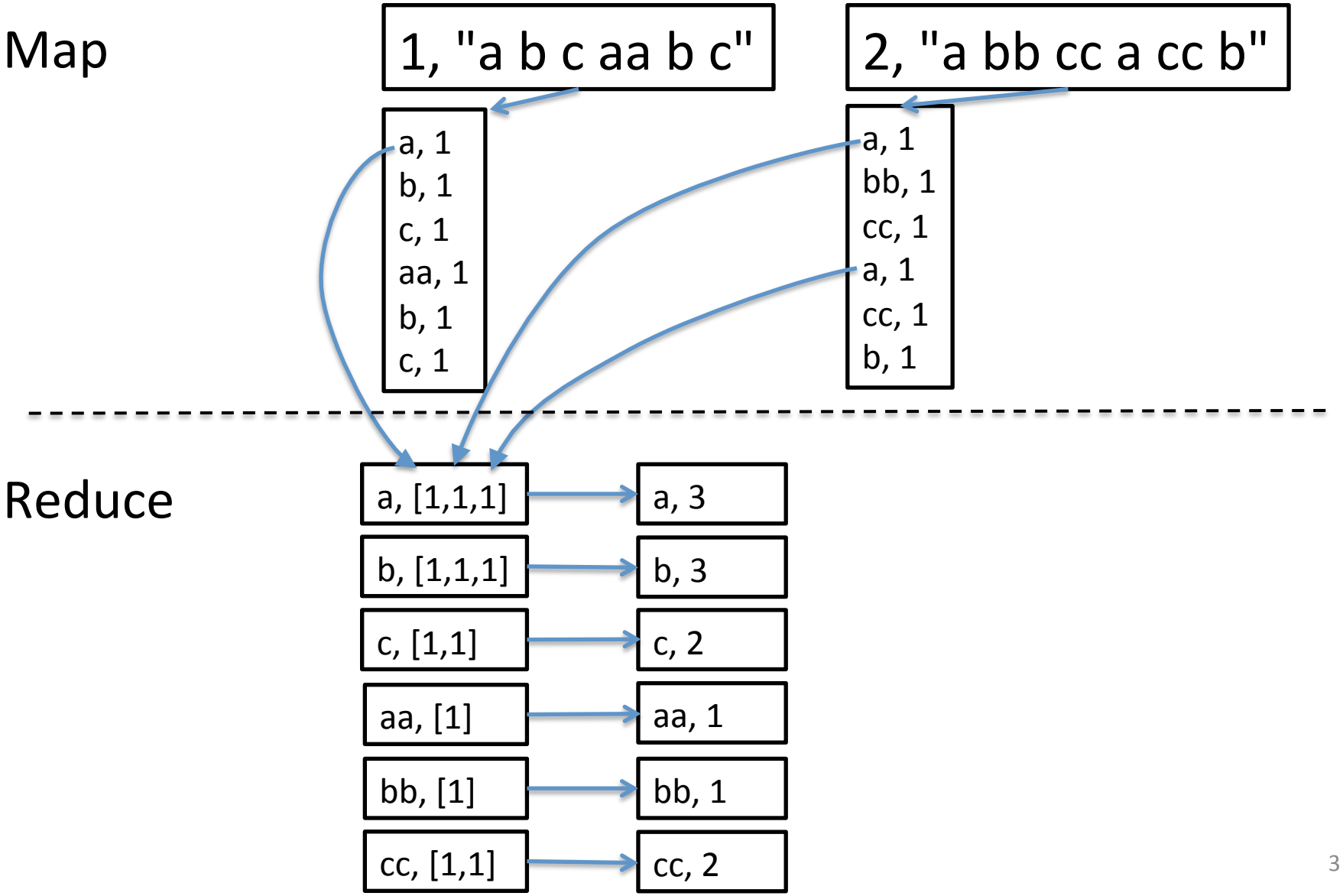
Map-Reduce debrief

Vincent Leroy

Map and Reduce

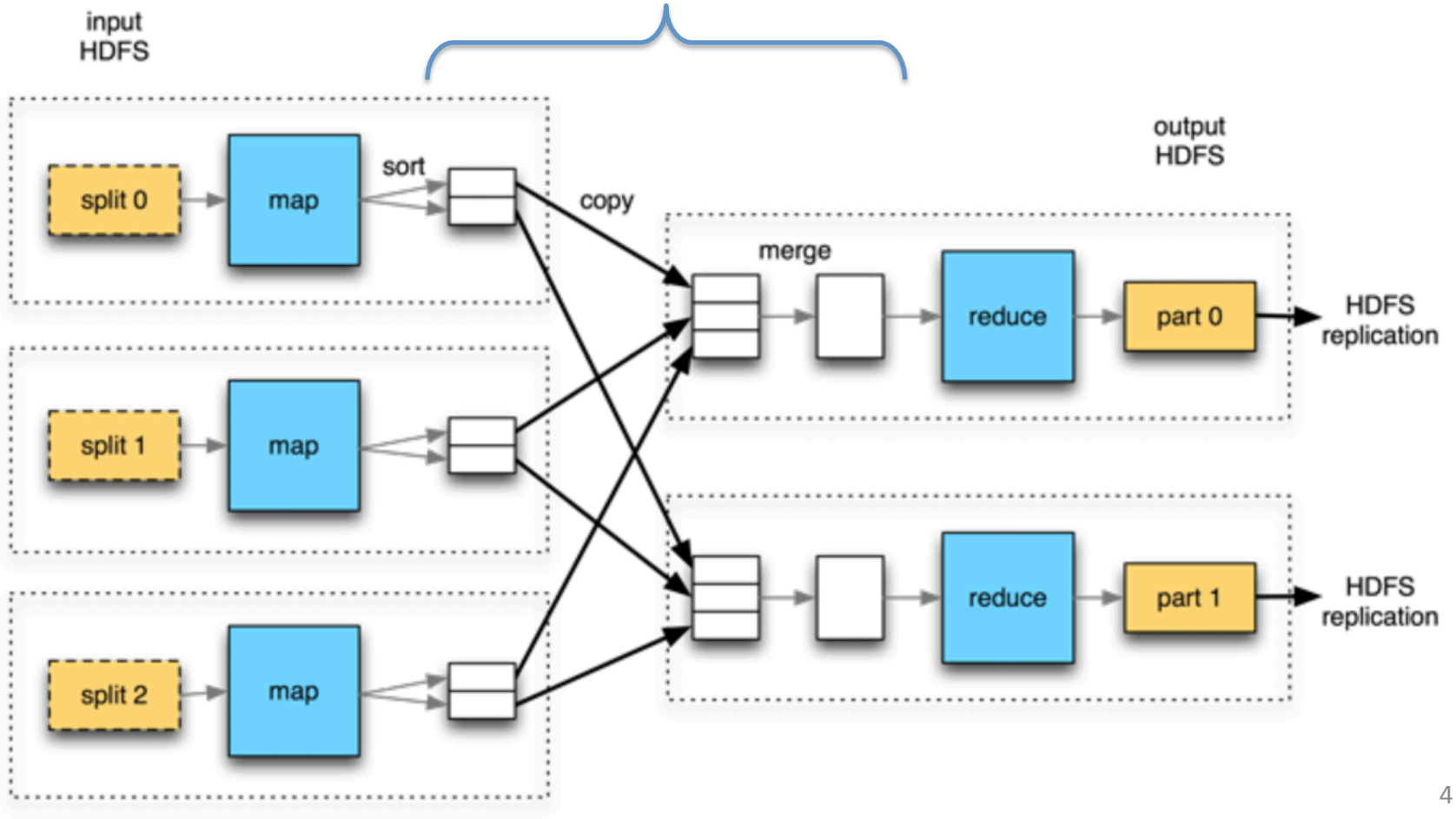
- Map applies a function *f* independently on every key/value pair (transformation)
  f(key, value) → *list(key, value)*
- Reduce applies *f* to **all** values associated with the same key (aggregation)
  f(key,list(value)) → *list(key,value)*

WordCount using Map and Reduce



Map- Reduce execution steps

Shuffle &Sort: group by key and transfer to reducer



Combine

- Aggregation on data produced by an instance of Map (1 server)
  f(key,list(value)) → *list(key,value)*
  - Looks like a reduce
  - Different contract: not all values, some values
  - Input and output types are the same

Hadoop Map-Reduce

- Parallelism
  - Deploys application on a cluster of servers
  - Map can be executed simultaneously on all input (key,value)
  - Reduce can be executed simultaneously for different keys
  - Data spread over multiple disks (throughput)
- Fault-tolerance
  - Reliable storage (HDFS)
  - Re-execute failed tasks

Flickr top-tags

- For each country, find the *k* most frequent tags
  - Ex *k*=5 country=CH
    - germany 3311
    - deutschland 2183
    - frankfurt 1024
    - hessen 870
    - stuttgart 618
- Rectangles are not the most accurate representation of country boundaries, but let's focus on Hadoop

Decomposing a problem as a Map-Reduce job

- Most frequent tags per country
  - Need to get all information about a country to compare tag frequency

  →Reduce with country as a key

Flickr Map Function

- 1 line is meta-data about a picture
  - Extract country
    - Latitude field 11
    - Longitude field 10
    - Country.getCountryAt
  - Extract tags
    - User-tags field 8
    - Tags separated by « , »
  - Emit (country,tag) for each tag

Flickr Map Function

```
public static class MyMapper extends Mapper<LongWritable, Text, Text, Text> {
    @Override
    protected void map(LongWritable key, Text value, Context context)
            throws IOException, InterruptedException, NumberFormatException {
        String[] data = value.toString().split("\t");
        String longitudeText = data[10];
        String latitudeText = data[11];
        String userTags = URLDecoder.decode(data[8], "UTF-8");
        if (!longitudeText.isEmpty() && !latitudeText.isEmpty() && !userTags.isEmpty()) {
            Country c = Country.getCountryAt(Double.parseDouble(latitudeText), Double.parseDouble(longitudeText));
            if (c != null) {
                for (String tag : userTags.split(",")) {
                    context.write(new Text(c.toString()), new Text(tag));
                }
            }
        }
    }
}
```

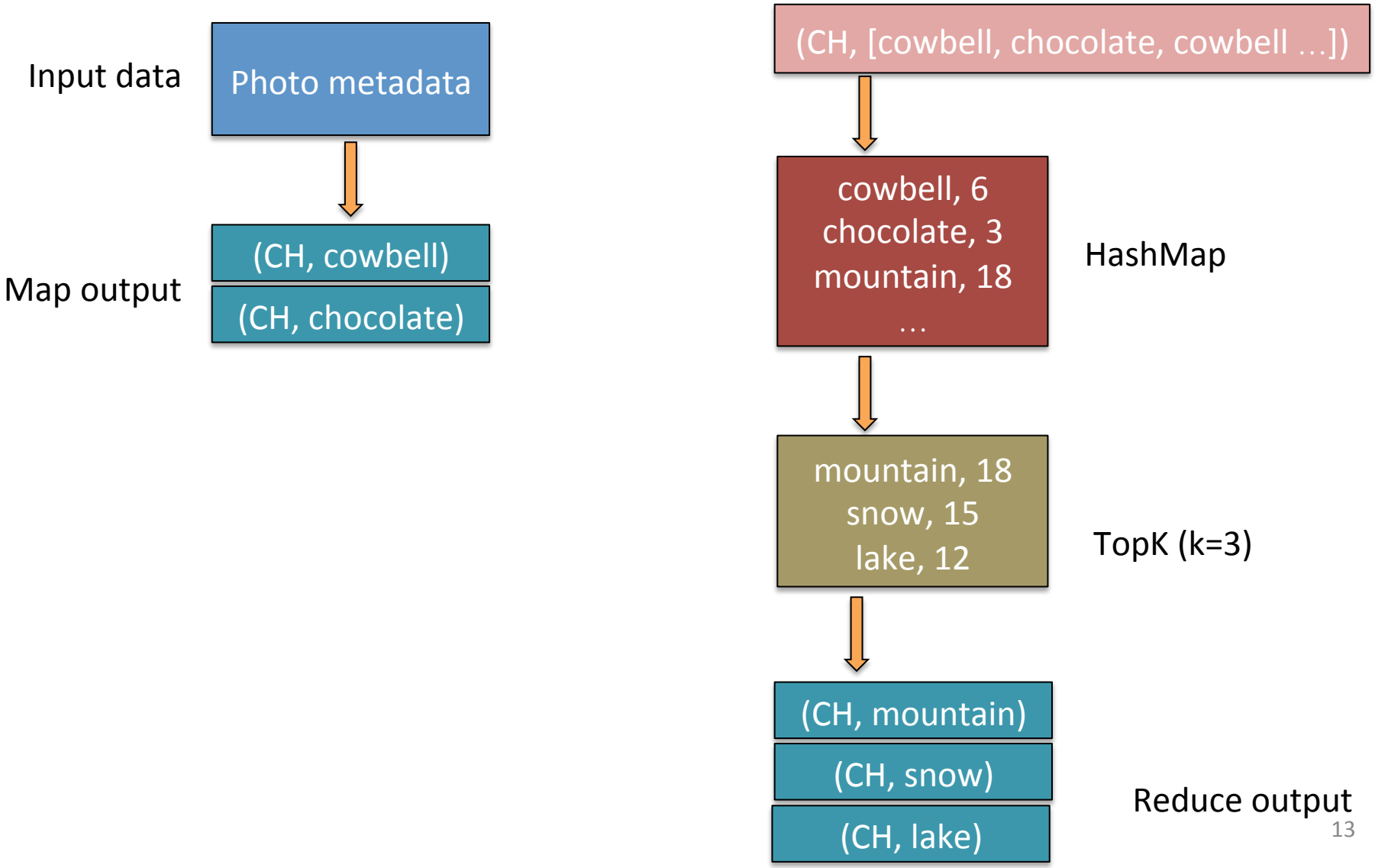
Flickr Reduce Function

- Key = country, Values = tags
  - Compute frequency of tags
    - Aggregate counts using a HashMap
  - Find *k* most frequent tags
    - TopK class provided
  - Emit *k* (country, tag) pairs

Flickr Reduce Function

```
public static class MyReducer extends Reducer<Text, Text, Text, Text> {
    @Override
    protected void reduce(Text key, Iterable<Text> values, Context context)
            throws IOException, InterruptedException {
        Map<String, Integer> map = new HashMap<String, Integer>();
        for (Text value : values) {
            if (map.containsKey(value.toString()))
                map.put(value.toString(), map.get(value.toString()) + 1);
            else
                map.put(value.toString(), 1);
        }
        TopK<String> tk = new TopK<>(context.getConfiguration().getInt("K", 1));
        for (String k : map.keySet()) {
            tk.addElement(k, map.get(k));
        }
        for (TopK.TopKEntry<String> e : tk.getResult()) {
            context.write(key, new Text(e.getElem() + " (freq:" + e.getScore() + ")"));
        }
    }
}
```

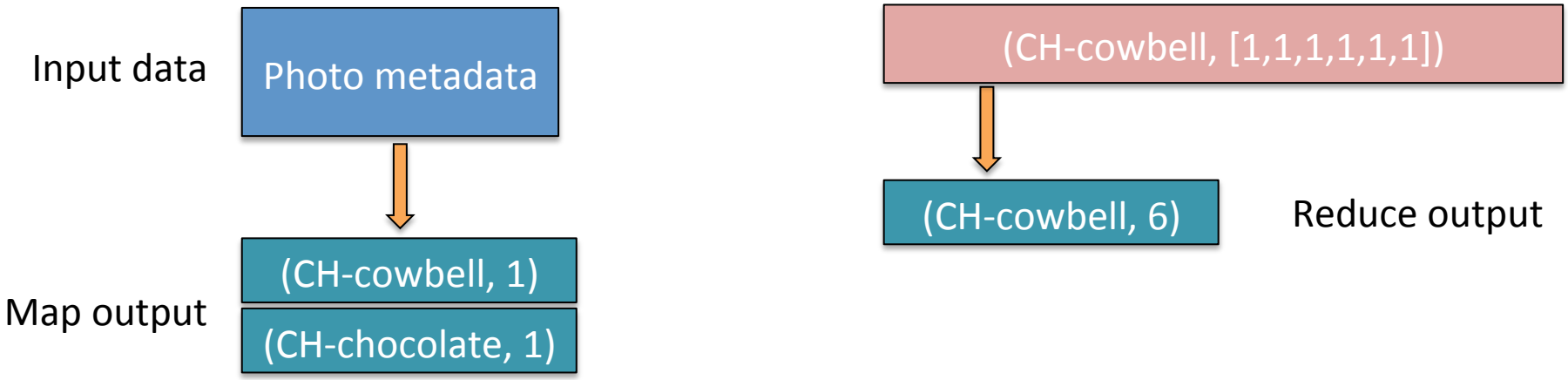
1 job solution overview



1 job solution: memory usage

- Size of HashMap: number of distinct tags used in this country
  - Users add noise to the data (random tags)
  - Can we avoid it?
- HashMap used because
  - Values arrive in a random order (secondary sort for advanced users)
  - Need to remember frequency of all tags as long as we haven't enumerated all values

2 job solution: phase 1 (aka WordCount)



2 job solution: phase 2