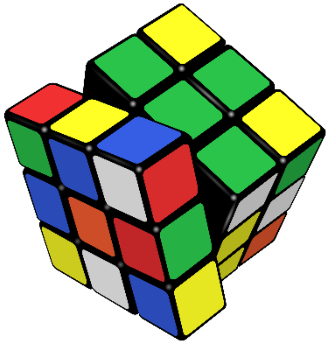


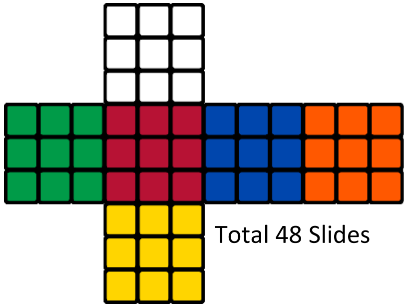
Big Data Engineering For Analytics

Introducing Apache Spark Framework

Suria R Asai

(suria@nus.edu.sg)

NUS-ISS



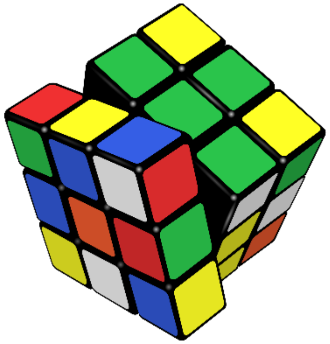
Total 48 Slides

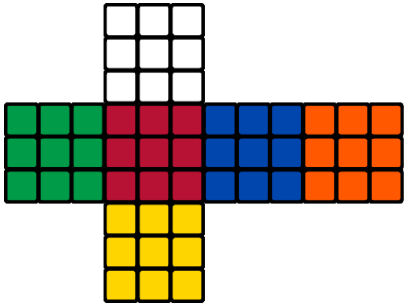
Learning Objectives

- Understand Apache Spark Architecture
- Learn Apache Spark Ecosystem Components
- Learn Various Types of Spark Cluster Managers

Agenda

- Apache Spark Architecture
- Unified Solution Stack
- Apache Spark Shell and Session
- Spark Examples
- Summary





Big Data Engineering For Analytics

Apache Spark Architecture

Lightning-fast cluster computing

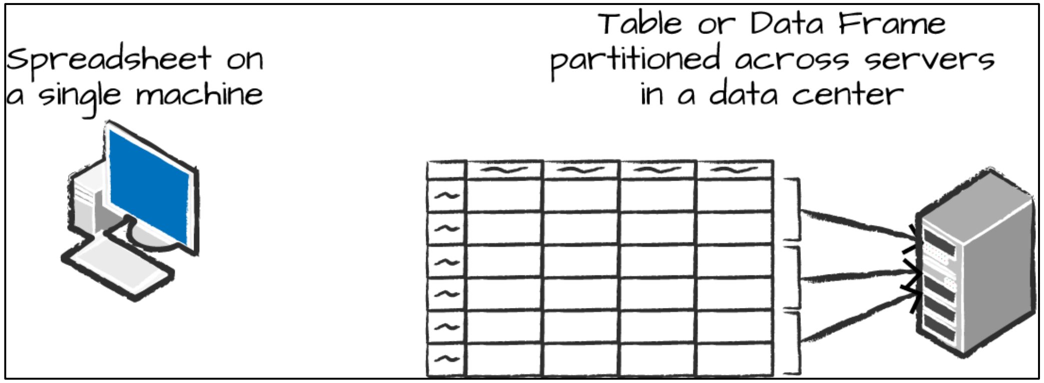
Big Data Engineering For Analytics

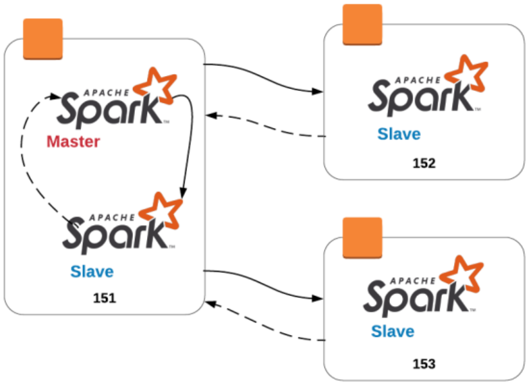
- **Apache Spark™** is a fast and general purpose engine for large-scale data processing.
  - **Speed**
    - Run programs up to 100x faster than Hadoop MapReduce in memory, or 10x faster on disk.
  - Ease of Use, Modularity and Extensibility
    - Offers high-level APIs to users, such as **Java, Scala, Python, R**.
  - Generality
    - Combine **SQL**, **streaming**, and complex **analytics**.
  - **Runs** Everywhere
    - Spark runs on Hadoop, Mesos, standalone, or in the cloud. It can access diverse data sources including HDFS, Cassandra, HBase, and S3.

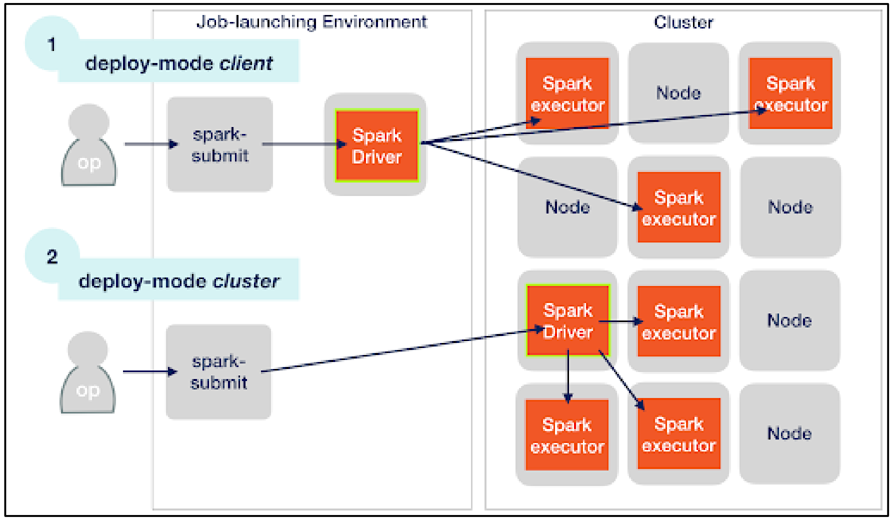
Design Goals of Apache Spark

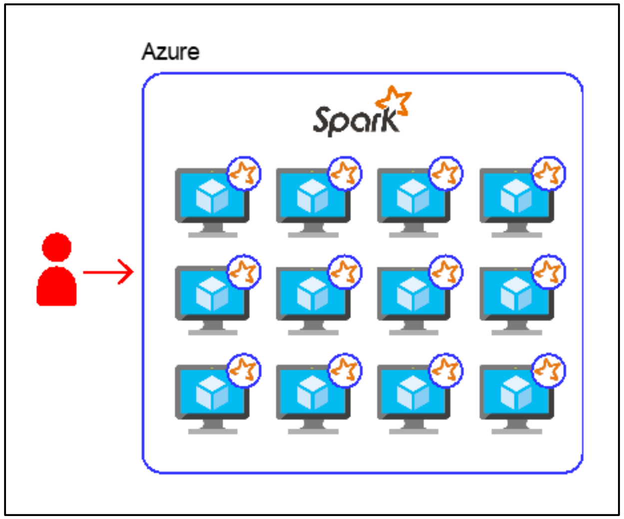
- Distributed Computation Engine
  - Based on JVM and Functional Programming;
  - Supports Immutability/ Streams/ Transformations;
  - Concise Syntax.
- Goals:
  - cluster computing platform designed to be **fast and general-purpose**.
  - support **more types of computations**; designed to cover workloads including batch applications, iterative algorithms, interactive queries, and streaming.
  - process large datasets using '**In-Memory**' methods; makes it easy and inexpensive to combine different processing types to create data analysis pipelines.
  - designed to be **highly accessible**, offering simple APIs in Python, Java, Scala, and SQL, and rich built-in libraries to integrate with other data tools.

Distributed Vs single-machine analysis

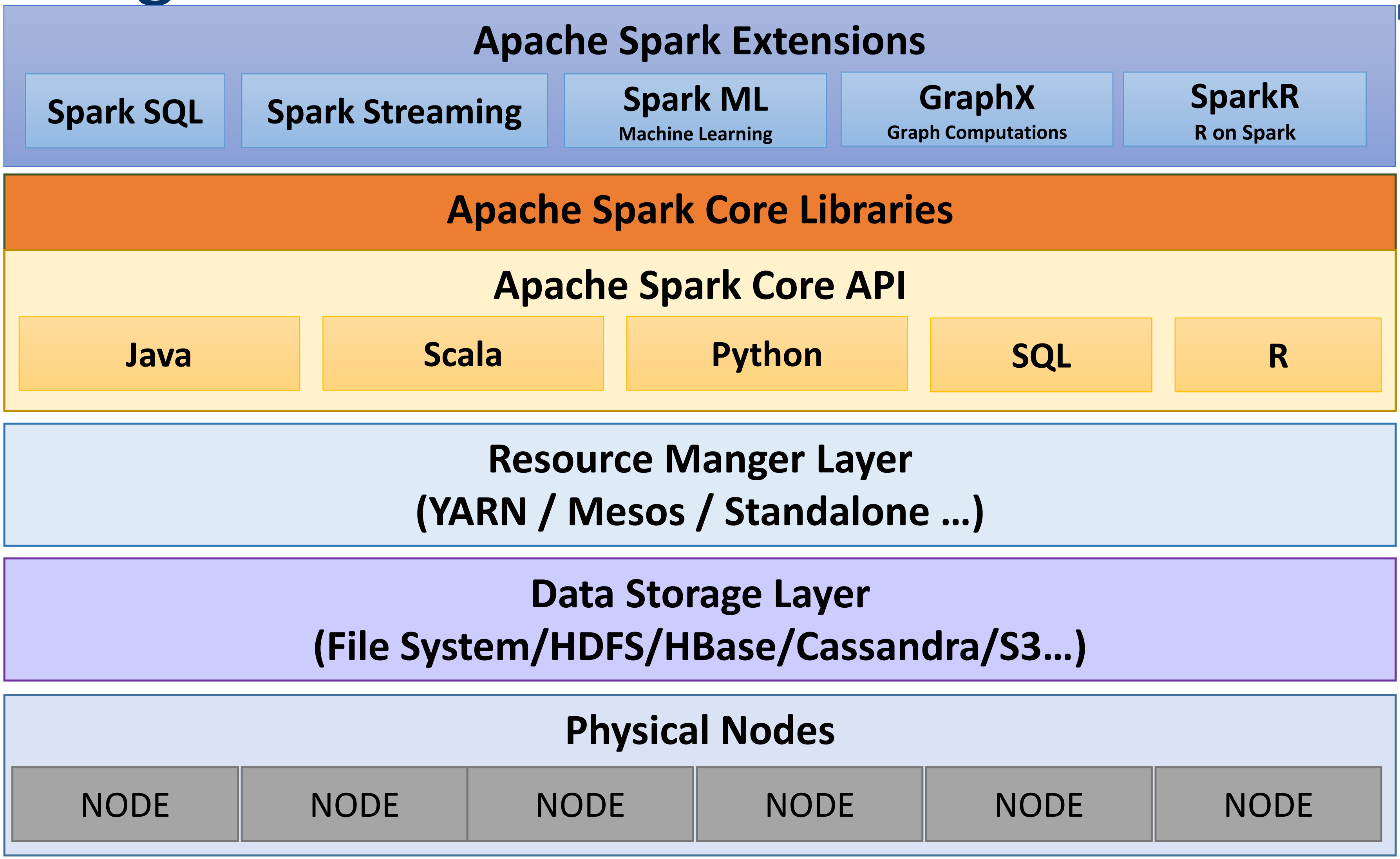


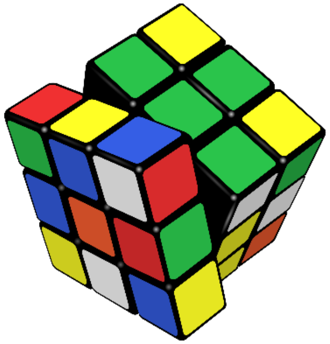


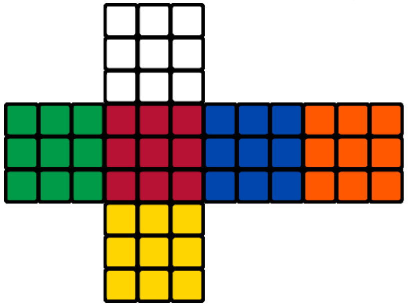




High-level architecture

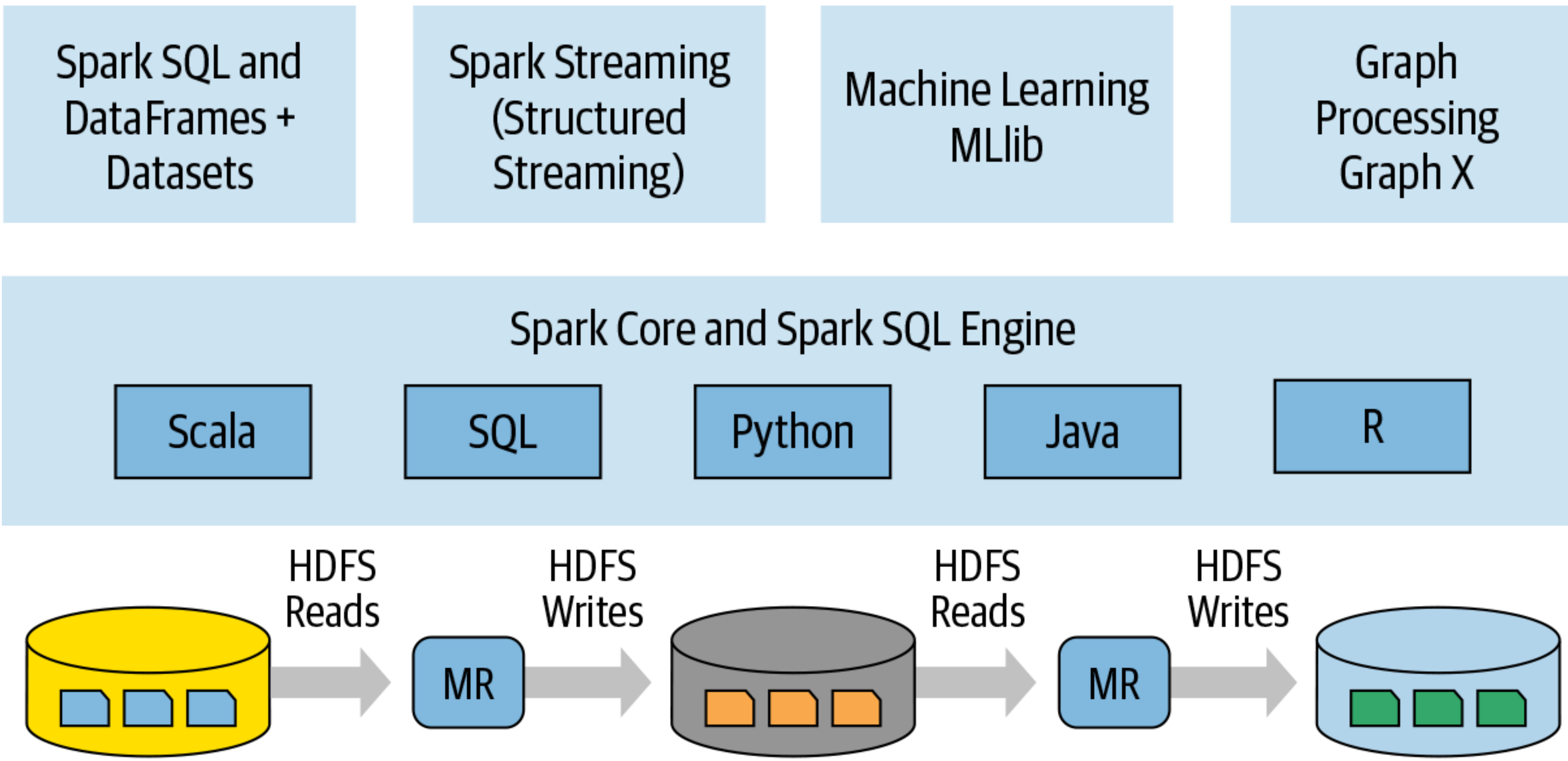






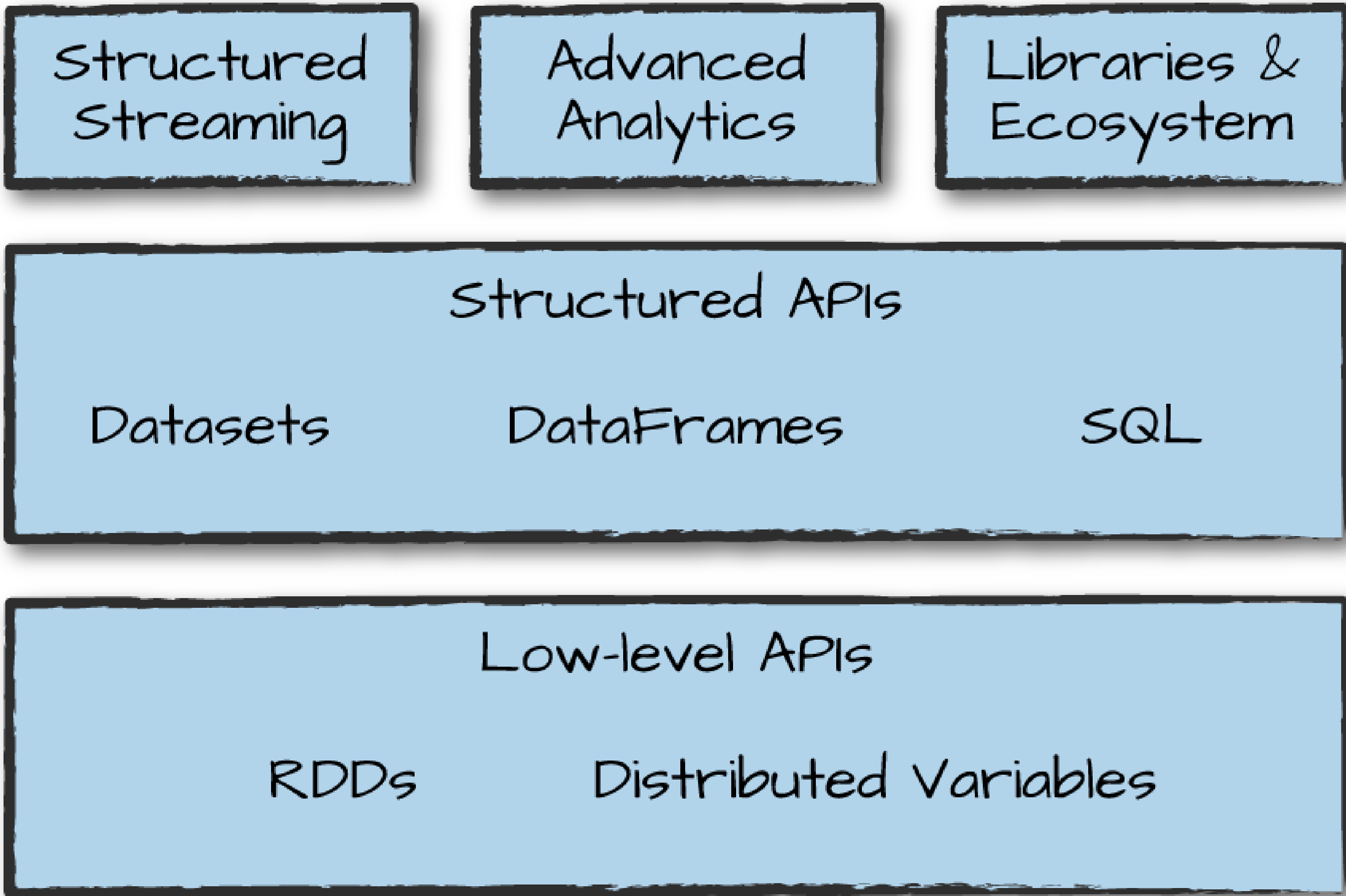
Unified Solution Stack

Unified Solution Stack



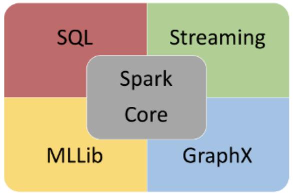
Reference: Learning Spark, 2nd Edition

Spark End User Libraries



Spark Core

- Provides services such as **memory** pool, **scheduling** of cluster, massively **parallel** processing constructs, basic IO constructs and abstractions
- Comprises of basic **components** such as **RDD** (Resilient Distributed Data Sets), Dataframes, Datasets and Graph Frames.
- The core APIs available perform **operations** on ETL basic abstractions
- Shared or **distributed variables** can be created and managed. Example of such are broadcast variables and accumulators



RDD (Resilient Distributed Dataset)

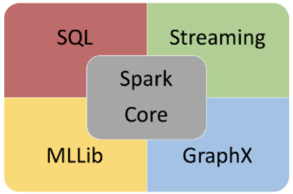
- RDD (Resilient Distributed Dataset)
  - Resilient – if data in memory is lost, it can be recreated
  - Distributed – processed across the cluster
  - Dataset – collection of rows and columns - data can come from a file or from any other source
  - RDDs are the fundamental unit of data in Spark
  - Most Spark programming consists of performing operations on RDDs
- Contained in an RDD
  - Set of dependencies on parent RDDs – lineage
  - Set of partitions – Atomic pieces of a dataset
- Ways to create an RDD
  - From a file or set of files
  - From data in memory
  - From another RDD
  - From a DataFrame or DataSet
  - From Local Collection
  - From DataSources

RDD, DataFrame, DataSet

- Spark Release
  - RDD :The RDD APIs have been on Spark since the 1.0 release.
  - DataFrames :Spark introduced DataFrames in Spark 1.3 release.
  - DataSet :Spark introduced Dataset in Spark 1.6 release.
- Data Formats
  - RDD: It can easily and efficiently process data which is structured as well as unstructured
  - DataFrame: It works only on structured and semi-structured data.
  - DataSet: It also efficiently processes structured and unstructured data. It represents data in the form of JVM objects of row or a collection of row object. Which is represented in tabular forms through encoders.
- We can move RDD to DataFrame (if RDD is in tabular format) by toDF() method or we can do the reverse by the .rdd method.
- RDD offers low-level functionality and control. The DataFrame and Dataset allow custom view and structure. It offers high-level domain-specific operations, saves space, and executes at high speed.

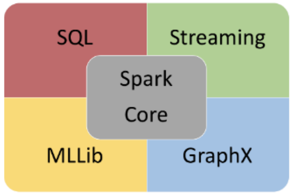
Spark SQL

- Work with structured and semi-structured data such as *Hive tables, MySQL tables, Parquet files, AVRO files, JSON files, CSV files, and more*.
- Spark SQL is one of the most technically involved components of Apache Spark.
  - ➢It powers both SQL queries and the DataFrame API.
  - ➢At the core of Spark SQL is the **Catalyst optimizer**, which leverages advanced programming language features (e.g. Scala's pattern matching and quasiquotes) in a novel way to build an extensible query optimizer.
- Catalyst is based on functional programming constructs in Scala and designed with these key two purposes:
  - ➢Easily add new optimization techniques and features to Spark SQL
  - ➢Enable external developers to extend the optimizer
    - (e.g. adding data source specific rules, support for new data types, etc.)



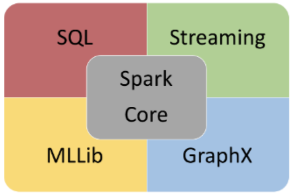
Spark Machine Learning

- Spark MLlib and ML are the Spark's packages to work with machine-learning algorithms. They provide the following:
  - ➢Inbuilt machine-learning algorithms such as Classification, Regression, Clustering, and more
  - ➢Features such as pipelining, vector creation, and more
- ML algorithms include:
  - ➢Classification: logistic regression and naive Bayes.
  - ➢Regression: generalized linear regression, survival regression.
  - ➢Decision trees, random forests, and gradient-boosted trees.
  - ➢Recommendation: alternating least squares (ALS)
  - ➢Clustering: K-means, Gaussian mixtures (GMMs)
  - ➢Frequent item sets, association rules, and sequential pattern mining
- ML workflow utilities include:
  - ➢Feature transformations: standardization, normalization, and hashing.
  - ➢ML Pipeline construction
  - ➢Model evaluation and hyper-parameter tuning
  - ➢ML persistence: saving and loading models and pipelines.



Spark Streaming

- Spark Streaming is a Spark component that enables processing of live streams of data.
- Stream processing involves:
  - ➢Input and output operations, transformations, persistence, and check pointing.
- Supports different types of stream processing:
  - ➢Both batch and window stream configurations
  - ➢Stream check pointing and processing using additional tools such as Kafka and Flume.
- Three Ways of Stream Processing
  - ➢Spark module functionality (for example, SQL, MLlib, and GraphX)
  - ➢External Systems such as Kinesis or ZeroMQ
  - ➢Create custom receivers for user-defined data sources

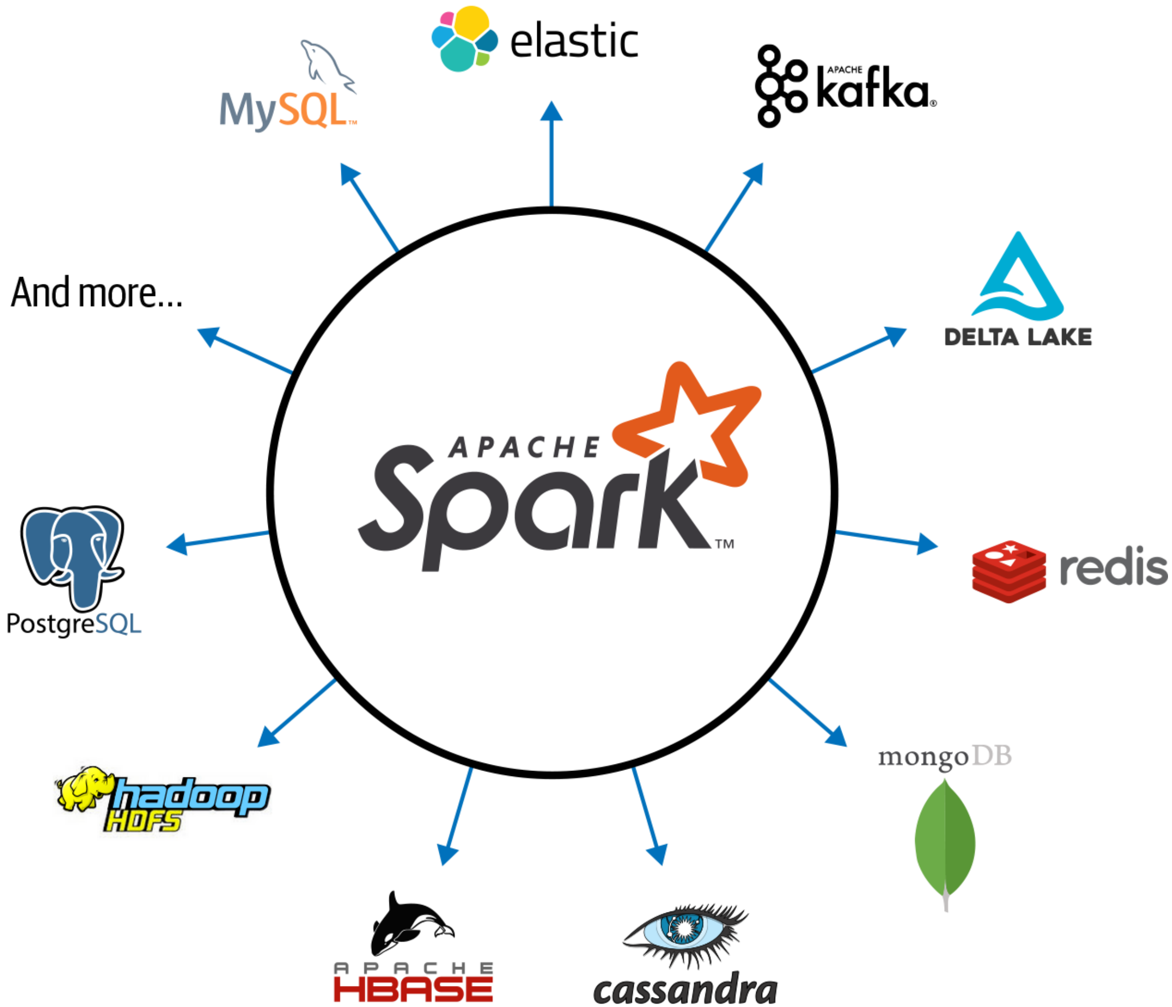


SparkR (R on Spark)

- SparkR is an R package that provides a light-weight frontend to use Apache Spark from R.
- The main idea behind SparkR was to explore different techniques to integrate the usability of R with the scalability of Spark.
- SparkR provides a distributed data frame implementation that supports operations like selection, filtering, aggregation etc. but on large datasets. SparkR also supports distributed machine learning using MLlib.
- There are various benefits of SparkR:
  - ➢ Data Sources API
  - ➢ Scalability to many cores and machines

https://spark.apache.org/docs/latest/sparkr.html

Apache Spark's ecosystem of connectors



Reference: Learning Spark, 2nd Edition

Big Data Engineering For Analytics

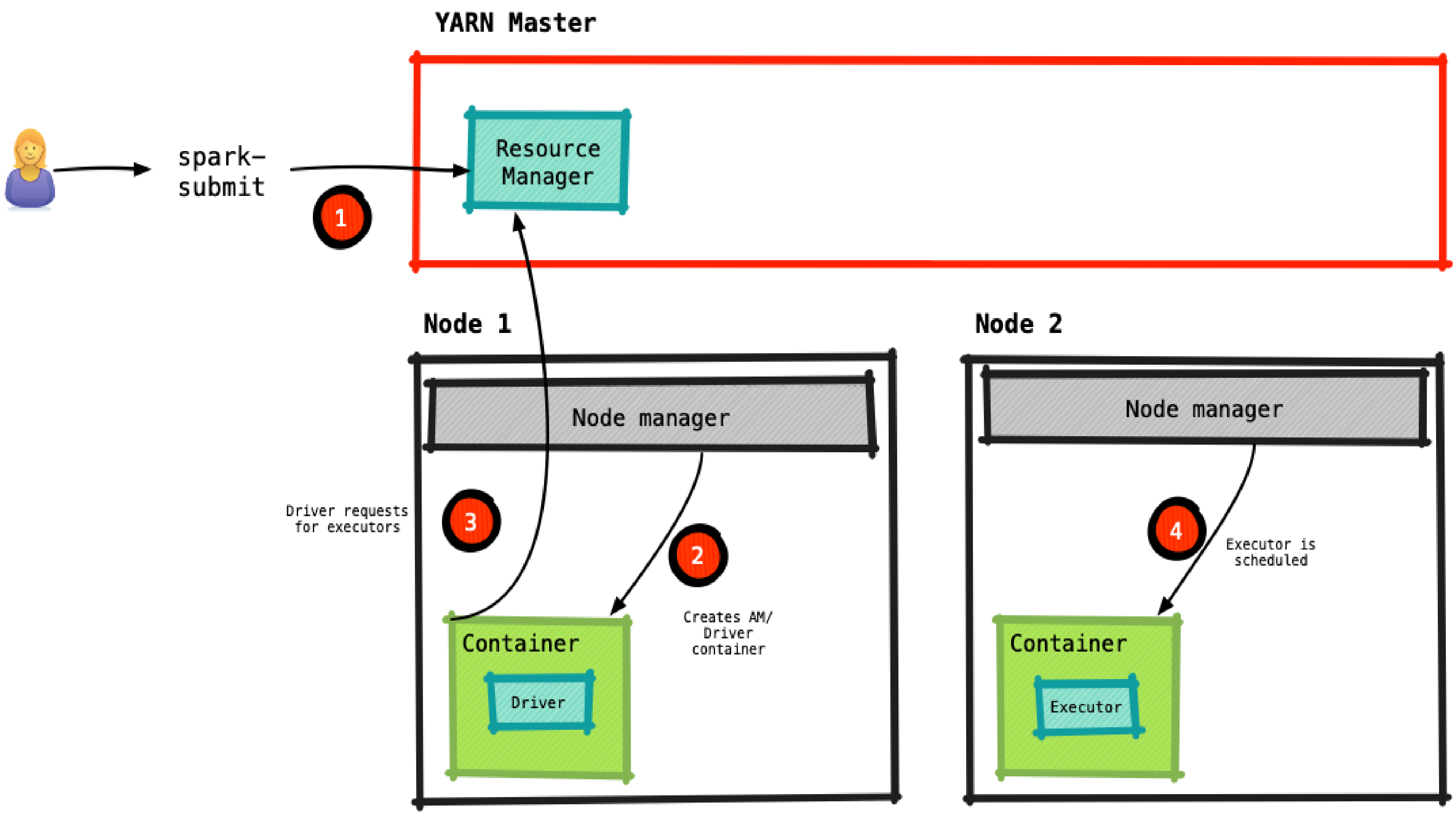
Apache Spark Shell and Spark Session

Spark Deployment Modes

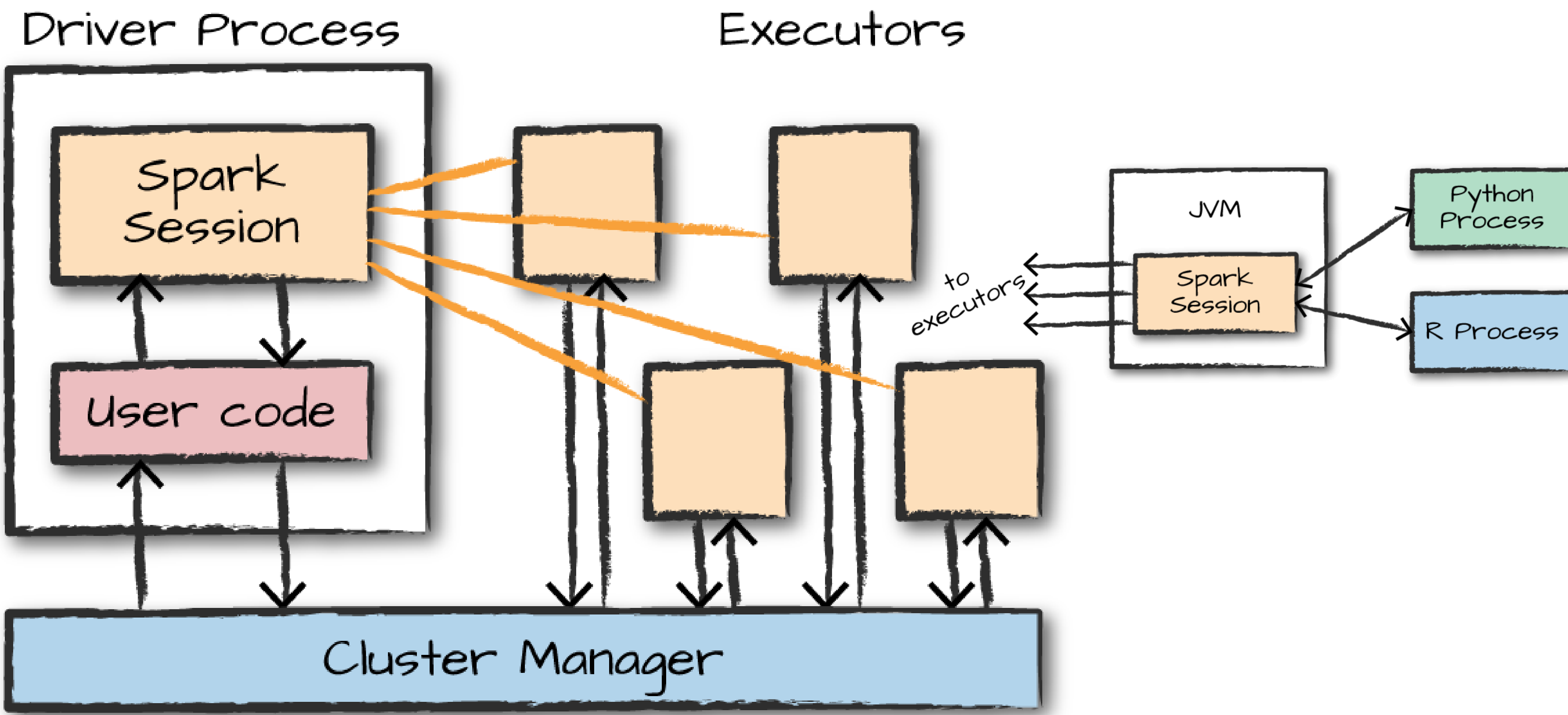
Big Data Engineering For Analytics

Mode	Spark Driver	Spark Executor	Cluster Manager
Local	JVM, Single Node	JVM	Runs on same host
Standalone	Any node in cluster	Each node launches JVM in cluster	Any host on cluster
YARN Client	Client	YARN NodeManager’s container	YARN Resource Manager
YARN Cluster	YARN App Master	YARN NodeManager’s container	YARN Resource Manager
Kubernetes	Kubernetes pod	Each worker runs within its own pod	Kubernetes Master

Spark on YARN (Cluster Mode)



The Spark Application



Spark Shell

- The Spark Shell provides interactive data exploration
- Writing Spark applications without the shell will be covered later

Python Shell: **`pyspark`**

```
$ pyspark

Welcome to
      ____              __
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ `/ __/  '_/
   /__ / .__/\_,_/_/ /_/\_\   version 1.3.0
      /_/

Using Python version 2.7.8 (default, Aug 27
2015 05:23:36)
SparkContext available as sc, HiveContext
available as sqlCtx.
>>>
```

Scala Shell: **`spark2-shell`**

```
$ spark2-shell

Welcome to
      ____              __
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ `/ __/  '_/
   /___/ .__/\_,_/_/ /_/\_\   version 1.3.0
      /_/

Using Scala version 2.10.4 (Java HotSpot(TM)
64-Bit Server VM, Java 1.7.0_67)
Created spark context..
Spark context available as sc.
SQL context available as sqlContext.

scala>
```

The Spark Context

Big Data Engineering For Analytics

- Every Spark application requires a Spark Context
  - ➢ The main entry point to the Spark API
- Prior to Spark 2.0, SparkContext for the core api, SQLContext for the spark-sql api, StreamingContext for the Dstream api
- With Spark 2.0, SparkSession is the single entry point
- Spark shell
  - ➢ automatically creates a SaprkContext as sc variable
  - ➢ Spark session available as ‘spark’

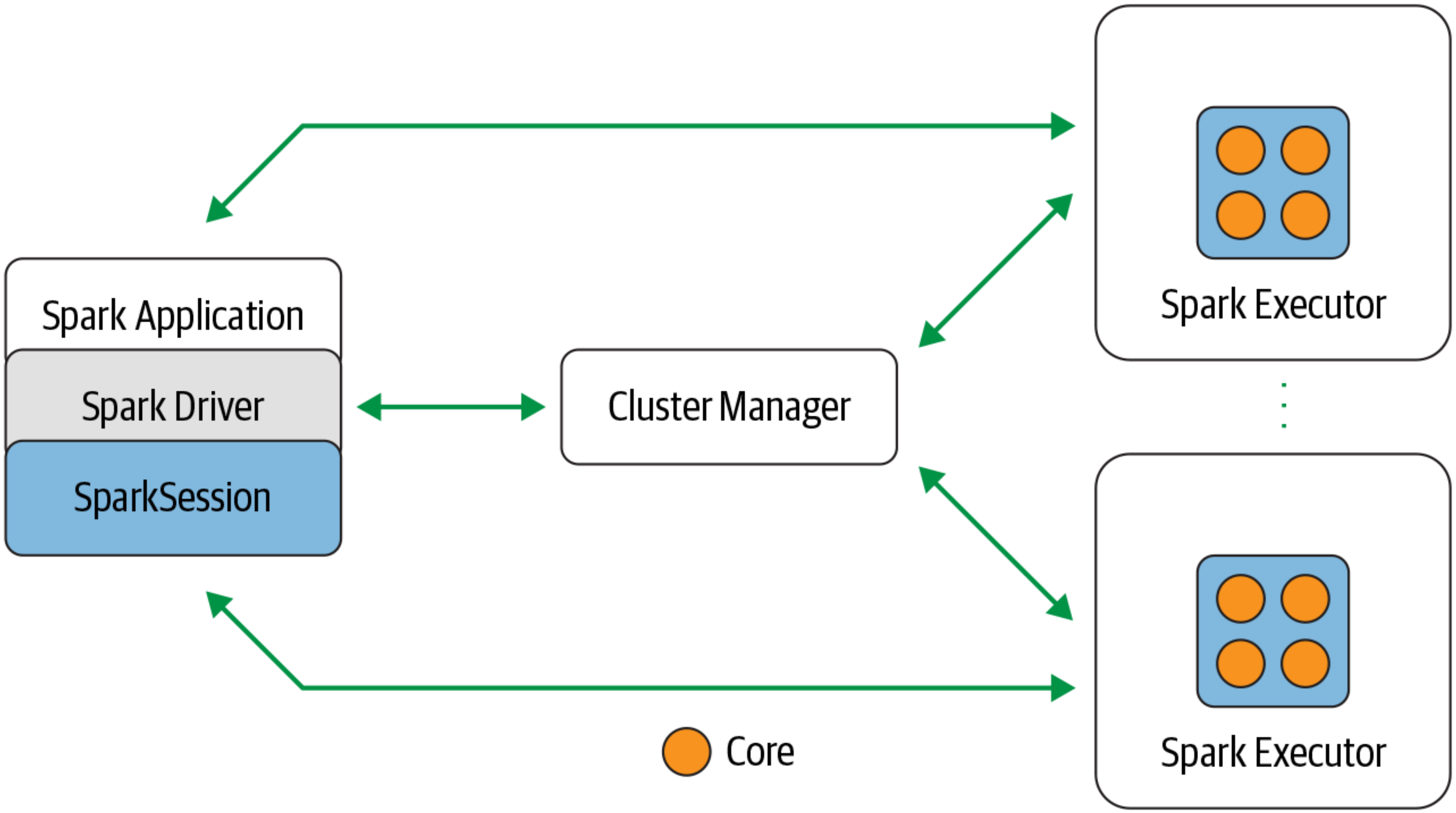
Select Command Prompt - spark-shell

```
C:\>spark-shell
20/08/27 10:43:37 WARN NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java cl
asses where applicable
Using Spark's default log4j profile: org/apache/spark/log4j-defaults.properties
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLevel(newLevel).
Spark context Web UI available at http://host.docker.internal:4040
Spark context available as 'sc' (master = local[*], app id = local-1598496236097).
Spark session available as 'spark'.
Welcome to
      ____              __
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ `/ __/  '_/
   /___/ .__/\_,_/_/ /_/\_\   version 2.4.5
      /_/

Using Scala version 2.11.12 (Java HotSpot(TM) 64-Bit Server VM, Java 1.8.0_191)
Type in expressions to have them evaluated.
Type :help for more information.

scala>
```

Spark Components



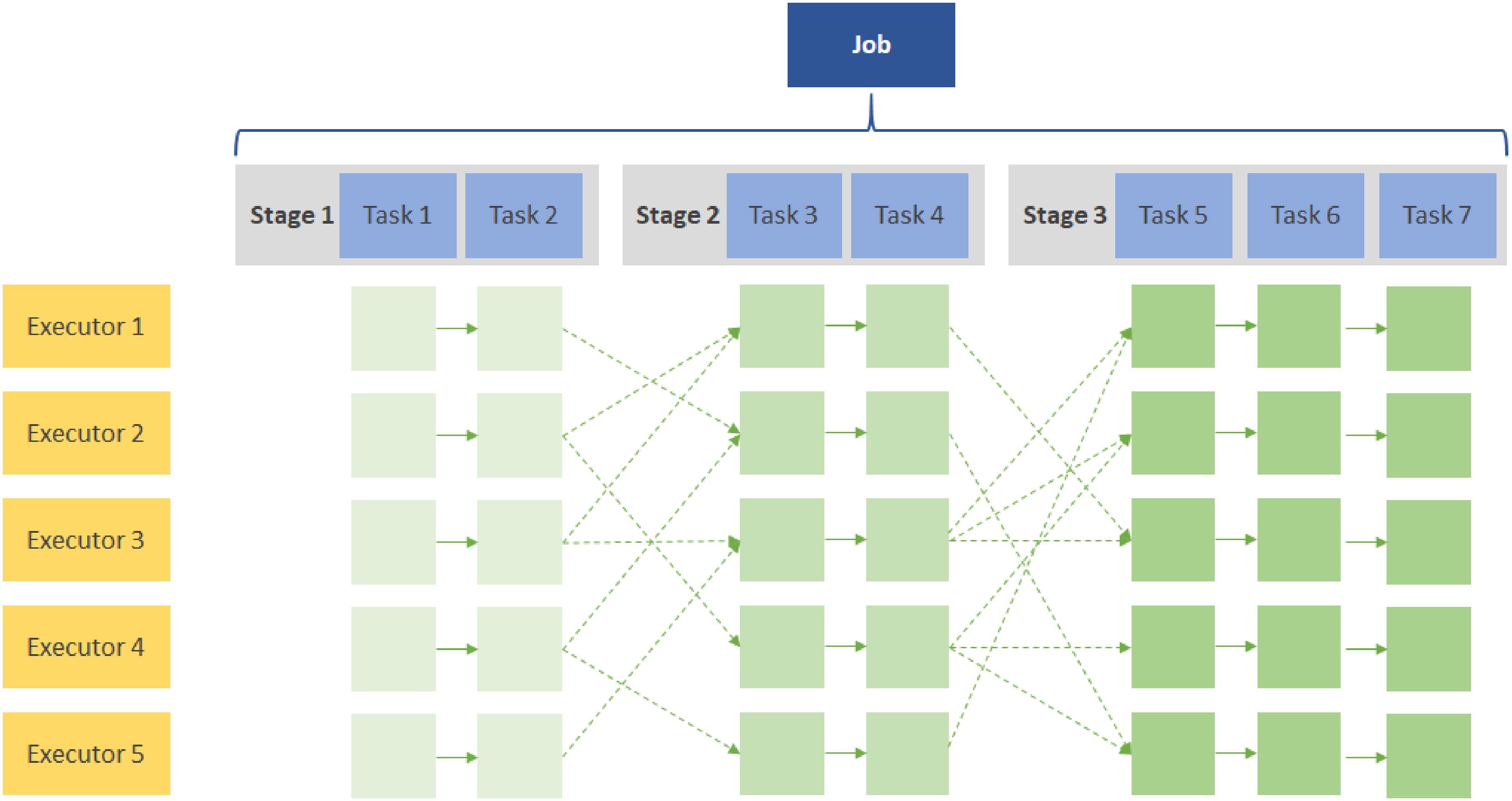
In Spark 2.0, the **SparkSession** became a unified conduit to all Spark operations and data. Subsumes the SparkContext, SQLContext, HiveContext, SparkConf, and StreamingContext.

Terminologies of Spark Cluster

Big Data Engineering For Analytics

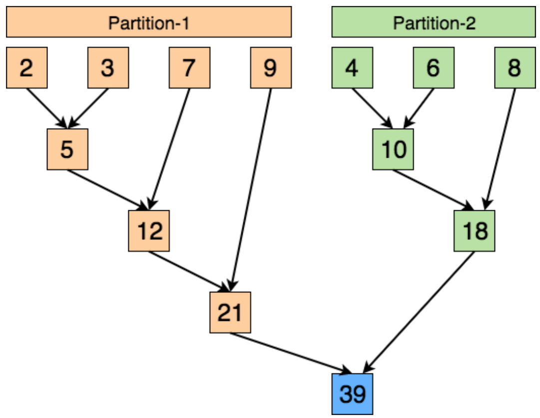
Term	Meaning
Application	User program built on Spark. Consists of a *driver program* and *executors* on the cluster.
Application jar	A jar containing the user's Spark application. In some cases users will want to create an "user jar" containing their application along with its dependencies. The user's 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)
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.

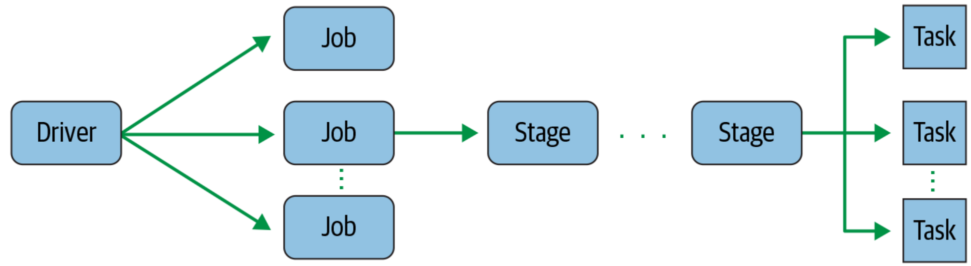
Spark Job, Stage, Task

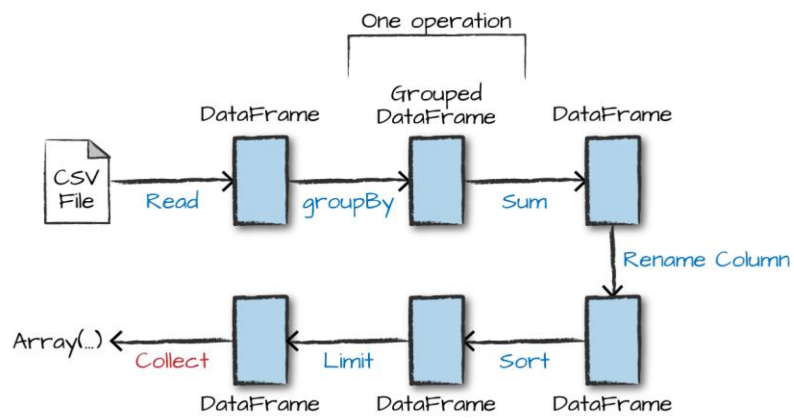


Narrow Wide Reduce Operations

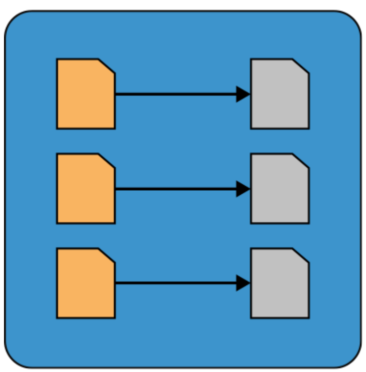
Reduction Concept



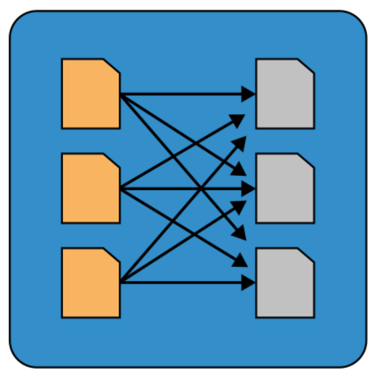




Narrow Dependencies



Wide Dependencies



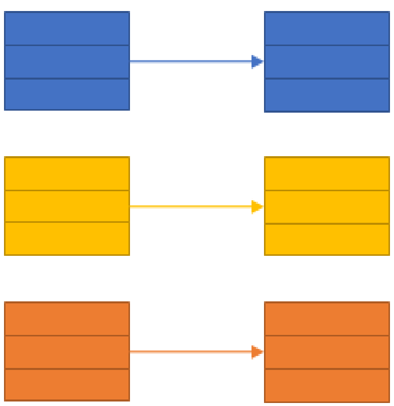
Spark Operations

Big Data Engineering For Analytics

Transformations	map filter sample groupByKey reduceByKey sortByKey	flatMap union join cogroup cross mapValues
Actions	collect reduce count save lookupKey	

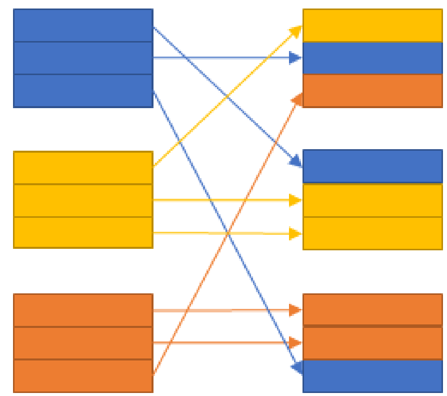
Narrow and Wide Operations

Narrow dependency

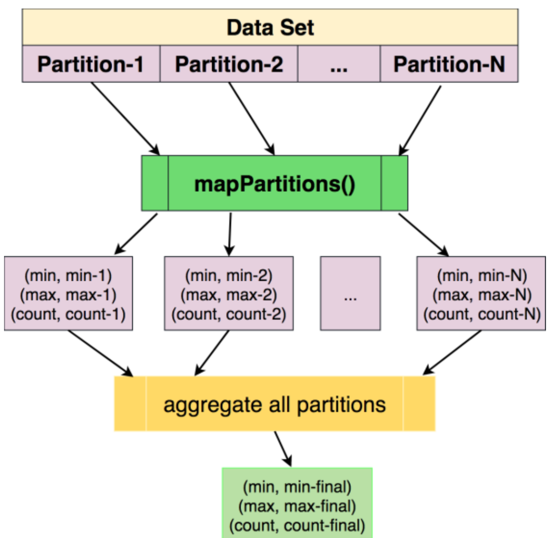


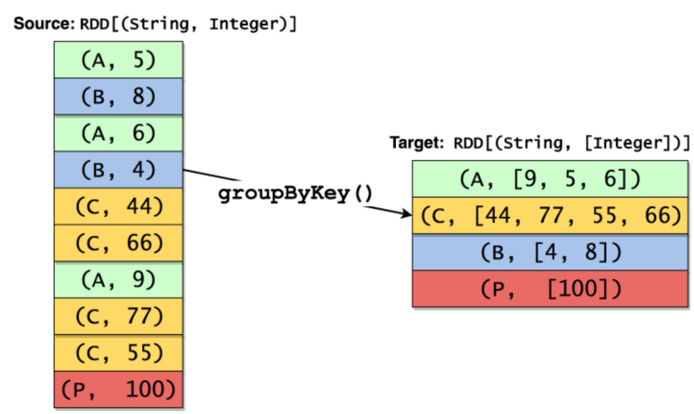
- Map
- FlatMap
- MapPartition
- Filter
- Sample
- Union

Wide dependency



- Intersection
- Distinct
- ReduceByKey
- GroupByKey
- Join
- Cartesian

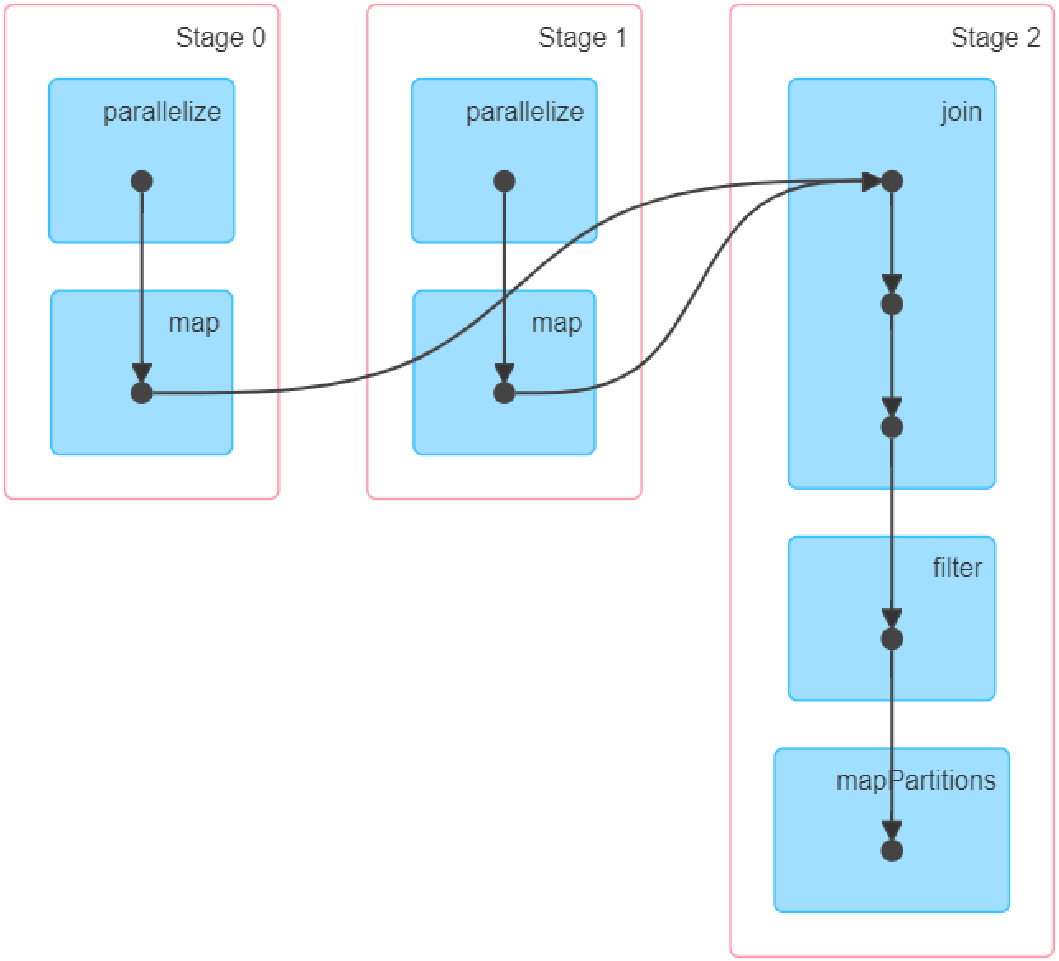




Spark Job, Stage, Task - Example

```
> val RDD1 = sc.parallelize(Array('1', '2', '3', '4', '5'))
  .map{ x => val xi = x.toInt; (xi, xi+1) }
> val RDD2 = sc.parallelize(Array('1', '2', '3', '4', '5'))
  .map{ x => val xi = x.toInt; (xi, xi*10) }
> val joinedRDD = RDD2.join(RDD1)
> val filteredRDD =  joinedRDD.filter{case (k,v) => k%2}
> Val resultRDD = filteredRDD.mapPartitions{iter => iter
.map{case(k, (v1,v2) => (k, v1+v2))}}
> resultRDD.collect()
Array[(Int, Int)] = Array((52, 573),(53,551))
```

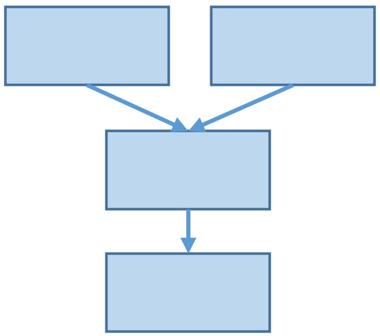
DAG visualization



- When creating RDDs, Spark generates two stage for each of RDD, shown as Stage 0 and Stage 1
- **Map** is narrow operation, it includes in Stage 0 and Stage 1
- **Join** is wide operation, Spark generates another stage – Stage 3
- **Filter** and **mapPartitions** are narrow operations, both include in Stage 3

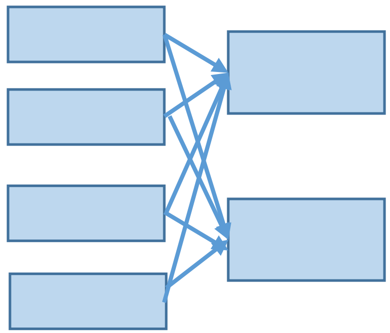
DAG visualization

Driver
(Define RDD objects)



rdd1.join(rdd2).groupby(…).filter()…

DAG Scheduler
(Like a queue planner)



Split graph into stages of tasks

Task Scheduler

Cluster Manager

Launch tasks via cluster manager

Worker

Threads

Block Manager

Executer tasks
Store and serve block

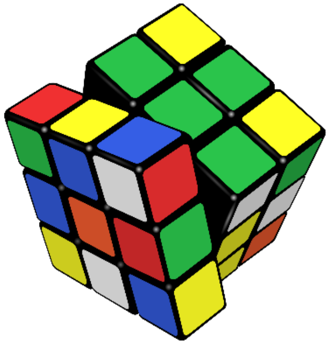
Simple Spark Apps: WordCount

- Definition:
- This simple program provides a good test case for parallel processing, since it:
  - requires a minimal amount of code
  - demonstrates use of both symbolic and numeric values
  - isn't many steps away from search indexing
  - serves as a "Hello World" for Big Data apps
- A distributed computing framework that can run WordCount efficiently in parallel at scale can likely handle much larger and more interesting compute problems

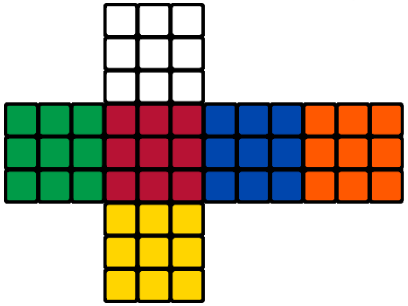
count how often each word appears in a collection of text documents

Demo

```
val f = sc.textFile("README.md")
val wc = f.flatMap(l => l.split(" ")).map(word => (word, 1)).reduceByKey(_ + _)
wc.saveAsTextFile("wc_out.txt")
```



Big Data Engineering For Analytics



Spark Examples

Data is a precious thing and will last longer than the systems themselves.

~Tim Berners-Lee

Sample Self-Contained Applications

Big Data Engineering For Analytics

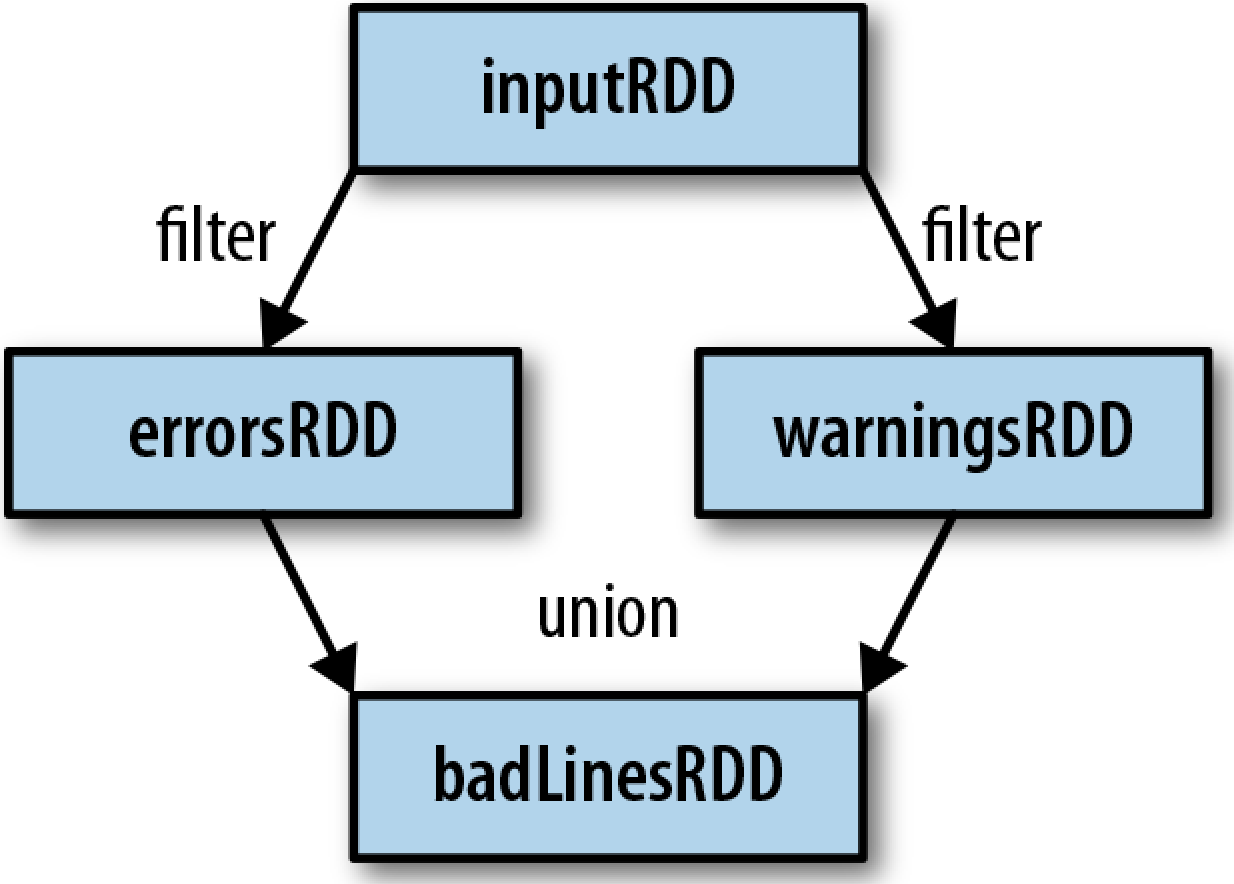
Demo

```
import org.apache.spark.sql.SparkSession
import org.apache.log4j.Level
import org.apache.log4j.Logger

object SimpleExample {
  def main(args: Array[String]) {
    Logger.getLogger("org").setLevel(Level.OFF)
    // Should be some file in your system
    val logFile = "C:/spark-2.4.5-bin-hadoop2.7/README.md"
    val spark = SparkSession.builder.appName("SimpleExample")
        .config("spark.master", "local").getOrCreate()
    val logData = spark.read.textFile(logFile).cache()
    val numAs = logData.filter(line => line.contains("a")).count()
    val numBs = logData.filter(line => line.contains("b")).count()
    println("Lines with a: %s, Lines with b: %s".format(numAs, numBs))
  }

}
```

Example: Log Mining

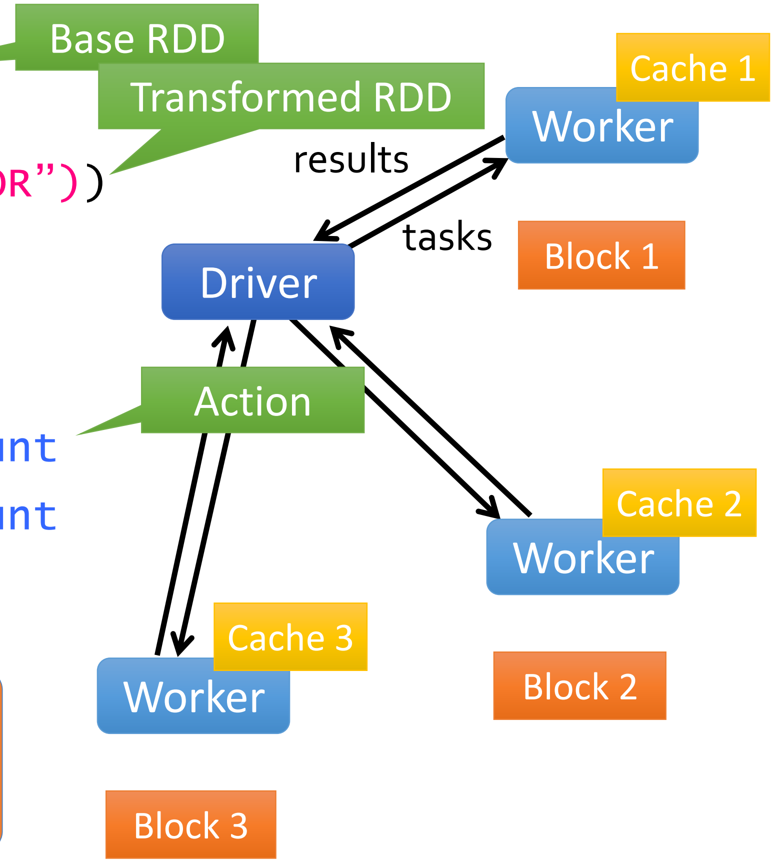


Example: Log Mining

Load error messages from a log into memory, then interactively search for various patterns

```
lines = spark.textFile("hdfs://...")
errors = lines.filter(_.startsWith("ERROR"))
messages = errors.map(_.split('\t')(2))
cachedMsgs = messages.cache()

cachedMsgs.filter(_.contains("foo")).count
cachedMsgs.filter(_.contains("bar")).count
. . .
```



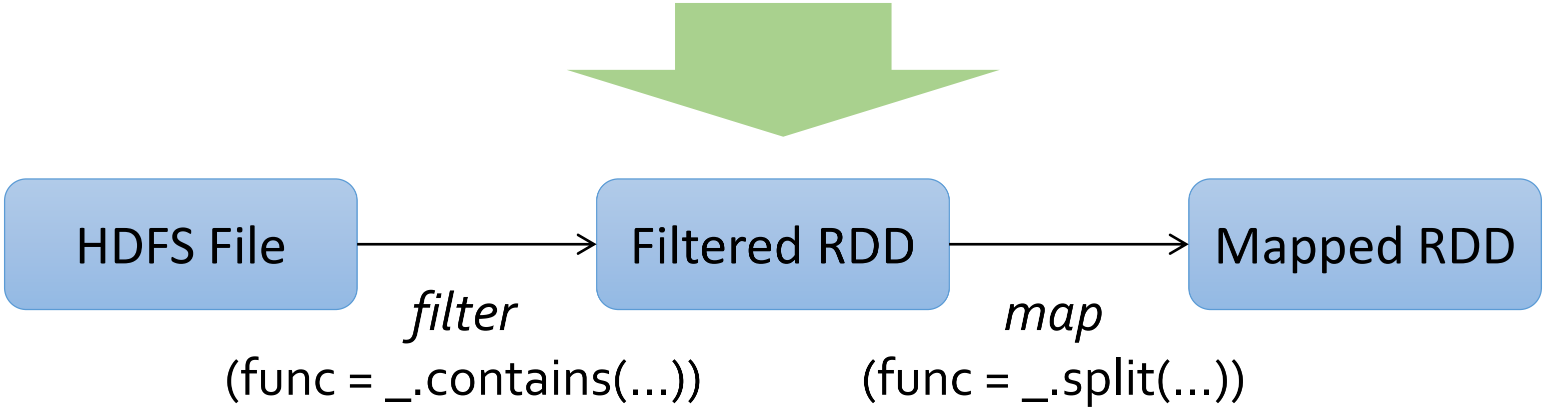
Result: scaled to 1 TB data in 5-7 sec
(vs 170 sec for on-disk data)

Lineage

Spark maintains *lineage* information that can be used to reconstruct lost partitions

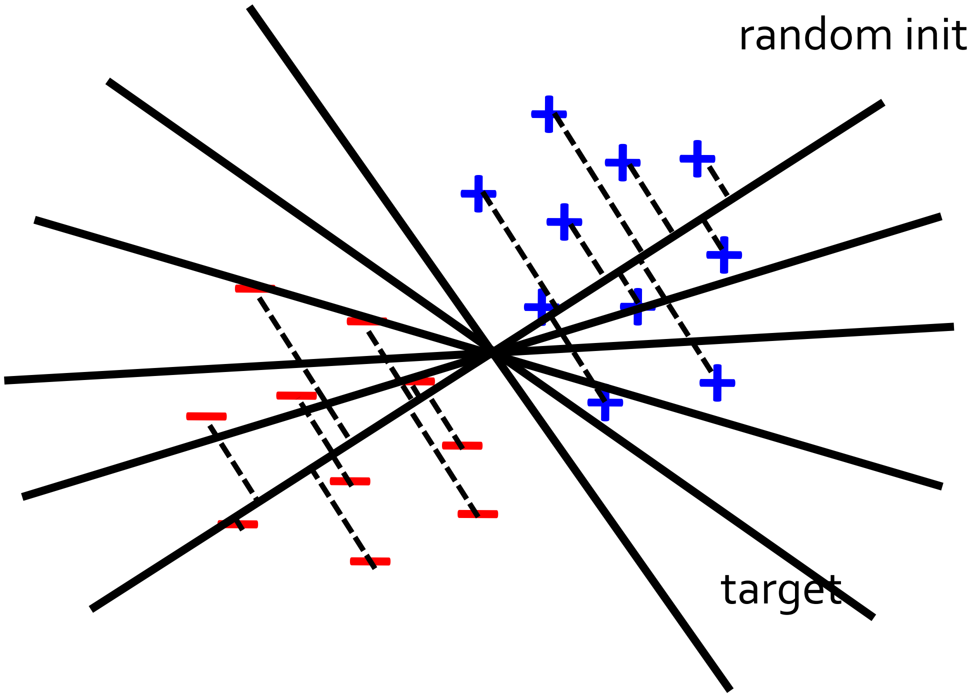
Ex:

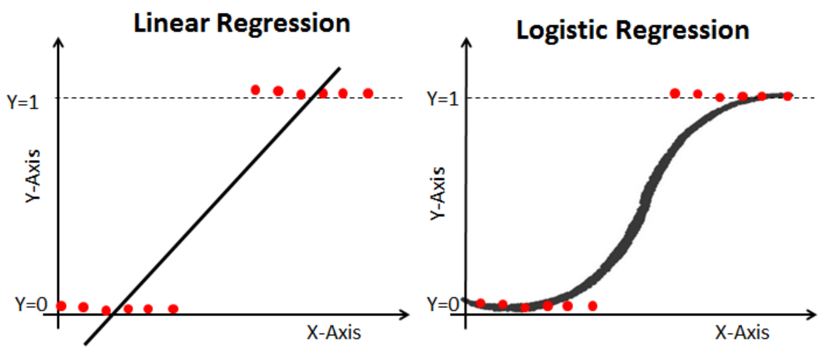
```
messages = textFile(...).filter(_.startsWith("ERROR"))
                        .map(_.split('\t')(2))
```



Example: Logistic Regression

Goal: find best line separating two sets of points





https://www.datacamp.com/community/tutorials/understanding-logistic-regression-python

Example: Logistic Regression

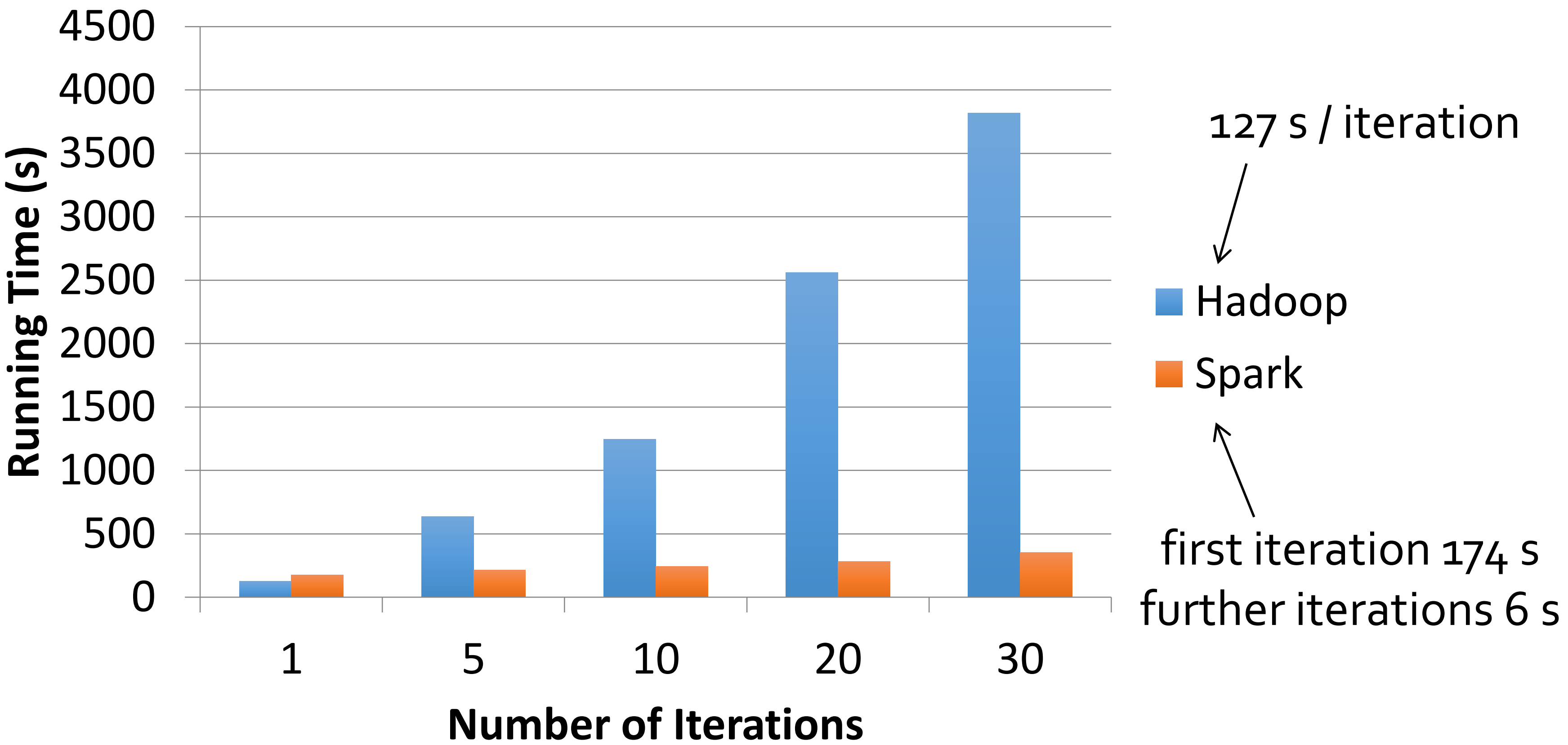
```
val data = spark.textFile(...).map(readPoint).cache()

var w = Vector.random(D)

for (i <- 1 to ITERATIONS) {
  val gradient = data.map(p =>
    (1 / (1 + exp(-p.y*(w dot p.x))) - 1) * p.y * p.x
  ).reduce(_ + _)
  w -= gradient
}

println("Final w: " + w)
```

Logistic Regression Performance



More Application Examples

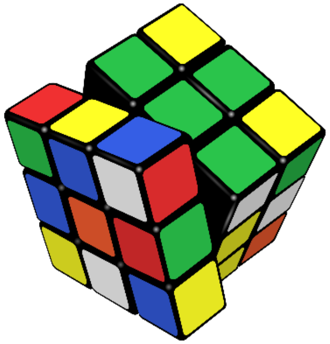
Big Data Engineering For Analytics

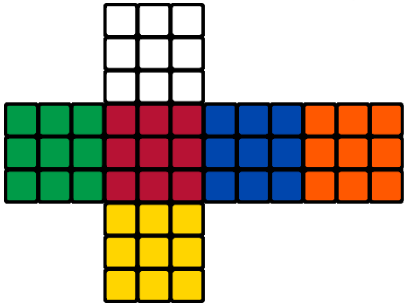
- In-memory data mining on Hive data (Conviva)
- Log analysis applications
- Weather TimeSeries Data Application
- Predictive analytics (Quantifind)
- City traffic prediction (Mobile Millennium); Twitter spam classification (Monarch); Collaborative filtering via matrix factorization
- Game industry, processing and discovering patterns from the potential firehose of real-time in-game events
- e-commerce industry, real-time transaction information could be passed to a streaming clustering algorithm like k-means or collaborative filtering like ALS
- Finance or security industry, the Spark stack could be applied to a fraud or intrusion detection system or risk-based authentication

…

https://github.com/databricks/reference-apps

http://spark.apache.org/examples.html





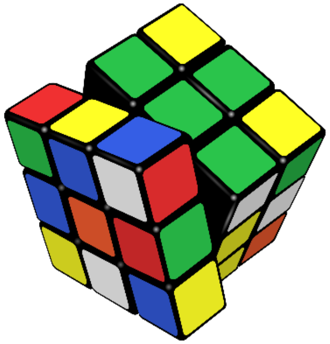
Summary

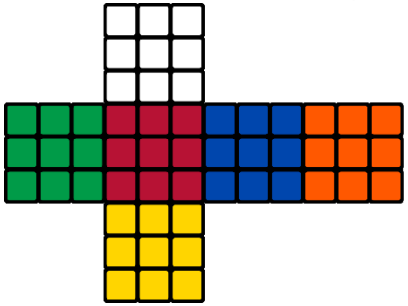
Simplicity is the ultimate sophistication.

~Leonardo da Vinci

Essential Points

- **Spark provides a simple, efficient, and powerful programming model for a wide range of apps**
  - There are a range of options available that allow the cluster creator to define attributes such as cluster size and storage type.
  - Every Spark application consists of a driver program that runs the user's main function and executes various parallel operations on a cluster.
  - Spark has the concept of a **big data pipeline** — from ETL to Analytics





References

Genius ain't anything more than elegant common sense.

~Josh Billings

References

- Official Spark Documentation and Wiki
- Programmer and User Guides
- Books:
  - Learning Spark, 2nd Edition, by Holden Karau, Andy Konwinski, Patrick Wendell and Matei Zaharia (O'Reilly Media)
  - Spark in Action, by Marko Bonaci and Petar Zecevic (Manning)
  - Fast Data Processing with Spark, by Krishna Sankar and Holden Karau (Packt Publishing)
  - Spark Cookbook, by Rishi Yadav (Packt Publishing)
  - Mastering Apache Spark, by Mike Frampton (Packt Publishing)