UE Large Scale Processing @ ESIGELEC
2019/2020

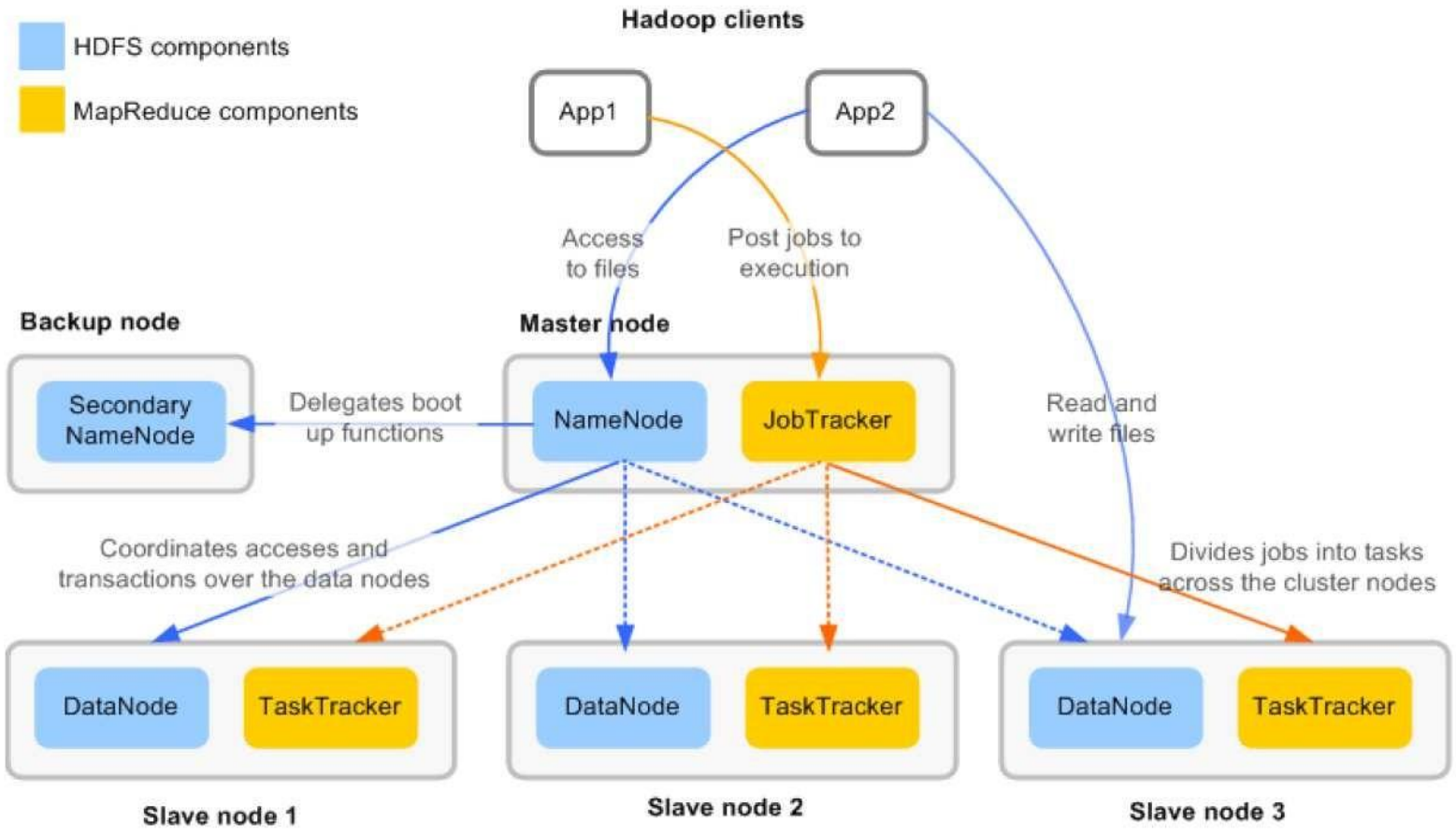
08 – Hadoop MapReduce Explained

Abdel Dadouche
DJZ Consulting

adadouche@hotmail.com
@adadouche

MapReduce V1 Explained

MapReduce V1 Architecture

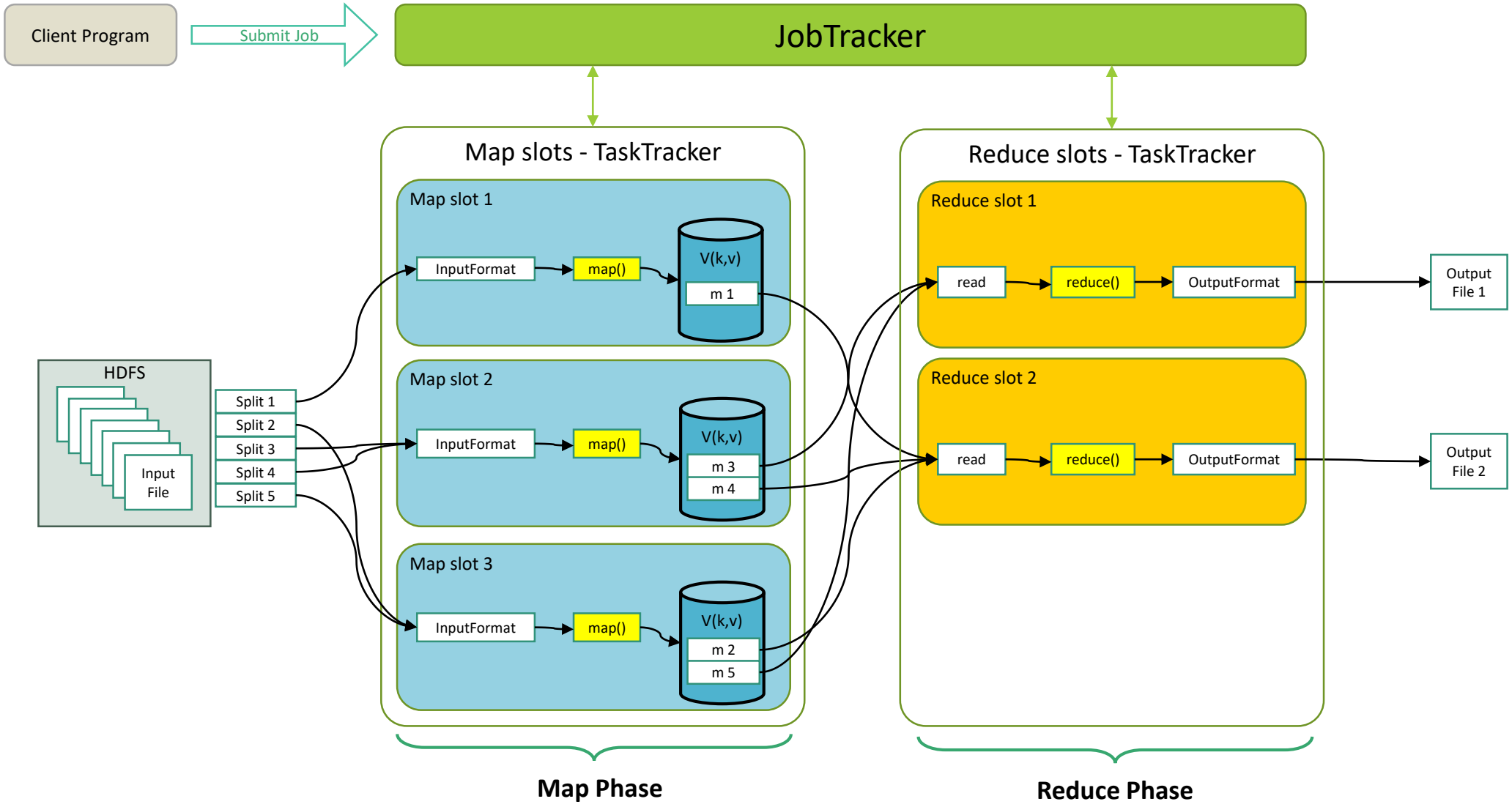


The Job Tracker in Hadoop MR v1

- One per cluster, usually on the master node like the HDFS Name Node
- Receives MapReduce jobs to execute using a JAR file
- Organize and monitor the overall job **tasks** execution (split, map, shuffle, reduce …)
- Based on the input job data (via the HDFS API), the tasks are optimally distributed across the cluster

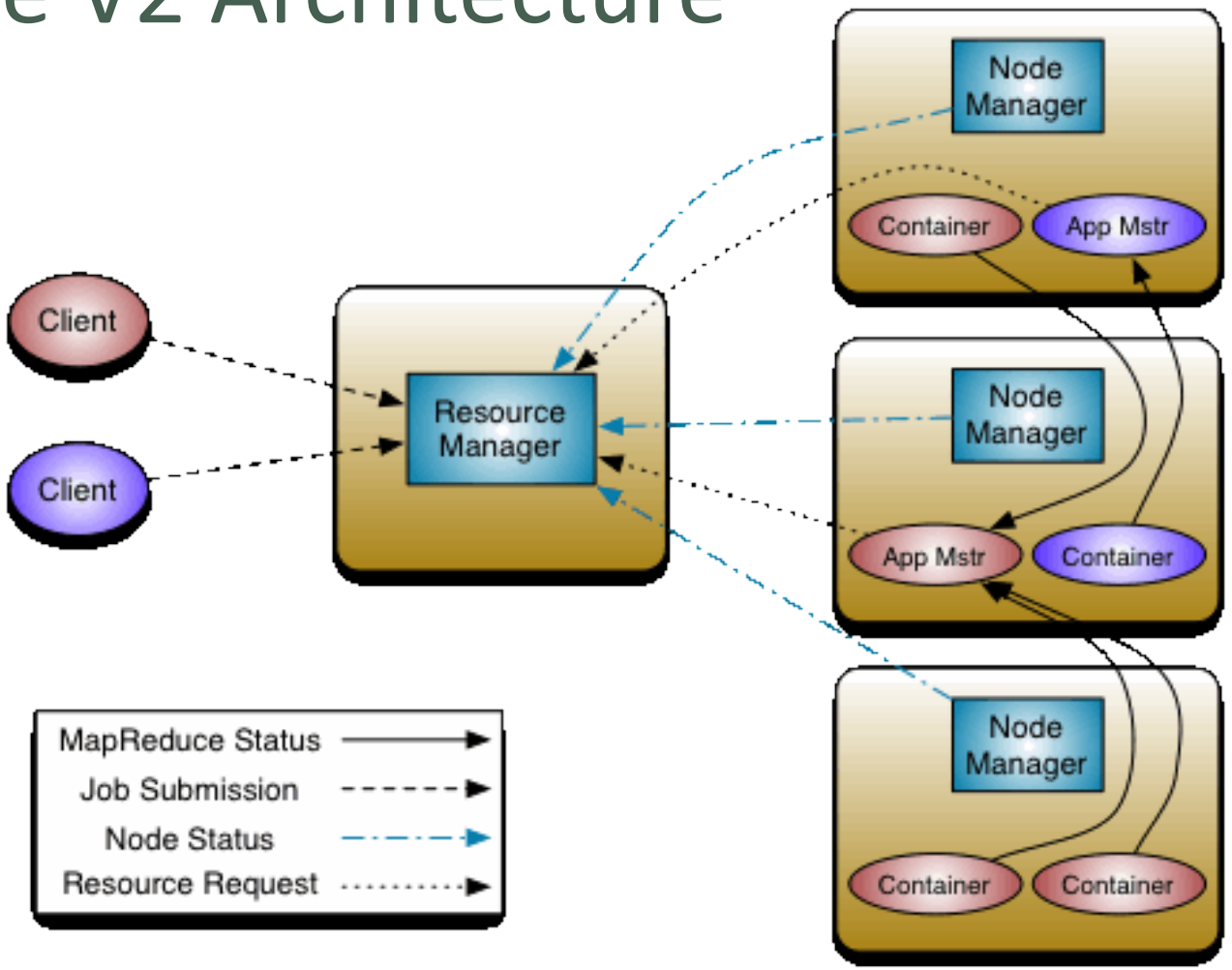
The Task Tracker in Hadoop MR v1

- Multiple, usually one per HDFS Data Node, but can be isolated nodes
- This is where the tasks of the jobs are executed
- Represents a “computation” unit in the cluster
- Each node has a certain number of “Map” and “Reduce” slots
- A Heartbeat status is send sent periodically to the JobTracker


Client Program
Submit Job
JobTracker
Map slots - TaskTracker
Map slot 1
InputFormat
map()
V(k,v)
m 1
Map slot 2
InputFormat
map()
V(k,v)
m 3
m 4
Map slot 3
InputFormat
map()
V(k,v)
m 2
m 5
HDFS
Input File
Split 1
Split 2
Split 3
Split 4
Split 5
Reduce slots - TaskTracker
Reduce slot 1
read
reduce()
OutputFormat
Reduce slot 2
read
reduce()
OutputFormat
Output File 1
Output File 2
Map Phase
Reduce Phase

Hadoop MapReduce v2 Explained

MapReduce V2 Architecture



The in Hadoop MR v2

- Resource Manager (RM)
  - One per cluster, usually on the master node like the HDFS Name Node
  - Orchestrate global resources between nodes
  - Resource arbitrator between application (jobs or not)
- Node Manager (NM)
  - Multiple per cluster, usually on the slave nodes like the HDFS Data Node
  - Report back to the Resource Manager for available resources and health

The in Hadoop MR v2

- Containers
  - Instantiated « on-demand » by the Resource Manager on a slave node with a set of resources
- Application Master
  - One per application / job
  - Runs as a Container but with a more important role
  - Can request the instantiation of multiple containers to run the application tasks (map, reduce or other tasks)

Hadoop MR v1 vs v2 in short

- Separate the Resource & Task Management
- No “slots” concept anymore, but “compute” resoruces available (CPU, RAM etc.)
- Most “JobTracker” features are now in the “Application Master”
- There can be multiple “Application Master” (but only one JT in MRv1)
- Supports both MapReduce and non-MapReduce jobs

MapReduce Input / Output Format

The MapReduce Job Input

- A MapReduce job “map” function usually requires a input constructed based on:
  - A location (for files or directories)
  - A record type (Input Format class)
  - An Input Splits size (the volume for each read operation)
    - The Input Splits size is usually equal to the HDFS block size, but there is no guarantee that the end of a block will correspond to the end of a record

Common Input Format Class

- TextInputFormat (default)
  - Text Files are broken into lines.
  - Either linefeed or carriage-return are used to signal end of line.
  - Keys are the position in the file, and values are the line of text
- KeyValueTextInputFormat
  - Each line is divided into key and value parts by a separator byte (tab by default). If no such a byte exists, the key will be the entire line and value will be empty
- SequenceFileInputFormat
  - Binary files containing serialized key-value pairs (and possibly metadata)
- SequenceFileAsTextInputFormat
  - Indentical to SequenceFileInputFormat but converts the sequence file key-value pair to Text objects using a toString() function

Common Types

- In the Java API, the key-value pair implements specific interfaces:
  - Key : ComparableWritable
  - Value: Writable

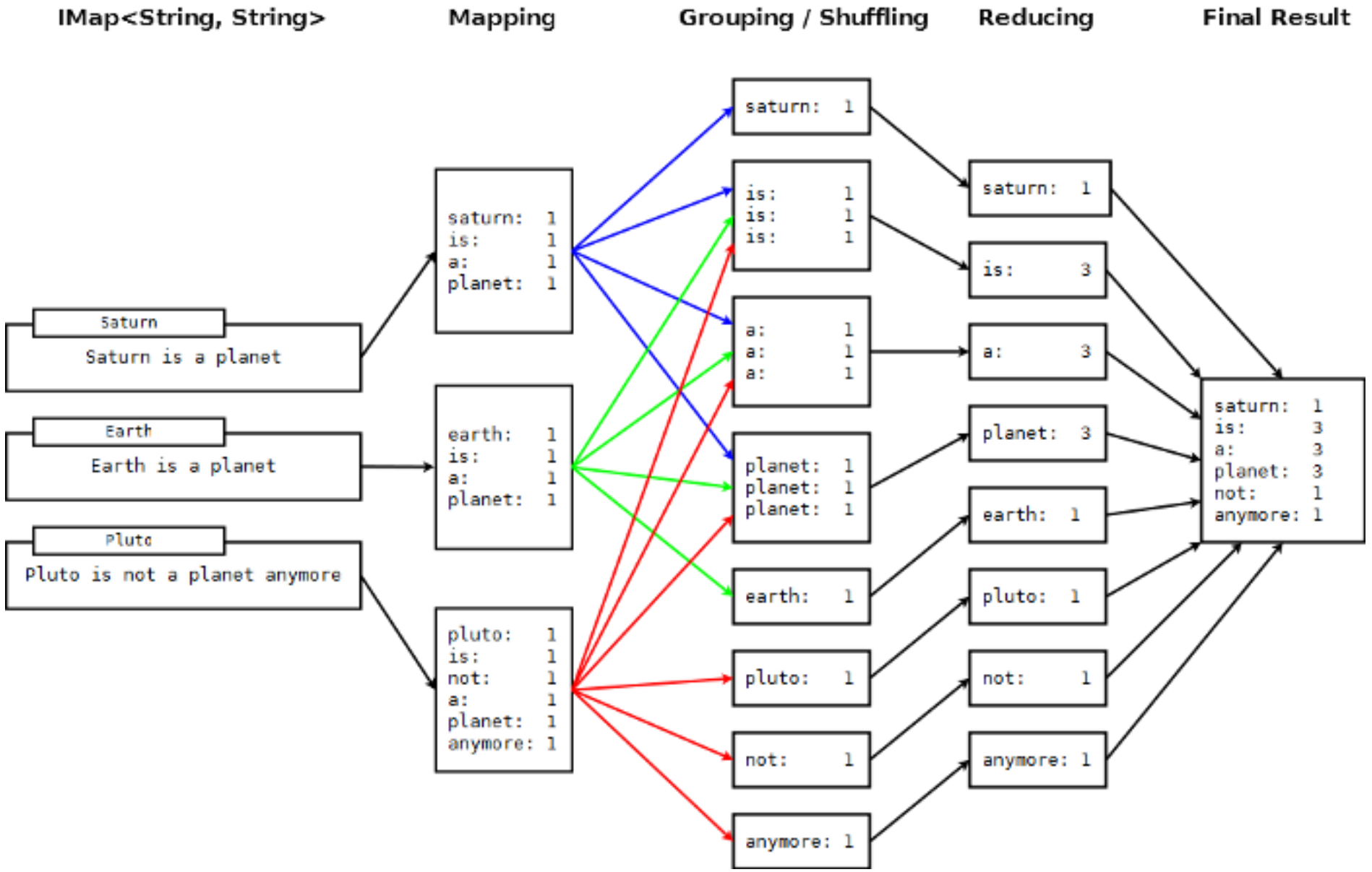
The default types for the key & value are:

- IntWritable
- LongWritable
- FloatWritable
- DoubleWritable
- Text etc..

The Word Count Example Explained

The Word Count example

- A record will correspond to a line from the text file separated by a “\n”
- Split:
  - You can choose to have a split per row or for multiple rows
  - As long as there is no word stored across a split
- Map
  - The key will the word itself. Every time the word will found, the value (“count”) will be incremented
- Shuffle
  - Group by “word” the “count” from each Map
- Reduce
  - Sum the “count” for each group of identical “word” from the Shuffle



Source: https://docs.hazelcast.org/docs/3.2/manual/html/mapreduce-essentials.html

The “Driver” class

- A Java class with a “main”
- Create a Configuration and a Job instance
- Defines the job elements:
  - The input & output format class
  - The key-value pair data types
  - The mapper class
  - The combiner class
  - The reducer class

```
public class WordCount {

  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(TokenizerMapper.class);

    job.setCombinerClass(IntSumReducer.class);

    job.setReducerClass(IntSumReducer.class);

    job.setOutputKeyClass(Text.class);

    job.setOutputValueClass(IntWritable.class);

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

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

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

The “Map” class

- Can be a inner class of the Driver class
- A Java class extending the “`Mapper`” class with the input & output key-value types
- Defines “`map`” function with the input key-value types and a context that stores the map result

```
public static class TokenizerMapper
  extends Mapper<Object, Text, Text, IntWritable> {

  private final static IntWritable
    one = new IntWritable(1);
  private Text word = new Text();

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

The “Reduce” class

- Can be a inner class of the Driver class
- A Java class extending the “`Reducer`” class with the input & output key-value types
- Defines “`map`” function with the input key-value types and a context that stores the reduce result

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

  private IntWritable result = new IntWritable();

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

Remarks

- The combiner (shuffle/sort) class is the same as the reducer one (that’s ok)
- There will be one part-r-XYZ result files per reduce operation where XYZ is an increment
- A _SUCCESS file is also created when the job is completed without any errors

Recap

- Divide and conquer approach
- Only a piece of code
- Be ready to code in Java!
- The architecture has evolved a lot (with YARN) but your MapReduce Job code remains the same