cloudera
Ask Bigger Questions

Introduction to Apache Hadoop

Tom Wheeler | OSCON 2013

About the Instructor…

Tom Wheeler

Software Engineer, etc.

Greater St. Louis Area | Information Technology and Services

Current: **Senior Curriculum Developer** at **Cloudera**

Past: Principal Software Engineer at Object Computing, Inc. (Boeing)

Software Engineer, Level IV at WebMD

Senior Programmer/Analyst at A.G. Edwards and Sons, Inc.

About the Presentation…

- What's ahead
  - Why the World Needs Hadoop
  - Fundamental Concepts
  - HDFS: The Hadoop Distributed File System
  - Data Processing with MapReduce
  - Using Apache Hadoop Effectively
  - The Hadoop Ecosystem
  - Questions and Answers

Why the World Needs Hadoop

And, by the way, what Is Hadoop?

Volume

- Every day…
  - More than 1.5 billion shares are traded on the NYSE
  - Facebook stores 2.7 billion comments and Likes
- Every minute…
  - Foursquare handles more than 2,000 check-ins
  - TransUnion makes nearly 70,000 updates to credit files
- And every second…
  - Banks process more than 10,000 credit card transactions

Velocity

- We are generating data faster than ever
  - Processes are increasingly automated
  - People are increasingly interacting online
  - Systems are increasingly interconnected

Variety

- We're producing a variety of data, including
  - Social network connections
  - Images
  - Audio
  - Video
  - Log files
  - Product rating comments
- Not all of this maps cleanly to the relational model

Disk Capacity and Price

- Fortunately, the size and cost of storage has kept pace
  - Capacity has increased while price has decreased

Year	Capacity (GB)	Cost per GB (USD)
1992	0.08	$3,827.20
1997	2.10	$157.00
2002	80.00	$3.74
2007	750.00	$0.35
2012	3,000.00	$0.05

Big Data Can Mean Big Opportunity

- One tweet is an anecdote
  - But a million tweets may signal important trends
- One person's product review is an opinion
  - But a million reviews might uncover a design flaw
- One person's diagnosis is an isolated case
  - But a million medical records could lead to a cure

“Find something where you provide a scarce complementary service to something that is getting ubiquitous and cheap…”

— Dr. Hal Varian, Google’s Chief Economist

"So what's getting ubiquitous and cheap? Data.

…and what is *complementary* to data? Analysis."

— Dr. Hal Varian, Google's Chief Economist

We Need a System that Scales

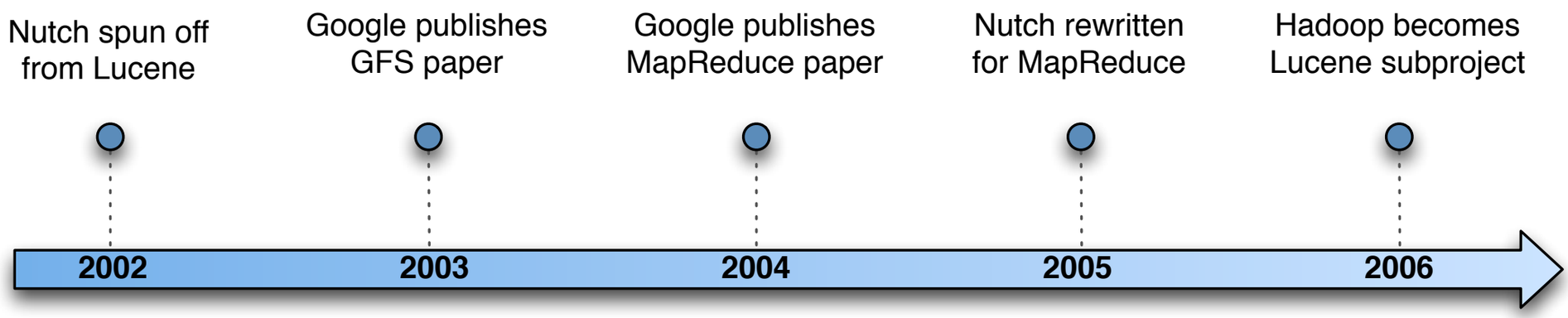
- Too much data for traditional tools
- Two key problems
  - How to reliably store this data at a reasonable cost
  - How to we process all the data we've stored

What is Apache Hadoop?

- Scalable data storage and processing
  - Distributed and fault-tolerant
  - Runs on standard hardware
- Two main components
  - Storage: the Hadoop Distributed File System (HDFS)
  - Processing: MapReduce

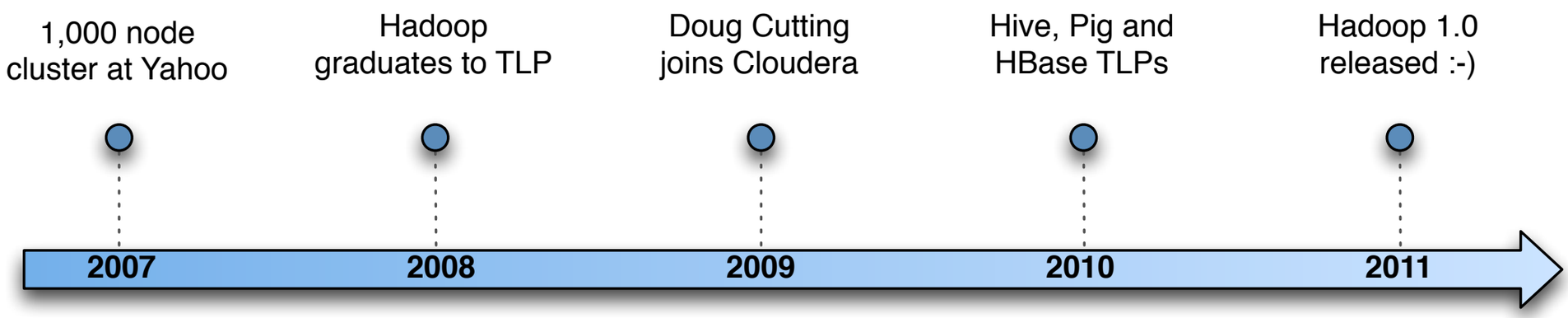
How Did Apache Hadoop Originate?

- Heavily influenced by Google's architecture
  - Notably, the Google Filesystem and MapReduce papers
- Other Web companies quickly saw the benefits
  - Early adoption by Yahoo, Facebook and others



How Did Apache Hadoop Originate (cont'd)?

- Hadoop eventually became its own top-level project
  - And later spawned many new projects of its own
  - Now many industries rely on it, including Fortune 50 firms



How Are Organizations Using Hadoop?

- Just a few examples...
  - Analytics
  - Product recommendations
  - Ad targeting
  - Fraud detection
  - Natural language processing
  - Route optimization

Fundamental Concepts

The Basics of Hadoop's Design

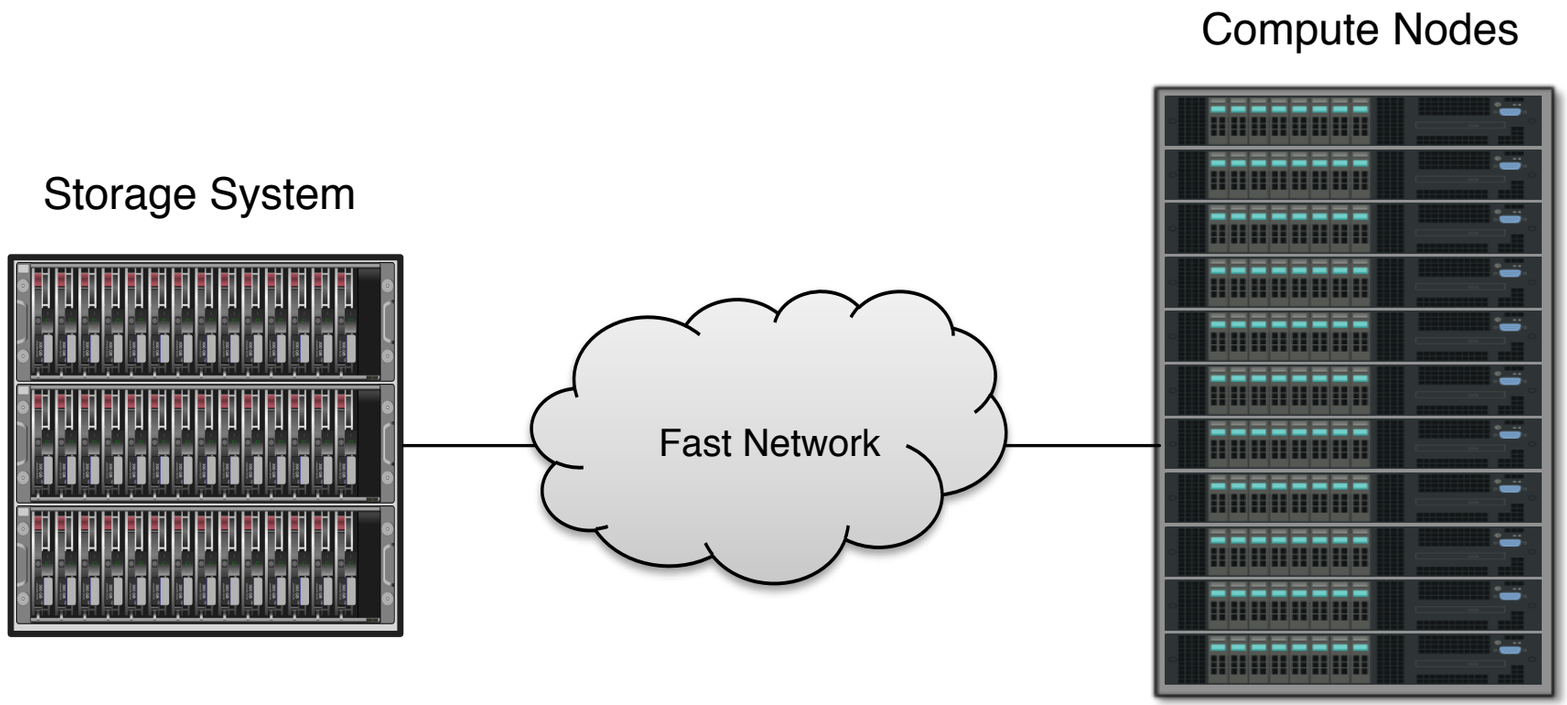
“In pioneer days they used oxen for heavy pulling, and when one ox couldn’t budge a log, we didn’t try to grow a larger ox”

— Grace Hopper, early advocate of distributed computing

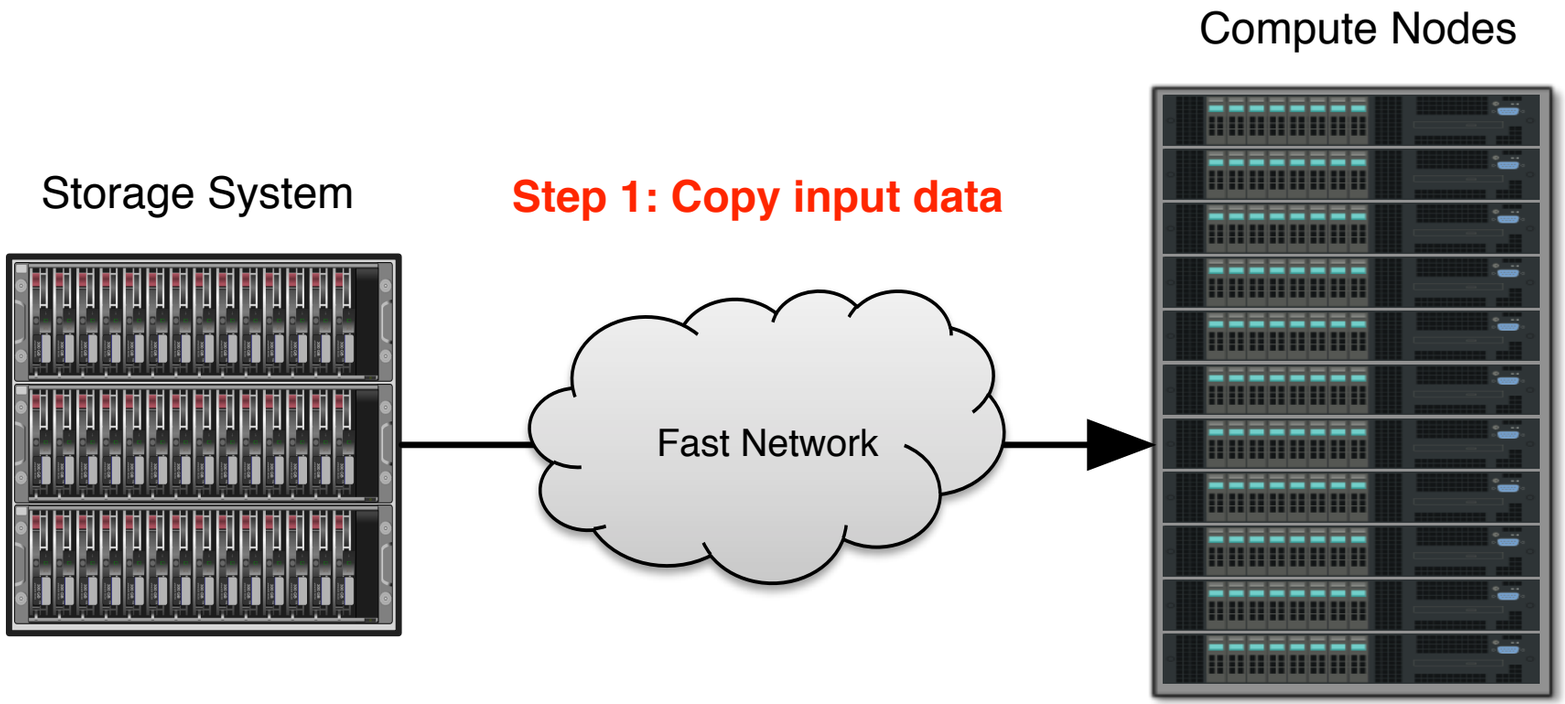
Comparing Hadoop to Other Systems

- Monolithic systems don’t scale
- Modern high-performance computing systems are distributed
  - They spread computations across many machines in parallel
  - Widely-used used for scientific applications
  - Let’s examine how a typical HPC system works

Architecture of a Typical HPC System



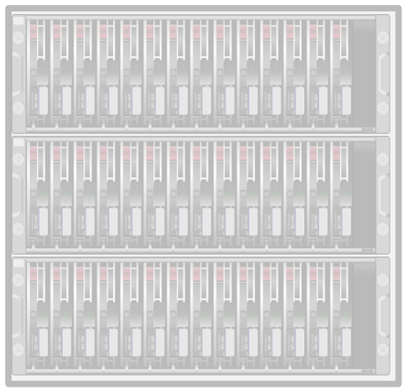
Architecture of a Typical HPC System



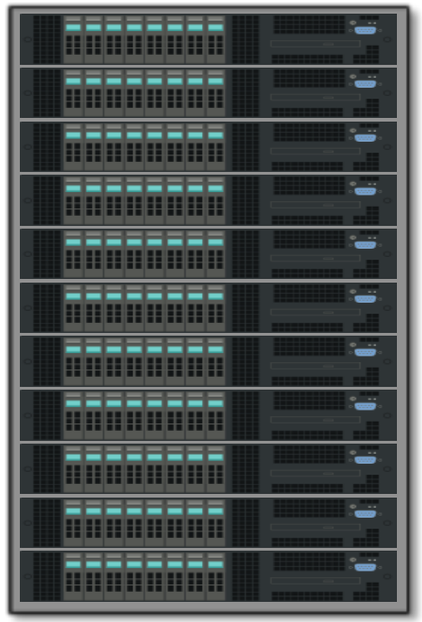
Architecture of a Typical HPC System

Compute Nodes

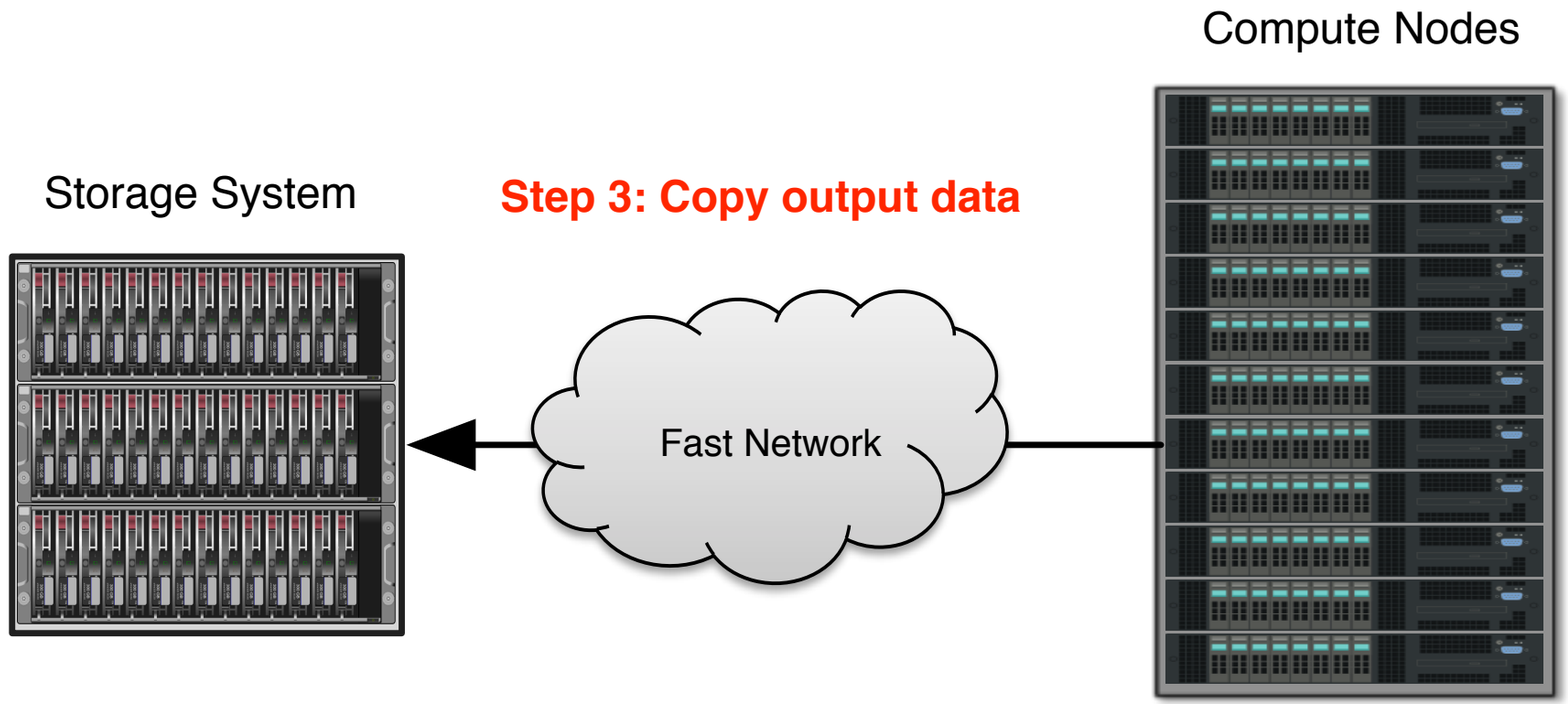
Storage System



Step 2: Process the data



Architecture of a Typical HPC System

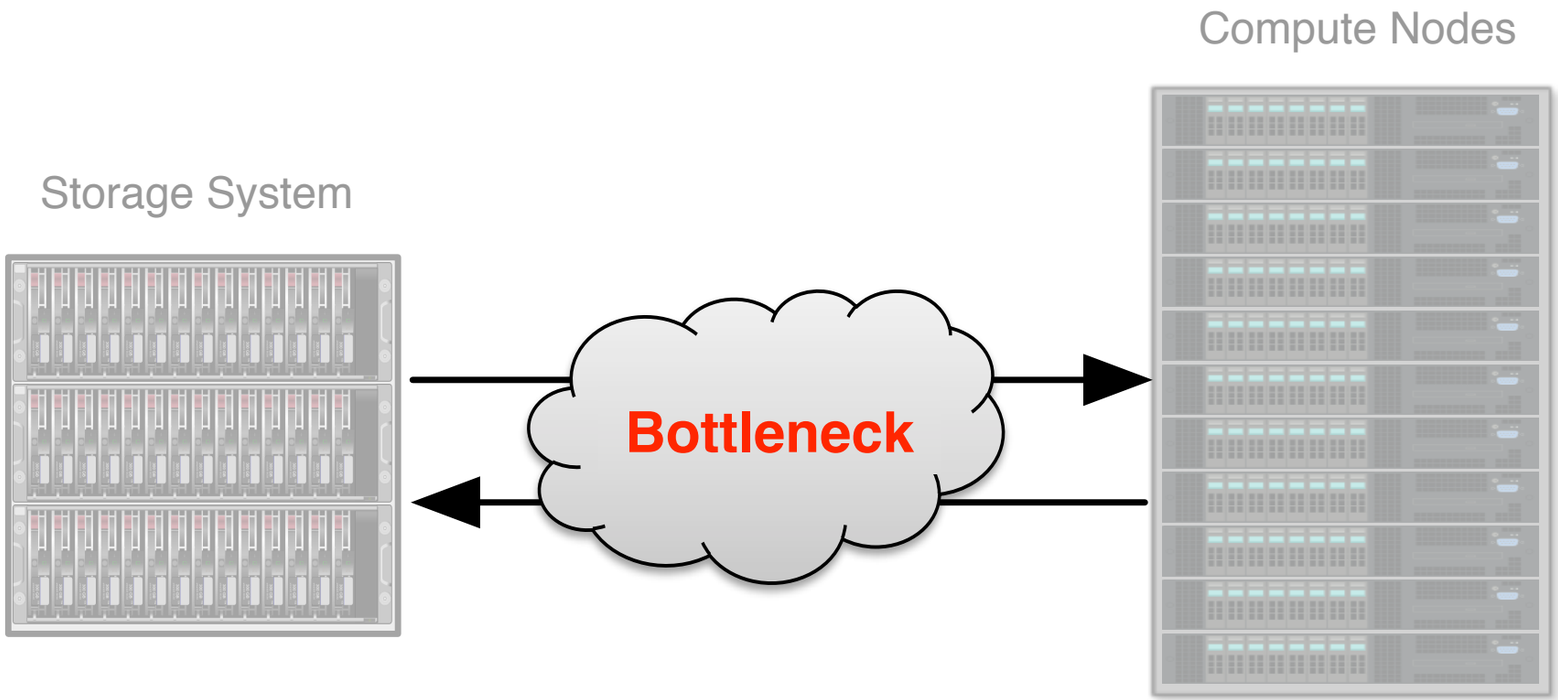


You Don’t Just Need Speed…

- The problem is that we have way more data than code

```
$ du -ks code/
1,083

$ du -ks data/
854,632,947,314
```

You Need Speed *At Scale*



Hadoop Fundamental: Data Locality

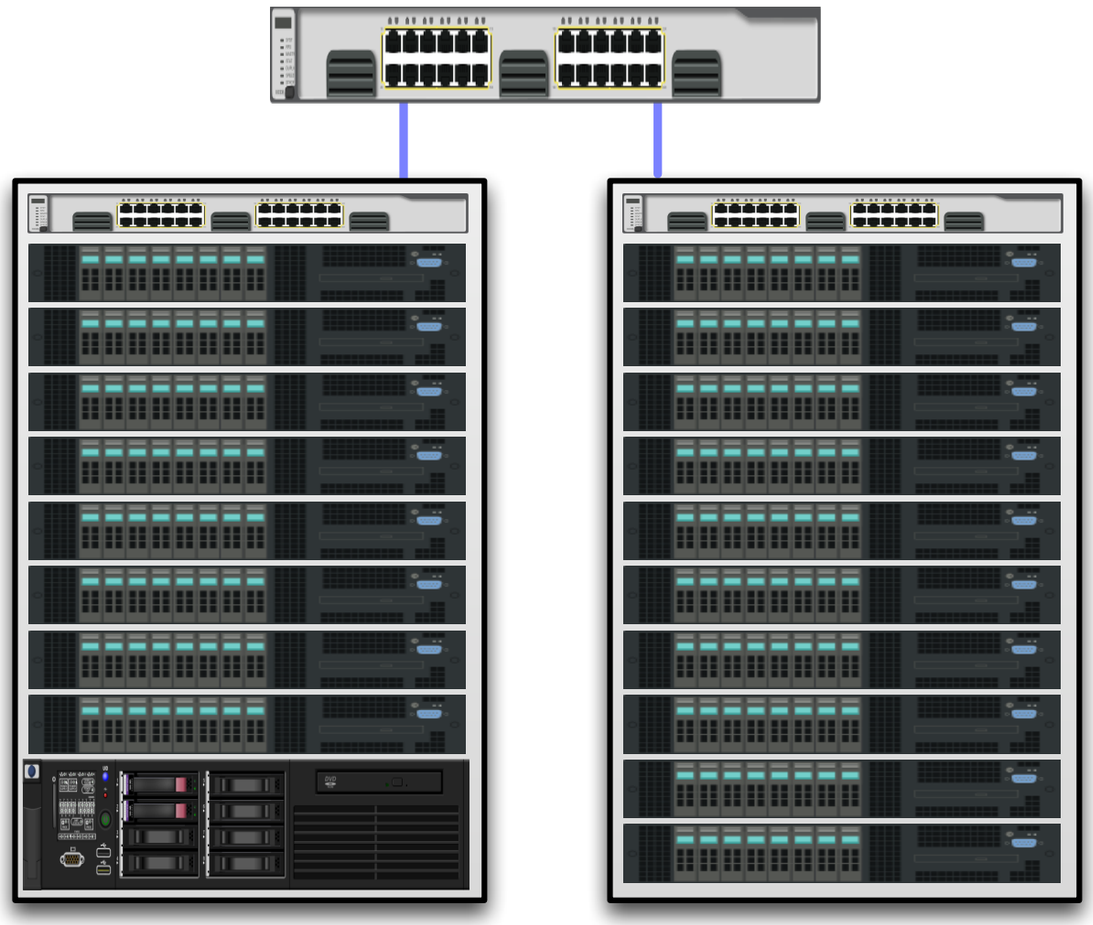
- This is a hallmark of Hadoop's design
  - Don't bring the data to the computation
  - Bring the computation to the data
- Hadoop uses the same machines for storage and processing
  - Significantly reduces need to transfer data across network

Hadoop Cluster Overview

- A cluster is made up of nodes
  - A node is simply a (typically rackmount) server
  - There may be a few – or a few thousand – nodes
  - Most are *slave* nodes, but a few are *master* nodes
  - Every node is responsible for both storage and processing
- Nodes are connected together by network switches
- Nearly all production clusters run Linux

Anatomy of a Small Cluster

- Two server racks containing nodes
- One master, many slaves
- “Top of Rack” network switch connects all the nodes within a rack
- Core network switch used to connect the racks together



More Hadoop Design Fundamentals

- Machine failure is unavoidable – embrace it
  - Build reliability into the system
- “More” is usually better than “faster”
  - Throughput matters more than latency

HDFS

Hadoop's Distributed Filesystem

HDFS: Hadoop Distributed File System

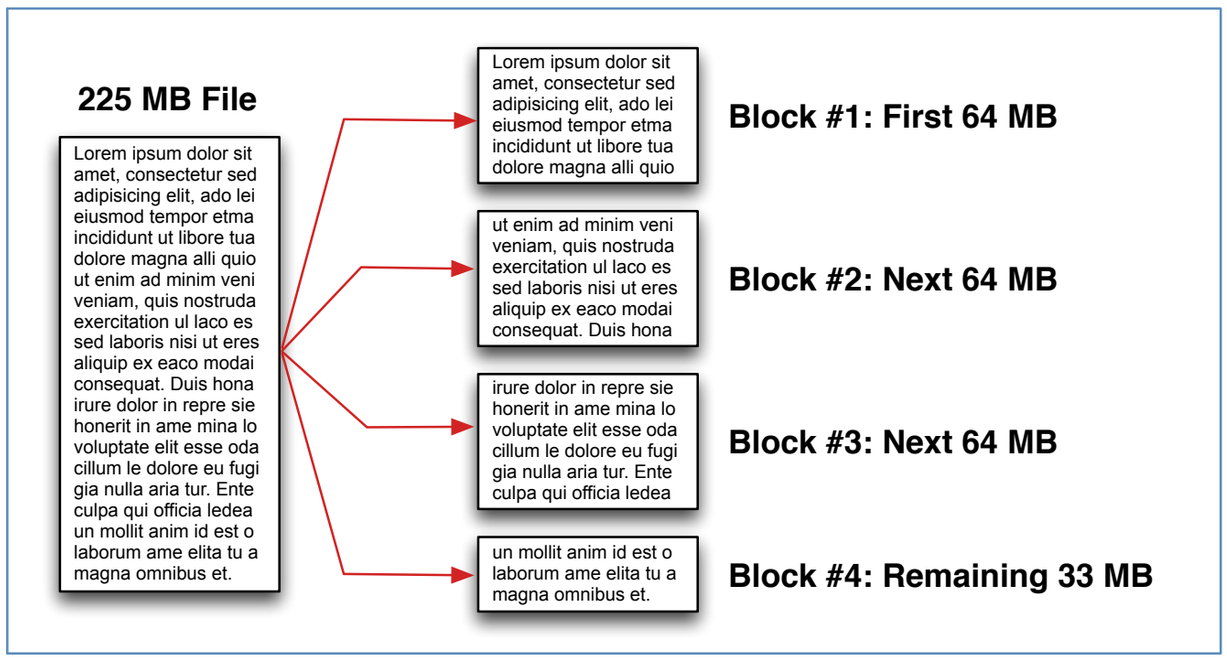
- Inspired by the Google File System
  - Reliable, low-cost storage for massive amounts of data
- Similar to a UNIX filesystem in some ways
  - Hierarchical
  - UNIX-style paths (e.g., `/sales/tom.txt`)
  - UNIX-style file ownership and permissions

HDFS: Hadoop Distributed File System

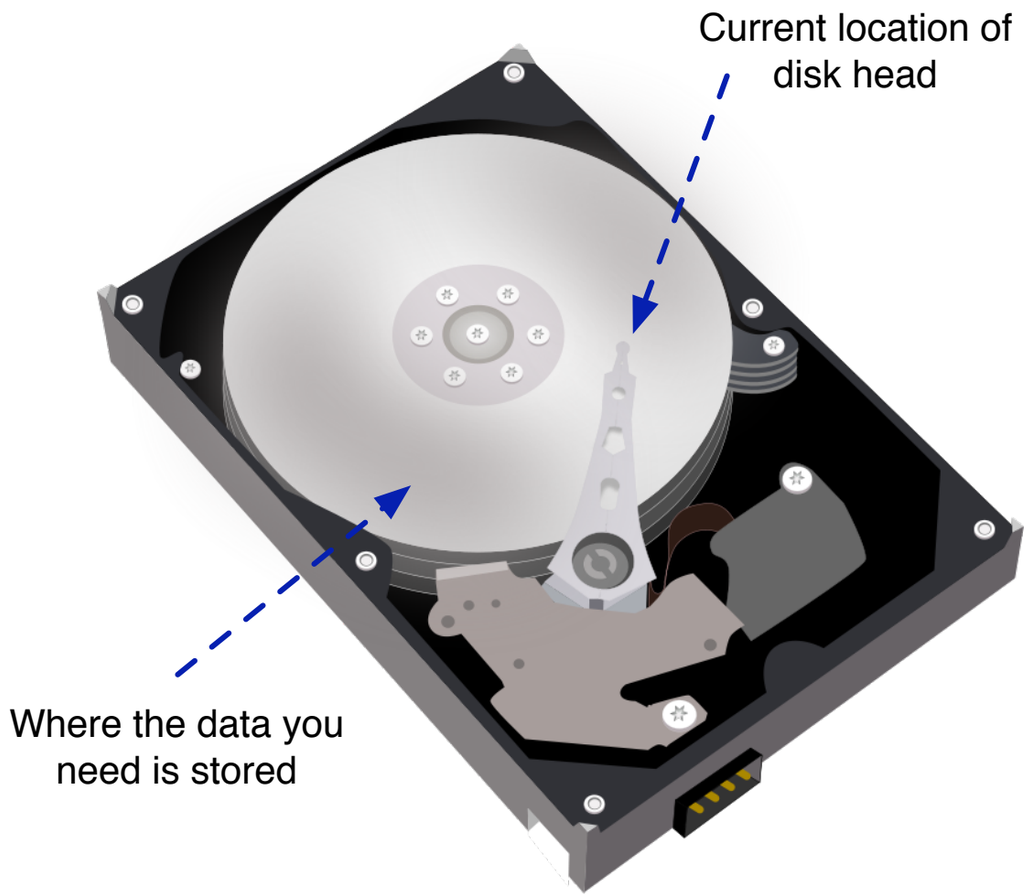
- There are also some major deviations from UNIX filesystems
  - Highly-optimized for processing data with MapReduce
    - Designed for sequential access to large files
    - Cannot modify file content once written
  - It's actually a user-space Java process
    - Accessed using special commands or APIs

HDFS Blocks

- Files added to HDFS are split into fixed-size blocks
  - Block size is configurable, but defaults to 64 megabytes



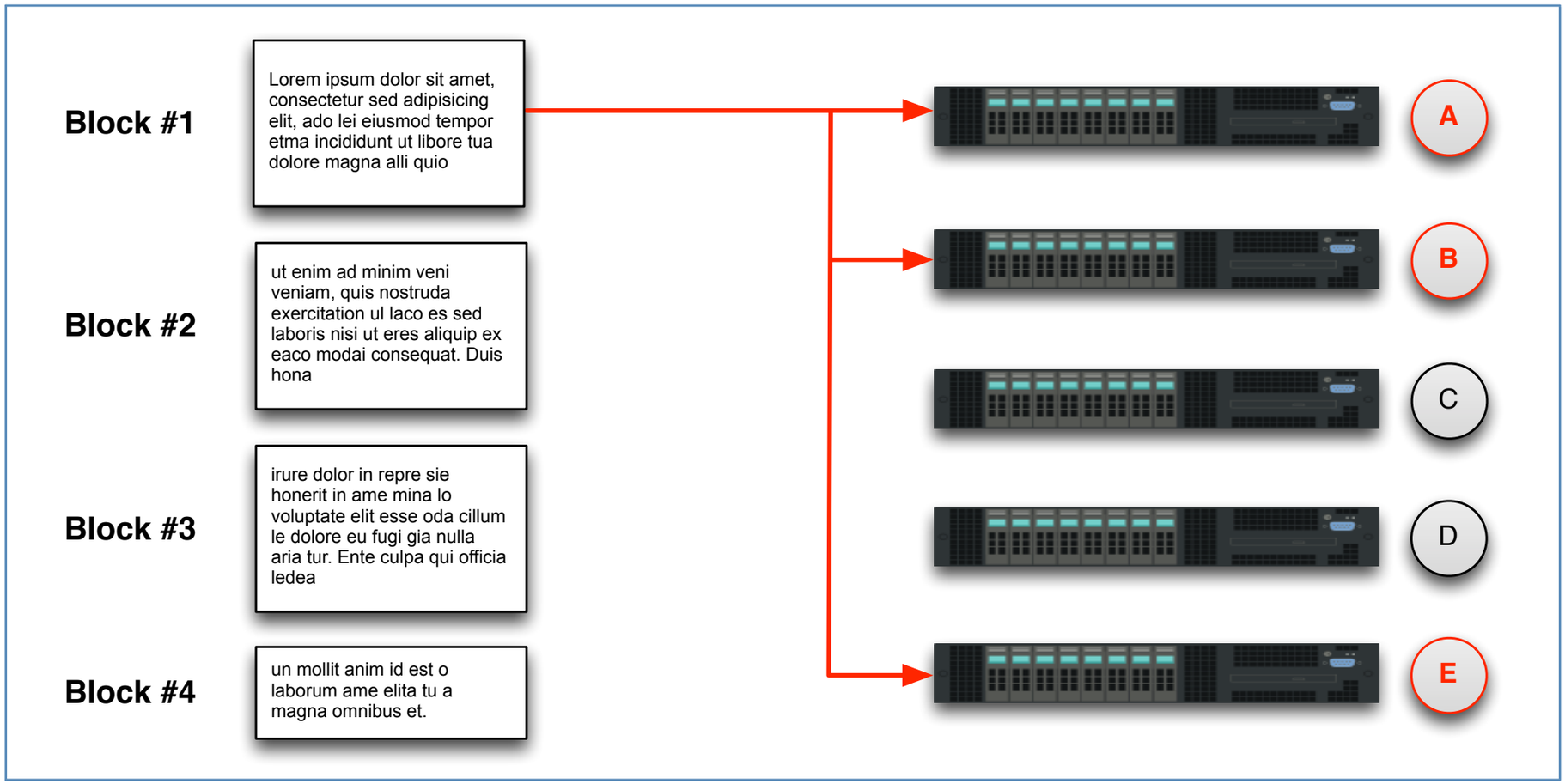
Why Does HDFS Use Such Large Blocks?



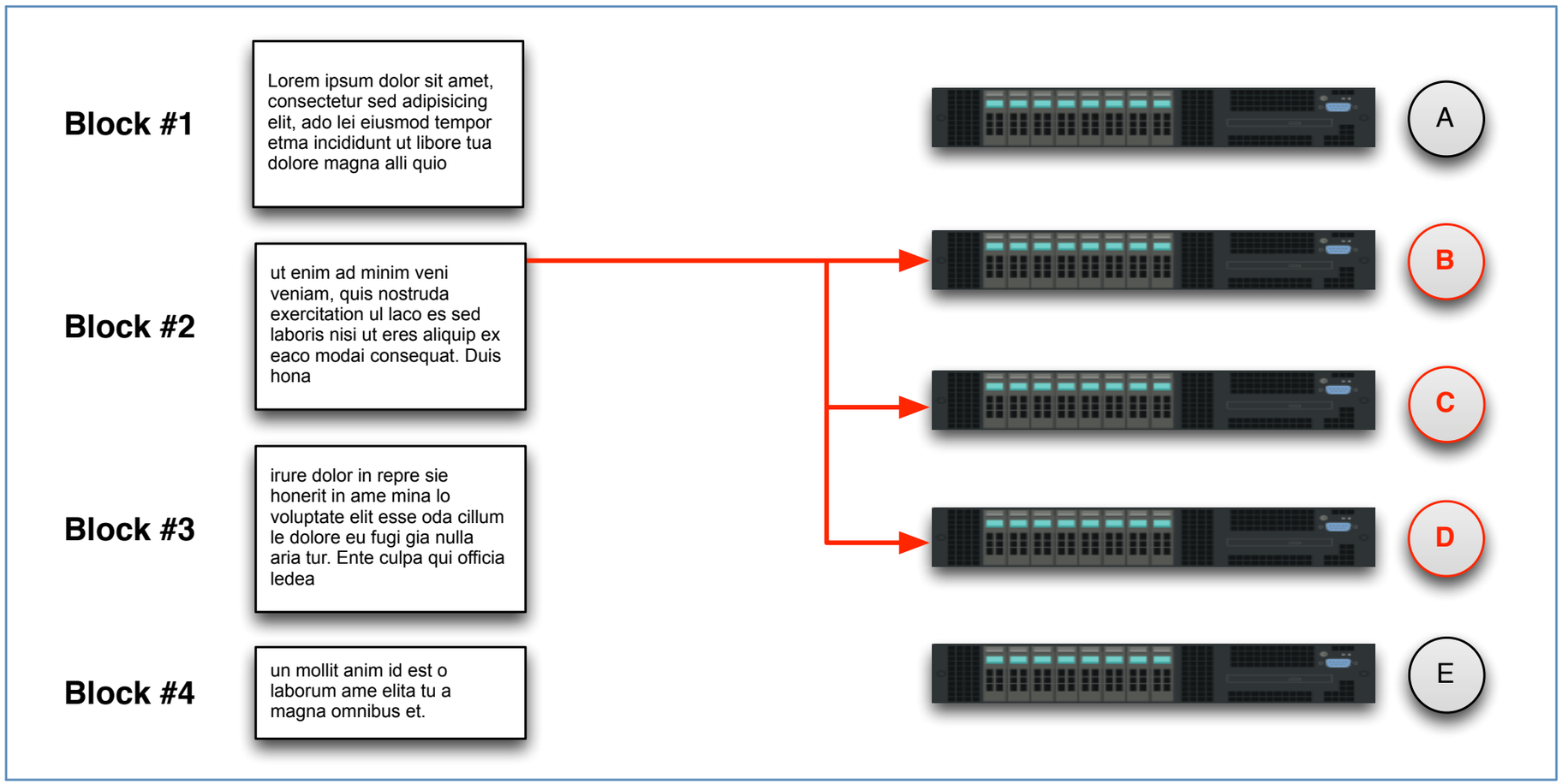
HDFS Replication

- Each block is then replicated across multiple nodes
  - Replication factor is also configurable, but defaults to three
- Benefits of replication
  - Availability: data isn't lost when a node fails
  - Reliability: HDFS compares replicas and fixes data corruption
  - Performance: allows for data locality
- Let's see an example of replication…

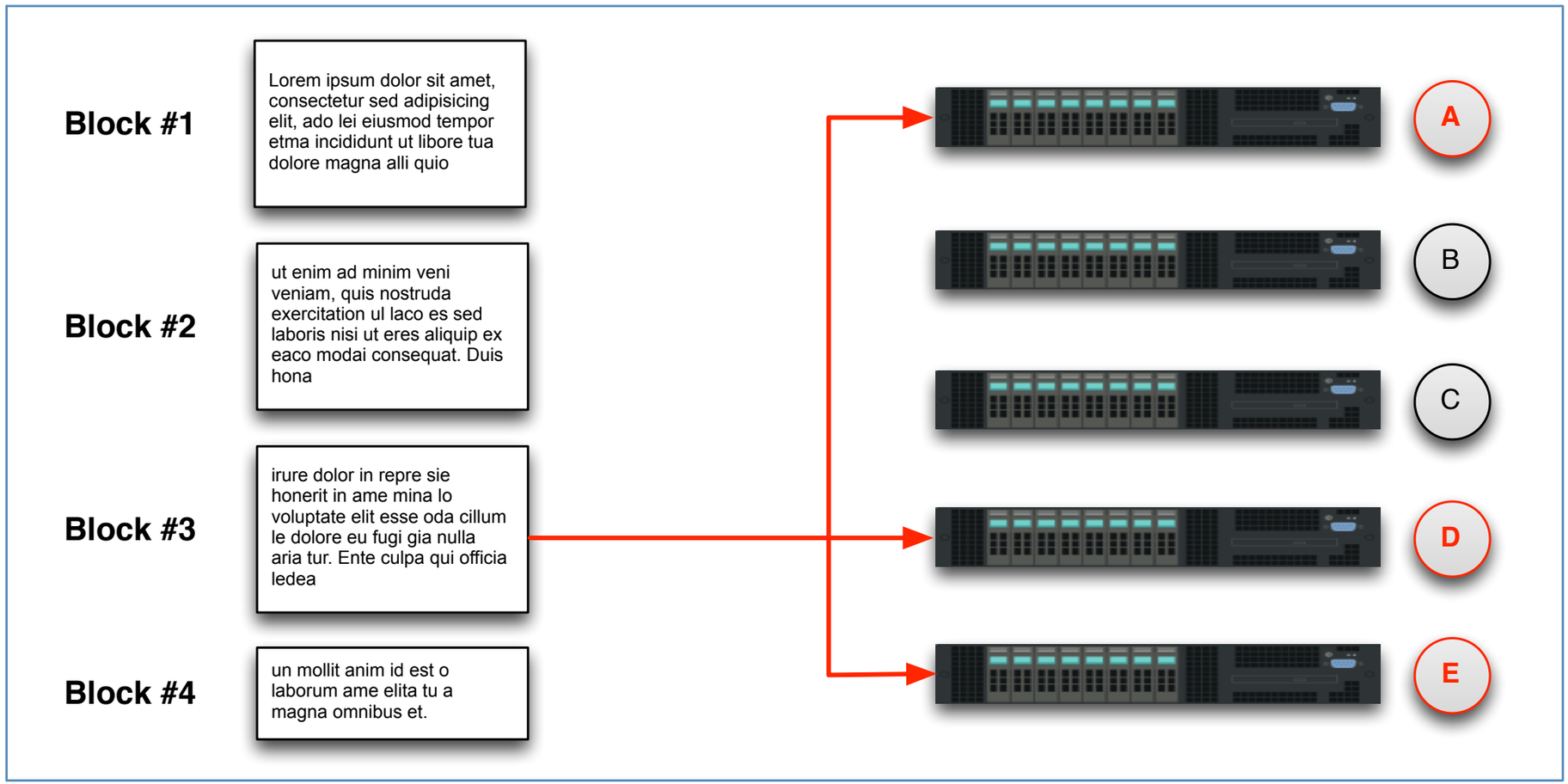
HDFS Replication (cont'd)



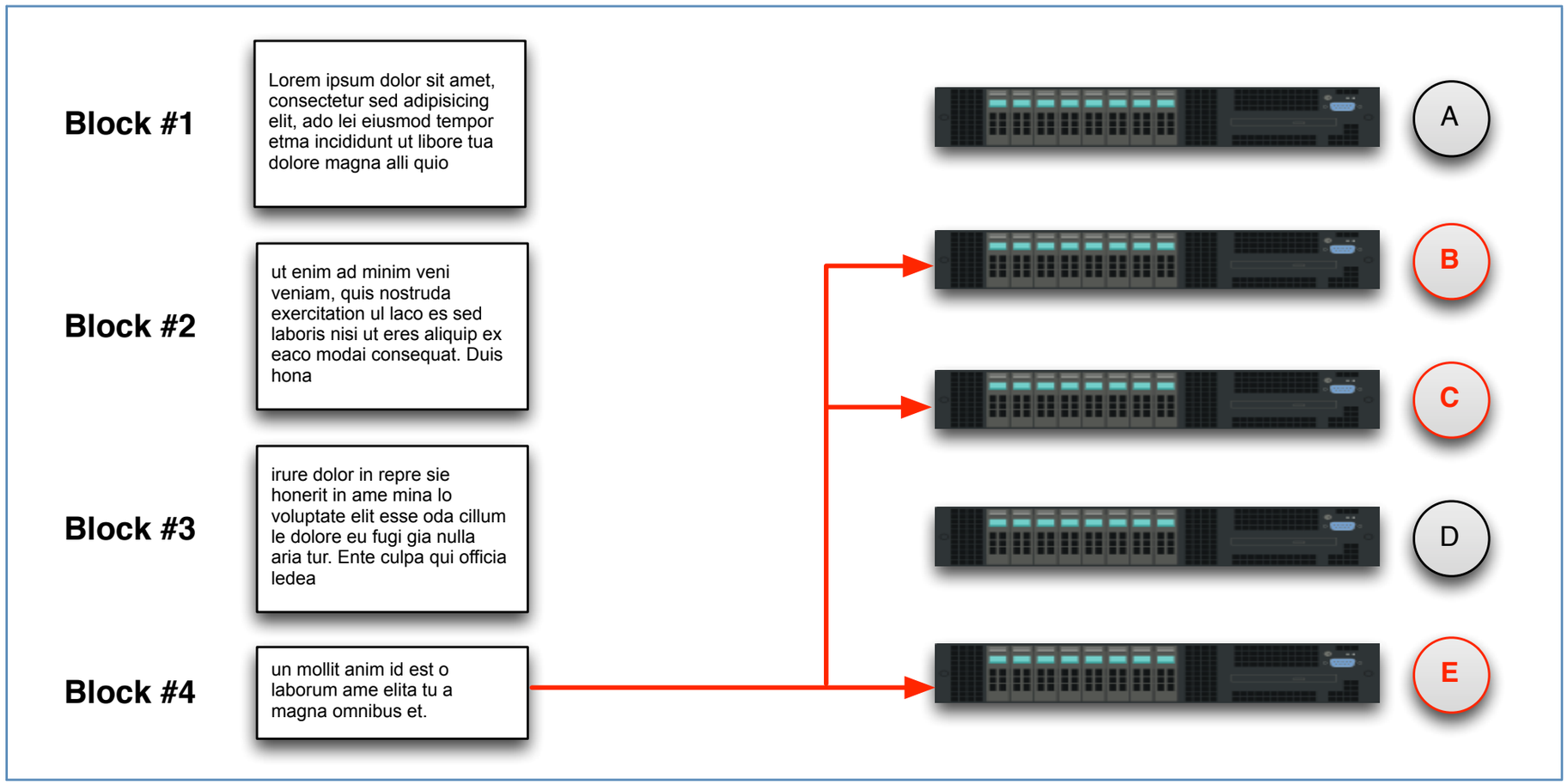
HDFS Replication (cont'd)



HDFS Replication (cont'd)

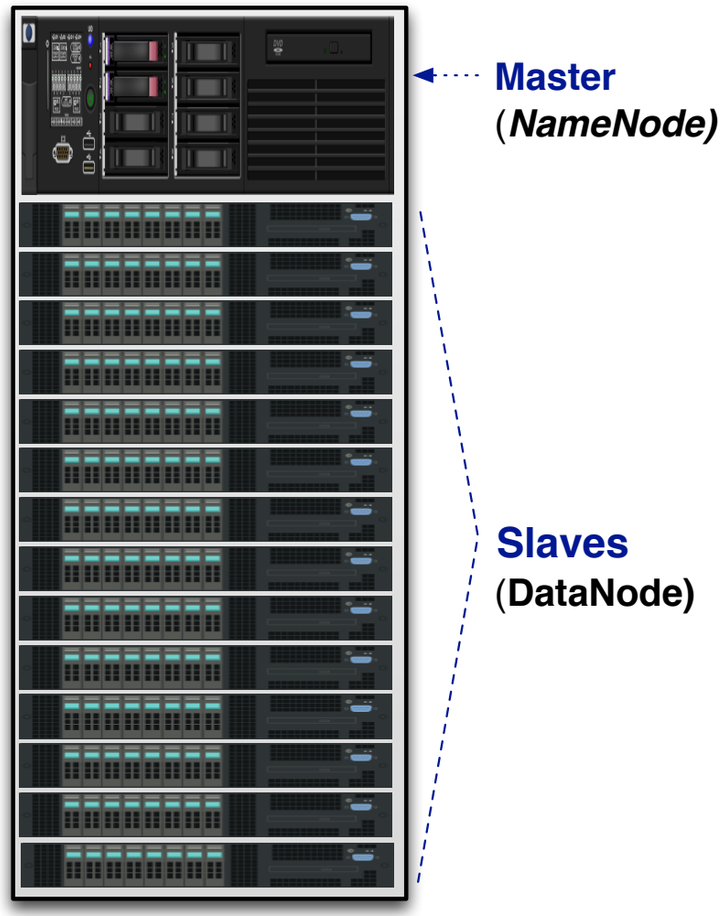


HDFS Replication (cont'd)



HDFS Architecture

- Hadoop has a master/slave architecture
- HDFS master daemon: NameNode
  - Manages namespace and metadata
  - Monitors slave nodes
- HDFS slave daemon: DataNode
  - Reads and writes actual data blocks



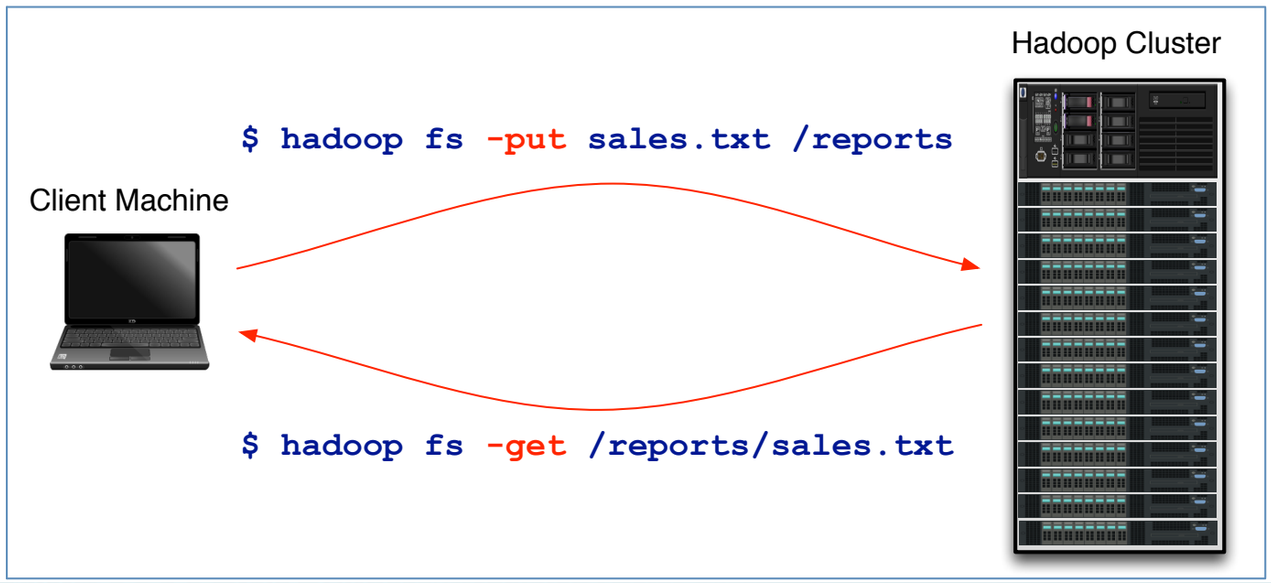
Accessing HDFS via the Command Line

- Users typically access HDFS via the `hadoop fs` command
  - Actions specified with subcommands (prefixed with a minus sign)
  - Most are similar to corresponding UNIX commands

```
$ hadoop fs -ls /user/tomwheeler

$ hadoop fs -cat /customers.csv

$ hadoop fs -rm /webdata/access.log

$ hadoop fs -mkdir /reports/marketing
```

Copying Local Data To and From HDFS

- Remember that HDFS is distinct from your local filesystem
  - `hadoop fs -put` copies local files ***to*** HDFS
  - `hadoop fs -get` fetches a local copy of a file ***from*** HDFS



HDFS Demo

- I will now demonstrate the following
  1. How to create a directory in HDFS
  2. How to copy a local file to HDFS
  3. How to display the contents of a file in HDFS
  4. How to remove a file from HDFS

MapReduce

A Scalable Data Processing Framework

MapReduce Introduction

- MapReduce is a programming model
  - It's a way of processing data
  - You can implement MapReduce in any language
- MapReduce has its roots in functional programming
  - Many languages have functions named `map` and `reduce`
  - These functions have largely the same purpose in Hadoop
- Popularized for large-scale processing by Google
- MapReduce processing in Hadoop is batch-oriented

Understanding Map and Reduce

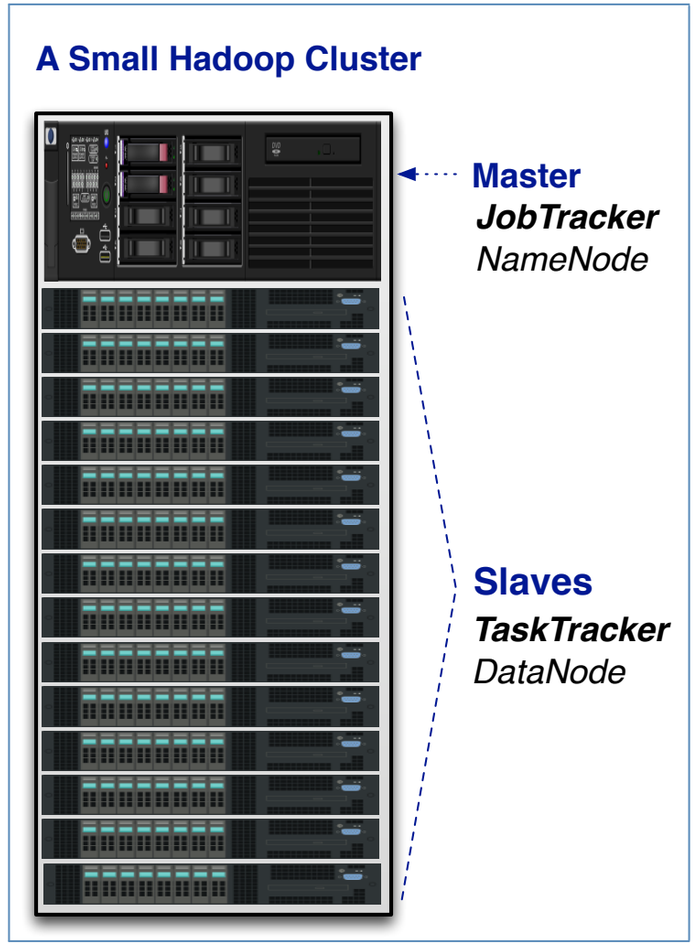
- MapReduce consists of two functions: Map and Reduce
  - The output from Map becomes the input to Reduce
  - Hadoop automatically sorts and groups data in between these
- The Map function always runs first
  - Typically used to filter, transform, or parse data
- The Reduce function is optional
  - Normally used to summarize data from the Map function
- Each piece is simple, but can be powerful when combined

MapReduce Benefits

- Scalability
  - Hadoop divides the processing job into individual tasks
  - Tasks execute in parallel (independently) across cluster
- Simplicity
  - Processes one record at a time
- Ease of use
  - Hadoop provides job scheduling and other infrastructure
  - Don’t have to write any file or network I/O code

MapReduce Architecture

- MapReduce master daemon: JobTracker
  - Accepts jobs from clients
  - Divides jobs into individual tasks
  - Assigns/monitors tasks on slave nodes
- MapReduce slave daemon: TaskTracker
  - Does the actual processing work
  - Reports status back to JobTracker
  - Collocated with the DataNode daemon



MapReduce Code for Hadoop

- Usually written in Java
  - This uses Hadoop's API directly
  - Data is passed as parameters to Map and Reduce methods
  - Output is emitted via Java method calls
- You can do basic MapReduce in other languages
  - Using the *Hadoop Streaming* wrapper program
  - Map and Reduce functions use STDIN / STDOUT for data
  - Some advanced features require Java code

MapReduce Example in Python

- The following example uses Python
  - Via Hadoop Streaming
- It processes log files and summarizes events by type
  - I'll explain both the data flow and the code

Job Input

- Here's the job input

```
2013-06-29 22:16:49.391 CDT INFO "This can wait"
2013-06-29 22:16:52.143 CDT INFO "Blah blah blah"
2013-06-29 22:16:54.276 CDT WARN "This seems bad"
2013-06-29 22:16:57.471 CDT INFO "More blather"
2013-06-29 22:17:01.290 CDT WARN "Not looking good"
2013-06-29 22:17:03.812 CDT INFO "Fairly unimportant"
2013-06-29 22:17:05.362 CDT ERROR "Out of memory!"
```

- Each map task gets a chunk of this data to process
  - This "chunk" is called an **InputSplit**

Python Code for Map Function

```
1  #!/usr/bin/env python
2
3  import sys
4
5  levels = ['TRACE', 'DEBUG', 'INFO',
6            'WARN', 'ERROR', 'FATAL']
7
8  for line in sys.stdin:
9      fields = line.split()
10     for field in fields:
11         field = field.strip().upper()
12         if field in levels:
13             print "%s\t1" % field
```

Define list of known log events

Split every line (record) we receive on standard input into fields, normalized by case

If this field matches a log level, print it, a tab separator, and the literal value 1

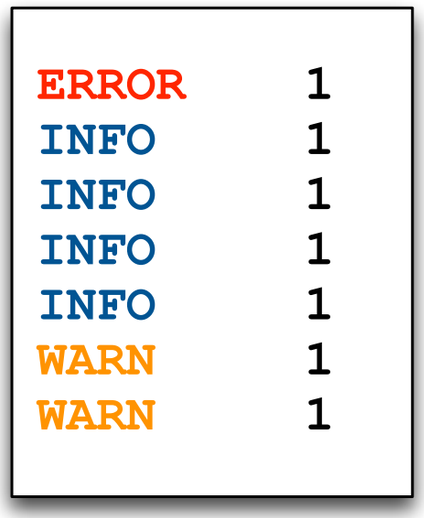
Output of Map Function

- The map function produces key/value pairs as output

```
INFO    1
INFO    1
WARN    1
INFO    1
WARN    1
INFO    1
ERROR   1
```

Input to Reduce Function

- The Reducer receives a key and all values for that key



  - Keys are always passed to reducers in sorted order
  - Although not obvious here, values are unordered

Python Code for Reduce Function

```
1  #!/usr/bin/env python
2
3  import sys
4
5  previous_key = ''
6  sum = 0
7
8  for line in sys.stdin:
9      key, value = line.split()
10
11     value = int(value)
12     # continued on next slide
13
```

Initialize loop variables

Extract the key and value passed via standard input

Python Code for Reduce Function

```
14      # continued from previous slide
15      if key == previous_key:
16          sum = sum + value
17      else:
18          if previous_key != '':
19              print '%s\t%i' % (previous_key, sum)
20          previous_key = key
21          sum = 1
22
23  print '%s\t%i' % (previous_key, sum)
```

If key unchanged, increment the count

If key changed, print sum for previous key

Re-init loop variables

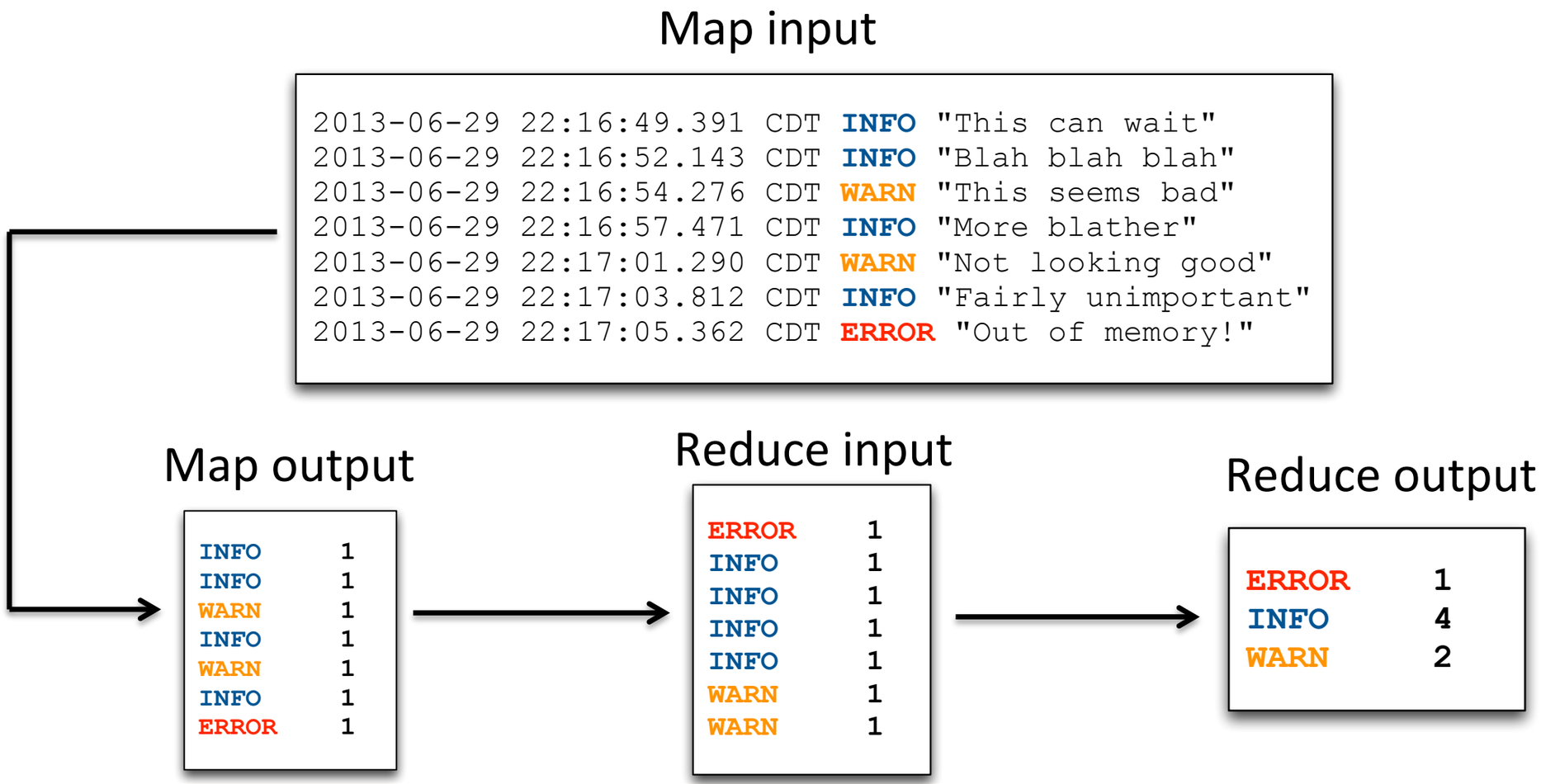
Print sum for final key

Output of Reduce Function

- Its output is a sum for each level

```
ERROR    1
INFO     4
WARN     2
```

Recap of Data Flow



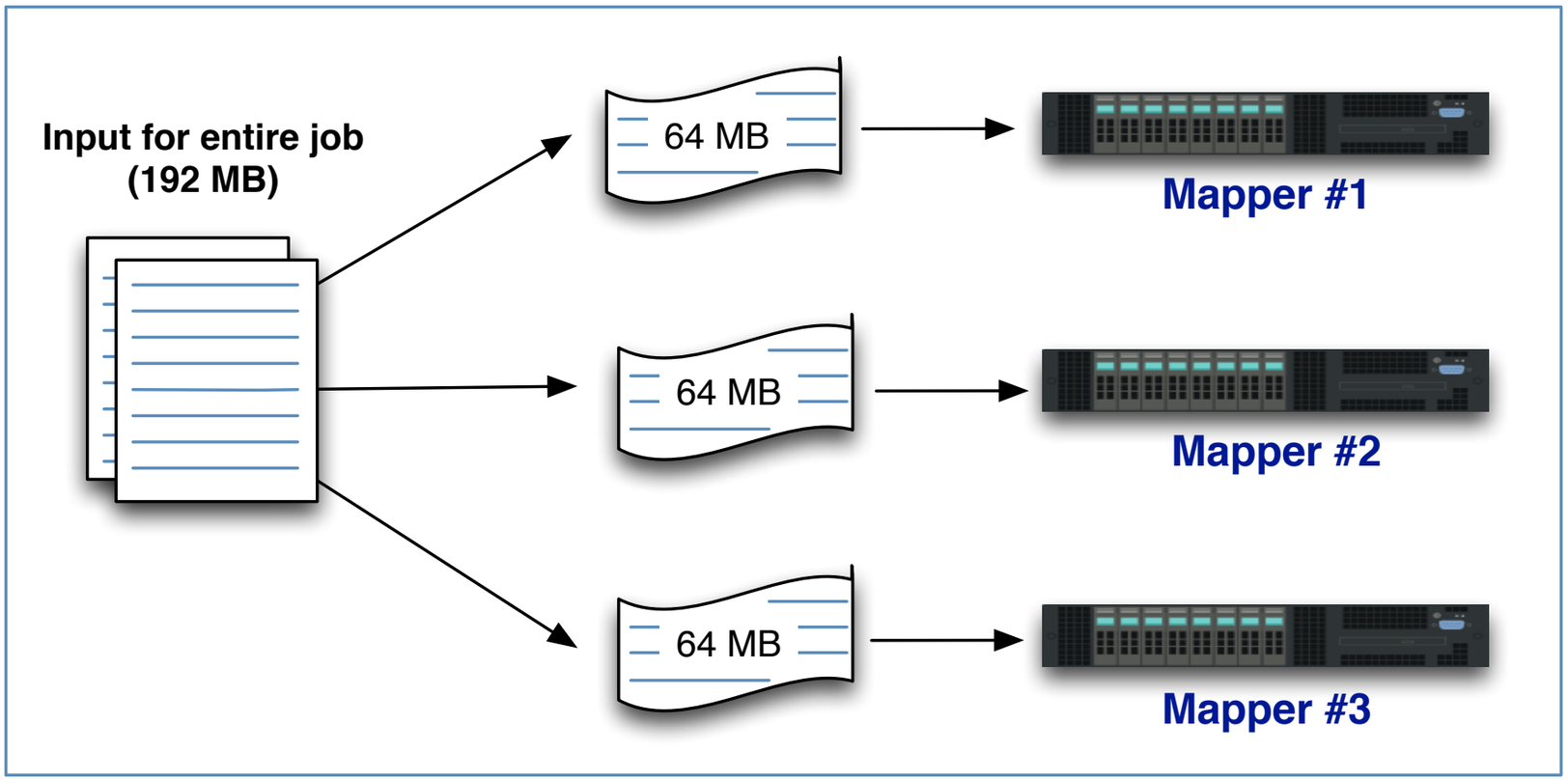
How to Run a Hadoop Streaming Job

- I'll demonstrate this now…

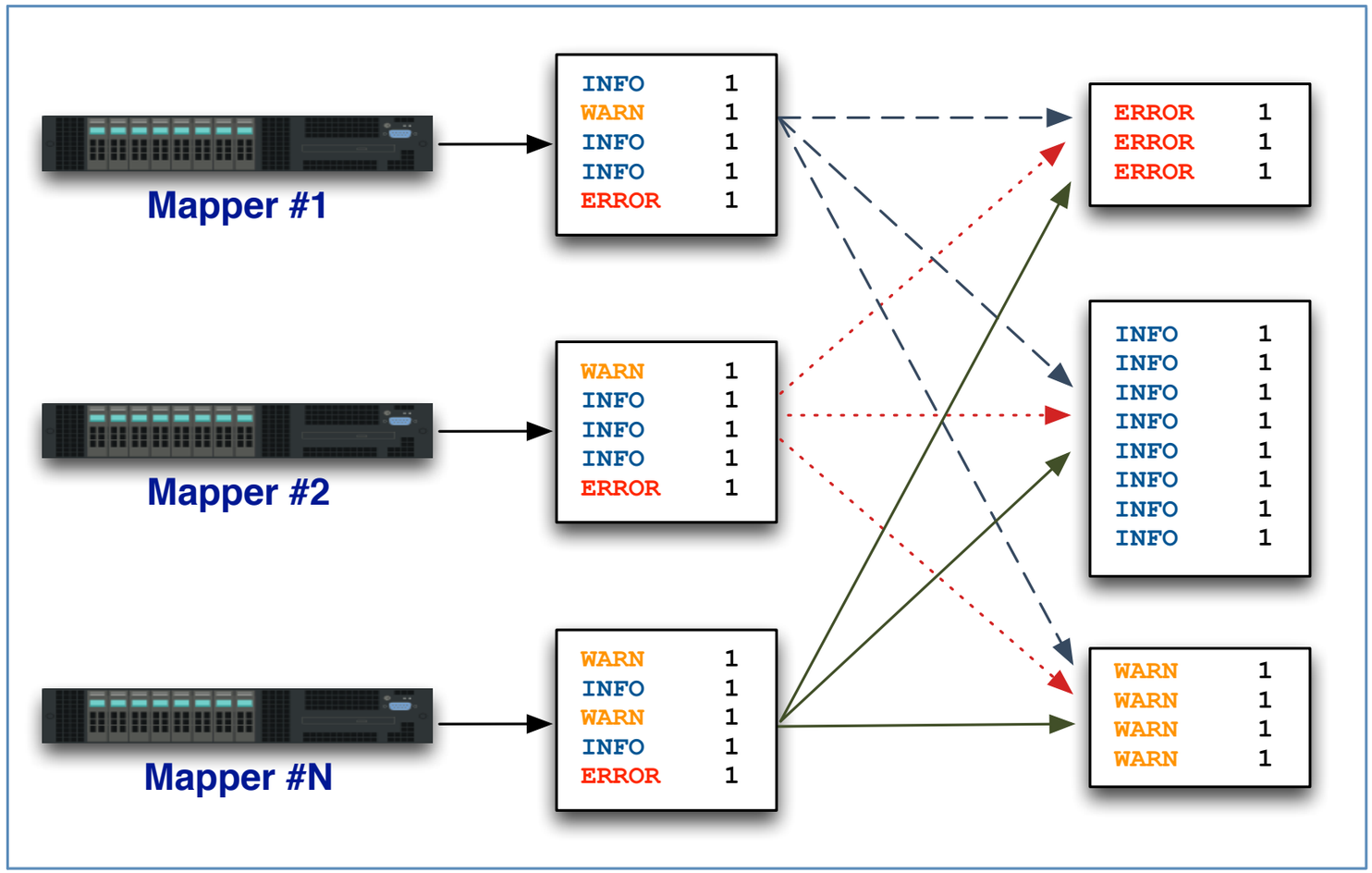
Visual Overview of Job Execution

- I'll now show you what's happening on the cluster itself
- My overview is based on the same code
  - But assumes more input data, to illustrate a few points

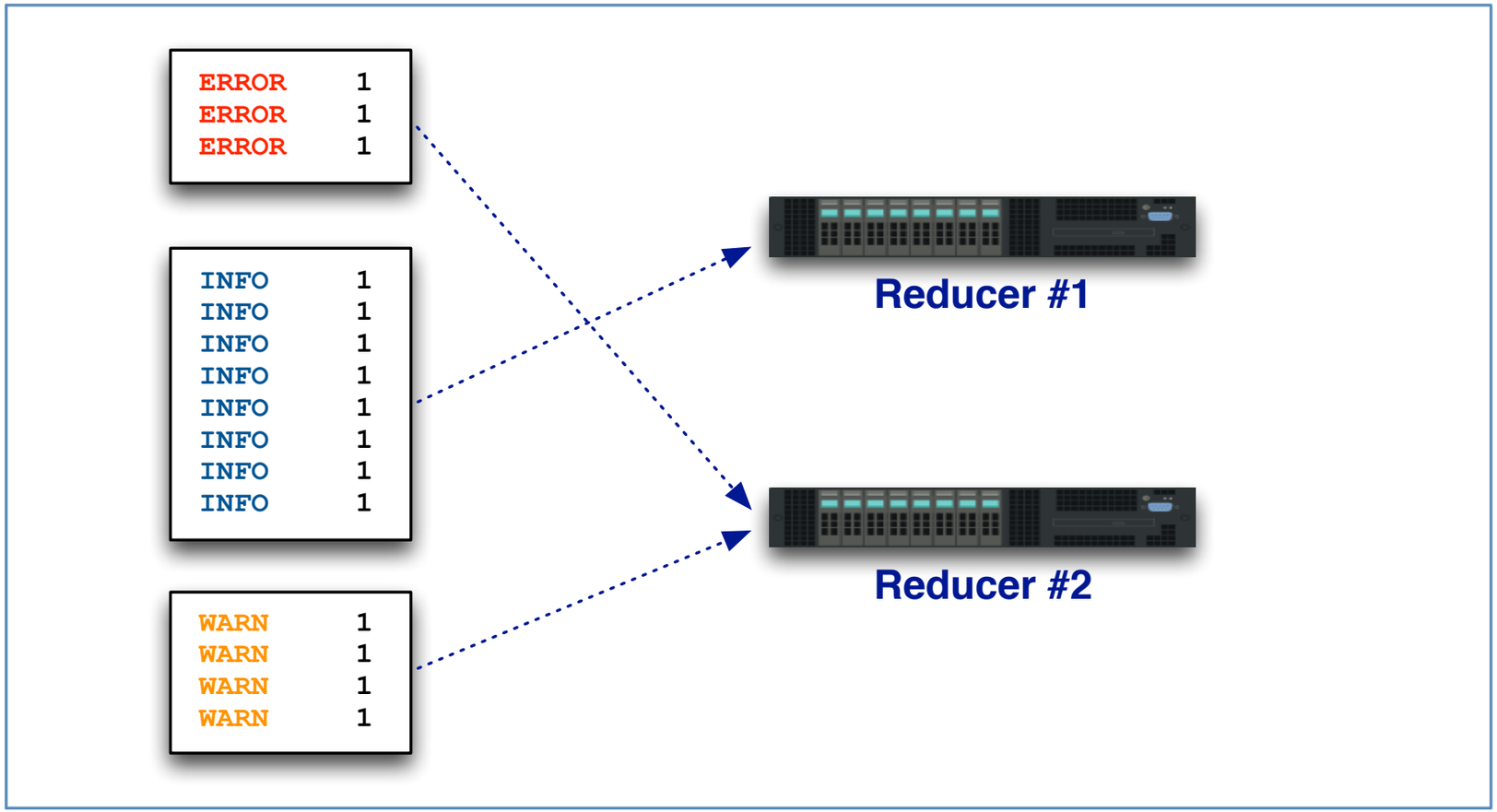
Job Execution: Input Fed to Map Tasks



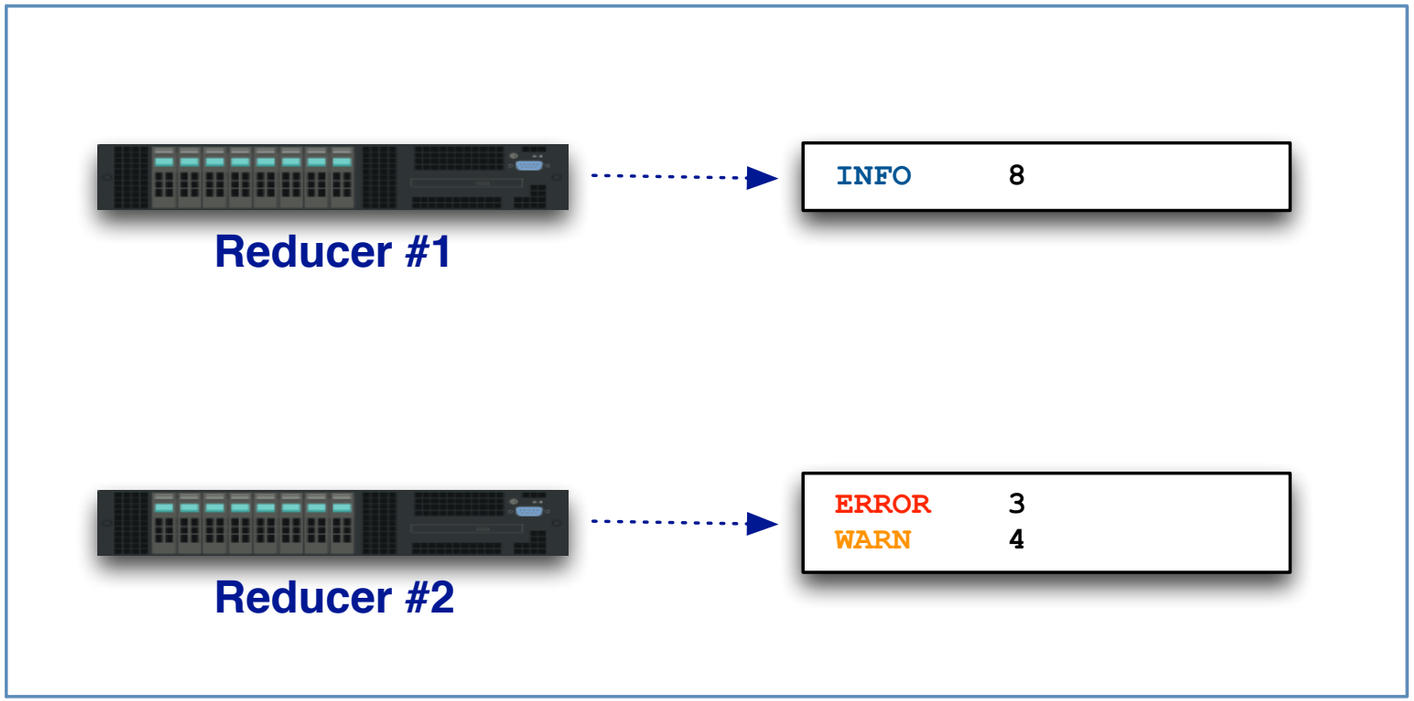
Job Execution: Shuffle and Sort



Job Execution: Reducer Input



Job Execution: Final Output



Hadoop's Java API

- MapReduce code is typically written in Java
  - Better performance
  - More opportunity for customization
- Concept is pretty much the same, with three minor differences
  - Input and output data is *typed*
  - Values are passed to Reducer using an Iterator
  - Job is configured and executed using a *driver* class

Java MR Job Example: Mapper

```
1  package com.cloudera.example;
2
3  import java.io.IOException;
4  import org.apache.hadoop.io.IntWritable;
5  import org.apache.hadoop.io.LongWritable;
6  import org.apache.hadoop.io.Text;
7  import org.apache.hadoop.mapreduce.Mapper;
8
9  public class LogEventParseMapper extends Mapper<LongWritable, Text,
10    Text, IntWritable> {
```

Input key and value types

Output key and value types

Java MR Job Example: Mapper

```
11      enum Level { TRACE, DEBUG, INFO, WARN, ERROR, FATAL };
12
13      /*
14       * The map method is invoked once for each line of text in the
15       * input data.  The method receives a key of type LongWritable
16       * (which corresponds to the byte offset in the current input
17       * file), a value of type Text (representing the line of input
18       * data), and a Context object (which allows us to print status
19       * messages, among other things).
20       */
21       @Override
22       public void map(LongWritable key, Text value, Context context)
23         throws IOException, InterruptedException {
```

Java MR Job Example: Mapper

```
24          String line = value.toString();
25
26          // ignore empty lines
27          if (line.trim().isEmpty()) {
28              return;
29          }
30
31          String[] fields = line.split(" ");
32
33          // ensure this line is not malformed
34          if (fields.length <= 3) {
35              return;
36          }
```

Convert value to a Java String

Defensive programming!

Split record into fields

Even more defensive programming!

Java MR Job Example: Mapper

```
37            String levelField = fields[3];
38
39            for (Level level : Level.values()) {
40                String levelName = level.name();
41
42                if (levelName.equalsIgnoreCase(levelField)) {
43                   context.write(new Text(levelName), new IntWritable(1));
44                }
45            }
46        }
47    }
```

Extract based on position

Check whether that value matches a level we defined

Output key and value

Java MR Job Example: Reducer

```
1  package com.cloudera.example;
2
3  import java.io.IOException;
4
5  import org.apache.hadoop.io.IntWritable;
6  import org.apache.hadoop.io.Text;
7  import org.apache.hadoop.mapreduce.Reducer;
8
9  public class LogEventSumReducer extends Reducer<Text, IntWritable,
10 Text, IntWritable> {
```

Input key and value types

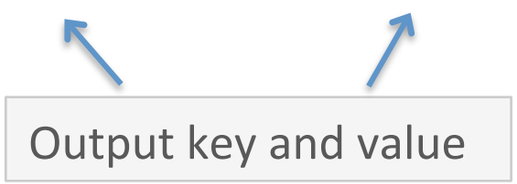
Output key and value types

Java MR Job Example: Reducer

```
11    /*
12     * The reduce method is invoked once for each key received from
13     * the shuffle and sort phase of the MapReduce framework.
14     * The method receives a key of type Text (representing the key),
15     * a set of values of type IntWritable, and a Context object.
16     */
17    @Override
18    public void reduce(Text key, Iterable<IntWritable> values,
19      Context context) throws IOException, InterruptedException {
```

Java MR Job Example: Reducer

```
20            // used to count the number of messages for this event type
21            int sum = 0;
22
23            // increment it for each new value received
24            for (IntWritable value : values) {
25                sum += value.get();
26            }
27
28            // Our output is the event type (key) and the sum (value)
29            context.write(key, new IntWritable(sum));
30        }
31    }
```



Java MR Job Example: Driver

```
1  package com.cloudera.example;
2
3  import org.apache.hadoop.fs.Path;
4  import org.apache.hadoop.io.IntWritable;
5  import org.apache.hadoop.io.Text;
6  import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
7  import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
8  import org.apache.hadoop.mapreduce.Job;
9
10 // The driver is just a regular Java class with a "main" method
11 public class Driver {
12
13     public static void main(String[] args) throws Exception {
```

Java MR Job Example: Driver

```
14        // validate commandline arguments (we require the user
15        // to specify the HDFS paths to use for the job; see below)
16        if (args.length != 2) {
17           System.out.printf("Usage: Driver <input dir> <output dir>\n");
18           System.exit(-1);
19        }
20
21        // Instantiate a Job object for our job's configuration.
22        Job job = new Job();
23
24        // configure input and output paths based on supplied arguments
25        FileInputFormat.setInputPaths(job, new Path(args[0]));
26        FileOutputFormat.setOutputPath(job, new Path(args[1]));
```

Java MR Job Example: Driver

```
27        // tells Hadoop to copy the JAR containing this class
28        // to cluster nodes, as required to run this job
29        job.setJarByClass(Driver.class);
30
31        // give the job a descriptive name. This is optional, but
32        // helps us identify this job on a busy cluster
33        job.setJobName("Log Event Counter Driver");
34
35        // Specify which classes to use for the Mapper and Reducer
36        job.setMapperClass(LogEventParseMapper.class);
37        job.setReducerClass(LogEventSumReducer.class);
```

Java MR Job Example: Driver

```
38            // specify the job's output key and value classes
39            job.setOutputKeyClass(Text.class);
40            job.setOutputValueClass(IntWritable.class);
41
42            // start the MapReduce job and wait for it to finish.
43            // if it finishes successfully, return 0; otherwise 1.
44            boolean success = job.waitForCompletion(true);
45            System.exit(success ? 0 : 1);
46        }
47    }
```

How to Run a Java MapReduce Job

- I'll demonstrate this now…

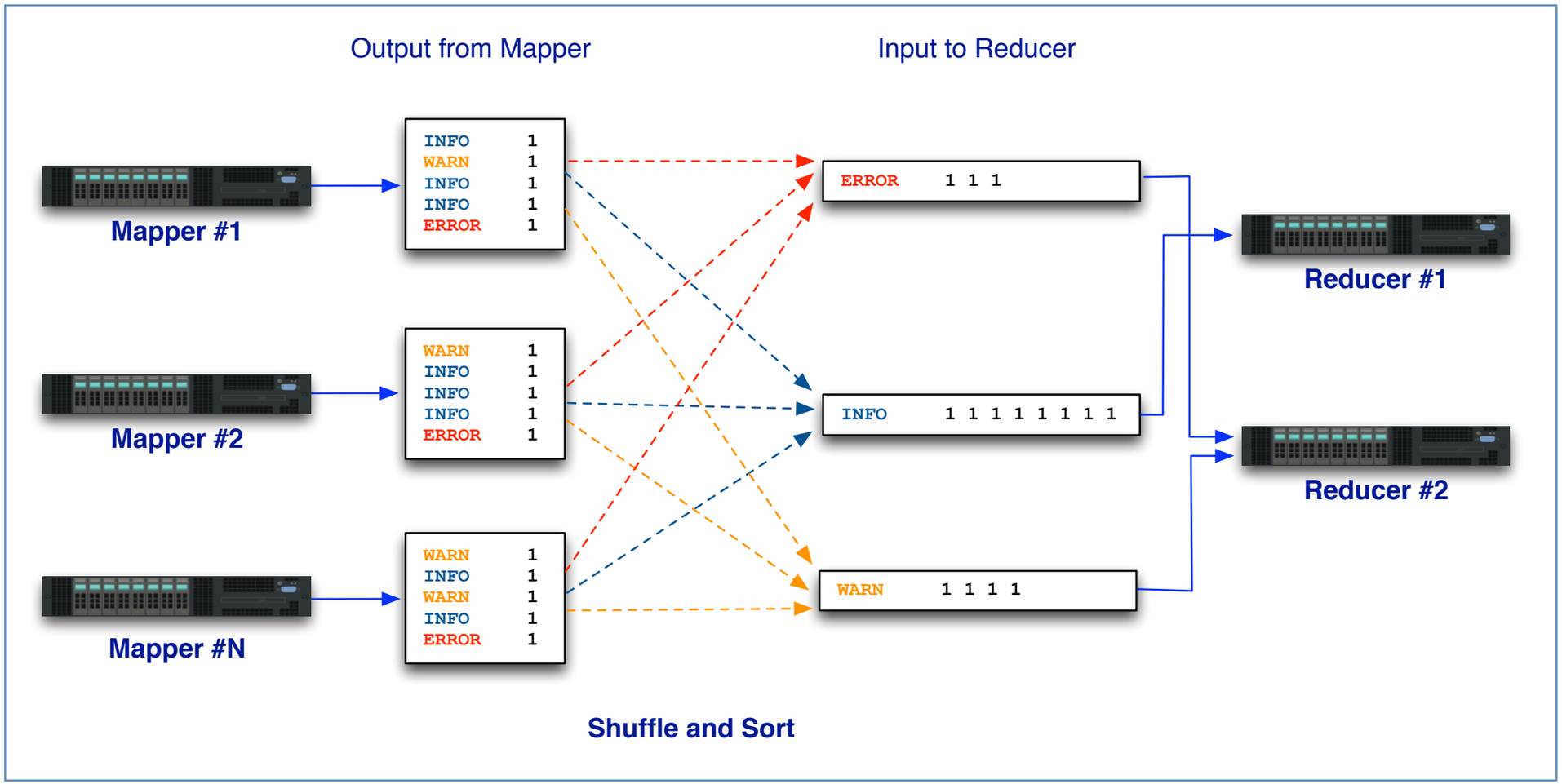
Using Apache Hadoop Effectively

Tips for the Developer

How Many Map and Reduce Tasks?

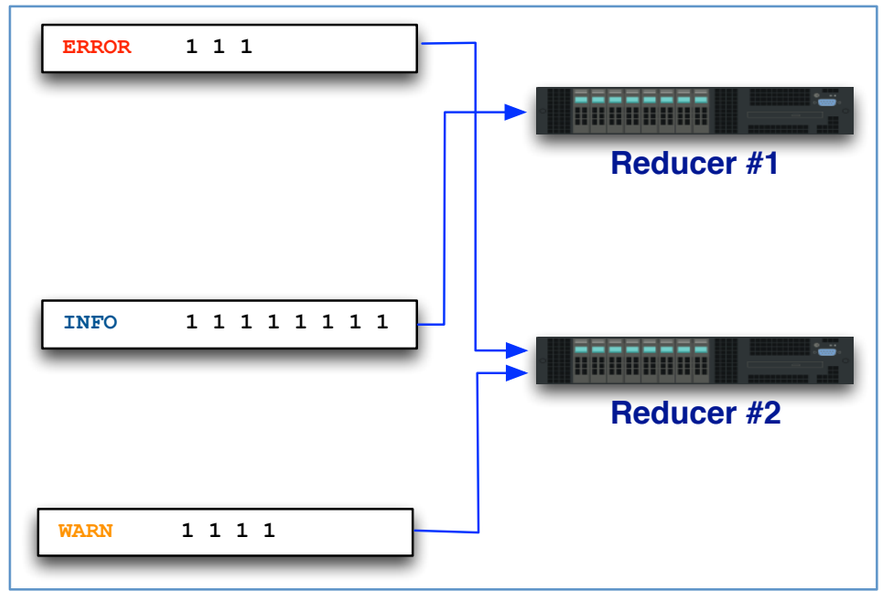
- Number of map tasks is determined by amount of input data
  - Typically one map task for each block in HDFS (e.g., 64 MB)
- Number of reduce tasks is determined by developer
  - A given key and all its values go to the *same reduce task*
- There is a single reduce task by default
  - This can be a bottleneck!

Physical View of Data Flow



How Reducer for a Key is Chosen

- Dividing the keyspace among reducers is called “partitioning”
- Default partitioner uses hashcode
  - Not appropriate for all data
  - Can cause uneven load
- You can write a custom partitioner
  - Using Hadoop’s Java API

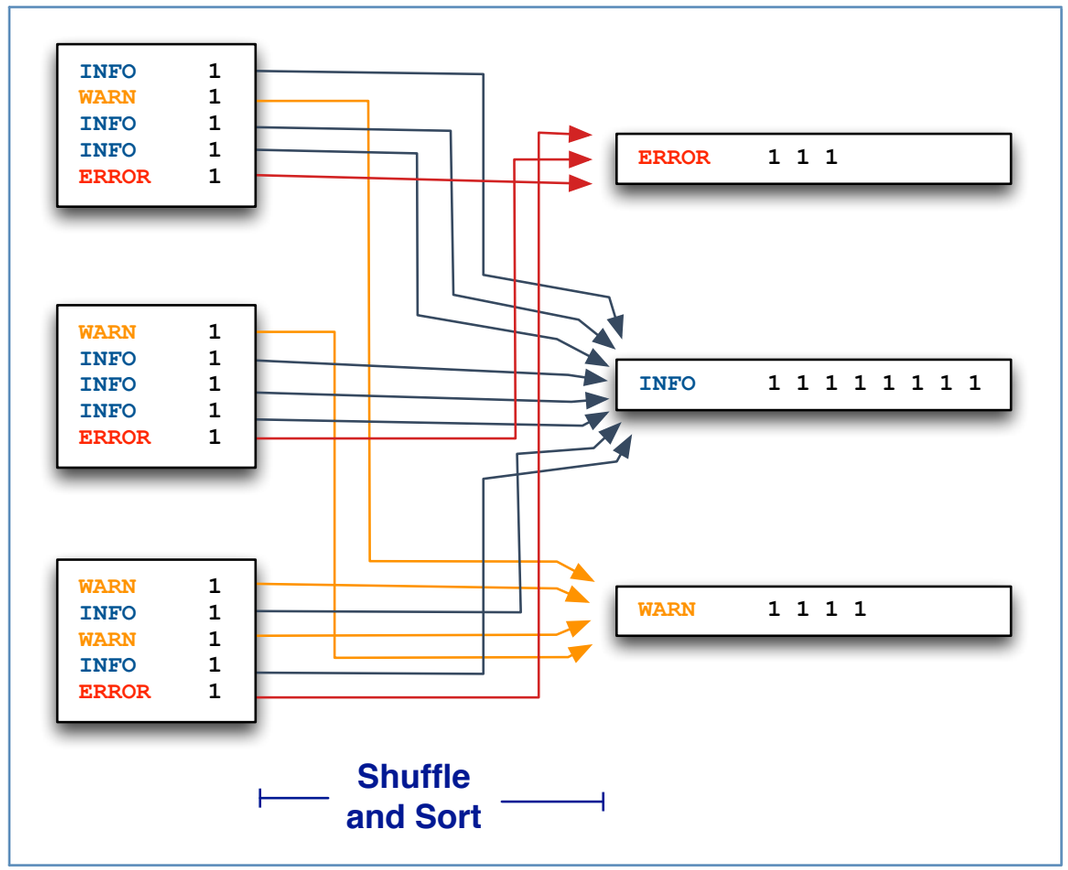


Java Custom Partitioner Example

```
1  package com.cloudera.example;
2
3  import org.apache.hadoop.mapreduce.Partitioner;
4
5  public class MyPartitioner extends Partitioner<Text, IntWritable> {
6
7      public int getPartition(Text key, IntWritable value,
8          int numReducers) {
9
10         // implement custom logic here
11         // return a value between 0 and (numReducers - 1)
12     }
13 }
```

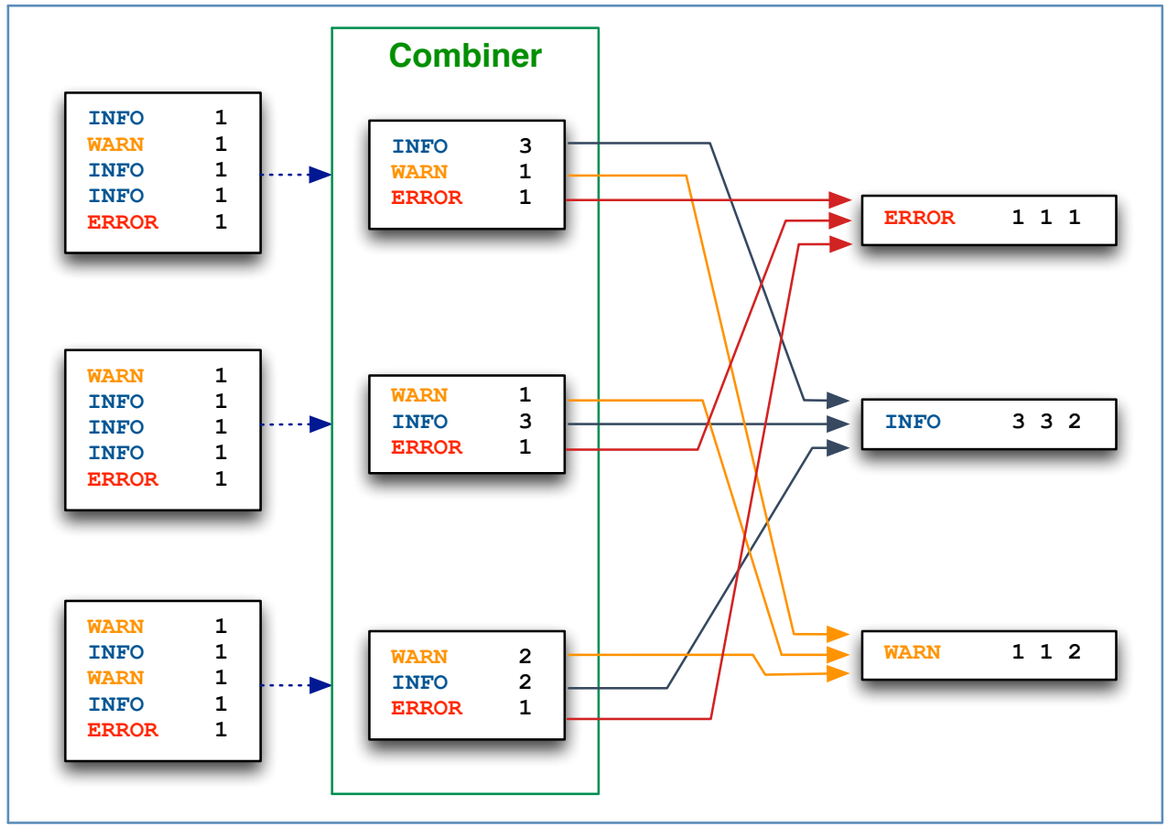
Hadoop Jobs are Often I/O Bound

- The “shuffle and sort” phase involves a lot of I/O
  - All values for a key must be copied from each node where the map task runs to the node where the relevant reduce task runs



Improving Performance with a Combiner

- A combiner is like a reducer that runs on a map task's output before the shuffle and sort phase
- Using one can decrease the amount of data that needs to be transferred



Using a Combiner

- A Combiner is implemented exactly like a reducer
  - Extend `org.apache.hadoop.mapreduce.Reducer`
- Can often reuse your Reducer class *as* the combiner
  - But only if the operation is associative and commutative
- Finally, specify your combiner class in the driver
  - Use the `job.setCombinerClass` method

Tips for Running at Scale

- Don’t assume input data is valid
- Test your code with a small data sample
  - Run it at scale once you’re sure that it works OK
- Write unit tests for your code
  - See MRUnit: http://mrunit.apache.org/
- Consider using a high-level tool like Pig, Hive, or Impala

When is Hadoop (Not) a Good Choice

- Hadoop may be a great choice when
  - You need to process non-relational (unstructured) data
  - You are processing large amounts of data
- Hadoop may not be a great choice when
  - You're processing small amounts of data
  - Your algorithms require communication among nodes
  - You need transactions or similar database features

The Hadoop Ecosystem

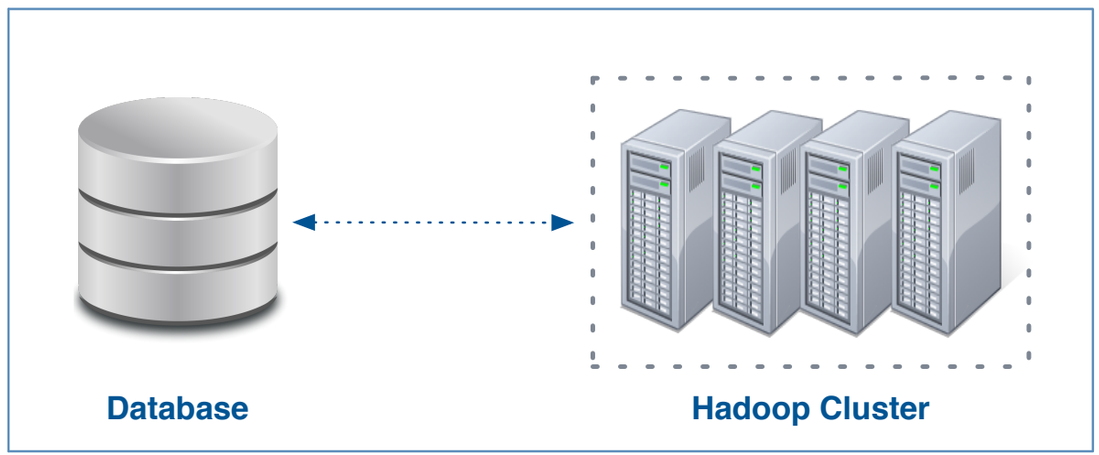
Open Source Tools that Complement Hadoop

The Hadoop Ecosystem

- Many related tools integrate with Hadoop
  - Data analysis
  - Database integration
  - Workflow management
  - Cluster management
- These are not considered “core Hadoop”
  - Rather, they’re part of the “Hadoop ecosystem”
  - Many are also open source Apache projects

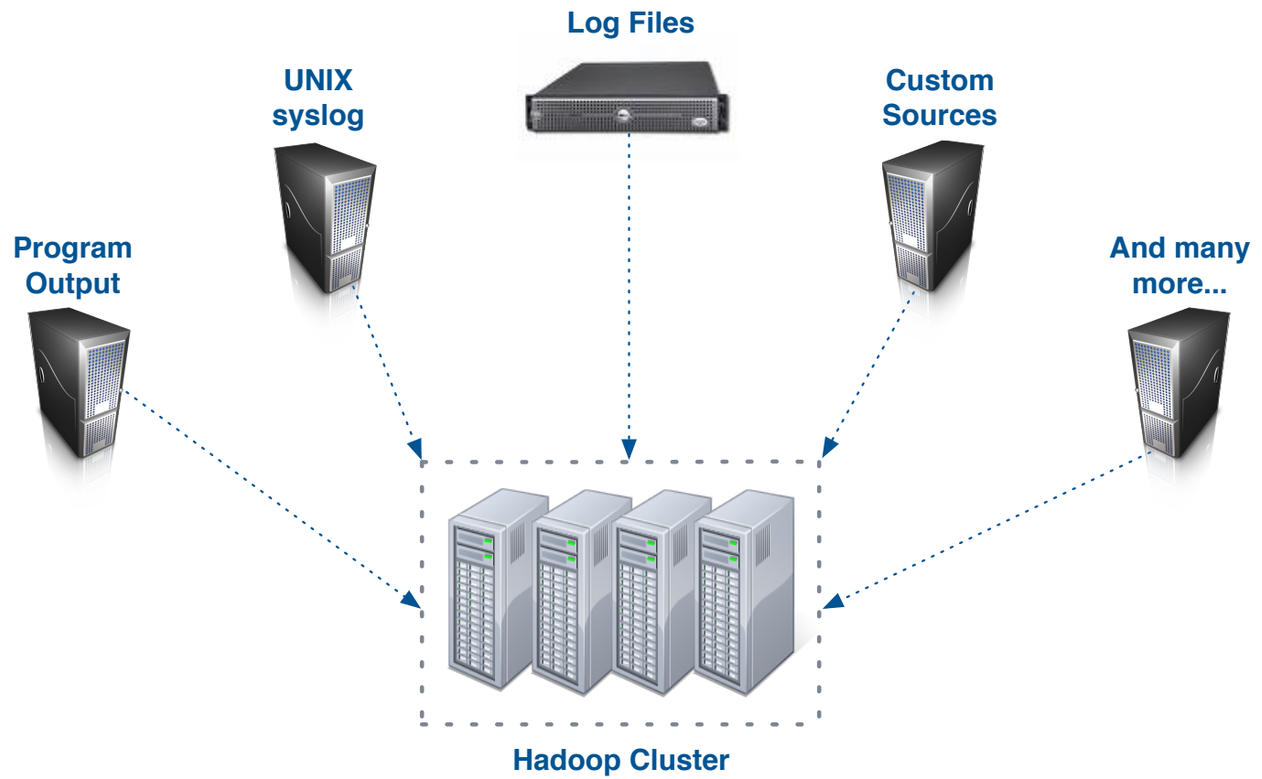
Apache Sqoop

- Sqoop exchanges data between RDBMS and Hadoop
- Can import entire DB, a single table, or a table subset into HDFS
  - Does this very efficiently via a Map-only MapReduce job
  - Can also export data from HDFS back to the database



Apache Flume

- **Flume imports data into HDFS *as it is being generated* by various sources**



Apache Pig

- Pig offers high-level data processing on Hadoop
  - An alternative to writing low-level MapReduce code

```
people = LOAD '/data/customers' AS (cust_id, name);
orders = LOAD '/data/orders' AS (ord_id, cust_id, cost);
groups = GROUP orders BY cust_id;
totals = FOREACH groups GENERATE group, SUM(orders.cost) AS t;
result = JOIN totals BY group, people BY cust_id;
DUMP result;
```

- Pig turns this into MapReduce jobs that run on Hadoop

Apache Hive

- Hive is another abstraction on top of MapReduce
  - Like Pig, it also reduces development time
  - Hive uses a SQL-like language called HiveQL


```
  SELECT customers.cust_id, SUM(cost) AS total
    FROM customers
    JOIN orders
      ON customers.cust_id = orders.cust_id
GROUP BY customers.cust_id
ORDER BY total DESC;
```

Apache Mahout

- Mahout is a scalable machine learning library
- Support for several categories of problems
  - Classification
  - Clustering
  - Collaborative filtering
  - Frequent itemset mining
- Many algorithms implemented in MapReduce
  - Can parallelize inexpensively with Hadoop

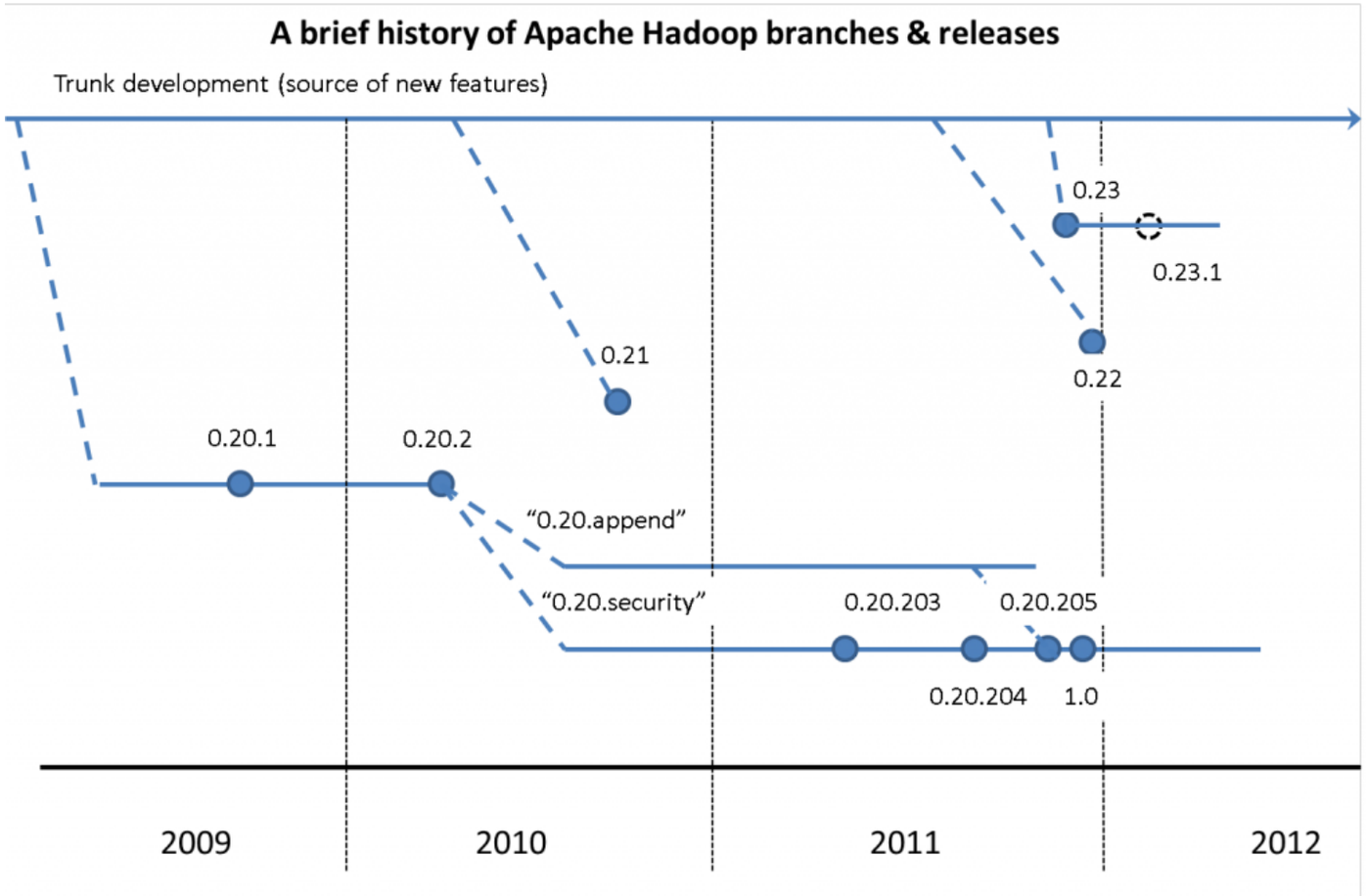
Apache HBase

- HBase is a NoSQL database built on top of Hadoop
- Can store massive amounts of data
  - Gigabytes, terabytes, and even petabytes of data in a table
  - Tables can have many thousands of columns
- Scales to provide very high write throughput
  - Hundreds of thousands of inserts per second

Cloudera Impala

- Massively parallel SQL engine which runs on a Hadoop cluster
  - Inspired by Google's Dremel project
  - Can query data stored in HDFS or HBase tables
- High performance
  - Typically > 10 times faster than Pig or Hive
  - High-level query language (subset of SQL)
- Impala is 100% open source (Apache-licensed)

Hadoop Versions: Historically Confusing



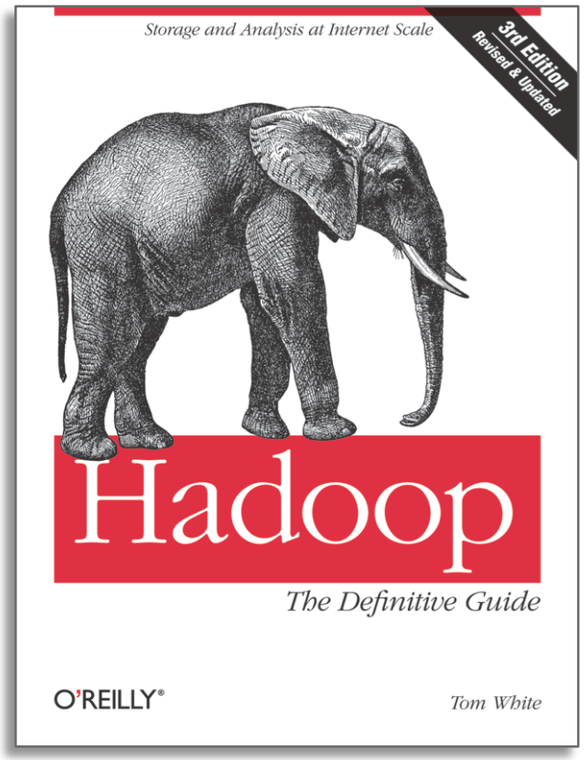
Hadoop Distributions

- Rather than tracking the stable version yourself…
  - Consider using a software distribution including Apache Hadoop
- Distributions typically include Flume, Sqoop, Pig, Hive, etc.
- Other benefits of a distribution
  - Integration testing helps ensure all tools work together
  - Easy installation and updates
  - Commercial support
  - Compatibility certification from hardware vendors

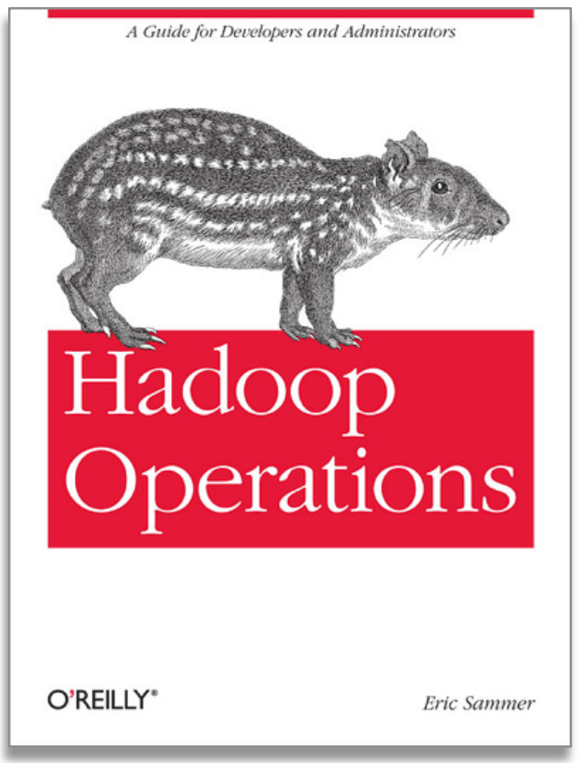
Cloudera's Distribution (CDH)

- Cloudera's Distribution including Apache Hadoop (CDH)
  - The most widely used distribution of Hadoop
  - Stable, proven and supported environment
- Combines Hadoop with many important ecosystem tools
  - Including all those I've mentioned
- How much does it cost?
  - It's completely free
  - Apache licensed – it's 100% open source too

Highly Recommended Books

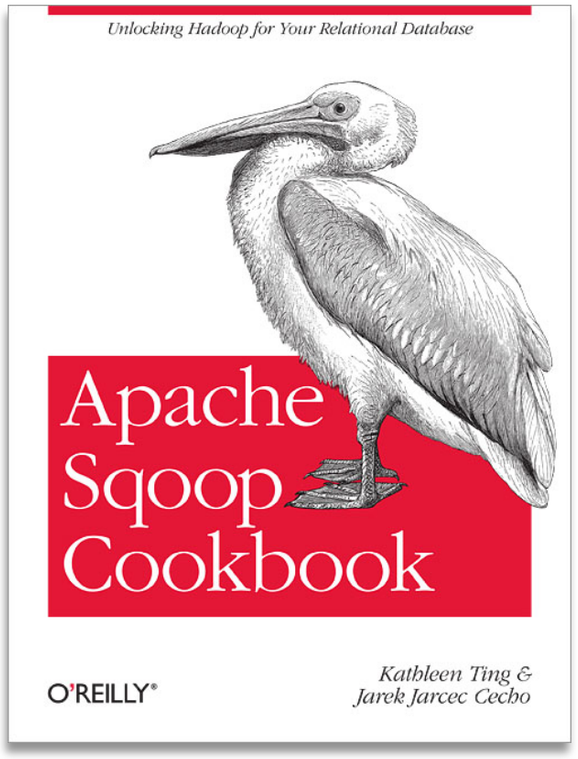


Author: Tom White
ISBN: 1-449-31152-0



Author: Eric Sammer
ISBN: 1-449-32705-2

Book Signing at OSCON



Authors: Ting and Cecho
ISBN: 1-4493-6462-4

- Great new book detailing how to use Apache Sqoop
- One of the authors, Kathleen Ting, will be signing copies at OSCON
  - Thursday at 12:45 PM
  - Go to the O'Reilly booth

More Hadoop Presentations at OSCON

- Want to learn how to quickly set up your *own* cluster?
  - Mark Grover: "Getting Hadoop, Hive and HBase up and running in less than 15 minutes"
  - Wednesday at 4:10 PM in Portland 256
- And for an interesting Hadoop use case…
  - Jesse Anderson: "Doing Data Science on NFL Play-by-Play"
  - Wednesday at 2:30 PM in Portland 256
- See schedule for even more Hadoop-related talks at OSCON

Questions?

- Thank you for attending!
- I'll be happy to answer any additional questions now…
  - If you have questions later, stop by Cloudera's booth
- Want to learn even more?
  - Cloudera training: developers, analysts, sysadmins, and more
  - Offered in more than 50 cities worldwide, and online too!
  - See http://university.cloudera.com/ for more info


cloudera®
Ask Bigger Questions