Sistemas Operativos
2º Curso
Doble Grado en Ingeniería Informática y Matemáticas

Tema 4:

Gestión de archivos

José Antonio Gómez Hernández, 2020.

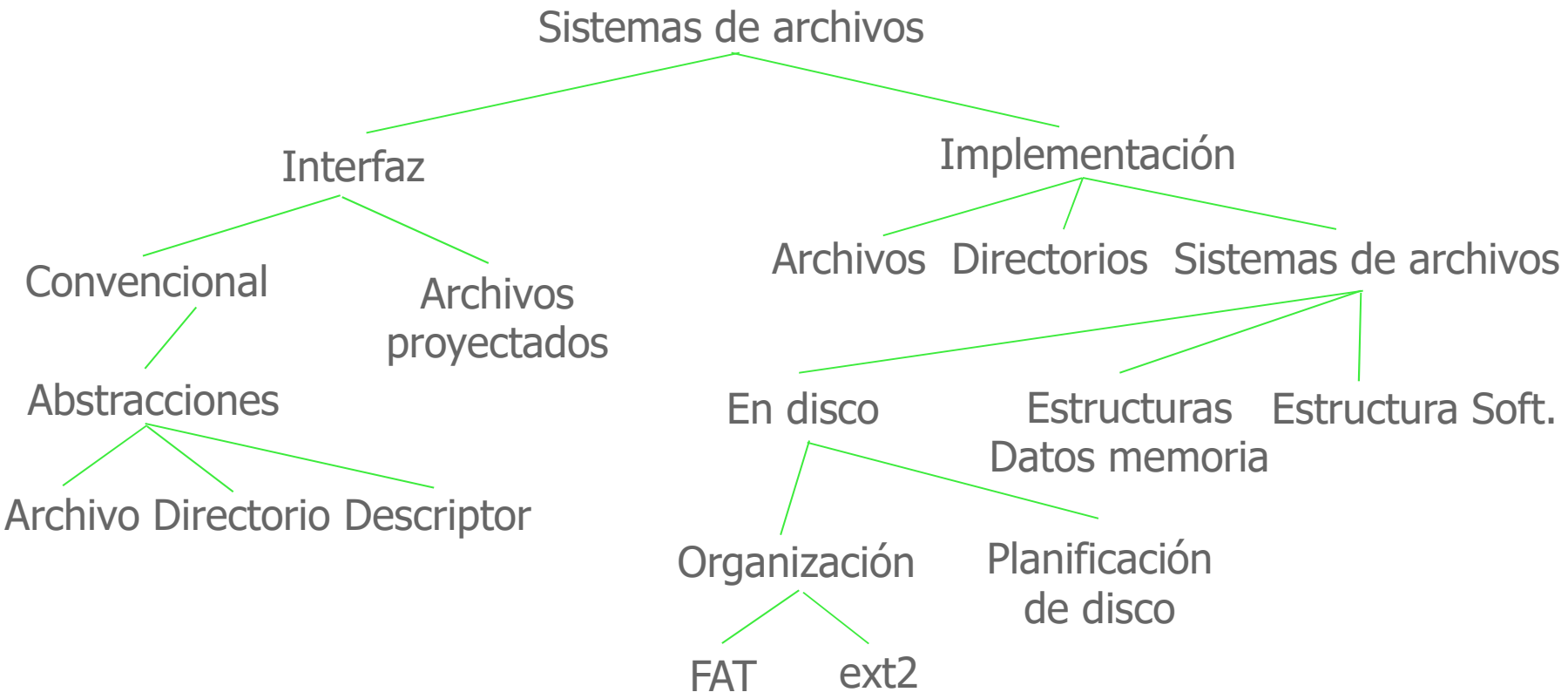
Objetivos

- Analizaremos las abstracciones que suministra el sistema operativo relacionadas con el almacenamiento permanente y como se implementan.
- Veremos las estructuras de datos de memoria que utiliza el sistema operativo para manejar archivos.
- Los métodos utilizados para asignar espacio en disco a los archivos, y cuales son sus principales ventajas e inconvenientes.
- Cómo se sigue la pista al espacio libre de disco.

Contenidos

- Interfaz de los sistemas de archivos.
- Diseño del software del sistema de archivos.
- Implementación de los sistemas de archivos.

Estructura del Tema



0. Revisión

Conceptos del Tema 4 de FS

Conceptos básicos

- Principales conceptos:
  - **Fichero** – unidad de almacenamiento persistente.
  - **Sistema de ficheros** – Componentes del SO que definen y establecen cómo se estructuran, identifican y manejan los archivos. Dos visiones:
  - **Estructura en disco** – Ejemplo, formatear disco.
  - **Software del SO para gestionar archivos** – oferta parte de la API

Directorios

- Un **Directorio** es un catalogo de nombres de usuario de archivos y la descripción de los propios archivos.
- Es un “archivo especial”, se implementa como TDA.
- Operaciones para manipularlo: `opendir()`, `readdir()`, `closedir()`, `seekdir()`, `...`
- Si bien hay diferentes implementaciones:
  - Preferible *estructura de árbol* – facilita la designación

1. Interfaz de los sistemas de archivos

Cómo accedemos a los archivos y directorios

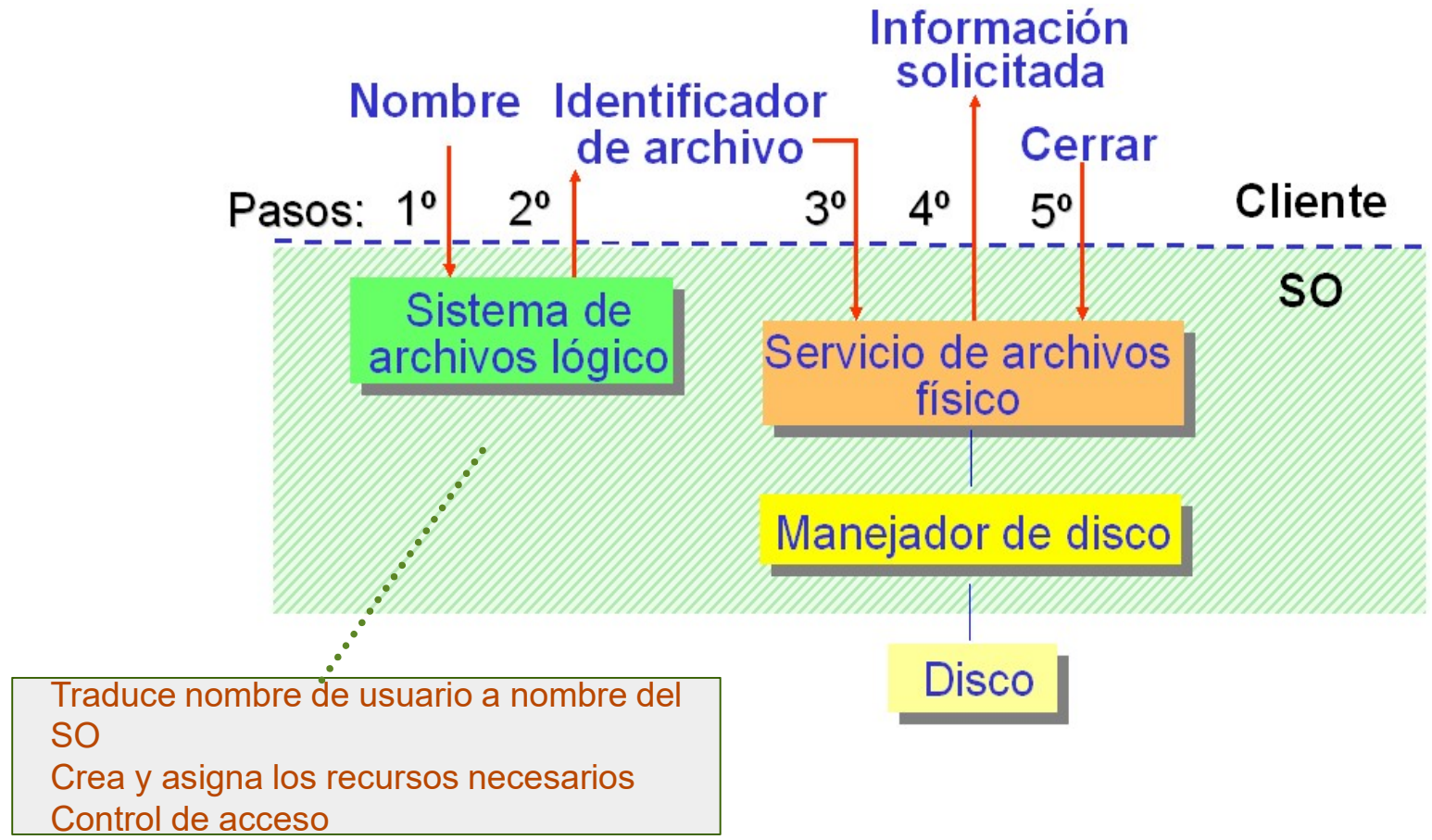
Interfaz: programa ejemplo (i)

```
# Programa de copia de un archivo
#include ...
#define BLKSIZE 1024
void main(int argc, char *argv[]) {
   int from_fd, to_fd, bytesread, byteswritten;
   char buf[BLKSIZE];
   char *bp;
   if (argc != 3) {
      fprintf(stderr, "Usage: %s from_file to_file\n", argv[0]);
      exit(1);
   }
   if ((from_fd = open(argv[1], O_RDONLY)) == -1) {
     . . .
      exit(1);
   }
   if ((to_fd = open(argv[2], O_WRONLY | O_CREAT | O_EXCL, S_IRUSR |
S_IWUSR)) == -1)
      . . .
      exit(1);
   }
```

Interfaz: programa ejemplo (ii)

```
while (bytesread = read(from_fd, buf, BLKSIZE)) {
      if ((bytesread == -1) && (errno != EINTR))
         break;              /* real error occurred on the descriptor */
      else if (bytesread > 0) {
         bp = buf;
         while(byteswritten = write(to_fd, bp, bytesread)) {
             if ((byteswritten == -1) && (errno != EINTR))
                break;
             else if (byteswritten == bytesread)
                break;
             else if (byteswritten > 0) {
                bp += byteswritten;  bytesread -= byteswritten;
             }
         }
         if (byteswritten == -1)
             break;
      }   }
   close(from_fd);
   close(to_fd);
   exit(0);
}
```

Esquema de funcionamiento



Reflexión

- La elección de las funciones de la API no es neutra, los diseñadores optaron por favorecer los tipos de operaciones más frecuentes en detrimento de las menos frecuentes.
  - Optimizar caminos frecuentes – menos llamadas al sistema
  - Favorecer accesos secuenciales

Atributos de un archivo

- Los **atributos** (metadatos) de un archivo son los datos que mantiene el SO para describir el mismo, es decir implementan la abstracción archivo.
- Algunos de estos atributos:
  - Nombre, tipo, ubicación, tamaño, protección, tiempos de creación, modificación y último acceso
  - Atributos extendidos:
    - En NTFS, Solaris, Apple: *Alternate Data Streams*: ej. crear/ver un flujo de un archivo:

```
c:\> echo Hola esto es un ADL > file.txt:Flujo.txt
c:\> more <file.txt:Flujo.txt
Hola esto es un ADL
```

Seudo-sistemas de archivos

- La interfaz para manipular archivos se ha mostrado muy potente y flexible.
- Idea: utilizar esta interfaz para acceder a datos o recursos del SO) que no son realmente sistemas de archivos:
  - Ejemplos: */proc*, */dev*, o */devfs*

Estructura de capas

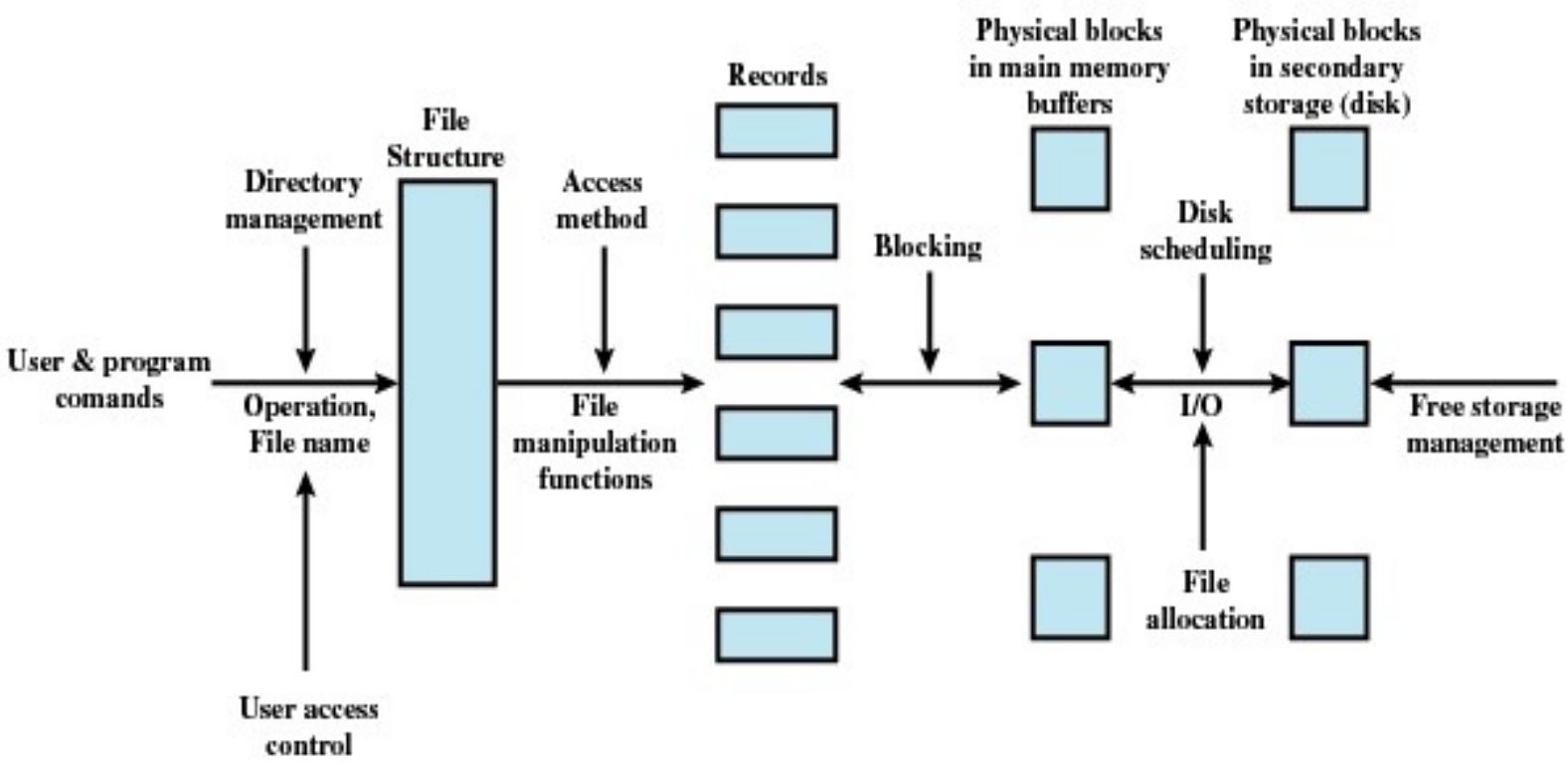


Figura 12.2 [Stallings2008]

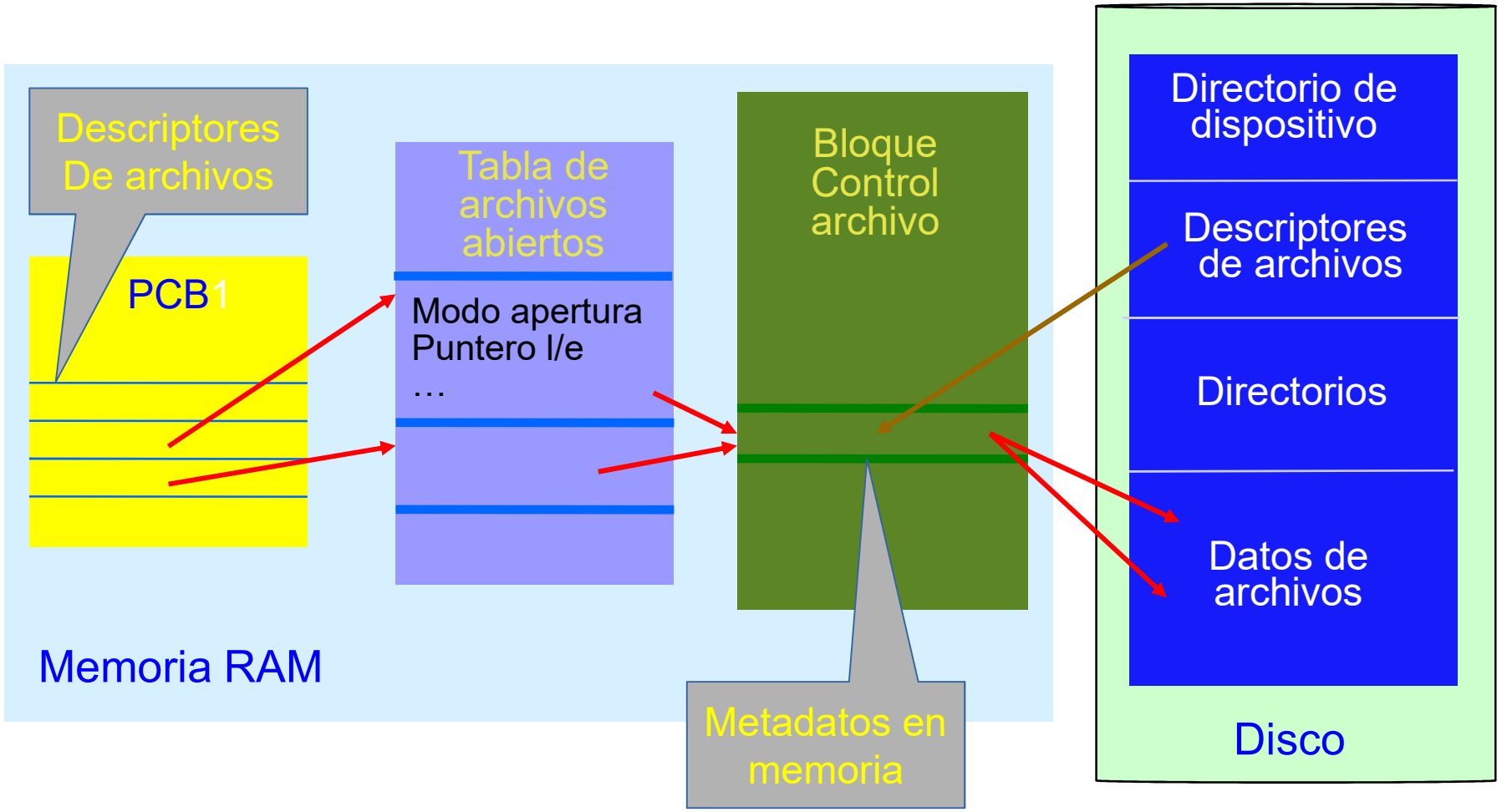
Búferes de E/S

- **Búfer** = memoria intermedia de almacenamiento temporal que se utiliza para mejorar el rendimiento.
- ¿Cómo mejoro el rendimiento de E/S?:
  - Acoplar velocidades entre CPU y dispositivos
  - Diferencias de tamaños de datos transferidos
  - Minimizar el tiempo en que un proceso esta bloqueado en espera de una operación L/E.
  - Desacoplar E/S de gestión de memoria.
- Los búferes se usan a nivel de SO y de bibliotecas.
- **Caché de disco** = conjunto de búferes de E/S de disco gestionados por un algoritmo de reemplazo.

2. Diseño software del sistema de archivos

Estructuras de datos en memoria y en disco para la gestión de los archivos

Estructura de datos de memoria

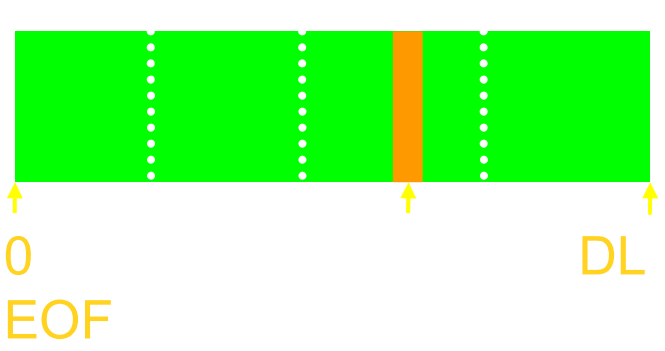


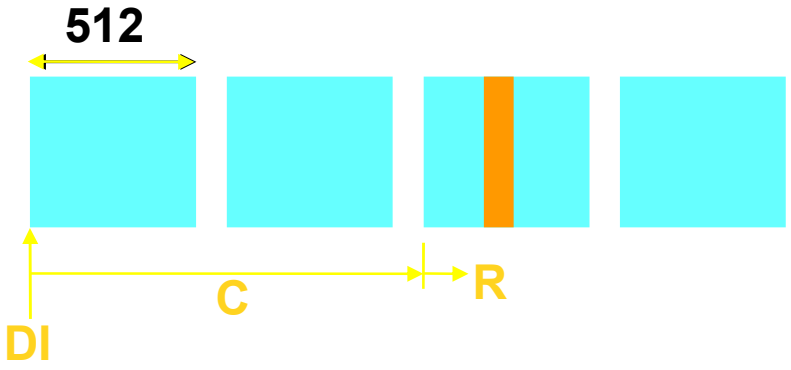
Gestión del almacenamiento secundario

- Métodos de asignación de ficheros:
  - Asignación contigua
  - Asignación encadenada o enlazada
  - Asignación indexada
- Gestión del espacio Libre
  - Tabla de bits
  - Porciones libres encadenadas
  - Indexación
  - Lista de bloques libres

Asignación contigua

- La correspondencia de una Dirección Lógica (DL) a una Física:
  - DL / Tam. Bloque = Cociente (C) y Resto ®
  - Bloque a acceder = C + Dirección_inicio
  - Desplazamiento en bloque = R



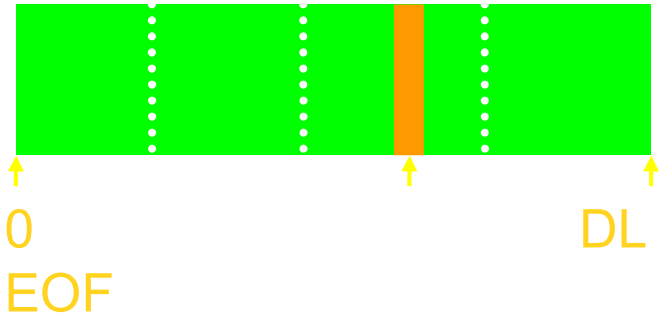


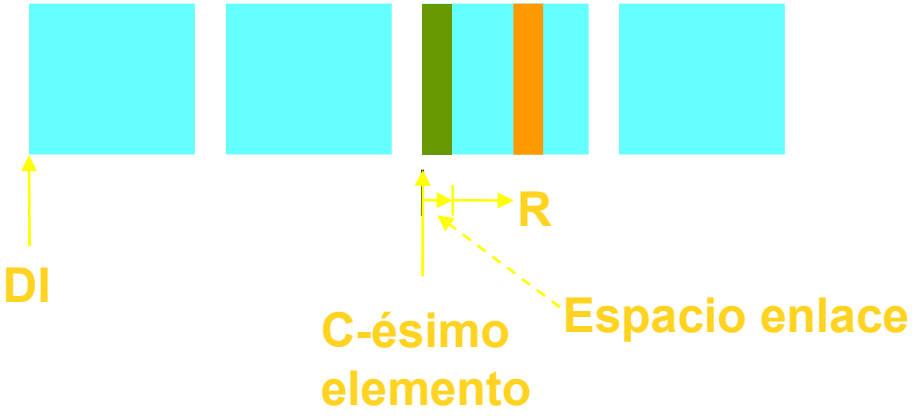
Asig. contigua: análisis

- Ventajas:
  - Sencillo; sólo necesitamos ubicación de comienzo (número bloque) y la longitud.
  - Acceso directo y rápido.
- Desventajas:
  - Asignación dinámica → fragmentación de disco.
  - Los archivos no pueden aumentar salvo que se realice compactación (es ineficiente).

Asignación enlazada

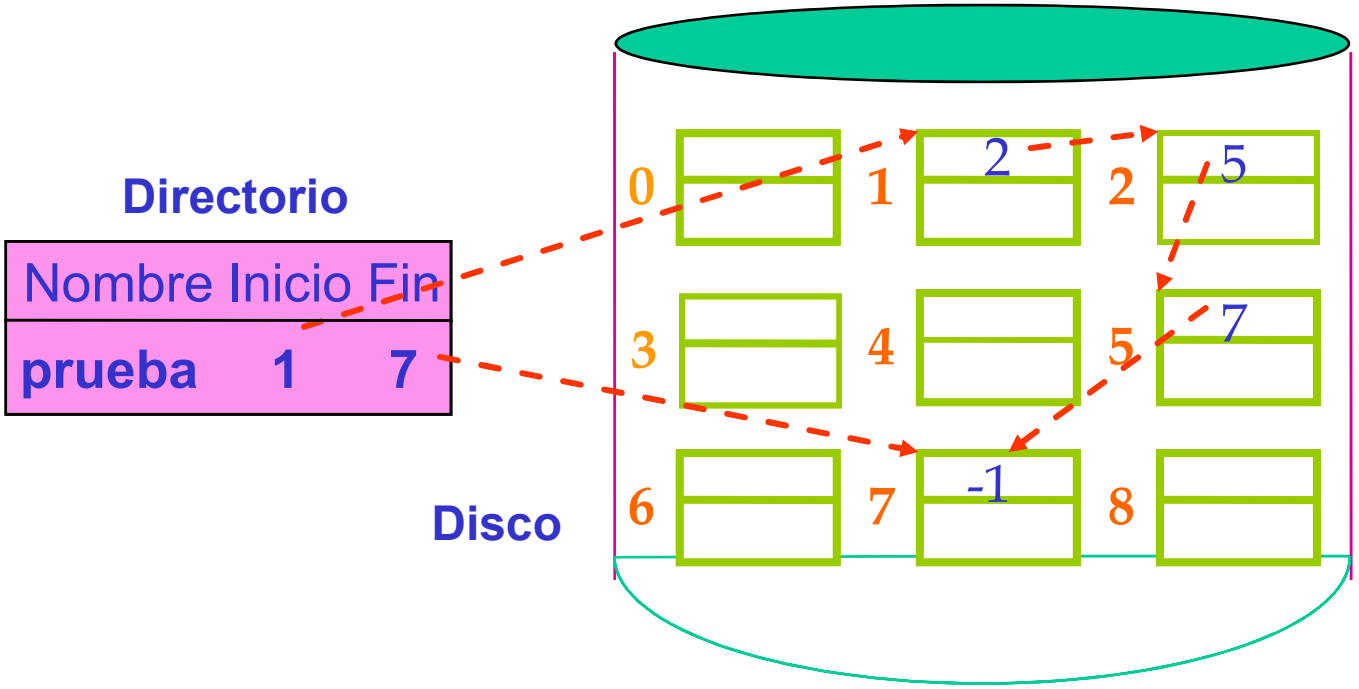
- Correspondencia: DL = C y R
  - Bloque acceder en C-ésimo bloque de la lista
  - Desplazamiento en el bloque = R +1





Asignación enlazada

- Asignar bloques conforme se necesitan y enlazarlos en una lista.
- P. ej. Sea el archivo con inicio en el bloque 1:

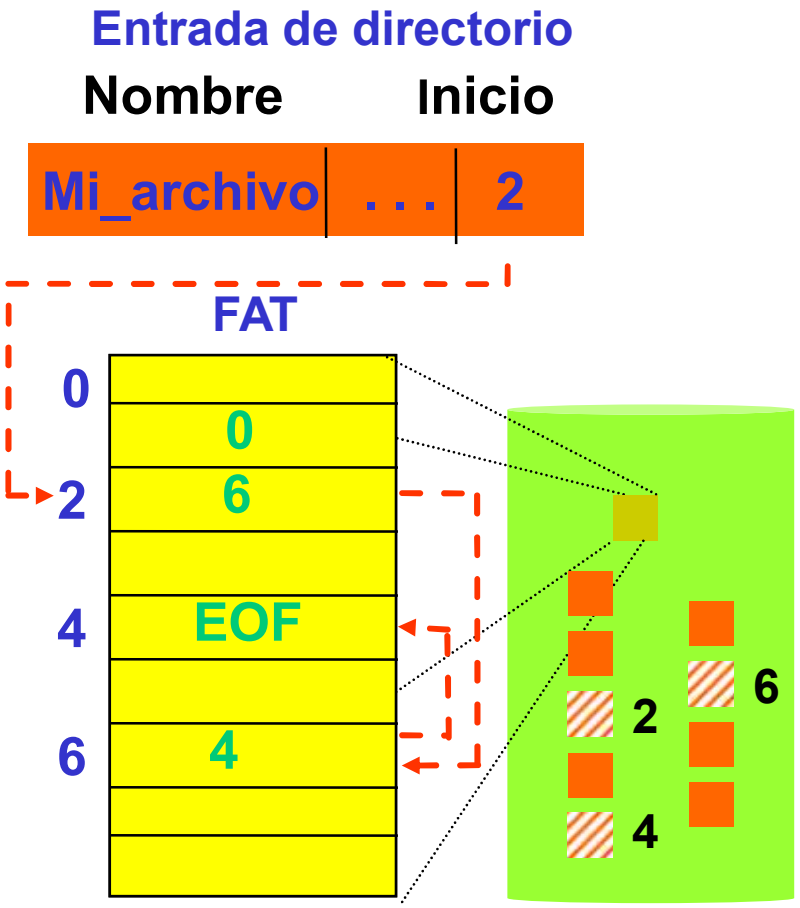


Asignación enlazada: análisis

- Ventajas:
  - Simple (sólo necesitamos dirección de inicio)
  - Crecimiento dinámico no fragmentación
- Inconvenientes:
  - Acceso aleatorio ineficiente.
  - Punteros consumen espacio; mejora : agrupación de bloques (*clúster*)
  - Problema de seguridad por perdida de punteros. Solución: lista doble enlazada sobrecarga.

Asignación enlazada: análisis

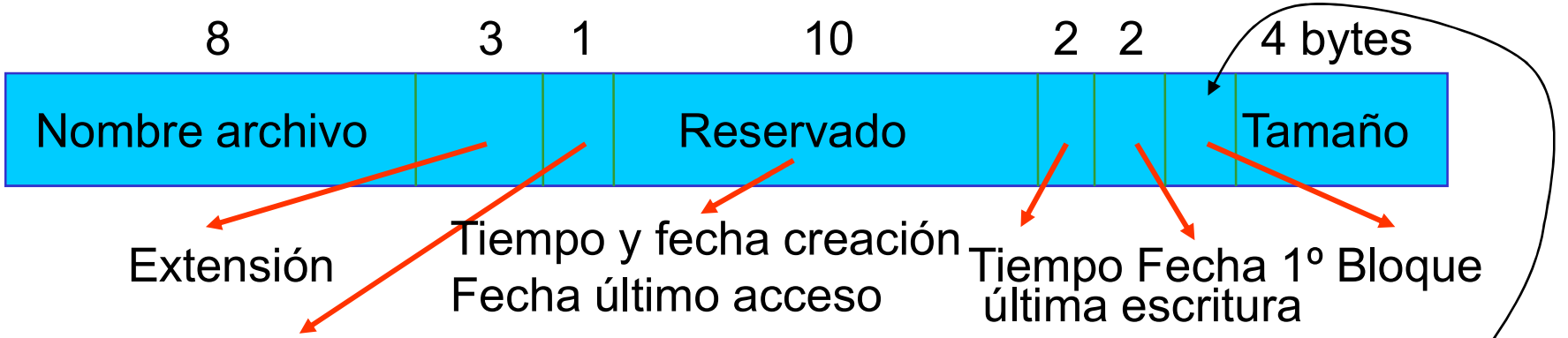
- Variante del método enlazado de Microsoft que actualmente se usa en los *pendrives*.
- Cada partición reserva un espacio para la FAT que contiene una entrada por bloque de disco y esta indexada por el número de bloque.
- Hay una copia de la FAT en caché para reducir el tiempo de búsqueda.



Sistemas de archivos FAT

- Estructuras de datos en el disco:

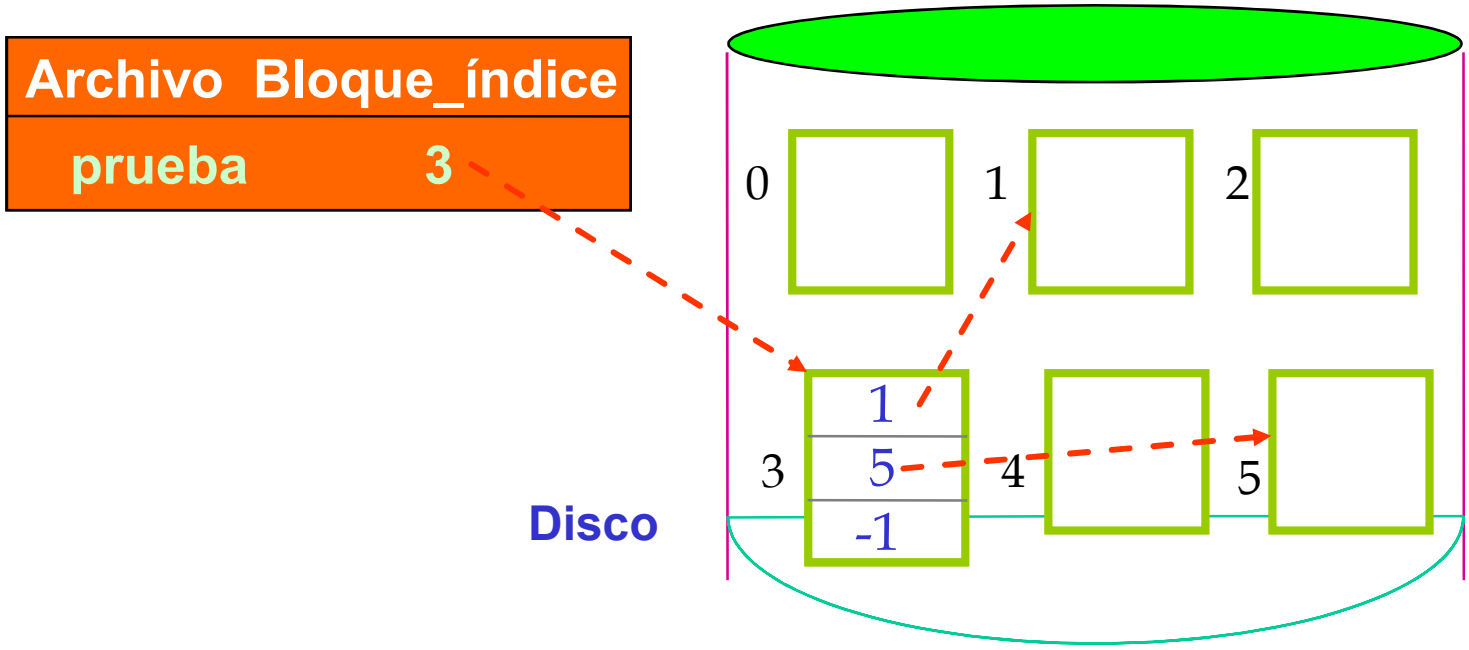
- Entrada de directorio:



Atributo (cada uno un bit):
R: Solo lectura
A: Archivo (1 cuando se modifica, 0 cuando se hace back-up)
S: Archivo de Sistema (no borrado con la orden *del*)
H: Archivo Oculto (no listado con *dir*)
D: Directorio
V: Etiqueta de Volumen

Asignación indexada

- Necesita uno o varios bloques que se usan como tabla de índices a bloques de datos por archivo.

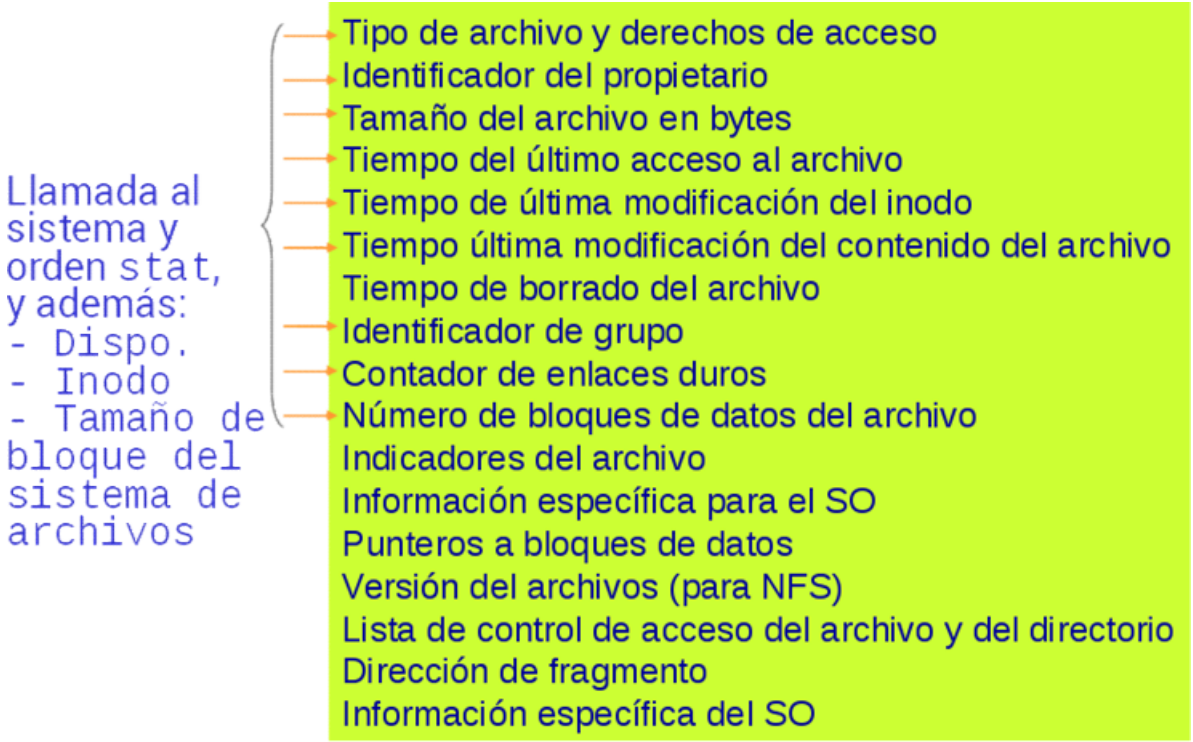


Asignación indexada: análisis

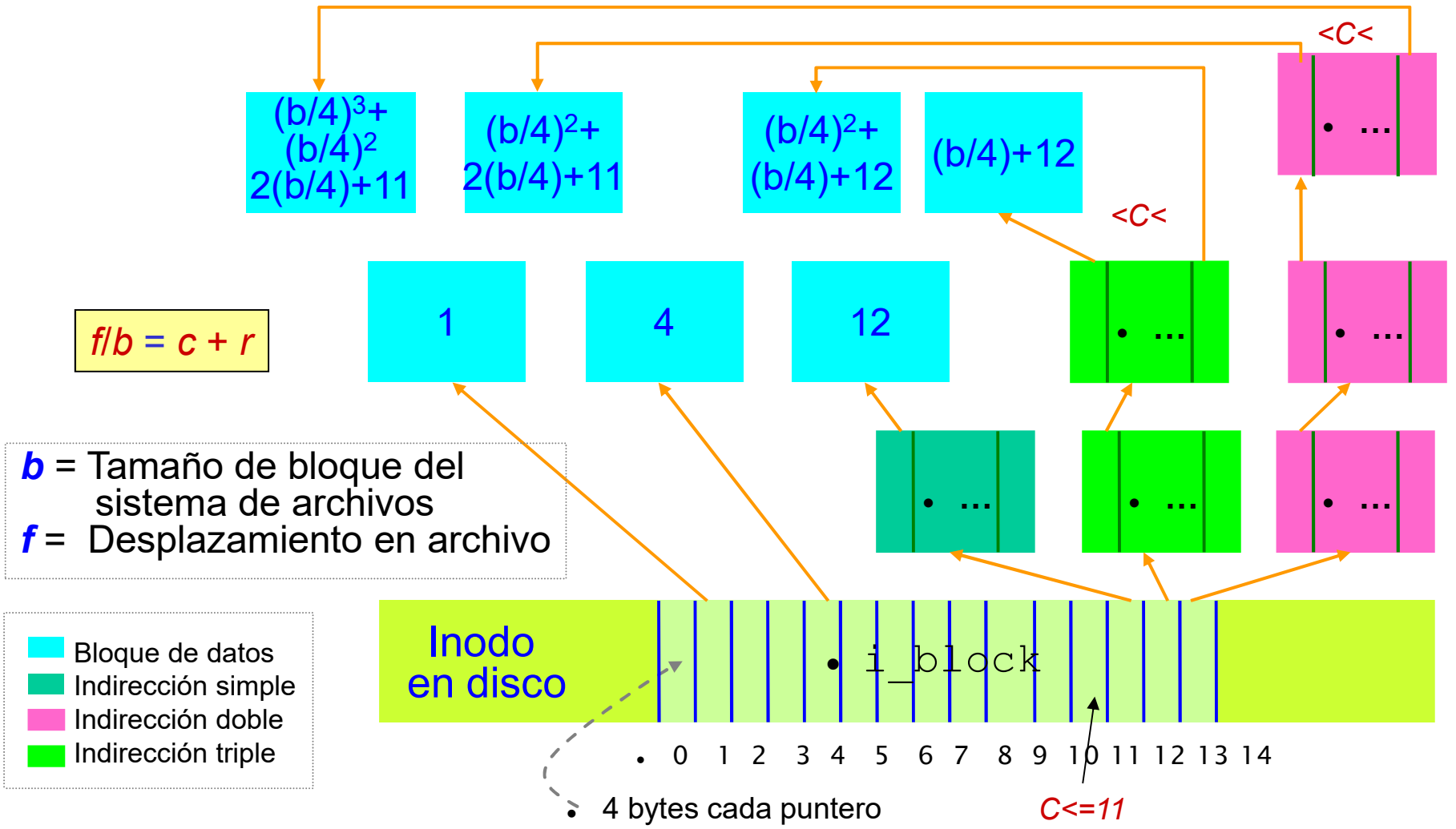
- Ventajas:
  - Acceso aleatorio
  - Acceso dinámico sin fragmentación, pero tiene la sobrecarga del bloque de índices.
- Desventajas:
  - Posible desperdicio de espacio en el bloque de índices
  - Tamaño del bloque de índices. Soluciones:
    - P. ej., UNIX utiliza bloques de índices enlazados multinivel.

Inodo en Linux

- Un archivo en disco viene representado por una estructura de datos denominada *inodo* (nodo índice).
- En sistemas *ext2* tiene la forma:

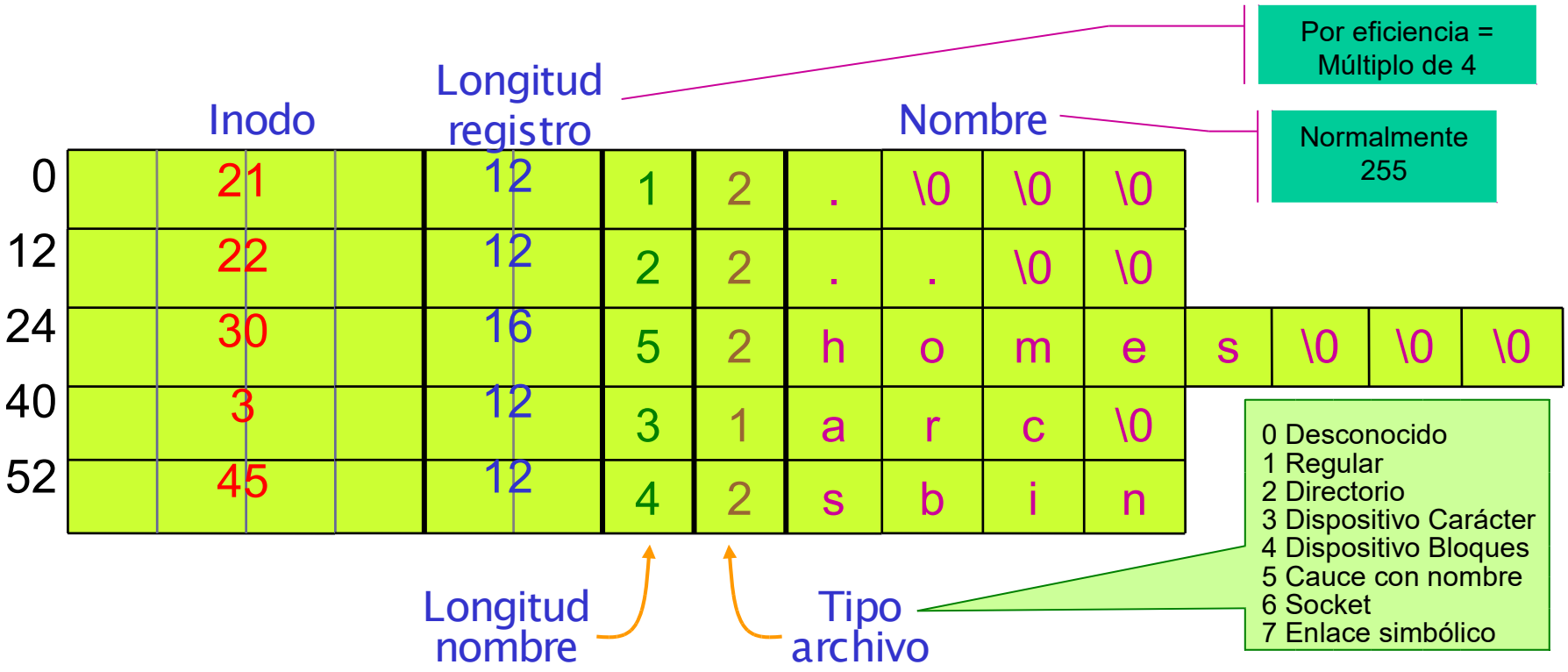


Inodo: direccionamiento de bloques



Directorios en Linux

- La estructura de datos directorio tiene la forma:

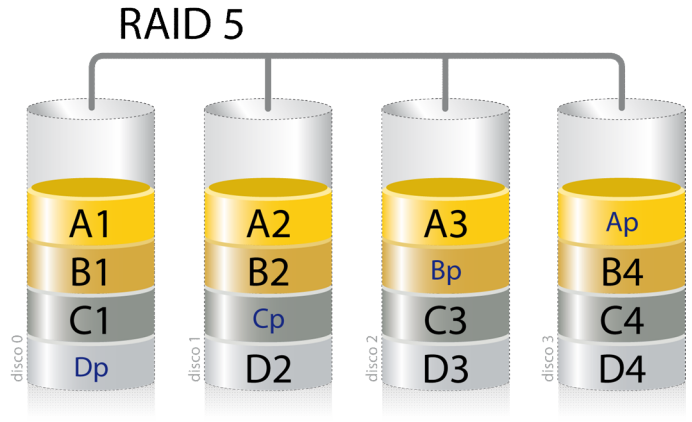


Gestión del área de intercambio

- El uso del área de intercambio depende de los algoritmos de gestión de memoria:
  - Intercambio - procesos completos.
  - Paginación - páginas de procesos.
- Lugar para la asignación de espacio en disco:
  - En el *sistema de archivos* (Windows) - Fácil de implementar pero ineficiente (fragmentación).
  - *Partición de disco* independiente (Linux, en general) - No utiliza estructura de directorios ni sistema de archivos, asignación contigua (un slot por página).

RAID

- RAID (Redundant Array of Inexpensive Disks): Consta de varios discos funcionando como una unidad donde cada bloque se descompone en sub-bloques que son almanenados en discos separados.
  - Las técnicas de despiece de datos mejora la tolerancia a fallos y realizan la transferencia y posicionamiento en paralelo.
- Varios esquemas:
  - RAID 0 – *Mirroring*
  - RAID 5 – Distribución con paridad



Fuente: https://es.wikipedia.org/wiki/RAID

Planificación de disco

- Planificación de disco: el SO reorganiza las peticiones de disco de cara a reducir el tiempo de posicionamiento del cabezal sobre el bloque de datos (distancia de posicionamiento).
  - Linux permite seleccionar varios algoritmos: *noop*, *deadline* y *cfq*. Podemos verlo con:
    - `$ cat /sys/block/sda/queue/scheduler`

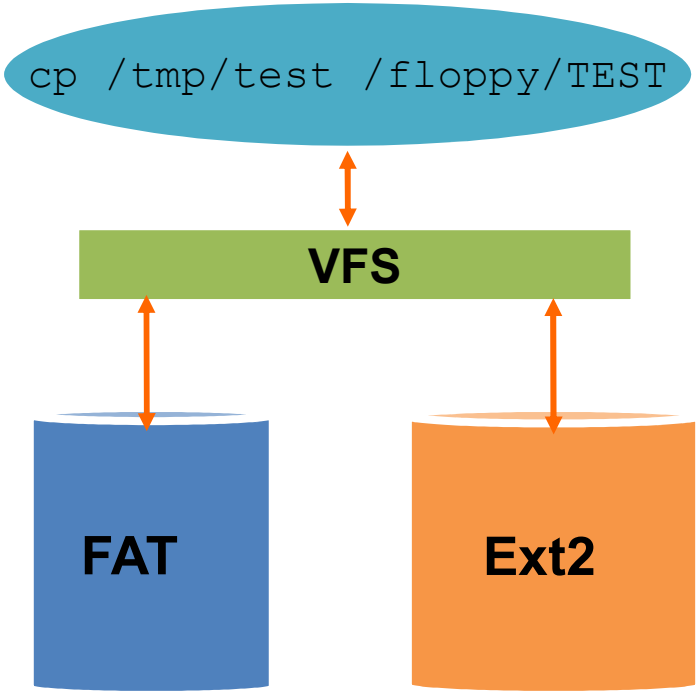
3. Implementación de los sistemas de archivos

Diseño e implementación de la gestión de archivos y directorios en Linux

Sistemas de archivos en Linux

- Linux introduce el Sistema de Archivos Virtual (VFS) con el objetivo de soportar diferentes tipos de sistemas de archivos.
- VFS es una capa software entre el usuario y los diferentes tipos de SA cuya misión es suministrar al usuario una interfaz única en el acceso a archivos independiente del sistema de archivos al que pertenece un archivo.

VFS: objetivo



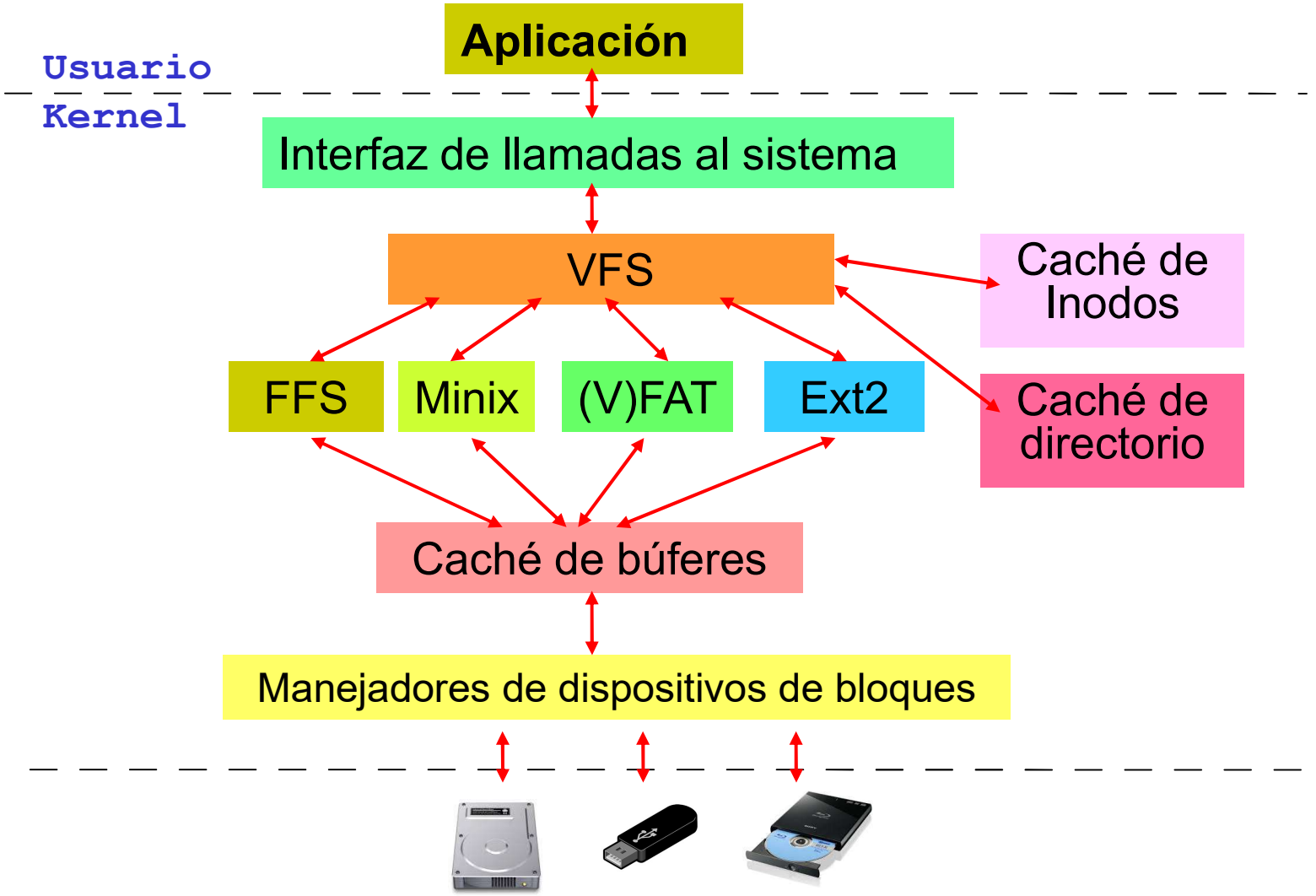

```
origen=open("floppy/TEST",
       O_RDONLY,0);
destino=open("/tmp/test",
   O_WRONLY|O_CREATE|O_TRUNCATE,
       0600);
do {
    l = read(orige,buf,4096);
    write(destino, buf, l);
} while (l);
close(origen);
close(destino);
```

Podemos ver como las operaciones para acceder a archivos son independientes del tipo de sistema de archivos.

Sistemas de archivos soportados

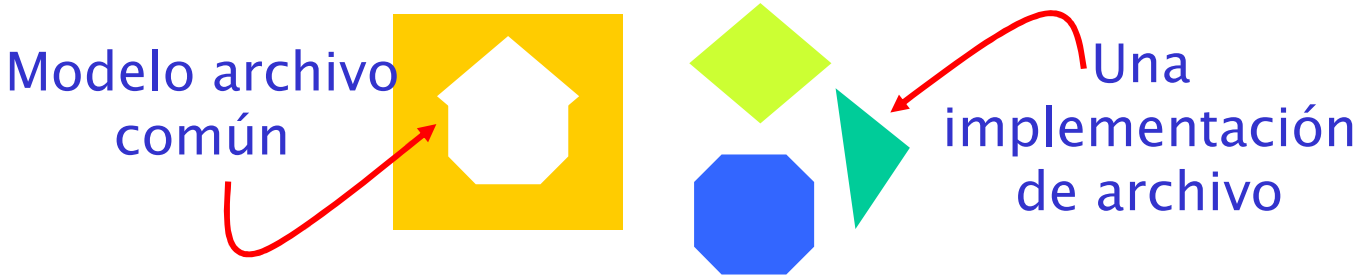
- Los sistemas soportados por VFS podemos agruparlos en:
  - Basados en disco – sistemas que gestionan una partición de disco, p. ej. Ext2, ms-dos, VFAT, NTFS, CDROM, FFS, ..
  - Basados en red – sistemas de archivos en red como NFS, SMB, AFS, …
  - Especiales – no gestionan disco sino que son la interfaz a otro tipo de objetos, como son /proc, /dev/pts.
- Cualquier sistema, una vez montado, tiene la misma forma de acceso.

VFS: estructura



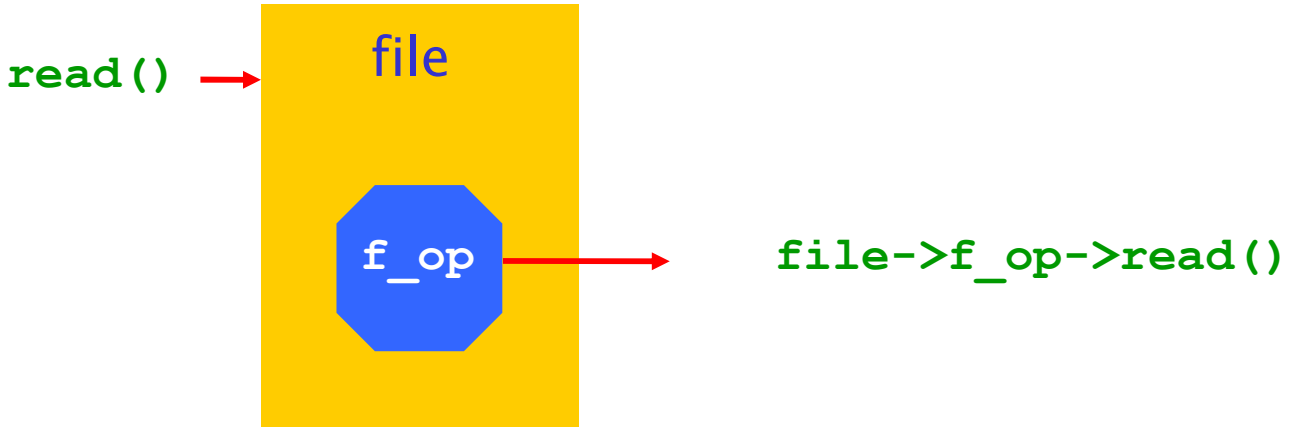
VFS: modelo archivo común

- Construir un modelo de **archivo común** capaz de representar todos los archivos de los diferentes sistemas de archivos soportados.
- Un sistema de archivos específico debe traducir su organización física a la del modelo de archivo común de VFS.



Archivo común: implementación

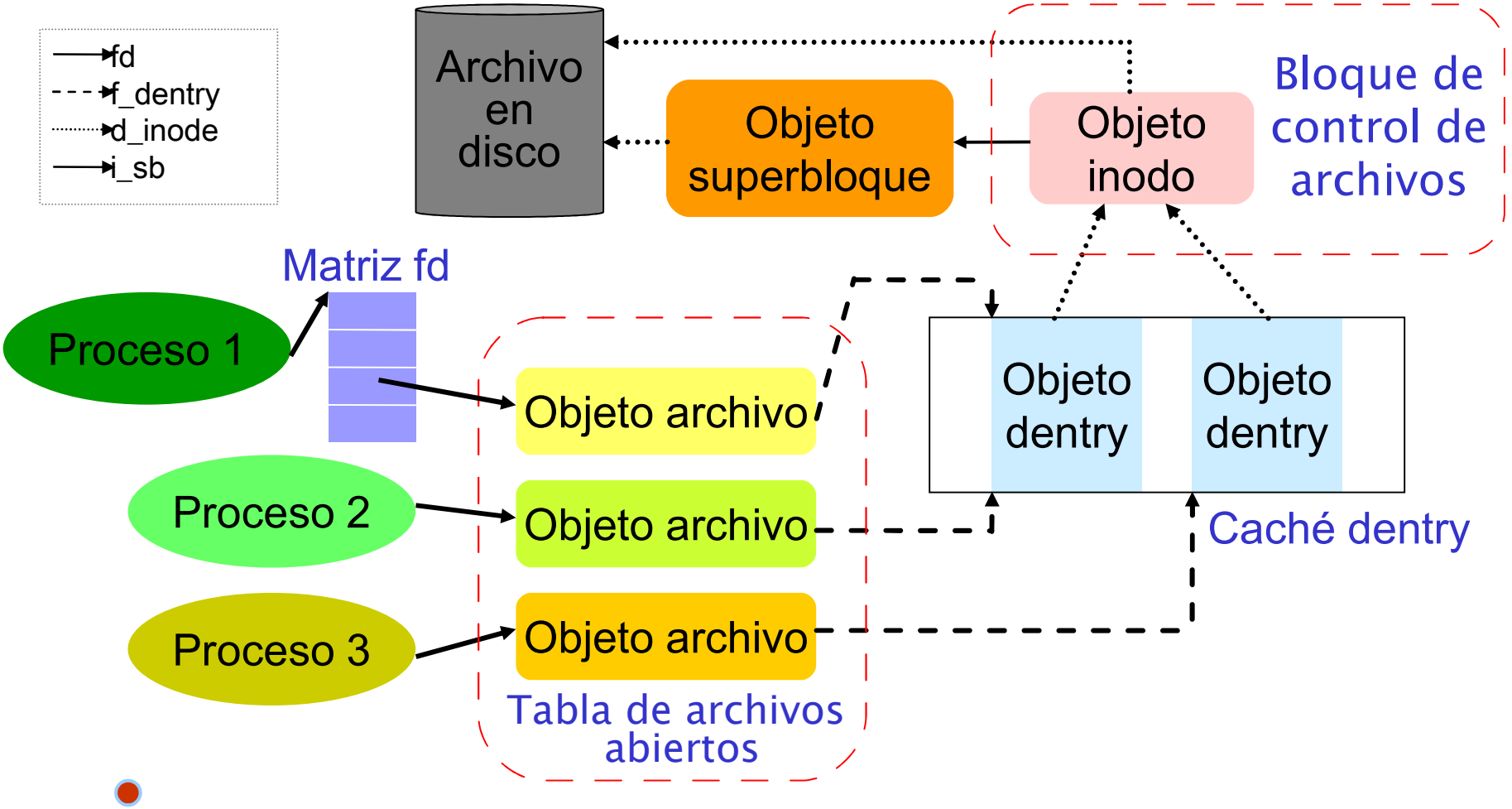
- Un modelo archivo común utiliza el modelo objeto (construcción software que define los datos y los métodos para operar sobre ellos).
- El archivo común es un objeto que contiene punteros a las funciones que implementan los métodos sobre una implementación concreta de sistema de archivos.



VFS: tipos de objetos

- Las estructuras de datos en memoria que usa VFS son:
  - **Objeto superbloque** – almacena información de un sistema de archivos montado. Se corresponde con el directorio de dispositivo.
  - **Objeto inodo** – almacena información de un sistema de archivos específico (bloque de control de archivos).
  - **Objeto archivo** – Describe la interacción entre un archivo abierto y un proceso (Tabla de archivos abiertos).
  - **Objeto dentry** – Enlace entre una entrada de directorio y el archivo correspondiente.

VFS: interacción procesos y objetos



Objeto inodo

- Objeto inodo, o inodo en memoria, es la ED que contiene los metadatos de un archivo en disco más campos dedicados a la gestión de las caché de inodos (listas de inodos sin usar, inodos en uso, y inodos sucios).
- Algunos de los campos más relevantes a nuestro estudio:
  - `i-list` –puntero a la lista de inodos.
  - `i_dentry` – punteros a la lista dentry.
  - `i_count` – contador de uso.
  - `i_op` – operaciones sobre inodos.
  - `u` – información específica del sist. Archivos.

Objeto archivo

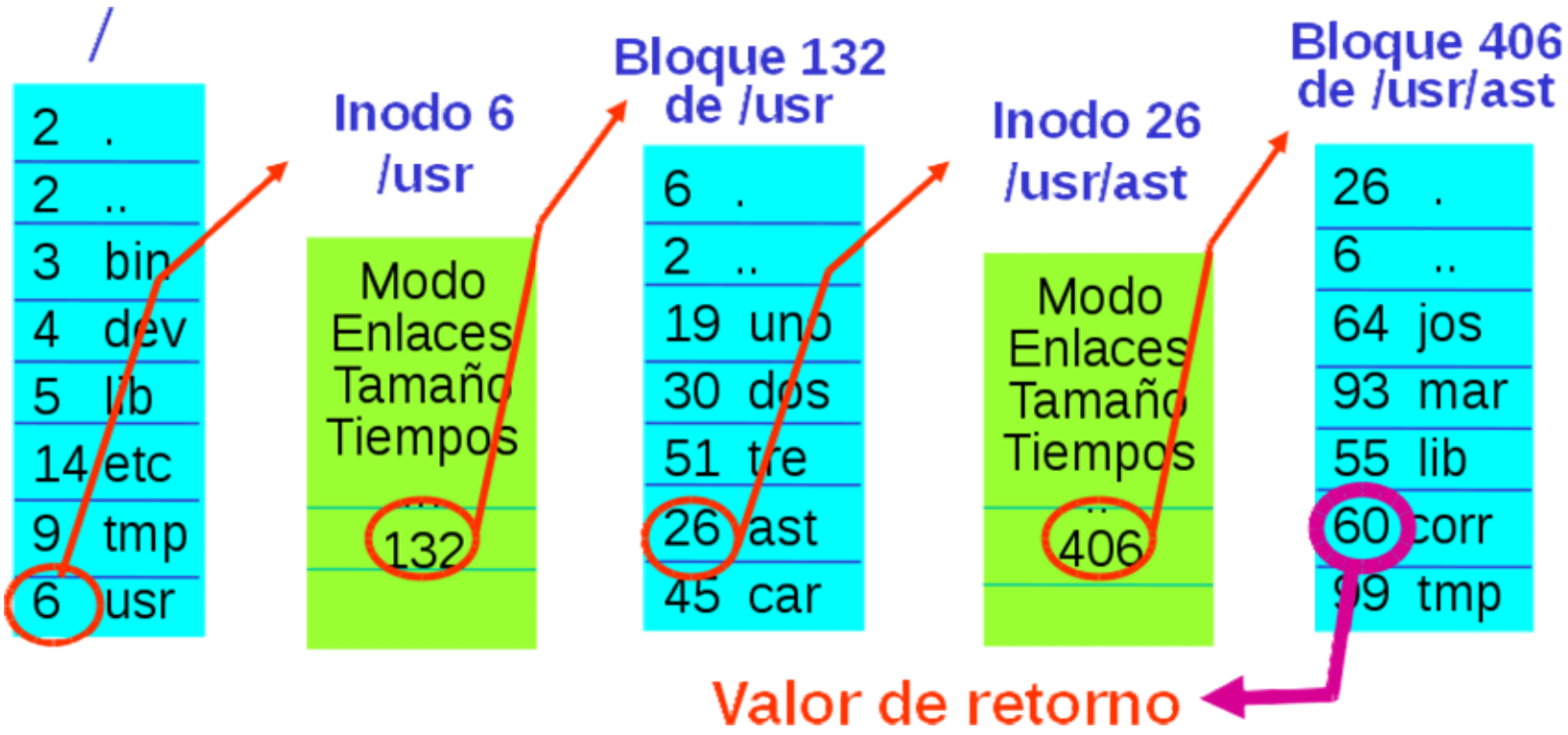
- Creamos el objeto cuando abrimos un archivo, y viene representado por una estructura `file`.
- Estos objetos no tiene imagen en disco pues representan una sesión de trabajo sobre un archivo.
- Los principales campo de este estructura:
  - `f_pos` – puntero de l/e.
  - `f_mode` – r/w, permisos, etc.
  - `f_op` – tabla de operaciones de archivos.
  - `f_count` – contador de uso, !=0 si esta en uso.
- Información para mantener los objetos en diferentes listas.

Traducción de nombres

- Los sistemas Unix usan dos espacios de nombres para archivos:
  - Espacio de nombres del SO – *inodos*.
  - Espacio de nombres de usuario - *pathname*
- El kernel debe establecer la correspondencia (traducir) entre los nombres de usuario y sus correspondientes inodos.
- Esta traducción es lenta y en algunos casos compleja (enlaces y puntos de montaje).

Traducción de nombres: `namei`

- ¿Traducir */usr/ast/corr* en su número de inodo?

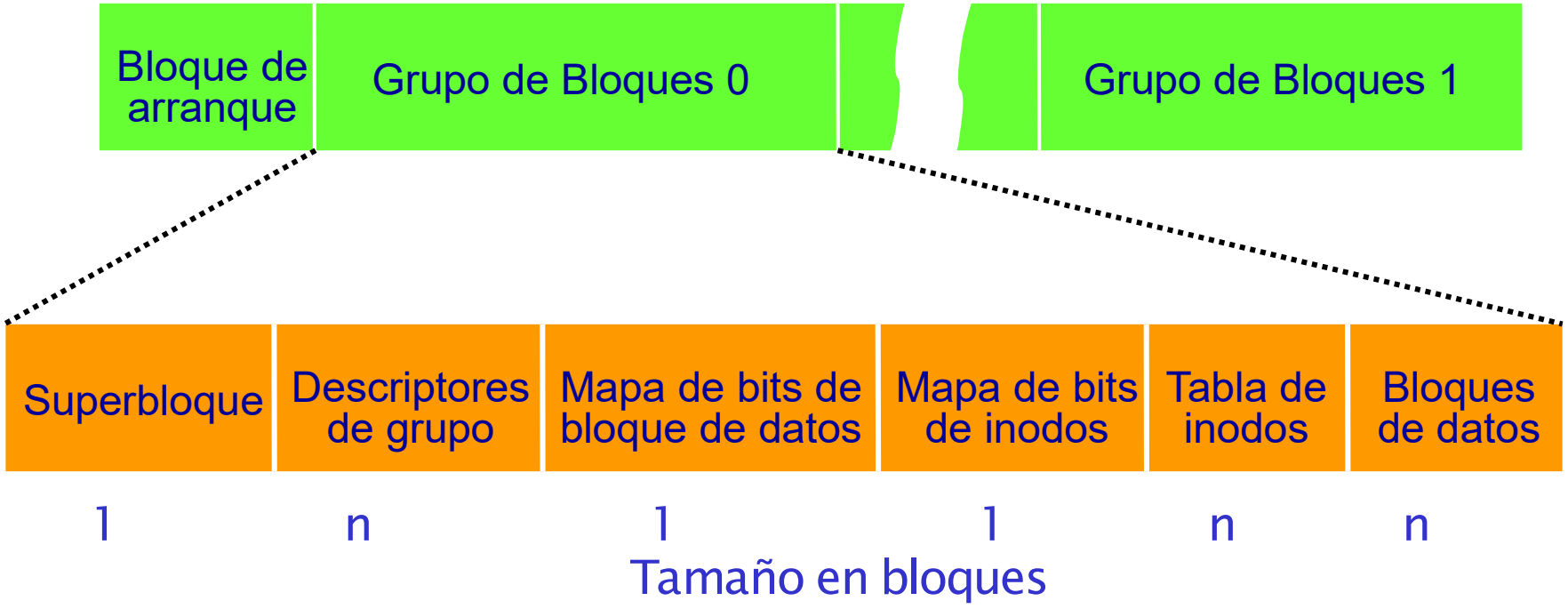


Objeto dentry (Directory Entry)

- El objeto *dentry* asocia el componente del *pathname* con su inodo asociado.
- Cuando VFS lee en memoria una entrada de directorio la transforma en objeto dentry (estructura dentry).
- Hay un objeto dentry por cada componente de pathname que busca un proceso. SE usan como cache para acelerar la traducción de nombres de archivos.
- Algunos campos:
  - `d_name` – nombre en la entrada
  - `d_inode` – inodo del archivo
  - `d_mounts` – dentry del raíz de un FS montado.
  - `d_covers` – dentry de un punto de montaje

Sistema de archivos *ext2* en disco

- La estructura de una partición ext2 en disco y de un grupo de bloques es:



Grupos de bloques

- Se introducen para reducir la fragmentación del disco - los bloques de datos de un archivo se intentan almacenar en el mismo grupo de bloques (menor distancia posicionamiento).
- Todos los GBs mantienen copia de seguridad del superbloque y de los descriptores de grupos, pero solo de utiliza la de GB0 (se puede indicar a */sbin/e2fsck* que utilice otra).
- Todos los grupos de bloques tienen igual tamaño y se almacenan secuencialmente. Su tamaño depende del tamaño de disco con una única restricción: el mapa de bits debe ocupar un bloque. Si *b*= tamaño de bloque (bytes), y *s*= tamaño en bloques de la partición de disco:
  - Un grupo debe tener como máximo un tamaño de 8 * b.
  - La partición tiene aproximadamente un número de grupos de s/(8*b).