SoftwareEntwicklung ▸ Beratung ▸ Schulung

# Hadoop
# Think Large!




Web Site:
www.soebes.com
Email:
training@soebes.com

Dipl.Ing.(FH) Karl Heinz Marbaise

# Agenda

1. Introduction
2. Hadoop Distributed File System (HDFS)
3. Map/Reduce programming paradigm
4. Map/Reduce and HDFS
5. HDFS Installation Requirements + DEMO
6. Real World Example(s)
7. Conclusion

# 1. Introduction

```
block 0: V--------- 0/0               0 1970-01-01 01:00:00 20090619040001--Volume Header--
.
block 1048: -rwxr-xr-x root/root    110912 2005-03-19 20:23:35 ./bin/ash
block 1266: lrwxrwxrwx root/root         0 2006-05-06 15:51:27 ./bin/awk -> gawk
.
block 1523: -rwxr-xr-x root/root     17884 2005-03-19 18:41:22 ./bin/rpm
.
block 5237: -rwxr-xr-x root/root     15428 2005-03-19 21:28:25 ./bin/echo
block 5269: -rwxr-xr-x root/root    235844 2005-03-22 19:58:00 ./bin/gawk
block 5731: -rwxr-xr-x root/root    115424 2005-03-19 21:21:08 ./bin/grep
block 5958: -rwxr-xr-x root/root     51844 2005-03-19 20:15:59 ./bin/gzip
block 6061: -rwxr-xr-x root/root     18000 2005-03-19 21:28:25 ./bin/kill
block 6098: lrwxrwxrwx root/root         0 2006-05-06 15:53:46 ./bin/mail -> /usr/bin/nail
.
block 10480: drwxr-xr-x root/root        0 2005-03-23 22:34:44 ./etc/ppp/ip-up.d/
block 10481: -rw-r--r-- root/root      222 2005-03-19 21:39:38 ./etc/raw
block 10483: -rw-r--r-- root/root     1615 2005-03-19 18:29:54 ./etc/rpc
block 10488: drwxr-xr-x root/root        0 2006-06-10 12:25:23 ./etc/ssh/
block 10489: -rw-r--r-- root/root     2384 2005-03-22 19:53:04 ./etc/ssh/ssh_config
block 10495: -rw-r----- root/root     3459 2006-05-07 12:30:13 ./etc/ssh/sshd_config
block 10503: -rw------- root/root      528 2006-05-06 16:02:58 ./etc/ssh/ssh_host_key
block 10506: -rw-r--r-- root/root      603 2006-05-06 16:03:01 ./etc/ssh/ssh_host_dsa_key.pub
block 10509: -rw-r--r-- root/root      223 2006-05-06 16:03:01 ./etc/ssh/ssh_host_rsa_key.pub
block 10511: -rw------- root/root   111892 2005-03-22 19:53:04 ./etc/ssh/moduli
block 10731: -rw-r--r-- root/root      332 2006-05-06 16:02:58 ./etc/ssh/ssh_host_key.pub
block 10733: -rw------- root/root      668 2006-05-06 16:03:01 ./etc/ssh/ssh_host_dsa_key
block 10736: -rw------- root/root      883 2006-05-06 16:03:01 ./etc/ssh/ssh_host_rsa_key
block 10739: drwxr-xr-x root/root        0 2007-01-20 18:13:19 ./etc/ssl/
```

# 1. Introduction Java implementation

```
public void process(String strLine) {
    String[] columns = strLine.split("[ ]+");
    if (strLine.startsWith("tar: /dev/nst0")) {
        getContext().getCounter(ContentType.EMPTY).increment(1);
        return;
  }
    if (columns[2].startsWith("-")) {
        long sizeInBytes = Long.parseLong(columns[4]);
        getContext().getCounter(ContentType.BYTES).increment(sizeInBytes);
        getContext().getCounter(ContentType.FILES).increment(1);
  }
    if (columns[2].startsWith("d")) {
        context.getCounter(ContentType.DIRECTORIES).increment(1);
  }
    if (columns[2].startsWith("l")) {
        getContext().getCounter(ContentType.LINKS).increment(1);
  }
    if (columns[2].startsWith("V")) {
        getContext().getCounter(ContentType.VOLUMNHEADER).increment(1);
  }
}
```

https://github.com/khmarbaise/hadoop-compare

# 1. Introduction
# Java implementation II

```
public void read(File logFile) {
    BufferedReader in = null;
    try {
        in = new BufferedReader(new FileReader(logFile));
        String str;
        while ((str = in.readLine()) != null) {
            process(str);
        }
    } catch (IOException e) {
        System.err.println("Failure during read:" + e.getMessage());
    } finally {
        try {
            in.close();
        } catch (IOException e) {
            System.err.println("Failure during close:" + e.getMessage());
        }
    }
}
```

https://github.com/khmarbaise/hadoop-compare

# 1. Introduction
# Java implementation III

```
  public static void main(String[] args) {
      TapeLogReader rlr = new TapeLogReader();
      Date started = new Date();
      System.out.println("Started at: " + started);
      rlr.read(new File(args[0]));
      Date ended = new Date();
      System.out.println("Stopped at: " + ended);
      System.out.println("Runtime: "
        + ((ended.getTime() - started.getTime()) / 1000.0)
        + " seconds");

      for (ContentType item : ContentType.values()) {
        Counter counter = rlr.getContext().getCounter(item);
        System.out.println("Counter: "
          + item.name() + " value:" + counter.getValue());
    }
}
```

https://github.com/khmarbaise/hadoop-compare

# 1. Introduction
# Test on Files

```
(18.08 GiB File)
hudson@build:~/hadoop-compare-0.0.1-SNAPSHOT> bin/tlr ../logs/result.log
Started at: Sat Feb 12 12:16:32 CET 2011
Stopped at: Sat Feb 12 12:26:09 CET 2011
Runtime: 577.15 seconds
Counter: FILES value:98129391
Counter: DIRECTORIES value:24770057
Counter: LINKS value:236933
Counter: VOLUMNHEADER value:305
Counter: EMPTY value:209
Counter: BYTES value:24594160609582

(21.91 GiB File)
hudson@build:~/hadoop-compare-0.0.1-SNAPSHOT> bin/tlr ../logs/r.log
Started at: Sat Feb 12 13:28:32 CET 2011
Stopped at: Sat Feb 12 13:39:29 CET 2011
Runtime: 656.252 seconds
Counter: FILES value:118228777
Counter: DIRECTORIES value:30492033
Counter: LINKS value:284020
Counter: VOLUMNHEADER value:364
Counter: EMPTY value:311
Counter: BYTES value:29684714834632
```

https://github.com/khmarbaise/hadoop-compare

# 1. Introduction

- What about if the file(s) become larger?
  - Let us assume the file will get 200 GiB* or 2 TiB or even larger.
  - Rule of thumb calculation:

| Size | Unit | Minutes | Hours | Days |
|---|---|---|---|---|
| 20 | GiB | 10 | | |
| 200 | GiB | 100 | 1.6 | |
| 2 | TiB | 1000 | 16.6 | 0.69 |
| 20 | TiB | 10000 | 166 | 6.94 |
| 200 | TiB | 100000 | 1666 | 69.4 |

*1999 IEC 60027-2 Amendment 2

# 1. Introduction

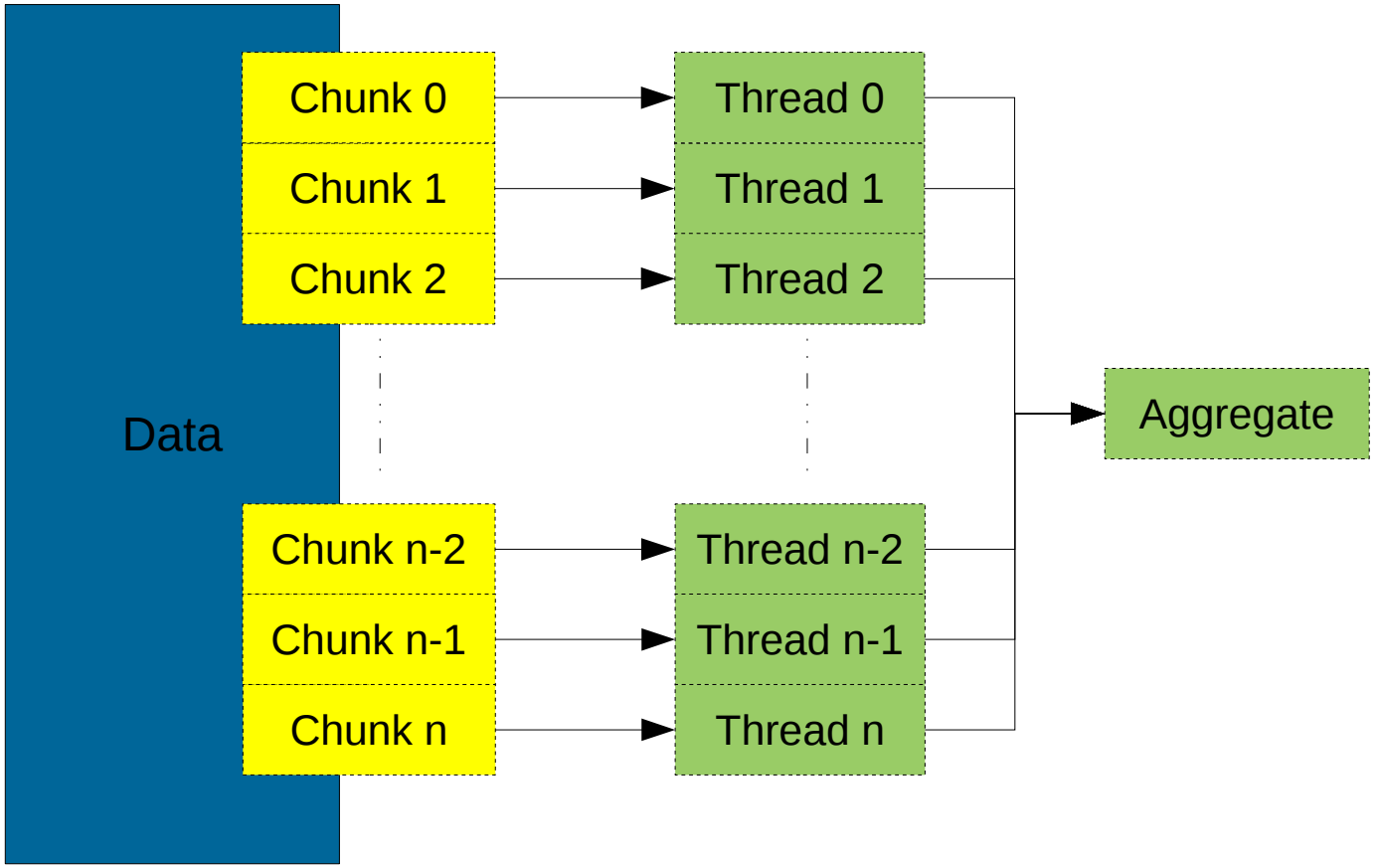
- What can we do to reduce the amount of time to analyze files? One solution might be parallelization:
  - Prerequisites:
    - Split the file(s) into chunks.
    - Synchronization between threads.
    - Limit the number of parallel running threads.
    - Handling of failures.
    - Etc.

# 1. Introduction

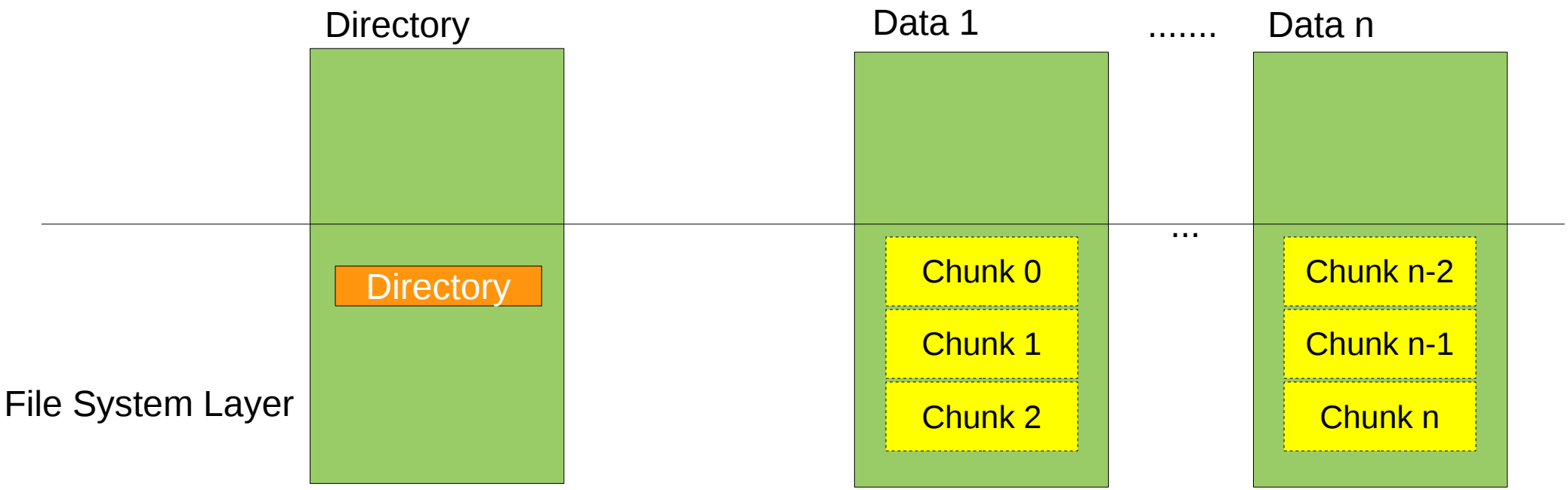
- Limitations:
  - Only on a single machine.
  - Complex implementation and furthermore very complex tests.
- Open Questions:
  - What about failure scenarios in particular hardware failures ?
    - Reliability?

# 1. Introduction

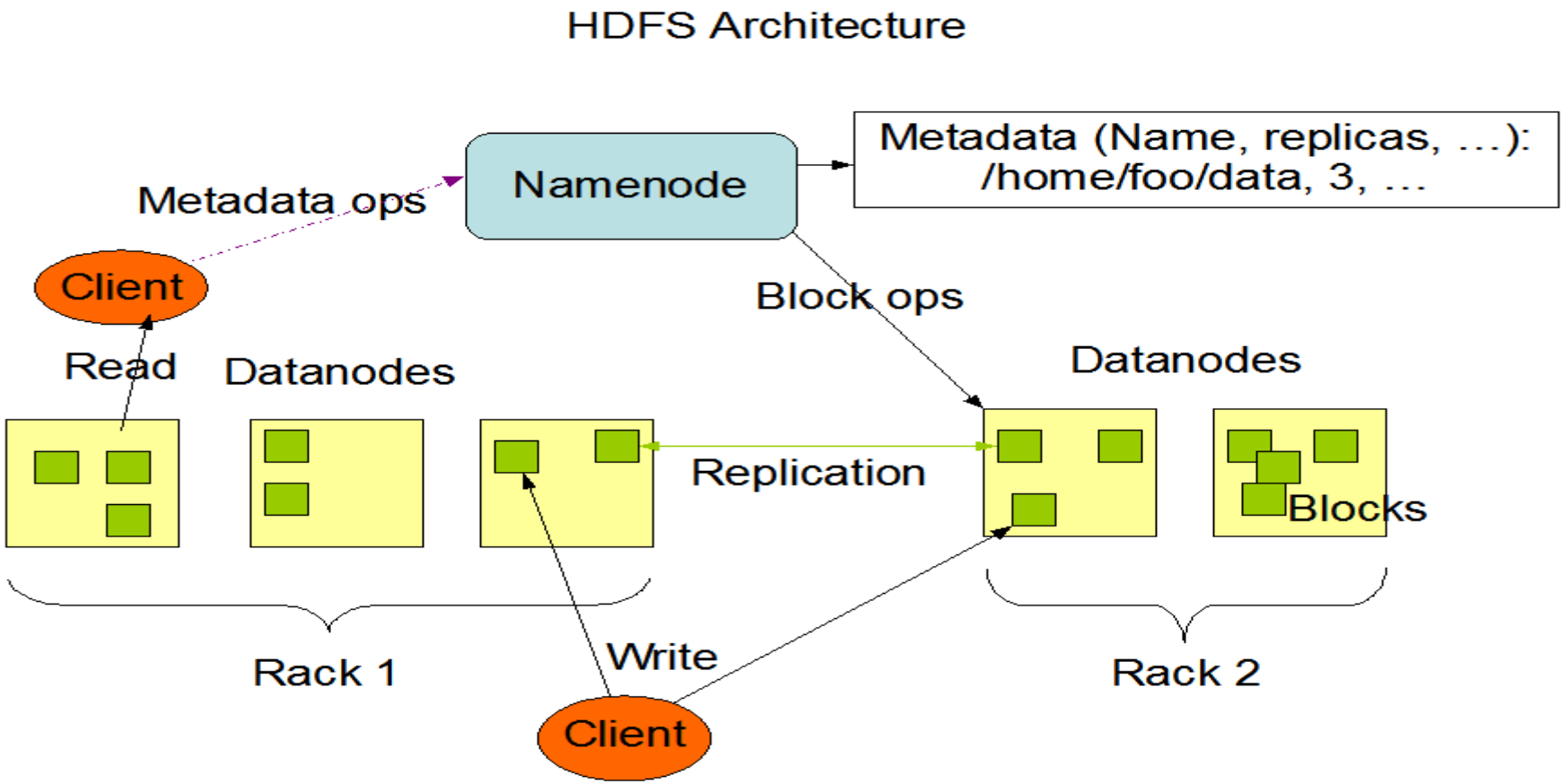
- Solution overview:



# 1. Introduction



# 2. Hadoop Distributes File System Architecture




http://hadoop.apache.org/hdfs/docs/current/hdfs_design.html

# 2. Hadoop Distributes File System Architecture

- Replication of individual files on different data notes.
  - The default replication factor is 3.
- Failures of machines
  - Hardware is not reliable.
- Activation of machines
  - Enhancement of the cluster.
- Deactivation of machines
  - Removing machines from your cluster.

# 2. Hadoop Distributes File System Architecture

- Name Node
  - Operations are written to logs, makes restart possible
    - The file system is in read only mode during a restart.
    - A secondary Name-Node periodically should read these logs, to speed up availability for restart.
    - A restart of the whole cluster is needed if the secondary Name-Node takes over (host names have changed).
    - Single point of failure and restart can take a long time (may be hours depending on the size of data stored in HDFS).

# 2. **H**adoop **D**istributes **F**ile **S**ystem Architecture

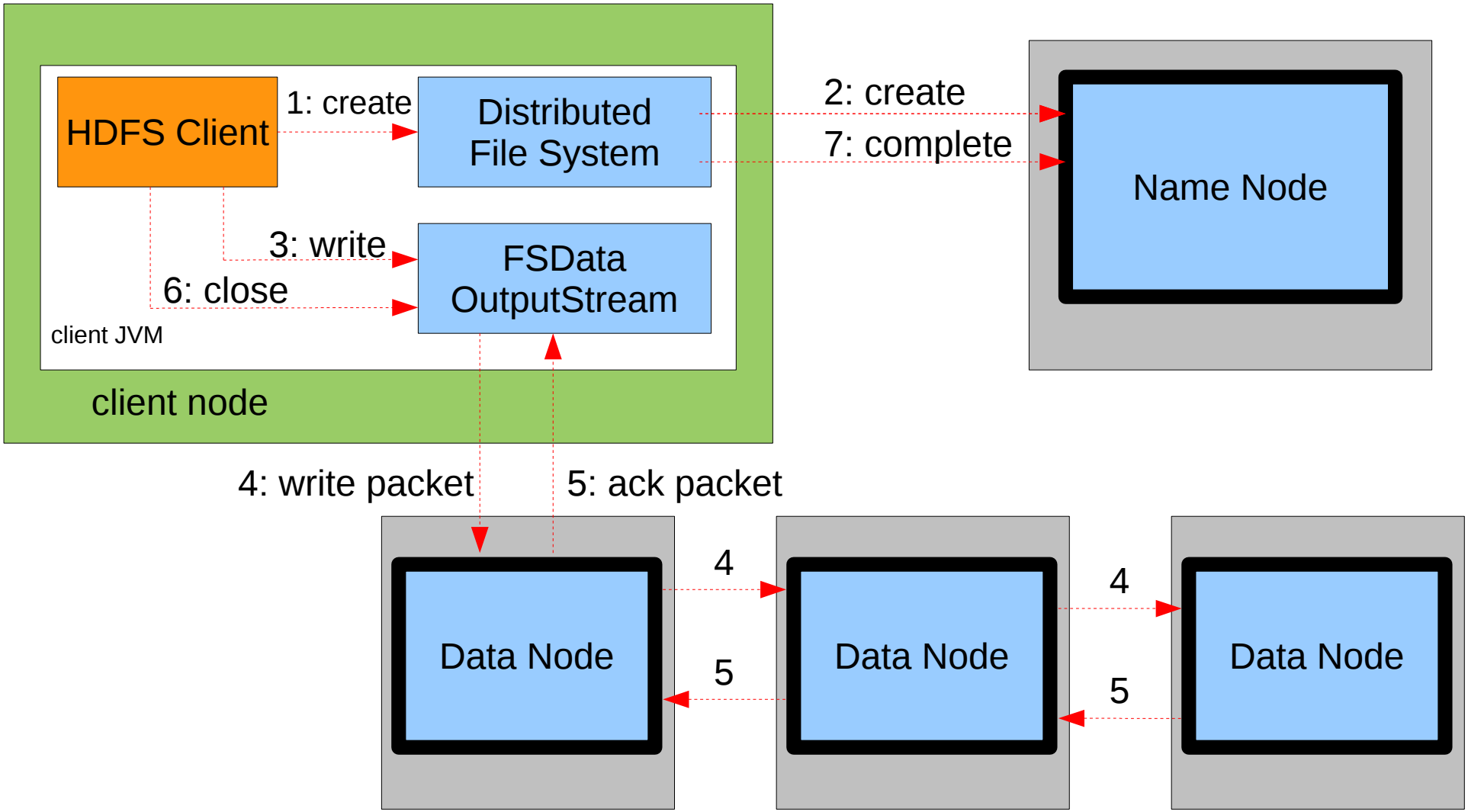
- Data Node
  - No identity.
  - Share nothing paradigm.
  - Commodity hardware.

# 2. **H**adoop **D**istributes **F**ile **S**ystem Architecture

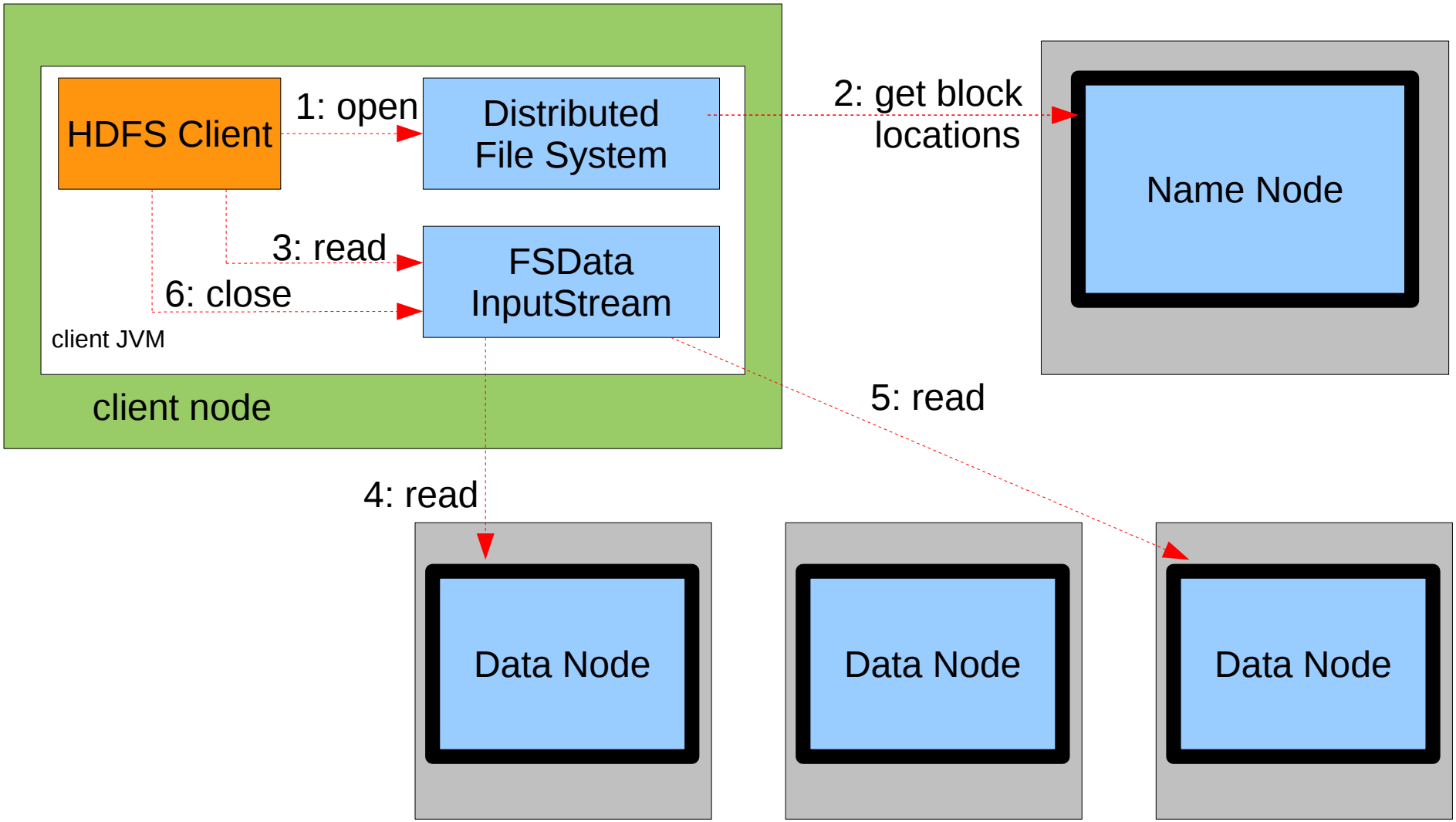
- Now we have a file system which can store files distributed over different machines.
  - This means you can have files which are larger than the storage in a single machine.

  - Pluggable files system
  - Access through API layer.
    - Default block size 64 MiB

# 2. Hadoop Distributes File System Architecture File Write

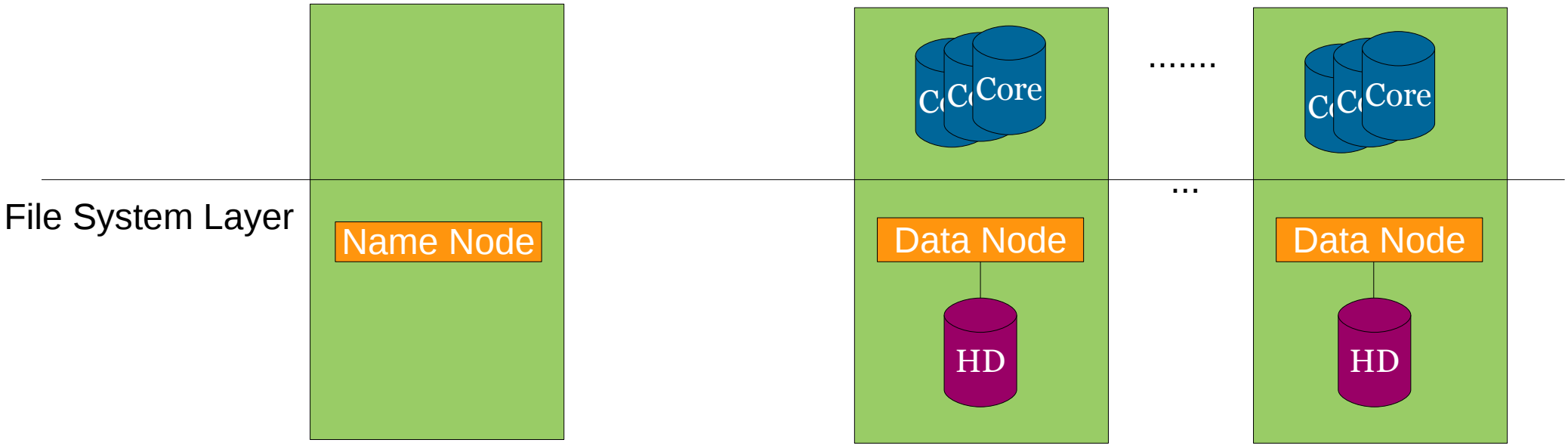


# 2. Hadoop Distributes File System Architecture File Read



# 2. Hadoop Distributes File System Architecture

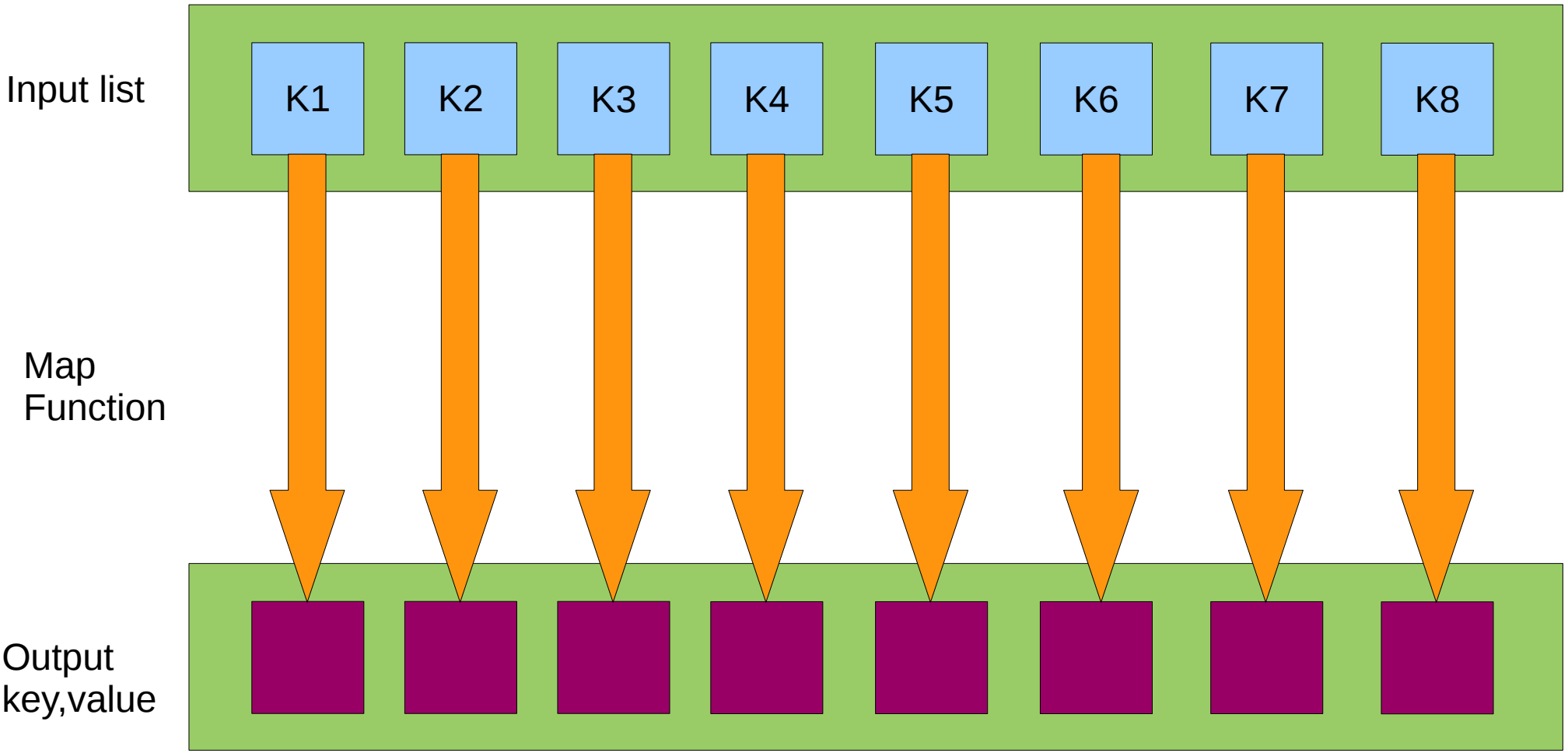
- What about the different cores of the Data Nodes?



# 3. Map/Reduce Mapper Theory

- The first phase of a Map/Reduce program is called the Map-Phase. A list of data elements are provided, one at a time, to a function called the Mapper, which transforms each element individually to an output data element.

# 3. Map/Reduce Mapper Theory



# 3. Map/Reduce Mapper Theory

- A typical example of the map function:

```
mapper (filename, file-contents):
  for each word in file-contents:
    emit (word, 1)
```

# 3. Map/Reduce Mapper Practice

- The Mapper of Hadoop:

```
public interface Mapper<K1, V1, K2, V2> extends
      JobConfigurable, Closeable {

    void map(
       K1 key, V1 value,
       OutputCollector<K2, V2> output,
       Reporter reporter
    ) throws IOException;
}
```

Example for Hadoop 0.20.0 API

# 3. Map/Reduce Mapper Practice

- The Mapper of Hadoop:

```
public class Mapper<KEYIN, VALUEIN, KEYOUT, VALUEOUT> {

    protected void map(KEYIN key, VALUEIN value, Context context)
       throws IOException, InterruptedException {

       context.write((KEYOUT) key, (VALUEOUT) value);

    }
}
```

Example for Hadoop 0.21.0 API

# 3. Map/Reduce Mapper Practice

- Let's start with an example in Java Code:

```
public class WordCountMap 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);
      }
    }
}
```

Example for Hadoop 0.20.0 API

# 3. Map/Reduce Mapper Practice

- Let's start with an example in Java Code:

```
public class WordCountMapper extends
    Mapper<LongWritable, Text, Text, LongWritable> {

    private LongWritable one = new LongWritable(1);
    private Text word = new Text();

    public void map(LongWritable key, Text value, Context context)
      throws InterruptedException, IOException {

        StringTokenizer token = new
              StringTokenizer(value.toString(), ".,-! \t\n\r\f");
        while(token.hasMoreTokens()) {
            word.set(token.nextToken());
            context.write(word, one);
           }
       }
}
```

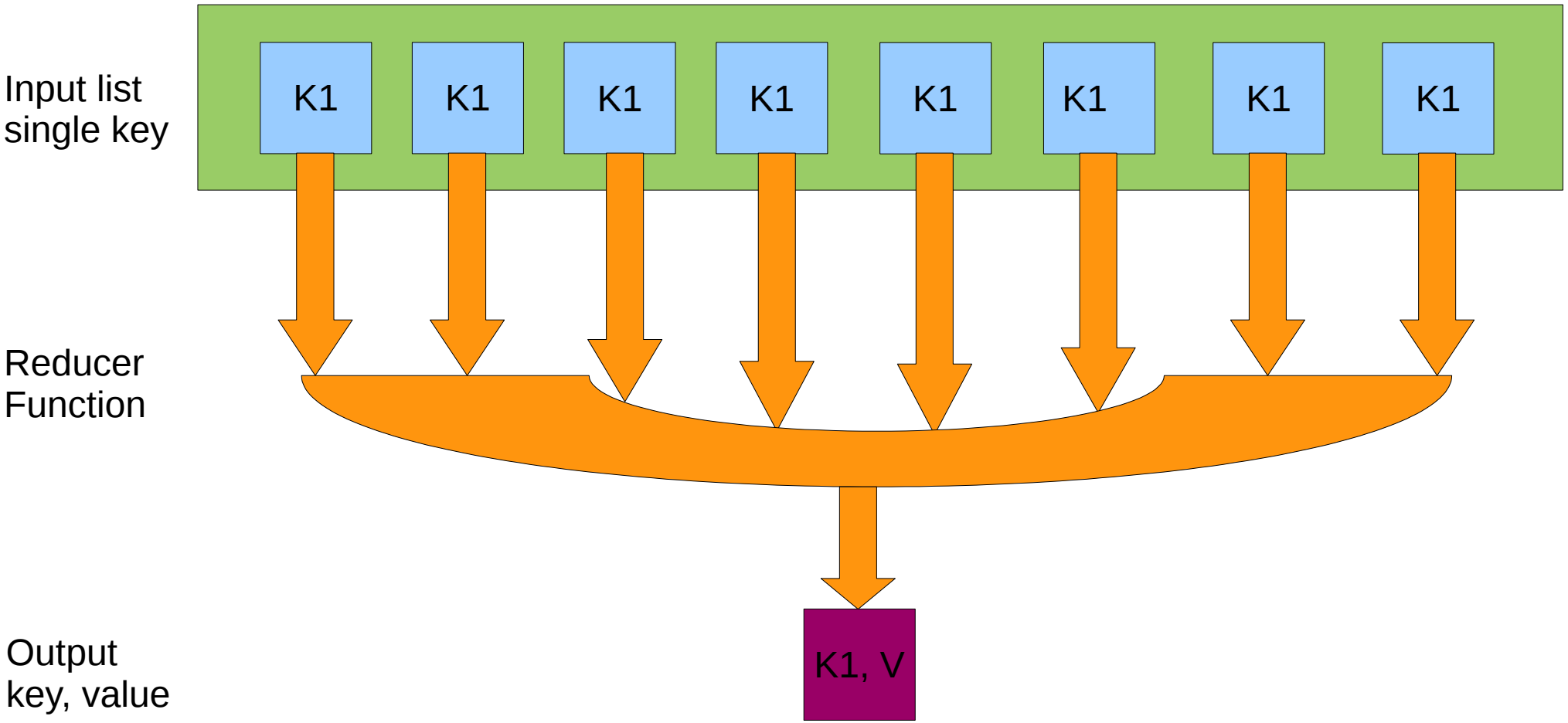
Example for Hadoop 0.21.0 API

# 3. Map/Reduce Mapper Practice

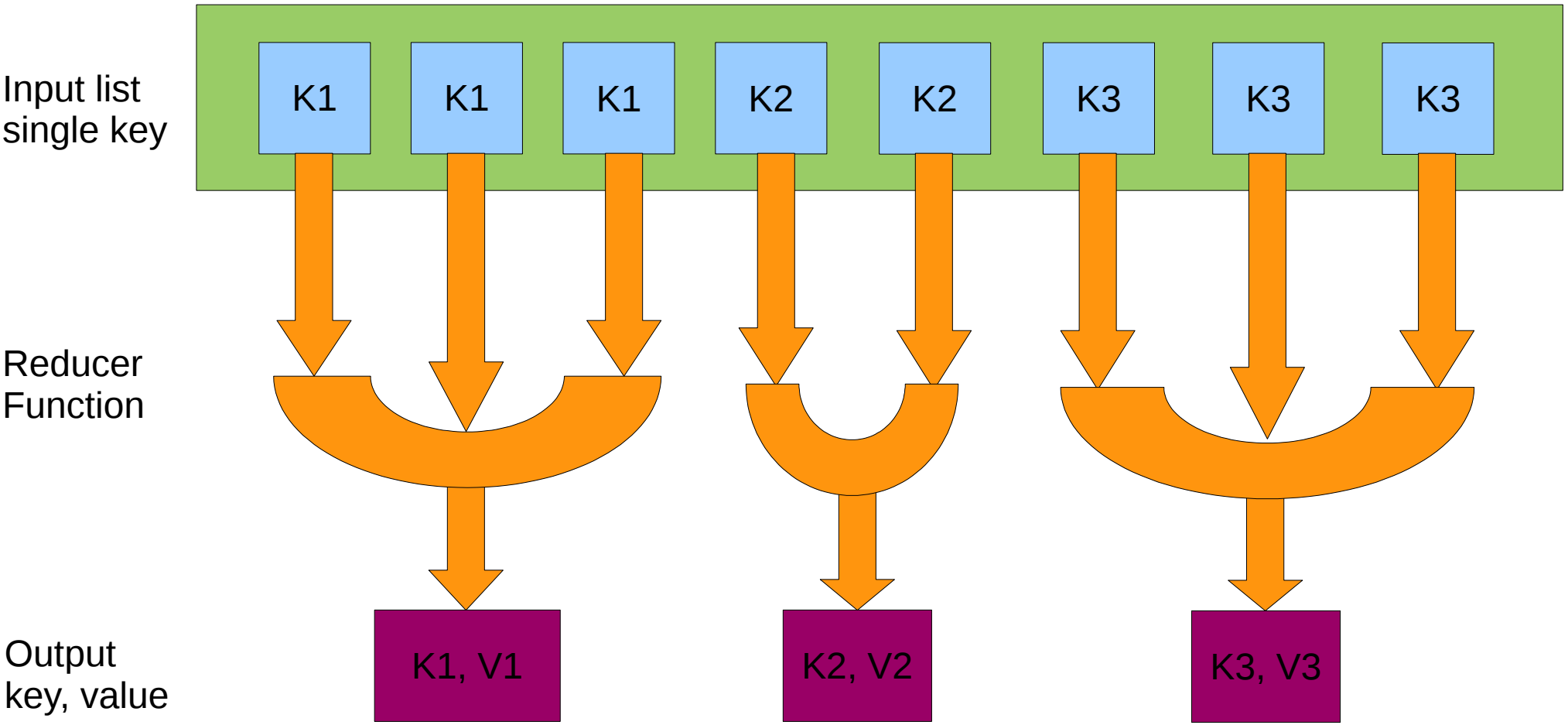
- Output of the mapper:

**this, 1**
**there, 1**
**goto, 1**
**help, 1**
**test, 1**
**test, 1**
**there, 1**
**those, 1**
**there, 1**
**this, 1**
**...**

# 3. Map/Reduce Reducer Theory



# 3. Map/Reduce Reducer Theory



# 3. Map/Reduce Reducer Theory

- A typical example of the reducer function:

```
reducer (word, values):
    sum = 0
    for each value in values:
        sum = sum + value
    emit (word, sum)
```

# 3. Map/Reduce Reducer Practice

```
public static class WordCountReduce extends MapReduceBase
    implements Reducer<Text, LongWritable, Text, LongWritable> {

    public void reduce(
        Text key, Iterator<LongWritable> values,
        OutputCollector<Text, LongWritable> output,
        Reporter reporter) throws IOException {

        int sum = 0;
        while (values.hasNext()) {
           sum += values.next().get();
        }
        output.collect(key, new LongWritable(sum));
    }
}
```

Example for Hadoop 0.20.0 API

# 3. Map/Reduce Reducer Practice

```
public class WordCountReducer extends
    Reducer<Text, LongWritable, Text, LongWritable> {

    @Override
    public void reduce(Text key, Iterable<LongWritable> values,
        Context context)
        throws InterruptedException, IOException {

        long summ = 0;
        for (LongWritable value : values) {
            summ += value.get();
        }
        context.write(key, new LongWritable(summ));
    }
}
```

Example for Hadoop 0.21.0 API

# 3. Map/Reduce Reducer Practice

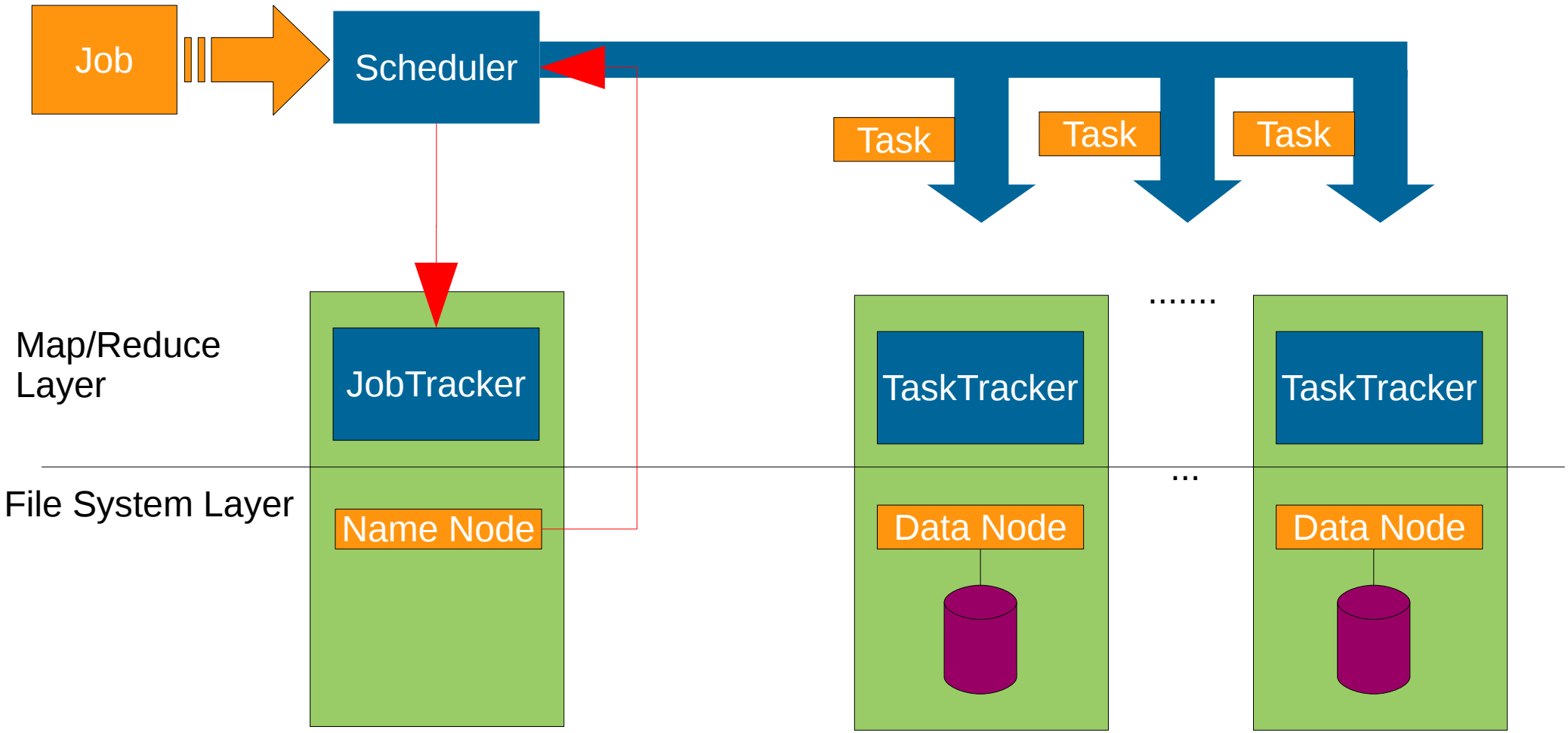
- Output of the mapper:

**this, 1**
**there, 1**
**goto, 1**
**help, 1**
**test, 1**
**test, 1**
**there, 1**
**those, 1**
**there, 1**
**this, 1**
**…**

- Output of the reducer:

**this, 2**
**there, 3**
**goto, 1**
**help, 1**
**test, 2**
**…**

# 4. Map/Reduce and HDFS



# 4. Map/Reduce and HDFS

```
public class WordCountRunner extends Configured implements Tool {
    @Override
    public int run(String[] args) throws Exception {
        Configuration conf = new Configuration();
        String[] otherArgs = new GenericOptionsParser(conf, args).getRemainingArgs();
        ...
        Job job = new Job(conf, "WordCount");
        job.setJarByClass(WordCountRunner.class);
        job.setMapperClass(WordCountMapper.class);
        job.setReducerClass(WordCountReducer.class);
        job.setOutputKeyClass(Text.class);
        job.setOutputValueClass(LongWritable.class);
        job.setMapOutputKeyClass(Text.class);
        job.setMapOutputValueClass(LongWritable.class);
        FileInputFormat.addInputPath(job, new Path(otherArgs[0]));
        FileOutputFormat.setOutputPath(job, new Path(otherArgs[1]));
        System.exit(job.waitForCompletion(true) ? 0 : 1);
        return 0;
    }
    public static void main(String[] args) throws Exception {
      int res = ToolRunner.run(new Configuration(), new WordCountRunner(), args);
      System.exit(res);
    }
}
```

# 4. Map/Reduce and HDFS Start a Job

- Start a Map/Reduce job:

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

# 5. **H**adoop **D**istributes **F**ile **S**ystem Installation Requirements

- GNU/Linux is supported as a development and production platform. Hadoop has been demonstrated on GNU/Linux clusters with 2000 nodes.
- Win32 is supported as a development platform. Distributed operation has not been well tested on Win32, so it is **not supported as a production platform**.

# 5. Hadoop Distributes File System Installation Requirements

- Java™ 1.6.x, preferably from Sun, must be installed.
- ssh must be installed and sshd must be running to use the Hadoop scripts that manage remote Hadoop daemons.
- **ssh localhost** via public key must be working.
- On the different machines a user called hadoop (obviously ;-) should be created and accessible via ssh from the name node.

# 5. Map/Reduce and HDFS Demo

DEMO

# 6. Real World: Facebook

- 21 PiB (30 PiB Nov. 2010)
- 2000 machines (3000 in Nov. 2010)
- 12 TiB per machine (a few have 24 TiB each)
- 1200 machines with 8 cores each
- 800 machines with 16 cores each
- 32 GiB RAM per machine

http://hadoopblog.blogspot.com/2010/05/facebook-has-worlds-largest-hadoop.html
http://developer.yahoo.com/blogs/hadoop/posts/2010/05/scalability_of_the_hadoop_dist/

# 6. Real World: Facebook

- 12 TiB of compressed data added per day
- 800 TiB of compressed data scanned per day (1 PiB per day Nov. 2010)
- 25,000 map-reduce jobs per day
- 65 millions files in HDFS
- 30,000 simultaneous clients to the HDFS NameNode

http://hadoopblog.blogspot.com/2010/05/facebook-has-worlds-largest-hadoop.html
http://developer.yahoo.com/blogs/hadoop/posts/2010/05/scalability_of_the_hadoop_dist/

# 6. Real World

- Take a look here:
  - http://wiki.apache.org/hadoop/PoweredBy
  - Hadoop World NYC

# 7. Conclusion

- HDFS is a file system write-once read multiple times (appending is also possible 0.21.0 API).
- Hadoop is not a replacement for RDBMS.
- It is not simple to write Map/Reduce jobs
  - May be take a look at Hive (DWH SQL), HTable, Pig etc.
- If a node fails not the whole cluster will fail only the throughput will reduce by a percentage rate.

# 7. Conclusion

- What is the size to start with?
  - You can start with 3(Data nodes)+1(Name Node) Nodes and continue to enhanced the size based on your requirements.
  - Or
    - Just rent a Hadoop cluster => Amazon....

# Appendix
# References

- Hadoop Homepage
  - http://hadoop.apache.org
- HDFS Homepage
  - http://hadoop.apache.org/hdfs
- Map/Reduce
  - http://hadoop.apache.org/mapreduce
- Google Map/Reduce
  - http://labs.google.com/papers/mapreduce.html

# Appendix
# References

- Hive
  - http://hive.apache.org/
- Pig
  - http://pig.apache.org/
- HBase
  - http://hbase.apache.org/
- Cloudera (Map/Reduce, Hadoop etc.)
  - http://www.cloudera.com/resources/?type=Training

# Appendix
# References

- Yahoo Developer Resources
  - http://developer.yahoo.com/hadoop/tutorial/index.html
- Yahoo M/R
  - http://developer.yahoo.com/hadoop/tutorial/module4.htr
- Videos about M/R
  - http://vimeo.com/3584536