4 Gestión de Archivos

- Introducción
- Diseño
- Interfaz
- Implementación

Objetivos

- ✓ Comprender el concepto de archivo, directorio y estructura de directorios
- ✓ Saber las distintas funciones básicas que se realizan sobre un sistema de archivos y cómo se pueden estructurar
- ✓ Conocer las ventajas e inconvenientes de distintos métodos de asignación de espacio del almacenamiento secundario y de gestión de espacio libre
- ✓ Abordar distintos aspectos de implementación y valorar sus ventajas e inconvenientes
- ✓ Comprender la importancia de la gestión del almacenamiento secundario y cómo se puede optimizar su funcionamiento

Concepto de archivo

- Colección de información relacionada y almacenada en un dispositivo de almacenamiento secundario
- Espacio de direcciones lógicas contiguas
- **Estructura interna** (lógica)
  - secuencia de bytes: el tipo del archivo determina su estructura (texto → caracteres,líneas y páginas, código fuente → secuencia de subrutinas y funciones)
  - secuencia de registros de longitud fija
  - secuencia de registros de longitud variable
- **Tipos** de archivos: regulares, directorios, de dispositivo
- **Formas de acceso**: secuencial, aleatorio, otros

Concepto de archivo (y II)

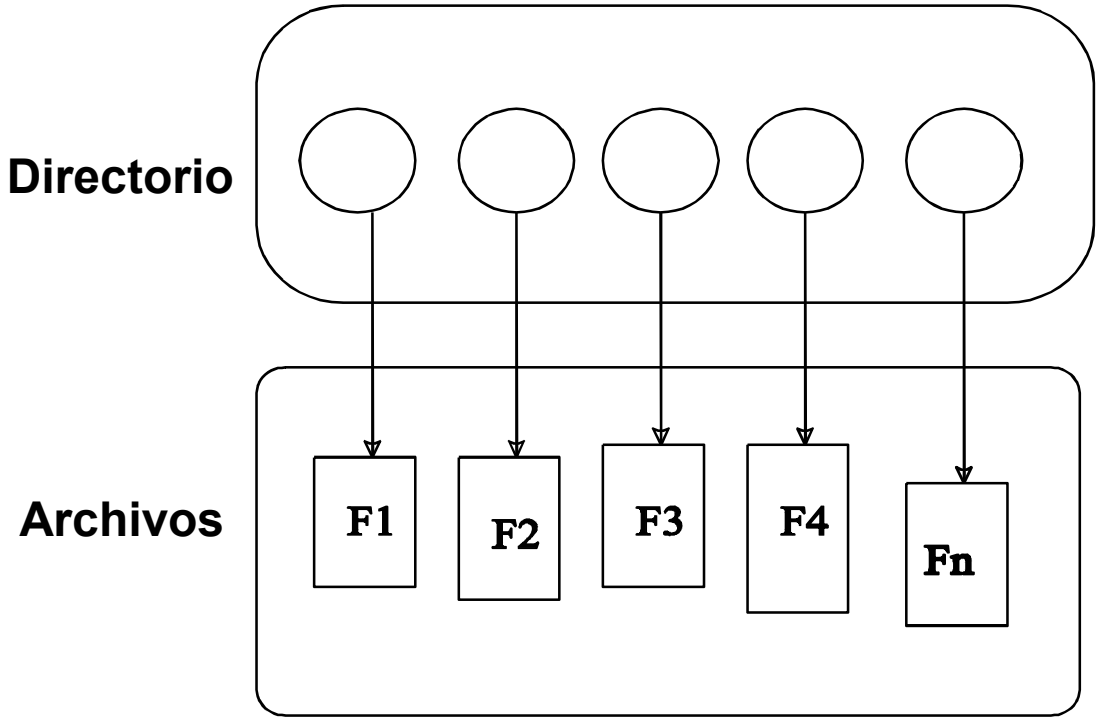
- Atributos (metadatos)
  - **Nombre**: única información en formato legible
  - **Tipo**: cuando el sistema soporte diferentes tipos
  - **Localización**: información sobre su localización en el dispositivo
  - **Tamaño**: tamaño actual del archivo
  - **Protección**: controla quién puede leer, escribir y ejecutar
  - **Tiempo**, **fecha** e **identificación** del usuario: necesario para protección, seguridad y monitorización

Concepto de archivo (y III)

- Operaciones sobre archivos
  - **Gestión**:
    - Crear
    - Borrar
    - Renombrar
    - Copiar
    - Establecer y obtener atributos
  - **Procesamiento**:
    - Abrir y Cerrar
    - Leer
    - Escribir (modificar, insertar, borrar información)

Estructura de Directorios

- Colección de nodos conteniendo información acerca de todos los archivos → **Organización**



- Tanto la estructura de directorios como los archivos residen en el almacenamiento secundario

Estructura de Directorios (y II)

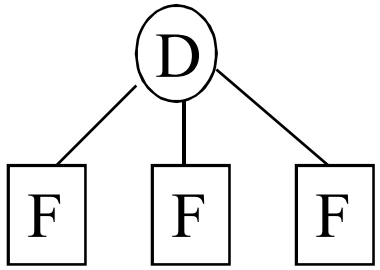
La organización (lógica) de los directorios debe proporcionar:

- **Eficiencia**: localización rápida de un archivo
- **Denominación**: adecuada a los usuarios
  - Dos usuarios pueden tener el mismo nombre para diferentes archivos
  - El mismo archivo puede tener varios nombres
- **Agrupación**: agrupar los archivos de forma lógica según sus propiedades (Ej. todos los programas en C)

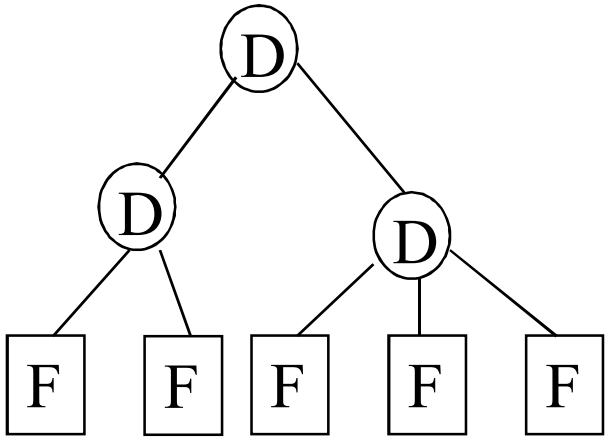
Estructura de Directorios (y III)

Casos:

- **A un único nivel**
  - Problema de denominación
  - Problema de agrupación

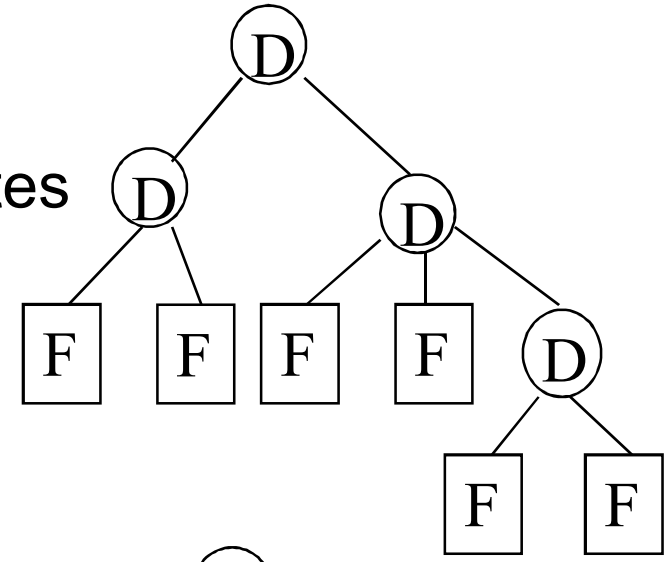


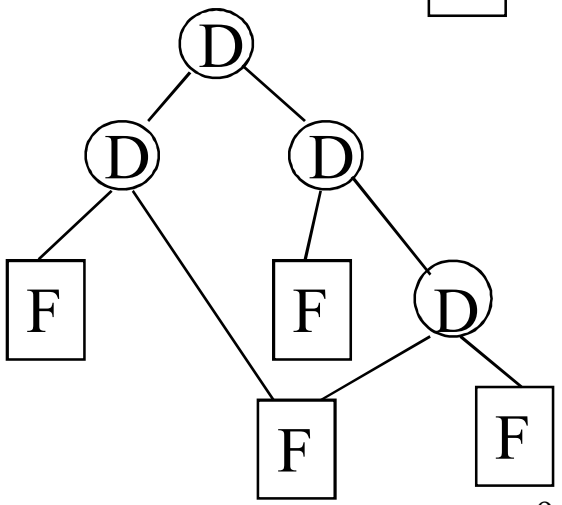
- **A dos niveles**
  - Nombres de camino
  - Diferentes usuarios pueden tener archivos con igual nombre
  - No hay posibilidad de agrupación



Estructura de Directorios (y IV)

- **En árbol**
  - Necesidad de búsquedas eficientes
  - Posibilidad de agrupación
  - Directorio actual (de trabajo)
  - Nombres de camino absolutos y relativos
- **En grafo**
  - Compartición de subdirectorios y archivos
  - Más flexibles y complejos





Protección

- Básicamente consiste en proporcionar un acceso controlado a los archivos
  - lo que puede hacerse
  - por quién
- Tipos de acceso
  - Leer
  - Escribir
  - Ejecutar
  - Añadir
  - Borrar
  - Listar

Protección: Listas y Grupos de acceso

- Principal solución a la protección: hacer el acceso dependiente del **identificativo del usuario**
- Las **listas de acceso** de usuarios individuales tiene el problema de la longitud
- Solución con clases de usuario
  - propietario
  - grupo
  - público
- Propuesta alternativa: Asociar un *password* con el archivo. Problemas:
  - Recordar todos
  - Si solo se asocia un *password* → acceso total o ninguno

Semánticas de consistencia

- Especifican cuándo las modificaciones de datos por un usuario se observan por otros usuarios
- Ejemplos:

1. **Semántica de Unix**
   - La escritura en un archivo es directamente observable
   - Existe un modo para que los usuarios compartan el puntero actual de posicionamiento en un archivo
2. **Semánticas de sesión** (Sistema de archivos de Andrew)
   - La escritura en un archivo no es directamente observable
   - Cuando un archivo se cierra, sus cambios sólo se observan en sesiones posteriores
3. **Archivos inmutables**
   - Cuando un archivo se declara como compartido, no se puede modificar

Funciones básicas del Sistema de Archivos

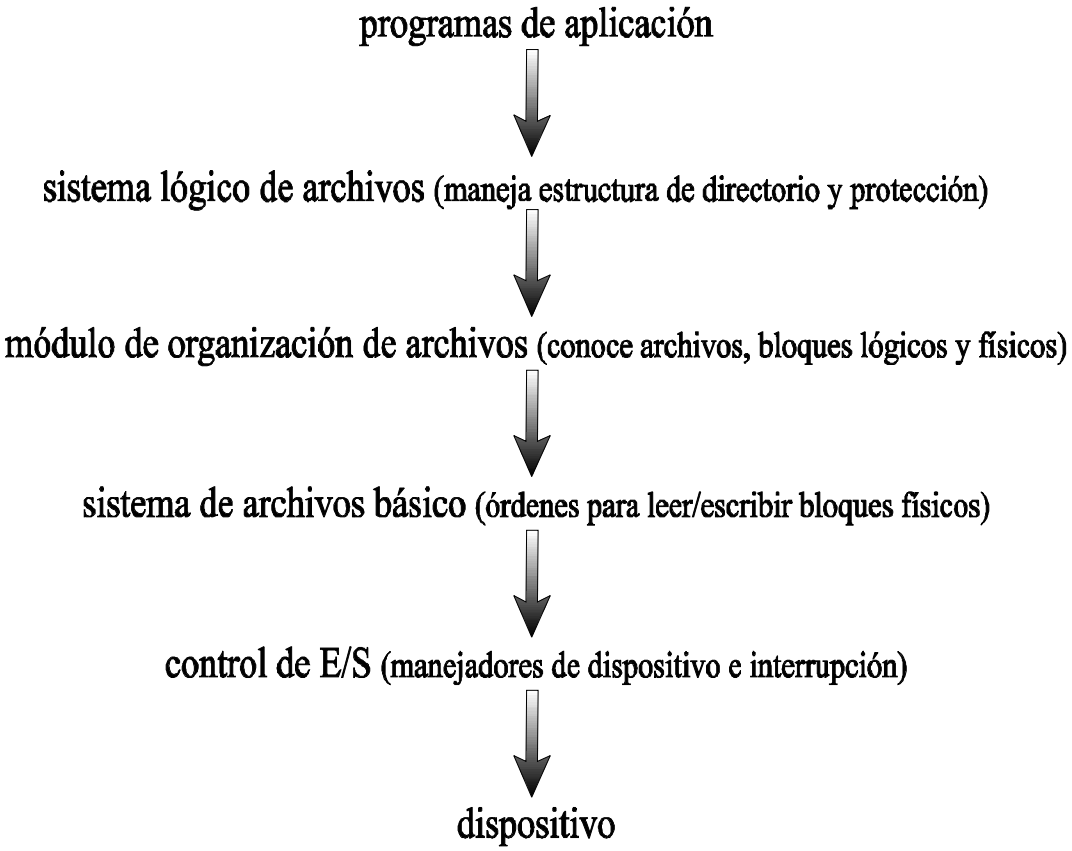
- Tener conocimiento de todos los archivos del sistema
- Controlar la compartición y forzar la protección de archivos
- Gestionar el espacio del sistema de archivos
  - Asignación/liberación del espacio en disco
- Traducir las direcciones lógicas del archivo en direcciones físicas del disco
  - Los usuarios especifican las partes que quieren leer/escribir en términos de direcciones lógicas relativas al archivo

Estructura del Sistema de Archivos

- Un sistema de archivos posee dos **problemas de diseño** diferentes:
  1. Definir cómo debe ver el usuario el sistema de archivos
     - definir un archivo y sus atributos
     - definir las operaciones permitidas sobre un archivo
     - definir la estructura de directorios
  2. Definir los algoritmos y estructuras de datos que deben crearse para establecer la correspondencia entre el sistema de archivos lógico y los dispositivos físicos donde se almacenan

Estructura del Sistema de Archivos (y II)

- Organización en niveles (capas)



- Por eficiencia, el SO mantiene una tabla indexada (por **descriptor de archivo**) de archivos abiertos

- **Bloque de control de archivo**: estructura con información de un archivo en uso

Métodos de Asignación de espacio: Contiguo

- Cada archivo ocupa un conjunto de bloques contiguos en disco
- **Ventajas**
  - Sencillo: solo necesita la localización de comienzo (nº de bloque) y la longitud
  - Buenos tanto el acceso secuencial como el directo
- **Desventajas**
  - No se conoce inicialmente el tamaño
  - Derroche de espacio (problema de la asignación dinámica → fragmentación externa)
  - Los archivos no pueden crecer, a no ser que se realice **compactación** → ineficiente

Contiguo (y II)

- **Asociación lógica a física**

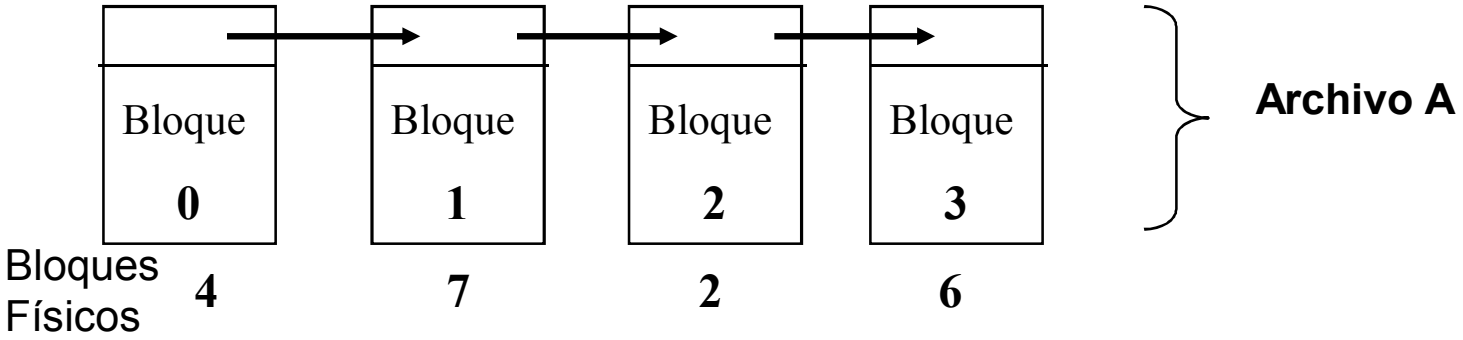
Supongamos que los bloques de disco son de 512 bytes:

Dirección lógica (DL)/512 → C(cociente), R(resto)

- Bloque a acceder = C + dirección de comienzo
- Desplazamiento en el bloque = R

Métodos de Asignación de espacio: No Contiguo - Enlazado

- Cada archivo es una lista enlazada de bloques de disco. Los bloques pueden estar dispersos en el disco



- **Ventajas**
  - Evita la fragmentación externa
  - El archivo puede crecer dinámicamente cuando hay bloques de disco libres → no es necesario compactar
  - Basta almacenar el puntero al primer bloque del archivo

No Contiguo - Enlazado (II)

- **Desventajas**
  - El acceso directo no es efectivo (si el secuencial)
  - Espacio requerido para los punteros de enlace.

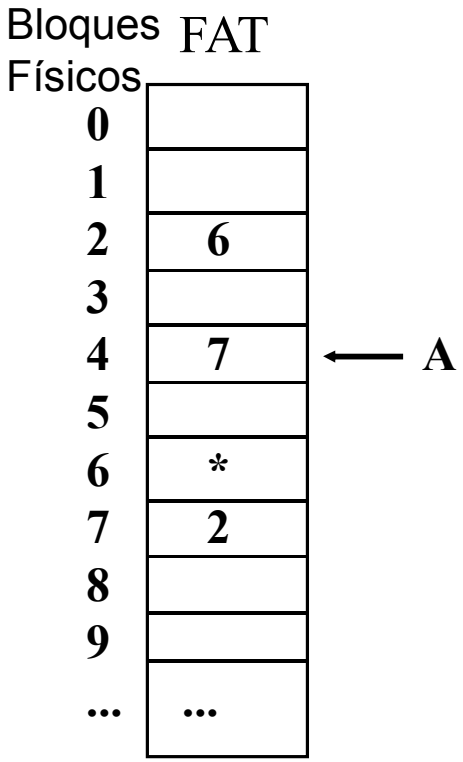
  Solución: agrupaciones de bloques (*clusters*)
  - Seguridad por la pérdida de punteros. Solución: lista doblemente enlazada (*overhead*)
- **Asociación lógica a física** (dirección = 1byte)

  Dirección lógica (DL)/511 → C(cociente), R(resto)

  - Bloque a acceder = C-ésimo
  - Desplazamiento en el bloque = R + 1

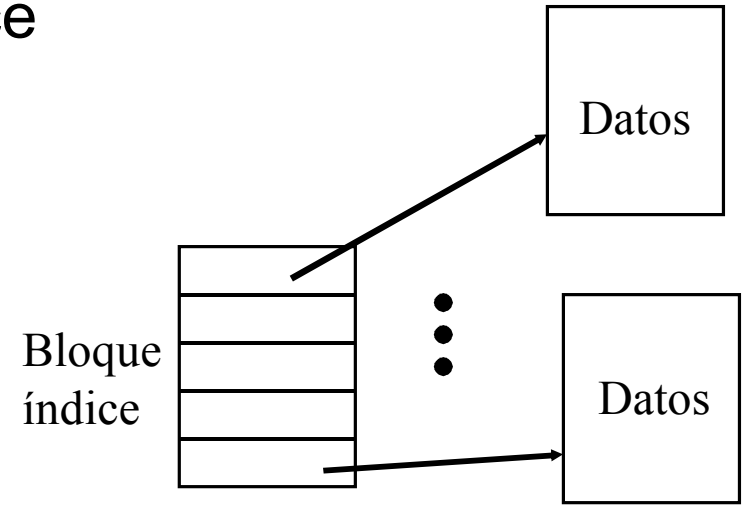
No Contiguo - Enlazado (y III)

- Tabla de Asignación de Archivos (***FAT***): variación del método enlazado (*Windows* y *OS/2*)
  - Reserva una sección del disco al comienzo de la partición para la FAT
  - Contiene una entrada por cada bloque del disco y está indexada por número de bloque de disco
  - Simple y eficiente siempre que esté en caché
  - Para localizar un bloque solo se necesita leer en la FAT → se optimiza el acceso directo
  - Problema: pérdida de punteros → doble copia de la FAT



Métodos de Asignación de espacio: No Contiguo - Indexado

- Todos los punteros a los bloques están juntos en una localización concreta: **bloque índice**
- El directorio tiene la localización a este bloque índice y cada archivo tiene asociado su propio bloque índice
- Para leer el i-*ésimo* bloque buscamos el puntero en la i-*ésima* entrada del bloque índice
- **Ventajas**
  - Buen acceso directo
  - No produce fragmentación externa



No Contiguo - Indexado (II)

- **Desventajas**
  - Posible desperdicio de espacio en los bloques índices
  - Tamaño del bloque índice. Soluciones:

  (a) Bloques índices enlazados

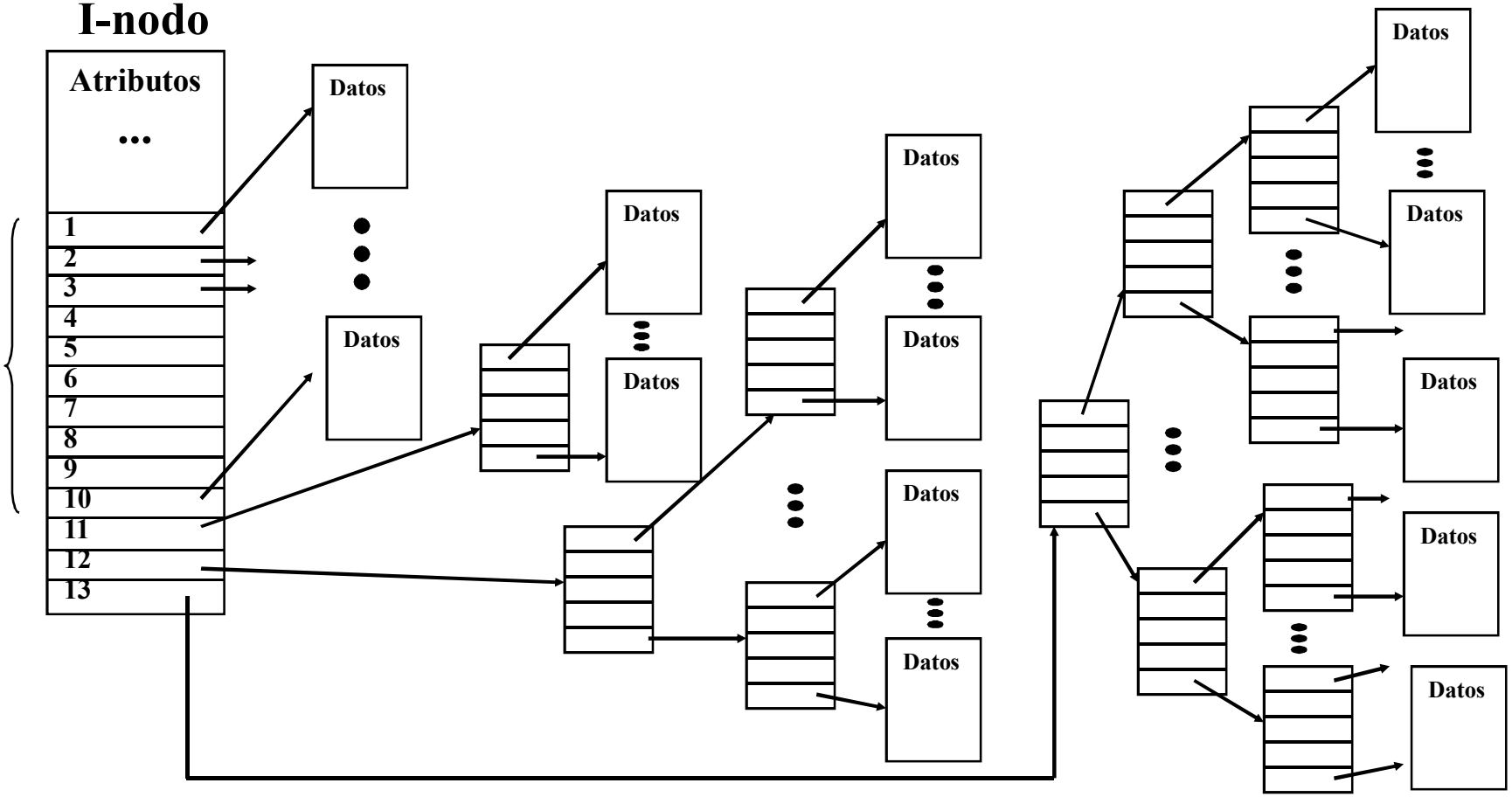
  (b) Bloques índices multinivel

  - Problema: acceso a disco necesario para recuperar la dirección del bloque para cada nivel de indexación
  - Solución: mantener algunos bloques índices en memoria principal

  (c) Esquema combinado (Unix)

No Contiguo - Indexado (y III)

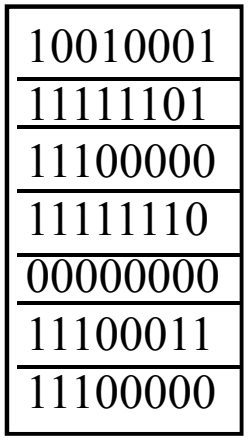
- Unix (**s5fs**)



Gestión de espacio libre

- El sistema mantiene una lista de los bloques que están libres: **lista de espacio libre**
- La **FAT** no necesita ningún método
- A pesar de su nombre, la lista de espacio libre tiene diferentes implementaciones:

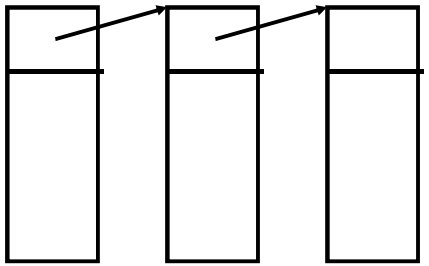
1. **Mapa o Vector de Bits**



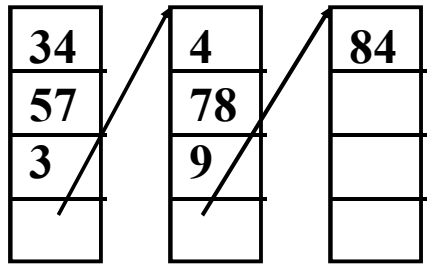
– Cada bloque se representa con un bit (0-Bloque libre; 1-Bloque ocupado)
– Fácil encontrar un bloque libre o *n* bloques libres consecutivos. Algunas máquinas tienen instrucciones específicas
– Fácil tener archivos en bloques contiguos
– Ineficiente si no se mantiene en memoria principal

Gestión de espacio libre (y II)

2. **Lista enlazada**
   - Enlaza todos los bloques libres del disco, guarda un puntero al primer bloque en un lugar concreto
   - No derrocha espacio
   - Relativamente ineficiente → No es normal atravesar bloques vacíos



3. **Lista enlazada con agrupación**
   - Cada bloque de la lista almacena *n-1* direcciones de bloques libres
   - Obtener muchas direcciones de bloques libres es rápido



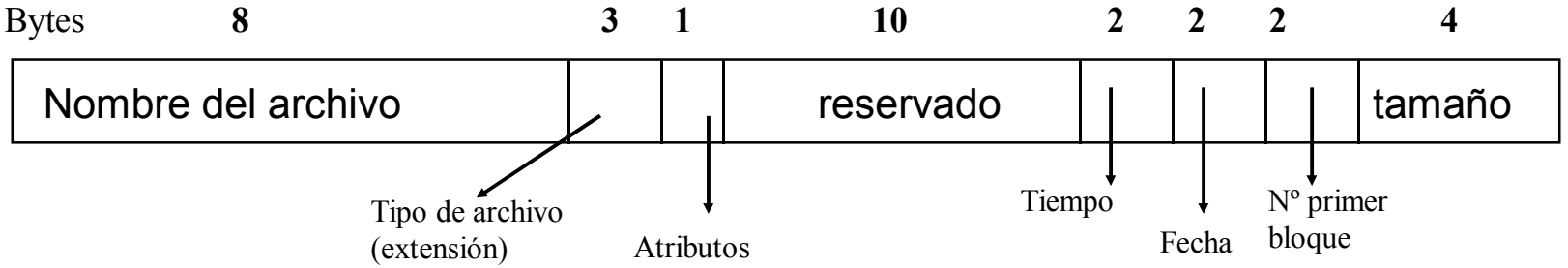
4. **Cuenta**
   - Cada entrada de la lista: una dirección de bloque libre y un contador del nº de bloques libres que le sigue

Implementación de Directorios

- Contenido de una entrada de directorio. **Casos**:

  (a) Nombre de Archivo + Atributos + Dirección de los bloques de datos (DOS)

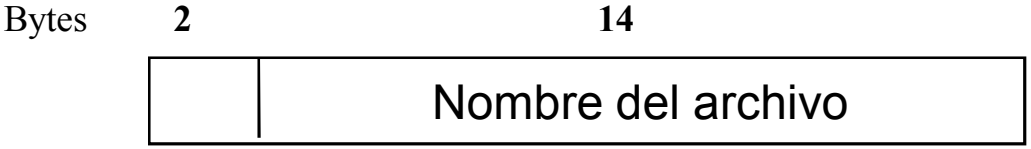
Entrada de directorio de MS-DOS



Implementación de Directorios (y II)

(**b**) Nombre de Archivo + Puntero a una estructura de datos que contiene toda la información relativa al archivo (UNIX)

Entrada de directorio de UNIX (s5fs)



- Cuando se abre un archivo
  - El SO busca en su directorio la entrada correspondiente
  - Extrae sus atributos y la localización de sus bloques de datos y los coloca en una tabla en memoria principal
  - Cualquier referencia posterior usa la información de dicha tabla

Implementación de Directorios (y III)

- Posibilidades respecto a la implementación:

1. **Lista lineal**
   - Sencillo de programar
   - Consume tiempo en las creaciones, búsquedas, ..., si no se utiliza una *caché* software

2. **Tabla *hash***
   - Decrementa el tiempo de búsqueda
   - Dificultades:
     - Tamaño fijo de la Tabla hash
     - Dependencias de la función hash sobre el tamaño de la tabla
     - Necesita previsión para **colisiones**

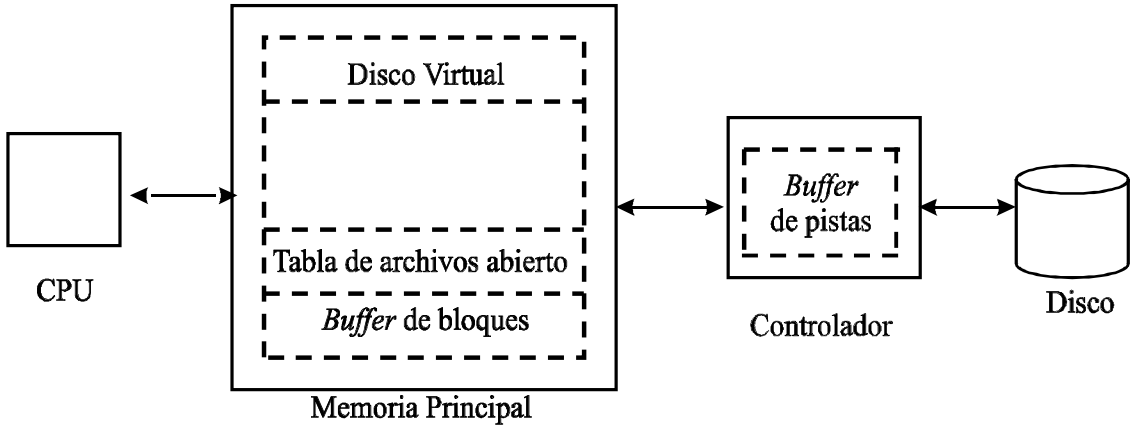
Implementación de Directorios (y IV)

- Implementación de archivos compartidos (o enlace):

1. Enlaces **simbólicos**
   - Se crea una nueva entrada en el directorio, se indica que es de tipo *enlace* y se almacena el camino de acceso absoluto o relativo del archivo al cual se va a enlazar
   - Se puede usar en entornos distribuidos
   - Gran número de accesos a disco
2. Enlaces **absolutos** (o *hard*)
   - Se crea una nueva entrada en el directorio y se copia la dirección de la estructura de datos con la información del archivo
   - Problema al borrar los enlaces: solución → **Contador de enlaces**

Eficiencia y Rendimiento

- Los discos suelen ser el principal cuello de botella del rendimiento del sistema
- La eficiencia depende de la asignación de disco y de la implementación de directorios utilizada
- Para proporcionar mejor rendimiento:



Varias localizaciones de *cache de disco*

1. **Caché de disco**: secciones de M.P. con bloques usados
2. **Discos virtuales** o discos RAM: almacén temporal. Su contenido es controlado por el usuario

Recuperación

- Como los archivos y directorios se mantienen tanto en MP como en disco, el sistema debe asegurar que un fallo no genere pérdida o inconsistencia de datos
- Distintas formas:

1. **Comprobador de consistencia**:
   - Compara los datos de la estructura de directorios con los bloques de datos en disco y trata cualquier inconsistencia
   - Más fácil en listas enlazadas que con bloques índices

2. Usar programas del sistema para realizar **copias de seguridad** (*backup*) de los datos de disco a otros dispositivos y de recuperación de los archivos perdidos

Estructura del Disco

- Desde el punto de vista del SO, el disco se puede ver como un array de bloques (B_0, B_1,...,B_{n-1})
- La información se referencia por una dirección formada por **varias partes**:
  - unidad (número de dispositivo),
  - superficie (o cara),
  - pista, y
  - sector
- Existe un esquema de asociación de la dirección de un bloque lógico B_i a dirección física (pista, sector, ...)
  - El área de asignación más pequeña es un bloque (1 o más sectores)
  - Fragmentación interna en los bloques

Planificación de Disco

- El SO puede mejorar el **tiempo medio de servicio** del disco
- Una petición se atiende en **tres fases**:
  1. **Posicionamiento** de la cabeza en la pista o cilindro
  2. **Latencia**: espera a que pase el bloque deseado
  3. **Transferencia** de los datos entre MP y disco

tº_total de servicio = tºposicionamiento + tº_latencia + tº_transferencia

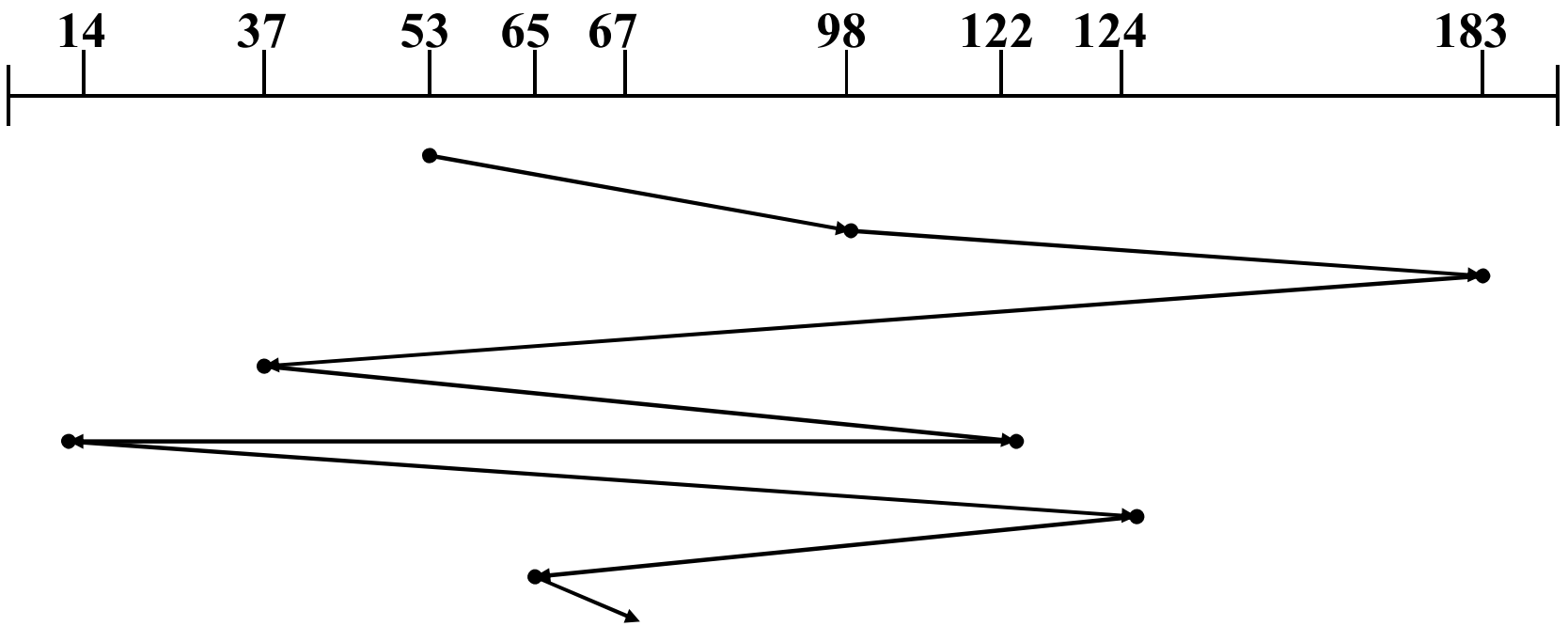
- La planificación intenta minimizar el **tiempo de posicionamiento** ≈ distancia de posicionamiento
- Si el disco está ocupado, las peticiones se encolan

Planificación de Disco (II)

- Información necesaria para una petición:
  - Si la operación es de entrada o de salida
  - Dirección de bloque
  - Dirección de memoria a donde, o desde donde, copiar los datos a transferir
  - Cantidad de información a transferir
- Existen distintos algoritmos de planificación de peticiones
- **Ejemplo**: Cola de peticiones (números de pistas) 98, 183, 37, 122, 14, 124, 65, 67. Inicialmente la cabeza está en la pista 53

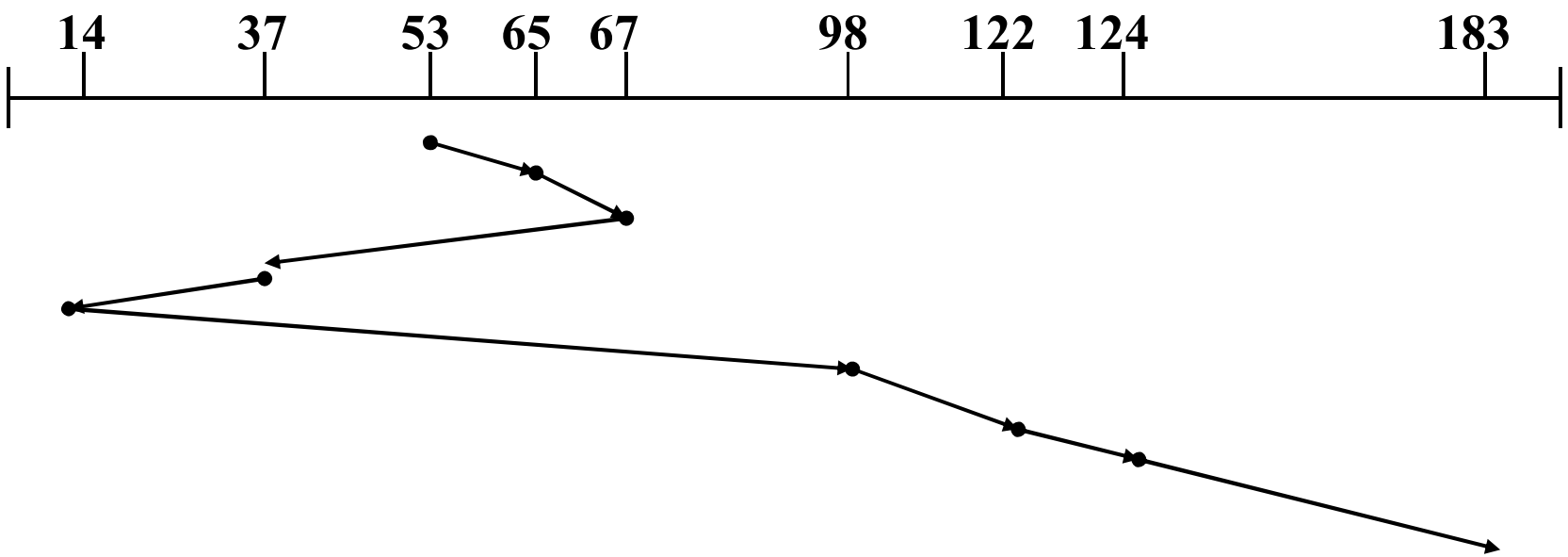
Planificación de Disco (III)

- **FCFS** (*First Come First Served*)



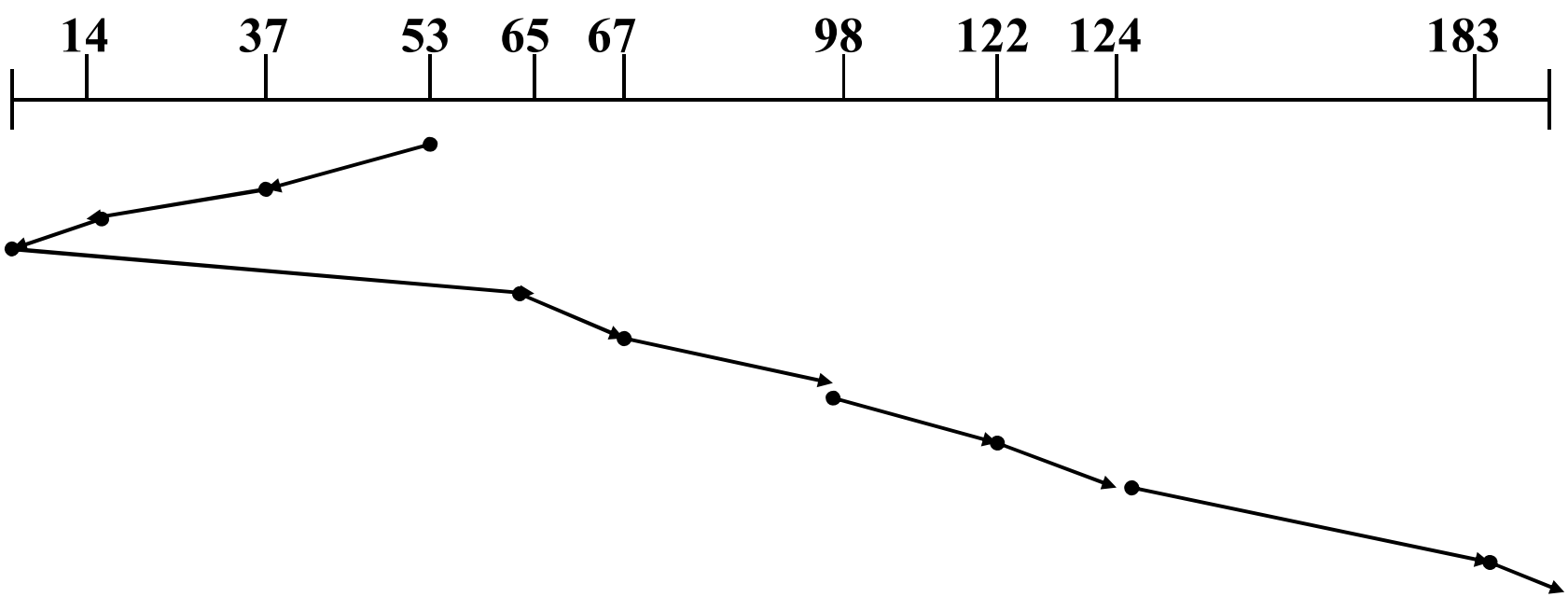
Planificación de Disco (IV)

- **SSTF** (*Shortest Seek Time First*): Primero la petición cuyo tºde posicionamiento sea más corto



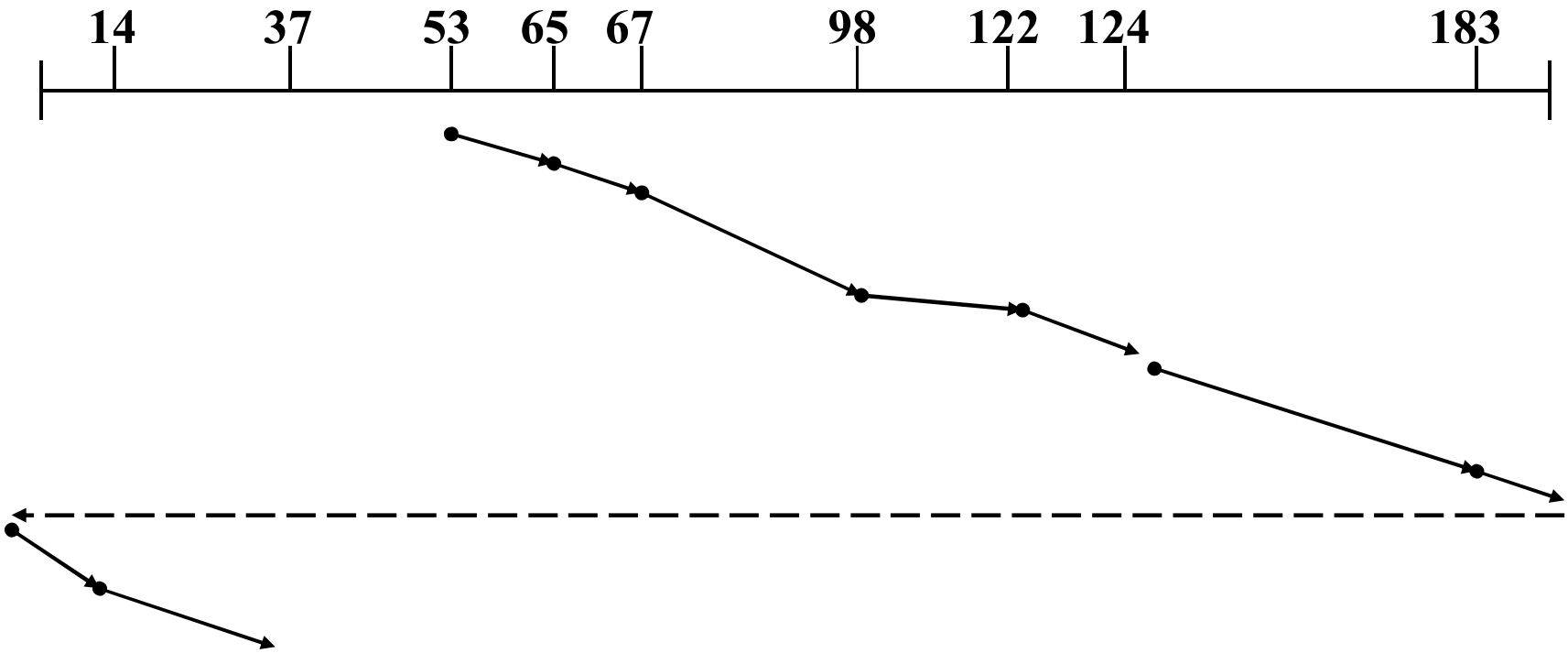
Planificación de Disco (V)

- **SCAN** La cabeza recorre el disco desde el principio al fin y desde el fin al principio sirviendo las peticiones para cada pista



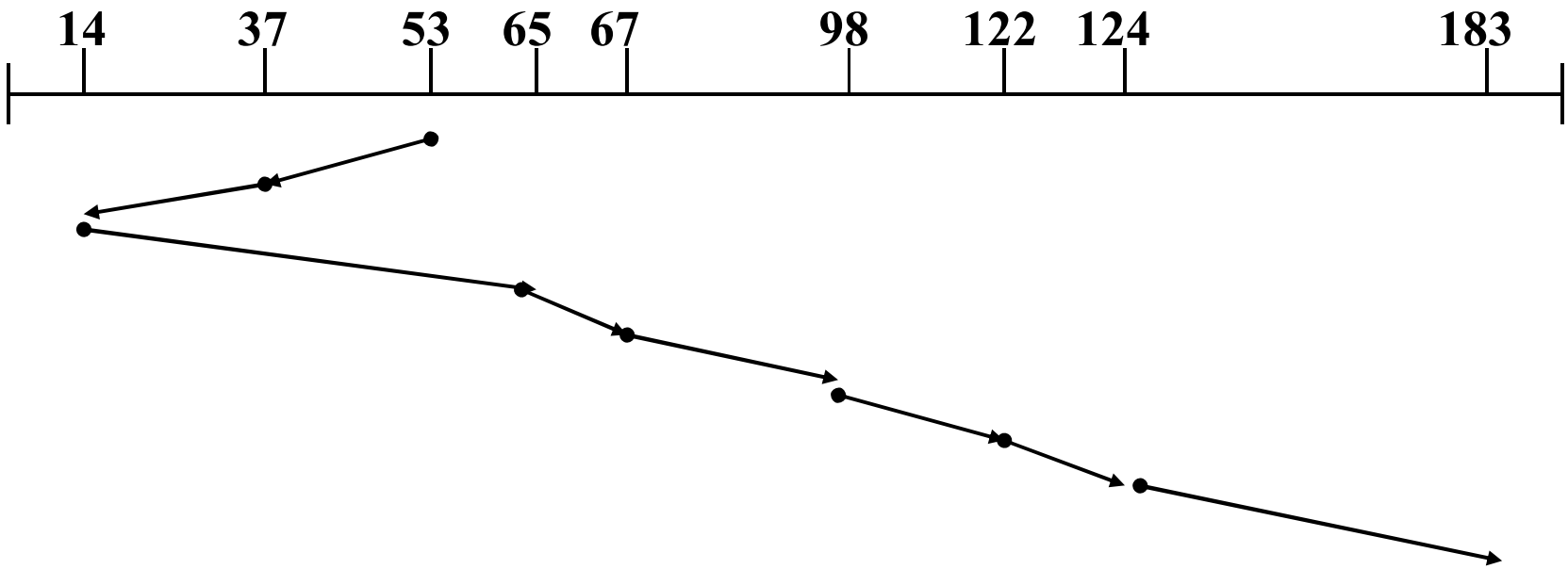
Planificación de Disco (VI)

- **C-SCAN** Supone que la lista de bloques es circular, cuando llega al final, comienza otra vez por el principio



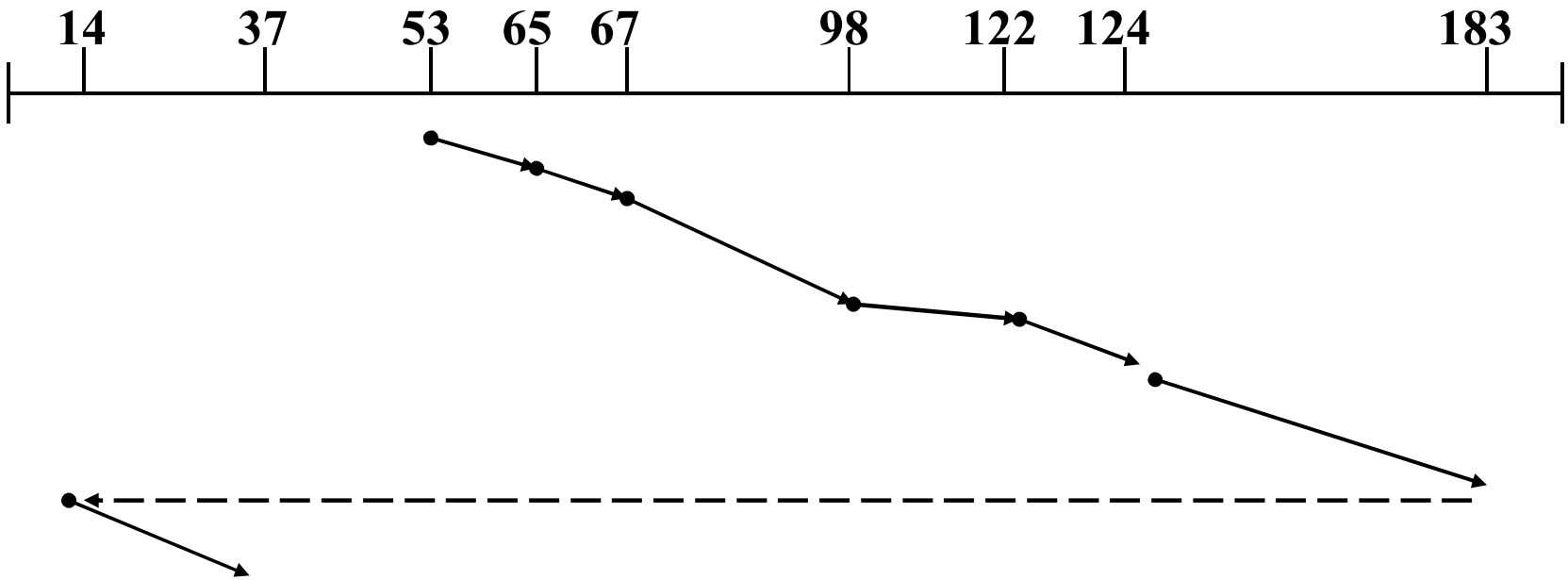
Planificación de Disco (VII)

- **LOOK** Solo se mueve hacia una dirección si hay peticiones pendientes



Planificación de Disco (VIII)

- **C-LOOK**

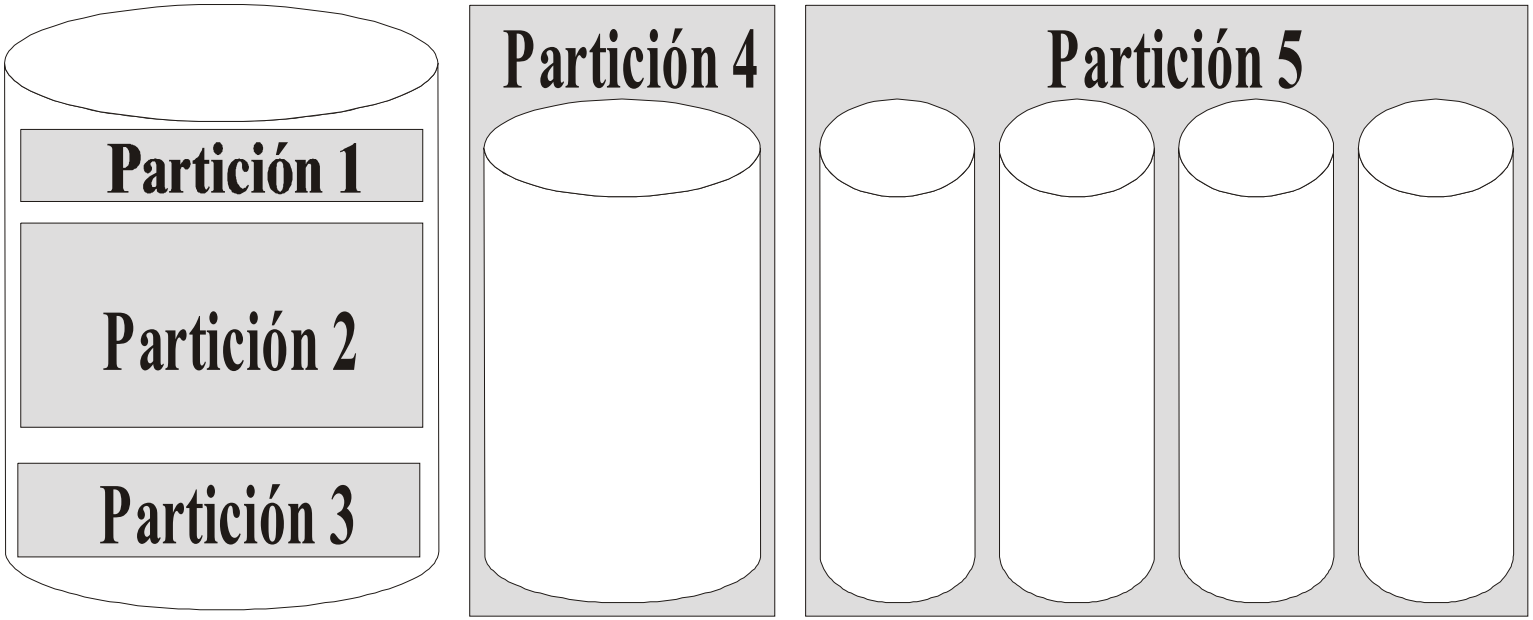


Selección de un algoritmo de planificación

- En general, para cualquier algoritmo, el rendimiento depende mucho del **número** y **tipo** de las peticiones
  - si la cola está prácticamente vacía, cualquier algoritmo es válido
- El servicio de peticiones puede estar muy influenciado por el método de asignación de espacio en disco utilizado
  - con asignación contigua: peticiones que reducen el tiempo de posicionamiento
  - con asignación no contigua (enlazado o indexado): mayor aprovechamiento de disco pero mayor tiempo de posicionamiento

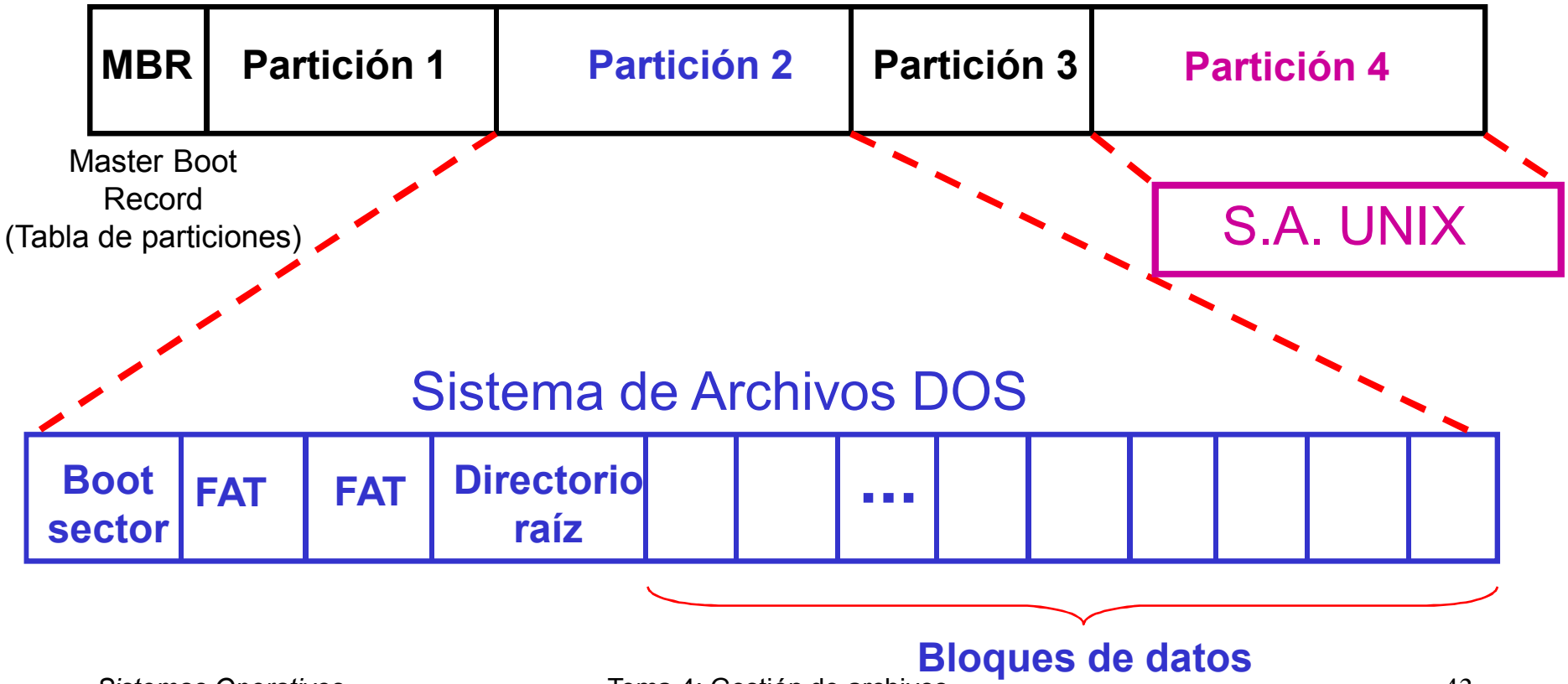
Gestión de disco

- **Partición** del disco



Gestión de disco (y II)

- Ejemplo de organización en particiones



Gestión de disco (y III)

- Formateo del disco:
  - **Físico**: pone los sectores (cabecera y código de corrección de errores) por pista
  - **Lógico**: escribe la información que el SO necesita para conocer y mantener los contenidos del disco (un directorio inicial vacío, FAT, lista de espacio libre, ...)
- Bloque de arranque para **inicializar** el sistema localizado por *bootstrap*
- Métodos necesarios para detectar y manejar bloques dañados

Gestión del Espacio de Intercambio

- La memoria virtual requiere el uso del disco como extensión de la memoria principal
- Su uso depende de los algoritmos de gestión de memoria del SO
  - Intercambio: procesos completos
  - Paginación: páginas de procesos
- La cantidad necesaria depende normalmente de la computadora:*PC, Workstation, ...*
- Algunos SO's permiten el uso de múltiples espacios de intercambio

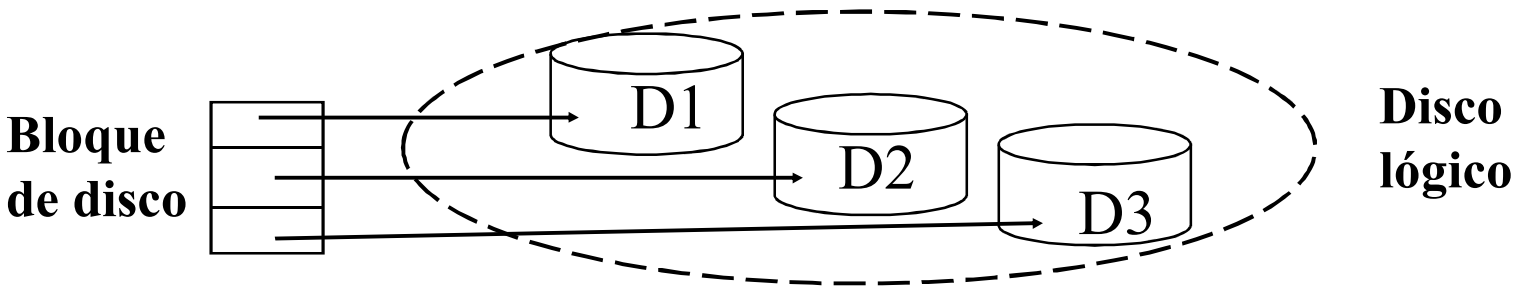
Gestión del Espacio de Intercambio (y II)

- Asignación del espacio de intercambio. Dos posibilidades:
  - En el propio **sistema de archivos** (p.e. Windows)

    → fácil de implementar pero ineficiente
  - En una **partición independiente**:
    - ☑ no utiliza estructura de directorios ni sistema de archivos
    - ☑ utiliza su propios algoritmos → eficiente
    - ☑ Problema: aumentan las necesidades de espacio en disco → fragmentación interna
- Algunos SO's admiten ambos métodos (Solaris2)

Eficiencia y Seguridad de Disco

- Técnica de **entremezclamiento** (*Striping*) de disco
  - ☑ Un grupo de discos se trata como una unidad. Cada bloque está formado por subbloques, y cada uno de éstos se almacenan en un disco diferente
  - ☑ Los discos realizan el posicionamiento y transfieren sus bloques **en paralelo** ⇒ decrementa el tiempo de transferencia del bloque



Eficiencia y Seguridad de Disco (y II)

- **RAID** (*Redundant Array of Independent Disks*)
  - ☑ Mejoras en el rendimiento y la seguridad
  - ☑ Organizaciones*:*
    - **Sombra** (*shawdowing*): mantiene duplicados de cada disco
    - **Paridad entremezclada** de bloques: los datos se escriben en cada disco del array en un bloque y se tiene un bloque extra de paridad en otro disco ⇒ los datos dañados de un disco se obtienen a partir de los datos del resto de discos

Subsistema de archivos Unix

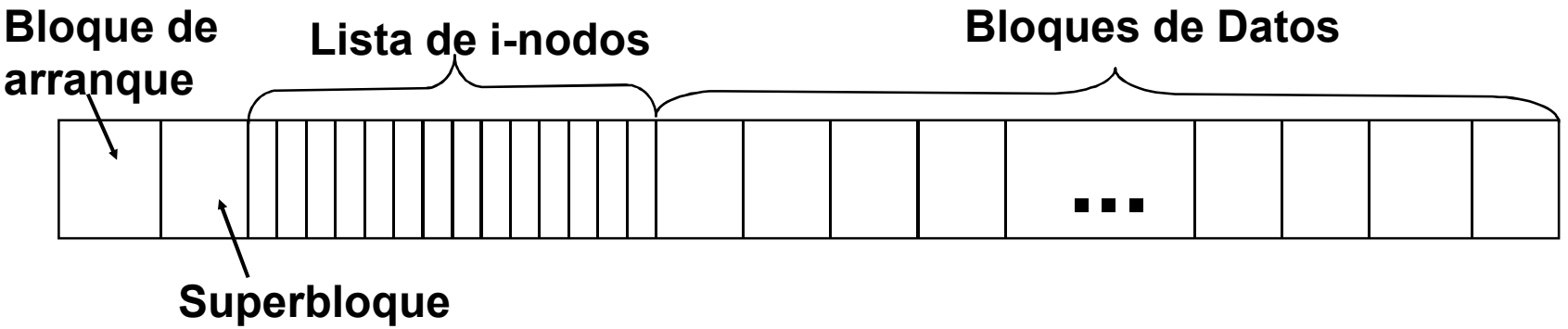
- **i-nodo**: representación interna de un archivo
- Un archivo tiene asociado un único i-nodo, aunque éste puede tener distintos nombres (**enlaces**)
- Si un proceso:
  - Crea un archivo → se le asigna un i-nodo
  - Referencia a un archivo por su nombre → se analizan permisos y se lleva el i-nodo a memoria principal

Introducción al subsistema de archivos (y II)

- Estructuras de datos en memoria principal relacionadas con los archivos:
  - (1) **Tabla de i-nodos**: el núcleo lee del sistema de archivos el i-nodo cuando se opera con él
  - (2) **Tabla de archivos**: mantiene información del puntero de lectura/escritura y los permisos de acceso del proceso al archivo
  - (3) **Tabla de descriptores de archivos**: identifica los archivos abiertos por el proceso
- 1 y 2 son estructuras globales, 3 es una estructura local a cada proceso

Estructura en disco del sistema de archivos

- Un sistema de archivos es una secuencia de bloques lógicos con la siguiente estructura (***s5fs***):
  - **Bloque de arranque**: primer sector, puede contener código de arranque para inicializar el SO
  - **Superbloque**: estado del sistema de archivos
  - **Lista de i-nodos**: tamaño estático especificado en la configuración del sistema de archivos
  - **Bloques de datos**: para archivos y para administración



Contenido de un i-nodo

- **Identificador del propietario** del archivo: UID, GID
- **Tipo de archivo** (regular, directorio, dispositivo, cauce).

Si es 0 ∅ el i-nodo está libre

- **Permisos de acceso**
- **Tiempos de acceso**: última modificación, último

acceso y última vez que se modificó el i-nodo

- **Contador de enlaces**
- Tabla de contenidos para las **direcciones de los datos**

en disco del archivo

- **Tamaño**

Contenido de un i-nodo de la tabla de i-nodos

- El núcleo mantiene en la tabla de i-nodos, además del contenido del i-nodo, los siguientes campos:
  - El **estado** del i-nodo en memoria, indicando si:
    - » el i-nodo está bloqueado
    - » un proceso está esperando a que el i-nodo se desbloquee
    - » la representación en memoria del i-nodo difiere de la copia en disco por cambio en los datos del i-nodo
    - » la representación en memoria del archivo difiere de la copia en disco por cambio en los datos del archivo
    - » el archivo es un punto de montaje

Contenido de un i-nodo de la tabla de i-nodos (y II)

– El **número de dispositivo lógico** del sistema de archivos que contiene al archivo

– El **número de i-nodo**

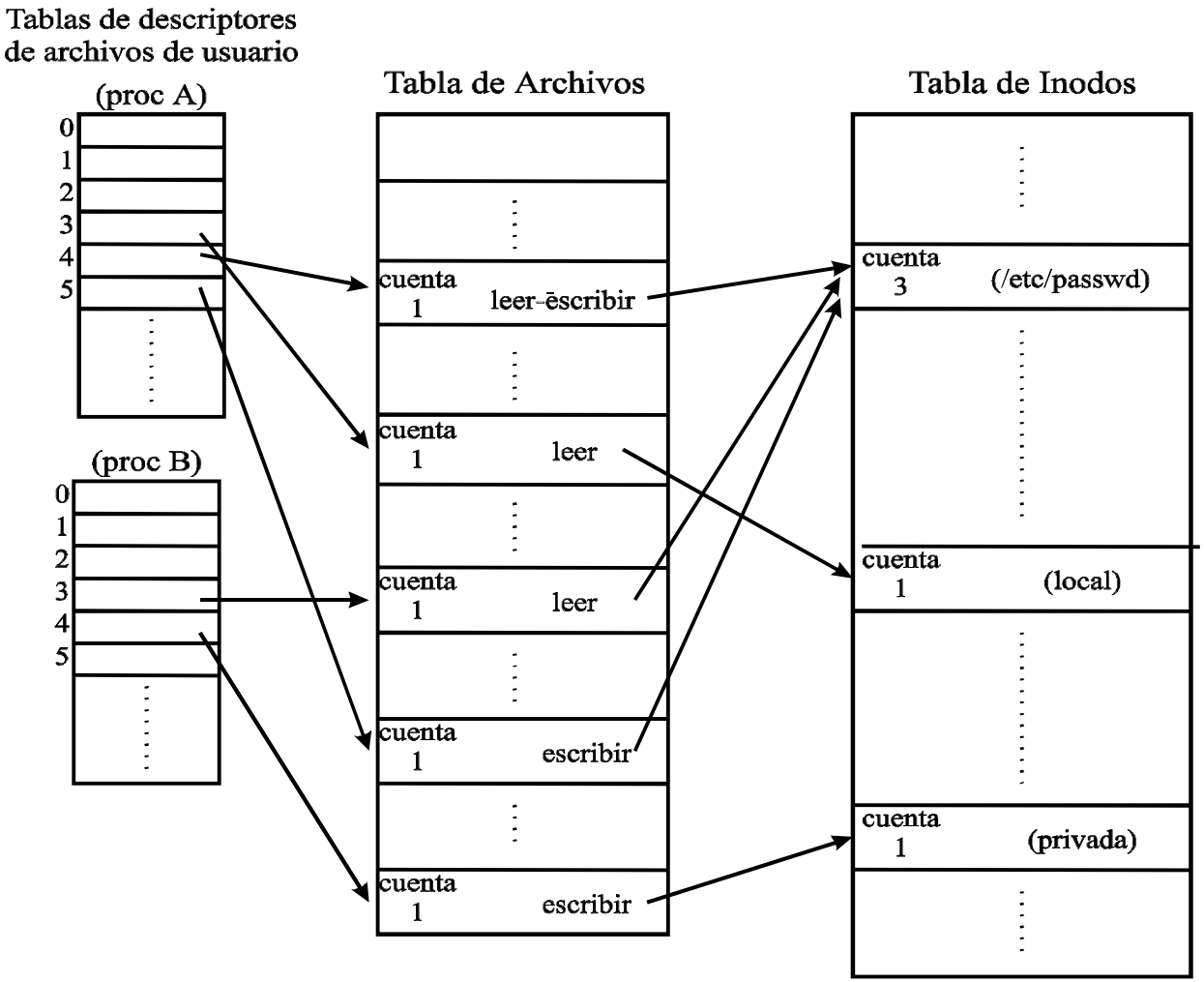
– **Punteros a otros i-nodos** en memoria: se usa una estructura hash para enlazar los i-nodos cuya clave de acceso se basa en el número de dispositivo lógico y el número de i-nodo. Existe una lista de i-nodos libres.

– **Contador de referencias**: número de referencias actuales al i-nodo (p.e. cuando varios procesos abren el mismo archivo). Un i-nodo sólo estará en la lista de i-nodos libres cuando su contador de referencias sea 0.

Tabla de archivos y Tabla de descriptores de archivos

- **Tabla de archivos**, contenido de cada entrada:
  - puntero al i-nodo correspondiente de la tabla de i-nodos
  - puntero de lectura/escritura
  - permisos de acceso
  - modo de apertura del archivo
  - contador de entradas de las tablas de descriptores de archivos asociados con esta entrada
- **Tabla de descriptores de archivos**, contenido de cada entrada:
  - puntero a la entrada de la tabla de archivos correspondiente

Estructuras de datos después de que dos procesos abran archivos



El sistema abre para cada proceso tres archivos por defecto (primeras entradas de la tabla de descriptores de archivos):

- **stdin** (teclado)
- **stdout** y **stderr** (pantalla)

Algoritmos a bajo nivel del S.A.

– **iget** = devuelve un i-nodo de la tabla de i-nodos identificado previamente, si no está, lo carga en la tabla
– **iput** = libera un i-nodo de la tabla de i-nodos
– **namei** = convierte un *pathname* a un número de i-nodo
– **alloc** y **free** = asignan y liberan bloques de disco
– **ialloc** e **ifree** = asignan y liberan i-nodos de disco
– **bmap** = traducción de direcciones lógicas a físicas

Algoritmo iget: acceso a i-nodo en memoria

Entrada: número de i-nodo de un SA
Salida: i-nodo bloqueado

```
while (no hecho) {
        if (i-nodo en tabla de i-nodos) {
                if (i-nodo bloqueado) {
                        bloquear (hasta i-nodo desbloqueado);
                        continue;  /* vuelve al while */
                }
                bloquear (i-nodo);
                if (i-nodo en la lista i-nodos libres)
                        eliminarlo de la lista de libres;
                contador_referencias ++;
                return (i-nodo);
        }
```

Algoritmo iget: acceso a i-nodos en memoria (y II)

```
        else /*i-nodo no está en la tabla de i-nodos*/ {
                if (no hay i-nodos en lista i-nodos libres)
                        return (error);
                liberar i-nodo de la lista i-nodos libres;
                actualizar nº i-nodo y Sistema de Archivos;
                colocar el i-nodo en la entrada correcta de
                        la tabla hash;
                leer i-nodo de disco; /* bread */
                inicializar i-nodo; /*contador_referencias=1*/
                return (i-nodo);
        }
} /* fin de iget */
```

Algoritmo iput: liberación de i-nodos en memoria

Entrada: puntero al i-nodo en memoria (tabla de i-nodos)
Salida: nada

```
{
bloquear (i-nodo);    /*si no lo está*/
  contador_referencias --;
  if (contador_referencias == 0)  {
       if (contador_enlaces == 0) {
            liberar bloques de disco del archivo;  /* free */
            poner tipo de archivo a 0;   /* i-nodo libre */
            liberar (i-nodo);  /* ifree */   }
       else {
            if (accedido(archivo) || cambiado(i-nodo) ||
                    cambiado(archivo) )
                actualizar i-nodo de disco;
       poner i-nodo en la lista de i-nodos libres;
       }
  }
desbloquear ( i-nodo);  } /* fin de iput */
```

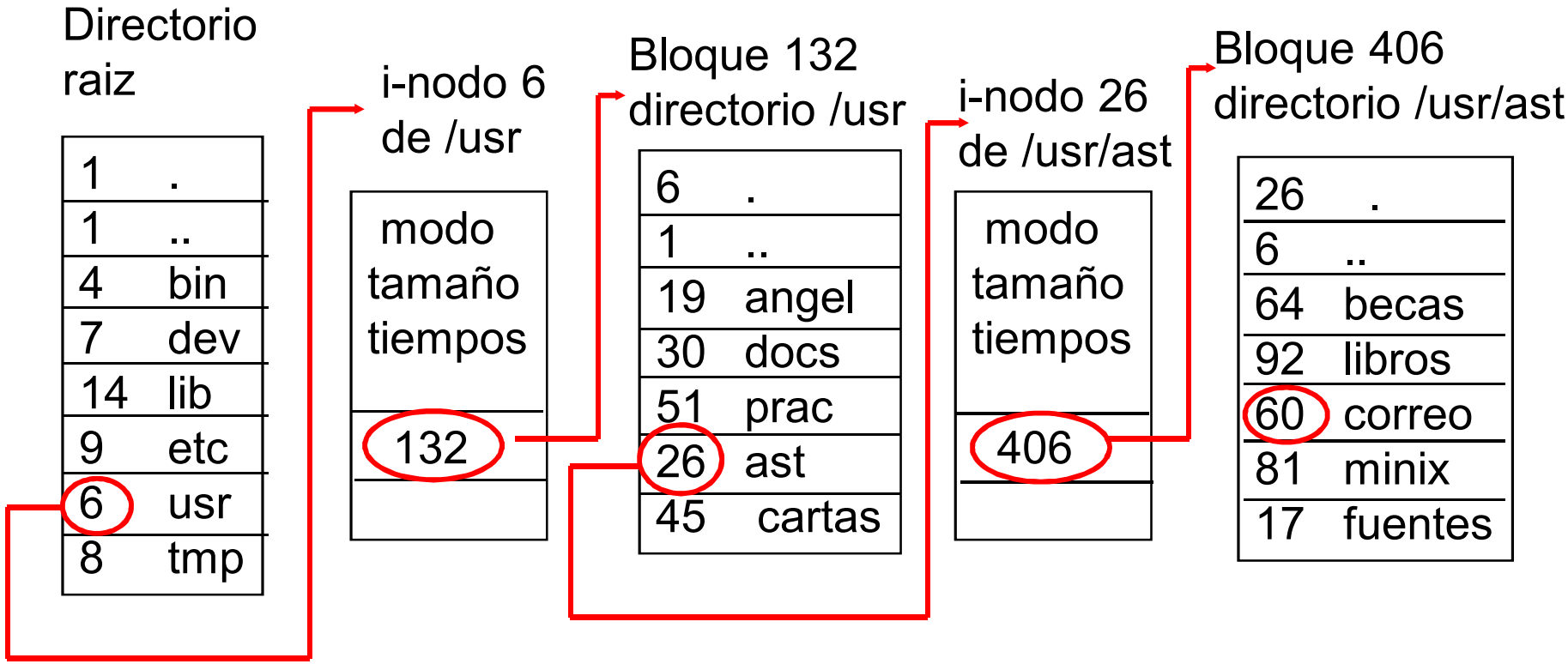
Archivos regulares y directorios

- **Archivo regular**
  - secuencia de bytes, comenzando en el byte 0
  - los procesos acceden a los datos mediante un desplazamiento (**offset**)
  - los archivos pueden contener **agujeros** (lseek, write)
- **Directorios**
  - diferencias entre BSD y SVR4 solventadas utilizando una biblioteca estándar de manejo de directorios
  - un directorio es un archivo cuyos datos son secuencias de entradas. Dos entradas especiales son **.** y **..**
  - entradas de directorios vacías ⇒ número i-nodo = 0
  - cada proceso tiene asociado un directorio actual
  - el i-nodo del directorio raíz se almacena en una variable global

Ejemplo de namei

- Supongamos el nombre de ruta */usr/ast/correo,*

¿cómo se convierte en un nº de i-nodo?



Contenido del superbloque

- Tamaño del sistema de archivos
- Número de bloques libres en el sistema de archivos
- Una lista de bloques libres disponibles y un índice al siguiente bloque libre de la lista de bloques libres
- Tamaño de la lista de i-nodos
- Número de i-nodos libres en el sistema
- Una lista de i-nodos libres y un índice al siguiente i-nodo libre de la lista de i-nodos libres
- Además, cuando está en memoria tiene:
  - Campos de bloqueos para las listas de bloques e i-nodos libres
  - Un campo indicando que el superbloque ha sido modificado

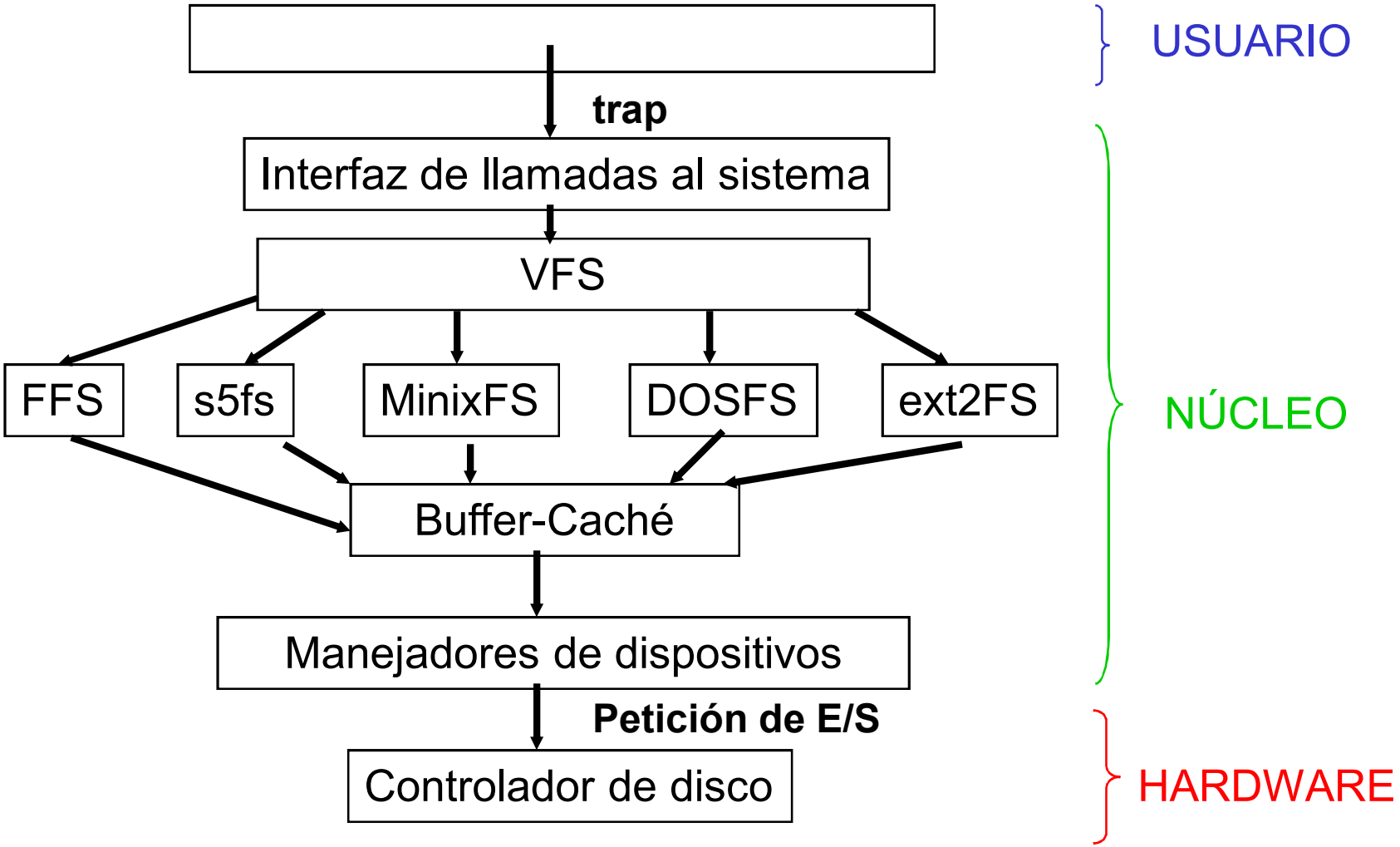
Problemas de s5fs

- Sólo una copia del superbloque → **menor fiabilidad**
- Los i-nodos se asignan aleatoriamente → **varios accesos** para archivos en un mismo directorio
- El i-nodo no se asocia de ninguna forma a sus bloques de datos
- La asignación de bloques de datos no es óptima, salvo inicialmente (después de crear el S.A.)
- Tamaño de bloque (de 1 a 4KB)
- Límite en número de i-nodos (2 bytes) y en nombre de archivos (14 caracteres)

ufs - unix file system

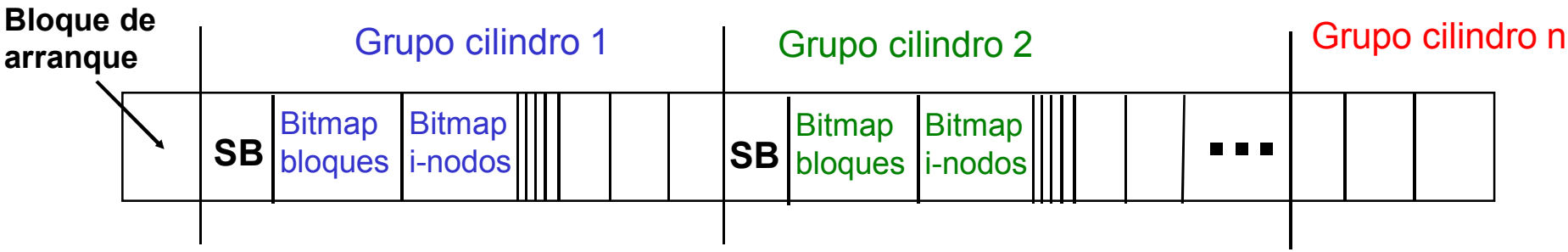
- **ufs** = interfaz vnode/vfs (virtual node/ virtual file system) + implementación FFS (Fast File System)
- **Objetivos**:
  - Soporte para distintos tipos de sistemas de archivos simultáneamente
  - Diferentes particiones con diferentes sistemas de archivos
  - Transparencia en el acceso a un sistema de archivos remoto
  - Añadir nuevos tipos de sistemas de archivos de forma modular

vnode/vfs


trap
Interfaz de llamadas al sistema
VFS
FFS
s5fs
MinixFS
DOSFS
ext2FS
Buffer-Caché
Manejadores de dispositivos
Petición de E/S
Controlador de disco
USUARIO
NÚCLEO
HARDWARE

FFS - Fast File System

- Partición dividida en uno o más grupos consecutivos de cilindros
- La información del superbloque tradicional se divide en:
  - Superbloque FFS: número, tamaño y localización de grupos de cilindros, tamaño bloque, nº total de i-nodos y bloques
  - Cada grupo de cilindros tiene su propia lista de i-nodos libres y de bloques libres



FFS (y II)

- Cada grupo de cilindros contiene una copia duplicada del superbloque
- El tamaño mínimo del bloque es **4KB** ↯ no se necesita tercer nivel de indexación en el i-nodo
- El bloque se puede dividir en fragmentos direccionables y asignables ↯ lista de bloques libres mediante mapa de bits
- Sólo los bloques directos del i-nodo pueden contener fragmentos
- Entradas variables en directorios y enlaces simbólicos

FFS (y III)

- **Políticas de asignación**:
  - Intenta situar i-nodos de archivos de un único directorio en el mismo grupo de cilindros
  - Crea un directorio en un grupo diferente al de su padre ☹ distribuye datos uniformemente
  - Intenta asignar bloques de datos en el mismo grupo que su i-nodo correspondiente
  - Cambia de grupo cuando se alcancen ocupaciones preestablecidas

Llamadas al Sistema

Devuelven un descriptor	Usan namei	Asignan y liberan i-nodos	Atributos de archivo	E/S archivos	Estructura S.Archivos	Manipulan árbol S.A.
open creat dup pipe close	open creat chdir chroot chown chmod stat link unlink mknod mount umount	creat mknod link unlink symlink	chown chmod stat fstat	read write lseek	mount umount	chdir chown

Llamada al sistema **open**

- Sintaxis:

```
da = open (<camino>, <opciones>, [<permisos>])
```

opciones: modo de apertura (O_RDONLY, O_WRONLY, O_RDWR, O_APPEND, O_CREAT, ...)

- Pasos:
  - Convierte el camino en número de i-nodo (**namei**)
  - Si (archivo no existe || no permisos) return (error)
  - Asigna una entrada en la tabla de archivos para este i-nodo e inicializa el contador y offset (por defecto, 0)
  - Asigna una entrada en la tabla de descriptores de archivos y la enlaza con la entrada de la tabla de archivos anterior
  - Desbloquear i-nodo
  - Devolver ***da*** (descriptor de archivo)

Cauces (*pipes*)

- Permiten la transferencia de datos y la sincronización entre procesos. Capacidad limitada y gestión FIFO
- En la última implementación son bidireccionales y soportan comunicación *full-duplex*

sin nombre

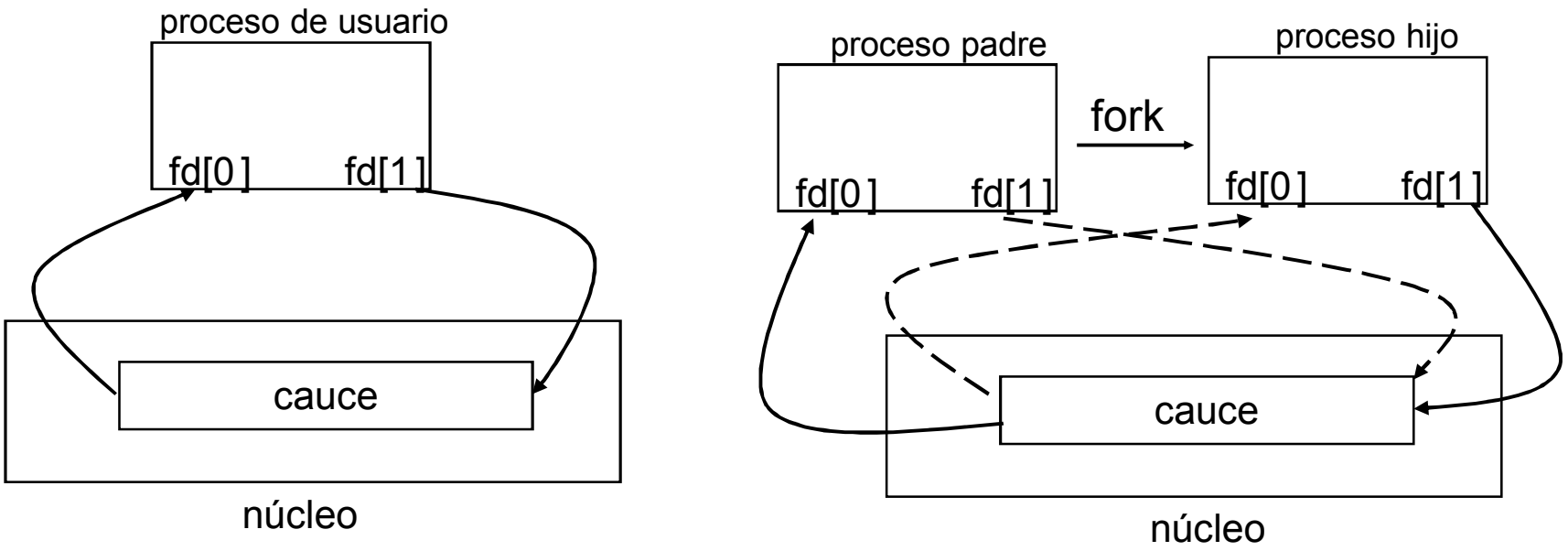
- se crean con **pipe**
- sólo el proceso que lo crea y sus descendientes pueden compartirlo
- son transitorios
- no tienen asociado un nombre de archivo
- tienen asociado un i-nodo en la Tabla de i-nodos

con nombre

- se crean con **mknod**
- todos los procesos con privilegios tienen acceso a él
- no son transitorios, ocupan permanentemente una entrada en un directorio
- tienen asociado un nombre de archivo
- tienen asociado un i-nodo en la Tabla de i-nodos y en disco

Cauces (y II)

int pipe (int fildes[2]) devuelve 0, si éxito, -1 si falla



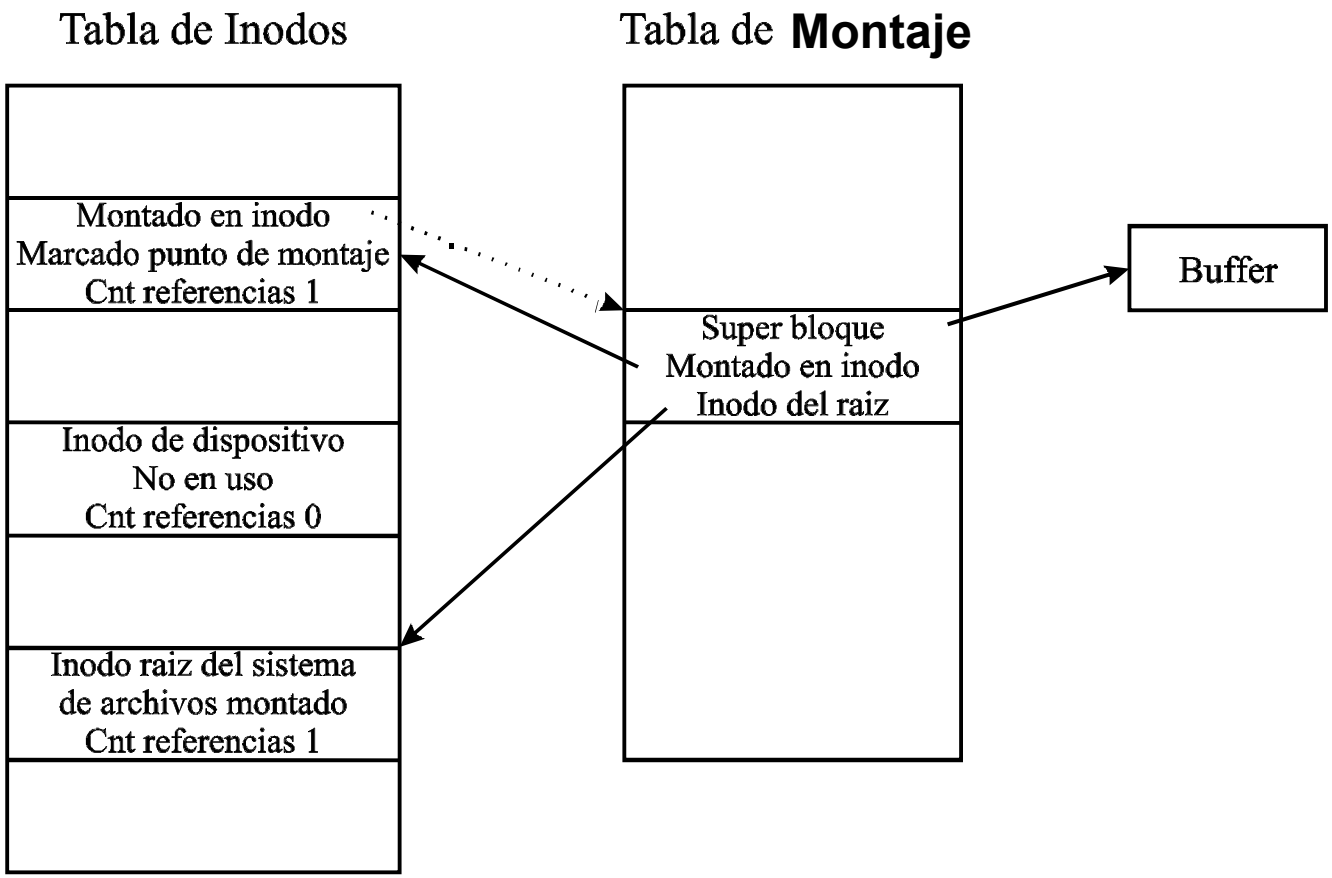
Montaje y desmontaje de un sistema de archivos

- La llamada al sistema **mount** conecta un sistema de archivos al sistema de archivos existente y la llamada **umount** lo desconecta

```
mount (<camino_especial>, <camino_directorio>, <opciones>)
```

- El núcleo tiene una tabla de montaje con una entrada por cada sistema de archivos montado:
  - número de dispositivo que identifica el SA montado
  - puntero a un buffer que contiene una copia del superbloque
  - puntero al i-nodo raíz del SA montado
  - puntero al i-nodo del directorio punto de montaje

Montaje y desmontaje de un sistema de archivos (y II)



Estructuras de Datos después de Montar

Buffer Caché

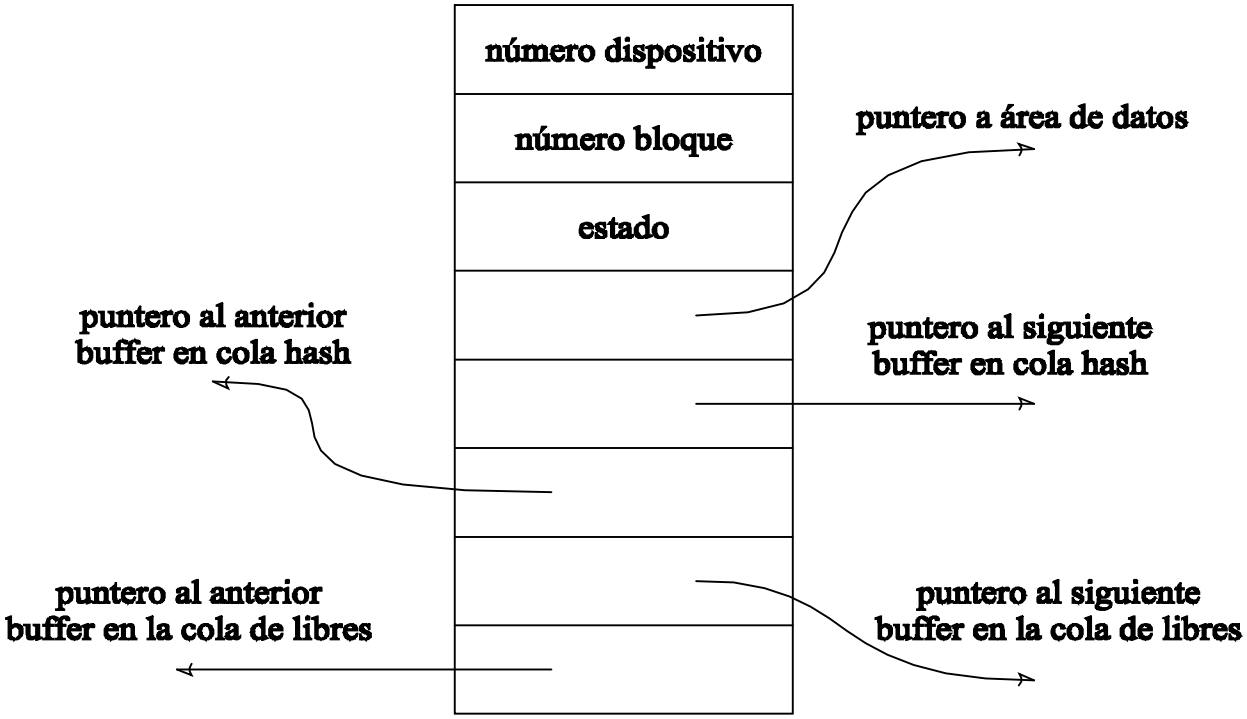
- Limita los accesos a disco aumentando la eficiencia
- Mantiene en memoria principal un depósito de datos interno que contiene los datos de los bloques de disco recientemente usados
- Cuando se leen datos del disco, el núcleo tiende a leerlos de la buffer caché (igual para escritura):
  - Si los datos están, no se tiene que acceder a disco
  - Si los datos no están se accede a disco y se introducen en la caché
- El tamaño de la buffer caché se puede determinar durante la inicialización del sistema
- En las implementaciones actuales la buffer caché sólo se utiliza para los metadatos

Partes de un buffer

1. **Copia de un bloque lógico de disco**. Un bloque de disco no se puede encontrar en mas de un buffer.
2. **Cabecera del buffer** que lo identifica. Contenido:
   - Número de dispositivo
   - Número de bloque
   - Puntero a la copia del bloque de disco
   - Estado, combinación de las siguientes condiciones:
     - » bloqueado (ocupado)
     - » contiene datos válidos
     - » **escritura retardada** (el núcleo debe escribir el contenido del buffer antes de reasignarlo)
     - » se está leyendo/escribiendo el contenido del buffer
     - » un proceso está esperando a que el buffer se libere

Partes de un buffer (y II)

- Punteros utilizados por los algoritmos de asignación para mantener la estructura del depósito de bufferes.



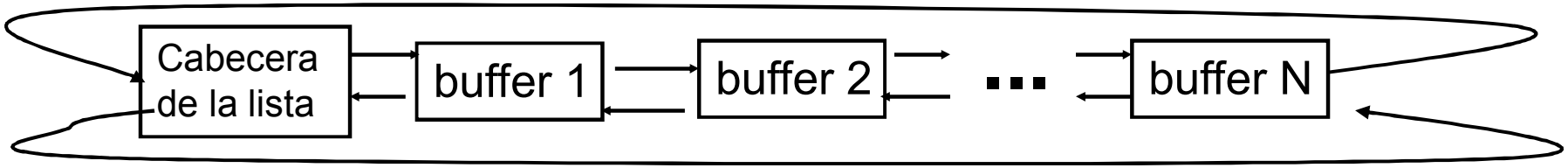
Cabecera del *Buffer*

Estructura de la buffer caché

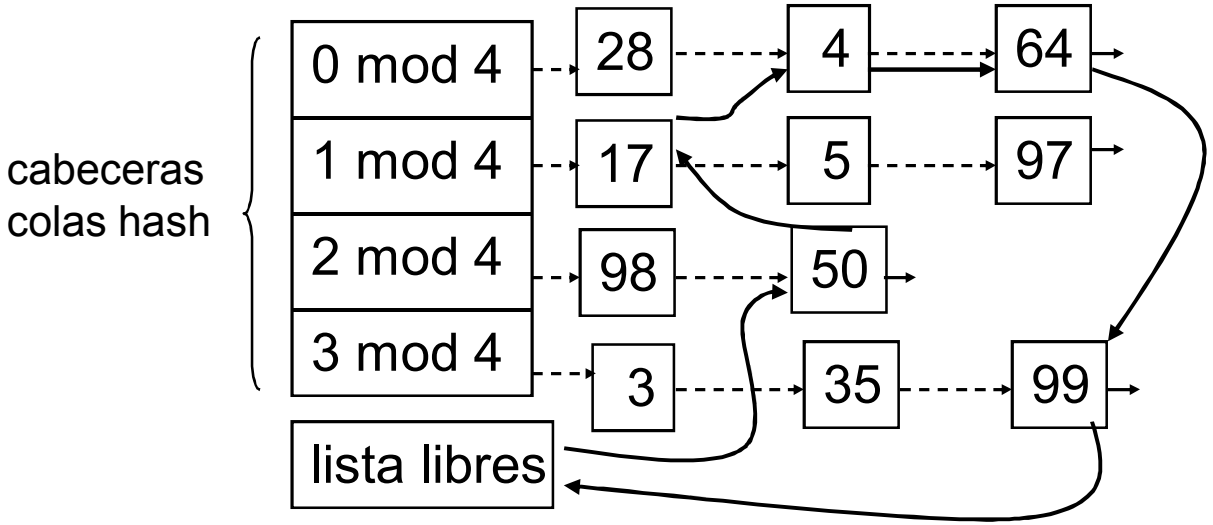
- Se gestiona mediante el algoritmo **LRU**
- **Organización**:
  - lista circular doblemente enlazada de bufferes libres
  - varias colas separadas (listas doblemente enlazadas). Utiliza una función *hash* basada en **nºdispositivo + nºbloque**
- El nº de elementos en una cola hash varía durante la vida del sistema
- Cada buffer está en una única cola hash aunque también puede estar en la lista de bufferes libres
- La posición del buffer dentro de la cola hash no es relevante

Estructura de la buffer caché (y II)

Inicialmente, todos los bufferes están libres



Después de un tiempo ...



- son **listas doblemente enlazadas**
- en el ejemplo sólo usamos el nº de bloque por simplicidad

Algoritmos a bajo nivel para la buffer caché

- **getblk** ↯ lo usan los algoritmos de lectura/escritura para asignar bufferes del depósito
- **brelse** ↯ libera un buffer
- **bread** ↯ lee un bloque de disco (usa getblk)
- **breada** ↯ lecturas asíncronas adelantadas de bloques de disco (usa getblk, brelse)
- **bwrite** ↯ escribe los contenidos de un buffer a un bloque de disco (usa brelse)

Escenarios para recuperar un buffer

1. El núcleo encuentra el bloque en su cola hash y su buffer está en la lista de libres

- marca el buffer como ocupado
- saca el buffer de la lista de libres

2. El núcleo no encuentra el bloque en su cola hash y hay bufferes en la lista de libres

- ¿primer buffer de la lista de libres marcado como escritura retardada?

  no → lo marca como ocupado, lo saca de la cola hash actual y lo inserta en la cola hash apropiada

  si → inicia su escritura asíncrona (cuando termine volverá a la cabeza de libres) y busca otro en la lista libres

Escenarios para recuperar un buffer (y II)

3. El núcleo no encuentra el bloque en la cola hash y la lista de libres está vacía

- bloquear al proceso en espera de un buffer libre

4. El núcleo encuentra el bloque en la cola hash pero está ocupado

- bloquear al proceso hasta que se libere
- **!cuidado!** cuando el proceso se desbloquee debe comenzar la comprobación desde el principio

Ventajas y desventajas

- **Ventajas**
  - acceso a disco uniforme, código más modular
  - reducción del trafico de disco, incrementando el rendimiento y decrementando el tº de respuesta
  - los algoritmos ayudan a asegurar la **integridad del sistema** ↯ secuencializan el acceso a los datos
- **Desventajas**
  - vulnerabilidad ante caídas del sistema
  - copia adicional de los datos en el proceso de usuario y en la caché. Solución en sistemas actuales ↯ **archivos mapeados**

Características de NTFS

- Permite nombres de archivo de hasta 255 caracteres
- Permite la gestión de medios de almacenamiento extraordinariamente grandes
- Incorpora mecanismos para garantizar la seguridad y la fiabilidad (redundancia de datos, usa RAID 1 y 5)
- Soporta el concepto de enlace (por compatibilidad con el estándar POSIX) y un sistema de cuotas
- Es capaz de recuperarse rápidamente después de una caída del sistema o de un fallo del disco
- Soporta el estándar *Unicode*
- Usa caché de disco con escritura retardada

Atributos de los archivos

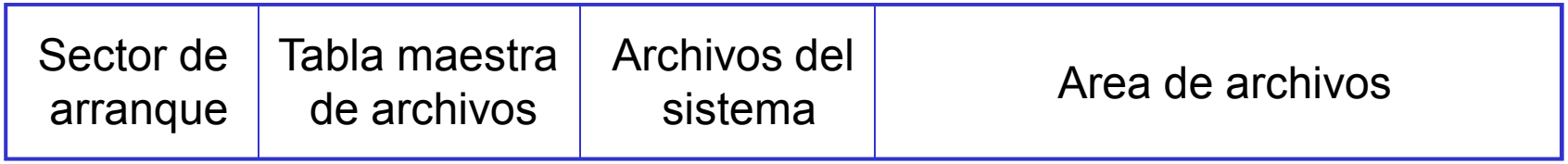
- Nombre
- Fecha de creación, último acceso y última modificación
- Número de serie del volumen
- Tamaño del archivo (64 bits)
- Número de enlaces (compatibilidad con Unix)
- Identificador único que el Ejecutivo asocia a un archivo en el momento en el que alguna hebra lo abra
- Permisos

Permisos sobre los archivos

- Lectura (R)
- Escritura (W)
- Ejecución (X)
- Borrado (D)
- Cambio de permisos (P)
- Ser nuevo propietario (O)
  - El administrador del sistema puede tomar la propiedad de cualquier archivo pero no transferirla de nuevo a ningún usuario

Estructura NTFS

- Componentes de un volumen NTFS:
  - Sector de arranque
  - Tabla maestra de archivos (MFT): contiene información de todos los archivos y directorios y del espacio disponible
  - Archivos del sistema (1 MBytes):
    - MFT2: copia de las tres primeras filas de MFT (garantiza el acceso a la MFT en caso de fallo de sector)
    - Archivo de registro: transacciones realizadas
    - Mapa de bits
    - Tabla de definición de atributos
  - Área de archivos



MFT *(Master File Table)*