Data Engineering

How MapReduce Works

Shivnath Babu

Lifecycle of a MapReduce Job

File Edit Options Buffers Tools Java Help

```java
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

--:--- mapreduce.java All L9 (Java/l Abbrev)----------

Wrote /home/shivnath/Desktop/mapreduce.java

Lifecycle of a MapReduce Job

File Edit Options Buffers Tools Java Help

```java
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)----------

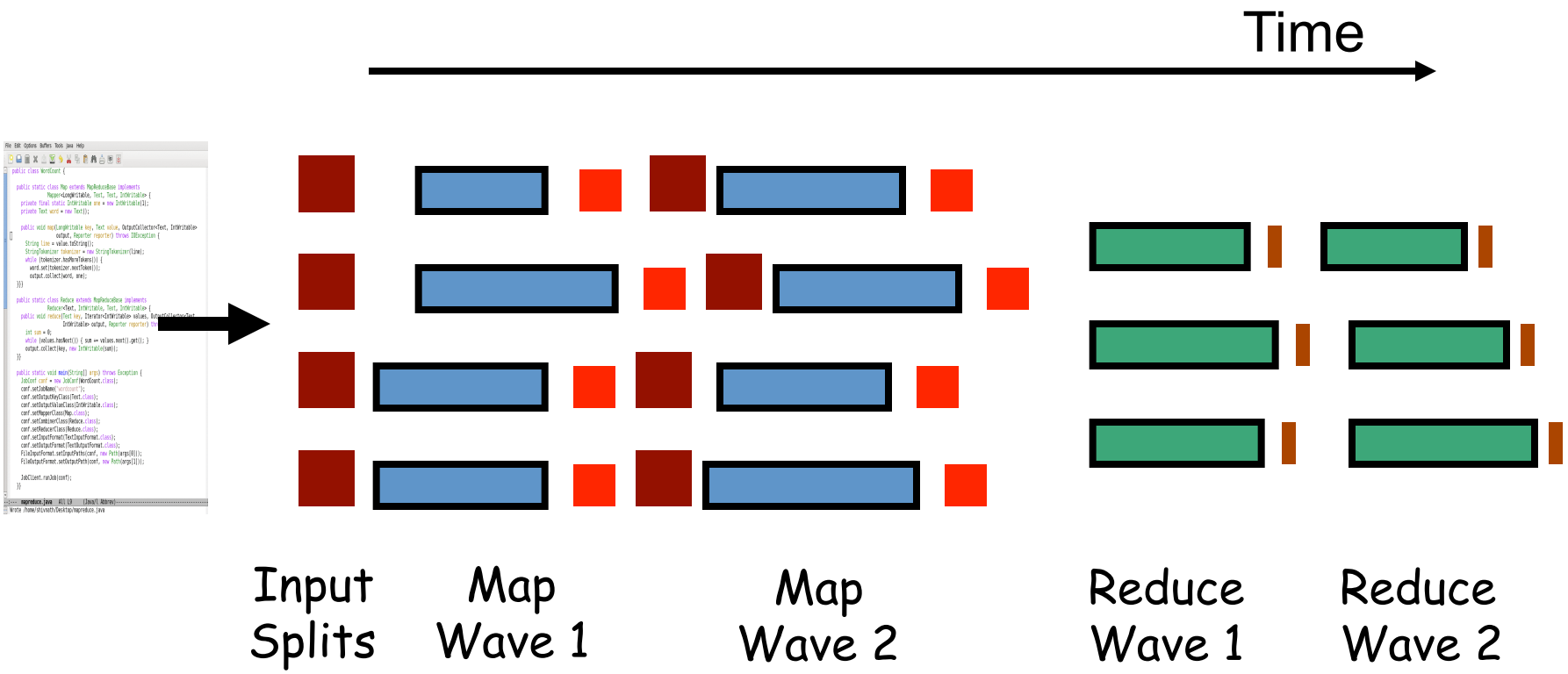
Wrote /home/shivnath/Desktop/mapreduce.java

Map function

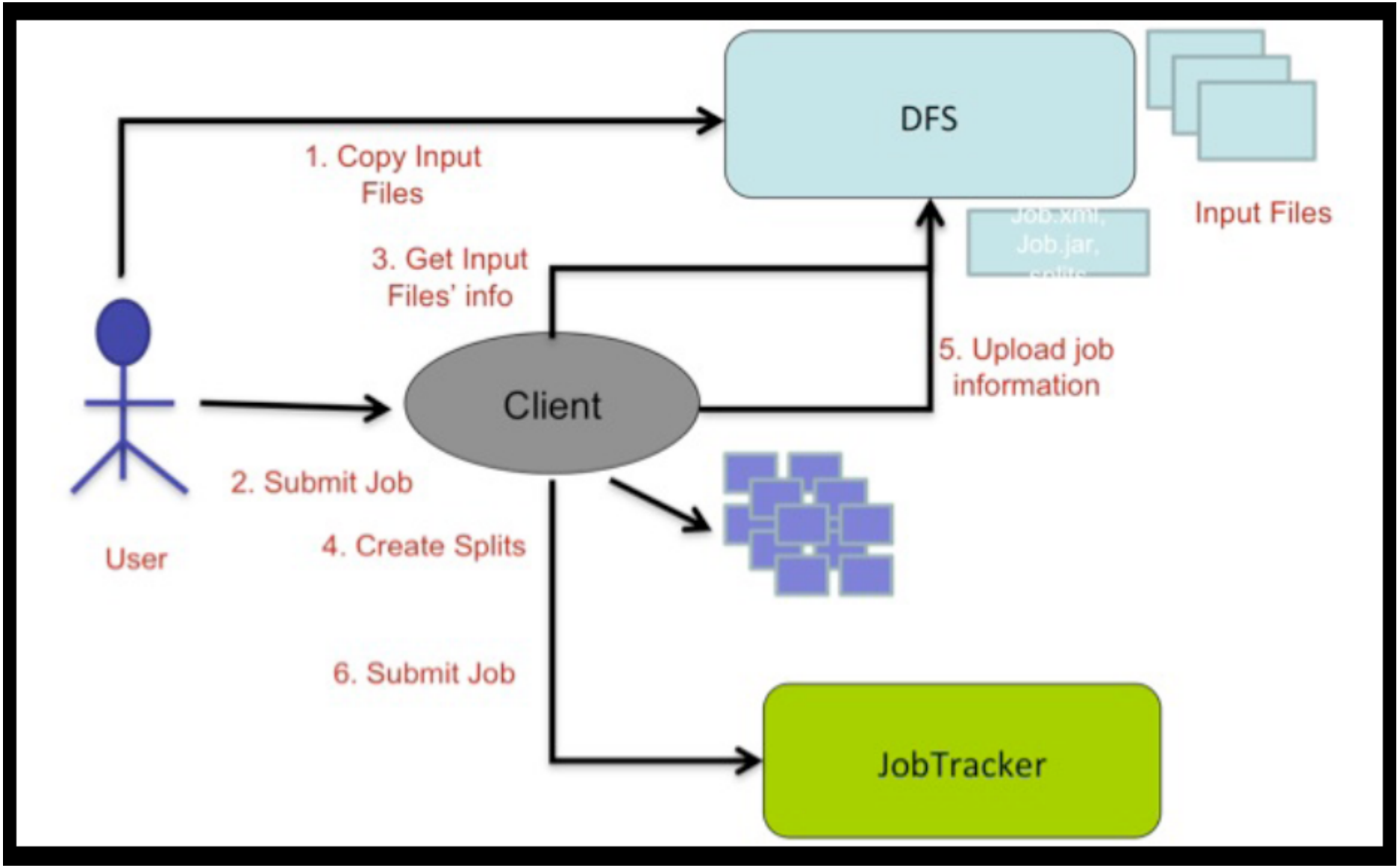
Reduce function

Run this program as a MapReduce job

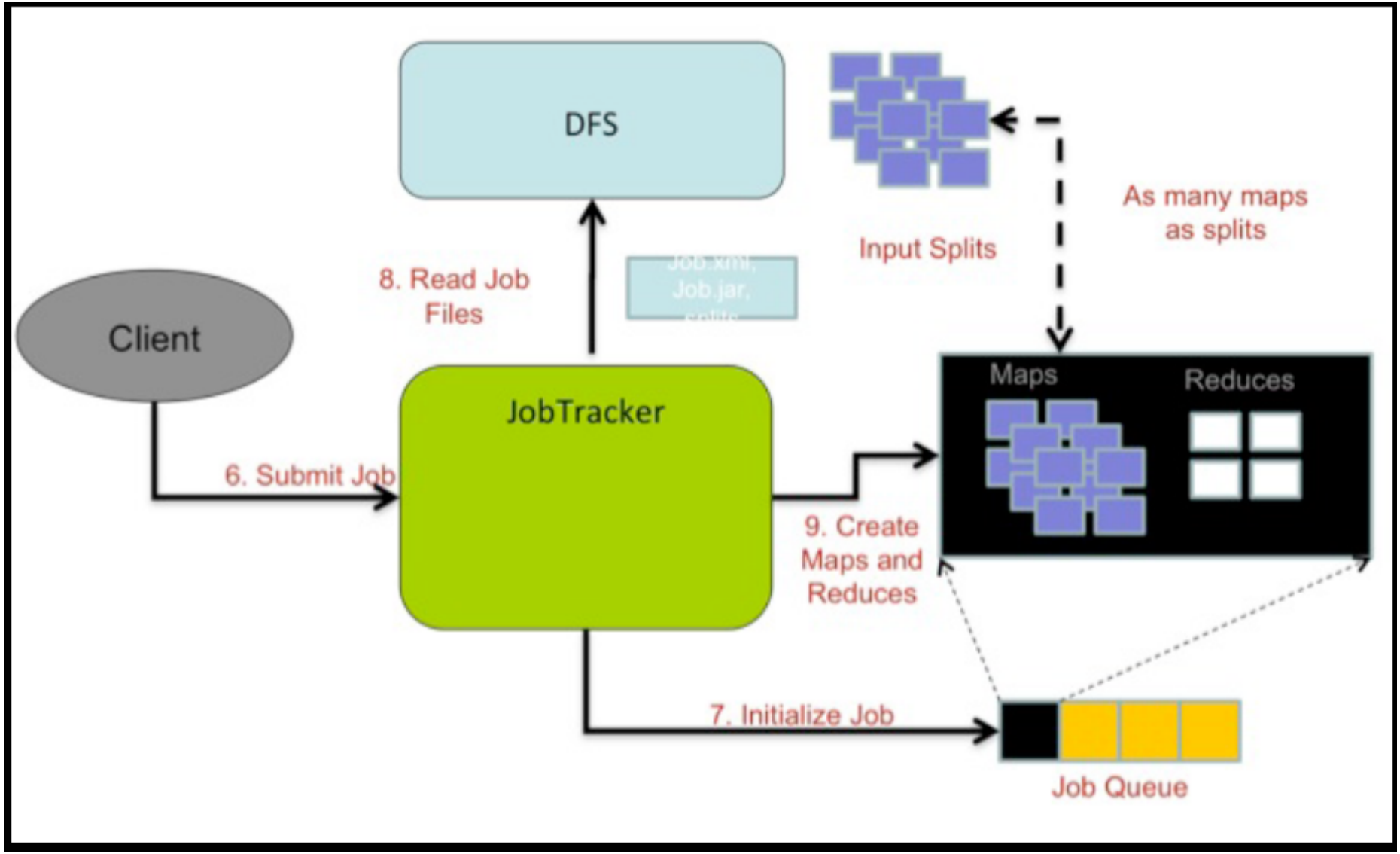
Lifecycle of a MapReduce Job



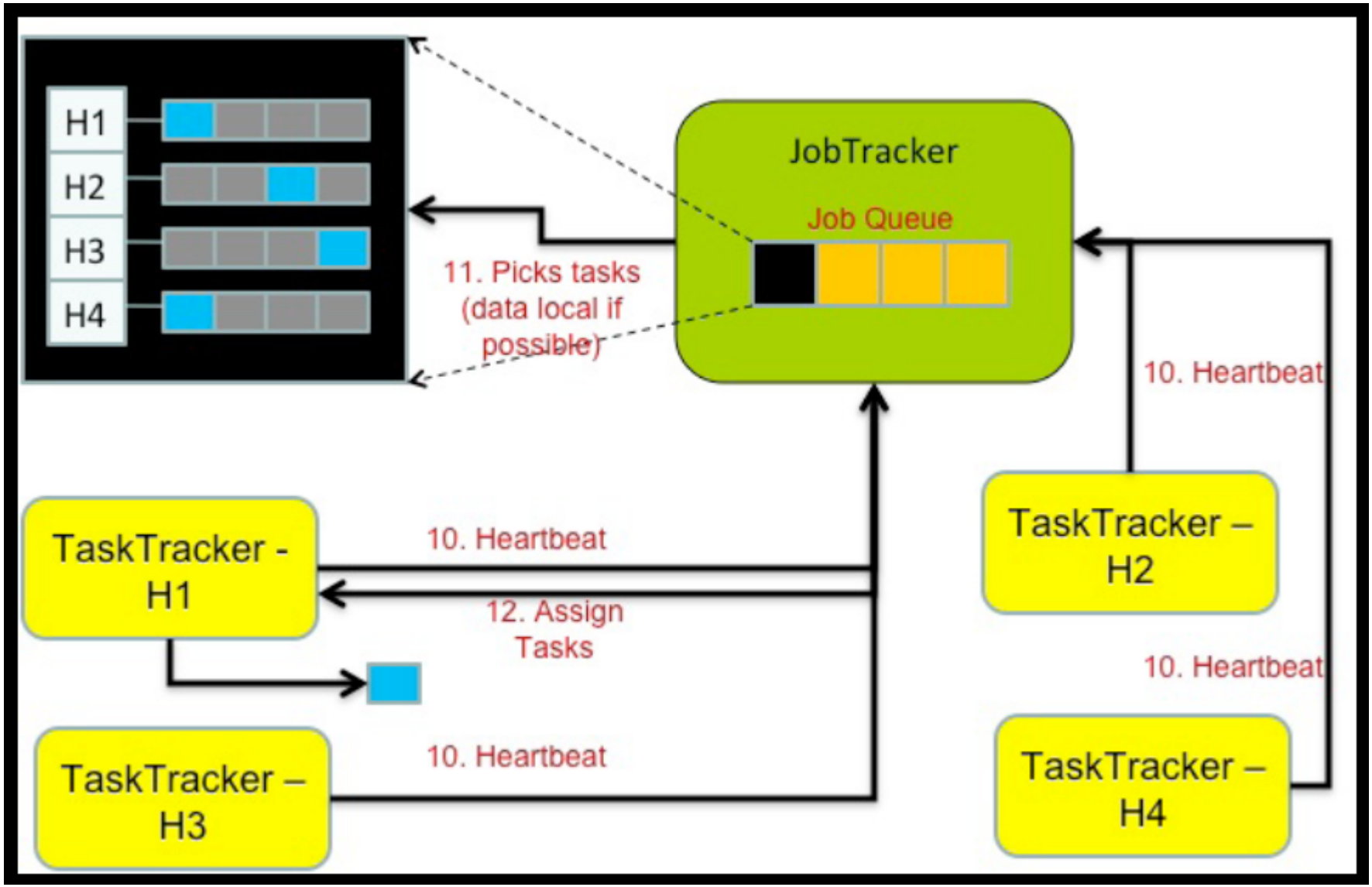
Job Submission



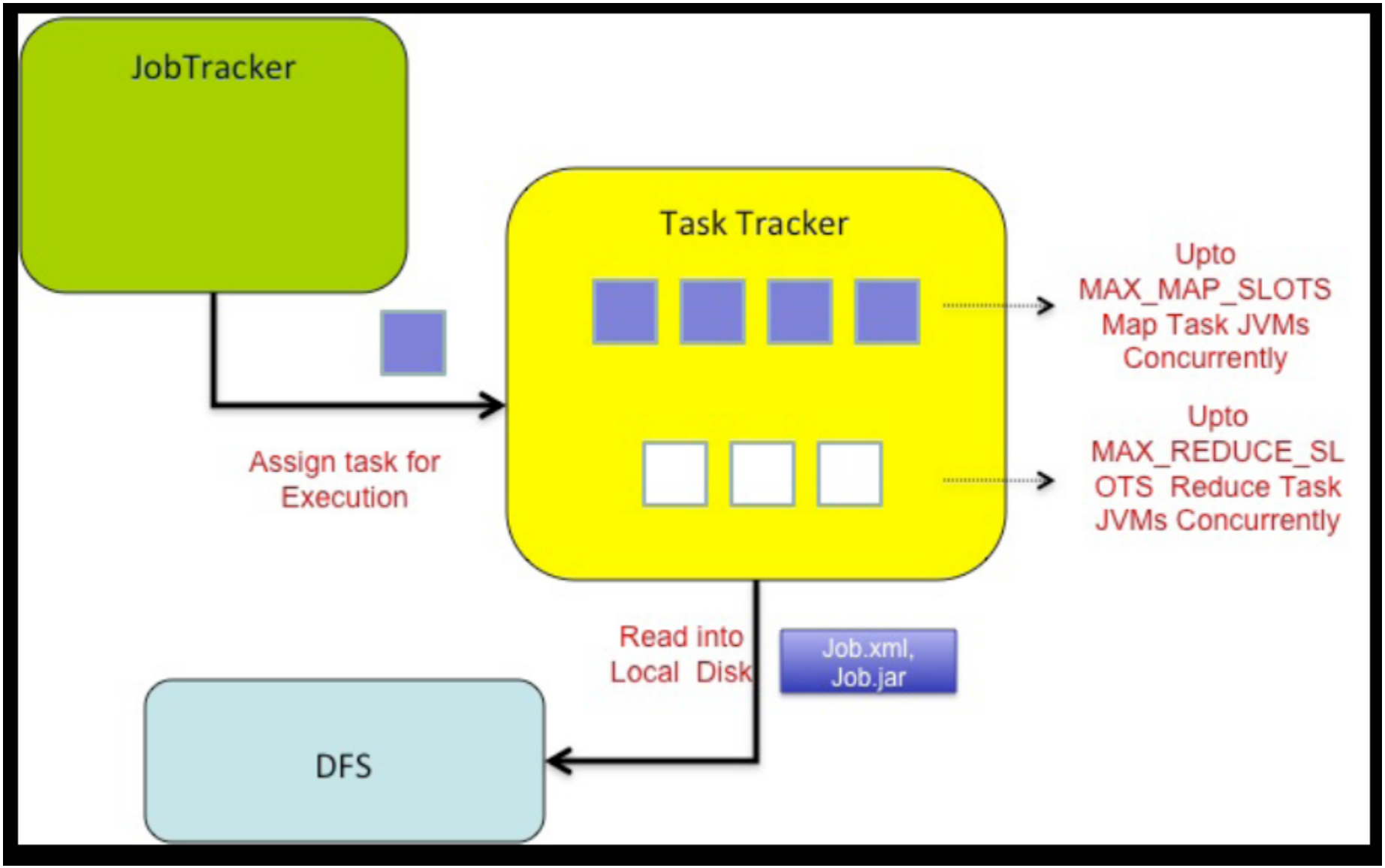
Initialization



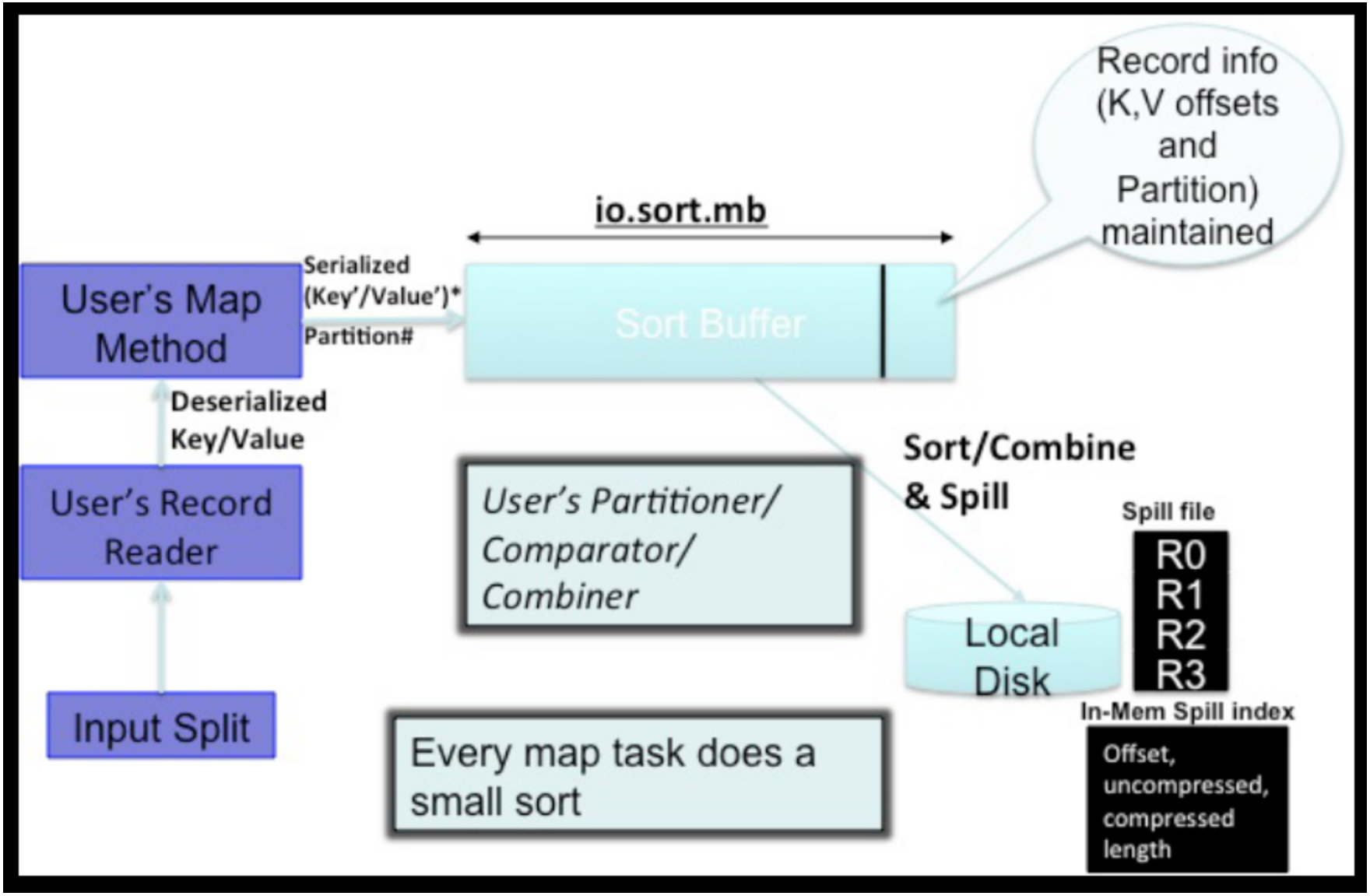
Scheduling


H1
H2
H3
H4
JobTracker
Job Queue
11. Picks tasks (data local if possible)
10. Heartbeat
TaskTracker - H1
10. Heartbeat
12. Assign Tasks
TaskTracker – H2
10. Heartbeat
TaskTracker – H3
10. Heartbeat
TaskTracker – H4

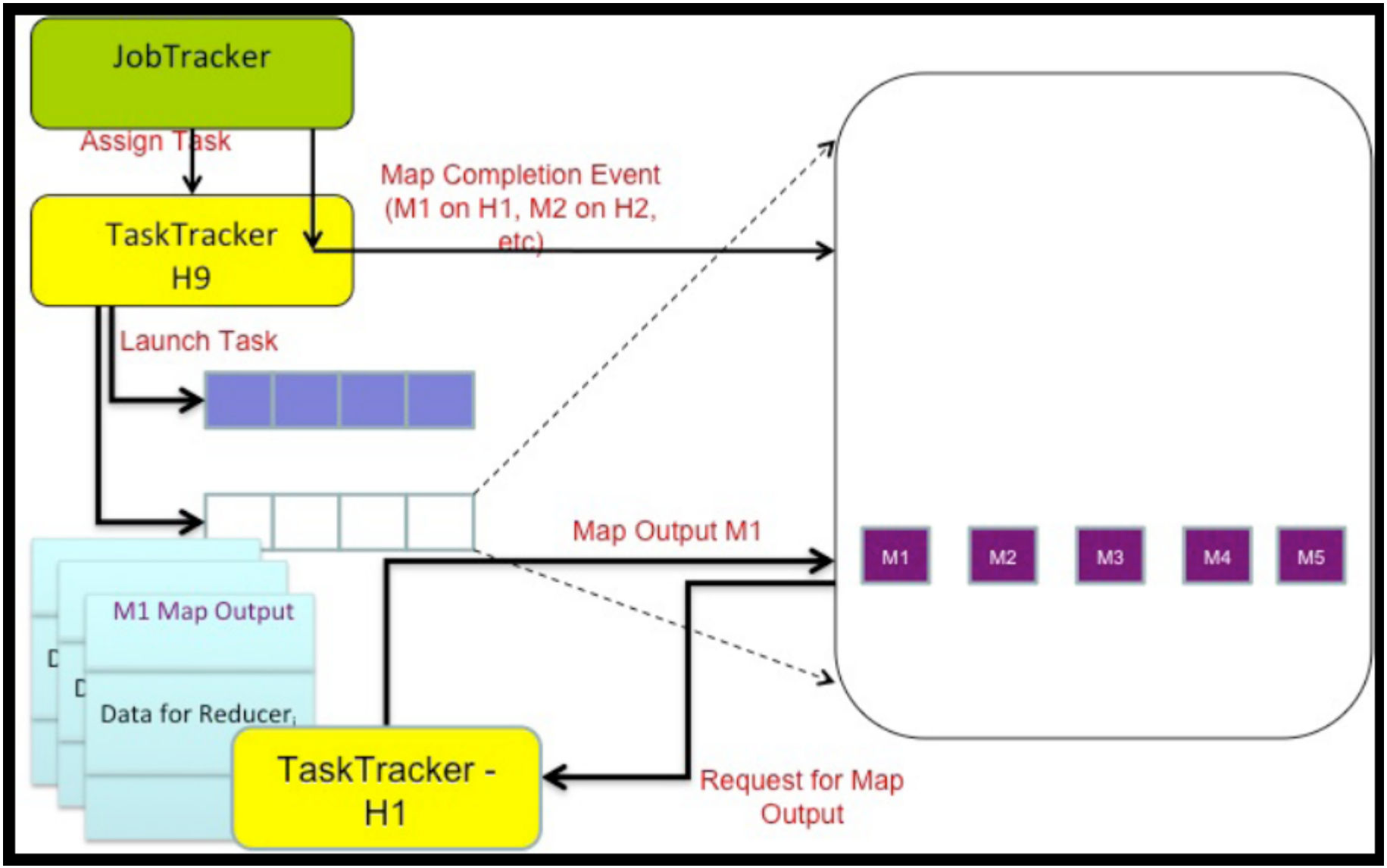
Execution


JobTracker
Task Tracker
Upto MAX_MAP_SLOTS Map Task JVMs Concurrently
Upto MAX_REDUCE_SLOTS Reduce Task JVMs Concurrently
Assign task for Execution
Read into Local Disk
Job.xml, Job.jar
DFS

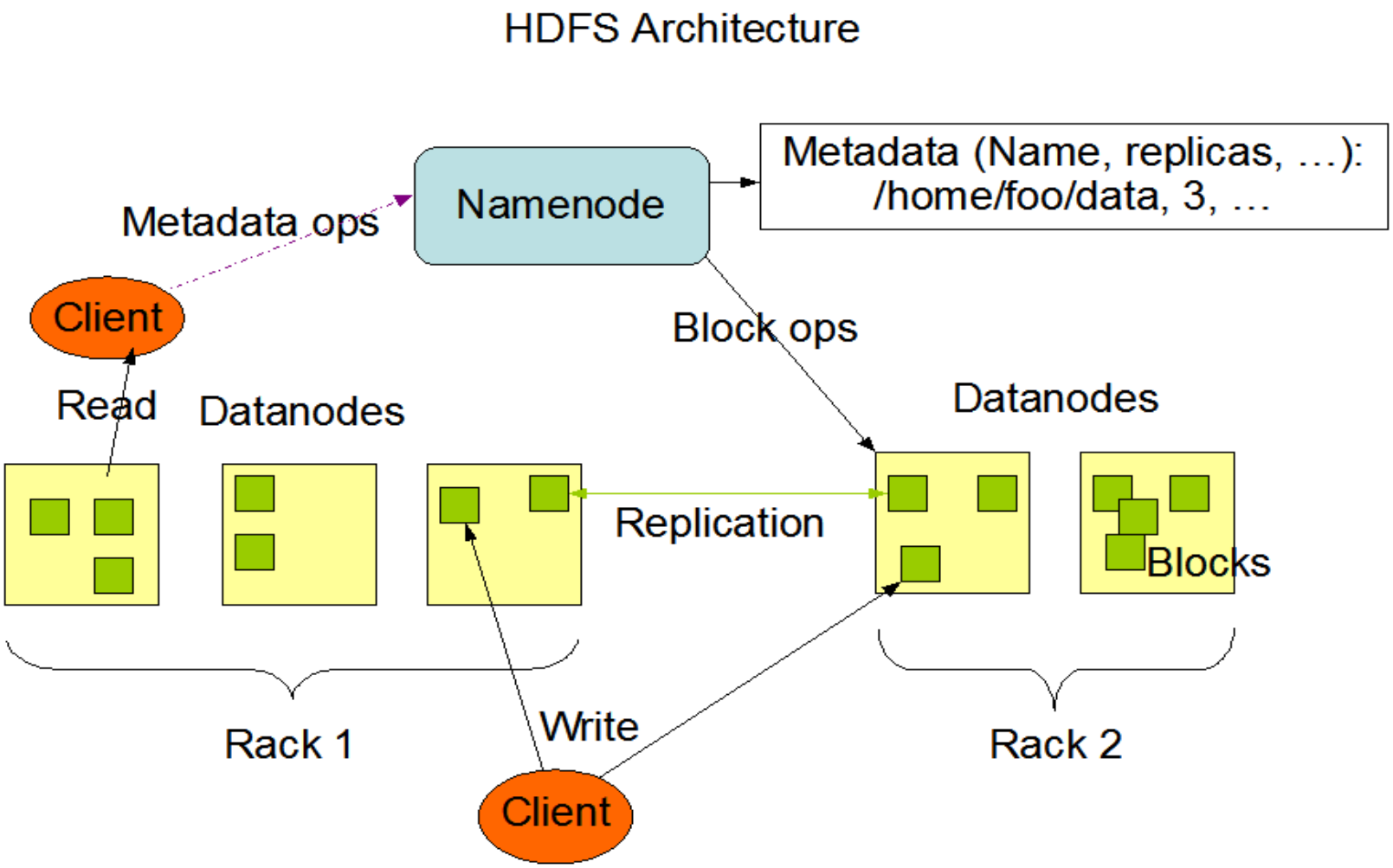
Map Task


Record info (K,V offsets and Partition) maintained
io.sort.mb
User's Map Method
Serialized (Key'/Value')*
Partition#
Sort Buffer
Deserialized Key/Value
User's Record Reader
Input Split
User's Partitioner/ Comparator/ Combiner
Sort/Combine & Spill
Spill file
R0
R1
R2
R3
Local Disk
In-Mem Spill index
Offset, uncompressed, compressed length
Every map task does a small sort

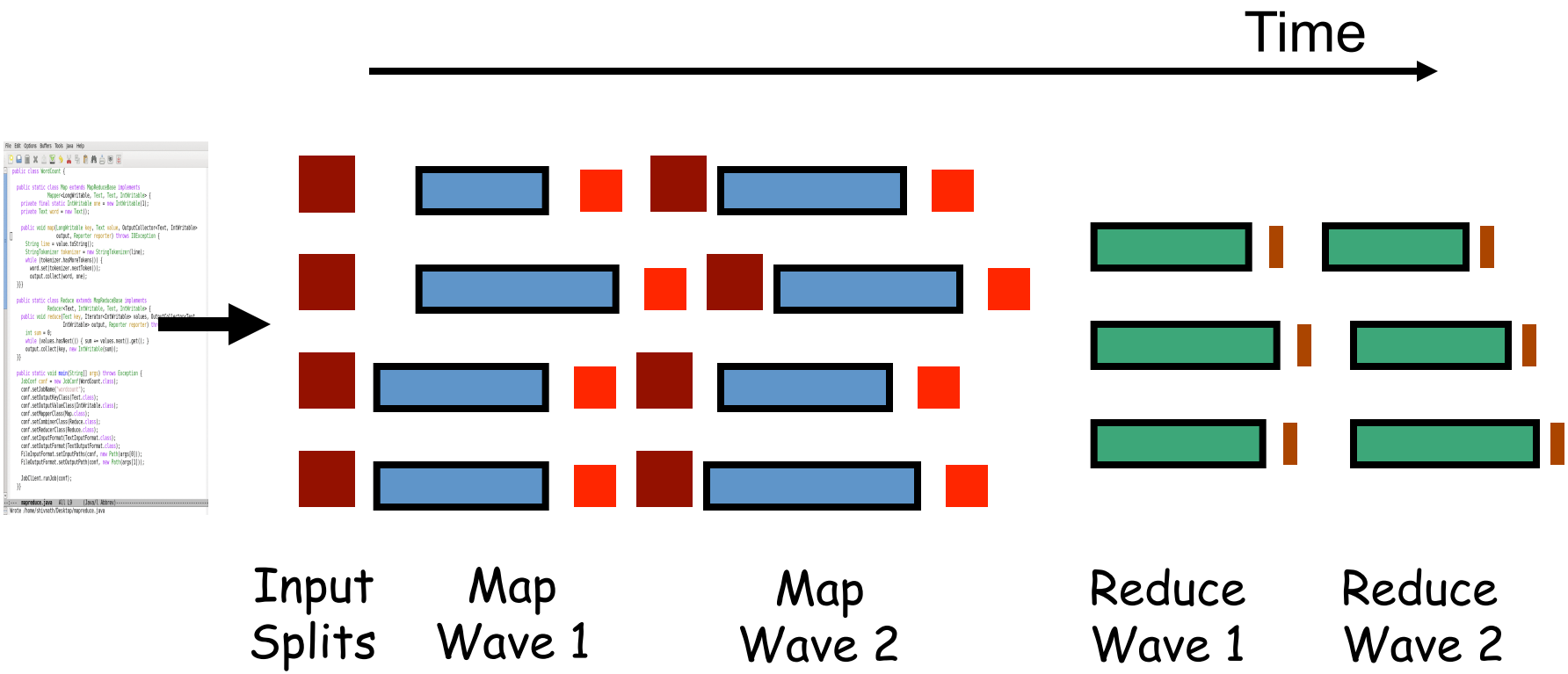
Reduce Tasks



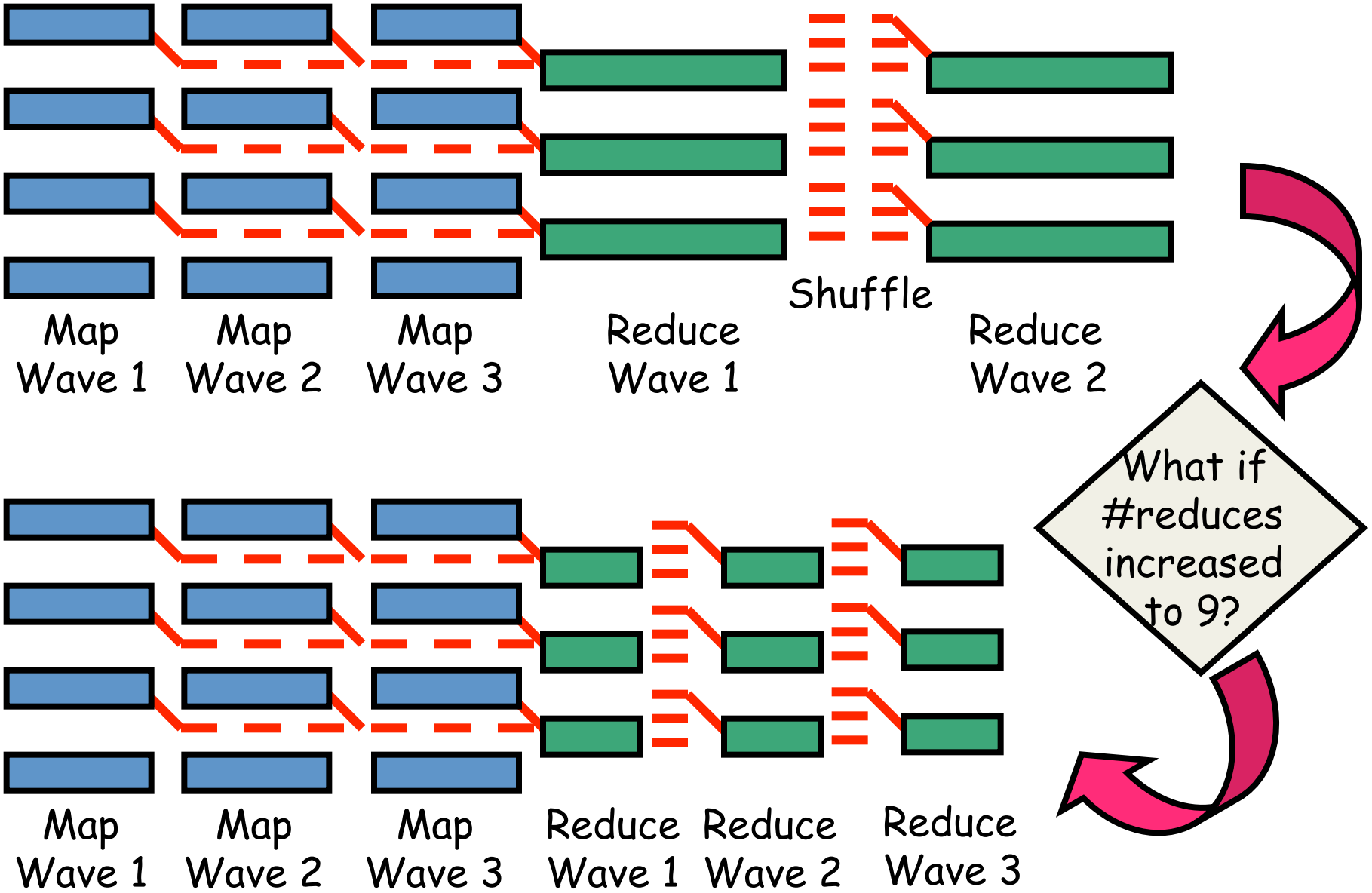
Hadoop Distributed File-System (HDFS)



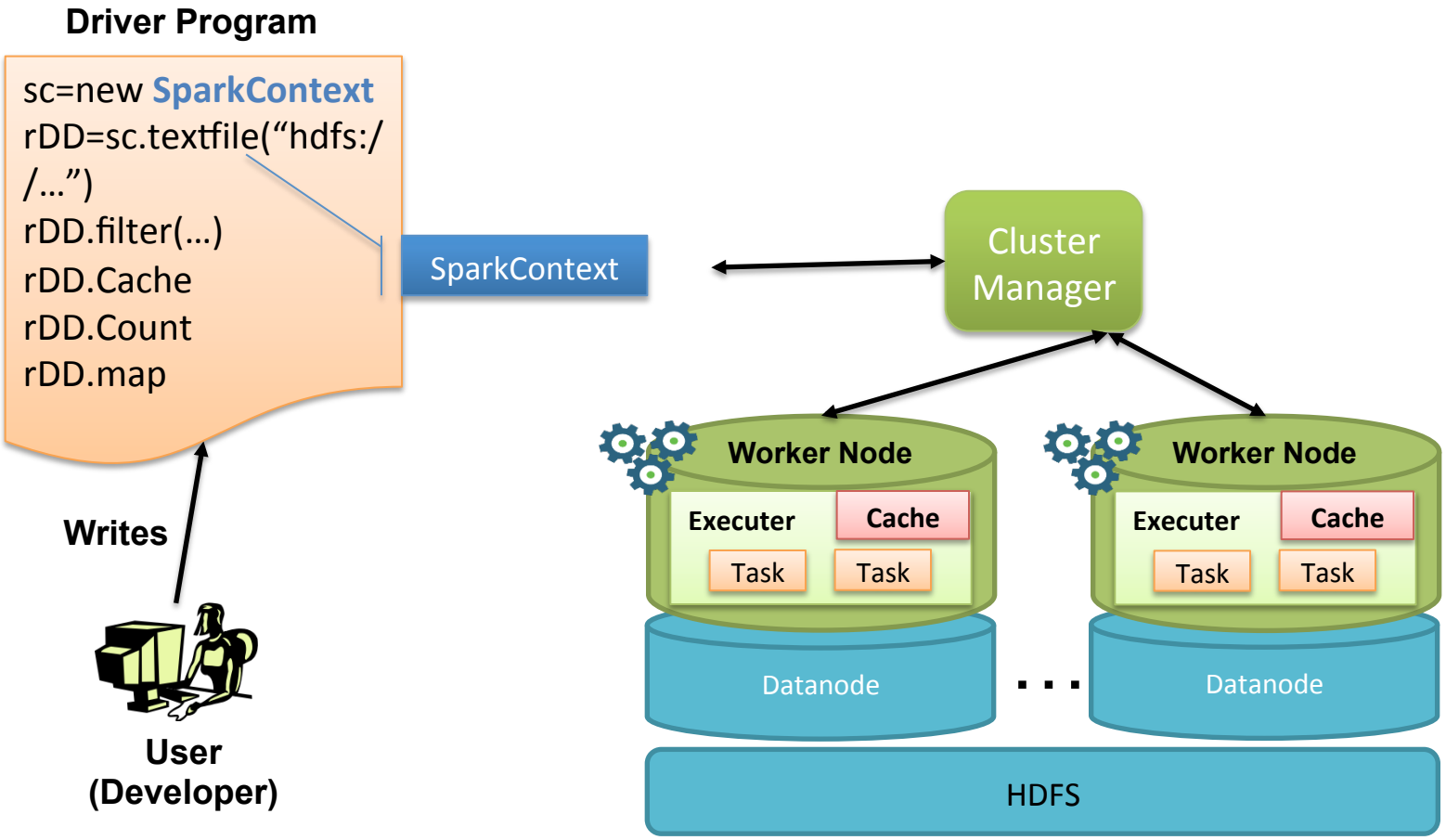
Lifecycle of a MapReduce Job



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


Map Wave 1
Map Wave 2
Map Wave 3
Reduce Wave 1
Shuffle
Reduce Wave 2
What if #reduces increased to 9?
Map Wave 1
Map Wave 2
Map Wave 3
Reduce Wave 1
Reduce Wave 2
Reduce Wave 3

Spark Programming Model



Taken from http://www.slideshare.net/fabiofumarola1/11-from-hadoop-to-spark-12

Spark Programming Model

Driver Program

```
sc=new SparkContext
rDD=sc.textfile("hdfs://...")
rDD.filter(...)
rDD.Cache
rDD.Count
rDD.map
```

RDD (Resilient Distributed Dataset)

- Immutable Data structure
- In-memory (explicitly)
- Fault Tolerant
- Parallel Data Structure
- Controlled partitioning to optimize data placement
- Can be manipulated using a rich set of operators

Writes

User (Developer)

RDD

- Programming Interface: Programmer can perform 3 types of operations

Transformations

- Create a new dataset from an existing one.
- Lazy in nature. They are executed only when some action is performed.
- Example :
  - Map(func)
  - Filter(func)
  - Distinct()

Actions

- Returns to the driver program a value or exports data to a storage system after performing a computation.
- Example:
  - Count()
  - Reduce(funct)
  - Collect
  - Take()

Persistence

- For caching datasets in-memory for future operations.
- Option to store on disk or RAM or mixed (Storage Level).
- Example:
  - Persist()
  - Cache()

How Spark works

- RDD: Parallel collection with partitions
- User application can create RDDs, transform them, and run actions.
- This results in a DAG (Directed Acyclic Graph) of operators.
- DAG is compiled into stages
- Each stage is executed as a collection of Tasks (one Task for each Partition).

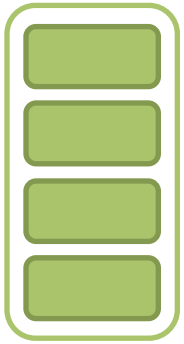
Summary of Components

- Task: The fundamental unit of execution in Spark
- Stage: Collection of Tasks that run in parallel
- DAG: Logical Graph of RDD operations
- RDD: Parallel dataset of objects with partitions

Example

sc.textFile("/wiki/pagecounts")

RDD[String]

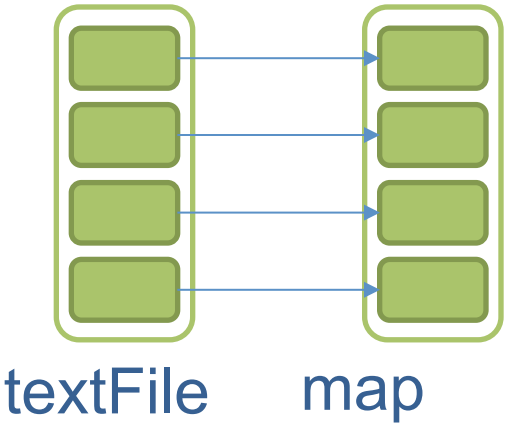


textFile

Example

```
sc.textFile("/wiki/pagecounts")
        .map(line => line.split("\t"))
```

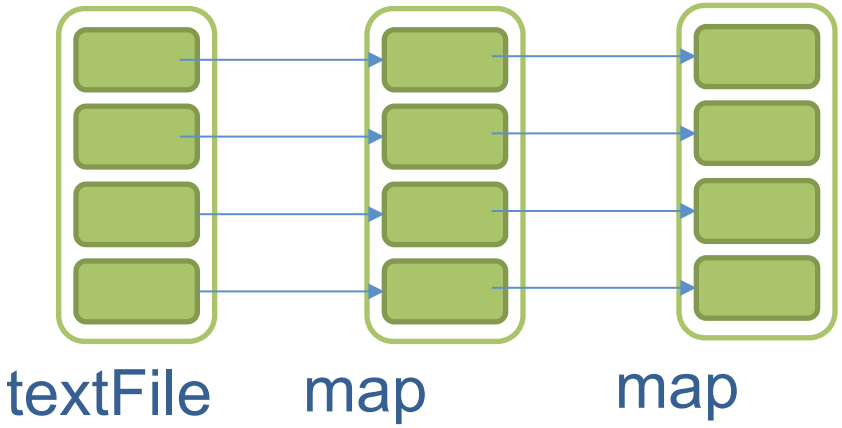
RDD[String]
RDD[List[String]]



Example

```
sc.textFile("/wiki/pagecounts")
        .map(line => line.split("\t"))
        .map(R => (R[0], int(R[1])))
```

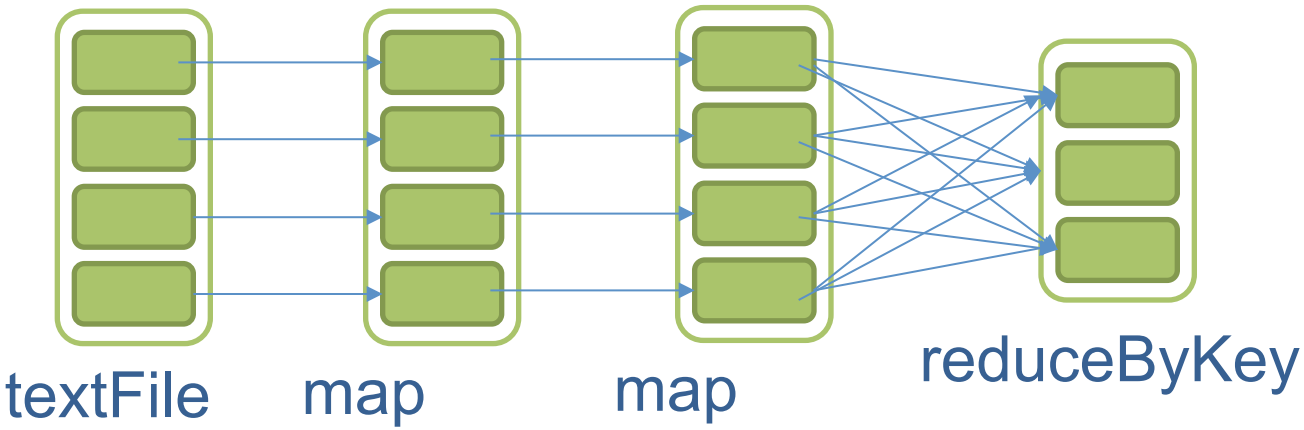
RDD[String]
RDD[List[String]]
RDD[(String, Int)]



Example

```
sc.textFile("/wiki/pagecounts")
        .map(line => line.split("\t"))
        .map(R => (R[0], int(R[1])))
        .reduceByKey(_+_)
```

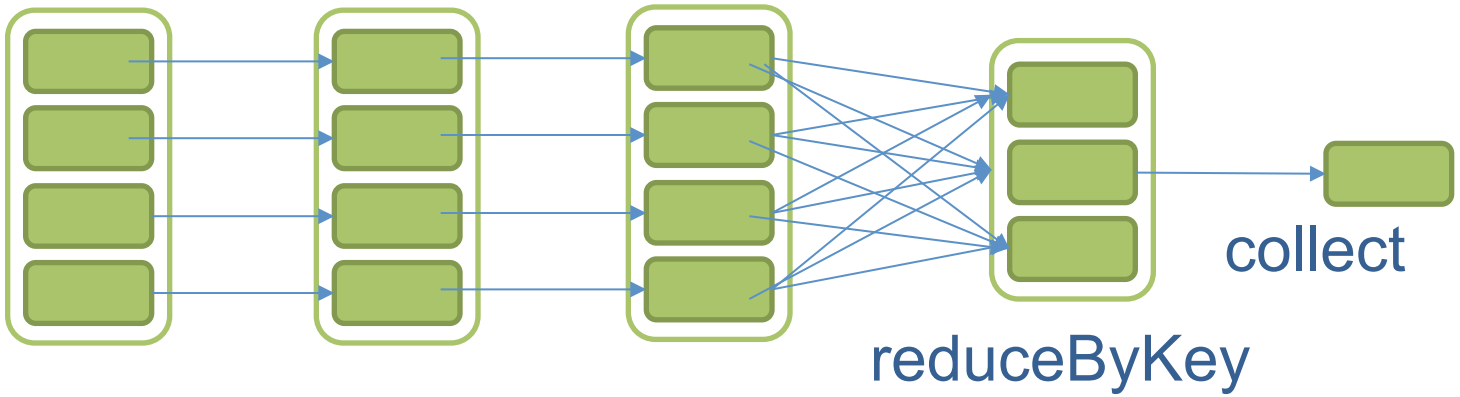
RDD[String]
RDD[List[String]]
RDD[(String, Int)]
RDD[(String, Int)]



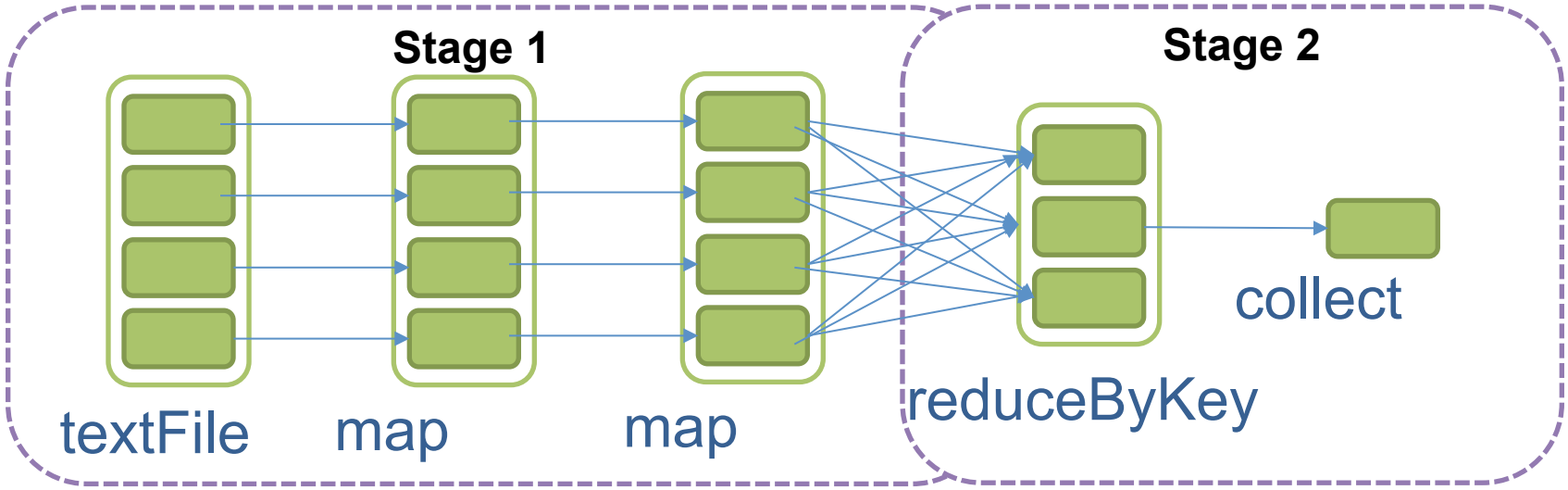
Example

```
sc.textFile("/wiki/pagecounts")
        .map(line => line.split("\t"))
        .map(R => (R[0], int(R[1])))
        .reduceByKey(_+_, 3)
        .collect()
```

RDD[String]
RDD[List[String]]
RDD[(String, Int)]
RDD[(String, Int)]
Array[(String, Int)]

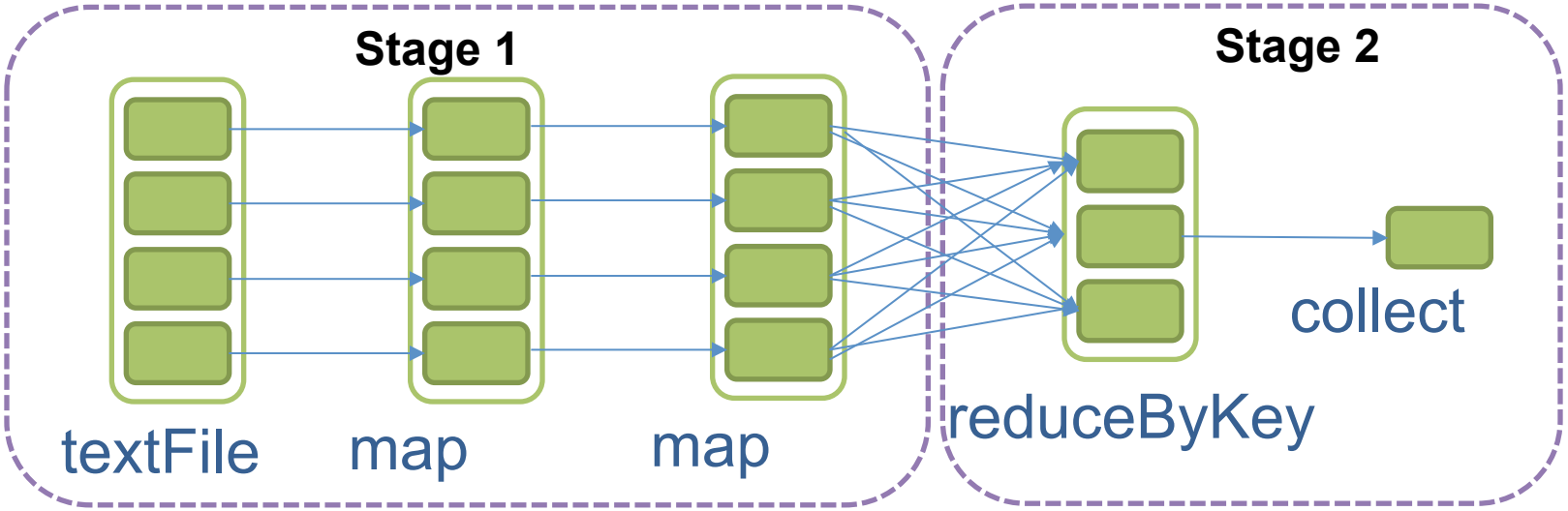


Execution Plan

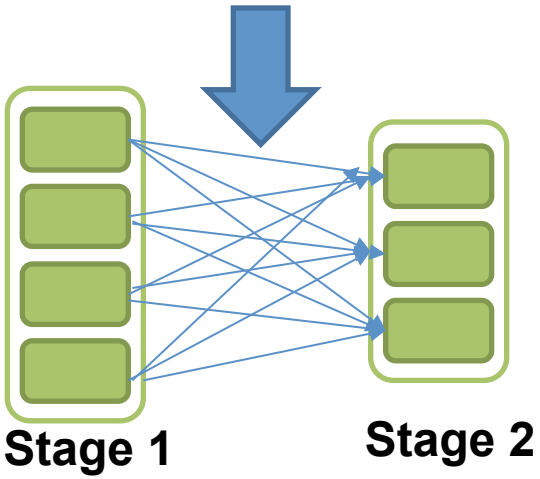


Stages are sequences of RDDs, that don't have a Shuffle in between

Execution Plan



1. Read HDFS split
2. Apply both the maps
3. Start partial reduce
4. Write shuffle data



1. Read shuffle data
2. Final reduce
3. Send result to driver program

Stage Execution

- Create a task for each Partition in the input RDD
- Serialize the Task
- Schedule and ship Tasks to Slaves

And all this happens internally (you don't have to do anything)

Spark Executor (Slave)