- Data processing

accenture

LEARNING OBJECTIVES

At the end of this unit, you should be able to:

- Batch Processing
- Real Time Processing
- Batch Vs Real time processing

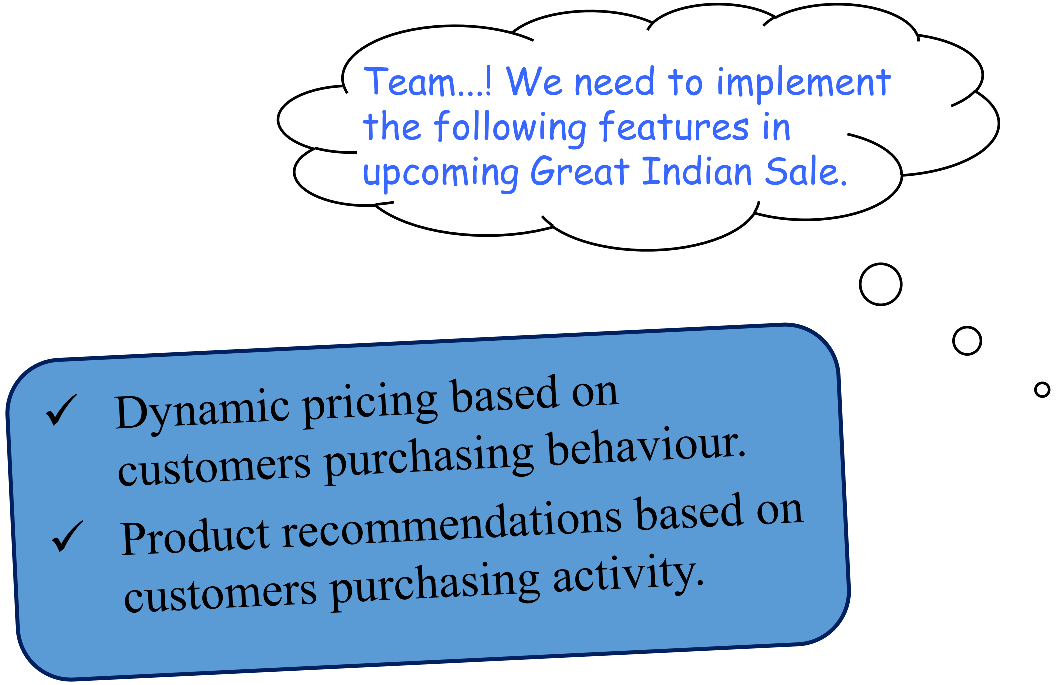
EVERY NOW AND THEN, ALMOST IN EVERY CORNER OF THE WORLD..... THIS HAPPENS..!!

Hey Bob..! Why everybody prefers buying products online?

Tina...! Bcz.. They get so many deals and many products are available in different categories.

Well !.. And also they get recommendations based on their purchasing behavior and product reviews...

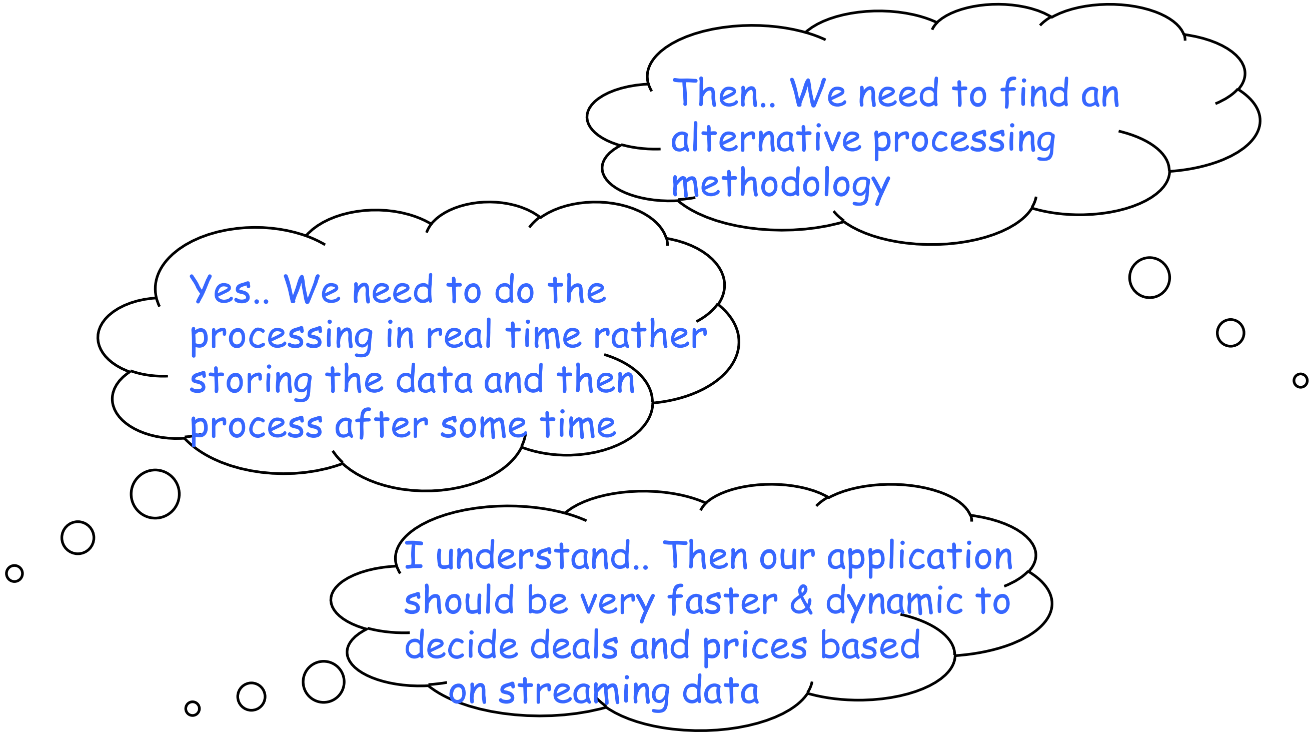
EVERY NOW AND THEN, ALMOST IN EVERY CORNER OF THE WORLD….. THIS HAPPENS..!!



Team…! We need to implement the following features in upcoming Great Indian Sale.

- ✓ Dynamic pricing based on customers purchasing behaviour.
- ✓ Product recommendations based on customers purchasing activity.

AT THE SAME TIME…. IN AN ECOMMERCE APPLICATION DEVELOPMENT & ANALYTICS TEAM

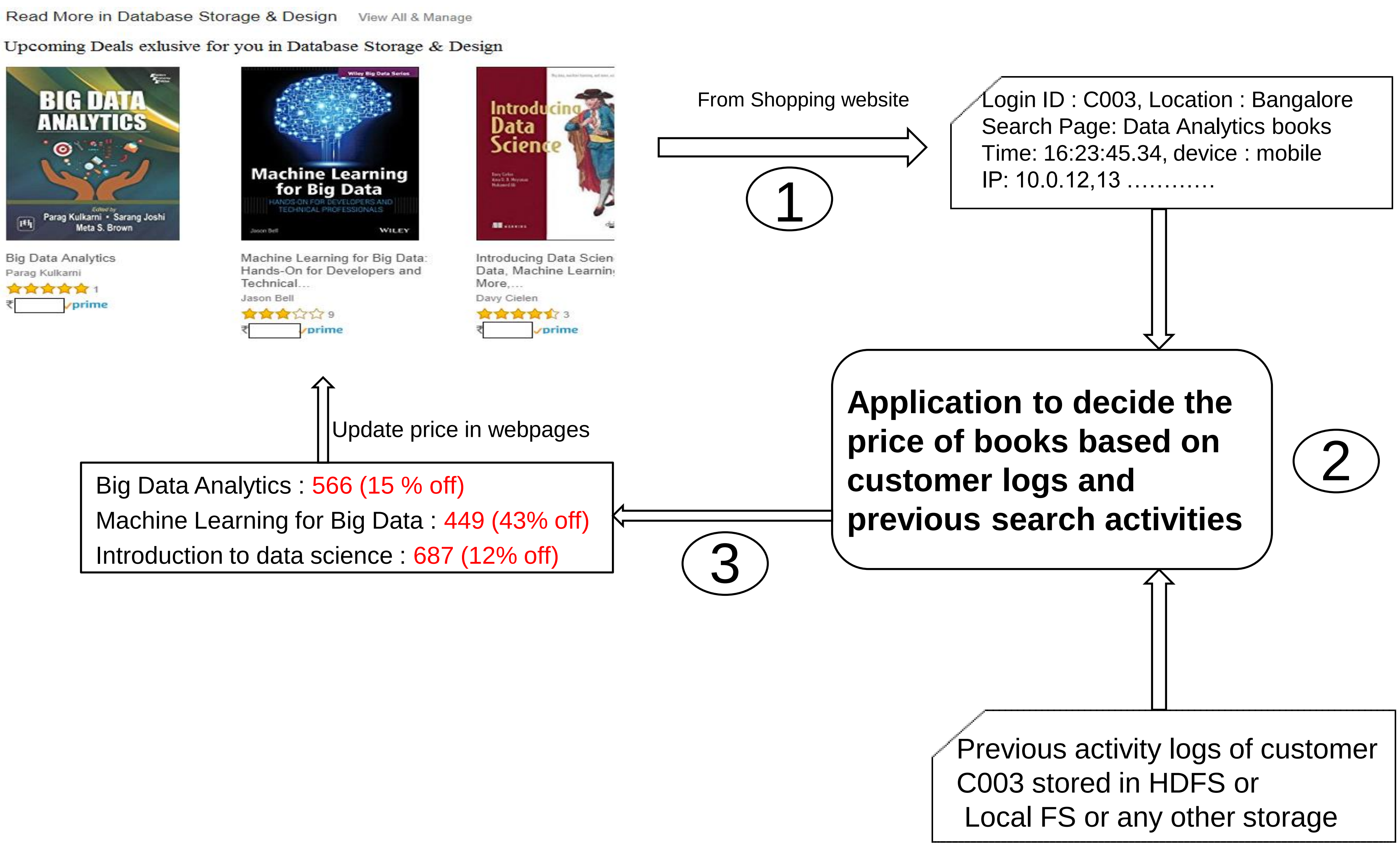


I understand.. Then our application should be very faster & dynamic to decide deals and prices based on streaming data

Yes.. We need to do the processing in real time rather storing the data and then process after some time

Then.. We need to find an alternative processing methodology

LOOKING AT THE SCENARIO, WHAT IS REQUIRED?



Read More in Database Storage & Design View All & Manage

Upcoming Deals exlusive for you in Database Storage & Design

BIG DATA ANALYTICS
Parag Kulkarni • Sarang Joshi
Meta S. Brown
Wiley Big Data Series
Machine Learning for Big Data
HANDS-ON FOR DEVELOPERS AND TECHNICAL PROFESSIONALS
Jason Bell
WILEY
Introducing Data Science

Big Data Analytics
Parag Kulkarni
★★★★★ 1
₹ prime

Machine Learning for Big Data: Hands-On for Developers and Technical...
Jason Bell
★★★☆☆ 9
₹ prime

Introducing Data Scien
Data, Machine Learnin
More,...
Davy Cielen
★★★★☆ 3
₹ prime

From Shopping website

1

Login ID : C003, Location : Bangalore
Search Page: Data Analytics books
Time: 16:23:45.34, device : mobile
IP: 10.0.12,13 …………

Application to decide the price of books based on customer logs and previous search activities

2

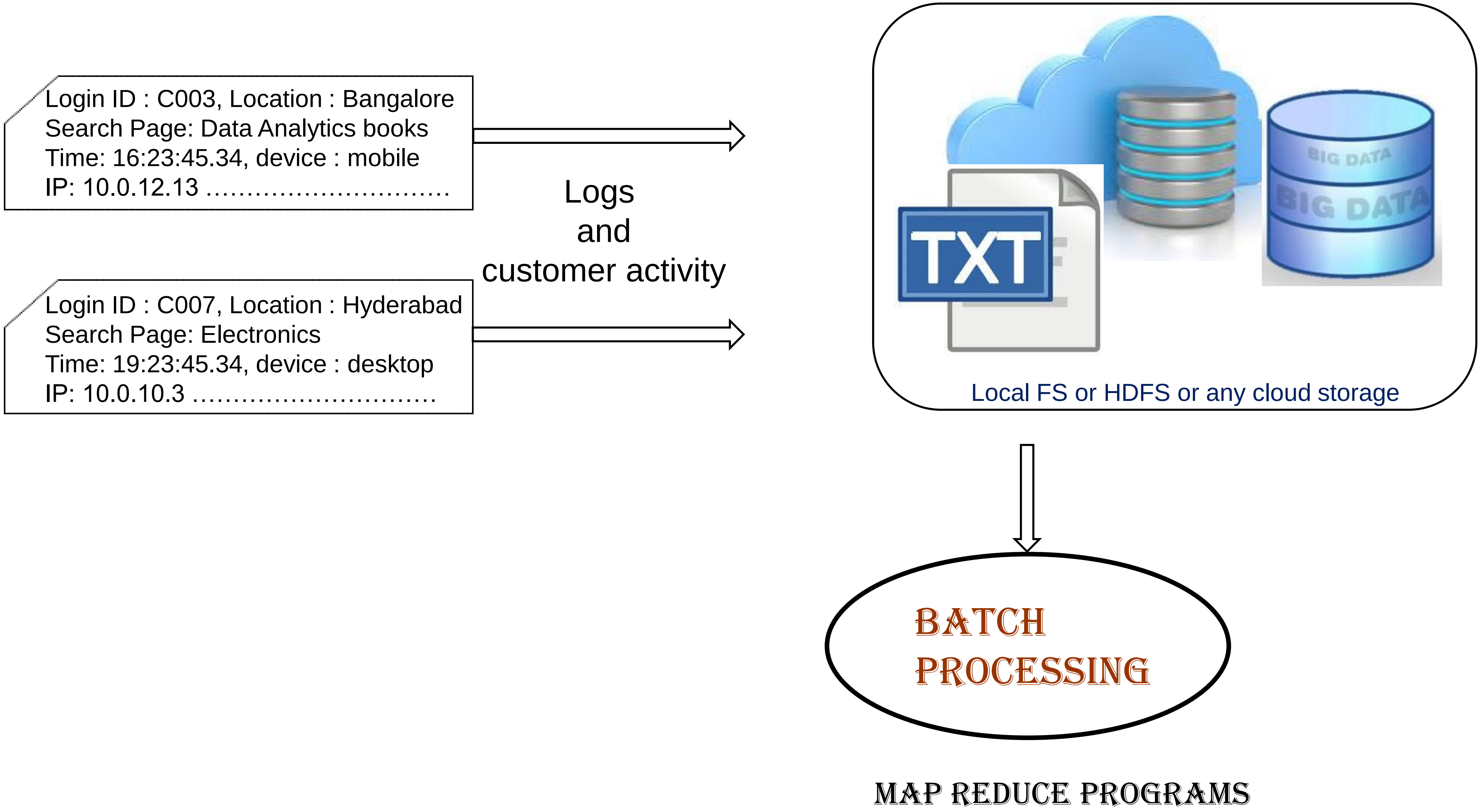
Previous activity logs of customer C003 stored in HDFS or
Local FS or any other storage

3

Big Data Analytics : 566 (15 % off)
Machine Learning for Big Data : 449 (43% off)
Introduction to data science : 687 (12% off)

Update price in webpages

WHAT WE HAVE DONE SO FAR ?



Login ID : C003, Location : Bangalore
Search Page: Data Analytics books
Time: 16:23:45.34, device : mobile
IP: 10.0.12.13 …………………………

Login ID : C007, Location : Hyderabad
Search Page: Electronics
Time: 19:23:45.34, device : desktop
IP: 10.0.10.3 …………………………

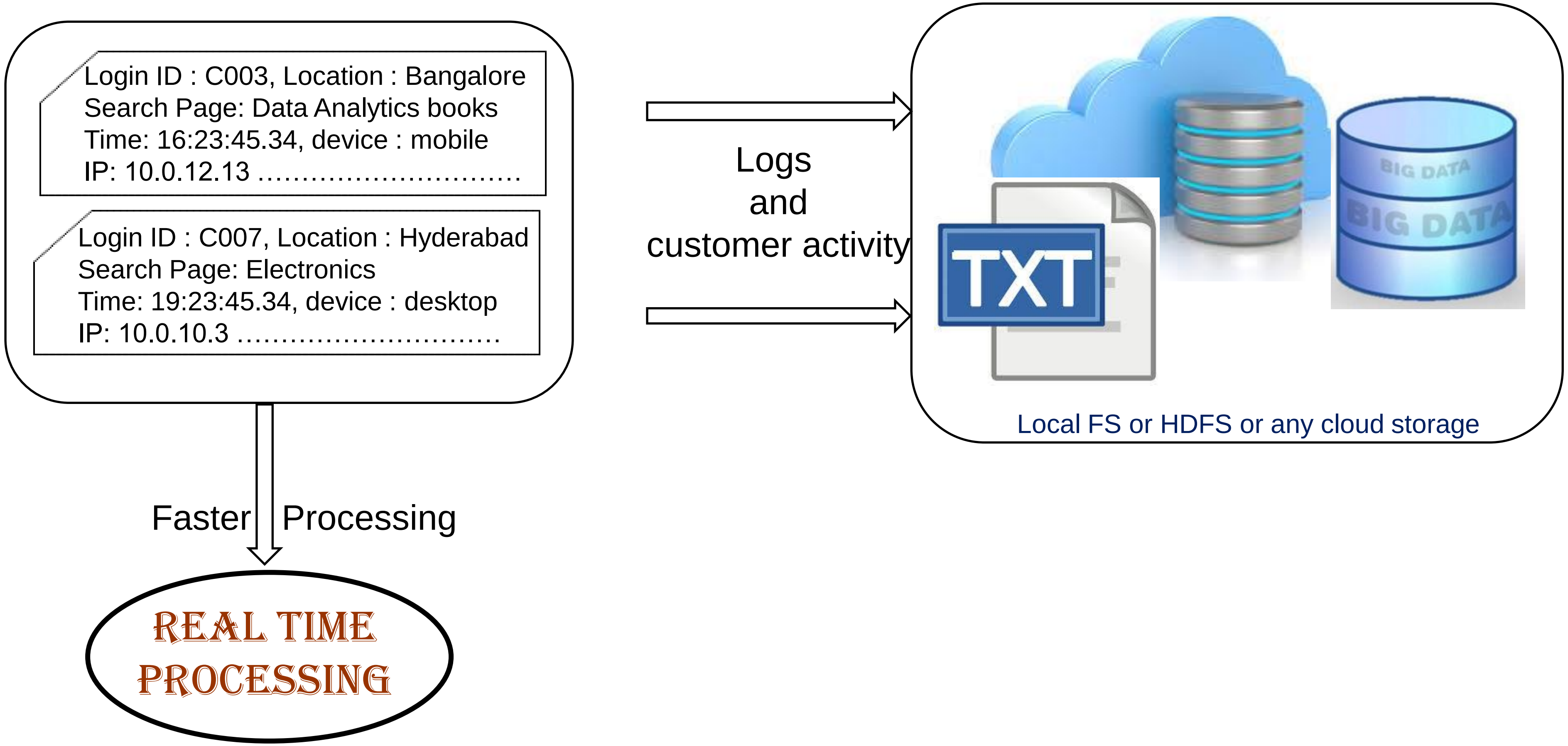
Logs
and
customer activity

Local FS or HDFS or any cloud storage

BATCH
PROCESSING

MAP REDUCE PROGRAMS

WHAT WE NEED TO DO?



```
Login ID : C003, Location : Bangalore
Search Page: Data Analytics books
Time: 16:23:45.34, device : mobile
IP: 10.0.12.13 …………………………

Login ID : C007, Location : Hyderabad
Search Page: Electronics
Time: 19:23:45.34, device : desktop
IP: 10.0.10.3 …………………………
```

Logs and customer activity

Local FS or HDFS or any cloud storage

Faster Processing

REAL TIME PROCESSING

HERE WE HAVE…

APACHE SPARK

The Framework for
Faster Batch & Real Time Data Processing

- Apache Spark

LEARNING OBJECTIVES

At the end of this unit, you should be able to:

- What is Spark?
- Why Spark?
- Spark Vs Hadoop

What is Spark ?

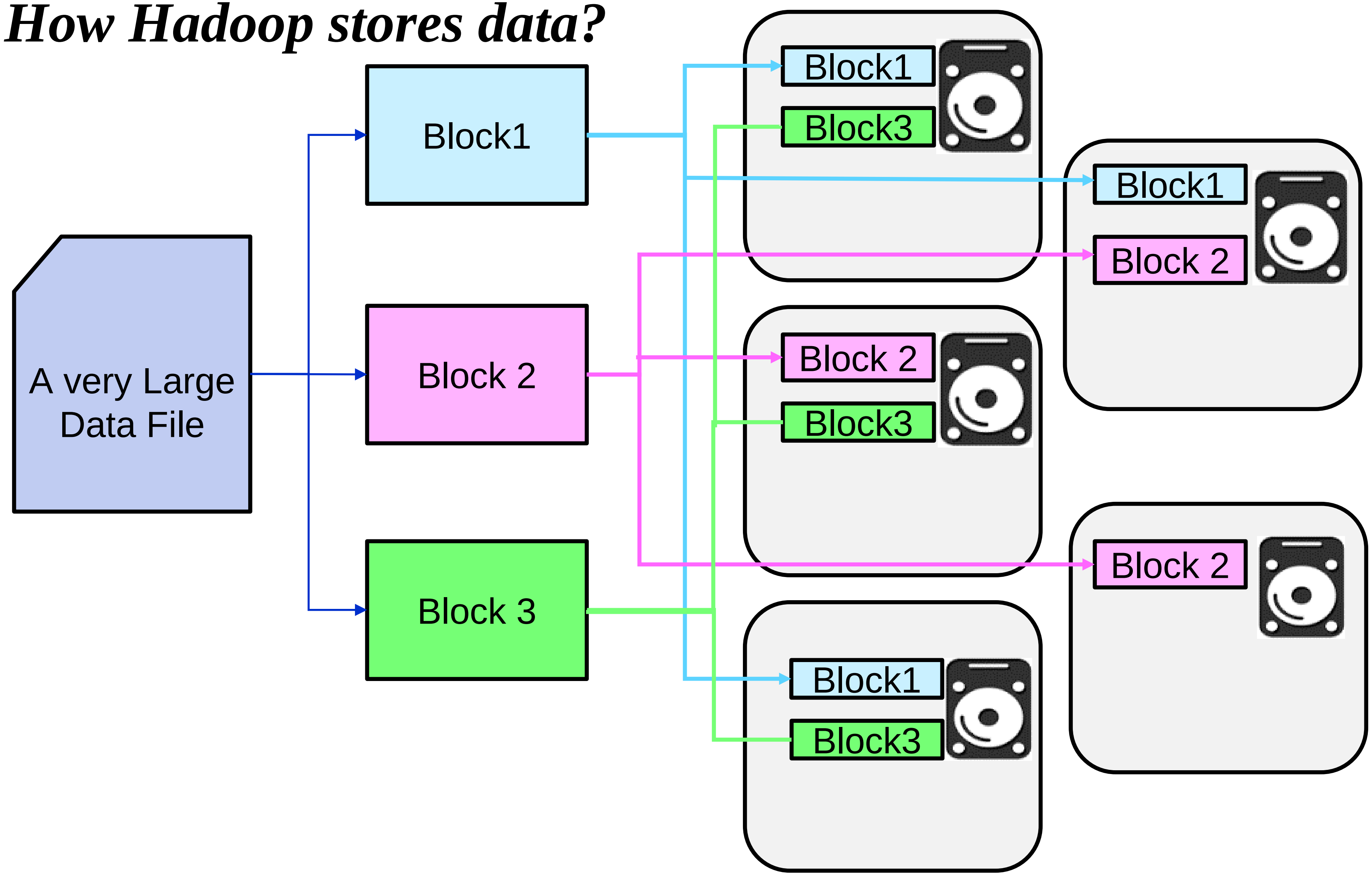
- Open source fast and general engine for large data processing.
- Developed in 2009 in UC Berkeley’s AMP Lab, and open sourced in 2010 as an Apache project.
- Written in Scala.
- In memory processing framework.
- Provides high-level APIs in Java, Scala, Python and R.

Why Spark ?

- Unified framework to manage big data requirements.
- Provides high level operators such as filter, map, etc..
- 100x faster in memory, 10x faster on disk.
- Spark Shell
  - - Interactive - for data exploration and testing.
  - - Scala or Python
- Spark Applications
  - - For large scale and data processing.
  - - Scala, Python or Java

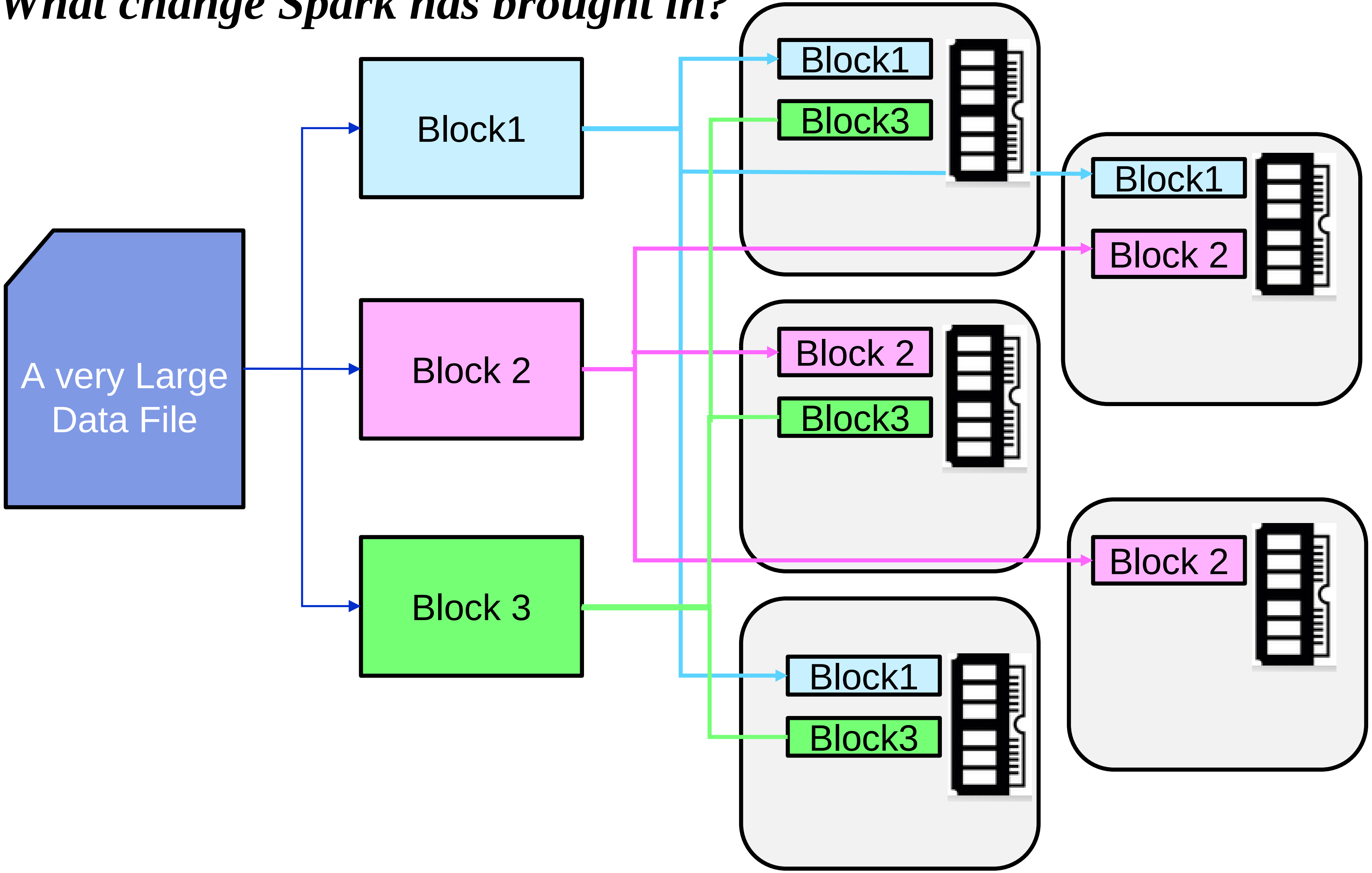
Spark Vs Hadoop

How Hadoop stores data?



Spark Vs Hadoop

What change Spark has brought in?



Spark Vs Hadoop

	Spark	**Hadoop**
Introduction	▪ Faster and general purpose data processing engine. ▪ Handles batch as well real time processing.	▪ Processes structured and unstructured data that are stored in HDFS. ▪ Handles only batch processing.

Spark Vs Hadoop

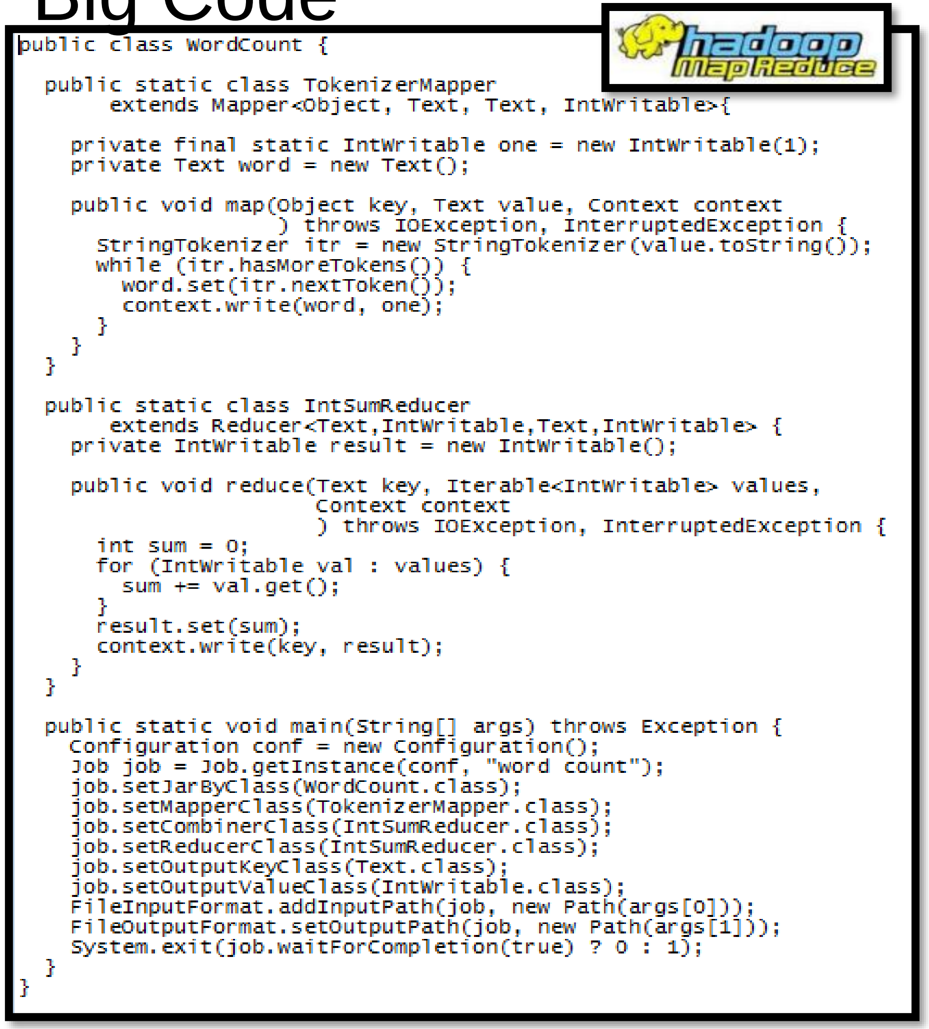
	Spark	**Hadoop**
Speed	▪ Run programs up to 100x faster than Hadoop MapReduce in memory, or 10x faster on disk. ▪ Reducing the number of read/write cycle to disk and storing intermediate data in-memory.	▪ Reads and writes from disk and that slows down the processing speed.

Spark Vs Hadoop

	Spark	**Hadoop**
Difficulty	▪ Provides high level operators. E.g. filter, map	▪ No high level operators. ▪ Need to hand code each and every operation.

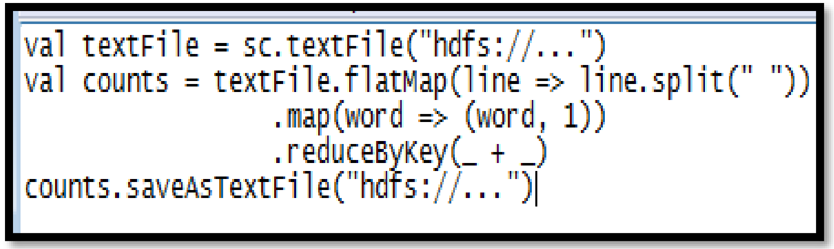
Spark Vs Hadoop

Big Code



```
public class WordCount {

  public static class TokenizerMapper
       extends Mapper<Object, Text, Text, IntWritable>{

    private final static IntWritable one = new IntWritable(1);
    private Text word = new Text();

    public void map(Object key, Text value, Context context
                    ) throws IOException, InterruptedException {
      StringTokenizer itr = new StringTokenizer(value.toString());
      while (itr.hasMoreTokens()) {
        word.set(itr.nextToken());
        context.write(word, one);
      }
    }
  }

  public static class IntSumReducer
       extends Reducer<Text,IntWritable,Text,IntWritable> {
    private IntWritable result = new IntWritable();

    public void reduce(Text key, Iterable<IntWritable> values,
                       Context context
                       ) throws IOException, InterruptedException {
      int sum = 0;
      for (IntWritable val : values) {
        sum += val.get();
      }
      result.set(sum);
      context.write(key, result);
    }
  }

  public static void main(String[] args) throws Exception {
    Configuration conf = new Configuration();
    Job job = Job.getInstance(conf, "word count");
    job.setJarByClass(WordCount.class);
    job.setMapperClass(TokenizerMapper.class);
    job.setCombinerClass(IntSumReducer.class);
    job.setReducerClass(IntSumReducer.class);
    job.setOutputKeyClass(Text.class);
    job.setOutputValueClass(IntWritable.class);
    FileInputFormat.addInputPath(job, new Path(args[0]));
    FileOutputFormat.setOutputPath(job, new Path(args[1]));
    System.exit(job.waitForCompletion(true) ? 0 : 1);
  }
}
```

Tiny Code



```
val textFile = sc.textFile("hdfs://...")
val counts = textFile.flatMap(line => line.split(" "))
                 .map(word => (word, 1))
                 .reduceByKey(_ + _)
counts.saveAsTextFile("hdfs://...")
```

- Spark Architecture

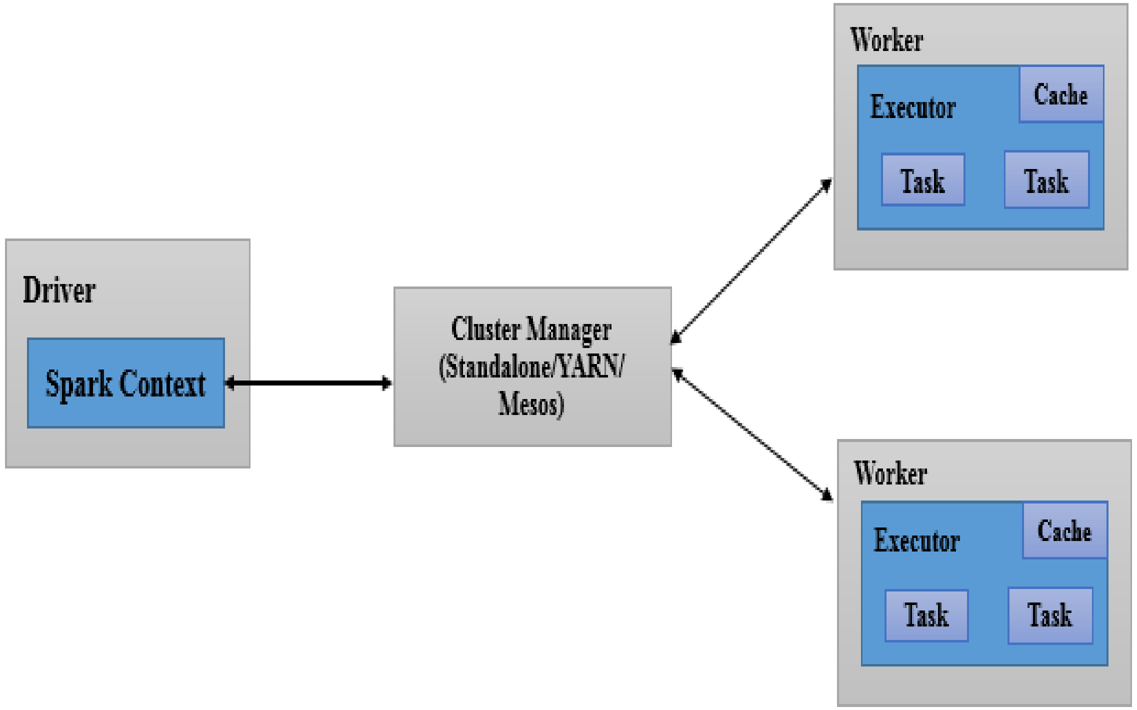
LEARNING OBJECTIVES

At the end of this unit, you should be able to:

- Architecture Overview
- Spark Cluster setup – Pseudo distributed mode
- Spark Shell
- Spark Context

ARCHITECTURE OVERVIEW

- Master/Slave architecture with cluster manager and two daemons.
- Daemons are:
  - Master – Master/ Driver Process
  - Worker – Slave Process



ARCHITECTURE OVERVIEW

- A spark cluster has single coordinator called driver and many distributed workers.
- Driver communicate with large number of distributed workers called executors.
- Cluster Manager is responsible for scheduling and allocating resources to a Spark Job.

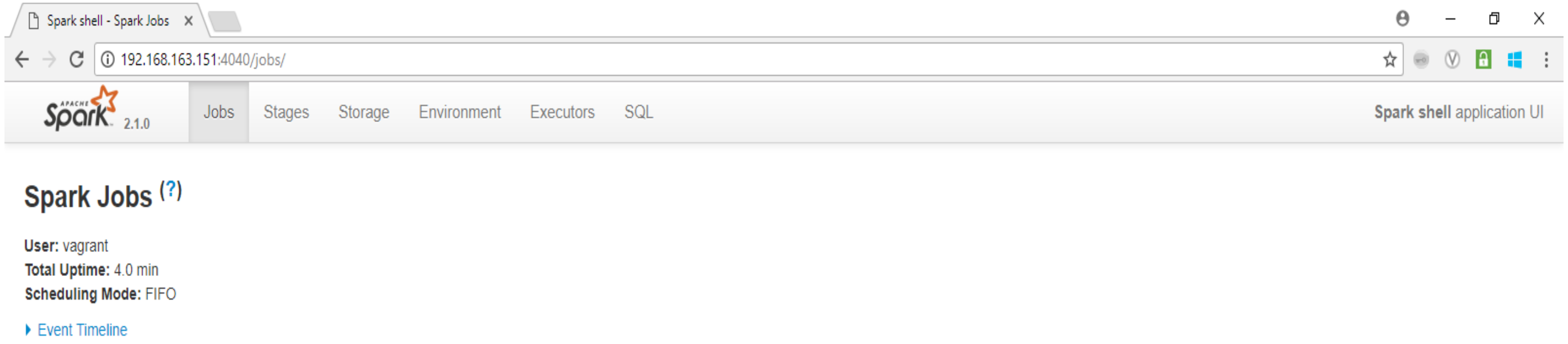
ARCHITECTURE OVERVIEW - ROLE OF DRIVER

- The driver program
  - Entry point of the Spark Shell (Scala, Python, and R).
  - Runs application main () function
  - Is the place where Spark Context is created.
- Responsible for:
  - Scheduling job, negotiating with the cluster manager.
  - Converting a user application into smaller execution units i.e. tasks.

- You can access running spark application information through default Web UI at port 4040.

ARCHITECTURE OVERVIEW - ROLE OF DRIVER

- **Spark shell application UI.**



Spark Jobs (?)

User: vagrant
Total Uptime: 4.0 min
Scheduling Mode: FIFO

▸ Event Timeline

ARCHITECTURE OVERVIEW - ROLE OF EXECUTORS

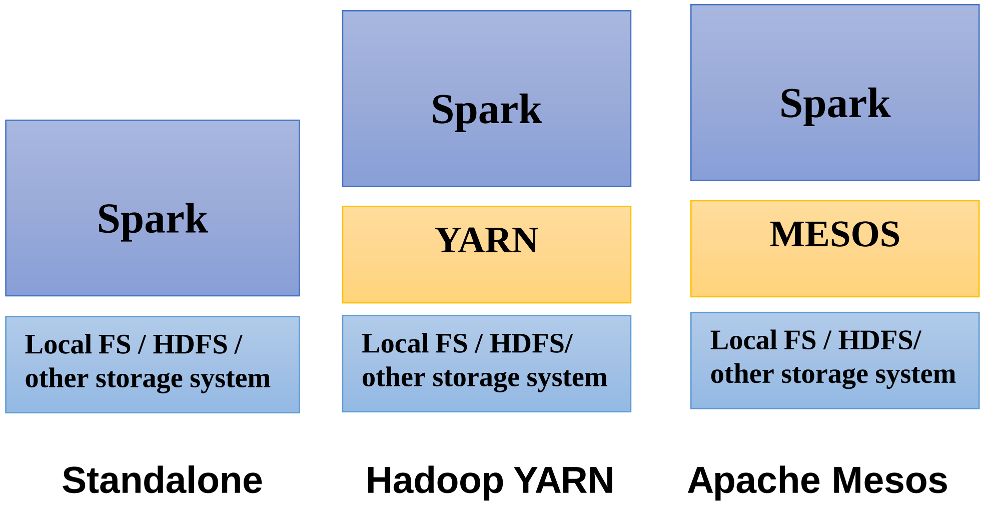
- Distributed agent.
- Responsible for:
- Executing tasks.
- Performing data processing.
- Reading from and Writing data to external sources.
- Storing computation results data in-memory, cache or disk.
- Interacting with the storage systems.

ARCHITECTURE OVERVIEW - ROLE OF CLUSTER MANAGER

- It is an external service.
- Responsible for:
  - acquiring resources and allocating them to a spark application.
  - allocation and deallocation of various physical resources such as CPU, memory, etc..,

ARCHITECTURE OVERVIEW - ROLE OF CLUSTER MANAGER

- Three types of cluster managers:
  - Standalone Cluster Manager
  - Hadoop YARN
  - Apache Mesos


Spark
Local FS / HDFS / other storage system
Standalone
Spark
YARN
Local FS / HDFS/ other storage system
Hadoop YARN
Spark
MESOS
Local FS / HDFS/ other storage system
Apache Mesos

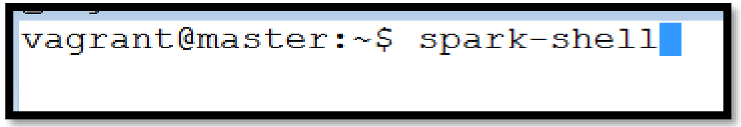
SPARK CLUSTER SETUP

Pseudo distributed mode cluster setup

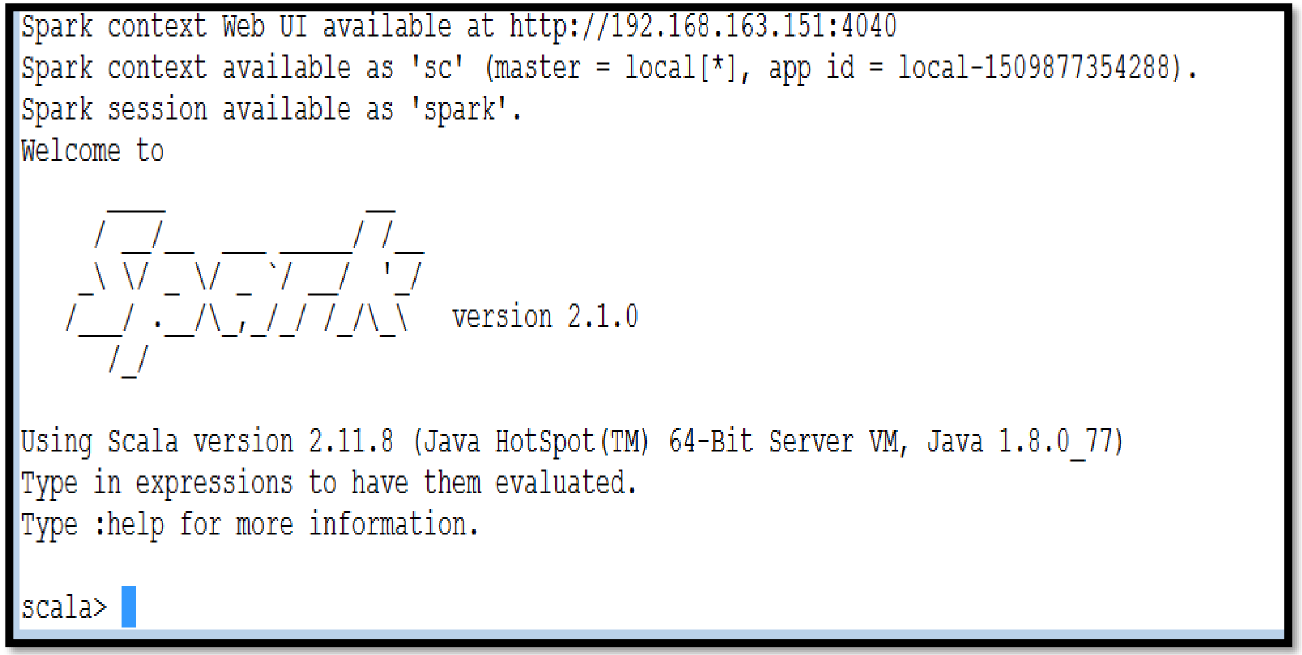
SPARK SHELL

- Provides interactive shell for data exploration and testing (REPL).
- To start Spark Shell, type spark-shell command as shown below.

Note: REPL (Read/Evaluate/Print Loop)



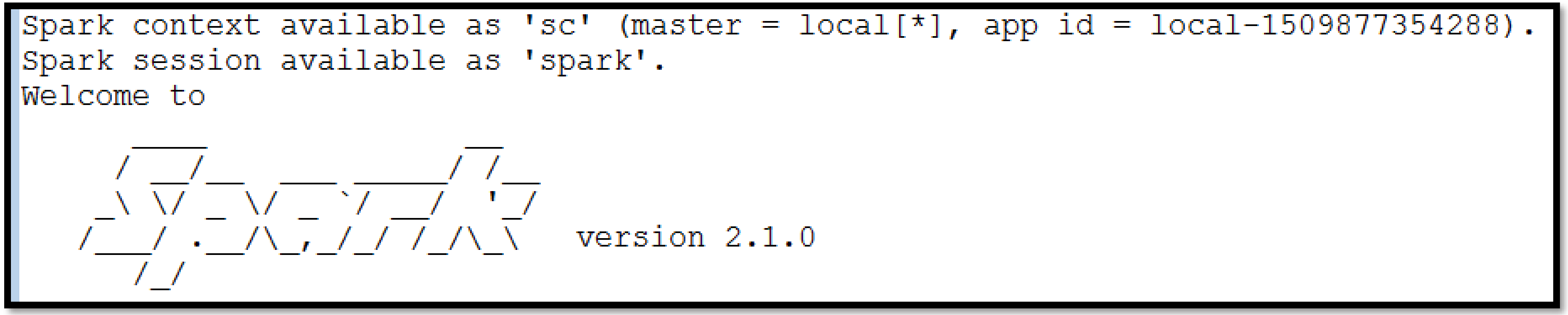
```
vagrant@master:~$ spark-shell
```



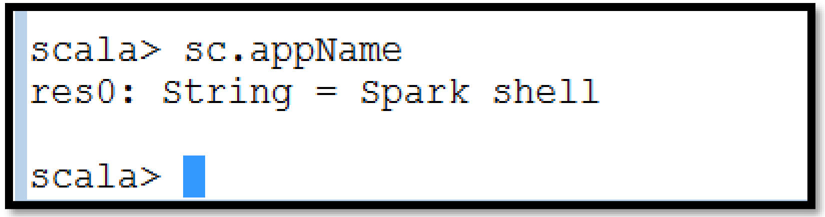
```
Spark context Web UI available at http://192.168.163.151:4040
Spark context available as 'sc' (master = local[*], app id = local-1509877354288).
Spark session available as 'spark'.
Welcome to
      ____              __
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ `/ __/  '_/
   /___/ .__/\_,_/_/ /_/\_\   version 2.1.0
      /_/

Using Scala version 2.11.8 (Java HotSpot(TM) 64-Bit Server VM, Java 1.8.0_77)
Type in expressions to have them evaluated.
Type :help for more information.

scala>
```

SPARK CONTEXT

- At high level, every Spark application requires Spark Context.
- Spark Context is entry point to the Spark API.
- Driver program communicates with Spark through Spark Context.



```
Spark context available as 'sc' (master = local[*], app id = local-1509877354288).
Spark session available as 'spark'.
Welcome to
      ____              __
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ `/ __/  '_/
   /___/ .__/\_,_/_/ /_/\_\   version 2.1.0
      /_/
```



```
scala> sc.appName
res0: String = Spark shell

scala>
```

- Spark Eco System

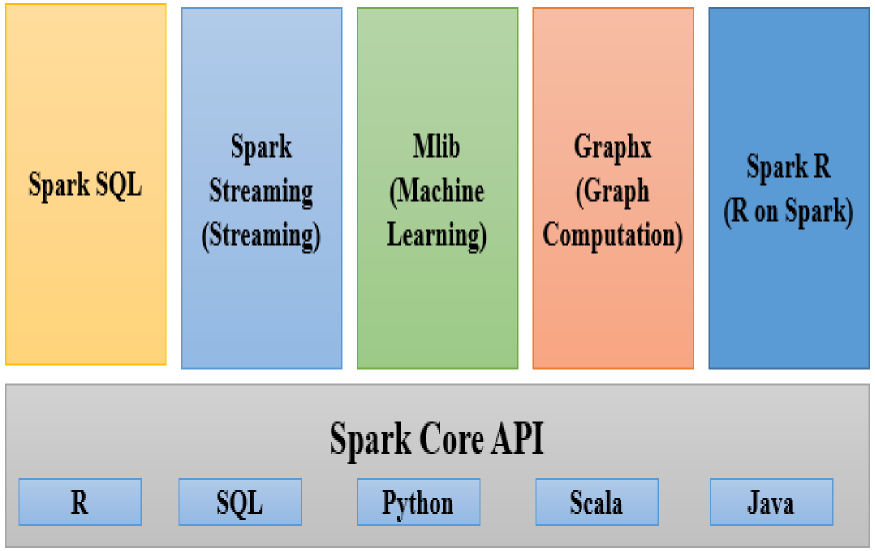
Spark Eco System

Spark Core(RDD):

- Fundamental data structure of Spark.
- RDD (Resilient Distributed Dataset), fault-tolerant collection of elements that can be operated on in parallel.

Spark SQL:

- Running SQL like queries on Spark data.



Spark Eco System

Spark Streaming:

- Scalable, fault-tolerant, high stream processing of live data streams.

MLib:

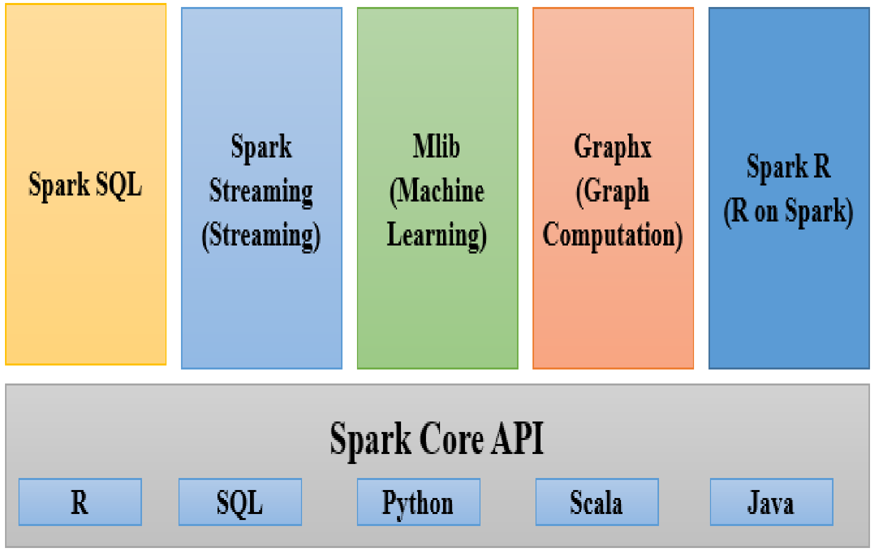
- Scalable practical machine learning.

Graphx:

- Graphs and graph-parallel computation.

SparkR:

- R package that provides a light-weight frontend to use Apache Spark from R.



- Introduction to RDD

accenture

LEARNING OBJECTIVES

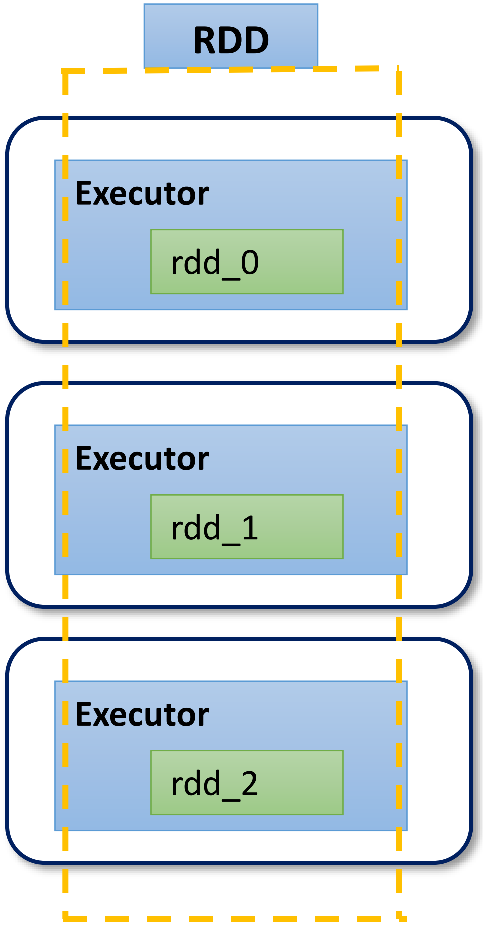
At the end of this unit, you should be able to:

- What is RDD?
- RDD Characteristics
- Partitions
- Read - Only
- RDD operations
- Creating an RDD

WHAT IS RDD?

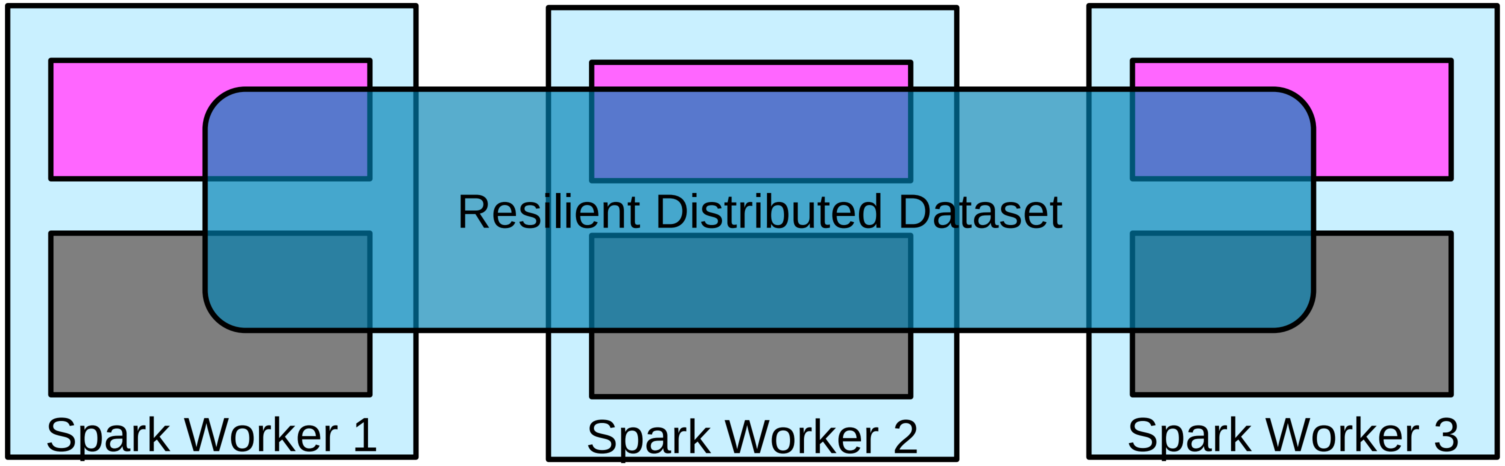
RDD (Resilient distributed dataset)

- Fundamental data structure of Apache Spark.
- Resilient - In built fault tolerance. If something goes wrong reconstruct from source (Lineage).
- Distributed - data is distributed in memory across the worker nodes.
- Dataset - represents records of the data.



WHAT IS RDD?

- RDD is an immutable collection of objects which computes on the different node of the cluster.
- Dataset in RDDs are logically partitioned across many nodes for parallel computation.



RDD CHARACTERISTICS

Partitions:

- RDD represent data in memory.
- To handle huge volume of data
  - data is divided in to partitions that are distributed to multiple machines called nodes.
  - process data in parallel.

RDD CHARACTERISTICS

Ready - Only:

- RDDs are immutable.
  - can not be modified in place.
  - to modify, there are only two types of operations.
    - Transformations
    - Actions

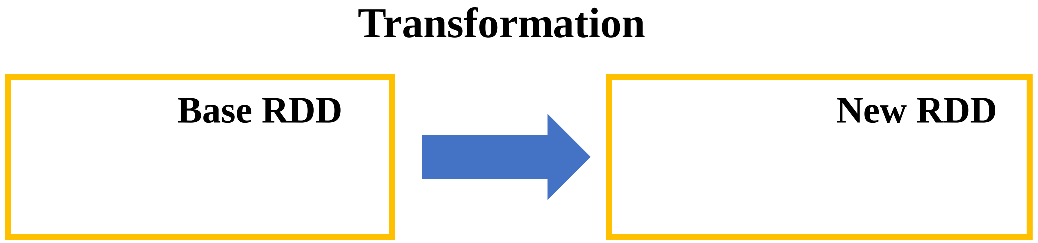
RDD OPERATIONS

Transformation:

- Converts an RDD into another RDD.
- Transform records in dataset.
- Once a dataset is loaded into memory, you can perform chain of transformation before you see some results.

Example:

- Filtering out only certain records.
- Extracting specific field.



RDD OPERATIONS

Actions:

- The actual data processing doesn’t happen immediately.
- It happens only user requests a result.

Example:

- First 10 rows
- A Sum
- A Count

Action
RDD
Value

CREATING AN RDD

- Two ways to create an RDD.
  - by loading an external dataset, or
  - by distributing a collection of objects (e.g., a list or set).

CREATING AN RDD - USING PARALLELIZED COLLECTIONS

Parallelized Collections:

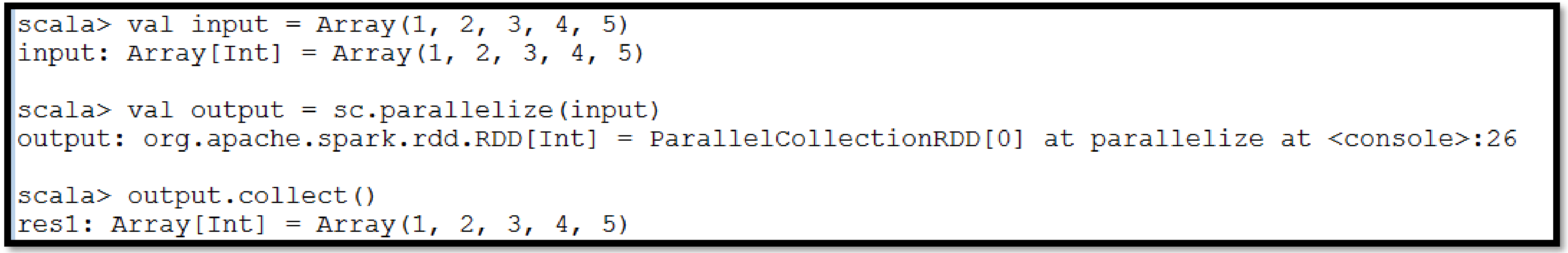
Objective: Creating an RDD using parallelized collection method of Spark Context.

Action:

```
val input = Array(1, 2, 3, 4, 5)
val output = sc.parallelize(input)
output.collect()    // To display output on the console
```

Note: collect() is an Action

Output:



```
scala> val input = Array(1, 2, 3, 4, 5)
input: Array[Int] = Array(1, 2, 3, 4, 5)

scala> val output = sc.parallelize(input)
output: org.apache.spark.rdd.RDD[Int] = ParallelCollectionRDD[0] at parallelize at <console>:26

scala> output.collect()
res1: Array[Int] = Array(1, 2, 3, 4, 5)
```

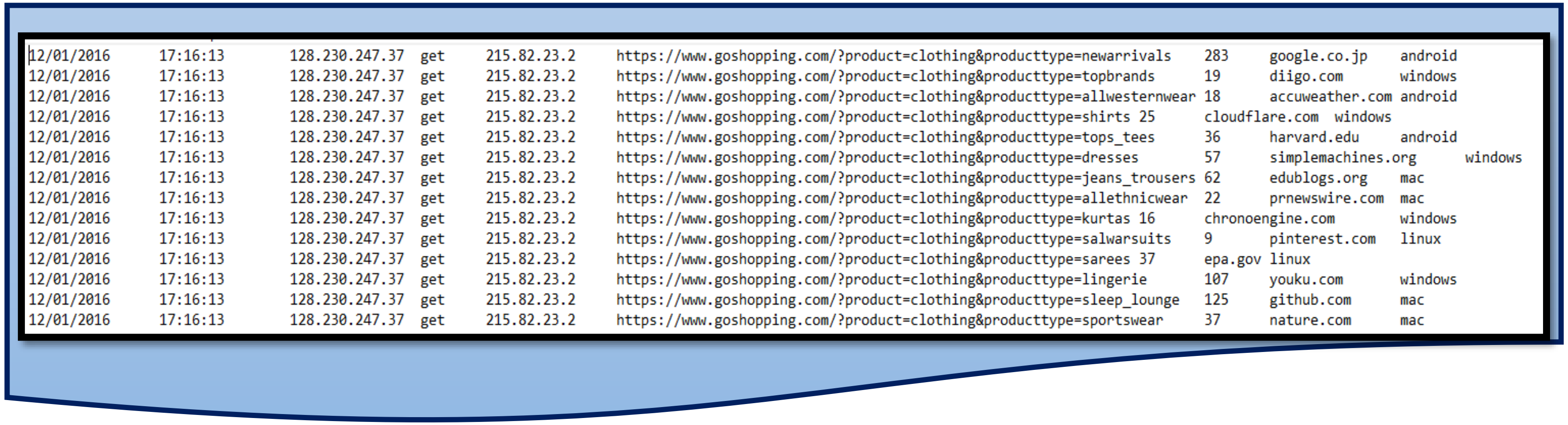
CREATING AN RDD – USING TEXT FILE

File based RDD:

TextFile RDD can be created using textFile method of Spark Context.

Input Data:

File Name: "goShopping_WebClicks.dat " (contains user activity)

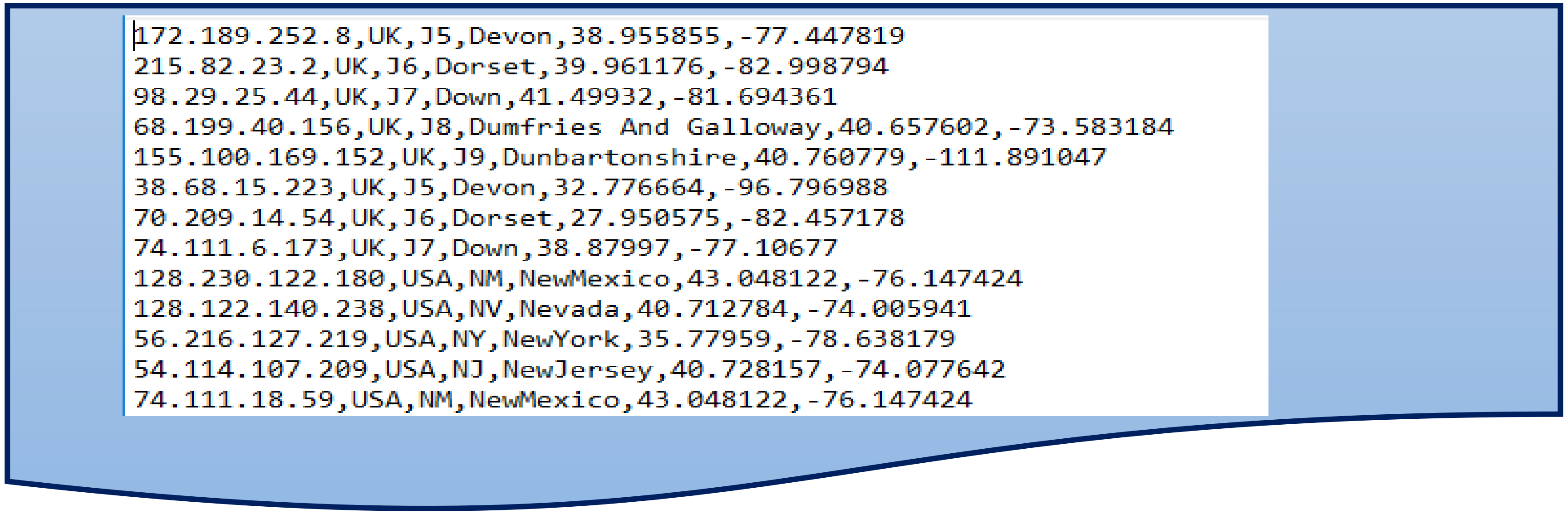


```
12/01/2016      17:16:13        128.230.247.37  get     215.82.23.2     https://www.goshopping.com/?product=clothing&producttype=newarrivals    283     google.co.jp    android
12/01/2016      17:16:13        128.230.247.37  get     215.82.23.2     https://www.goshopping.com/?product=clothing&producttype=topbrands      19      diigo.com       windows
12/01/2016      17:16:13        128.230.247.37  get     215.82.23.2     https://www.goshopping.com/?product=clothing&producttype=allwesternwear 18      accuweather.com android
12/01/2016      17:16:13        128.230.247.37  get     215.82.23.2     https://www.goshopping.com/?product=clothing&producttype=shirts 25      cloudflare.com  windows
12/01/2016      17:16:13        128.230.247.37  get     215.82.23.2     https://www.goshopping.com/?product=clothing&producttype=tops_tees      36      harvard.edu     android
12/01/2016      17:16:13        128.230.247.37  get     215.82.23.2     https://www.goshopping.com/?product=clothing&producttype=dresses        57      simplemachines.org      windows
12/01/2016      17:16:13        128.230.247.37  get     215.82.23.2     https://www.goshopping.com/?product=clothing&producttype=jeans_trousers 62      edublogs.org    mac
12/01/2016      17:16:13        128.230.247.37  get     215.82.23.2     https://www.goshopping.com/?product=clothing&producttype=allethnicwear  22      prnewswire.com  mac
12/01/2016      17:16:13        128.230.247.37  get     215.82.23.2     https://www.goshopping.com/?product=clothing&producttype=kurtas 16      chronoengine.com        windows
12/01/2016      17:16:13        128.230.247.37  get     215.82.23.2     https://www.goshopping.com/?product=clothing&producttype=salwarsuits    9       pinterest.com   linux
12/01/2016      17:16:13        128.230.247.37  get     215.82.23.2     https://www.goshopping.com/?product=clothing&producttype=sarees 37      epa.gov linux
12/01/2016      17:16:13        128.230.247.37  get     215.82.23.2     https://www.goshopping.com/?product=clothing&producttype=lingerie       107     youku.com       windows
12/01/2016      17:16:13        128.230.247.37  get     215.82.23.2     https://www.goshopping.com/?product=clothing&producttype=sleep_lounge   125     github.com      mac
12/01/2016      17:16:13        128.230.247.37  get     215.82.23.2     https://www.goshopping.com/?product=clothing&producttype=sportswear     37      nature.com      mac
```

CREATING AN RDD – USING TEXT FILE

Input Data:

File Name: "goShopping_IpLookup.txt" (contains user IP details)



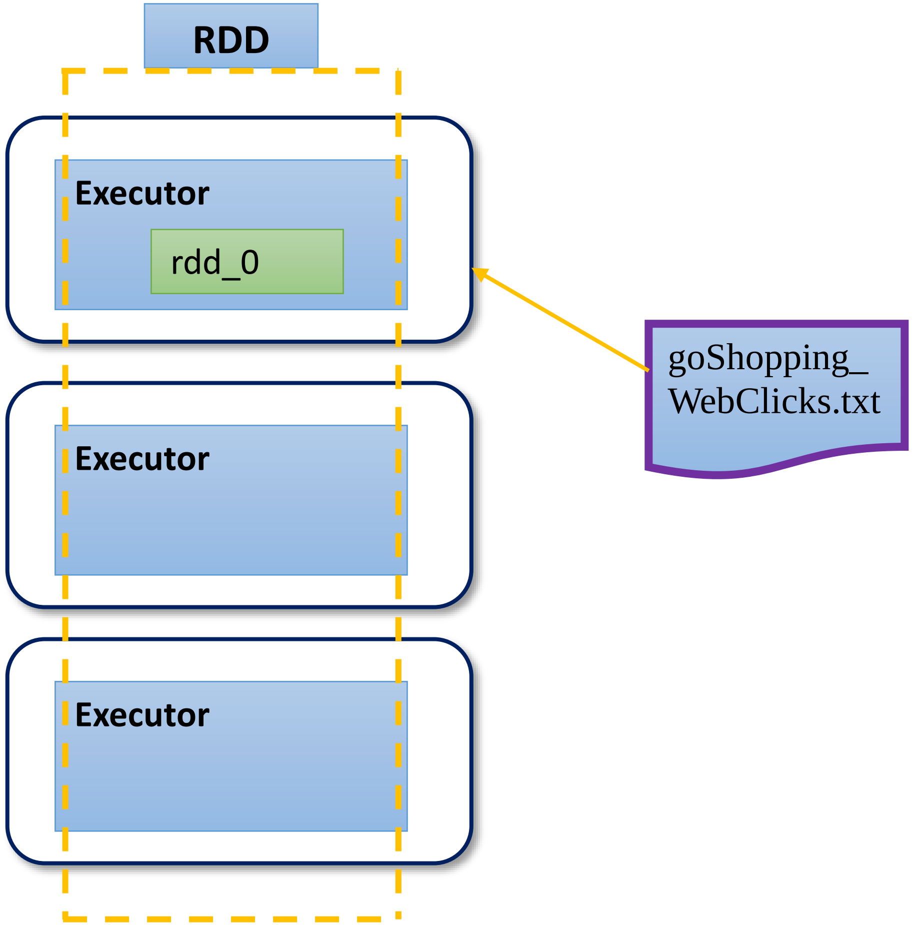
```
172.189.252.8,UK,J5,Devon,38.955855,-77.447819
215.82.23.2,UK,J6,Dorset,39.961176,-82.998794
98.29.25.44,UK,J7,Down,41.49932,-81.694361
68.199.40.156,UK,J8,Dumfries And Galloway,40.657602,-73.583184
155.100.169.152,UK,J9,Dunbartonshire,40.760779,-111.891047
38.68.15.223,UK,J5,Devon,32.776664,-96.796988
70.209.14.54,UK,J6,Dorset,27.950575,-82.457178
74.111.6.173,UK,J7,Down,38.87997,-77.10677
128.230.122.180,USA,NM,NewMexico,43.048122,-76.147424
128.122.140.238,USA,NV,Nevada,40.712784,-74.005941
56.216.127.219,USA,NY,NewYork,35.77959,-78.638179
54.114.107.209,USA,NJ,NewJersey,40.728157,-74.077642
74.111.18.59,USA,NM,NewMexico,43.048122,-76.147424
```

CREATING AN RDD – FILE PARTITIONING

Partitions - single file

- Partitions is done based on size.
- You can specify the number of partitions as shown below.

  textFile(filename, minPartitions)
- The default number of partitions: 2
- more number of partitions = more parallelization



sc.textFile("goShopping_WebClicks.txt",1)

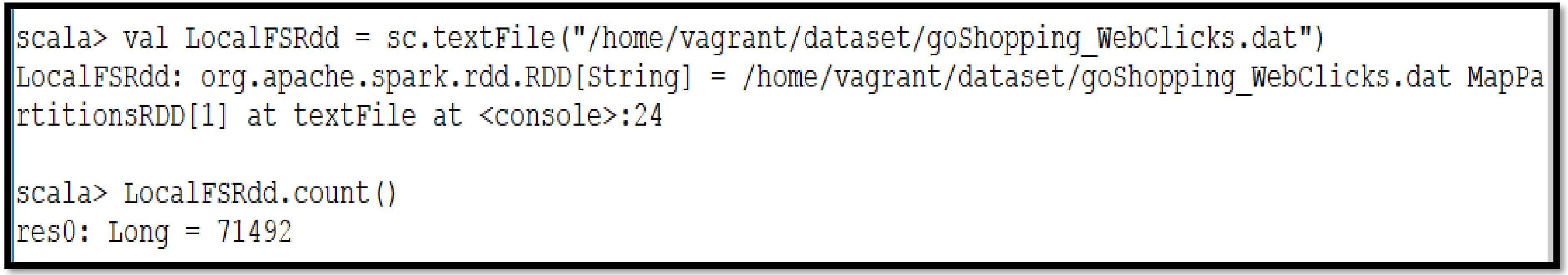
CREATING AN RDD – USING TEXT FILE

Objective: Create an RDD for user activity (Use Local File System).

Action:

```
val LocalFSRdd = sc.textFile("/home/vagrant/dataset/goShopping_WebClicks.dat")
// URL of the local FS
LocalFSRdd.count() // count() is an action, counts the number of records.
```

Output:



```
scala> val LocalFSRdd = sc.textFile("/home/vagrant/dataset/goShopping_WebClicks.dat")
LocalFSRdd: org.apache.spark.rdd.RDD[String] = /home/vagrant/dataset/goShopping_WebClicks.dat MapPa
rtitionsRDD[1] at textFile at <console>:24

scala> LocalFSRdd.count()
res0: Long = 71492
```

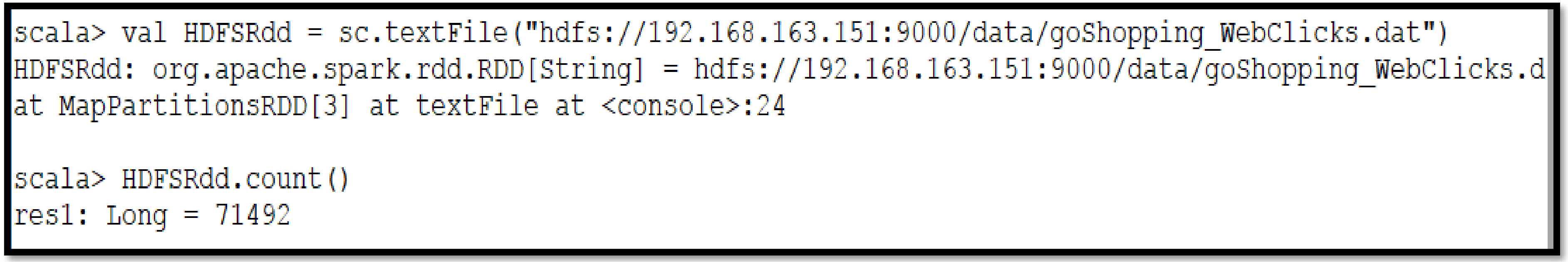
CREATING AN RDD – USING TEXT FILE

Objective: Create an RDD for user activity (Use HDFS File System).

Action:

```
val HDFSRdd =
sc.textFile("hdfs://192.168.163.151:9000/data/goShopping_WebClicks.dat") // URL of the HDFS path.
HDFSRdd.count()
```

Output:



```
scala> val HDFSRdd = sc.textFile("hdfs://192.168.163.151:9000/data/goShopping_WebClicks.dat")
HDFSRdd: org.apache.spark.rdd.RDD[String] = hdfs://192.168.163.151:9000/data/goShopping_WebClicks.d
at MapPartitionsRDD[3] at textFile at <console>:24

scala> HDFSRdd.count()
res1: Long = 71492
```

CREATING AN RDD – USING TEXT FILE

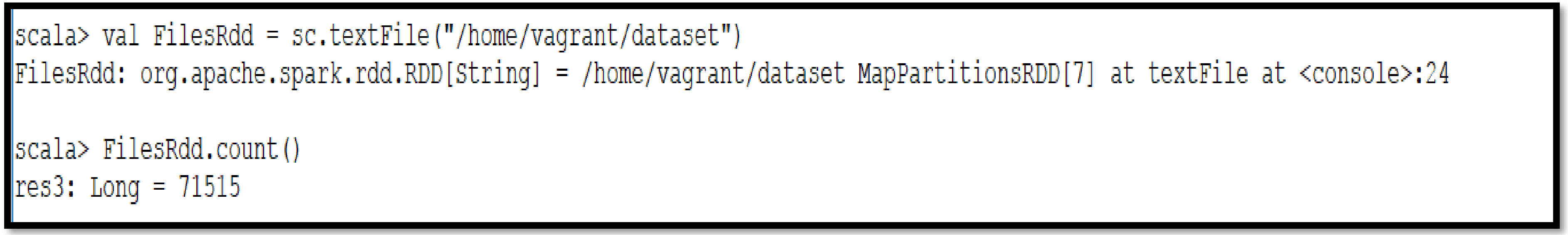
Objective: Create an RDD for user details and activity.

Action:

```
val FilesRdd = sc.textFile("/home/vagrant/dataset")
FilesRdd.count() //  count () is  an action.
```

Note: Directory dataset contains user activity and user details.

Output:



```
scala> val FilesRdd = sc.textFile("/home/vagrant/dataset")
FilesRdd: org.apache.spark.rdd.RDD[String] = /home/vagrant/dataset MapPartitionsRDD[7] at textFile at <console>:24

scala> FilesRdd.count()
res3: Long = 71515
```

- Transformations and Actions

LEARNING OBJECTIVES

At the end of this unit, you should be able to:

- Transformations
- Actions
- What is Pair RDDs
- Creating Pair RDDs
- Persisting RDDs
- Storage Level
- Accumulators
- Broadcast Variables

Actions

- Actions return values to the driver program.
- Common actions are:
  - reduce(func)
  - collect()
  - count()
  - first()
  - take(n)
  - foreach(println)
  - saveAsTextFile(path)

Action
RDD
Value

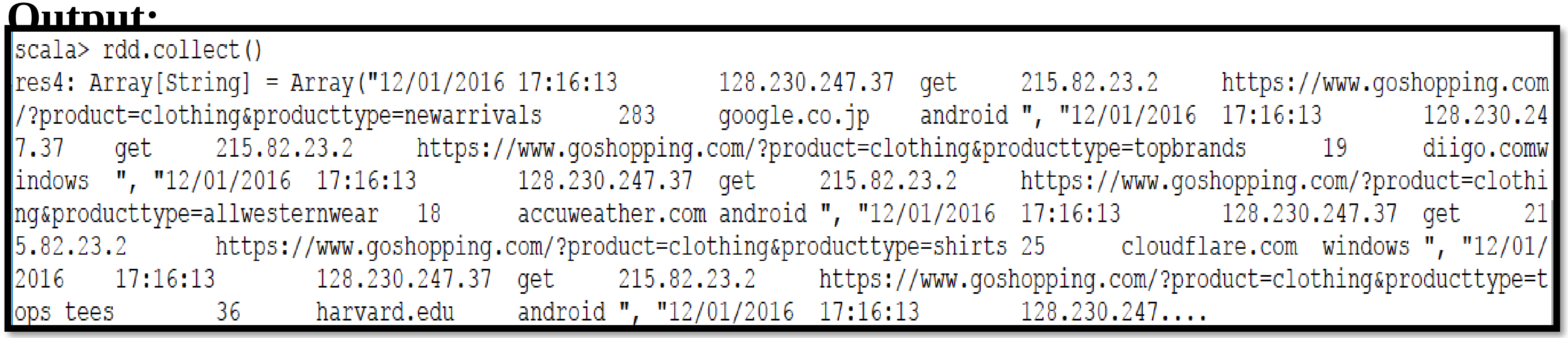
Actions – Collect()

Objective: Display the records of goShopping_WebClicks.dat file.

Action:

```
val rdd = sc.textFile("/home/vagrant/dataset/goShopping_WebClicks.dat")
rdd.collect()
```

Output:



```
scala> rdd.collect()
res4: Array[String] = Array("12/01/2016 17:16:13        128.230.247.37  get     215.82.23.2     https://www.goshopping.com
/?product=clothing&producttype=newarrivals      283     google.co.jp    android ", "12/01/2016  17:16:13        128.230.24
7.37    get     215.82.23.2     https://www.goshopping.com/?product=clothing&producttype=topbrands      19      diigo.comw
indows  ", "12/01/2016  17:16:13        128.230.247.37  get     215.82.23.2     https://www.goshopping.com/?product=clothi
ng&producttype=allwesternwear   18      accuweather.com android ", "12/01/2016  17:16:13        128.230.247.37  get     21
5.82.23.2       https://www.goshopping.com/?product=clothing&producttype=shirts 25      cloudflare.com  windows ", "12/01/
2016    17:16:13        128.230.247.37  get     215.82.23.2     https://www.goshopping.com/?product=clothing&producttype=t
ops tees        36      harvard.edu     android ", "12/01/2016  17:16:13        128.230.247....
```

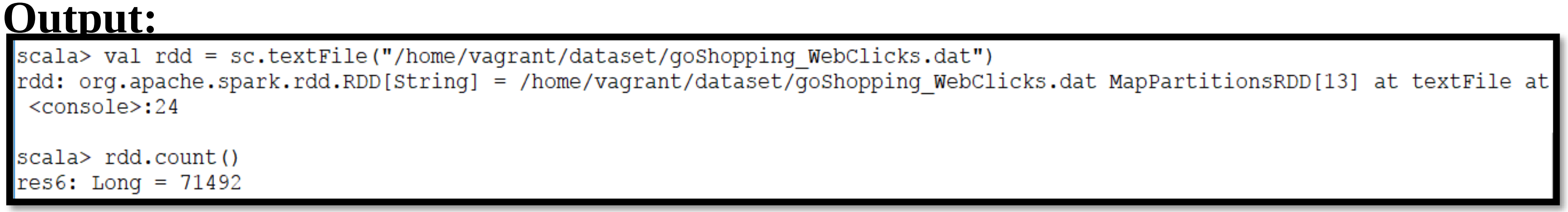
Actions – Count()

Objective: Count the number of records of goShopping_WebClicks.dat file.

Action:

```
val rdd = sc.textFile("/home/vagrant/dataset/goShopping_WebClicks.dat")
rdd.count()
```

Output:



```
scala> val rdd = sc.textFile("/home/vagrant/dataset/goShopping_WebClicks.dat")
rdd: org.apache.spark.rdd.RDD[String] = /home/vagrant/dataset/goShopping_WebClicks.dat MapPartitionsRDD[13] at textFile at
 <console>:24

scala> rdd.count()
res6: Long = 71492
```

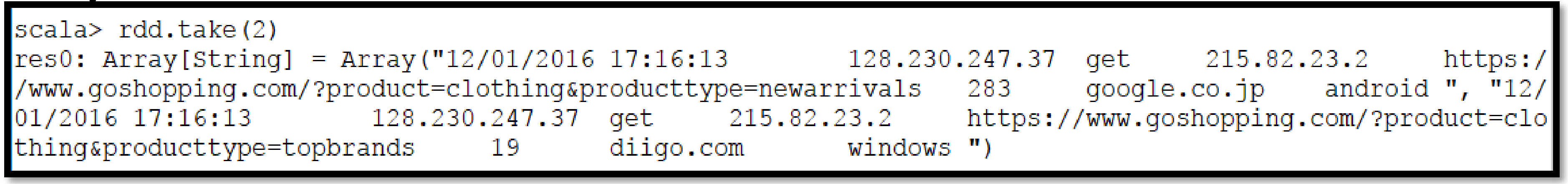
Actions – take(n)

Objective: Display first 2 records of goShopping_WebClicks.dat file.

Action:

```
val rdd = sc.textFile("/home/vagrant/dataset/goShopping_WebClicks.dat")
rdd.take(2)
```

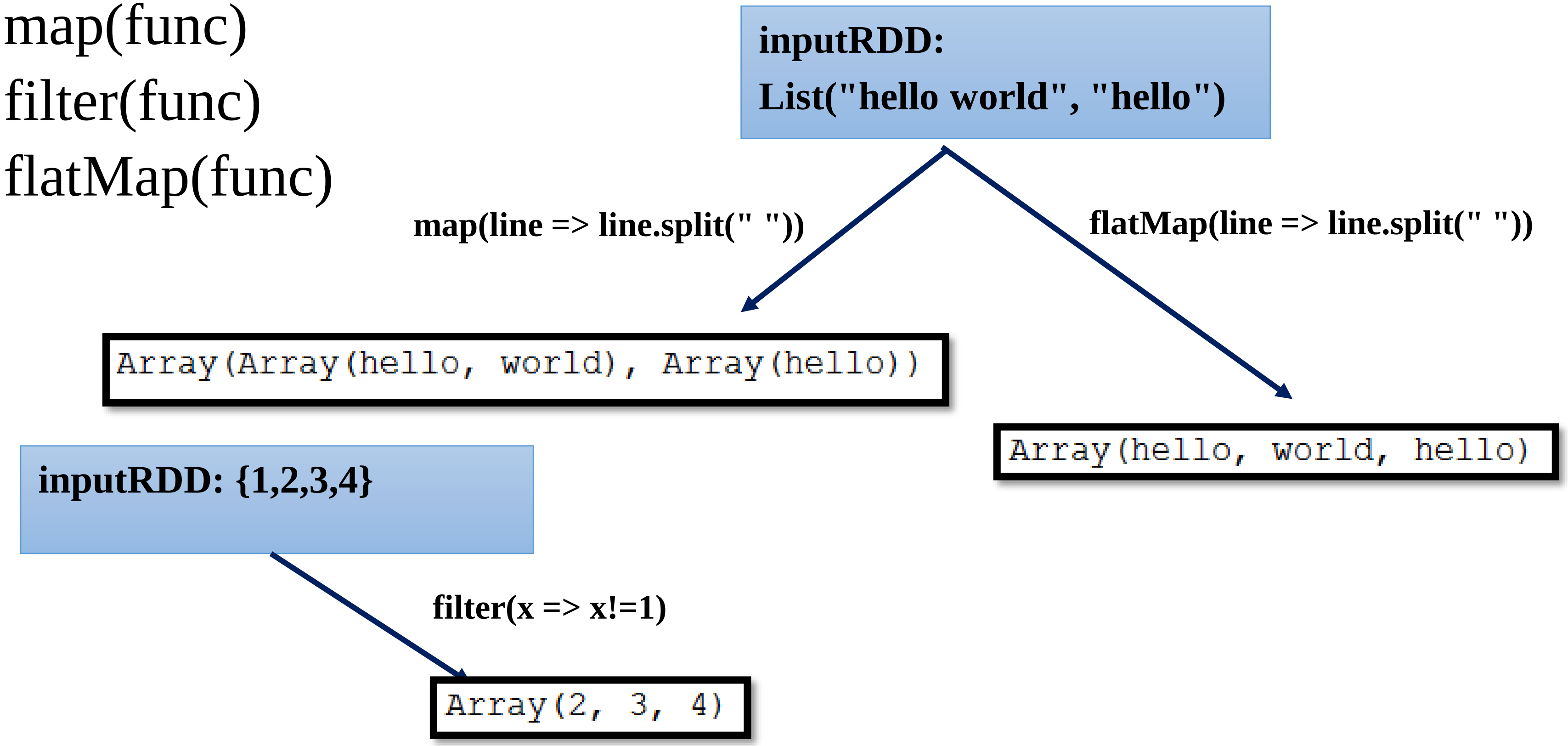
Output:



```
scala> rdd.take(2)
res0: Array[String] = Array("12/01/2016 17:16:13        128.230.247.37  get     215.82.23.2     https:/
/www.goshopping.com/?product=clothing&producttype=newarrivals   283     google.co.jp    android ", "12/
01/2016 17:16:13        128.230.247.37  get     215.82.23.2     https://www.goshopping.com/?product=clo
thing&producttype=topbrands     19      diigo.com       windows ")
```

Transformations

- Creating a new RDD from an existing RDD.
- Common transformations are:
  - map(func)
  - filter(func)
  - flatMap(func)



inputRDD:
List("hello world", "hello")

map(line => line.split(" "))

```
Array(Array(hello, world), Array(hello))
```

flatMap(line => line.split(" "))

```
Array(hello, world, hello)
```

inputRDD: {1,2,3,4}

filter(x => x!=1)

```
Array(2, 3, 4)
```

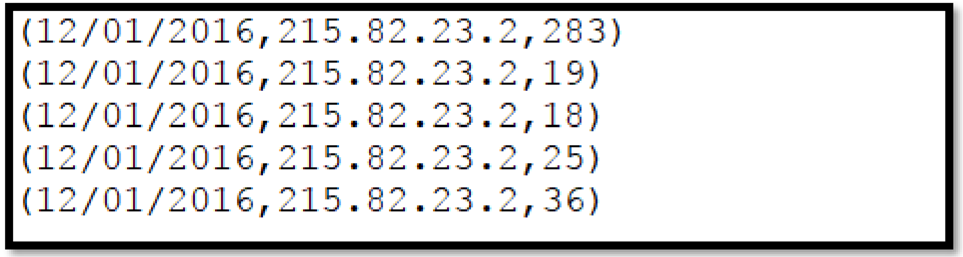
Transformations – map(func)

Objective: Retrieve date, customer ip, time spent on shopping site .

Action:

```
val rdd = sc.textFile("/home/vagrant/dataset/goShopping_WebClicks.dat")
val split = rdd.map(line => line.split("\t"))
val result = split.map(field => (field(0),field(4),field(6)))
result.take(5).foreach(println)
```

Output:



```
(12/01/2016,215.82.23.2,283)
(12/01/2016,215.82.23.2,19)
(12/01/2016,215.82.23.2,18)
(12/01/2016,215.82.23.2,25)
(12/01/2016,215.82.23.2,36)
```

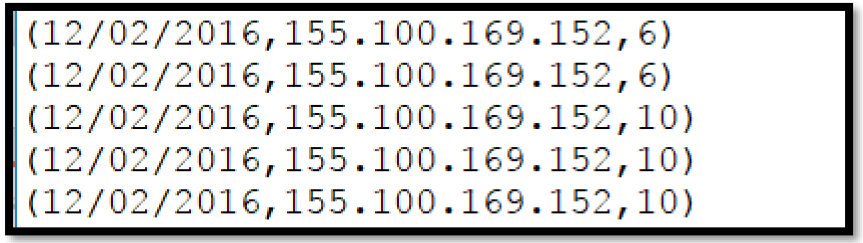
Transformations – filter(func)

Objective: Retrieve date, customer ip, time spent on shopping site only on "12/01/2016" .

Action:

```
val rdd = sc.textFile("/home/vagrant/dataset/goShopping_WebClicks.dat")
val filtered = rdd.filter(line => line.split("\t")(0).equals(("12/02/2016")))
val split = filtered.map(line => line.split("\t"))
val result = split.map(field => (field(0),field(4),field(6)))
result.take(5).foreach(println)
```

Output:

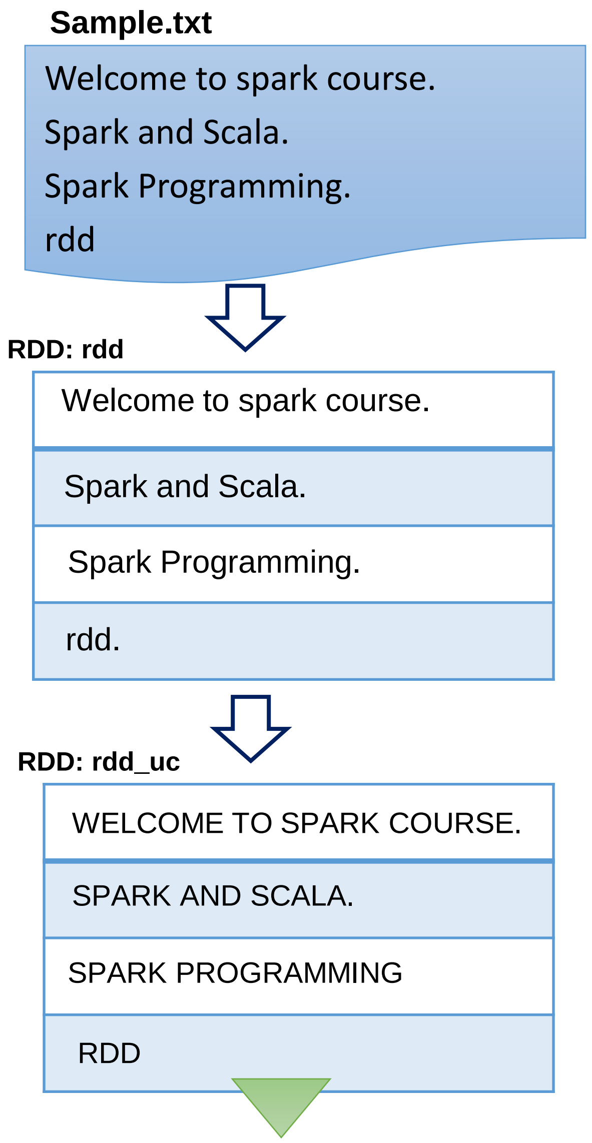


```
(12/02/2016,155.100.169.152,6)
(12/02/2016,155.100.169.152,6)
(12/02/2016,155.100.169.152,10)
(12/02/2016,155.100.169.152,10)
(12/02/2016,155.100.169.152,10)
```

Lazy Evaluation

- All transformations are lazy, in that they do not compute their results until an action is called.

```
> val rdd = sc.textFile("sample.txt")

  val rdd_uc =rdd.map(line =>
  line.toUpperCase())

  rdd_uc.collect()
```



Sample.txt

Welcome to spark course.
Spark and Scala.
Spark Programming.
rdd

RDD: rdd

Welcome to spark course.
Spark and Scala.
Spark Programming.
rdd.

RDD: rdd_uc

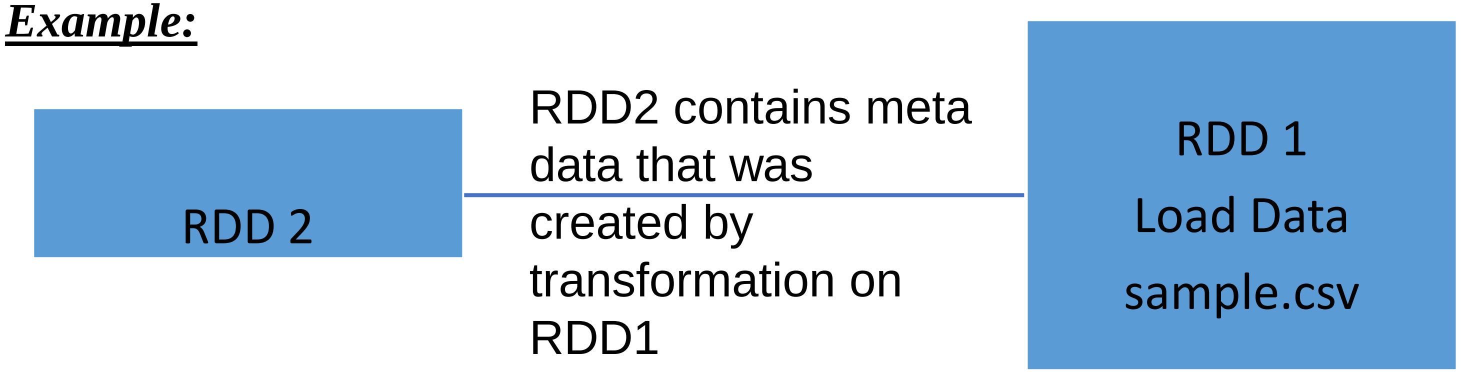
WELCOME TO SPARK COURSE.
SPARK AND SCALA.
SPARK PROGRAMMING
RDD

Lazy Evaluation

RDD Lineage and toDebugString():

- Spark maintains each RDDs lineage i.e. previous RDD which it depends.
- When an RDD is created, it just holds metadata.
    - A transformation that created the RDD.
    - Its parent RDD from which it was created.

<u>Example:</u>



RDD 2

RDD2 contains meta data that was created by transformation on RDD1

RDD 1

Load Data

sample.csv

What is Pair RDDs?

- Special form of RDD.
- Each element in an Pair RDD is a key-value pair.
- Key-value can be of any type.

Pair RDD

(key, value)
(key, value)
(key, value)

- Common Pair RDDs are:
  - groupByKey([numTasks])
  - reduceByKey(func, [numTasks]):

Pair RDDs are useful:

- sorting, grouping, etc..

Creating Pair RDDs

- To create Pair RDDs:
  - get data in the form of key, value pair.

- Commonly used functions to create Pair RDDs:
  - groupByKey
  - reduceByKey

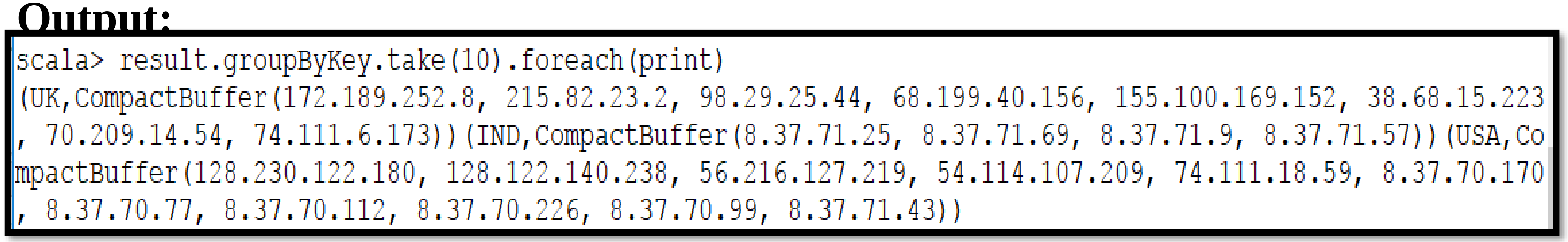
Pair RDDs – groupByKey([numTasks])

Objective: Display the list of IP addresses for each country .

Action:

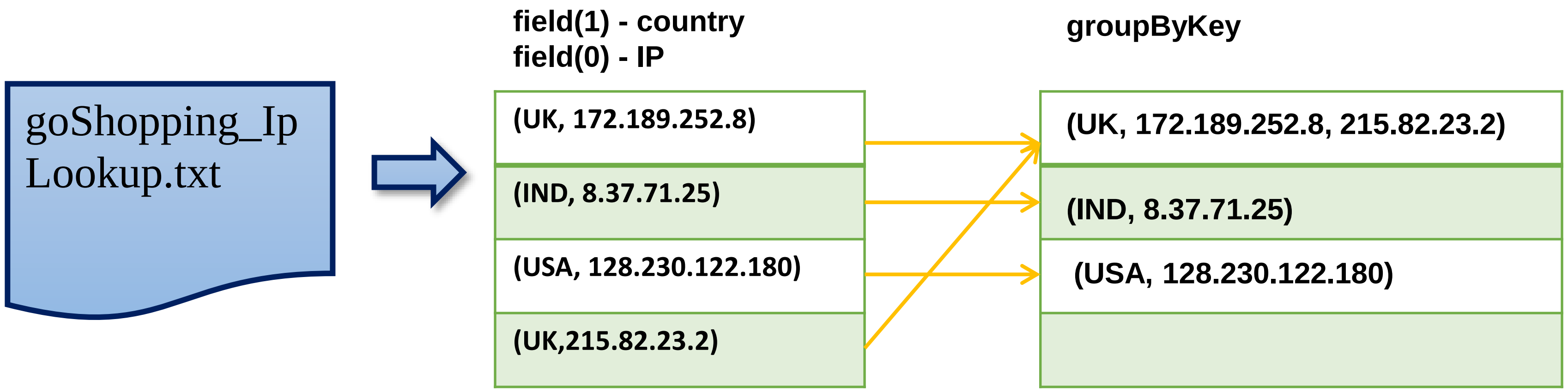
```
val rdd = sc.textFile("/home/vagrant/dataset/goShopping_IpLookup.txt")
val split = rdd.map(line => line.split(","))
val result = split.map(field => (field(1),field(0)))
result.groupByKey.take(10).foreach(print)
```

Output:



```
scala> result.groupByKey.take(10).foreach(print)
(UK,CompactBuffer(172.189.252.8, 215.82.23.2, 98.29.25.44, 68.199.40.156, 155.100.169.152, 38.68.15.223
, 70.209.14.54, 74.111.6.173))(IND,CompactBuffer(8.37.71.25, 8.37.71.69, 8.37.71.9, 8.37.71.57))(USA,Co
mpactBuffer(128.230.122.180, 128.122.140.238, 56.216.127.219, 54.114.107.209, 74.111.18.59, 8.37.70.170
, 8.37.70.77, 8.37.70.112, 8.37.70.226, 8.37.70.99, 8.37.71.43))
```

Pair RDDs – groupByKey([numTasks])



goShopping_Ip Lookup.txt

field(1) - country
field(0) - IP

(UK, 172.189.252.8)

(IND, 8.37.71.25)

(USA, 128.230.122.180)

(UK,215.82.23.2)

groupByKey

(UK, 172.189.252.8, 215.82.23.2)

(IND, 8.37.71.25)

(USA, 128.230.122.180)

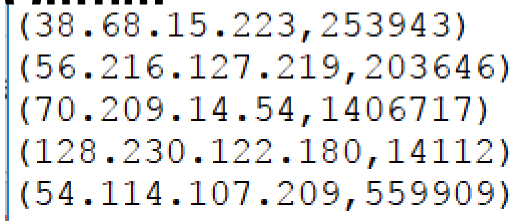
Pair RDDs – reduceByKey(func, [numTasks])

Objective: Find total time spent on shopping site by each customer.

Action:

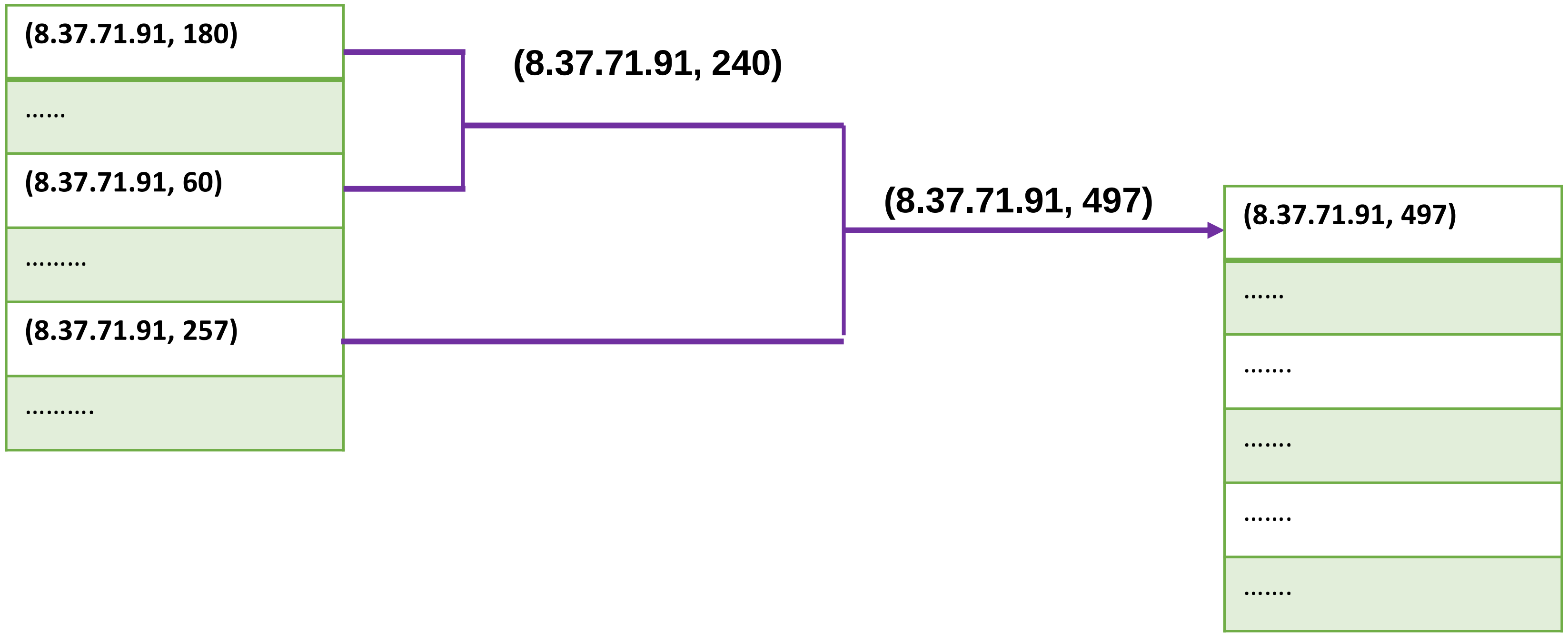
```
val rdd = sc.textFile("/home/vagrant/dataset/goShopping_WebClicks.dat")
val split = rdd.map(line => line.split("\t"))
val result = split.map(field => (field(4),field(6).toInt)). reduceByKey((v1,v2) =>
v1 + v2)
result.take(5).foreach(println)
```

Output:



```
(38.68.15.223,253943)
(56.216.127.219,203646)
(70.209.14.54,1406717)
(128.230.122.180,14112)
(54.114.107.209,559909)
```

Pair RDDs – reduceByKey(func, [numTasks])



(8.37.71.91, 180)

……

(8.37.71.91, 60)

………

(8.37.71.91, 257)

……….

(8.37.71.91, 240)

(8.37.71.91, 497)

(8.37.71.91, 497)

……

…….

…….

…….

…….

Persisting RDDs

- One of the important features of Spark is is persisting (or caching) a dataset in memory across operations.

- Persisting an RDD:
  - each node stores any partitions of it that it computes in memory and reuses them in other actions on that dataset.
  - allows future actions to be much faster.
  - use persist() or cache() methods.

- The difference between cache() and persist():
  - cache() ==   default storage level (MEMORY_ONLY).
  - persist() ==   specify various storage levels.

Storage Levels

Storage Level	Meaning
MEMORY_ONLY	▪ Default level. ▪ Store RDD as deserialized Java objects in the JVM
MEMORY_AND_DISK	▪ Store RDD as deserialized Java objects in the JVM. ▪ If the RDD does not fit in memory, store the partitions that don't fit on disk, and read them from there when they're needed.
DISK_ONLY	▪ Store the RDD partitions only on disk.

Persisting RDDs - Example

```
val rdd = sc.textFile("/home/vagrant/dataset/goShopping_WebClicks.dat")
import org.apache.spark.storage.StorageLevel
rdd.persist(StorageLevel.MEMORY_ONLY)
```

Accumulators

Accumulators:

- Variables that are used for aggregating information across the executors.

- Simple syntax for aggregating values from worker nodes back to the driver program.

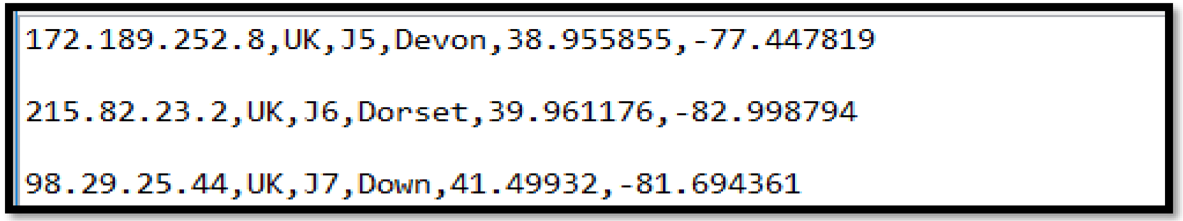
Use Case:

- Find out the number of blank logs (blank lines) .
- Number of times the network failed.
- Number of times zero sales were recorded.

Accumulators

Objective: Find the number of blank lines present in "goShopping_IpLookup.txt" .

Input Data: "goShopping_IpLookup.txt"

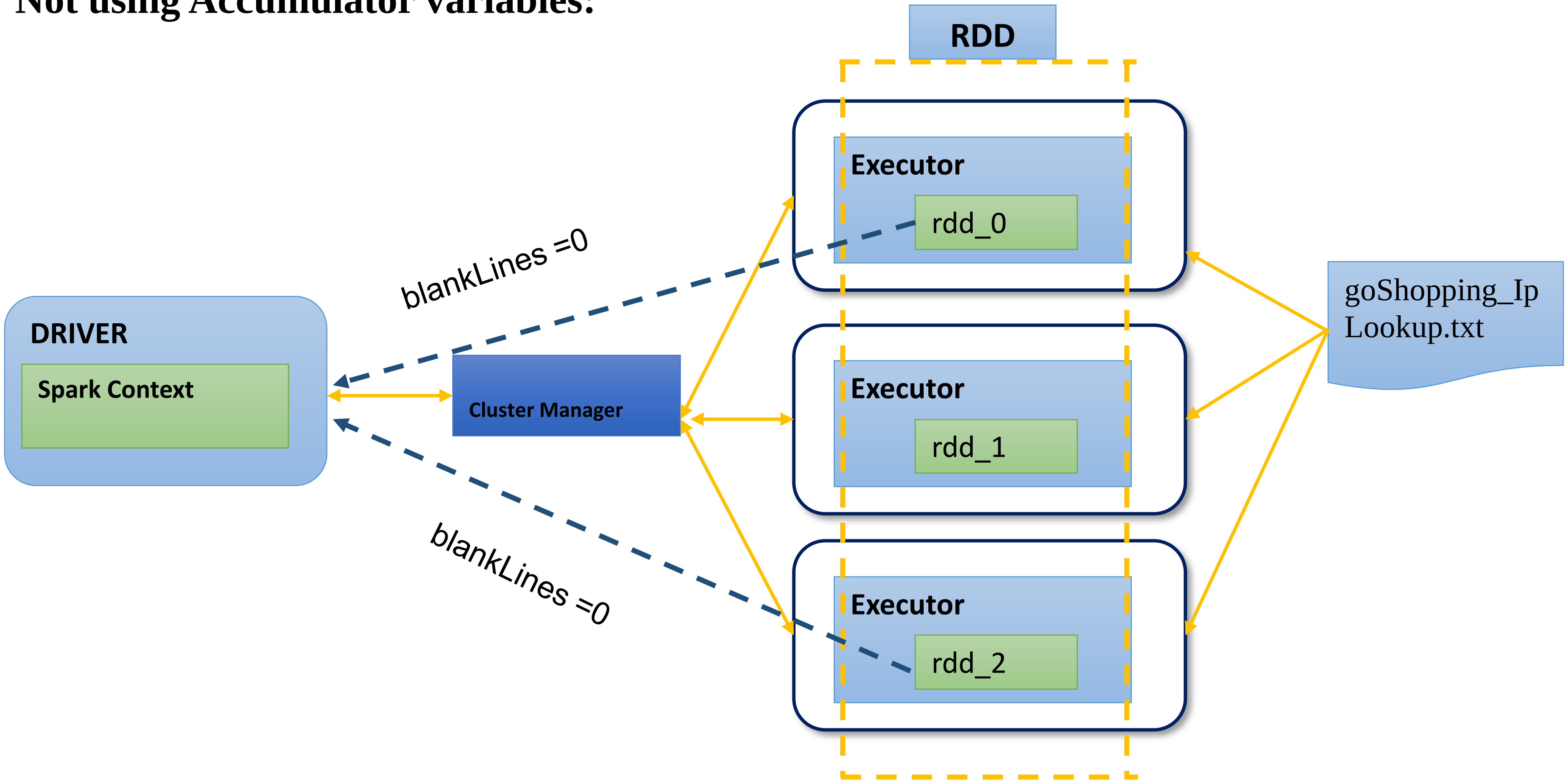


```
172.189.252.8,UK,J5,Devon,38.955855,-77.447819

215.82.23.2,UK,J6,Dorset,39.961176,-82.998794

98.29.25.44,UK,J7,Down,41.49932,-81.694361
```

Number of blank lines: 2

Accumulators

Not using Accumulator variables:



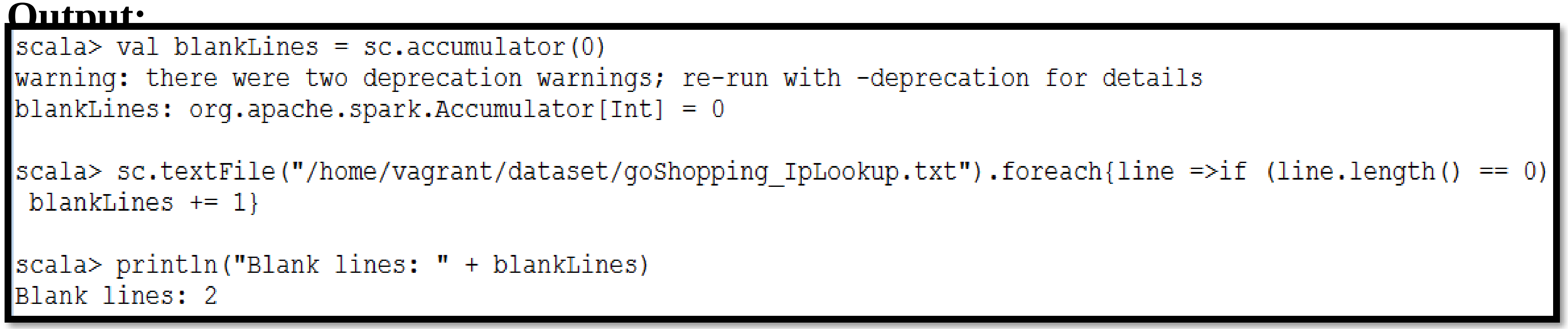
Accumulators

Using Accumulator variables:

Action:

```
val blankLines = sc.accumulator(0)
sc.textFile("/home/vagrant/dataset/goShopping_IpLookup.txt").foreach{line =>if (line.length() == 0) blankLines += 1}
println("Blank lines: " + blankLines)
```

Output:



```
scala> val blankLines = sc.accumulator(0)
warning: there were two deprecation warnings; re-run with -deprecation for details
blankLines: org.apache.spark.Accumulator[Int] = 0

scala> sc.textFile("/home/vagrant/dataset/goShopping_IpLookup.txt").foreach{line =>if (line.length() == 0)
 blankLines += 1}

scala> println("Blank lines: " + blankLines)
Blank lines: 2
```

Broadcast Variables

Broadcast Variables:

- To keep a read-only variable cached on each machine rather than shipping a copy of it with tasks.
- Use SparkContext.broadcast(v) to create broadcast variable.
- Variable v acts as a broadcast variable.
- value() can be used to access the broadcast variable.
- Useful when tasks across multiple stages need the same data.

Broadcast Variables

Objective:

Categorize the user IP as type "**AccidentVisit**" if time spent is less than 60 secs , else as type " **PurposeVisit** ".

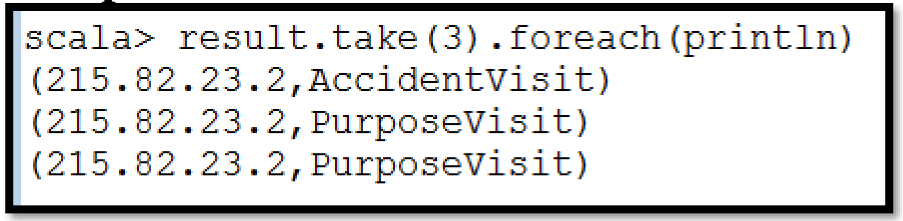
" goShopping_WebClicks.dat "

Broadcast Variables

Action:

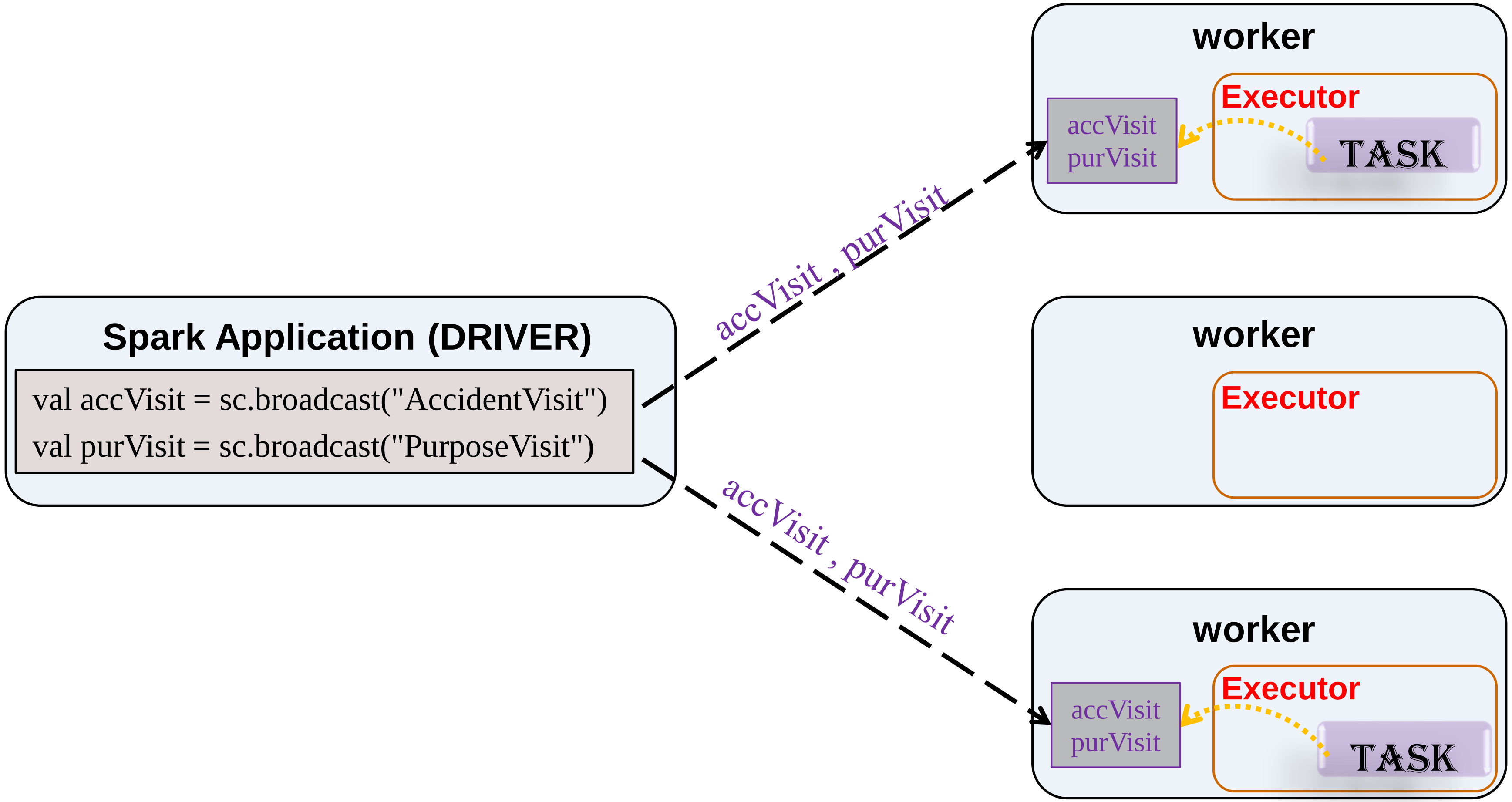
```
val accVisit = sc.broadcast("AccidentVisit")
val purVisit = sc.broadcast("PurposeVisit")
val rdd = sc.textFile("/home/vagrant/data/goShopping_WebClicks.dat",2)
val split = rdd.map(line => line.split("\t"))
val result = split.map(field =>
(field(4),if(field(6).toInt>=60){accVisit.value}else{purVisit.value}))
result.take(3).foreach(println)
```

Output:



```
scala> result.take(3).foreach(println)
(215.82.23.2,AccidentVisit)
(215.82.23.2,PurposeVisit)
(215.82.23.2,PurposeVisit)
```

Broadcast Variables



```
val accVisit = sc.broadcast("AccidentVisit")
val purVisit = sc.broadcast("PurposeVisit")
```

Questions

THANK YOU