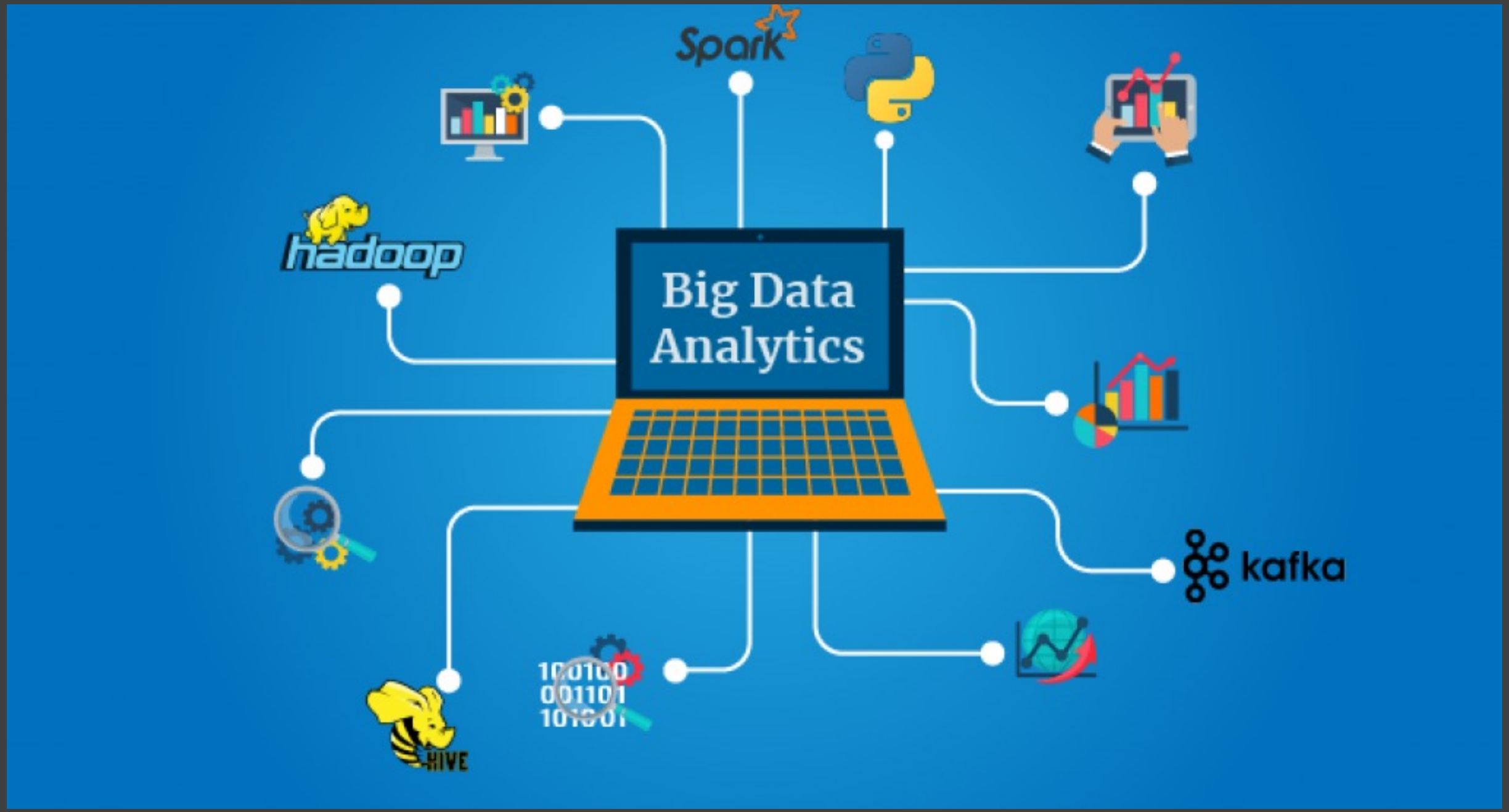
Cloud-based Data Analytics

Lecture 2

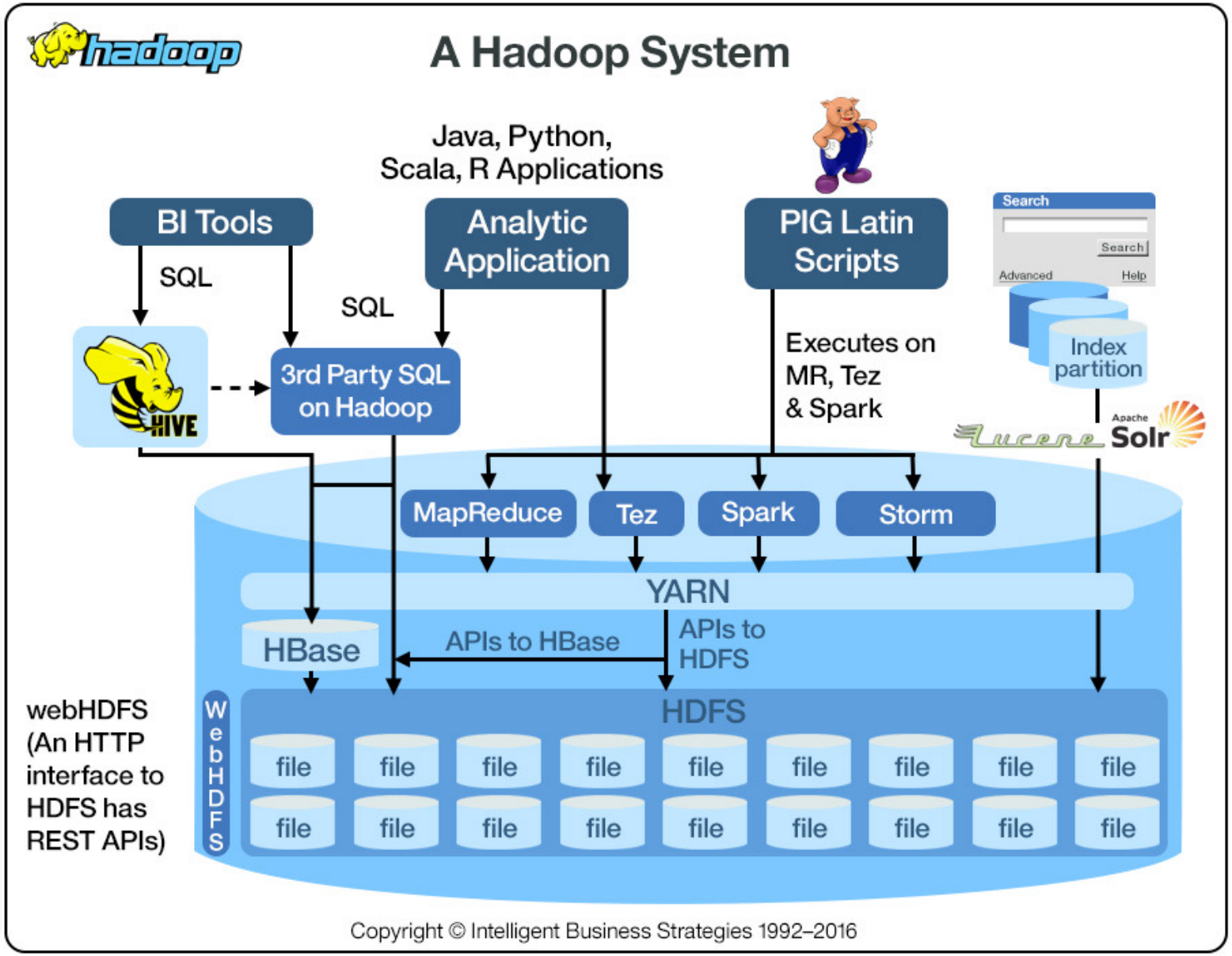
Content of the lecture

- We will cover the goals of Hadoop and HDFS.
- We also cover MapReduce and KMeans Algorithm.
- Examples and applications of MapReduce.

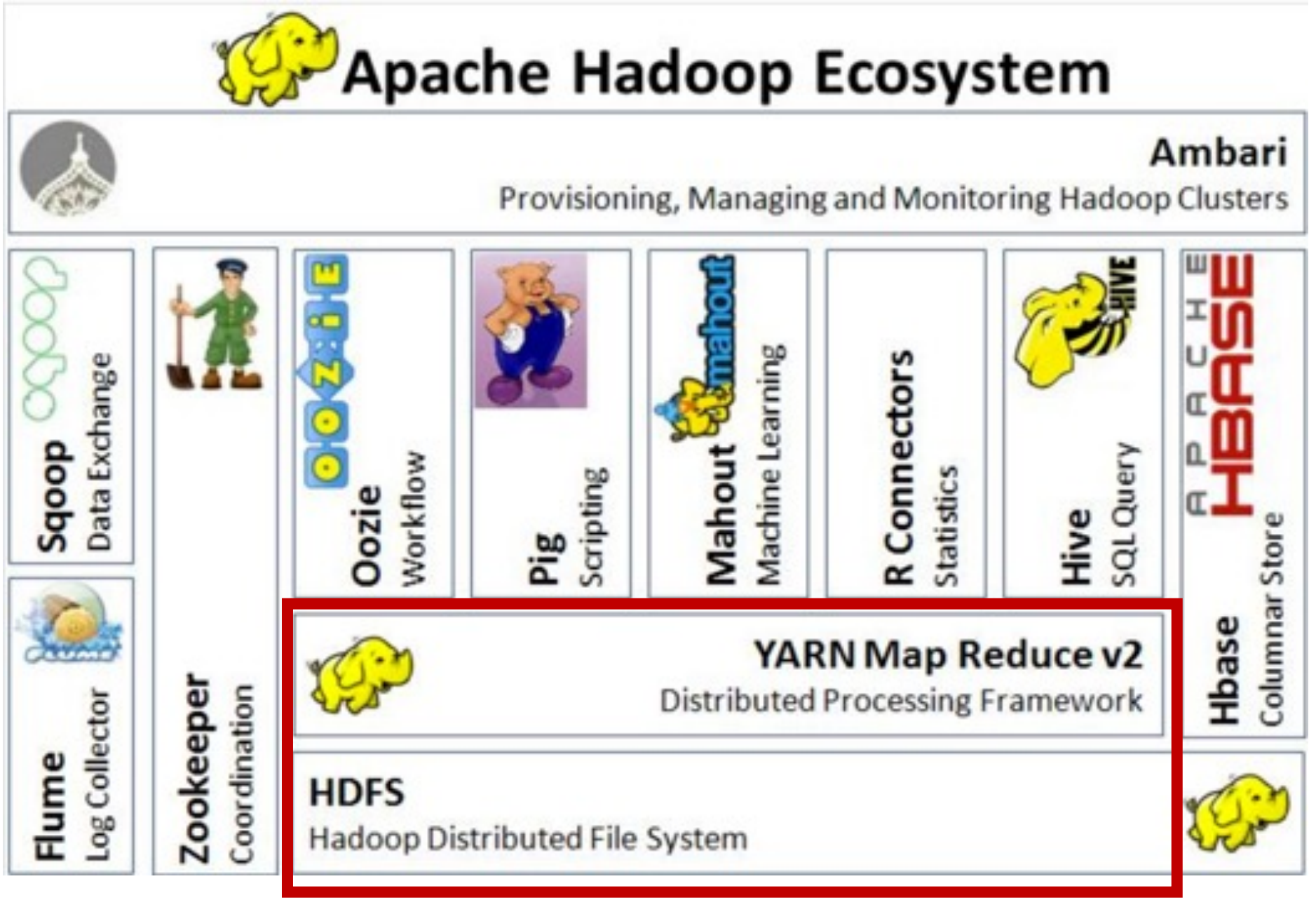

Spark
hadoop
Big Data
Analytics
kafka
HIVE

Hadoop

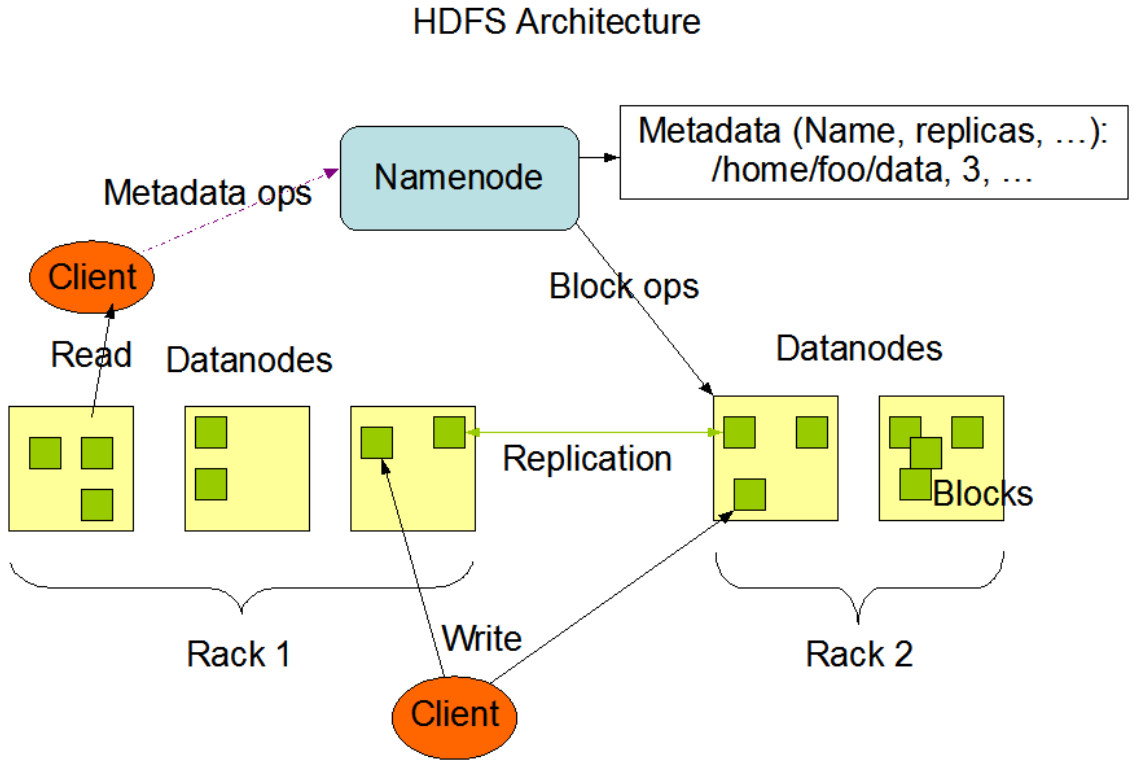
- Hadoop was created by Doug Cutting and Mike Cafarella in 2005
- It was originally developed to support distribution of the Nutch Search Engine Project.
- Apache Hadoop is an open-source software framework for storage and large-scale data processing of the datasets on clusters of commodity hardware.



Hadoop Ecosystem Major Components

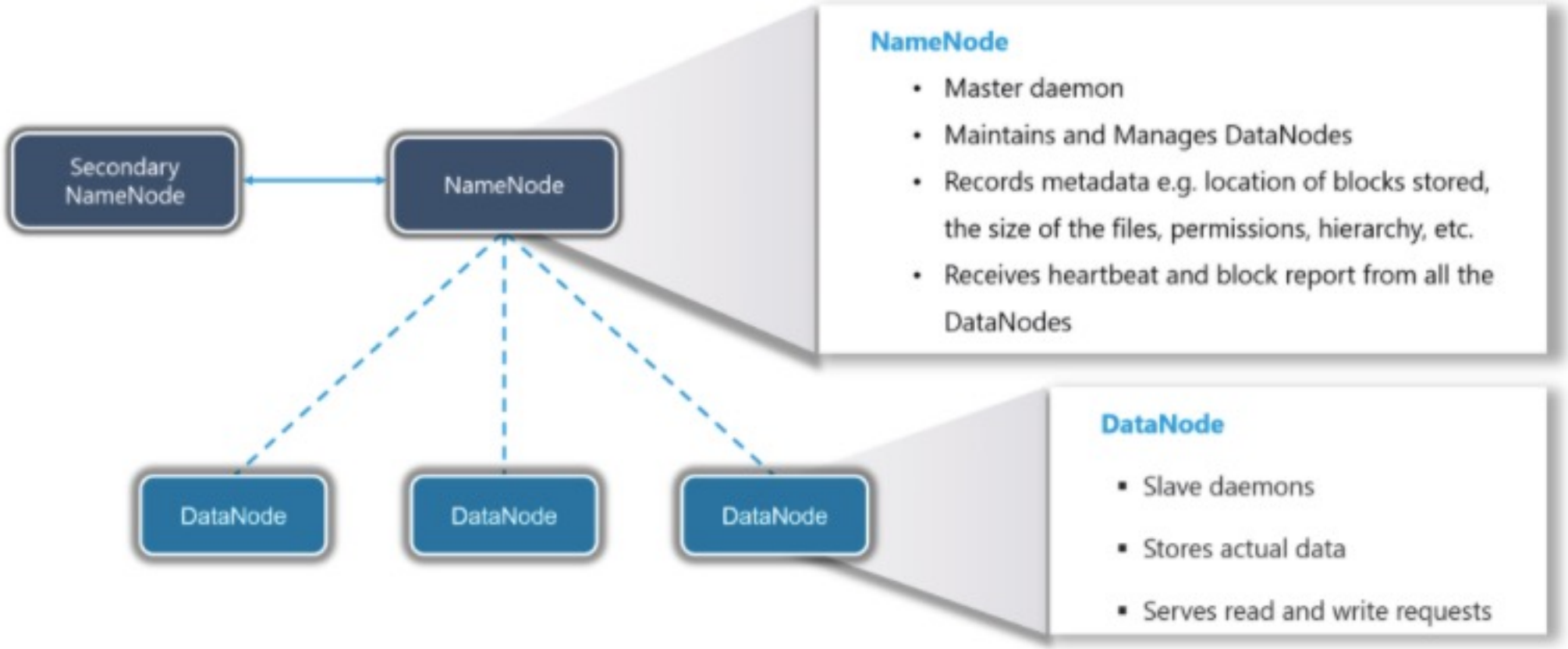

Apache Hadoop Ecosystem
Ambari
Provisioning, Managing and Monitoring Hadoop Clusters
Sqoop
Data Exchange
Flume
Log Collector
Zookeeper
Coordination
Oozie
Workflow
Pig
Scripting
Mahout
Machine Learning
R Connectors
Statistics
Hive
SQL Query
Hbase
Columnar Store
YARN Map Reduce v2
Distributed Processing Framework
HDFS
Hadoop Distributed File System

Hadoop Distributed File System



The Hadoop Distributed File System (HDFS) is a distributed file system designed to run on commodity hardware. It has many similarities with existing distributed file systems. HDFS is highly fault-tolerant and is designed to be deployed on low-cost hardware.

Hadoop Distributed File System



HDFS creates an abstraction just like virtualization. HDFS logically is a single unit for storing Big Data, but in actual you are storing your data across multiple nodes in a distributed fashion. HDFS follows master-slave architecture.

YARN

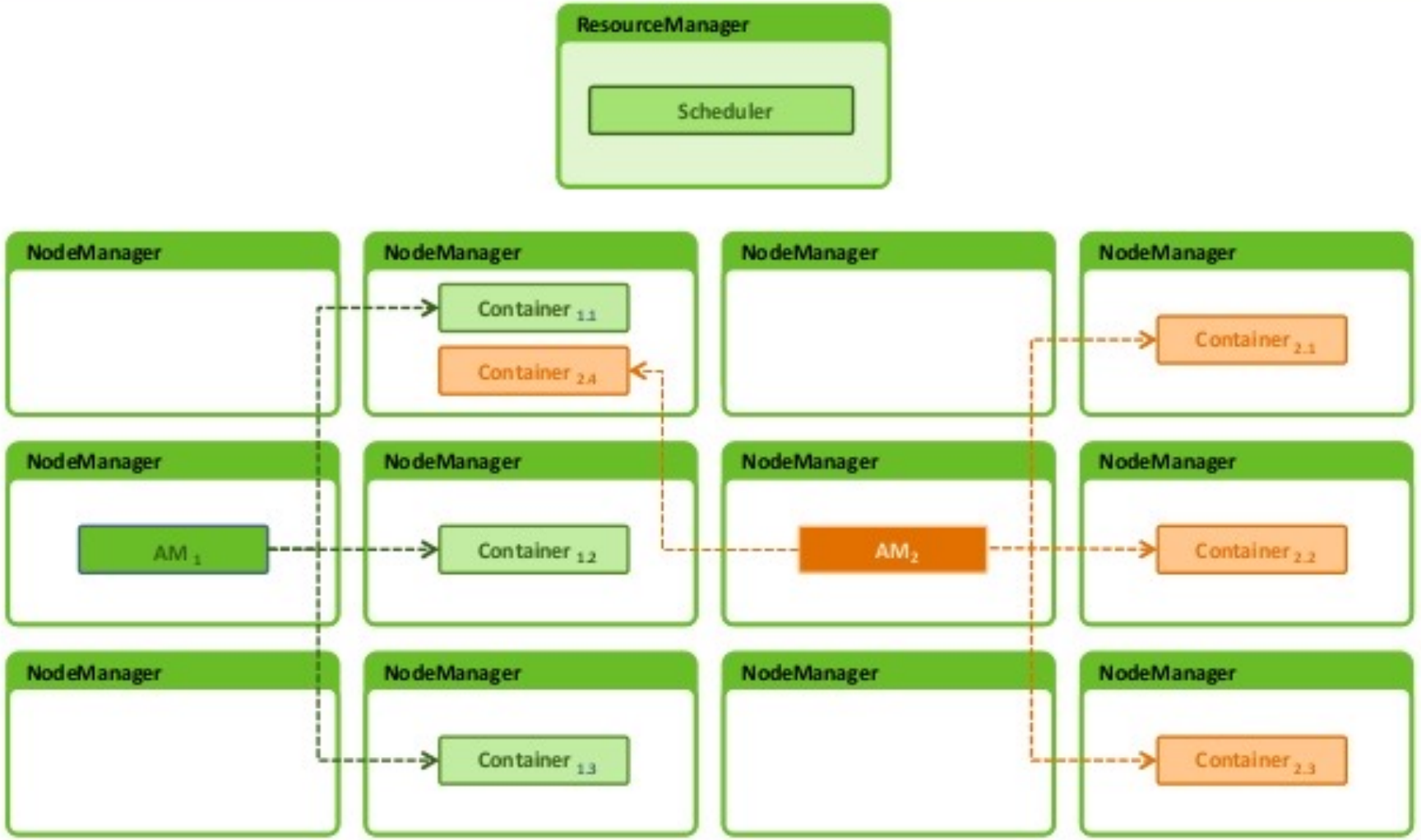
- YARN – Yet Another Resource Negotiator
- Apache Hadoop YARN is the resource management and job scheduling technology in the open-source Hadoop distributed processing framework.
- YARN is responsible for allocating system resources to the various applications running in a Hadoop cluster and scheduling tasks to be executed on different cluster nodes.

YARN Components

Apache Hadoop YARN Architecture consists of the following main components:

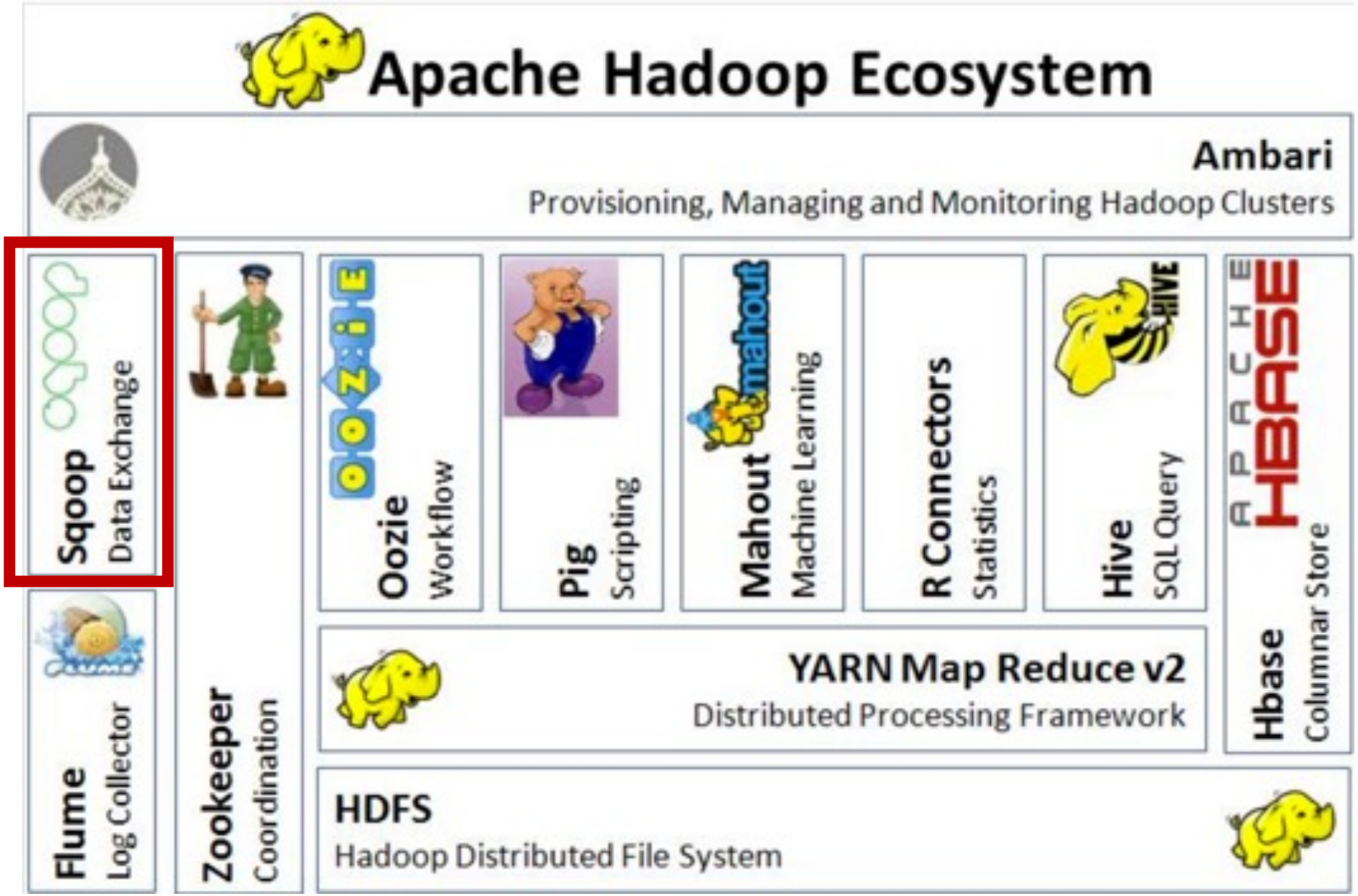
- **Resource Manager**: Runs on a master daemon and manages the resource allocation in the cluster.
- **Node Manager**: They run on the slave daemons and are responsible for the execution of a task on every single Data Node.
- **Application Master**: Manages the user job lifecycle and resource needs of individual applications. It works along with the Node Manager and monitors the execution of tasks.
- **Container**: Package of resources including RAM, CPU, Network, HDD etc. on a single node.

YARN Architecture



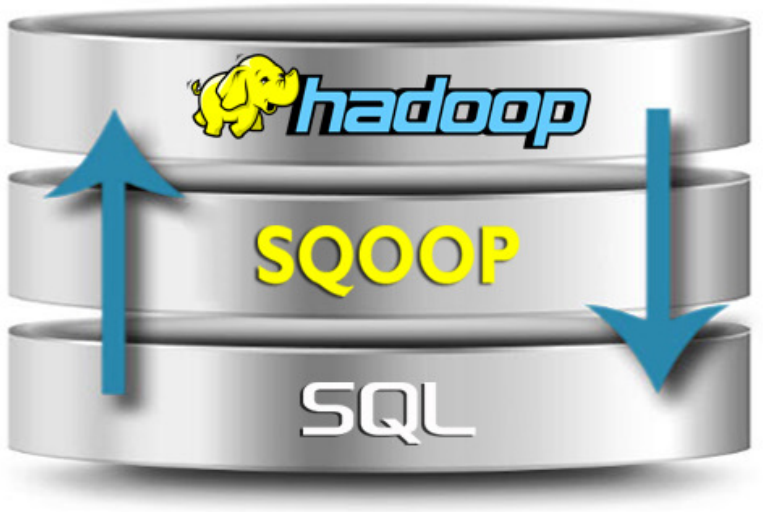
MapReduce

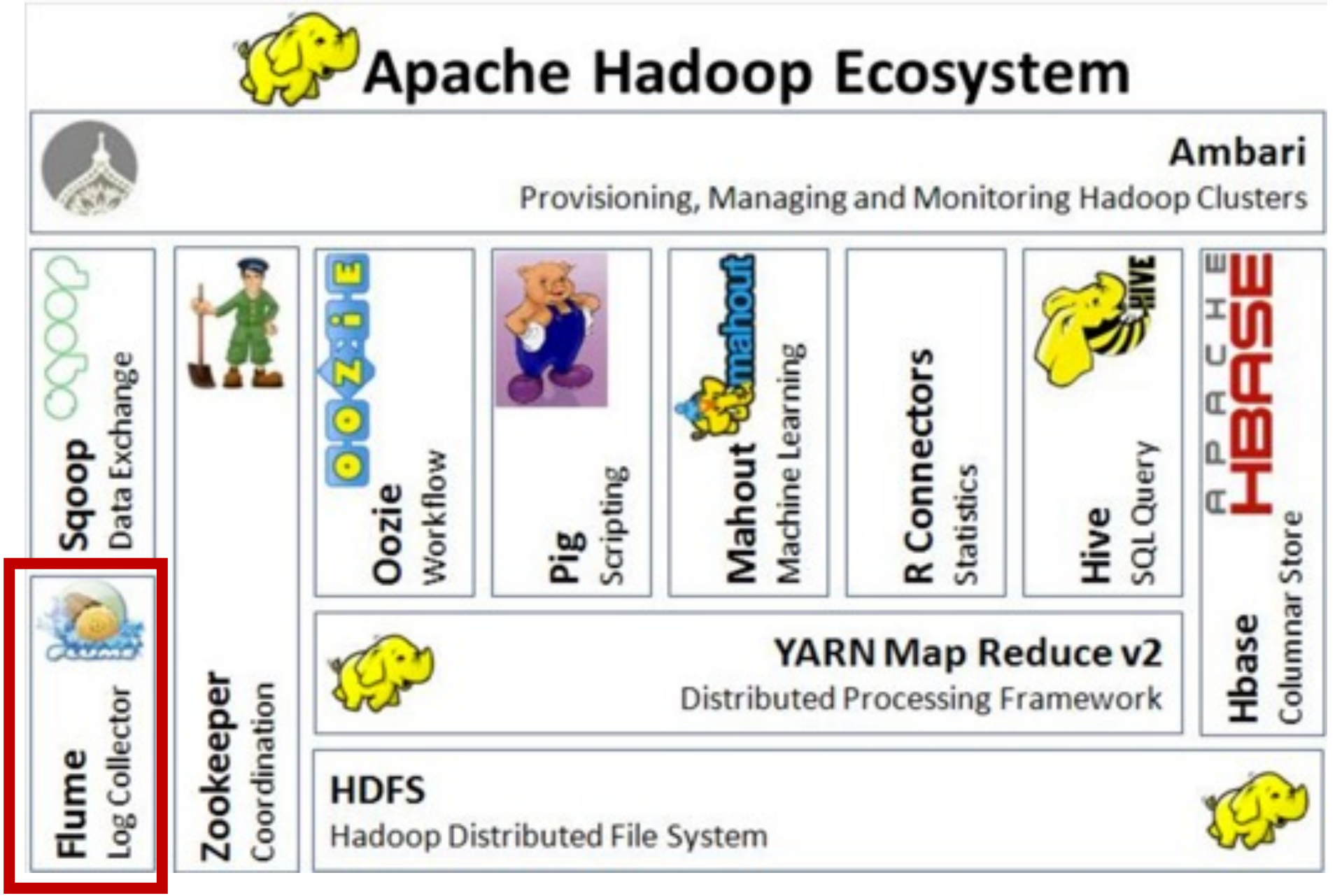
- MapReduce is a programming model and an associated implementation for processing and generating large datasets.
- Users Specify a map function that processes a key/value pair to generate a set of intermediate key/value pairs, and a reduce function that merges all intermediate values associated with the same intermediate key.


Apache Hadoop Ecosystem
Ambari
Provisioning, Managing and Monitoring Hadoop Clusters
Sqoop
Data Exchange
Flume
Log Collector
Zookeeper
Coordination
Oozie
Workflow
Pig
Scripting
Mahout
Machine Learning
R Connectors
Statistics
Hive
SQL Query
Hbase
Columnar Store
YARN Map Reduce v2
Distributed Processing Framework
HDFS
Hadoop Distributed File System

Apache Sqoop

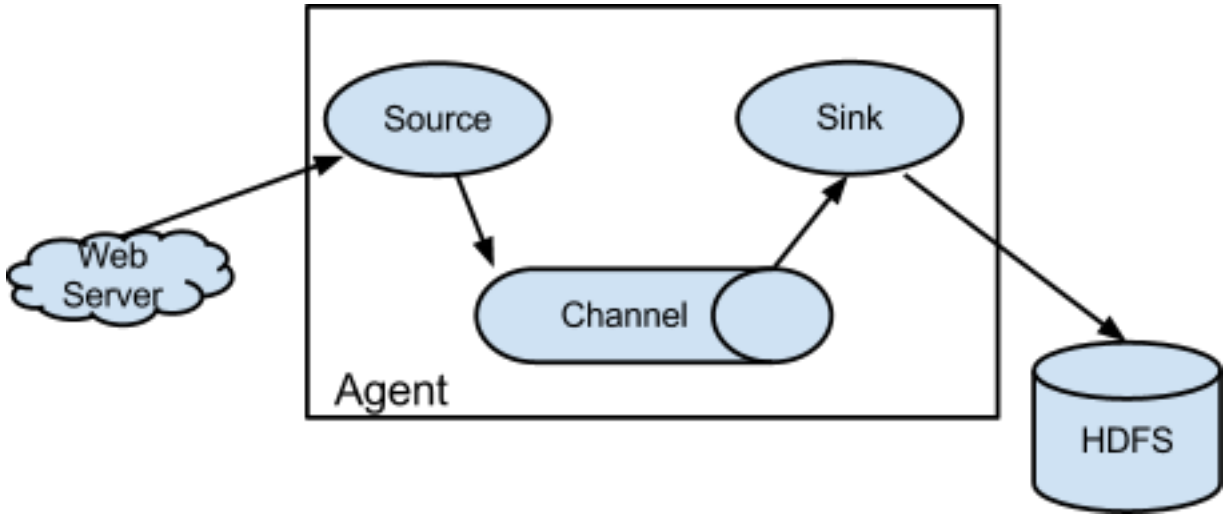
- Tool designed for efficiently transferring bulk data between Apache Hadoop and structured datastores such as relational databases.

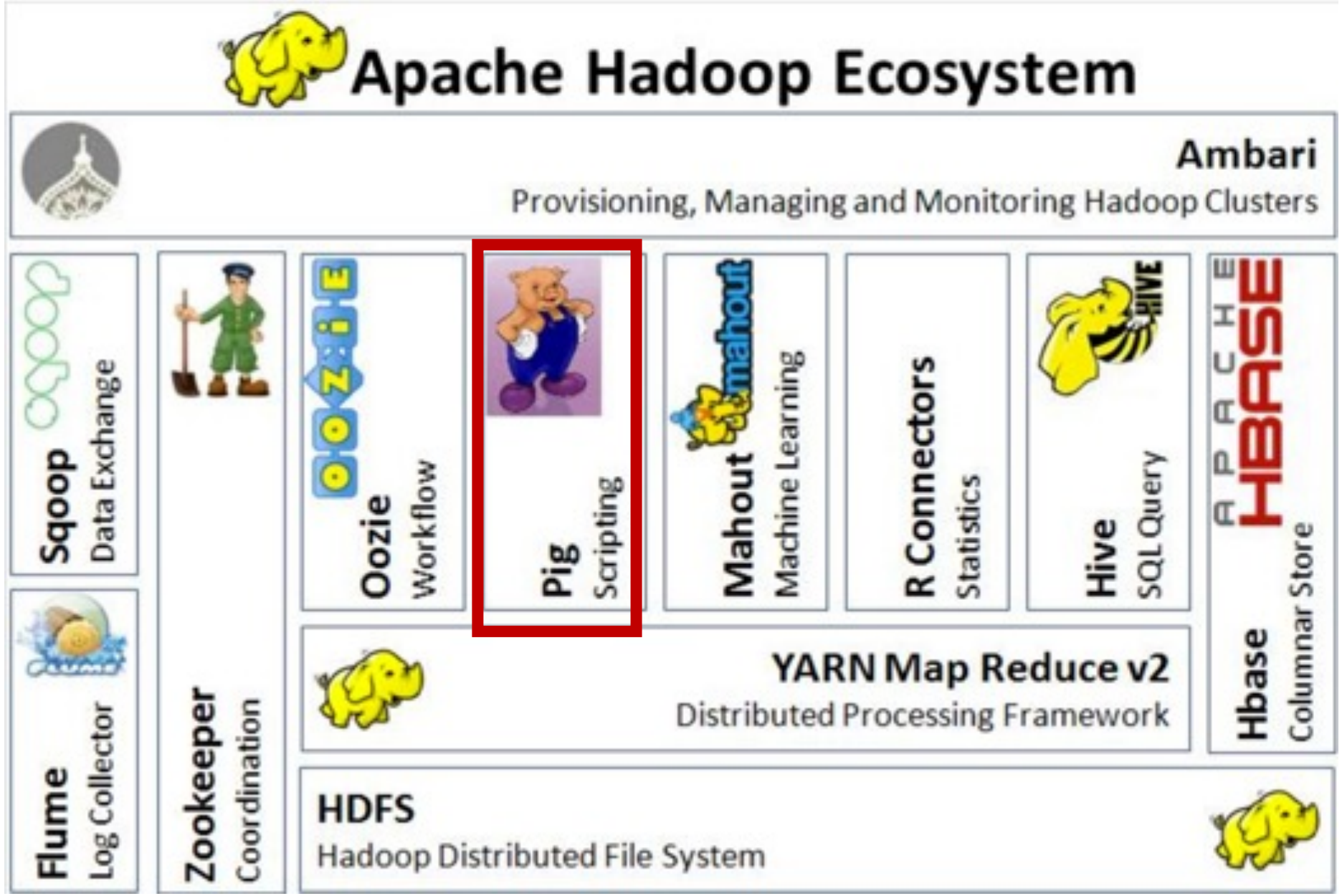



Apache Hadoop Ecosystem
Ambari
Provisioning, Managing and Monitoring Hadoop Clusters
Sqoop
Data Exchange
Flume
Log Collector
Zookeeper
Coordination
Oozie
Workflow
Pig
Scripting
Mahout
Machine Learning
R Connectors
Statistics
Hive
SQL Query
Hbase
Columnar Store
YARN Map Reduce v2
Distributed Processing Framework
HDFS
Hadoop Distributed File System

Flume

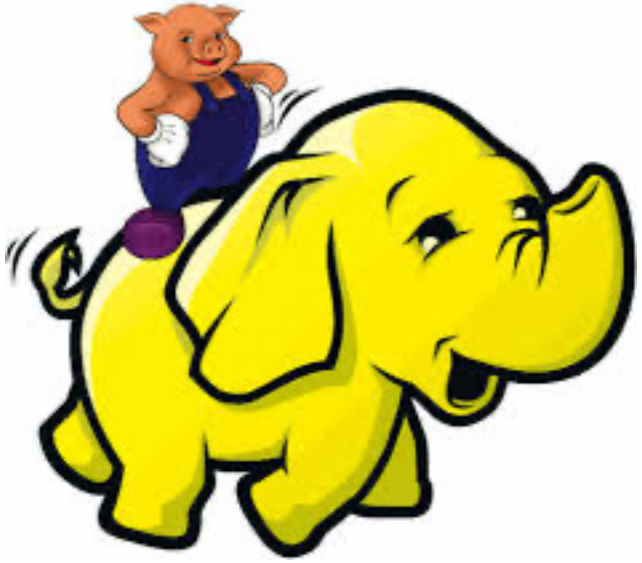
- Distributed, reliable and available service for efficiently collecting , aggregating and moving large amounts of log data
- It has a simple and very flexible architecture based on streaming data flows. Its quite robust and fall tolerant and its tunable to enhance the reliability mechanisms, fail over, recovery and all the other mechanisms that keep the cluster safe and reliable.
- It uses simple extensible data model that allows us to apply all kinds of online analytic applications.

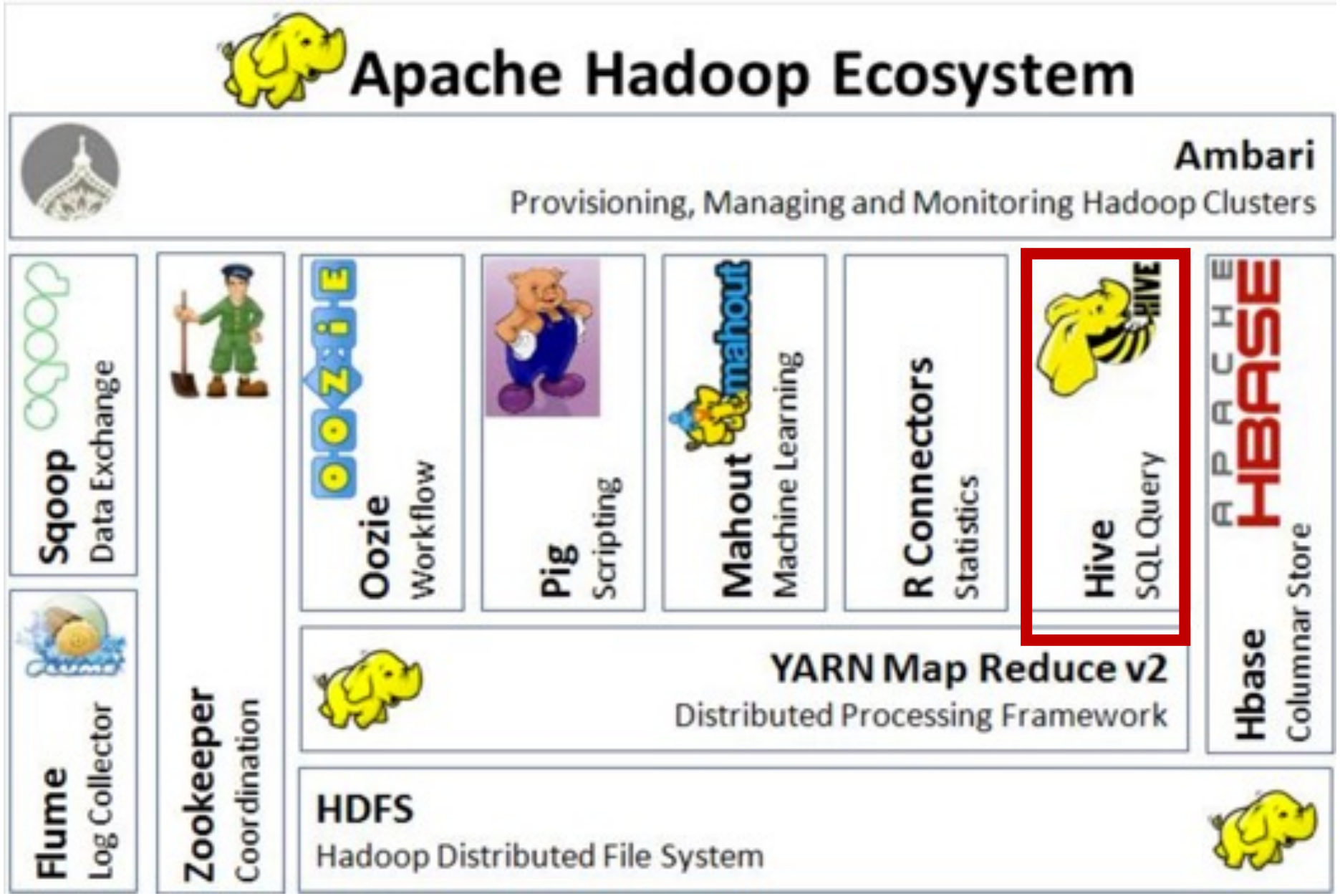



Apache Hadoop Ecosystem
Ambari
Provisioning, Managing and Monitoring Hadoop Clusters
Sqoop
Data Exchange
Flume
Log Collector
Zookeeper
Coordination
Oozie
Workflow
Pig
Scripting
Mahout
Machine Learning
R Connectors
Statistics
Hive
SQL Query
Hbase
Columnar Store
YARN Map Reduce v2
Distributed Processing Framework
HDFS
Hadoop Distributed File System

Apache Pig

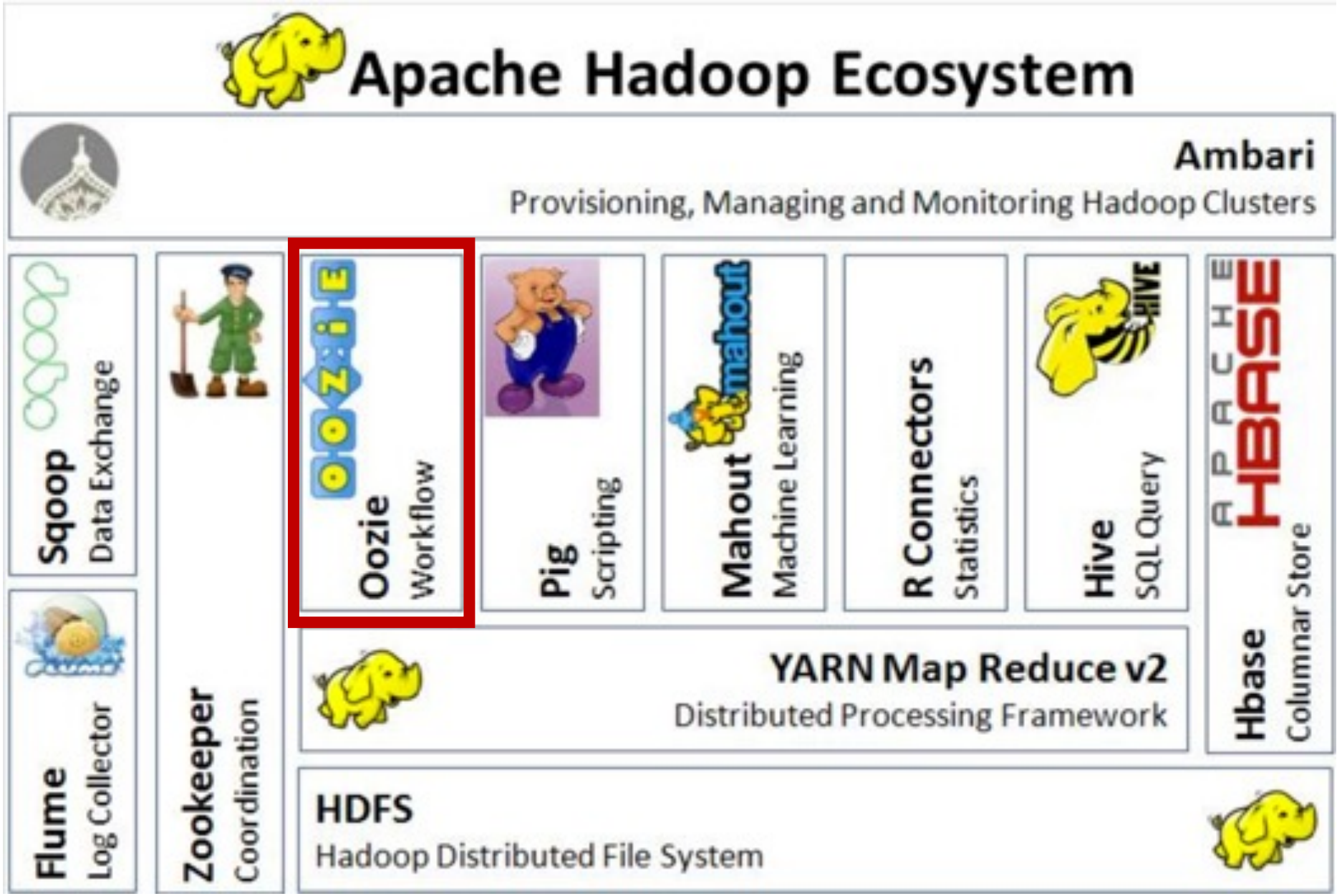
- High level programming on the top of Hadoop MapReduce
- The language: Pig Latin
- Data analysis problems as data flows
- Originally developed at Yahoo 2006




Apache Hadoop Ecosystem
Ambari
Provisioning, Managing and Monitoring Hadoop Clusters
Sqoop
Data Exchange
Flume
Log Collector
Zookeeper
Coordination
Oozie
Workflow
Pig
Scripting
Mahout
Machine Learning
R Connectors
Statistics
Hive
SQL Query
Hbase
Columnar Store
YARN Map Reduce v2
Distributed Processing Framework
HDFS
Hadoop Distributed File System

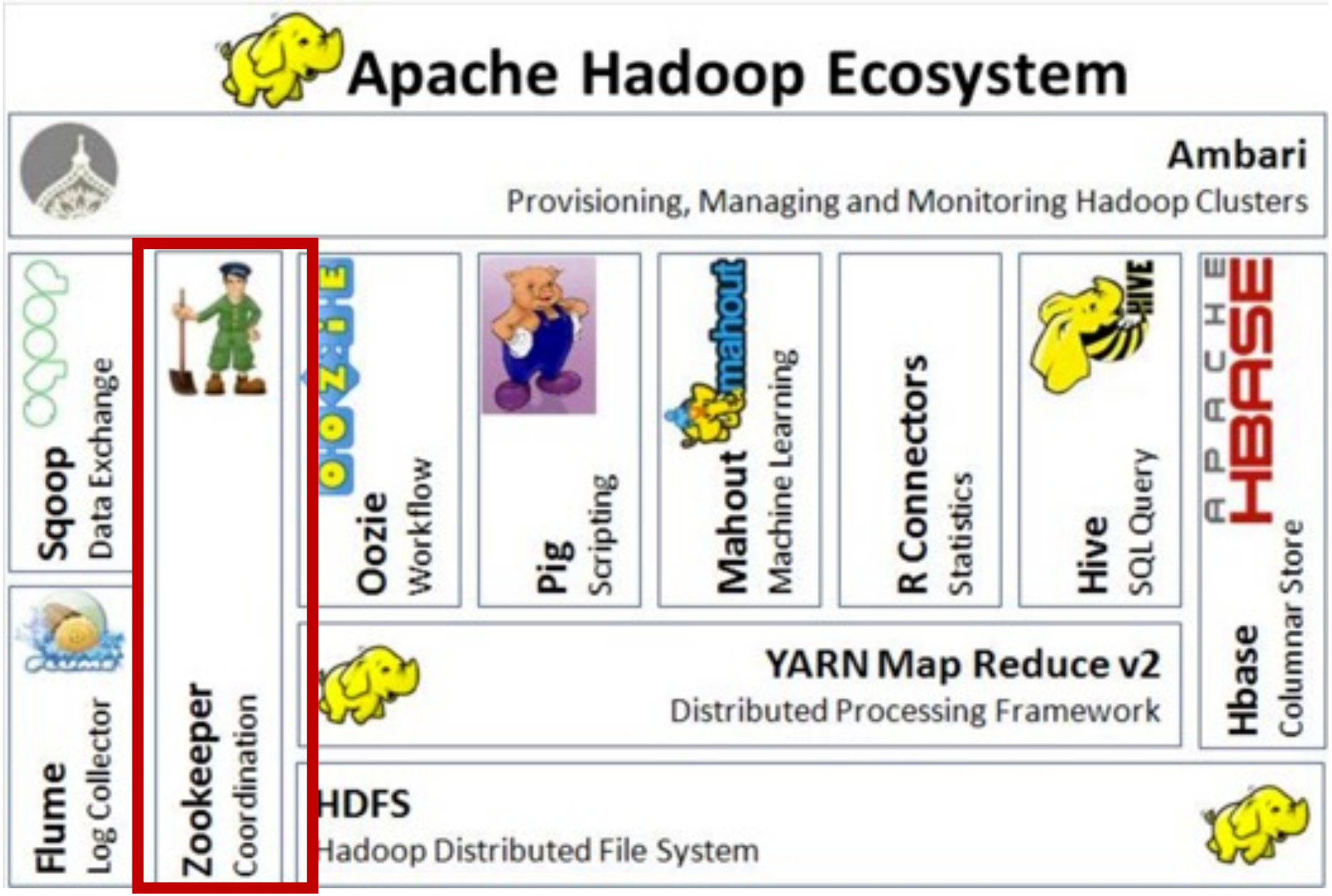
Hive

- Hive is a distributed data management for Hadoop.
- It supports SQL-like query option HiveSQL (HSQL) to access big data.
- It can primarily used for data mining purpose.
- It runs top of Hadoop.


Apache Hadoop Ecosystem
Ambari
Provisioning, Managing and Monitoring Hadoop Clusters
Sqoop
Data Exchange
Flume
Log Collector
Zookeeper
Coordination
Oozie
Workflow
Pig
Scripting
Mahout
Machine Learning
R Connectors
Statistics
Hive
SQL Query
APACHE HBASE
Hbase
Columnar Store
YARN Map Reduce v2
Distributed Processing Framework
HDFS
Hadoop Distributed File System

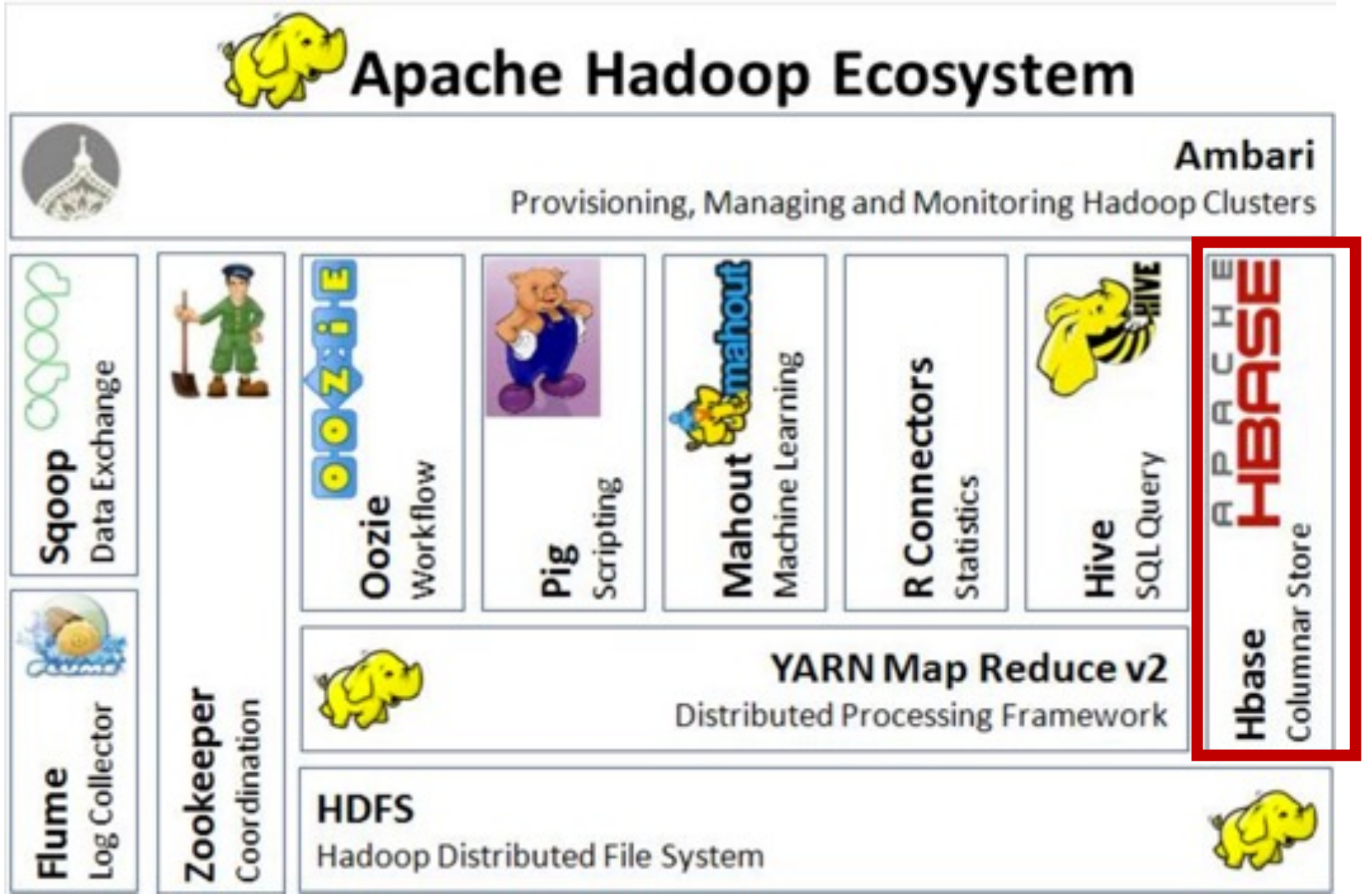
Oozie

- Workflow scheduler system to manage Apache Hadoop jobs
- Oozie Coordinator jobs
- Supports MapReduce, Pig, Hive and Sqoop etc.


Apache Hadoop Ecosystem
Ambari
Provisioning, Managing and Monitoring Hadoop Clusters
Sqoop
Data Exchange
Flume
Log Collector
Zookeeper
Coordination
Oozie
Workflow
Pig
Scripting
Mahout
Machine Learning
R Connectors
Statistics
Hive
SQL Query
Hbase
Columnar Store
YARN Map Reduce v2
Distributed Processing Framework
HDFS
Hadoop Distributed File System

Zookeeper

- Zookeeper is a highly reliable distributed coordination kernel, which can be used for distributed locking, configuration management, leadership election, work queues…..
- Zookeeper is a replicated service that holds the metadata of distributed applications.
- Key attributed of such data
  - Small Size
  - Performance Sensitive
  - Dynamic
  - Critical
- In very simple words, it is a central store of key-value using which distributed systems can coordinate. Since it needs to handle the load, Zookeeper itself run on many machines.


Apache Hadoop Ecosystem
Ambari
Provisioning, Managing and Monitoring Hadoop Clusters
Sqoop
Data Exchange
Flume
Log Collector
Zookeeper
Coordination
Oozie
Workflow
Pig
Scripting
Mahout
Machine Learning
R Connectors
Statistics
Hive
SQL Query
APACHE HBASE
Hbase
Columnar Store
YARN Map Reduce v2
Distributed Processing Framework
HDFS
Hadoop Distributed File System

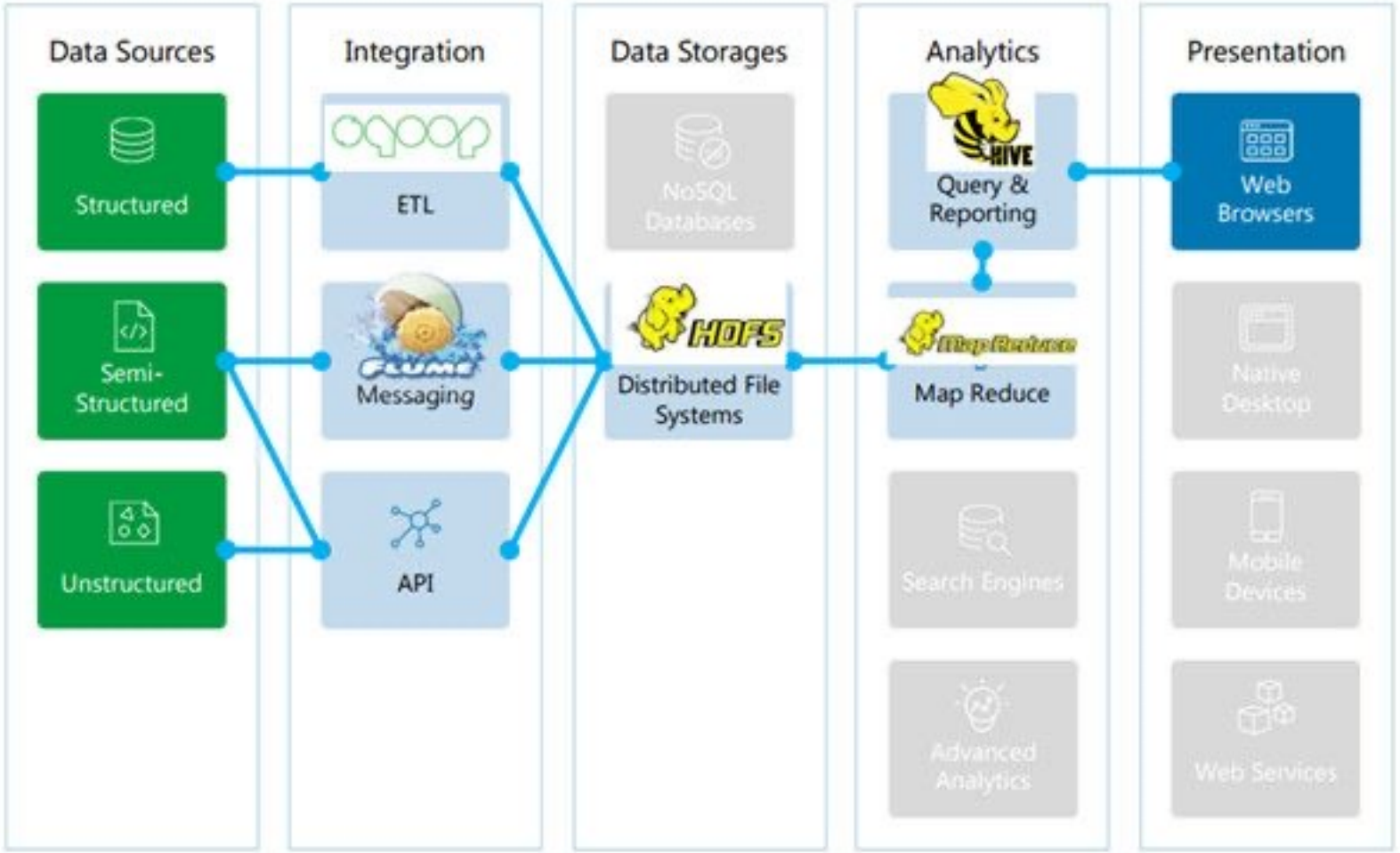
HBase

- HBase is an open source, distributed database, developed by Apache Software foundation.
- Initially, it was Google Big Table, afterwards it was renamed as HBase and it is primarily written in Java.
- HBase can store massive amounts of data from terabytes to petabytes.

Apache Cassandra

- Apache Cassandra is highly scalable, distributed and high-performance NoSQL database. Cassandra is designed to handle a huge amount of data.
- Cassandra handles the huge amount of data with its distributed architecture.
- Data is placed on different machines with more than one replication factor that provides high availability and no single point of failure.

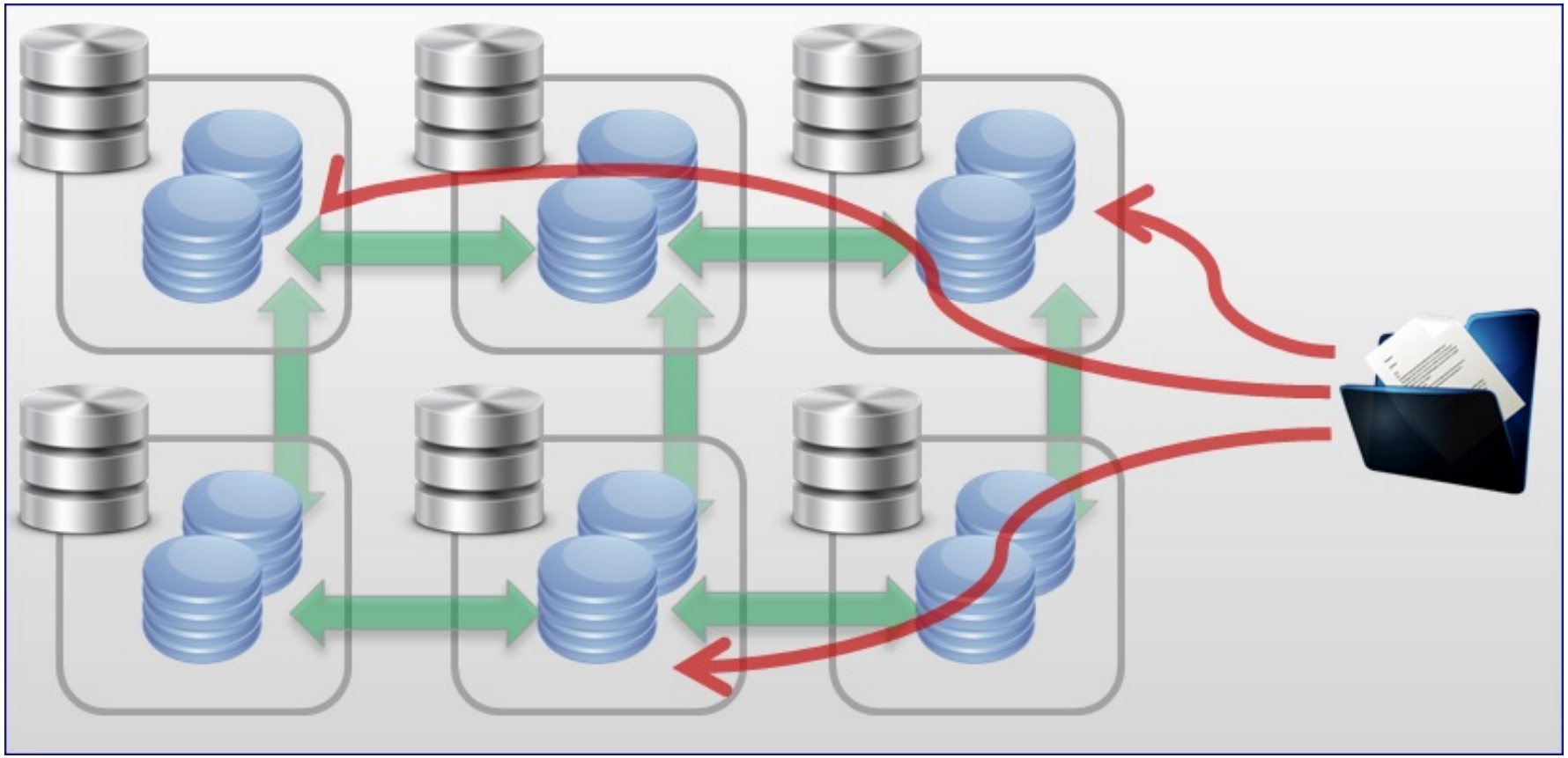
Big Data Architecture using Hadoop



Recap

- In this part of the lecture, we saw a brief overview of following Big Data Enabling Technologies:
  - Apache Hadoop
  - Hadoop Ecosystem
  - HDFS Architecture
  - YARN
  - NoSQL
  - Hive
  - MapReduce
  - Zookeeper
  - Cassandra
  - HBase

Hadoop Design Goals



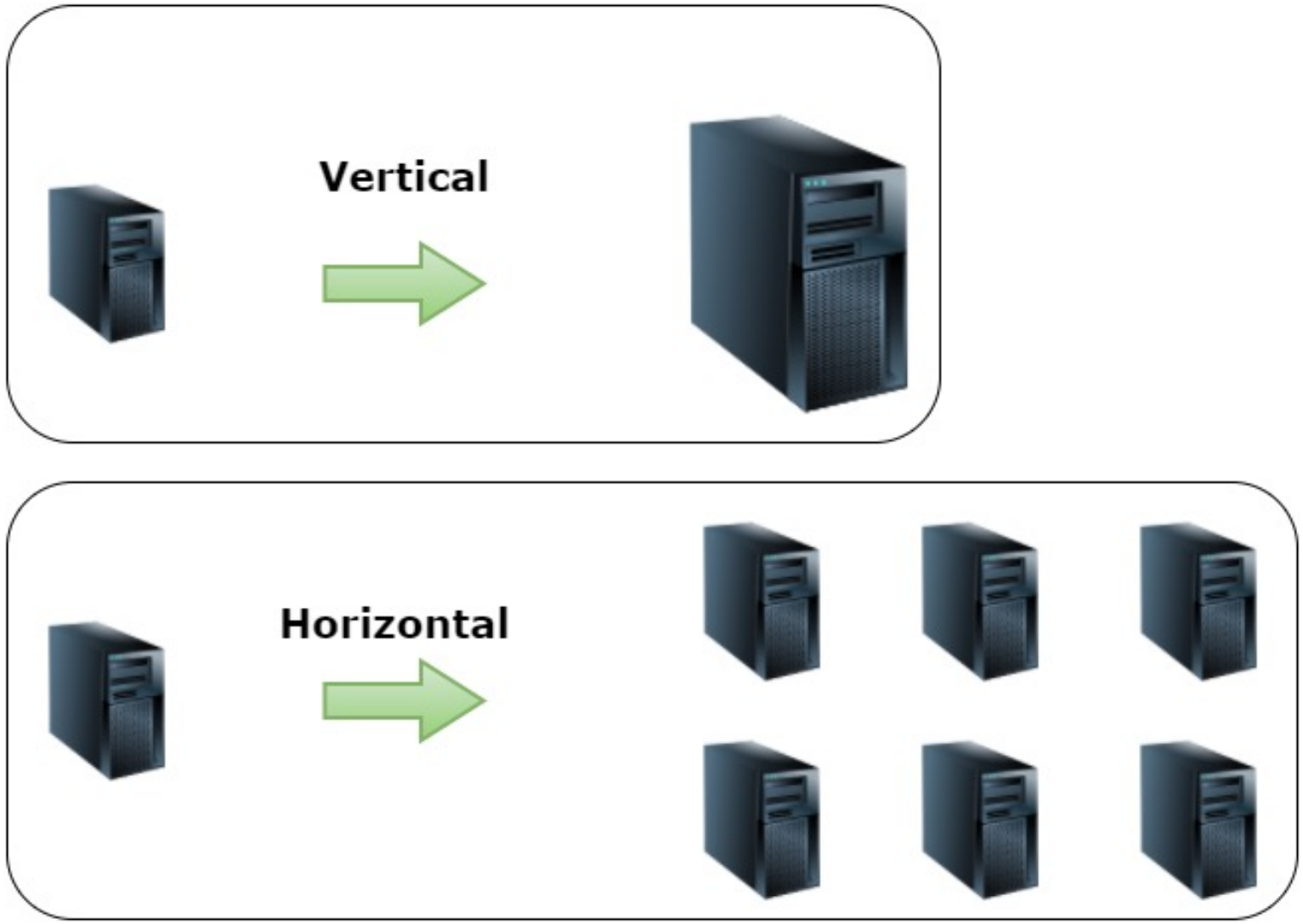
Moving Computation to Data

- Hadoop started as simple batch processing framework.
- The idea behind Hadoop is that instead of moving data to computation, we move computation to data.

Scalability

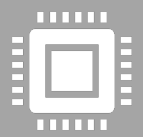
- Scalability is at core of a Hadoop System
- We have cheap computing storage
- We can distribute and scale across very easily in a very cost-effective manner.

Scaling



Reliability

Hardware failures handle automatically



If we think about individual machine or rack of machines, or a large cluster or supercomputer, they all fail at some point of time or some of their components will fail. These failures are so common that we must account for them ahead of time.

And all of these are handled within Hadoop framework system. Apache's Hadoop MapReduce and HDFS components were originally derived from the Google's MapReduce and Google's file system.

New Approach to Data : Keep all the Data

- A new approach is, we can keep all the data we have, and we can take that data and analyze it in new interesting ways.
- We can do something with schema and read style.
- We can allow new analysis. We can bring more data into simple algorithms, which has shown that with more granularity, we can achieve often better results.

Recap

- Design goals for Hadoop:
  - Reliability
  - Scalability
  - Keeping all the data
  - Moving computation to the data

Hadoop Distributed File System

Introduction

- Hadoop provides distributed file system and a framework for the analysis and transformation of very large datasets using MapReduce paradigm.
- An important characteristic of Hadoop is the partitioning of data and computation across many (hundreds and thousands) of hosts and executing these application computations in parallel close to their data.
- A Hadoop cluster scales computation capacity, storage capacity and I/O bandwidth by simply adding commodity servers. Hadoop clusters at Yahoo! spans 25,000 servers, and store 25 petabytes of application data, with the largest cluster being 3500 servers. One hundred other organizations reported using Hadoop worldwide.

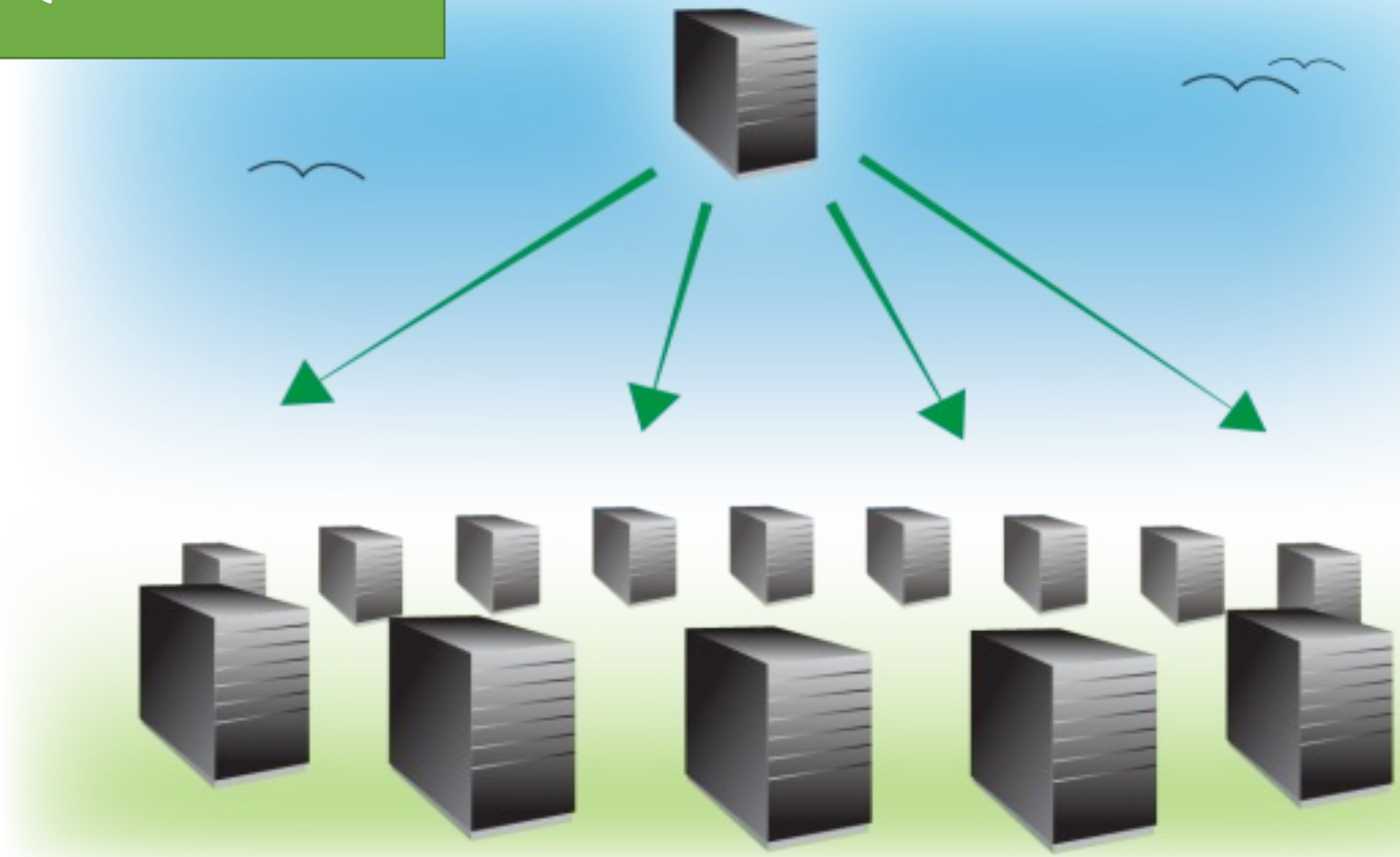
Hadoop Design Concepts

- Scalable distributed filesystem: So, as you add disks you get scalable performance. And as you add more, you are adding lot of disks and that scales out the performance.
- Distributed data on local disks on several nodes
- Low-cost commodity hardware: A lot of performance is achieved out of it because you are aggregating performance of these hardware.

HDFS Design Goals

- Hundreds/Thousands of nodes and disks:
  - It means there's a higher probability of hardware failure. So the design needs to handle node/disk failures.
- Probability across heterogenous software/hardware:
  - Implementation across lots of different kinds of software and hardware.
- Handle large datasets:
  - Need to handle terabyte and petabytes of data.
- Enable processing with high throughput

Question ??



1 Machine
4 I/O Channels
Each channel 100 MB/s
Take 60 minutes to read the data

15 Machines
4 I/O Channels
Each channel 100 MB/s
Take _____ minutes to read the data

- Question: If one machine takes 60 minutes to read 1 TB data, then how much time 15 machines will take to read the same data?

Techniques to meet HDFS Design Goals

- Simplified coherency model:
  - The idea is to write once and read many times. And that simplifies the number of operations required to commit write.
- Data Replication:
  - Helps to handle hardware failures.
  - Try to spread the data, same piece of data on different node.
- Move computation close to data:
  - You are not moving data around. That improves your performance and throughput.
- Relax POSIX requirements to increase throughput:
  - HDFS relaxes a few POSIX requirements to enable streaming access to file system data.

Components of Hadoop

What is Hadoop?

What are the components of Hadoop?

What is HDFS?

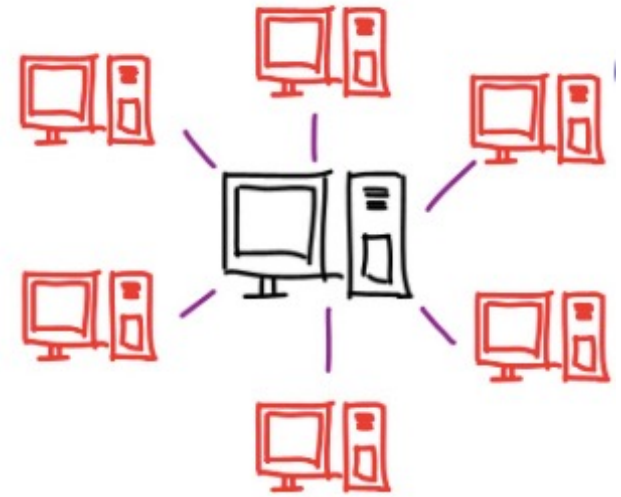
HDFS Architecture

Hadoop MapReduce

Hadoop MapReduce Example

What is Hadoop?

- Hadoop is a framework that allows you to store large volume of data on several node machines



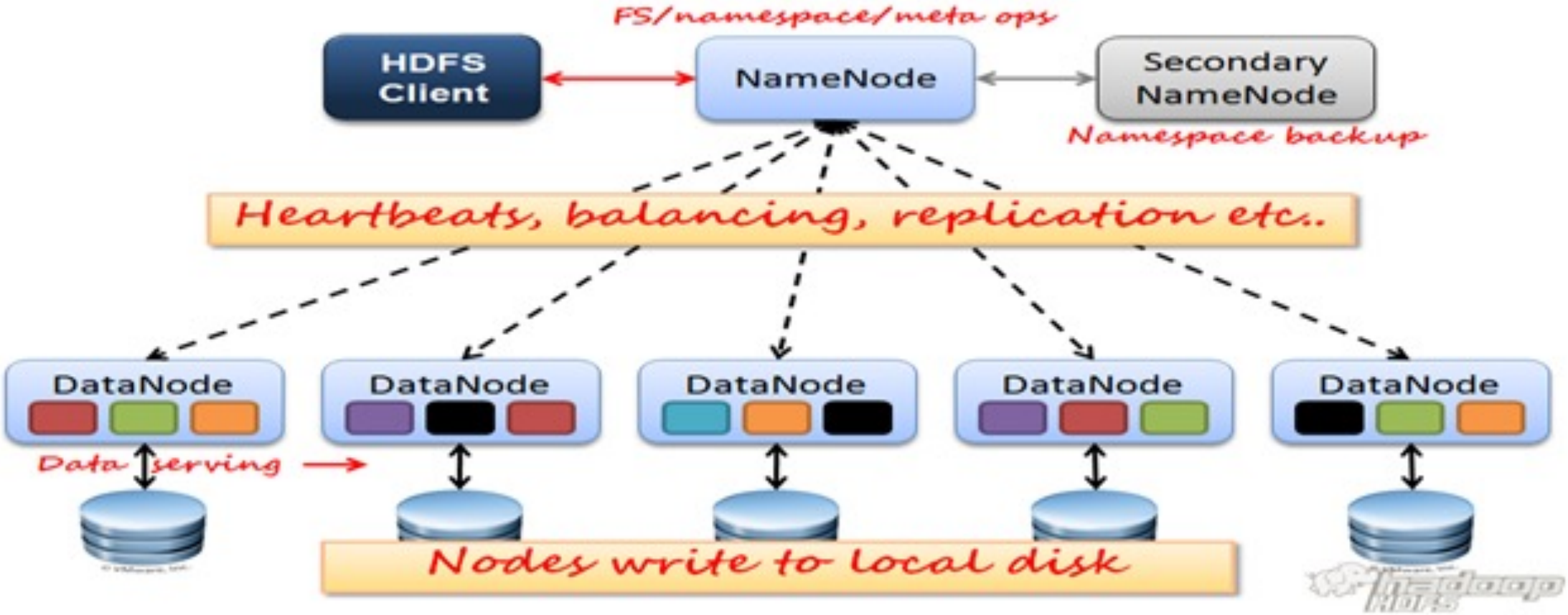
- Hadoop also helps in processing data in a parallel manner

Components of Hadoop

What is HDFS?

- HDFS is the storage layer of Hadoop that stores data in multiple data servers
- Data is divided into multiple blocks
- Stores them over multiple nodes of cluster

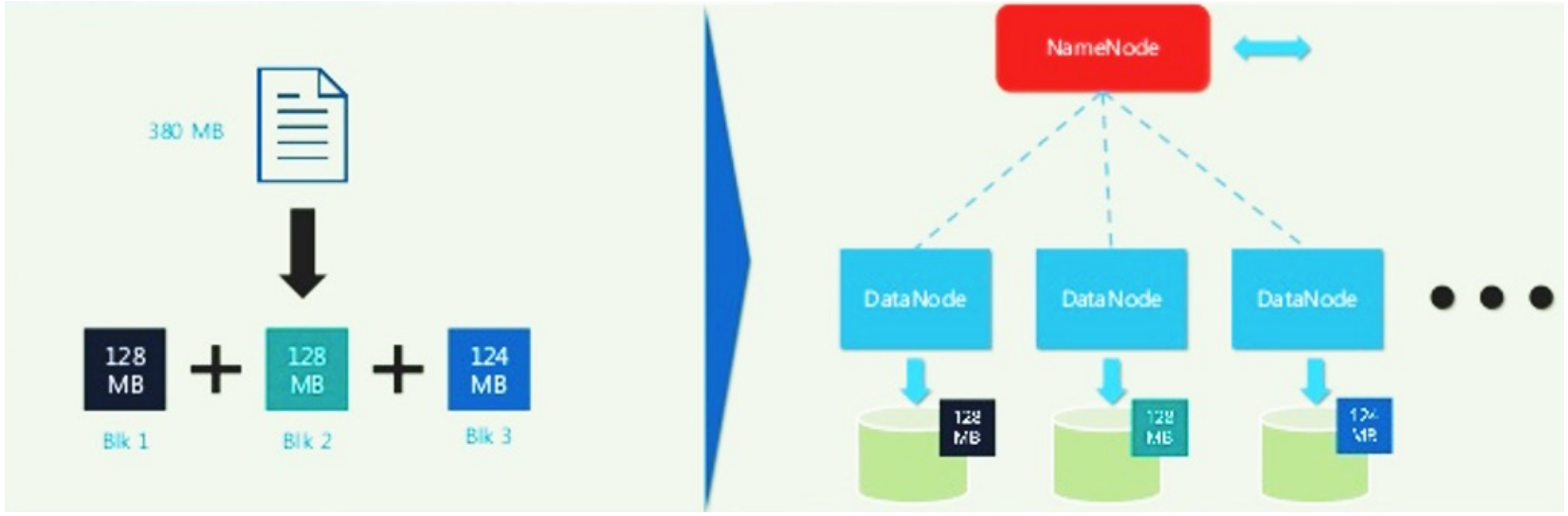
What is HDFS?



- HDFS is the storage layer of Hadoop that stores data in multiple data servers

HDFS Blocks

- Each file on HDFS is stored in blocks
- A typical block size used by HDFS is 128 MB. Thus, an HDFS file is chopped up into 128 MB chunks, and if possible, each chunk will reside on a different DataNode.
- If we have an sample.txt of size 380 MB. Below is the representation how it will be stored.

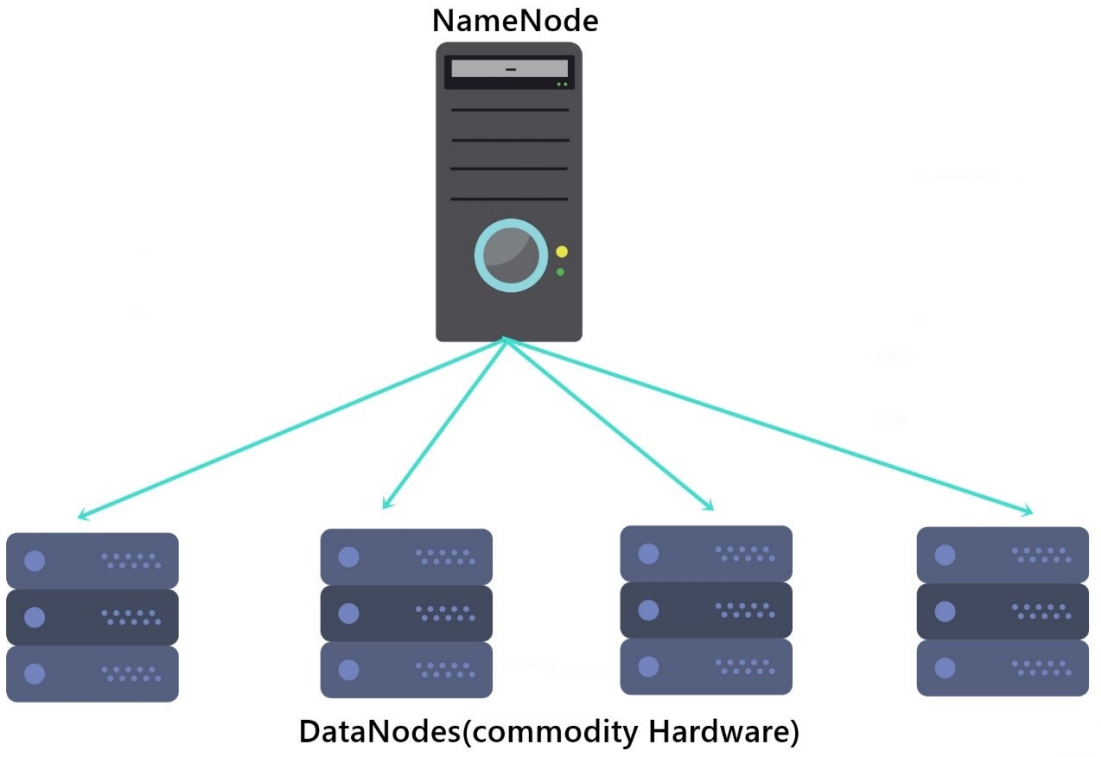


Question?

How many blocks will be created if a file size 500 MB is copied to HDFS?

NameNode and DataNode

- NameNode:
  - Master Daemon
  - Maintains and manages DataNodes
  - Records metadata e.g., location of blocks stored, the size of files, permissions, hierarchy etc.
  - Receives heartbeat and block report from all the DataNodes
- DataNode:
  - Slave Daemons
  - Store actual data
  - Servers read and write requests from the client



Recap

- In this part of the lecture, we have seen an overview of Hadoop and discussed HDFS.

Hadoop MapReduce

Content

- In this part of the lecture, we will discuss the ‘MapReduce paradigm’ and its internal working and implementation overview.
- We will see examples of MapReduce and different applications of MapReduce.

Introduction

- MapReduce is a programming model and an associated implementation for processing and generating large data sets.
- Users specify a **map** function processes a key/value pairs to generate a set of immediate key/value pairs, and a **reduce** function that merges all intermediate values associated with the same intermediate key.
- Many real-world tasks are expressible in this model.

Contd…

- Programs written in this functional style are **automatically parallelized and executed** on a large cluster of commodity machines.
- The **run-time system** takes care of the details of partitioning the input data, scheduling the program's execution across a set of machines, handling machines failures, and managing the required inter-machine communication.
- This **allows programmers to easily utilize the resources** of a large distributed system without any experience with parallel and distributed system.
- A **typical MapReduce computation** processes many terabytes of data on thousands of machines. Hundreds of MapReduce programs have been implemented and more than one thousand MapReduce jobs executed on Google's clusters everyday.

Motivation of MapReduce (Why)

- Large Scale Data Processing
  - Want to use 1000's of CPU
  - But don't want hassle of managing things
- MapReduce Architecture provides
  - Automatic parallelization and distribution
  - Fault tolerance
  - I/O scheduling
  - Monitoring and status updates
- MapReduce Advantages
  - You are not moving the Data
  - You may have many computers counting it in parallel.

What is MapReduce

- Terms are borrowed from functional language (e.g., Lisp)
- Sum of Squares:
  - (map squares ‘(1 2 3 4))
    - Output: (1 4 9 16)
    - [processes each record sequentially and independently]
  - (reduce + ‘(1 4 9 16))
    - (+16(+9(+4 1)))
    - Output: ()
    - [processes set of all records in batches]
- Let’s consider a sample application: Wordcount
  - You have given a huge dataset (e.g., Wikipedia dump or all of Shakespeare’s work) and asked to list the count for each of the words in each of the documents therein

Programming Model

- The computation takes a set of **input key/value pairs** and produces a set of **output key/value pairs**.
- The user of the MapReduce library expresses the computation as two functions:
  - Map
  - Reduce

(i) Map Abstraction

- **Map**, written by the user, takes an input pair and produces a set of **intermediate key/value pairs**.

- The MapReduce library groups together all intermediate values associated with the same **intermediate key 'I'** and passes them to **Reduce function**.

(ii) Reduce Abstraction

- The **reduce** function, also written by the user, accepts an **intermediate key '*I*'** and a set of values for that key.
- It merges these values to form a possibly **smaller set of values**.
- Typically, just zero or one output is produced per Reduce invocation. The **intermediate values** are supplied to the user's reduce function via an **iterator**.
- This allows us to handle lists of values that are too large to fit in memory.

MapReduce Functions for Word Count

- Map (key, value):

```
//key : document name; value: text of document
for  each word w in value:
        (emit w, 1)
```

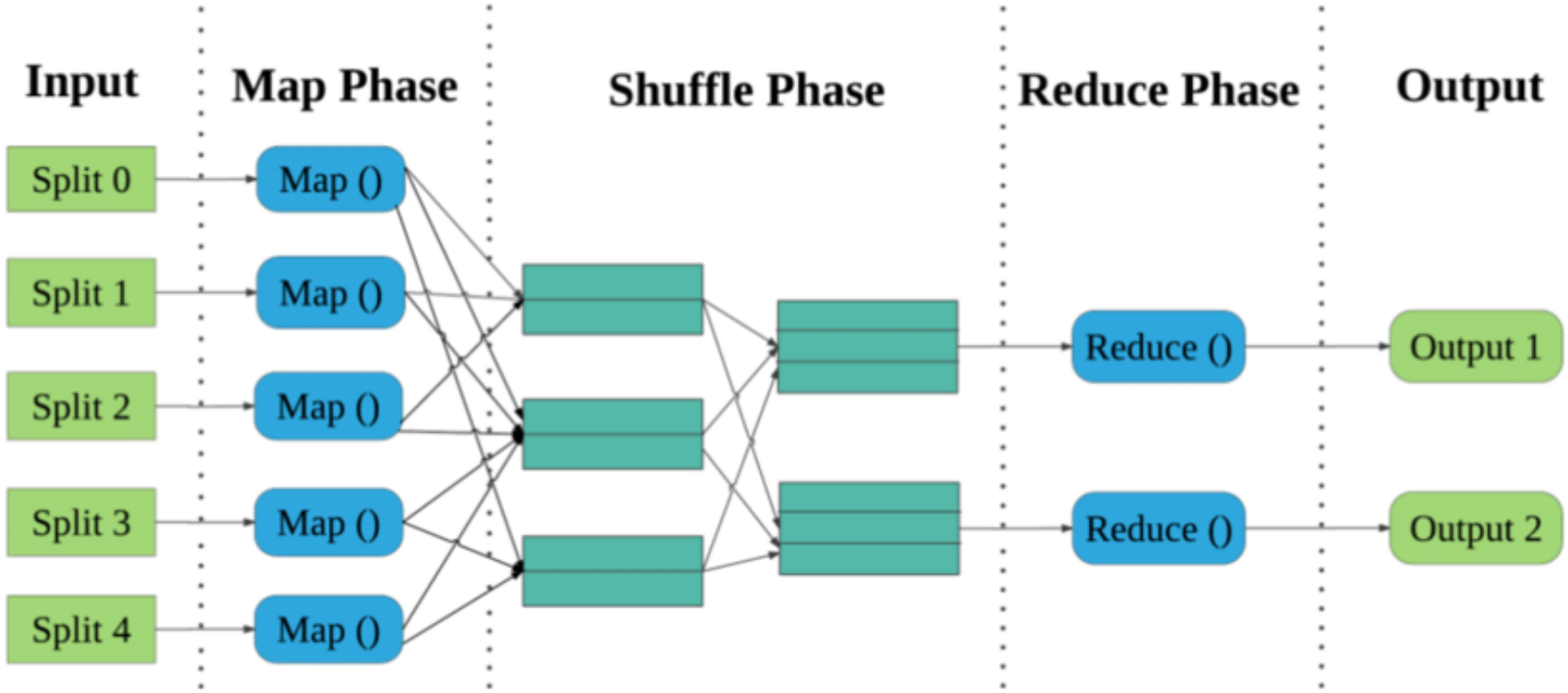
- Reduce (key, values):

```
//key: a word, values: an iterator over counts
        result  = 0
        for each count v in values:
                result +=v
        emit (key, result)
```

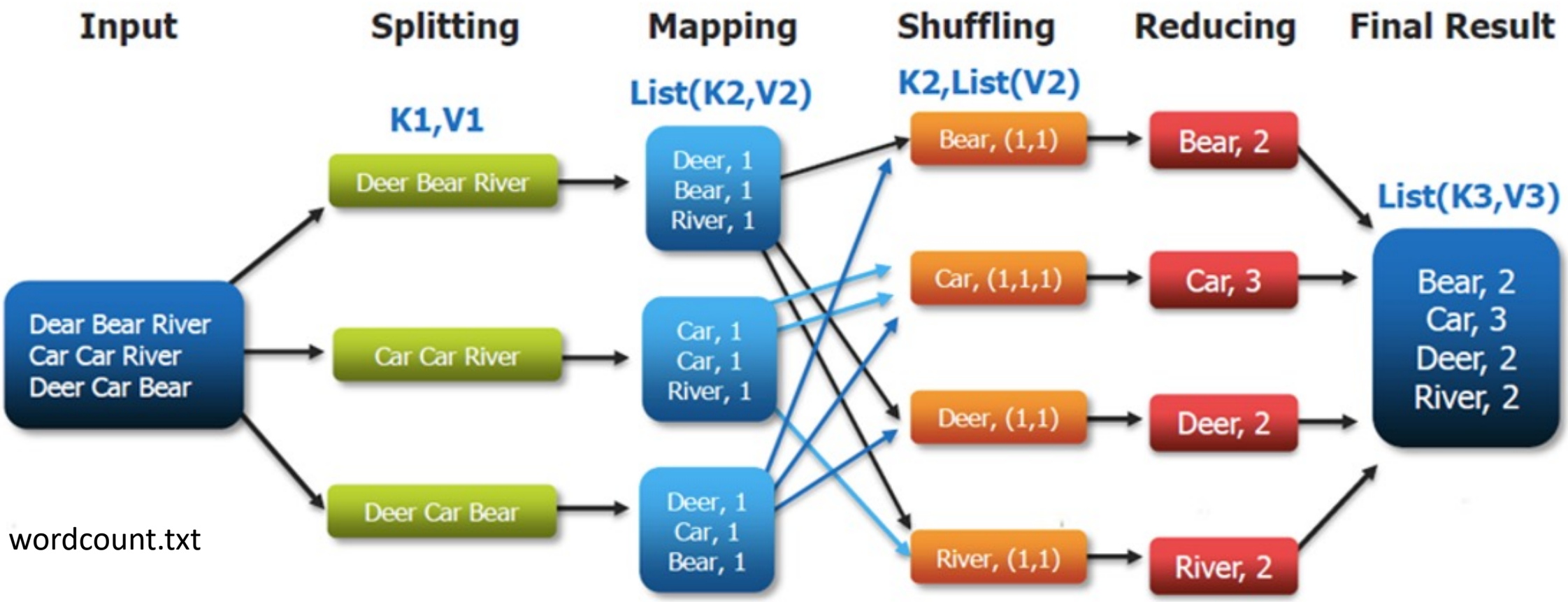
Map-Reduce Functions

- **Input**: a set of key/value pairs
- User supplies two functions:
  - map(k, v) -> list(k1, v1)
  - reduce(k1, list(v1)) -> v2
- (k1/v1) is an intermediate key/value pair
- **Output**: is the set of (k1, v2) pairs

Map Reduce

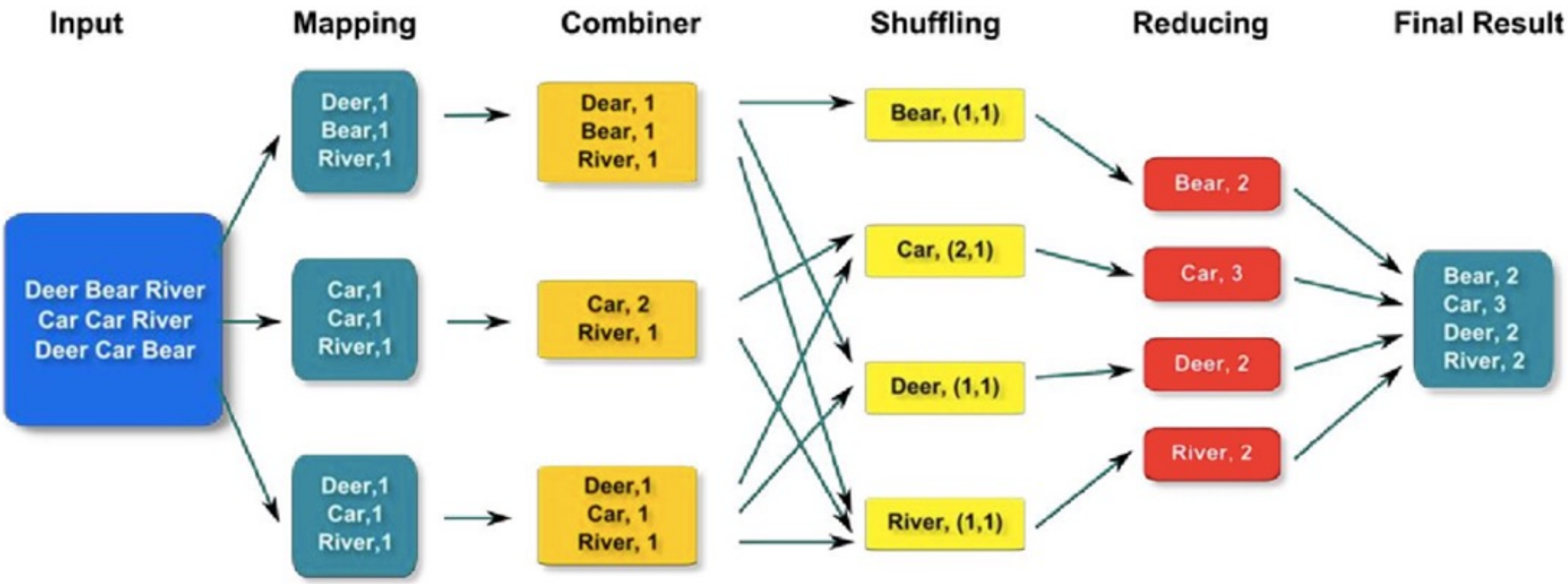

Input
Map Phase
Shuffle Phase
Reduce Phase
Output
Split 0
Split 1
Split 2
Split 3
Split 4
Map ()
Map ()
Map ()
Map ()
Map ()
Reduce ()
Reduce ()
Output 1
Output 2

Map Reduce – Word Count



wordcount.txt

Map Reduce with Combiner



Map Reduce – Word count Program

Word Count - Process

WordCount.txt:

Deer Bear Deer

Car Car River

Deer Car Bear

Mapper's Input:

0, Deer Bear Deer

15, Car Car River

……

Key : LongWrtitable

-byte offset (hexadecimal number)

Value : Text (entire line)

- Mapper's Output:

Deer, 1

Bear, 1

Deer, 1

……

Key : Text

Value : IntWritable (integer)

- Shuffle:

Deer, [1, 1]

Bear, [1]

……

- Reducer's Input

Deer, [1, 1]

Bear, [1]

……

Key : Text

Value : IntWritable

- Reducer's Output

Deer, 2

Bear, 1

……

Key : Text

Value : IntWritable

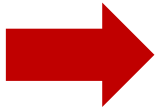
Map Reduce – Import Packages

```
import java.io.IOException;
import java.util.*;

import org.apache.hadoop.fs.Path;
import org.apache.hadoop.conf.*;
import org.apache.hadoop.io.*;

import org.apache.hadoop.mapreduce.*;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.input.TextInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.hadoop.mapreduce.lib.output.TextOutputFormat;
```

All these packages are present in hadoop-common.jar



All these packages are present in hadoop-madreduce-client-core.jar

Map Reduce – Mapper class

```
public static class Map extends
Mapper<LongWritable, Text, Text, IntWritable> {
```

Name of Mapper class which inherits Super Class Mapper

Mapper class takes 4 arguments i.e. Mapper <KEYIN, VALUEIN, KEYOUT, VALUEOUT>

Map Reduce – Reducer class

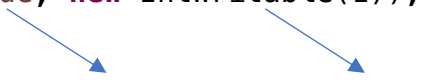
Name of Reducer class which inherits Super Class Reducer

```
public static class Reduce extends Reducer<Text,
IntWritable, Text, IntWritable> {
```

Reducer class takes 4 arguments i.e. Mapper <KEYIN, VALUEIN, KEYOUT, VALUEOUT>

Map Reduce – Mapper

```
public class WordCount {
    public static class Map extends Mapper<LongWritable, Text, Text, IntWritable> {
        public void map(LongWritable key, Text value, Context context) throws IOException, InterruptedException {
              // Deer Bear Deer
              // key : byte offset –
              // value : Deer Bear Deer
            String line = value.toString();
            StringTokenizer tokenizer = new StringTokenizer(line);
            while (tokenizer.hasMoreTokens()) {
                value.set(tokenizer.nextToken());
                // Deer
                // Bear
                // Deer
                context.write(value, new IntWritable(1));
                // Deer, 1
                // Bear, 1
                // Deer,1
            }
        }
    }
```



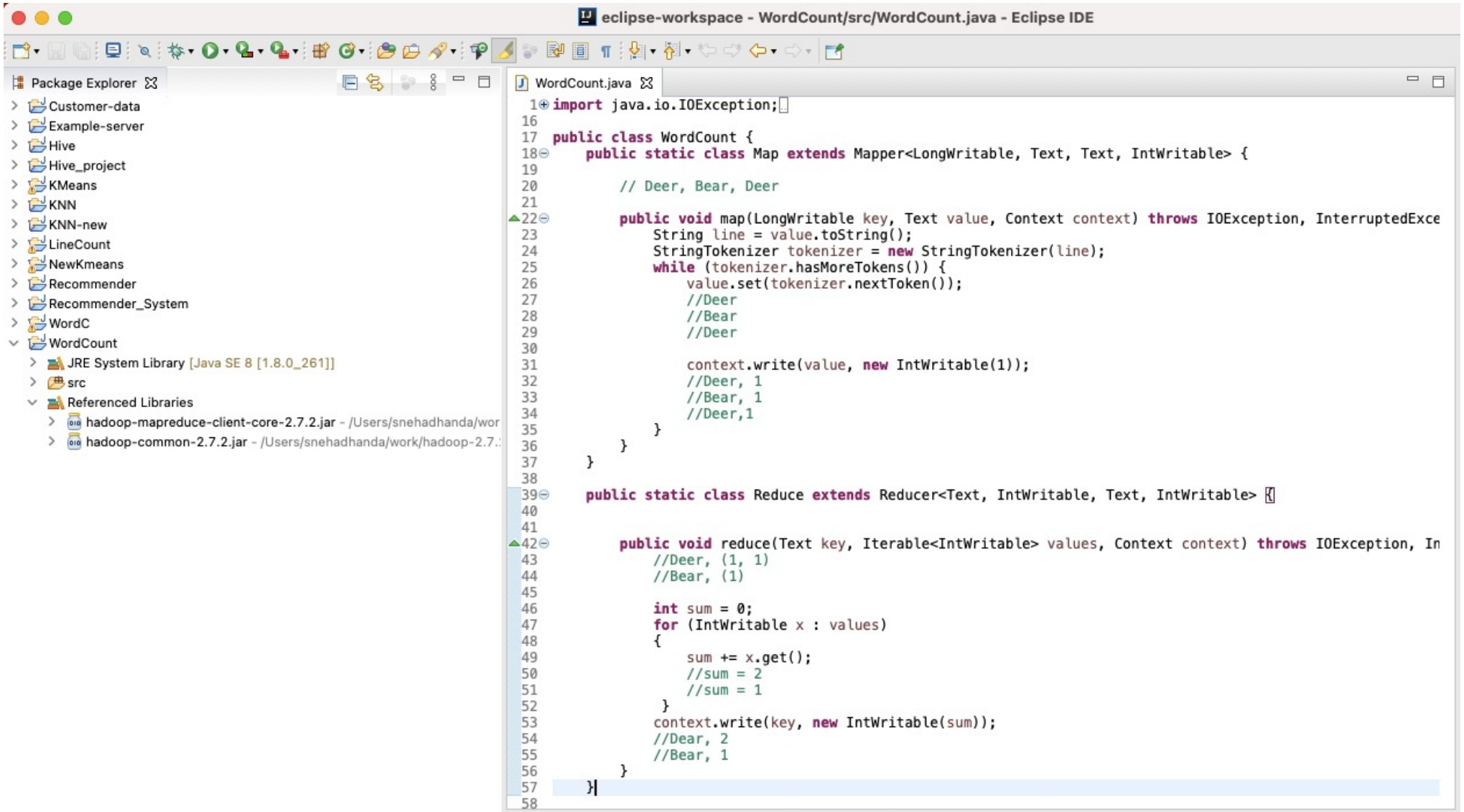
Map Reduce – Reducer

```
public static class Reduce extends Reducer<Text, IntWritable, Text, IntWritable> {
        public void reduce(Text key, Iterable<IntWritable> values, Context context) throws
IOException, InterruptedException {
        //Deer, (1, 1)
        //Bear, (1)
        int sum = 0;
            for (IntWritable x : values)
            {
                sum += x.get();
                //sum = 2
                //sum = 1
             }
            context.write(key, new IntWritable(sum));
            //Dear, 2
            //Bear, 1
    }
    }
```

Map Reduce – Main

```
public static void main(String[] args) throws Exception {
        Configuration conf = new Configuration();
        Job job = Job.getInstance(conf, "word count");
        job.setJarByClass(WordCount.class);
        job.setMapperClass(Map.class);
        job.setReducerClass(Reduce.class);
       //job.setCombinerClass(Reduce.class);
        job.setOutputKeyClass(Text.class);
        job.setOutputValueClass(IntWritable.class);
        job.setInputFormatClass(TextInputFormat.class);
        job.setOutputFormatClass(TextOutputFormat.class);
        FileInputFormat .setInputPaths(job, new Path(args[0]));
        FileOutputFormat.setOutputPath(job, new Path(args[1]));
        // hadoop jar Wordcount.jar /input.txt /output
        // hadoop jar Wordcount.jar WordCount /input.txt /output
        System.exit(job.waitForCompletion(true) ? 0 : 1);
    }
}
```

WordCount Program - Eclipse



Hadoop Commands

Basic File Command in HDFS

- hadoop fs –cmd <args>
  - hdfs dfs
- Adding files
  - hdfs dfs -mkdir /path/to/directory
  - hdfs dfs -put /path/to/directory
- Retrieving files
  - hdfs dfs -get
- Open a file
  - hdfs dfs -cat /path/to/directory
- Deleting files
  - hdfs dfs -rm -r /path/to/directory
- List files
  - hdfs dfs -ls /
- Help
  - hdfs dfs -help

Hadoop Commands

Hadoop – Start Command

```
snehadhanda@Snehas-MacBook-Air ~ % start-dfs.sh && start-yarn.sh
                  WARN util.NativeCodeLoader: Unable to load native-hadoop library for y
our platform... using builtin-java classes where applicable
Starting namenodes on [localhost]
localhost: starting namenode, logging to /Users/snehadhanda/work/hadoop-2.7.2/logs/hadoo
p-snehadhanda-namenode-Snehas-MacBook-Air.local.out
localhost: starting datanode, logging to /Users/snehadhanda/work/hadoop-2.7.2/logs/hadoo
p-snehadhanda-datanode-Snehas-MacBook-Air.local.out
Starting secondary namenodes [0.0.0.0]
0.0.0.0: starting secondarynamenode, logging to /Users/snehadhanda/work/hadoop-2.7.2/log
s/hadoop-snehadhanda-secondarynamenode-Snehas-MacBook-Air.local.out
                  WARN util.NativeCodeLoader: Unable to load native-hadoop library for y
our platform... using builtin-java classes where applicable
starting yarn daemons
resourcemanager running as process 2254. Stop it first.
localhost: nodemanager running as process 2340. Stop it first.
snehadhanda@Snehas-MacBook-Air ~ %
```

Hadoop – List Daemons

```
snehadhanda@Snehas-MacBook-Air ~ % jps
9251 DataNode
2340 NodeManager
10597 Jps
9354 SecondaryNameNode
6492
9166 NameNode
2254 ResourceManager
snehadhanda@Snehas-MacBook-Air ~ %
```

Hadoop – File System Health Check Command

```
snehadhanda@Snehas-MacBook-Air ~ % hadoop fsck /
DEPRECATED: Use of this script to execute hdfs command is deprecated.
Instead use the hdfs command for it.

.............Status: HEALTHY
 Total size:    40379764 B
 Total dirs:    8
 Total files:   14
 Total symlinks:                0
 Total blocks (validated):      9 (avg. block size 4486640 B)
 Minimally replicated blocks:   9 (100.0 %)
 Over-replicated blocks:        0 (0.0 %)
 Under-replicated blocks:       0 (0.0 %)
 Mis-replicated blocks:         0 (0.0 %)
 Default replication factor:    1
 Average block replication:     1.0
 Corrupt blocks:                0
 Missing replicas:              0 (0.0 %)
 Number of data-nodes:          1
 Number of racks:               1


The filesystem under path '/' is HEALTHY
snehadhanda@Snehas-MacBook-Air ~ %
```

Hadoop – List Files Command

```
snehadhanda@Snehas-MacBook-Air ~ % hdfs dfs -ls /
21/01/06 13:53:29 WARN util.NativeCodeLoader: Unable to load native-hadoop library for y
our platform... using builtin-java classes where applicable
Found 1 items
drwxr-xr-x   - snehadhanda supergroup          0 2021-01-06 12:14 /Dataset
snehadhanda@Snehas-MacBook-Air ~ % hdfs dfs -ls /Dataset
21/01/06 13:53:41 WARN util.NativeCodeLoader: Unable to load native-hadoop library for y
our platform... using builtin-java classes where applicable
Found 3 items
-rw-r--r--   1 snehadhanda supergroup   36938010 2021-01-05 17:29 /Dataset/points.txt
-rw-r--r--   1 snehadhanda supergroup         43 2021-01-06 12:14 /Dataset/wc.txt
-rw-r--r--   1 snehadhanda supergroup         63 2021-01-05 17:25 /Dataset/word.txt
snehadhanda@Snehas-MacBook-Air ~ %
```

Hadoop – Commands

```
snehadhanda@Snehas-MacBook-Air ~ % hdfs dfs -mkdir /Demo

21/01/06 13:57:12 WARN util.NativeCodeLoader: Unable to load native-hadoop library for y
our platform... using builtin-java classes where applicable
snehadhanda@Snehas-MacBook-Air ~ % hdfs dfs -put /Users/snehadhanda/work/shakespeare.txt
 /Demo
21/01/06 13:58:05 WARN util.NativeCodeLoader: Unable to load native-hadoop library for y
our platform... using builtin-java classes where applicable
snehadhanda@Snehas-MacBook-Air ~ % hdfs dfs -ls /Demo
21/01/06 13:58:16 WARN util.NativeCodeLoader: Unable to load native-hadoop library for y
our platform... using builtin-java classes where applicable
Found 1 items
-rw-r--r--   1 snehadhanda supergroup    2555806 2021-01-06 13:58 /Demo/shakespeare.txt
snehadhanda@Snehas-MacBook-Air ~ %
```

Common Error

```
snehadhanda@Snehas-MacBook-Air ~ % hdfs dfs -ls /
zsh: command not found: hdfs
snehadhanda@Snehas-MacBook-Air ~ % source .bash_profile
snehadhanda@Snehas-MacBook-Air ~ % hdfs dfs -ls /
21/01/06 14:01:10 WARN util.NativeCodeLoader: Unable to load native-hadoop libra
ry for your platform... using builtin-java classes where applicable
Found 2 items
drwxr-xr-x   - snehadhanda supergroup          0 2021-01-06 12:14 /Dataset
drwxr-xr-x   - snehadhanda supergroup          0 2021-01-06 13:58 /Demo
snehadhanda@Snehas-MacBook-Air ~ %
```

Run - Map Reduce Jar File

```
snehadhanda@Snehas-MacBook-Air ~ % source .bash_profile
snehadhanda@Snehas-MacBook-Air ~ % hadoop jar /Users/snehadhanda/work/WordCount.
jar WordCount /Dataset/wc.txt /mroutput2

21/01/06 14:03:09 WARN util.NativeCodeLoader: Unable to load native-hadoop libra
ry for your platform... using builtin-java classes where applicable
21/01/06 14:03:10 INFO Configuration.deprecation: session.id is deprecated. Inst
ead, use dfs.metrics.session-id
21/01/06 14:03:10 INFO jvm.JvmMetrics: Initializing JVM Metrics with processName
=JobTracker, sessionId=
21/01/06 14:03:10 WARN mapreduce.JobResourceUploader: Hadoop command-line option
 parsing not performed. Implement the Tool interface and execute your applicatio
n with ToolRunner to remedy this.
21/01/06 14:03:10 INFO input.FileInputFormat: Total input paths to process : 1
21/01/06 14:03:10 INFO mapreduce.JobSubmitter: number of splits:1
21/01/06 14:03:10 INFO mapreduce.JobSubmitter: Submitting tokens for job: job_lo
cal897821535_0001
21/01/06 14:03:11 INFO mapreduce.Job: The url to track the job: http://localhost
:8080/
21/01/06 14:03:11 INFO mapreduce.Job: Running job: job_local897821535_0001
21/01/06 14:03:11 INFO mapred.LocalJobRunner: OutputCommitter set in config null
21/01/06 14:03:11 INFO output.FileOutputCommitter: File Output Committer Algorit
hm version is 1
```

Map Reduce - Result

```
        Map-Reduce Framework
                Map input records=3
                Map output records=9
                Map output bytes=80
                Map output materialized bytes=104
                Input split bytes=101
                Combine input records=0
                Combine output records=0
                Reduce input groups=4
                Reduce shuffle bytes=104
                Reduce input records=9
                Reduce output records=4
                Spilled Records=18
                Shuffled Maps =1
                Failed Shuffles=0
                Merged Map outputs=1
                GC time elapsed (ms)=6
                Total committed heap usage (bytes)=507510784
        Shuffle Errors
                BAD_ID=0
                CONNECTION=0
                IO_ERROR=0
                WRONG_LENGTH=0
                WRONG_MAP=0
                WRONG_REDUCE=0
        File Input Format Counters
                Bytes Read=43
        File Output Format Counters
                Bytes Written=28
snehadhanda@Snehas-MacBook-Air ~ %
```

Without Combiner

```
        Map-Reduce Framework
                Map input records=3
                Map output records=9
                Map output bytes=80
                Map output materialized bytes=50
                Input split bytes=101
                Combine input records=9
                Combine output records=4
                Reduce input groups=4
                Reduce shuffle bytes=50
                Reduce input records=4
                Reduce output records=4
                Spilled Records=8
                Shuffled Maps =1
                Failed Shuffles=0
                Merged Map outputs=1
                GC time elapsed (ms)=7
                Total committed heap usage (bytes)=508559360
        Shuffle Errors
                BAD_ID=0
                CONNECTION=0
                IO_ERROR=0
                WRONG_LENGTH=0
                WRONG_MAP=0
                WRONG_REDUCE=0
        File Input Format Counters
                Bytes Read=43
        File Output Format Counters
                Bytes Written=28
snehadhanda@Snehas-MacBook-Air ~ %
```

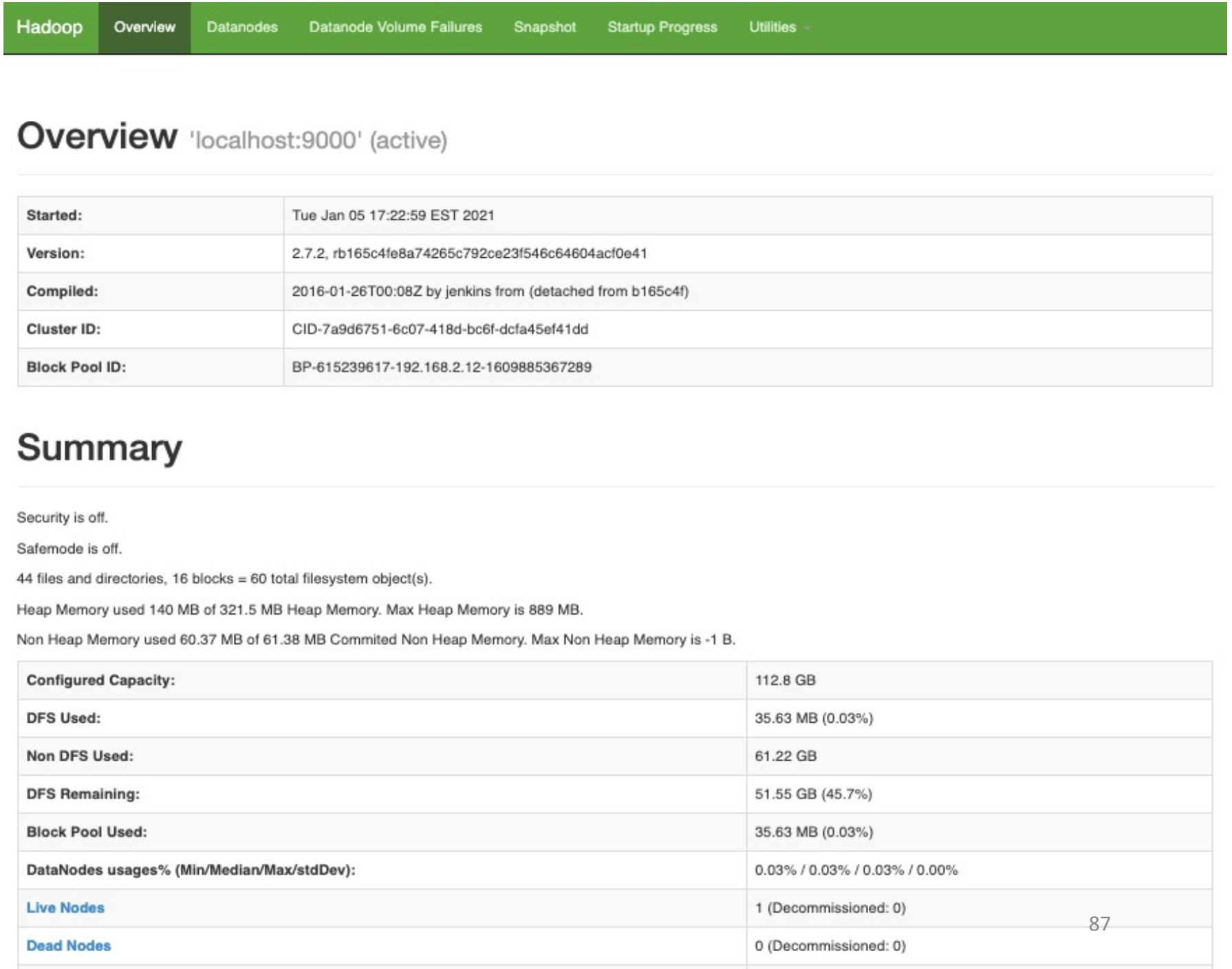
With Combiner

Map Reduce - Result

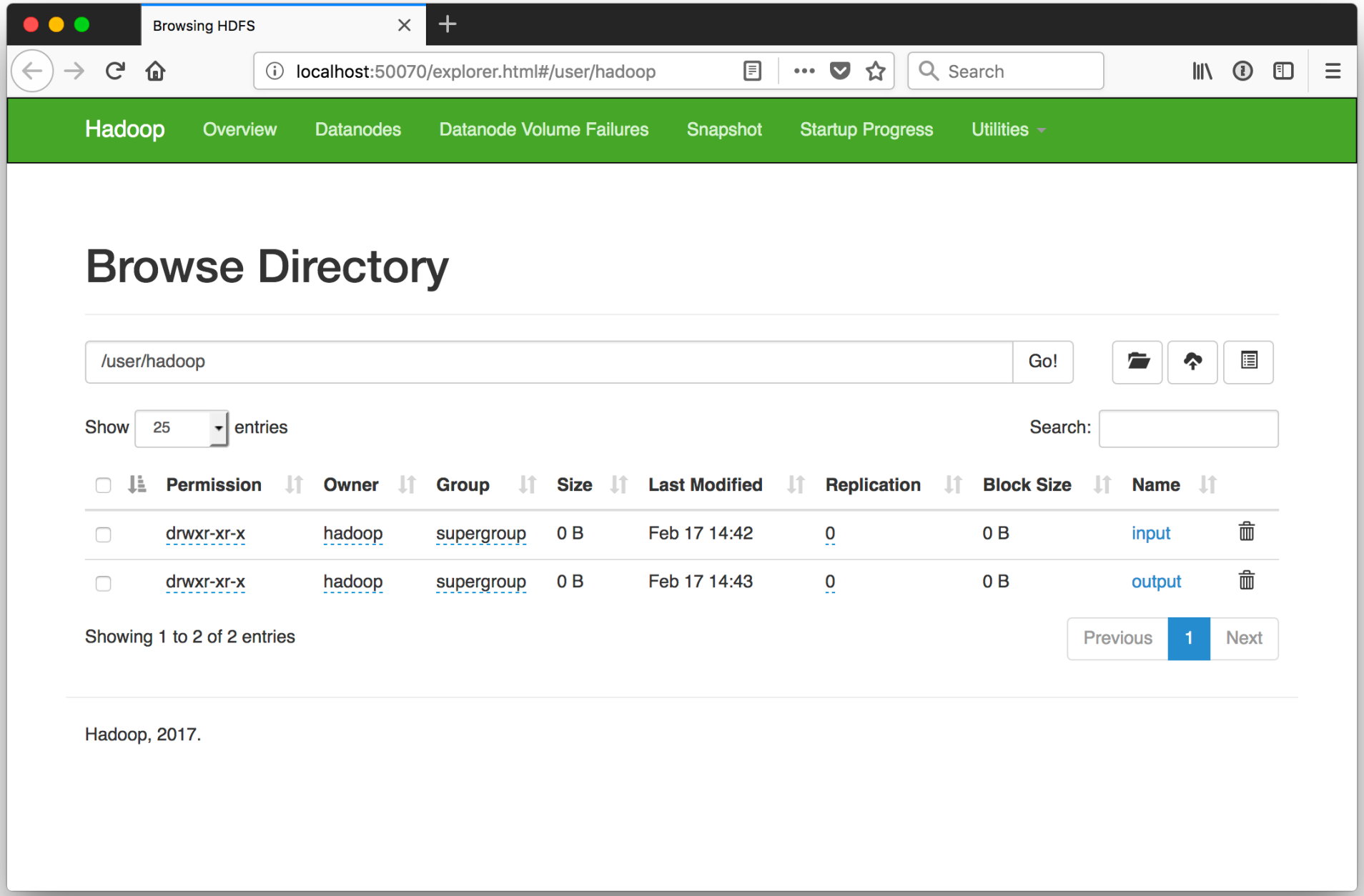
```
snehadhanda@Snehas-MacBook-Air ~ % hdfs dfs -cat /mroutput2/part-r-00000

21/01/10 20:23:04 WARN util.NativeCodeLoader: Unable to load native-hadoop library
for your platform... using builtin-java classes where applicable
Bear    2
Car     3
Deer    2
River   2
snehadhanda@Snehas-MacBook-Air ~ %
```

localhost: 50070



localhost: 50070



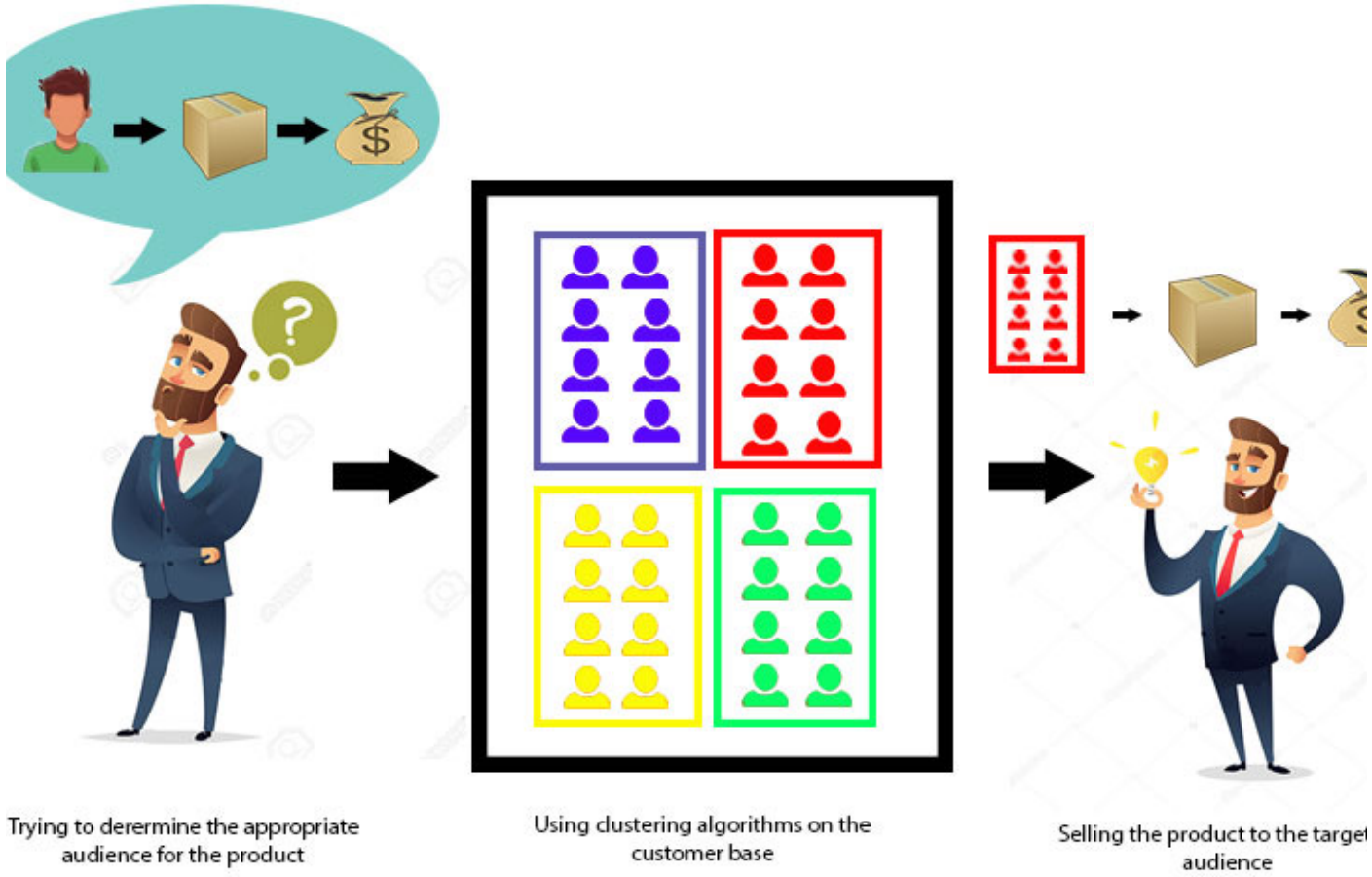
Recap

- In this part of the lecture, we have seen an overview of Hadoop MapReduce, Hadoop commands and Wordcount program using MapReduce for big data analysis.

Machine Learning Algorithm K-means using MapReduce for Big Data Analytics

Content

- In this part of the lecture, we will discuss machine learning classification algorithm k-means using mapreduce for big data analytics.



Cluster Analysis Overview

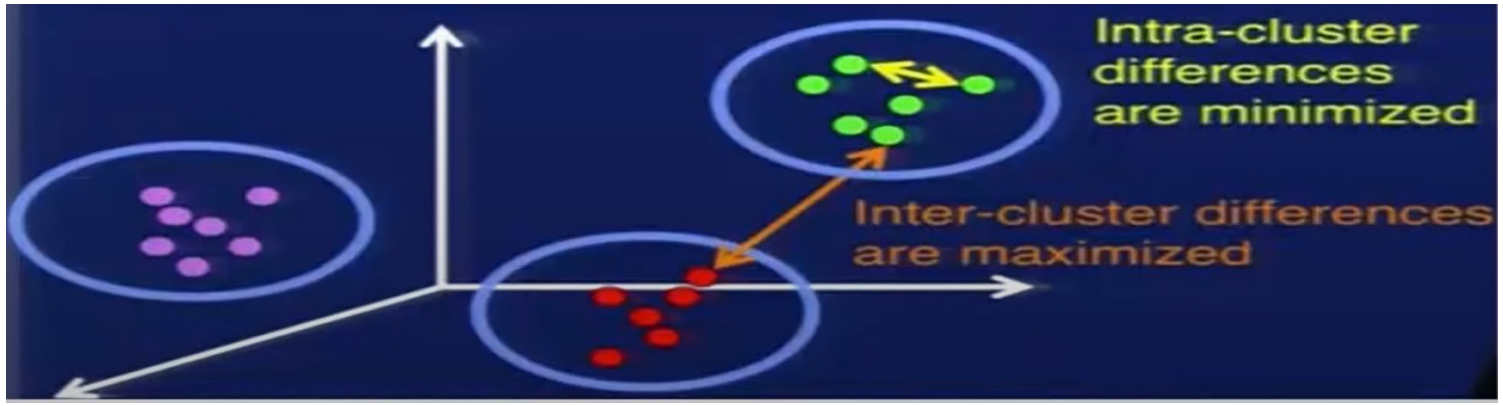
- Goal: Organize similar items into groups
- In cluster analysis, the goal is to organize similar items in given data set into groups or clusters. By segmenting given data into clusters, we can analyse each cluster more carefully.
- Note that cluster analysis is also referred to as clustering.

Applications: Cluster Analysis

- Segment customer base into groups: A very common application of cluster analysis is to divide your customer base into segments based on their purchasing histories. For example, you can segment customers into those who have purchased science fiction books and videos, versus those who tend to buy notification books, versus those who have bought many children's books. This way, you can provide more targeted suggestions to each different group.
- Characterize different weather patterns for a region: Some other examples of cluster analysis are characterizing different weather patterns for a region.
- Group news articles into topics: Grouping the latest news articles into topics to identify the trending topics of the day.
- Discover crime hot spots: Discovering hot spots for different types of crime from police reports in order to provide sufficient police presence for problem areas.

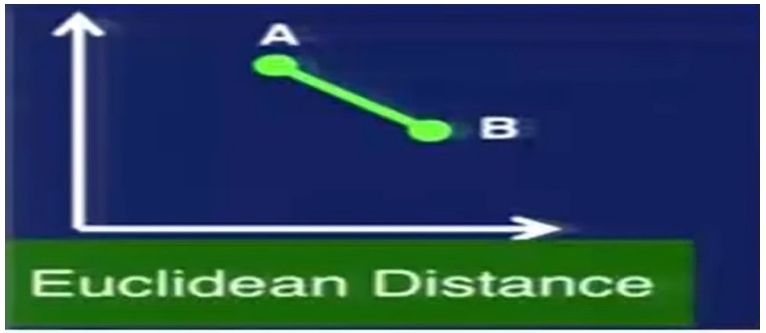
Cluster Analysis

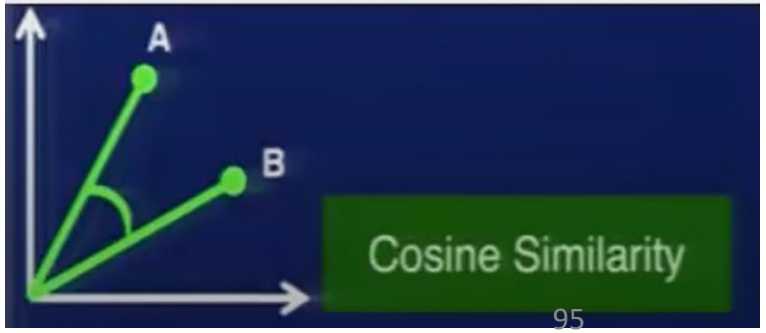
- **Divides data into clusters:** Cluster analysis divides all the samples in a data set into groups. In this diagram, we see red, green and purple data points are clustered together. In which group a sample will be placed is based on some measure of similarity.
- **Similar items are placed in same cluster:** The goal of cluster analysis is to segment data so that differences between samples in the same cluster are minimized as shown by the yellow arrow, and differences between samples of different clusters are maximized, as shown by orange arrow. Visually, we can think of this as getting samples in each cluster to be as close together as possible, and the samples from the different clusters to be as far apart as possible.



Similarity Measures

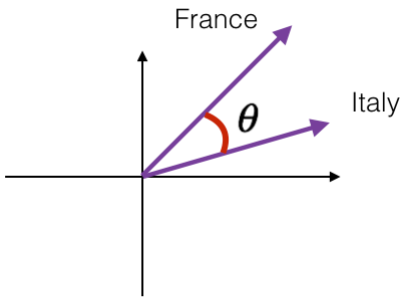
- Cluster analysis requires some sort of metric to measure similarity between two samples. Some common similarity measures are:
- **Euclidean distance**, which is the distance along a straight line between two points, A and B as shown in this plot.
- **Manhattan distance,** which is calculated on a strictly horizontal and vertical path, as shown in the right plot. To go from point A to point B, you can only step along either the x-axis or the y-axis in a two-dimensional case. So, the path to calculate the Manhattan distance consists of segments along the axes instead of along a diagonal path, as with Euclidean distance.
- **Cosine Similarity** measures the cosine of the angle between points A to B, as shown in the bottom plot. Since distance measures such as Euclidean distance are often used to measure similarity between samples in clustering algorithms, note that it may be necessary to normalize the input variables so that no one value dominates the similarity calculation.





Cosine Similarity- Normalize

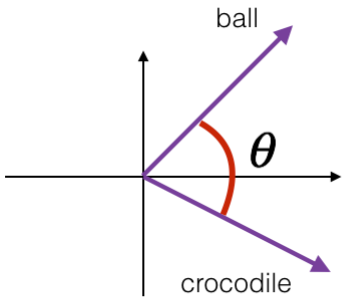
- Not a proper distance metric
- Efficient to compute for sparse vectors



France and Italy are quite similar

θ is close to 0°

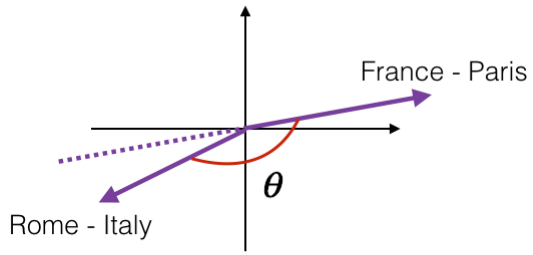
$\cos(\theta) \approx 1$



ball and crocodile are not similar

θ is close to 90°

$\cos(\theta) \approx 0$



the two vectors are similar but opposite
the first one encodes (city - country)
while the second one encodes (country - city)

θ is close to 180°

$\cos(\theta) \approx -1$

$$\text{similarity} = \cos(\theta) = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\|\|\mathbf{B}\|} = \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2}\sqrt{\sum_{i=1}^{n} B_i^2}},$$

Cluster Analysis Key Points

Cluster Analysis is an unsupervised task. This means that there is no target label for any sample in the dataset.

In general, there is no correct clustering results. The best set of clusters is highly dependent on how the resulting clusters will be used.

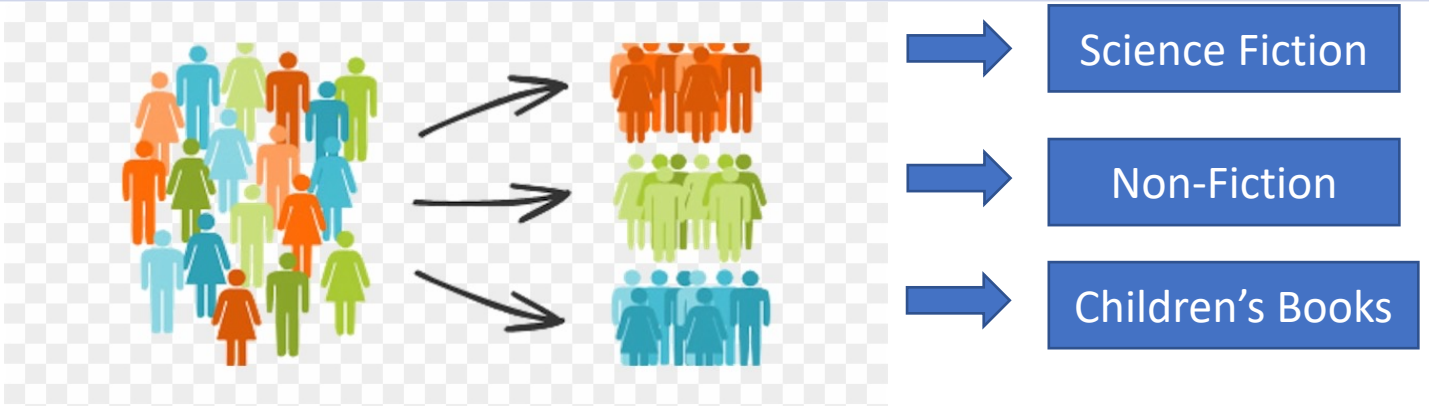
Clusters don't come with labels. You may end up with five different clusters at the end of a cluster analysis process, but you don't know what each cluster represents. Only by analyzing samples in each cluster you can come out with reasonable labels for your clusters. Given all this, it is important to keep in mind that interpretation and analysis of the clusters are required to make sense of and make use of results of cluster analysis.

Interpretation and Analysis required to make sense of clustering results!

Uses of Cluster Results

Data Segmentation

Analysis of each segment can provide insights: There are several ways that the results of cluster analysis can be used. The most obvious is data segmentation and the benefits that come from that. If we segment the customer base into different types of readers, the resulting insights can be used to provide more effective marketing to the different customer groups based on their preferences. For example, analyzing each segment separately can provide valuable insights into each group's likes, dislikes and purchasing behavior, just like we see science fiction, non-fiction and children's books preference here.



Cluster Analysis Summary

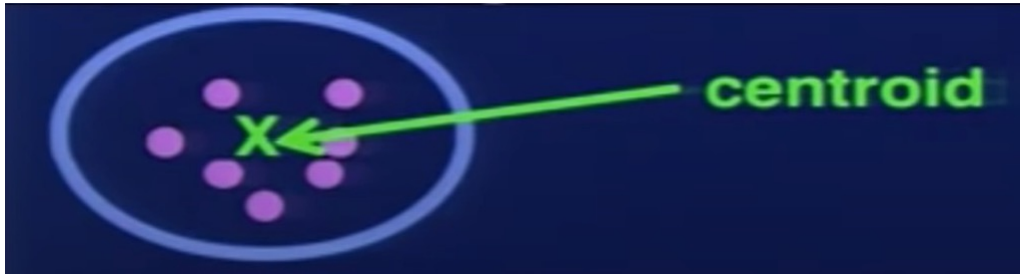
ORGANIZE SIMILAR ITEMS INTO GROUPS

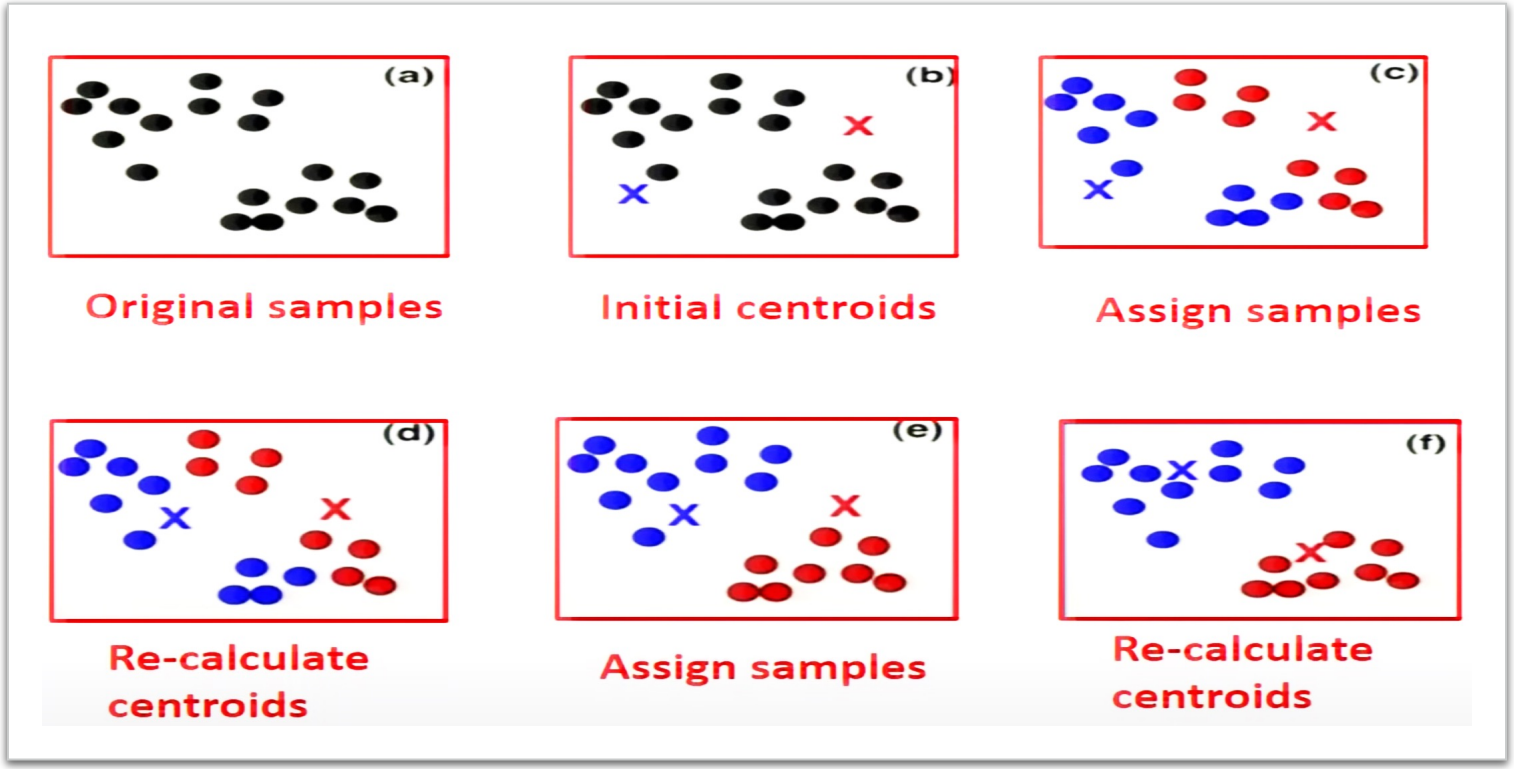
ANALYSING CLUSTERS OFTEN LEADS TO USEFUL INSIGHTS ABOUT DATA

CLUSTERS REQUIRE ANALYSIS INTERPRETATION

K-Means Algorithm

- Select k initial centroid (clusters centers)
- Repeat
  - Assign each sample to closet centroid
  - Calculate the mean of cluster to determine new centroid
- Until some stopping criterion is reached





K-Means Algorithm example

Choosing Initial Centroids

- Issue:
  - Final clusters are sensitive to initial centroids
- Solution:
  - Run K-means multiple times with different random initial centroids and choose best results

Choosing value for k

Approaches: Choosing optimal value for k is always a big question in using k-means. There are several methods to determine the value of k.

Visualization: Visualization techniques can be used to determine the dataset to see if there are natural grouping of the samples. Scatter plots and use of dimensionality reduction are useful here, to visualize the data.

Application-Dependent: A good value of k is application-dependent. So, domain knowledge of the application can drive the selection for the value of k. For example, if you want to cluster the types of products customers are purchasing, a natural choice for k might be the number of product categories that you offer. Or k might be selected to represent the geographical locations of respondents to a survey. In this case, a good value for k would be the number of regions you are interested in analyzing.

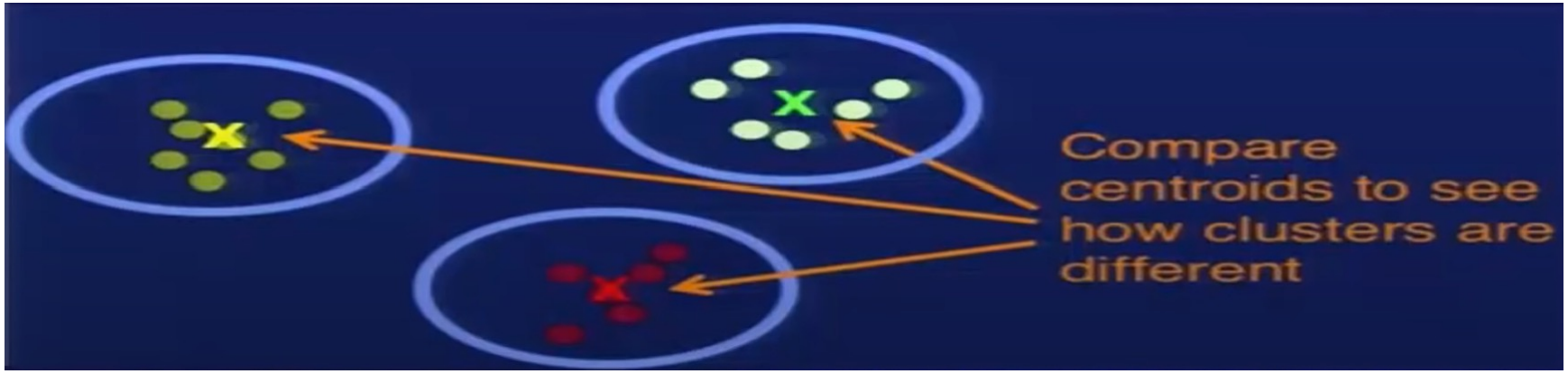
Data Driven: There are also data-driven method for determining the value of k. These methods calculate symmetric for different values of k to determine the best selections of k. One of such method is the elbow method.

Stopping Criteria

- **When to stop iteration?**
  - **No changes to centroids:** How do you know when to stop iterating when using k-means? One obvious stopping criterion is when there are no changes to the centroids. This means that no samples would change cluster assignments. And recalculating the centroids will not result any changes. So additional iterations will not bring about any more changes to the cluster results.
  - **Number of samples changing clusters is below threshold:** The stopping criterion can be relaxed to the second stopping criterion: which is when the number of sample changing clusters is below a certain threshold, say 1% for example. At this point, the clusters are changing by only a few samples, resulting in only minimal changes to the final cluster results. So, the algorithm can be stopped here.

Interpretation of the Results

- **Examine cluster centroids**
  - **How are clusters different?** At the end of k-means we have a set of clusters, each with a centroid. Each centroid is the mean of the samples assigned to the cluster. You can assume that the centroid is a representative sample for that cluster. So, to interpret cluster analysis results, we can examine the cluster centroids. Comparing the values of the variables between the centroids will reveal how different and alike clusters are and provide insights into what each cluster represents. For example, if the value for age is different for different customer clusters, this indicates that the clusters are encoding different customer segments by age, among other variables.



K-Means Summary

Classic algorithm for cluster analysis

Simple to understand, implement and is efficient

Value of K must be specified

Final clusters are sensitive to initial centroids

Parallel K-means using MapReduce

MapReduce 1 iteration of k-means

- **Classify:** Assign observations to closest cluster center

$$z_i \leftarrow \arg\min_j ||\mu_j - \mathbf{x}_i||_2^2$$

Map: For each data point, given ({μj},xi), emit(zi,xi)

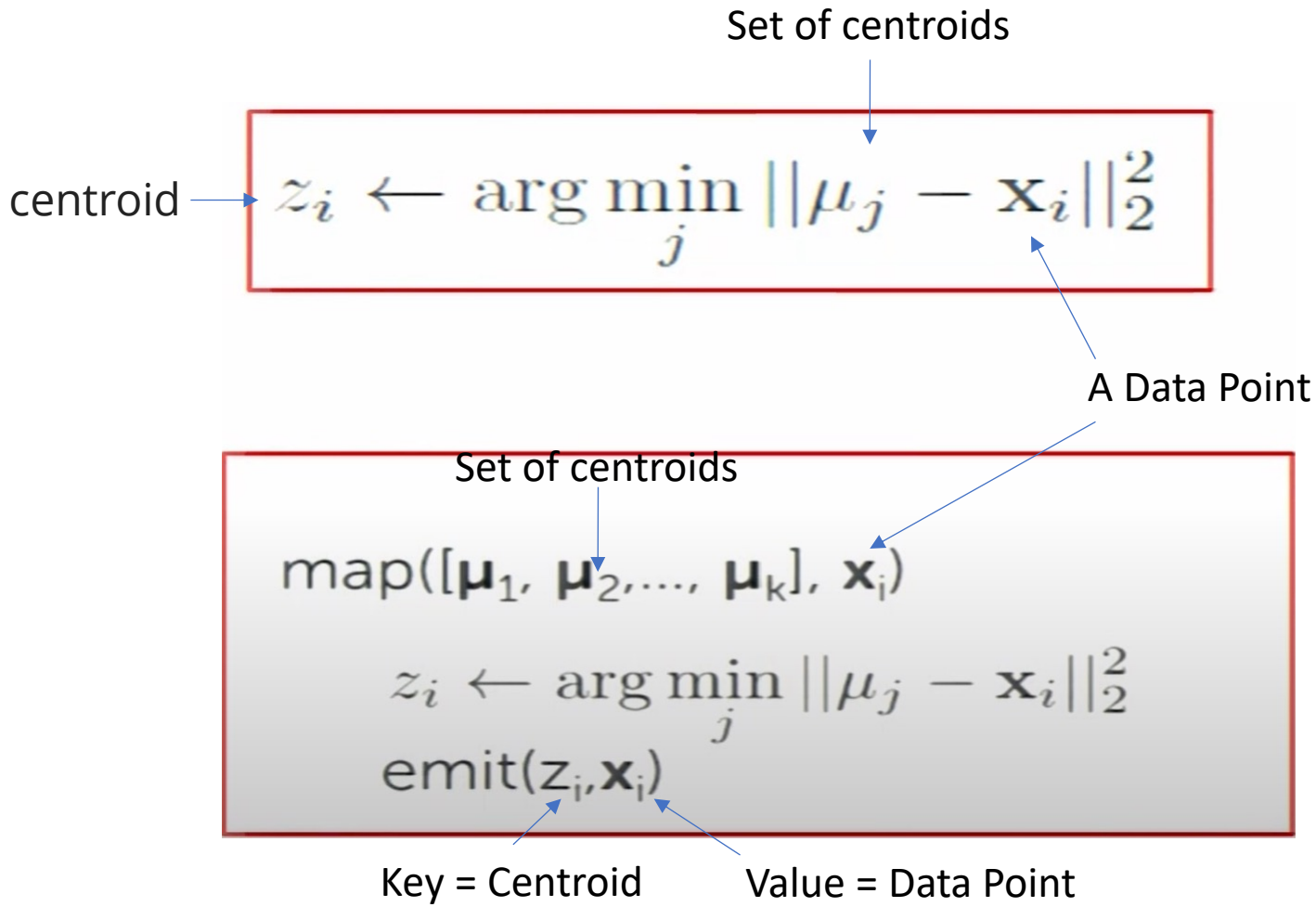
- **Recenter:** Revise cluster centers as mean of assigned observations

$$\mu_j = \frac{1}{n_j} \sum_{i:z_i=k} \mathbf{x}_i$$

Reduce: Average over all points in cluster j (zi=k)

Classification set as Map

- Classify: Assign observations to closest cluster center



Recenter step as Reduce

- Recentre: Revise cluster centers as mean of assigned observations

$$\mu_j = \frac{1}{n_j} \sum_{i:z_i=k} \mathbf{x}_i$$

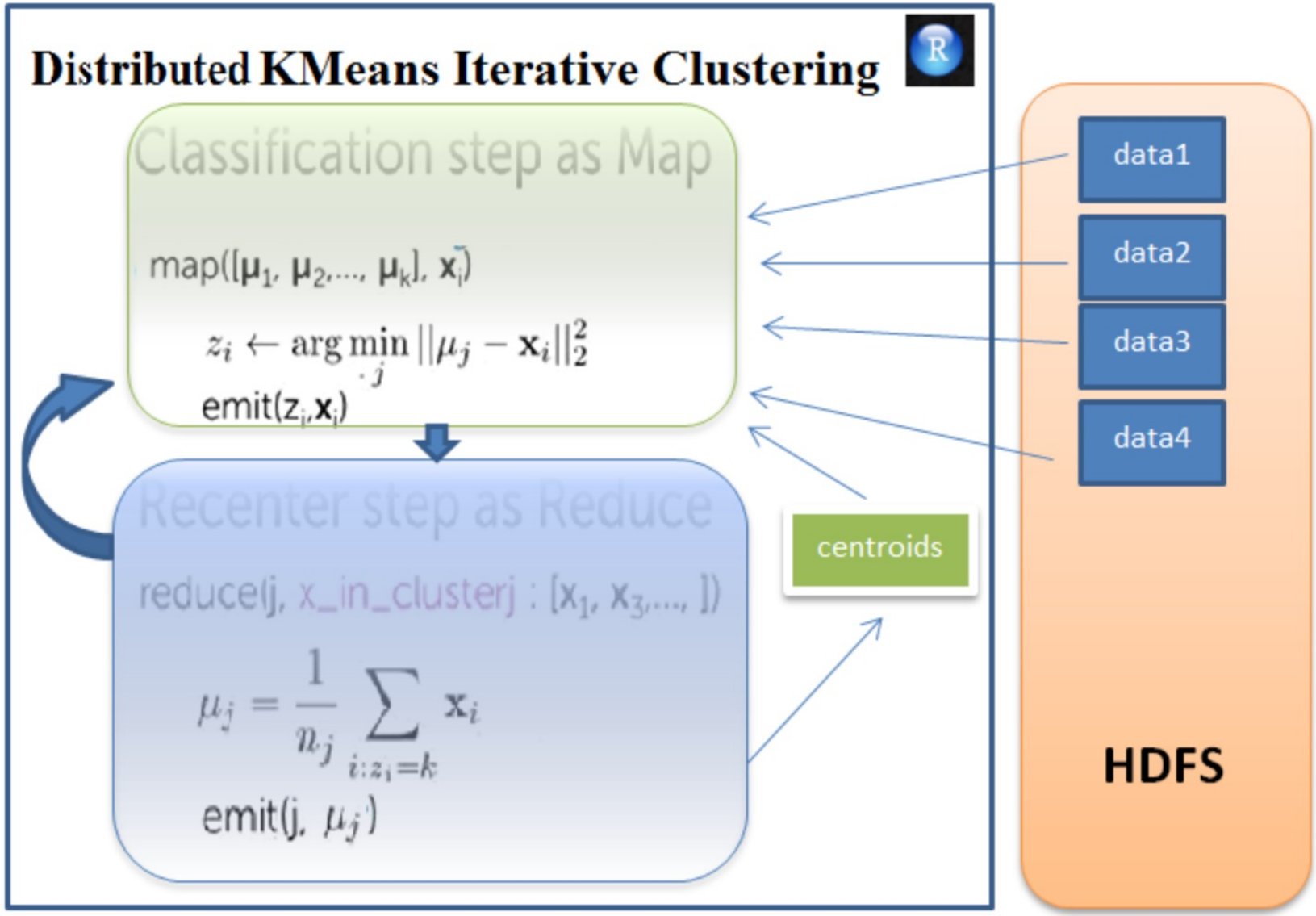
Key = Centroid

```
reduce(j, x_in_cluster j : [x1, x3,..., ])
    sum = 0
    count = 0
    for x in x_in_cluster j
    sum += x
    count += 1
    emit(j, sum/count)
```

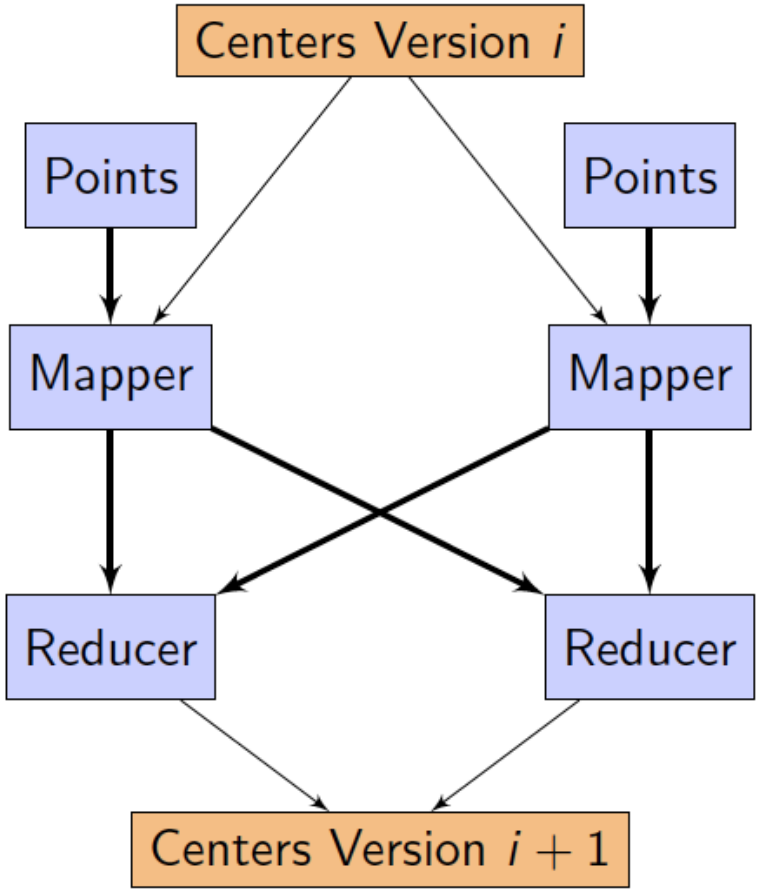
Value = Set of Data Points assigned to the **Centroid j**

Key = Centroid

Value = New mean


Distributed KMeans Iterative Clustering
Classification step as Map
map([μ1, μ2,..., μk], xi)
$z_i \leftarrow \arg\min_j ||\mu_j - \mathbf{x}_i||_2^2$
emit(zi,xi)
Recenter step as Reduce
reduce(j, x_in_clusterj : [x1, x3,...,])
$\mu_j = \frac{1}{n_j} \sum_{i:z_i=k} \mathbf{x}_i$
emit(j, μj)
centroids
data1
data2
data3
data4
HDFS

Distributed K-Means Iterative Clustering



Find Nearest Center

Key is Center, Value is Movie

Average Ratings

Summary of Parallel k-means using MapReduce

- Map : Classification step;
  - Data parallel over data points

- Reduce: Recompute means;
  - Data parallel over centers

Recap

- In this part of the lecture, we have seen an overview of cluster analysis and discussed machine learning algorithm k-means using MapReduce for big data analysis.

Applications of MapReduce

Applications of MapReduce

- Here are few simple applications of interesting programs that can easily expressed as MapReduce computation.
- **Distributed Grep:** The map function emits a line if it matches a supplied pattern. The reduce function is an identity function that just copies the supplied intermediate data to the output.
- **Count of URL Access Frequency:** The map function processes logs of web page requests and outputs (URL; 1). The reduce function adds together all values for the same URL and emits a (URL; total count) pair.
- **ReverseWeb-Link Graph:** The map function outputs (target; source) pairs for each link to target URL found in a page named source. The reduce function concatenates the list of all source URL's associated with a given target URL and emits the pair: (target; list(source))

Contd…

- **Term-Vector per Host:** A term vector summarizes the most important words that occur in a document or a set of documents as a list of (word; frequency) pairs.
- The map function emits a (hostname, term vector) pair for each input document (where the hostname is extracted from the URL of the document).
- The reduce function is passed all per-document term vectors for a given host. It adds these term vectors together, throwing away infrequent terms, and then emits a final (hostname; term vector) pair.

Contd…

- **Inverted Index:** The map function parses each document and emits a sequence of word (word; document ID) pairs. The reduce function accepts all pairs for a given word, sorts the corresponding document ID's and emits a (word; list (document ID)) pair. The set of all output pairs forms a simple inverted index. It is easy to augment this computation to keep track of word positions.
- **Distributed Sort:** The map function extracts they key from each record and emits a (key; record) pair. The reduce function emits all pairs unchanged.

Examples of MapReduce

Example 1: Counting words of different lengths

- The map function takes a value and outputs key:value pairs.
- For instance; if we define a map function that takes a string and outputs the length of the word as the key and the word itself as the value then
- Map (steve) would return 5:steve and
- Map (savannah) would return 8:savanaah
- This allows us to run the map function against values in parallel and provides a huge advantage.

Example 1: Counting words of different lengths

Before we get to reduce function, the mapreduce framework groups all the values together by key, so if the map functions output the following- key : value pairs

3:the

3:and

3:you

4:then

4:what

4:when

5:steve

5:where

8:savannah

8:research

- They are grouped as:

- 3: [the, and, you]
- 4: [then, what, when]
- 5: [where, steve]
- 8: [savannah, research]

Example 1: Counting words of different lengths

- The reductions can also be done in parallel, again providing a huge advantage. We can then look at these final results and see that there were only two words of length 5 in the corpus, etc.
- The most common example of mapreduce is for counting the number of times words occur in a corpus.

Example 1: Counting words of different lengths

- Each of these lines would then be passed as an to reduce function which accepts a key and a list of values.
- In this instance, we might be trying to figure out how many words certain lengths exist, so our reduce function will just count the number of items in the list and output the key with the size of the list, like:
- 3:3
- 4:3
- 5:2
- 8:2

Example 2: Build an Inverted Index

- Input
- tweet1, (‘I love pancakes for breakfast’)
- tweet2, (‘I dislike pancakes’)
- tweet3, (‘What should I eat for breakfast ’)
- tweet4, (‘I love to eat’)

- Desired Output
- ‘pancakes’, (tweet1, tweet2)
- ‘breakfast’, (tweet1, tweet3)
- ‘eat’, (tweet3, tweet4)
- ‘love’, (tweet1, tweet4)
- ……..

References

- https://hadoop.apache.org/docs/r1.2.1/mapred_tutorial.html#Purpose
- http://hadoop.apache.org/docs/current/hadoop-project-dist/hadoop-hdfs/hdfs-default.xml