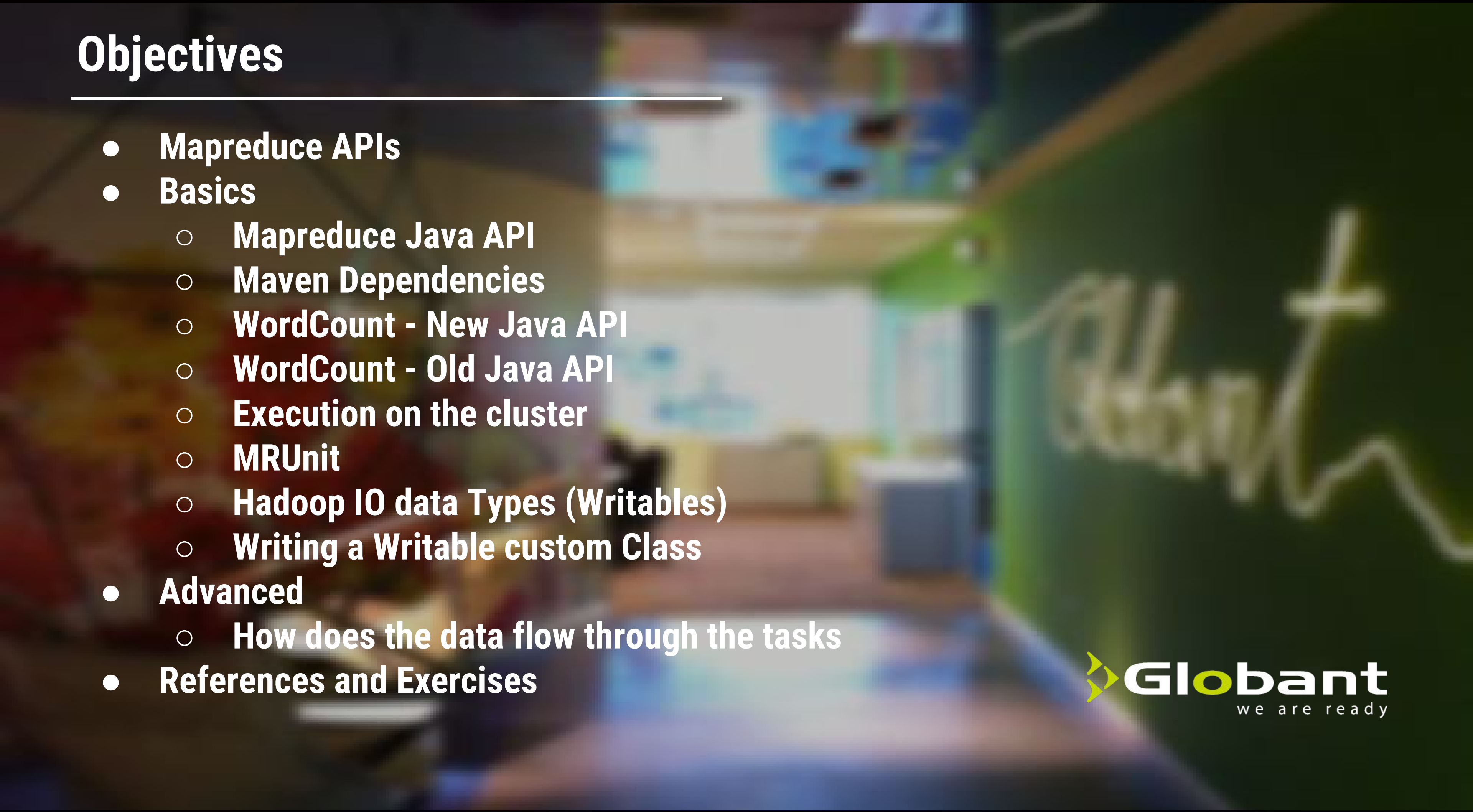
Are you ready for the future?

High Performance Solutions, Backend Development and Integration Services, Embedded Software Development, Big Data (Visualization, Architecture, Science), Business Intelligence & Analytics

Globant
we are ready

BIG DATA & HIGH PERFORMANCE

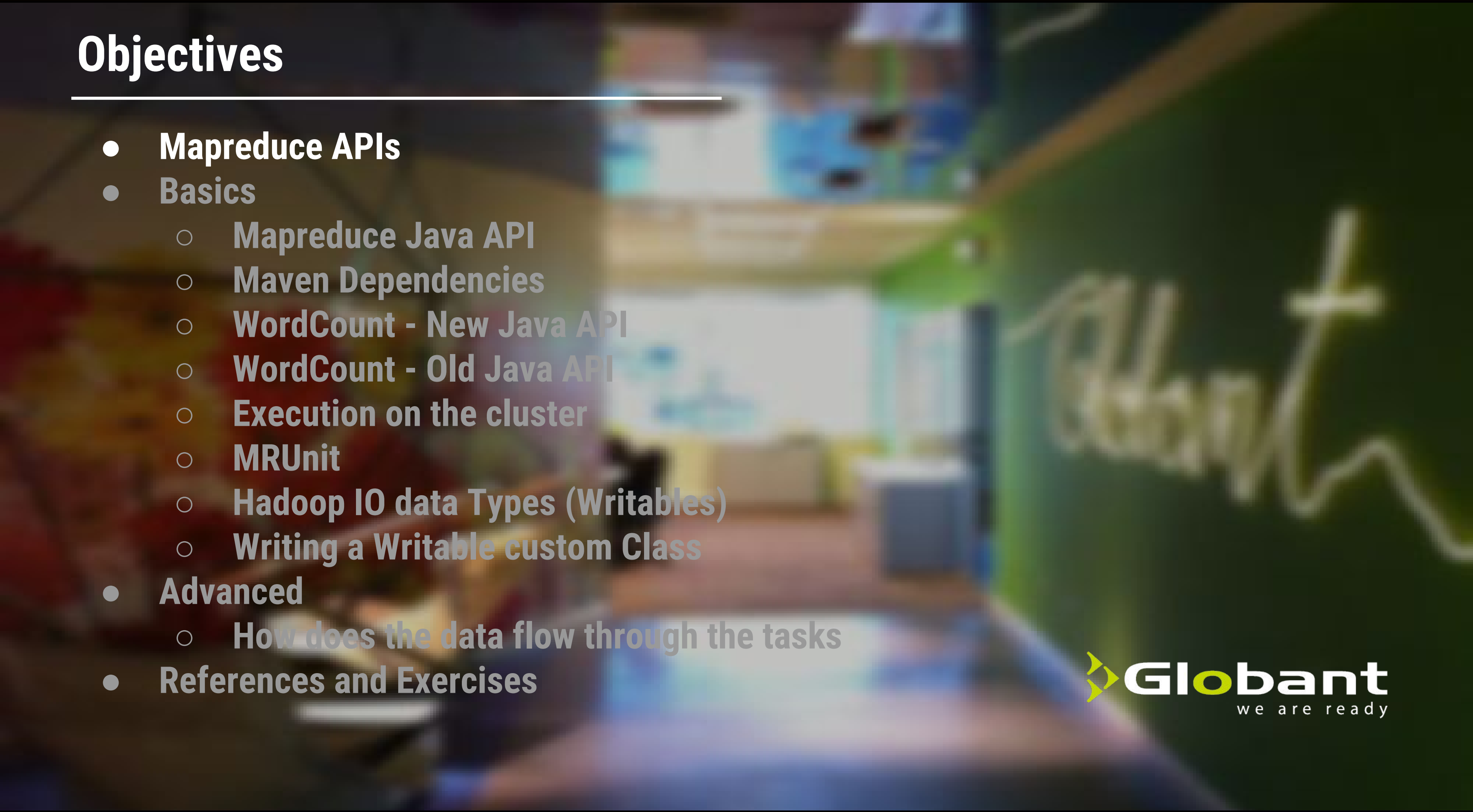
AUG 2014



Objectives

- Mapreduce APIs
- Basics
    - Mapreduce Java API
    - Maven Dependencies
    - WordCount - New Java API
    - WordCount - Old Java API
    - Execution on the cluster
    - MRUnit
    - Hadoop IO data Types (Writables)
    - Writing a Writable custom Class
- Advanced
    - How does the data flow through the tasks
- References and Exercises

Globant
we are ready



Objectives

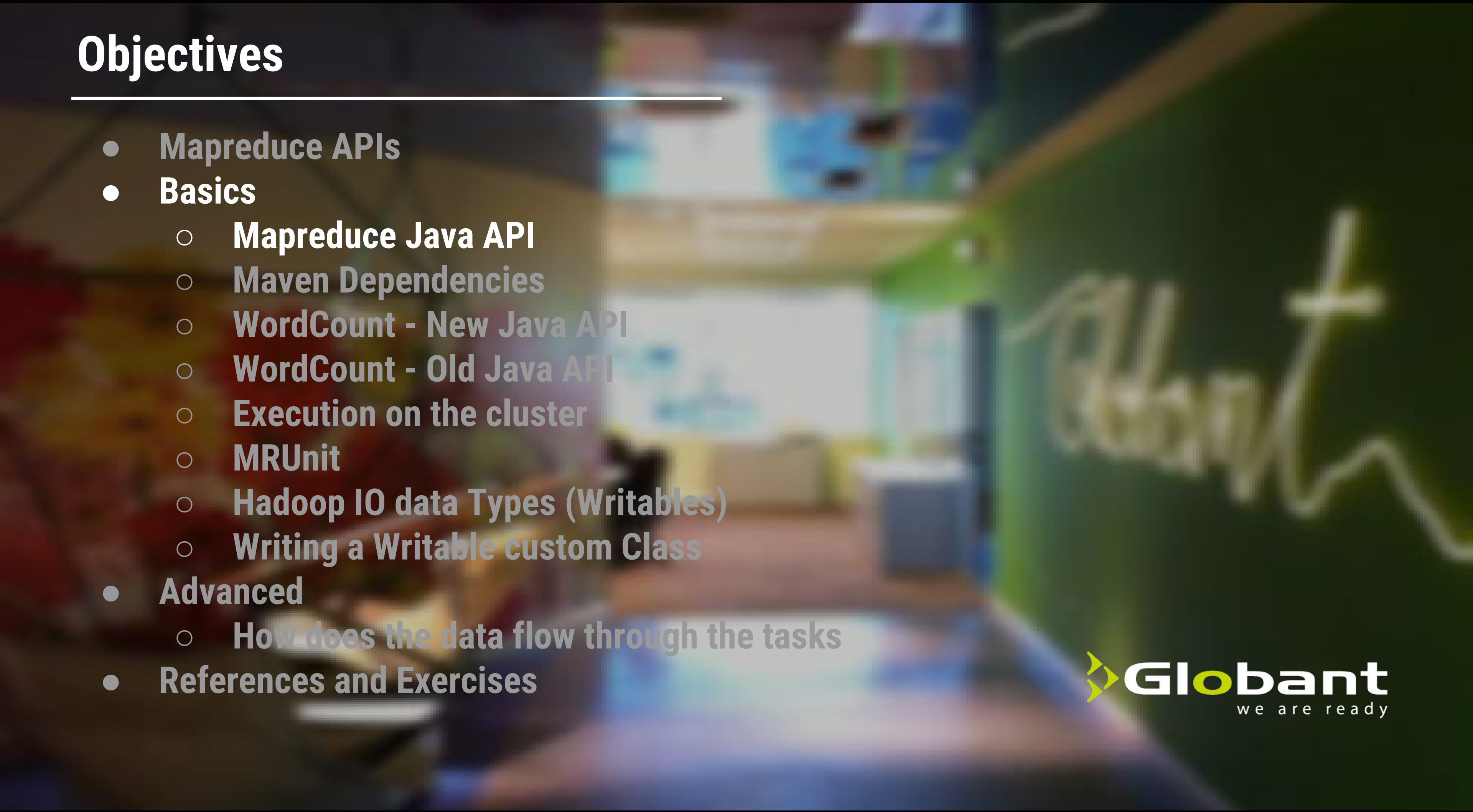
- Mapreduce APIs
- Basics
  - Mapreduce Java API
  - Maven Dependencies
  - WordCount - New Java API
  - WordCount - Old Java API
  - Execution on the cluster
  - MRUnit
  - Hadoop IO data Types (Writables)
  - Writing a Writable custom Class
- Advanced
  - How does the data flow through the tasks
- References and Exercises

Globant
we are ready

MapReduce APIs

- Streaming API (last session)
  - works practically any programming language (python, bash, perl, c++, etc.)
  - ideal for prototyping !
  - mapper script called one time per split.
  - reducer inputs are (key,value) pairs sorted by key, not (key,list-of-values)
  - reducer script called one time for all full pairs of (key, value)

- Java API (today)
  - requires to know Java, hadoop packages, etc.
  - a bit better performance and more control
  - a bit lengthier to debug.
  - mapper method invoked one time per record.
  - reducer method invoked one time per each pair of (key, list-of-values)

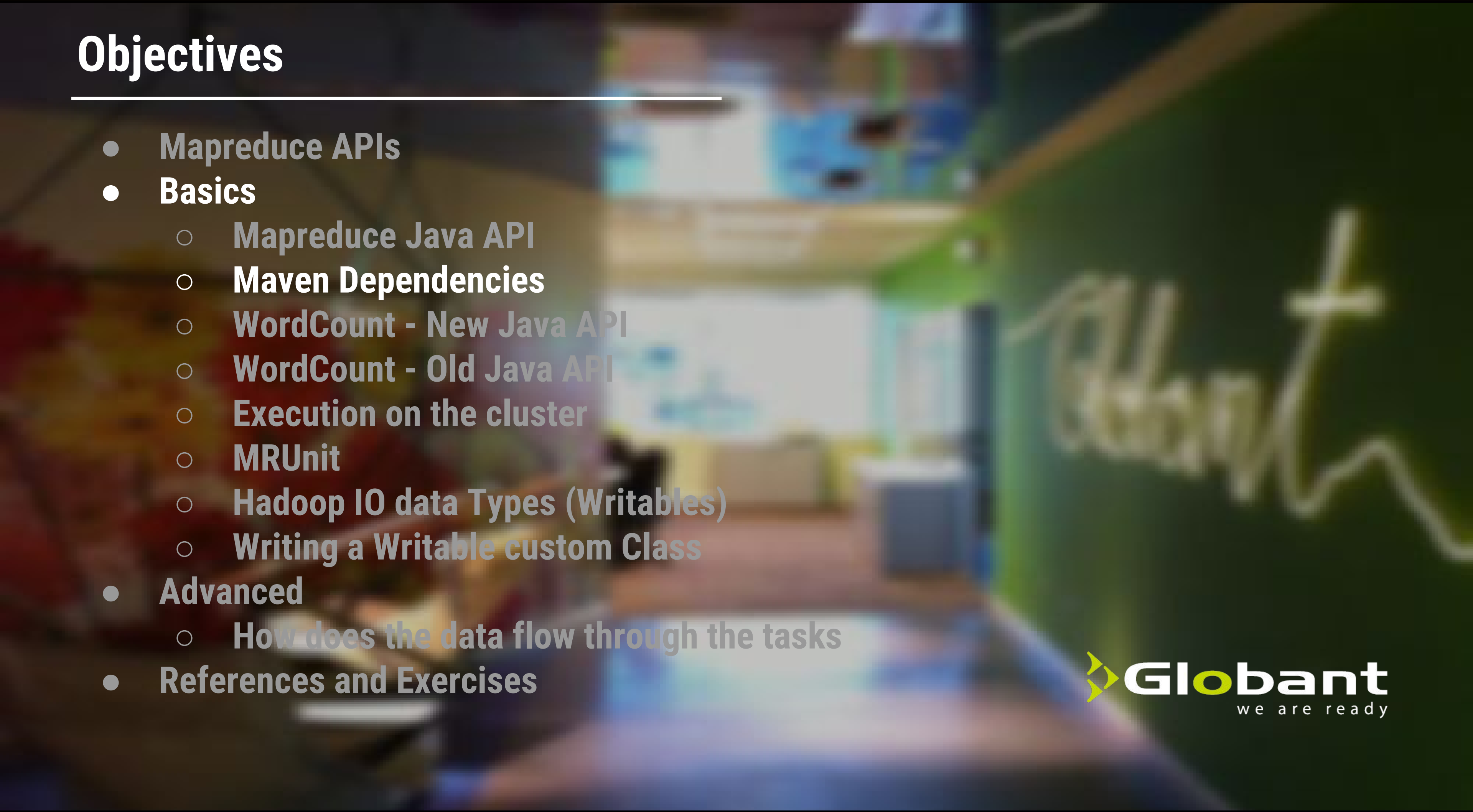


Objectives

- Mapreduce APIs
- Basics
  - Mapreduce Java API
  - Maven Dependencies
  - WordCount - New Java API
  - WordCount - Old Java API
  - Execution on the cluster
  - MRUnit
  - Hadoop IO data Types (Writables)
  - Writing a Writable custom Class
- Advanced
  - How does the data flow through the tasks
- References and Exercises

MapReduce Java API

- Two Mapreduce Java API versions: Old API (up to 0.20) and New API (after 0.21)
- Two Mapreduce Framework implementations: Mapreduce 1 (Classic) and Mapreduce 2 (YARN)
- Both APIs work on both Framework implementations (We'll focus on NEW API).
- (Key,Value) pairs fed one at a time automatically by framework to mapper/reducer methods.
- The Developer must (minimally):
  - write Mapper class with map( ) method
  - write Reducer class with reduce( ) method
  - write Java Driver class with run( ) method and static main method
- The framework serializes/deserializes built-in data types (using the Writable interface).



Objectives

- Mapreduce APIs
- Basics
  - Mapreduce Java API
  - Maven Dependencies
  - WordCount - New Java API
  - WordCount - Old Java API
  - Execution on the cluster
  - MRUnit
  - Hadoop IO data Types (Writables)
  - Writing a Writable custom Class
- Advanced
  - How does the data flow through the tasks
- References and Exercises

Maven Dependencies (add to pom.xml)

```
...
  <dependencies>
    <dependency>
      <groupId>org.apache.hadoop</groupId>
      <artifactId>hadoop-client</artifactId>
      <version>2.5.2</version>           ← (the Cluster has Apache Hadoop 2.5.2 and the VM has 2.6.0)
      <scope>provided</scope>
    </dependency>


    <dependency>
      <groupId>org.apache.mrunit</groupId>
      <artifactId>mrunit</artifactId>
      <version>1.1.0</version>
      <scope>test</scope>
      <classifier>hadoop2</classifier>
    </dependency>
  </dependencies>
...
```

Objectives

- Mapreduce APIs
- Basics
  - Mapreduce Java API
  - Maven Dependencies
  - WordCount - New Java API
  - WordCount - Old Java API
  - Execution on the cluster
  - MRUnit
  - Hadoop IO data Types (Writables)
  - Writing a Writable custom Class
- Advanced
  - How does the data flow through the tasks
- References and Exercises

WordCount pseudocode

```
mapper (filename, file-contents):
  for each word in file-contents:
    emit (word, 1)




reducer (word, values):
  sum = 0
  for each value in values:
    sum = sum + value
  emit (word, sum)
```

WordCountMapper.java (New API)

```java
package com.hadoop.example.wordcount;

import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Mapper;
import java.io.IOException;

public class WordCountMapper extends Mapper<LongWritable, Text, Text, LongWritable> {
	private static final LongWritable ONE = new LongWritable( 1 );

	@Override
	public void map(LongWritable key, Text value, Context context) throws IOException, InterruptedException {
		String line = value.toString().toLowerCase();
		String[] words = line.split( " " );
		for (String word : words ) {
			context.write( new Text(word), ONE );
		}
	}
}
```

WordCountReducer.java (New API)

```java
package com.hadoop.example.wordcount;

import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Reducer;
import java.io.IOException;

public class WordCountReducer extends Reducer<Text, LongWritable, Text, LongWritable> {

	private LongWritable result = new LongWritable();

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

WordCountDriver.java - part 1 (New API)

```java
package com.hadoop.example.wordcount;

import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.conf.Configured;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.hadoop.util.Tool;
import org.apache.hadoop.util.ToolRunner;
```

WordCountDriver.java - part 2 (New API)

```java
public class WordCountDriver extends Configured implements Tool {
    public static void main(String[] args) throws Exception {
        int res = ToolRunner.run( new Configuration(), new WordCountDriver(), args );
        System.exit( res );
    }

    @Override
    public int run(String[] args) throws Exception {
        if (args.length != 2) {
            System.out.println("Usage: hadoop jar hadoop-wordcount-example-1.0-SNAPSHOT-job.jar"
                                        + " [generic options] <in> <out>");
            return 1;
        }
        Job job = new Job( getConf(), "WordCount" );
        job.setJarByClass( getClass() );
        job.setMapperClass( WordCountMapper.class );
        job.setCombinerClass( WordCountReducer.class );
        job.setReducerClass( WordCountReducer.class );
        job.setOutputKeyClass( Text.class );
        job.setOutputValueClass( LongWritable.class );
        FileInputFormat.addInputPath( job, new Path(args[0]) );
        FileOutputFormat.setOutputPath( job, new Path(args[1]) );
        boolean success = job.waitForCompletion( true );
        return success ? 0 : 1;
    }
}
```

Objectives

- Mapreduce APIs
- Basics
  - Mapreduce Java API
  - Maven Dependencies
  - WordCount - New Java API
  - WordCount - Old Java API
  - Execution on the cluster
  - MRUnit
  - Hadoop IO data Types (Writables)
  - Writing a Writable custom Class
- Advanced
  - How does the data flow through the tasks
- References and Exercises

WordCountMapper.java (Old API)

```
package com.hadoop.example.wordcount;

import java.io.IOException;


import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapred.MapReduceBase;                  // MapreduceBase and Mapper merged in new API as class Mapper
import org.apache.hadoop.mapred.Mapper;                         // new API uses package org.apache.hadoop.mapreduce
import org.apache.hadoop.mapred.OutputCollector;                // OutputCollector and Reporter merged in new API as Context
import org.apache.hadoop.mapred.Reporter;

public class WordCountMapper extends MapReduceBase implements Mapper<LongWritable,Text,Text,LongWritable>{

        private static final LongWritable ONE = new LongWritable( 1 );

        public void map(LongWritable key, Text value, OutputCollector<Text,LongWritable> output, Reporter reporter) throws
IOException{
                String line = value.toString();
                String[] words = line.split( " " );
                for (String word: words) {
                        output.collect(new Text(word), ONE );
                }
        }

}
```

WordCountReducer.java (Old API)

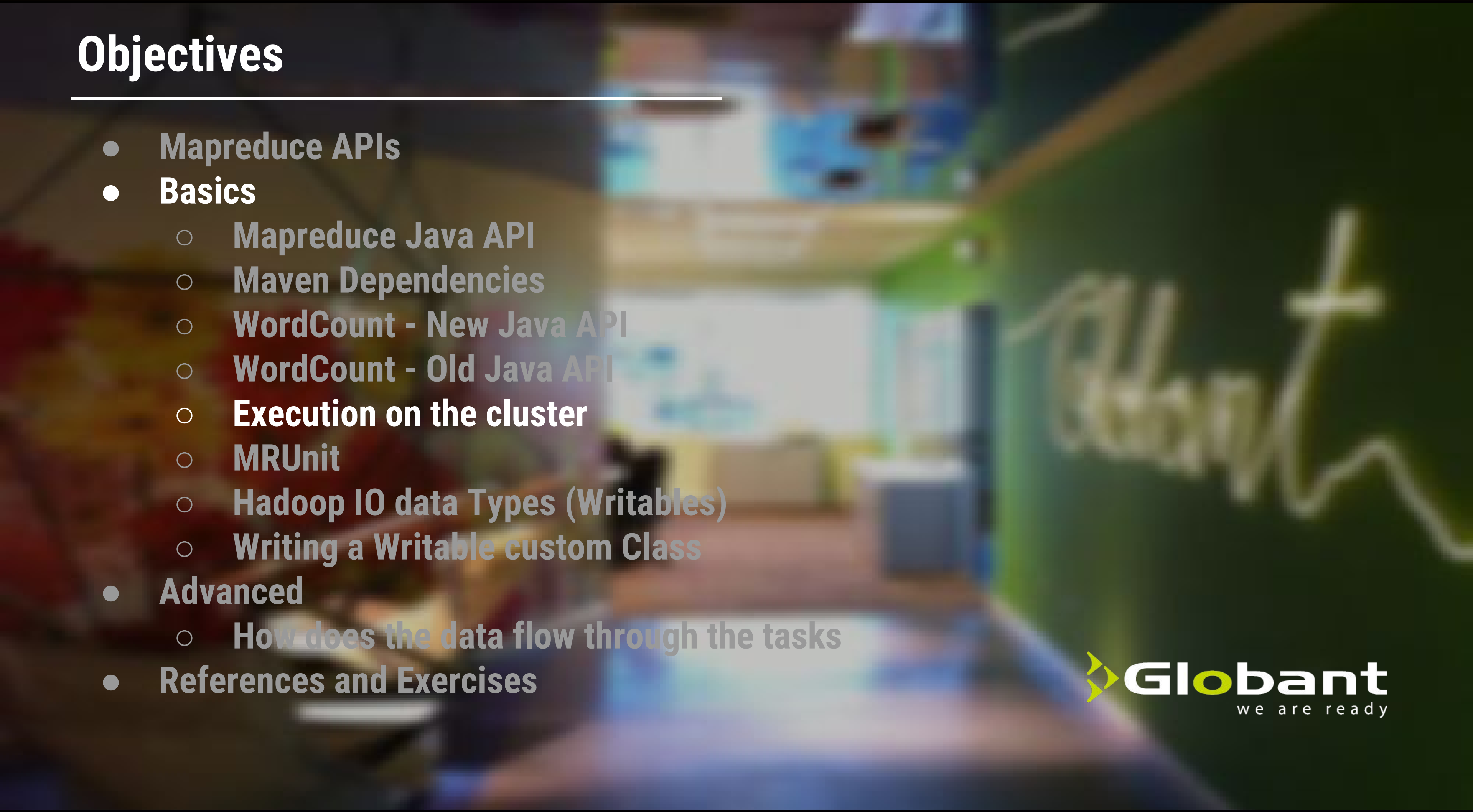
```
package com.hadoop.example.wordcount;

import java.io.IOException;
import java.util.Iterator;

import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapred.MapReduceBase;                  // MapreduceBase and Reducer merged in new API as class Reducer
import org.apache.hadoop.mapred.Reducer;                        // new API uses package org.apache.hadoop.mapreduce
import org.apache.hadoop.mapred.OutputCollector;                // OutputCollector and Reporter merged in new API as Context
import org.apache.hadoop.mapred.Reporter;

public class WordCountReducer extends MapReduceBase implements Reducer<Text, LongWritable, Text, LongWritable> {

	public void reduce(Text key, Iterator values, OutputCollector<Text, LongWritable> output, Reporter reporter) throws
IOException {
		long sum = 0;
		while( values.hasNext() ) {
			sum += values.next().get();
		}
		output.collect(key, new LongWritable(sum));
	}
}
```

WordCountDriver.java - part 1 (Old API)

```
package com.hadoop.example.wordcount;
import java.io.IOException;

import org.apache.hadoop.conf.Configured;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapred.FileInputFormat;                 // new API uses package org.apache.hadoop.mapreduce
import org.apache.hadoop.mapred.FileOutputFormat;
import org.apache.hadoop.mapred.JobClient;                       // JobClient and JobConf merged in new API as class Job
import org.apache.hadoop.mapred.JobConf;
import org.apache.hadoop.util.Tool;
import org.apache.hadoop.util.ToolRunner;
```

WordCountDriver.java - part 2 (Old API)

```java
public class WordCountDriver extends Configured implements Tool {

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


	public int run(String[] args) throws IOException{
		Path inputPath = new Path( args[0] );
		Path outputPath = new Path( args[1] );
		JobConf conf = new JobConf( WordCount.class );
		conf.setJobName( "WordCount" );
		FileInputFormat.setInputPaths( conf, inputPath );
		FileOutputFormat.setOutputPath( conf, outputPath );
		conf.setMapperClass( WordCountMapper.class );
		conf.setReducerClass( WordCountReducer.class );
		conf.setMapOutputKeyClass( Text.class );
		conf.setMapOutputValueClass( LongWritable.class );
		conf.setOutputKeyClass( Text.class );
		conf.setOutputValueClass( LongWritable.class );
		JobClient.runJob( conf );
		return 0;
	}
}
```



Objectives

- Mapreduce APIs
- Basics
  - Mapreduce Java API
  - Maven Dependencies
  - WordCount - New Java API
  - WordCount - Old Java API
  - Execution on the cluster
  - MRUnit
  - Hadoop IO data Types (Writables)
  - Writing a Writable custom Class
- Advanced
  - How does the data flow through the tasks
- References and Exercises

Globant
we are ready

WordCount Execution on cluster

Compile and build the jar file using maven

```
$ mvn clean package
```

copy the jar file to the cluster

```
$ scp -i ./Training.pem \
target/hadoop-wordcount-example-1.0-SNAPSHOT-job.jar \
hadoop@54.172.223.97:~/mapreduce/juan.gaviria
```

Connect to the cluster

```
$ ssh -i ./Training.pem hadoop@54.172.223.97
```

Execute the mapreduce

```
$ hadoop \
jar ~/mapreduce/juan.gaviria/hadoop-wordcount-example-1.0-SNAPSHOT-job.jar \
com.hadoop.example.wordcount.WordCountDriver \
-D mapred.map.tasks=5 \                                  ←optional
-D mapred.reduce.tasks=2 \                               ←optional
/user/hadoop/mapreduce/data/books \                      ←hdfs input dir
/user/hadoop/mapreduce/juan.gaviria/<hdfs-output-dir>    ←hdfs output dir
```

Objectives

- Mapreduce APIs
- Basics
  - Mapreduce Java API
  - Maven Dependencies
  - WordCount - New Java API
  - WordCount - Old Java API
  - Execution on the cluster
  - MRUnit
  - Hadoop IO data Types (Writables)
  - Writing a Writable custom Class
- Advanced
  - How does the data flow through the tasks
- References and Exercises

WordCountMapReduceTest.java - part 1 (New API)

```java
package com.hadoop.example.wordcount;

import java.io.IOException;
import java.util.ArrayList;
import java.util.List;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mrunit.mapreduce.MapDriver;
import org.apache.hadoop.mrunit.mapreduce.ReduceDriver;
import org.junit.Before;
import org.junit.Test;

public class WordCountMapReduceTest {
	MapDriver<LongWritable, Text, Text, LongWritable> mapDriver;
	ReduceDriver<Text, LongWritable, Text, LongWritable> reduceDriver;

	private static final LongWritable ONE = new LongWritable( 1 );

	@Before
	public void setUp() {
		WordCountMapper mapper = new WordCountMapper();
		WordCountReducer reducer = new WordCountReducer();
		mapDriver = MapDriver.newMapDriver(mapper);
		reduceDriver = ReduceDriver.newReduceDriver(reducer);
	}
```

WordCountMapReduceTest.java - part 2 (New API)

```java
@Test
public void testMapper() throws IOException {
      LongWritable inKey =  new LongWritable( 0 );
      Text inValue = new Text( "bar foo bar" );
      mapDriver.withInput( inKey, inValue );

      Text outKey1= new Text( "bar" );
      LongWritable outValue1 = ONE;
      Text outKey2= new Text( "foo" );
      LongWritable outValue2 = ONE;
      Text outKey3= new Text( "bar" );
      LongWritable outValue3 = ONE;
      mapDriver.withOutput( outKey1, outValue1 );
      mapDriver.withOutput( outKey2, outValue2 );
      mapDriver.withOutput( outKey3, outValue3 );
      mapDriver.runTest( true );
}
```

WordCountMapReduceTest.java - part 3 (New API)

```
      @Test
      public void testReducer() throws IOException {
            Text inKey = new Text( "bar" );
            List<LongWritable> inListValues = new ArrayList<LongWritable>();
            inListValues.add( ONE );
            inListValues.add( ONE );
            reduceDriver.withInput( inKey, inListValues );

            Text outKey = new Text( inKey.toString() );
            LongWritable outValue = new LongWritable( 2 );
            reduceDriver.withOutput( outKey, outValue );
            reduceDriver.runTest();
      }

}
```

Objectives

- Mapreduce APIs
- Basics
  - Mapreduce Java API
  - Maven Dependencies
  - WordCount - New Java API
  - WordCount - Old Java API
  - Execution on the cluster
  - MRUnit
  - Hadoop IO data Types (Writables)
  - Writing a Writable custom Class
- Advanced
  - How does the data flow through the tasks
- References and Exercises

Globant
we are ready

Hadoop IO Data Types (1)

- "Writables" are built-in datatypes in `org.apache.hadoop.io` optimized for serialization.
- Any serialization frameworks can be used, e.g., Avro (http://avro.apache.org/) is very popular.
- The framework serializes/deserializes types using the `Writable` interface:

```
package org.apache.hadoop.io;
import java.io.DataOutput;
import java.io.DataInput;
import java.io.IOException;

public interface Writable {
        void write(DataOutput out) throws IOException;          ← Writes to binary stream
        void readFields(DataInput in) throws IOException;       ← Reads from binary stream
}
```

Hadoop IO Data Types (2)

> The framework **sorts keys** using the good old `Comparable` interface:

```
package java.lang;
public interface Comparable<T> {
        int CompareTo(T in);         ← Reads from binary stream
}
```

> Also needs to override `equals()`, `hashCode()` and `toString()` from `java.lang.Object`

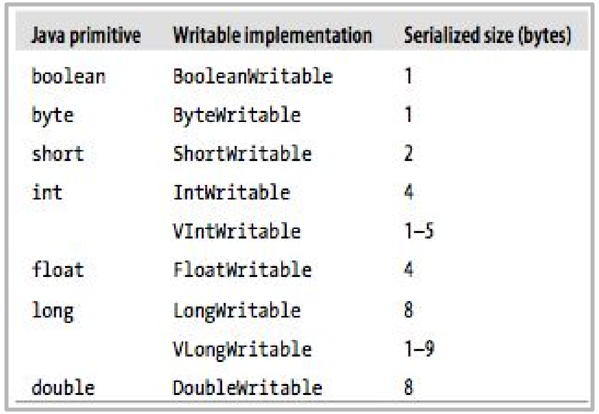
> A **default constructor** is needed too.

> The framework defines the `WritableComparable` subinterface:

```
package org.apache.hadoop.io;
public interface WritableComparable<T> extends Writable, Comparable<T> {
}
```

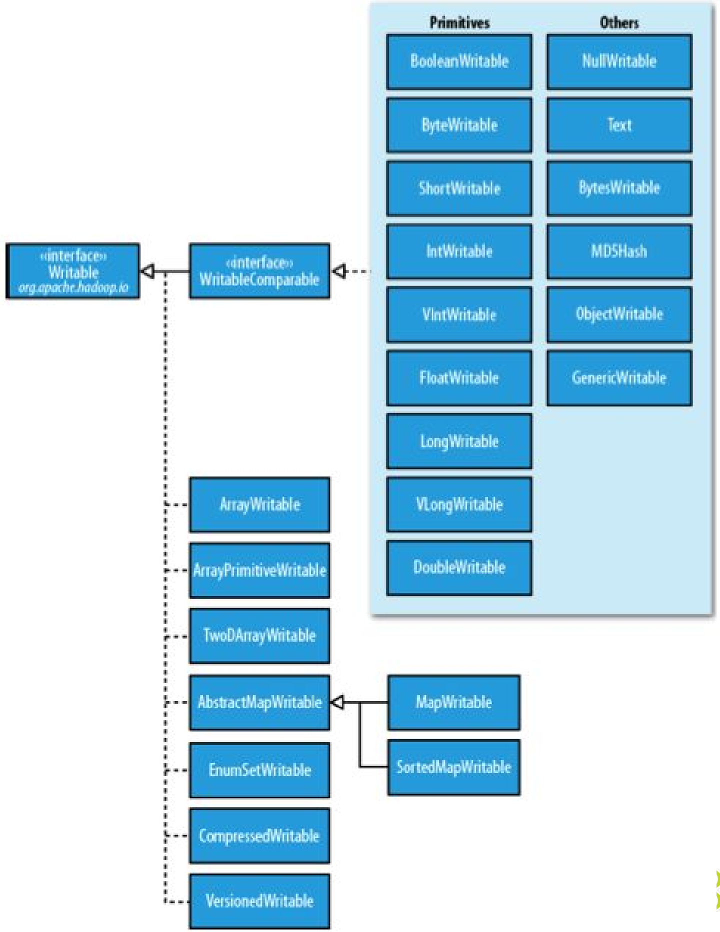
Hadoop IO Data Types (3)

> Hierarchy of `Writable` subclasses*:



Java primitive	Writable implementation	Serialized size (bytes)
boolean	BooleanWritable	1
byte	ByteWritable	1
short	ShortWritable	2
int	IntWritable	4
	VIntWritable	1–5
float	FloatWritable	4
long	LongWritable	8
	VLongWritable	1–9
double	DoubleWritable	8

> use `get()` and `set()` methods to access the primitive values.



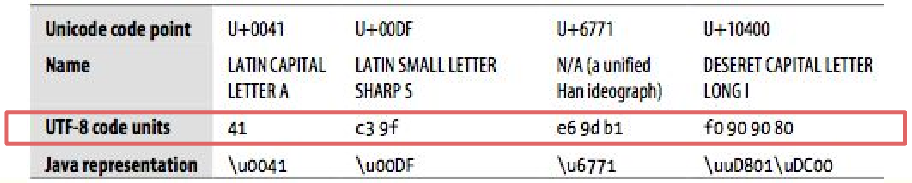
*Source: Hadoop: The Definitive Guide

Hadoop IO Data Types (4)

> Text is the implementation for java.lang.String class. However some differences:

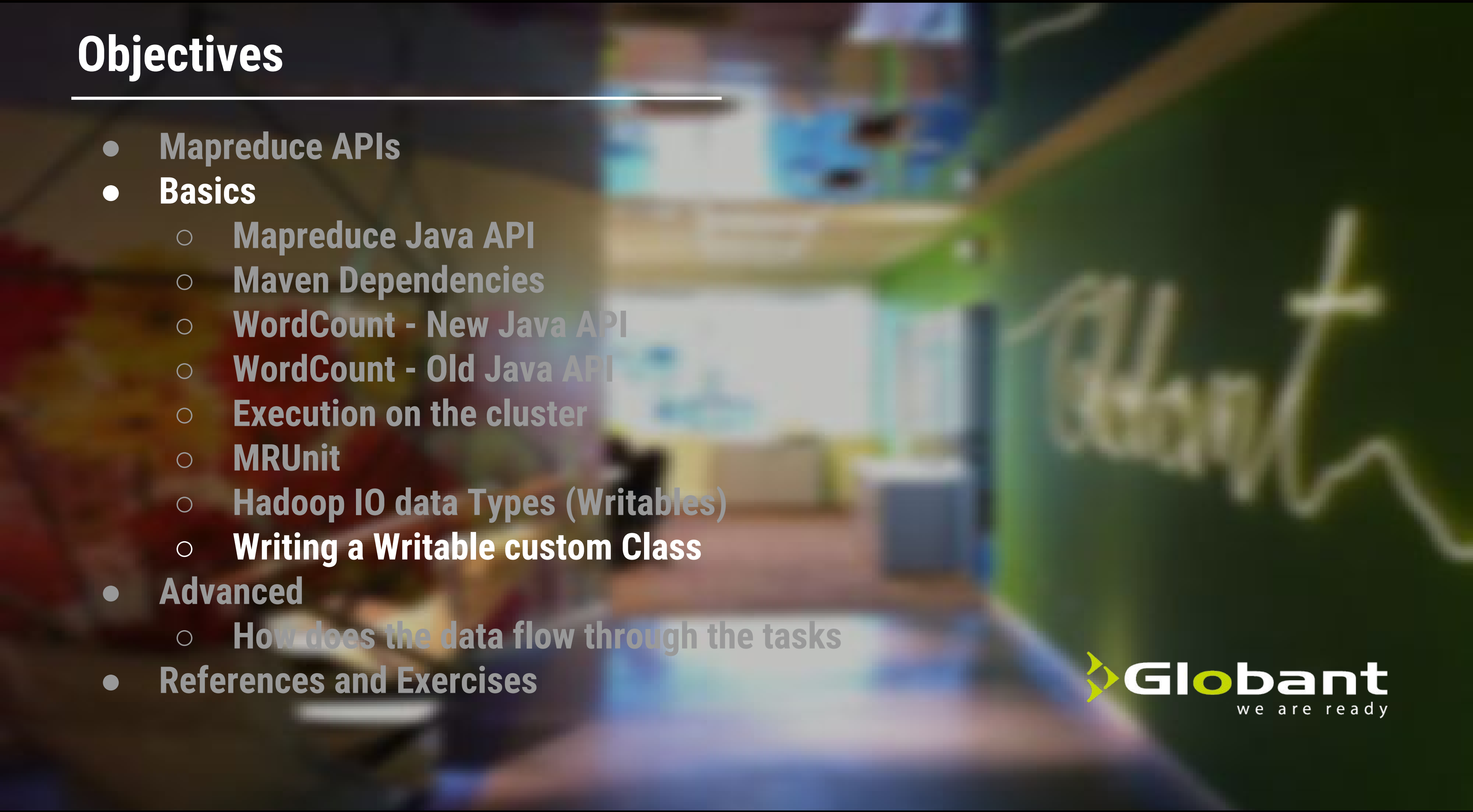
- String uses Unicode chars, Text uses UTF-8. For ASCII differences are unnoticed
- String charAt() returns char value, Text charAt() returns int value
- String indexOf() returns char positions, Text find() returns byte positions
- String length() returns char positions, Text length() returns byte positions
- For iterations, easier to use String representation than Text
- Example*:

```
String s = "\u0041\u00DF\u6771\uD801\uDC00";          // s.lenght():5  , s.indexOf("\u6771"):2 , s.charAt(3): "\uD801"
Text t = new Text("\u0041\u00DF\u6771\uD801\uDC00");  // t.lenght():10 , t.find("\u6771")   :3 , t.charAt(3): 0x6771
```



Unicode code point	U+0041	U+00DF	U+6771	U+10400
Name	LATIN CAPITAL LETTER A	LATIN SMALL LETTER SHARP S	N/A (a unified Han ideograph)	DESERET CAPITAL LETTER LONG I
UTF-8 code units	41	c3 9f	e6 9d b1	f0 90 90 80
Java representation	\u0041	\u00DF	\u6771	\uuD801\uDC00

*Source: Hadoop: The Definitive Guide



Objectives

- Mapreduce APIs
- **Basics**
  - Mapreduce Java API
  - Maven Dependencies
  - WordCount - New Java API
  - WordCount - Old Java API
  - Execution on the cluster
  - MRUnit
  - Hadoop IO data Types (Writables)
  - **Writing a Writable custom Class**
- Advanced
  - How does the data flow through the tasks
- References and Exercises

Globant
we are ready

Hadoop IO Data Types (5) - Custom Class Example*

```java
import java.io.*;
import org.apache.hadoop.io.*;

public class TextPair implements WritableComparable<TextPair> {
        private Text first; private Text second;

        public TextPair() {
                set(new Text(), new Text());
        }

        public TextPair(String first, String second) {
                set(new Text(first), new Text(second));
        }

        public TextPair(Text first, Text second) {
                set(first, second);
        }

        public void set(Text first, Text second) {
                this.first = first;
                this.second = second;
        }

        public Text getFirst() {
                return first;
        }

        public Text getSecond() {
                return second;
        }
```

*Source: Hadoop: The Definitive Guide

Hadoop IO Data Types (6) - Custom Class Example*

```
@Override
public void write(DataOutput out) throws IOException {
        first.write(out);
        second.write(out);
}

@Override
public void readFields(DataInput in) throws IOException {
        first.readFields(in);
        second.readFields(in);
}

@Override
public int hashCode() {
        return first.hashCode() * 163 + second.hashCode();
}

@Override
public boolean equals(Object o) {
        if (o instanceof TextPair) {
                TextPair tp = (TextPair) o;
                return first.equals(tp.first) && second.equals(tp.second);
        }
        return false;
}

@Override
public String toString() {
        return first + "\t" + second;
}
```

*Source: Hadoop: The Definitive Guide

Hadoop IO Data Types (7) - Custom Class Example*

```
        @Override
        public int compareTo(TextPair tp) {
                int cmp = first.compareTo(tp.first);
                if (cmp != 0) {
                        return cmp;
                }
                return second.compareTo(tp.second);
        }

}
```

*Source: Hadoop: The Definitive Guide

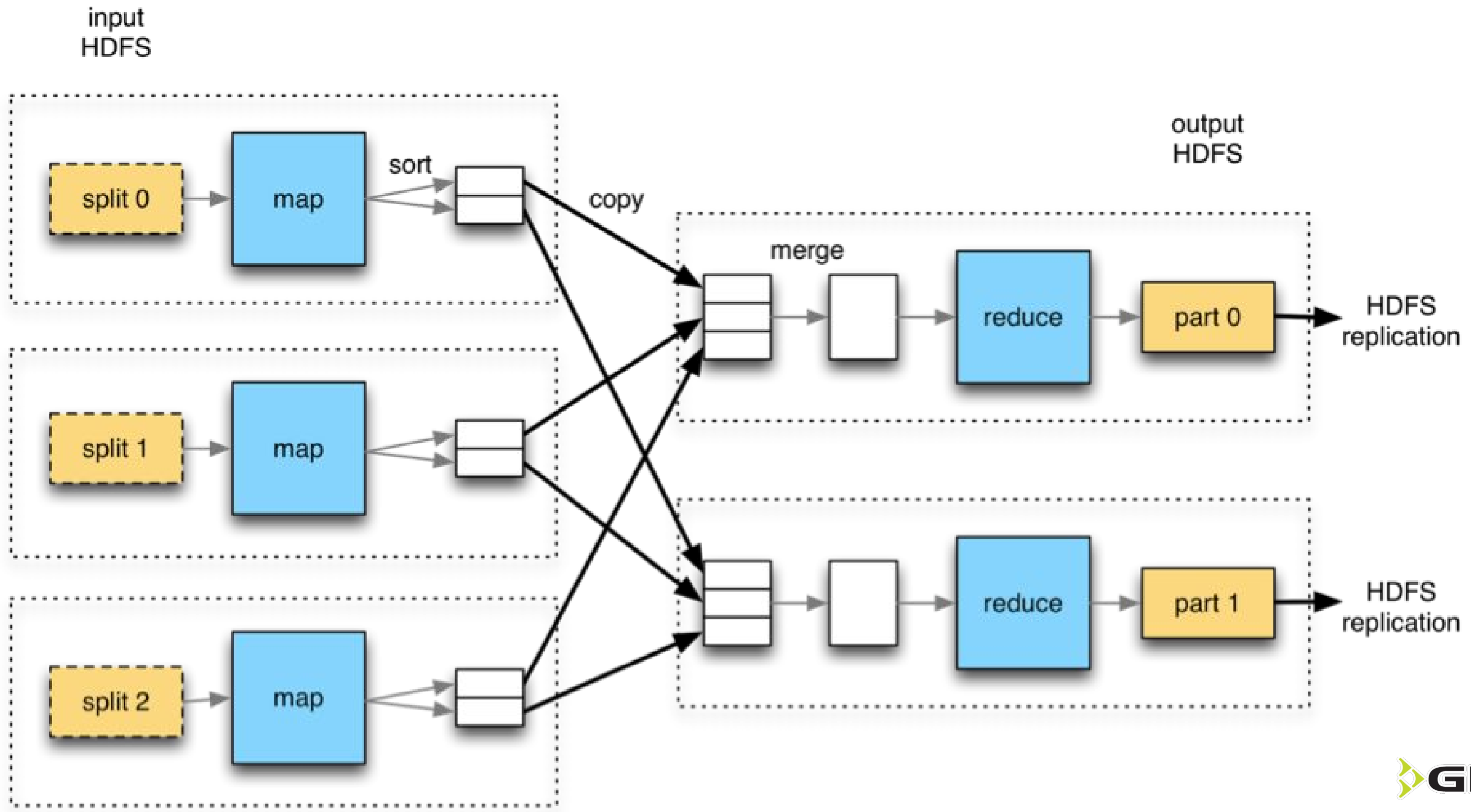
Objectives

- Mapreduce APIs
- Basics
  - Mapreduce Java API
  - Maven Dependencies
  - WordCount - New Java API
  - WordCount - Old Java API
  - Execution on the cluster
  - MRUnit
  - Hadoop IO data Types (Writables)
  - Writing a Writable custom Class
- Advanced
  - How does the data flow through the tasks
- References and Exercises

Globant
we are ready

MapReduce Java Advanced (1)

How does the data flow through the tasks?



MapReduce Java Advanced (2)

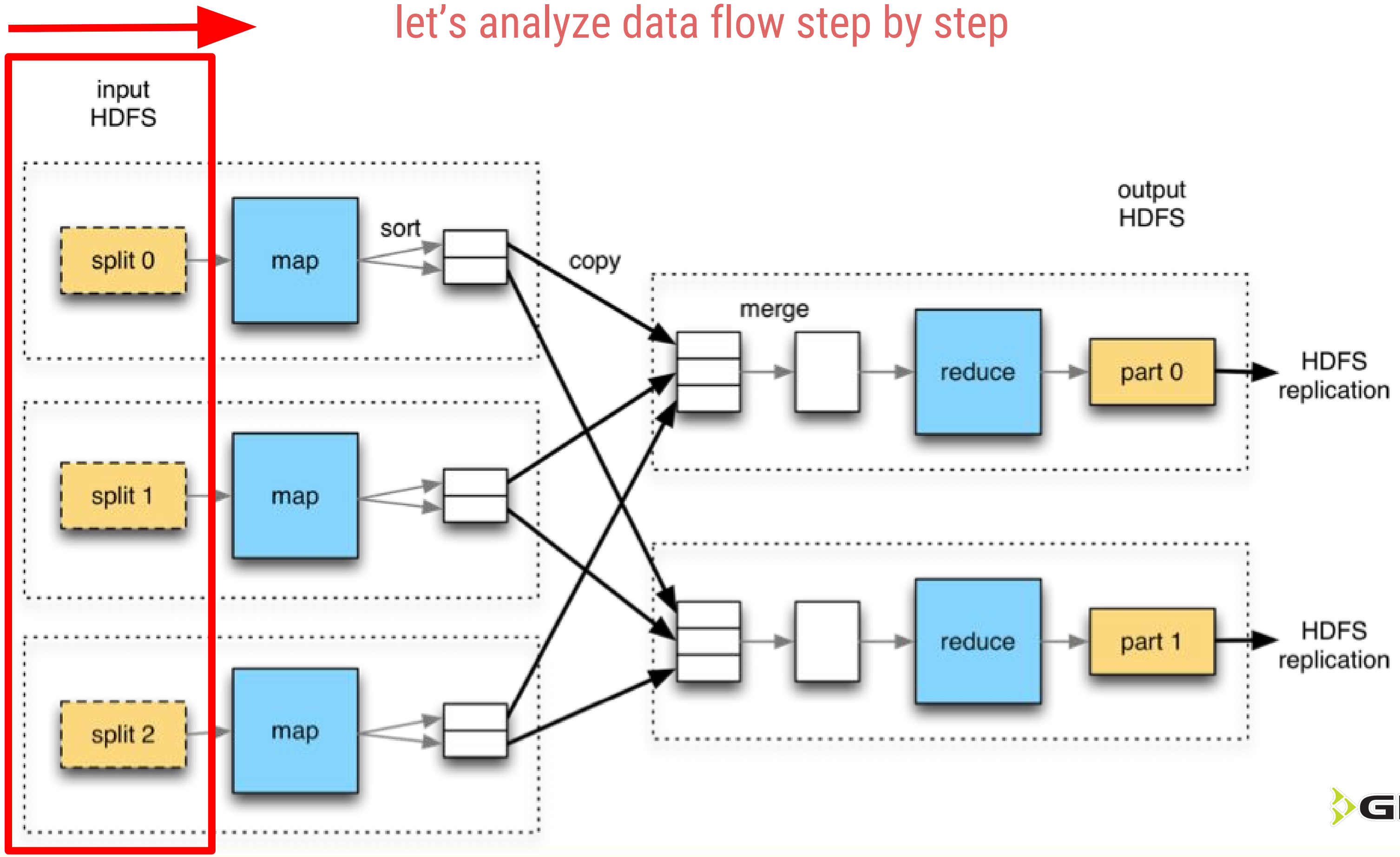
> Let’s considered the default settings of a mapreduce job (in red):

```
public class MinimalMapReduceWithDefaults extends Configured implements Tool {

        @Override
        public int run(String[] args) throws Exception {
                Job job = new Job( getConf(), "WordCount" );
                job.setJarByClass( getClass() );
                FileInputFormat.addInputPath( job, new Path(args[0]) );
                FileOutputFormat.setOutputPath( job, new Path(args[1]) );
                job.setInputFormatClass( TextInputFormat.class );
                job.setMapperClass( Mapper.class );
                job.setMapOutputKeyClass( LongWritable.class );
                job.setMapOutputValueClass( Text.class );
                job.setPartitionerClass( HashPartitioner.class );
                job.setNumReduceTasks( 1 );
                job.setReducerClass( Reducer.class );
                job.setOutputKeyClass( LongWritable.class );
                job.setOutputValueClass( Text.class );
                job.setOutputFormatClass( TextOutputFormat.class );
                return job.waitForCompletion(true) ? 0 : 1;
        }

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

*Source: Hadoop: The Definitive Guide

MapReduce Java Advanced (3)

let's analyze data flow step by step



MapReduce Java Advanced (4)

> how the splits work:

```
public class MinimalMapReduceWithDefaults extends Configured implements Tool {

        @Override
        public int run(String[] args) throws Exception {
                Job job = new Job( getConf(), "WordCount" );
                job.setJarByClass( getClass() );
                FileInputFormat.addInputPath( job, new Path(args[0]) );
                FileOutputFormat.setOutputPath( job, new Path(args[1]) );
                job.setInputFormatClass( TextInputFormat.class );
                job.setMapperClass( Mapper.class );
                job.setMapOutputKeyClass( LongWritable.class );
                job.setMapOutputValueClass( Text.class );
                job.setPartitionerClass( HashPartitioner.class );
                job.setNumReduceTasks( 1 );
                job.setReducerClass( Reducer.class );
                job.setOutputKeyClass( LongWritable.class );
                job.setOutputValueClass( Text.class );
                job.setOutputFormatClass( TextOutputFormat.class );
                return job.waitForCompletion(true) ? 0 : 1;
        }

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

*Source: Hadoop: The Definitive Guide

MapReduce Java Advanced (5)

- abstract `InputSplit` represents a Split:

```
public abstract class InputSplit {
        public abstract long getLength() throws IOException, InterruptedException;
        public abstract String[] getLocations() throws IOException, InterruptedException;
}
```

- Extended class `FileSplit` is used. It represents splits of files with methods like:
    - `getPath()`: local file path.
    - `getStart()`: byte offset of the start of the split from the beginning of the file.
    - `getLength()`: size of the split.

MapReduce Java Advanced (6)

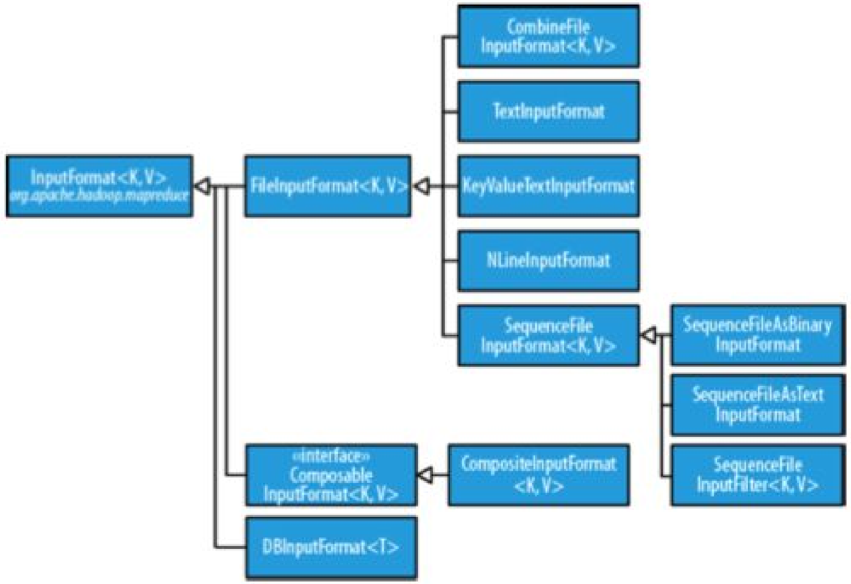
- abstract InputFormat is the base to handle inputs (text files, binary files, databases, etc):

```
public abstract class InputFormat<K, V> {
        public abstract List<InputSplit> getSplits(JobContext context) throws IOException, InterruptedException;
        public abstract RecordReader<K, V> createRecordReader(InputSplit split,TaskAttemptContext context) throws IOException,
InterruptedException;
}
```

- Extended class FileInputFormat defines input files in Java Driver
- 2nd-level extended class TextInputFormat:
  - is the default InputFormat class for the job.
  - parses text files into LongWritable and Text for the mapper.
- The client calls getSplits() of InputFormat to obtain the splits for the job and sends them to the JobTracker(JT)/ResourceManager(RM)-AppMaster(AM)
- The JT/RM-AM passes the splits to the TaskTrackers/NodeManager (and they to the map tasks via context)

MapReduce Java Advanced (7)

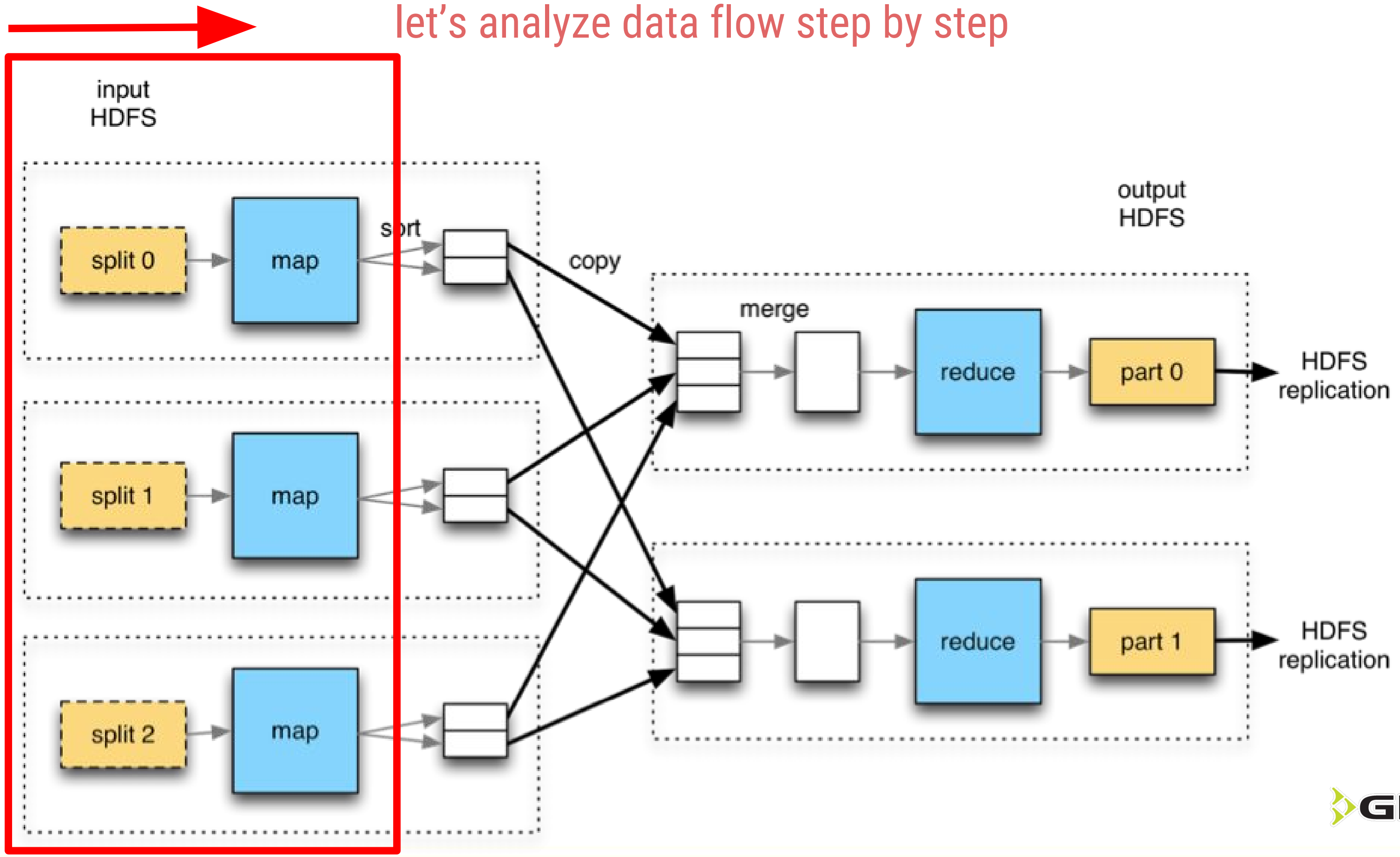
> Hierarchy of `InputFormat` subclasses*:


InputFormat<K, V>
org.apache.hadoop.mapreduce
FileInputFormat<K, V>
CombineFile InputFormat<K, V>
TextInputFormat
KeyValueTextInputFormat
NLineInputFormat
SequenceFile InputFormat<K, V>
SequenceFileAsBinary InputFormat
SequenceFileAsText InputFormat
SequenceFile InputFilter<K, V>
«interface» Composable InputFormat<K, V>
CompositeInputFormat <K, V>
DBInputFormat<T>

*Source: Hadoop: The Definitive Guide

MapReduce Java Advanced (8)

let's analyze data flow step by step



MapReduce Java Advanced (9)

- how the map task works:

```
public class MinimalMapReduceWithDefaults extends Configured implements Tool {

        @Override
        public int run(String[] args) throws Exception {
                Job job = new Job( getConf(), "WordCount" );
                job.setJarByClass( getClass() );
                FileInputFormat.addInputPath( job, new Path(args[0]) );
                FileOutputFormat.setOutputPath( job, new Path(args[1]) );
                job.setInputFormatClass( TextInputFormat.class );
                job.setMapperClass( Mapper.class );
                job.setMapOutputKeyClass( LongWritable.class );
                job.setMapOutputValueClass( Text.class );
                job.setPartitionerClass( HashPartitioner.class );
                job.setNumReduceTasks( 1 );
                job.setReducerClass( Reducer.class );
                job.setOutputKeyClass( LongWritable.class );
                job.setOutputValueClass( Text.class );
                job.setOutputFormatClass( TextOutputFormat.class );
                return job.waitForCompletion(true) ? 0 : 1;
        }

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

*Source: Hadoop: The Definitive Guide

MapReduce Java Advanced (10)

> abstract Mapper implements run():

```
public void run(Context context) throws IOException, InterruptedException {
        setup( context );
        while ( context.nextKeyValue() ) {
                map(context.getCurrentKey(), context.getCurrentValue(), context);
        }
        cleanup(context);
}
```

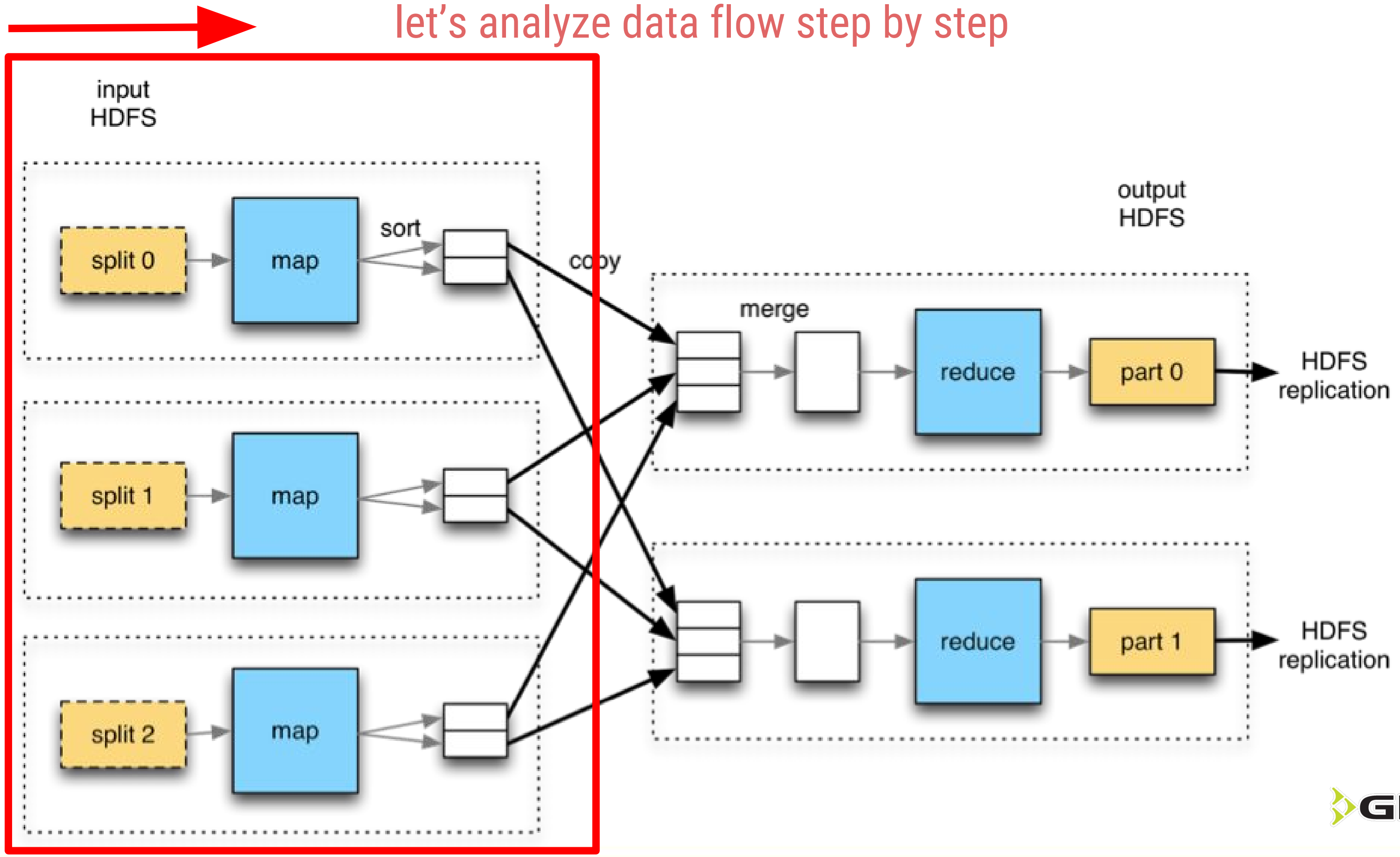
> setup() method passes the split to createRecordReader() of InputFormat to open the file and obtain an iterator of records for the part of the file.

> custom map() is called for each of the records passing key, value

> cleanup() closes files, releases resources, etc.

MapReduce Java Advanced (11)

let's analyze data flow step by step



MapReduce Java Advanced (12)

> How the partition works

```
public class MinimalMapReduceWithDefaults extends Configured implements Tool {

        @Override
        public int run(String[] args) throws Exception {
                Job job = new Job( getConf(), "WordCount" );
                job.setJarByClass( getClass() );
                FileInputFormat.addInputPath( job, new Path(args[0]) );
                FileOutputFormat.setOutputPath( job, new Path(args[1]) );
                job.setInputFormatClass( TextInputFormat.class );
                job.setMapperClass( Mapper.class );
                job.setMapOutputKeyClass( LongWritable.class );
                job.setMapOutputValueClass( Text.class );
                job.setPartitionerClass( HashPartitioner.class );
                job.setNumReduceTasks( 1 );
                job.setReducerClass( Reducer.class );
                job.setOutputKeyClass( LongWritable.class );
                job.setOutputValueClass( Text.class );
                job.setOutputFormatClass( TextOutputFormat.class );
                return job.waitForCompletion(true) ? 0 : 1;
        }

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

*Source: Hadoop: The Definitive Guide

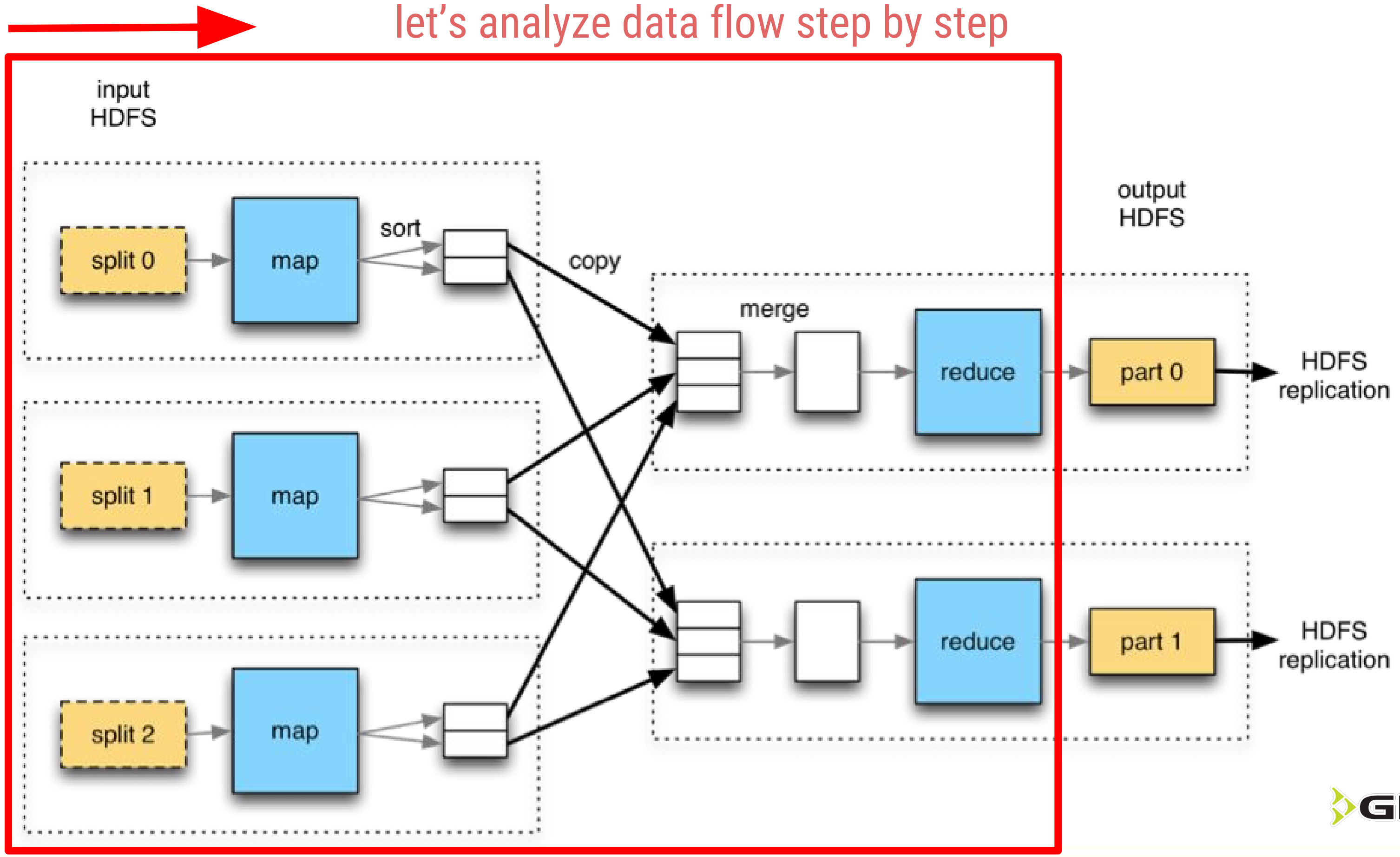
MapReduce Java Advanced (13)

- The default partition function:

```
public class HashPartitioner<K, V> extends Partitioner<K, V> {
	public int getPartition(K key, V value,int numReduceTasks) {
		return (key.hashCode() & Integer.MAX_VALUE) % numReduceTasks;
	}
}
```

MapReduce Java Advanced (14)

let's analyze data flow step by step



MapReduce Java Advanced (15)

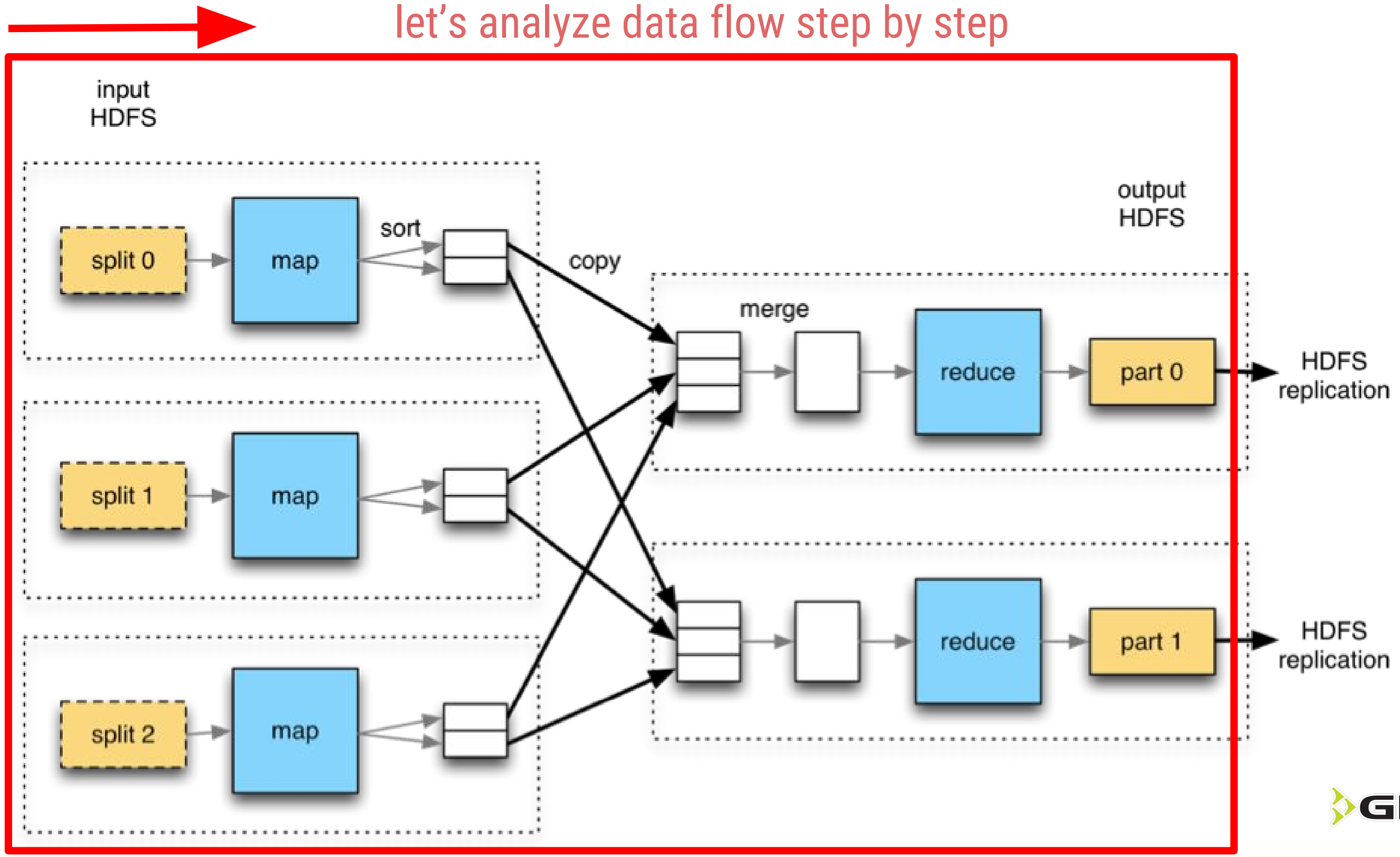
> How the reduce task works

```
public class MinimalMapReduceWithDefaults extends Configured implements Tool {

        @Override
        public int run(String[] args) throws Exception {
                Job job = new Job( getConf(), "WordCount" );
                job.setJarByClass( getClass() );
                FileInputFormat.addInputPath( job, new Path(args[0]) );
                FileOutputFormat.setOutputPath( job, new Path(args[1]) );
                job.setInputFormatClass( TextInputFormat.class );
                job.setMapperClass( Mapper.class );
                job.setMapOutputKeyClass( LongWritable.class );
                job.setMapOutputValueClass( Text.class );
                job.setPartitionerClass( HashPartitioner.class );
                job.setNumReduceTasks( 1 );
                job.setReducerClass( Reducer.class );
                job.setOutputKeyClass( LongWritable.class );
                job.setOutputValueClass( Text.class );
                job.setOutputFormatClass( TextOutputFormat.class );
                return job.waitForCompletion(true) ? 0 : 1;
        }

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

*Source: Hadoop: The Definitive Guide

MapReduce Java Advanced (16)

let's analyze data flow step by step



MapReduce Java Advanced (15)

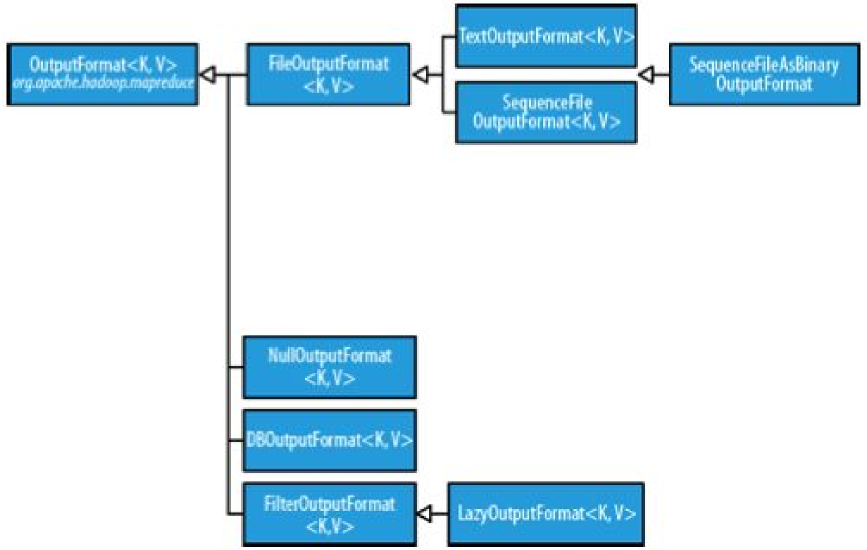
>

```
public class                                          extends              implements
        @Override
        public int run(String[] args) throws Exception {
                Job job = new Job( getConf(), "WordCount" );
                job.setJarByClass( getClass() );
                FileInputFormat.addInputPath( job, new Path(args[0]) );
                FileOutputFormat.setOutputPath( job, new Path(args[1]) );
                job.setInputFormatClass( TextInputFormat.class );
                job.setMapperClass( Mapper.class );
                job.setMapOutputKeyClass( LongWritable.class );
                job.setMapOutputValueClass( Text.class );
                job.setPartitionerClass( HashPartitioner.class );
                job.setNumReduceTasks( 1 );
                job.setReducerClass( Reducer.class );
                job.setOutputKeyClass( LongWritable.class );
                job.setOutputValueClass( Text.class );
                job.setOutputFormatClass( TextOutputFormat.class );
                return job.waitForCompletion(true) ? 0 : 1;
        }

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

*Source: Hadoop: The Definitive Guide

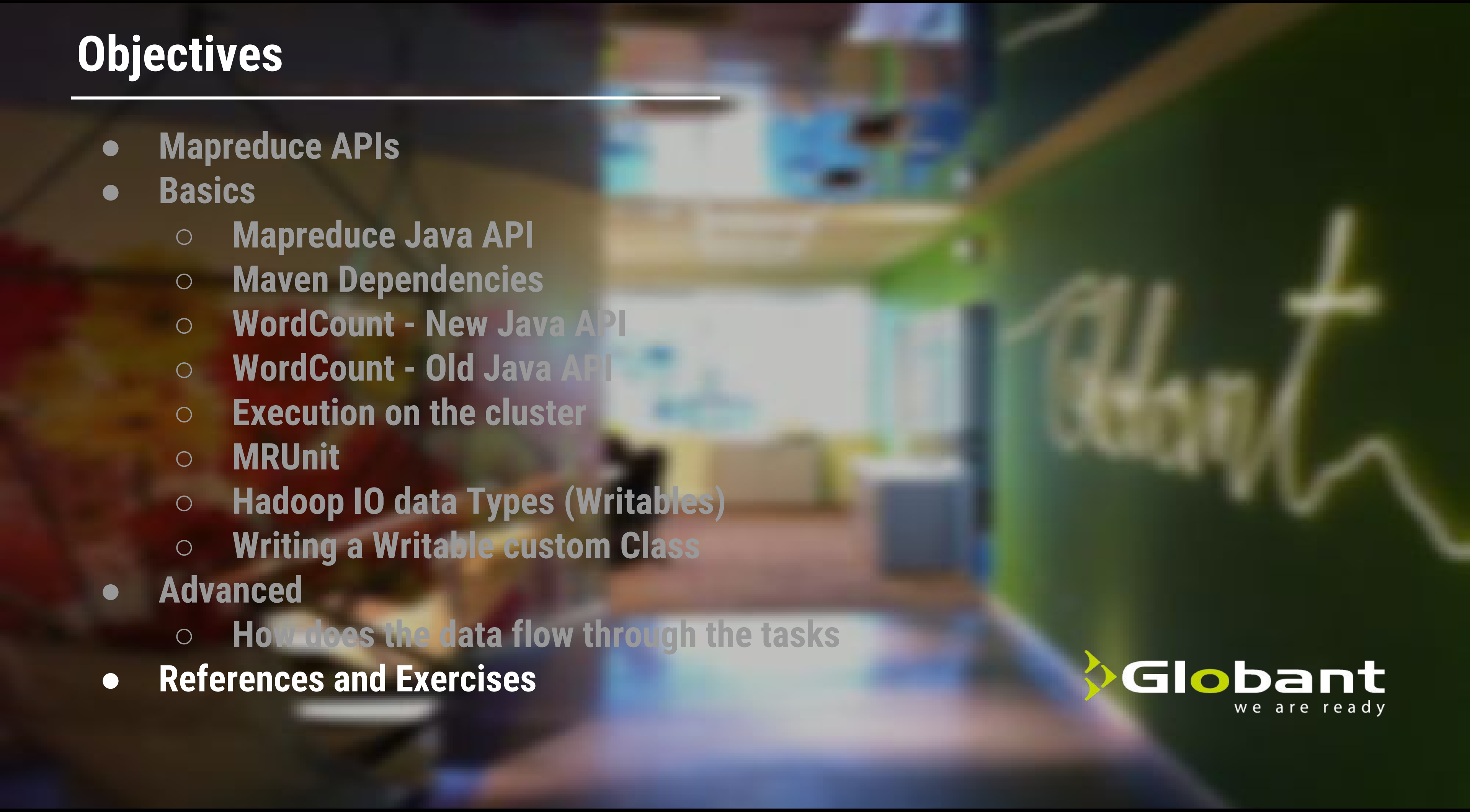
MapReduce Java Advanced (16)

- abstract `OutputFormat` has a hierarchy corresponding to `InputFormat`



- extended `FileOutputFormat` is the default OutputFormat class for the job.
- `FileOutputFormat` writes reducer output key,value pair to part-XXXX file using their `toString()` method

Objectives



- Mapreduce APIs
- Basics
  - Mapreduce Java API
  - Maven Dependencies
  - WordCount - New Java API
  - WordCount - Old Java API
  - Execution on the cluster
  - MRUnit
  - Hadoop IO data Types (Writables)
  - Writing a Writable custom Class
- Advanced
  - How does the data flow through the tasks
- **References and Exercises**

Globant
we are ready

Mapreduce - References

- Includes some documentation about the old api

http://hadoop.apache.org/docs/r1.2.1/mapred_tutorial.html

- *White, Tom*. Hadoop: The Definitive Guide

Mapreduce - Exercise 1

Inverted Index

Given a directory with books in txt format, write a mapreduce which outputs an inverted index, i.e., a table that associates a word with the books and the corresponding positions at which it occurs (http://en.wikipedia.org/wiki/Inverted_index).

Dataset URL: here

HDFS Cluster DataSet path: `/user/hadoop/mapreduce/data/books`

hint 1: Suggested output example (not real data):

```
Love     alice_in_wonderland.txt:100,the_prince.txt:900,the_prince.txt:1050
```

hint 2: The mapper doesn't receive explicitly the filename as input, but the context has access to the `InputSplit` and casting it to a `FileSplit` object, the `getPath()` method can be used.

Mapreduce - Exercise 2

Column-wise Variance (s^2) of a matrix

Given a csv file without headers, calculate the sample variance (s^2) of each column. (http://en.wikipedia.org/wiki/Variance)

$$s^2 = \frac{1}{(N-1)} \sum_{i=1}^{N} (x_i - \bar{x})^2$$

HDFS DataSet path: `/user/hadoop/mapreduce/data/matrix`

¡¡¡DON'T DOWNLOAD IT FROM THE CLUSTER $$$$!!!

hint 1: Suggested output: columnIndex<tab>sampleVariance.

Example:

```
0    135.6
1    2.2
2    536.9
...
```

hint 2: Assume the file has only numeric values and no entries are missing (no NULLs or empty).

hint 3: Focus on the mapreduce. (If any) other plumbing code is required, do it manually, afterwards implement it if you have the time (preferably with python).

Mapreduce - Exercise 2

Column-wise Variance (s^2) of a matrix

question 1:

Assuming your matrix is very big (both rows and columns), how many reducers does it make sense to have?

question 2:

How many mapreduce jobs do you need?

$$\boxed{s^2 = \frac{1}{(N-1)} \sum_{i=1}^{N} (x_i - \bar{x})^2}$$

Thank You!

Globant
we are ready