TEMA 2. Sistemas de Ficheros

Tema 2. Sistemas de Ficheros

2.1 Ficheros

- Concepto de fichero.
- Atributos y tipos. Permisos.

2.2 Directorios

- Jerarquía. Ruta absoluta y relativa
- Estructura de directorio
- Enlaces

2.3 Interfaz del Sistema

- CLI
- Llamadas al sistema

2.3 Sistemas de Ficheros

- Virtual File System
- Objetos: entradas de directorio, inodos, ficheros y superbloques
- Fragmentación y localidad
- Gestión de bloques de disco

TEMA 2.1 Ficheros

Ficheros (I)

Un fichero es una **abstracción** que permite almacenar y **manipular información** como una secuencia de bytes agrupada bajo un mismo **nombre**.

Se identifican con descriptores (`int`)

Los *ficheros* se crean y usan en un espacio de nombres común y jerárquico (*path*).

El SO define una serie de operaciones típicas como `read`, `write`, `open`, `close`...

***Everything is a file*:** La *mayoría* de los objetos del sistema se manejan como secuencias de bytes mediante un descriptor de ficheros (fd) y una interfaz común:

- Fichero regular
- Directorio
- Dispositivo E/S en modo bloque
- Dispositivo E/S en modo caracter
- Socket (conexión de red)
- Tubería con nombre
- Enlace simbólico

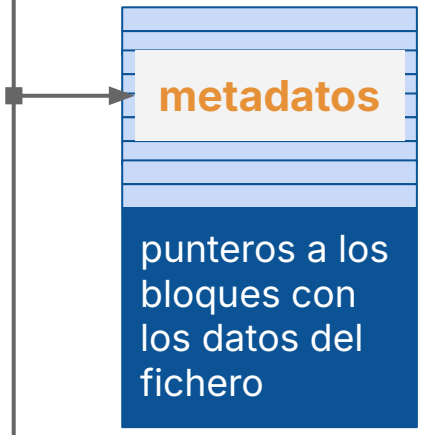
Ficheros (II)

El SO asocia a cada fichero metadatos, que se guardan en una estructura especial llamada **i-nodo *(index node)*:**

- **Dispositivo**: que almacena el fichero (e i-nodo).

Major (*driver*/clase)	**Minor** (*dispositivo*)

- Número de **i-nodo** que identifica unívocamente al fichero en el sistema de ficheros
- **Tipo** de fichero y **modo** (permisos)
- Identificadores del **usuario** y **grupo** propietario
- Número de **enlaces *duros*** al fichero
- **Tamaño** en bytes para ficheros regulares y enlaces
- Tamaño de **bloque E/S**
- Número de **Bloques** de 512 bytes usado por el fichero
- Marcas de tiempo para:
  - **creación**
  - **acceso** al fichero ej. read(2)
  - **modificación** del fichero ej. write(2)
  - **cambio** del i-nodo ej. chown(2)



Más información `inode(7)`

Ficheros (III)

```
$ echo 'a' > fichero.txt
```

```
Tamaño  = 2bytes  = ‘a’ + ‘\n’
Bloques = 8 * 512b = 4096b = bloque del sistema de ficheros 4Kb (ver stat -f)
```

```
$ stat fichero.txt
  File: fichero.txt
  Size: 2               Blocks: 8          IO Block: 4096   regular file
Device: 254,2 Inode: 11443365     Links: 1
Access: (0644/-rw-r--r--)  Uid: ( 1000/   ruben)   Gid: ( 1000/   ruben)
Access: 2025-07-23 20:04:49.397640090 +0200
Modify: 2025-07-23 20:28:29.113796897 +0200
Change: 2025-07-23 20:28:29.113796897 +0200
 Birth: 2025-07-23 20:04:49.397640090 +0200
```

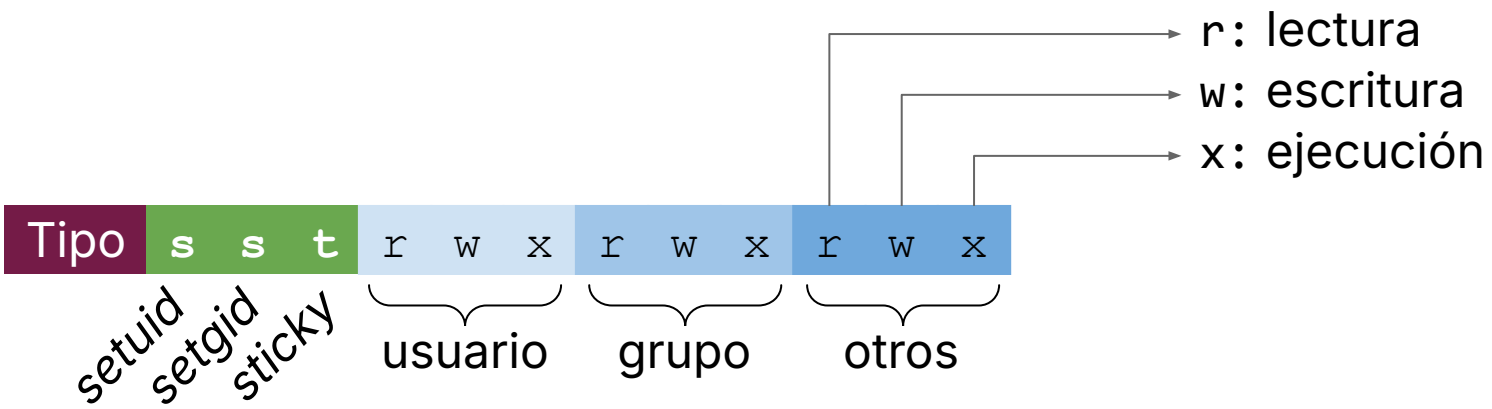
```
$ ls -l /dev/dm-2
brw-rw---- 1 root disk 254, 2 Jul 22 09:20 /dev/dm-2
```

b = dispositivo en modo **b**loque

```
$ ls -l fichero.txt
-rw-r--r-- 1 ruben ruben 2 Jul 23 20:28 fichero.txt
$ ls -i fichero.txt
11443365 fichero.txt
```

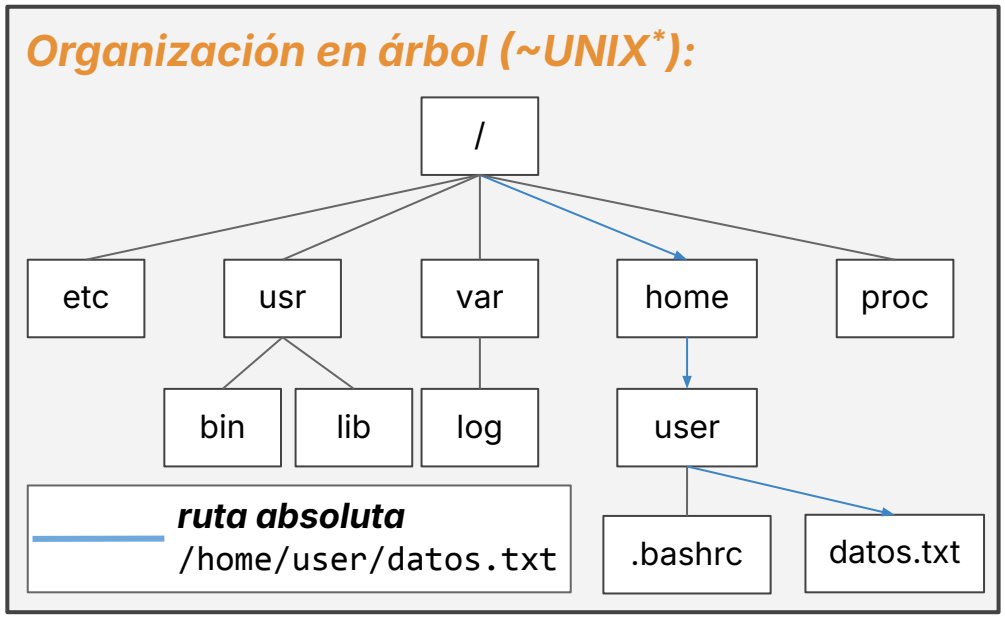
Ficheros (IV)

- Atributo **modo** del i-nodo
- Permisos especiales en **ficheros ejecutables (+x)** (Tema 3 - procesos):
  - **setuid**: El UID del proceso se establece al UID del propietario del fichero
  - **setgid**: El GID del proceso se establece al GID del propietario del fichero
- Permisos especiales en **directorios**:
  - **ejecución (x)**: permite cambiar al directorio (cd)
  - **setgid**: El grupo GID del fichero se crea con el GID del directorio (en lugar del GID del proceso)
  - **sticky**: Los ficheros sólo pueden ser borrados o renombrados por el propietario del fichero, por el propietario del directorio o por un proceso privilegiado (ej. `/tmp`)

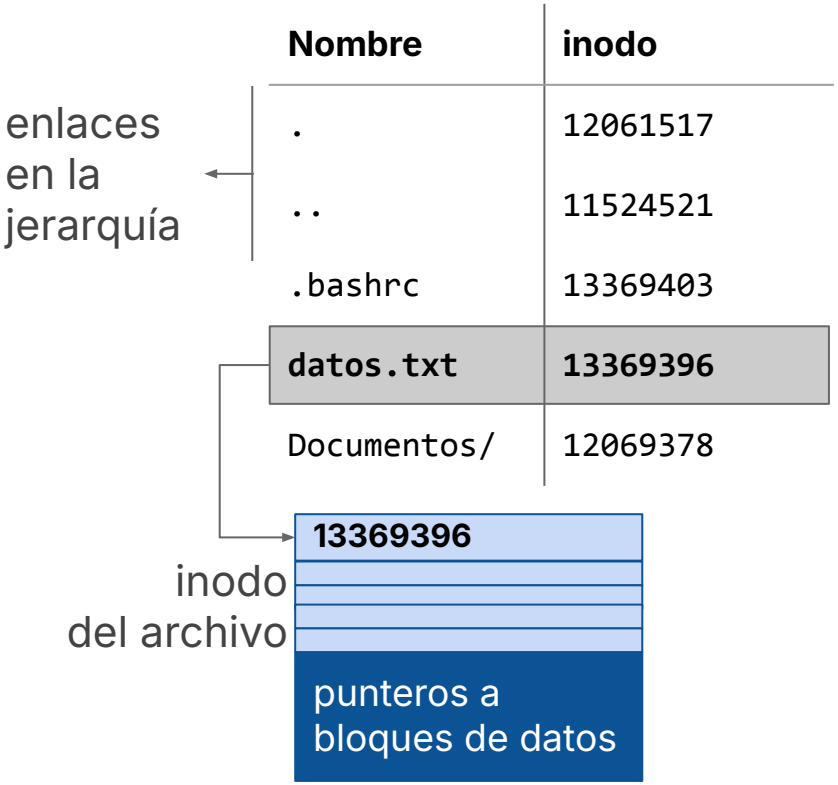


TEMA 2.2 Directorios

Directorios (I)



- Un **directorio** es un **fichero** especial que permite agrupar otros ficheros.
- El directorio contiene un **mapa de nombres de ficheros** y su **inodo** asociado:



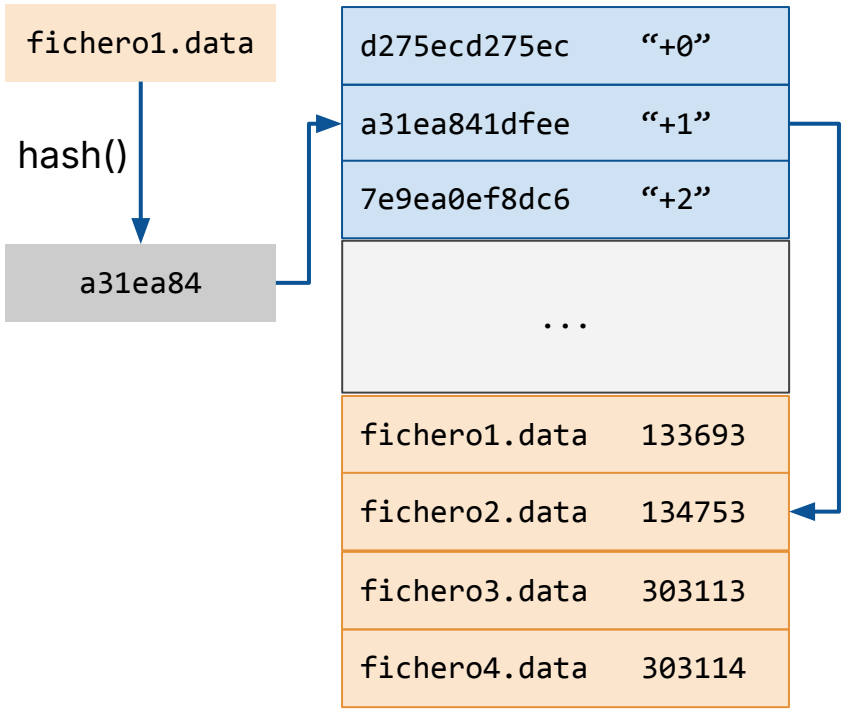
- El mapa nombre-inodo, según el tamaño, se puede almacenar en :
  - el propio inodo directorio
  - un bloque de datos
  - varios bloques de datos

* El nombre, propósito y ubicación de cada directorio está definido por *Filesystem Hierarchy Standard*

Directorios (II)

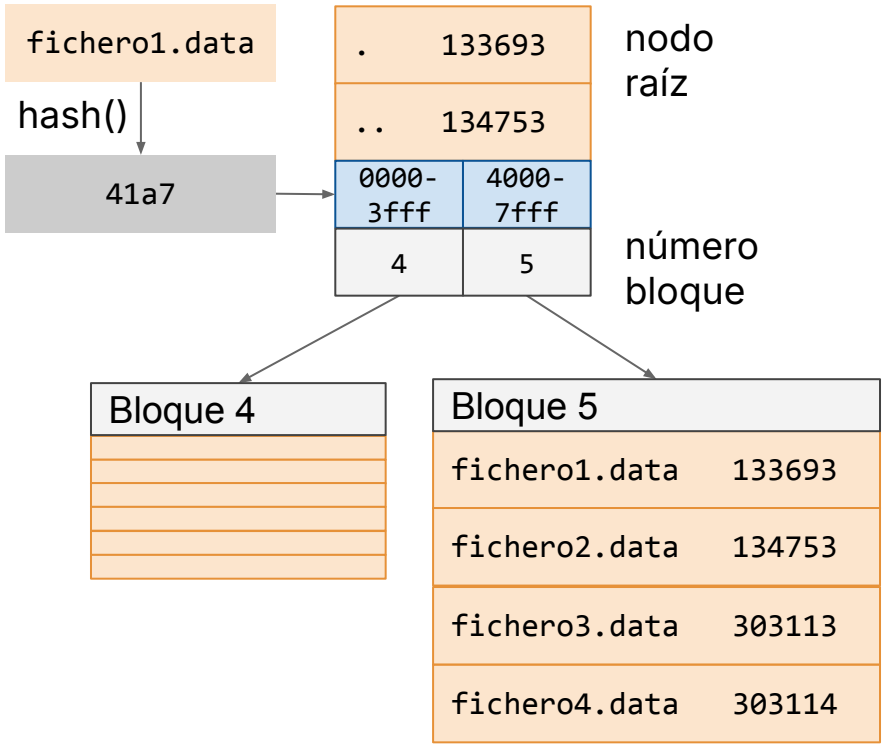
- El sistema de ficheros incluyen mecanismos de aceleración de búsqueda del inodo de los ficheros de un directorio.
- Las entradas pueden incluir alguna información adicional (ej. tipo de fichero)

Hash Tables (xfs)



Si la tabla hash es muy grande se distribuye en varios bloques usando una estructura en árbol B+tree

HTree (ext3, ext4)



Similar al árbol B+tree de 1 o 2 niveles, los nodos hoja tienen un array lineal con: "nombre de archivo - inodo"

Directorios (III)

```
debugfs:  htree etc
Root node dump:
      Reserved zero: 0
      Hash Version: 1
      Info length: 8
      Indirect levels: 0
      Flags: 0
Number of entries (count): 4
Number of entries (limit): 123
Checksum: 0xf19f40af
Entry #0: Hash 0x00000000, block 1
Entry #1: Hash 0x4b855aee, block 3
Entry #2: Hash 0x8a3857da, block 2
Entry #3: Hash 0xc0ba6778, block 4
```

shadow

1 Calcular el hash() del nombre del archivo

```
debugfs:  dx_hash shadow
Hash of shadow is 0xbfa46a3c (minor 0x3d714e0f)
```

2 Realizar búsqueda binaria en el árbol

0x8a.. < **0xbf**a.. < **0xc**0..

```
Entry #2: Hash 0x8a3857da, block 2
Reading directory block 2, phys 7055
49 0x8a3857da-e3130757 (20) host.conf
1073 0x94215c58-a42e8af0 (20) locale.conf
537 0xbba0b902-120a4dbc (16) man.conf
1486 0xbfa46a3c-3d714e0f (16) shadow
1135 0xacd5c1a8-e44116d2 (16) depmod.d
...
1528 0x9c50defa-3881a447 (336) fcron   leaf block checksum: 0xe4d61261
Entry #3: Hash 0xc0ba6778, block 4
```

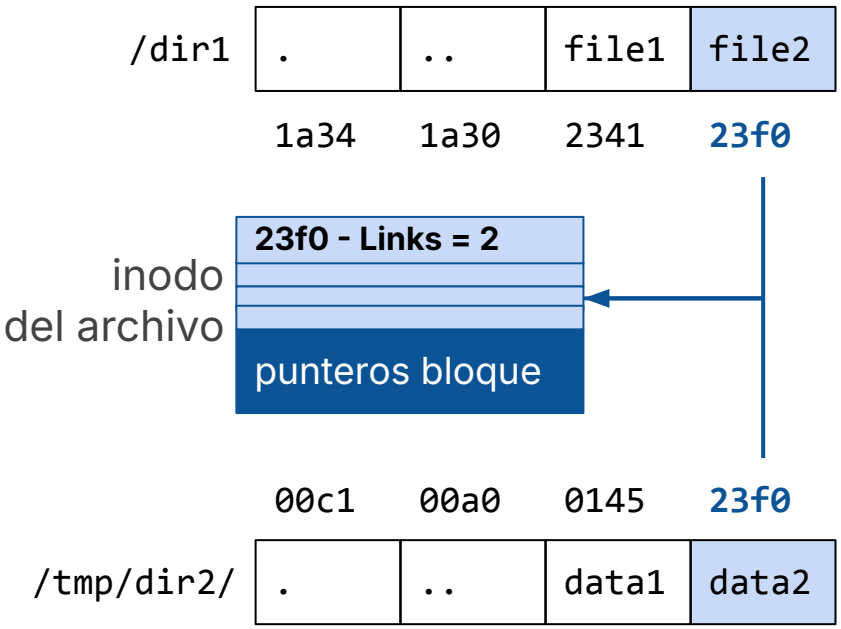
3 Buscar en el bloque el nombre

```
$ ls -il etc/shadow
1486 -r-------- 1 root root 554 Sep 25 18:45 etc/shadow
```

Enlaces (I)

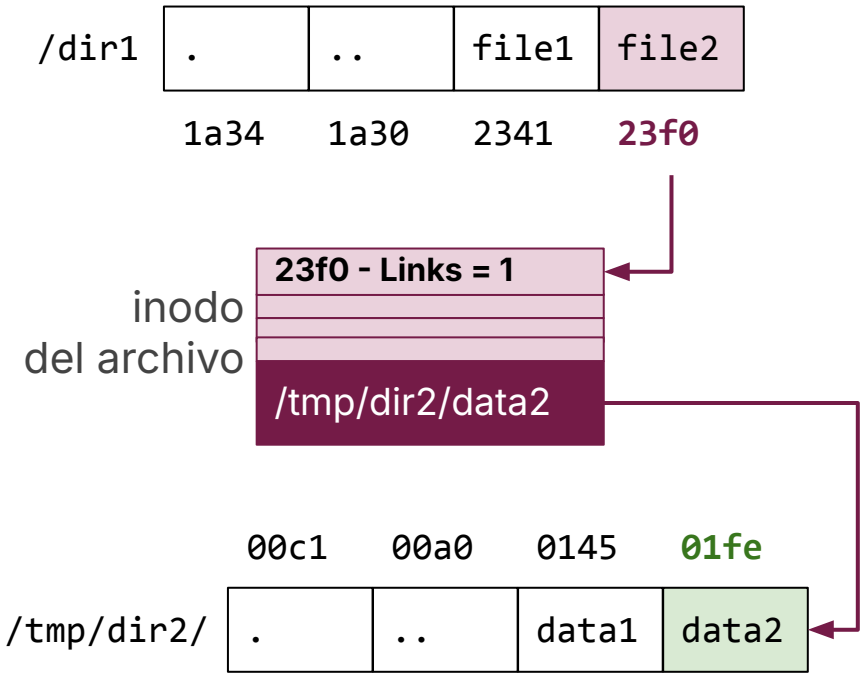
Enlaces Duros

- Cada entrada en un directorio que asocia un nombre con el mismo i-nodo.
- **Número de enlaces** (`st_nlink`): número de entradas en un directorio que apuntan a ese i-nodo
- Los enlaces duros solo pueden referir a i-nodos en en el **mismo SF**.



Enlaces Simbólicos

- Fichero especial que guarda la ruta a otro archivo o directorio (en el inodo o en un bloque de datos)
- Tamaño del fichero es la longitud de la ruta.
- Puede referir a **distintos SF**.



Enlaces (II)

```
$ mkdir dir1
$ touch file1
$ ln -s file1 symlink
$ ln file1 hardlink
```

El directorio tiene 2 enlaces. ¿Cuáles?

```
$ ls -l
total 4
drwxr-xr-x 2 ruben ruben 4096 Feb  3 10:38 dir1
-rw-r--r-- 2 ruben ruben    0 Feb  3 10:38 file1
-rw-r--r-- 2 ruben ruben    0 Feb  3 10:38 hardlink
lrwxrwxrwx 1 ruben ruben    5 Feb  3 10:39 symlink -> file1
```

symlink es un enlace simbólico (l)
que tiene el path (file1 = 5 bytes)

```
$ ls -i1
12489415 dir1
12489469 file1
12489469 hardlink
12489493 symlink
```

file1 y hardlink1:

- refieren al mismo inodo 12489469
- Hay 2 enlaces a este inodo

TEMA 2.3 Interfaz del Sistema

Interfaz del Sistema Operativo. CLI (I)

Línea de Comandos

Comandos para consultar y actualizar el estado (inodo) de un fichero:

- `stat(1)` muestra información sobre el fichero o sistema de ficheros
- `touch(1)` actualiza los tiempos de acceso y modificación
- `chown(1)` cambia el propietario (usuario y/o grupo)
- `chmod(1)` cambia el modo (permisos)
- `ln(1)` creación de enlaces simbólicos y duros
- `readlink(1)` leer el path de un enlace simbólico

Otros comandos estudiados en el **Tema 1**:

- `ls(1)` listar los contenidos de un directorio y atributos de los ficheros
- `rm(1)`, `cp(1)`, `mv(1)` borrado, copia y renombrado de ficheros
- `mkdir(1)`, `rmdir(1)` creación y borrado de directorios.

Interfaz del Sistema Operativo. API (I)

stat(2)

<sys/stat.h>

Información sobre el fichero (lstat versión para enlaces simbólicos)

- `int stat(const char *pathname, struct stat *buf);`
- `int fstat(int fd, struct stat *buf);`

```
struct stat {
    dev_t     st_dev;     /* Dispositivo que lo contiene*/
    ino_t     st_ino;     /* Inodo */
    mode_t    st_mode;    /* Tipo de fichero y permisos */
    nlink_t   st_link;    /* Número de enlaces rígidos */
    uid_t     st_uid;     /* UID del propietario */
    gid_t     st_gid;     /* GID del propietario */
    dev_t     st_rdev;    /* Disp. si fichero especial */
    off_t     st_size;    /* Tamaño en bytes */
    blksize_t st_blksize;/* Tamaño bloque E/S ficheros */
    blkcnt_t  st_blocks;  /* Bloques físicos asignados */
    time_t    st_atime;   /* Último acceso */
    time_t    st_mtime;   /* Última modificación */
    time_t    st_ctime;   /* Último cambio de estado */
}
```

Interfaz del Sistema Operativo. API (II)

<unistd.h>

- Crear un enlace rígido (*hard link*):

```
int link(const char *old, const char *new);
```

  - No puede hacerse a otro sistema de ficheros ni con directorios

- Crear un enlace simbólico (*soft link* o *symlink*):

```
int symlink(const char *old, const char *new);
```

  - Puede hacerse a otro sistemas de ficheros y con directorios
  - El fichero original puede no existir

- Leer el contenido de la ruta de un enlace simbólico:

```
int readlink(const char *path, char *buf, size_t bufsize);
```

  - El tamaño del enlace puede determinarse con `lstat(2)`
  - La cadena devuelta en buf no contiene el carácter de fin de cadena

- Eliminar un nombre de fichero y posiblemente el fichero al que se refiere:

```
int unlink(const char *name);
```

  - Borra la entrada del directorio y decrementa el número de enlaces en el inodo
  - Si número de enlaces es 0 se elimina y el espacio se libera
  - El fichero no se eliminará mientras que exista un proceso que lo mantenga abierto.

Interfaz del Sistema Operativo. API (III)

Abrir y opcionalmente crear un fichero:

```
int open(const char *path, int flags);
int open(const char *path, int flags, mode_t mode);
```

- `flags` debe indicar el modo de acceso y puede incluir otras opciones:
  - `O_RDONLY`: Acceso de sólo lectura
  - `O_WRONLY`: Acceso de sólo escritura
  - `O_RDWR`: Acceso de lectura y escritura
  - `O_CREAT`: Si el fichero no existe, se crea con los permisos en `mode` (si se omite, se usa un valor arbitrario de la pila)
  - `O_EXCL`: Con `O_CREAT` para provocar un error si el fichero existe
  - `O_TRUNC`: Se trunca el tamaño del fichero a 0
  - `O_APPEND`: Antes de realizar cualquier escritura se posiciona el puntero de fichero a la última posición.
- `mode` indica los permisos a aplicar en caso de que se cree un nuevo fichero (con la opción `O_CREAT`). En octal o como OR bit a bit de *flags*.
- Devuelve un descriptor de fichero con el puntero de acceso posicionado al principio del fichero, o -1 si ocurre un error (y establece `errno`)

Interfaz del Sistema Operativo. API (IV)

- Leer, escribir, posicionar y cerrar ficheros:

```
ssize_t write(int fd, void *buffer, size_t count);
ssize_t read(int fd, void *buffer, size_t count);
off_t lseek(int fd, off_t offset, int whence);
int close(int fd);
```

  - **whence** puede ser `SEEK_SET`, `SEEK_CUR` o `SEEK_END`
  - **No** deben **mezclarse** estas llamadas al sistema **con funciones de librería** (ej. `fopen`, `fread`, `fwrite`… de `stdio.h` o clases `fstream` en C++)
- La escritura de ficheros se realiza a través de la *caché* de páginas, acelerando el acceso (puede evitarse con la opción `O_DIRECT`).
- Sincronizar un fichero caché de páginas y bloques de disco:

```
int fsync(int fd);
```

  - La llamada se bloquea hasta que el dispositivo informa de que la transferencia se ha completado

Interfaz del Sistema Operativo. API (V)

```
<sys/types.h>
<dirent.h>
```

- Abrir un directorio:

```
DIR *opendir(const char *name);
```

  - Devuelve un puntero al flujo de directorio, posicionado en la primera entrada del directorio
  - El tipo de datos `DIR` se usa de forma similar al tipo `FILE` especificado por la librería de E/S estándar

- Leer entradas de un directorio:

```
struct dirent *readdir(DIR *dir);
```

  - La función retorna una estructura `dirent` que apunta a la siguiente entrada en el directorio, y `NULL` cuando llega al final u ocurre un error
  - El único campo contemplado por el estándar POSIX es `d_name`, de longitud variable (menor que `NAME_MAX`)

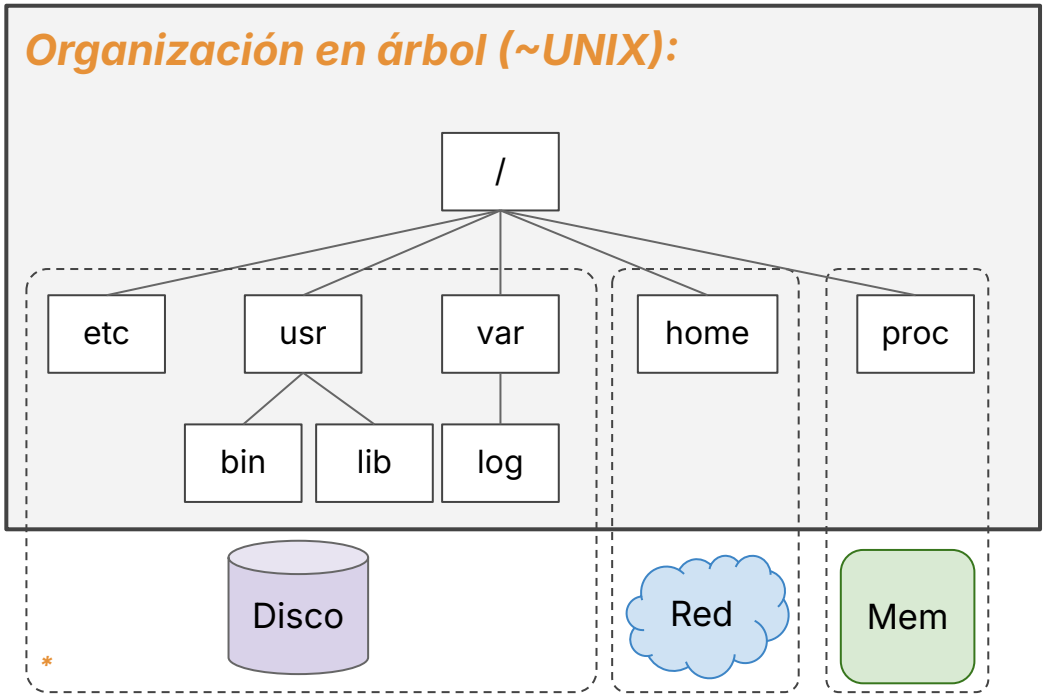
- Cerrar un directorio:

```
int closedir(DIR *dir);
```

TEMA 2.4 Sistemas de Ficheros

Sistemas de Ficheros

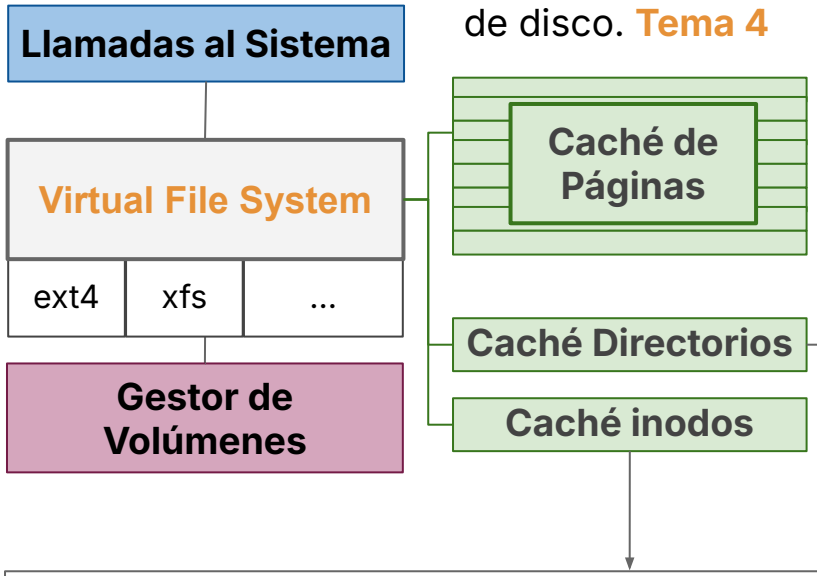
- Un sistema de ficheros permite organizar datos en ficheros y carpetas.
- El sistema de ficheros ofrece un interfaz (***POSIX***) sobre estos ficheros
  - Abrir, cerrar, leer, escribir..
- El kernel soporta distintos sistemas de ficheros accesibles desde un interfaz común independiente del tipo.



* Un sistema de ficheros accesible en el árbol está *montado*

Sistemas de Ficheros. VFS (I)

- Interfaz común a diferentes tipos de sistemas de ficheros. Permite a las aplicaciones usar el mismo interfaz del sistema (p.ej. `open()` o `read()`).
- VFS construye abstracciones de objetos (ficheros, entradas de directorio, inodos y superbloques) que usa el kernel para manipular ficheros.



Entradas de Directorios

- Representan elementos de una ruta en el SF (p.ej. /home/user/data: /, home, user, data)
- Cada entrada `dentry` incluye metadatos y el inodo asociado
- Las `dentry` se guardan en memoria (caché de directorios, `dcache`) para acelerar el acceso.
- Cache negativa (guarda rutas inexistentes)

inodos

- Información genérica (modo, tamaño, tipo...) (`inode`) uniforme para cada SF.
- El inodo se construye usando la información del *inodo* (ext4) del SF en particular
- Los inodos se guardan en memoria (caché de inodos) para acelerar el acceso.

Sistemas de Ficheros. VFS (II)

Superbloque

- Representan información global de un sistema de ficheros montado
- Se corresponde con un bloque de control del SF, información de control almacenada en sectores específicos del disco (varias copias), del sistema de ficheros
- Información:
  - Tipo de sistema de fichero
  - Tamaño de bloque
  - Tamaño máximo de fichero
  - Estado (*dirty flag)*
  - Listas de control de acceso,...

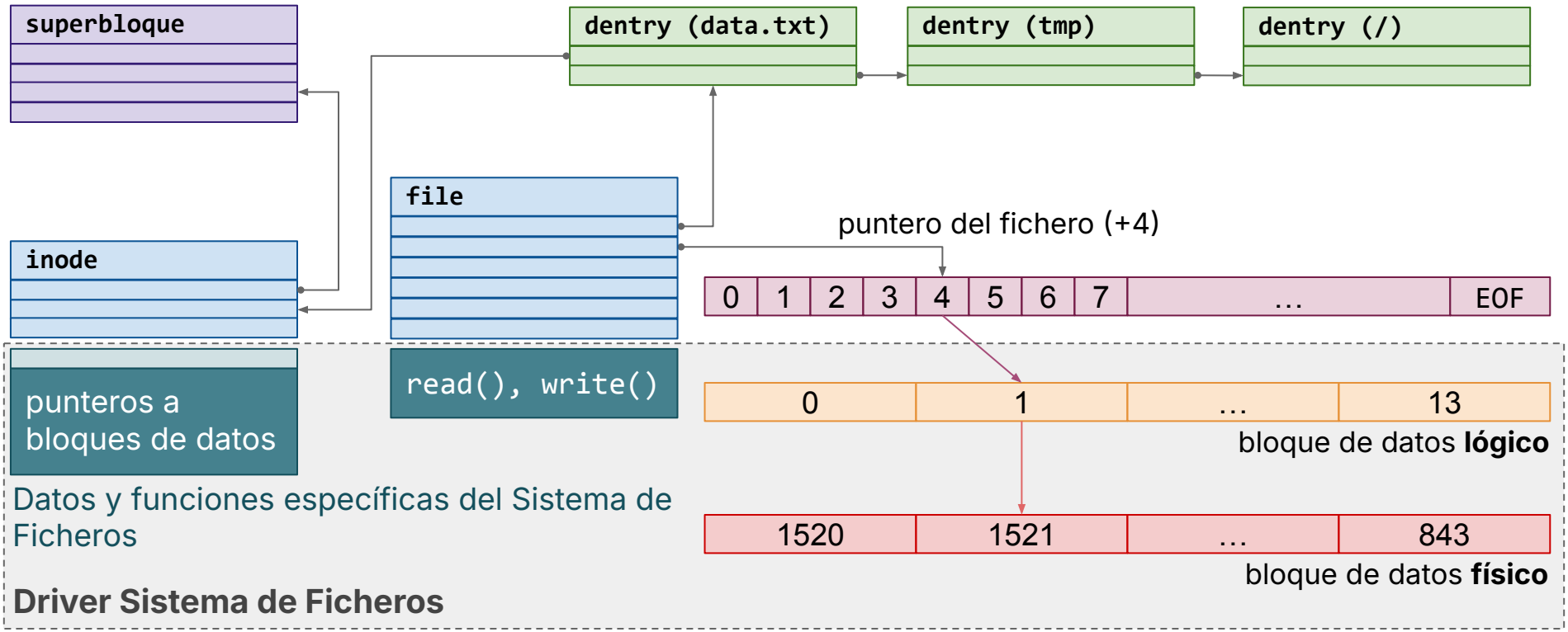
Interfaz del Sistema

- statfs(2) obtiene estadísticas del sistema de ficheros accediendo al superbloque y algunas de las operaciones asociadas.
- Comandos `df -T` y `stat -f` muestran algunas de estas características

Sistemas de Ficheros. VFS (III)

Ficheros

- Representación en memoria de un fichero **abierto** por un proceso
- Contiene **métodos**, **implementados** por el **SF,** para realizar operaciones sobre el fichero (read, write, seek, open...)
- Diferentes objetos fichero pueden referir al mismo archivo
- Puntero al fichero, representado como un desplazamiento desde el inicio

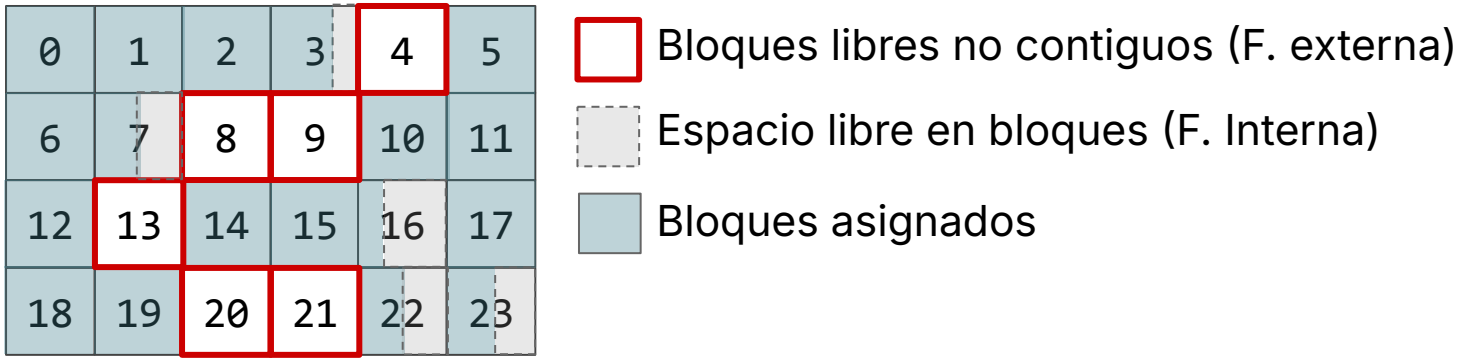


Sistemas de Ficheros. Gestión de Bloques (I)

Asignación de Bloques

- **Localidad Espacial**. Mantener los datos de un fichero juntos mejora el rendimiento:
  - Discos mecánicos. Menos movimiento de la cabeza lectora.
  - SSD. Menos operaciones (de mayor tamaño) potencialmente concentradas en un bloque. Optimización de la asignación del espacio del disco.
- **Fragmentación**:
  - **Externa**: El espacio libre está fragmentado en huecos relativamente pequeños no contiguos del disco. (rendimiento)
  - **Interna:** Espacio no utilizado dentro de un bloque de disco. (espacio desperdiciado)

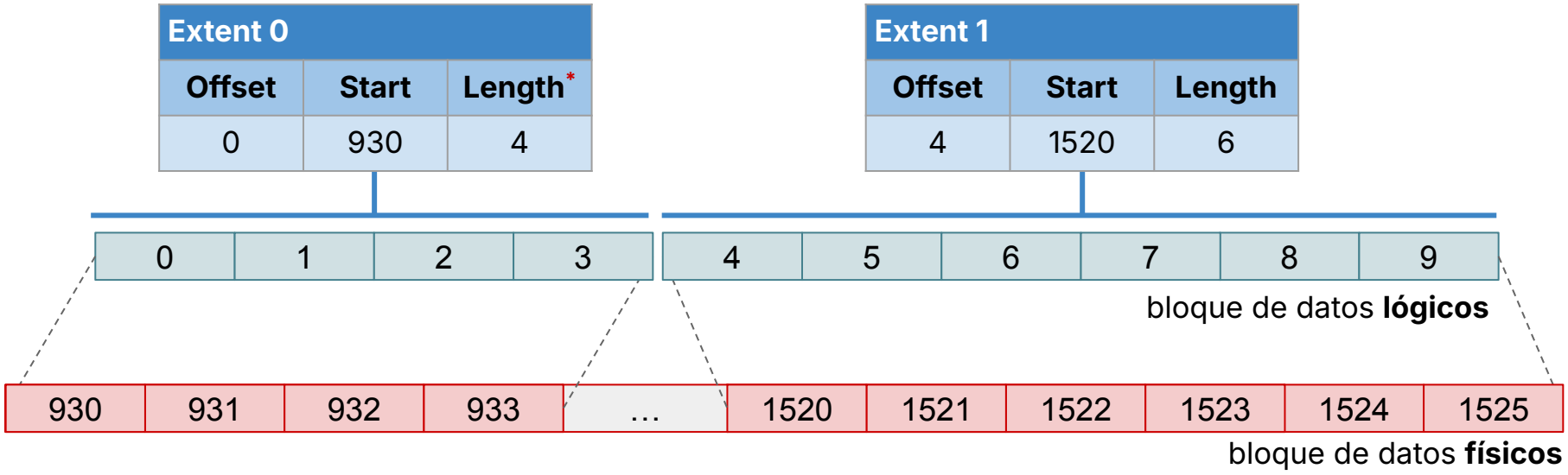
Disco. Bloques Físicos



Sistemas de Ficheros. Gestión de Bloques (II)

Estrategias para mitigar la fragmentación

- ***Extents.*** Grupo de bloques contiguos (bloque inicial + número) en lugar de usar bloques individuales en la gestión.
- **Asignación retardada.** Los bloques físicos se asignan cuando los datos tienen que escribirse en disco (de la caché de páginas).
- **Preasignación de bloques.**
  - Asignación de más bloques de los solicitados (ej. ext4 multi-block allocator).
  - Reserva de bloques contiguos para un fichero `fallocate(2)`.

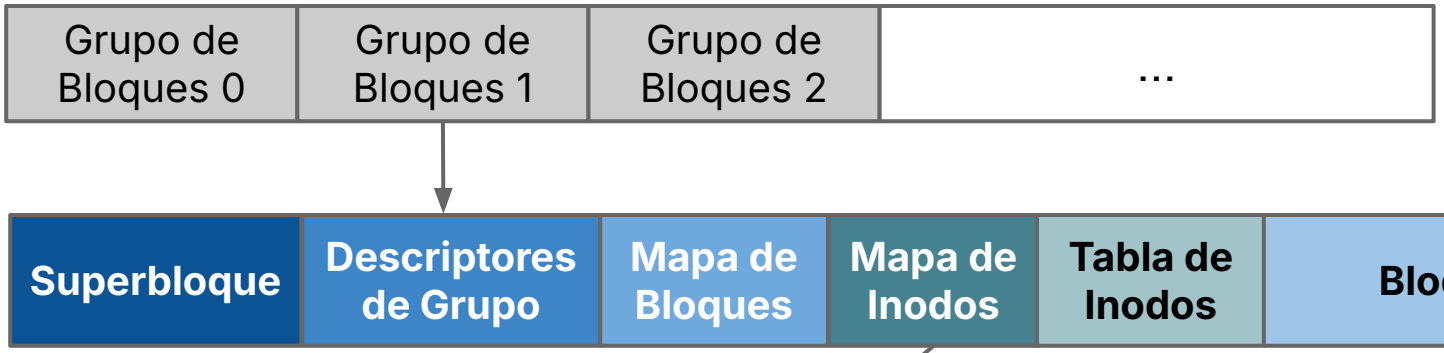


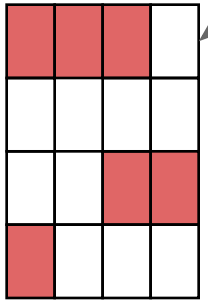
Length*, tamaño máximo fijo (ej. ext4 2^{15} * 4K bloques = 128MB, ver ext4_extents.h)

Sistemas de Ficheros. Gestión de Bloques (II)

Ejemplo. Estructura del Sistema de ficheros ext4

- **Agrupación de bloques.** Las operaciones se restringen a un subconjunto de los bloques del disco.
- **Localidad de bloques para inodos y directorios.** Mantener los bloques de datos junto con el inodo y directorio para aprovechar la localidad temporal.





Mapa de bits que indica si el inodo o bloque está libre .

Alternativas:

- Lista de bloques libres
- B+tree de bloques libres por número de bloque y tamaño (ej. XFS)

Sistemas de Ficheros. Gestión de Bloques (I)

Asignación Contigua

- Los bloques de datos del fichero se asignan de forma contigua.
- Fácil acceso secuencial y aleatorio.
- Fragmentación externa.
- Aplicación: Medios de solo lectura, o con tamaños de ficheros conocidos.
- Ejemplos: CD-ROM (ISO9660), cintas (LFTS)

Fichero	Bloques Físicos
archivo.1	0,1,2,3,4,5,6,7
archivo.2	10,11,12,13
archivo.3	14,15,16,17,18,19
archivo.4	27,28,29,30,31,32

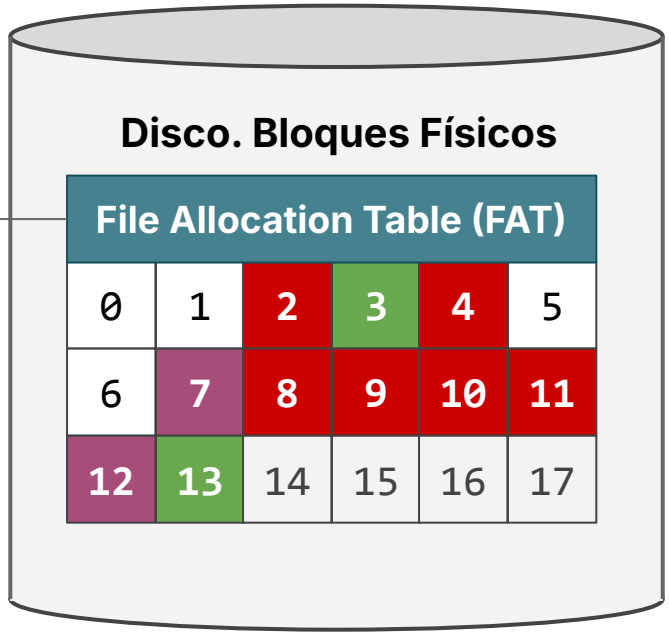
Sistemas de Ficheros. Gestión de Bloques (II)

Bloques Enlazados

- Los bloques que constituyen el fichero se almacenan como una lista enlazada.
- **Sin fragmentación externa**
- **Asignación dinámica simple** pero **sin localidad**
- La tabla FAT es un **único punto de fallo**
- Aplicación: USBs (No escala para discos grandes)
- Ejemplo: FAT16, FAT32.

Entradas de directorio	
Nombre	**Primer Bloque**
archivo.1	2
archivo.2	3

Bloque	Estado
0	FREE
1	FREE
2	4
3	13
4	9
5	FREE
6	FREE
7	BAD
8	EOF
9	10
10	11
11	8
12	BAD
13	EOF
...	



El acceso a un byte en el fichero consiste en determinar el bloque en el que se encuentra y recorrer la lista

Sistemas de Ficheros. Gestión de Bloques (VI)

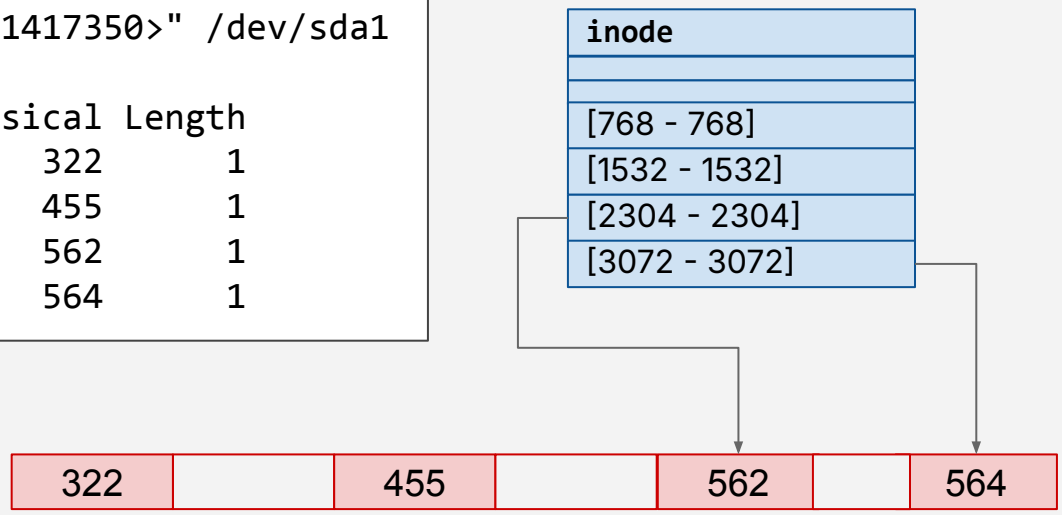
Árboles Balanceados de Bloques B+tree. ext4

- Árbol de hasta 5 niveles
- El inodo guarda hasta 4 extents sin necesidad de crear el árbol
- La estructura extent ocupa 12 bytes. Para bloques de 4K se pueden guardar hasta 4096/12 ~ 340 extents por bloque.
- Los extents ocupan hasta 128 MB como máximo

Ejemplo

- Archivo con 4 secciones de 3 Kbytes escritas cada 1Mbyte (extents = 1 bloque)

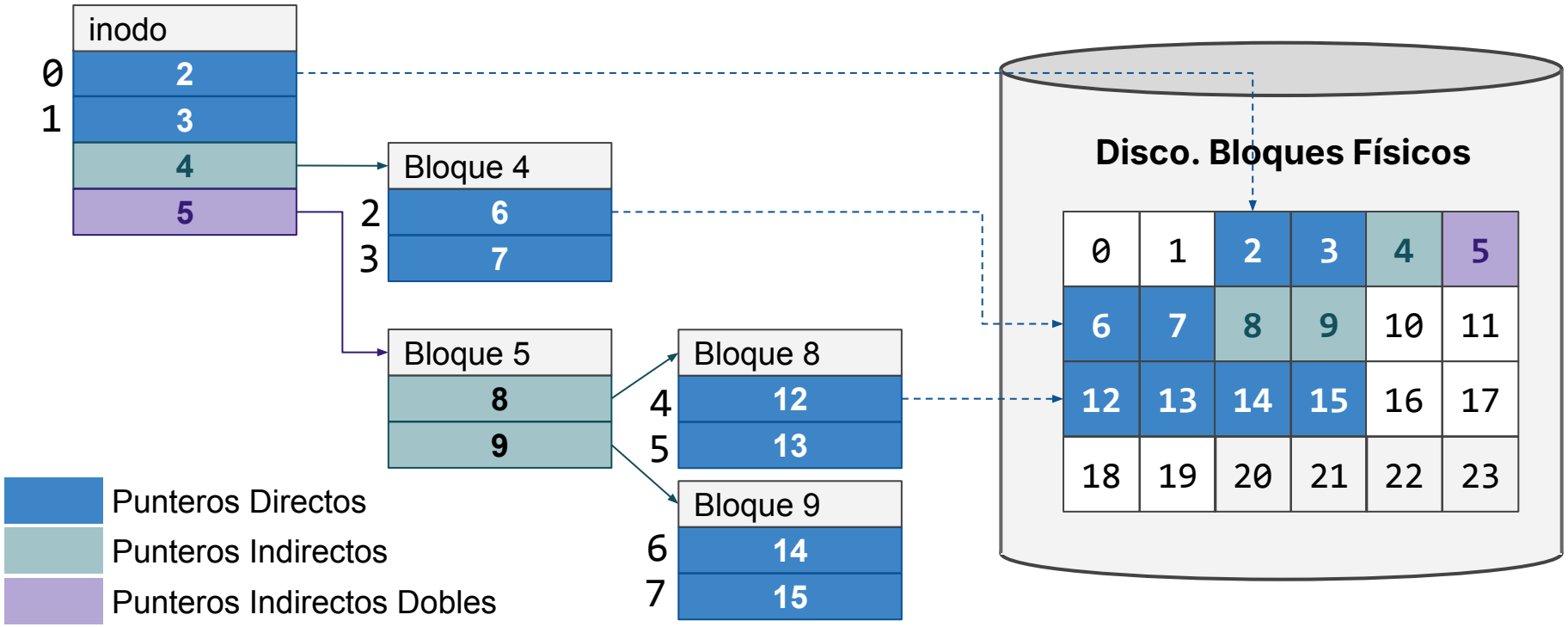
```
$ sudo debugfs -R "dump_extents <11417350>" /dev/sda1
debugfs 1.47.2 (1-Jan-2025)
Level Entries       Logical     Physical Length
 0/ 0   1/  4    768 -   768  322 -  322      1
 0/ 0   2/  4   1536 -  1536  455 -  455      1
 0/ 0   3/  4   2304 -  2304  562 -  562      1
 0/ 0   4/  4   3072 -  3072  564 -  564      1
```



Sistemas de Ficheros. Gestión de Bloques (III)

Bloques Indexados

- Los bloques que constituyen el fichero se indexan en uno o más niveles
- **Reducida fragmentación externa**
- Acceso muy **eficiente** para **ficheros pequeños**
- Operaciones de borrado **ineficientes** para **ficheros grandes**
- Aplicación: sistemas de ficheros de propósito general
- Ejemplos: ext2, UFS



Sistemas de Ficheros. Gestión de Bloques (IV)

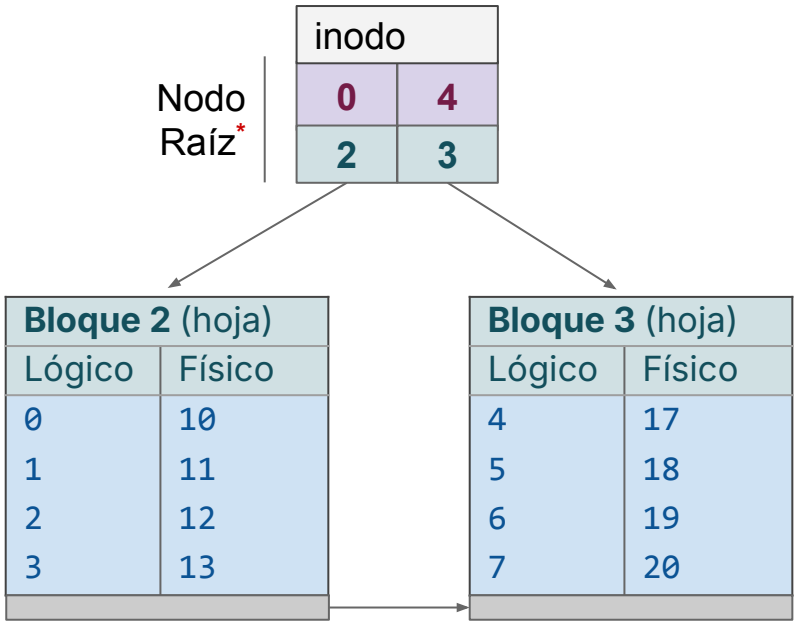
Bloques Indexados

- El acceso a un byte en el fichero consiste en determinar:
  - El bloque lógico en el que se encuentra
  - Determinar el nivel dónde está el bloque lógico (directos, indirectos, indirectos dobles)
  - Determinar el bloque de disco dentro del nivel para el bloque lógico
- El **tamaño máximo**
  - $Tt = Tb \cdot (D + I + I^2)$
  - `Tb` Tamaño de bloque
  - `D` Entradas directas
  - `I` índices por bloque (`Tb / Tíndice`)
- Ejemplo `ext2` (con **indexación triple**):
  - Tamaño de bloque 1 KiB, 4 bytes para índice bloque
  - $Tt = 2^{10} \cdot (12 + 2^8 + 2^{16} + 2^{24}) \simeq 2^{34} = 16$ GiB
  - *Ejercicio*: Calcular el número de bloques empleados en almacenar índices en este caso.

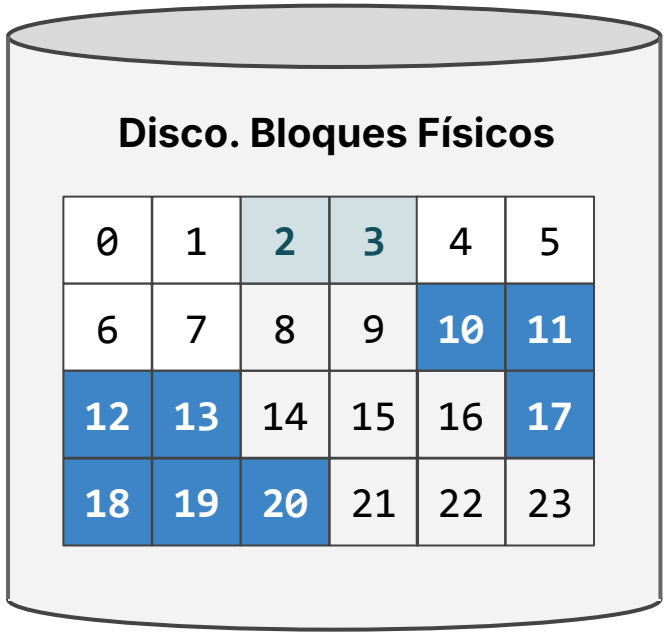
Sistemas de Ficheros. Gestión de Bloques (V)

Árboles Balanceados de Bloques B+tree

- Usan extents en lugar de bloques individuales
- Los extents se guardan en una estructura B+tree de profundidad reducida.
- **Gestión eficiente para ficheros grandes** (búsquedas rápidas, mayor tamaño de fichero)
- Aplicación: Usado en los SF actuales
- Ejemplos: ext4, xfs



Enlace entre los nodos hoja para acceso secuencial



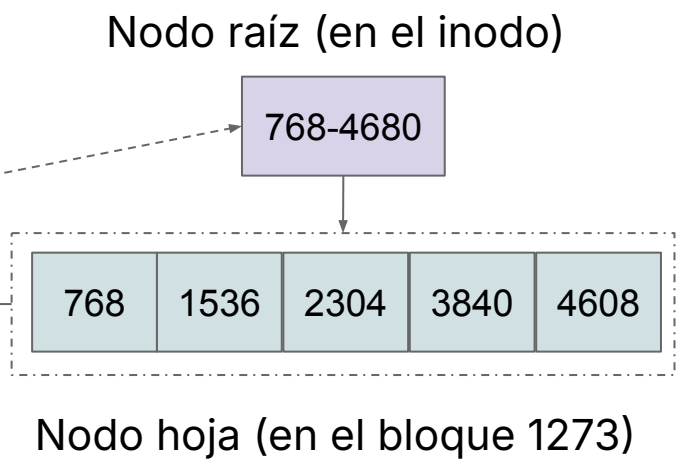
* Para ficheros pequeños el nodo raíz guarda punteros directos a bloque

Sistemas de Ficheros. Gestión de Bloques (VII)

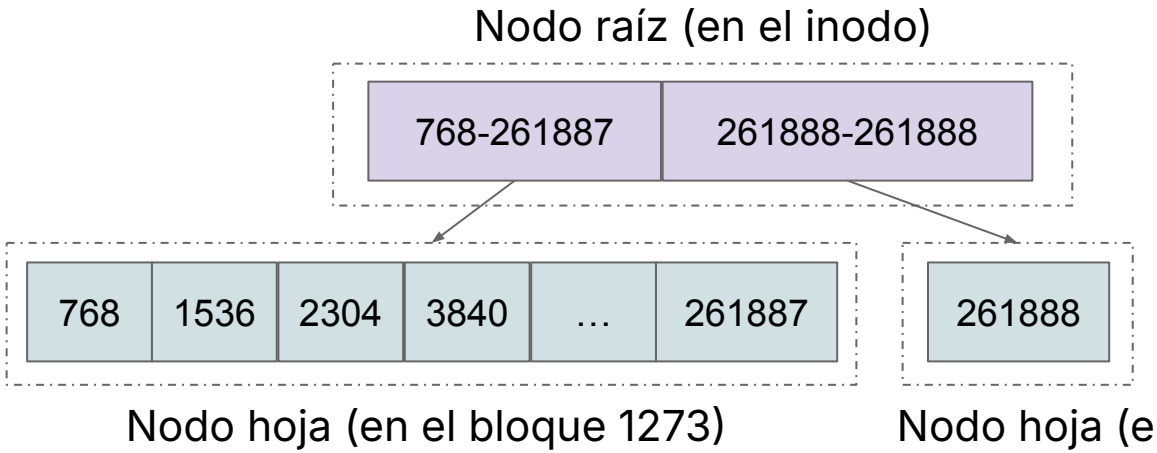
Ejemplo

- Añadimos dos secciones más de 3 Kbytes (2 extents adicionales)

```
$ sudo debugfs -R "dump_extents <11417350>" /dev/sda1
debugfs 1.47.2 (1-Jan-2025)
Level Entries       Logical     Physical Length
 0/ 1   1/  1   768 -  4608   1273          3841
 1/ 1   1/  6   768 -   768   322 -  322      1
 1/ 1   2/  6  1536 -  1536   455 -  455      1
 1/ 1   3/  6  2304 -  2304   562 -  562      1
 1/ 1   4/  6  3072 -  3072   564 -  564      1
 1/ 1   5/  6  3840 -  3840   666 -  666      1
 1/ 1   6/  6  4608 -  4608   677 -  677      1
```



- Después de añadir un total de 341 bloques al fichero



La implementación ext4 parte el nodo en el punto de inserción, ver [ext4/extents.c]

Sistemas de Ficheros. Gestión de Bloques (VIII)

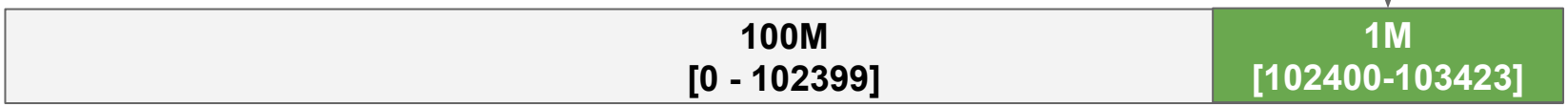
Ficheros Dispersos (sparse)

- Son ficheros parcialmente vacíos (regiones del fichero sin datos)
- El sistema de ficheros no escribe (ni lee) estos bloques del disco
- Implementación:
  - Bloques Indexados: Sólo se inicializan los índices necesarios para los bloques usados
  - Extents: Los extents que se corresponden con estas regiones no están presentes

Ejemplo.

Tamaño de bloque 1K. Bloques 102400-103423 (1M) sin inicializar.

[Extent 0] Offset: 102400 Length: 1024 Start: 21504




```
#Fichero de 101M con solo el último M escrito
$ dd if=/dev/zero of=sparse.img bs=1M seek=100 count=1
```

```
$ ls -slh | grep sparse
1.0M -rw-r--r-- 1 ruben ruben 101M Feb 3 10:38 sparse.img
```

Bloques asignados y tamaño del fichero son diferentes