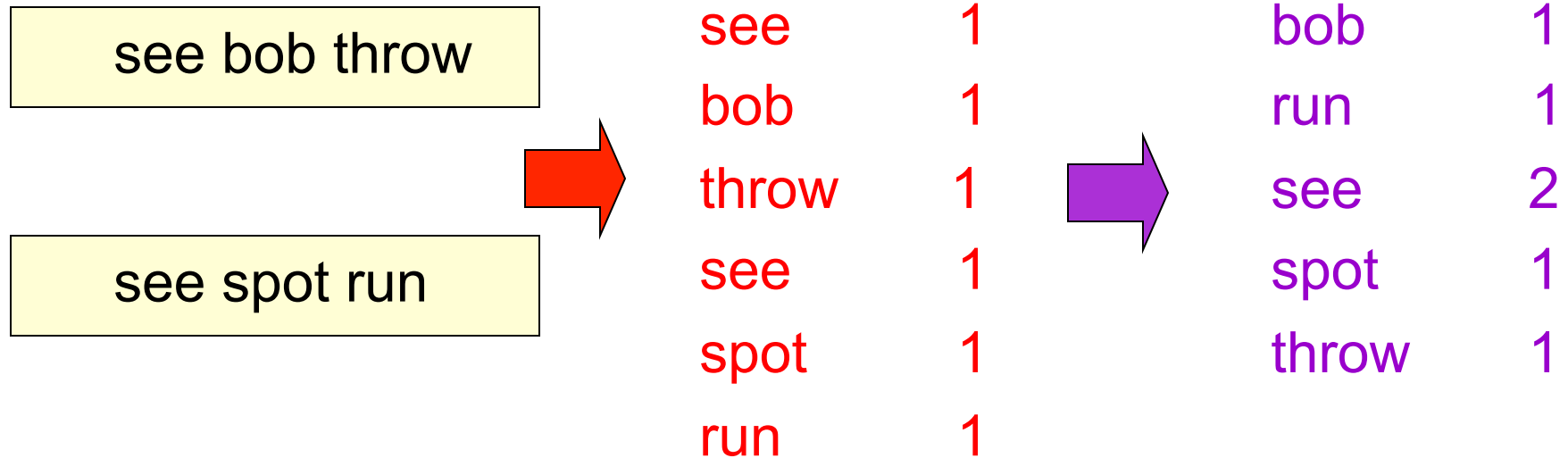


Data-intensive Computing Systems

Introduction to MapReduce and Hadoop

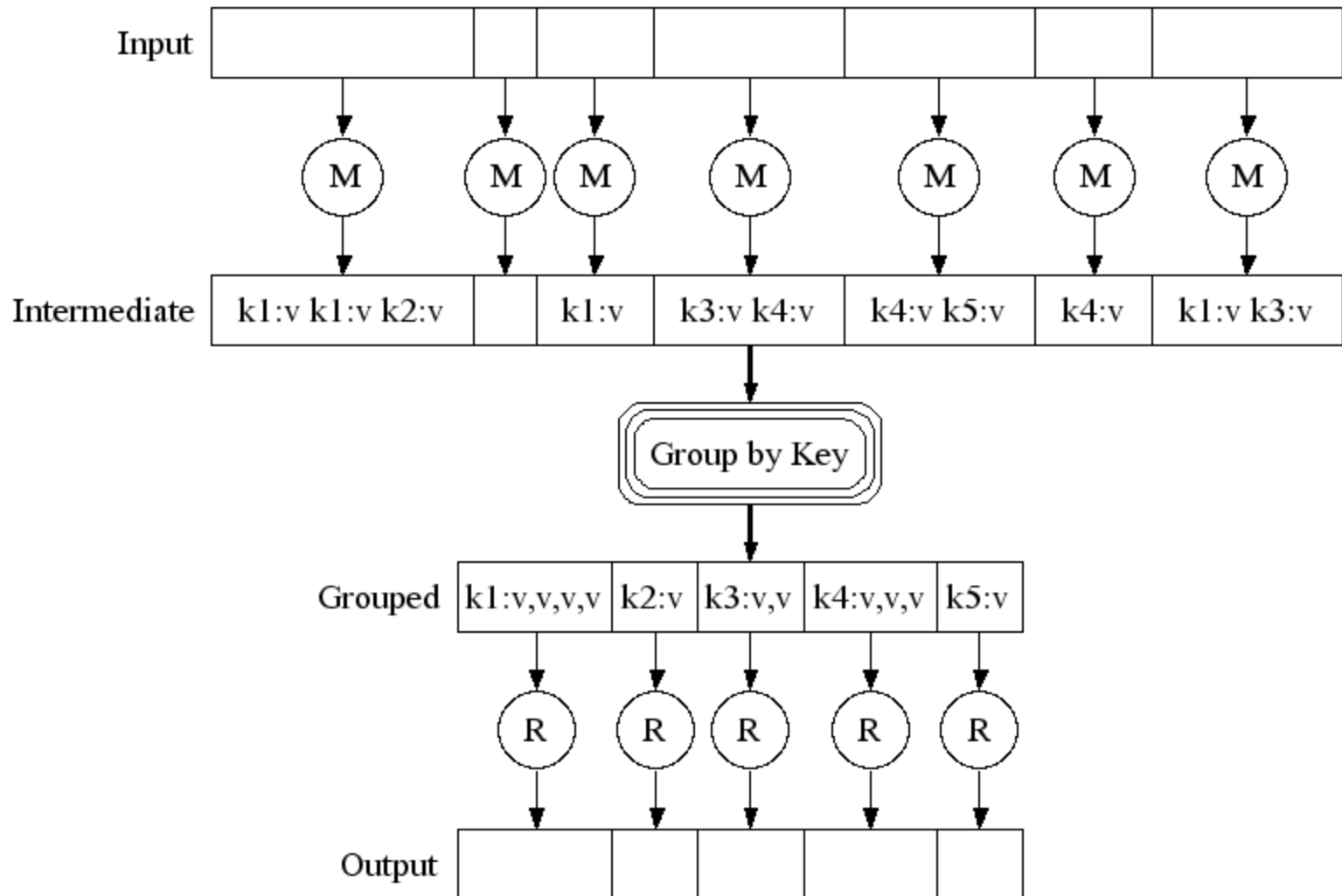
Shivnath Babu

Word Count over a Given Set of Web Pages

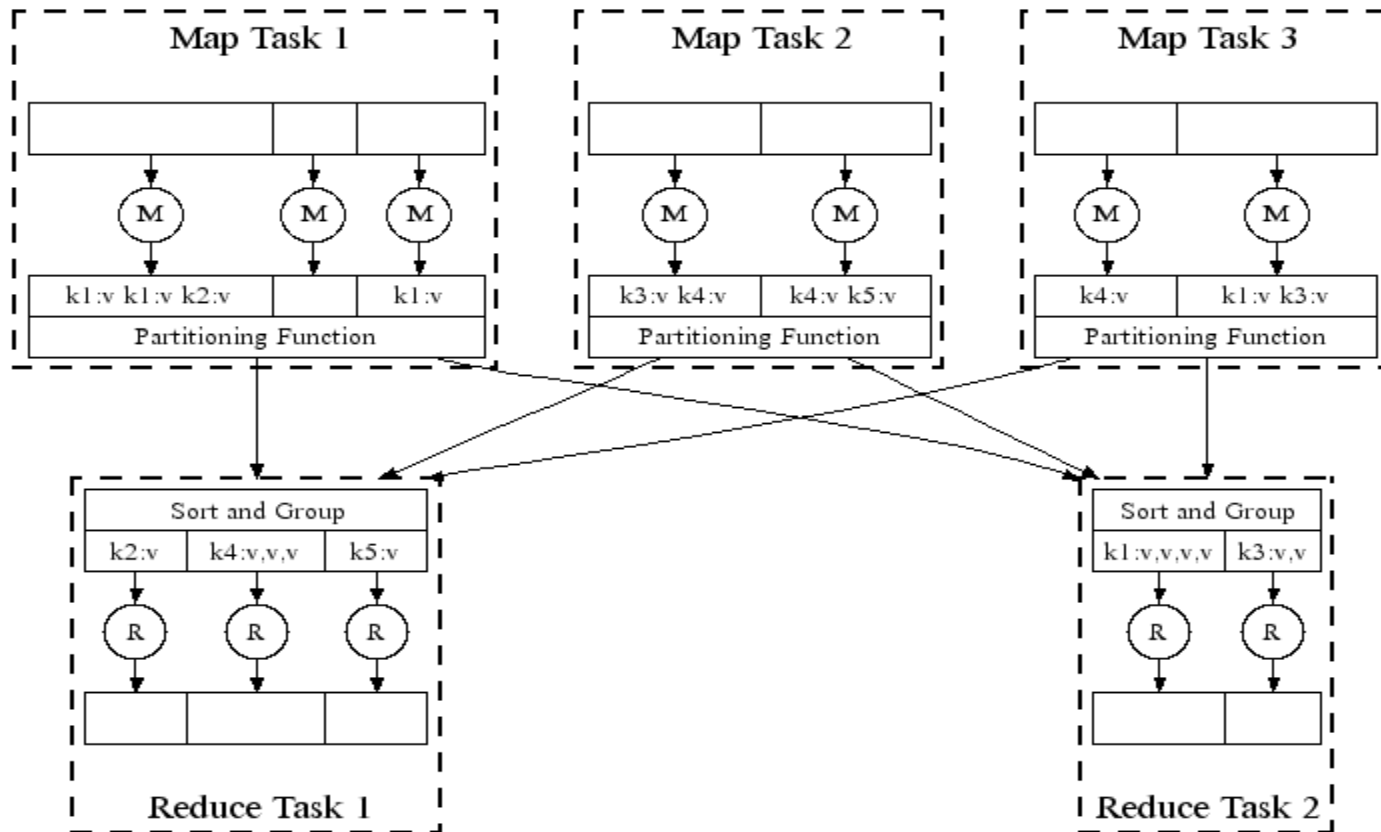


Can we do word count in parallel?

The MapReduce Framework (pioneered by Google)



Automatic Parallel Execution in MapReduce (Google)



Handles failures automatically, e.g., restarts tasks if a node fails; runs multiples copies of the same task to avoid a slow task slowing down the whole job

MapReduce in Hadoop (1)

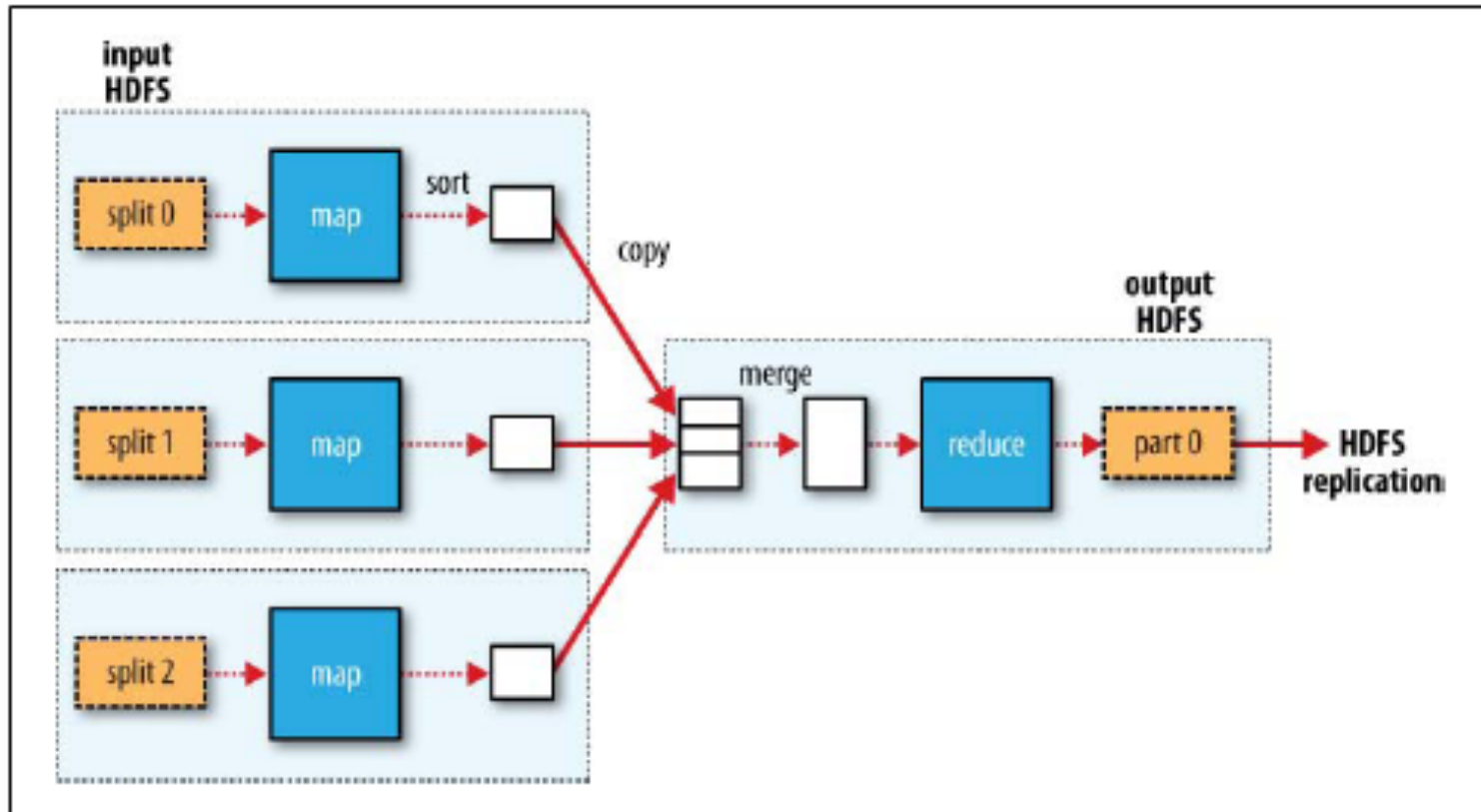


Figure 2-2. MapReduce data flow with a single reduce task

MapReduce in Hadoop (2)

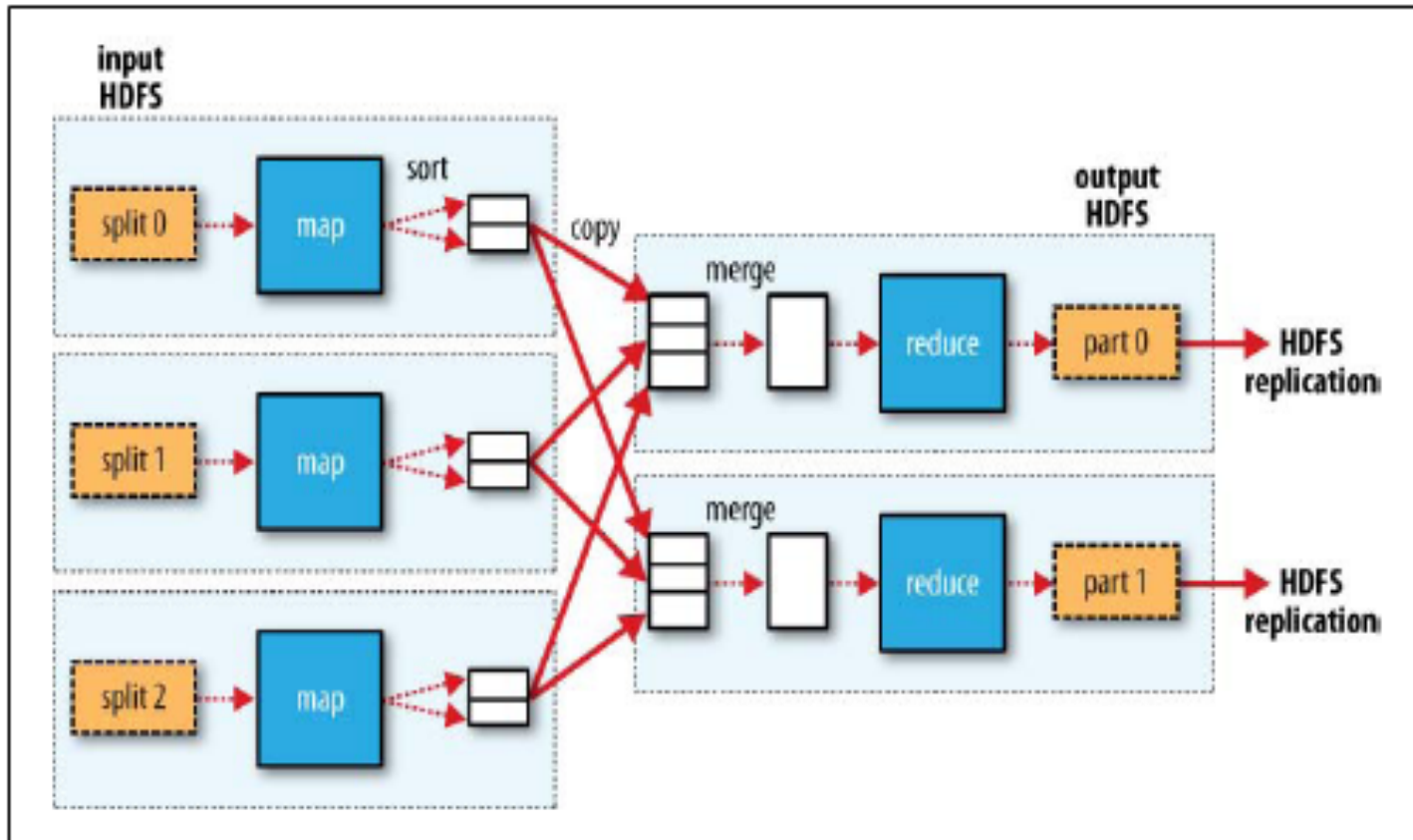


Figure 2-3. MapReduce data flow with multiple reduce tasks

MapReduce in Hadoop (3)

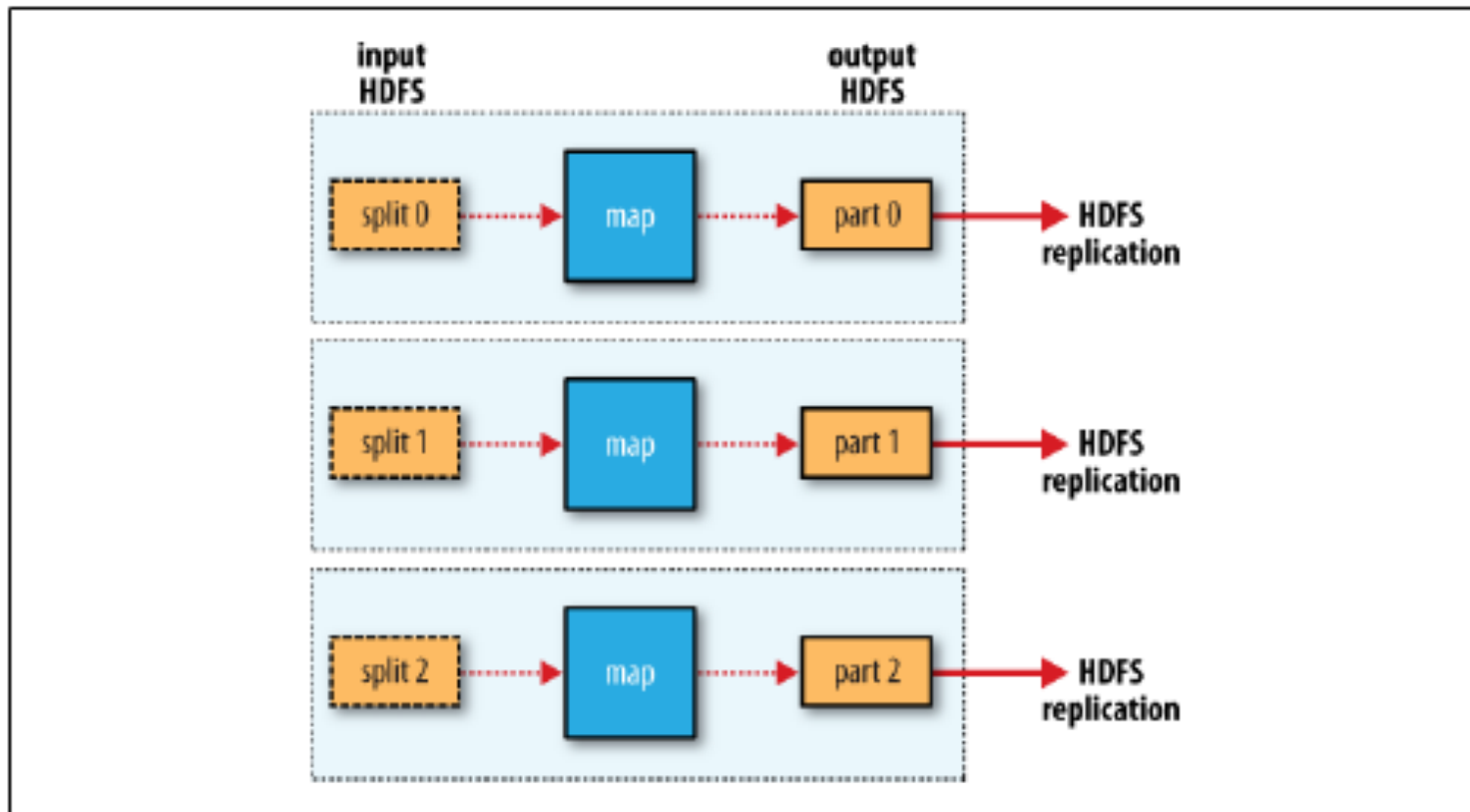
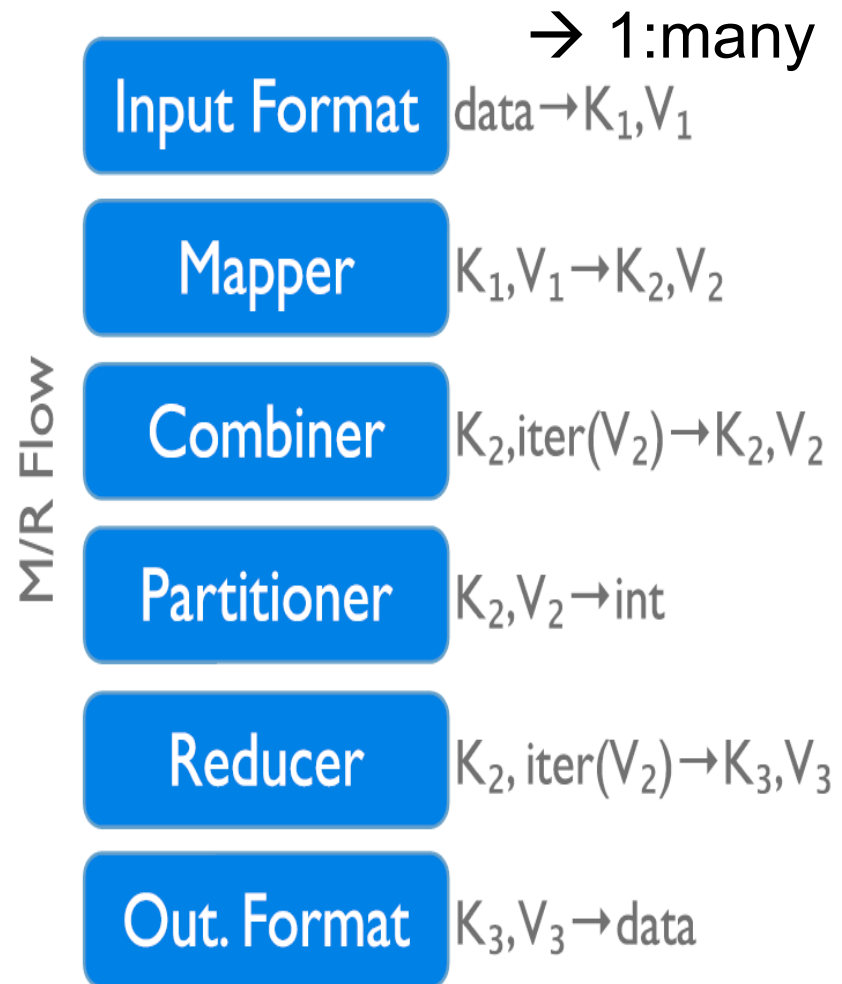


Figure 2-4. MapReduce data flow with no reduce tasks

Data Flow in a MapReduce Program in Hadoop

- InputFormat
- Map function
- Partitioner
- Sorting & Merging
- Combiner
- Shuffling
- Merging
- Reduce function
- OutputFormat



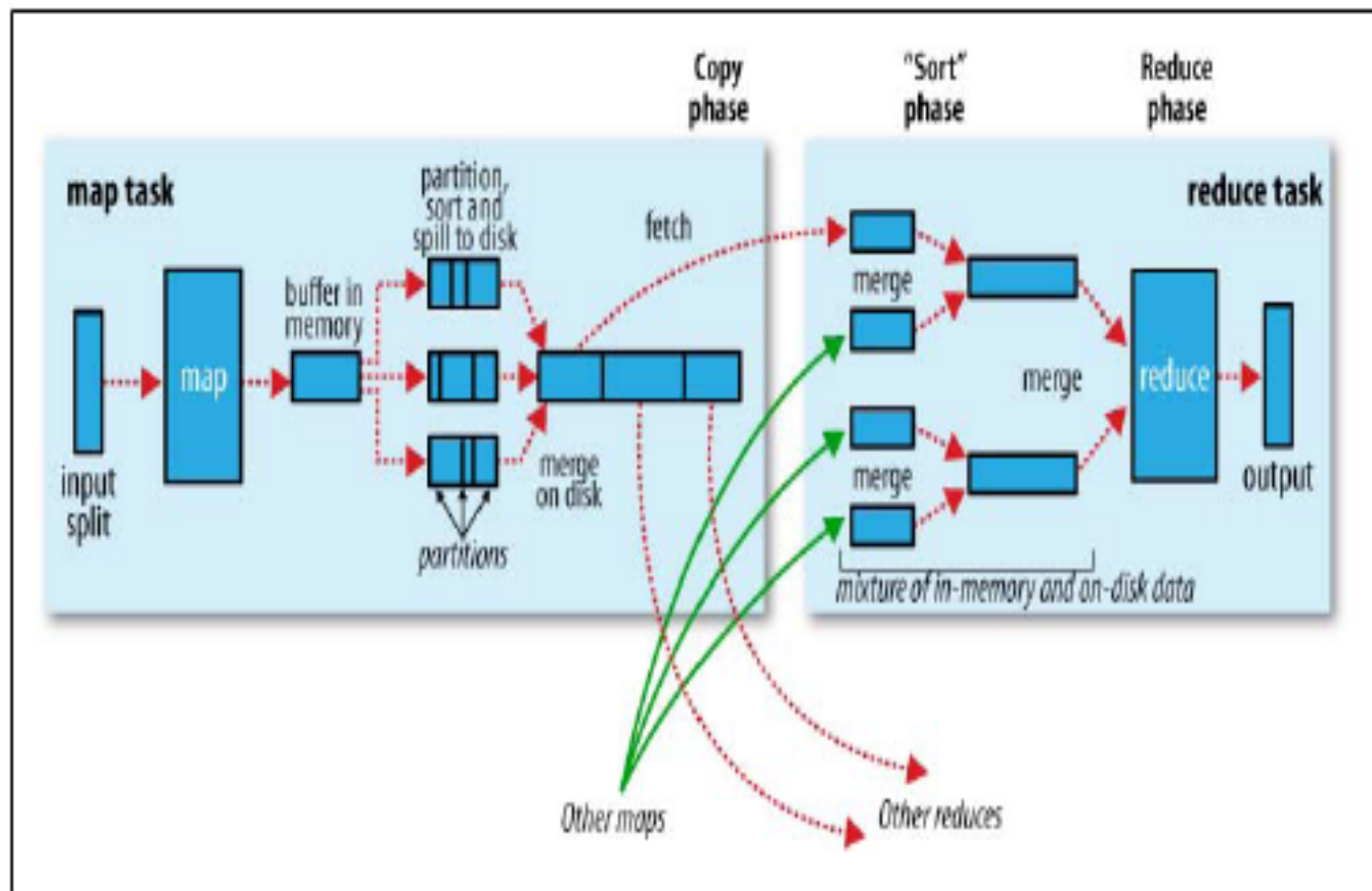


Figure 6-4. Shuffle and sort in MapReduce

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);
        }
    }
}
```

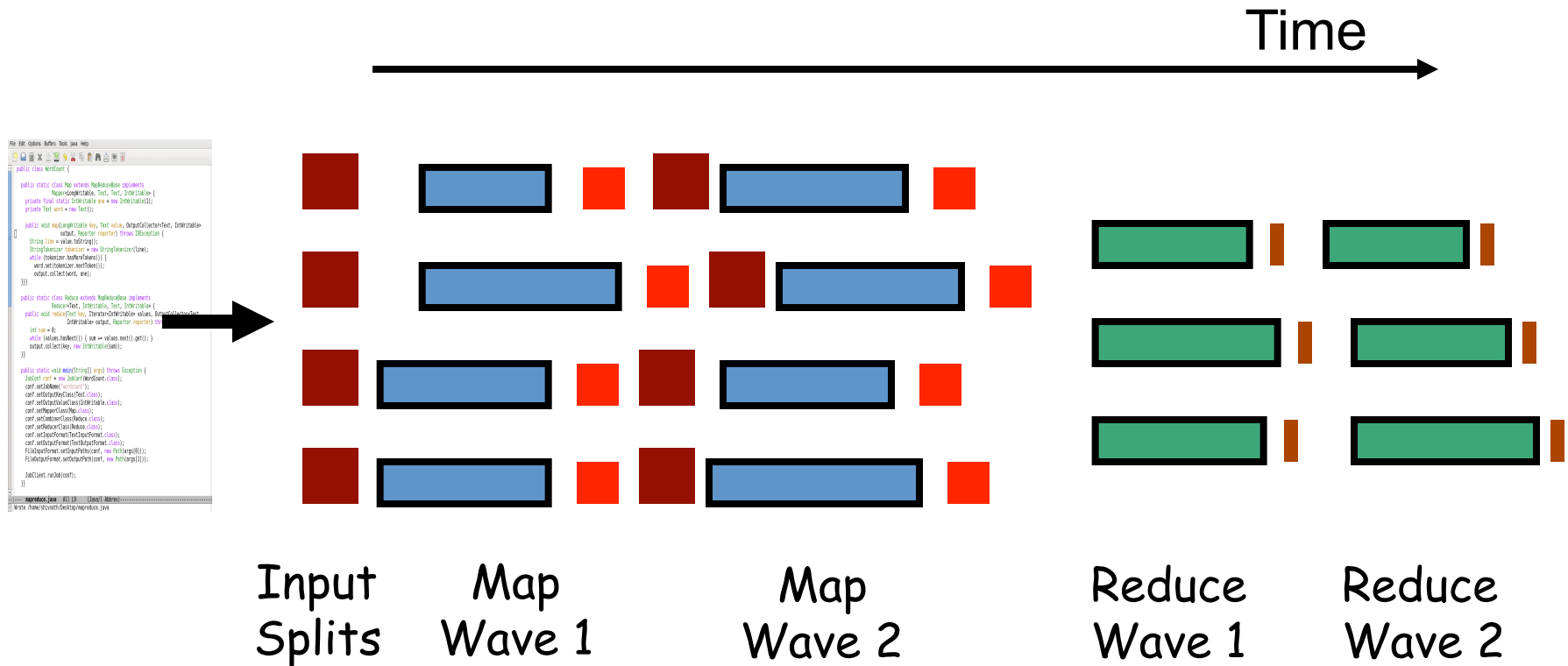
Map function

Reduce function

Run this program as a MapReduce job

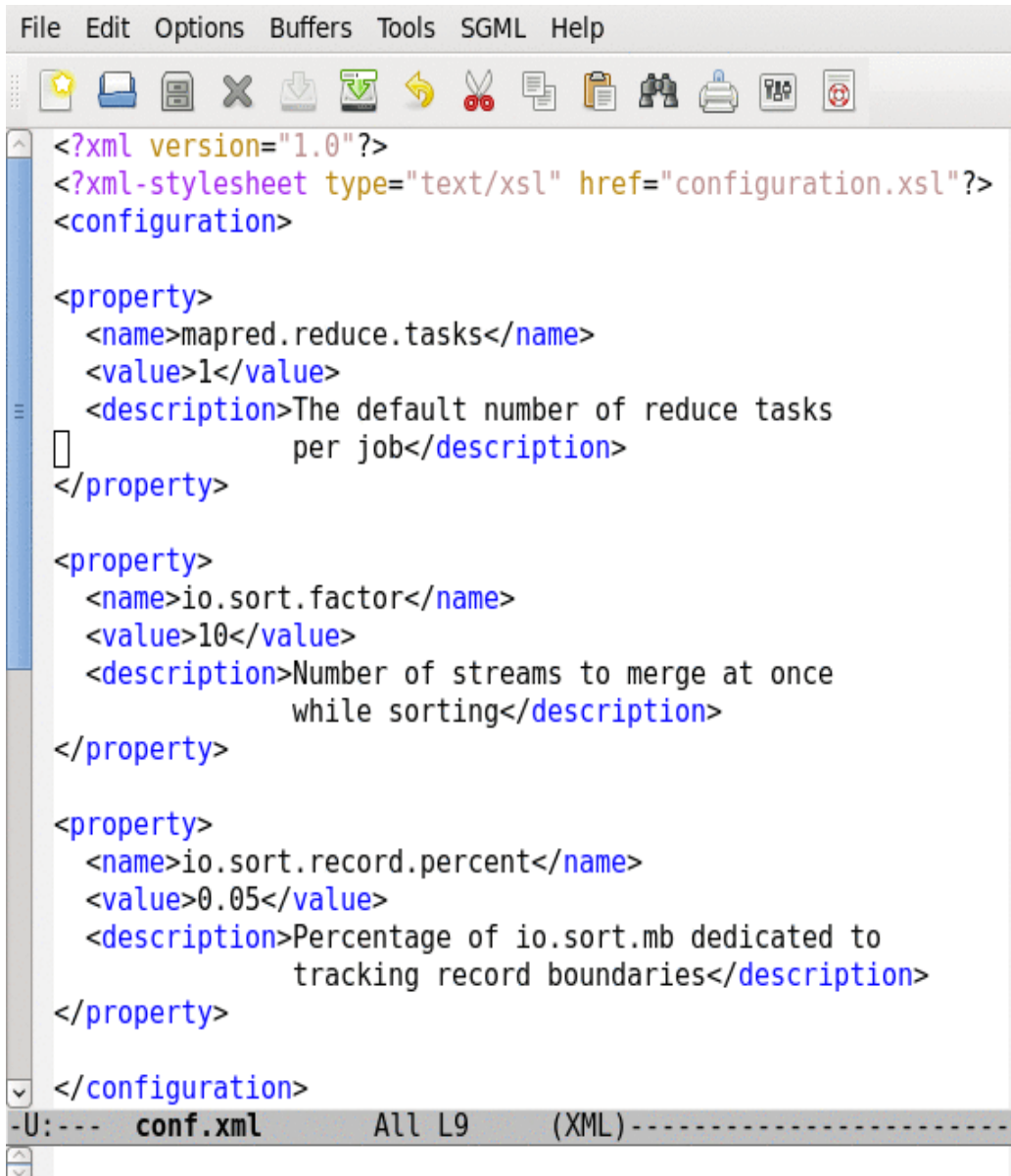
mapreduce.java All L9 (Java/l Abbrev) -----
Wrote /home/shivnath/Desktop/mapreduce.java

Lifecycle of a MapReduce Job



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

Job Configuration Parameters

A screenshot of an XML editor window. The title bar shows 'File Edit Options Buffers Tools SGML Help'. The toolbar contains icons for file operations like opening, saving, and printing. The main text area displays XML code for a Hadoop configuration file. The code starts with an XML declaration and a stylesheet reference, followed by a root element 'configuration'. Inside, there are three 'property' elements, each with 'name', 'value', and 'description' sub-elements. The first property is 'mapred.reduce.tasks' with value '1'. The second is 'io.sort.factor' with value '10'. The third is 'io.sort.record.percent' with value '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

How to sort data using Hadoop?

Let us look at a complete
example MapReduce program
in Hadoop