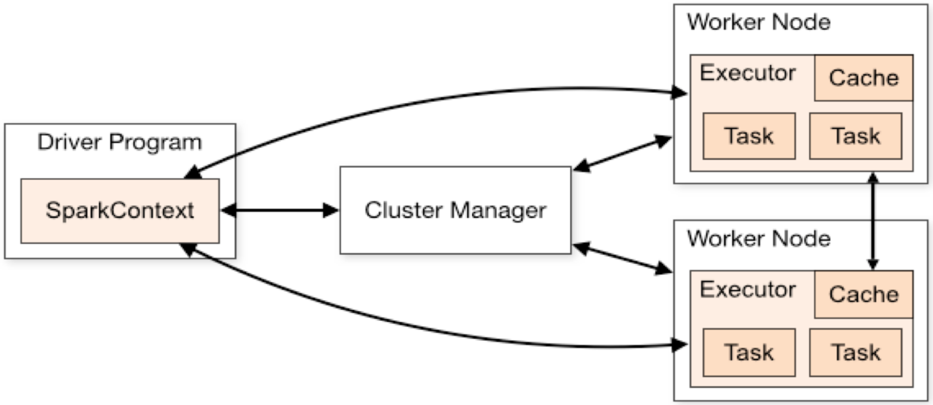
SPARK CLUSTER OVERVIEW

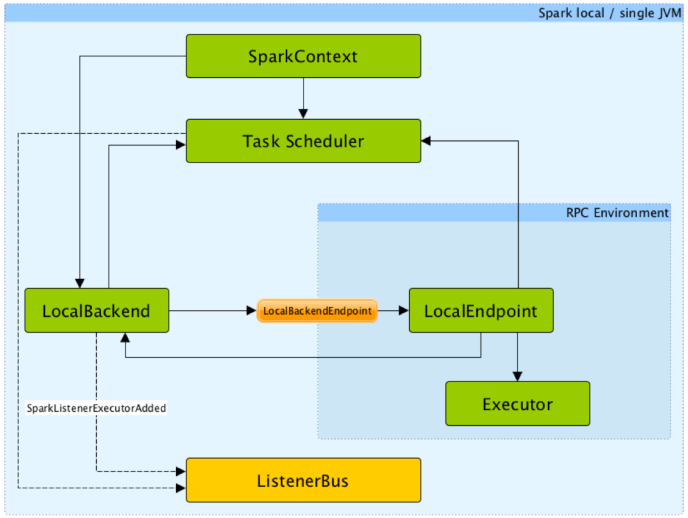
Spark Execution modes

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

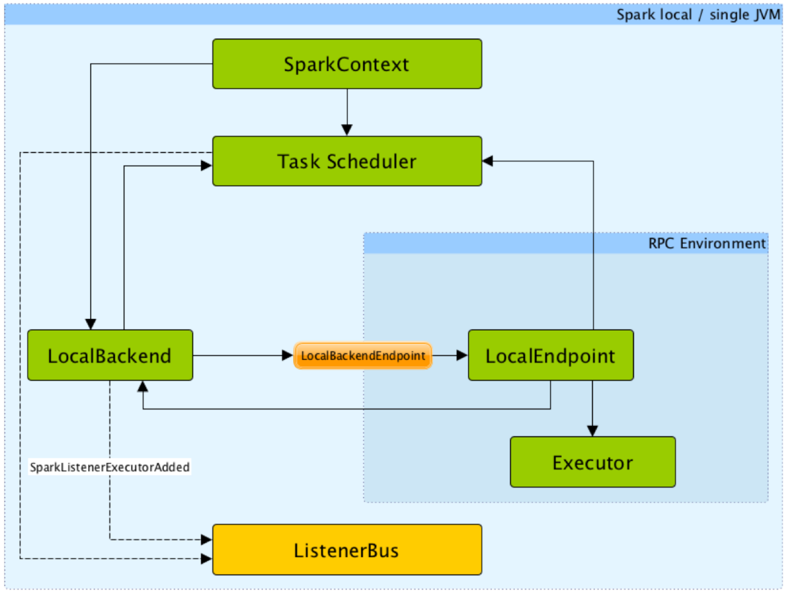


- Local mode



Spark Execution – Local mode

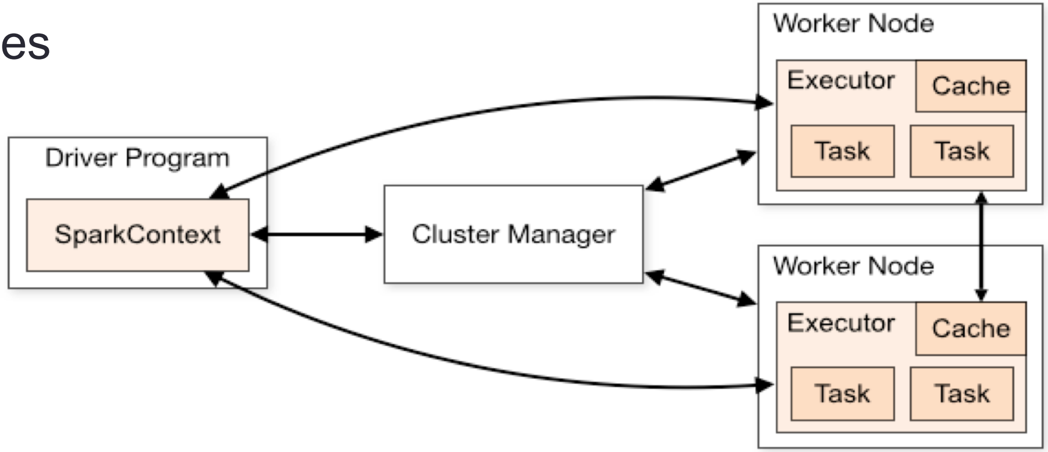
- In this non-distributed single-JVM deployment mode.
- Spark spawns all the execution components - driver, executor, LocalSchedulerBackend, and master - in the same single JVM.
- The default parallelism is the number of threads as specified in the master URL.
- This is the only mode where a driver is used for execution



Spark Execution – Cluster mode

It currently provides several options:

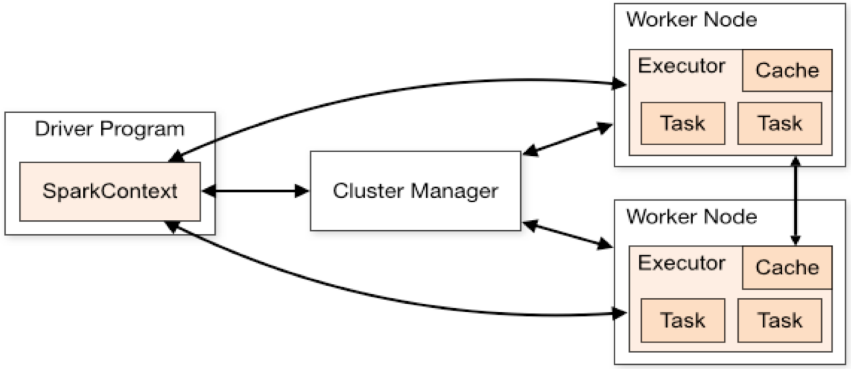
- **Standalone Deploy Mode**: simplest way to deploy Spark on a private cluster
- Apache Mesos
- Hadoop YARN
- Kubernetes



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*.

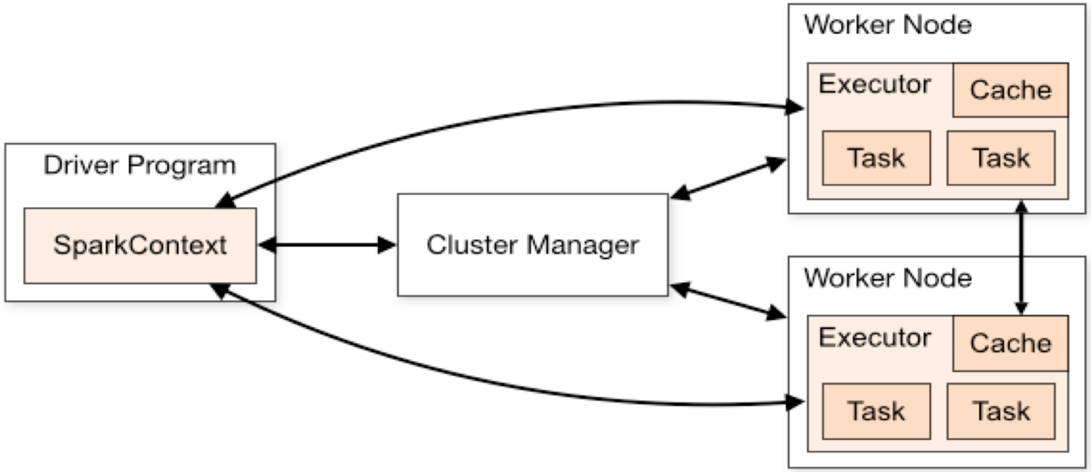


- Each *worker* in the cluster is managed by an *executor.*
- The *executor* manages computation as well as storage and caching on each machine.

(*) *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

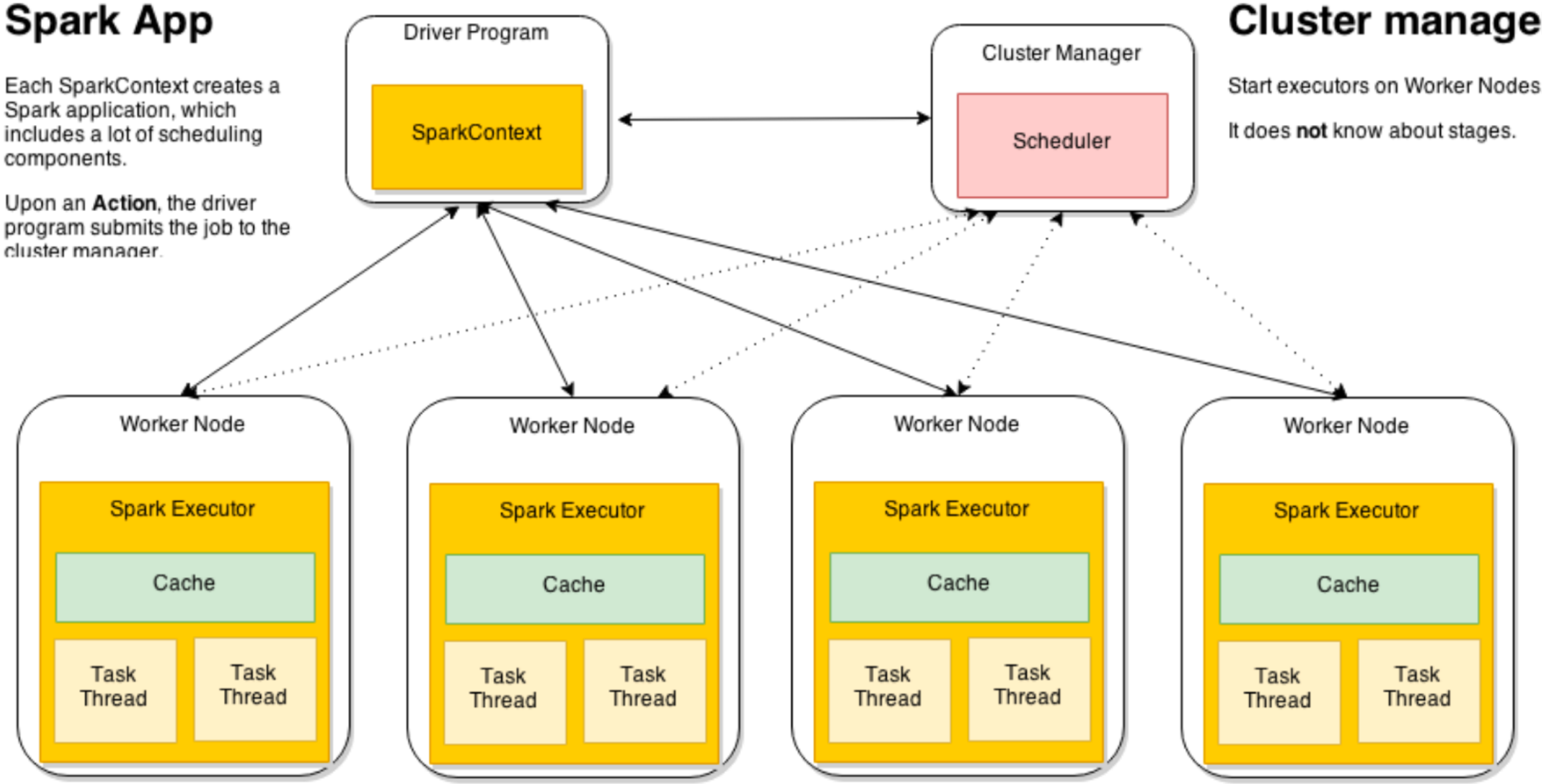
Start executors on Worker Nodes.

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

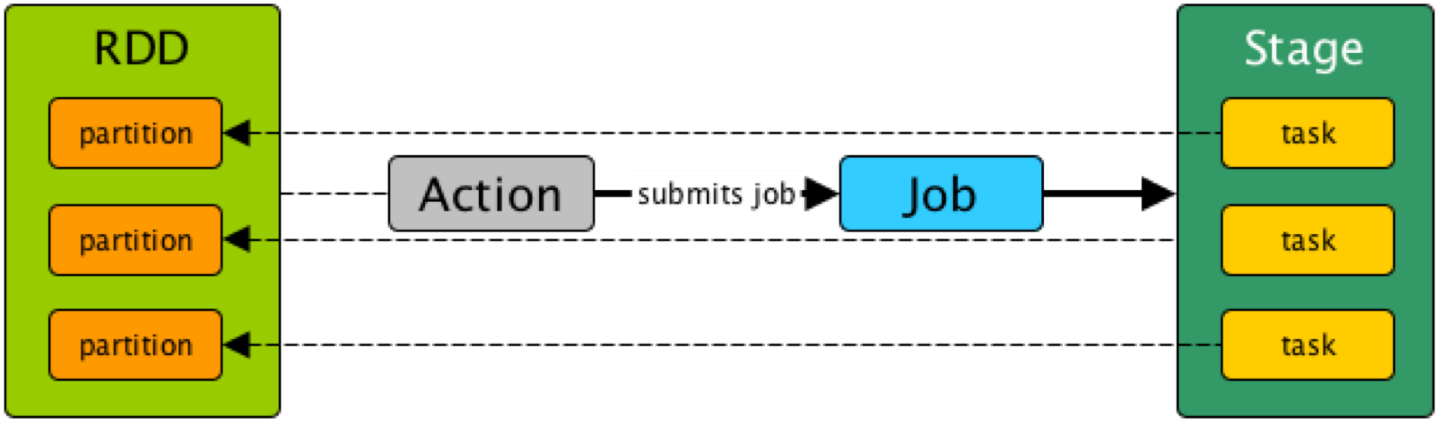
For standalone clusters supports two deploy modes.
They distinguish where the *driver* process runs:

- *Client mode*: the *driver* is launched in the *same process as the client* that submits the application.

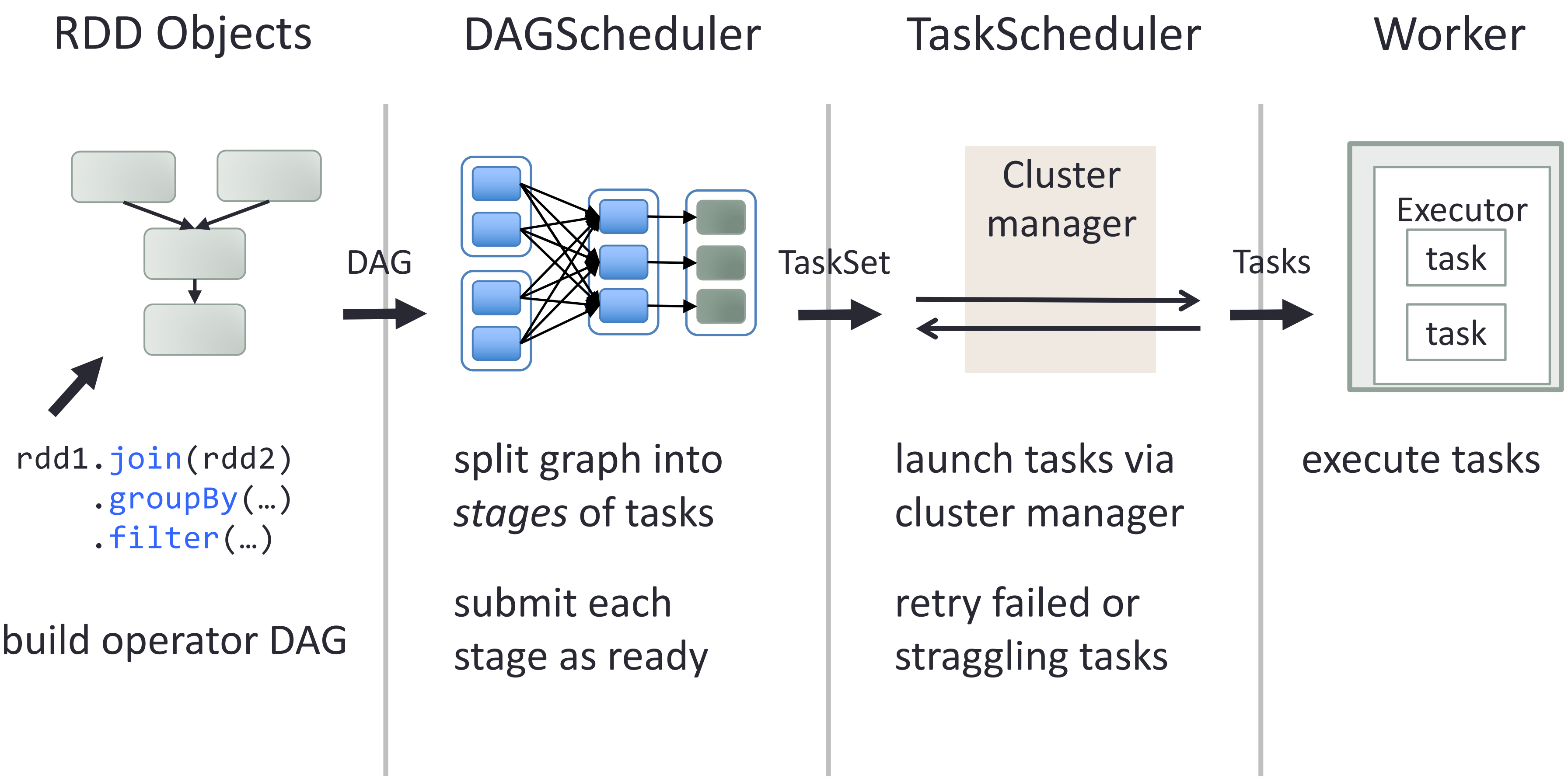
- *Cluster mode*: the *driver* is launched from *one of the Worker processes* inside the cluster.
  - The client process exits as soon as it fulfils its responsibility of submitting the application without waiting for the application to finish.

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
- RDD: Parallel dataset with partitions
- DAG: Logical Graph of RDD operations



Job scheduling



RDD Objects

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

build operator DAG

DAG

DAGScheduler

split graph into *stages* of tasks

submit each stage as ready

TaskSet

TaskScheduler

Cluster manager

launch tasks via cluster manager

retry failed or straggling tasks

Tasks

Worker

Executor

task

task

execute tasks

Spark Application – wordcount.py

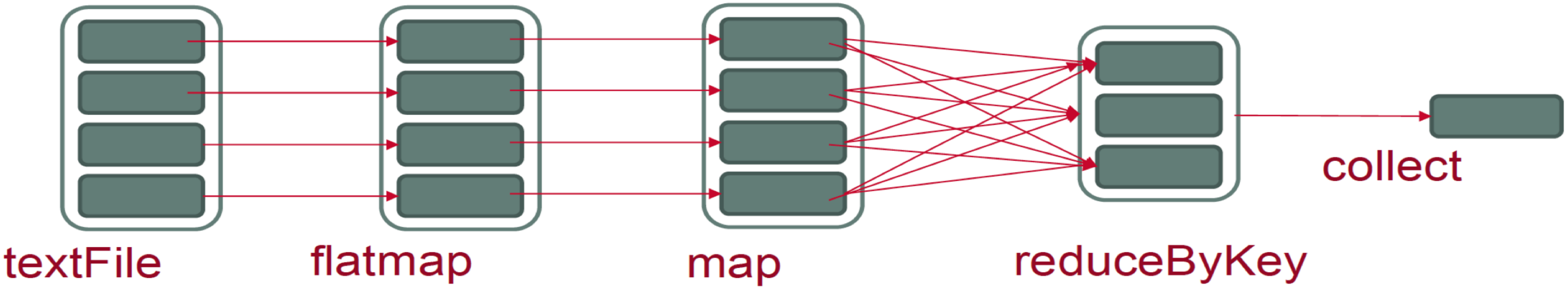
The application that we are going to create is a simple “wordcount”:

- 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 for each word.

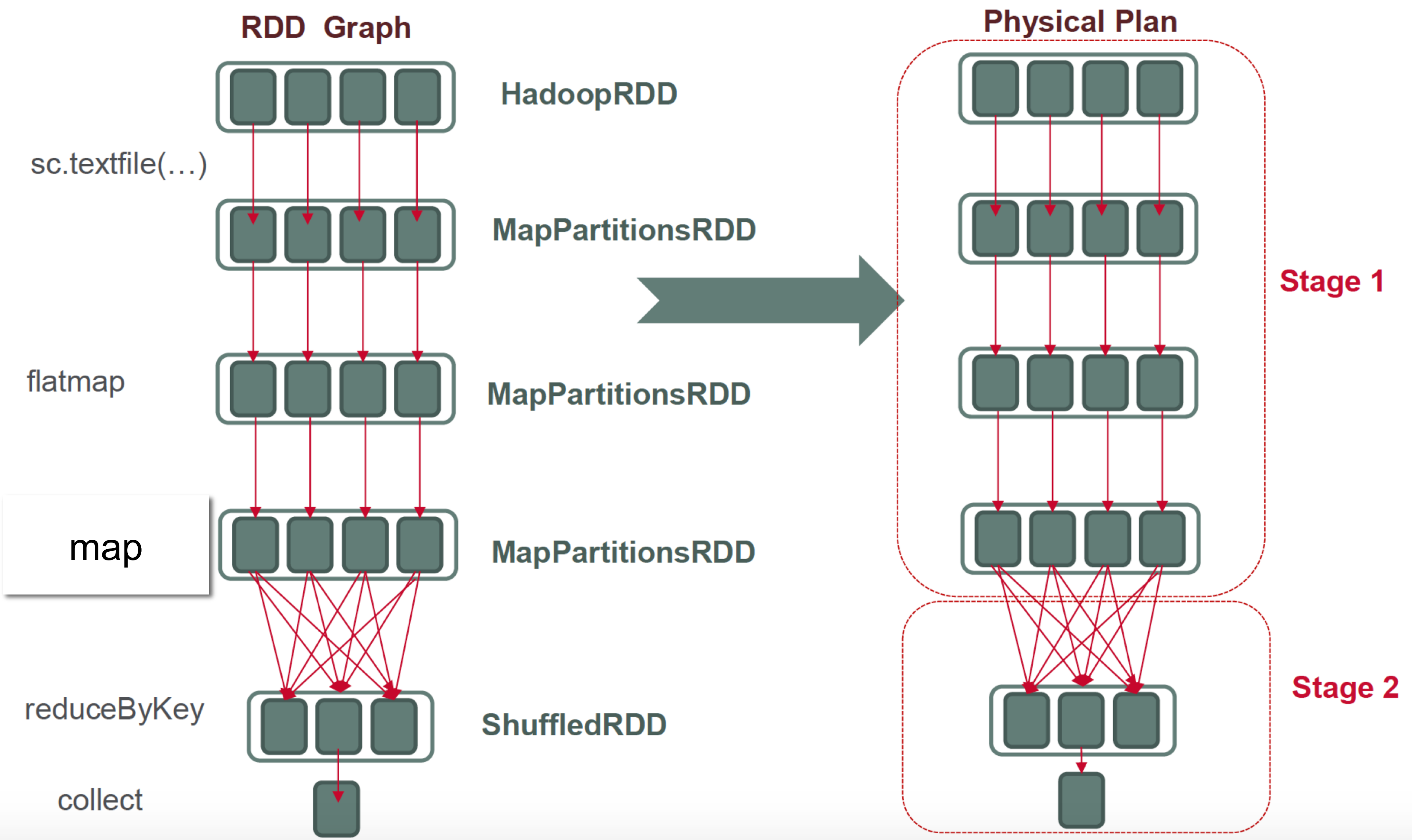
Spark Application – wordcount.py

```
import sys
from pyspark import SparkContext, SparkConf


if __name__ == "__main__":
    conf = SparkConf().setAppName("Spark Count")
    sc = SparkContext(conf=conf)
    logFile = "../spark-2.4.0-bin-hadoop2.7/README.md"
    textFile = sc.textFile(logFile)
    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))
```

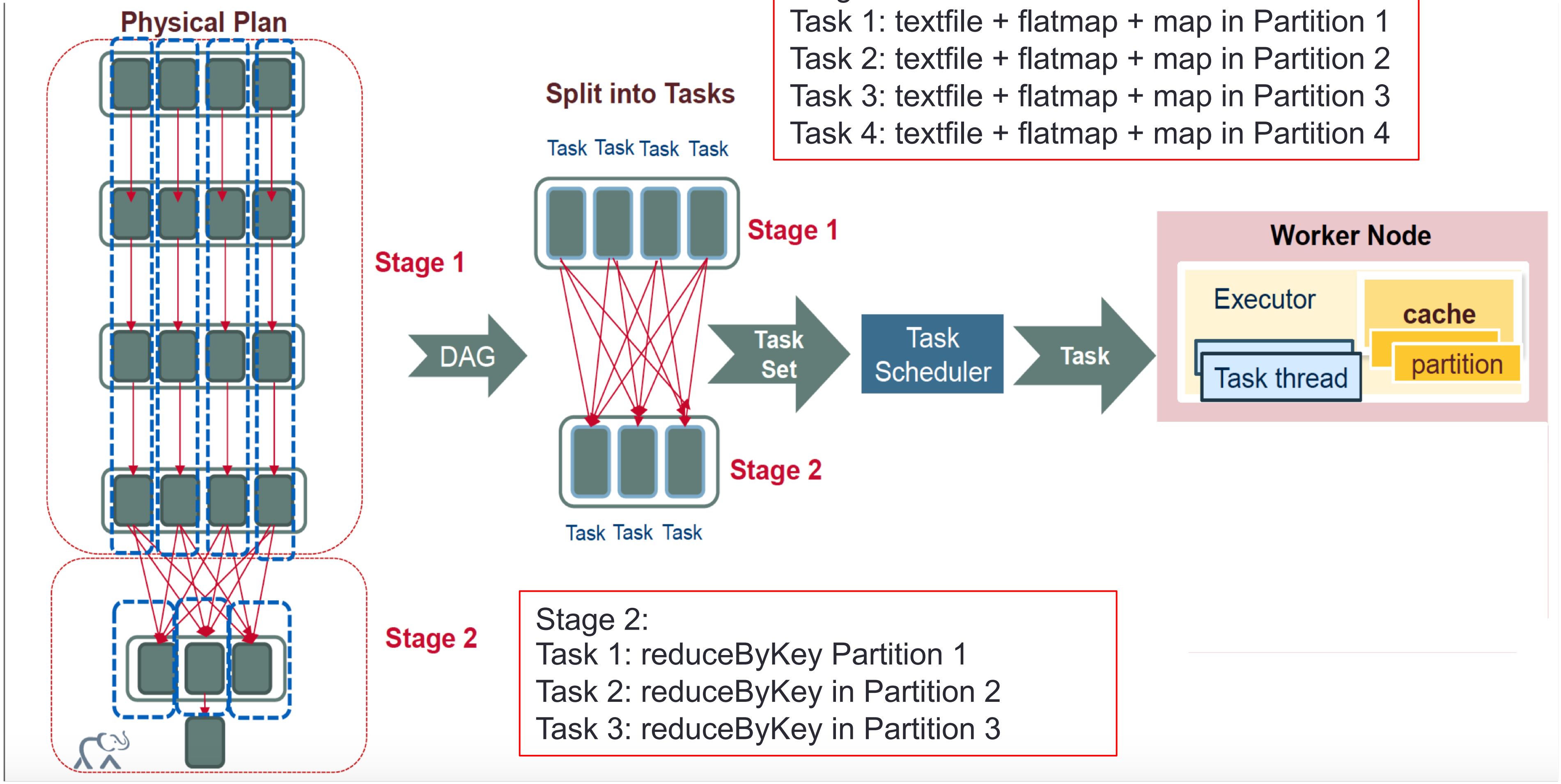


RDD DAG -> Physical Execution plan



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

Execution plan -> Stages and Tasks



Stage 1:
Task 1: textfile + flatmap + map in Partition 1
Task 2: textfile + flatmap + map in Partition 2
Task 3: textfile + flatmap + map in Partition 3
Task 4: textfile + flatmap + map in Partition 4

Stage 2:
Task 1: reduceByKey Partition 1
Task 2: reduceByKey in Partition 2
Task 3: reduceByKey in Partition 3

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

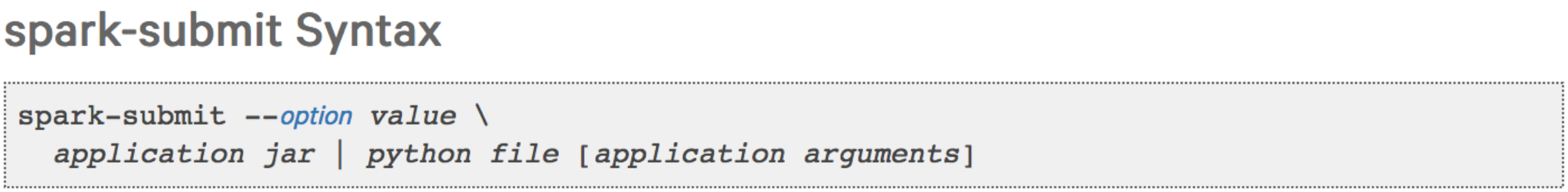
We are going to use Jupyter Notebooks for running our walkthroughs & lab exercises.

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

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/EPCCed/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

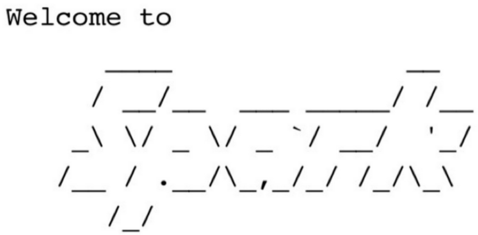
- master – Determines how to run the job:
  - spark://r1i2n5:7077
  - local
- driver-memory
  - amount memory available for the driver process.
- executor-memory
  - amount of memory allocated to the executor process
- executor-cores
  - total number of cores allocated to the executor process
- total-executor-cores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

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

Running notebooks in your laptop

- **Prerequisites: Anaconda, Python3**
- Get Spark from the [downloads page](https://blog.sicara.com/get-started-pyspark-jupyter-guide-tutorial-ae2fe84f594f) 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 PYSPARK_DRIVER_PYTHON=jupyter`**
- **`>> export PYSPARK_DRIVER_PYTHON_OPTS='notebook'`**
- `>> pyspark`

vim ~/.bash_profile

Cirrus

- High-performance computing cluster
- One of the EPSRC Tier-2 National HPC Services.
- 280 nodes: 36 Intel Xeon CPUs, hyper threading, 256GB
- 406 TB of storage- Lustre
- Link: http://www.cirrus.ac.uk/
-