LEARN.
CONNECT.
ELEVATE.

YCBS 257 - Data at Scale (Winter 2019)

Instructor: Khaled Tannir

McGill | School of Continuing Studies

mcgill.ca
/continuingstudies

School of Continuing Studies

YCBS 257-256 / 257 - Data at Scale (BIG DATA)

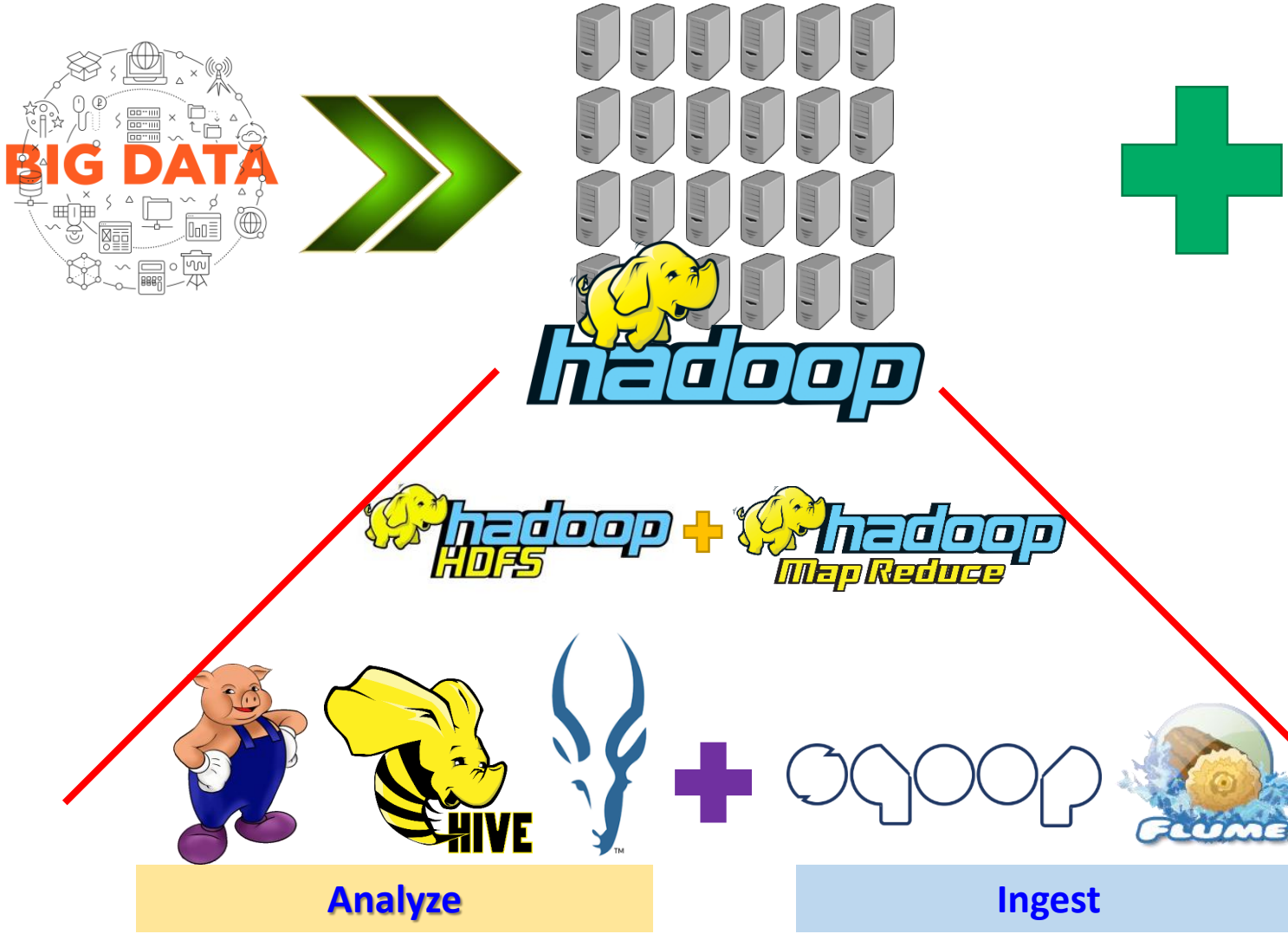
Course 9

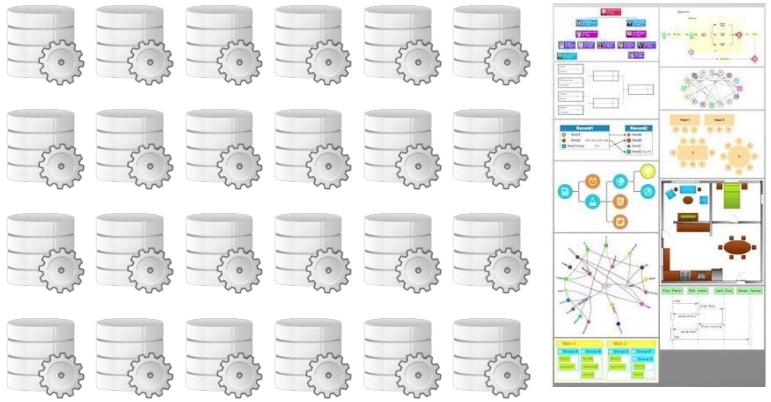
In-Memory Data Processing

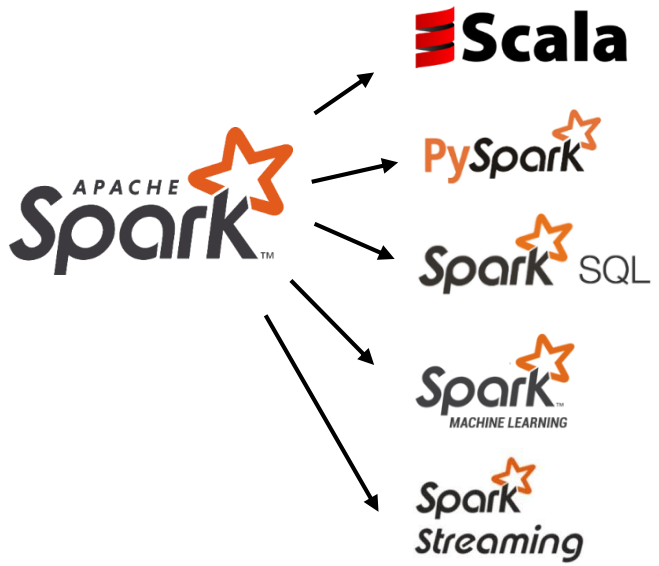
Khaled Tannir

McGill

Machine Learning at Scale







Machine Learning at Scale

Volume

Low Velocity — High Velocity

Distributed Storage

Volume

On Disk — In-Memory

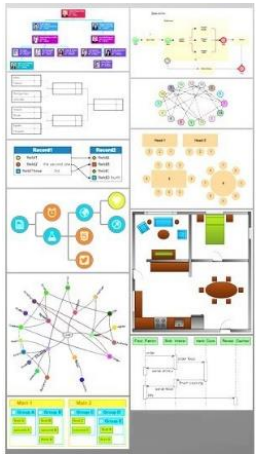
Distributed Processing

Velocity

Low Velocity — High Velocity

Variety

Unstructured — Structured



Theme of this Course

In-Memory Data Processing

- *Apache Spark*
  - *Core Concepts*
  - *RDDs, DataFrames*
  - *SparkSQL*
  - *Core Concepts*

Apache Spark

In-memory Large Scale Data Processing

McGill

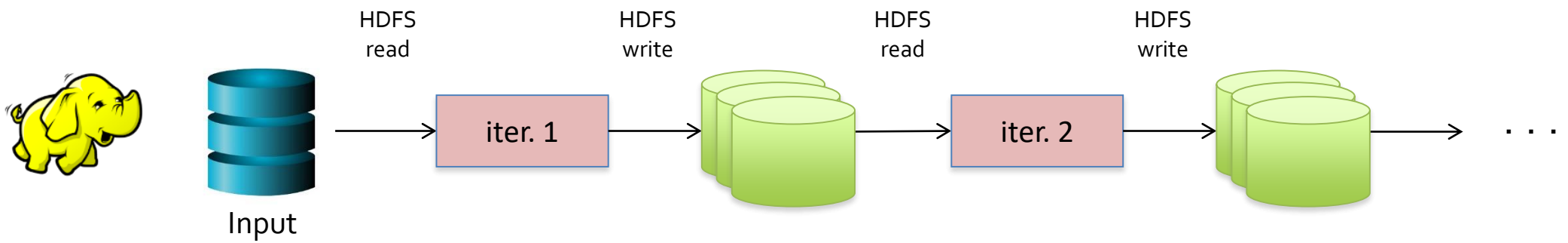
FALL 2018

What is Apache Spark?

- In-memory analytics open source framework
- 10x (on disk) - 100x (In-Memory) faster than Hadoop
- Highly compatible with Hadoop Storage API and can run on top of it
- Designed for executing complex multi-stage applications, like machine learning
- Developer can write programs in Scala, Python or Java
- Written in Scala and Initially started at UC Berkeley AMPLab in 2009

http://spark.apache.org

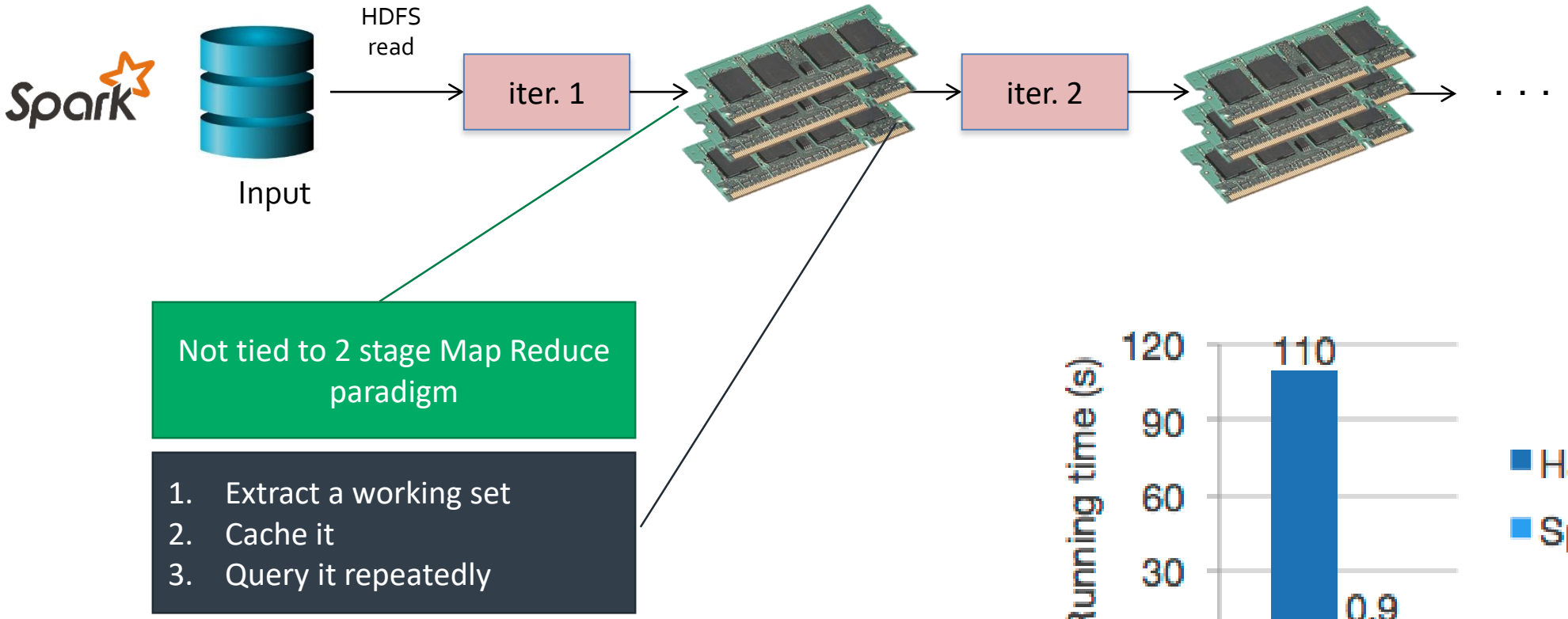
Why do I need Spark?



- **Most of Machine Learning Algorithms are iterative because each iteration can improve the results**
- **With Disk based approach each iteration's output is written to disk making it slow**

Why do I need Spark?

Spark execution flow



Logistic regression in Hadoop and Spark

Spark Performance : Terasort

	Hadoop World Record	Spark 100 TB	Spark 1 PB
Data Size	102.5 TB	100 TB	1000 TB
Elapsed Time	72 mins	23 mins	234 mins
# Nodes	2100	206	190
# Cores	50400	6592	6080
# Reducers	10,000	29,000	250,000
Rate	1.42 TB/min	4.27 TB/min	4.27 TB/min
Rate/node	0.67 GB/min	20.7 GB/min	22.5 GB/min
Sort Benchmark Daytona Rules	Yes	Yes	No
Environment	dedicated data center	EC2 (i2.8xlarge)	EC2 (i2.8xlarge)

Spark Stack

- **Spark SQL**

  For SQL and unstructured data processing

- **MLib**

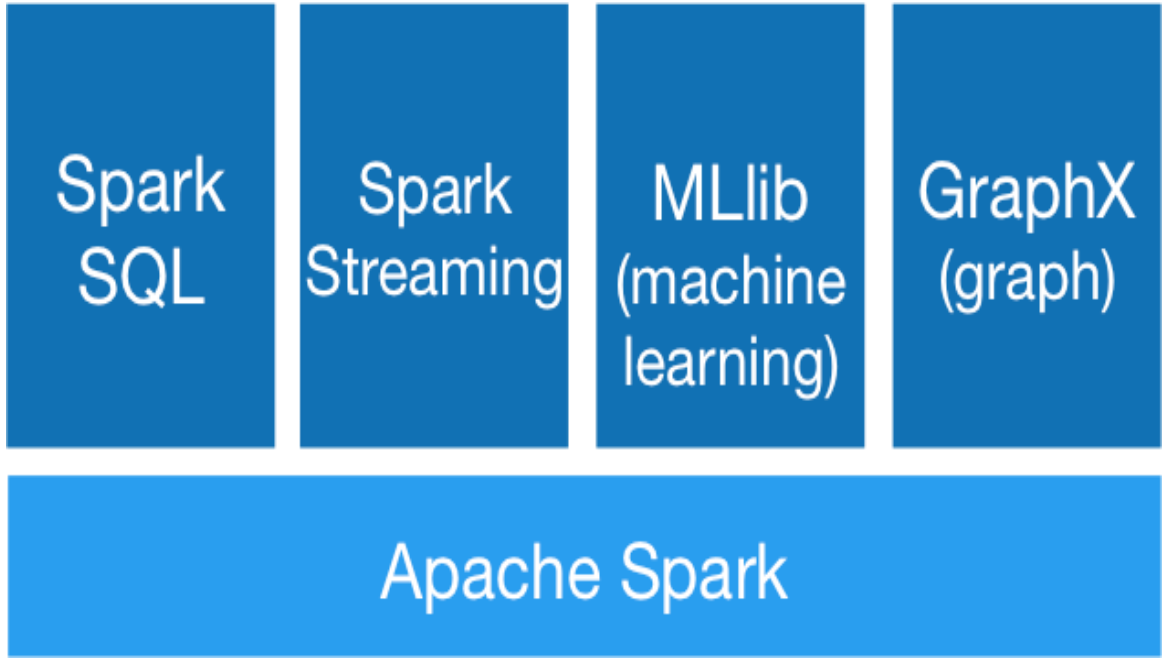
  Machine Learning Algorithms

- **GraphX**

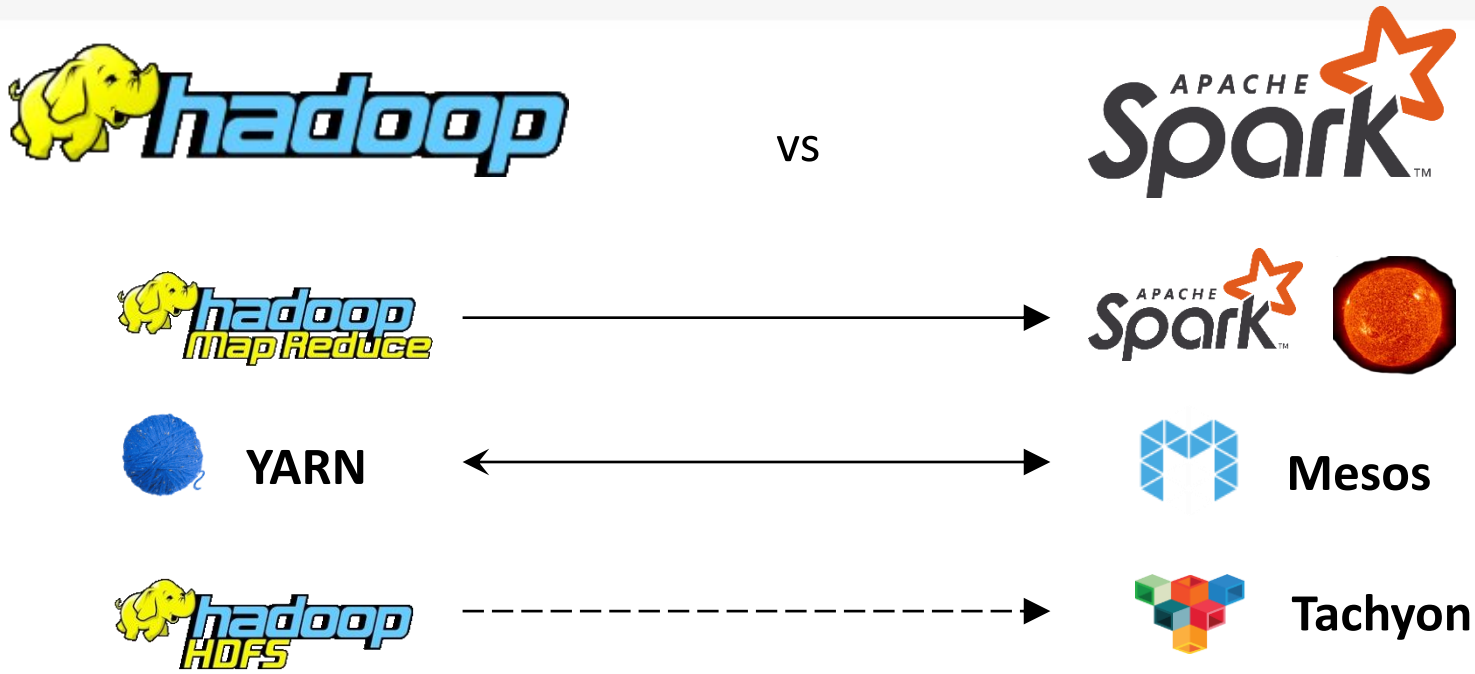
  Graph Processing

- **Spark Streaming**

  Stream processing of live data streams



Hadoop vs Spark

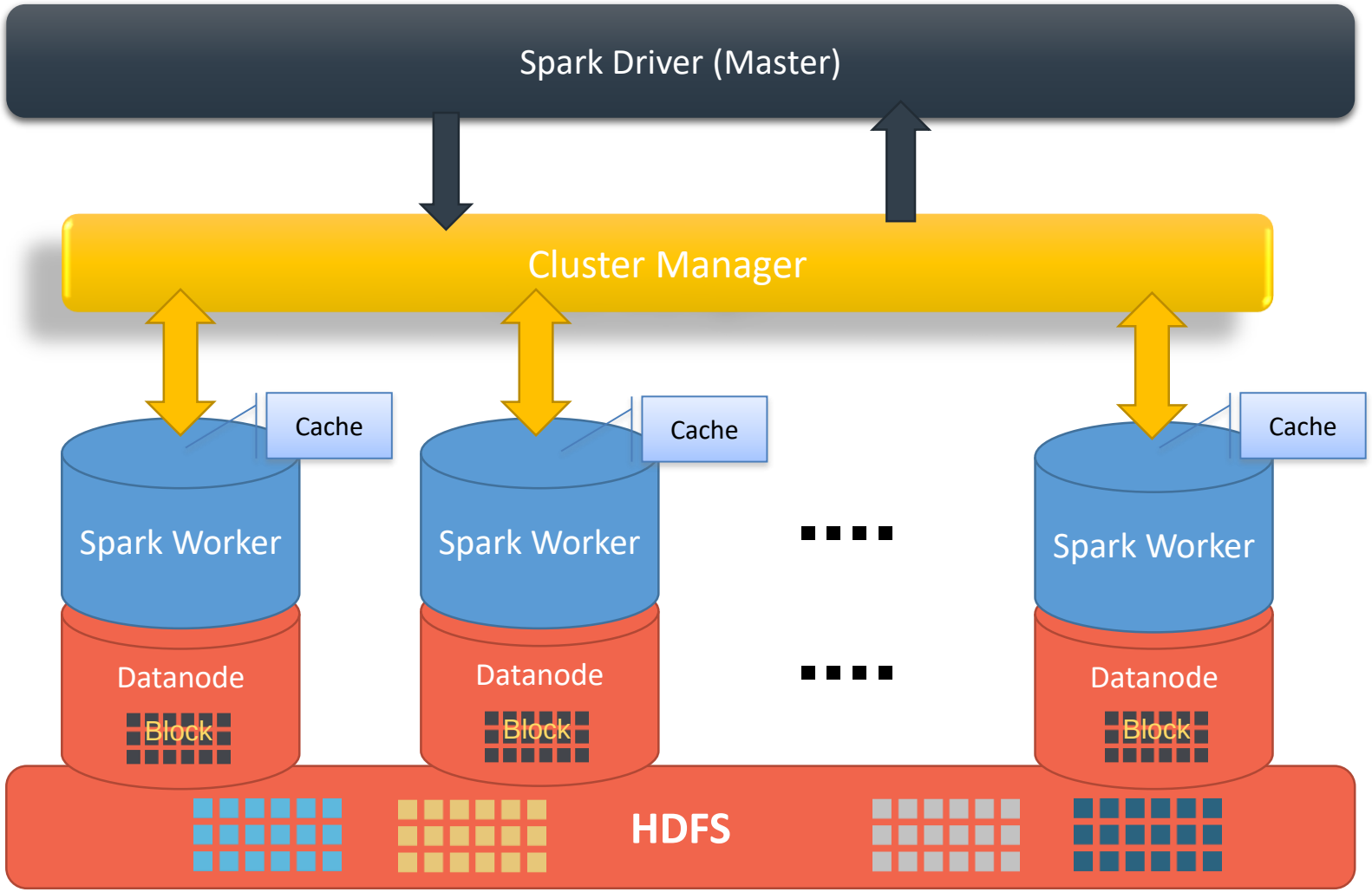


Still need Hadoop?

- Yes, why Hadoop?
  - HDFS and YARN
  - MapReduce is used for many cases
  - Others tools : Sqoop, Flume, etc…

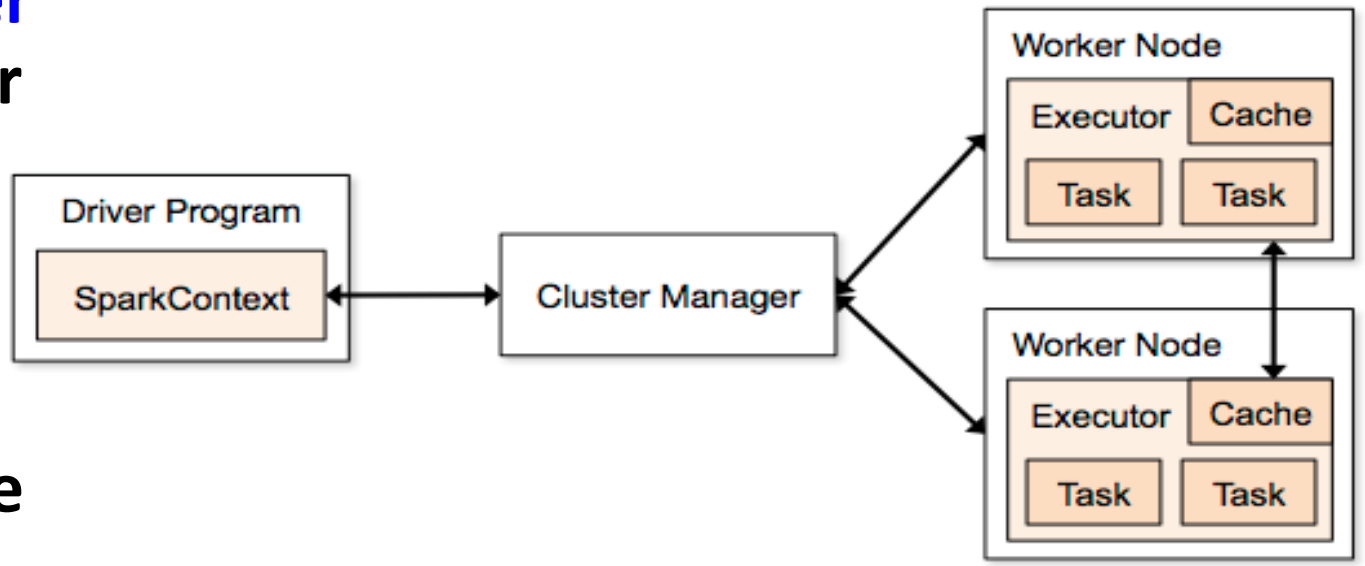
- Spark can use other storage system and cluster management system
  - Amazon S3
  - Mesos

Spark architecture



Execution Flow

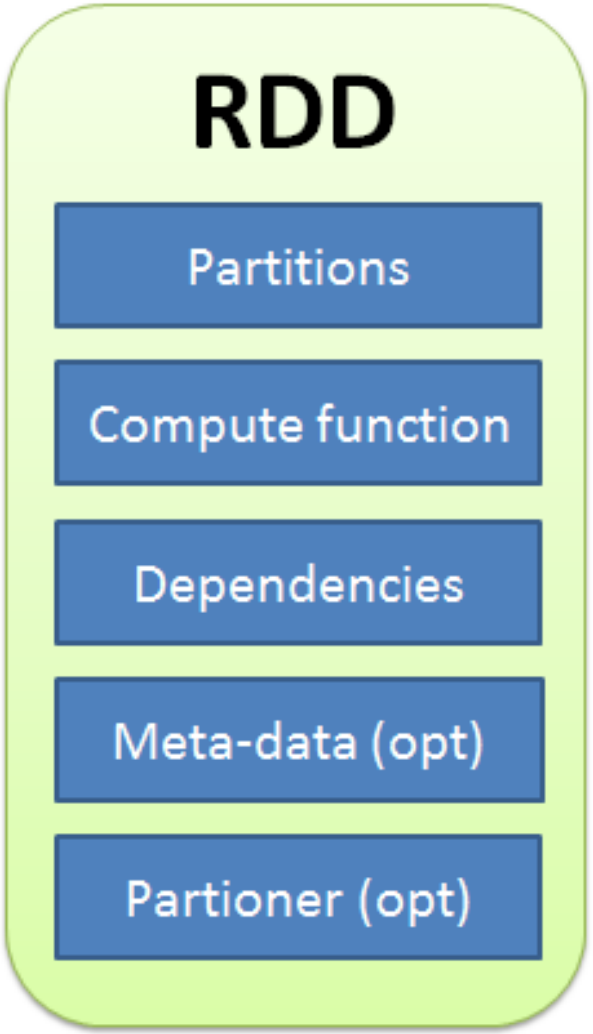
- The Master connect to the Cluster Manager to allocate resources for an application
- Acquire Executors to: process, launch tasks and cache data
- Executors get an Application code
- Send tasks to run to Executors



Spark RDDs

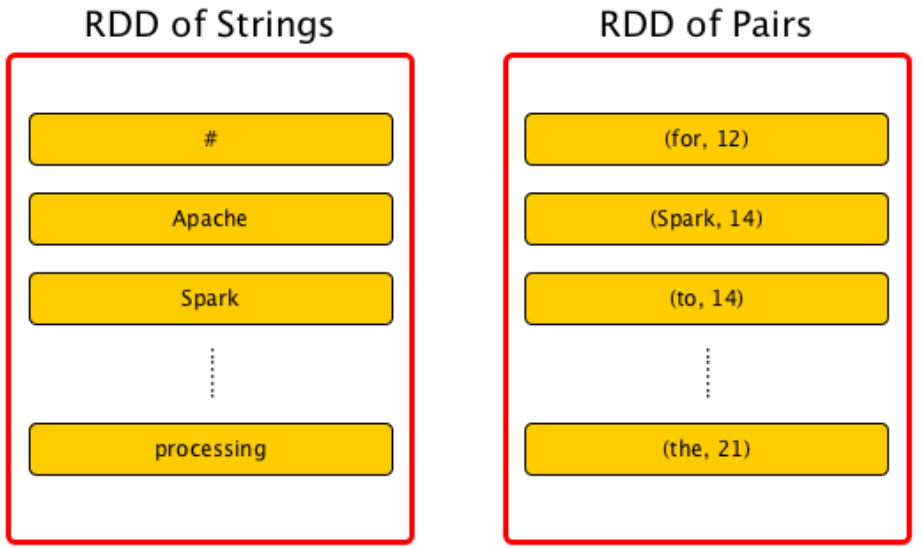
- **Resilient** : able to recompute missing or damaged partitions due to node failures
- **Distributed** : with data residing on multiple nodes in a cluster.
- **Dataset** : is a collection of partitioned data with primitive values

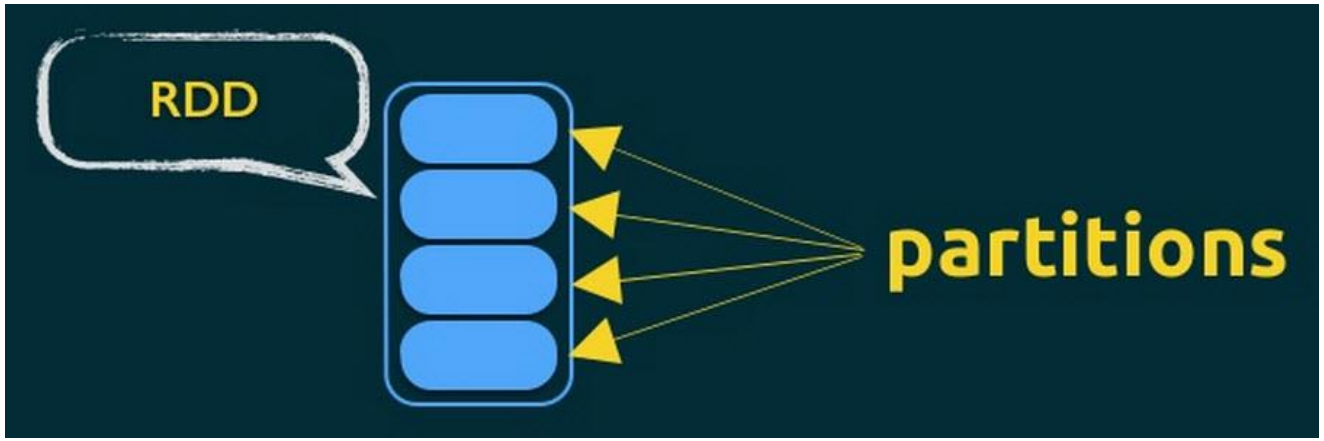
RDDs are Immutable (or Read-Only)

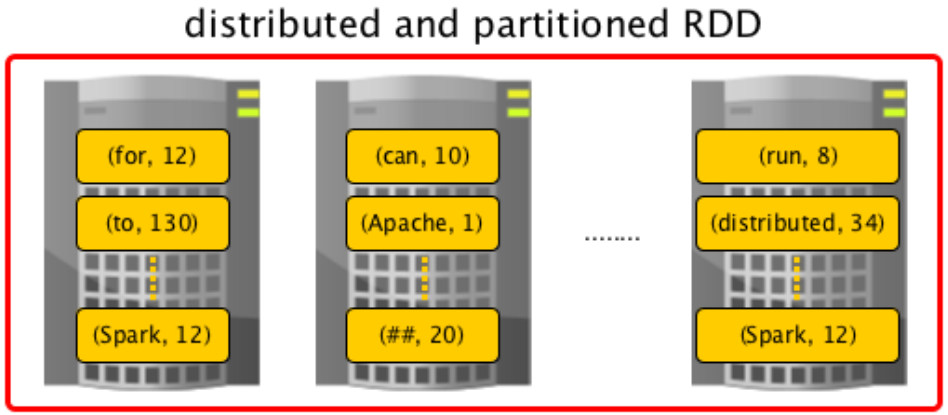


How to create an RDD?

- From a file
- From data in memory
- From another RDD (Transformation)







RDD

- **Programmer can perform 3 types of operations**

Transformations

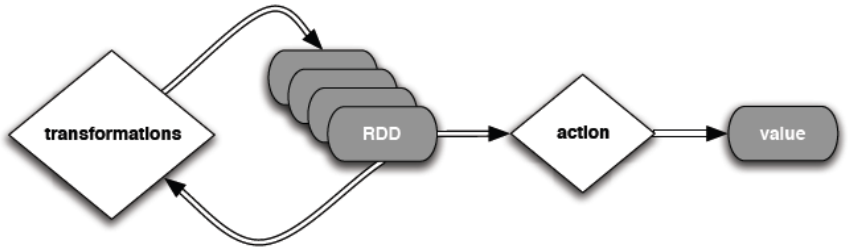
- Create a new dataset from and existing one.
- Lazy in nature. They are executed only when some action is performed.
- Example :
  - Map(func)
  - Filter(func)
  - Distinct()

Actions

- Returns to the driver program a value or exports data to a storage system after performing a computation.
- Example:
  - Count()
  - Reduce(funct)
  - Collect
  - Take()

Persistence

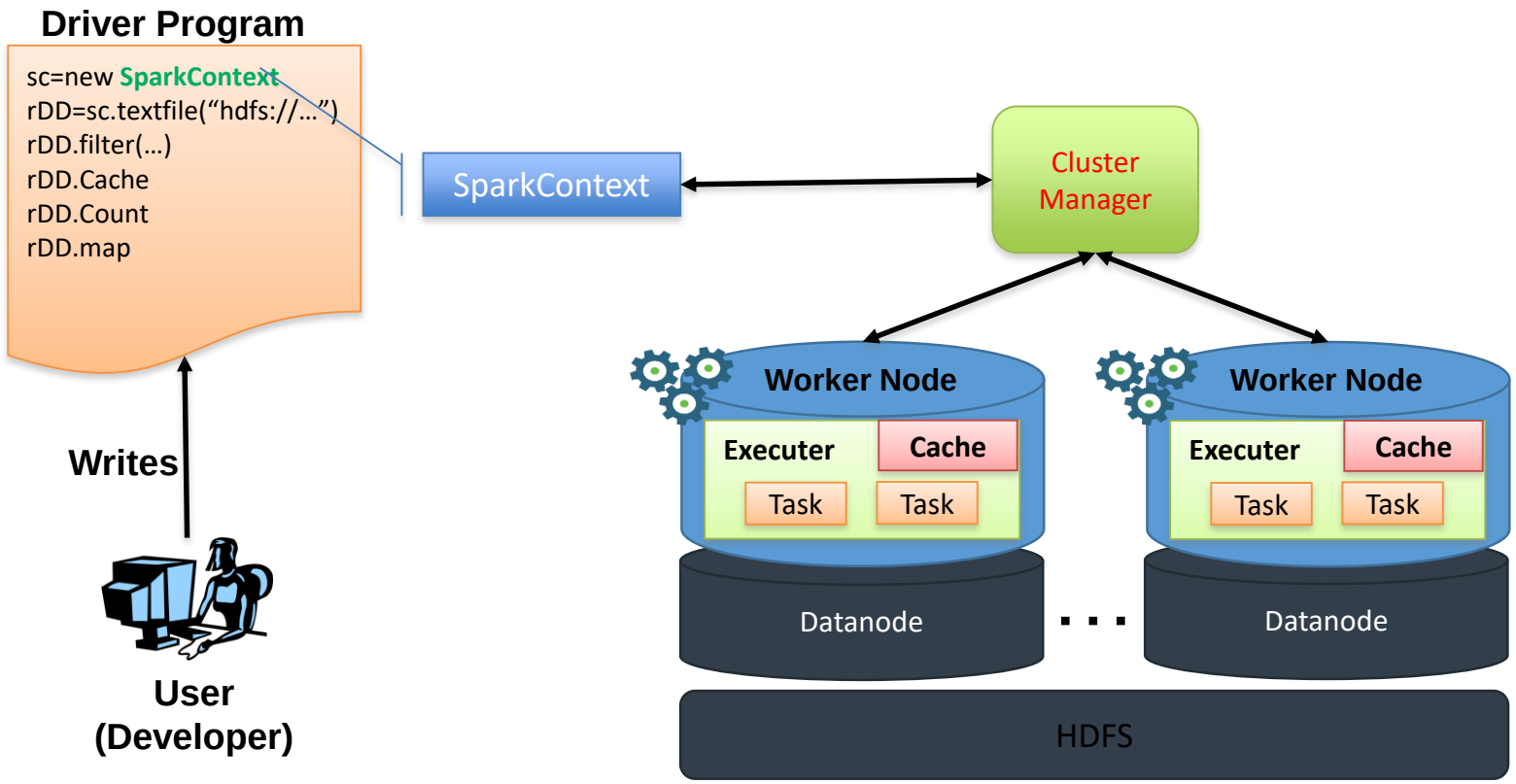
- For caching datasets in-memory for future operations.
- Option to store on disk or RAM or mixed (Storage Level).
- Example:
  - Persist()
  - Cache()



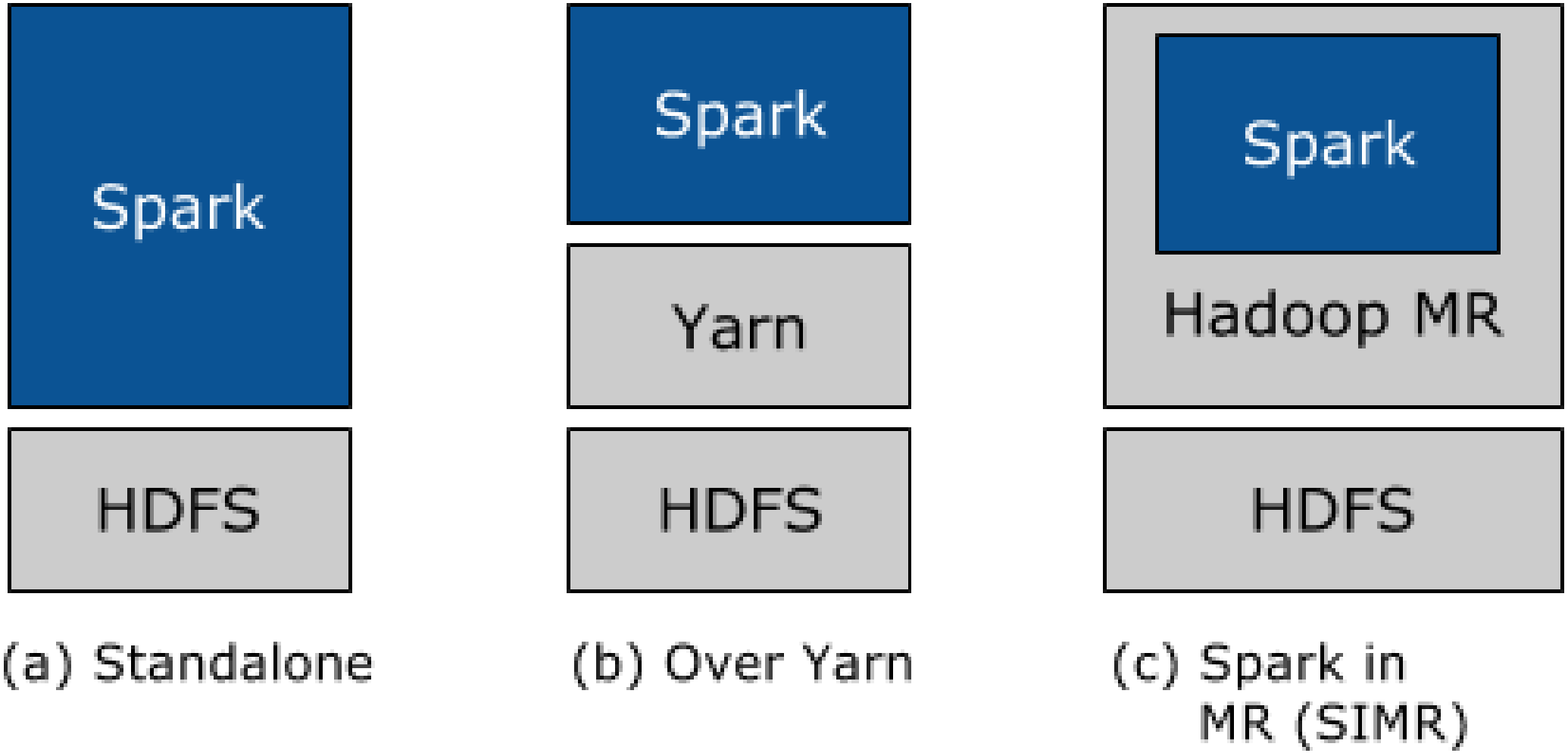
1. Transformations
   a. Map
   b. Filter
   c. FlatMap
   d. Sample
   e. Union
   f. Intersect
   g. Distinct
   h. GroupByKey
   i. ....

2. Actions
   a. Reduce
   b. Collect
   c. Count
   d. First
   e. Take(n)
   f. TakeSample
   g. SaveAsTextFile
   h. ....

Spark Programming Model



Spark Running Modes



(a) Standalone

(b) Over Yarn

(c) Spark in MR (SIMR)

Spark Shell

- **spark-shell** run Spark shell for **Scala**
- **pyspark** run Spark shell for **Python**

```
./bin/spark-shell --master local[2]
```

master	*description*
`local`	run Spark locally with one worker thread (no parallelism)
`local[K]`	run Spark locally with K worker threads (ideally set to # cores)
`spark://HOST:PORT`	connect to a Spark standalone cluster; PORT depends on config (7077 by default)
`mesos://HOST:PORT`	connect to a Mesos cluster; PORT depends on config (5050 by default)

```
./bin/run-example SparkPi 10
```

This will run 10 iterations to calculate the value of Pi

web console : http://localhost:4040

Basic operations…

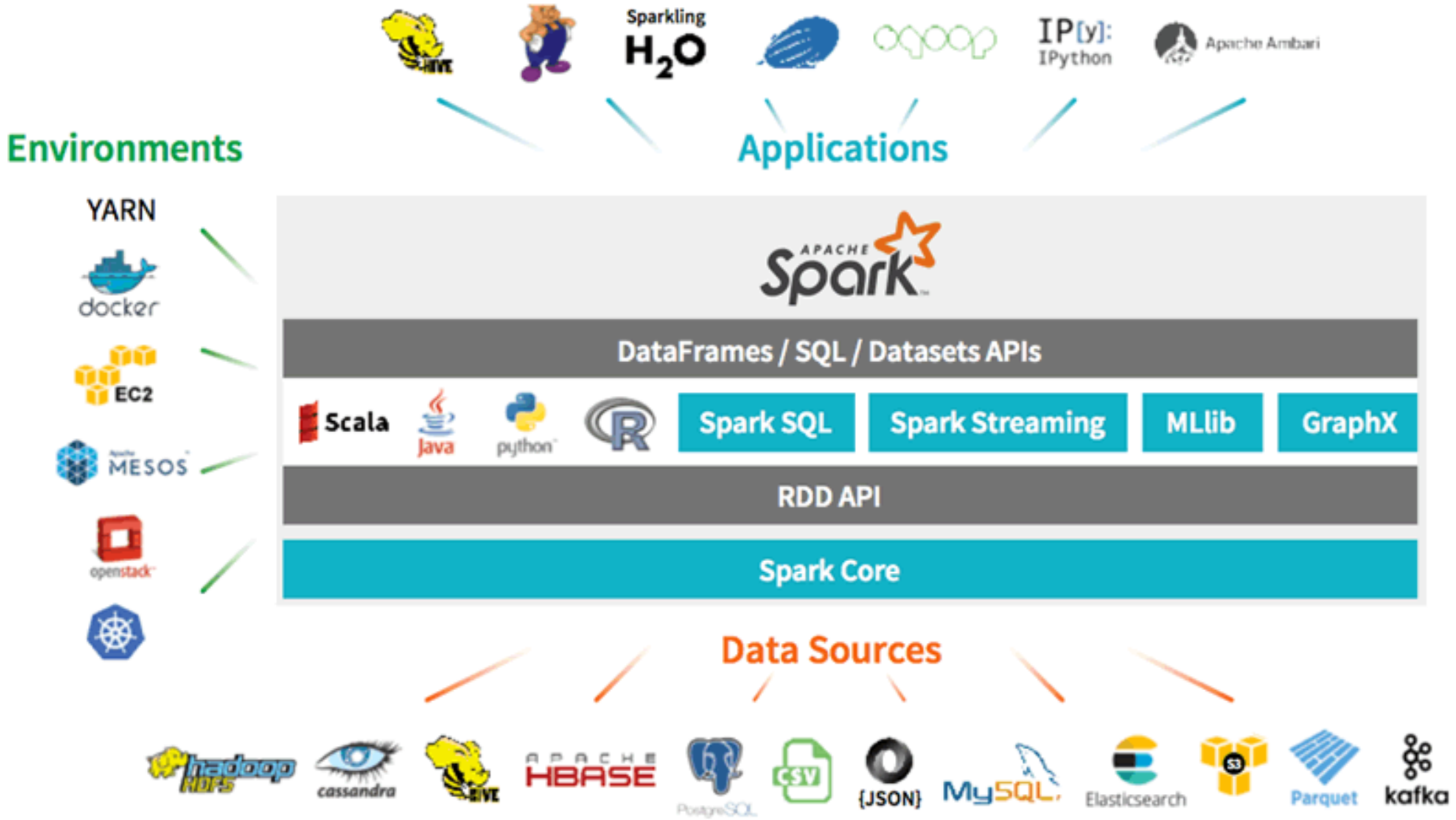
```
scala> val textFile = sc.textFile("README.md")
textFile: spark.RDD[String] = spark.MappedRDD@2ee9b6e3
```

```
scala> textFile.count() // Number of items in this RDD
ees0: Long = 126

scala> textFile.first() // First item in this RDD
res1: String = # Apache Spark
```

```
scala> val linesWithSpark = textFile.filter(line =>
line.contains("Spark"))
linesWithSpark: spark.RDD[String] = spark.FilteredRDD@7dd4af09
```

```
Simplier - Single liner:
scala> textFile.filter(line => line.contains("Spark")).count()
// How many lines contain "Spark"?
res3: Long = 15
```

Spark Ecosystem



SparkSQL

Introduction

Spark SQL

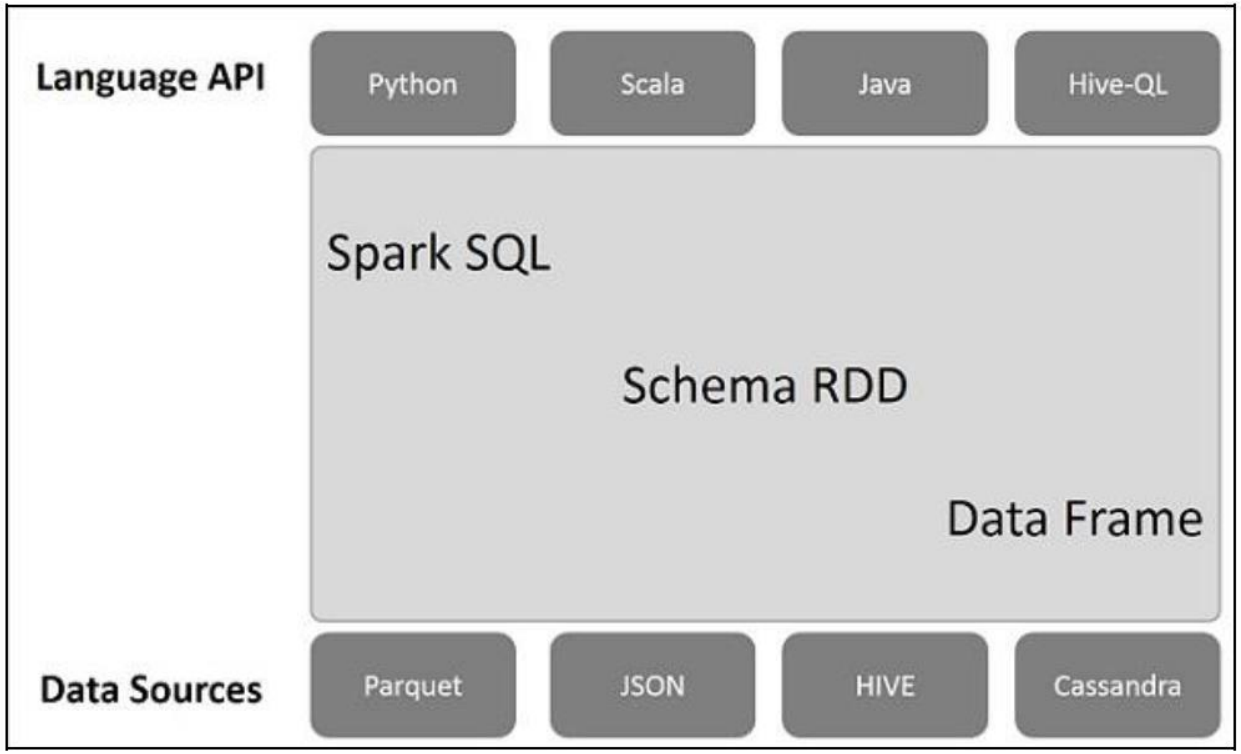
- Spark SQL is a Spark module for structured data processing
- Spark SQL is a component on top of Spark Core that introduces new data abstraction called SchemaRDD.
- Spark SQL was first released in Spark 1.0 (May, 2014).
- It provides a programming abstraction called DataFrame and can act as distributed SQL query engine.

Spark SQL Architecture

- **Language API :**
  - Spark is compatible with different languages and Spark SQL.
  - It is also, supported by these languages-API (python, scala, java, HiveQL).

- **Schema RDD:**
  - Spark Core is designed with RDD structure.
  - Generally, Spark SQL works on schemas, tables, and records.

- **Data Sources:**
  - Usually the Data source for spark-core is a text file, Avro file, etc.
  - Data Sources for Spark SQL are Parquet file, JSON document, HIVE tables, and Cassandra database.



What is a DataFrame?

Spark SQL

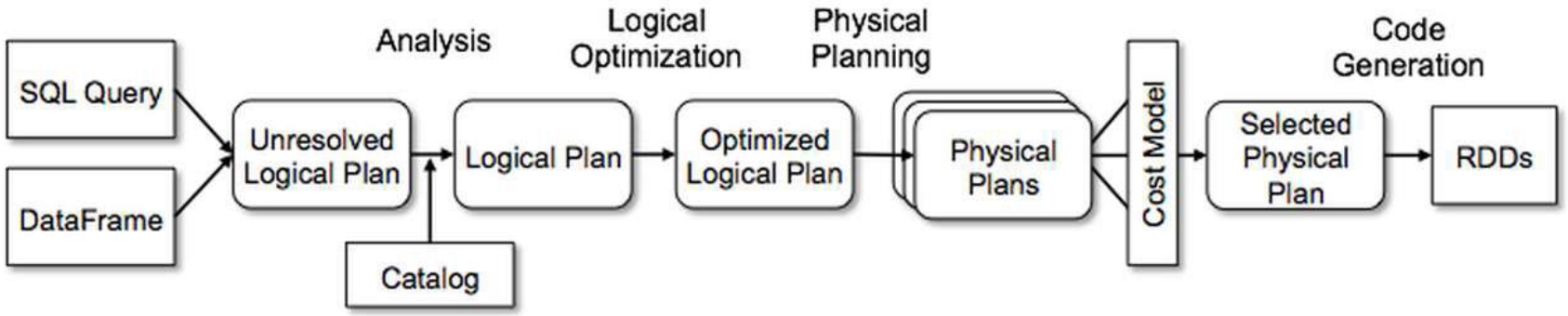
- **DataFrame is a container for Logical Plan**

  Logical Plan is a tree which represents data and schema

- **Every transformation is represented as tree manipulation**

  All tree manipulations are done using scala case class

- **Logical plan will be converted to physical plan for execution**



SparkSQL : SQLContext

- **Used to Create DataFrames**
  - **Spark Shell automatically creates a SparkContext as the sqlContext variable**
  - **HiveContext**

    An instance of the Spark SQL execution engine that integrates with data stored in Hive

- **DataFrame API**

  One use of Spark SQL is to execute SQL queries written using either a basic SQL syntax or HiveQL

DataFrame API Usage Example

Spark SQL

- **Creating a DataFrame**

```
val flightsDF = …   ← Creation from a CSV, JSON, Hive etc.
```

Example:

```
val path = "examples/flights.json"
val flightsDF = sqlContext.read.json(path)
```

- **Register as a Temporary table**

```
flightsDF.registerTempTable("flights")
```

DataFrame and SQL APIs Examples

- DataFrame API

```
flightsDF.select("Origin", "Dest", "DepDelay”)
         .filter($"DepDelay" > 15).show(5)
```

- SQL API

```
SELECT Origin, Dest, DepDelay
FROM flights
WHERE DepDelay > 15 LIMIT 5
```

Results

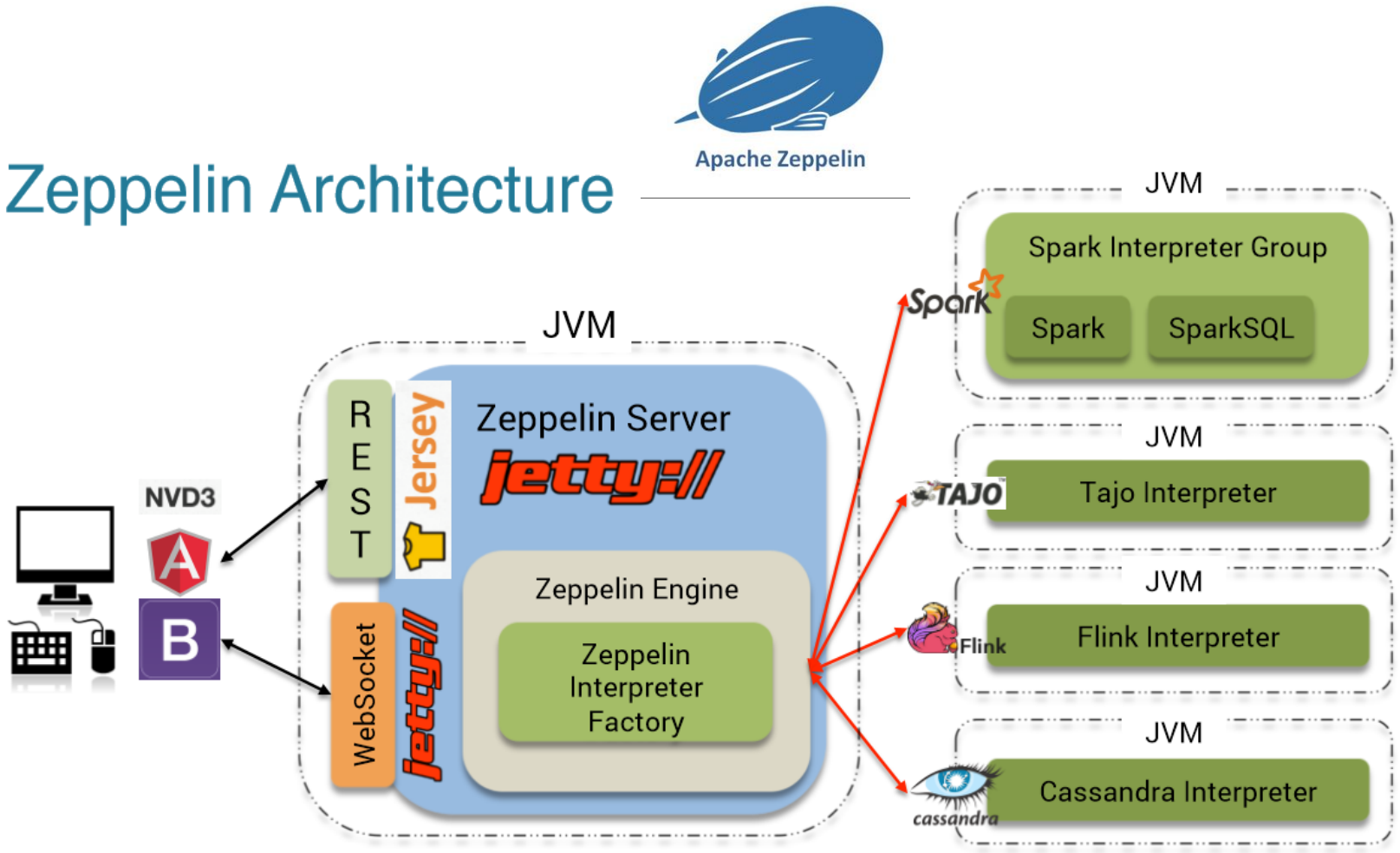
```
+------+----+--------+
|Origin|Dest|DepDelay|
+------+----+--------+
|   IAD| TPA|      19|
|   IND| BWI|      34|
|   IND| JAX|      25|
|   IND| LAS|      67|
|   IND| MCO|      94|
+------+----+--------+
```

SparkSQL : Example (Scala)

```
import org.apache.spark.sql.catalyst.encoders.ExpressionEncoder

import org.apache.spark.sql.Encoder

import spark.implicits._

val employeeDF =
spark.sparkContext.textFile("examples/src/main/resources/employee.txt")
     .map(_.split(",")).map(attributes => Employee(attributes(0),
     attributes(1).trim.toInt)).toDF()

employeeDF.createOrReplaceTempView("employee")

val youngstersDF = spark.sql("SELECT name, age FROM employee WHERE age BETWEEN 18
AND 30")

youngstersDF.map(youngster => "Name: " + youngster(0)).show()
```

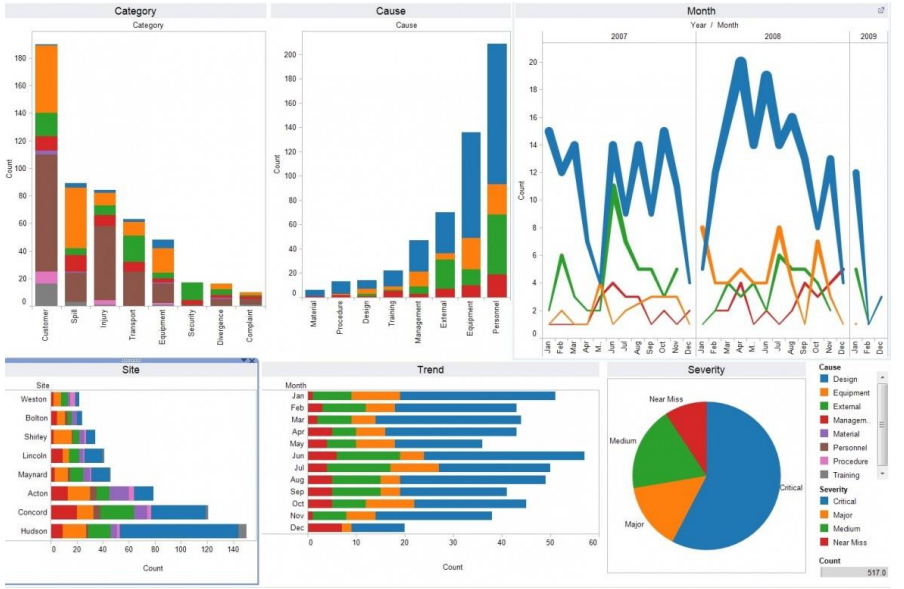
What is Apache Zeppelin?

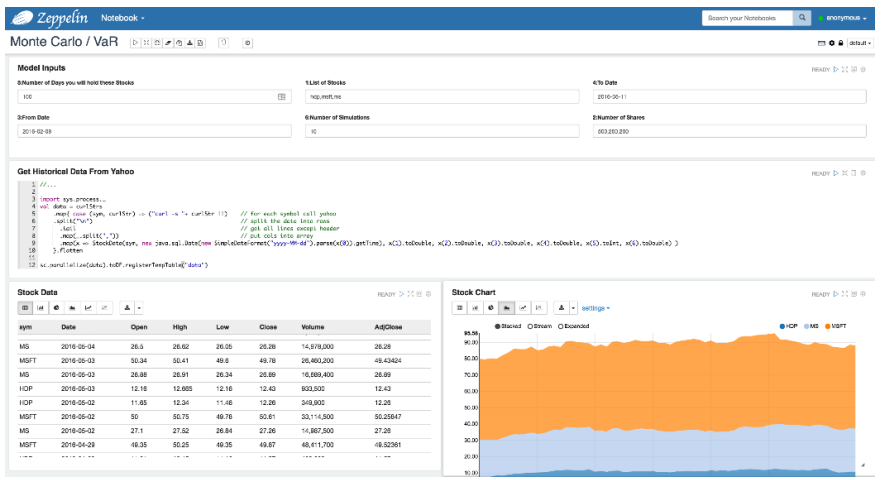
- A web-based notebook for interactive analytics
- Deeply integrated with Spark and Hadoop
- Supports multiple language backends



What is a Zeppelin Note/Notebook?

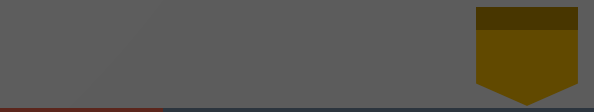
- A web based graphical user interface (GUI) for small pieces of code
- Write the code in a browser
- Zeppelin sends the code to the backend for execution and retrieves the resulting data
- Zeppelin visualizes the data
- Zeppelin Note = Set of (Paragraphs / Cells)
- Other Features - Sharing / Collaboration / Reports / Import / Export



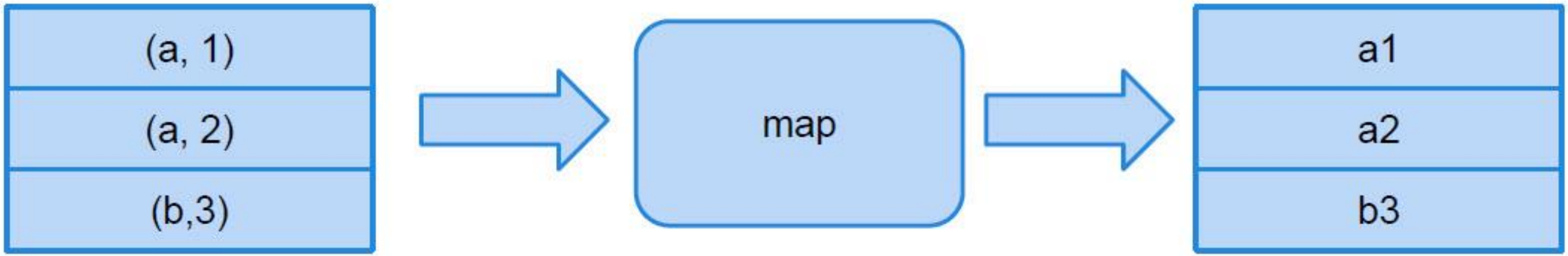


Appendices

Spark RDDs and code sample

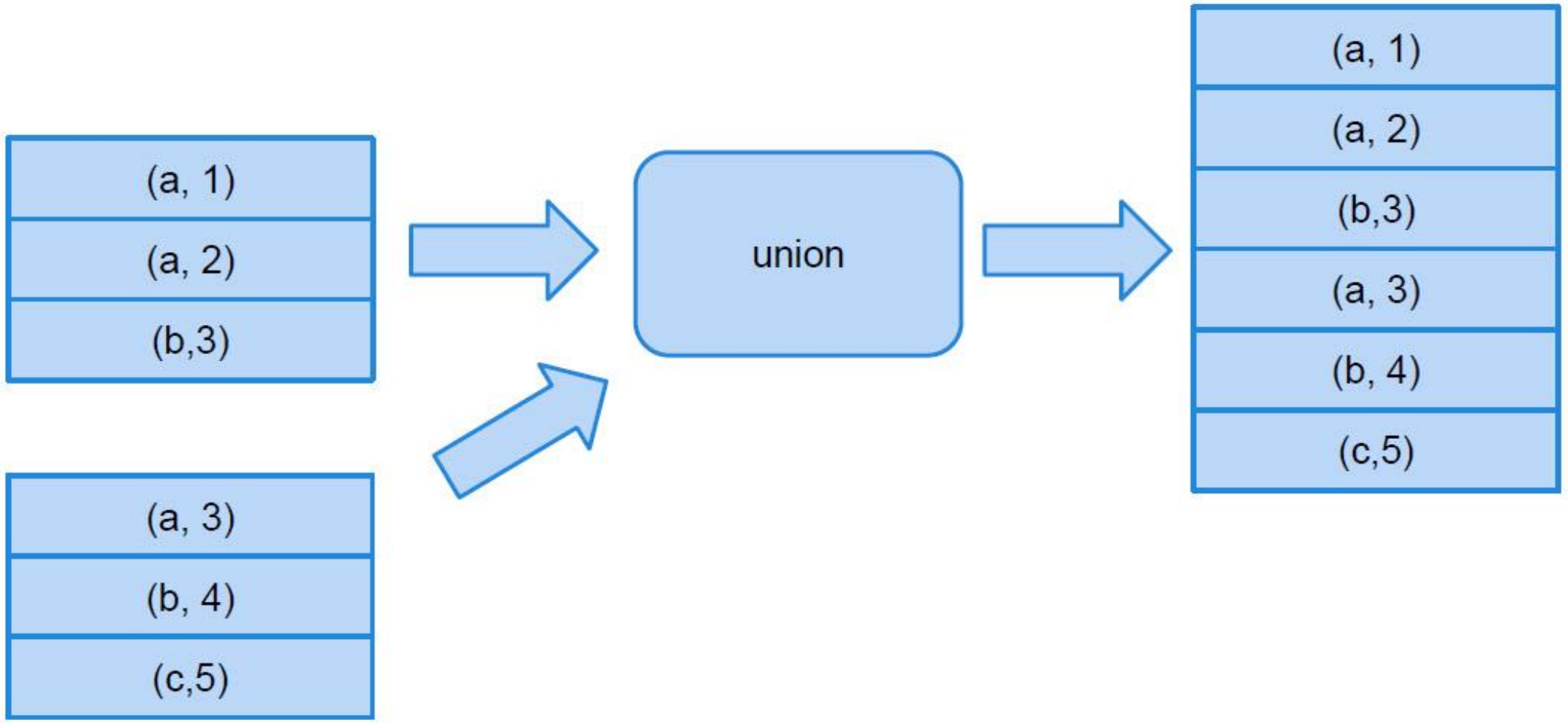


RDD : Transformation – map()

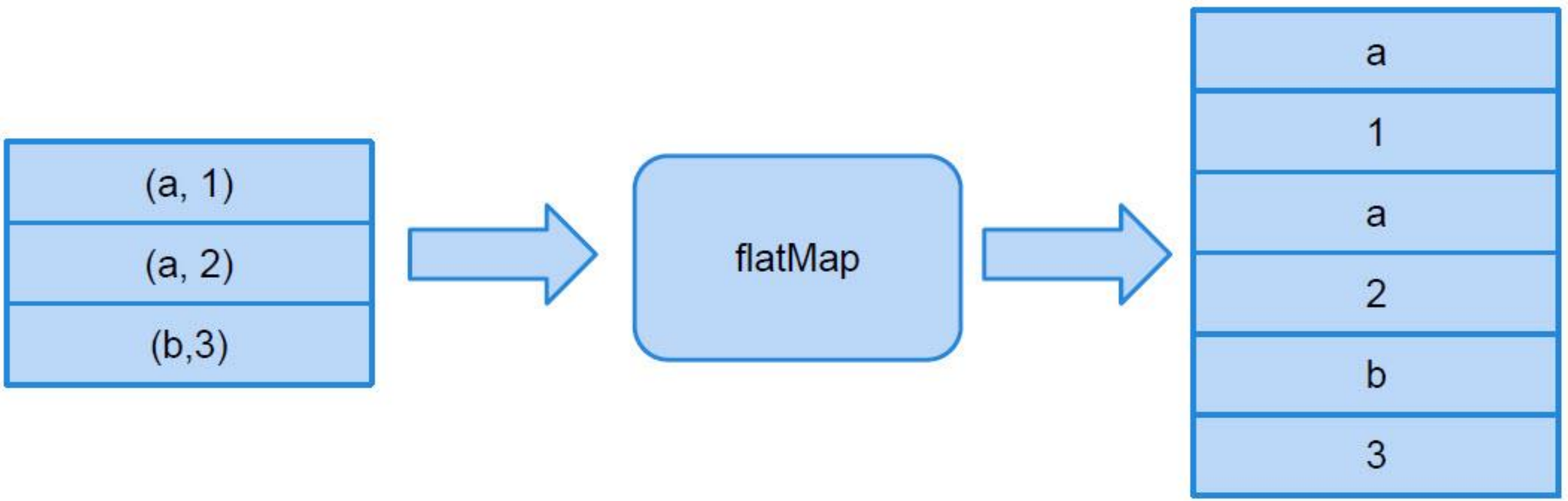


RDD : Transformation – filter()

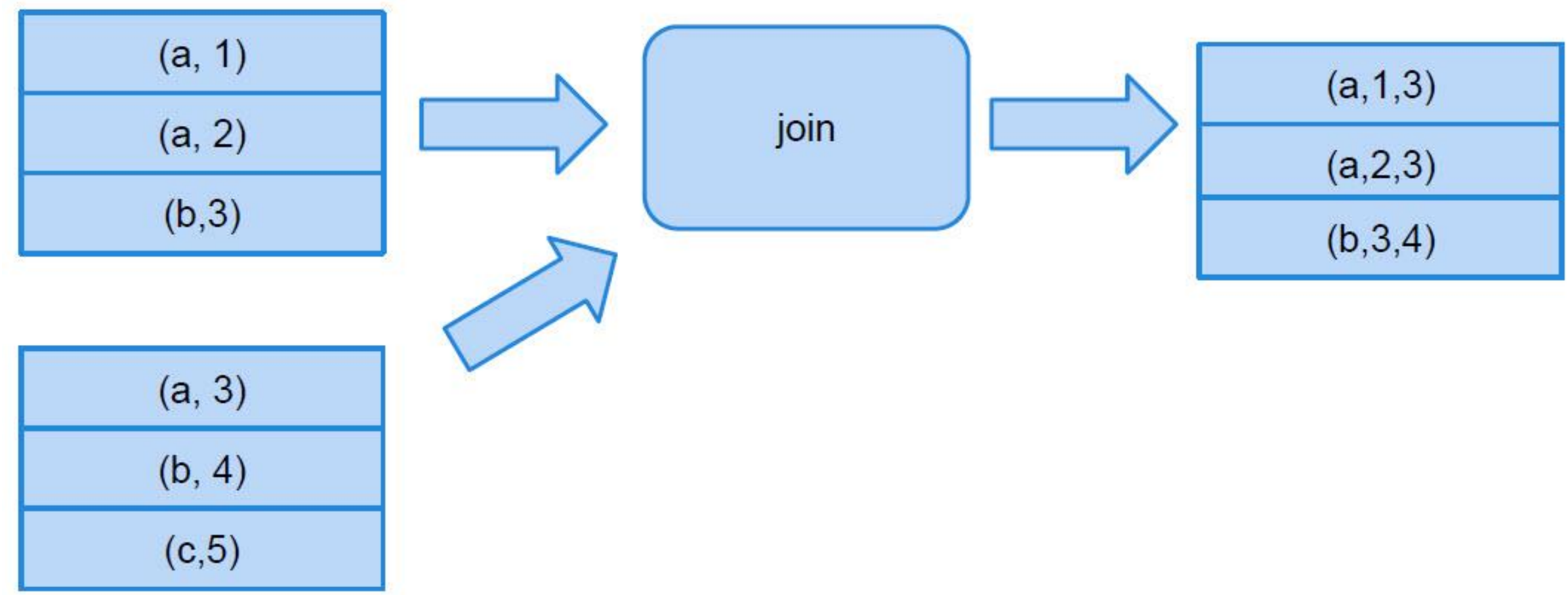
RDD : Transformation – union()



RDD : Transformation – flatmap()



RDD : Transformation – join()



Map (Python)

```
>>> a= [1,2,3]

>>> def add1(x) : return x+1

>>> map(add1, a)

    Result: [2,3,4]
```

Filter (Python)

```
>>> a= [1,2,3,4]

>>> def isOdd(x) : return x%2==1

>>> filter(isOdd, a)

    Result: [1,3]
```

Reduce (Python)

```
>>> a= [1,2,3,4]

>>> def add(x,y) : return x+y

>>> reduce(add, a)

    Result: 10
```

Lambda (Python)

```
>>> (lambda x: x + 1)(3)
      Result: 4

>>> map((lambda x: x + 1), [1,2,3])

     Result: [2,3,4]
```

WordCount in Python

```
file = spark.textFile("file://...")

counts = file.flatMap(lambda line: line.split(" "))\
              .map(lambda word: (word, 1))\
              .reduceByKey(lambda a, b: a + b)

counts.saveAsTextFile("hdfs://...")
```

RDDs : Transformations (1)

Transformation & Purpose	Example & Result
filter(func) Purpose: new RDD by selecting those data elements on which func returns true	scala> val rdd = sc.parallelize(List("ABC","BCD","DEF")) scala> val filtered = rdd.filter(_.contains("C")) scala> filtered.collect() Result: Array[String] = Array(ABC, BCD)
map(func) Purpose: return new RDD by applying func on each data element	scala> val rdd=sc.parallelize(List(1,2,3,4,5)) scala> val times2 = rdd.map(_*2) scala> times2.collect() Result: Array[Int] = Array(2, 4, 6, 8, 10)
flatMap(func) Purpose: Similar to map but func returns a Seq instead of a value. For example, mapping a sentence into a Seq of words	scala> val rdd=sc.parallelize(List("Spark is awesome","It is fun")) scala> val fm=rdd.flatMap(str=>str.split(" ")) scala> fm.collect() Result: Array[String] = Array(Spark, is, awesome, It, is, fun)

RDDs : Transformations (2)

Transformation & Purpose	Example & Result
reduceByKey(*func,[numTasks]*) **Purpose:** To aggregate values of a key using a function. "numTasks" is anoptional parameter to specify number of reduce tasks	scala> val word1=fm.map(word=>(word,1)) scala> val wrdCnt=word1.reduceByKey(_+_) scala> wrdCnt.collect() **Result:** Array[(String, Int)] = Array((is,2), (It,1), (awesome,1), (Spark,1), (fun,1))
groupByKey([*numTasks]*) **Purpose:** To convert (K,V) to (K,Iterable<V>)	scala> val cntWrd = wrdCnt.map{case (word, count) => (count, word)} scala> cntWrd.groupByKey().collect() **Result:** Array[(Int, Iterable[String])] = Array((1,ArrayBuffer(It, awesome, Spark, fun)), (2,ArrayBuffer(is)))
distinct([*numTasks]*) **Purpose:** Eliminate duplicates from RDD	scala> fm.distinct().collect() **Result:** Array[String] = Array(is, It, awesome, Spark, fun)

RDDs : Actions (1)

Transformation & Purpose	Example & Result
count() **Purpose:** Get the number of data elements in the RDD	scala> val rdd = sc.parallelize(List('A','B','C')) scala> rdd.count() **Result:** Long = 3
collect() **Purpose:** get all the data elements in an RDD as an Array	scala> val rdd = sc.parallelize(List('A','B','C')) scala> rdd.collect() **Result:** Array[Char] = Array(A, B, C)
reduce(*func*) **Purpose:** Aggregate the data elements in an RDD using this function which takes two arguments and returns one	scala> val rdd = sc.parallelize(List(1,2,3,4)) scala> rdd.reduce(_+_) **Result:** Int = 10
take (*n*) **Purpose:** fetch first n data elements in an RDD. Computed by driver program.	Scala> val rdd = sc.parallelize(List(1,2,3,4)) scala> rdd.take(2) Result: Array[Int] = Array(1, 2)

RDDs : Actions (2)

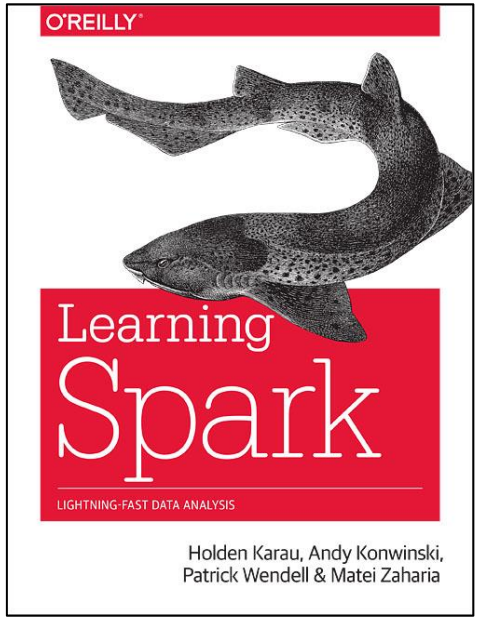
Transformation & Purpose	Example & Result
foreach(*func*) **Purpose:** execute function for each data element in RDD. Usually used to update an accumulator(discussed later) or interacting with external systems.	Scala> val rdd = sc.parallelize(List(1,2)) scala> rdd.foreach(x=>println("%s*10=%s". format(x,x*10))) **Result:** 1*10=10 2*10=20
first() **Purpose:** retrieves the first data element in RDD. Similar to take(1)	scala> val rdd = sc.parallelize(List(1,2,3,4)) scala> rdd.first() **Result:** Int = 1
saveAsTextFile(*path*) **Purpose**: Writes the content of RDD to a text file or a set of text files to local file system/HDFS	scala> val hamlet = sc.textFile("readme.txt") scala> hamlet.filter(_.contains("Spark")). saveAsTextFile("filtered") **Result:** …/filtered$ ls _SUCCESS part-00000 part-00001

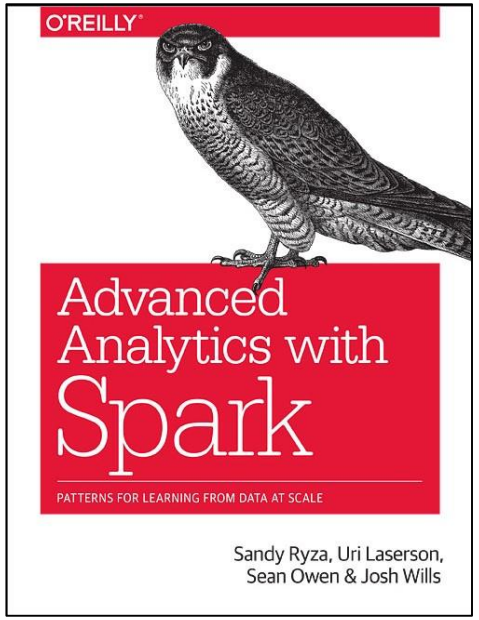
It’s time for a break

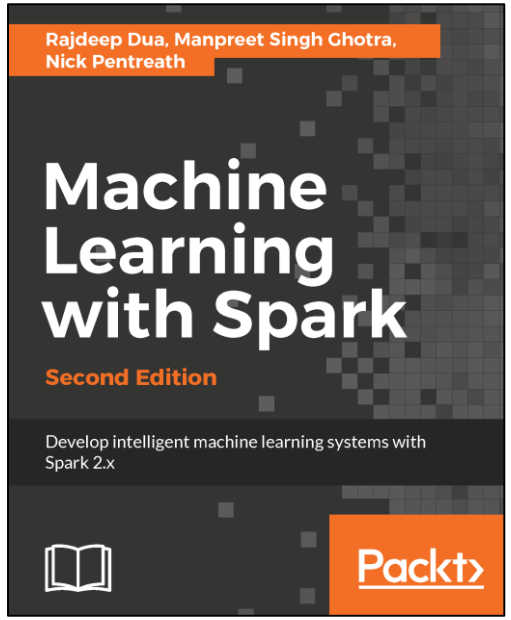
Grab some coffee, We’ll be back in 15min

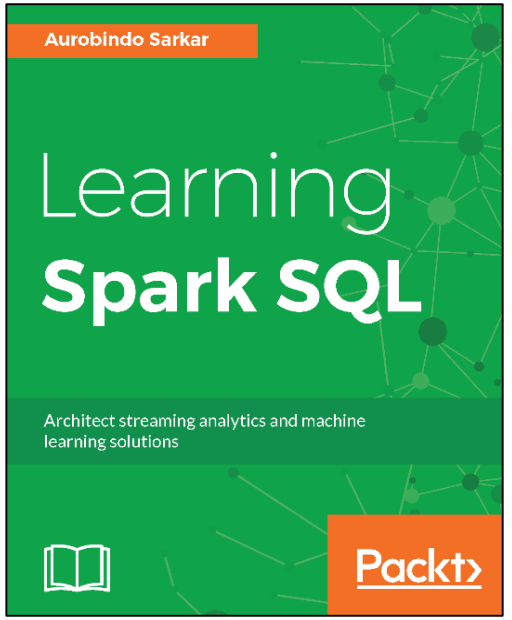

EXERCICE

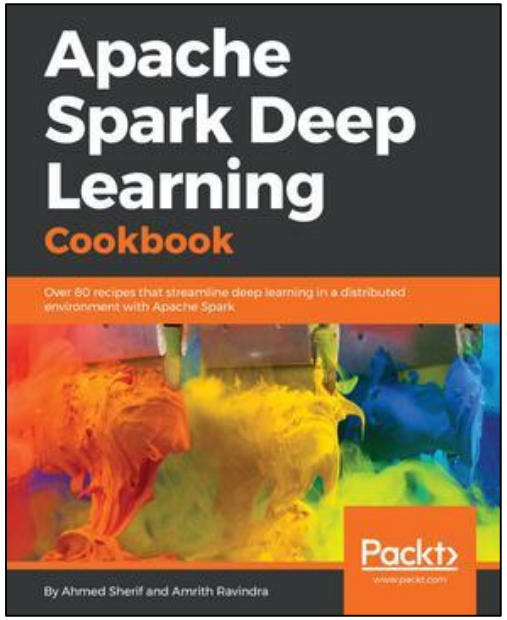
Resources











Online Resources

http://spark.apache.org/

Thank You