Big Data

Diplomado de Datos 2021

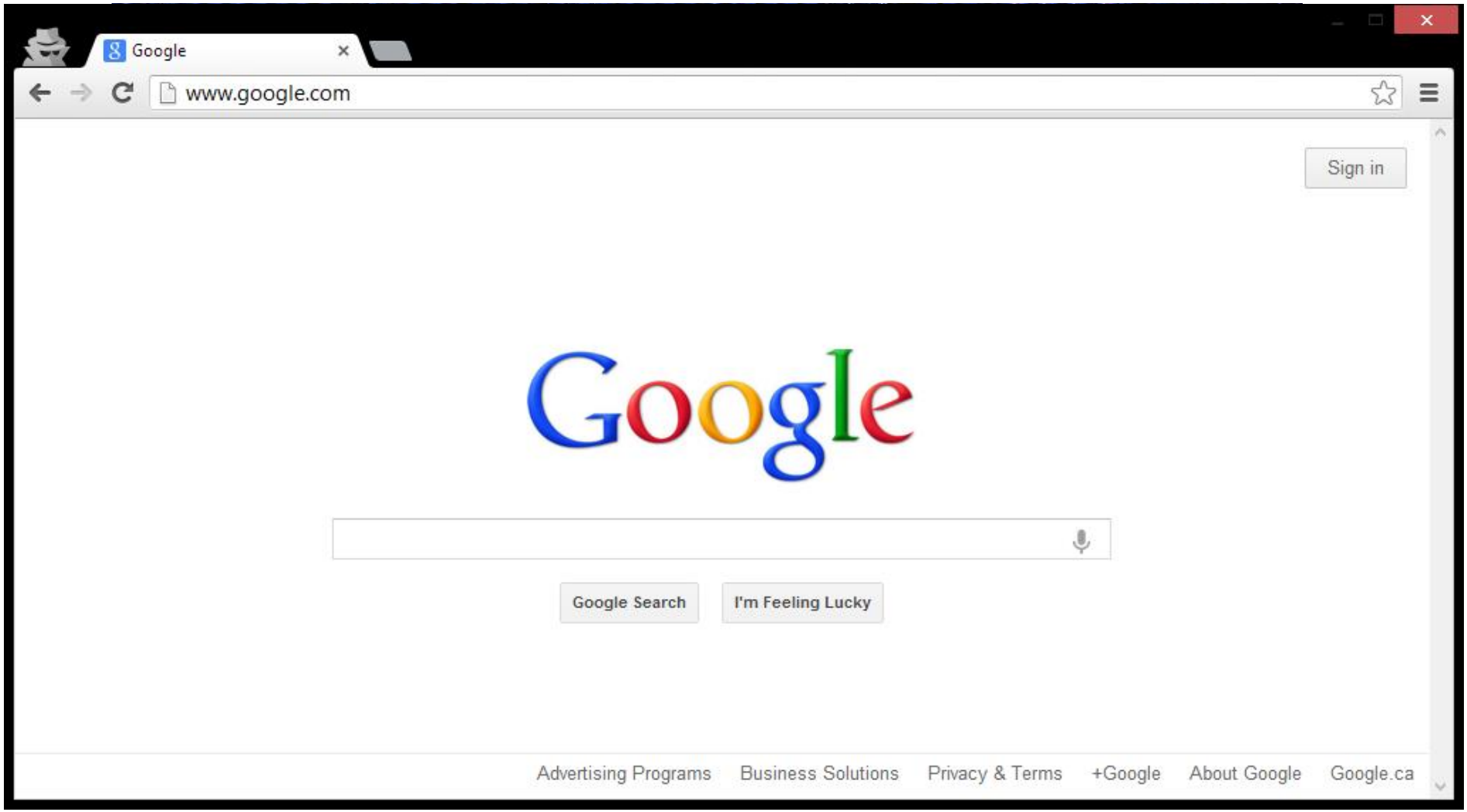
Clase 2: DFS y MapReduce

Aidan Hogan
aidhog@gmail.com

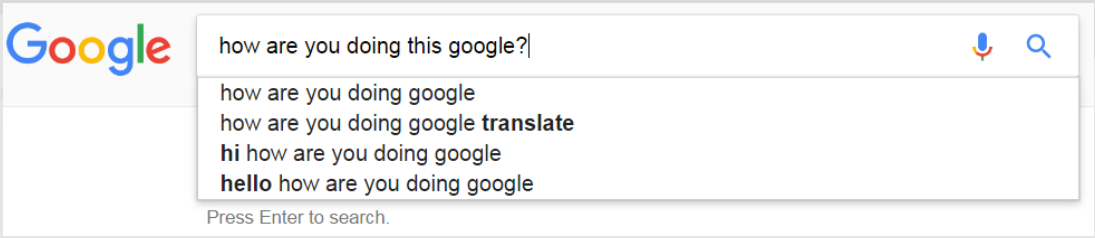
Gestión de datos masivos
... como lo hace google ...

Dentro de Google: 1997/98

Dentro de Google: 2017



Implementando la búsqueda de Google



¿Qué procesos y algoritmos necesita Google para implementar su búsqueda de la Web?

Crawling

1. Parsear enlaces de las páginas
2. Ordenar los enlaces para descargar
3. Descargar páginas, GOTO 1

Indexación

1. Parsear keywords de las páginas
2. Indexar sus keywords
3. Administrar actualizaciones

Ranking

1. ¿Qué tan relevante es una página?
2. ¿Qué tan importante es?
3. ¿Cuántos clics tiene?

…

Implementando la búsqueda de Google

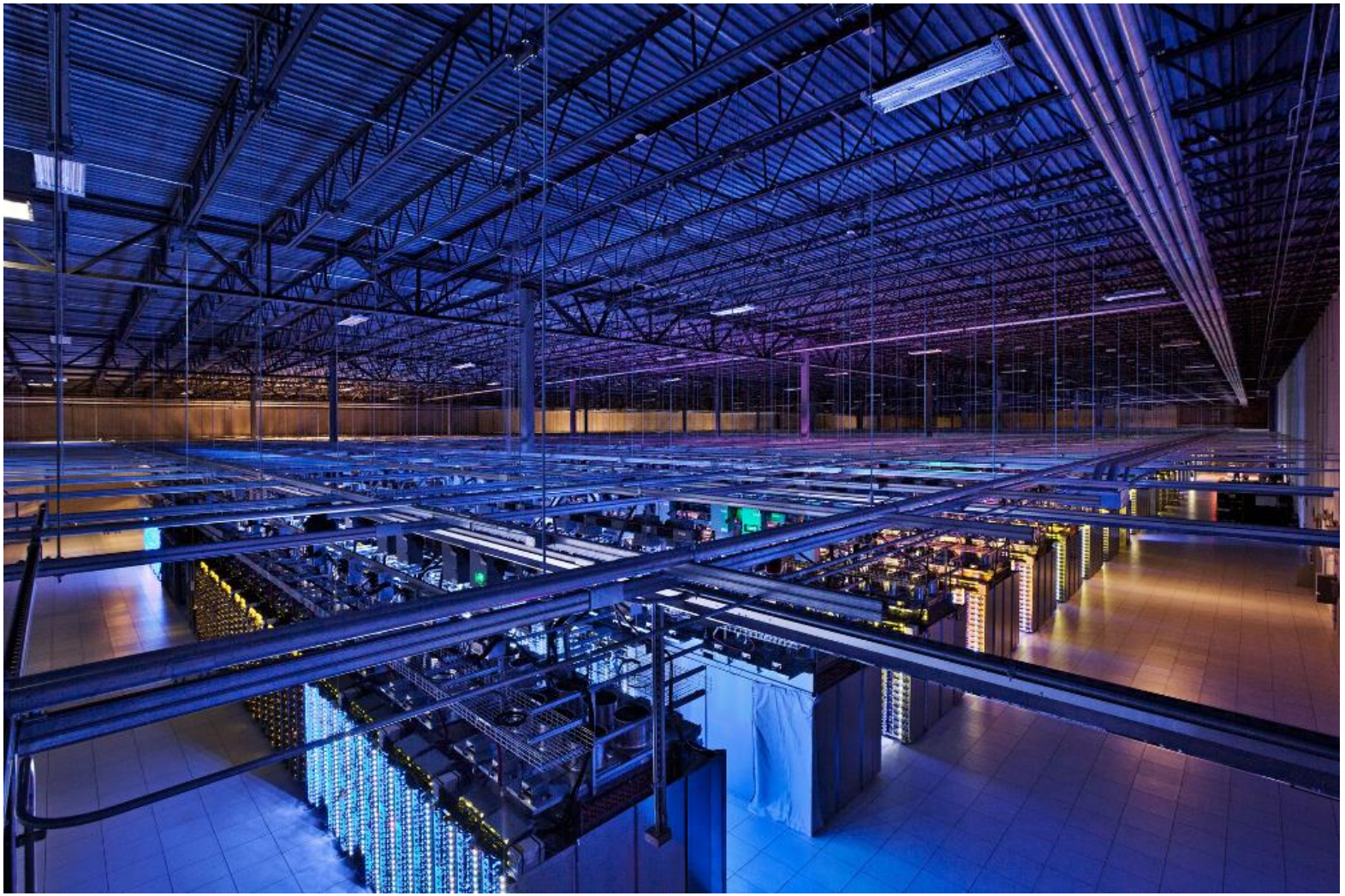
Google

≈ 100 PB / día

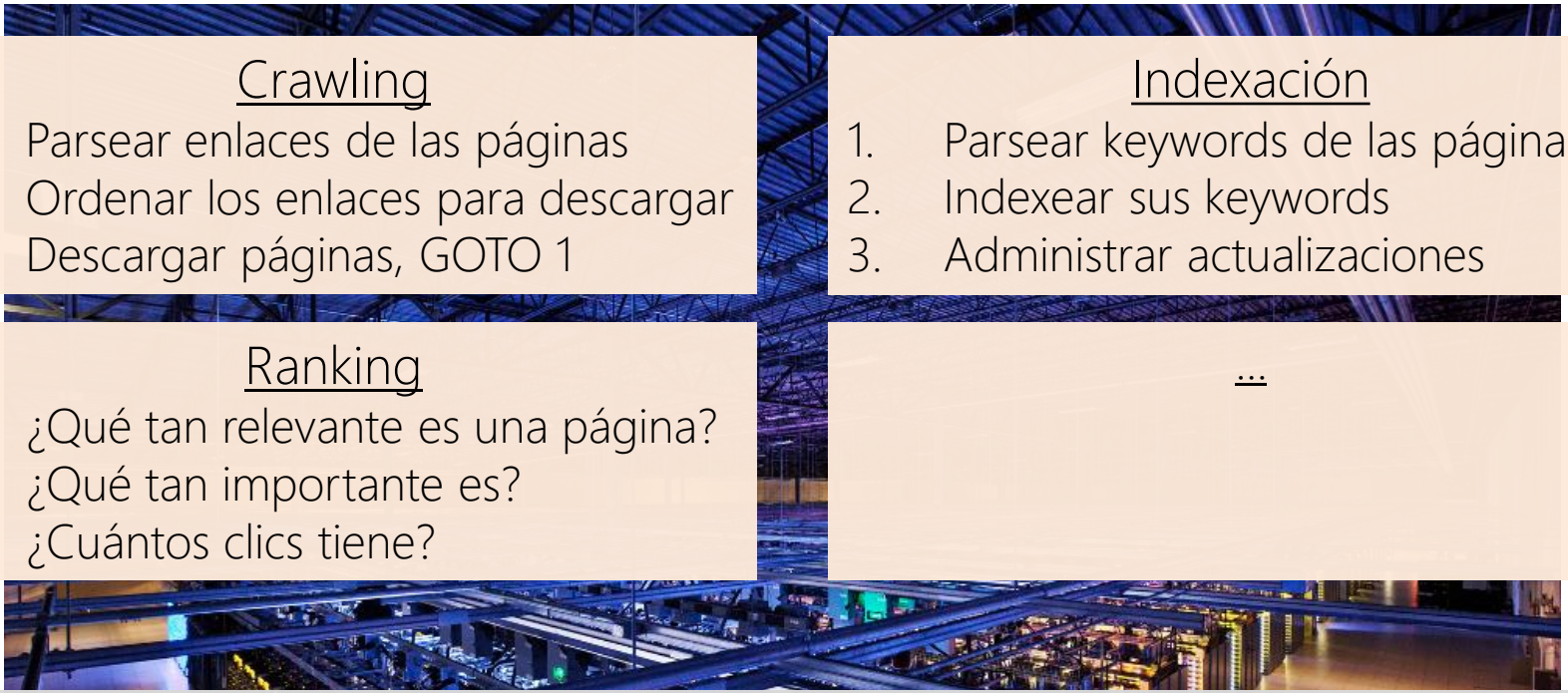
≈ 2,000,000 Wiki / día

(2014, procesados)

Implementando la búsqueda de Google



Implementando la búsqueda de Google



Crawling

1. Parsear enlaces de las páginas
2. Ordenar los enlaces para descargar
3. Descargar páginas, GOTO 1

Indexación

1. Parsear keywords de las páginas
2. Indexear sus keywords
3. Administrar actualizaciones

Ranking

1. ¿Qué tan relevante es una página?
2. ¿Qué tan importante es?
3. ¿Cuántos clics tiene?

…

¿Cuáles abstracciones distribuidas podemos considerar para facilitar aplicaciones así?

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

… al menos para empezar

Google File System (GFS)

(Sistema de archivos de Google)

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.

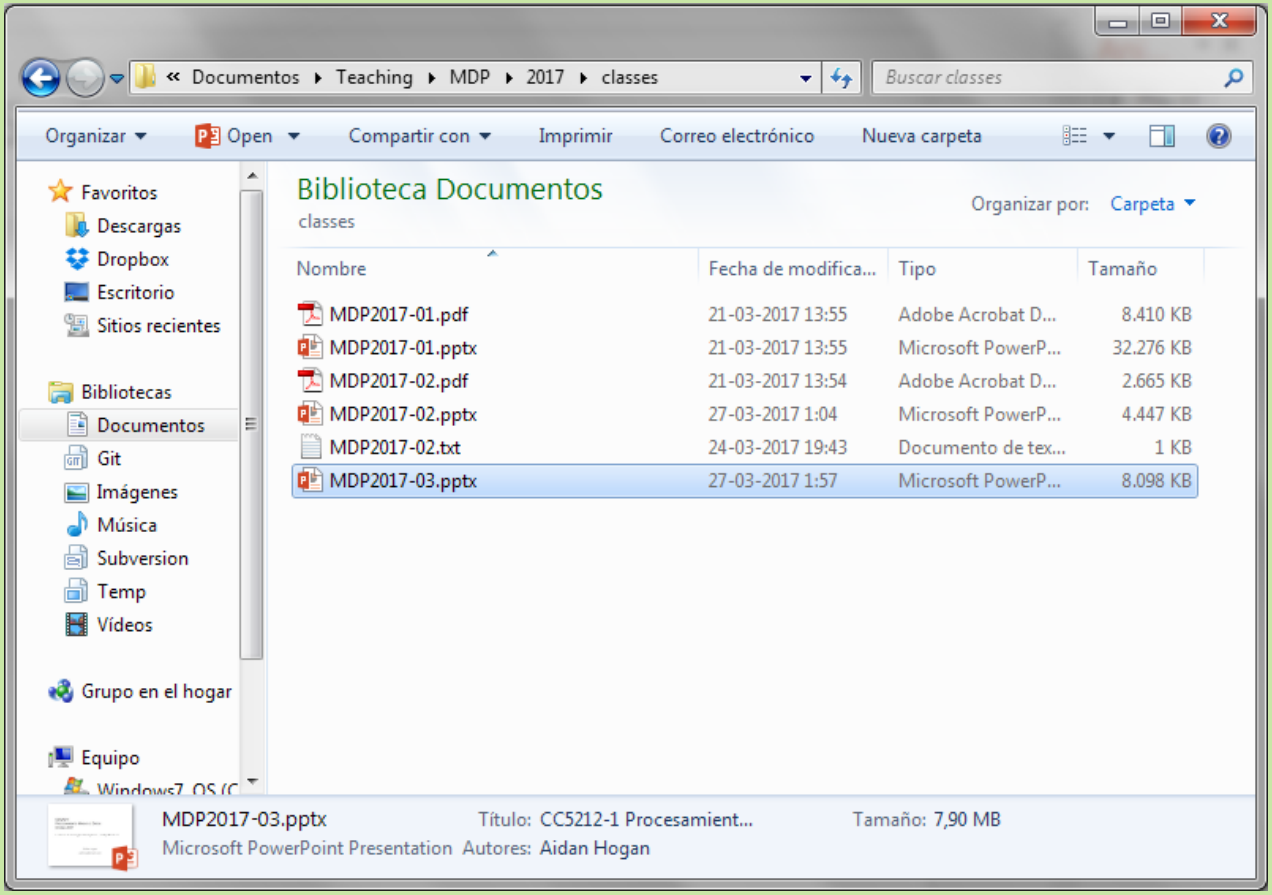
1. INTRODUCTION

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.

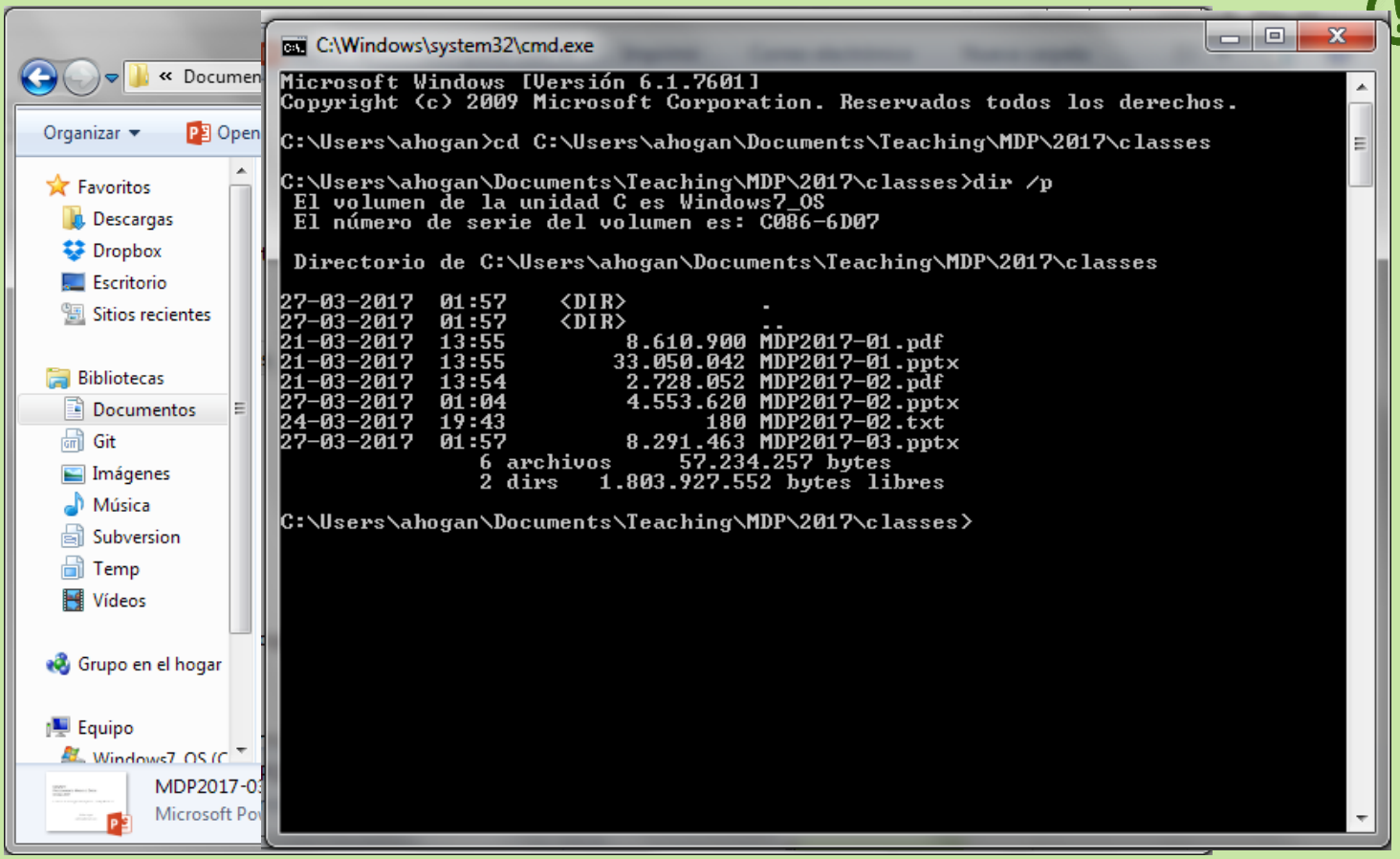
Google File System

¿Qué es un sistema de archivos (un "file-system")?



Google File System

¿Qué es un sistema de archivos (un "file-system")?

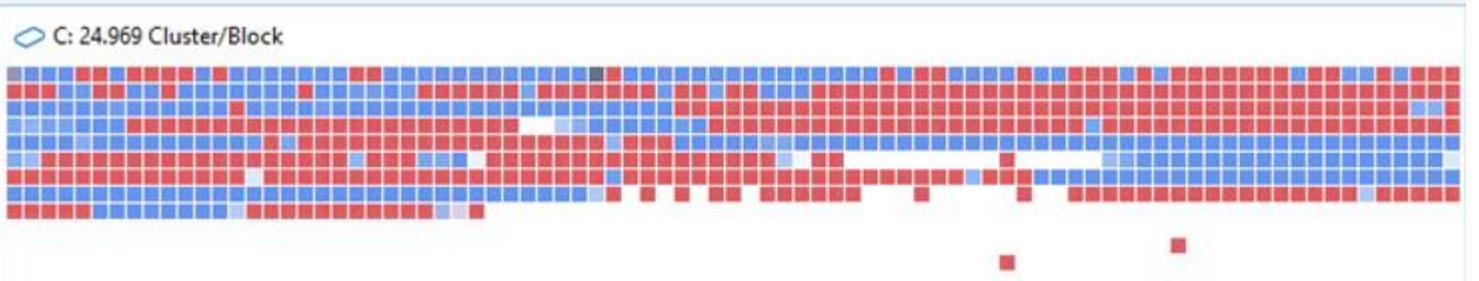


Google File System

¿Qué hace un sistema de archivos?

1. Divide un archivo en bloques de almacenamiento
   - Guarda la ubicación y secuencia de los bloques de un archivo

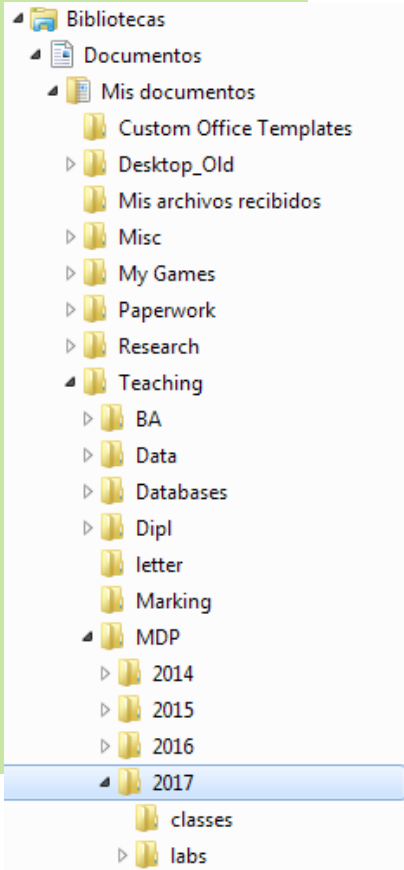
Drive	Name	Action	Status	Total files	Frag. files	Degree of fragmentation	Size	Free	File system	Current file/folder	Rer
C:	System	Ready	0%	232.771	379	0,67%	194,96 GB	104,42 GB	NTFS		
D:		Ready	0%	41	0	0,00%	10,00 GB	9,95 GB	NTFS		
E:	Data II	Ready	0%	121.351	8.398	52,06%	172,78 GB	108,62 GB	NTFS		



Google File System

¿Qué hace un sistema de archivos?

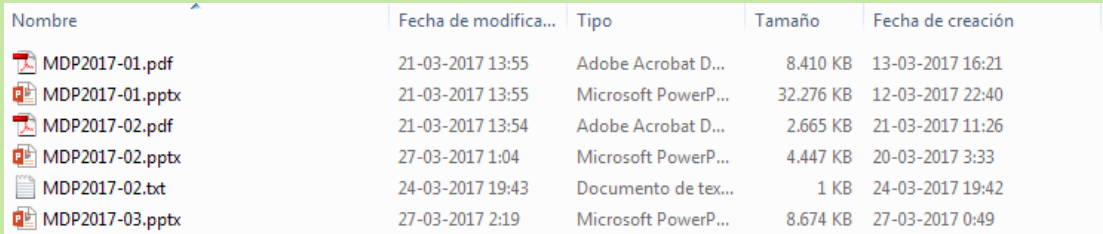
1. Divide un archivo en bloques de almacenamiento
   - Guarda la ubicación y secuencia de los bloques de un archivo
2. Organiza una estructura jerárquica de carpetas
   - Explora sub-carpetas y archivos en las carpetas



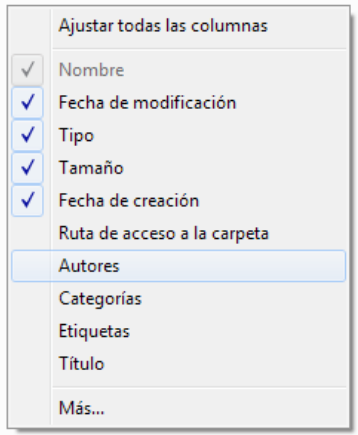
Google File System

¿Qué hace un sistema de archivos?

1. Divide un archivo en bloques de almacenamiento
   - Guarda la ubicación y secuencia de los bloques de un archivo
2. Organiza una estructura jerárquica de carpetas
   - Explora sub-carpetas y archivos en las carpetas
3. Guarda los meta datos de los archivos
   - Tamaño de los archivos, fecha de creación



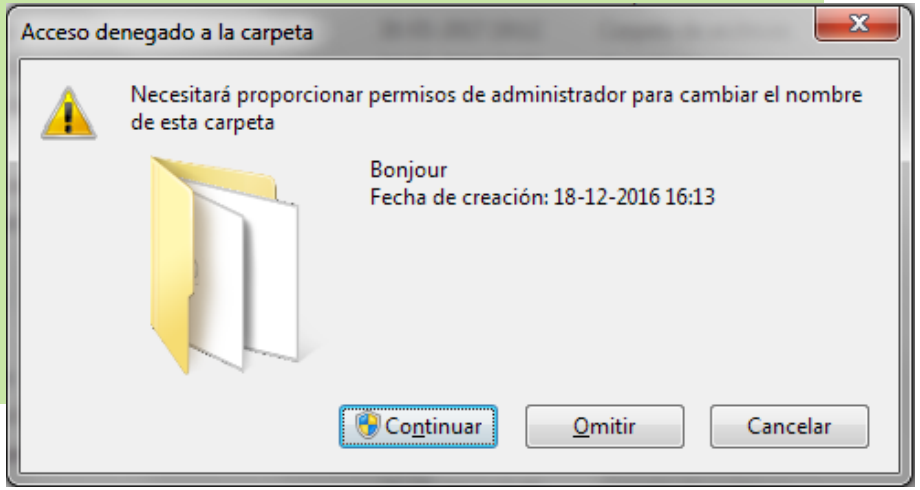
Nombre	Fecha de modifica...	Tipo	Tamaño	Fecha de creación
MDP2017-01.pdf	21-03-2017 13:55	Adobe Acrobat D...	8.410 KB	13-03-2017 16:21
MDP2017-01.pptx	21-03-2017 13:55	Microsoft PowerP...	32.276 KB	12-03-2017 22:40
MDP2017-02.pdf	21-03-2017 13:54	Adobe Acrobat D...	2.665 KB	21-03-2017 11:26
MDP2017-02.pptx	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



Google File System

¿Qué hace un sistema de archivos?

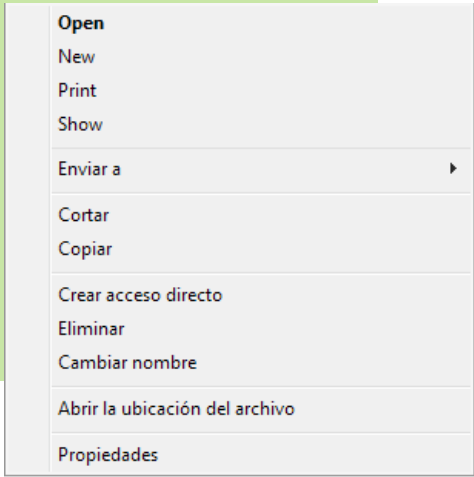
1. Divide un archivo en bloques de almacenamiento
   - Guarda la ubicación y secuencia de los bloques de un archivo
2. Organiza una estructura jerárquica de carpetas
   - Explora sub-carpetas y archivos en las carpetas
3. Guarda los meta datos de los archivos
   - Tamaño de los archivos, fecha de creación
4. Protege los archivos
   - Propiedad, permisos, bloqueos de sincronización



Google File System

¿Qué hace un sistema de archivos?

1. Divide un archivo en bloques de almacenamiento
   - Guarda la ubicación y secuencia de los bloques de un archivo
2. Organiza una estructura jerárquica de carpetas
   - Explora sub-carpetas y archivos en las carpetas
3. Guarda los meta datos de los archivos
   - Tamaño de los archivos, fecha de creación
4. Protege los archivos
   - Propiedad, permisos, bloqueos de sincronización
5. Provee una interfaz para interactuar con archivos
   - Crear, borrar, copiar, etc.



Google File System

¿Qué hace el "Google File System"?

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

¿Qué hace el "Google File System"?

1. Divide un archivo en bloques de almacenamiento
   - Guarda la ubicación y secuencia de los bloques de un archivo
2. Organiza una estructura jerárquica de carpetas
   - Explora sub-carpetas y archivos en las carpetas
3. Guarda los meta datos de los archivos
   - Tamaño de los archivos, fecha de creación
4. Protege los archivos
   - Propiedad, permisos, bloqueos de sincronización
5. Provee una interfaz para interactuar con archivos
   - Crear, borrar, copiar, etc.

Lo mismo, pero distribuido:
(y abstrae la distribución)

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: Suposiciones

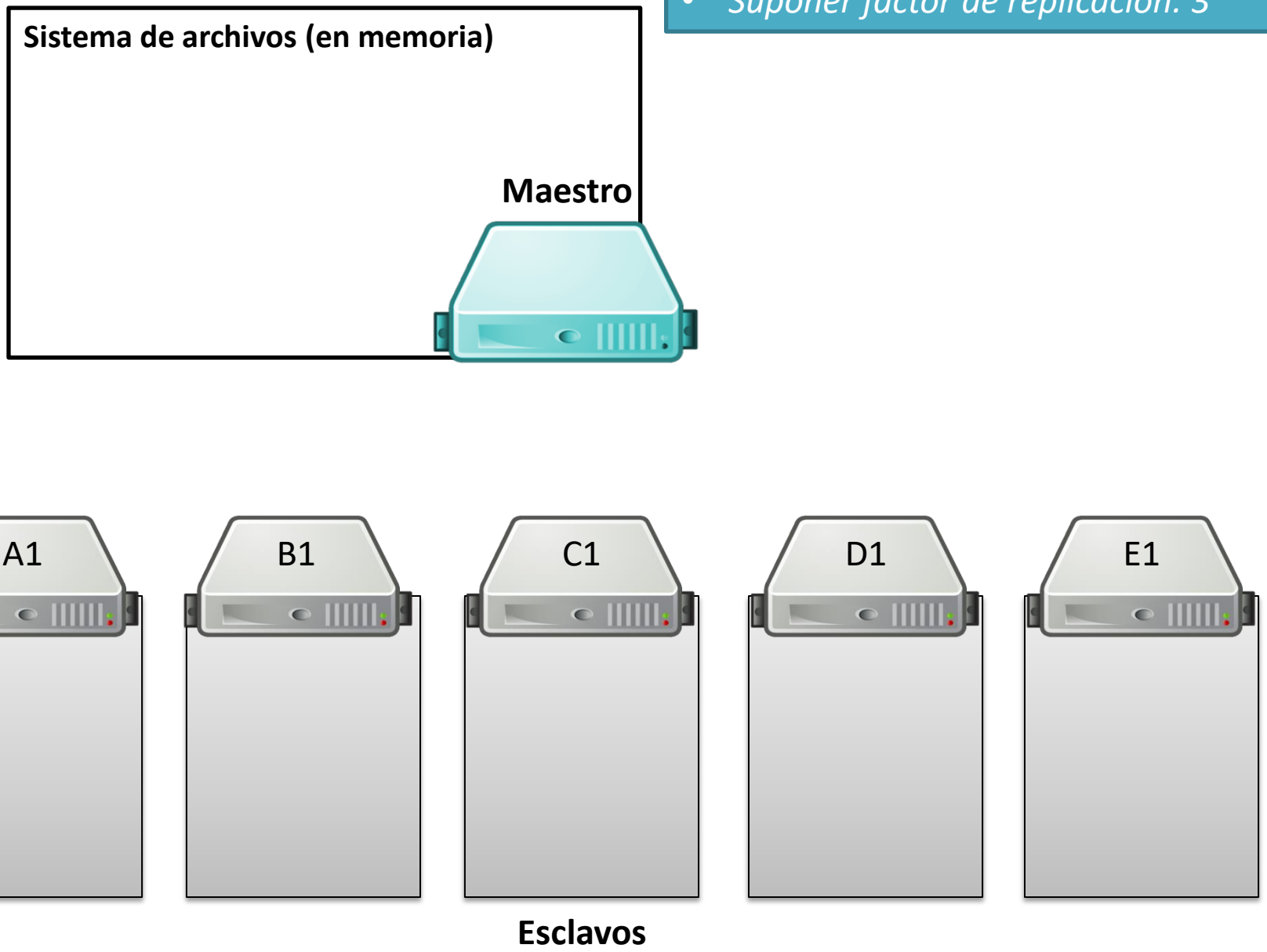
- Archivos enormes 
- Archivos frecuentemente leídos o anexados ("appended")
- La concurrencia es importante
- Fallas frecuentes
- El "streaming" es importante

GFS: Términos originales

- Bloques = "Chunks"
- Esclavos = "Chunk-Servers"

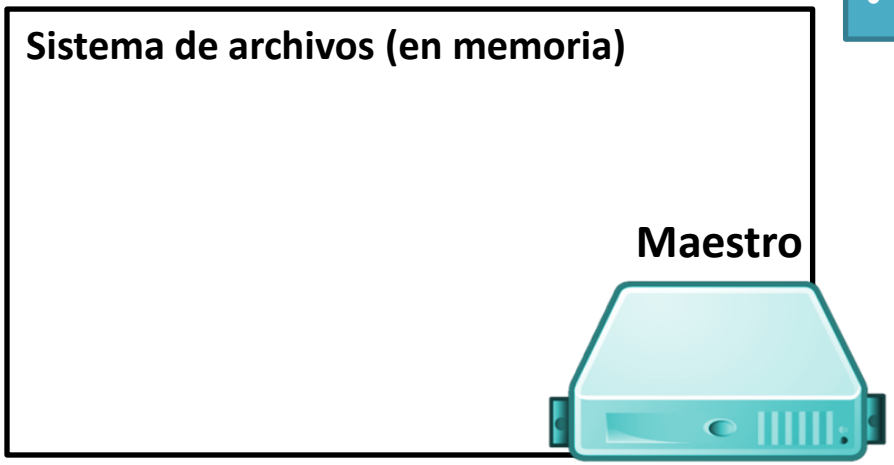
GFS: Arquitectura

- *64 MB por bloque*
- *Etiqueta de 64 bit para cada bloque*
- *Suponer factor de replicación: 3*



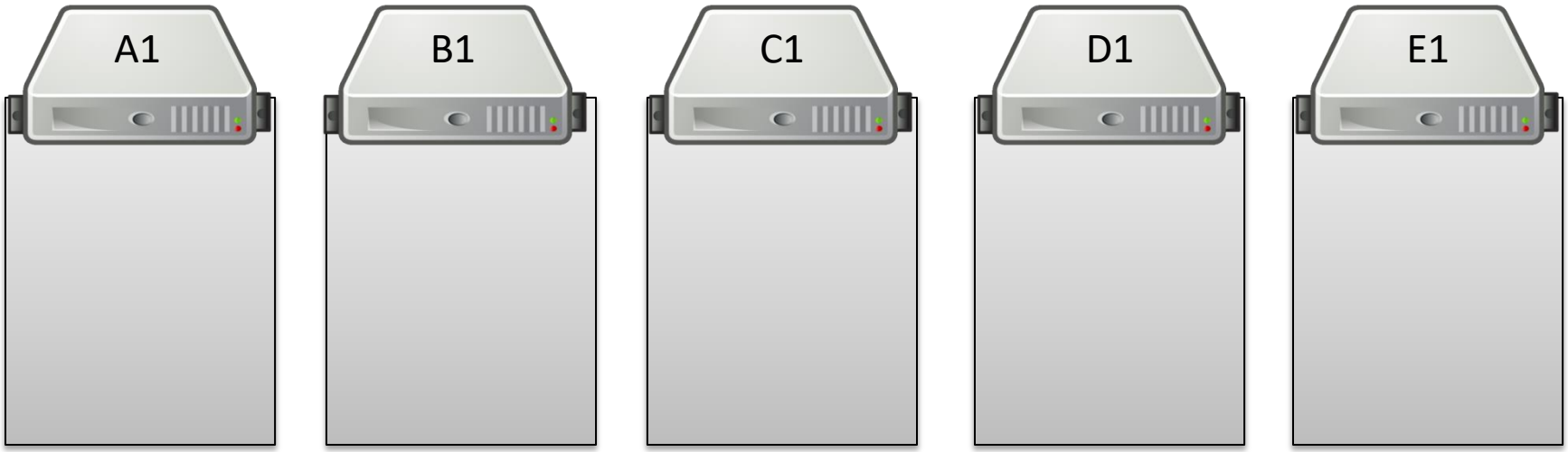
GFS: Escritura

- *64 MB por bloque*
- *Etiqueta de 64 bit para cada bloque*
- *Suponer factor de replicación: 3*



`blue.txt`
(150 MB: 3 bloques)

¿Cómo deberíamos proceder con la escritura?

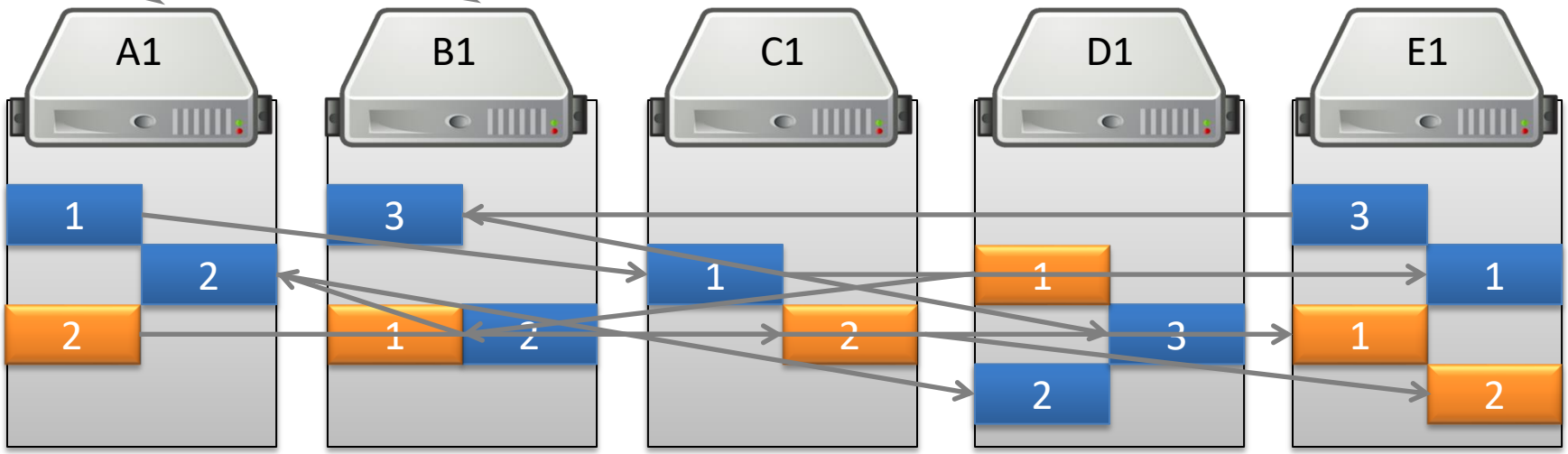


GFS: Escritura

- *64 MB por bloque*
- *Etiqueta de 64 bit para cada bloque*
- *Suponer factor de replicación: 3*

Sistema de archivos (en memoria)
`/blue.txt` [3 bloques]
1: {A1, C1, E1}
2: {A1, B1, D1}
3: {B1, D1, E1}
`/orange.txt` [2 bloques]
1: {B1, D1, E1}
2: {A1, C1, E1}

`blue.txt`
(150 MB: 3 bloques)
`orange.txt`
(100 MB: 2 bloques)



Esclavos

GFS: Tolerancia a fallos

- *64 MB por bloque*
- *Etiqueta de 64 bit para cada bloque*
- *Suponer factor de replicación: 3*

Sistema de archivos (en memoria)

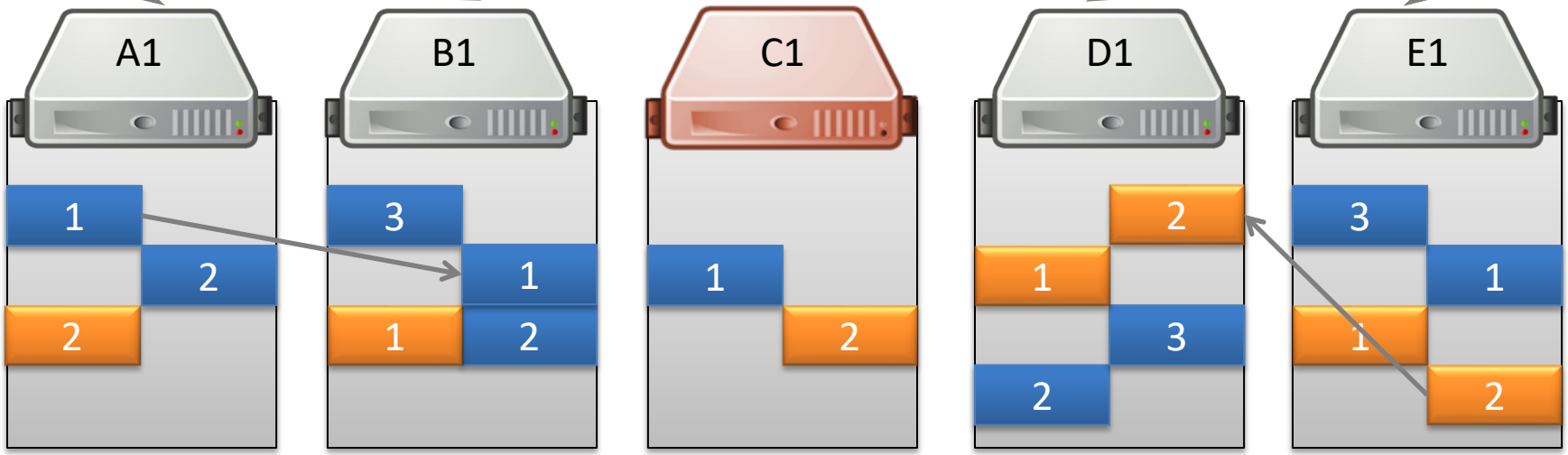
`/blue.txt` [3 bloques]
1: {A1, B1, E1}
2: {A1, B1, D1}
3: {B1, D1, E1}

`/orange.txt` [2 bloques]
1: {B1, D1, E1}
2: {A1, D1, E1}

Maestro

`blue.txt`
(150 MB: 3 bloques)

`orange.txt`
(100 MB: 2 bloques)



Esclavos

GFS: Lecturas directas

- *64 MB por bloque*
- *Etiqueta de 64 bit*
- *Suponer factor de*

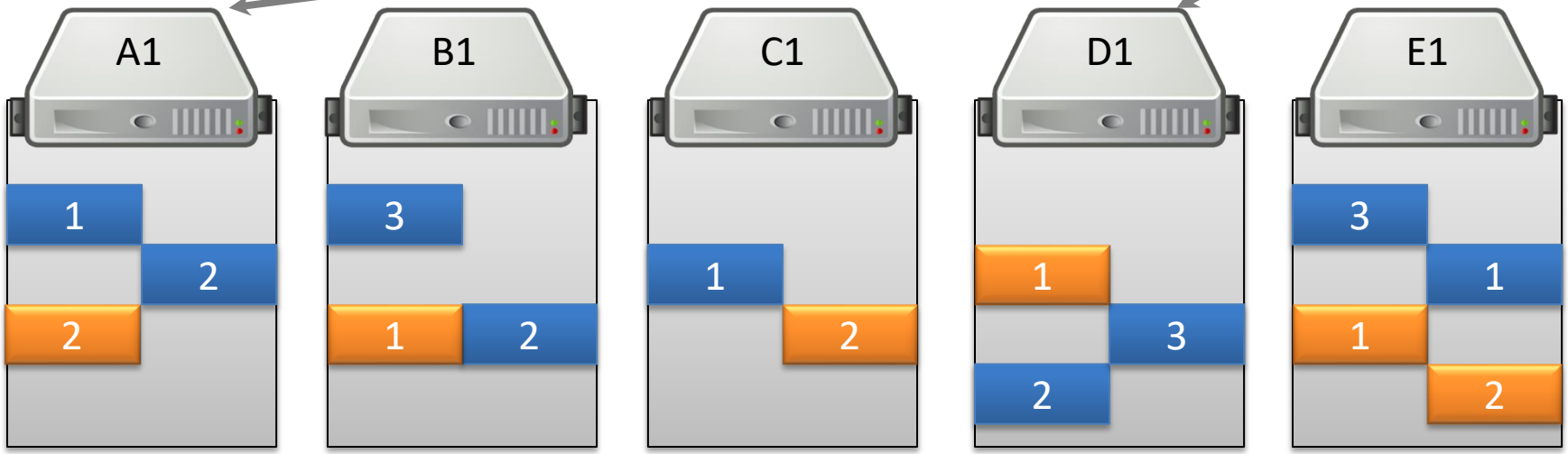
Sistema de archivos (en memoria)
`/blue.txt` [3 bloques]
1: {A1, C1, E1}
2: {A1, B1, D1}
3: {B1, D1, E1}
`/orange.txt` [2 bloques]
1: {B1, D1, E1}
2: {A1, C1, E1}

Busco
`/blue.txt`

1 2 3

Maestro

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



Esclavos

GFS: Replicas Primarias

- *64 MB por bloque*
- *Etiqueta de 64 bit*
- *Suponer factor de*

Sistema de archivos (en memoria)
/blue.txt [3 bloques]
1: {A1, C1, E1}
2: {A1, B1, D1}
3: {B1, D1, E1}
/orange.txt [2 bloques]
1: {B1, D1, E1}
2: {A1, C1, E1}

Maestro

Quiero cambiar el bloque 2 de /blue.txt

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

2

COMMIT
ACK
COMMIT
COMMIT
ACK
ACK

A1 B1 C1 D1 E1

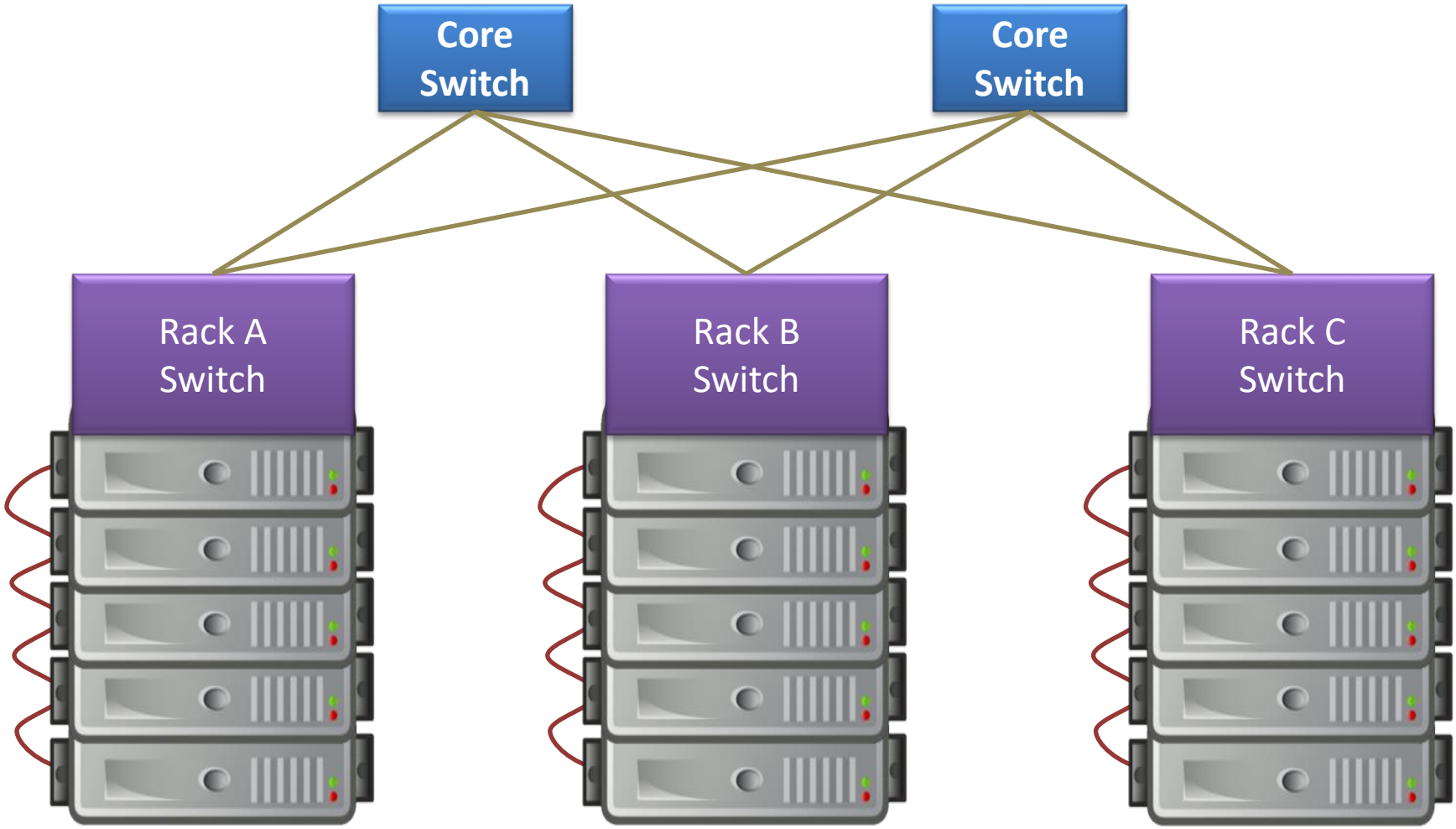
2 2 2 2

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

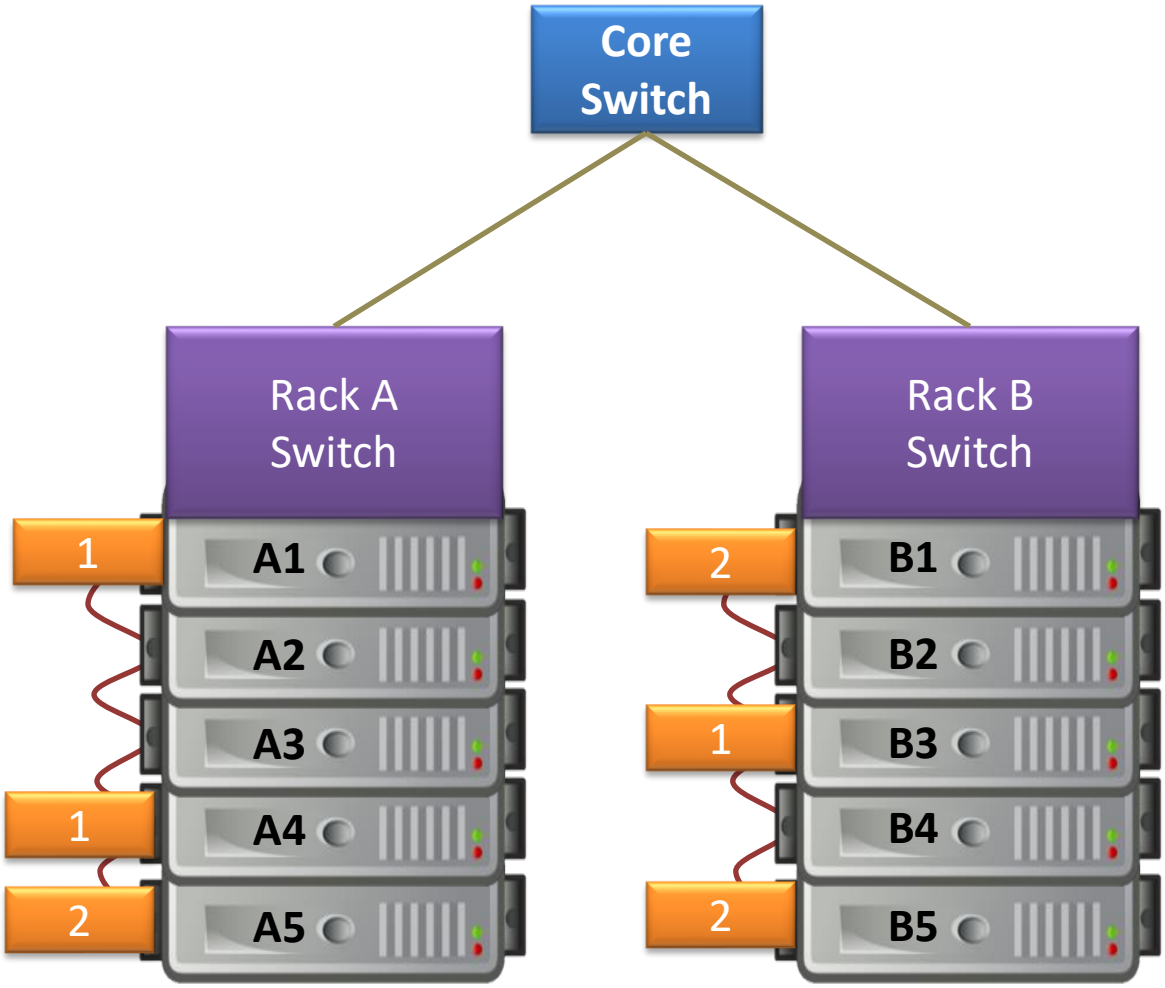
Esclavos

GFS: Conocimiento sobre Racks

GFS: Conocimiento sobre Racks



GFS: Conocimiento sobre Racks



Archivos:

`/orange.txt`

1: {A1, A4, B3}

2: {A5, B1, B5}

Racks:

A: {A1, A2, A3, A4, A5}

B: {B1, B2, B3, B4, B5}

GFS: Otras operaciones

Rebalanceo:

- Distribuir el almacenamiento equitativamente

Borrado:

- Renombrar el archivo con nombre de archivo oculto
  - Para recuperar, volver a renombrar a versión original
  - Si no, será eliminado después de tres días

Monitoreo de réplicas obsoletas:

- Esclavo “muerto” reaparece con datos antiguos: maestro guardará información de la versión y reciclará bloques viejos

GFS: ¿Debilidades?

¿Cuáles son las debilidades principales del GFS?

Maestro es un único punto de falla

- Usar replicación de hardware
- ¡Logs y checkpoints!

Maestro es un cuello de botella

- Usar una máquina más poderosa
- Minimizar tráfico por nodo maestro

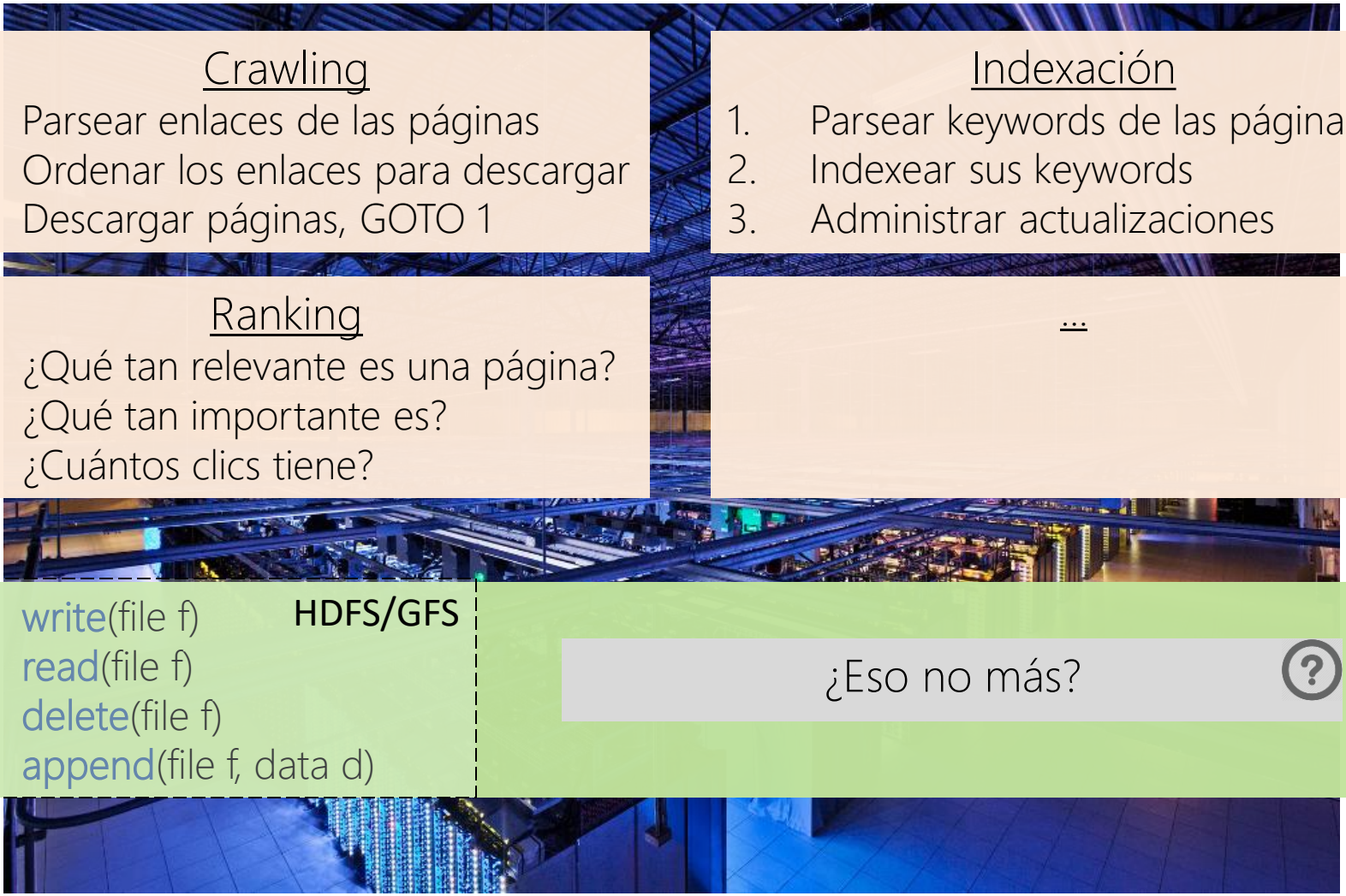
Meta-datos del nodo maestro se mantiene en memoria

- Cada bloque sólo necesita 64 bytes en memoria
- Datos del bloque pueden ser consultados en cada esclavo

Hadoop Distributed File System (HDFS)

- Versión open-source de GFS
  - Esclavo = "Data node"
  - Maestro = "Name node"

Implementando la búsqueda de Google



Crawling

1. Parsear enlaces de las páginas
2. Ordenar los enlaces para descargar
3. Descargar páginas, GOTO 1

Indexación

1. Parsear keywords de las páginas
2. Indexear sus keywords
3. Administrar actualizaciones

Ranking

1. ¿Qué tan relevante es una página?
2. ¿Qué tan importante es?
3. ¿Cuántos clics tiene?

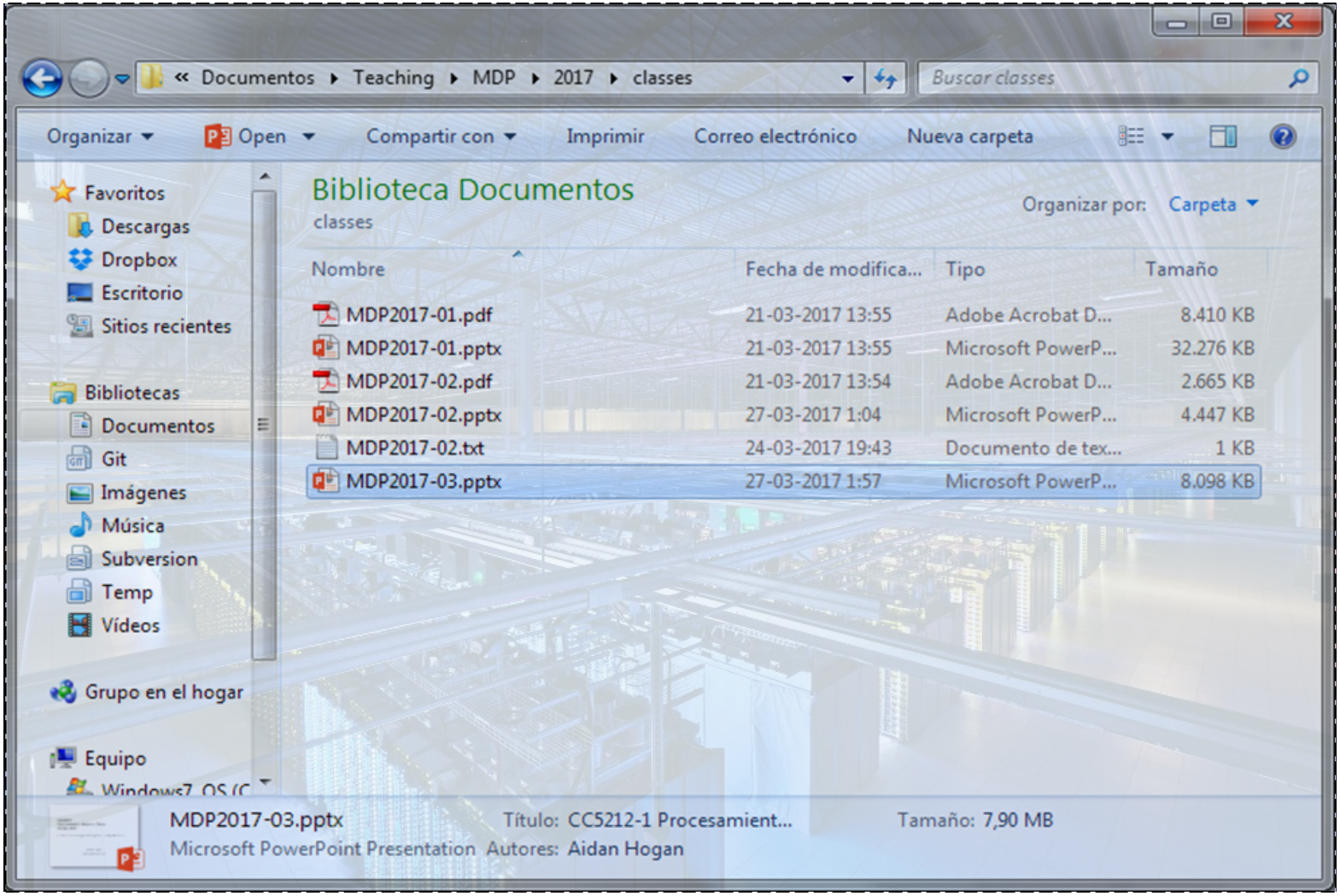
...

HDFS/GFS

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

¿Eso no más?

… solo un sistema de archivos (distribuido)



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

Empecemos con una tarea simple

¿Cómo podemos hacer un conteo de palabras distribuido?

¿Contar partes en la memoria principal de cada máquina?

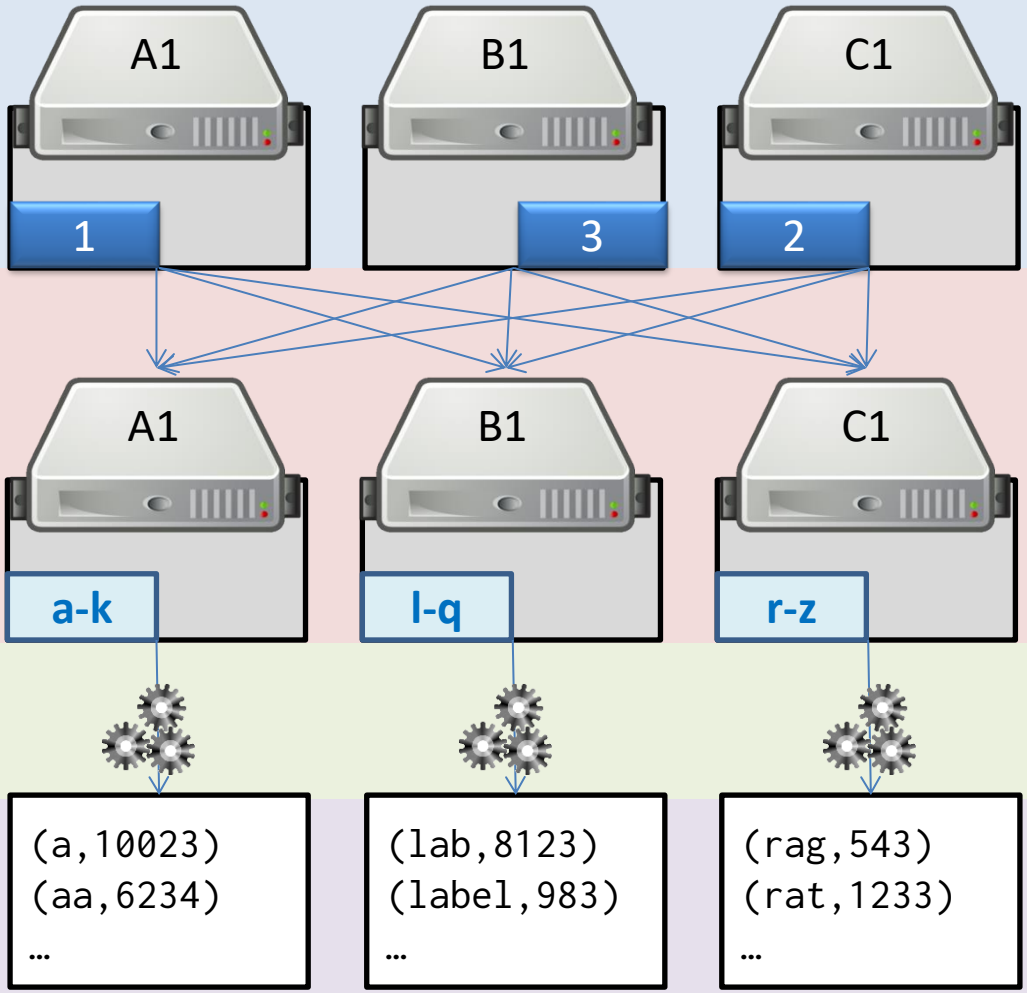
¿Pero si una parte no cabe en memoria (e.g., 40-gramas)?

¿Y cómo podemos hacer la unificación/suma de los conteos?

¿Contar partes usando el disco duro de cada máquina?

¿Y de nuevo, cómo podemos hacer la suma de los conteos?

Conteo de palabras distribuido



Entrada

Datos en GFS/HDFS

Partición

Transmisión

Ordenar

Contar

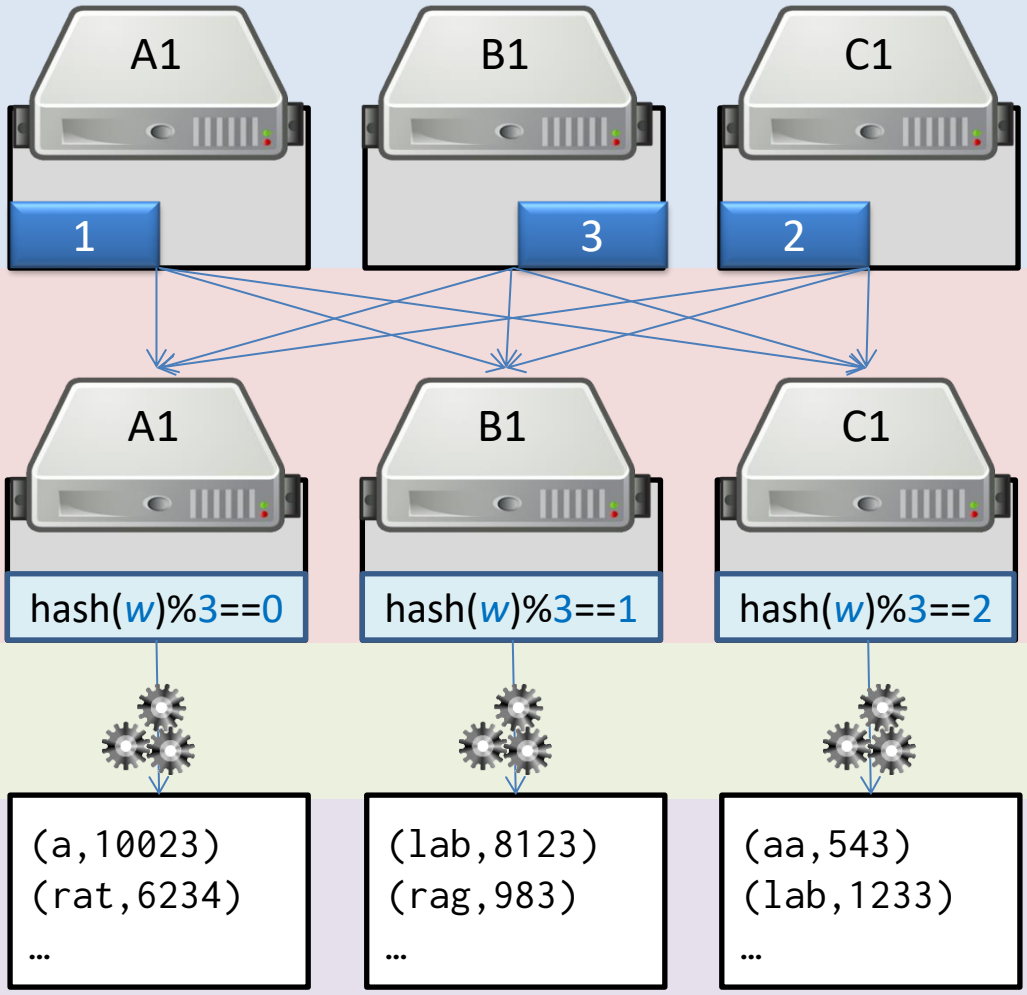
Salida

Datos en GFS/HDFS

¿Mejor método de partición?

$\text{HASH}(w)\%m$

Conteo de palabras distribuido



Entrada

Datos en GFS/HDFS

Partición

Transmisión

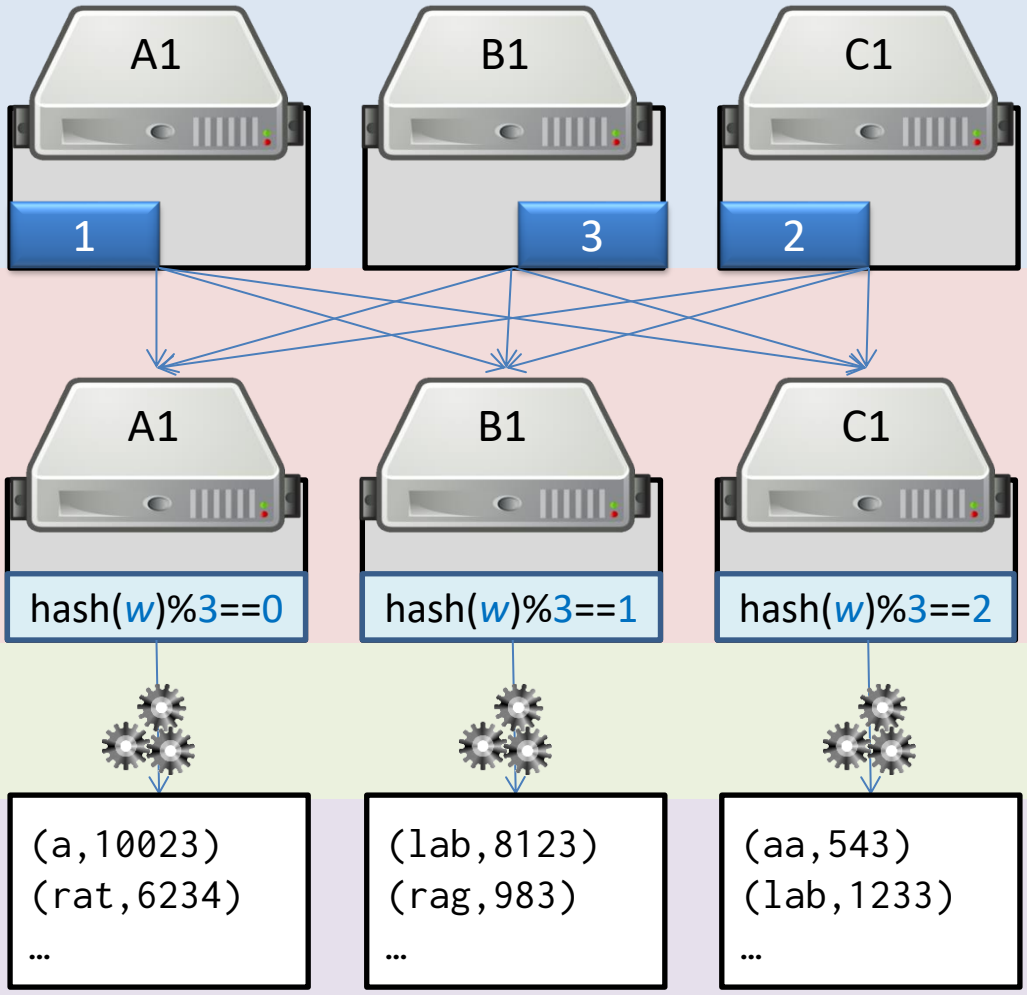
Ordenar

Contar

Salida

Datos en GFS/HDFS

MapReduce



Entrada

Datos en GFS/HDFS

Partición (Map)

Transmisión

Ordenar

Contar (Reduce)

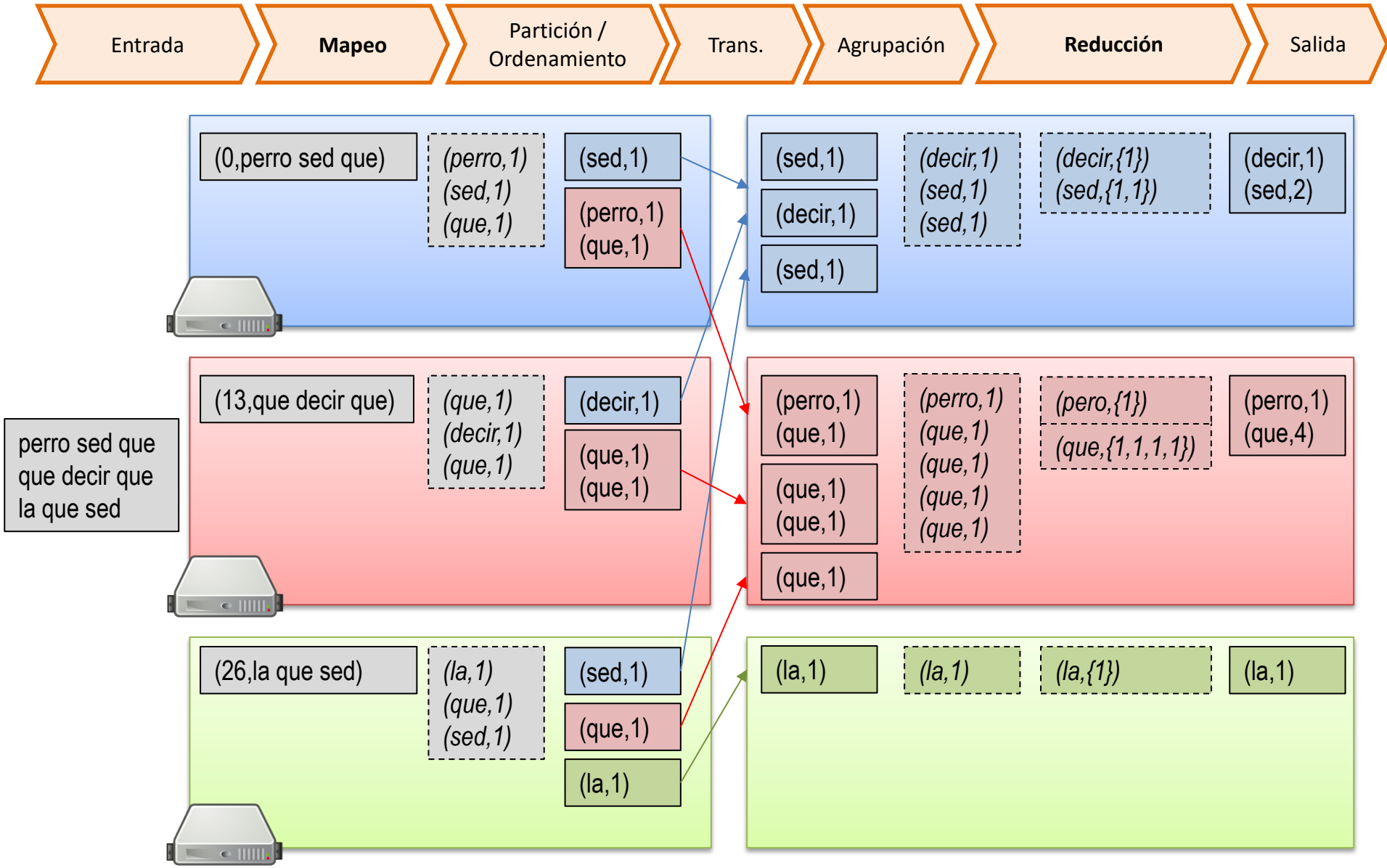
Salida

Datos en GFS/HDFS

MapReduce: Pseudocódigo de conteo de palabras

```
import map_reduce map_out reduce_out

main(String entrada, String salida)
    map_reduce(entrada, map, reduce, salida)

map(int numero, String linea) //lee de entrada
    for each palabra in linea
        map_out(palabra, 1)

// el framework ...
//    ... particiona por palabra
//    ... agrupa los valores de cada palabra
//    ... ejecuta reduce ...

reduce(String llave, int[] valores)
    reduce_out(llave, sum(valores)) //escribe a salida
```

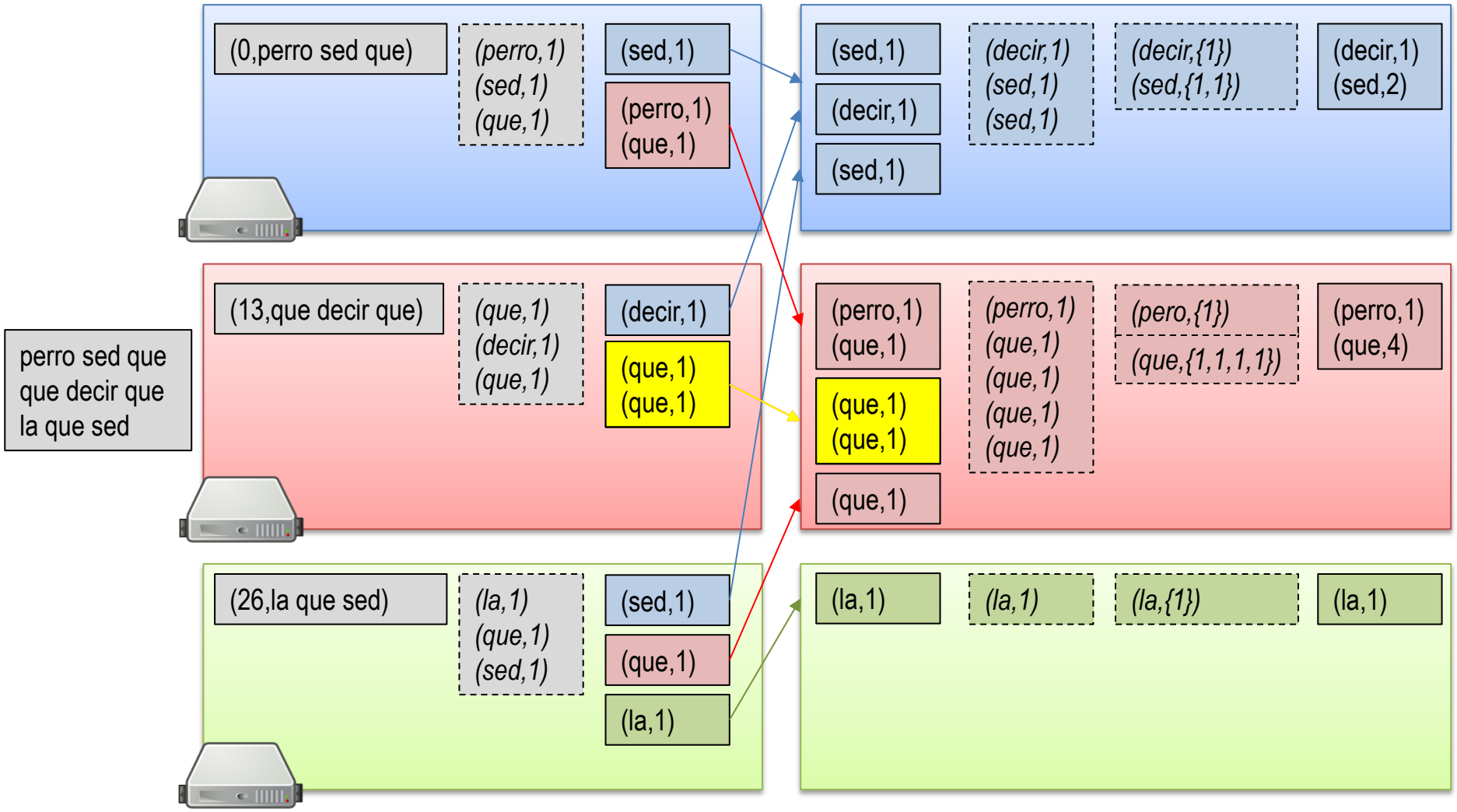
MapReduce: Conteo de Palabras



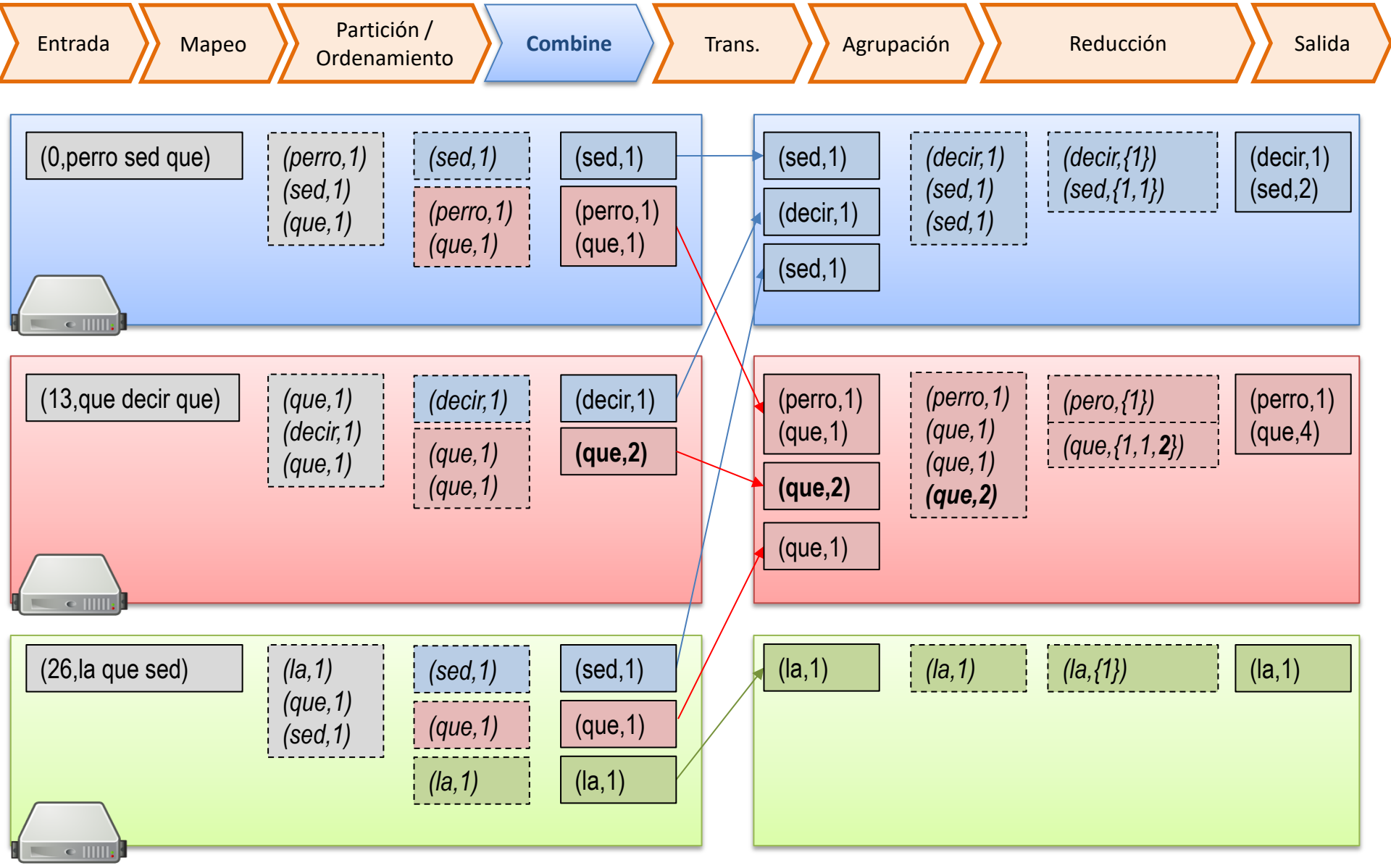
MapReduce:
Una optimización con el "combiner"

MapReduce: Conteo de Palabras

¿Hay algo que podríamos optimizar fácilmente aquí?



MapReduce (con el "Combiner")



MapReduce (con el "Combiner")

- Se define el "combiner" como una reducción
- A menudo, se puede usar la reducción misma como un combiner sin cambiar nada

```
import map_reduce map_out reduce_out

main(String entrada, String salida)
   map_reduce(entrada, map, reduce, reduce, salida)
// el primer reduce es el combiner

map(int numero, String linea) //lee de entrada
   for each palabra in linea
       map_out(palabra, 1)

// el framework ...
//    ... particiona por palabra
//    ... agrupa los valores de cada palabra
//    ... ejecuta reduce ...

reduce(String llave, int[] valores)
   reduce_out(llave, sum(valores))
```

MapReduce (con el "Combiner")

- No siempre se puede usar la reducción como un combiner
- Usando un conteo en vez de una suma, el combiner no funciona

```
import map_reduce map_out reduce_out

main(String entrada, String salida)
   map_reduce(entrada, map, reduce, reduce, salida)
// el primer reduce es el combiner

map(int numero, String linea) //lee de entrada
   for each palabra in linea
       map_out(palabra, 1)

// el framework ...
//    ... particiona por palabra
//    ... agrupa los valores de cada palabra
//    ... ejecuta reduce ...

reduce(String llave, int[] valores)
   reduce_out(llave, count(valores))
```

MapReduce (con el "Combiner")

¿Cuándo se puede usar la misma reducción como un combiner?

1. Cuando se produzcan pares con el mismo tipo de llave/valor que el mapeo (porque se mezclan ambos en la reducción)
2. Cuando la operación de reducción "#" sea
   - conmutativa: a#b = b#a
   - asociativa a#(b#c) = (a#b)#c = a#b#c

   es decir, cuando se puedan combinar los valores en cualquier orden con cualquier combinación de argumentos sin afectar el resultado final; por ejemplo:
   - +, ×, max, min, etc., están bien
   - promedio no: p(a,p(b,c)) ≠ p(p(a,b),c) ≠ p(a,b,c)

¡El combiner es opcional!

MapReduce:
Tareas más complejas

Tareas más complejas

- A veces se necesitan varias tareas
  - Cada tarea es un join/una agrupación
  - Se puede encadenar varias tareas así
- Se pueden tener varios mapeos con una reducción
  - Por ejemplo, para hacer un join sobre varios archivos
- Una reducción necesita al menos un mapeo
  - Incluso si el mapeo simplemente "copia" cada par
  - El mapeo inicia la partición/transmisión/ordenamiento

Ventas totales

Recibos.tsv

Recibo	Producto
R1401	P306
R1401	P306
R1401	P504
R1402	P007
R1402	P306
R1403	P306
R1403	P504
...	...

Tiempo.tsv

Recibo	Tiempo
R1403	19:00
R1401	18:59
R1402	19:01
...	...

Productos.tsv

Producto	Nombre	Valor
P306	Zanahoria 500g	500
P504	CocaCola 3L	1400
P007	Comfort	1200
...	...	

Entrada

¿Computar ventas totales por hora?

(Se asume que los precios no cambien)

(Hay varias soluciones; aquí va una ...)

VentasPorHora.tsv

Hora	Suma
...	...
18:00–18:59	2400
19:00–19:59	3600
...	...

Salida

Ventas totales: Una solución

Recibos.tsv

Recibo	Producto
R1401	P306
R1401	P306
R1401	P504
R1402	P007
R1402	P306
R1403	P306
R1403	P504
...	...

Tiempo.tsv

Recibo	Tiempo
R1403	19:00
R1401	18:59
R1402	19:01
...	...

Productos.tsv

Producto	Nombre	Valor
P306	Zanahoria 500g	500
P504	CocaCola 3L	1400
P007	Comfort	1200
...	...	

Entrada

- Map_{1A} (entrada: `Recibos.tsv`)
  - $(R,P) \mapsto \{(R,P)\}$
- Map_{1B} (entrada: `Tiempo.tsv`)
  - $(R,T) \mapsto \{(R,\text{hora}(T))\}$
- Reduce_1 (entrada: Map_{1A}, Map_{1B})
  - $(R, [P_1,\ldots,P_n,H]) \mapsto \{(P_1,H), \ldots, (P_n,H)\}$
- Map_{2A} (entrada: `Productos.tsv`)
  - $(P,(N,V)) \mapsto \{(P,V)\}$
- Map_{2B} (entrada: Reduce_1)
  - $(P,H) \mapsto \{(P,H)\}$
- Reduce_2 (entrada: Map_{2A}, Map_{2B})
  - $(P, [H_1,\ldots,H_n,V]) \mapsto \{(H_1,V),\ldots,(H_n,V)\}$
- Map_3 (entrada: Reduce_2)
  - $(H,V) \mapsto \{(H,V)\}$
- Reduce_3 (entrada: Map_3)
  - $(H, [V_1,\ldots,V_n]) \mapsto \{(H,\Sigma_{i=1}^{n} V_i)\}$
  - salida: `VentasPorHora.tsv`

Ventas totales: Combiner?

Recibos.tsv

Recibo	Producto
R1401	P306
R1401	P306
R1401	P504
R1402	P007
R1402	P306
R1403	P306
R1403	P504
...	...

Tiempo.tsv

Recibo	Tiempo
R1403	19:00
R1401	18:59
R1402	19:01
...	...

Productos.tsv

Producto	Nombre	Valor
P306	Zanahoria 500g	500
P504	CocaCola 3L	1400
P007	Comfort	1200
...	...	

¿Se puede usar la reducción como un combiner aquí?

Entrada

- Map_{1A} (entrada: `Recibos.tsv`)
  - $(R,P) \mapsto \{(R,P)\}$
- Map_{1B} (entrada: `Tiempo.tsv`)
  - $(R,T) \mapsto \{(R,\text{hora}(T))\}$
- Reduce_1 (entrada: Map_{1A}, Map_{1B})
  - $(R, [P_1,\ldots,P_n,H]) \mapsto \{(P_1,H), \ldots, (P_n,H)\}$

No

- Map_{2A} (entrada: `Productos.tsv`)
  - $(P,(N,V)) \mapsto \{(P,V)\}$
- Map_{2B} (entrada: Reduce_1)
  - $(P,H) \mapsto \{(P,H)\}$
- Reduce_2 (entrada: Map_{2A}, Map_{2B})
  - $(P, [H_1,\ldots,H_n,V]) \mapsto \{(H_1,V),\ldots,(H_n,V)\}$

No

- Map_3 (entrada: Reduce_2)
  - $(H,V) \mapsto \{(H,V)\}$
- Reduce_3 (entrada: Map_3)
  - $(H, [V_1,\ldots,V_n]) \mapsto \{(H,\Sigma_{i=1}^{n} V_i)\}$
  - salida: `VentasPorHora.tsv`

Sí: Reduce_3

MapReduce: Programación

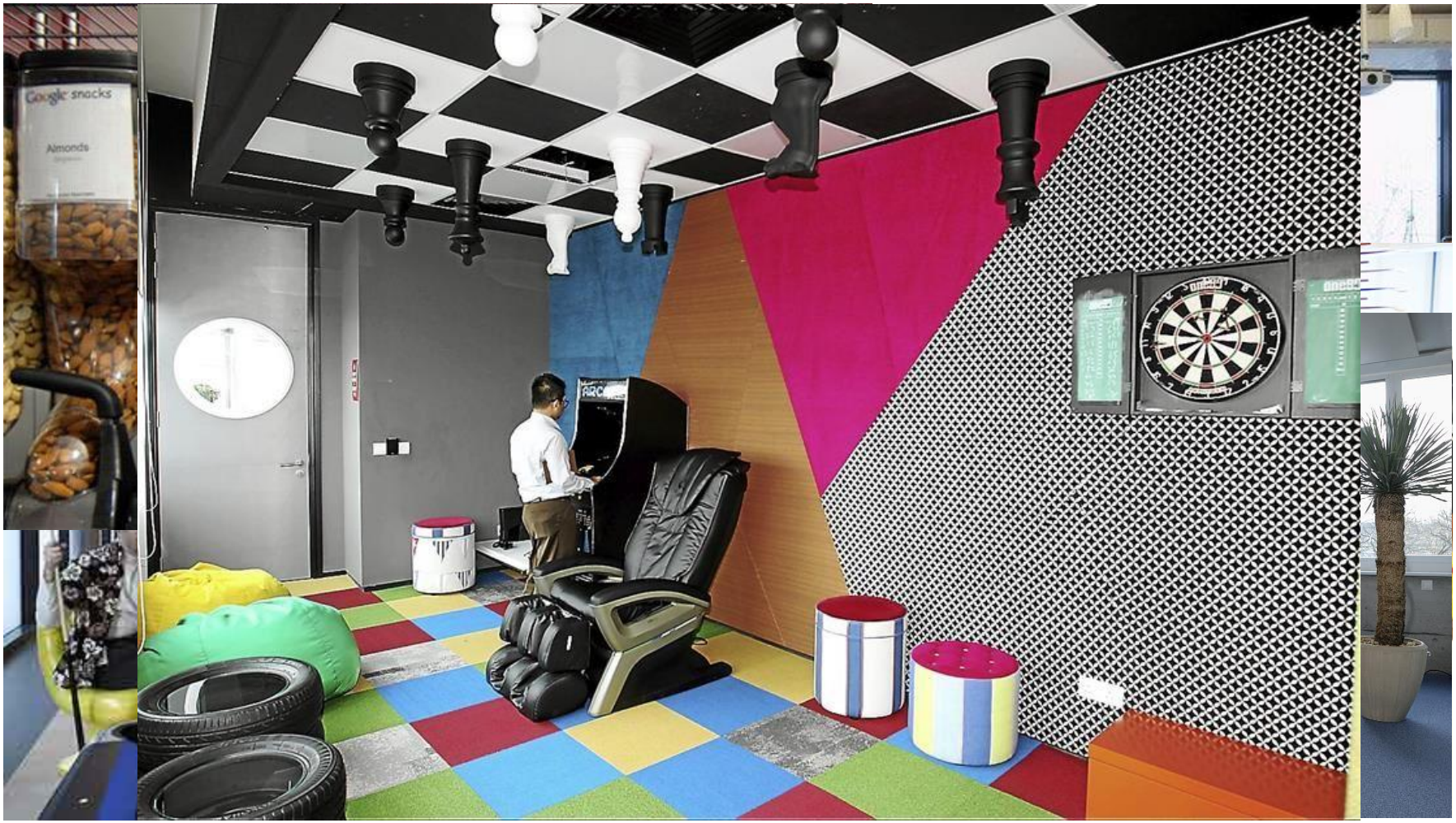
MapReduce: Beneficios para los programadores

- **Se encarga de la implementación a bajo nivel:**
  - Fácil manejo de inputs y outputs
  - No es necesario el manejo de comunicación entre máquinas
  - No es necesario implementar ordenamiento y agrupación

- **Máquinas abstractas (transparencia):**
  - Tolerancia a fallas
  - Ubicaciones físicas abstractas
  - Agregar / remover máquinas
  - Balanceo de carga

MapReduce: Beneficios para los programadores

Más tiempo para …



HDFS ≈ DFS

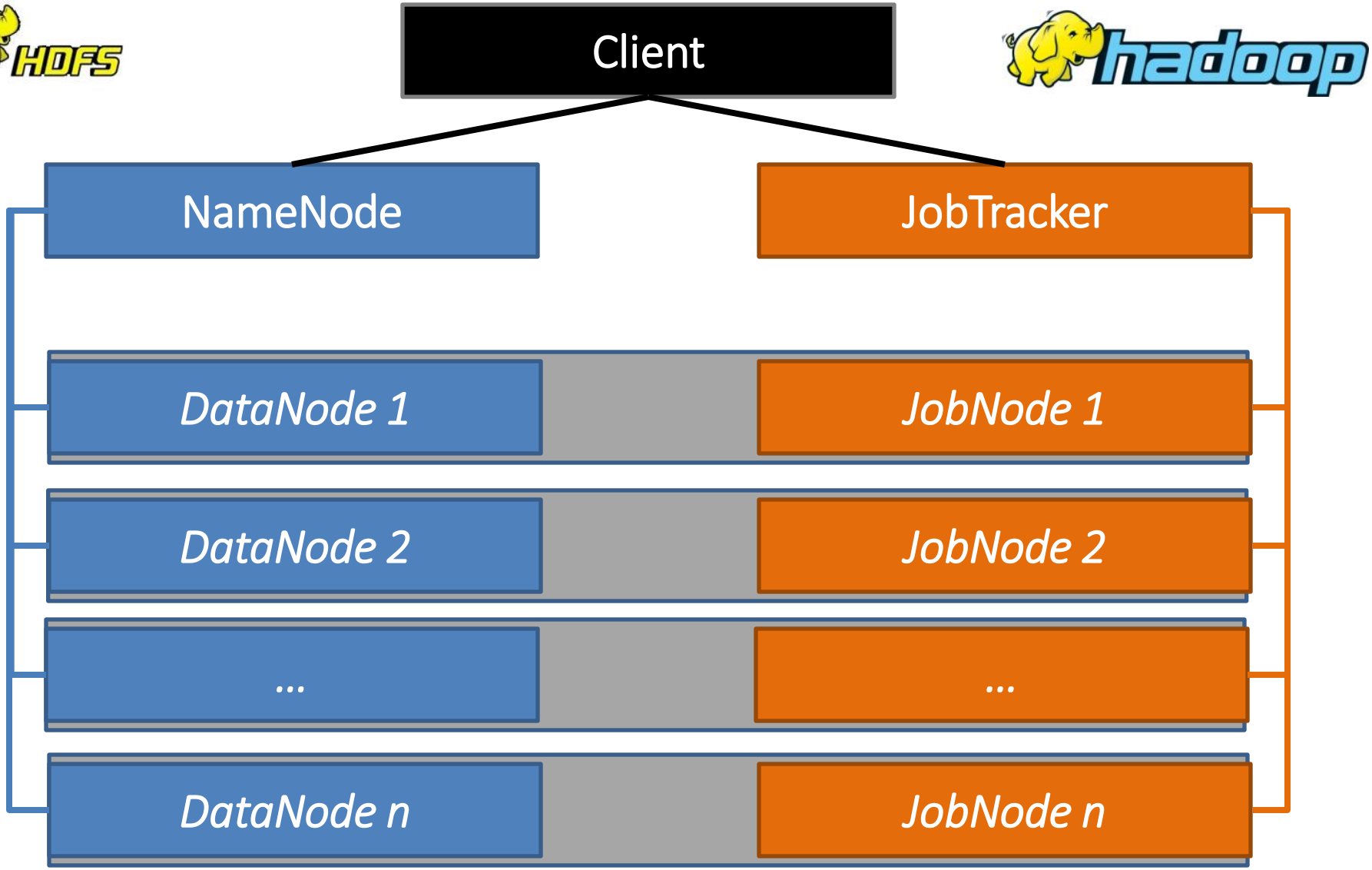
HADOOP ≈ MAPREDUCE

Hadoop

- Implementación open source de MapReduce
- Basado en HDFS

Hadoop: Versión Open Source de MapReduce

HDFS / Hadoop: Arquitectura



Preguntas?