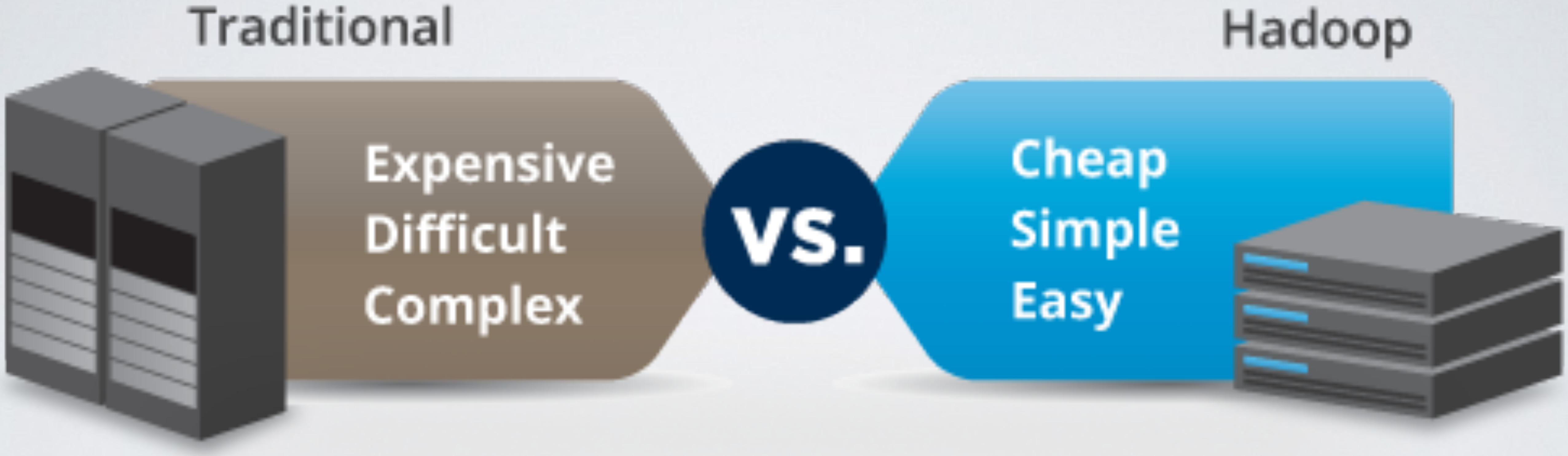
# hadoop

…An Introduction

# WHY?

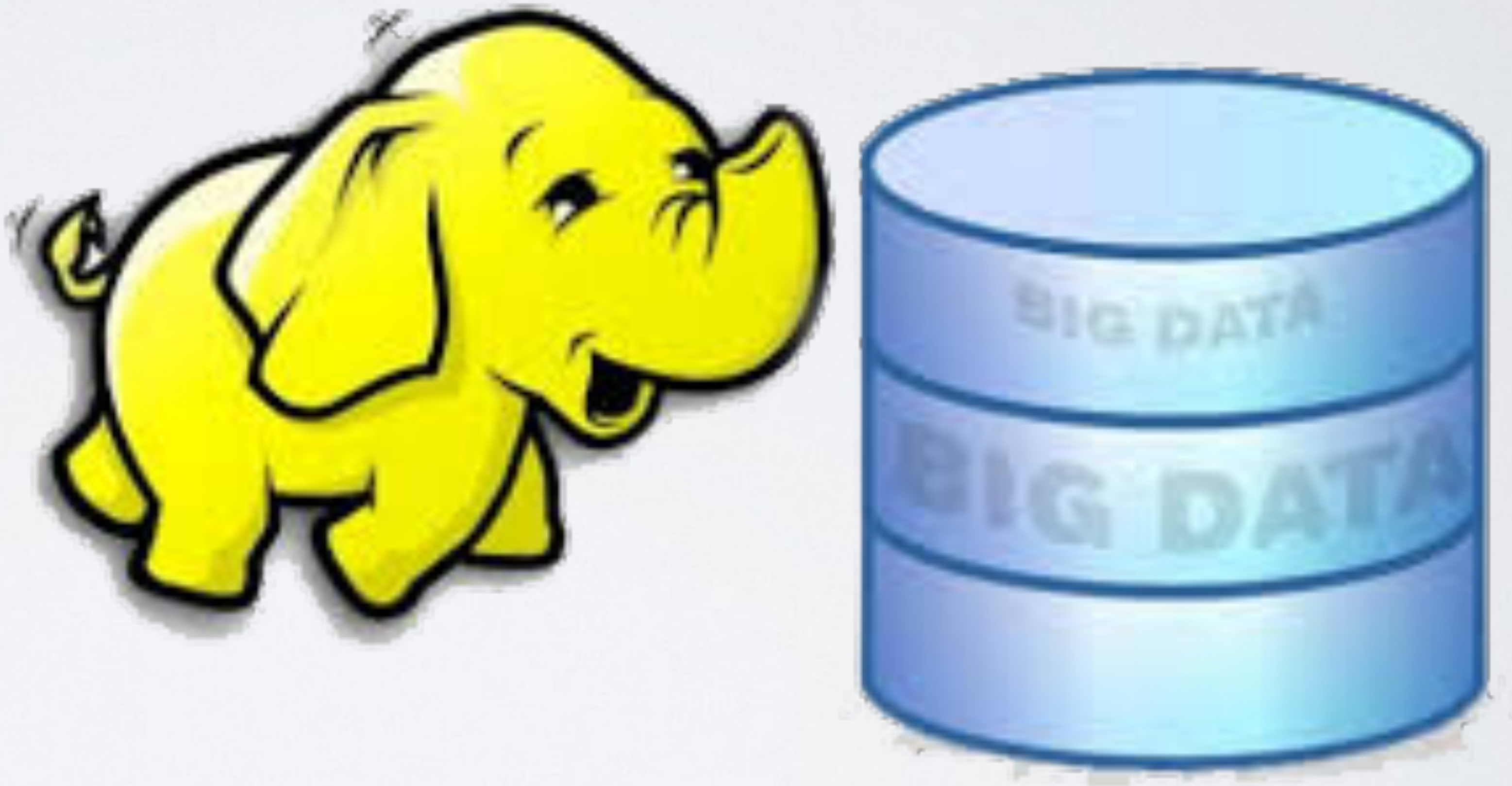
- Volume - so much
- Velocity - so fast
- Variety - so many types

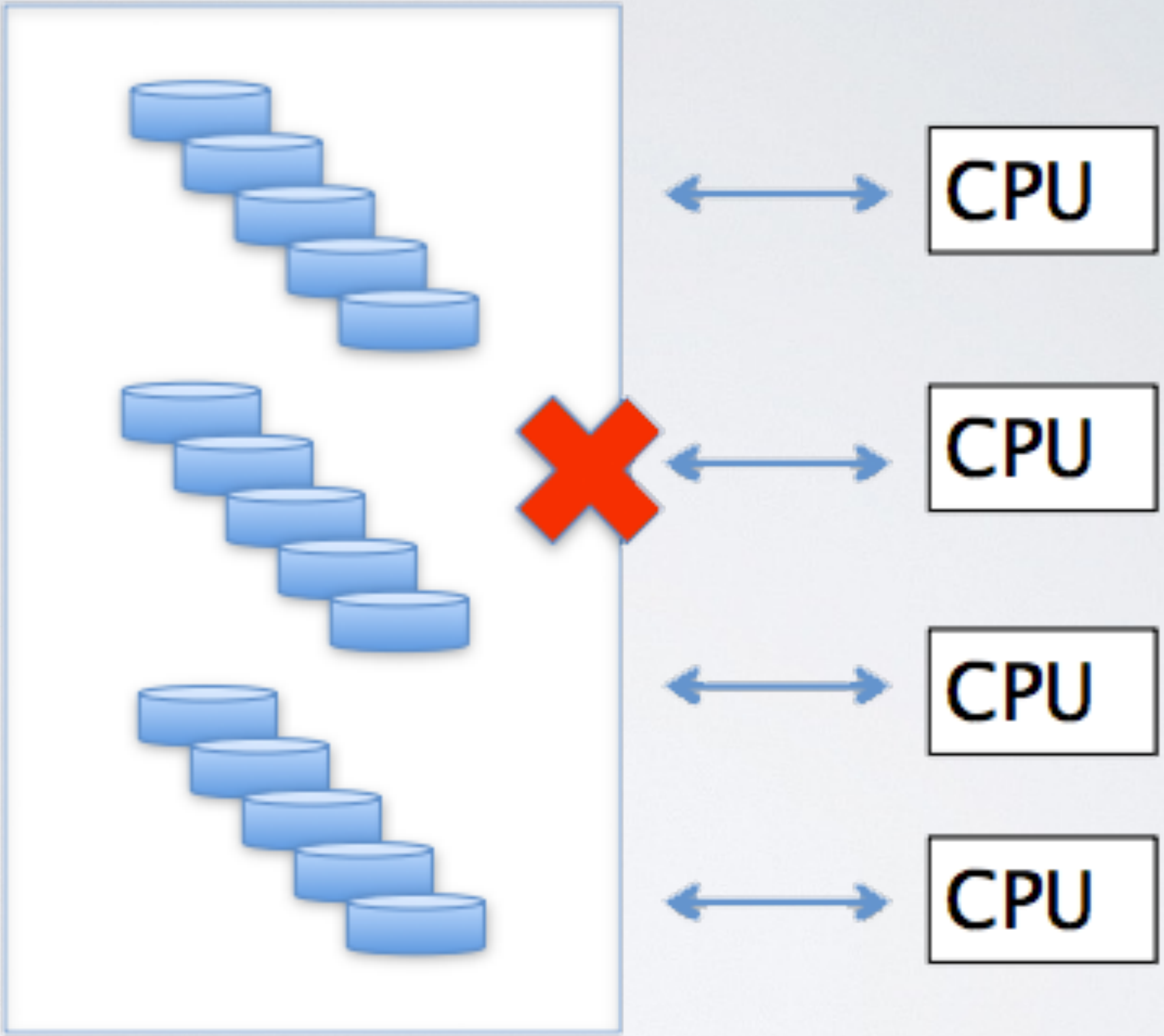


The old way just wasn't cutting it

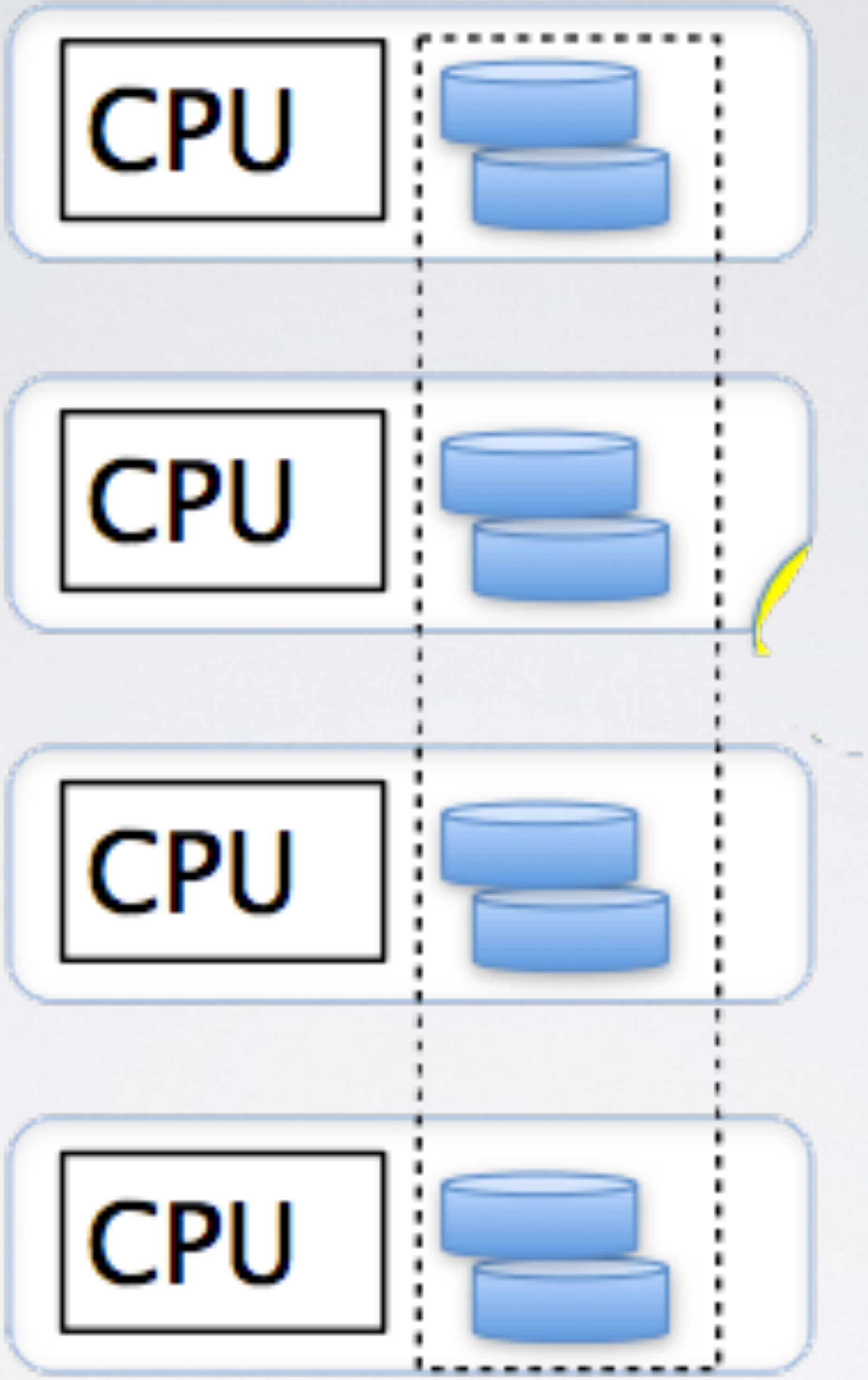
# SO HOW DO WE DO IT?

- HDFS
- Map Reduce





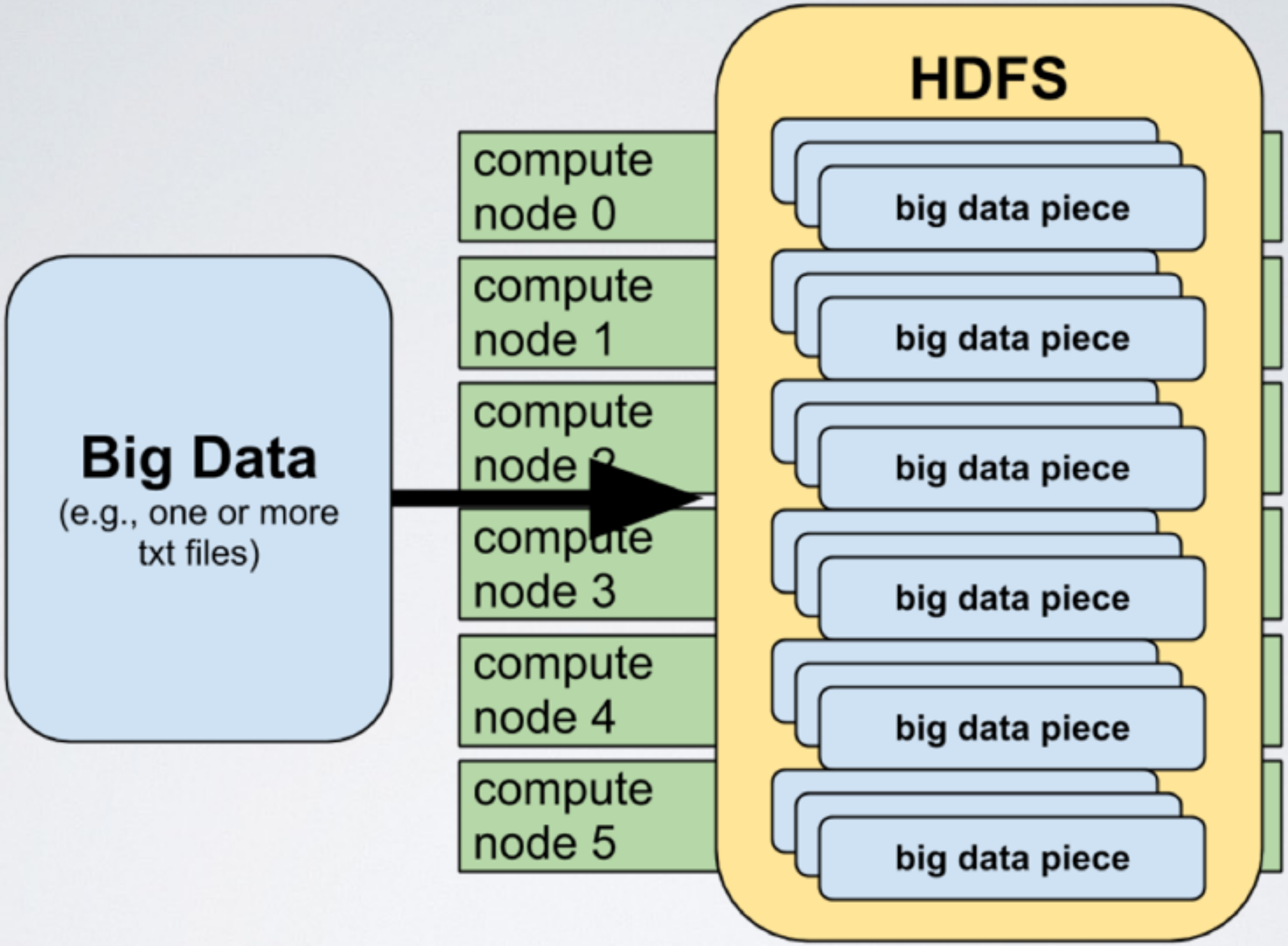
It just doesn't scale



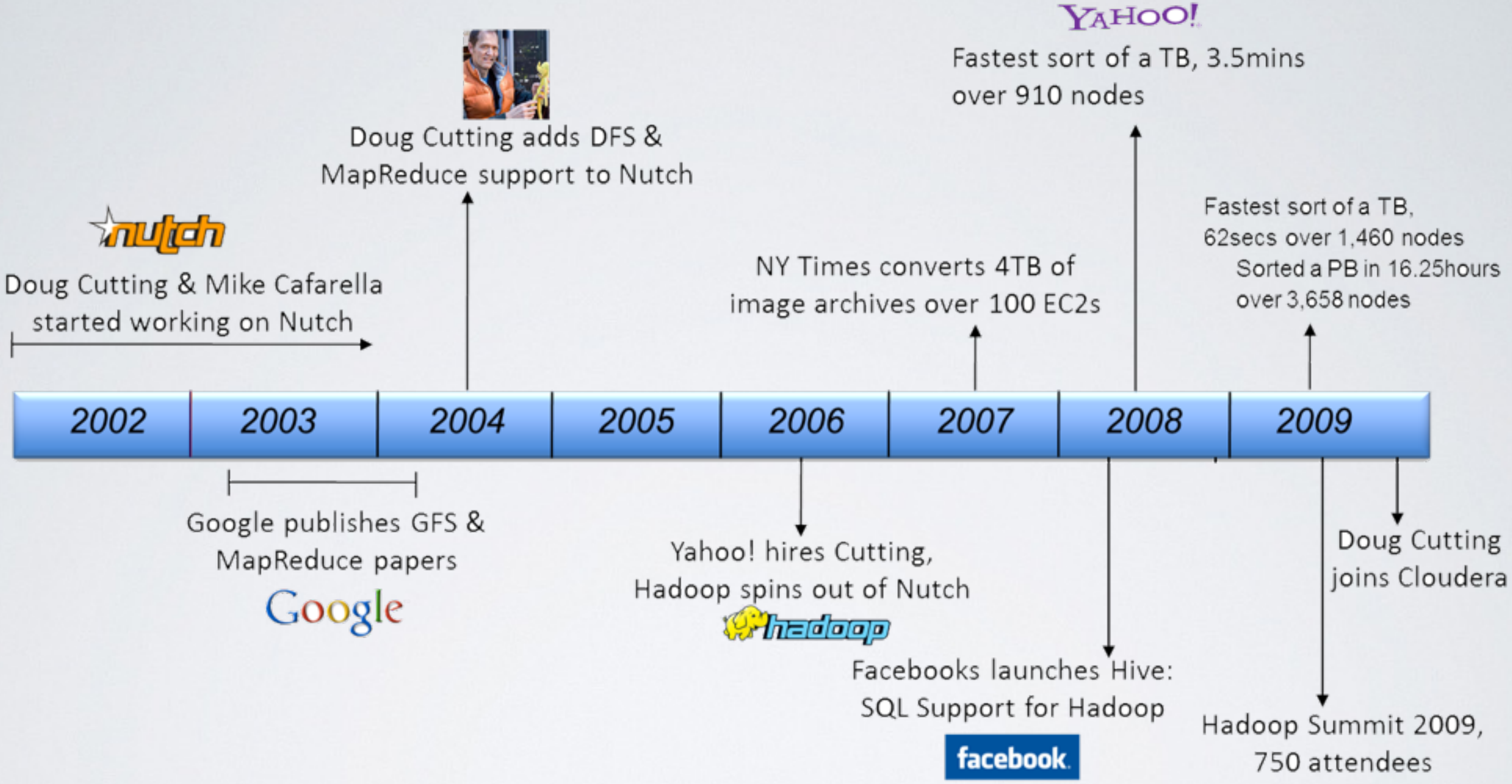
That’s better

# ENTER HADOOP

- Open source Java
- Framework for big data
- Scalable
- Fault tolerant
- Commodity hardware



Distribute the data and the processing



Doug Cutting & Mike Cafarella started working on Nutch

Google publishes GFS & MapReduce papers

Doug Cutting adds DFS & MapReduce support to Nutch

Yahoo! hires Cutting, Hadoop spins out of Nutch

NY Times converts 4TB of image archives over 100 EC2s

Fastest sort of a TB, 3.5mins over 910 nodes

Facebooks launches Hive: SQL Support for Hadoop

Fastest sort of a TB, 62secs over 1,460 nodes
Sorted a PB in 16.25hours over 3,658 nodes

Doug Cutting joins Cloudera

Hadoop Summit 2009, 750 attendees

# A brief history

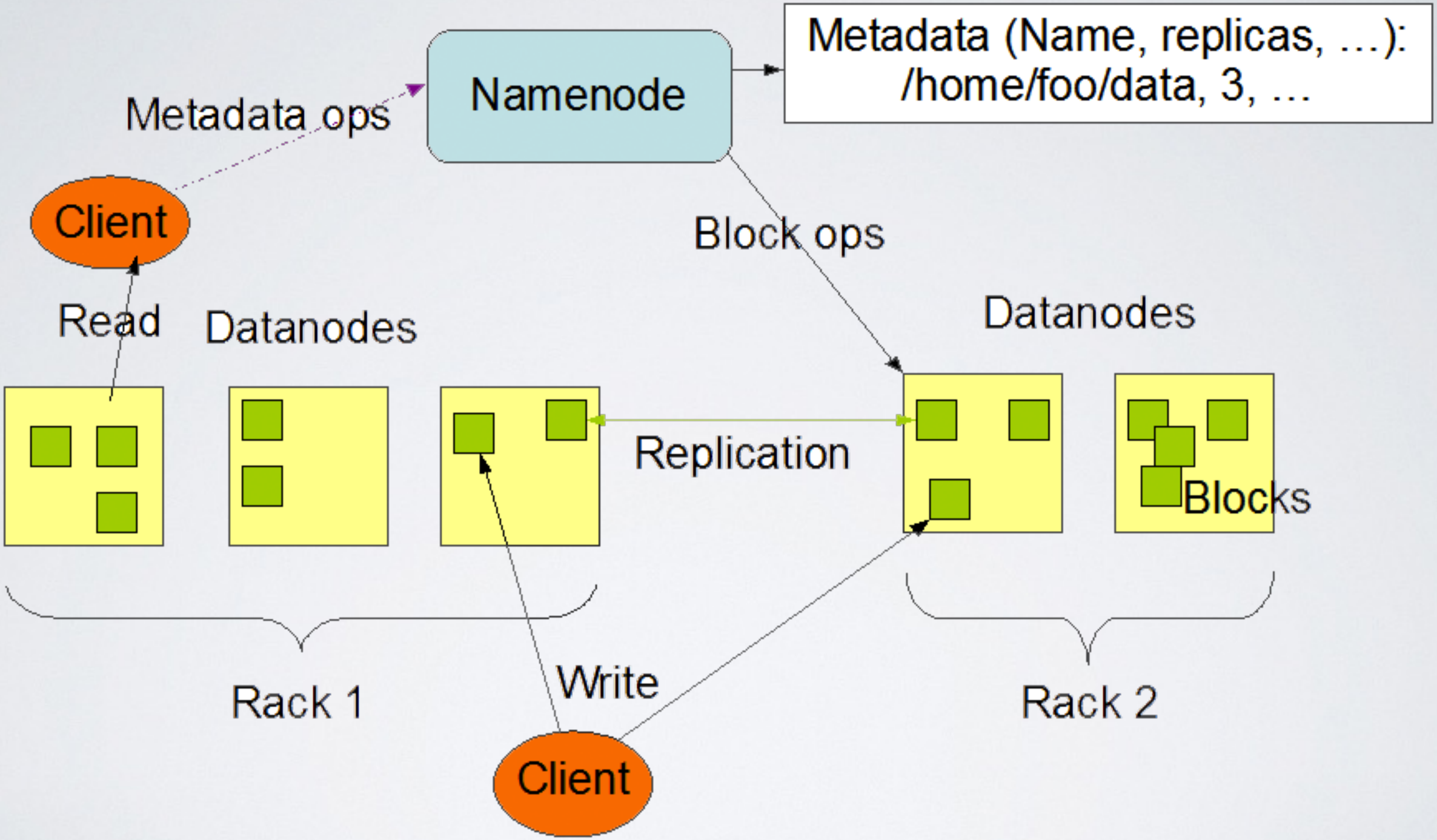
facebook
YAHOO!
twitter
NETFLIX
LinkedIn
The New York Times
eHarmony
su
Microsoft
IBM
ebay
amazon

Some hadoop users

# HDFS

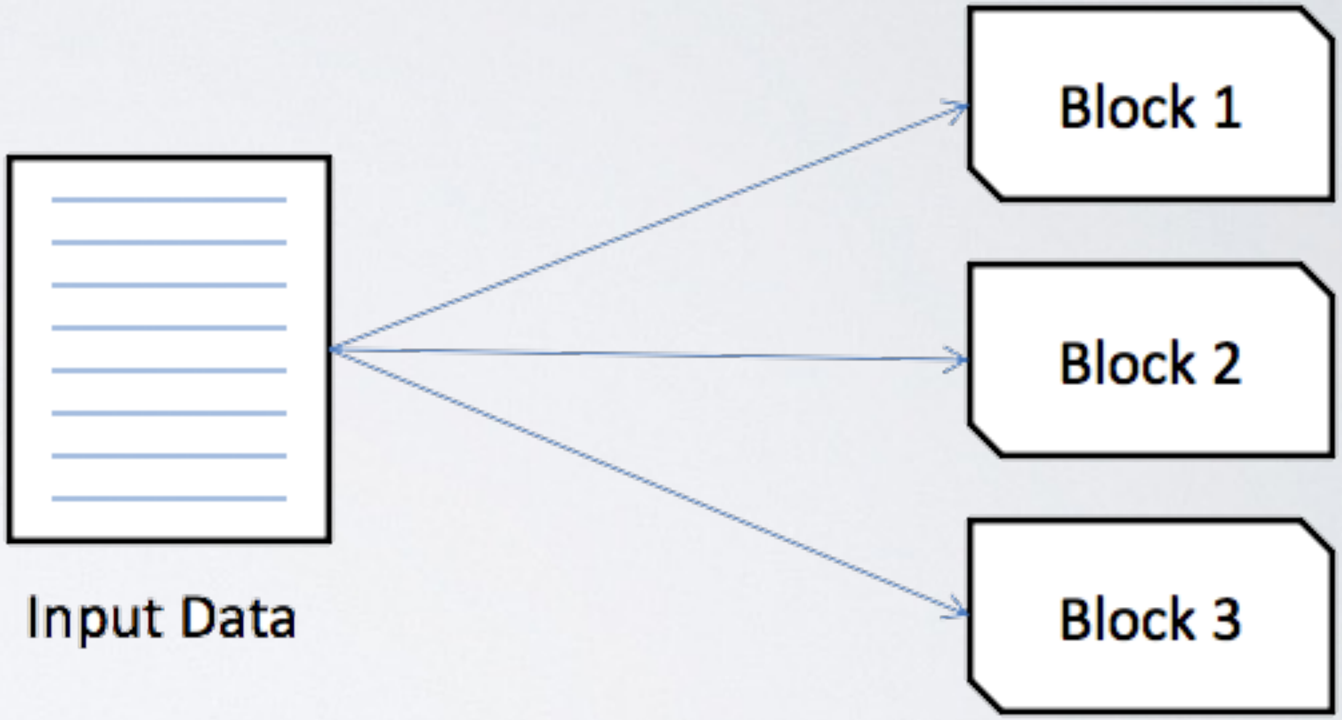
- Master/slave architecture
- Fault tolerant
- Commodity hardware
- Streaming access to FS data

# HDFS Architecture



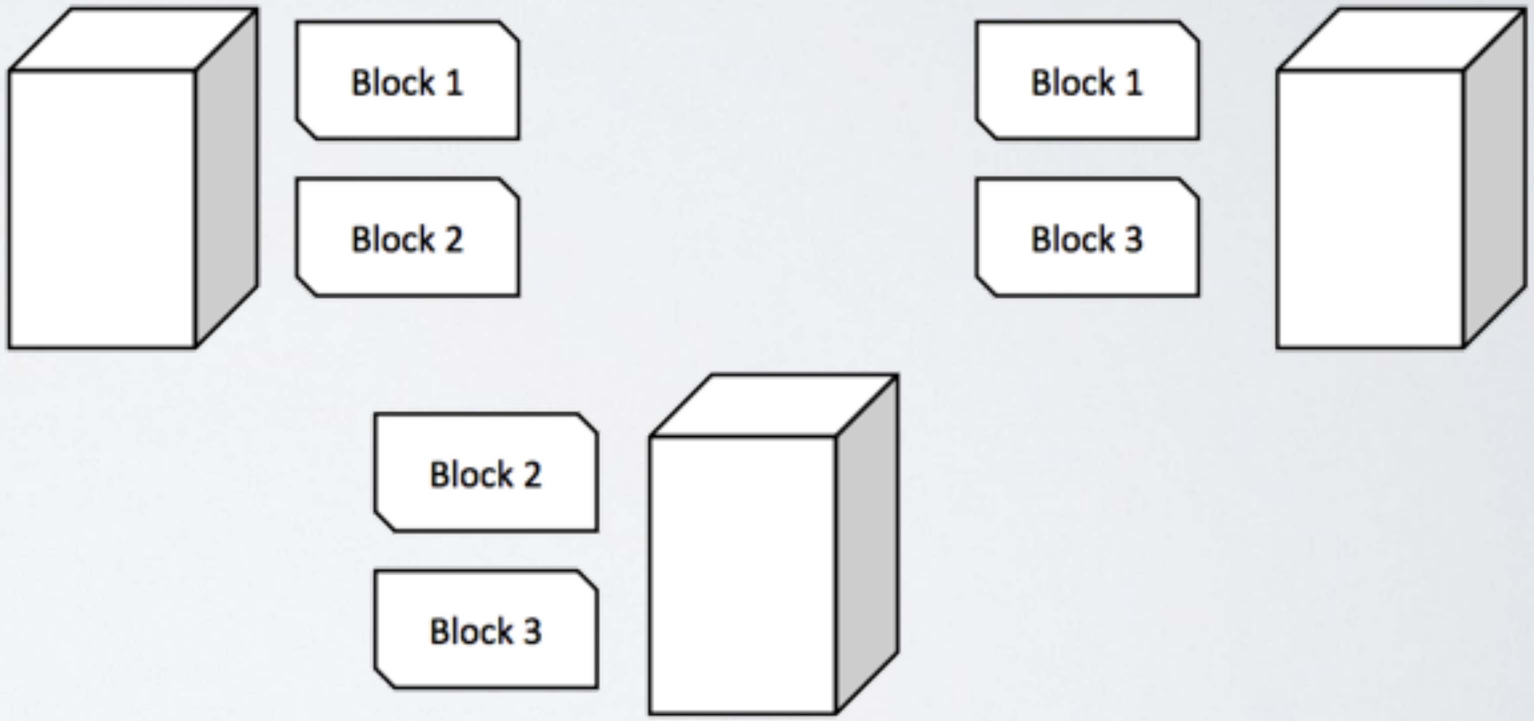
# HDFS - BLOCKS

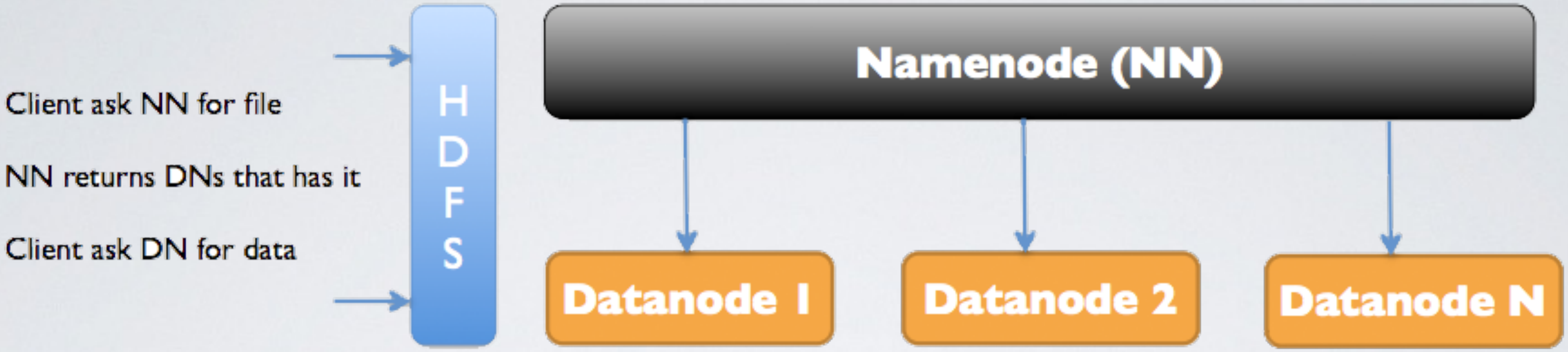
- Splits data into blocks
- Easier to distribute work



# HDFS - REPLICATION

- Blocks replicated across nodes
- Fault tolerant
- 3x replications by default



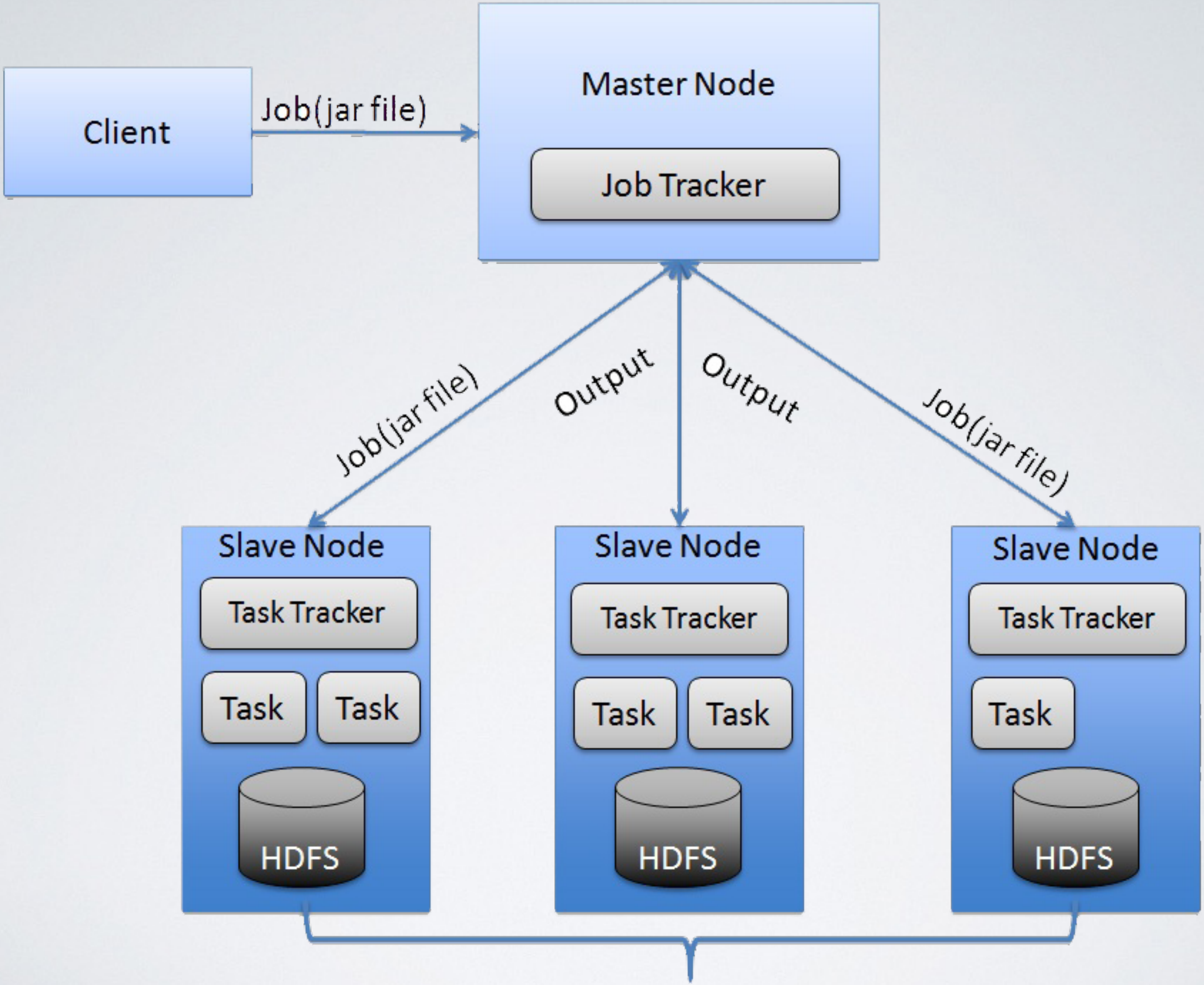


## Namenode - Master

- FS metadata
- Block replication
- R/W access to files

## Datanode - Slave

- Contains data
- Informs namenode



Client

Job(jar file)

Master Node

Job Tracker

Job(jar file)

Output

Output

Job(jar file)

Slave Node

Task Tracker

Task

Task

HDFS

Slave Node

Task Tracker

Task

Task

HDFS

Slave Node

Task Tracker

Task

HDFS

Data replicated across nodes

# Map

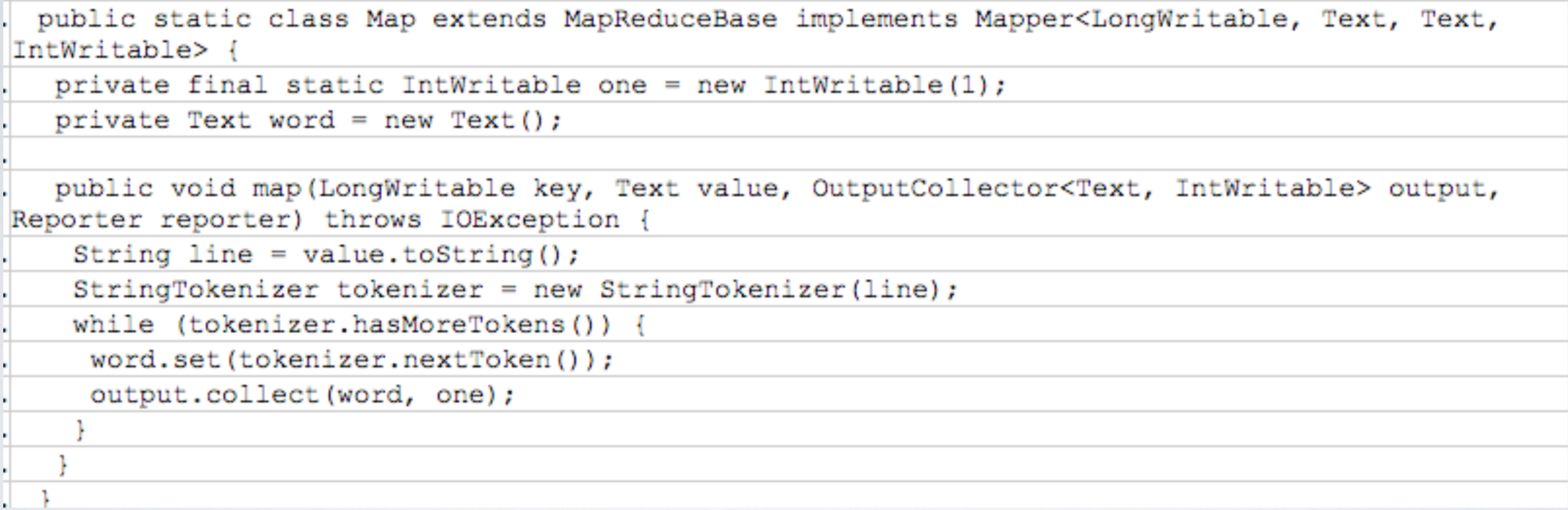
```
def sqr(n):
  return n * n

list = [1,2,3,4]

map(sqr, list) -> [1,4,9,16]
```

# Reduce

```
def add(i, j):
  return i + j

list = [1,2,3,4]

reduce(add, list) -> 10
```

# MapReduce

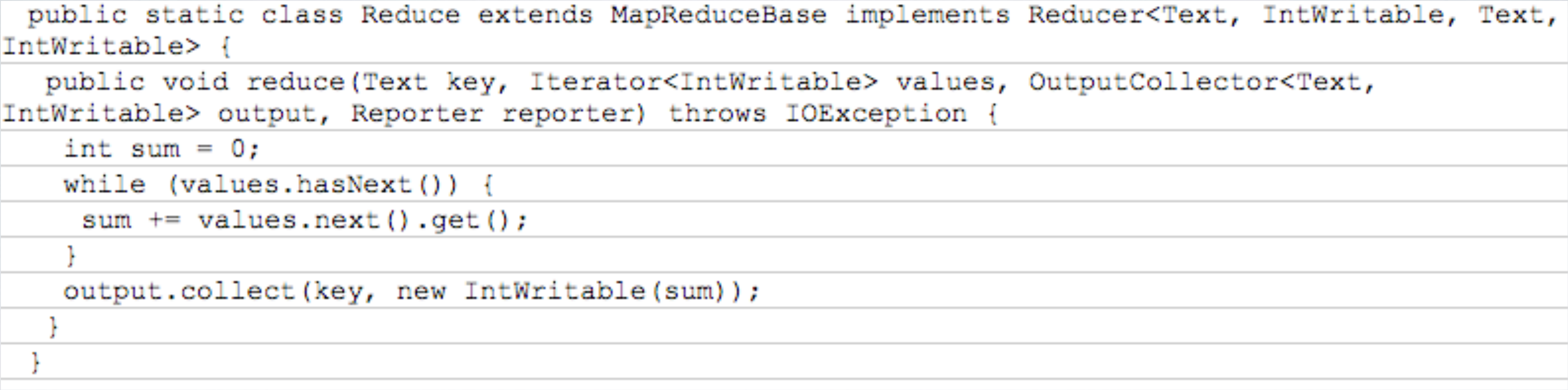
```
def MapReduce(data, mapper, reducer):
  return reduce(reducer, map(mapper, data))

MapReduce(list, sqr, add) -> 30
```



```
 public static class Map 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 tokenizer = new StringTokenizer(line);
   while (tokenizer.hasMoreTokens()) {
    word.set(tokenizer.nextToken());
    output.collect(word, one);
   }
  }
 }
```

# MAP

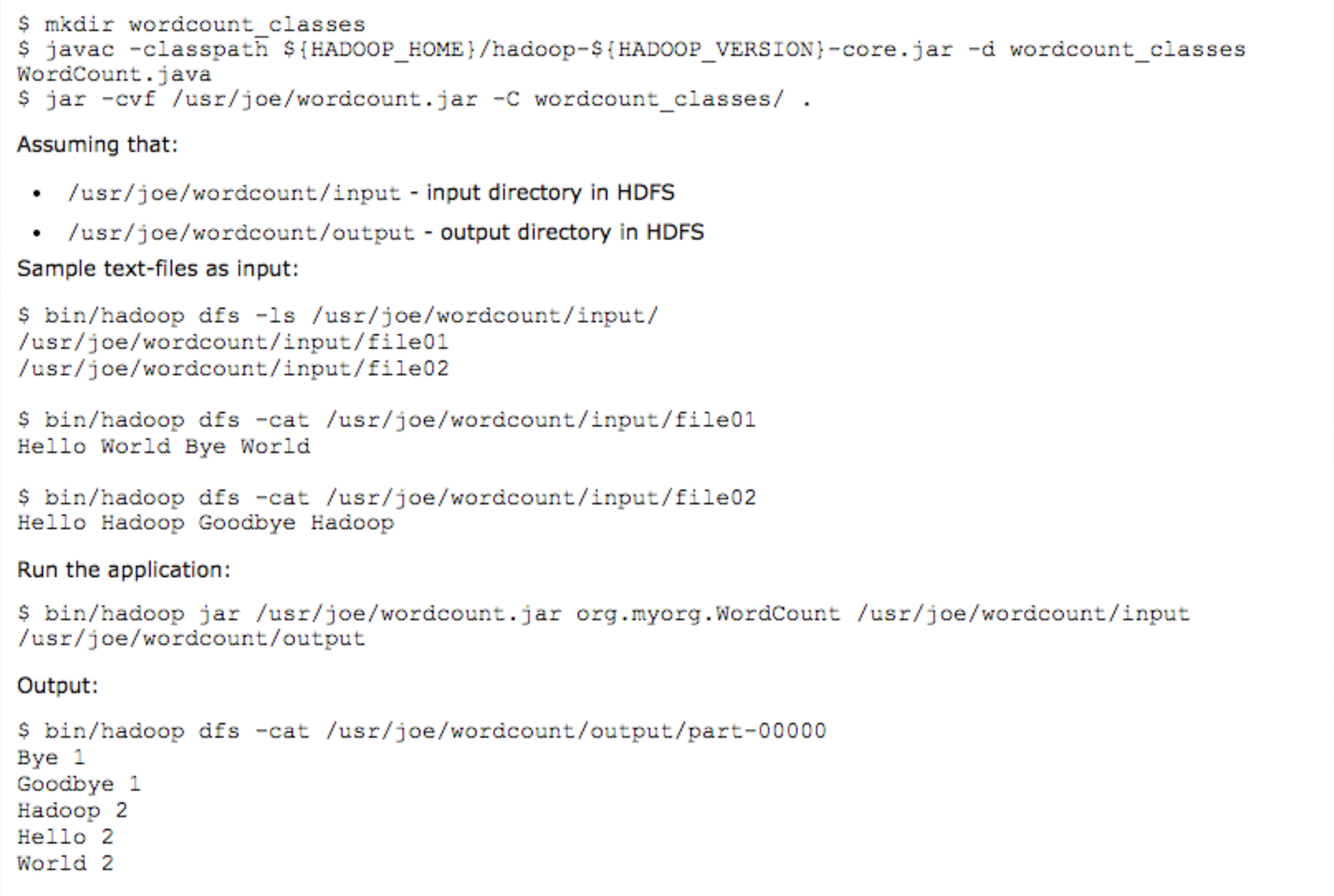
wordcount.java



```
 public static class Reduce 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));
  }
 }
```

# REDUCE

wordcount.java



```
$ mkdir wordcount_classes
$ javac -classpath ${HADOOP_HOME}/hadoop-${HADOOP_VERSION}-core.jar -d wordcount_classes
WordCount.java
$ jar -cvf /usr/joe/wordcount.jar -C wordcount_classes/ .
```

Assuming that:

- `/usr/joe/wordcount/input` - input directory in HDFS
- `/usr/joe/wordcount/output` - output directory in HDFS

Sample text-files as input:

```
$ bin/hadoop dfs -ls /usr/joe/wordcount/input/
/usr/joe/wordcount/input/file01
/usr/joe/wordcount/input/file02

$ bin/hadoop dfs -cat /usr/joe/wordcount/input/file01
Hello World Bye World

$ bin/hadoop dfs -cat /usr/joe/wordcount/input/file02
Hello Hadoop Goodbye Hadoop
```

Run the application:

```
$ bin/hadoop jar /usr/joe/wordcount.jar org.myorg.WordCount /usr/joe/wordcount/input
/usr/joe/wordcount/output
```

Output:

```
$ bin/hadoop dfs -cat /usr/joe/wordcount/output/part-00000
Bye 1
Goodbye 1
Hadoop 2
Hello 2
World 2
```

# RUNNING IT

wordcount.java