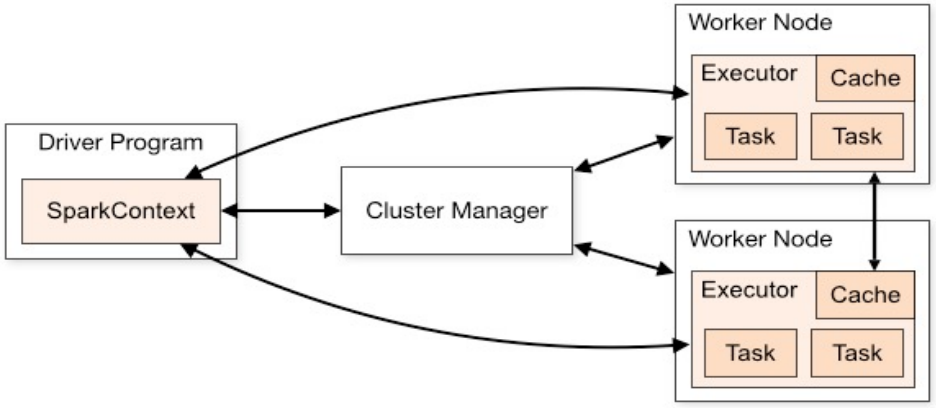
SPARK CLUSTER OVERVIEW

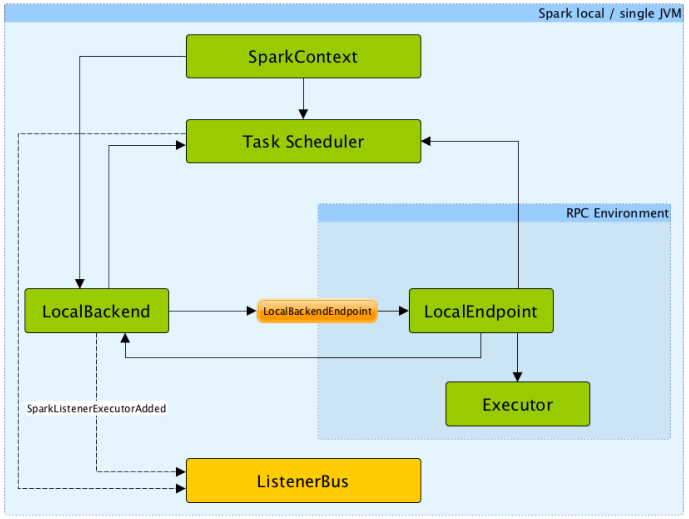
Spark Execution modes

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

Cluster mode

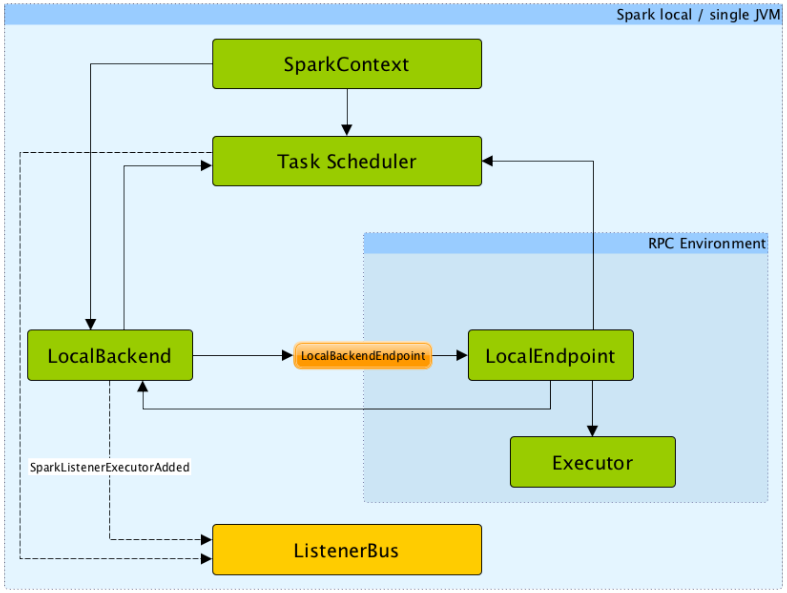


Local mode



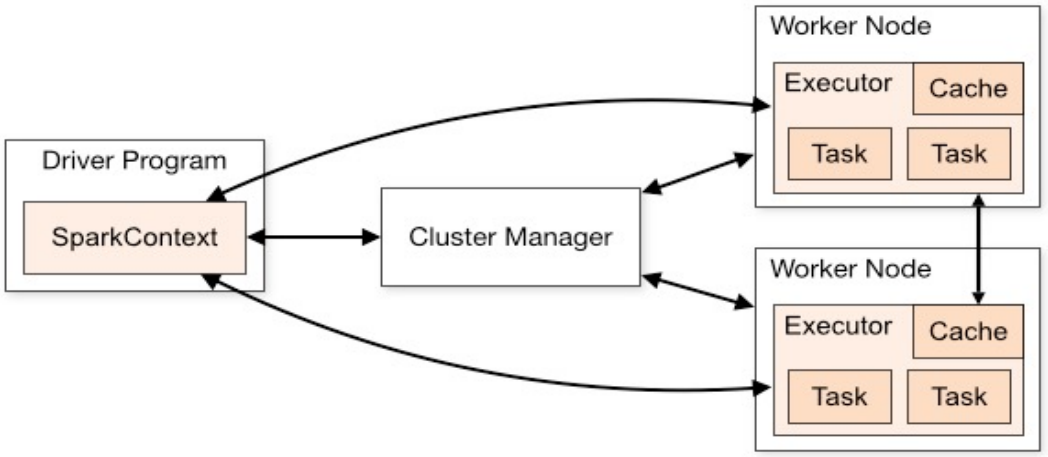
Spark Execution – Local mode

- 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

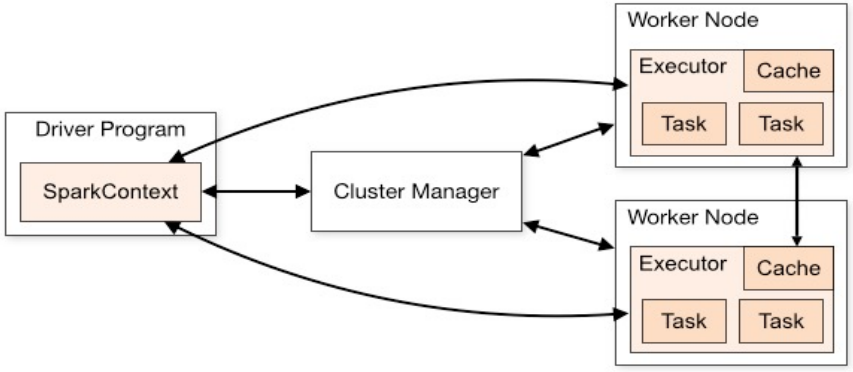
- **Spark Driver**:
  - Separate process to execute user applications
  - Creates SparkContext to schedule jobs execution and negotiate with cluster manager



- **Executors**:
  - Executors run tasks scheduled by the driver
  - Store computation results in memory, on disk or off-heap
  - Interact with storage systems
  - Executors manage computation as well as storage and caching

Spark Execution – Cluster mode

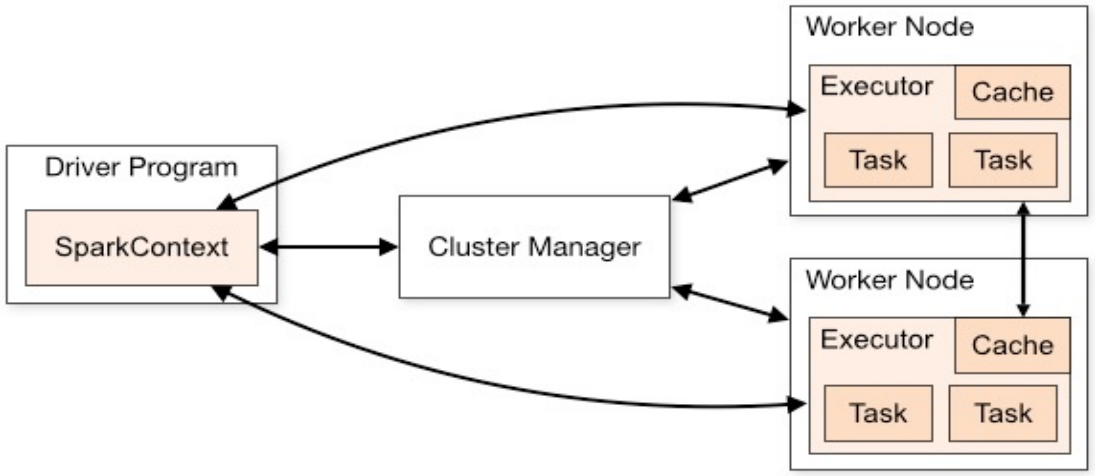
- **Spark Cluster**:
  - Mesos
  - YARN
  - Kubernetes
  - Spark Standalone –simplest way to deploy Spark on a private cluster



Spark is agnostic to the underlying cluster manager

Spark Execution – Cluster Mode

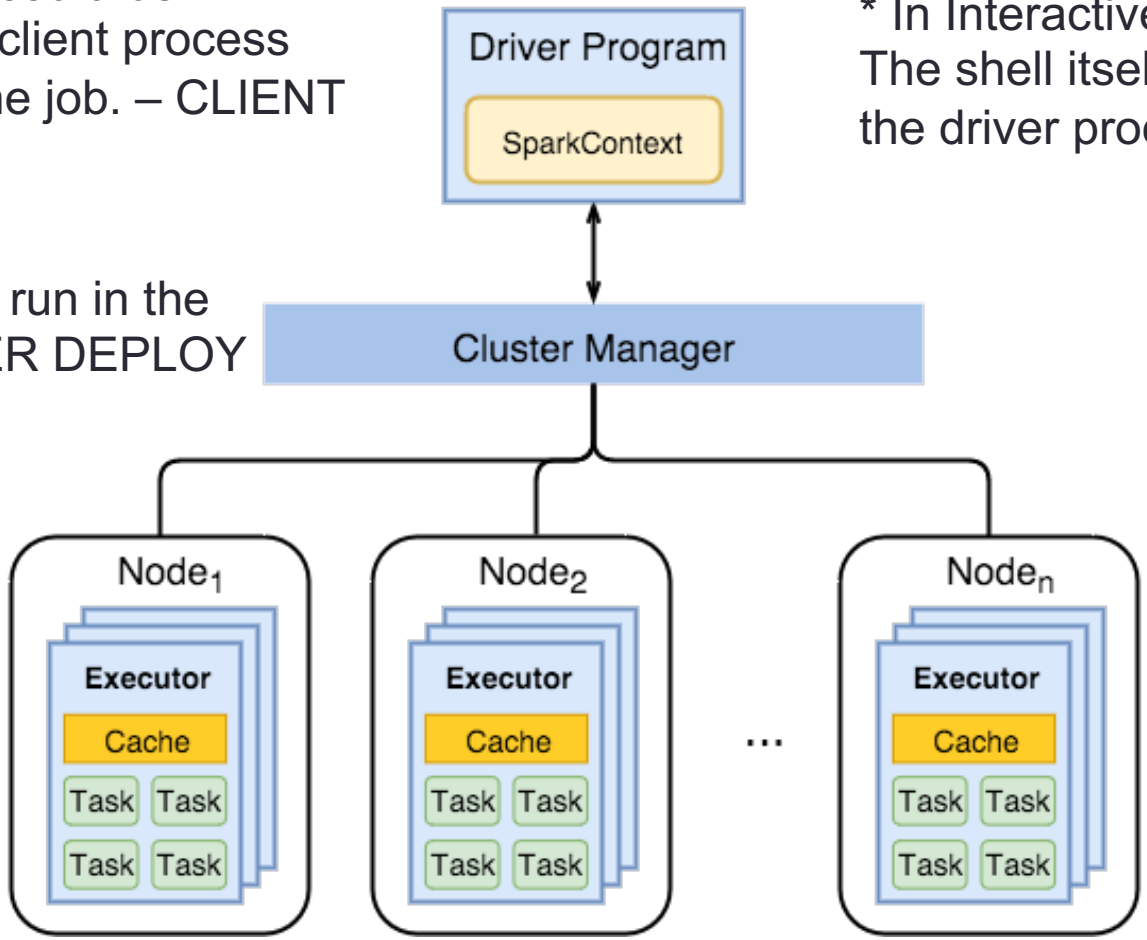
- The application code is sent from the *driver* to the *executors*, and the executors specify the context and the various *tasks* to be run.
- The *driver* program must listen for and accept incoming connections from its executors throughout its lifetime.



Spark Execution -- Cluster Mode

* Driver process could be the same as the client process used to initiate the job. – CLIENT DEPLOY MODE

* Driver process run in the luster - CLUSTER DEPLOY MODE



* In Interactive Mode, The shell itself is the driver process

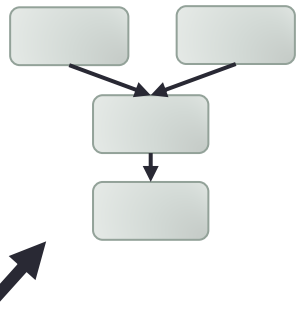
Job scheduling

RDD Objects

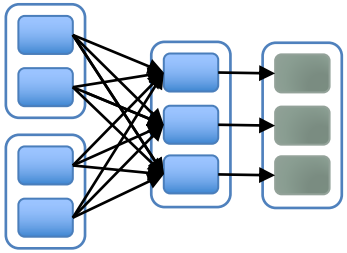
DAGScheduler

TaskScheduler

Worker

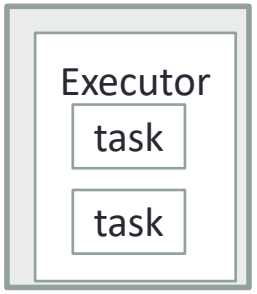


DAG



TaskSet

Tasks




```
rdd1.join(rdd2)
    .groupBy(…)
    .filter(…)
```

build operator DAG

split graph into *stages* of tasks

submit each stage as ready

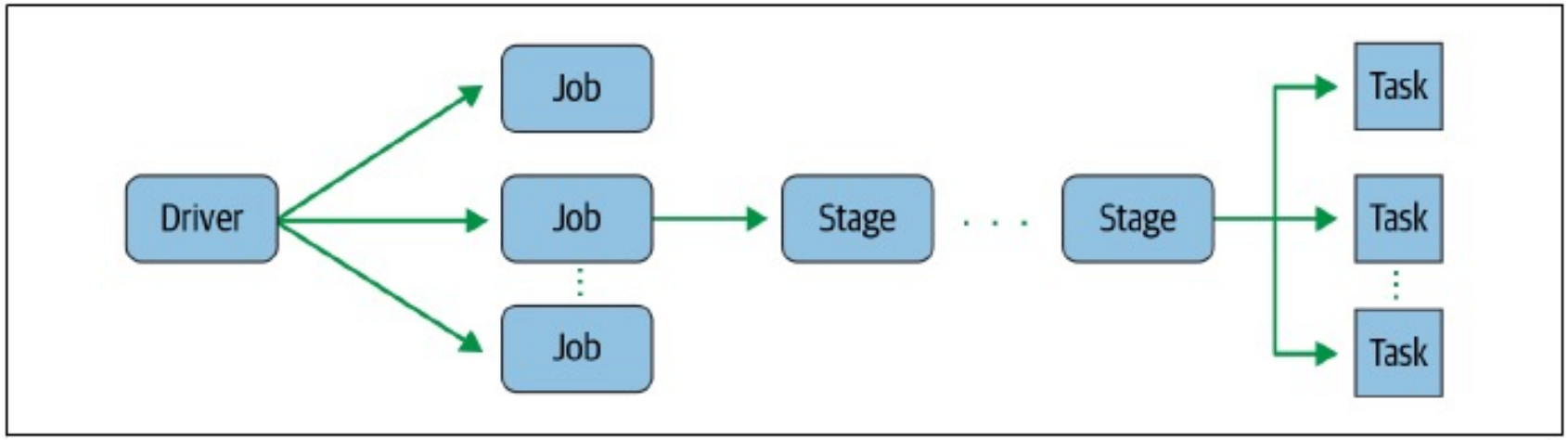
launch tasks via cluster manager

retry failed or straggling tasks

execute tasks

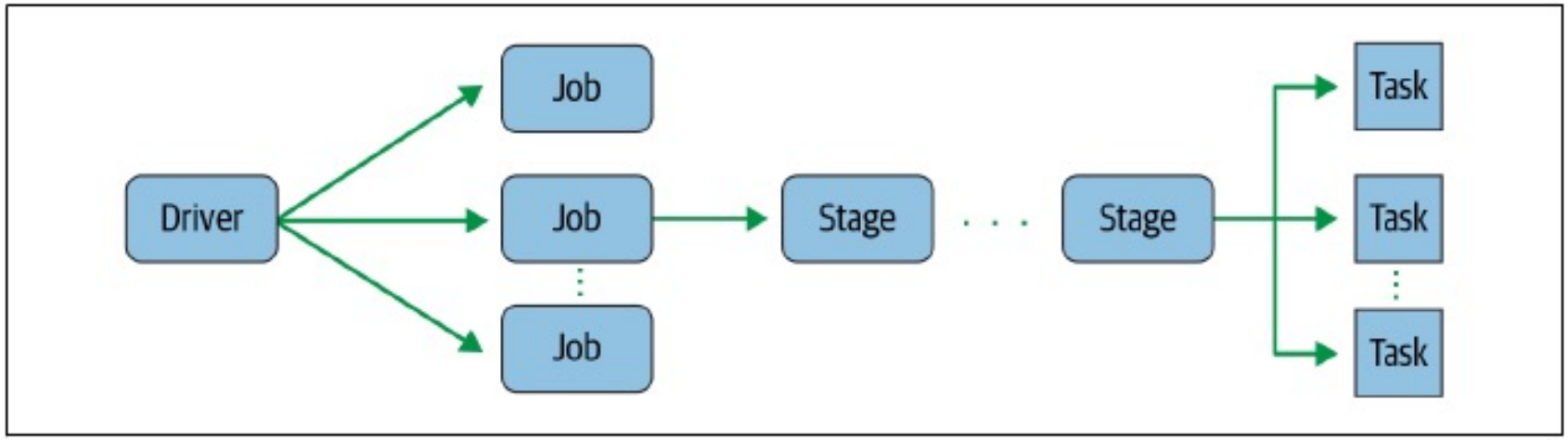
Spark Components

- Job: is work submitted to Spark**.** We have 1 job per action. Jobs are divided into **stages**
- Stage: is a sequence of **Tasks** that can all be run together, **in parallel**, without a shuffle
- Task: single unit of work that will be sent to a spark executor. A task compromise a series of instructions (.map, .filter) applied to a single partition. Every task in the stage executes the same set of instructions
- RDD: Parallel dataset with partitions



Spark Components

- A Spark application can have many jobs.
- A job can have many stages.
- A stage can have many tasks.
- A task executes a series of instructions



Partition

- **Partition** is a logical chunk of your RDD/Dataset.
- RDD is split into **Partitions** so that each **Executor core** can operate on a single part, enabling parallelization.
- A partition can be processed by a single **Executor core**.
- Example: If you have 4 partitions and you have 4 executor cores, you can process the tasks of a **Stage** in parallel, in a single pass.

Spark Application – wordcount.py

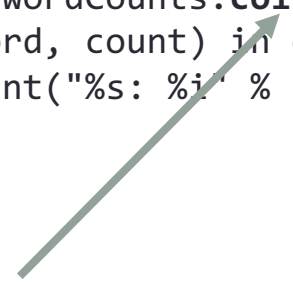
The application that we are going to create is a simple “wordcount”:

- Performs a ***textFile*** operation to read an input file
- ***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))
```



action

How many actions?

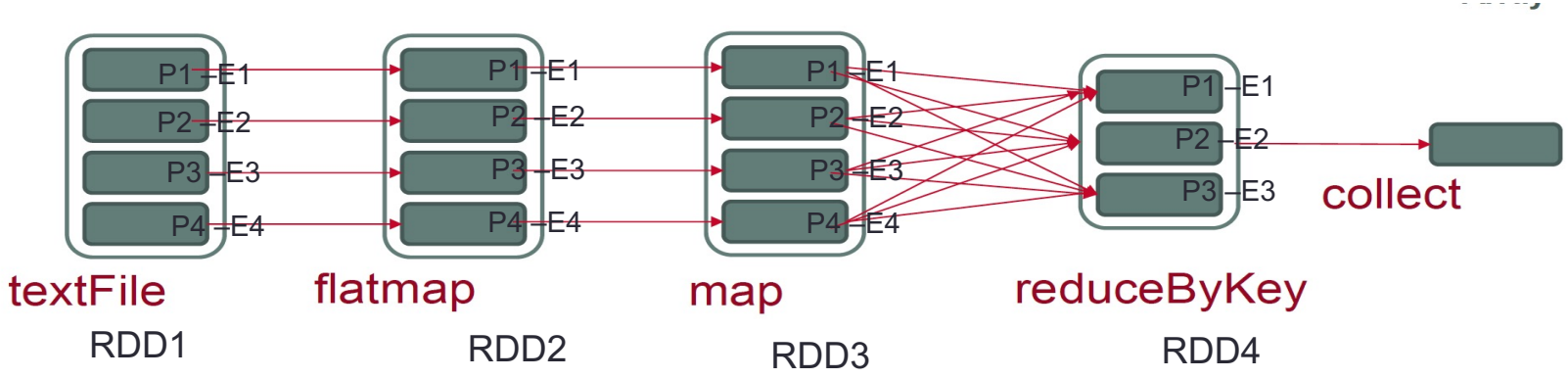
1 action → 1 Job

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))
```

1 JOB

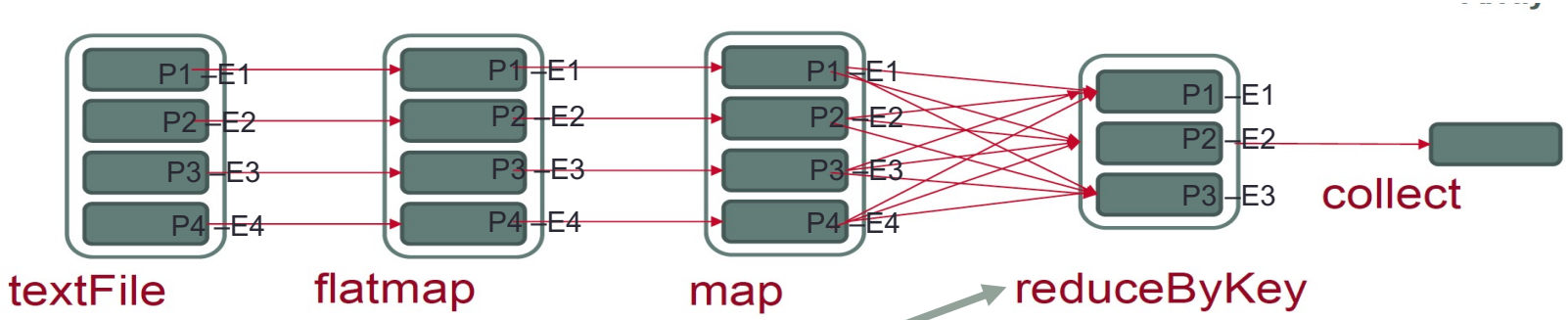


Spark Application – wordcount.py

```
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))
```

1 JOB



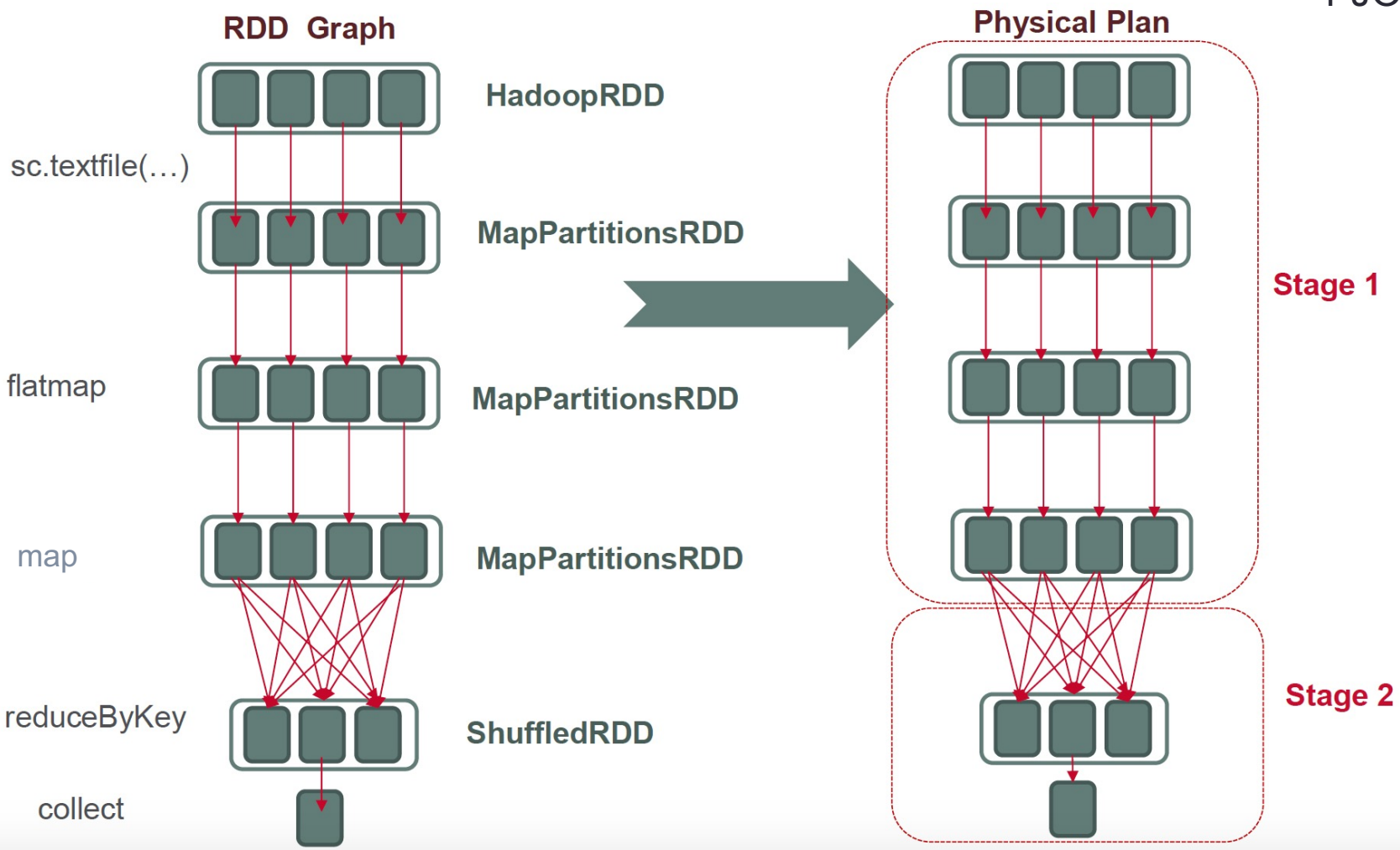
How many shuffles?

shuffle

1 shuffle → 2 stages

RDD DAG -> Physical Execution plan

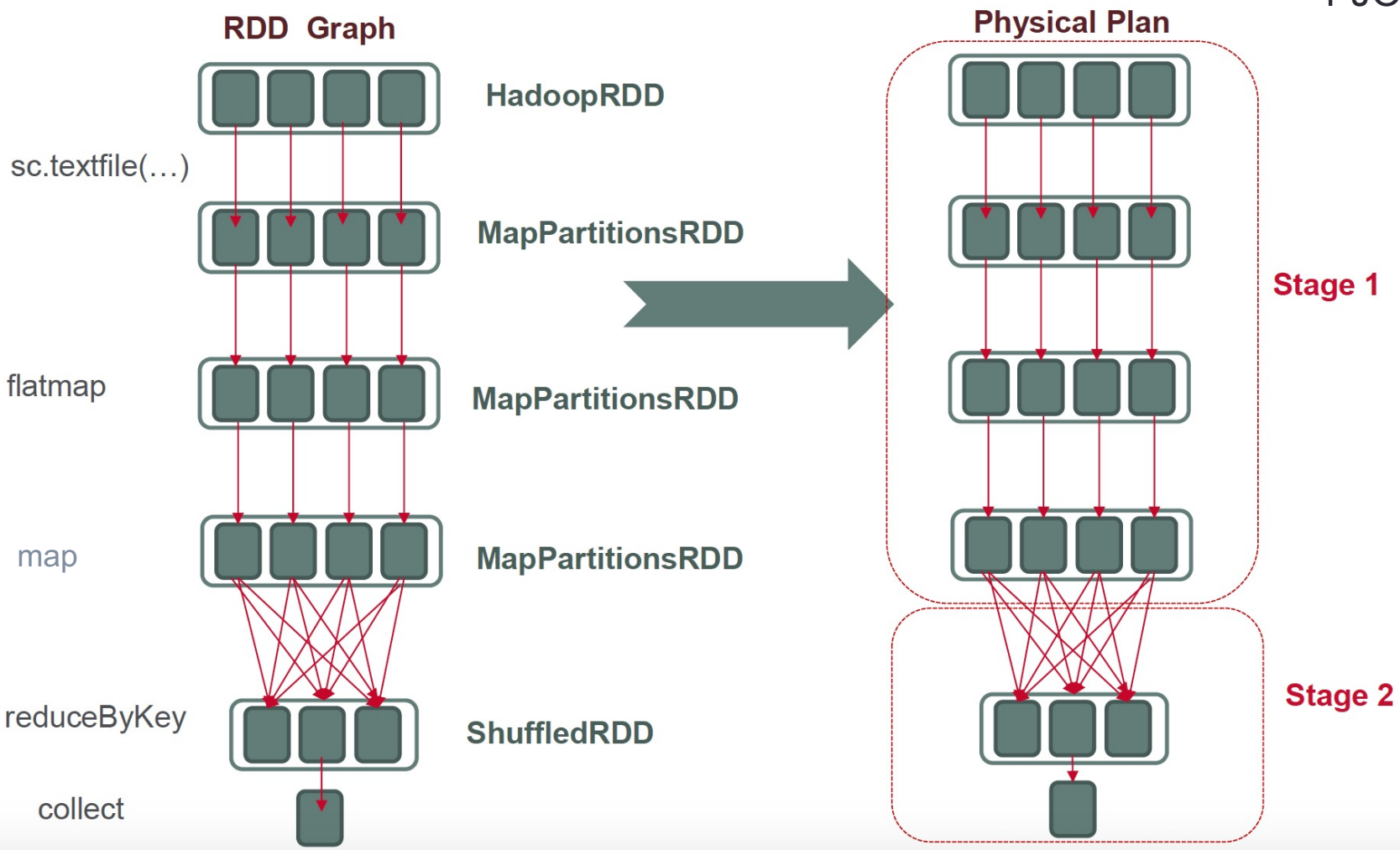
1 JOB



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

RDD DAG -> Physical Execution plan

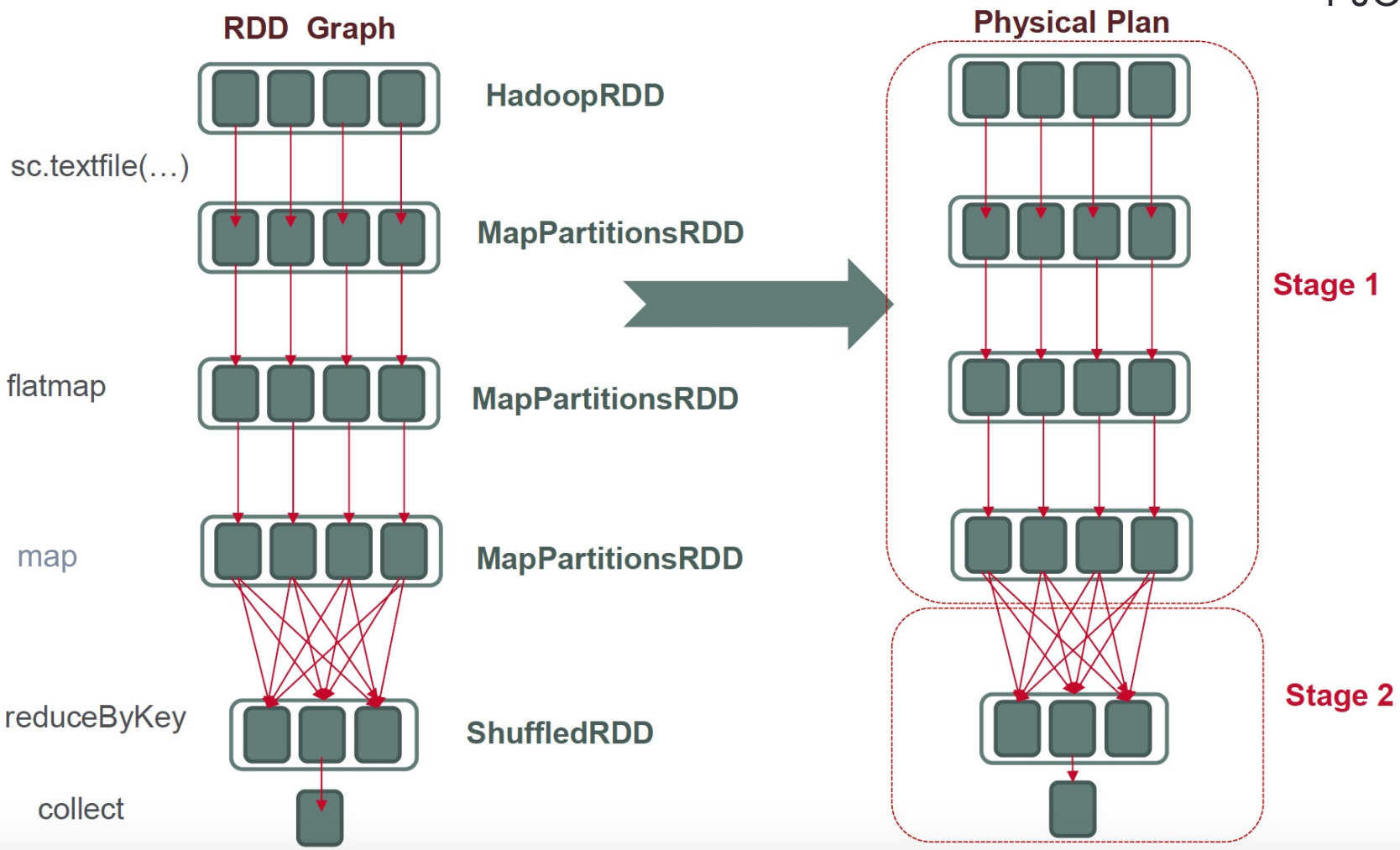
1 JOB



How many partitions in Stage 1 ? 4 Partitions 4 tasks

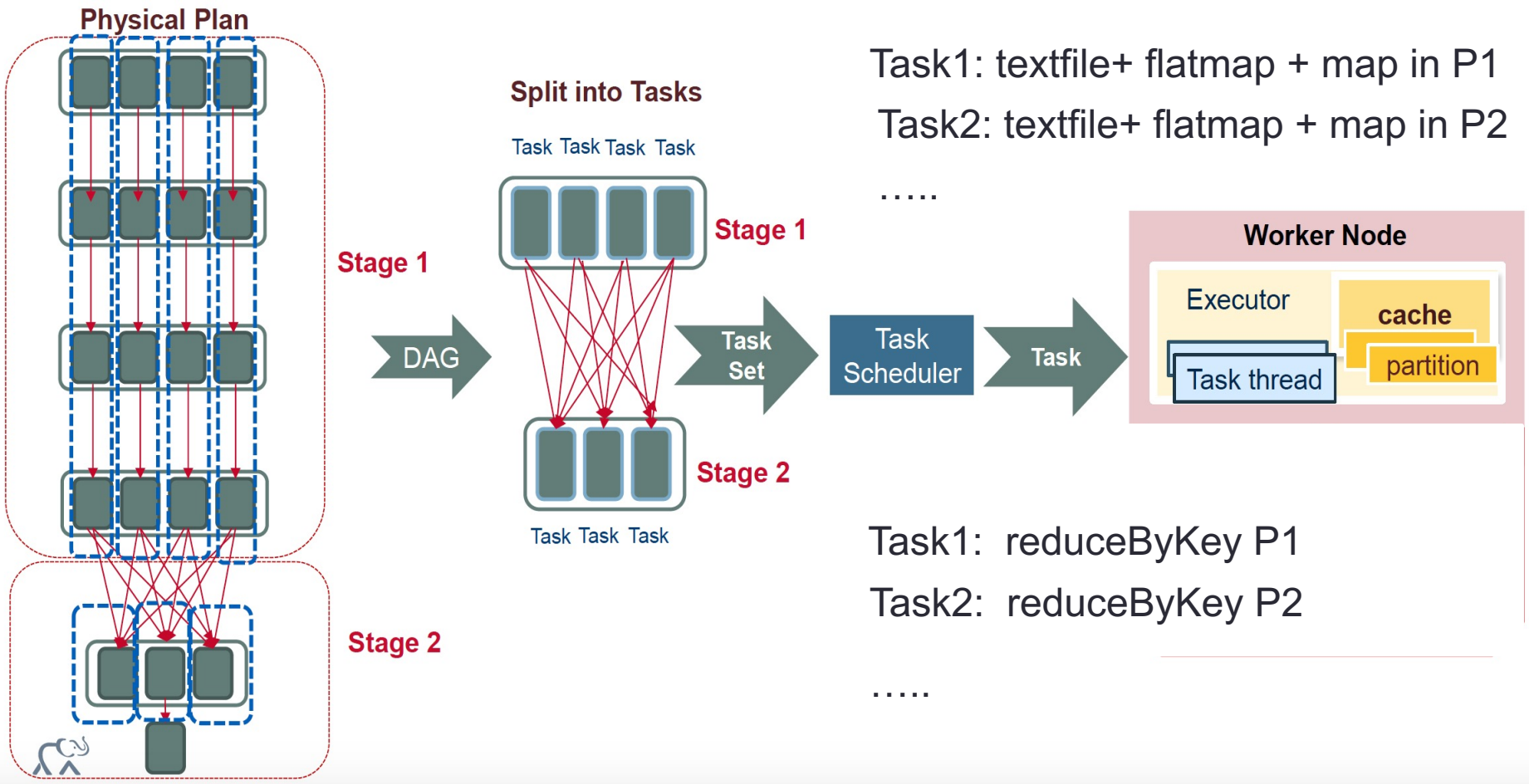
RDD DAG -> Physical Execution plan

1 JOB



How many partitions in Stage 2 ? 3 Partitions 3 tasks

Execution plan -> Stages and Tasks

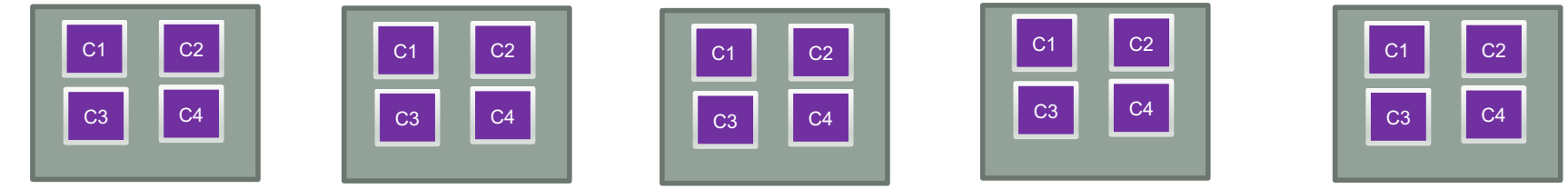


Each **Task** is executed as a single thread in an **Executor**!

Spark executors, tasks and partitions

- The number of tasks is given by the number of partitions of an RDD/DataFrame
- The number of tasks which an executor can process in parallel is given by its number of cores.
- How many task can be executed in parallel given the following example ?:

  1000 partitions and 5 executors which 4 cores each



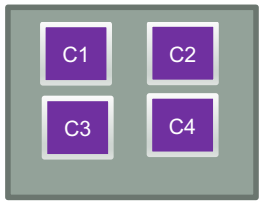
Spark executors, tasks and partitions

•The number of tasks is given by the number of partitions of anRDD/DataFrame

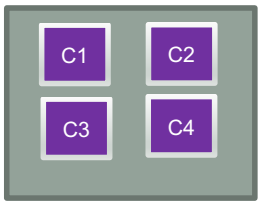
•The number of tasks which an executor can process in parallel is given by its number of cores.

•How many task can be executed in parallel given the following example ?:

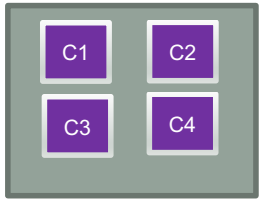
1000 partitions and 5 executors which 4 cores each



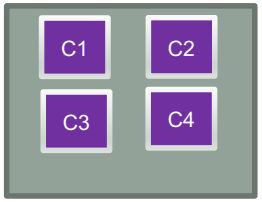
Executor 1
- 200 partitions per Ex.
- 50 partitions per Core



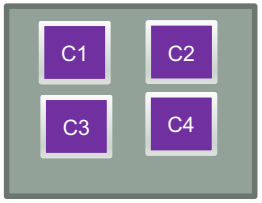
Executor 2
- 200 partitions per Ex.
- 50 partitions per Core



Executor 3
- 200 partitions per Ex.
- 50 partitions per Core



Executor 4
- 200 partitions per Ex.
- 50 partitions per Core



Executor 5
- 200 partitions per Ex.
- 50 partitions per Core

1 Executor can run 4 tasks in parallel == Num of cores.
4 task per executor x 5 executors = 20 Tasks in parallel.

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**.

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/rosafilgueira/Seminar_MUIA/blob/main/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).

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

Running jobs in Cirrus

- PBSPro 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
```

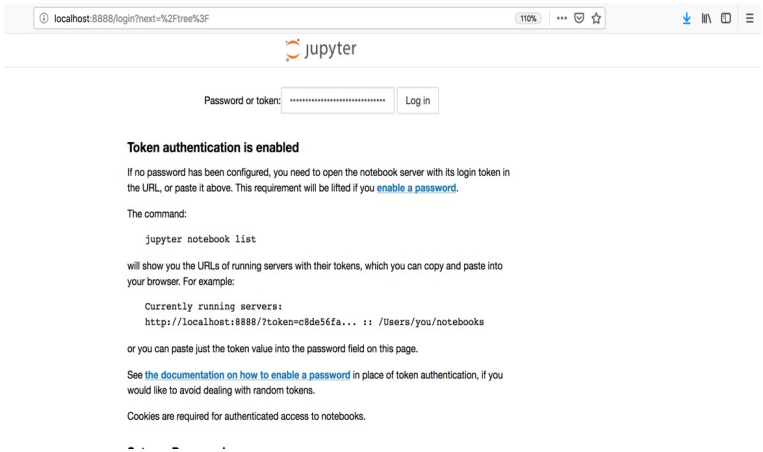
  - Your session will end:
    - It hits the requested **walltime**
    - Typing **exit command** within the session

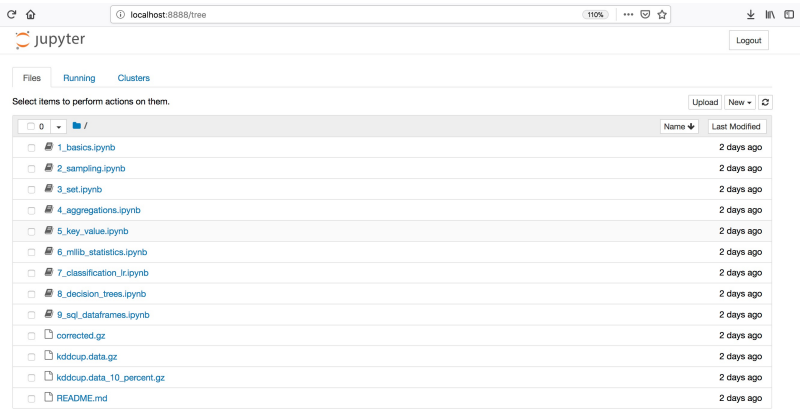
https://cirrus.readthedocs.io/en/latest/user-guide/batch.html#interactive-jobs

Jupyter notebooks

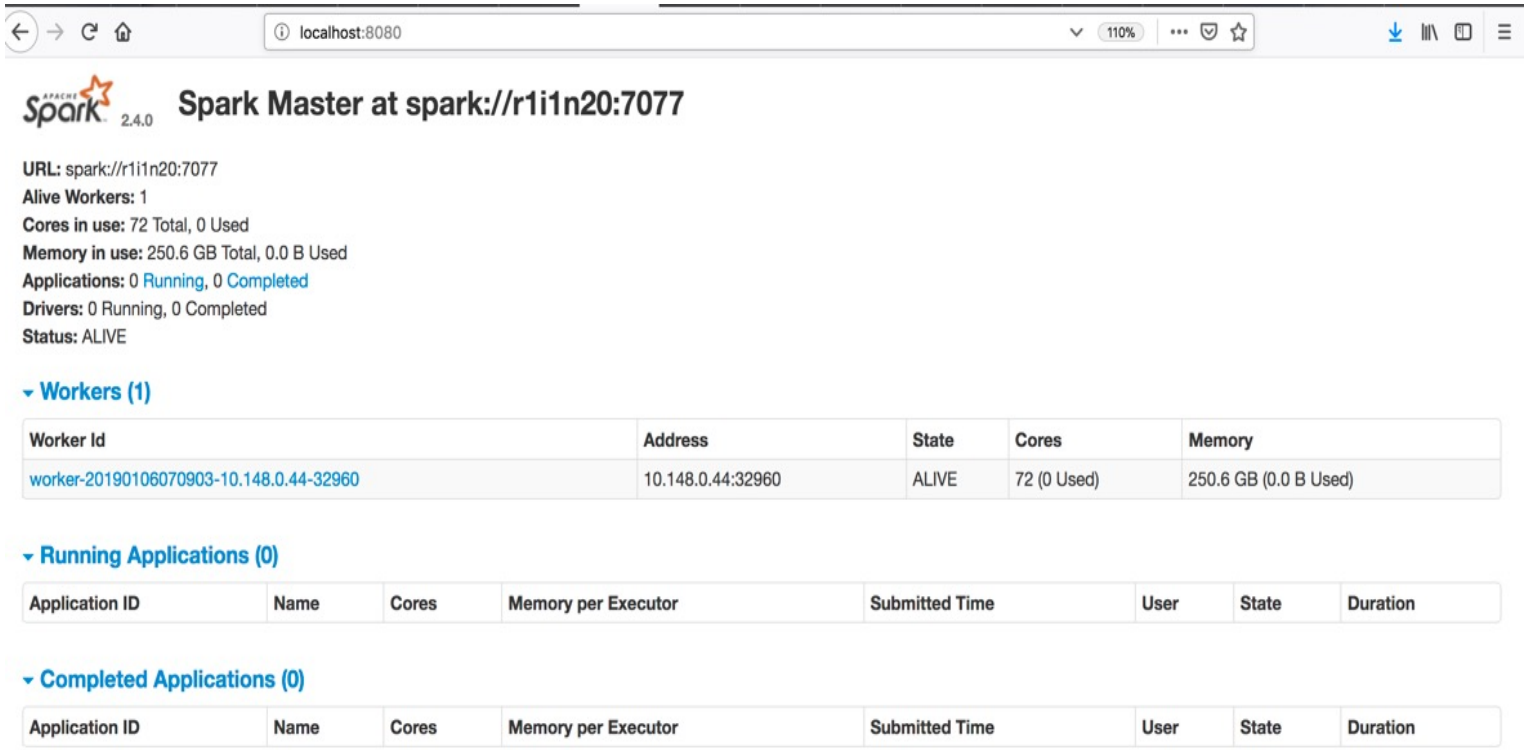
- Start the jupyter server:
  ./start_Jupyter_local.sh <master> → It will give you a token, like this one:
  http://0.0.0.0:8888/?token=2d5e554b2397355c334b8c3367503b06c4f6f95a26151795

- 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





Master Spark UI

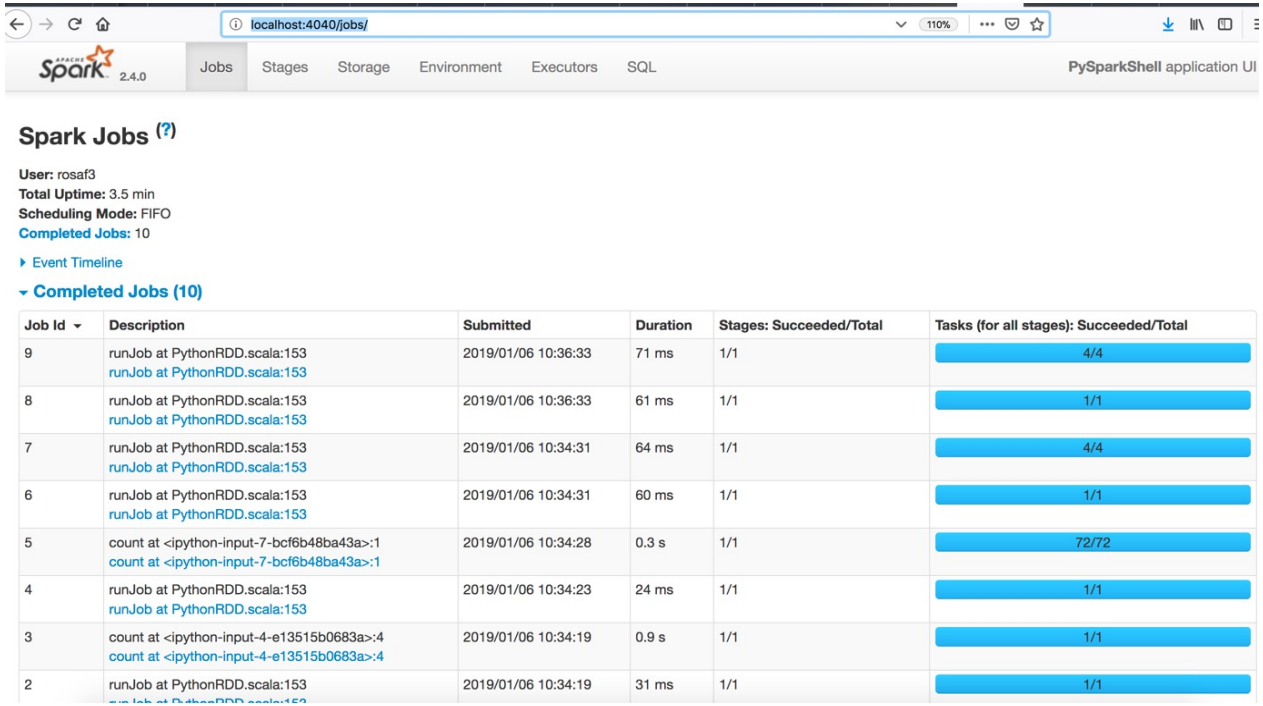


The spark setup can be monetarised via the **Master's web UI**

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

Got to a Web browser and type `localhost:8080`

Driver Spark UI



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, Java 10 or Java 11**
- 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/rosafilgueira/Seminar_MUIA.git`
- `>> cd walkthrough_examples`
- `>> export SPARK_HOME=[INSTALLATION_PATH]/spark-3.2.1-bin-hadoop3.2/`
- **`>> export PYSPARK_DRIVER_PYTHON=jupyter`**
- **`>> export PYSPARK_DRIVER_PYTHON_OPTS='notebook'`**
- `>> $SPARK_HOME/bin/pyspark`