Select MR

Where MR

Group by MR

Having MR

Join MR

We've looked at each of these operations in MapReduce

Select MR

Where MR

Group by MR

Having MR

Join MR

Overall, these constructs lend themselves really well to the MapReduce paradigm

Select MR

Where MR

Group by MR

Having MR

Join MR

A MapReduce strategy for a join can get pretty complicated

Join MR

Joins are really complex operations, even in relational databases

Join MR

There are so many types of joins

Outer

Inner

Left

Right

Join MR

Outer
Inner
Left
Right

Each would require a different strategy

Join MR

Let's look at

An outer join where the join key is unique in the left table

NAMES

SYMBOL	NAME
RIL	Reliance
NESTLEIND	Nestle

TRADES

SYMBOL	SERIES	OPEN	HIGH	LOW	CLOSE

```
SELECT N.SYMBOL, N.NAME, T.HIGH , T.TIMESTAMP
FROM  NAMES N
OUTER JOIN TRADES T
ON N.SYMBOL = T.SYMBOL
```

Let's say we have a table with

Symbol, Name

NAMES

SYMBOL	NAME
RIL	Reliance
NESTLEIND	Nestle

TRADES

SYMBOL	SERIES	OPEN	HIGH	LOW	CLOSE

```
SELECT N.SYMBOL, N.NAME, T.HIGH , T.TIMESTAMP
FROM  NAMES N
OUTER JOIN TRADES T
ON N.SYMBOL = T.SYMBOL
```

For every record in Names, Trades

NAMES

SYMBOL	NAME
RIL	Reliance
NESTLEIND	Nestle

TRADES

SYMBOL	SERIES	OPEN	HIGH	LOW	CLOSE

```
SELECT N.SYMBOL, N.NAME, T.HIGH , T.TIMESTAMP
FROM  NAMES N
OUTER JOIN TRADES T
ON N.SYMBOL = T.SYMBOL
```

Write out a record with these columns

NAMES

SYMBOL	NAME
RIL	Reliance
NESTLEIND	Nestle

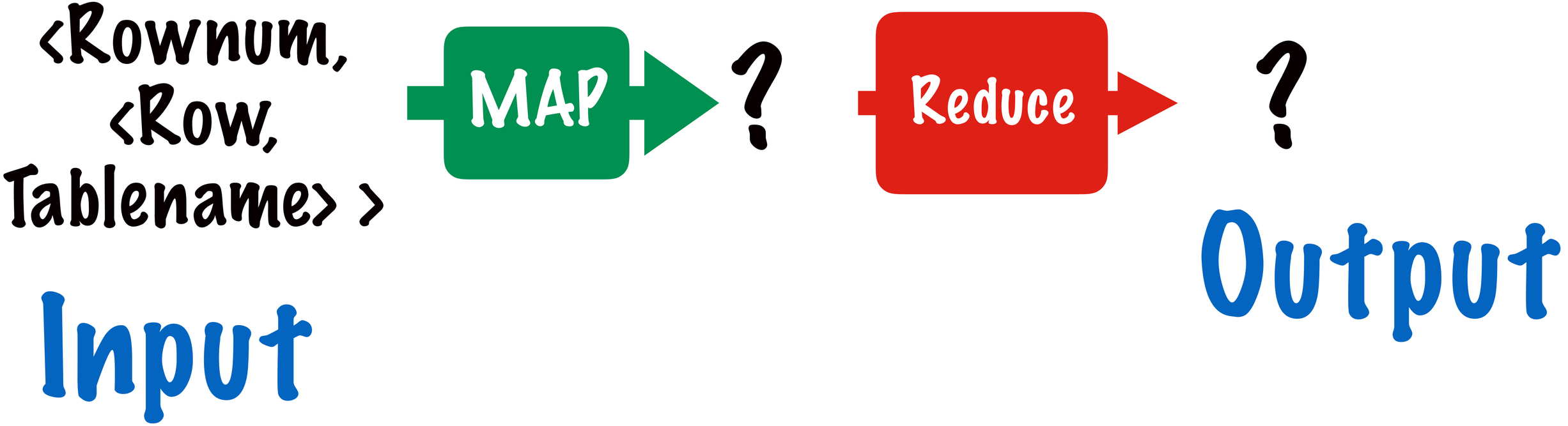
TRADES

SYMBOL	SERIES	OPEN	HIGH	LOW	CLOSE

```
SELECT N.SYMBOL, N.NAME, T.HIGH , T.TIMESTAMP
FROM  NAMES N
OUTER JOIN TRADES T
ON N.SYMBOL = T.SYMBOL
```

If the symbol matches

```
SELECT N.SYMBOL, N.NAME, T.HIGH , T.TIMESTAMP
FROM  NAMES N
OUTER JOIN TRADES T
ON N.SYMBOL = T.SYMBOL
```



<Rownum,
<Row, Tablename> >

MAP → ?

The input row is a line in a csv file

20MICRONS	EQ	31.8	31.8	30.75	30.85	30.8	31.15	76530	2389916.9	01-APR-2014	538

Or

NESTLEIND	**Nestle**

<Rownum,
<Row, Tablename> >

MAP → ?

20MICRONS	EQ	31.8	31.8	30.75	30.85	30.8	31.15	76530	2389916.9	01-APR-2014	538

NESTLEIND	**Nestle**

These come to the Mapper as a Text object

<Rownum,
<Row, Tablename> >

MAP → ?

20MICRONS	EQ	31.8	31.8	30.75	30.85	30.8	31.15	76530	2389916.9	01-APR-2014	538

NESTLEIND	**Nestle**

The Mapper should parse the Text to extract all the fields

<Rownum,
<Row, Tablename> >

MAP → ?

NAMES N

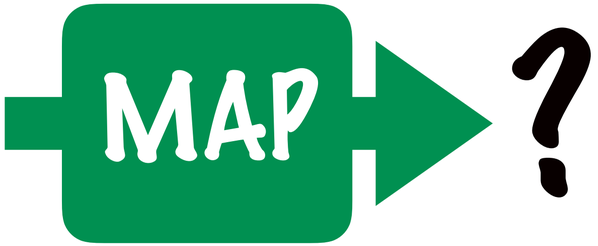
N.SYMBOL, N.NAME

TRADES T

T.SYMBOL,
T.HIGH , T.TIMESTAMP

We can use MultipleInputs class in Java to use a different Mapper for each table

<Rownum,
<Row, Tablename> >



?

N.SYMBOL, N.NAME

T.SYMBOL, T.HIGH ,
T.TIMESTAMP

The reducer should be able to combine records with the same Symbol

ON N.SYMBOL = T.SYMBOL

<Rownum,
<Row, Tablename> >

<Symbol, ? >

N.SYMBOL, N.NAME

T.SYMBOL,
T.HIGH , T.TIMESTAMP

The value should just be the remaining columns of the record

<Rownum,
<Row, Tablename> >

<Symbol, Other
Columns >

N.SYMBOL, N.NAME

T.SYMBOL,
T.HIGH , T.TIMESTAMP

We should also keep track of which table this row comes from

<Rownum,
<Row, Tablename> >

<Symbol,
<Other Columns , Table>>

N.SYMBOL, N.NAME

RIL	Reliance

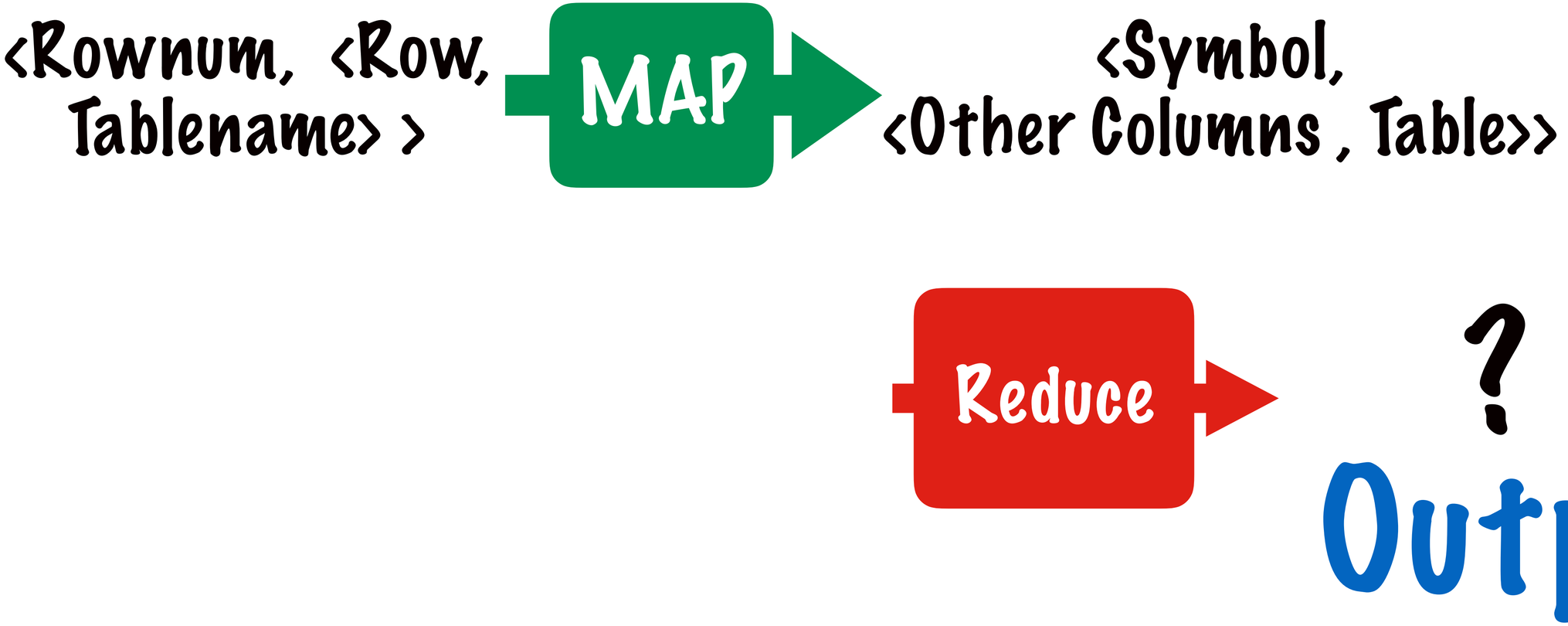
RIL	“Reliance” , N

T.SYMBOL, T.HIGH , T.TIMESTAMP

RIL	100	02DEC2014

RIL	“100, 02DEC2014” , T

```
SELECT N.SYMBOL, N.NAME, T.HIGH , T.TIMESTAMP
FROM  NAMES N
OUTER JOIN TRADES T
ON N.SYMBOL = T.SYMBOL
```



<Symbol,
<Other Columns , Table>>

Reduce

?

RIL	“102, 04MAY2015” , T

RIL	“Reliance” , N

RIL	“100, 02DEC2014” , T

In the Names table, Symbol is a Unique Key

<Symbol,
<Other Columns , Table>>

Reduce

?

RIL	“102, 04MAY2015” , T

RIL	“Reliance” , N

RIL	“100, 02DEC2014” , T

So, we know that for each Symbol, there will be at most 1 row from Names

<Symbol,
<Other Columns , Table>>

Reduce

?

RIL	“102, 04MAY2015” , T

RIL	“Reliance” , N

RIL	“100, 02DEC2014” , T

It would be more efficient if the first record the reducer sees is the one from Names

<Symbol,
<Other Columns , Table>>

Reduce

?

RIL	“Reliance” , N

RIL	“102, 04MAY2015” , T

RIL	“100, 02DEC2014” , T

It would be more efficient if the first record the reducer sees is the one from Names

<Symbol,
<Other Columns , Table>>

Reduce

?

RIL	“Reliance” , N

RIL	“102, 04MAY2015” , T

RIL	“100, 02DEC2014” , T

In this case we can just extract the record from the first value and iteratively join with the others

<Symbol,
<Other Columns , Table>>

Reduce

?

RIL	“Reliance” , N
RIL	“102, 04MAY2015” , T
RIL	“100, 02DEC2014” , T

RIL, Reliance	102, 04MAY2015
RIL, Reliance	100, 02DEC2014

<Symbol,
<Other Columns , Table>>

Reduce

<<Symbol, Name>,
<High, Timestamp>>

RIL	“Reliance” , N

RIL	“102, 04MAY2015” , T

RIL	“100, 02DEC2014” , T

RIL, Reliance	102, 04MAY2015

RIL, Reliance	100, 02DEC2014

<Symbol,
<Other Columns , Table>>

Reduce

<<Symbol, Name>,
<High, Timestamp>>

RIL	“Reliance” , N

RIL	“102, 04MAY2015” , T

RIL	“100, 02DEC2014” , T

We can use Secondary sort to make sure the records appear in this order

< <Symbol, Table>
Other columns>

Reduce

<<Symbol, Name>,
<High, Timestamp>>

RIL , N	“Reliance”

RIL , T	“102, 04MAY2015”

RIL, T	“100, 02DEC2014”

The key should include the table

< <Symbol, Table>
Other columns>

Reduce → <<Symbol, Name>,
<High, Timestamp>>

RIL , N	“Reliance”
RIL , T	“102, 04MAY2015”
RIL, T	“100, 02DEC2014”

We will sort using both parts of the key

< <Symbol, Table>
Other columns>

Reduce → <<Symbol, Name>, <High, Timestamp>>

RIL , N	“Reliance”
RIL , T	“102, 04MAY2015”
RIL, T	“100, 02DEC2014”

It would be easier to do so if the table name were an integer

< <Symbol, 0/1>
Other columns>

Reduce

<<Symbol, Name>,
<High, Timestamp>>

RIL , 0	"Reliance"
RIL , 1	"102, 04MAY2015"
RIL, 1	"100, 02DEC2014"

It would be easier to do so if the table name were an integer

< <Symbol, 0/1>
Other columns>

Reduce

<<Symbol, Name>,
<High, Timestamp>>

RIL	0	"Reliance"
RIL	1	"102, 04MAY2015"
RIL	1	"100, 02DEC2014"

We will merge using only the first part of the key

< <Symbol, 0/1>
Other columns>

Reduce

<<Symbol, Name>,
<High, Timestamp>>

RIL , 0	“Reliance”

RIL , 1	“102, 04MAY2015”

RIL, 1	“100, 02DEC2014”

Sorting adds efficiency only if one of the tables is joined on a Unique Key

< <Symbol, 0/1>
Other columns>

Reduce

<<Symbol, Name>,
<High, Timestamp>>

RIL , 0	"Reliance"

RIL , 1	"102, 04MAY2015"

RIL, 1	"100, 02DEC2014"

If there are multiple records from both tables, we have to generate one output row for every input pair

< <Symbol, 0/1>
Other columns>

Reduce

<<Symbol, Name>,
<High, Timestamp>>

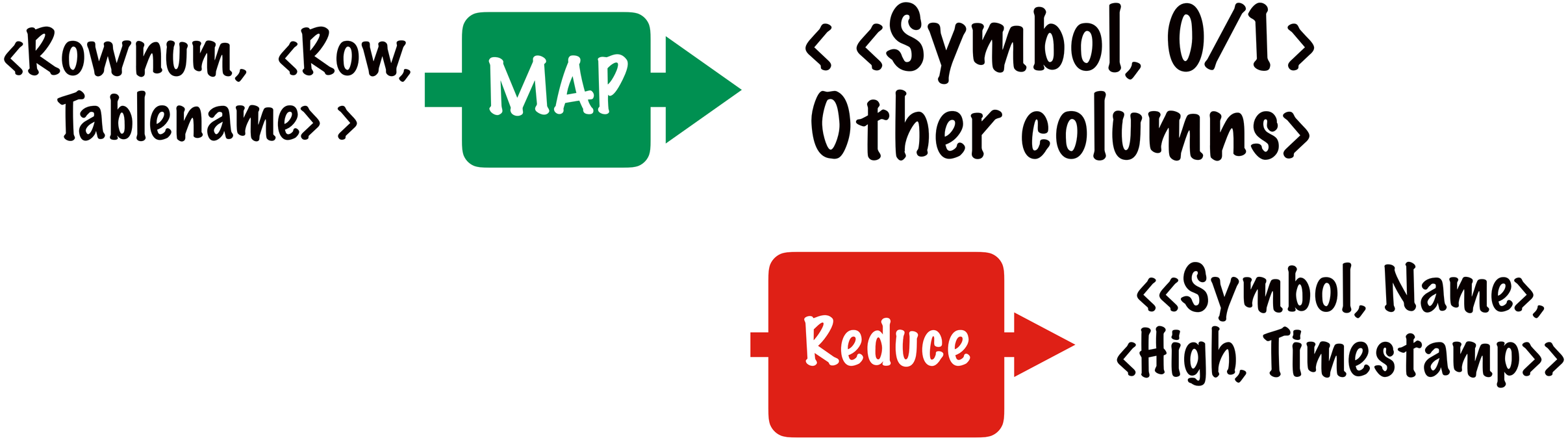
RIL , 0	“Reliance”

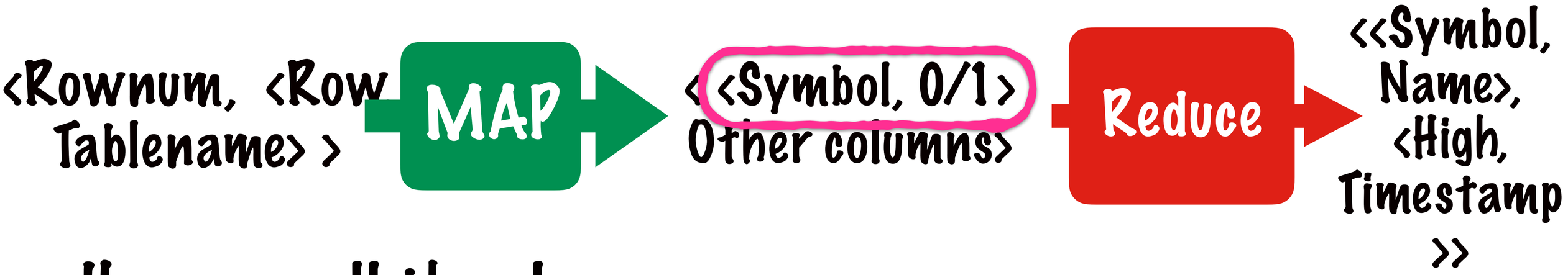
RIL , 1	“102, 04MAY2015”

RIL, 1	“100, 02DEC2014”

The strategy would be different again if it were a left or right join

```
SELECT N.SYMBOL, N.NAME, T.HIGH , T.TIMESTAMP
FROM  NAMES N
OUTER JOIN TRADES T
ON N.SYMBOL = T.SYMBOL
```

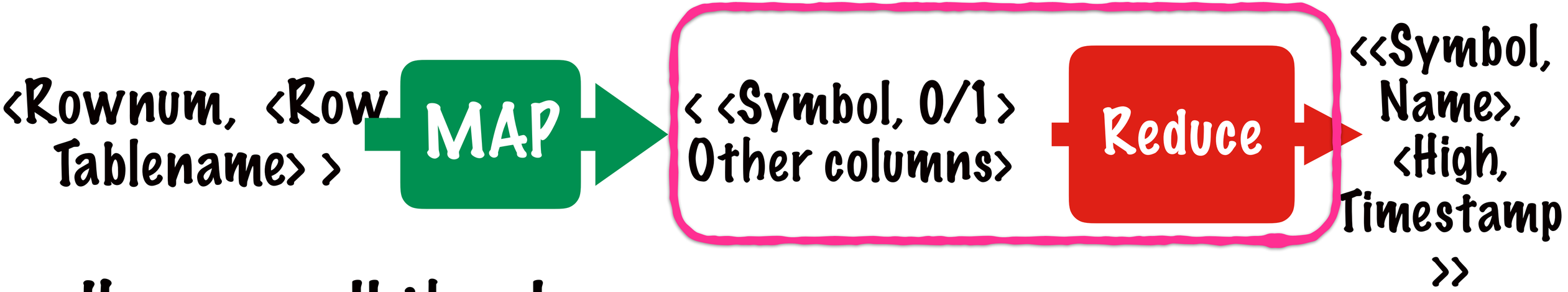




Here are all the classes we'll need

TextIntPair

This class will be a WritableComparable, whose compareTo() will be used for sorting



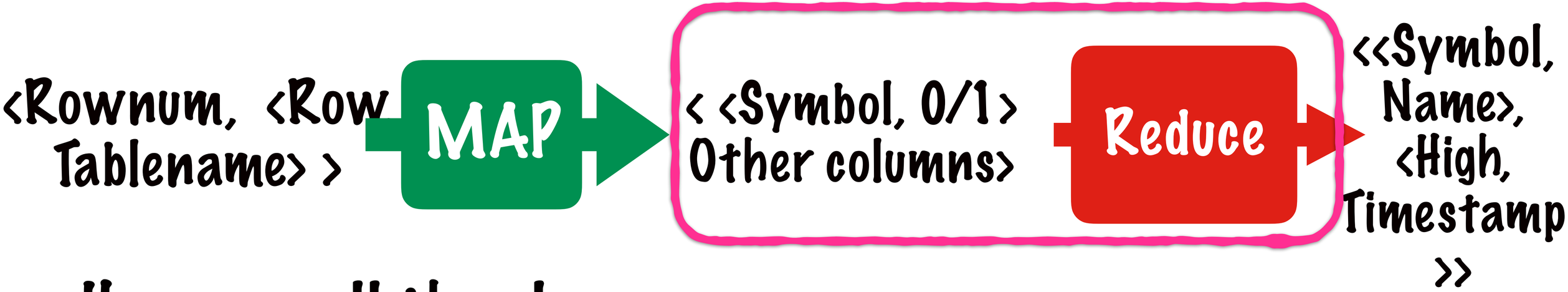
Here are all the classes we'll need

TextIntPair

FirstPartitioner

GroupComparator

To make sure partitioning and grouping is done only on 1st part of the key



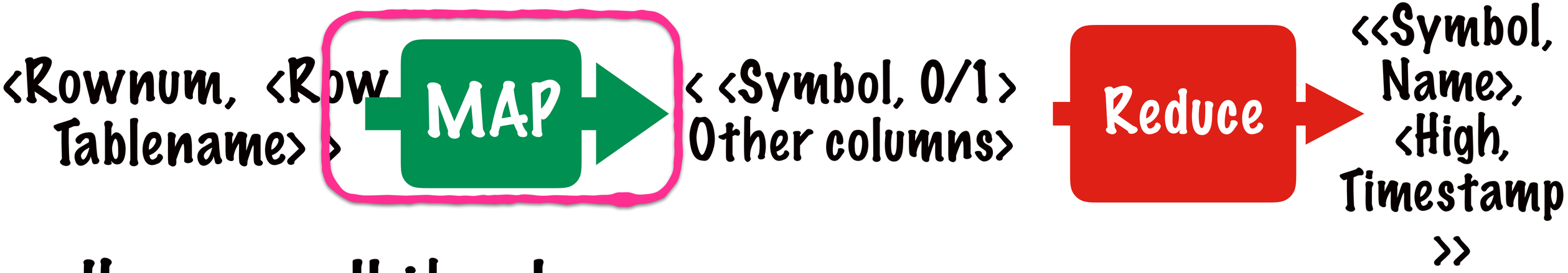
Here are all the classes we'll need

TextIntPair

FirstPartitioner

GroupComparator

These 3 classes will drive the secondary sort



Here are all the classes we'll need

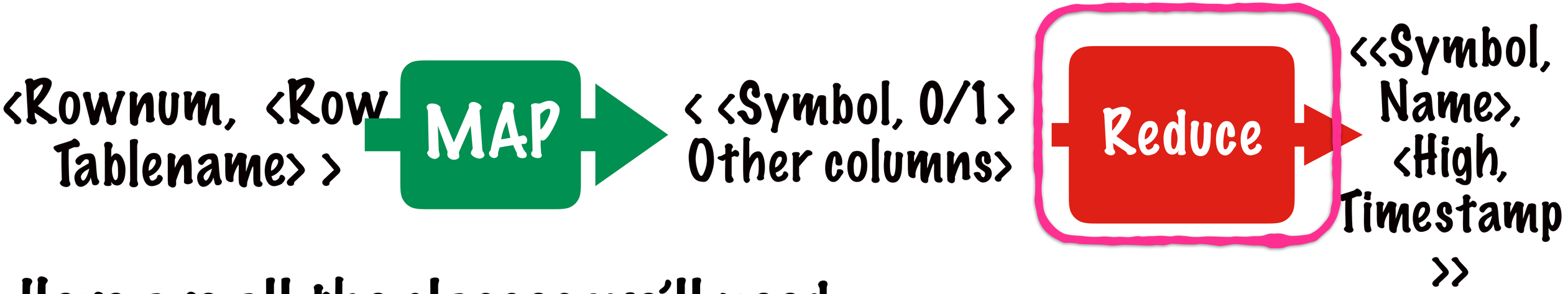
TextIntPair

FirstPartitioner

GroupComparator

NamesMapper

TradesMapper



Here are all the classes we'll need

TextIntPair

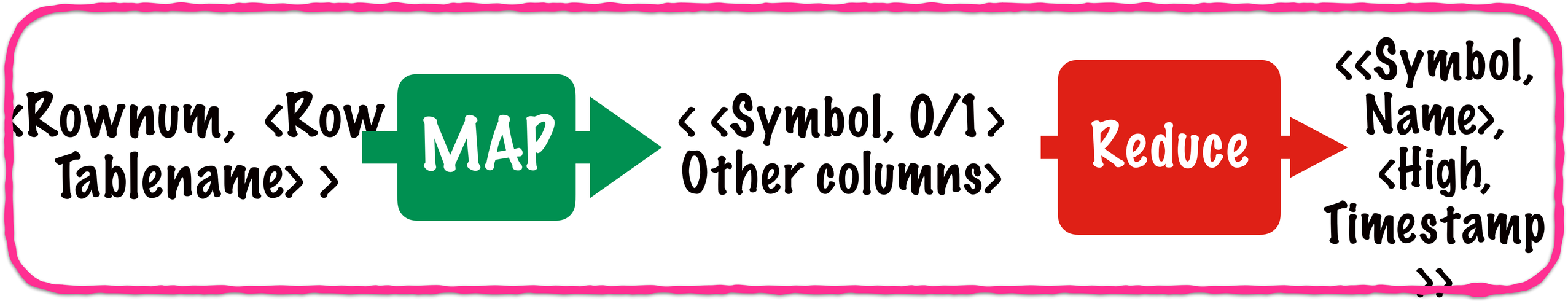
NamesMapper

TradesMapper

FirstPartitioner

GroupComparator

Reduce



Here are all the classes we'll need

TextIntPair

NamesMapper

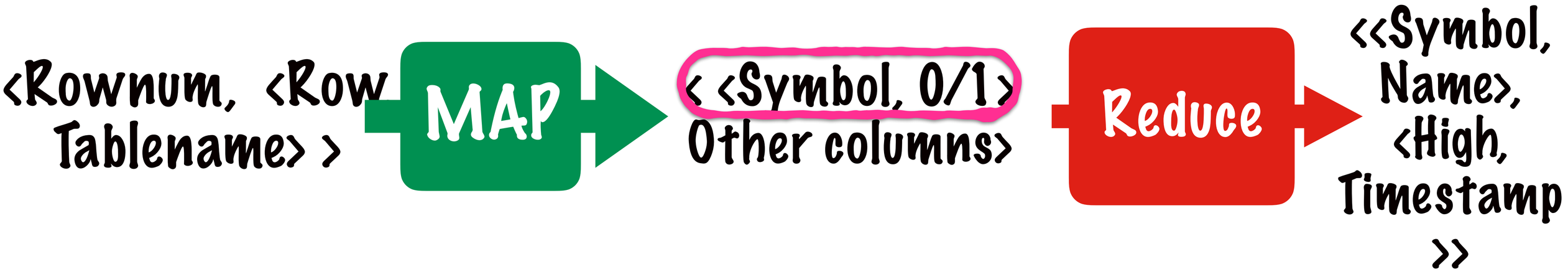
TradesMapper

FirstPartitioner

GroupComparator

Reduce

Join



TextIntPair

NamesMapper

TradesMapper

FirstPartitioner

GroupComparator

Reduce

Here are all the classes we'll need

Join

join.TextIntPair

```
public class TextIntPair implements
WritableComparable<TextIntPair> {

    private Text first;
    private IntWritable second;

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

    public Text getFirst() {
        return first;
    }
    public IntWritable getSecond() {
        return second;
    }


    public TextIntPair(){
        set(new Text(),new IntWritable());
    }

    public TextIntPair(String first, Integer second){
        set(new Text(first),new IntWritable(second));
    }
```

This is a class we've encountered before

join.TextIntPair

```
public class TextIntPair implements WritableComparable<TextIntPair> {

    private Text first;
    private IntWritable second;

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

    public Text getFirst() {
        return first;
    }
    public IntWritable getSecond() {
        return second;
    }


    public TextIntPair(){
        set(new Text(),new IntWritable());
    }

    public TextIntPair(String first, Integer second){
        set(new Text(first),new IntWritable(second));
    }
```

It has 2 members

join.TextIntPair

```
public class TextIntPair implements WritableComparable<TextIntPair> {

    private Text first;
    private IntWritable second;

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

    public Text getFirst() {
        return first;
    }
    public IntWritable getSecond() {
        return second;
    }


    public TextIntPair(){
        set(new Text(),new IntWritable());
    }

    public TextIntPair(String first, Integer second){
        set(new Text(first),new IntWritable(second));
    }
```

The first will represent the Key i.e. Symbol

join.TextIntPair

```
public class TextIntPair implements WritableComparable<TextIntPair> {

    private Text first;
    private IntWritable second;

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

    public Text getFirst() {
        return first;
    }
    public IntWritable getSecond() {
        return second;
    }


    public TextIntPair(){
        set(new Text(),new IntWritable());
    }

    public TextIntPair(String first, Integer second){
        set(new Text(first),new IntWritable(second));
    }
```

The second will represent the file/table

join.TextIntPair

```
@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 compareTo(TextIntPair tp){
    int cmp = first.compareTo(tp.getFirst());
    if (cmp!=0){
        return cmp;
    }
    return second.compareTo(tp.second);
}
```

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

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

@Override
```

The compareTo() method is important for the secondary sort

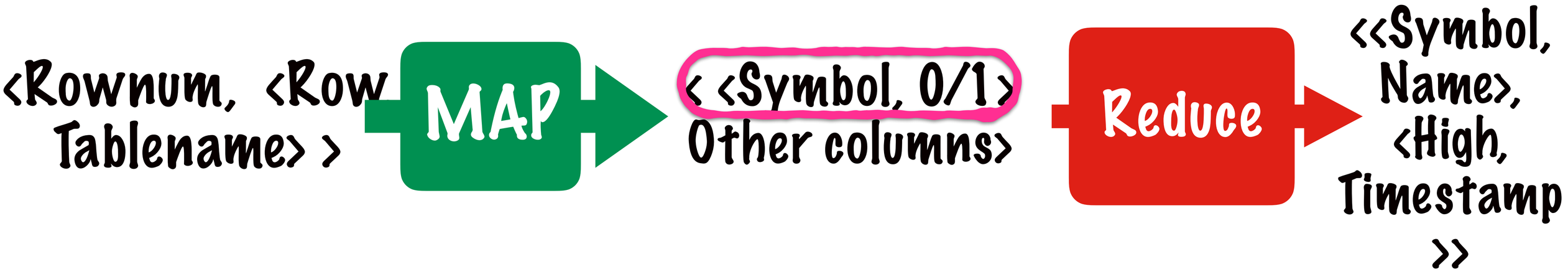
join.TextIntPair

```java
@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);
}
```

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

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

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

@Override
```

This method sorts using both the members



TextIntPair

NamesMapper

TradesMapper

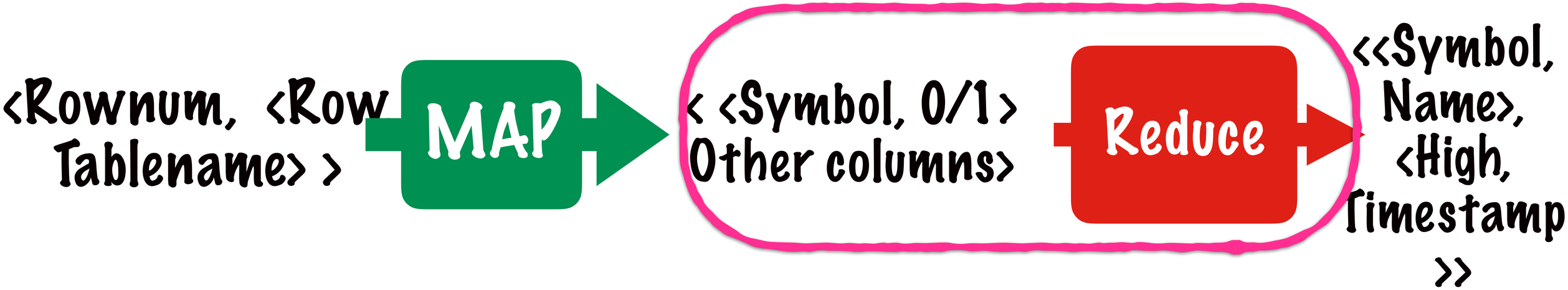
FirstPartitioner

GroupComparator

Reduce

Here are all the classes we'll need

Join



TextIntPair

NamesMapper

TradesMapper

FirstPartitioner

GroupComparator

Reduce

Here are all the classes we'll need

Join

join.FirstPartitioner

```
public class FirstPartitioner extends Partitioner<TextIntPair,Text>{
  @Override
  public int getPartition(TextIntPair key, Text value, int numReduceTasks){
     return (key.getFirst().hashCode()*Integer.MAX_VALUE) % numReduceTasks;
    }
}
```

This class will do the Partitioning for Secondary Sort

join.FirstPartitioner

```
public class FirstPartitioner extends Partitioner<TextIntPair,Text>{
  @Override
  public int getPartition(TextIntPair key, Text value, int numReduceTasks){
     return (key.getFirst().hashCode()*Integer.MAX_VALUE) % numReduceTasks;
    }
}
```

We need to partition only using the first part of the key

join.GroupComparator

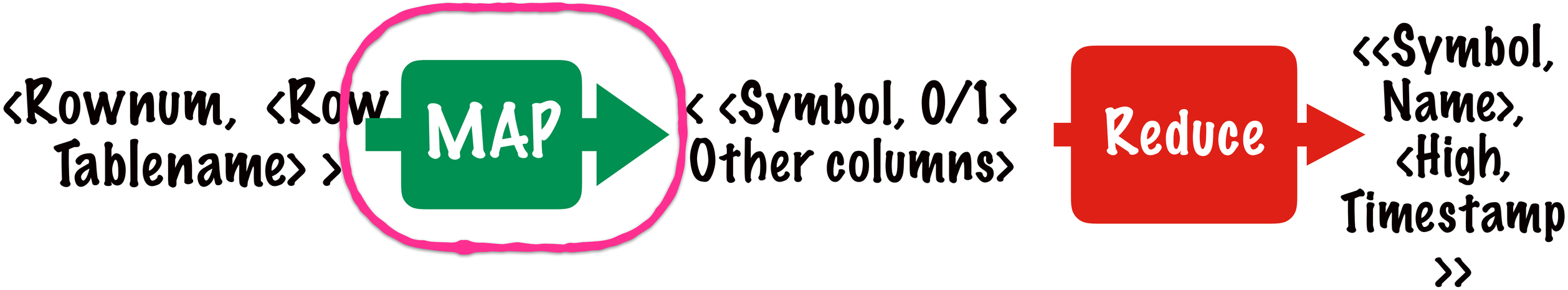
```
public class GroupComparator extends WritableComparator {
    protected GroupComparator(){
        super(secondarySort.TextIntPair.class, true);
    }

    @Override
    public int compare(WritableComparable w1, WritableComparable w2){

        secondarySort.TextIntPair p1 = (secondarySort.TextIntPair) w1;
        secondarySort.TextIntPair p2 = (TextIntPair) w2;

      return p1.getFirst().compareTo(p1.getFirst());
    }

}
```

Grouping needs to use only the first member of the pair

join.GroupComparator

```
public class GroupComparator extends WritableComparator {
    protected GroupComparator(){
        super(secondarySort.TextIntPair.class, true);
    }

    @Override
    public int compare(WritableComparable w1, WritableComparable w2){

        secondarySort.TextIntPair p1 = (secondarySort.TextIntPair) w1;
        secondarySort.TextIntPair p2 = (TextIntPair) w2;

      return p1.getFirst().compareTo(p1.getFirst());
    }

}
```

Grouping needs to use only the first member of the pair



TextIntPair

NamesMapper

TradesMapper

FirstPartitioner

GroupComparator

Reduce

Here are all the classes we'll need

Join

join.NamesMapper

```
public class NamesMapper extends Mapper<LongWritable,Text,TextIntPair,Text> {
    @Override
    public void map(LongWritable key, Text value, Context context) throws IOException, Interrupted

        String line = value.toString();
        String[] data = line.split(",");


        if(data[0].equals("SYMBOL")){
            return;
        }

        TextIntPair keyOut = new TextIntPair(new Text (data[0]), new IntWritable(0));

        String output = data[1];

        context.write(keyOut, new Text(output));
    }

}
```

join.NamesMapper

```
public class NamesMapper extends Mapper<LongWritable,Text,TextIntPair,Text> {
    @Override
    public void map(LongWritable key, Text value, Context context) throws IOException, Interrupted

        String line = value.toString();
        String[] data = line.split(",");
```

RIL	Reliance

```
){
            return;
        }

        TextIntPair keyOut = new TextIntPair(new Text (data[0]), new IntWritable(0));

        String output = data[1];

        context.write(keyOut, new Text(output));
    }

}
```

RIL , 0	“Reliance”

join.NamesMapper

RIL	Reliance

RIL , 0	“Reliance”

```
public class NamesMapper extends Mapper<LongWritable,Text,TextIntPair,Text> {
    @Override
    public void map(LongWritable key, Text value, Context context) throws IOException, InterruptedE

        String line = value.toString();
        String[] data = line.split(",");


        if(data[0].equals("SYMBOL")){
            return;
        }

        TextIntPair keyOut = new TextIntPair(new Text (data[0]), new IntWritable(0));

        String output = data[1];

        context.write(keyOut, new Text(output));
    }

}
```

join.NamesMapper

RIL	Reliance

RIL , 0	“Reliance”

```
public class NamesMapper extends Mapper<LongWritable,Text,TextIntPair,Text> {
    @Override
    public void map(LongWritable key, Text value, Context context) throws IOException, InterruptedException{

        String line = value.toString();
        String[] data = line.split(",");
        if(data[0].equals("SYMBOL")){
            return;
        }

        TextIntPair keyOut = new TextIntPair(new Text (data[0]), new IntWritable(0));

        String output = data[1];

        context.write(keyOut, new Text(output));
    }

}
```

Parse the line and check that it’s not a header row

join.NamesMapper

RIL	Reliance

RIL , 0	“Reliance”

```
public class NamesMapper extends Mapper<LongWritable,Text,TextIntPair,Text> {
    @Override
    public void map(LongWritable key, Text value, Context context) throws IOException, InterruptedException{

        String line = value.toString();
     String[] data = line.split(",");
     if(data[0].equals("SYMBOL")){
            return;
        }
```

```
TextIntPair keyOut = new TextIntPair(new Text (data[0]),
new IntWritable(0));
```

```
        String output = data[1];

        context.write(keyOut, new Text(output));
    }

}
```

join.NamesMapper

RIL	Reliance

RIL , 0	“Reliance”

```
public class NamesMapper extends Mapper<LongWritable,Text,TextIntPair,Text> {
    @Override
    public void map(LongWritable key, Text value, Context context) throws IOException, InterruptedException{

        String line = value.toString();
     String[] data = line.split(",");
     if(data[0].equals("SYMBOL")){
            return;
        }

        TextIntPair keyOut = new TextIntPair(new Text (data[0]), new IntWritable(0));

        String output = data[1];

        context.write(keyOut, new Text(output));
    }

}
```

join.TradesMapper

```
public class TradesMapper extends Mapper<LongWritable,Text,TextIntPair,Text> {
    @Override
    public void map(LongWritable key, Text value, Context context) throws IOException, InterruptedException{

        String line = value.toString();
        String[] data = line.split(",");


        if(data[0].equals("SYMBOL")){
            return;
        }

        TextIntPair keyOut = new TextIntPair(new Text (data[0]), new IntWritable(1));

        String output = data[4]+"\t"+data[10];
                        // High,       date

            context.write(keyOut, new Text(output));
        }

    }
```

join.TradesMapper

```
public class TradesMapper extends Mapper<LongWritable,Text,TextIntPair,Text> {
    @Override
    public void map(LongWritable key, Text value, Context context) throws IOException, InterruptedException{

        String line = value.toString();
        String[] data = line.split(",");


        if(data[0].equals("SYMBOL")){
```

RIL	100	02DEC2014

RIL, 1	“100, 02DEC2014”

```
        TextIntPair keyOut = new TextIntPair(new Text (data[0]), new IntWritable(1));

        String output = data[4]+"\t"+data[10];
                        // High,       date

            context.write(keyOut, new Text(output));
        }

    }
```

join.TradesMapper

RIL	100	02DEC2014	→	RIL, 1	“100, 02DEC2014”

```
public class TradesMapper extends Mapper<LongWritable,Text,TextIntPair,Text> {
    @Override
    public void map(LongWritable key, Text value, Context context) throws IOException, InterruptedException{

        String line = value.toString();
        String[] data = line.split(",");


        if(data[0].equals("SYMBOL")){
            return;
        }

        TextIntPair keyOut = new TextIntPair(new Text (data[0]), new IntWritable(1));

        String output = data[4]+"\t"+data[10];
                        // High,       date

            context.write(keyOut, new Text(output));
        }

    }
```

join.TradesMapper

RIL	100	02DEC2014

RIL, 1	“100, 02DEC2014”

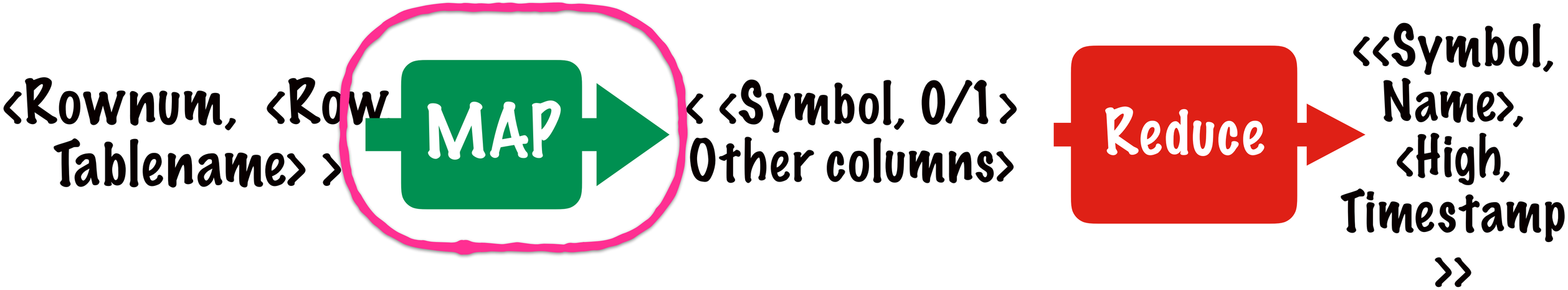
```
public class TradesMapper extends Mapper<LongWritable,Text,TextIntPair,Text> {
    @Override
    public void map(LongWritable key, Text value, Context context) throws IOException, InterruptedException{

        String line = value.toString();
        String[] data = line.split(",");


        if(data[0].equals("SYMBOL")){
            return;
        }

        TextIntPair keyOut = new TextIntPair(new Text(data[0]), new IntWritable(1));

        String output = data[4]+"\t"+data[10];
                        // High,       date

            context.write(keyOut, new Text(output));
        }

    }
```

join.TradesMapper

RIL	100	02DEC2014

RIL, 1	“100 02DEC2014”

```
public class TradesMapper extends Mapper<LongWritable,Text,TextIntPair,Text> {
    @Override
    public void map(LongWritable key, Text value, Context context) throws IOException, InterruptedException{

        String line = value.toString();
        String[] data = line.split(",");


        if(data[0].equals("SYMBOL")){
            return;
        }

        TextIntPair keyOut = new TextIntPair(new Text (data[0]), new IntWritable(1));

        String output = data[4]+"\t"+data[10];
                                // High,        Timestamp

            context.write(keyOut, new Text(output));
        }

    }
```



TextIntPair

NamesMapper

TradesMapper

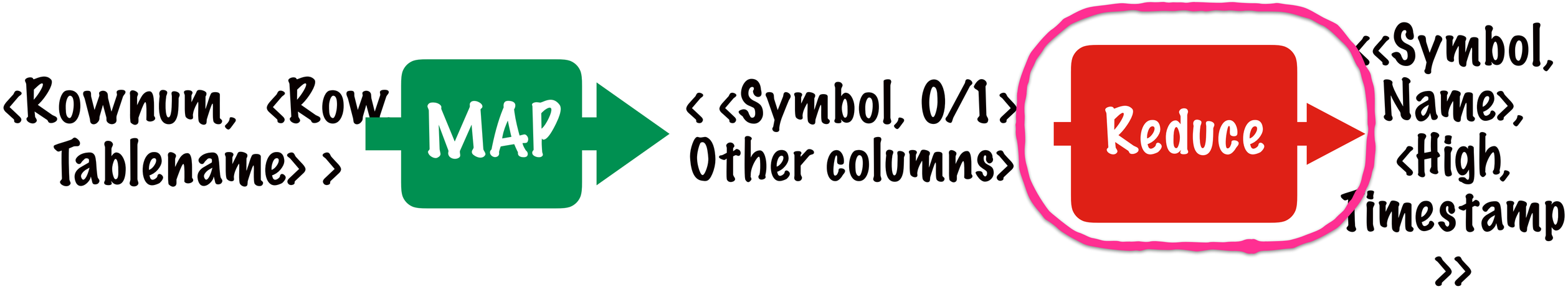
FirstPartitioner

GroupComparator

Reduce

Here are all the classes we'll need

Join



TextIntPair
NamesMapper
TradesMapper
FirstPartitioner
GroupComparator
Reduce

Here are all the classes we'll need

Join

join.Reduce

```
public class Reduce extends Reducer<TextIntPair, Text,Text,Text> {
    @Override
    public void reduce(TextIntPair key, Iterable<Text> values, Context context) throws IOException, In
        String keyOut = new String("");

        for (Text value:values)
        {
            if (key.getSecond().get()==0){
                keyOut = key.getFirst().toString()+"\t"+value.toString();
                continue;
            }

            context.write(new Text(keyOut) , value);
        }


    }
}
```

join.Reduce

```
public class Reduce extends Reducer<TextIntPair, Text,Text,Text> {
    @Override
                                                rable<Text> values, Context context) throws IOException, In

        for (Text value:values)

                                                tring()+"\t"+value.toString();
                continue;
            }

                                                , value);

    }
}
```

RIL , 0	"Reliance"

RIL , 1	"102 04MAY2015"

RIL, 1	"100 02DEC2014"

RIL Reliance	102 04MAY2015

RIL Reliance	100 02DEC2014

join.Reduce

RIL , 0	“Reliance”
RIL , 1	“102 04MAY2015”
RIL, 1	“100 02DEC2014”

RIL Reliance	102 04MAY2015
RIL Reliance	100 02DEC2014

```
public class Reduce extends Reducer<TextIntPair, Text,Text,Text> {
    @Override
    public void reduce(TextIntPair key, Iterable<Text> values, Context context) throws IOException, Inter
        String keyOut = new String("");

        for (Text value:values)
        {
            if (key.getSecond().get()==0){
                keyOut = key.getFirst().toString()+"\t"+value.toString();
                continue;
            }

            context.write(new Text(keyOut) , value);
        }


    }
}
```

join.Reduce

RIL , 0	"Reliance"
RIL , 1	"102 04MAY2015"
RIL, 1	"100 02DEC2014"

RIL Reliance	102 04MAY2015
RIL Reliance	100 02DEC2014

```
public class Reduce extends Reducer<TextIntPair, Text,Text,Text> {
    @Override
    public void reduce(TextIntPair key, Iterable<Text> values, Context context) throws IOException, Inter
        String keyOut = new String("");

        for (Text value:values)
        {
        if (key.getSecond().get()==0){
        keyOut = key.getFirst().toString()+"\t"+value.toString();
        continue;
                }

            context.write(new Text(keyOut) , value);
        }


    }
}
```

join.Reduce

RIL , 0	"Reliance"
RIL , 1	"102 04MAY2015"
RIL, 1	"100 02DEC2014"

RIL Reliance	102 04MAY2015
RIL Reliance	100 02DEC2014

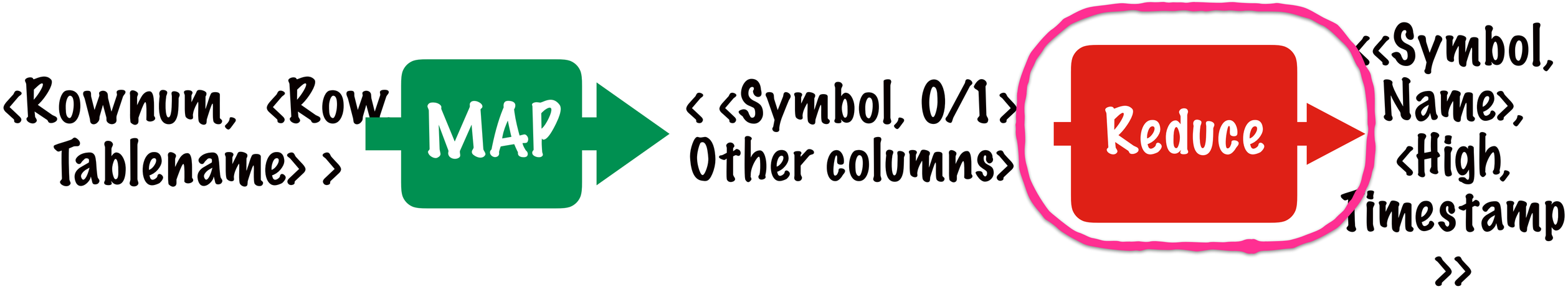
```
public class Reduce extends Reducer<TextIntPair, Text,Text,Text> {
    @Override
    public void reduce(TextIntPair key, Iterable<Text> values, Context context) throws IOException, Inter
        String keyOut = new String("");

        for (Text value:values)
        {
        if (key.getSecond().get()==0){
        keyOut = key.getFirst().toString()+"\t"+value.toString();
        continue;
                }

            context.write(new Text(keyOut) , value);
        }


    }
}
```

join.Reduce

RIL , 0	“Reliance”
RIL , 1	“102 04MAY2015”
RIL, 1	“100 02DEC2014”

RIL Reliance	102 04MAY2015
RIL Reliance	100 02DEC2014

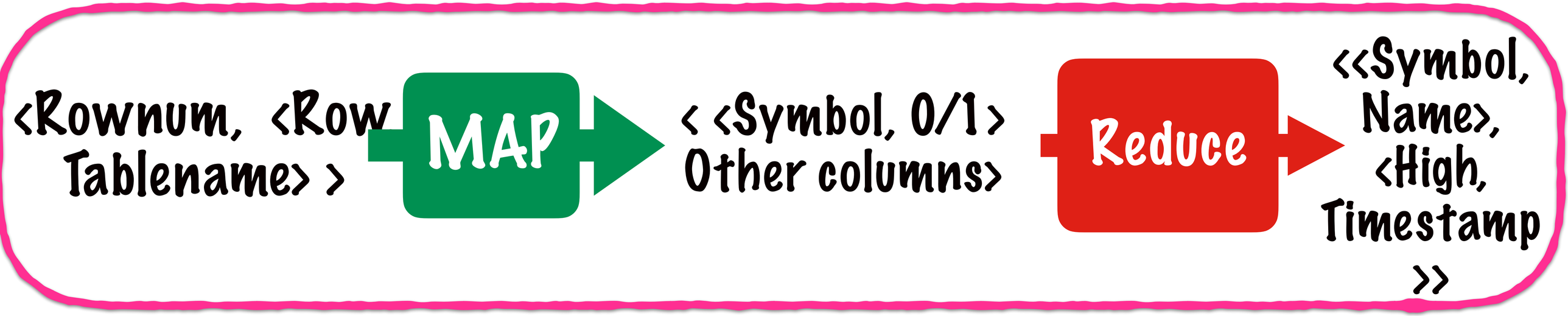
```
public class Reduce extends Reducer<TextIntPair, Text,Text,Text> {
    @Override
    public void reduce(TextIntPair key, Iterable<Text> values, Context context) throws IOException, Inter
        String keyOut = new String("");

        for (Text value:values)
        {
            if (key.getSecond().get()==0){
                keyOut = key.getFirst().toString()+"\t"+value.toString();
                continue;
            }

                context.write(new Text(keyOut) , value);
        }


    }
}
```



TextIntPair
NamesMapper
TradesMapper
FirstPartitioner
GroupComparator
Reduce

Here are all the classes we'll need

Join



TextIntPair
NamesMapper
TradesMapper
FirstPartitioner
GroupComparator
Reduce

Here are all the classes we'll need

Join

join.Join

3 arguments

Names file
Trades file
Output directory

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

      if(args.length !=3){
          System.err.println("Invalid Command");
          System.err.println("Usage: <input path1> <input path2> <output path>");
          return -1;
      }

        Configuration conf = new Configuration();

        Job job = new Job(conf, "Join");

        job.setJarByClass(getClass());
```

```java
Path namesPath = new Path(args[0]);
Path tradesPath = new Path(args[1]);

FileOutputFormat.setOutputPath(job, new Path(args[2]));
```

```java
        MultipleInputs.addInputPath(job, namesPath, TextInputFormat.class, NamesMapper.class);
        MultipleInputs.addInputPath(job, tradesPath, TextInputFormat.class, TradesMapper.class);
        job.setReducerClass(Reduce.class);

        job.setMapOutputKeyClass(TextIntPair.class);
        job.setMapOutputValueClass(Text.class);
        job.setPartitionerClass(FirstPartitioner.class);
        job.setGroupingComparatorClass(GroupComparator.class);

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

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

join.Join

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

        if(args.length != 3){
            System.err.println("Invalid Command");
            System.err.println("Usage: <input path1> <input path2> <output path>");
            return -1;
        }

        Configuration conf = new Configuration();

        Job job = new Job(conf, "Join");

        job.setJarByClass(getClass());


        Path namesPath = new Path(args[0]);
        Path tradesPath = new Path(args[1]);

        FileOutputFormat.setOutputPath(job, new Path(args[2]));
```

Use MultipleInputs to point each file to it's relevant Mapper class

```
MultipleInputs.addInputPath(job, namesPath, TextInputFormat.class,
NamesMapper.class);

MultipleInputs.addInputPath(job, tradesPath, TextInputFormat.class,
TradesMapper.class);
```

```
        job.setReducerClass(Reduce.class);

        job.setMapOutputKeyClass(TextIntPair.class);
        job.setMapOutputValueClass(Text.class);
        job.setPartitionerClass(FirstPartitioner.class);
        job.setGroupingComparatorClass(GroupComparator.class);

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

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

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

join.Join

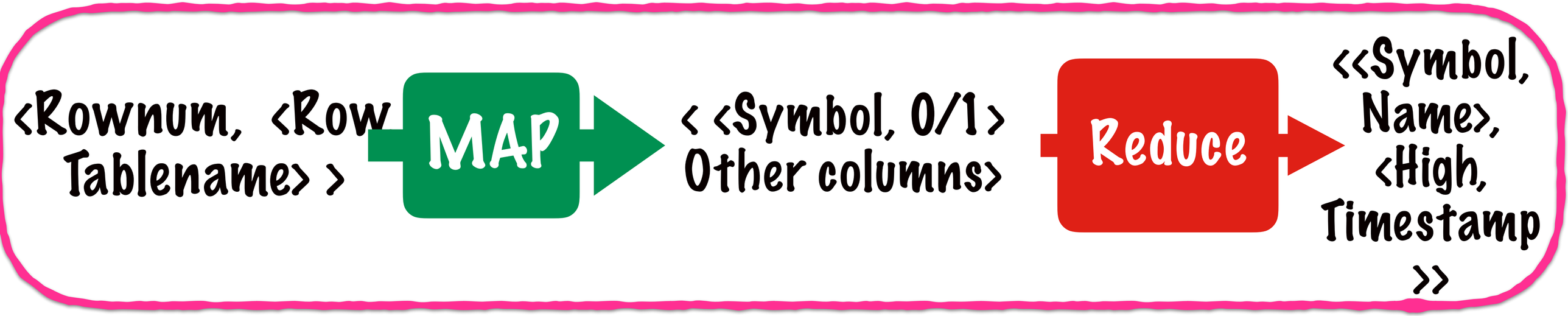
```java
public class Join extends Configured implements Tool{
    @Override
    public int run(String[] args) throws Exception{

        if(args.length !=3){
            System.err.println("Invalid Command");
            System.err.println("Usage: <input path1> <input path2> <output path>");
            return -1;
        }

        Configuration conf = new Configuration();

        Job job = new Job(conf, "Join");

        job.setJarByClass(getClass());


        Path namesPath = new Path(args[0]);
        Path tradesPath = new Path(args[1]);

        FileOutputFormat.setOutputPath(job, new Path(args[2]));

        MultipleInputs.addInputPath(job, namesPath, TextInputFormat.class, NamesMapper.class);
        MultipleInputs.addInputPath(job, tradesPath, TextInputFormat.class, TradesMapper.class);
        job.setReducerClass(Reduce.class);

        job.setMapOutputKeyClass(TextIntPair.class);
        job.setMapOutputValueClass(Text.class);
        job.setPartitionerClass(FirstPartitioner.class);
        job.setGroupingComparatorClass(GroupComparator.class);

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

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

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

Set the Partitioner and Grouping Comparator classes

join.Join

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

        if(args.length !=3){
            System.err.println("Invalid Command");
            System.err.println("Usage: <input path1> <input path2> <output path>");
            return -1;
        }

        Configuration conf = new Configuration();

        Job job = new Job(conf, "Join");

        job.setJarByClass(getClass());


        Path namesPath = new Path(args[0]);
        Path tradesPath = new Path(args[1]);

        FileOutputFormat.setOutputPath(job, new Path(args[2]));

        MultipleInputs.addInputPath(job, namesPath, TextInputFormat.class, NamesMapper.class);
        MultipleInputs.addInputPath(job, tradesPath, TextInputFormat.class, TradesMapper.class);
        job.setReducerClass(Reduce.class);

        job.setMapOutputKeyClass(TextIntPair.class);
        job.setMapOutputValueClass(Text.class);
        job.setPartitionerClass(FirstPartitioner.class);
        job.setGroupingComparatorClass(GroupComparator.class);

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

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

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

Here we'll set up a job which uses all the other classes



TextIntPair
NamesMapper
TradesMapper
FirstPartitioner
GroupComparator
Reduce

Here are all the classes we'll need

Join

Select MR

Where MR

Group by MR

Having MR

Join MR

As you can see, Joins in MapReduce are pretty complex to write

Select MR

Where MR

Group by MR

Having MR

Join MR

They're still worth it though, because of the parallelization

Select MR

Where MR

Group by MR

Having MR

Join MR

This is why there are a lot of **Higher level frameworks** on top of Hadoop which make it easier to express joins

Pig

Hive

are 2 examples of such frameworks