Writing and Deploying Spark Applications

Prof. Hyuk-Yoon Kwon

Spark on a Cluster

In this chapter you will learn

- How to write a Spark Application
- How to run a Spark Application or the Spark Shell on a YARN cluster
- How to access and use the Spark Application Web UI
- How to configure application properties and logging

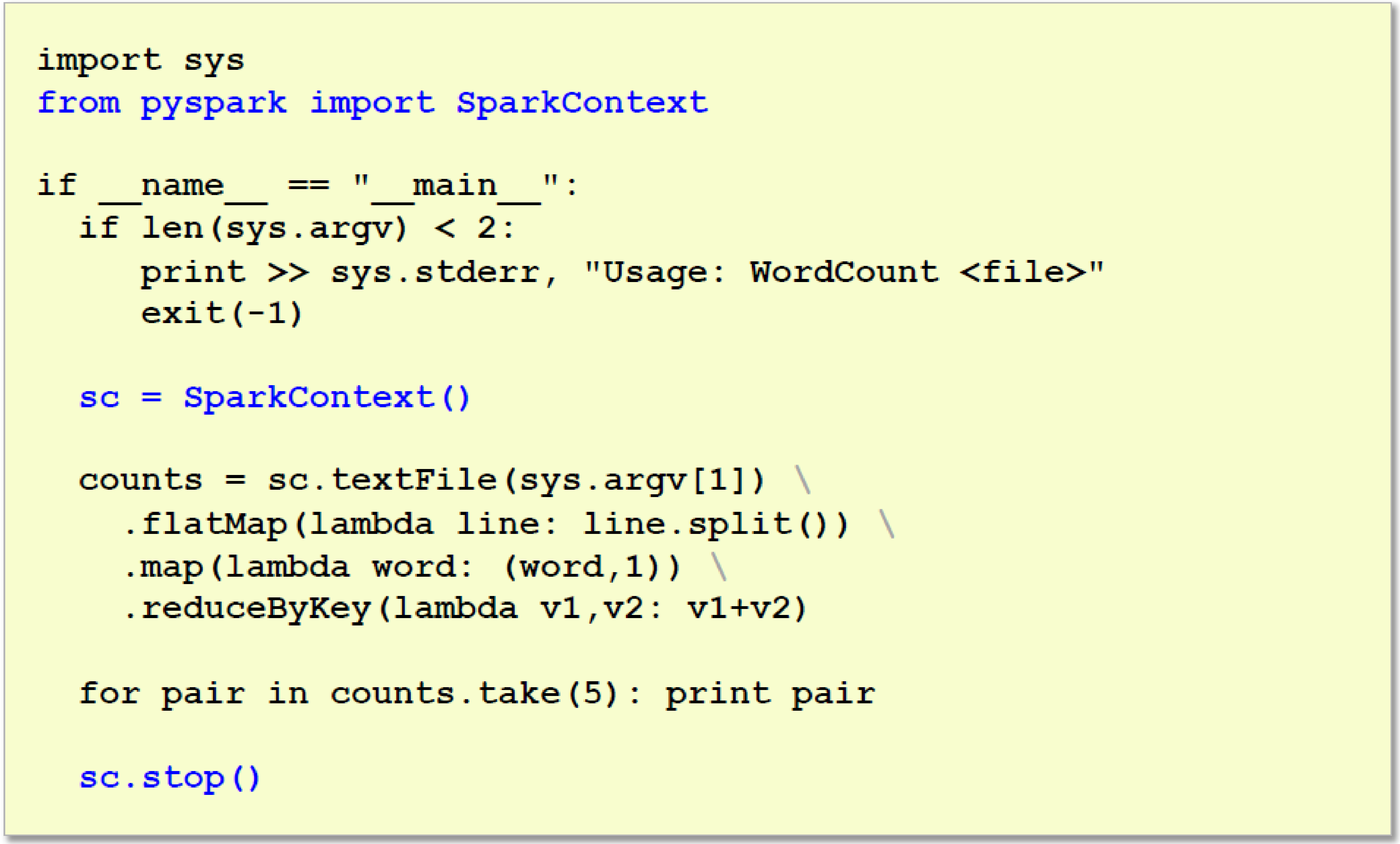
Spark Shell vs. Spark Applications

- **The Spark Shell allows interactive exploration and manipulation of data**
  - REPL using Python or Scala
- **Spark applications run as independent programs**
  - Python, Scala, or Java
  - e.g., ETL processing, Streaming, and so on

The SparkContext

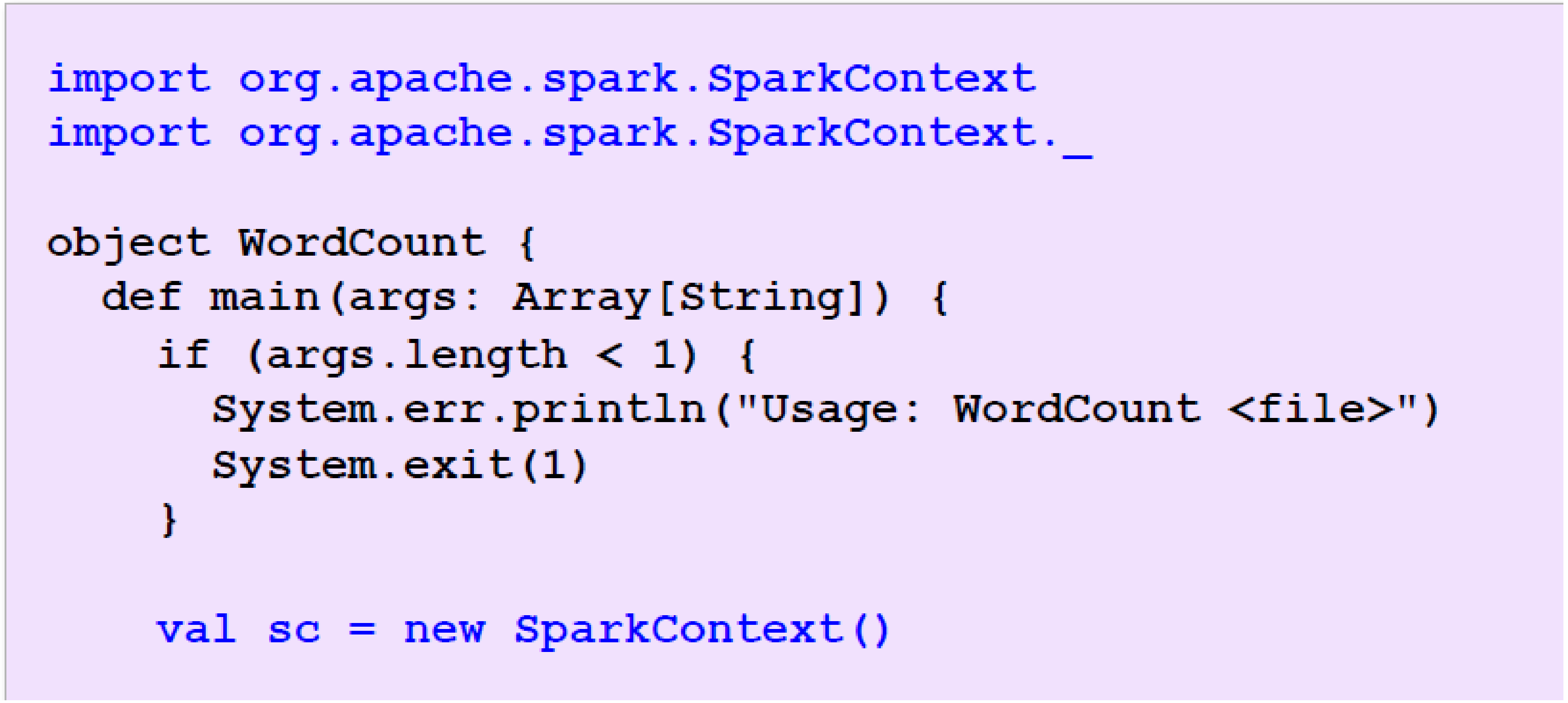
- **Every Spark program needs a SparkContext**
  - The interactive shell creates one for you
- **In your own Spark application you create your own SparkContext**
  - Named `sc` by convention
  - Call `sc.stop` when program terminates

Python Example: WordCount



```
import sys
from pyspark import SparkContext

if __name__ == "__main__":
  if len(sys.argv) < 2:
     print >> sys.stderr, "Usage: WordCount <file>"
     exit(-1)

  sc = SparkContext()

  counts = sc.textFile(sys.argv[1]) \
    .flatMap(lambda line: line.split()) \
    .map(lambda word: (word,1)) \
    .reduceByKey(lambda v1,v2: v1+v2)

  for pair in counts.take(5): print pair

  sc.stop()
```

Scala Example: WordCount



```
import org.apache.spark.SparkContext
import org.apache.spark.SparkContext._

object WordCount {
  def main(args: Array[String]) {
    if (args.length < 1) {
      System.err.println("Usage: WordCount <file>")
      System.exit(1)
    }

    val sc = new SparkContext()
```

Running a Spark Application

- The easiest way to run a Spark Application is using the spark-submit script

Python

```
$ spark-submit WordCount.py fileURL
```

Scala

Java

```
$ spark-submit --class WordCount \
  MyJarFile.jar fileURL
```

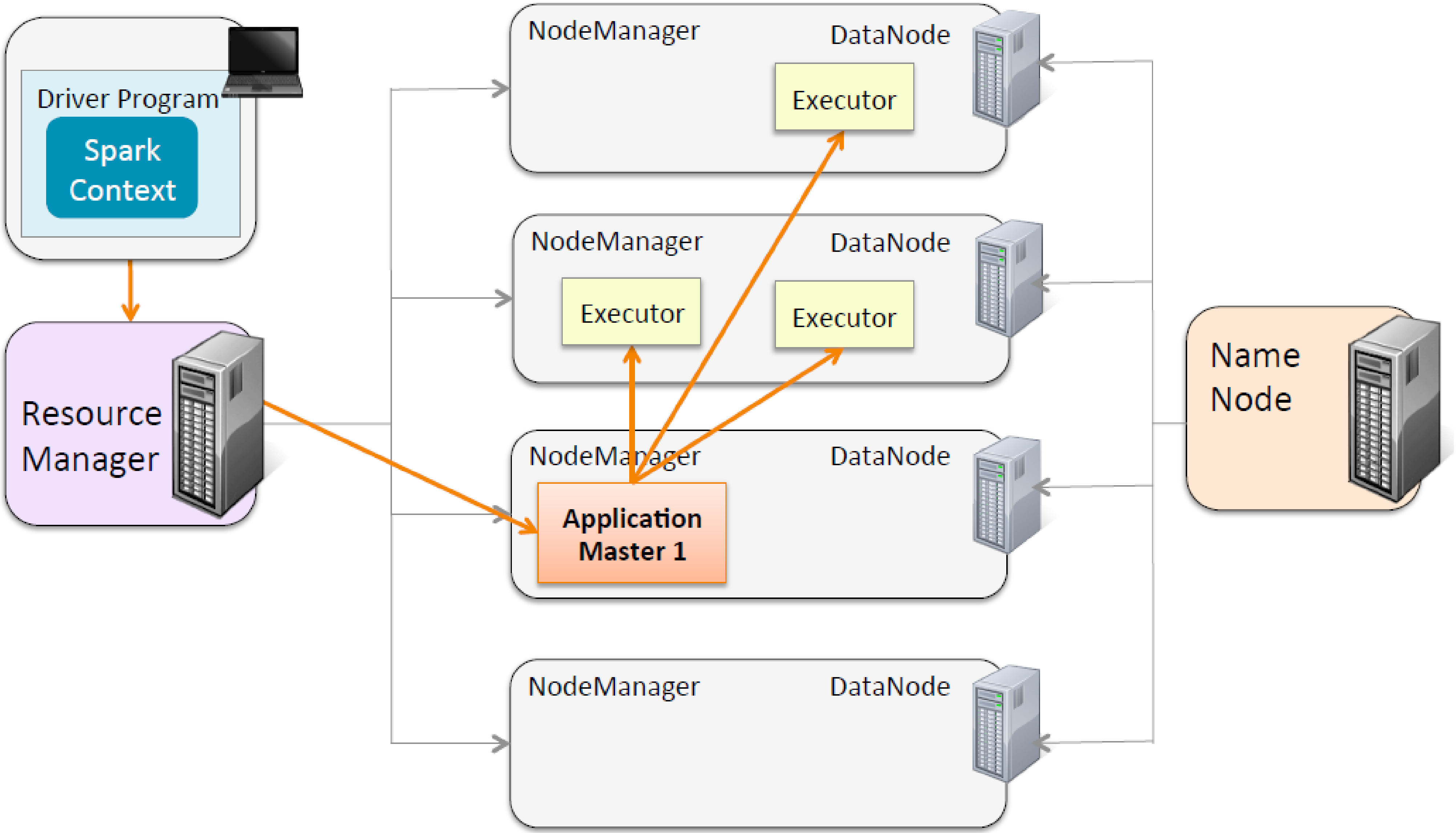
Spark Application Cluster Options

- **Spark can run**
  - Locally
    - No distributed processing
  - Locally with multiple worker threads
  - On a cluster
- **Local mode is useful for development and testing**
- **Production use is almost always on a cluster**

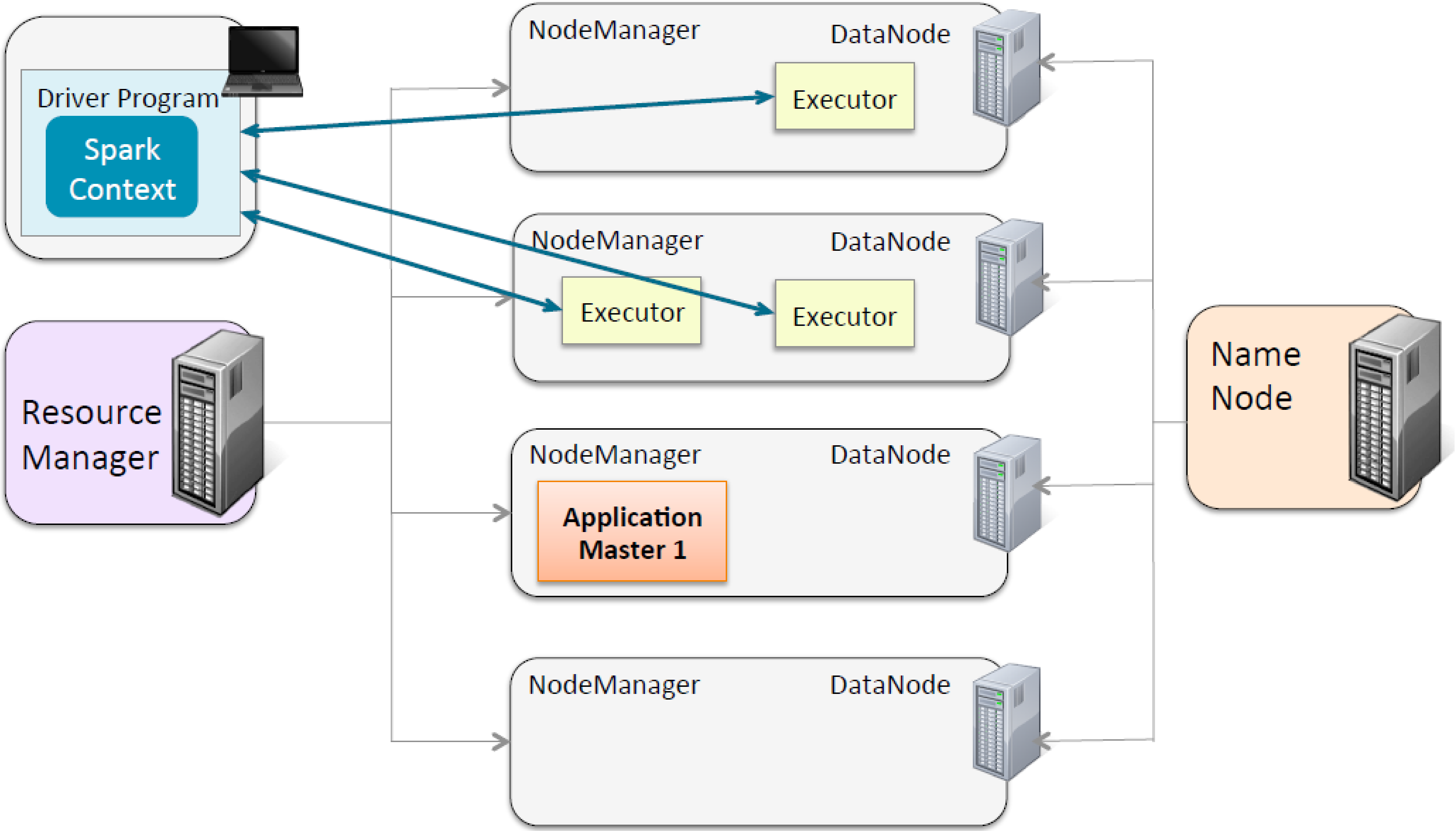
Supported Cluster Resource Managers

- **Hadoop YARN**
  - Included in CDH
  - Most common for production sites
  - Allows sharing cluster resources with other applications (e.g. MapReduce, Impala)
- **Spark Standalone**
  - Included with Spark
  - Easy to install and run
  - Limited configurability and scalability
  - Useful for testing, development, or small systems
- **Apache Mesos**
  - First platform supported by Spark
  - Now used less often

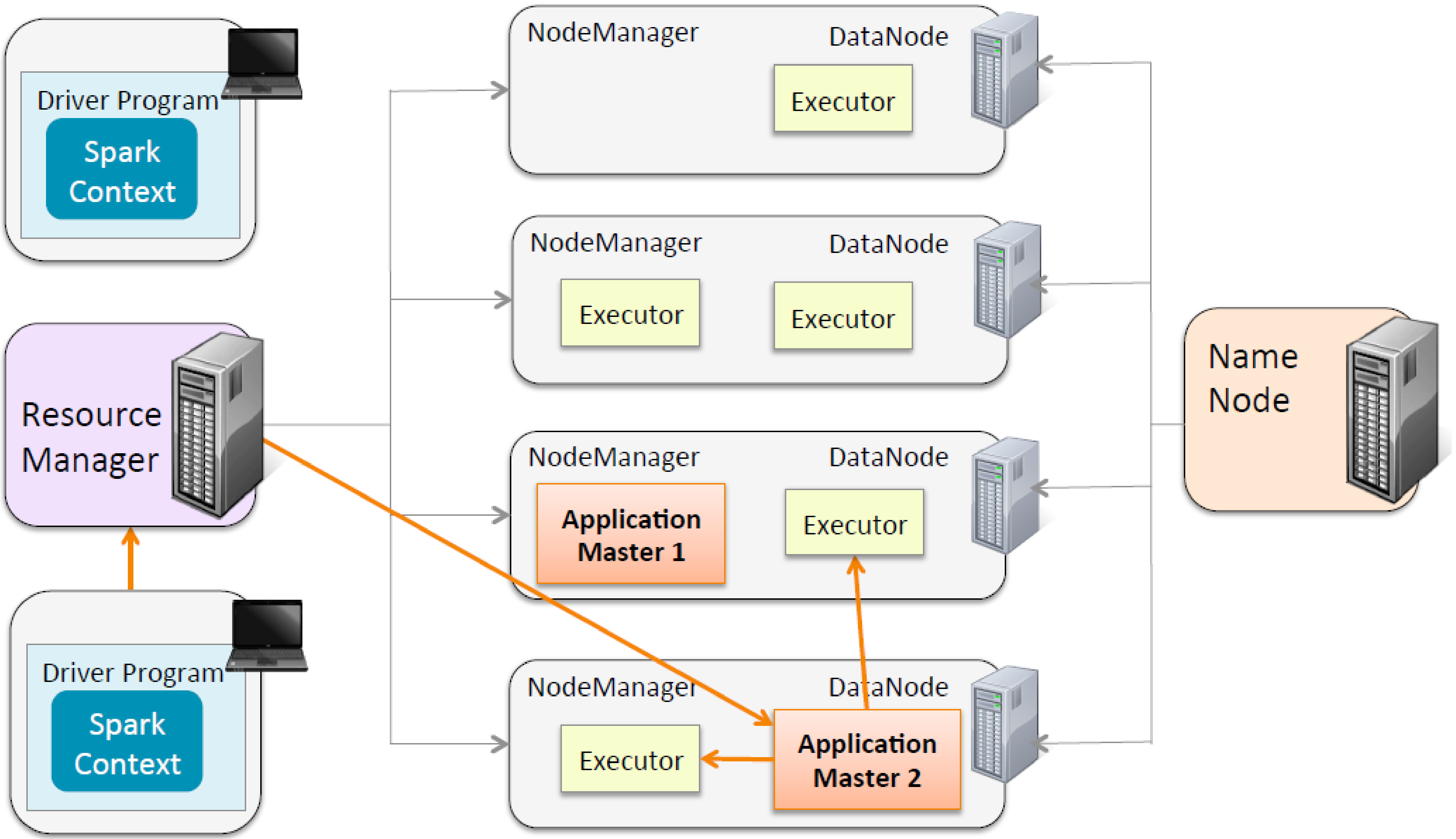
How Spark Runs on YARN: Client Mode (1)



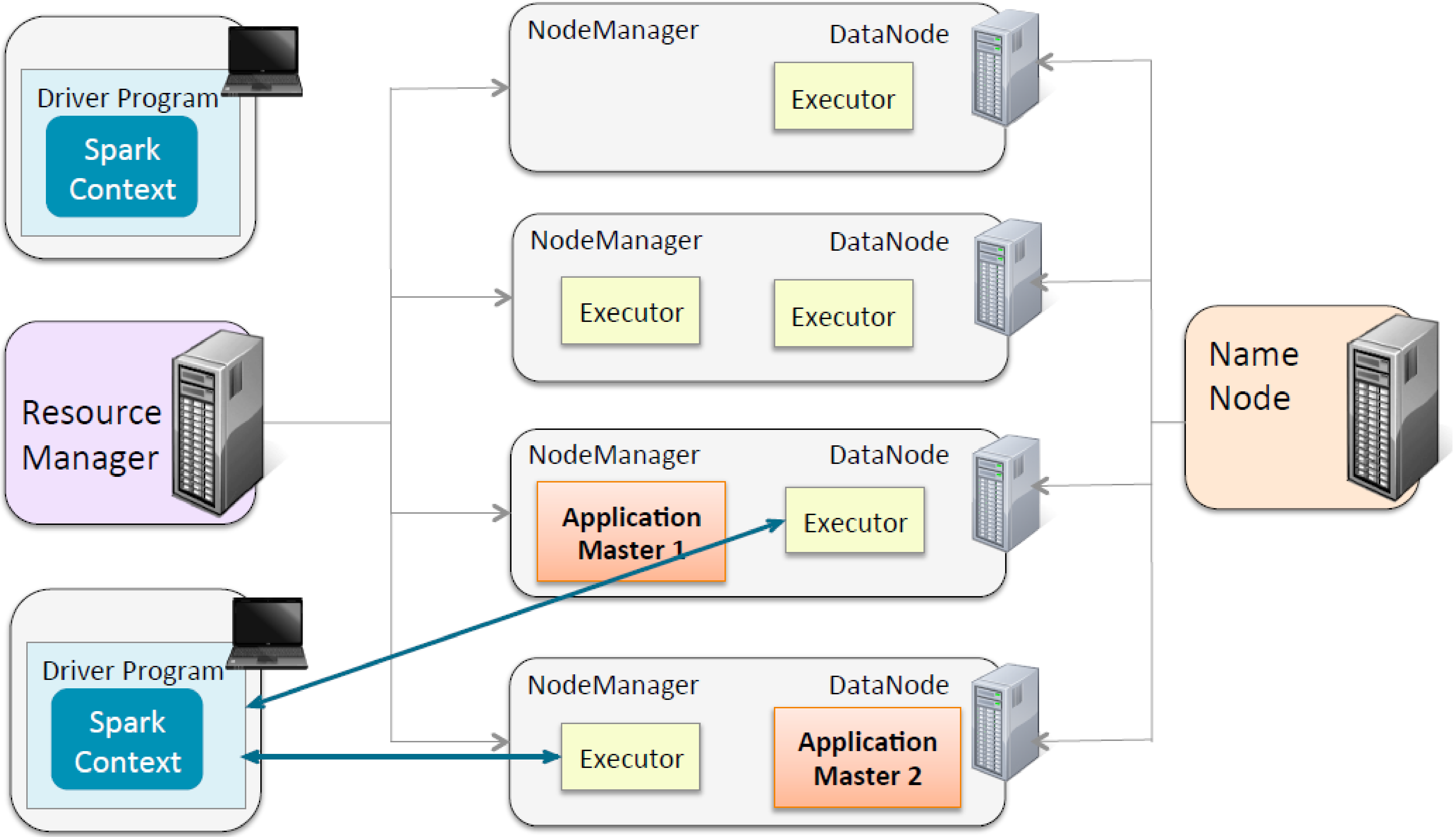
How Spark Runs on YARN: Client Mode (2)



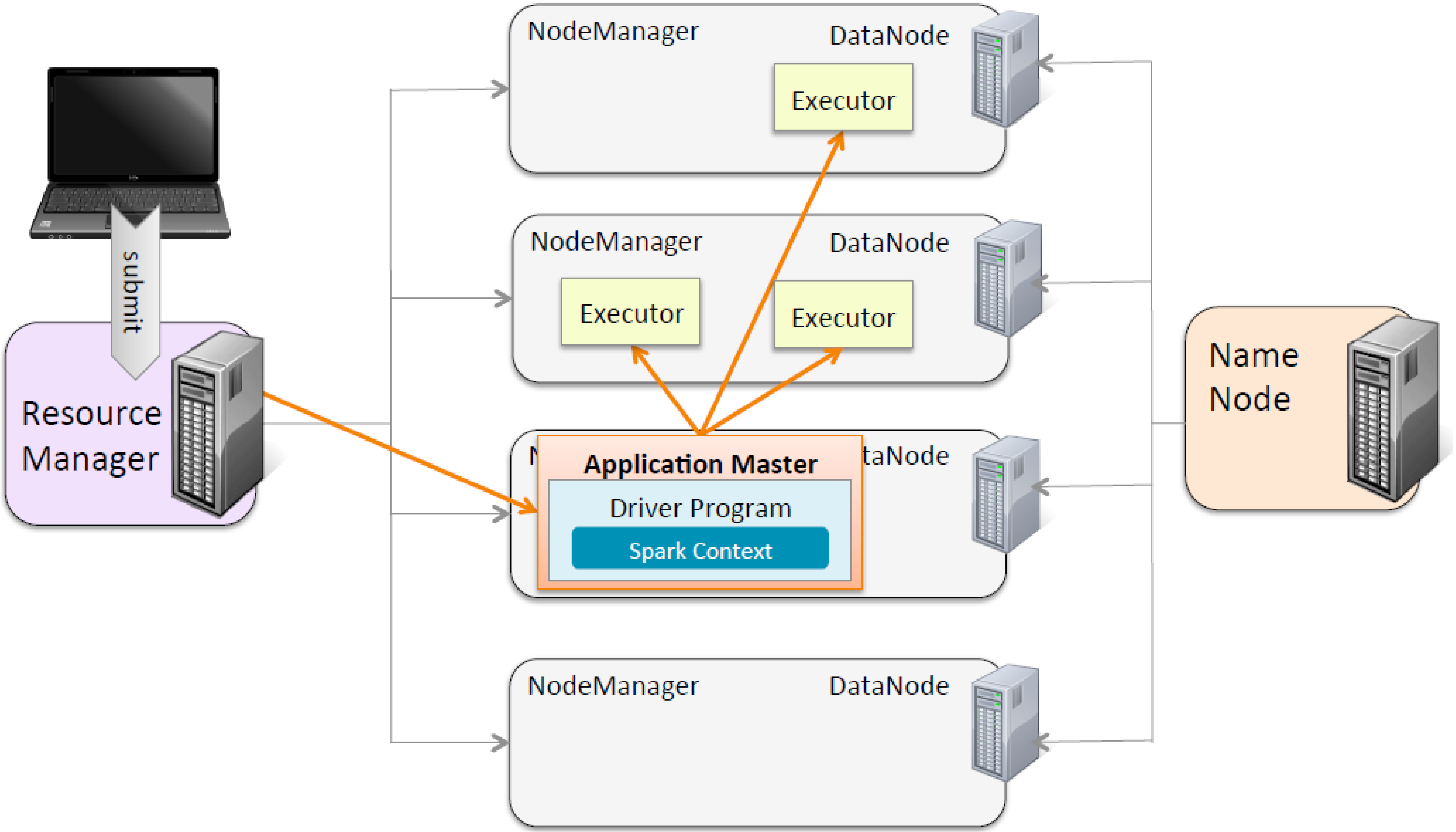
How Spark Runs on YARN: Client Mode (3)



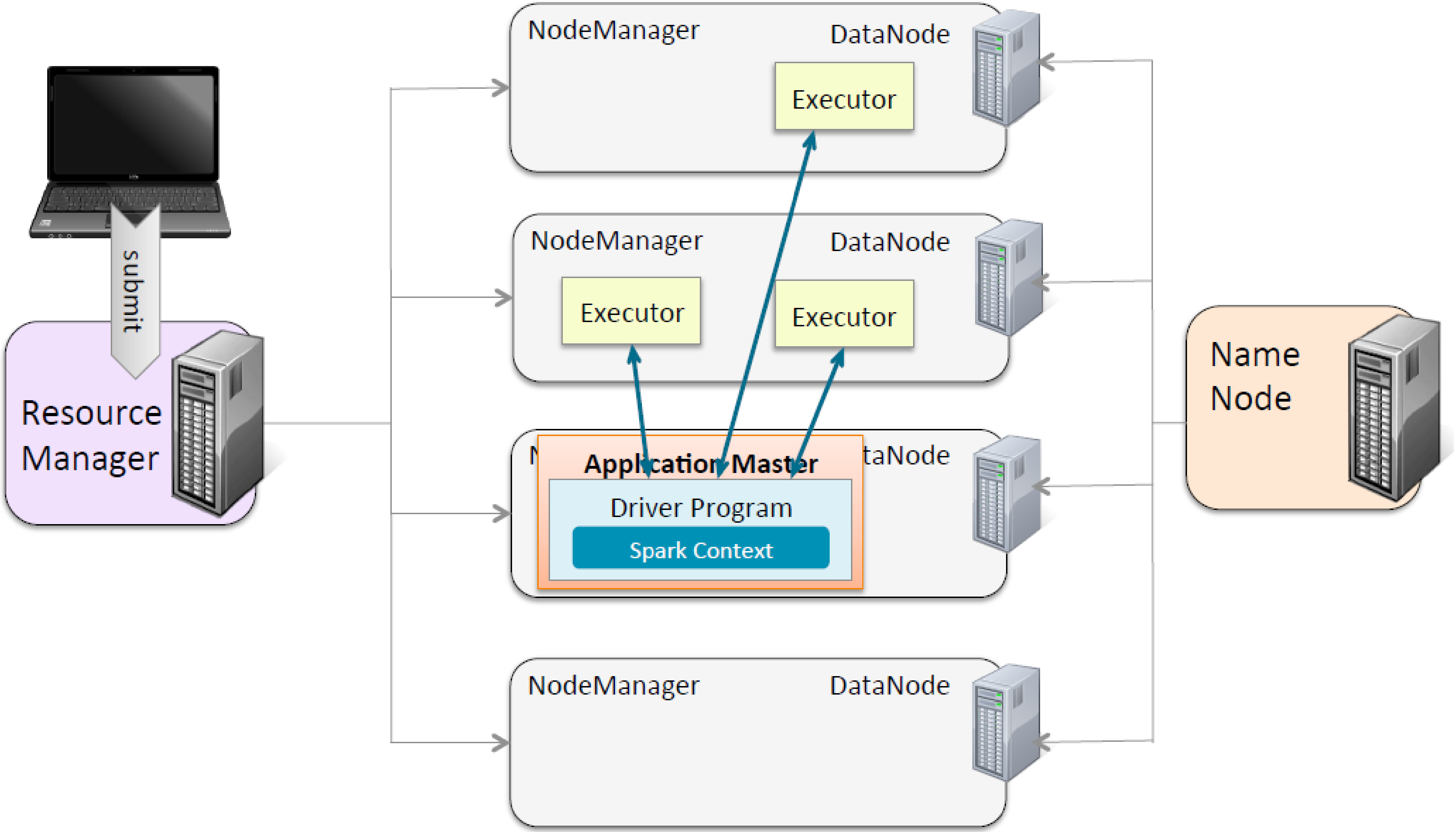
How Spark Runs on YARN: Client Mode (4)



How Spark Runs on YARN: Cluster Mode (1)



How Spark Runs on YARN: Cluster Mode (2)



Running a Spark Application Locally

- **Use `spark-submit --master` to specify cluster option**
  - Local options
    - **`local[*]`** – run locally with as many threads as cores (default)
    - **`local[n]`** – run locally with n threads
    - **`local`** – run locally with a single thread

Python

```
$ spark-submit --master local[3] \
  WordCount.py fileURL
```

Scala

Java

```
$ spark-submit --master local[3] --class \
  WordCount MyJarFile.jar fileURL
```

Practice: Write and Run a Spark Application

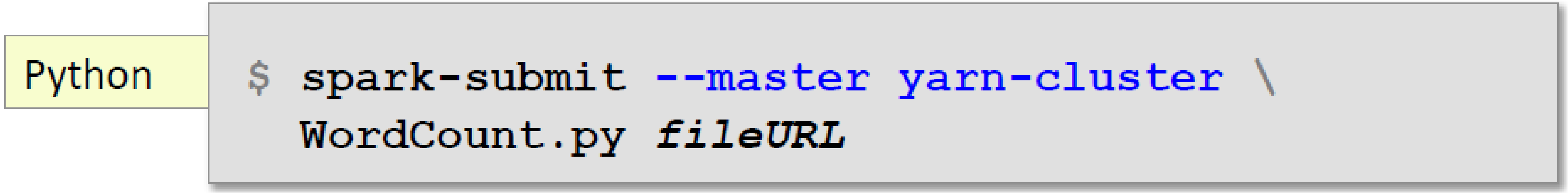
- **Data files (HDFS)**
  - Data path: /louadacre/weblogs
  - If it is not located in the path, upload the data from the local system into HDFS

- **In this practice, you will write your own Spark application instead of using interactive Spark Shell application**
  - Before running your program, be sure to exit from the Spark Shell
  - Download CountJPGs.py from the e-class

- **Write a simple program that counts the number of JPG requests in a web log file. The name of the file should be passed into the program as an argument.**
  1. Set up a SparkContext
  2. In the body of the program, load the file passed into the program, count the number of JPG requests, and display the count.
  3. Run the program, passing the names of the log files to process

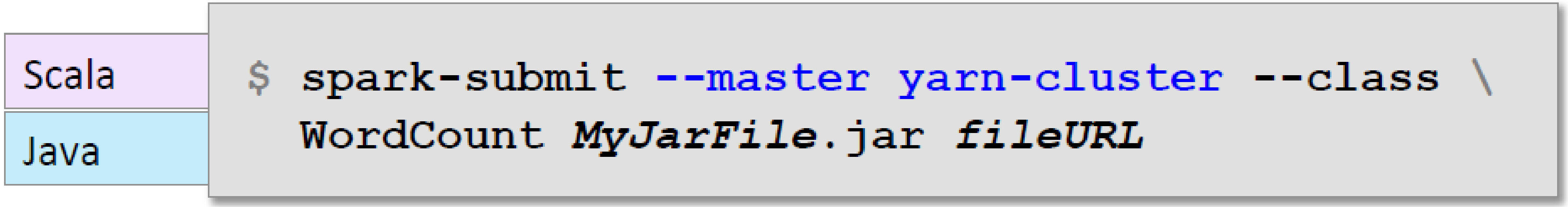
Running a Spark Application on a Cluster

- Use `spark-submit --master` to specify cluster option
  - Cluster options
    - `yarn-client`
    - `yarn-cluster`
    - `spark://`*`masternode`*`:port` (Spark Standalone)
    - `mesos://masternode:port` (Mesos)



Python

```
$ spark-submit --master yarn-cluster \
  WordCount.py fileURL
```



Scala

Java

```
$ spark-submit --master yarn-cluster --class \
  WordCount MyJarFile.jar fileURL
```

Starting the Spark Shell on a Cluster

- **The Spark Shell can also be run on a cluster**
- **Pyspark and spark-shell both have a `--master` option**
  - `yarn` (client mode only)
  - Spark or Mesos cluster manager URL
  - `local[*]` – run with as many threads as cores (default)
  - `local[`*`n`*`]` – run locally with *n* worker threads
  - `local` – run locally without distributed processing

Python

```
$ pyspark --master yarn
```

Scala

```
$ spark-shell --master yarn
```

Options When Submitting a Spark Application to a Cluster

- **Some other `spark-submit` options for clusters**
  - `--jars` – additional JAR files (Scala and Java only)
  - `--py-files` – additional Python files (Python only)
  - `--driver-java-options` – parameters to pass to the driver JVM
  - `--executor-memory` – memory per executor (e.g. 1000M, 2G) (Default: 1G)
  - `--packages` -- Maven coordinates of an external library to include
- **Plus several YARN-specific options**
  - `--num-executors`
  - `--queue`
- **Show all available options**
  - `--help`

Practice: Run a Spark Application on a Cluster

Submit a Spark application to the cluster

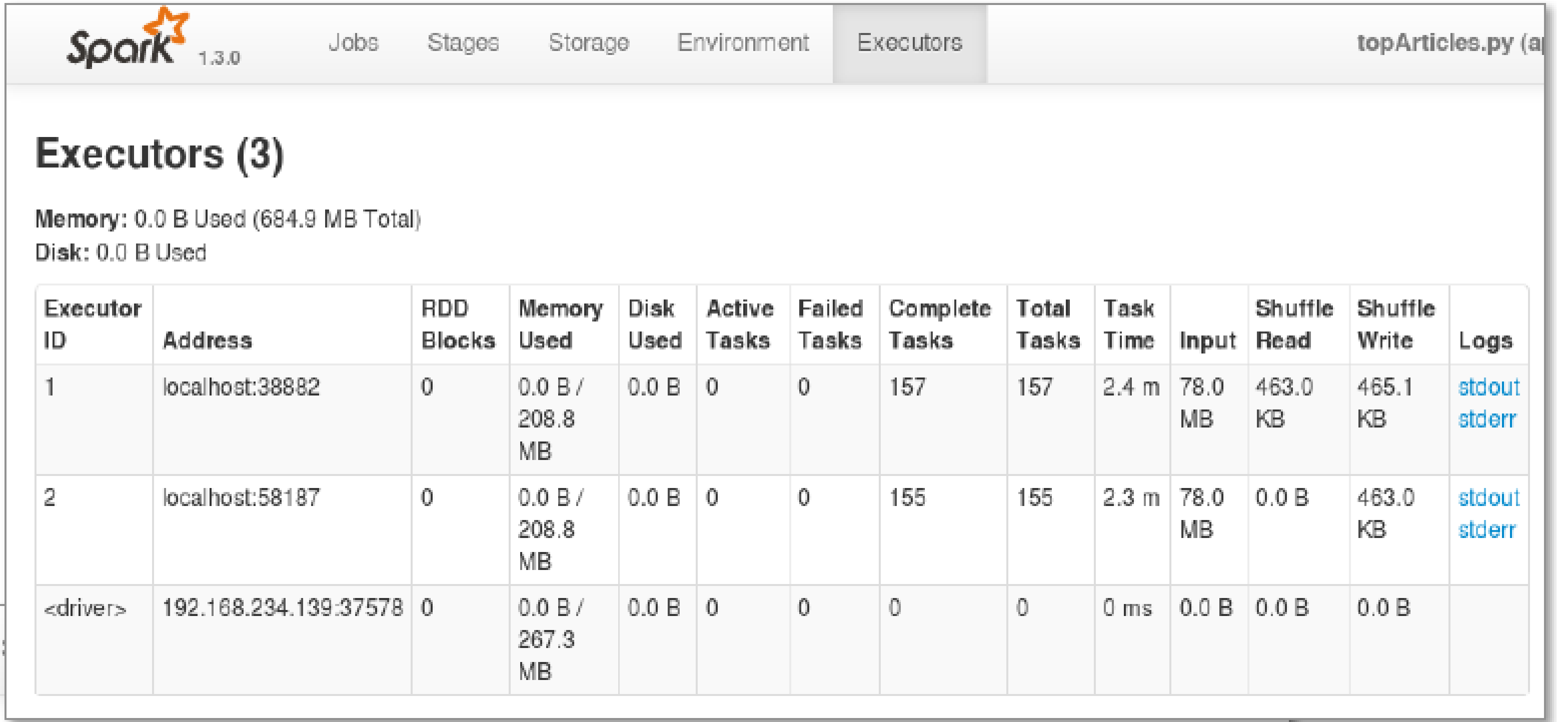
- In the previous practice, you ran a Python application using spark-submit. By default, spark-submit runs the application locally. In this section, run the application on the YARN cluster instead.

Directions

1. re-run the program specifying the cluster master in order to run it on the cluster
2. After starting the application, open Firefox and visit the YARN Resource Manager UI using the provided bookmark (or going to URL http://localhost:8088)
3. In a terminal window, take note of the your application's ID (e.g., application_1234…) using "yarn logs …" command

The Spark Application Web UI

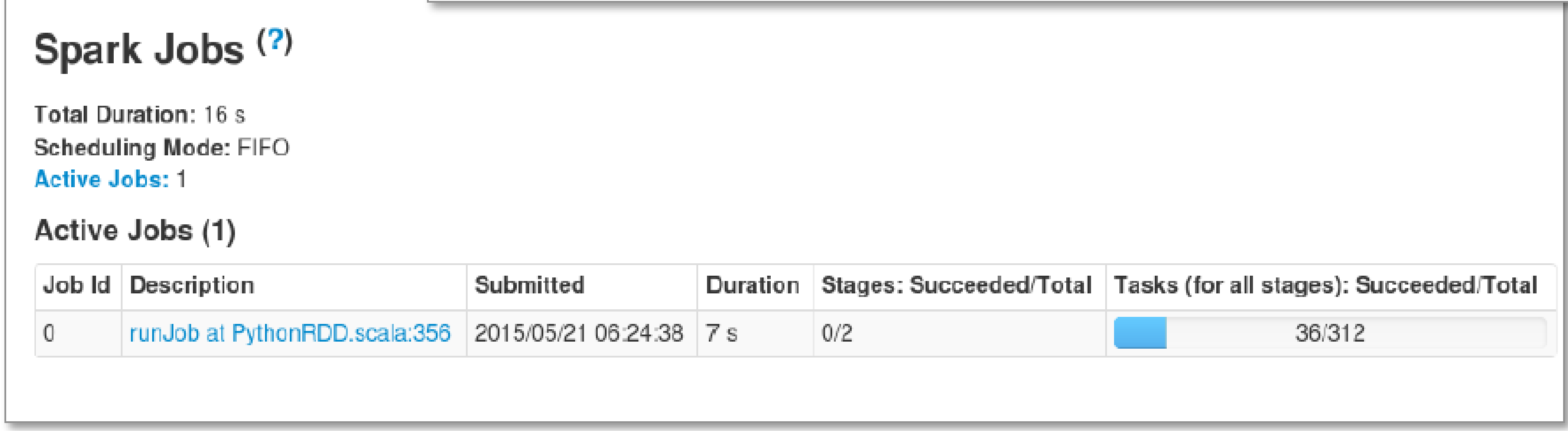
The Spark UI lets you monitor running jobs, and view statistics and configuration



Executors (3)

Memory: 0.0 B Used (684.9 MB Total)
Disk: 0.0 B Used

Executor ID	Address	RDD Blocks	Memory Used	Disk Used	Active Tasks	Failed Tasks	Complete Tasks	Total Tasks	Task Time	Input	Shuffle Read	Shuffle Write	Logs
1	localhost:38882	0	0.0 B / 208.8 MB	0.0 B	0	0	157	157	2.4 m	78.0 MB	463.0 KB	465.1 KB	stdout stderr
2	localhost:58187	0	0.0 B / 208.8 MB	0.0 B	0	0	155	155	2.3 m	78.0 MB	0.0 B	463.0 KB	stdout stderr
<driver>	192.168.234.139:37578	0	0.0 B / 267.3 MB	0.0 B	0	0	0	0	0 ms	0.0 B	0.0 B	0.0 B	



Spark Jobs (?)

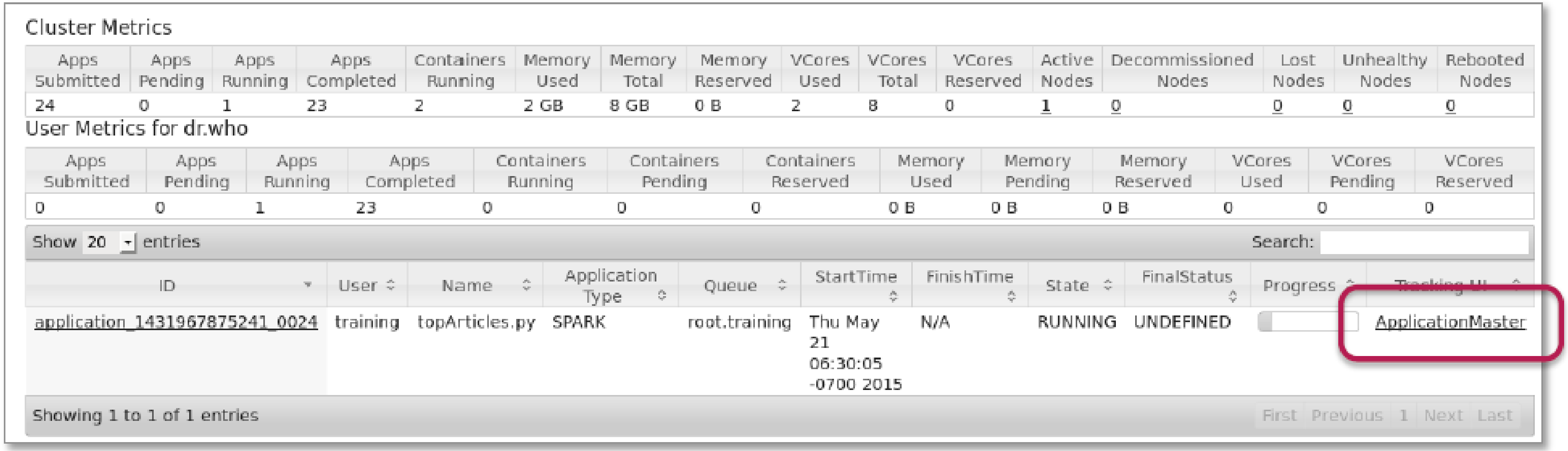
Total Duration: 16 s
Scheduling Mode: FIFO
Active Jobs: 1

Active Jobs (1)

Job Id	Description	Submitted	Duration	Stages: Succeeded/Total	Tasks (for all stages): Succeeded/Total
0	runJob at PythonRDD.scala:356	2015/05/21 06:24:38	7 s	0/2	36/312

Accessing the Spark UI

- **The Web UI is run by the Spark drivers**
  - When running locally: `http://localhost:4040`
  - When running on a cluster, access via the cluster UI, e.g. YARN UI



Cluster Metrics

Apps Submitted	Apps Pending	Apps Running	Apps Completed	Containers Running	Memory Used	Memory Total	Memory Reserved	VCores Used	VCores Total	VCores Reserved	Active Nodes	Decommissioned Nodes	Lost Nodes	Unhealthy Nodes	Rebooted Nodes
24	0	1	23	2	2 GB	8 GB	0 B	2	8	0	1	0	0	0	0

User Metrics for dr.who

Apps Submitted	Apps Pending	Apps Running	Apps Completed	Containers Running	Containers Pending	Containers Reserved	Memory Used	Memory Pending	Memory Reserved	VCores Used	VCores Pending	VCores Reserved
0	0	1	23	0	0	0	0 B	0 B	0 B	0	0	0

Show 20 entries Search:

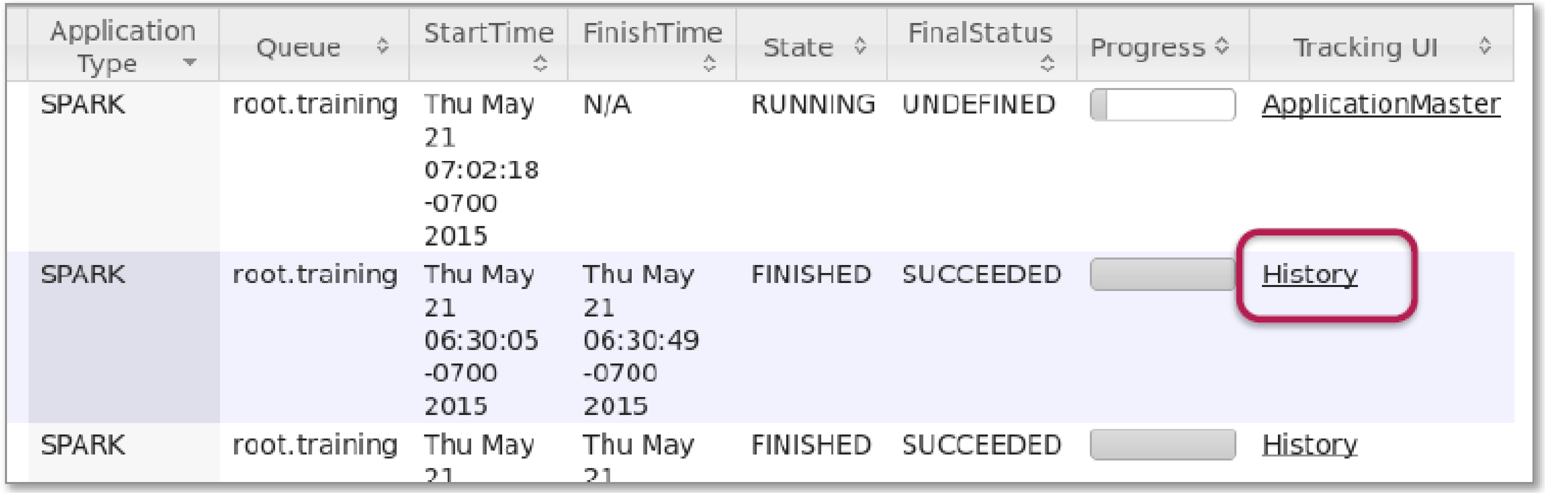
ID	User	Name	Application Type	Queue	StartTime	FinishTime	State	FinalStatus	Progress	Tracking UI
application_1431967875241_0024	training	topArticles.py	SPARK	root.training	Thu May 21 06:30:05 -0700 2015	N/A	RUNNING	UNDEFINED		ApplicationMaster

Showing 1 to 1 of 1 entries

Viewing Spark Job History (1)

- **Viewing Spark Job History**
  - Spark UI is only available while the application is running
  - Use Spark History Server to view metrics for a completed application
    - Optional Spark component

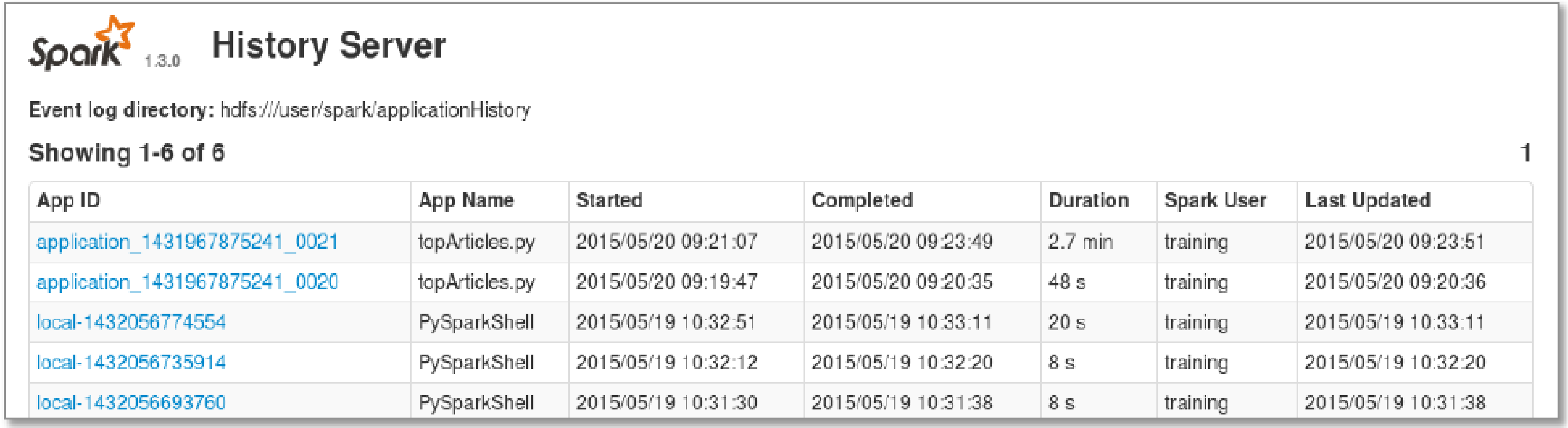
- **Accessing the History Server**
  - For local jobs, access by URL
    - E.g. **`localhost:18080`**
  - For YARN Jobs, click History link in YARN UI



Application Type	Queue	StartTime	FinishTime	State	FinalStatus	Progress	Tracking UI
SPARK	root.training	Thu May 21 07:02:18 -0700 2015	N/A	RUNNING	UNDEFINED		ApplicationMaster
SPARK	root.training	Thu May 21 06:30:05 -0700 2015	Thu May 21 06:30:49 -0700 2015	FINISHED	SUCCEEDED		History
SPARK	root.training	Thu May 21	Thu May 21	FINISHED	SUCCEEDED		History

Viewing Spark Job History (2)

- Spark History Server



History Server

Event log directory: hdfs:///user/spark/applicationHistory

Showing 1-6 of 6 1

App ID	App Name	Started	Completed	Duration	Spark User	Last Updated
application_1431967875241_0021	topArticles.py	2015/05/20 09:21:07	2015/05/20 09:23:49	2.7 min	training	2015/05/20 09:23:51
application_1431967875241_0020	topArticles.py	2015/05/20 09:19:47	2015/05/20 09:20:35	48 s	training	2015/05/20 09:20:36
local-1432056774554	PySparkShell	2015/05/19 10:32:51	2015/05/19 10:33:11	20 s	training	2015/05/19 10:33:11
local-1432056735914	PySparkShell	2015/05/19 10:32:12	2015/05/19 10:32:20	8 s	training	2015/05/19 10:32:20
local-1432056693760	PySparkShell	2015/05/19 10:31:30	2015/05/19 10:31:38	8 s	training	2015/05/19 10:31:38

Spark Application Configuration

- **Spark provides numerous properties for configuring your application**
- **Some example properties**
  - **`spark.master`**
  - **`spark.app.name`**
  - **`spark.local.dir`** – where to store local files such as shuffle output (default **`/tmp`**)
  - **`spark.ui.port`** – port to run the Spark Application UI (default **`4040`**)
  - **`spark.executor.memory`** – how much memory to allocate to each Executor (default **`512m`**)
  - And many more...
    - See Spark Configuration page for more details

Spark Application Configuration

- **Spark Applications can be configured**
  - Declaratively *or*
  - Programmatically

Declarative Configuration Options

- `spark-submit` script
  - e.g., `spark-submit --driver-memory 500M`
- Properties file
  - Tab- or space-separated list of properties and values
  - Load with `spark-submit --properties-file` ***`filename`***
  - Example:

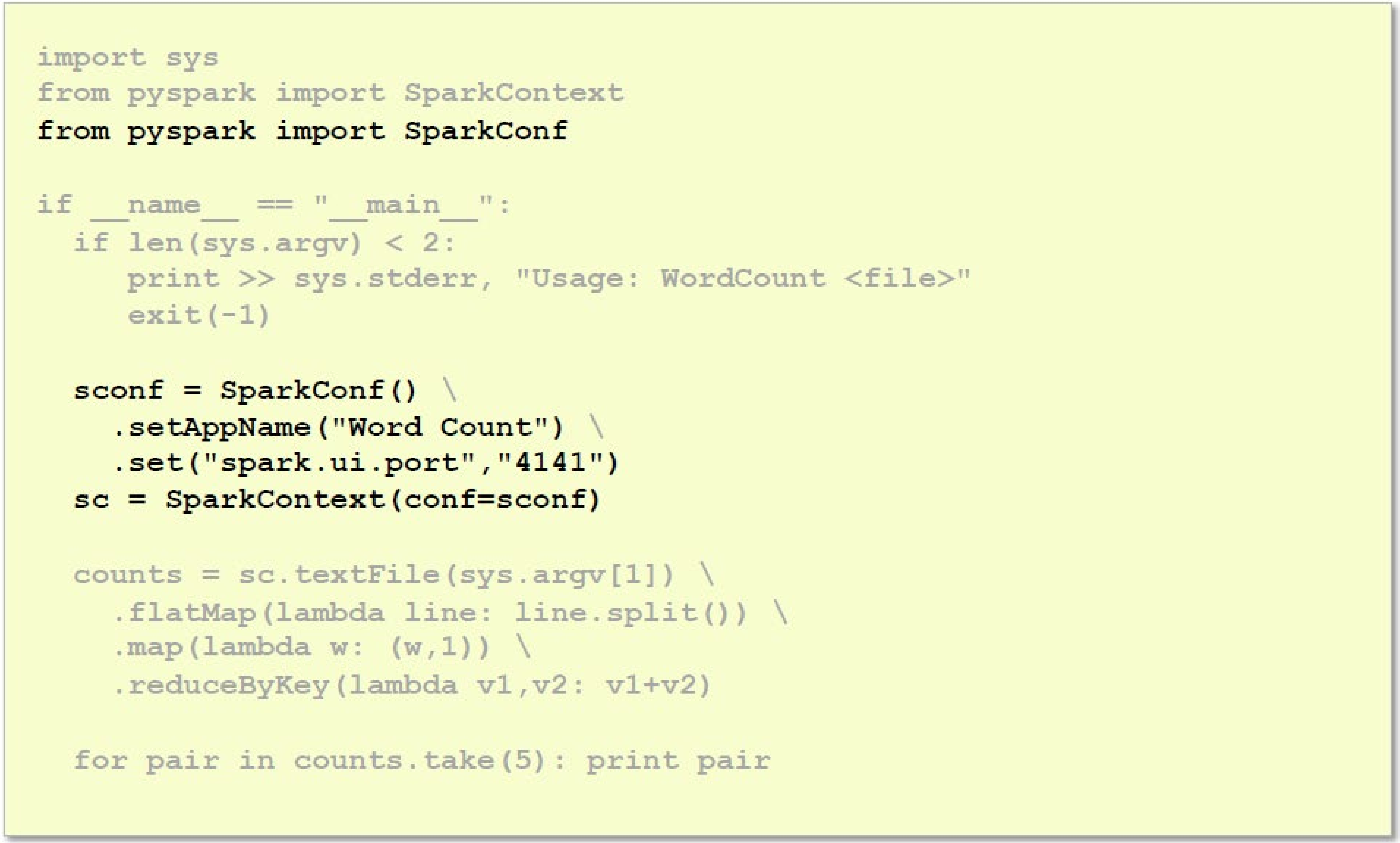
```
spark.master       spark://masternode:7077
spark.local.dir    /tmp
spark.ui.port      4444
```

- Site defaults properties file
  - `$SPARK_HOME/conf/spark-defaults.conf`
  - Template file provided

Setting Configuration Properties Programmatically

- Spark configuration settings are part of the SparkContext
- Configure using a SparkConf object
- Some example functions
  - `setAppName(`*`name`*`)`
  - `setMaster(`*`master`*`)`
  - `set(`*`property-name, value`*`)`
- `set` functions return a SparkConf object to support chaining

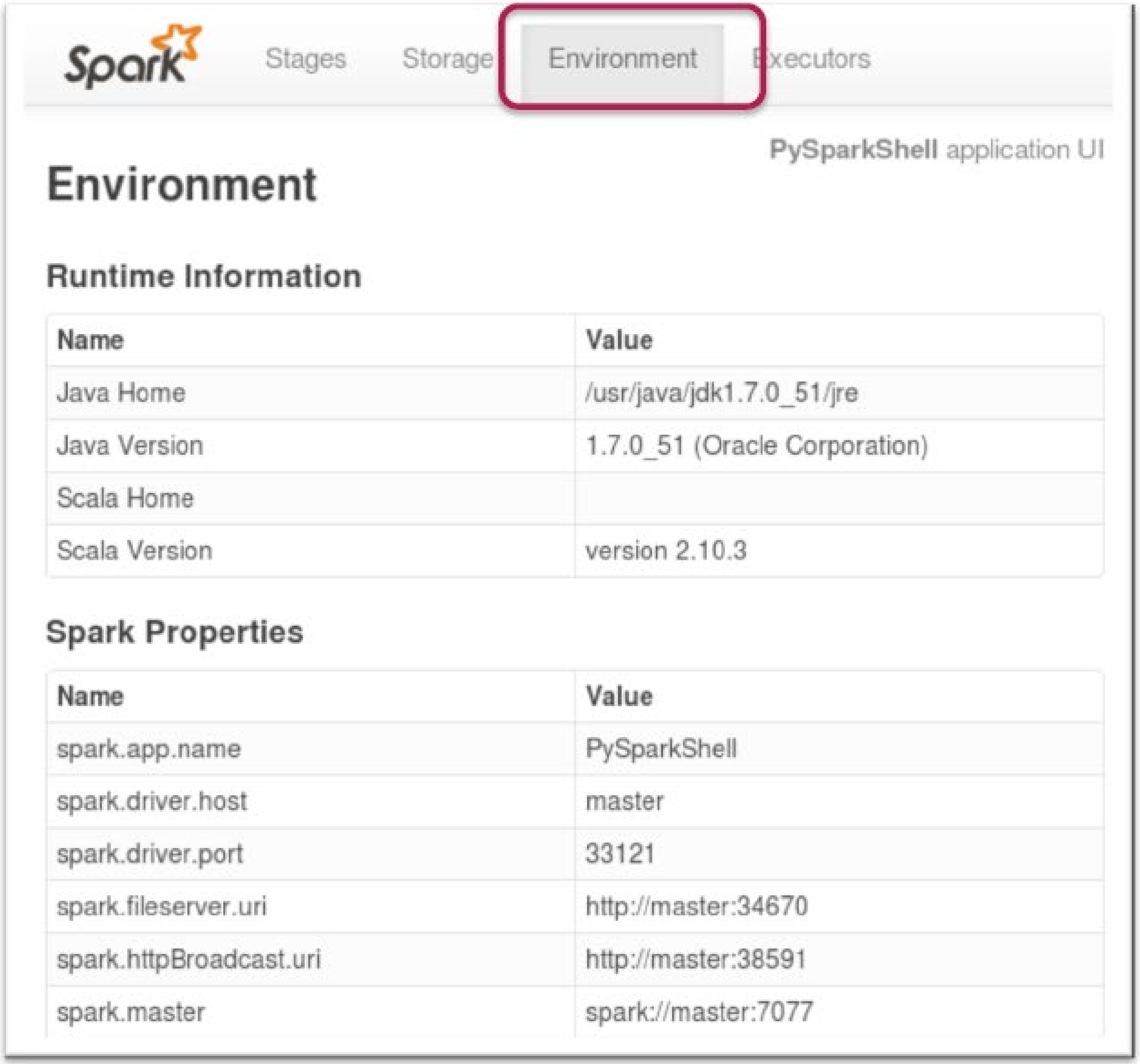
SparkConf Example



```
import sys
from pyspark import SparkContext
from pyspark import SparkConf

if __name__ == "__main__":
  if len(sys.argv) < 2:
     print >> sys.stderr, "Usage: WordCount <file>"
     exit(-1)

  sconf = SparkConf() \
    .setAppName("Word Count") \
    .set("spark.ui.port","4141")
  sc = SparkContext(conf=sconf)

  counts = sc.textFile(sys.argv[1]) \
    .flatMap(lambda line: line.split()) \
    .map(lambda w: (w,1)) \
    .reduceByKey(lambda v1,v2: v1+v2)

  for pair in counts.take(5): print pair
```

Viewing Spark Properties

- You can view the Spark property setting in the Spark Application UI



Environment

Runtime Information

Name	Value
Java Home	/usr/java/jdk1.7.0_51/jre
Java Version	1.7.0_51 (Oracle Corporation)
Scala Home	
Scala Version	version 2.10.3

Spark Properties

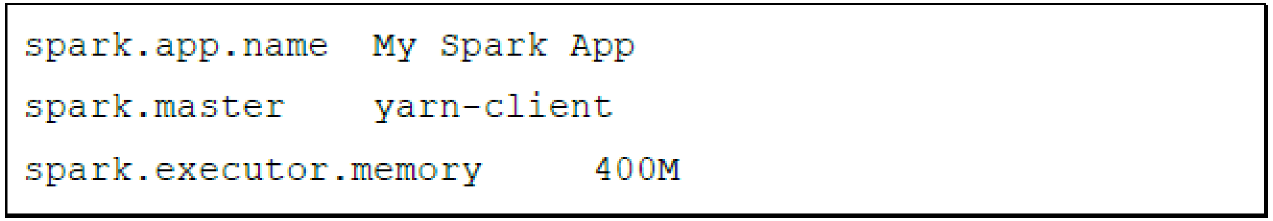
Name	Value
spark.app.name	PySparkShell
spark.driver.host	master
spark.driver.port	33121
spark.fileserver.uri	http://master:34670
spark.httpBroadcast.uri	http://master:38591
spark.master	spark://master:7077

Practice: Configure a Spark Application

- **Rerun the CountJPGs Python program you wrote in the previous exercise, this time specifying an application name.**
  - Use “--name“ option
- **Visit the Resource Manager UI again and note the application name listed is the one specified in the command line**

Set configuration options in a configuration file

- Change directories to your exercise working directory
- Using a text editor, create a file in the working directory called myspark.conf, containing settings for the properties shown below



```
spark.app.name  My Spark App
spark.master    yarn-client
spark.executor.memory     400M
```

- Re-run your application, this time using the properties file instead of using the script options to configure spark properties
- While the application is running, view the YARN UI and confirm that the Spark application name is correctly displayed as “My Spark APP”

Spark Logging

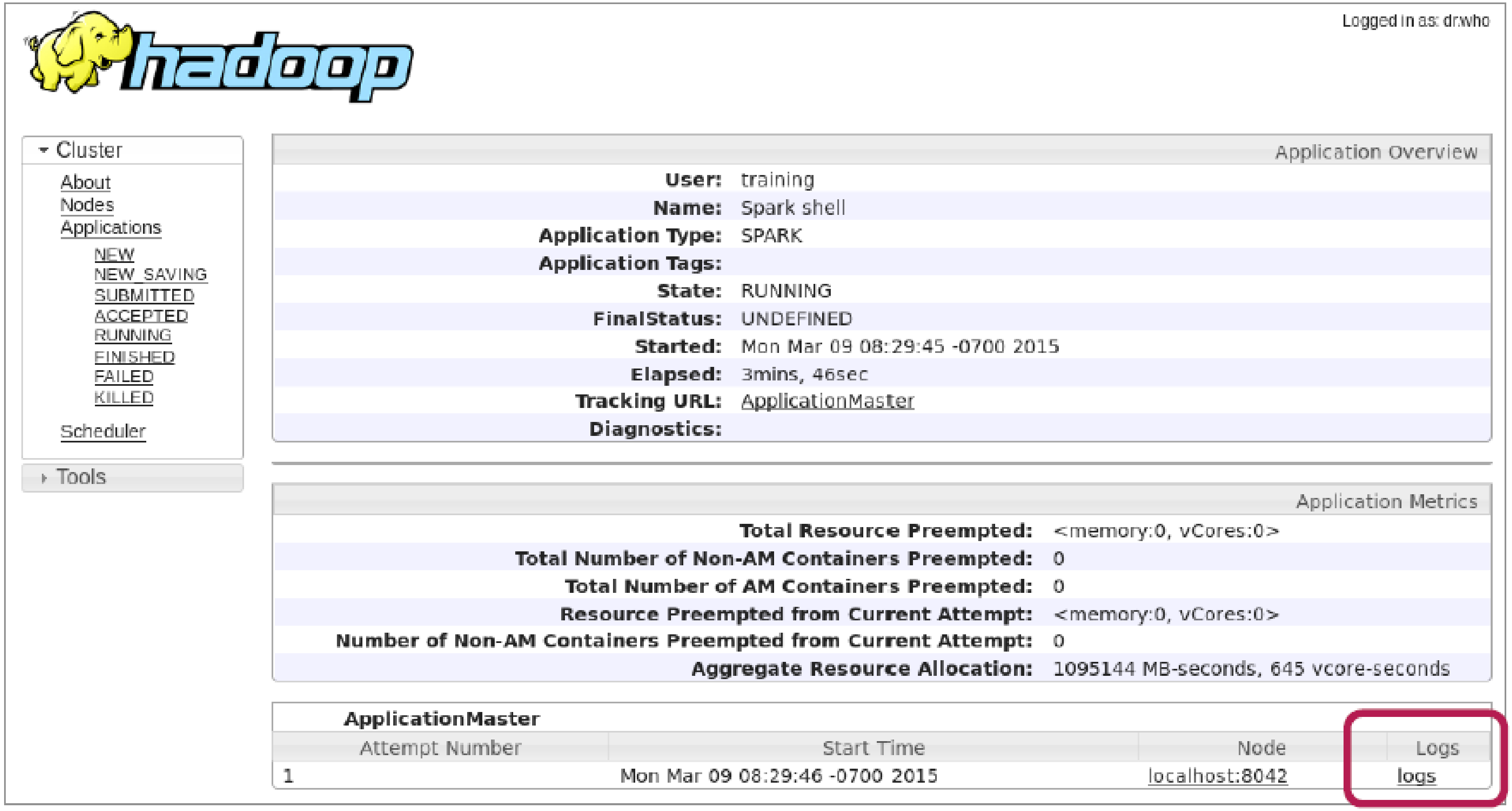
- **Spark uses Apache Log4j for logging**
  - Allows for controlling logging at runtime using a properties file
    - Enable or disable logging, set logging levels, select output destination
  - For more info see `http://logging.apache.org/log4j/1.2/`
- **Log4j provides several logging levels**
  - Fatal
  - Error
  - Warn
  - Info
  - Debug
  - Trace
  - Off

Spark Log Files (1)

- **Log file locations depend on your cluster management platform**
- YARN
  - If log aggregation off, logs are stored locally on each worker node
  - If log aggregation is on, logs are stored in HDFS
    - Default **`/var/log/hadoop-yarn`**
    - Access via **`yarn logs`** command or YARN RM UI

```
$ yarn application -list

Application-Id                  Application-Name Application-Type…
application_1441395433148_0003  Spark shell      SPARK    …
application_1441395433148_0001  myapp.jar        MAPREDUCE   …

$ yarn logs -applicationId <appid>
…
```

Spark Log Files (2)



Application Overview

User:	training
Name:	Spark shell
Application Type:	SPARK
Application Tags:	
State:	RUNNING
FinalStatus:	UNDEFINED
Started:	Mon Mar 09 08:29:45 -0700 2015
Elapsed:	3mins, 46sec
Tracking URL:	ApplicationMaster
Diagnostics:	

Application Metrics

Total Resource Preempted:	<memory:0, vCores:0>
Total Number of Non-AM Containers Preempted:	0
Total Number of AM Containers Preempted:	0
Resource Preempted from Current Attempt:	<memory:0, vCores:0>
Number of Non-AM Containers Preempted from Current Attempt:	0
Aggregate Resource Allocation:	1095144 MB-seconds, 645 vcore-seconds

ApplicationMaster

Attempt Number	Start Time	Node	Logs
1	Mon Mar 09 08:29:46 -0700 2015	localhost:8042	logs

Configuring Spark Logging (1)

- Logging levels can be set for the cluster, for individual applications, or even for specific components or subsystems
- Default for machine: `$SPARK_HOME/conf/log4j.properties`
  - Start by copying `log4j.properties.template`

log4j.properties.template

```
# Set everything to be logged to the console
log4j.rootCategory=INFO, console
log4j.appender.console=org.apache.log4j.ConsoleAppender
log4j.appender.console.target=System.err
…
```

Configuring Spark Logging (2)

- **Spark will use the first `log4j.properties` file it finds in the Java classpath**

- **Spark Shell will read `log4j.properties` from the current directory**
  - Copy **`log4j.properties`** to the working directory and edit

…*my-working-directory*/log4j.properties

```
# Set everything to be logged to the console
log4j.rootCategory=DEBUG, console
log4j.appender.console=org.apache.log4j.ConsoleAppender
log4j.appender.console.target=System.err
…
```

Practice: Set Logging Levels

- **Copy the template file /etc/sparkconf/log4j.properties.template to log4j.properties in your exercise working directory**
- **Edit log4j.properties. The first line currently reads**

```
log4j.rootCategory=INFO, console
```

Replace INFO with DEBUG

```
log4j.rootCategory=DEBUG, console
```

- **Rerun your Spark application. Because the current directory is on the Java classpath, your log4j.properties file will set the logging level to DEBUG**
- **Notice that the output now contains both the INFO messages it did before and DEBUG messages. Debug logging can be useful when debugging, testing, or optimizing your code, but in most cases generates unnecessarily distracting output**

- Edit the log4j. Properties file to replace DEBUG with WARN and try again. This time notice that no INFO or DEBUG messages are displayed, only WARN messages.
- You can also set the log level for the Spark Shell by placing the log4j.properties file in your working directory before starting the Spark shell. Try starting the shell from the directory in which you placed the file and note that only WARN messages now appear.

Essential Points (1)

- Use the Spark Shell application for interactive data exploration
- Write a Spark application to run independently
- Spark applications require a Spark Context object
- Spark applications are run using the `spark-submit` script
- Spark configuration parameters can be set at runtime using the `spark-submit` script or programmatically using a `SparkConf` object
- Spark uses log4j for logging
  - Configure using a `log4j.properties` file

Essential Points (2)

- **Spark is designed to run on a cluster**
  - Spark includes a basic cluster management platform called Spark Standalone
  - Can also run on Hadoop YARN and Mesos
- **The master distributes tasks to individual workers in the cluster**
  - Tasks run in *executors* – JVMs running on worker nodes
- **Spark clusters work closely with HDFS**
  - Tasks are assigned to workers where the data is physically stored when possible