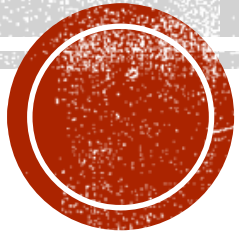
# MAPREDUCE

3.1 MapReduce Paradigm



# WHAT IS A MAPREDUCE?

- Terms borrowed from functional language (e.g., Lisp)
- Sum of squares:
- **(map square (1 2 3 4))**

  Output (1 4 9 16)

  [processes each word sequentially and independently]
- **(reduce + (1 4 9 16))**

  (+16 (+9 (+4 (+1))))

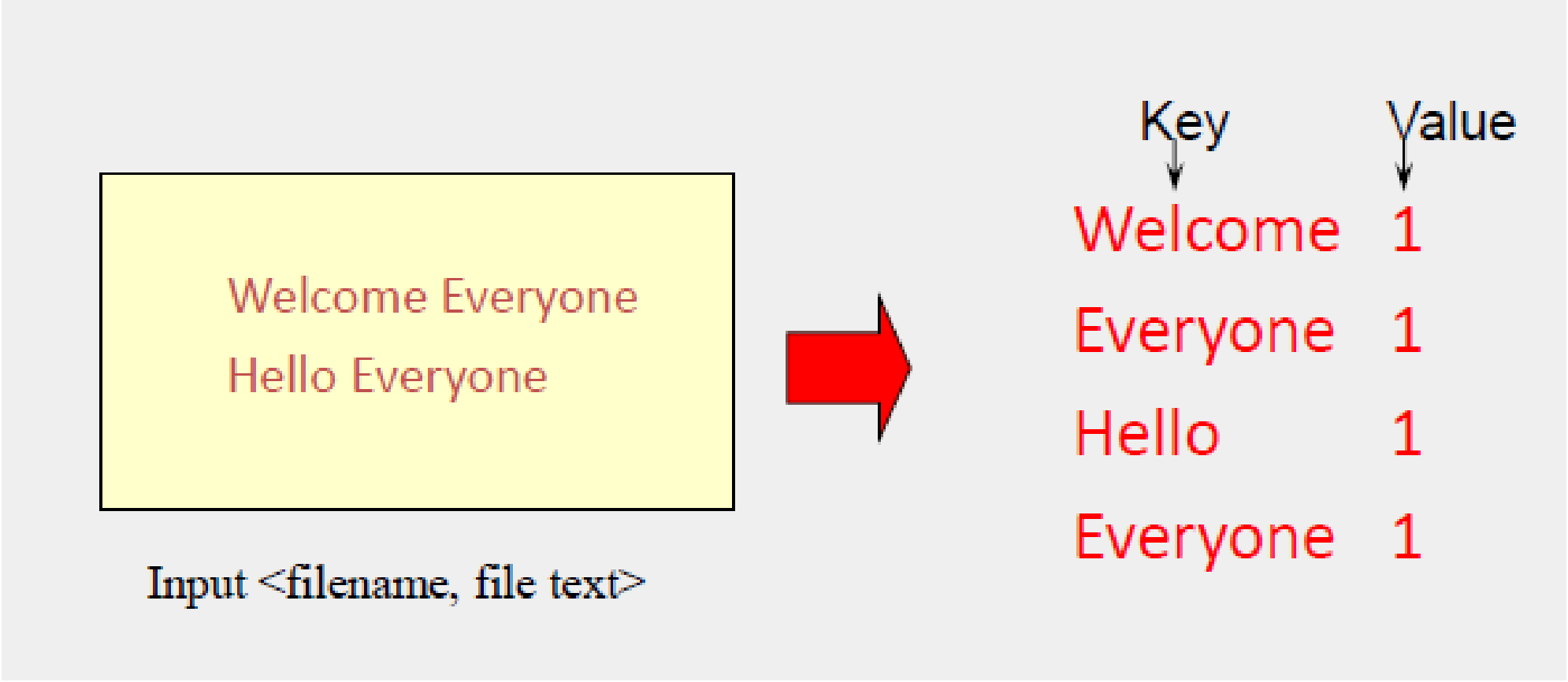
  Output: 30

  [processes set of all records in batches]
- Let's consider a sample application: Word Count
  - You are given a huge dataset(e.g., Wikipedia dump or all of Shakepeare's works) and asked to list the count for each of the words in each of the documents therein



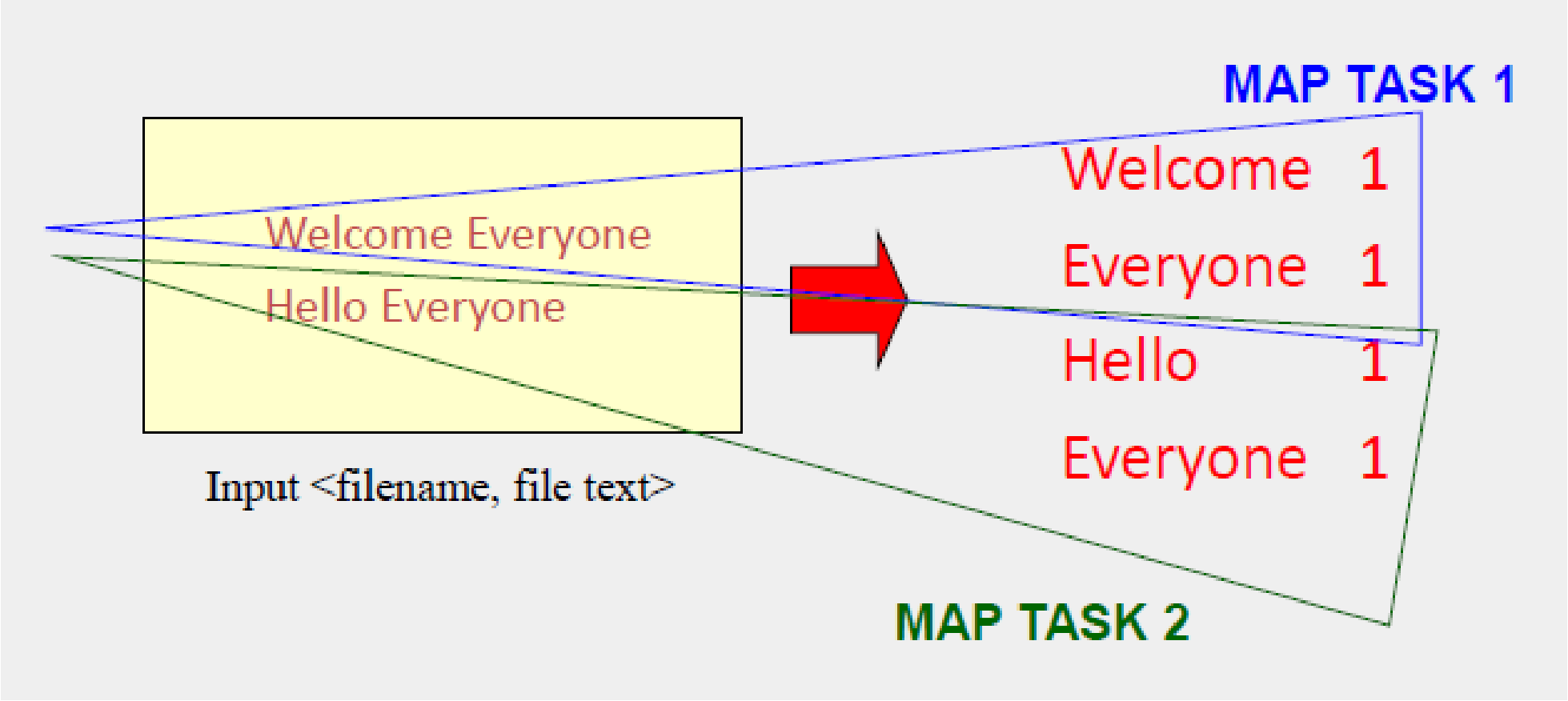
# MAP

- Process individual records to generate intermediate key/value pairs.



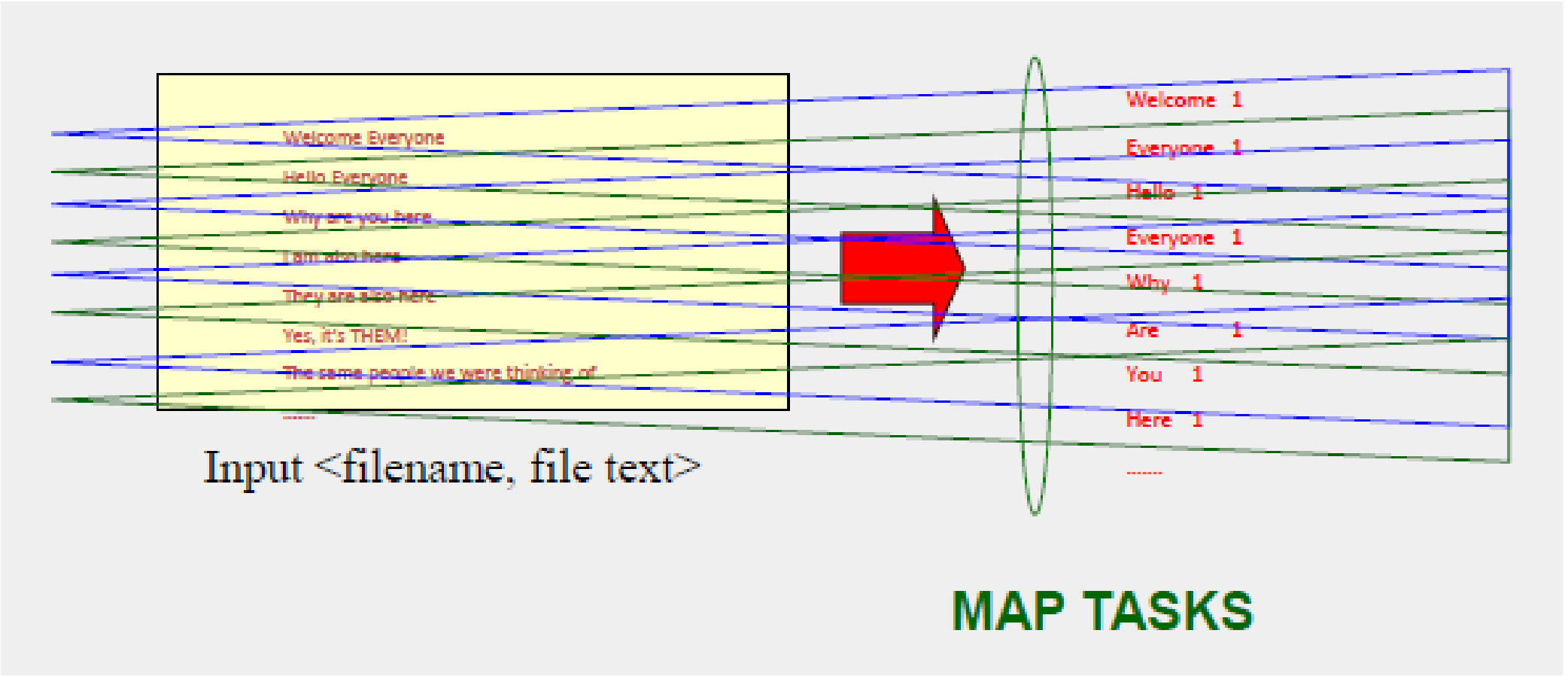
# MAP

- Parallely process individual records to generate intermediate key/value pairs.



# MAP

- Parallely process a large number of individual records to generate intermediate key/value pairs.

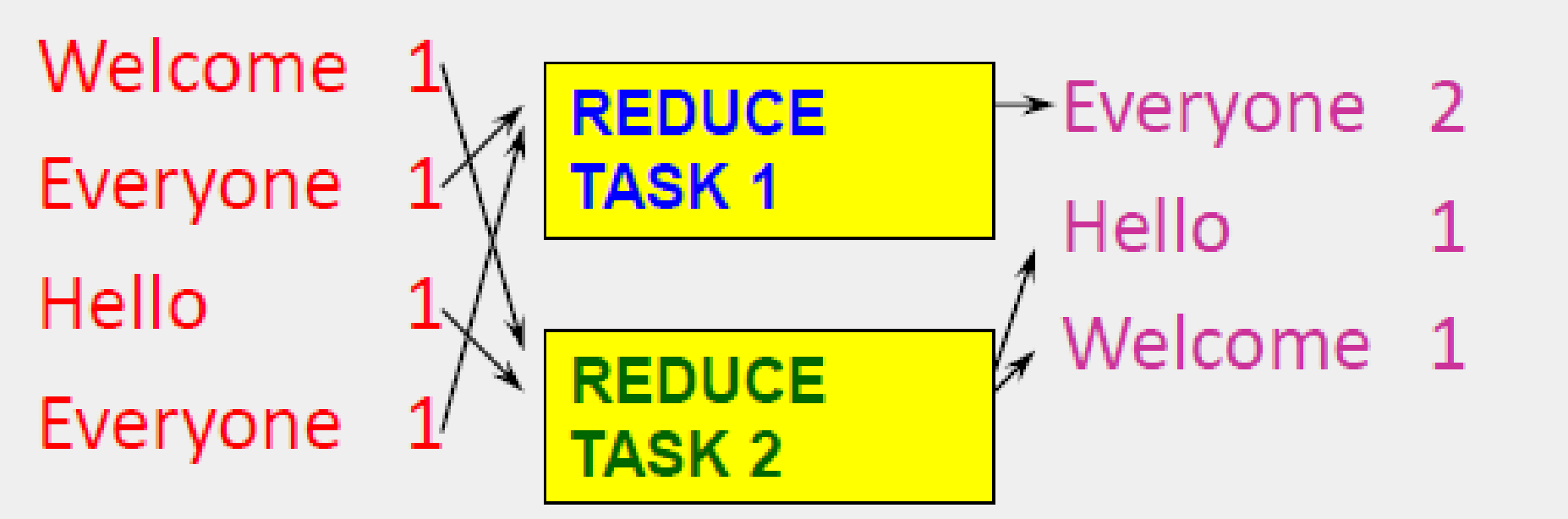


# REDUCE

- Reduce processes and merges all intermediate values associated per key.

# REDUCE

- Each key assigned to one Reduce
- Parallely processes and merges all intermediate values by partitioning keys.




- Popular: hash partitioning, i.e. key is assigned reduce # = hash(key)% number of reduce servers.



# HADOOP CODE - MAP

```
public static class MapClass extends MapReduceBase
             implements Mapper<LongWritable, Text, Text,
               IntWritable> {
  private final static IntWritable one =
    new IntWritable(1);
  private Text word = new Text();

  public void map( LongWritable key, Text value,
             OutputCollector<Text, IntWritable> output,
             Reporter reporter)
    throws IOException {
    String line = value.toString();
    StringTokenizer itr = new StringTokenizer(line);
    while (itr.hasMoreTokens()) {
             word.set(itr.nextToken());
             output.collect(word, one);
    }
  }
} // Source: http://developer.yahoo.com/hadoop/tutorial/module4.html#wordcount
```

# HADOOP CODE - REDUCE

```
public static class ReduceClass extends MapReduceBase
            implements Reducer<Text, IntWritable, Text,
            IntWritable> {
  public void reduce(
            Text key,
            Iterator<IntWritable> values,
            OutputCollector<Text, IntWritable> output,
            Reporter reporter)
      throws IOException {
            int sum = 0;
            while (values.hasNext()) {
              sum += values.next().get();
            }
            output.collect(key, new IntWritable(sum));
  }
} // Source: http://developer.yahoo.com/hadoop/tutorial/module4.html#wordcount
```

# HADOOP CODE - DRIVER

```
// Tells Hadoop how to run your Map-Reduce job
public void run (String inputPath, String outputPath)
            throws Exception {
  // The job. WordCount contains MapClass and Reduce.
  JobConf conf = new JobConf(WordCount.class);
  conf.setJobName("mywordcount");
  // The keys are words
  (strings) conf.setOutputKeyClass(Text.class);
  // The values are counts (ints)
  conf.setOutputValueClass(IntWritable.class);
  conf.setMapperClass(MapClass.class);
  conf.setReducerClass(ReduceClass.class);
  FileInputFormat.addInputPath(
            conf, newPath(inputPath));
  FileOutputFormat.setOutputPath(
            conf, new Path(outputPath));
  JobClient.runJob(conf);
} // Source: http://developer.yahoo.com/hadoop/tutorial/module4.html#wordcount
```

# COPYRIGHT AND IP

- Cloud Computing Concepts, Coursera
  - Indranil Gupta, University of Illinois Urbana Champaign