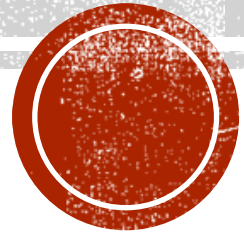
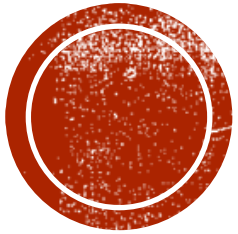
BIC DATA ALGORITHMS



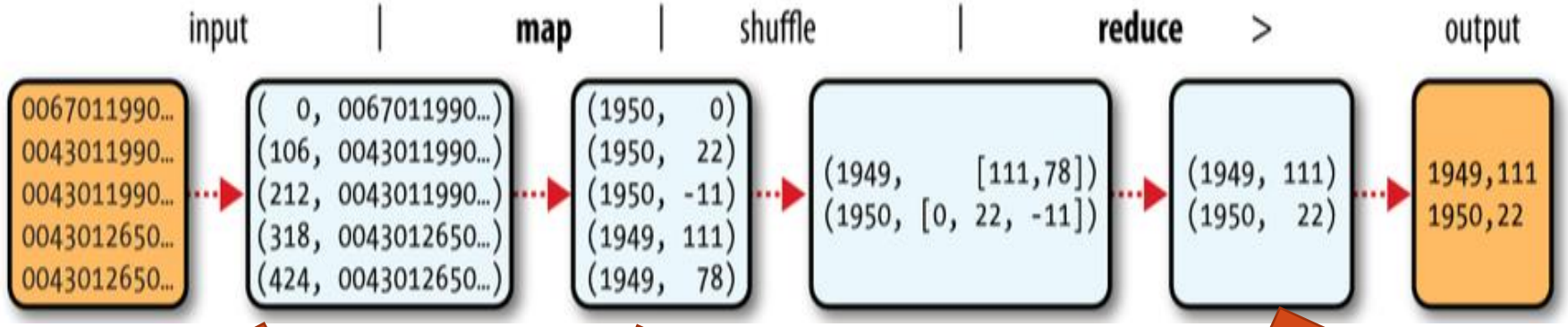
BIG DATA ALGORITHMS - AGENDA

- Secondary Sort – Will be covered in Lecture
- Top *N* – Covered in Homework
- Recommendation Engines – Posted on Canvas
  - "People You May Know" [Optional]



WHY DO WE NEED SECONDARY SORTING?

REMEMBER MAX TEMPERATURE PROBLEM?



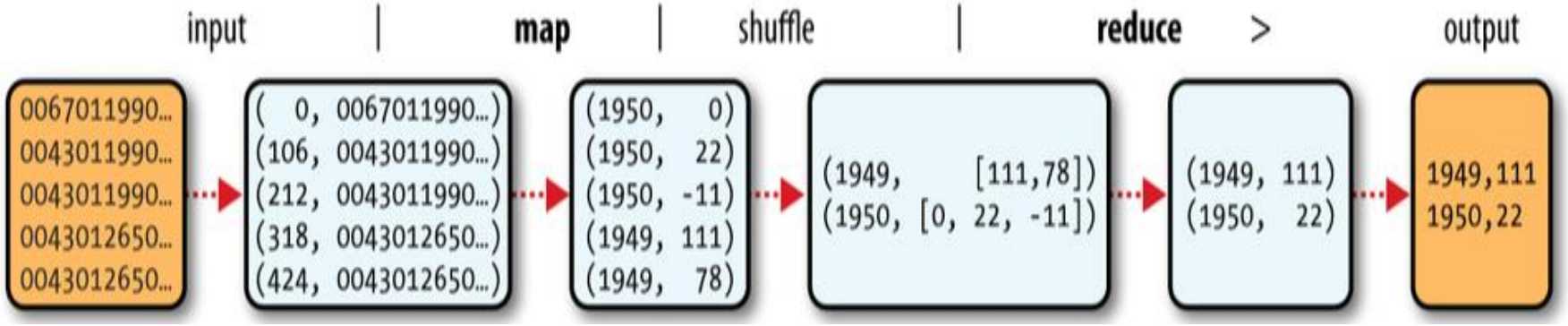
These keys (0, 106, 212, etc.) represent the offset of the input split inside the file.

These keys (1949, 1950, etc.) represent the year where the temperature was taken.

The reducer has to sort all the values for a given year.

- **Wouldn't be more efficient to send only the maximum temperature from the mapper to the reducer instead of sorting all values at the reducer?**

DO YOU REMEMBER?



- Sorting and merging at the reducer's phase consumes **LONG TIME** and may lead to **Memory bottleneck**
- **What if we can avoid sorting at the reducer?**
- **Why do you have to sort in the reducer if Hadoop Framework sorts and shuffles already for you?!**
- **Can't we just use the Hadoop Sorting and Shuffling to get the ouput data in the format we are looking for?**

WHY DO WE NEED SECONDARY SORTING IN MAPREDUCE? SORTING PROBLEM – SORTING BY VALUES

- The MapReduce framework **sorts the records by key before they reach the reducers**.
  - For any particular key, however, **the values are not sorted**.
  - The order that the values appear is not even stable from one run to the next, since they come from different map tasks, which may finish at different times from run to run.
- Generally speaking, most **MapReduce programs are written so as not to depend on the order that the values appear** to the reduce function.
  - However, it's possible to impose an order on the values by sorting and grouping the keys in a particular way.
- So, what is the solution?
  - **Make the values of interest members of the key!**
    - **How?!!**
      - **Let's take a look!**

SECONDARY SORTING SOLUTIONS FOR MAPREDUCE

This technique will enable us to sort the values (in ascending or descending order) passed to each reducer.

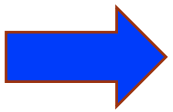
Use the MapReduce framework

- Creating a composite key by adding a part of, or the entire value to, the natural key to achieve the sorting

SECONDARY SORTING PROBLEM

- The problem is related to sorting values associated with a key other than the natural key alone
- Example: **let's calculate the max temperature for every month in the year.**
- Naturally, the key would be composite of year-month and our reducer would find the maximum temperature among all the values it receives:

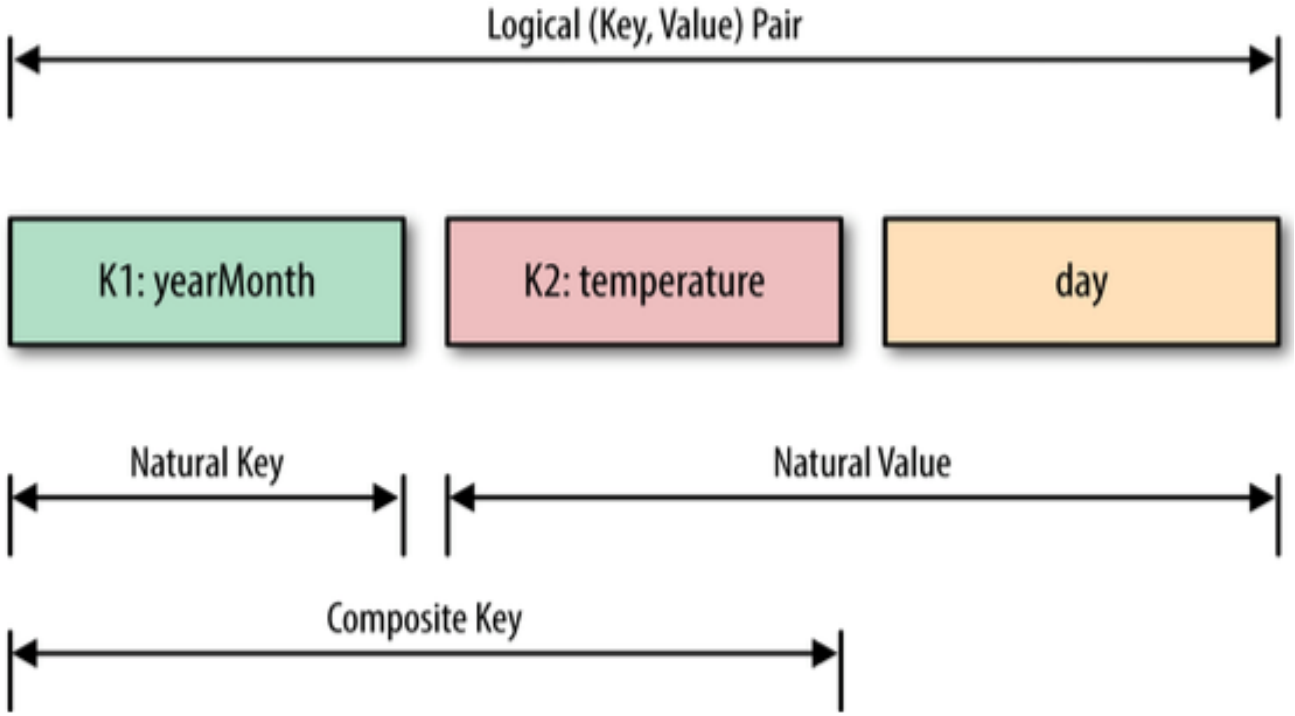
2012, 01, 01, 5
2012, 01, 02, 45
2012, 01, 03, 35
2012, 01, 04, 10
...
2001, 11, 01, 46
2001, 11, 02, 47
2001, 11, 03, 48
2001, 11, 04, 40
...



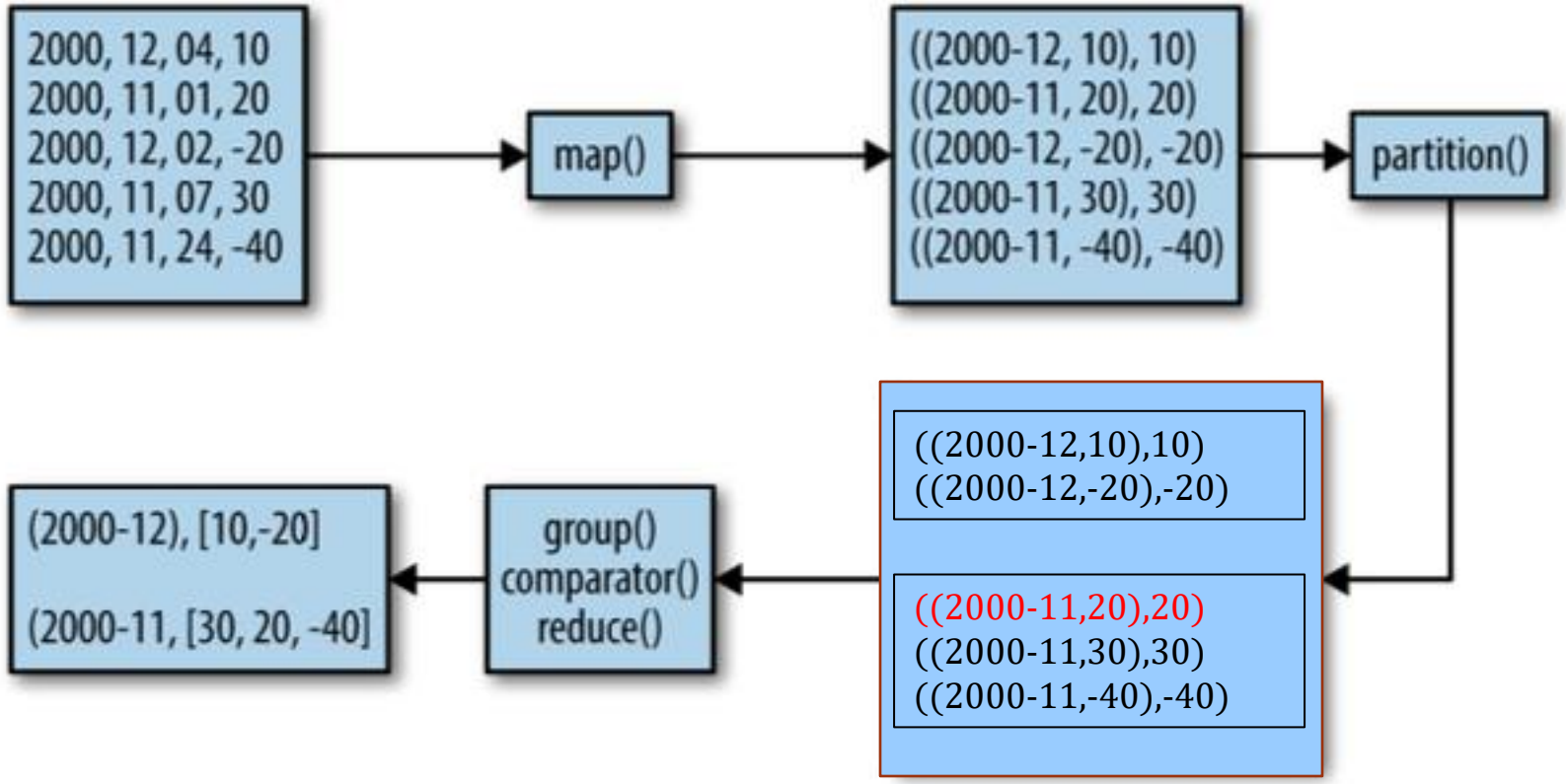
2012-01: 5, 10, 35, 45, ...
2001-11: 40, 46, 47, 48, ...

SECONDARY SORTING – HOW TO? FORM A COMPOSITE KEY

- Example: Find the temperature for every year-month with the values sorted in descending order

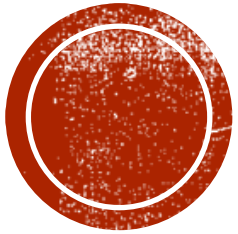


DATA FLOW OF THE SECONDARY SORTING USING MAPREDUCE



USING MAPREDUCE FRAMEWORK TECHNIQUE

1. Use the *Value-to-Key* Conversion design pattern
   - Form a composite intermediate key, (K, V_1) where K is a natural key and V_1 is the secondary key
2. Let the MapReduce execution framework do the sorting (rather than sorting in the memory with your code)
3. Preserve state across multiple key-value pairs to handle processing
   - Overwrite the partitioner class to partition based on natural key K
   - Overwrite the grouping comparator to group the data base on natural key K



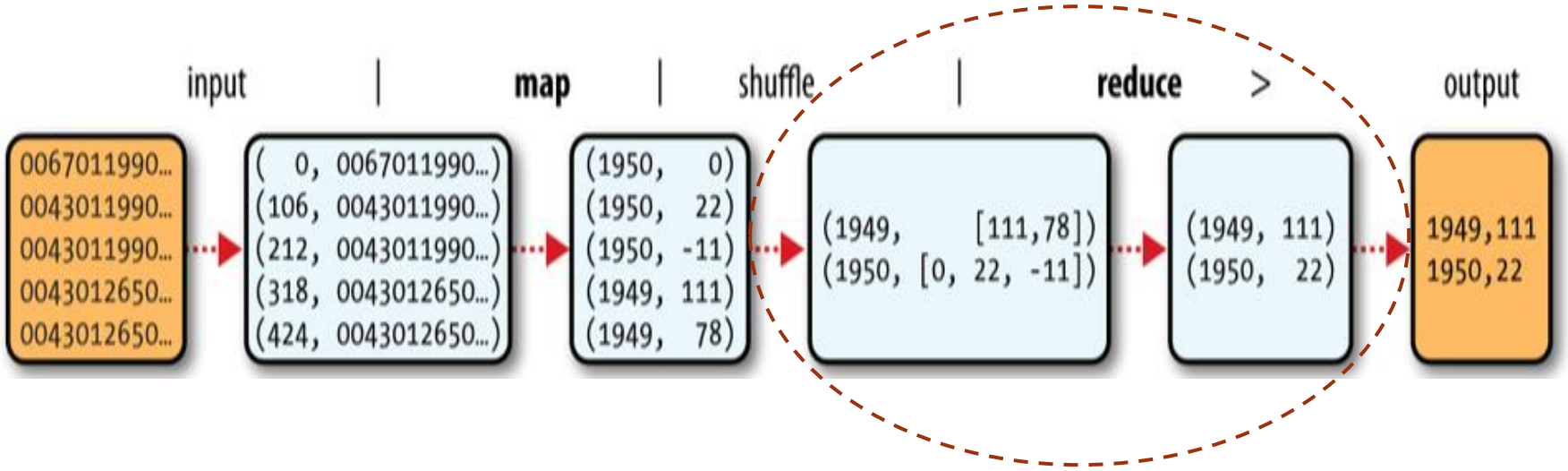
DO YOU REMEMBER MAXTEMPERATURE PROGRAM?

SORTING BY VALUES – MAX TEMPERATURE

- Consider the MapReduce program for calculating the maximum temperature for each year.
  - If we arranged for the values (temperatures) to be sorted in descending order, we wouldn't have to iterate through them to find the maximum.
  - We could take the first for each year and ignore the rest.
  - This approach is not the most efficient way to solve this particular problem but it illustrates how secondary sort works in general

FIND MAXIMUM TEMP FOR EACH YEAR

- The simple solution
  - Reducer iterates through all temps for each year to find the maximum



- A better way (- Secondary Sort Problem -)
  - Organize data such that the highest temperature is the first on the list

MAP STAGE

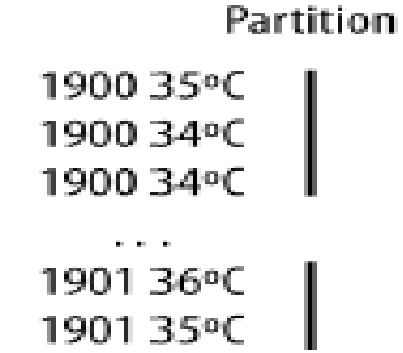
- Change keys to be a composite of year and temperature.
- Change the sort order for keys to be by years (ascending) and then temperature (descending)

```
1900 35°C
1900 34°C
1900 34°C
...
1901 36°C
1901 35°C
```

- If all we did was changing the key, then this wouldn't help since now records for the same year would not in general go to the same reducer since they have different keys.
  - For example, (1900, 35) and (1900, 34) could go to different reducers.

SHUFFLE STAGE

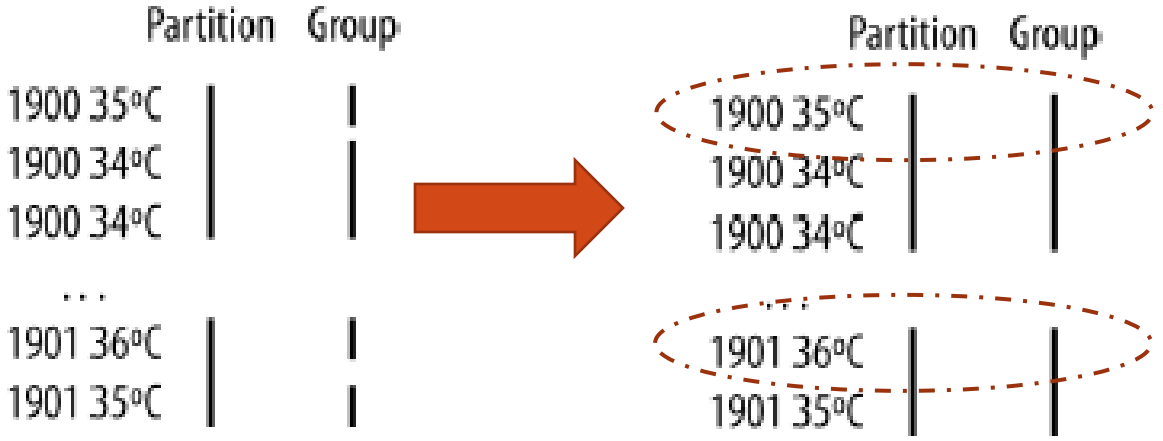
- Change how the keys are partitioned to be based on the year only so that all data belong to one year would go to the same reducer
- By setting a partitioner to partition by the year part of the key, we can guarantee that records for the same year go to the same reducer.



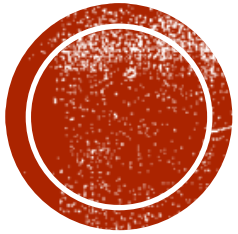
- This still isn't enough to achieve our goal.
  - A partitioner ensures only that one reducer receives all the records for a year, it doesn't change the fact that the reducer groups by key within the partition.

MERGE STAGE

- Change how keys are grouped to be based on the year only



- We will see all the records for the same year in one reduce group and since they are sorted by temperature in descending order, the first is the maximum temperature



CODE WALKTHROUGH IS YOURS!

REVISE MAPPING OUTPUT KEY

```
@Override
protected void map(LongWritable key, Text value,
    Context context) throws IOException, InterruptedException {

  parser.parse(value);
  if (parser.isValidTemperature()) {
    context.write(new IntPair(parser.getYearInt(),
        parser.getAirTemperature()), NullWritable.get());
  }
}
```

- We create a key representing the year and temperature using IntPair Writable implementation.
- We don't need to carry any information in the value field.

REVISE KEY COMPARISON - SORT MAP OUTPUT STAGE

```
public static class KeyComparator extends WritableComparator {
  protected KeyComparator() {
    super(IntPair.class, true);
  }

  @Override
  public int compare(WritableComparable w1, WritableComparable w2) {
    IntPair ip1 = (IntPair) w1;
    IntPair ip2 = (IntPair) w2;
    int cmp = IntPair.compare(ip1.getFirst(), ip2.getFirst());
    if (cmp != 0) {
      return cmp;
    }
    return -IntPair.compare(ip1.getSecond(), ip2.getSecond()); //reverse
  }
}
```

- To sort keys by year (ascending) and temperature (descending), we use a custom key comparator that extracts the fields and performs the appropriate comparisons.

REVISE PARTITIONING – SHUFFLE STAGE

```
public static class FirstPartitioner
    extends Partitioner<IntPair, NullWritable> {

  @Override
  public int getPartition(IntPair key, NullWritable value, int numPartitions) {
    // multiply by 127 to perform some mixing
    return Math.abs(key.getFirst() * 127) % numPartitions;
  }
}
```

- We set the partitioner to partition by the first field of the key (the year), using a custom partitioner (FirstPartitioner).

REVISE GROUPING – MERGE STAGE

```
public static class GroupComparator extends WritableComparator {
  protected GroupComparator() {
    super(IntPair.class, true);
  }
  @Override
  public int compare(WritableComparable w1, WritableComparable w2) {
    IntPair ip1 = (IntPair) w1;
    IntPair ip2 = (IntPair) w2;
    return IntPair.compare(ip1.getFirst(), ip2.getFirst());
  }
}
```

- To group keys by year, we use a custom comparator to extract the first field of the key for comparison.

SIMPLE REDUCE METHOD

```
static class MaxTemperatureReducer
    extends Reducer<IntPair, NullWritable, IntPair, NullWritable> {

  @Override
  protected void reduce(IntPair key, Iterable<NullWritable> values,
      Context context) throws IOException, InterruptedException {

    context.write(key, NullWritable.get());
  }
}
```

- The reducer emits the first key, which due to the secondary sorting, it's an IntPair for the year and it's maximum temperature.
- If your application needs to access all the sorted values, you will have to populate the value fields since in the reducer, you can retrieve only the first key.

SETTING UP IN THE DRIVER

```
@Override
public int run(String[] args) throws Exception {
  Job job = JobBuilder.parseInputAndOutput(this, getConf(), args);
  if (job == null) {
    return -1;
  }

  job.setMapperClass(MaxTemperatureMapper.class);
  job.setPartitionerClass(FirstPartitioner.class);
  job.setSortComparatorClass(KeyComparator.class);
  job.setGroupingComparatorClass(GroupComparator.class);
  job.setReducerClass(MaxTemperatureReducer.class);
  job.setOutputKeyClass(IntPair.class);
  job.setOutputValueClass(NullWritable.class);

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

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