VICTORIA UNIVERSITY OF WELLINGTON
TE HERENGA WAKA

AIML427 Big Data

Week 9-10: Hadoop MapReduce

Dr Qi Chen

School of Engineering and Computer Science

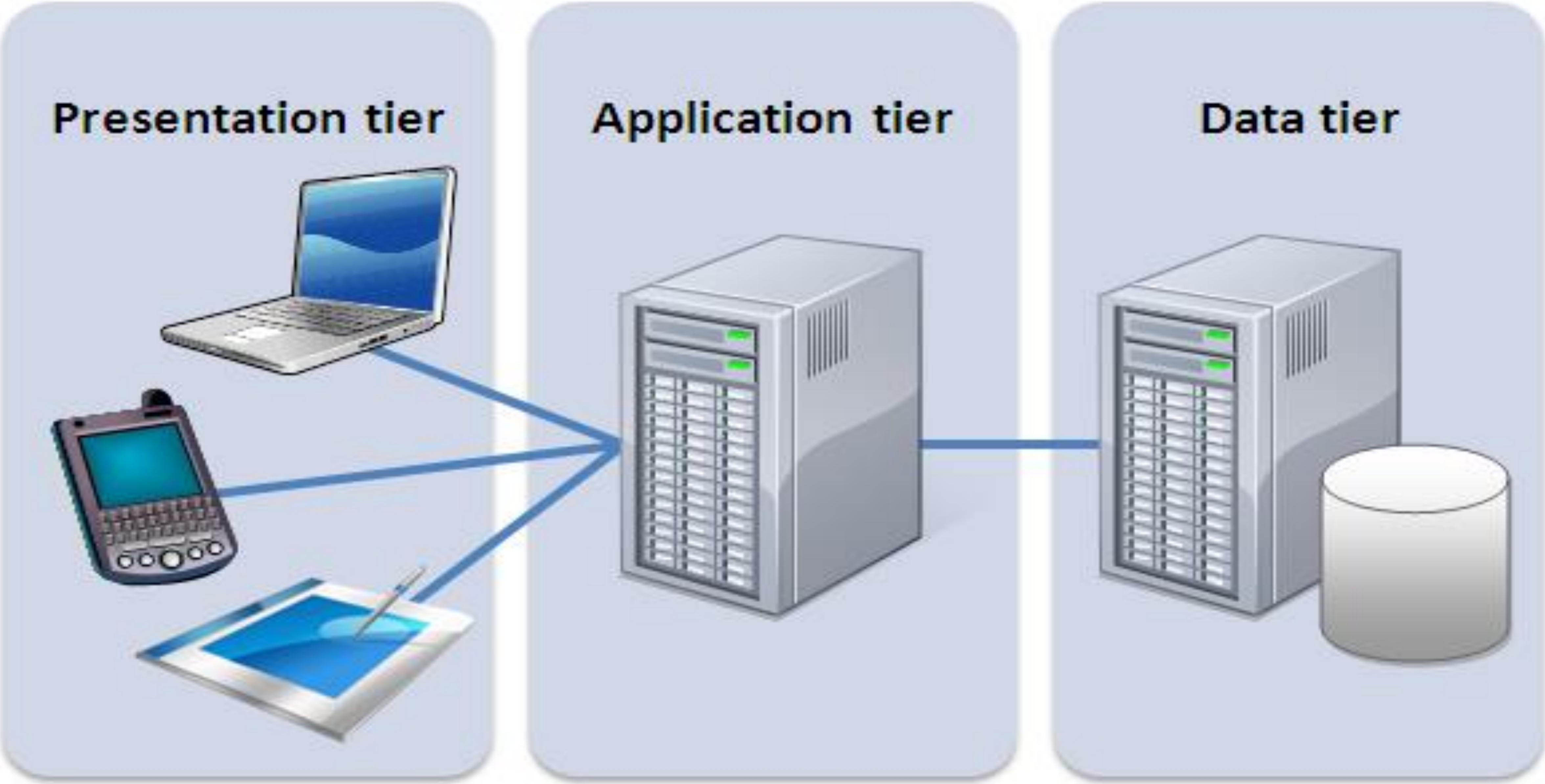
Victoria University of Wellington

Qi.Chen@ecs.vuw.ac.nz

Outline

- Machine learning tools and big data challenges
- MapReduce framework and Hadoop
- Hadoop core components
- Hadoop running modes
- Hadoop architecture
- Hadoop distributed file system (HDFS)
- MapReduce

Common Application Architecture



- How these systems look like in the big data era?

Big Data Challenges for ML

- Storage – Since data is very big, storing such huge amount of data is very difficult.
- Analytics – In Big Data, most of the time we are unaware of the kind of data we are dealing with. So processing and analysing that data is even more difficult.
- Data Quality – In the case of Big Data, data is very messy, inconsistent and incomplete.
- Discovery – Using a powerful algorithm to find patterns and insights are very difficult.
- Security – Since the data is huge in size, keeping it secure is another challenge.

Machine Learning (ML) Tools

Library	Open Source?	Scalable?	Language support	Algorithm support
MATLAB	No	No	Mostly C	High
R	Yes	No	R	High
Weka	Yes	No	Java	High
Sci-Kit Learn	Yes	No	Python	
Apache Mahout	Yes	Yes	Java	Medium
Spark ML	Yes	Yes	Scala, Java, R, Python	

Divide-And-Conquer

- Scalability to large data volumes:
  - Scan 100 TB on 1 node @ 50 MB/sec = 23 days
  - Scan on 1000-node cluster = 33 minutes
- Divide-And-Conquer by partitioning data

A single machine can not manage large volumes of data efficiently

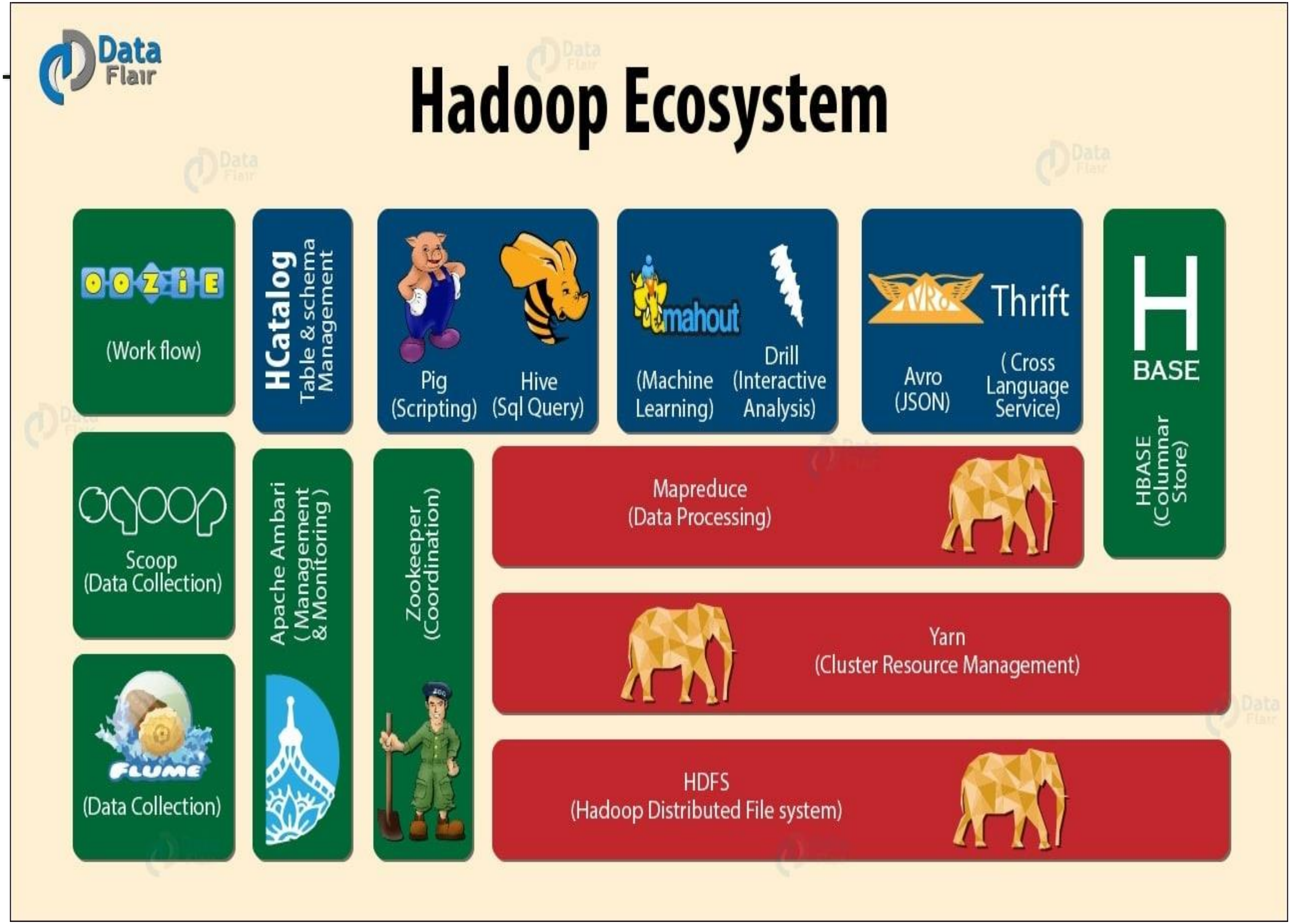
MapReduce Paradigm and Hadoop

- MapReduce paradigm (Dean and Ghemawat, 2004)
  - Data-parallel programming model
  - An associated parallel and distributed implementation for commodity clusters
- Pioneered by Google: Processes 20 PB of data per day
  - July 2008 - Hadoop wins Terabyte Sort Benchmark (sorted 1 terabyte of data in 209 seconds, which beat the previous record of 297 seconds)
- Popularized by Hadoop:
  - An open-source implementation of MapReduce paradigm.
  - It supports the distributed storage and processing of large data sets across clusters of computers using simple programming models.

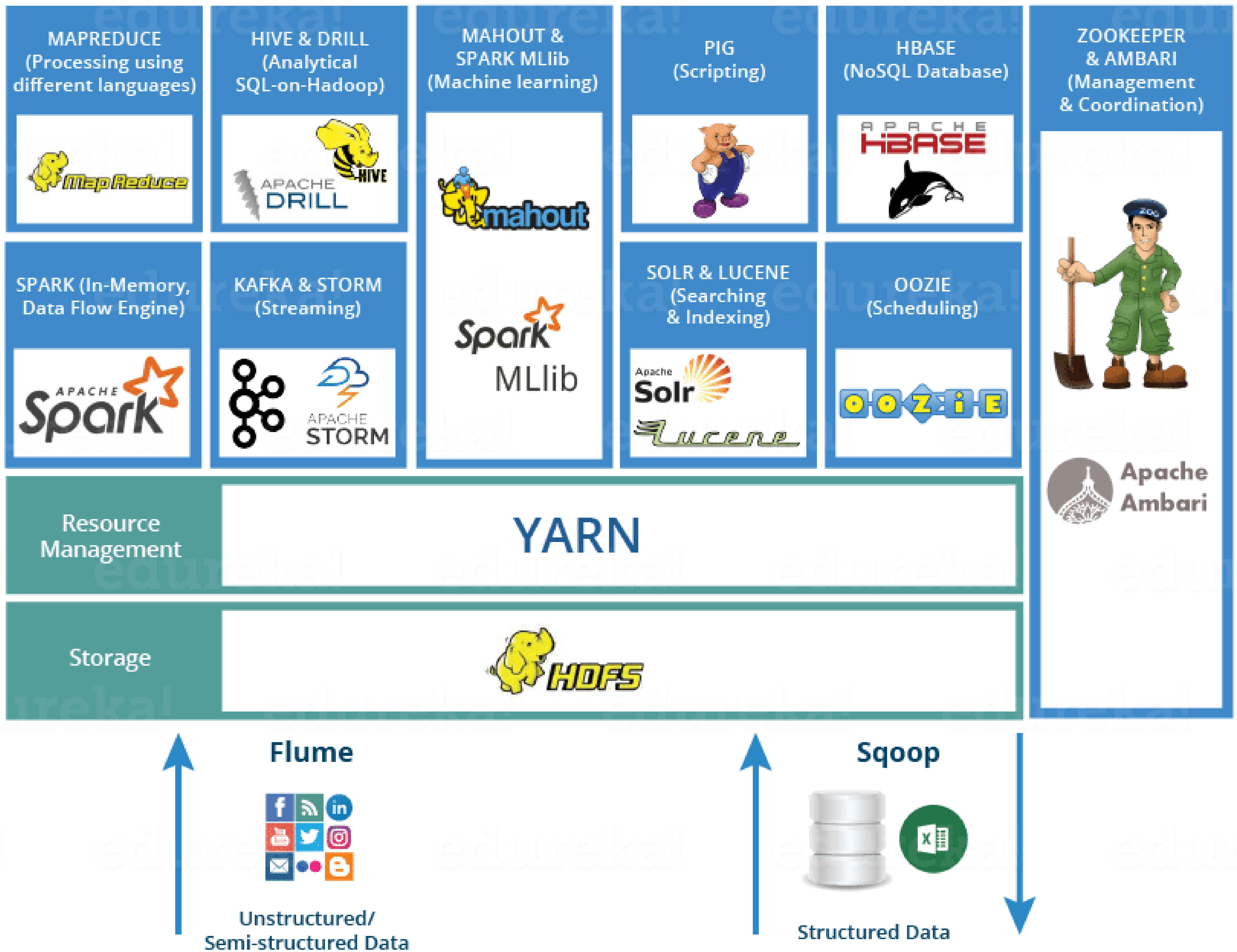
Dean, J. and Ghemawat, S. (2004). MapReduce: simplified data processing on large clusters. Commun. ACM, 51(1):107-113

Hadoop's Advantages

1. Simplicity: We can store huge files as they are (raw) without specifying any schema. MapReduce model hides complexity of distribution and fault tolerance.
2. High scalability: We can add any number of nodes, hence enhancing performance dramatically.
3. Reliability: It stores and process data reliably on the cluster despite machine failure.
4. High availability: Data is highly available despite hardware failure. If a machine or hardware crashes, then we can access data from another path.
5. Economic: Hadoop runs on a cluster of commodity hardware (nodes and network). Automatic fault-tolerance (fewer administrators). Easy to use (fewer programmers).



https://www.edureka.co/blog/hadoop-ecosystem


MAPREDUCE
(Processing using different languages)
HIVE & DRILL
(Analytical SQL-on-Hadoop)
MAHOUT & SPARK MLlib
(Machine learning)
PIG
(Scripting)
HBASE
(NoSQL Database)
ZOOKEEPER & AMBARI
(Management & Coordination)
SPARK (In-Memory, Data Flow Engine)
KAFKA & STORM
(Streaming)
SOLR & LUCENE
(Searching & Indexing)
OOZIE
(Scheduling)
Resource Management
YARN
Storage
HDFS
Flume
Unstructured/ Semi-structured Data
Sqoop
Structured Data
Apache Ambari

Hadoop Core Components

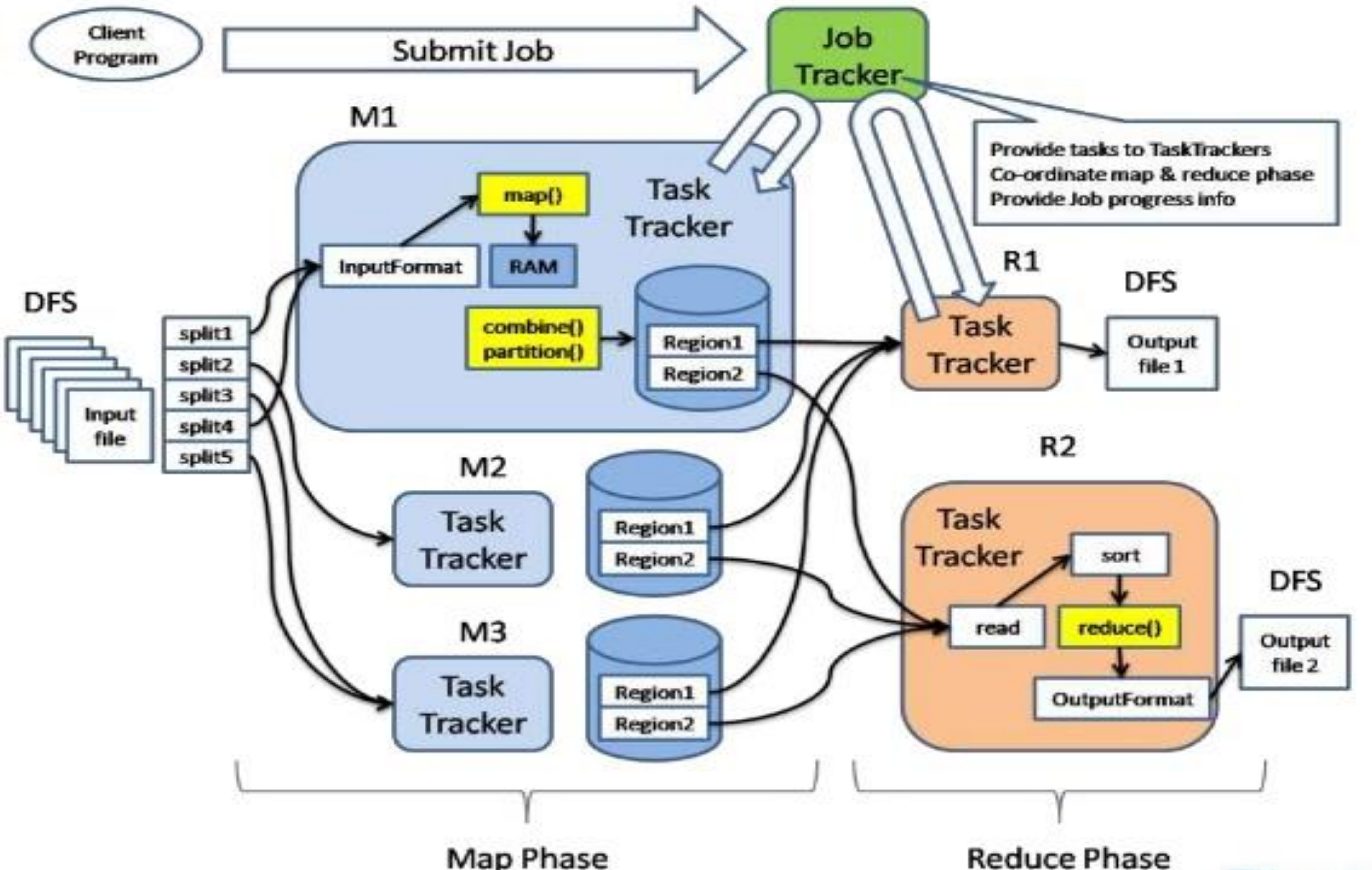
- Hadoop Distributed File System (HDFS): A distributed file system that
  - provides high-throughput access to application data.
  - distributes and stores very large files on a cluster of commodity hardware.

- Yet Another Resource Negotiator (YARN): A framework for job scheduling and cluster resource management.

- MapReduce: A software framework for parallel processing of large structured and unstructured data stored in HDFS.
  - It works by breaking the processing into phases, map and reduce.

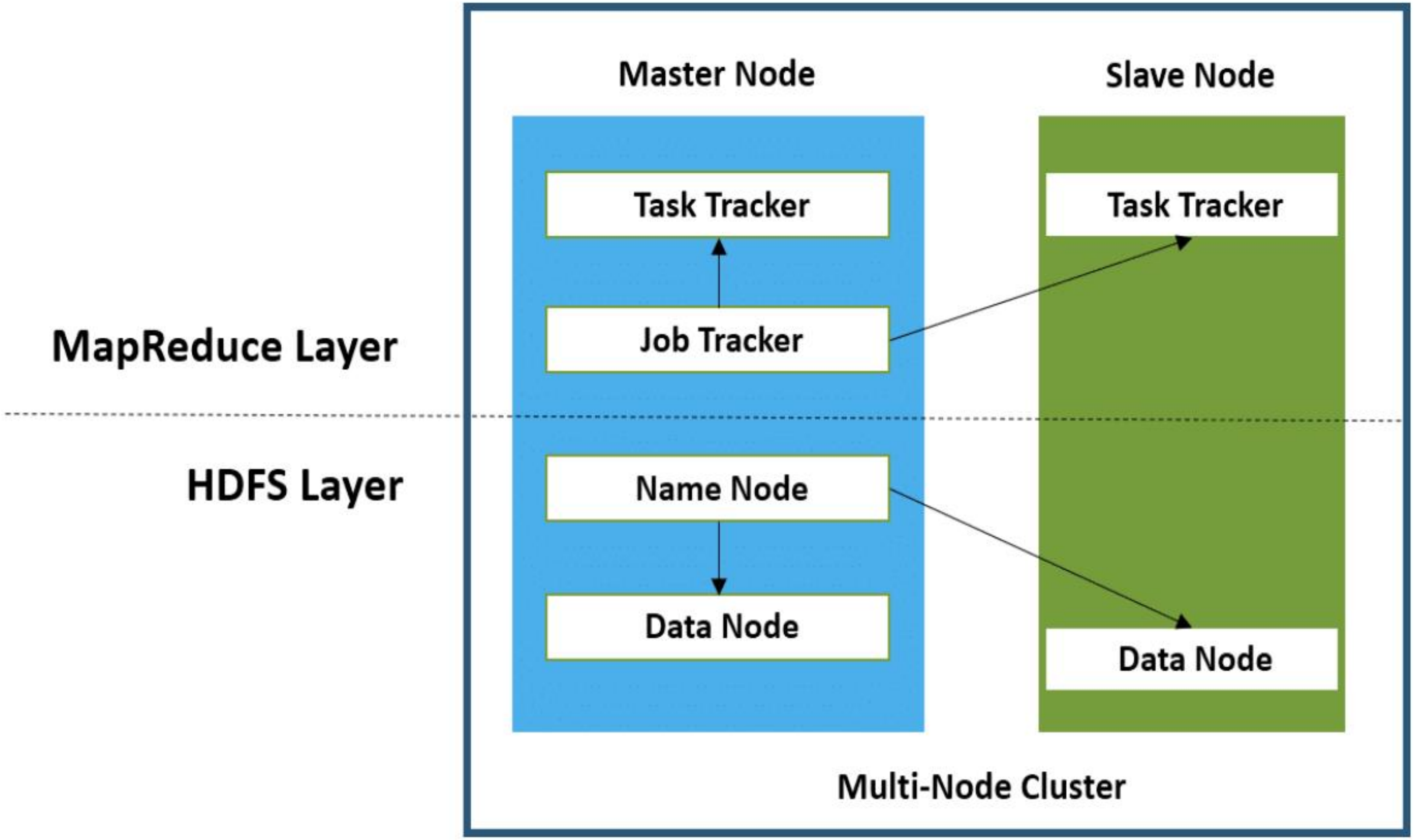
Hadoop's Architecture

Flow



https://www.slideshare.net/subhaskghosh/03-deconstructing-map-reduce-job-step-by-step

High Level Architecture of Hadoop



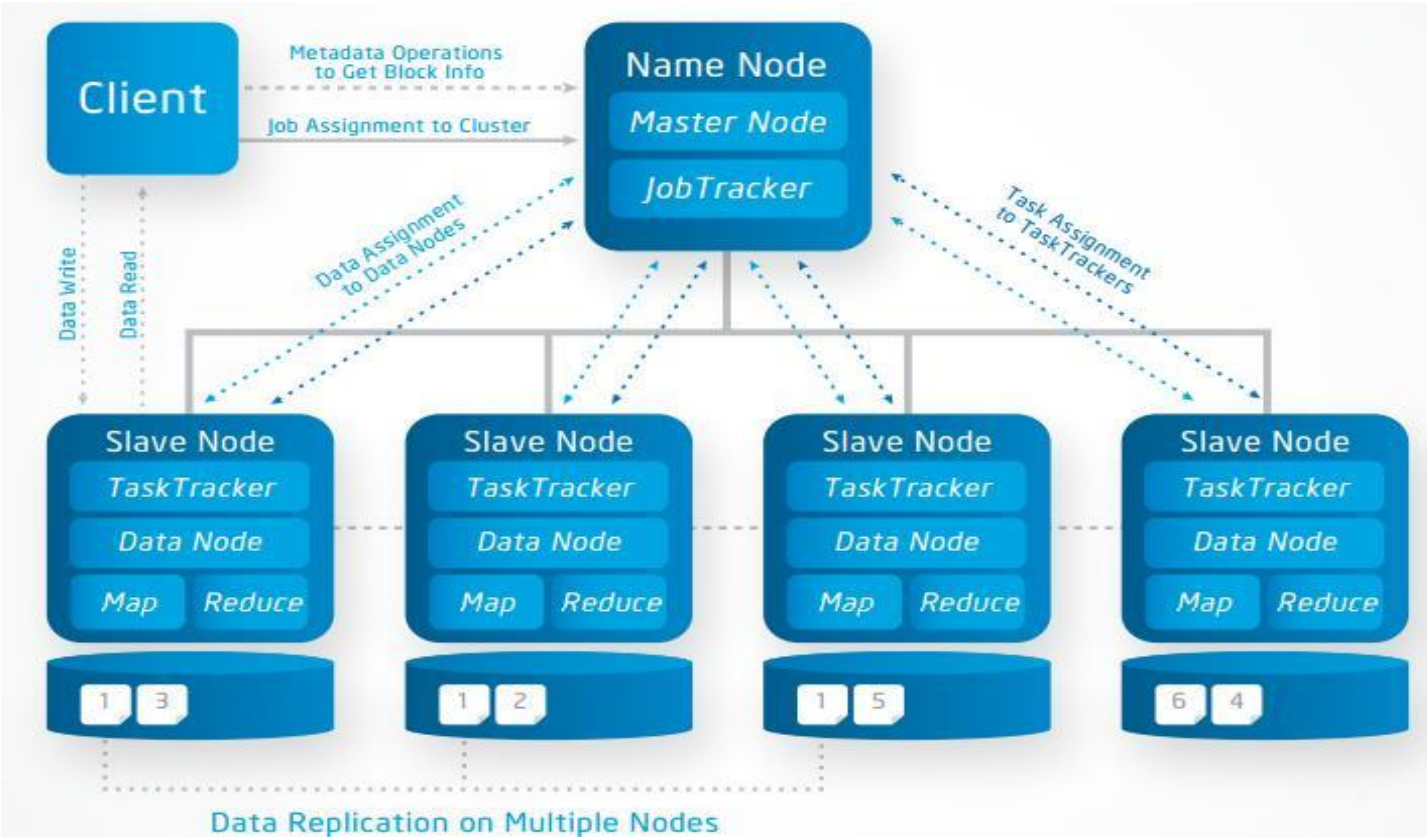
https://intellipaat.com/blog/tutorial/hadoop-tutorial/introduction-hadoop/?US

HDFS NameNode and DataNode

- An HDFS file is chopped into data blocks., each can reside on a different DataNode.
- NameNode: a master server that:
  - stores meta-data i.e., number of blocks, their location, replicas and other details.
  - manages file system namespace by executing naming, closing, opening files and directories.
  - maintains and manages the data nodes or slave nodes
  - assigns tasks to data nodes.
- DataNode usually one per node in the cluster.
  - Store actual data.
  - Performs read and write operation as per request for the clients.
  - Create, delete and replicate data blocks according to the instruction of NameNode.

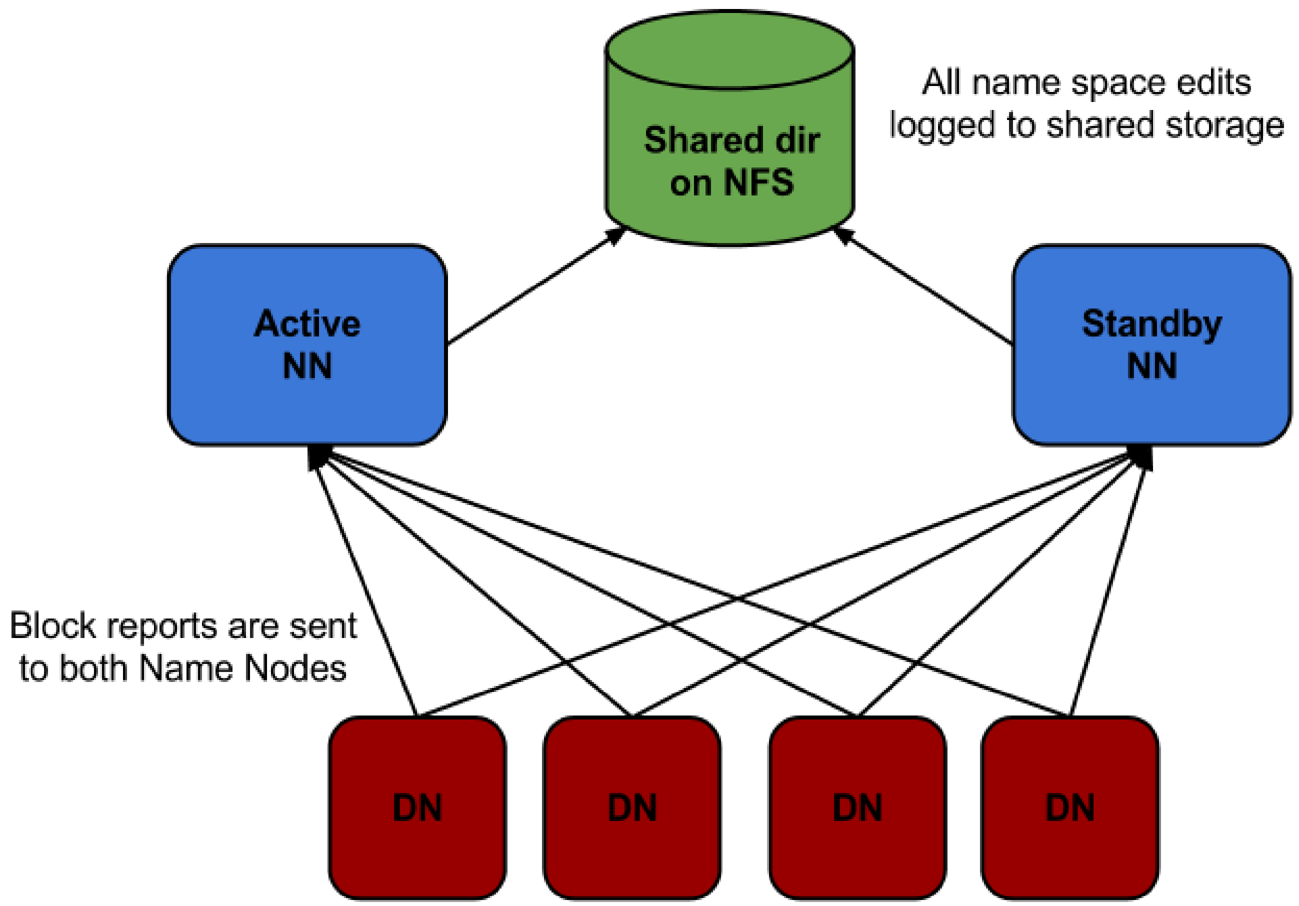
HDFS NameNode and DataNode (cont.)

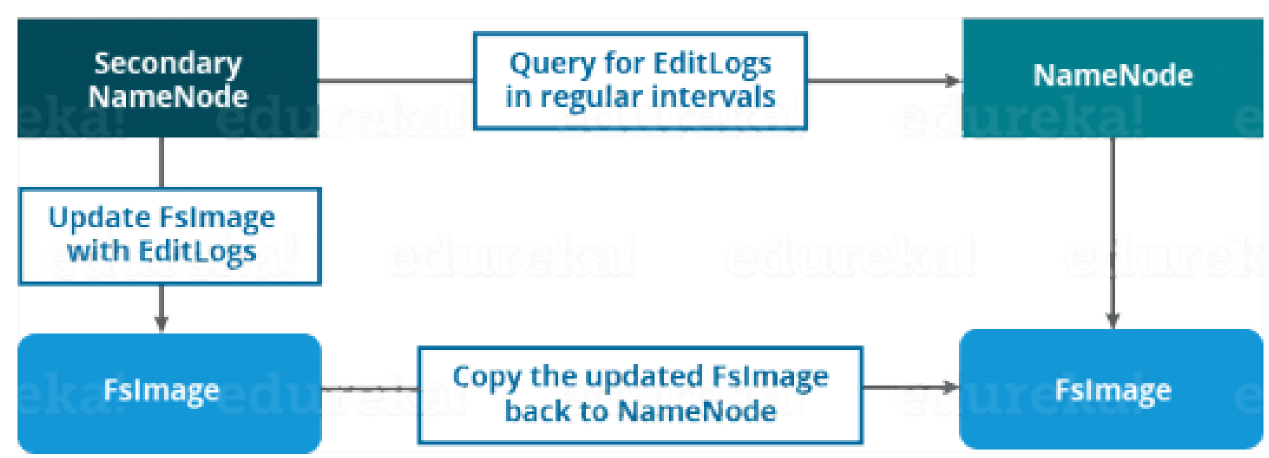
- NameNode periodically receives a Heartbeat and a Blockreport from each of the DataNodes.



Standby NameNode

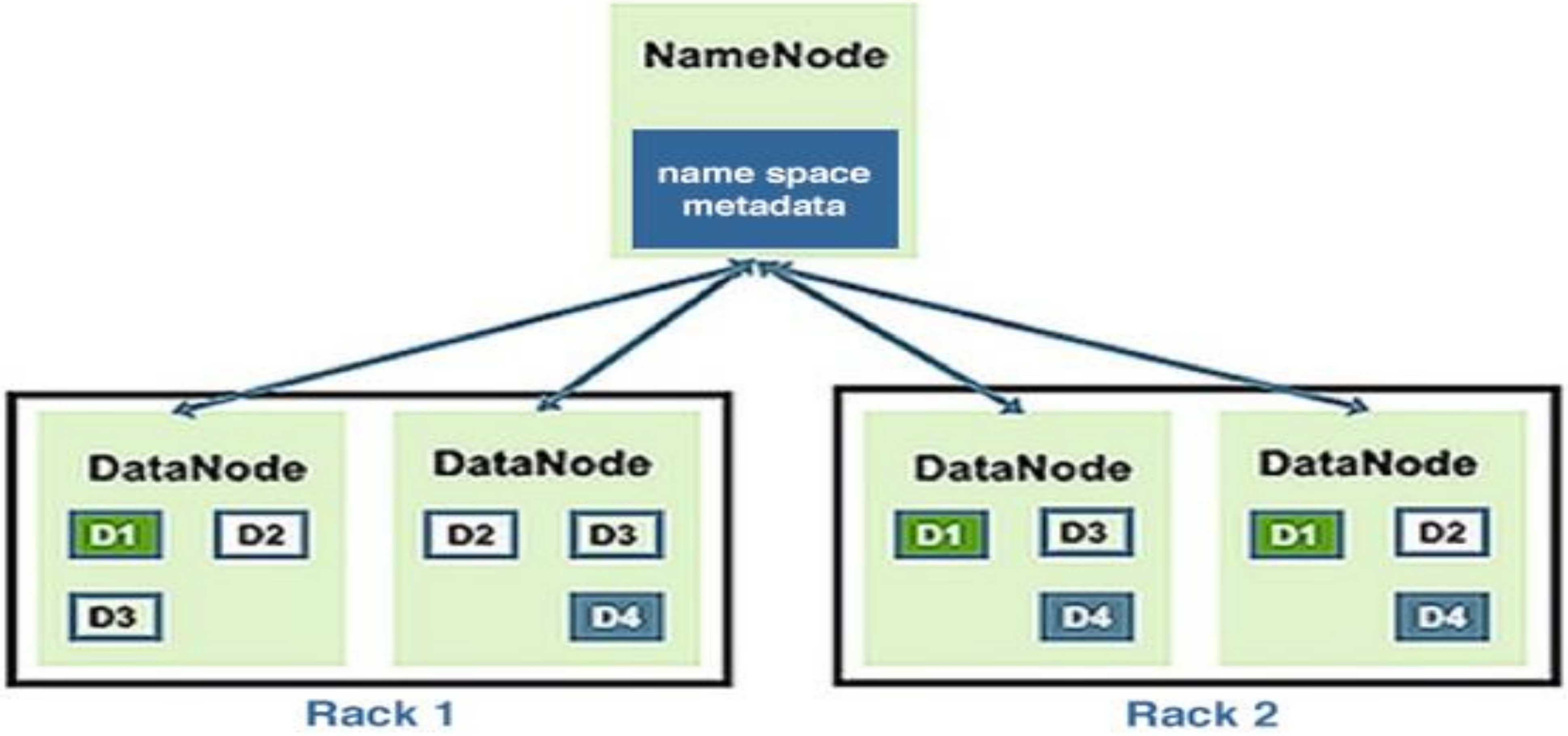
- Standby namenode is an extra nameNode that:
  - provides high availability for hadoop architecture
  - to avoid the single point of Failure (SPOF).
- If active NameNode fails, then standby Namenode takes all the responsibility of active node and cluster continues to work.





Data Replication

- HDFS replicates data to provide fault-tolerance when storing data in commodity hardware despite the higher chance of failures.
- The blocks of a file are replicated in different nodes.
- The block size and replication factor are configurable.
  - the default replication factor is 3



Hadoop Running Modes

- Local (Standalone) Mode:
  - runs in a single-node as a single Java process.
  - does not support the use of HDFS => uses the local file system for input and output operation.
  - used for debugging purpose
  - is the default mode. No custom configuration required for configuration files.
- Pseudo-Distributed Mode:
  - all daemons are running on one node.
  - but each daemon runs in a separate Java process
  - both Master and Slave node are the same.
- Fully-Distributed Mode:
  - all daemons execute in separate nodes of a multi-node cluster.
  - allows separate nodes for Master and Slave.

Hadoop Running Modes (cont.)

Component	Property	Standalone	Pseudo-distributed	Fully distributed
Core	fs.default.name	file:/// (default)	hdfs://localhost/	hdfs://namenode/
HDFS	dfs.replication	N/A	1	3 (default)
MapReduce	yarn.resourcemanager.hostname	local (default)	http://localhost:8088/	ResourceManager host.

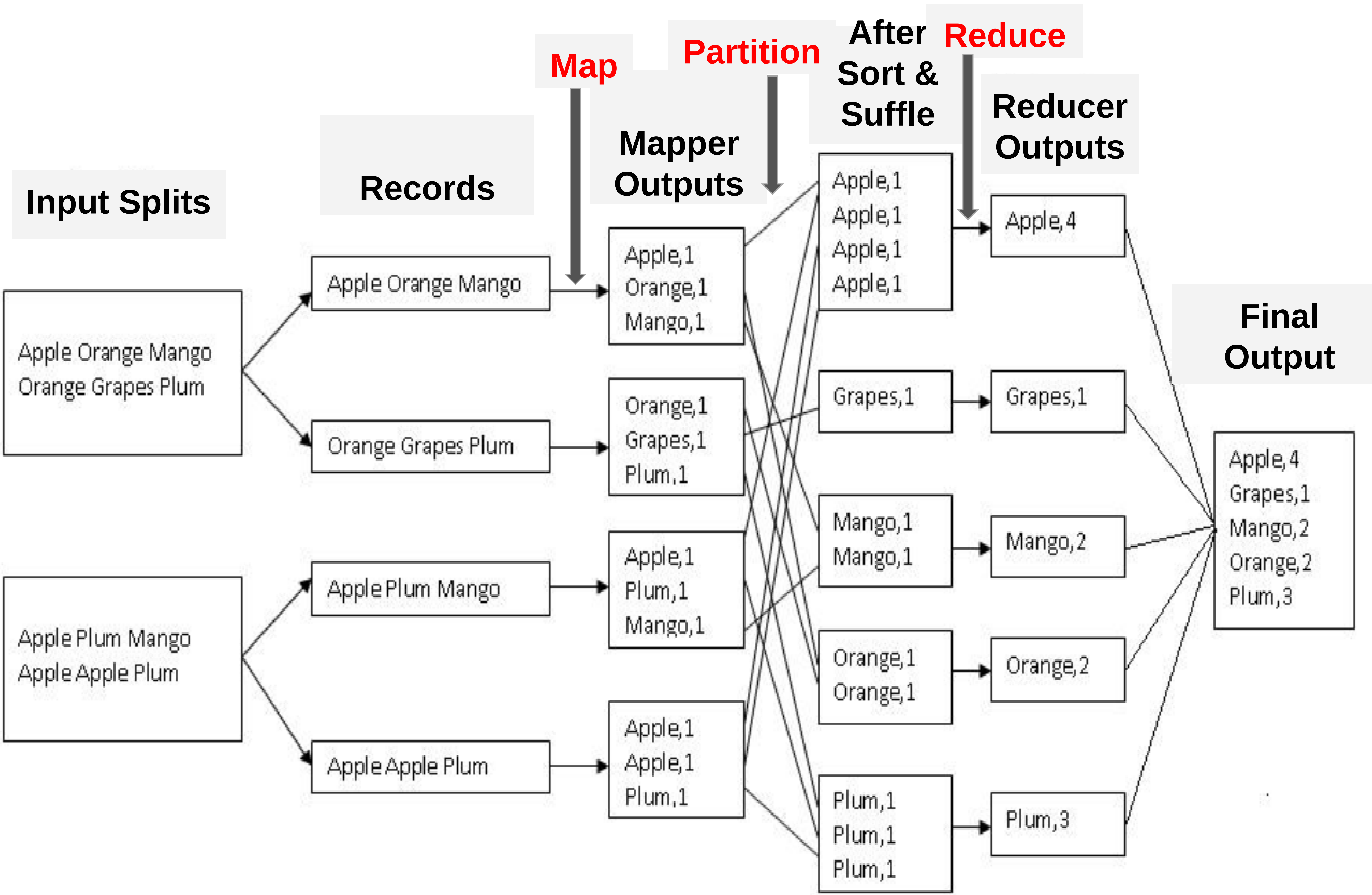
https://hadoop.apache.org/docs/r2.8.2/hadoop-project-dist/hadoop-common/SingleCluster.html

https://hadoop.apache.org/docs/r2.8.2/hadoop-project-dist/hadoop-common/ClusterSetup.html

Hadoop MapReduce (MR)

- MapReduce overcome the challenges of big data processing:
  - Cost-efficient: MR distributes the data over multiple commodity machines, because keeping the big data in one server or as database cluster is very expensive and hard to manage.
  - Time-efficient: MR moves computation rather than data, because analysing the big data in a single machine takes a lot of time.
- MapReduce's data-parallel programming model hides complexity of distribution and fault tolerance
- In a MapReduce program, Map() and Reduce() are processed in two phases:
  - Map() performs complex logic code such as filtering, grouping.
  - Reduce() specify light-weight processing like aggregates and summarizes the result produced by map function.
- MapReduce requires that the operations performed at the reduce task to be both associative and commutative.

A simple MapReduce program



The Number of Tasks

- Number of tasks can radically change the performance of Hadoop. Task setup takes a while, so it is best if the maps take at least a minute to execute.
- The right level of parallelism for maps seems to be around 10-100 maps per-node.
- The number of Map tasks is driven by the number of data blocks of the input files.
- If we have a block size of 128 MB
  - and 10TB of input data => we will have ~ 82K maps.
  - and 25GB of input data => ? maps.

The number of Reduce Tasks

- It is valid to set the number of reduce-tasks to zero: so called "Map-only job".
  - In this case the output of the map-tasks directly go to the output files in the distributed file-system as the final output.
  - Also, the framework doesn't sort the map-outputs before writing it out to HDFS.
- The number of reduce tasks is internally calculated from the size of data if it is not explicitly specified.

- Increasing the number of tasks:
  - Increases load balancing (+)
  - Lowers the cost of failures (+)
  - Increases the framework overhead (-)

Combiner

- **Problem**: A Map task may output many key-value pairs with the same key.
  - causing Hadoop to shuffle (move) all those values over the network, incurring a significant overhead.
- Combiner is mini-reducer that perform local reduce task.
- Each combine processes on output of a single mapper or split.
- Optimisation to reduce bandwidth. Note that:
  - No guarantees on being called => Not to use the combiner to perform any essential tasks.
  - Maybe only applied to a subset of map outputs
- Often is the same class as Reducer.

Key-Value Pair

- The key-value pair is the record entity that MapReduce job receives for execution.
- Generally:
  - map: <key1, value1> -> list <key2, value2>
  - [combine: <key2, list<value2>> -> list <key2, value2>]
  - reduce: <key2, list<value2>> -> list <key3, value3>

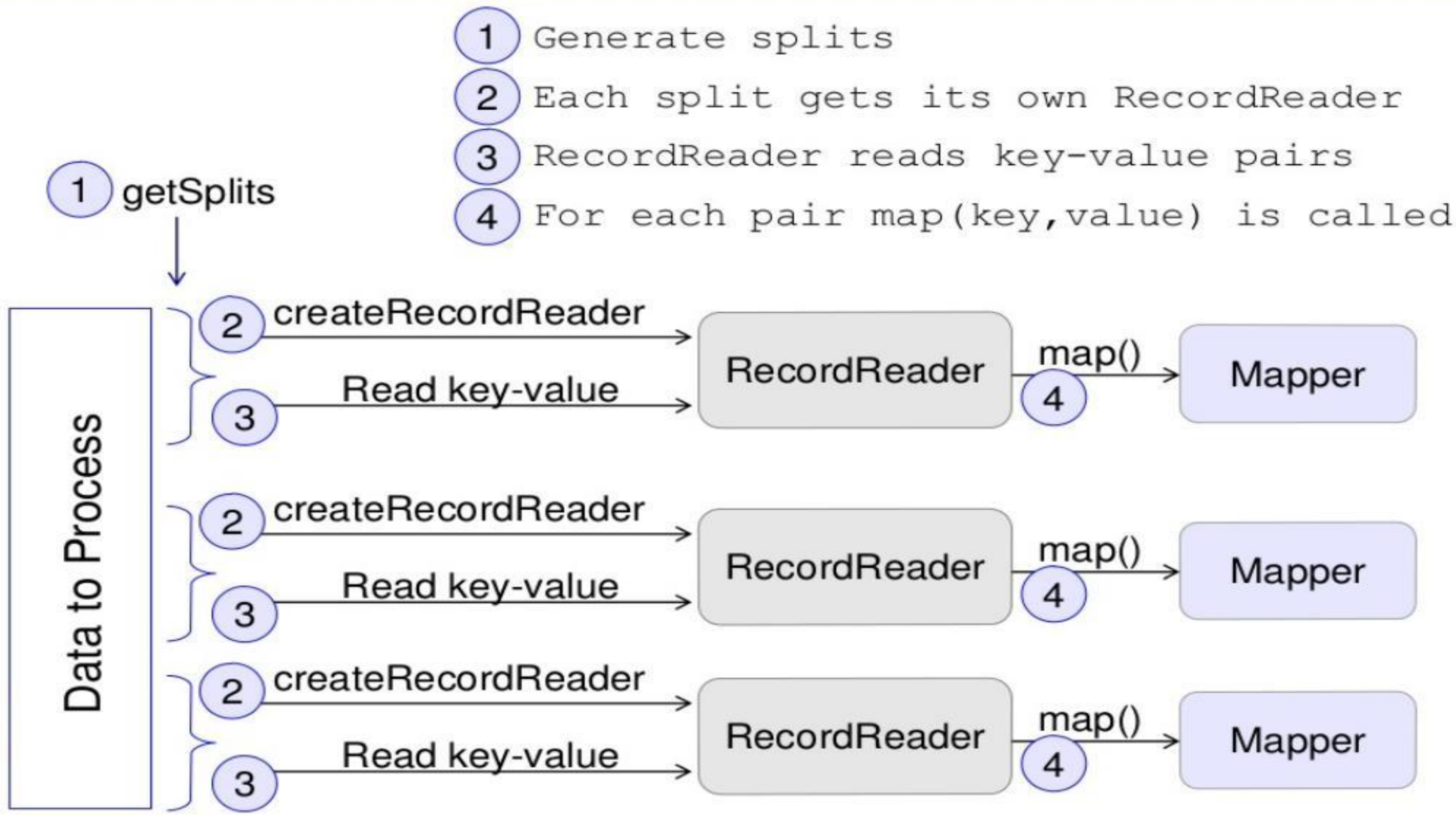
- The key and value classes have to be serializable by the framework and hence need to implement the Writable interface.
- The key classes have to implement the WritableComparable interface to facilitate sorting by the framework.
- Key-value pair enables MapReduce to work with unstructured and semi-structured data.

InputSplit

- Splits are a set of logically arranged records
  - A set of lines in a file
  - A set of rows in a database table
- Each instance of mapper will process a single split
  - Map instance processes one record at a time
    - map(key, value) is called for each record.
- Splits are implemented by extending InputSplit class
- However, we don’t usually need to deal with splits directly
  - It is InputFormat’s responsiblity

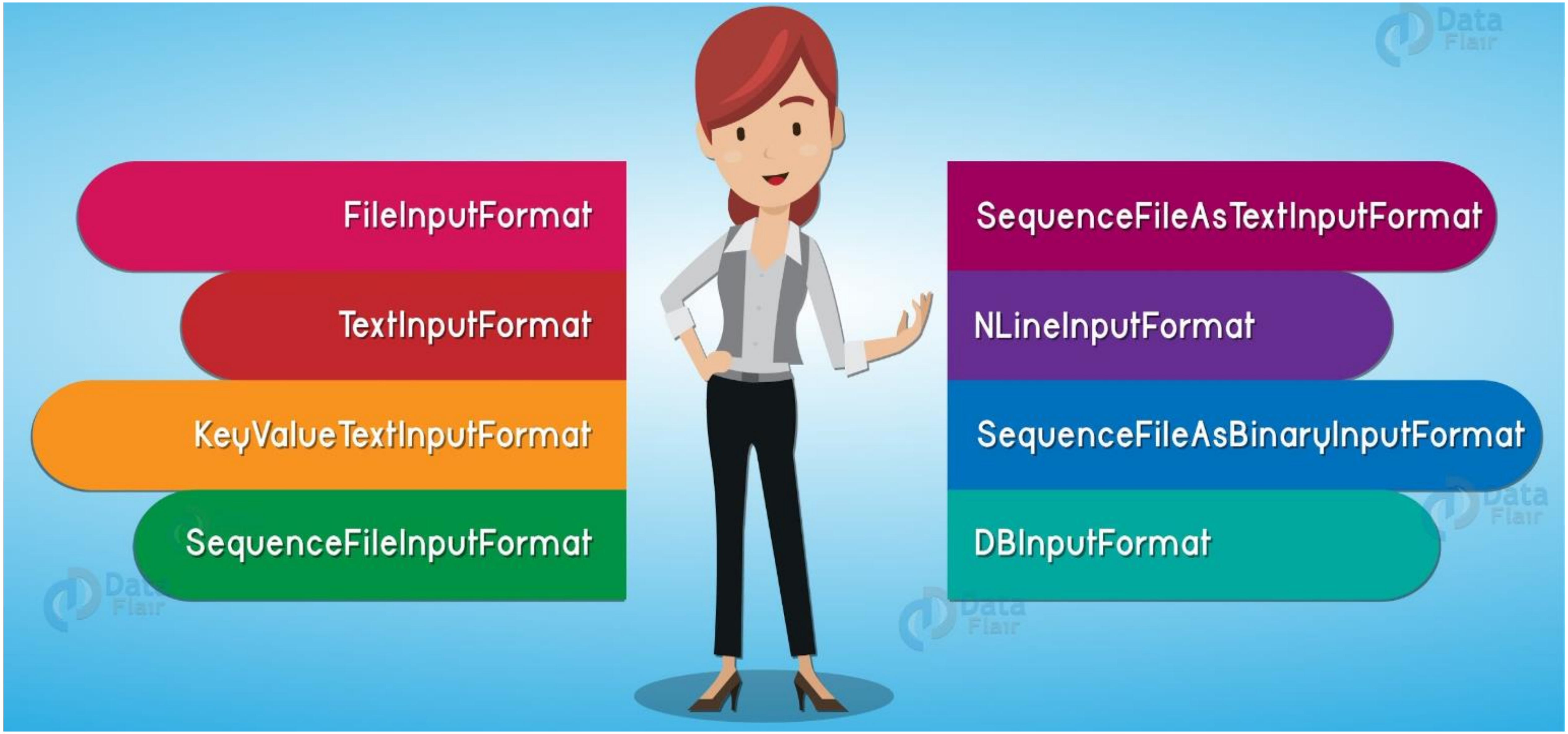
InputFormat

- The InputFormat performs the splitting of the input data into the key-value pair inputs for the mappers.
- It defines how the input files are split up and read in Hadoop.



Hadoop InputFormat

- Configure on a Job object:
  - job.setInputFormatClass (XXXInputFormat.class);



Hadoop InputFormat (cont.)

- FileInputFormat:
  - Is the base class for all file-based InputFormats.
  - Specifies input directory where data files are located.
  - Read all files and divides these files into one or more InputSplits.
- TextInputFormat:
  - Default format.
  - Useful for unformatted data or line-based records like log files.

Split	Single HDFS block (can be configured)
Record	Single line of text; linefeed or carriage-return used to locate end of line
Key	LongWritable - Position in the file
Value	Text - line of text (excluding line terminators)

Hadoop InputFormat (cont.)

- KeyValueTextInputFormat: similar to TextInputFormat

Split	Single HDFS block (can be configured)
Record	Single line of text
Key	Text - First value before delimiter
Value	Text - the rest of the line (excluding line terminators)

- If a line does not contain the delimiter, the whole line will be treated as the key and the value will be empty.
- The default delimiter is tab. It can be set to another by:

```
Configuration conf = new Configuration();
conf.set("mapreduce.input.keyvaluelinerecordreader.key.value.separator", ",");
```

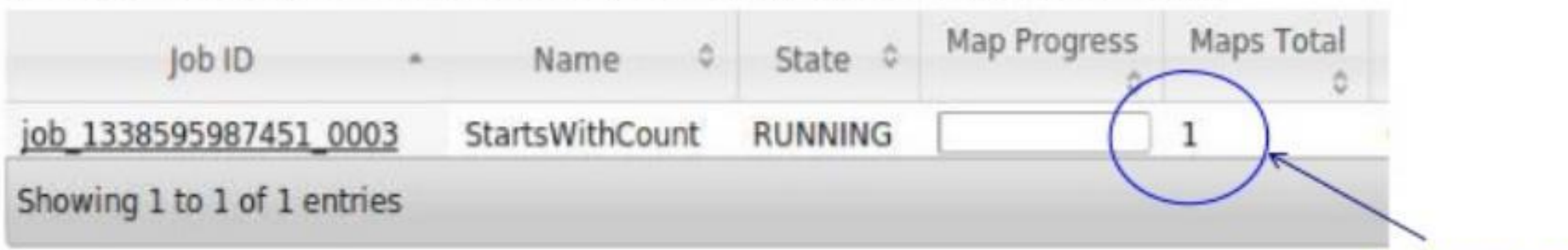
Hadoop InputFormat (cont.)

- NLineInputFormat: used for plain text files.

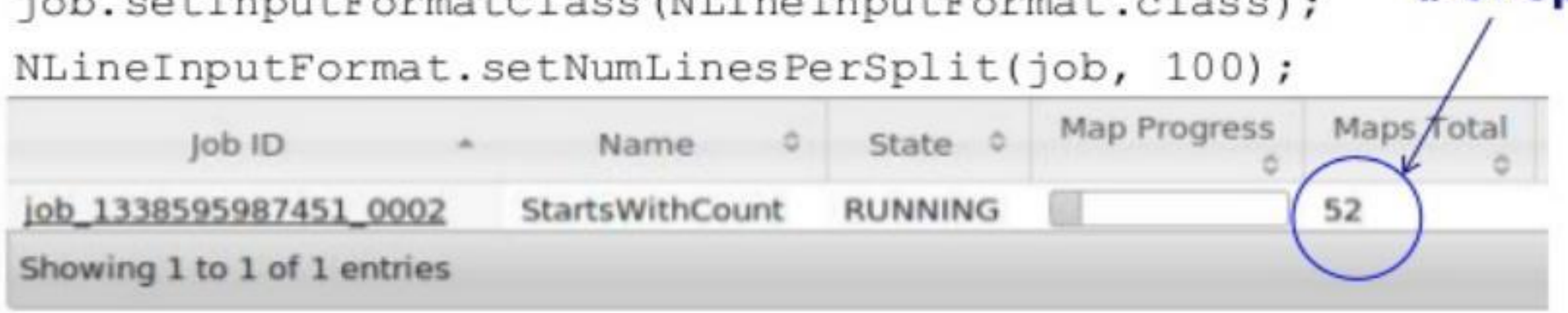
Split	N lines. Set by: NLineInputFormat.setNumLinesPerSplit(job,N);
Record	Single line of text
Key	LongWritable - Position in the file
Value	Text - line of text (excluding line terminators)

Input is /training/playArea/hamlet.txt
- 5159 lines
- 206.3k

```
job.setInputFormatClass(TextInputFormat.class);
```




```
job.setInputFormatClass(NLineInputFormat.class);
NLineInputFormat.setNumLinesPerSplit(job, 100);
```



OutputFormat

- Specification for writing data
- Implementation of OutputFormat<Key,Value>
- TextOutputFormat is the default implementation
  - Output records as lines of text
  - Key and values are tab separated.
  - Key and values may be of any type.
- OutputFormat:
  - validates output specification for that job.
    - E.g.: check if the output directory existed => returns an error.
  - creates implementation of RecordWriter
  - creates implementation of OutputCommitter
    - Set-up and clean-up Job's and Task's artifacts
    - Commit or discard tasks output.

MapReduce Job

- Job is the primary interface for a user to describe a map-reduce job.
- Job configuration is done through a Configuration object

```
Configuration conf = new Configuration();
Job job = new Job(conf);
```

- Job is used to specify the Mapper, Reducer, InputFormat, OutputFormat, Combiner, Partitioner, etc.
- Note that the framework tries to faithfully execute the job as-is described, however:
  - Some configuration parameters might have been marked as final by administrators and hence cannot be altered.
  - Some parameters interact subtly with the rest of the framework and/or job-configuration and is relatively more complex for the user to control finely (e.g. setNumMapTasks(int)).

MapReduce Job Configuration (cont.)

```
//Set Mapper, Combiner and Reducer
job.setMapperClass(MyJob.MyMapper.class);
job.setCombinerClass(MyJob.MyReducer.class);
job.setReducerClass(MyJob.MyReducer.class);
//Set Input and Output Format
job.setInputFormat(SequenceFileInputFormat.class);
job.setOutputFormat(SequenceFileOutputFormat.class);
//Set Input and Output Path
FileInputFormat.setInputPaths(job, new Path("in"));
FileOutputFormat.setOutputPath(job, new Path("out"));
```

Hadoop Data Types

- Hadoop uses the Writable interface based classes as the data types for the MapReduce computations.
- Choosing the appropriate Writable data types for your input, intermediate, and output data is important for the performance and the programmability of MapReduce programs.
- The reducer's input key-value pair data types should match the mapper's output key-value pairs.

Hadoop Data Types

- Hadoop built-in data types for both key and value:
  - IntWritable
  - LongWritable
  - BooleanWritable
  - FloatWritable
  - ByteWritable: a sequence of bytes
  - Text: a UTF8 text
  - VIntWritable and VLongWritable: variable length integer and long values
  - NullWritable: a zero-length Writable type that can be used when you don't want to use a key or value type

Hadoop Data Types (cont.)

- Hadoop build-in data types can only be used as value types.
  - ArrayWritable: This stores an array of values belonging to a Writable type. To use this type as the value type of a reducer's input, you need to create a subclass of ArrayWritable to specify the type of the Writable values stored in it.
  - TwoDArrayWritable: This stores a matrix of values belonging to the same Writable type. Similarly, you need to specify the type of the stored values by creating a subclass of this type.
  - MapWritable: This stores a map of key-value pairs. Keys and values should be of the Writable data types.
  - SortedMapWritable: This stores a sorted map of key-value pairs. Keys should implement the WritableComparable interface.

Data Types Example

- Specify the data types for key-value pairs using the generic-type variables.

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

input types output types

```
    public void map(LongWritable key, Text value, Context context) {
        ......  }
}
public class SampleReducer extends Reducer
    <Text, IntWritable, Text, IntWritable> {
```

input types output types

```
    public void reduce(Text key, Iterable<IntWritable> values,
                       Context context) {
            ......  }
}
```

Data Type Configuration

- Specify the output data types for both the reducer and the mapper

```
job.setOutputKeyClass(Text.class);
job.setOutputValueClass(IntWritable.class);
```

- If mapper has different data types for the output key-value pairs:

```
job.setMapOutputKeyClass(Text.class);
job.setMapOutputValueClass(IntWritable.class);
```

Set Input & Output Paths

- Set the input paths to the job.

  `FileInputFormat.setInputPaths(job, new Path(inputPath));`

- Set multiple HDFS input paths:
  - Set the array of Paths as the list of inputs for the job:

    `FileInputFormat.setInputPaths(job, Path... inputPaths)`

  - Or by providing a comma-separated list of paths:

    `FileInputFormat.setInputPaths(job, commaSepartedString)`

  - Or use the addInputPath() to add input paths:

    `FileInputFormat.addInputPath(job, Path path)`

- Set the output path to the job.

  `FileOutputFormat.setOutputPath(job, new Path(String));`

WordCount Example

```
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.*;
// Note: org.apache.hadoop.mapred is an older API
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;

public class WordCount {              //Driver class
        // Map class
    // Reduce class
    // Main function
}
```

WordCount Example - Map class

```
public static class Map extends Mapper
    <LongWritable, Text, Text, IntWritable> {
        private final static IntWritable one = new IntWritable(1);
        private Text word = new Text();

        public void map(LongWritable key, Text value, Context context)
        throws IOException, InterruptedException {
            String line = value.toString();
            StringTokenizer tokenizer = new StringTokenizer(line);
            while (tokenizer.hasMoreTokens()) {
                word.set(tokenizer.nextToken());
                context.write(word, one);
            }
        }
    }
```

WordCount Example - Reduce class

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

WordCount Example - Main function

```
public static void main(String[] args) throws Exception {
    Configuration conf = new Configuration();
    conf.set(MRJobConfig.NUM_MAPS, "3");
    Job job = Job.getInstance(conf, "Word Count New");
    job.setOutputKeyClass(Text.class);
    job.setOutputValueClass(IntWritable.class);
    job.setMapperClass(Map.class);
    job.setCombinerClass(Reduce.class);
    job.setReducerClass(Reduce.class);

    FileInputFormat.addInputPath(job, new Path(args[0]));
    FileOutputFormat.setOutputPath(job, new Path(args[1]));

    System.exit(job.waitForCompletion(true) ? 0 : 1);
  }
```

https://hadoop.apache.org/docs/r2.8.0/hadoop-mapreduce-client/hadoop-mapreduce-client-core/MapReduceTutorial.html

Where can I access to a Hadoop platform?

- Cloud platform with Hadoop installation

amazon web services™ | EC2

Windows Azure™

Google Actual Cloud Platform

- Install your own cluster.

ECS Hadoop Cluster (CO246)

ECS Hadoop Cluster

- To help you get started with your assignment, school has install a Hadoop cluster in the lab CO246 which allows you to try out some basic operations with a Hadoop cluster.
- The installed version is 3.3.6.
- Use your ECS account to access this Hadoop cluster which is a 8-node Hadoop cluster including:
  - co246a-1.ecs.vuw.ac.nz
  - co246a-2.ecs.vuw.ac.nz
  - ...
  - co246a-7.ecs.vuw.ac.nz
  - co246a-8.ecs.vuw.ac.nz

  and:

  - NameNode: co246a-a.ecs.vuw.ac.nz
  - YARN resource manager host: co246a-9.ecs.vuw.ac.nz
- [Lab tutorial]

Exploring HDFS

- HDFS supports a traditional hierarchical file organization.
- A user or an application can create directories and store files inside these directories.
- hadoop fs <args> or hdfs dfs <args>
- ls
  - Usage: hadoop fs -ls [-R] [-t] [-S] [-r] [-u] <args>
    - -R: Recursively list subdirectories encountered.
    - -t: Sort output by modification time (most recent first).
    - -S: Sort output by file size.
    - -r: Reverse the sort order.
    - -u: Use access time rather than modification time for display and sorting.
- mkdir: create directory
  - Usage: hadoop fs -mkdir [-p] <paths>
    - -p: creating parent directories along the path.

Exploring HDFS (cont.)

- put: Copy local files to HDFS. Also reads input from stdin and writes to destination if the source is set to "-"
  - Usage: hadoop fs -put [-f] [- | <localsrc1> ..]. <dst>
  - hadoop fs -put -f localfile1 localfile2 /user/hadoop/hadoopdir
- get: Copy files to the local file system.
  - Usage: hadoop fs -get [-f] <src> <localdst>
- cp: Copy files from source to destination.
  - Usage: hadoop fs -cp [-f] URI [URI ...] <dest>
  - hadoop fs -cp /user/hadoop/file1 /user/hadoop/file2
  - hadoop fs -cp /user/hadoop/file1 /user/hadoop/file2 /user/hadoop/dir
- mv: Moves files from source to destination
  - Usage: hadoop fs -mv URI [URI ...] <dest>
- rm: Delete files specified as args
  - Usage: hadoop fs -rm [-r] [-skipTrash] [-safely] URI [URI ...]

Exploring HDFS (cont.)

- appendToFile Append single src, or multiple srcs from local file system to the destination file system.
  - Usage: hadoop fs -appendToFile <localsrc> ... <dst>
  - hadoop fs -appendToFile localfile /user/hadoop/hadoopfile
  - hadoop fs -appendToFile localfile1 localfile2 /user/hadoop/hadoopfile
  - hadoop fs -appendToFile - hdfs://nn.example.com/hadoop/hadoopfile Reads the input from stdin.
- cat: Copies source paths to stdout.
  - Usage: hadoop fs -cat URI [URI ...]
  - hadoop fs -cat /user/hadoop/file1 /user/hadoop/file1
- copyFromLocal: -f overwrite if exist.
  - Usage: hadoop fs -copyFromLocal <localsrc> URI
  - hadoop fs -copyFromLocal -f localfile /user/hadoop/
- copyToLocal:
  - Usage: hadoop fs -copyToLocal URI <localdst>

Compile & Running a MapReduce Program

- Compile the MapReduce program: WordCount.java
  - mkdir wordcount_classes
  - javac -d wordcount_classes WordCount.java
  - jar cvf wordcount.jar -C wordcount_classes/ .
  - ls
- Run:
  - hadoop jar wordcount.jar myPackage.WordCount input output
- Check the results:
  - hdfs dfs -cat output/part-r-00000

- Step by step tutorial of how to use the school cluster can be found at the lab tutorial document.

Set Environment Variables

- Run:
  - export HADOOP_VERSION=3.3.6
  - export HADOOP_HOME=/local/Hadoop/hadoop-$HADOOP_VERSION
  - export PATH=${PATH}:$HADOOP_HOME/bin
  - export HADOOP_CONF_DIR=${HADOOP_HOME}/etc/hadoop
  - Or save the bove three lines into a file and source it:
    - source AIML427_hadoop_setup.csh
- Set PATH for Java:
  - need java

- set CLASSPATH for Hadoop:
  - hadoop classpath --glob > setup_hadoop_classpath.csh
  - vim setup_hadoop_classpath.csh: add setenv CLASSPATH at the beginning and delete the last component of the line.
  - source setup_hadoop_classpath.csh

Where to get big data?

- UCI machine learning repository: https://archive.ics.uci.edu/ml/index.html
- Kaggle datasets (machine learning competitions): https://www.kaggle.com/competitions
- Transportation Statistics: *https://www.transtats.bts.gov/DL_SelectFields.asp?Table_ID=236*
- Government open data: *https://open.canada.ca/data/en/dataset?portal_type=dataset*
- The CIA World Factbook (provides information on the history, population, economy, government, infrastructure, and military of 267 countries): https://www.cia.gov/library/publications/download/
- Financial dataset from Lending Club: https://www.lendingclub.com/info/download-data.action
- Research data from Yahoo: http://webscope.sandbox.yahoo.com/index.php

Where to get big data?

- Amazon AWS public dataset http://aws.amazon.com/public-data-sets/
- Labeled visual data from Image Net http://www.image-net.org
- Compiled YouTube dataset http://netsg.cs.sfu.ca/youtubedata/
- Collected rating data from the MovieLens site http://grouplens.org/datasets/movielens/
- Movie dataset http://www.imdb.com/interfaces
- Social science data http://www.icpsr.umich.edu/icpsrweb/ICPSR/studies
- Datasets from World Bank http://data.worldbank.org
- Rich set of data from datahub https://datahub.io/dataset
- Yelp's academic dataset https://www.yelp.com/academic_dataset
- Source of data from GitHub https://github.com/caesar0301/awesome-public-datasets
- Dataset archives from Reddit https://www.reddit.com/r/datasets/

References

- https://hadoop.apache.org/docs/stable/hadoop-mapreduce-client/hadoop-mapreduce-client-core/MapReduceTutorial.html

- Hadoop MapReduce cookbook, Perera, Srinath and Gunarathne, Thilina (2013).

- https://medium.datadriveninvestor.com/the-why-and-how-of-mapreduce-17c3d99fa900

- https://data-flair.training/blogs/hadoop-mapreduce-tutorial/