MR Job Configuration

MR Job Configuration

Let's say you have set up a MR job
You can configure the MR job in 2 ways

Command line options

Job object

Command line options

Hadoop has a few helper classes/ interfaces that allow you to configure your job from the command line

GenericOptionsParser

ToolRunner

Tool

Command line options

```
public static void main(String[] args) throws Exception {
    if(args.length !=2){
        System.err.println("Invalid Command");
        System.err.println("Usage: WordCount <input path> <output path>");
        System.exit(0);
    }
    Configuration conf = new Configuration();
    Job job = new Job(conf, "wordcount");
    job.setJarByClass(WordCount.class);
    job.setJobName("Word Count");
    FileInputFormat.addInputPath(job, new Path(args[0]));
    FileOutputFormat.setOutputPath(job, new Path(args[1]));
    job.setMapperClass(WordCountMapper.class);
    job.setReducerClass(WordCountReducer.class);
    job.setOutputKeyClass(Text.class);
    job.setOutputValueClass(IntWritable.class);
    System.exit(job.waitForCompletion(true)?0:1);
}
```

Here is the Main Class of an MR job we have seen before

Command line options

```java
public class WordCountwithTool extends Configured implements Tool {
    @Override
    public int run(String[] args) throws Exception{

        if(args.length !=2){
            System.err.println("Invalid Command");
            System.err.println("Usage: <input path> <output path>");
            return -1;
        }

        Job job = Job.getInstance(getConf());
        job.setJobName("select");
        job.setJarByClass(WordCountwithTool.class);

        job.setOutputKeyClass(LongWritable.class);
        job.setOutputValueClass(Text.class);

        job.setMapperClass(WordCountMapper.class);
        job.setReducerClass(WordCountReducer.class);

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

        return job.waitForCompletion(true) ? 0 : 1;
    }

    public static void main(String[] args) throws Exception {
        int exitCode = ToolRunner.run(new WordCountwithTool(), args);
        System.exit(exitCode);
    }
}
```

Here is the Main Class of an MR job we have seen before

Command line options

```
public static void main(String[] args) throws Exception {
    if(args.length !=2){
        System.err.println("Invalid Command");
        System.err.println("Usage: WordCount <input path> <output path>");
        System.exit(0);

    }
        Configuration conf = new Configuration();
        Job job = new Job(conf, "wordcount");

   job.setJarByClass(WordCount.class);
    job.setJobName("Word Count");
    FileInputFormat.addInputPath(job, new Path(args[0]));
    FileOutputFormat.setOutputPath(job, new Path(args[1]));
    job.setMapperClass(WordCountMapper.class);
    job.setReducerClass(WordCountReducer.class);
    job.setOutputKeyClass(Text.class);
    job.setOutputValueClass(IntWritable.class);
    System.exit(job.waitForCompletion(true)?0:1);
}
```

In this class you set up a Job

Command line options

And set all its properties

```
public static void main(String[] args) throws Exception {
    if(args.length !=2){
        System.err.println("Invalid Command");
        System.err.println("Usage: WordCount <input path> <output path>");
        System.exit(0);
    }
    Configuration conf = new Configuration();
    Job job = new Job(conf, "wordcount");
    job.setJarByClass(WordCount.class);
        job.setJobName("Word Count");
        FileInputFormat.addInputPath(job, new Path(args[0]));
        FileOutputFormat.setOutputPath(job, new Path(args[1]));
        job.setMapperClass(WordCountMapper.class);
        job.setReducerClass(WordCountReducer.class);
        job.setOutputKeyClass(Text.class);
        job.setOutputValueClass(IntWritable.class);
    System.exit(job.waitForCompletion(true)?0:1);
}
```

Command line options

```
public static void main(String[] args) throws Exception {
    if(args.length !=2){
        System.err.println("Invalid Command");
        System.err.println("Usage: WordCount <input path> <output path>");
        System.exit(0);
    }
    Configuration conf = new Configuration();
    Job job = new Job(conf, "wordcount");
    job.setJarByClass(WordCount.class);
           job.setJobName("Word Count");
    FileInputFormat.addInputPath(job, new Path(args[0]));
    FileOutputFormat.setOutputPath(job, new Path(args[1]));
    job.setMapperClass(WordCountMapper.class);
    job.setReducerClass(WordCountReducer.class);
    job.setOutputKeyClass(Text.class);
    job.setOutputValueClass(IntWritable.class);
    System.exit(job.waitForCompletion(true)?0:1);
}
```

You create a JAR with all your code and run the job at the command line

Command line options

Run the job at the command line

```
$ hadoop jar <JARpath> <mainclass> <inputFile> <outputDir>
```

Here's what the output will look like

```
16/04/17 14:51:08 INFO client.RMProxy: Connecting to ResourceManager at localhost/127.0.0.1:8032
16/04/17 14:51:09 WARN mapreduce.JobResourceUploader: Hadoop command-line option parsing not performed
. Implement the Tool interface and execute your application with ToolRunner to remedy this.
16/04/17 14:51:09 INFO input.FileInputFormat: Total input paths to process : 1
16/04/17 14:51:09 INFO mapreduce.JobSubmitter: number of splits:1
16/04/17 14:51:09 INFO mapreduce.JobSubmitter: Submitting tokens for job: job_1460805726946_0012
16/04/17 14:51:10 INFO impl.YarnClientImpl: Submitted application application_1460805726946_0012
16/04/17 14:51:10 INFO mapreduce.Job: The url to track the job: http://192.168.0.113:8088/proxy/applic
ation_1460805726946_0012/
16/04/17 14:51:10 INFO mapreduce.Job: Running job: job_1460805726946_0012
16/04/17 14:51:18 INFO mapreduce.Job: Job job_1460805726946_0012 running in uber mode : false
16/04/17 14:51:18 INFO mapreduce.Job:  map 0% reduce 0%
16/04/17 14:51:24 INFO mapreduce.Job:  map 100% reduce 0%
16/04/17 14:51:28 INFO mapreduce.Job:  map 100% reduce 100%
16/04/17 14:51:28 INFO mapreduce.Job: Job job_1460805726946_0012 completed successfully
```

Command line options

```
$ hadoop jar <JARpath> <mainclass> <inputFile> <outputDir>
```

```
16/04/17 14:51:08 INFO client.RMProxy: Connecting to ResourceManager at localhost/127.0.0.1:8032
16/04/17 14:51:09 WARN mapreduce.JobResourceUploader: Hadoop command-line option parsing not performed
. Implement the Tool interface and execute your application with ToolRunner to remedy this.
16/04/17 14:51:09 INFO input.FileInputFormat: Total input paths to process : 1
16/04/17 14:51:09 INFO mapreduce.JobSubmitter: number of splits:1
16/04/17 14:51:09 INFO mapreduce.JobSubmitter: Submitting tokens for job: job_1460805726946_0012
16/04/17 14:51:10 INFO impl.YarnClientImpl: Submitted application application_1460805726946_0012
16/04/17 14:51:10 INFO mapreduce.Job: The url to track the job: http://192.168.0.113:8088/proxy/applic
ation_1460805726946_0012/
16/04/17 14:51:10 INFO mapreduce.Job: Running job: job_1460805726946_0012
16/04/17 14:51:18 INFO mapreduce.Job: Job job_1460805726946_0012 running in uber mode : false
16/04/17 14:51:18 INFO mapreduce.Job:  map 0% reduce 0%
16/04/17 14:51:24 INFO mapreduce.Job:  map 100% reduce 0%
16/04/17 14:51:28 INFO mapreduce.Job:  map 100% reduce 100%
16/04/17 14:51:28 INFO mapreduce.Job: Job job_1460805726946_0012 completed successfully
```

Hadoop gives a warning

Command line options

Run the job at the command line

```
$ hadoop jar <JARpath> <mainclass> <inputFile> <outputDir>
```

```
16/04/17 14:51:09 WARN mapreduce.JobResourceUploader
: Hadoop command-line option parsing not performed.
Implement the Tool interface and execute your applic
ation with ToolRunner to remedy this.
```

When you run a job from the command line , you can specify options to override the default configuration

Command line options

Run the job at the command line

```
$ hadoop jar
-D mapred.reduce.tasks=2\
<JARpath> <mainclass> <inputFile> <out
```

```
16/04/17 14:51:09 WARN mapreduce.JobResourceUploader
: Hadoop command-line option parsing not performed.
Implement the Tool interface and execute your applic
ation with ToolRunner to remedy this.
```

For example, here we set the number of reducers our job should use

Command line options

Run the job at the command line

```
$ hadoop jar
-D mapred.reduce.tasks=2\
<JARpath> <mainclass> <inputFile> <outputDir>
```

```
16/04/17 14:51:09 WARN mapreduce.JobResourceUploader
: Hadoop command-line option parsing not performed.
Implement the Tool interface and execute your applic
ation with ToolRunner to remedy this.
```

The way we've currently set up our job, this option won't be parsed and used

Command line options

Run the job at the command line

```
$ hadoop jar
-D mapred.reduce.tasks=2\
<JARpath> <mainclass> <inputFile> <outputDir>
```

```
16/04/17 14:51:09 WARN mapreduce.JobResourceUploader
: Hadoop command-line option parsing not performed.
Implement the Tool interface and execute your applic
ation with ToolRunner to remedy this.
```

Our Main class needs to implement the Tool Interface

Command line options

Run the job at the command line

```
$ hadoop jar
-D mapred.reduce.tasks=2\
<JARpath> <mainclass> <inputFile> <outputDir>
```

```
16/04/17 14:51:09 WARN mapreduce.JobResourceUploader
: Hadoop command-line option parsing not performed.
Implement the Tool interface and execute your applic
ation with ToolRunner to remedy this.
```

Our Main class needs to implement the Tool Interface

Command line options

Run the job at the command line

```
$ hadoop jar
-D mapred.reduce.tasks=2\
<JARpath> <mainclass> <inputFile> <outputDir>
```

```
16/04/17 14:51:09 WARN mapreduce.JobResourceUploader
: Hadoop command-line option parsing not performed.
Implement the Tool interface and execute your applic
ation with ToolRunner to remedy this.
```

The Interface has a run() where our Job setup will be implemented

Command line options

Run the job at the command line

```
$ hadoop jar
-D mapred.reduce.tasks=2\
<JARpath> <mainclass> <inputFile> <outputDir>
```

```
16/04/17 14:51:09 WARN mapreduce.JobResourceUploader
: Hadoop command-line option parsing not performed.
Implement the Tool interface and execute your applic
ation with ToolRunner to remedy this.
```

In the main() method we will user ToolRunner.run() to execute the Job

Command line options

Run the job at the command line

```
$ hadoop jar
-D mapred.reduce.tasks=2\
<JARpath> <mainclass> <inputFile> <outputDir>
```

```
16/04/17 14:51:09 WARN mapreduce.JobResourceUploader
: Hadoop command-line option parsing not performed.
Implement the Tool interface and execute your applic
ation with ToolRunner to remedy this.
```

ToolRunner in turn uses GenericOptionsParser

Command line options

Run the job at the command line

Tool

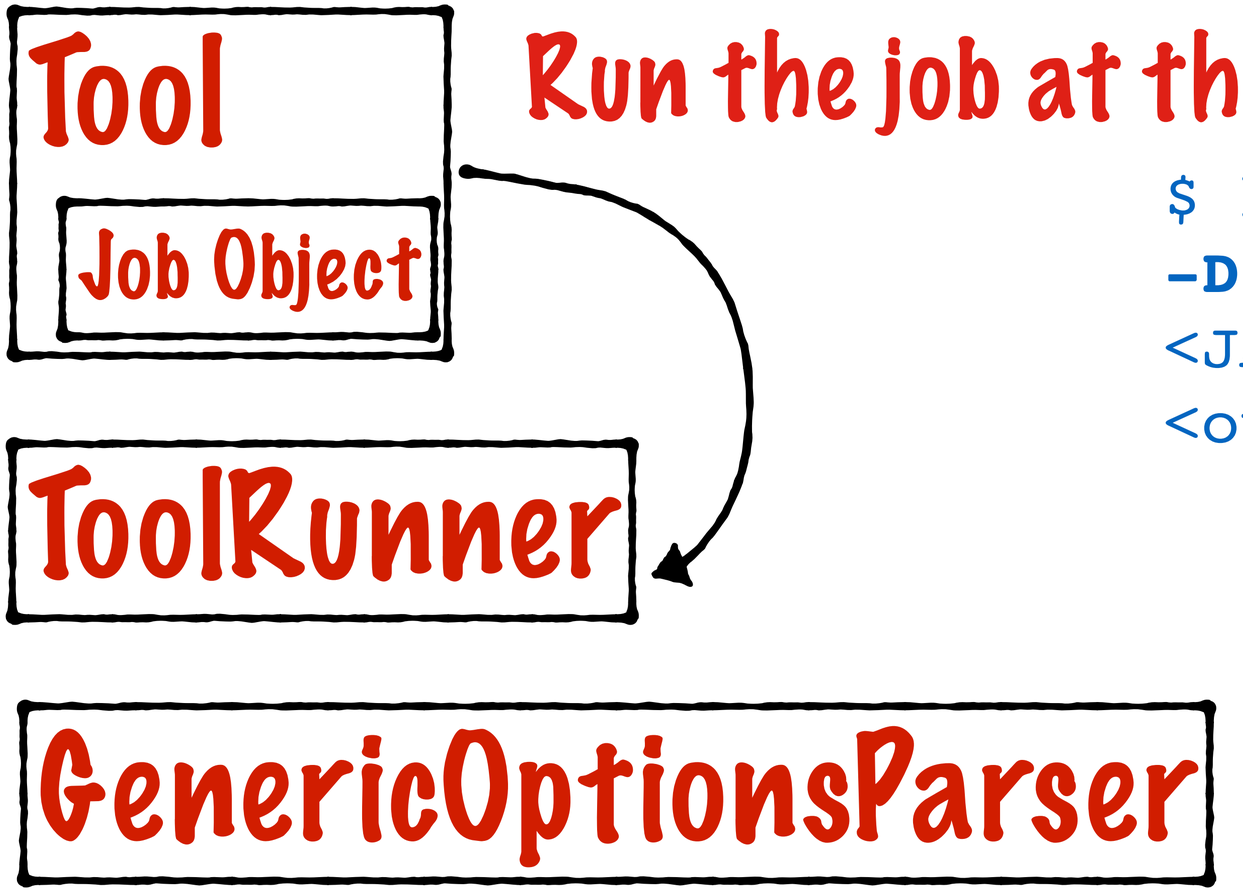
ToolRunner

GenericOptionsParser

```
$ hadoop jar
-D mapred.reduce.tasks=2\
<JARpath> <mainclass> <inputFile> \
<outputDir>
```

The Main class implements Tool

Command line options



Run the job at the command line

```
$ hadoop jar
-D mapred.reduce.tasks=2\
<JARpath> <mainclass> <inputFile> \
<outputDir>
```

This Tool is passed on to ToolRunner.run()

Command line options

Run the job at the command line

ToolRunner.run()

Tool Object

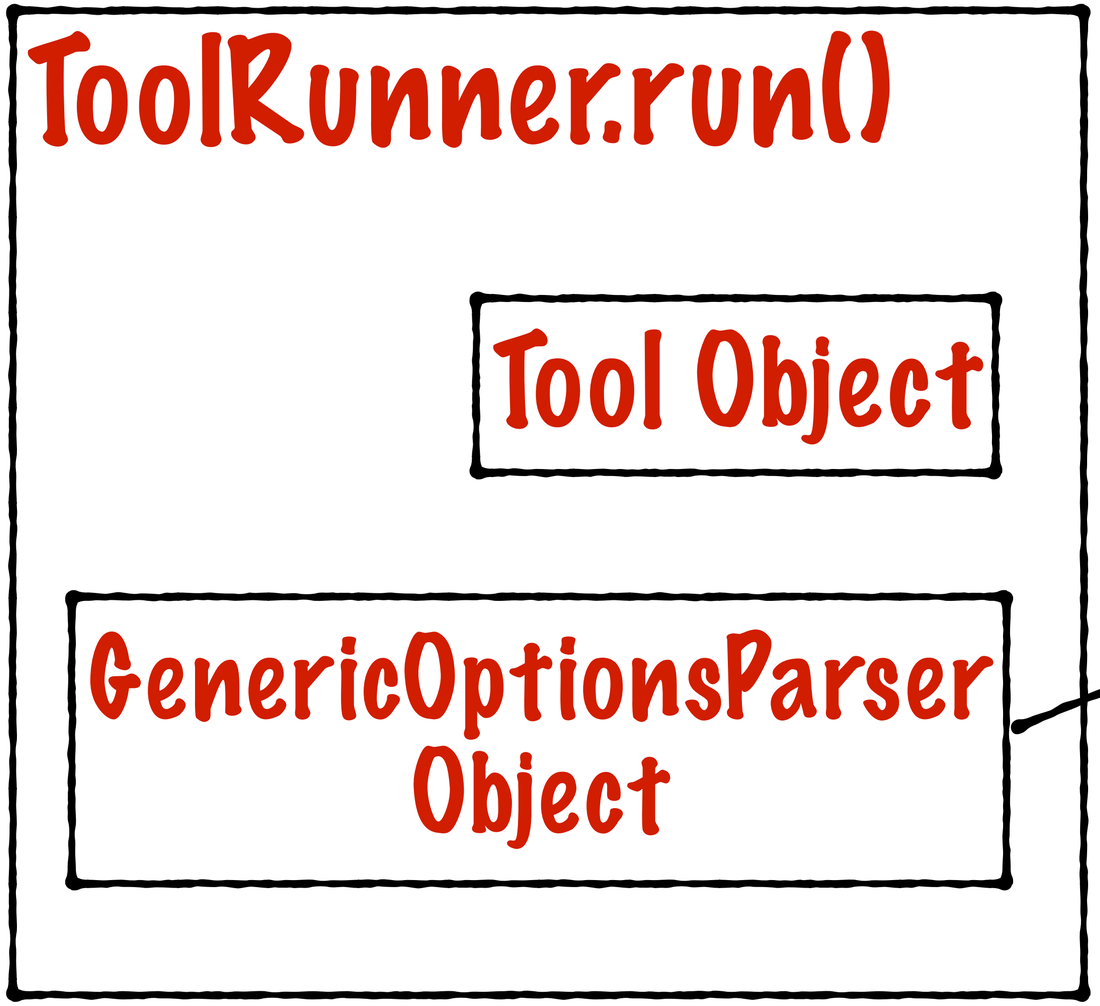
GenericOptionsParser

```
$ hadoop jar
-D mapred.reduce.tasks=2\
<JARpath> <mainclass> <inputFile> \
<outputDir>
```

This will in turn instantiate a GenericOptionsParser

Command line options

Run the job at the command line

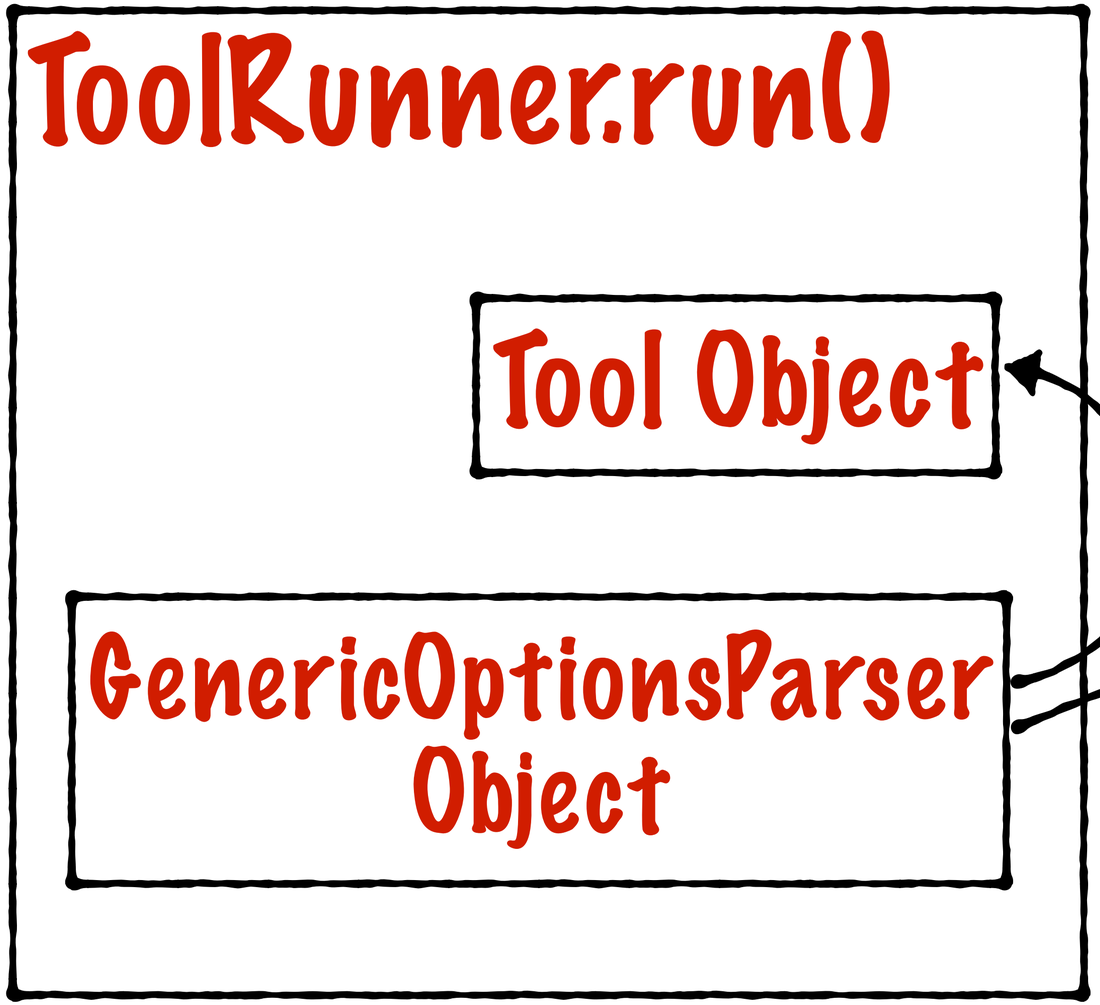



```
$ hadoop jar
-D mapred.reduce.tasks=2\
<JARpath> <mainclass> <inputFile> \
<outputDir>
```

The parser will parse the command line options

Command line options

Run the job at the command line

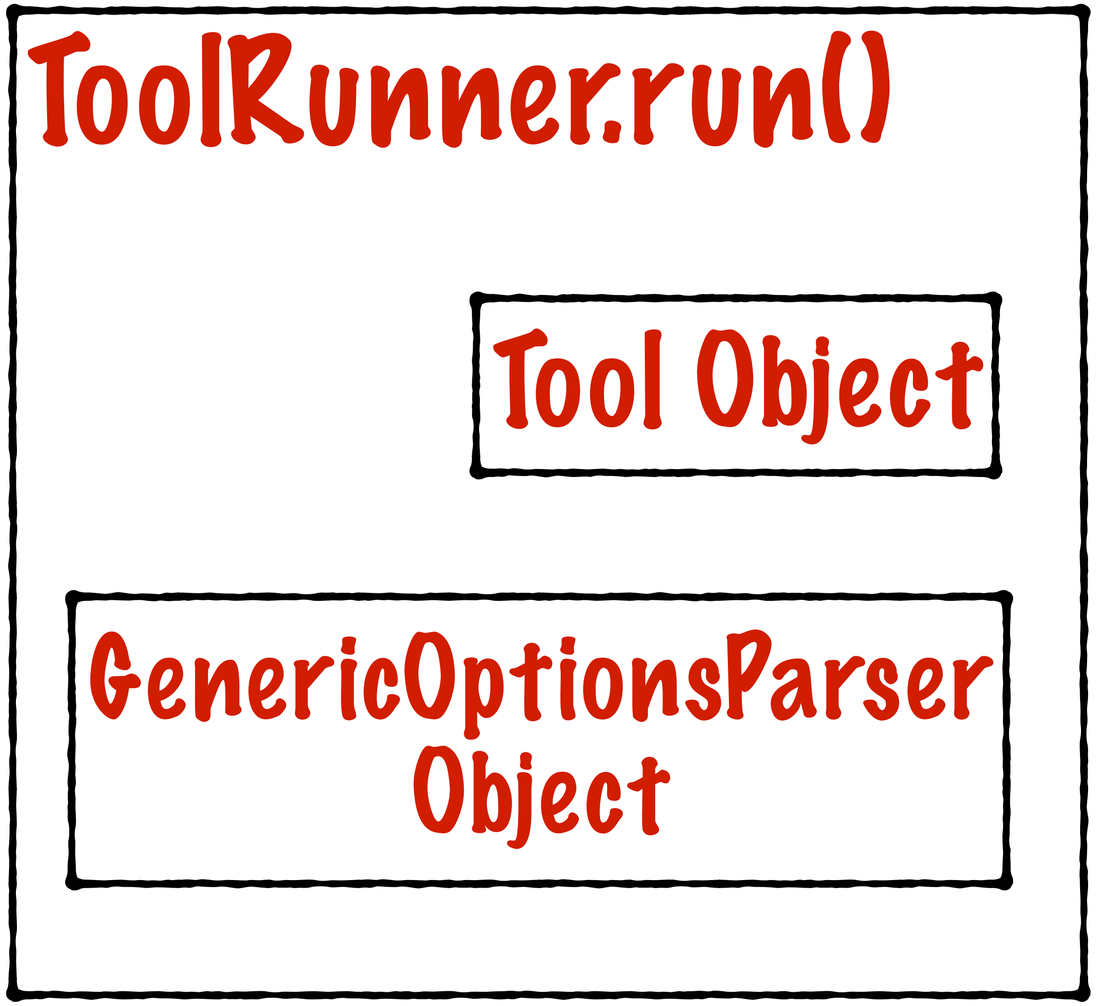



```
$ hadoop jar
-D mapred.reduce.tasks=2\
<JARpath> <mainclass> <inputFile> \
<outputDir>
```

Any configuration options will be set on the Tool's and in turn the Job's configuration

Command line options

Run the job at the command line




```
$ hadoop jar
-D mapred.reduce.tasks=2\
<JARpath> <mainclass> <inputFile> \
<outputDir>
```

Any remaining arguments are passed on to the Tool's own run() method

Command line options

Tool

ToolRunner

GenericOptionsParser

```
$ hadoop jar
-D mapred.reduce.tasks=2\
<JARpath> <mainclass> <inputFile> \
<outputDir>
```

Let's see how to change our

Main class now

Command line options

```
public class WordCountwithTool extends Configured implements Tool {
    @Override
    public int run(String[] args) throws Exception{

        if(args.length !=2){
    System.err.println("Invalid Command");
    System.err.println("Usage: <input path> <output path>");
    return -1;
     }

Job job = Job.getInstance(getConf());
job.setJobName("wordCountWithTool");
job.setJarByClass(WordCountwithTool.class);

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

job.setMapperClass(WordCountMapper.class);
job.setReducerClass(WordCountReducerWithMin.class);

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

return job.waitForCompletion(true) ? 0 : 1;

}

public static void main(String[] args) throws Exception {
    int exitCode = ToolRunner.run(new WordCountwithTool(), args);
    System.exit(exitCode);
}
}
```

Our Main Class now implements the Tool Interface

Command line options

```
public class WordCountwithTool extends Configured
implements Tool {
        @Override
    public int run(String[] args) throws Exception{

        if(args.length !=2){
    System.err.println("Invalid Command");
    System.err.println("Usage: <input path> <output path>");
    return -1;
     }

Job job = Job.getInstance(getConf());
job.setJobName("wordCountWithTool");
job.setJarByClass(WordCountwithTool.class);

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

job.setMapperClass(WordCountMapper.class);
job.setReducerClass(WordCountReducerWithMin.class);

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

return job.waitForCompletion(true) ? 0 : 1;

    }

    public static void main(String[] args) throws Exception {
        int exitCode = ToolRunner.run(new WordCountwithTool(), args);
        System.exit(exitCode);
    }
}
```

The Tool Interface extends the Configurable Interface

Command line options

```
public class WordCountwithTool extends Configured
implements Tool {

        @Override
    public int run(String[] args) throws Exception{

        if(args.length !=2){
    System.err.println("Invalid Command");
    System.err.println("Usage: <input path> <output path>");
    return -1;
     }

Job job = Job.getInstance(getConf());
job.setJobName("wordCountWithTool");
job.setJarByClass(WordCountwithTool.class);

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

job.setMapperClass(WordCountMapper.class);
job.setReducerClass(WordCountReducerWithMin.class);

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

return job.waitForCompletion(true) ? 0 : 1;

    }

    public static void main(String[] args) throws Exception {
        int exitCode = ToolRunner.run(new WordCountwithTool(), args);
        System.exit(exitCode);
    }
}
```

All implementations of Tool need to also implement the Configurable Interface

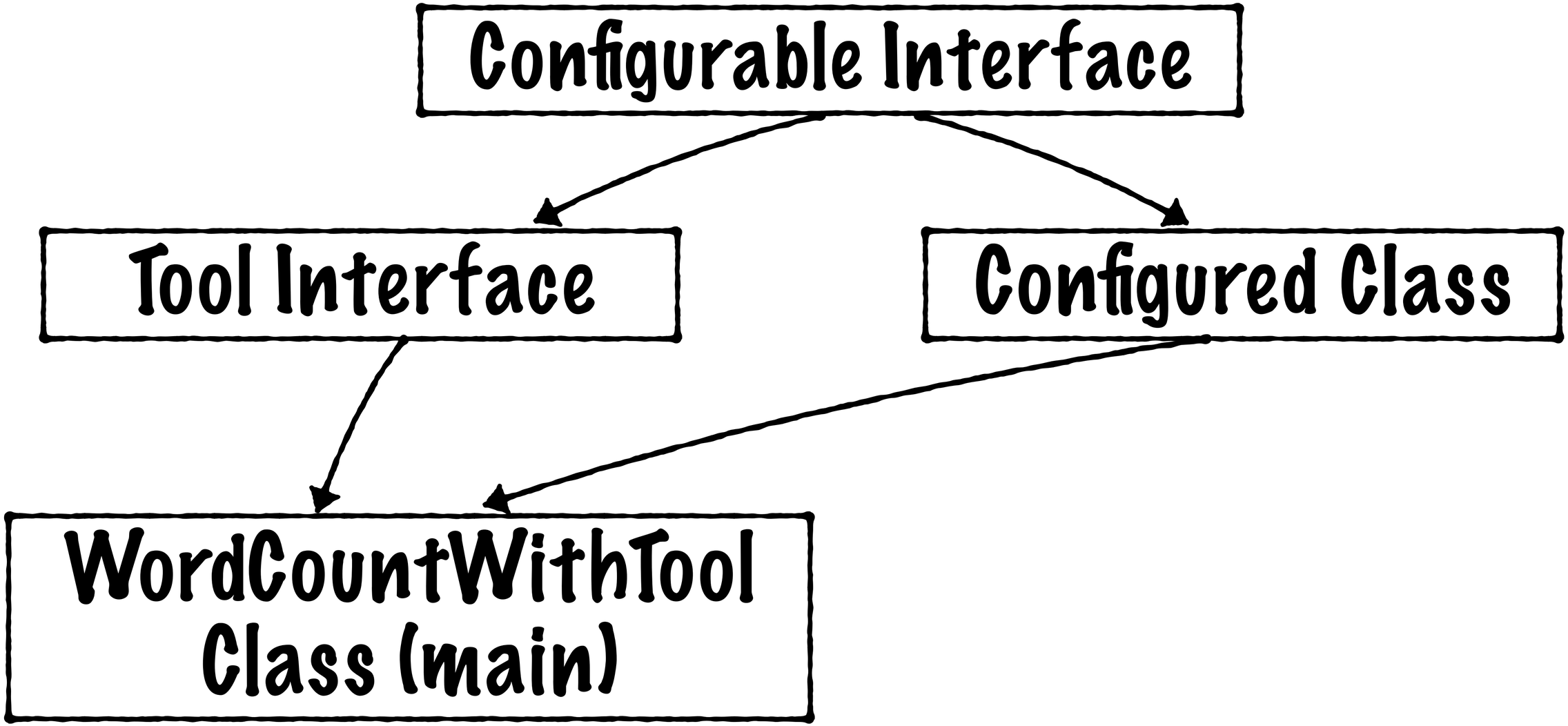
Command line options

```
public class WordCountwithTool extends Configured
implements Tool {
```

```
        @Override
    public int run(String[] args) throws Exception{

        if(args.length !=2){
    System.err.println("Invalid Command");
    System.err.println("Usage: <input path> <output path>");
    return -1;
     }

Job job = Job.getInstance(getConf());
job.setJobName("wordCountWithTool");
job.setJarByClass(WordCountwithTool.class);

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

job.setMapperClass(WordCountMapper.class);
job.setReducerClass(WordCountReducerWithMin.class);

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

return job.waitForCompletion(true) ? 0 : 1;

    }

    public static void main(String[] args) throws Exception {
        int exitCode = ToolRunner.run(new WordCountwithTool(), args);
        System.exit(exitCode);
    }
}
```

Configured is an implementation of the Configurable Interface

Command line options



Command line options

```
public class WordCountwithTool extends Configured implements Tool {

      @Override
  public int run(String[] args) throws Exception{


        if(args.length !=2){
    System.err.println("Invalid Command");
    System.err.println("Usage: <input path> <output path>");
    return -1;
     }

Job job = Job.getInstance(getConf());
job.setJobName("wordCountWithTool");
job.setJarByClass(WordCountwithTool.class);

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

job.setMapperClass(WordCountMapper.class);
job.setReducerClass(WordCountReducerWithMin.class);

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

return job.waitForCompletion(true) ? 0 : 1;


        }
    public static void main(String[] args) throws Exception {
        int exitCode = ToolRunner.run(new WordCountwithTool(), args);
        System.exit(exitCode);
    }
}
```

All our Job setup now sits inside the run() method of the Tool

Command line options

```
public class WordCountwithTool extends Configured implements Tool {
    @Override
    public int run(String[] args) throws Exception{
      if(args.length !=2){
    System.err.println("Invalid Command");
    System.err.println("Usage: <input path> <output path>");
    return -1;
     }

    Job job = Job.getInstance(getConf());
    job.setJobName("wordCountWithTool");
    job.setJarByClass(WordCountwithTool.class);

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

    job.setMapperClass(WordCountMapper.class);
    job.setReducerClass(WordCountReducerWithMin.class);

    Path inputFilePath = new Path(args[0]);
    Path outputFilePath = new Path(args[1]);
    FileInputFormat.addInputPath(job, inputFilePath);
    FileOutputFormat.setOutputPath(job, outputFilePath);
    return job.waitForCompletion(true) ? 0 : 1;

    }

    public static void main(String[] args) throws Exception {
        int exitCode = ToolRunner.run(new WordCountwithTool(), args);
        System.exit(exitCode);
    }
}
```

This method needs to return an int which will be used as the System exit code

Command line options

```
public class WordCountwithTool extends Configured implements Tool {
    @Override
    public int run(String[] args) throws Exception{
      if(args.length !=2){
    System.err.println("Invalid Command");
    System.err.println("Usage: <input path> <output path>");
    return -1;
     }

    Job job = Job.getInstance(getConf());
    job.setJobName("wordCountWithTool");
    job.setJarByClass(WordCountwithTool.class);

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

    job.setMapperClass(WordCountMapper.class);
    job.setReducerClass(WordCountReducerWithMin.class);

    Path inputFilePath = new Path(args[0]);
    Path outputFilePath = new Path(args[1]);
    FileInputFormat.addInputPath(job, inputFilePath);
    FileOutputFormat.setOutputPath(job, outputFilePath);
 return job.waitForCompletion(true) ? 0 : 1;

    }
```

In the Main method, we'll call Toolrunner.run() and pass it our Tool class

```
  public static void main(String[] args) throws Exception {
          int exitCode = ToolRunner.run(new WordCountwithTool(), args);
          System.exit(exitCode);
      }
}
```

Command line options

```
public class WordCountwithTool extends Configured implements Tool {
    @Override
    public int run(String[] args) throws Exception{
      if(args.length !=2){
    System.err.println("Invalid Command");
    System.err.println("Usage: <input path> <output path>");
    return -1;
     }

    Job job = Job.getInstance(getConf());
    job.setJobName("wordCountWithTool");
    job.setJarByClass(WordCountwithTool.class);

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

    job.setMapperClass(WordCountMapper.class);
    job.setReducerClass(WordCountReducerWithMin.class);

    Path inputFilePath = new Path(args[0]);
    Path outputFilePath = new Path(args[1]);
    FileInputFormat.addInputPath(job, inputFilePath);
    FileOutputFormat.setOutputPath(job, outputFilePath);
 return job.waitForCompletion(true) ? 0 : 1;

    }
```

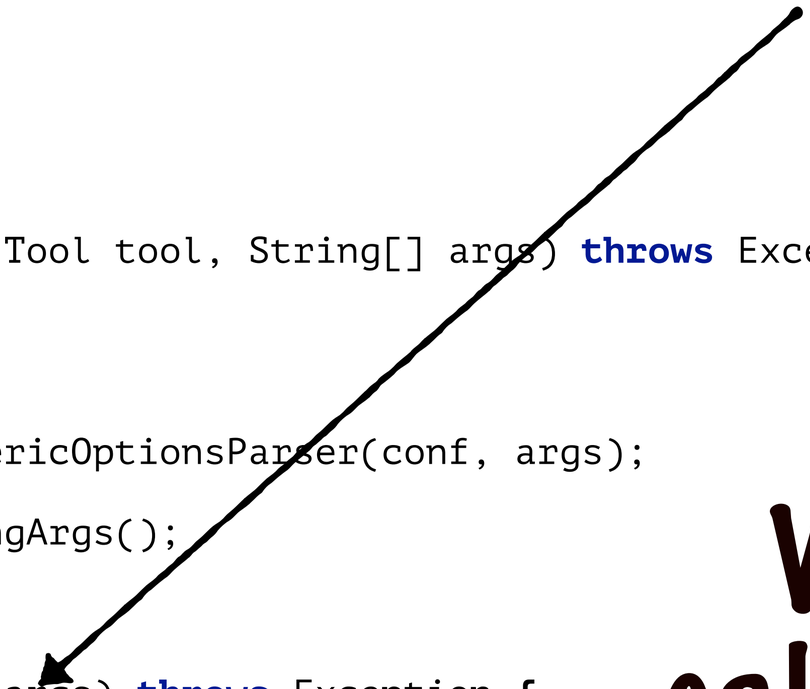
Let's look at the ToolRunner to understand what happens next

```
  public static void main(String[] args) throws Exception {
          int exitCode = ToolRunner.run(new WordCountwithTool(), args);
          System.exit(exitCode);
      }
}
```

Command line options

ToolRunner

```
ToolRunner.run(new WordCountwithTool(), args);
```



```
public class ToolRunner {
    public ToolRunner() {}

    public static int run(Configuration conf, Tool tool, String[] args) throws Exception {
        if(conf == null) {
            conf = new Configuration();
        }

        GenericOptionsParser parser = new GenericOptionsParser(conf, args);
        tool.setConf(conf);
        String[] toolArgs = parser.getRemainingArgs();
        return tool.run(toolArgs);
    }

    public static int run(Tool tool, String[] args) throws Exception {
                return run(tool.getConf(), tool, args);
       }

    public static void printGenericCommandUsage(PrintStream out) {..}

    public static boolean confirmPrompt(String prompt) throws IOException {...}
}
```

We are calling this method

Command line options

ToolRunner

```
ToolRunner.run(new WordCountwithTool(), args);
```

```
public class ToolRunner {
    public ToolRunner() {}

    public static int run(Configuration conf, Tool tool, String[] args) throws Exception {
      if(conf == null) {
       conf = new Configuration();
       }
      GenericOptionsParser parser = new GenericOptionsParser(conf, args);
      tool.setConf(conf);
      String[] toolArgs = parser.getRemainingArgs();
      return tool.run(toolArgs);
       }

    public static int run(Tool tool, String[] args) throws Exception {
                return run(tool.getConf(), tool, args);
        }

   public static void printGenericCommandUsage(PrintStream out) {..}

   public static boolean confirmPrompt(String prompt) throws IOException {...}
}
```

Command line options

ToolRunner

```
ToolRunner.run(new WordCountwithTool(), args);
```

```
public class ToolRunner {
    public ToolRunner() {}

public static int run(Configuration conf, Tool tool, String[] args) throws Exception {
     if(conf == null) {
            conf = new Configuration();}

   GenericOptionsParser parser = new GenericOptionsParser(conf, args);
   tool.setConf(conf);

    String[] toolArgs = parser.getRemainingArgs();
  return tool.run(toolArgs);
    }

    public static int run(Tool tool, String[] args) throws Exception {
           return run(tool.getConf(), tool, args);
    }
   public static void printGenericCommandUsage(PrintStream out) {..}
   public static boolean confirmPrompt(String prompt) throws IOException {…}
}
```

The run() method instantiates a GenericOptionsParser

Command line options

ToolRunner

```
ToolRunner.run(new WordCountwithTool(), args);
```

```
public class ToolRunner {
    public ToolRunner() {}

public static int run(Configuration conf, Tool tool, String[] args) throws Exception {
     if(conf == null) {
            conf = new Configuration();}
```

```
   GenericOptionsParser parser = new GenericOptionsParser(conf, args);
   tool.setConf(conf);
```

```
    String[] toolArgs = parser.getRemainingArgs();
  return tool.run(toolArgs);
    }
```

```
      public static int run(Tool tool, String[] args) throws Exception {
            return run(tool.getConf(), tool, args);
      }
   public static void printGenericCommandUsage(PrintStream out) {..}
   public static boolean confirmPrompt(String prompt) throws IOException {…}
}
```

This will read the command line arguments and set them in the Configuration object of the Tool

Command line options

ToolRunner

```
ToolRunner.run(new WordCountwithTool(), args);
```

```
public class ToolRunner {
    public ToolRunner() {}

public static int run(Configuration conf, Tool tool, String[] args) throws Exception {
     if(conf == null) {
            conf = new Configuration();}
      GenericOptionsParser parser = new GenericOptionsParser(conf, args);

      tool.setConf(conf);

   String[] toolArgs = parser.getRemainingArgs();
     return tool.run(toolArgs);
     }
```

```
    public static int run(Tool tool, String[] args) throws Exception {
            return run(tool.getConf(), tool, args);
    }

  public static void printGenericCommandUsage(PrintStream out) {..}

  public static boolean confirmPrompt(String prompt) throws IOException {…}
}
```

We get the remaining arguments from the parser and pass them on to the Tool's run() method

Command line options

ToolRunner

```
ToolRunner.run(new WordCountwithTool(), args);
```

```
public class ToolRunner {
    public ToolRunner() {}

public static int run(Configuration conf, Tool tool, String[] args) throws Exception {
     if(conf == null) {
            conf = new Configuration();}
      GenericOptionsParser parser = new GenericOptionsParser(conf, args);

      tool.setConf(conf);

   String[] toolArgs = parser.getRemainingArgs();
     return tool.run(toolArgs);
     }
```

```
    public static int run(Tool tool, String[] args) throws Exception {
            return run(tool.getConf(), tool, args);
    }

  public static void printGenericCommandUsage(PrintStream out) {..}

  public static boolean confirmPrompt(String prompt) throws IOException {…}
}
```

This will now run our Job with all the options, both from the command line and the Job object

Command line options

ToolRunner

```
ToolRunner.run(new WordCountwithTool(), args);
```

```
public class ToolRunner {
    public ToolRunner() {}

public static int run(Configuration conf, Tool tool, String[] args) throws Exception {
     if(conf == null) {
            conf = new Configuration();}
```

```
   GenericOptionsParser parser = new GenericOptionsParser(conf, args);
   tool.setConf(conf);
```

```
    String[] toolArgs = parser.getRemainingArgs();
  return tool.run(toolArgs);
    }
```

```
    public static int run(Tool tool, String[] args) throws Exception {
            return run(tool.getConf(), tool, args);
    }
  public static void printGenericCommandUsage(PrintStream out) {..}
  public static boolean confirmPrompt(String prompt) throws IOException {…}
}
```

You can directly use a GenericOptionsParser in your Main class instead of through Tool and ToolRunner

Command line options

ToolRunner

```
ToolRunner.run(new WordCountwithTool(), args);
```

```
public class ToolRunner {
    public ToolRunner() {}

public static int run(Configuration conf, Tool tool, String[] args) throws Exception {
     if(conf == null) {
            conf = new Configuration();}

   GenericOptionsParser parser = new GenericOptionsParser(conf, args);
   tool.setConf(conf);

    String[] toolArgs = parser.getRemainingArgs();
  return tool.run(toolArgs);
    }

    public static int run(Tool tool, String[] args) throws Exception {
            return run(tool.getConf(), tool, args);
    }
   public static void printGenericCommandUsage(PrintStream out) {..}
   public static boolean confirmPrompt(String prompt) throws IOException {…}
}
```

However, it is recommended to use the Tool Interface to implement your Main Class and not to directly use GenericOptionsParser

Command line options

Tool

ToolRunner

GenericOptionsParser

```
$ hadoop jar
-D mapred.reduce.tasks=2\
<JARpath> <mainclass> <inputFile> \
<outputDir>
```

Once you have used these 3 helper classes, you can really control the configuration of your Job

Command line options

Tool

ToolRunner

GenericOptionsParser

```
$ hadoop jar
-D mapred.reduce.tasks=2\
<JARpath> <mainclass> <inputFile> \
<outputDir>
```

Over and above all the Hadoop options, you can even set custom parameters

Command line options

```
$ hadoop jar
-D reducer.minWordCount=5\
<JARpath> <mainclass> <inputFile> \
<outputDir>
```

You can do this when you want to pass in options to the Mapper/Reducer which in turn will govern their behavior

Command line options

```
$ hadoop jar
-D reducer.minWordCount=5\
<JARpath> <mainclass> <inputFile> \
<outputDir>
```

Suppose in your word count job, you want to allow the user to set a count threshold for words

Command line options

```
$ hadoop jar
-D reducer.minWordCount=5\
<JARpath> <mainclass> <inputFile> \
<outputDir>
```

In the reducer you can ask for the value of this config parameter and use it

```
public class WordCountReducerWithMin extends Reducer<Text,
IntWritable,Text,IntWritable> {
    @Override
    public void reduce(Text key, Iterable<IntWritable> values, Context context)
throws IOException, InterruptedException{

        int minCount=0;
        Configuration config = context.getConfiguration();
        try {
            minCount = Integer.parseInt(config.get("reducer.minWordCount"));
        } catch (Exception e){

        }

        int count = 0;
        for (IntWritable value:values)
        {
            count += value.get();
        }

        if (count>=minCount) {
```

Command line options

```
ublic class WordCountReducerWithMin extends Reducer<Text, IntWritable,Text,IntWritable> {
   @Override
   public void reduce(Text key, Iterable<IntWritable> values, Context context) throws IOException, InterruptedException{

      int minCount=0;
      Configuration config = context.getConfiguration();
      try {
            minCount = Integer.parseInt(config.get("reducer.minWordCount"));
           } catch (Exception e){}

      int count = 0;
      for (IntWritable value:values)
      {
          count += value.get();
      }

      if (count>=minCount) {

          context.write(key, new IntWritable(count));
      }
   }
```

If the user specifies a minCount, we'll get it from the context

Command line options

```
ublic class WordCountReducerWithMin extends Reducer<Text, IntWritable, Text, IntWritable> {
   @Override
   public void reduce(Text key, Iterable<IntWritable> values, Context context) throws IOException, InterruptedException{

        int minCount=0;
  Configuration config = context.getConfiguration();
  try {
        minCount = Integer.parseInt(config.get("reducer.minWordCount"));

      } catch (Exception e){}

        int count = 0;
        for (IntWritable value:values)
        {
            count += value.get();
        }

      if (count>=minCount) {

        context.write(key, new IntWritable(count));
            }
  }
```

We only write out the <word, count> if it satisfies the threshold

MR Job Configuration

Let's say you have set up a MR job
You can configure the MR job in 2 ways

Command line options

Job object

MR Job Configuration

Let's say you have set up a MR job
You can configure the MR job in 2 ways

Command line options

Job object

MR Job Configuration

Job object

For every MR Job, a Job object
is setup in the **Main Class**

The Job Object has a bunch of
Configurable Options

MR Job Configuration

Job object

All of these have a default configuration

The user can control them by setting them to a specific class

Configurable Options

```
setInputFormatClass()
setMapOutputKeyClass()
setMapOutputValueClass()
setOutputKeyClass()
setOutputValueClass()
setMapperClass()
setCombinerClass()
setPartitionerClass()
setSortComparatorClass()
setGroupingComparatorClass()
setReducerClass()
setOutputFormatClass()
```

MR Job Configuration

Job object

Configurable Options

Here are the options you would normally leave to the default setting

```
setInputFormatClass()
setMapOutputKeyClass()
setMapOutputValueClass()
setOutputKeyClass()
setOutputValueClass()
setMapperClass()
setCombinerClass()
setPartitionerClass()
setSortComparatorClass()
setGroupingComparatorClass()
setReducerClass()
setOutputFormatClass()
```

MR Job Configuration

Job object

Configurable Options

Here are the options you would set

```
setInputFormatClass()
setMapOutputKeyClass()
setMapOutputValueClass()
setOutputKeyClass()
setOutputValueClass()
setMapperClass()
setCombinerClass()
setPartitionerClass()
setSortComparatorClass()
setGroupingComparatorClass()
setReducerClass()
setOutputFormatClass()
```

MR Job Configuration

Job object

Configurable Options

```
setInputFormatClass()
setMapOutputKeyClass()
setMapOutputValueClass()
setOutputKeyClass()
setOutputValueClass()
setMapperClass()
setCombinerClass()
setPartitionerClass()
setSortComparatorClass()
setGroupingComparatorClass()
setReducerClass()
setOutputFormatClass()
```

Let's go through each of them

MR Job Configuration

Job object

Input and Output formats
are by default set to

TextInputFormat.Class

TextOutputFormat.Class

Configurable Options

setInputFormatClass()

setMapOutputKeyClass()
setMapOutputValueClass()
setOutputKeyClass()
setOutputValueClass()
setMapperClass()
setCombinerClass()
setPartitionerClass()
setSortComparatorClass()
setGroupingComparatorClass()
setReducerClass()

setOutputFormatClass()

MR Job Configuration

Job object

Configurable Options

With this option input is expected to be in the form of text files

`setInputFormatClass()`

TextInputFormat.Class

Outputs will also be text files

`setOutputFormatClass()`

TextOutputFormat.Class

MR Job Configuration

Job object

You can set this to any class that extends the **InputFormat** class

Configurable Options

`setInputFormatClass()`

TextInputFormat.Class

MR Job Configuration

Job object

This class decides how to break up the input files into InputSplits

Each InputSplit will be processed by 1 mapper

Configurable Options

`setInputFormatClass()`

InputFormat class

MR Job Configuration

Job object

This class will also read each record in the input file and convert it to a <key,value> pair for the mapper

Configurable Options

`setInputFormatClass()`

InputFormat class

MR Job Configuration

Job object

Configurable Options

Similarly, for the output you can use any class that extends the **OutputFormat** class

`setInputFormatClass()`

InputFormat class

`setOutputFormatClass()`

OutputFormat class

MR Job Configuration

Job object

Configurable Options

InputFormat class

`setInputFormatClass()`

`setMapOutputKeyClass()`

`setMapOutputValueClass()`

`setOutputKeyClass()`

`setOutputValueClass()`

`setMapperClass()`

`setCombinerClass()`

`setPartitionerClass()`

`setSortComparatorClass()`

`setGroupingComparatorClass()`

`setReducerClass()`

OutputFormat class

`setOutputFormatClass()`

MR Job Configuration

Job object

Configurable Options

These are classes that the user would normally always set

```
setInputFormatClass()
setMapOutputKeyClass()
setMapOutputValueClass()
setOutputKeyClass()
setOutputValueClass()
setMapperClass()
setCombinerClass()
setPartitionerClass()
setSortComparatorClass()
setGroupingComparatorClass()
setReducerClass()
setOutputFormatClass()
```

MR Job Configuration

Job object

Configurable Options

If you don't set them, they will just act like Identity functions

setInputFormatClass()

setMapOutputKeyClass()

setMapOutputValueClass()

setOutputKeyClass()

setOutputValueClass()

setMapperClass()

setCombinerClass()

setPartitionerClass()

setSortComparatorClass()

setGroupingComparatorClass()

setReducerClass()

setOutputFormatClass()

MR Job Configuration

Job object

Configurable Options

The output will basically be the input - unchanged

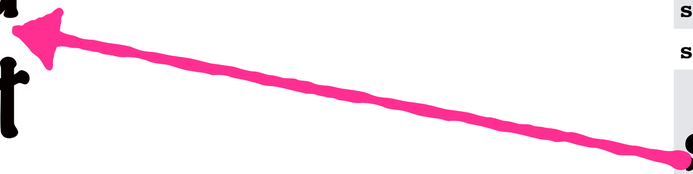
```
setInputFormatClass()
setMapOutputKeyClass()
setMapOutputValueClass()
setOutputKeyClass()
setOutputValueClass()
setMapperClass()
setCombinerClass()
setPartitionerClass()
setSortComparatorClass()
setGroupingComparatorClass()
setReducerClass()
setOutputFormatClass()
```

MR Job Configuration

Job object

Configurable Options

Here you should set a class that extends the Mapper class



setInputFormatClass()

setMapOutputKeyClass()

setMapOutputValueClass()

setOutputKeyClass()

setOutputValueClass()

setMapperClass()

setCombinerClass()

setPartitionerClass()

setSortComparatorClass()

setGroupingComparatorClass()

setReducerClass()

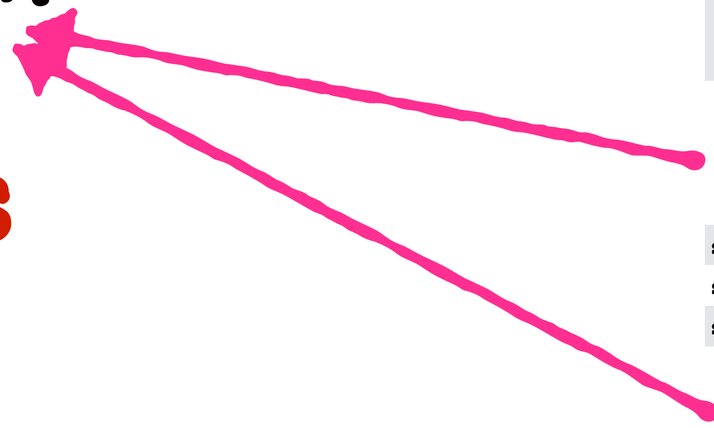
setOutputFormatClass()

MR Job Configuration

Job object

Configurable Options

Here you should set a class that extends the Reducer class



setInputFormatClass()

setMapOutputKeyClass()

setMapOutputValueClass()

setOutputKeyClass()

setOutputValueClass()

setMapperClass()

setCombinerClass()

setPartitionerClass()

setSortComparatorClass()

setGroupingComparatorClass()

setReducerClass()

setOutputFormatClass()

MR Job Configuration

Job object

All of these should be set to a subclass of WritableComparable class

Configurable Options

- setInputFormatClass()
- setMapOutputKeyClass()
- setMapOutputValueClass()
- setOutputKeyClass()
- setOutputValueClass()
- setMapperClass()
- setCombinerClass()
- setPartitionerClass()
- setSortComparatorClass()
- setGroupingComparatorClass()
- setReducerClass()
- setOutputFormatClass()

MR Job Configuration

Job object

Configurable Options

Output classes should be consistent with the Reducer's output type parameters

```
setInputFormatClass()
setMapOutputKeyClass()
setMapOutputValueClass()
setOutputKeyClass()
setOutputValueClass()
setMapperClass()
setCombinerClass()
setPartitionerClass()
setSortComparatorClass()
setGroupingComparatorClass()
setReducerClass()
setOutputFormatClass()
```

MR Job Configuration

Job object

Configurable Options

By default it's assumed that Mapper and Reducer have the same output types

```
setInputFormatClass()
setMapOutputKeyClass()
setMapOutputValueClass()
setOutputKeyClass()
setOutputValueClass()
setMapperClass()
setCombinerClass()
setPartitionerClass()
setSortComparatorClass()
setGroupingComparatorClass()
setReducerClass()
setOutputFormatClass()
```

MR Job Configuration

Job object

Configurable Options

If the Mapper and Reducer have different output types, you need to set these options

```
setInputFormatClass()
setMapOutputKeyClass()
setMapOutputValueClass()
setOutputKeyClass()
setOutputValueClass()
setMapperClass()
setCombinerClass()
setPartitionerClass()
setSortComparatorClass()
setGroupingComparatorClass()
setReducerClass()
setOutputFormatClass()
```

MR Job Configuration

Job object

Configurable Options

The Map Output classes should be consistent with the Mapper's output type parameters

```
setInputFormatClass()
setMapOutputKeyClass()
setMapOutputValueClass()
setOutputKeyClass()
setOutputValueClass()
setMapperClass()
setCombinerClass()
setPartitionerClass()
setSortComparatorClass()
setGroupingComparatorClass()
setReducerClass()
setOutputFormatClass()
```

MR Job Configuration

Job object

Configurable Options

The Map Output classes should also be consistent with the Reducer's input type parameters

```
setInputFormatClass()
setMapOutputKeyClass()
setMapOutputValueClass()
setOutputKeyClass()
setOutputValueClass()
setMapperClass()
setCombinerClass()
setPartitionerClass()
setSortComparatorClass()
setGroupingComparatorClass()
setReducerClass()
setOutputFormatClass()
```

MR Job Configuration

Job object

These classes help perform the intermediate operations between Map and Reduce

Configurable Options

```
setInputFormatClass()
setMapOutputKeyClass()
setMapOutputValueClass()
setOutputKeyClass()
setOutputValueClass()
setMapperClass()
setCombinerClass()
```

```
setPartitionerClass()
```

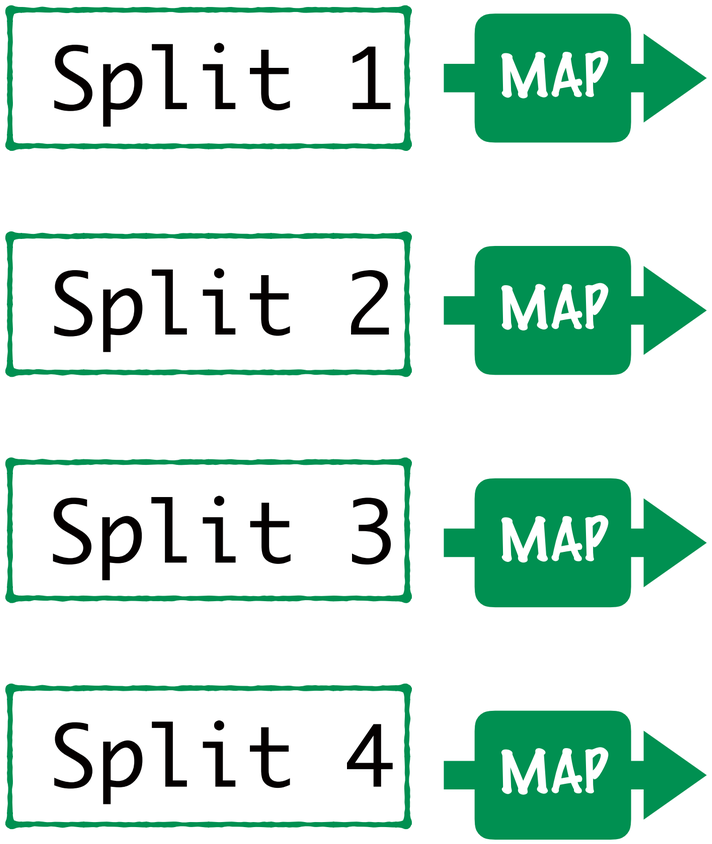
```
setSortComparatorClass()
```

```
setGroupingComparatorClass()
```

```
setReducerClass()
setOutputFormatClass()
```

RECAP

Shuffle and Sort

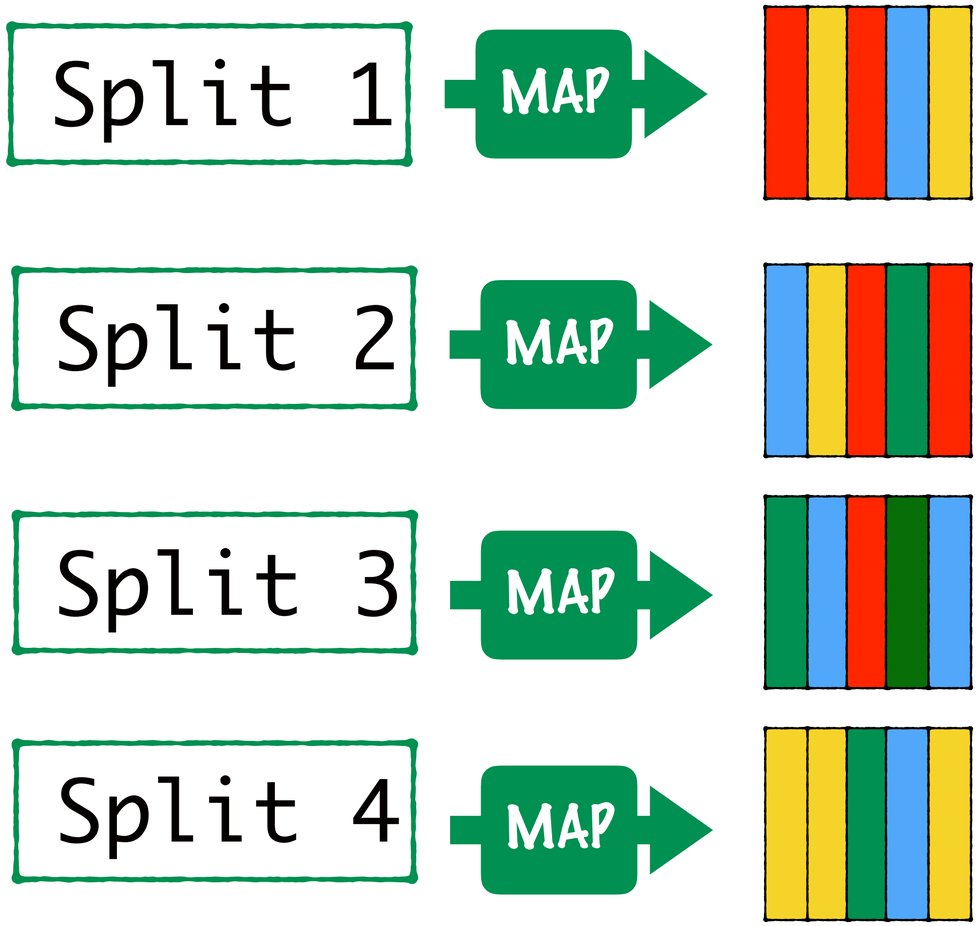


Let's say our input file had been split into 4 blocks by HDFS

There will 4 mappers

RECAP

Shuffle and Sort



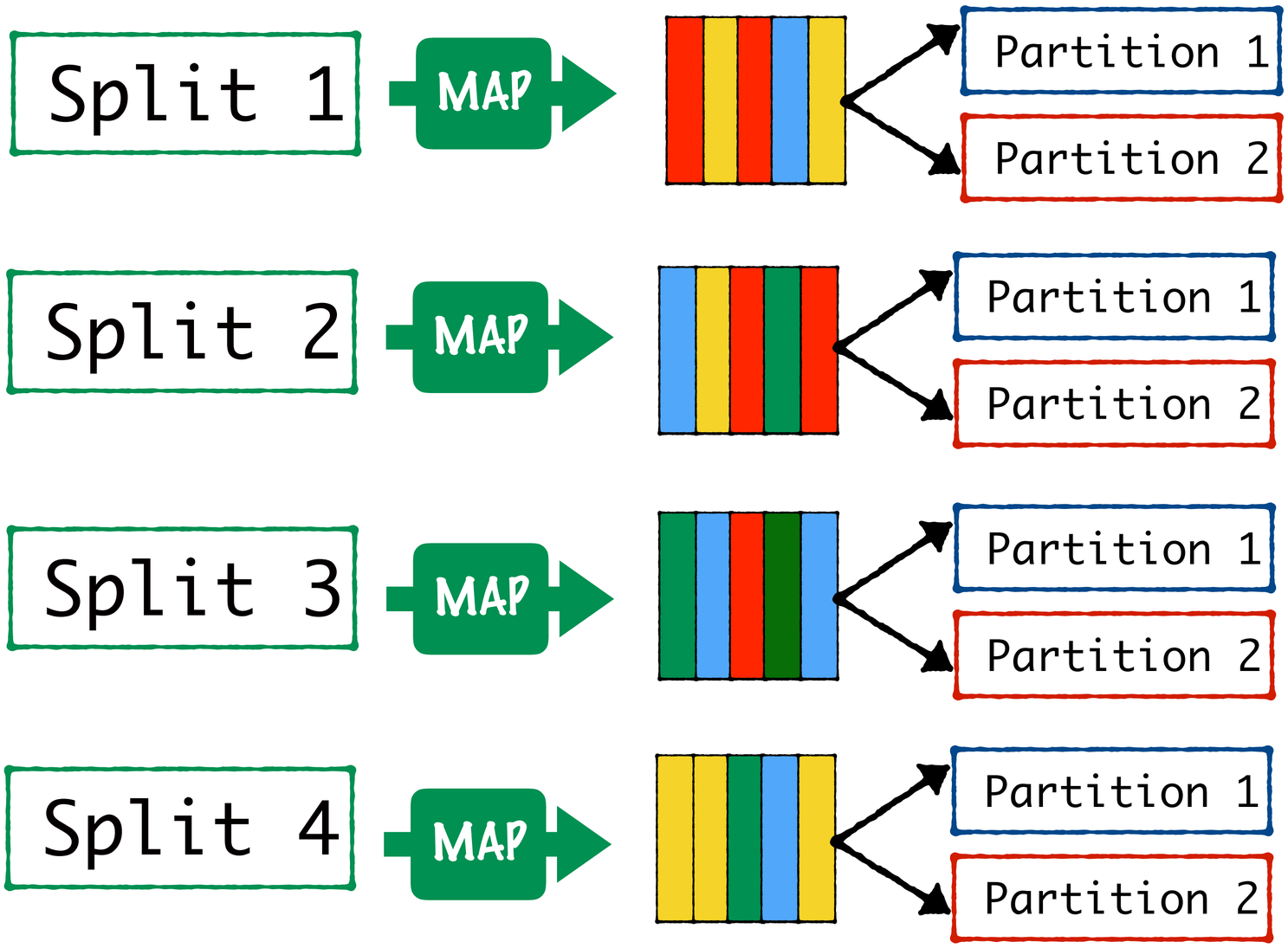
The mappers output <Key, Value> pairs

Each block here represents a <Key, Value> pair

The color represents the key

RECAP

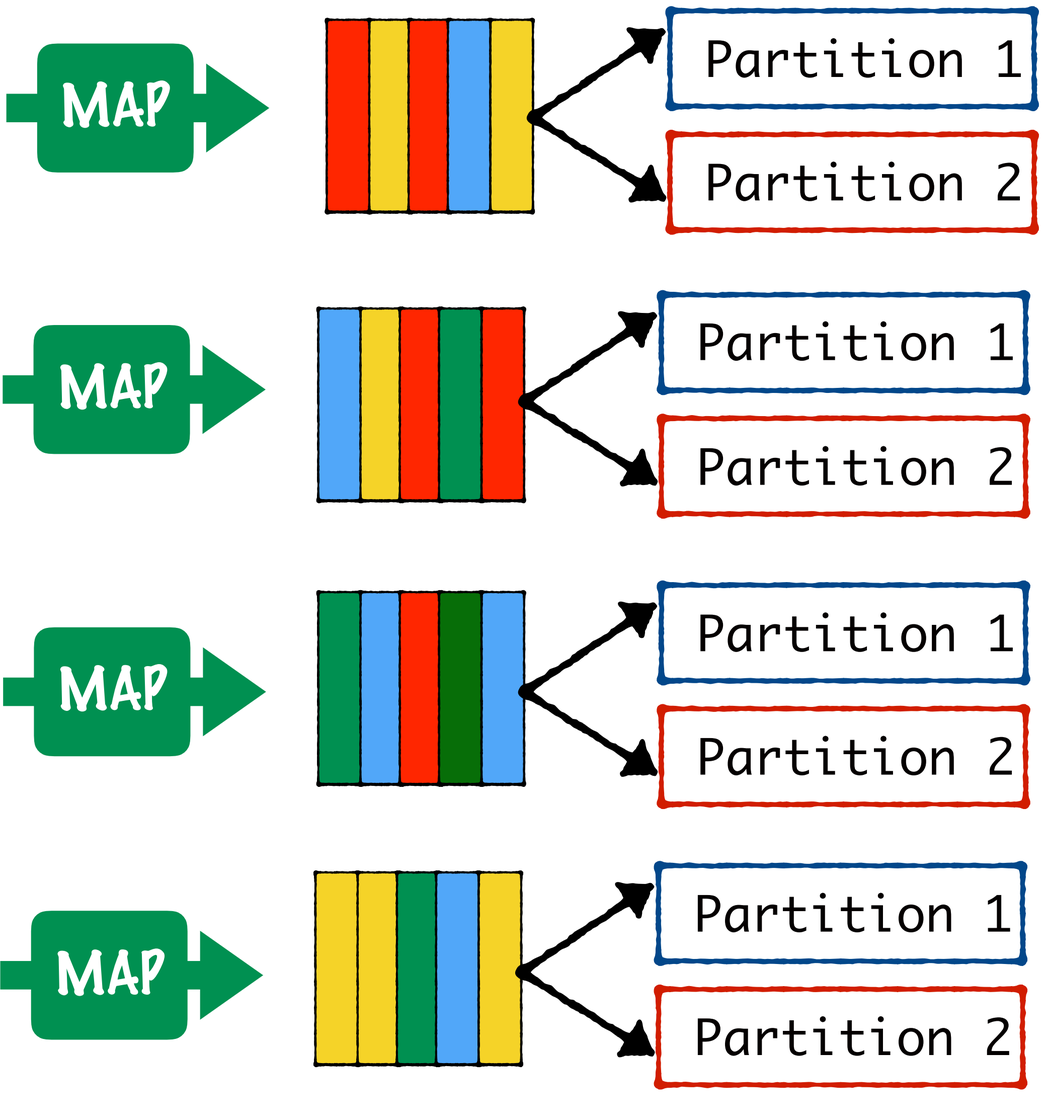
Shuffle and Sort



When there are 2 reducers, the output of each map is first partitioned into 2 partitions

RECAP

Shuffle and Sort

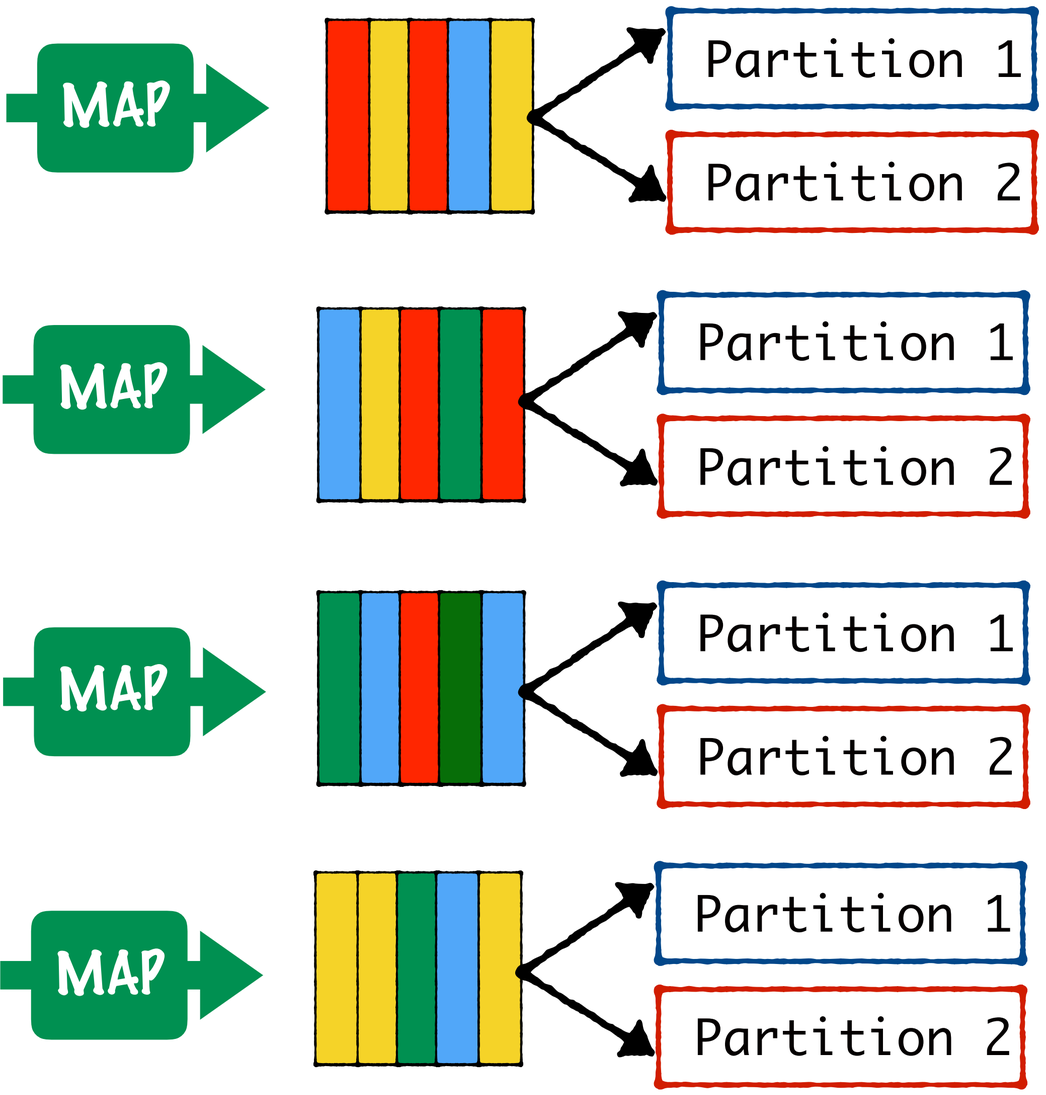


The map output is assigned to a partition based on the key

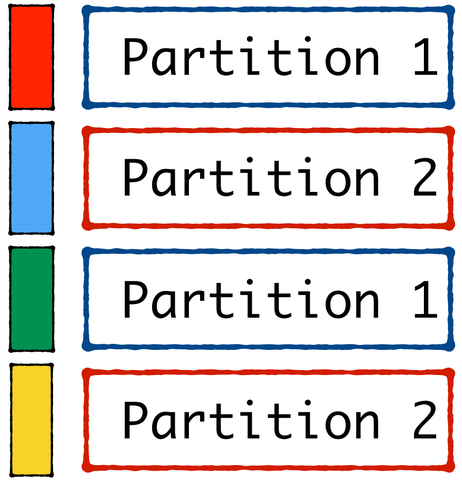
All values with the same key must be assigned to the same partition

RECAP

Shuffle and Sort

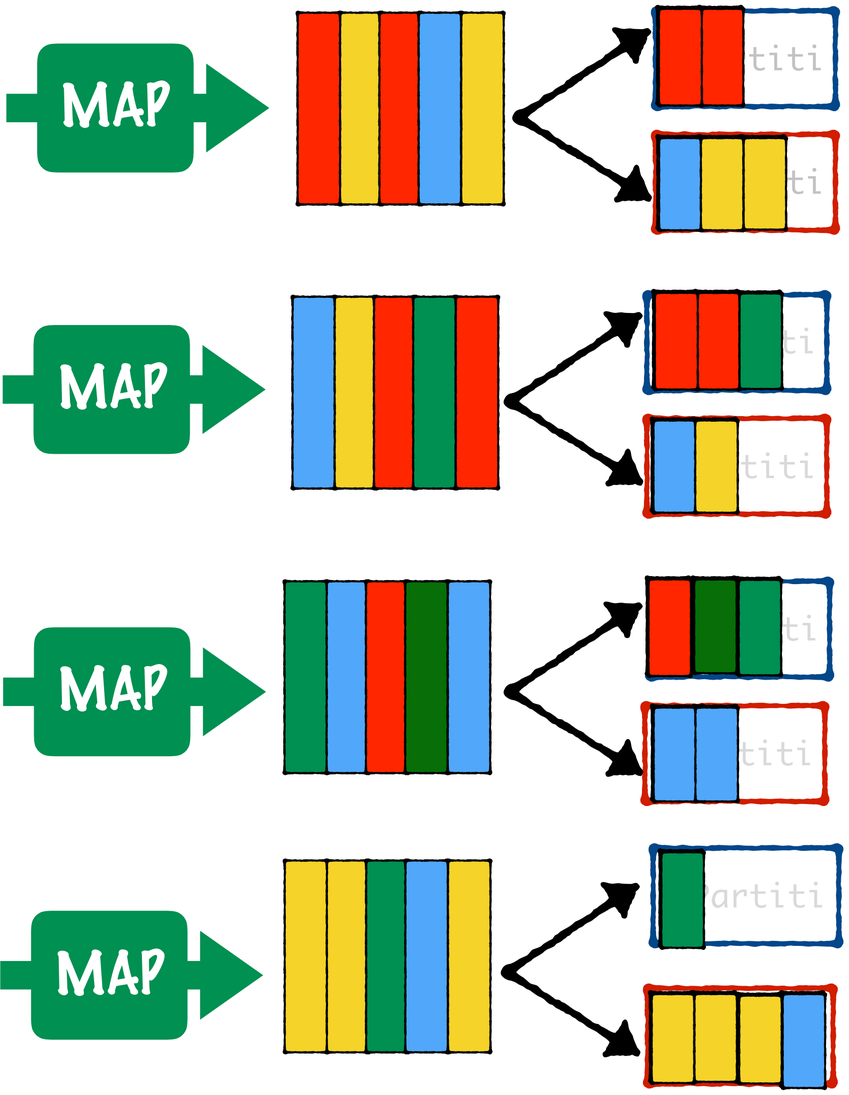


All values with the same key must be assigned to the same partition

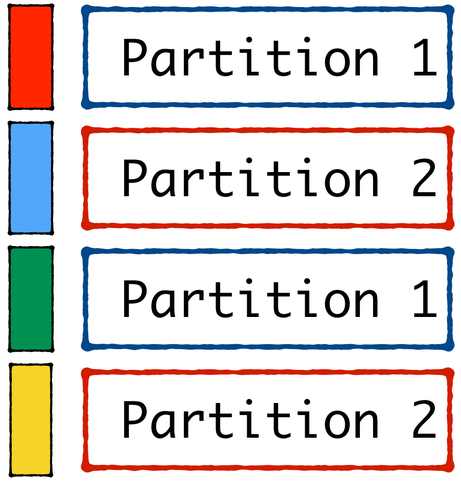


RECAP

Shuffle and Sort



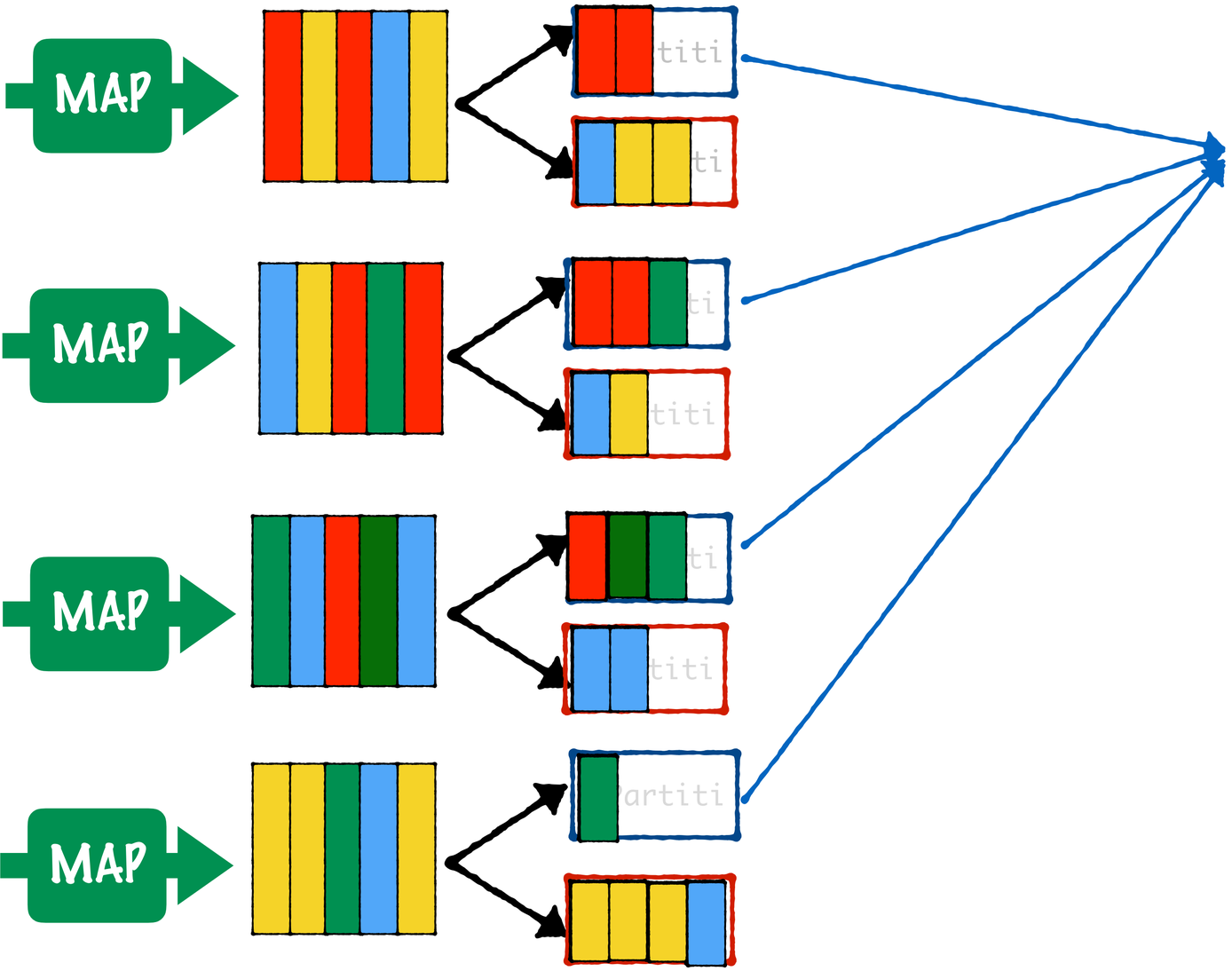
All values with the same key must be assigned to the same partition



RECAP

Shuffle and Sort

Partitioning

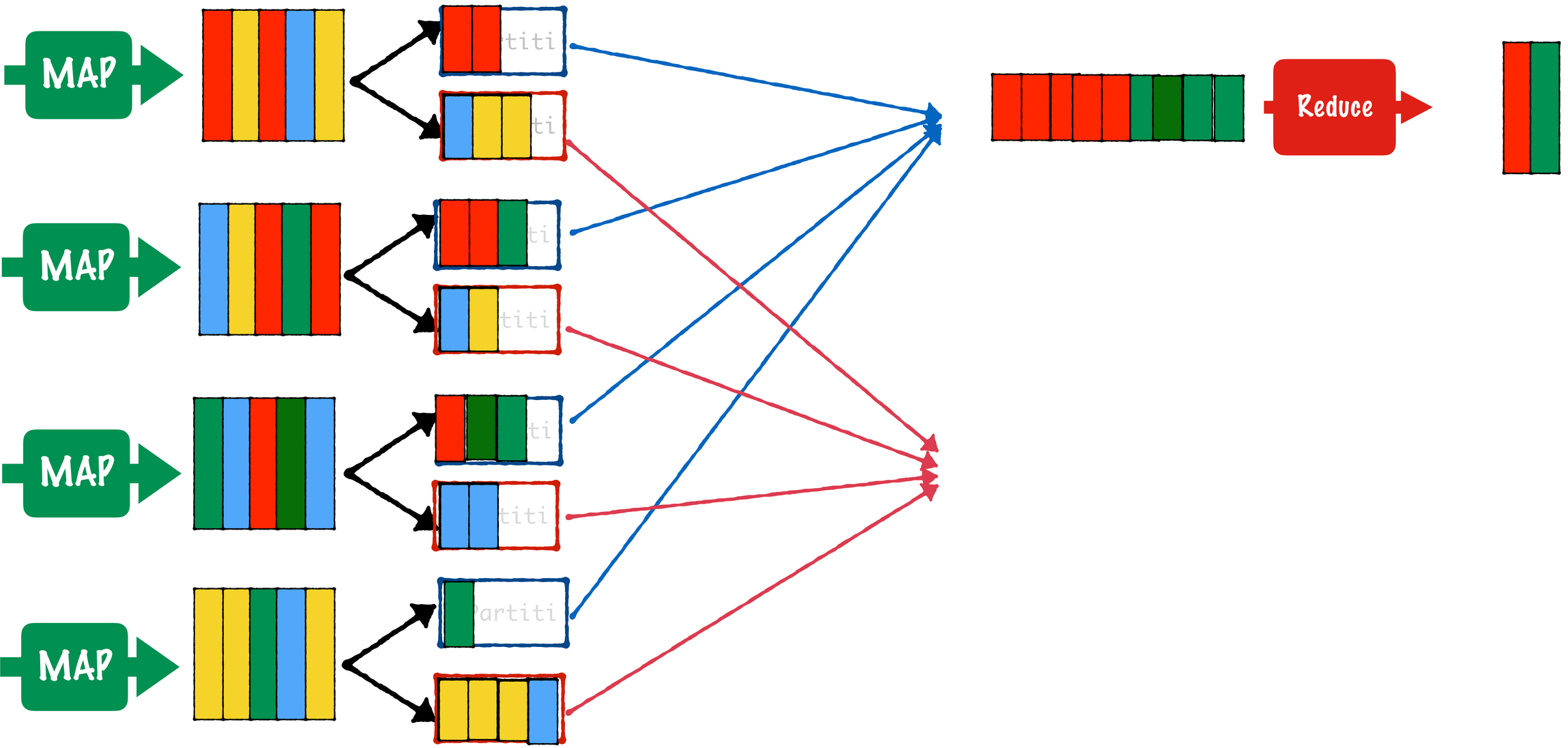


The data in each partition is sent to 1 reducer

RECAP

Shuffle and Sort

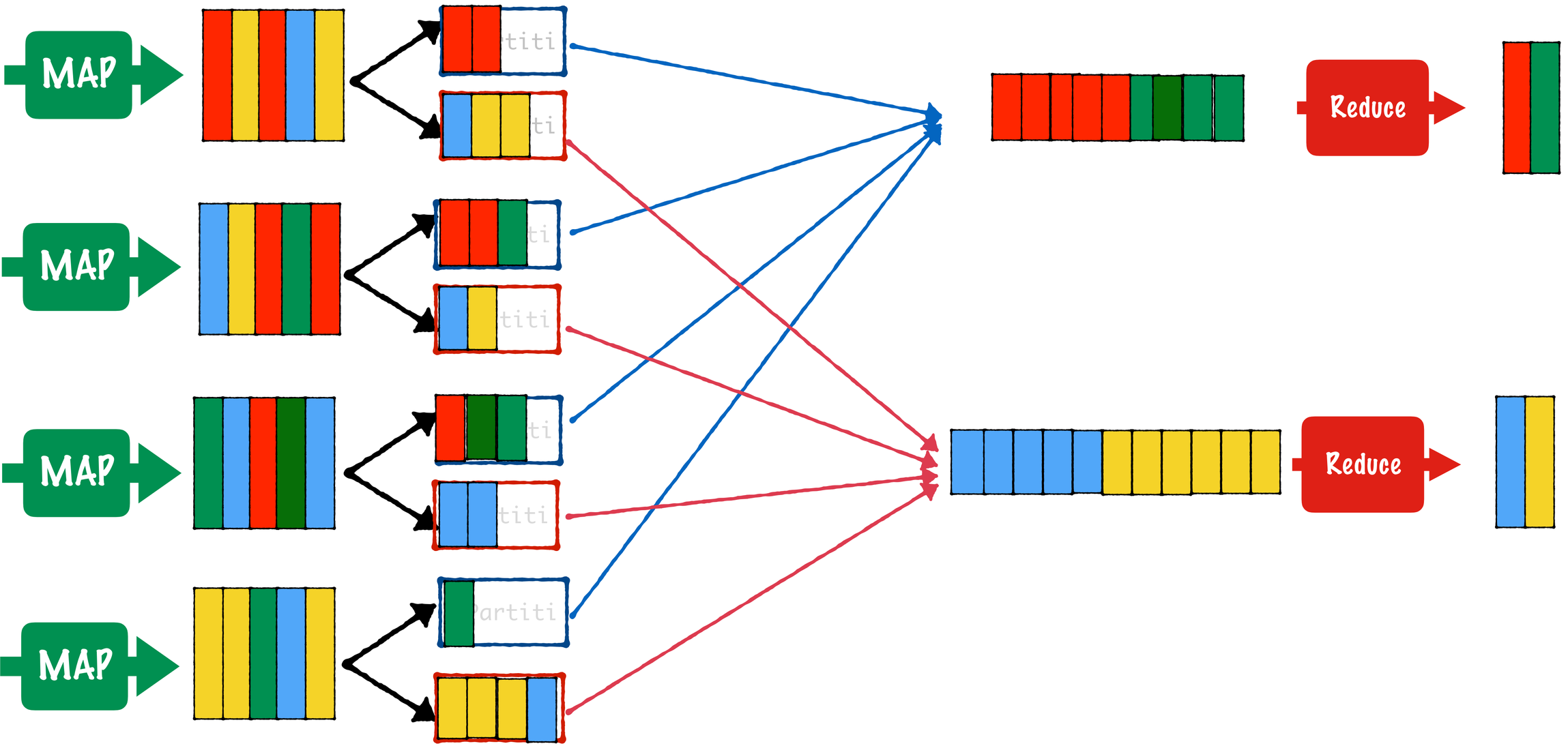
Partitioning

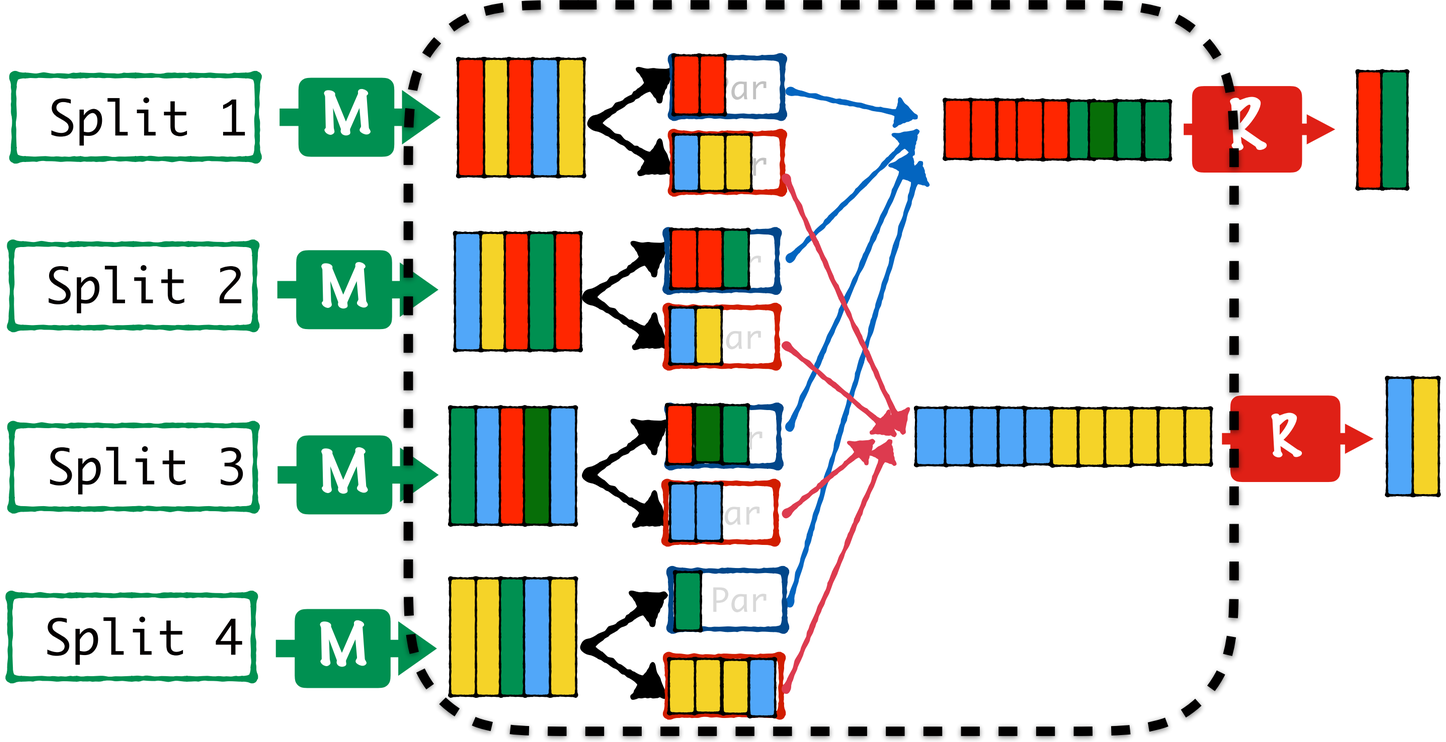


RECAP

Shuffle and Sort

Partitioning





There are three operations between the Map and Reduce

1. Partitioning
2. Sorting by Key
3. Grouping/Merging by Key

MR Job Configuration

Job object

Configurable Options

```
setInputFormatClass()
setMapOutputKeyClass()
setMapOutputValueClass()
setOutputKeyClass()
setOutputValueClass()
setMapperClass()
setCombinerClass()
setPartitionerClass()
setSortComparatorClass()
setGroupingComparatorClass(
setReducerClass()
setOutputFormatClass()
```

1. Partitioning
2. Sorting by Key
3. Grouping/Merging by Key

It's possible for the user to actually control all 3

MR Job Configuration

Job object

1. Partitioning

The default is the **HashPartitioner** class

Configurable Options

- setInputFormatClass()
- setMapOutputKeyClass()
- setMapOutputValueClass()
- setOutputKeyClass()
- setOutputValueClass()
- setMapperClass()
- setCombinerClass()
- **setPartitionerClass()**
- setSortComparatorClass()
- setGroupingComparatorClass()
- setReducerClass()
- setOutputFormatClass()

MR Job Configuration

Job object

1. Partitioning

HashPartitioner will assign a partition based on the hash value of the key

Configurable Options

```
setInputFormatClass()
setMapOutputKeyClass()
setMapOutputValueClass()
setOutputKeyClass()
setOutputValueClass()
setMapperClass()
setCombinerClass()
setPartitionerClass()
setSortComparatorClass()
setGroupingComparatorClass()
setReducerClass()
setOutputFormatClass()
```

MR Job Configuration

Job object

1. Partitioning

You can set it to any class that extends the Partitioner Class

Configurable Options

```
setInputFormatClass()
setMapOutputKeyClass()
setMapOutputValueClass()
setOutputKeyClass()
setOutputValueClass()
setMapperClass()
setCombinerClass()
setPartitionerClass()
setSortComparatorClass()
setGroupingComparatorClass()
setReducerClass()
setOutputFormatClass()
```

MR Job Configuration

Job object

Configurable Options

2. Sorting by Key

This too is not normally set by the user

```
setInputFormatClass()
setMapOutputKeyClass()
setMapOutputValueClass()
setOutputKeyClass()
setOutputValueClass()
setMapperClass()
setCombinerClass()
setPartitionerClass()
```

setSortComparatorClass()

```
setGroupingComparatorClass()
setReducerClass()
setOutputFormatClass()
```

2. Sorting by Key

`setSortComparatorClass()`

There are 2 ways to sort keys in Hadoop

1. Keys implement WritableComparable and Hadoop uses the compareTo to sort keys

2. Set up a comparator using setSortComparatorClass which sorts the keys

2. Sorting by Key

`setSortComparatorClass()`

Usually Map Output Keys are a subclass of WritableComparable

This is what we've worked with so far

2. Sorting by Key

`setSortComparatorClass()`

WritableComparators can sort the serialized byte streams of your outputs directly

2. Sorting by Key

`setSortComparatorClass()`

WritableComparators can sort
the serialized byte streams of
your outputs directly

Set the Comparator using
the setSortComparatorClass
method on the Job object

2. Sorting by Key

`setSortComparatorClass()`

WritableComparators can sort
the serialized byte streams of
your outputs directly

Because these work on raw byte streams rather than on deserialized object they work faster than using the WritableComparable

2. Sorting by Key

`setSortComparatorClass()`

WritableComparable

If you use a custom WritableComparable without a WritableComparator

the **compareTo() method** of the WritableComparable is used

MR Job Configuration

Job object

Configurable Options

2. Sorting by Key

You can override the default by setting this to a custom class that extends **WritableComparator**

```
setInputFormatClass()
setMapOutputKeyClass()
setMapOutputValueClass()
setOutputKeyClass()
setOutputValueClass()
setMapperClass()
setCombinerClass()
setPartitionerClass()
```

`setSortComparatorClass()`

```
setGroupingComparatorClass()
setReducerClass()
setOutputFormatClass()
```

MR Job Configuration

Job object

Configurable Options

3. Grouping/ Merging by Key

```
setInputFormatClass()
setMapOutputKeyClass()
setMapOutputValueClass()
setOutputKeyClass()
setOutputValueClass()
setMapperClass()
setCombinerClass()
setPartitionerClass()
setSortComparatorClass()
setGroupingComparatorClass()
setReducerClass()
setOutputFormatClass()
```

3. Grouping/ Merging by Key

`setGroupingComparatorClass()`

This is used to group together values with the same Key

3. Grouping/ Merging by Key

`setGroupingComparatorClass()`

These are then passed to the reducer as <Key, Group of Values>

3. Grouping/ Merging by Key

`setGroupingComparatorClass()`

By default, a similar strategy as that of SortComparator is used to compare the Keys

3. Grouping/ Merging by Key

`setGroupingComparatorClass()`

I.E. either the WritableComparable.compareTo() or the WritableComparator

MR Job Configuration

Job object

Configurable Options

3. Grouping/ Merging by Key

You can override this using any class that **extends WritableComparator**

```
setInputFormatClass()
setMapOutputKeyClass()
setMapOutputValueClass()
setOutputKeyClass()
setOutputValueClass()
setMapperClass()
setCombinerClass()
setPartitionerClass()
setSortComparatorClass()
setGroupingComparatorClass()
setReducerClass()
setOutputFormatClass()
```

MR Job Configuration

Job object

Configurable Options

```
setInputFormatClass()
setMapOutputKeyClass()
setMapOutputValueClass()
setOutputKeyClass()
setOutputValueClass()
setMapperClass()
setCombinerClass()
setPartitionerClass()
setSortComparatorClass()
setGroupingComparatorClass()
setReducerClass()
setOutputFormatClass()
```

In the next few classes, let's dig a little deeper into customizing our MR Jobs