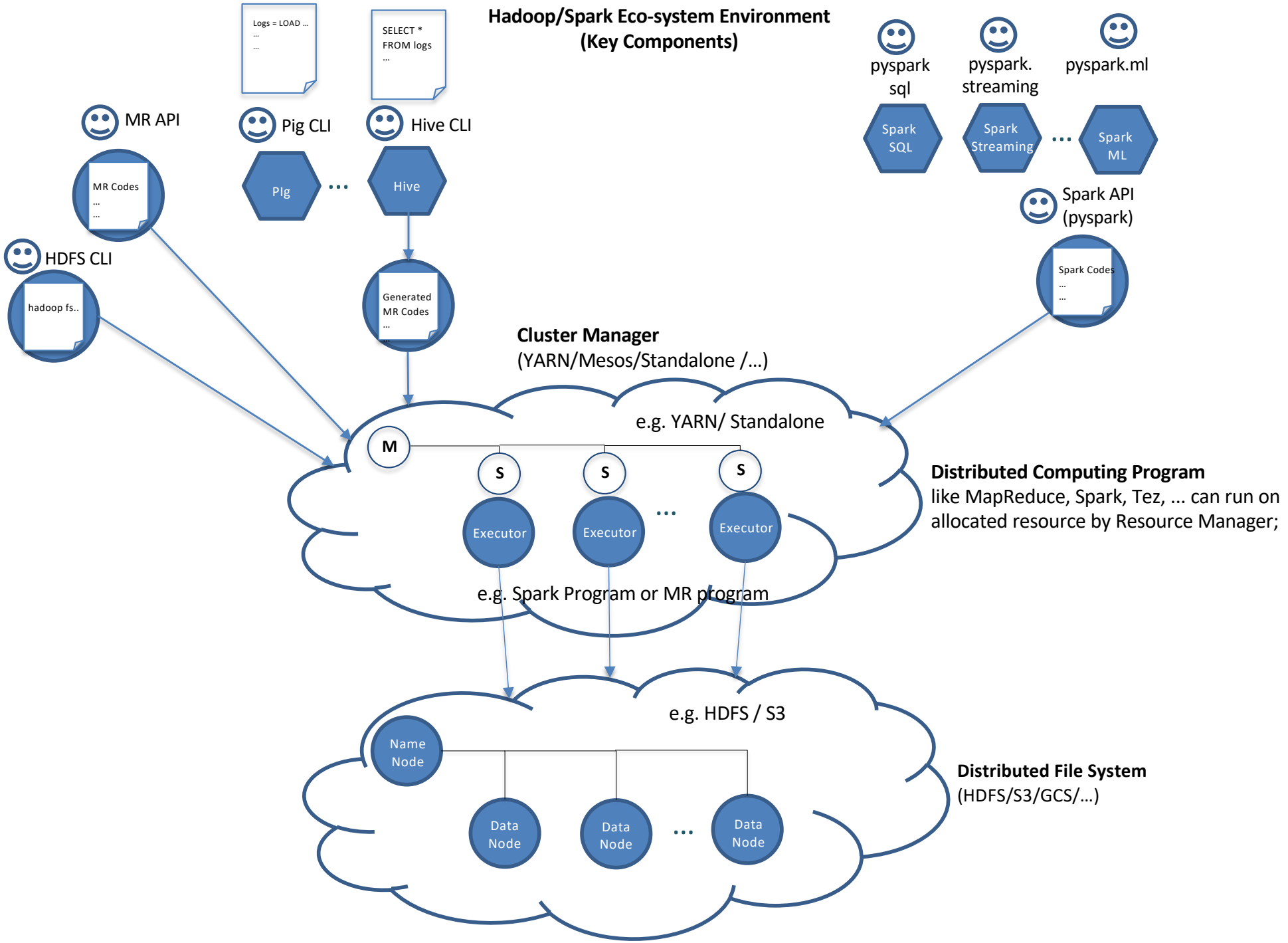
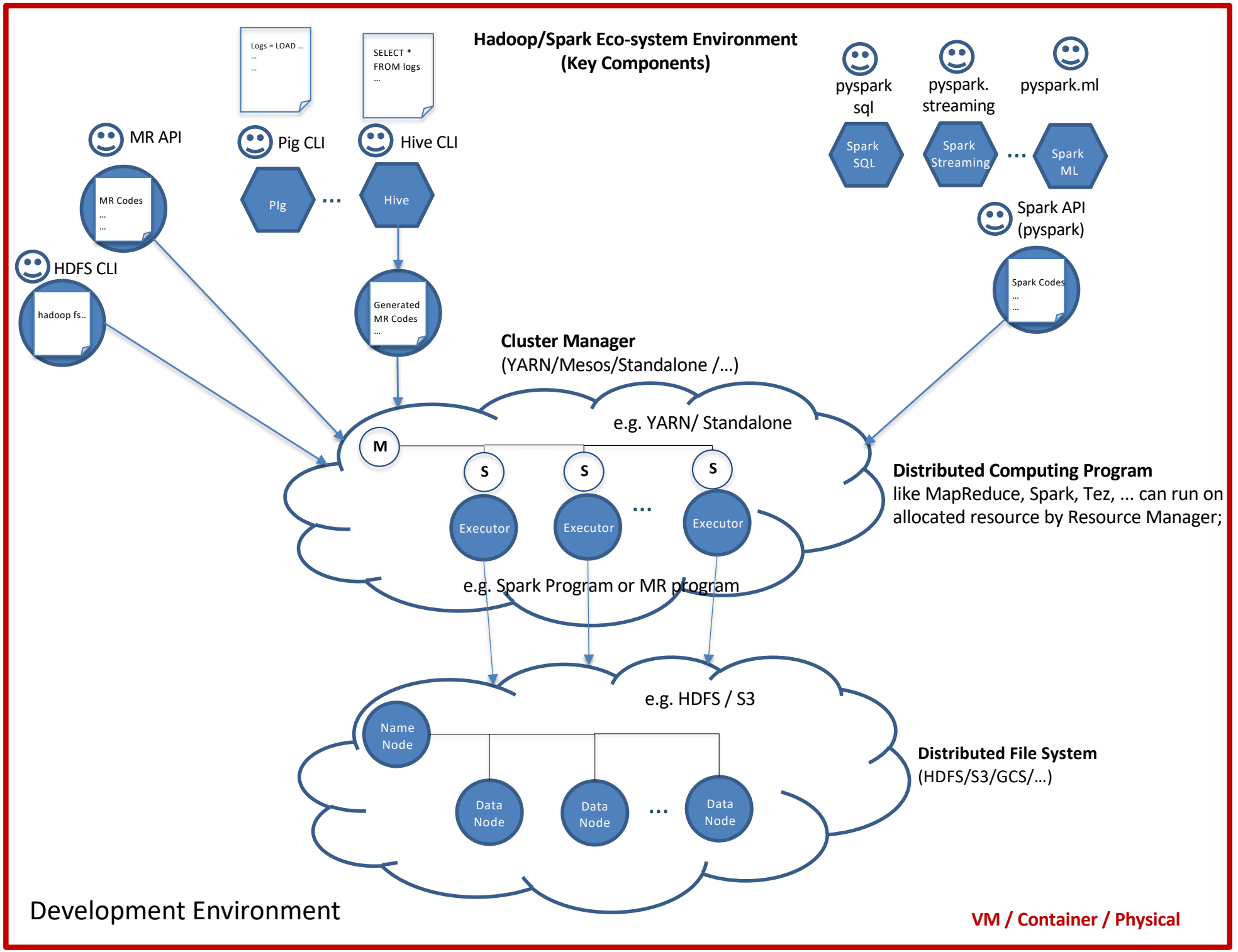
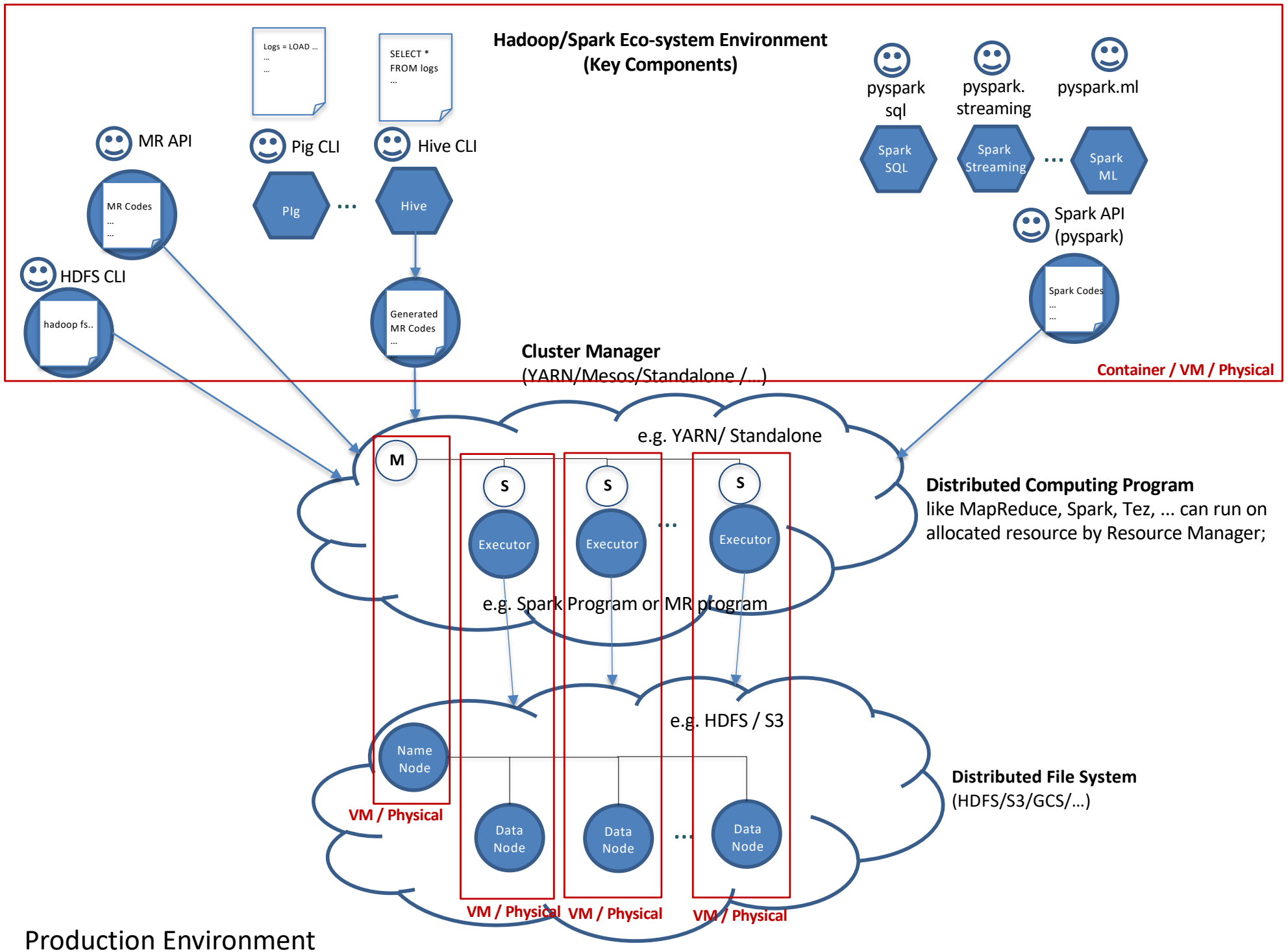
Spark Core Introduction

Hadoop/Spark Eco-system Overview (Cont.)


Hadoop/Spark Eco-system Environment
(Key Components)
Logs = LOAD ...
...
...
SELECT *
FROM logs
...
MR API
Pig CLI
Hive CLI
pyspark
sql
pyspark.
streaming
pyspark.ml
Spark
SQL
Spark
Streaming
Spark
ML
MR Codes
...
...
Pig
Hive
Spark API
(pyspark)
HDFS CLI
hadoop fs..
Generated
MR Codes
...
Spark Codes
...
...
Cluster Manager
(YARN/Mesos/Standalone /...)
e.g. YARN/ Standalone
M
S
S
S
Executor
Executor
Executor
Distributed Computing Program
like MapReduce, Spark, Tez, ... can run on
allocated resource by Resource Manager;
e.g. Spark Program or MR program
e.g. HDFS / S3
Name
Node
Data
Node
Data
Node
Data
Node
Distributed File System
(HDFS/S3/GCS/...)

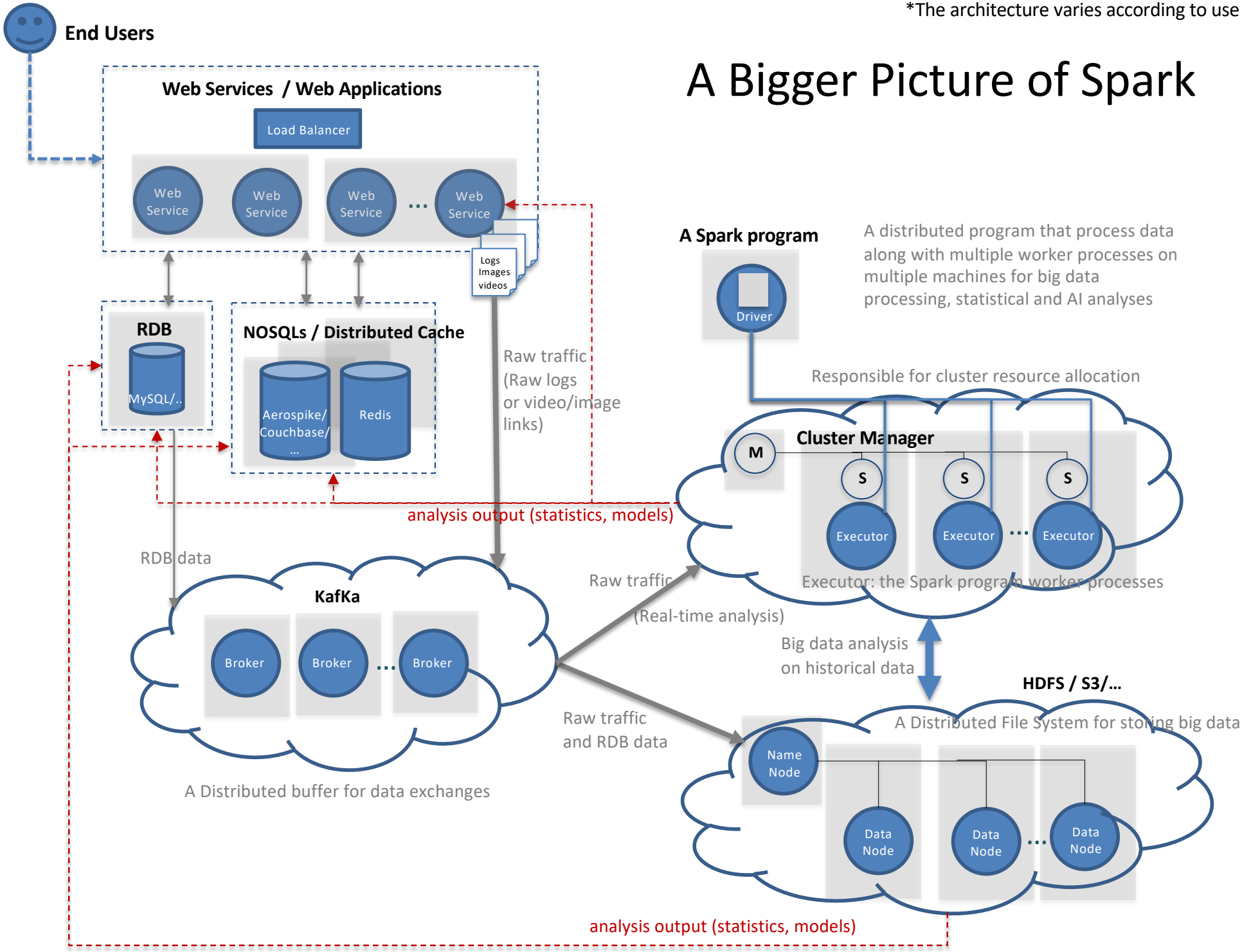

Hadoop/Spark Eco-system Environment
(Key Components)
Logs = LOAD ...
...
...
SELECT *
FROM logs
...
pyspark
sql
pyspark.
streaming
pyspark.ml
MR API
Pig CLI
Hive CLI
Spark
SQL
Spark
Streaming
...
Spark
ML
MR Codes
...
...
Pig
...
Hive
Spark API
(pyspark)
HDFS CLI
Spark Codes
...
...
hadoop fs..
Generated
MR Codes
...
...
Cluster Manager
(YARN/Mesos/Standalone /...)
e.g. YARN/ Standalone
M
S
S
S
Executor
Executor
...
Executor
Distributed Computing Program
like MapReduce, Spark, Tez, ... can run on
allocated resource by Resource Manager;
e.g. Spark Program or MR program
e.g. HDFS / S3
Name
Node
Data
Node
Data
Node
...
Data
Node
Distributed File System
(HDFS/S3/GCS/...)
Development Environment
VM / Container / Physical



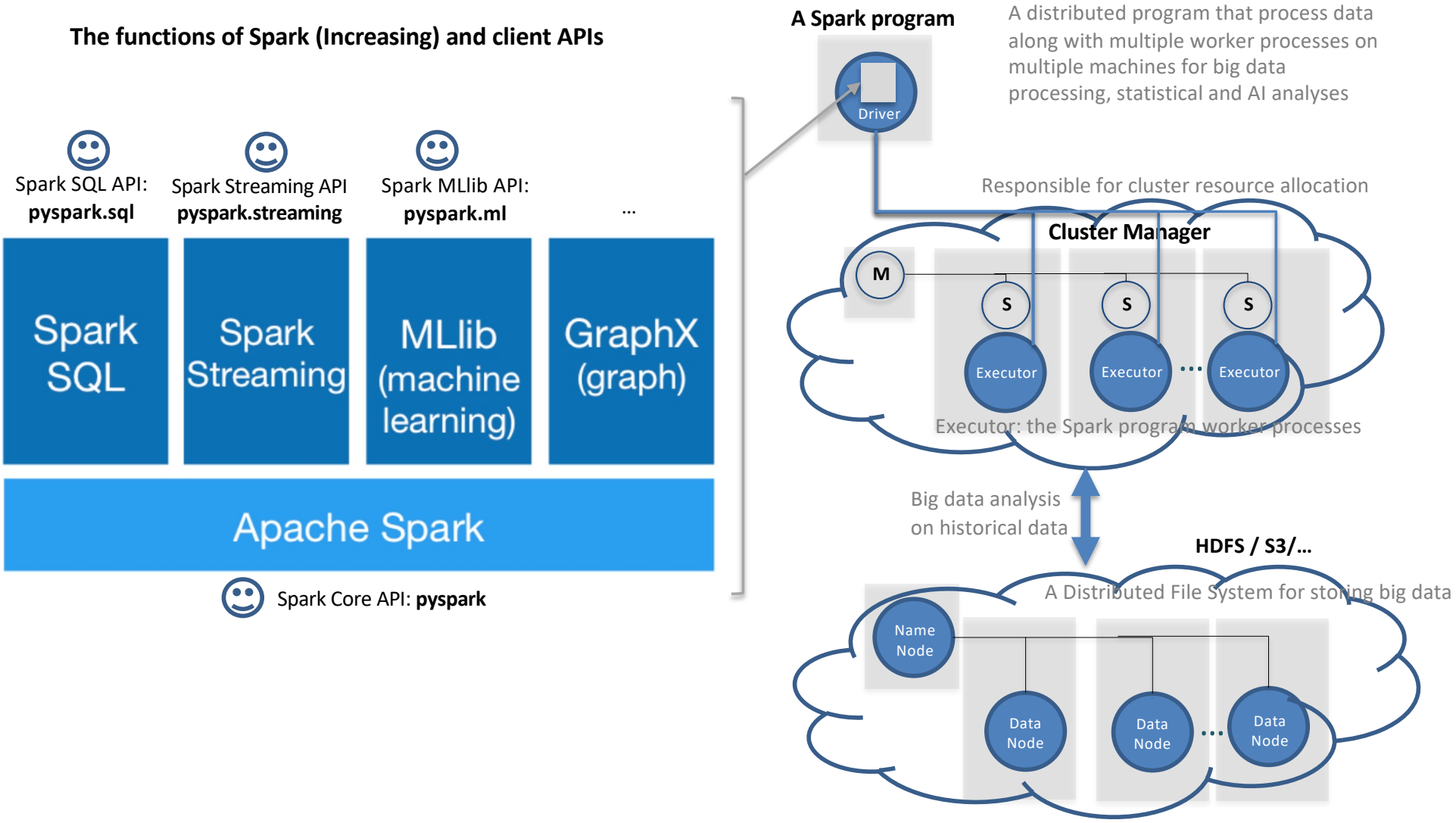
Production Environment

*The architecture varies according to use cases

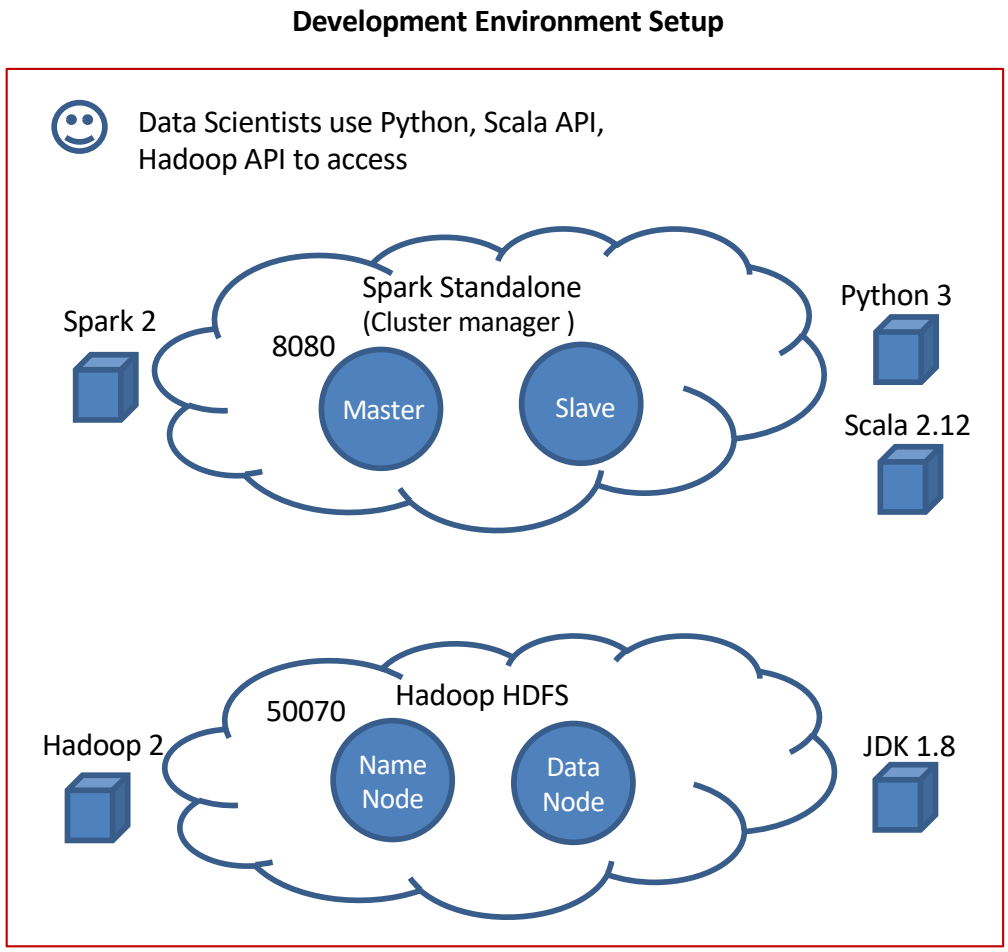
A Bigger Picture of Spark



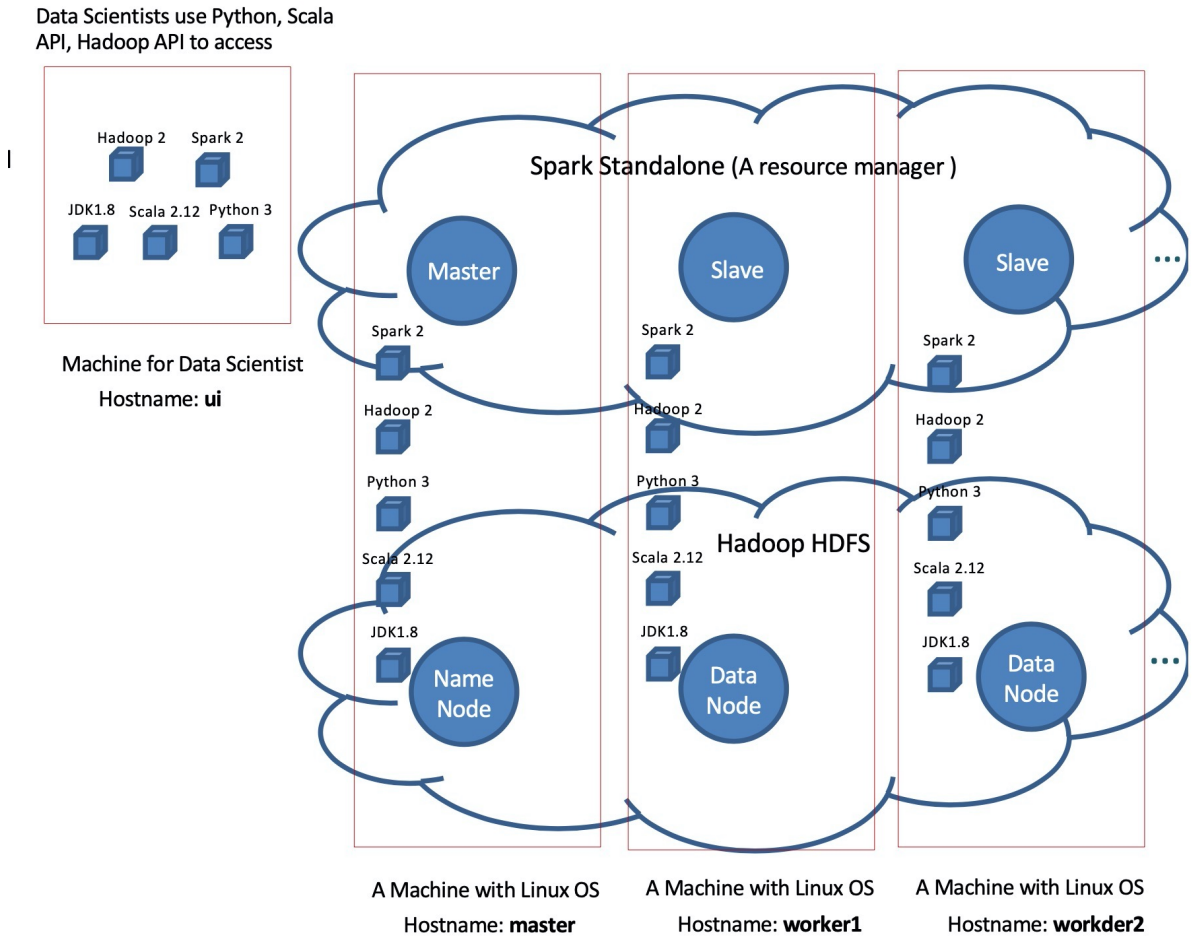
The Spark Eco-System Libraries



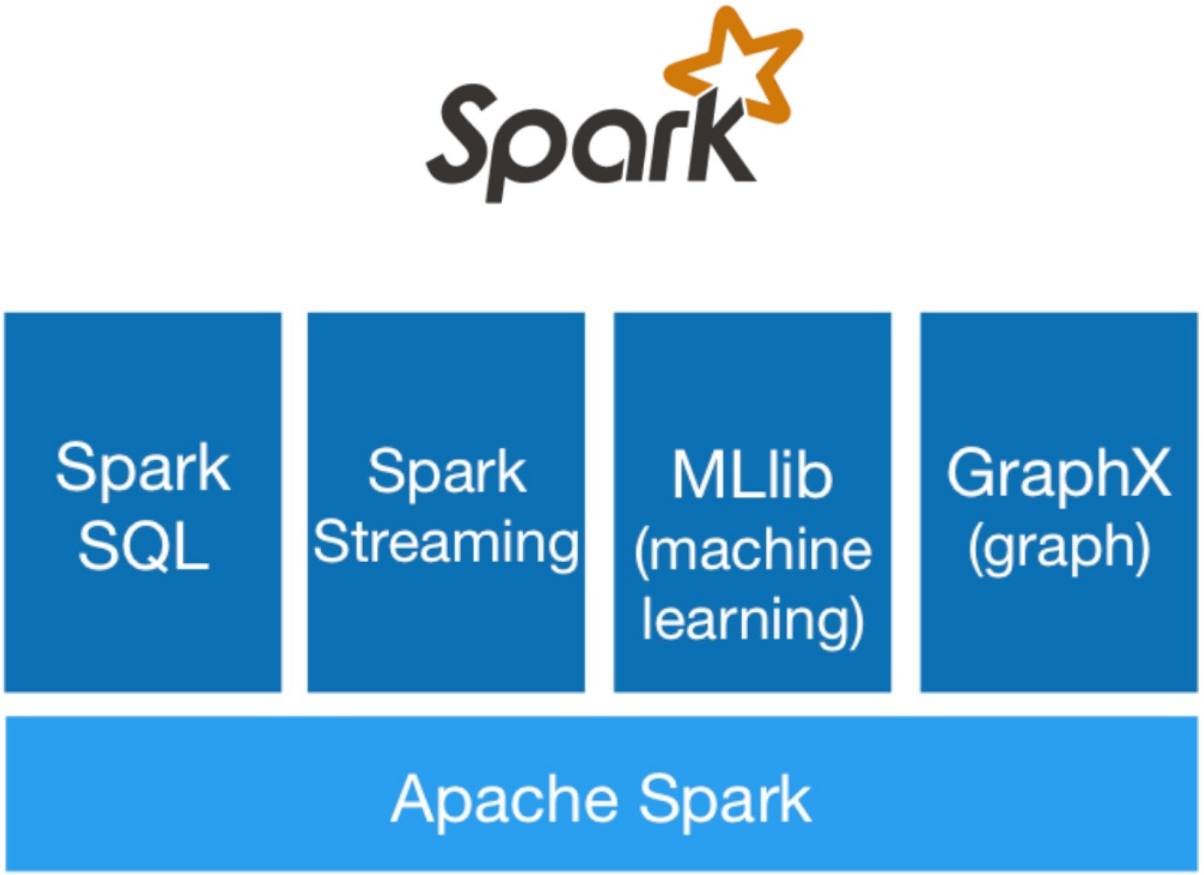
Demo:
Development Environment Setup



Demo (later on)
Production Environment Setup

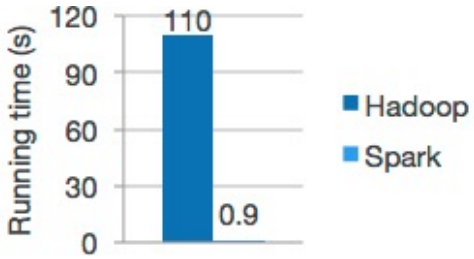


Apache Spark™ is a fast and general engine for large-scale data processing.



Speed

Run programs up to 100x faster than Hadoop MapReduce in memory, or 10x faster on disk.



Ease of Use

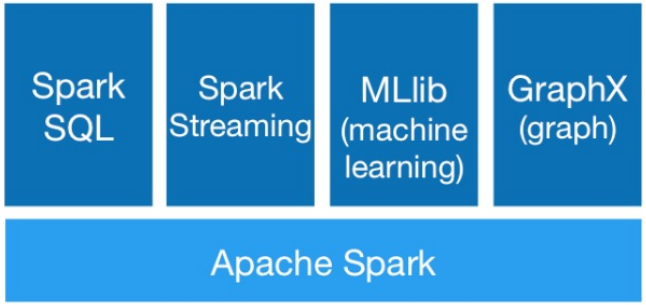
Write applications quickly in Java, Scala, Python, R.

```
text_file = spark.textFile("hdfs://...")

text_file.flatMap(lambda line: line.split())
    .map(lambda word: (word, 1))
    .reduceByKey(lambda a, b: a+b)
```

Word count in Spark's Python API

Generality

Combine SQL, streaming, and complex analytics.



Runs Everywhere

Spark runs on Hadoop, Mesos, standalone, or in the cloud. It can access diverse data sources including HDFS, Cassandra, HBase, and S3.

History

- Spark was initially started by Matei Zaharia at UC Berkeley AMPLab in 2009, and open sourced in 2010 under a BSD license.
- In 2013, the project was donated to the Apache Software Foundation and switched its license to Apache 2.0. In February 2014, Spark became a Top-Level Apache Project.[7]
- In November 2014, the engineering team at Databricks used Spark and set a new world record in large scale sorting.[8]

https://en.wikipedia.org/wiki/Apache_Spark

Hive and Pig on Spark

- Hive on Spark
  - https://cwiki.apache.org/confluence/display/Hive/Hive+on+Spark%3A+Getting+Started
- Pig on Spark
  - https://cwiki.apache.org/confluence/display/PIG/Pig+on+Spark

Get average temperature from BIG weather data

200701hourly.txt

```
Wban Number, YearMonthDay, Time, Station Type, Maintenance Indicator, Sky Conditions, Visibility, Weather Type, Dry Bulb Temp
03011,20070101,0050,AO2 ,-,CLR                                      ,10SM   ,-,30,1,23,29  , 5   ,150,-,0  ,30.14,-,-,AA,
03011,20070101,0150,AO2 ,-,CLR                                      ,10SM   ,-,28,1,21,31  , 6   ,160,-,0  ,30.13,-,-,AA,
03011,20070101,0250,AO2 ,-,SCT120                                   ,10SM   ,-,30,1,23,29  , 4   ,150,-,0  ,30.13,-,-,AA,
03011,20070101,0350,AO2 ,-,CLR                                      ,10SM   ,-,30,1,23,29  , 3   ,140,-,0  ,30.12,-,-,AA,
03011,20070101,0450,AO2 ,-,CLR                                      ,10SM   ,-,30,5,23,34  , 5   ,150,-,0  ,30.10,-,-,AA,
03011,20070101,0550,AO2 ,-,CLR                                      ,10SM   ,-,32,1,25,26  , 4   ,130,-,0  ,30.10,-,-,AA,
03011,20070101,0650,AO2 ,-,CLR                                      ,10SM   ,-,32,1,25,26  , 0   ,000,-,0  ,30.11,-,-,AA,
03011,20070101,0750,AO2 ,-,CLR                                      ,10SM   ,-,28,5,21,37  , 0   ,000,-,0  ,30.13,-,-,AA,
03011,20070101,0850,AO2 ,-,CLR                                      ,10SM   ,-,28,5,21,37  , 0   ,000,-,0  ,30.14,-,-,AA,
03011,20070101,0950,AO2 ,-,CLR                                      ,10SM   ,-,32,3,25,29  , 0   ,000,-,0  ,30.14,-,-,AA,
03011,20070101,1050,AO2 ,-,CLR                                      ,10SM   ,-,32,3,25,29  , 4   ,290,-,0  ,30.14,-,-,AA,
03011,20070101,1150,AO2 ,-,CLR                                      ,10SM   ,-,36,1,27,22  , 4   ,270,-,0  ,30.11,-,-,AA,
03011,20070101,1250,AO2 ,-,CLR                                      ,10SM   ,-,37,1,27,22  , 5   ,270,-,0  ,30.09,-,-,AA,
03011,20070101,1350,AO2 ,-,CLR                                      ,10SM   ,-,37,1,27,22  , 5   ,280,-,0  ,30.09,-,-,AA,
03011,20070101,1450,AO2 ,-,CLR                                      ,10SM   ,-,36,5,27,27  , 6   ,270,-,0  ,30.10,-,-,AA,
03011,20070101,1550,AO2 ,-,CLR                                      ,10SM   ,-,34,5,27,29  , 6   ,280,-,0  ,30.12,-,-,AA,
03011,20070101,1630,AO2 ,-,CLR                                      ,10SM   ,-,34,7,27,32  , 5   ,280,-,0  ,30.13,-,-,AA,
03011,20070101,1750,AO2 ,-,CLR                                      ,10SM   ,-,32,7,25,35  , 0   ,000,-,0  ,30.15,-,-,AA,
03011,20070101,1850,AO2 ,-,CLR                                      ,10SM   ,-,32,9,25,38  , 3   ,140,-,0  ,30.17,-,-,AA,
03011,20070101,1950,AO2 ,-,CLR                                      ,10SM   ,-,28,10,23,47  , 4   ,120,-,0  ,30.18,-,-,AA
03011,20070101,2050,AO2 ,-,CLR                                      ,10SM   ,-,28,9,23,45  , 0   ,000,-,0  ,30.19,-,-,AA,
03011,20070101,2150,AO2 ,-,CLR                                      ,10SM   ,-,27,9,21,47  , 5   ,150,-,0  ,30.19,-,-,AA,
03011,20070101,2250,AO2 ,-,CLR                                      ,10SM   ,-,27,9,21,47  , 0   ,000,-,0  ,30.21,-,-,AA,
03011,20070101,2350,AO2 ,-,CLR                                      ,10SM   ,-,27,7,21,43  , 5   ,140,-,0  ,30.21,-,-,AA,
03011,20070102,0050,AO2 ,-,CLR                                      ,10SM   ,-,23,9,19,55  , 3   ,100,-,0  ,30.21,-,-,AA,
03011,20070102,0150,AO2 ,-,CLR                                      ,10SM   ,-,23,7,19,50  , 5   ,140,-,0  ,30.23,-,-,AA,
```

In MapReduce Codes

The driver (main)

```java
package iii.mr101;

import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.*;
import org.apache.hadoop.mapred.*;


public class AvgTemp {

    public static void main(String[] args) throws Exception {

        JobConf conf = new JobConf(AvgTemp.class);
        conf.setJobName("Avg Temp");

        conf.setInputFormat(SequenceFileInputFormat.class);
        conf.setOutputKeyClass(Text.class);
        conf.setOutputValueClass(IntWritable.class);

        conf.setMapperClass(AvgTempMapper.class);
        conf.setReducerClass(AvgTempReducer.class);

        FileInputFormat.addInputPath(conf, new Path(args[0]));
        FileOutputFormat.setOutputPath(conf, new Path(args[1]));

        JobClient.runJob(conf);
    }
}
```

mapper

```java
package iii.mr101;

import java.io.*;

import org.apache.commons.lang.StringUtils;
import org.apache.hadoop.io.*;
import org.apache.hadoop.mapred.*;

public class AvgTempMapper extends MapReduceBase
implements Mapper<LongWritable, Text, Text, IntWritable>  {

    public void map(LongWritable key, Text value,
            OutputCollector<Text, IntWritable> output, Reporter reporter)
    throws IOException {

        String[] line = value.toString().split(",");

        String dataPart = line[1];
        String temp = line[10];

        if (StringUtils.isNumeric(temp) && !temp.equals("") ){
            output.collect(new Text(dataPart), new IntWritable(Integer.parseInt(temp)));
        }
    }
}
```

reducer

```java
package iii.mr101;

import java.io.*;
import java.util.*;

import org.apache.hadoop.io.*;
import org.apache.hadoop.mapred.*;

public  class AvgTempReducer extends MapReduceBase
    implements Reducer<Text, IntWritable, Text, IntWritable> {

    public void reduce(Text key, Iterator<IntWritable> values,
        OutputCollector<Text, IntWritable> output, Reporter reporter)
    throws IOException {

        int sumTemps = 0;
        int numItems = 0;

        while (values.hasNext()) {
            sumTemps += values.next().get();
            numItems +=1;
        }
        output.collect(key,  new IntWritable(sumTemps / numItems));
    }
}
```

And Build, package, deploy,…

In Spark Codes

```
from pyspark import SparkConf, SparkContext

def is_good(record):
    try:
        temp = int(record.split(",")[10])
    except ValueError:
        return False
    return True


if __name__ == "__main__":
    sc = SparkContext()

    records = sc.textFile("hdfs://localhost/user/cloudera/spark101/avg_temperature/weather")

    good_records = records.filter(is_good)

    day_temp = good_records.map(lambda x: (x.split(",")[1], int(x.split(",")[10])))

    result = day_temp.combineByKey(lambda v: (v, 1), lambda acc, v: (acc[0] + v, acc[1] + 1),
                                   lambda acc1, acc2: (acc1[0] + acc2[0], acc1[1] + acc2[1])) \
        .map(lambda x: (x[0], x[1][0] / x[1][1]))

    for line in result.collect():
        print(line)
```

In Hive or Spark SQL

Hive or Spark SQL*

```
SELECT year, AVG(temperature)
FROM records
GROUP BY year;
```

Spark API for Python, Scala and Java at a glance

Python

```python
lines = sc.textFile("data.txt")
lineLengths = lines.map(lambda s: len(s))
totalLength = lineLengths.reduce(lambda a, b: a + b)
```

Scala

```scala
val lines = sc.textFile("data.txt")
val lineLengths = lines.map(s => s.length)
val totalLength = lineLengths.reduce((a, b) => a + b)
```

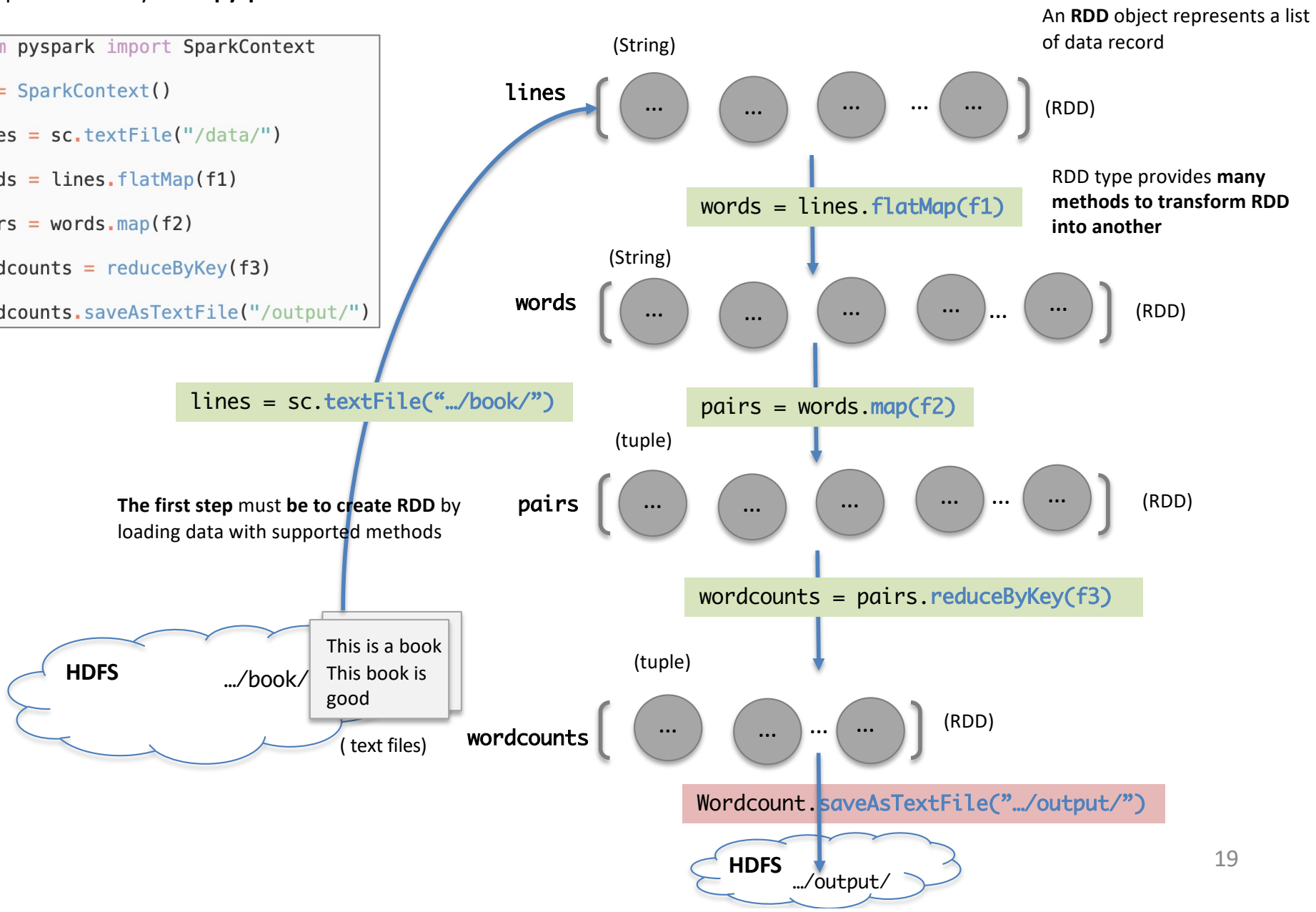
Java

```java
JavaRDD<String> lines = sc.textFile("data.txt");
JavaRDD<Integer> lineLengths = lines.map(s -> s.length());
int totalLength = lineLengths.reduce((a, b) -> a + b);
```

Spark Core API Programming Model

Spark API for Python: **pyspark**

```
1  from pyspark import SparkContext
2
3  sc = SparkContext()
4
5  lines = sc.textFile("/data/")
6
7  words = lines.flatMap(f1)
8
9  pairs = words.map(f2)
10
11 wordcounts = reduceByKey(f3)
12
13 wordcounts.saveAsTextFile("/output/")
```



Demo: Dive into Spark Core API

- Word counting on book *War and peace*

war_and_peace.txt

```
The Project Gutenberg EBook of War and Peace, by Leo Tolstoy

This eBook is for the use of anyone anywhere at no cost and with almost
no restrictions whatsoever.  You may copy it, give it away or re-use it
under the terms of the Project Gutenberg License included with this
eBook or online at www.gutenberg.org


Title: War and Peace

Author: Leo Tolstoy

Translators: Louise and Aylmer Maude

Posting Date: January 10, 2009 [EBook #2600]

Last Updated: March 15, 2013

Language: English

Character set encoding: Latin-1

*** START OF THIS PROJECT GUTENBERG EBOOK WAR AND PEACE ***

An Anonymous Volunteer, and David Widger



WAR AND PEACE

By Leo Tolstoy/Tolstoi

CONTENTS

BOOK ONE: 1805

CHAPTER I

CHAPTER II

CHAPTER III
```

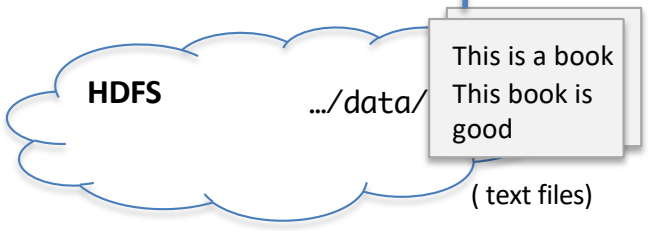
Demo: Dive into Spark Core API

Spark API for Python: **pyspark**

```
sc = SparkContext()
lines = sc.textFile("hdfs://devenv/user/spark/spark101/wordcount/data")
words = lines.flatMap(lambda x: x.split(" "))
paris = words.map(lambda x: (x, 1))
wordcounts = paris.reduceByKey(lambda x, y: x + y)
word_counts.saveAsTextFile("hdfs://devenv/user/spark/spark101/wordcount/output")
```

lines = sc.textFile(“.../data/”)

The first step must **be to create RDD** by loading data with supported methods



lines “This is a book”, “This book is”, “good” (RDD)

words = lines.flatMap(lambda x: x.split(“ “))

words “This”,”is”,”a”,”book”,”This”,”book”,”is”, “good” (RDD)

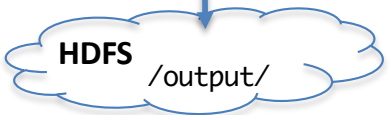
pairs = words.map(lambda x: (x, 1))

pairs (“This”,1),(”is”,1),(”a”,1),(”book”,1),(“This”,1),(”book”,1),(”is”,1),(“good”,1)) (RDD)

wordcounts = pairs.reduceByKey(lambda x, y: x + y)

wordcounts (“This”,2),(”is”,2),(”a”,1),(”book”,2),(“good”,1) (RDD)

wordcounts.saveAsTextFile(”…/output/”)



Demo: Dive into Spark

- wordcount.py

```
sc = SparkContext()

lines = sc.textFile("hdfs://devenv/user/spark/spark101/wordcount/data")

words = lines.flatMap(lambda x: x.split(" "))

word_counts = words.map(lambda x: (x, 1)).reduceByKey(lambda x, y: x + y)

word_counts.saveAsTextFile("hdfs://devenv/user/spark/spark101/wordcount/output")
```

Run a Spark program

- spark-submit

```
spark-submit --master spark://devenv:7077 wordcount.py
```

1. Parse command line arguments
2. Set env. variables
3. python wordcount.py

(PYSPARK_PYTHON)

Run a Spark program interactively

pyspark command (for Python)

```
spark@devenv:~$ pyspark --master spark://devenv:7077
Python 3.7.6 (default, Jan  8 2020, 19:59:22)
Type 'copyright', 'credits' or 'license' for more information
IPython 7.12.0 -- An enhanced Interactive Python. Type '?' for help.
20/09/08 15:49:51 WARN util.Utils: Your hostname, devenv resolves to a loopback
20/09/08 15:49:51 WARN util.Utils: Set SPARK_LOCAL_IP if you need to bind to ano
20/09/08 15:49:52 WARN util.NativeCodeLoader: Unable to load native-hadoop libra
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLeve
Welcome to
      ____              __
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ `/ __/  '_/
   /__ / .__/\_,_/_/ /_/\_\   version 2.4.5
      /_/

Using Python version 3.7.6 (default, Jan  8 2020 19:59:22)
SparkSession available as 'spark'.

In [1]: sc
Out[1]: <SparkContext master=spark://devenv:7077 appName=PySparkShell>
```

spark-shell command (for Scala)

```
spark@devenv:~$ spark-shell --master spark://devenv:7077
20/09/08 15:47:21 WARN util.Utils: Your hostname, devenv resolves to a loopback address:
20/09/08 15:47:21 WARN util.Utils: Set SPARK_LOCAL_IP if you need to bind to another addr
20/09/08 15:47:22 WARN util.NativeCodeLoader: Unable to load native-hadoop library for yo
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLevel(newLeve
Spark context Web UI available at http://192.168.186.133:4040
Spark context available as 'sc' (master = spark://devenv:7077, app id = app-20200908154
Spark session available as 'spark'.
Welcome to
```

Note:

- Environment variables related to which python and interactive python shell to use:

```
export PYSPARK_PYTHON=python

# use ipython as the interactive shell
export PYSPARK_DRIVER_PYTHON=ipython
```

```
export PYSPARK_PYTHON=python

# use jupyter as the interactive shell
export PYSPARK_DRIVER_PYTHON=jupyter
export PYSPARK_DRIVER_PYTHON_OPTS=notebook
```

- RDD's methods **take(N)** and **collect**() are often used in the interactive shell to show the RDD elelments or part of them as we transform them
  - Verfiy each step
  - Test logics
  - etc.
- The details of take(N) and collect() will be introudce later

More arguments for Spark-submit, spark-shell and pyspark

```
Options:
  --master MASTER_URL         spark://host:port, mesos://host:port, yarn, or local.
  --deploy-mode DEPLOY_MODE   Whether to launch the driver program locally ("client") or
                              on one of the worker machines inside the cluster ("cluster")
                              (Default: client).
  --class CLASS_NAME          Your application's main class (for Java / Scala apps).
  --name NAME                 A name of your application.
  --jars JARS                 Comma-separated list of local jars to include on the driver
                              and executor classpaths.
  --packages                  Comma-separated list of maven coordinates of jars to include
                              on the driver and executor classpaths. Will search the local
                              maven repo, then maven central and any additional remote
                              repositories given by --repositories. The format for the
                              coordinates should be groupId:artifactId:version.
  --exclude-packages          Comma-separated list of groupId:artifactId, to exclude while
                              resolving the dependencies provided in --packages to avoid
                              dependency conflicts.
  --repositories              Comma-separated list of additional remote repositories to
                              search for the maven coordinates given with --packages.
  --py-files PY_FILES         Comma-separated list of .zip, .egg, or .py files to place
                              on the PYTHONPATH for Python apps.
  --files FILES               Comma-separated list of files to be placed in the working
                              directory of each executor.

  --conf PROP=VALUE           Arbitrary Spark configuration property.
  --properties-file FILE      Path to a file from which to load extra properties. If not
                              specified, this will look for conf/spark-defaults.conf.
```

SparkContext

- Driver programs access Spark through a SparkContext object, which represents a connection to a computing cluster

RDD

- Resilient Distributed Dataset
  - An immutable distributed collection of objects
  - Can contain any type of Scala, Java, or Python objects, including user-defined classes.
- Create a RDD by
  - loading an external dataset
  - distributing a collection of objects

Create a RDD by loading an external dataset

- Supported datasets
  - Text files
  - Sequence files
  - Parquet files
  - Avro files
  - Any sources that support the Hadoop *InputFormat* class implementation

Create a RDD by distributing a collection of objects

- Parallelized collections are created by calling SparkContext's parallelize method on an existing collection in your driver program (a Scala Seq)
- The elements of the collection are copied to form a distributed dataset that can be operated on in parallel

```
data = [1, 2, 3, 4, 5]
distData = sc.parallelize(data)
```

RDD Methods - Transformation

- Transformations are operations on RDDs that return a new RDD
- All transformations are *lazy* and only computed when an action requires a result

```
sc = SparkContext()

lines = sc.textFile("hdfs://devenv/user/spark/spark101/wordcount/data")

words = lines.flatMap(lambda x: x.split(" "))

paris = words.map(lambda x: (x, 1))

wordcounts = paris.reduceByKey(lambda x, y: x + y)

word_counts.saveAsTextFile("hdfs://devenv/user/spark/spark101/wordcount/output")
```

RDD **Methods** - **Action**

- Actions are operations that return a final value to the driver program or write data to an external storage system
- Actions force the evaluation of the transformations required for the RDD

```
sc = SparkContext()

lines = sc.textFile("hdfs://devenv/user/spark/spark101/wordcount/data")

words = lines.flatMap(lambda x: x.split(" "))

paris = words.map(lambda x: (x, 1))

wordcounts = paris.reduceByKey(lambda x, y: x + y)

word_counts.saveAsTextFile("hdfs://devenv/user/spark/spark101/wordcount/output")
```

.take(20) . collect() .reduce(f)

Example of RDD transformation and action methods

- Transformation methods
  - map(f)
  - flatMap(f)
  - filter(f)
  - distinct()
  - reduceByKey(f)
  - groupByKey(f)
  - mapValues(f)
  - …
- Action methods:
  - saveAsTextFile(…)
  - saveAsHadoopFile(…)
  - reduce(f)
  - collect()
  - take(…)
  - first()
  - foreachPartition(f)
  - foreach(f)
  - …

Spark Program Run-time Excution Flow

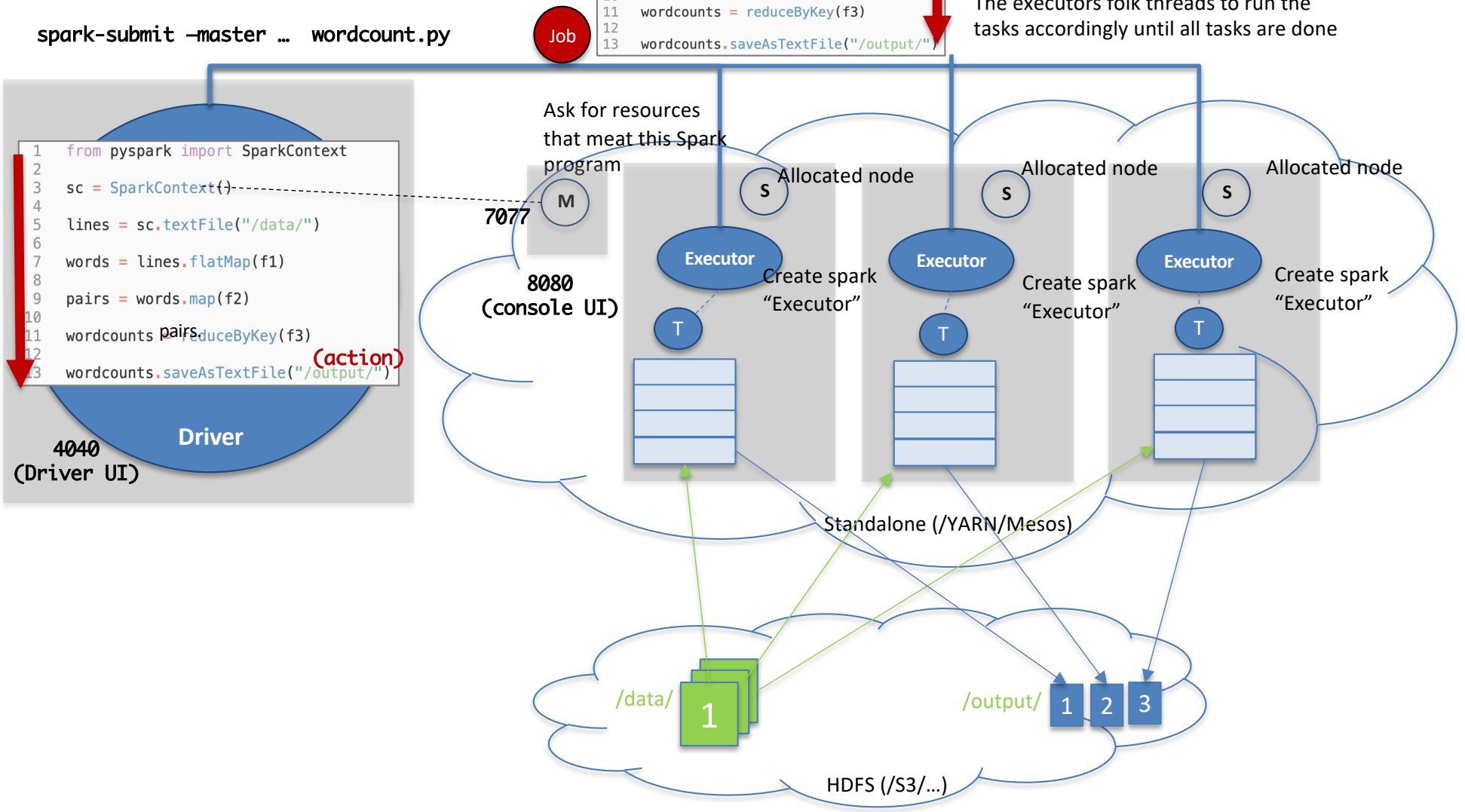
Spark API for Python: **pyspark**

Tasks are internal instructions based on your codes

```
5  lines = sc.textFile("/data/")
6
7  words = lines.flatMap(f1)
8
9  pairs = words.map(f2)
10
11 wordcounts = reduceByKey(f3)
12
13 wordcounts.saveAsTextFile("/output/")
```

RDD's action method trigger real execution by submitting a job with tasks to the executors

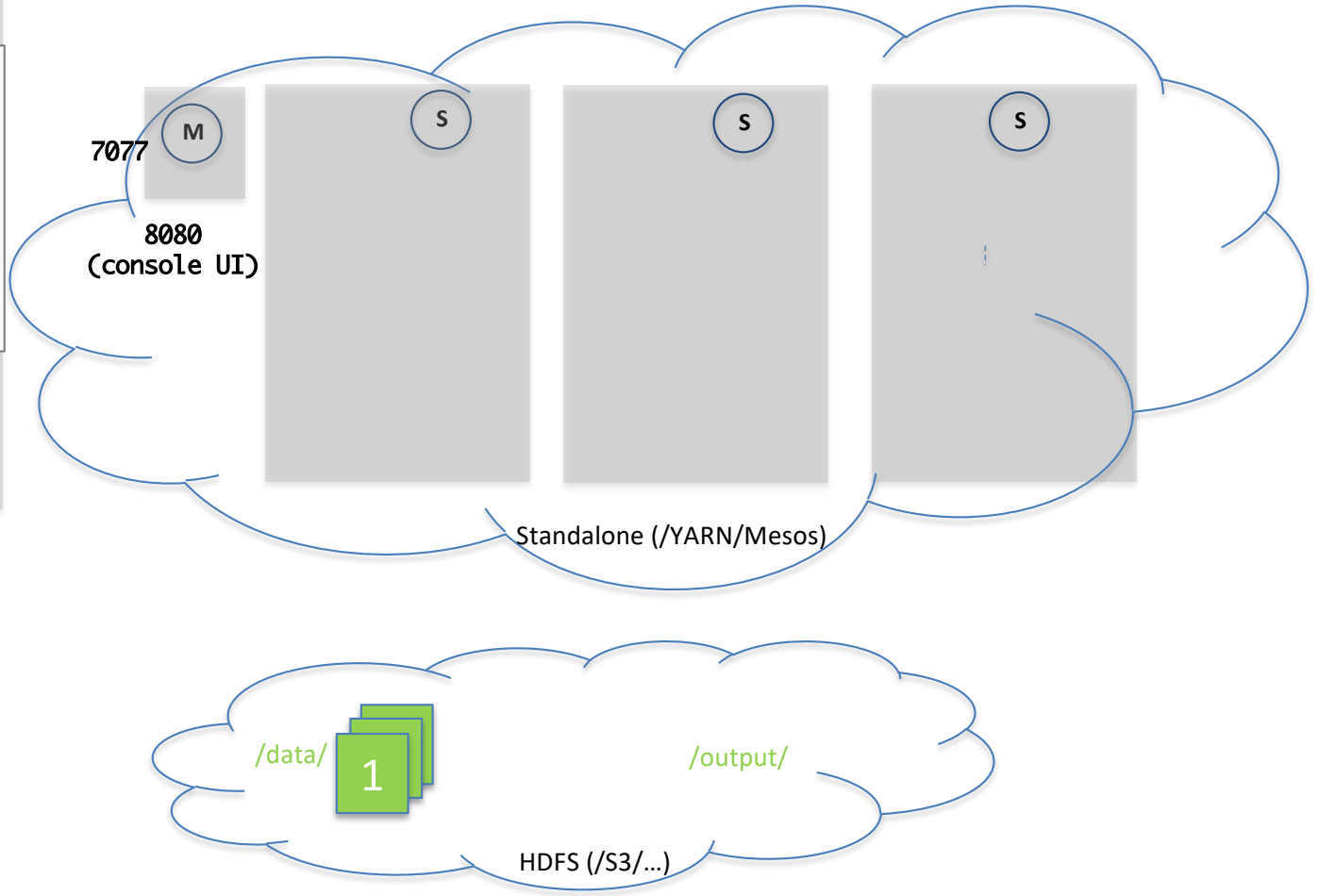
The executors folk threads to run the tasks accordingly until all tasks are done



Spark Program Run-time Excution Flow (step by step)

Spark API for Python: **pyspark**

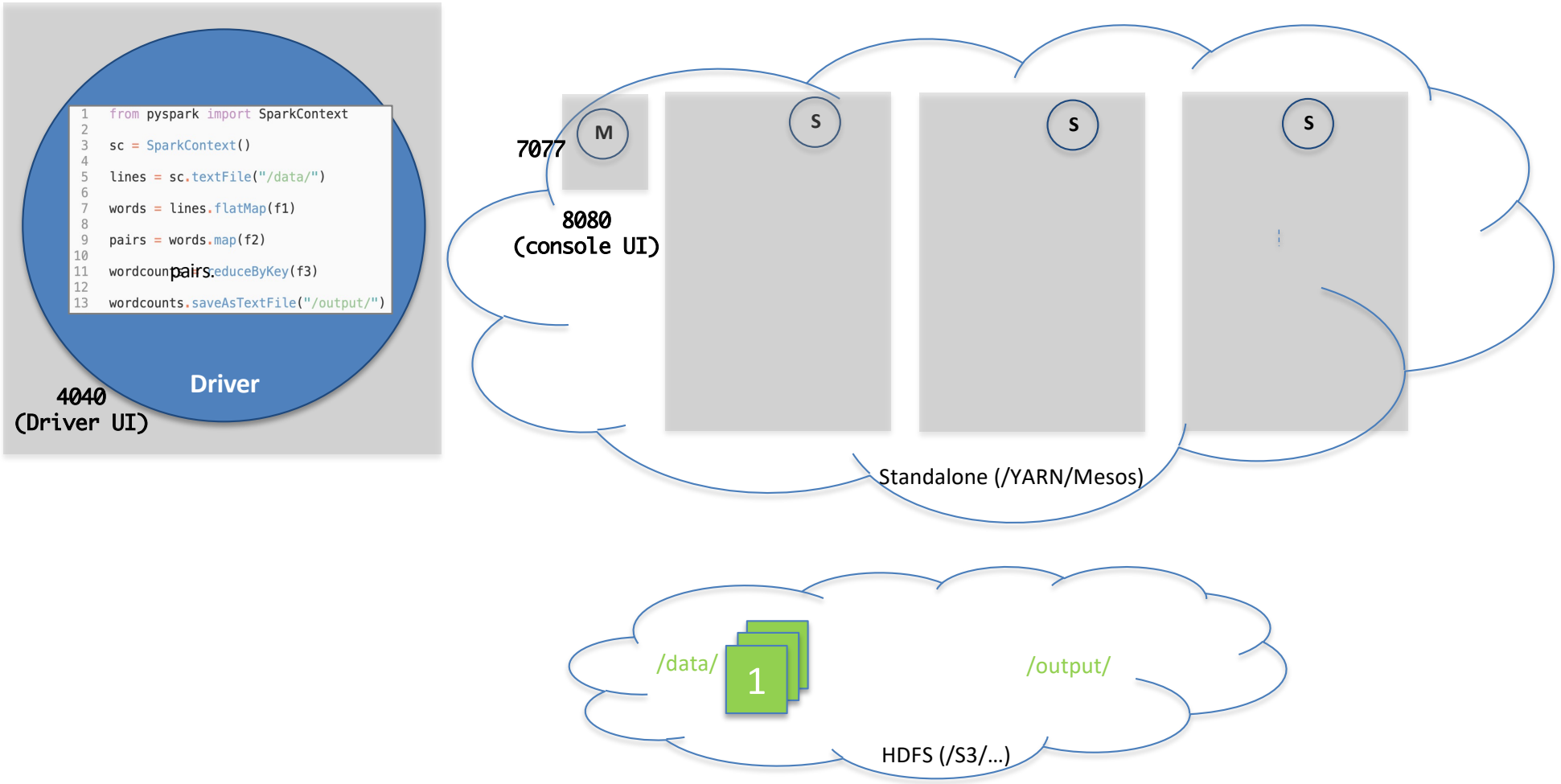
```
1  from pyspark import SparkContext
2
3  sc = SparkContext()
4
5  lines = sc.textFile("/data/")
6
7  words = lines.flatMap(f1)
8
9  pairs = words.map(f2)
10
11 wordcounts = pairs.reduceByKey(f3)
12
13 wordcounts.saveAsTextFile("/output/")
```



Spark Program Run-time Excution Flow (step by step)

Spark API for Python: **pyspark**

```
spark-submit –master …  wordcount.py
```

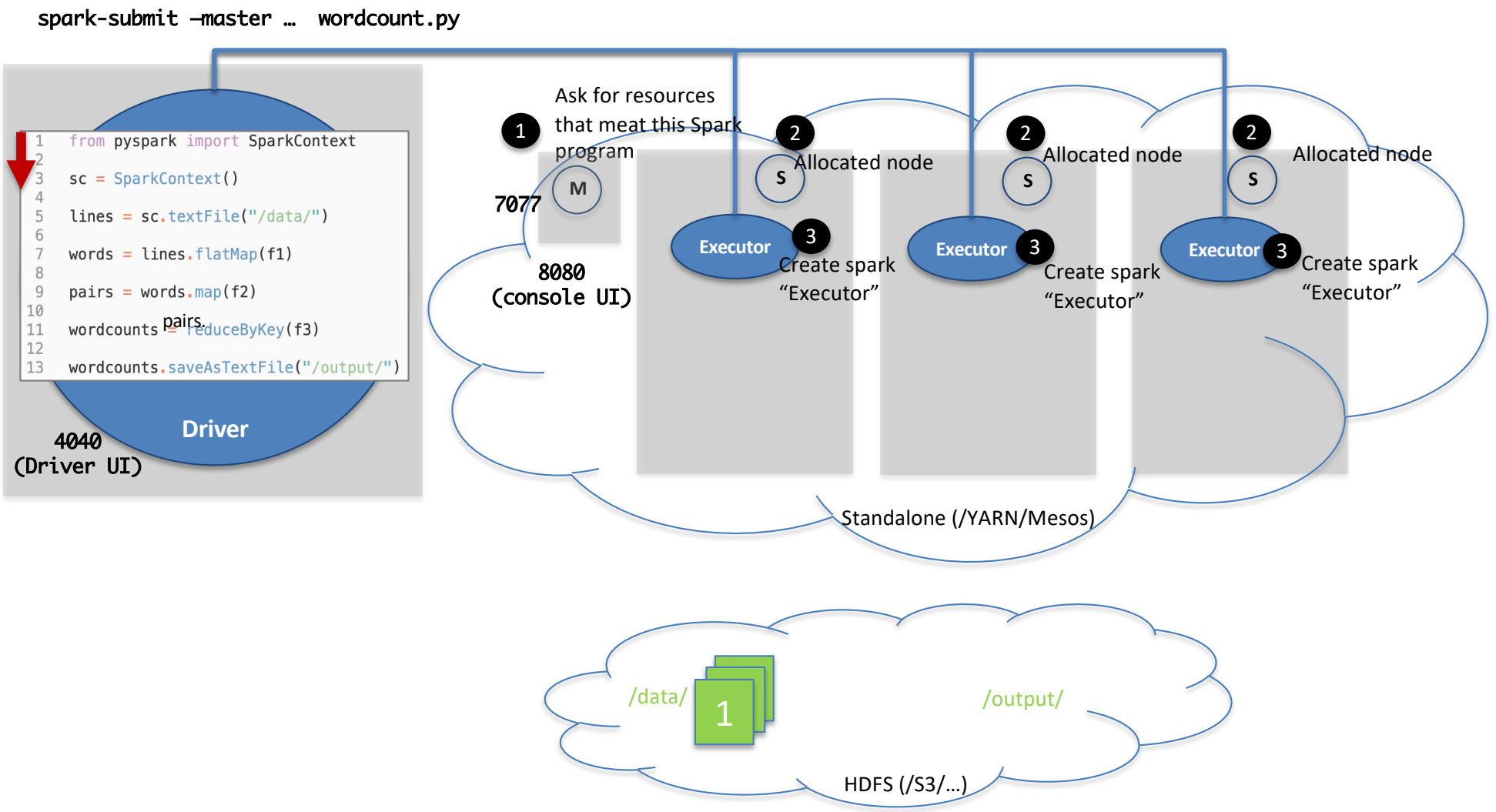


Spark Program Run-time Excution Flow (step by step)

Spark API for Python: **pyspark**

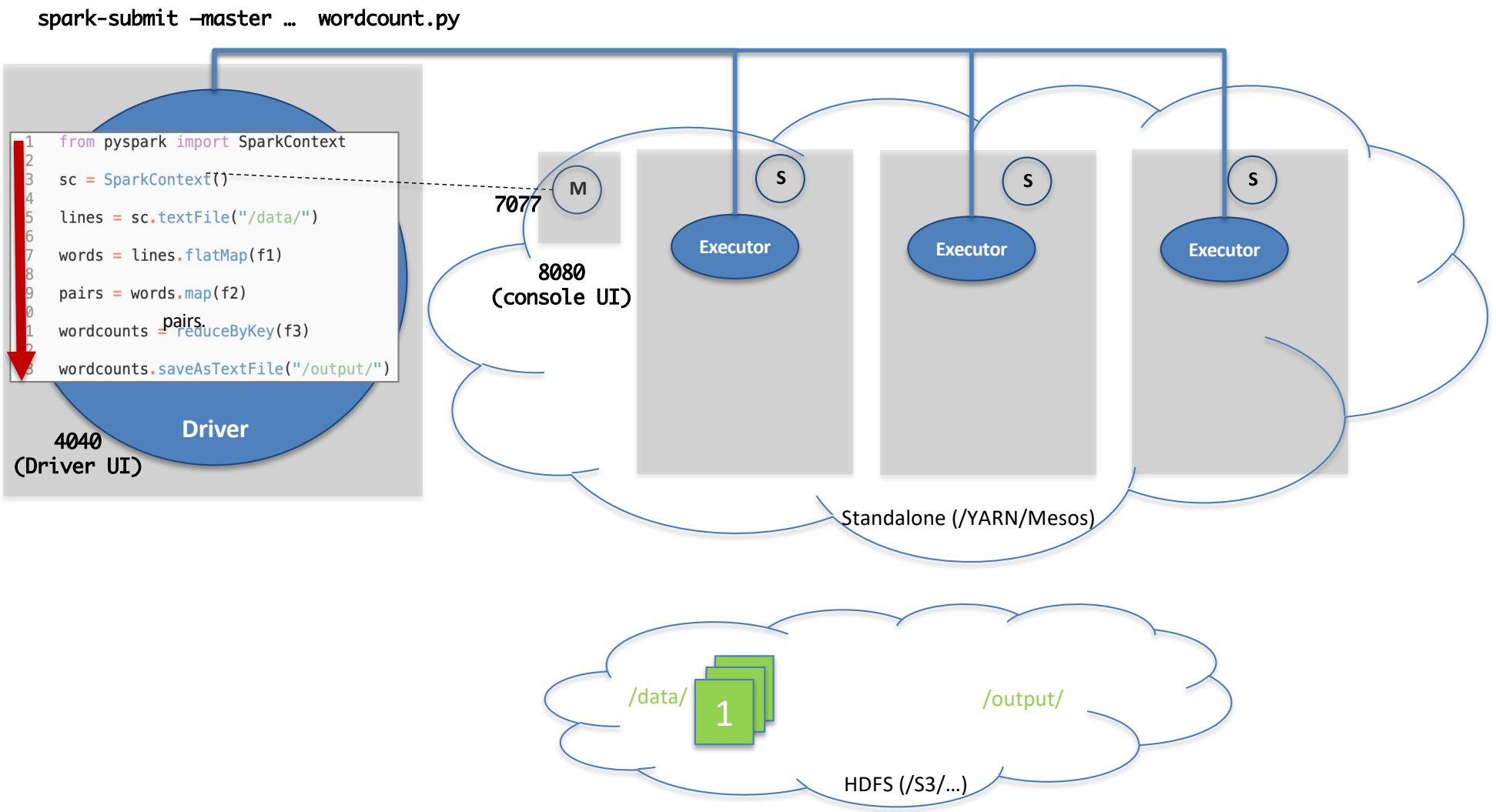
A Spark Program is in fact a set of distributed processes consisting **of the Driver and several Executors (workers)**

It is the executors who process the data. The Driver makes the order.



Spark Program Run-time Excution Flow (step by step)

Spark API for Python: **pyspark**



Spark Program Run-time Excution Flow (step by step)

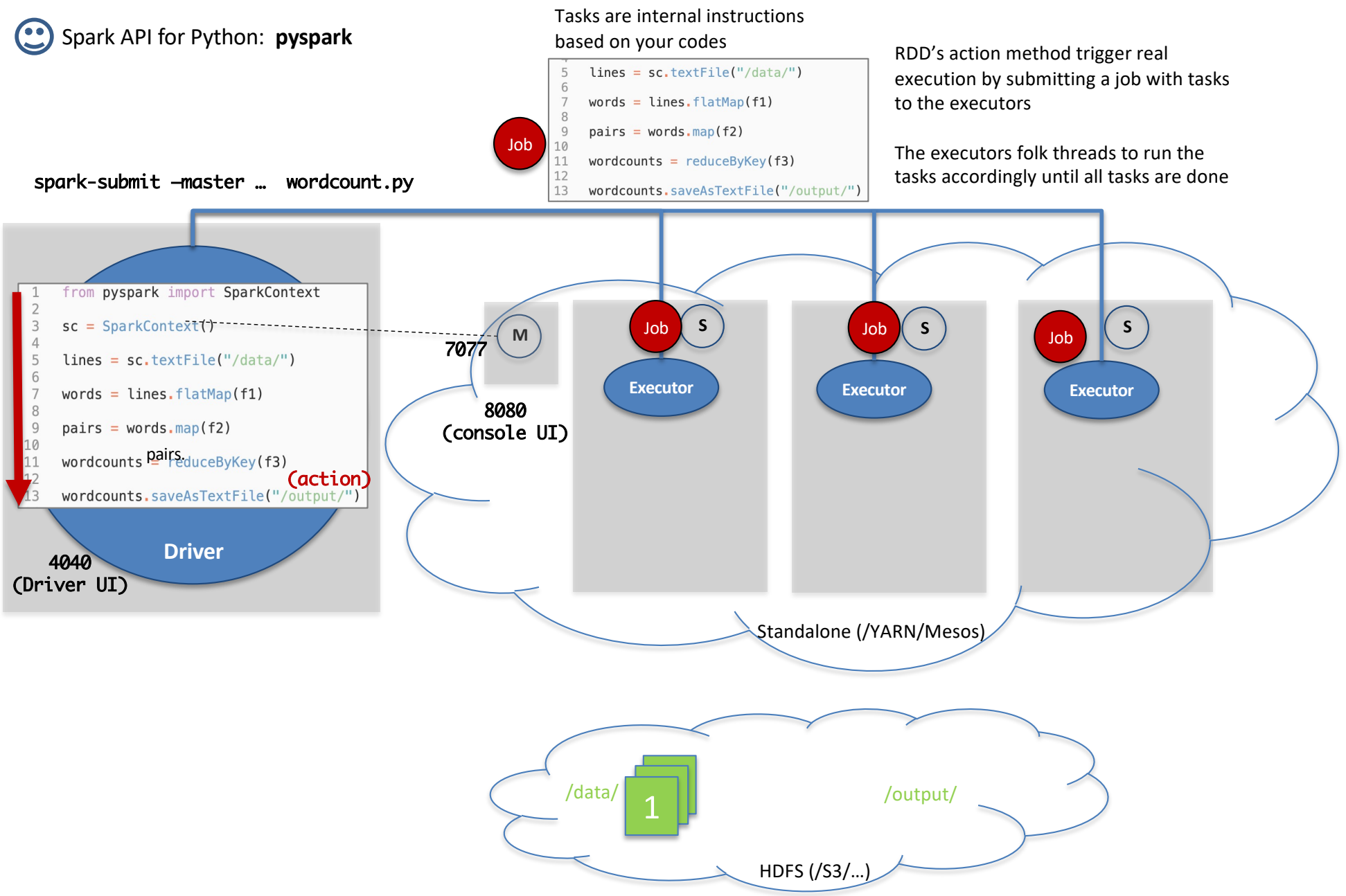
Spark API for Python: **pyspark**

Tasks are internal instructions based on your codes

```
5  lines = sc.textFile("/data/")
6
7  words = lines.flatMap(f1)
8
9  pairs = words.map(f2)
10
11 wordcounts = reduceByKey(f3)
12
13 wordcounts.saveAsTextFile("/output/")
```

RDD's action method trigger real execution by submitting a job with tasks to the executors

The executors folk threads to run the tasks accordingly until all tasks are done



Spark Program Run-time Excution Flow (step by step)

Spark API for Python: **pyspark**

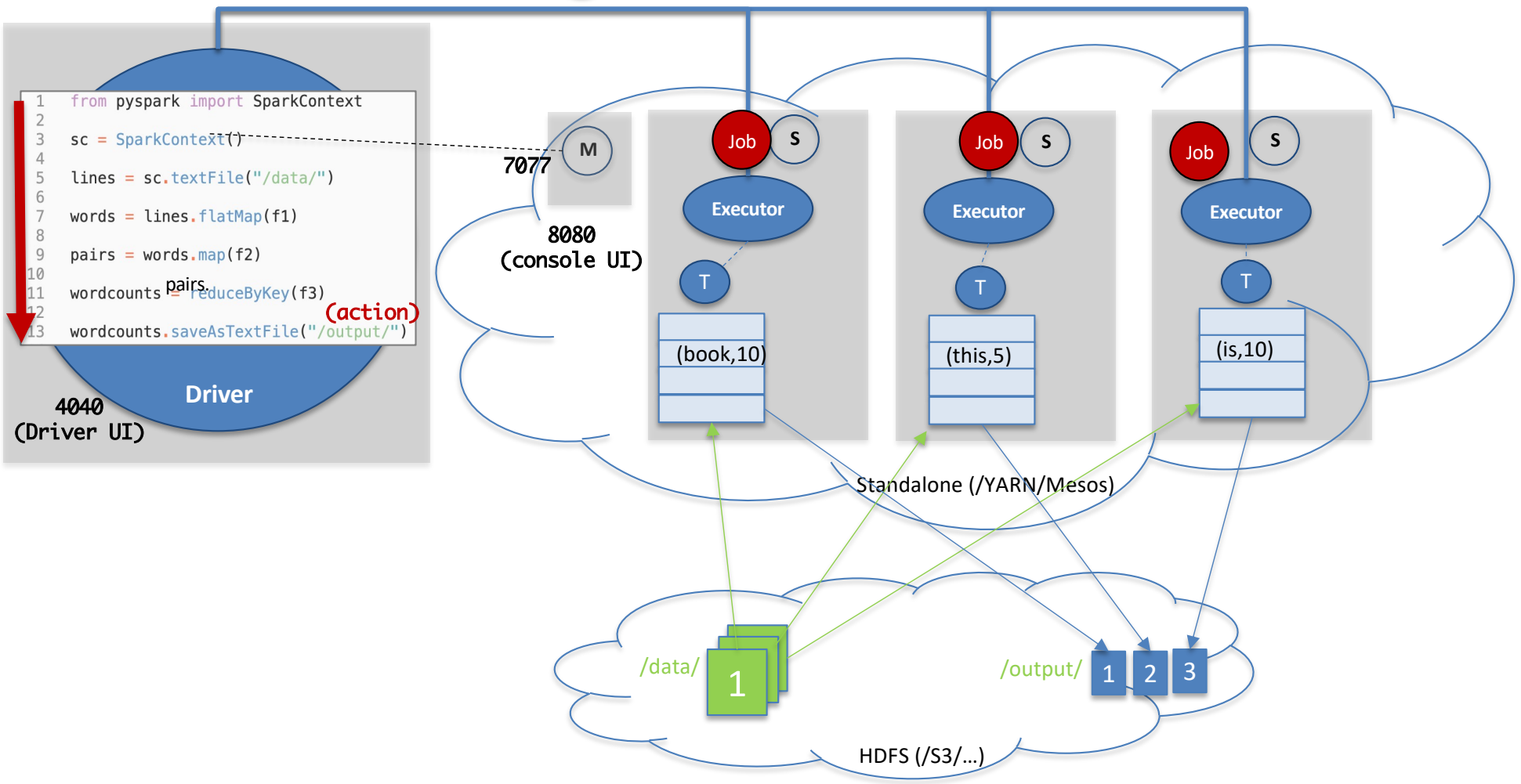
Tasks are internal instructions based on your codes

```
5  lines = sc.textFile("/data/")
6
7  words = lines.flatMap(f1)
8
9  pairs = words.map(f2)
10
11 wordcounts = pairs.reduceByKey(f3)
12
13 wordcounts.saveAsTextFile("/output/")
```

RDD's action method trigger real execution by submitting a job with tasks to the executors

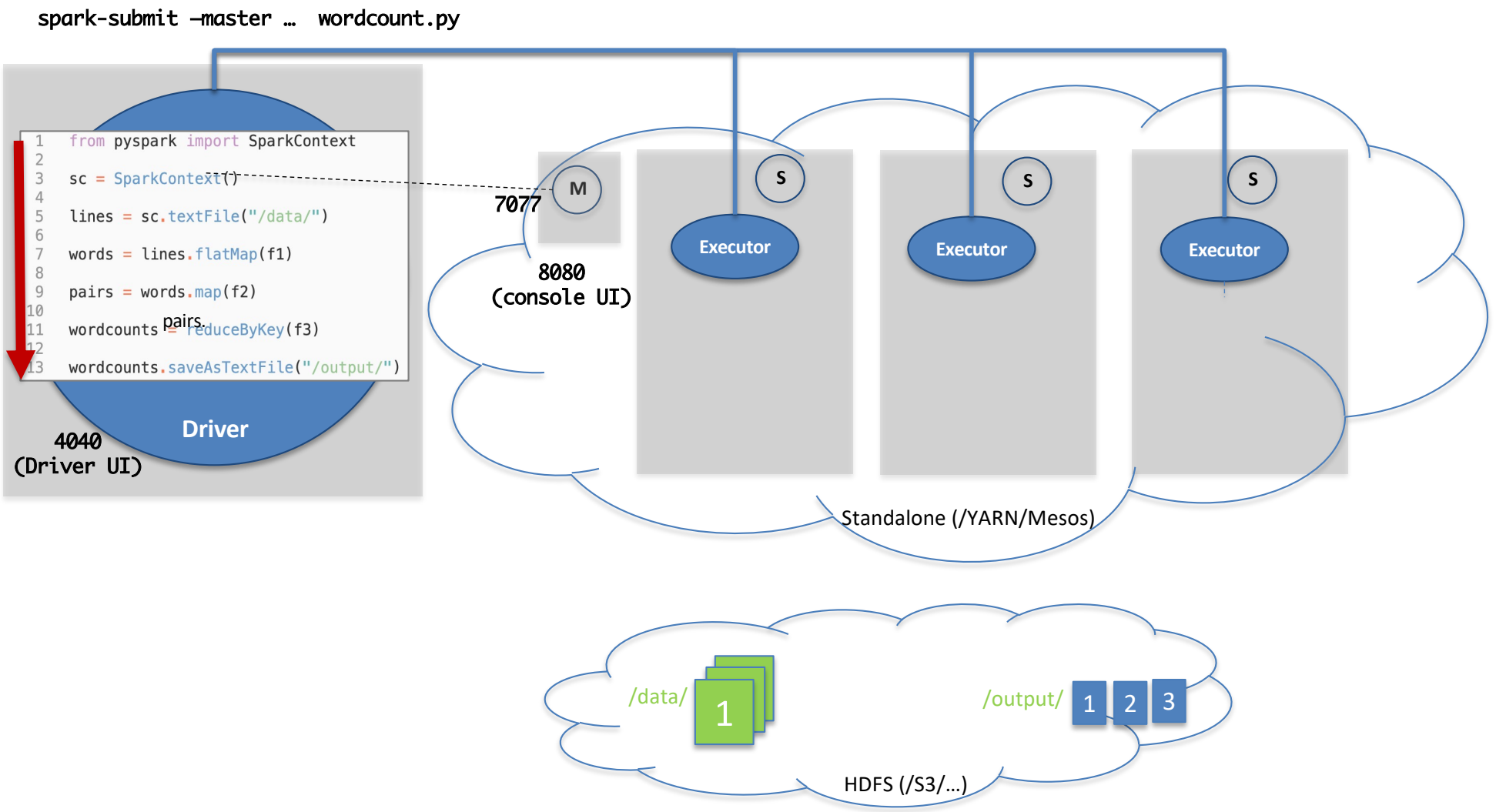
The executors folk threads to run the tasks accordingly until all tasks are done

`spark-submit –master … wordcount.py`



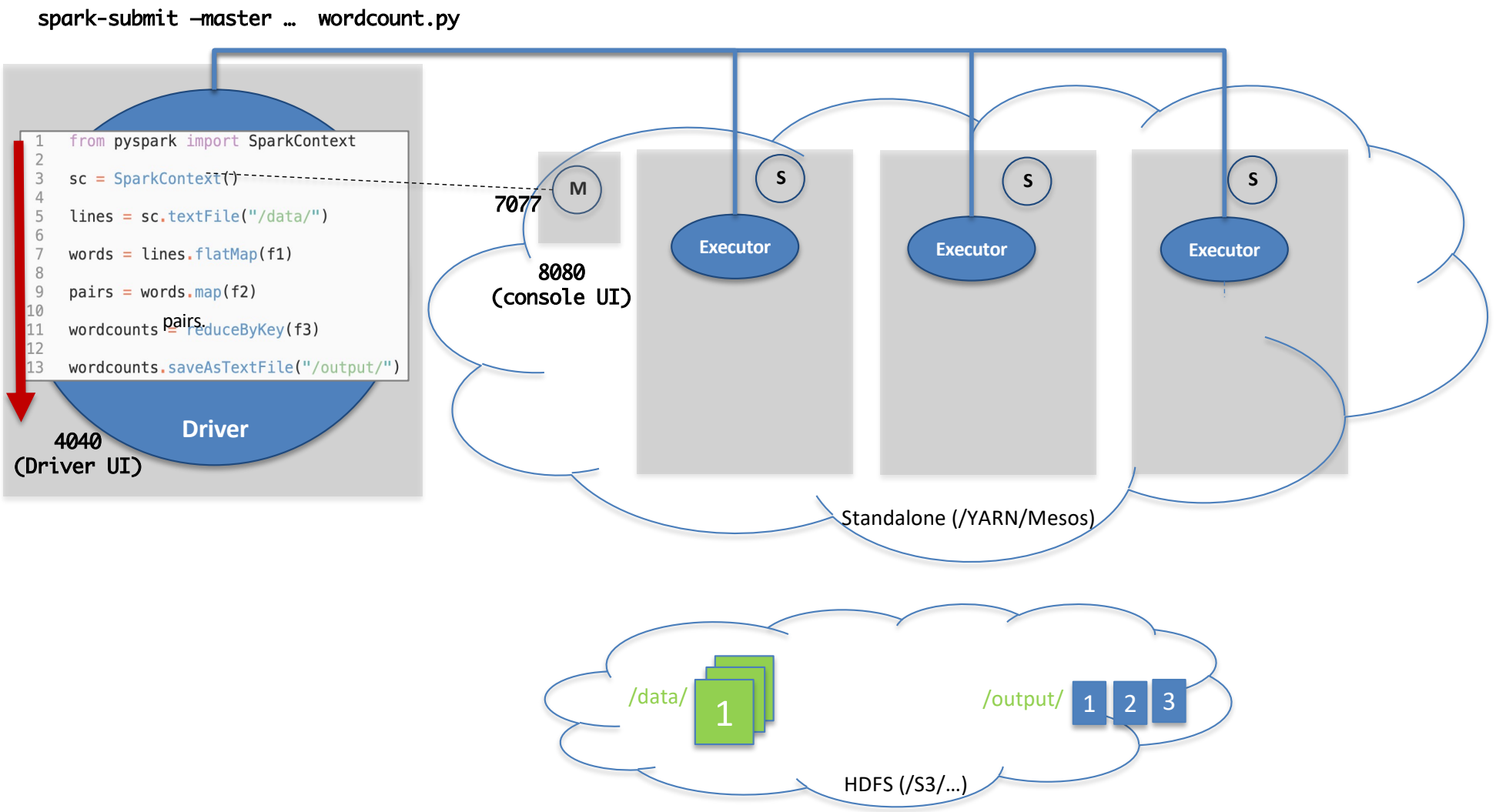
Spark Program Run-time Excution Flow (step by step)

Spark API for Python: **pyspark**



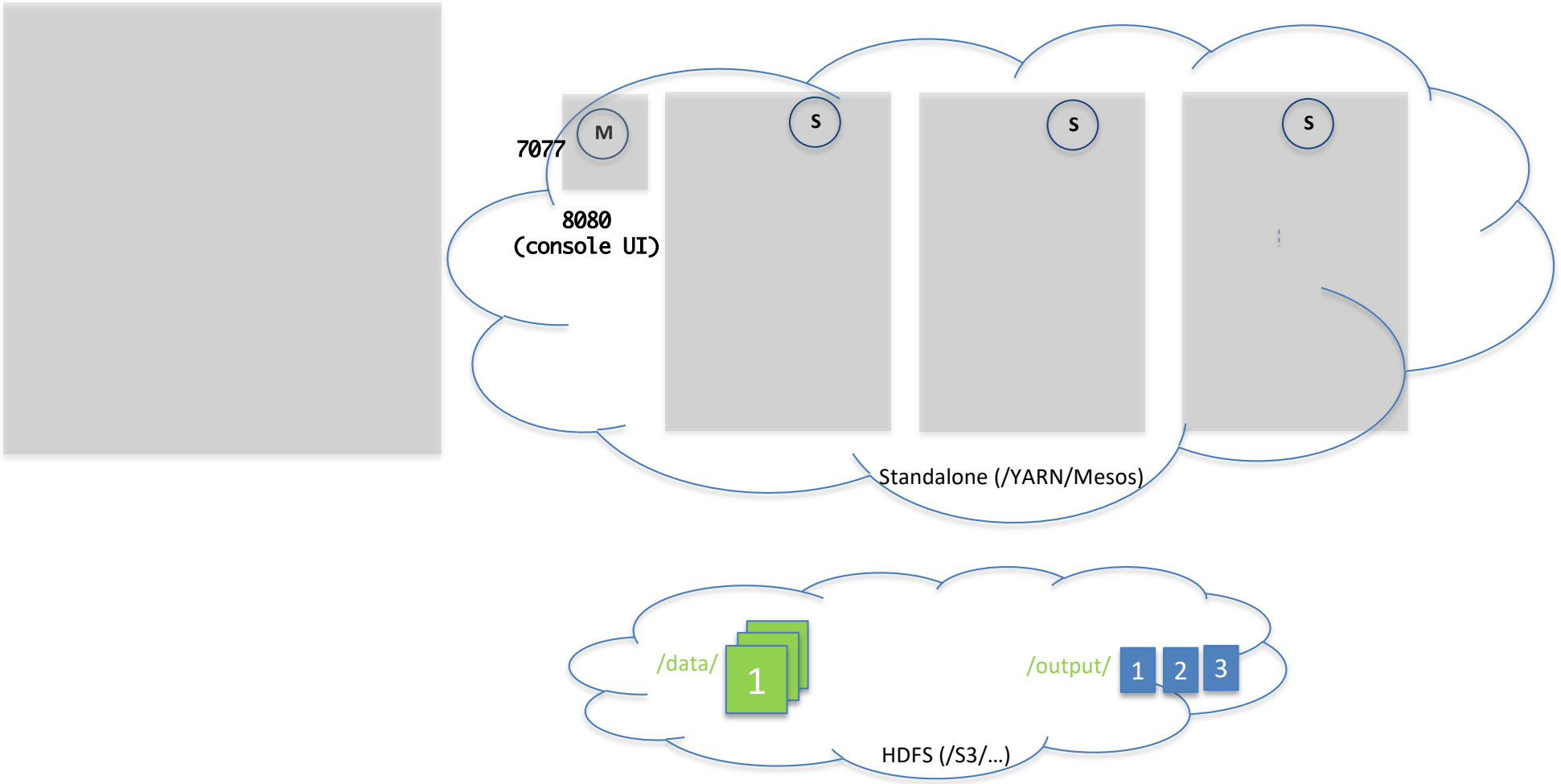
Spark Program Run-time Excution Flow

Spark API for Python: **pyspark**

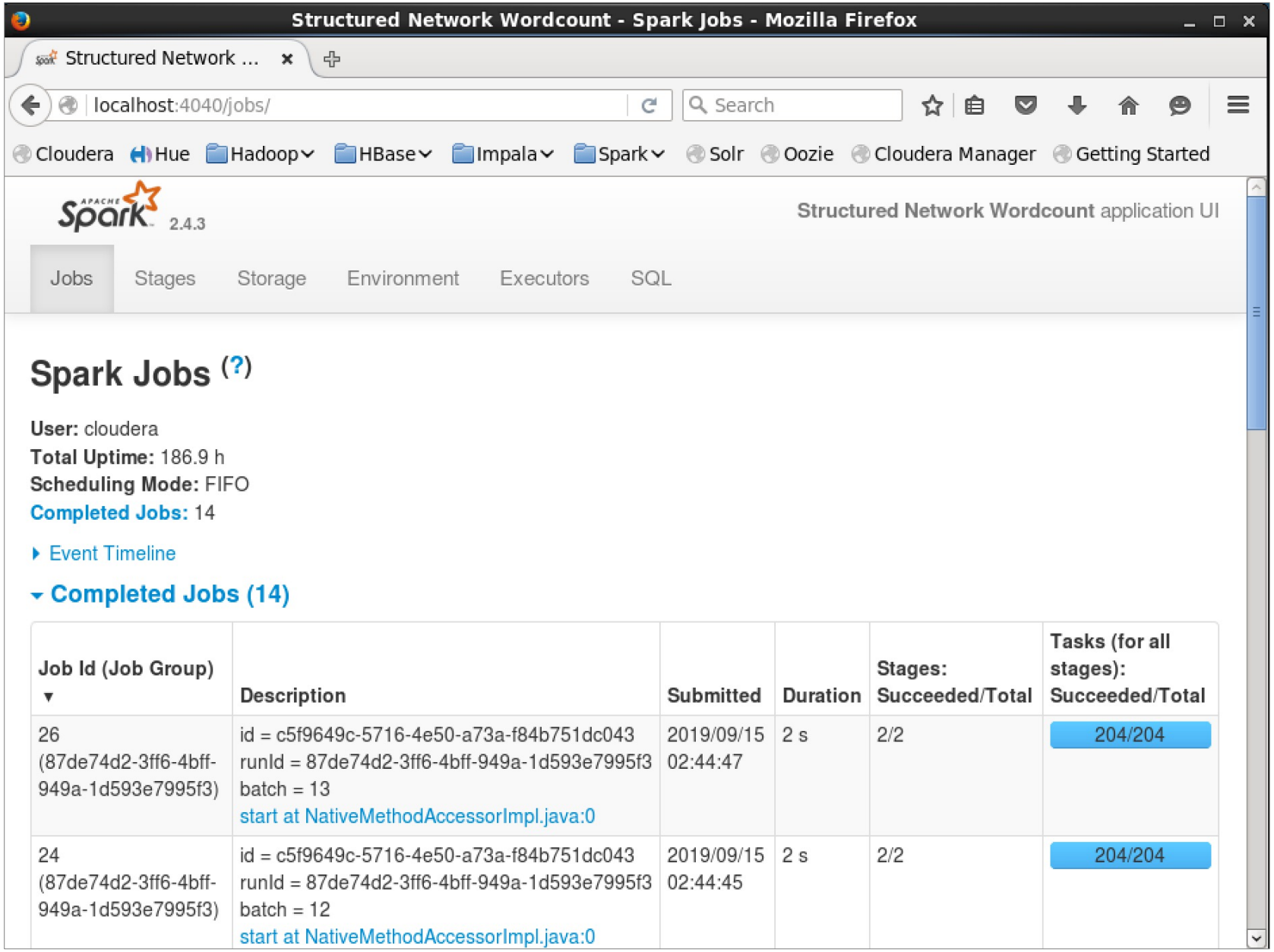


Spark Program Run-time Excution Flow (step by step)

Spark API for Python: **pyspark**



The Driver opens 4040 [4041,4042,…] port for job-level monitoring



Common RDD Transformation Methods: map

map(*f*, *preservesPartitioning=False*)

Return a new RDD by applying a function to each element of this RDD.

```
>>> rdd = sc.parallelize(["b", "a", "c"])
>>> sorted(rdd.map(lambda x: (x, 1)).collect())
[('a', 1), ('b', 1), ('c', 1)]
```

```
In [22]: words = lines.flatMap(lambda x: x.split(" "))

In [23]: words.take(5)
Out[23]: ['The', 'Project', 'Gutenberg', 'EBook', 'of']

In [24]: pairs = words.map(lambda x: (x, 1))

In [25]: pairs.take(5)
Out[25]: [('The', 1), ('Project', 1), ('Gutenberg', 1), ('EBook', 1), ('of', 1)]
```

```
In [7]: from afinn import Afinn

In [8]: model = Afinn()

In [9]: messages = sc.parallelize(["I am happy", "he is sad", "what time is it?"])

In [10]: messages.take(3)
Out[10]: ['I am happy', 'he is sad', 'what time is it?']

In [11]: scores = messages.map(lambda x: model.score(x))

In [12]: scores.take(3)
Out[12]: [3.0, -2.0, 0.0]
```

Reference:

- "Google" afinn for more info
- Install afinn model

  pip install afinn

Common RDD Transformation Methods: flatMap

flatMap(*f*, *preservesPartitioning=False*)

Return a new RDD by first applying a function to all elements of this RDD, and then flattening the results.

```
>>> rdd = sc.parallelize([2, 3, 4])
>>> sorted(rdd.flatMap(lambda x: range(1, x)).collect())
[1, 1, 1, 2, 2, 3]
>>> sorted(rdd.flatMap(lambda x: [(x, x), (x, x)]).collect())
[(2, 2), (2, 2), (3, 3), (3, 3), (4, 4), (4, 4)]
```

```
In [10]: lines = sc.textFile("hdfs://devenv/user/spark/spark101/wordcount/data")

In [11]: lines.take(5)
Out[11]:
['The Project Gutenberg EBook of War and Peace, by Leo Tolstoy',
 '',
 'This eBook is for the use of anyone anywhere at no cost and with almost',
 'no restrictions whatsoever.  You may copy it, give it away or re-use it',
 'under the terms of the Project Gutenberg License included with this']

In [12]: words = lines.flatMap(lambda x: x.split(" "))

In [13]: words.take(15)
Out[13]:
['The',
 'Project',
 'Gutenberg',
 'EBook',
 'of',
 'War',
 'and',
 'Peace,',
 'by',
 'Leo',
 'Tolstoy',
 '',
 'This',
 'eBook',
 'is']
```

Common RDD Transformation Methods: filter

filter(*f*)

Return a new RDD containing only the elements that satisfy a predicate.

```
>>> rdd = sc.parallelize([1, 2, 3, 4, 5])
>>> rdd.filter(lambda x: x % 2 == 0).collect()
[2, 4]
```

```
In [16]: numbers = sc.parallelize([2, 3, 1, 4, 6, 7])

In [17]: numbers.collect()
Out[17]: [2, 3, 1, 4, 6, 7]

In [18]: even_numbers = numbers.filter(lambda x: x % 2 == 0)

In [19]: even_numbers.collect()
Out[19]: [2, 4, 6]
```

Common RDD Transformation Methods: distinct

distinct(*numPartitions=None*) ¶

Return a new RDD containing the distinct elements in this RDD.

```
>>> sorted(sc.parallelize([1, 1, 2, 3]).distinct().collect())
[1, 2, 3]
```

Common RDD Transformation Methods: sort By

sortBy(*keyfunc, ascending=True, numPartitions=None*)

Sorts this RDD by the given keyfunc

```
>>> tmp = [('a', 1), ('b', 2), ('1', 3), ('d', 4), ('2', 5)]
>>> sc.parallelize(tmp).sortBy(lambda x: x[0]).collect()
[('1', 3), ('2', 5), ('a', 1), ('b', 2), ('d', 4)]
>>> sc.parallelize(tmp).sortBy(lambda x: x[1]).collect()
[('a', 1), ('b', 2), ('1', 3), ('d', 4), ('2', 5)]
```

```
In [21]: data = sc.parallelize(["b", "c", "A", "C", "a", "B"])

In [22]: data.collect()
Out[22]: ['b', 'c', 'A', 'C', 'a', 'B']

In [23]: sorted_data = data.sortBy(lambda x: x)

In [24]: sorted_data.collect()
Out[24]: ['A', 'B', 'C', 'a', 'b', 'c']

In [25]: sorted_data = data.sortBy(lambda x: x.lower())

In [26]: sorted_data.collect()
Out[26]: ['A', 'a', 'b', 'B', 'c', 'C']

In [27]: sorted_data = data.sortBy(lambda x: x.lower(), ascending=False)

In [28]: sorted_data.collect()
Out[28]: ['c', 'C', 'b', 'B', 'A', 'a']
```

Common RDD Transformation Methods: union

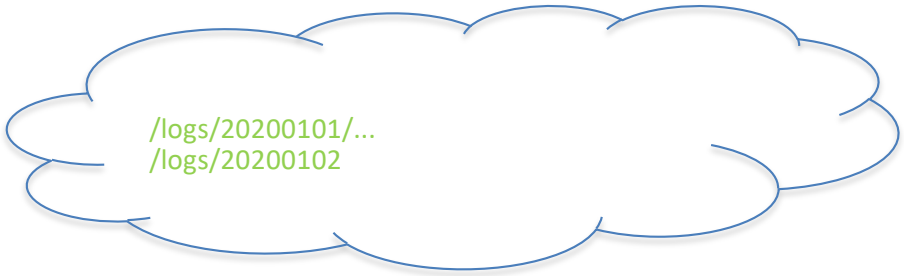
union(*rdds*)

Build the union of a list of RDDs.

This supports unions() of RDDs with different serialized formats, although this forces them to be reserialized using the default serializer:

```
>>> path = os.path.join(tempdir, "union-text.txt")
>>> with open(path, "w") as testFile:
...     _ = testFile.write("Hello")
>>> textFile = sc.textFile(path)
>>> textFile.collect()
[u'Hello']
>>> parallelized = sc.parallelize(["World!"])
>>> sorted(sc.union([textFile, parallelized]).collect())
[u'Hello', 'World!']
```

rdd3 = rdd1.union(rdd2)



Common RDD Action Methods: collect, take, first

collect()

Return a list that contains all of the elements in this RDD.

take(*num*)

Take the first num elements of the RDD.

It works by first scanning one partition, and use the results from that partition to estimate the number of additional partitions needed to satisfy the limit.

Translated from the Scala implementation in RDD#take().

```
>>> sc.parallelize([2, 3, 4, 5, 6]).cache().take(2)
[2, 3]
>>> sc.parallelize([2, 3, 4, 5, 6]).take(10)
[2, 3, 4, 5, 6]
>>> sc.parallelize(range(100), 100).filter(lambda x: x > 90).take(3)
[91, 92, 93]
```

first()

Return the first element in this RDD.

```
>>> sc.parallelize([2, 3, 4]).first()
2
>>> sc.parallelize([]).first()
Traceback (most recent call last):
    ...
ValueError: RDD is empty
```

Common RDD Action Methods: count

count()

Return the number of elements in this RDD.

```
>>> sc.parallelize([2, 3, 4]).count()
3
```

Common RDD Action Methods: reduce

reduce(*f*)

Reduces the elements of this RDD using the specified commutative and associative binary operator. Currently reduces partitions locally.

```
In [1]: numbers = sc.parallelize([5, 3, 2, 7, 2, 1, 6])

In [2]: numbers.collect()
Out[2]: [5, 3, 2, 7, 2, 1, 6]

In [3]: sum_of_numbers = numbers.reduce(lambda x, y: x + y)

In [4]: sum_of_numbers
Out[4]: 26
```

```
In [5]: numbers = sc.parallelize([5, 3, 2, 7, 2, 1, 6])

In [6]: def f(x, y):
   ...:     if x > y:
   ...:         return x
   ...:     else:
   ...:         return y
   ...:

In [7]: max_num = numbers.reduce(f)

In [8]: max_num
Out[8]: 7
```

Working on data in key-value pairs

```
In [19]: day_temp.take(10)
Out[19]:
[(u'20070101', 23),
 (u'20070101', 21),
 (u'20070101', 23),
 (u'20070101', 23),
 (u'20070101', 23),
 (u'20070101', 25),
 (u'20070101', 25),
 (u'20070101', 21),
 (u'20070101', 21),
 (u'20070101', 25)]
```

Common RDD Transformation Methods: mapValues

mapValues(*f*) ¶

Pass each value in the key-value pair RDD through a map function without changing the keys; this also retains the original RDD's partitioning.

```
>>> x = sc.parallelize([("a", ["apple", "banana", "lemon"]), ("b", ["grapes"])])
>>> def f(x): return len(x)
>>> x.mapValues(f).collect()
[('a', 3), ('b', 1)]
```

```
In [6]: data = sc.parallelize([("c", 2), ("a", 3), ("b", 1)])

In [7]: data.collect()
Out[7]: [('c', 2), ('a', 3), ('b', 1)]

In [8]: result = data.mapValues(lambda x: x * 2)

In [9]: result.collect()
Out[9]: [('c', 4), ('a', 6), ('b', 2)]
```

```
In [22]: user_msg = sc.parallelize([("John", "I am fine"), ("May", "The earthquake really scared me")])

In [23]: model = Afinn()

In [24]: def score(msg):
    ...:     global model
    ...:     return model.score(msg)
    ...:

In [25]: user_msg_score = user_msg.mapValues(score)

In [26]: user_msg_score.collect()
Out[26]: [('John', 2.0), ('May', -2.0)]
```

Common RDD Transformation Methods: reduceByKey

reduceByKey(*func*, *numPartitions=None*) ¶

Merge the values for each key using an associative reduce function.

This will also perform the merging locally on each mapper before sending results to a reducer, similarly to a “combiner” in MapReduce.

Output will be hash-partitioned with **numPartitions** partitions, or the default parallelism level if **numPartitions** is not specified.

```
In [4]: sales = sc.parallelize([(1, 30), (2, 25), (1, 20), (2, 40), (2, 15), (1, 50)])

In [5]: def f(x, y):
   ...:     return x + y
   ...:

In [6]: user_total = sales.reduceByKey(f)

In [7]: user_total.collect()
Out[7]: [(2, 80), (1, 100)]
```

```
In [10]: sales = sc.parallelize([(1, 30), (2, 25), (1, 20), (2, 40), (2, 15), (1, 50)])

In [11]: def f(x, y):
    ...:     if x > y:
    ...:         return x
    ...:     else:
    ...:         return y
    ...:

In [12]: user_max = sales.reduceByKey(f)

In [13]: user_max.collect()
Out[13]: [(2, 40), (1, 50)]
```

Demo

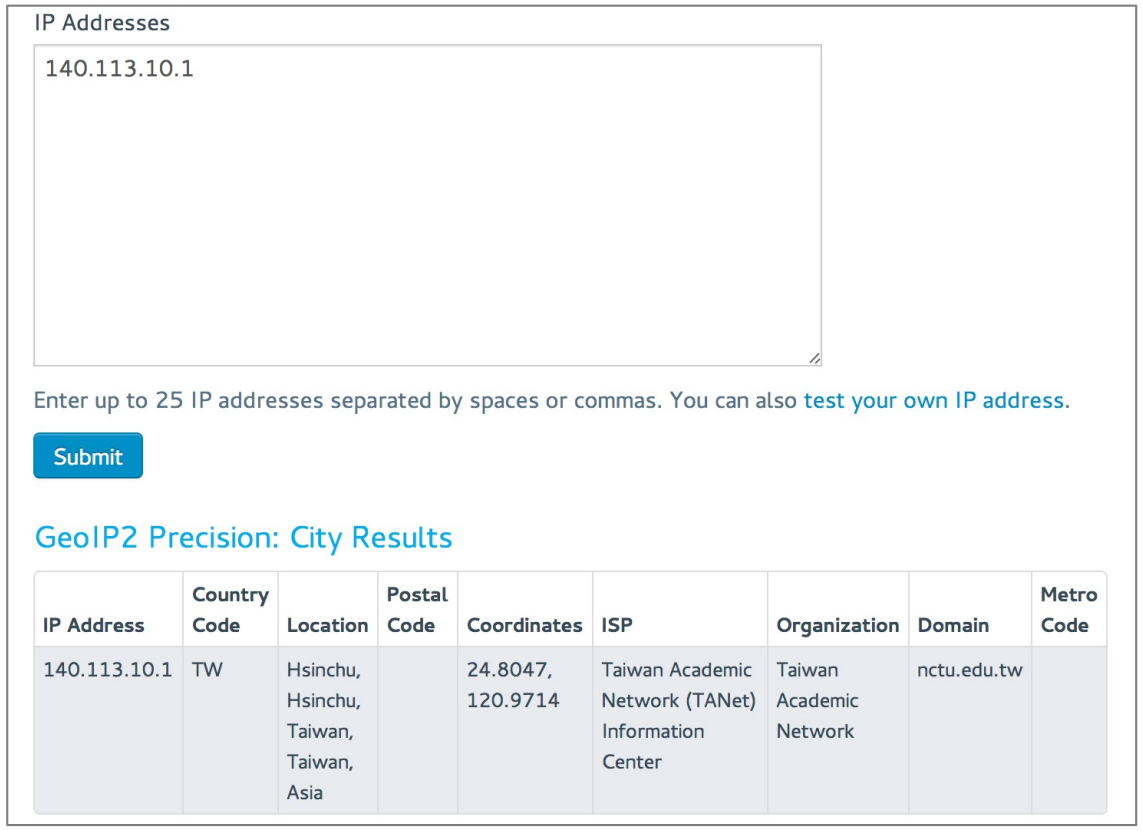
- Caculate daily average temperature on the weather dataset

Demo

- Web logs analysis by calculating top IPs and where the users were from geographically

Demo

- IP Database
  - Determine country, state/region, city, US postal code, US area code, metro code, latitude, and longitude information from the IP addresses worldwide.
  - **MaxMind** IP DB is relatively accurate to the city-level and cost-efficient



IP Addresses

140.113.10.1

Enter up to 25 IP addresses separated by spaces or commas. You can also test your own IP address.

Submit

GeoIP2 Precision: City Results

IP Address	Country Code	Location	Postal Code	Coordinates	ISP	Organization	Domain	Metro Code
140.113.10.1	TW	Hsinchu, Hsinchu, Taiwan, Taiwan, Asia		24.8047, 120.9714	Taiwan Academic Network (TANet) Information Center	Taiwan Academic Network	nctu.edu.tw	

https://www.maxmind.com/en/geoip2-precision-demo

Common RDD Transformation Methods: groupByKey

groupByKey(*numPartitions=None*)

Group the values for each key in the RDD into a single sequence. Hash-partitions the resulting RDD with numPartitions partitions.

Note: If you are grouping in order to perform an aggregation (such as a sum or average) over each key, using reduceByKey or aggregateByKey will provide much better performance.

```
In [17]: sales = sc.parallelize([(1, 30), (2, 25), (1, 20), (2, 40), (2, 15), (1, 50)])

In [18]: user_records = sales.groupByKey()

In [19]: user_records.collect()
Out[19]:
[(2, <pyspark.resultiterable.ResultIterable at 0x7fe69b4cdd10>),
 (1, <pyspark.resultiterable.ResultIterable at 0x7fe69b4cde10>)]
```

Each Iterable object contain values that have the same key

```
In [20]: for r in user_records.collect()[0][1]:
    ...:     print(r)
    ...:
25
40
15
```

Common RDD Transformation Methods: join

join(*other*, *numPartitions=None*)

Return an RDD containing all pairs of elements with matching keys in **self** and **other**.

Each pair of elements will be returned as a (k, (v1, v2)) tuple, where (k, v1) is in **self** and (k, v2) is in **other**.

Performs a hash join across the cluster.

```
>>> x = sc.parallelize([("a", 1), ("b", 4)])
>>> y = sc.parallelize([("a", 2), ("a", 3)])
>>> sorted(x.join(y).collect())
[('a', (1, 2)), ('a', (1, 3))]
```

Common RDD Transformation Methods: leftOuterJoin, rightOuterJoin

leftOuterJoin(*other*, *numPartitions=None*) ¶

Perform a left outer join of **self** and **other**.

For each element (k, v) in **self**, the resulting RDD will either contain all pairs (k, (v, w)) for w in **other**, or the pair (k, (v, None)) if no elements in **other** have key k.

Hash-partitions the resulting RDD into the given number of partitions.

```
>>> x = sc.parallelize([("a", 1), ("b", 4)])
>>> y = sc.parallelize([("a", 2)])
>>> sorted(x.leftOuterJoin(y).collect())
[('a', (1, 2)), ('b', (4, None))]
```

rightOuterJoin(*other*, *numPartitions=None*)

Perform a right outer join of **self** and **other**.

For each element (k, w) in **other**, the resulting RDD will either contain all pairs (k, (v, w)) for v in this, or the pair (k, (None, w)) if no elements in **self** have key k.

Hash-partitions the resulting RDD into the given number of partitions.

```
>>> x = sc.parallelize([("a", 1), ("b", 4)])
>>> y = sc.parallelize([("a", 2)])
>>> sorted(y.rightOuterJoin(x).collect())
[('a', (2, 1)), ('b', (None, 4))]
```

Common RDD Transformation Methods: fullOuterJoin

fullOuterJoin(*other*, *numPartitions=None*)

Perform a right outer join of **self** and **other**.

For each element (k, v) in **self**, the resulting RDD will either contain all pairs (k, (v, w)) for w in **other**, or the pair (k, (v, None)) if no elements in **other** have key k.

Similarly, for each element (k, w) in **other**, the resulting RDD will either contain all pairs (k, (v, w)) for v in **self**, or the pair (k, (None, w)) if no elements in **self** have key k.

Hash-partitions the resulting RDD into the given number of partitions.

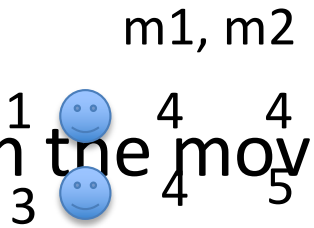
```
>>> x = sc.parallelize([("a", 1), ("b", 4)])
>>> y = sc.parallelize([("a", 2), ("c", 8)])
>>> sorted(x.fullOuterJoin(y).collect())
[('a', (1, 2)), ('b', (4, None)), ('c', (None, 8))]
```

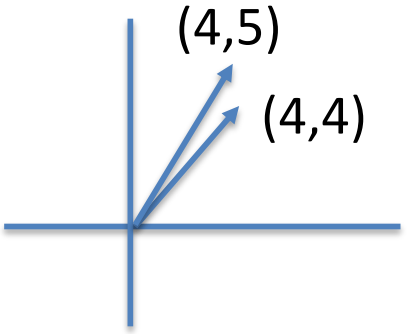
Common RDD Transformation Methods: example for each join type

```
In [30]: x = sc.parallelize([("a", 1), ("b", 4)])

In [31]: y = sc.parallelize([("a", 2), ("a", 3), ("c", 1)])

In [32]: joined = x.join(y)

In [33]: joined.collect()
Out[33]: [('a', (1, 2)), ('a', (1, 3))]

In [34]: ljoined = x.leftOuterJoin(y)

In [35]: ljoined.collect()
Out[35]: [('b', (4, None)), ('a', (1, 2)), ('a', (1, 3))]

In [36]: rjoined = x.rightOuterJoin(y)

In [37]: rjoined.collect()
Out[37]: [('c', (None, 1)), ('a', (1, 2)), ('a', (1, 3))]

In [38]: fjoined = x.fullOuterJoin(y)

In [39]: fjoined.collect()
Out[39]: [('b', (4, None)), ('c', (None, 1)), ('a', (1, 2)), ('a', (1, 3))]
```

Demo

- Calculate movie similarity from the movie data set





Similarity

$$\cos(\theta) = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\|\|\mathbf{B}\|} = \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2}\sqrt{\sum_{i=1}^{n} B_i^2}},$$

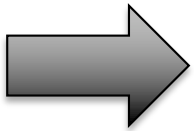
Demo

For those users who wathced the 2 movies, they gave raitings to each movie and caculate how similar the 2 movies are in terms of cosine similarity ratings

How many users watched this 2 movies

uid mid 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
253 465 5   891628467
305 451 3   886324817
6   86  3   883603013
62  257 2   879372434
286 1014    5   879781125
200 222 5   876042340
210 40  3   891035994
224 29  3   888104457
303 785 3   879485318
122 387 5   879270459
194 274 2   879539794
291 1042    4   874834944
234 1184    2   892079237
119 392 4   886176814
167 486 4   892738452
299 144 4   877881320
291 118 2   874833878
308 1   4   887736532
95  546 2   879196566
38  95  5   892430094
102 768 2   883748450
63  277 4   875747401
160 234 5   876861185
50  246 3   877052329
```



```
((155, 655), (0.9376268998353996, 64)),
((155, 527), (0.9246347060490497, 47)),
((155, 763), (0.9164987696292339, 36)),
((155, 235), (0.8595668459274877, 55)),
((155, 871), (0.8108878578850678, 13)),
((155, 1119), (0.9685219447071304, 28)),
((155, 1095), (0.9847982464479192, 6)),
((155, 1467), (1.0, 1)),
((155, 171), (0.8636720533015911, 18)),
((155, 651), (0.9421833038218413, 45)),
((155, 291), (0.934231719654843, 22)),
((155, 383), (0.8715419652402461, 13)),
((155, 735), (0.9171019716934296, 45)),
((155, 455), (0.8645756728867536, 32)),
((7, 51), (0.9411764734600292, 54)),
((7, 815), (0.94009895392923, 75)),
((7, 707), (0.9713198055306043, 31)),
((7, 183), (0.9639396975489566, 192)),
((7, 167), (0.9376825472812673, 44)),
((7, 155), (0.9107280285219985, 63)),
((7, 31), (0.9621903716618524, 107)),
((7, 743), (0.8730491945191836, 32)),
((7, 95), (0.9446838550185377, 139)),
((7, 179), (0.9575002788490408, 139)),
((7, 747), (0.9487554205837383, 64)),
((7, 287), (0.9537057816929797, 52)),
((7, 411), (0.9230088438224416, 114)),
((7, 191), (0.9568307872084653, 158)),
((7, 1107), (0.9223600335530932, 13)),
((7, 1039), (0.9629387230776822, 55)),
```

*~/Desktop/spark101/movies_similarity/movie_ratings/**

Persistence

- Spark RDDs are lazily evaluated, Spark will recompute the RDD and all of its dependencies each time we call an action on the RDD
- We can ask Spark to cache the data. Spark has many levels of persistence to choose from.
- You can mark an RDD to be persisted using the persist() or cache() methods on it.

Persistence

```
import re
from pyspark import SparkContext

if __name__ == "__main__":
    sc = SparkContext()

        lines = sc.textFile("hdfs://devenv/user/spark/spark101/wordcount/data")

        words = lines.flatMap(lambda x: re.compile(r'\W+', re.UNICODE).split(x.lower()))

        word_counts = words.map(lambda x: (x, 1)).reduceByKey(lambda x, y: x + y)
        word_counts.persist()
        word_counts.saveAsTextFile("hdfs://devenv/user/spark/spark101/wordcount/output3")
        word_counts_list = word_counts.collect()

        for e in word_counts_list:
            print(e)
```

- Normall, before appying 2 or more than 2 action on the same RDD object, persistance are supposed to be used.
- The first action triger a job that runs full logics plus creates the cache in the cluster
- The subsequent jobs on the same RDD can use the cache for outputs (if the cache does not expire)

Persistence

Storage Level	Meaning
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 (Java and Scala)	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 (Java and Scala)	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.
OFF_HEAP (experimental)	Similar to MEMORY_ONLY_SER, but store the data in off-heap memory. This requires off-heap memory to be enabled.

Accumulators

- Accumulators are variables that are only “added” to through an associative and commutative operation and can therefore be efficiently supported in parallel.
- Can be used to implement counters (as in MapReduce) or sums

```
>>> accum = sc.accumulator(0)
>>> accum
Accumulator<id=0, value=0>

>>> sc.parallelize([1, 2, 3, 4]).foreach(lambda x: accum.add(x))
...
10/09/29 18:41:08 INFO SparkContext: Tasks finished in 0.317106 s

>>> accum.value
10
```

Accumulator

```
from pyspark import SparkConf, SparkContext

def is_good(record):
    try:
        temp = int(record.split(",")[10])
    except ValueError:
        bad_record_counter.add(1)
        return False
    return True


if __name__ == "__main__":
    sc = SparkContext()
    bad_record_counter = sc.accumulator(0)
    records = sc.textFile("hdfs://devenv/user/spark/spark101/avg_temperature/data")

    good_records = records.filter(is_good)

    day_temp = good_records.map(lambda x: (x.split(",")[1], int(x.split(",")[10])))

    result = day_temp.mapValues(lambda x: (x, 1)) \
        .reduceByKey(lambda x, y: (x[0] + y[0], x[1] + y[1])) \
        .map(lambda x: (x[0], x[1][0] / x[1][1]))

    for line in result.collect():
        print(line)

    print("Bad record count: {}".format(bad_record_counter.value))
```

Demo:
Multi-node Hadoop/Spark Deployment