

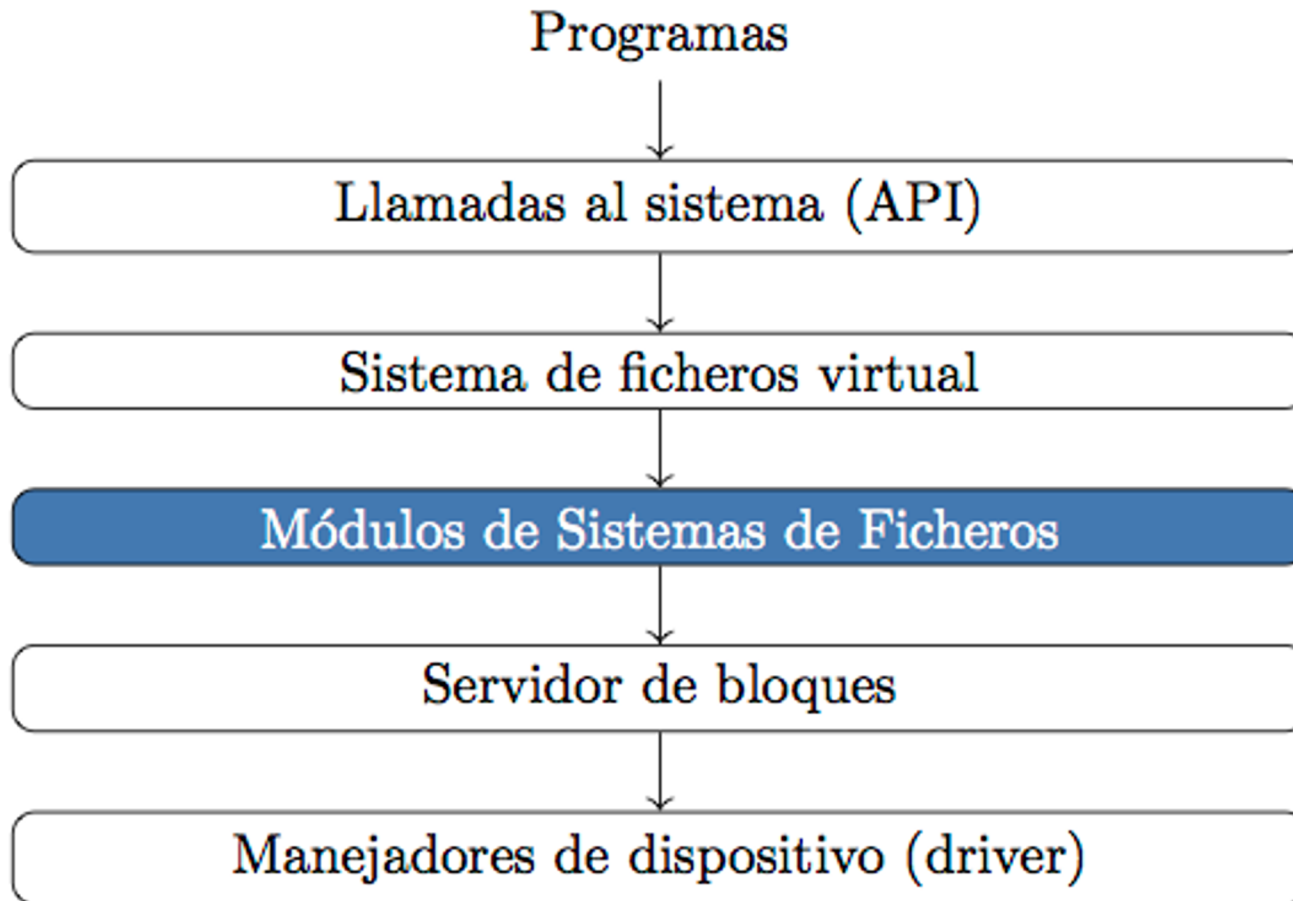


Sistemas Operativos

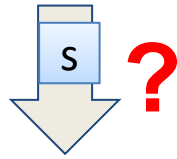
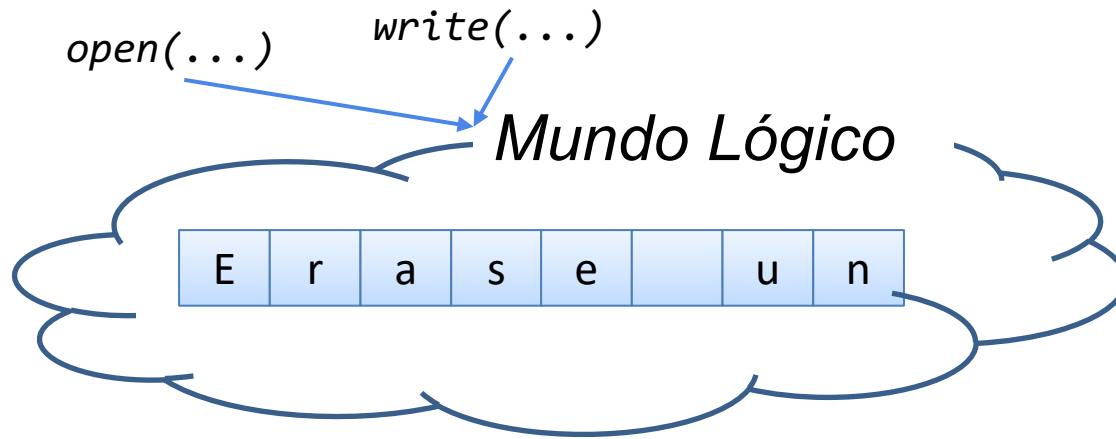
Curso
2020-2021

Módulo 2: Gestión de archivos y directorios
De la visión lógica a la física

Gestión de ficheros



Aterrizando en el mundo físico



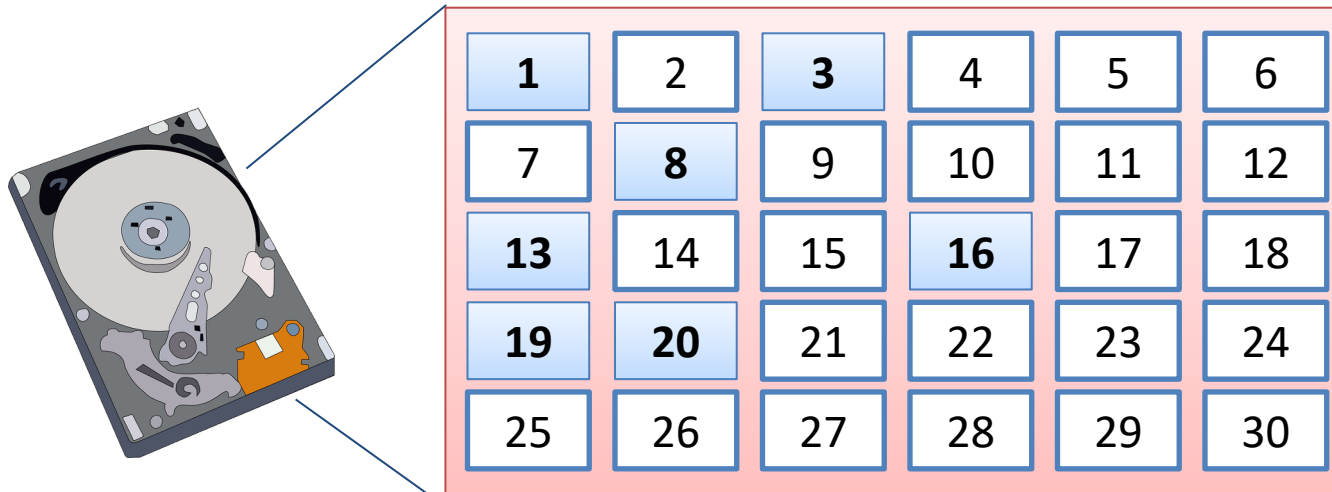
Mundo Físico

- ¿Dónde escribimos cada nuevo byte del fichero?
- ¿Cómo encontramos la información que escribimos anteriormente?



Modelo de dispositivo de almacenamiento

- Consideramos el dispositivo como una secuencia de bloques de **tamaño fijo**
 - Tamaño habitual 4KB
 - Operaciones permitidas: lectura/escritura de n bytes consecutivos en un bloque
 - Ejemplo: Escribir “abcd” a partir del byte 33 del bloque 10
 - Y, ¿cómo borramos información de un bloque?
 - ¿Escribiendo 0's ? Mala idea



Entonces, ¿de qué va esto?

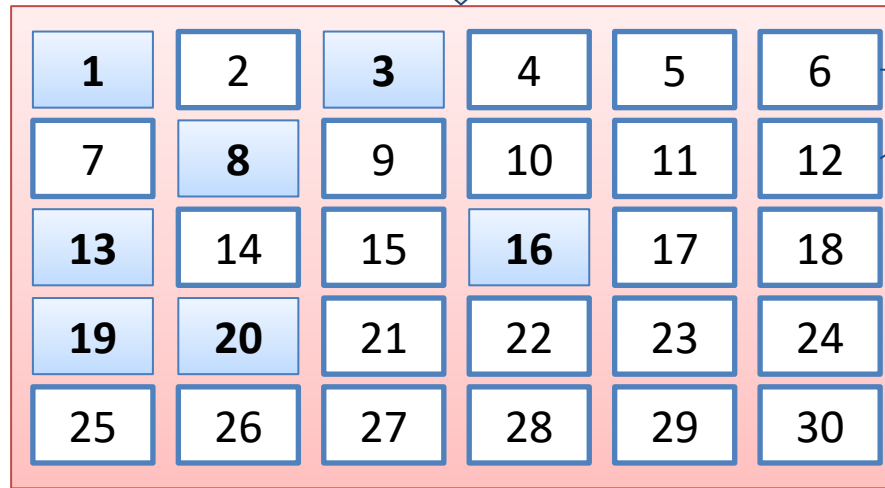


Visión lógica

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21		
E	r	a	s	e			u	n	a			v	e	z		u	n		s	i	s	t	e



Visión física



4 KB (4096 bytes) por bloque

- Asignar los bytes de un fichero a los bloques del fichero:
 - **Idea1** (muy mala): asigno cada byte del fichero donde me dé la gana

<i>Byte fichero</i>	<i>Bloque disco</i>	<i>Byte en el bloque</i>
0 ('E')	8	313
1 ('r')	12	4010

Consideraciones sobre *Idea1*



- **Ejemplo:** fichero de texto con 10.000 caracteres → necesito una tabla de 10.000 entradas, con 2 enteros en cada entrada
 - ¿Y dónde guardo esa tabla? En el disco claro....
 - Y eso para cada fichero del sistema
- ¿Cómo sabemos si un *byte* de un bloque está libre u ocupado?
 - ‘*Si está a 0, está libre*’....¡¡ que ya hemos dicho que no !!
- Y, ¿dónde está el nombre del fichero? ¿Y el directorio en el que está?

Recapitulando: tareas pendientes

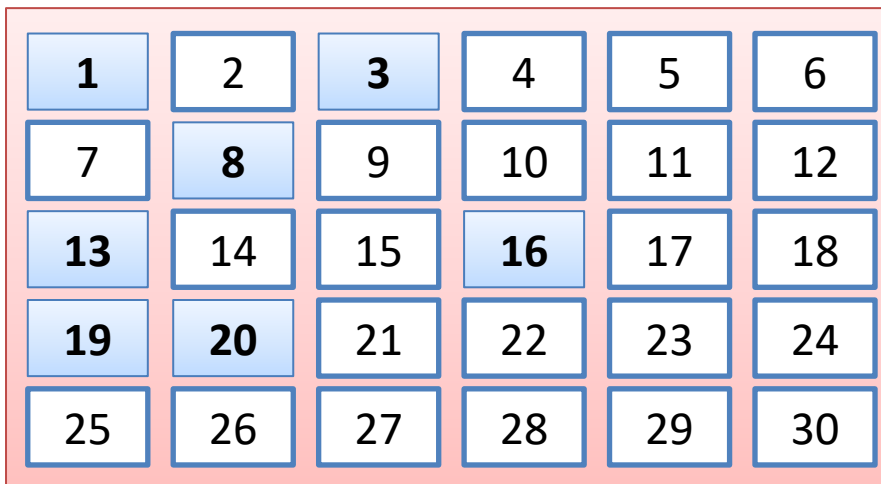
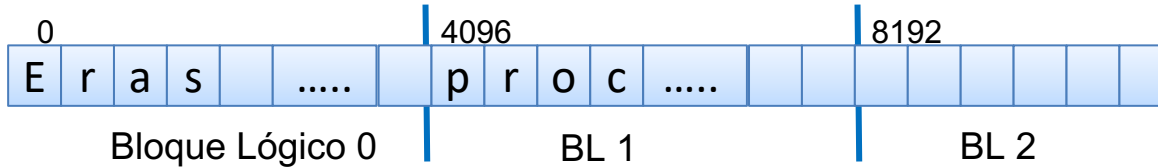


- Encontrar una *relación* que, dado un *byte* lógico de un fichero, nos diga la posición del *byte físico*:
 - ¿Es factible a nivel de *byte*? ¿Alguna idea?
- Gestión del espacio libre en disco
 - ¿Cómo saber si un bloque de disco está libre/ocupado?
 - ¿Y un byte dentro de ese bloque?
- ¿Cómo representar directorios?
 - ¿Qué información contendrá?
 - ¿Y otros tipos de ficheros?



Encontrando respuestas: lógico -> físico

- **Propuesta2:** simplificación razonable (y realista): la traducción se hace a nivel de bloque y no de byte
 - Bloque lógico: secuencia de N bytes consecutivos en la visión lógica (bloque físico, también de N bytes)



<i>(BL, byte)</i>	<i>(BF, byte)</i>
(0,0) ('E')	(8, 0)
(0,1) ('r')	(8, 1)
(1,2) ('o')	(19,2)

Índice de byte **NO** cambia

Efectos laterales de *propuesta2*



- Sólo es necesario gestionar la traducción de BL a BF, no *byte a byte*
 - 4096 veces más fácil para bloques de 4KB....
 - El contenido de un fichero se almacenará en una serie de bloques físicos (todos del mismo tamaño)
- Espacio libre gestionado también a nivel de bloque
 - Sólo hay que indicar qué bloques están libre
 - Una vez más...¿cómo indicarlo? Volveremos sobre esto más tarde
- Desperdicio de espacio
 - El último bloque de cada fichero estará, muy probablemente, algo infrautilizado

Compro *propuesta2*....

Preguntas relevantes por contestar



- Creamos un fichero llamado '*prueba.txt*' con un contenido inicial de 5000 *x*'s
 - ¿En qué zonas del disco escribo las *x*'s?
 - ¿Cómo relaciono el nombre del fichero con su ubicación en disco?
 - Si, más adelante, abro el fichero (*open()*) y leo (*read()*) el byte de la posición 1234, ¿dónde lo busco en el disco?
 - ¿Cómo borro 1000 *x*'s a partir de la posición 2000?
 - Si añado 1000 *z*'s, ¿cómo encuentro espacio libre en el disco para asignarlo a este fichero?

Ejercicio de clase: DIY

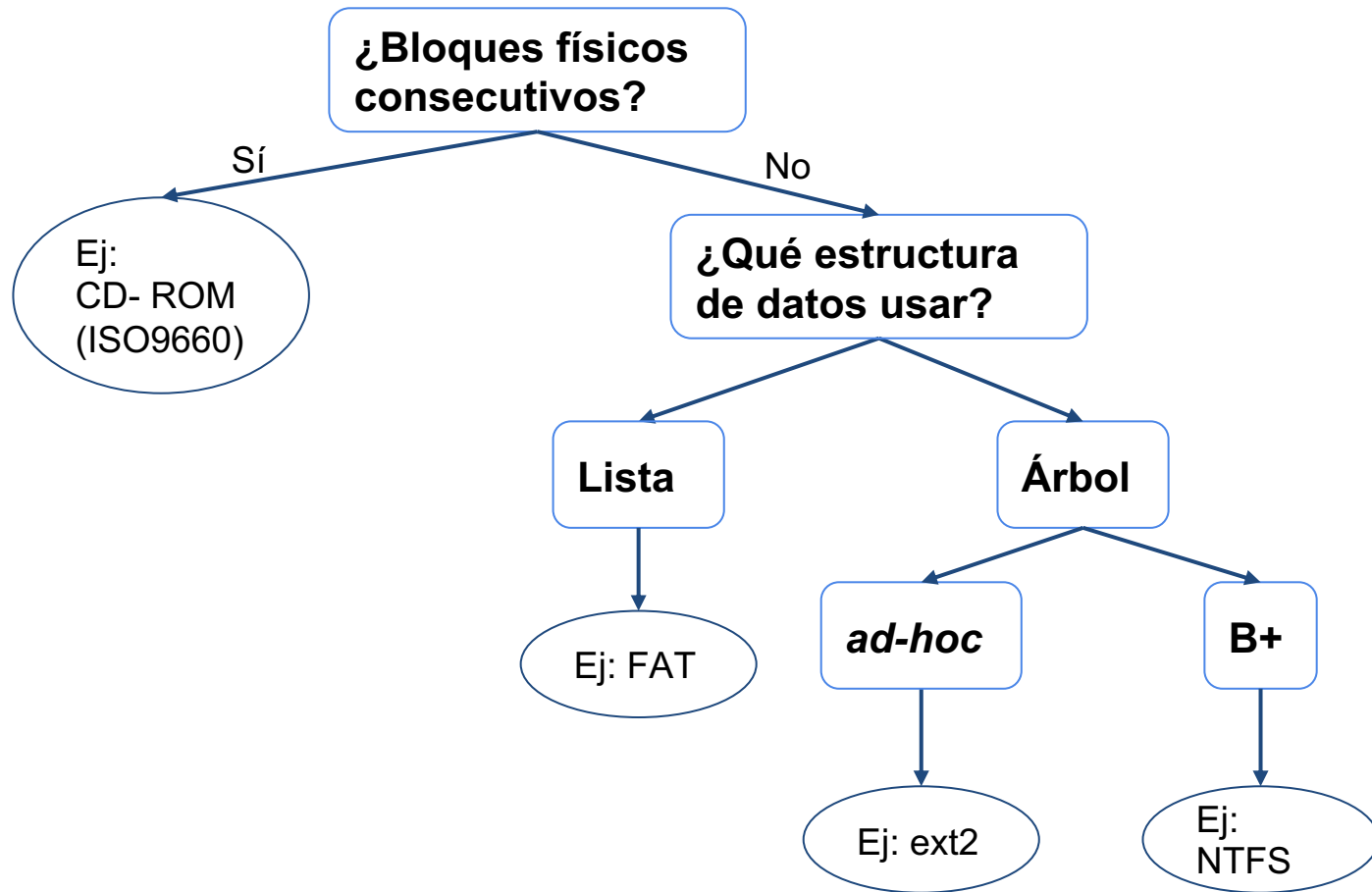


- Diseña (no implementa) tu propio sistema de ficheros
 - Asumiendo *propuesta2* (asignación a nivel de bloques)
 - Con un único directorio que alberga todos los ficheros
 - Respondiendo a todas las preguntas anteriores (al menos, a nivel ‘conceptual’; no hace falta una implementación final)
- Comprueba que tu sistema permite crear, borrar y editar ficheros.
 - Y que es posible recuperar la información cuando reiniciemos el sistema
- Intenta dar una idea de la eficiencia de tu sistema: ¿cuántos acceso a disco implica una lectura de un fichero?
 - Indica qué información permanece en disco y cuál puede estar (temporalmente) en memoria



Desempolvando EDA

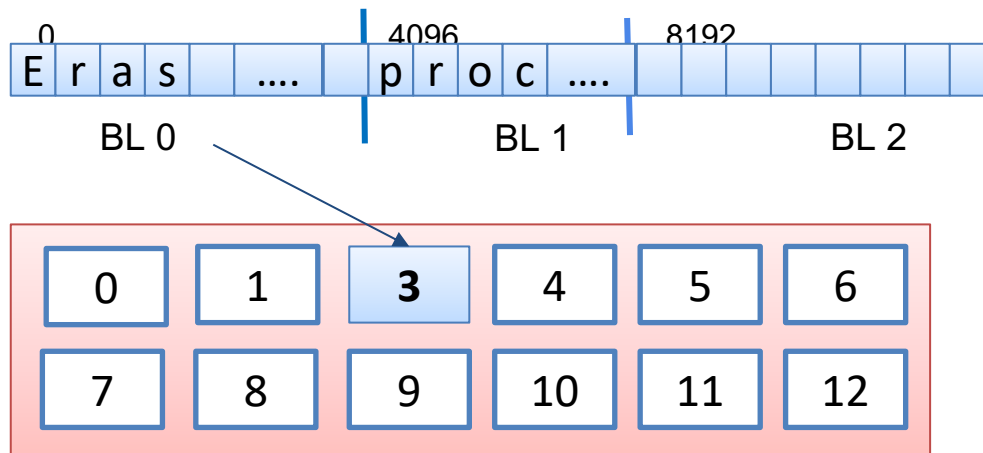
- ¿Cómo modelarías e implementarías esa traducción?





Bloques físicos consecutivos

- Todos los bloques físicos de un fichero deben estar consecutivos en el dispositivo
 - La traducción se resume en saber en qué bloque físico está el primer bloque lógico



Ejemplo real:
sistema de ficheros
ISO-9660 utilizado
en *CD-ROMs*

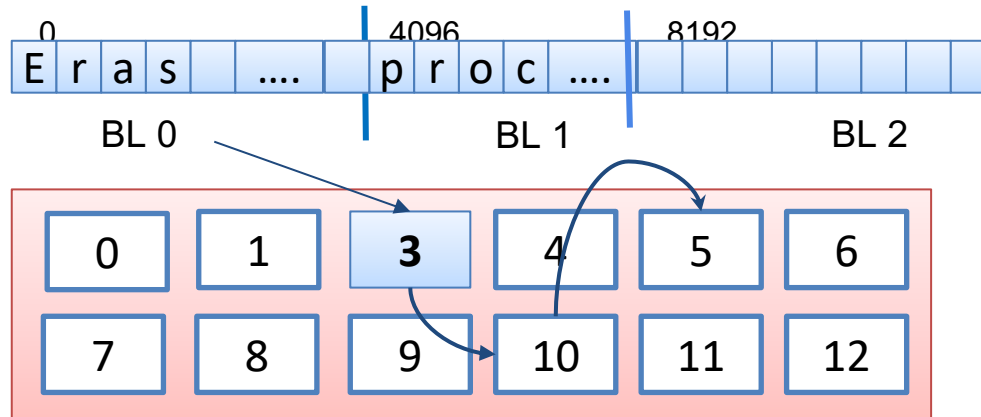
A pensar...

- ¿En qué bloque físico está la 'r' del bloque lógico 1?
- ¿Qué información se debe almacenar (en disco) de cada fichero (además de su contenido)?
- ¿Qué ocurre si queremos añadir contenido a un fichero?
- ¿Y si queremos borrar el BL 1?



Fichero como lista de bloques

- Eliminamos la restricción de bloques físicos consecutivos
 - Organizamos los bloques de un fichero como una *lista enlazada* de bloques físicos



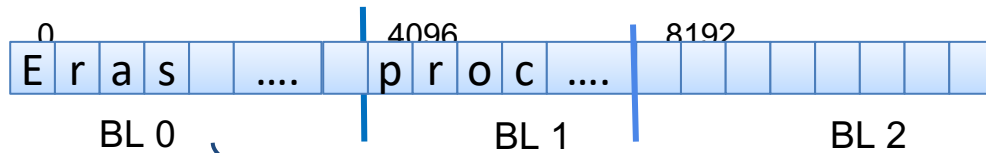
A pensar...

- ¿En qué bloque físico está la 'r' del bloque lógico 1?
- ¿Qué información se debe almacenar (en disco) de cada fichero (además de su contenido)?
- ¿Qué ocurre si queremos añadir contenido a un fichero?
- ¿Y si queremos borrar el BL 1?

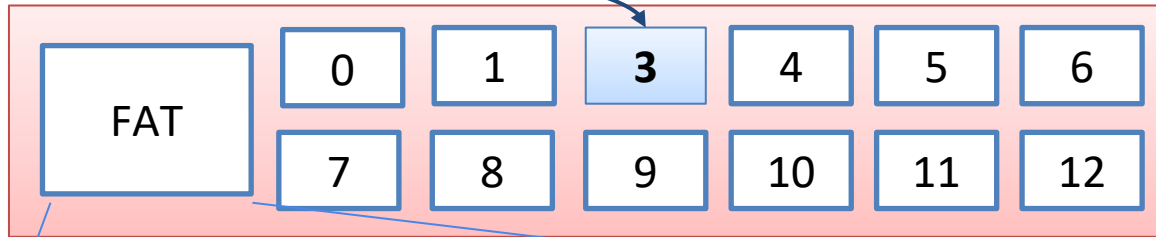


Ejemplo real: FAT

- El sistema FAT es un ejemplo concreto del uso de listas enlazadas
 - Particularidad: el *puntero a siguiente* se almacena en una tabla (FAT) aparte, no al final de cada de bloque



BL 0 → BF 3
BL 1 → BF 10
BL 2 → BF 5



0	1	2	3	4	5	6	7	8	9	10	11
L	L	4	10	eof	eof	L	L	11	L	5	15

Tabla FAT

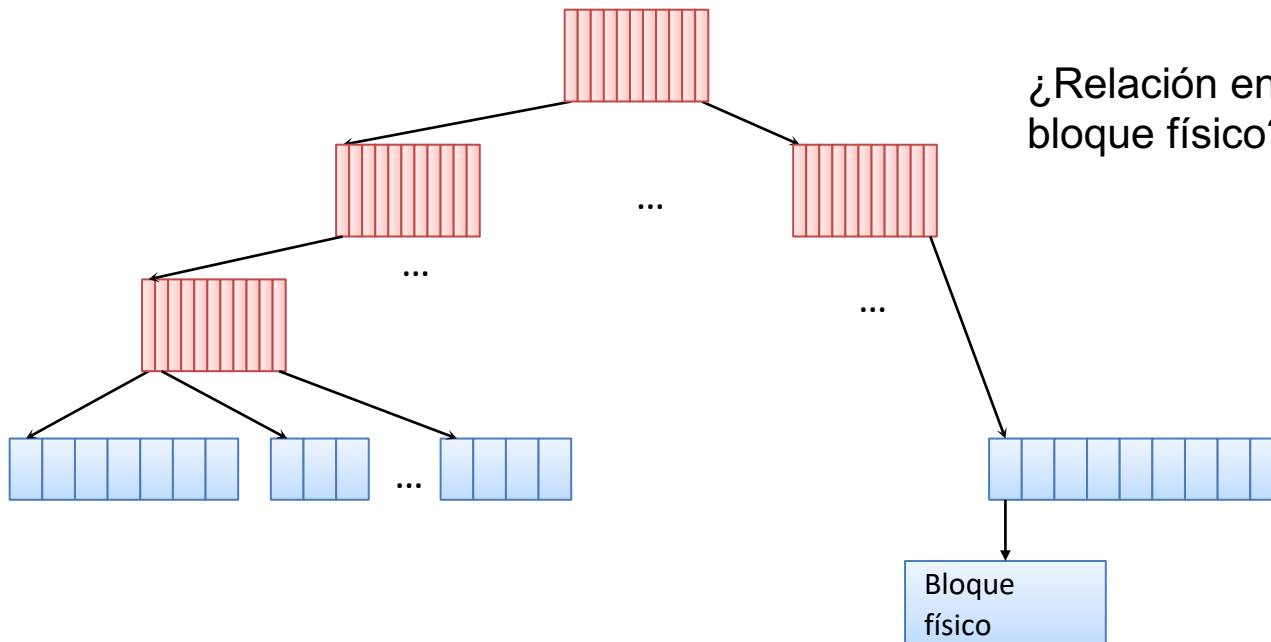
L → Ese bloque físico está **libre** (0,1...)
eof → Último bloque de un fichero

Única duda por resolver: ¿dónde está el **nombre** del fichero?



Fichero como árbol de bloques

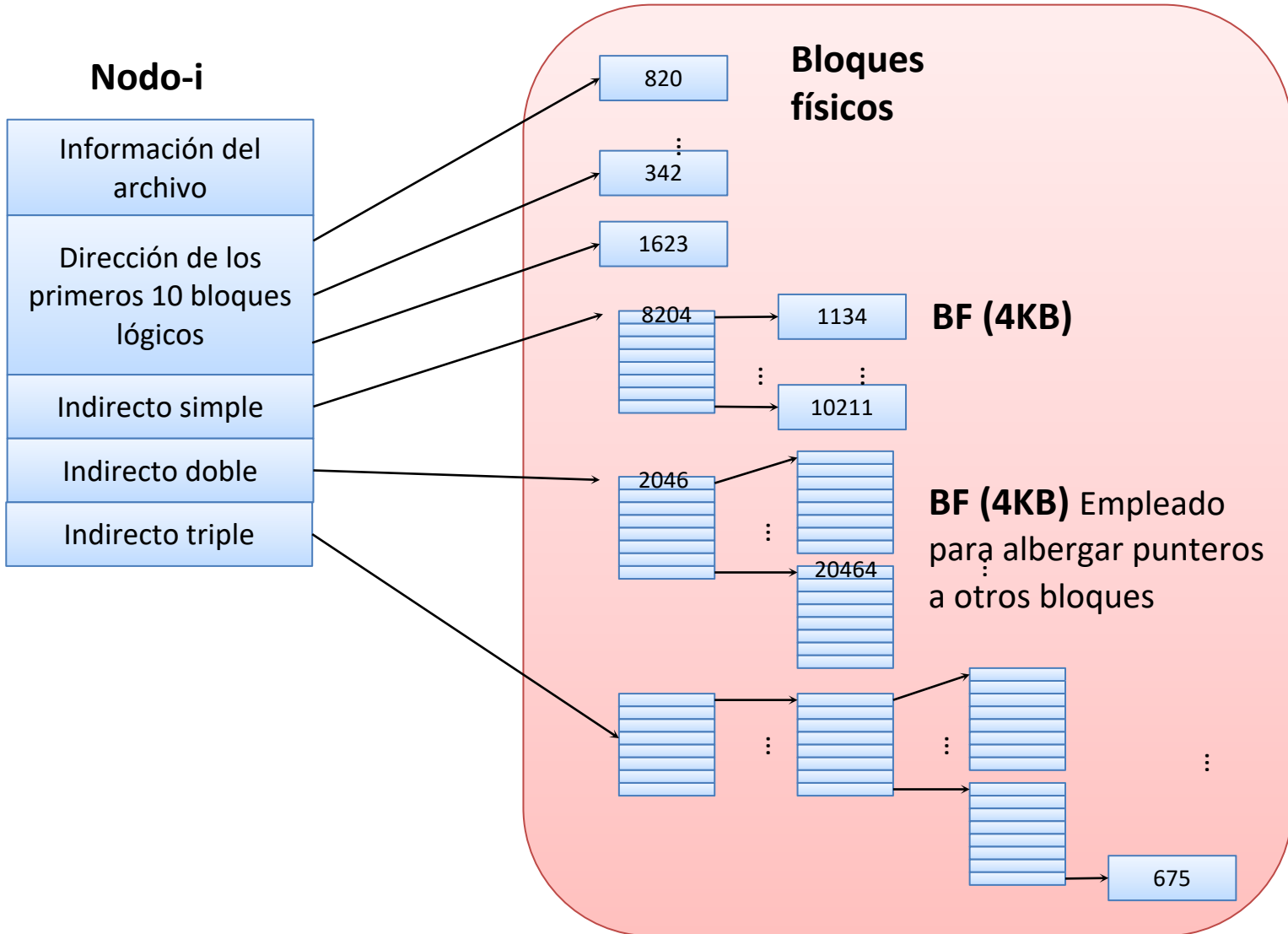
- Cada fichero se representará con un árbol de bloques físicos
- Nuevamente, múltiples alternativas de implementación, pero concepto común
 - Árboles balanceados, B, B+



¿Relación entre bloque lógico y bloque físico?



Ejemplo real: tipo Unix



Más sobre nodos-i



- Hay un nodo-i por cada fichero (o directorio) del sistema de ficheros
- Un nodo-i NO contiene el nombre del fichero que representa
 - Pero sí otra información como propietario, fechas, permisos, tamaños...
 - Y, por supuesto, los punteros directos, indirectos...
- ¿Dónde se almacenan los nodos-i? ¿Cuántos hay? ¿Se pueden crear dinámicamente?
- Ejercicio: si los bloques son de 2KB, propón un posible nodo-i (como los anteriores) para un fichero de 123KB?
 - ¿Cómo saber en qué bloque físico está el bloque lógico 12?

nodos-i y gestión del espacio libre



- Un nodo-i nos dice qué bloques físicos pertenecen a un fichero
 - Pero no dice nada de qué bloques físicos están libres
- FAT era *autocontenida*: tenía las dos informaciones
- ¿Ideas para representar el espacio libre?
 - Lista enlazada de bloques libres
 - Mapa de bits
- En los sistemas tipo Unix se usa el concepto de **mapa de bits**
 - Tabla con una entrada por bloque físico
 - Cada entrada es un solo bit: libre u ocupado
- Se usan 2 mapas: uno para controlar los bloques libres u otro para indicar qué nodos-i están libres

Contenido de un disco FAT y nodos- i



FAT



UNIX



- Sea un disco de 1GB con bloques de 4KB:
 - En FAT32, cada entrada de la tabla es de 32 bits. ¿Cuánto ocupa la tabla FAT?
 - Si un nodo-i ocupa 64 bytes, se limita el máximo número de ficheros a 1024. Se destina un bloque al *Boot* y otro bloque al *Superbloque*. ¿Cuánto ocupa el mapa de bits? (simplificación: ocupa un número entero de bloques)

Cubriendo el *gap* lógico-físico



- Cada sistema de ficheros implementa su versión de *read,write...*
 - Pero el esqueleto es muy similar

```
int sys_read(.....,int size,void* buf,...) {
    first_logic_byte = file.curPos;
    last_logic_byte = file.curPos +size -1;
    listaBLs = f1(first_logic_byte,last_logic_byte,BLOCKSIZE);
    foreach BL in listaBLs
        listaBFs.insert(traduce(BL))

    (primerOffset,lastOffset) = f2(file.curPos,size,BLOCKSIZE);
    addr = leerBloquesFísicos(listaBFs);
    copiaMem(addr,buf,primerOffset,lastOffset,size);
    ..... // comprobaciones de si quedaban size
}
```

¿Qué hay de los directorios?



- Es la entidad que relaciona el nombre de un fichero con su ubicación en el disco
- Es un tipo de fichero, pero no un *fichero regular*
 - Fichero regular: ficheros de texto, Excel, vídeos...
 - Directorio: fichero con una estructura concreta
- En general, un directorio es una lista en la que cada elemento tiene el **nombre del fichero** y....
 - ¿Qué información más debe contener en un sistema FAT?
 - ¿Y en un sistema basado en nodos-i?
- Los directorios ocupan bloques físicos (de datos) exactamente igual que los ficheros regularres

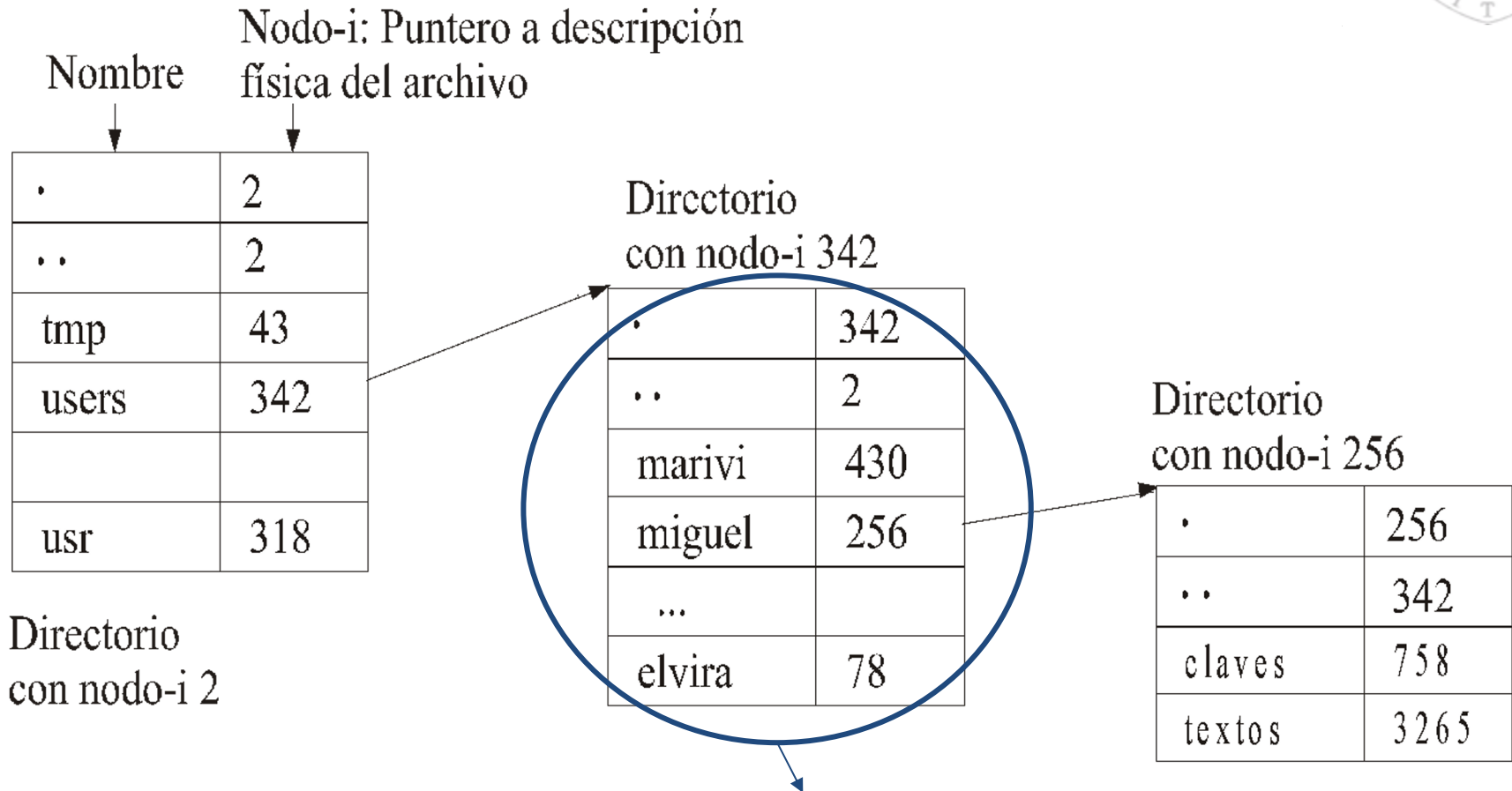


Nombres jerárquicos

- Nombre absoluto: especificación del nombre respecto a la raíz (/ en LINUX, \ en Windows).
- Nombre relativo: especificación del nombre respecto a un directorio distinto del raíz
 - Ejemplo: (Estamos en /users/) miguel/claves
 - Relativos al dir. de trabajo o actual: aquel en el se está al indicar el nombre relativo. En Linux se obtiene con `pwd`
- Directorios especiales:
 - Directorio de trabajo `'.'`
 - Ejemplo: `cp /users/miguel/claves .`
 - Directorio *padre* `'..'`
 - Ejemplo: `ls ..`
 - Directorio `HOME`: el directorio base del usuario



Interpretación de nombres en LINUX. I



Bloque de datos (físico) con el contenido del directorio.

Interpretación de nombres en LINUX. II

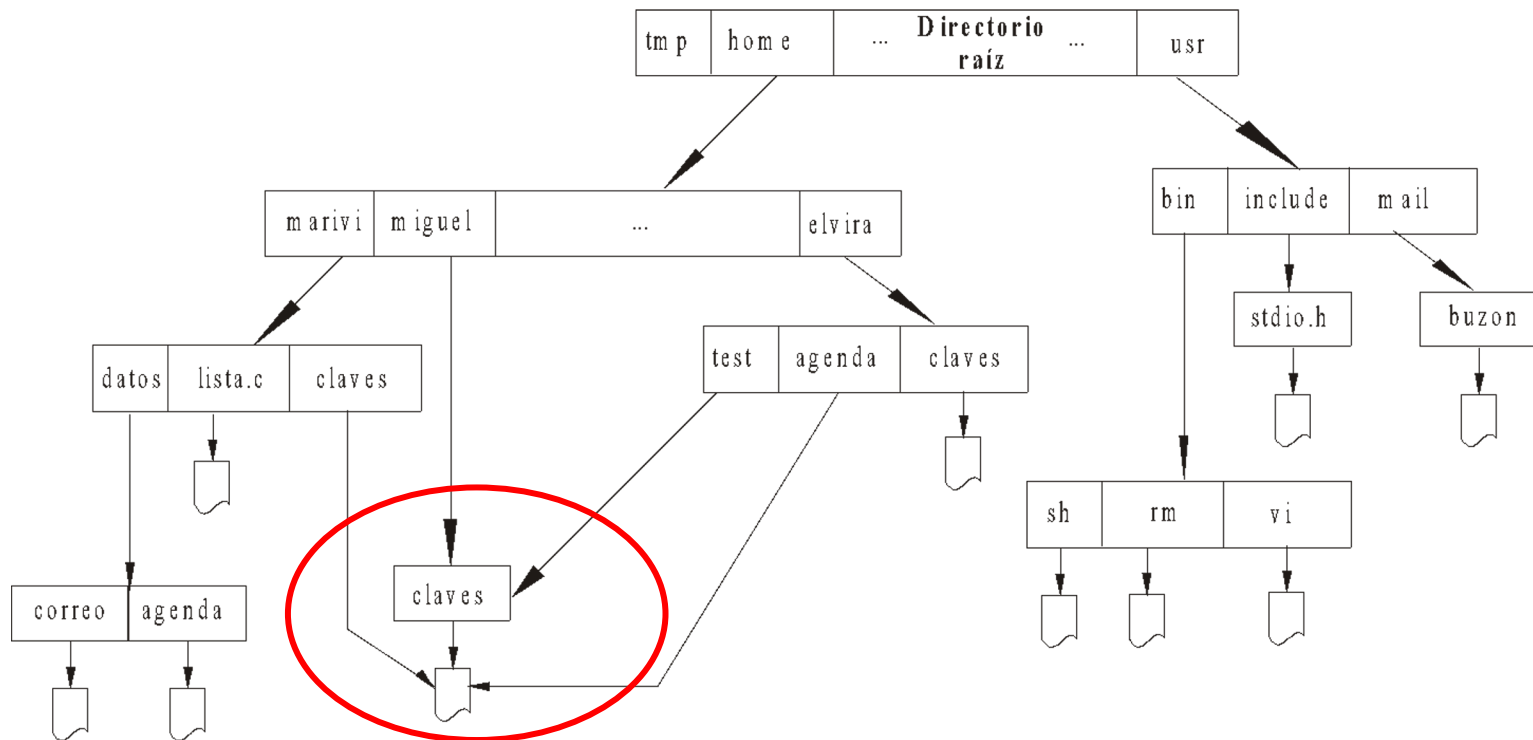


- Interpretar `/users/miguel/claves`
 - Traer a memoria entradas archivo con nodo-i 2
 - Se busca dentro `users` y se obtiene el nodo-i 342
 - Traer a memoria entradas archivo con nodo-i 342
 - Se busca dentro `miguel` y se obtiene el nodo-i 256
 - Traer a memoria entradas archivo con nodo-i 256
 - Se busca dentro `claves` y se obtiene el nodo-i 758
 - Se lee el nodo-i 758 y ya se tienen los datos del archivo
- ¿Cuándo parar?
 - Se ha encontrado el nodo-i del archivo
 - No se ha encontrado y no hay más subdirectorios
 - Estamos en un directorio y no contiene la siguiente componente del nombre (por ejemplo, `miguel`).

Grafo (no árbol) acíclico de directorios



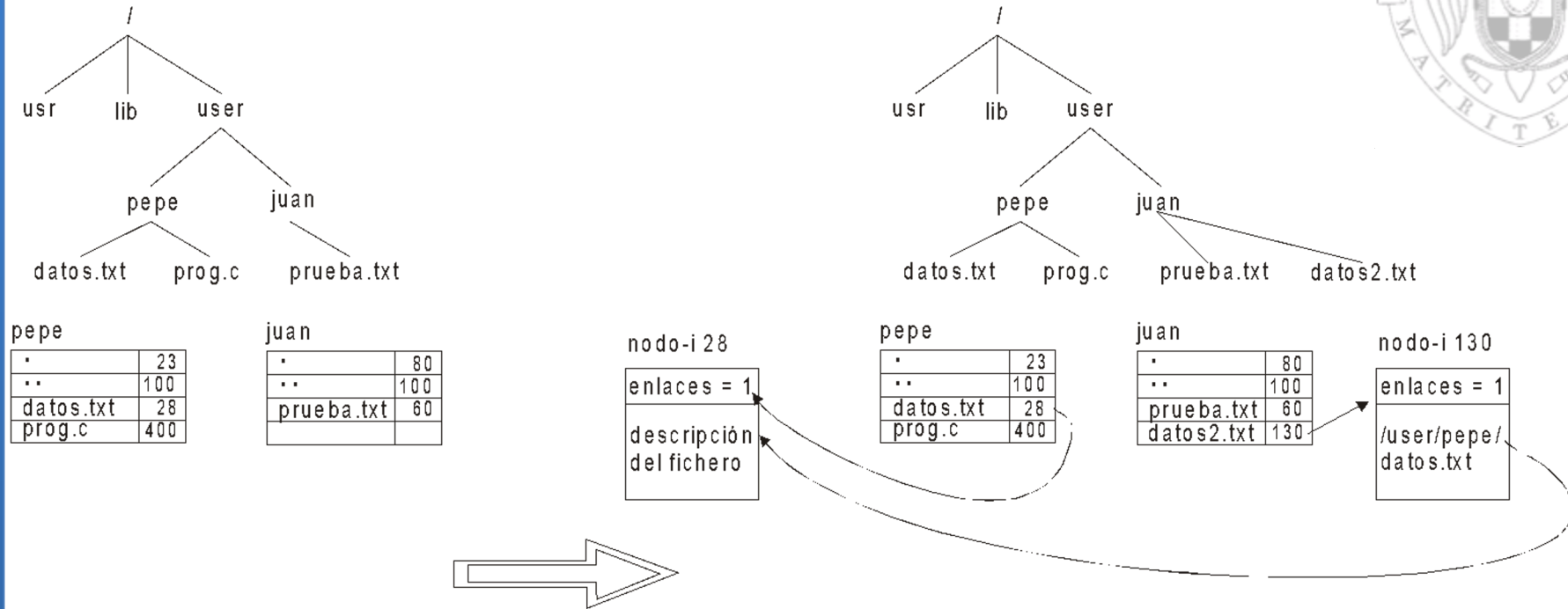
- Tienen archivos y subdirectorios compartidos
 - Fácil en sistemas UNIX..... id pensándolo para FAT



- En sistemas basados en nodos-i hay 2 mecanismos
 - Enlaces simbólicos y enlaces físicos



Enlace simbólico

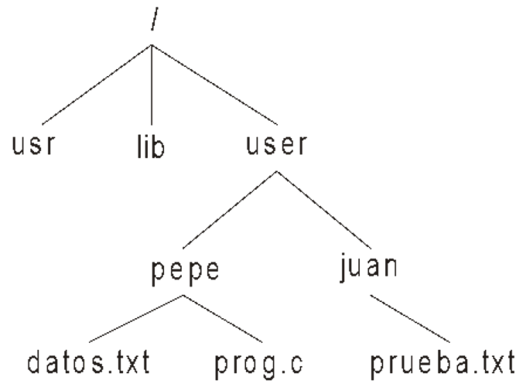


`In -s /user/pepe/datos.txt /user/juan/datos2.txt`

- `datos2.txt` es un nuevo fichero, con su propio `nodo-i`
 - Su contenido, es una cadena de caracteres con la ruta `/user/pepe/datos.txt`
 - Es un **nuevo tipo de fichero** (no es fichero regular)
- ¿Sería posible este tipo de fichero en FAT?

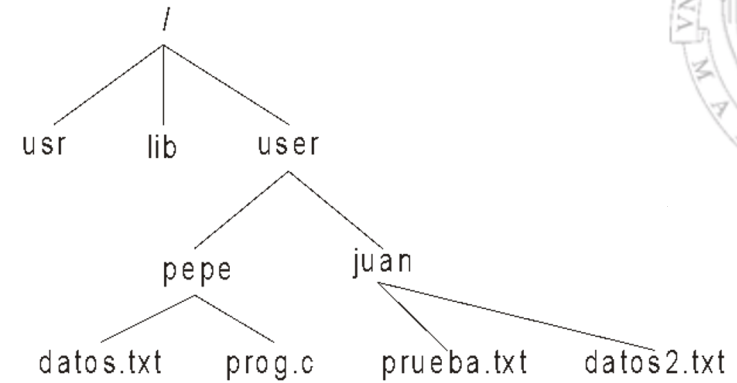


Enlace duro (o rígado o físico)



pepe	
.	23
..	100
datos.txt	28
prog.c	400

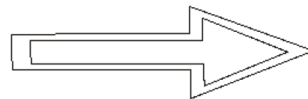
juan	
.	80
..	100
prueba.txt	60



pepe	
.	23
..	100
datos.txt	28
prog.c	400

juan	
.	80
..	100
prueba.txt	60
datos2.txt	28

nodo-i 28
enlaces = 2
descripción del fichero



In /user/pepe/datos.txt /user/juan/datos2.txt

- *datos2.txt* es **otro nombre** del mismo fichero. NO hay un nuevo nodo-i
 - Se incrementa el contador de enlaces que tiene cada nodo-i. ¿Qué pasa al borrar?
- ¿Sería posible esto en FAT?

Ejemplo



■ Enlace físico

```
$ echo "Hola" > archivo
```

```
$ ln archivo archivo_fis
```

```
$ stat archivo # ver "Links" (es 2)
```

```
$ stat archivo_fis #ver "Links" (es 2)
```

- Comparar campos "Inode" (son iguales)

■ Enlace simbólico

```
$ ln -s archivo archivo_sim
```

```
$ stat archivo # ver "Links" (no cambia)
```

```
$ stat archivo_sim # ver "Links" (es 1)
```

- Comparar campos "Inode" (son diferentes)

Sistemas de Ficheros



- El sistema de ficheros permite organizar la información dentro de los dispositivos de almacenamiento secundario en un formato inteligible para el SO.
- Previamente a la instalación del SF, es necesario dividir físicamente, o lógicamente, los discos en particiones o volúmenes.
- Una partición es una porción de un disco a la que se la dota de una identidad propia y que puede ser manipulada por el SO como una entidad lógica independiente.
- Una vez creadas las particiones, el SO debe crear las estructuras de los SF dentro de esas particiones. Para ello se proporcionan mandatos como `format` o `mkfs` al usuario.

Jerarquía de directorios

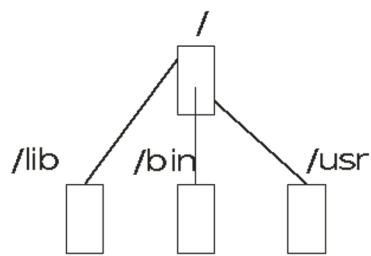


- ¿Árbol único de directorios?
 - Por dispositivo lógico en Windows (c:\users\miguel\claves, j:\pepe\tmp, ...)
 - Para todo el sistema en UNIX (/users/miguel/claves, /pepe/tmp, ...).
- Hacen falta servicios para construir la jerarquía: `mount` y `umount`.
 - `mount /dev/hda /users`
 - `umount /users`
- En Linux, el fichero `/tmp/dev/hola.txt` y `/tmp/cur/ola.txt` pueden estar en dispositivos físicos diferentes
 - ¿Ideas para el proceso de traducción de la ruta?

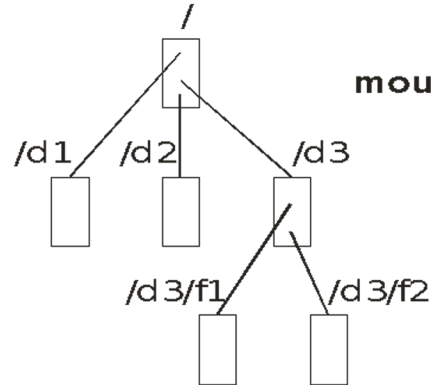
Montado de Sistemas de archivos o particiones



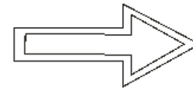
Volumen raiz
(/dev/hd0)



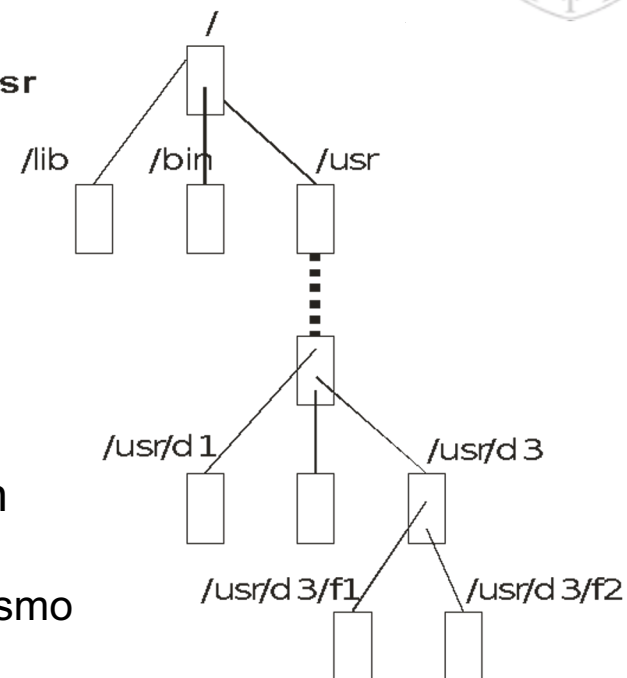
Volumen sin montar
(/dev/hd1)



mount /dev/hd1 /usr



Volumen montado



- Cada uno de estos árboles está, físicamente, en dispositivos diferentes
 - Sólo tras el *montado* quedan integrados en el mismo árbol
 - ¿Cómo? ¿Se modifican los discos para indicar esta situación?
- En el próximo tema veremos qué información del sistema de ficheros se almacena en disco y cuál en memoria principal

Ejemplo

- Creamos el archivo vacío (1.2e5 bloques)
dd if=/dev/zero of=/tmp/disk.img count=120000
- Creamos el sistema de ficheros sobre el archivo
mkfs -t ext3 -b 1024 /tmp/disk.img
- Montamos el sistema de ficheros
mkdir disk
mount -t ext3 -o loop /tmp/disk.img disk
mount
- Copiamos un archivo existente
cp archivo.txt disk/
ls disk
- Desmontamos el sistema de ficheros
umount disk
ls disk
rm /tmp/disk.img

