Sistemas Operativos

Sistema de Archivos II

Curso 2024

Facultad de Ingeniería, UDELAR

Agenda

1. Implementación del sistema de archivos (continuación)
    - Estructura de los directorios
    - Métodos de asignación
    - Administración del espacio libre
2. Sistema de archivos virtual
3. Ejemplo UNIX

Implementación del sistema de archivos (continuación)

Estructura de los directorios

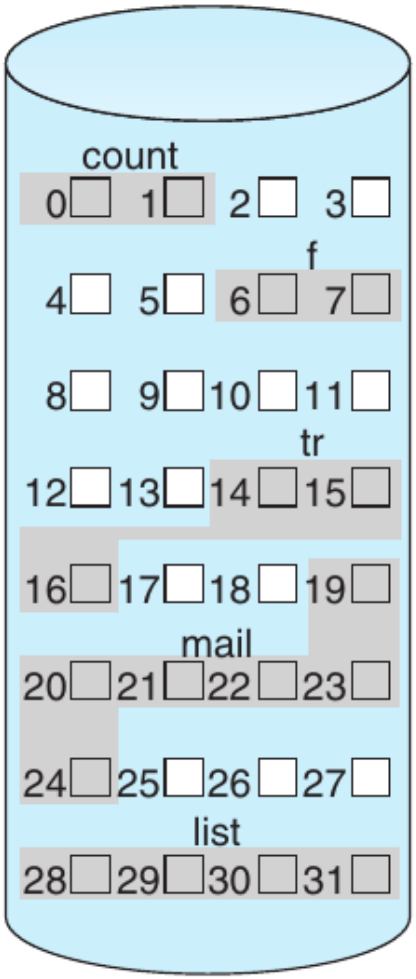
- Los directorios contienen la información de los archivos que pertenecen a él (como mínimo: nombre y referencia de ubicación).
- Para organizar esta información existen varias alternativas. Por ejemplo:
  - Lista lineal: el nombre de cada archivo y un puntero sus bloques de datos son dispuestos en una 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 un clave que ayuda a identificar en que bloque se encuentra la entrada buscada. Luego la búsqueda se resuelve de forma lineal.

Métodos de asignación

- Para la organización de los datos de un archivo en disco se tienen, en general, tres métodos:
  - Asignación contigua (*Contiguous Allocation*): 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 (*Linked Allocation*): Los bloques de datos forman una lista encadenada. Es necesario una referencia al primer y último bloque de datos.
  - Asignación indexada (*Indexed Allocation*): Se mantiene una tabla en donde cada entrada referencia a un bloque de datos.

Asignación contigua (Contiguous Allocation)

- 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.

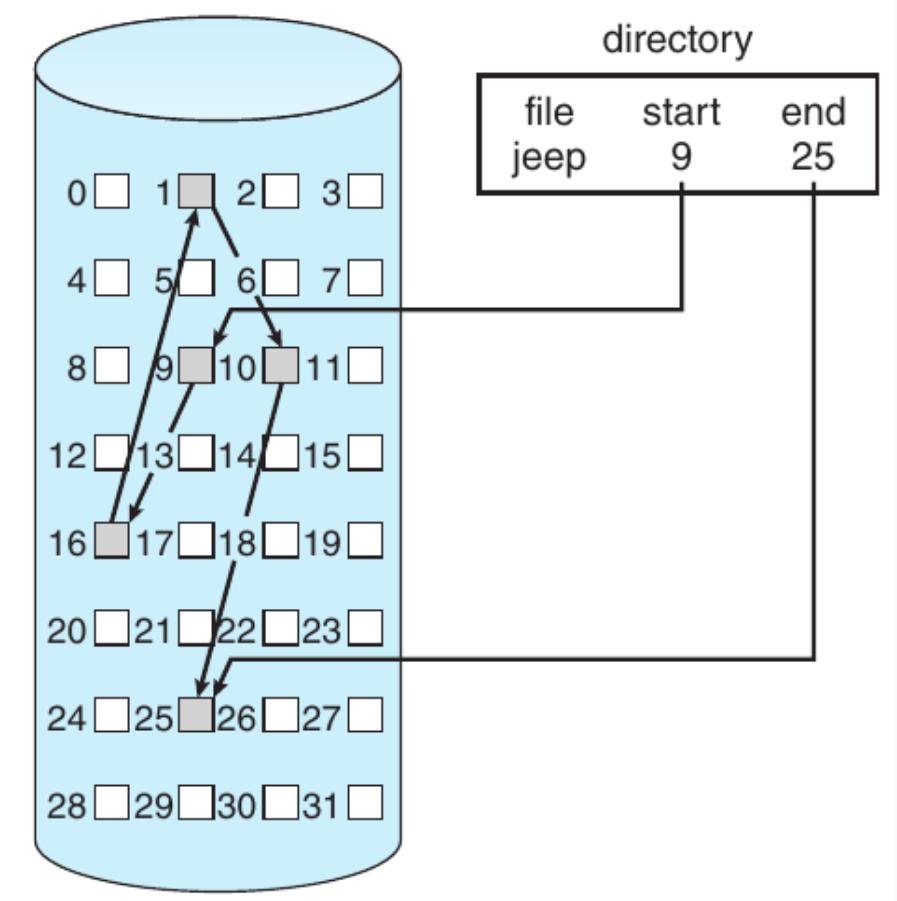


directory

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

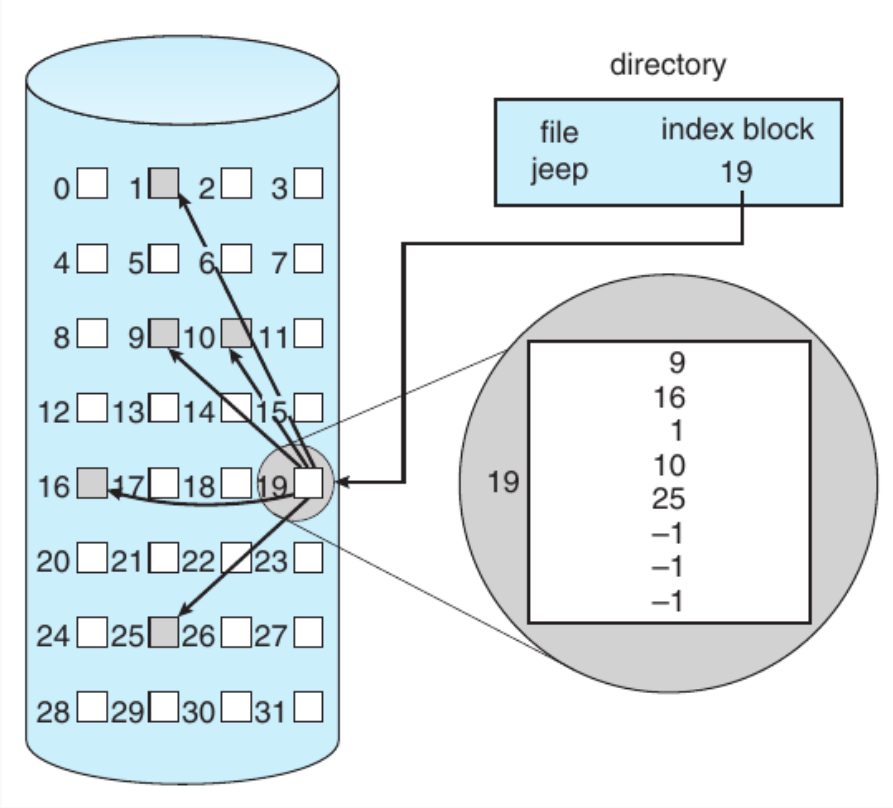
Asignación en forma de lista (Linked Allocation)

- 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 (Indexed Allocation)

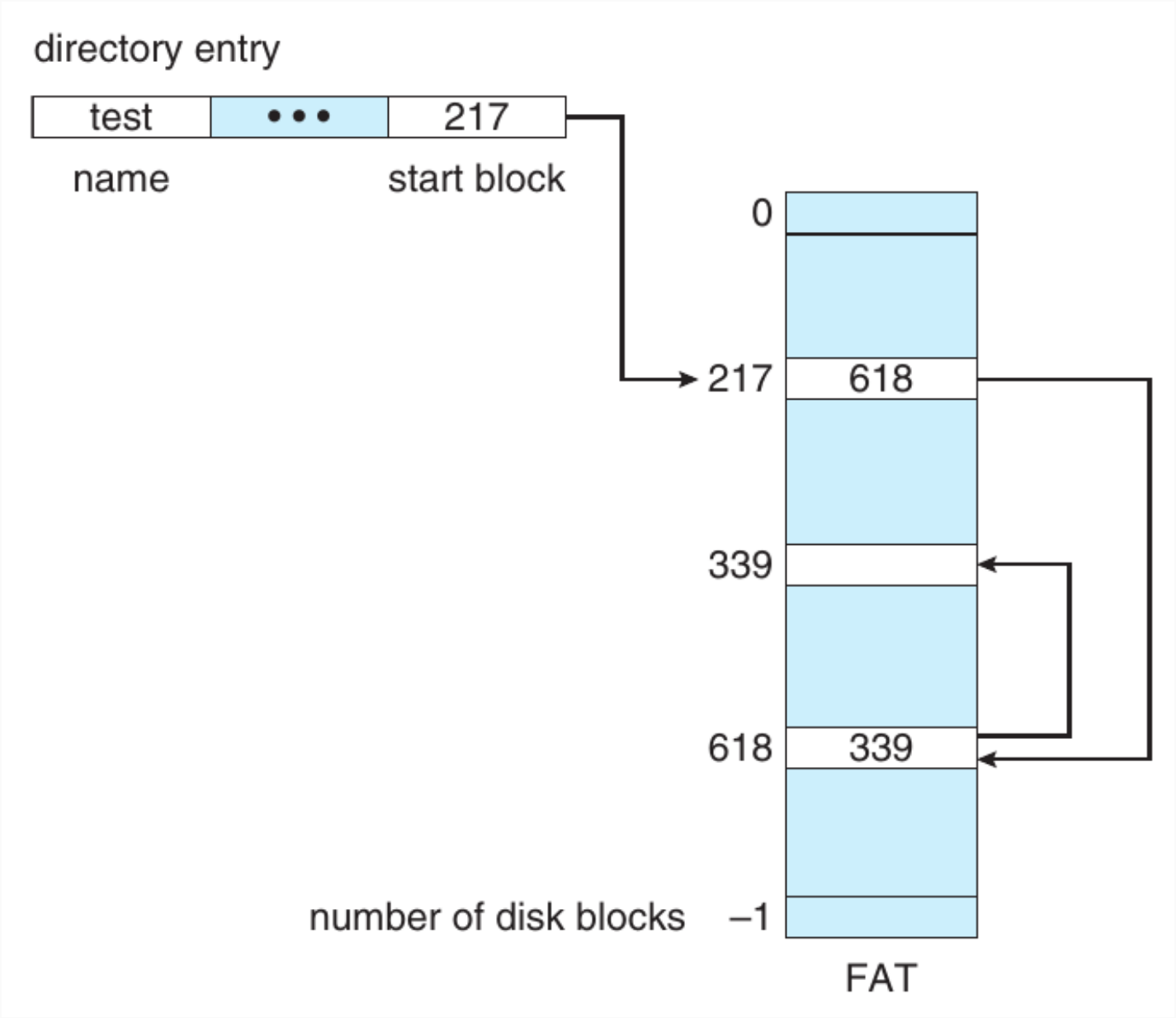
- Los bloques son accedidos directamente a través del bloque de indexación (*index block*).
- El bloque de indexación ocupa lugar por lo que se trata de que sea lo más pequeño posible.
- Una estrategia utilizada es la indexación en varios niveles. Algunos índices hacen referencia a bloques de datos (directos)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



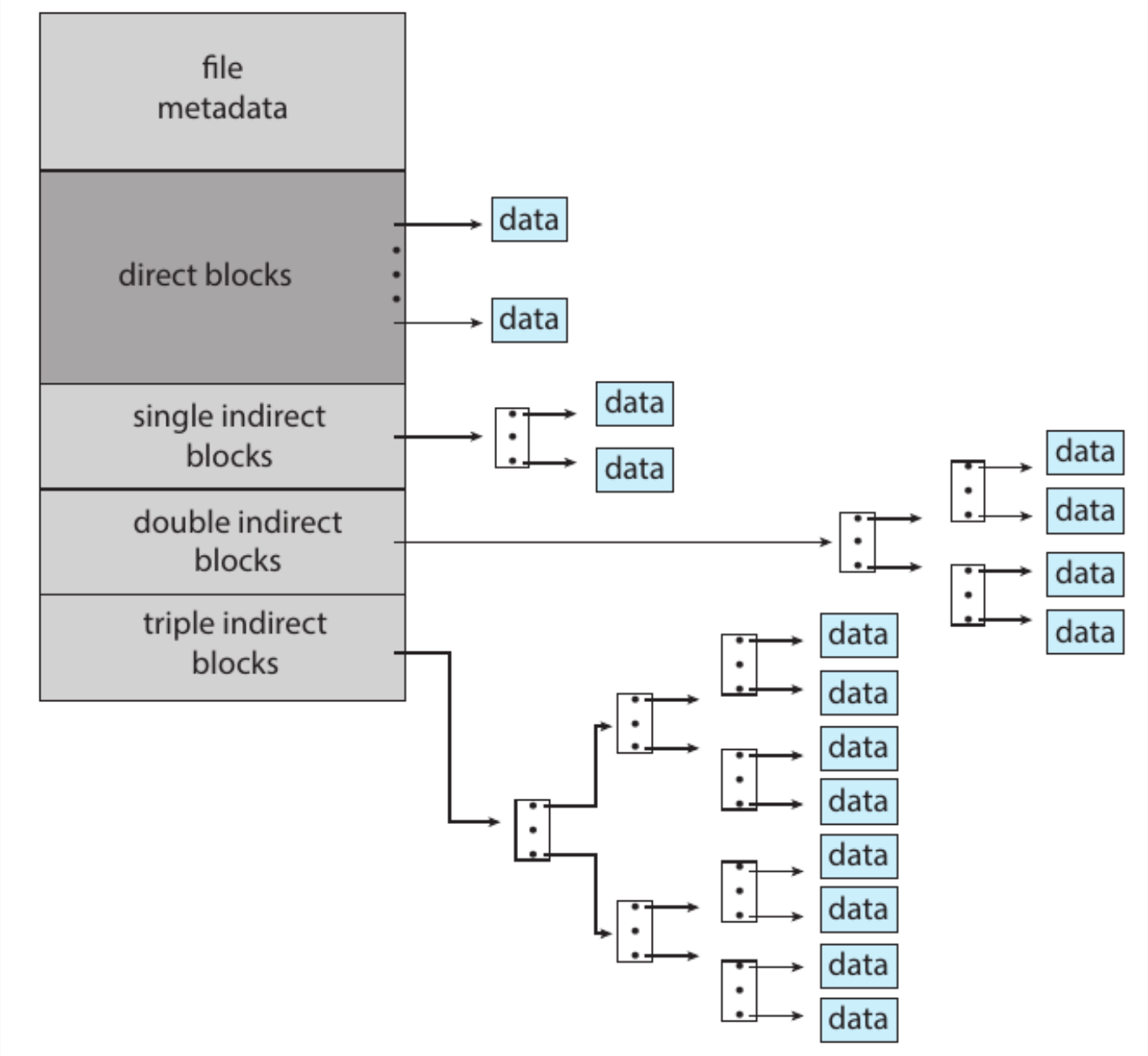
Asignación en forma de lista (Linked Allocation): Ejemplo FAT

- 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 (Linked Allocation): Ejemplo FAT



Asignación indexada (Indexed Allocation): 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 (*Bit Vector, Bit Map*): se dispone de un bit para cada bloque del dispositivo, que representa si está ocupado o libre.
  - Lista de bloques libres (*Linked list*): Se mantiene una lista encadenada con los bloques libres a través de los bloques. Es necesario una referencia al primer bloque.
  - Agrupación (*Grouping*): es una variación de la lista encadenada. En cada bloque de la lista se contiene un grupo de bloques libres.
  - Conteo (*Counting*): se mantiene una lista en donde cada bloque contiene información de cuantos bloques contiguos, a partir de él, están libres.

Sistema de archivos virtual

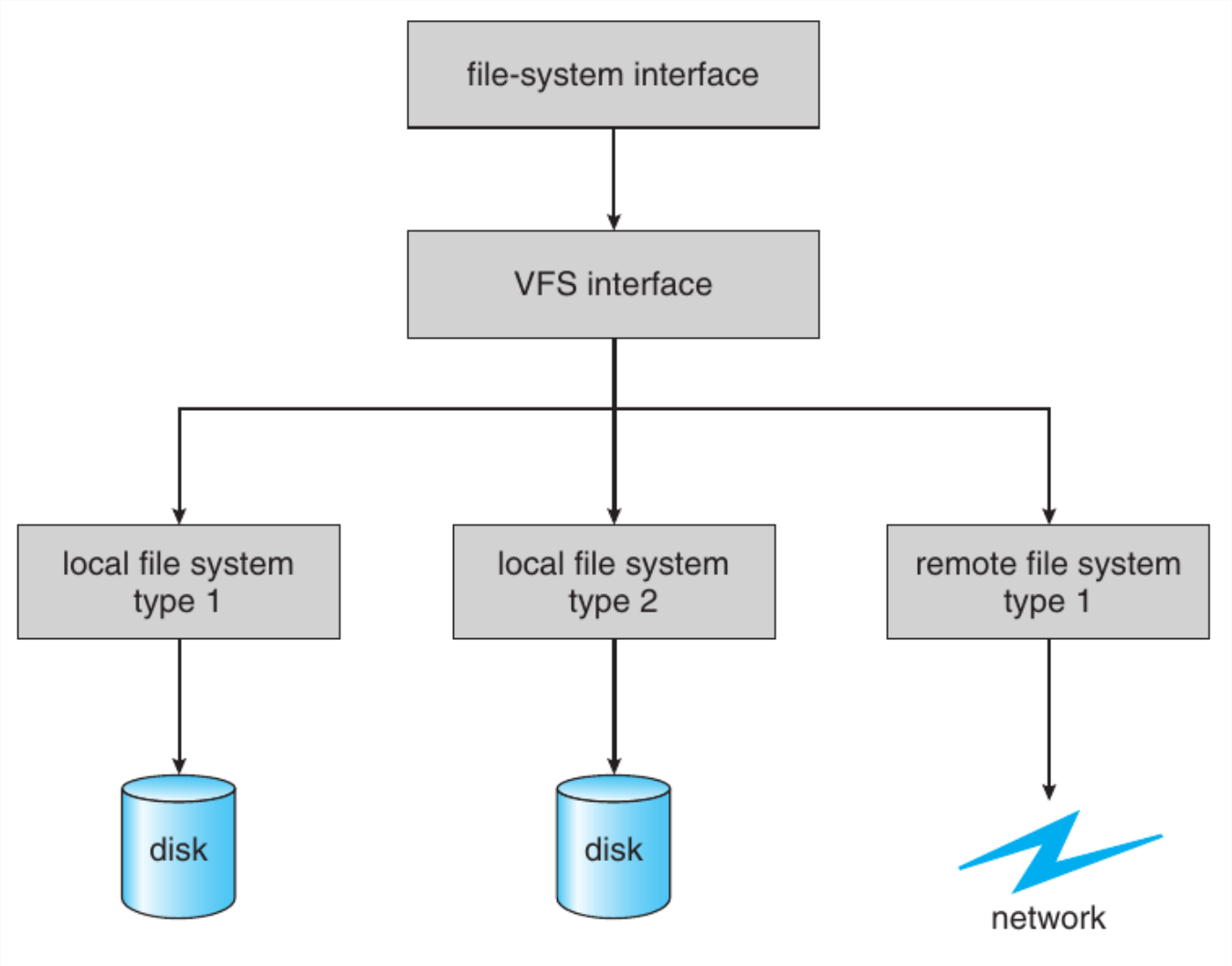
Sistema de archivos virtual

- 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



Ejemplo UNIX

Ejemplo UNIX

- Cada partición contiene un bloque descriptor del sistema de archivo llamado super-block.
- El super-block contiene:
  - Nombre del volúmen.
  - Cantidad máxima de archivos (inodos). Cantidad de archivos utilizados y libres.
  - Cantidad de bloques de datos, cantidad de bloques utilizados y libres.
  - Referencia a comienzo de bloques de datos, de indexación y de vector de bits.
  - Información de conteo.
  - etc.

Ejemplo UNIX

- La administración del espacio libre se realiza a través de mapa de bits (*bit vector*). Se disponen varios bloques al comienzo de la partición.
- El bloque de control de archivo es la estructura inode. Los inodos son identificados por un número, que es único a nivel del sistema de archivos. Los inodos poseen un tipo:
  - archivo común
  - directorio
  - enlace simbólico
  - pipe
  - socket
- Utiliza un método de asignación por indexación.

Ejemplo UNIX

- Los directorios son representados como un archivo (inodo), en donde los datos son entradas que tienen los nombres de los archivos y el número de inodo correspondiente.
- Si es un soft link, se tiene la ruta (*path*) del archivo al cual referencian.
- Los hard links son tratados en forma natural, ya que la pertenencia de un archivo a un directorio está en los datos del directorio y se referencia al número de inodo.

Ejemplo ejercicio FAT

```
const MAX_BLOQUES = 16777216; // 2 ^ 24
type bloque = array [0..4095] of byte;
type entradaDir = Record
    usado: bit;                          // (1 bit)
    nombre: array [0..22] of char;       // (23 bytes)
    inicio: integer;                     // (4 bytes)
    tipo: {ARCHIVO, DIRECTORIO};         // (1 bit)
    tam: integer;                        // (4 bytes)
    reservado: array[0..5] of bit;       // (6 bits)
end; // 32 bytes

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;

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

Ejemplo ejercicio Inodos

```
type block = array [0..511] of byte; // 512 bytes

type dir_entry = Record
        name : array [1..12] of char;  // 12 * 8 bits
        type : (file,dir);             // 1 bit
        used : boolean;                // 1 bit
        inode_num : int;             // 16 bits
        perms : array [1..14] of bit;  // 14 bits
    End;

type inode = Record
        inode_num : int;                 // 16 bits
        used      : boolean;                 // 1 bit
        data      : array [1..5] of int;// 5 * 16 bits
        tope      : 0..5;                    // 3 bits
        type      : (file, dir);             // 1 bit
        size      : int;                  // 16 bits
        reserved  : array[1..11] of bit;    // 11 bits
    End;

type inode_table = array [0..max_inode_on_disk] of inode;
type disk = array [0..max_blocks_on_disk] of block;
Var
  TI   : inode_table;
  D    : disk;
```