RDBMS, MapReduce and Spark

- With MapReduce, process data 10x faster than RDBMS
- With Spark, process data 10x faster than MapReduce

Features of MapReduce

- Simple programming model
  - No system programming (OS)
- Fault Tolerant (Reliable)
- Scalable

- Popular → Already in use across many projects

Limitations of MapReduce

- MapReduce applications are high latency jobs
- Doesn’t go well for ML type applications (applications that are iterative)
- Too many disk read writes (intermediate phases)
- No capabilities for processing live streams of data (MapReduce is a batch processing engine)
- Mostly Java was used as the programming language
- No Interactive Environment

Spark Research

- https://spark.apache.org/research.html

What is Spark?

- A general purpose, large scale, unified data processing engine
- Polyglot → Spark jobs can be written in Python, Scala, R, Java and SQL
- Spark offers capabilities to process live data streams in near real time
- Spark offers libraries for implementing machine learning
- Spark offers in-memory computation capabilities

How to launch a Spark Application?

Launch pyspark

```
$ pyspark

>>> sc

# sc --> The Spark Context object (Connection to the Spark cluster)
```

Getting started with RDDs

```
>>> sc.setLogLevel("ERROR")

>>> x = sc.textFile("/user/cloudera/Stocks")
>>> x.collect()
>>> x.take(10)
>>> x.first()

>>> for i in x.take(10): print(i)

>>> y = x.first()
>>> y
>>> type(y)
>>> y.split()
>>> y.split(',')
>>> y.split(',')[1]

# Get distinct stock symbols

>>> z = x.map(lambda y: y.split(',')[1]).distinct()
>>> z.collect()

>>> for i in z.collect(): print(i)
>>> z.count()
```

A simple Spark program

Get distinct stocks

```
# Get distinct stock symbols

>>> stocksRDD = sc.textFile("/user/cloudera/Stocks")

>>> stockSymbolRDD = stocksRDD.map(lambda y: y.split(',')[1]).distinct()

>>> stockSymbolRDD.collect()
```

Get maximum close price per stock symbol

```
# Get maximum close price per stock symbol
>>> stocksRDD = sc.textFile("/user/cloudera/Stocks")
>>> stockSymbolCloseRDD = stocksRDD.map(lambda y: (y.split(',')[1], float(y.split(',')[6])))
>>> maxClosePriceRDD = stockSymbolCloseRDD.reduceByKey(lambda a, b: round(max(a, b)))
>>> maxClosePriceRDD.collect()
```

RDD Examples

```
>>> x = sc.parallelize([(1, 2), (3, 4)])
>>> y = x.keys()
>>> y.collect()

>>> y = x.values()
>>> y.collect()

>>> x = sc.parallelize([1,2,3,4,5])
>>> y = sc.parallelize([3,4,5,6,7])

>>> z = x.union(y)
>>> z.collect()

>>> z = x.intersection(y)
>>> z.collect()

>>> z = x.subtract(y)
>>> z.collect()

>>> x = sc.parallelize([2,4,1])
>>> x.max()
>>> x.sum()
>>> x.mean()
>>> x.stdev()
>>> sc.parallelize([1, 2, 3]).variance()
>>> sc.parallelize([1, 2, 3]).stats()
```

RDD Examples

```
>>> x = sc.parallelize([("a", 1), ("b", 2)])
>>> y = sc.parallelize([("a", 3), ("a", 4), ("b", 5)])
>>> z = x.join(y)
>>> z.collect()

>>> x = sc.parallelize([1, 2])
>>> y = sc.parallelize([3, 4])
>>> z = x.cartesian(y)
>>> z.collect()
```

2 ways to create RDDs

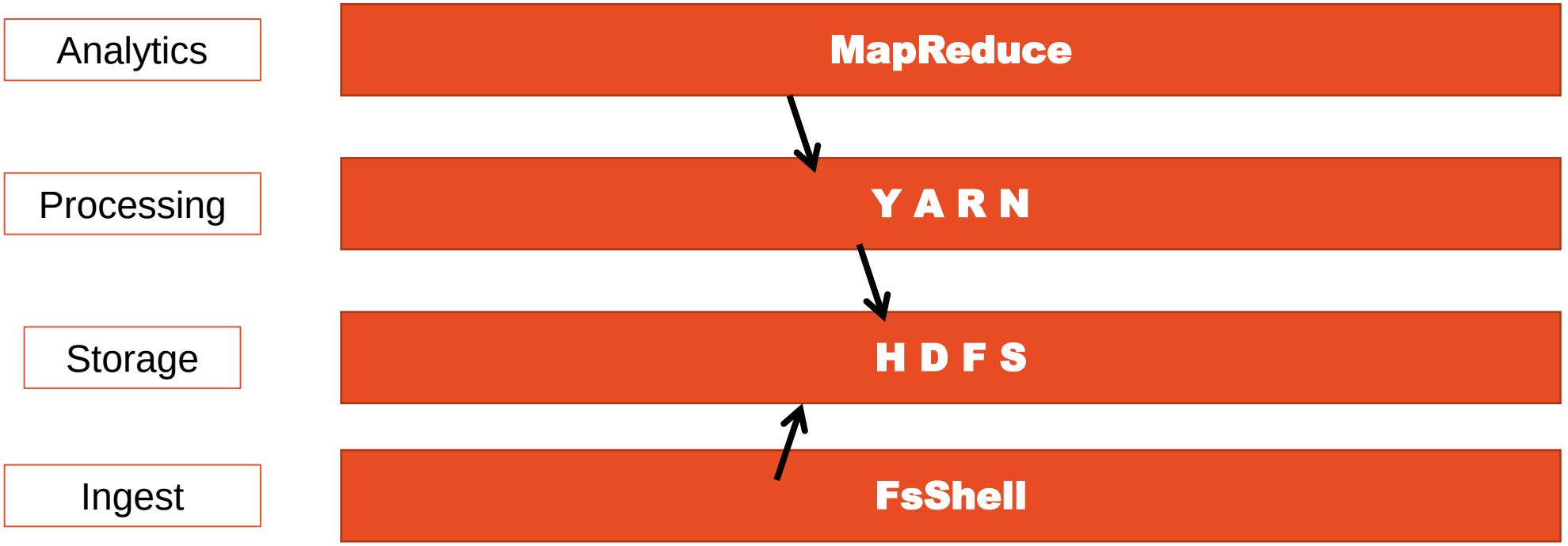
- sc.parallelize()
  - parallelize a collection
- sc.textFile()
  - reference data stored in an external storage system (Ex. HDFS)

RDDs Operations (2 types)

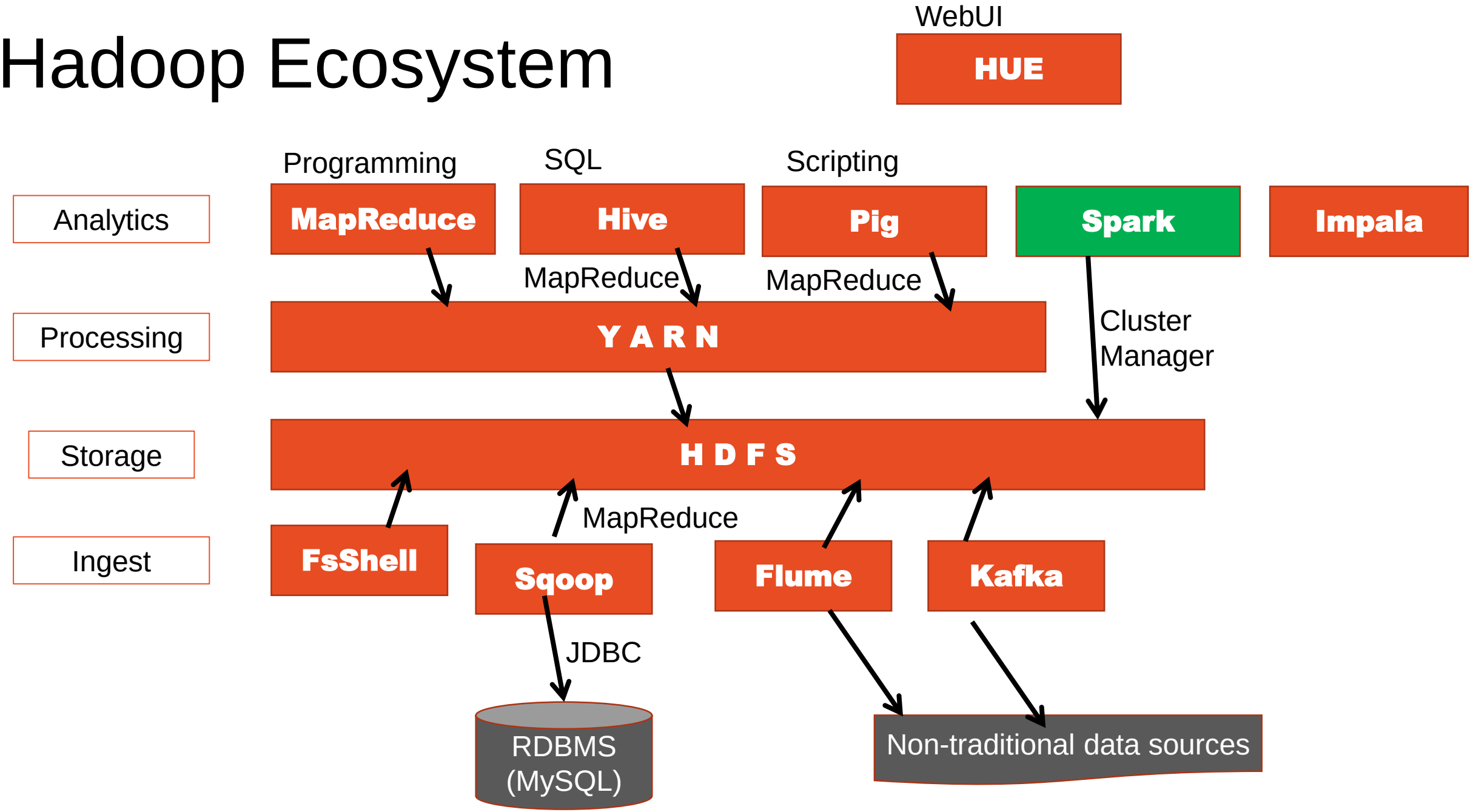
- Transformations
- Actions

- *All operations on RDDs are either 'Transformations' or 'Actions’*
- *RDDs are immutable*

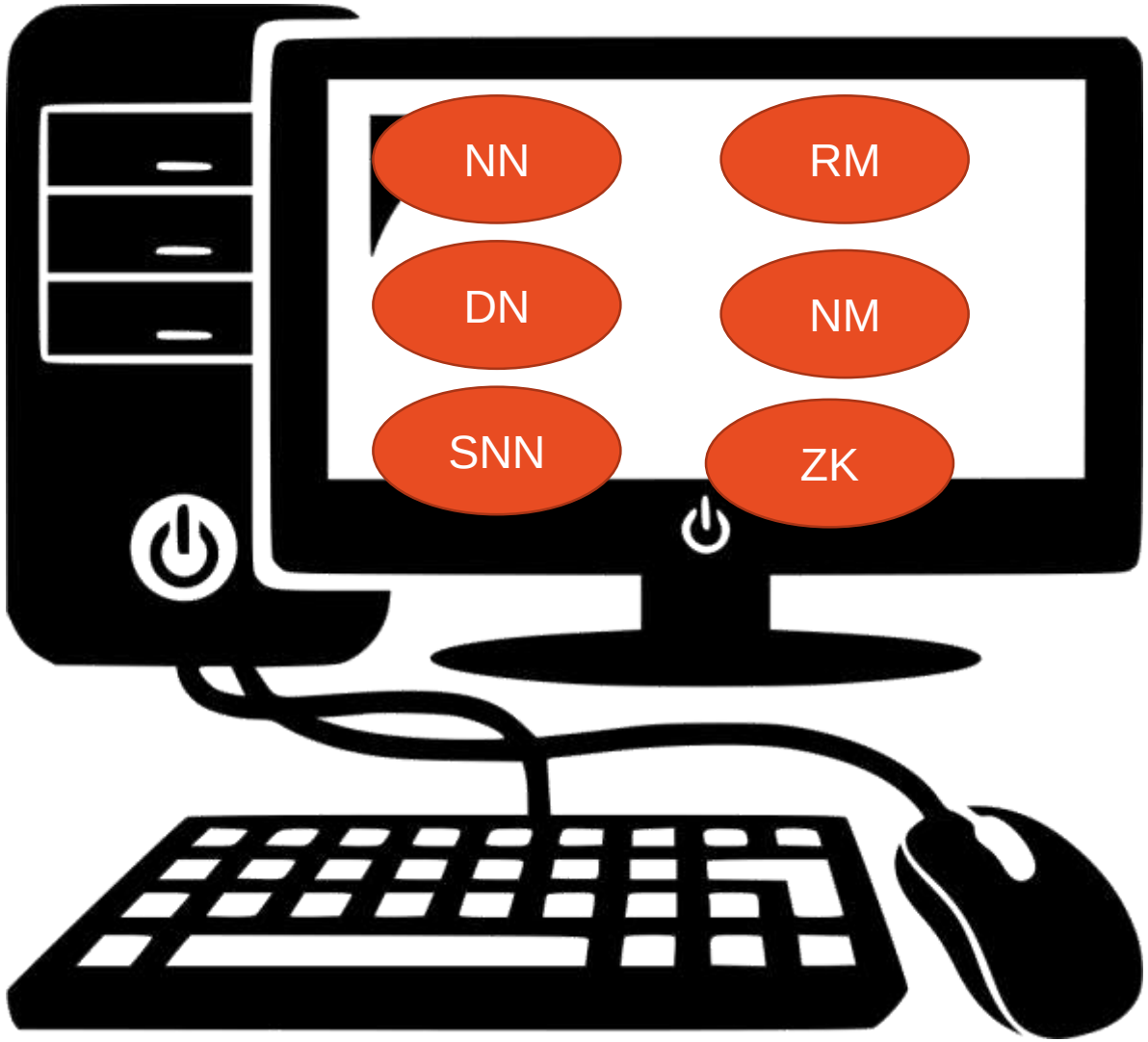
Core Hadoop

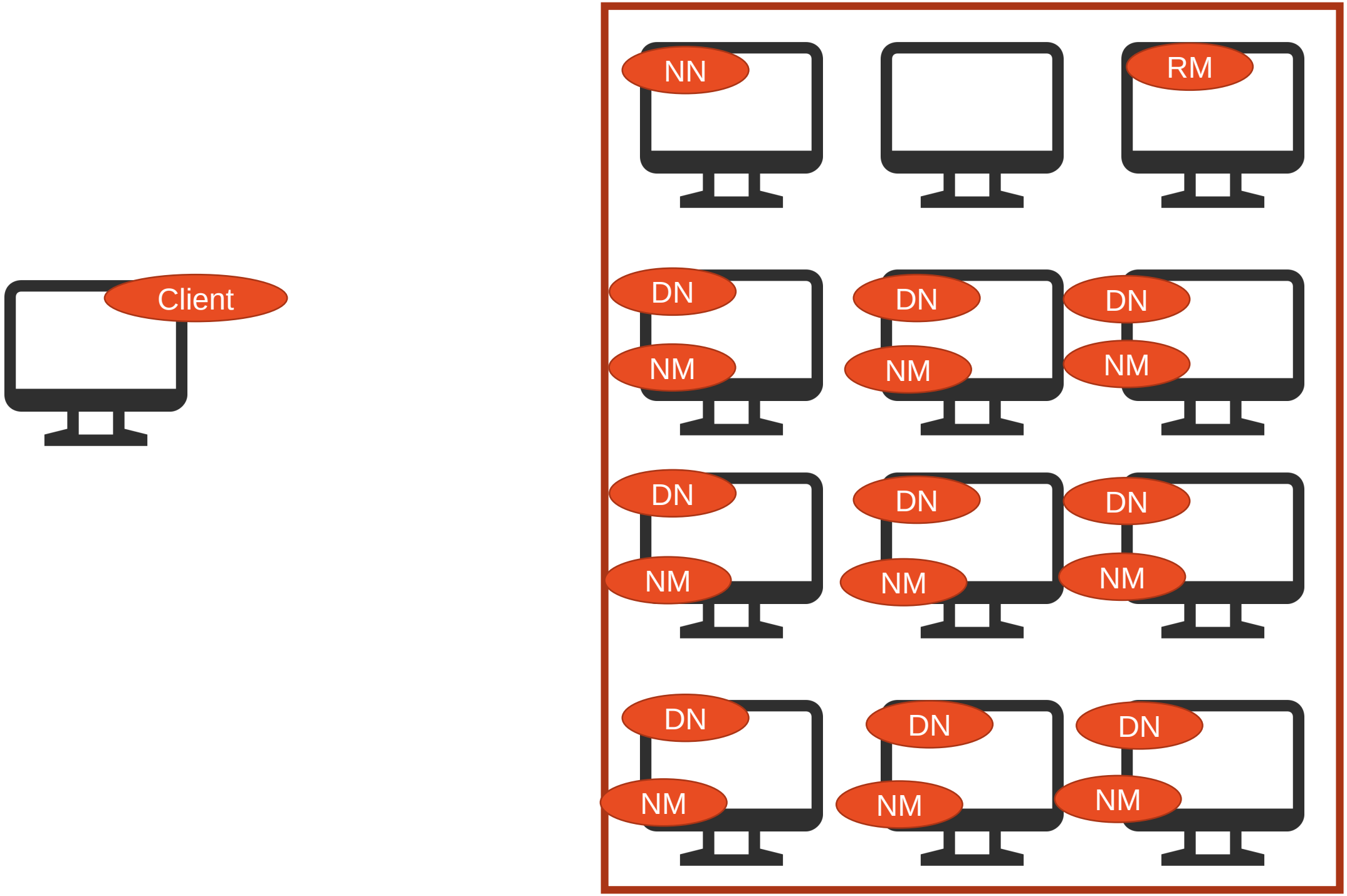

Analytics
Processing
Storage
Ingest
MapReduce
YARN
HDFS
FsShell

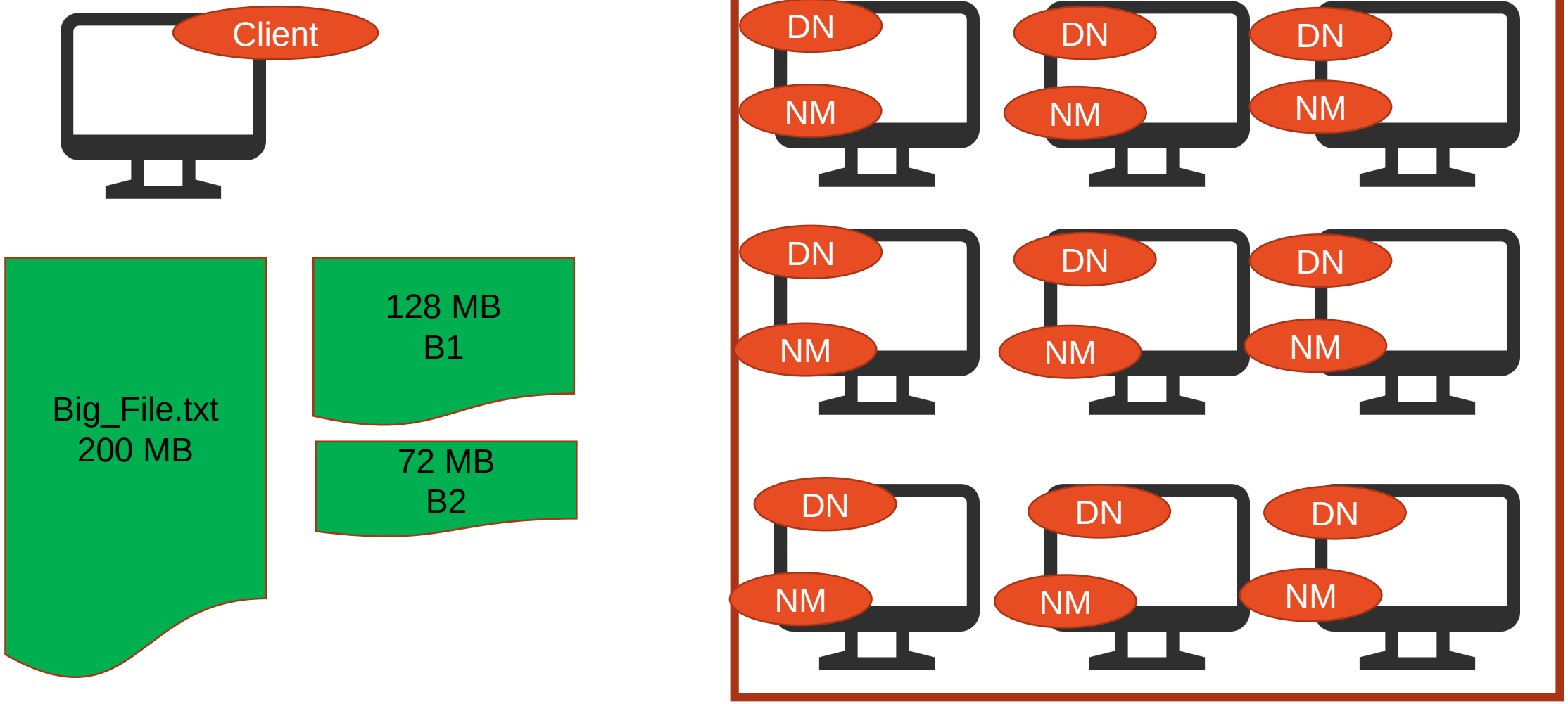
Hadoop Ecosystem



Hadoop Setup – Pseudo Distributed Mode




Client
NN
RM
DN
NM
DN
NM
DN
NM
DN
NM
DN
NM
DN
NM
DN
NM
DN
NM
DN
NM


Client
Big_File.txt
200 MB
128 MB
B1
72 MB
B2
NN
RM
DN
NM

RDD Examples

```
$ hadoop fs -mkdir orders_data

$ hadoop fs -put /home/cloudera/Downloads/orders orders_data/

# Count records in the orders dataset

$ pyspark

>>> sc.setLogLevel("ERROR")

>>> ordersRDD = sc.textFile("/user/cloudera/orders_data")

>>> ordersRDD.count()
```

RDD Examples

```
# Get distinct order_status from the orders dataset

>>> ordersRDD.first()

>>> x = ordersRDD.first()

>>> x.split(',')

>>> x.split(',')[3]

>>> ordersRDD.map(lambda x: x.split(',')[3]).distinct().collect()
```

RDD Examples

```
# Get count by order_status

>>> from operator import add

>>> ordersRDD = sc.textFile("/user/cloudera/orders_data")

>>> ordersRDD.map(lambda x: (x.split(',')[3], 1)).reduceByKey(add).collect()
```

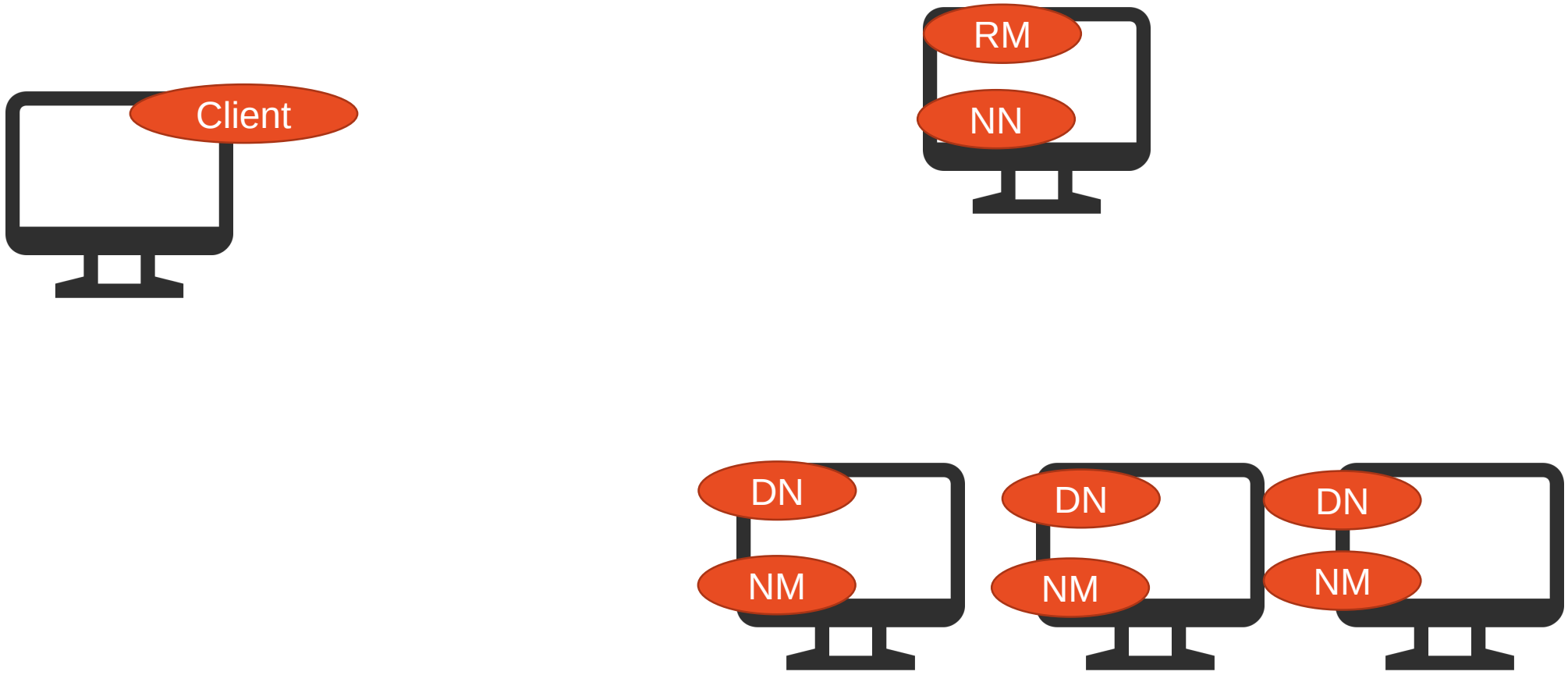
RDD Examples

```
# Get count of CLOSED and COMPLETED orders

>>> ordersRDD.filter(lambda x: (x.split(',')[3] == 'CLOSED' or x.split(',')[3]
== 'COMPLETE')).count()
```

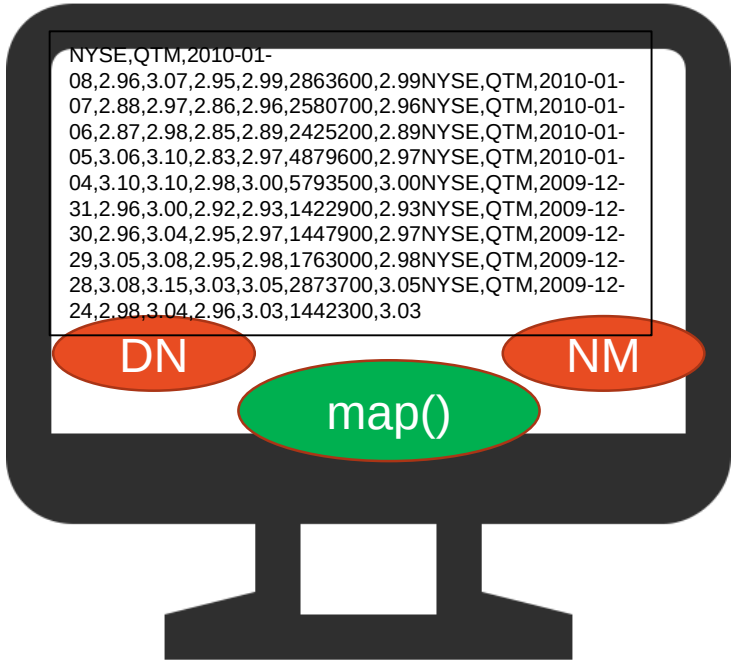
RDD Examples

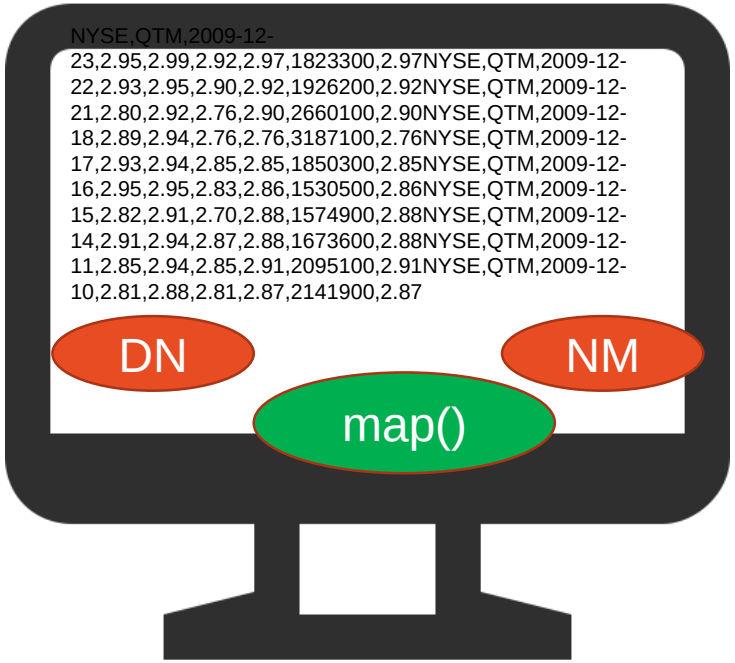
```
# Get distinct order_status from the orders dataset

>>> ordersRDD.first()

>>> x = ordersRDD.first()

>>> x.split(',')

>>> x.split(',')[3]

>>> ordersRDD.map(lambda x: x.split(',')[3]).distinct().collect()
```

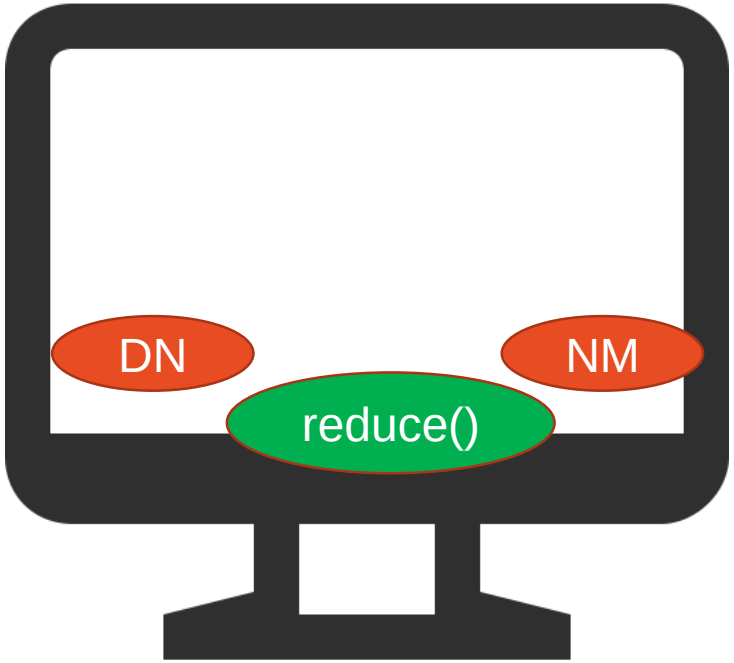
A small Hadoop cluster – 4 nodes



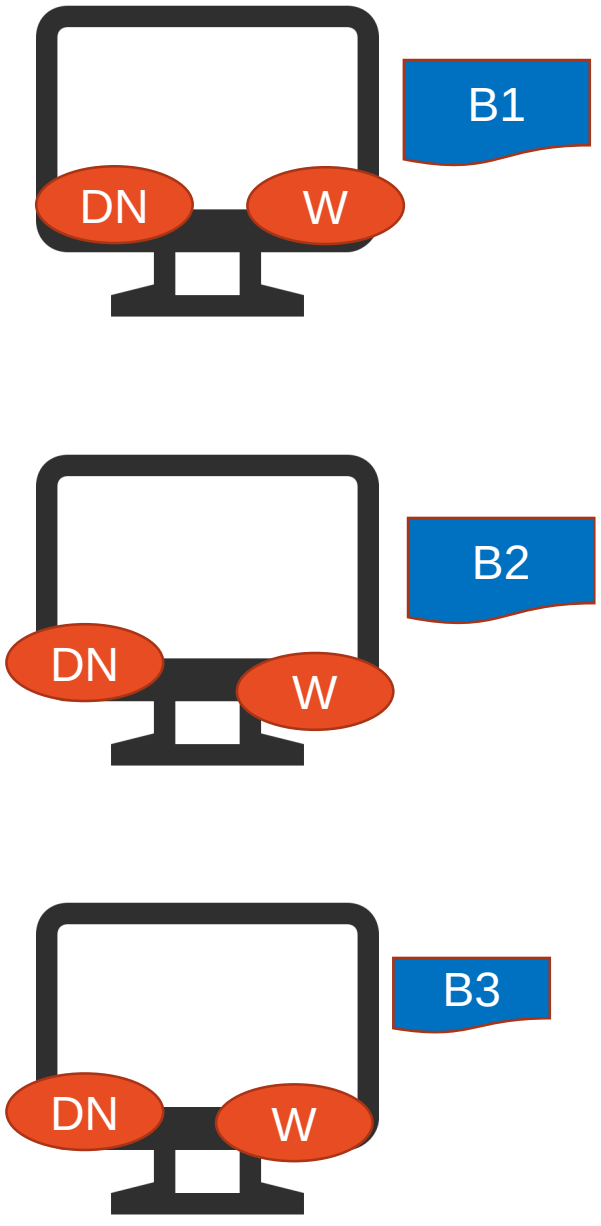
A small Hadoop cluster – 4 nodes



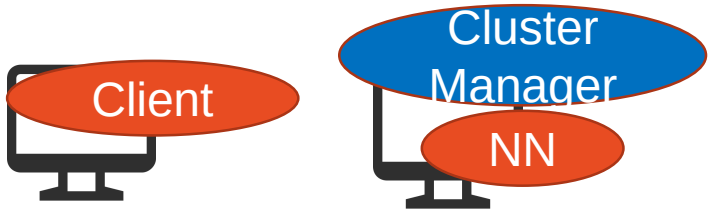


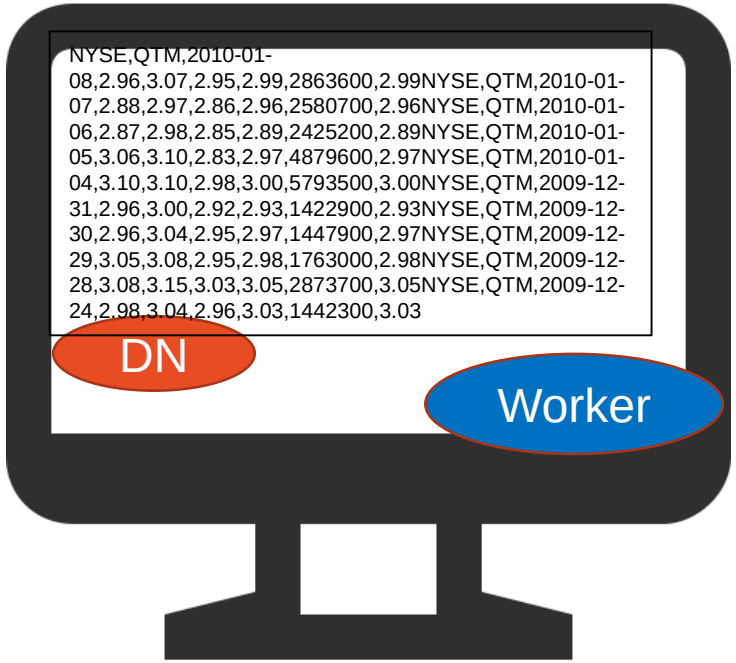


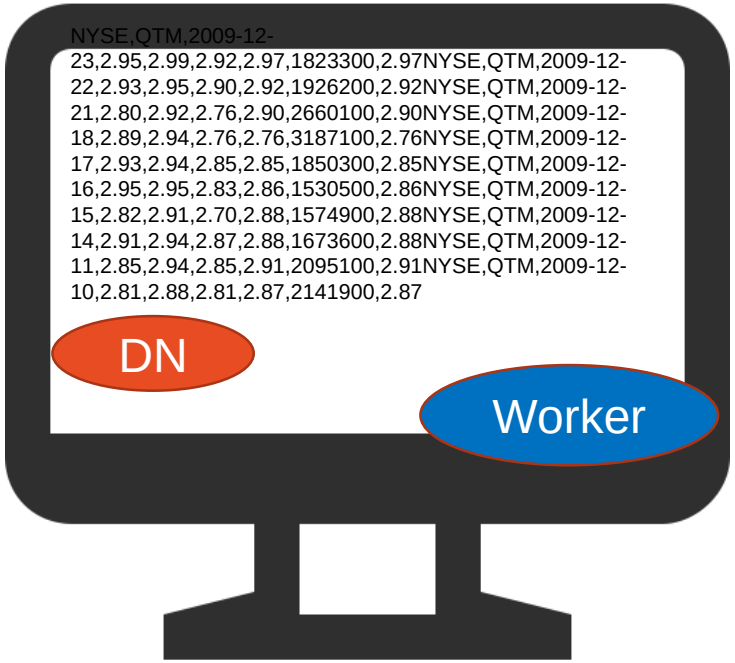
File1.txt

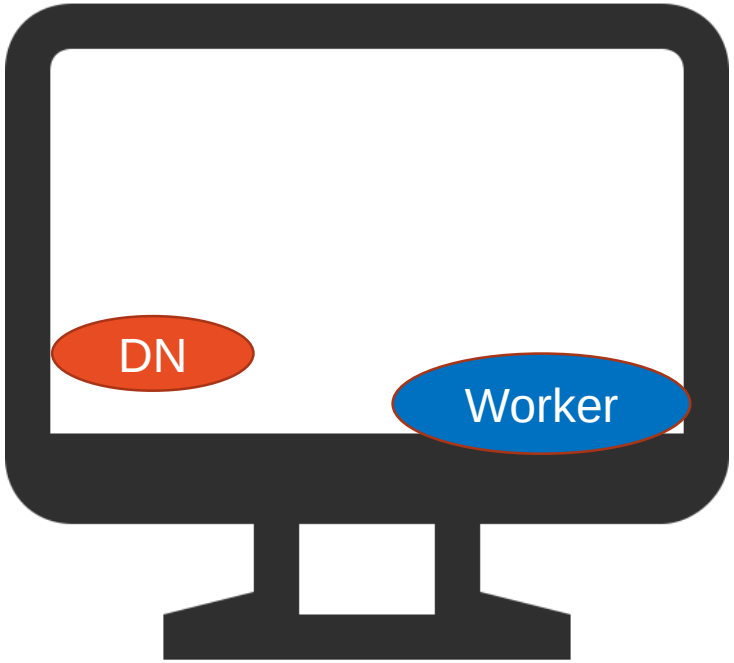


Spark on Hadoop (without YARN)









Web Service for a Spark Application

- Every Spark Application will have a webservice, and it binds on port number 4040 (by default)
- Launch Spark Application
  - $ pyspark
- Launch a browser and type in http://localhost:4040

RDD Operations

- Transformations
- Actions

RDD Operations

- Transformations
  - Transformations convert one RDD into another RDD
  - Commonly used Transformations (map, filter, flatMap etc)
  - Transformations are lazily evaluated; they do not compute the results right away. Instead, they just remember the transformations applied to the dataset

RDD Operations

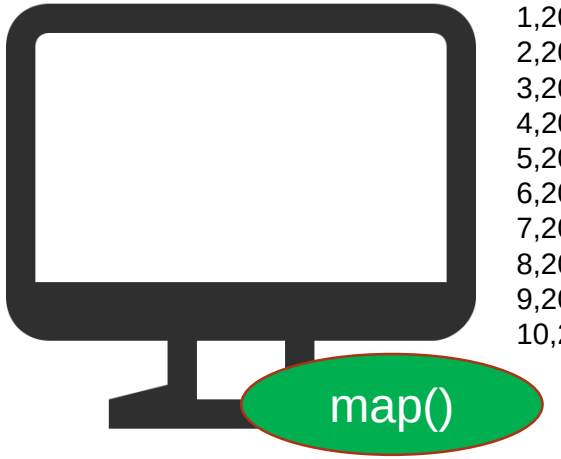
- Transformations
  - Transformations convert one RDD into another RDD
  - Commonly used Transformations (map, filter, flatMap etc)
  - Transformations are lazily evaluated; they do not compute the results right away. Instead, they just remember the transformations applied to the dataset
  - *Note: Lazy Evaluation makes Spark efficient*

RDD Operations

- Actions
  - Action computes the result from the RDD and returns it to the user
  - Commonly used Actions (collect, take, first, saveAsTextFile, reduce etc)
  - The transformations are only computed when an action needs to return a result to the user

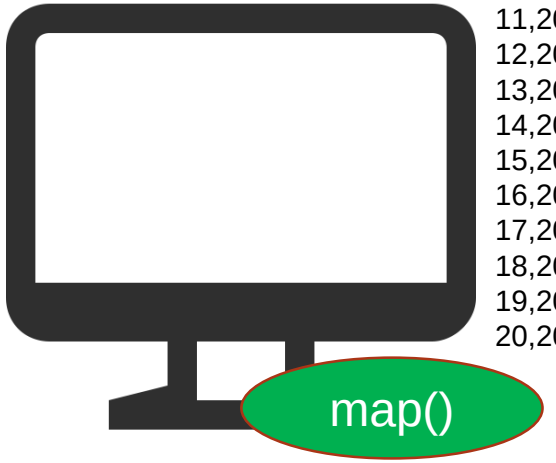
Spark remembers all the transformations and applies them when an action is called

MapReduce analogy – Orders dataset



1,2013-07-25 00:00:00.0,11599,CLOSED
2,2013-07-25 00:00:00.0,256,PENDING_PAYMENT
3,2013-07-25 00:00:00.0,12111,COMPLETE
4,2013-07-25 00:00:00.0,8827,CLOSED
5,2013-07-25 00:00:00.0,11318,COMPLETE
6,2013-07-25 00:00:00.0,7130,COMPLETE
7,2013-07-25 00:00:00.0,4530,COMPLETE
8,2013-07-25 00:00:00.0,2911,PROCESSING
9,2013-07-25 00:00:00.0,5657,PENDING_PAYMENT
10,2013-07-25 00:00:00.0,5648,PENDING_PAYMENT

CLOSED, 1
PENDING_PAYMENT, 1
COMPLETE, 1
CLOSED, 1
COMPLETE, 1….
….



11,2013-07-25 00:00:00.0,918,PAYMENT_REVIEW
12,2013-07-25 00:00:00.0,1837,CLOSED
13,2013-07-25 00:00:00.0,9149,PENDING_PAYMENT
14,2013-07-25 00:00:00.0,9842,PROCESSING
15,2013-07-25 00:00:00.0,2568,COMPLETE
16,2013-07-25 00:00:00.0,7276,PENDING_PAYMENT
17,2013-07-25 00:00:00.0,2667,COMPLETE
18,2013-07-25 00:00:00.0,1205,CLOSED
19,2013-07-25 00:00:00.0,9488,PENDING_PAYMENT
20,2013-07-25 00:00:00.0,9198,PROCESSING

PAYMENT_REVIEW, 1
CLOSED, 1
PENDING_PAYMENT, 1
PROCESSING, 1
…..

RDD examples - WordCount using Spark's Python API

```
>>> dataPath = "file:///home/cloudera/Downloads/File1.txt"

>>> rawFileRDD = sc.textFile(dataPath)

>>> wordsRDD = rawFileRDD.flatMap(lambda x: x.split())

>>> wordsMapRDD = wordsRDD.map(lambda y: (y, 1))

>>> wordCountRDD = wordsMapRDD.reduceByKey(lambda a, b: a+b)

>>> wordCountRDD.collect()
```

RDD Examples - WordCount refactored

```
>>> dataPath = "file:///home/cloudera/Downloads/File1.txt"

>>> wordCountRDD = sc.textFile(dataPath).flatMap(lambda x: x.split()).map(lambda y: (y,
    1)).reduceByKey(lambda a, b: a+b)

>>> wordCountRDD.collect()
```

Spark Non-interactive mode

- *spark-submit* shell script allows you to execute your Spark applications
- *spark-submit* utility submits your Spark application to a Spark cluster for execution

RDD Examples - Orders placed in December 2013

```
>>> ordersRDD = sc.textFile("/user/cloudera/retail_data/orders")

>>> ordersDec2013RDD = ordersRDD.filter(lambda x: x.split(',')[1][:7] == '2013-12')

>>> ordersDec2013RDD.saveAsTextFile("/user/cloudera/retail_data/OP10")
```

Orders placed in December 2013

```
>>> ordersRDD = sc.textFile("/user/cloudera/retail_data/orders")

>>> ordersDec2013RDD = ordersRDD.filter(lambda x: x.split(',')[1][:7] == '2013-12')

>>> ordersDec2013RDD.saveAsTextFile("/user/cloudera/retail_data/OP10")
```

Download orders and customers dataset from https://bit.ly/31sbX5v and store in HDFS

Write PySpark code to find customers who have not placed any orders

RDD Examples
Customers who have not placed orders

```
Write PySpark code to find customers who have not placed any orders

# orders dataset
>>> ordersRDD = sc.textFile("/orders")
>>> ordersMapRDD = ordersRDD.map(lambda x: (int(x.split(",")[2]), 1))

# customers dataset
>>> customersRDD = sc.textFile("/customers")
>>> customersMapRDD = customersRDD.map(lambda y: (int(y.split(",")[0]), y.split(",")[1]+' '+y.split(",")[2]))

# Using subtractByKey
>>> customersNoOrdersRDD = customersMapRDD.subtractByKey(ordersMapRDD)

>>> customersNoOrdersRDD.count()

>>> for i in customersNoOrdersRDD.values().collect(): print(i)
```

Spark Architecture

- Application
- Jobs
- Stages
- Tasks
- Executor
- Cache
- Cluster Manager
- Driver Program
- Spark Context

- RDD Persistence
- - Implicit Cache
- - Explicit Cache
- RDD Operations
- - Transformations
- - Actions
- RDD Dependency
- - Narrow
- - Wide
- Shuffle

RDD Examples
WordCount using Spark's Scala API

```
scala> val rawFileRDD = sc.textFile("file:///home/cloudera/Downloads/File1.txt")

scala> val wordsRDD = rawFileRDD.flatMap(x => x.split(" "))

scala> val wordsMapRDD = wordsRDD.map(y => (y, 1))

scala> val wordCountRDD = wordsMapRDD.reduceByKey((a, b) => a+b)

scala> wordCountRDD.collect()

scala> wordCountRDD.collect.foreach(println)
```

```
# Download file1.txt from https://bit.ly/31sbX5v

$ hadoop fs -mkdir /user/cloudera/SampleData

$ hadoop fs -D dfs.blocksize=10m -put Downloads/File1.txt SampleData/

$ spark-shell

scala> val rawFileRDD = sc.textFile("/user/cloudera/SampleData")

scala> rawFileRDD.getNumPartitions
```

Download dataset from https://bit.ly/31sbX5v and store in HDFS

Write PySpark code to find revenue for every order id

Using PySpark RDD API, compute revenue for every order id

```
Using PySpark RDD API, compute revenue for every order id


>>> orderItemsRDD = sc.textFile("user/cloudera/order_items")

>>> orderItemsMapRDD = orderItemsRDD.map(lambda x: (int(x.split(",")[1]), round(float(x.split(",")
[4]),2)))

>>> from operator import add

>>> revenueForEachOrderId = orderItemsMapRDD.reduceByKey(add, 1)

>>> revenueForEachOrderId.saveAsTextFile("/user/cloudera/revenue_per_order")
```

>>> revenueSortedRDD = revenueForEachOrderId.sortBy(lambda x: x[1], False)

>>> for i in revenueSortedRDD.take(10): print(i)

What is SparkContext?

- Every Spark application has a driver program
- The program consists of a bunch of instructions to Spark
- Driver program uses a SparkContext to communicate with the Spark cluster
- Spark needs a cluster manager to manage the compute resources on the nodes of the cluster
- The driver program uses SparkContext to contact the cluster manager

What is Cluster Manager?

- Cluster manager helps Spark in acquiring compute resources (called Executors) on the worker nodes
- Cluster manager is responsible for cluster resource management
- Spark supports the following cluster manager types
  - Standalone --> Cluster Manager that is included with Spark
  - YARN --> Hadoop's MapReduce engine, that can also run MapReduce jobs. (Spark in YARN mode)
  - Mesos --> Is the distributed systems kernel that handles resource management and scheduling across entire datacenter and cloud environments

What are Executors?

- Compute resources on worker nodes
- Cluster manager launches Java processes on worker nodes called executors
- Once the executors are launched, they register with the driver program and then they are ready to execute instructions
- Executors stay up for the entire duration of the application

Components of SparkContext

- DAG Scheduler
  - DAG Scheduler breaks down the job into smaller units of work (Stages and Tasks)
  - Creates a blueprint of execution
- Task Scheduler
  - Tasks are the smallest units of execution and are assigned to the Executors by the Task Scheduler

Using PySpark RDD API, compute revenue for every order id

```
Using PySpark RDD API, compute revenue for every order id


>>> orderItemsRDD = sc.textFile("user/cloudera/order_items")

>>> orderItemsMapRDD = orderItemsRDD.map(lambda x: (int(x.split(",")[1]), round(float(x.split(",")
[4]),2)))

>>> from operator import add

>>> revenueForEachOrderId = orderItemsMapRDD.reduceByKey(add, 1)

>>> revenueForEachOrderId.saveAsTextFile("/user/cloudera/revenue_per_order")
```

Download commodities_data.csv dataset from https://bit.ly/31sbX5v and store in HDFS

Write PySpark code to list commodity with highest price index annually