

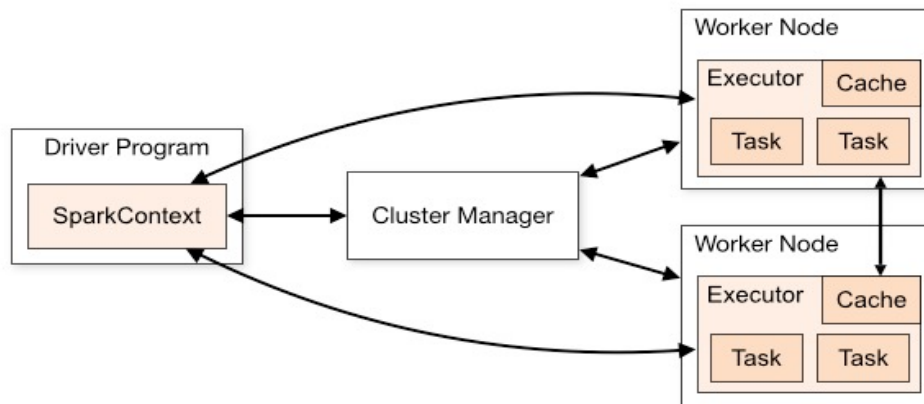
# SPARK CLUSTER OVERVIEW

---

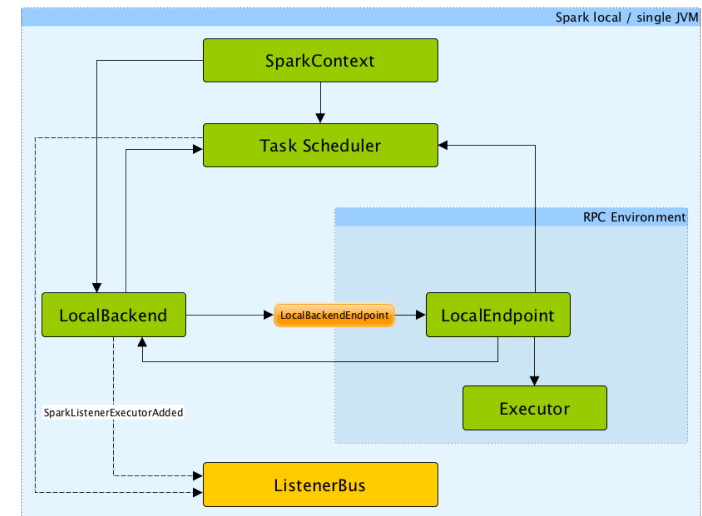
# Spark Execution modes

It is possible to run a spark application using **cluster mode**, **local mode** (pseudo-cluster) or with an **interactive** shell (*pyspark* or *spark-shell*).

## Cluster mode

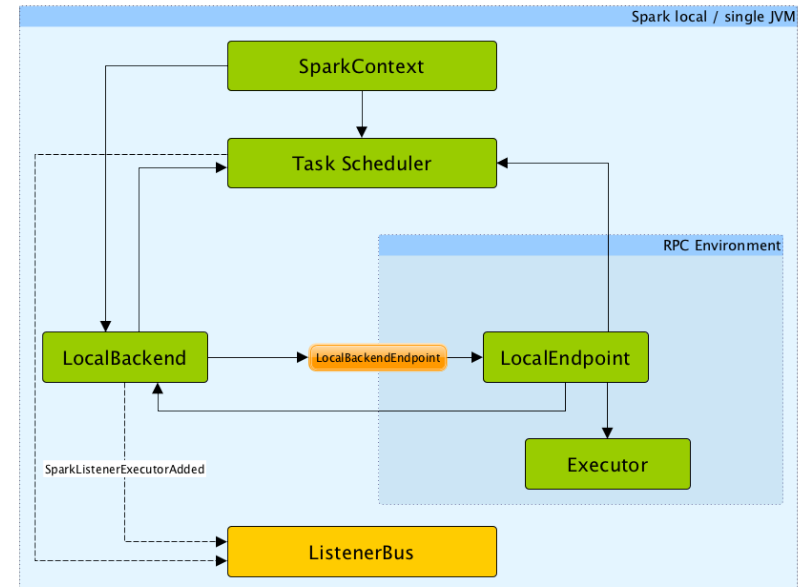


## Local mode



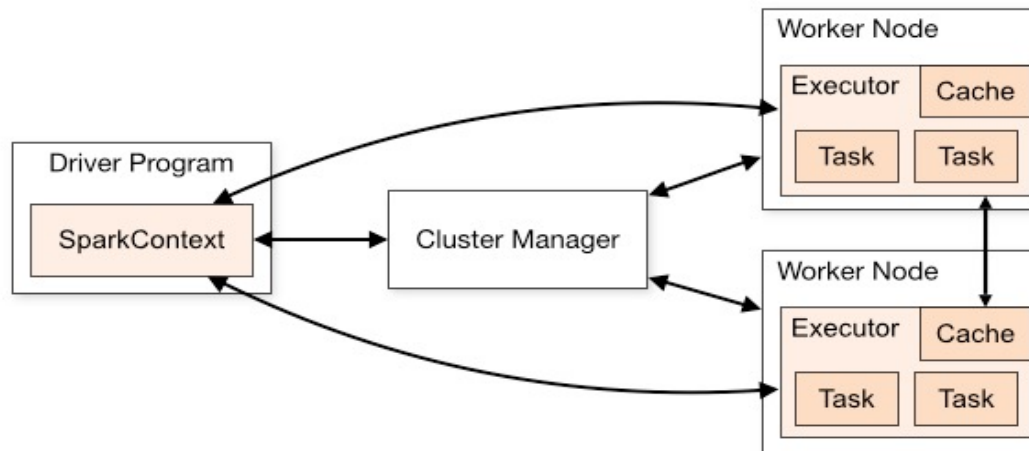
# Spark Execution – Local mode

- In this non-distributed single-JVM deployment mode.
- Spark spawns all the execution components - [driver](#), [executor](#), [LocalSchedulerBackend](#), and [master](#) - in the same single JVM.
- The default parallelism is the number of threads as specified in the [master URL](#).



# Spark Execution – Cluster mode

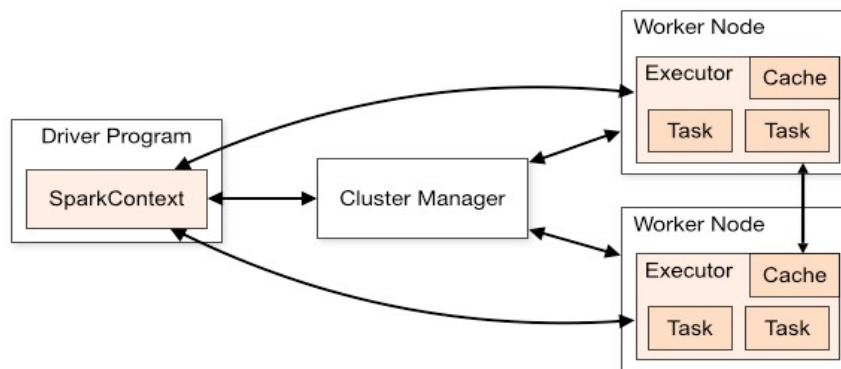
- **Standalone**: simplest way to deploy Spark on a private cluster
- Apache Mesos
- Hadoop YARN
- Kubernetes



Spark is agnostic to the underlying cluster manager

## Spark Execution – Cluster mode

- Spark applications are run as independent sets of processes, coordinated by a SparkContext in a (\*) *driver* program.
- The *context* connects to the cluster manager *which allocates resources*.

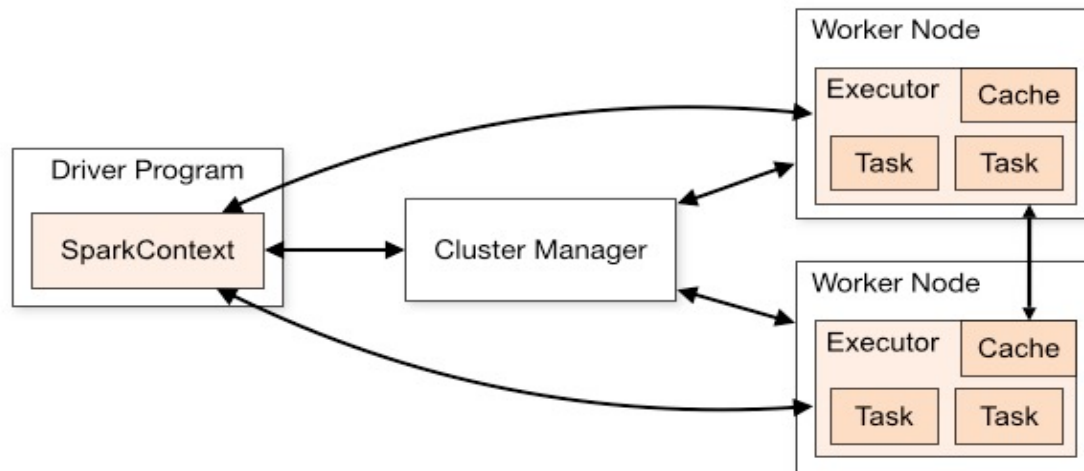


- Each *worker* in the cluster is managed by an *executor*.
- The *executor* manages computation as well as storage and caching on each machine.

(\*) *driver* → process running the *main()* function of the application and creating the *SparkContext*

## Spark Execution – Cluster mode

- The application code is sent from the *driver* to the *executors*, and the executors specify the context and the various *tasks* to be run.
- The *driver* program must listen for and accept incoming connections from its executors throughout its lifetime.

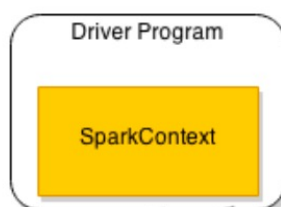


# Spark Execution – Cluster mode

## Spark App

Each SparkContext creates a Spark application, which includes a lot of scheduling components.

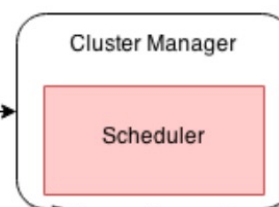
Upon an **Action**, the driver program submits the job to the cluster manager.



## Cluster manager

Start executors on Worker Nodes.

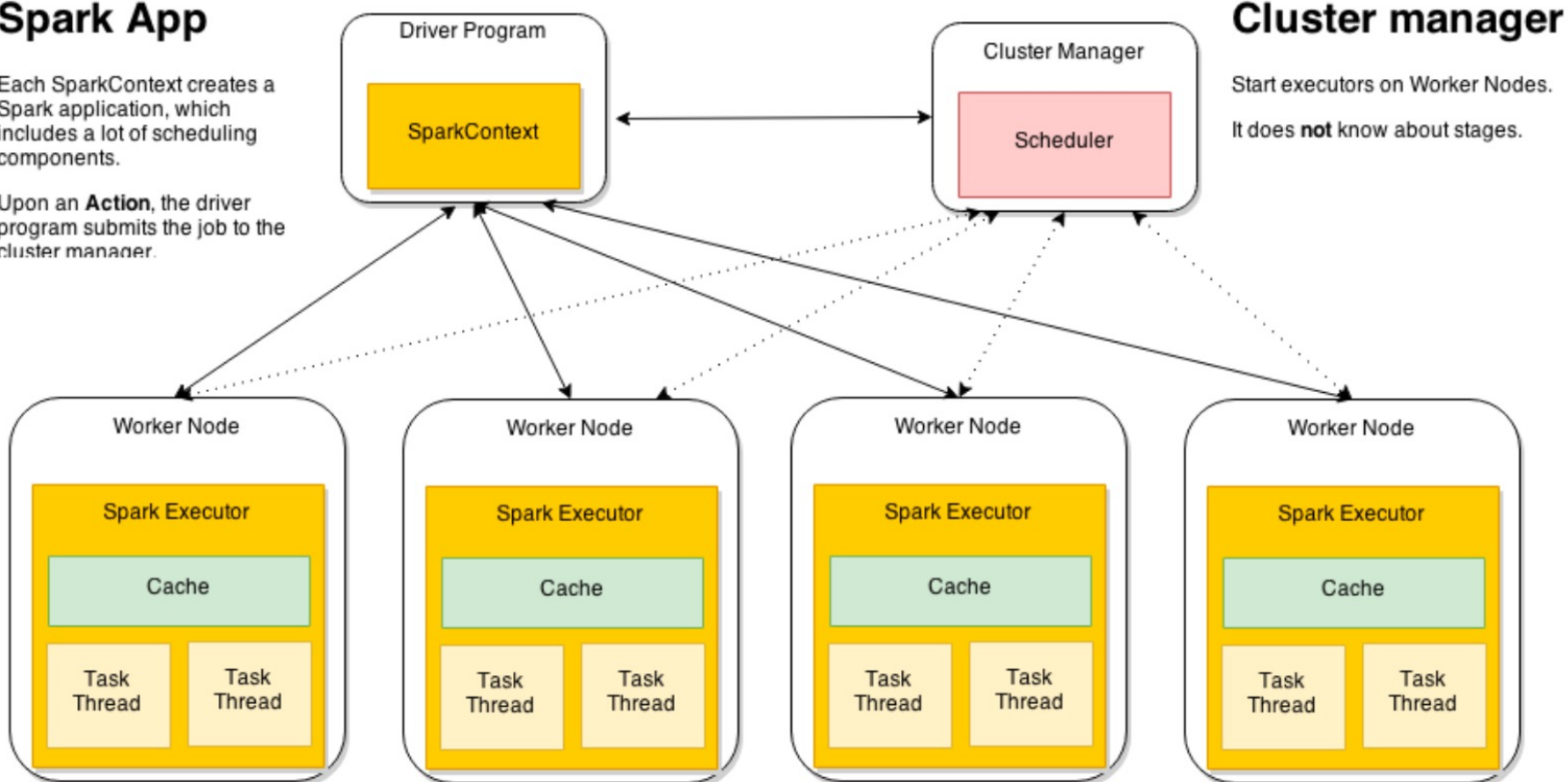
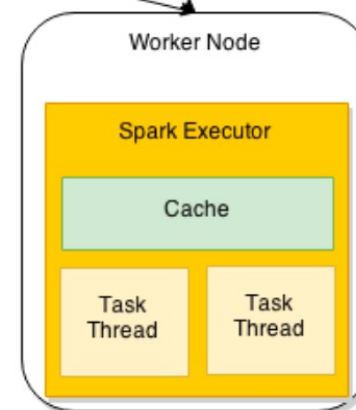
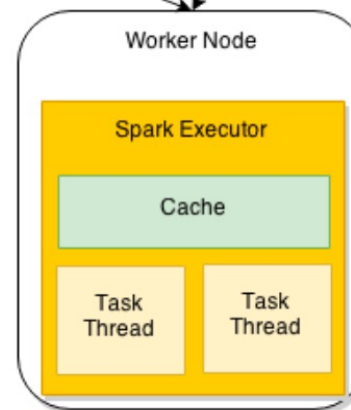
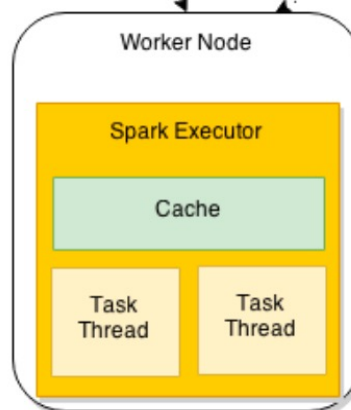
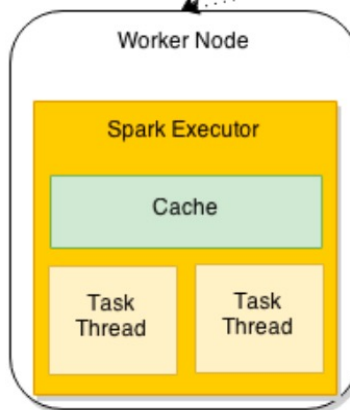
It does **not** know about stages.



## Worker

Launch Spark Executor in a process.

Tasks are launched in separate threads, one per each core on the worker node (can be configured)



# Spark – Standalone Cluster – Deploy modes

For standalone clusters supports two deploy modes.  
They distinguish where the *driver* process runs:

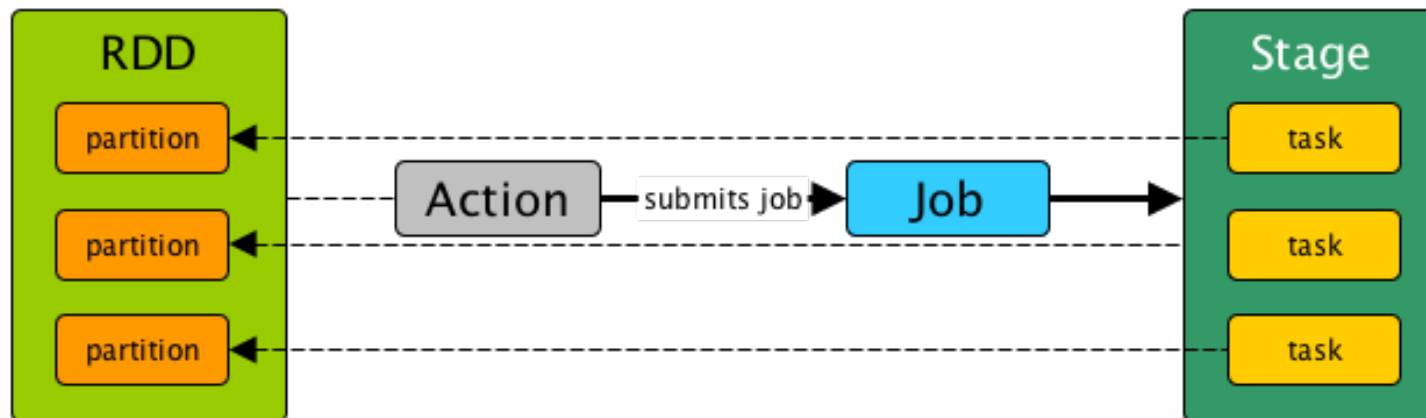
- *Client mode (by default)*: the *driver* is launched in the *same process as the client* that submits the application.
- *Cluster mode*: the *driver* is launched from *one of the Worker processes* inside the cluster.
  - The client process exits as soon as it fulfils its responsibility of submitting the application without waiting for the application to finish.

Note: Currently, the **standalone mode** does not support **cluster mode** for **Python applications**.

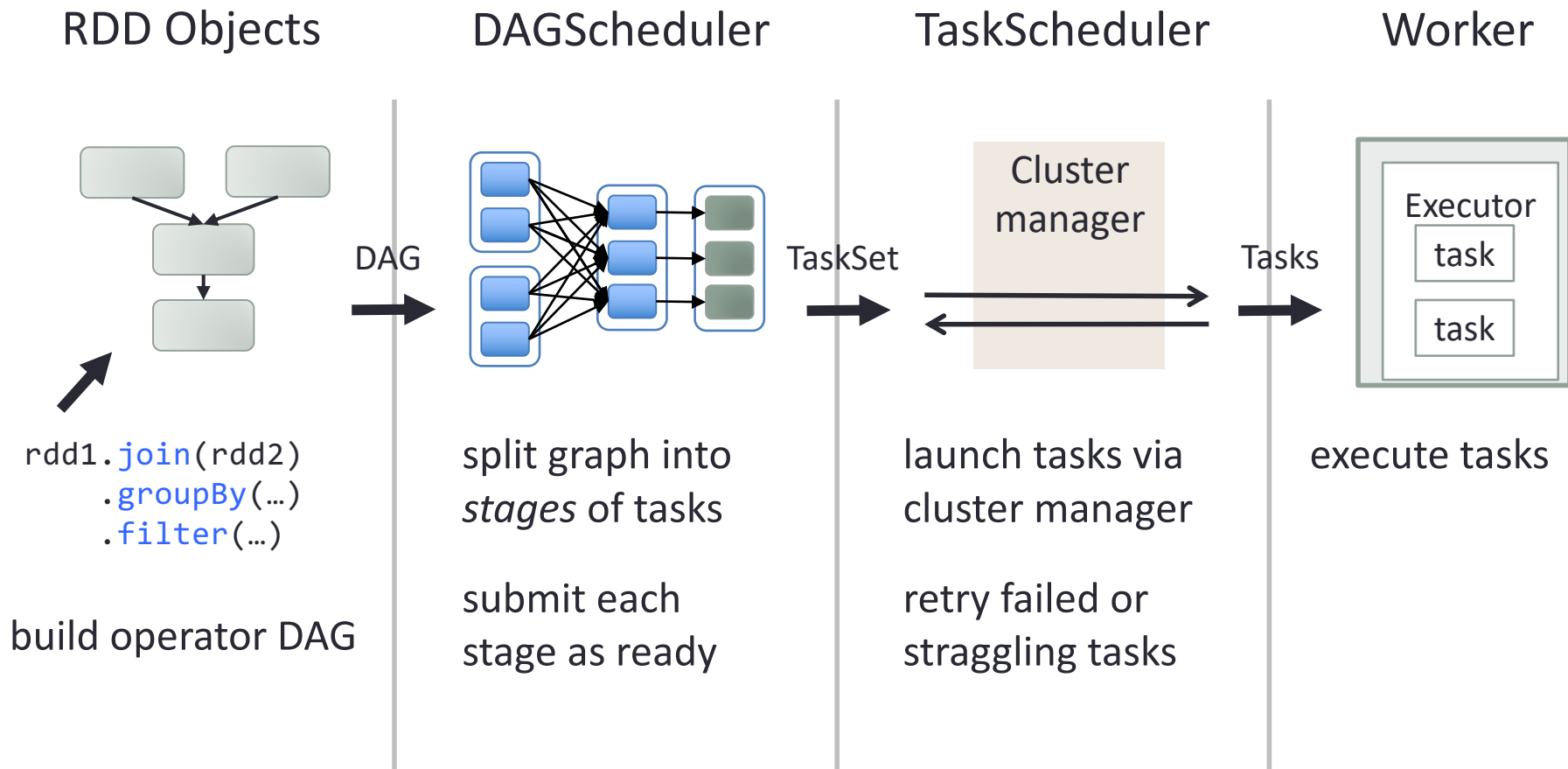


# Spark Components

- Task: individual unit of work sent to one executor over a sequences of partitions
- Job : set of tasks executed as a result of an action
- Stage: set of tasks in a job that can be executed in parallel – at partition level
- RDD: Parallel dataset with partitions
- DAG: Logical Graph of RDD operations



# Job scheduling



## Spark Application – wordcount.py

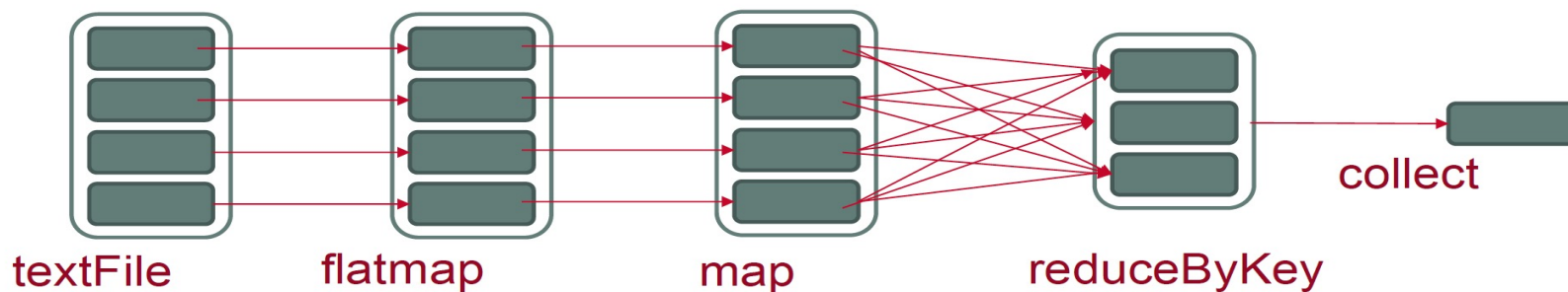
The application that we are going to create is a simple “wordcount”:

- Performs a ***textFile*** operation to read an input file in HDFS
- ***flatMap*** operation to split each line into words
- ***map*** operation to form (word, 1) pairs
- ***reduceByKey*** operation to sum the counts (all the '1') for each word

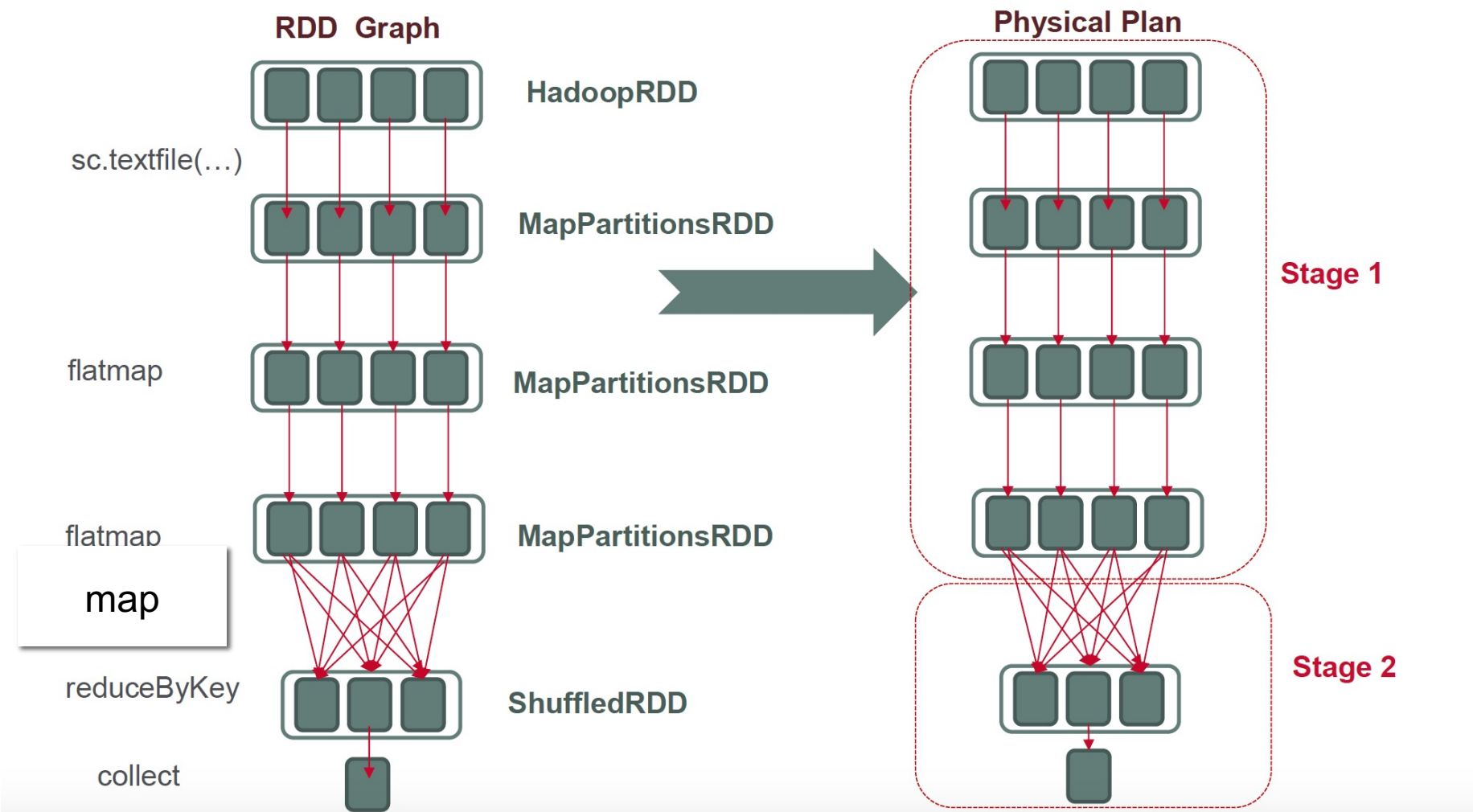
# Spark Application – wordcount.py

```
import sys
from pyspark import SparkContext, SparkConf

if __name__ == "__main__":
    conf = SparkConf().setAppName("Spark Count")
    sc = SparkContext(conf=conf)
    inputFile = sys.argv[1]
    textFile = sc.textFile(inputFile)
    wordCounts = textFile.flatMap(lambda line: line.split()).\
        map(lambda word: (word, 1)).reduceByKey(lambda a, b: a+b)
    output=wordCounts.collect()
    for (word, count) in output:
        print("%s: %i" % (word, count))
```

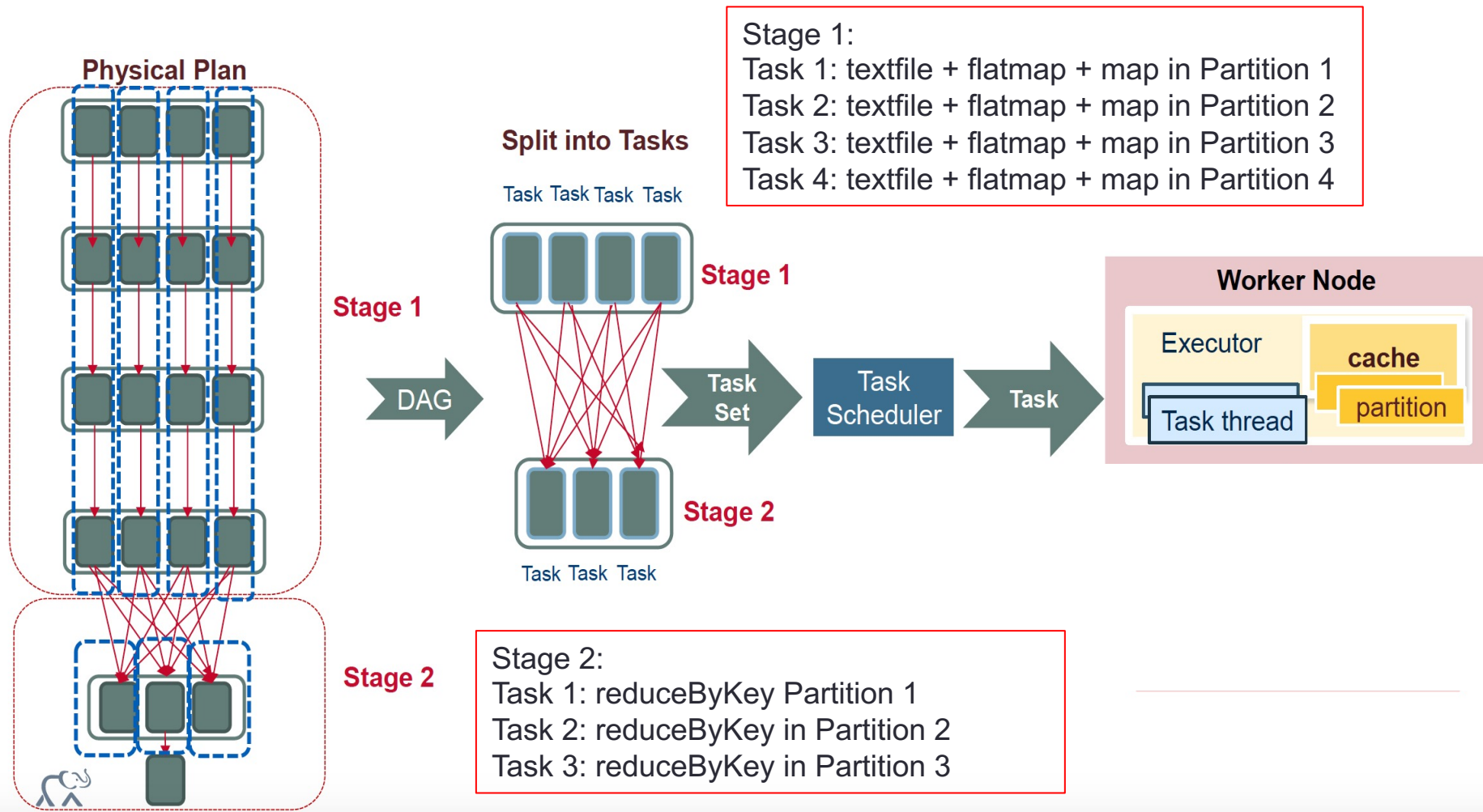


# RDD DAG -> Physical Execution plan



Initial RDD distributed among 4 partitions. Final RDD distributed among 3 partitions

# Execution plan -> Stages and Tasks



Operations that can run on the same partition are executed in stages

# Running Spark Applications

- **Notebooks** are great for:
  - developing and testing quickly experiment with the data
  - demos and collaborating with other people
- **Spark-submit** jobs are more likely to be used in **production**.

# Running Spark with Jupyter Notebooks

We are going to use Jupyter Notebooks for running our walkthroughs & lab exercises.

First we need to do the following steps:

- Copying all the material necessaire in our accounts in Cirrus
- Starting an interactive session in a node
- Starting a spark cluster (standalone) in that node
- Starting a Jupyter session connected with pyspark

All the information can be found at “Get\_Started\_Notebooks\_Cirrus”:

[https://github.com/EPCCed/prace-spark-for-data-scientists/blob/master/Get\\_Started\\_Notebooks\\_Cirrus.pdf](https://github.com/EPCCed/prace-spark-for-data-scientists/blob/master/Get_Started_Notebooks_Cirrus.pdf)



# Submit job via spark-submit

## spark-submit Syntax

```
spark-submit --option value \  
  application jar | python file [application arguments]
```

Check the guide - Submitting Spark Applications:

[https://github.com/EPCed/prace-spark-for-data-scientists/blob/master/Spark\\_Applications/Submitting\\_Spark\\_Applications.pdf](https://github.com/EPCed/prace-spark-for-data-scientists/blob/master/Spark_Applications/Submitting_Spark_Applications.pdf)

## Submit job via spark-submit

```
$SPARK_HOME/bin/spark-submit \  
--class <main-class> \  
--master <master-url> \  
--deploy-mode <deploy-mode> \  
--conf \  
....  
<application-jar> [arguments] |  
<python file >[arguments]
```

## Some spark-submit options

- master – Determines how to run the job:
  - spark://r1i2n5:7077
  - local
- driver-memory
  - amount memory available for the driver process.
- executor-memory
  - amount of memory allocated to the executor process
- executor-cores
  - total number of cores allocated to the executor process
- total-executor-cores
  - Total number of cores available for all executors.

**Note:** <https://spark.apache.org/docs/latest/submitting-applications.html>

# Cirrus

- High-performance computing cluster
- One of the EPSRC Tier-2 National HPC Services.
- 280 nodes: **36 Intel Xeon CPUs, hyper threading, 256GB**
  - Each node has ( virtually ) 72 cores
- 406 TB of storage- Lustre
- Link: <http://www.cirrus.ac.uk/>

<https://cirrus.readthedocs.io/en/latest/user-guide/connecting.html>

# Cirrus

- Connecting to Cirrus

```
ssh [userID]@login.cirrus.ac.uk
```

- Two types of nodes:
  - Login – access to outside network
  - Computing – only network between nodes ( no to outside world).
- For cloning the repository -> use the login node
  - git clone <https://github.com/EPCed/prace-spark-for-data-scientists.git>

<https://cirrus.readthedocs.io/en/latest/user-guide/connecting.html>

# Running jobs in Cirrus

- PBSPPro to schedule jobs
  - Submission script to submit a job a queue
  - Interactive jobs → are also available
    - To submit a request for an interactive job reserving 1 nodes (72 physical cores) for 1 hour you would issue the following qsub command from the command line

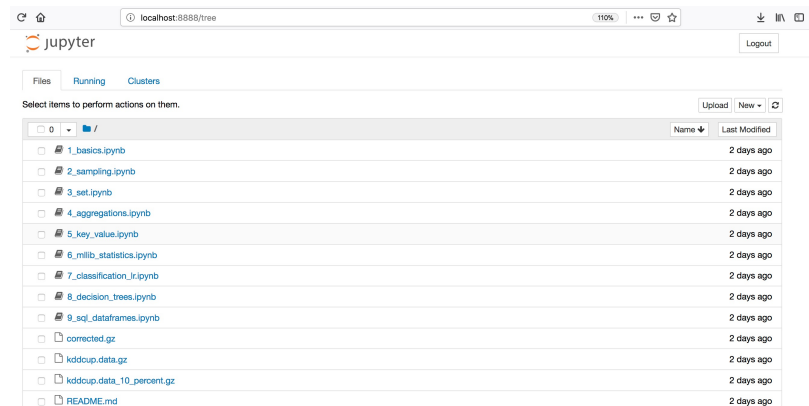
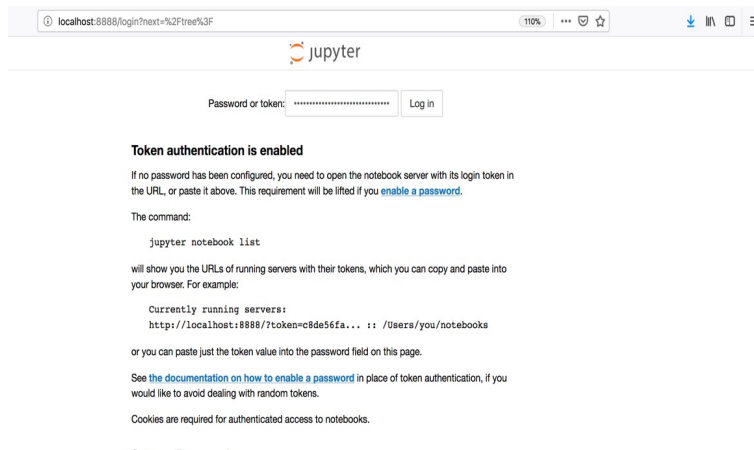
```
qsub -IVl select=3:ncpus=36,walltime=05:00:00,place=scatter:excl -A y15 -q  
$1-j oe
```

- Your session will end:
  - It hits the requested **walltime**
  - Typing **exit command** within the session

<https://cirrus.readthedocs.io/en/latest/user-guide/batch.html#interactive-jobs>


# Jupyter notebooks

- Start the jupyter server:  
`./start_Jupyter_local.sh <master>` → It will give you a token, like this one:  
<http://0.0.0.0:8888/?token=2d5e554b2397355c334b8c3367503b06c4f6f95a26151795>
- Open **another terminal** and type the following command  
`>> ssh USER@login.cirrus.ac.uk -L8888:MASTER NODE:8888`  
Got to a Web browser and type → <http://localhost:8888>



All the information can be found at “Get\_Started\_Notebooks\_Cirrus”:  
[https://github.com/EPCCed/prace-spark-for-data-scientists/blob/master/Get\\_Started\\_Notebooks\\_Cirrus.pdf](https://github.com/EPCCed/prace-spark-for-data-scientists/blob/master/Get_Started_Notebooks_Cirrus.pdf)

# Master Spark UI



2.4.0

Spark Master at spark://r1i1n20:7077

URL: spark://r1i1n20:7077

Alive Workers: 1

Cores in use: 72 Total, 0 Used

Memory in use: 250.6 GB Total, 0.0 B Used

Applications: 0 Running, 0 Completed

Drivers: 0 Running, 0 Completed

Status: ALIVE

Workers (1)

Worker Id	Address	State	Cores	Memory
worker-20190106070903-10.148.0.44-32960	10.148.0.44:32960	ALIVE	72 (0 Used)	250.6 GB (0.0 B Used)

Running Applications (0)

Application ID	Name	Cores	Memory per Executor	Submitted Time	User	State	Duration
----------------	------	-------	---------------------	----------------	------	-------	----------

Completed Applications (0)

Application ID	Name	Cores	Memory per Executor	Submitted Time	User	State	Duration
----------------	------	-------	---------------------	----------------	------	-------	----------

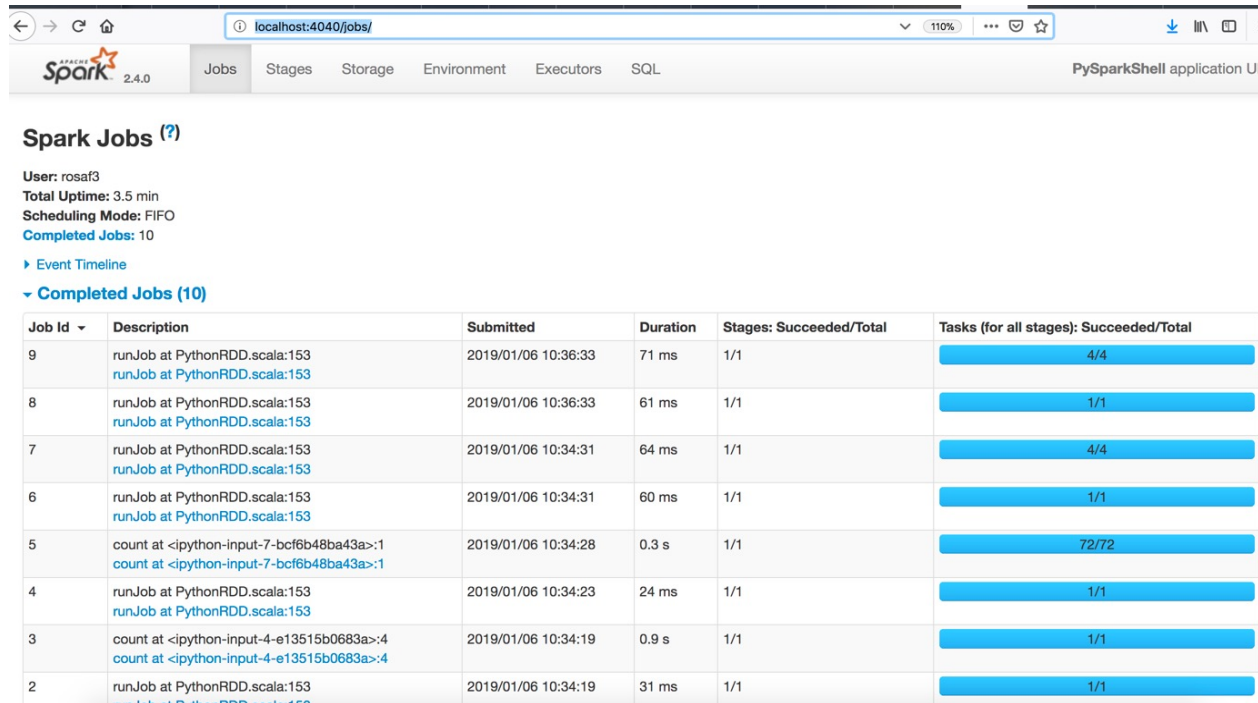
The spark setup can be monetarised via the **Master's web UI**

```
>> ssh USER@login.cirrus.ac.uk -L8080:MASTER NODE:8080
```

Got to a Web browser and type localhost:8080



# Driver Spark UI



The screenshot shows the Spark Driver UI in a web browser. The address bar is `localhost:4040/jobs/`. The Spark logo and version 2.4.0 are visible. The navigation bar includes links for Jobs, Stages, Storage, Environment, Executors, and SQL. The page title is "Spark Jobs (?)". Below the title, it shows "User: rosaf3", "Total Uptime: 3.5 min", "Scheduling Mode: FIFO", and "Completed Jobs: 10". There is a link for "Event Timeline" and a section for "Completed Jobs (10)". A table lists the jobs with columns: Job Id, Description, Submitted, Duration, Stages: Succeeded/Total, and Tasks (for all stages): Succeeded/Total. The table contains 10 rows of job data.

Job Id	Description	Submitted	Duration	Stages: Succeeded/Total	Tasks (for all stages): Succeeded/Total
9	runJob at PythonRDD.scala:153 <a href="#">runJob at PythonRDD.scala:153</a>	2019/01/06 10:36:33	71 ms	1/1	4/4
8	runJob at PythonRDD.scala:153 <a href="#">runJob at PythonRDD.scala:153</a>	2019/01/06 10:36:33	61 ms	1/1	1/1
7	runJob at PythonRDD.scala:153 <a href="#">runJob at PythonRDD.scala:153</a>	2019/01/06 10:34:31	64 ms	1/1	4/4
6	runJob at PythonRDD.scala:153 <a href="#">runJob at PythonRDD.scala:153</a>	2019/01/06 10:34:31	60 ms	1/1	1/1
5	count at <python-input-7-bcf6b48ba43a>:1 count at <python-input-7-bcf6b48ba43a>:1	2019/01/06 10:34:28	0.3 s	1/1	72/72
4	runJob at PythonRDD.scala:153 <a href="#">runJob at PythonRDD.scala:153</a>	2019/01/06 10:34:23	24 ms	1/1	1/1
3	count at <python-input-4-e13515b0683a>:4 count at <python-input-4-e13515b0683a>:4	2019/01/06 10:34:19	0.9 s	1/1	1/1
2	runJob at PythonRDD.scala:153 <a href="#">runJob at PythonRDD.scala:153</a>	2019/01/06 10:34:19	31 ms	1/1	1/1

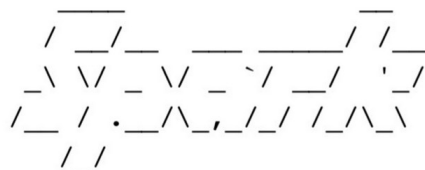
Every SparkContext launches a web UI (**Spark driver's web UI**), by default on port 4040, that displays useful information about the application.

```
ssh USER@login.cirrus.ac.uk -L4040:DRIVER NODE:4040
web browser → localhost:4040
```

# Running notebooks in your laptop

- **Prerequisites: Anaconda, Python3**
- Get Spark from the [downloads page](#) of the project website  
(<https://blog.sicara.com/get-started-pyspark-jupyter-guide-tutorial-ae2fe84f594f> )
- Check if pyspark is properly install → type pyspark in a terminal

Welcome to



- >> git clone <https://github.com/EPCCed/prace-spark-for-data-scientists.git>
- >> cd walkthrough\_examples
- >> export SPARK\_HOME=[INSTALLATION\_PATH]/spark-2.4.0-bin-hadoop2.7/
- >> export PYSPARK\_DRIVER\_PYTHON=jupyter
- >> export PYSPARK\_DRIVER\_PYTHON\_OPTS='notebook'
- >> \$SPARK\_HOME/bin/pyspark