Example 23:
MapReduce with HBase tables

Objective: Count Notifications by Notification type

Objective: Count Notifications by Notification type

HBase only provides 'Create-Read-Update-Delete' operations

Operations across row/ tables are not supported

Ex: Joins, Group by

Objective: Count Notifications by Notification type

You can integrate HBase with Hadoop's MapReduce

With MapReduce you can perform any data processing task

As long as you relax the requirement of low latency

Objective: Count Notifications by Notification type

To set up a MapReduce task, just define 2 functions

map() reduce()

Objective: Count Notifications by Notification type

map() reduce()

The rest is taken care of by Hadoop!

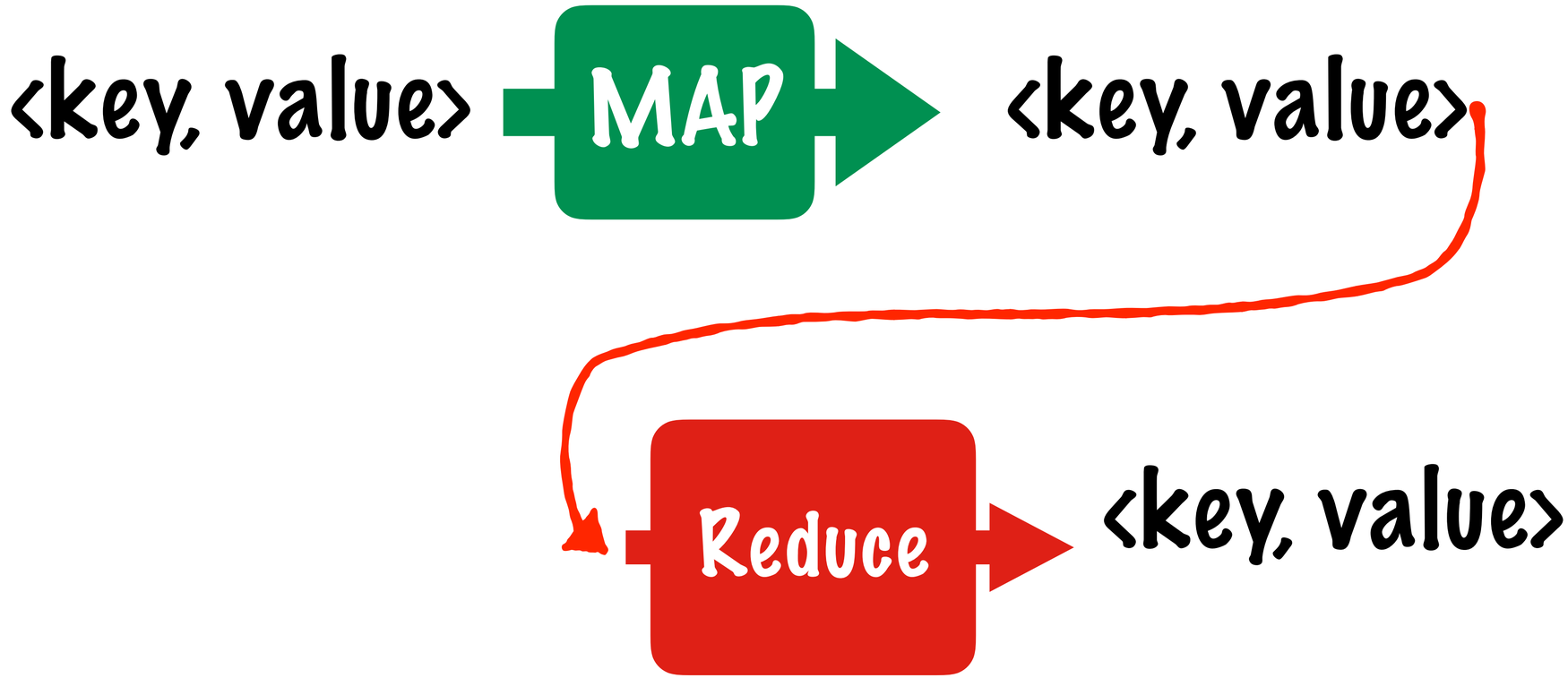
MapReduce

map() reduce()

This paradigm is
driven by a key insight

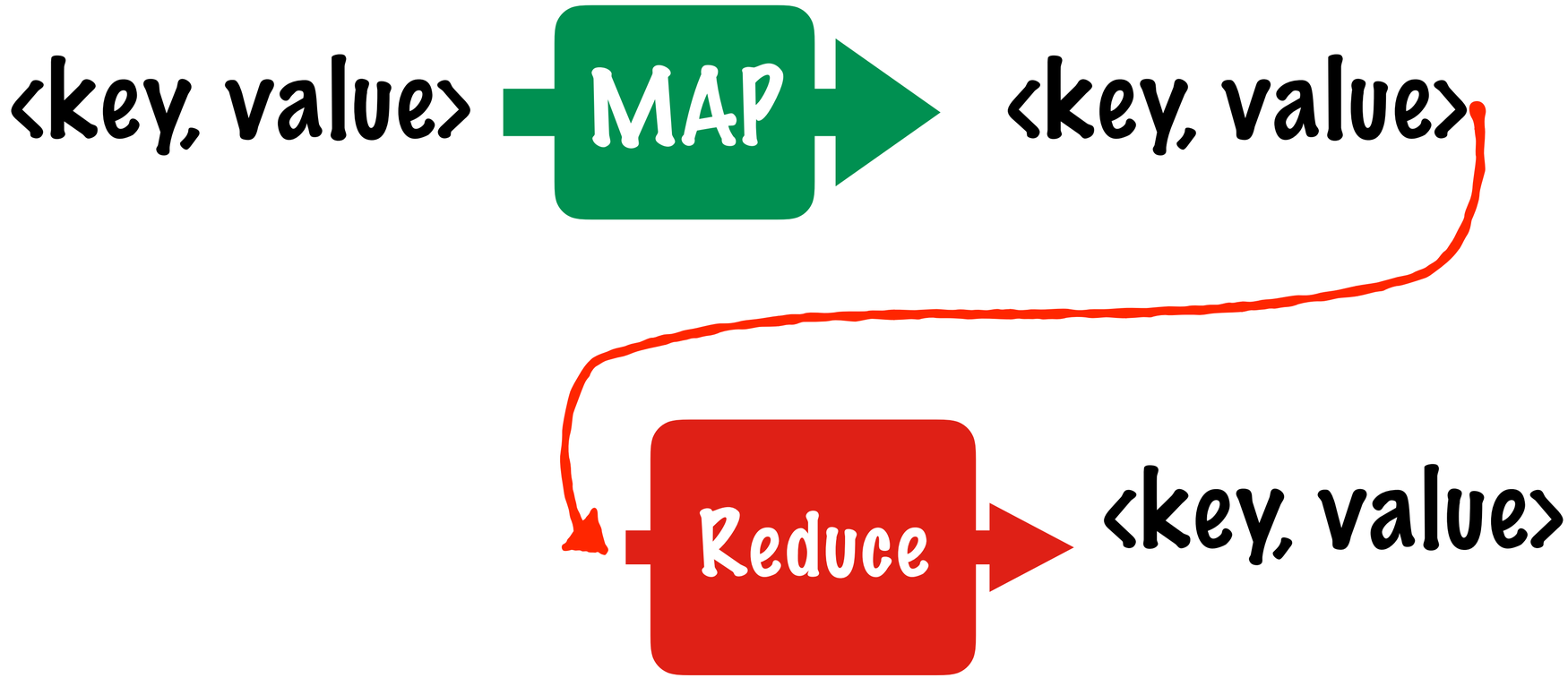
Objective: Count Notifications by Notification type

All the inputs and outputs in MapReduce are in the form of key, value pairs



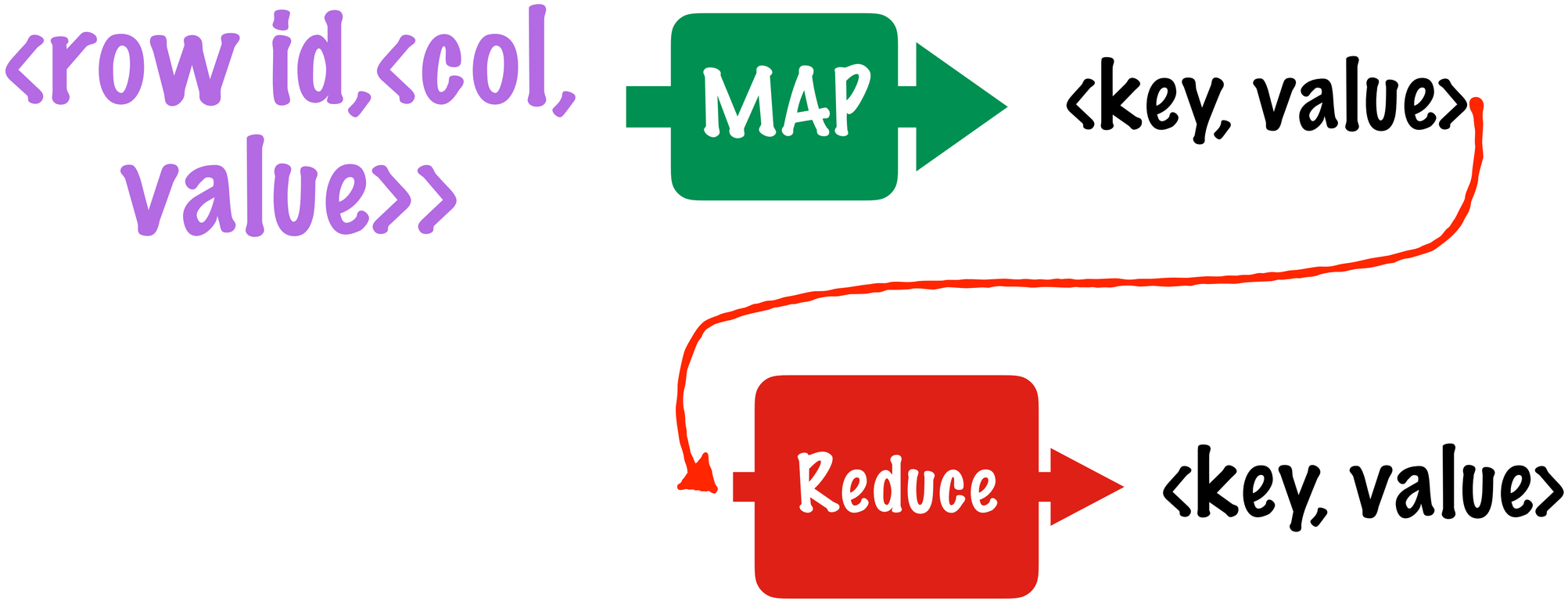
Objective: Count Notifications by Notification type

An HBase table is a map consisting of key value pairs



Objective: Count Notifications by Notification type

An HBase table is a map consisting of key value pairs



<row id,<col, value>>

MAP → ? → Reduce → ?

Input

row	column	value
2	attributes : type	Friend request status
2	attributes : for user	Daniel
2	attributes : from user	Ryan
3	attributes : type	Like
3	attributes : for user	Brendan
3	attributes : from user	Rick
3	attributes : for_thing	link
3	attributes : link	“link”

Output

row id	column	value
Friend request	metrics:count	10
Like	metrics:count	15
Comment	metrics:count	3

<row id,<col, value>>

This represents all columns for 1 row id in the notifications table

row	column	value
2	attributes : type	Friend request status
2	attributes : for user	Daniel
2	attributes : from user	Ryan

<row id,<col, value>>

row	column	value
2	attributes : type	Friend request status
2	attributes : for user	Daniel
2	attributes : from user	Ryan

<FriendRequest,1>

<row id,<col, value>>

<type,1>

This logic is implemented in Java inside the map() function of a Mapper class

<row id,<col, value>> 

<type,1>

TableMapper is a special subclass of the Mapper class that can read from HBase tables

<row id,<col, value>>

MAP

<type,1>

Reduce

?

Input

row	column	value
2	attributes : type	Friend request status
2	attributes : for user	Daniel
2	attributes : from user	Ryan
3	attributes : type	Like
3	attributes : for user	Brendan
3	attributes : from user	Rick
3	attributes : for_thing	link
3	attributes : link	“link”

Output

row id	column	value
Friend request	metrics:count	10
Like	metrics:count	15
Comment	metrics:count	3

<type,1> → Reduce → ?

The reducer will combine the values with the same type to compute the counts per notification

row id	column	value
Friend request	metrics:count	10
Like	metrics:count	15
Comment	metrics:count	3

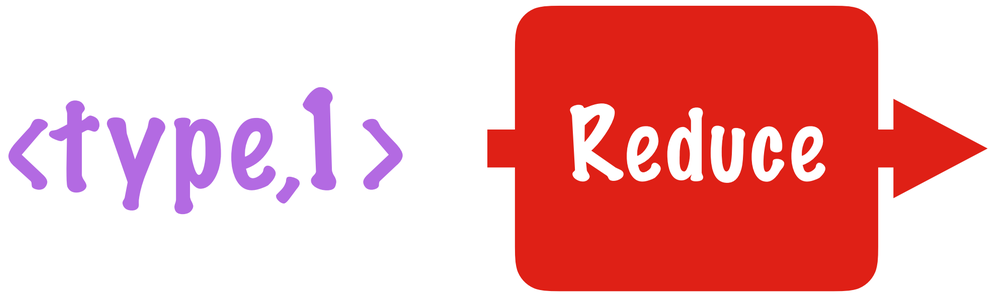
This logic is implemented in Java inside the reduce() function of a Reducer class

row id	column	value
Friend request	metrics:count	10
Like	metrics:count	15
Comment	metrics:count	3

<type,1> → Reduce → ?

TableReducer is a special subclass of the Reducer class that can write to HBase tables

row id	column	value
Friend request	metrics:count	10
Like	metrics:count	15
Comment	metrics:count	3



TableReducer

produces Put objects that will be written to the output table

row id	column	value
Friend request	metrics:count	10
Like	metrics:count	15
Comment	metrics:count	3

<type,1> Reduce <null,Put>

row id	column	value
Friend request	metrics:count	10
Like	metrics:count	15
Comment	metrics:count	3

<row id,<col, value>>

MAP

<type,1>

Reduce

<null,Put>

Input

row	column	value
2	attributes : type	Friend request status
2	attributes : for user	Daniel
2	attributes : from user	Ryan
3	attributes : type	Like
3	attributes : for user	Brendan
3	attributes : from user	Rick
3	attributes : for_thing	link
3	attributes : link	“link”

Output

row id	column	value
Friend request	metrics:count	10
Like	metrics:count	15
Comment	metrics:count	3

Objective: Count Notifications by Notification type

All Hadoop MapReduce programs have the same flow

Step 1: Write a map() function

`TableMapper`

Step 2: Write a reduce() function

`TableReducer`

Step 3: Setup a driver that points to our map and reduce implementations

TableMapper Class

<input key type,
input value type,
output key type,
output value type>

Mapper Class

TableReducer Class

<input key type,
input value type,
output key type,
output value type>

Reducer Class

Both the mapper and reducer are generic classes with type parameters

TableMapper Class

<input key type,
input value type,
output key type,
output value type>

Mapper Class

TableReducer Class

<input key type,
input value type,
output key type,
output value type>

Reducer Class

The output types of the Mapper should match the input types of the Reducer

TableMapper Class

<input key type,
input value type,
output key type,
output value type>

Mapper Class

TableReducer Class

<input key type,
input value type,
output key type,
output value type>

Reducer Class

Hadoop uses the **ImmutableBytesWritable** and **Result** classes to represent keys, values in HBase tables

TableMapper Class

TableReducer Class

These 2 classes are used by a Job that is configured in the Main Class

Main Class

Job Object

TableMapper Class

TableReducer Class

The Job has a bunch of properties that need to be configured

Main Class

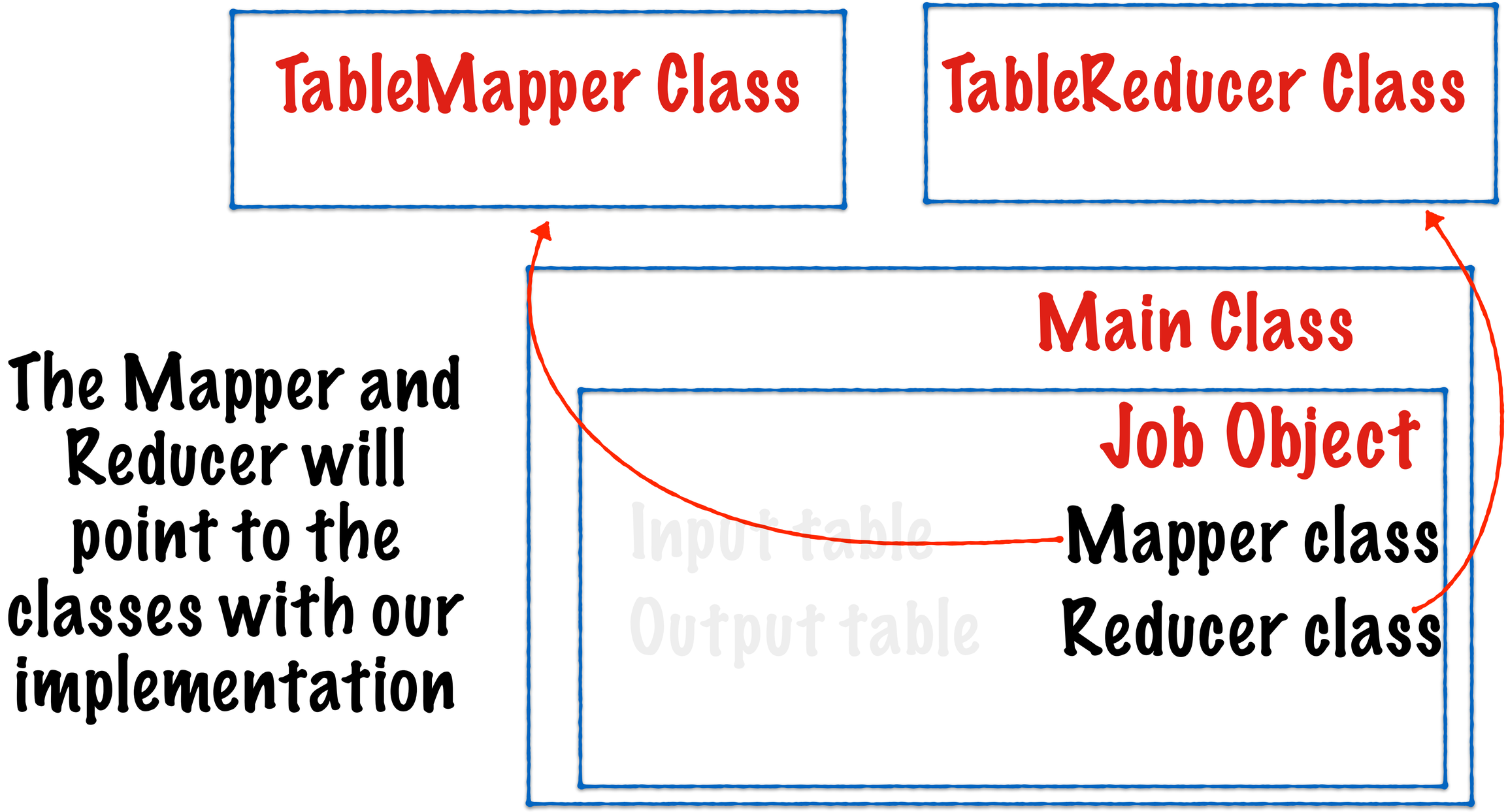
Job Object

Input table

Output table

Mapper class

Reducer class



The Mapper and Reducer will point to the classes with our implementation

TableMapper Class

TableReducer Class

Main Class

Job Object

Input table

Output table

Mapper class

Reducer class

TableMapper Class

```
public  class MyMapper extends TableMapper<Text, IntWritable> {
    public static final byte[] CF = "attributes".getBytes();
    public static final byte[] ATTR1 = "type".getBytes();

    private final IntWritable ONE = new IntWritable(1);
    private Text text = new Text();

    public void map(ImmutableBytesWritable row, Result value, Context context)
throws IOException, InterruptedException {
        String val = new String(value.getValue(CF, ATTR1));
        text.set(val);     // we can only emit Writables...

        context.write(text, ONE);
    }
}
```

TableMapper Class

This class represents the code for the step

TableMapper Class

```
public class MyMapper extends TableMapper<Text, IntWritable>{
    public static final byte[] CF = "attributes".getBytes();
    public static final byte[] ATTR1 = "type".getBytes();

    private final IntWritable ONE = new IntWritable(1);
    private Text text = new Text();

    public void map(ImmutableBytesWritable row, Result value, Context context) throws IOException, InterruptedException {
        String val = new String(value.getValue(CF, ATTR1));
        text.set(val);     // we can only emit Writables...

        context.write(text, ONE);
    }
}
```

TableMapper is a subclass of Hadoop's Mapper class

TableMapper Class

```
public class MyMapper extends TableMapper<Text, IntWritable>{
    public static final byte[] CF = "attributes".getBytes();
    public static final byte[] ATTR1 = "type".getBytes();

    private final IntWritable ONE = new IntWritable(1);
    private Text text = new Text();

    public void map(ImmutableBytesWritable row, Result value, Context context) throws IOException, InterruptedException {
        String val = new String(value.getValue(CF, ATTR1));
        text.set(val);     // we can only emit Writables...

        context.write(text, ONE);
    }
}
```

TableMapper is used to read and map data from HBase tables

TableMapper Class

Output
key type

```
public class MyMapper extends TableMapper<Text, IntWritable>{
    public static final byte[] CF = "attributes".getBytes();
    public static final byte[] ATTR1 = "type".getBytes();

    private final IntWritable ONE = new IntWritable(1);
    private Text text = new Text();

    public void map(ImmutableBytesWritable row, Result value, Context context) throws IOException, InterruptedException {
        String val = new String(value.getValue(CF, ATTR1));
        text.set(val);     // we can only emit Writables...

        context.write(text, ONE);
    }
}
```

Output
Value type

TableMapper has
2 type parameters

TableMapper Class

```
public class MyMapper extends TableMapper<Text, IntWritable>{
    public static final byte[] CF = "attributes".getBytes();
    public static final byte[] ATTR1 = "type".getBytes();

    private final IntWritable ONE = new IntWritable(1);
    private Text text = new Text();

    public void map(ImmutableBytesWritable row, Result value, Context context) throws IOException, InterruptedException {
        String val = new String(value.getValue(CF, ATTR1));
        text.set(val);     // we can only emit Writables...

        context.write(text, ONE);
    }
}
```

<type,1>

TableMapper Class

```
public class MyMapper extends TableMapper<Text, IntWritable>{
    public static final byte[] CF = "attributes".getBytes();
    public static final byte[] ATTR1 = "type".getBytes();

    private final IntWritable ONE = new IntWritable(1);
    private Text text = new Text();

    public void map(ImmutableBytesWritable row, Result value, Context context) throws IOException, InterruptedException {
        String val = new String(value.getValue(CF, ATTR1));
        text.set(val);     // we can only emit Writables...

        context.write(text, ONE);
    }
}
```

Normally mappers also have type parameters for the inputs

TableMapper Class

```
public class MyMapper extends TableMapper<Text, IntWritable>{
    public static final byte[] CF = "attributes".getBytes();
    public static final byte[] ATTR1 = "type".getBytes();

    private final IntWritable ONE = new IntWritable(1);
    private Text text = new Text();

    public void map(ImmutableBytesWritable row, Result value, Co
        String val = new String(value.getValue(CF, ATTR1));
        text.set(val);     // we can only emit Writables...

        context.write(text, ONE);
    }
}
```

The input key type and value type for TableMapper are automatically set by Hadoop

<ImmutableBytesWritable,Result>

TableMapper Class

```
public  class MyMapper extends TableMapper<Text, IntWritable> {
    public static final byte[] CF = "attributes".getBytes();
    public static final byte[] ATTR1 = "type".getBytes();

    private final IntWritable ONE = new IntWritable(1);
    private Text text = new Text();

    public void map(ImmutableBytesWritable row, Result value, Context co
        String val = new String(value.getValue(CF, ATTR1));
        text.set(val);     // we can only emit Writables...

        context.write(text, ONE);
    }
}
```

We will be checking the column name for every value in the table

TableMapper Class

```
public  class MyMapper extends TableMapper<Text, IntWritable> {
    public static final byte[] CF = "attributes".getBytes();
    public static final byte[] ATTR1 = "type".getBytes();

    private final IntWritable ONE = new IntWritable(1);
    private Text text = new Text();

    public void map(ImmutableBytesWritable row, Result value, Context context) throws
        String val = new String(value.getValue(CF, ATTR1));
        text.set(val);     // we can only emit Writables...

        context.write(text, ONE);
    }
}
```

Setting up these variables as byte arrays ahead of time, will save us doing the conversion with every value

TableMapper Class

```
public  class MyMapper extends TableMapper<Text, IntWritable> {
    public static final byte[] CF = "attributes".getBytes();

    public static final byte[] ATTR1 = "type".getBytes();

    private final IntWritable ONE = new IntWritable(1);
    private Text text = new Text();

    public void map(ImmutableBytesWritable row, Result value, Context context) throws IO
      String val = new String(value.getValue(CF, ATTR1));
      text.set(val);     // we can only emit Writables...

      context.write(text, ONE);
    }
}
```

The output objects are also set up beforehand and will be reused

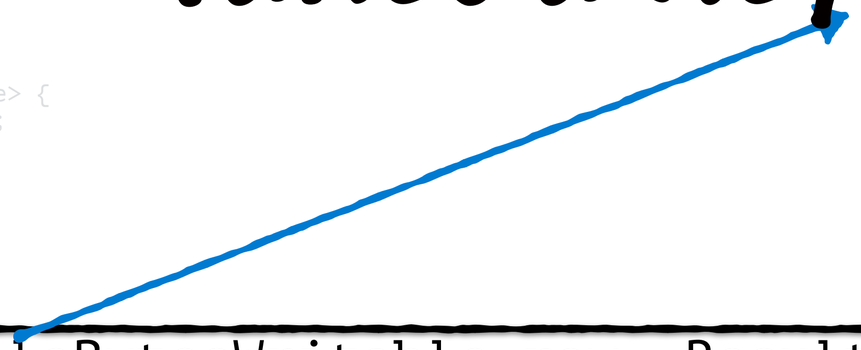
TableMapper Class

The logic of the Mapper is implemented in the map method

```
public  class MyMapper extends TableMapper<Text, IntWritable> {
    public static final byte[] CF = "attributes".getBytes();
    public static final byte[] ATTR1 = "type".getBytes();

    private final IntWritable ONE = new IntWritable(1);

    private Text text = new Text();

    public void map(ImmutableBytesWritable row, Result value, Context
context) throws IOException, InterruptedException {
                String val = new String(value.getValue(CF, ATTR1));
        text.set(val);     // we can only emit Writables...

        context.write(text, ONE);
    }
}
```

TableMapper Class

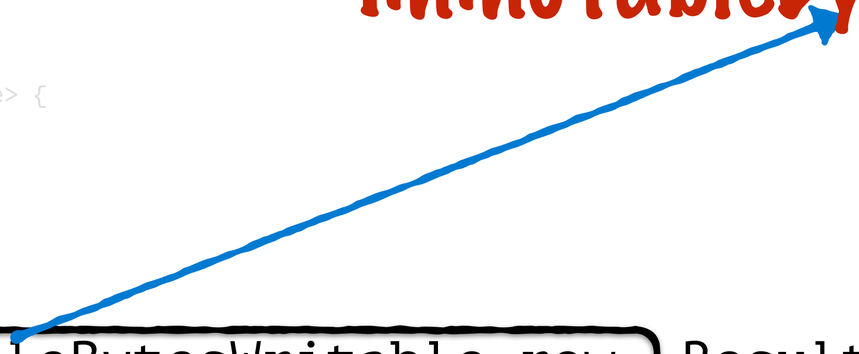
The map() method takes a Key and a Value



```
public  class MyMapper extends TableMapper<Text, IntWritable> {
    public static final byte[] CF = "attributes".getBytes();
    public static final byte[] ATTR1 = "type".getBytes();

    private final IntWritable ONE = new IntWritable(1);

    private Text text = new Text();

    public void map(ImmutableBytesWritable row, Result value, Context
context) throws IOException, InterruptedException {
                String val = new String(value.getValue(CF, ATTR1));
        text.set(val);     // we can only emit Writables...

        context.write(text, ONE);
    }
}
```

TableMapper Class

The Keys in HBase Tables are row ids represented using an ImmutableBytesWritable



```
public  class MyMapper extends TableMapper<Text, IntWritable> {
    public static final byte[] CF = "attributes".getBytes();
    public static final byte[] ATTR1 = "type".getBytes();

    private final IntWritable ONE = new IntWritable(1);

    private Text text = new Text();

    public void map(ImmutableBytesWritable row, Result value, Context
context) throws IOException, InterruptedException {
                String val = new String(value.getValue(CF, ATTR1));
        text.set(val);     // we can only emit Writables...

        context.write(text, ONE);
    }
}
```

TableMapper Class

All the columns and values for that row are represented by a Result object

```
public  class MyMapper extends TableMapper<Text, IntWritable> {
    public static final byte[] CF = "attributes".getBytes();
    public static final byte[] ATTR1 = "type".getBytes();

    private final IntWritable ONE = new IntWritable(1);

    private Text text = new Text();

    public void map(ImmutableBytesWritable row, Result value, Context
context) throws IOException, InterruptedException {
                String val = new String(value.getValue(CF, ATTR1));
        text.set(val);     // we can only emit Writables...

        context.write(text, ONE);
    }
}
```

TableMapper Class

Look up the value for the relevant column family and column in the Result object

```
public  class MyMapper extends TableMapper<Text, IntWritable> {
    public static final byte[] CF = "attributes".getBytes();
    public static final byte[] ATTR1 = "type".getBytes();

    private final IntWritable ONE = new IntWritable(1);
    private Text text = new Text();

    public void map(ImmutableBytesWritable row, Result value, Context context) throws IOException, InterruptedException {
        String val = new String(value.getValue(CF, ATTR1));
        text.set(val);      // we can only emit Writables...
        context.write(text, ONE);
    }
}
```

TableMapper Class

```
public  class MyMapper extends TableMapper<Text, IntWritable> {
    public static final byte[] CF = "attributes".getBytes();
    public static final byte[] ATTR1 = "type".getBytes();

    private final IntWritable ONE = new IntWritable(1);
    private Text text = new Text();

    public void map(ImmutableBytesWritable row, Result value, Context context) throws IOException, InterruptedException {
        String val = new String(value.getValue(CF, ATTR1));
        text.set(val);      // we can only emit Writables...
        context.write(text, ONE);
    }
}
```

Write the output <type,1> to the context

TableMapper Class

TableReducer Class

Main Class

Job Object

Input table

Output table

Mapper class

Reducer class

TableReducer Class

```
public  class MyTableReducer extends TableReducer<Text, IntWritable,
ImmutableBytesWritable> {
    public static final byte[] CF = "metrics".getBytes();
    public static final byte[] COUNT = "count".getBytes();

    public void reduce(Text key, Iterable<IntWritable> values, Context context)
throws IOException, InterruptedException {
        int i = 0;
        for (IntWritable val : values) {
            i += val.get();
        }
        Put put = new Put(Bytes.toBytes(key.toString()));
        put.addColumn(CF, COUNT, Bytes.toBytes(i));

        context.write(null, put);
    }
}
```

TableReducer Class

This class represents the code for the step

TableReducer Class

This class is a generic

```
public class MyTableReducer extends
TableReducer
<Text,
IntWritable,
ImmutableBytesWritable> {
```

Input Key Type

Input Value type

Output Key type

```
    public static final byte[] CF = "metrics".getBytes();
    public static final byte[] COUNT = "count".getBytes();

    public void reduce(Text key, Iterable<IntWritable> values, Context context) throws IOException, InterruptedException {
        int i = 0;
        for (IntWritable val : values) {
            i += val.get();
        }
        Put put = new Put(Bytes.toBytes(key.toString()));
        put.addColumn(CF, COUNT, Bytes.toBytes(i));

        context.write(null, put);
    }
}
```

The output value type is set by default to a Put

TableReducer Class

```
public class MyTableReducer extends
TableReducer
<Text,
IntWritable,
ImmutableBytesWritable> {
    public static final byte[] CF = "metrics".getBytes();
    public static final byte[] COUNT = "count".getBytes();

    public void reduce(Text key, Iterable<IntWritable> values, Context context) throws IOException, InterruptedException {
        int i = 0;
        for (IntWritable val : values) {
            i += val.get();
        }
        Put put = new Put(Bytes.t                (k                g());
        put.addColumn(CF, COUNT,                 to                );

        context.write(null, put);
    }
}
```

In our example, the output from map has the form <type , 1>

TableReducer Class

```
public  class MyTableReducer extends TableReducer<Text, IntWritable, ImmutableBytesWritable> {
    public static final byte[] CF = "metrics".getBytes();
    public static final byte[] COUNT = "count".getBytes();

    public void reduce(Text key, Iterable<IntWritable> values, Context context) throws IOException, InterruptedException {
        int i = 0;
        for (IntWritable val : values) {
            i += val.get();
        }
        Put put = new Put(Bytes.toBytes(key.toString()));
        put.addColumn(CF, COUNT, Bytes.toBytes(i));

        context.write(null, put);
    }
}
```

Set up variables to represent the column family and column in the output table

TableReducer Class

```
public  class MyTableReducer extends TableReducer<Text, IntWritable, ImmutableBytesWritable> {
    public static final byte[] CF = "metrics".getBytes();
    public static final byte[] COUNT = "count".getBytes();
```

```
public void reduce(Text key, Iterable<IntWritable> values,
Context context) throws IOException, InterruptedException {
```

```
        int i = 0;
        for (IntWritable val : values) {
            i += val.get();
        }
        Put put = new Put(Bytes.toBytes(key.toString()));
        put.addColumn(CF, COUNT, Bytes.toBytes(i));

        context.write(null, put);
    }
}
```

This method holds the logic for the the reduce step

TableReducer Class

```
public  class MyTableReducer extends TableReducer<Text, IntWritable, ImmutableBytesWritable> {
    public static final byte[] CF = "metrics".getBytes();
    public static final byte[] COUNT = "count".getBytes();

public void reduce(Text key, Iterable<IntWritable> values,
Context context) throws IOException, InterruptedException {

                        int i = 0;

        for (IntWritable val : values) {
            i += val.get();
        }
        Put put = new Put(Bytes.toBytes(key.toString()));
        put.addColumn(CF, COUNT, Bytes.toBytes(i));

        context.write(null, put);
    }
}
```

Hadoop does the job of collecting all values with the same key into an Iterable

TableReducer Class

```
public  class MyTableReducer extends TableReducer<Text, IntWritable, ImmutableBytesWritable> {
    public static final byte[] CF = "metrics".getBytes();
    public static final byte[] COUNT = "count".getBytes();

    public void reduce(Text key, Iterable<IntWritable> values, Context context) throws IOException, InterruptedException {
        int i = 0;
        for (IntWritable val : values) {
            i += val.get();
        }
        Put put = new Put(Bytes.toBytes(key.toString()));
        put.addColumn(CF, COUNT, Bytes.toBytes(i));

        context.write(null, put);
    }
}
```

Sum all the 1s to get the count for a particular type

TableReducer Class

```
public  class MyTableReducer extends TableReducer<Text, IntWritable, ImmutableBytesWritable> {
    public static final byte[] CF = "metrics".getBytes();
    public static final byte[] COUNT = "count".getBytes();

    public void reduce(Text key, Iterable<IntWritable> values, Context context) throws IOException, InterruptedException {
        int i = 0;
        for (IntWritable val : values) {
            i += val.get();
        }
        Put put = new Put(Bytes.toBytes(key.toString()));
        put.addColumn(CF, COUNT, Bytes.toBytes(i));

        context.write(null, put);
    }
}
```

Create a Put object with row id = notification type

TableReducer Class

```
public  class MyTableReducer extends TableReducer<Text, IntWritable, ImmutableBytesWritable> {
    public static final byte[] CF = "metrics".getBytes();
    public static final byte[] COUNT = "count".getBytes();

    public void reduce(Text key, Iterable<IntWritable> values, Context context) throws IOException, InterruptedException {
        int i = 0;
        for (IntWritable val : values) {
            i += val.get();
        }
        Put put = new Put(Bytes.toBytes(key.toString()));
        put.addColumn(CF, COUNT, Bytes.toBytes(i));

        context.write(null, put);
    }
}
```

Add the value to the relevant column for that row id

TableReducer Class

```
public  class MyTableReducer extends TableReducer<Text, IntWritable, ImmutableBytesWritable> {
    public static final byte[] CF = "metrics".getBytes();
    public static final byte[] COUNT = "count".getBytes();

    public void reduce(Text key, Iterable<IntWritable> values, Context context) throws IOException, InterruptedException {
        int i = 0;
        for (IntWritable val : values) {
            i += val.get();
        }
        Put put = new Put(Bytes.toBytes(key.toString()));
        put.addColumn(CF, COUNT, Bytes.toBytes(i));

        context.write(null, put);
    }
}
```

Write the output to context

TableMapper Class

TableReducer Class

Main Class

Job Object

Input table

Output table

Mapper class

Reducer class

Main Class

We'll write a driver class whose main() method will point to our Mapper and Reducer

```java
package com.company;

import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
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;

public class WordCount {

    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);



        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);
    }
}
```

Main Class

Some boilerplate to setup a new Job object

```
public class Main {

    public static void main(String[] args) throws Exception {

        Configuration conf = HBaseConfiguration.create();
        Job job = new Job(conf, "ExampleSummary");
        job.setJarByClass(Main.class);

        String sourceTable = "notifications";
        String targetTable = "summary";

        Scan scan = new Scan();
        scan.setCaching(500);        // 1 is the default in Scan, which will be bad for MapReduce jobs
        scan.setCacheBlocks(false);  // don't set to true for MR jobs
        // set other scan attrs

        TableMapReduceUtil.initTableMapperJob(
                sourceTable,        // input table
                scan,               // Scan instance to control CF and attribute selection
                MyMapper.class,     // mapper class
                Text.class,         // mapper output key
                IntWritable.class,  // mapper output value
                job);
        TableMapReduceUtil.initTableReducerJob(
                targetTable,        // output table
                MyTableReducer.class,    // reducer class
                job);
        job.setNumReduceTasks(1);   // at least one, adjust as required

        boolean b = job.waitForCompletion(true);
        if (!b) {
            throw new IOException("error with job!");
        }

    }

}
```

Main Class

```
public class Main {
    public static void main(String[] args) throws Exception {

        Configuration conf = HBaseConfiguration.create();
        Job job = new Job(conf, "ExampleSummary");
        job.setJarByClass(Main.class);

        String sourceTable = "notifications";
        String targetTable = "summary";

        Scan scan = new Scan();
        scan.setCaching(500);        // 1 is the default in Scan, which will be bad for MapReduce jobs
        scan.setCacheBlocks(false);  // don't set to true for MR jobs
        // set other scan attrs

        TableMapReduceUtil.initTableMapperJob(
                sourceTable,        // input table
                scan,               // Scan instance to control CF and attribute selection
                MyMapper.class,     // mapper class
                Text.class,         // mapper output key
                IntWritable.class,  // mapper output value
                job);
        TableMapReduceUtil.initTableReducerJob(
                targetTable,        // output table
                MyTableReducer.class,    // reducer class
                job);
        job.setNumReduceTasks(1);   // at least one, adjust as required

        boolean b = job.waitForCompletion(true);
        if (!b) {
            throw new IOException("error with job!");
        }

    }
}
```

The names of the input and output tables

Main Class

```
Scan scan = new Scan();
scan.setCaching(500);
scan.setCacheBlocks(false);
```

We set up a Scan object to read through the rows in the input table

Main Class

```
Scan scan = new Scan();
scan.setCaching(500);
scan.setCacheBlocks(false);
```

The scan object needs to have some properties set for performance tuning

Main Class

```
TableMapReduceUtil.initTableMapperJob(
        sourceTable,
              scan,
        MyMapper.class,      // mapper class
        Text.class,          // mapper output key
        IntWritable.class,   // mapper output value
        job);
```

Set up the Mapper class

Main Class

```
TableMapReduceUtil.initTableMapperJob(
                  sourceTable,
                        scan,
                  MyMapper.class,      // mapper class
                  Text.class,          // mapper output key
                  IntWritable.class,   // mapper output value
                  job);
```

The input table and a Scan object to read from it

Main Class

```
TableMapReduceUtil.initTableMapperJob(
                    sourceTable,
                            scan,
                    MyMapper.class,      // mapper class
                    Text.class,          // mapper output key
                    IntWritable.class,   // mapper output value
                    job);
```

The TableMapper class

Main Class

```
TableMapReduceUtil.initTableMapperJob(
                sourceTable,
                      scan,
                MyMapper.class,          // mapper class
                Text.class,              // mapper output key
                IntWritable.class,       // mapper output value
                job);
```

The output datatypes for the Mapper

Main Class

Set up the Reducer

```
TableMapReduceUtil.initTableReducerJob(
                targetTable,
                MyTableReducer.class, // reducer class
                job);
        job.setNumReduceTasks(1);
```

Main Class

The output table

```
TableMapReduceUtil.initTableReducerJob(
                  targetTable,
                  MyTableReducer.class, // reducer class
                  job);
        job.setNumReduceTasks(1);
```

Main Class

TableReducer class

```
TableMapReduceUtil.initTableReducerJob(
                targetTable,
                MyTableReducer.class, // reducer class
                job);
        job.setNumReduceTasks(1);
```

Main Class

Setting the number of reducers

```
TableMapReduceUtil.initTableReducerJob(
                targetTable,
                MyTableReducer.class, // reducer class
                job);
        job.setNumReduceTasks(1);
```

Main Class

```
public class Main {
    public static void main(String[] args) throws Exception {
        Configuration conf = HBaseConfiguration.create();
        Job job = new Job(conf, "ExampleSummary");
        job.setJarByClass(Main.class);

        String sourceTable = "notifications";
        String targetTable = "summary";

        Scan scan = new Scan();
        scan.setCaching(500);        // 1 is the default in Scan, which will be bad for MapReduce jobs
        scan.setCacheBlocks(false);  // don't set to true for MR jobs
        // set other scan attrs

        TableMapReduceUtil.initTableMapperJob(
                sourceTable,        // input table
                scan,               // Scan instance to control CF and attribute selection
                MyMapper.class,     // mapper class
                Text.class,         // mapper output key
                IntWritable.class,  // mapper output value
                job);
        TableMapReduceUtil.initTableReducerJob(
                targetTable,        // output table
                MyTableReducer.class,    // reducer class
                job);
        job.setNumReduceTasks(1);   // at least one, adjust as required

    }
}
```

waitForCompletion() method will submit the Job for execution and wait for it to complete

```
boolean b = job.waitForCompletion(true);
if (!b) {
    throw new IOException("error with job!");
}
```

Code Along:

Setting up a notification service

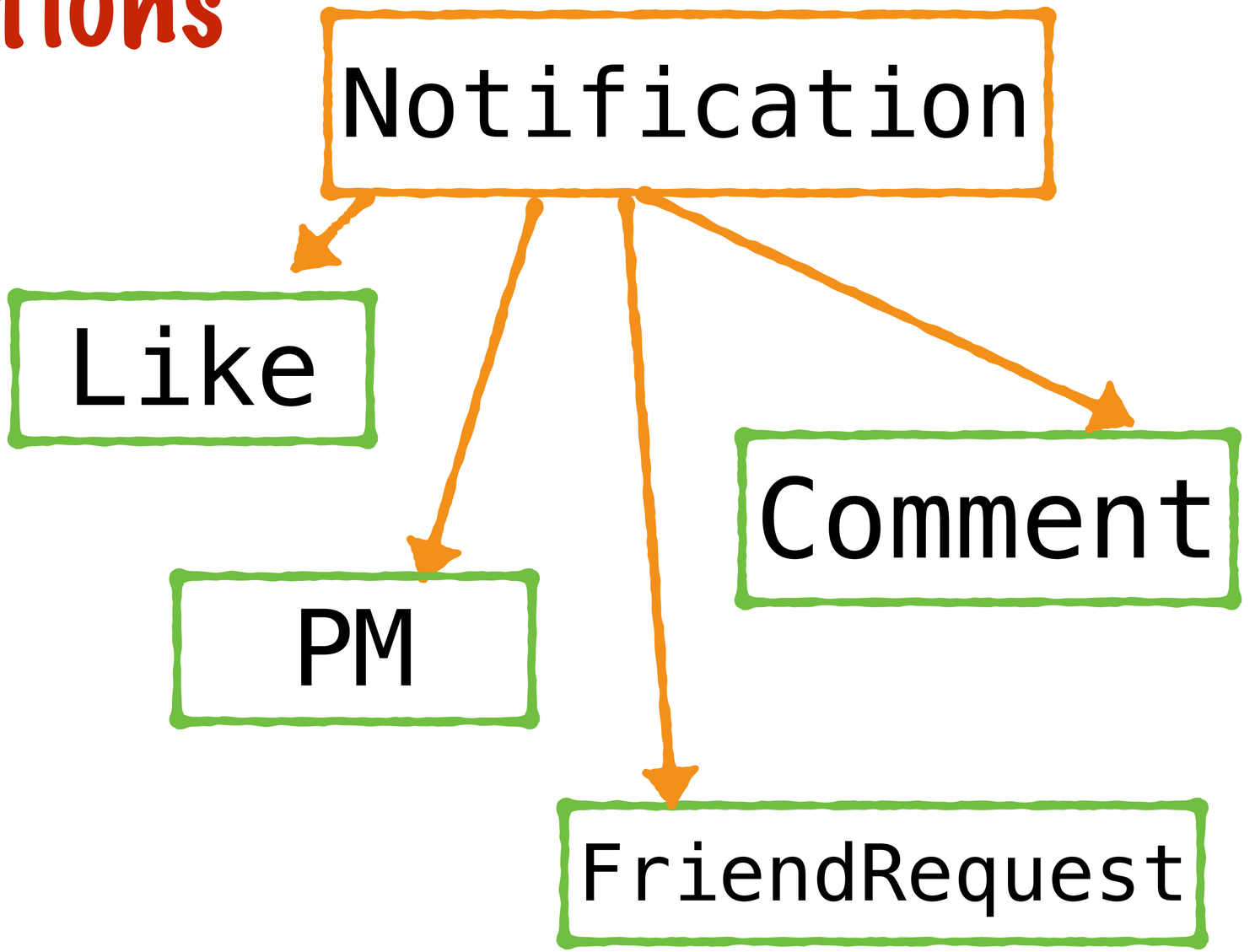
Setting up a notification service

Let's set up a Notification service that can

1. Create different types of notifications
2. Retrieve notifications for a specified user

Types of Notifications

We'll set up a class hierarchy to represent different types of notifications



Types of Notifications

Each notification has a separate set of attributes that will be inserted into HBase

- Like
- Comment
- PM
- FriendRequest

Types of Notifications

Each notification knows how to display itself

- Like
- Comment
- PM
- FriendRequest

Setting up a notification service

Let's set up a Notification service that can

1. Create different types of notifications
2. Retrieve notifications for a specified user

NotificationManager

1. Create different types of notifications
2. Retrieve notifications for a specified user

NotificationManager

```
public class NotificationManager {

    public Notification createNotification(Notification.NotificationType type

    public void addNotification(Notification notification) throws IOException

    public List<Notification> getUserNotifications(String user) throws Except

    private Map<String,String> parseResults(Result result){...}
}
```

A Factory method to create different types of Notifications, given the type and attributes

NotificationManager

```
public class NotificationManager {

    public Notification createNotification(Notification.NotificationType type

    public void addNotification(Notification notification) throws IOException

    public List<Notification> getUserNotifications(String user) throws Except

    private Map<String,String> parseResults(Result result){...}
}
```

A method to add any type of Notification to a HBase notifications table

NotificationManager

```
public class NotificationManager {

    public Notification createNotification(Notification.NotificationType type

    public void addNotification(Notification notification) throws IOExceptio

    public List<Notification> getUserNotifications(String user) throws Except

    private Map<String,String> parseResults(Result result){...}
}
```

It creates a Put object and writes it to HBase

NotificationManager

```
public class NotificationManager {

    public Notification createNotification(Notification.NotificationType type

    public void addNotification(Notification notification) throws IOException

    public List<Notification> getUserNotifications(String user) throws Except

    private Map<String,String> parseResults(Result result){...}
}
```

A method to return notifications for a given user

NotificationManager

```
public class NotificationManager {

    public Notification createNotification(Notification.NotificationType type

    public void addNotification(Notification notification) throws IOException

    public List<Notification> getUserNotifications(String user) throws Except

    private Map<String,String> parseResults(Result result){...}
}
```

It uses a Scan object with a Filter to retrieve all notifications for a user

NotificationManager

```
public class NotificationManager {

    public Notification createNotification(Notification.NotificationType type

    public void addNotification(Notification notification) throws IOException

    public List<Notification> getUserNotifications(String user) throws Except

    private Map<String,String> parseResults(Result result){...}
}
```

A utility method to parse Result objects returned by HBase

NotificationManager

```
public class NotificationManager {

    public Notification createNotification(Notification.NotificationType type

    public void addNotification(Notification notification) throws IOException

    public List<Notification> getUserNotifications(String user) throws Except

    private Map<String,String> parseResults(Result result){...}
}
```

This will be used by the getUserNotifications method