CHULALONGKORN
BUSINESS SCHOOL
FLAGSHIP FOR LIFE

Thaveewat Khanan

2603615 Organization Data Management

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 ?

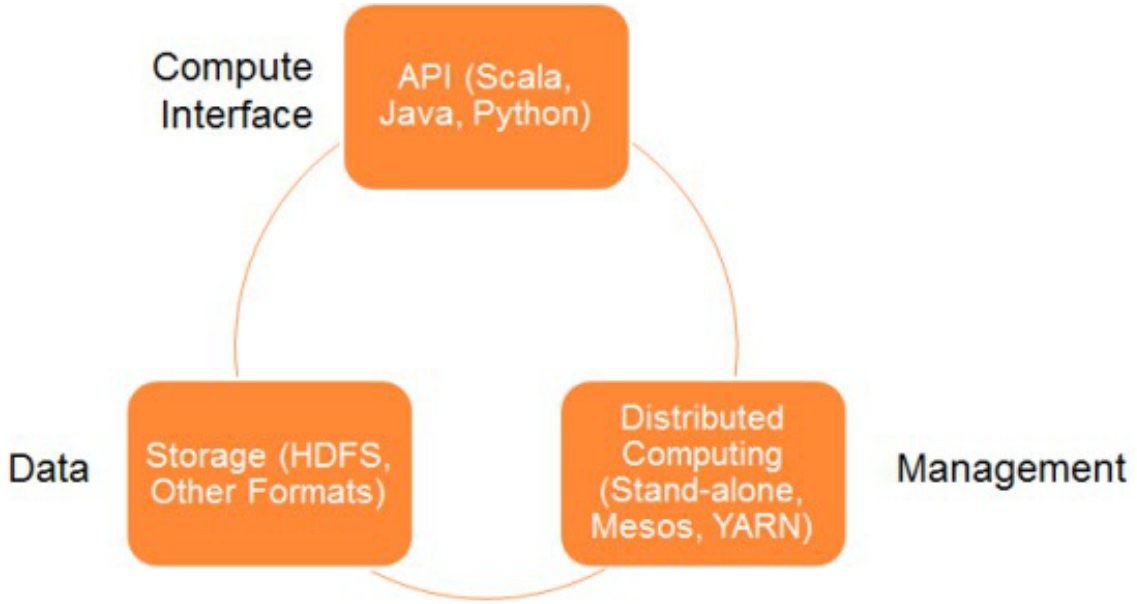
CHULALONGKORN BUSINESS SCHOOL
FLAGSHIP FOR LIFE

Framework for distributed processing.

In-memory, fault tolerant data structures

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

Open source

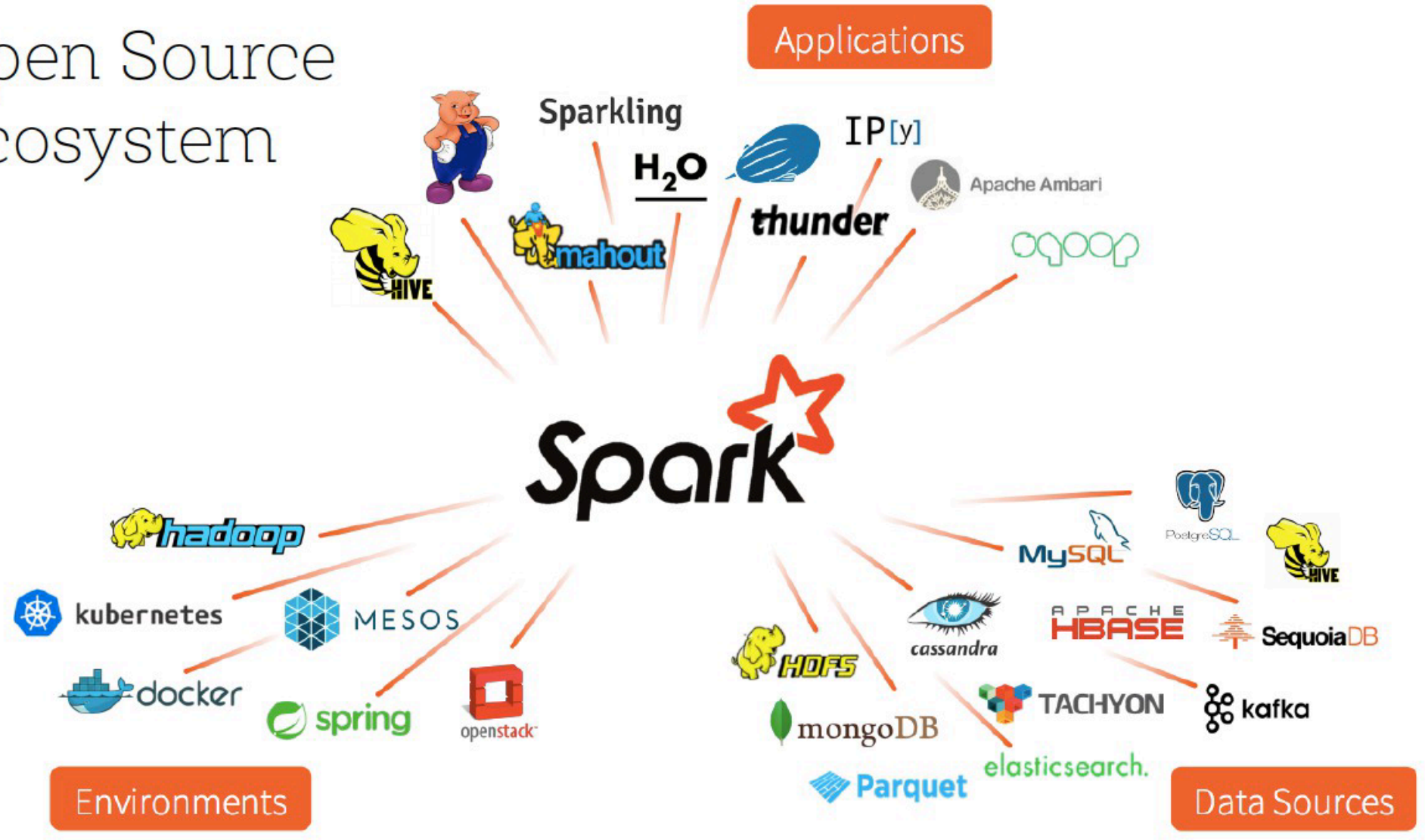


Why Spark ?

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

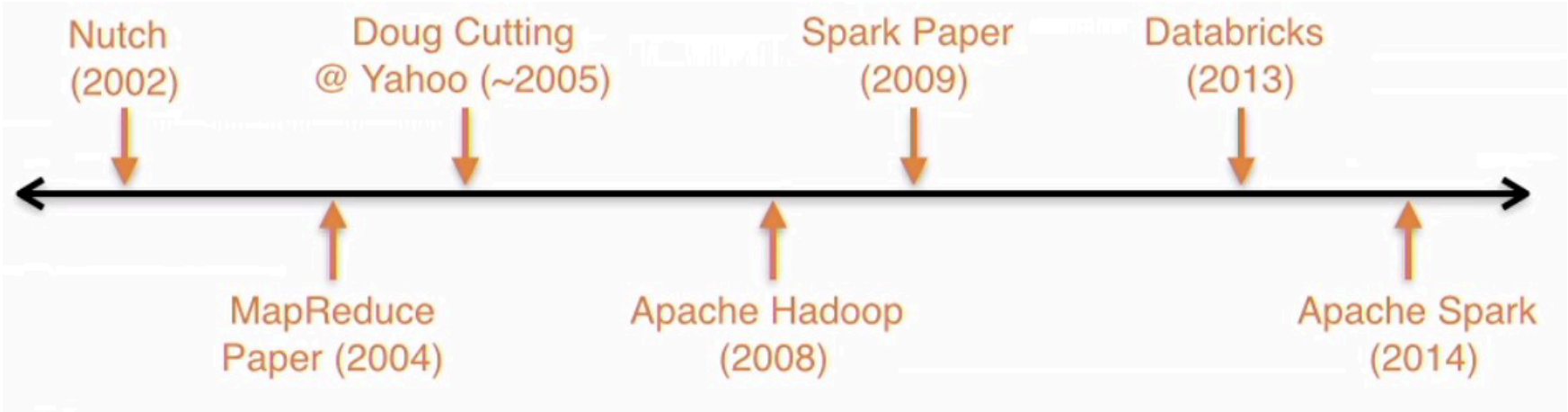


Source: Jump start into Apache Spark and Databricks

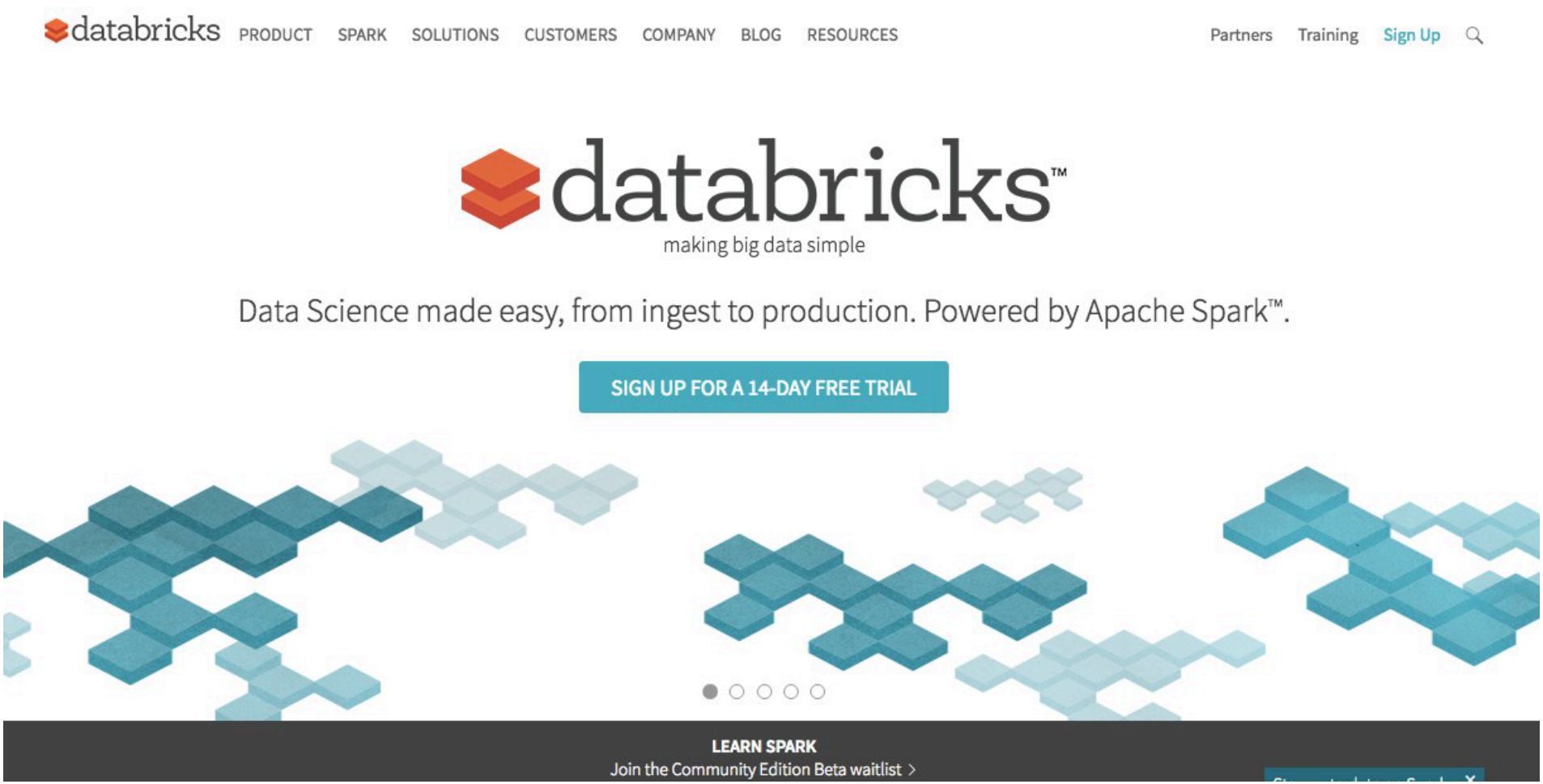
Spark: History

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



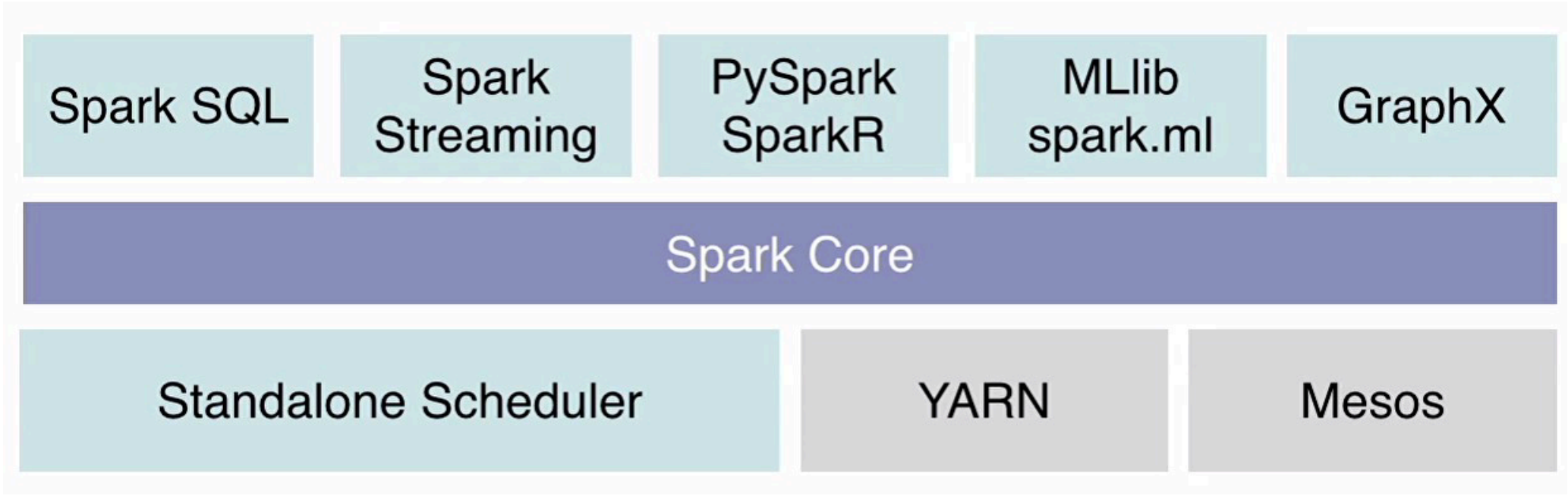
CHULALONGKORN
BUSINESS SCHOOL
FLAGSHIP FOR LIFE

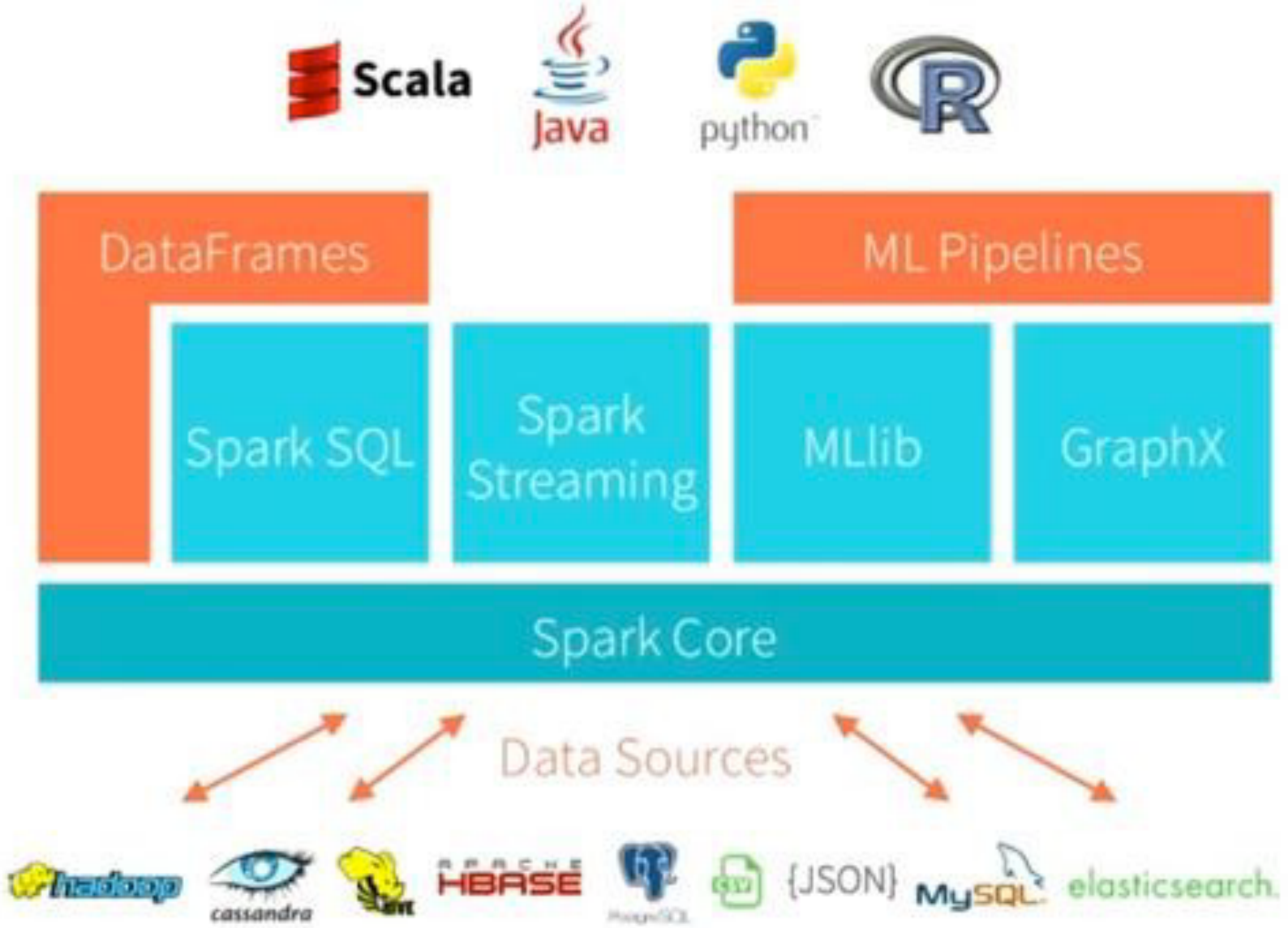


Thaveewat Khanan

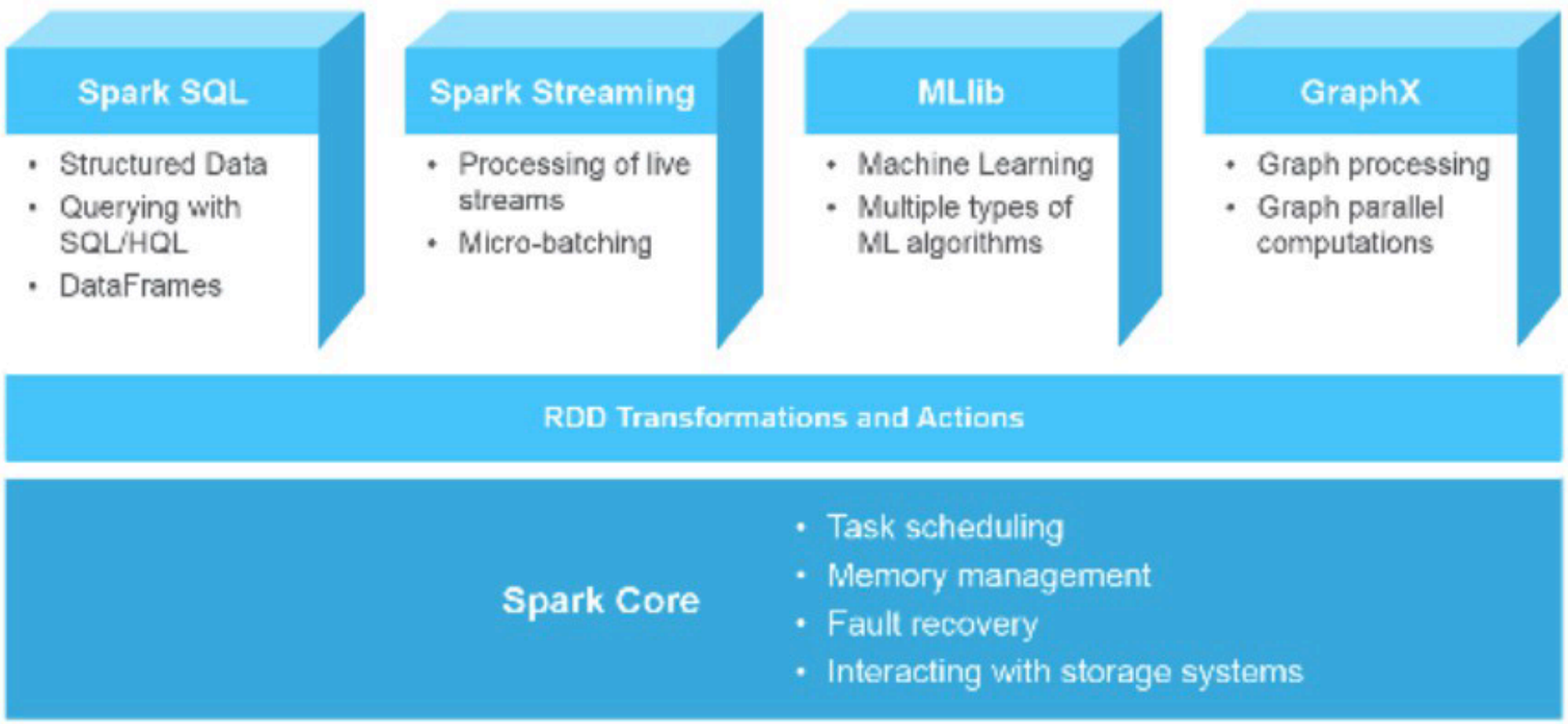
2603615 Organization Data Management

Spark Platform

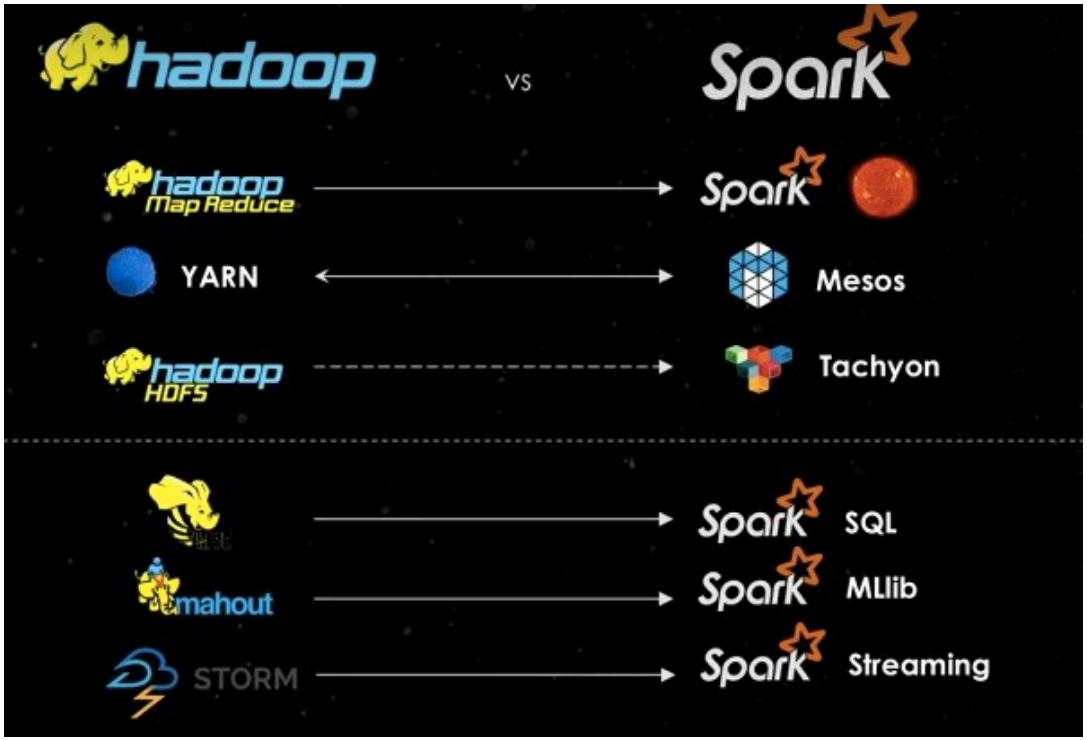


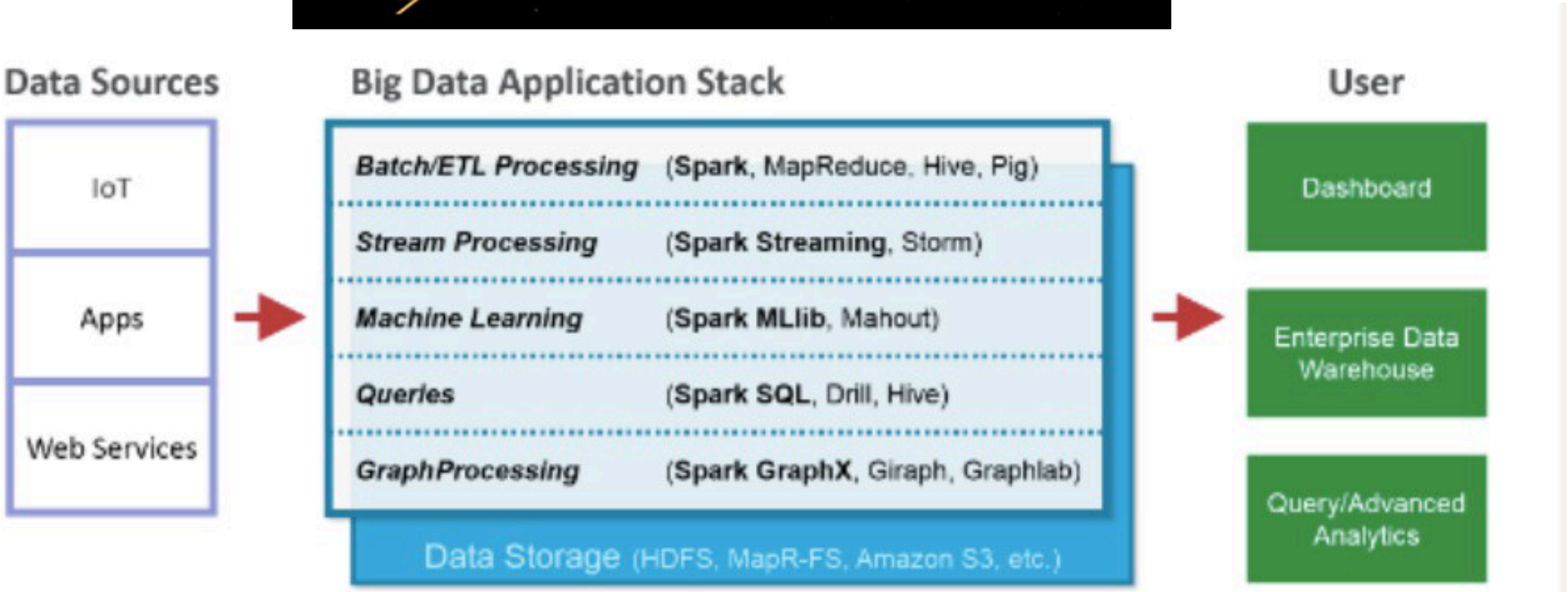

Scala
Java
python
R
DataFrames
ML Pipelines
Spark SQL
Spark Streaming
MLlib
GraphX
Spark Core
Data Sources
hadoop
cassandra
HBASE
{JSON}
MySQL
elasticsearch

Spark Platform



Source: MapR Academy





Source: MapR Academy

Do we still need Hadoop?

CHULALONGKORN BUSINESS SCHOOL
FLAGSHIP FOR LIFE

Yes, why Hadoop?

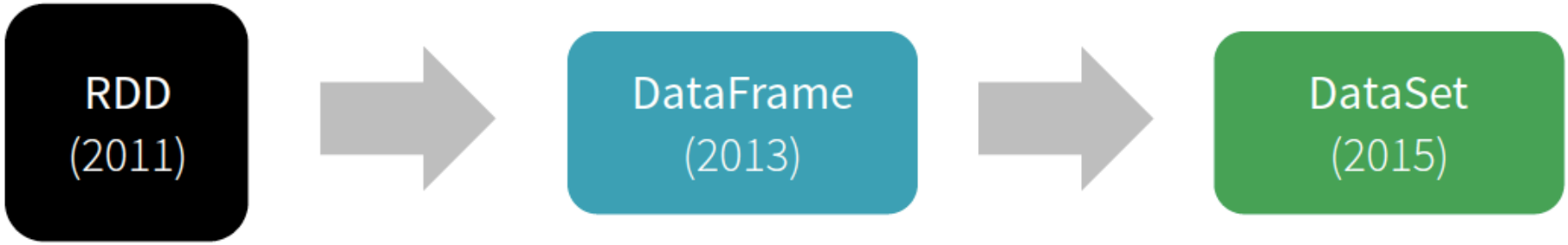
- HDFS
- YARN
- MapReduce is mature and still be appropriate for certain workloads
- Other services: Sqoop, Flume, etc.

But you can still use other resource management, storages

- Spark Standalone
- Amazon S3
- Mesos

History of Spark APIs

CHULALONGKORN BUSINESS SCHOOL
FLAGSHIP FOR LIFE



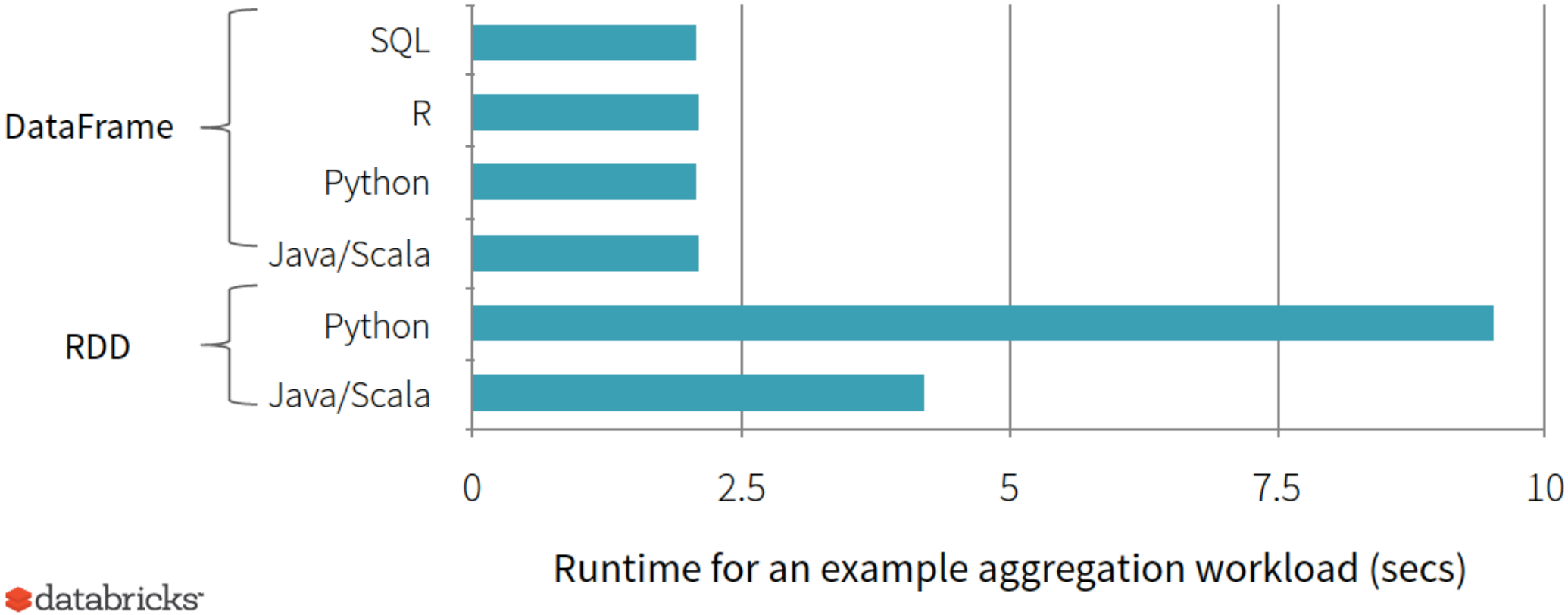
- Distribute collection of JVM objects
- Functional Operators (map, filter, etc.)

- 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.

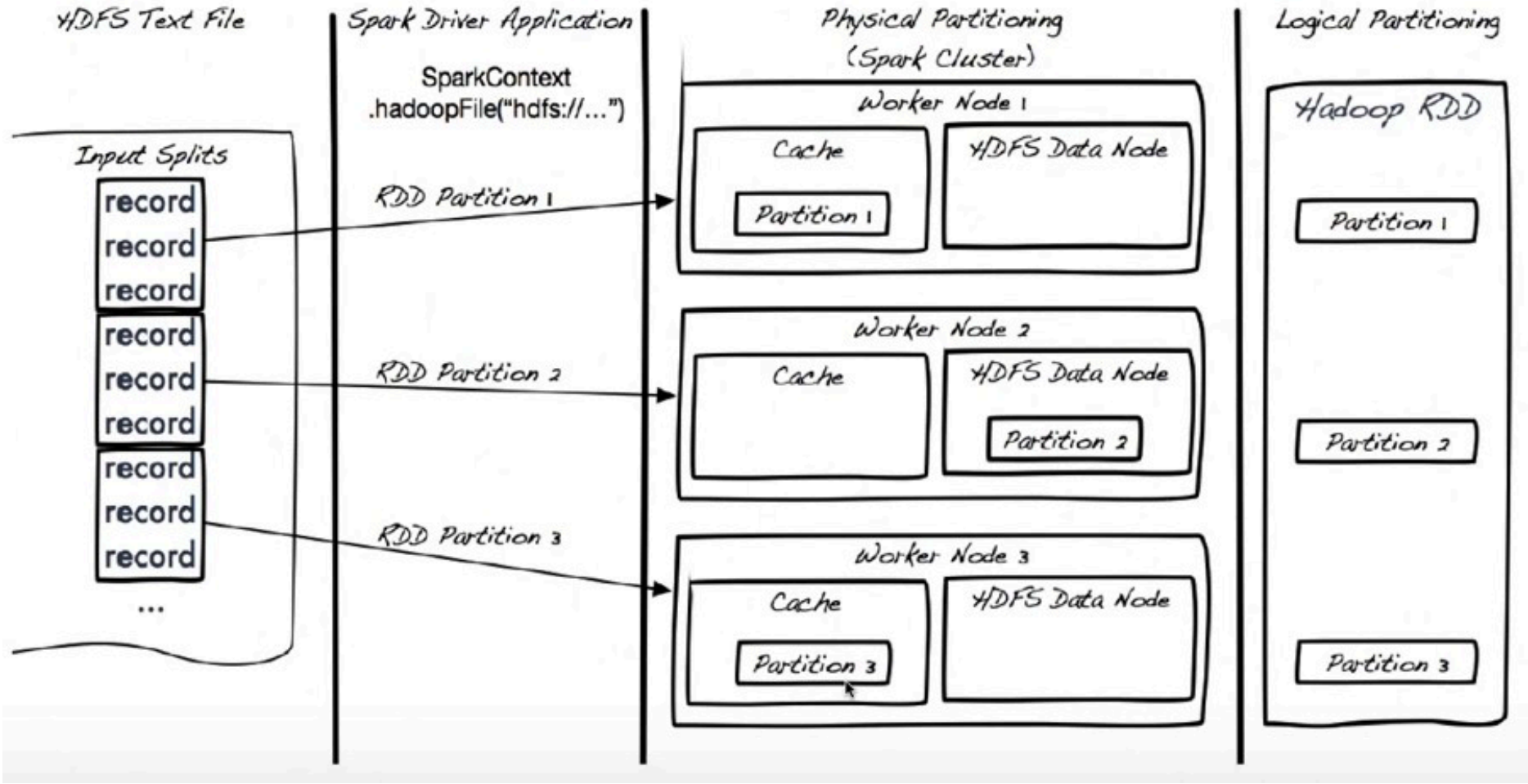
Distributed: data is chucked into partitions and stored in memory acress the custer.

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")



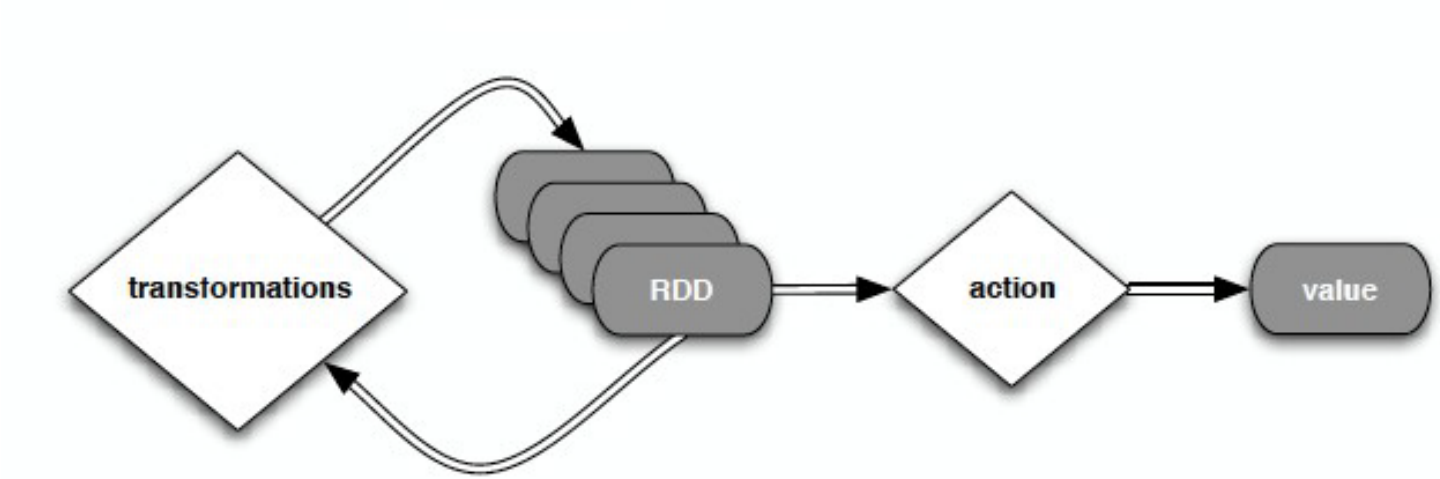
Source: Pspark: A brain-friendly introduction

RDD: Operations

CHULALONGKORN BUSINESS SCHOOL
FLAGSHIP FOR LIFE

Transformations: transformations are lazy (not computed immediately)

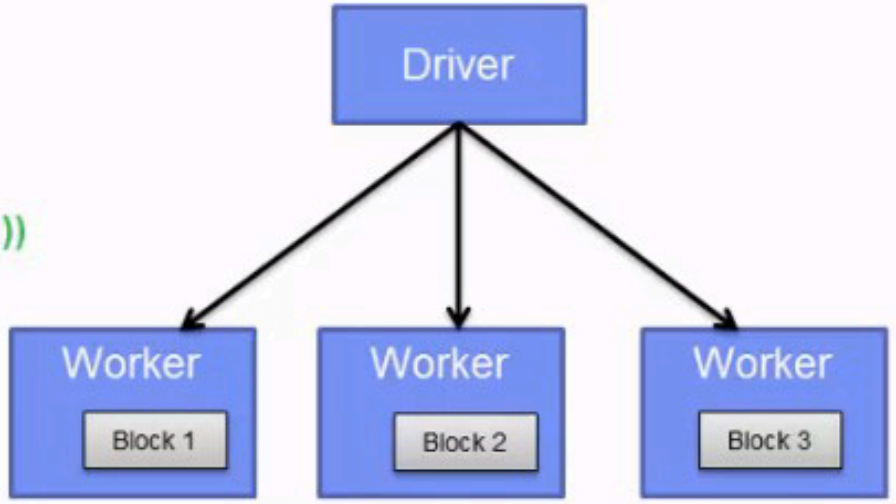
Actions: the transformed RDD gets recomputed when an action is run on it (default)



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()
```

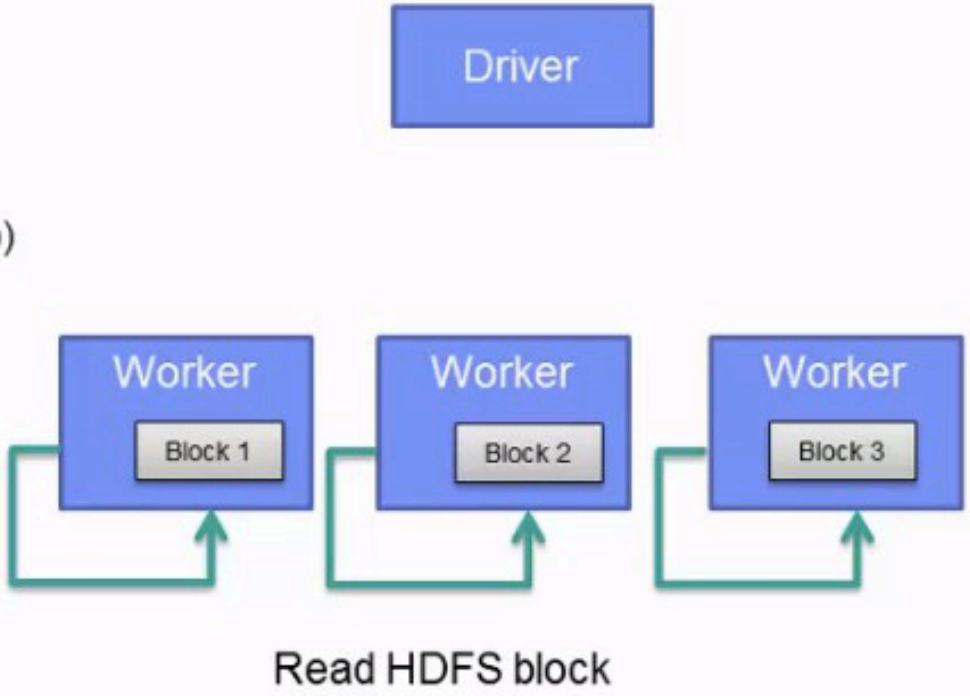


Driver sends the code to be executed on each block

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()
```

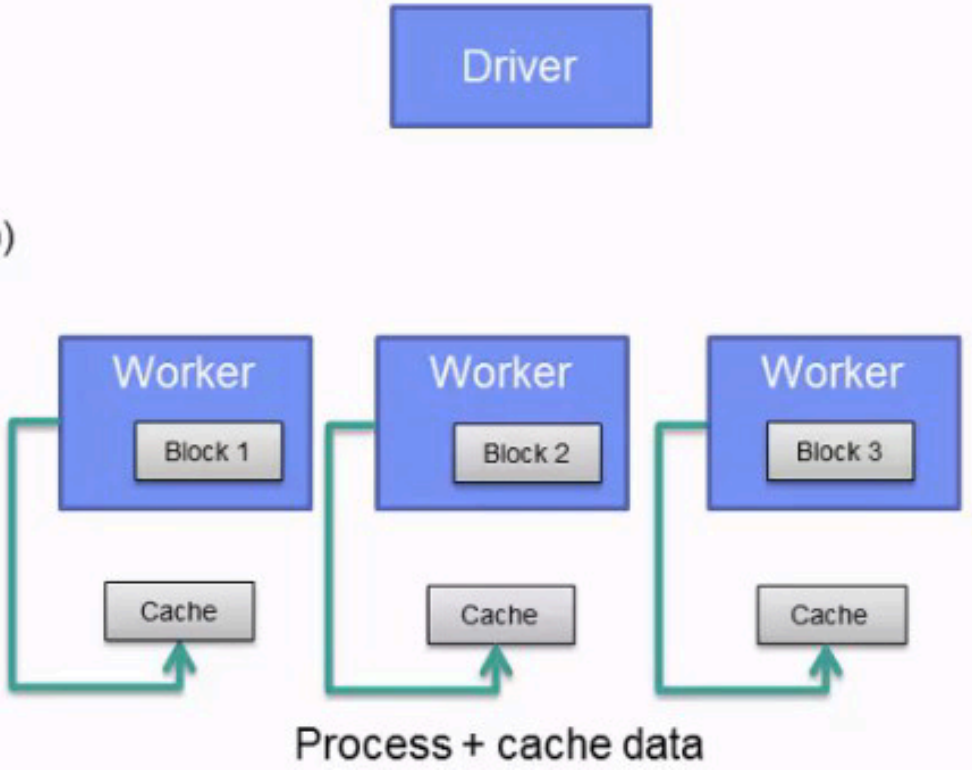


Source: Spark Fundamentals I Big Data Usibersity

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()
```

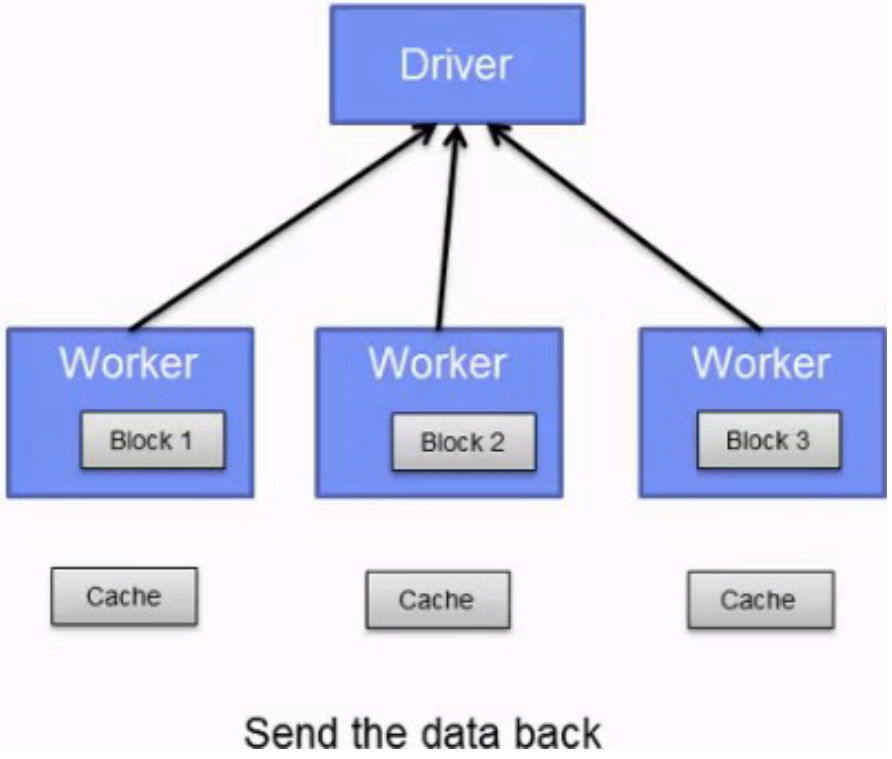


Source: Spark Fundamentals I Big Data Usibersity

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()
```

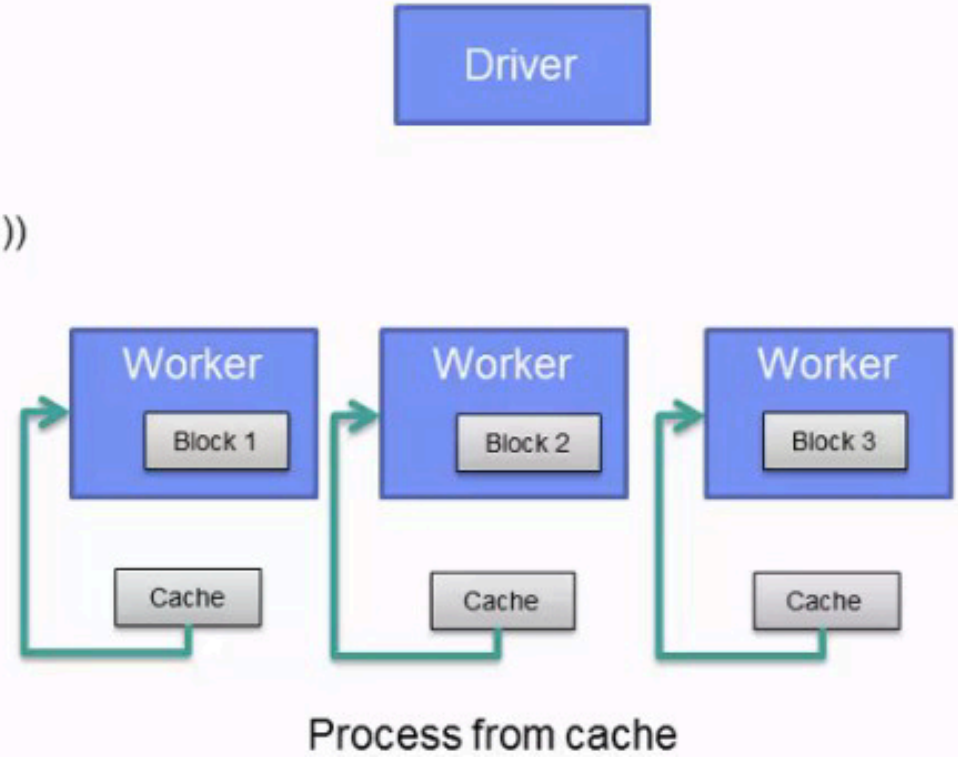


Source: Spark Fundamentals I Big Data Usibersity

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()
```

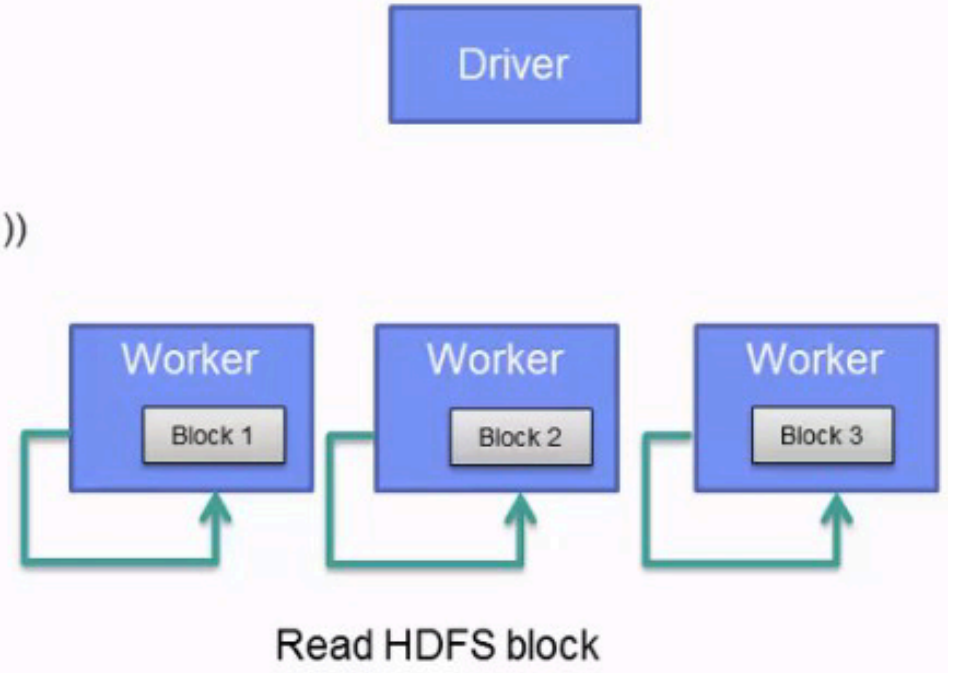


Source: Spark Fundamentals I Big Data Usibersity

What happens when an action is executed

CHULALONGKORN BUSINESS SCHOOL
FLAGSHIP FOR LIFE

```
// 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

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), ...
```

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

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)]))=>

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

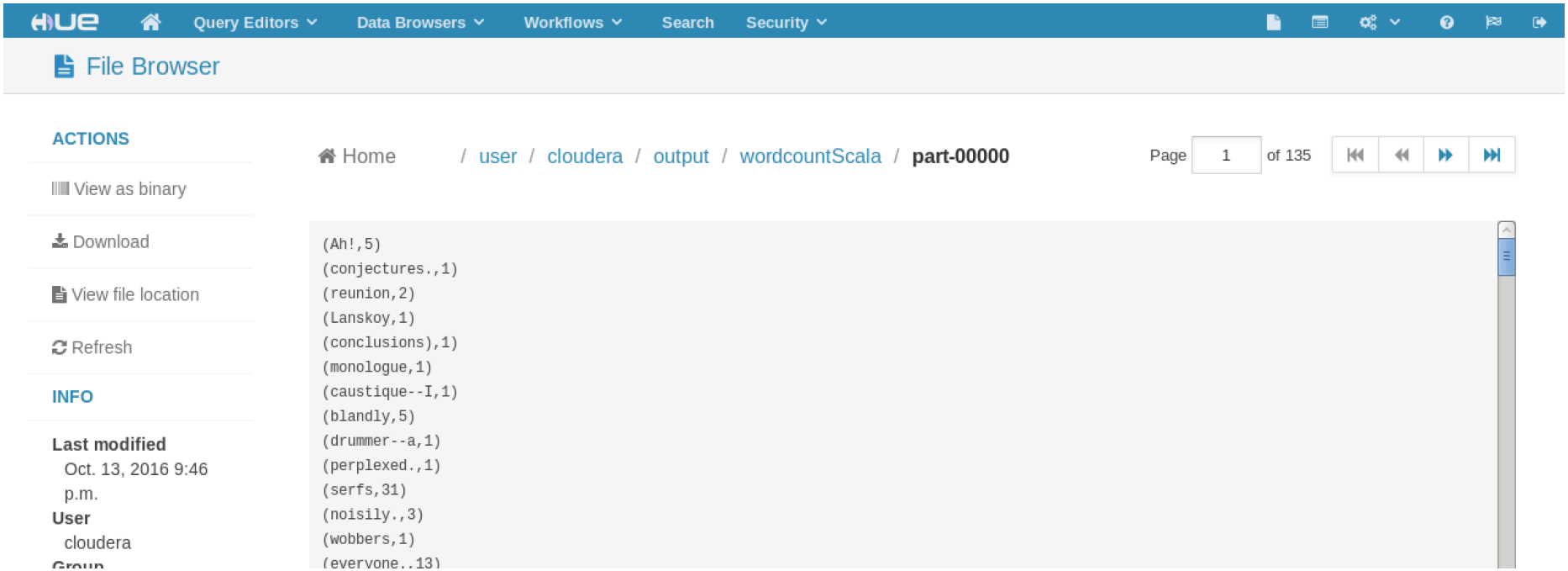
scala> sc

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

Spark Program in Scala: WordCount

CHULALONGKORN BUSINESS SCHOOL
FLAGSHIP FOR LIFE

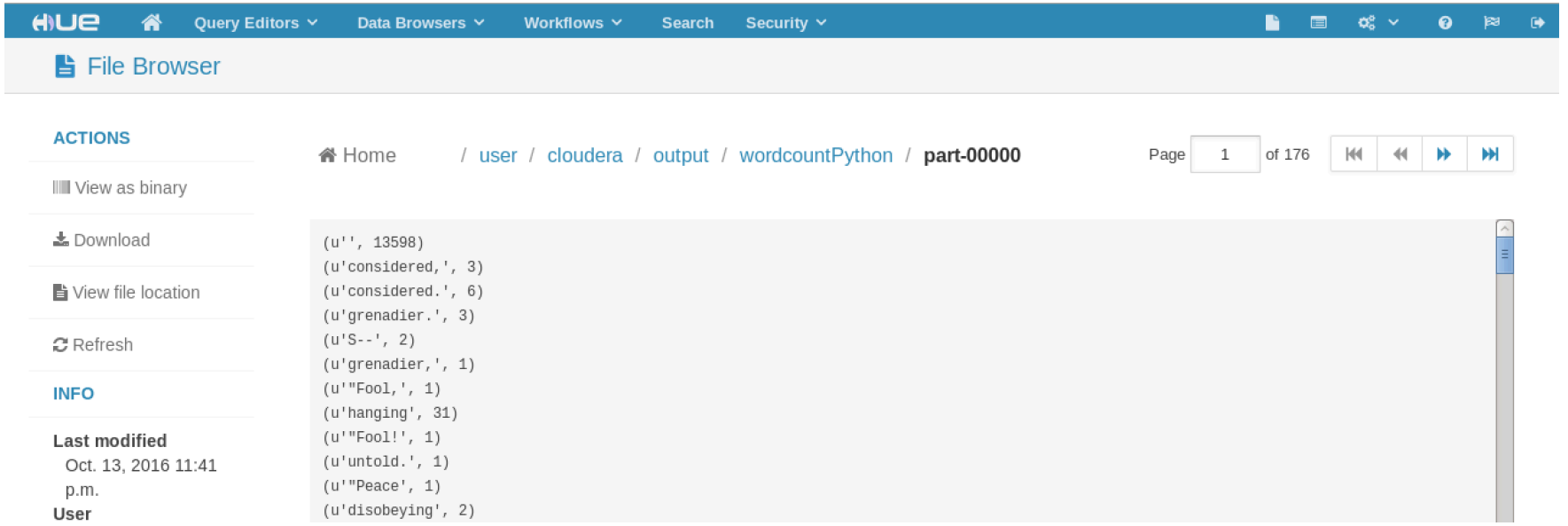
```
scala> val file = sc.textFile("hdfs:///user/cloudera/input/PG2600.txt")
scala> val wc = file.flatMap(l => l.split(" ")).map(word =>(word, 1)).reduceByKey(_ + _)
scala> wc.saveAsTextFile("hdfs:///user/cloudera/output/wordcountScala")
```



Spark Program in Python: WordCount

CHULALONGKORN BUSINESS SCHOOL
FLAGSHIP FOR LIFE

```
$ 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")
```



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”)
```

Key-Value Operations

CHULALONGKORN BUSINESS SCHOOL
FLAGSHIP FOR LIFE

```
>>> pet = sc.parallelize([("cat",1),("dog",1),("cat",2)])
>>> pet2 = pet.reduceByKey(lambda x, y:x+y)
>>> pet3 = pet.groupByKey()
>>> pet4 = pet.sortByKey()
```

Loading data from MySQL

CHULALONGKORN BUSINESS SCHOOL
FLAGSHIP FOR LIFE

Download MySQL driver & Start Spark-shell

$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")

Spark SQL

CHULALONGKORN BUSINESS SCHOOL
FLAGSHIP FOR LIFE

DataFrame

A distributed collection of rows organied into named columns.

An abstraction for selecting, filtering, aggregating, and plotting structured data.

Previously => SchemaRDD

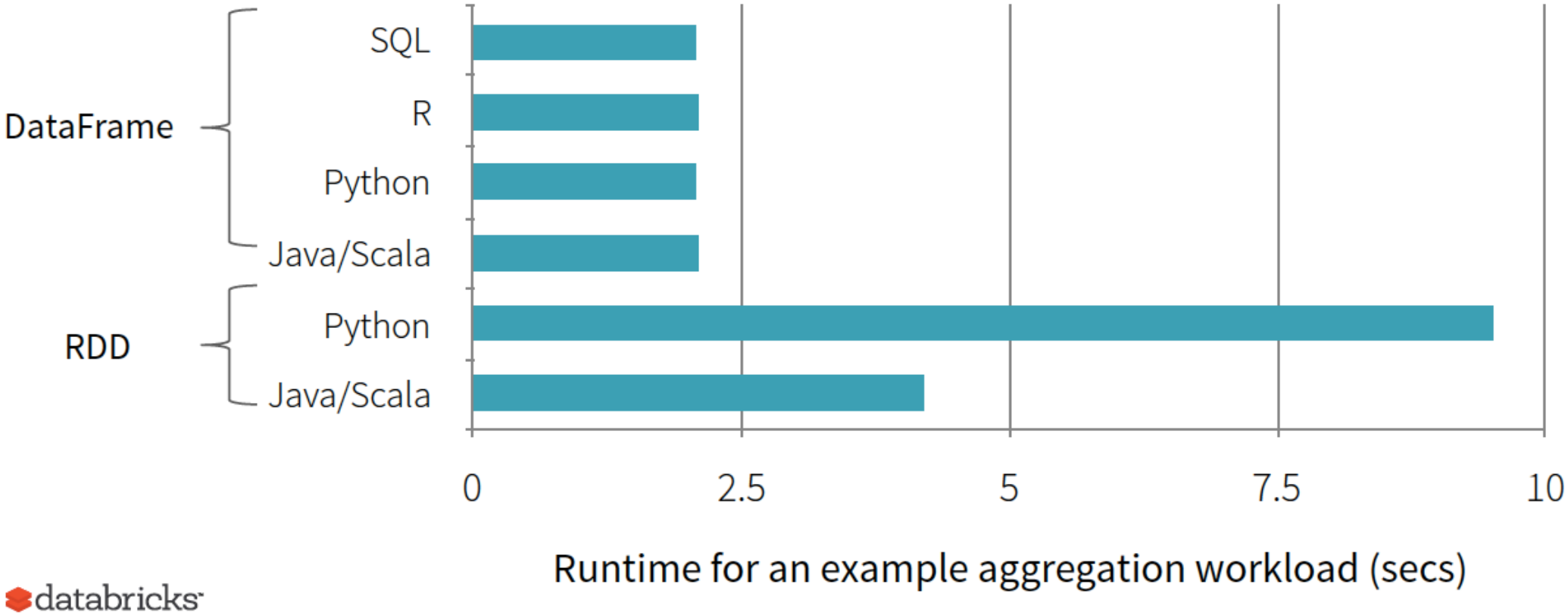
Creating and running Spark program faster

– Write less code

– Read less data

– Let the optimizer do the hard work

Benefit of Logical Plan: Performance Parity Across Languages



databricks

Source: Jump start into Apache Spark and Databricks

SparkSQL can leverage the Hive metastore

Hive Metastore can also be leveraged by a wide array of applications

– Spark

– Hive

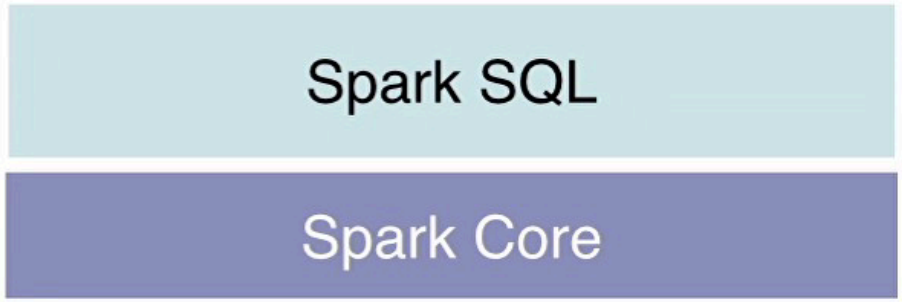
– Impala

Available from HiveContext

```
context = ps.HiveContext(sc)

# query with SQL
results = context.sql(
  "SELECT * FROM people")

# apply Python transformation
names = results.map(lambda p: p.name)
```



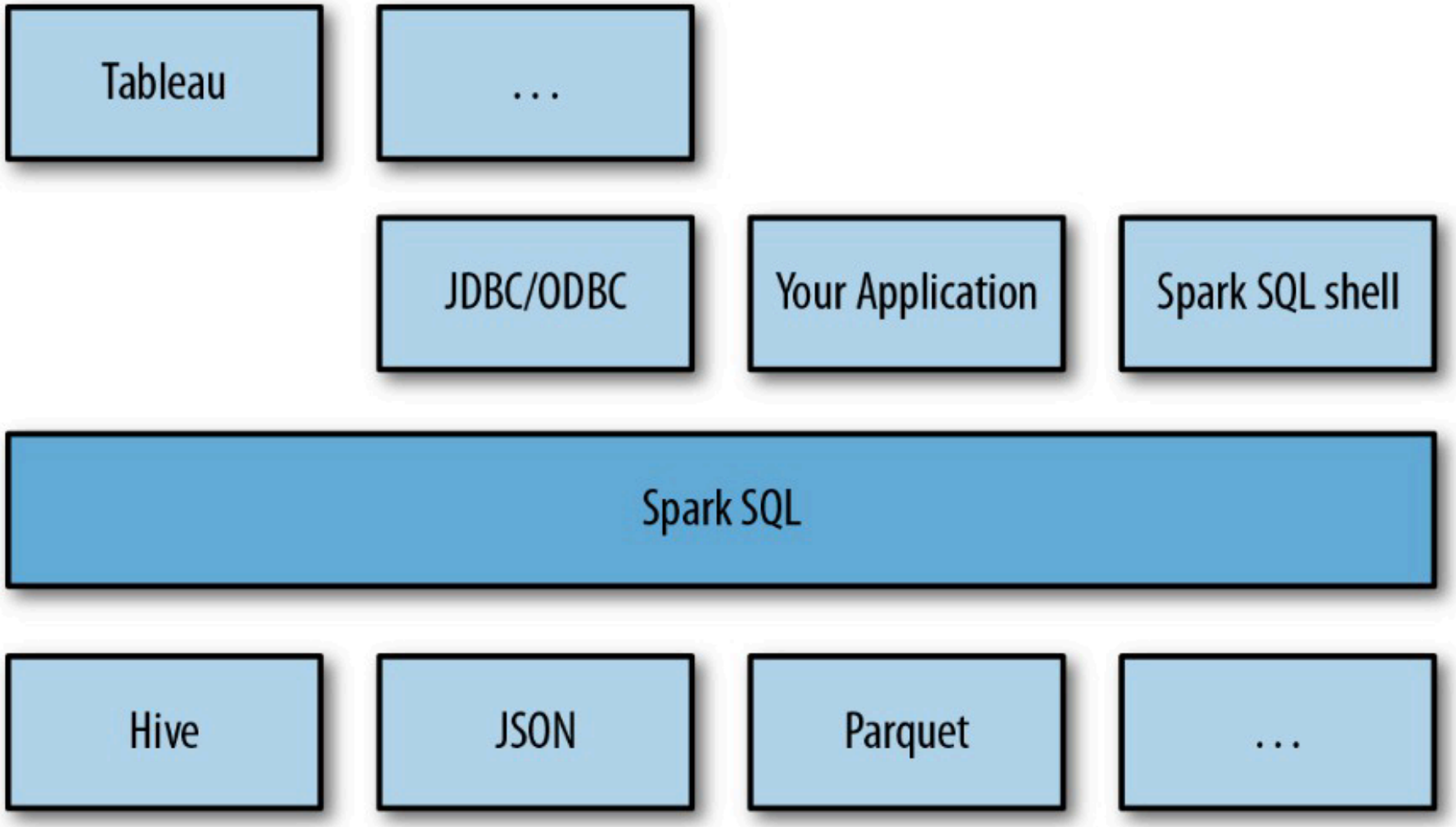
Unified interface for structured data

amplab UC BERKELEY databricks

Image credit: http://barrymieny.deviantart.com/

Spark SQL usage

CHULALONGKORN BUSINESS SCHOOL
FLAGSHIP FOR LIFE



Source: Learning Spark, O'Reilly Media, Inc.

Link Hive Metastore with Spark-Shell

CHULALONGKORN BUSINESS SCHOOL
FLAGSHIP FOR LIFE

```
$ spark-shell --jars mysql-connector-java-5.1.23.jar
scala > val sqlContext = new org.apache.spark.sql.hive.HiveContext(sc)
scala> sqlContext.sql("CREATE TABLE IF NOT EXISTS movie(userid
STRING, movieid STRING, rating INT, timestamp STRING) ROW FORMAT
DELIMITED FIELDS TERMINATED BY '\t' LINES TERMINATED BY '\n'")
scala> sqlContext.sql("LOAD DATA LOCAL INPATH '/home/cloudera/
movielens_dataset/ml-100k/u.data' INTO TABLE movie")
scala> val result = sqlContext.sql("SELECT * FROM movie")
scala> result.show()
```

```
scala> result.show()
+------+-------+------+---------+
|userid|movieid|rating|timestamp|
+------+-------+------+---------+
|   196|    242|     3|881250949|
|   186|    302|     3|891717742|
|    22|    377|     1|878887116|
|   244|     51|     2|880606923|
|   166|    346|     1|886397596|
|   298|    474|     4|884182806|
|   115|    265|     2|881171488|
```

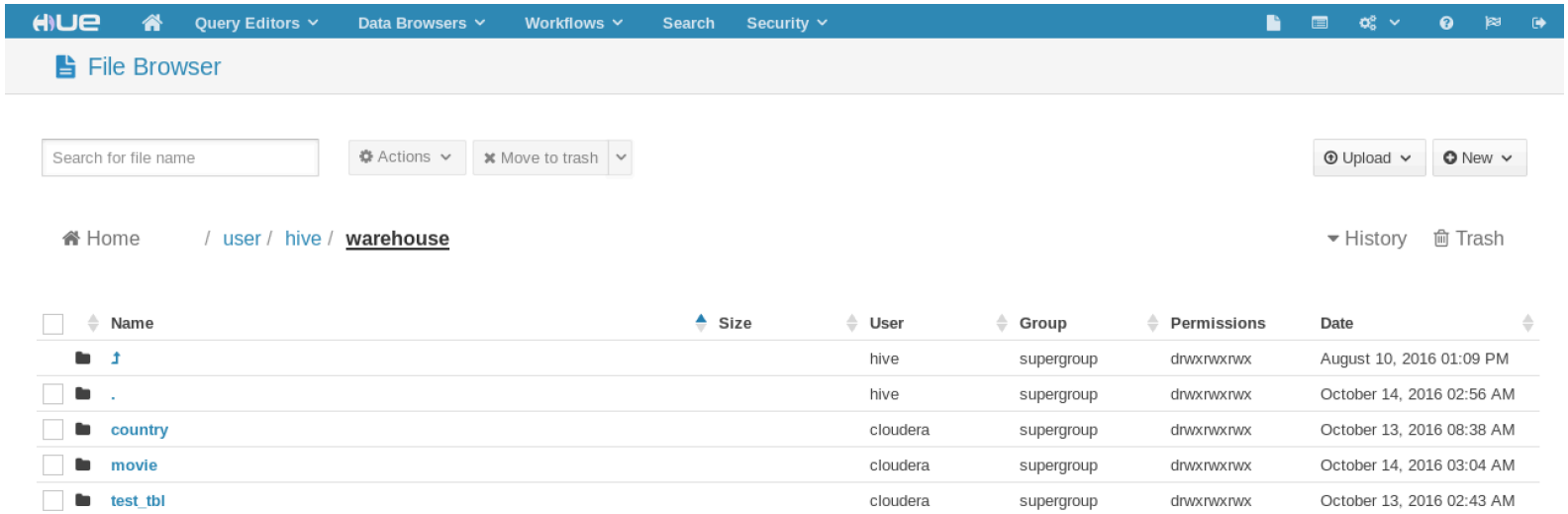
Link Hive Metastore with Spark-Shell

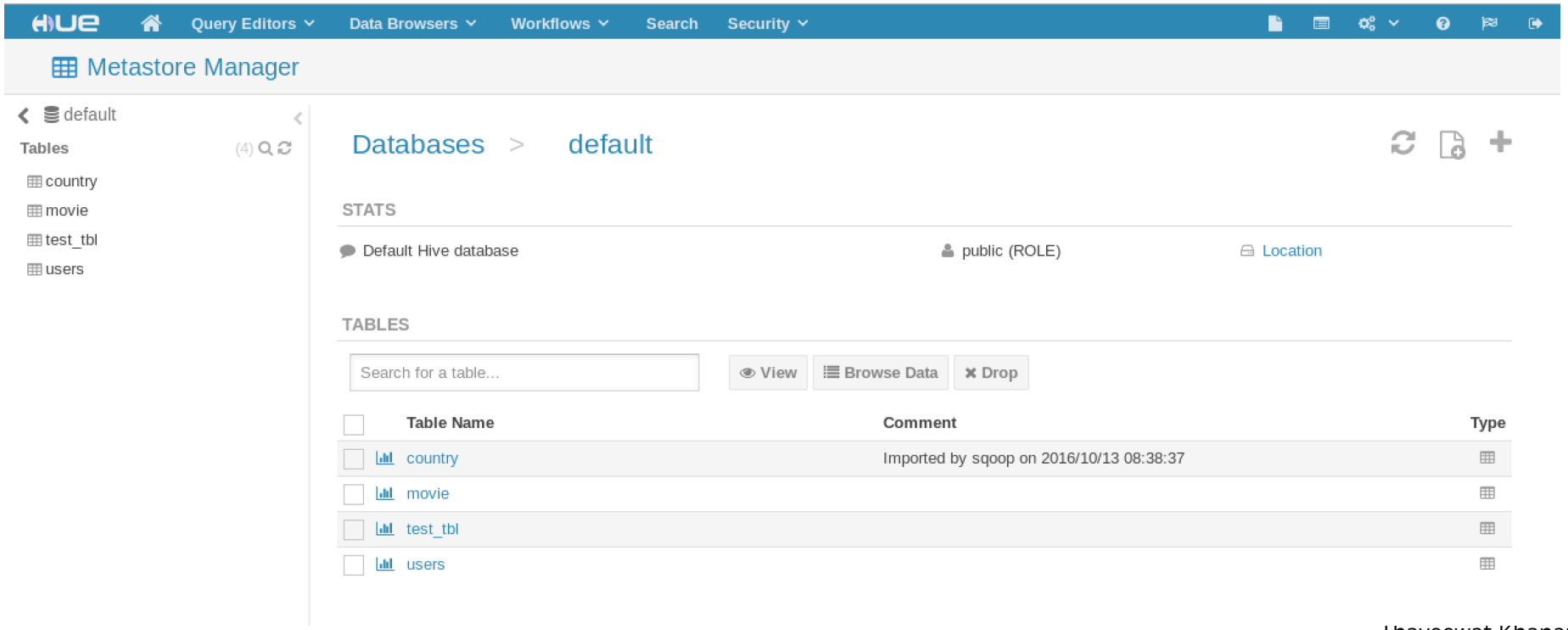
CHULALONGKORN
BUSINESS SCHOOL
FLAGSHIP FOR LIFE

Copy the configuration file

```
$sudo cp /usr/lib/hive/conf/hive-site.xml /usr/lib/spark/conf/
```

```
$ spark-shell
scala > val sqlContext = new org.apache.spark.sql.hive.HiveContext(sc)
scala> sqlContext.sql("CREATE TABLE IF NOT EXISTS movie(userid
STRING, movieid STRING, rating INT, timestamp STRING) ROW FORMAT
DELIMITED FIELDS TERMINATED BY '\t' LINES TERMINATED BY '\n'")
scala> sqlContext.sql("LOAD DATA LOCAL INPATH '/home/cloudera/
movielens_dataset/ml-100k/u.data' INTO TABLE movie")
scala> val result = sqlContext.sql("SELECT * FROM movie")
scala> result.show()
```


HUE
Query Editors
Data Browsers
Workflows
Search
Security
File Browser
Search for file name
Actions
Move to trash
Upload
New
Home / user / hive / warehouse
History
Trash
Name
Size
User
Group
Permissions
Date
hive supergroup drwxrwxrwx August 10, 2016 01:09 PM
hive supergroup drwxrwxrwx October 14, 2016 02:56 AM
country cloudera supergroup drwxrwxrwx October 13, 2016 08:38 AM
movie cloudera supergroup drwxrwxrwx October 14, 2016 03:04 AM
test_tbl cloudera supergroup drwxrwxrwx October 13, 2016 02:43 AM


HUE
Query Editors
Data Browsers
Workflows
Search
Security
Metastore Manager
default
Tables (4)
country
movie
test_tbl
users
Databases > default
STATS
Default Hive database
public (ROLE)
Location
TABLES
Search for a table...
View
Browse Data
Drop
Table Name
Comment
Type
country
Imported by sqoop on 2016/10/13 08:38:37
movie
test_tbl
users

Spark SQL Meals Data

CHULALONGKORN BUSINESS SCHOOL
FLAGSHIP FOR LIFE

Upload a data to HDFS

```
$ wget https://github.com/bobbylovemovie/trainbigdata/raw/master/Spark/events.txt
$ wget https://github.com/bobbylovemovie/trainbigdata/raw/master/Spark/meals.txt
$ hadoop fs -put events.txt /user/cloudera/input
$ hadoop fs -put meals.txt /user/cloudera/input
```

Spark SQL : Preparing data

CHULALONGKORN BUSINESS SCHOOL
FLAGSHIP FOR LIFE

```
$ pyspark
>>> meals_rdd =sc.textFile("hdfs:///user/cloudera/input/meals.txt")
>>> events_rdd =sc.textFile("hdfs:///user/cloudera/input/events.txt")
>>> header_meals = meals_rdd.first()
>>> header_events = events_rdd.first()
>>> meals_no_header = meals_rdd.filter(lambda row:row != header_meals)
>>> events_no_header =events_rdd.filter(lambda row:row != header_events)
>>> meals_json = meals_no_header.map(lambda
row:row.split(';')).map(lambda row_list:dict(zip(header_meals.split(';'),
row_list)))
>>> events_json = events_no_header.map(lambda
row:row.split(';')).map(lambda row_list:dict(zip(header_events.split(';'),
row_list)))
```

```
>>> import json
>>> def type_conversion(d, columns) :
...     for c in columns:
...         d[c] = int(d[c])
...     return d
...
>>> meal_typed = meals_json.map(lambda
j:json.dumps(type_conversion(j, ['meal_id','price'])))
>>> event_typed = events_json.map(lambda
j:json.dumps(type_conversion(j, ['meal_id','userid'])))
```

Spark SQL : Create DataFrame

```
>>> meals_dataframe = sqlContext.jsonRDD(meal_typed)
>>> events_dataframe = sqlContext.jsonRDD(event_typed)
>>> meals_dataframe.head()
    Row(dt=u'2013-01-01', meal_id=1, price=10, type=u'french')
>>> meals_dataframe.printSchema()
```

```
root
 |-- dt: string (nullable = true)
 |-- meal_id: long (nullable = true)
 |-- price: long (nullable = true)
 |-- type: string (nullable = true)
```

Running SQL Query

```
>>> meals_dataframe.registerTempTable('meals')
>>> events_dataframe.registerTempTable('events')
>>> sqlContext.sql("SELECT * FROM meals LIMIT 5").collect()
```

```
[Row(dt=u'2013-01-01', meal_id=1, price=10, type=u'french'), Row(dt=u'2013-01-01
', meal_id=2, price=13, type=u'chinese'), Row(dt=u'2013-01-02', meal_id=3, price
=9, type=u'mexican'), Row(dt=u'2013-01-03', meal_id=4, price=9, type=u'italian')
, Row(dt=u'2013-01-03', meal_id=5, price=12, type=u'chinese')]
```

```
>>> meals_dataframe.take(5)
```

```
[Row(dt=u'2013-01-01', meal_id=1, price=10, type=u'french'), Row(dt=u'2013-01-01
', meal_id=2, price=13, type=u'chinese'), Row(dt=u'2013-01-02', meal_id=3, price
=9, type=u'mexican'), Row(dt=u'2013-01-03', meal_id=4, price=9, type=u'italian')
, Row(dt=u'2013-01-03', meal_id=5, price=12, type=u'chinese')]
```

Spark SQL : More complex query

```
>>> sqlContext.sql("""
...     SELECT type, COUNT(type) AS cnt FROM
...     meals
...     INNER JOIN
...     events on meals.meal_id = events.meal_id
...     WHERE
...     event = 'bought'
...     GROUP BY
...     type
...     ORDER BY cnt DESC
...     """).collect()
```

```
[Row(type=u'italian', cnt=22575), Row(type=u'french', cnt=16179), Row(type=u'mex
ican', cnt=8792), Row(type=u'japanese', cnt=6921), Row(type=u'chinese', cnt=6267
), Row(type=u'vietnamese', cnt=3535)]
```

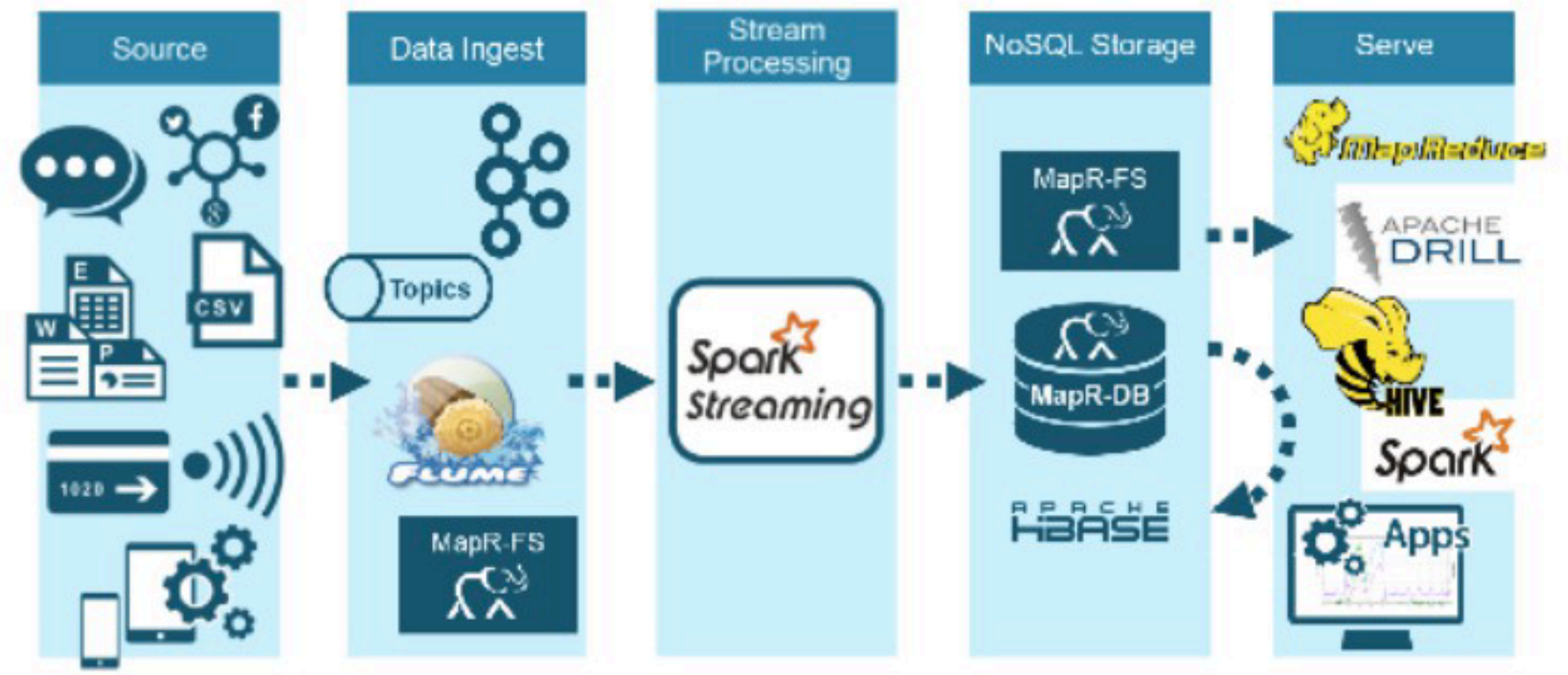
CHULALONGKORN
BUSINESS SCHOOL
FLAGSHIP FOR LIFE

Spark Streaming

Thaveewat Khanan

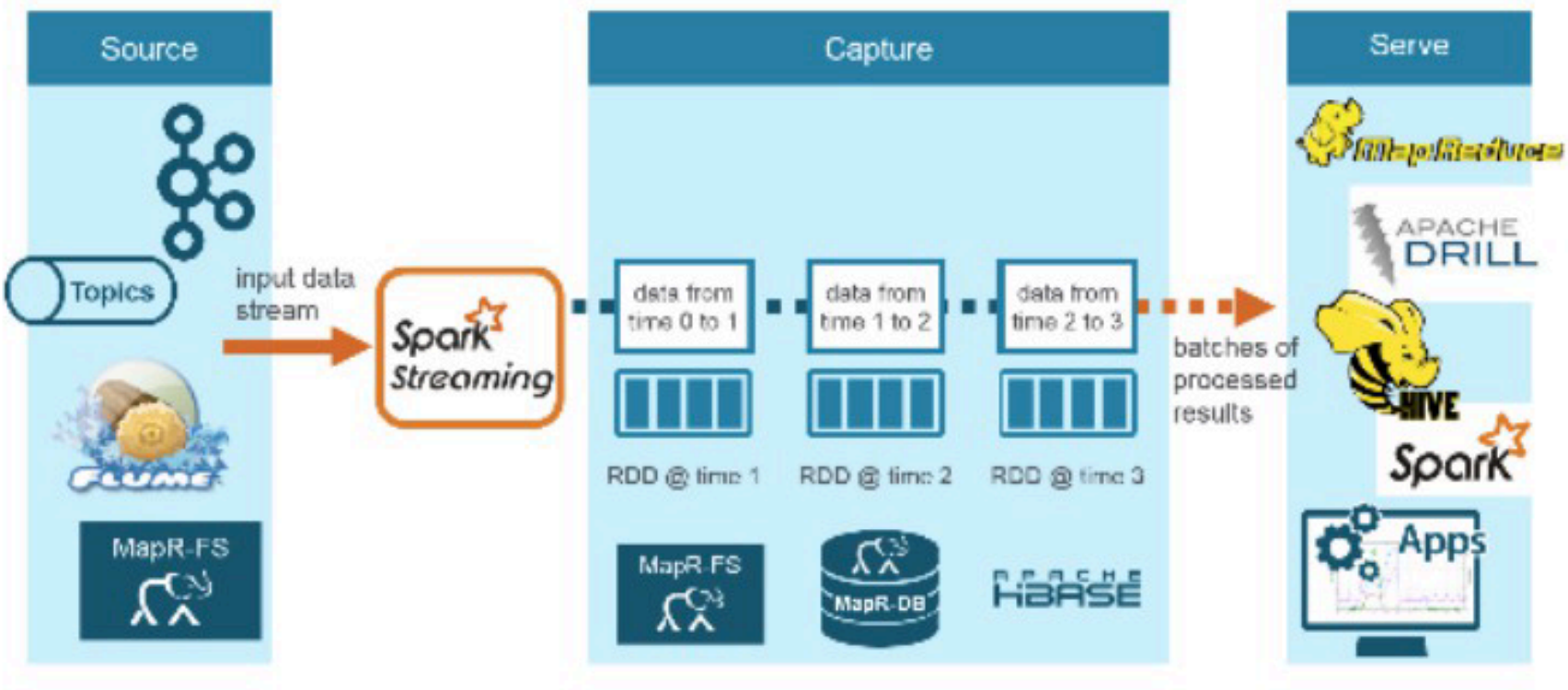
2603615 Organization Data Management

Stream Process Architecture



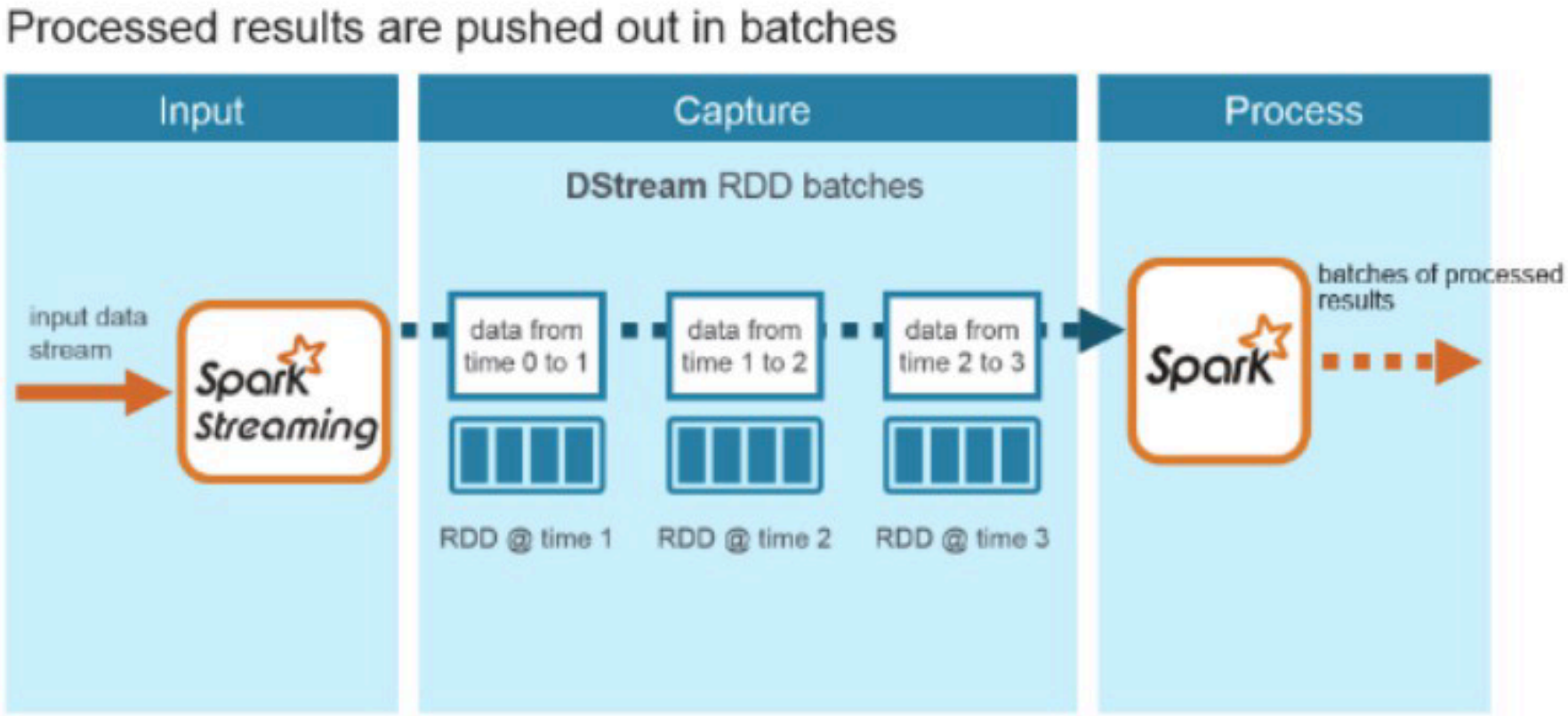
Source: MapR Academy

Spark Streaming Architecture



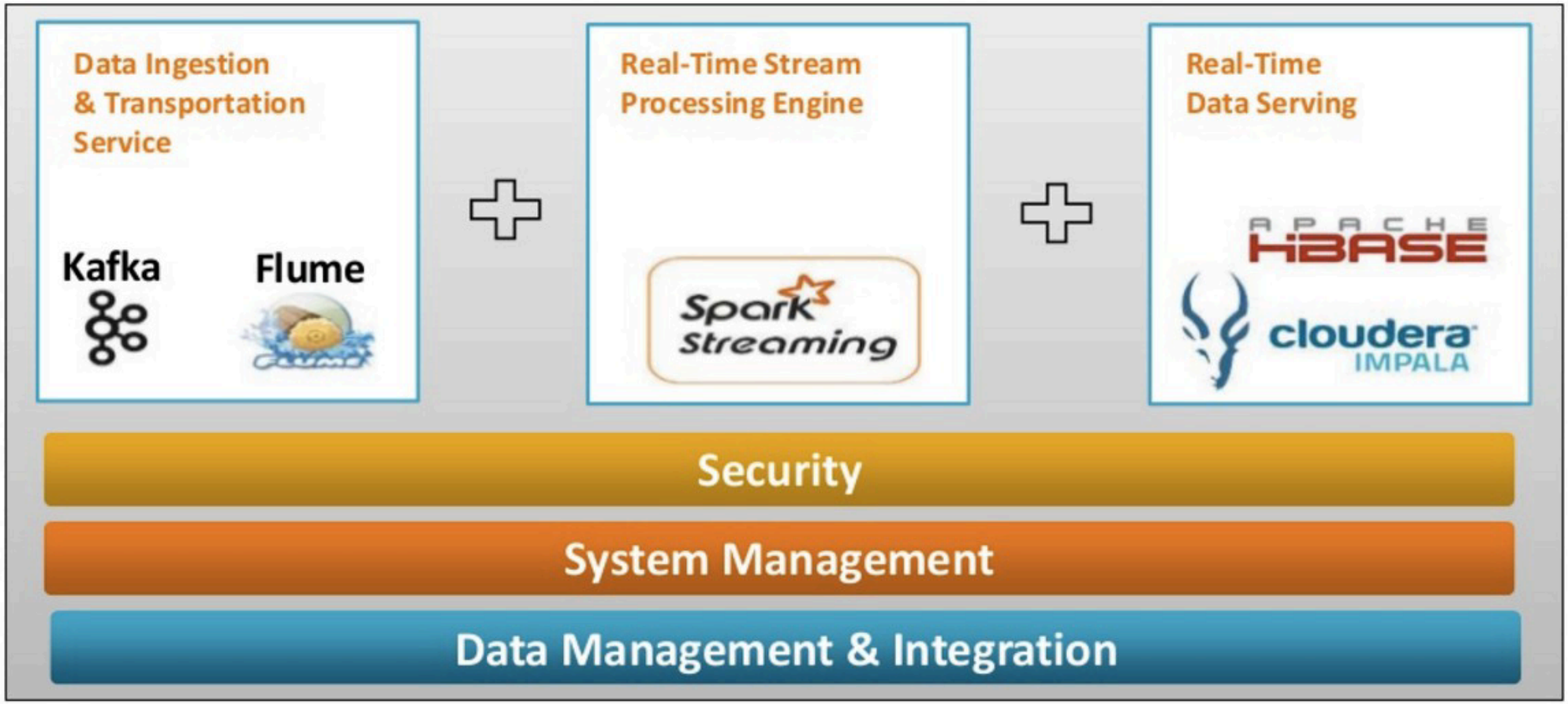
Source: MapR Academy

Processing Spark DStreams



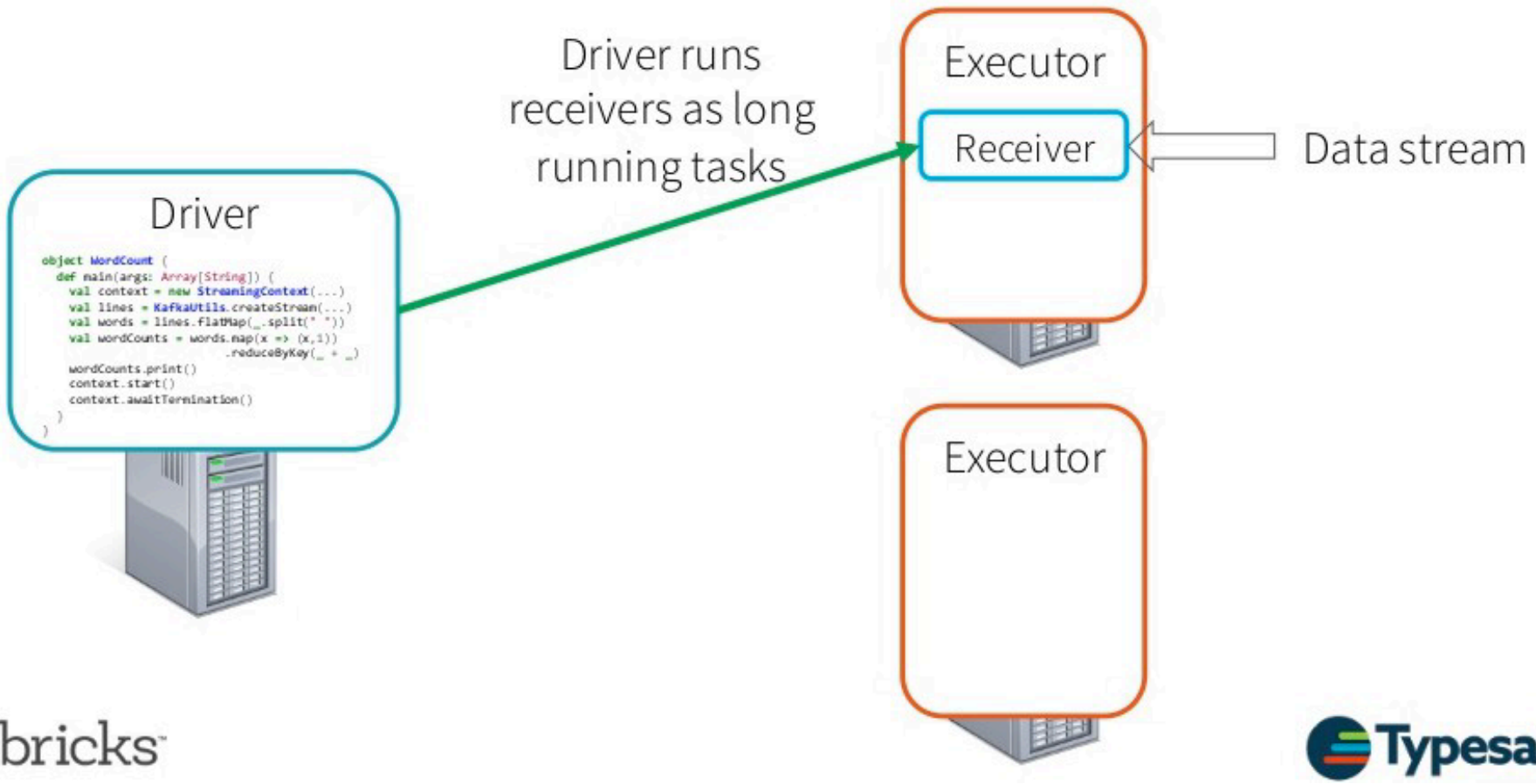
Source: MapR Academy

Streaming Architecture



Source: Cloudera

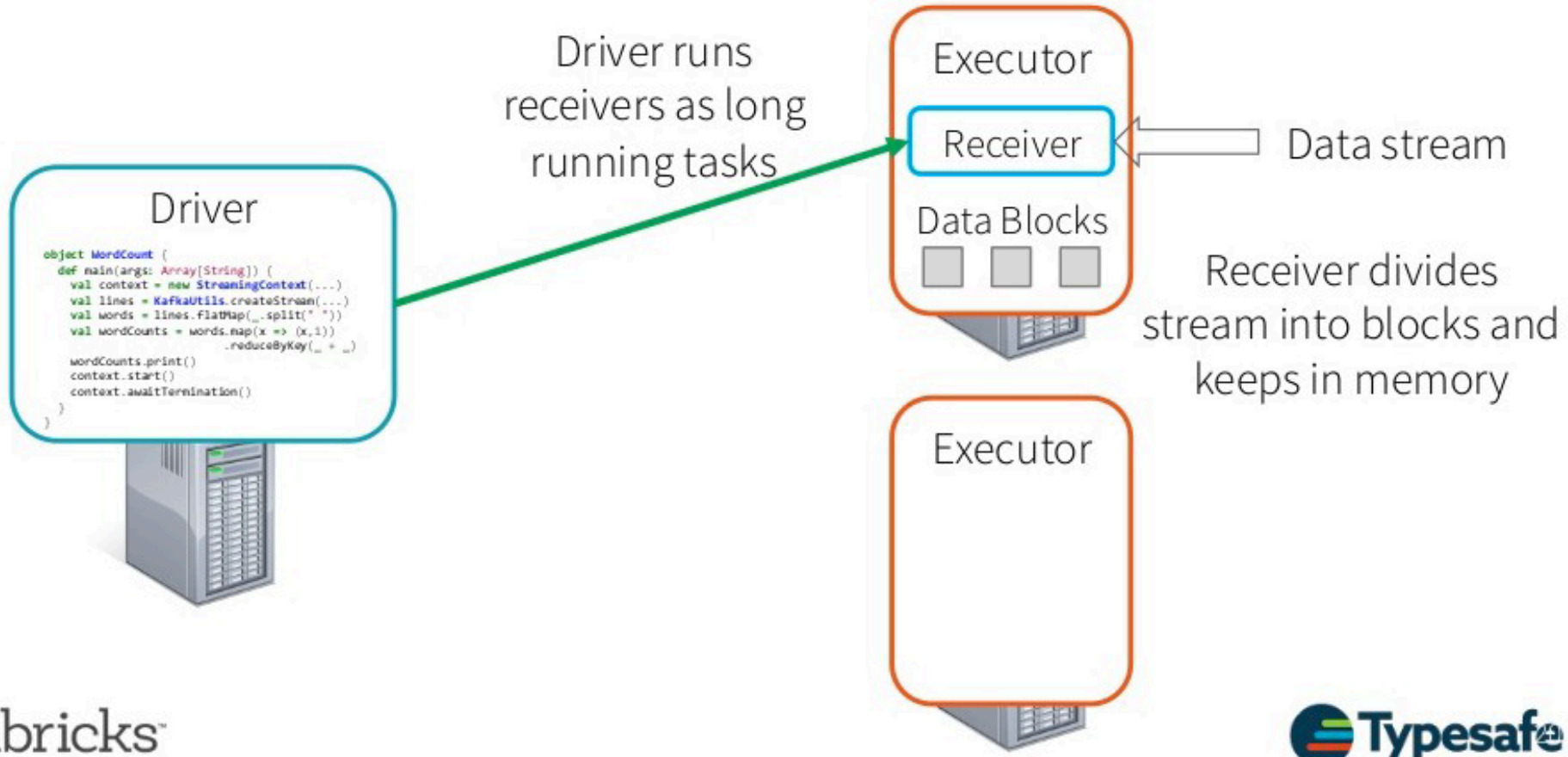
Spark Streaming Application: Receive data



databricks

Typesafe

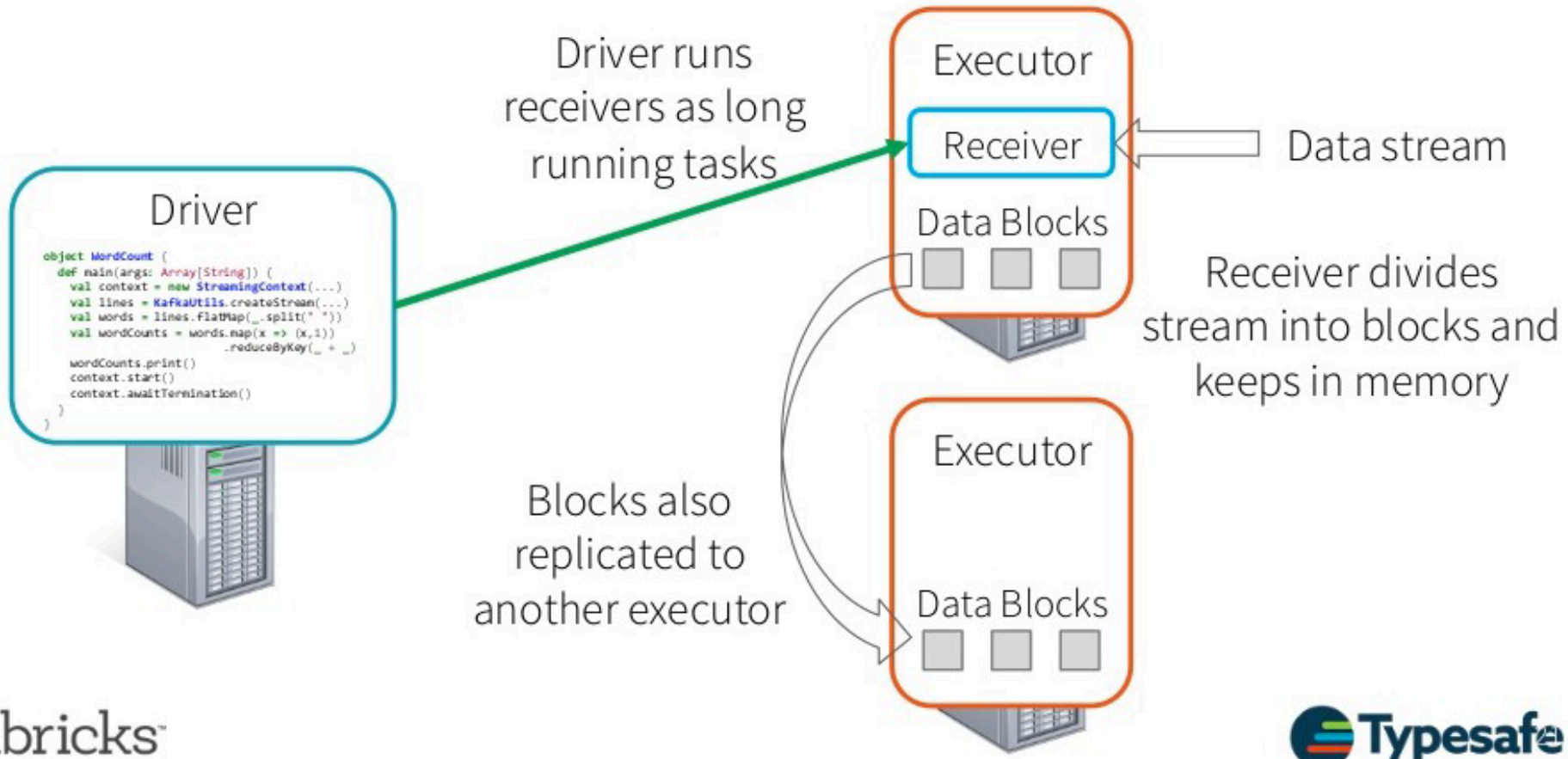
Spark Streaming Application: Receive data



databricks

Typesafe

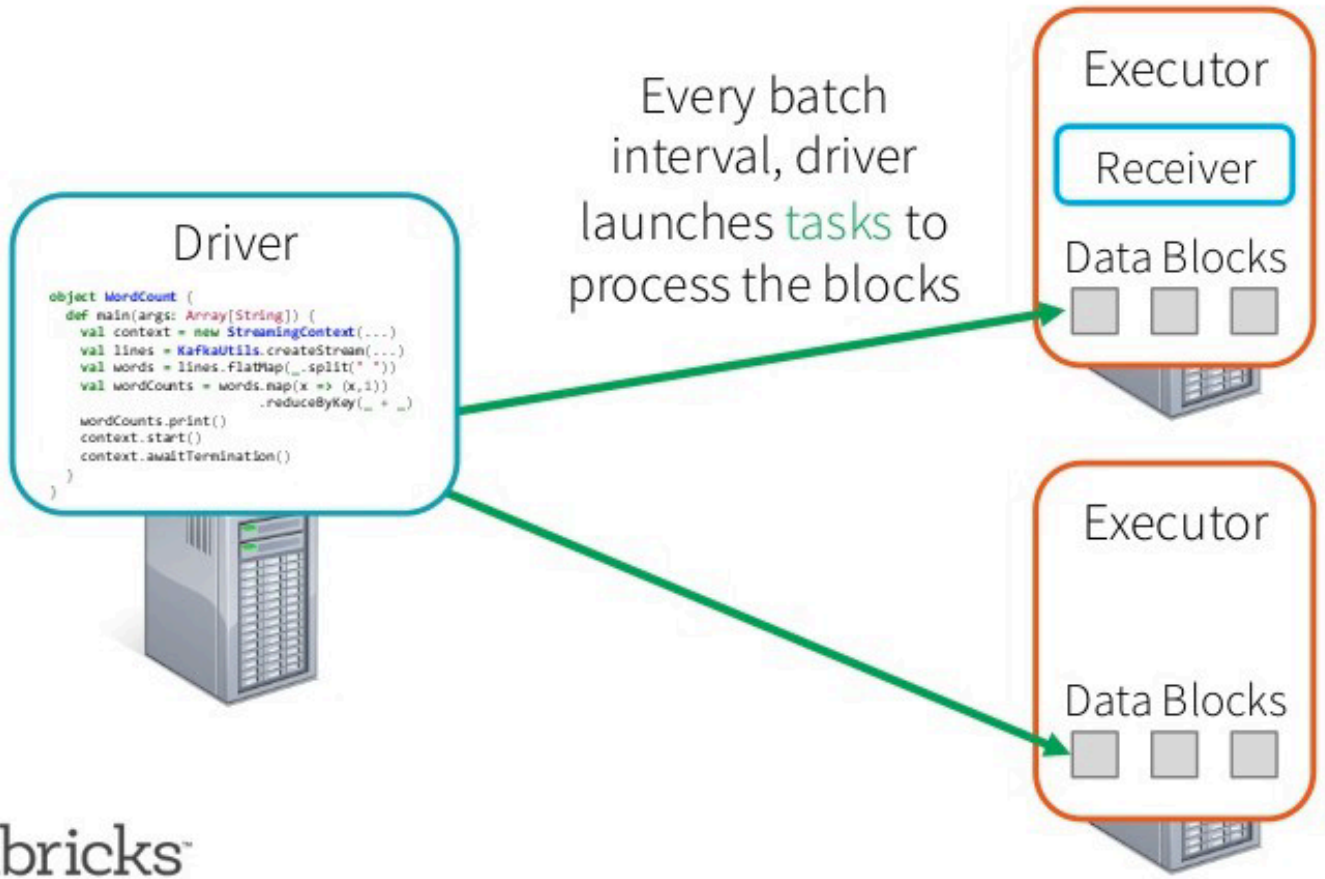
Spark Streaming Application: Receive data



databricks

Typesafe

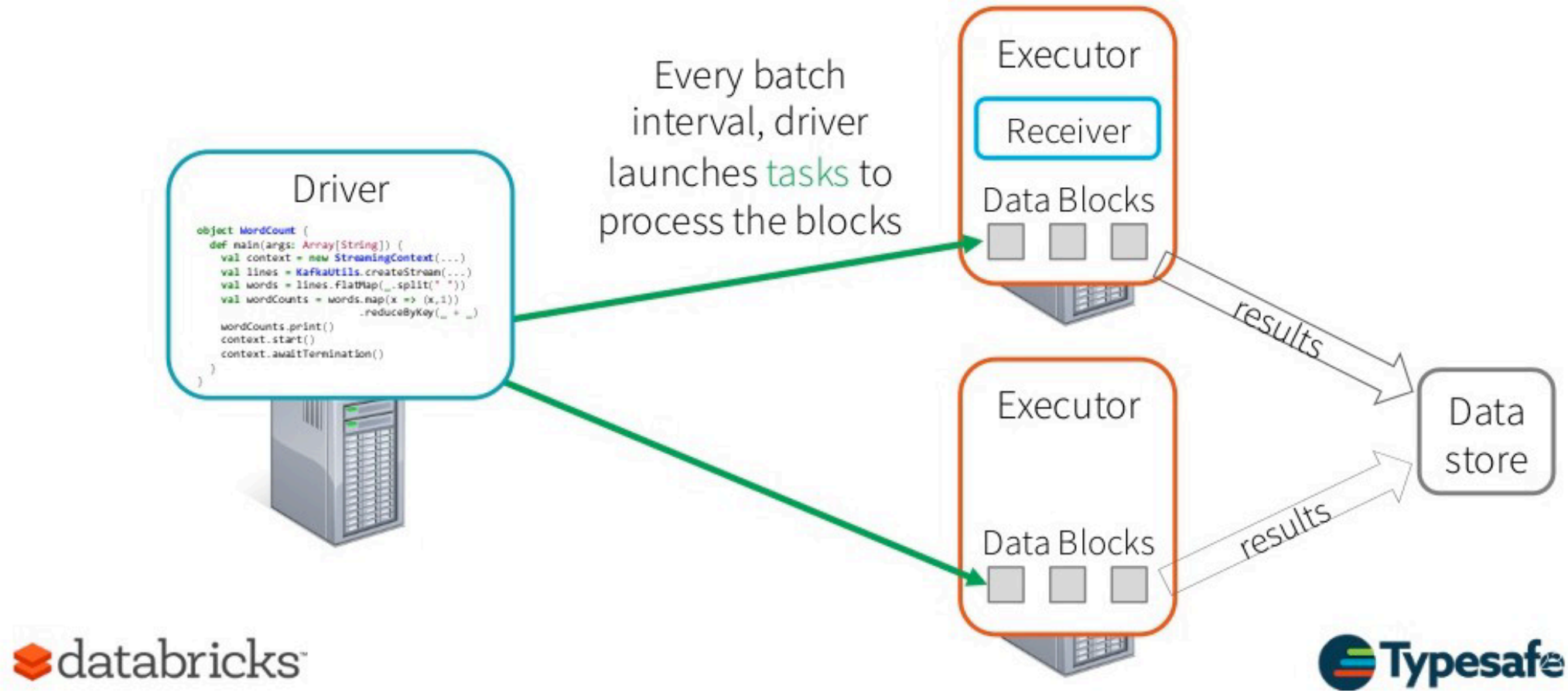
Spark Streaming Application: Process data



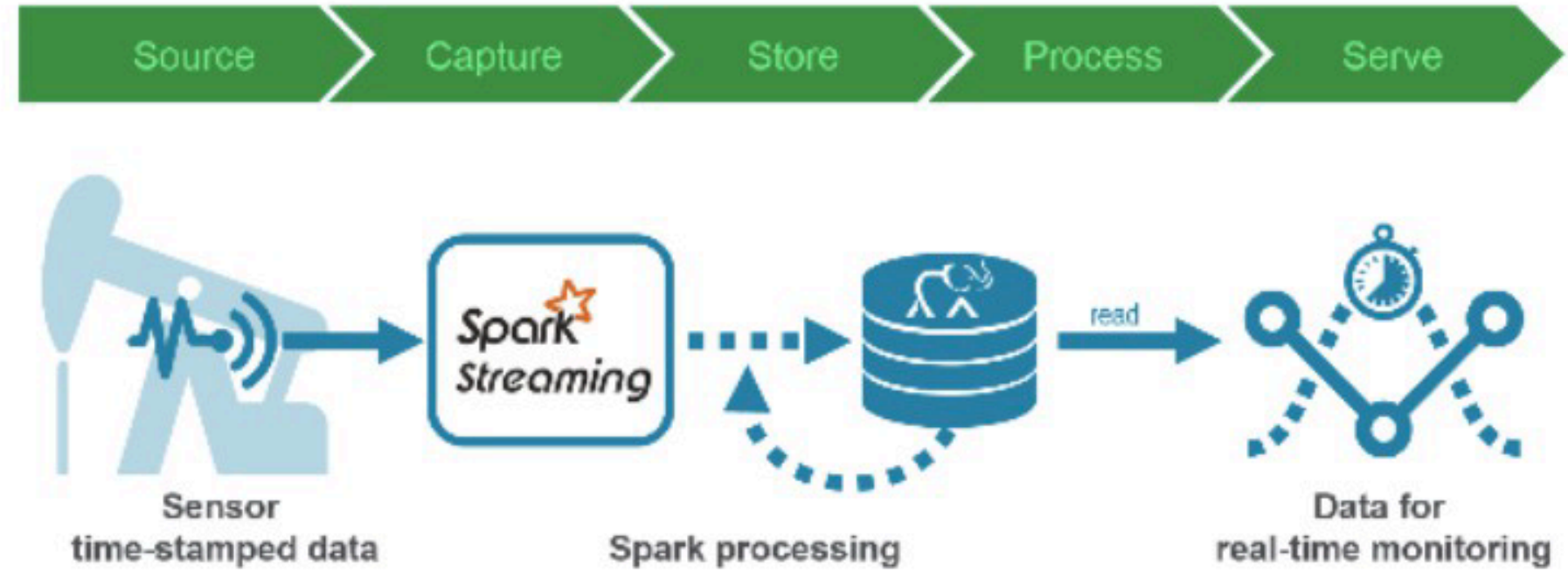
databricks

Typesafe

Spark Streaming Application: Process data



Use Case: Time Series Data



Source: MapR Academy

DStream Functional operations

flatMap(flatMapFunc)

filter(filterFunc)

map(mapFunc)

mapPartitions(mapPartFunc, preservePartitioning)

forEachRDD(forEachFunc

DStream Output operations

print()

saveAsHadoopFiles(...)

saveAsTextFiles(...)

saveAsObjectFiles(...)

saveAsNewAPIHadoopFiles(...)

forEachRDD(..)

Spark Streaming

CHULALONGKORN BUSINESS SCHOOL
FLAGSHIP FOR LIFE

Get Example Code

$ wget https://github.com/bobbylovemovie/trainbigdata/raw/master/Spark/StreamingWordCount.py

```
1  from pyspark import SparkContext
2  from pyspark.streaming import StreamingContext
3
4  # Create a local StreamingContext with two working thread and batch interval of 1 second
5  sc = SparkContext("local[2]", "NetworkWordCount")
6  ssc = StreamingContext(sc, 10)
7
8  lines = ssc.socketTextStream("localhost", 9999)
9  words = lines.flatMap(lambda line: line.split(" "))
10
11 # Count each word in each batch
12 pairs = words.map(lambda word: (word, 1))
13 wordCounts = pairs.reduceByKey(lambda x, y: x + y)
14
15 # Print the first ten elements of each RDD generated in this DStream to the console
16 wordCounts.pprint()
17 #wordCounts.saveAsTextFiles("hdfs:///user/cloudera/output/sparkstream/sparkstream")
18 ssc.start()             # Start the computation
19 ssc.awaitTermination()  # Wait for the computation to terminate
```

Spark Streaming

CHULALONGKORN BUSINESS SCHOOL
FLAGSHIP FOR LIFE

Run Python Spark

$ spark-submit StreamingWordCount.py

Running the netcat server on another window

$ nc -lk 9999

```
[cloudera@quickstart ~]$ nc -lk 9999
Hello Bigdata Training
```

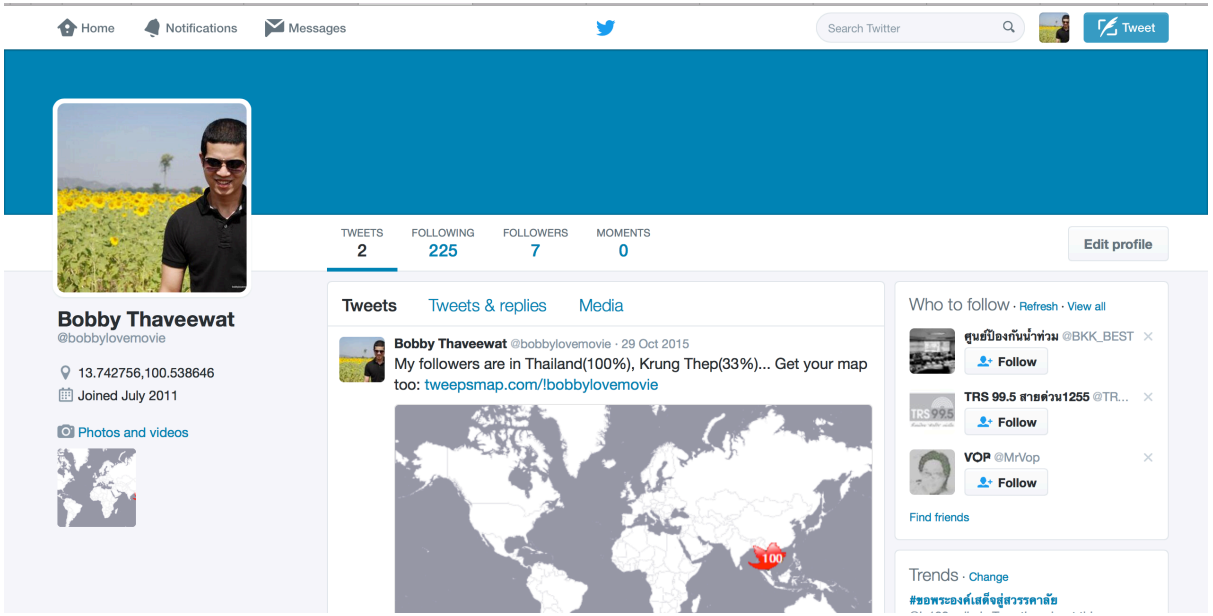
Result SparkStreaming

```
-------------------------------------------
Time: 2016-10-15 08:22:30
-------------------------------------------
(u'', 1)
(u'Training', 1)
(u'Hello', 1)
(u'Bigdata', 1)
```

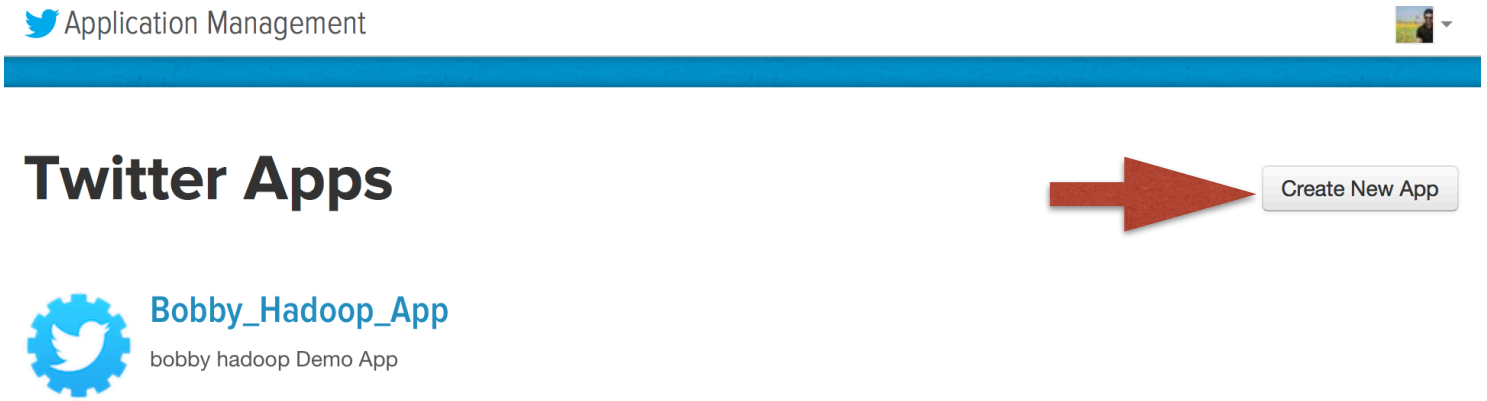
Streaming Twitter data

CHULALONGKORN BUSINESS SCHOOL
FLAGSHIP FOR LIFE

Create a new Twitter App Login to your Twitter @ twitter.com



Create a new Twitter App https://apps.twitter.com



Thaveewat Khanan
2603615 Organization Data Management

Create a new Twitter App (cont.)

CHULALONGKORN BUSINESS SCHOOL
FLAGSHIP FOR LIFE

Enter all the details in the application:

Application Details

Name *

Bobby_SparkStreaming_Demo_App

Your application name. This is used to attribute the source of a tweet and in user-facing authorization screens. 32 characters max.

Description *

Streaming Twitter data Demo App

Your application description, which will be shown in user-facing authorization screens. Between 10 and 200 characters max.

Website *

http://www.it.acc.chula.ac.th

Your application's publicly accessible home page, where users can go to download, make use of, or find out more information about your application. This fully-qualified URL is used in the source attribution for tweets created by your application and will be shown in user-facing authorization screens.
(If you don't have a URL yet, just put a placeholder here but remember to change it later.)

Callback URL

Where should we return after successfully authenticating? OAuth 1.0a applications should explicitly specify their oauth_callback URL on the request token step, regardless of the value given here. To restrict your application from using callbacks, leave this field blank.

Developer Agreement

☑ Yes, I have read and agree to the Twitter Developer Agreement.

Create a new Twitter App (cont.)

CHULALONGKORN BUSINESS SCHOOL
FLAGSHIP FOR LIFE

Your application will be created:

Bobby_SparkStreaming_Demo_App

Test OAuth

Details | Settings | Keys and Access Tokens | Permissions

Streaming Twitter data Demo App

http://www.it.acc.chula.ac.th

Organization

Information about the organization or company associated with your application. This information is optional.

Organization	None
Organization website	None

Application Settings

Your application's Consumer Key and Secret are used to authenticate requests to the Twitter Platform.

Access level	Read and write (modify app permissions)
Consumer Key (API Key)	eSevkYKyO94uGtFxxHoGwXDpX (manage keys and access tokens)
Callback URL	None
Callback URL Locked	No
Sign in with Twitter	Yes
App-only authentication	https://api.twitter.com/oauth2/token
Request token URL	https://api.twitter.com/oauth/request_token

Create a new Twitter App (cont.)

CHULALONGKORN BUSINESS SCHOOL
FLAGSHIP FOR LIFE

Click on Keys and Access Tokens:

Bobby_SparkStreaming_Demo_App

Details | Settings | Keys and Access Tokens | Permissions

Application Settings

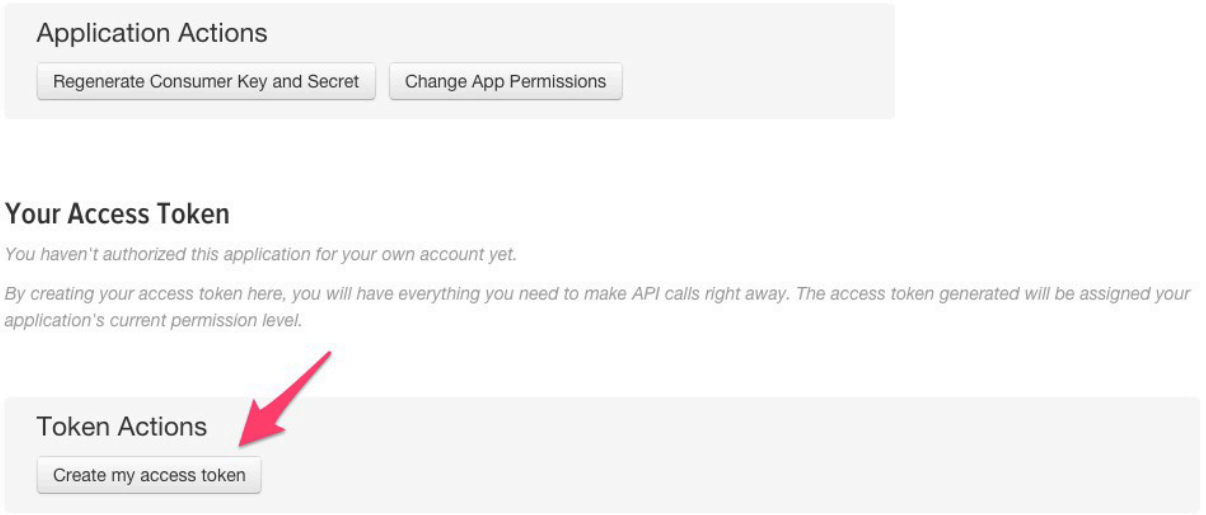
Keep the "Consumer Secret" a secret. This key should never be human-readable in your application.

Consumer Key (API Key)	eSevkYKyO94uGtFxxHoGwXDpX
Consumer Secret (API Secret)	OvjqlF8VmNScaeMGZjz9e1flfq0TxehmwkMJ454wHCCUG0AvED
Access Level	Read and write (modify app permissions)
Owner	bobbylovemovie
Owner ID	344100790

Create a new Twitter App (cont.)

CHULALONGKORN BUSINESS SCHOOL
FLAGSHIP FOR LIFE

Click on Keys and Access Tokens:



Your Access Token

This access token can be used to make API requests on your own account's behalf. Do not share your access token secret with anyone.

Access Token	344100790-rOcjGTw9wdmD5FVwacOaR7ZriGPRILBZiha43Tyg
Access Token Secret	2t41euCCQiwmdkihxkBWAS2rOTfTJpTAHUy27fOAZILWE
Access Level	Read and write
Owner	bobbylovemovie
Owner ID	344100790

Download the third-party libraries

```
$ wget https://github.com/bobbylovemovie/trainbigdata/raw/master/Spark/
twitter4j-core-4.0.2.jar
$ wget https://github.com/bobbylovemovie/trainbigdata/raw/master/Spark/
twitter4j-stream-4.0.2.jar
$ wget https://github.com/bobbylovemovie/trainbigdata/raw/master/Spark/
spark-streaming-twitter_2.10-1.2.0.jar
```

Run Spark-shell

```
$ spark-shell --jars spark-streaming-twitter_2.10-1.2.0.jar,twitter4j-
stream-4.0.2.jar,twitter4j-core-4.0.2.jar
```

Running Spark commands

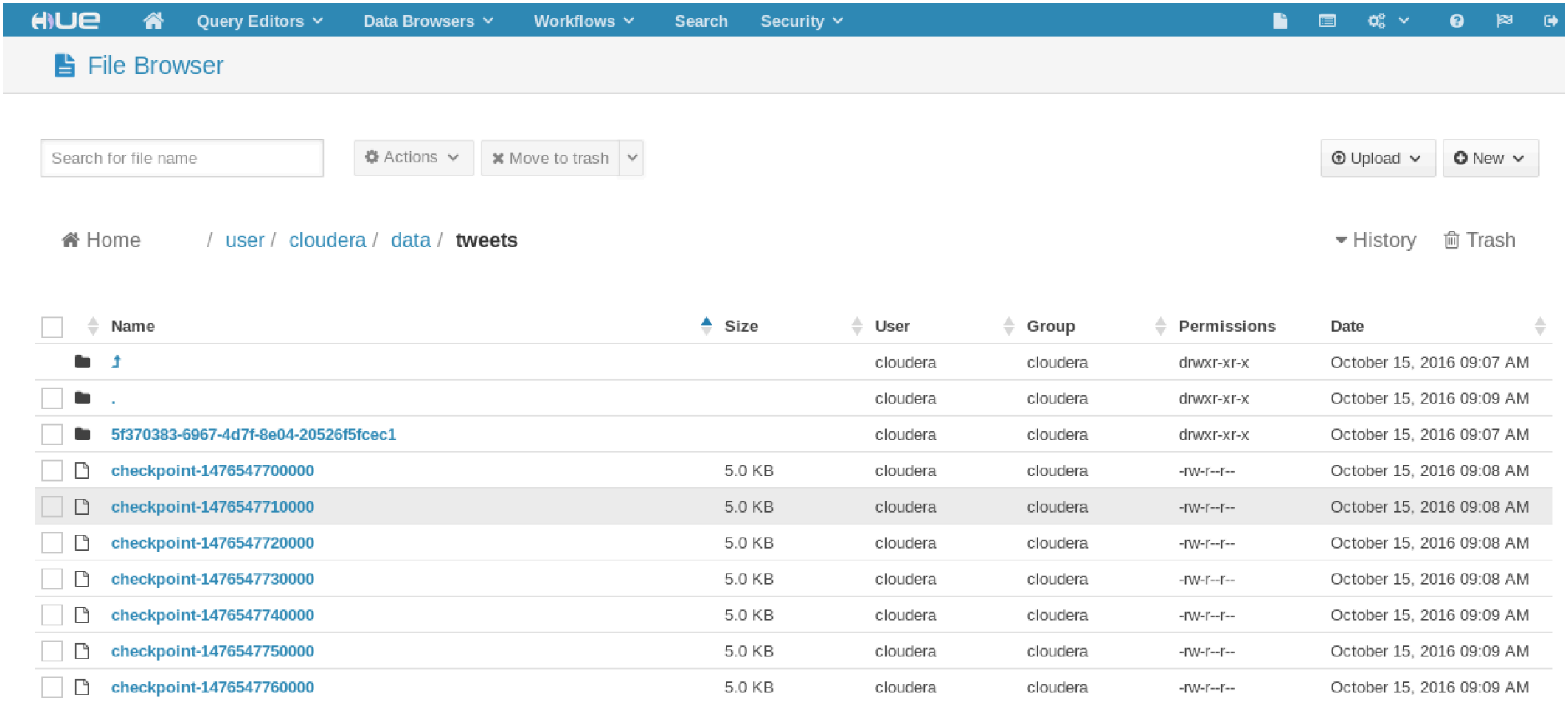
```
$ scala> :paste
// Entering paste mode (ctrl-D to finish)
import org.apache.spark.streaming.twitter._
import twitter4j.auth._
import twitter4j.conf._
import org.apache.spark.streaming.{Seconds, StreamingContext}
import org.apache.spark._
import org.apache.spark.streaming._
import org.apache.spark.streaming.StreamingContext._
val ssc = new StreamingContext(sc, Seconds(10))
val cb = new ConfigurationBuilder
```

Running Spark commands(cont.)

CHULALONGKORN BUSINESS SCHOOL
FLAGSHIP FOR LIFE

```
cb.setDebugEnabled(true).setOAuthConsumerKey("eSevkYKyO94uGtFxxHoG
wXDpX").setOAuthConsumerSecret("OvjqIF8VmNScaeMGZjz9e1flfq0TxehmW
kMJ454wHCCUG0AvED").setOAuthAccessToken("344100790-
rOcjGTw9wdmD5FVwacOaR7ZriGPRILBZiha43Tyg").setOAuthAccessTokenSe
cret("2t41euCCQiwmdkihxkBWAS2rOTfTJpTAHUy27fOAZILWE")
val auth = new OAuthAuthorization(cb.build)
val tweets = TwitterUtils.createStream(ssc,Some(auth))
val status = tweets.map(status => status.getText)
status.print
ssc.checkpoint("hdfs:///user/cloudera/data/tweets")
ssc.start
ssc.awaitTermination
```

CHULALONGKORN BUSINESS SCHOOL
FLAGSHIP FOR LIFE



	Name	Size	User	Group	Permissions	Date
	..		cloudera	cloudera	drwxr-xr-x	October 15, 2016 09:07 AM
	.		cloudera	cloudera	drwxr-xr-x	October 15, 2016 09:09 AM
	5f370383-6967-4d7f-8e04-20526f5fcec1		cloudera	cloudera	drwxr-xr-x	October 15, 2016 09:07 AM
	checkpoint-1476547700000	5.0 KB	cloudera	cloudera	-rw-r--r--	October 15, 2016 09:08 AM
	checkpoint-1476547710000	5.0 KB	cloudera	cloudera	-rw-r--r--	October 15, 2016 09:08 AM
	checkpoint-1476547720000	5.0 KB	cloudera	cloudera	-rw-r--r--	October 15, 2016 09:08 AM
	checkpoint-1476547730000	5.0 KB	cloudera	cloudera	-rw-r--r--	October 15, 2016 09:08 AM
	checkpoint-1476547740000	5.0 KB	cloudera	cloudera	-rw-r--r--	October 15, 2016 09:09 AM
	checkpoint-1476547750000	5.0 KB	cloudera	cloudera	-rw-r--r--	October 15, 2016 09:09 AM
	checkpoint-1476547760000	5.0 KB	cloudera	cloudera	-rw-r--r--	October 15, 2016 09:09 AM

Thaveewat Khanan
2603615 Organization Data Management