

CPS216: Advanced Database Systems
(Data-intensive Computing Systems)

How MapReduce Works (in Hadoop)

Shivnath Babu

Lifecycle of a MapReduce Job

```
File Edit Options Buffers Tools Java Help
public class WordCount {
    public static class Map extends MapReduceBase implements
        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, OutputCollector<Text, IntWritable>
            output, Reporter reporter) throws IOException {
            String line = value.toString();
            StringTokenizer tokenizer = new StringTokenizer(line);
            while (tokenizer.hasMoreTokens()) {
                word.set(tokenizer.nextToken());
                output.collect(word, one);
            }
        }
    }
    public static class Reduce extends MapReduceBase implements
        Reducer<Text, IntWritable, Text, IntWritable> {
        public void reduce(Text key, Iterator<IntWritable> values, OutputCollector<Text,
            IntWritable> output, Reporter reporter) throws IOException {
            int sum = 0;
            while (values.hasNext()) { sum += values.next().get(); }
            output.collect(key, new IntWritable(sum));
        }
    }
    public static void main(String[] args) throws Exception {
        JobConf conf = new JobConf(WordCount.class);
        conf.setJobName("wordcount");
        conf.setOutputKeyClass(Text.class);
        conf.setOutputValueClass(IntWritable.class);
        conf.setMapperClass(Map.class);
        conf.setCombinerClass(Reduce.class);
        conf.setReducerClass(Reduce.class);
        conf.setInputFormat(TextInputFormat.class);
        conf.setOutputFormat(TextOutputFormat.class);
        FileInputFormat.setInputPaths(conf, new Path(args[0]));
        FileOutputFormat.setOutputPath(conf, new Path(args[1]));
        JobClient.runJob(conf);
    }
}
```

Map function

Reduce function

Run this program as a
MapReduce job

Lifecycle of a MapReduce Job

```
File Edit Options Buffers Tools Java Help
public class WordCount {

    public static class Map extends MapReduceBase implements
        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, OutputCollector<Text, IntWritable>
            output, Reporter reporter) throws IOException {
            String line = value.toString();
            StringTokenizer tokenizer = new StringTokenizer(line);
            while (tokenizer.hasMoreTokens()) {
                word.set(tokenizer.nextToken());
                output.collect(word, one);
            }
        }
    }

    public static class Reduce extends MapReduceBase implements
        Reducer<Text, IntWritable, Text, IntWritable> {
        public void reduce(Text key, Iterator<IntWritable> values, OutputCollector<Text,
            IntWritable> output, Reporter reporter) throws IOException {
            int sum = 0;
            while (values.hasNext()) { sum += values.next().get(); }
            output.collect(key, new IntWritable(sum));
        }
    }

    public static void main(String[] args) throws Exception {
        JobConf conf = new JobConf(WordCount.class);
        conf.setJobName("wordcount");
        conf.setOutputKeyClass(Text.class);
        conf.setOutputValueClass(IntWritable.class);
        conf.setMapperClass(Map.class);
        conf.setCombinerClass(Reduce.class);
        conf.setReducerClass(Reduce.class);
        conf.setInputFormat(TextInputFormat.class);
        conf.setOutputFormat(TextOutputFormat.class);
        FileInputFormat.setInputPaths(conf, new Path(args[0]));
        FileOutputFormat.setOutputPath(conf, new Path(args[1]));

        JobClient.runJob(conf);
    }
}

----- mapreduce.java All L9 (Java/l Abbrev) -----
Wrote /home/shivnath/Desktop/mapreduce.java
```

Map function

Reduce function

Run this program as a
MapReduce job

Lifecycle of a MapReduce Job

Time

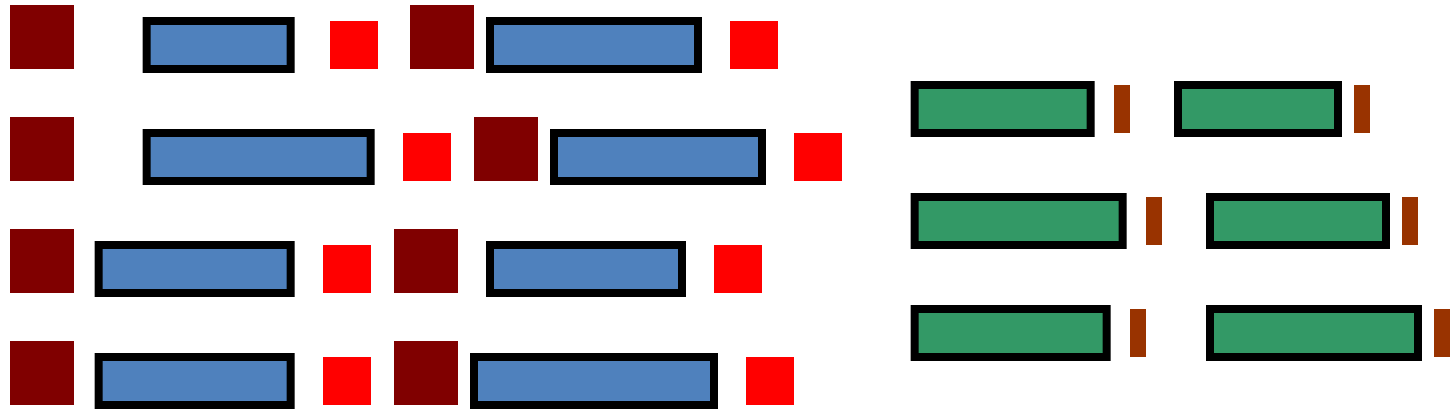
```
File Edit Options Window Help Java IDE
public class WordCount {
    public static class MyMapper implements org.apache.hadoop.mapreduce.Mapper {
        private final org.apache.hadoop.conf.Configuration conf;
        private Text src = new Text();

        public void map(org.apache.hadoop.io.Text value, org.apache.hadoop.mapreduce.Mapper.Context context)
            throws IOException, InterruptedException {
            String line = value.toString();
            StringTokenizer tokenizer = new StringTokenizer(line);
            while (tokenizer.hasMoreTokens()) {
                word = tokenizer.nextToken();
                output.collect(word, one);
            }
        }
    }

    public static class MyReducer implements org.apache.hadoop.mapreduce.Reducer {
        private final org.apache.hadoop.conf.Configuration conf;
        private org.apache.hadoop.io.Text value;
        private org.apache.hadoop.io.Text key;

        public void reduce(org.apache.hadoop.io.Text key, org.apache.hadoop.mapreduce.Reducer.Context context)
            throws IOException, InterruptedException {
            int sum = 0;
            while (value.hasMoreTokens()) {
                output.collect(key, sum);
            }
        }
    }

    public static void main(String[] args) throws Exception {
        Job job = new Job(conf, "WordCount");
        job.setJarByClass(WordCount.class);
        job.setMapperClass(MyMapper.class);
        job.setReducerClass(MyReducer.class);
        conf.setBoolean("mapreduce.job.reduce.shuffle.parallelcopies", 10);
        FileInputFormat.setInputPaths(job, new Path("input"));
        FileOutputFormat.setOutputPath(job, new Path("output"));
        job.waitForCompletion(true);
    }
}
WordCount.java 88:13 [OK] [Main] [Main]
C:\Users\james\Desktop\WordCount\WordCount.java
```



Input Splits

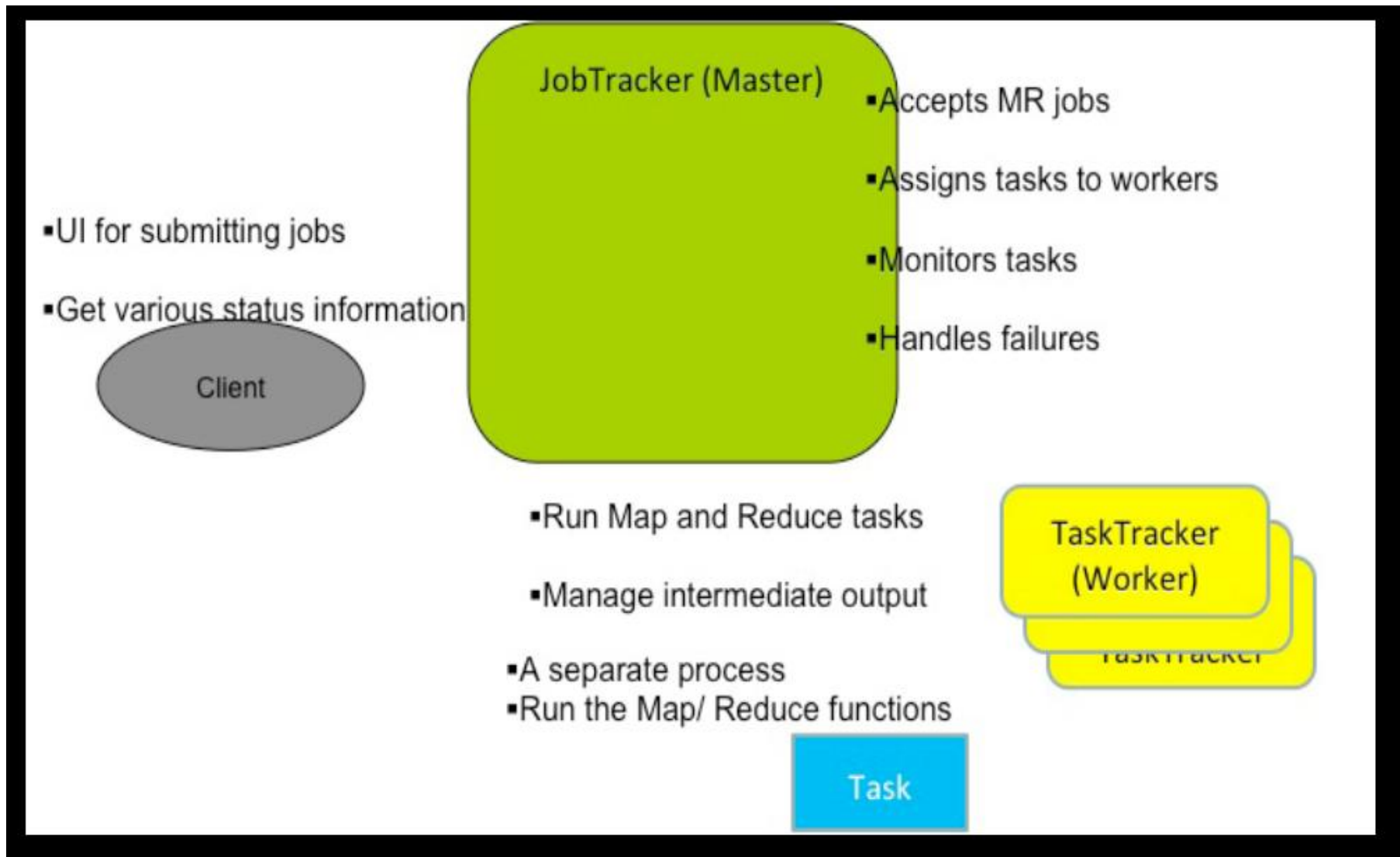
Map Wave 1

Map Wave 2

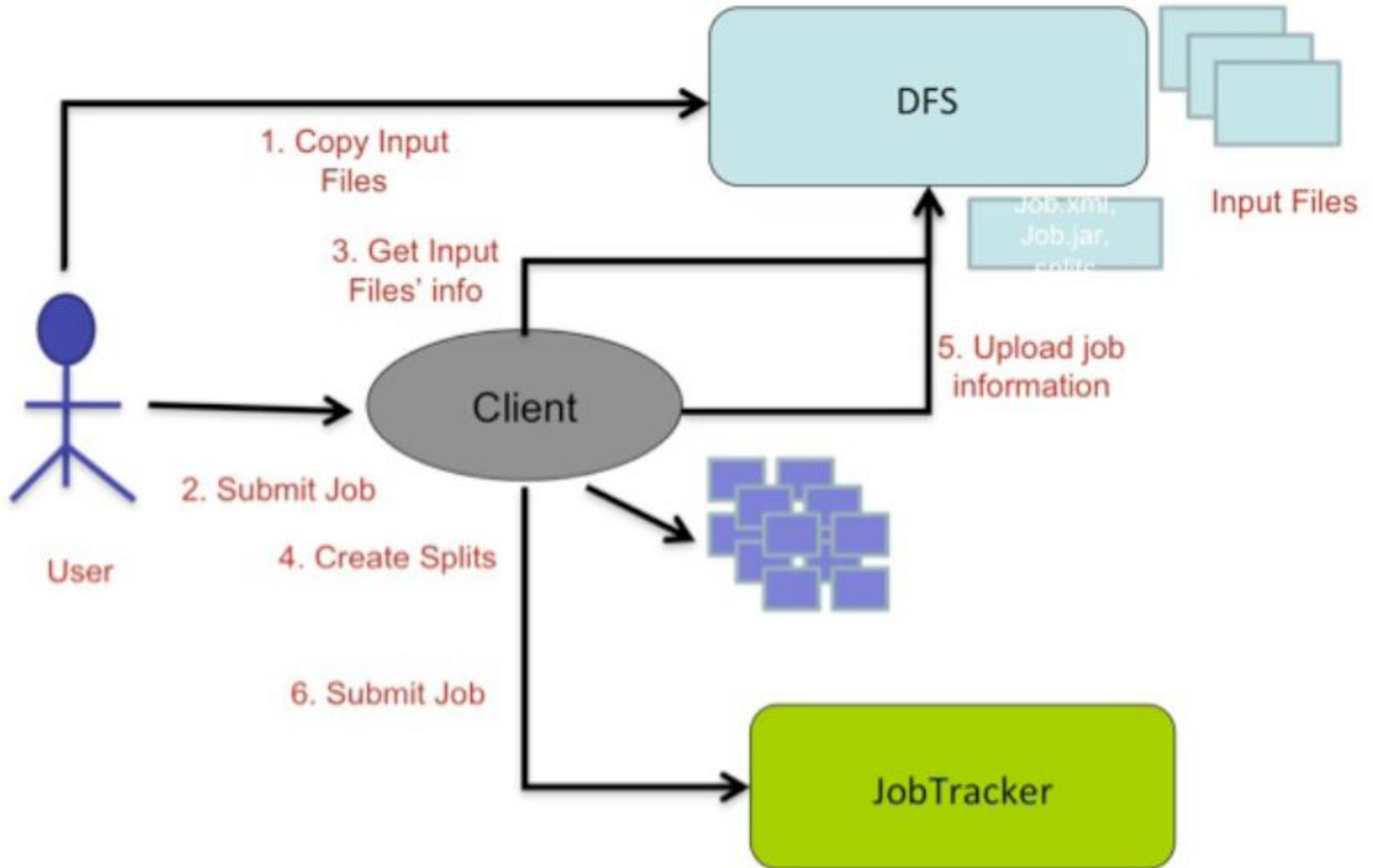
Reduce Wave 1

Reduce Wave 2

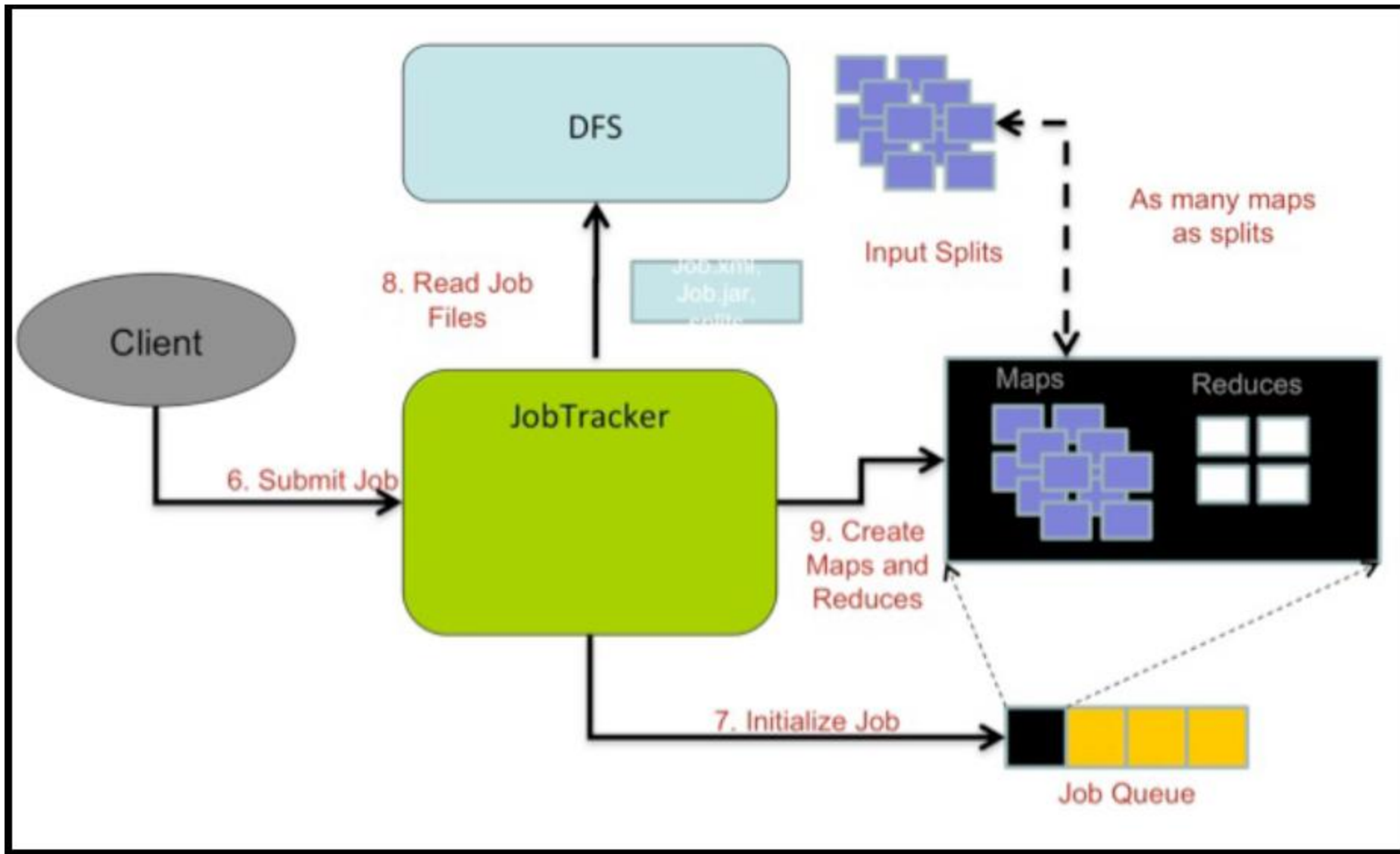
Components in a Hadoop MR Workflow



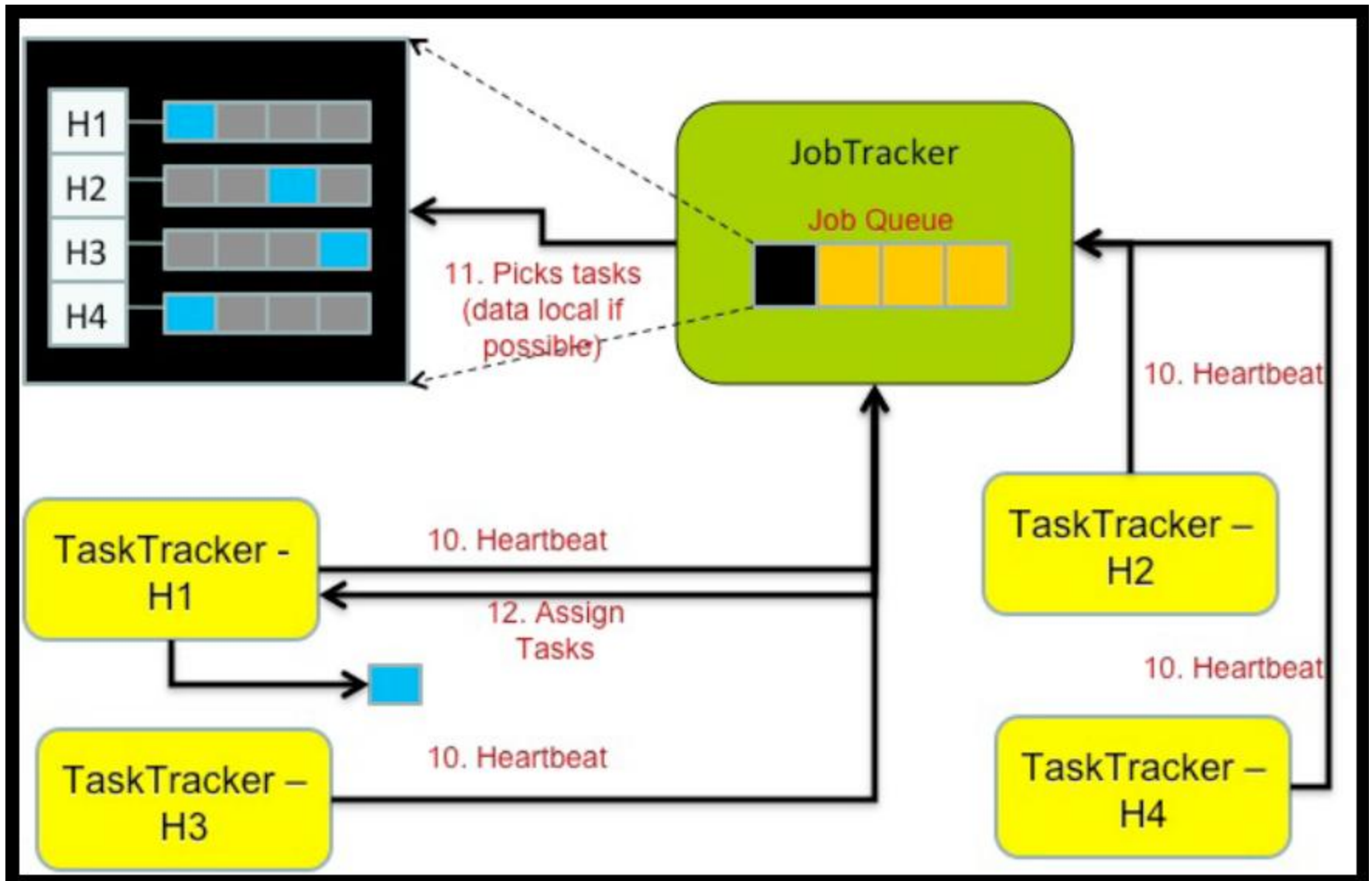
Job Submission



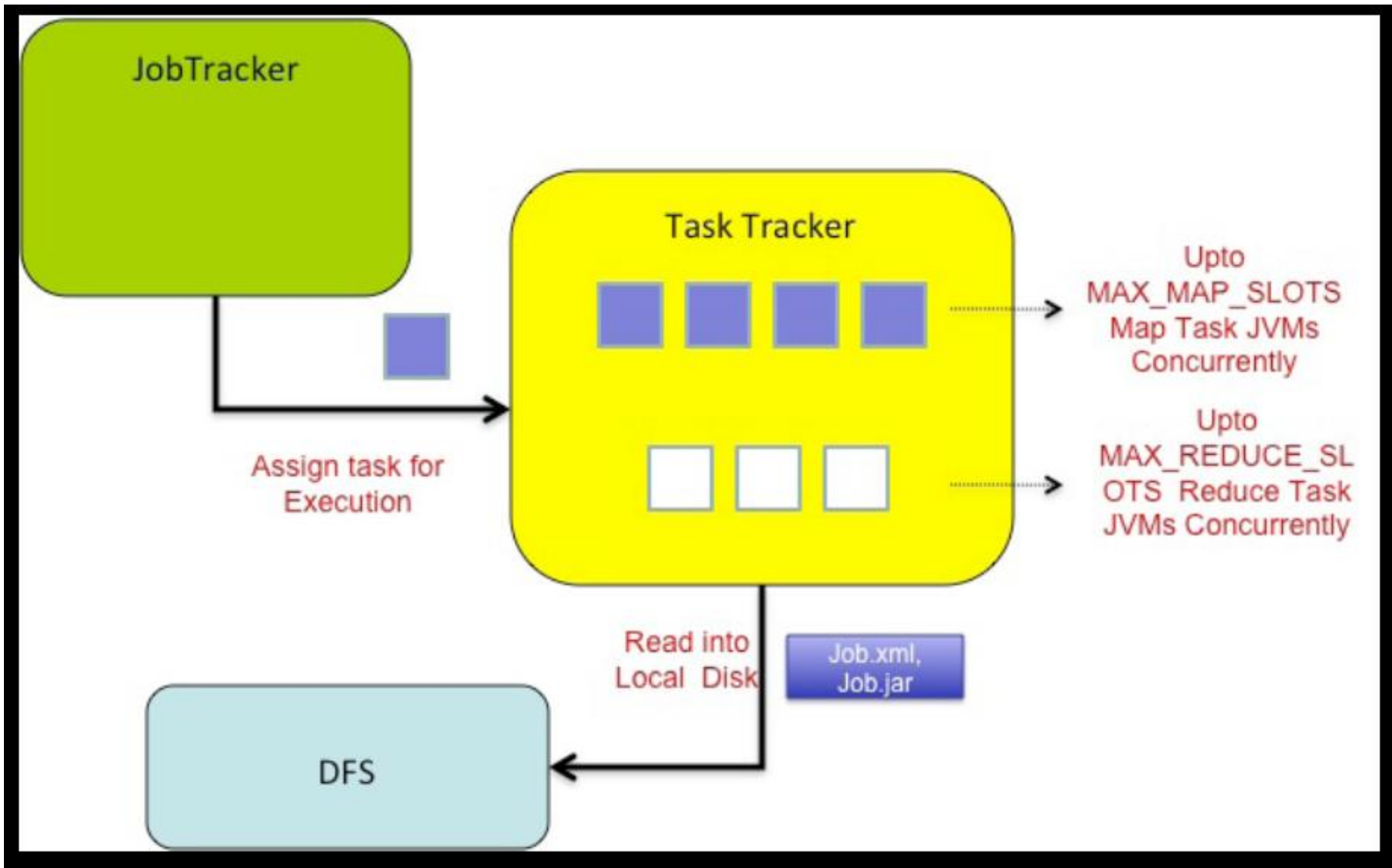
Initialization



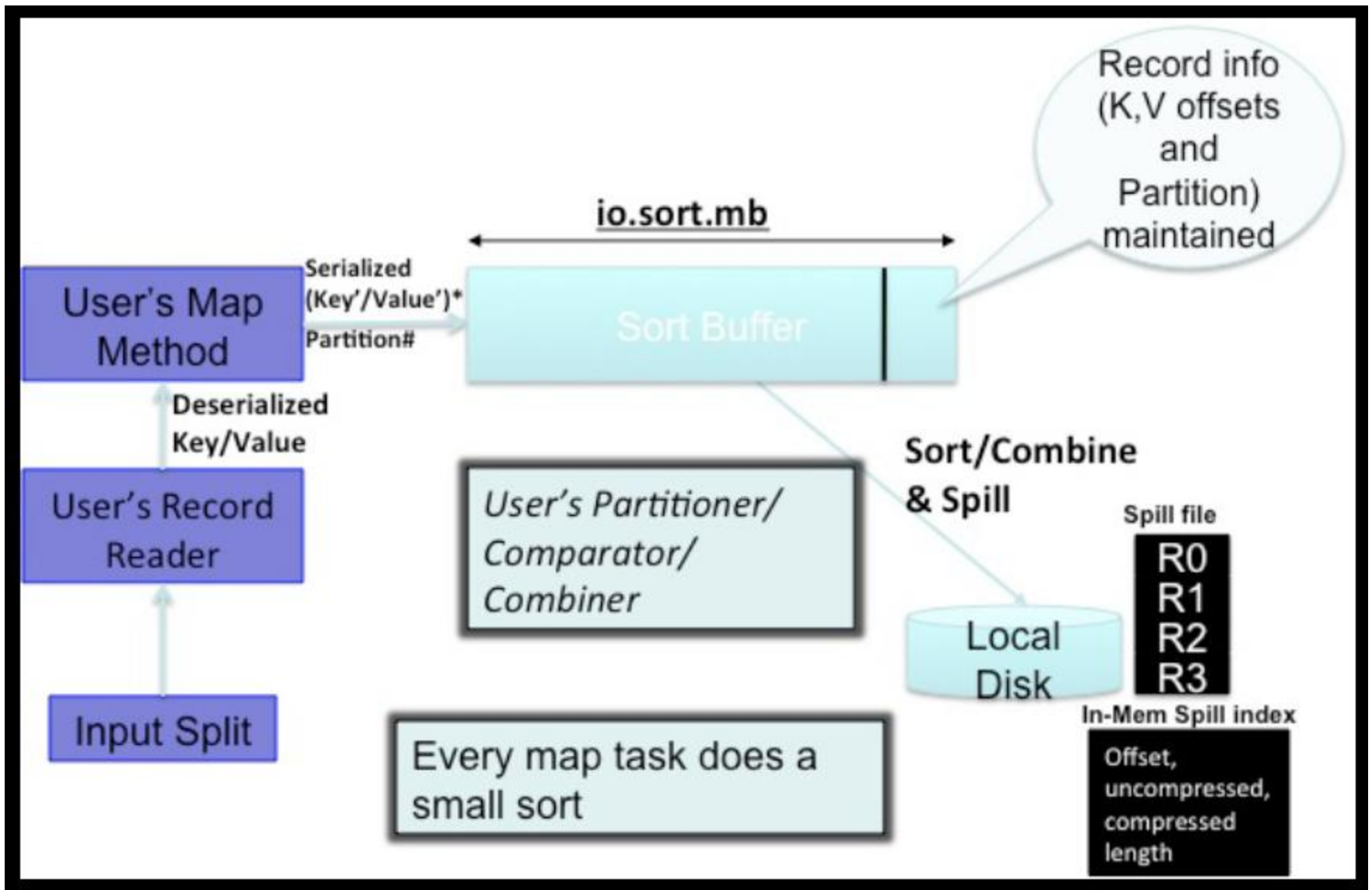
Scheduling



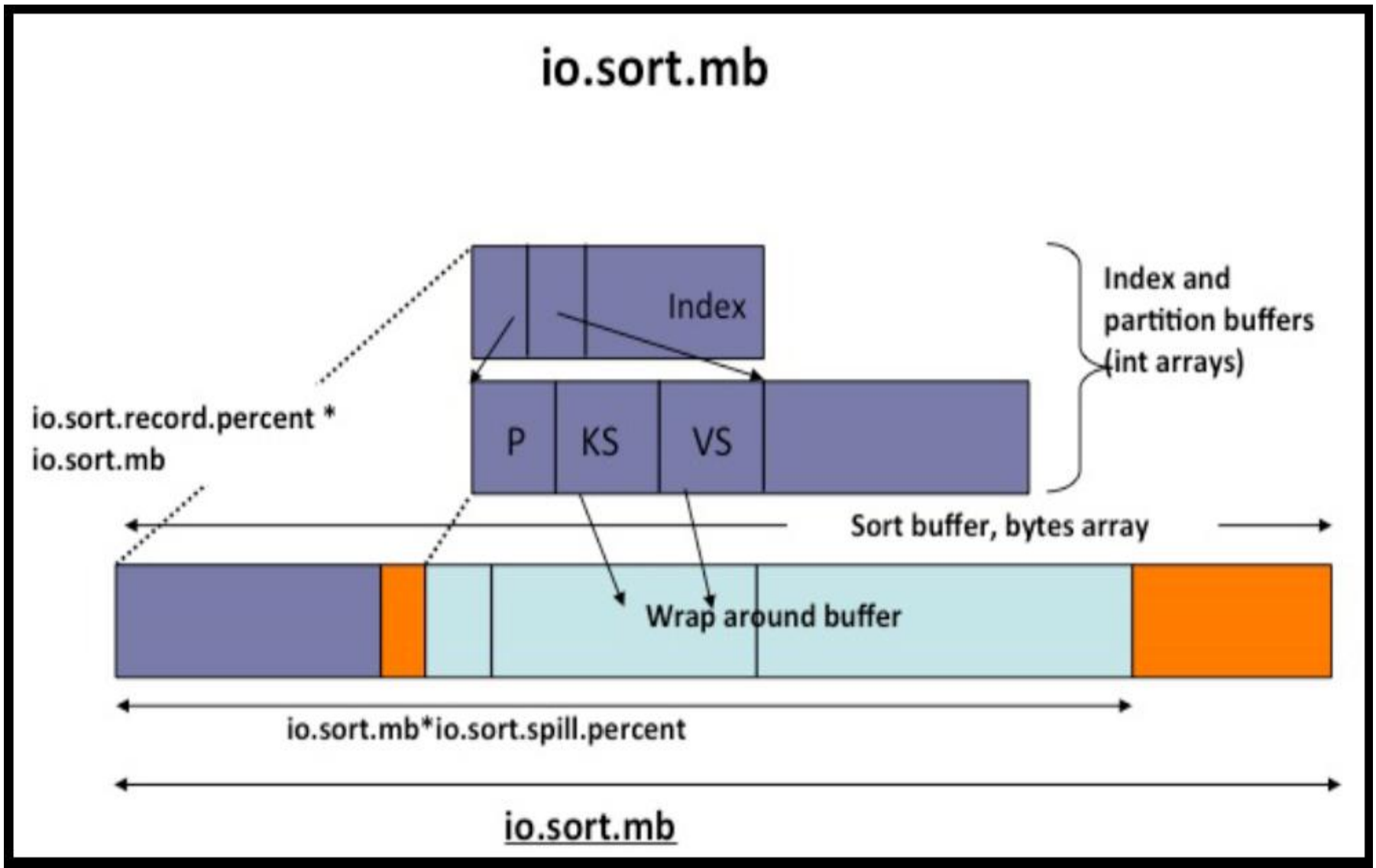
Execution



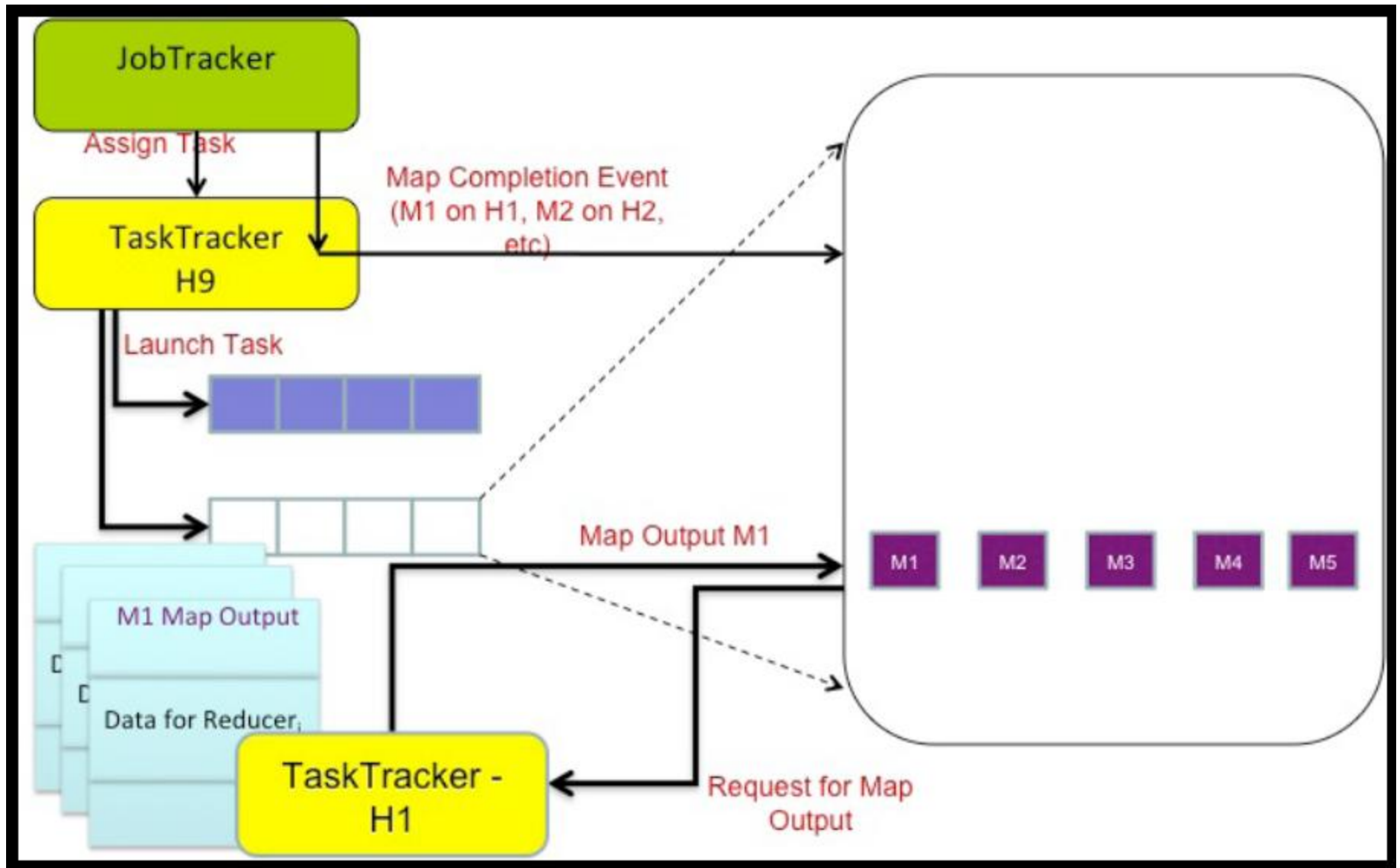
Map Task



Sort Buffer



Reduce Tasks



Quick Overview of Other Topics (Will Revisit Them Later in the Course)

- Dealing with failures
- Hadoop Distributed FileSystem (HDFS)
- Optimizing a MapReduce job

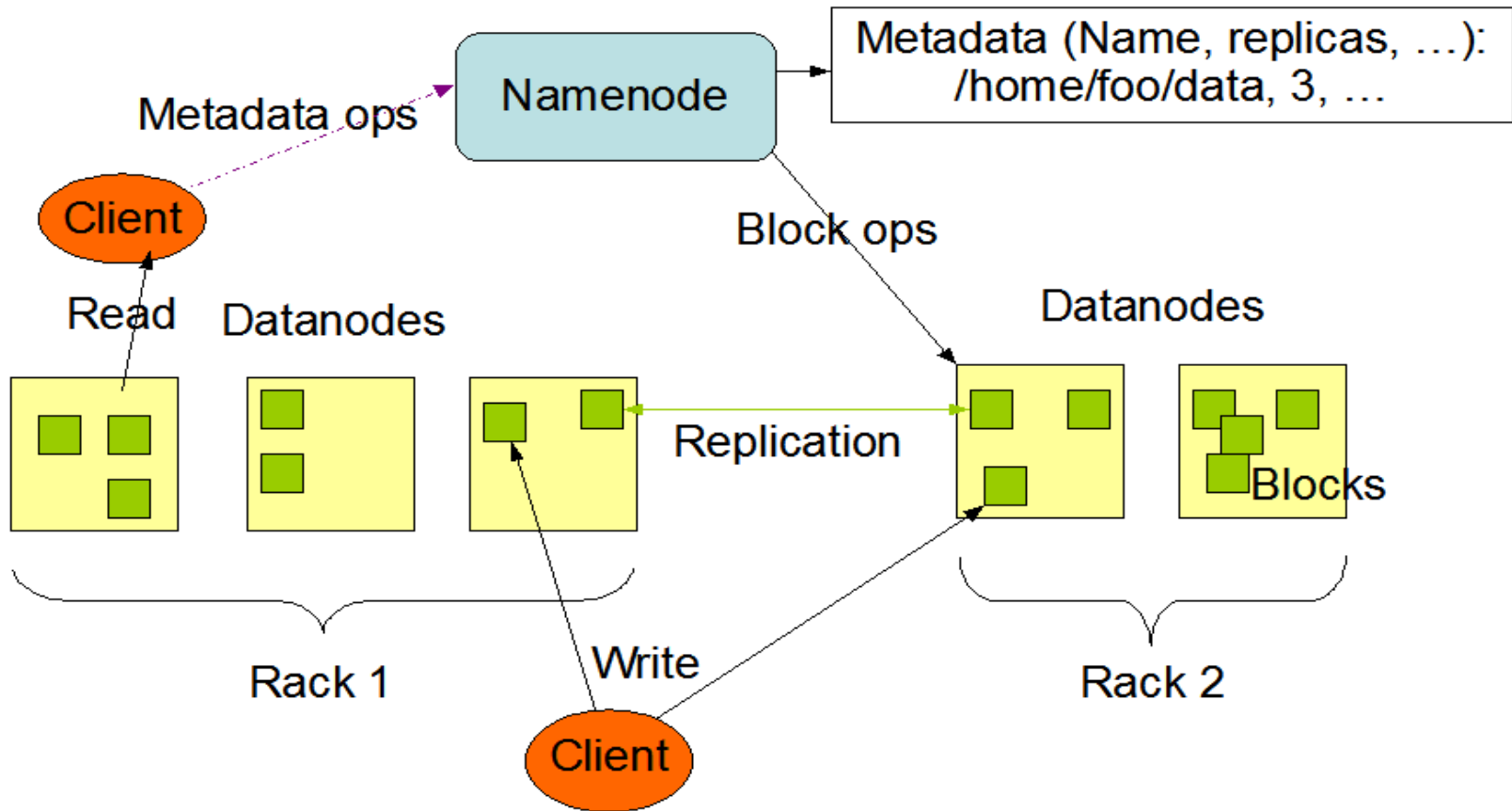
Dealing with Failures and Slow Tasks

- What to do when a task fails?
 - Try again (retries possible because of **idempotence**)
 - Try again somewhere else
 - Report failure
- What about slow tasks: stragglers
 - Run another version of the same task in parallel. Take results from the one that finishes first
 - What are the pros and cons of this approach?

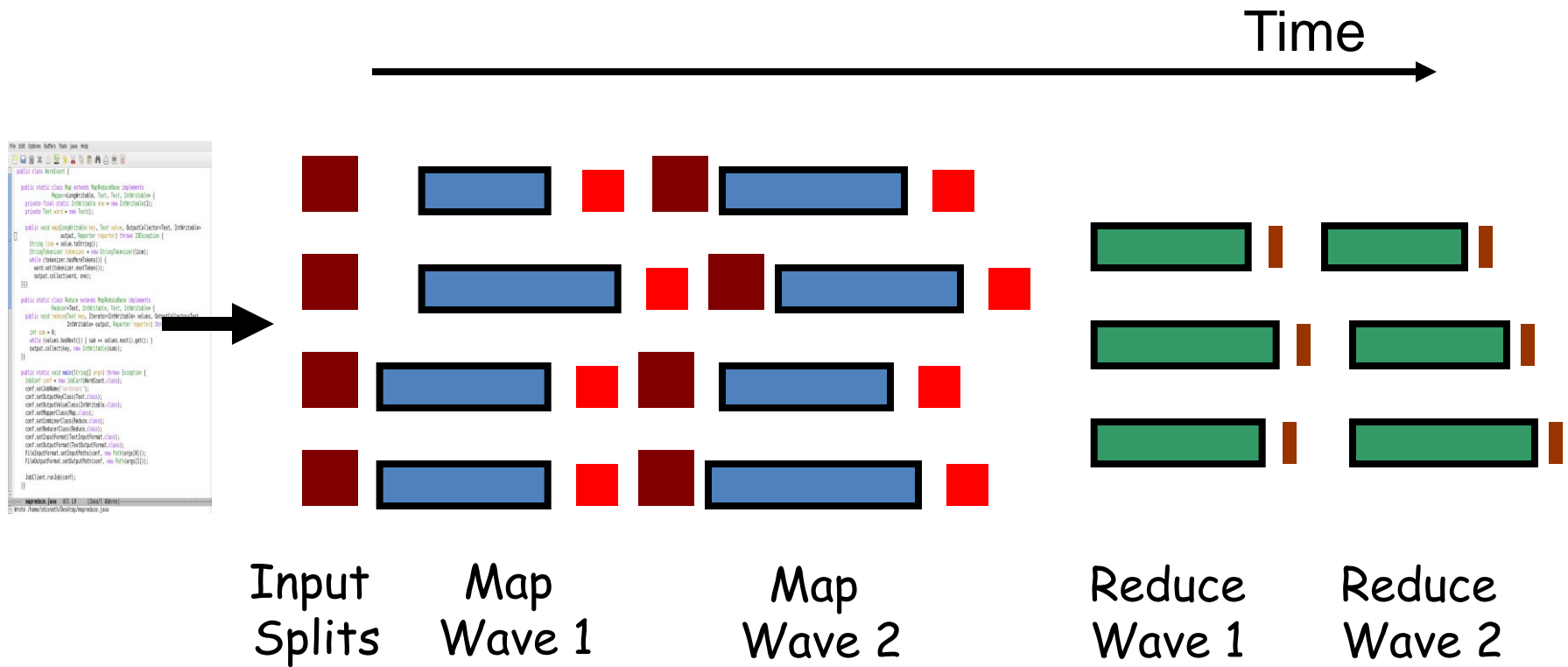
**Fault tolerance is of
high priority in the
MapReduce framework**

HDFS Architecture

HDFS Architecture

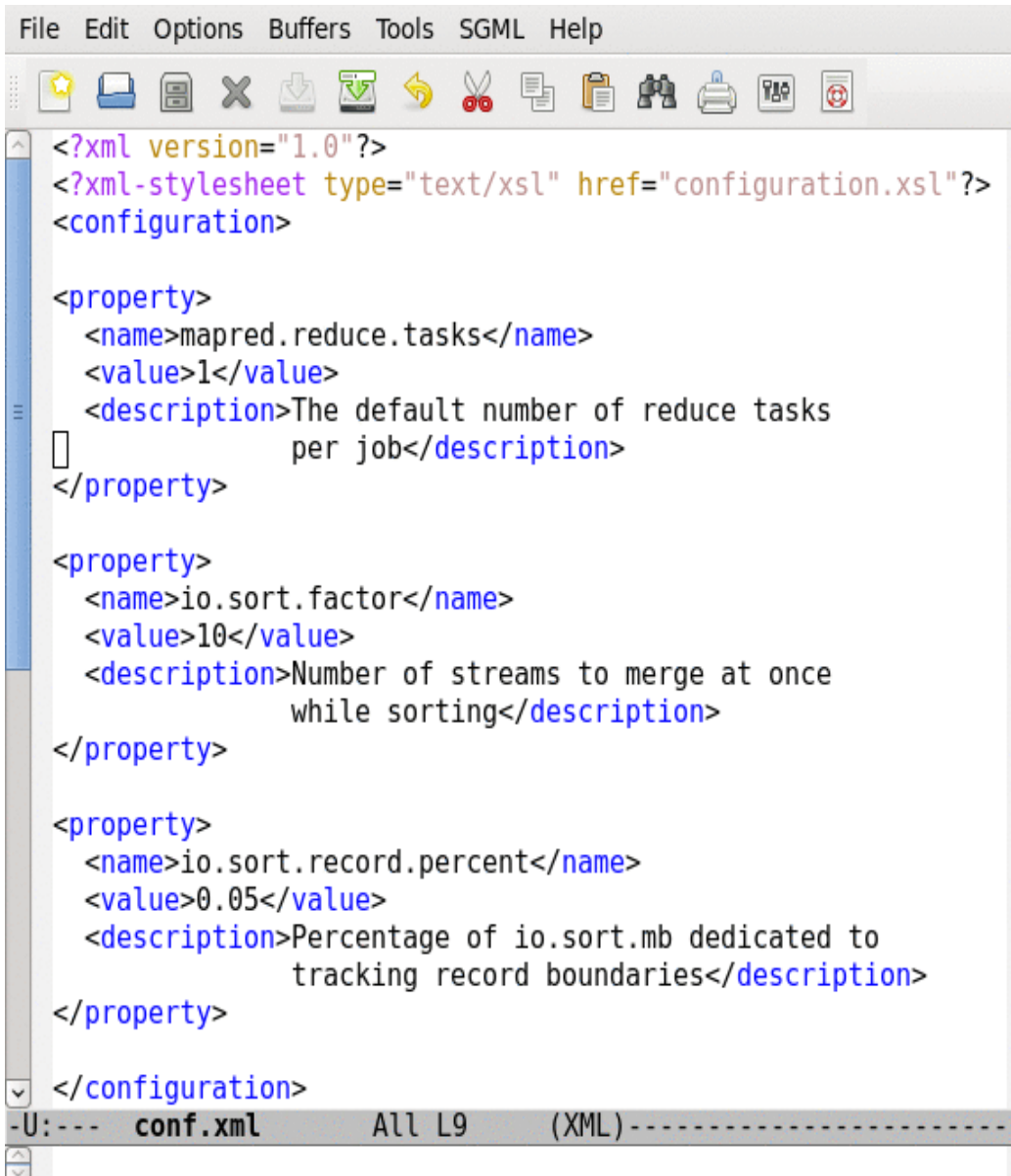


Lifecycle of a MapReduce Job



How are the number of splits, number of map and reduce tasks, memory allocation to tasks, etc., determined?

Job Configuration Parameters



The image shows a screenshot of an XML editor window. The title bar includes 'File Edit Options Buffers Tools SGML Help'. The toolbar contains icons for various actions like opening, saving, and printing. The main text area displays XML code for a Hadoop configuration file. The code defines three properties: 'mapred.reduce.tasks' with a value of 1, 'io.sort.factor' with a value of 10, and 'io.sort.record.percent' with a value of 0.05. The status bar at the bottom shows '-U:--- conf.xml All L9 (XML)-----'.

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="configuration.xsl"?>
<configuration>

<property>
  <name>mapred.reduce.tasks</name>
  <value>1</value>
  <description>The default number of reduce tasks
    per job</description>
</property>

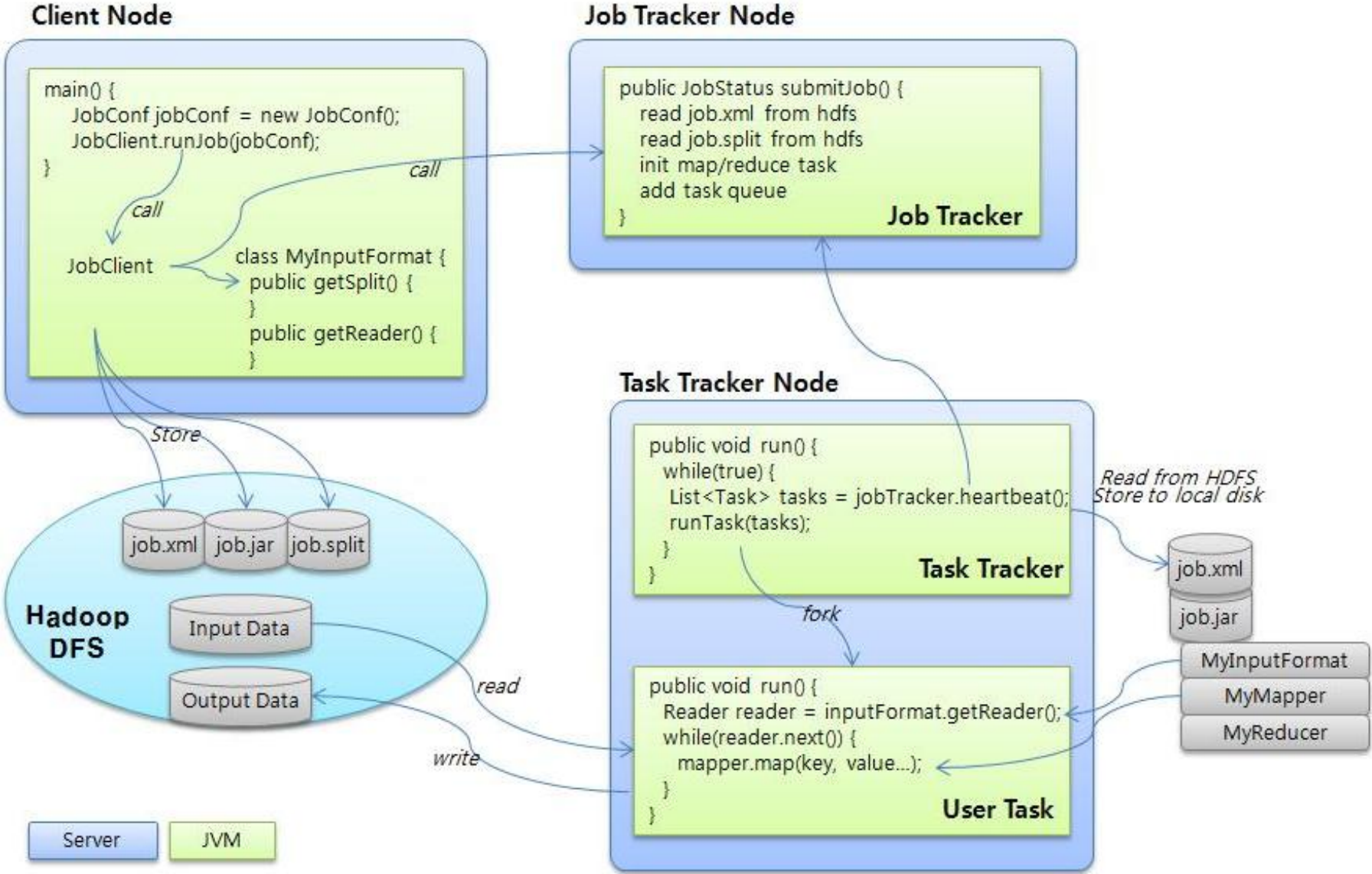
<property>
  <name>io.sort.factor</name>
  <value>10</value>
  <description>Number of streams to merge at once
    while sorting</description>
</property>

<property>
  <name>io.sort.record.percent</name>
  <value>0.05</value>
  <description>Percentage of io.sort.mb dedicated to
    tracking record boundaries</description>
</property>

</configuration>
```

- 190+ parameters in Hadoop
- Set manually or defaults are used

Hadoop Job Configuration Parameters



Tuning Hadoop Job Conf. Parameters

- Do their settings impact performance?
- What are ways to set these parameters?
 - Defaults -- are they good enough?
 - Best practices -- the best setting can depend on data, job, and cluster properties
 - Automatic setting

Experimental Setting

- Hadoop cluster on 1 master + 16 workers
- Each node:
 - 2GHz AMD processor, 1.8GB RAM, 30GB local disk
 - Relatively ill-provisioned!
 - Xen VM running Debian Linux
 - Max 4 concurrent maps & 2 reduces
 - Maximum map wave size = $16 \times 4 = 64$
 - Maximum reduce wave size = $16 \times 2 = 32$
- Not all users can run large Hadoop clusters:
 - Can Hadoop be made competitive in the 10-25 node, multi GB to TB data size range?

Parameters Varied in Experiments

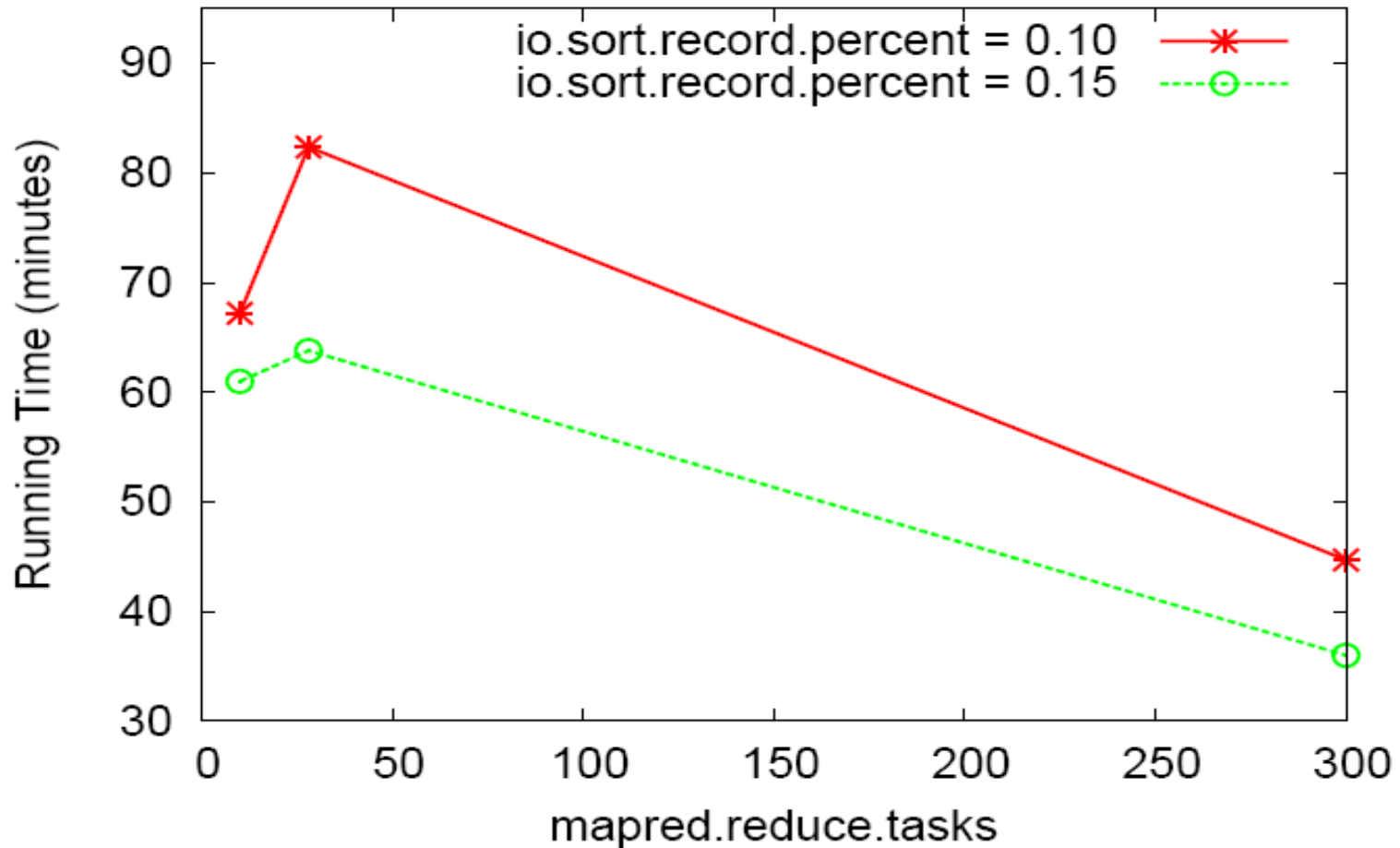
Parameter Name	Description and Use	Default Value	Values Considered
mapred.reduce.tasks	Number of reducer tasks	1	[5,300]
io.sort.factor	Number of sorted streams to merge at once during sorting	10	[10,500]
io.sort.mb	Size in MegaBytes of map-side buffer for sorting	100	[100,200]
io.sort.record.percent	Fraction of io.sort.mb dedicated to metadata storage	0.05	[0.05,0.15]
io.file.buffer.size	Buffer size used to read/write (intermediate) sequence files	4K	32K
mapred.child.java.opts	Java control options for all mapper and reducer tasks	-Xmx200m	-Xmx[200m,300m]
mapred.inmem.merge.threshold	Reduce-side trigger for in-memory merging; off when 0	1000	0
mapred.job.shuffle.input.buffer.percent	% of reducer task's heap to buffer map outputs	0.7	{0.7,0.8}
mapred.job.shuffle.merge.percent	Usage threshold of mapred.job.shuffle.input.buffer.percent to trigger reduce-side merge in parallel with the copying of map outputs	0.66	{0.66,0.8}
mapred.job.reduce.input.buffer.percent	% of reducer task's heap to buffer map outputs while applying reduce	0	{0,0.8}
dfs.replication	Block replication factor in Hadoop's HDFS filesystem	3	2
dfs.block.size	HDFS block size (equal to amount of data processed per mapper task)	64MB	128MB

Hadoop 50GB TeraSort

Row#	mapred. reduce.tasks	io.sort. factor	io.sort.record. percent	Job Running Time
1	10	10	0.10	1hrs, 25mins, 25sec
2	10	10	0.15	1hrs, 14mins, 54sec
3	10	500	0.10	1hrs, 7mins, 11sec
4	10	500	0.15	1hrs, 1mins, 1sec
5	28	10	0.10	1hrs, 22mins, 54sec
6	28	10	0.15	1hrs, 4mins, 57sec
7	28	500	0.10	1hrs, 22mins, 24sec
8	28	500	0.15	1hrs, 3mins, 46sec
9	300	10	0.10	45mins, 22sec
10	300	10	0.15	35mins, 9sec
11	300	500	0.10	44mins, 38sec
12	300	500	0.15	35mins, 56sec

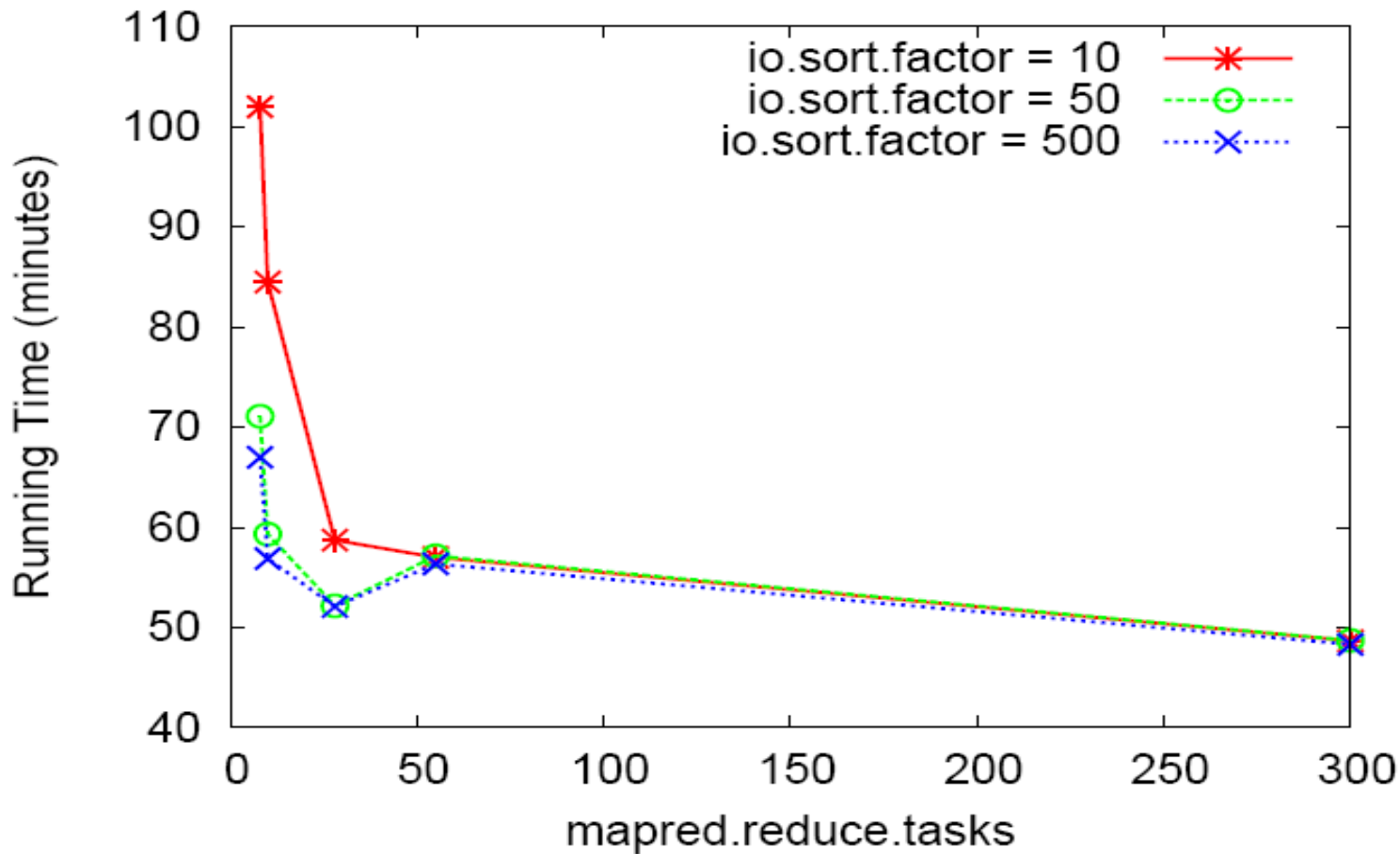
- Varying number of reduce tasks, number of concurrent sorted streams for merging, and fraction of map-side sort buffer devoted to metadata storage

Hadoop 50GB TeraSort



- Varying number of reduce tasks for different values of the fraction of map-side sort buffer devoted to metadata storage (with `io.sort.factor = 500`)

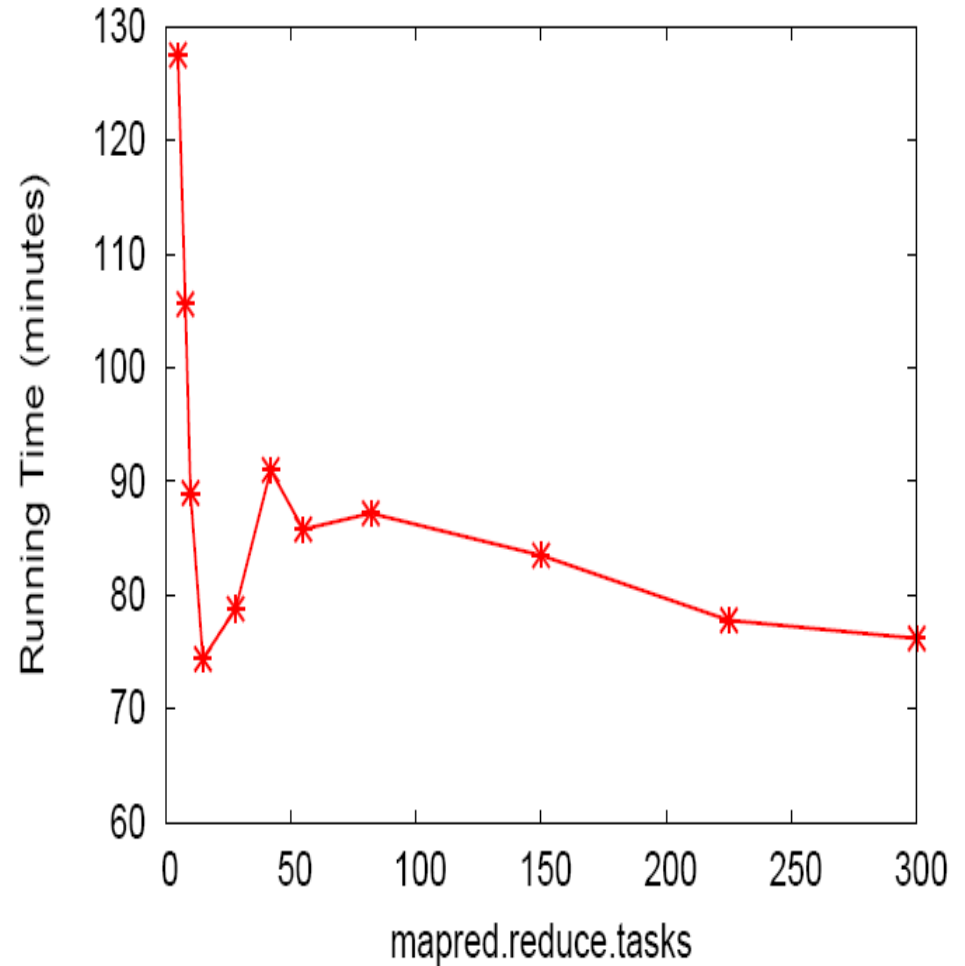
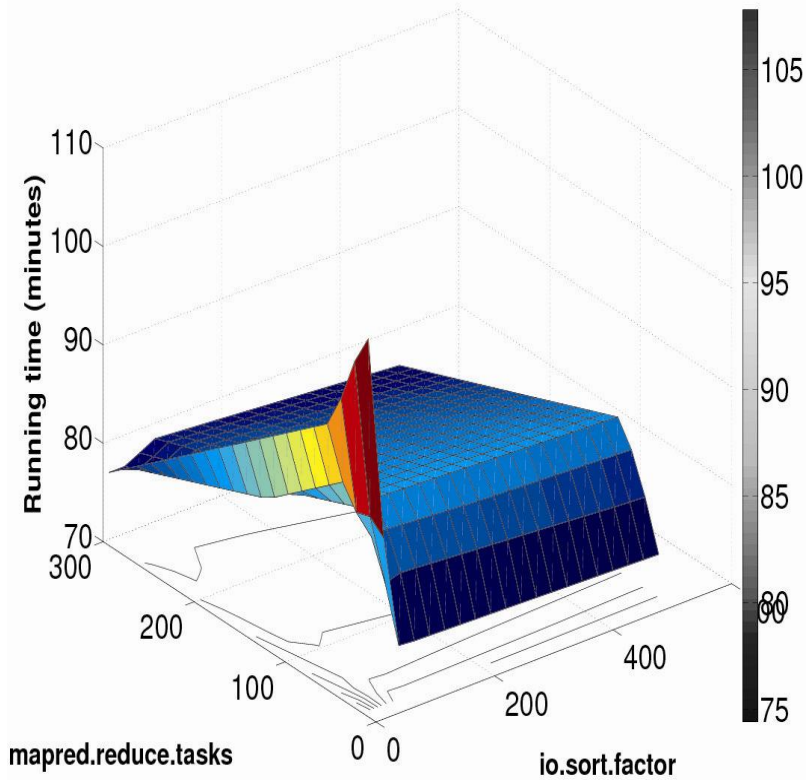
Hadoop 50GB TeraSort



- Varying number of reduce tasks for different values of `io.sort.factor` (`io.sort.record.percent = 0.05`, default)

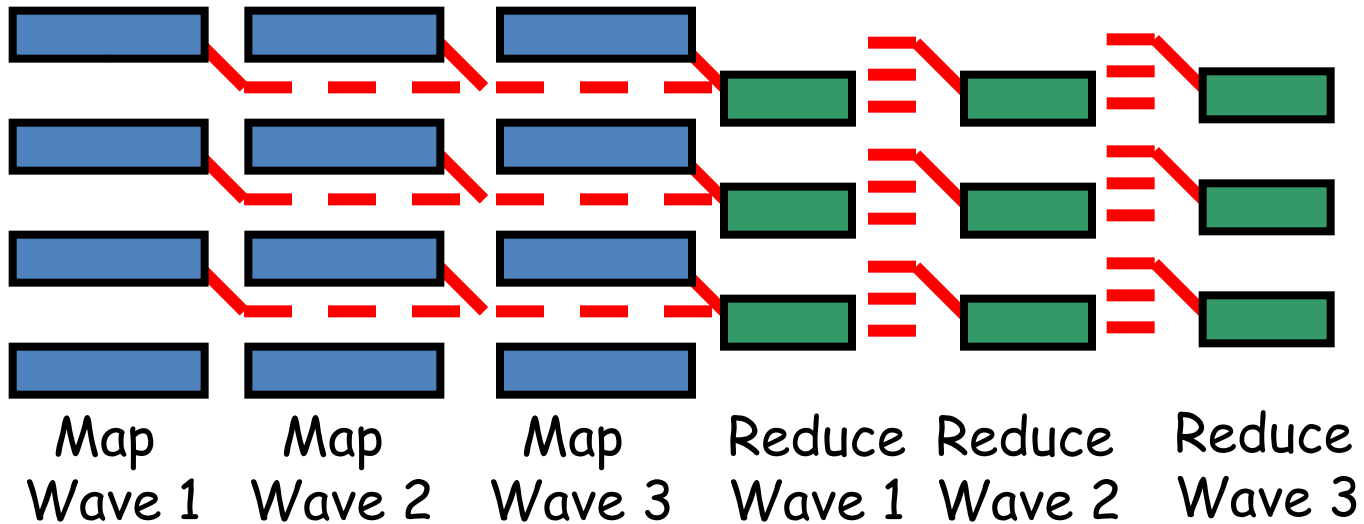
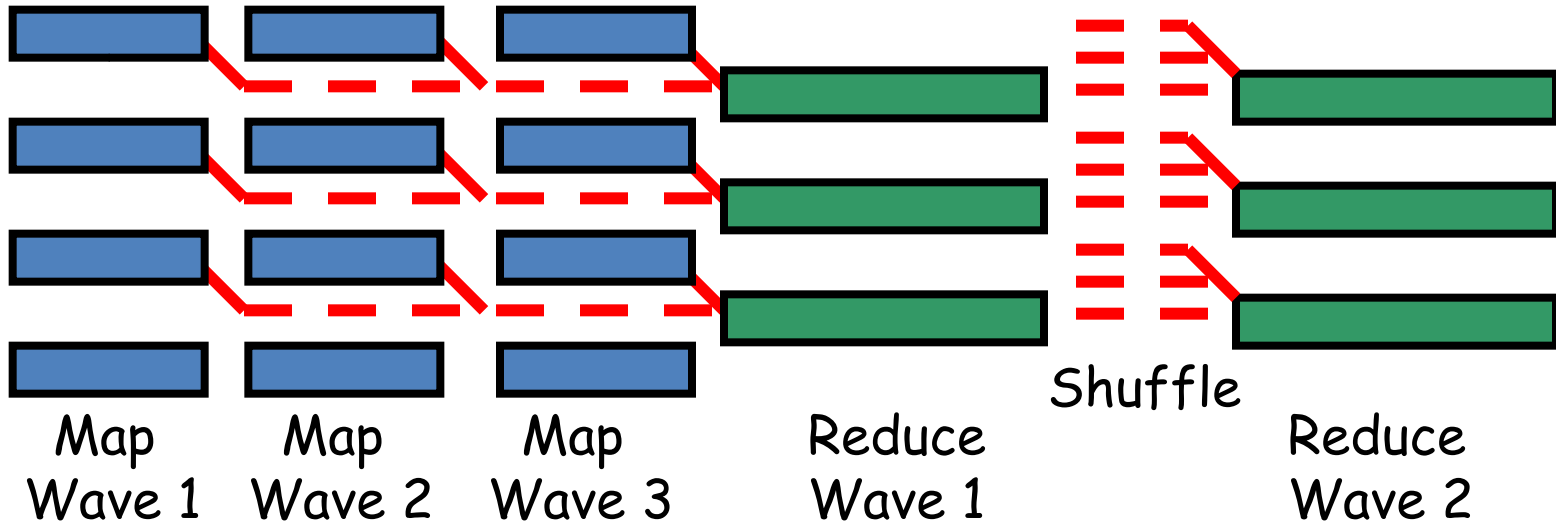
Hadoop 75GB TeraSort

75GB TeraSort in Hadoop



- 1D projection for io.sort.factor=500

Automatic Optimization? (Not yet in Hadoop)



What if #reduces increased to 9?