Hadoop for HPCers:
A Hands-On Introduction

Jonathan Dursi, SciNet
Michael Nolta, CITA

Part I: Overview, HDFS

Agenda

- VM Test
- High Level Overview
- Hadoop FS
- Map Reduce
- Hadoop MR + Python
- Pig
- YARN;
- Spark

SciNet compute • calcul CANADA

Detailed VM instructions

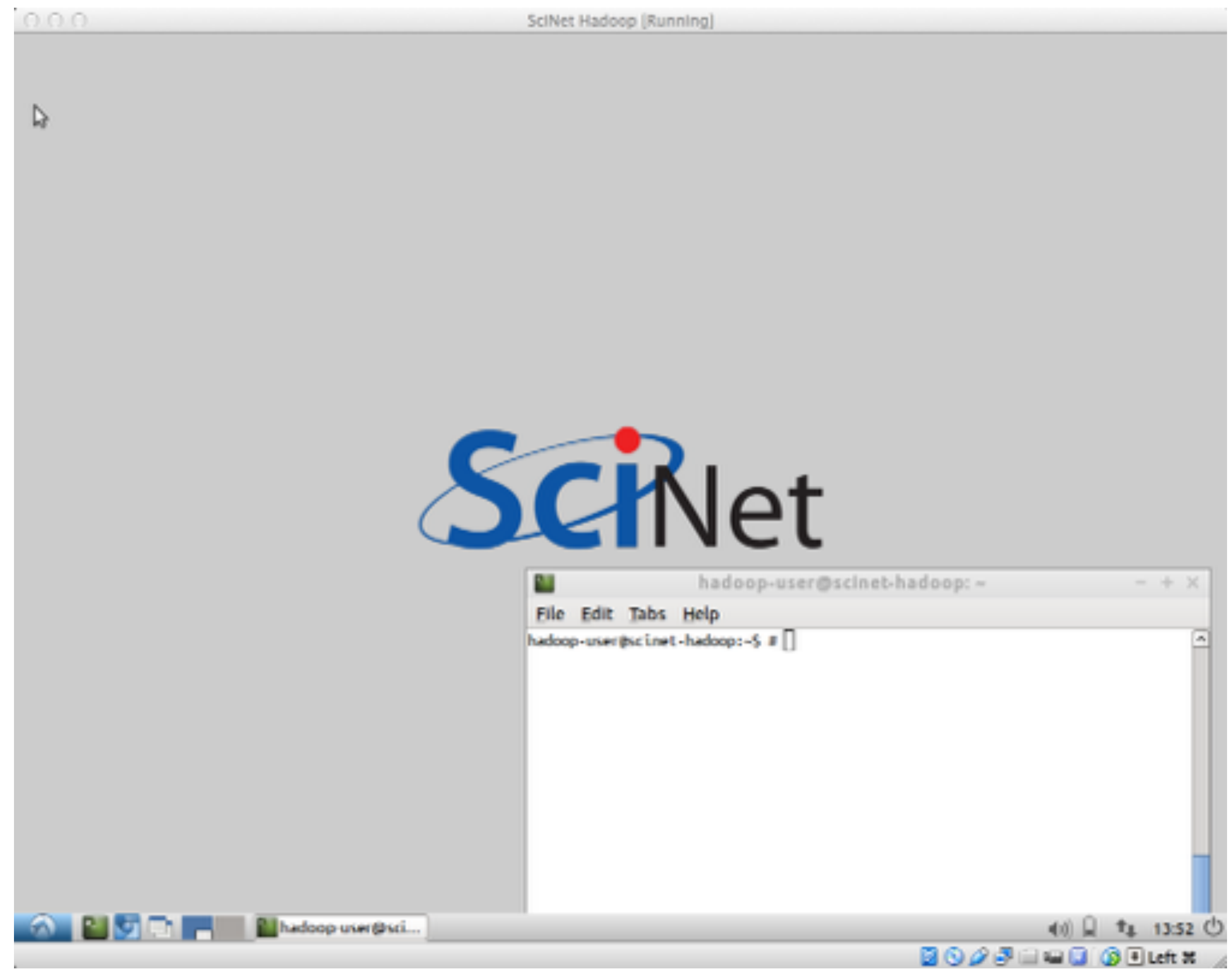
- Install VirtualBox (free) for your system.
- Download the course appliance from
  https://support.scinet.utoronto.ca/~ljdursi/Hadoop/VMs/GUI/GUI-VM.ova
- Start Virtual box; click "New"; give your VM a name. Select "Linux" as Type, and "Ubuntu" as Version. Give your VM at least 2048MB RAM, more would be better.
- File -> Import an Appliance, and import the .ova file you've downloaded
- **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 vagrant, password is vagrant.
- Open a terminal; run "source bin/init.sh"

Let's Get Started!

- Fire up your course VM
- Open terminal; type

```
source bin/init.sh
cd wordcount/streaming
make
```

- You've run your (maybe) first Hadoop job!



Hadoop

- 2007 OSS implementation of 2004 Google MapReduce paper
- Originally consisted of distributed filesytem HDFS, core runtime, an implementation of Map-Reduce.
- Hardest to understand for HPCers: Java
- We'll mainly be working in Python

Hadoop Ecosystem

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

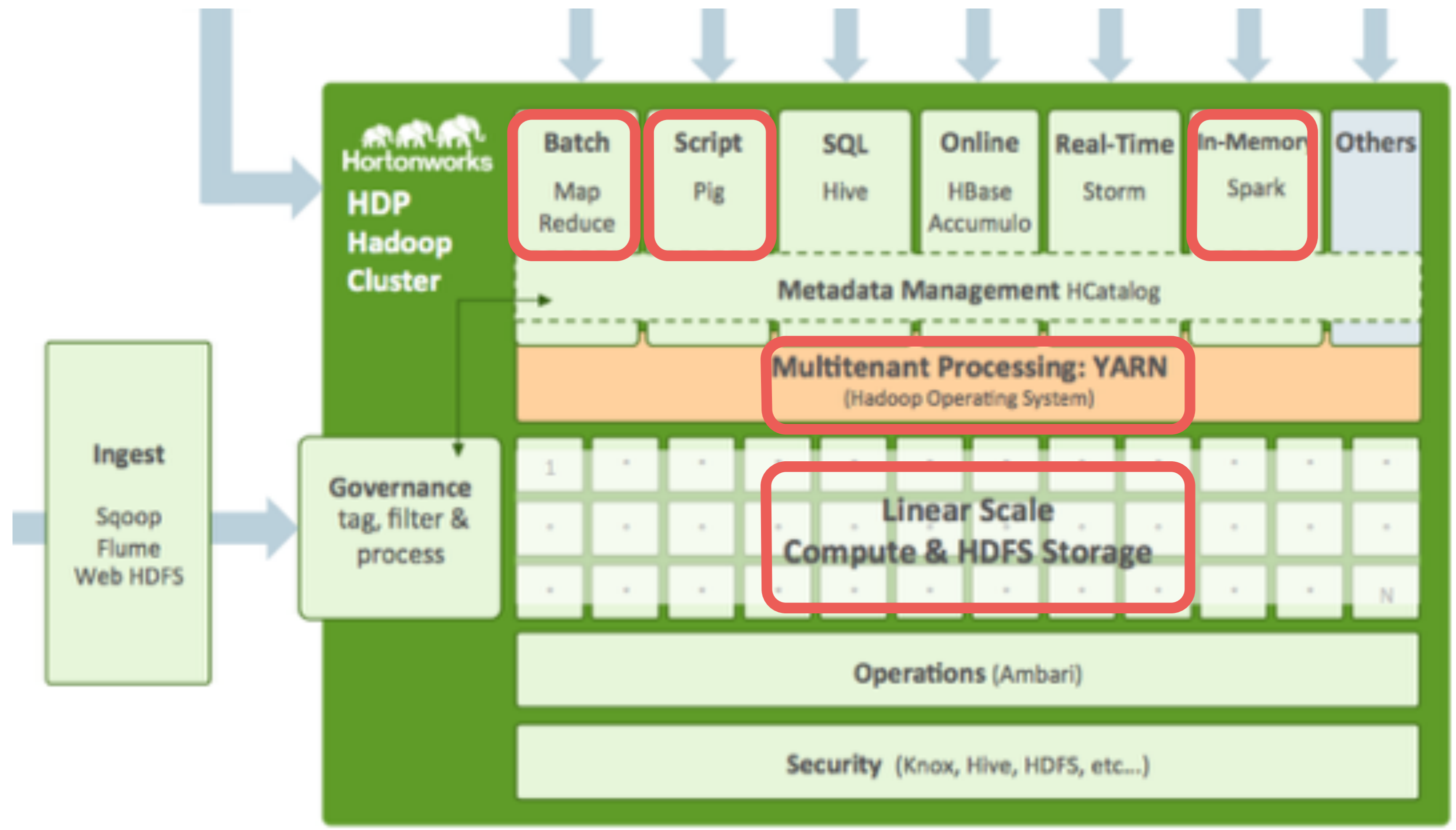
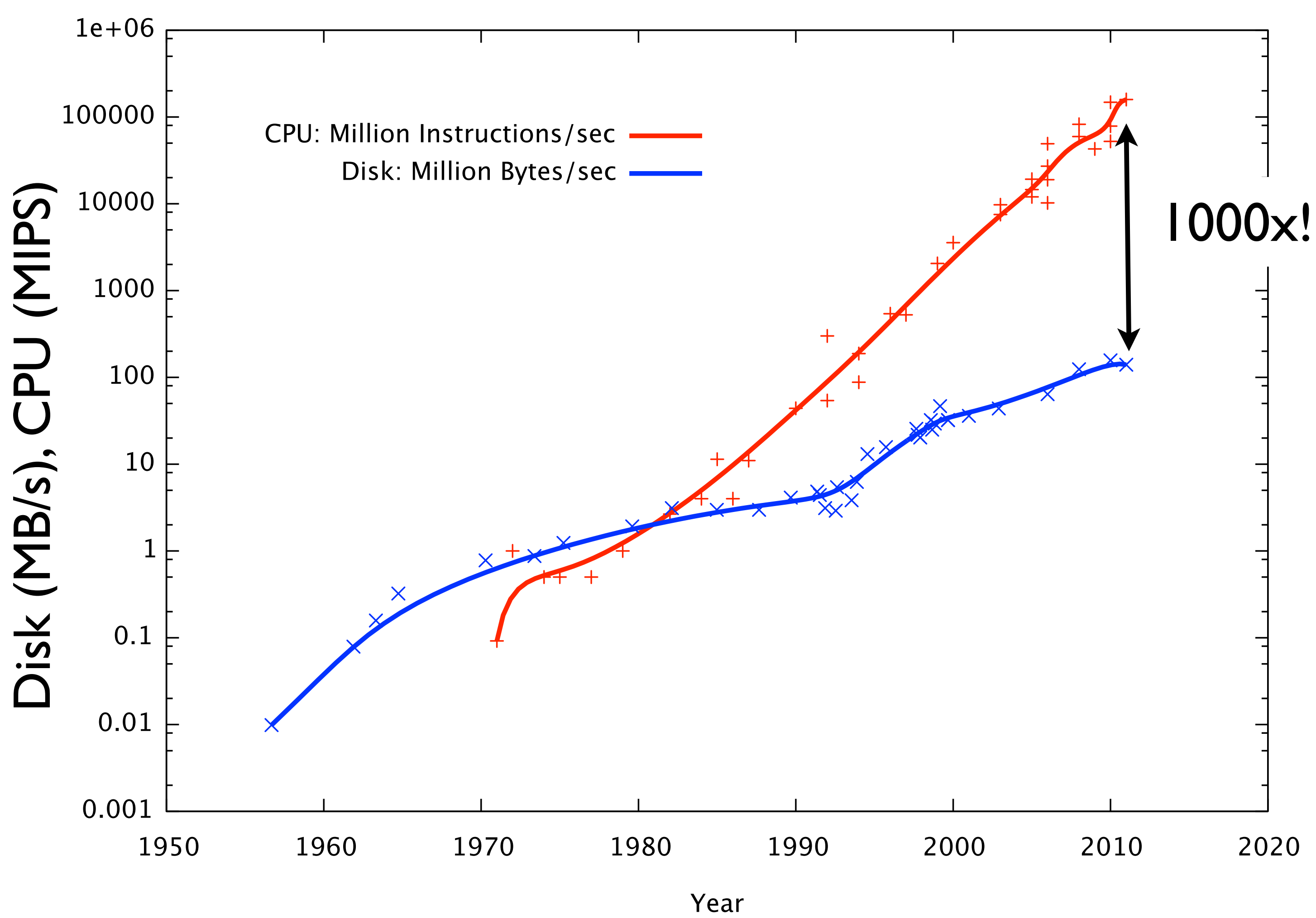


Image from
http://hortonworks.com/industry/manufacturing/

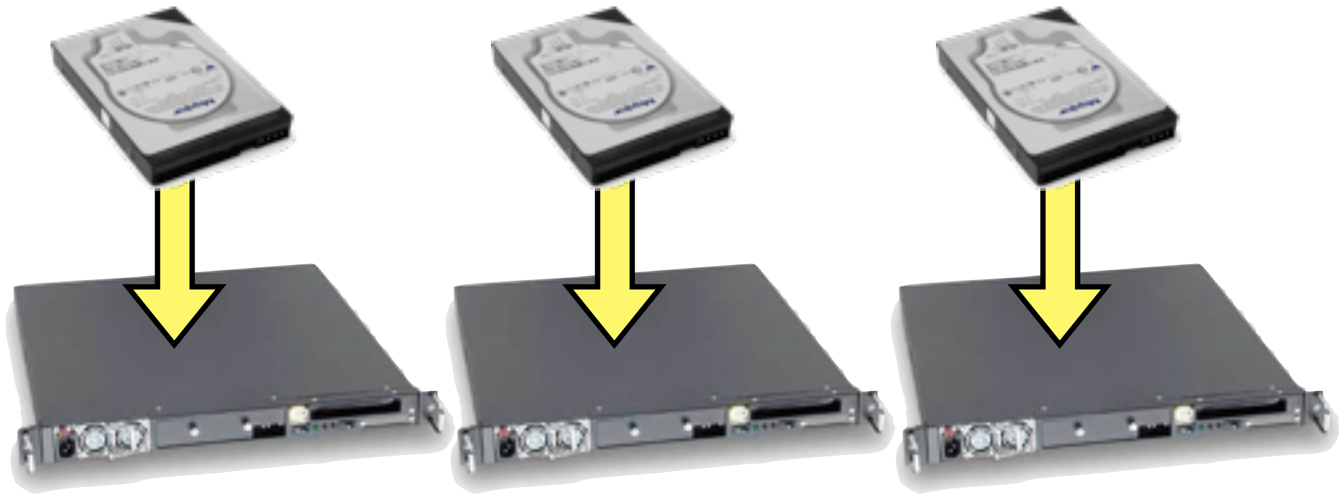
Data Intensive Computing

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



Scale-Out

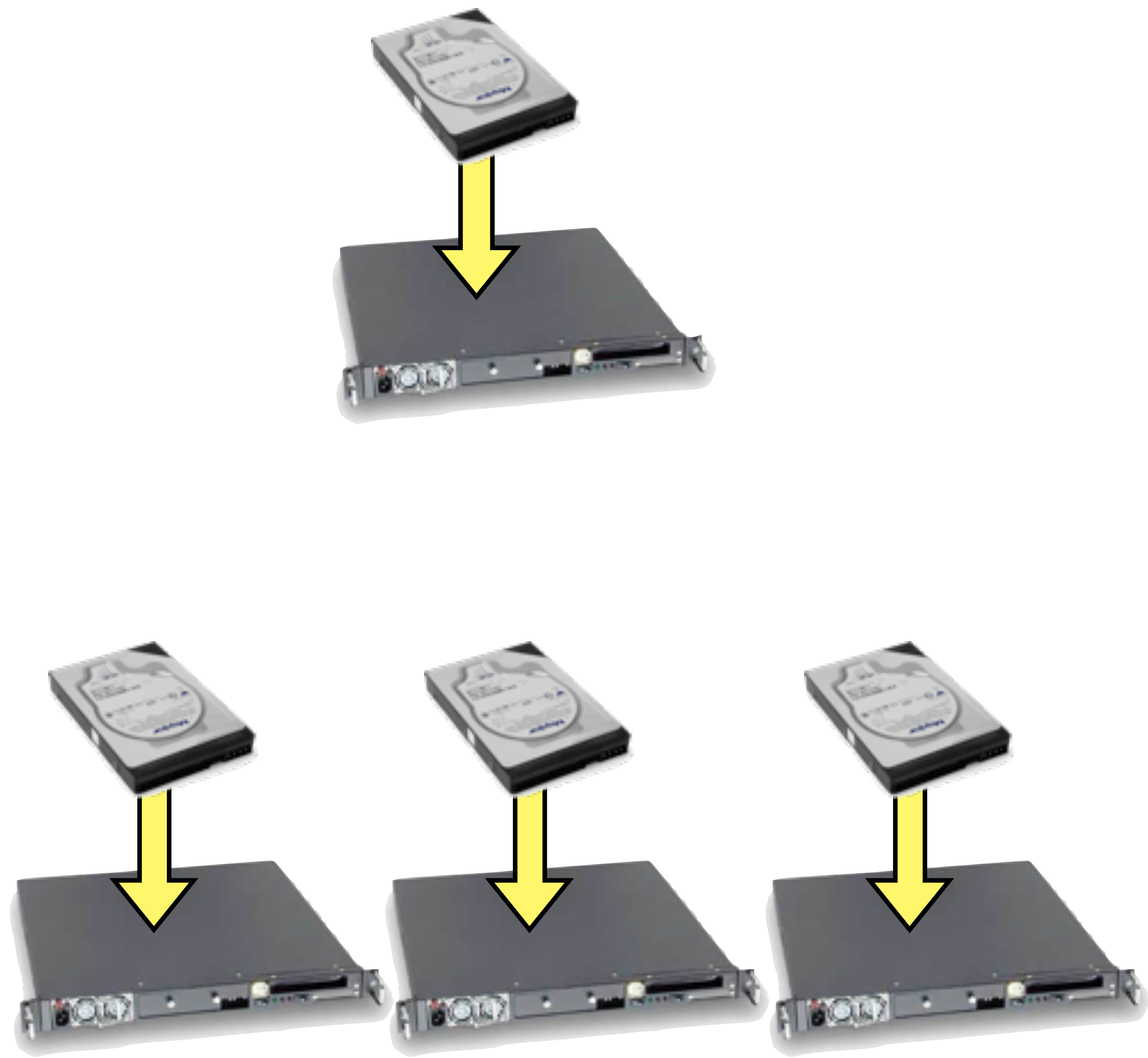
- Disk streaming speed ~ 50MB/s
- 3TB =17.5 hrs
- 1PB = 8 months
- Scale-out (weak scaling) - filesystem distributes data on ingest



SciNet
compute • calcul
CANADA

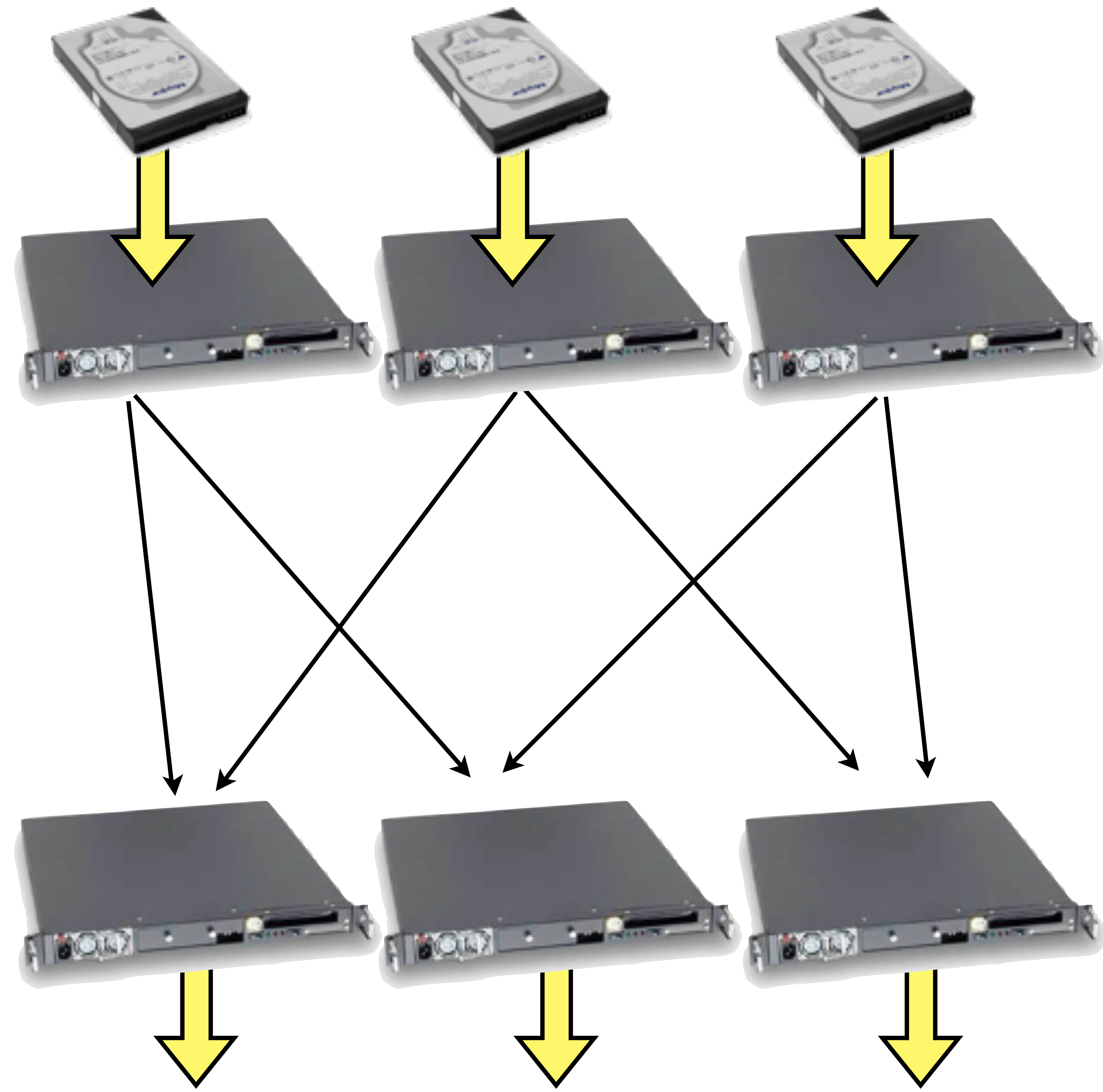
Scale-Out

- Seeking too slow
  - ~10ms for a seek
  - Enough time to read half a megabyte
- Batch processing
- Go through entire data set in one (or small number) of passes



Combining results

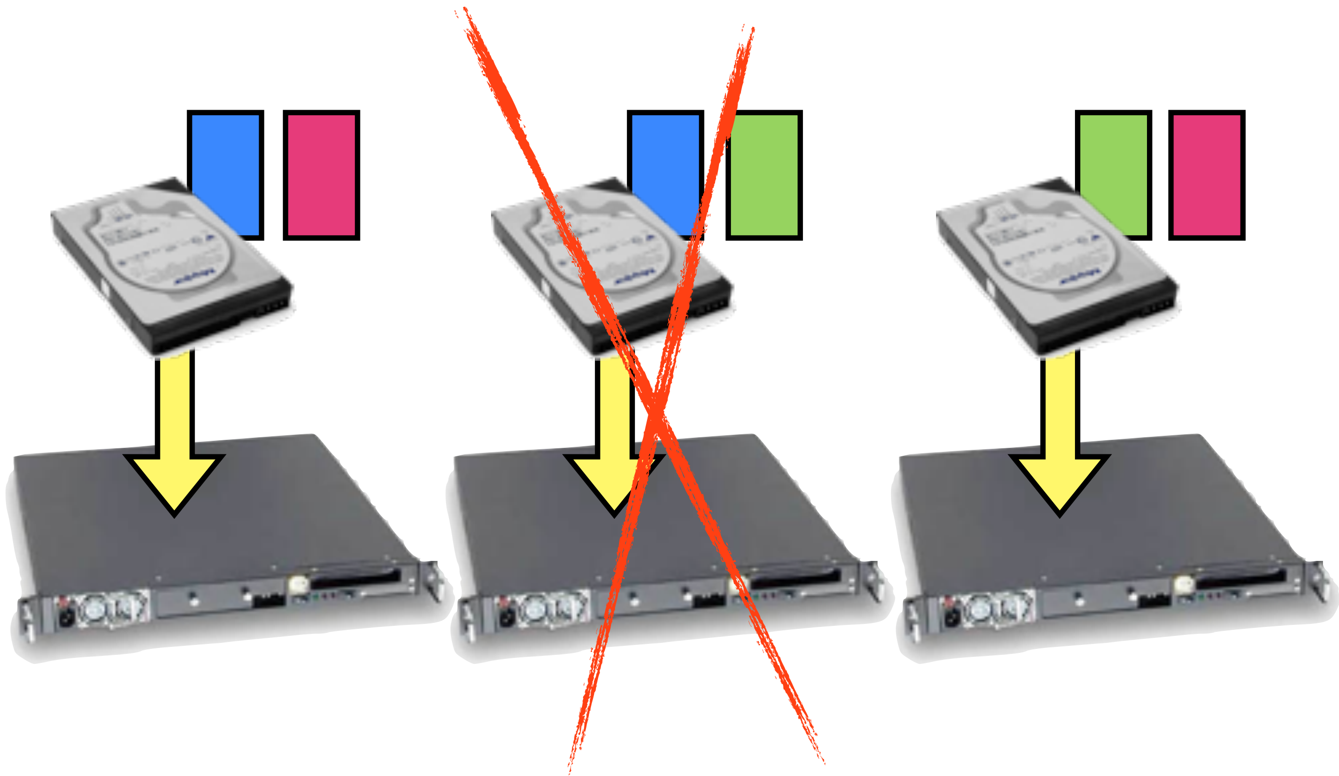
- Each node pre-processes its local data
- Shuffles its data to a small number of other nodes
- Final processing, output is done there



SciNet
compute • calcul
CANADA

Fault Tolerance

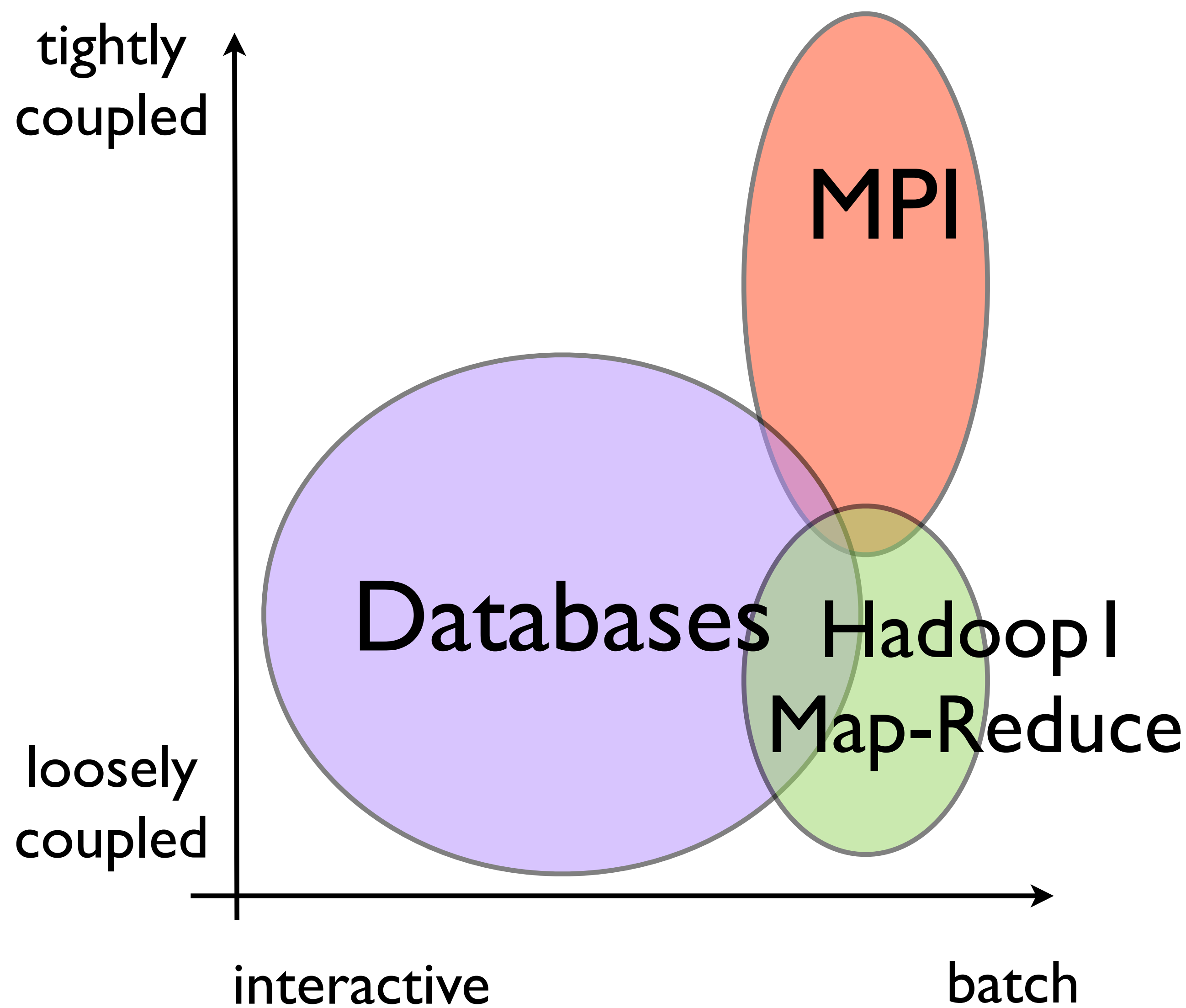
- Data also replicated upon ingest
- Runtime watches for dead tasks, restarts them on live nodes
- Re-replicates



SciNet compute • calcul CANADA

What is it good at?

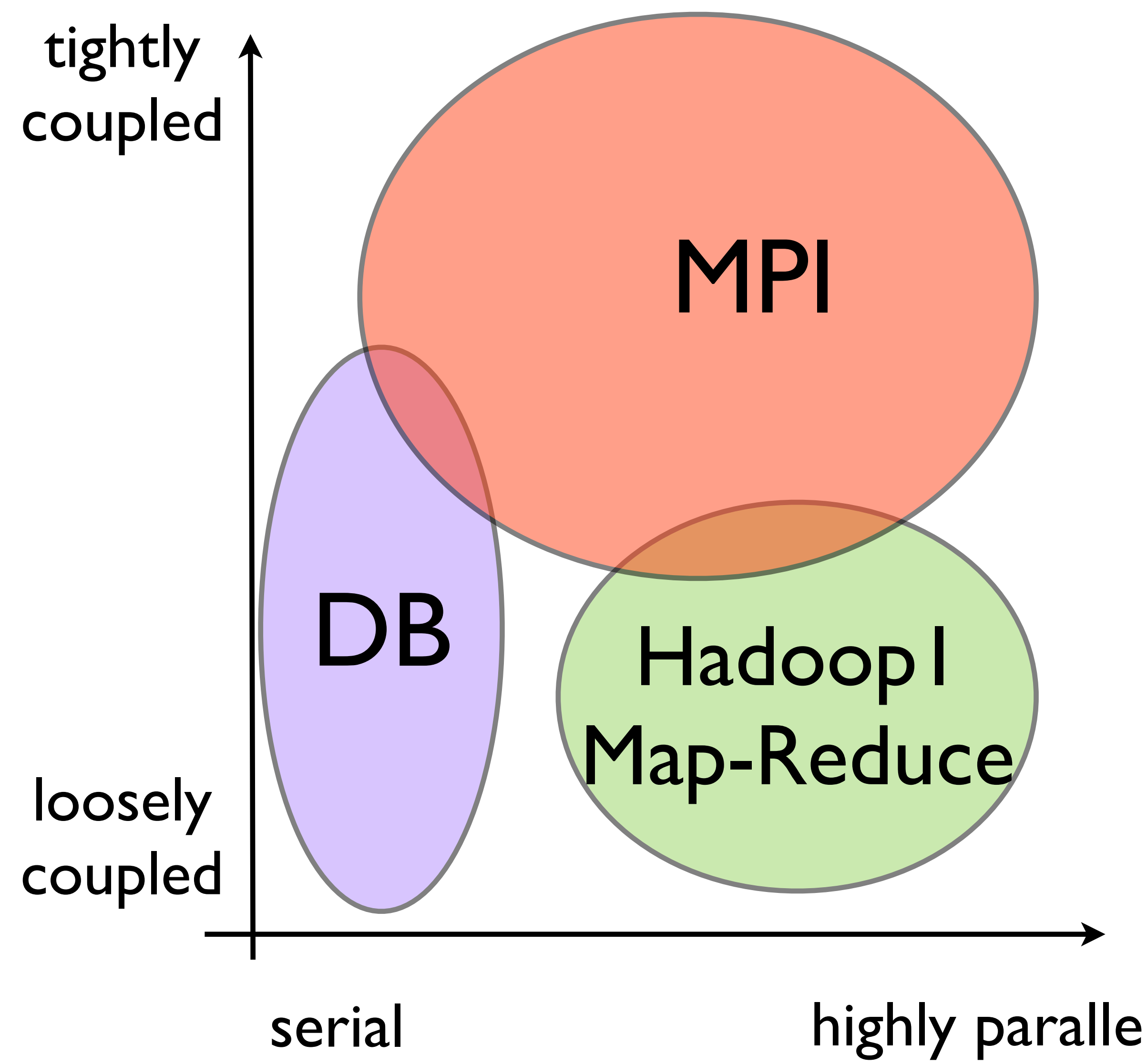
- “Classic” Hadoop 1.x is all about batch processing of massive amounts of data
- (Not much point below ~1TB)
- Map-Reduce is relatively loosely coupled; one “shuffle” phase.
- Very strong weak scaling in this model - more data, more nodes.
- Batch: process all data in one go w/ classic Map Reduce
-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



SciNet compute • calcul CANADA

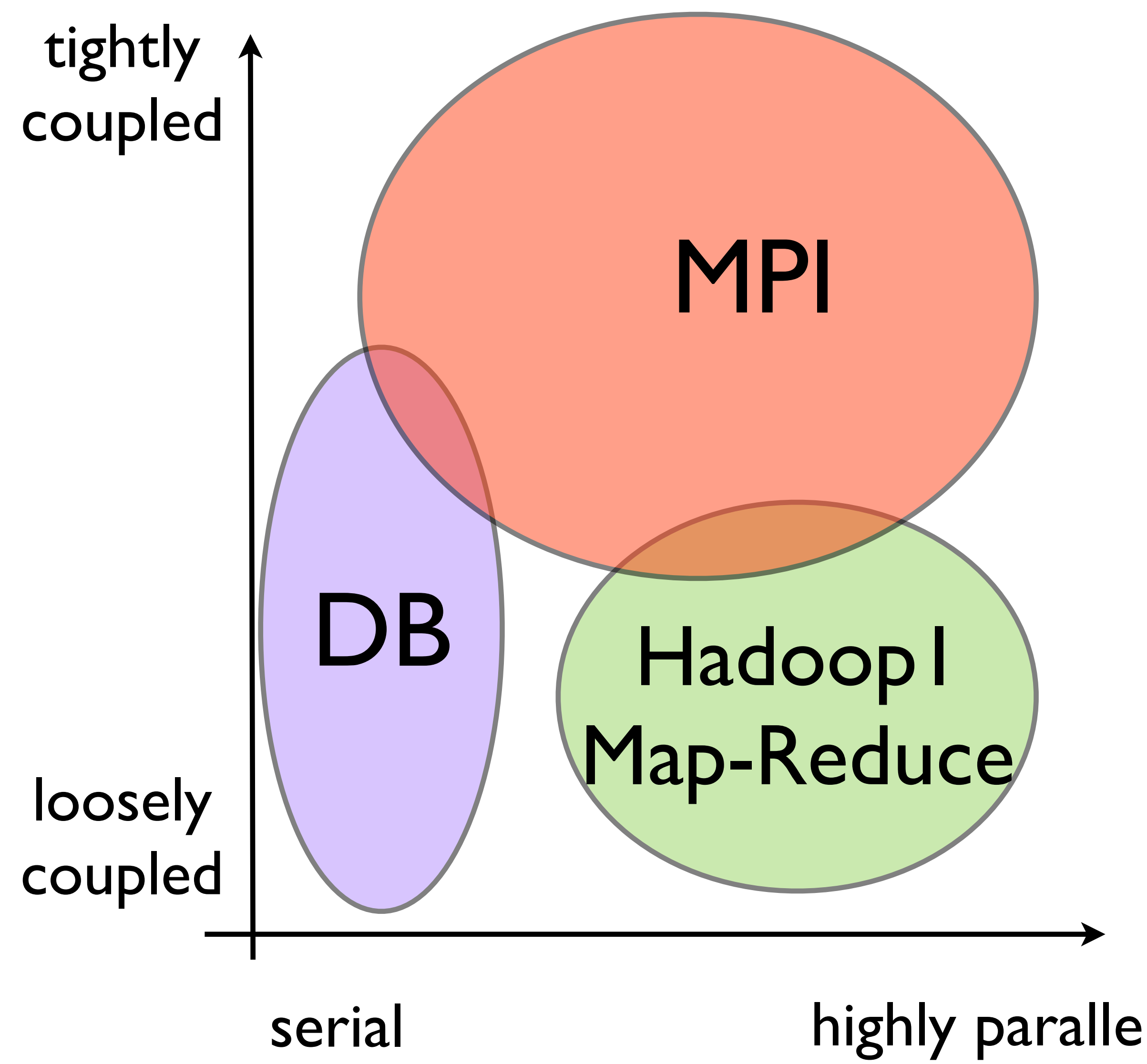
What is it good at?

- 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.

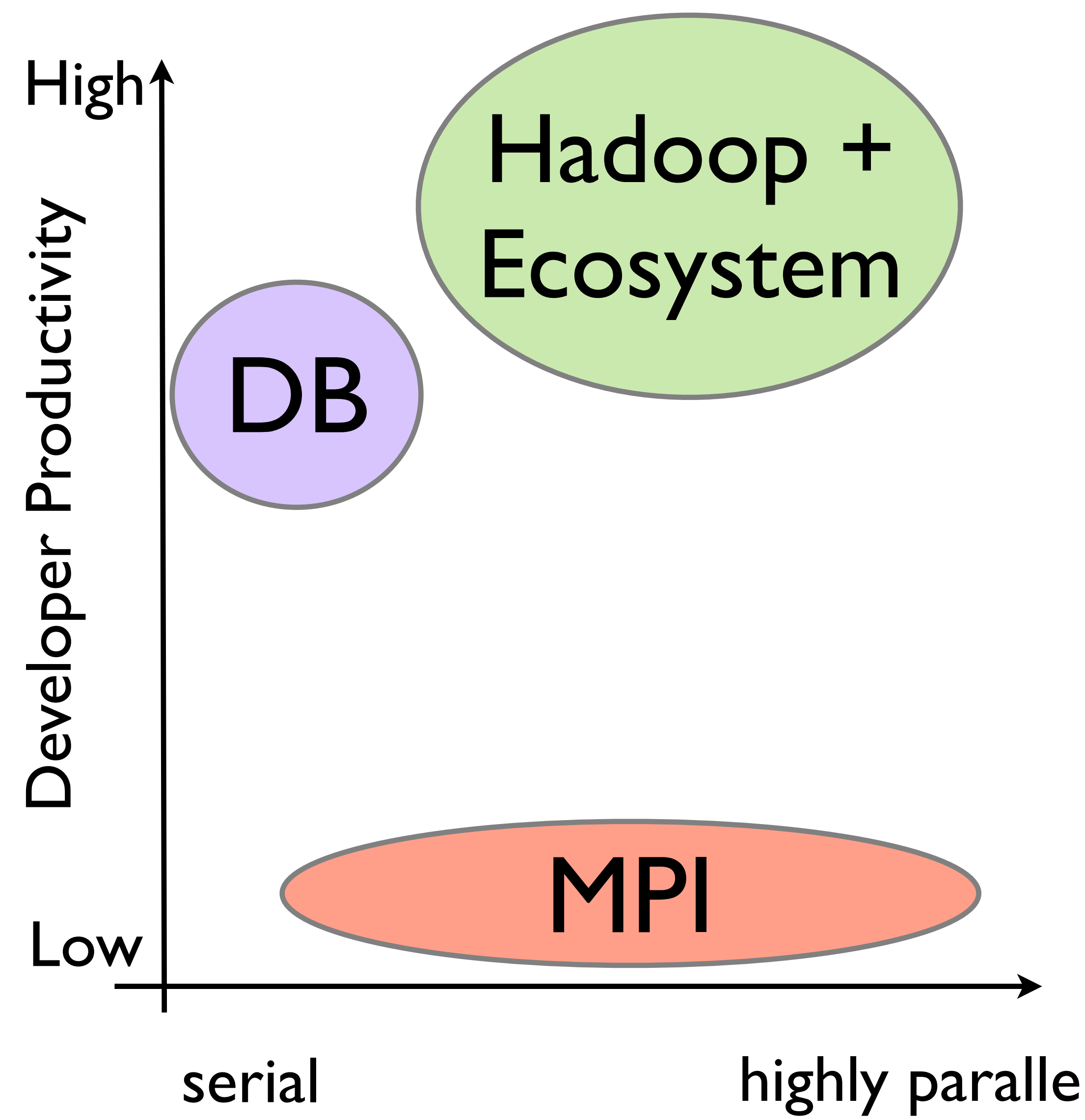


Hadoop vs HPC

- 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."

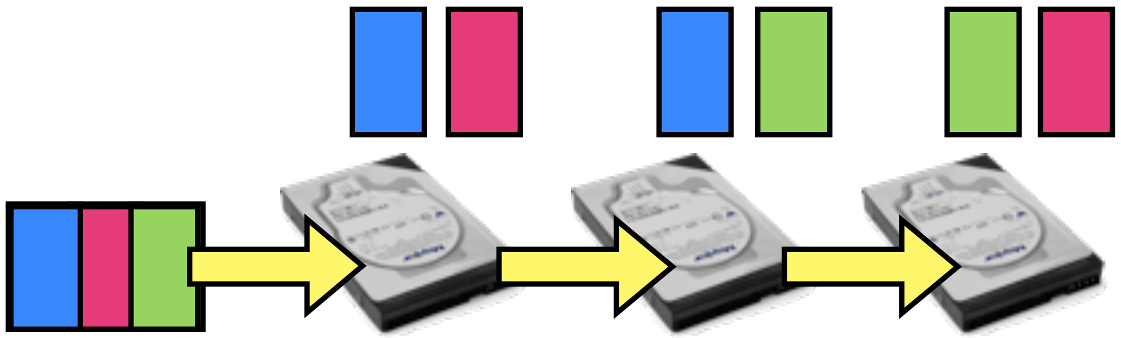
Hadoop vs HPC

- 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.



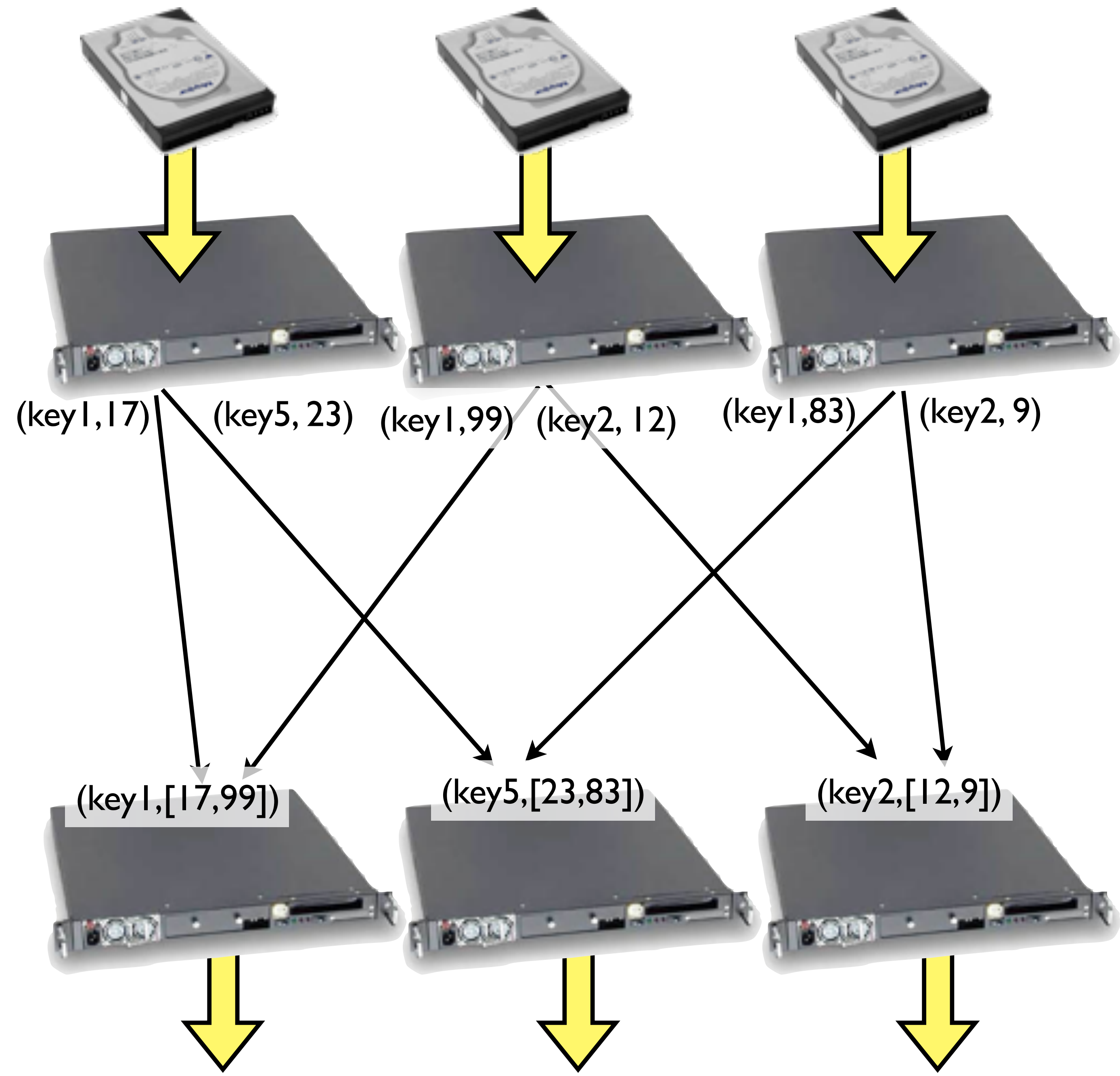
Data Distribution: Disk

- Hadoop and similar architectures handle the hardest 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

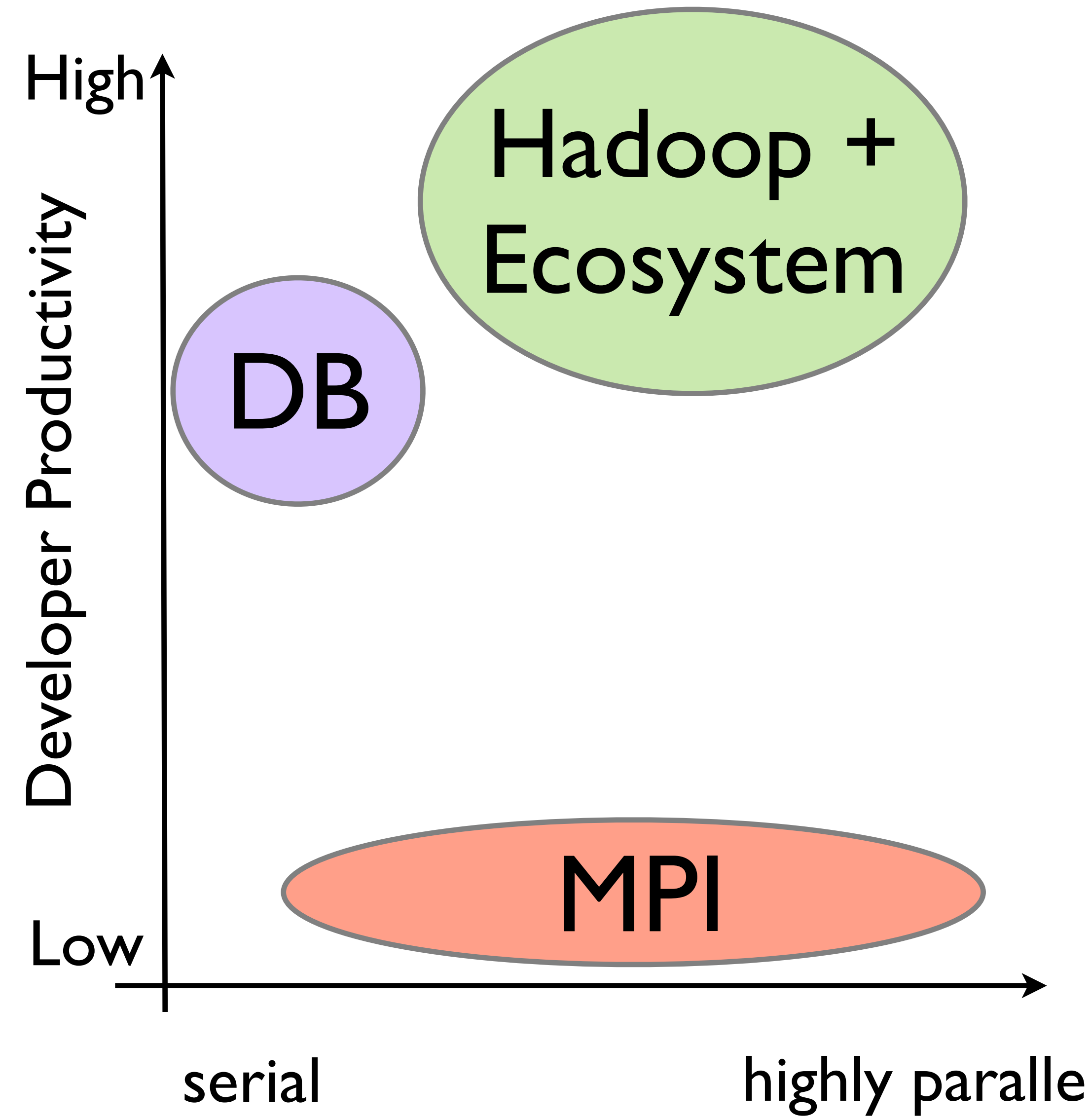
- On network: Map Reduce (eg) works in terms of key-value pairs.
- Preprocessing (map) phase ingests data, emits (k,v) pairs
- Shuffle phase assigns reducers, gets all pairs with same key onto that reducer.
- Programmer does not have to design communication patterns



SciNet compute • calcul CANADA

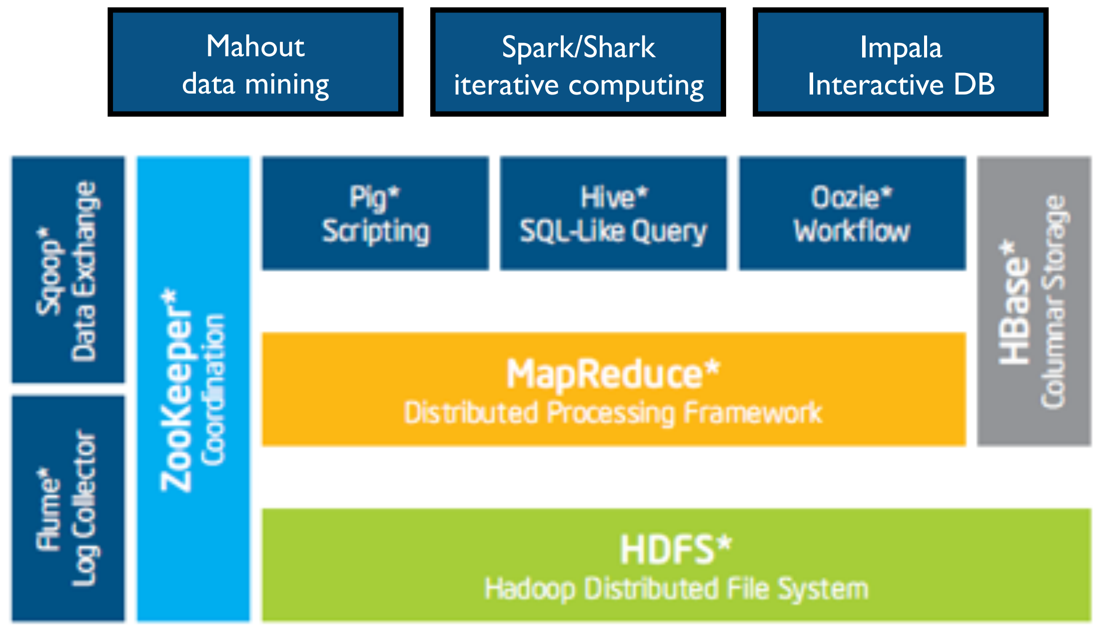
Makes the problem easier

- 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



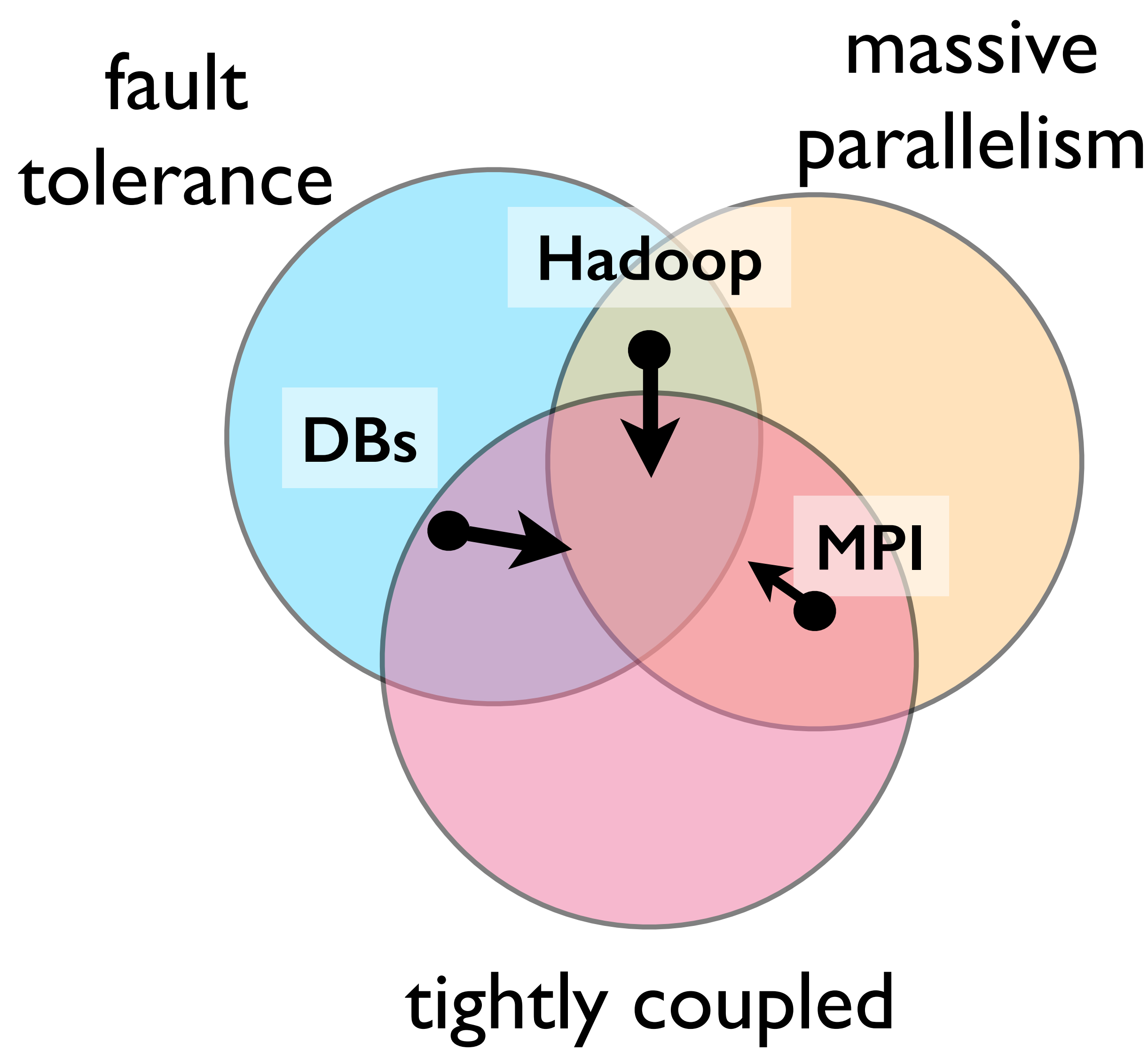
SciNet compute • calcul CANADA

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

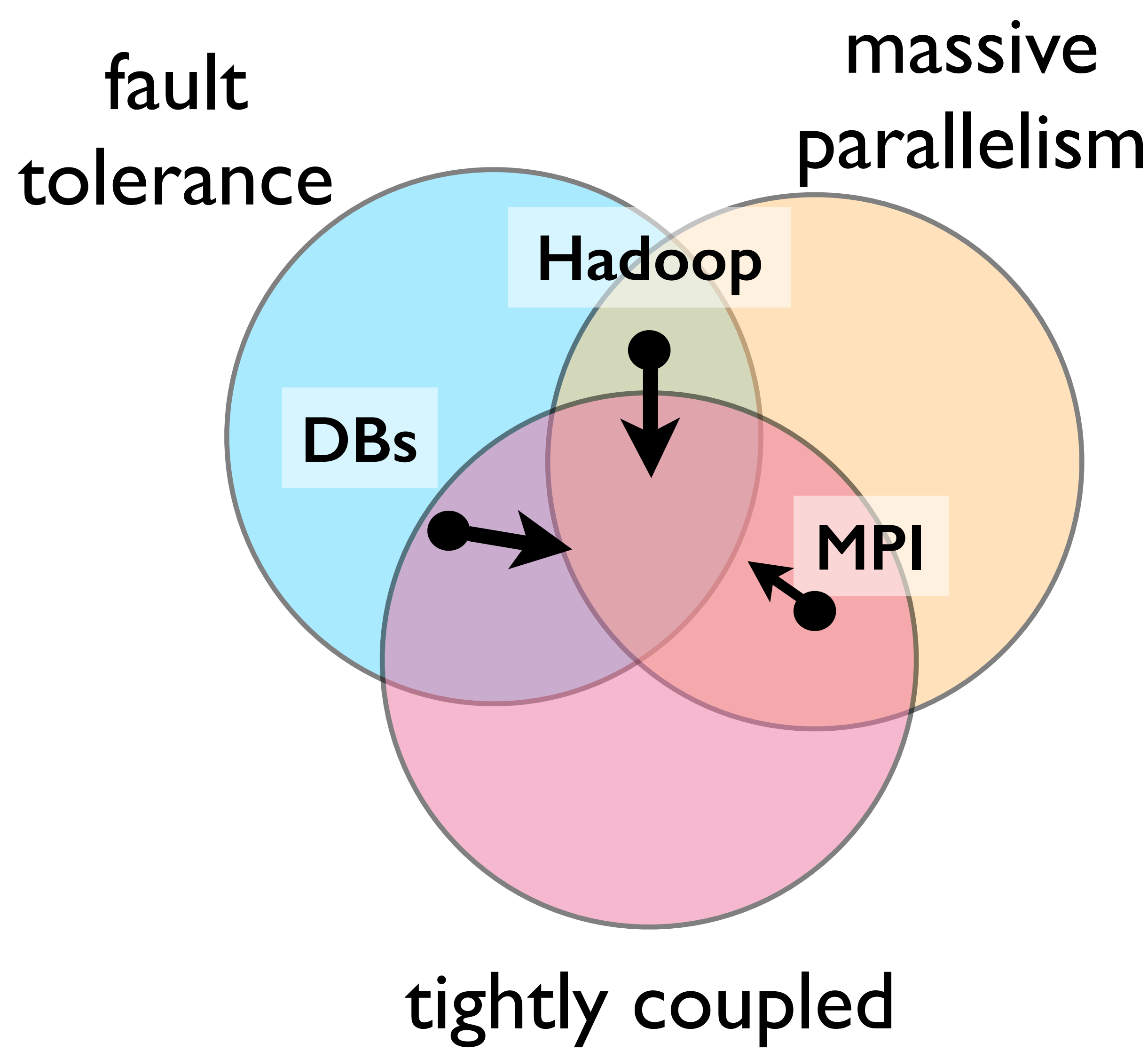
- These models are all converging at the largest scales
- Good ideas are good ideas.
- MPI is trying to grow fault tolerance (but MPI codes?)
- Relational DBs are scaling up



SciNet compute • calcul CANADA

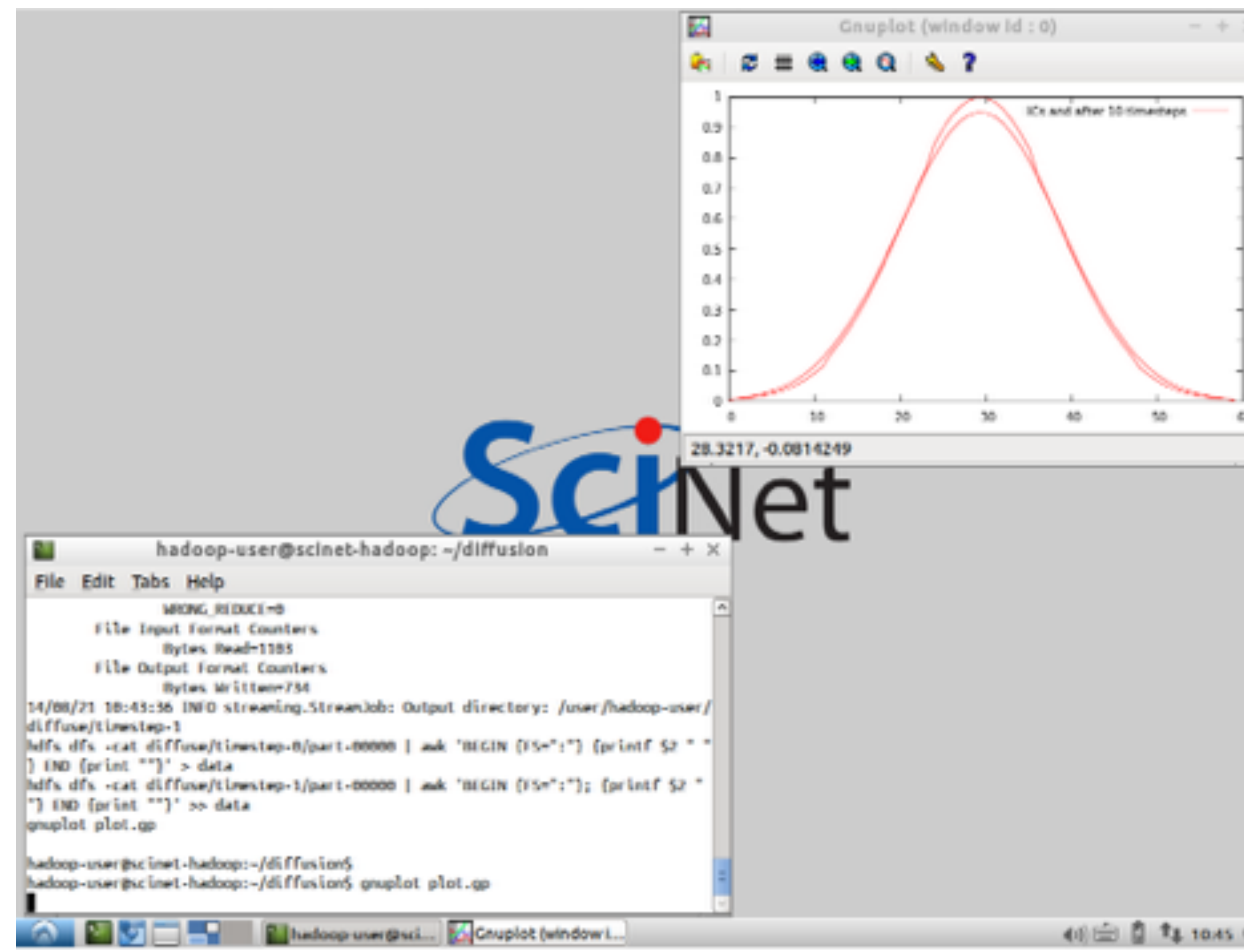
1: Everything's Converging

- People are building tools for tightly coupled computation atop Hadoop-like frameworks
- Hadoop is growing tightly coupled *much* faster than MPI codes are growing 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".
- Will look at some numerical computation examples:
    - Distributed 1d diffusion PDE
    - non-linear least squares
    - sparse matrix multiplication
- And Hadoop 2.x allows models of computation much beyond MapReduce.



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
```

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

SciNet compute • calcul CANADA

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 onto the Hadoop file system (hdfs dfs -put)
- 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 APIs
- 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	

SciNet compute • calcul CANADA

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* especially low latency
- No caching (what's the point if primarily for streaming reads)
- Poor support for seeking around files
- Poor support for zillions of files
- Have to use separate API to see filesystem

Modelled 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.

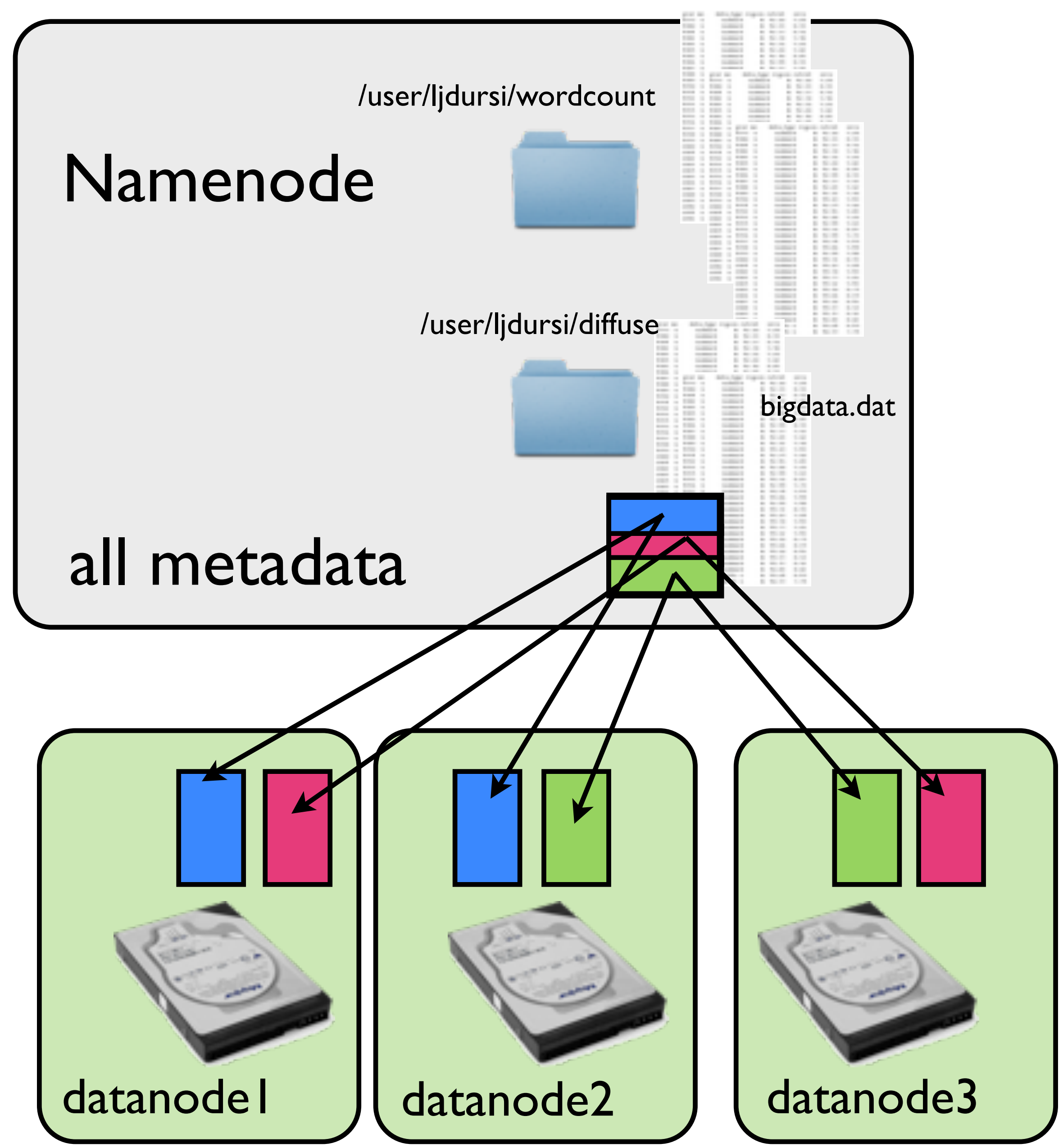
Datanodes and Namenode

Two different types of nodes in the filesystem

Namenode - stores all metadata and block locations in memory.

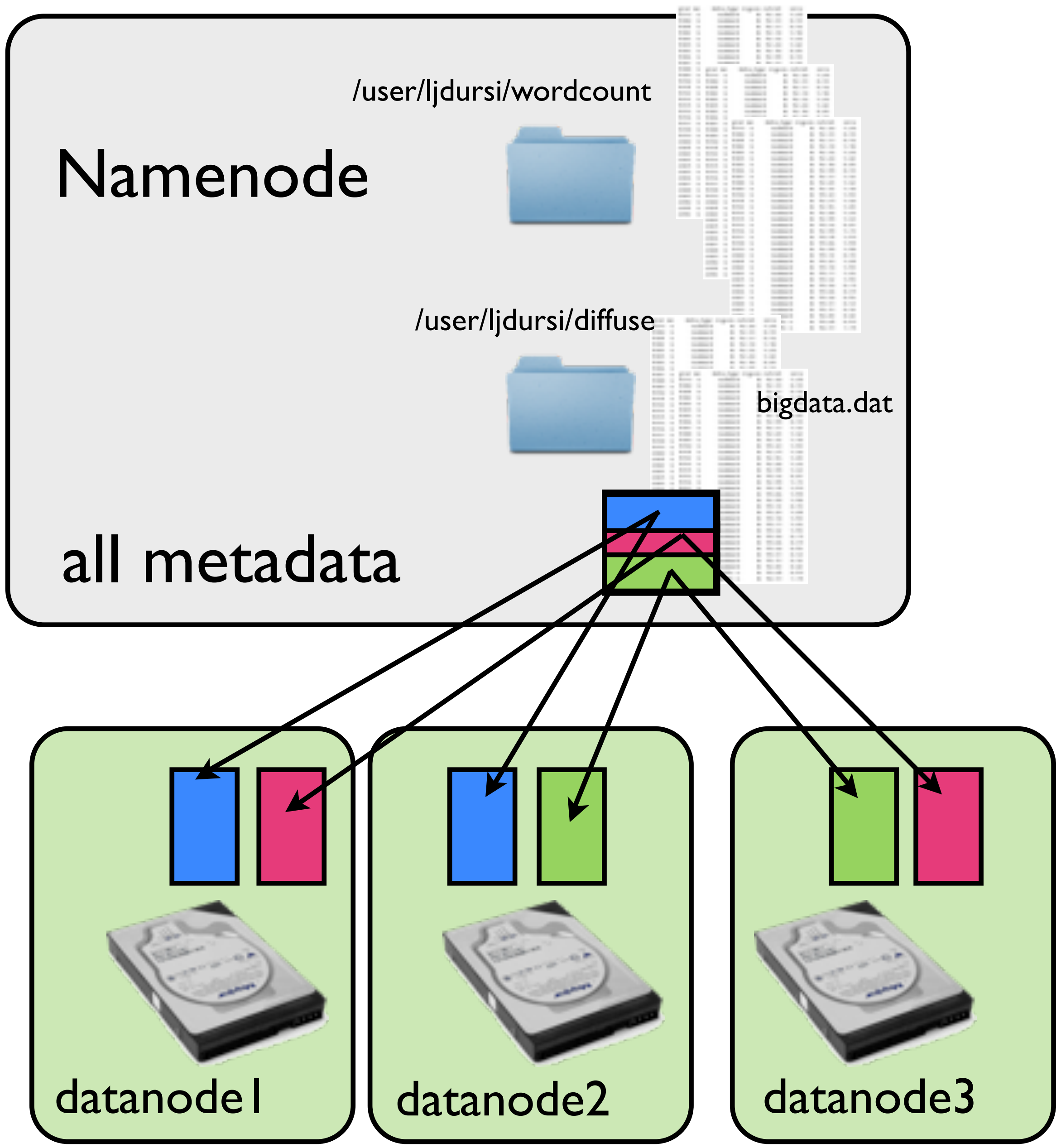
- Metadata updates are stored to persistent journal
- Lots of files bad

Datanodes - store and retrieve blocks for client or namenode



Datanodes and Namenode

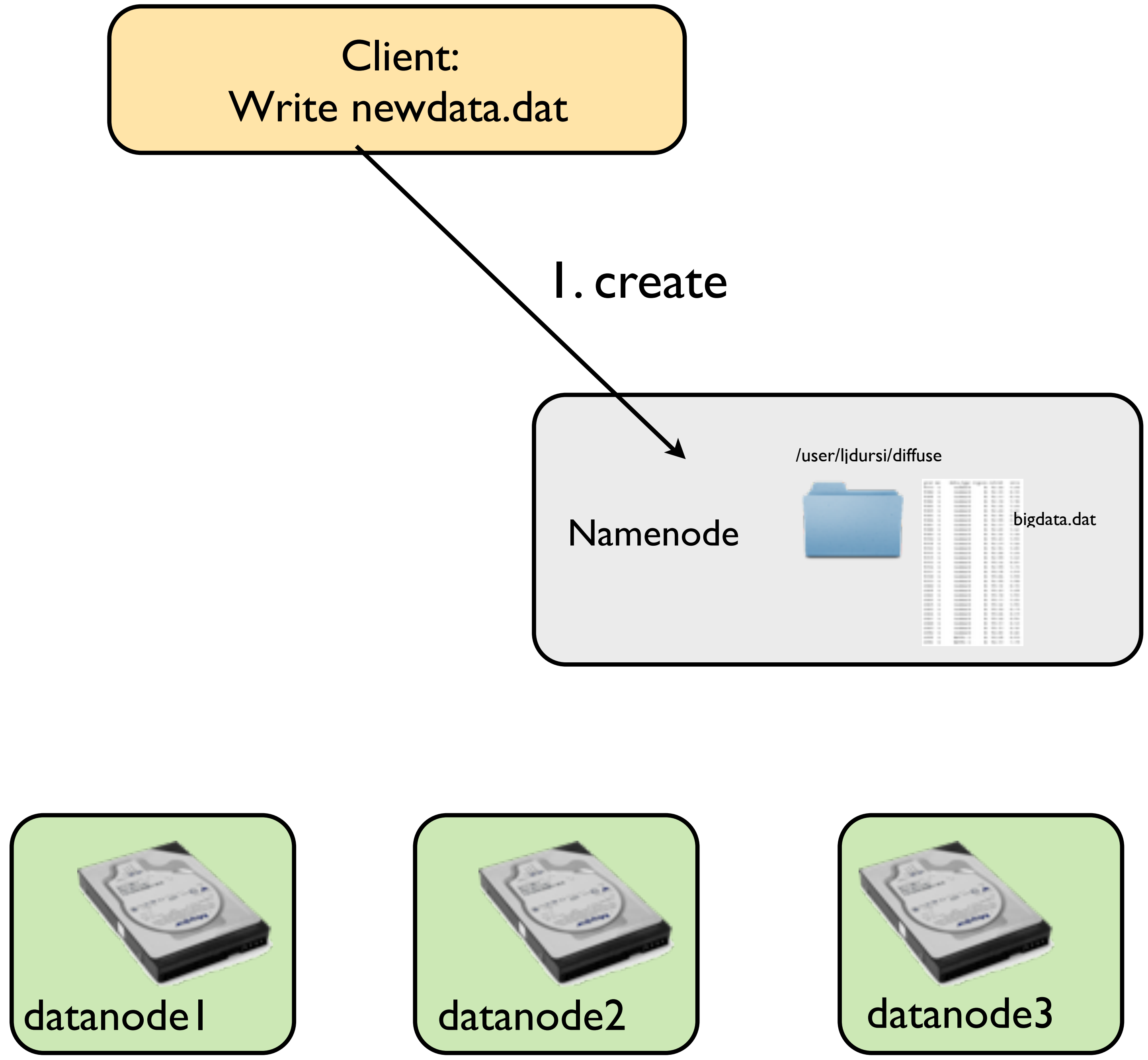
- Newer versions of Hadoop - federation (different namenodes for /user, /data, /project , etc)
- Newer versions of Hadoop - High Availability namenode pairs



Writing a file

Writing a file multiple stage process

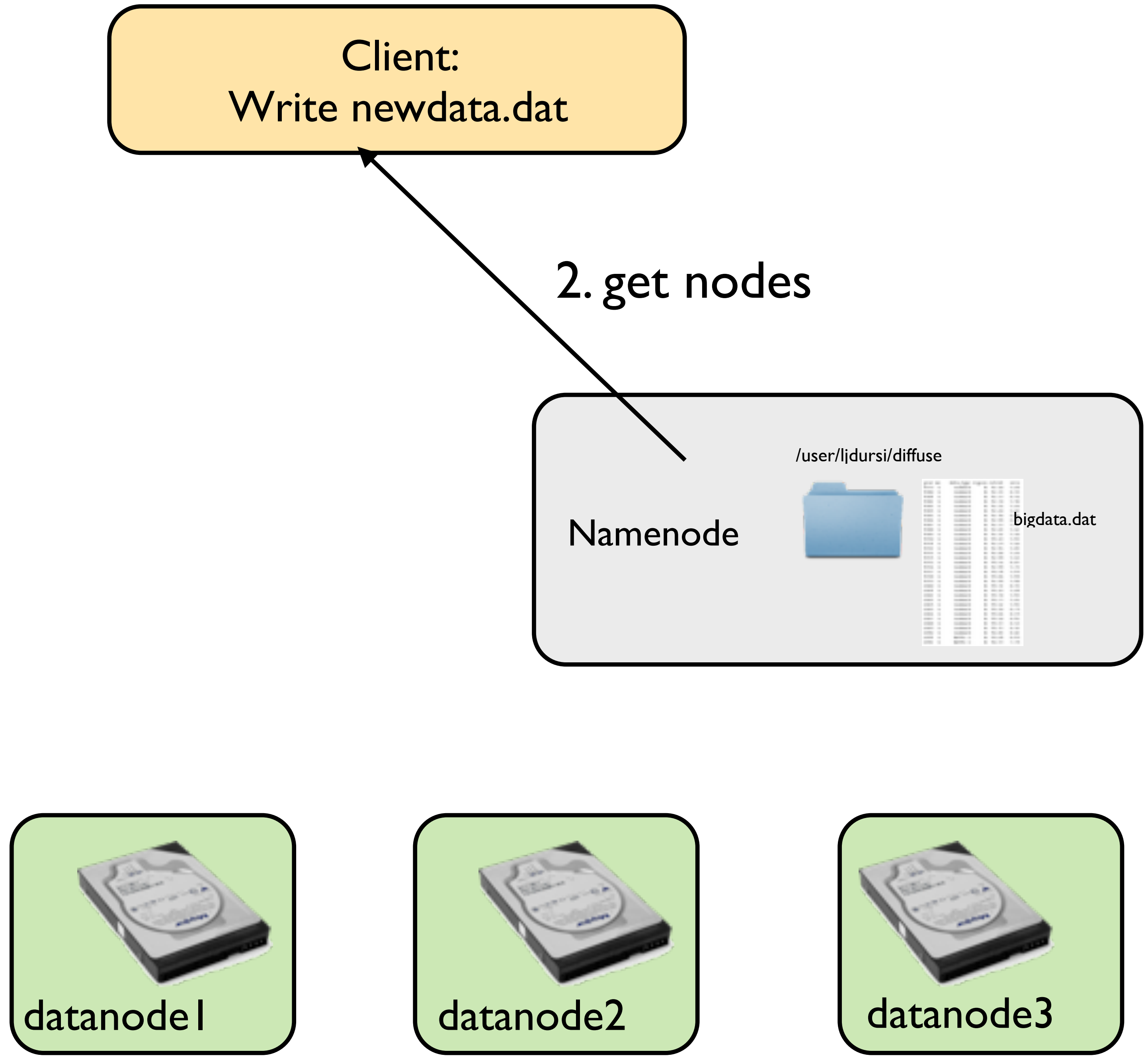
- •Create file
- •Get nodes for blocks
- •Start writing
- •Data nodes coordinate replication
- •Get ack back
- •Complete



Writing a file

Writing a file multiple stage process

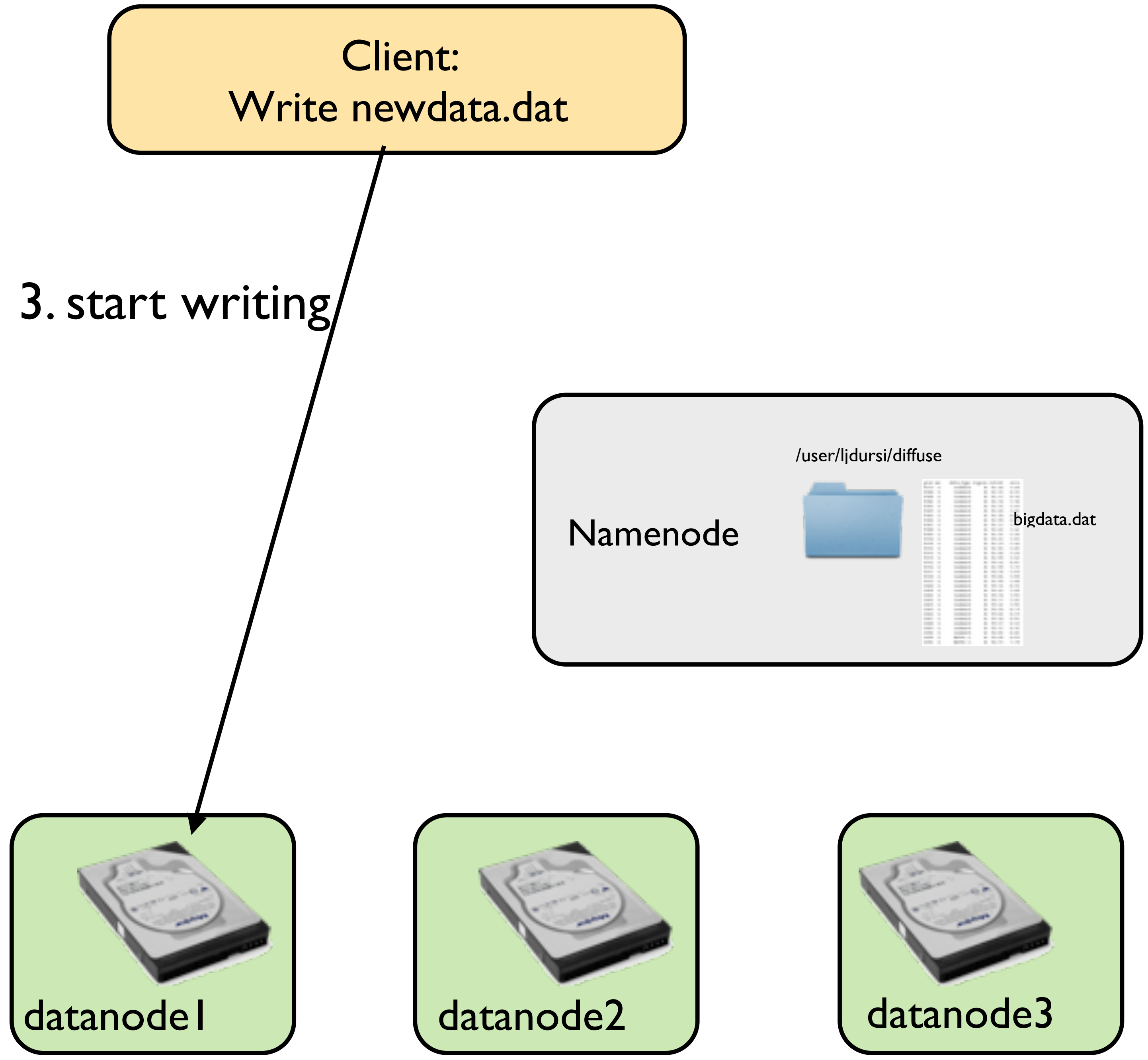
- •Create file
- •Get nodes for blocks
- •Start writing
- •Data nodes coordinate replication
- •Get ack back
- •Complete



Writing a file

Writing a file multiple stage process

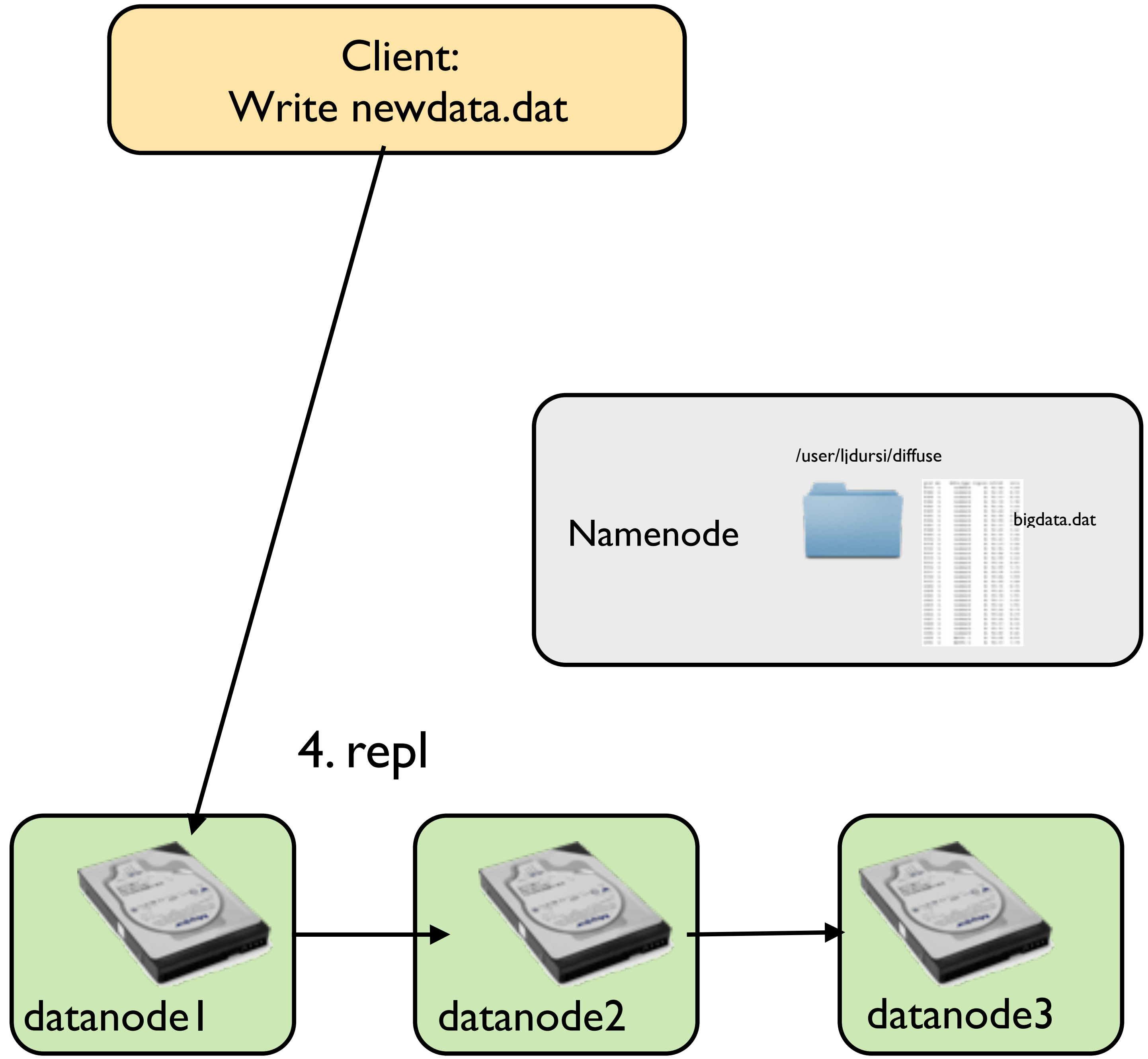
- Create file
- Get nodes for blocks
- Start writing
- Data nodes coordinate replication
- Get ack back
- Complete



Writing a file

Writing a file multiple stage process

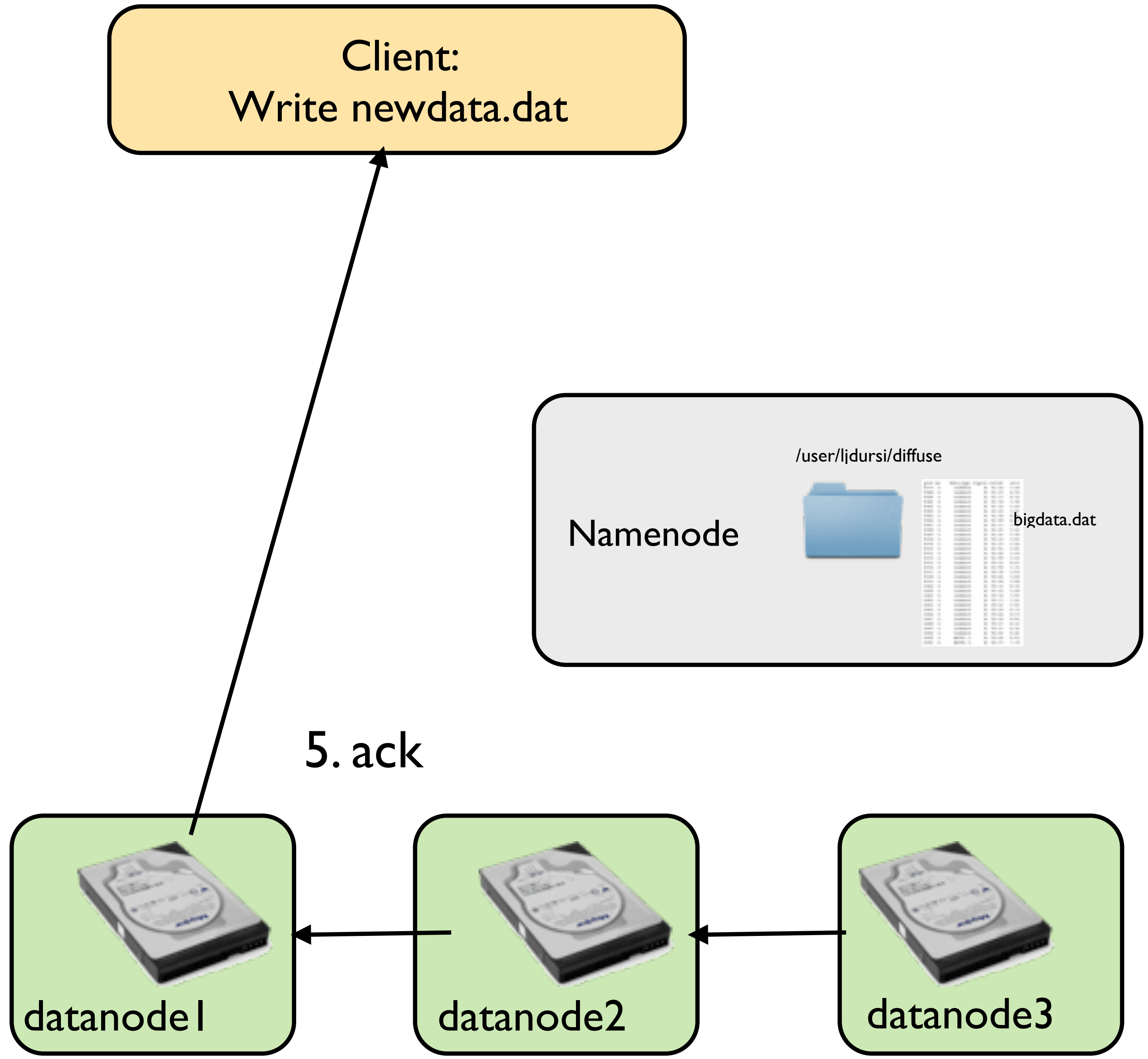
- Create file
- Get nodes for blocks
- Start writing
- Data nodes coordinate replication
- Get ack back
- Complete



Writing a file

Writing a file multiple stage process

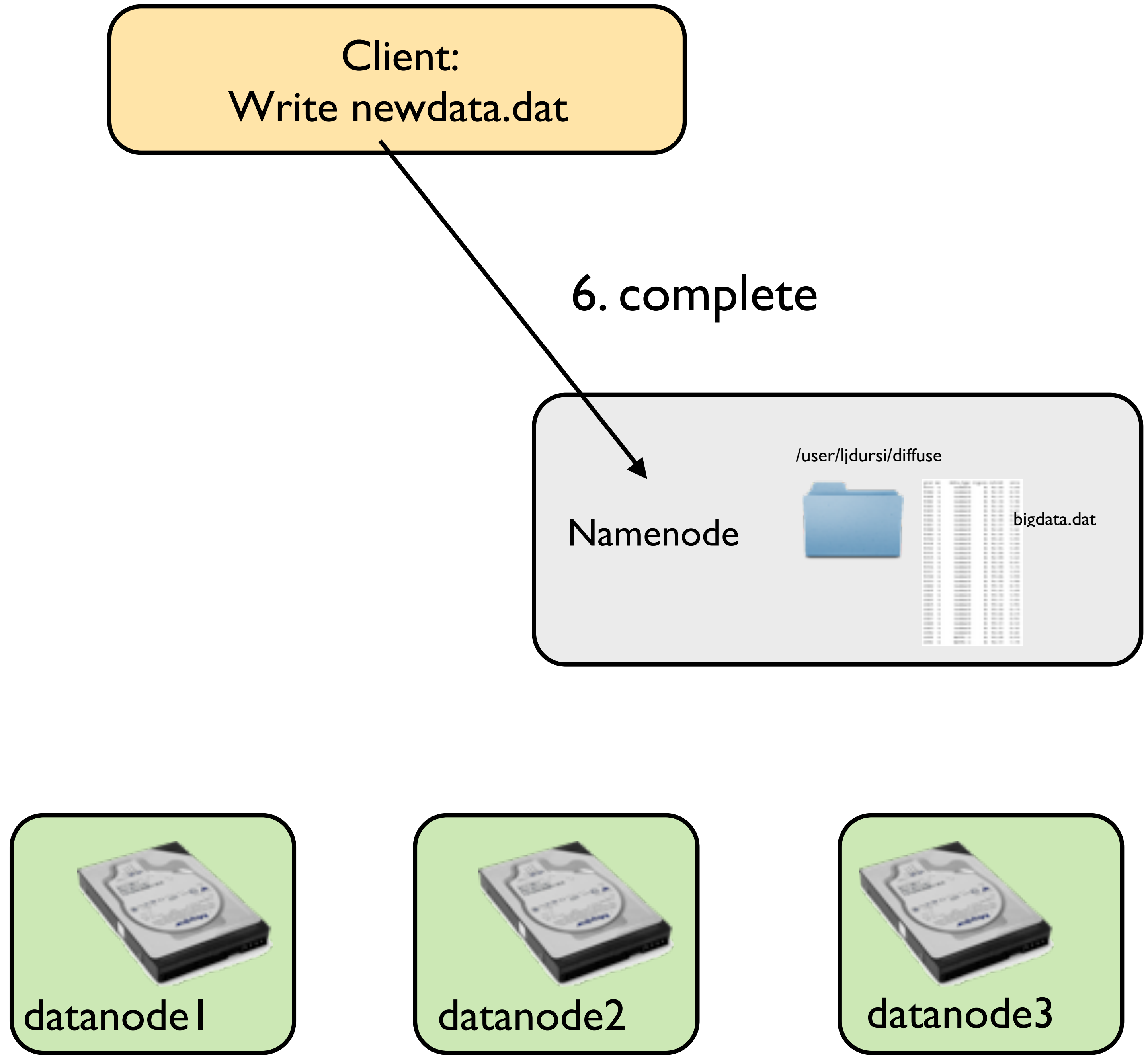
- Create file
- Get nodes for blocks
- Start writing
- Data nodes coordinate replication
- Get ack back (while writing)
- Complete



Writing a file

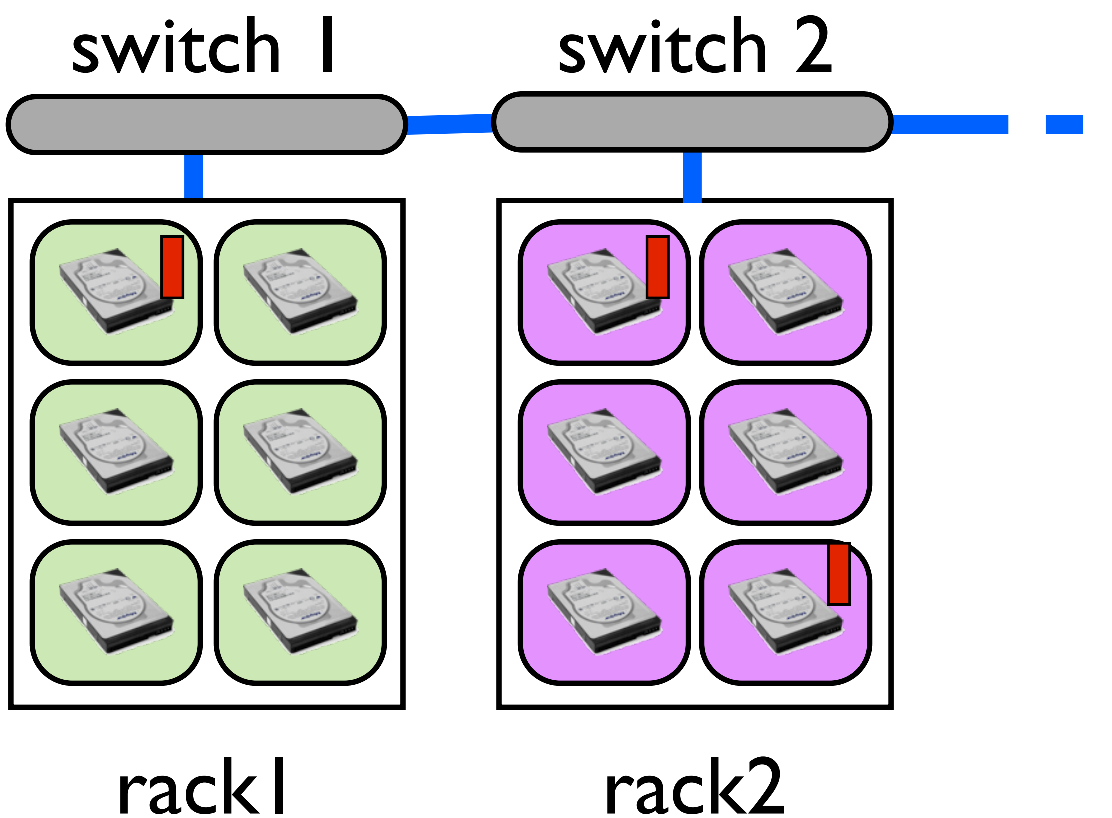
Writing a file multiple stage process

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



Where to Replicate?

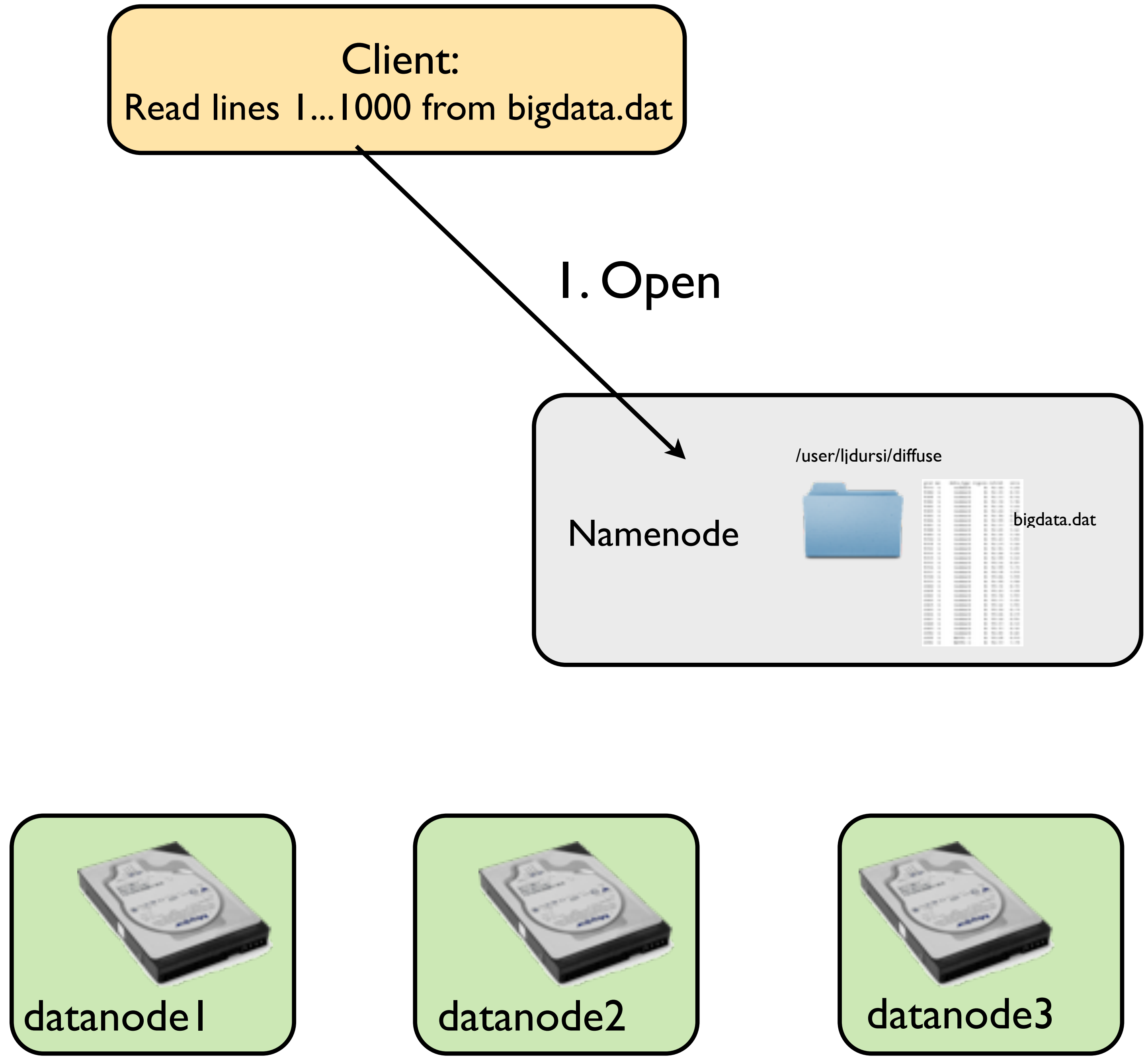
- Tradeoff to choosing replication locations
- Close: faster updates, less network bandwidth
- Further: better failure tolerance
- Default strategy: first copy on different location on same node, second on different "rack"(switch), third on same rack location, different node.
- Strategy configurable.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



Reading a file

Reading a file shorter

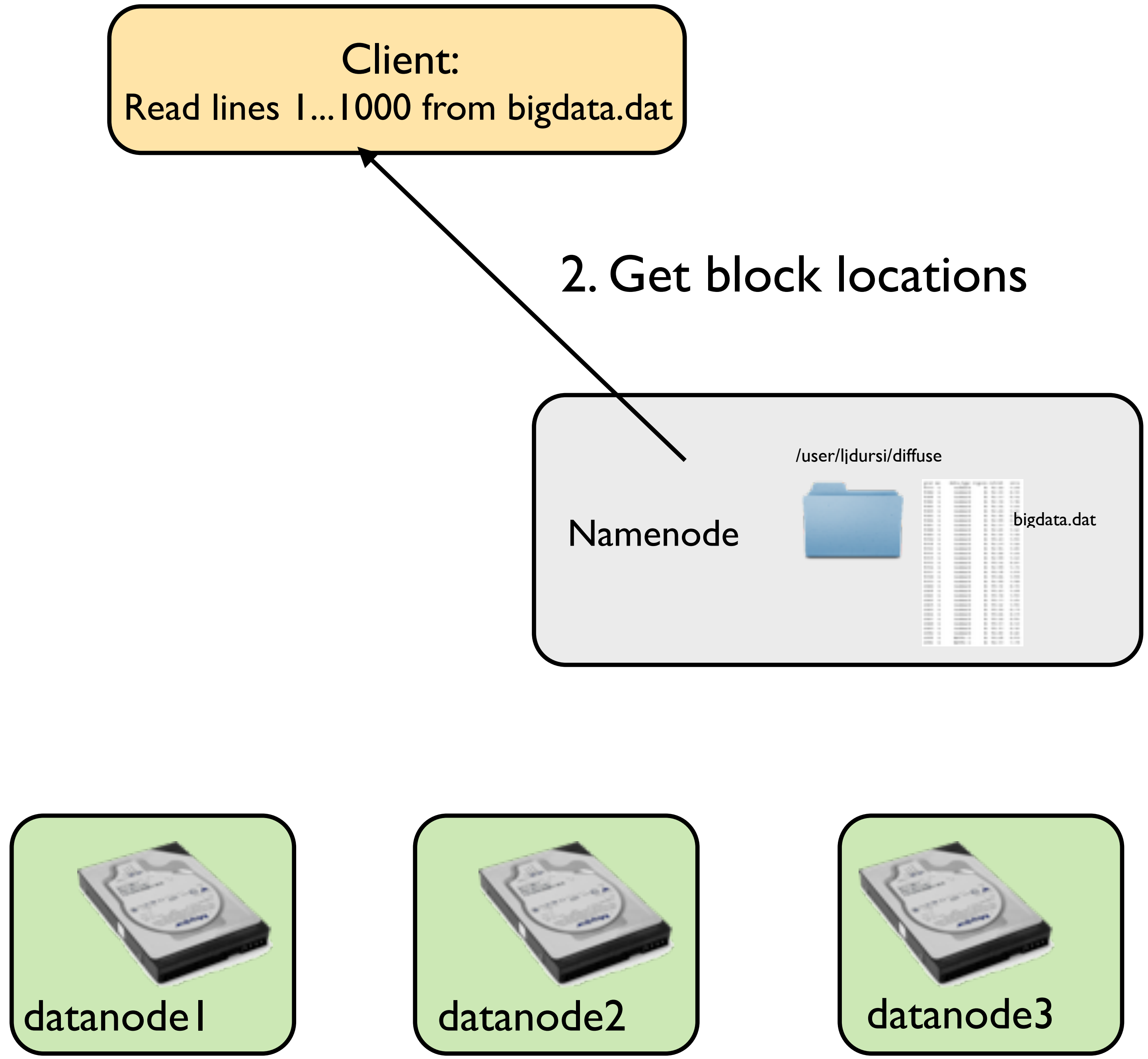
- Get block locations
- Read from a replica



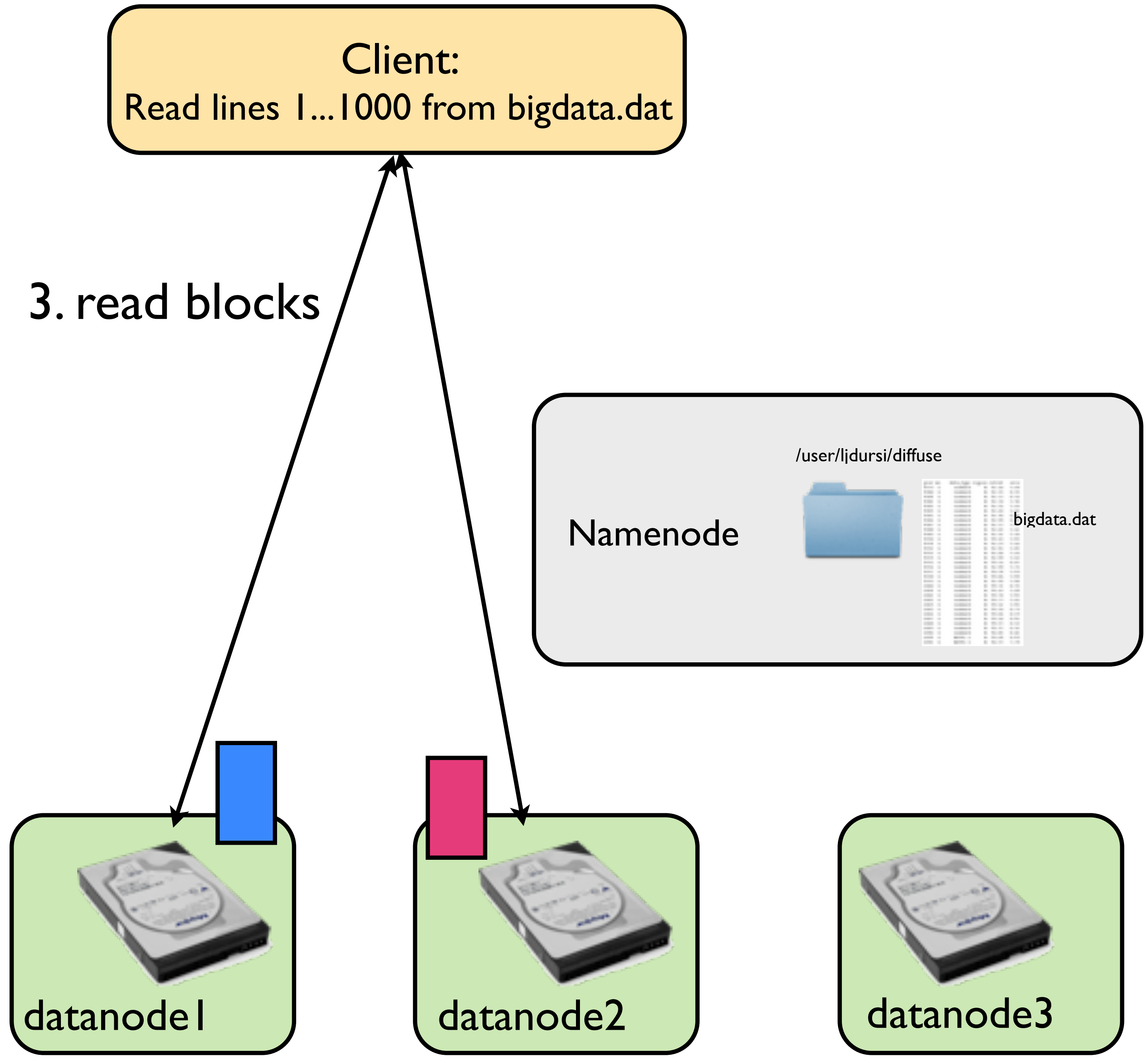
Reading a file

Reading a file shorter

- Get block locations
- Read from a replica



Reading a file



Reading a file shorter

- Get block locations
- Read from a replica

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 FSs

```
<configuration>
  <property>
    <name>fs.defaultFS</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>
```

SciNet compute • calcul CANADA

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.defaultFS</name>
    <value>hdfs://localhost:9000</value>
  </property>

</configuration>
```

SciNet compute • calcul CANADA

Configuring HDFS

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

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

```
<configuration>
. . .

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

</configuration>
```

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=2.3.0
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 directory is `/user/${username}`
- Nothing like a "cd"

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

Using HDFS Hands On

- In ~/hdfs-test, follow directions in README: From shell,
  - create a directory
  - copy (put) a file
  - list a directory
- Can also browse file system via web interface: http://vm-ip:50070/explorer.html

  eg

  http://192.168.33.10:50070/explorer.html

```
hdfs dfs -mkdir /user/hadoop-user/hdfs-test
hdfs dfs -ls /user/hadoop-user
hdfs dfs -ls /user/hadoop-user/hdfs-test
hdfs dfs -put data.dat /user/hadoop-user/hdfs-test
hdfs dfs -ls /user/hadoop-user/hdfs-test
```

Using HDFS Hands On

- In general, the data files you send to HDFS will be *large* (or else why bother with Hadoop)
- Don't want to be constantly copying back and forth - view, append "in place"
- Several APIs to accessing the HDFS - Java, C++, Python
- Here, we use one to get a file's status, and read some data from it at some given offset

```python
from pywebhdfs.webhdfs import PyWebHdfsClient

hdfs = PyWebHdfsClient(host='localhost',port='
my_file = 'user/hadoop-user/hdfs-test/data.dat

print 'Status of file: ', my_file
status = hdfs.get_file_dir_status(my_file)
print status

print 'Second 500 bytes of file: ',my_file
data = hdfs.read_file(my_file,offset=500,lengt

print data
```

hdfs-test/readfile.py

SciNet compute • calcul CANADA

Part II: MapReduce

Hadoop MR Job Workflow

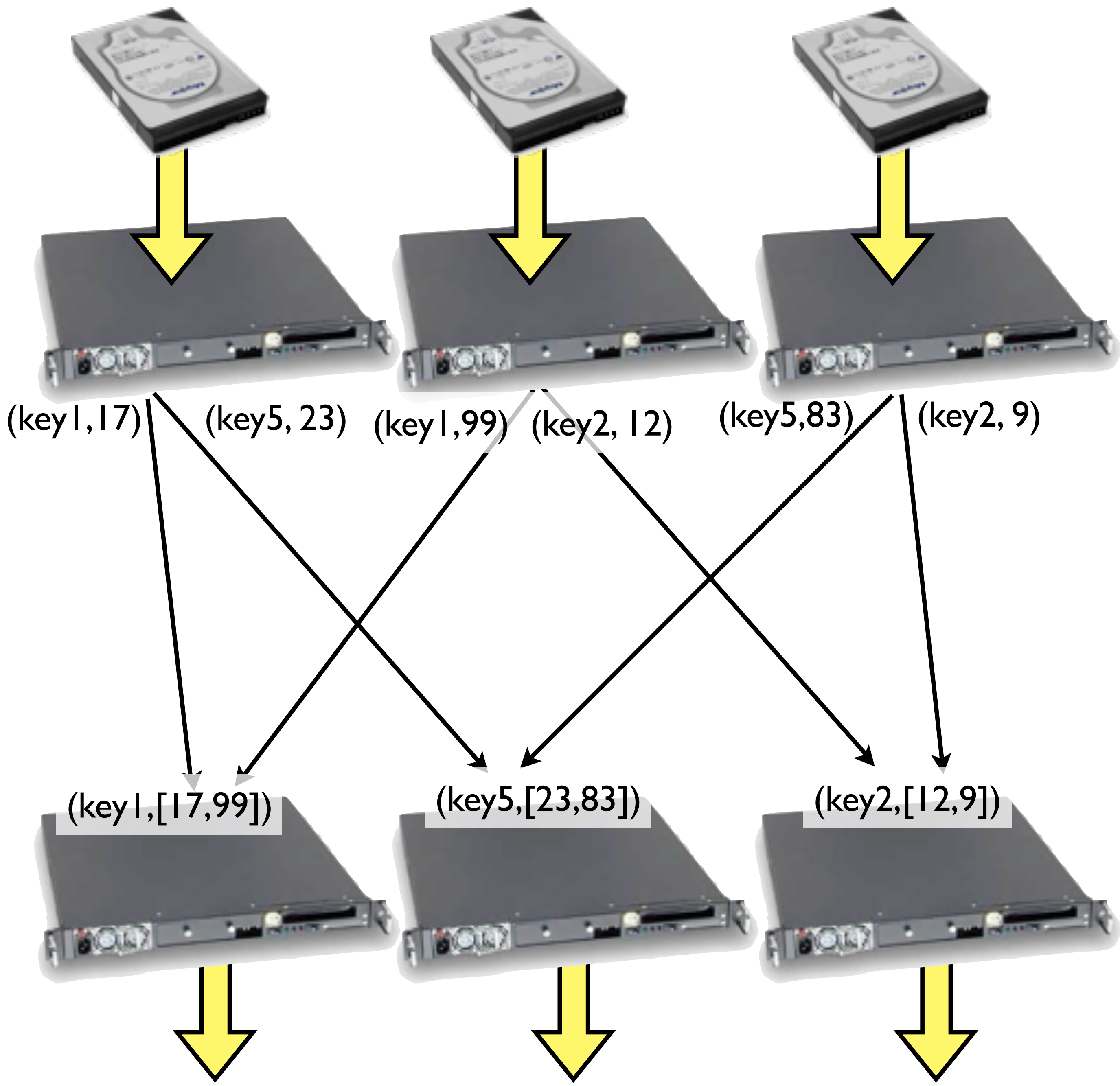
```
wordcount.jar: WordCount.java
	mkdir -p wordcount_classes
	javac -classpath $(CLASSPATH) -Xlint:deprecation \
	      -d wordcount_classes WordCount.java
	jar -cvf wordcount.jar -C wordcount_classes .
```

Building the program

Running a "Map Reduce" program...

MapReduce

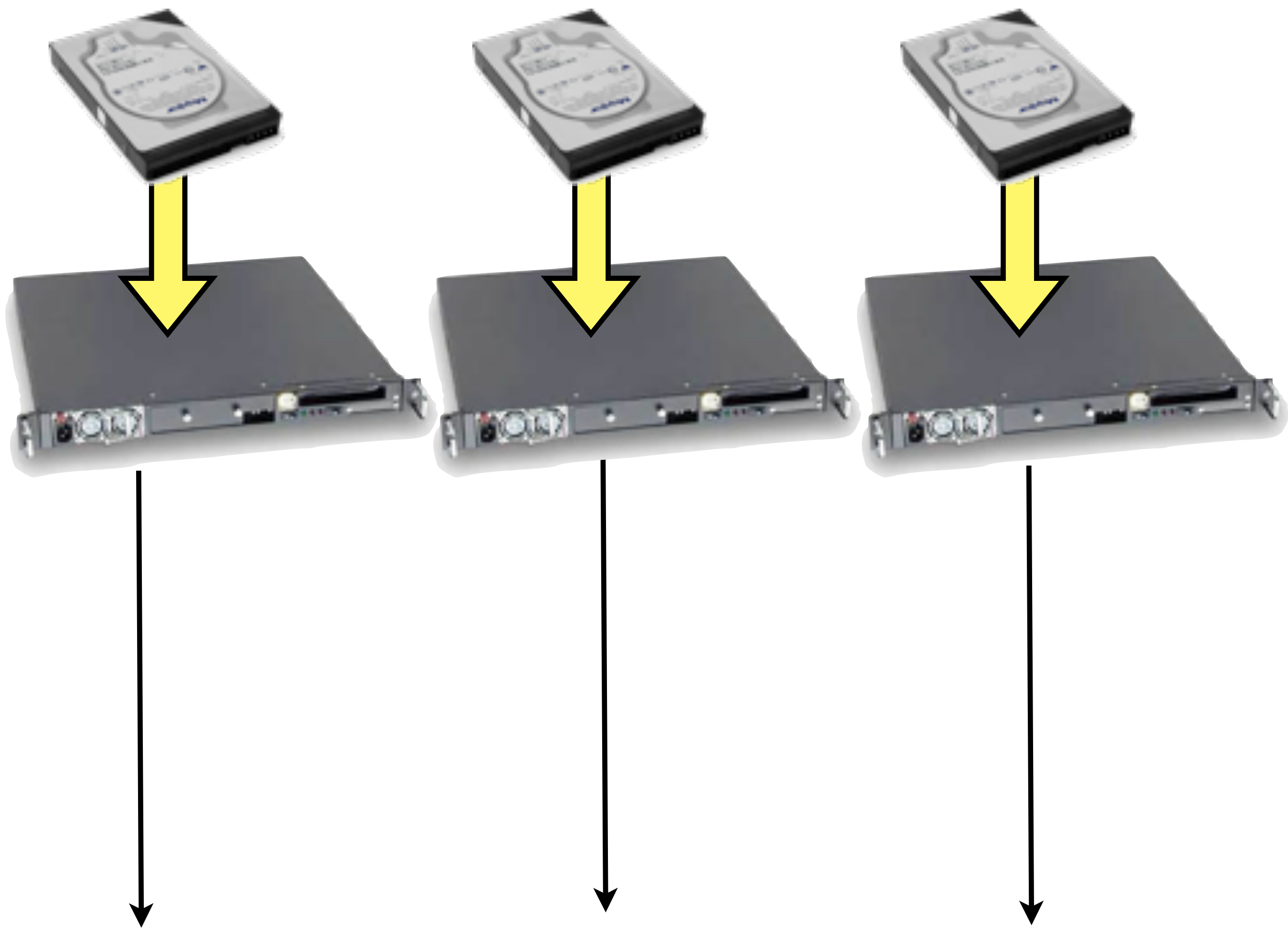
- Two classes of compute tasks: a Map and a Reduce
- Map processes one "element" at a time, emits results as (key, value) pairs.
- All results with same key are gathered to the same reducers
- Reducers process list of values, emit results as (key, value) pairs.



SciNet
compute • calcul
CANADA

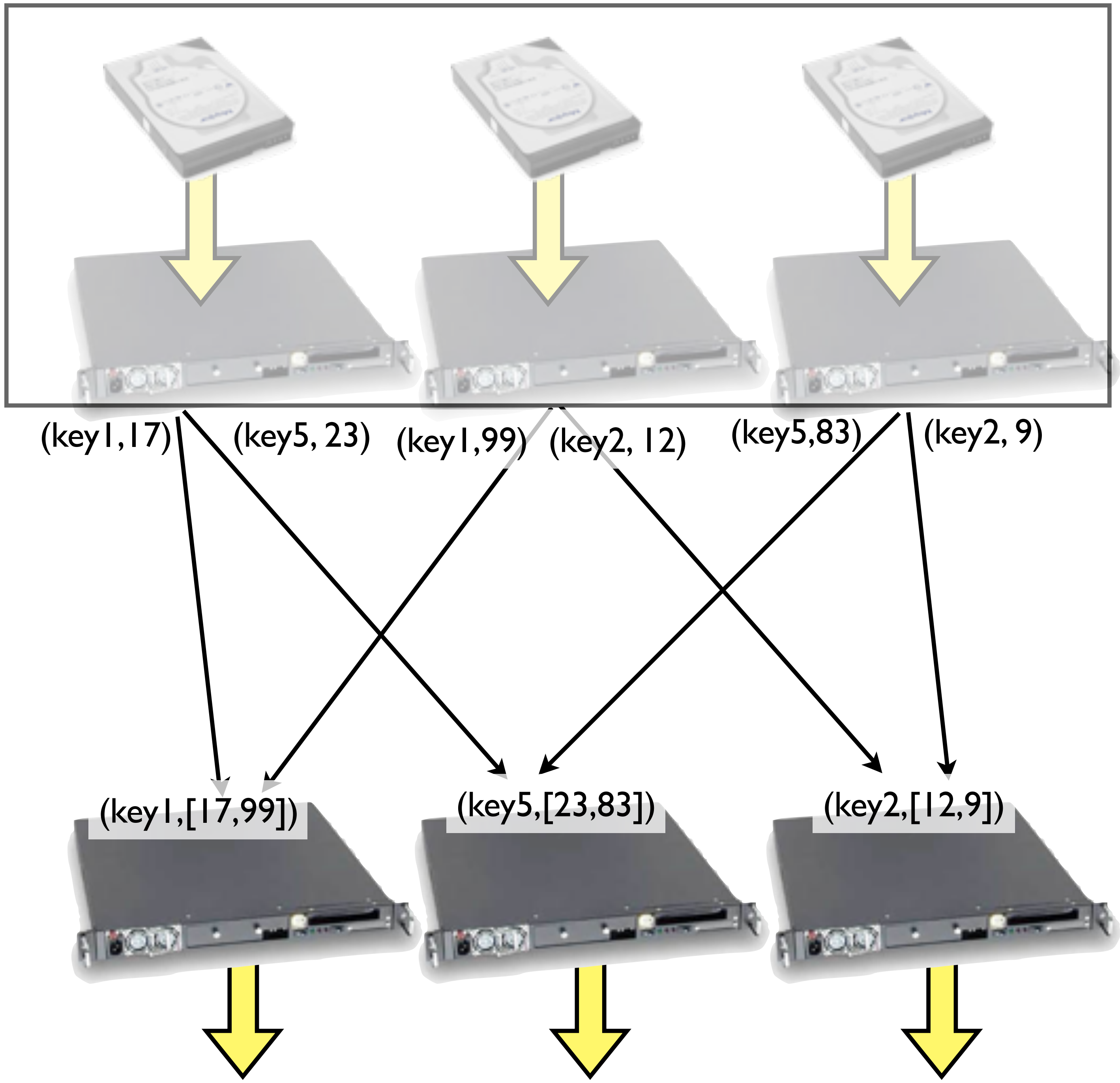
Map

- 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)



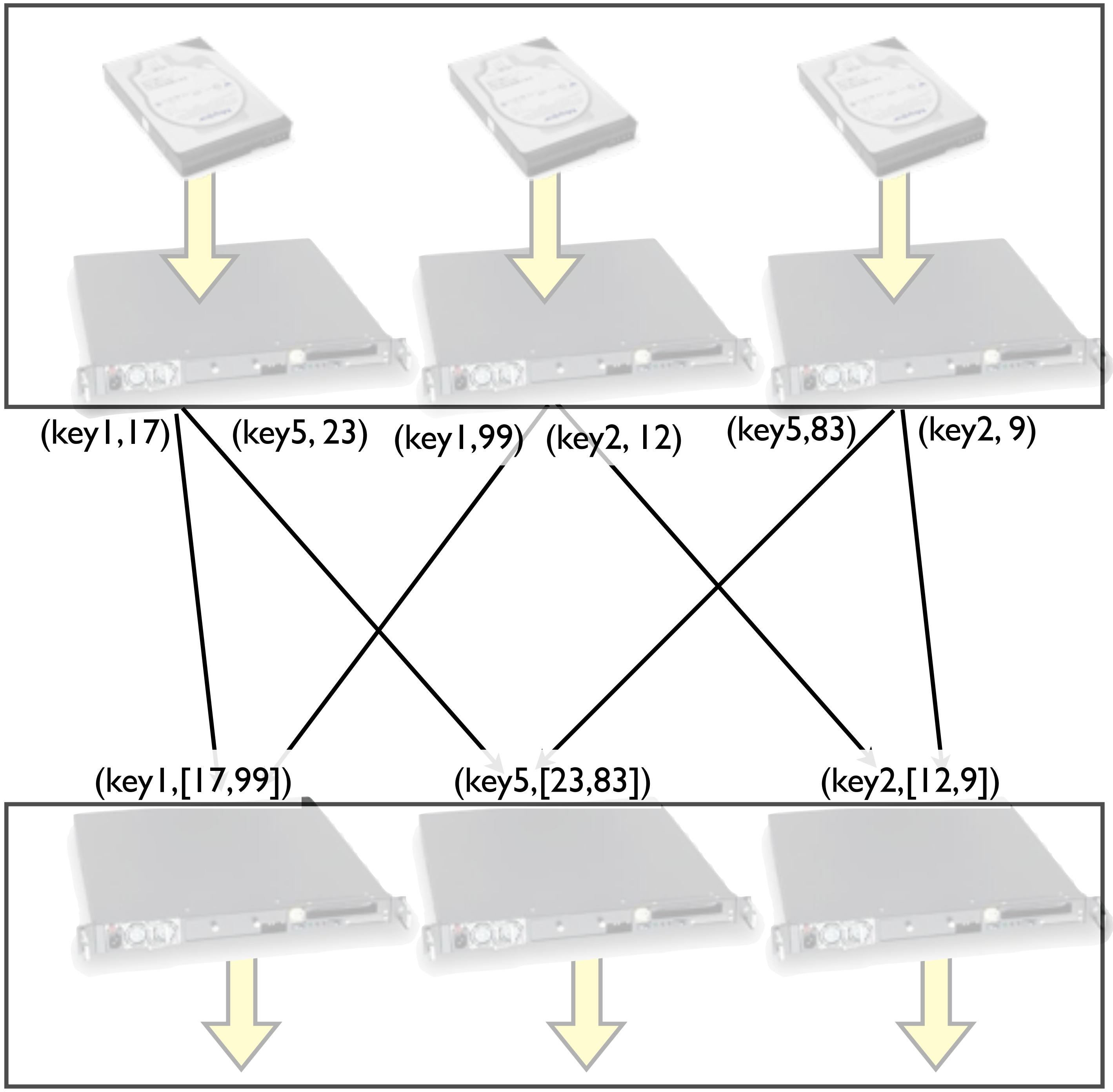
Reduce

- 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

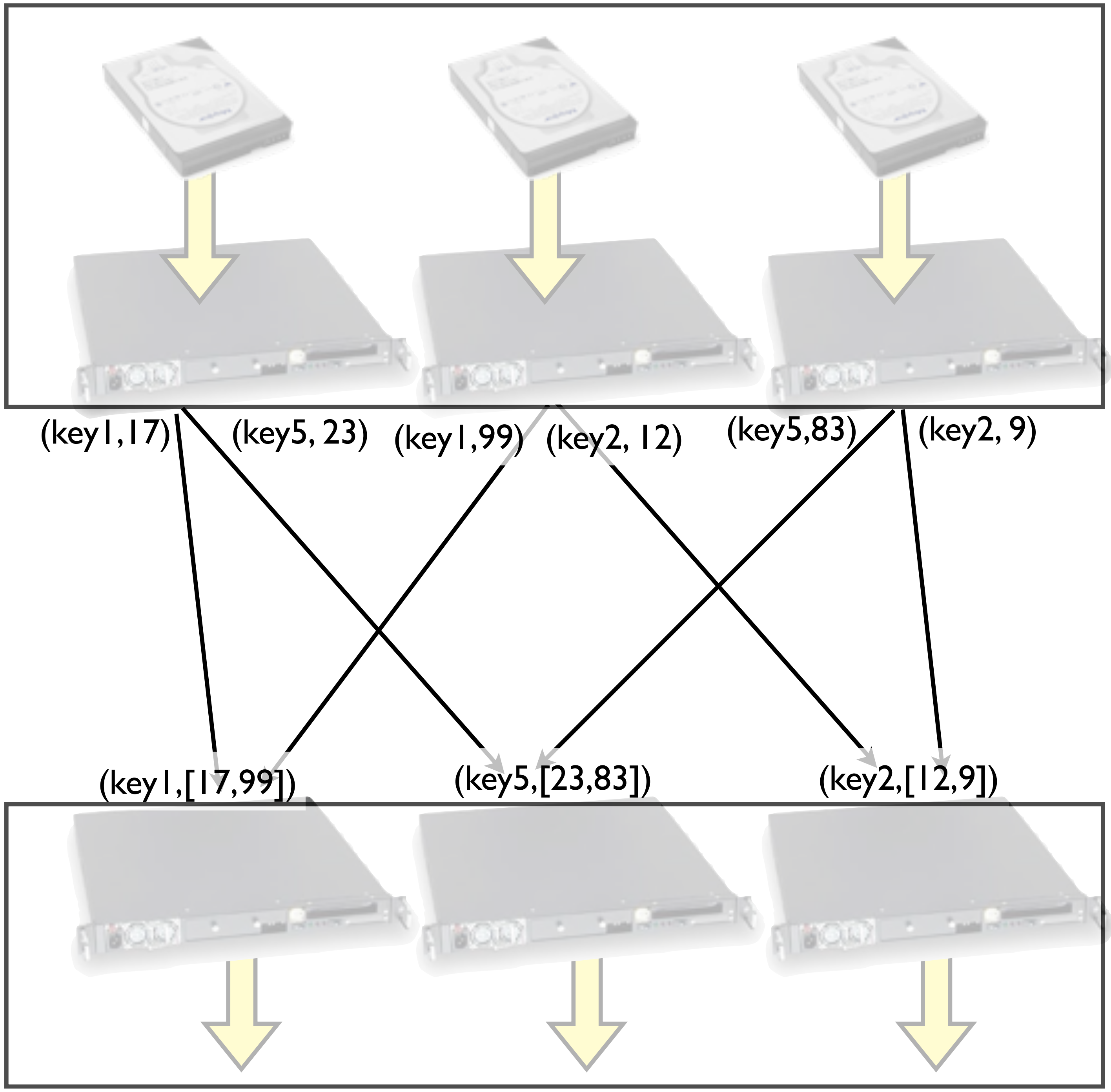
- The shuffle is part of the Hadoop magic
- By default, keys are hashed and hash space is partitioned between reducers
- On reducer, gathered (k,v) pairs from mappers are sorted by key, then merged together by key
- Reducer then runs on one (k,[v]) tuple at a time



SciNet compute • calcul CANADA

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
```

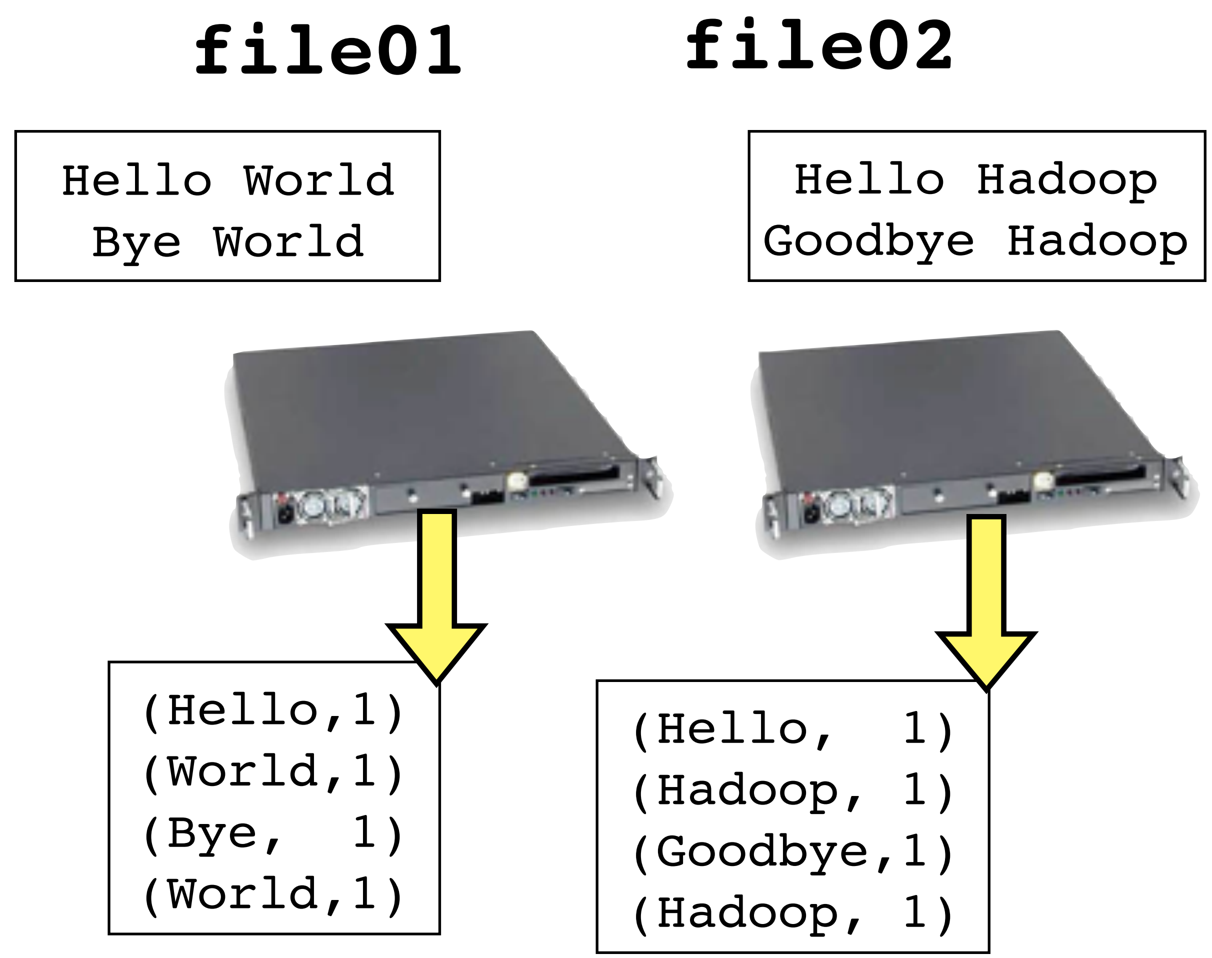
output/part-00000

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

SciNet compute • calcul CANADA

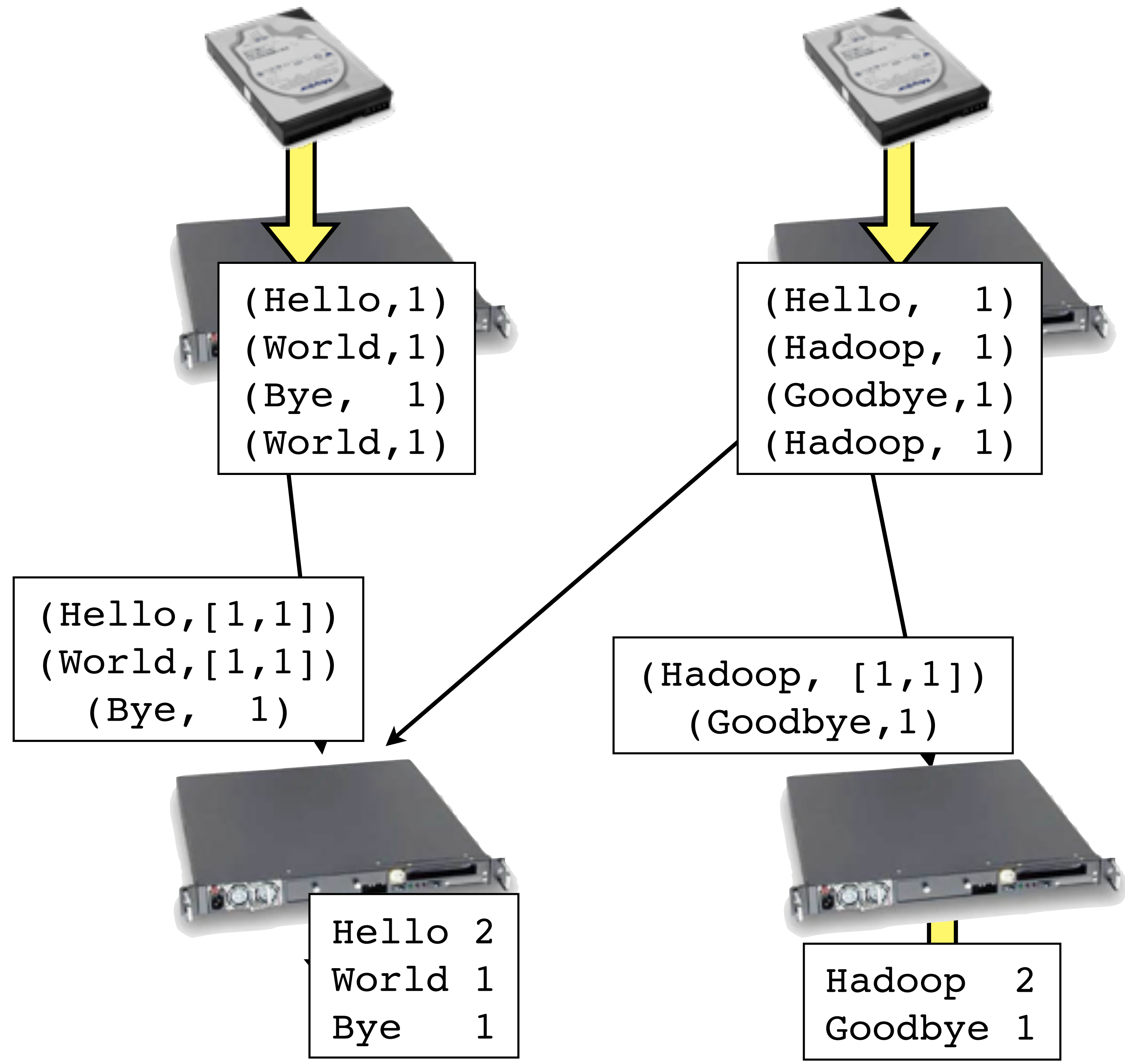
Word Count

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



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.

```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);
  }
}
```

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);
  }
}
```

Python/Hadoop count

- Before looking at 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

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

reduce.py

```
#!/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
Hello     1
World     1
Bye       1
World     1
Hello     1
Hadoop    1
Goodbye   1
Hadoop    1
```

Python/Hadoop count

Can use this approach in serial using standard shell tools:

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

SciNet compute • calcul CANADA

Python/Hadoop count

Can use this approach in serial using standard shell tools:

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

Python/Hadoop count

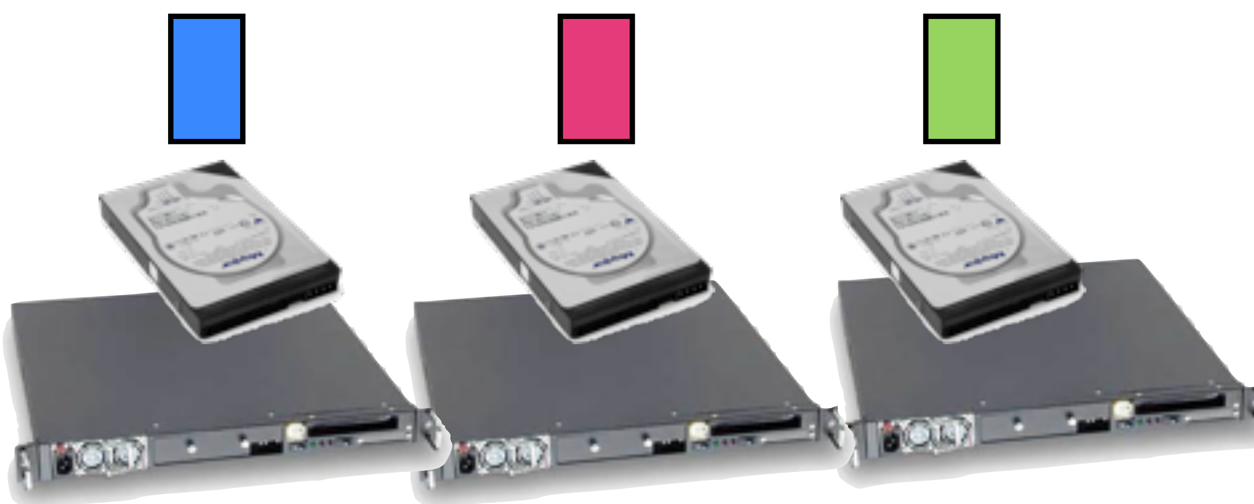
- Can also fire this off in parallel with Hadoop
- "streaming interface", designed to work with other languages
- Hadoop decides how many maps, reduces to fire off

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

- Other interfaces for more programatic interfaces (Pipes - C++; mrjob/pydoop/… - better Python interface, etc)
- Streaming seems to work roughly as well or better if can live with text data

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 replica is mapped usually (but fault tolerance)



SciNet compute • calcul CANADA

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.

```
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);
    }
  }
}
```

Mapper and I/O

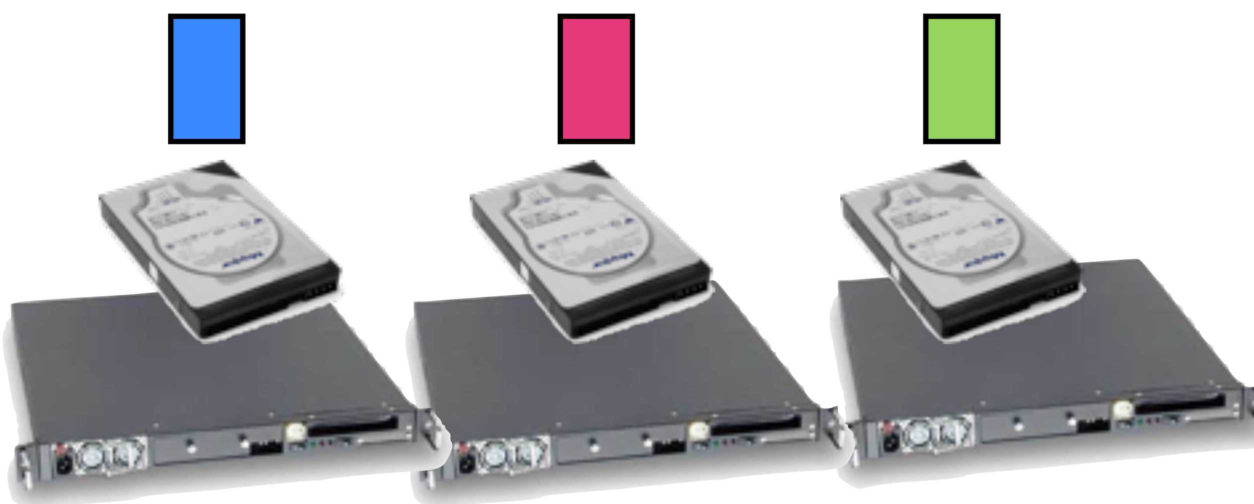
- The code for your mapper processes one record
- The map process executes it for every record in the split
- In streaming, record == line of text
- It gets passed in one (key, value) pair, and output with a new (key, value) pair.

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)
```

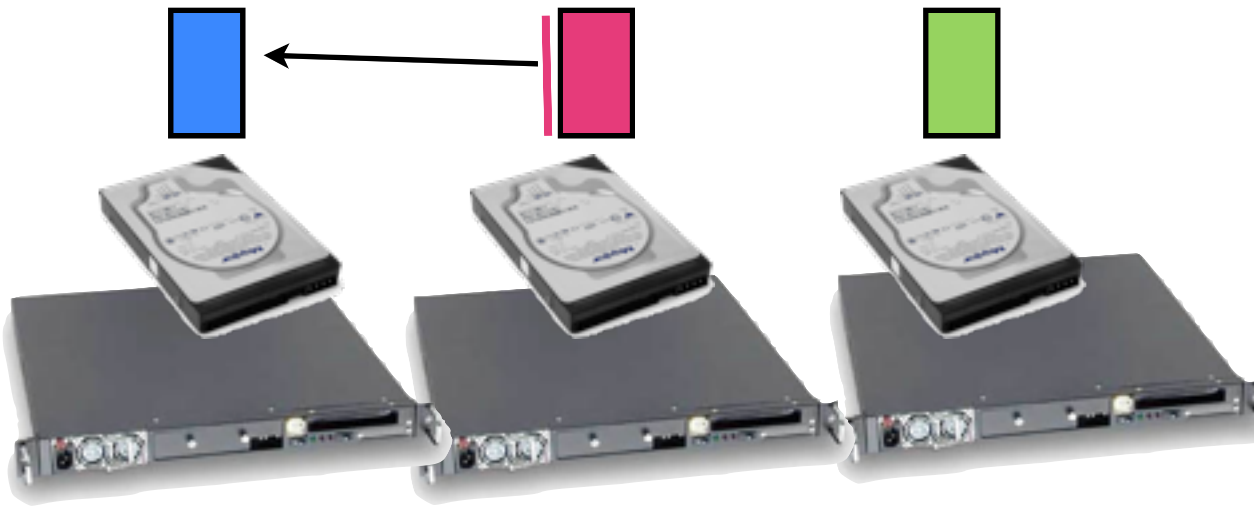
Mapper and I/O

- 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)



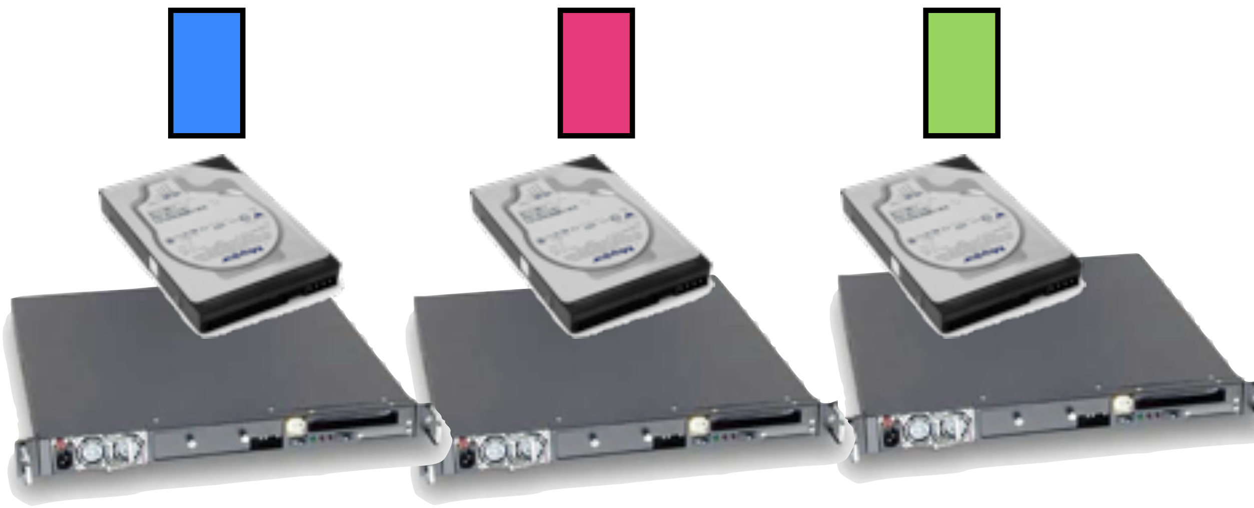
Mapper and I/O

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



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



Quick Quiz

The command to the right isn't quite an accurate model of how the Hadoop streaming job is run

One mapper per file (and per split if large files)

What would be a better command-line equivalent? (Assume one reducer)

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

SciNet compute • calcul CANADA

Mapper and I/O

- In full Hadoop API, 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);
  }
}
```

SciNet compute • calcul CANADA

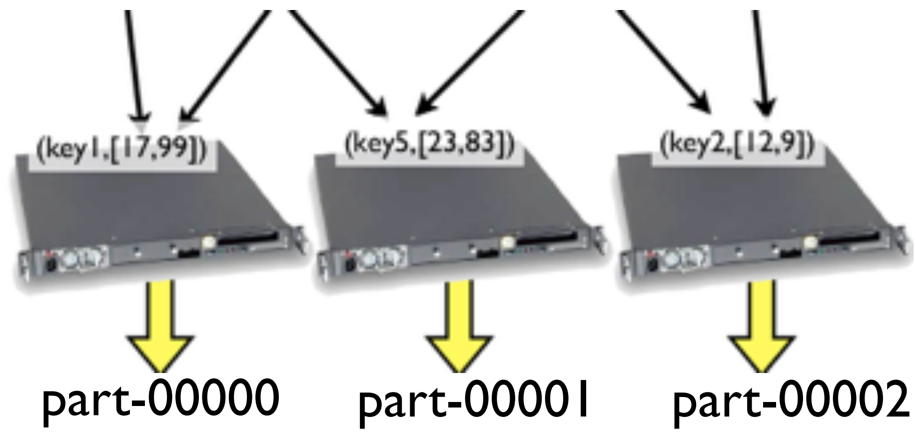
Reducer and I/O

- 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-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);
  }
}
```

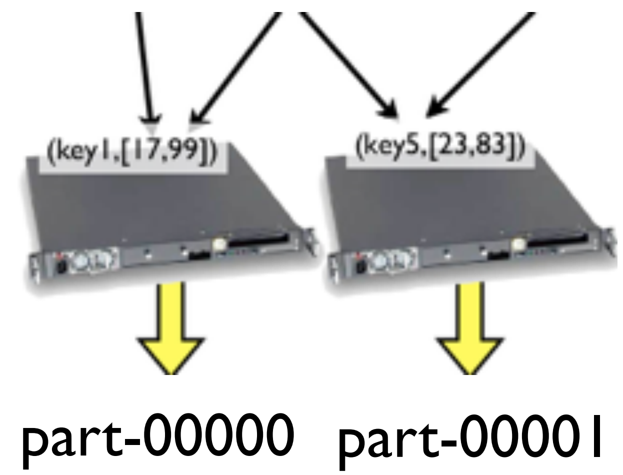
Number of reducers

- 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-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



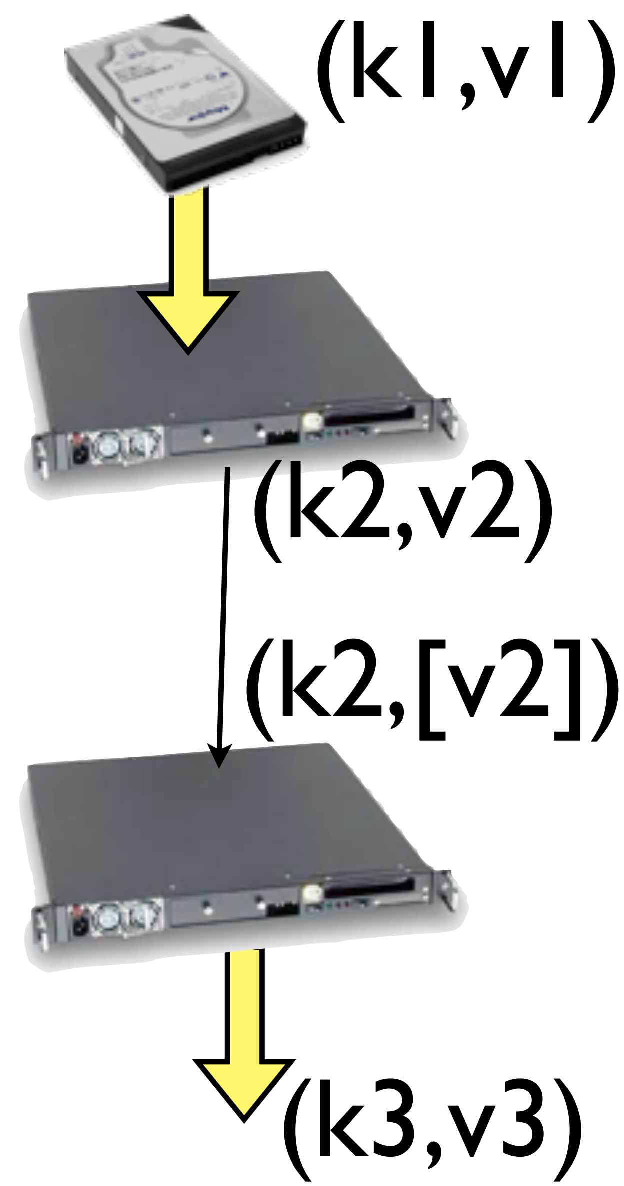

```
Goodbye 1
Hadoop  2
```

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

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);
  }
}
```

SciNet compute • calcul CANADA

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)

```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()) {
      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.)

```
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));
  }
}
```

Combiners

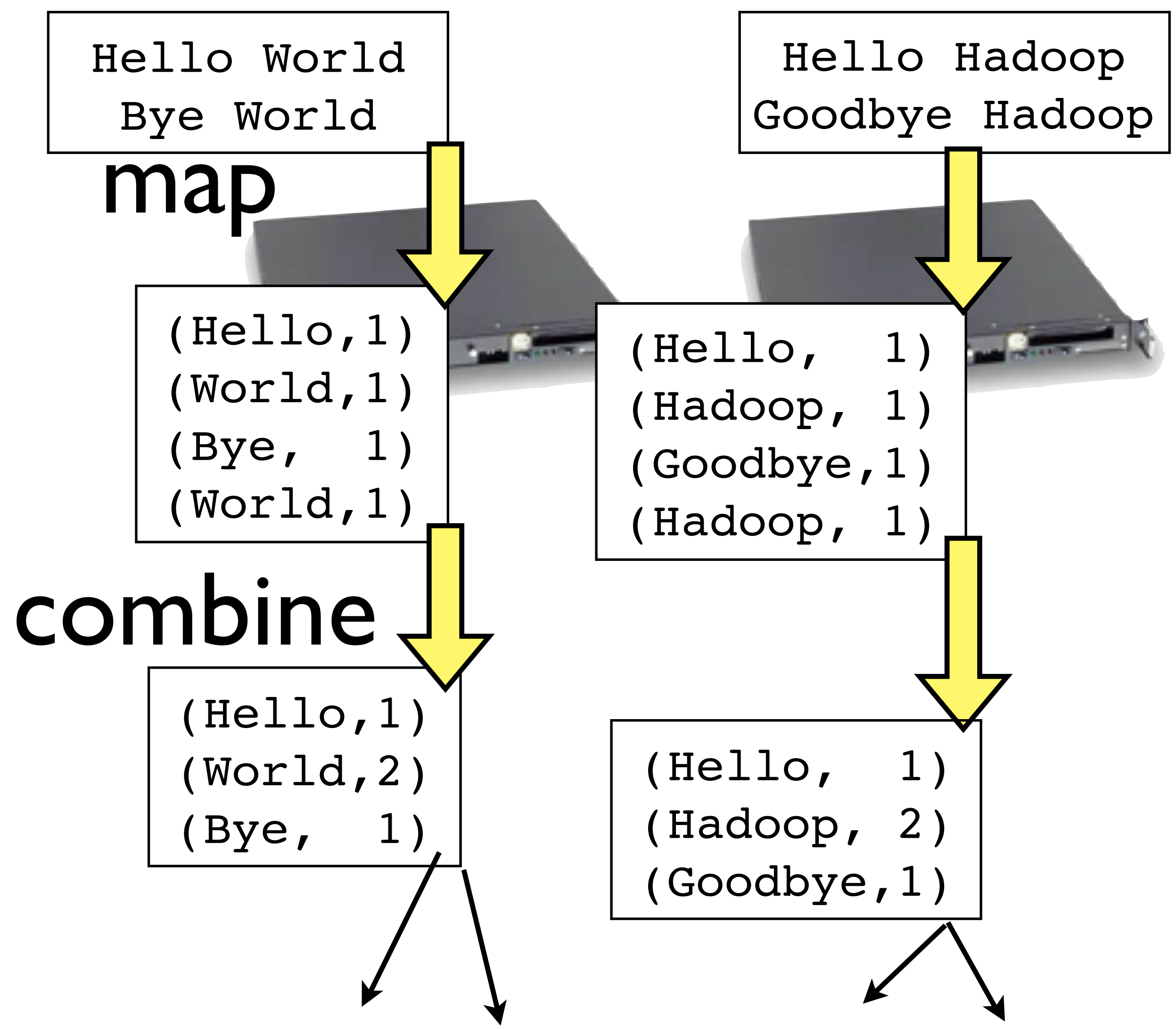
- One more useful thing to know
- You can have a "combiner".
- Run by each mapper on the output of the mapper, before its fed to the shuffle.
- Required (k2,[v2]) → (k2,v2)

```
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

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



SciNet compute • calcul CANADA

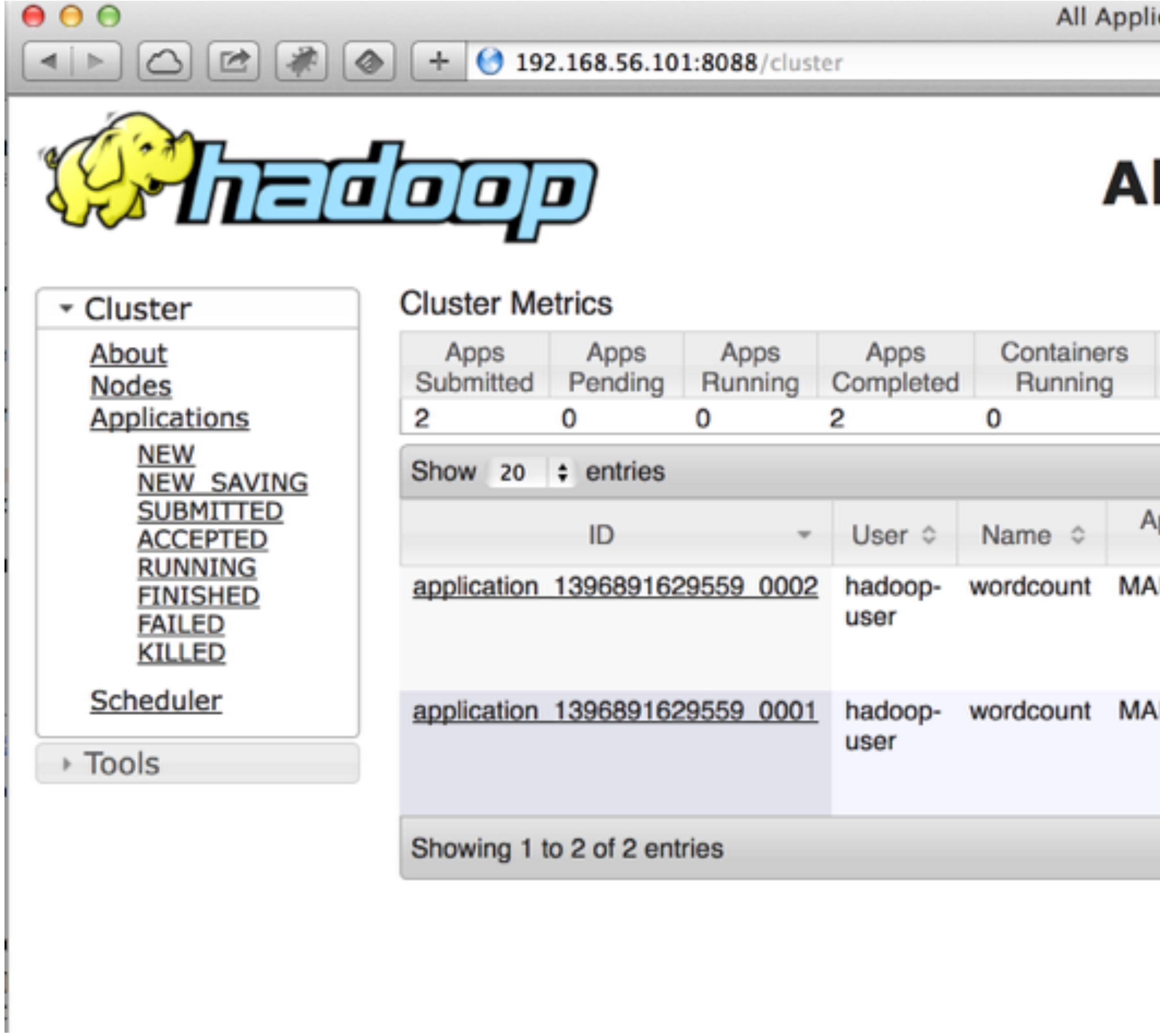
Combiners

- 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.

```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);
  }
}
```

Web Monitor

- Open browser on laptop
- go to (e.g.) http://192.168.33.10:8088
- 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



Map Reduce for Data Exploration

- Recall: assumption is data is very large (at least multiple TB)
- *Don't* want to make many passes through it.
- Many important initial data exploration tasks fit well into map-reduce framework with one or few passes.
- Consider simple summary statistics like min/mean/max:

map:

```
...
0.925
0.055
-0.430
-0.257
-0.473
0.348
...
```

-0.430 → (min,-.430)
-0.430 → (mean, 1,-.430)
-0.430 → (max, -.430)

combine/reduce:

(min, [0.925,0.055,-.430…,0.348]) → (min,-.430)

(max, [0.925,0.055,-.430…,0.348]) → (max,0.925)

~/examples/minmeanmax

SciNet compute • calcul CANADA

Histograms

- How about histogram?
- If knew bin locations, pretty easy - exactly like word count
  - mapper calculates bin (key)
  - emit (key, 1)
  - combiner/reducer sums counts
- But normally you want to make histogram from smallest to largest; how do you do that here? Can't calculate min/max and assign bins at the same time.

map:

```
…
0.925
0.055
-0.430   ──→ (bin11, 1)
-0.257
-0.473
0.348
…
```

combine/reduce:

(bin11, [1,1,1....1]) → (bin11,17)

(bin27,[1,1,1,...1]) → (bin27,83)

SciNet compute • calcul CANADA

Chained MR jobs

- Run two MR jobs.
- Min/Mean/Max - find extent of data
- Histogram - assign bins based on Min/Mean/Max, # of desired bins.
- Downside: 2 passes through data

map:

```
…
0.925
0.055
-0.430
-0.257
-0.473
0.348
…
```

-0.430 → (min,-.430)

-0.430 → (mean, 1,-.430)

-0.430 → (max, -.430)

combine/reduce:

(min, [0.925,0.055,-.430…,0.348]) → (min,-.430)

(max, [0.925,0.055,-.430…,0.348]) → (max,0.925)

map:

```
…
0.925
0.055
-0.430
-0.257
-0.473
0.348
…
```

-0.430 → (bin11, 1)

combine/reduce:

(bin11, [1,1,1….1]) → (bin11,17)

(bin27,[1,1,1,…1]) → (bin27,83)

SciNet compute • calcul CANADA

Distributed cache

- Can send additional files out to mappers/reducers
- Task can then read in to memory and use
- Should be small!
- Here, nbins + min/max data is enough to let us calculate the bin #.

```
hadoop jar $(TOOLLIBS_DIR)/hadoop-streaming-$(HADOOP_VERSION).jar \
    -files ./hist-map.py,./hist-combine-reduce.py,./nbins,./mmm \
    -mapper ./hist-map.py  -combiner ./hist-combine-reduce.py \
        -reducer ./hist-combine-reduce.py \
    -input $(INPUT_DIR) \
    -output  $(OUTPUT_DIR) ; \
```

SciNet compute • calcul CANADA

Distributed cache

- Data is read at initialization time of the map task
- From there, get dx, xmim
- |(x-xmin)/dx| = bin num.

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

# read parameters from distributed cache - nbins, minmeanm
f = open('nbins','r')
params = f.readline().strip().split()
nbins = int(params[0])
f.close()

f = open('mmm','r')
params = f.readline().strip().split()
xmin = float(params[1])
params = f.readline().strip().split()
params = f.readline().strip().split()
xmax = float(params[1])
f.close()

dx = (xmax-xmin)/nbins
```

histogram/hist-map.py

Histogram example

```
$ make run-test
cat input/small-data.dat | ./mmm-map.py  | ./mmm-combine.py \
         | sort |  ./mmm-reduce.py  > mmm
cat mmm
max  1.759767
mean 0.014911
min  -2.153456
cat input/small-data.dat | ./hist-map.py  | \
         ./hist-combine-reduce.py | sort -n | ./hist-combine-reduce.py
-2.023015    1
-1.762134    2
-1.501252    8
-1.240371    43
-0.979489    133
-0.718608    381
-0.457726    695
-0.196844    947
 0.064037    1021
 0.324919    861
 0.585800    531
 0.846682    254
 1.107563    92
 1.368445    25
 1.629326    6
```

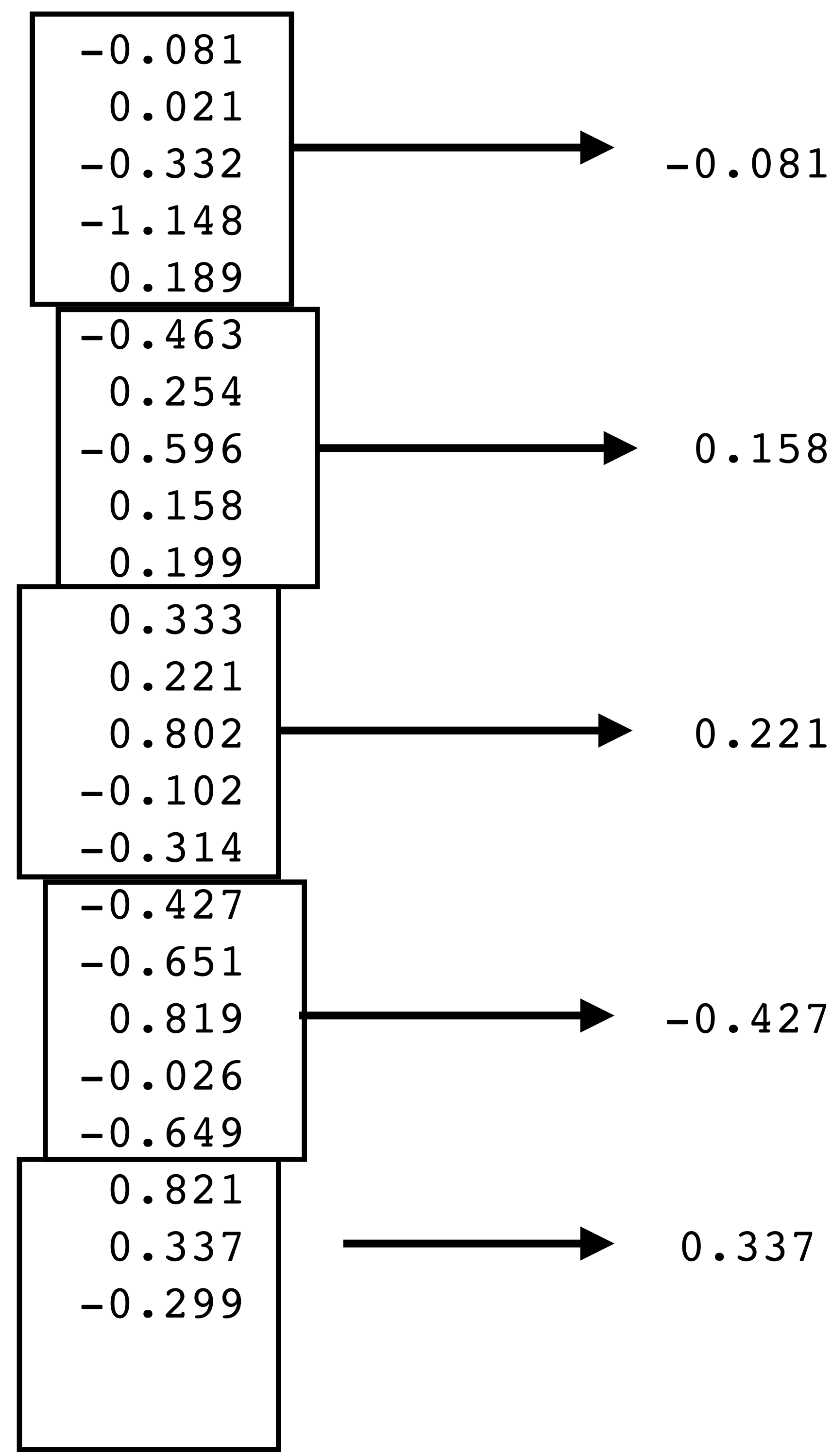
Median?

- Mean was pretty straightforward
- How about something like median?
- Normal approach: sort, take middle element
- Can't have entire data set ni memory
- Sort _all_ billion data points to get one number?

```
…
0.925
0.055
-0.430
-0.257
-0.473
0.348
…
```

Median?

- Very effective approximate approach: Median of Medians
- Partition list into groups of 5 (say); find median of those
- Then find median of those numbers…



Median?

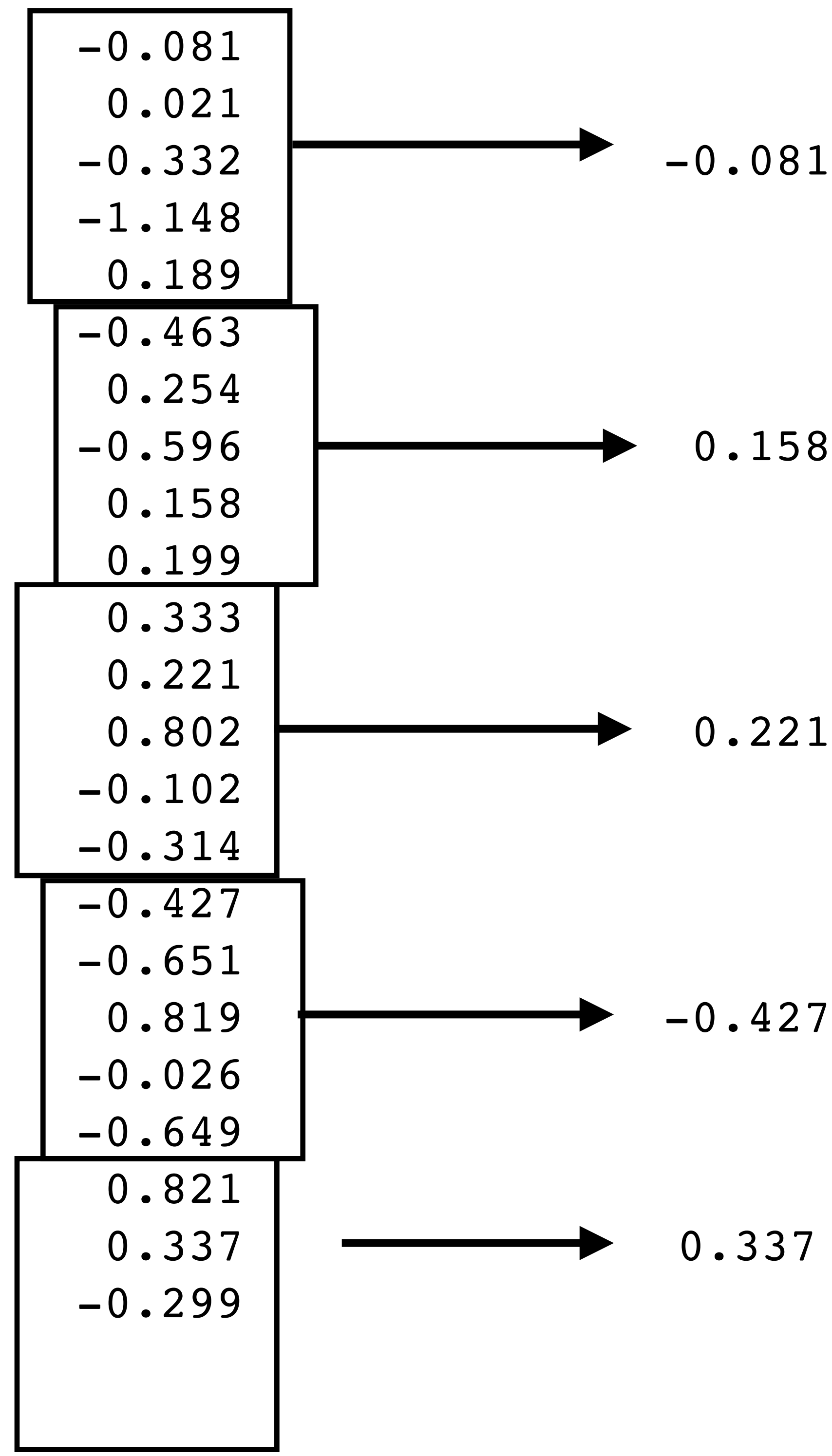
- Amount of data read/written:

$$N\left(1+\frac{1}{5}\right)+\frac{N}{5}\left(1+\frac{1}{5}\right)+\cdots$$

$$=\frac{6}{5}N\sum_{i=0}^{\infty}\frac{1}{5^i}$$

$$=\frac{3}{2}N$$

- Lots of job startups, but amount of I/O quite reasonable

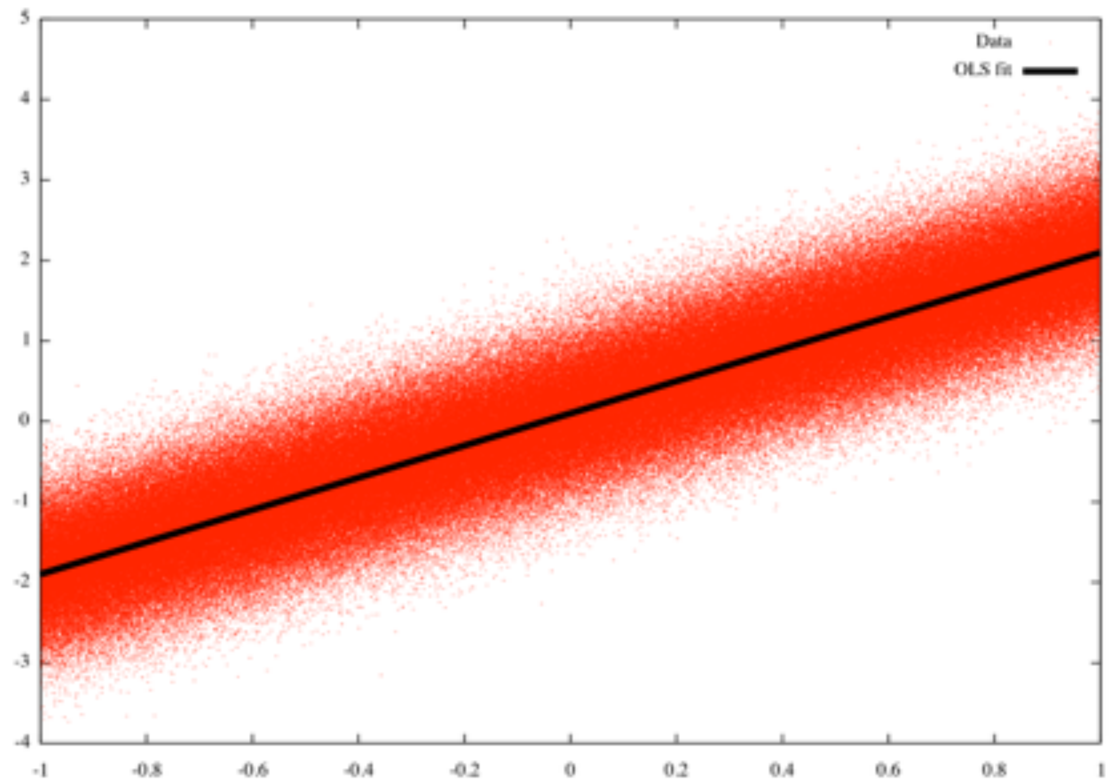


One or few-pass

- As data volumes get larger, more important to be able to process data in a single pass, or at least with O(N) amount of data read/written
- "Online" or "streaming" algorithms have been an active topic of research - many exist and are already coded up.
- Even simple ones are subtle! Randomly select items with equal priority without knowing before hand how many items there are - reservoir

Ordinary Least-Squares

- (eg, linear least squares)
- A very simple starting point for examining data given a model
- Here we'll fit a simple linear model (linear regression)
- `cd ~/examples/ols/1d`
- `make`



SciNet
compute • calcul
CANADA

1d Least-Squares Linear regression

$$\hat{y}_i = a\hat{x}_i + b + \epsilon$$

- Final results - a and b - are simple sums of values evaluated at each point.
- Need sum(x), sum(y), sum(xy), sum(x^2), count.
- How do we do this with map-reduce?
  - How many reducers?
  - Can/should we use a combiner?

$$a, b = \operatorname{argmin} \sum_i (\hat{y}_i - y_i)^2$$

$$= \operatorname{argmin} \sum_i (ax_i + b - y_i)^2$$

$$\frac{\partial r}{\partial b} = \sum_i 2\,(ax_i + b - y_i) = 0$$

$$nb = \sum_i y_i - a \sum_i x_i$$

$$b = \bar{y} - a\bar{x}$$

$$\frac{\partial r}{\partial a} = \sum_i 2\,(ax_i + b - y_i)\,x_i = 0$$

$$= \sum_i 2\,(ax_i + \bar{y} - a\bar{x} - y_i)\,x_i$$

$$a = \frac{\sum_i x_i y_i - n\bar{x}\bar{y}}{\sum_i x_i^2 - n\bar{x}\bar{x}}$$

SciNet compute • calcul CANADA

OLS Map

- Map is what we apply at each point.
- Here is fairly straightforward emit the terms we need to sum.
- $(1, x_i, y_i, x_i\, y_i, x_i^2)$
- Note that key is 1 - could be anything, just has to be the same everywhere.
- Key is the same because…

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

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

    words = line.split()
    x = float(words[0])
    y = float(words[1])

    count = 1
    sumx  = x
    sumy  = y
    sumxy = x*y
    sumxx = x*x

    key = "1"
    print '%s\t%f\t%f\t%f\t%f\t%f' %
        (key, count, sumx, sumy, sumxy, sumxx)
```

ols/1d/map.py

SciNet compute • calcul CANADA

OLS Reduce

- ..only need 1 reducer - reducing everything to just two values
- (*Could* use one reducer for slope, another for intercept, but they need much of the same data - no win).
- Sum up all the individual terms, do very simple processing when all summed received.

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

count = sumx = sumy = 0
sumxy = sumxx = count = 0

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

    key, pcount, psumx, psumy,
                psumxy, psumxx = line.split()
    count = count + float(pcount)
    sumx  = sumx  + float(psumx)
    sumy  = sumy  + float(psumy)
    sumxy = sumxy + float(psumxy)
    sumxx = sumxx + float(psumxx)

covxy = sumxy - (1./count)*sumx*sumy
varx  = sumxx - (1./count)*sumx*sumx
meanx = sumx/count
meany = sumy/count

m = covxy/varx
b = meany - m*meanx

print '%s\t%s' % ("out","y = " + str(m) + " x + " + str(b))
```

ols/1d/reduce.py

SciNet compute • calcul CANADA

OLS Combine

- Combine is basically just the reduce without the after-sum processing yet.
- Avoids sending unnecessary data across the network - just do the partial sums.
- Note: in streaming mode, could do the partial summing in mapper; but that's an accident of the implementation.
- If need to go beyond streaming interface (binary readers, etc) can't do that any more.

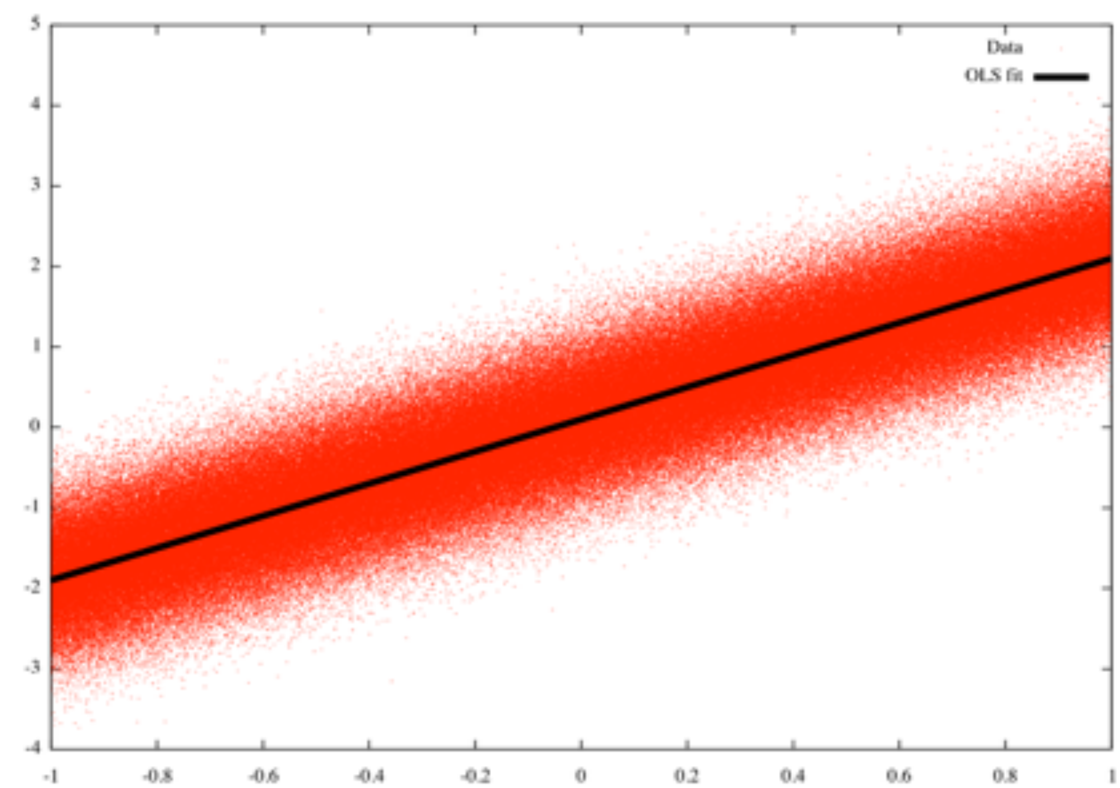
```python
#!/usr/bin/env python
import sys

count = sumx = sumy = 0
sumxy = sumxx = 0

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

    key, pcount, psumx, psumy,
        psumxy, psumxx = line.split()
    count = count + float(pcount)
    sumx  = sumx  + float(psumx)
    sumy  = sumy  + float(psumy)
    sumxy = sumxy + float(psumxy)
    sumxx = sumxx + float(psumxx)

print '%s\t%f\t%f\t%f\t%f\t%f' %
    (key, count, sumx, sumy, sumxy, sumxx)
```

ols/1d/combine.py

SciNet compute • calcul CANADA

OLS 1D

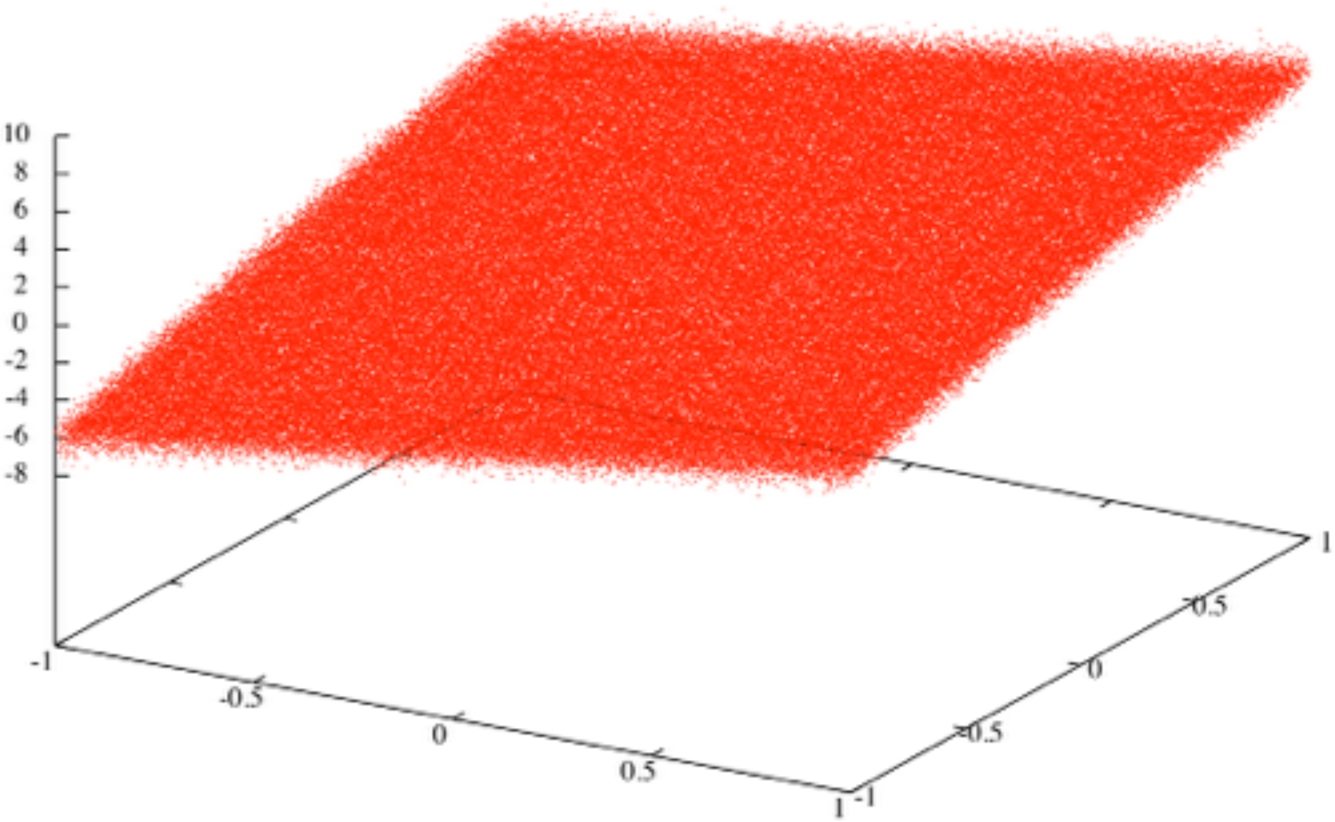
- Run the example if you haven't already
- Compare it with the speed of running it with cat/map/reduce in the shell, locally
- What are the speed differences, and why (the `time` command may be useful here)
- How do the results compare as a function of size? (make input/small-data.dat and change the Makefile to run the hadoop script on the smaller size file).



SciNet compute • calcul CANADA

OLS 3D

- Should really be OLS 2d
- `cd ~/examples/ols/3d`
- `make`
- Same basic structure…



Multi-d Least-Squares Linear regression

- The generalization of 1d is matrix operations
- X is a matrix of multiple variables (columns) and points (rows)
- One column is all ones (for the constant term)
- Requires a matrix solve for the vector of unknowns β

1d

$$a = \frac{\sum_i x_i y_i - n\bar{x}\bar{y}}{\sum_i x_i^2 - n\bar{x}\bar{x}}$$

multi-d

$$\hat{y}_i = \beta X_i + \epsilon$$

$$\hat{\beta} = \left(X^T X\right)^{-1} X^T \mathbf{y}$$

$$\left(X^T X\right) \beta = X^T \mathbf{y}$$

Multi-D OLS: Map

- map.py isn't that different, except:
  - Can use numpy in the scripts (must be installed on all cluster nodes)
  - $X^T X$ is generalization of x^2
  - $X^T y$ is generalization of $x\ y$
- Note we're wasting a lot of time working with numbers and then changing them back and forth to strings - that's just because we're using streaming interface for simplicity.

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

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

    words = line.split()
    x = float(words[0])
    y = float(words[1])
    z = float(words[2])

    xvec = numpy.array([1.,x,y])

    xTx  = numpy.outer(xvec,xvec)
    xTy  = z*xvec

    count = 1

    key = "1"
    print '%s\t%d\t%s\t%s' %
        (key, count,
         numpy.array_str(xTy).translate(None,'[]\n'),
        numpy.array_str(xTx).translate(None,'[]\n'))
```

ols/3d/map.py

Multi-D OLS: Reduce

- Similarly, reduce.py takes advantage of numpy arrays, "fromstring";
- can use linear algebra package to easily solve for parameters
- Note that multi-d case is actually shorter than 1d case, and more general.
- Again, combiner is just the reducer without the final solve step.

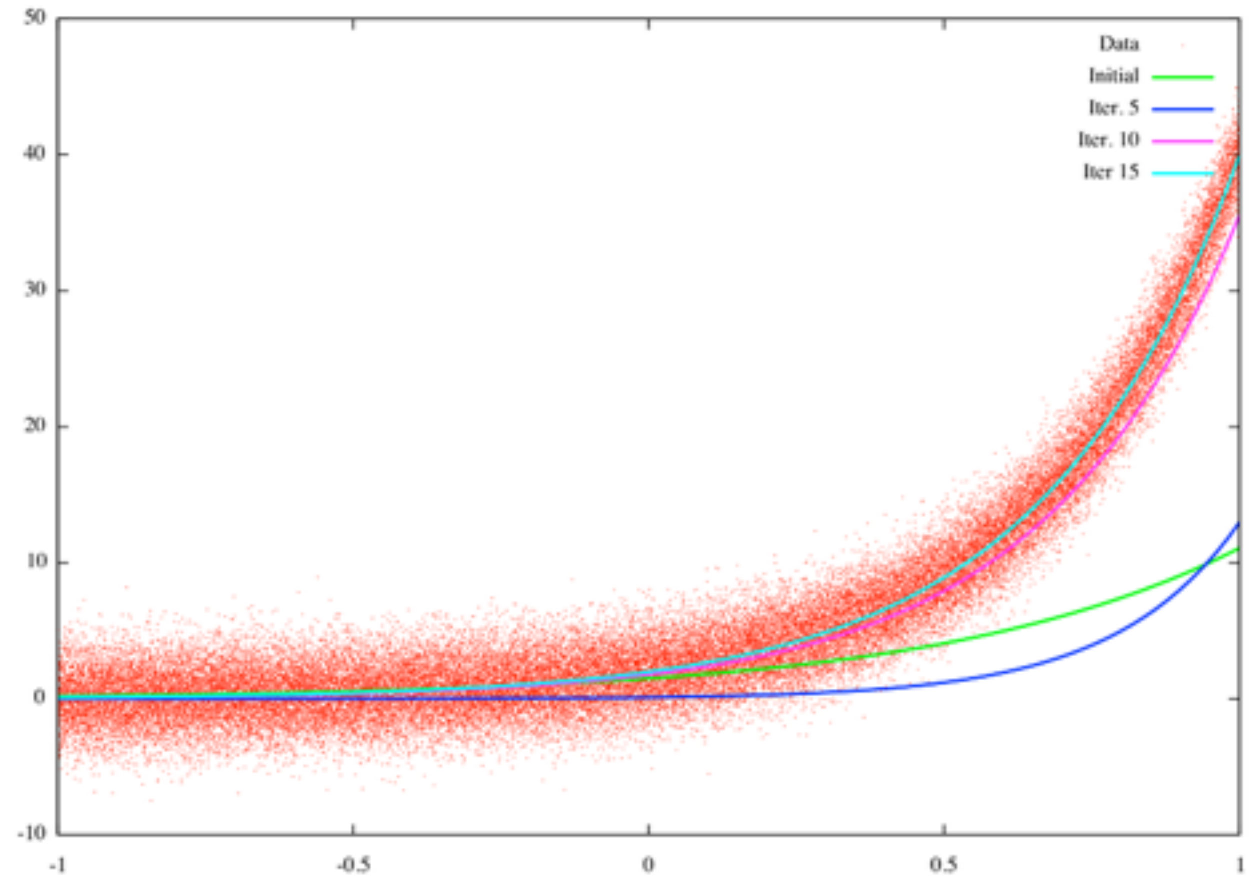
```python
#!/usr/bin/env python
import sys
import numpy
import numpy.linalg

xTx = numpy.zeros((3,3))
xTy = numpy.zeros(3)
count = 0

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

    key, pcount, pxTy, pxTx, = line.split('\t')
    count = count + float(pcount)
    xTy   = xTy + numpy.fromstring(pxTy, sep=' ')
    xTx   = xTx + numpy.fromstring(pxTx, sep=' ').reshape(3,3)

params = numpy.linalg.solve(xTx,xTy)
print '%s\t%s' % ("out",numpy.array_str(params))
```

SciNet compute • calcul CANADA

ols/3d/reduce.py

Nonlinear Least Squares

- However, a lot of scientific computing is inherently iterative
- Have to repeatedly calculate things
- Non-linear least squares - when parameters couple
- `cd ~/examples/nlls/mapreduce`
- `make run-test`
- `make run`



$$y(x) = ae^{bx}$$

Nonlinear Least Squares

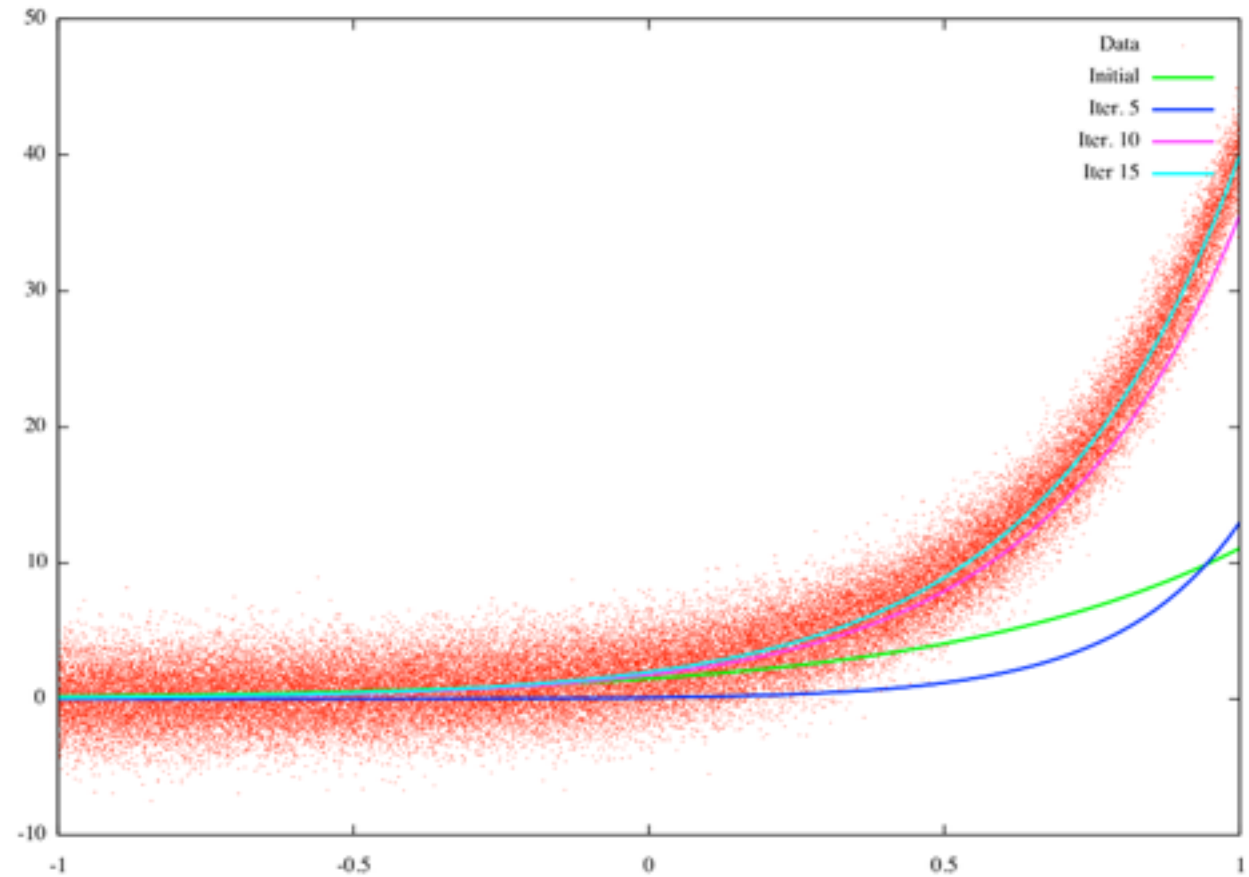
- We choose a very simple method - Gauss-Newton.
- (Not robust enough to use in real work, but simple. More complex methods work work similarly here)

$$f_i^{(k)} = y_i - a^{(k)} \exp(b^{(k)} x_i)$$

$$J_{i,j} = \frac{\partial f_i^{(k)}}{\partial \beta_j}$$

$$= \left(\frac{\partial f_i^{(k)}}{\partial a}, \frac{\partial f_i^{(k)}}{\partial b} \right)$$

$$\mathrm{J}^T \mathrm{J} \boldsymbol{\Delta} \beta^{(k)} = \mathrm{J}^T \mathbf{f}$$



$$y(x) = ae^{bx}$$

SciNet compute • calcul CANADA

Nonlinear Least Squares

- As with multi-D OLS, this method comes down to a linear solve, with all the terms in the matrix and vector just dot products of items easily calculated from the points
  - Map - emit *f* df_i/da, *f* df_i/db, etc
  - Combine/reduce: sum terms (form dot products)
  - Reduce: do linear solve
- Difference: at the end of the linear solve, we don't get the final parameters; we just get an update to the parameters.

$$f_i^{(k)} = y_i - a^{(k)} \exp(b^{(k)} x_i)$$

$$J_{i,j} = \frac{\partial f_i^{(k)}}{\partial \beta_j} = \left(\frac{\partial f_i^{(k)}}{\partial a}, \frac{\partial f_i^{(k)}}{\partial b} \right)$$

$$\mathrm{J}^T \mathrm{J} \boldsymbol{\Delta} \beta^{(k)} = \mathrm{J}^T \mathbf{f}$$

nlls/Makefile

- Iteration is done in scripting the MR jobs.
- Copy starting parameters into file
- For each iteration,
  - Run MR job to update
  - copy output into parameter file
  - display parameters (for our benefit)

```
NITERS=25

run: inputs
	cp orig.params params
	for iter in `seq $(NITERS)`; do \
		hdfs dfs -rm -f -r $(OUTPUT_DIR) ;\
		hadoop jar $(TOOLLIBS_DIR)/hadoop-streaming-$(HADOOP_V
		-files ./map.py,./combine.py,./reduce.py,./params\
		-mapper ./map.py  -combiner ./combine.py  -reducer ./r
		-input $(INPUT_DIR) \
		-output  $(OUTPUT_DIR) ; \
		hdfs dfs -cat $(OUTPUT_FILE) > params; \
		cat params ;\
	done
```

nlls/Makefile

- NITERS job startup overhead:
  - Can take several seconds to start up a job each time; adds up.
- Each map job, new parameters are read in from disk

```
NITERS=25

run: inputs
	cp orig.params params
	for iter in `seq $(NITERS)`; do \
		hdfs dfs -rm -f -r $(OUTPUT_DIR) ;\
		hadoop jar $(TOOLLIBS_DIR)/hadoop-streaming-$(HADOOP_V
		-files ./map.py,./combine.py,./reduce.py,./params\
		-mapper ./map.py  -combiner ./combine.py  -reducer ./r
		-input $(INPUT_DIR) \
		-output  $(OUTPUT_DIR) ; \
		hdfs dfs -cat $(OUTPUT_FILE) > params; \
		cat params ;\
	done
```

nlls/Makefile

- Worse: each iteration, map job reads entire data set from disk…
- even though nothing has changed since last time!
- No way to persist data in memory over multiple MR jobs
- All persistence is done to disk.
- We'll see better frameworks for handling this type of problem shortly.

```
NITERS=25

run: inputs
    cp orig.params params
    for iter in `seq $(NITERS)`; do \
        hdfs dfs -rm -f -r $(OUTPUT_DIR) ;\
        hadoop jar $(TOOLLIBS_DIR)/hadoop-streaming-$(HA
        -files ./map.py,./combine.py,./reduce.py,./param
        -mapper ./map.py  -combiner ./combine.py  -reduc
        -input $(INPUT_DIR) \
        -output  $(OUTPUT_DIR) ; \
        hdfs dfs -cat $(OUTPUT_FILE) > params; \
        cat params ;\
    done
```

SciNet
compute • calcul
CANADA

More Map-Reduce

- The above examples are fairly obvious applications of map + reduce, even if (for NLLS) the original Hadoop MR approach isn't very efficient.
  - MPI equivalent: previous examples could all be done with local computation + MPI_Reduce() operations.
- The functional map + reduce approach can be applied to a large number of technical computing tasks, even ones where the approach isn't so obvious.
- Worth seeing how it works in these cases

Sparse Matrix multiplication

- `cd ~/examples/matmult`
- Reads in matrix name, row, column, value
- Calculate C = AB
- Here A is a permutation matrix and B = A^T, so should get identity matrix back.

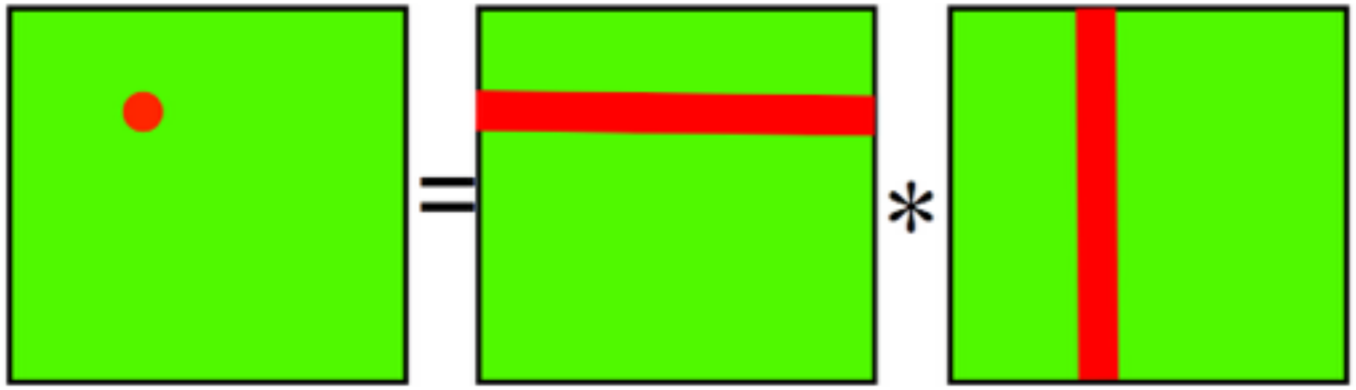
`input/part-00000`

```
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

Sparse Matrix multiplication

- How will this work in Map-Reduce?
- An often fruitful way to approach this question - what should the reduce step loop like?

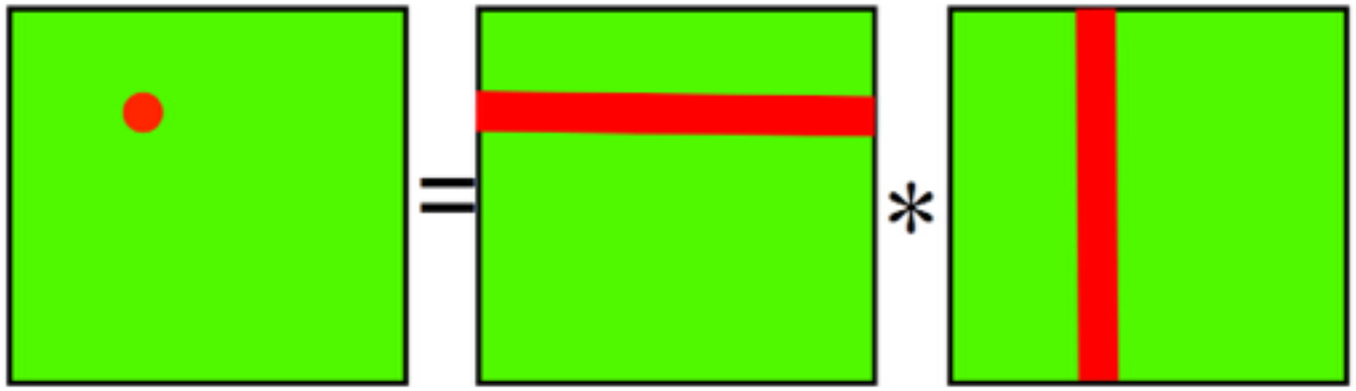


$$C_{i,j} = \sum_k A_{i,k} B_{k,j}$$

SciNet
compute • calcul
CANADA

Sparse Matrix multiplication

- One approach (which an be improved on): have (i,j) be the key.
- The reducer then calculates the dot product of whatever elements were in row i of A and whatever elements were in column j of B.

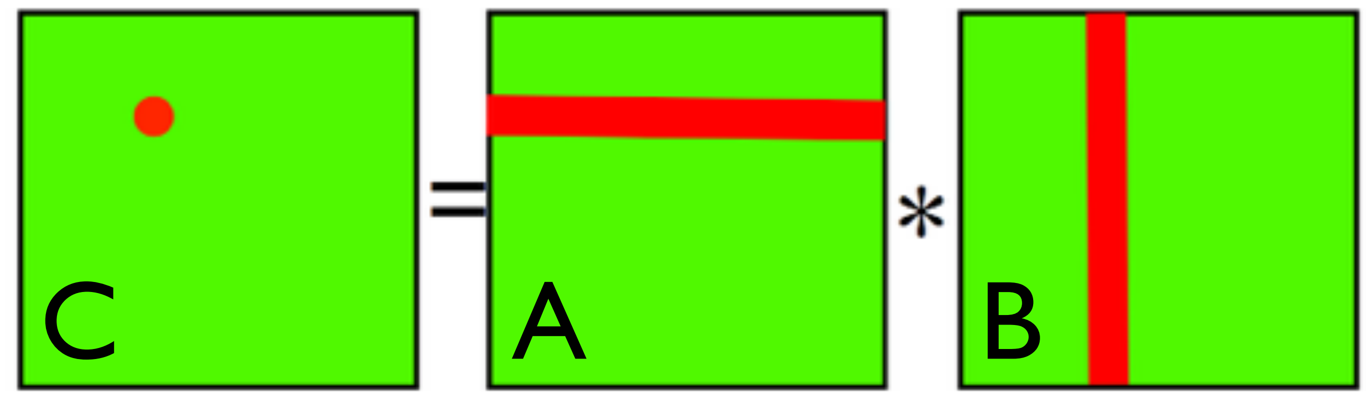


$$C_{i,j} = \sum_k A_{i,k} B_{k,j}$$

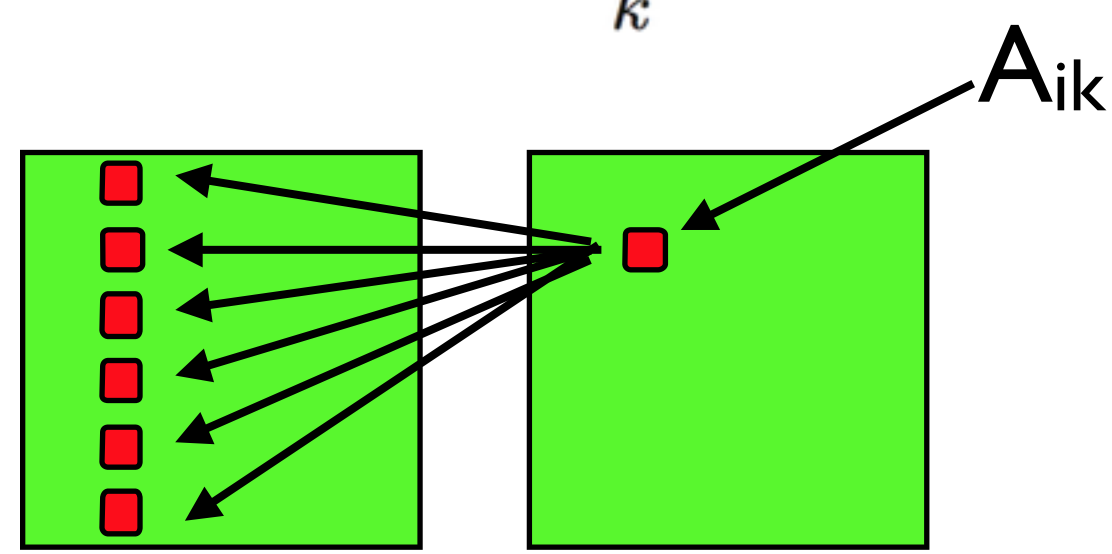
SciNet
compute • calcul
CANADA

Sparse Matrix multiplication

- That means every C_{ij} will need A_{ik}
- So mapper reading A_{ik} will have to emit key-value pairs
  (i,j) ("A", i, k, value)
  for all j.
- Similarly, mapper reading B_{kj} will have to emit key value pairs
  (i,j) ("B", k, j, value)
  for all values of i.
- Then it's the job of the reducer for (i,j) to find any matching ks in the Alist of items from A, B and dot product them together.



$$C_{i,j} = \sum_k A_{i,k} B_{k,j}$$



Sparse Matrix multiplication

- So mapper reading A_{ik} will have to emit key-value pairs
  (i,j) ("A", i, k, value)
  for all j.
- Similarly, mapper reading B_{kj} will have to emit key value pairs
  (i,j) ("B", k, j, value)
  for all values of i.
- Then it's the job of the reducer for (i,j) to find any matching ks in the Alist of items from A, B and dot product them together.

```
import sys
maxrows=100

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

    matrix, row, col, val = line.split()
    row = int(row)
    col = int(col)
    val = int(val)

    if matrix == 'A':
        value = matrix+'-'+str(col)+'-'+str(val)
        for j in xrange(maxrows):
            key = str(row)+'-'+str(j)
            print '%s\t%s' % ( key, value )
    else:
        value = matrix+'-'+str(row)+'-'+str(val)
        for i in xrange(maxrows):
            key = str(i)+'-'+str(col)
            print '%s\t%s' % ( key, value )
```

matmult/map.py

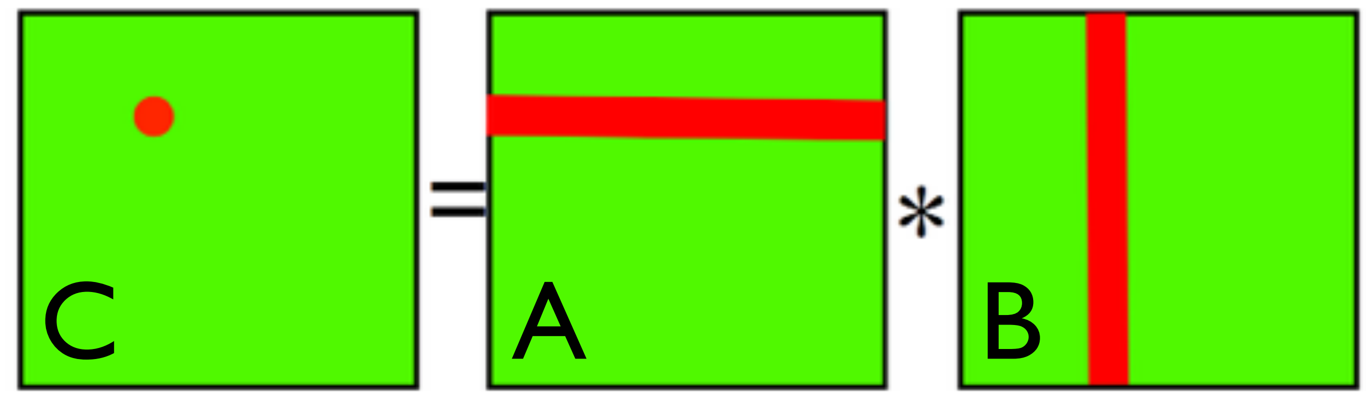
Sparse Matrix multiplication

- So mapper reading A_{ik} will have to emit key-value pairs
  (i,j) (“A”, i, k, value)
  for all j.
- Similarly, mapper reading B_{kj} will have to emit key value pairs
  (i,j) (“B”, k, j, value)
  for all values of i.
- Then it’s the job of the reducer for (i,j) to find any matching ks in the Alist of items from A, B and dot product them together.

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

    key, value = line.split()
    row, col = key.split('-')
    matrix, index, val = value.split('-')

    row = int(row)
    col = int(col)
    val = int(val)
    index  = int(index)

    if row != currentRow or col != currentCol:
        # new key
        if currentRow is not None:
            # calculate, print results
            matrixElement = calcSum(matAvalues, matBvalues)
            printSum(currentRow, currentCol, matrixElement)
            matAvalues = {}
            matBvalues = {}

        # new currentRow/col
        currentRow = row
        currentCol = col

    # put value in maps
    putValue(matrix, index, val, matAvalues, matBvalues)
```

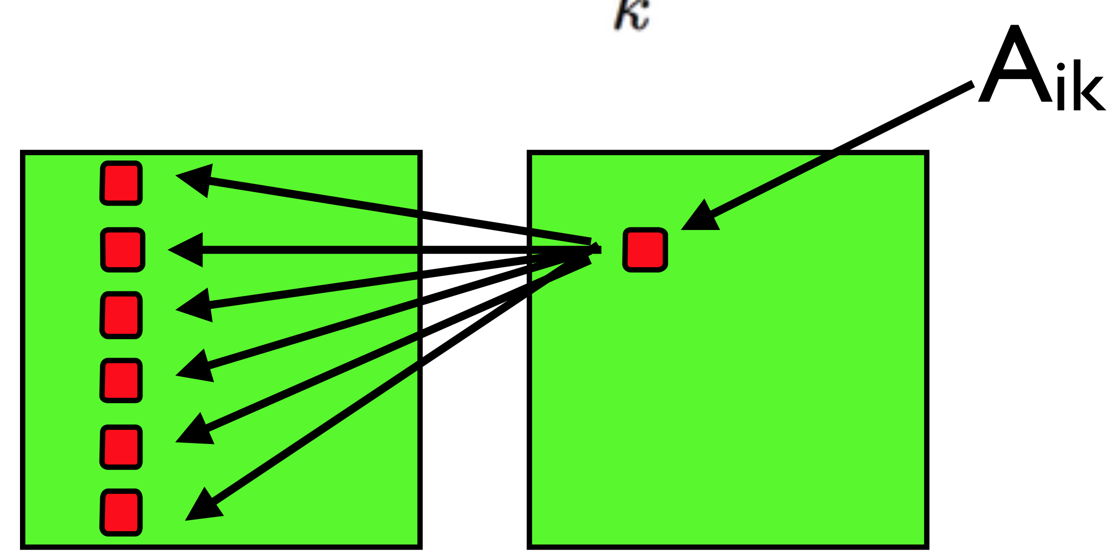
matmult/reduce.py

Sparse Matrix multiplication



- Note that efficiency (eg, communications) could be improved by breaking the matrix up into tiles, rather than individual elements (just as with single-node matrix multiplication)
- Note too if you know something about the structure of the sparse matrix (eg, banded diagonal) can do better.

$$C_{i,j} = \sum_k A_{i,k} B_{k,j}$$

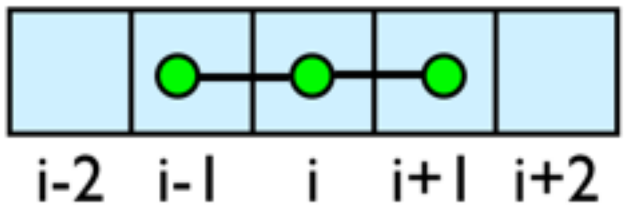


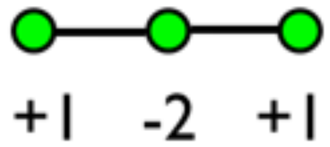
One-D Diffusion

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

```
cd ~/examples/diffuse/mapreduce
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 ...
```

One-D Diffusion

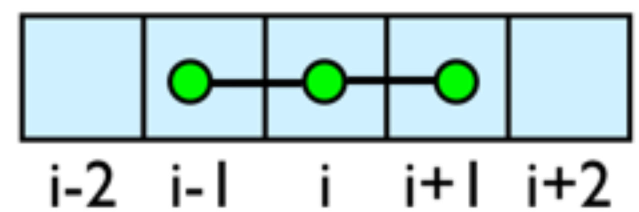
- Key: domain segment
- Value: subarray (numpy)
- What information does the reducer need to update the values?

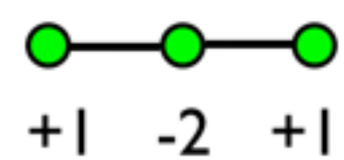
```
 0: 0.0050365 0.00709477 0.01360237 ...
1: 0.16004214 0.19533521 0.28114455 ...
2: 0.84731875 0.89604445 0.96817042 ...
3: 0.74742274 0.68483447 0.55549607 ...
4: 0.10984817 0.08720647 0.05310277 ...
```

One-D Diffusion

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

- Reducer needs guard cell data - one value from each neighbouring segment.
- Map: Read (segment#,data), send leftmost piece of data to key segment-1, rightmost to segment +1, all data to segment
- Reducer for each segment#: update entire subregion with guardcells.





One-D Diffusion

- Reducer needs guard cell data - one value from each neighbouring segment.
- Map: Read (segment#,data), send leftmost piece of data to key segment-1, rightmost to segment +1, all data to segment
- Reducer for each segment#: update entire subregion with guardcells.

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

import sys
import numpy
from StringIO import StringIO

for line in sys.stdin:
    line = line.strip()
    segment, datastr = line.split(':')

    segment = int(segment)
    data = numpy.genfromtxt(StringIO(datastr))

    # left guardcell
    if (segment > 0):
        print '%d:L%f' % (segment-1,data[0])

    # right guardcell
    print '%d:R%f' % (segment+1,data[-1])

    # all data
    print line
```

~/examples/diffuse/mapreduce/map.py

SciNet compute • calcul CANADA

One-D Diffusion

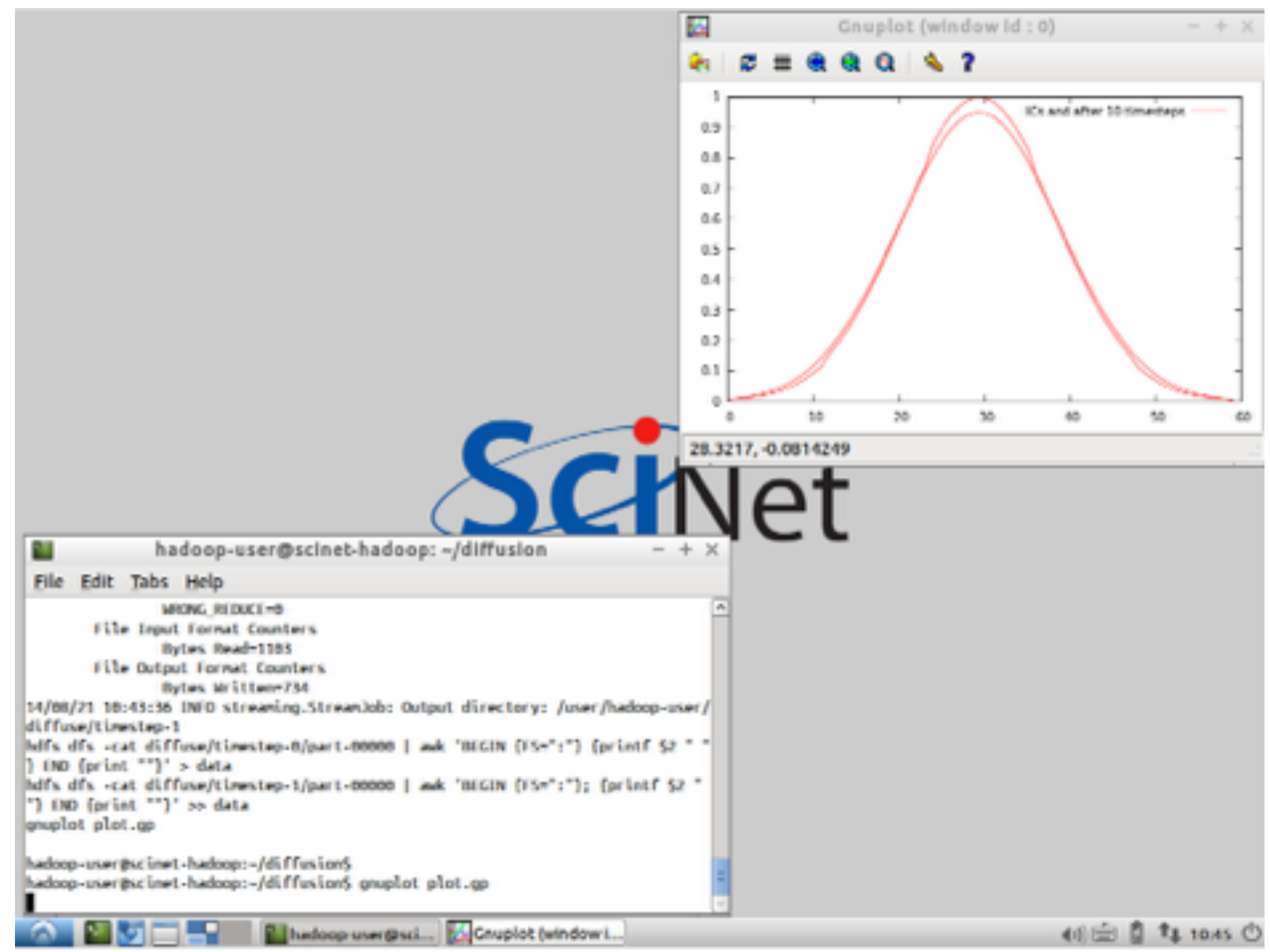
- Reducer needs guard cell data - one value from each neighbouring segment.
- Map: Read (segment#,data), send leftmost piece of data to key segment-1, rightmost to segment +1, all data to segment
- Reducer for each segment#: update entire subregion with guardcells.

```
[...]

def calculate(seg, leftGC, rightGC, data, coeff):
    if seg is not None and data is not None:
        alldata = numpy.append( numpy.append(rightGC, data), leftGC)
        return alldata[1:-1] + coeff*( alldata[0:-2] - 2.*alldata[1:-1] + alldata[2:] )
    else:
        return None

[...]

for line in sys.stdin:
    line = line.strip()
    segment, datastr = line.split(':')

    segment = int(segment)
    if segment != currentSegment:
        # output previous data
        if currentSegment is not None:
            updatedData = calculate(currentSegment, leftGC, rightGC, data, coeff)
            printData(currentSegment, updatedData)

        # init new segment
        currentSegment = segment
    leftGC = 0.
    rightGC = 0.
    data = None

    firstchar = datastr[0]
    if firstchar == 'L':
        leftGC = float(datastr[1:])
    elif firstchar == 'R':
        rightGC = float(datastr[1:])
    else:
        data = numpy.genfromtxt(StringIO(datastr))

if currentSegment is not None:
    updatedData = calculate(currentSegment, leftGC, rightGC, data, coeff)
    printData(currentSegment, updatedData)
```

~/examples/diffuse/mapreduce/reduce.py

One-D Diffusion

- Running - it works!
- Communications-wise, similar to MPI.
- But every iteration involves reading, writing entire domain to disk.



Part III: Pig

Part IV: YARN, Spark

Questions?

Extra Slides

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

```
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)
- vagrant/vagrant

VM:

```
$ ifconfig | grep 192
          inet addr: 192.168.33.10 [...]
```

Host

```
$ scp WordCount.java vagrant@192.168.33.10:
vagrant@192.168.33.10's password: vagrant
```

SciNet compute • calcul CANADA

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);
    }
  }
}
```

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);
    }
  }
}
```

SciNet compute • calcul CANADA

WordCount by Doc

- Reducer is exactly the same

```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));
  }
}
```

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	

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

SciNet compute • calcul CANADA

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

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

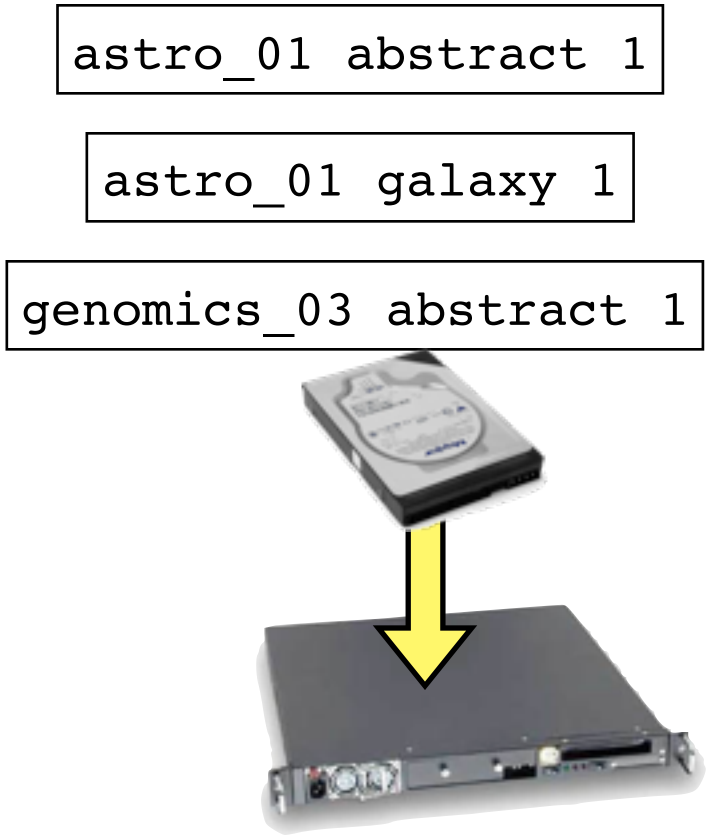
Means key is... ?

Means data it needs is... ?

SciNet compute • calcul CANADA

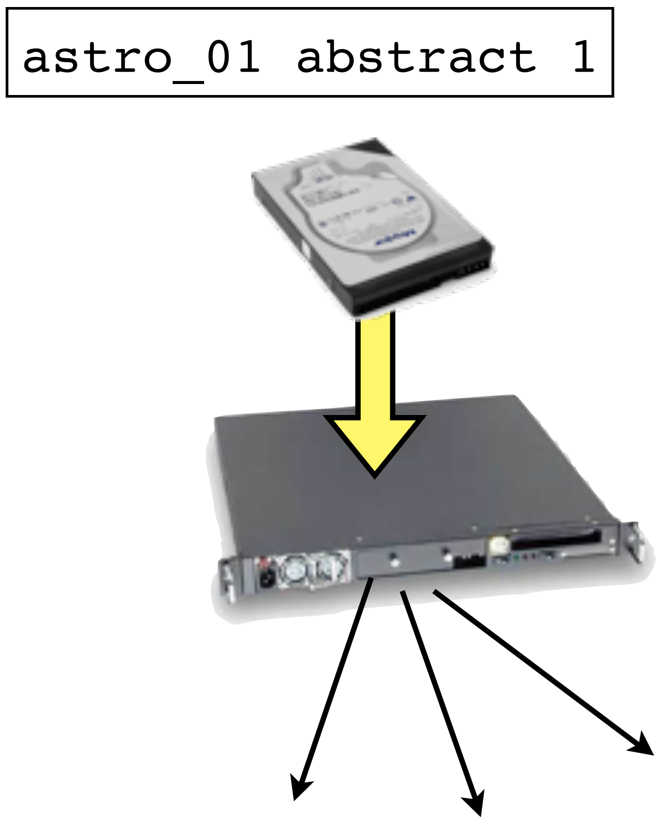
Document Similarity: Mapper

- 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?



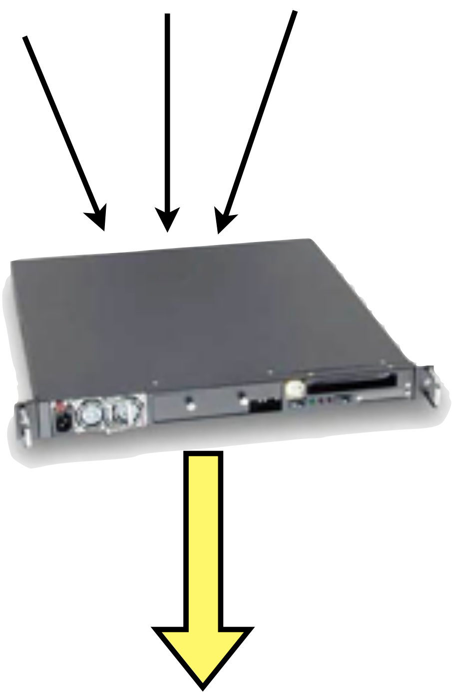
Document Similarity: Mapper

- 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...)



SciNet
compute • calcul
CANADA

Document Similarity: Reducer



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

SciNet compute • calcul CANADA

Document Similarity: Mapper

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

```
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);
      }
   }
}
```

SciNet compute • calcul CANADA

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);
    }
  }
```

SciNet 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)) );
}
```

SciNet compute • calcul CANADA

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

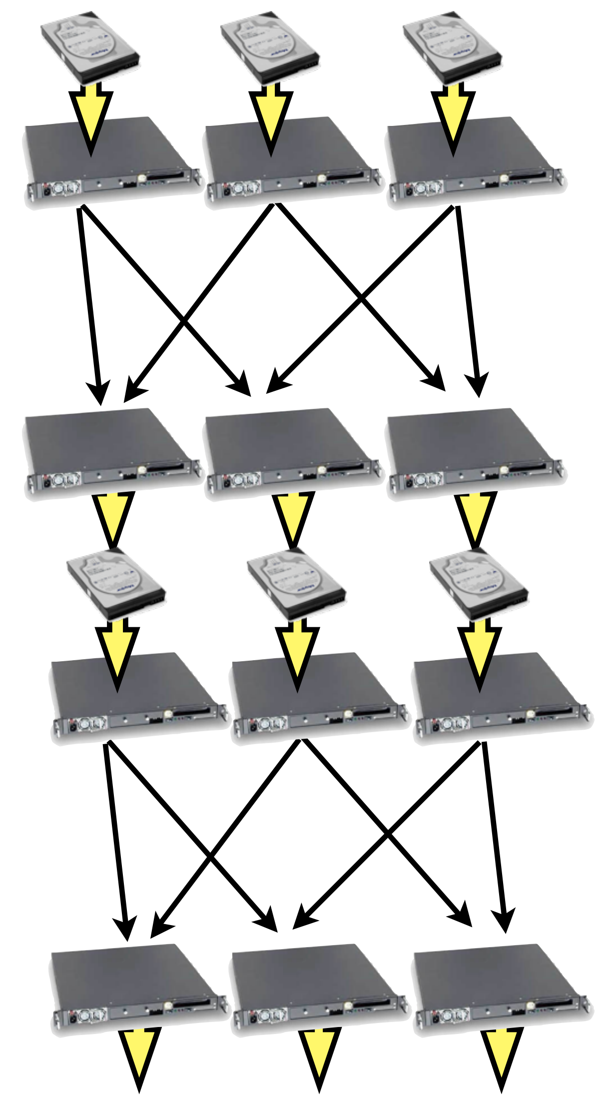
Where do they come from?

```
"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
```

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.

$$S_{i,j} = \mathbf{w_i} \cdot \mathbf{w_j}$$

$$S = WW^T$$

SciNet compute • calcul CANADA