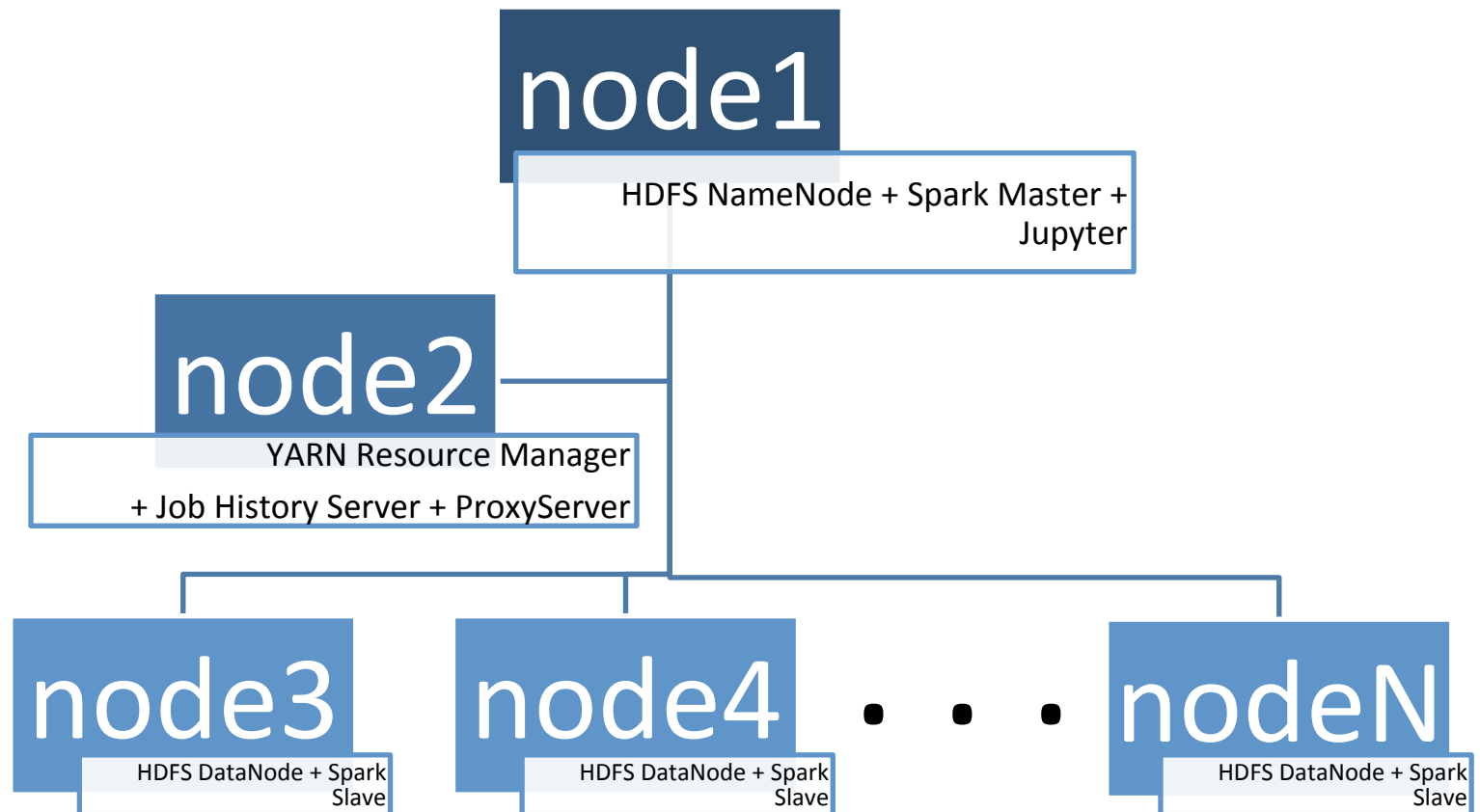


Spark Cluster Overview

Rosa Filgueira

Spark cluster configuration

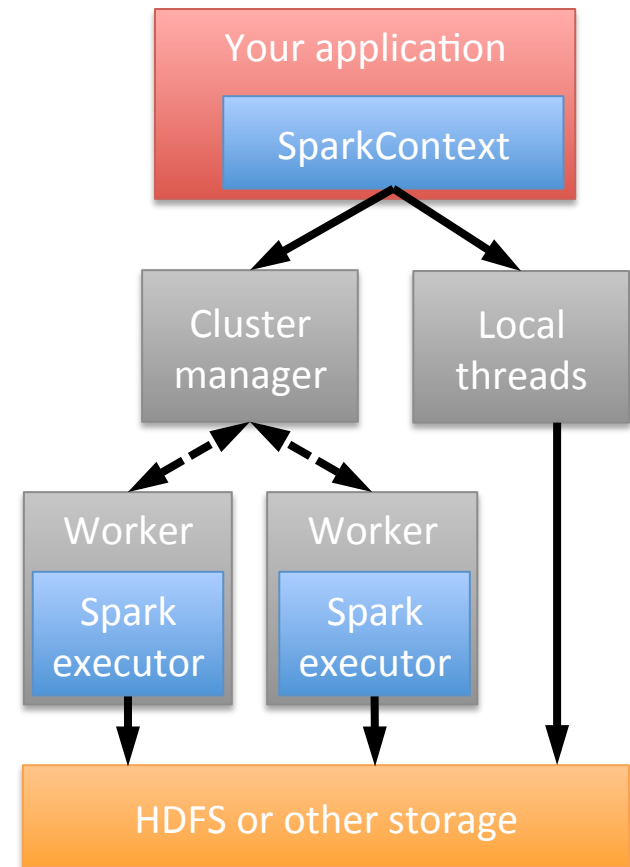


Spark cluster configuration

- Software:
 - Apache Spark-2.1.1
 - Hadoop-2.8
 - Anaconda3-4.4
 - 64-bit CentOS6.5
- 4 VMs with 2GB Memory each

Software Components

- Spark runs as a library in your program (one instance per app)
- Runs tasks locally or on a cluster
 - Standalone deploy cluster, YARN
- Accesses storage via Hadoop InputFormat API
 - Can use HBase, HDFS, S3, ...



Monitoring the cluster services

- HDFS NameNode

<http://10.211.55.101:50070/dfshealth.html>

- Resource Manager

<http://10.211.55.102:8088/cluster>

- Spark History

<http://10.211.55.101:18080>

- Jupyter Notebook

<http://10.211.55.101:8888>

HDFS NameNode UI

[Hadoop](#)[Overview](#)[Datanodes](#)[Datanode Volume Failures](#)[Snapshot](#)[Startup Progress](#)[Utilities](#) ▾

Overview 'node1:8020' (active)

Started:	Mon Jun 12 15:54:46 +0100 2017
Version:	2.8.0, r91f2b7a13d1e97be65db92ddabc627cc29ac0009
Compiled:	Fri Mar 17 04:12:00 +0000 2017 by jdu from branch-2.8.0
Cluster ID:	CID-76d9e1ed-7d16-47c6-848a-0f7ab6862aa0
Block Pool ID:	BP-735560476-10.211.55.101-1496928722806

Summary

HDFS NameNode UI

10.211.55.101:50070/explorer.html#/user/root

Hadoop Overview Datanodes Datanode Volume Failures Snapshot Startup Progress Utilities ▾

Browse Directory

Show entries Search:

Permission	Owner	Group	Size	Last Modified	Replication	Block Size	Name	
-rw-r--r--	root	supergroup	3.73 KB	Jun 06 12:43	3	128 MB	README.md	
drwxr-xr-x	root	supergroup	0 B	Jun 06 12:35	0	0 B	.sparkStaging	

Showing 1 to 2 of 2 entries Previous **1** Next


Hadoop, 2017.

Resource Manager UI

10.211.55.102:8088/cluster

Big data

☆ 📄 ⬇️ 🏠 📌 🐛



All Applications

Cluster

About

Nodes

Node Labels

Applications

NEW

NEW SAVING

SUBMITTED

ACCEPTED

RUNNING

FINISHED

FAILED

KILLED

Scheduler

Tools

Cluster Metrics

Apps Submitted	Apps Pending	Apps Running	Apps Completed	Containers Running	Memory
5	5	0	0	0	0 B

Cluster Nodes Metrics

Active Nodes	Decommissioning Nodes	Decommissioned Nodes
0	0	0

Scheduler Metrics

Scheduler Type	Scheduling Resource Type	Minimum Allocation
Capacity Scheduler	[MEMORY]	<memory:1024, vCores:1>


Show 20 entries

ID	User	Name	Application Type	Queue	Application Priority	StartTime	Finish
application_1497279303970_0007	root	QuasiMonteCarlo	MAPREDUCE	default	0	Mon Jun 12 16:47:32 +0100 2017	N/A
application_1497279303970_0006	root	org.apache.spark.examples.SparkPi	SPARK	default	0	Mon Jun 12 16:39:27 +0100 2017	N/A

Spark History UI

10.211.55.101:18080

spark anaconda vagrant

 **History Server**

Event log directory: hdfs://node1/tmp/spark-events

Last updated: 06/06/2017, 13:50:29

Search:

App ID	App Name	Started	Completed	Duration	Spark User	Last Updated	Event Log
local-1496753398812	PySparkShell	2017-06-06 12:49:56	2017-06-06 12:50:29	33 s	root	2017-06-06 12:50:29	Download
app-20170606124543-0005	PySparkShell	2017-06-06 12:45:41	2017-06-06 12:47:03	1.4 min	root	2017-06-06 12:47:03	Download
app-20170606123511-0004	PySparkShell	2017-06-06 12:35:09	2017-06-06 12:36:36	1.4 min	root	2017-06-06 12:36:36	Download
app-20170606122347-0003	PySparkShell	2017-06-06 12:23:44	2017-06-06 12:34:18	11 min	root	2017-06-06 12:34:18	Download
app-20170606120721-0002	PySparkShell	2017-06-06 12:07:19	2017-06-06 12:12:10	4.8 min	root	2017-06-06 12:12:10	Download
local-1496749732745	PySparkShell	2017-06-06 11:48:51	2017-06-06 11:54:08	5.3 min	root	2017-06-06 11:54:08	Download
local-1496749577058	PySparkShell	2017-06-06 11:46:15	2017-06-06 11:46:39	24 s	root	2017-06-06 11:46:39	Download
local-1496749368028	PySparkShell	2017-06-06 11:42:46	2017-06-06 11:43:14	28 s	root	2017-06-06 11:43:14	Download
local-1496749298022	PySparkShell	2017-06-06 11:41:36	2017-06-06 11:42:00	24 s	root	2017-06-06 11:42:00	Download
app-20170606114015-0001	PySparkShell	2017-06-06 11:40:13	2017-06-06 11:41:20	1.1 min	root	2017-06-06 11:41:20	Download
app-20170606113623-0000	Spark Pi	2017-06-06 11:36:20	2017-06-06 11:36:32	12 s	root	2017-06-06 11:36:32	Download
application_1496748187136_0002	Spark Pi	2017-06-06 11:34:56	2017-06-06 11:35:30	35 s	root	2017-06-06 11:35:30	Download
application_1496748187136_0001	Spark Pi	2017-06-06 11:33:08	2017-06-06 11:33:57	48 s	root	2017-06-06 11:33:57	Download

Spark History UI

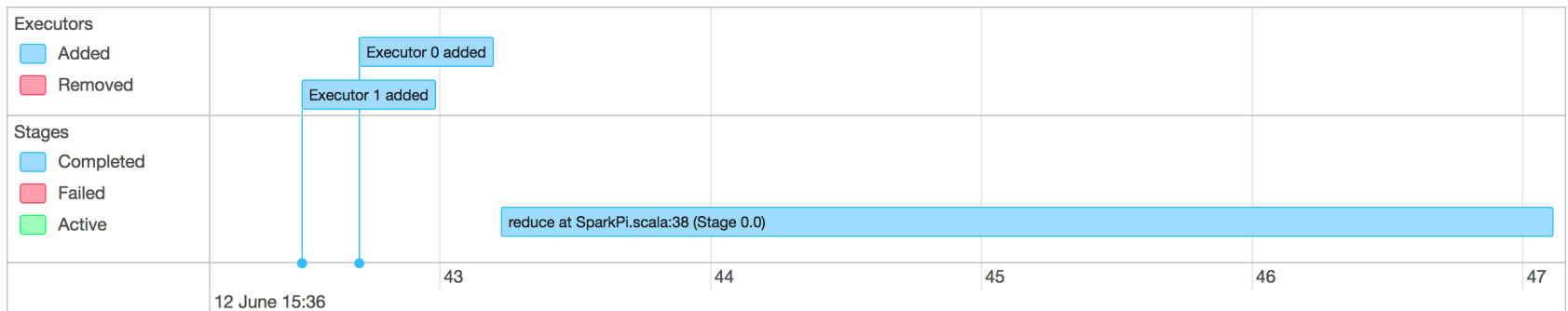
Details for Job 0

Status: SUCCEEDED

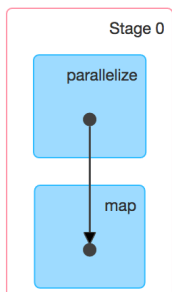
Completed Stages: 1

▼ Event Timeline

☐ Enable zooming



▼ DAG Visualization



Jupyter Notebook

The screenshot shows a Jupyter Notebook interface in a web browser. The address bar displays the URL `10.211.55.101:8888/notebooks/lab1_1-Solution.ipynb`. The notebook title is `lab1_1-Solution`, and the last checkpoint is noted as 'Last Friday at 4:32 PM (autosaved)'. The interface includes a menu bar with options: File, Edit, View, Insert, Cell, Kernel, Widgets, and Help. Below the menu is a toolbar with icons for saving, adding, deleting, and other actions. The main content area displays the following text:

Exercise 1: Spark basics & word count & Pi

This first exercise will introduce the basic Spark concepts and operations. We finish by performing a word count on a small text.

The following material will be covered:

- Part 1: Using the Jupyter notebook*
- Part 2: Creating RDDs*
- Part 3: Simple transformations and actions*
- Part 4: Word count*
- Part 5: Estimation PI by using MonteCarlo Method*

We will look closer at the following Spark operations:

- `parallelize()`, `persist()`, `collect()`, `count()`, `map()`, `flatMap()`, `filter()`, `reduce()`, `take()`, `takeOrdered()`, `first()`, `top()`, `textFile()`

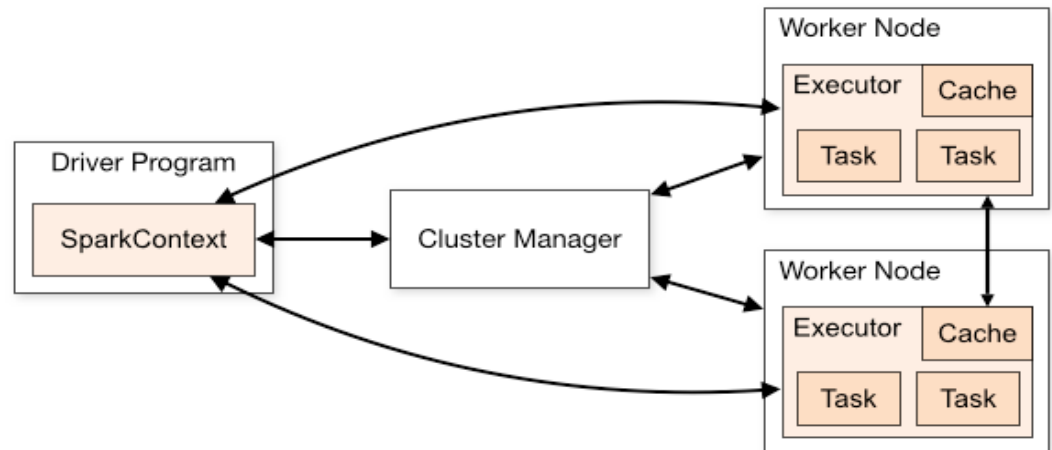
During the exercises, the following resources might come in handy:

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**.

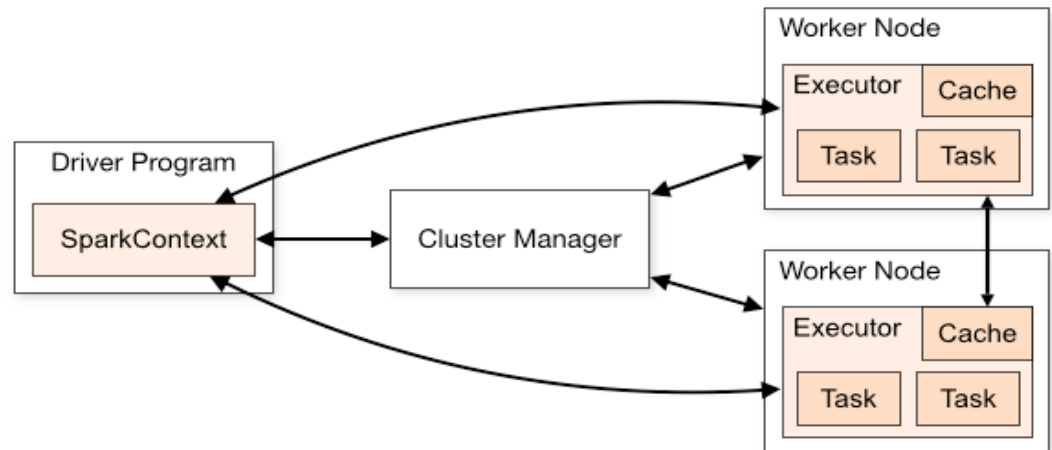
Spark Execution

- Spark applications are run as independent sets of processes, coordinated by a SparkContext in a *driver* program.
- The context will connect to some cluster manager (e.g. YARN, Standalone, Local) which allocates system resources.
- Each worker in the cluster is managed by an *executor*, which is in turn managed by the SparkContext.
- The executor manages computation as well as storage and caching on each machine.

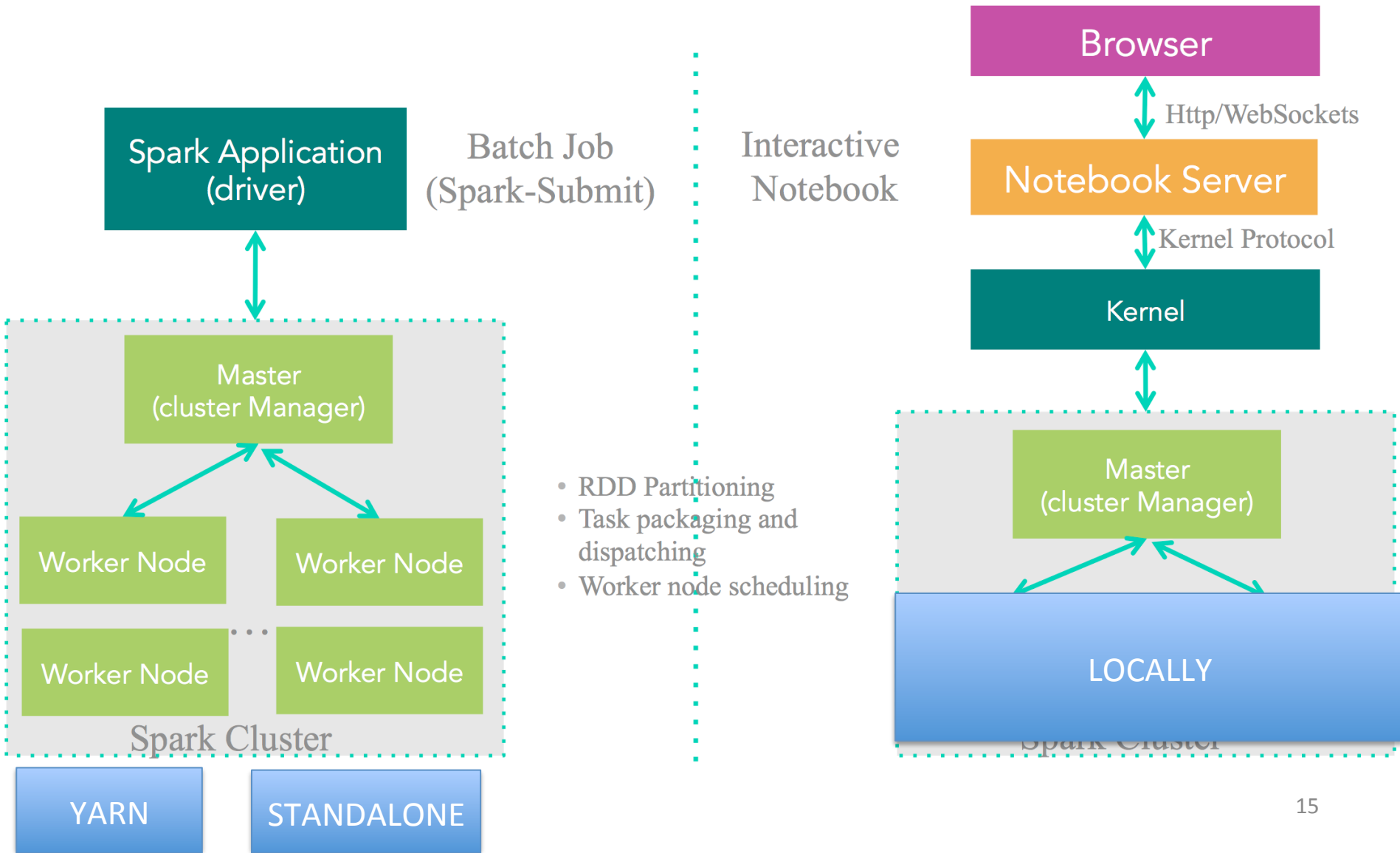


Spark Execution

- What is important to note is that application code is sent from the driver to the executors, and the executors specify the context and the various *tasks* to be run.
- The executors communicate back and forth with the driver for data sharing or for interaction.
- Drivers are key participants in Spark jobs, and therefore, they should be on the same network as the cluster

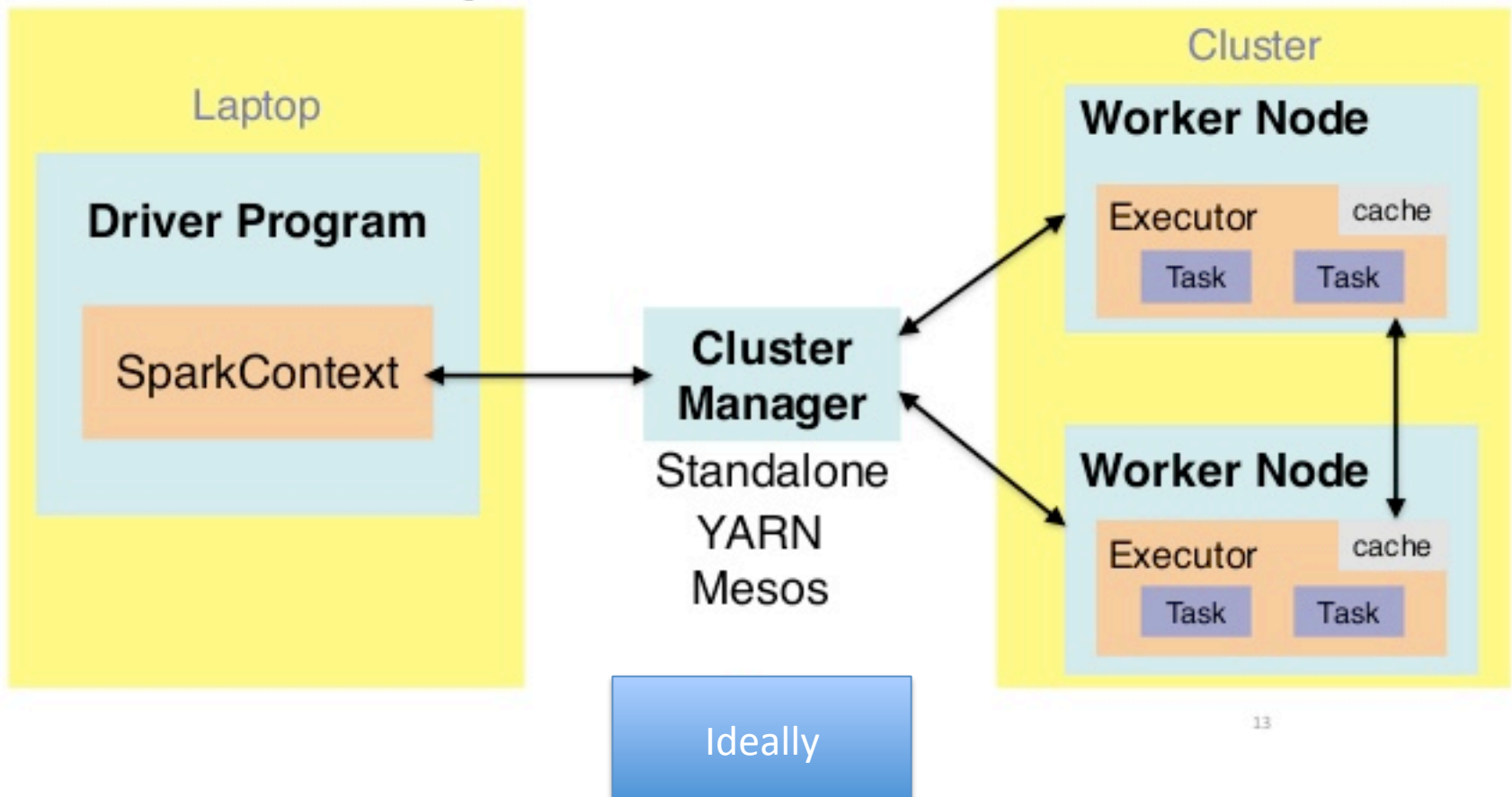


Running Spark Applications

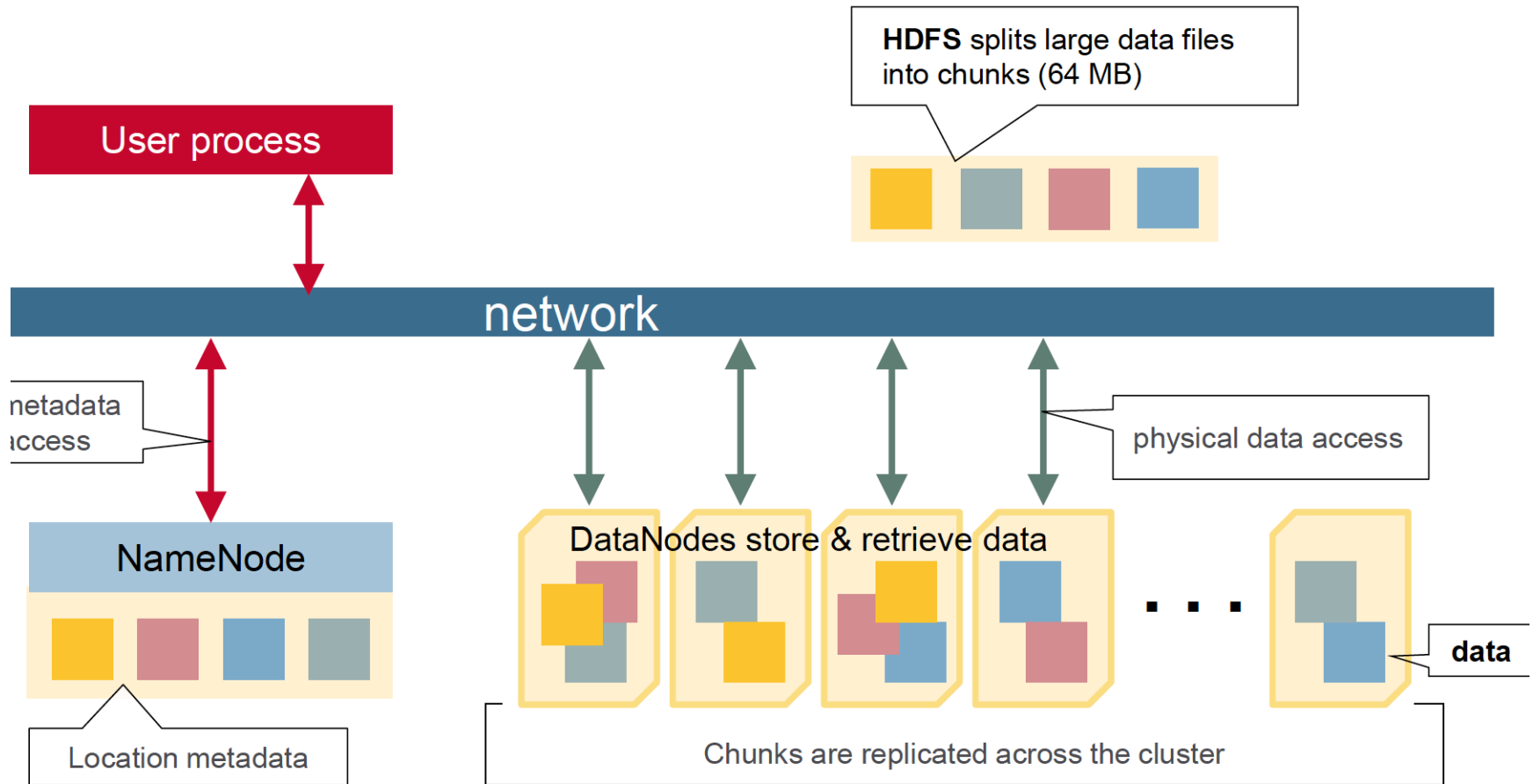


Another possibility

Spark Execution Context



HDFS



HDFS

- Hadoop Distributed File System
- The primary command is: **hdfs dfs**
- Has a lot of basic commands
- Copy a local file to HDFS:
[-put [-f] [-p] [-l] [-d] <localsrc> ... <dst>]
- Assignment:
 - Type the following command in your shell to copy the Readme File into HDFS

```
>> hdfs dfs -put /usr/local/spark/README.md README.md
```

HDFS NameNode UI

http://10.211.55.101:50070/

10.211.55.101:50070/explorer.html#/user/root

Big data

Hadoop Overview Datanodes Datanode Volume Failures Snapshot Startup Progress Utilities

Browse Directory

/user/root Go!

Show 25 entries Search:

Permission	Owner	Group	Size	Last Modified	Replication	Block Size	Name	
-rw-r--r--	root	supergroup	3.73 KB	Jun 08 14:51	3	128 MB	README.md	

HDFS

Common commands

- **-ls [path]**
list directories, if no path is specified it assumes your home directory:
/user/[username]
- **copyFromLocal [local] [hdfs]**
copy data from the local filesystem to HDFS
- **-copyToLocal [hdfs] [local]**
copy data from HDFS to the local filesystem
- **-cat [hdfs]**
show the contents of files directly from HDFS
- **-mv, -cp, -df, -du**
move, copy, disk free, disk used
- **-rm, -rmdir, -expunge**
removal commands

Spark Application

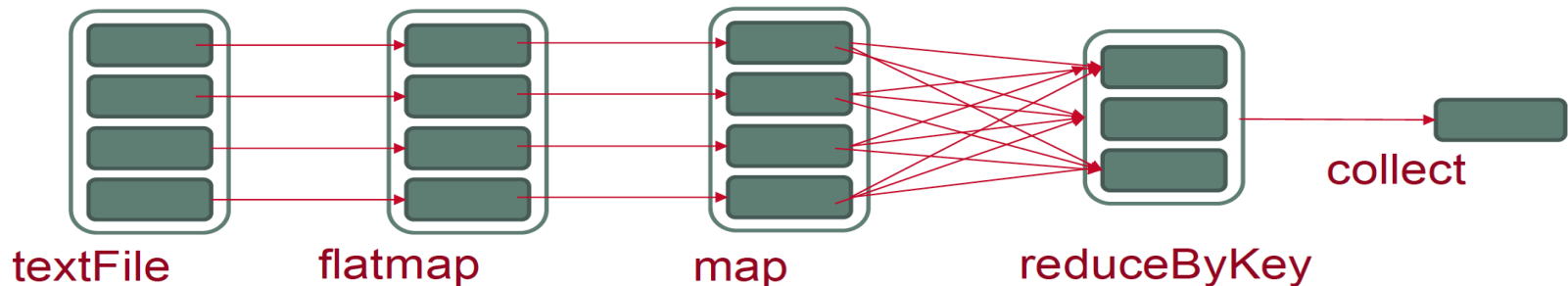
The application that we are going to create is a simple word count:

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

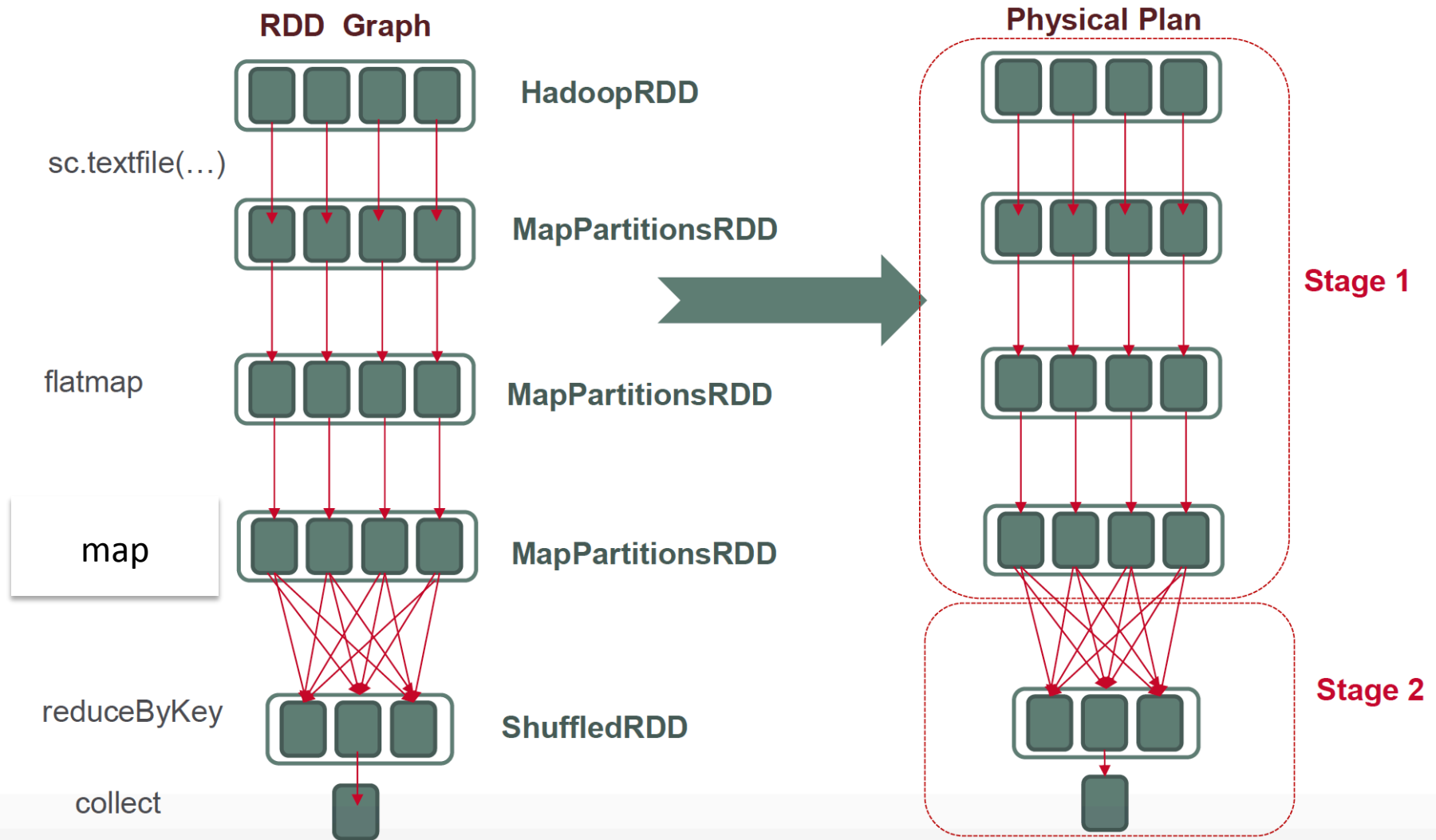
Spark Application

```
import sys
from pyspark import SparkContext, SparkConf

if __name__ == "__main__":
    conf = SparkConf().setAppName("Spark Count")
    sc = SparkContext(conf=conf)
    logFile = "README.md"
    textFile = sc.textFile(logFile)
    wordCounts = textFile.flatMap(lambda line:
    line.split()).map(lambda word: (word, 1)).reduceByKey(lambda a,
    b: a+b)
    numWords=wordCounts.collect()
```

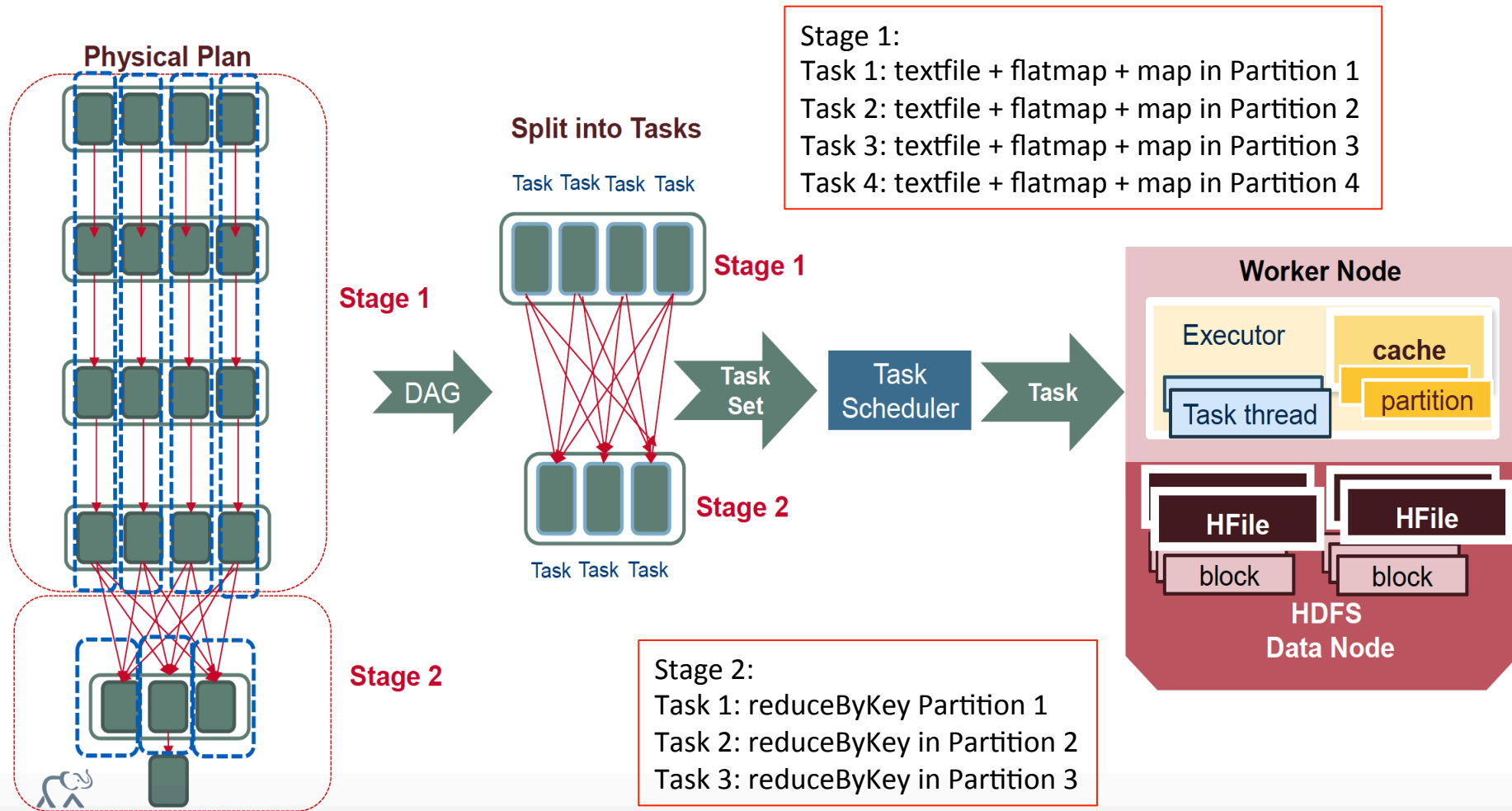


Spark RDD DAG -> Physical Execution plan



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

Physical Execution plan -> Stages and Tasks

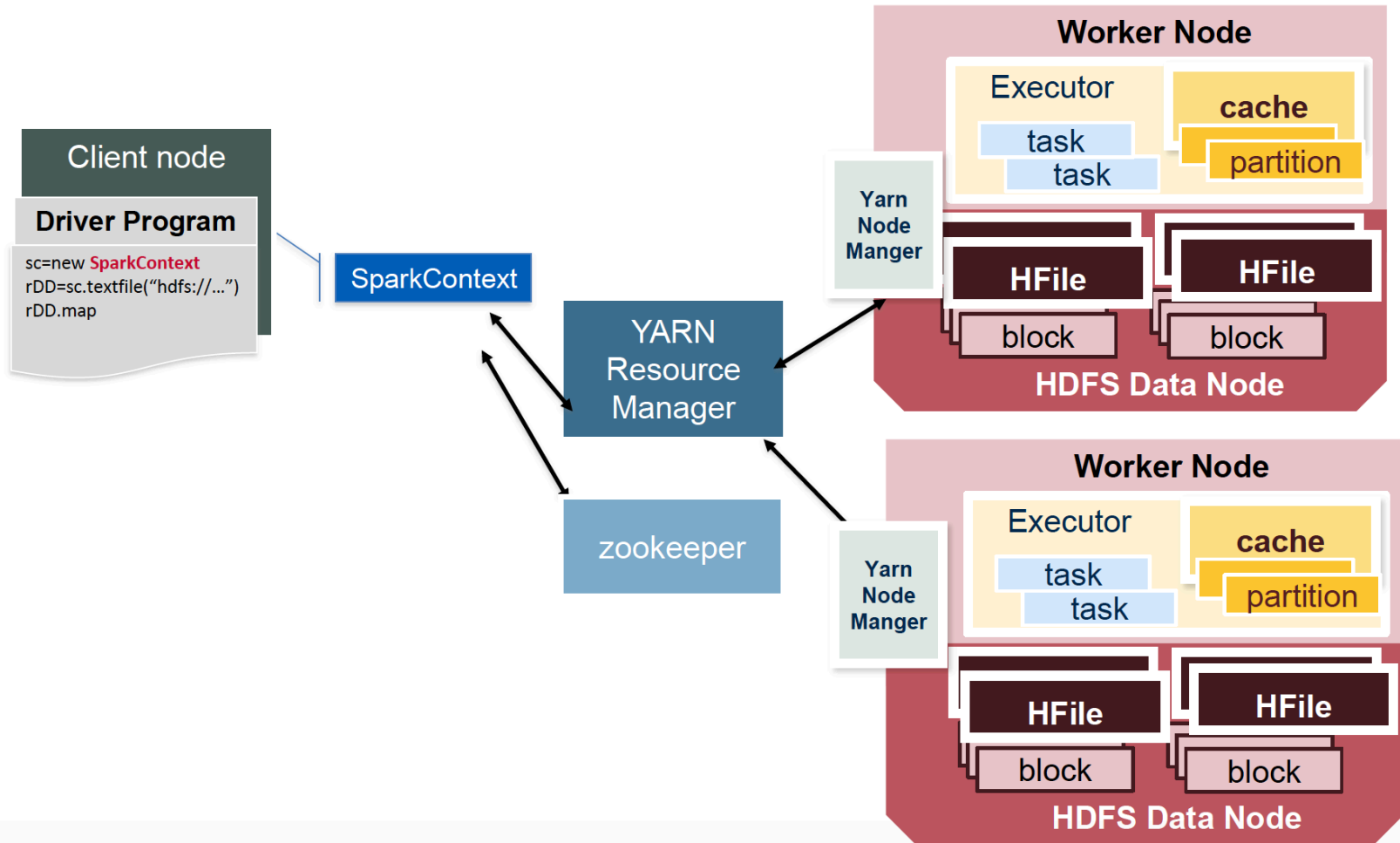


Operations that can run on the same partition are executed in stages

Spark Components

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

Spark Application on Hadoop cluster with YARN



Submit job via spark-submit

spark-submit Syntax

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

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:
 - ‘yarn-cluster’
 - **spark://node1:7077**
 - 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

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

Submit our application via spark-submit

- Yarn-cluster: `$SPARK_HOME/bin/spark-submit --master yarn-cluster wordcount.py`
- Standalone: `: $SPARK_HOME/bin/spark-submit --master spark://node1:7077 wordcount.py`
- Local: `$SPARK_HOME/bin/spark-submit --master local wordcount.py`
- And check the Spark History UI to see the results:
 - <http://10.211.55.101:18080>

Understanding your Apache Spark Application Through Visualization

- Have a look to this website to understand the Spark History UI:

<https://databricks.com/blog/2015/06/22/understanding-your-spark-application-through-visualization.html>