Sistemas Operativos

Sistema de Archivos II

Curso 2025

Facultad de Ingeniería, UDELAR

Agenda

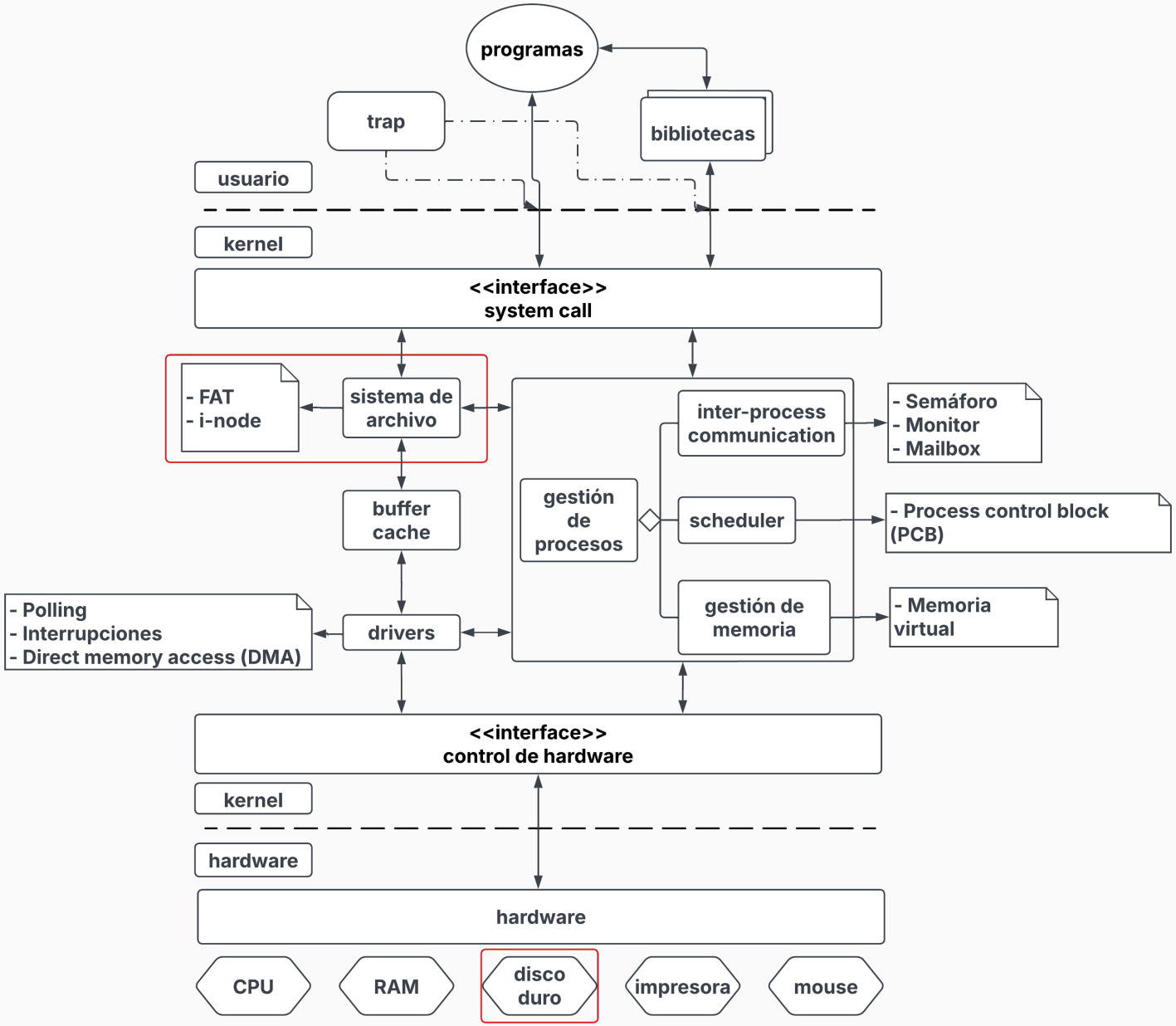
1. Repaso clase anterior

2. Sistema de archivos virtual

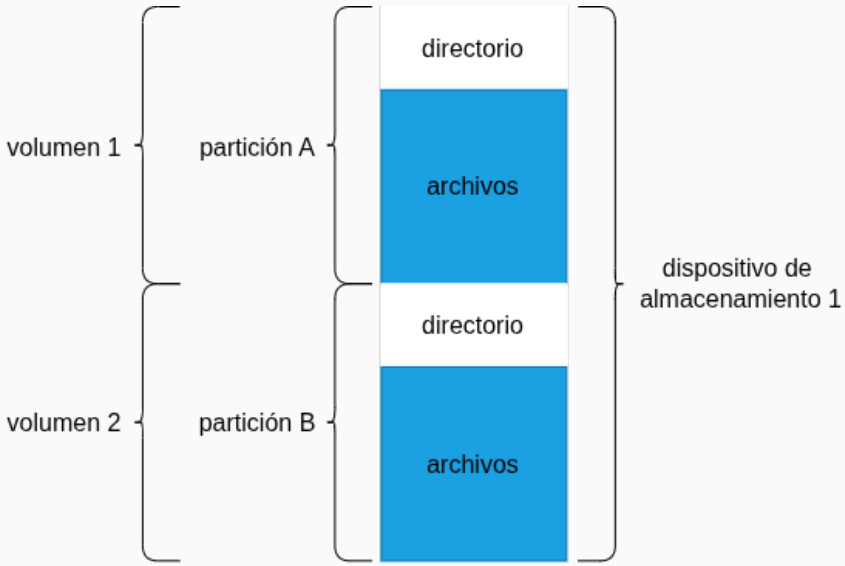
3. Ejercicios de sistemas de archivos

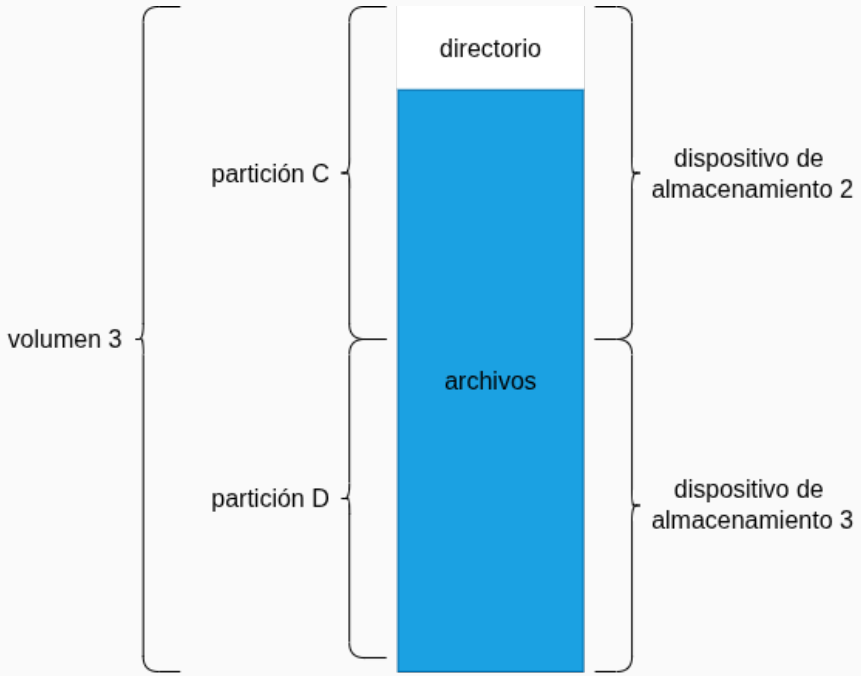
Repaso clase anterior

Arquitectura del sistema operativo



Organización





Estructura: en dispositivo

- Los dispositivos físicos contienen las estructuras:
  - Bloque de control para el arranque por volumen (*boot control block*): es necesario para iniciar el sistema operativo en el volumen. Este bloque es típicamente el primer bloque del volumen y puede estar vacío.
  - Bloque de control de volumen por volumen (*volume control block*): contiene información del volumen como cantidad de bloques, tamaño del bloque, bloques usados y libres, etc.
  - Estructura de directorios por sistema de archivos: para la organización de los archivos, incluye información de directorios, nombres de archivos, punteros al archivo, etc.
  - Bloque de control del archivo por archivo (*file control block (FCB)*): contiene los detalles del archivo, en particular debe indicar que bloques del disco contienen la información del archivo.

Estructura: en memoria

- El sistema operativo mantiene en memoria las estructuras:
  - La tabla de montado con información de los volúmenes montados.
  - La caché de estructura de directorio con la información de los directorios accedidos recientemente.
  - La tabla de descriptores de archivos abiertos a nivel global del sistema: contiene una copia del FCB de cada archivo abierto.
  - La tabla de descriptores de archivos abiertos por cada proceso del sistema: contiene punteros a la tabla global de archivos abiertos.

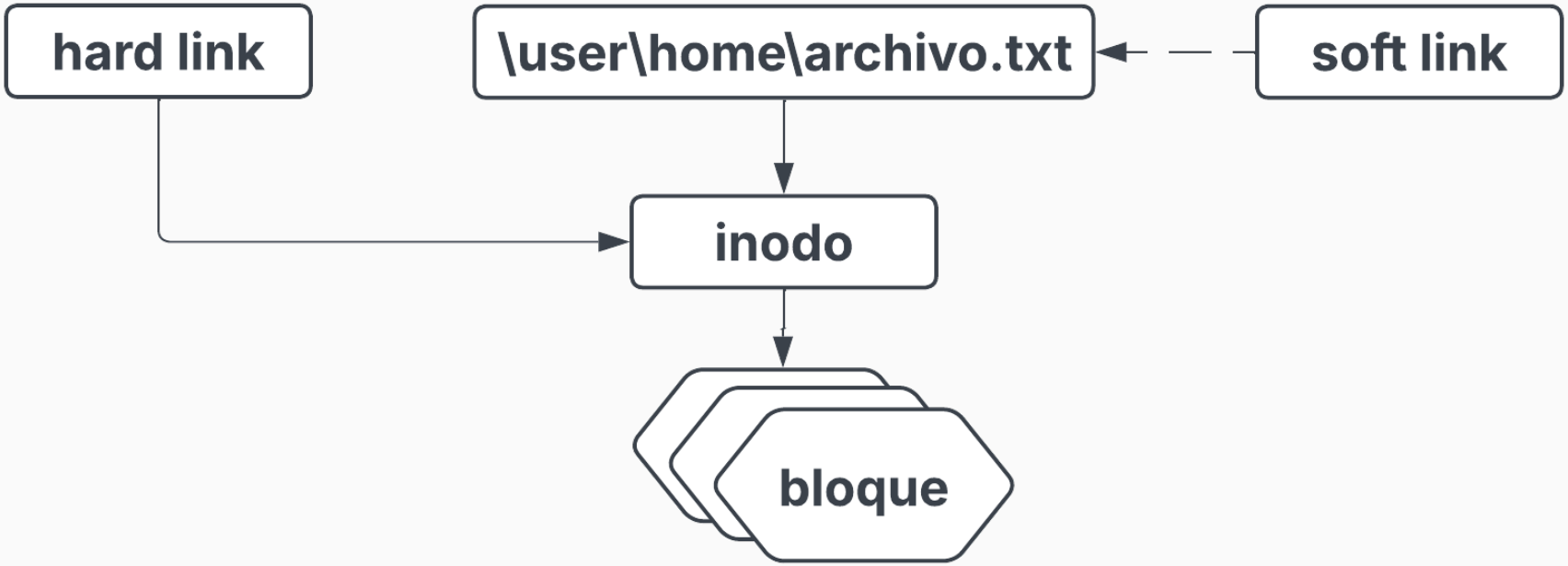
Estructura: por archivo

- Cada archivo en el sistema tiene un bloque de control.
- El bloque de control contiene varios atributos de conteo, permisos y donde están los datos del mismo:
  - Fechas (creación, acceso, modificación).
  - Usuario propietario y grupo propietario.
  - Permisos del archivo.
  - Tamaño del archivo.
  - Bloques de datos del archivo.

Estructura: por directorio

- Los directorios contienen la información de los archivos que pertenecen a él (como mínimo: nombre y referencia de ubicación).
- Para organizar la información existen varias alternativas. Por ejemplo:
  - Lista lineal: el nombre de cada archivo y un puntero sus bloques de datos son dispuestos en una lista (casi) lineal. En la búsqueda, inserción, borrado, etc., es necesario un acceso lineal.
  - Tabla de hash abierto: con el nombre del archivo se genera una clave que ayuda a identificar en que bloque se encuentra la entrada buscada. Luego la búsqueda se resuelve de forma lineal.

Estructura: por directorio – Grafo

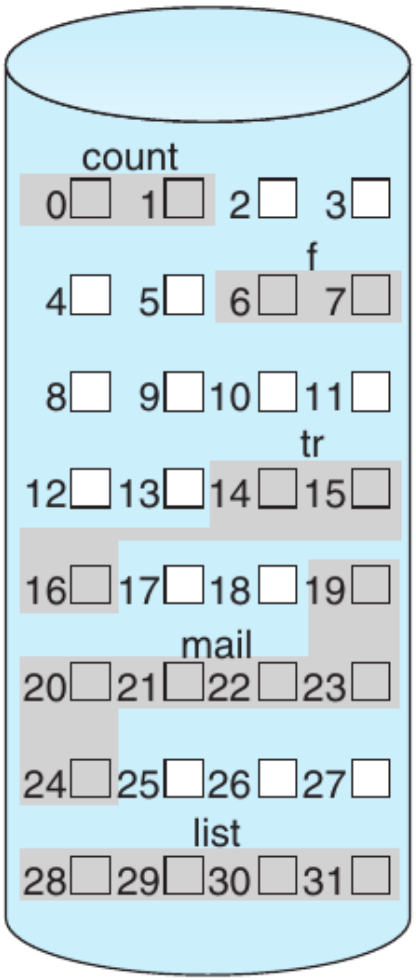


Métodos de asignación

- Para la organización de los datos de un archivo en disco se tienen, en general, tres métodos:
  - Asignación contigua: Los datos son dispuestos en forma contigua. Para mantener la información es necesario saber en que bloque comienzan los datos y la cantidad de bloques de datos.
  - Asignación en forma de lista: Los bloques de datos forman una lista encadenada. Es necesario una referencia al primer y último bloque de datos.
  - Asignación indexada: Se mantiene una tabla en donde cada entrada referencia a un bloque de datos.

Asignación contigua

- Sufre de fragmentación externa.
- Es necesario reubicar continuamente los archivos si crecen en tamaño.
- Se utilizan técnicas de asignación de tamaños más grandes para prever el crecimiento futuro de los archivos.

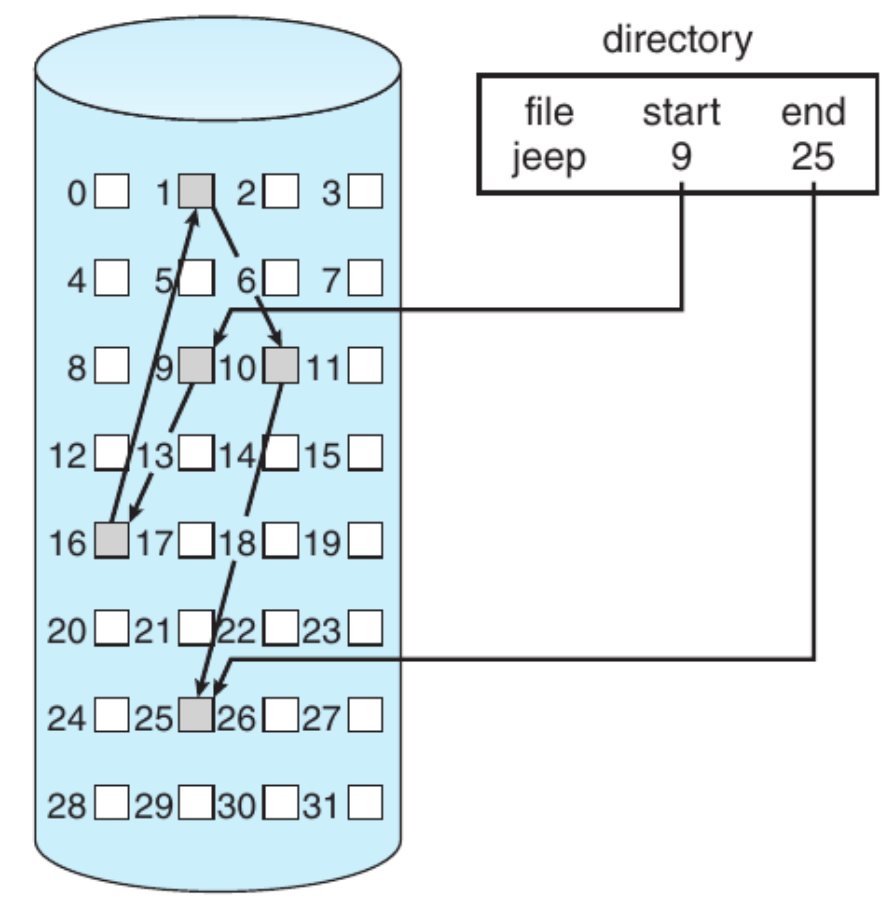


directory

file	start	length
count	0	2
tr	14	3
mail	19	6
list	28	4
f	6	2

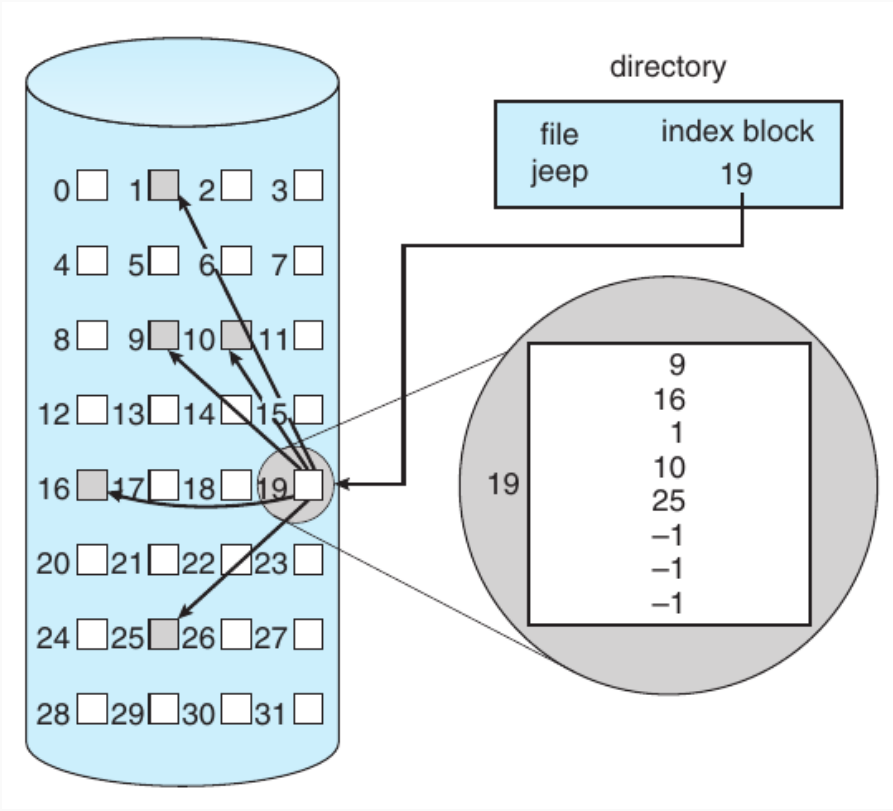
Asignación en forma de lista

- Soluciona el problema de la fragmentación externa.
- El acceso a los bloques es de orden lineal.
- Los punteros ocupan espacio en los bloques.
- La pérdida de una referencia genera la pérdida de gran parte de información del archivo.



Asignación indexada

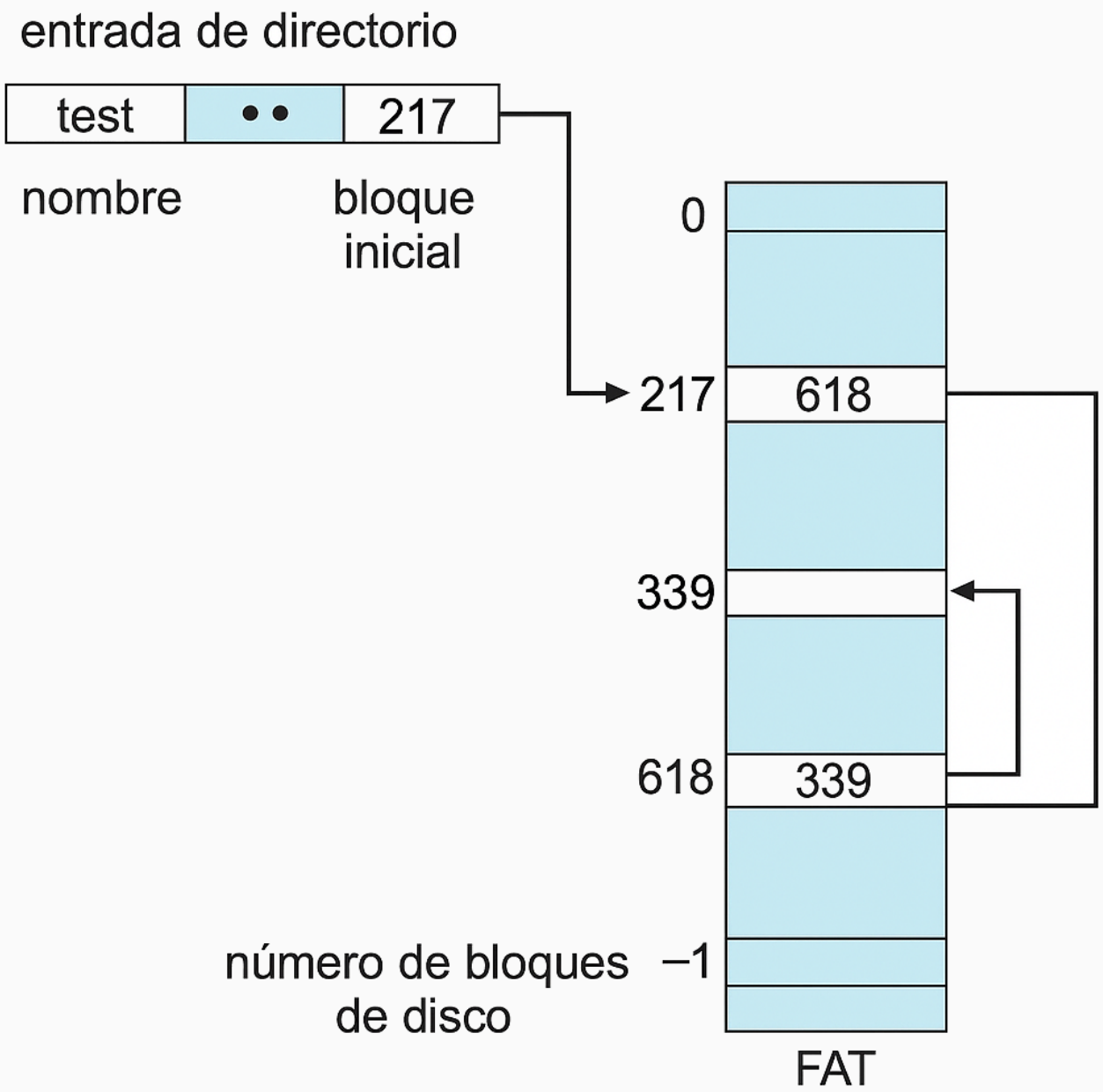
- Los bloques son accedidos directamente a través del bloque de indexación (*index block*).
- El bloque de indexación ocupa lugar por lo que se trata de que sea lo más pequeño posible.
- Una estrategia utilizada es la indexación en varios niveles. Algunos índices hacen referencia a bloques de datos (directos) y otros a bloques con referencias a bloques de datos (indirectos).



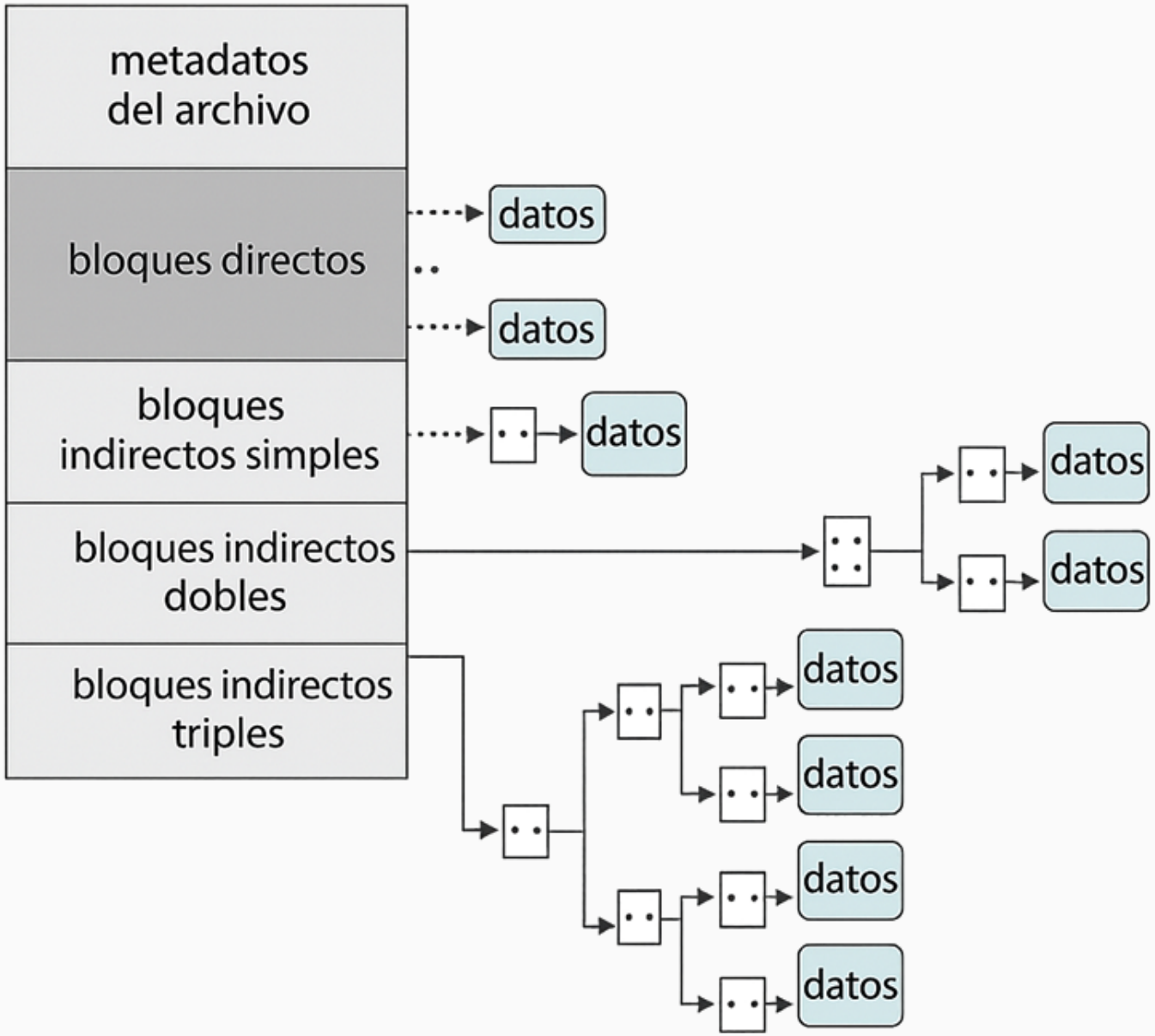
Asignación en forma de lista: Ejemplo FAT

- Tiene una tabla de asignación de archivos (*File Allocation Table*) al comienzo de cada volumen.
- Esta tabla contiene la lista de bloques de cada archivo.
- Tiene una entrada por cada bloque de disco, y es indexada por el número de bloque.

Asignación en forma de lista: Ejemplo FAT



Asignación indexada: Ejemplo UNIX



Administración del espacio libre

- En el sistema de archivos es necesario mantener que bloques están ocupados y cuales están libres.
- Alternativas posibles para la administración de los bloques:
  - Vector de bits, mapa de bits: se dispone de un bit para cada bloque del dispositivo, que representa si está ocupado o libre.
  - Lista de bloques libres: Se mantiene una lista encadenada con los bloques libres a través de los bloques. Es necesario una referencia al primer bloque.
  - Agrupación: es una variación de la lista encadenada. En cada bloque de la lista se contiene un grupo de bloques libres.
  - Conteo: se mantiene una lista en donde cada bloque contiene información de cuantos bloques contiguos, a partir de él, están libres.

Sistema de archivos virtual

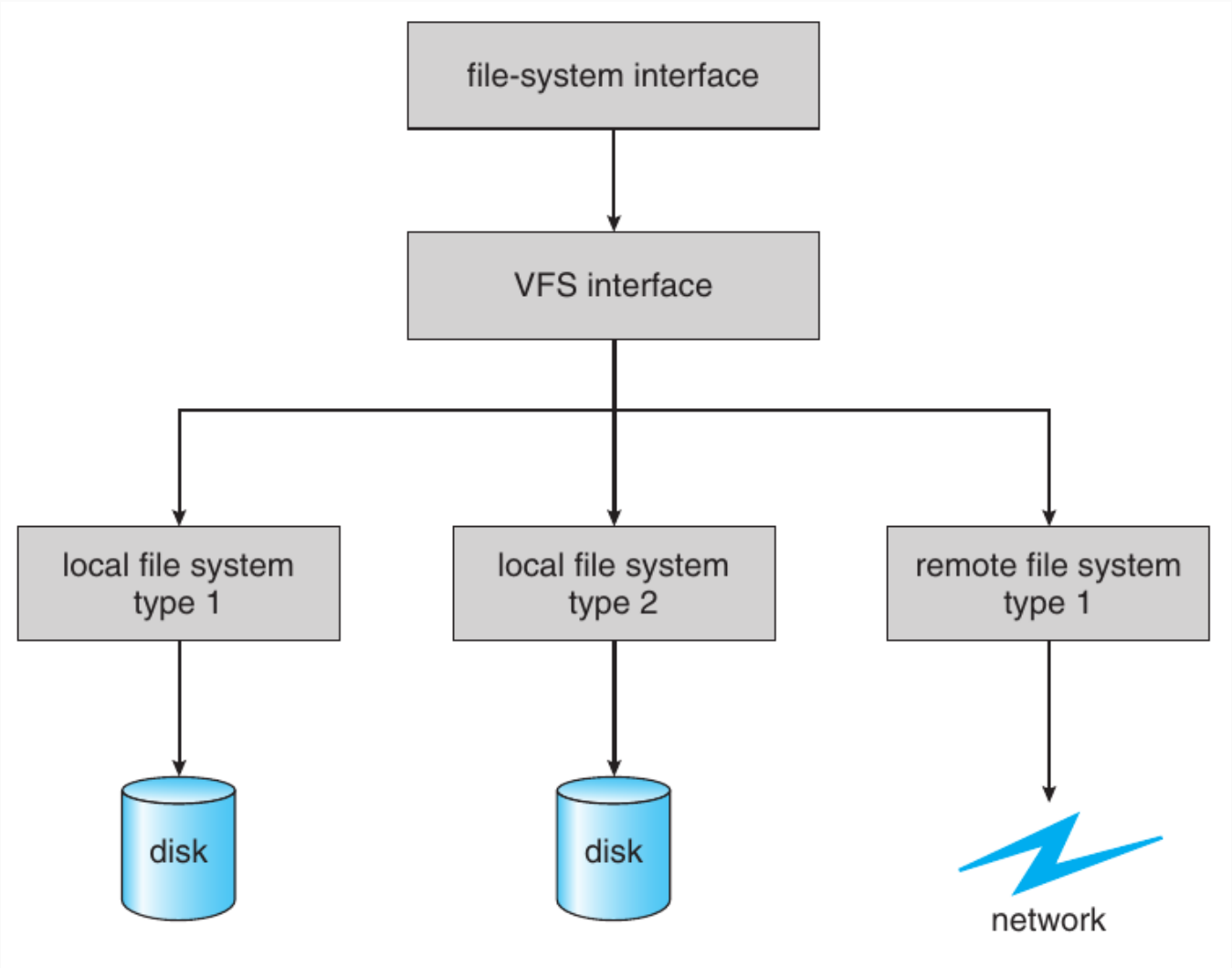
Sistema de archivos virtual

- Es común que un sistema operativo se acceda a más de una implementación de sistema de archivos (ufs, ext4, btrfs, jfs, ntfs, etc.).
- Se utilizan técnicas de orientación a objetos para lograr mantener una estructura independiente del sistema de archivos que se utilice.
- Se genera una estructura en tres capas:
  - Interfaz del sistema de archivo (llamadas a sistema open, read, etc.).
  - Sistema de archivos virtual (*Virtual File System*).
  - Implementación especifica del sistema de archivo.

Sistema de archivos virtual

- El sistema de archivos virtual provee de dos funcionalidades importantes:
  - Propone una interfaz genérica de sistema de archivo que es independiente del tipo de sistema de archivo. De esta forma, se logra un acceso transparente al sistema de archivos.
  - Propone un bloque de control de archivo virtual que puede representar tanto archivos locales como remotos.

Sistema de archivos virtual



Ejercicios de sistemas de archivos

Problema FAT

Se tiene un sistema de archivos que utiliza una estrategia indexada simple como muestra la siguiente estructura de datos:

Estructura 1/2

```
const MAX_BLOQUES = 16777216; // 2 ^ 24
type bloque = array [0..4095] of byte;
type entradaDir = Record
    usado: bit; // entrada usada o no (1 bit)
    nombre: array [0..22] of char; // nombre del elemento (23
        bytes)
    inicio: integer; // direccion de comienzo (4 bytes)
    tipo: {ARCHIVO, DIRECTORIO}; // tipo de elemento (1 bit)
    tam: integer; // tamaño del archivo en bytes (4 bytes)
    reservado: array[0..5] of bit;// espacio reservado x el
        sistema (6 bits)
end; // 32 bytes
```

Estructura 2/2

```
type fat = Array [0..(MAX_BLOQUES - 1)] of -2..(MAX_BLOQUES-1);
type disco = Array [0..(MAX_BLOQUES-1)] of sector;
type mapaBits = Array [0..MAX_BLOQUES-1] of bit;

type ubicacionEntrada = Record
    entrada: entradaDir;
    bloque: integer;
    indice: integer; // Indice de la entrada de directorio
        dentro del bloque
end;

var F: fat;
    D: disk;
    MB: mapaBits;
```

Problema FAT

- Las variables F, D y MB son globales.
- La variable F siempre se encuentra cargada en memoria con el contenido de la FAT. No se debe modelar la persistencia en disco de la FAT que además se sabe que se encuentra fuera del rango 0..MAX_BLOQUES.
- Archivos y directorios son alojados en D usando la estructura de la FAT. El directorio raíz comienza en el sector número 0 de D.
- En la FAT el valor -1 representa “fin de archivodirectorio” y el -2 “bloque libre”.

Problema FAT

Se dispone de los siguientes procedimientos:

- `Procedure leerBlq(numBloque: int, var buff: bloque) : boolean`
  Lee de disco el bloque `numBloque`, pasado como parámetro, y carga el contenido leído en el parámetro de salida `buff`. Retorna `true` en caso de que la operación se ejecute con éxito y `false` en caso contrario.
- `Procedure escrBlq(numBloque: int, buff: bloque) : boolean`
  Escribe en el bloque `numBloque`, pasado como parámetro, la información que se encuentra en el parámetro `buff`. Retorna `true` en caso de que la operación se ejecute con éxito y `false` en caso contrario.

Problema FAT

Implementar una función que busque un archivo en un directorio. Retorna la ubicacionEntrada del archivo correspondiente en caso que exista, y null en caso contrario.

```
function buscarArchivo(entradaDir directorio,char* nomArchivo):ubicacionEntrada
```

Problema inodo

Un sistema de archivos usa una estrategia indexada de dos niveles con 7 bloques directos en el primer nivel y 1 bloque de indirección simple en el segundo nivel, y un mapa de bits para administrar el espacio libre del disco.

Las variables IT, MB, y D son globales, el inodo número 0 es el directorio raíz y las entradas de los directorios son almacenadas utilizando un array de entradas.

Estructura 1/2

```
const MAX_BLOQUES = 65536; // 2^16
const TAM_BLOQUE = 4096;   // 2^12
const MAX_INODOS = 8192;   // 2^13

type bloque = array [0..TAM_BLOQUE-1] of byte;
type mapa_bits = array [0..MAX_BLOQUES-1] of bit;

type entrada_dir = Record
    usado: bool;                    // 1 bit
    nombre: array [0..123] of char; // 124 bytes
    es_dir: bool;                   // 1 bit
    inodo_num: int16;               // 2 bytes
    permisos: array [0..13] of bit; // 14 bits
end; // 128 bytes
```

Estructura 2/2

```
type inodo = Record
    usado: bool;                        // 1 bit
    inodo_num: int16;                   // 2 bytes
    es_dir: bool;                       // 1 bit
    tamano: int32;                      // 4 bytes
    directo: array [0..7] of int16;     // 16 bytes
    directo_tope: int16;                // 2 bytes
    indirecto: int16;                   // 2 bytes
    indirecto_tope: int16;              // 2 bytes
    reservado: array [0..29] of bit; // 30 bits
end; // 32 bytes

type inodos_tabla = array [0..MAX_INODOS-1] of inodo;
type disco = array [0..MAX_BLOQUES-1] of bloque;

var IT: inodos_tabla;
    MB: mapa_bits;
    D: disco;
```

Problema inodo

Se dispone de los siguientes procedimientos:

`leerBloque(d: disco; bloque_num: 0..MAX_BLOQ-1; buffer: array [0..1023] of byte): bool`, lee desde el disco d el bloque con índice bloque_num en la variable buffer. Retorna verdadero en caso de que la operación haya sido ejecutada con éxito y falso en caso contrario.

`obtenerBase(path: array of char): array of char`, retorna la primer parte del camino en path. Por ejemplo:

```
obtenerBase('/home/sistoper/a.txt')= 'home'
```

`obtenerResto(path: array of char): array of char`, retorna el resto del camino en path sin la primer parte. Por ejemplo:

```
obtenerResto('/home/sistoper/a.txt')= '/sistoper/a.txt'
```

Problema inodo

Implemente una función que dado el nombre de un elemento (archivo o directorio) y el índice de un bloque asignado a un directorio, busque el elemento en el bloque. La función debe retornar en `inodo_num` el número de inodo correspondiente al elemento encontrado, o -1 si no es encontrado. La función retorna `falso` en caso de error y `verdadero` en caso contrario, y tiene la siguiente firma: `buscarEntrada(nombre: array of char; bloqueid: int16; var inodo_num: int16): bool`