



PARTNERSHIP FOR ADVANCED COMPUTING IN EUROPE

# Introduction to Apache Spark

---

Slavko Žitnik

*University of Ljubljana, Faculty for computer and information science*



PARTNERSHIP FOR ADVANCED COMPUTING IN EUROPE

## Agenda

- ▶ About Apache Spark
- ▶ Spark execution modes
- ▶ Basic Spark data structures (RDDs)
- ▶ Hands-on tutorial
  - ▶ RDDs and operations
  - ▶ DataFrames, User defined functions and SparkSQL
- ▶ Hands-on lab exercises – Jupyter notebooks
- ▶ Hands-on lab exercise – Spark on an HPC
  - ▶ Apache Spark deployment using Slurm
- ▶ Challenge exercises (independent work) & debug session



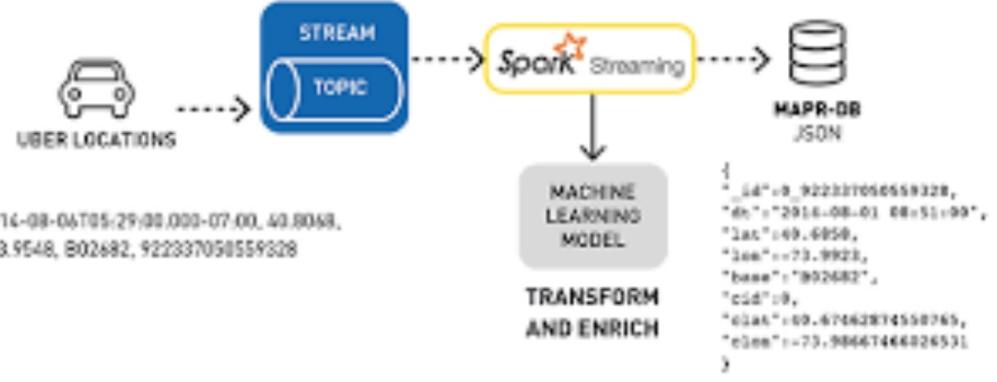
PARTNERSHIP FOR ADVANCED COMPUTING IN EUROPE

## About Apache Spark

- ▶ Fast, expressive, general-purpose in-memory cluster computing framework compatible with Apache Hadoop and built around speed, ease of use and streaming analytics
- ▶ Faster and easier than Hadoop MapReduce\*
- ▶ Large community and 3rd party libraries
- ▶ Provides high-level APIs (Java, Scala, Python, R)
- ▶ Supports variety of workloads
  - ▶ interactive queries, streaming, machine learning and graph processing

## Apache Spark Use cases

- ▶ Logs processing (Uber)
- ▶ Event detection and real-time analysis
- ▶ Interactive analysis
- ▶ Latency reduction
- ▶ Advanced ad-targeting (Yahoo!)
- ▶ Recommendation systems (Netflix, Pinterest)
- ▶ Fraud detection
- ▶ Sentiment analysis (Twitter)
- ▶ ...



## Apache Spark Use cases

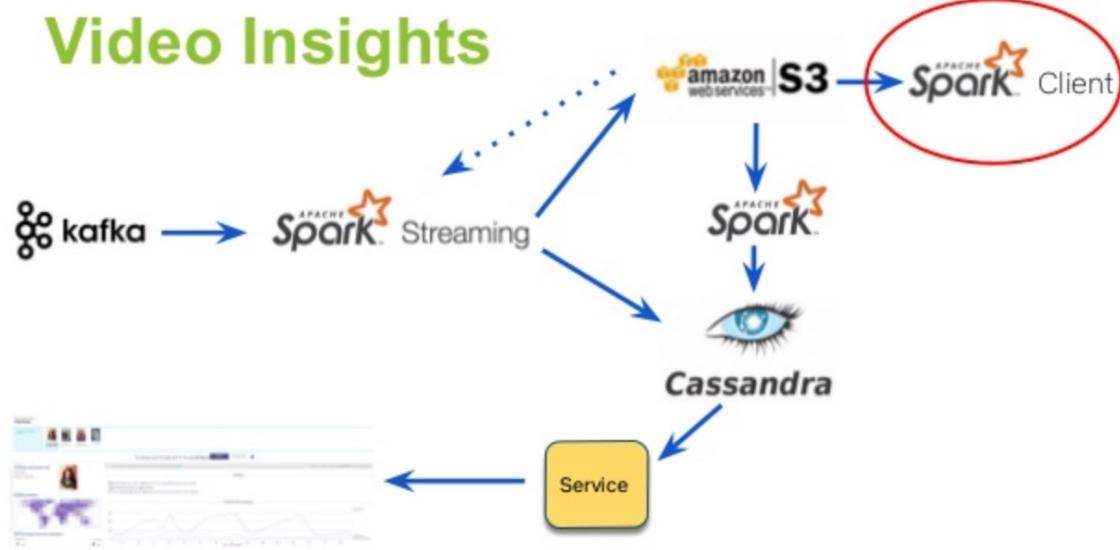
- ▶ Logs processing (Uber)
- ▶ Event detection and real-time analysis
- ▶ Interactive analysis
- ▶ Latency reduction
- ▶ Advanced ad-targeting (Yahoo!)
- ▶ Recommendation systems (Netflix, Pinterest)
- ▶ Fraud detection
- ▶ Sentiment analysis (Twitter)
- ▶ ...



## Apache Spark Use cases

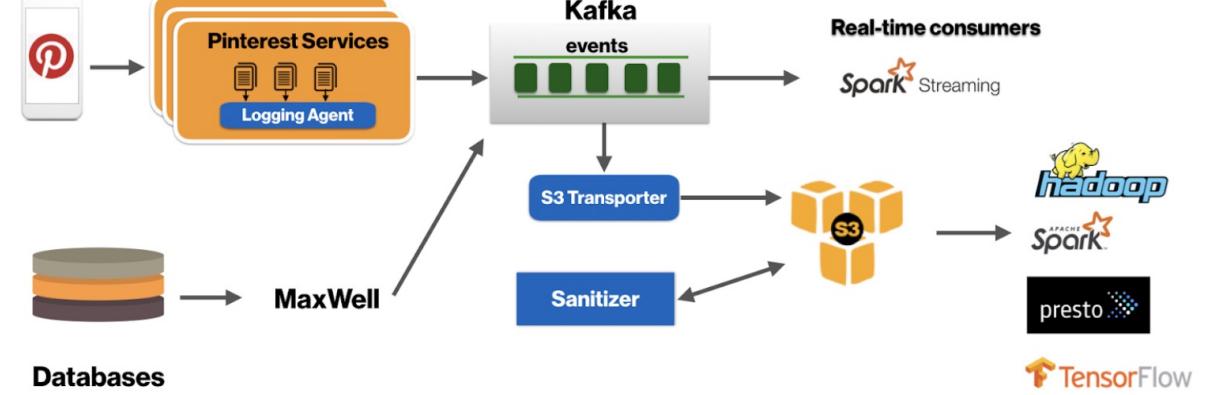
- ▶ Logs processing (Uber)
- ▶ Event detection and real-time analysis
- ▶ Interactive analysis
- ▶ Latency reduction
- ▶ Advanced ad-targeting (Yahoo!)
- ▶ Recommendation systems (Netflix, Pinterest)
- ▶ Fraud detection
- ▶ Sentiment analysis (Twitter)
- ▶ ...

## Video Insights

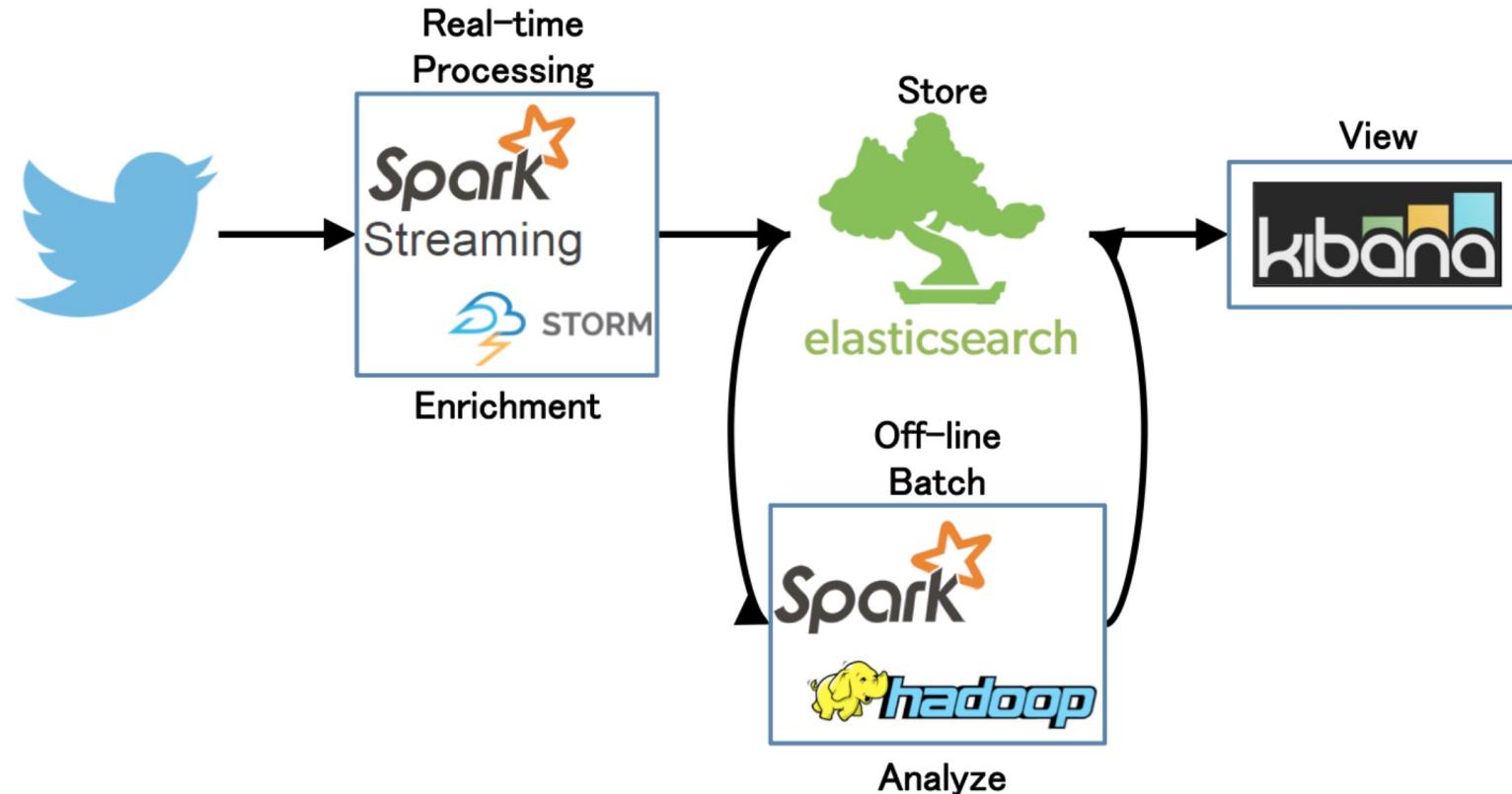


## Apache Spark Use cases

- ▶ Logs processing (Uber)
- ▶ Event detection and real-time analysis
- ▶ Interactive analysis
- ▶ Latency reduction
- ▶ Advanced ad-targeting (Yahoo!)
- ▶ Recommendation systems (Netflix, Pinterest)
- ▶ Fraud detection
- ▶ Sentiment analysis (Twitter)
- ▶ ...

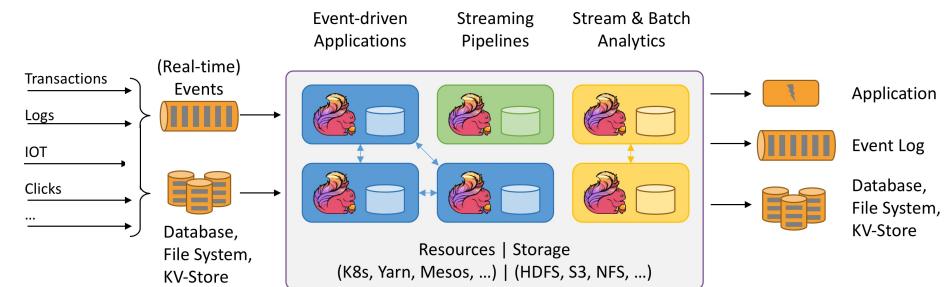
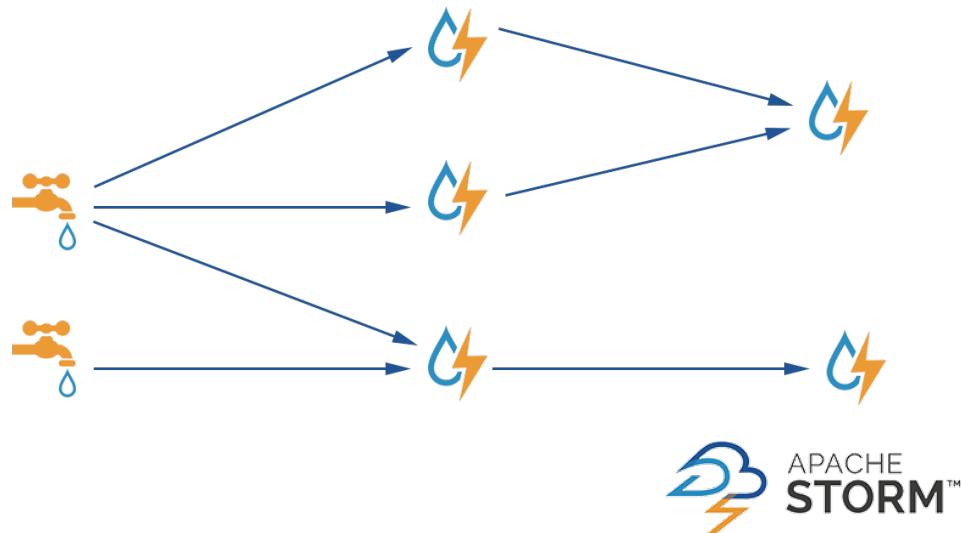
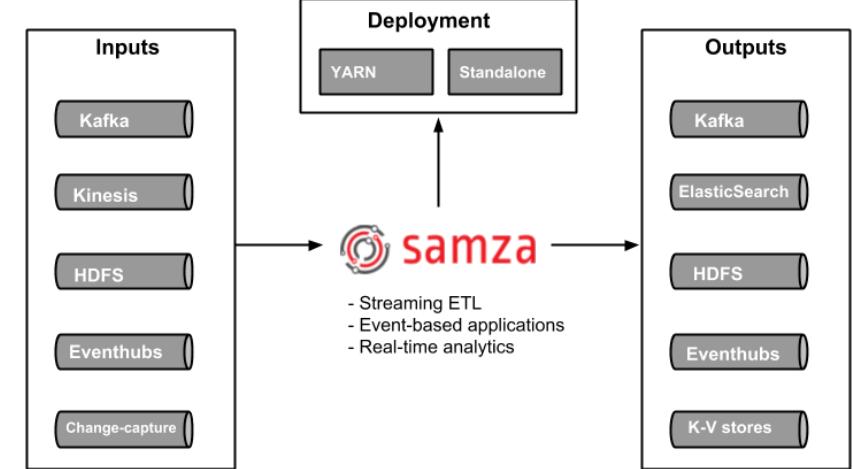


## Apache Spark general setup: Twitter sentiment analysis



## Apache Spark alternatives

- ▶ Apache Samza (library/framework)
- ▶ Apache Storm (real-time stream processing)
- ▶ Apache Flink (native streaming support for all workloads)

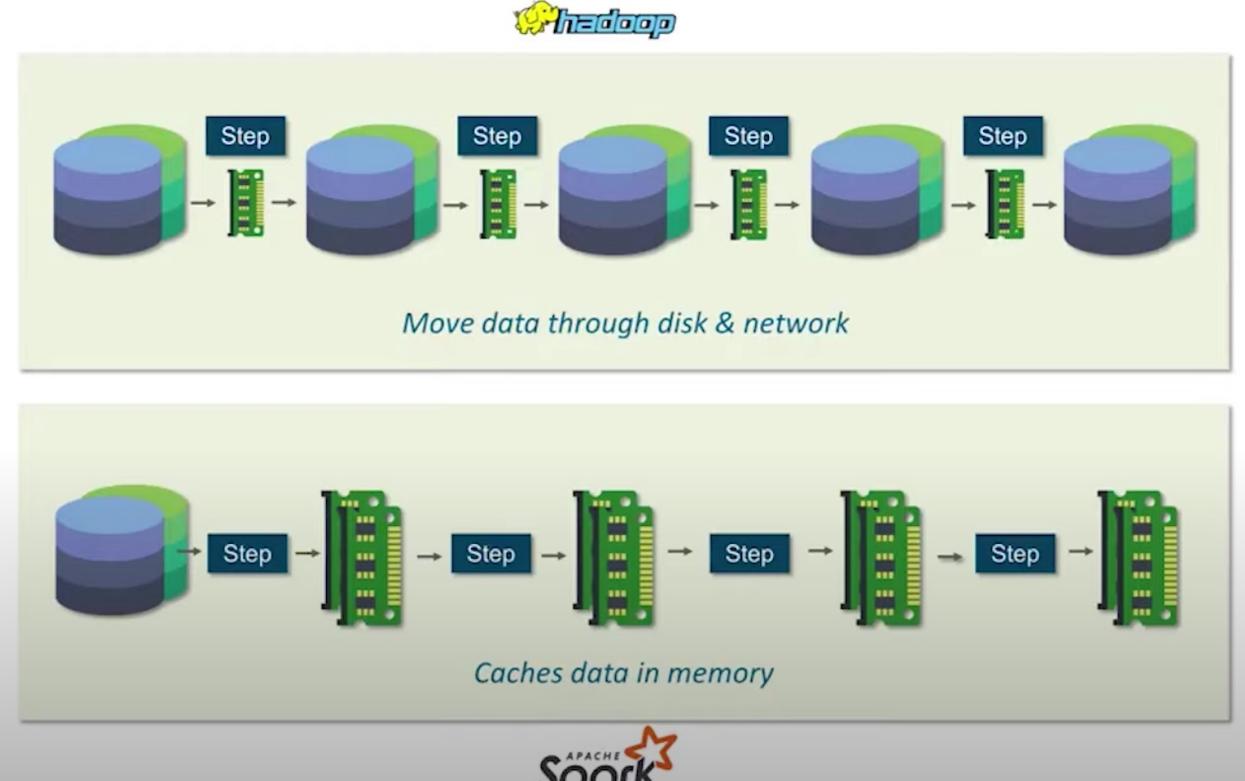


## Apache Spark 3

- ▶ Latest release – v3.1.1. (March 2, 2021)
- ▶ Improvements over Spark 2 (v2.4.1)
  - ▶ Python 2 deprecated
  - ▶ Adaptive execution of Spark SQL (merging intermediate results among workers)
  - ▶ Dynamic partition pruning
  - ▶ Support for **deep learning** (GPU support for Nvidia, AMD, Intel)
  - ▶ Better Kubernetes integration
  - ▶ Graph features (Morpheus as extension of Cypher, neo4j support)
  - ▶ ACID transactions (for Delta Lake storage)
  - ▶ Apache Arrow data format integration (columnar format for analytics)
  - ▶ ...

## Hadoop MapReduce vs. Apache Spark

- ▶ Big data frameworks
  - ▶ Performance
  - ▶ Ease of use
  - ▶ Costs
  - ▶ Data processing
  - ▶ Fault tolerance
  - ▶ Security
- ▶ Hadoop
  - ▶ Archival data analysis
- ▶ Spark
  - ▶ Real-time data analysis



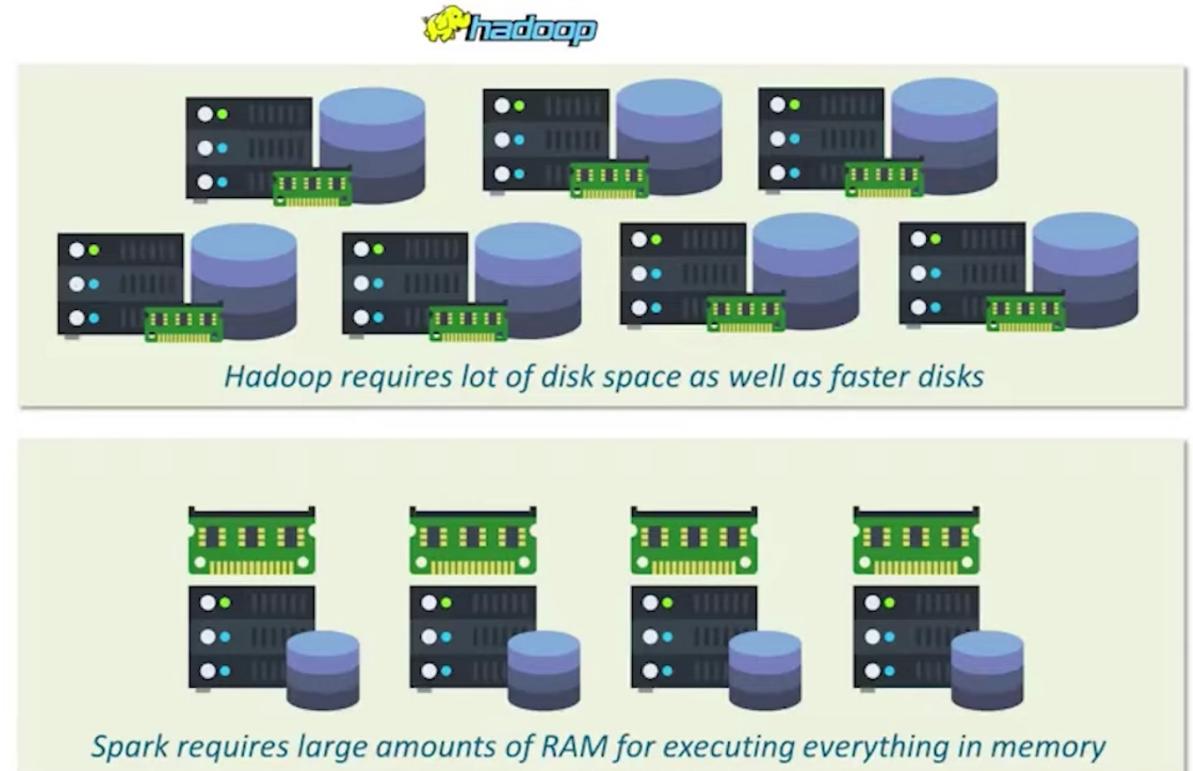
## Hadoop MapReduce vs. Apache Spark

- ▶ Big data frameworks
  - ▶ Performance
  - ▶ Ease of use
  - ▶ Costs
  - ▶ Data processing
  - ▶ Fault tolerance
  - ▶ Security
- ▶ Hadoop
  - ▶ Archival data analysis
- ▶ Spark
  - ▶ Real-time data analysis



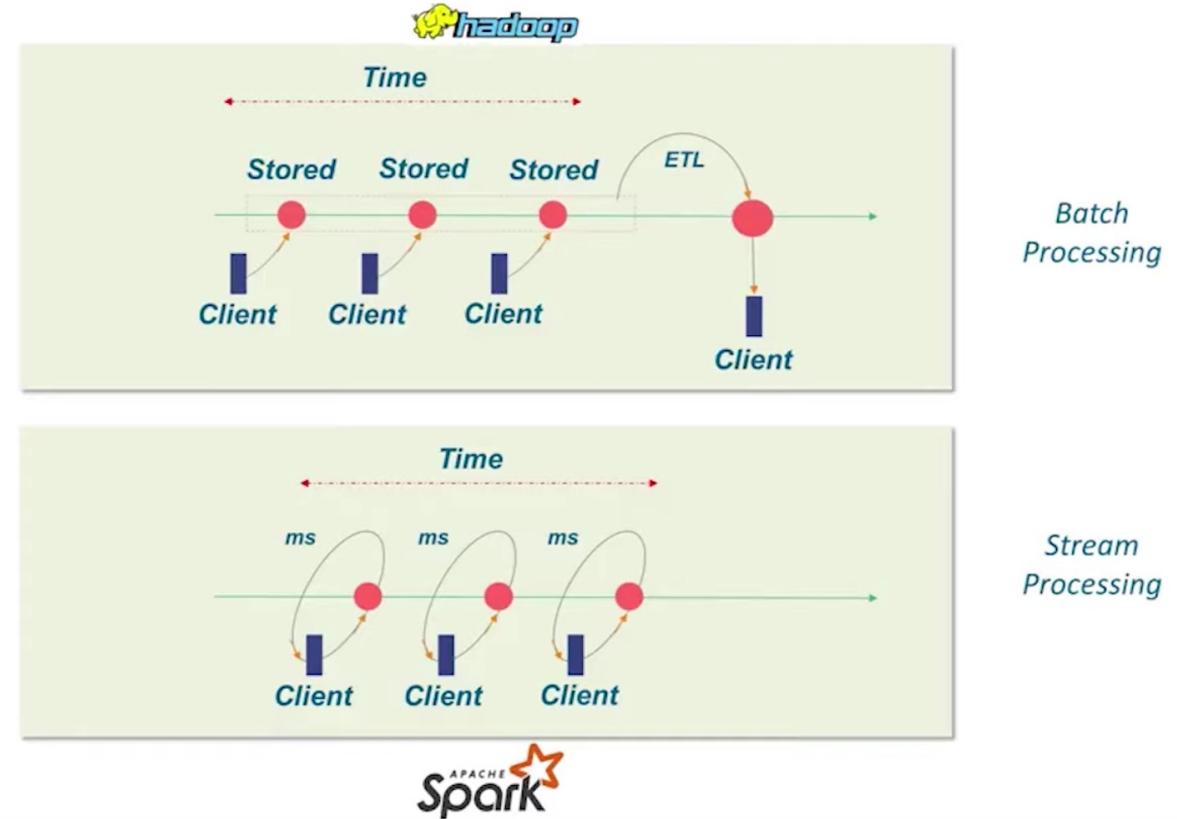
## Hadoop MapReduce vs. Apache Spark

- ▶ Big data frameworks
  - ▶ Performance
  - ▶ Ease of use
  - ▶ Costs
  - ▶ Data processing
  - ▶ Fault tolerance
  - ▶ Security
- ▶ Hadoop
  - ▶ Archival data analysis
- ▶ Spark
  - ▶ Real-time data analysis



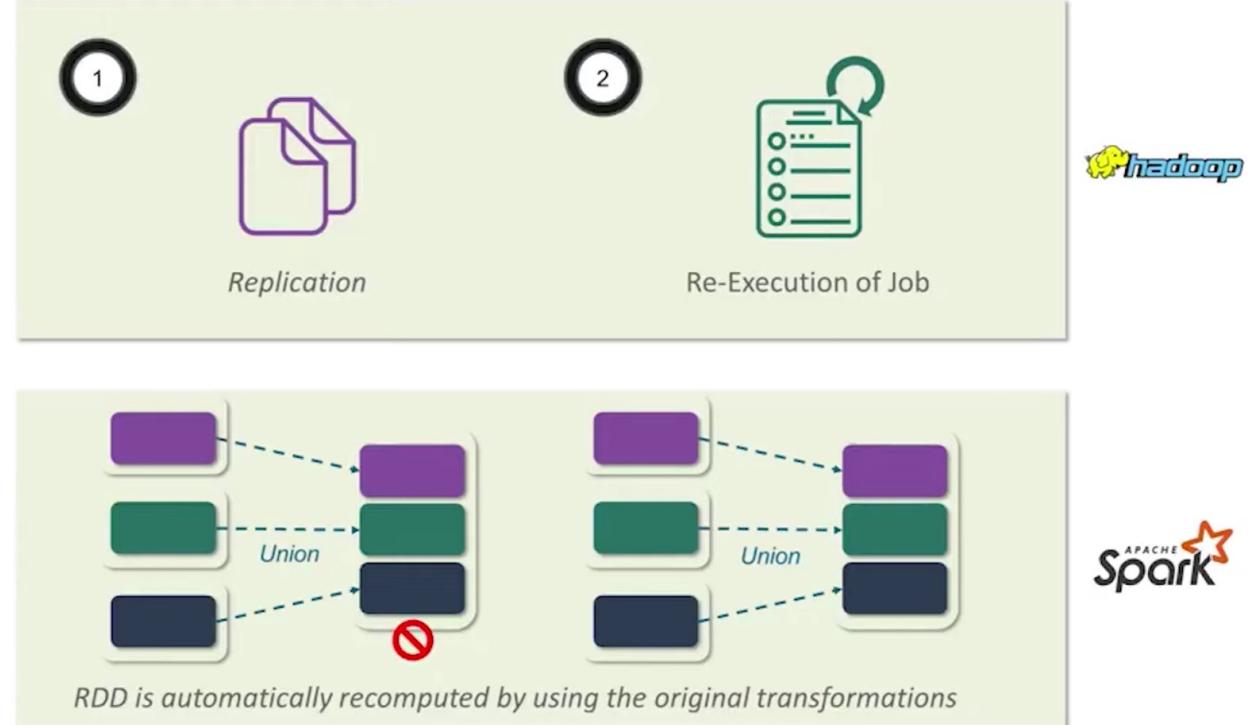
## Hadoop MapReduce vs. Apache Spark

- ▶ Big data frameworks
  - ▶ Performance
  - ▶ Ease of use
  - ▶ Costs
  - ▶ Data processing
  - ▶ Fault tolerance
  - ▶ Security
- ▶ Hadoop
  - ▶ Archival data analysis
- ▶ Spark
  - ▶ Real-time data analysis



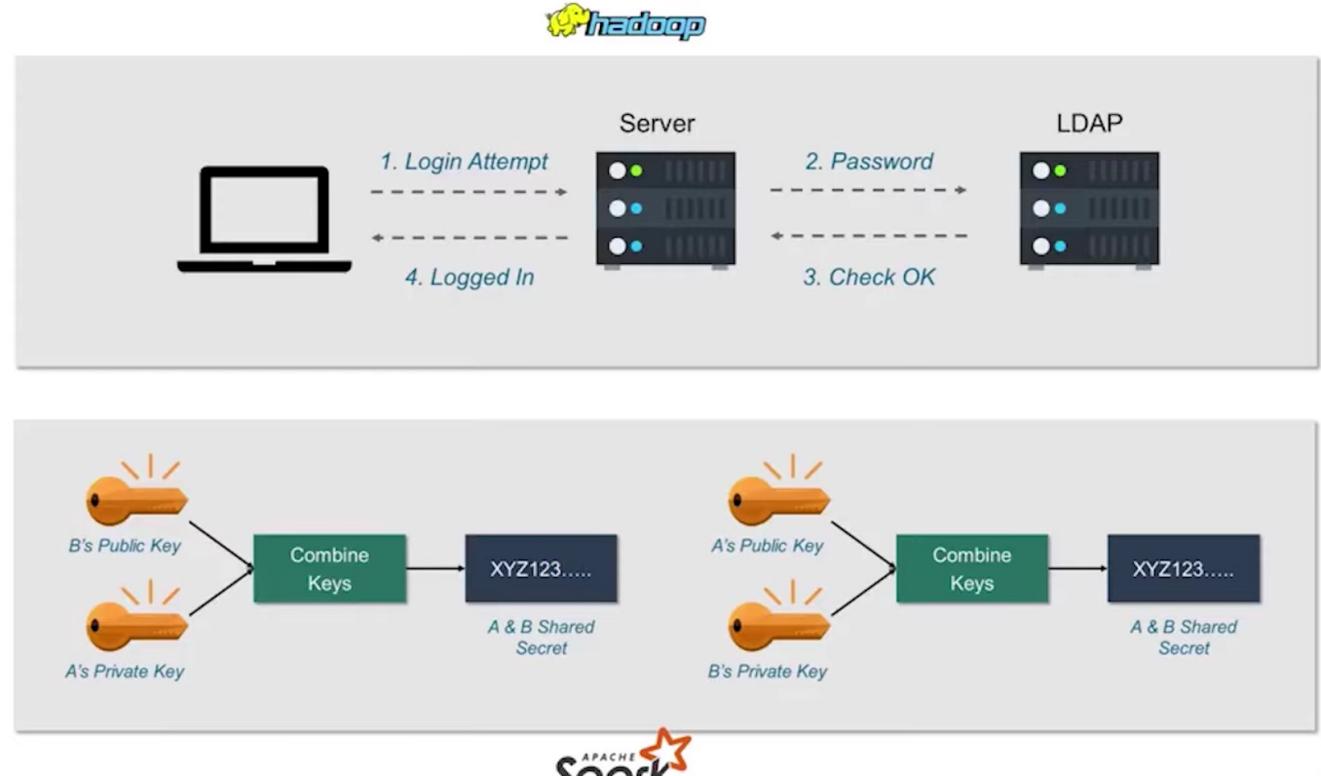
## Hadoop MapReduce vs. Apache Spark

- ▶ Big data frameworks
  - ▶ Performance
  - ▶ Ease of use
  - ▶ Costs
  - ▶ Data processing
  - ▶ Fault tolerance
  - ▶ Security
- ▶ Hadoop
  - ▶ Archival data analysis
- ▶ Spark
  - ▶ Real-time data analysis



## Hadoop MapReduce vs. Apache Spark

- ▶ Big data frameworks
  - ▶ Performance
  - ▶ Ease of use
  - ▶ Costs
  - ▶ Data processing
  - ▶ Fault tolerance
  - ▶ Security
- ▶ Hadoop
  - ▶ Archival data analysis
- ▶ Spark
  - ▶ Real-time data analysis



## Apache Spark ecosystem

Spark SQL

Spark Streaming

Machine learning

GraphX

3rd party library

Apache Spark Core

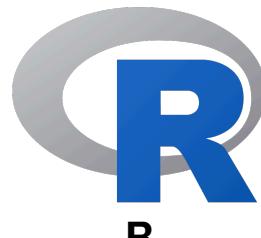
*Standalone scheduler*

EC2

Hadoop YARN

Apache Mesos

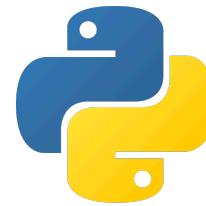
Kubernetes



R



Java



Python



Scala

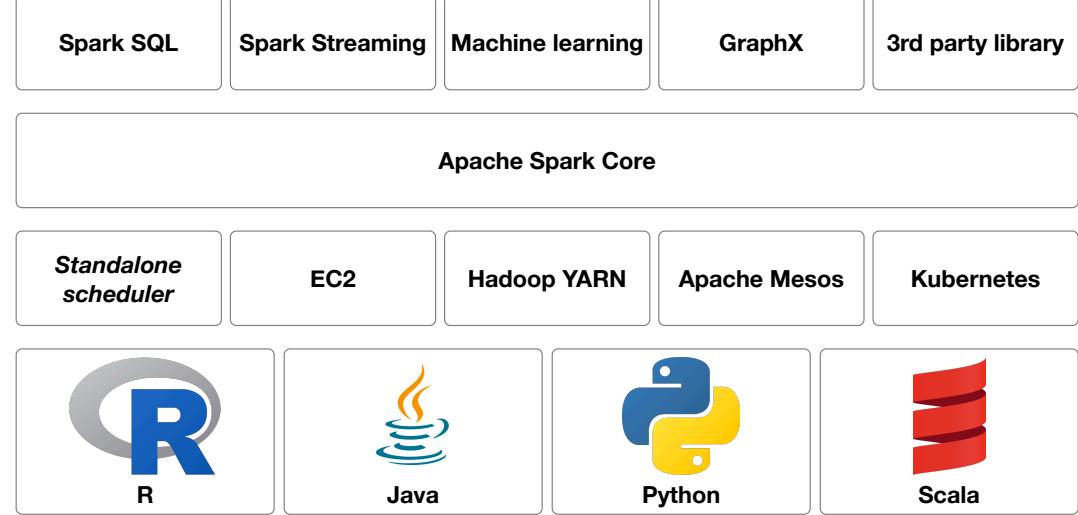
## Spark ecosystem: Spark core

### ► Core functionalities

- ▶ task scheduling
- ▶ memory management
- ▶ fault recovery
- ▶ storage systems interaction
- ▶ etc.

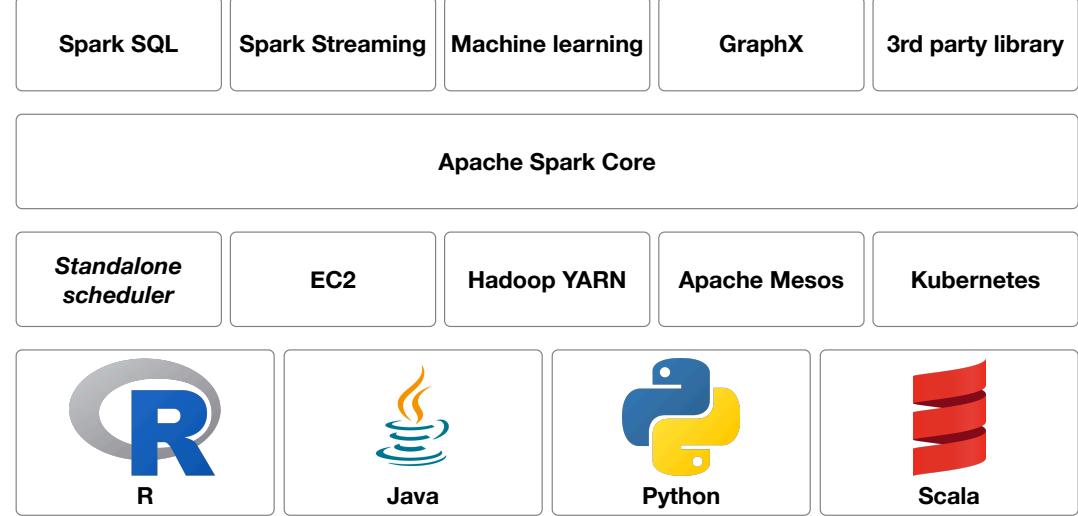
### ► Basic data structure definitions/abstractions

- ▶ Resilient Distributed Data sets (RDDs)
  - ▶ main Spark data structure
- ▶ Directed Acyclic Graph (DAG)



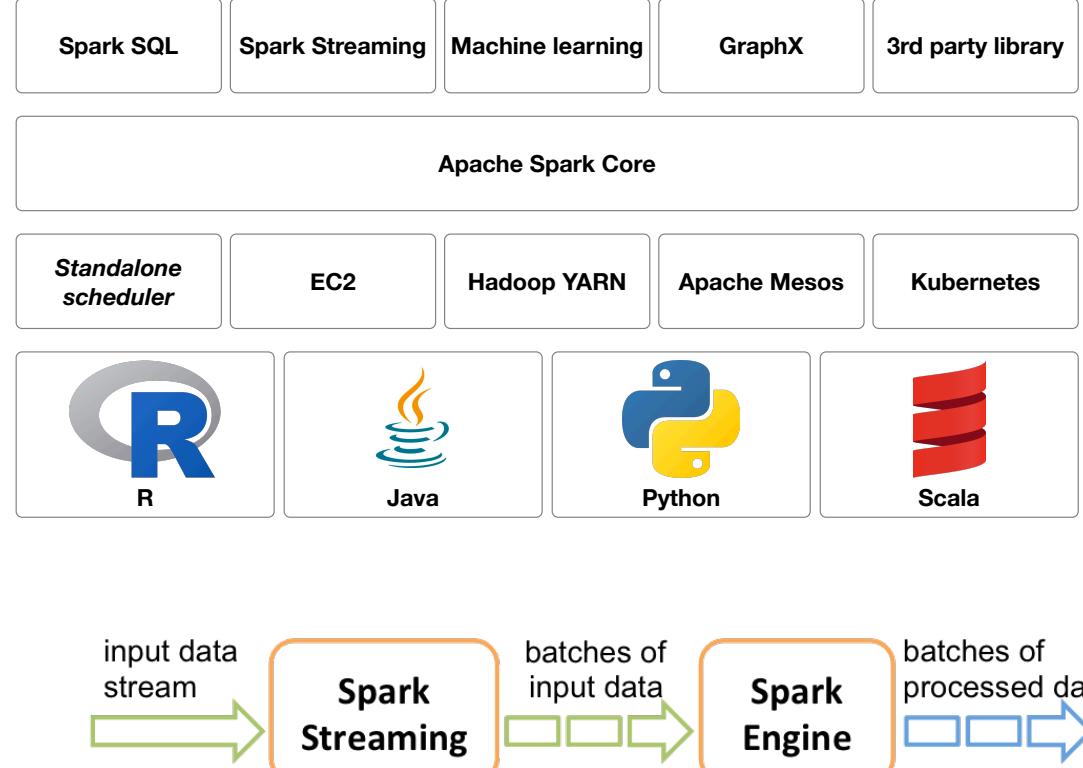
## Spark ecosystem: Spark SQL

- ▶ Structured data manipulation
  - ▶ Data Frames definition
- ▶ Table-like data representation
  - ▶ RDDs extension
  - ▶ Schema definition
- ▶ SQL queries execution
- ▶ Native support for schema-based data
  - ▶ Hive, Parquet, JSON, CSV



## Spark ecosystem: Streaming

- ▶ Data analysis of streaming data
  - ▶ e.g. tweets, log messages
- ▶ Features of stream processing
  - ▶ High-throughput
  - ▶ Fault-tolerant
  - ▶ End-to-end
  - ▶ Exactly-once
- ▶ High-level abstraction of a discretized stream
  - ▶ Dstream represented as a sequence of RDDs
- ▶ Spark 2.3+ , Continuous Processing
  - ▶ end-to-end latencies as low as 1ms



[Spark SQL](#)
[Spark Streaming](#)
[Machine learning](#)
[GraphX](#)
[3rd party library](#)
**Apache Spark Core**
[Standalone scheduler](#)
[EC2](#)
[Hadoop YARN](#)
[Apache Mesos](#)
[Kubernetes](#)


## Spark ecosystem: MLlib

- ▶ Common ML functionalities
  - ▶ ML Algorithms
    - ▶ common learning algorithms such as classification, regression, clustering, and collaborative filtering
  - ▶ Featurization
    - ▶ feature extraction, transformation, dimensionality reduction, and selection
  - ▶ Pipelines
    - ▶ tools for constructing, evaluating, and tuning ML Pipelines
  - ▶ Persistence
    - ▶ saving and load algorithms, models, and Pipelines
  - ▶ Utilities
    - ▶ linear algebra, statistics, data handling, etc.
- ▶ Two APIs
  - ▶ RDD-based API (`spark.mllib package`)
  - ▶ Spark 2.0+, DataFrame-based API (`spark.ml package`)
- ▶ Methods scale out across the cluster by default

[Spark SQL](#)
[Spark Streaming](#)
[Machine learning](#)
[GraphX](#)
[3rd party library](#)
**Apache Spark Core**
[Standalone scheduler](#)
[EC2](#)
[Hadoop YARN](#)
[Apache Mesos](#)
[Kubernetes](#)


R



Java



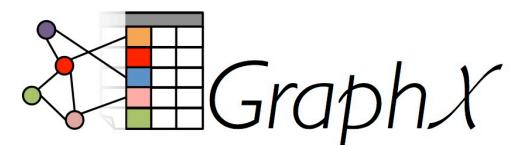
Python



Scala

## Spark ecosystem: GraphX

- ▶ Support for graphs and graph-parallel computation
  - ▶ Extension of RDDs (Graph)
    - ▶ directed multigraph with properties on vertices and edges
- ▶ Graph computation operators
  - ▶ subgraph, joinVertices, and aggregateMessages, etc.
  - ▶ PregelAPI support





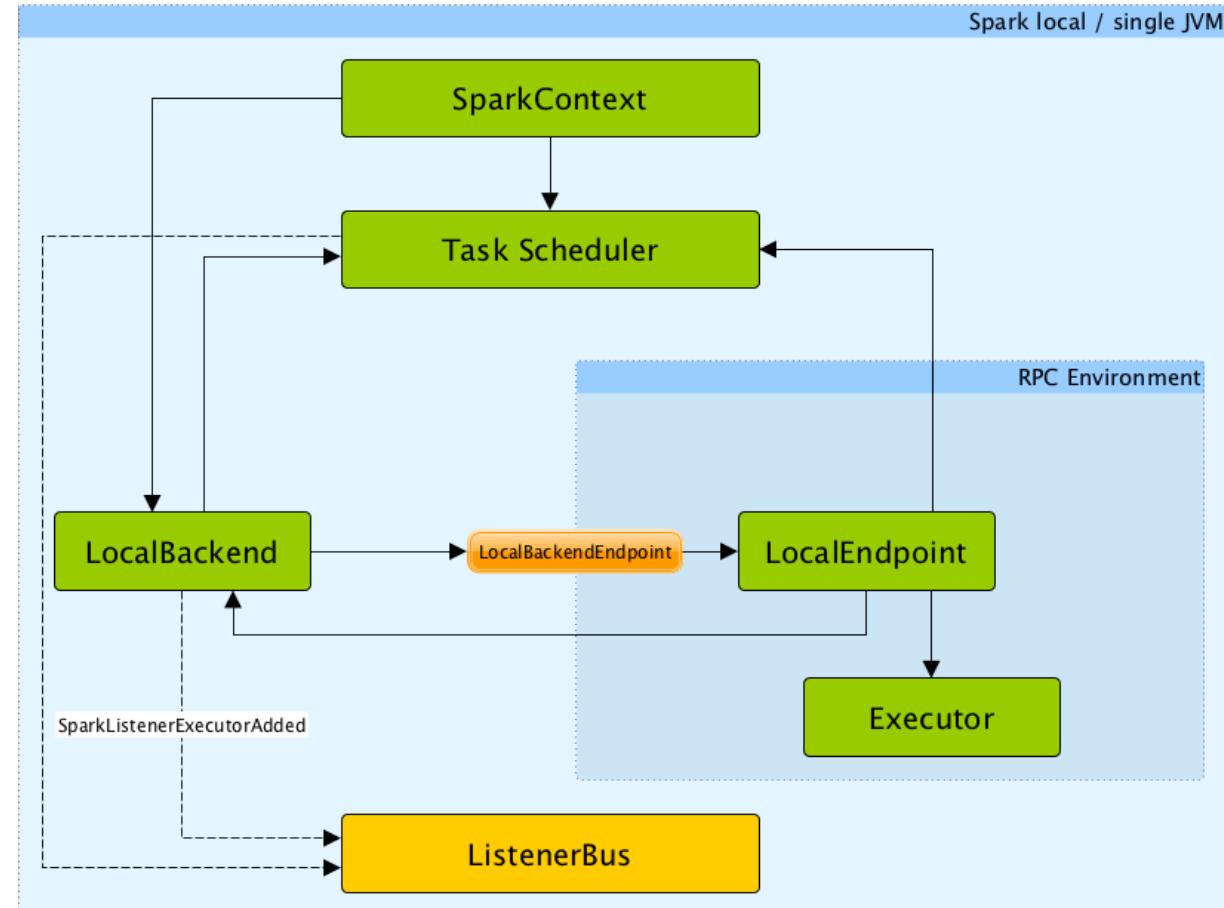
PARTNERSHIP FOR ADVANCED COMPUTING IN EUROPE

## Spark Execution modes

- ▶ Local mode
  - ▶ „Pseudo-cluster“ ad-hoc setup using script
- ▶ Cluster mode
  - ▶ Running via cluster manager
- ▶ Interactive mode
  - ▶ Direct manipulation in a shell (*pyspark*, *spark-shell*)

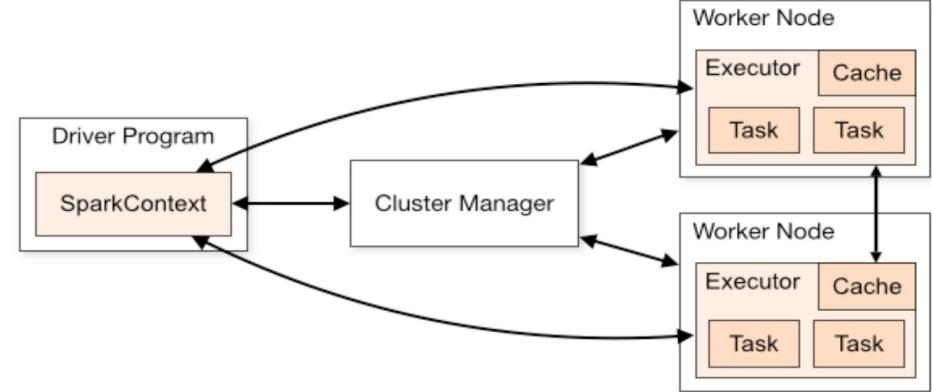
## Spark execution modes Local mode

- ▶ Non-distributed single-JVM deployment mode
- ▶ Spark library spawns (in a JVM)
  - ▶ driver
  - ▶ scheduler
  - ▶ master
  - ▶ executor
- ▶ Parallelism is the number of threads defined by a parameter N in a spark master URL
  - ▶ *local[N]*



## Spark execution modes Cluster mode

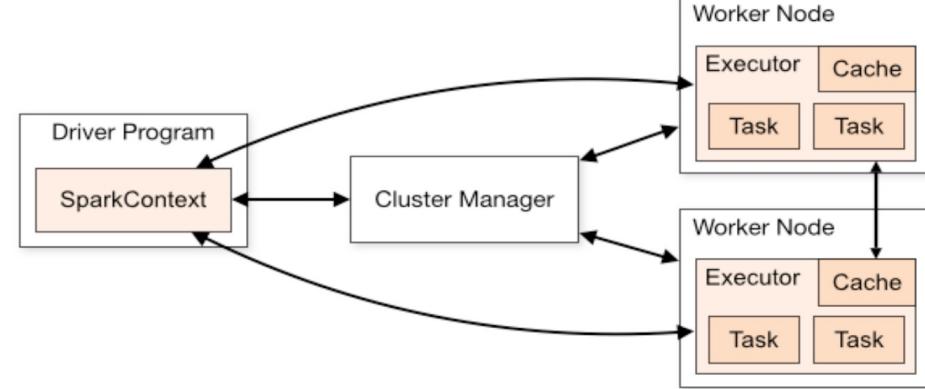
- ▶ Deployment on a private cluster
  - ▶ Apache Mesos
  - ▶ Hadoop YARN
  - ▶ Kubernetes
  - ▶ Standalone mode, ...



## Spark execution modes Cluster mode

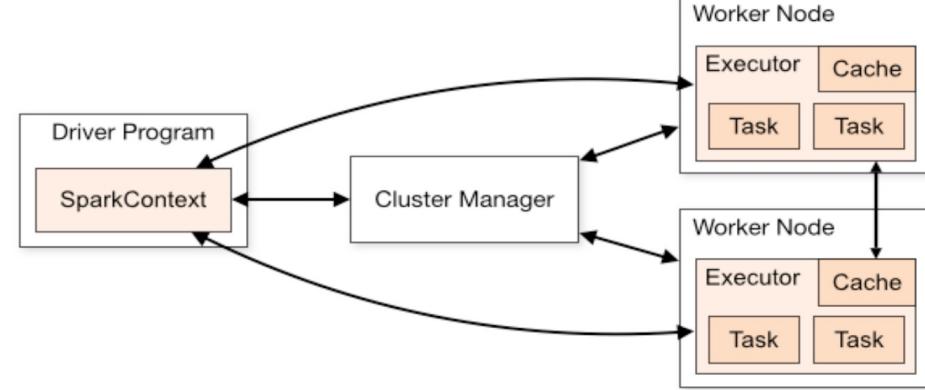
- ▶ Components

- ▶ Worker
  - ▶ Node in a cluster, managed by an executor
  - ▶ Executor manages computation, storage and caching
- ▶ Cluster manager
  - ▶ Allocates resources via SparkContext with Driver program
- ▶ Driver program
  - ▶ A program holding SparkContext and main code to execute in Spark
  - ▶ Sends application code to executors to execute
  - ▶ Listens to incoming connections from executors



## Spark execution modes Cluster mode

- ▶ Deploy modes (*standalone clusters*)
  - ▶ Client mode (default)
    - ▶ Driver runs in the same process as client that submits the app
  - ▶ Cluster mode
    - ▶ Driver launched from a worker process
    - ▶ Client process exits immediately after application submission



## Spark Execution process

### 1. Data preparation/import

- ▶ RDDs creation – i.e. parallel dataset with partitions

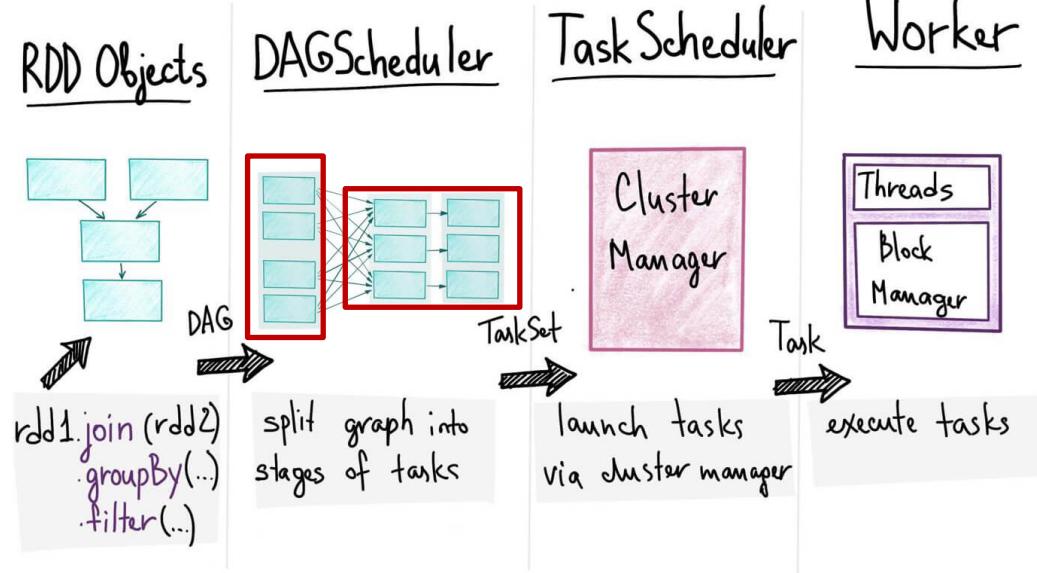
### 2. Transformations/actions definition\*

- ▶ Creation of tasks (units of work) sent to one executor
- ▶ Job is a set of tasks executed by an action\*

### 3. Creation of a directed acyclic graph (DAG)

- ▶ Contains a graph of RDD operations
- ▶ Definition of stages – set of tasks to be executed in parallel (i.e. at a partition level)

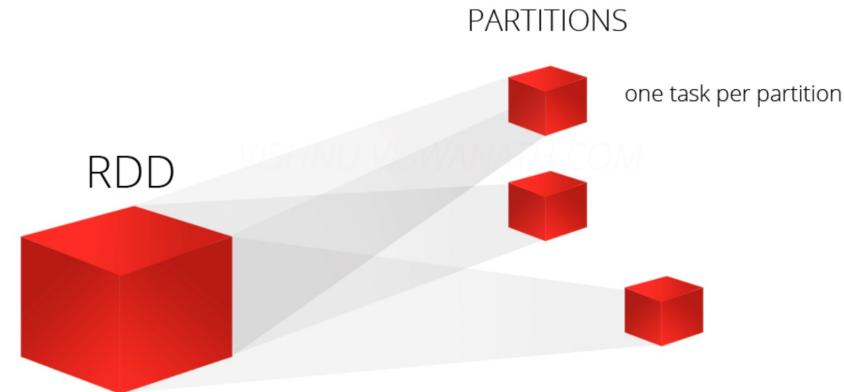
### 4. Execution of a program



@luminousmen.com

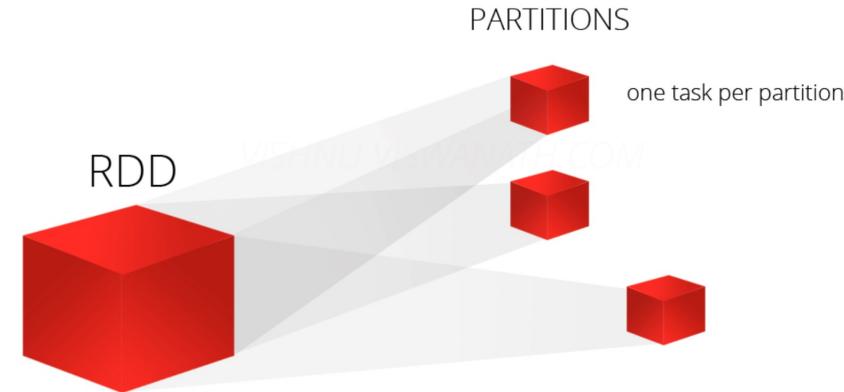
## Spark Programming concepts (Resilient distributed datasets - RDDs)

- ▶ Basic data representation in Spark
- ▶ A distributed collection of items – *partitions*
  - ▶ Enables operations to be performed in parallel
- ▶ Immutable (read-only)
- ▶ Fault tolerant
  - ▶ “Recipe“ of data transformations is preserved, so a partition can be re-created at any time
- ▶ Caching
  - ▶ Different storage levels possible
- ▶ Supports a set of Spark transformations and actions



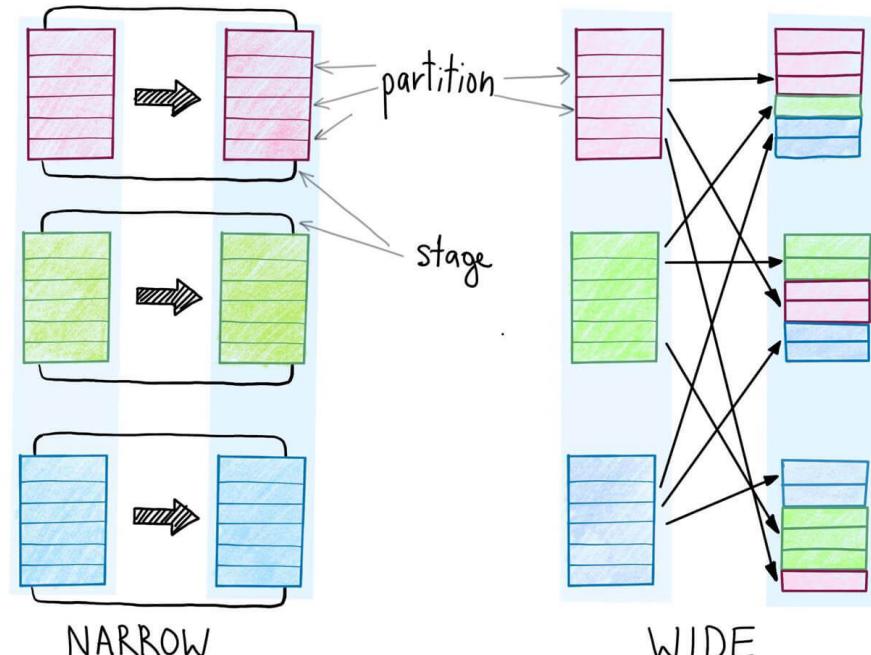
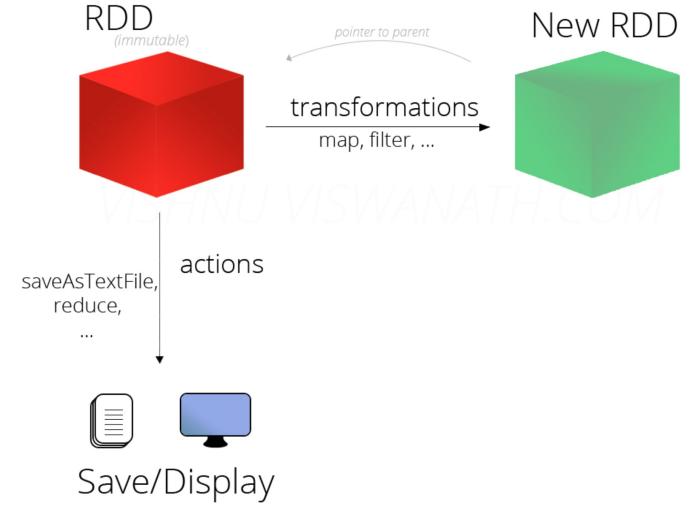
## Spark Programming concepts (Resilient distributed datasets - RDDs)

- ▶ Computations are expressed using
  - ▶ creation of new RDDs
  - ▶ transforming existing RDDs
  - ▶ operations on RDDs to compute results (actions)
- ▶ Distributes the data within RDDs across nodes (executors) in the cluster and parallelizes calculations



## RDD Operations

- ▶ RDDs enable following operations
  - ▶ **transformations**
    - ▶ lazy operations that return a new RDD from input RDDs
    - ▶ narrow or wide types
    - ▶ examples: map, filter, join, groupByKey...
  - ▶ **actions**
    - ▶ return a result or write to storage,
    - ▶ execute transformations
    - ▶ examples: count, collect, save



## RDD Transformations vs. actions



= easy



= medium

## Essential Core & Intermediate Spark Operations



### General

- map
- filter
- flatMap
- mapPartitions
- mapPartitionsWithIndex
- groupBy
- sortBy

### Math / Statistical

- sample
- randomSplit

### Set Theory / Relational

- union
- intersection
- subtract
- distinct
- cartesian
- zip

### Data Structure / I/O

- keyBy
- zipWithIndex
- zipWithUniqueId
- zipPartitions
- coalesce
- repartition
- repartitionAndSortWithinPartitions
- pipe

- 
- reduce
  - collect
  - aggregate
  - fold
  - first
  - take
  - foreach
  - top
  - treeAggregate
  - treeReduce
  - foreachPartition
  - collectAsMap

- count
- takeSample
- max
- min
- sum
- histogram
- mean
- variance
- stdev
- sampleVariance
- countApprox
- countApproxDistinct

- takeOrdered

- saveAsTextFile
- saveAsSequenceFile
- saveAsObjectFile
- saveAsHadoopDataset
- saveAsHadoopFile
- saveAsNewAPIHadoopDataset
- saveAsNewAPIHadoopFile

# Hands On Preparation

1. Use NoMachine to login to UL FS's HPC **OR** open your Terminal (preferred)

2. Open Terminal/Console/"Konzola"

3. Clone Workshop Git repository and enter its folder

```
git clone https://github.com/szitnik/Apache-Spark-Workshop.git
cd Apache-Spark-Workshop
```

4. Enter the following commands

```
module load Spark/2.4.0-Hadoop-2.7-Java-1.8
```

```
python3 -m venv spark-workshop-env
```

```
. spark-workshop-env/bin/activate
```

```
(spark-workshop-env) [campus02@viz Apache-Spark-Workshop]$
pip install --upgrade pip
pip install pyspark jupyter
python
```

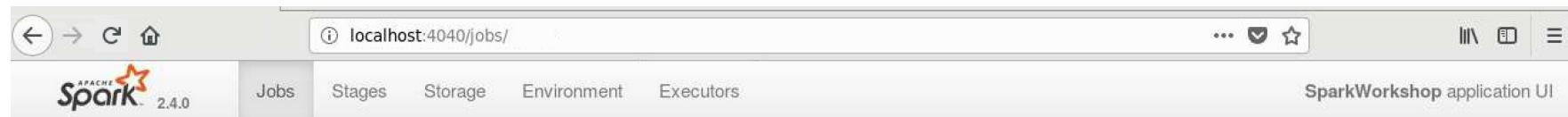
```
Python 3.6.8 (default, Apr 25 2019, 21:02:35)
[GCC 4.8.5 20150623 (Red Hat 4.8.5-36)] on linux
Type "help", "copyright", "credits" or "license" for more information.
```

## RDDs in Spark

- We will use pySpark library interactively

```
import pyspark  
  
sc = pyspark.SparkContext(appName='SparkWorkshop', master='local[1]')
```

```
2020-09-15 16:28:01 WARN NativeCodeLoader:62 - Unable to load native-hadoop library for your platform... using builtin-java classes where applicable  
Setting default log level to "WARN".  
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLevel(newLevel).  
2020-09-15 16:28:02 WARN Utils:66 - Service 'SparkUI' could not bind on port 4040. Attempting port 4041.  
>>> █
```



### Spark Jobs (?)

User: campus02  
Total Uptime: 10 s  
Scheduling Mode: FIFO

► Event Timeline

## RDDs in Spark

- ▶ Creation of RDDs

- ▶ From a collection

```
rdd1 = sc.parallelize([('John', 23), ('Mark', 11), ('Jenna', 44),  
                      ('Sandra', 61)])
```

- ▶ From a file

```
rdd2 = sc.textFile('data/IMDB Dataset.csv')
```

- ▶ Basic transformations map(), filter(), flatMap()

```
older = rdd1.filter(lambda x: x[1] > 18)  
anonymized = older.map(lambda x: (x[0][0], x[1]))
```

```
birthdays = rdd1.map(lambda x: list(range(1, x[1]+1)))  
birthdays = rdd1.flatMap(lambda x: list(range(1, x[1]+1)))
```

```
birthdays.map(lambda x: (x, 1)).groupByKey() \  
.mapValues(lambda vals: len(list(vals))).sortByKey()
```

## RDDs in Spark

- ▶ Further actions, transformations

```
rdd2.take(2)
```

```
def organize(line):  
    data = line.split('",')  
    data = data if len(data) == 2 else line.split(',')  
    return (data[1], data[0][1:51] + ' ...')  
  
movies = rdd2.filter(lambda x: x != 'review,sentiment').map(organize)  
  
movies.count() // 50.000  
movies = movies.filter(lambda x: x[0] in ['positive', 'negative'])  
movies.count() // 45.936  
movieCounts = movies.groupByKey().map(lambda x: (x[0], len(x[1])))
```

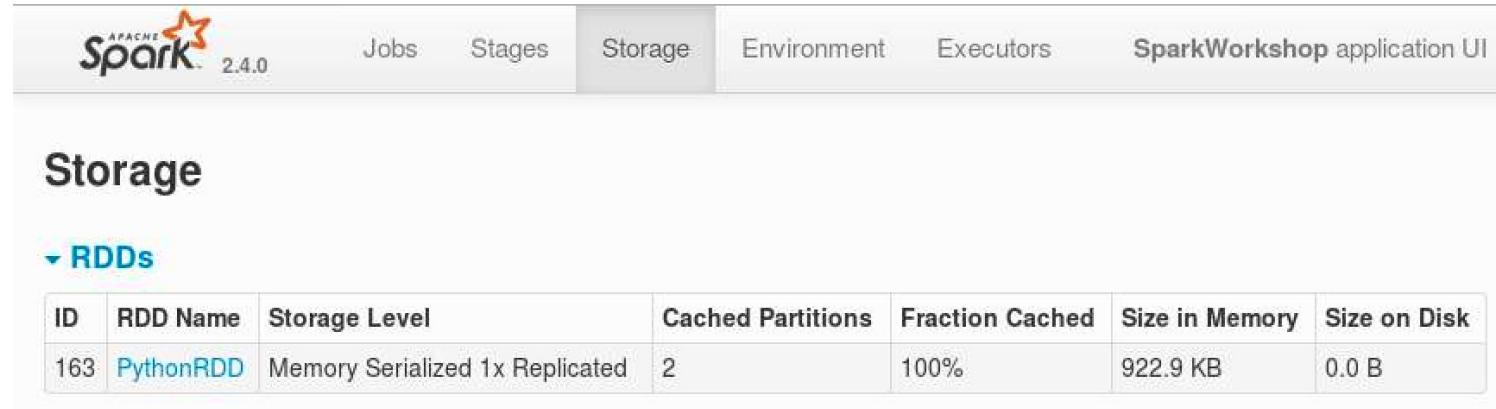
## RDDs in Spark

### ► Caching

```
movies.take(2)
```

```
posReviews = movies.filter(lambda x: x[0] == 'positive').map(lambda x: x[1])
negReviews = movies.filter(lambda x: x[0] == 'negative').map(lambda x: x[1])
```

```
posReviews.cache().collect()
```



The screenshot shows the Apache Spark 2.4.0 Storage UI. The top navigation bar includes links for Jobs, Stages, Storage (which is highlighted), Environment, Executors, and SparkWorkshop application UI. Below the navigation bar, the word "Storage" is displayed in large, bold, black font. Underneath, there is a section titled "RDDs" with a dropdown arrow icon. A table below lists the cached RDD details:

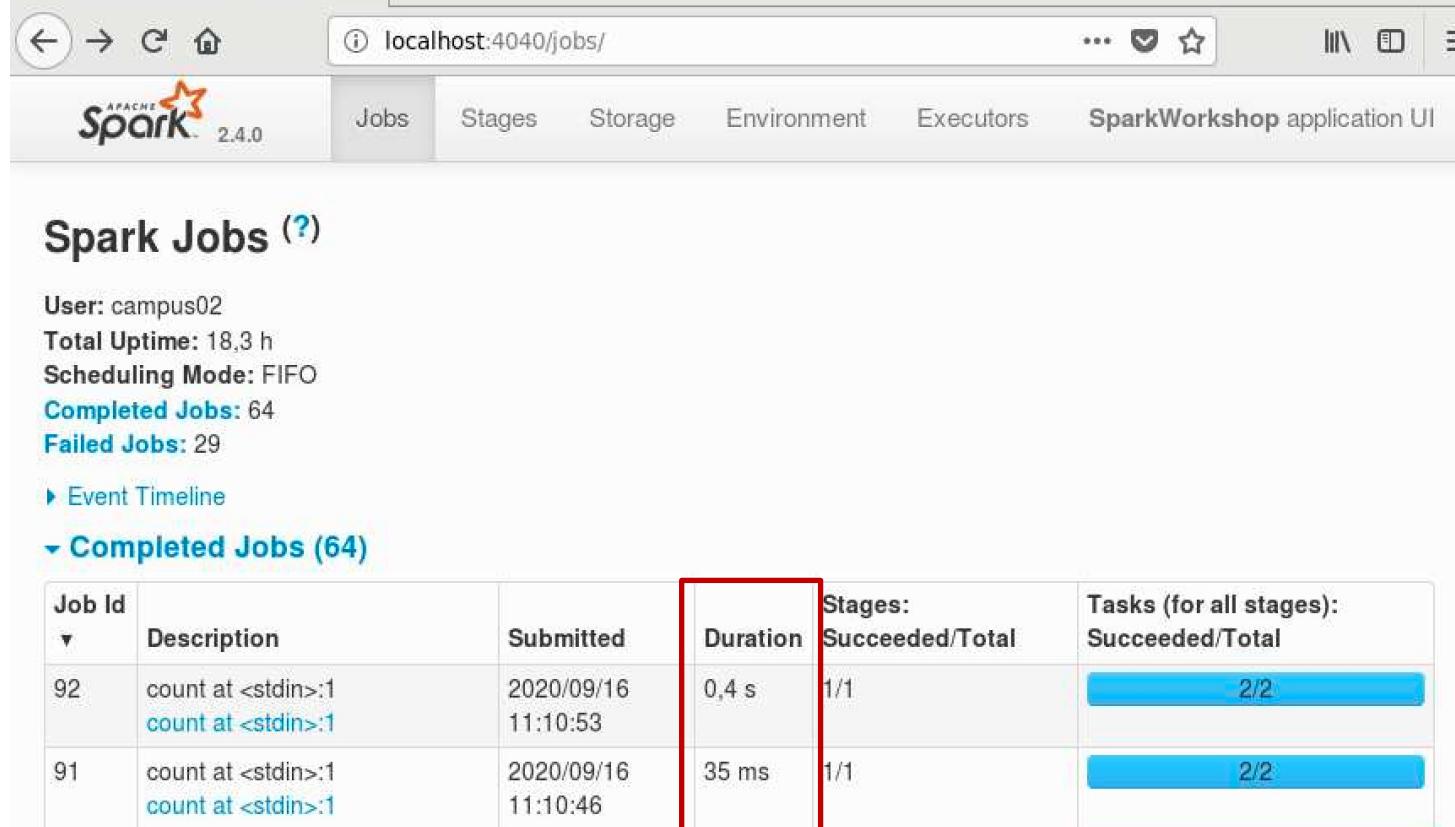
ID	RDD Name	Storage Level	Cached Partitions	Fraction Cached	Size in Memory	Size on Disk
163	PythonRDD	Memory Serialized 1x Replicated	2	100%	922.9 KB	0.0 B

## RDDs in Spark

### ► Caching

```
posReviews.filter(lambda x: 'good' in x).count() //605
```

```
negReviews.filter(lambda x: 'bad' in x).count() //788
```



The screenshot shows the Apache Spark 2.4.0 UI at [localhost:4040/jobs/](http://localhost:4040/jobs/). The UI has tabs for Jobs, Stages, Storage, Environment, Executors, and SparkWorkshop application UI. The Jobs tab is selected.

### Spark Jobs (?)

User: campus02  
 Total Uptime: 18,3 h  
 Scheduling Mode: FIFO  
 Completed Jobs: 64  
 Failed Jobs: 29

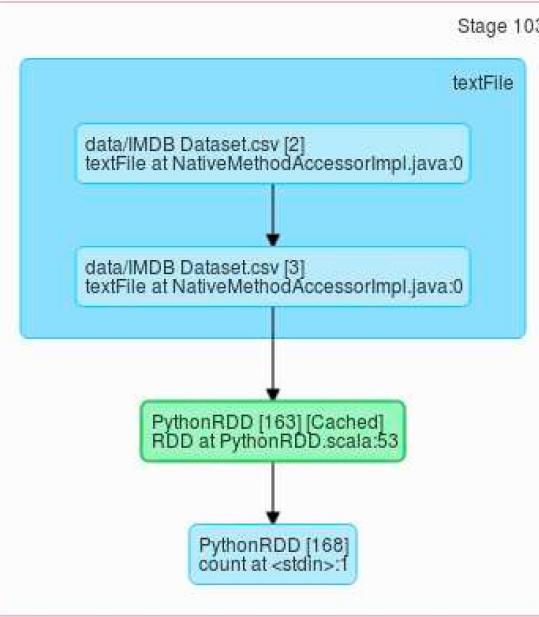
► Event Timeline

▼ **Completed Jobs (64)**

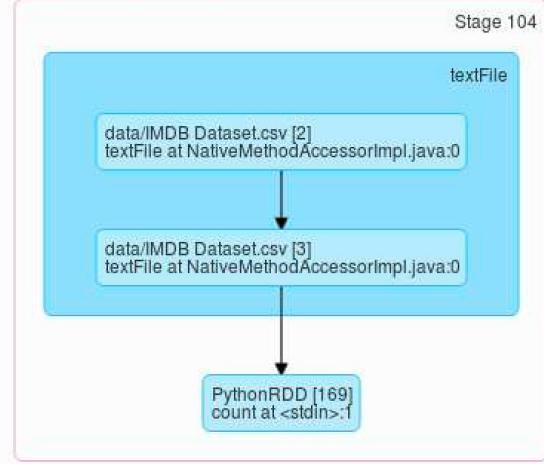
Job Id	Description	Submitted	Duration	Stages: Succeeded/Total	Tasks (for all stages): Succeeded/Total
92	count at <stdin>:1 count at <stdin>:1	2020/09/16 11:10:53	0,4 s	1/1	2/2
91	count at <stdin>:1 count at <stdin>:1	2020/09/16 11:10:46	35 ms	1/1	2/2

# RDDs in Spark

## ▼ DAG Visualization



## ▼ DAG Visualization



## ▼ Tasks (2)

Index	ID	Attempt	Status	Locality Level	Executor ID	Host	Launch Time	Duration	GC Time	Input Size / Records	Errors
0	138	0	SUCCESS	PROCESS_LOCAL	driver	localhost	2020/09/16 11:10:53	0,2 s	4 ms	32.1 MB / 25311	
1	139	0	SUCCESS	PROCESS_LOCAL	driver	localhost	2020/09/16 11:10:53	0,2 s	4 ms	31.1 MB / 24690	

## ▼ Tasks (2)

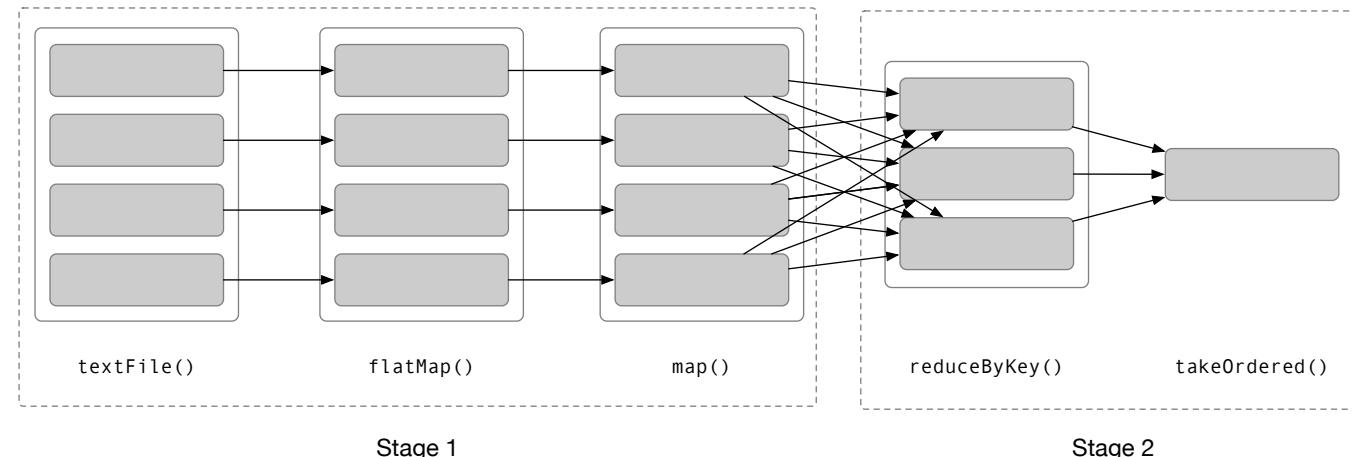
Index	ID	Attempt	Status	Locality Level	Executor ID	Host	Launch Time	Duration	GC Time	Input Size / Records	Errors
0	136	0	SUCCESS	PROCESS_LOCAL	driver	localhost	2020/09/16 11:10:46	18 ms		469.2 KB / 16	
1	137	0	SUCCESS	PROCESS_LOCAL	driver	localhost	2020/09/16 11:10:46	12 ms		453.7 KB / 16	

## RDDs in Spark

- DAG exploration

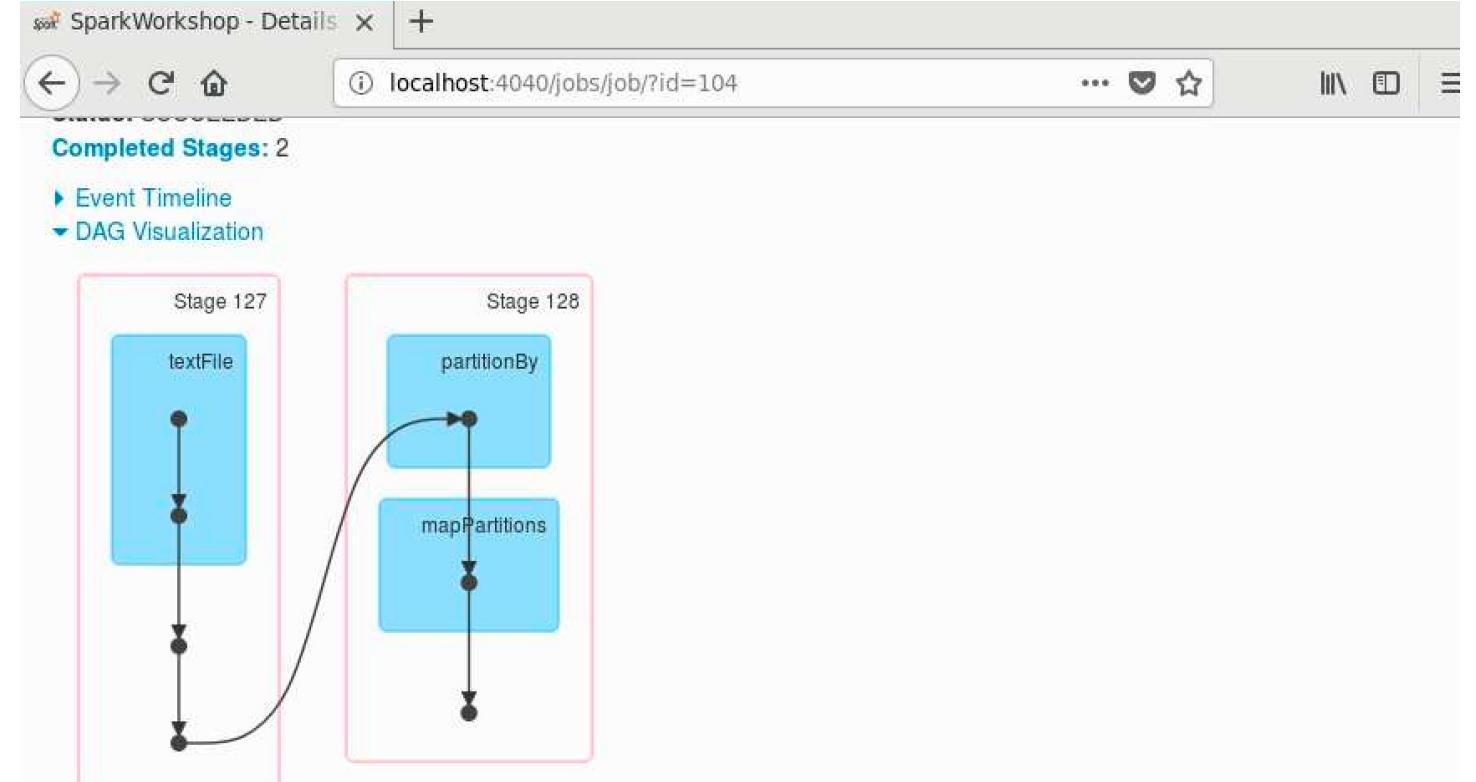
```
def splitLine(line):
    return line.replace(',', ' ').replace('"', ' ').replace('.', ' ').split()

rdd2 = sc.textFile('data/IMDB Dataset.csv', 4)
wordCounts = rdd2.flatMap(splitLine).map(lambda word: (word, 1)). \
    reduceByKey(lambda a,b: a+b, 3)
wordCounts.takeOrdered(10, key = lambda x: -x[1])
```



## RDDs in Spark

- ▶ DAG exploration  
(admin console  
result)



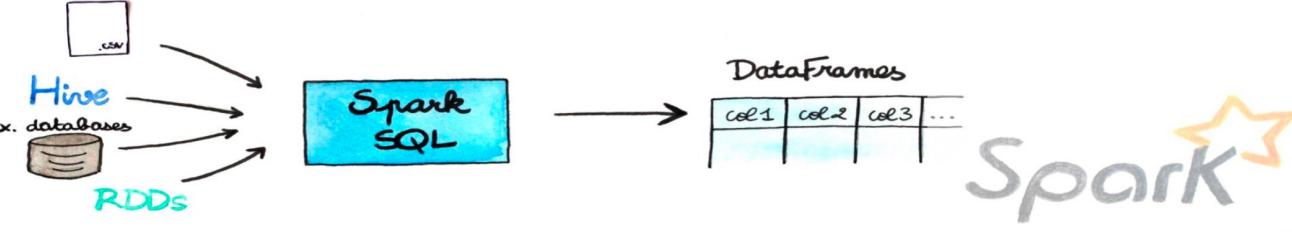
## DataFrames (= RDDs + schema) in Spark

- ▶ Spark SQL enables read/write from/to files, JSON, databases, etc.
- ▶ DataFrames interoperable with Pandas dataframe
- ▶ DataFrames creation ...

```
from pyspark.sql import SQLContext, Row
from pyspark.sql.types import StructType, IntegerType, StringType, StructField
sqlContext = SQLContext(sc)

df1 = sqlContext.createDataFrame(rdd1, ["name", "age"])
```

```
ExampleRow = Row("name", "age")
rdd1a = rdd1.map(lambda x: ExampleRow(x[0], x[1]))
df1 = sqlContext.createDataFrame(rdd1a)
```



Name	Age	Height
Person		

## DataFrames (= RDDs + schema) in Spark

- ... DataFrames creation ...

```
schema = StructType([StructField("name", StringType(), False), \
                     StructField("age", IntegerType(), True)])
```

```
df1 = sqlContext.createDataFrame(rdd1, schema)
```

```
df1.show() +----+---+
| name|age|
+----+---+
| John| 23|
| Mark| 11|
| Jenna| 44|
| Sandra| 61|
+----+---+
```

```
df1.printSchema()
```

```
root
 |-- name: string (nullable = false)
 |-- age: integer (nullable = true)
```

## DataFrames (= RDDs + schema) in Spark

- ▶ ... DataFrames creation

```
df2 = sqlContext.read.format('csv').option('header', 'true'). \
        option('mode', 'DROPMALFORMED').load('data/IMDB Dataset.csv')

df2.show(5)+-----+-----+
|           review|sentiment|
+-----+-----+
|One of the other ...| positive|
|Basically there's...| negative|
|I sure would like...| positive|
|This show was an ...| negative|
|Encouraged by the...| negative|
+-----+-----+
only showing top 5 rows

df2.printSchema()

root
 |-- review: string (nullable = true)
 |-- sentiment: string (nullable = true)
```

# DataFrames and User Defined Functions (UDF) in Spark

- ▶ User defined functions are custom functions to run against the "database" directly
- ▶ Caveats
  - ▶ Optimization problems (especially in pySpark!)
  - ▶ Special values handling by the programmer (e.g. null values)
- ▶ Approaches to use UDFs
  - ▶ `df = df.withColumn`
  - ▶ `df = sqlContext.sql("SELECT * FROM <UDF>")`
  - ▶ `rdd.map(UDF())`

# DataFrames and User Defined Functions (UDF) in Spark

## ► Examples

```
from pyspark.sql.functions import udf

reviewLen = udf(lambda r: len(r), IntegerType())
reviewSnippet = udf(lambda r: r[0:50] + '...', StringType())

df2 = df2.withColumn('reviewLength', reviewLen('review'))
df2 = df2.withColumn('reviewSnippet', reviewSnippet('review'))
```

# DataFrames and User Defined Functions (UDF) in Spark

## ► Examples

```
def words(review, type = 'positive'):  
    sentimentWords = ['good', 'great', 'nice', 'awesome']  
    if type == 'negative':  
        sentimentWords = ['bad', 'worst', 'ugly', 'scary']  
    return sum(map(lambda w: review.count(w), sentimentWords))  
  
positiveWords = udf(lambda r: words(r), IntegerType())  
negativeWords = udf(lambda r: words(r, 'negative'), IntegerType())  
  
df2 = df2.withColumn('positiveWords', positiveWords('review'))  
df2 = df2.withColumn('negativeWords', negativeWords('review'))  
  
df2 = df2.drop('review')
```

# DataFrames and User Defined Functions (UDF) in Spark

## ► Examples

```
df2.cache().show()
+-----+-----+-----+-----+
|sentiment|reviewSnippet|reviewLength|positiveWords|negativeWords|
+-----+-----+-----+-----+
| positive|One of the other ...|1761|0|0|
| negative|Basically there's...|748|0|0|
| positive|I sure would like...|726|1|1|
| negative|This show was an ...|934|0|0|
| negative|Encouraged by the...|681|0|0|
| positive|If you like origi...|176|0|0|
| negative|"Phil the Alien i...|582|0|0|
| negative|I saw this movie ...|937|0|0|
| negative|The cast played S...|662|1|1|
| positive|This a fantastic ...|275|1|1|
| negative|Kind of drawn in ...|830|0|0|
| positive|Some films just s...|720|0|0|
| negative|This movie made i...|1322|0|0|
| positive|I remember this f...|639|0|0|
| negative|An awful film! It...|741|1|1|
| positive|After the success...|1813|1|1|
| positive|What an absolutel...|346|0|0|
| negative|This was the wors...|808|2|2|
| positive|The Karen Carpent...|680|3|3|
| negative|This film tried t...|836|0|0|
+-----+-----+-----+-----+
only showing top 20 rows
```

## Spark SQL DataFrame operations

- ▶ Examples

```
df2.select('sentiment', 'positiveWords').show(3)
```

```
+-----+-----+
|sentiment|positiveWords|
+-----+-----+
| positive|          0|
| negative|          0|
| positive|          1|
+-----+-----+
only showing top 3 rows
```

```
df2.select(df2['sentiment'], df2['positiveWords']).show(3)
```

```
+-----+-----+
|sentiment|positiveWords|
+-----+-----+
| positive|          0|
| negative|          0|
| positive|          1|
+-----+-----+
only showing top 3 rows
```

```
df2.select(df2['sentiment'], df2['positiveWords']). \
filter(df2['positiveWords'] > 10).show(3)
```

```
+-----+-----+
|sentiment|positiveWords|
+-----+-----+
| positive|         18|
| negative|         11|
| positive|         11|
+-----+-----+
only showing top 3 rows
```

```
df2.groupBy('sentiment').count().show()
```

```
+-----+-----+
|sentiment|count|
+-----+-----+
| positive|14897|
| negative|13792|
+-----+-----+
```

```
df2.summary().show()
```

```
+-----+-----+-----+-----+
|summary|sentiment| reviewSnippet| positiveWords| negativeWords|
+-----+-----+-----+-----+
| count| 28691|        28691|       28691|      28691|
| mean|   null|           null| 1.002474643616465| 1.002474643616465|
| stddev|  null|           null| 1.351616569521083| 1.351616569521083|
| min| positive|!!! Spoiler alert...|          0|          0|
| 25%|   null|           null|          0|          0|
| 50%|   null|           null|          1|          1|
| 75%|   null|           null|          1|          1|
| max| positive|ý thýnk uzak ýs t...|         24|         24|
+-----+-----+-----+-----+
```

## Spark SQL SQL operations

### ► Examples

```
df2.createOrReplaceTempView('imdb')
```

```
sqlContext.sql('SELECT * FROM imdb WHERE positiveWords > 10 LIMIT 5').show()
```

sentiment	reviewSnippet	positiveWords	negativeWords
positive	This is a great G...	18	18
negative	I just watched th...	11	11
positive	I bought this a w...	11	11
positive	I can't say too m...	11	11
positive	"First an explana...	14	14

```
sqlContext.sql('SELECT sentiment, count(*) FROM imdb GROUP BY sentiment').show()
```

sentiment	count(1)
positive	14897
,positive	2
negative	13792

# Lab exercise – Spark on an HPC

► Move to spark-hpc folder:

- ▶ 00\_clean.sh
  - ▶ Script to clean logs, data generated by running scripts
- ▶ 01\_run-sbatch.sh
  - ▶ Prepare and submit scripts with a Spark job
- ▶ conf/spark-env.sh
  - ▶ Env variables for worker folder and log folder set
- ▶ job.py
  - ▶ Pyspark source code (i.e. simple pi calculation script)
- ▶ logs/
  - ▶ Spark and slurm log folder
- ▶ NOTES.txt
  - ▶ Short Slurm commands reference
- ▶ spark-job-TEMPLATE.sh
  - ▶ Slurm script for job submission
- ▶ workers/
  - ▶ Workers working directories

00\_clean.sh  
01\_run-sbatch.sh  
conf  
job.py  
logs  
NOTES.txt  
spark-job-TEMPLATE.sh  
workers

# Lab exercise – Spark on an HPC

## ► job.py

```
from pyspark import SparkContext, SparkConf
from random import random
from operator import add

spark_conf = SparkConf()
sc = SparkContext(appName='SparkWorkshop Python Pi', conf = spark_conf)

n = 100000000

def f(_):
    x = random() * 2 - 1
    y = random() * 2 - 1
    return 1 if x ** 2 + y ** 2 <= 1 else 0

count = sc.parallelize(range(1, n + 1), 2).map(f).reduce(add)

print("\n\n#####")
print("Pi is roughly %f" % (4.0 * count / n))
print("#####\n\n")
```

# Lab exercise – Spark on an HPC

► spark-job-TEMPLATE.sh

```
#!/bin/bash
#SBATCH -N 1
#SBATCH -t 00:10:00
#SBATCH --ntasks-per-node 3
#SBATCH --cpus-per-task 2
#SBATCH --output=logs/slurm_stdout_err_%j.log

module purge
module load Spark/2.4.0-Hadoop-2.7-Java-1.8

# MAIN DIR
export JOB_SPARK_DIR="CURRENT_DIR"

export JOB_SPARK_LOG_DIR="${JOB_SPARK_DIR}/logs"
export JOB_SPARK_WORKER_DIR="${JOB_SPARK_DIR}/workers"
export SPARK_CONF_DIR="${JOB_SPARK_DIR}/conf"

# Start Master
start-master.sh

# Start Workers
start-slave.sh spark://`hostname`:7077

# Spark cluster running to submit jobs
# via driver from login node
#sleep infinity

# Run driver on an HPC node
spark-submit --master spark://`hostname`:7077 "${JOB_SPARK_DIR}/job.py"
```

# Lab exercise – Spark on an HPC

## ► 01\_run-sbatch.sh

```
#!/bin/bash

cp spark-job-TEMPLATE.sh spark-job.sh
sed -i "s#CURRENT_DIR#${PWD}#g" spark-job.sh
sbatch spark-job.sh

rm -rf spark-job.sh
```

# Lab exercise – Spark on an HPC

- ▶ Run spark application on an HPC (commands):

```
./01_run-sbatch.sh
```

```
(spark-workshop-env) [campus02@viz spark-hpc]$ ./01_run-sbatch.sh
Submitted batch job 51438
```

```
squeue -u campus02
```

JOBID	PARTITION	NAME	USER	ST	TIME	NODES	NODELIST(REASON)
51438	westmere	spark-jo	campus02	R	0:17	1	cn17

```
sacct -j 51438
```

JobID	JobName	Partition	Account	AllocCPUS	State	ExitCode
51438	spark-job+	westmere		12	COMPLETED	0:0
51438.batch	batch			12	COMPLETED	0:0
51438.extern	extern			12	COMPLETED	0:0



PARTNERSHIP FOR ADVANCED COMPUTING IN EUROPE

# Lab exercise – Spark on an HPC

- ▶ Check the output log:

```
cat logs/slurm_stdout_err_51438.log
```

```
....  
2020-09-16 21:42:51 INFO DAGScheduler:54 - Job 0 finished: reduce at ...
```

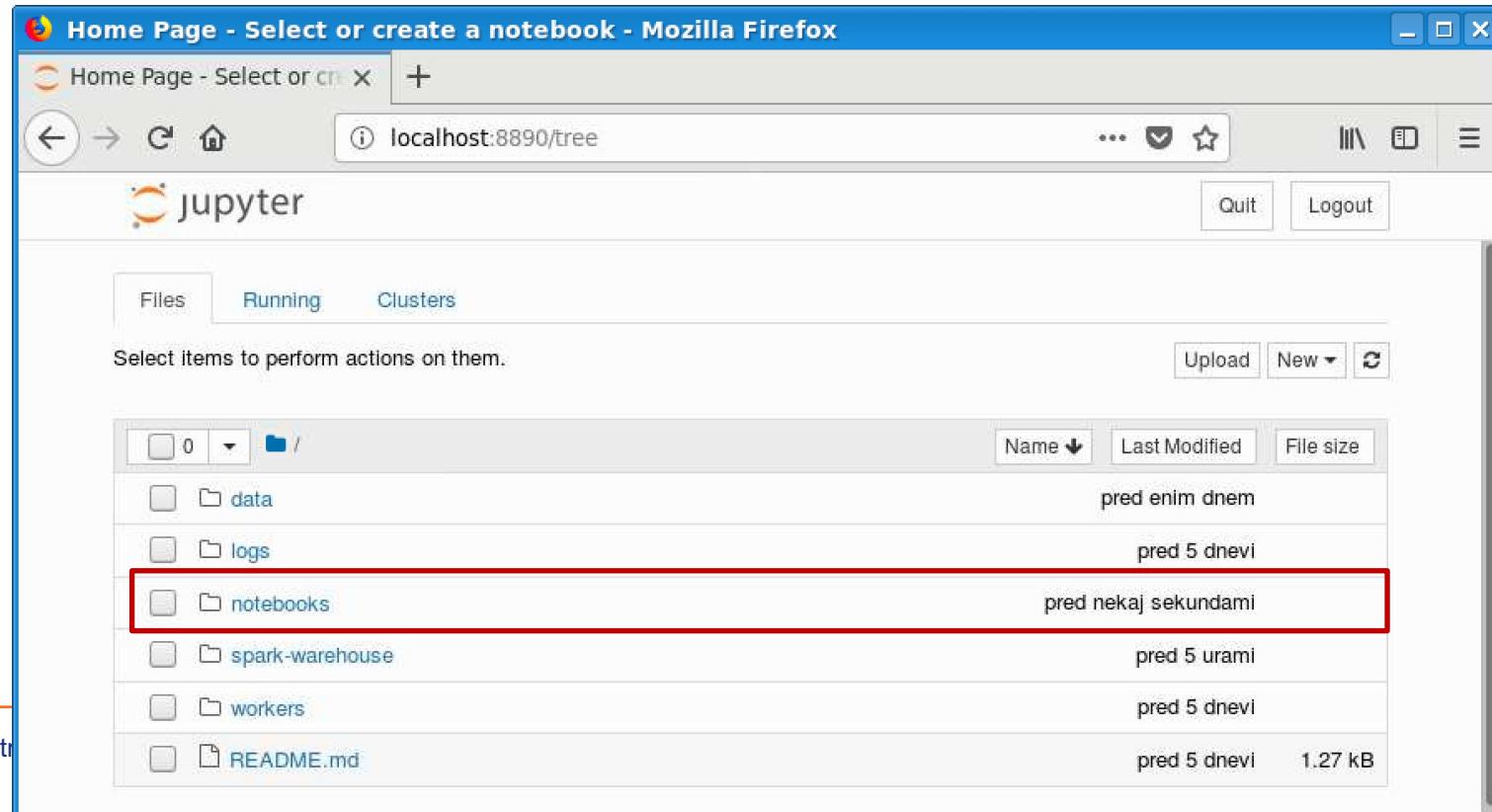
```
#####
#Pi is roughly 3.141645
#####
```

```
2020-09-16 21:42:51 INFO SparkContext:54 - Invoking stop() from shutdown hook  
...
```

# Challenge exercises - Jupyter

- ▶ Run jupyter notebook command in the project folder and run notebooks from notebooks folder

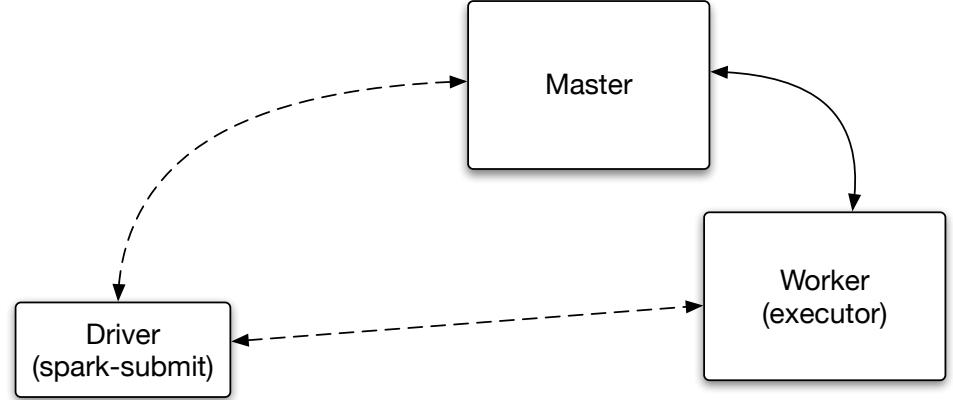
```
(spark-workshop-env) [campus02@viz Apache-Spark-Workshop]$ jupyter notebook
```



# Challenge exercises

- ▶ Run **Lab 1** notebook
  - ▶ (Option A) Run it using embedded Spark within PySpark
  - ▶ (Option B) Setup a Spark cluster and run Lab 1 script using it (**check further slides**)
- ▶ Solve **Spark Pi** notebook
- ▶ Solve **Lab 2** notebook challenge
  
- ▶ Train a classifier to predict movie review sentiment
  - ▶ Use the provided *IMBD reviews.csv* data and split it to train and test set
  - ▶ Extract features (e.g. TF-IDF), train model (e.g. SVM) and test it
  - ▶ See MLlib documentation at <https://spark.apache.org/docs/latest/ml-guide.html>
  
- ▶ Try using **Spark on an HPC** (**check further slides**) (**why not offered by default in HPCs?**)

# Lab 1 – Spark cluster submit



- ▶ Download and unpack Spark binaries from
  - ▶ <https://downloads.apache.org/spark/> (select \*bin-hadoop\* tgz archive)
- ▶ Start master node

```
SPARK_MASTER_HOST=127.0.0.1 ./sbin/start-master.sh
```

  - ▶ Check log to find out Spark master endpoint and Web UI URL
- ▶ Start worker nodes

```
SPARK_WORKER_INSTANCES=1 SPARK_WORKER_CORES=1 SPARK_WORKER_MEMORY=2g
./sbin/start-slave.sh spark://localhost:7077
```
- ▶ Update Lab 1 notebook (*SparkConf*) and save it as Python script

# Lab 1 – Spark cluster submit

- ▶ Submit script as a Spark job

```
SPARK_MASTER_HOST=127.0.0.1 ./bin/spark-submit --master  
spark://localhost:7077 Lab-1.py
```

- ▶ Observe the driver output

- ▶ Stopping Spark

- ▶ Stop driver
- ▶ Stop slaves

```
SPARK_WORKER_INSTANCES=1 SPARK_WORKER_CORES=1 SPARK_WORKER_MEMORY=2g  
.sbin/stop-slave.sh spark://localhost:7077
```

- ▶ Stop master

```
.sbin/stop-master.sh
```

# Lab exercise – Spark on an HPC

- ▶ Use more nodes with workers on an HPC
  - ▶ Adapt HPC lab exercise to be run on multiple nodes
  - ▶ Hint:  
[https://info.gwdg.de/wiki/doku.php?id=wiki:hpc:slurm\\_sbatchescript\\_for\\_spark\\_applications](https://info.gwdg.de/wiki/doku.php?id=wiki:hpc:slurm_sbatchescript_for_spark_applications)
- ▶ Viewing event logs in the SparkUI after Slurm job is finished
  - ▶ Replicate the HPC exercise from before, retrieve logs and run history server to explore SparkUI
  - ▶ Hints: <https://researchcomputing.princeton.edu/faq/spark-via-slurm>

## References

- ▶ <https://training.databricks.com/visualapi.pdf>
- ▶ <https://events.prace-ri.eu/event/896/>
- ▶ <https://luminousmen.com/post/spark-core-concepts-explained>
- ▶ [https://info.gwdg.de/wiki/doku.php?id=wiki:hpc:slurm\\_sbatch\\_script\\_for\\_spark\\_applications](https://info.gwdg.de/wiki/doku.php?id=wiki:hpc:slurm_sbatch_script_for_spark_applications)
- ▶ <https://researchcomputing.princeton.edu/faq/spark-via-slurm>



**THANK YOU FOR YOUR ATTENTION**

[www.prace-ri.eu](http://www.prace-ri.eu)