“It’s difficult to imagine the power that you’re going to have when so many different sorts of data are available”

Tim Berners-Lee, WWW Inventor, 2007

Lecture C2
Dataflow Processing

CS205: Computing Foundations for Computational Science
Dr. Ignacio M. Llorente
Spring Term 2021

INSTITUTE FOR APPLIED COMPUTATIONAL SCIENCE
AT HARVARD UNIVERSITY

HARVARD
School of Engineering and Applied Sciences

Before We Start

Where We Are

Computing Foundations for Computational and Data Science

How to use modern computing platforms in solving scientific problems

Intro: Large-Scale Computational and Data Science

A. Parallel Processing Fundamentals

B. Parallel Computing

C. Parallel Data Processing

C1. Batch Data Processing

C2. Dataflow Processing

C3. Stream Data Processing

Wrap-Up: Advanced Topics

CS205: Contents

APPLICATION SOFTWARE

Application Parallelism

Program Design

Application Software

BIG COMPUTE

OpenACC

Optimization

OpenMP

MPI

Programming Model

Spark

Map-Reduce

BIG DATA

Slurm

Platform

Hadoop

Architecture

amazon web services

ODYSSEY
HARVARD FAS RESEARCH COMPUTING

FAS RC

Cloud Computing

Computing Cluster

Before We Start

Where We Are

Concepts → Programming → Platform

Batch Data Processing => MapReduce

3/18	3/23	Lab
Lecture C1	Hands-on H4	Lab I8
Batch Data Processing	MapReduce Programming	MapReduce Hadoop Cluster
(Quiz & Reading)		

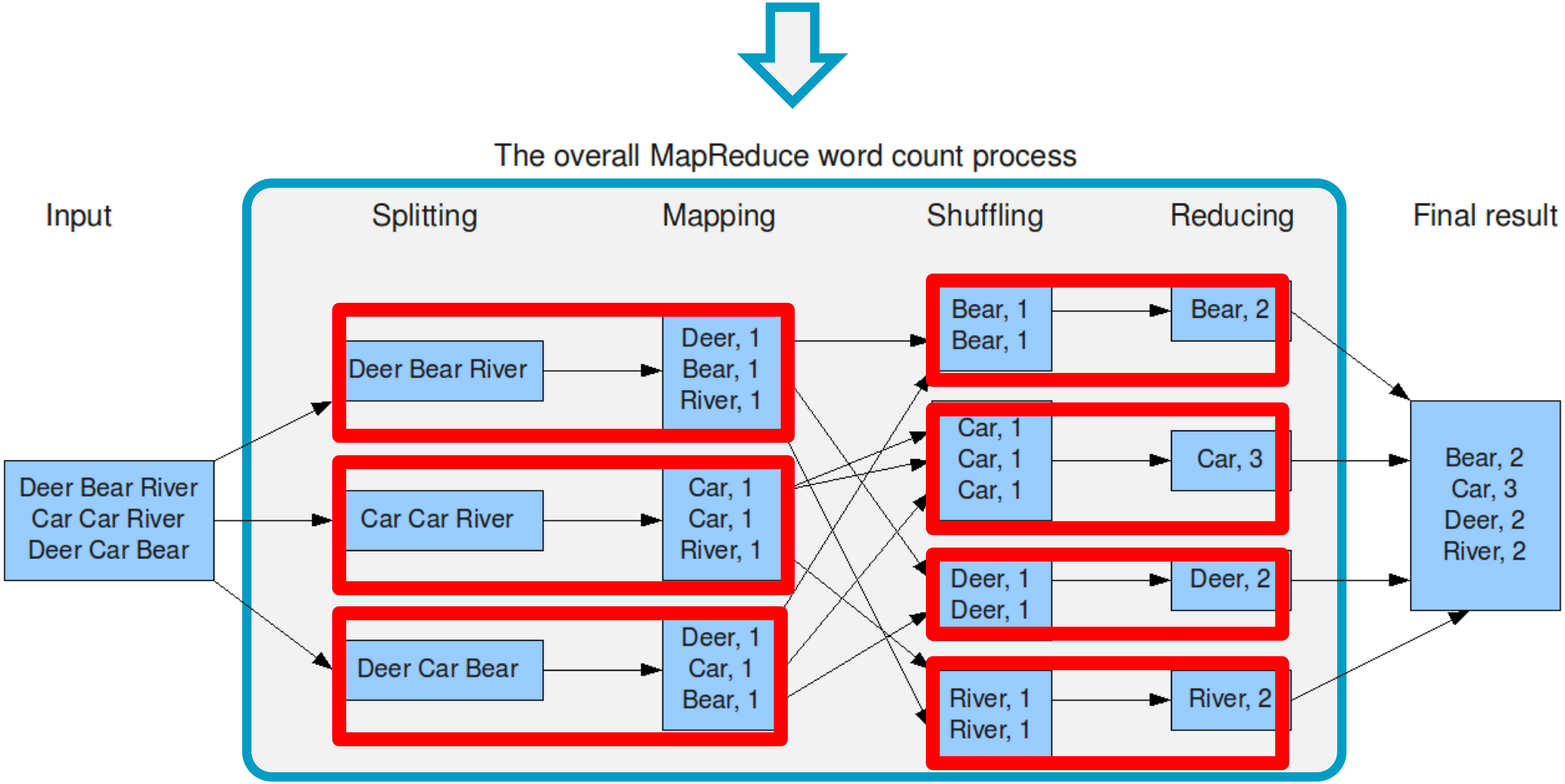
Dataflow Processing => Spark

3/25	3/30	Lab	Lab
Lecture C2	Hands-on H5	Lab I9	Lab I10
Dataflow Processing	Spark Programming	Spark Single Node	Spark Cluster
(Quiz & Reading)			

Context

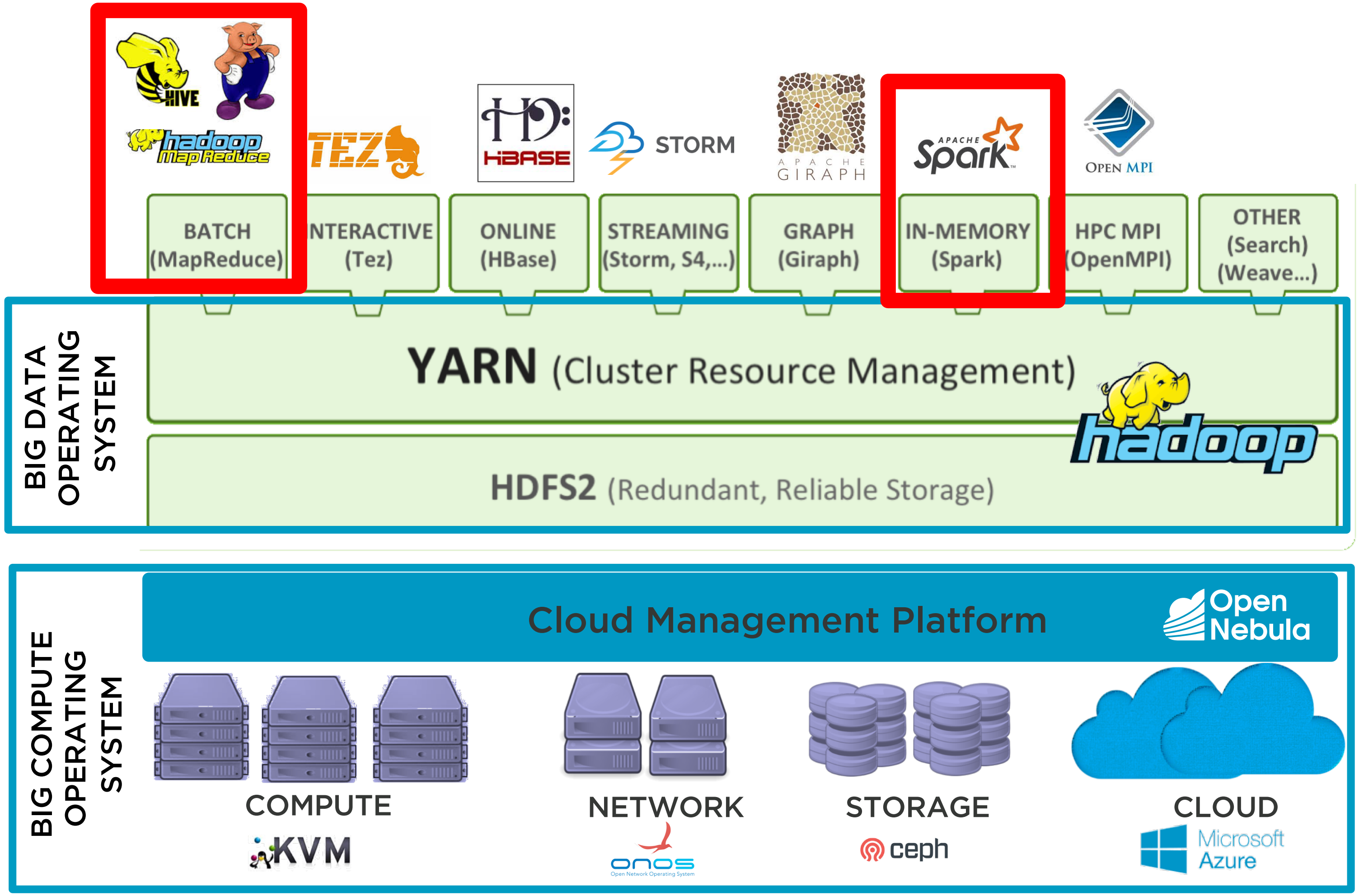
The MapReduce Programming Model

JOB DESCRIPTION
(map / reduce)



Context

The Hadoop Platform



Roadmap

Dataflow Processing

MapReduce Limitations

The Spark Execution Engine

The Spark Programming Model

The Spark Ecosystem

Hadoop Limitations

First Impressions about MapReduce?

What are your first impressions about MapReduce?

Sources: https://www.packtpub.com/books/content/getting-started-apache-hadoop-and-apache-spark

MapReduce Limitations

MapReduce Is for One-Pass Computations of Large Data Sets

Suitability of the Programming Model

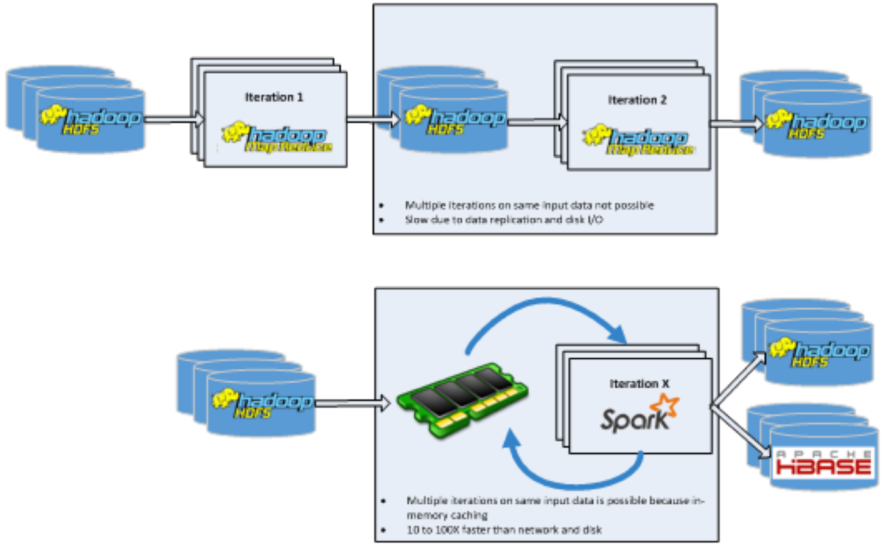
- Any **use case should be converted into MapReduce pattern** where each step in the data processing workflow requires one Map phase and one Reduce phase
- Work **from/to disk**, which is too slow for small data, interactive queries, iterative jobs, streaming…

Complex Deployment

- MapReduce **always requires clusters** that are hard to set up and manage, and the integration of several tools for different big data use cases
- Separate modules require separate **administration**

Inefficient Multi-pass Computations

- The output **data between each step has to be stored in the DFS** before the next step can begin, which is slow due to replication & disk storage



Sources: https://www.packtpub.com/books/content/getting-started-apache-hadoop-and-apache-spark

The Spark Execution Engine

In-Memory Cluster Computing

Overcoming MapReduce Limitations

- In-memory data sharing across DAGs, so that different jobs can work with the same data
- Develop complex, multi-step data pipelines using DAG patterns
- Simple deployment and management

Complements Hadoop

Not a modified version of Hadoop
Can work standalone or on Hadoop common
Supports Scala, Python and Java

MapReduce is special-purpose Big Dataflow Processing

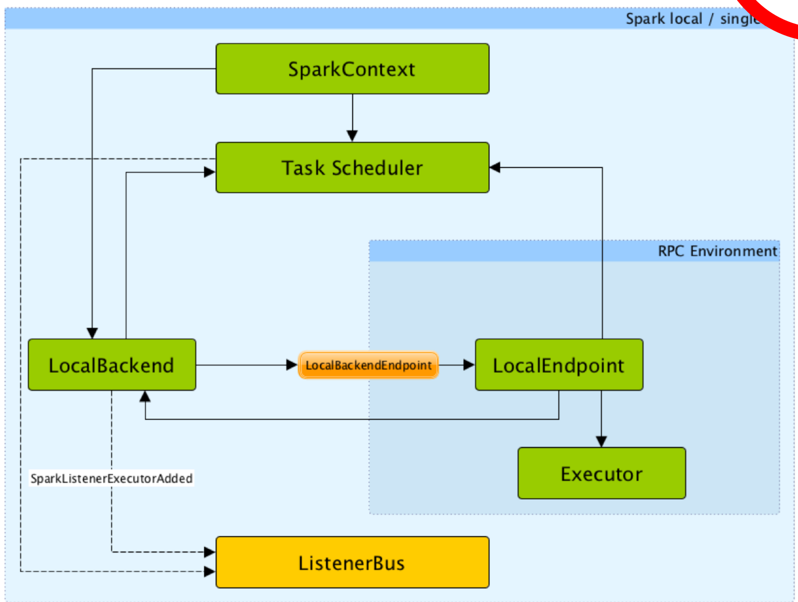
Spark is general-purpose Big Dataflow Processing

The Spark Execution Engine

Deployment Models

Local

I9

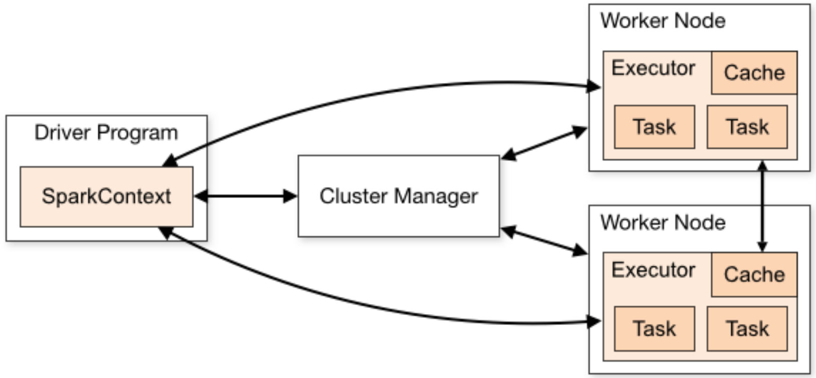


Sources: https://jaceklaskowski.gitbooks.io/mastering-apache-spark/spark-local.html

- Non-distributed **single-node deployment mode**, Spark spawns all the execution components in the same single node.
- <u>Multi-core!</u>

Cluster

I10



Sources: https://spark.apache.org/docs/latest/cluster-overview.html

- **Standalone** – Simple cluster manager included with Spark that makes it easy to set up a cluster
- **Apache Mesos** – General cluster manager that can also run Hadoop MapReduce and service applications.
- **Hadoop YARN** – Resource manager in Hadoop 2.
- <u>Multi-core and multi-node!</u>

The Spark Execution Engine

Functional Programming

Functional Parallel Programming

- Application decomposed into a set of connections (functions/**transformations**) as Directed Acyclic Graphs (DAG), with emphasis on "**data flow**" between the nodes

Computation of Distributed Collections of Data

- **Resilient Distributed Datasets** (RDD). The environment only lets you make a collection of immutable data sets that are **distributed** across a cluster such that they can be automatically **re-built upon node failure**.
- **Coarse-grained transformations**. A program is a set of **parallel transformations** (e.g, `map`, `filter`, `join`, $\cdots$,) that compute on RDDs.

Declarative Definition of Computations

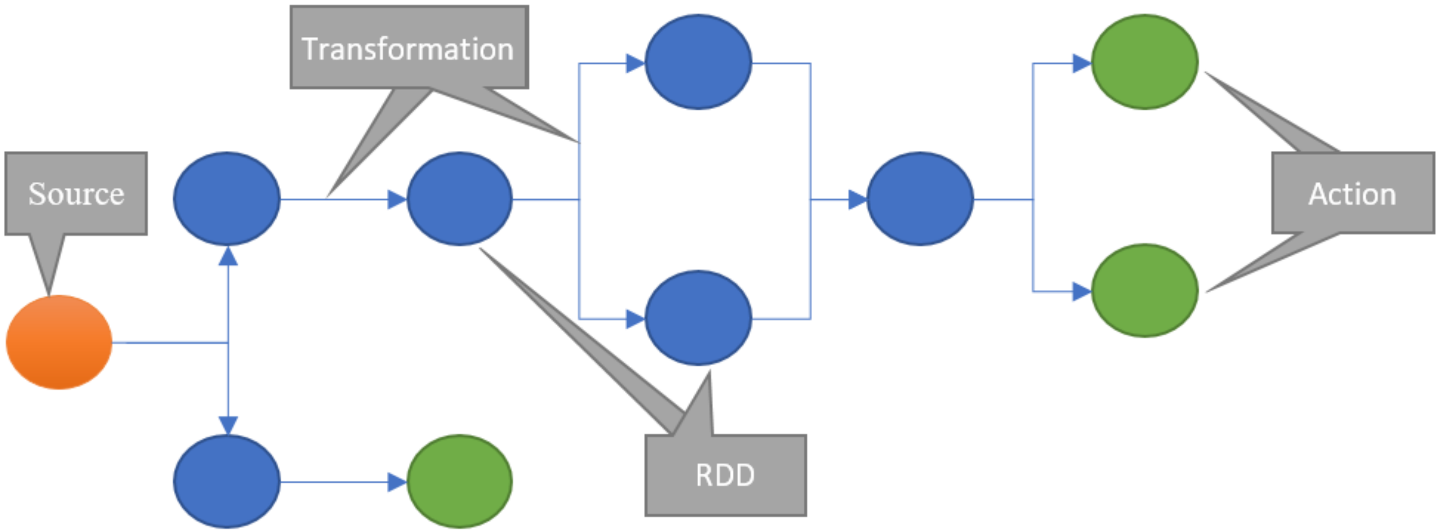
- **Functions**: Mathematically, like mapping from set A (domain) to set B (co-domain), and computationally, transformation of input into output
- **Composition (*pipelining*) of functions**: Written `f.g(x)` and interpreted as `g(f(x))` i.e, apply the function `f` to `x`, and then apply `g` to the the result of `f(x)`

The Spark Execution Engine

DAG Parallelism

A Parallel Program is Modelled as a DAG

- Vertices (nodes) representing **RDDs**
- Edges (arrows) representing functions that are either
  - **Transformations** : RDD => RDD (eg. map, filter, groupBy, join)
  - **Actions** : RDD => result (eg., count, reduce)



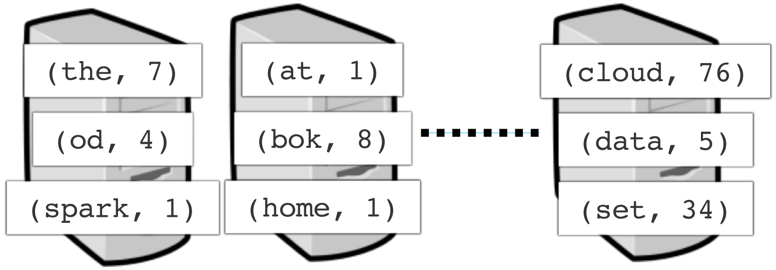
Sources: medium.com/towards-data-science/apache-spark-101-3f961c89b8c5

The Spark Programming Model

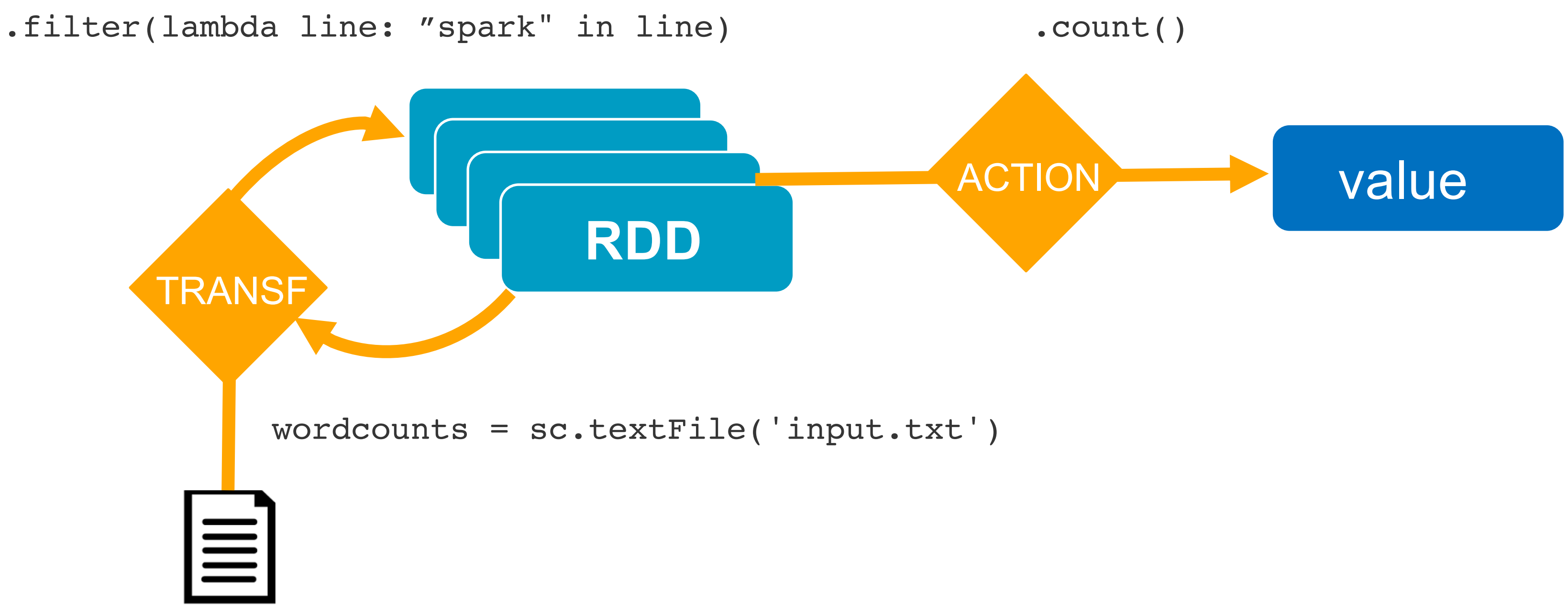
The Basics

The Fundamental Data Structure - Resilient Distributed Dataset

- **Resilient:** Fault-tolerant
- **Distributed:** Multiple-node
- **Dataset:** Collection of partitioned data organized in records



Operations: Transformations and Actions



The Spark Programming Model

The WordCount Example with Spark

A Pipeline of Transformations

```
wordcounts = sc.textFile('input.txt')
```

'The Project Gutenberg EBook of Moby Dick; or The Whale, by Herman'
'Melville. This eBook is for the use of anyone anywhere at no cost and'

```
.map(lambda x: x.replace(',',' ').replace('.',' '). lower())
```

'the project gutenberg eBook of moby dick or the whale by herman'
'melville this eBook is for the use of anyone anywhere at no cost and'

```
.flatMap(lambda x: x.split())
```

'the' 'project' 'gutenberg' 'eBook' 'of' 'moby' 'dick' 'or' 'the 'whale'
'by' 'herman' 'melville' 'this' 'eBook' 'is' 'for' 'the' 'use' 'of'

```
.map(lambda x: (x, 1))
```

'(the, 1)' '(project ,1)' '(gutenberg, 1)' '(eBook, 1)' '(of, 1)' '(moby
, 1)' '(dick, 1)' '(or, 1)' '(the, 1)' '(whale, 1)' '(by, 1)'

```
.reduceByKey(lambda x,y:x+y)
```

'(the, 11)' '(project ,10)' '(gutenberg, 9)' '(eBook, 37)' '(of, 15)'
'(moby , 5)' '(dick, 7)' '(or, 9)' '(the, 9)' '(whale, 123)' '(by, 98)'

The Spark Programming Model

The WordCount Example with Spark

```
from pyspark import SparkConf, SparkContext
import string

conf = SparkConf().setMaster('local').setAppName('WordCount')
sc = SparkContext(conf = conf)

RDDvar = sc.textFile("input.txt")

words = RDDvar.flatMap(lambda line: line.split())

result = words.map(lambda word:
(str(word.lower()).translate(None,string.punctua
tion),1))

aggreg1 = result.reduceByKey(lambda a, b: a+b)

aggreg1.saveAsTextFile("output.txt")
```

The Spark Programming Model

Parallel Execution

NUMBER OF TASKS CREATED BY SPARK TO PROCESS IN PARALLEL EACH RDD

↓

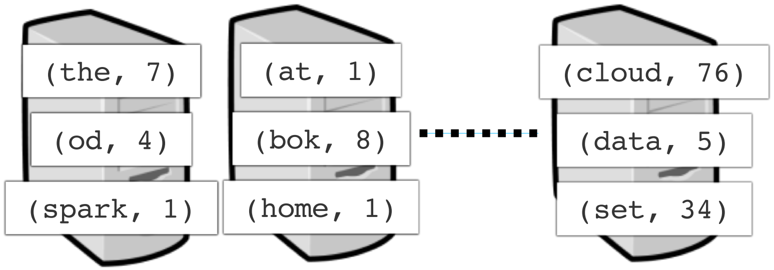
NUMBER OF NODES AND THREADS PER NODE

The Spark Programming Model

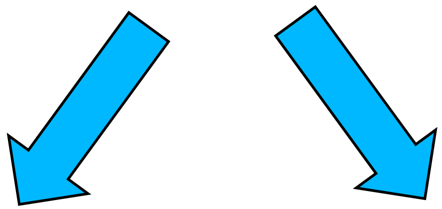
Parallel Execution

Application Parallelism

- A **task** is created to process each partition
- **RDD partitions (1)** given by the programmer (`parallelize`) for new created data, or **(2)** determined by the parent RDD, or defined by the underlying file system



`ratings.csv` is 709 MB



Local FS
709/32 = 22 partitions

HDFS
709/128 = 6partitions

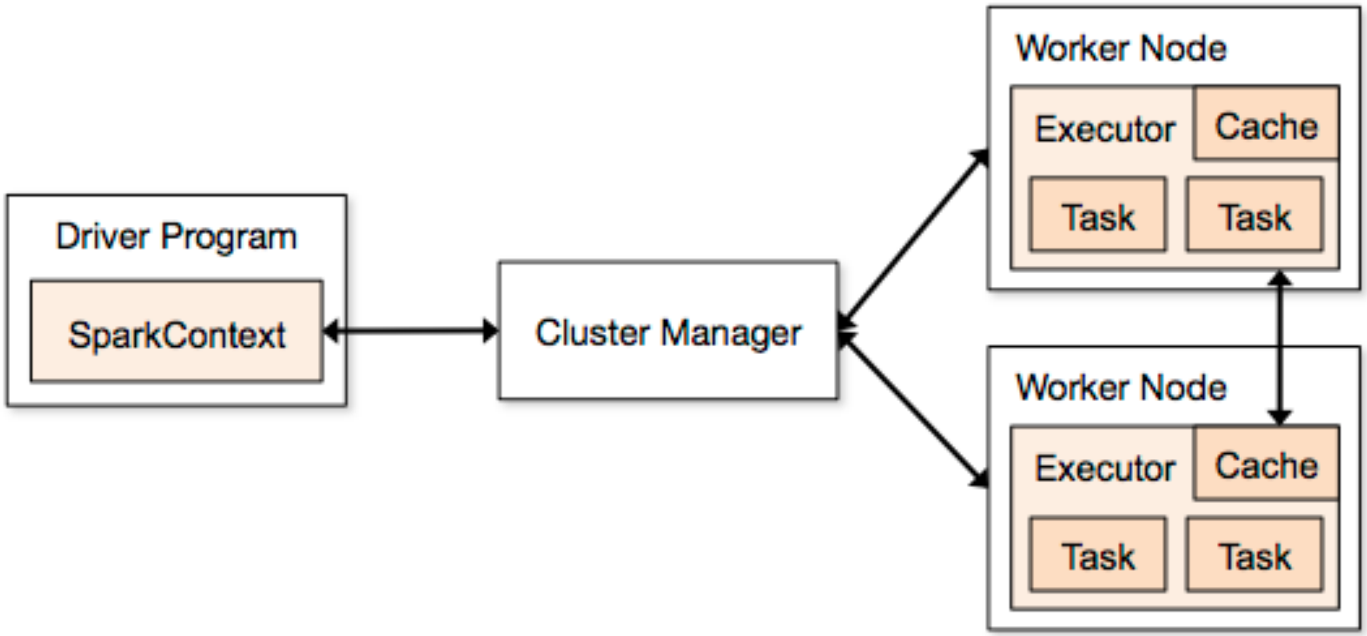
The Spark Programming Model

Parallel Execution

System Parallelism

- In **local mode** `setMaster` in the `SparkConf`
- In **cluster mode**, determined by the executors (one per node) and the threads (cores) per executor

  `--num-executors --executor-cores` in `spark-submit`



The Spark Programming Model

Parallel Execution

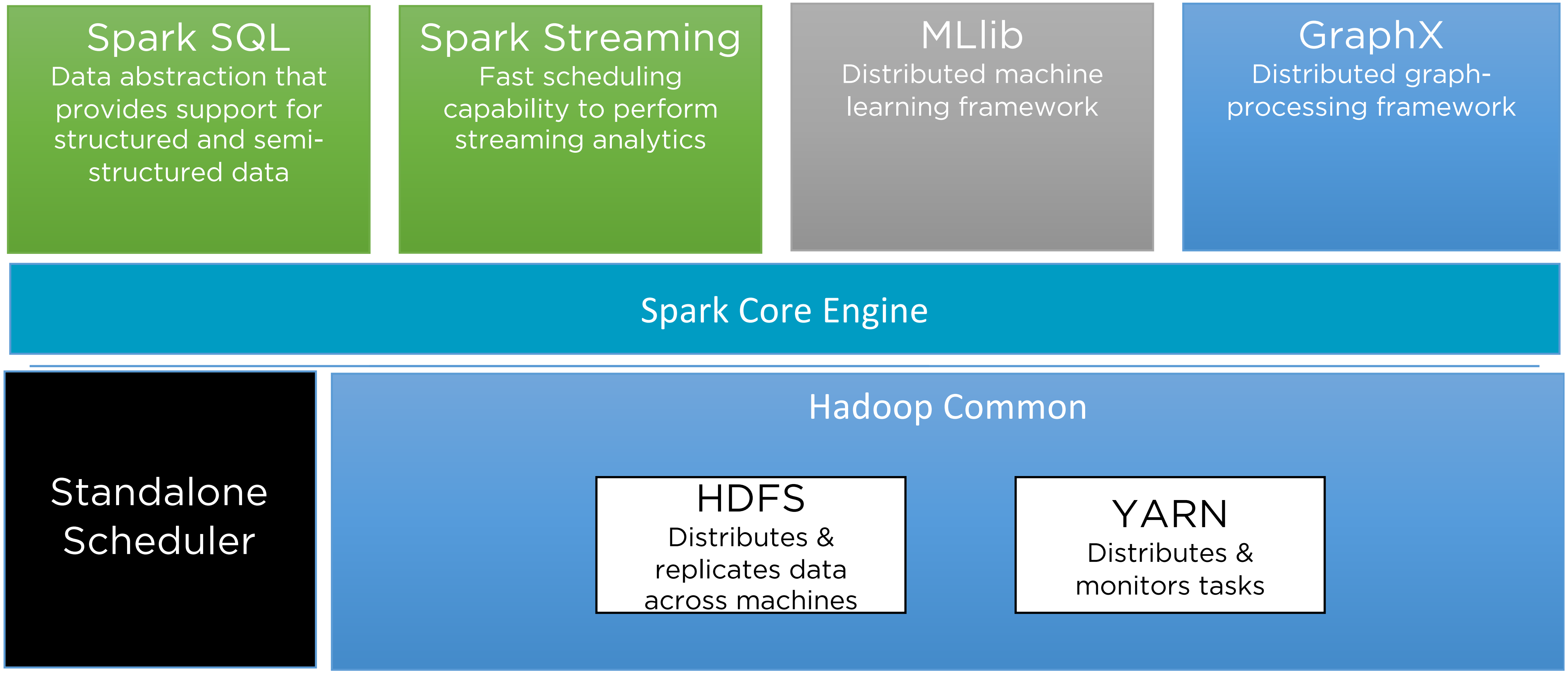
Compute PI Number with Spark

```
from pyspark import SparkConf, SparkContext
import string

conf = SparkConf().setMaster('local[2]').setAppName('Pi')
sc = SparkContext(conf = conf)

N = 10000000
delta_x = 1.0 / N
print sc.parallelize( xrange (N),4 ).map( lambda i: (i +0.5) * delta_x ).map( lambda x: 4 / (1 + x **2) ).reduce ( lambda a, b: a+b) * delta_x
```

Execute with different number of partitions and threads, and compare number of tasks and execution time

The Spark Ecosystem

Spark Ecosystem



The Spark Ecosystem

DataFrames

Higher Level Abstraction that Gives a Tabular View of Data

- Like an RDD, a DataFrame is an **immutable distributed collection of data**.
- Unlike an RDD, data is organized into a tabular format, a **two-dimensional array-like structure**.
- Makes large **data sets processing easier**, and allows Spark to run certain **optimizations** on the finalized query or processing operation.

```
df = spark.read.json("examples/src/main/resources/people.json")
```

```
# Displays the content
df.show()
# +----+-------+
# | age|   name|
# +----+-------+
# |null|Michael|
# |  30|   Andy|
# |  19| Justin|
# +----+-------+
```

```
# Select a Column
df.select("name").show()
# +-------+
# |   name|
# +-------+
# |Michael|
# |   Andy|
# | Justin|
# +-------+
```

```
# Select people older
df.filter(df['age'] > 21).show()
# +----+-------+
# | age|   name|
# +----+-------+
# |  30|   Andy|
# +----+-------+
```

The Spark Ecosystem

Machine Learning with Spark

Supervised learning

Training data contains both input vector and desired output. We also called it as labeled data.

Classification:

- Naive Bayes
- SVM
- Random Decision Forests

Regression:

- Linear Regression
- Logistic Regression

Unsupervised learning

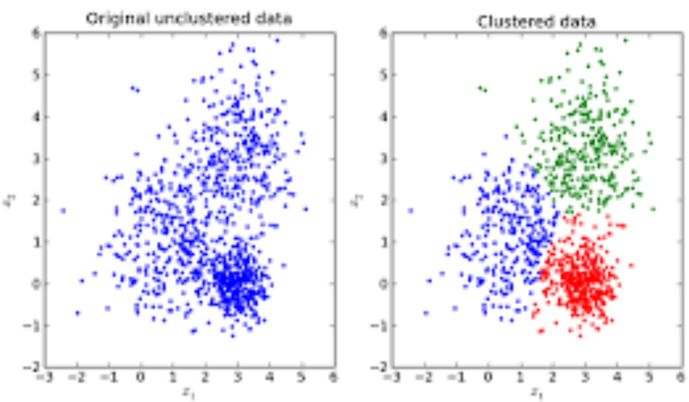
Training data sets without labels.

Clustering:

- K-means clustering

Dimensionality reduction

- Principal component analysis
- SVD



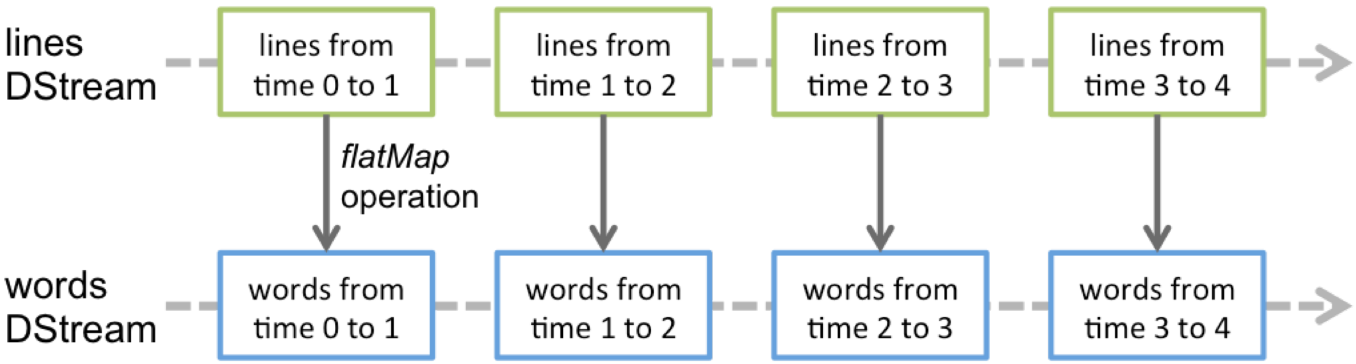
The Spark Ecosystem

Streaming with Spark

New file in dir
New content in file
TCP socket
Flume
Kafka
Kinesis

DStream (micro-batching)

- A continuous data stream is discretized into a continuous series of RDD



Next Steps

- Get ready for next **lab**:
  I9. Install Spark in Local
  I10. Spark Clusters

- Get ready for next **hands-on**:
  H6. Spark Programming (Thursday 3/30)

Questions

Dataflow Processing

http://piazza.com/harvard/spring2021/cs205/home