Hadoop for HPCers:
A Hands-On Introduction

Jonathan Dursi, SciNet
Michael Nolta, CITA

Part I: Overview, MapReduce

Agenda

- VM Test
- High Level Overview
- Hadoop FS
- Map Reduce
- Hadoop MR + Python
- Hadoop MR
- Break
- Hands On with Examples
  - Word count
  - Inverted Index
  - Document Similarity
  - Matrix Multiplication
  - Diffusion

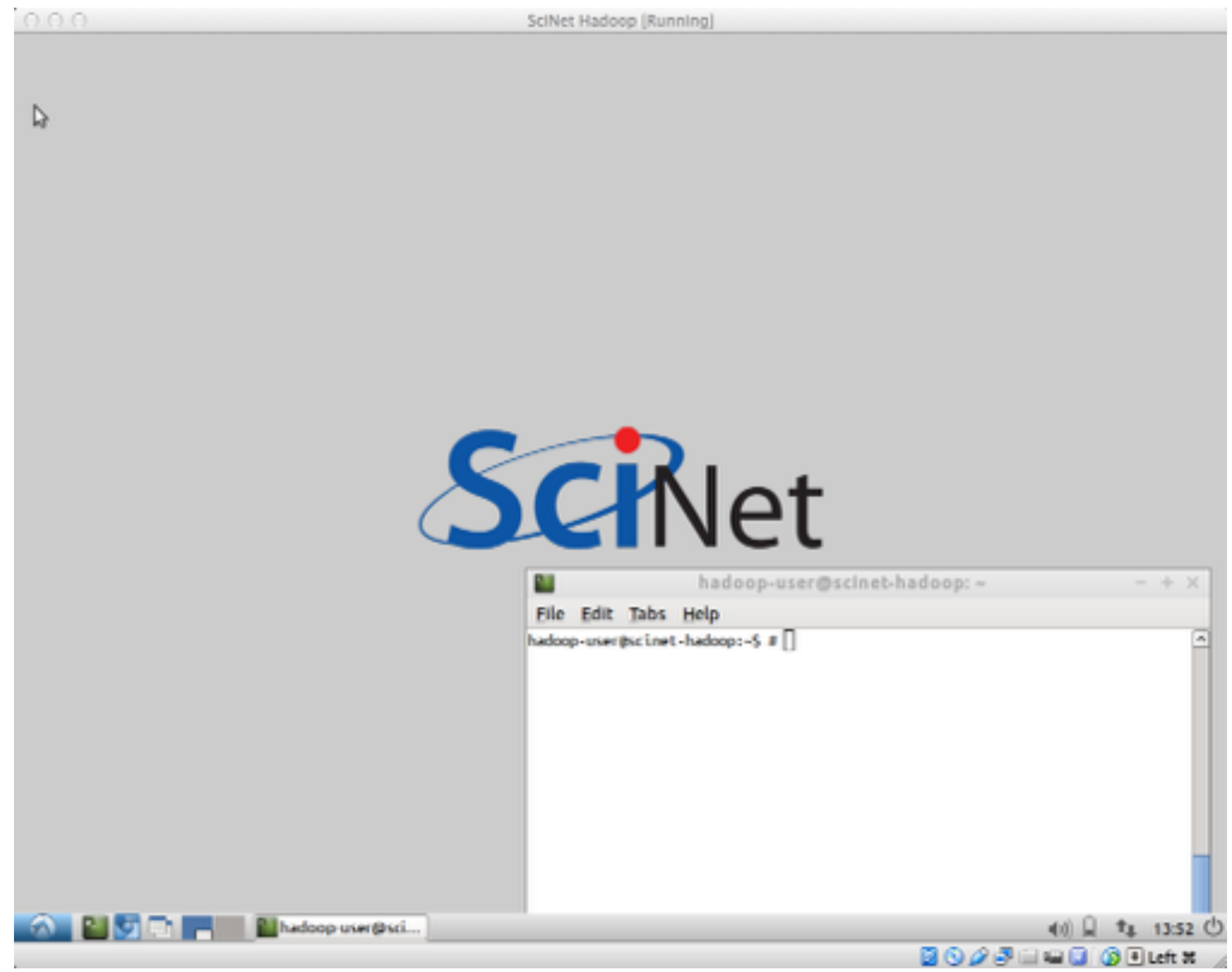
SciNet compute • calcul CANADA

Detailed VM instructions

- Install VirtualBox (free) for your system.
- Download and unzip the course VM from http://support.scinet.utoronto.ca/~ljdursi/SciNetHadoopVM.zip
- Start Virtual box; click "New"; give your VM a name. Select "Linux" as Type, and "Ubuntu" as Version. Give your VM at least 2048MB RAM, more would be better.
- Select "Use an existing virtual hard drive", and choose the .vdi file you downloaded. Click "Create".
- Before starting your VM, enable easy network access between the host and VM.
  - Go into the VirtualBox app preferences VirtualBox > Preferences > Network and, if one doesn't already exist, add a host-only network.
  - Select the new VM and click "Settings". Under "System", make sure "Enable IO APIC" is checked. Then under "Network", select "Adapter 2", Enable it, and attach it to "Host-only adapter". Click "OK". This will allow you to easily transfer files to and from your laptop and the virtual machine.
  - Also under "System", then "Processor", give your VM a couple of cores to play with; for safety, you might want to bring down the Execution cap to 50% or so.
- Start the VM; username is hadoop-user, password is hadoop.
- Open a terminal; run "source ./init.sh"

Let's Get Started!

- Fire up your course VM
- Open terminal;
```
source init.sh
cd wordcount
make
```
- You've run your (maybe) first Hadoop job!

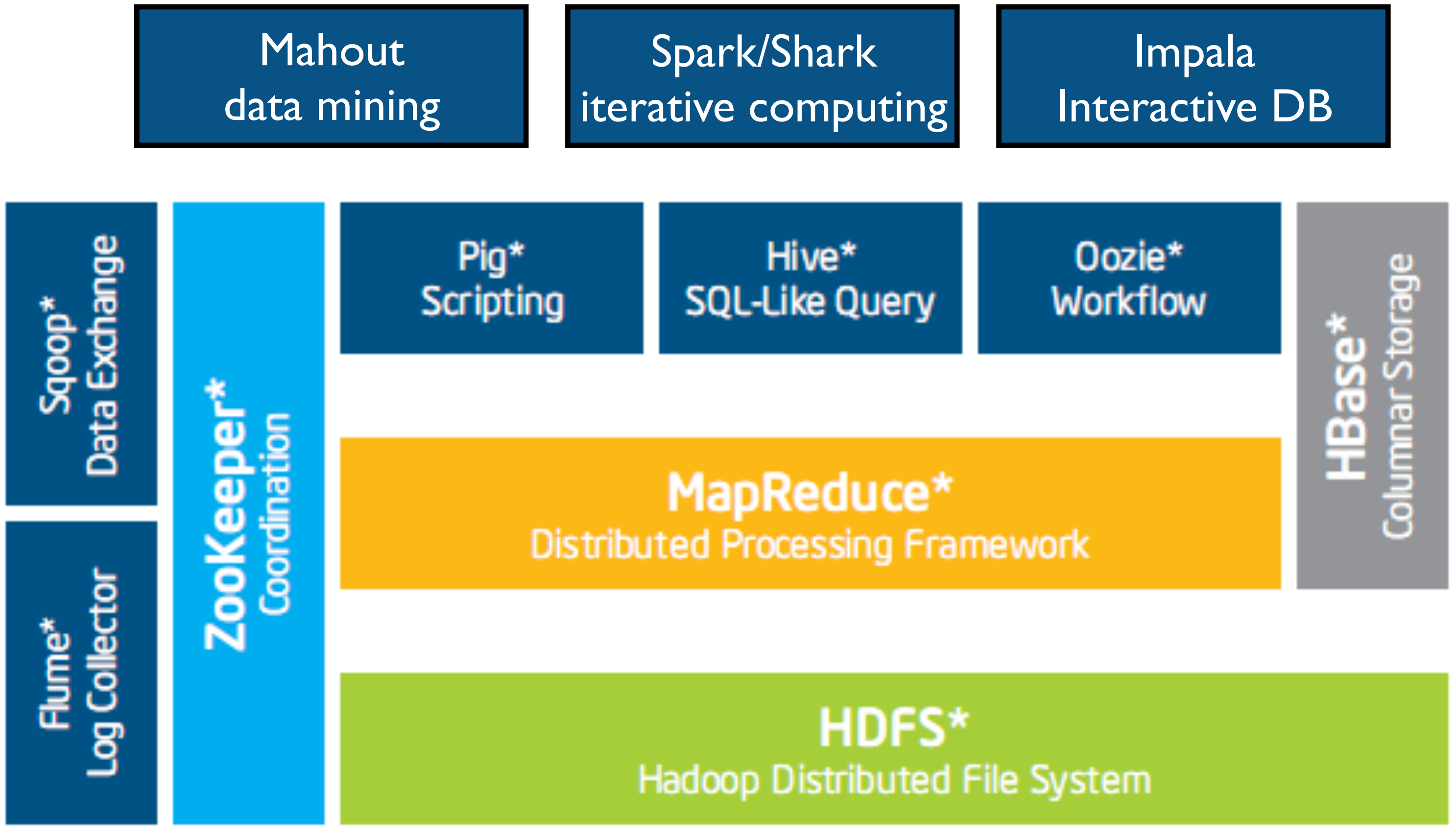


Hadoop

- 2007 OSS implementation of 2004 Google MapReduce paper
- Consists of distributed filesytem HDFS, core runtime, an implementation of Map-Reduce.
- Hardest to understand for HPCers: Java
- Pronounced "Hay-doop".

Hadoop Ecosystem

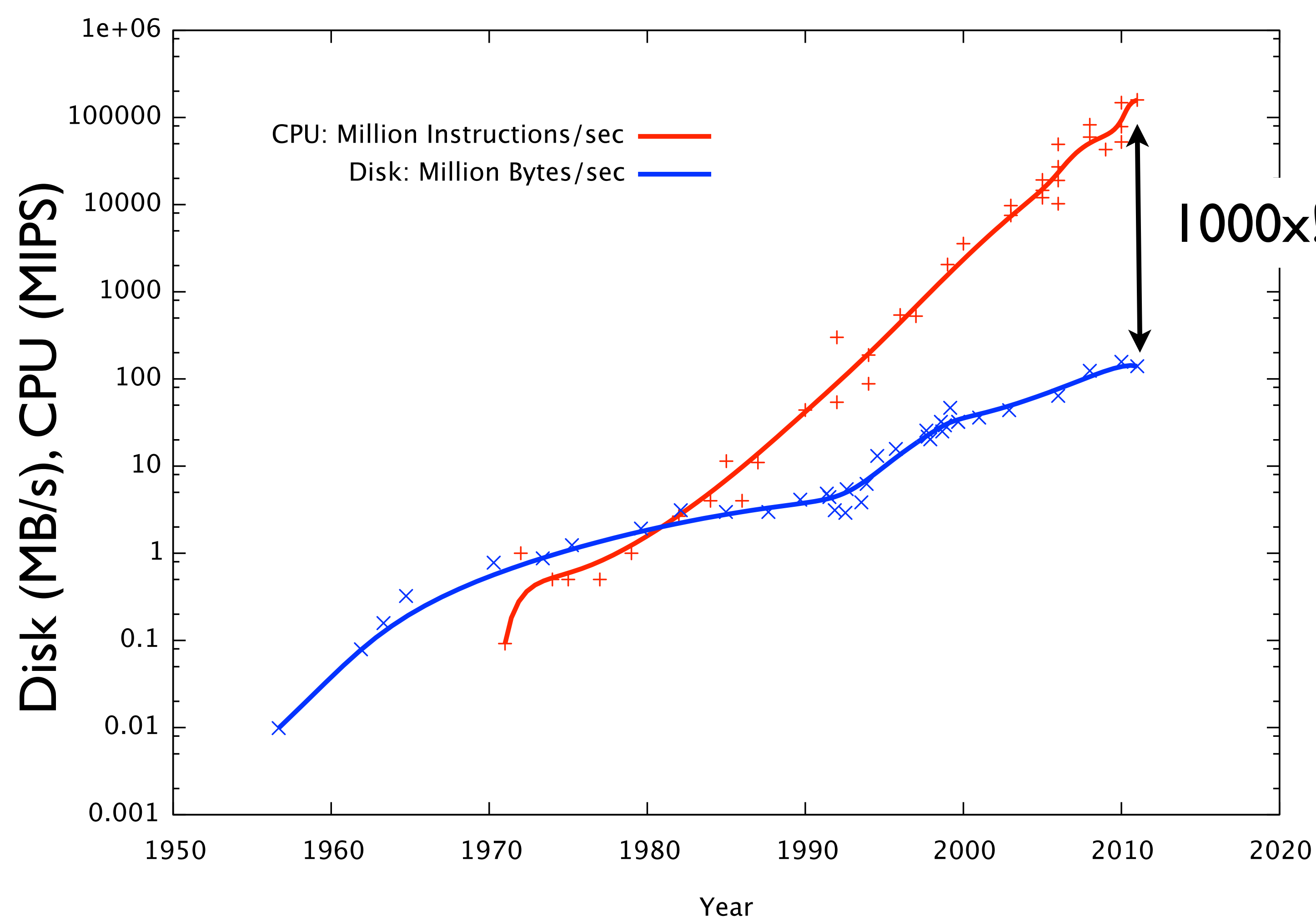
- 2008+ - usage exploded
- Creation of many tools building atop Hadoop infrastructure
- Met a real need



SciNet compute • calcul CANADA

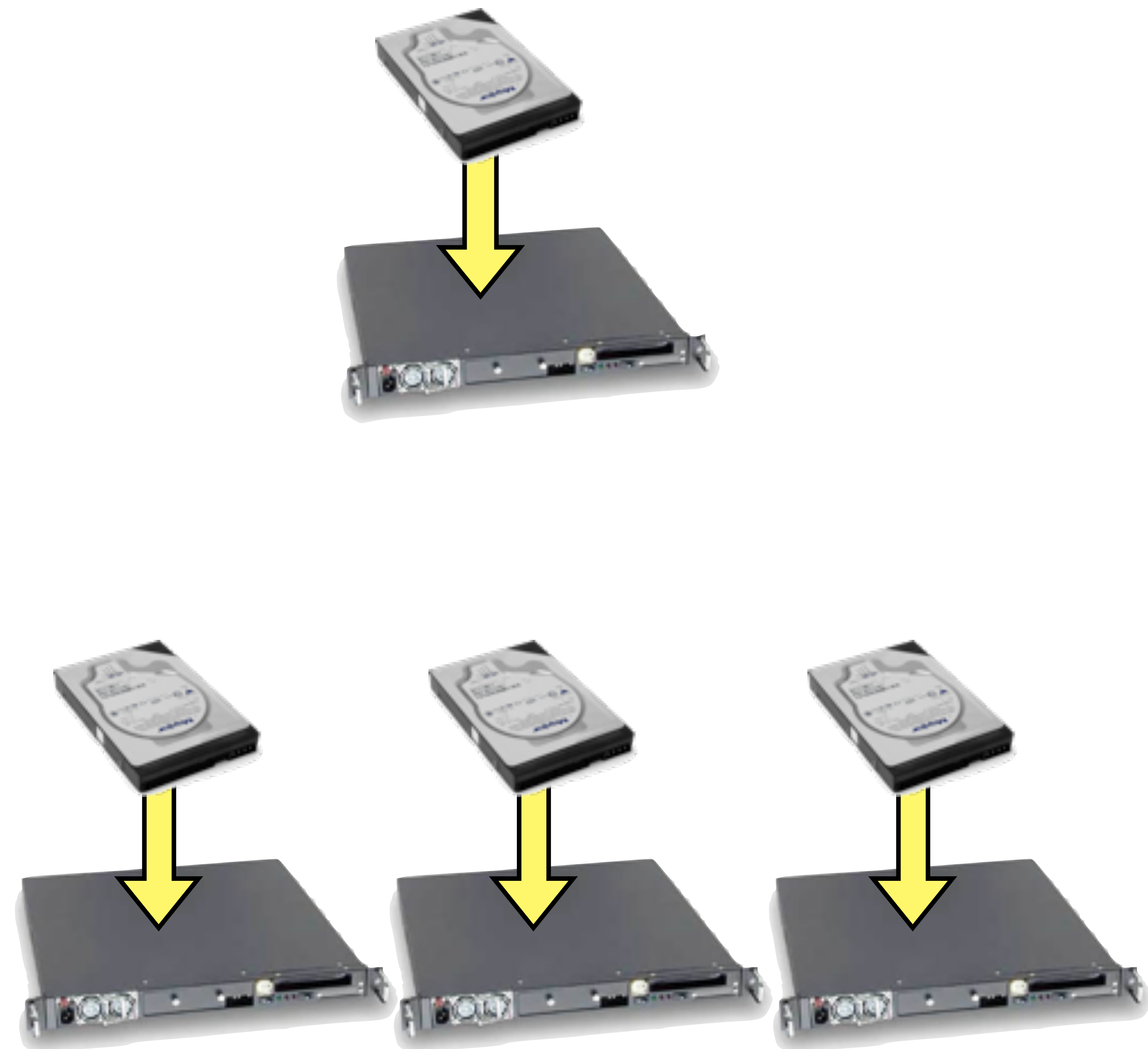
Data Intensive Computing

- Data volumes increasing massively
- Clusters, storage capacity increasing massively
- Disk speeds are not keeping pace.
- Seek speeds even worse than read/write



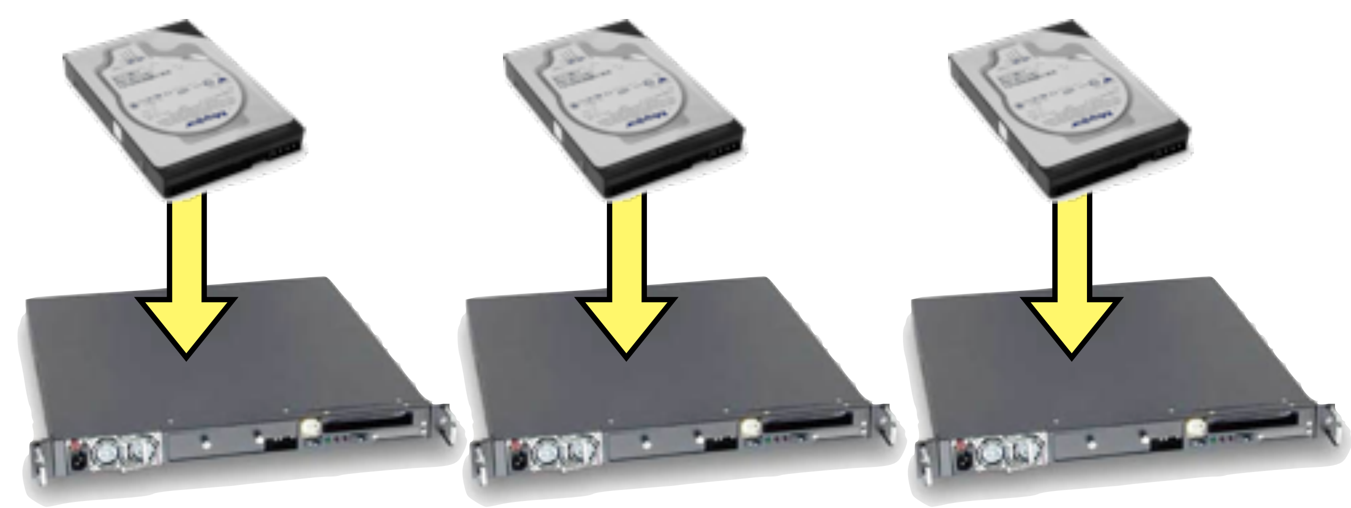
Scale-Out

- Disk streaming speed ~ 50MB/s
- 3TB =17.5 hrs
- 1PB = 8 months
- Scale-out (weak scaling) - filesystem distributes data on ingest



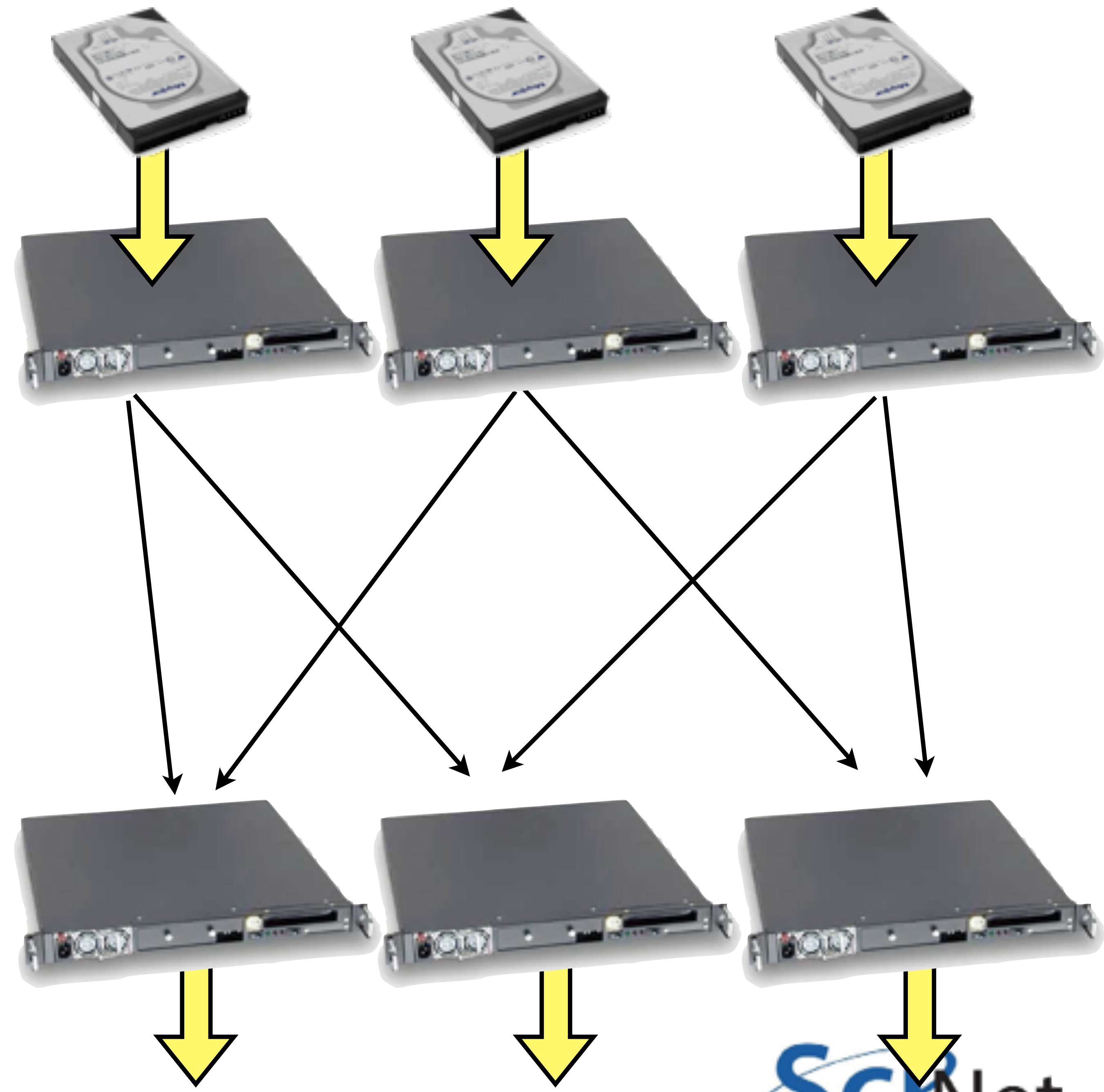
Scale-Out

- Seeking too slow
- Batch processing
- Pass through entire data set in one (or small number) of passes



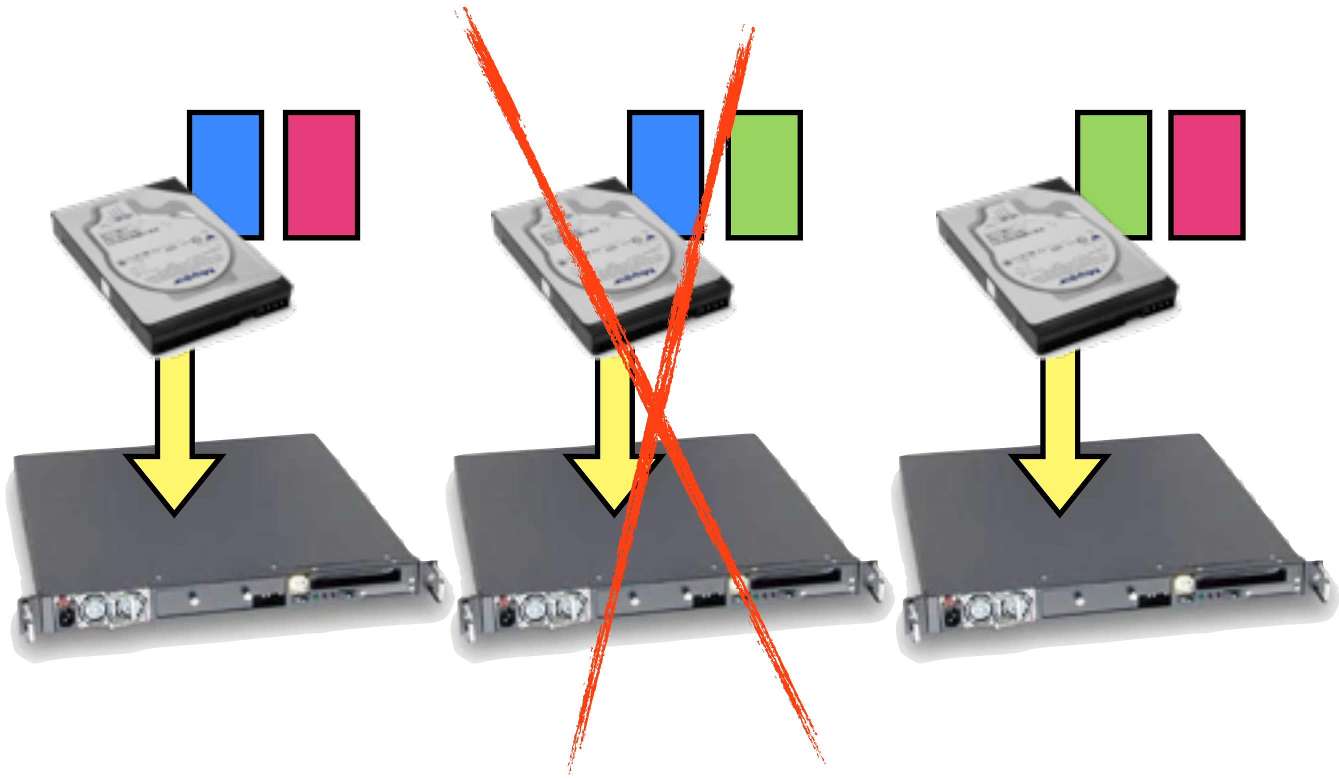
Combining results

- Each node pre-processes its local data
- Shuffles its data to a small number of other nodes
- Final processing, output is done there



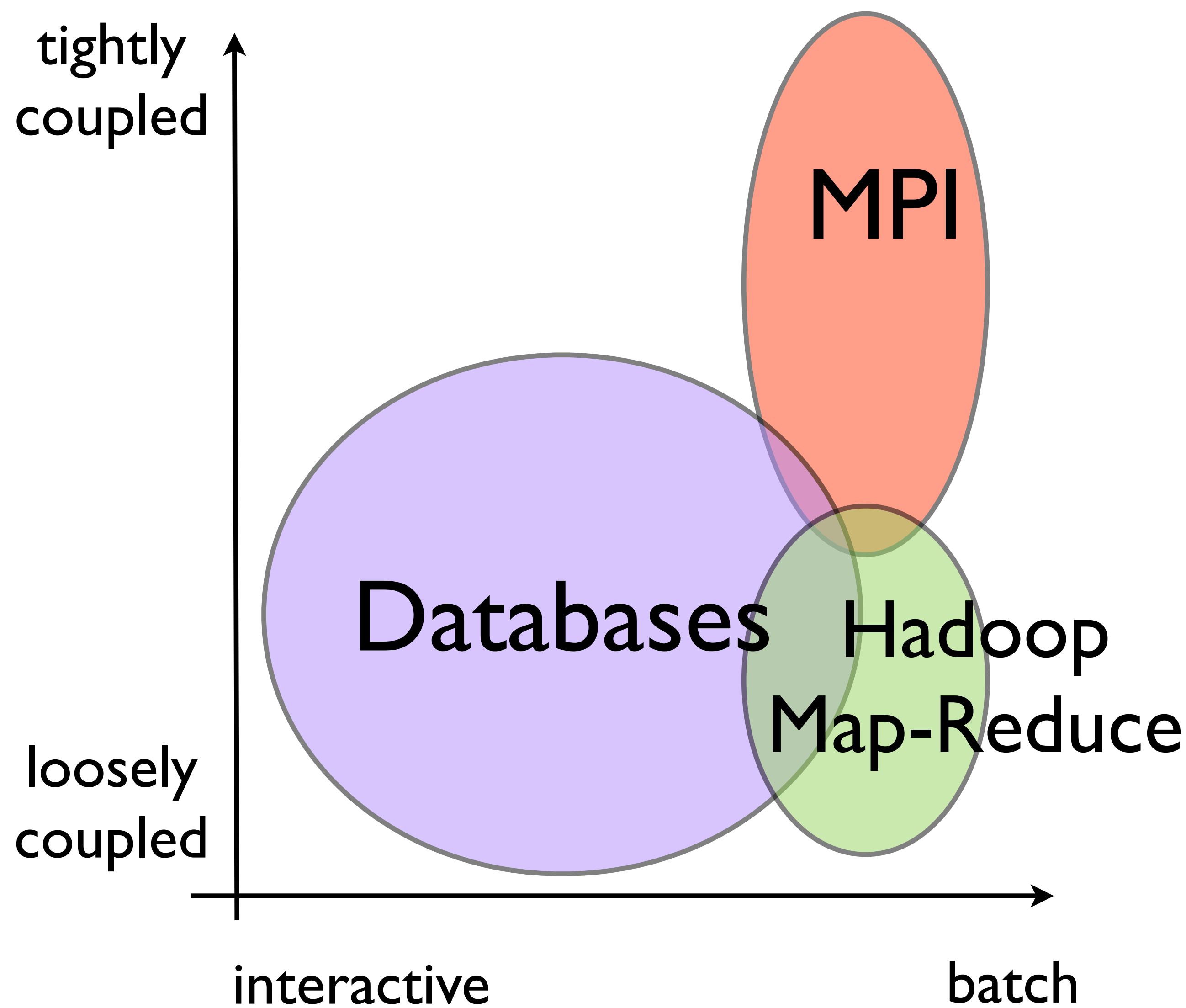
Fault Tolerance

- Data also replicated upon ingest
- Runtime watches for dead tasks, restarts them on live nodes
- Re-replicates



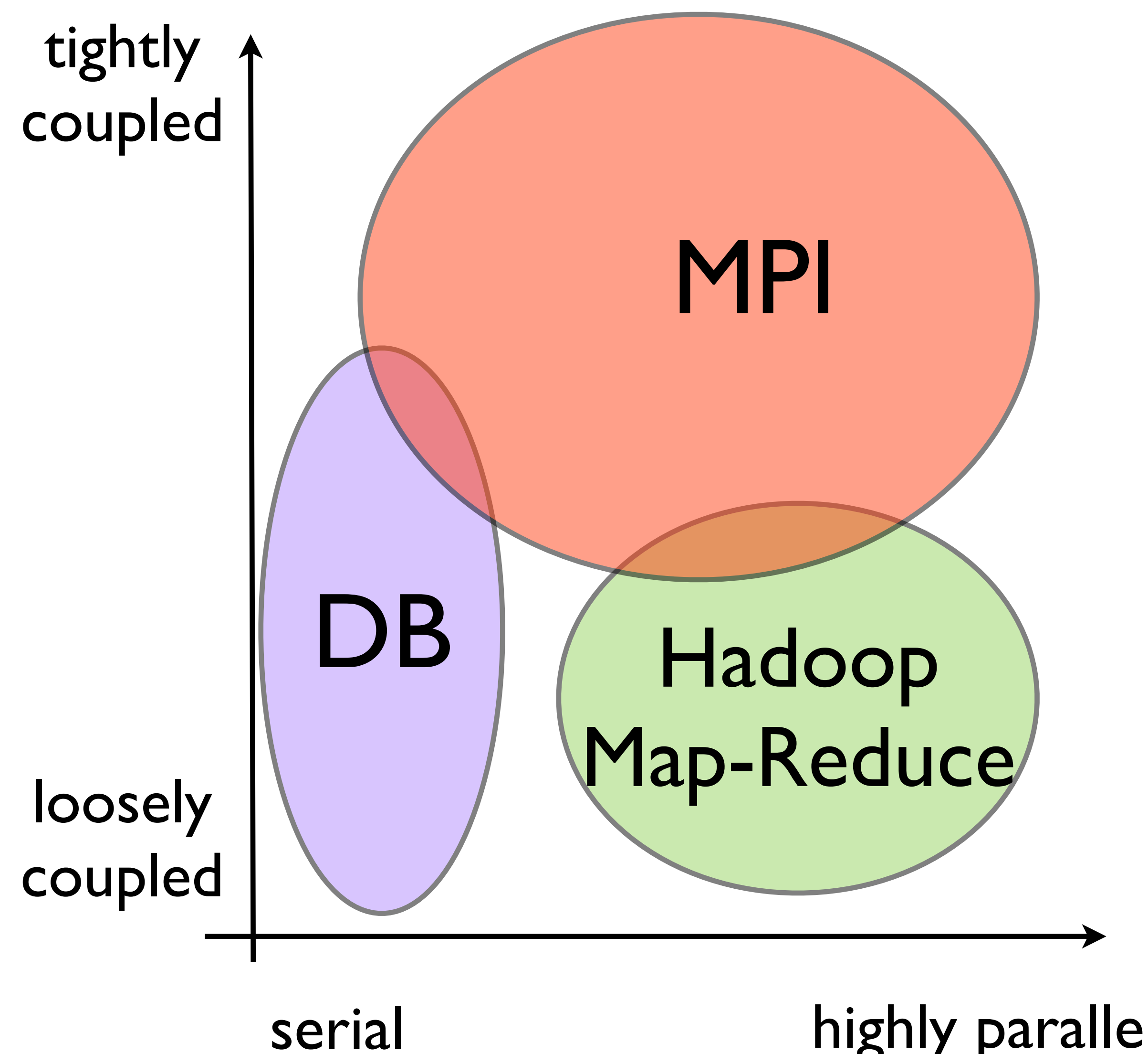
What is it good at?

- “Classic” Hadoop is all about batch processing of massive amounts of data
- (Not much point below ~1TB)
- Map-Reduce is relatively loosely coupled; one “shuffle” phase.
- Very strong weak scaling in this model - more data, more nodes.
- Batch: process all data in one go w/ classic Map Reduce
- (New Hadoop has many other capabilities besides batch)



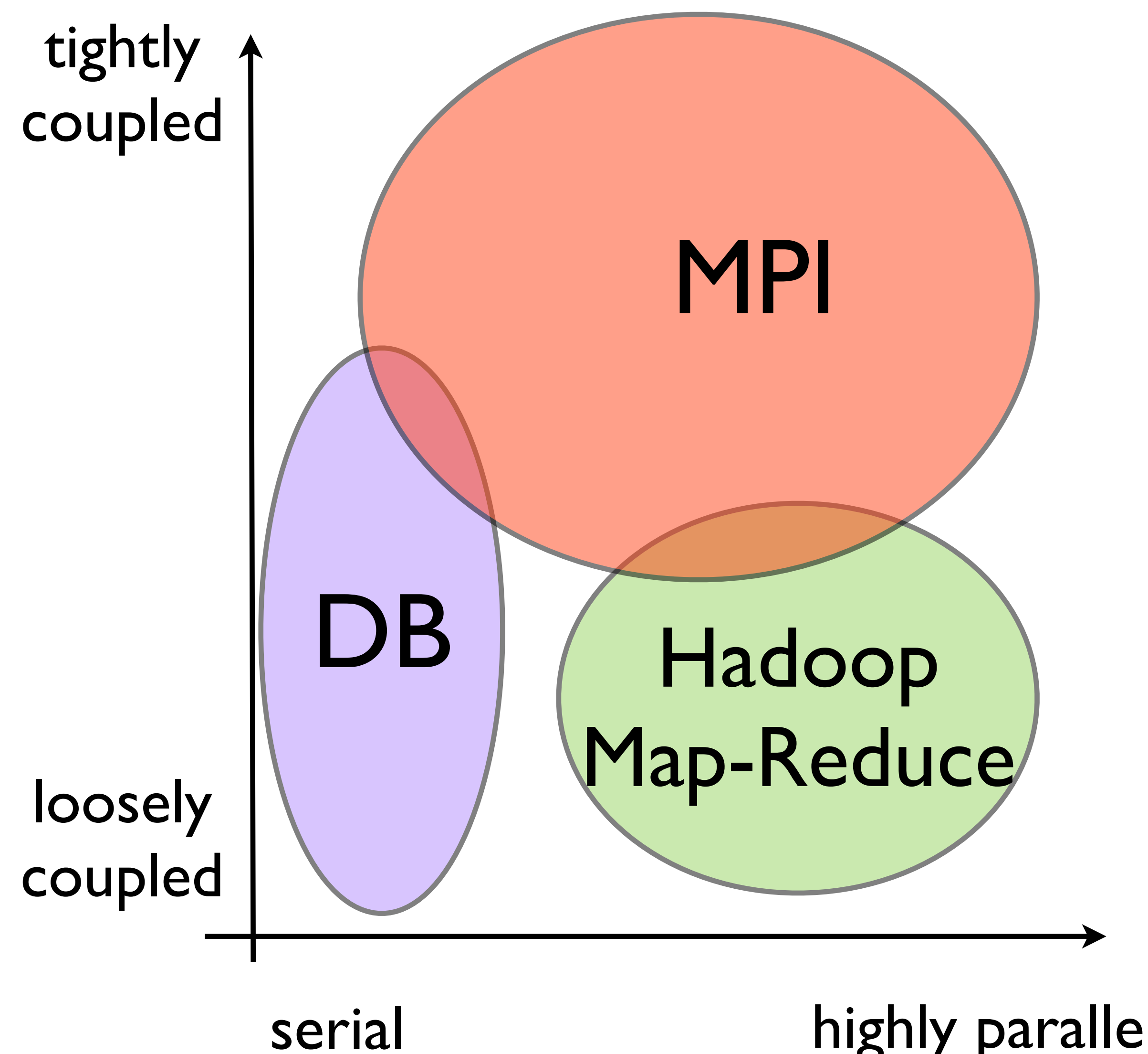
What is it good at?

- Compare with databases - very good at working on small subsets of large databases
- DBs - very interactive for many tasks
- DBs - have been difficult to scale



What is it good at?

- Compare with HPC (MPI)
- Also typically batch
- Can (and does) go up to enormous scales
- Works extremely well for very tightly coupled problems: zillions of iterations/ timesteps/exchanges.

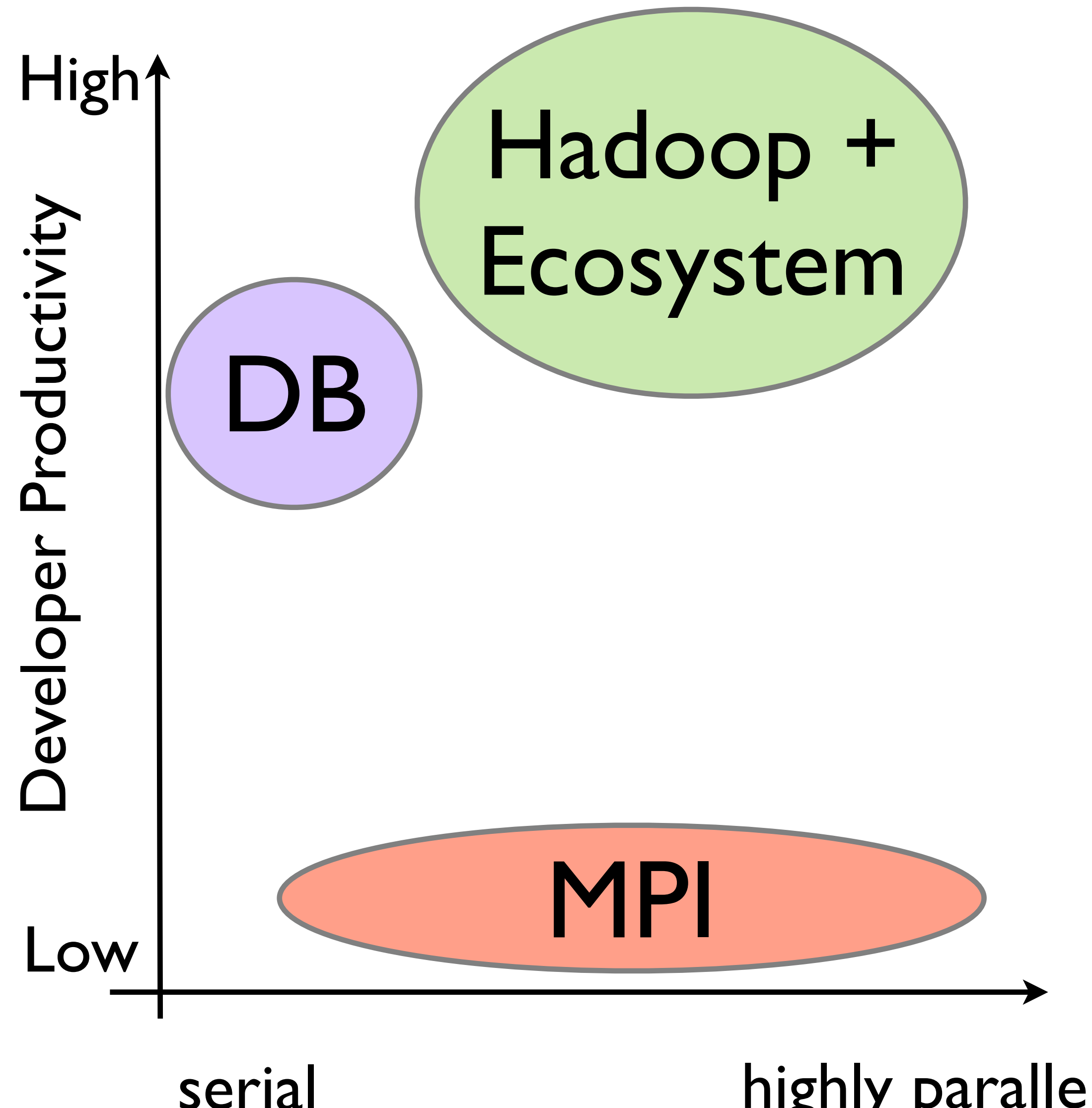


Hadoop vs HPC

- We HPCers might be tempted to an unseemly smugness.
- "They solved the problem of disk-limited, loosely-coupled, data analysis by throwing more disks at it and weak scaling? Ooooooooh."

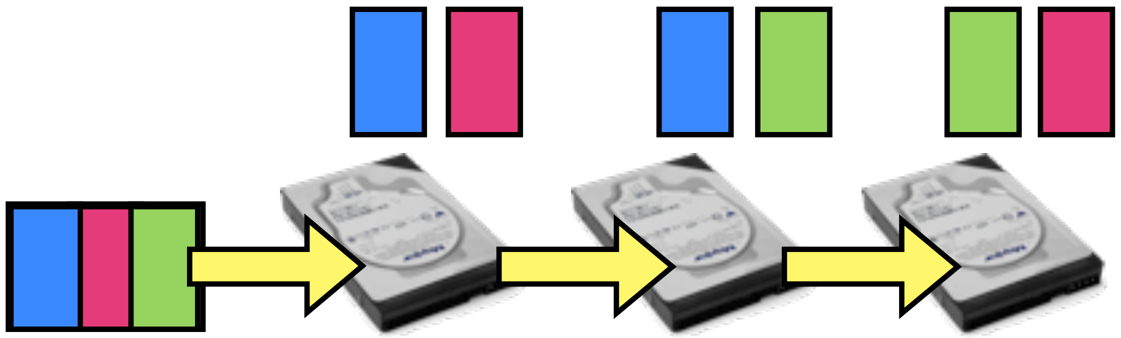
Hadoop vs HPC

- We'd be wrong.
- A single novice developer can write real, scalable, 1000+ node data-processing tasks in Hadoop-family tools in an afternoon.
- MPI... less so.



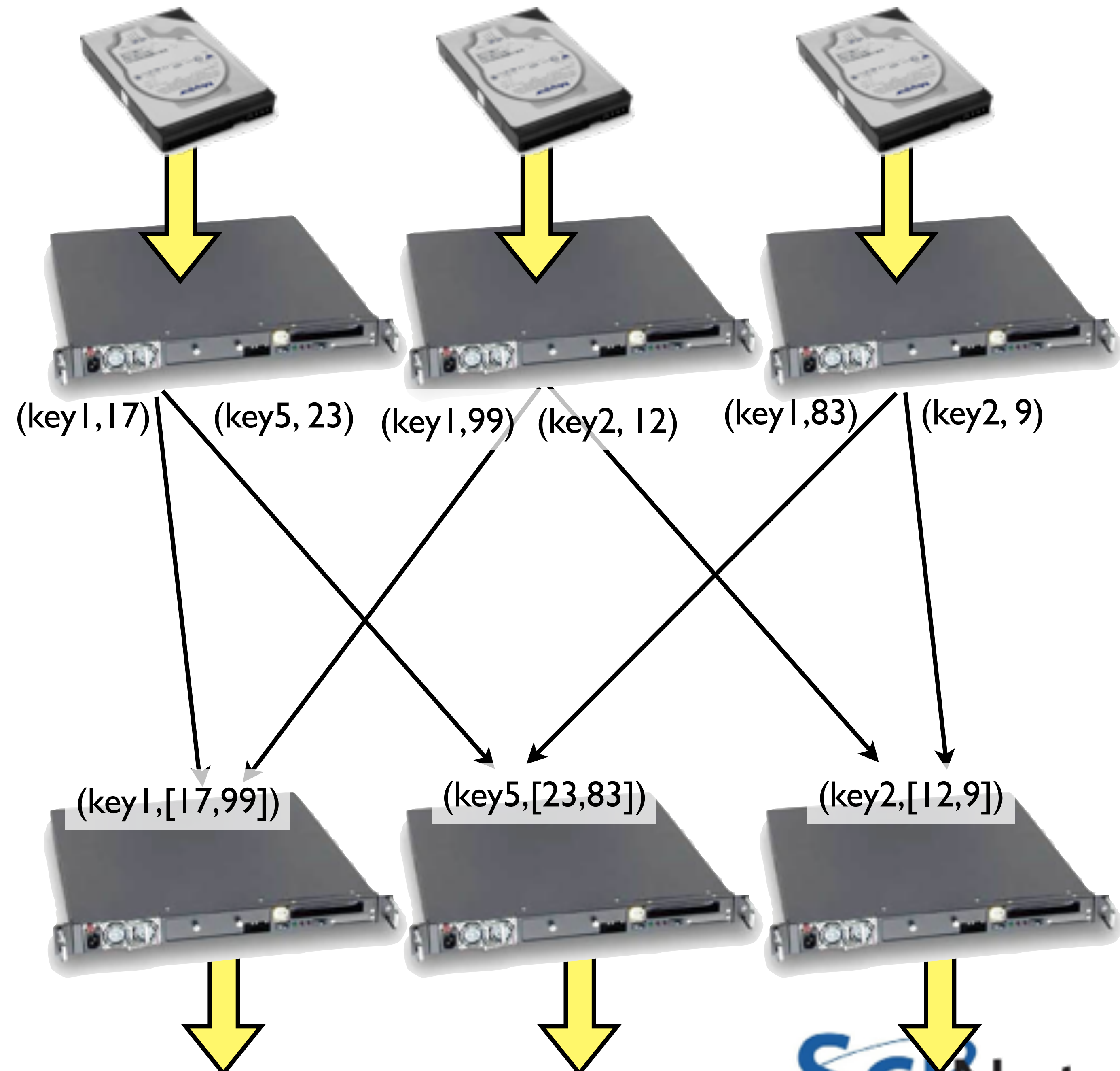
Data Distribution: Disk

- Hadoop and similar architectures handle one hard part of parallelism for you - data distribution.
- On disk: HDFS distributes, replicates data as it comes in
- Keeps track; computations local to data



Data Distribution: Network

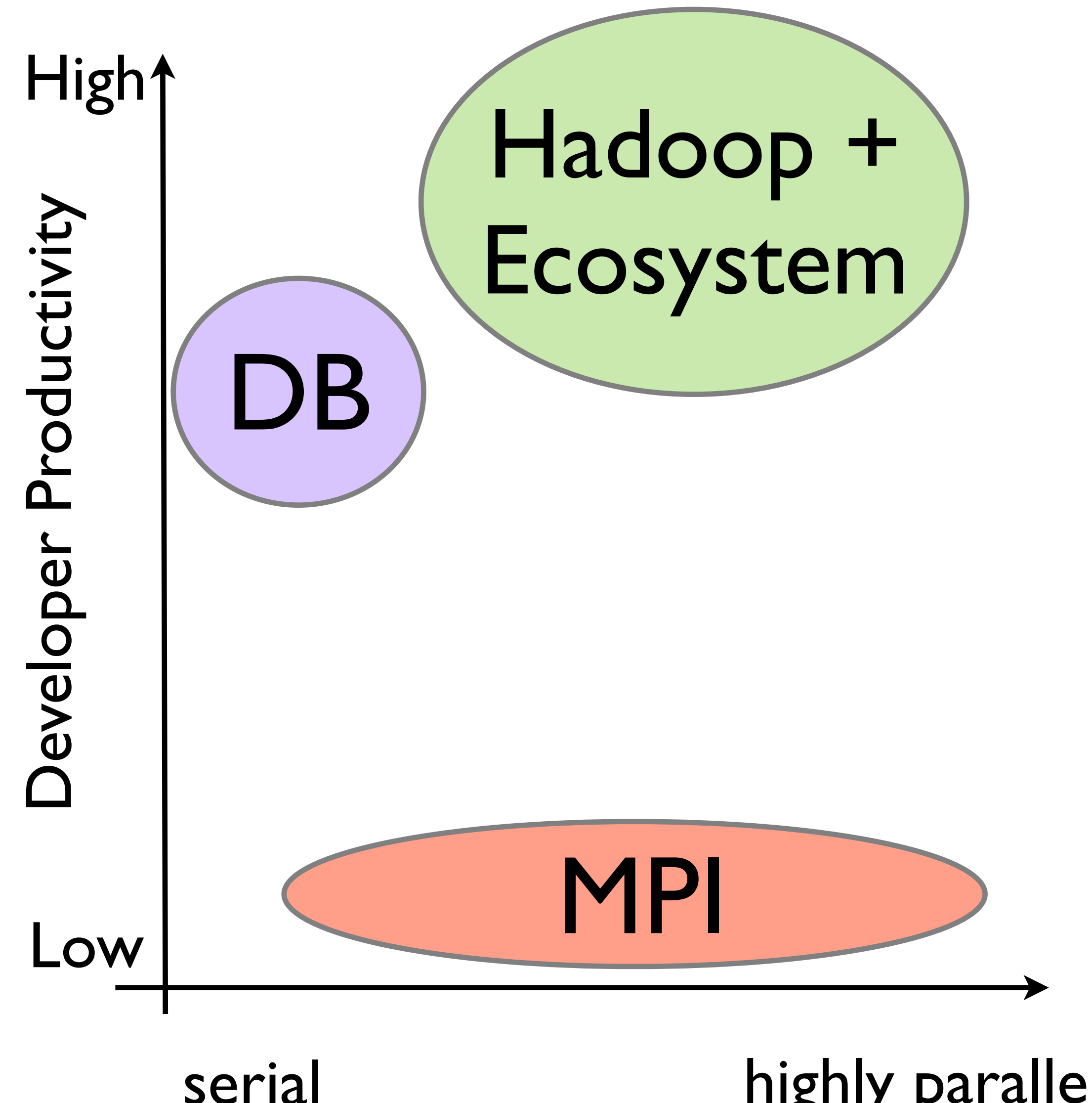
- On network: Map Reduce works in terms of key-value pairs.
- Preprocessing (map) phase ingests data, emits (k,v) pairs
- Shuffle phase assigns reducers, gets all pairs with same key onto that reducer.
- Programmer does not have to design communication patterns



SciNet compute • calcul CANADA

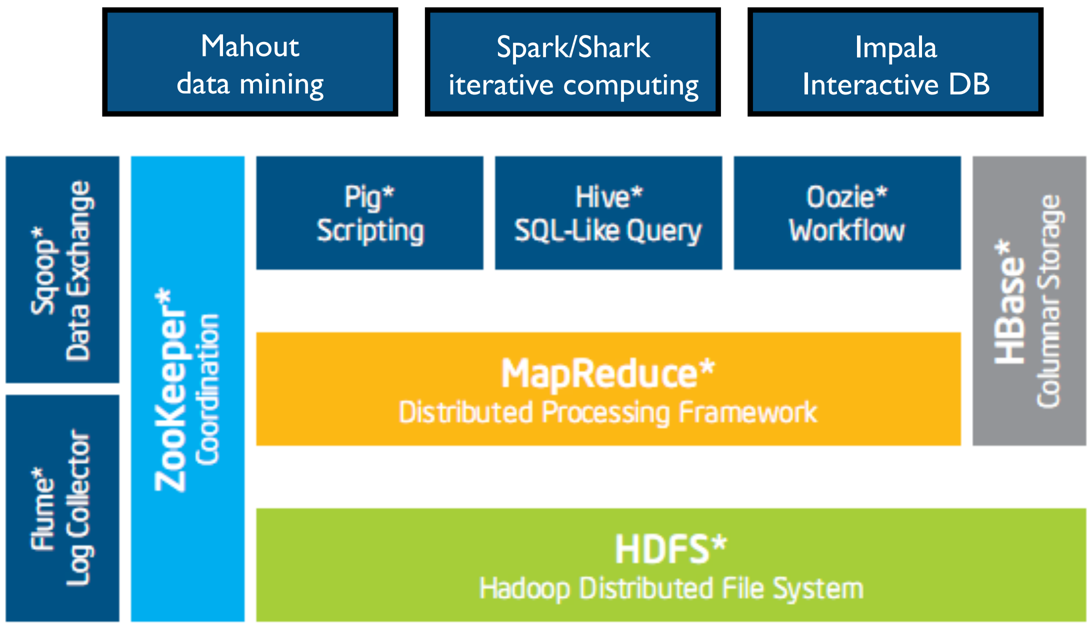
Makes the problem easier

- Decomposing the problem, and,
- Getting the intermediate data where it needs to go,
- ... are the hardest parts of parallel programming with HPC tools.
- Hadoop does that for you automatically for a wide range of problems.



Built a reusable substrate

- The filesystem (HDFS) and the MapReduce layer were very well architected.
- Enables many higher-level tools
- Data analysis, machine learning, NoSQL DBs,...
- Extremely productive environment
- And Hadoop 2.x (YARN) is now much much more than just MapReduce

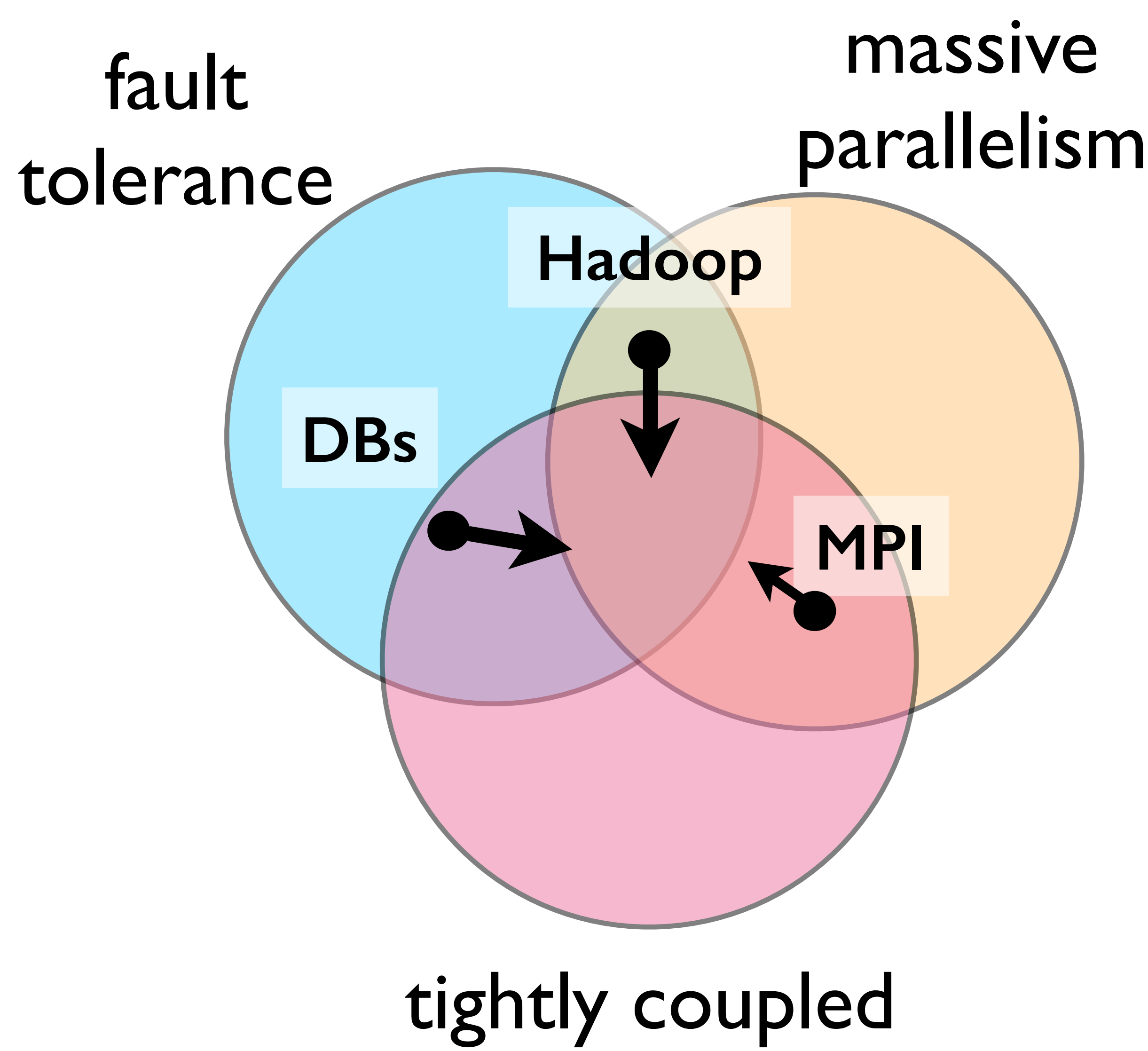


Hadoop ~~vs~~ *and* HPC

- Not either-or anyway
- Use HPC to generate big / many simulations, Hadoop to analyze results
- Use Hadoop to preprocess huge input data sets (ETL), and HPC to do the tightly coupled computation afterwards.
- Besides, ...

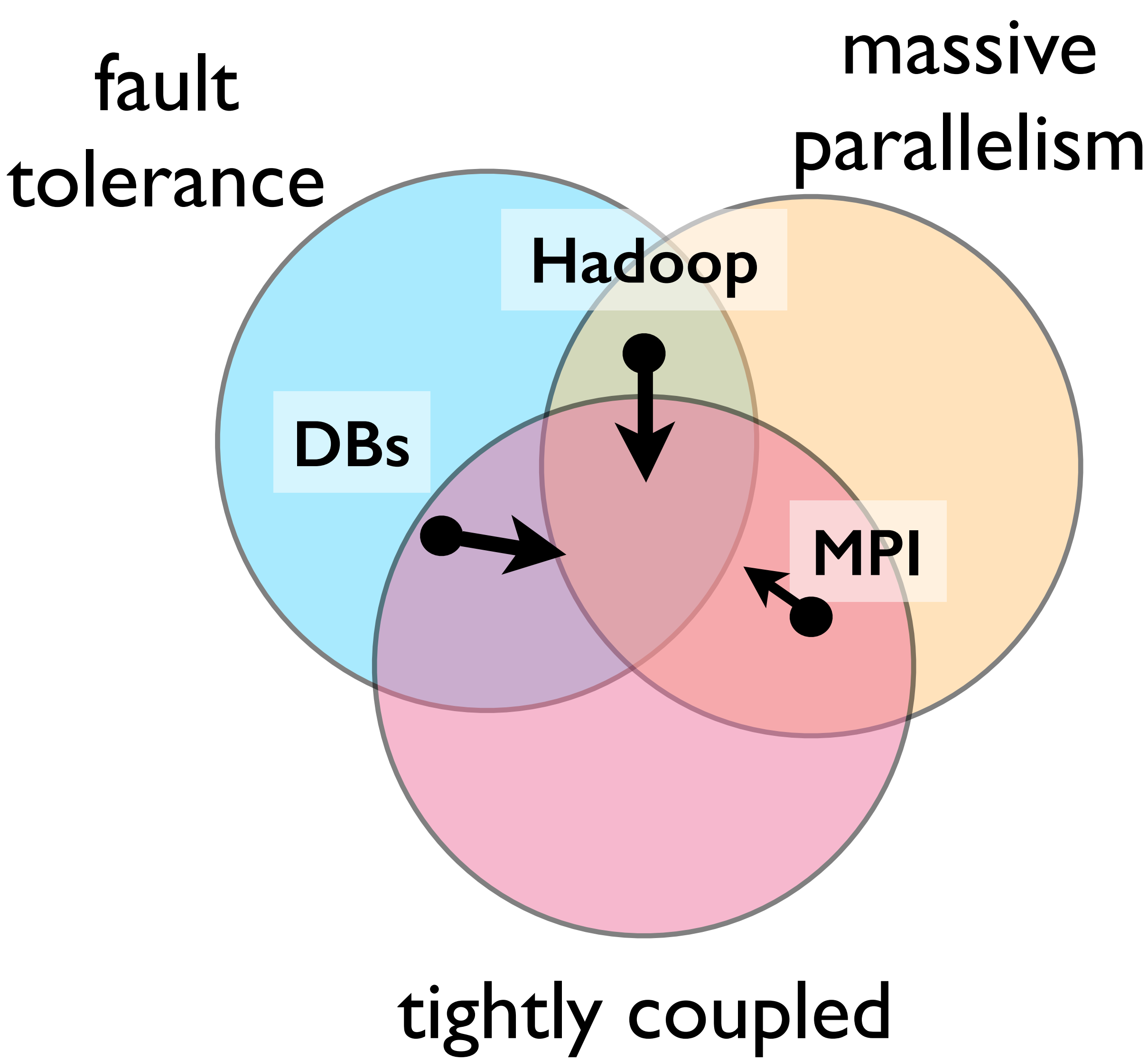
1: Everything's Converging

- These models are all converging at the largest scales
- Good ideas are good ideas.
- MPI is trying to grow fault tolerance (but MPI codes?)
- DBs are trying to scale up



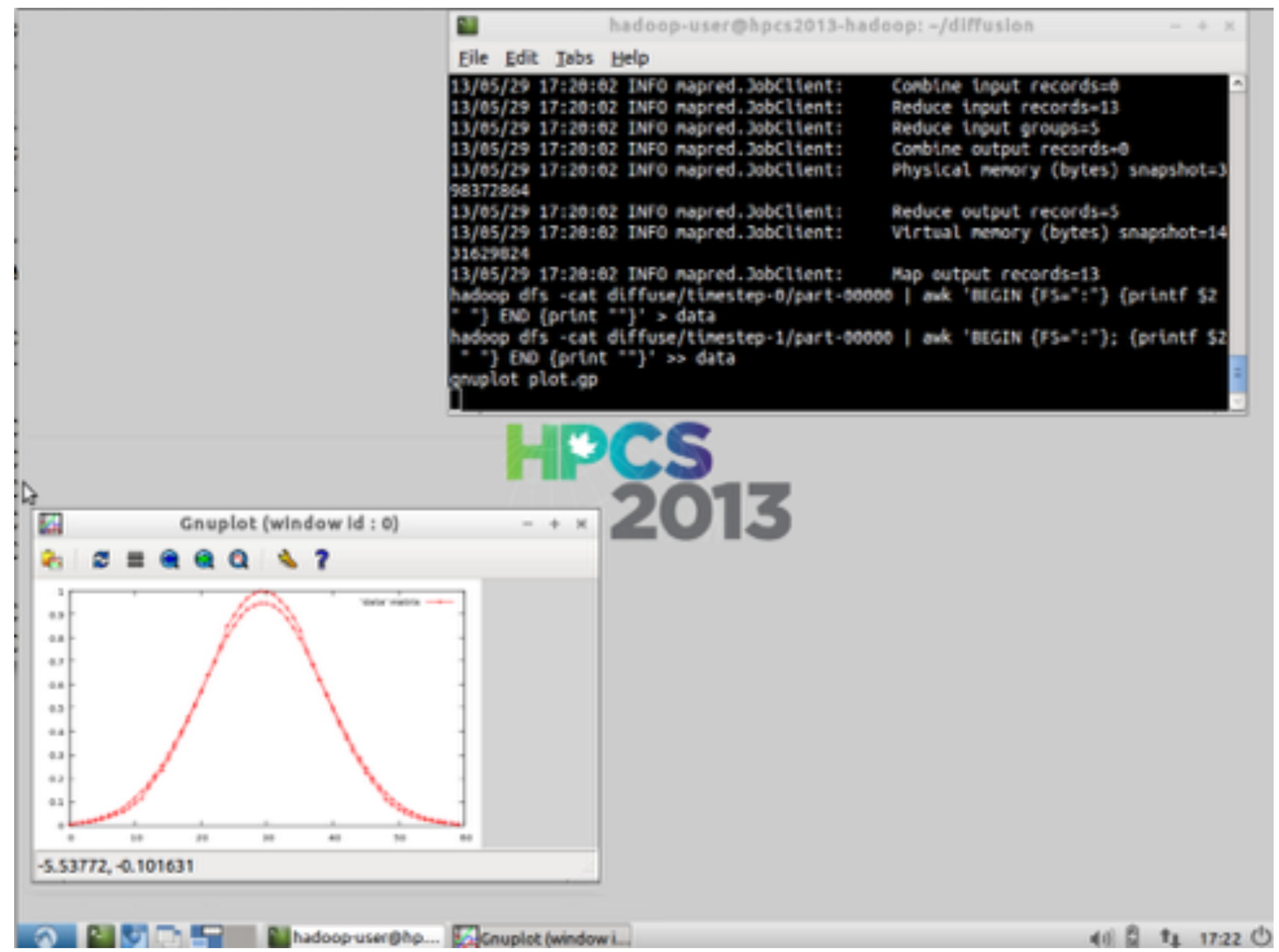
1: Everything's Converging

- People are building tools for tightly coupled computation atop Hadoop-like frameworks
- Hadoop will probably do tightly coupled well long before MPI codes do fault tolerance (cf. Spark, Giraph…)



2: Computation is Computation

- These models of computation aren't that different
- Different problems fall in different models' "sweet spots".
- In VM, cd ~/diffusion; make
- (will take a while)
- Distributed 1d diffusion PDE
- Will also look at (more reasonably) sparse matrix multiplication



Hadoop Job Workflow

- Let's take a look at the Makefile in the wordcount example
- Three basic tasks; building program; copying files in; running; getting output

```
INPUT_DIR   = /user/$(USER)/wordcount/input
OUTPUT_DIR  = /user/$(USER)/wordcount/output
OUTPUT_FILE = $(OUTPUT_DIR)/part-00000

run: wordcount.jar
	hadoop dfs -test -e $(INPUT_DIR)/file01 \
	  || hadoop dfs -put input/file01 $(INPUT_DIR)/file01
	hadoop dfs -test -e $(INPUT_DIR)/file02 \
	  || hadoop dfs -put input/file02 $(INPUT_DIR)/file02
	-hadoop dfs -rmr $(OUTPUT_DIR)
	hadoop jar wordcount.jar org.hpcs2013.WordCount \
	       $(INPUT_DIR) $(OUTPUT_DIR)
	hadoop dfs -cat $(OUTPUT_FILE)

wordcount.jar: WordCount.java
	mkdir -p wordcount_classes
	javac -classpath $(HADOOP_PREFIX)/hadoop-core-$(HADOOP_VER
	      -d wordcount_classes WordCount.java
	jar -cvf wordcount.jar -C wordcount_classes .

clean:
	-rm wordcount.jar
	-rm -r wordcount_classes
	-hadoop dfs -rmr $(INPUT_DIR)
	-hadoop dfs -rmr $(OUTPUT_DIR)

.PHONY: clean run
```

SciNet compute • calcul CANADA

Hadoop Job Workflow

```
wordcount.jar: WordCount.java
	mkdir -p wordcount_classes
	javac -classpath $(HADOOP_PREFIX)/hadoop-core-$(HADOOP_VERSION).jar \
		-d wordcount_classes WordCount.java
	jar -cvf wordcount.jar -C wordcount_classes .
```

- Building program; compile to bytecode against the current version of Hadoop
- Build a .jar file which contains all the relevant classes; this .jar file gets shipped off in its entirety to workers

Hadoop Job Workflow

```
run: wordcount.jar
	hadoop dfs -test -e $(INPUT_DIR)/file01 \
	  || hadoop dfs -put input/file01 $(INPUT_DIR)/file01
	hadoop dfs -test -e $(INPUT_DIR)/file02 \
	  || hadoop dfs -put input/file02 $(INPUT_DIR)/file02
	-hadoop dfs -rmr $(OUTPUT_DIR)
	hadoop jar wordcount.jar org.hpcs2013.WordCount \
	       $(INPUT_DIR) $(OUTPUT_DIR)
	hadoop dfs -cat $(OUTPUT_FILE)
```

- Running the program: must first copy the input files (hdfs dfs -put) onto the Hadoop file system
- Remove (hdfs dfs -rm -r) the output directory if it exists
- Run the program by specifying the input jar file and the class of the program, and give it any arguments
- Type out (cat) the output file.

The Hadoop Filesystem

hdfs dfs -[cmd]

- HDFS is a distributed parallel filesytem
- Not a general purpose file system
  - doesn't implement posix
  - can't just mount it and view files
- Access via "hdfs dfs" commands
- Also programatic (java) API
- Security slowly improving

cat	mkdir
chgrp	movefromLocal
chmod	mv
chown	put
copyFromLocal	rm
copyToLocal	rmr
cp	setrep
du	stat
dus	tail
expunge	test
get	text
getmerge	touchz
ls	
lsr	

The Hadoop Filesystem

Required to be:

- able to deal with large files, large amounts of data
- scalable
- reliable in the presence of failures
- fast at reading contiguous streams of data
- only need to write to new files or append to files
- require only commodity hardware

The Hadoop Filesystem

As a result:

- Replication
- Supports mainly high bandwidth, not low latency
- No caching (what's the point if just streaming reads)
- Poor support for seeking around files
- Poor support for zillions of files
- Have to use separate API to see filesystem

Modeled after Google File System (2004 Map Reduce paper)

Blocks in HDFS

- HDFS is a block-based file system.
- A file is broken into blocks, these blocks are distributed across nodes
- Blocks are large; 64MB is default, many installations use 128MB or larger
- Large block size - time to stream a block much larger than time disk time to access the block.
- hdfs fsck / -files -blocks lists all blocks in all files

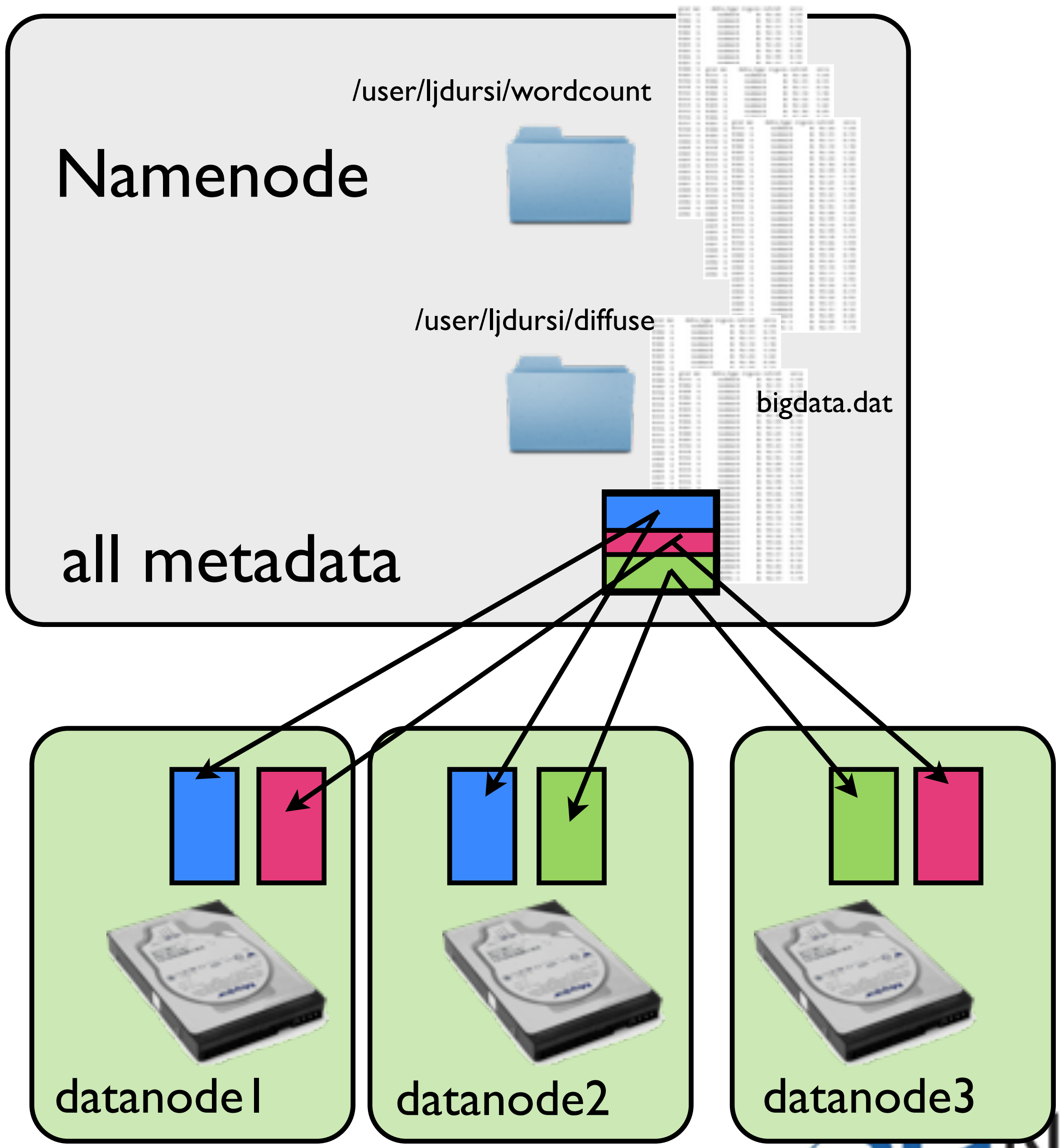
Datanodes and Namenode

Two different types of nodes in the filesystem

Namenode - stores all metadata and block locations in memory.

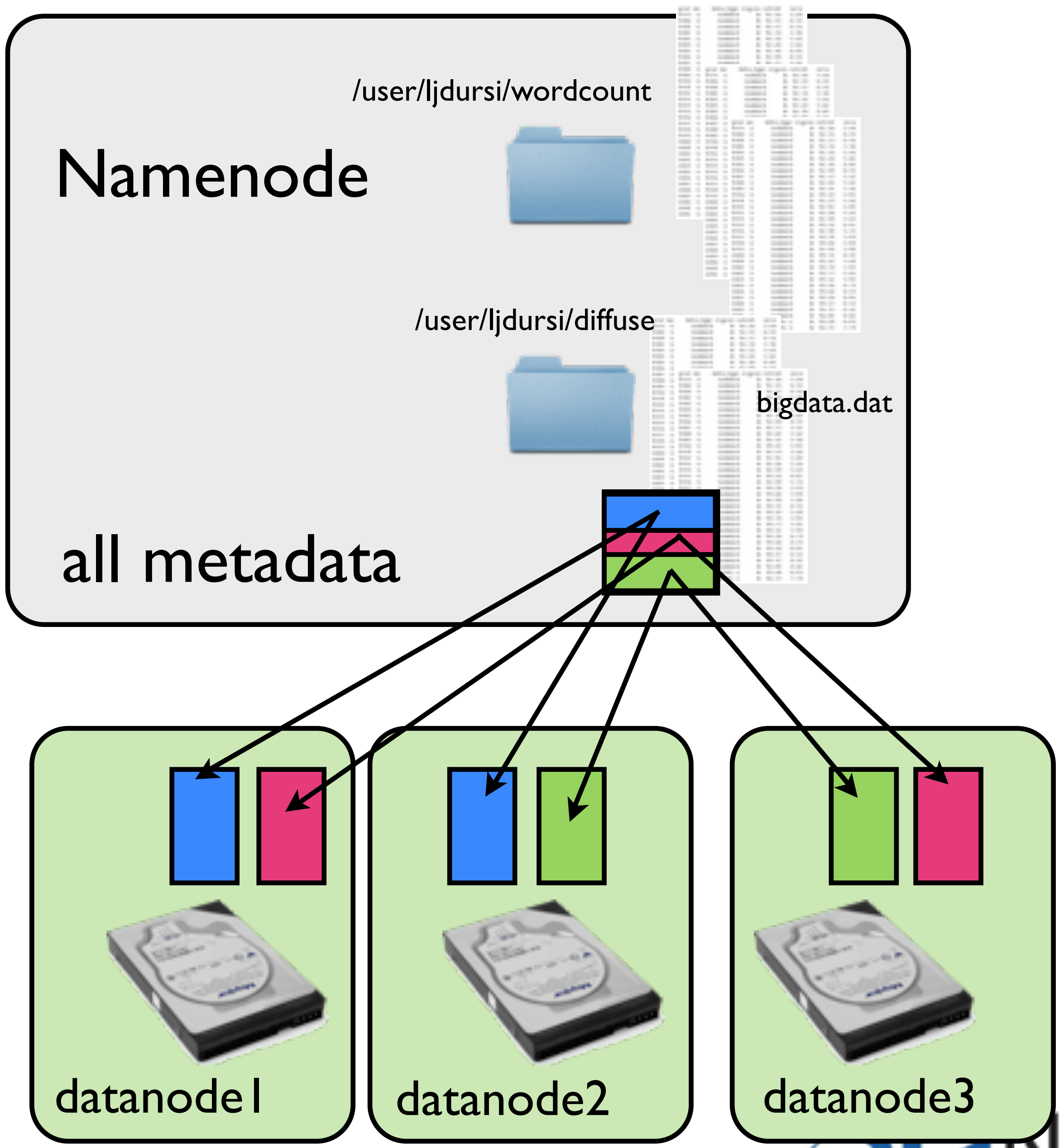
- Updates are stored to persistent journal
- Lots of files bad

Datanodes - store and retrieve blocks for client or namenode



Datanodes and Namenode

- Newer versions of Hadoop - federation (different namenodes for /user, /data, /project , etc)
- Newer versions of Hadoop - High Availability namenode pairs

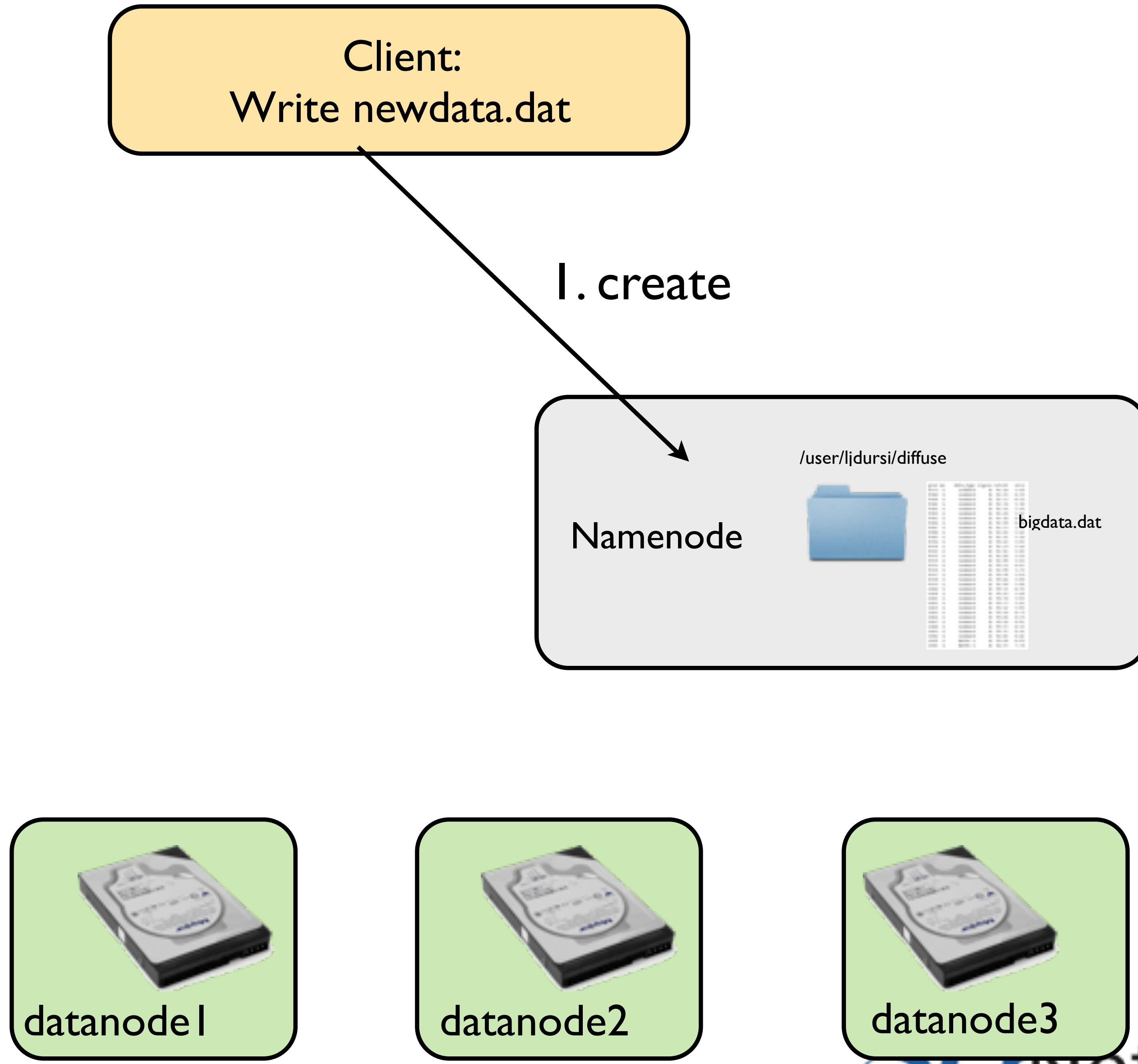


SciNet compute • calcul CANADA

Writing a file

Writing a file multiple stage process

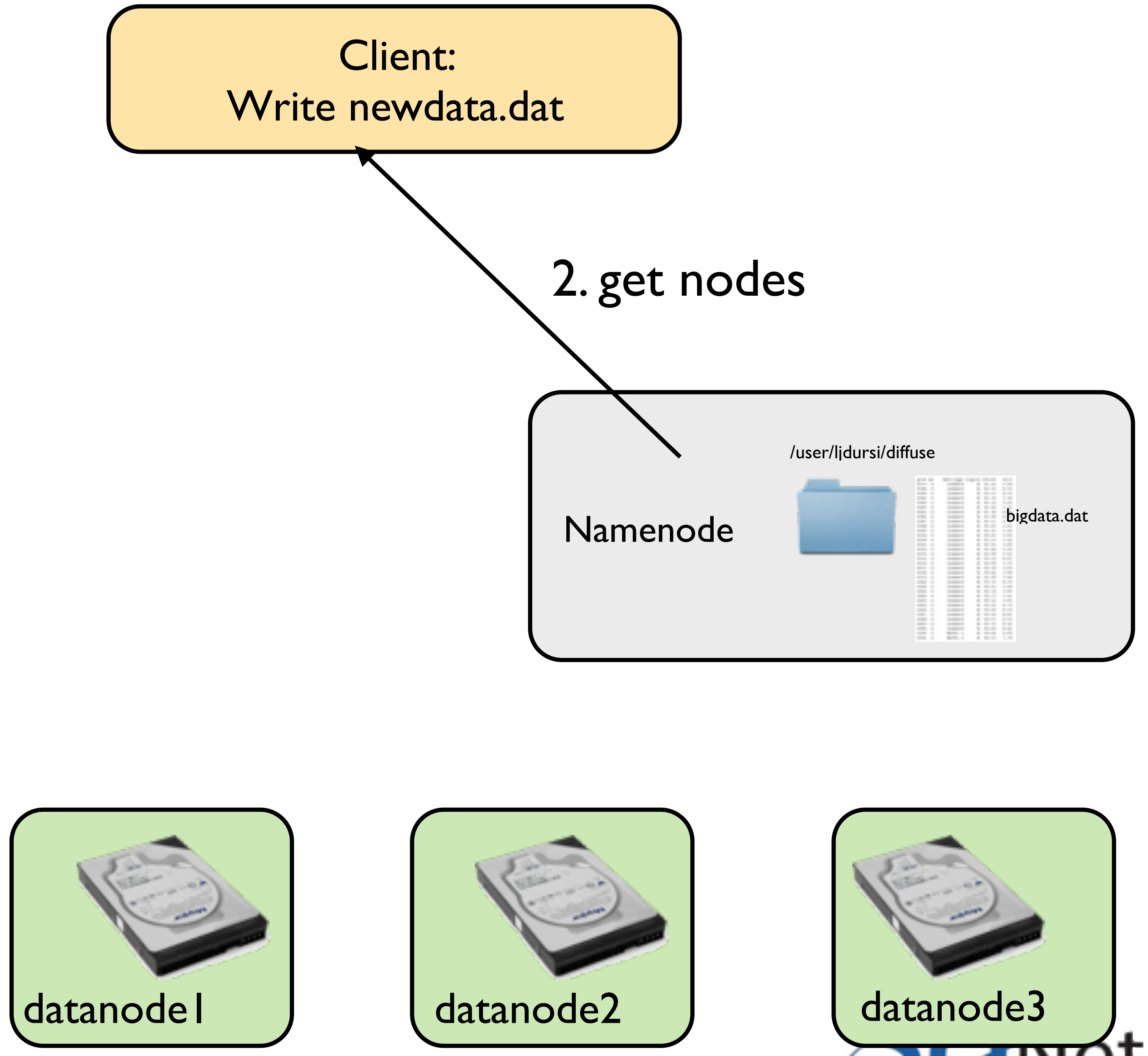
- Create file
- Get nodes for blocks
- Start writing
- Data nodes coordinate replication
- Get ack back
- Complete



Writing a file

Writing a file multiple stage process

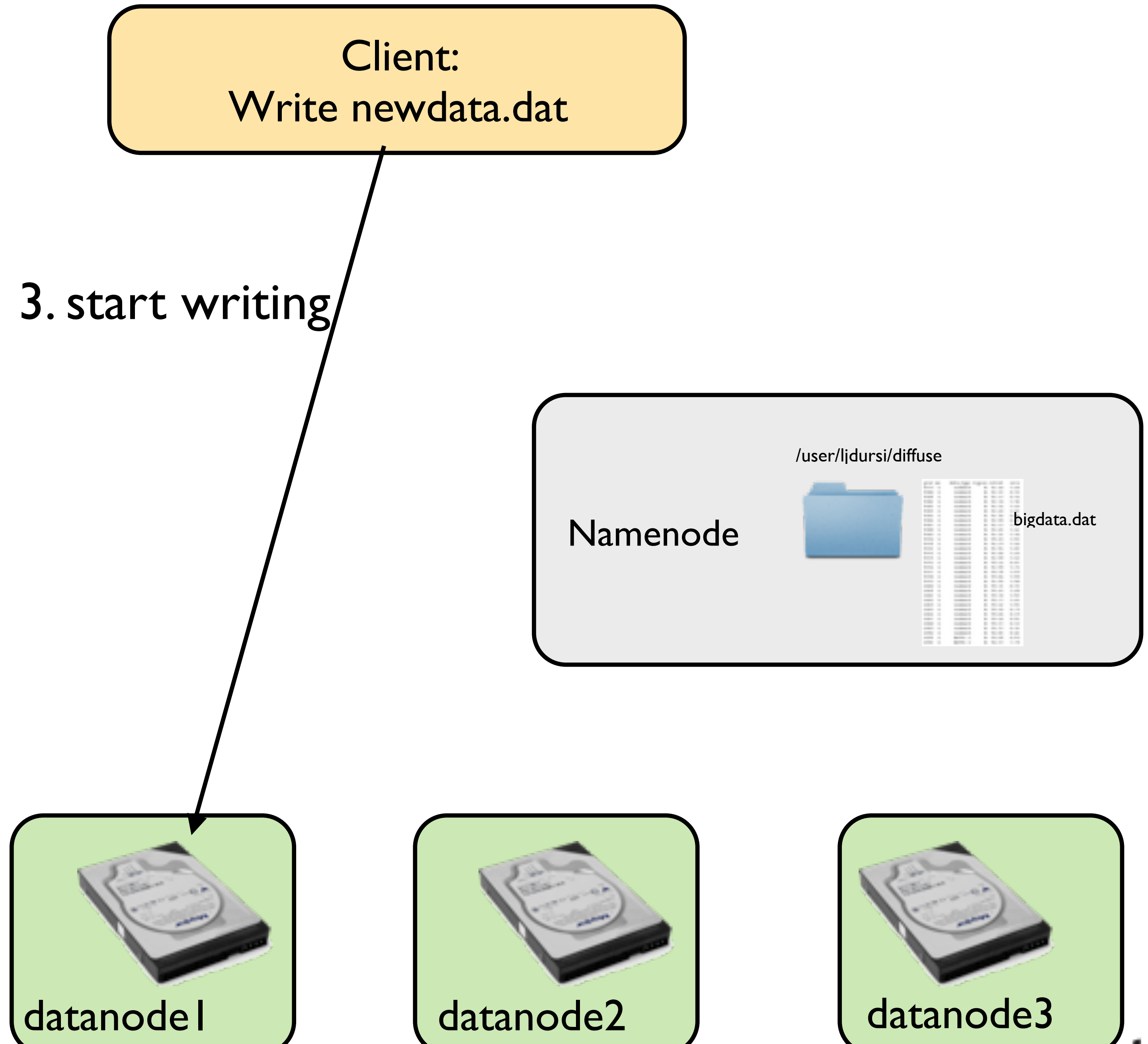
- Create file
- Get nodes for blocks
- Start writing
- Data nodes coordinate replication
- Get ack back
- Complete



Writing a file

Writing a file multiple stage process

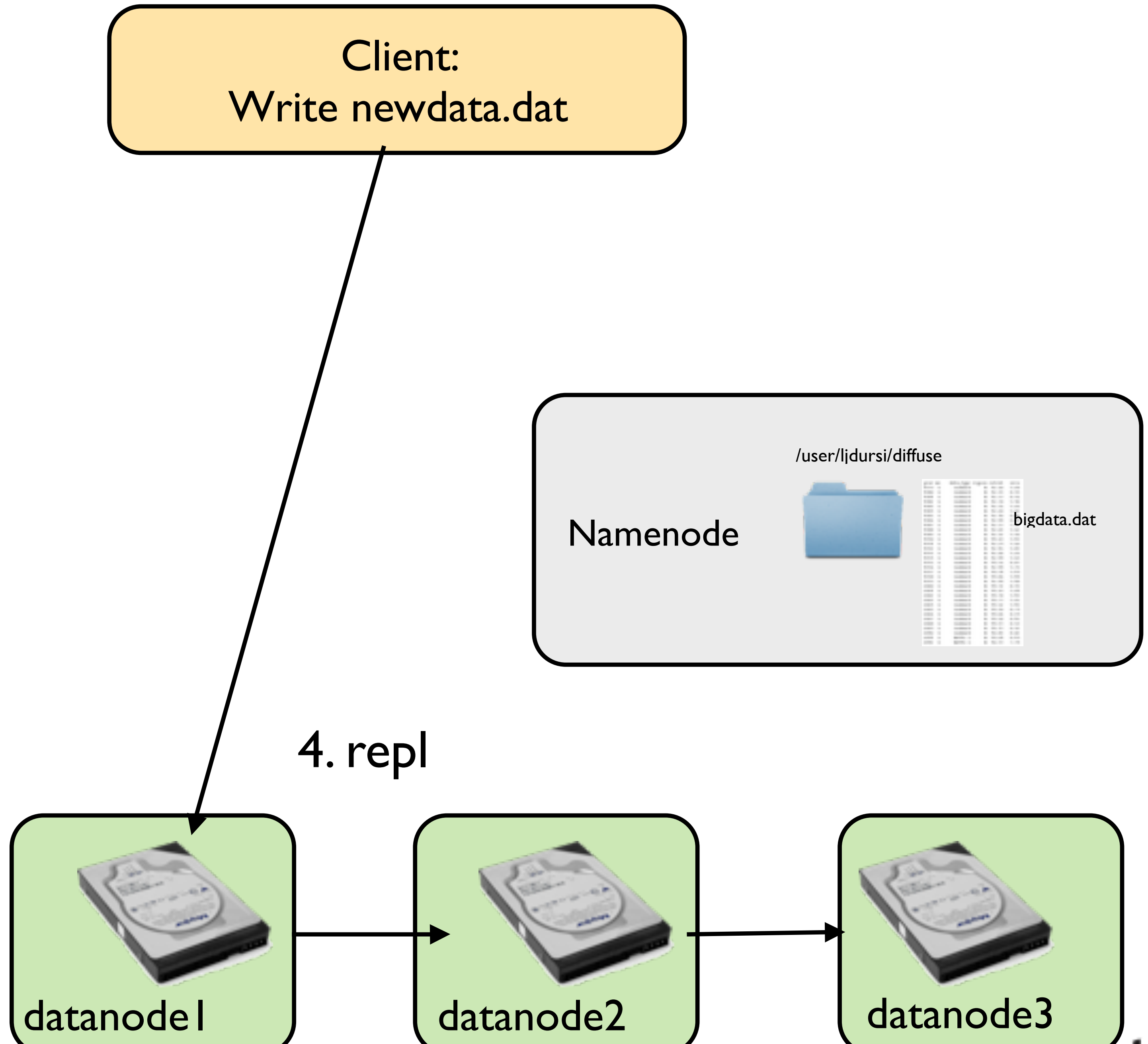
- Create file
- Get nodes for blocks
- Start writing
- Data nodes coordinate replication
- Get ack back
- Complete



Writing a file

Writing a file multiple stage process

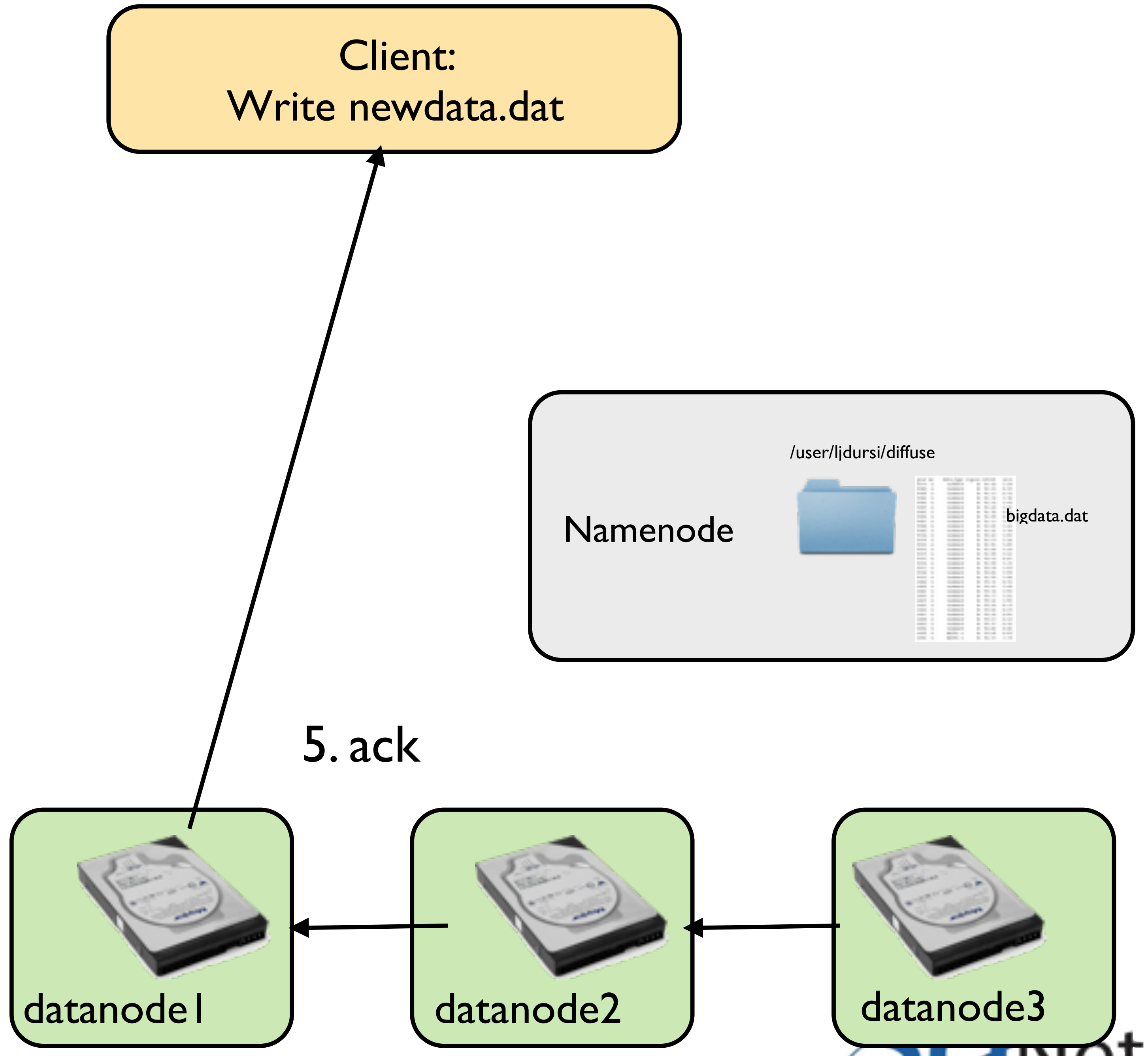
- Create file
- Get nodes for blocks
- Start writing
- Data nodes coordinate replication
- Get ack back
- Complete



Writing a file

Writing a file multiple stage process

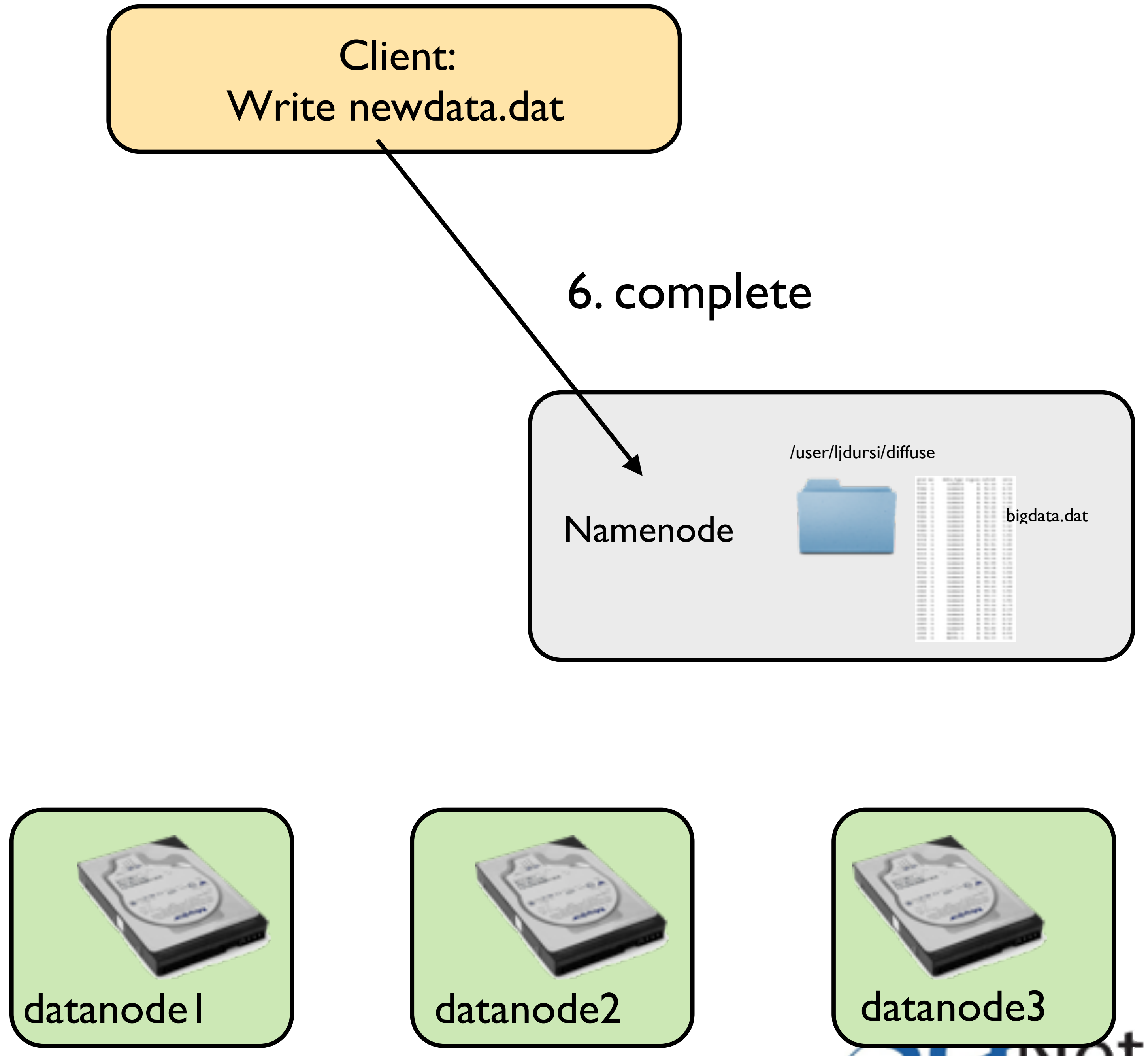
- Create file
- Get nodes for blocks
- Start writing
- Data nodes coordinate replication
- Get ack back (while writing)
- Complete



Writing a file

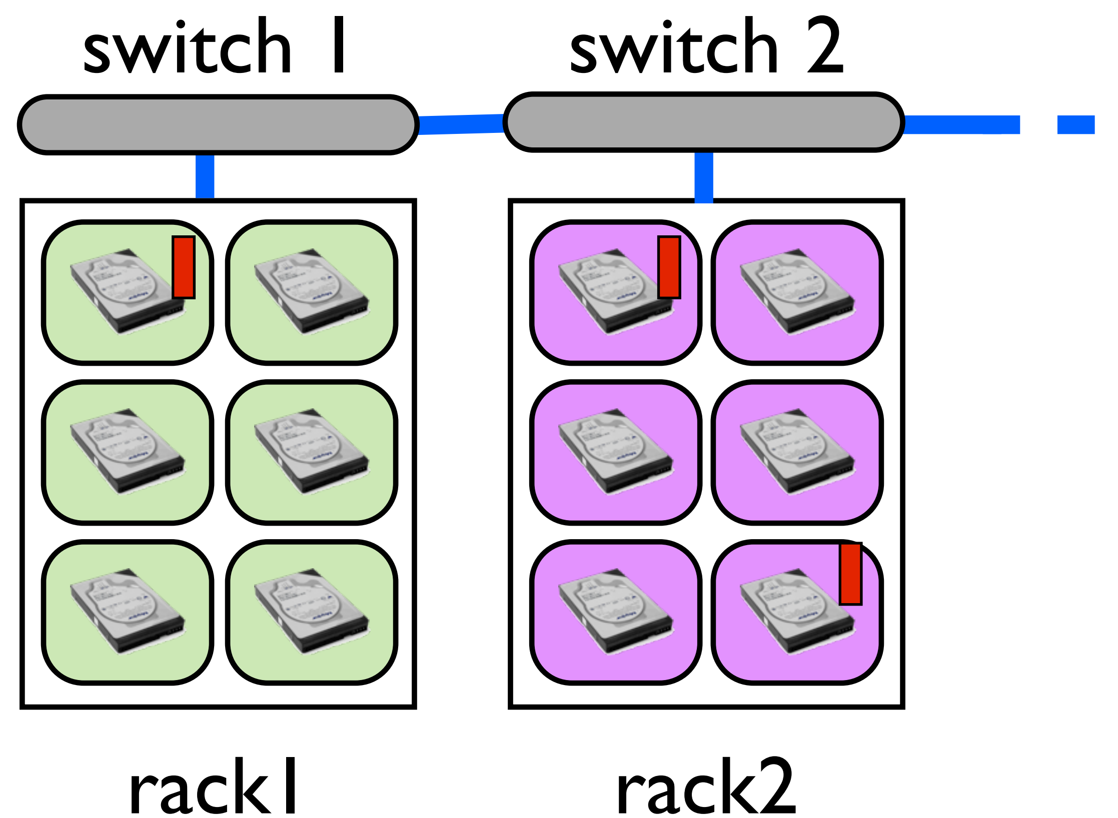
Writing a file multiple stage process

- •Create file
- •Get nodes for blocks
- •Start writing
- •Data nodes coordinate replication
- •Get ack back
- •Complete



Where to Replicate?

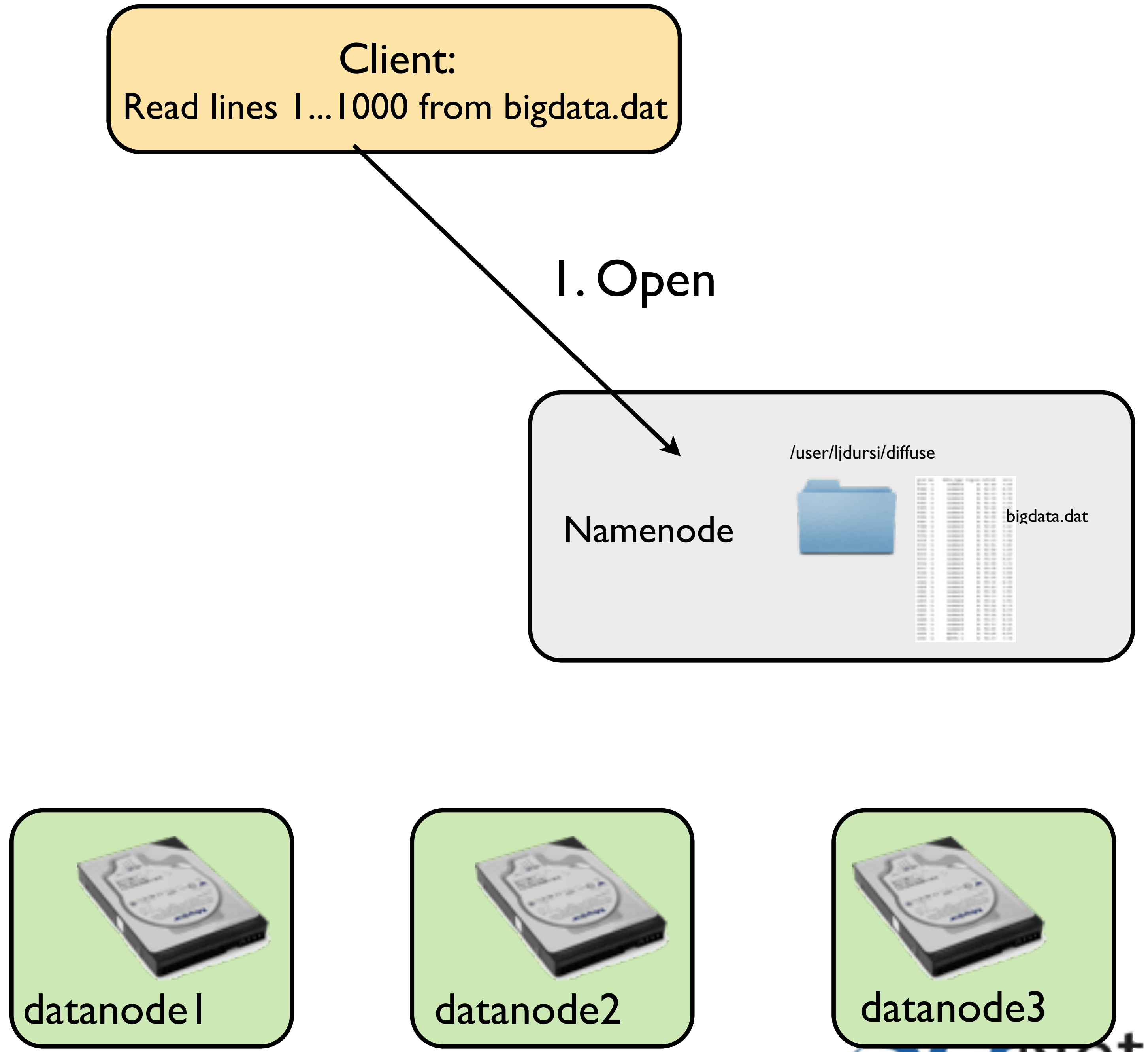
- Tradeoff to choosing replication locations
- Close: faster updates, less network bandwidth
- Further: better failure tolerance
- Default strategy: first copy on different location on same node, second on different "rack"(switch), third on same rack location, different node.
- Strategy configurable.
- Need to configure Hadoop file system to know location of nodes



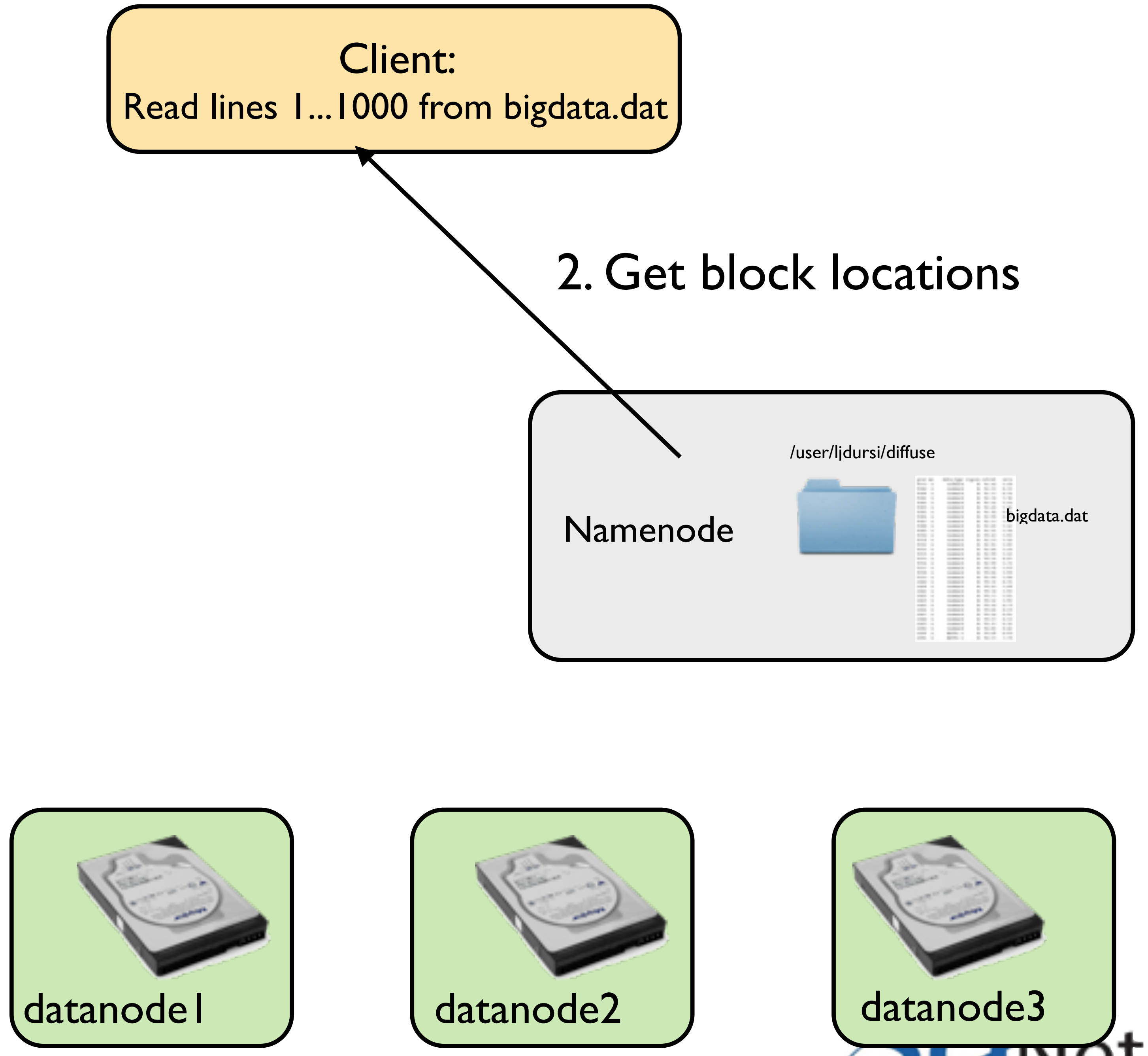
Reading a file

Reading a file shorter

- Get block locations
- Read



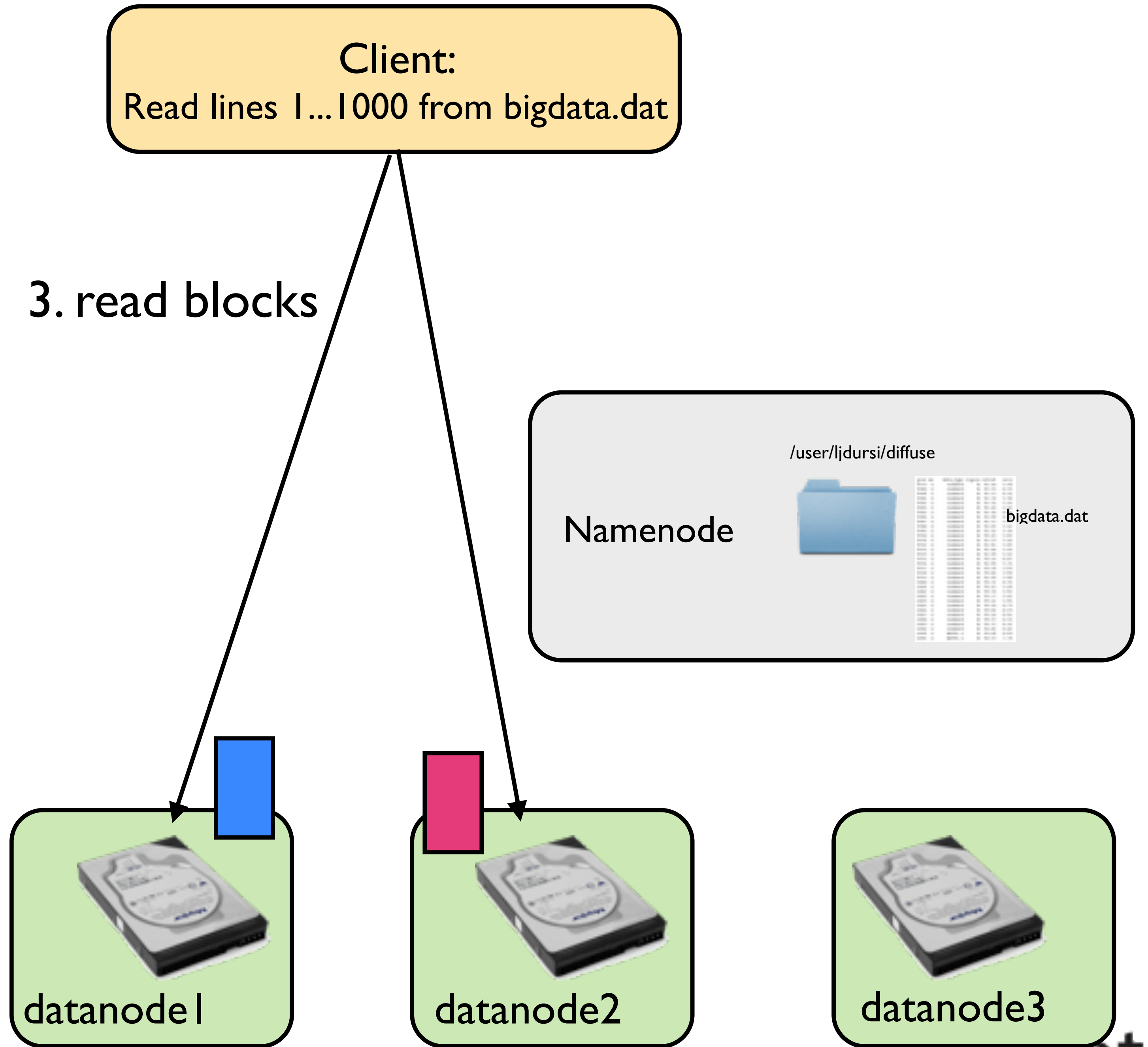
Reading a file



Reading a file shorter

- Get block locations
- Read

Reading a file



Reading a file shorter

- Get block locations
- Read

Configuring HDFS

- Need to tell HDFS how to set up filesystem
- data.dir, name.dir - where on local system (eg, local disk) to write data
- parameters like replication - how many copies to make
- default name - default file system to use
- Can specify multiple

```
<configuration>
  <property>
    <name>fs.default.name</name>
    <value>hdfs://your.server.name.com:9000</value>
  </property>

  <property>
    <name>dfs.data.dir</name>
    <value>/home/username/hdfs/data</value>
  </property>

  <property>
    <name>dfs.name.dir</name>
    <value>/home/username/hdfs/name</value>
  </property>

  <property>
    <name>dfs.replication</name>
    <value>3</value>
  </property>
</configuration>
```

Configuring HDFS

$HADOOP_PREFIX/etc/hadoop/core-site.xml

For us:

- Only one node to be used, our laptops
- default: localhost

```
<configuration>

  <property>
    <name>fs.default.name</name>
    <value>hdfs://localhost:9000</value>
  </property>

</configuration>
```

Configuring HDFS

$HADOOP_PREFIX/etc/hadoop/hdfs-site.xml

```
<configuration>
. . .

  <property>
    <name>dfs.replication</name>
    <value>1</value>
  </property>

</configuration>
```

- Since only one node, need to specify replication factor of 1, or will always fail

Configuring HDFS

- Also need to make sure that environment variables are set
- path to Java, path to Hadoop

~/.bashrc

```
...

export JAVA_HOME=/usr/lib/jvm/default-java
export HADOOP_VERSION=1.1.2
export HADOOP_PREFIX=/path/to/hadoop-${HADOOP_VERSION}

...
```

$HADOOP_PREFIX/etc/hadoop/hadoop-env.sh

```
...

export JAVA_HOME=/usr/lib/jvm/default-java

...
```

Configuring HDFS

```
$ ssh-keygen -t dsa -P '' -f ~/.ssh/id_dsa
$ cat ~/.ssh/id_dsa.pub >> ~/.ssh/authorized_keys
```

- Finally, have to make sure that passwordless login is enabled
- Can start processes on various FS nodes

Configuring HDFS

- Once configuration files are set up, can format the namenode like so
- Then you can start up just the file systems:

Done for you in init.sh

```
. . .
$ hdfs namenode -format
$ start-dfs.sh
. . .
```

Using HDFS

- Now once the file system is up and running, you can copy files back and forth
- `get/put, copyFromLocal/ copyToLocal`
- Default wd is `/user/${username}`
- Nothing like a "cd"
- Try copying a Makefile or something to HDFS, doing an ls, then copying it back and make sure it's stayed same.

hadoop fs -[cmd]

cat	mkdir
chgrp	movefromLocal
chmod	mv
chown	put
copyFromLocal	rm
copyToLocal	rmr
cp	setrep
du	stat
dus	tail
expunge	test
get	text
getmerge	touchz
ls	
lsr	

SciNet compute • calcul CANADA

Hadoop Job Workflow

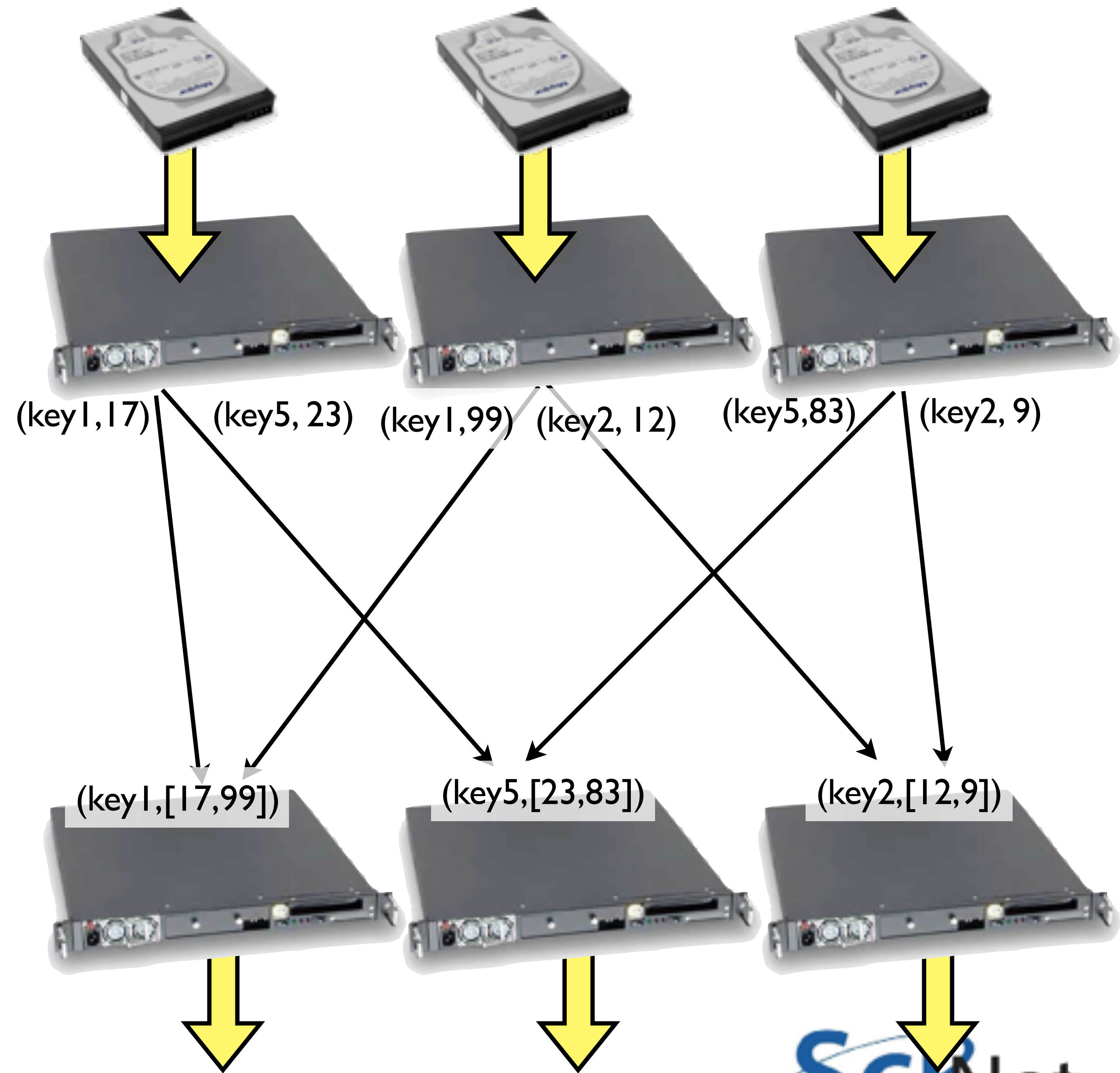
```
wordcount.jar: WordCount.java
	mkdir -p wordcount_classes
	javac -classpath $(CLASSPATH) -Xlint:deprecation \
		-d wordcount_classes WordCount.java
	jar -cvf wordcount.jar -C wordcount_classes .
```

Building the program

Running a "Map Reduce" program...

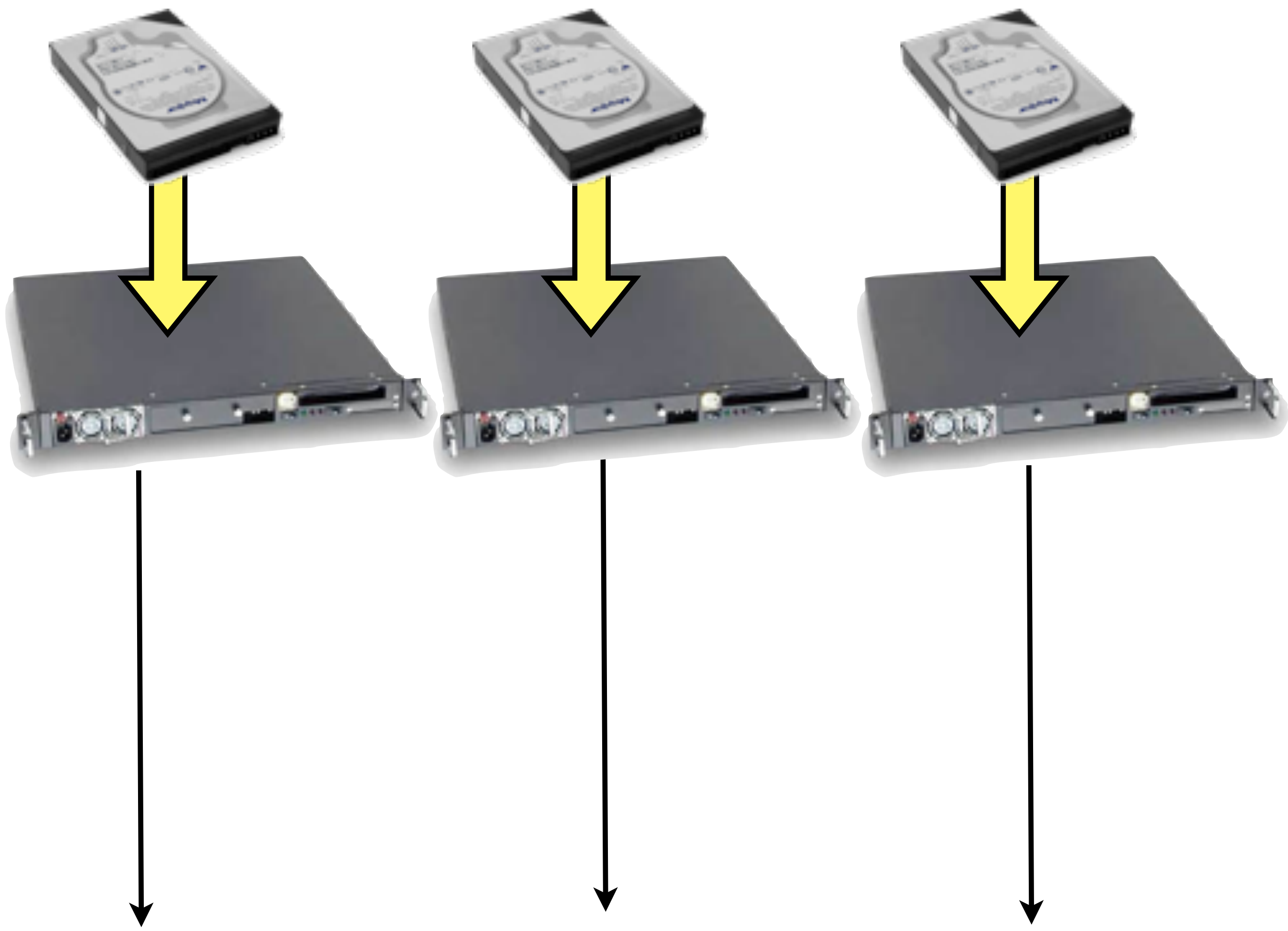
MapReduce

- Two classes of compute tasks: a Map and a Reduce
- Map processes one "element" at a time, emits results as (key, value) pairs.
- All results with same key are gathered to the same reducers
- Reducers process list of values, emit results as (key, value) pairs.



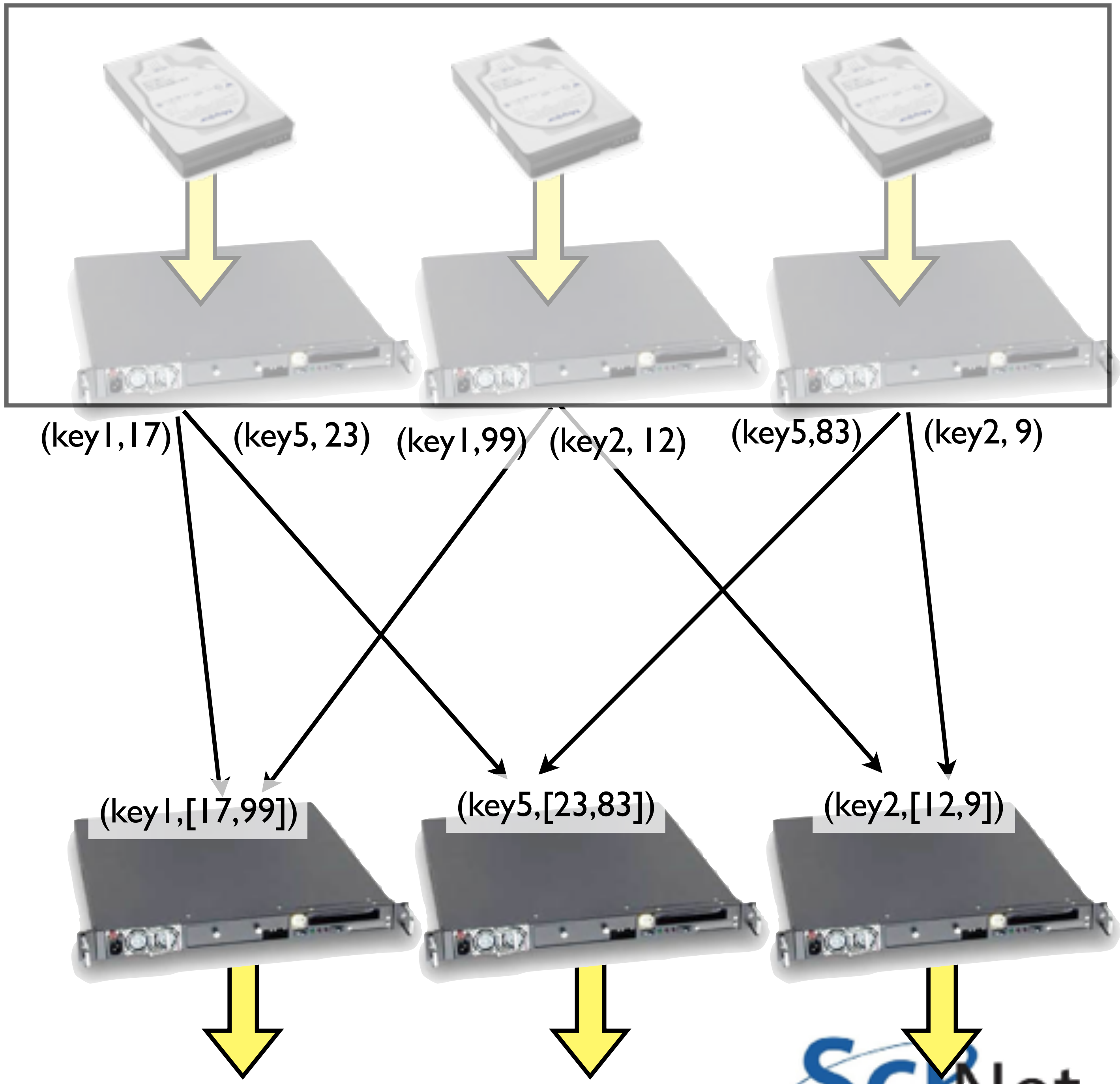
Map

- All coupling is done during the "shuffle" phase
- Embarrassingly parallel task - all map
- Take input, map it to output, done.
- (Famous case: NYT using Hadoop to convert 11 million image files to PDFs - almost pure serial farm job)



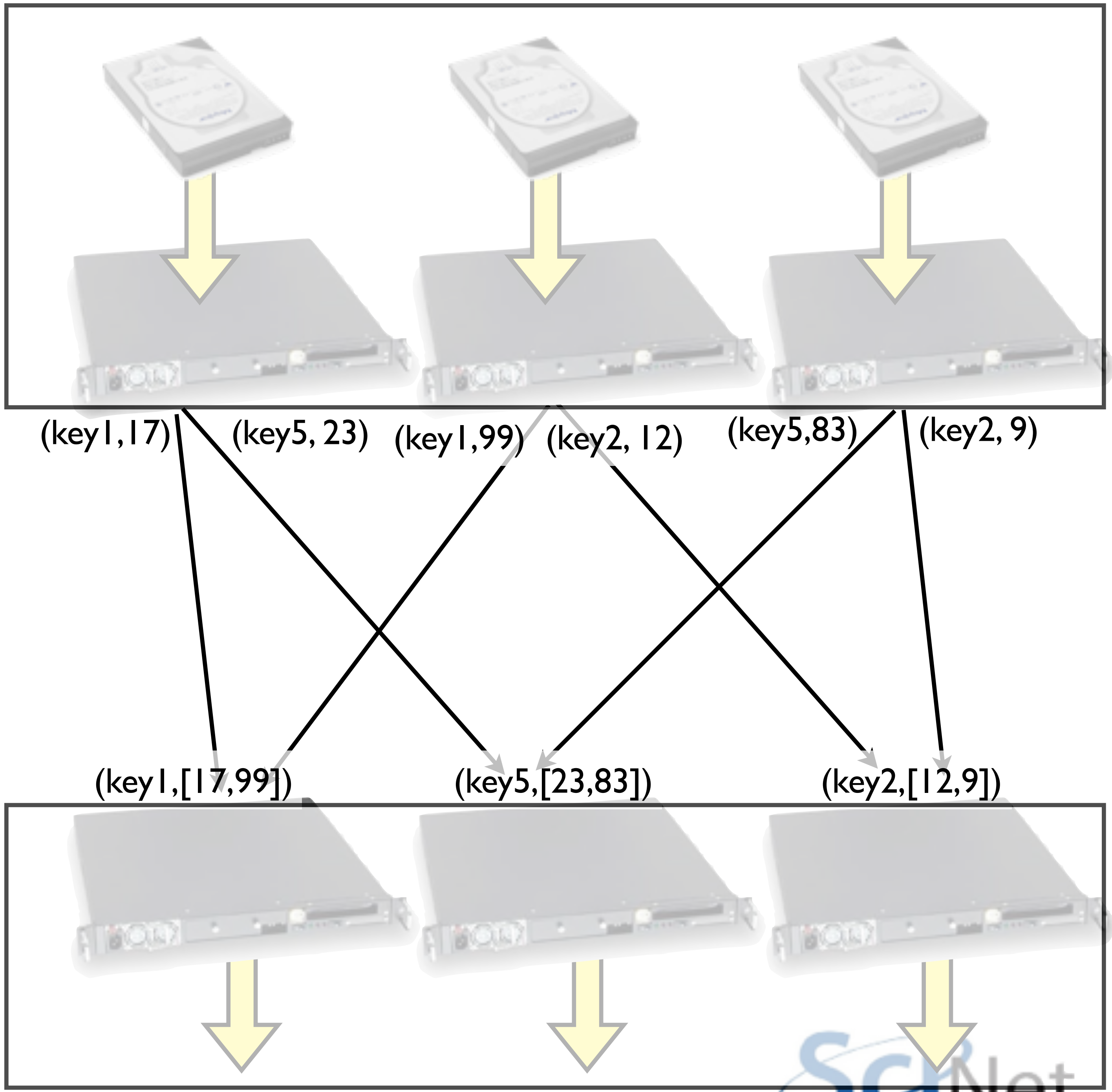
Reduce

- Reducing gives the coupling
- In the case of the NYT task, not quite embarrassingly parallel; images from multi-page articles
- Convert a page at a time, gather images with same article id onto node for conversion.



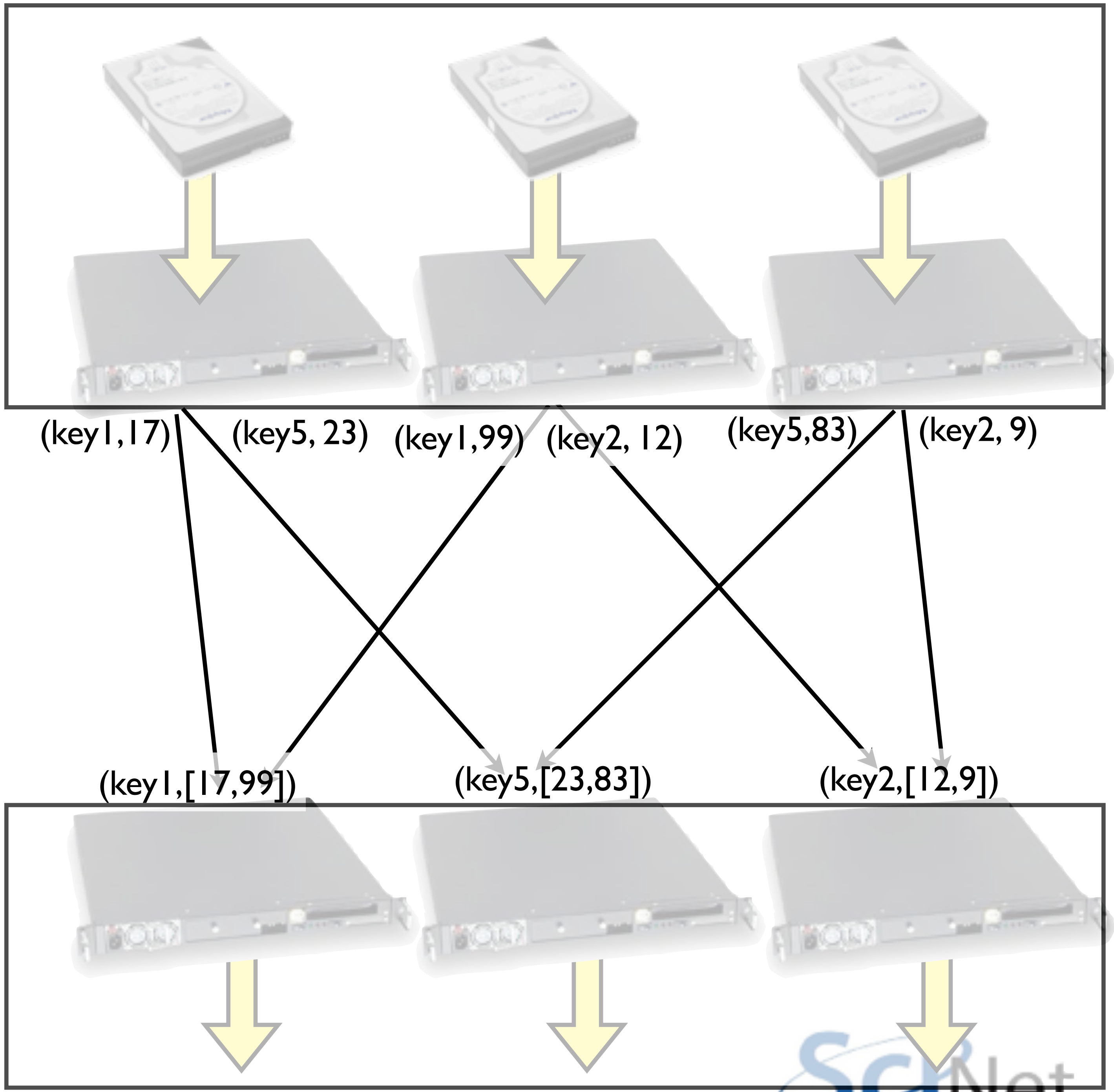
Shuffle

- The shuffle is part of the Hadoop magic
- By default, keys are hashed and hash space is partitioned between reducers
- On reducer, gathered (k,v) pairs from mappers are sorted by key, then merged together by key
- Reducer then runs on one (k,[v]) tuple at a time



Shuffle

- If you do know something about the structure of the problem, can supply your own partitioner
- Assign keys that are "similar" to each other to same node
- Reducer still only sees one (k, [v]) tuple at a time.



Word Count

- Was used as an example in the original MapReduce paper
- Now basically the "hello world" of map reduce
- Do a count of words of some set of documents.
- A simple model of many actual web analytics problem

`file01`

```
Hello World
Bye World
```

`file02`

```
Hello Hadoop
Goodbye Hadoop
```

`output/part-00000`

```
Hello     2
World     2
Bye       1
Hadoop    2
Goodbye   1
```

Word Count

- How would you do this with a huge document?
- Each time you see a word, if it's a new word, add a tick mark beside it, otherwise add a new word with a tick
- ...But hard to parallelize (updating the list)

file01

```
Hello World
Bye World
```

file02

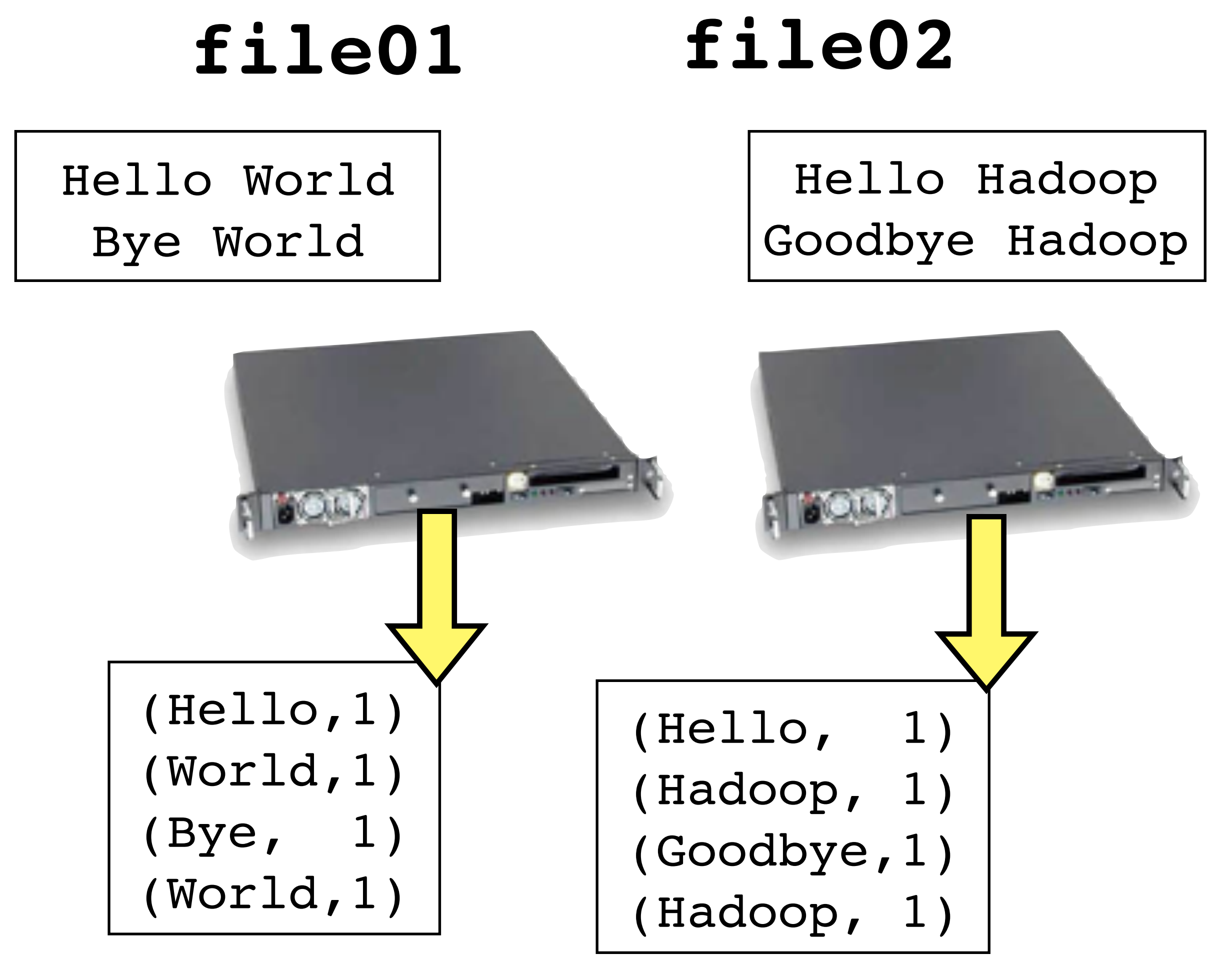
```
Hello Hadoop
Goodbye Hadoop
```

output/part-00000

```
Hello     2
World     2
Bye       1
Hadoop    2
Goodbye   1
```

Word Count

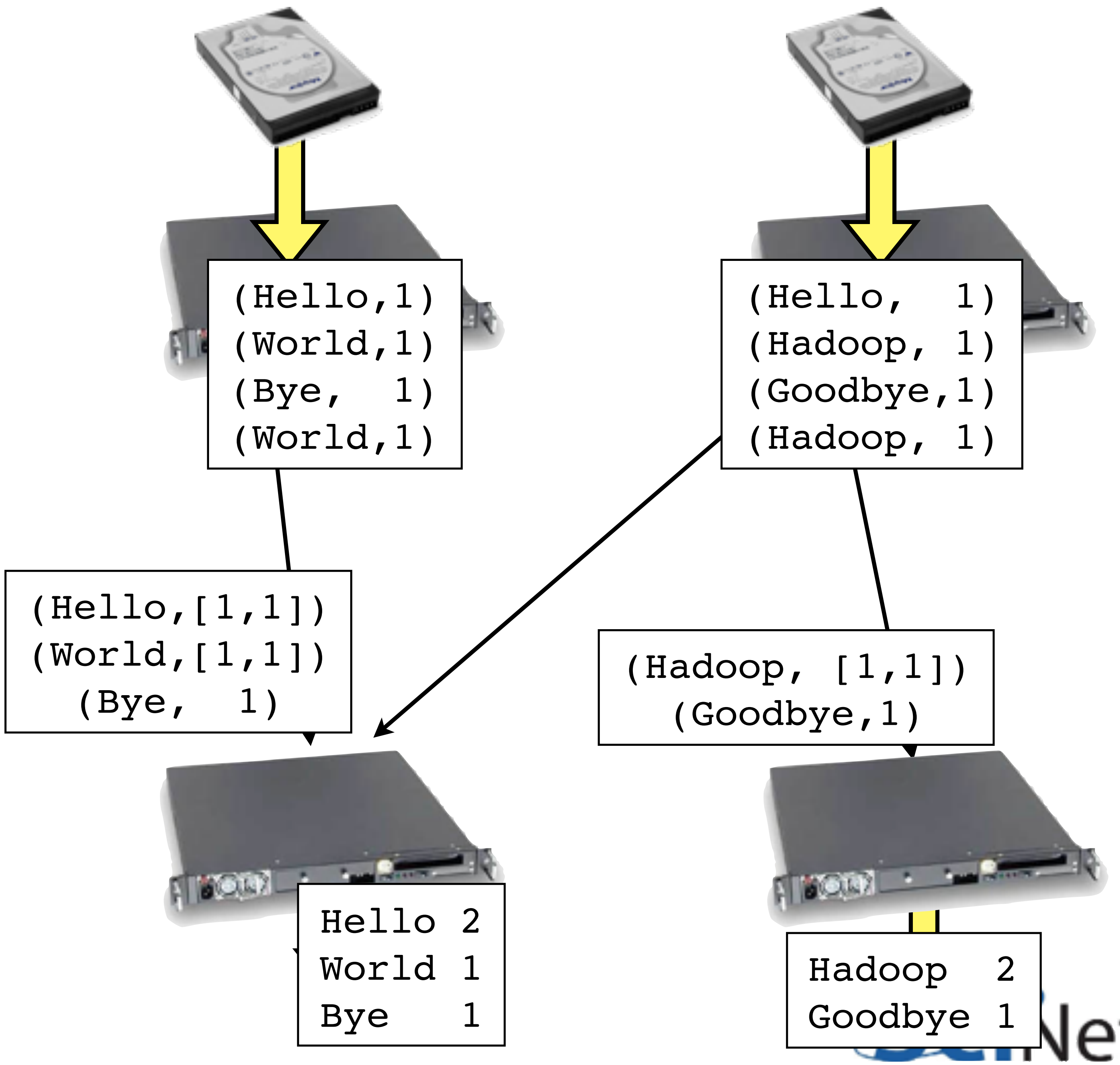
- MapReduce way - all hard work is done by the shuffle - eg, automatically.
- Map: just emit a 1 for each word you see



SciNet compute • calcul CANADA

Word Count

- Shuffle assigns keys (words) to each reducer, sends (k,v) pairs to appropriate reducer
- Reducer just has to sum up the ones



Hadoop Job Workflow

```
wordcount.jar: WordCount.java
    mkdir -p wordcount_classes
    javac -classpath $(CLASSPATH) -Xlint:deprecation \
          -d wordcount_classes WordCount.java
    jar -cvf wordcount.jar -C wordcount_classes .
```

- Building the program
- Class is expected to have particular methods
- Let's look at WordCount.java

main()

- The main() routine in a MapReduce computation creates a Job with a Configuration
- Set details of Input/Output, etc
- Then runs the job.

```
public class WordCount {

  /* ... */

  public static void main(String[] args) throws Exception {

    if (args.length != 2) {
      System.err.println("Usage: wordcount <in> <out>");
      System.exit(2);
    }

    Job job = Job.getInstance(new Configuration());
    job.setJobName("wordcount");
    job.setJarByClass(WordCount.class);

    job.setMapperClass(Map.class);
    job.setCombinerClass(Reduce.class);
    job.setReducerClass(Reduce.class);

    job.setOutputKeyClass(Text.class);
    job.setOutputValueClass(IntWritable.class);

    FileInputFormat.setInputPaths(job, new Path(args[0]));
    FileOutputFormat.setOutputPath(job, new Path(args[1]));

    job.submit();
    job.waitForCompletion(true);
  }
}
```

main()

- The heart of doing work in Hadoop originally was MapReduce
- Create a Map routine and a Reduce routine
- Wire those into the job.
- (Reduce is optional)

```
public class WordCount {

  /* ... */

  public static void main(String[] args) throws Exception {

    if (args.length != 2) {
      System.err.println("Usage: wordcount <in> <out>");
      System.exit(2);
    }

    Job job = Job.getInstance(new Configuration());
    job.setJobName("wordcount");
    job.setJarByClass(WordCount.class);

    job.setMapperClass(Map.class);
    job.setCombinerClass(Reduce.class);
    job.setReducerClass(Reduce.class);

    job.setOutputKeyClass(Text.class);
    job.setOutputValueClass(IntWritable.class);

    FileInputFormat.setInputPaths(job, new Path(args[0]));
    FileOutputFormat.setOutputPath(job, new Path(args[1]));

    job.submit();
    job.waitForCompletion(true);
  }
}
```

compute • calcul
CANADA

Python/Hadoop count

- Before getting into the Java, let's look at a language probably more familiar to most of us.
- A mapper task - just reads the stdin stream pointed at it, spits out tab-separated lines (word,1)

map.py

```
#!/usr/bin/env python

import sys

for line in sys.stdin:
    line = line.strip()

    words = line.split()
    for word in words:
        print '%s\t%s' % (word, 1)
```

Python/Hadoop count

reduce.py

- A simple reducer
- gets partitioned sorted streams of
  (Hello,1)
  (Hello,1)
  (Goodbye,1)
- and sums the counts
- prints (word,sum) at end

```
#!/usr/bin/env python
import sys

current_word = None
current_count = 0
word = None

for line in sys.stdin:
    line = line.strip()

    word, count = line.split('\t', 1)
    count = int(count)

    if current_word == word:
        current_count += count
    else:
        if current_word:
            print '%s\t%s' % (current_word, current_count)
        current_count = count
        current_word = word

if current_word == word:
    print '%s\t%s' % (current_word, current_count)
```

SciNet compute • calcul CANADA

Python/Hadoop count

Can use this approach in serial using standard shell tools:

```
$ cd wordcount-streaming

$ cat input/*
Hello World Bye World
Hello Hadoop Goodbye Hadoop

$ cat input/* | ./map.py | sort | ./reduce.py
Bye 1
Goodbye   1
Hadoop 2
Hello  2
World  2
```

Python/Hadoop count

- Can also fire this off in parallel with Hadoop
- "streaming interface", designed to work with other languages
- Hadoop decides how many maps, reduces to fire off

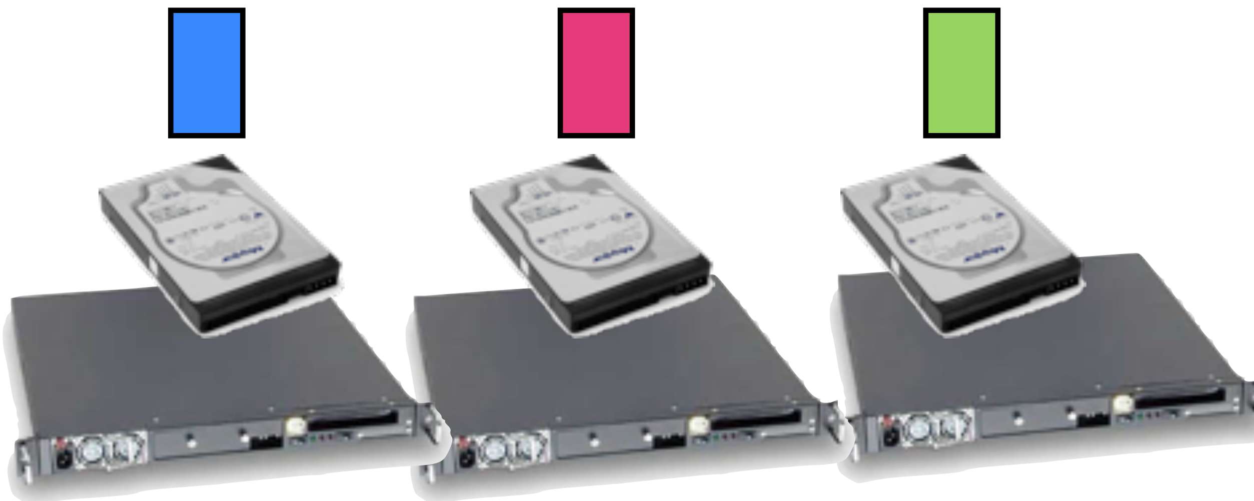
```
$ hadoop jar $(STREAMING_DIR)hadoop-streaming-$(HADOOP_VERSION).jar \
      -file ./map.py       -mapper ./map.py \
      -file ./reduce.py    -reducer ./reduce.py \
      -input $(INPUT_DIR) \
      -output  $(OUTPUT_DIR)
```

- Other interfaces for more programatic interfaces (Pipes - C++; Dumbo - better Python interface, etc)
- Streaming seems to work roughly as well or better

SciNet compute • calcul CANADA

Number of mappers

- Mapping is tightly tied to the Hadoop file system
- Block-oriented
- "Input splits" - blocks of underlying input files
- One mapper handles all the records in one split
- One mapper per input split
- Only one replication is mapped usually



Mapper and I/O

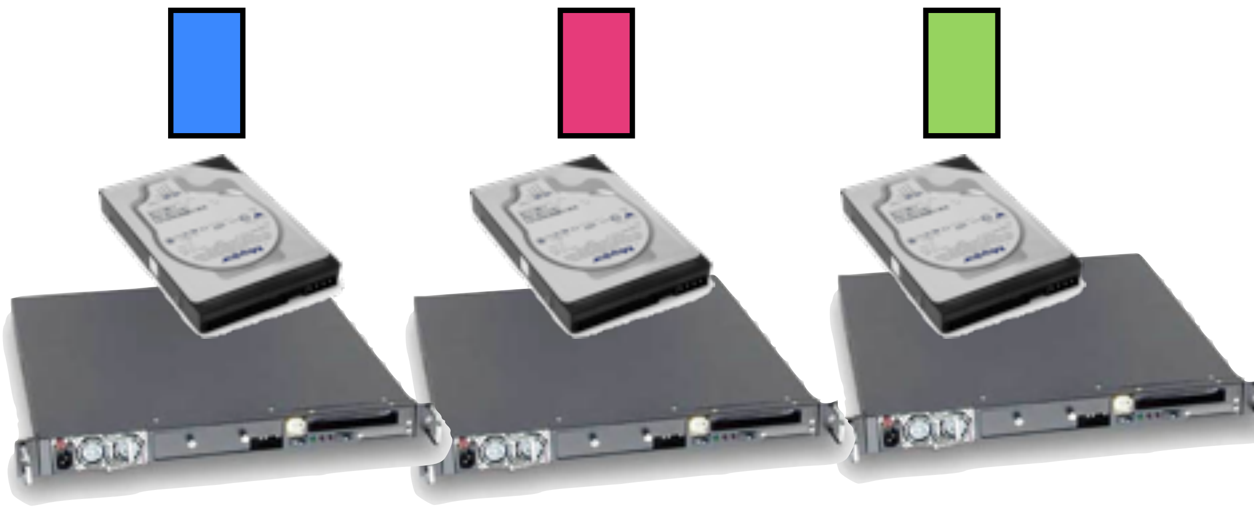
- The code for your mapper processes one record
- The map process executes it for every record in the split
- It gets passed in one (key, value) pair, and updates an “Output Collector” with a new (key, value) pair.

```java
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);
    }
  }
}
```

SciNet compute • calcul CANADA

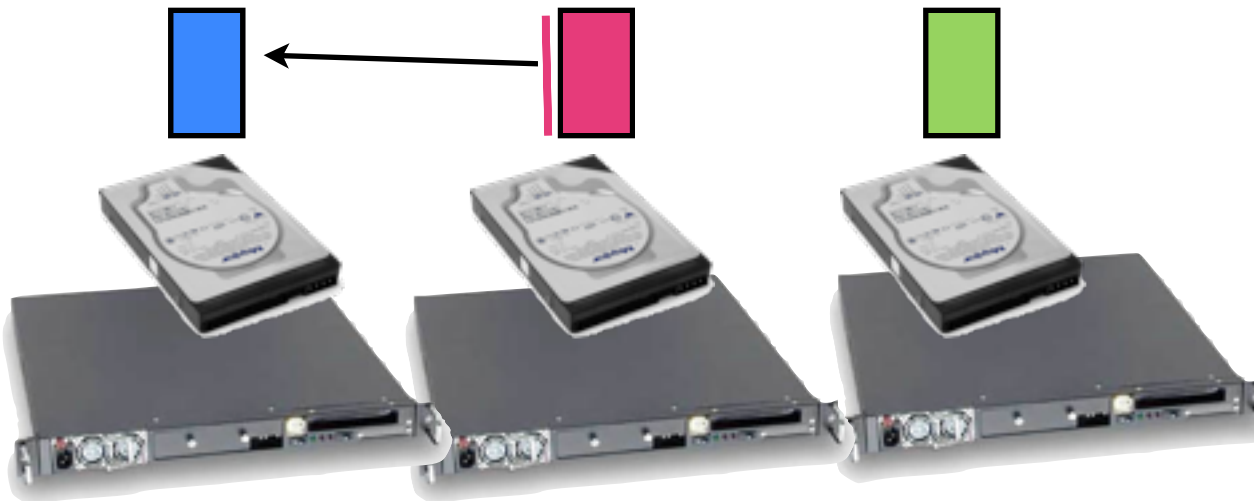
Mapper and I/O

- Mapper works one record at a time
- That means the input file format must have a way to indicate "end of record".
- We're going to be using plain text file, because easy to understand, but there are others (often more appropriate for our examples)



Mapper and I/O

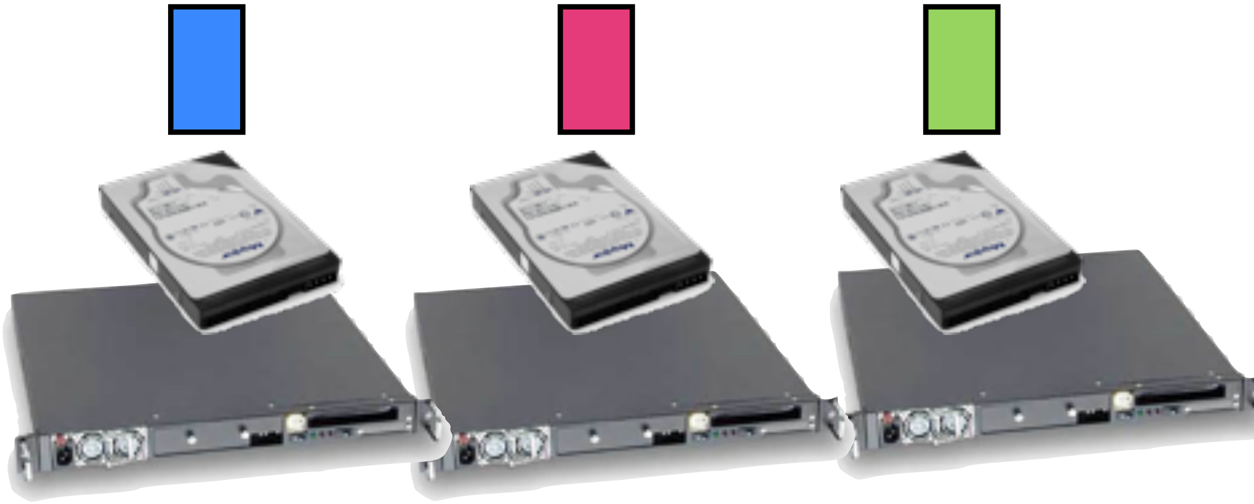
- If record crosses block boundary, must be sent across network
- Another good reason for large blocks - small fraction of data has to be sent



SciNet compute • calcul CANADA

Mapper and I/O

- Mappers can work with compressed a files
- But obviously works best if the compression algorithm is "splittable" - do you need to read the whole file to understand a chunk?
- bzip2 - slow but splittable
- Other possibilities



Mapper and I/O

- Mapper doesn't explicitly do any I/O
- Input is wired up at job configuration time
- Set Input format and input paths

```
public class WordCount {

  /* ... */

  public static void main(String[] args) throws Exception {

    if (args.length != 2) {
      System.err.println("Usage: wordcount <in> <out>");
      System.exit(2);
    }

    Job job = Job.getInstance(new Configuration());
    job.setJobName("wordcount");
    job.setJarByClass(WordCount.class);

    job.setMapperClass(Map.class);
    job.setCombinerClass(Reduce.class);
    job.setReducerClass(Reduce.class);

    job.setOutputKeyClass(Text.class);
    job.setOutputValueClass(IntWritable.class);

    FileInputFormat.setInputPaths(job, new Path(args[0]));
    FileOutputFormat.setOutputPath(job, new Path(args[1]));

    job.submit();
    job.waitForCompletion(true);
  }
}
```

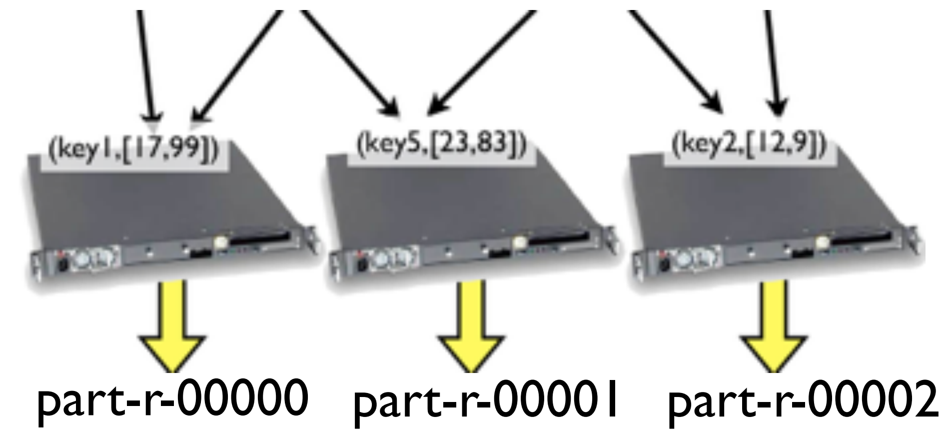
Reducer and I/O

- Similarly, reducer doesn't explicitly do any I/O
- Set the output format, and the output Key/Value types that will be written.
- Send output to an OutputCombiner, and output gets sent out.
- At the end, each reducer writes out its own file, part-r-N

```
public class WordCount {

  /* ... */

  public static void main(String[] args) throws Exception {

    if (args.length != 2) {
      System.err.println("Usage: wordcount <in> <out>");
      System.exit(2);
    }

    Job job = Job.getInstance(new Configuration());
    job.setJobName("wordcount");
    job.setJarByClass(WordCount.class);

    job.setMapperClass(Map.class);
    job.setCombinerClass(Reduce.class);
    job.setReducerClass(Reduce.class);

    job.setOutputKeyClass(Text.class);
    job.setOutputValueClass(IntWritable.class);

    FileInputFormat.setInputPaths(job, new Path(args[0]));
    FileOutputFormat.setOutputPath(job, new Path(args[1]));

    job.submit();
    job.waitForCompletion(true);
  }
}
```

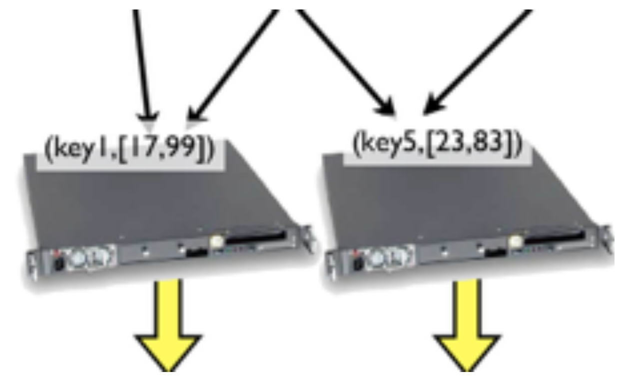
Number of reducers

- Number of mappers set by input splits
- Can suggest reducing that
- Set of reducers is by default chosen based on input size amongst other things
- Our problems here - always so small that only one is used (only part-r-00000)



Number of reducers

- Can explicitly set number of reduce tasks
- Try this - in streaming example, do make run-2reducers
- or in WordCount.java, main, add line job.setNumReduceTasks(2);
- Different reducers get different words (keys), different outputs from these keys
- hdfs dfs -getmerge : gets all files in a directory and cat’s them



part-00000 part-00001

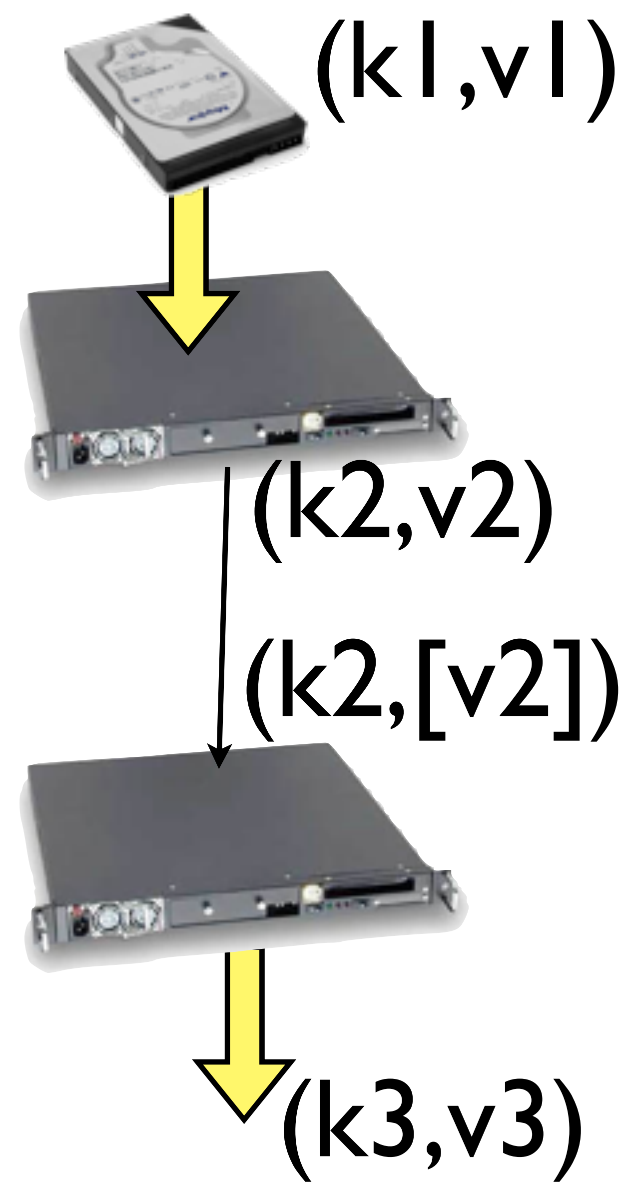
```
Goodbye 1
Hadoop  2
```

```
Bye     1
Hello   2
World   2
```

SciNet compute • calcul CANADA

MapReduce in Java

- In a strongly typed language, we have to pay a bit more attention to types than with just text streams
- Everything's a key-value pair, but don't have to have same type.
- In our examples, always using TextInputFormat, so (k1,v1) is always going to be Object (line # w/in split) and Text, but others could change



MapReduce in Java

- Input types determined input format
- Reduce outputs specified by the Output Key/Value classes
- If not specified, assumed output of mapper (=input of reduce) same as output of reduce. (k2=k3, v2=v3)

```
public class WordCount {

  /* ... */

  public static void main(String[] args) throws Exception {

    if (args.length != 2) {
      System.err.println("Usage: wordcount <in> <out>");
      System.exit(2);
    }

    Job job = Job.getInstance(new Configuration());
    job.setJobName("wordcount");
    job.setJarByClass(WordCount.class);

    job.setMapperClass(Map.class);
    job.setCombinerClass(Reduce.class);
    job.setReducerClass(Reduce.class);

    job.setOutputKeyClass(Text.class);
    job.setOutputValueClass(IntWritable.class);

    FileInputFormat.setInputPaths(job, new Path(args[0]));
    FileOutputFormat.setOutputPath(job, new Path(args[1]));

    job.submit();
    job.waitForCompletion(true);
  }
}
```

Map in Java

- Map implements Mapper<k1,v1,k2,v2>
- Note "special" types - IntWritable, not Integer; Text, not String
- Hadoop comes with its own set of classes which "wrap" standard classes but implement Write methods for serialization (to network or disk).

```
public static class Map
    extends Mapper<Object, Text, Text, IntWritable> {

  private final static IntWritable one = new IntWritable(1);
  private Text word = new Text();

  @Override
  public void map(Object key,
                  Text value,
                  Context context)
    throws IOException, InterruptedException {
    String line = value.toString();
    StringTokenizer tokenizer = new StringTokenizer(line);
    while (tokenizer.hasMoreTokens()) {
      word.set(tokenizer.nextToken());
      context.write(word, one);
    }
  }
}
```

Map in Java

- k2,v2 - Text, IntWriteable
- eg, (“word”, 1)
- Actual work done is very minimal;
- Get the string out of the Text value;
- Tokenize it (split it by spaces)
- While there are more tokens,
- emit (word,one)

```
public static class Map
    extends Mapper<Object, Text, Text, IntWritable> {

  private final static IntWritable one = new IntWritable(1);
  private Text word = new Text();

  @Override
  public void map(Object key,
                  Text value,
                  Context context)
      throws IOException, InterruptedException {
    String line = value.toString();
    StringTokenizer tokenizer = new StringTokenizer(line);
    while (tokenizer.hasMoreTokens()) {
      word.set(tokenizer.nextToken());
      context.write(word, one);
    }
  }
}
```

Reduce in Java

- k2,v2 - Text, IntWriteable (check)
- k3,v3 also Text,IntWriteable
- Incoming values for a given key are pre-concatenated into an iterable
- (couldn't do this for streaming interface; don't know enough about structure of keys/values.)

```java
public static class Reduce
  extends Reducer<Text, IntWritable, Text, IntWritable> {

  @Override
  public void reduce(Text key,
                     Iterable<IntWritable> valueList,
                     Context context)
        throws IOException, InterruptedException {
    int sum = 0;
    Iterator<IntWritable> values = valueList.iterator();
    while (values.hasNext()) {
      sum += values.next().get();
    }
    context.write(key, new IntWritable(sum));
  }
}
```

SciNet compute • calcul CANADA

Reduce in Java

- Work is very simple.
- Operates on a single (k,[v]).
- Loop over values (have to .get() the Integer from the IntWritable)
- sum them up
- Make a new IntWritable with value from sum
- Collect (key,sum)

```java
public static class Reduce
  extends Reducer<Text, IntWritable, Text, IntWritable> {

  @Override
  public void reduce(Text key,
                     Iterable<IntWritable> valueList,
                     Context context)
         throws IOException, InterruptedException {
    int sum = 0;
    Iterator<IntWritable> values = valueList.iterator();
    while (values.hasNext()) {
      sum += values.next().get();
    }
    context.write(key, new IntWritable(sum));
  }
}
```

SciNet compute • calcul CANADA

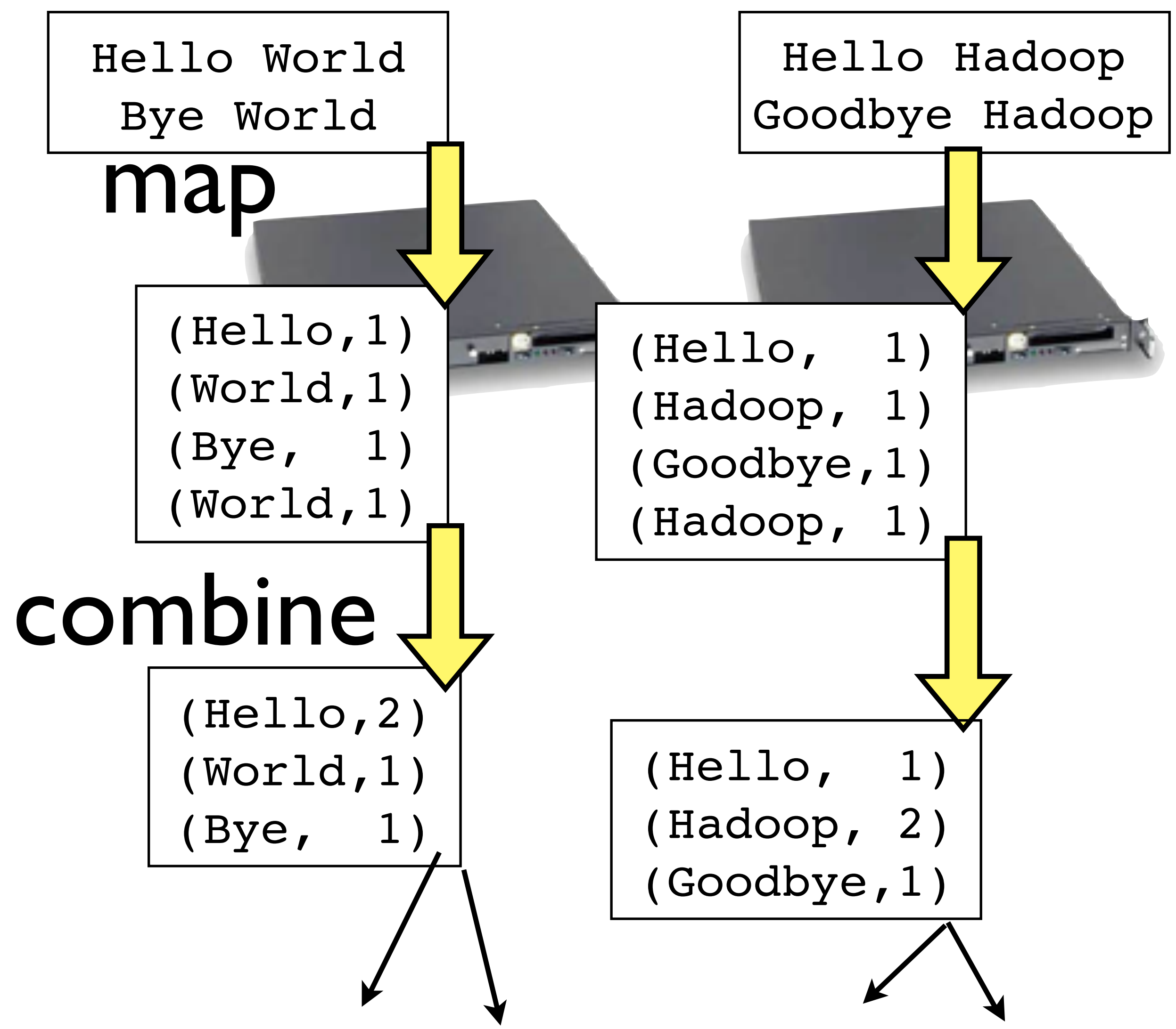
Combiners

- One more useful thing to know
- You can have a "combiner".
- Run by each mapper on the output of the mapper, before its fed to the shuffle.
- Required (k2,[v2])→(k2,v2)

```java
public class WordCount {

  /* ... */

  public static void main(String[] args) throws Exception {

    if (args.length != 2) {
      System.err.println("Usage: wordcount <in> <out>");
      System.exit(2);
    }

    Job job = Job.getInstance(new Configuration());
    job.setJobName("wordcount");
    job.setJarByClass(WordCount.class);

    job.setMapperClass(Map.class);
    job.setCombinerClass(Reduce.class);
    job.setReducerClass(Reduce.class);

    job.setOutputKeyClass(Text.class);
    job.setOutputValueClass(IntWritable.class);

    FileInputFormat.setInputPaths(job, new Path(args[0]));
    FileOutputFormat.setOutputPath(job, new Path(args[1]));

    job.submit();
    job.waitForCompletion(true);
  }
}
```

Combiners

- One more useful thing to know
- You can have a "combiner".
- Run by each mapper on the output of the mapper, before its fed to the shuffle.
- Required (k2,[v2]) → (k2,v2)
- Dumb to send every (the,1) over the network; combine lets you collate the output of each mapper individually before feeding to reducers



Combiners

- In this case, the combiner is just the reducer
- Not all problems lend themselves to the obvious use of a combiner, and in general it won't be identical to the reducer.
- If reducer is commutative and associative, can use as the combiner.

```
public class WordCount {

  /* ... */

  public static void main(String[] args) throws Exception {

    if (args.length != 2) {
      System.err.println("Usage: wordcount <in> <out>");
      System.exit(2);
    }

    Job job = Job.getInstance(new Configuration());
    job.setJobName("wordcount");
    job.setJarByClass(WordCount.class);

    job.setMapperClass(Map.class);
    job.setCombinerClass(Reduce.class);
    job.setReducerClass(Reduce.class);

    job.setOutputKeyClass(Text.class);
    job.setOutputValueClass(IntWritable.class);

    FileInputFormat.setInputPaths(job, new Path(args[0]));
    FileOutputFormat.setOutputPath(job, new Path(args[1]));

    job.submit();
    job.waitForCompletion(true);
  }
}
```

SciNet
compute • calcul CANADA

First hands-on

- More to get you into the mode of writing Java
- We have the same example in wordcount-worksheet, but with the guts of map, reduce left out.
- Practice writing the code. Feel free to google for how to do things in Java, but don't just blast the lines from examples...
- Can use your favourite local editor and scp file to VM

```java
public static class Map
    extends Mapper<Object, Text, Text, IntWritable> {

  private final static IntWritable one = new IntWritable(1);
  private Text word = new Text();

  @Override
  public void map(Object key,
                  Text value,
                  Context context)
      throws IOException, InterruptedException {
    String line = value.toString();
    StringTokenizer tokenizer = new StringTokenizer(line);
    while (tokenizer.hasMoreTokens()) {

  /*  ... context.write( , ) */

    }
  }
}

public static class Reduce
  extends Reducer<Text, IntWritable, Text, IntWritable> {

  @Override
  public void reduce(Text key,
                     Iterable<IntWritable> valueList,
                     Context context)
         throws IOException, InterruptedException {
    int sum = 0;
    Iterator<IntWritable> values = valueList.iterator();
    while (values.hasNext()) {
      /* update sum */
    }
    /* context.write( , ) */
  }
}
```

First hands-on

- To copy files back and forth, find the IP a of the VM
- (We enabled this in virtualbox with the IO APIC/Adapter 2 stuff)
- hadoop-user/hadoop

VM:

```
$ ifconfig | grep 192
          inet addr:192.168.56.101 [...]
```

Host

```
$ scp WordCount.java hadoop-user@192.168.56.101:
 hadoop-user@192.168.56.101's password: hadoop
```

SciNet compute • calcul CANADA

Web Monitor

- Open browser on laptop
- go to (e.g.) http://192.168.56.101:8088
- Look at the previous jobs run
- Hadoop has to keep track of the running of individual map, reduce tasks and job status for fault-tolerance reasons
- Presents a nice web interface to the hadoop cluster

Beyond WordCount

- Let's start going a little bit beyond simple wordcount
- `cd ~/inverted-index`
  `make run`
- First, take a look at word count broken down by document
- 5 new papers each from 8 disciplines, taken from arxiv, pdftotext

astro_01

```
abstract galaxy
supernova star
```

genomics_03

```
abstract gene
expression dna
```

output/part-00000

```
astro_01 abstract 1
astro_01 galaxy 1
genomics_03 abstract 1
genomics_03 gene 1
```

WordCount by Doc

- Map is a little more sophisticated - strips out "stop words" ('the', 'and', ...)
- Also only pay attention to "words" > 3 letters (strip out noise from pdf-to-text conversion - eqns, etc)

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

  FileSplit filesplit = (FileSplit)context.getInputSplit();
  String fileName = filesplit.getPath().getName();

  String line = (value.toString()).replaceAll("[^a-z\\sA-Z]"
  StringTokenizer tokenizer = new StringTokenizer(line);
  while (tokenizer.hasMoreTokens()) {
    String newWord = (tokenizer.nextToken()).toLowerCase();
    if ( (!stopwords.contains(newWord)) && (newWord.length()
        word.set( fileName + " " + newWord );
        context.write(word, one);
    }
  }
}
```

SciNet
compute • calcul
CANADA

WordCount by Doc

- Mapper: while the value here is still one, the key is now filename + “ ” + word
- (why?)

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

  FileSplit filesplit = (FileSplit)context.getInputSplit();
  String fileName = filesplit.getPath().getName();

  String line = (value.toString()).replaceAll("[^a-z\\sA-Z]"
  StringTokenizer tokenizer = new StringTokenizer(line);
  while (tokenizer.hasMoreTokens()) {
    String newWord = (tokenizer.nextToken()).toLowerCase();
    if ( (!stopwords.contains(newWord)) && (newWord.length()
        word.set( fileName + " " + newWord );
        context.write(word, one);
    }
  }
}
```

compute • calcul CANADA

WordCount by Doc

- Reducer is exactly the same

```
public static class Reduce
    extends Reducer<Text, IntWritable, Text, IntWritable> {

  @Override
  public void reduce(Text key,
                     Iterable<IntWritable> valueList,
                     Context context) throws IOException, InterruptedException {
    int sum = 0;
    Iterator<IntWritable> values = valueList.iterator();
    while (values.hasNext()) {
      sum += values.next().get();
    }
    context.write(key, new IntWritable(sum));
  }
}
```

Inverted Index:

- Want to use this as a starting point to build an inverted index
- For each word, in what documents does it occur?
- What is going to be the key out of the mapper? The value?
- What is going to be the reduction operation?

astro_01

```
abstract galaxy
supernova star
```

genomics_03

```
abstract gene
expression dna
```

output/part-00000

```
abstract   astro_01 genomics_03
galaxy     astro_01
gene       genomics_03
supernova  astro_01
expression genomics_03
```

Hands on:

- Implement the inverted index
- For now, don't worry about repeated items
- InvertedIndex.java
- Test with `make runinverted`

astro_01

```
abstract galaxy
supernova star
```

genomics_03

```
abstract gene
expression dna
```

output/part-00000

```
abstract   astro_01 genomics_03
galaxy     astro_01
gene       genomics_03
supernova  astro_01
expression genomics_03
```

Document Similarity

- Wordcount-by-document:
- "Bag of words" approach
- Document is characterized by its wordcounts
- Can find similarity of two documents through normalized dot product of their vector representation.

astro_01	**genomics_03**
`abstract galaxy supernova expression`	`abstract gene expression dna`

astro_01 {abstract:1, galaxy:1, supernova:1, star:1}

genomics_03 {abstract:1, gene:1, expression:1, dna:1}

$$S_{a,g} = \frac{\mathbf{w_a} \cdot \mathbf{w_g}}{||w_a|| \cdot ||w_g||}$$

SciNet compute • calcul CANADA

Document Similarity

astro_01 **genomics_03**

```
abstract galaxy
supernova expression
```

```
abstract gene
expression dna
```

astro_01 {abstract:1, galaxy:1, supernova:1, star:1}

genomics_03 {abstract:1, gene:1, expression:1, dna:1}

```
cd ~/document-similarity
make
```

$$S_{a,g} = \frac{\mathbf{w_a} \cdot \mathbf{w_g}}{||w_a|| \cdot ||w_g||}$$

SciNet
compute • calcul
CANADA

Document Similarity

- Wordcount-by-document:
- "Bag of words" approach
- Document is characterized by its wordcounts
- Can find similarity of two documents through normalized dot product of their vector representation.

	abstract	galaxy	expression	gene	supernova	dna	star
a=	1	1			1		1
g=	1		1	1		1	

$$
\begin{aligned}
S_{a,g} &= \frac{\mathbf{w_a} \cdot \mathbf{w_g}}{||w_a|| \cdot ||w_g||} \\
&= \frac{1}{2 \cdot 2} \\
&= \frac{1}{4}
\end{aligned}
$$

Document Similarity

- So taken the bags-of-words as a given, how do we do the computation?
- What's the map phase, and the reduce phase?

	abstract	galaxy	expression	gene	supernova	dna	star
a=	1	1			1		1
g=	1		1	1		1	

$$
\begin{aligned}
S_{a,g} &= \frac{\mathbf{w_a} \cdot \mathbf{w_g}}{||w_a|| \cdot ||w_g||} \\
&= \frac{1}{2 \cdot 2} \\
&= \frac{1}{4}
\end{aligned}
$$

Document Similarity

- Easiest to think about the reduce phase first.
- What is going to be the single computation done by a single reducer?
- And what information does it need to perform that computation?

	abstract	galaxy	expression	gene	supernova	dna	star
a=	1	1			1		1
g=	1		1	1		1	

$$
\begin{aligned}
S_{a,g} &= \frac{\mathbf{w_a} \cdot \mathbf{w_g}}{||w_a|| \cdot ||w_g||} \\
&= \frac{1}{2 \cdot 2} \\
&= \frac{1}{4}
\end{aligned}
$$

Document Similarity: Reducer

- The single piece of computation that needs to be done at the reduce stage are the matrix elements Sa,g.
- The computation is straightforward.
- What is the key?
- What data does it need?

Reducer

$$S_{a,g} = \frac{\mathbf{w_a} \cdot \mathbf{w_g}}{||w_a|| \cdot ||w_g||}$$

Means key is... ?

Means data it needs is... ?

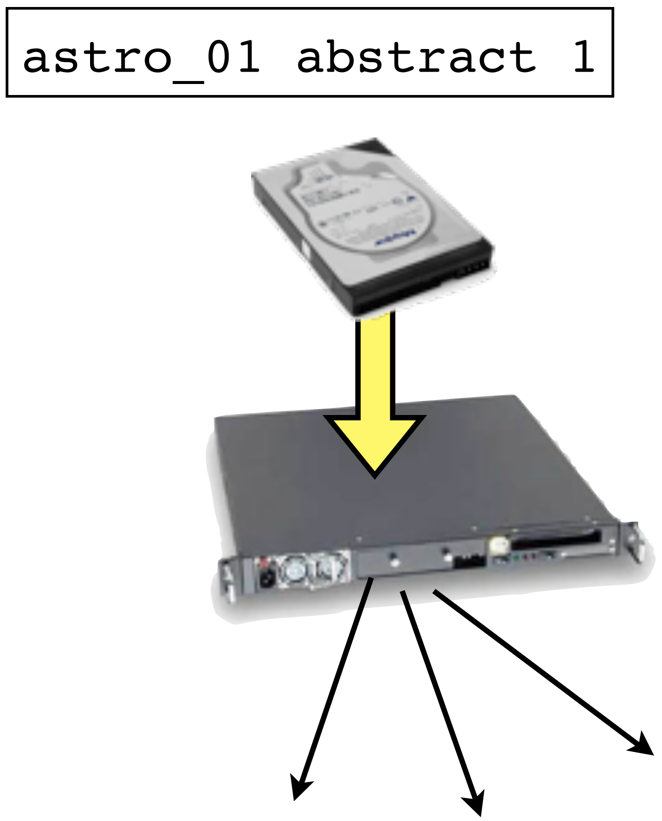
Document Similarity: Mapper

- It’s the mapper’s job to read in the data and direct it to the correct reducer by setting the key
- So mapper reads in (astro_01, “abstract 1”).
- Which reducer needs that information?

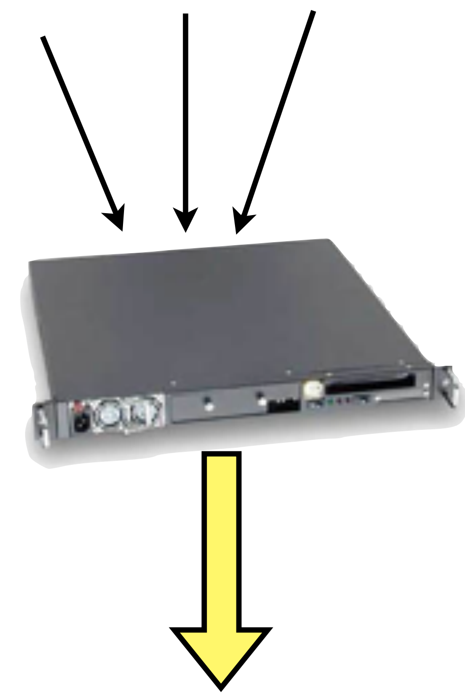
```
astro_01 abstract 1
astro_01 galaxy 1
genomics_03 abstract 1
```

Document Similarity: Mapper

- Map phase: "broadcast" (astro_01, "abstract 1") to all key pairs that will need astro_01
- key: "astro_01 x", x=astro_02, astro_03, ...genomics_01,...
- value: "astro_01 abstract 1"
- (We're just putting everything in text strings here but we could have keys and values which were tuples...)



Document Similarity: Reducer



- Reducer: Collect all (say) “astro_01 genomics_03” keys
- Sort into elements for the two documents
- Calculate the result

Document Similarity: Mapper

- Map: loop over documents
- emit value for each document pair

```java
public void map(Object key,
                Text value,
                Context context)
                throws IOException, InterruptedException {

  String line = value.toString().trim();
  String[] items = line.split("\\s+");
  String doc = items[0];

  for ( String otherdocs : documents ) {
      Text docpair = new Text();
      int order = otherdocs.compareTo(doc);
      if ( order < 0 ) {
            docpair.set(otherdocs + " " + doc);
            context.write(docpair, value);
      } else if ( order > 0 ) {
            docpair.set(doc + " " + otherdocs);
            context.write(docpair, value);
      }
    }
}
```

Document Similarity: Reducer

- Reducer:
  - Put values into appropriate sparse vector
  - (Parsing is just because we're using text for everything, which you really wouldn't do)

```
public void reduce(Text key,
                   Iterable<Text> valueList,
                   Context context) throws IOException, I

  Double sum = 0.0;
  String docs[] = (key.toString()).split("\\s+");
  HashMap<String,Double> doc1words = new HashMap<String,D
  HashMap<String,Double> doc2words = new HashMap<String,D
  Iterator<Text>values = valueList.iterator();

  while (values.hasNext()) {
    String line = values.next().toString().trim();
    String terms[] = line.split("\\s+");

    if (terms.length != 3) continue;

    String docname = terms[0];
    String word    = terms[1];
    Double count   = Double.parseDouble(terms[2]);

    if ( docname.equals(docs[0]) ) {
        doc1words.put(word, count);
    } else {
        doc2words.put(word, count);
    }
  }
```

compute • calcul CANADA

Document Similarity: Reducer

- Then the computation is easy.

```
    Double doc1mag = 0.;
    Double doc2mag = 0.;

    for ( Double value : doc1words.values() ) {
      doc1mag += value*value;
    }
    doc1mag = Math.sqrt(doc1mag);

    for ( Double value : doc2words.values() ) {
      doc2mag += value*value;
    }
    doc2mag = Math.sqrt(doc2mag);

    for ( String word : doc1words.keySet() ) {
      if (doc2words.containsKey(word)) {
          sum += doc1words.get(word)*doc2words.get(word);
      }
    }

    context.write( key, new DoubleWritable(sum/(doc1mag*doc2mag)) );
  }
```

Document Similarity: But where did we get..

`astro_01` {abstract:1, galaxy:1, supernova:1, star:1}

`genomics_03` {abstract:1, gene:1, expression:1, dna:1}

But we need as input:

- The wordcounts by document
- The list of documents

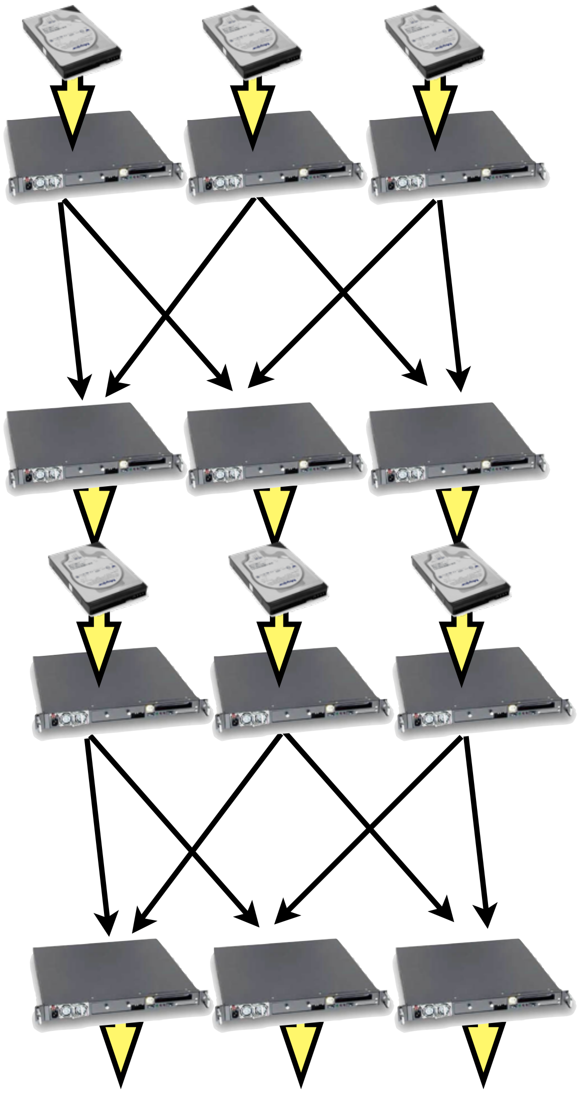
Where do they come from?

```
"astro_01", "astro_02", "astro_03", "astro_04",
    "astro_05", "cell_bio_01", "cell_bio_02", "cell_bio_03",
    "cell_bio_04", "cell_bio_05", "computational_finance_01",
    "computational_finance_02", "computational_finance_03",
    "computational_finance_04", "computational_finance_05",
    "crypto_01", "crypto_02", "crypto_03", "crypto_04", "crypto_05",
    "databases_03", "databases_04", "databases_05", "databases_01",
    "databases_02", "genomics_01", "genomics_02", "genomics_03",
    "genomics_04", "genomics_05", "pdes_01", "pdes_02", "pdes_03",
    "pdes_04", "pdes_05", "robotics_01", "robotics_02", "robotics_03",
```

SciNet compute • calcul CANADA

Document Similarity: But where did we get..

- Chains of Map-Reduce Jobs!
- 1st pass - wordcounts, document list
- 2nd pass - similarity scores
- Can do this programatically (within main), or just by running 2 hadoop jobs...



Document Similarity: But where did we get..

- Chains of Map-Reduce Jobs!
- 1st pass - wordcounts
- 2nd pass - similarity scores

```
BASE_DIR    = /user/$(USER)/document-similarity/
INPUT_DIR    = $(BASE_DIR)/input
INTERMEDIATE_DIR = $(BASE_DIR)/intermediate
OUTPUT_DIR   = $(BASE_DIR)output
OUTPUT_FILE = $(OUTPUT_DIR)/part-00000

run: wordcount.jar similarity.jar
	hadoop dfs -test -e $(INPUT_DIR)/ \
	    || hadoop dfs -put input $(BASE_DIR)
	hadoop jar wordcount.jar org.hpcs2013.WordCount \
	       $(INPUT_DIR) $(INTERMEDIATE_DIR)
	hadoop jar similarity.jar org.hpcs2013.Similarity \
	       $(INTERMEDIATE_DIR) $(OUTPUT_DIR)
	hadoop dfs -cat $(OUTPUT_FILE) | sort -n -k 3
```

SciNet compute • calcul CANADA

A note on similarity

- Ignore the normalization for a second
- Just the dot products
- What we've done is a sparse matrix multiplication entirely in Hadoop.

$$S_{i,j} = \mathbf{w_i} \cdot \mathbf{w_j}$$
$$S = WW^T$$

Matrix multiplication

- `cd ~/matmult`
- Reads in matrix name, rows, columns
- Hands on - fill in the map, reduce.

`input/part-00000`

```
A 0 52 1
A 1 59 1
A 10 86 1
A 11 92 1
A 12 39 1
A 13 44 1
A 14 57 1
A 15 55 1
...
B 16 95 1
B 26 73 1
B 27 22 1
```

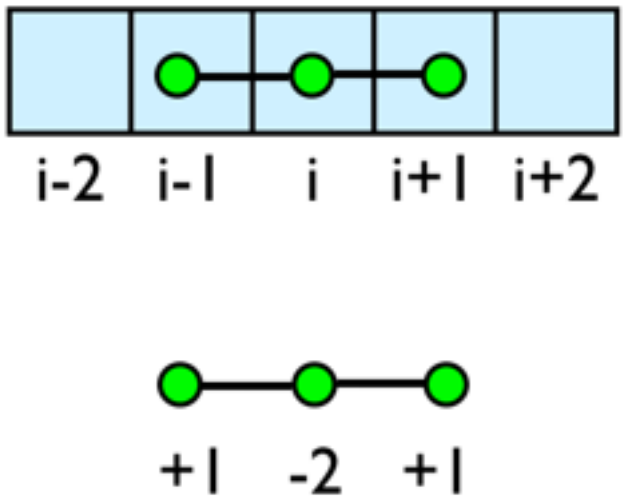
SciNet compute • calcul CANADA

One-D Diffusion

$$\left.\frac{d^2Q}{dx^2}\right|_i \approx \frac{Q_{i+1}-2Q_i+Q_{i-1}}{\Delta x^2}$$

```
cd ~/diffuse
make clean
make
```

- Implements a 1d diffusion PDE



One-D Diffusion

Inputs:

- Pre-broken up domain
- 1d gaussian
- constant diffusion - should maintain Gaussianity

What is the map?

What is the reduce?

```
 0: 0.0050365 0.00709477 0.01360237 ...
1: 0.16004214 0.19533521 0.28114455 ...
2: 0.84731875 0.89604445 0.96817042 ...
3: 0.74742274 0.68483447 0.55549607 ...
4: 0.10984817 0.08720647 0.05310277 ...
```

Questions?