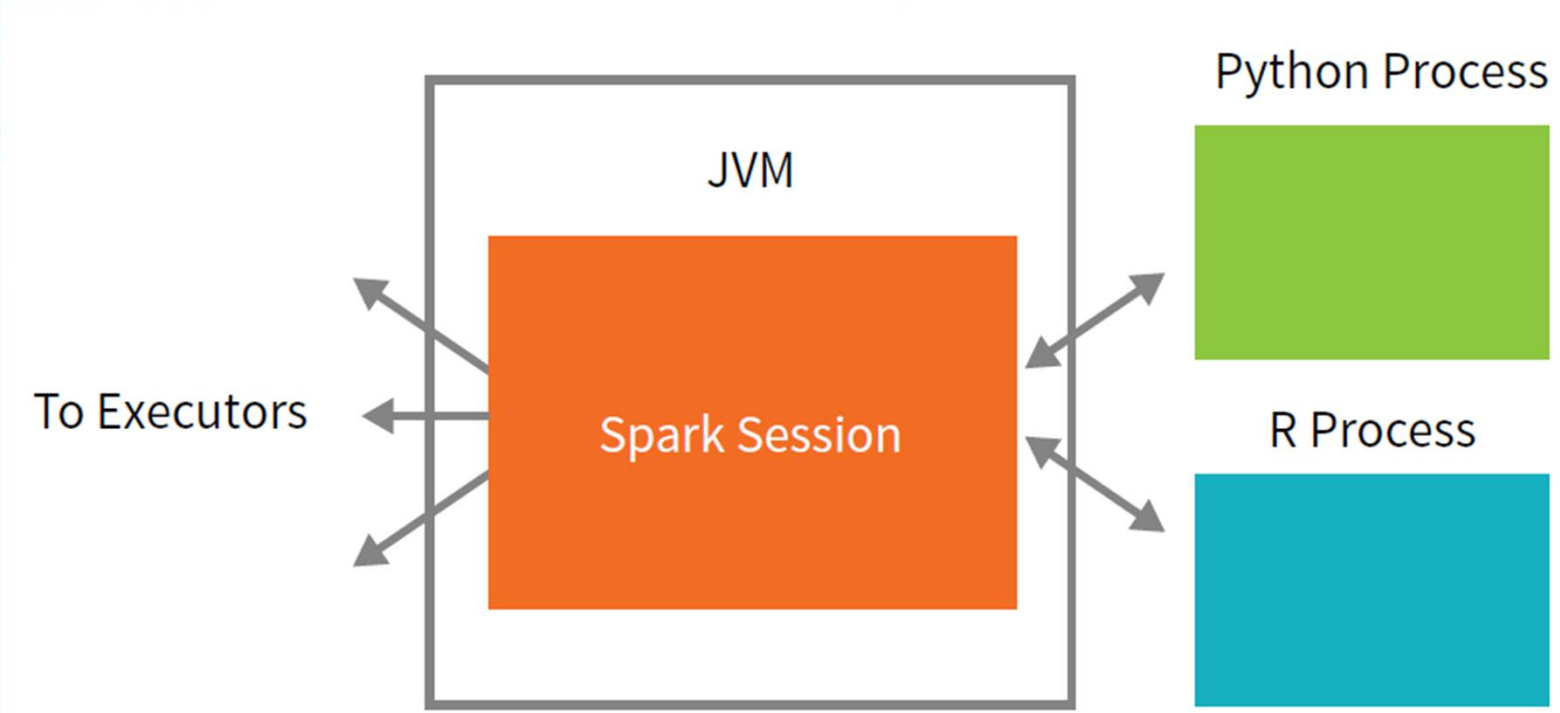
Spark Architecture – Cluster Manager

- The cluster manager controls physical machines and allocates resources to Spark Applications.
- This can be one of several core cluster managers:
  - Spark's standalone cluster manager
  - YARN
  - Mesos
  - Kubernetes
- This means that there can be multiple Spark Applications running on a cluster at the same time.

Spark's Language APIs

- **Scala** - Spark is written in Scala. It is Spark's "default" language.
- **Java** - You can write Spark code in Java.
- **Python** - Python supports nearly all constructs that Scala supports.
- **SQL** - Spark supports ANSI SQL 2003 standard. This makes it easy for analysts and non-programmers to leverage the bigdata powers of Spark.
- **R** - Spark has two commonly used R libraries, one as a part of Spark core (SparkR) and another as an R community driven package (sparklyr).

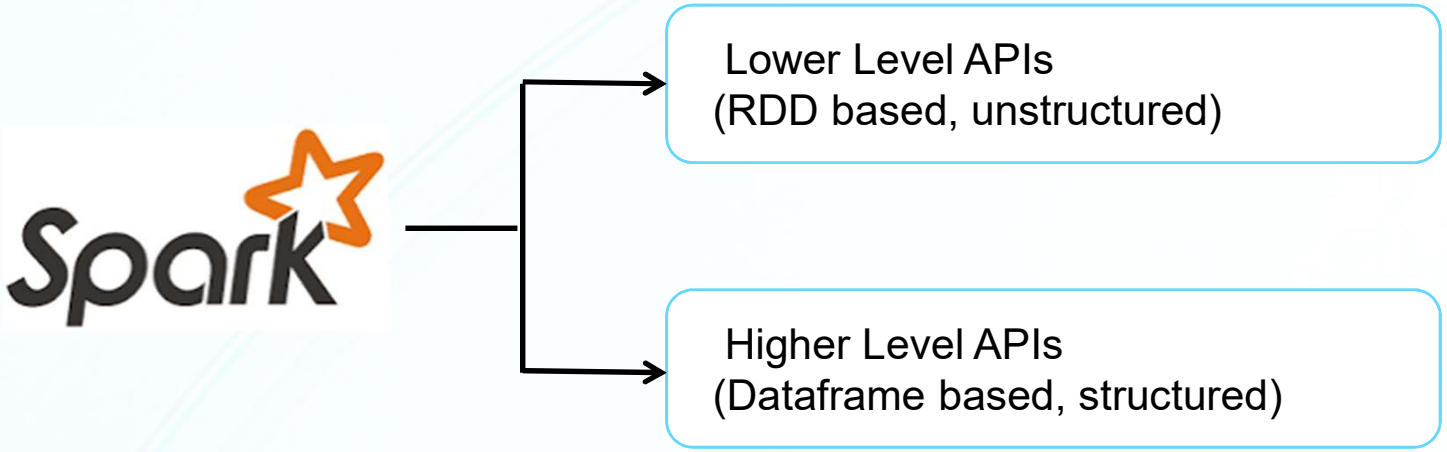
Spark's Language APIs



Spark's Language APIs

- Each language API will maintain the same core concepts that we described before.
- There is a SparkSession available to the user, the SparkSession will be the entrance point to running Spark code.
- When using Spark from a Python or R, the user never writes explicit JVM instructions, but instead writes Python and R code that Spark will translate into code that Spark can then run on the executor JVMs.

Spark's APIs


Spark
Lower Level APIs
(RDD based, unstructured)
Higher Level APIs
(Dataframe based, structured)

SparkSession

- We control our Spark Application through a driver process.
- This driver process manifests itself to the user as an object called the ***SparkSession***.
- The SparkSession instance is the way Spark executes user-defined manipulations across the cluster.
- There is a one to one correspondence between a SparkSession and a Spark Application.
- In Scala and Python the variable is available as ***spark*** when you start up the console.

SparkSession

```
// in Scala
val myRange = spark.range(1000).toDF("number")

# in Python
myRange = spark.range(1000).toDF("number")
```

- We created a DataFrame with one column containing 1000 rows with values from 0 to 999. This range of number represents a ***distributed collection***.
- When run on a cluster, each part of this range of numbers exists on a different executor.

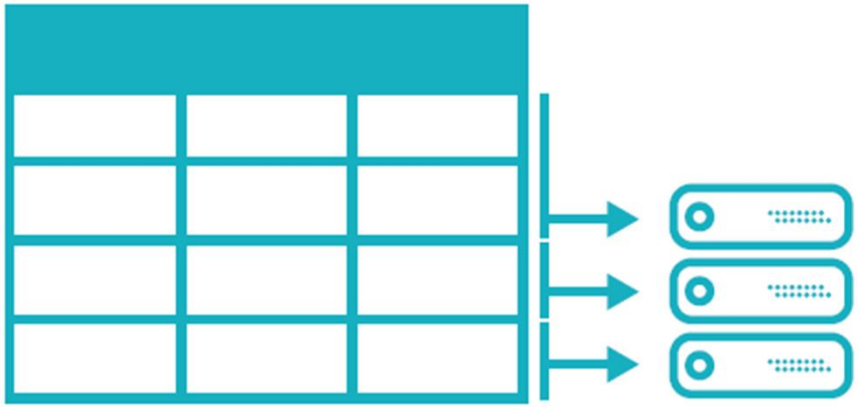
DataFrames

- A **DataFrame** is the most common Structured API and simply represents a table of data with rows and columns.
- The list that defines the columns and the types within those columns is called the ***schema***.
- You can think of a DataFrame as a spreadsheet with named columns.
- Spark DataFrame can span thousands of computers.

DataFrames

Spreadsheet on a single machine

Table or DataFrame partitioned across servers in a data center



Spark Abstractions

- Spark has several core abstractions:
  - DataSets
  - DataFrames
  - SQL Tables
  - Resilient Distributed Datasets (RDDs).
- These different abstractions all represent distributed collections of data.
- The easiest and most efficient are DataFrames, which are available in all languages.

Partitions

- To allow every executor to perform work in parallel, Spark breaks up the data into chunks called ***partitions***.
- A partition is a collection of rows that sit on one physical machine in your cluster.
- A DataFrame's partitions represent how the data is physically distributed across the cluster of machines during execution.
- If you have one partition, Spark will have a parallelism of only one, even if you have thousands of executors.
- If you have many partitions but only one executor, Spark will still have a parallelism of only one because there is only one computation resource.

Transformations

- In Spark, the core data structures are immutable meaning they cannot be changed once created.
- This might seem like a strange concept at first, if you cannot change it, how are you supposed to use it?
- In order to “change” a DataFrame you will have to instruct Spark how you would like to modify the DataFrame you have into the one that you want.
- These instructions are called ***transformations***.

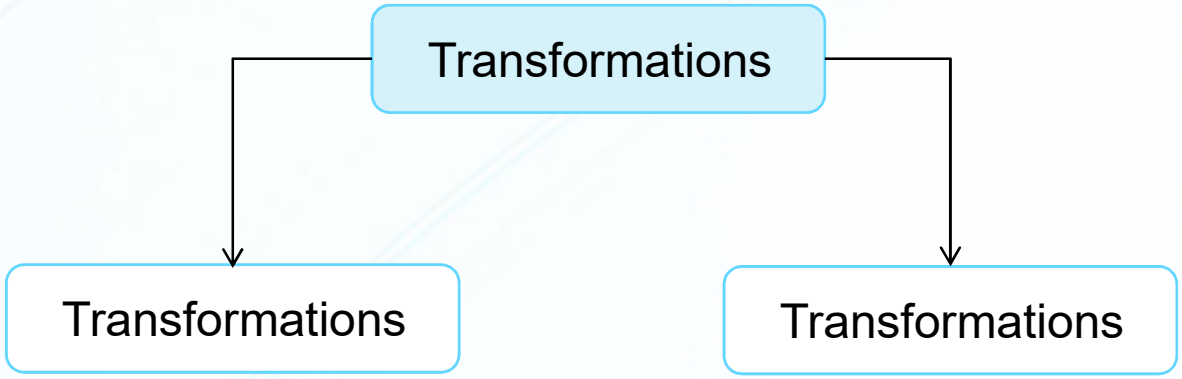
Transformations

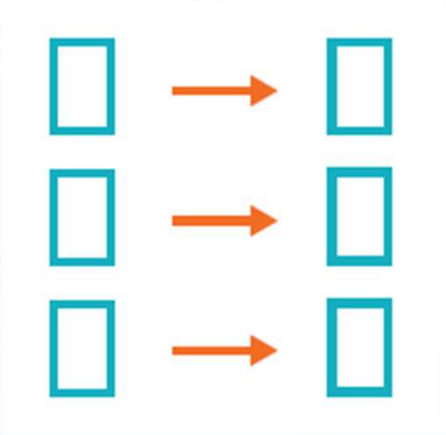
```
// in Scala
val divisBy2 = myRange.where("number % 2 = 0")

# in Python
divisBy2 = myRange.where("number % 2 = 0")
```

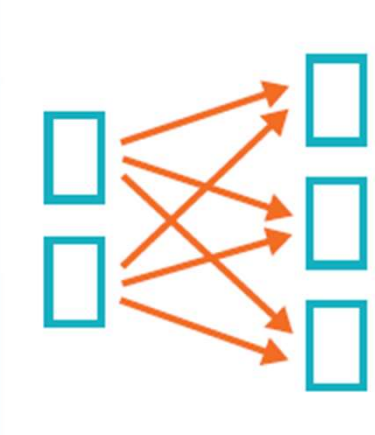
- Transformations are the core of how you will be expressing your business logic using Spark.
- Transformations are simply ways of specifying different series of data manipulation

Transformations





Narrow transformations
1 to 1



Wide transformations
1 to N (shuffles)

Lazy Evaluation

- ***Lazy evaluation*** means that Spark will wait until the very last moment to execute the graph of computation instructions.
- In Spark, instead of modifying the data immediately when we express some operation, we build up a plan of transformations that we would like to apply to our source data.
- Spark, by waiting until the last minute to execute the code, will compile this plan from your raw DataFrame transformations, to an efficient physical plan that will run as efficiently as possible across the cluster.
- This provides immense benefits to the end user because Spark can optimize the entire data flow from end to end.

Lazy Evaluation

- An example of this is something called “predicate pushdown” on DataFrames.
- If we build a large Spark job but specify a filter at the end that only requires us to fetch one row from our source data, the most efficient way to execute this is to access the single record that we need.
- Spark will actually optimize this for us by pushing the filter down automatically.

Actions

- Transformations allow us to build up our logical transformation plan. To trigger the computation, we run an ***action***.
- An action instructs Spark to compute a result from a series of transformations.
- Ex: `divsBy2.count()`

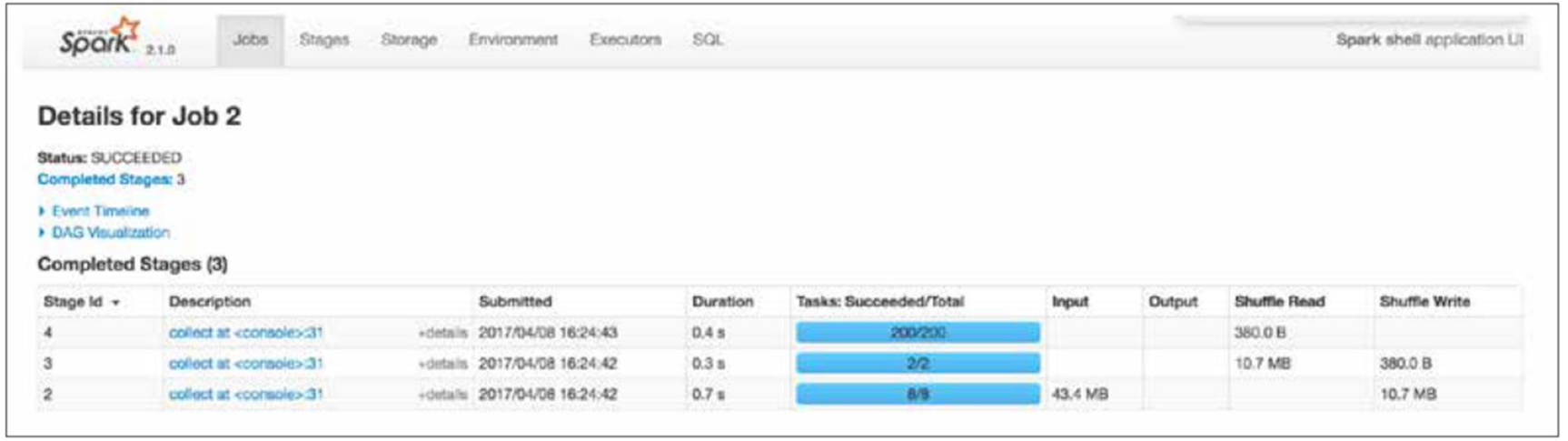
Actions

- There are three kinds of actions:
  - actions to view data in the console
  - actions to collect data to native objects in the respective language
  - actions to write to output data sources

Spark UI

- During Spark's execution of the previous code block, users can monitor the progress of their job through the Spark UI.
- The Spark UI is available on port 4040 of the driver node. If you are running in local mode this will just be the http://localhost:4040.
- The Spark UI maintains information on the state of our Spark jobs, environment, and cluster state.
- It's very useful, especially for tuning and debugging.

Spark UI


Spark 2.1.0
Jobs
Stages
Storage
Environment
Executors
SQL
Spark shell application UI
Details for Job 2
Status: SUCCEEDED
Completed Stages: 3
Event Timeline
DAG Visualization
Completed Stages (3)
Stage Id
Description
Submitted
Duration
Tasks: Succeeded/Total
Input
Output
Shuffle Read
Shuffle Write
4 collect at <console>:31 +details 2017/04/08 16:24:43 0.4 s 200/200 380.0 B
3 collect at <console>:31 +details 2017/04/08 16:24:42 0.3 s 2/2 10.7 MB 380.0 B
2 collect at <console>:31 +details 2017/04/08 16:24:42 0.7 s 8/8 43.4 MB 10.7 MB

Spark UI

Completed Jobs (3)

Job Id	Description	Submitted	Duration	Stages: Succeeded/Total	Tasks (for all stages): Succeeded/Total
2	collect at <console>:36	2018/09/12 06:18:23	0.3 s	2/2	4/4
1	collect at <console>:34	2018/09/12 06:17:37	51 ms	1/1	2/2
0	collect at <console>:32	2018/09/12 06:16:54	0.4 s	1/1	2/2

Completed Stages (2)

Stage Id	Description	Submitted	Duration	Tasks: Succeeded/Total	Input	Output	Shuffle Read	Shuffle Write
3	collect at <console>:36 +details	2018/09/12 06:18:23	84 ms	2/2			592.0 B	
2	map at <console>:33 +details	2018/09/12 06:18:23	0.1 s	2/2	1278.0 B			592.0 B

workspace - Wordcount/src/scala/WordCount.scala - Scala IDE

Spark 1.6.0 Jobs Stages Storage Environment Executors SQL

Environment

Runtime Information

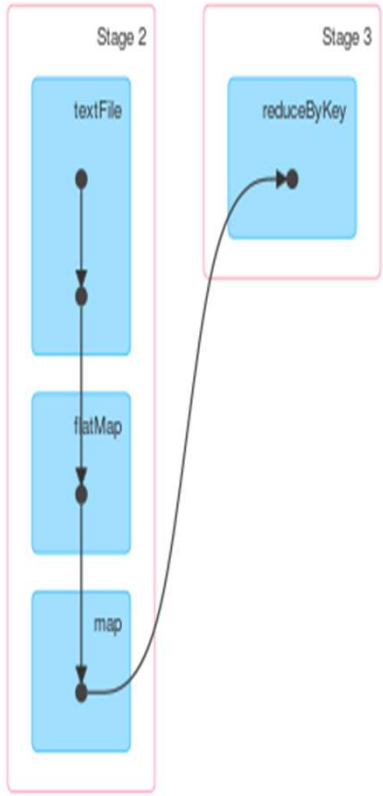
Name	Value
Java Home	/usr/java/jdk1.7.0_67-cloudera/jre
Java Version	1.8.0_181 (Oracle Corporation)
Scala Version	version 2.10.5

Spark Properties

Name	Value
spark.app.id	local-1536757947907

Status: SUCCEEDED
Completed Stages: 2

Event Timeline
DAG Visualization



Submitting Spark Applications to the Cluster

- The spark-submit script (in bin directory) is used to launch applications on a cluster.

```
./bin/spark-submit
      --class <main-class>
      --master <master-url>
      <application-jar> [application-arguments]
```

`spark-submit` command options

```
# Run application locally on 8 cores
./bin/spark-submit \
  --class org.apache.spark.examples.SparkPi \
  --master local[8] \
  /path/to/examples.jar \
  100

# Run on a Spark standalone cluster in client deploy mode
./bin/spark-submit \
  --class org.apache.spark.examples.SparkPi \
  --master spark://207.184.161.138:7077 \
  --executor-memory 20G \
  --total-executor-cores 100 \
  /path/to/examples.jar \
  1000

# Run on a Spark standalone cluster in cluster deploy mode with supervise
./bin/spark-submit \
  --class org.apache.spark.examples.SparkPi \
  --master spark://207.184.161.138:7077 \
  --deploy-mode cluster \
  --supervise \
  --executor-memory 20G \
  --total-executor-cores 100 \
  /path/to/examples.jar \
  1000
```

THANK YOU