CHULALONGKORN
BUSINESS SCHOOL
FLAGSHIP FOR LIFE

Thaveewat Khanan

Introduction

CHULALONGKORN BUSINESS SCHOOL
FLAGSHIP FOR LIFE

A fast and general engine for large scale data processing

An open source big data processing framework built around speed, ease of use, and sophisticated analytics. Spark enables applications in Hadoop clusters to run up to 100 times faster in memory and 10 times faster even when running on disk.

What is Spark ?

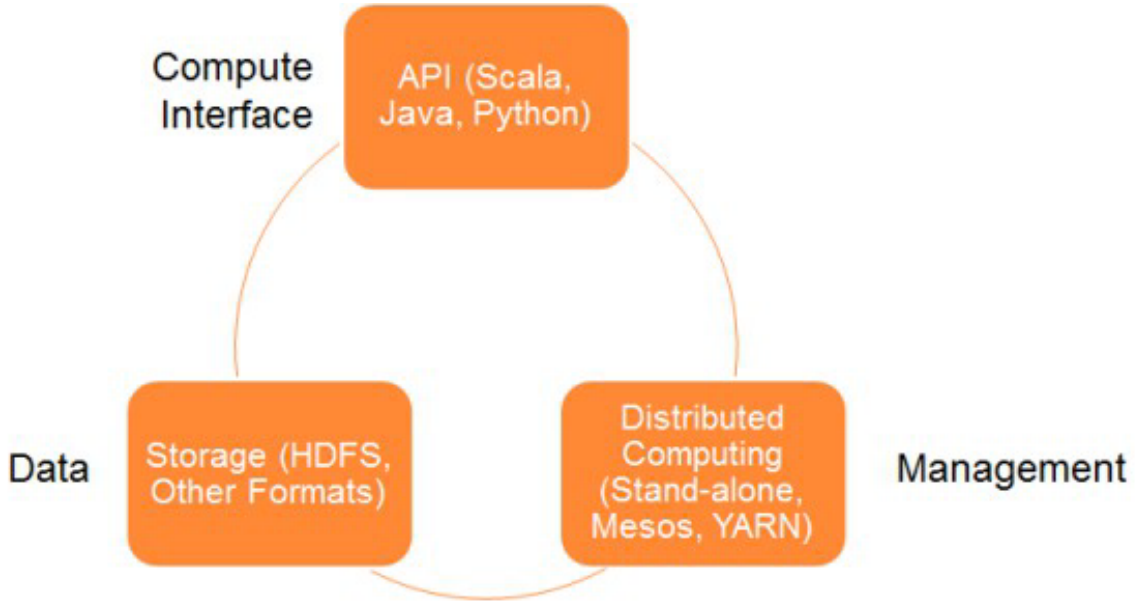
CHULALONGKORN BUSINESS SCHOOL
FLAGSHIP FOR LIFE

Framework for distributed processing.

In-memory, fault tolerant data structures

Flexible APIs in Scala, Java, Python, SQL, R

Open source

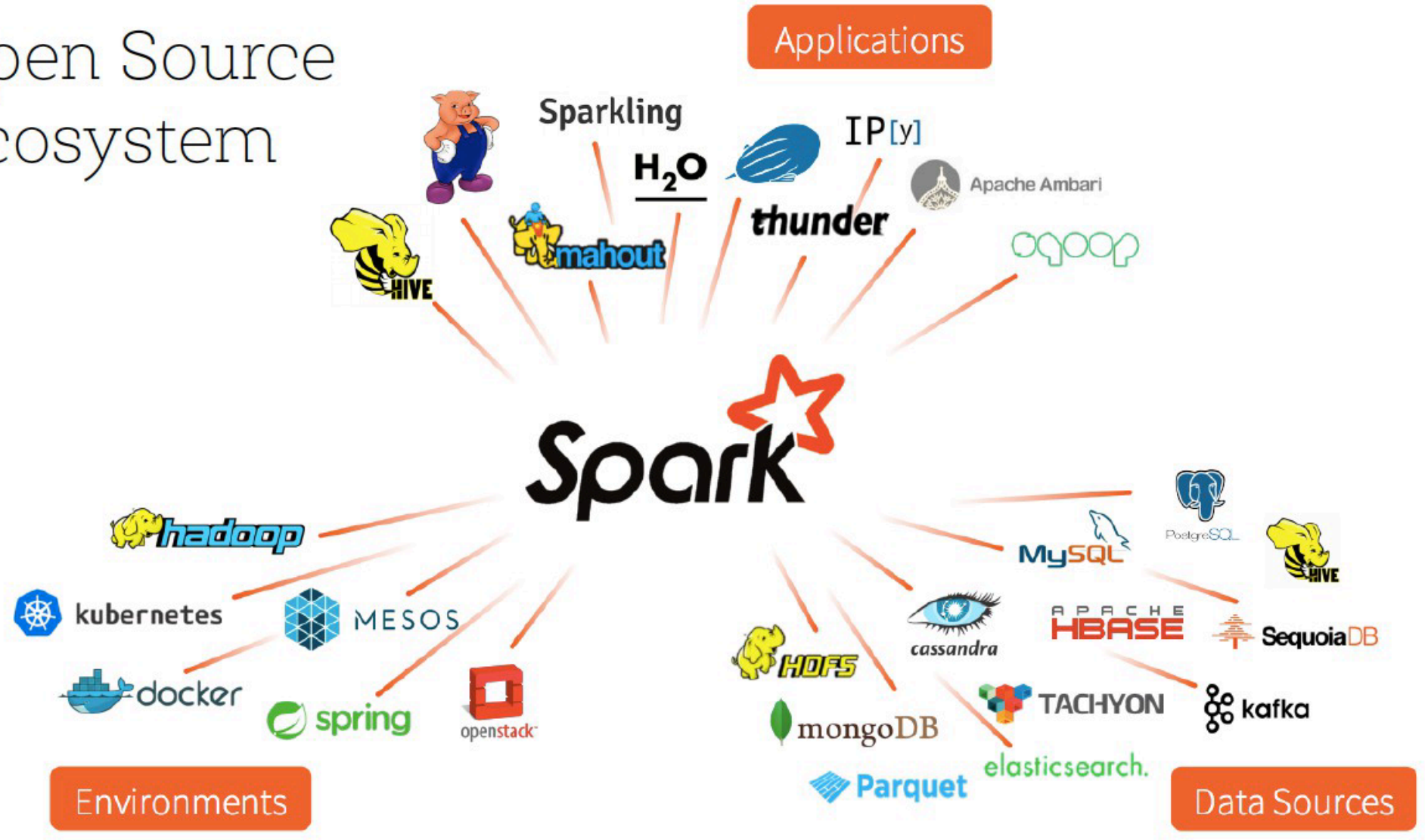


Why Spark ?

CHULALONGKORN BUSINESS SCHOOL
FLAGSHIP FOR LIFE

- Handle Petabytes of data
- Significant faster than MapReduce
- Simple and intutive APIs
- General framework
  - Runs anywhere
  - Handles (most) any I/O
  - Interoperable libraries for specific use-cases

Open Source Ecosystem

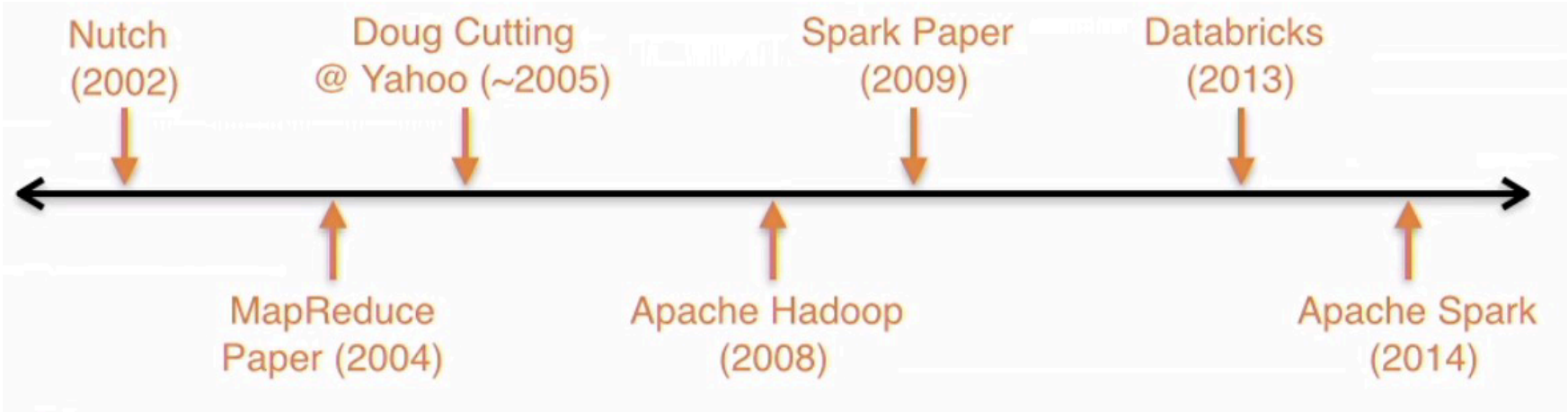


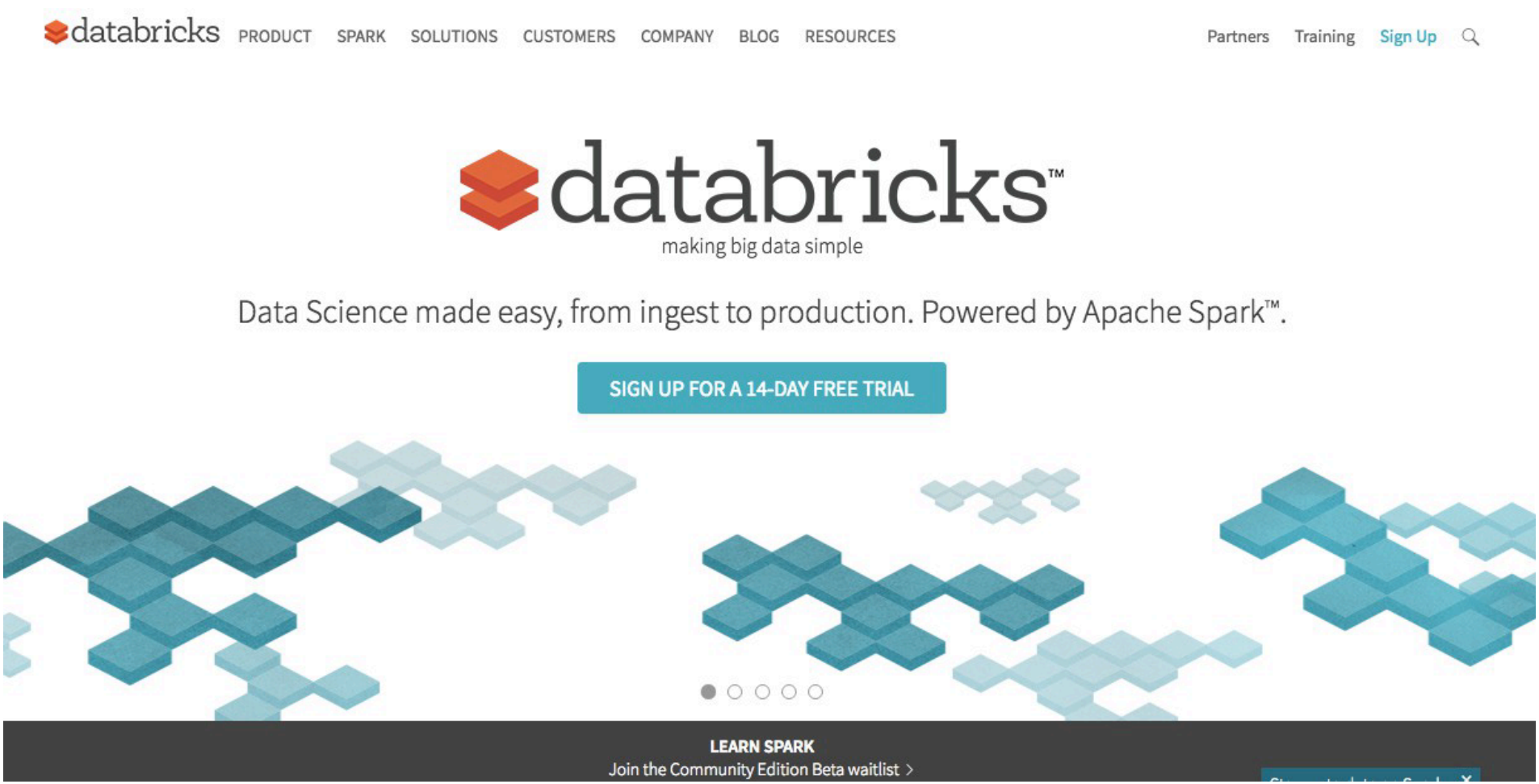
Source: Jump start into Apache Spark and Databricks

Spark: History

CHULALONGKORN BUSINESS SCHOOL
FLAGSHIP FOR LIFE

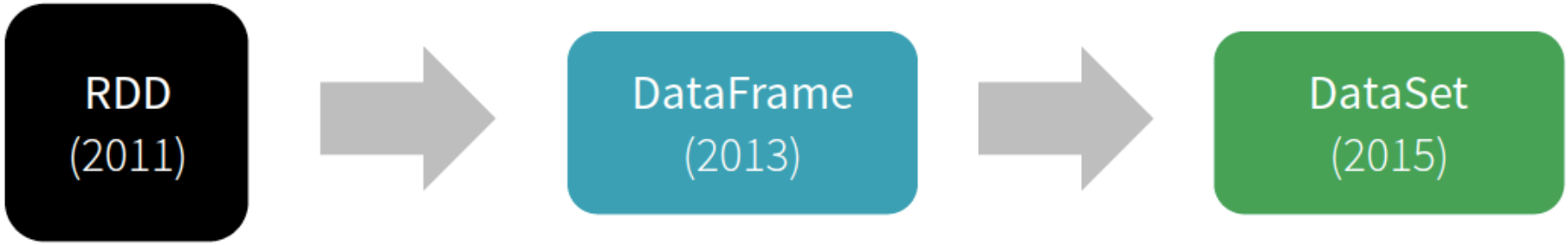
- Founded by AMPlab, UC Berkeley
- Created by Matei Zaharia (PhD Thesis)
- Maintained by Apache Software Foundation
- Commercial support by Databricks




databricks
PRODUCT
SPARK
SOLUTIONS
CUSTOMERS
COMPANY
BLOG
RESOURCES
Partners
Training
Sign Up
databricks™
making big data simple
Data Science made easy, from ingest to production. Powered by Apache Spark™.
SIGN UP FOR A 14-DAY FREE TRIAL
LEARN SPARK
Join the Community Edition Beta waitlist >

History of Spark APIs

CHULALONGKORN BUSINESS SCHOOL
FLAGSHIP FOR LIFE



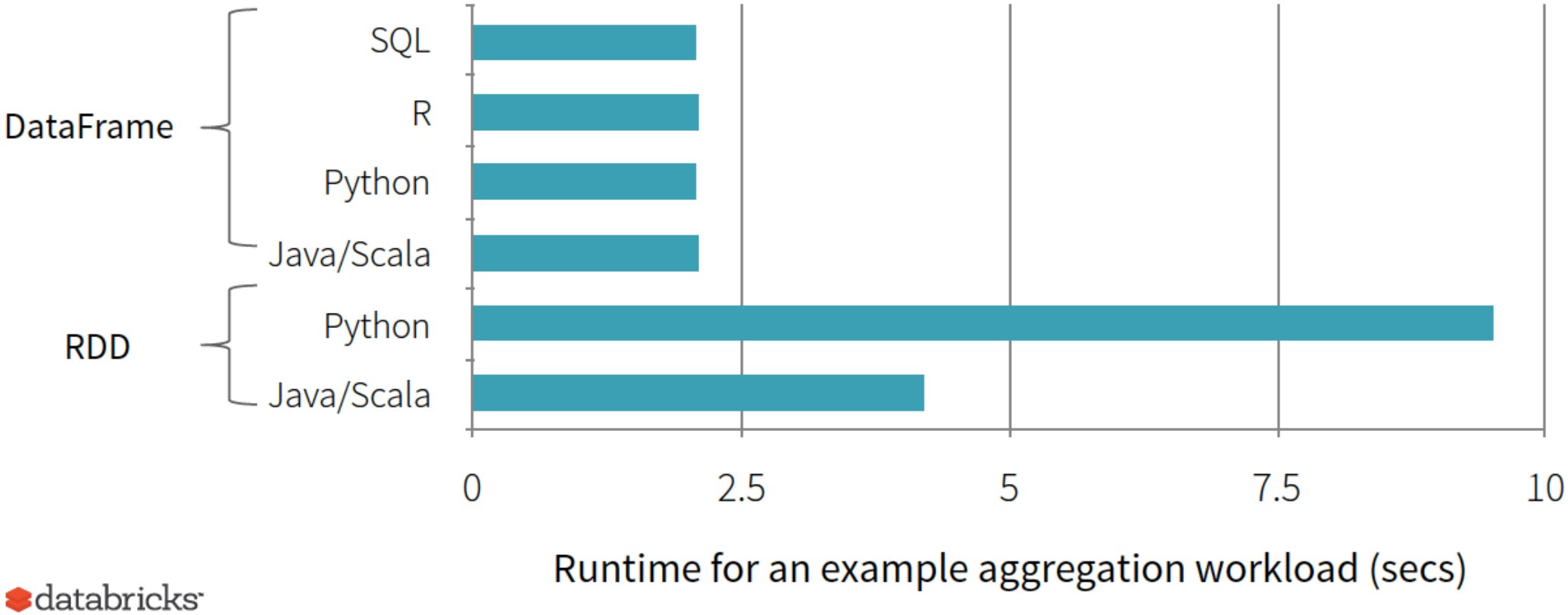
- Distribute collection of JVM objects
- Functional Operators (map, filter, etc.)

- Distribute collection of Row objects
- Expression-based operations and UDFs
- Logical plans and optimizer
- Fast/efficient internal representations

- Internally rows, externally JVM objects
- "Best of both worlds" **type safe + fast**

Source: Jump start into Apache Spark and Databricks

Benefit of Logical Plan: Performance Parity Across Languages



databricks

Source: Jump start into Apache Spark and Databricks

What is a RDD ?

CHULALONGKORN BUSINESS SCHOOL
FLAGSHIP FOR LIFE

Resilient: if the data in memory (or on a node) is lost, it can be recreated.

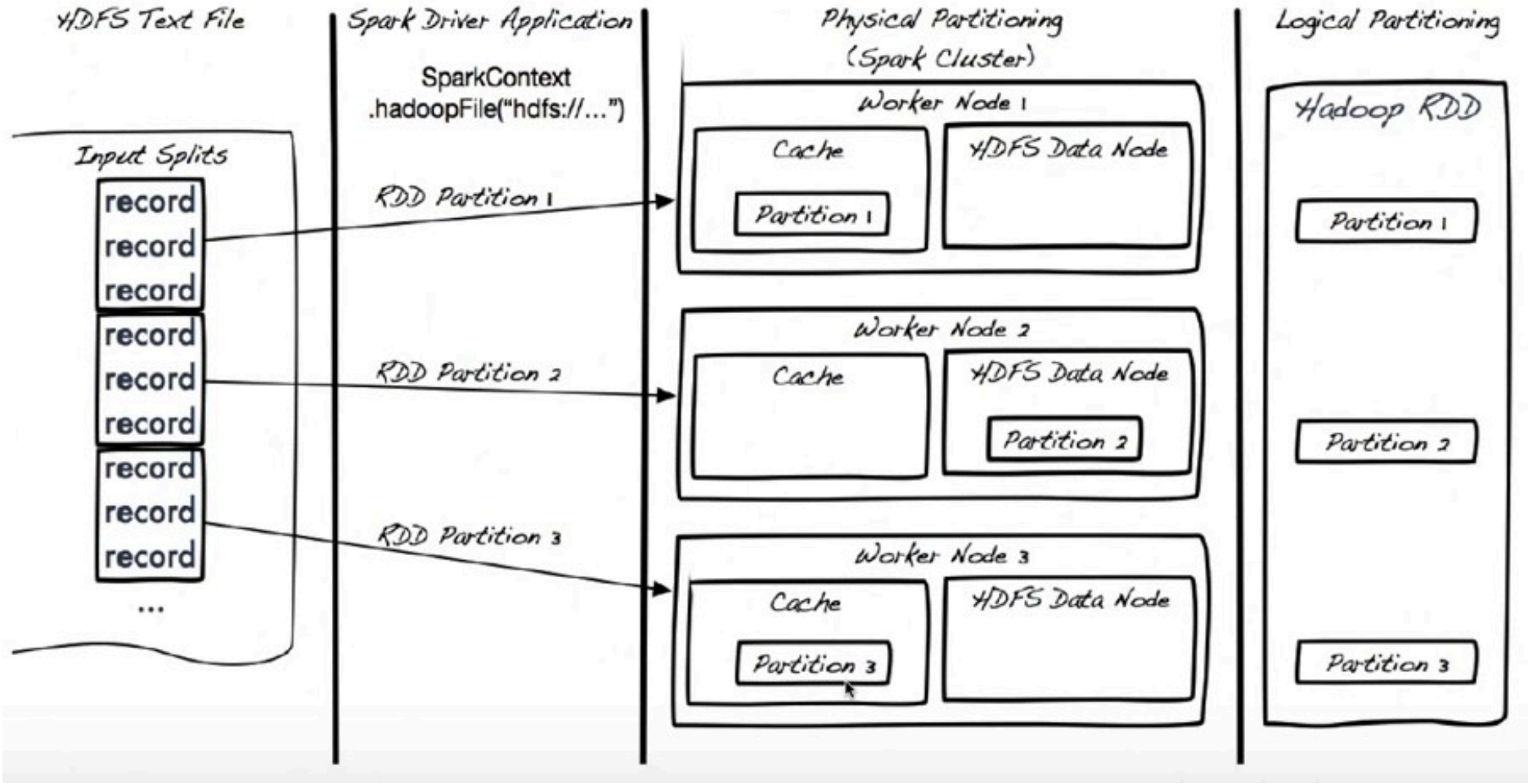
Distributed: data is chucked into partitions and stored in memory acress the custer.

Dataset: initial data can come from a table or be created programmatically

RDD Creation

CHULALONGKORN BUSINESS SCHOOL
FLAGSHIP FOR LIFE

hdfsData = sc.textFile("hdfs://data.txt")



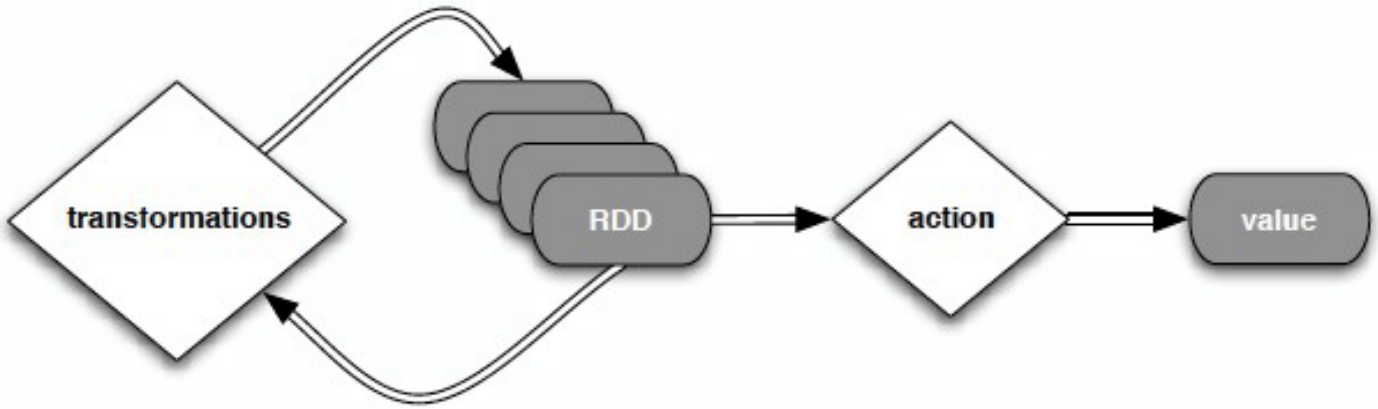
Source: Pspark: A brain-friendly introduction

RDD: Operations

CHULALONGKORN BUSINESS SCHOOL
FLAGSHIP FOR LIFE

Transformations: transformations are lazy (not computed immediately)

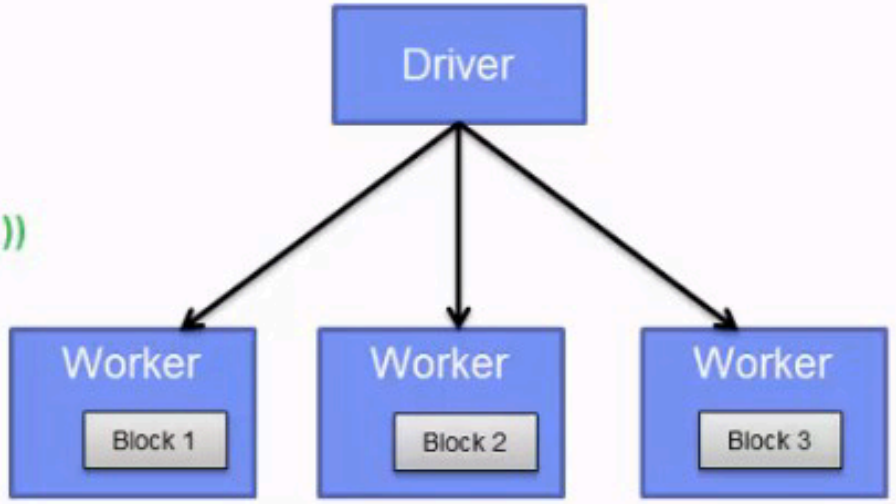
Actions: the transformed RDD gets recomputed when an action is run on it (default)



What happens when an action is executed

CHULALONGKORN BUSINESS SCHOOL
FLAGSHIP FOR LIFE

```
// Creating the RDD
val logFile = sc.textFile("hdfs://...")
// Transformations
val errors = logFile.filter(_.startsWith("ERROR"))
val messages = errors.map(_.split("\t")).map(r => r(1))
//Cache
messages.cache()
// Actions
messages.filter(_.contains("mysql")).count()
messages.filter(_.contains("php")).count()
```

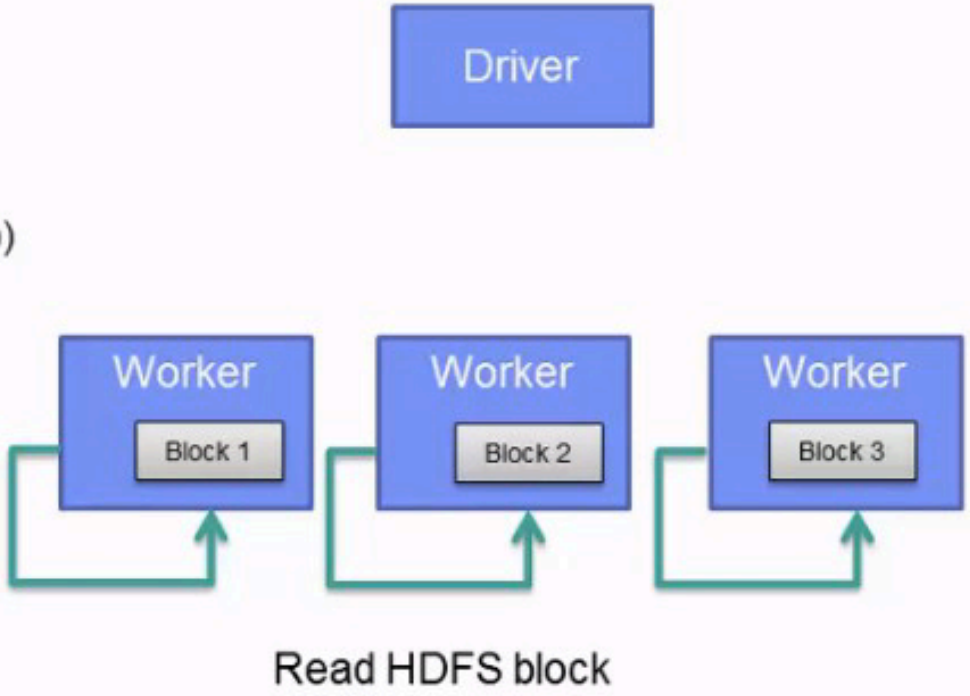


Driver sends the code to be executed on each block

Source: Spark Fundamentals I Big Data Usibersity

What happens when an action is executed


```
// Creating the RDD
val logFile = sc.textFile("hdfs://...")
// Transformations
val errors = logFile.filter(_.startsWith("ERROR"))
val messages = errors.map(_.split("\t")).map(r => r(1))
//Caching
messages.cache()
// Actions
messages.filter(_.contains("mysql")).count()
messages.filter(_.contains("php")).count()
```

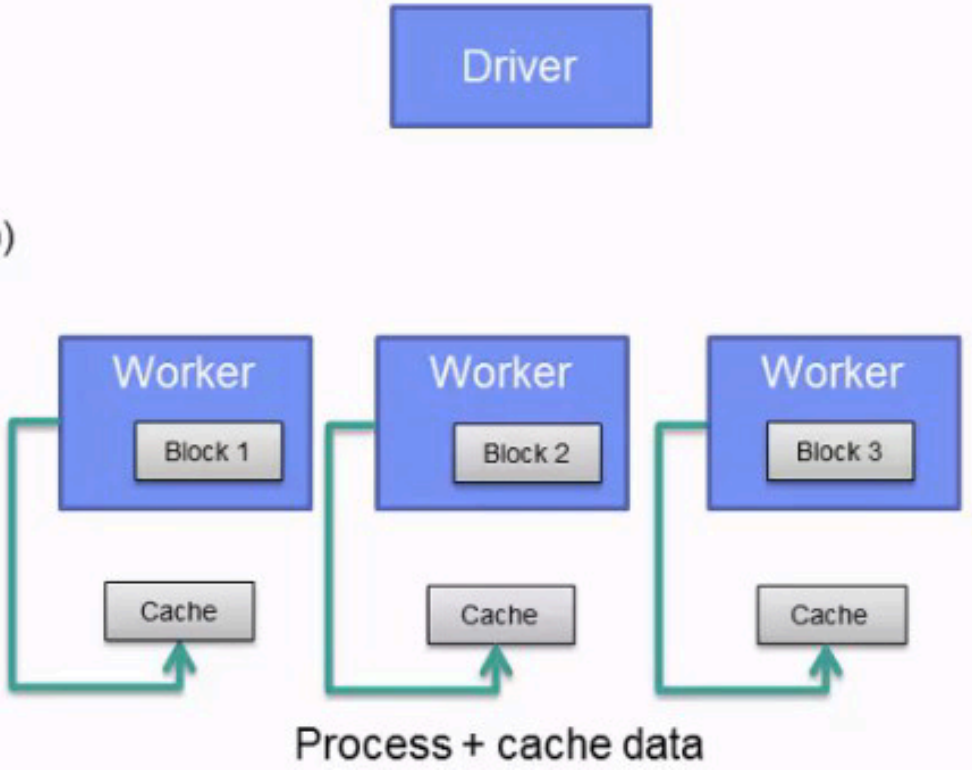


Source: Spark Fundamentals I Big Data Usibersity

What happens when an action is executed

CHULALONGKORN BUSINESS SCHOOL
FLAGSHIP FOR LIFE

```
// Creating the RDD
val logFile = sc.textFile("hdfs://...")
// Transformations
val errors = logFile.filter(_.startsWith("ERROR"))
val messages = errors.map(_.split("\t")).map(r => r(1))
//Caching
messages.cache()
// Actions
messages.filter(_.contains("mysql")).count()
messages.filter(_.contains("php")).count()
```

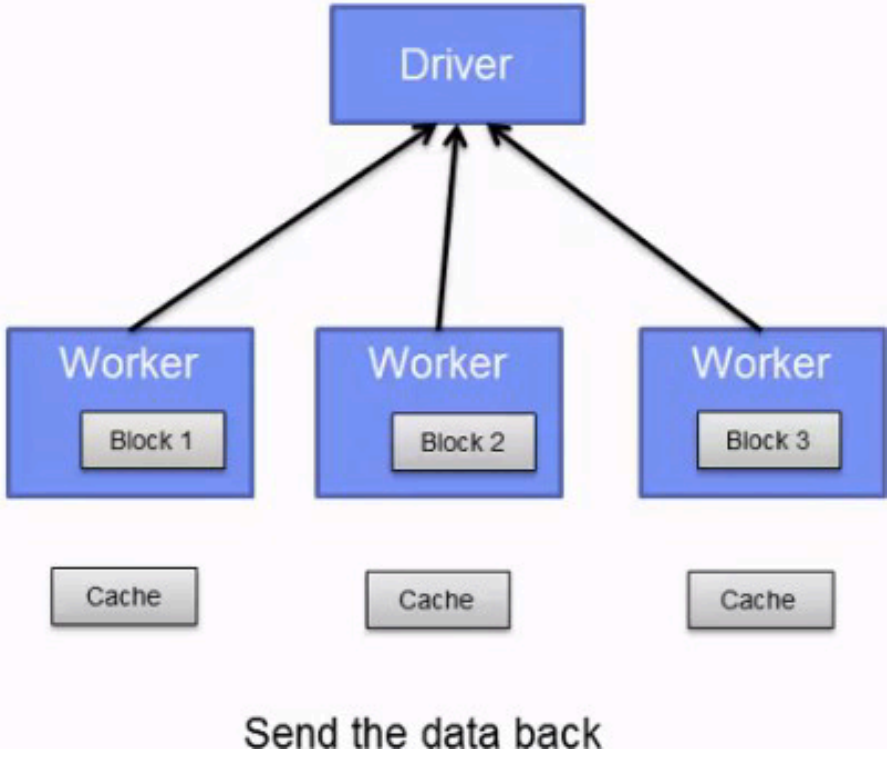


Source: Spark Fundamentals I Big Data Usibersity

What happens when an action is executed

CHULALONGKORN BUSINESS SCHOOL
FLAGSHIP FOR LIFE

```
// Creating the RDD
val logFile = sc.textFile("hdfs://...")
// Transformations
val errors = logFile.filter(_.startsWith("ERROR"))
val messages = errors.map(_.split("\t")).map(r => r(1))
//Caching
messages.cache()
// Actions
messages.filter(_.contains("mysql")).count()
messages.filter(_.contains("php")).count()
```

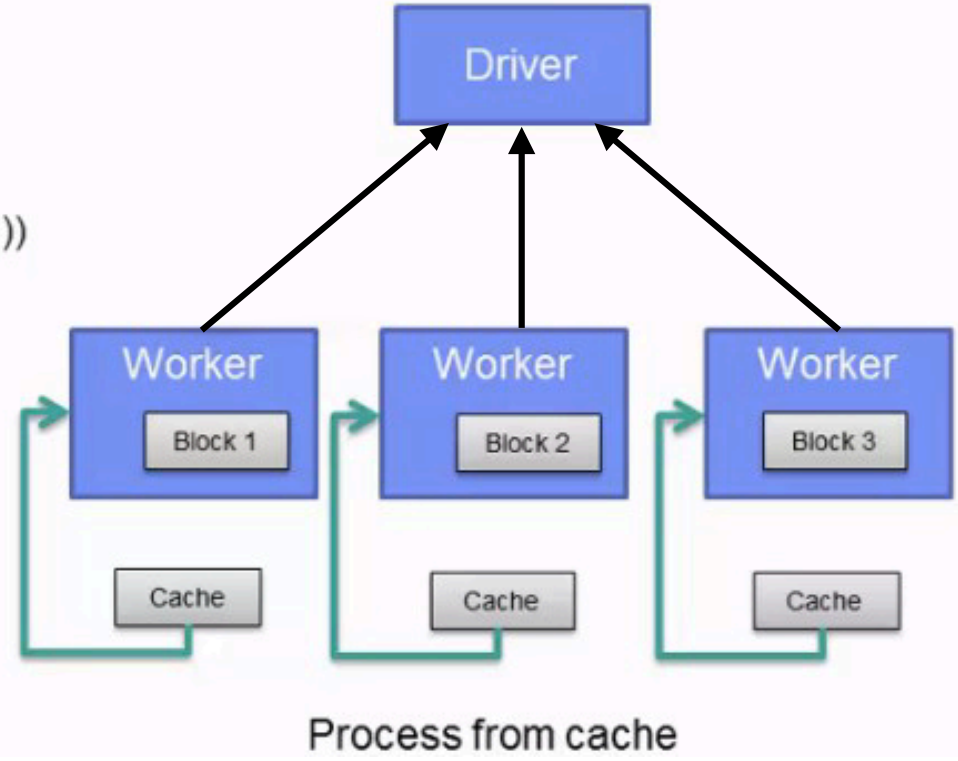


Source: Spark Fundamentals I Big Data Usibersity

What happens when an action is executed

CHULALONGKORN BUSINESS SCHOOL
FLAGSHIP FOR LIFE

```
// Creating the RDD
val logFile = sc.textFile("hdfs://...")
// Transformations
val errors = logFile.filter(_.startsWith("ERROR"))
val messages = errors.map(_.split("\t")).map(r => r(1))
//Caching
messages.cache()
// Actions
messages.filter(_.contains("mysql")).count()
messages.filter(_.contains("php")).count()
```



Source: Spark Fundamentals I Big Data Usibersity

What happens when an action is executed

CHULALONGKORN BUSINESS SCHOOL
FLAGSHIP FOR LIFE

```
16/11/01 01:03:54 INFO storage.BlockManagerMaster: Registered BlockManager
Welcome to
      ____              __
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ `/ __/  '_/
   /__ / .__/\_,_/_/ /_/\_\   version 1.6.0
      /_/

Using Python version 2.6.6 (r266:84292, Jul 23 2015 15:22:56)
SparkContext available as sc, HiveContext available as sqlContext.
>>> rdd1 = sc.parallelize(range(1, 11))
>>> rdd1 = rdd1.filter(lambda n : n % 2 == 0)
>>> rdd1 = rdd1.map(lambda n : n ** 2)
>>> rdd1 = rdd1.map(lambda n : n / 2)
>>> ans = rdd1.reduce(lambda a, b : a + b)
16/11/01 01:04:52 INFO scheduler.DAGScheduler: ResultStage 0 (reduce at <stdin>:
1) finished in 0.360 s
16/11/01 01:04:52 INFO scheduler.TaskSetManager: Finished task 0.0 in stage 0.0
(TID 0) in 329 ms on localhost (1/1)
16/11/01 01:04:52 INFO scheduler.TaskSchedulerImpl: Removed TaskSet 0.0, whose t
asks have all completed, from pool
16/11/01 01:04:52 INFO scheduler.DAGScheduler: Job 0 finished: reduce at <stdin>
:1, took 0.990547 s
>>> print(ans)
110
```

[1,2,3,4,5,6,7,8,9,10]
[2,4,6,8,10]
[4, 16, 36, 64, 100]
[2, 8, 18, 32, 50]

Source: Spark Fundamentals I Big Data Usibersity

Spark:Transformation

CHULALONGKORN BUSINESS SCHOOL
FLAGSHIP FOR LIFE

transformation	*description*
map(*func*)	return a new distributed dataset formed by passing each element of the source through a function *func*
filter(*func*)	return a new dataset formed by selecting those elements of the source on which *func* returns true
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)
sample(*withReplacement, fraction, seed*)	sample a fraction *fraction* of the data, with or without replacement, using a given random number generator *seed*
union(*otherDataset*)	return a new dataset that contains the union of the elements in the source dataset and the argument
distinct([*numTasks*]))	return a new dataset that contains the distinct elements of the source dataset

Spark:Transformation

CHULALONGKORN BUSINESS SCHOOL
FLAGSHIP FOR LIFE

transformation	description
groupByKey([*numTasks*])	when called on a dataset of (K, V) pairs, returns a dataset of (K, Seq[V]) pairs
reduceByKey(*func*, [*numTasks*])	when called on a dataset of (K, V) pairs, returns a dataset of (K, V) pairs where the values for each key are aggregated using the given reduce function
sortByKey([*ascending*], [*numTasks*])	when called on a dataset of (K, V) pairs where K implements Ordered, returns a dataset of (K, V) pairs sorted by keys in ascending or descending order, as specified in the boolean ascending argument
join(*otherDataset*, [*numTasks*])	when called on datasets of type (K, V) and (K, W), returns a dataset of (K, (V, W)) pairs with all pairs of elements for each key
cogroup(*otherDataset*, [*numTasks*])	when called on datasets of type (K, V) and (K, W), returns a dataset of (K, Seq[V], Seq[W]) tuples – also called groupWith
cartesian(*otherDataset*)	when called on datasets of types T and U, returns a dataset of (T, U) pairs (all pairs of elements)

Single RDD Transformation

CHULALONGKORN BUSINESS SCHOOL
FLAGSHIP FOR LIFE

filter females to analyze female buying patterns

```
male1, male2, female1 -> female1
```

map squared values

```
2, 5, 6 -> 4, 25, 36
```

flatMap to break up a sentence into words

```
my name is ray -> my, name, is, ray
```

find the distinct values in a dataset

```
apple, apple, banana -> apple, banana
```

sample two values at random

```
apple, banana, guava -> banana, apple
```

Source: Jspark: A brain-friendly introduction

Multiple RDD Transformation

CHULALONGKORN BUSINESS SCHOOL
FLAGSHIP FOR LIFE

union

```
apple, orange, banana, guava,
banana, pear
```

intersection

```
banana
```

subtract anything shown in Dataset B from Dataset A

```
apple, orange
```

cartesian (every possible pair combo)

```
(apple, guava), (apple, banana), ...
```

Dataset A

apple
orange
banana

Dataset B

guava
banana
pear

Source: Jspark: A brain-friendly introduction

Pair RDD Transformation

CHULALONGKORN BUSINESS SCHOOL
FLAGSHIP FOR LIFE

- reduceByKey
- groupByKey
- combineByKey
- mapValues
- flatMapValues
- keys
- values
- subtractByKey
- join
- rightOuterJoin
- leftOuterJoin
- cogroup
- sortByKey

Source: Jspark: A brain-friendly introduction

Spark:Actions

CHULALONGKORN BUSINESS SCHOOL
FLAGSHIP FOR LIFE

action	description
`reduce(func)`	aggregate the elements of the dataset using a function *func* (which takes two arguments and returns one), and should also be commutative and associative so that it can be computed correctly in parallel
`collect()`	return all the elements of the dataset as an array at the driver program – usually useful after a filter or other operation that returns a sufficiently small subset of the data
`count()`	return the number of elements in the dataset
`first()`	return the first element of the dataset – similar to *take(1)*
`take(n)`	return an array with the first *n* elements of the dataset – currently not executed in parallel, instead the driver program computes all the elements
`takeSample(withReplacement, fraction, seed)`	return an array with a random sample of *num* elements of the dataset, with or without replacement, using the given random number generator seed

Spark:Actions

CHULALONGKORN BUSINESS SCHOOL
FLAGSHIP FOR LIFE

action	description
saveAsTextFile(*path*)	write 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 will call `toString` on each element to convert it to a line of text in the file
saveAsSequenceFile(*path*)	write the elements of the dataset as a Hadoop `SequenceFile` in a given path in the local filesystem, HDFS or any other Hadoop-supported file system. Only available on RDDs of key-value pairs that either implement Hadoop's `Writable` interface or are implicitly convertible to `Writable` (Spark includes conversions for basic types like `Int`, `Double`, `String`, etc).
countByKey()	only available on RDDs of type `(K, V)`. Returns a `Map` of `(K, Int)` pairs with the count of each key
foreach(*func*)	run a function *func* on each element of the dataset – usually done for side effects such as updating an accumulator variable or interacting with external storage systems

Transformations

```
>>> nums = sc.parallelize([1,2,3])
>>> squared = nums.map(lambda x : x*x)
>>> even = squared.filter(lambda x: x%2 == 0)
>>> evens = nums.flatMap(lambda x: range(x))
```

Actions

```
>>> nums = sc.parallelize([1,2,3])
>>> nums.collect()
>>> nums.take(2)
>>> nums.count()
>>> nums.reduce(lambda:x, y:x+y)
>>> nums.saveAsTextFile("hdfs:///user/cloudera/output/test”)
```

Spark Programming

CHULALONGKORN BUSINESS SCHOOL
FLAGSHIP FOR LIFE

Functional tools in Python

- map
- filter
- reduce
- lambda
- IterTools
  - Chain, flatmap

map

```
>>> a= [1,2,3]
>>> def add1(x) : return x+1
>>> map(add1, a)
```

Result: [2,3,4]

```
val input = sc.parallelize(List(1,2,3,4))
val result = input.map(x => x*x)
println(result.collect().mkString(","))
```

Result: [1,4,9,16]

Filter

```
>>> a= [1,2,3,4]
>>> def isOdd(x) : return x%2==1
>>> filter(isOdd, a)
Result: [1,3]
```

Reduce

```
>>> a= [1,2,3,4]
>>> def add(x,y) : return x+y
>>> reduce(add, a)
Result: 10
```

lambda

>>> (lambda x: x + 1)(3)

Result: 4

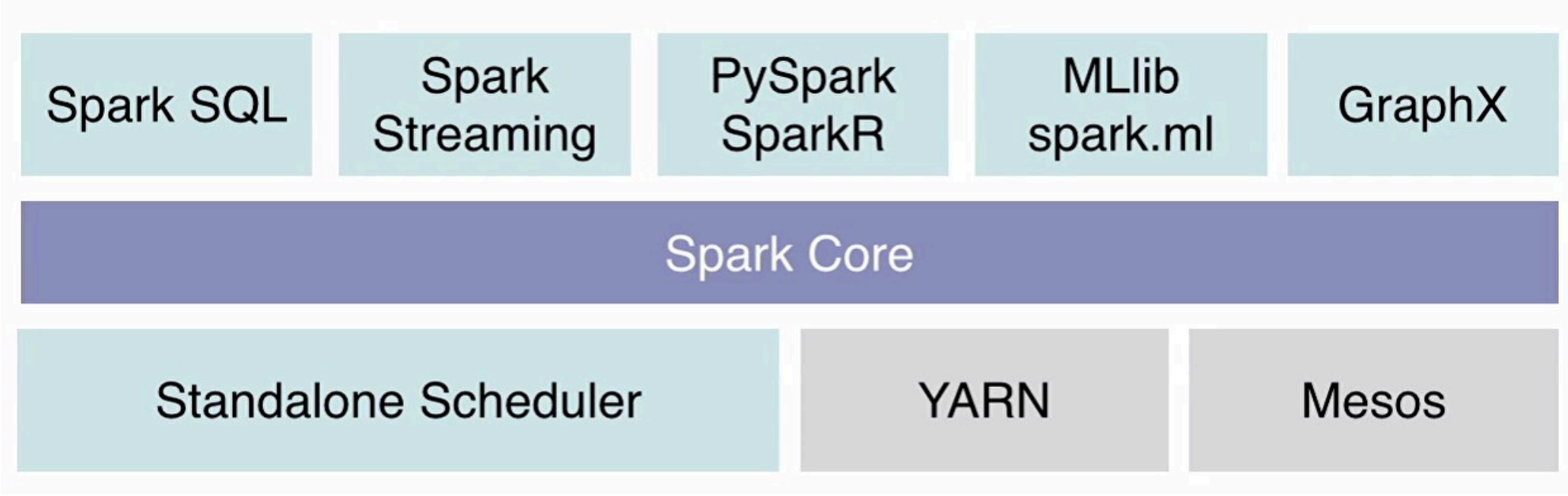
>>> map((lambda x: x + 1), [1,2,3])

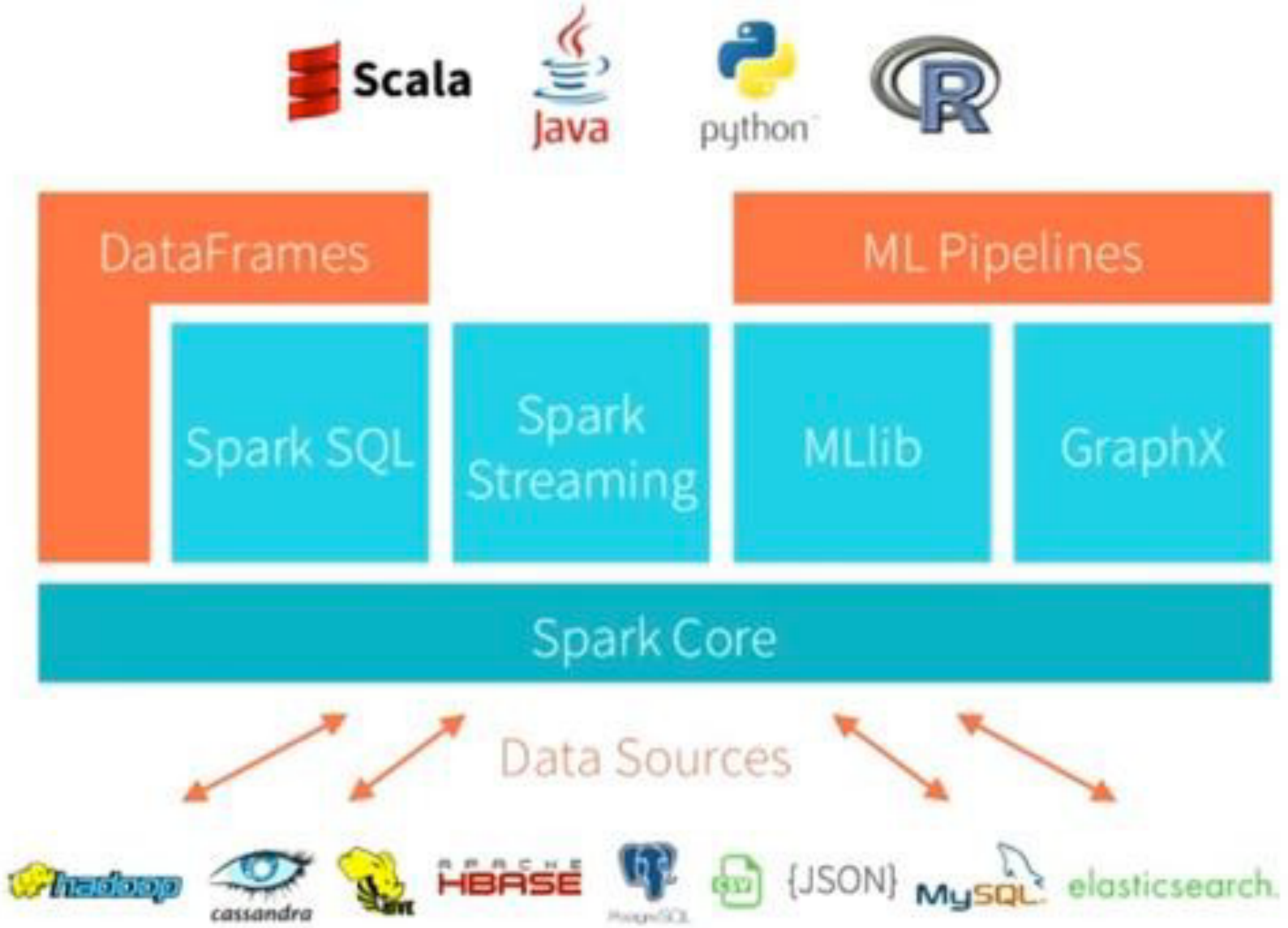
Result: [2,3,4]

Exercises

- (lambda x: 2*x)(3) => ?
- map(lambda x: 2*x, [1,2,3]) =>
- map(lambda t: t[0], [(1,2), (3,4), (5,6)]) =>
- reduce(lambda x,y: x+y, [1,2,3]) =>
- reduce(lambda x,y: x+y, map(lambda t: t[0], [(1,2), (3,4), (5,6)]))=>

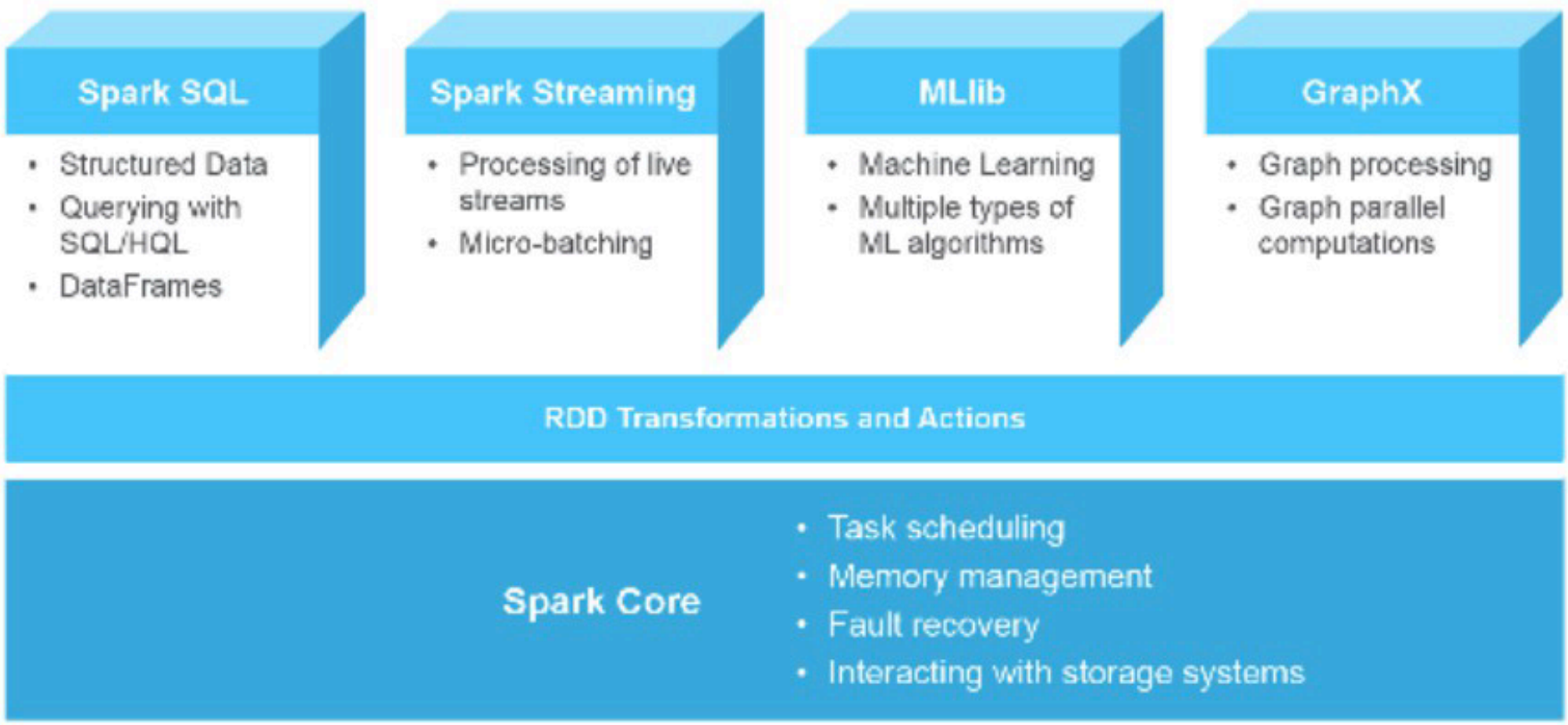
Spark Platform



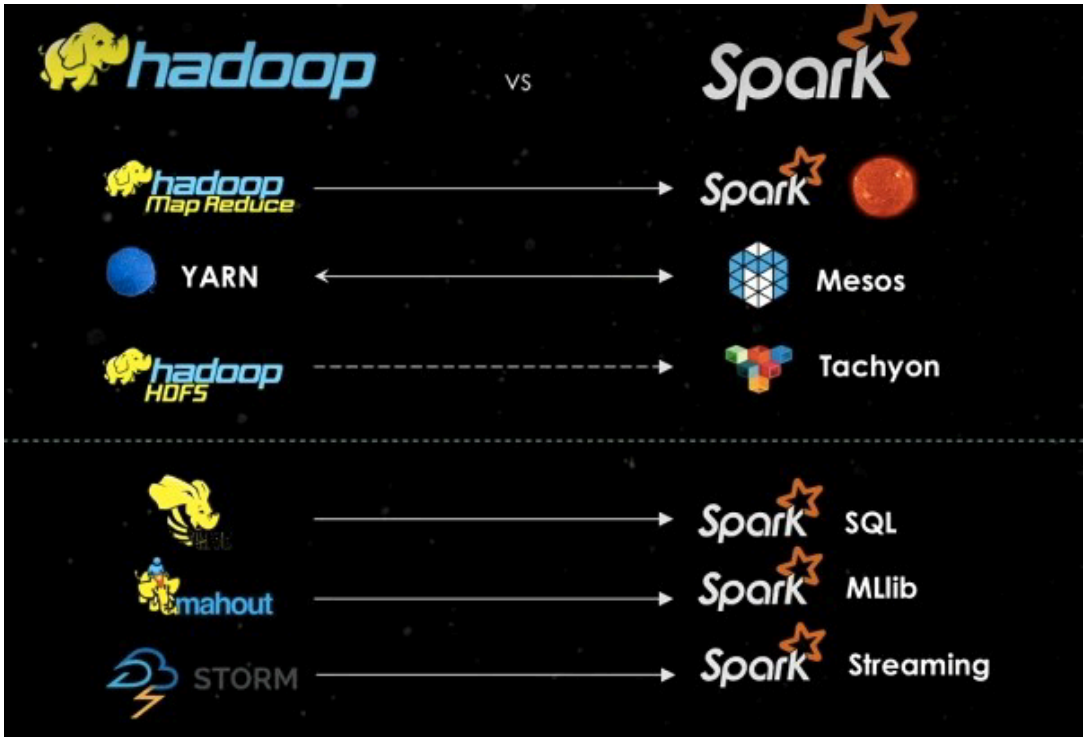

Scala
Java
python
R
DataFrames
ML Pipelines
Spark SQL
Spark Streaming
MLlib
GraphX
Spark Core
Data Sources
hadoop
cassandra
HBASE
{JSON}
MySQL
elasticsearch

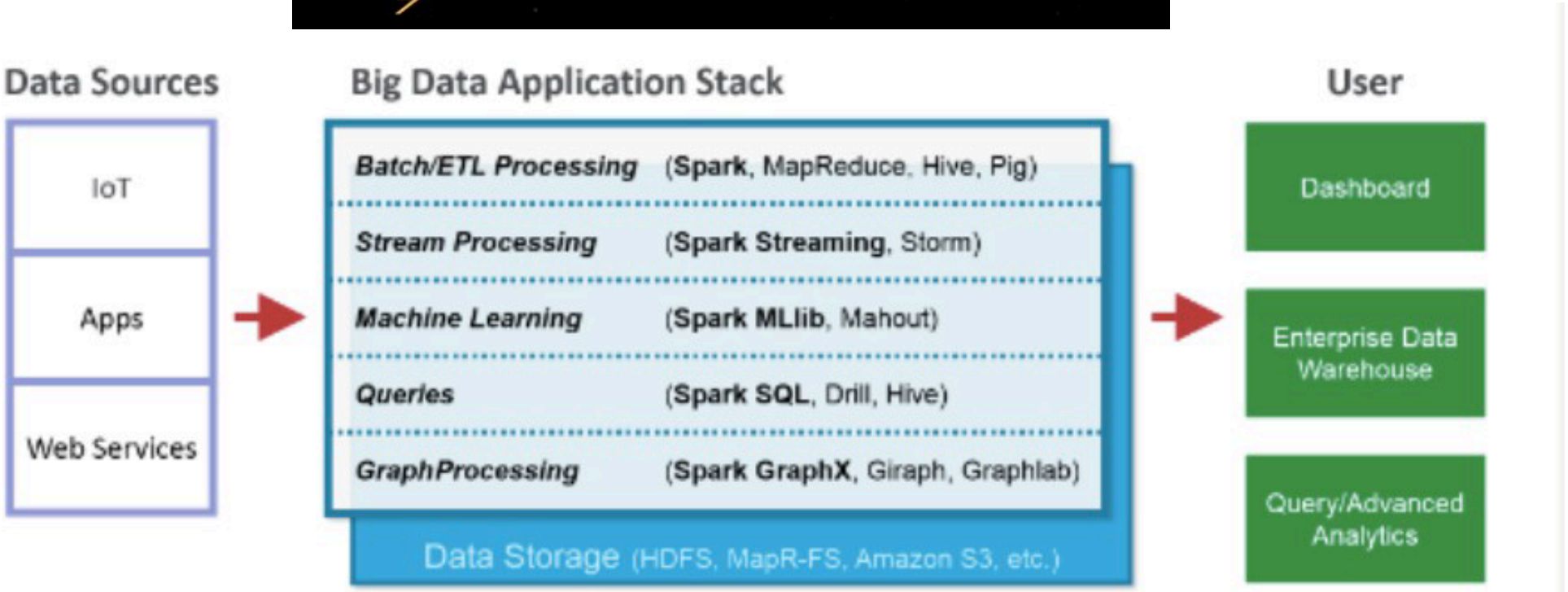
Spark Platform

CHULALONGKORN BUSINESS SCHOOL
FLAGSHIP FOR LIFE



Source: MapR Academy





Source: MapR Academy

Do we still need Hadoop?

CHULALONGKORN BUSINESS SCHOOL
FLAGSHIP FOR LIFE

Yes, why Hadoop?

- HDFS
- YARN
- MapReduce is mature and still be appropriate for certain workloads
- Other services: Sqoop, Flume, etc.

But you can still use other resource management, storages

- Spark Standalone
- Amazon S3
- Mesos

Start Spark-shell

CHULALONGKORN BUSINESS SCHOOL
FLAGSHIP FOR LIFE

$spark-shell

```
[root@quickstart 201402_babs_open_data]# spark-shell
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/usr/lib/zookeeper/lib/slf4j-log4j12-1.7.5.jar!/or
g/slf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/usr/jars/slf4j-log4j12-1.7.5.jar!/org/slf4j/impl/
StaticLoggerBinder.class]
SLF4J: See http://www.slf4j.org/codes.html#multiple_bindings for an explanation.
SLF4J: Actual binding is of type [org.slf4j.impl.Log4jLoggerFactory]
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel).
Welcome to
      ____              __
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ `/ __/  '_/
   /___/ .__/\_,_/_/ /_/\_\   version 1.6.0
      /_/

Using Scala version 2.10.5 (Java HotSpot(TM) 64-Bit Server VM, Java 1.7.0_67)
Type in expressions to have them evaluated.
Type :help for more information.
```

Testing SparkContext

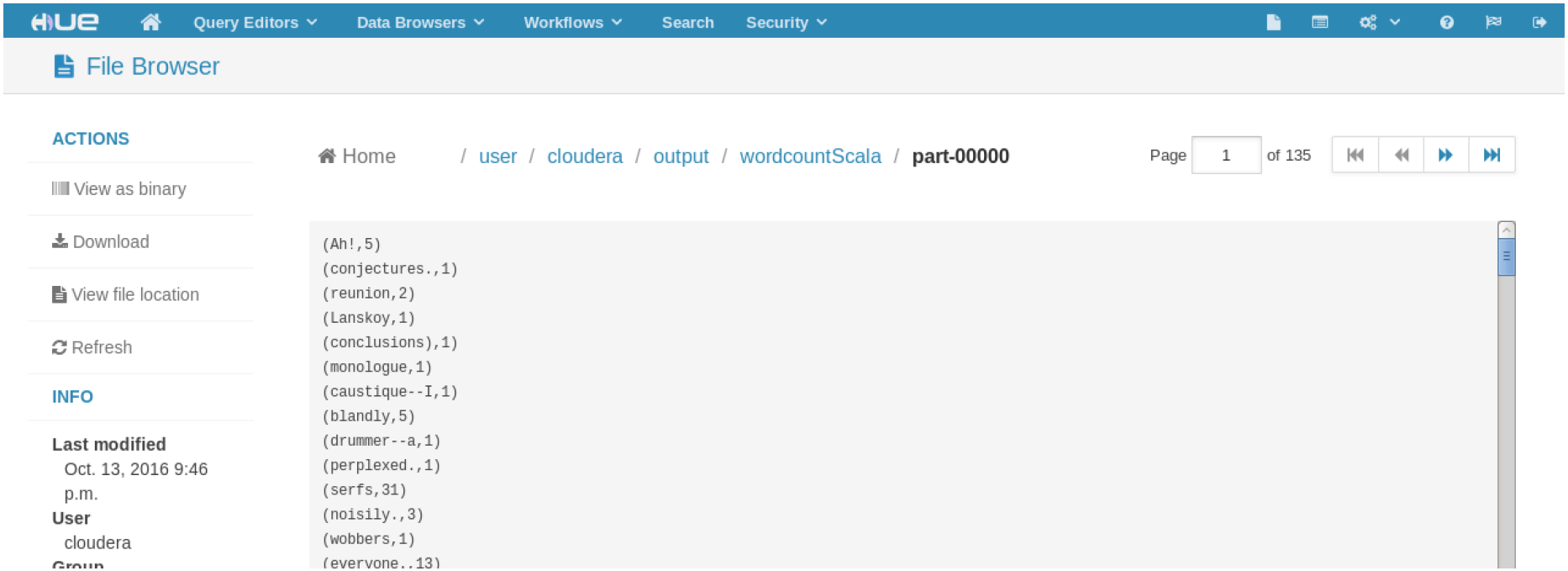
scala> sc

```
scala> sc
res0: org.apache.spark.SparkContext = org.apache.spark.SparkContext@18c07e25
```

Spark Program in Scala: WordCount

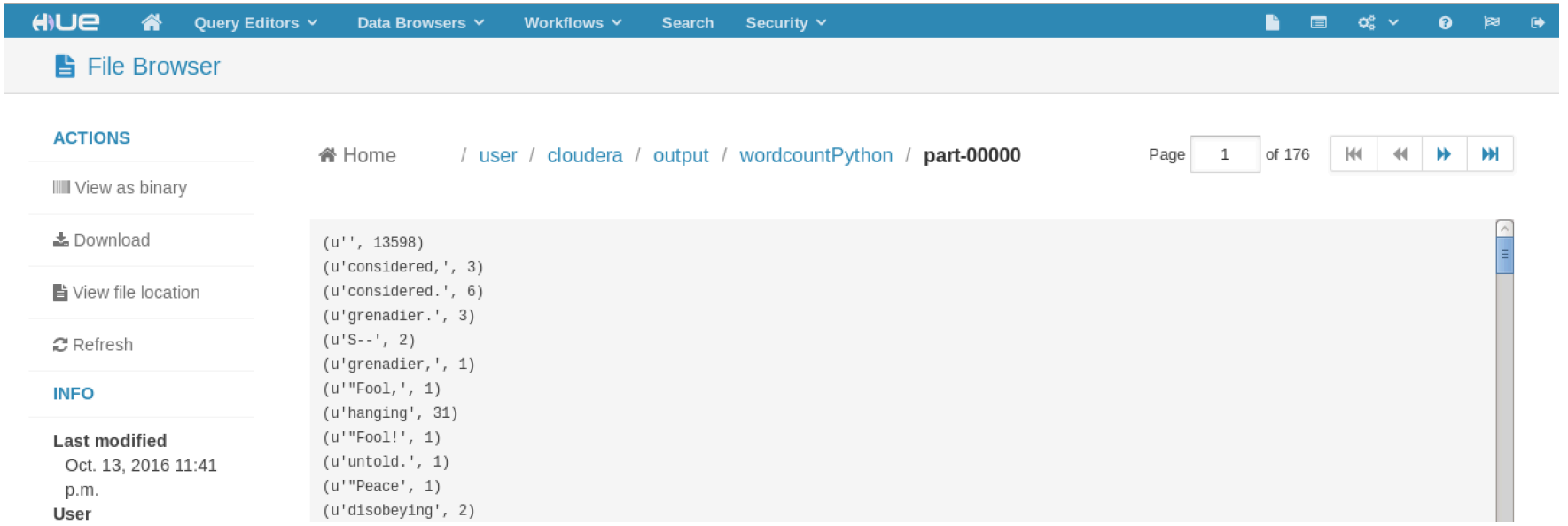
CHULALONGKORN BUSINESS SCHOOL
FLAGSHIP FOR LIFE

```
scala> val file = sc.textFile("hdfs:///user/cloudera/input/PG2600.txt")
scala> val wc = file.flatMap(l => l.split(" ")).map(word =>(word, 1)).reduceByKey(_ + _)
scala> wc.saveAsTextFile("hdfs:///user/cloudera/output/wordcountScala")
```



Spark Program in Python: WordCount


```
$ pyspark
>>> from operator import add
>>> file =sc.textFile("hdfs:///user/cloudera/input/PG2600.txt")
>>> wc = file.flatMap(lambda x: x.split(' ')).map(lambda x:(x,
1)).reduceByKey(add)
>>> wc.saveAsTextFile("hdfs:///user/cloudera/output/
wordcountPython")
```



Loading data from MySQL

CHULALONGKORN BUSINESS SCHOOL
FLAGSHIP FOR LIFE

Download MySQL driver & Start Spark-shell

Open New Terminal

$ mkdir spark

$ cd spark

$wget https://github.com/bobbylovemovie/trainbigdata/raw/master/Spark/mysql-connector-java-5.1.23.jar

Running Spark-shell

$ spark-shell --jars mysql-connector-java-5.1.23.jar

```
16/06/28 15:23:35 WARN shortcircuit.DomainSocketFactory: The short-circuit local rea
ds feature cannot be used because libhadoop cannot be loaded.
SQL context available as sqlContext.

scala>
```

```
$ scala> :paste
val url="jdbc:mysql://localhost:3306/test_mysql_db"
val username = "root"
val password = "cloudera"
import org.apache.spark.rdd.JdbcRDD
import java.sql.{Connection, DriverManager, ResultSet}
Class.forName("com.mysql.jdbc.Driver").newInstance
val myRDD = new JdbcRDD( sc, () =>
DriverManager.getConnection(url,username,password) ,
"SELECT * FROM country_tbl LIMIT ?, ?", 0, 5, 2, r =>r.getString("id") + ", " +
r.getString("country"))
myRDD.count
myRDD.foreach(println)
```

Output

CHULALONGKORN BUSINESS SCHOOL
FLAGSHIP FOR LIFE

```
// Exiting paste mode, now interpreting.

1, USA
2, CANADA
4, Brazil
61, Japan
65, Singapore
66, Thailand
url: String = jdbc:mysql://localhost:3306/test_mysql_db
username: String = root
password: String = cloudera
import org.apache.spark.rdd.JdbcRDD
import java.sql.{Connection, DriverManager, ResultSet}
myRDD: org.apache.spark.rdd.JdbcRDD[String] = JdbcRDD[0] at JdbcRDD at <console>
:39
```

scala> myRDD.saveAsTextFile("hdfs:///user/cloudera/output/mysqlFromSpark")