CC5212-1

PROCESAMIENTO MASIVO DE DATOS

OTOÑO 2017

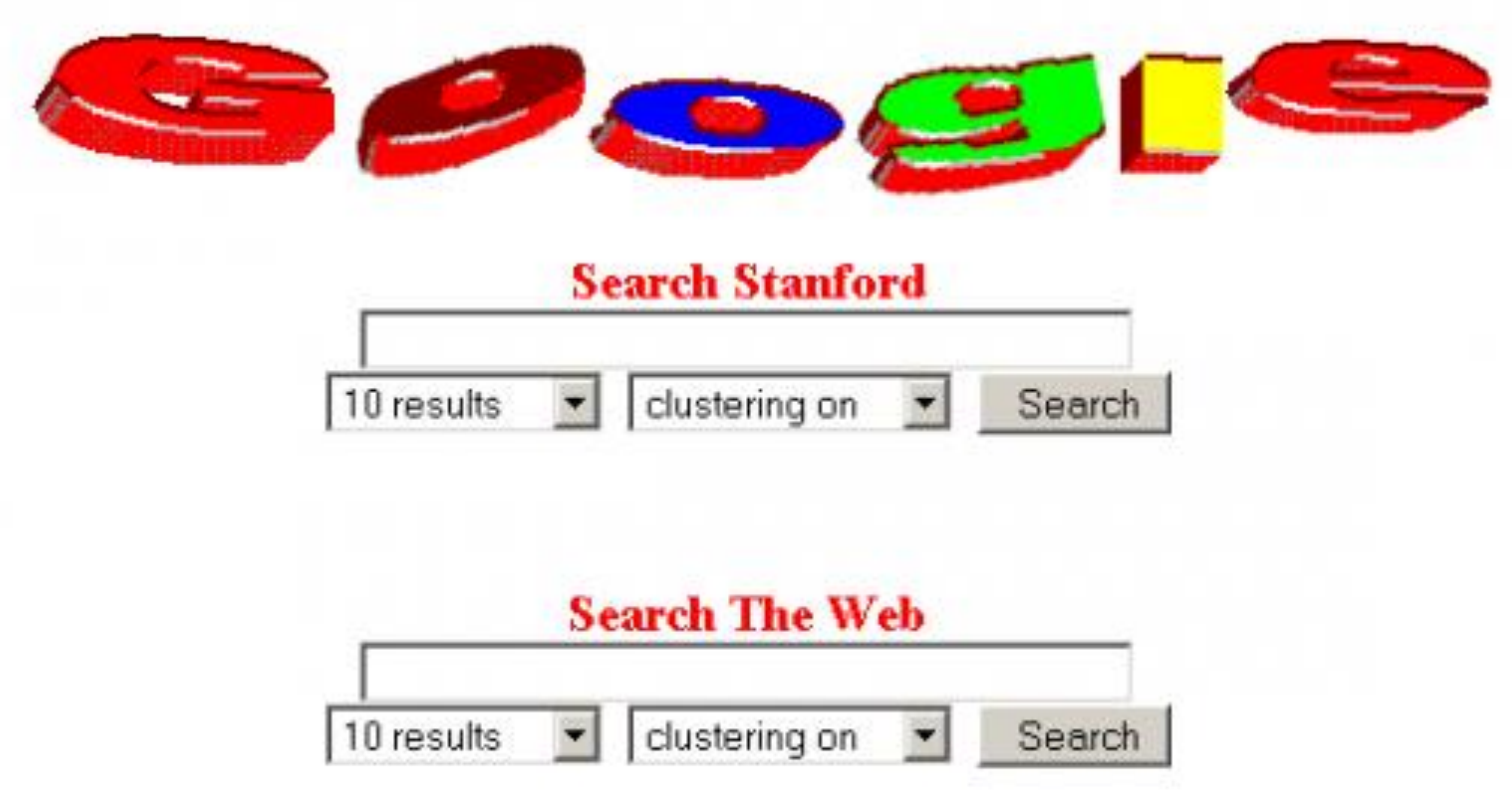
Lecture 3: Google File System + MapReduce

Aidan Hogan

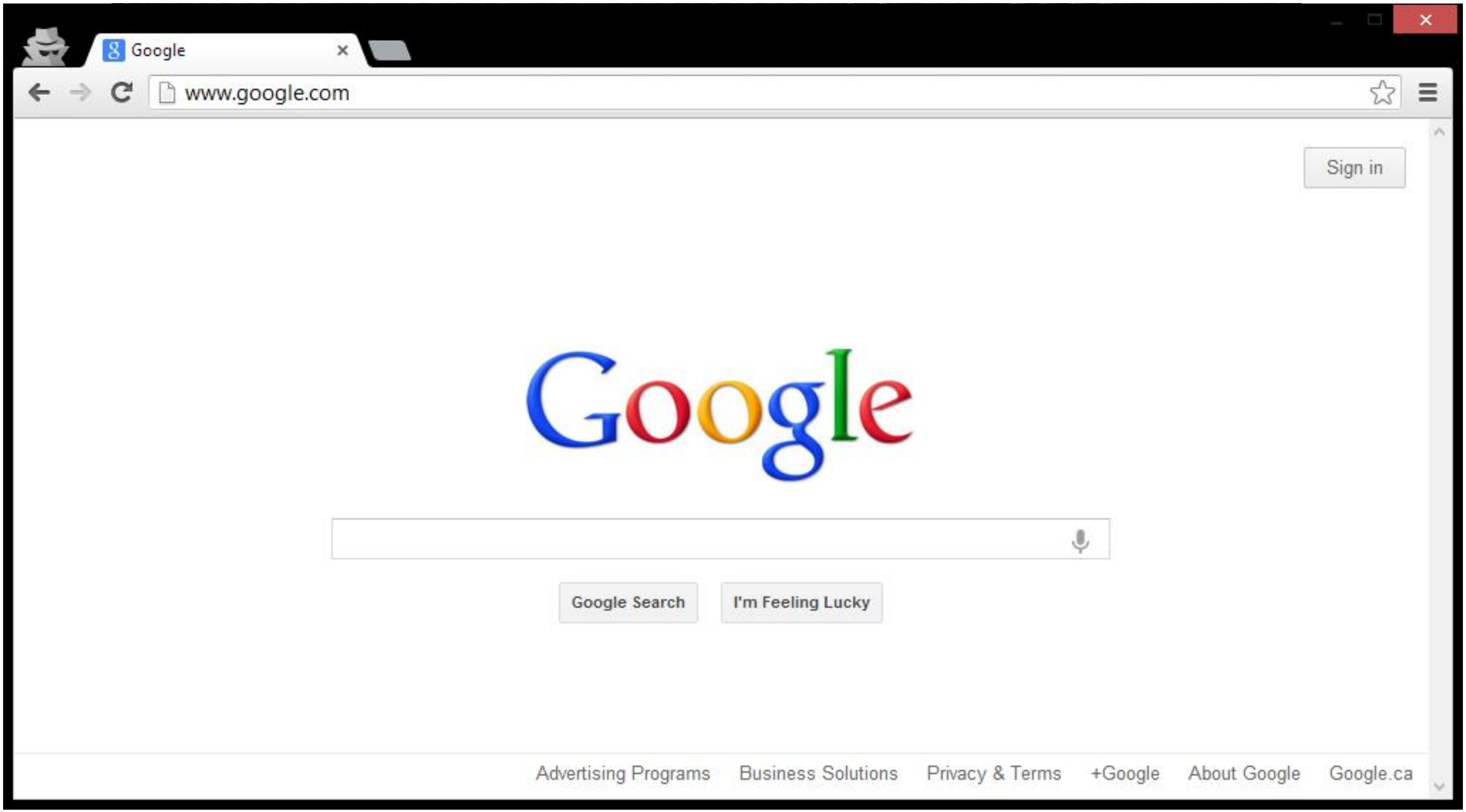
aidhog@gmail.com

MASSIVE DATA PROCESSING (THE GOOGLE WAY …)

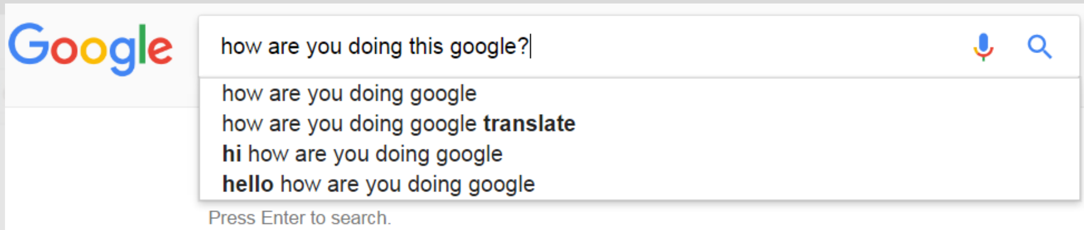
Inside Google circa 1997/98



Inside Google circa 2017



Building Google Web-search



What processes/algorithms does Google need to implement Web search?

Crawling

1. Parse links from webpages
2. Schedule links for crawling
3. Download pages, GOTO 1

Indexing

1. Parse keywords from webpages
2. Index keywords to webpages
3. Manage updates

Ranking

1. How relevant is a page? (TF-IDF)
2. How important is it? (PageRank)
3. How many users clicked it?

…

Building Google Web-search

Google
≈ 100 PB / day
≈ 2,000,000 Wiki / day
(2014, processed)

Building Google Web-search

Google
≈ 100 PB / day
≈ 2,000,000 Wiki / day
(2014, process

3. Download pa 1

Ranking

1. How relevant is a page? (TF-IDF)
2. How important is it? (PageRank)
3. How many users clicked it?

Implementing on thousands of machines

Crawling

1. Parse links from webpages
2. Schedule links for crawling
3. Download pages, GOTO 1

Indexing

1. Parse keywords from webpages
2. Index keywords to webpages
3. Manage updates

Ranking

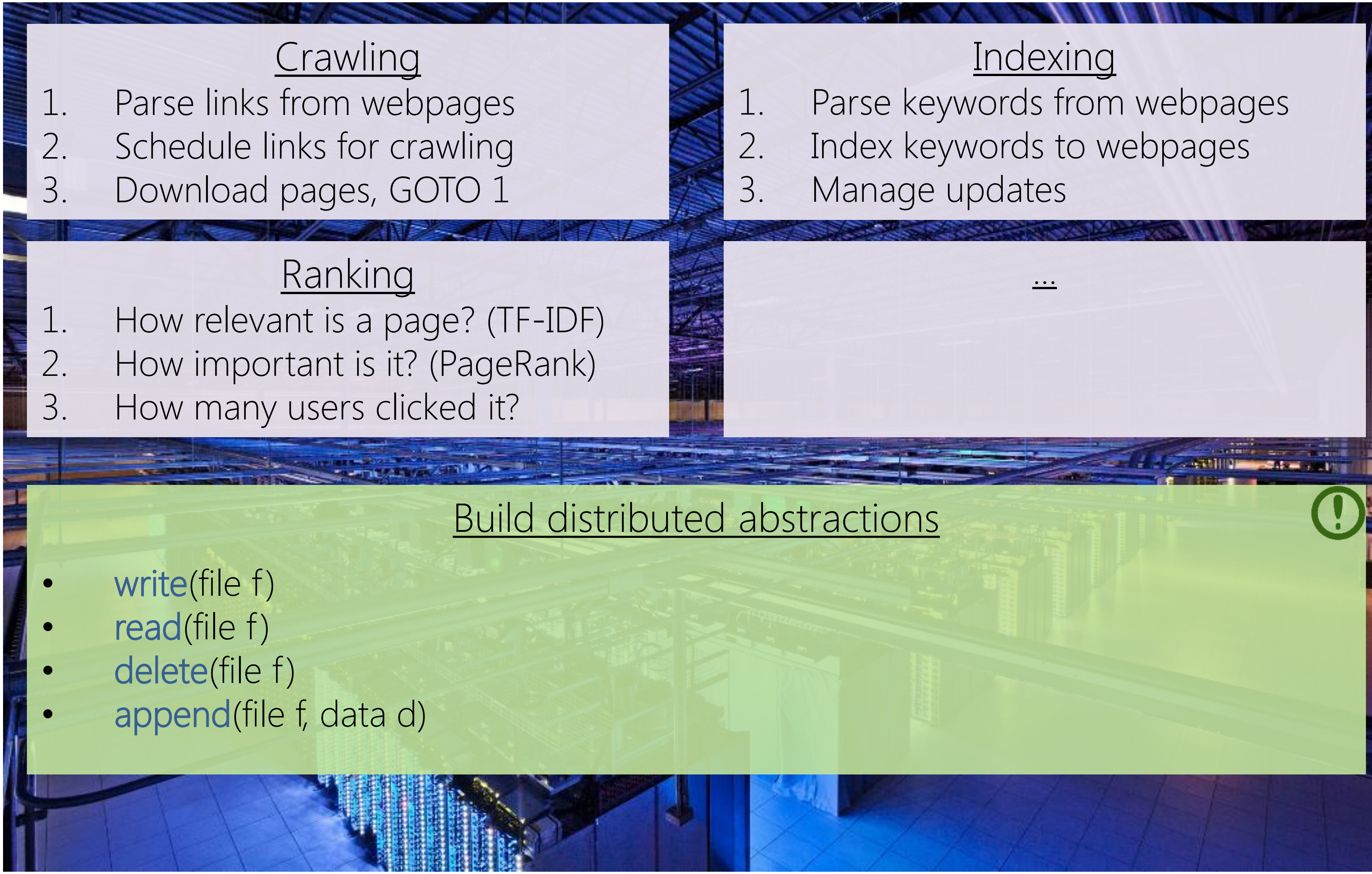
1. How relevant is a page? (TF-IDF)
2. How important is it? (PageRank)
3. How many users clicked it?

...

If we implement each task separately …

… re-implement storage
… re-implement retrieval
… re-implement distributed processing
… re-implement communication
… re-implement fault-tolerance
… re-implement the wheel

Implementing on thousands of machines



Crawling

1. Parse links from webpages
2. Schedule links for crawling
3. Download pages, GOTO 1

Indexing

1. Parse keywords from webpages
2. Index keywords to webpages
3. Manage updates

Ranking

1. How relevant is a page? (TF-IDF)
2. How important is it? (PageRank)
3. How many users clicked it?

...

Build distributed abstractions

- write(file f)
- read(file f)
- delete(file f)
- append(file f, data d)

GOOGLE FILE SYSTEM (GFS)

Google File System (GFS): White-Paper

The Google File System

Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung
Google*

ABSTRACT

We have designed and implemented the Google File System, a scalable distributed file system for large distributed data-intensive applications. It provides fault tolerance while running on inexpensive commodity hardware, and it delivers high aggregate performance to a large number of clients.

While sharing many of the same goals as previous distributed file systems, our design has been driven by observations of our application workloads and technological environment, both current and anticipated, that reflect a marked departure from some earlier file system assumptions. This has led us to reexamine traditional choices and explore radically different design points.

The file system has successfully met our storage needs. It is widely deployed within Google as the storage platform for the generation and processing of data used by our service as well as research and development efforts that require large data sets. The largest cluster to date provides hundreds of terabytes of storage across thousands of disks on over a thousand machines, and it is concurrently accessed by hundreds of clients.

In this paper, we present file system interface extensions designed to support distributed applications, discuss many aspects of our design, and report measurements from both micro-benchmarks and real world use.

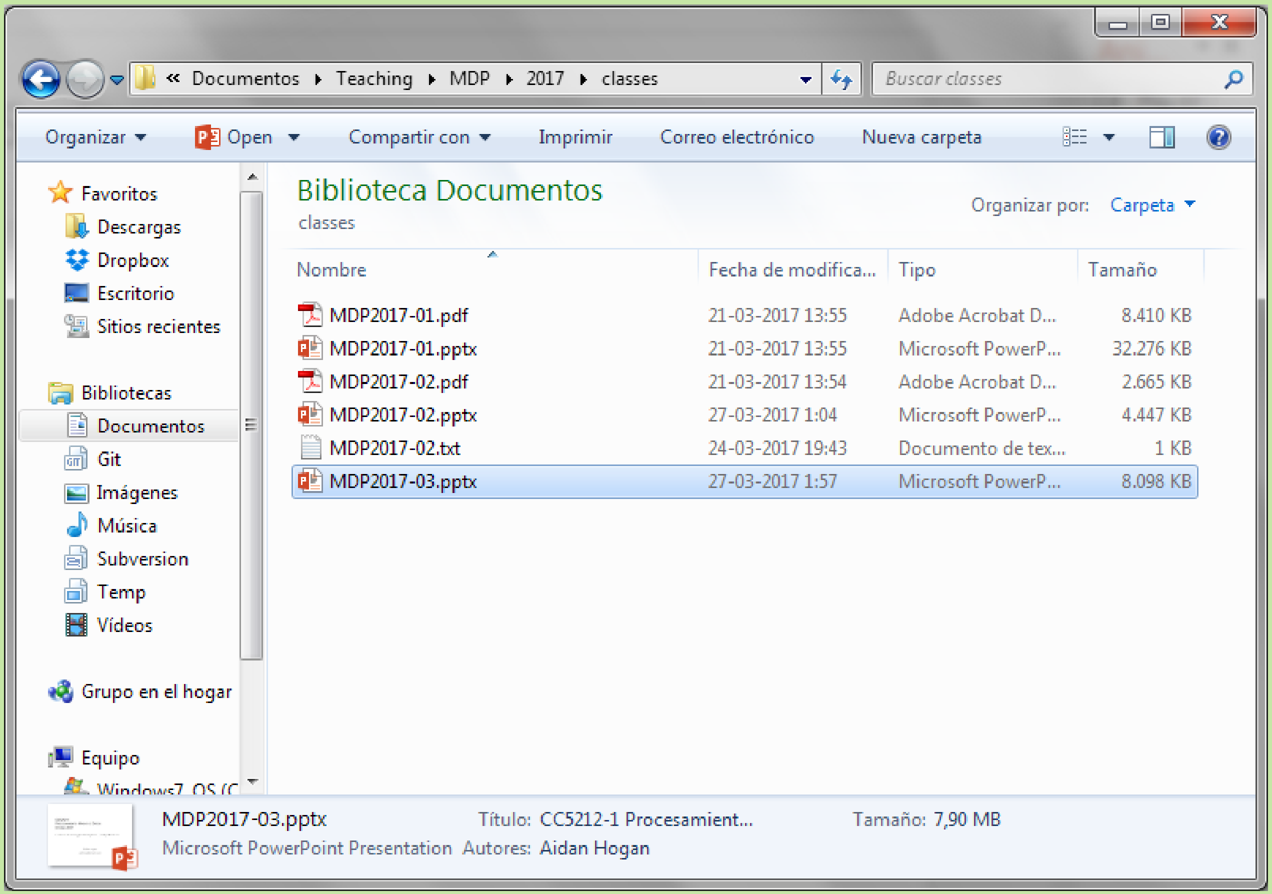
1. INTRODUCTION

We have designed and implemented the Google File System (GFS) to meet the rapidly growing demands of Google's data processing needs. GFS shares many of the same goals as previous distributed file systems such as performance, scalability, reliability, and availability. However, its design has been driven by key observations of our application workloads and technological environment, both current and anticipated, that reflect a marked departure from some earlier file system design assumptions. We have reexamined traditional choices and explored radically different points in the design space.

First, component failures are the norm rather than the exception. The file system consists of hundreds or even thousands of storage machines built from inexpensive commodity parts and is accessed by a comparable number of client machines. The quantity and quality of the components virtually guarantee that some are not functional at any given time and some will not recover from their current failures. We have seen problems caused by application bugs, operating system bugs, human errors, and the failures of disks, memory, connectors, networking, and power supplies. Therefore, constant monitoring, error detection, fault tolerance, and automatic recovery must be integral to the system.

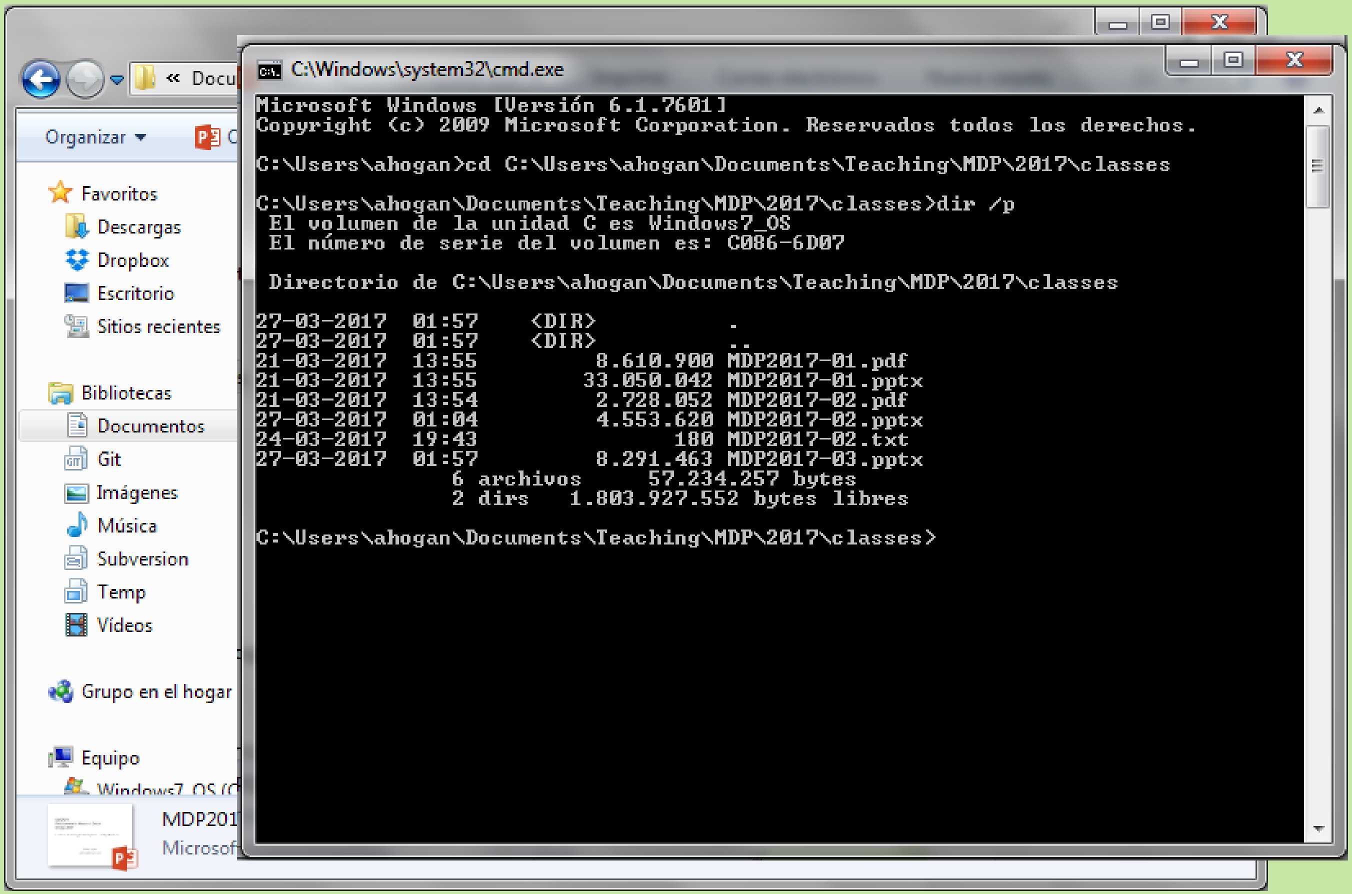
Google File System

What is a “file-system”?



Google File System

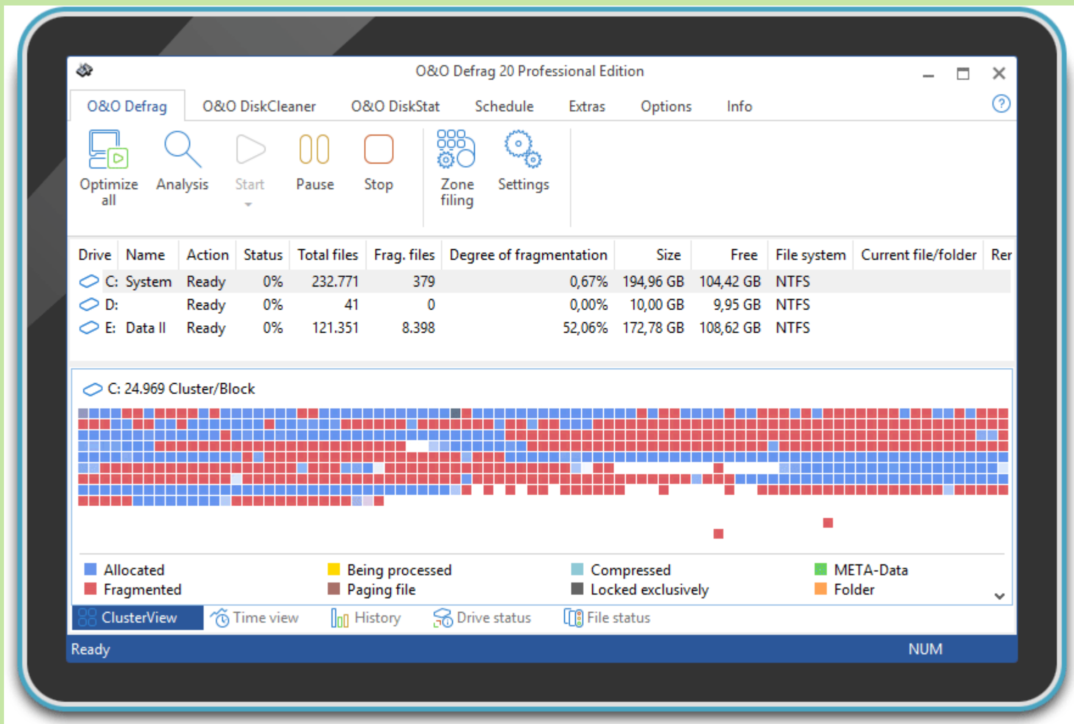
What is a "file-system"?



Google File System

What does a "file-system" do?

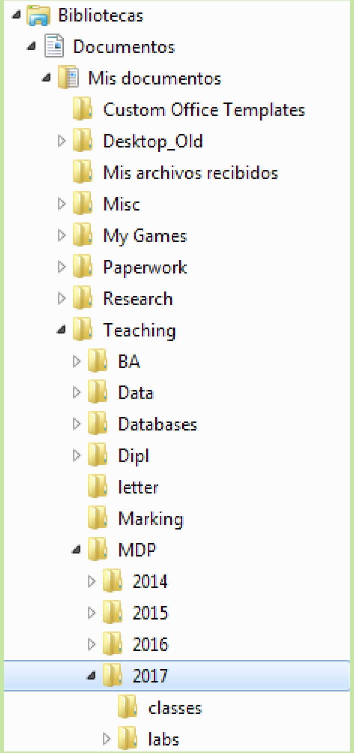
1. Splits a file up into chunks (blocks/clusters) of storage
   - Remembers location and sequence of chunks for a file



Google File System

What does a “file-system” do?

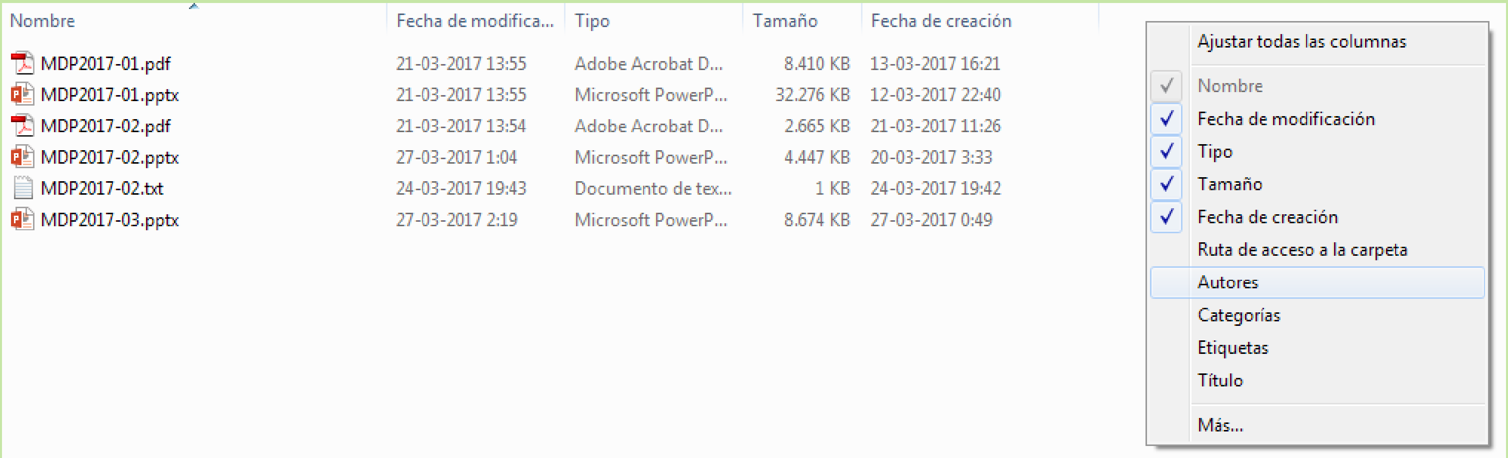
1. Splits a file up into chunks (blocks/clusters) of storage
   - Remembers location and sequence of chunks for a file
2. Organises a hierarchical directory structure
   - Tracks sub-directories and files in directories



Google File System

What does a "file-system" do?

1. Splits a file up into chunks (blocks/clusters) of storage
   - Remembers location and sequence of chunks for a file
2. Organises a hierarchical directory structure
   - Tracks sub-directories and files in directories
3. Tracks file meta-data
   - File size, date created, date last modified
   - Ownership, permissions, locks



Google File System

What does a “file-system” do?

1. Splits a file up into chunks (blocks/clusters) of storage
   - Remembers location and sequence of chunks for a file
2. Organises a hierarchical directory structure
   - Tracks sub-directories and files in directories
3. Tracks file meta-data
   - File size, date created, date last modified
   - Ownership, permissions, locks
4. Provides read/write/update/delete interface, etc.

Google File System

What does “Google File System” do?

1. Splits a file up into chunks (blocks/clusters) of storage
   - Remembers location and sequence of chunks for a file
2. Organises a hierarchical directory structure
   - Tracks sub-directories and files in directories
3. Tracks file meta-data
   - File size, date created, date last modified
   - Ownership, permissions, locks
4. Provides read/write/update/delete interface, etc.

Same thing, just distributed:

In this paper, we present file system interface extensions designed to support distributed applications, discuss many aspects of our design, and report measurements from both micro-benchmarks and real world use.

of disks, memory, connectors, networking, and power supplies. Therefore, constant monitoring, error detection, fault tolerance, and automatic recovery must be integral to the system.

Google File System

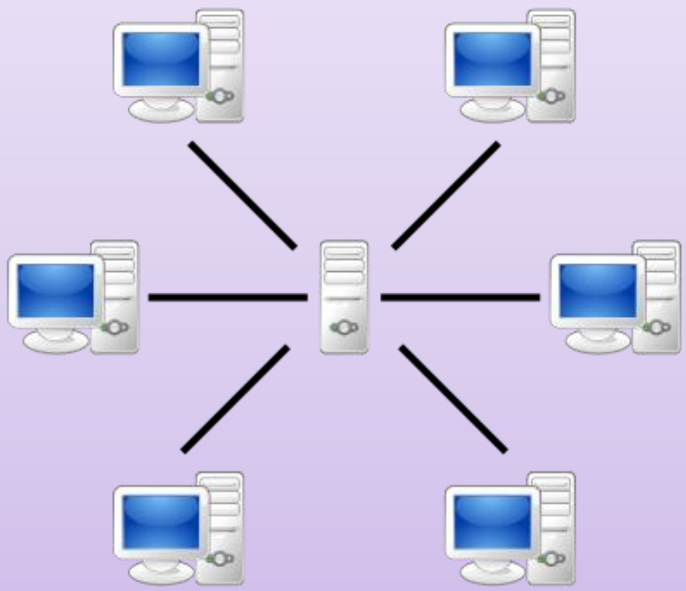
What would
Transparency / Flexibility / Reliability / Performance / Scalability
mean for a distributed file system?

- Transparency: Like a normal file-system
- Flexibility: Can mount new machines
- Reliability: Has replication
- Performance: Fast storage / retrieval
- Scalability: Can store a lot of data / support a lot of machines

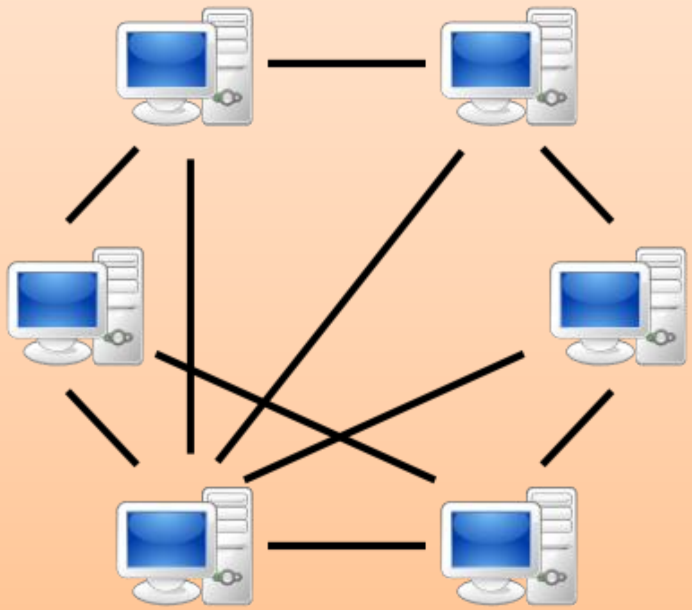
Google File System

So which architecture do you think Google uses?

Client–Server?



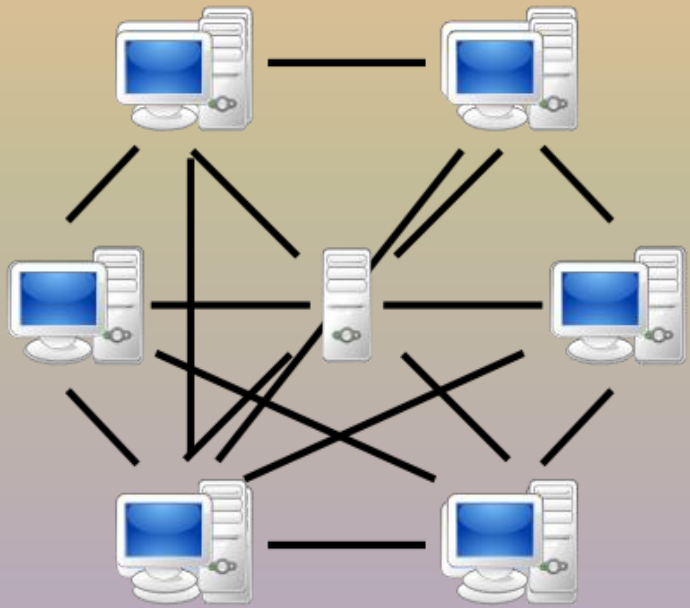
Peer-To-Peer?



Google File System

So which architecture do you think Google uses?

Client–Peer-To-Server-To-Peer-Server-Client!

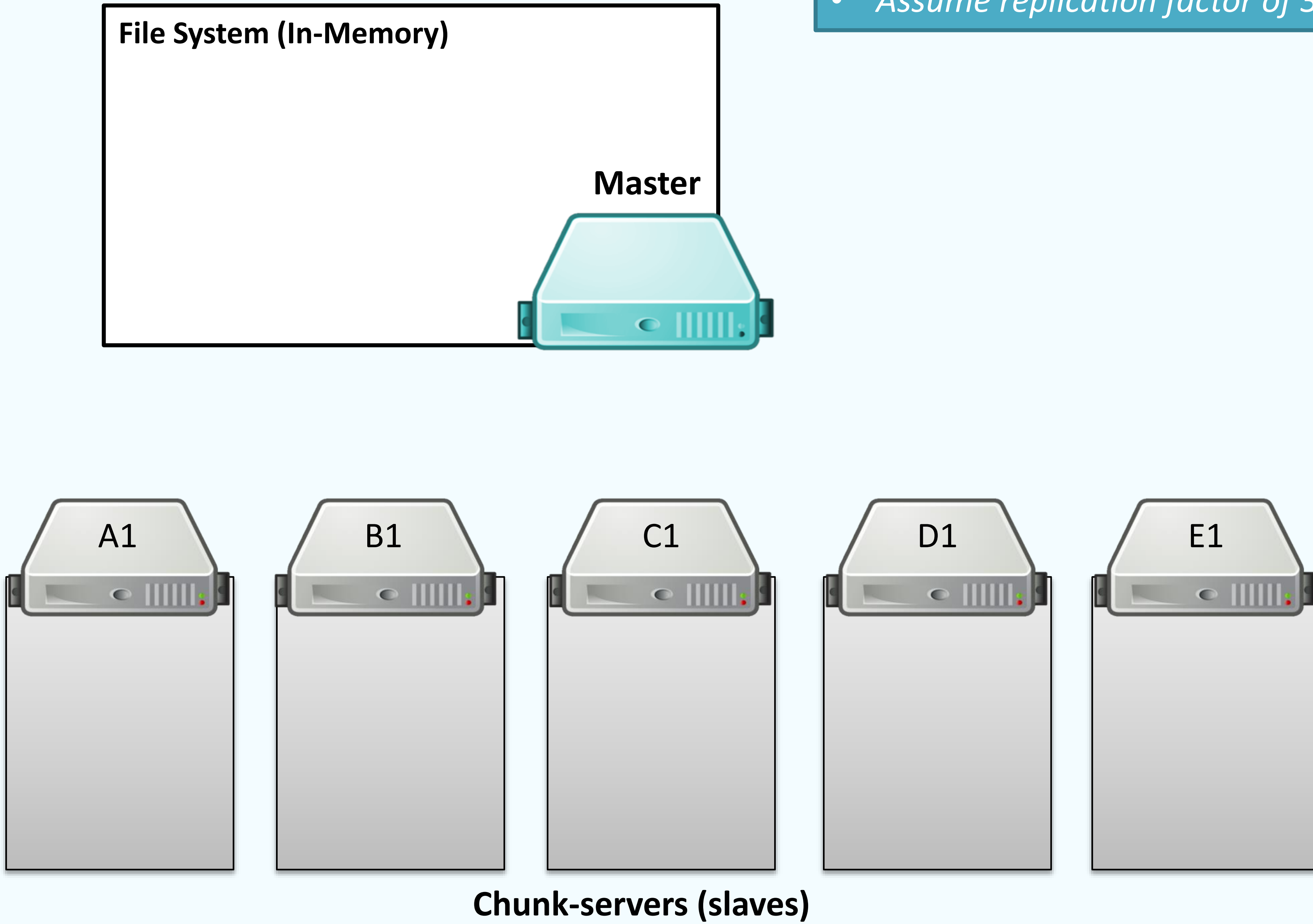


Google File System: Assumptions

- Files are huge
- Files often read or appended
- Concurrency important
- Failures are frequent
- Streaming important

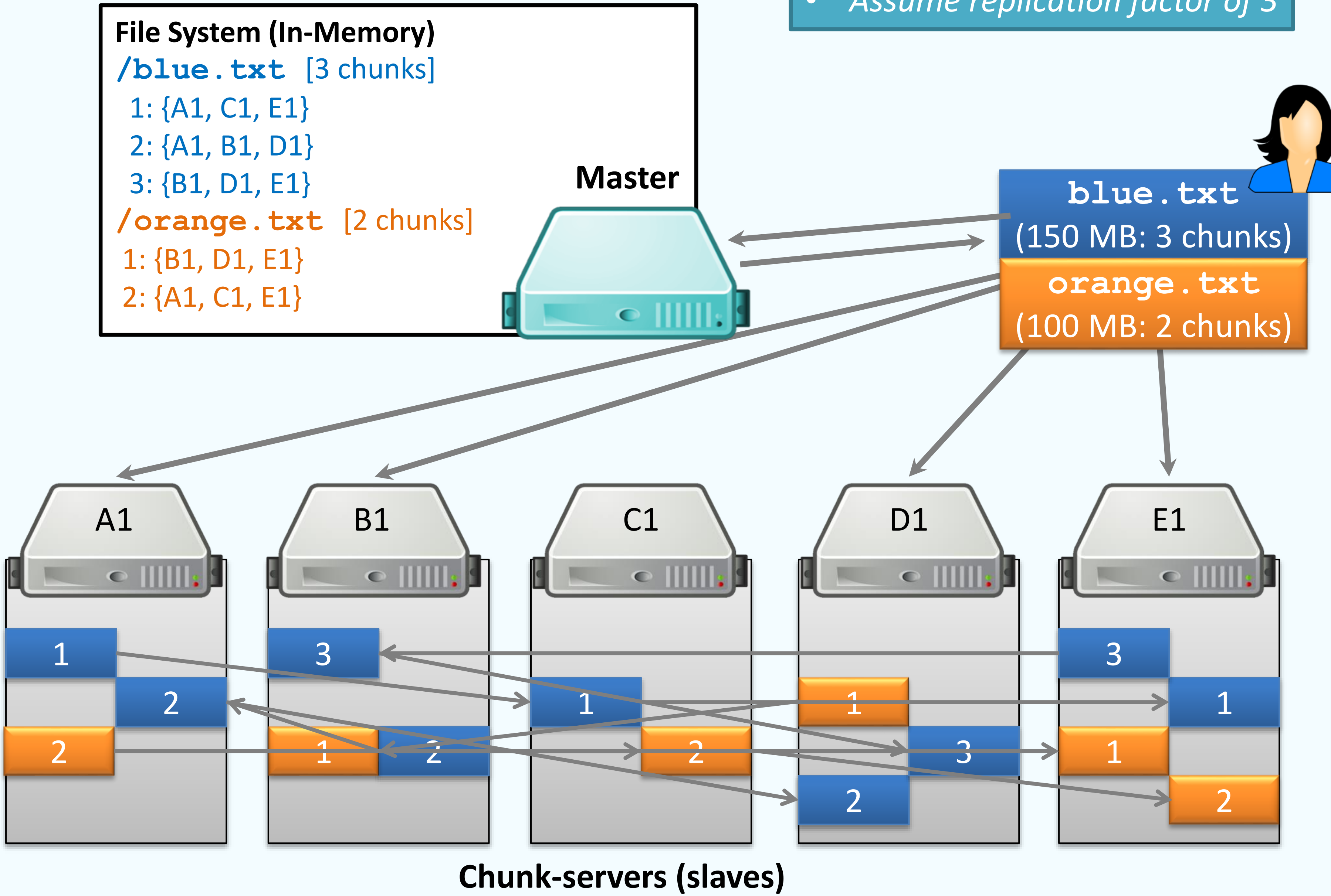
GFS: Architecture

- *64 MB per chunk*
- *64 bit label for each chunk*
- *Assume replication factor of 3*



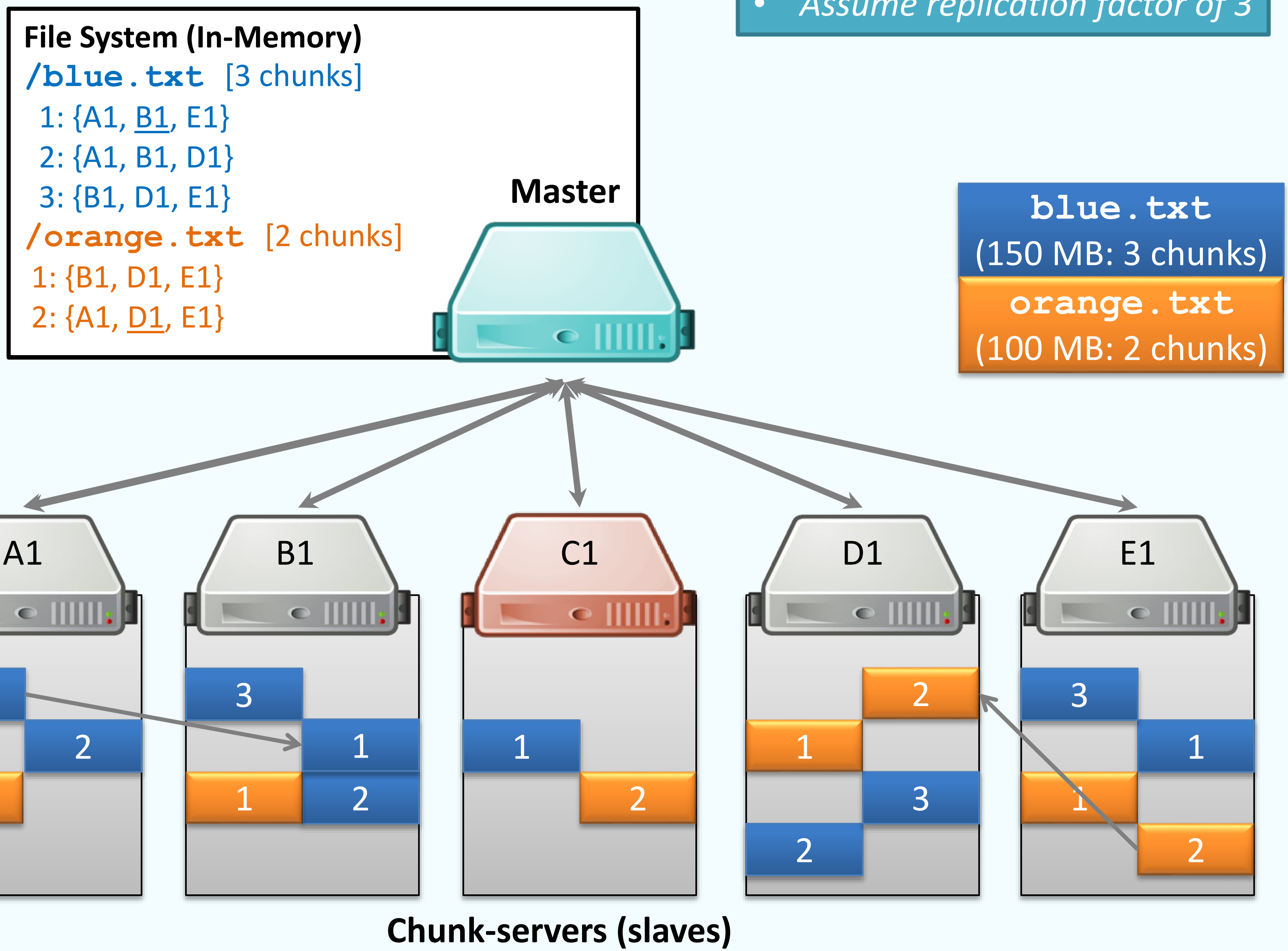
GFS: Pipelined Writes

- *64 MB per chunk*
- *64 bit label for each chunk*
- *Assume replication factor of 3*



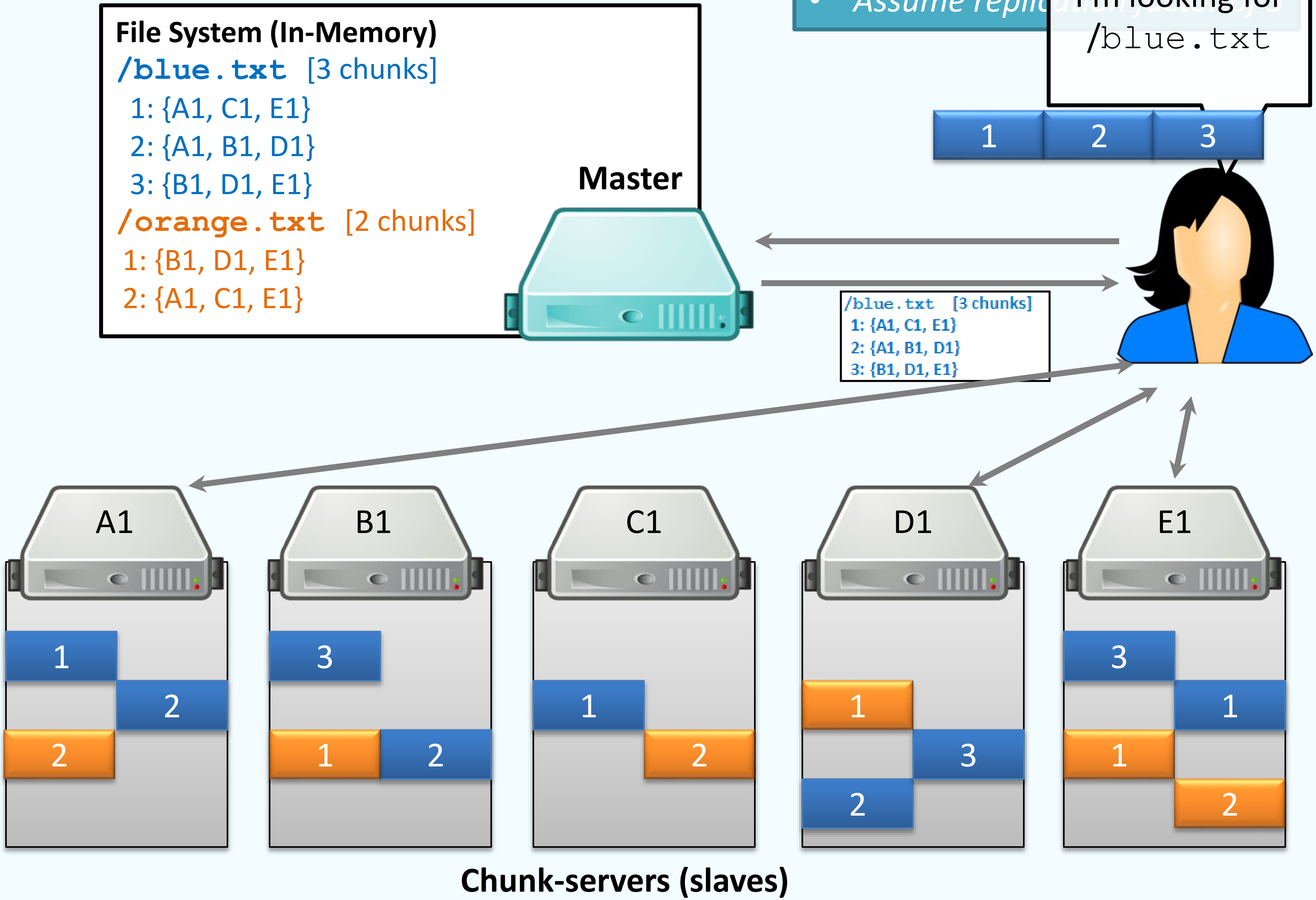
GFS: Fault Tolerance

- *64 MB per chunk*
- *64 bit label for each chunk*
- *Assume replication factor of 3*



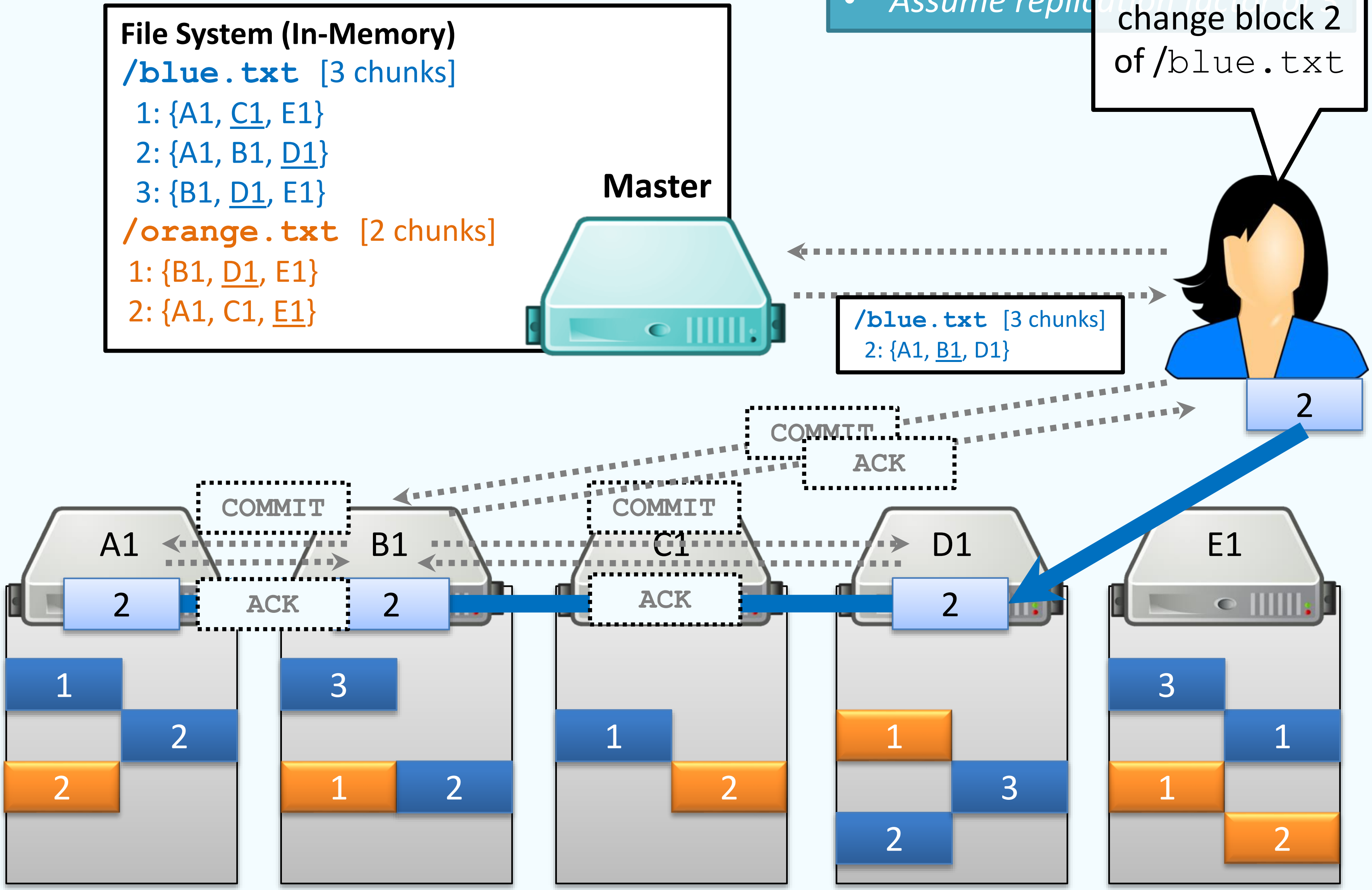
GFS: Direct Reads

- *64 MB per chunk*
- *64 bit label for each chunk*
- *Assume replic*



GFS: Primary Replicas

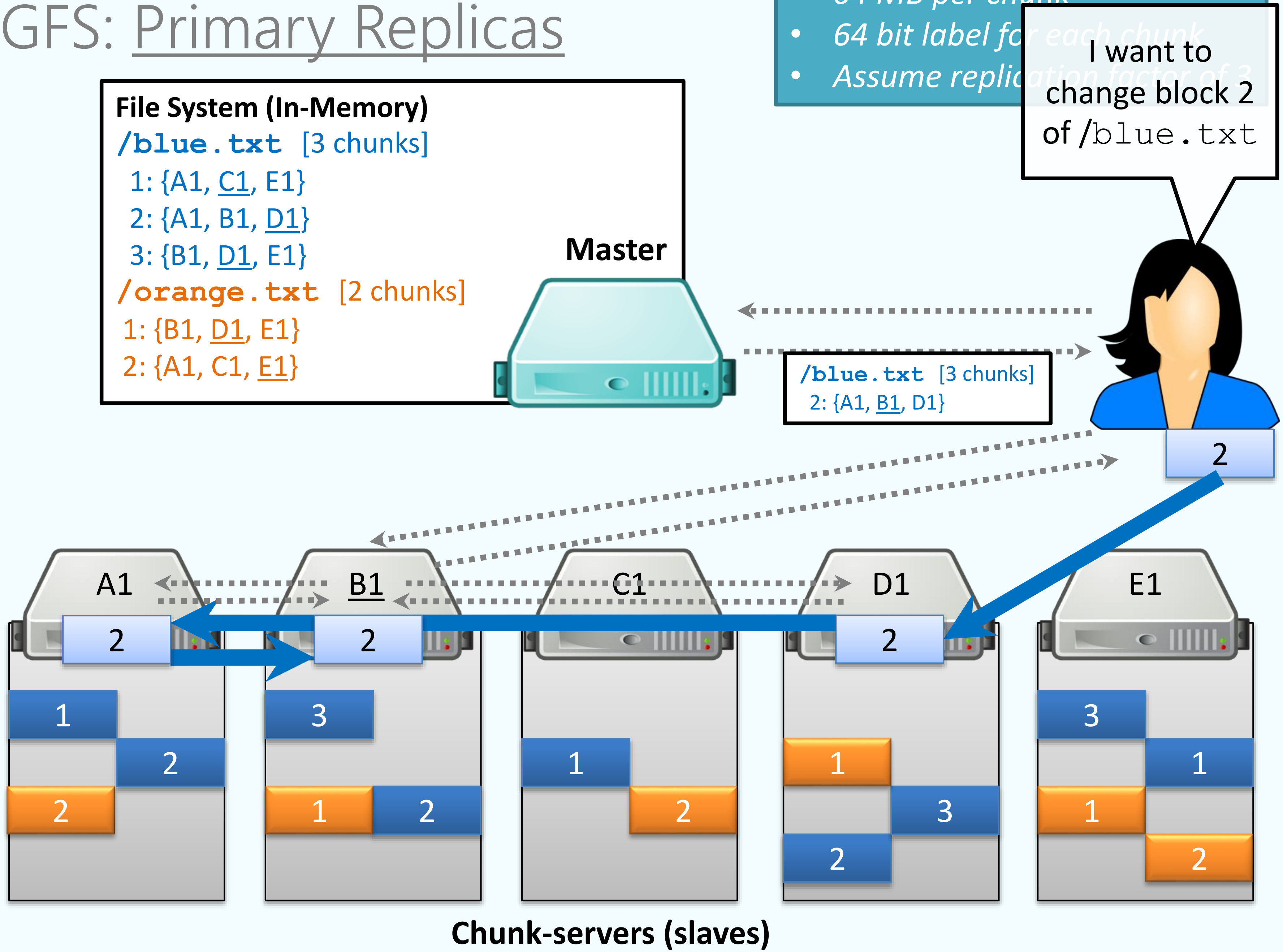
- *64 MB per chunk*
- *64 bit label for each chunk*
- *Assume replication factor of 3*



Chunk-servers (slaves)

GFS: Primary Replicas

- *64 MB per chunk*
- *64 bit label for each chunk*
- *Assume replication factor of 3*

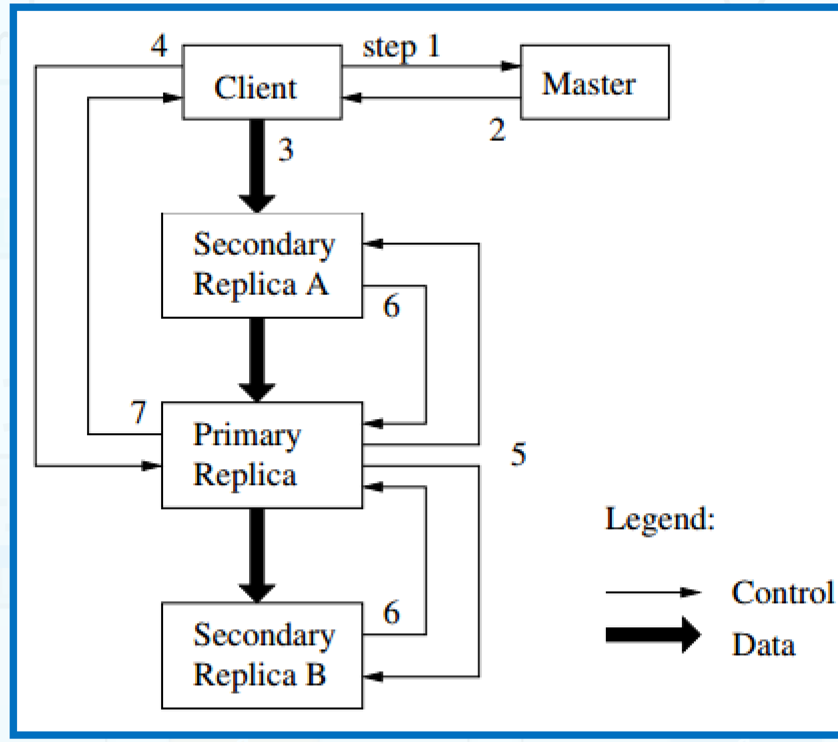


GFS: Primary Replicas

Master assigns leases to one replica: a “primary replica”

Client wants to change a file:

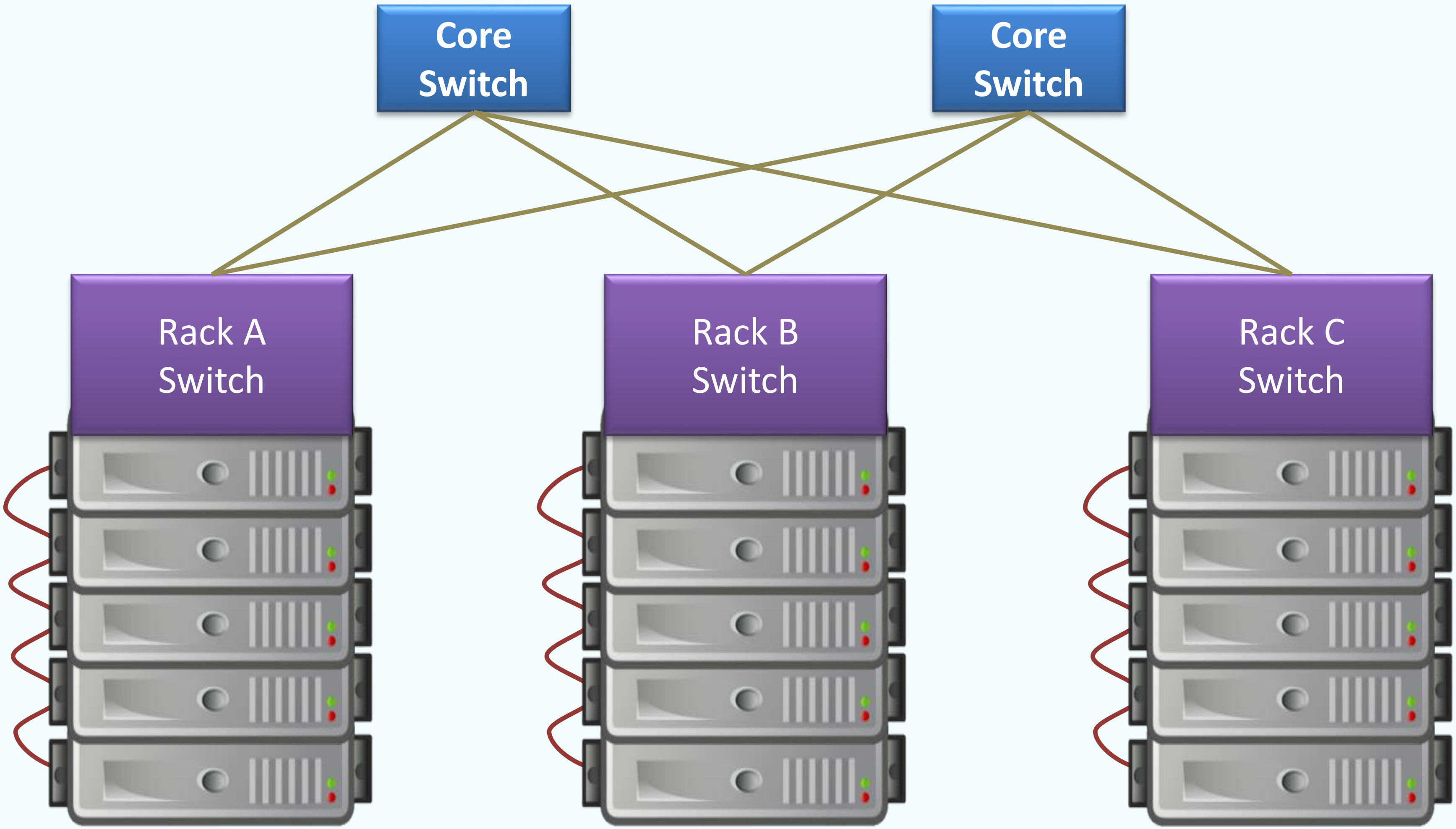
1. Client asks Master for the replicas (incl. primary)
2. Master returns replica info to the client
3. Client sends change data
4. Client asks primary to commit the changes
5. Primary asks secondaries to commit the changes
6. Secondaries acknowledge to primary
7. Primary acknowledges to client



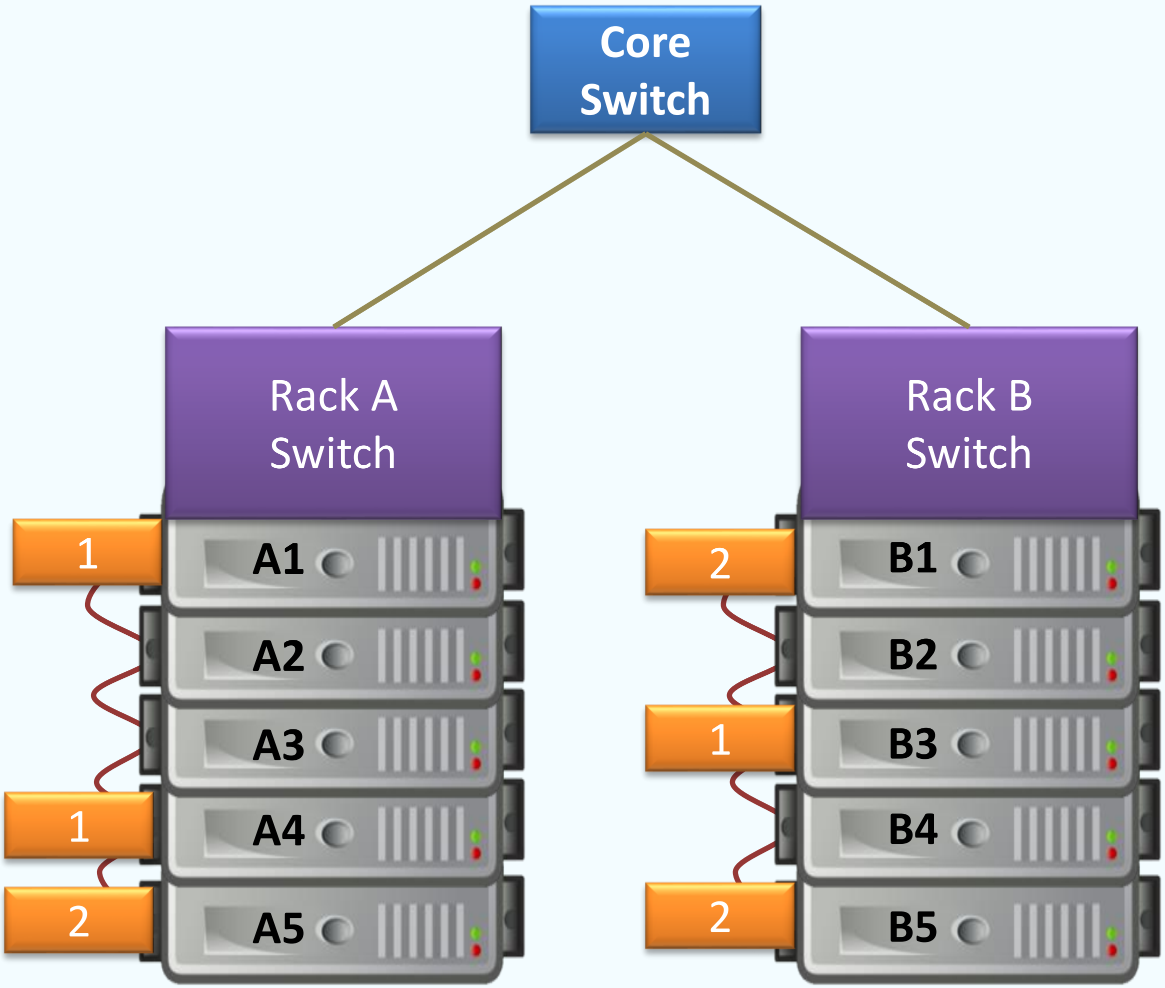
Data and control flow kept separate

GFS: Rack Awareness

GFS: Rack Awareness



GFS: Rack Awareness



Files:
`/orange.txt`
1: {A1, A4, B3}
2: {A5, B1, B5}

Racks:
A: {A1, A2, A3, A4, A5}
B: {B1, B2, B3, B4, B5}

GFS: Rack Awareness

- Make sure replicas not on same rack
  - In case rack switch fails!
- But communication can be slower:
  - Within rack: pass one switch (rack switch)
  - Across racks: pass three switches (two racks and a core)
- (Typically) pick two racks
  - Two nodes in same rack, one node in another rack
    - (Assuming 3x replication)
- Umm, only necessary if more than one rack

GFS: Other Operations

Rebalancing:

- Spread storage out evenly

Deletion:

- Just rename the file with hidden file name
  - To recover, rename back to original version
  - Otherwise, three days later will be wiped

Monitoring Stale Replicas:

- Dead slave reappears with old data?
  - Master keeps version info

GFS: Weaknesses?

What are the main weaknesses of GFS?

Master node single point of failure

- Use hardware replication
- Logs and checkpoints

Master node is a bottleneck

- Use more powerful machine
- Minimise master node traffic

Master-node metadata kept in memory

Each chunk needs 64 bytes to address

- Chunk data can be queried from each slave
- Keep each chunk large (fewer chunks)

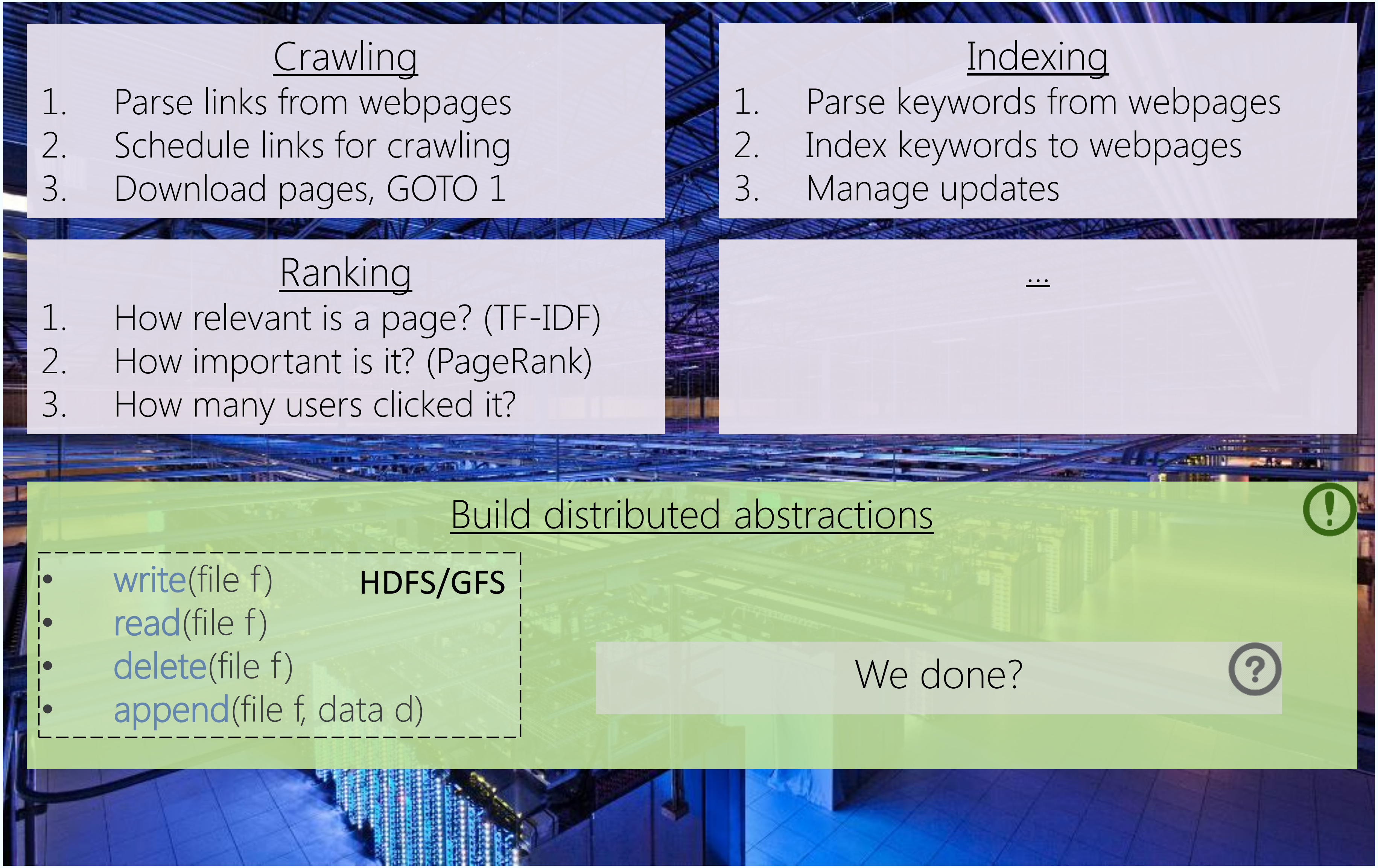
Hadoop Distributed File System

- Open source version of GFS

- HDFS-to-GFS translation guide ...
  - Data-node = Chunkserver/Slave
  - Name-node = Master

- Same idea except ...
  - GFS is proprietary (hidden in Google)
  - HDFS is open source (Apache!)

Implementing on thousands of machines



<u>Crawling</u>

1. Parse links from webpages
2. Schedule links for crawling
3. Download pages, GOTO 1

<u>Indexing</u>

1. Parse keywords from webpages
2. Index keywords to webpages
3. Manage updates

<u>Ranking</u>

1. How relevant is a page? (TF-IDF)
2. How important is it? (PageRank)
3. How many users clicked it?

<u>...</u>

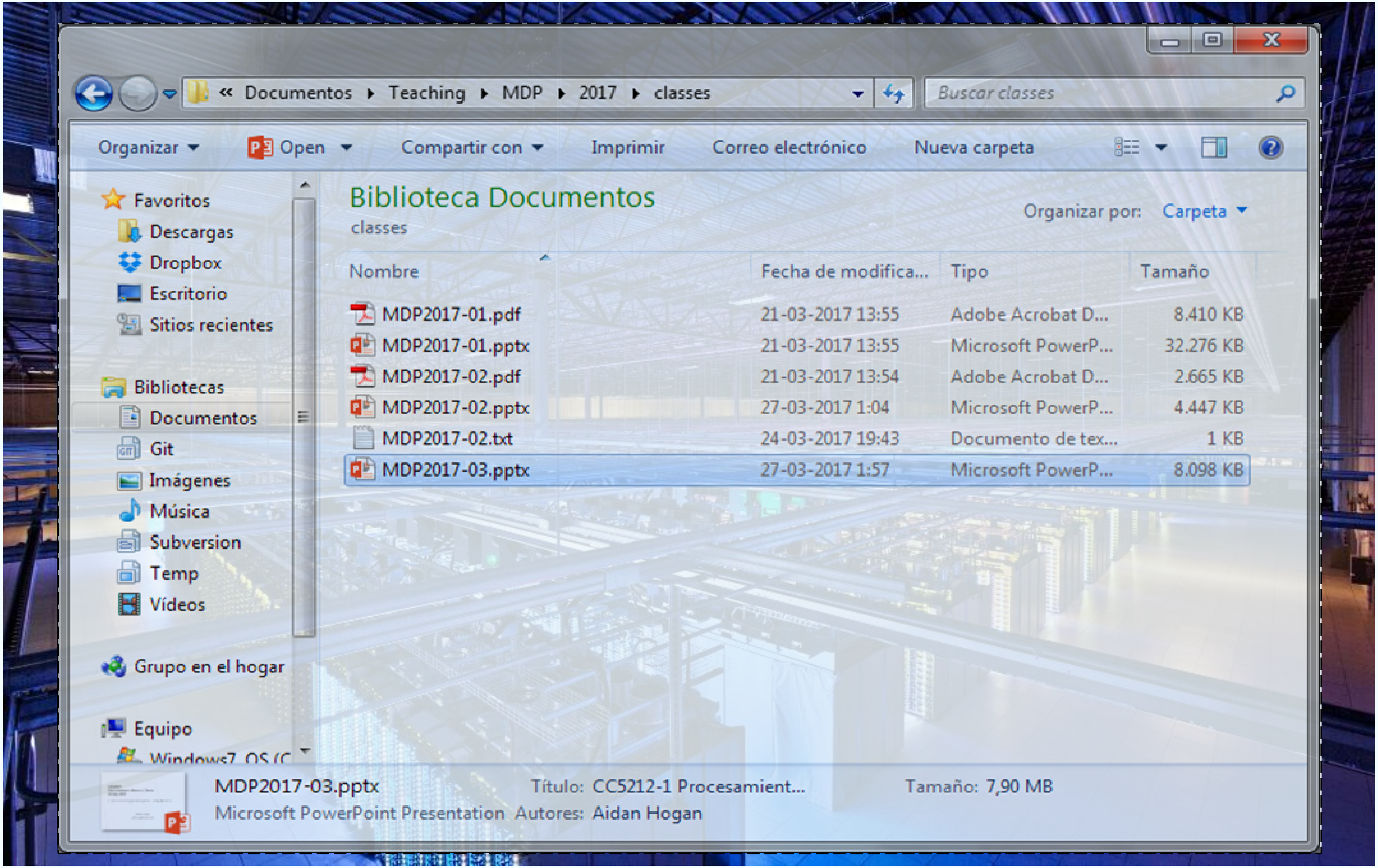
<u>Build distributed abstractions</u>

HDFS/GFS

- write(file f)
- read(file f)
- delete(file f)
- append(file f, data d)

We done?

Implementing on thousands of machines



GOOGLE'S MAPREDUCE

MapReduce: White-Paper

MapReduce: Simplified Data Processing on Large Clusters

Jeffrey Dean and Sanjay Ghemawat

jeff@google.com, sanjay@google.com

Google, Inc.

Abstract

MapReduce is a programming model and an associated implementation for processing and generating large data sets. Users specify a *map* function that processes a key/value pair to generate a set of intermediate key/value pairs, and a *reduce* function that merges all intermediate values associated with the same intermediate key. Many real world tasks are expressible in this model, as shown in the paper.

Programs written in this functional style are automatically parallelized and executed on a large cluster of commodity machines. The run-time system takes care of the details of partitioning the input data, scheduling the program's execution across a set of machines, handling machine failures, and managing the required inter-machine communication. This allows programmers without any experience with parallel and distributed systems to easily utilize the resources of a large distributed system.

given day, etc. Most such computations are conceptually straightforward. However, the input data is usually large and the computations have to be distributed across hundreds or thousands of machines in order to finish in a reasonable amount of time. The issues of how to parallelize the computation, distribute the data, and handle failures conspire to obscure the original simple computation with large amounts of complex code to deal with these issues.

As a reaction to this complexity, we designed a new abstraction that allows us to express the simple computations we were trying to perform but hides the messy details of parallelization, fault-tolerance, data distribution and load balancing in a library. Our abstraction is inspired by the *map* and *reduce* primitives present in Lisp and many other functional languages. We realized that most of our computations involved applying a *map* operation to each logical "record" in our input in order to compute a set of intermediate key/value pairs, and then

Let's start with one task

How could we do a distributed word count?

Count parts in memory on different machines and merge?

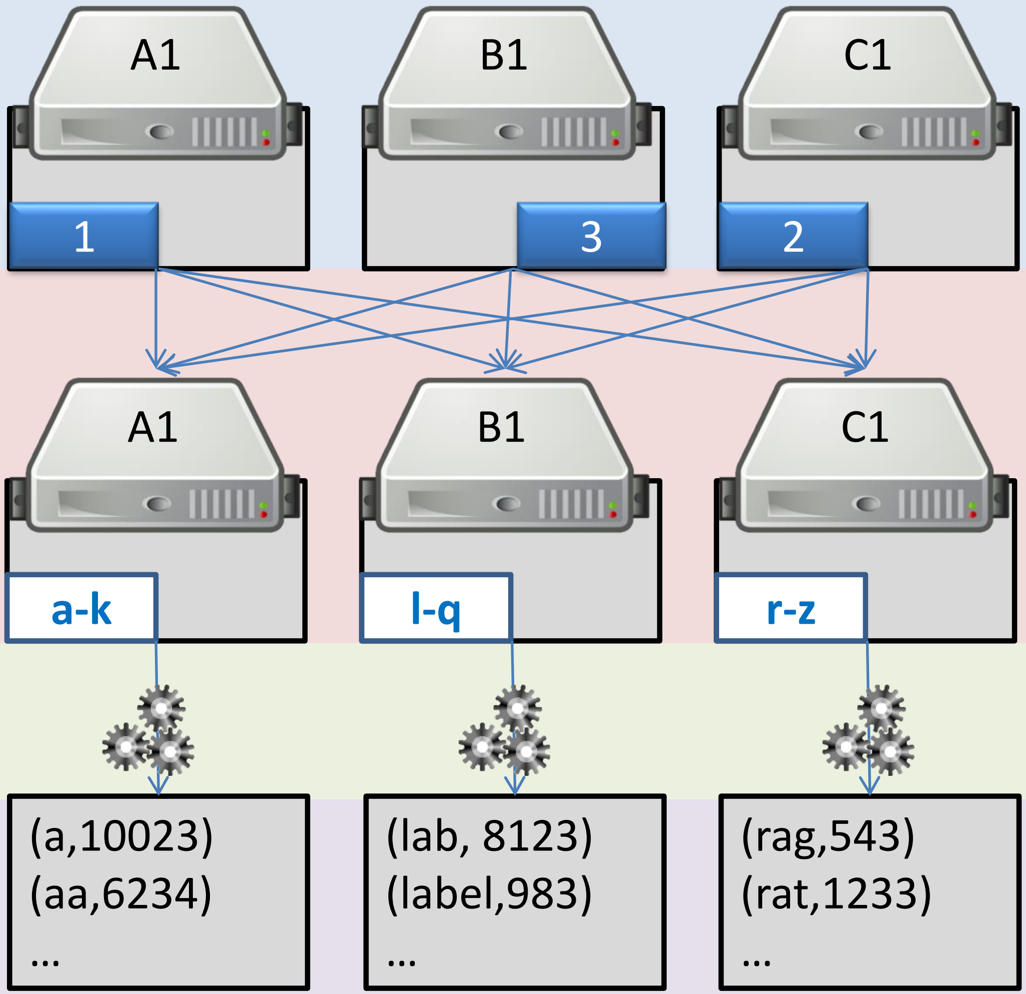
But if the data don't fit in memory (e.g., 4-grams)?

And how to do that merge (sum counts for word *w* across machines)?

Count parts on-disk on different machines and merge?

Again, how to do that merge?

Distributed word count



Input

File on Distr. File System

Partition

Distr. Sort/Count

Output

File on Distr. File System

Better partitioning? $\mathrm{HASH}(\mathtt{w})\%m$

Distributed word count

Can we abstract any general framework?

Define input as a set of key value pairs $I \subseteq T_{IK} \times T_{IV}$

For example, $I = \{(1, \texttt{"soy una linea"}), (2, \texttt{"soy otra linea"})\}$

T_{IK} is the set of all `int`, T_{IV} is the set of all `string`

Define map as a function $I \to 2^M$ where $M \subseteq T_{MK} \times T_{MV}$

For example, $\text{map}(1, \texttt{"soy una linea"}) := \{(\texttt{"soy"}, 1), (\texttt{"una"}, 1), (\texttt{"linea"}, 1)\}$

T_{MK} is the set of all `string`, T_{MV} is the set of all `int`

Define reduce as a function $2^M \to 2^R$ where $R \subseteq T_{RK} \times T_{RV}$

For example, $\text{reduce}\big(\{(\texttt{"soy"}, 1), (\texttt{"soy"}, 1)\}\big) := \{(\texttt{"soy"}, 2)\}$

T_{RK} is the set of all `string`, T_{RV} is the set of all `int`

Can we be more specific for reduce?

MapReduce

Can we abstract any general framework?

Define input as a set of key value pairs $I \subseteq T_{IK} \times T_{IV}$

For example, $I = \{(1, \texttt{"soy una linea"}), (2, \texttt{"soy otra linea"})\}$
T_{IK} is the set of all `int`, T_{IV} is the set of all `string`

Define map as a function $I \rightarrow 2^{M}$ where $M \subseteq T_{MK} \times T_{MV}$

For example, map$(1, \texttt{"soy una linea"}) := \{(\texttt{"soy"}, 1), (\texttt{"una"}, 1), (\texttt{"linea"}, 1)\}$
T_{MK} is the set of all `string`, T_{MV} is the set of all `int`

Define reduce as a function $\boxed{T_{MK} \times 2^{T_{MV}} \rightarrow 2^{R}}$ where $R \subseteq T_{RK} \times T_{RV}$

For example, reduce$\boxed{(\texttt{"soy"}, \{1, 1\})} := \{(\texttt{"soy"}, 2)\}$
T_{RK} is the set of all `string`, T_{RV} is the set of all `int`

In general, we must assume bags/multisets (sets with duplicates)

MapReduce: Main Idea

Can we abstract any general framework?

Given I

... compute map over all $i \in I$

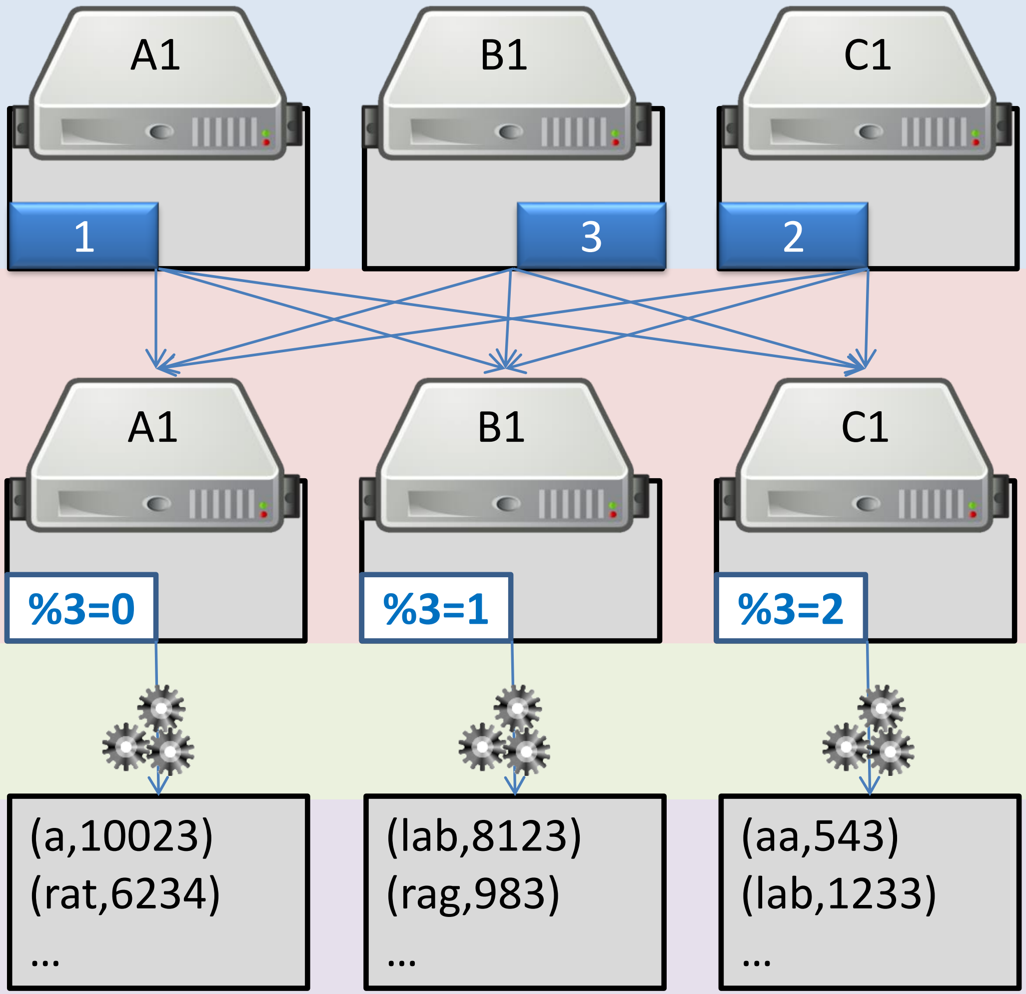
... group resulting set by map key

... apply reduce over groups

← But how to implement this part in a distributed system

1. Partition by map key
2. Sort (in parallel) by map key
3. Apply reduce / Profit

MapReduce: Word count



Input

File on Distr. File System

Map

Partition

Distr. Sort

Reduce

Output

File on Distr. File System

MapReduce (in more detail)

1. Input: Read from the cluster (e.g., a DFS)
   - Chunks raw data for mappers
   - Maps raw data to initial (key_{in},$value_{in}$) pairs

What might Input contain in the word-count case?

2. Map: For each (key_{in},$value_{in}$) pair, generate zero-to-many (key_{map},$value_{map}$) pairs
   - key_{in} /$value_{in}$ can be diff. type to key_{map} /$value_{map}$

What might Map do in the word-count case?

MapReduce (in more detail)

3. Partition: Assign sets of key_{map} values to reducer machines

   How might Partition work in the word-count case?

4. Shuffle: Data are moved from mappers to reducers (e.g., using DFS)

5. Comparison/Sort: Each reducer sorts the data by key using a comparison function
   - *Sort is taken care of by the framework*

MapReduce (in more detail)

6. Reduce: For each key_{map}, takes the bag of $value_{map}$ entries with that key, and produces zero-to-many outputs, i.e., $(key_{reduce}, value_{reduce})$ pairs

   How might Reduce work in the word-count case?

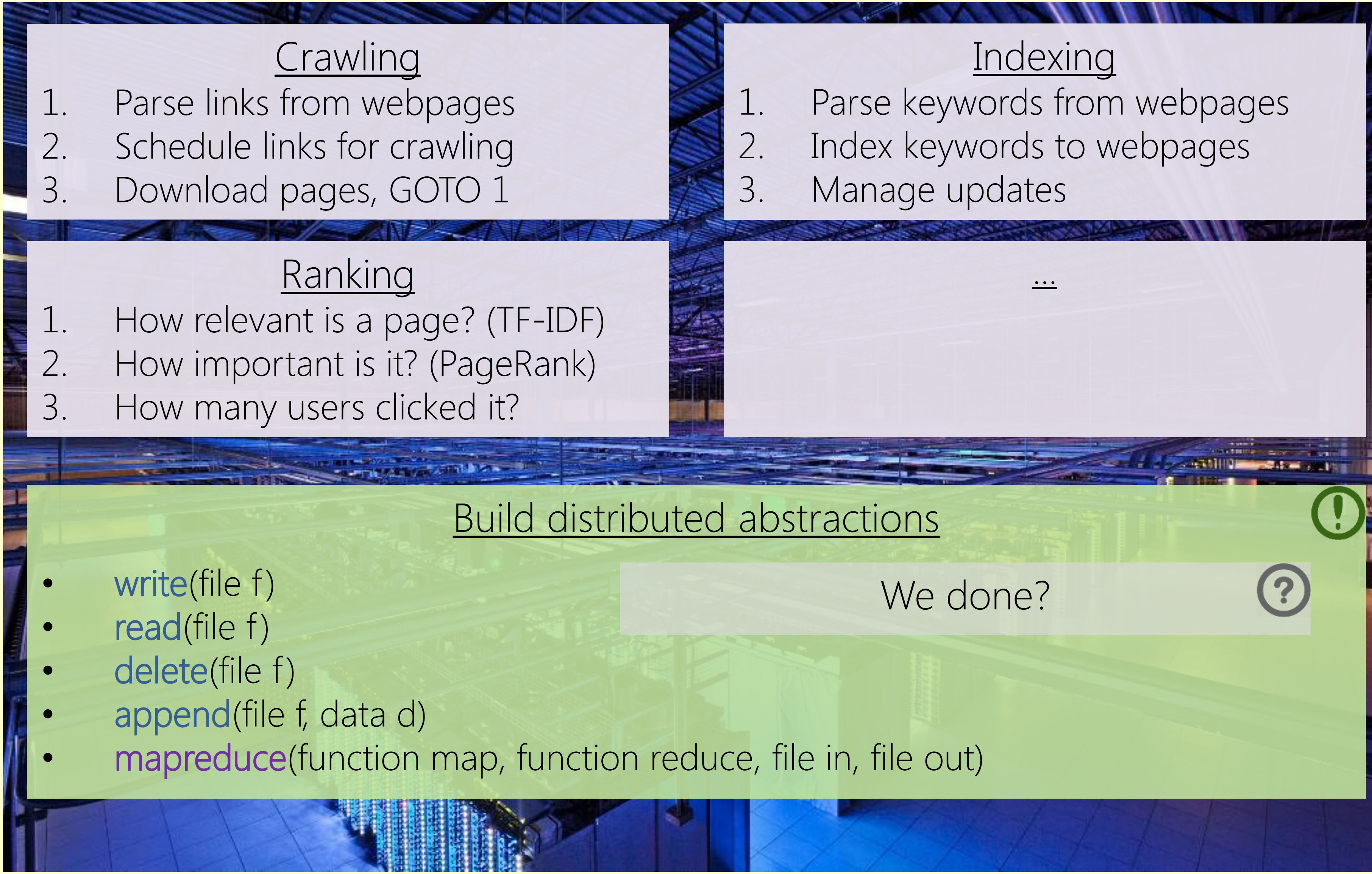
7. Output: Writes the results from the reducers to the distributed file system

MapReduce: Word count pseudo-code

```
function map(String name, String document):
  // name: document name
  // document: document contents
  for each word w in document:
    emit (w, 1)

function reduce(String word, Iterator partialCounts):
  // word: a word
  // partialCounts: a list of aggregated partial counts
  sum = 0
  for each pc in partialCounts:
    sum += ParseInt(pc)
  emit (word, sum)
```

Implementing on thousands of machines



<u>Crawling</u>

1. Parse links from webpages
2. Schedule links for crawling
3. Download pages, GOTO 1

<u>Indexing</u>

1. Parse keywords from webpages
2. Index keywords to webpages
3. Manage updates

<u>Ranking</u>

1. How relevant is a page? (TF-IDF)
2. How important is it? (PageRank)
3. How many users clicked it?

<u>...</u>

<u>Build distributed abstractions</u>

- write(file f)
- read(file f)
- delete(file f)
- append(file f, data d)
- mapreduce(function map, function reduce, file in, file out)

We done?

More complex example?

1: ReceiptItems.tsv

Receipt ID	Item ID
R1401	I306
R1401	I306
R1401	I504
R1402	I007
R1402	I306
R1403	I306
R1403	I504
...	...

2: ReceiptTimes.tsv

Receipt ID	Time
R1403	19:00
R1401	18:59
R1402	19:01
...	...

3: ItemDetails.tsv

Item ID	Name	Price ($)
I306	Zanahoria 500g	500
I504	CocaCola 3L	1400
I007	Comfort	1200
...	...	

Compute total sales per hour?

Output

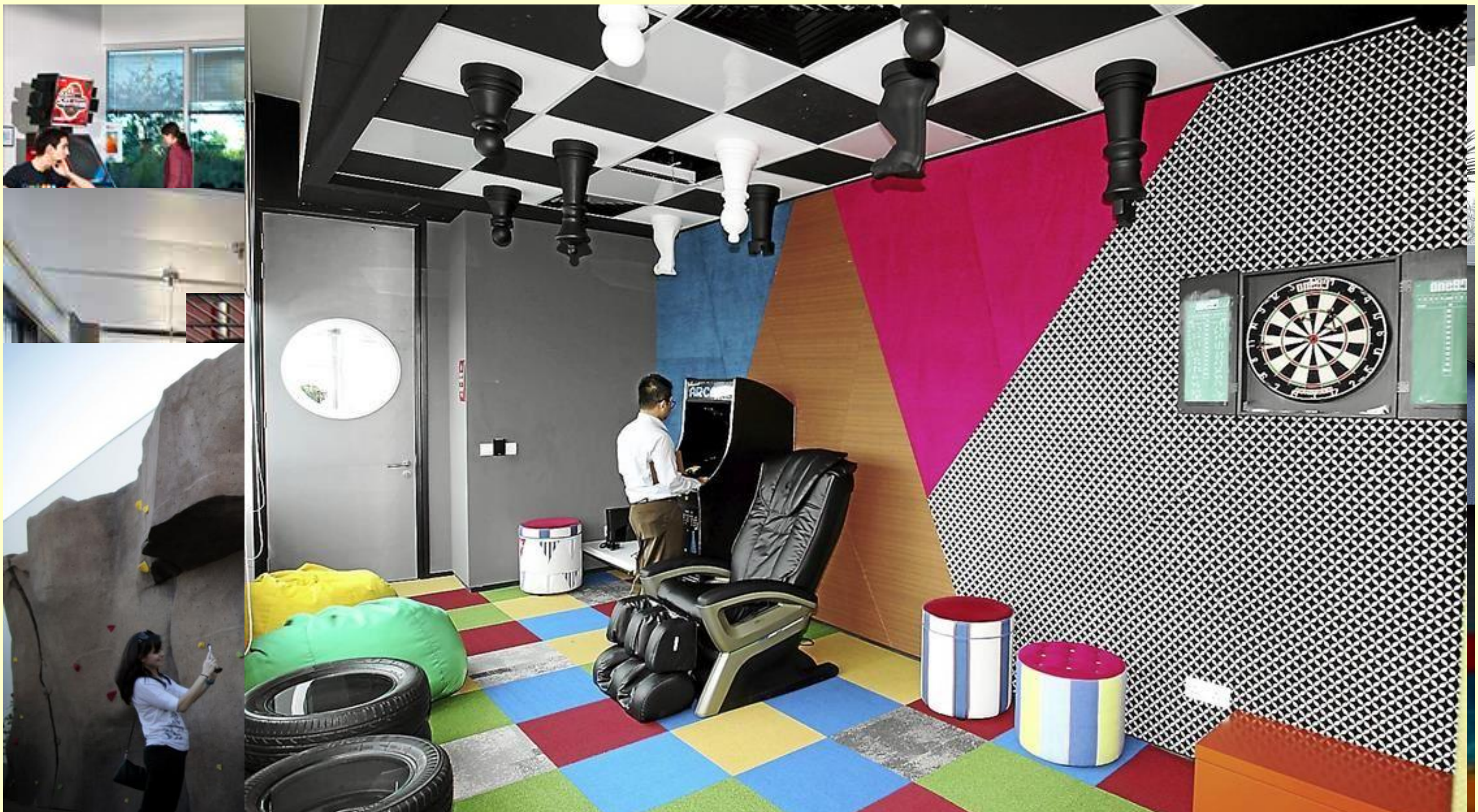
Hour	Total
...	...
18:00–18:59	$2400
19:00–19:59	$3600
...	...

MapReduce: Benefits for Programmers

- Takes care of low-level implementation:
  - Easy to handle inputs and output
  - No need to handle network communication
  - No need to write sorts or joins
- Abstracts machines (transparency)
  - Fault tolerance (through heart-beats)
  - Abstracts physical locations
  - Add / remove machines
  - Load balancing

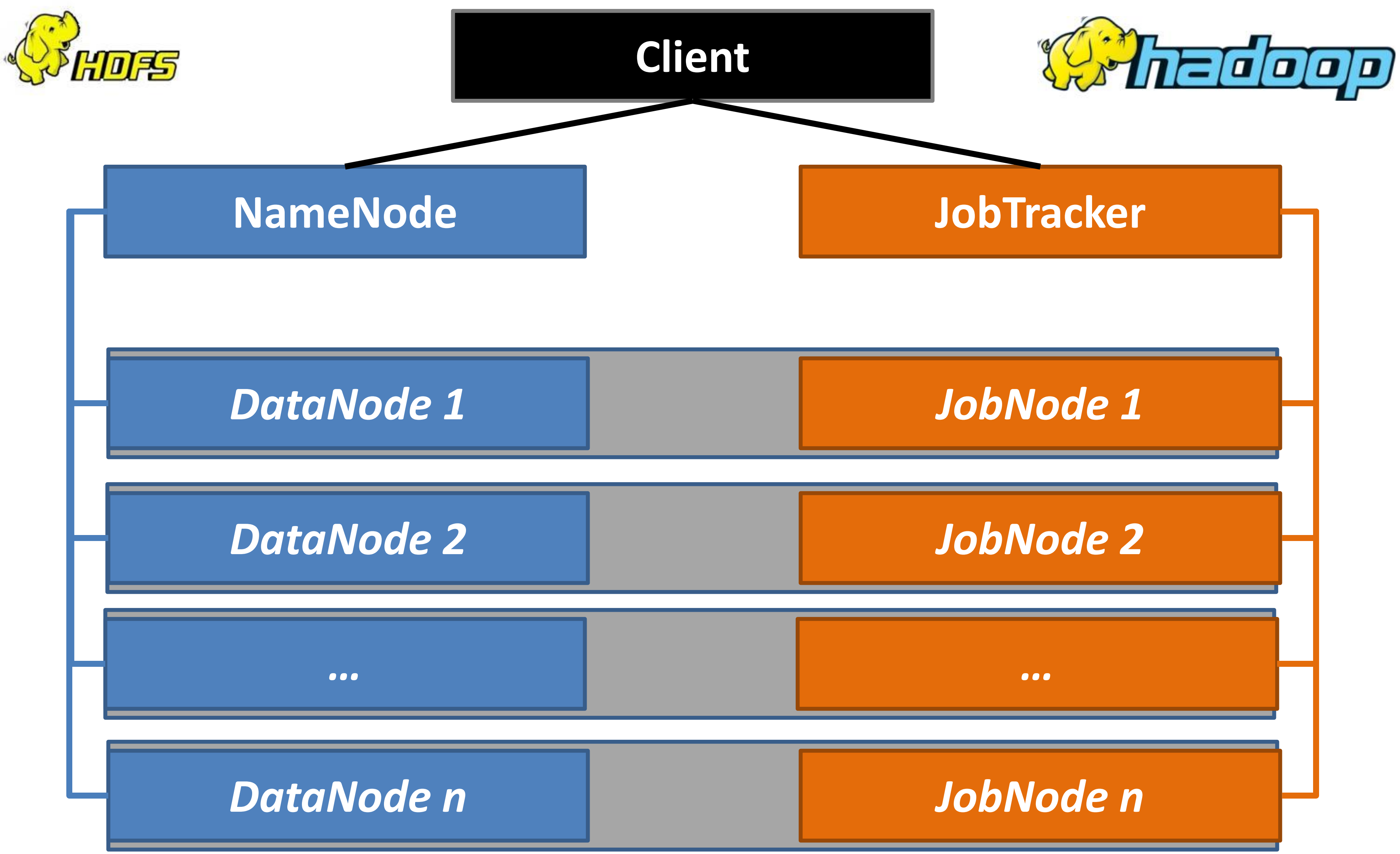
MapReduce: Benefits for Programmers

(Time for more important things)

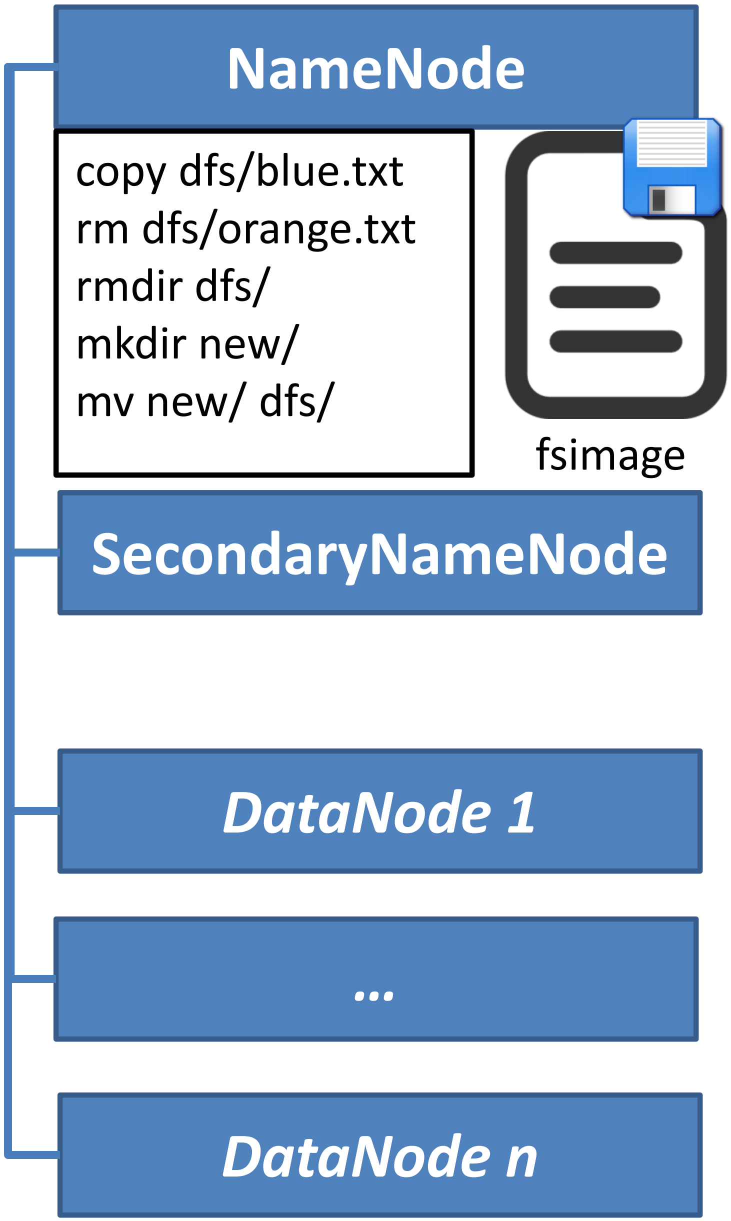


HADOOP OVERVIEW

HDFS / Hadoop Architecture



HDFS: Traditional / SPOF



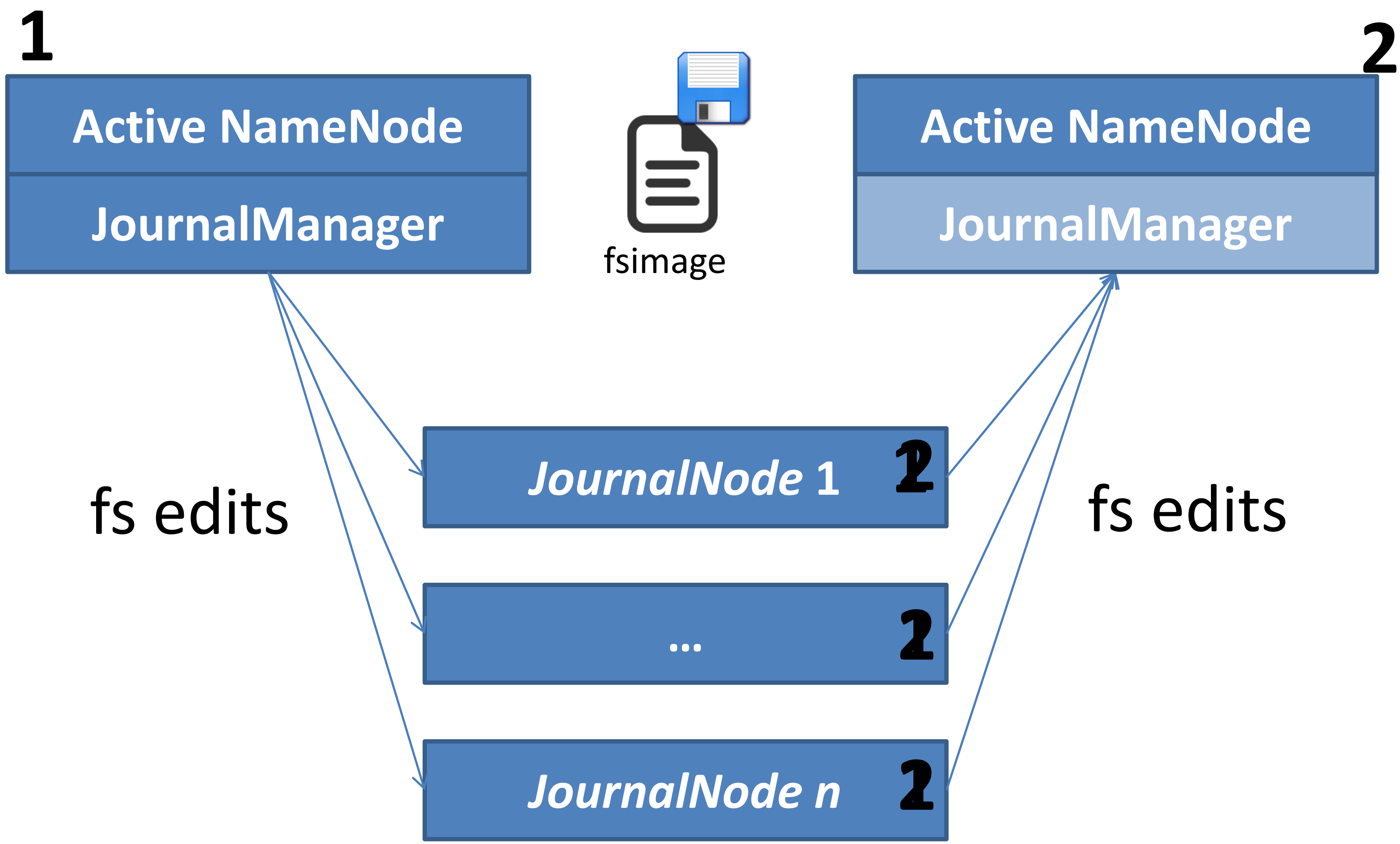
1. NameNode appends edits to log file
2. SecondaryNameNode copies log file and image, makes <u>checkpoint</u>, copies image back
3. NameNode loads image on start-up and makes remaining edits

SecondaryNameNode <u>not</u> a backup NameNode

What is the secondary name-node?

- Name-node quickly logs all file-system actions in a sequential (but messy) way
- Secondary name-node keeps the main `fsimage` file up-to-date based on logs
- When the primary name-node boots back up, it loads the `fsimage` file and applies the remaining log to it
- Hence secondary name-node helps make boot-ups faster, helps keep file system image up-to-date and takes load away from primary

Hadoop: High Availability



(REFERENCE MATERIAL FOR LAB)

PROGRAMMING WITH HADOOP

1. Input/Output (cmd)

> hdfs dfs

cluster.dcc.uchile.cl - PuTTY

```
hadoop@cluster-m:~/hadoop-2.3.0/logs$ hdfs dfs
Usage: hadoop fs [generic options]
        [-appendToFile <localsrc> ... <dst>]
        [-cat [-ignoreCrc] <src> ...]
        [-checksum <src> ...]
        [-chgrp [-R] GROUP PATH...]
        [-chmod [-R] <MODE[,MODE]... | OCTALMODE> PATH...]
        [-chown [-R] [OWNER][:[GROUP]] PATH...]
        [-copyFromLocal [-f] [-p] <localsrc> ... <dst>]
        [-copyToLocal [-p] [-ignoreCrc] [-crc] <src> ... <localdst>]
        [-count [-q] <path> ...]
        [-cp [-f] [-p] <src> ... <dst>]
        [-createSnapshot <snapshotDir> [<snapshotName>]]
        [-deleteSnapshot <snapshotDir> <snapshotName>]
        [-df [-h] [<path> ...]]
        [-du [-s] [-h] <path> ...]
        [-expunge]
        [-get [-p] [-ignoreCrc] [-crc] <src> ... <localdst>]
        [-getmerge [-nl] <src> <localdst>]
        [-help [cmd ...]]
        [-ls [-d] [-h] [-R] [<path> ...]]
        [-mkdir [-p] <path> ...]
        [-moveFromLocal <localsrc> ... <dst>]
        [-moveToLocal <src> <localdst>]
        [-mv <src> ... <dst>]
        [-put [-f] [-p] <localsrc> ... <dst>]
        [-renameSnapshot <snapshotDir> <oldName> <newName>]
        [-rm [-f] [-r|-R] [-skipTrash] <src> ...]
        [-rmdir [--ignore-fail-on-non-empty] <dir> ...]
        [-setrep [-R] [-w] <rep> <path> ...]
        [-stat [format] <path> ...]
```

1. Input/Output (Java)

```
public class HDFSHelloWorld {

    public static final String theFilename = "hello.txt";
    public static final String message = "Hello, world!\n";

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

        Configuration conf = new Configuration();
        FileSystem fs = FileSystem.get(conf);

        Path filenamePath = new Path(theFilename);

        try {
            if (fs.exists(filenamePath)) {
                // remove the file first
                fs.delete(filenamePath, false);
            }

            FSDataOutputStream out = fs.create(filenamePath);
            out.writeUTF(message);
            out.close();

            FSDataInputStream in = fs.open(filenamePath);
            String messageIn = in.readUTF();
            System.out.print(messageIn);
            in.close();
        } catch (IOException ioe) {
            System.err.println("IOException during operation: " + ioe.toString());
            System.exit(1);
        }
    }
}
```

Creates a file system for default configuration

Check if the file exists; if so delete

Create file and write a message

Open and read back

1. Input (Java)

InputFormat:	Description:	Key:	Value:
TextInputFormat	Default format; reads lines of text files	The byte offset of the line	The line contents
KeyValueInputFormat	Parses lines into key, val pairs	Everything up to the first tab character	The remainder of the line
SequenceFileInputFormat	A Hadoop-specific high-performance binary format	user-defined	user-defined

2. Map

Mapper<InputKeyType,
InputValueType,
MapKeyType,
MapValueType>

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

    private final IntWritable one = new IntWritable(1);
    private Text paperTitle = new Text();


    /**
     * @throws InterruptedException
     *
     */
     @Override
    public void map(Object key, Text value, Context output)
                    throws IOException, InterruptedException {
        String line = value.toString();
        String[] paperCitedByPaper = line.split(SPLIT_REGEX);
        paperTitle.set(paperCitedByPaper[0]);
        output.write(paperTitle, one);
    }
}
```

(input) key: file offset.
(input) value: line of the file.
context: handles output and logging.

Emit output

(Writable *for values*)

```
package ejemplo;

import java.io.DataInput;
import java.io.DataOutput;
import java.io.IOException;

import org.apache.hadoop.io.Writable;

public class WritableCitation implements Writable {
    public String citingPaper;
    public String citingVenue;
    public int mentions;

    public WritableCitation(String citingPaper, String citingVenue, int mentions) {
        this.citingPaper = citingPaper;
        this.citingVenue = citingVenue;
        this.mentions = mentions;
    }

    public void write(DataOutput out) throws IOException {
        out.writeUTF(citingPaper);
        out.writeUTF(citingVenue);
        out.writeInt(mentions);
    }

    public void readFields(DataInput in) throws IOException {
        citingPaper = in.readUTF();
        citingVenue = in.readUTF();
        mentions = in.readInt();
    }

    public String toString() {
        return citingPaper +"\t" + citingVenue + "\t" + mentions;
    }
}
```

Same order

(not needed in the running example)

(WritableComparable *for keys/values*)

```
public class WritableComparableCitation implements WritableComparable<WritableComparableCitation> {
    public String citingPaper;
    public String citingVenue;
    public int mentions;

    public WritableComparableCitation(String citingPaper, String citingVenue, int mentions) {...
    public void write(DataOutput out) throws IOException {...
    public void readFields(DataInput in) throws IOException {...
    public String toString() {...

    public int compareTo(WritableComparableCitation other) {
        int comp = citingPaper.compareTo(other.citingPaper);
        if(comp==0){
            comp = citingVenue.compareTo(other.citingVenue);
            if(comp == 0){
                comp = Integer.compare(mentions, other.mentions);
            }
        }
        return comp;
    }

    public boolean equals(Object o) {
        if(o==null) return false;
        if(o==this) return true;
        if (!(o instanceof WritableComparableCitation)) return false;
        WritableComparableCitation wcp = (WritableComparableCitation)o;
        return citingPaper.equals(wcp.citingPaper) && this.citingVenue.equals(wcp.citingVenue)
                && this.mentions == wcp.mentions;
    }

    public int hashCode() {
        return citingPaper.hashCode() ^ citingVenue.hashCode() ^ mentions;
    }
}
```

New Interface

Same as before

Needed to sort keys

Needed for default partition function

(not needed in the running example)

3. Partition

PartitionerInterface

```
package ejemplo;

import org.apache.hadoop.mapred.JobConf;

public class PartitionCites<E> implements Partitioner<WritableComparableCitation, E> {

    @Override
    public int getPartition(WritableComparableCitation key, E val, int machines) {
        return Math.abs(key.hashCode() % machines);
    }

    @Override
    public void configure(JobConf arg0) {
    }

}
```

(This happens to be the default partition method!)

(not needed in the running example)

4. Shuffle

5. Sort/Comparison

```java
public class WritableComparableCitation implements WritableComparable<WritableComparableCitation> {
    public String citingPaper;
    public String citingVenue;
    public int mentions;

    public WritableComparableCitation(String citingPaper, String citingVenue, int mentions) {
    public void write(DataOutput out) throws IOException {
    public void readFields(DataInput in) throws IOException {
    public String toString() {

    public int compareTo(WritableComparableCitation other) {
        int comp = citingPaper.compareTo(other.citingPaper);
        if(comp==0){
            comp = citingVenue.compareTo(other.citingVenue);
            if(comp == 0){
                comp = Integer.compare(mentions, other.mentions);
            }
        }
        return comp;
    }

    public boolean equals(Object o) {
        if(o==null) return false;
        if(o==this) return true;
        if (!(o instanceof WritableComparableCitation)) return false;
        WritableComparableCitation wcp = (WritableComparableCitation)o;
        return citingPaper.equals(wcp.citingPaper) && this.citingVenue.equals(wcp.citingVenue)
                && this.mentions == wcp.mentions;
    }

    public int hashCode() {
        return citingPaper.hashCode() ^ citingVenue.hashCode() ^ mentions;
    }
}
```

Methods in WritableComparator

(not needed in the running example)

6. Reduce

Reducer<MapKey, MapValue, OutputKey, OutputValue>

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

    /**
     * @throws InterruptedException
     */
    @Override
    public void reduce(Text key, Iterable<IntWritable> values,
            Context output) throws IOException, InterruptedException {
        int sum = 0;
        for(IntWritable value: values) {
            sum += value.get();
        }
        output.getCounter("citations", key.toString().substring(0, 1)).increment(1);;
        output.write(key, new IntWritable(sum));
    }
}
```

key: as emitted from map

values: iterator over all values for that key

context for output

Write to output

7. Output / Input (Java)

```
public class HDFSHelloWorld {

    public static final String theFilename = "hello.txt";
    public static final String message = "Hello, world!\n";

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

        Configuration conf = new Configuration();
        FileSystem fs = FileSystem.get(conf);

        Path filenamePath = new Path(theFilename);

        try {
            if (fs.exists(filenamePath)) {
                // remove the file first
                fs.delete(filenamePath, false);
            }

            FSDataOutputStream out = fs.create(filenamePath);
            out.writeUTF(message);
            out.close();

            FSDataInputStream in = fs.open(filenamePath);
            String messageIn = in.readUTF();
            System.out.print(messageIn);
            in.close();
        } catch (IOException ioe) {
            System.err.println("IOException during operation: " + ioe.toString());
            System.exit(1);
        }
    }
}
```

Creates a file system for default configuration

Check if the file exists; if so delete

Create file and write a message

Open and read back

7. Output (Java)

OutputFormat:	Description
TextOutputFormat	Default; writes lines in "key \t value" form
SequenceFileOutputFormat	Writes binary files suitable for reading into subsequent MapReduce jobs
NullOutputFormat	Disregards its inputs

Control Flow

```java
public static void main(String[] args) throws Exception {
    Configuration conf = new Configuration();
    String[] otherArgs = new GenericOptionsParser(conf, args).getRemainingArgs();
    if (otherArgs.length != 2) {
        System.err.println("Usage: CitationCount <in> <out>");
        System.exit(2);
    }
    String inputLocation = otherArgs[0];
    String outputLocation = otherArgs[1];

    Job job = Job.getInstance(new Configuration());

    FileInputFormat.setInputPaths(job, new Path(inputLocation));
    FileOutputFormat.setOutputPath(job, new Path(outputLocation));

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

    job.setMapperClass(CitationCountMapper.class);
    job.setCombinerClass(CitationCountReducer.class);
    job.setReducerClass(CitationCountReducer.class);

    job.setJarByClass(CitationCount.class);
    job.waitForCompletion(true);
}
```

Create a JobClient, a JobConf and pass it the main class

Set input and output paths

Set the type of map and output keys and values in the configuration

Set the mapper class

Set the reducer class (and optionally "**combiner**")

Run and wait for job to complete.

More in Hadoop: Combiner

- Map-side "mini-reduction"
- Keeps a fixed-size buffer in memory
- Reduce within that buffer
  - e.g., count words in buffer
  - Lessens bandwidth needs
- In Hadoop: can simply use Reducer class ☺

More in Hadoop: Counters

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

    /**
     * @throws InterruptedException
     */
    @Override
    public void reduce(Text key, Iterable<IntWritable> values,
            Context output) throws IOException, InterruptedException {
        int sum = 0;
        for(IntWritable value: values) {
            sum += value.get();
        }
        output.getCounter("citations", key.toString().substring(0, 1)).increment(1);
        output.write(key, new IntWritable(sum));
    }
}
```

Context has a group of maps of counters

More in Hadoop: Chaining Jobs

- Sometimes we need to chain jobs
- In Hadoop, can pass a set of Jobs to the client
- **`x.addDependingJob(y)`**

More in Hadoop: Distributed Cache

- Some tasks need “global knowledge”
  - For example, a white-list of conference venues and journals that should be considered in the citation count
  - Typically small
- Use a distributed cache:
  - Makes data available locally to all nodes
  - (Use sparingly!!)

RECAP

Distributed File Systems

- Google File System (GFS)
  - **Master and Chunkslaves**
  - Replicated pipelined writes
  - Direct reads
  - Minimising master traffic
  - Fault-tolerance: self-healing
  - Rack awareness
  - Consistency and modifications
- Hadoop Distributed File System
  - **NameNode and DataNodes**

MapReduce

1. Input
2. Map
3. Partition
4. Shuffle
5. Comparison/Sort
6. Reduce
7. Output

MapReduce/GFS Revision

- GFS: distributed file system
  - Implemented as HDFS

- MapReduce: distributed processing framework
  - Implemented as Hadoop

Hadoop

- **`FileSystem`**
- **`Mapper<InputKey,InputValue,MapKey,MapValue>`**
- **`OutputCollector<OutputKey,OutputValue>`**
- **`Writable, WritableComparable<Key>`**
- **`Partitioner<KeyType,ValueType>`**
- **`Reducer<MapKey,MapValue,OutputKey,OutputValue>`**
- **`JobClient/JobConf`**

…

Questions?