Principles of Spark :

Name: อวยชัย ภิรมย์รื่น

Tel. : 086-813-5354

e-mail : p.Auoychai@gmail.com

# ทำความรู้จัก Apache Spark:

Apache® Spark™ is a powerful open source processing engine built around speed, ease of use, and sophisticated analytics. It was originally developed at UC Berkeley in 2009.

The team that created Apache Spark founded Databricks in 2013.

https://docs.databricks.com/

https://databricks.com/

# ทำความรู้จัก Apache Spark: (What is Spark)

# Distributed and Highly scalable in-memory data analytic system

# Interoperate with API   Java , Scala , R , Python and SQL

# 100x faster than Hadoop MapReduce in memory , or 10x faster on disk.

# Apache Spark ใช้ทำอะไร :

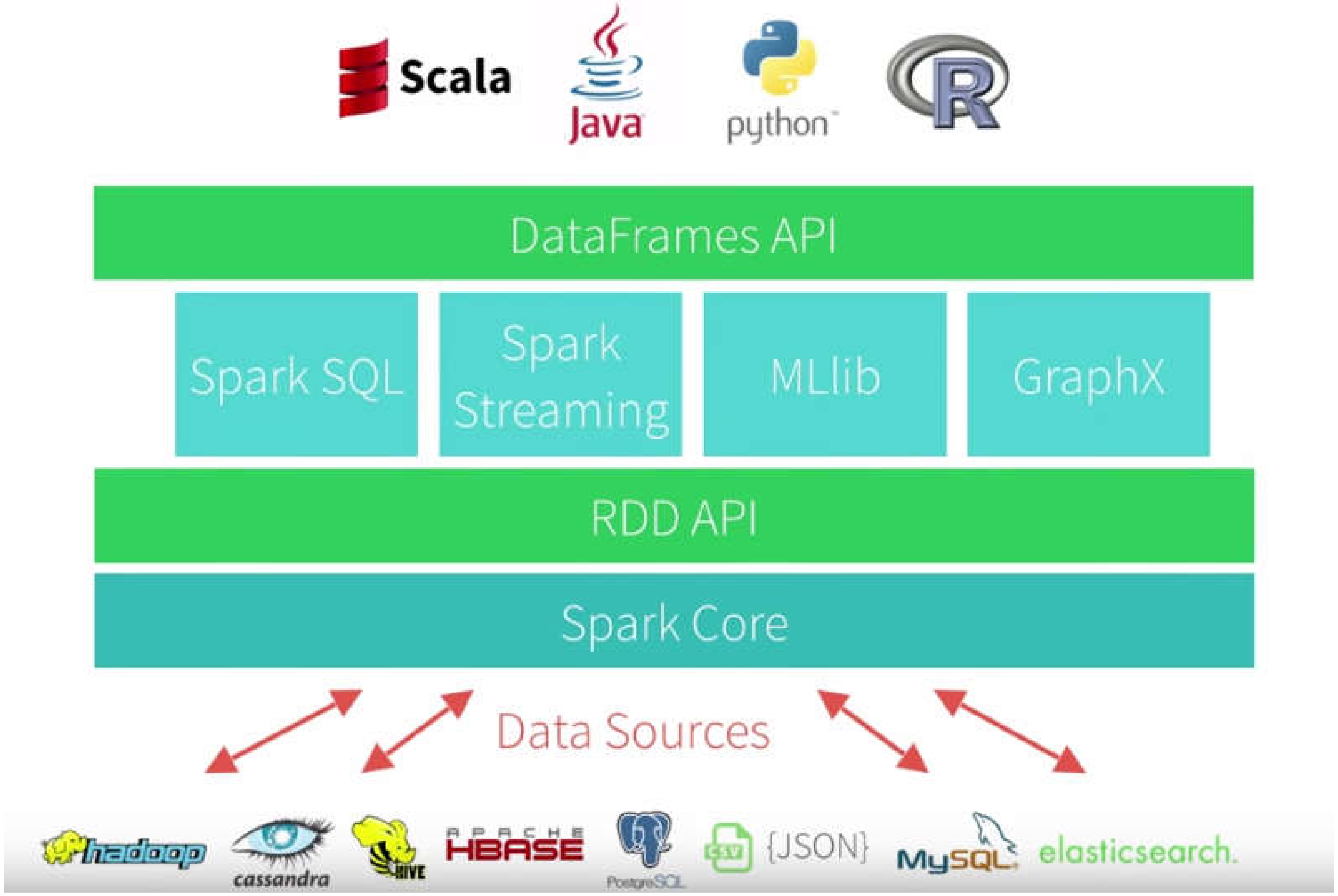
# Library for build data application

# To perform interactive as ad-hoc data analysis

# Running on Laptop , Hadoop , Apache Mesos , Standalone or Cloud.

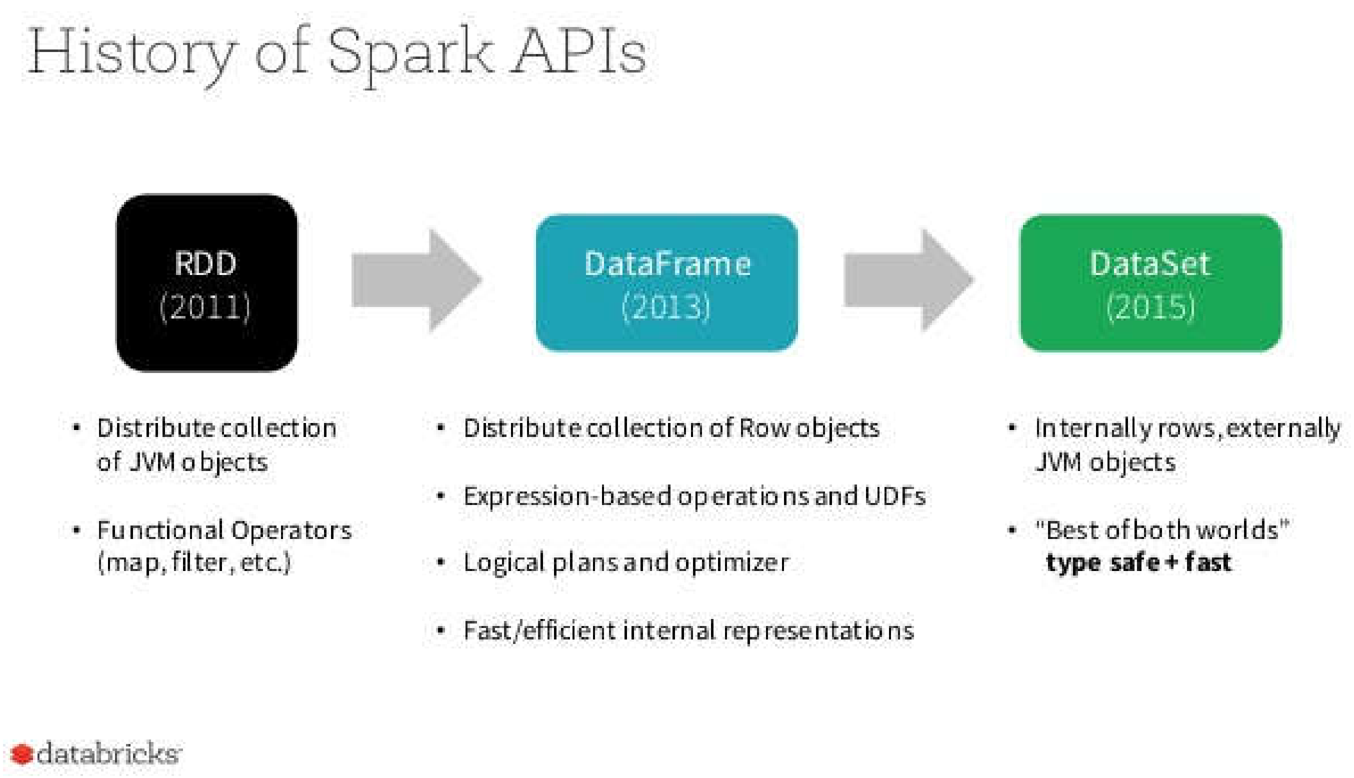
# Support data source including HDFS , Apache Cassandra , Apache Hbase , and S3.

# Apache Spark Ecosystem :



https://www.linkedin.com/pulse/apache-spark-big-data-dataframe-things-know-abhishek-choudhary

# Apache Spark : Core Concept



History of Spark APIs

RDD (2011) → DataFrame (2013) → DataSet (2015)

RDD (2011)

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

DataFrame (2013)

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

DataSet (2015)

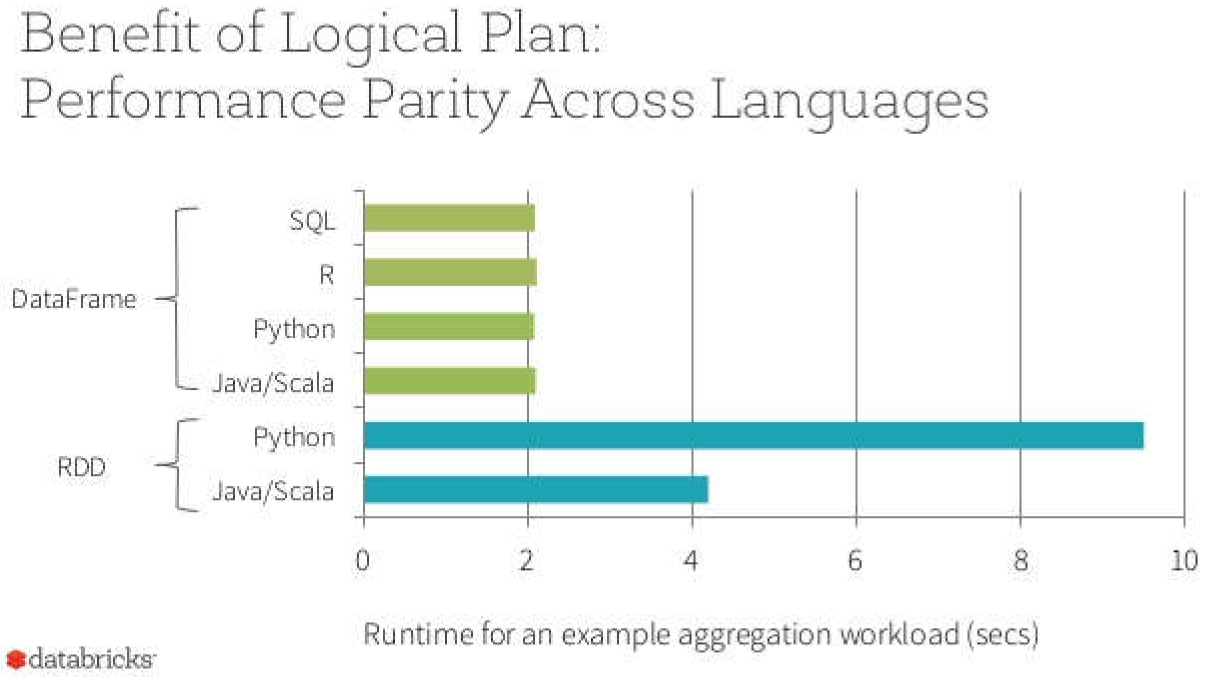
- Internally rows, externally JVM objects
- "Best of both worlds" **type safe + fast**

databricks

https://databricks.com/blog/2016/07/14/a-tale-of-three-apache-spark-apis-rdds-dataframes-and-datasets.html

https://image.slidesharecdn.com/jumpstartintoapachesparkanddatabricks-160212150759/95/jump-start-into-apache-spark-and-databricks-13-638.jpg?cb=1463623478

# Apache Spark : Core Concept



https://www.linkedin.com/pulse/apache-spark-big-data-dataframe-things-know-abhishek-choudhary

# Spark Component:

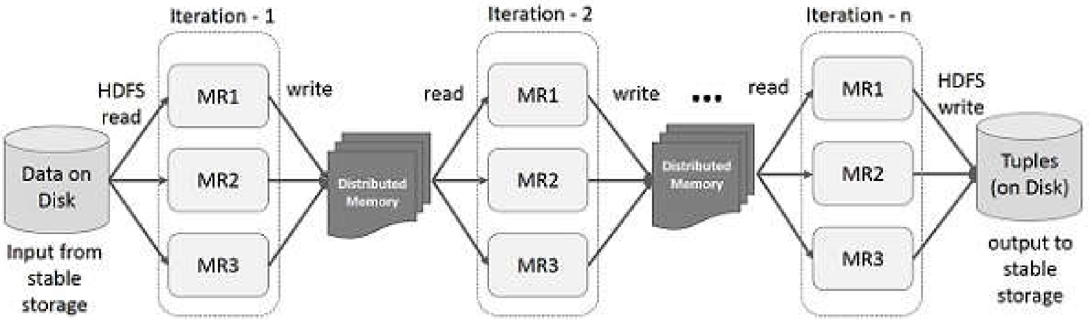
# Spark Core :

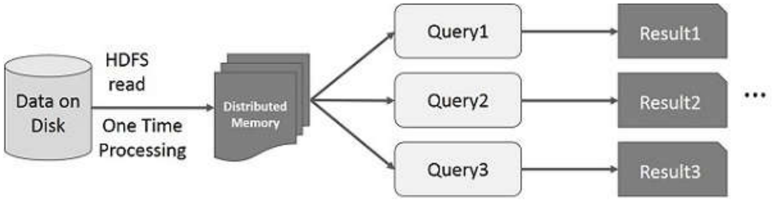
- Spark Core is the underlying general execution engine for spark platform that all other functionality is built upon. It provides In-Memory computing and referencing datasets in external storage systems.

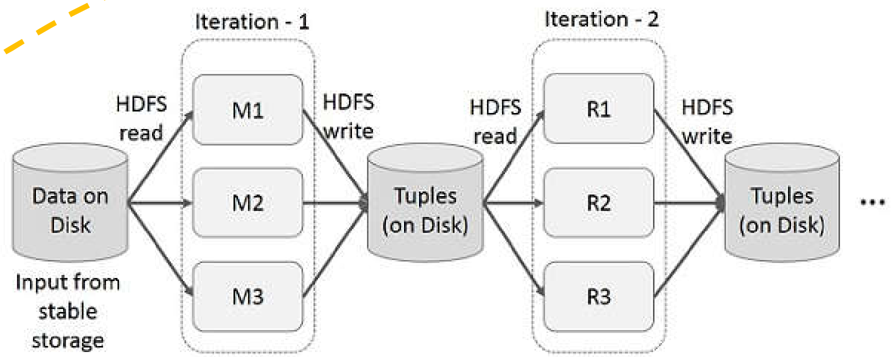
    - Task Scheduling / Memory Management / Fault Recovery
    - Memory Management
    - Realized by Resilient Distributed Datasets (RDDs) - Data collection item.

Spark RDD: 10 to 100 times faster than Hadoop

# Spark Core : Realized by Resilient Distributed Datasets (RDDs) - Data collection item.







replication, serialization, and **disk IO**.

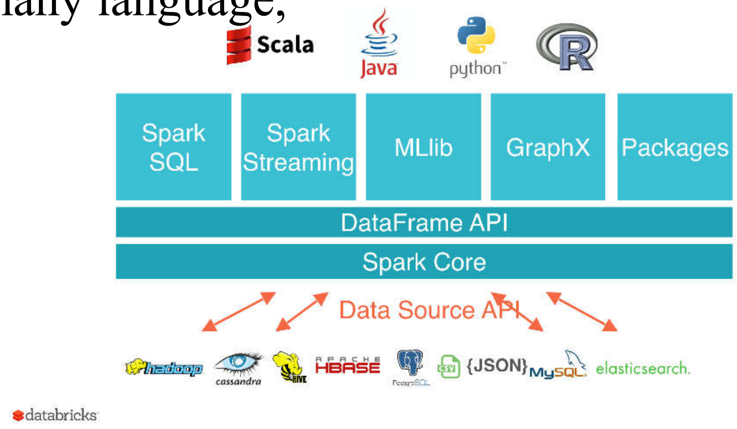
# Spark Component:

# Spark SQL:

Spark SQL is a component on top of Spark Core that introduces a new data abstraction called SchemaRDD, which provides support for structured and semi-structured data.

- Support Querying via SQL
- Support many source of data
    - Hive Table , Parquet , JSON
- Programmatic data manipulation to RDDs in many language,

  Python , Java , Scala

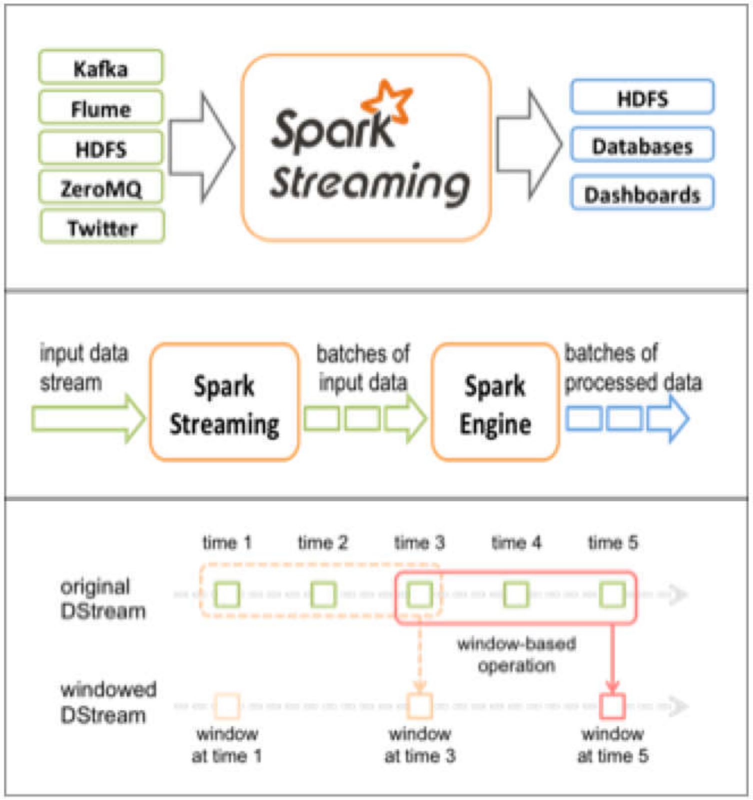


# Spark Component:

# Spark Streaming :

Spark Streaming leverages Spark Core's fast scheduling capability to perform streaming analytics. It ingests data in mini-batches and performs RDD (Resilient Distributed Datasets) transformations on those mini-batches of data.

- Live stream data processing
  - Example, …
- Provided API for manipulating data streams based on RDD API.

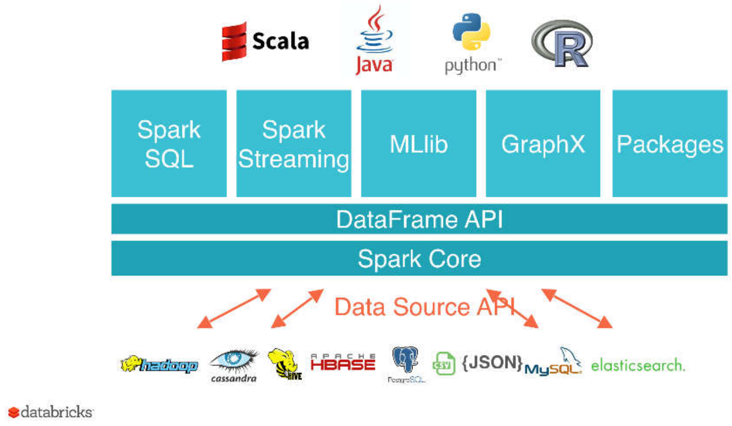


http://techkites.blogspot.com/2015/02/implementing-real-time-trending-engine.html

# Spark Component:

# Spark MLlib (Machine Learning Library) :

- Provides multiple types of machine learning algorithms ,
    - Classification , Regression , Clustering , Collaborative filtering.



Spark SQL

Spark Streaming

MLlib

GraphX

Packages

DataFrame API

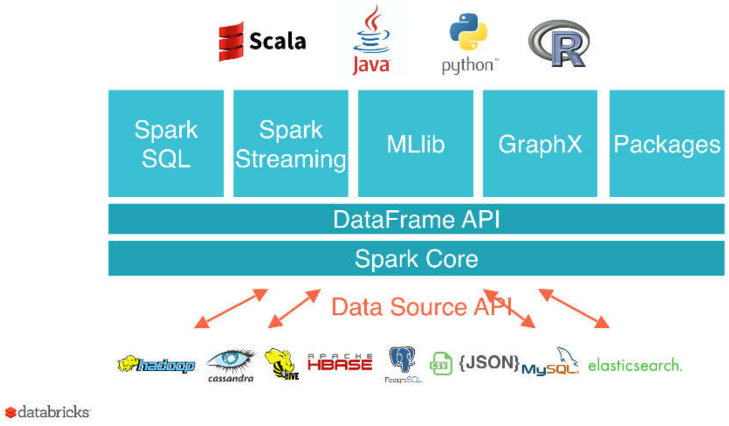
Spark Core

Data Source API

# Spark Component:

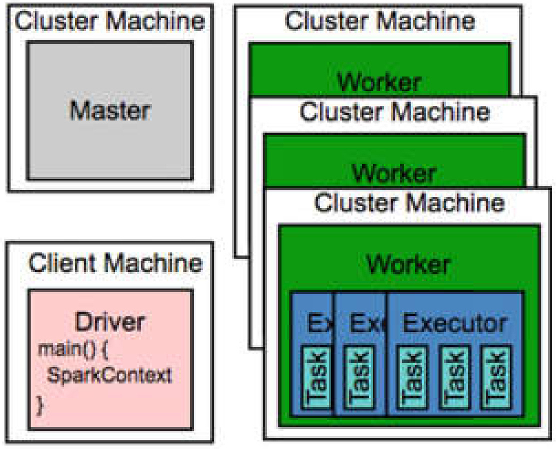
# Spark GraphX :

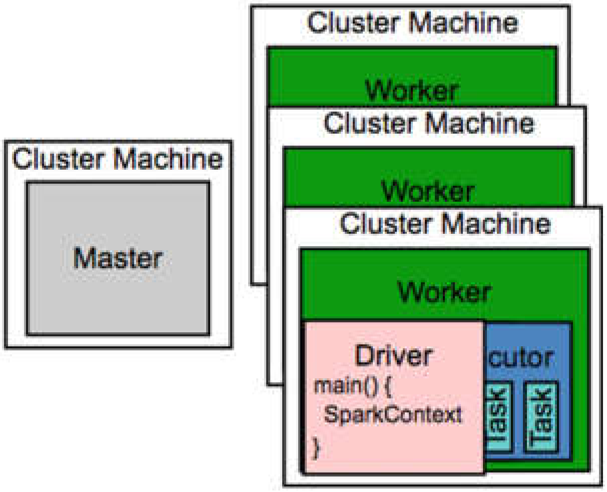
GraphX is a distributed graph-processing framework on top of Spark. It provides an API for expressing graph computation that can model the user-defined graphs by using Pregel abstraction API. It also provides an optimized runtime for this abstraction.



# Spark Deployment Model:

- Local Execution
- Standalone Cluster
- YARN / Cluster

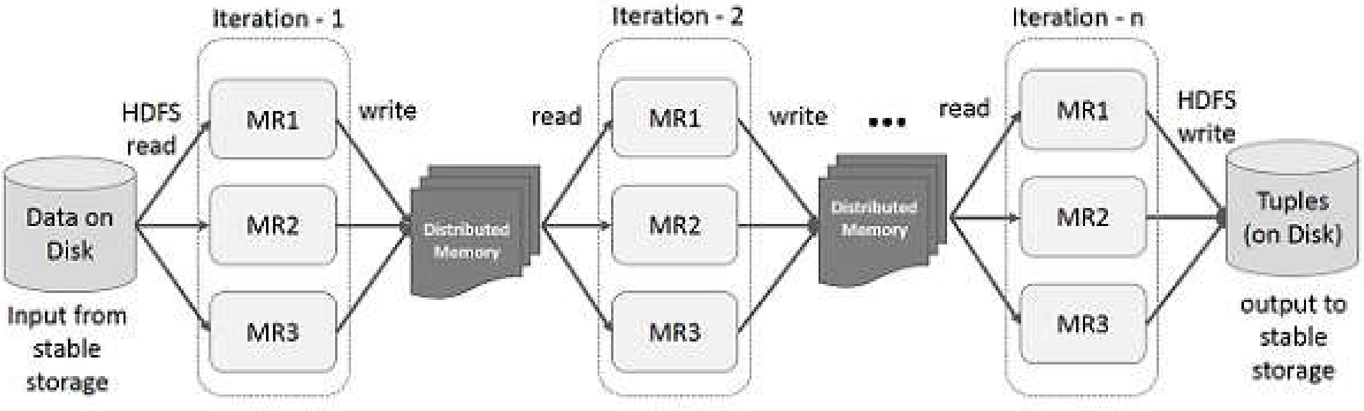


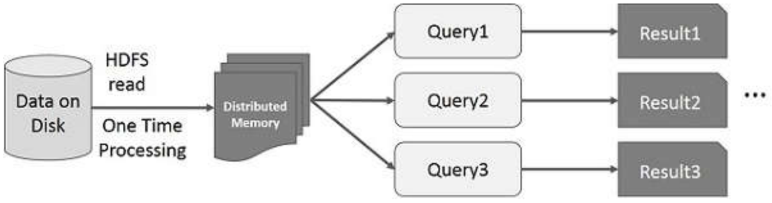


# Spark Programming Model :

- RDD : Resilient Distributed Datasets

  Fundamental data structure of Spark. It is an immutable distributed collection of objects. Each dataset in **RDD** is divided into logical partitions, which may be computed on different nodes of the cluster

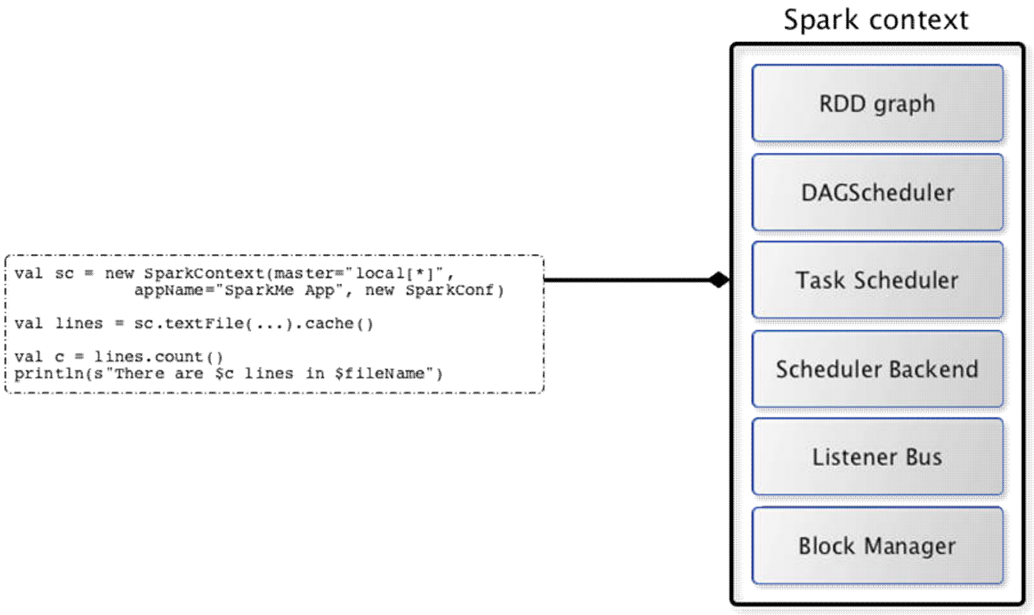




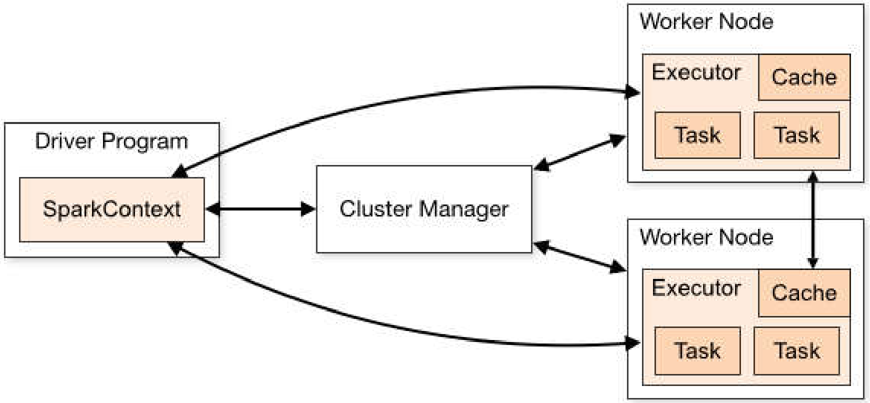
MapReduce on the fly in memory:
- Map , Filter , Reduce

# Spark Programming Model :

# Spark Application Context :



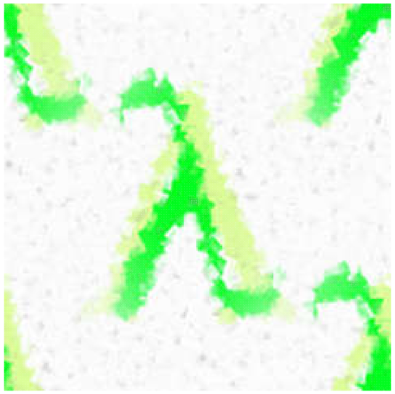
```
val sc = new SparkContext(master="local[*]",
            appName="SparkMe App", new SparkConf)

val lines = sc.textFile(...).cache()

val c = lines.count()
println(s"There are $c lines in $fileName")
```



# Core Concept Map/Reduce Prgramming :

- Basic concept Map-Filter-Reduce Function

  - All apply with List Data Structure:



- Lambda Function

The lambda operator or lambda function is a way to create small anonymous functions, i.e. functions without a name.

```
>> f = lambda x , y : y: x+y
>> f ( 1 , 1 )   // 2
```

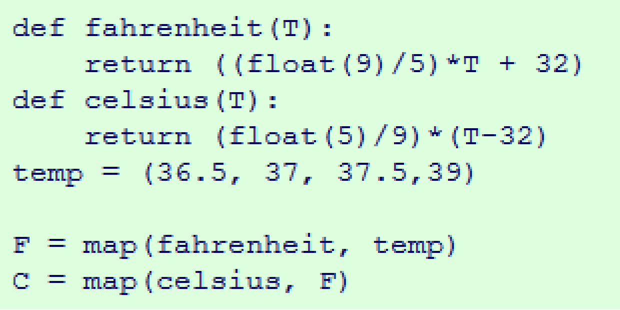
http://www.python-course.eu/lambda.php

# Core Concept Map/Reduce Prgramming :

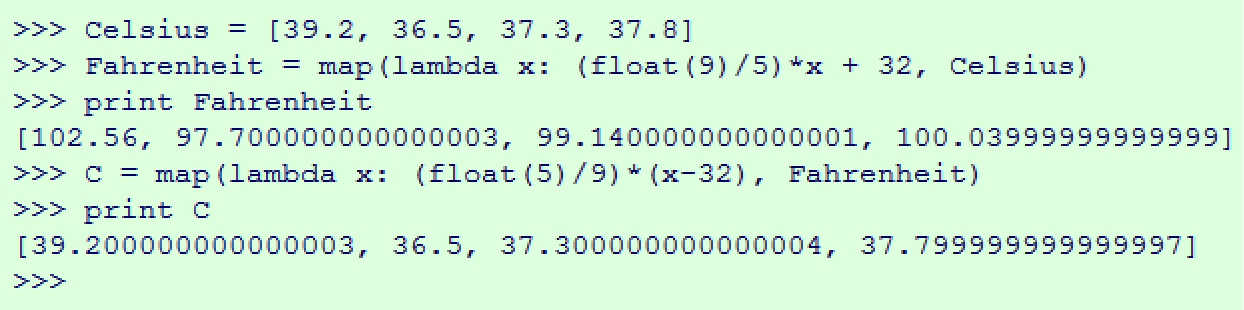
- Basic concept Map-Filter-Reduce Function

  - The map( ) Function

  The first argument *func* is the name of a function and the second a sequence (e.g. a list) *seq*. *map()* applies the function *func* to all the elements of the sequence *seq*. It returns a new list with the elements changed by *func*



```
def fahrenheit(T):
    return ((float(9)/5)*T + 32)
def celsius(T):
    return (float(5)/9)*(T-32)
temp = (36.5, 37, 37.5,39)

F = map(fahrenheit, temp)
C = map(celsius, F)
```



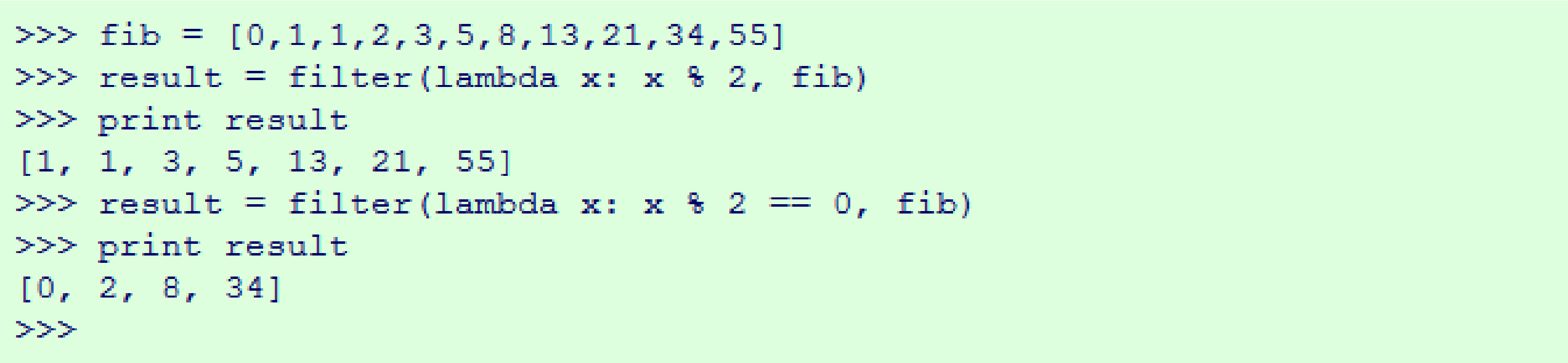
```
>>> Celsius = [39.2, 36.5, 37.3, 37.8]
>>> Fahrenheit = map(lambda x: (float(9)/5)*x + 32, Celsius)
>>> print Fahrenheit
[102.56, 97.700000000000003, 99.140000000000001, 100.039999999999999]
>>> C = map(lambda x: (float(5)/9)*(x-32), Fahrenheit)
>>> print C
[39.200000000000003, 36.5, 37.300000000000004, 37.799999999999997]
>>>
```

# Core Concept Map/Reduce Prgramming :

- The Filter( ) Function

The function filter(function, list) offers an elegant way to filter out all the elements of a list, for which the function *function* returns True.

The function filter(f,l) needs a function f as its first argument. f returns a Boolean value, i.e. either True or False. This function will be applied to every element of the list *l*. Only if f returns True will the element of the list be included in the result list.

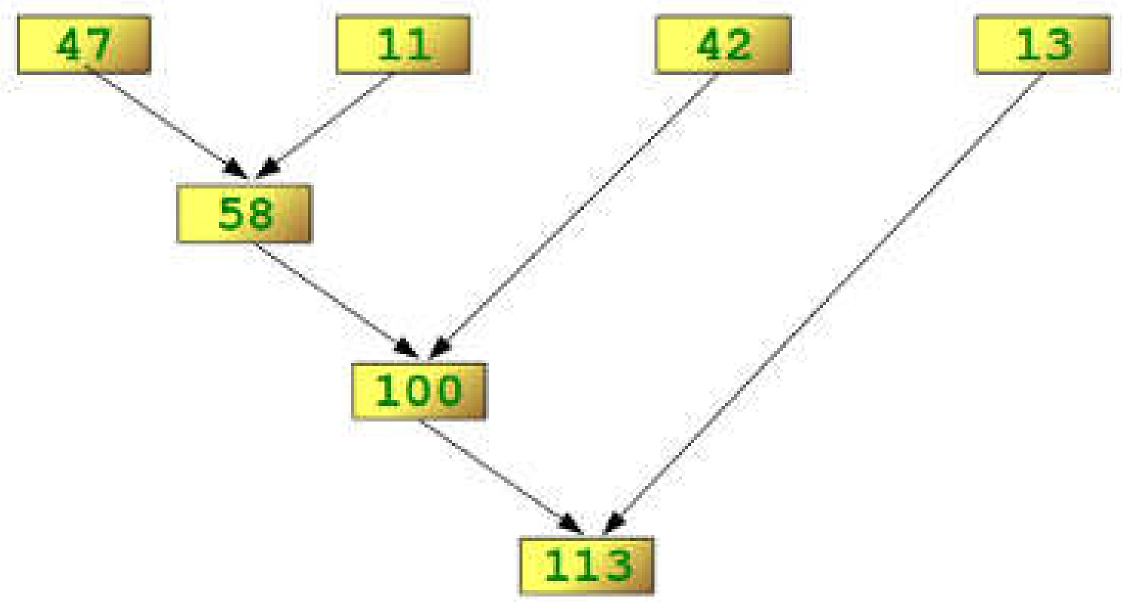


```
>>> fib = [0,1,1,2,3,5,8,13,21,34,55]
>>> result = filter(lambda x: x % 2, fib)
>>> print result
[1, 1, 3, 5, 13, 21, 55]
>>> result = filter(lambda x: x % 2 == 0, fib)
>>> print result
[0, 2, 8, 34]
>>>
```

# Core Concept Map/Reduce Prgramming :

- The Reduce ( ) Function

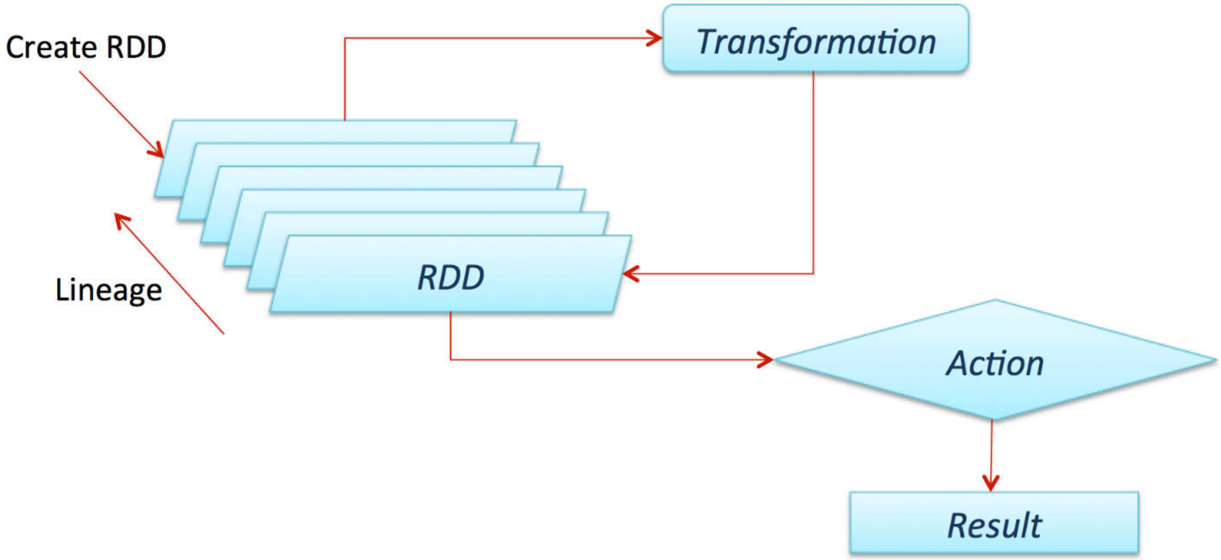
```
>>> reduce(lambda x,y: x+y, [47,11,42,13])
113
```



# Spark Core Concept : RDD Operation

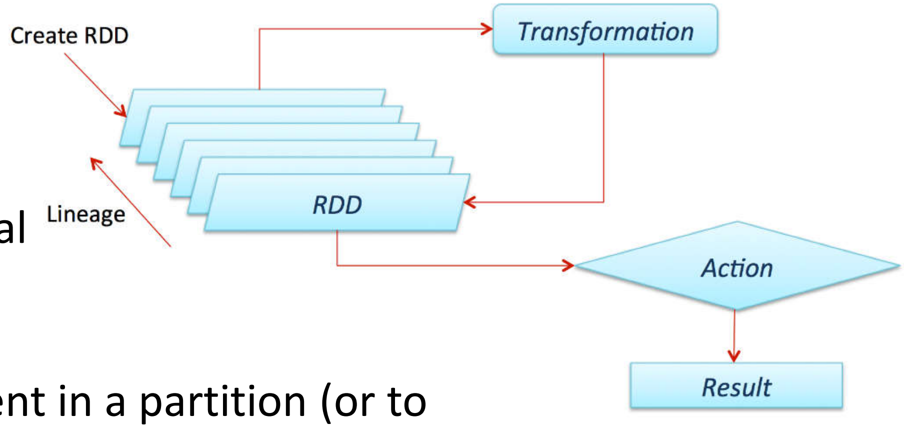
■ RDD : Workflow

1. Create an RDD from a data source.
2. Apply transformations to an RDD.
3. Apply actions to an RDD.



https://media.licdn.com/mpr/mpr/AAEAAQAAAAAAAAszAAAAJGE5M2UwYzlmLTcxYWEtNGVhNS1iZWFkLWI0ZWE1ZjlkZjIwMg.png

# Spark Core Concept : RDD Operation



RDD : Operation

Operations on RDDs are divided into several groups:

•Transformations

- apply user function to every element in a partition (or to the whole partition)
- apply aggregation function to the whole dataset (groupBy, sortBy)
- introduce dependencies between RDDs to form DAG
- provide functionality for repartitioning (repartition, partitionBy)

•Actions

- trigger job execution
- used to materialize computation results

•Extra: persistence

- explicitly store RDDs in memory, on disk or off-heap (cache, persist)
- checkpointing for truncating RDD lineage

https://media.licdn.com/mpr/mpr/AAEAAQAAAAAAAAszAAAAJGE5M2UwYzlmLTcxYWEtNGVhNS1iZWFkLWI0ZWE1ZjlkZjIwMg.png

# Programming Model : RDD Operation

Transformations & Meaning

map(func)
Returns a new distributed dataset, formed by passing each element of the source through a function **func**.

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).

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).

mapPartitions(func)
Similar to map, but runs separately on each partition (block) of the RDD, so **func** must be of type Iterator<T> ⇒ Iterator<U> when running on an RDD of type T.

etc:
https://www.tutorialspoint.com/apache_spark/apache_spark_core_programming.htm

# Programming Model : RDD Operation

Action & Meaning

reduce(func)
Aggregate the elements of the dataset using a function **func** (which takes two arguments and returns one). The function should be commutative and associative so that it can be computed correctly in parallel.

collect()
Returns all the elements of the dataset as an array at the driver program. This is usually useful after a filter or other operation that returns a sufficiently small subset of the data.

count()
Returns the number of elements in the dataset.

first()
Returns the first element of the dataset (similar to take (1)).

take(n)
Returns an array with the first **n** elements of the dataset.

etc:
https://www.tutorialspoint.com/apache_spark/apache_spark_core_programming.htm

# Programming Model : RDD Operation

Persistence : Storage Level

MEMORY_ONLY
Store RDD as deserialized Java objects in the JVM. If the RDD does not fit in memory, some partitions will not be cached and will be recomputed on the fly each time they're needed. This is the default level.

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.

MEMORY_ONLY_SER
Store RDD as *serialized* Java objects (one byte array per partition). This is generally more space-efficient than deserialized objects, especially when using a fast serializer, but more CPU-intensive to read.

MEMORY_AND_DISK_SER
Similar to MEMORY_ONLY_SER, but spill partitions that don't fit in memory to disk instead of recomputing them on the fly each time they're needed.

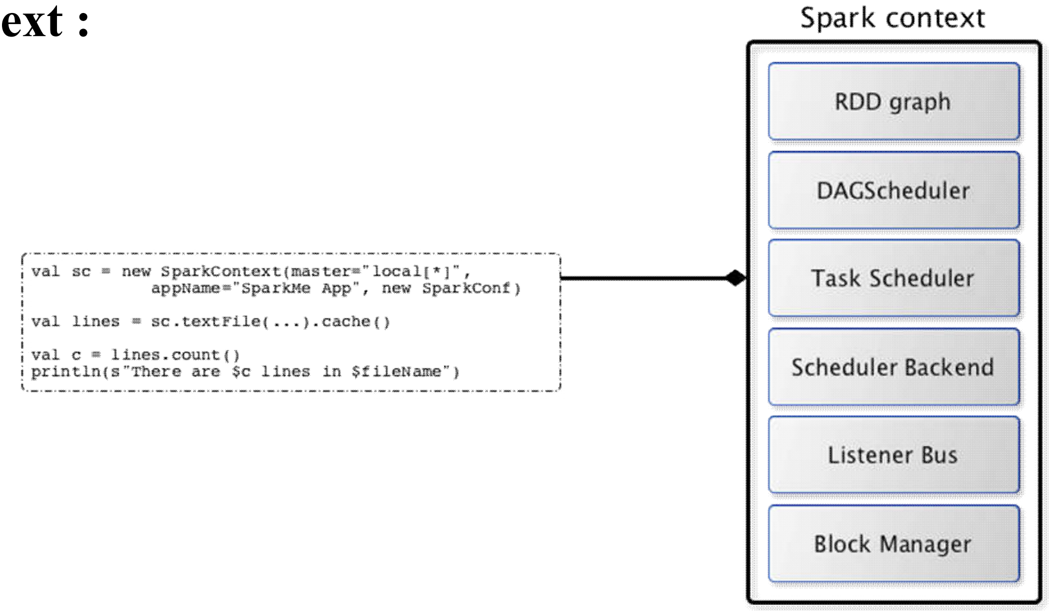
DISK_ONLY
Store the RDD partitions only on disk.

MEMORY_ONLY_2, MEMORY_AND_DISK_2, etc.
Same as the levels above, but replicate each partition on two cluster nodes.

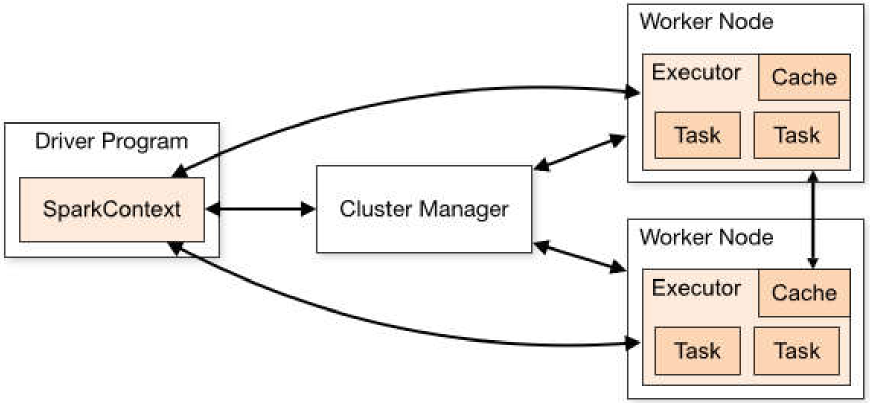
# Programming Model : RDD Operation

# Spark Application Context :

- SparkContext
- SQLContect
- HiveContext
- StreamingContext



```
val sc = new SparkContext(master="local[*]",
                appName="SparkMe App", new SparkConf)

val lines = sc.textFile(...).cache()

val c = lines.count()
println(s"There are $c lines in $fileName")
```



https://jaceklaskowski.gitbooks.io/mastering-apache-spark/content/spark-sparkcontext.html
https://blogs.msdn.microsoft.com/bigdatasupport/2015/09/14/understanding-sparks-sparkconf-sparkcontext-sqlcontext-and-hivecontext/

# Programming Guild :

- RDD

  https://spark.apache.org/docs/latest/programming-guide.html

- DataFrame (SQL)

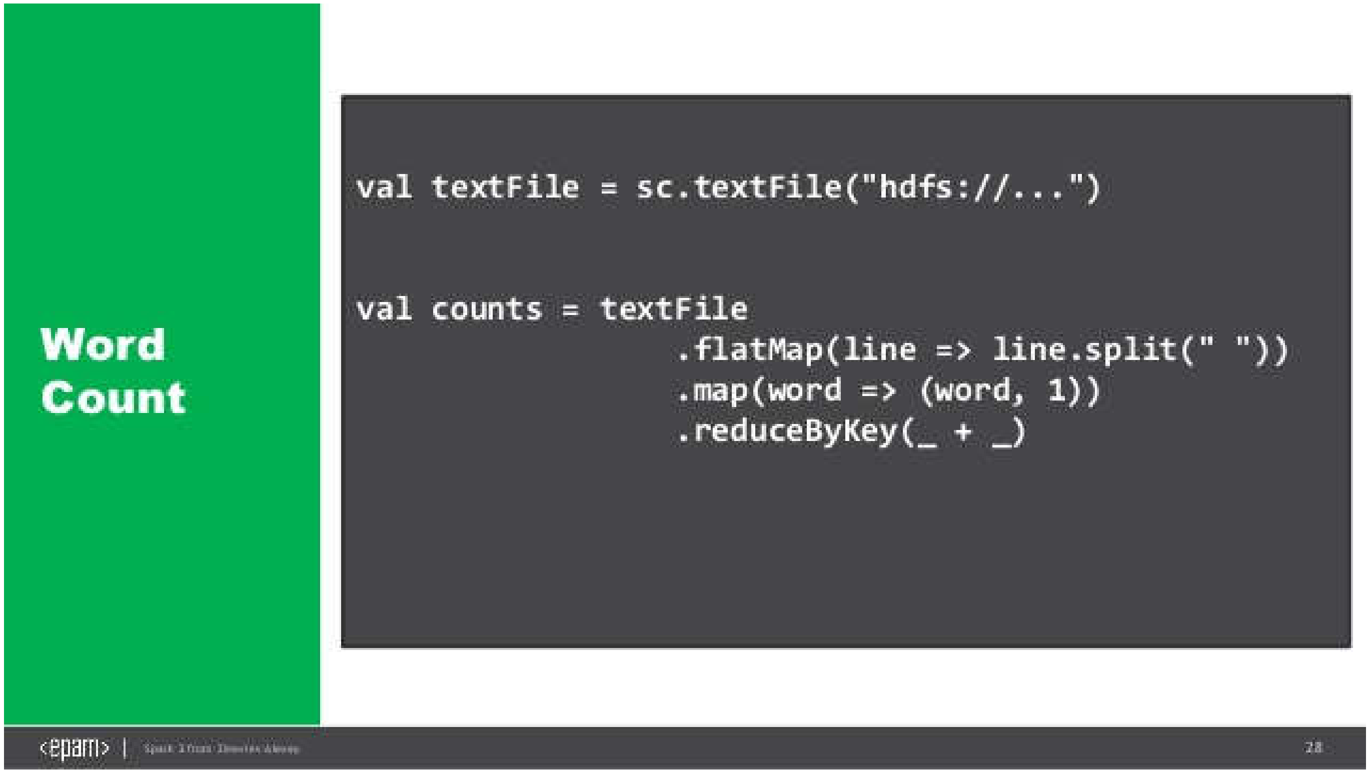
  https://spark.apache.org/docs/latest/sql-programming-guide.html

- Dstream (Streamming)

  https://spark.apache.org/docs/latest/structured-streaming-programming-guide.html

# Programming Review : RDD

- Word Count:

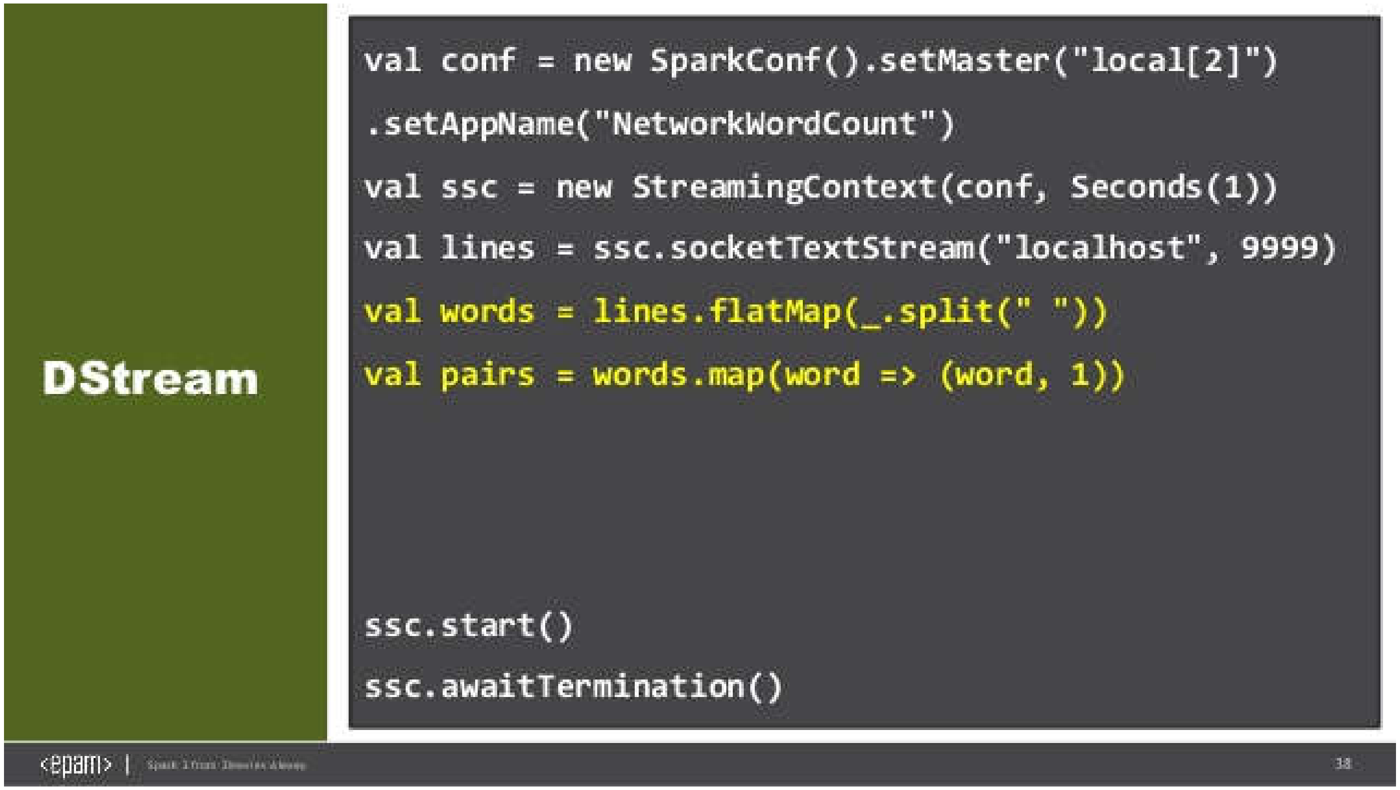


Word Count

```
val textFile = sc.textFile("hdfs://...")

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

https://image.slidesharecdn.com/spark2zinovievforit-subbotnik-161028213227/95/joker16-spark-2-api-changes-structured-streaming-encoders-28-638.jpg?cb=1483965803

# Programming Review : Streaming



DStream

```
val conf = new SparkConf().setMaster("local[2]")
.setAppName("NetworkWordCount")
val ssc = new StreamingContext(conf, Seconds(1))
val lines = ssc.socketTextStream("localhost", 9999)
val words = lines.flatMap(_.split(" "))
val pairs = words.map(word => (word, 1))



ssc.start()
ssc.awaitTermination()
```

<epam> | Spark 2 from Zinoviev Alexey 38

https://image.slidesharecdn.com/spark2zinovievforit-subbotnik-161028213227/95/joker16-spark-2-api-changes-structured-streaming-encoders-38-638.jpg?cb=1483965803

# Hand-On:

- Spark Installation
- Spark Programming
  - RDD
  - SQL
  - Streaming

# Spark Installation:

- Download & Install Scalar (http://www.scala-lang.org/download/)

```
sudo apt-get update
sudo apt-get install scala
```

- Download & Install Spark

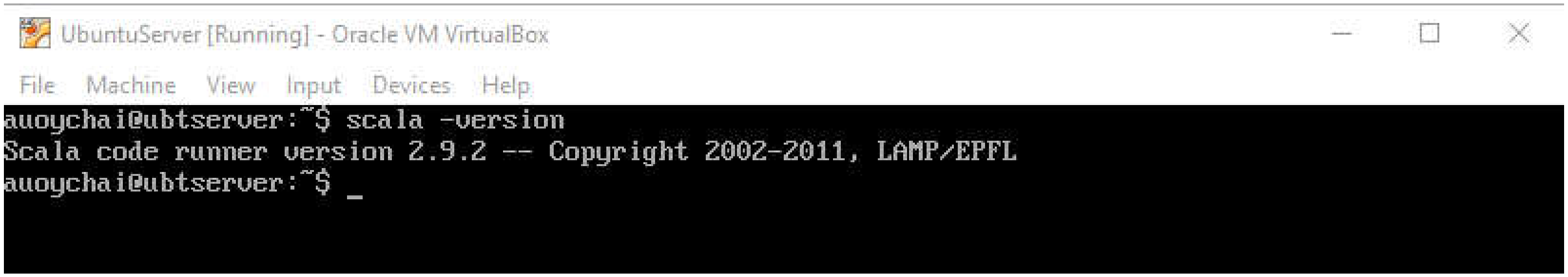
```
wget  https://d3kbcqa49mib13.cloudfront.net/spark-2.1.1-bin-hadoop2.7.tgz
tar xvf spark-2.1.1.bin-hadoop2.6.tgz
mv spark-2.1.1.bin-hadoop2.6   /usr/local/spark
Sudo chown –R /usr/local/spark
```

** Set Environment Variable (.bashrc)

# Verify Installation Completed :

- Scalar

$scala -version



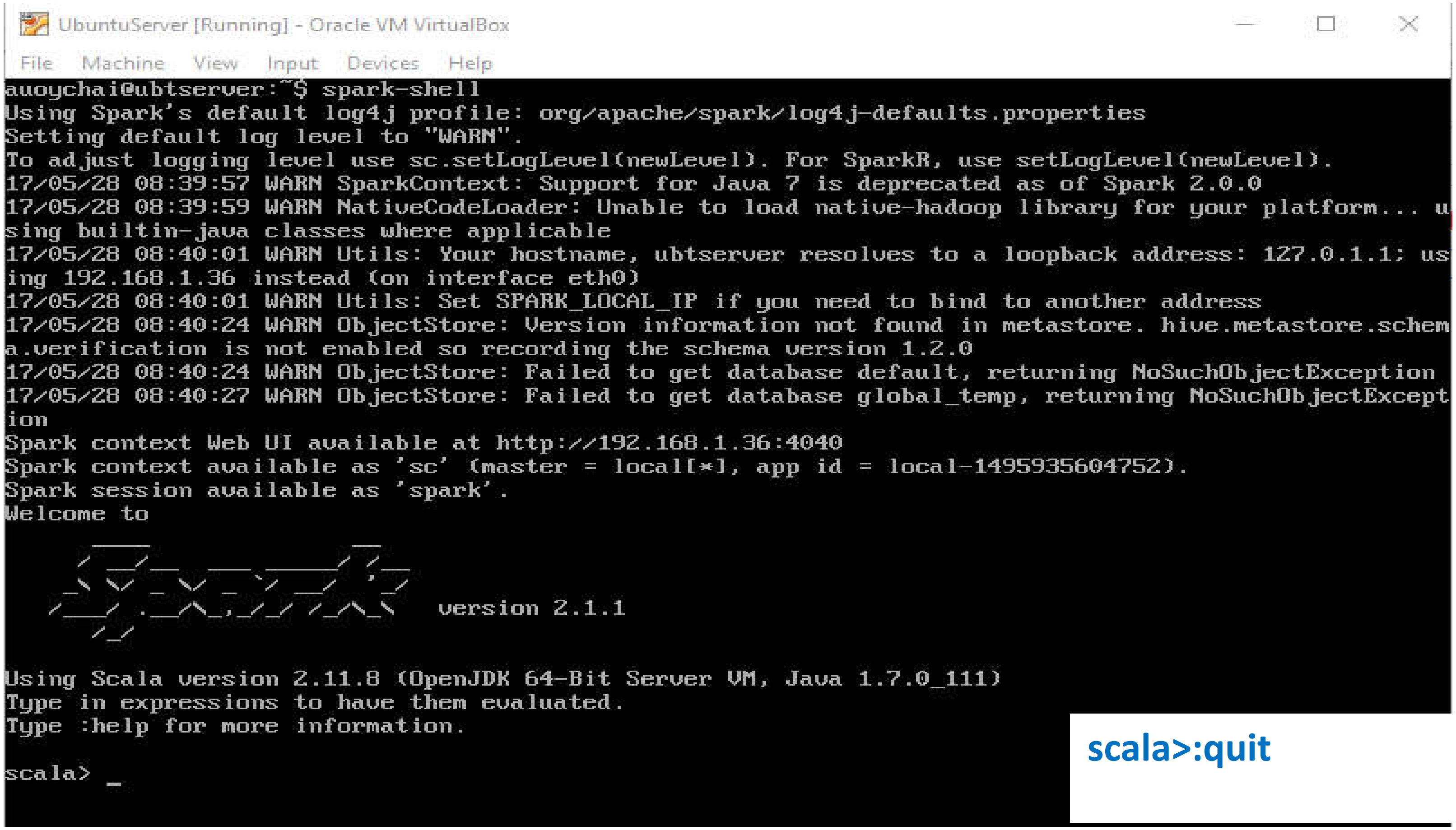
UbuntuServer [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help

```
auoychai@ubtserver:~$ scala -version
Scala code runner version 2.9.2 -- Copyright 2002-2011, LAMP/EPFL
auoychai@ubtserver:~$ _
```

# Verify Installation Completed :

- Spark

$spark-shell

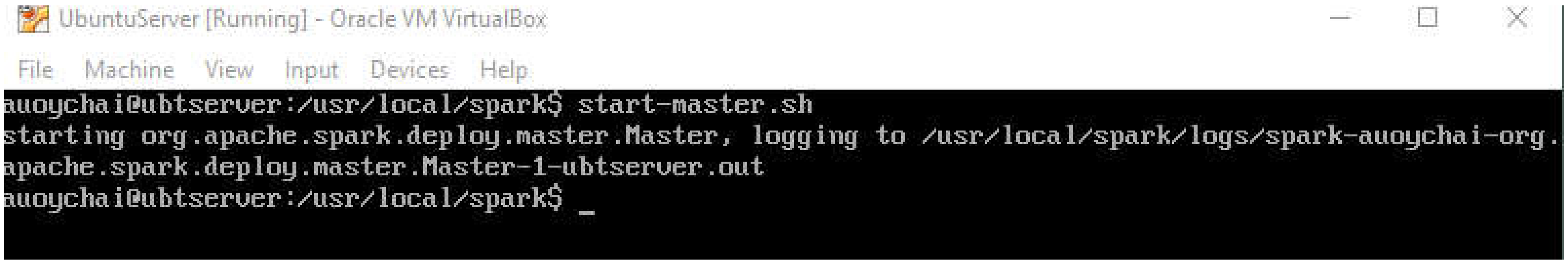

UbuntuServer [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help

```
auoychai@ubtserver:~$ spark-shell
Using Spark's default log4j profile: org/apache/spark/log4j-defaults.properties
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLevel(newLevel).
17/05/28 08:39:57 WARN SparkContext: Support for Java 7 is deprecated as of Spark 2.0.0
17/05/28 08:39:59 WARN NativeCodeLoader: Unable to load native-hadoop library for your platform... u
sing builtin-java classes where applicable
17/05/28 08:40:01 WARN Utils: Your hostname, ubtserver resolves to a loopback address: 127.0.1.1; us
ing 192.168.1.36 instead (on interface eth0)
17/05/28 08:40:01 WARN Utils: Set SPARK_LOCAL_IP if you need to bind to another address
17/05/28 08:40:24 WARN ObjectStore: Version information not found in metastore. hive.metastore.schem
a.verification is not enabled so recording the schema version 1.2.0
17/05/28 08:40:24 WARN ObjectStore: Failed to get database default, returning NoSuchObjectException
17/05/28 08:40:27 WARN ObjectStore: Failed to get database global_temp, returning NoSuchObjectExcept
ion
Spark context Web UI available at http://192.168.1.36:4040
Spark context available as 'sc' (master = local[*], app id = local-1495935604752).
Spark session available as 'spark'.
Welcome to
      ____              __
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ `/ __/  '_/
   /___/ .__/\_,_/_/ /_/\_\   version 2.1.1
      /_/

Using Scala version 2.11.8 (OpenJDK 64-Bit Server VM, Java 1.7.0_111)
Type in expressions to have them evaluated.
Type :help for more information.

scala> _
```

scala>:quit

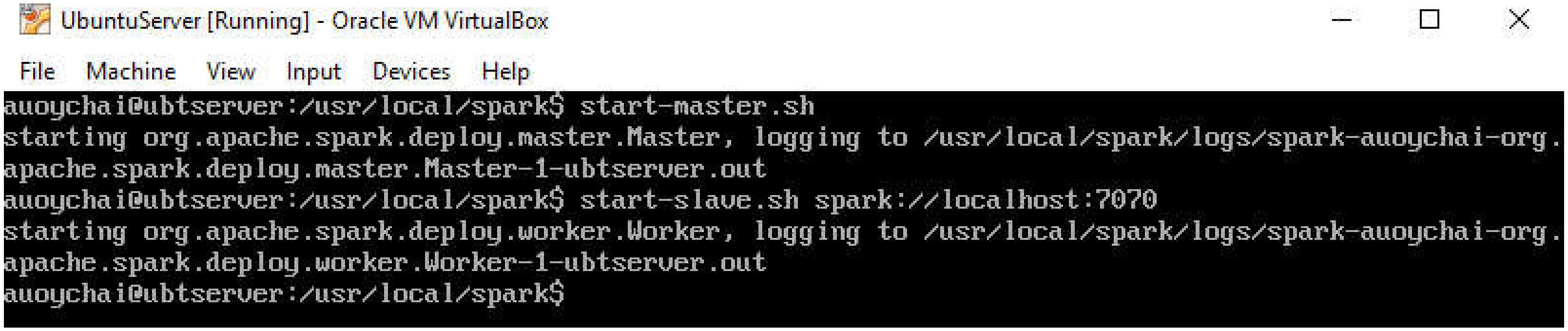
# How to Start Spark:

:/usr/local/spark$start-master.sh

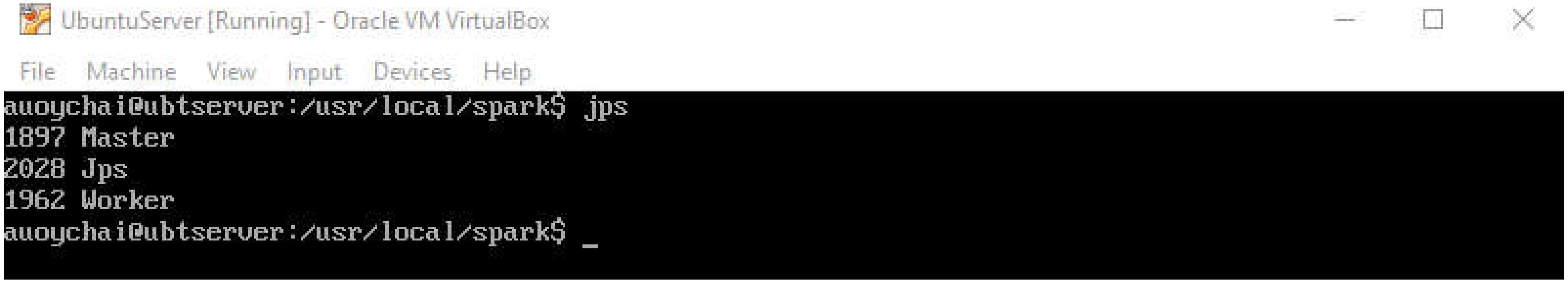


```
auoychai@ubtserver:/usr/local/spark$ start-master.sh
starting org.apache.spark.deploy.master.Master, logging to /usr/local/spark/logs/spark-auoychai-org.
apache.spark.deploy.master.Master-1-ubtserver.out
auoychai@ubtserver:/usr/local/spark$ _
```

:/usr/local/spark$start-slave.sh spark://localhost:7077



```
auoychai@ubtserver:/usr/local/spark$ start-master.sh
starting org.apache.spark.deploy.master.Master, logging to /usr/local/spark/logs/spark-auoychai-org.
apache.spark.deploy.master.Master-1-ubtserver.out
auoychai@ubtserver:/usr/local/spark$ start-slave.sh spark://localhost:7070
starting org.apache.spark.deploy.worker.Worker, logging to /usr/local/spark/logs/spark-auoychai-org.
apache.spark.deploy.worker.Worker-1-ubtserver.out
auoychai@ubtserver:/usr/local/spark$
```



```
auoychai@ubtserver:/usr/local/spark$ jps
1897 Master
2028 Jps
1962 Worker
auoychai@ubtserver:/usr/local/spark$ _
```

# Started Check-up :

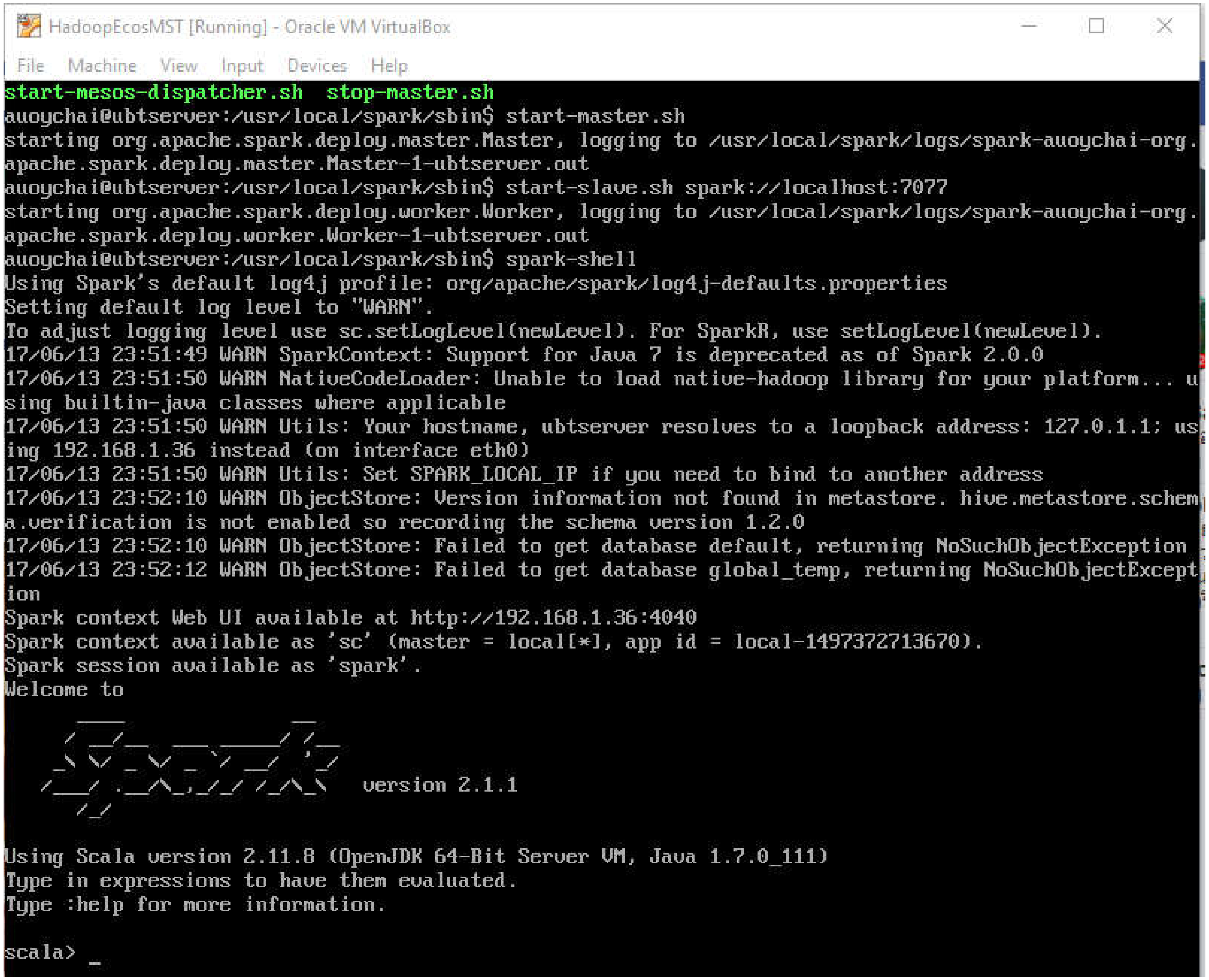
-- Master:Port:7077

-- Slave:

-- UI: http://[IP]:8080

# Spark RDD Programming : Start Spark Shell

$spark-shell



```
start-mesos-dispatcher.sh  stop-master.sh
auoychai@ubtserver:/usr/local/spark/sbin$ start-master.sh
starting org.apache.spark.deploy.master.Master, logging to /usr/local/spark/logs/spark-auoychai-org.
apache.spark.deploy.master.Master-1-ubtserver.out
auoychai@ubtserver:/usr/local/spark/sbin$ start-slave.sh spark://localhost:7077
starting org.apache.spark.deploy.worker.Worker, logging to /usr/local/spark/logs/spark-auoychai-org.
apache.spark.deploy.worker.Worker-1-ubtserver.out
auoychai@ubtserver:/usr/local/spark/sbin$ spark-shell
Using Spark's default log4j profile: org/apache/spark/log4j-defaults.properties
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLevel(newLevel).
17/06/13 23:51:49 WARN SparkContext: Support for Java 7 is deprecated as of Spark 2.0.0
17/06/13 23:51:50 WARN NativeCodeLoader: Unable to load native-hadoop library for your platform... u
sing builtin-java classes where applicable
17/06/13 23:51:50 WARN Utils: Your hostname, ubtserver resolves to a loopback address: 127.0.1.1; us
ing 192.168.1.36 instead (on interface eth0)
17/06/13 23:51:50 WARN Utils: Set SPARK_LOCAL_IP if you need to bind to another address
17/06/13 23:52:10 WARN ObjectStore: Version information not found in metastore. hive.metastore.schem
a.verification is not enabled so recording the schema version 1.2.0
17/06/13 23:52:10 WARN ObjectStore: Failed to get database default, returning NoSuchObjectException
17/06/13 23:52:12 WARN ObjectStore: Failed to get database global_temp, returning NoSuchObjectExcept
ion
Spark context Web UI available at http://192.168.1.36:4040
Spark context available as 'sc' (master = local[*], app id = local-1497372713670).
Spark session available as 'spark'.
Welcome to
      ____              __
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ `/ __/  '_/
   /___/ .__/\_,_/_/ /_/\_\   version 2.1.1
      /_/

Using Scala version 2.11.8 (OpenJDK 64-Bit Server VM, Java 1.7.0_111)
Type in expressions to have them evaluated.
Type :help for more information.

scala> _
```

# Spark RDD Programming : WordCound

```
scala> val file = sc.textFile(“/home/auoychai/dataset/license.txt”)

scala> file.collect()

scala>file.count()

scala>file.first()

scala>file.take(5)

scala>file.takeOrder(5)

scala> val wc = file.flatMap(l=>l.split(“ ”)).map(word=>(word,1)).reduceByKey(_+_)

scala>wc.saveAsTextFile(“/home/auoychai/wclicense”)
```

# Spark RDD Programming : WordCound

```
scala> val file = sc.textFile(“hdfs:///user/inuput/license.txt”)

scala> val wc = file.flatMap(l=>l.split(“
”)).map(word=>(word,1)).reduceByKey(_+_)

scala>wc.saveAsTextFile(“hdfs:///user/output/wclicense”)
```

# Spark SQL Programming :

# Spark Streaming Programming :

The End