CC5212-1
PROCESAMIENTO MASIVO DE DATOS
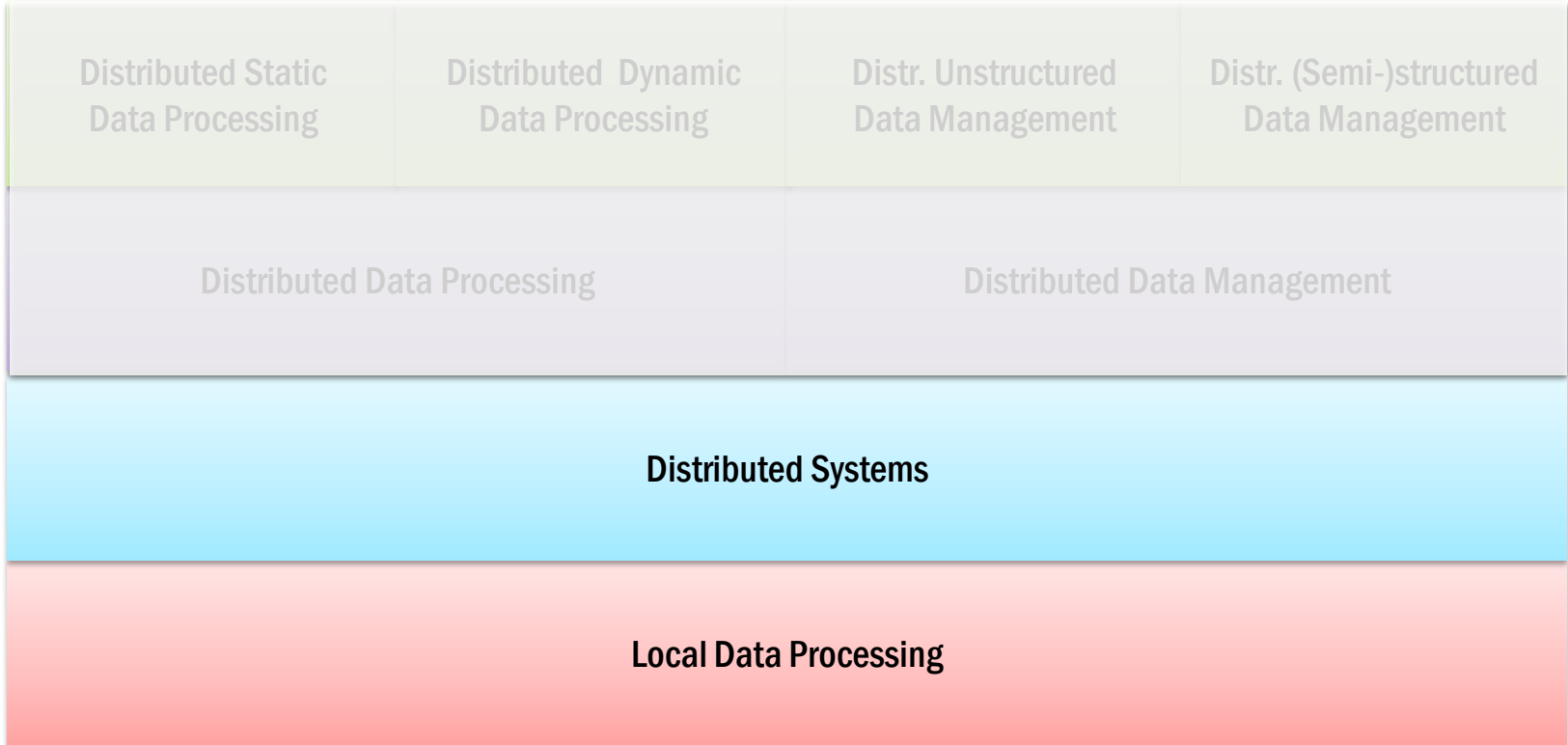OTOÑO 2023
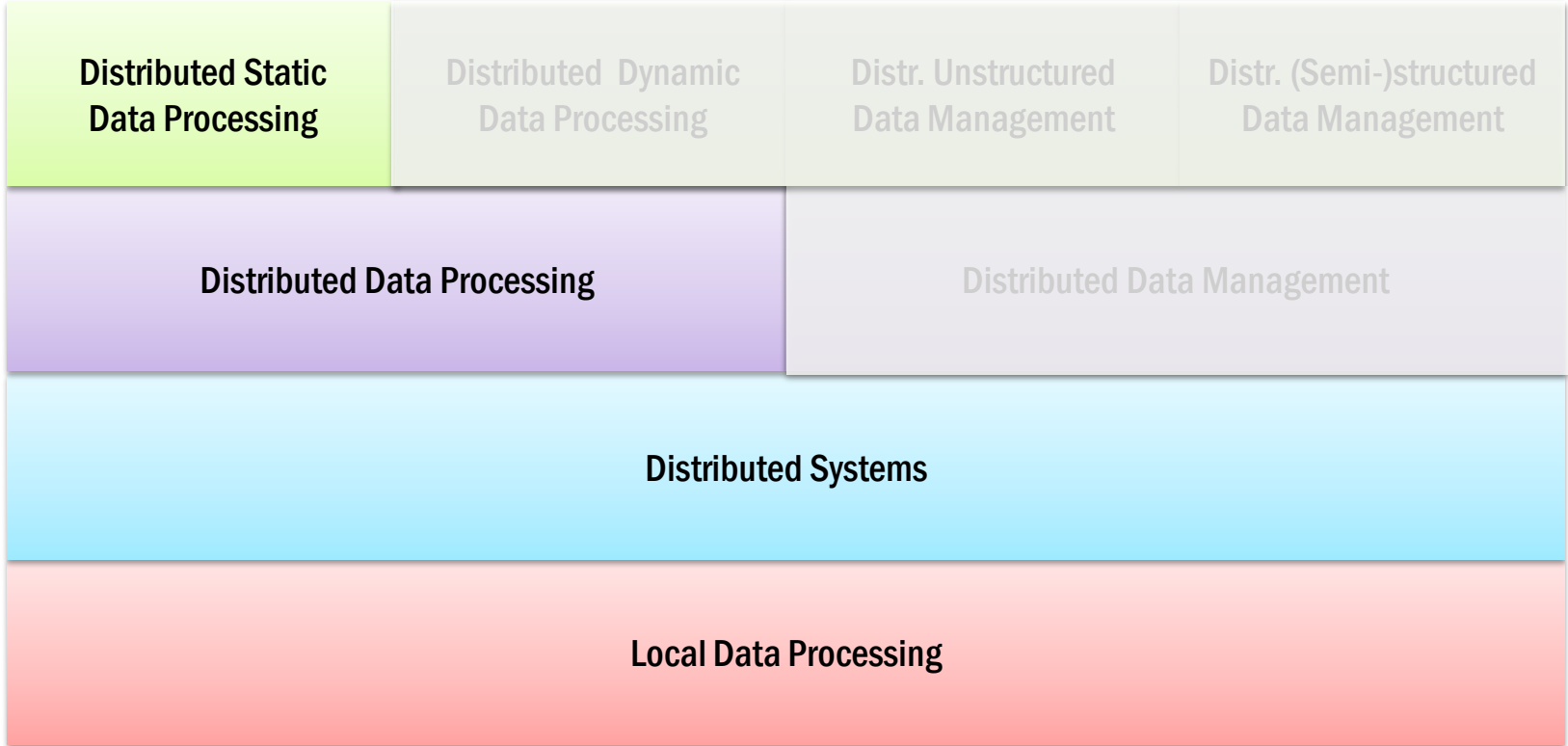
Lecture 3
DFS/HDFS + MapReduce/Hadoop

Aidan Hogan
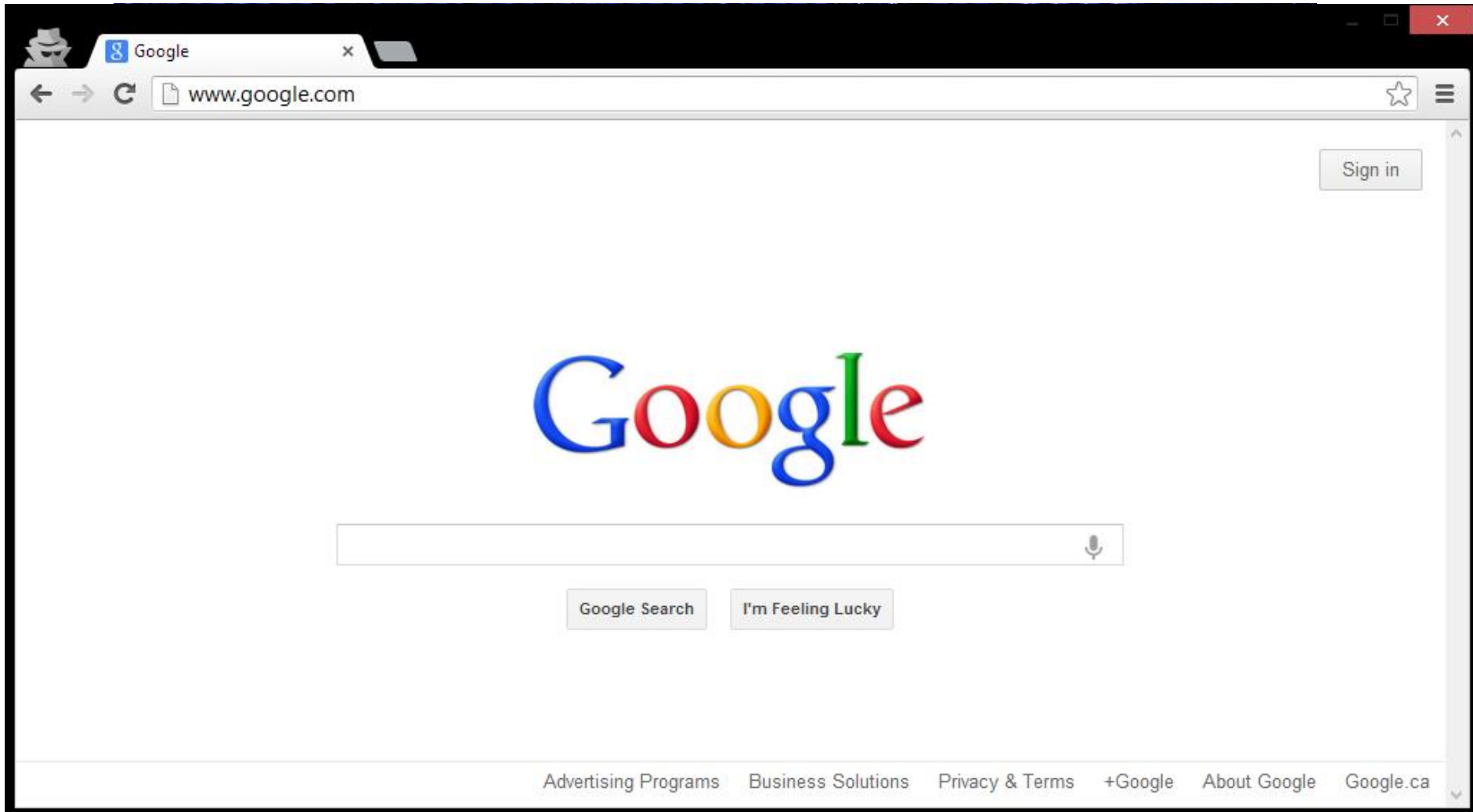
aidhog@gmail.com

Distributed Static Data Processing	Distributed Dynamic Data Processing	Distr. Unstructured Data Management	Distr. (Semi-)structured Data Management
Distributed Data Processing		Distributed Data Management	
Distributed Systems			
Local Data Processing			

Distributed Static Data Processing	Distributed Dynamic Data Processing	Distr. Unstructured Data Management	Distr. (Semi-)structured Data Management
Distributed Data Processing		Distributed Data Management	
Distributed Systems			
Local Data Processing			

Massive Data Processing in Google

Distributed Computing in Google

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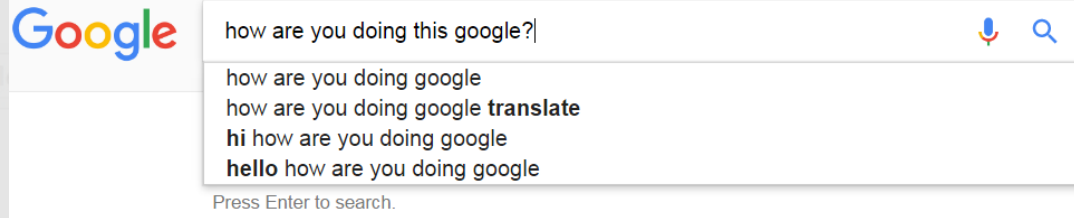
What processes/algorithms does Google need to implement Web search?

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

$\approx$ 100 PB / day

$\approx$ 2,000,000 Wiki / day

(2014, processed)

Google

≈ 100 PB / day

≈ 2,000,000 Wiki / day
(2014, processed)

3. Download pa...   3.   Mana...updates

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
    ... and then re-implement those again

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)

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.
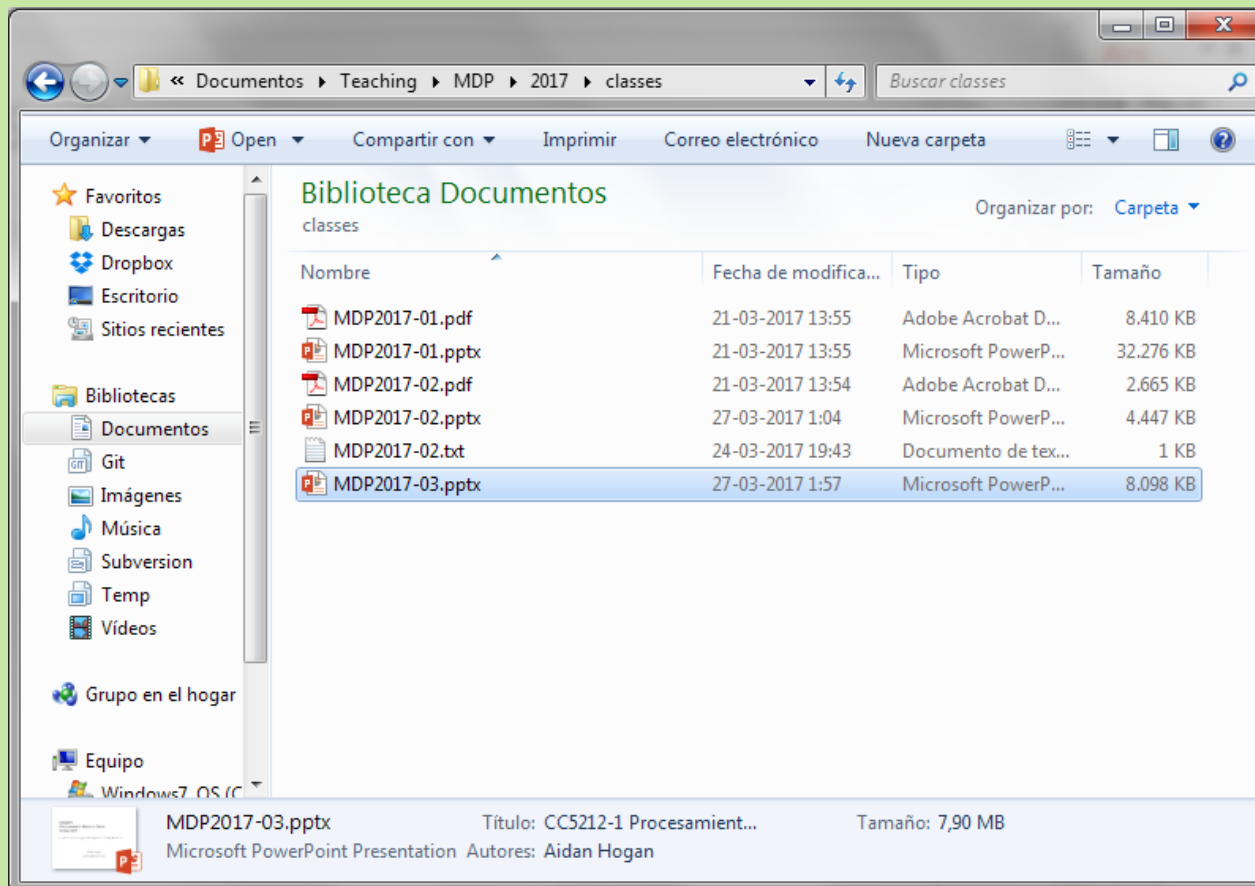
1. INTRODUCTION

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

Google File System

Google File System

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

Google File System

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

```
▲ 🗂 Bibliotecas
  ▲ 📄 Documentos
    ▲ 📑 Mis documentos
         Custom Office Templates
       ▷ Desktop_Old
         Mis archivos recibidos
       ▷ Misc
       ▷ My Games
       ▷ Paperwork
       ▷ Research
      ▲ Teaching
       ▷ BA
       ▷ Data
       ▷ Databases
       ▷ Dipl
         letter
         Marking
        ▲ MDP
         ▷ 2014
         ▷ 2015
         ▷ 2016
         ▲ 2017
              classes
            ▷ labs
```

Google File System

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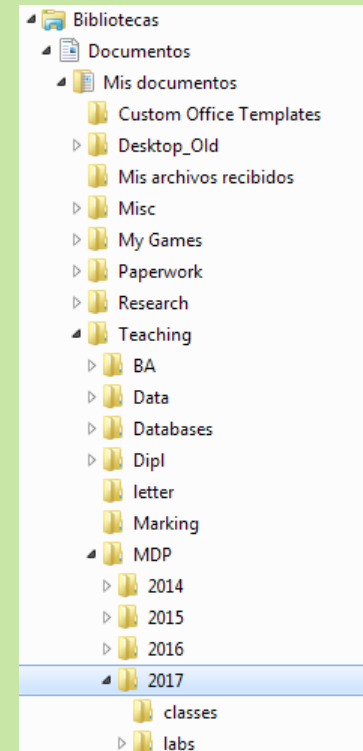   - Remembers location and sequence of chunks for a file
2. Organises a hierarchical directory structure
   - Tracks sub-directories and files in directories
3. Tracks file meta-data
   - File size, date created, date last modified
   - Ownership, permissions, locks

Nombre	Fecha de modifica...	Tipo	Tamaño	Fecha de creación
MDP2017-01.pdf	21-03-2017 13:55	Adobe Acrobat D...	8.410 KB	13-03-2017 16:21
MDP2017-01.pptx	21-03-2017 13:55	Microsoft PowerP...	32.276 KB	12-03-2017 22:40
MDP2017-02.pdf	21-03-2017 13:54	Adobe Acrobat D...	2.665 KB	21-03-2017 11:26
MDP2017-02.pptx	27-03-2017 1:04	Microsoft PowerP...	4.447 KB	20-03-2017 3:33
MDP2017-02.txt	24-03-2017 19:43	Documento de tex...	1 KB	24-03-2017 19:42
MDP2017-03.pptx	27-03-2017 2:19	Microsoft PowerP...	8.674 KB	27-03-2017 0:49

Ajustar todas las columnas

✓ Nombre
✓ Fecha de modificación
✓ Tipo
✓ Tamaño
✓ Fecha de creación
Ruta de acceso a la carpeta
Autores
Categorías
Etiquetas
Título

Más...

Google File System

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
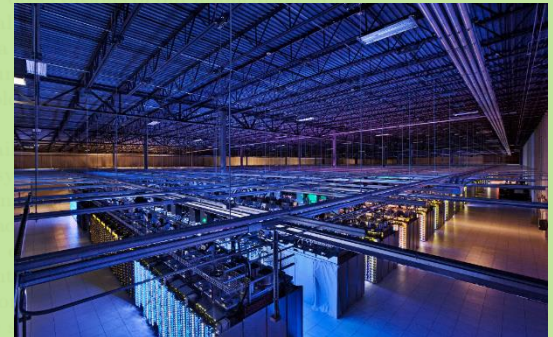4. Provides read/write/update/delete interface, etc.

Google File System

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:

Google File System

So which architecture do you think Google uses?

Client–Server?

Peer-To-Peer?

Google File System

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

So how should Google design its Distributed File System?

GFS: Architecture
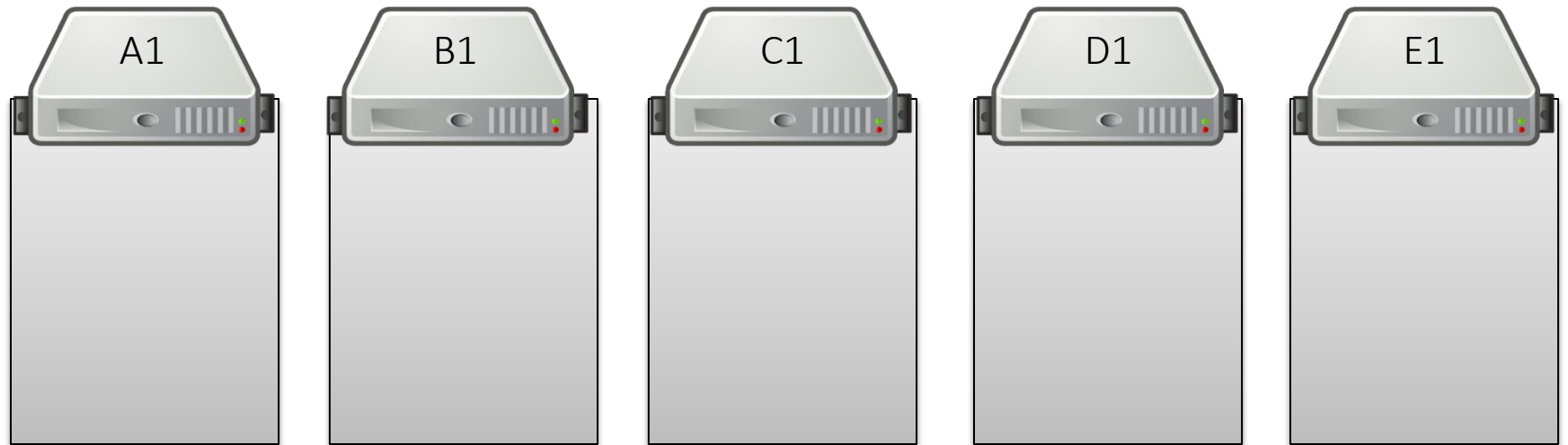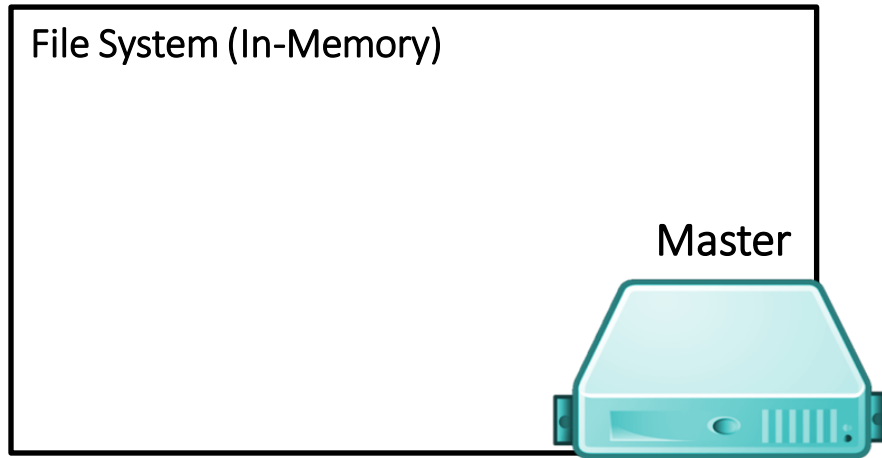
File System (In-Memory)

Master

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

A1   B1   C1   D1   E1
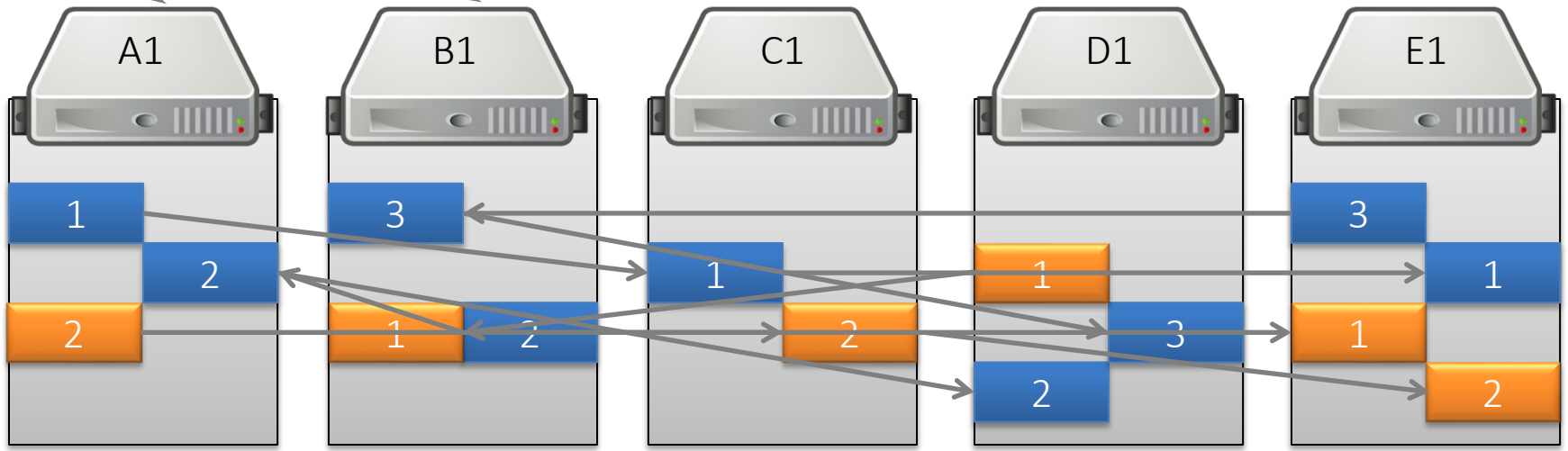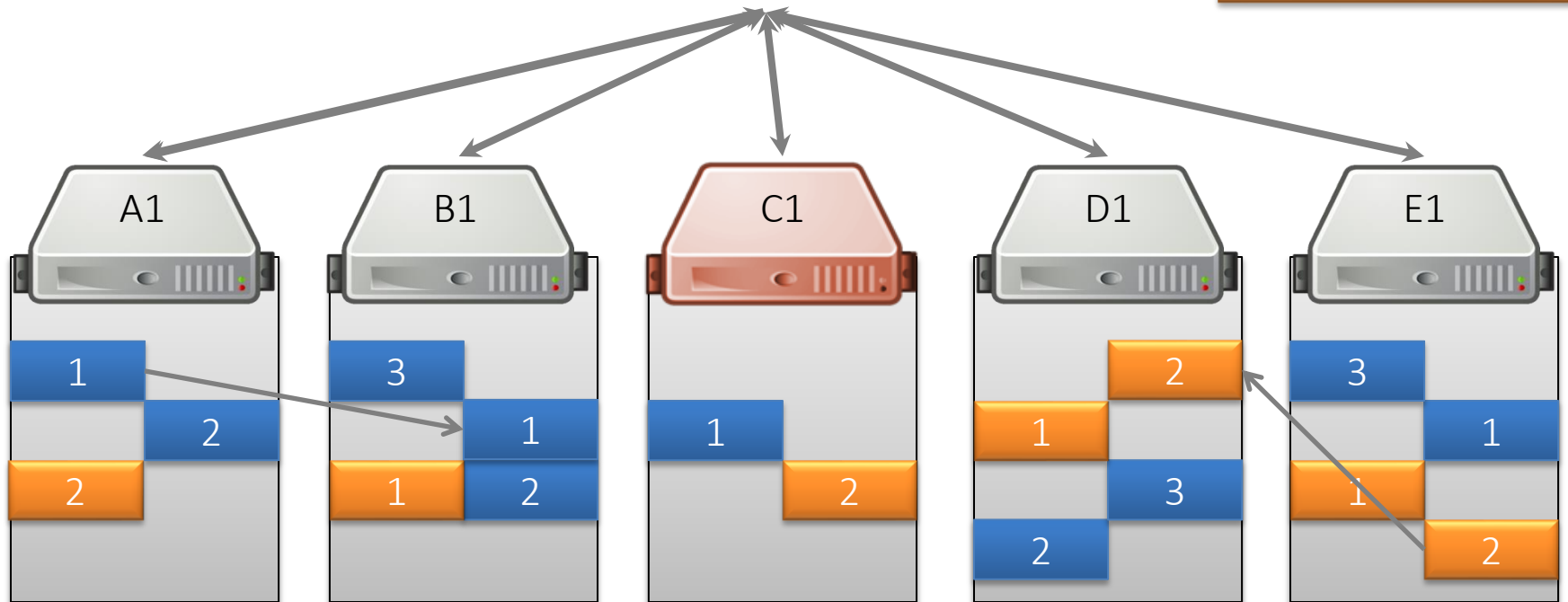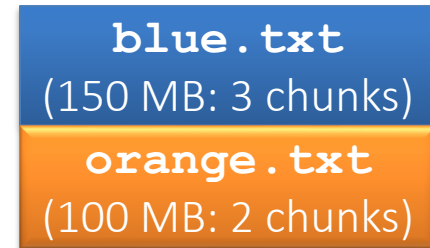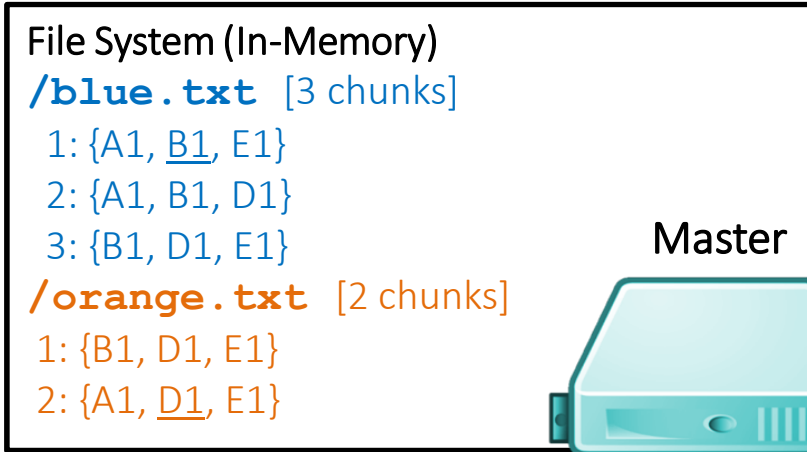
Chunk-servers

GFS: Direct Reads

File System (In-Memory)
/blue.txt [3 chunks]
  1: {A1, C1, E1}
  2: {A1, B1, D1}
  3: {B1, D1, E1}
/orange.txt [2 chunks]
  1: {B1, D1, E1}
  2: {A1, C1, E1}

Master

- 64 MB per chunk
- 64 bit label for each chunk
- Assume replica...

I'm looking for
/blue.txt

1   2   3

/blue.txt   [3 chunks]
1: {A1, C1, E1}
2: {A1, B1, D1}
3: {B1, D1, E1}

A1   B1   C1   D1   E1

Chunk-servers
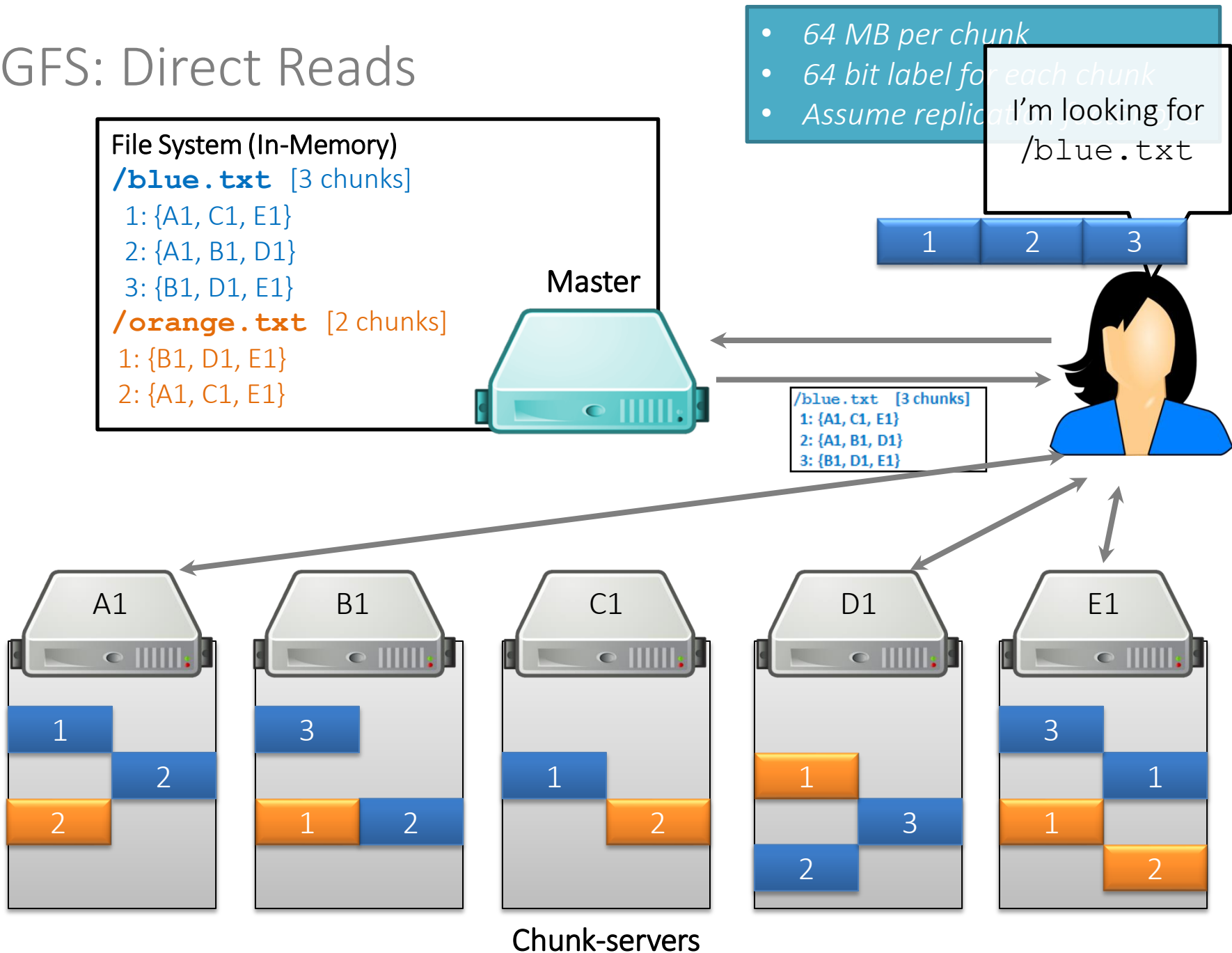
GFS: Primary Replicas

File System (In-Memory)
/blue.txt [3 chunks]
  1: {A1, C1, E1}
  2: {A1, B1, D1}
  3: {B1, D1, E1}
/orange.txt [2 chunks]
  1: {B1, D1, E1}
  2: {A1, C1, E1}

Master

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

I want to change block 2 of /blue.txt
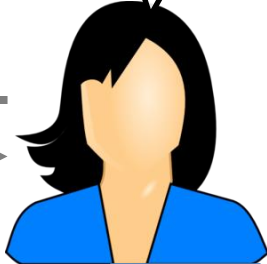
/blue.txt [3 chunks]
  2: {A1, B1, D1}

2

A1   B1   C1   D1   E1

2   2   2

1   3   1   1   3
2   2   3   1
2   1 2   2   1

Chunk-servers

GFS: Rack Awareness

GFS: Rack Awareness

GFS: Rack Awareness

Core Switch

Rack A Switch

Rack B Switch

A1
A2
A3
A4
A5

B1
B2
B3
B4
B5
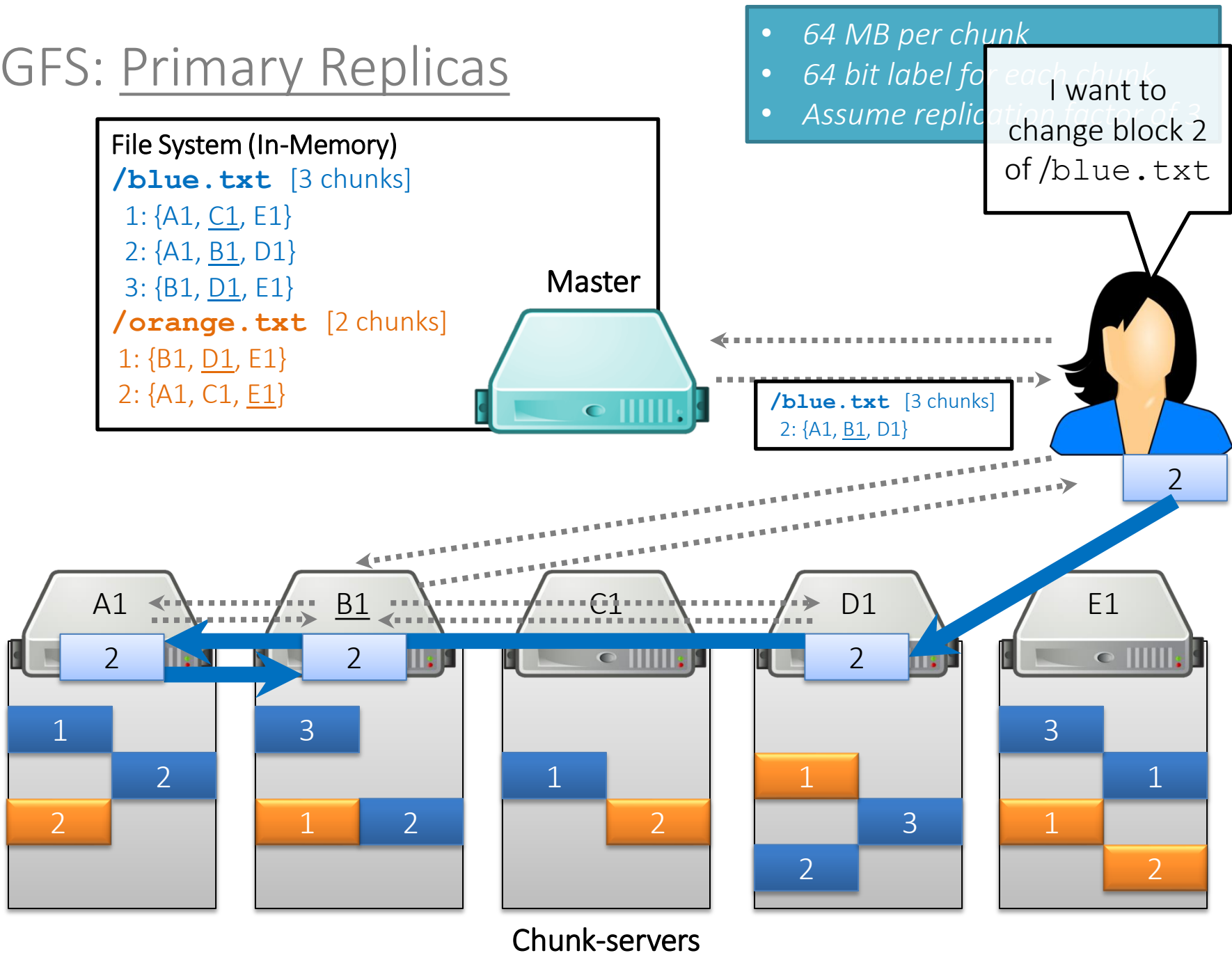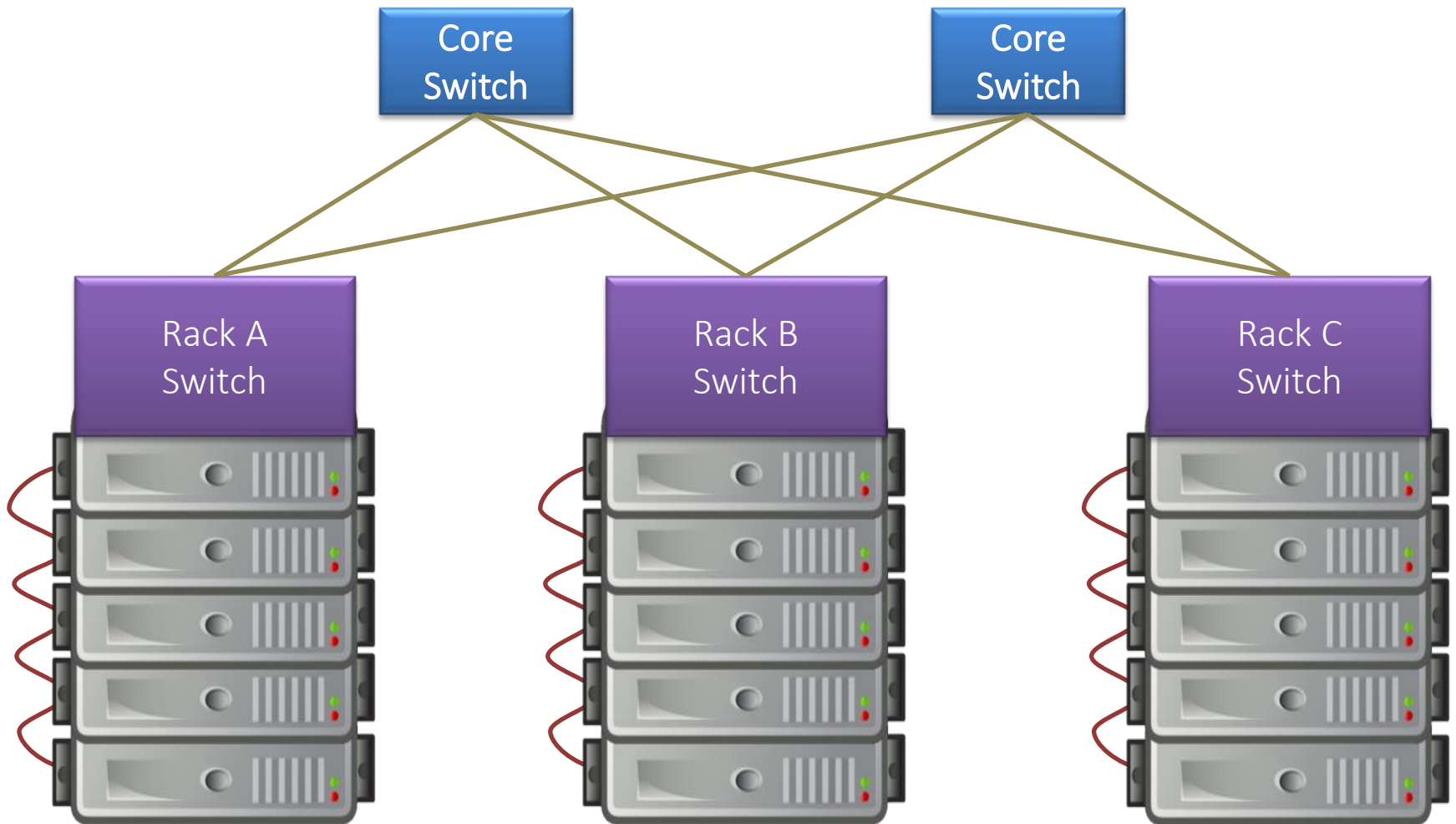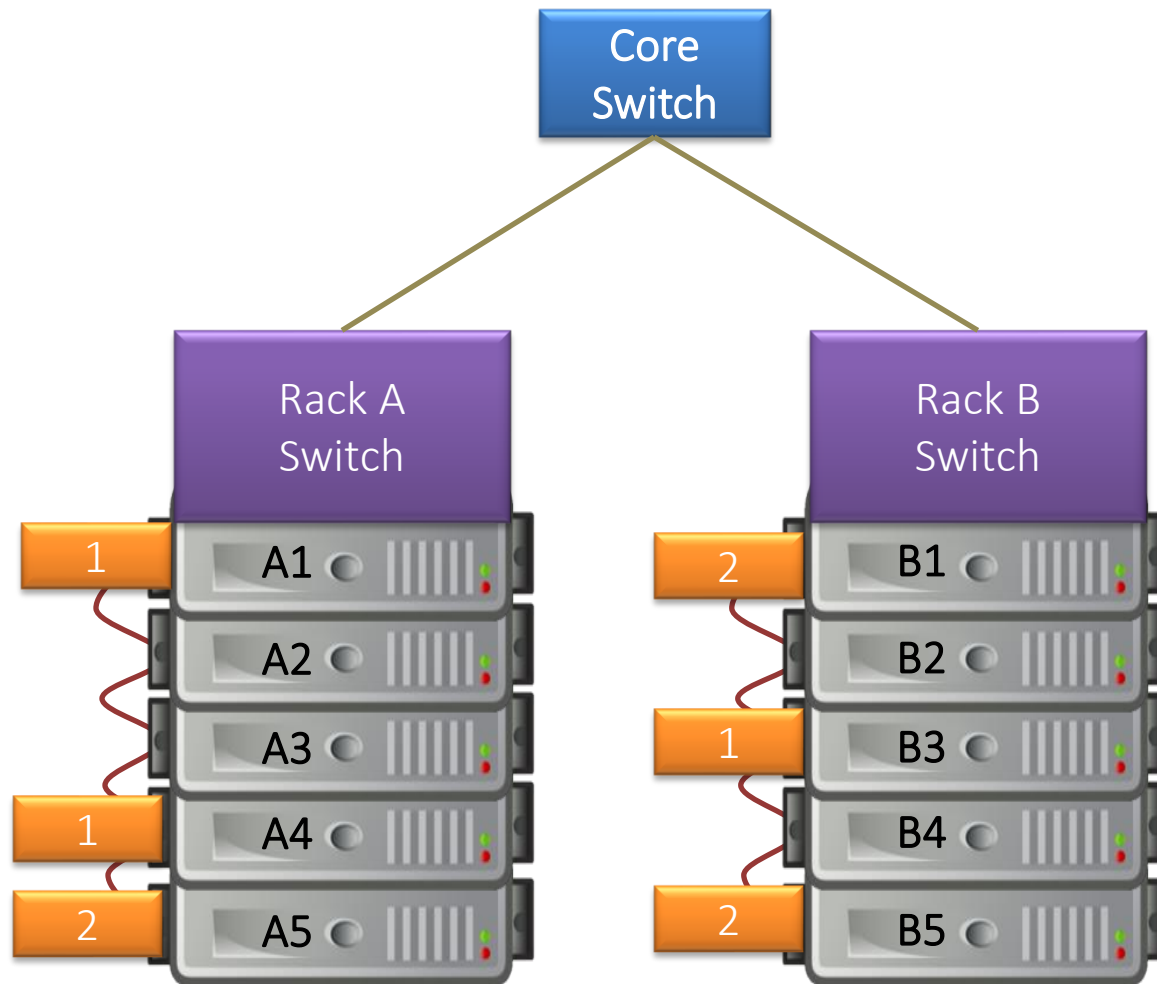
Files:
/orange.txt
1: {A1, A4, B3}
2: {A5, B1, B5}

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

GFS: 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 chunkserver 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 chunkserver
- Keep each chunk large (fewer chunks)

Hadoop Distributed File System

- Open source version of GFS

- HDFS-to-GFS translation guide …
  - Data-node = Chunkserver
  - Name-node = Master

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

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)   HDFS/GFS
- read(file f)
- delete(file f)
- append(file f, data d)

We done?

Implementing on thousands of machines
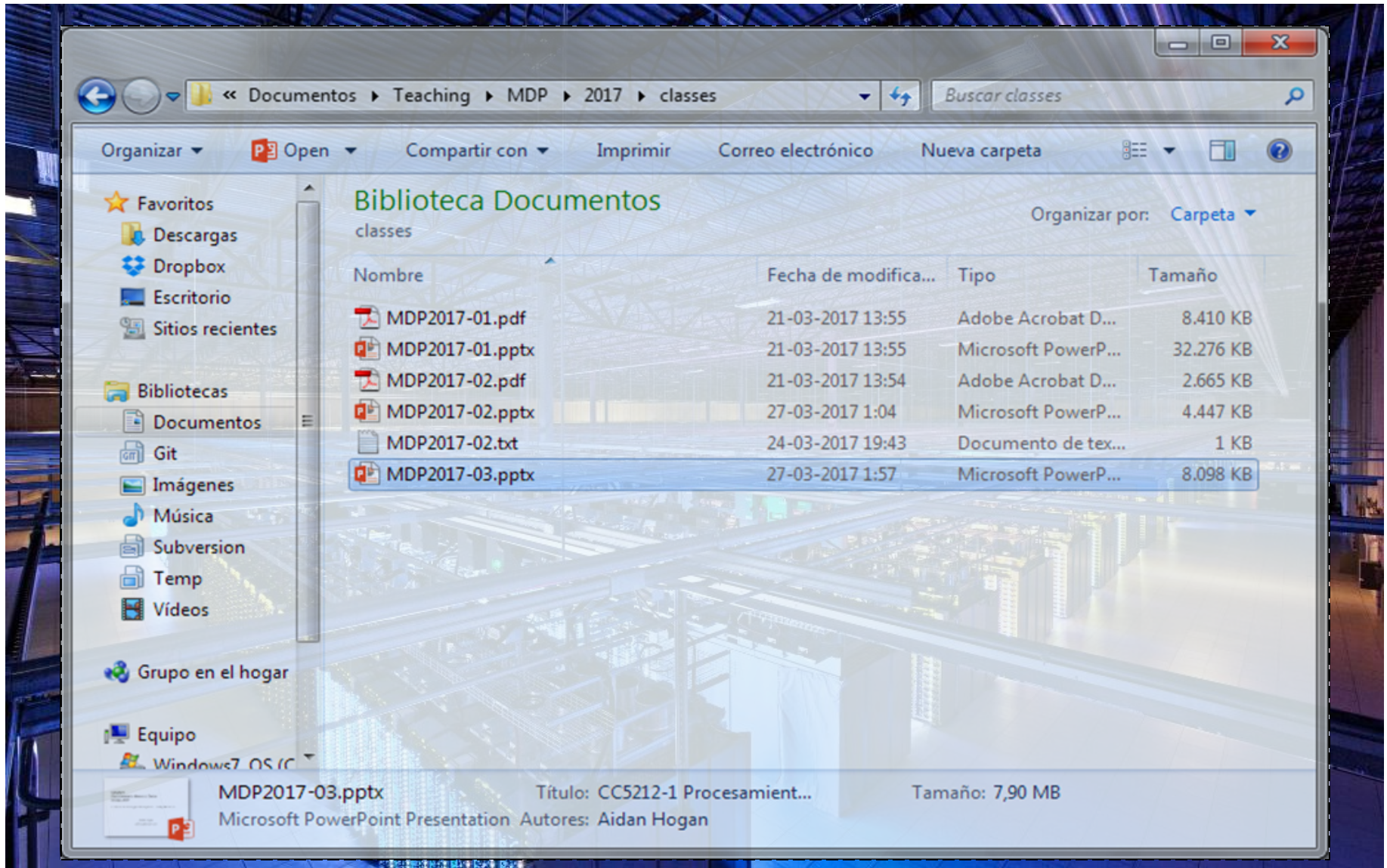
Google's MapReduce

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
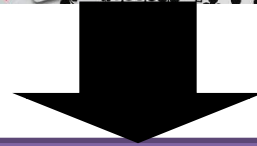
2♠×391, 3♠×192, 4♠×266, ..., Q♥×123, K♥×321, A♥×402

Moving to word count ...

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
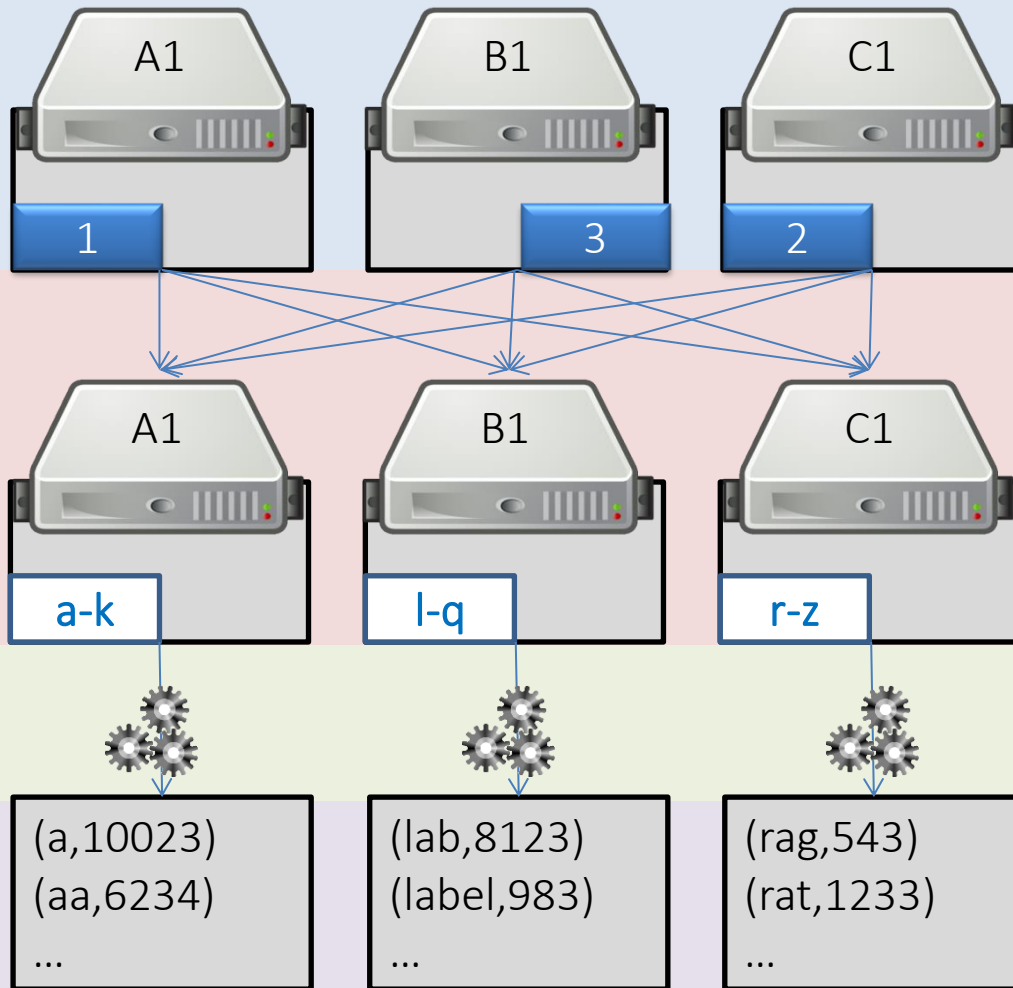
Input
File on Distr. File System

Partition

Distr. Sort/Count

Output
File on Distr. File System

A1	B1	C1
1	3	2

A1	B1	C1
a-k	l-q	r-z

(a,10023)	(lab,8123)	(rag,543)
(aa,6234)	(label,983)	(rat,1233)
…	…	…

Better partitioning? $\text{HASH}(\mathbf{w})\%m$

MapReduce: Under the Hood

MapReduce

1. Input

2. **Map**

3. Partition [Sort]

4. Shuffle

5. Merge Sort
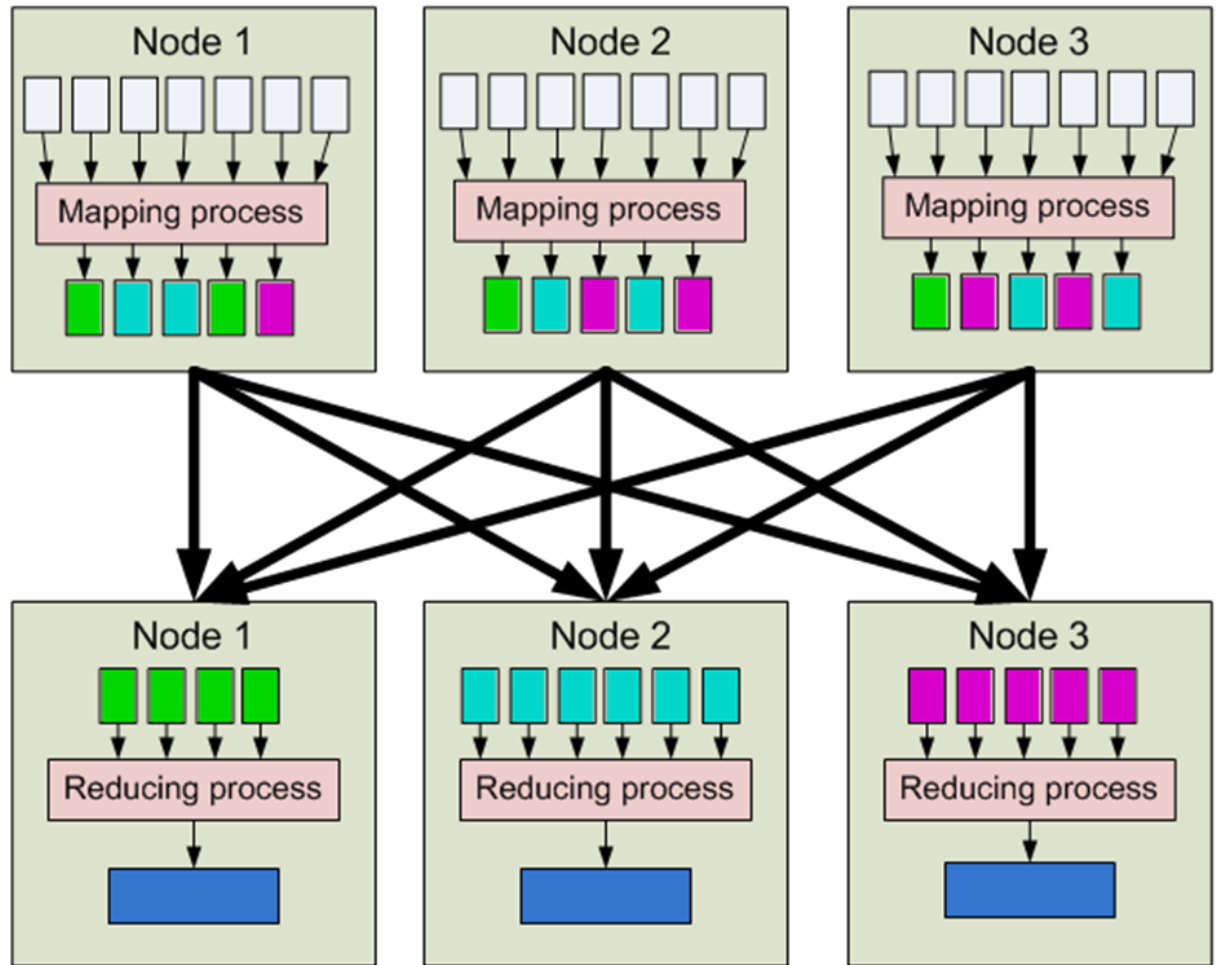
6. **Reduce**

7. Output

MapReduce: Counting Words

Input | Map | Partition / [Sort] | Shuffle | Merge Sort | Reduce | Output
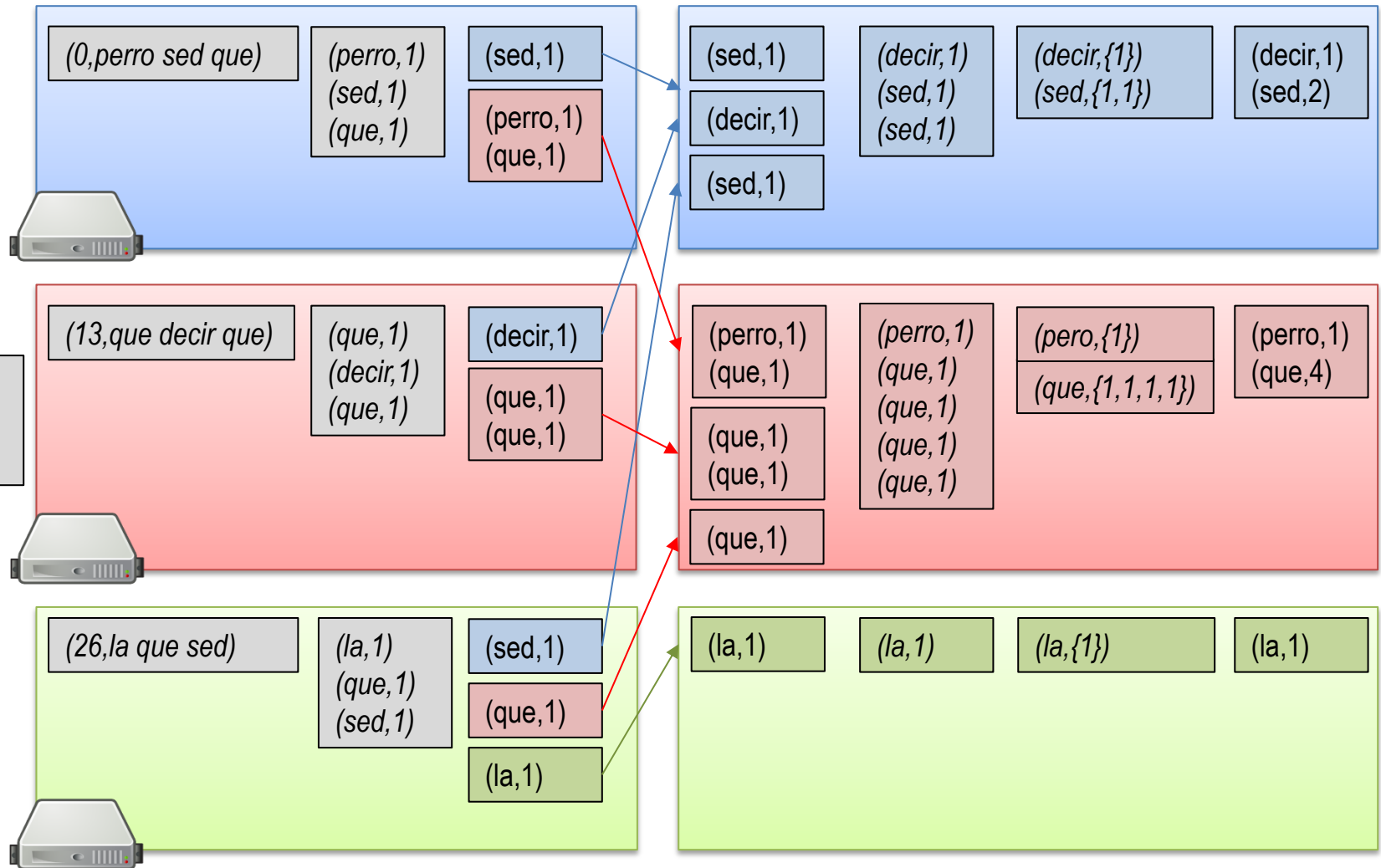
perro sed que que decir que la que sed

(0,perro sed que)

(perro,1)
(sed,1)
(que,1)

(sed,1)
(perro,1)
(que,1)

(sed,1)
(decir,1)
(sed,1)

(decir,1)
(sed,1)
(sed,1)

(decir,{1})
(sed,{1,1})

(decir,1)
(sed,2)

(13,que decir que)

(que,1)
(decir,1)
(que,1)

(decir,1)
(que,1)
(que,1)

(perro,1)
(que,1)

(que,1)
(que,1)

(que,1)

(perro,1)
(que,1)
(que,1)
(que,1)
(que,1)
(que,1)

(pero,{1})

(que,{1,1,1,1})

(perro,1)
(que,4)

(26,la que sed)

(la,1)
(que,1)
(sed,1)

(sed,1)
(que,1)
(la,1)

(la,1)

(la,1)

(la,{1})

(la,1)

MapReduce: Combiner

1. Input

2. **Map**

3. Partition [Sort]
   ("*Combine*")

4. Shuffle

5. Merge Sort

6. **Reduce**

7. Output

MapReduce: Combiner

| Input | Map | Partition / [Sort] | **Combine** | Shuffle | Merge Sort | Reduce | Output |

Blue server:

(0,perro sed que)

(perro,1)
(sed,1)
(que,1)

(sed,1)

(perro,1)
(que,1)

(sed,1)

(perro,1)
(que,1)

(sed,1)

(decir,1)

(sed,1)

(decir,1)
(sed,1)
(sed,1)

(decir,{1})
(sed,{1,1})

(decir,1)
(sed,2)

Red server:

(13,que decir que)

(que,1)
(decir,1)
(que,1)

(decir,1)

(que,1)
(que,1)

(decir,1)

(que,2)

(perro,1)
(que,1)

(que,2)

(que,1)

(perro,1)
(que,1)
(que,1)
(que,2)

(pero,{1})

(que,{1,1,**2**})

(perro,1)
(que,4)

Green server:

(26,la que sed)

(la,1)
(que,1)
(sed,1)

(sed,1)

(que,1)

(la,1)

(sed,1)

(que,1)

(la,1)

(la,1)

(la,1)

(la,{1})

(la,1)

MapReduce: Combiner

1. Input

2. **Map**

3. Partition [Sort]
   ("*Combine*")

4. Shuffle

5. Merge Sort

6. **Reduce**

7. Output

Combiner reduces data shuffled here!

MapReduce:

More complex tasks

Supermarket Example
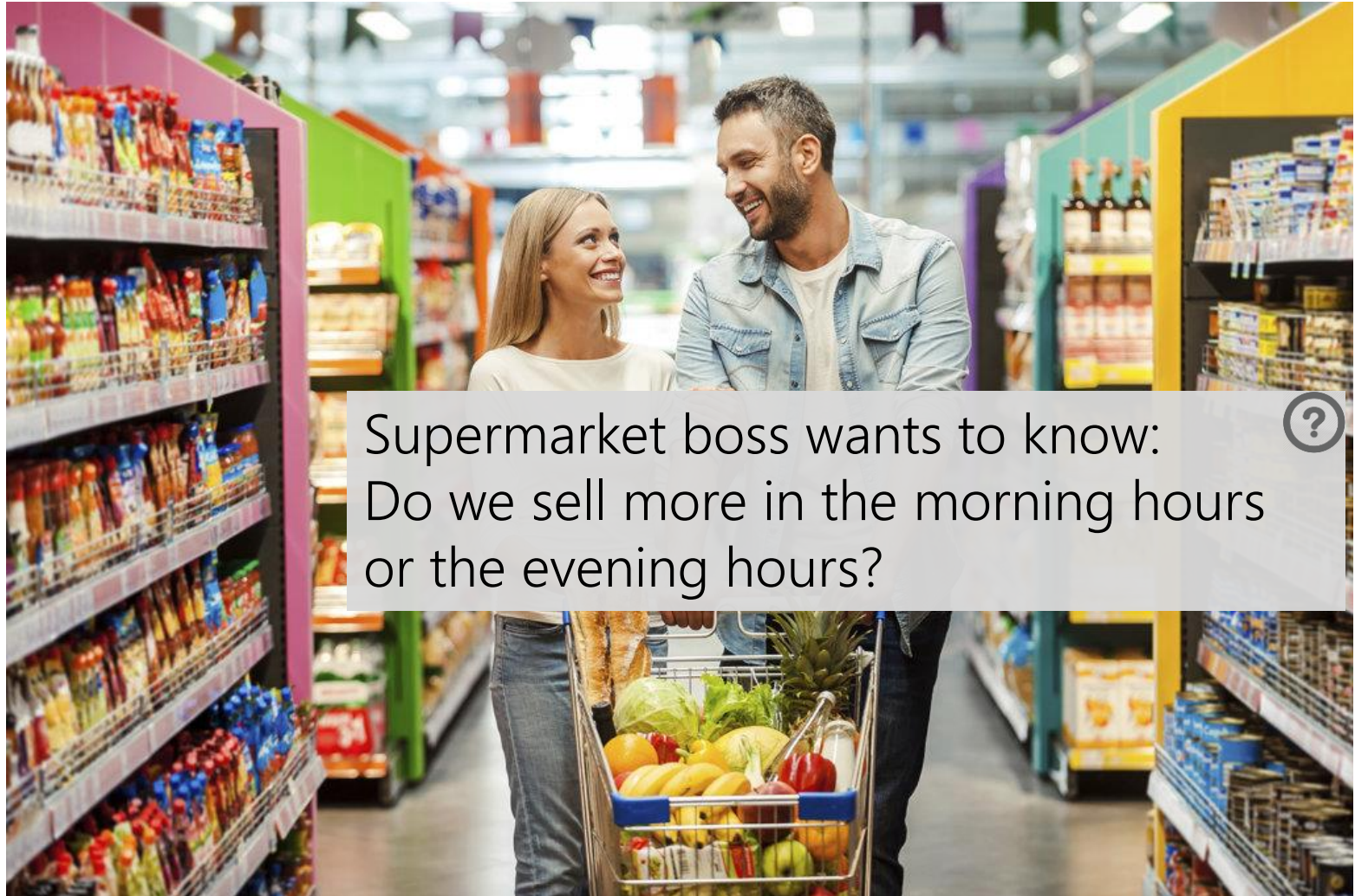
Supermarket boss wants to know:
Do we sell more in the morning hours or the evening hours?

MapReduce: Supermarket Example

Purchase

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

Time

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

Item

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

Compute total sales per hour of the day?

SalesPerHour

Hour	Total
...	...
18:00–18:59	$24871569670
19:00–19:59	$36576125100
...	...

MapReduce: Supermarket Example

Purchase

RECEIPT ID	ITEM ID
R1401	I306
R1401	I306
R1401	I504
R1402	I007
R1402	I306
R1403	I306
R1403	I504
. . .	. . .

Time

RECEIPT ID	TIME
R1403	19:00
R1401	18:59
R1402	19:01
. . .	. . .

Item

ITEM ID	NAME	PRICE ($)
I306	Zanahoria 500g	500
I504	CocaCola 3L	1400
I007	Comfort	1200
. . .	. . .	

- Map_{1A} (input: `Purchase`)
  - $(R,I) \mapsto \{(R,I)\}$
- Map_{1B} (input: `Time`)
  - $(R,T) \mapsto \{(R,\text{hour}(T))\}$
- Reduce_1 (input: Map_{1A}, Map_{1B})
  - $(R, [I_1,\ldots,I_n,H]) \mapsto$ $\{(I_1,H), \ldots, (I_n,H)\}$

- Map_{2A} (input: `Item`)
  - $(I,(N,P)) \mapsto \{(I,P)\}$
- Map_{2B} (input: Reduce_1)
  - $(I,H) \mapsto \{(I,H)\}$
- Reduce_2 (input: Map_{2A}, Map_{2B})
  - $(I, [H_1,\ldots,H_n,P]) \mapsto$ $\{(H_1,P),\ldots,(H_n,P)\}$

- Map_3 (input: Reduce_2)
  - $(H,P) \mapsto \{(H,P)\}$
- Reduce_3 (input: Map_3)
  - $(H, [P_1,\ldots,P_n]) \mapsto \{(H,\Sigma_{i=1}^{n}P_i)\}$
  - output: `SalesPerHour`

... one possible solution.

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)
- mapreduce(function map, function reduce, file in, file out)

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
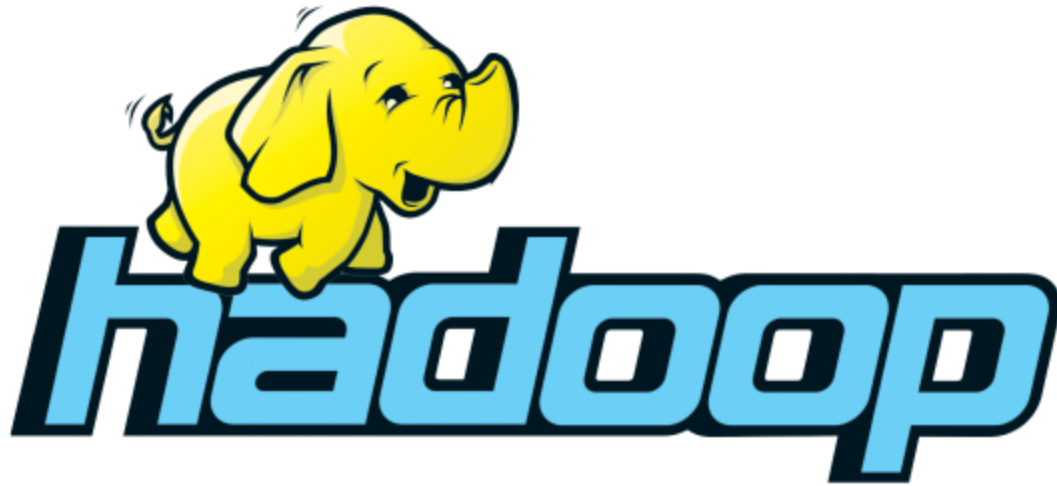 - Add / remove machines
  - Load balancing

MapReduce: Benefits for Programmers

(Time for more important things)

HDFS/Hadoop Overview

Hadoop: Open Source MapReduce

HDFS / Hadoop Core Architecture
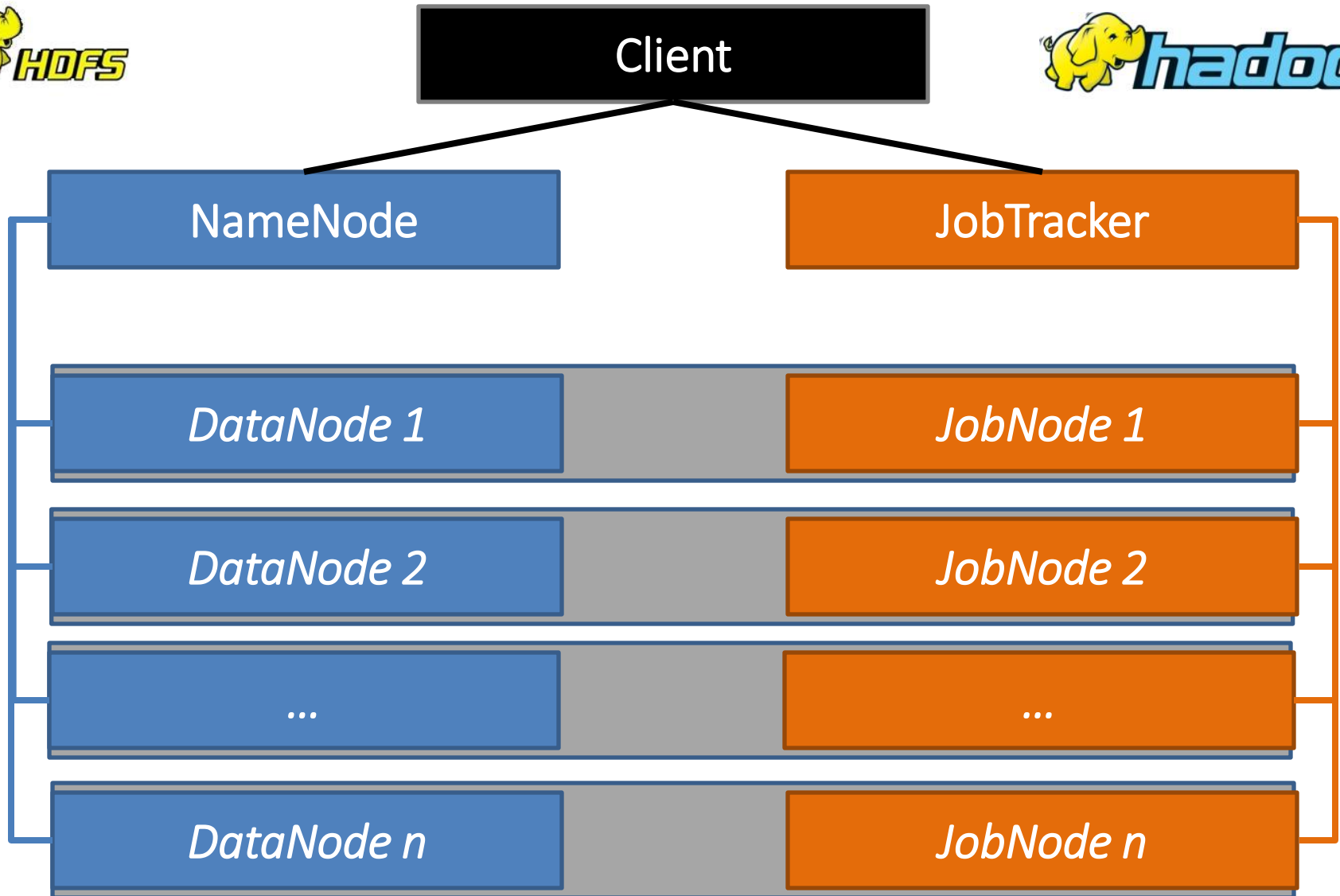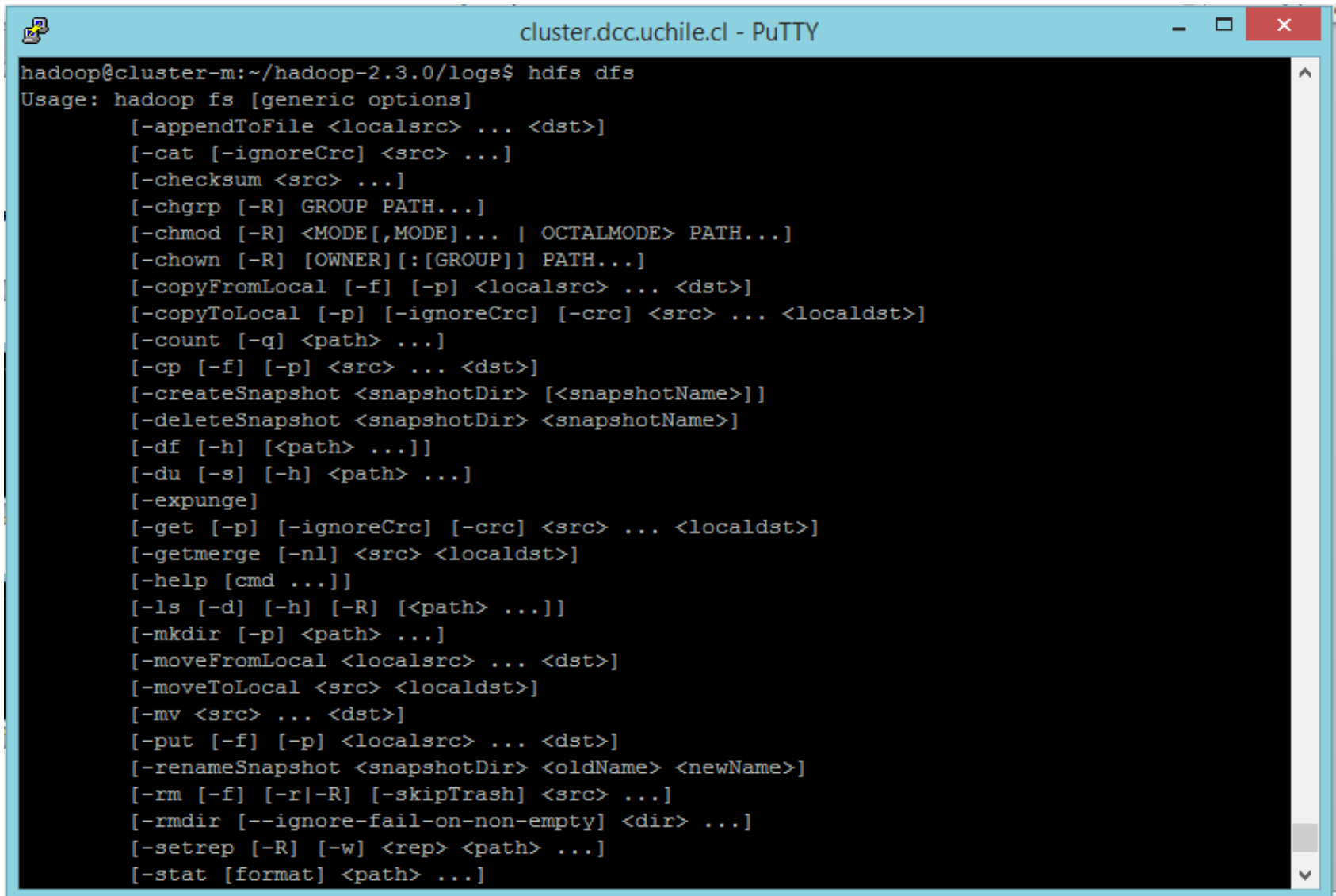
PROGRAMMING WITH HADOOP

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 (Java)

```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

Handles the input for you!

```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*)

```java
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*)

```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;
    }
}
```

New Interface

Same as before

Needed to sort keys

Needed for default partition function

(not needed in the running example)

3. Partition

PartitionerInterface

```java
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

Handles the output for you!

```java
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)

```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.

Can use a reducer as a combiner!

MORE IN HADOOP

More in Hadoop: Multiple Inputs

```java
public class RevenuePerHour {
    public static void main(String[] args) throws Exception {
        Configuration conf = new Configuration();
        String[] otherArgs = new GenericOptionsParser(conf, args).getRemainingArgs();
        if (otherArgs.length != 4) {
            System.err.println("Usage: WordCount <in1> <in2> <in3> <tmp1> <tmp2> <out>");
            System.exit(2);
        }

        Job job1 = Job.getInstance(new Configuration());
        MultipleInputs.addInputPath(job1, new Path(otherArgs[0]),
                TextInputFormat.class, ReceiptItemsMapper.class);
        MultipleInputs.addInputPath(job1, new Path(otherArgs[1]),
                TextInputFormat.class, ReceiptTimesMapper.class);
        FileOutputFormat.setOutputPath(job1, new Path(otherArgs[3]));

        job1.setReducerClass(ItemsTimesReducer.class);
        job1.setMapOutputKeyClass(Text.class);
        job1.setMapOutputValueClass(Text.class);
        job1.setOutputKeyClass(Text.class);
        job1.setOutputValueClass(Text.class);
        job1.waitForCompletion(true);
```

Multiple inputs, different map for each

One reducer

More in Hadoop: Chaining Jobs

```java
public class RevenuePerHour {
    public static void main(String[] args) throws Exception {
        Configuration conf = new Configuration();
        String[] otherArgs = new GenericOptionsParser(conf, args).getRemainingArgs();
        if (otherArgs.length != 4) {
            System.err.println("Usage: WordCount <in1> <in2> <in3> <tmp1> <tmp2> <out>");
            System.exit(2);
        }

        Job job1 = Job.getInstance(new Configuration());
        MultipleInputs.addInputPath(job1, new Path(otherArgs[0]),
                TextInputFormat.class, ReceiptItemsMapper.class);
        MultipleInputs.addInputPath(job1, new Path(otherArgs[1]),
                TextInputFormat.class, ReceiptTimesMapper.class);
        FileOutputFormat.setOutputPath(job1, new Path(otherArgs[3]));

        job1.setReducerClass(ItemsTimesReducer.class);
        job1.setMapOutputKeyClass(Text.class);
        job1.setMapOutputValueClass(Text.class);
        job1.setOutputKeyClass(Text.class);
        job1.setOutputValueClass(Text.class);
        job1.waitForCompletion(true);

        Job job2 = Job.getInstance(new Configuration());
        MultipleInputs.addInputPath(job2, new Path(otherArgs[2]),
                TextInputFormat.class, ItemsTimesMapper.class);
        MultipleInputs.addInputPath(job2, new Path(otherArgs[3]),
                TextInputFormat.class, ItemsPricesMapper.class);
        FileOutputFormat.setOutputPath(job2, new Path(otherArgs[4]));

        job2.setReducerClass(TimesPricesReducer.class);
        job2.setMapOutputKeyClass(LongWritable.class);
        job2.setMapOutputValueClass(Text.class);
```

Run and wait

Output of Job1 set to Input of Job2

Number of Reducers

```
job.setNumReduceTasks(1);
```

Set number of parallel reducer tasks for the job

Why would we ask for 1 reduce task? (?)

Output requires a merge on one machine (for example, sorting, top-*k*) (!)

More in Hadoop: Counters

```java
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: Distributed Cache

- Some tasks need "global knowledge"

- Use a distributed cache:
  - Makes global data available locally to all nodes
    - On the local hard-disk of each machine
  - Should be used sparingly, for small data volumes

Questions?