Tema 1: Implementación del sistema de archivos

1. Introducción
2. Implementación
3. Estructura del almacenamiento secundario

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

Tema 1: Implementación del sistema de archivos

1. Introducción
2. Implementación
3. Estructura del almacenamiento secundario

1. Introducción: 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)

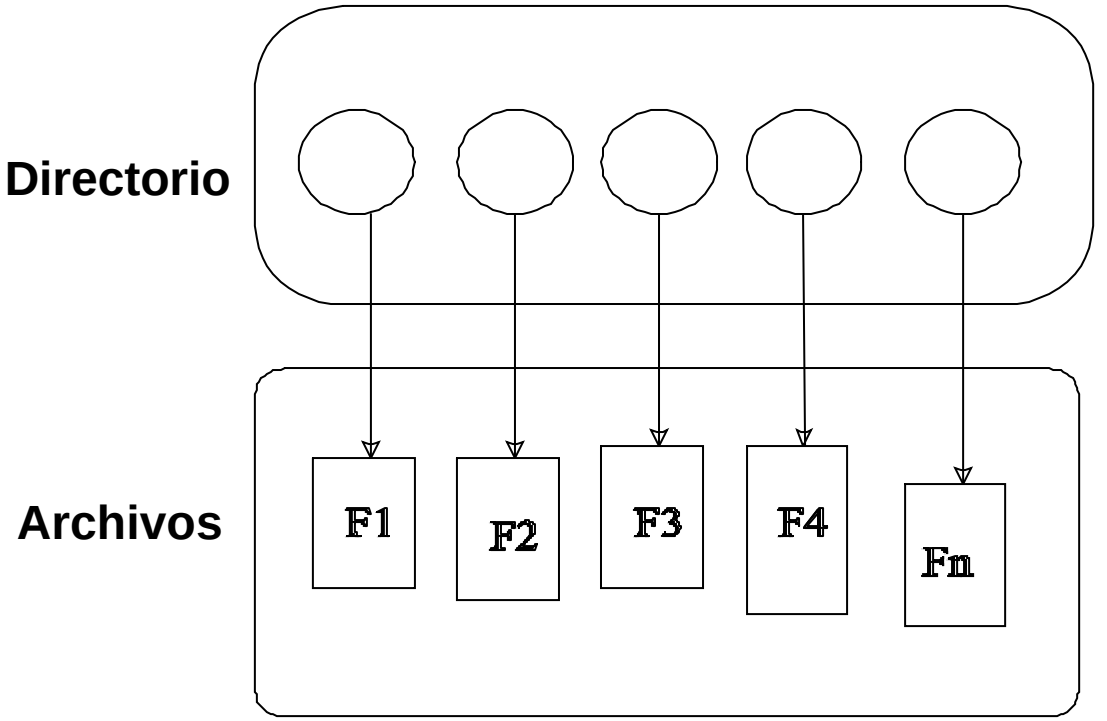
- Atributos (metadatos)
  - **Nombre**: única información en formato legible
  - **Tipo**: cuando el sistema soporte diferentes tipos
  - **Localización**: información sobre su localización en el dispositivo
  - **Tamaño**: tamaño actual del archivo
  - **Protección**: controla quién puede leer, escribir 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)

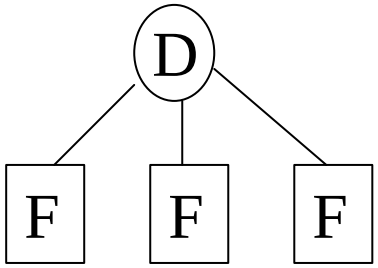
La organización (lógica) de los directorios debe proporcionar:

- **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)

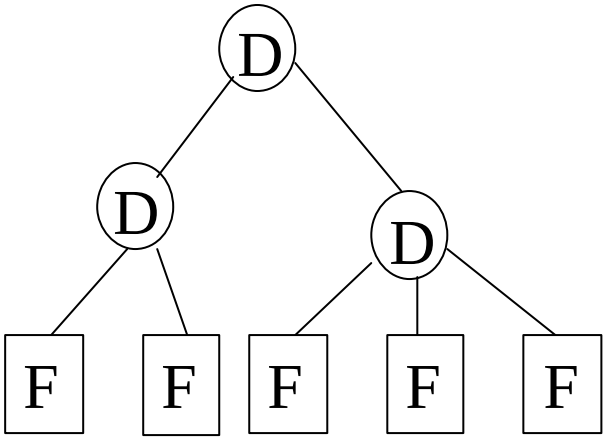
Estructura de Directorios (y III)

Casos:

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

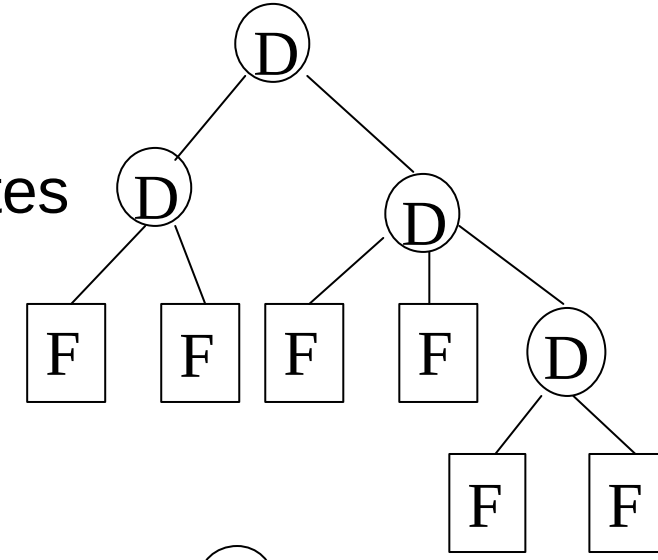


- **A dos niveles**
  - Nombres de camino
  - Diferentes usuarios pueden tener archivos con igual nombre
  - No hay posibilidad de agrupación

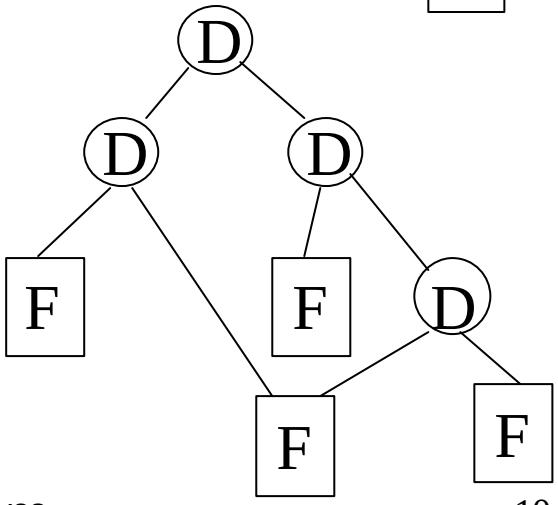


Estructura de Directorios (y IV)

- **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

Tema 1: Implementación del sistema de archivos

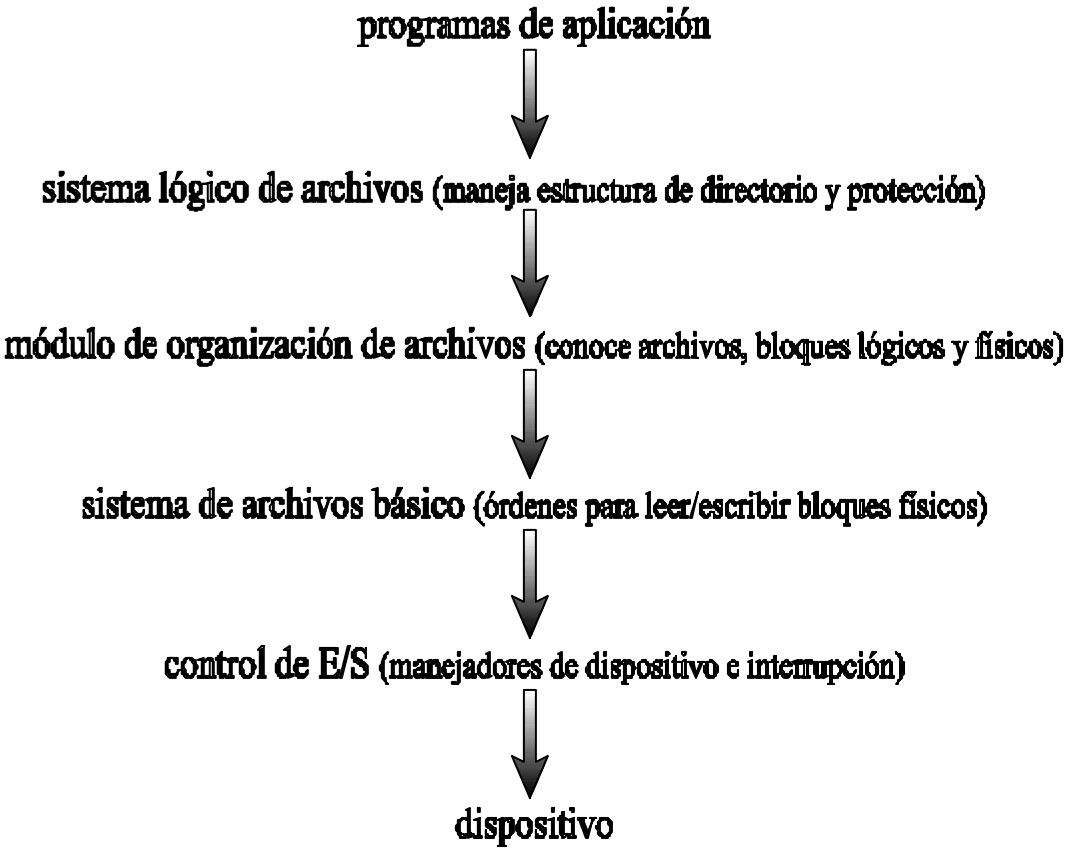
1. Introducción
2. Implementación
3. Estructura del almacenamiento secundario

2. Implementación: 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**

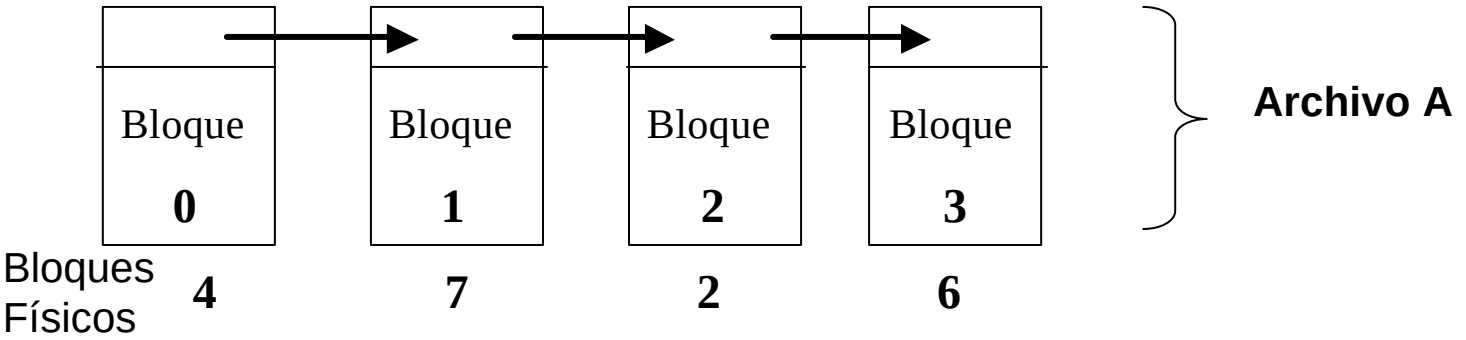
Supongamos que los bloques de disco son de 512 bytes:

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)

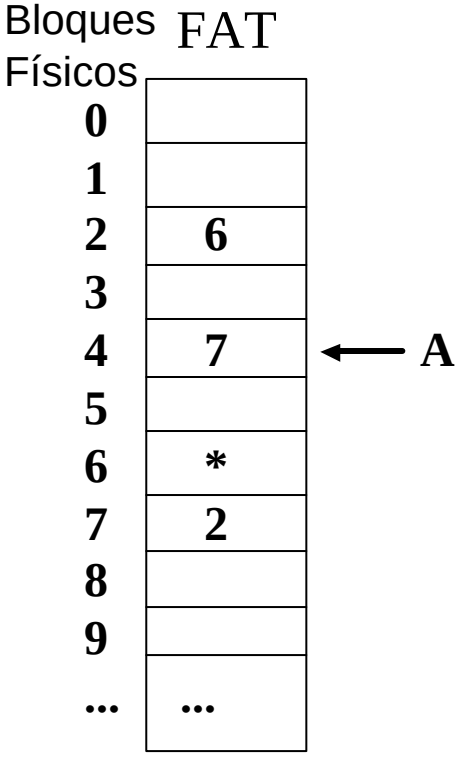
- **Desventajas**
  - El acceso directo no es efectivo (si el secuencial)
  - Espacio requerido para los punteros de enlace. Solución: agrupaciones de bloques (*clusters*)
  - Seguridad por la pérdida de punteros. Solución: lista doblemente enlazada (*overhead*)
- **Asociación lógica a física** (dirección = 1byte)

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

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

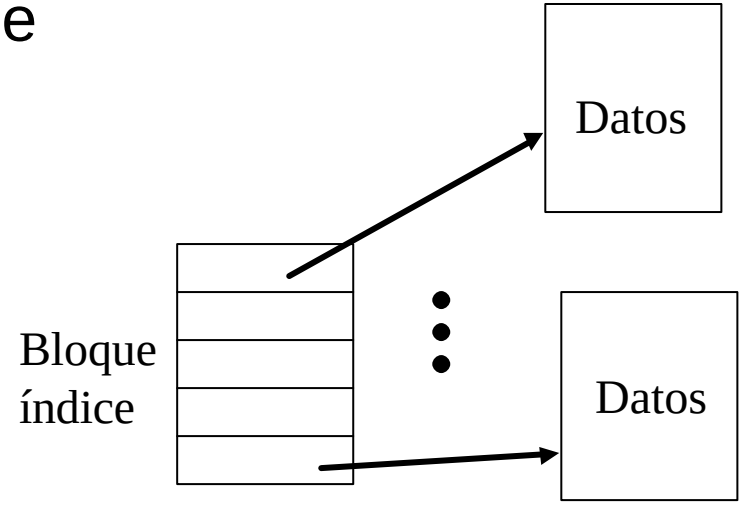
No Contiguo - Enlazado (y III)

- 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

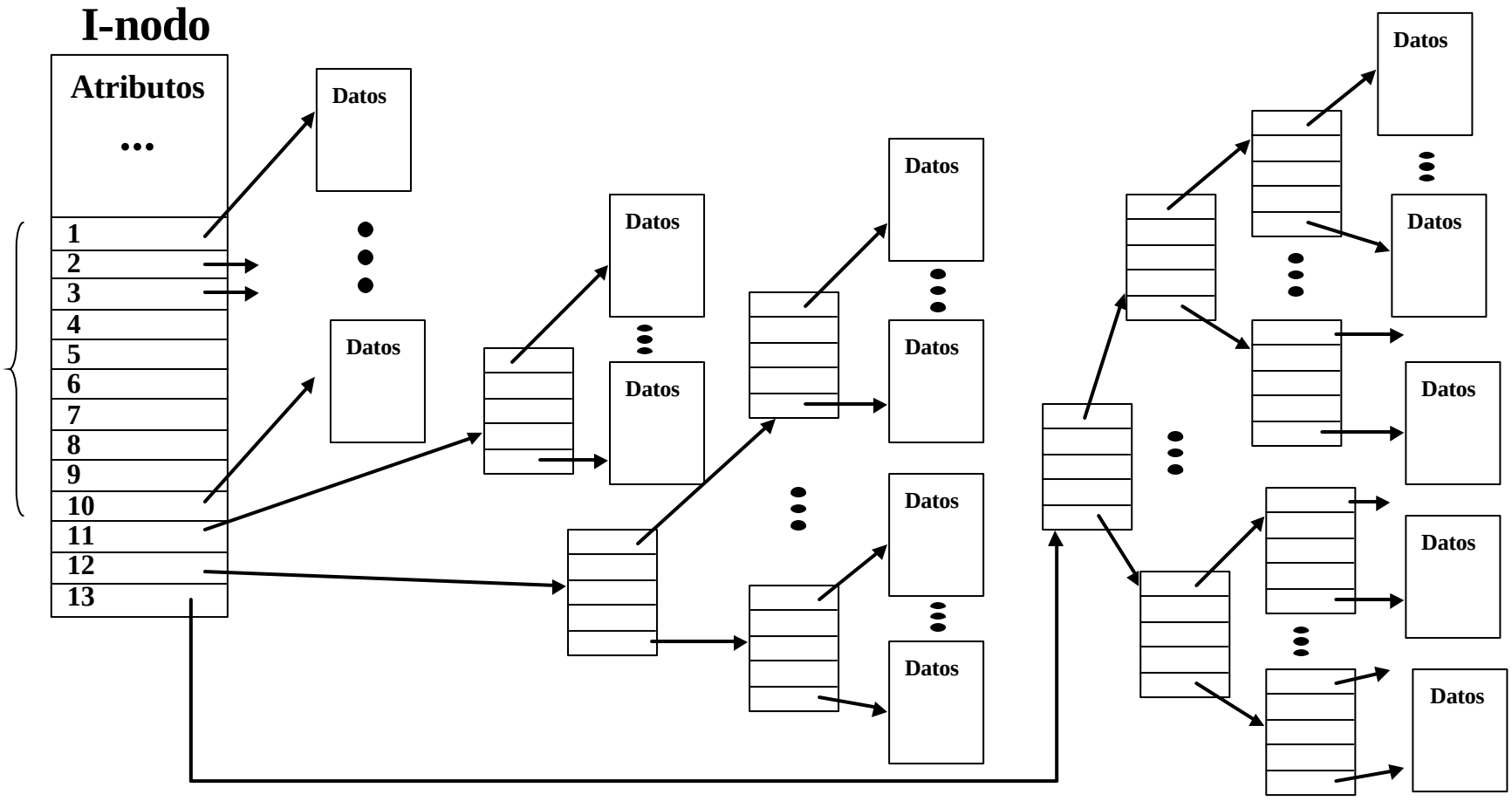
  (b) Bloques índices multinivel

  - 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:

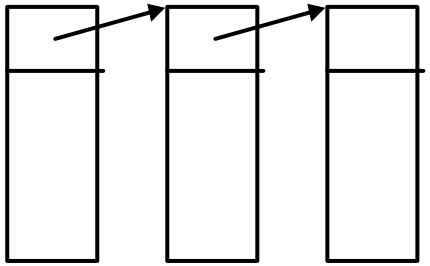
1. **Mapa o Vector de Bits**

10010001
11111101
11100000
11111110
00000000
11100011
11100000

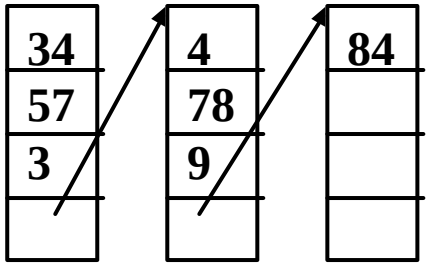
– Cada bloque se representa con un bit (0-Bloque libre; 1-Bloque ocupado)
– Fácil encontrar un bloque libre o *n* bloques libres consecutivos. Algunas máquinas tienen instrucciones específicas
– Fácil tener archivos en bloques contiguos
– Ineficiente si no se mantiene en memoria principal

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



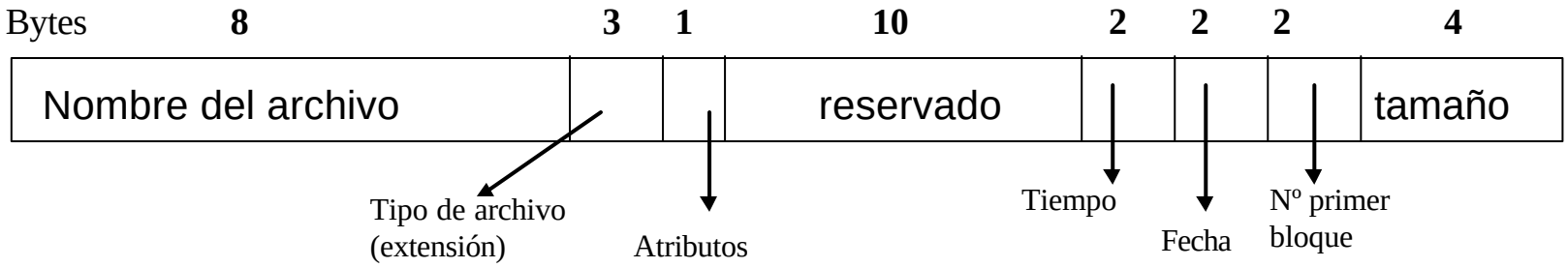
3. **Lista enlazada con agrupación**
   – Cada bloque de la lista almacena *n-1* direcciones de bloques libres
   – Obtener muchas direcciones de bloques libres es rápido
4. **Cuenta**
   – Cada entrada de la lista: una dirección de bloque libre y un contador del nº de bloques libres que le sigue

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)

Bytes	2	14
		Nombre del archivo

- 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**

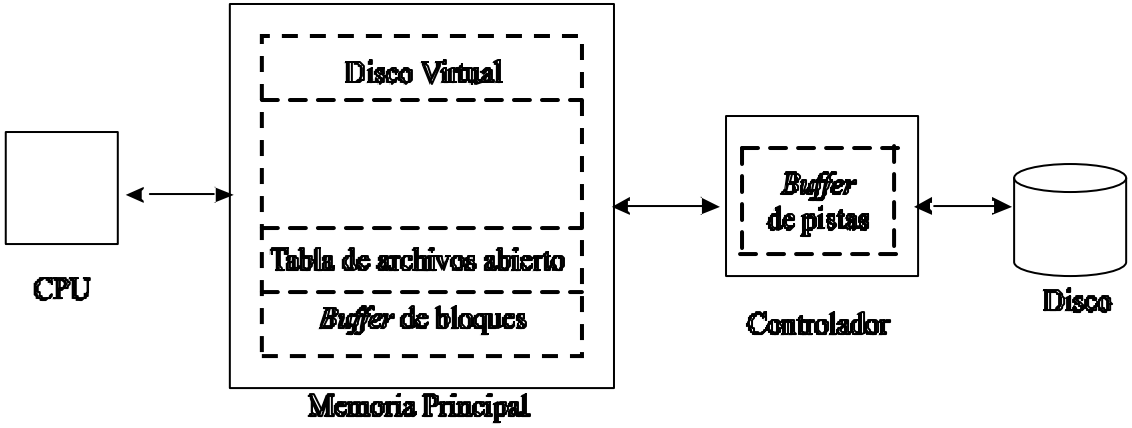
Implementación de Directorios (y IV)

- 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

Tema 1: Implementación del sistema de archivos

1. Introducción
2. Implementación
3. Estructura del almacenamiento secundario

3. Estructura del almacenamiento secundario: 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

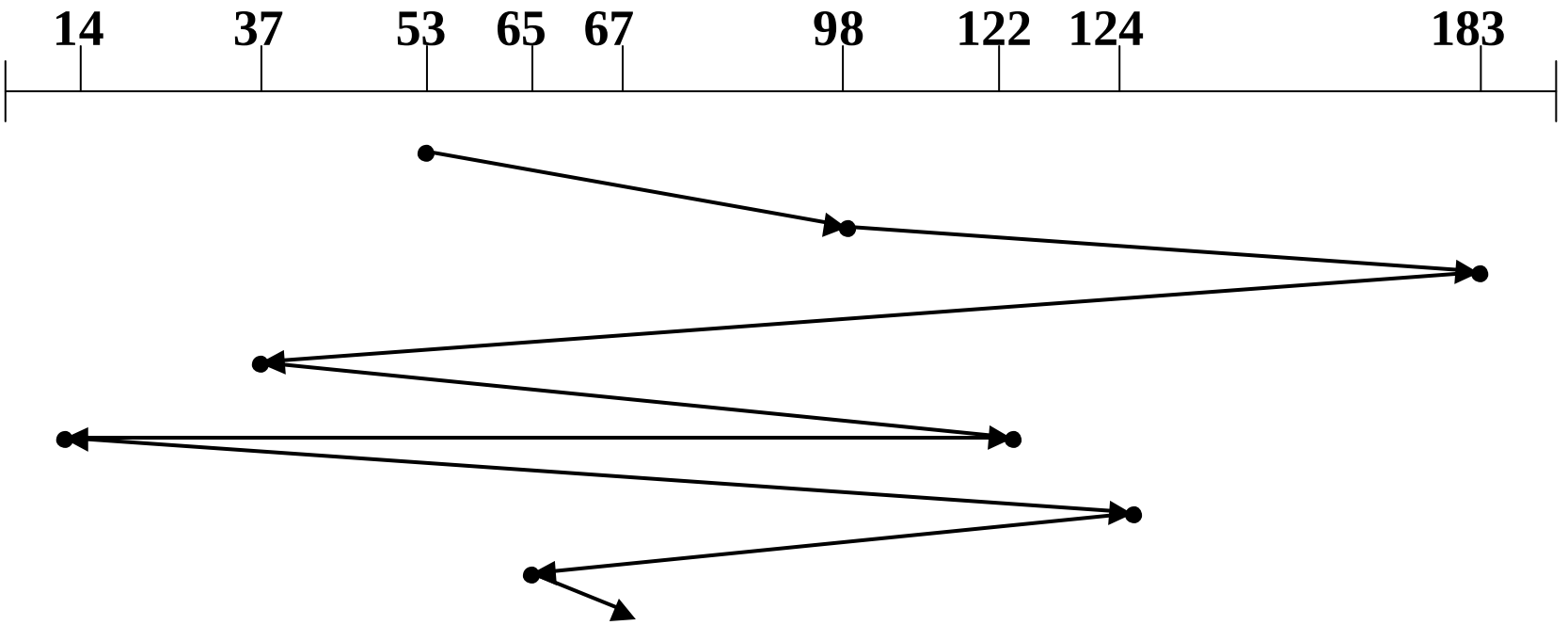
- La planificación intenta minimizar el **tiempo de posicionamiento** ≈ distancia de posicionamiento
- Si el disco está ocupado, las peticiones se encolan

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

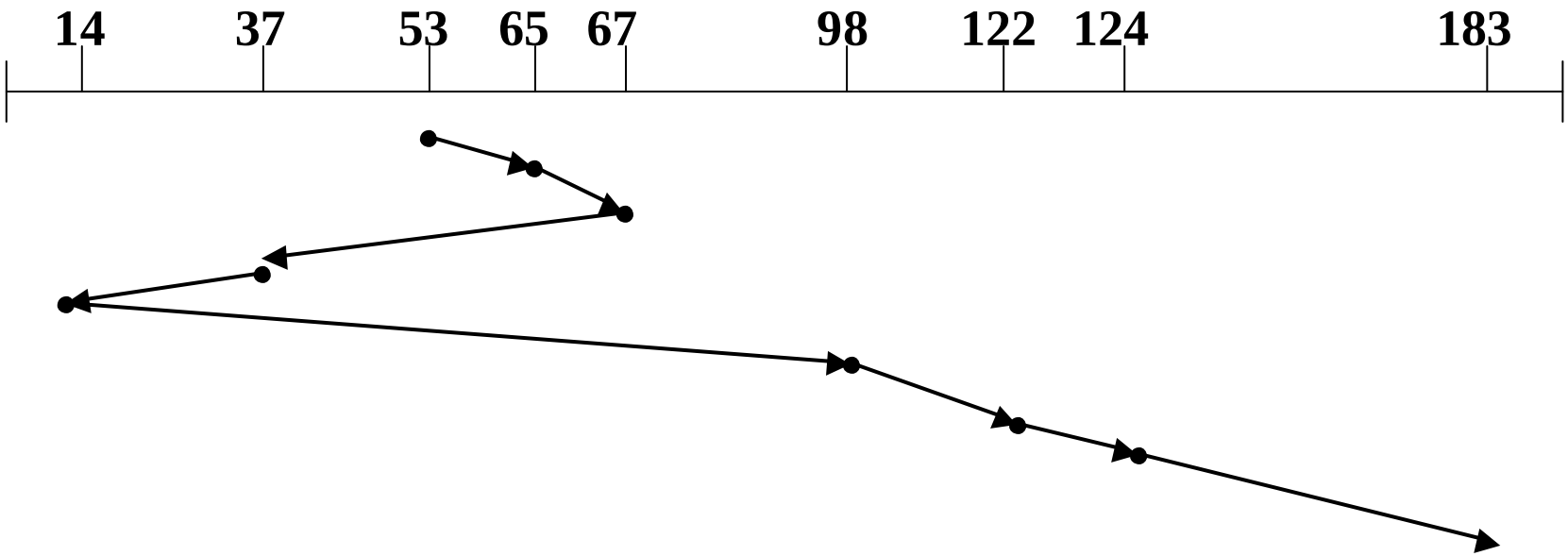
Planificación de Disco (III)

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



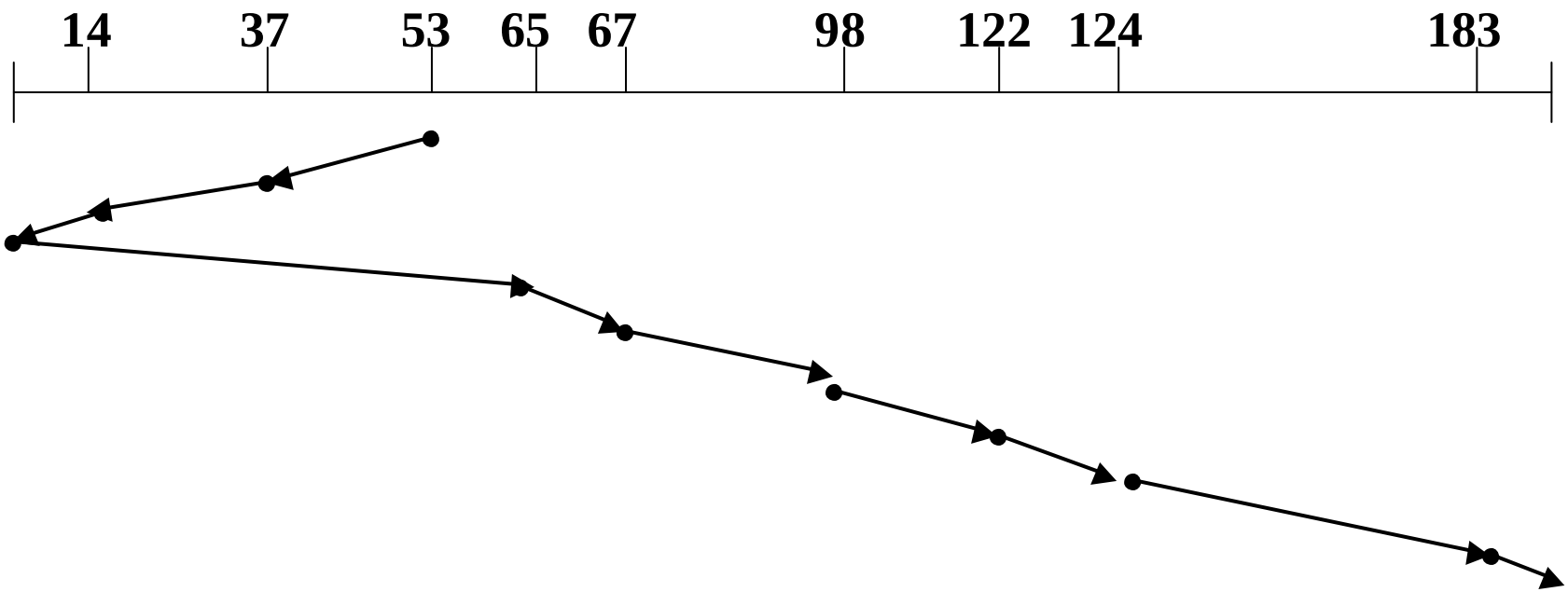
Planificación de Disco (IV)

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



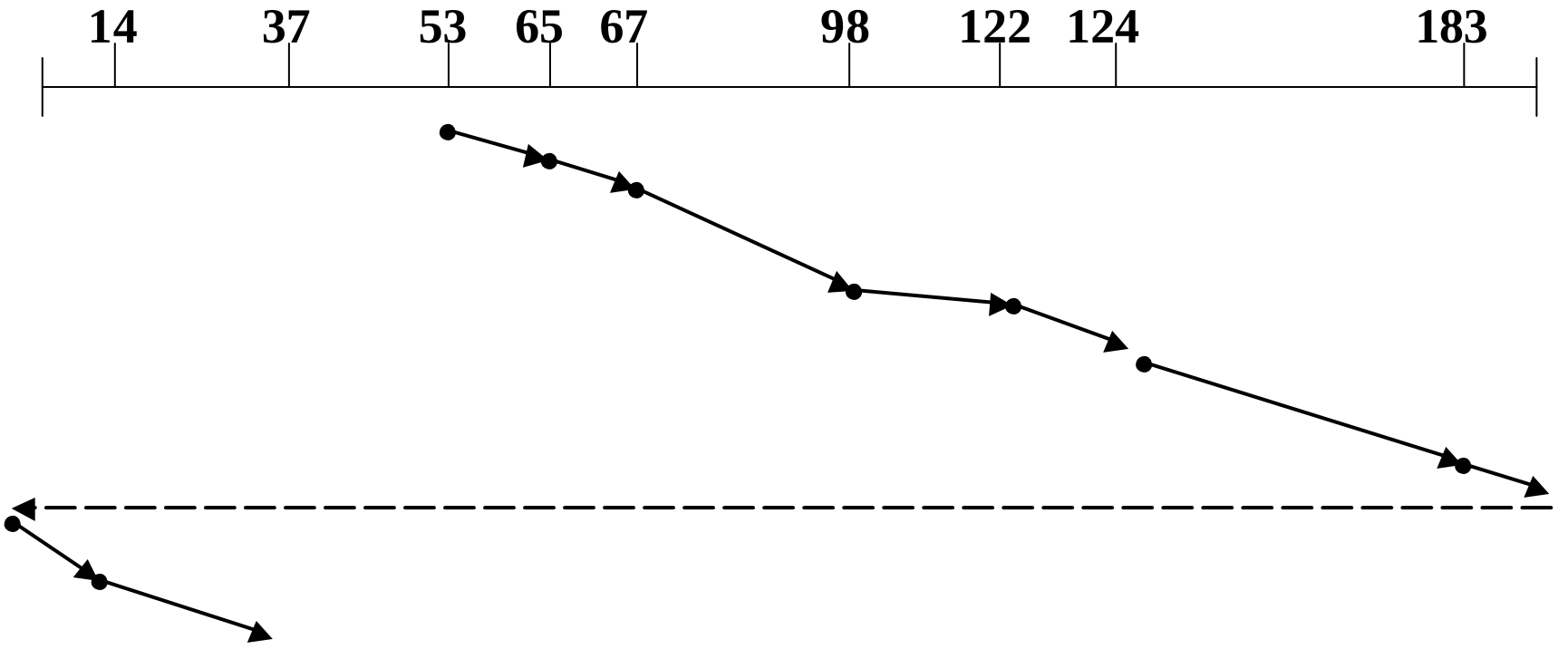
Planificación de Disco (V)

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



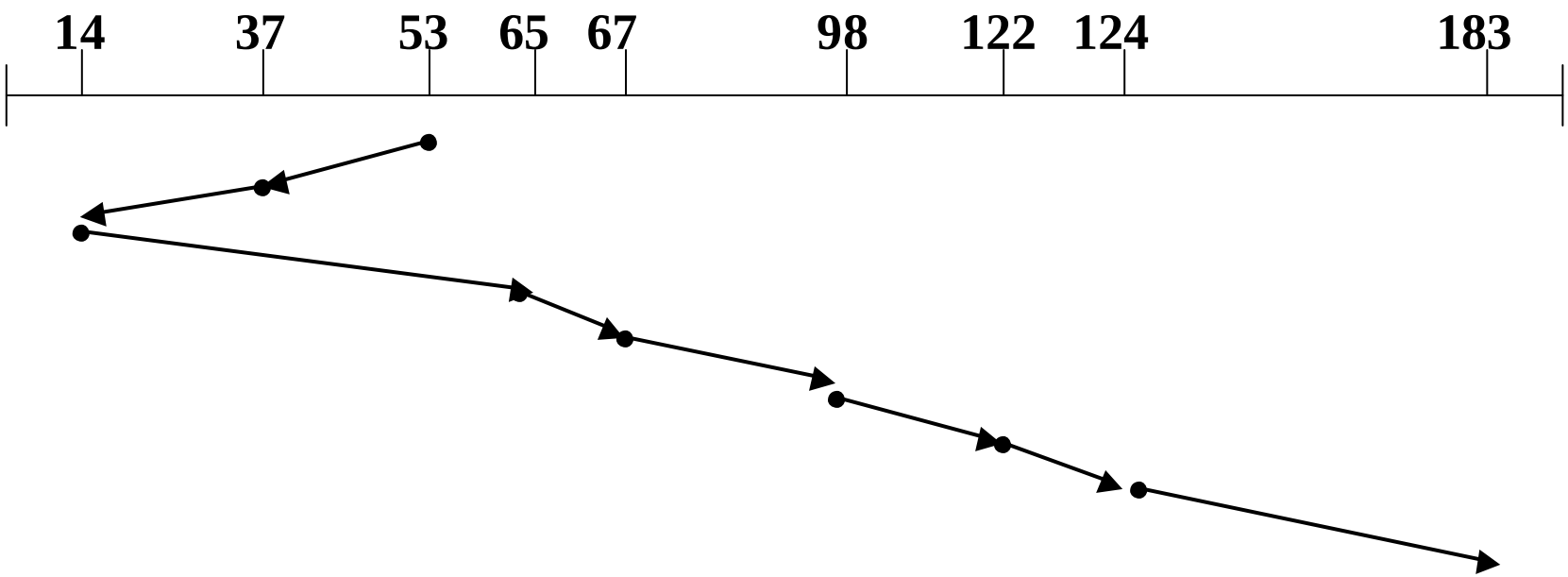
Planificación de Disco (VI)

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



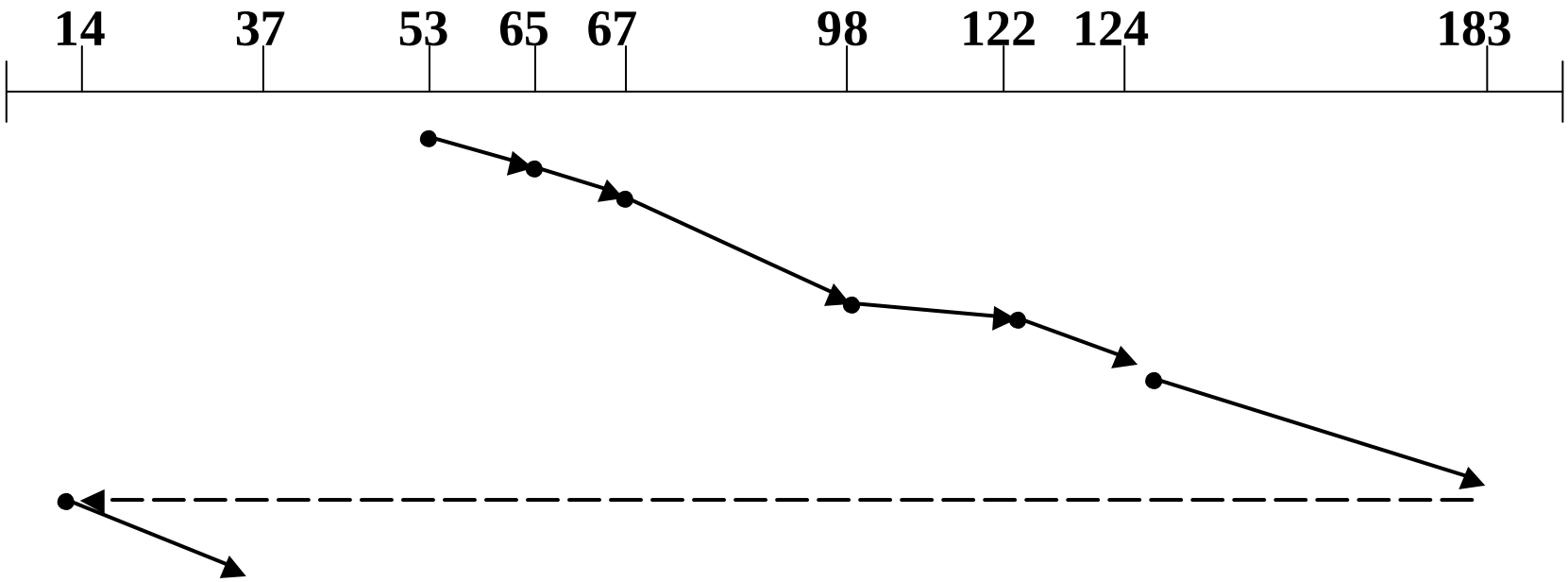
Planificación de Disco (VII)

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



Planificación de Disco (VIII)

- **C-LOOK**

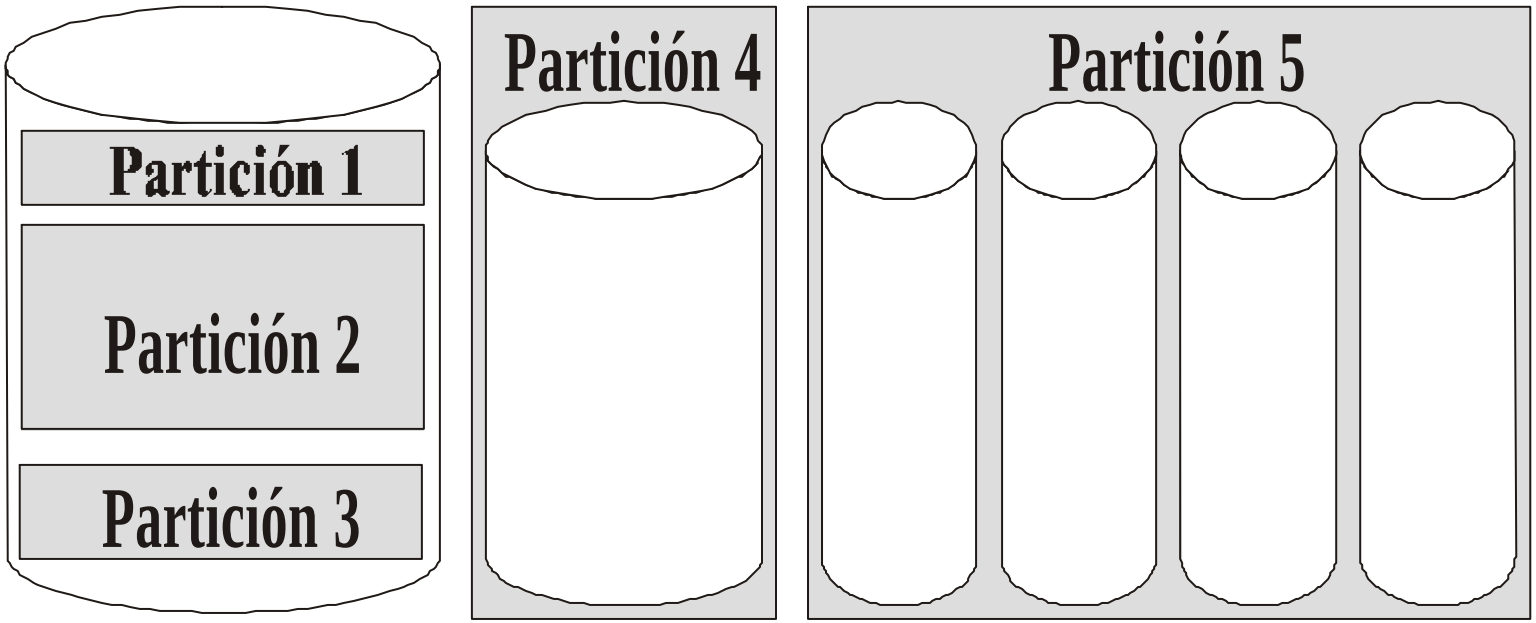


Selección de un algoritmo de planificación

- 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

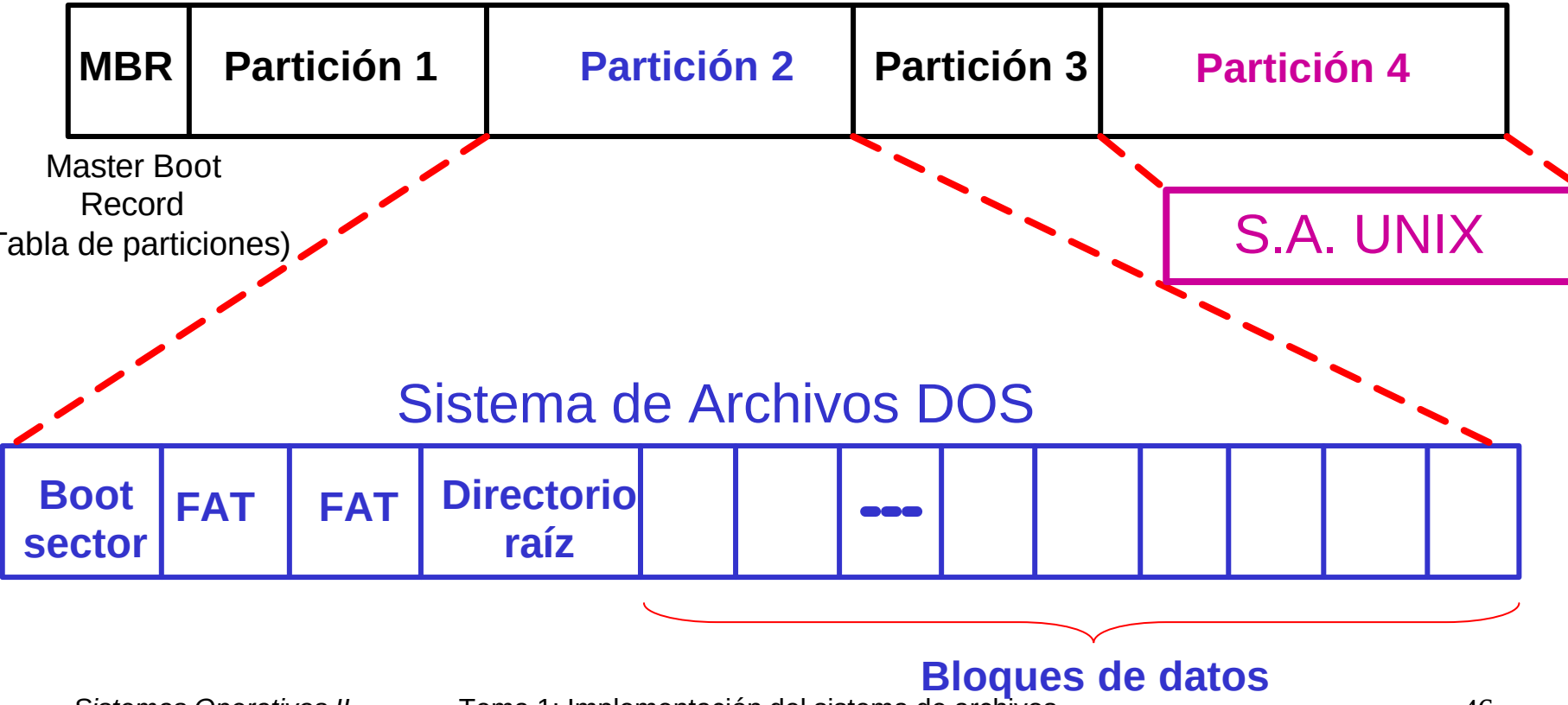
Gestión de disco

- **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)

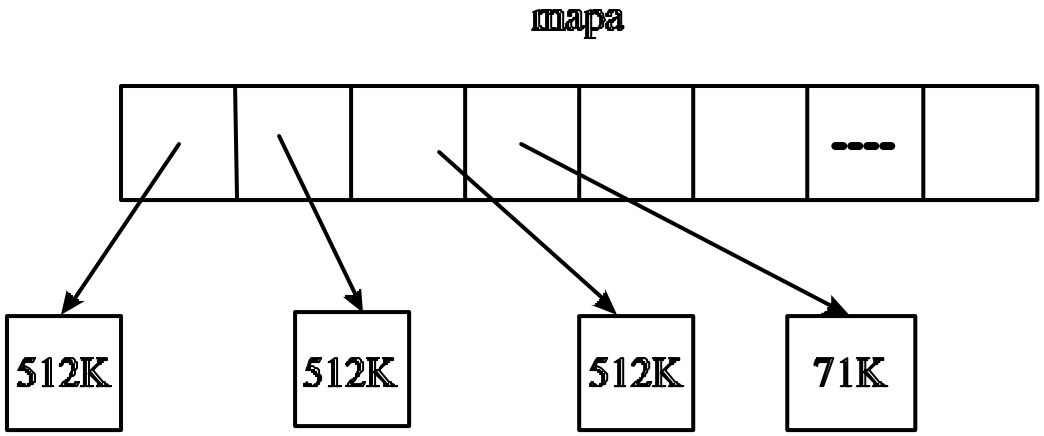
- Asignación del espacio de intercambio. Dos posibilidades:
  - En el propio **sistema de archivos** (p.e. Windows)

  → 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)

Gestión del Espacio de Intercambio (y III)

- 4.3BSD asigna espacio de intercambio cuando se **lanza** el proceso y utiliza una partición independiente
- El espacio de direcciones de un proceso se divide en páginas y éstas se guardan en el espacio de intercambio
- El núcleo utiliza dos mapas de intercambio por proceso:

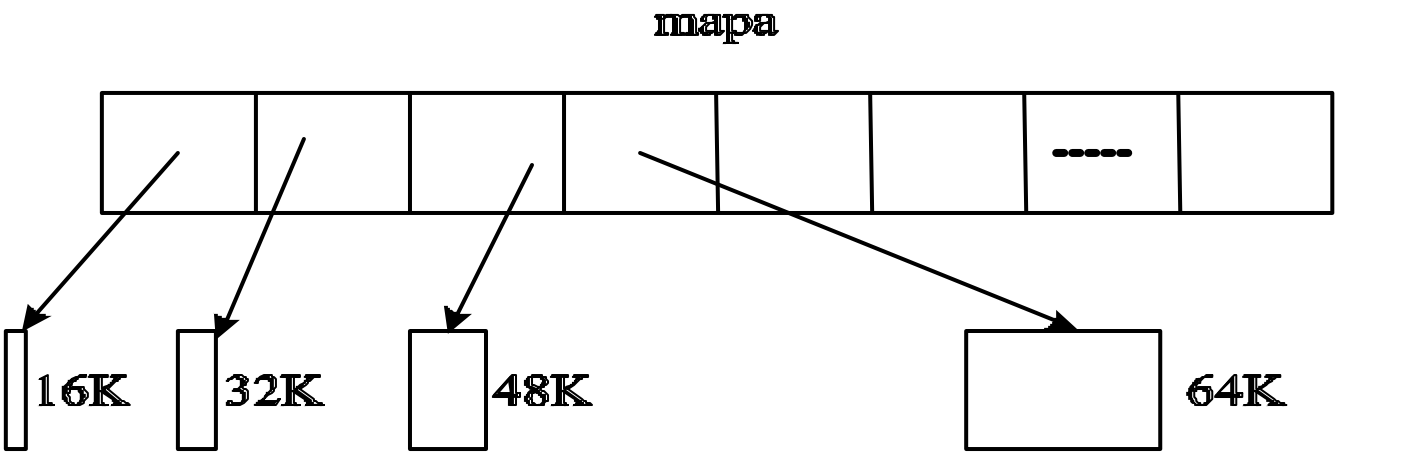
1. **Mapa de texto** (tamaño fijo)



4.3BSD Mapa de Intercambio del Segmento de Texto

Gestión del Espacio de Intercambio (y IV)

2. **Mapa de datos** (tamaño variable). Dado un índice *i*, el bloque apuntado por la entrada *i* tiene tamaño *i*16K*

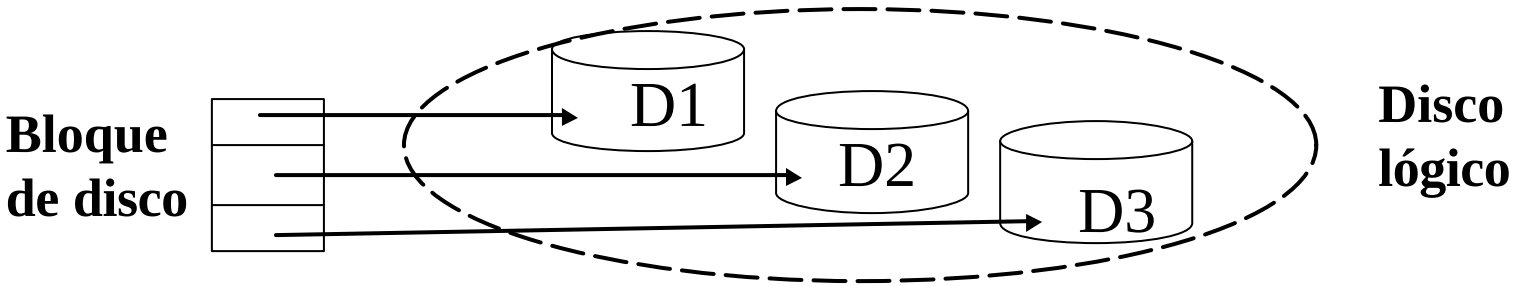


4.3BSD Mapa de Intercambio del Segmento de Datos

- En sistemas modernos, con más memoria principal, se asigna espacio de intercambio solo cuando la página se saca de memoria (Solaris 2)

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

Fin del tema 1: Implementación del sistema de archivos

1. Introducción
2. Implementación
3. Estructura del almacenamiento secundario