Sistemas de Ficheros

Sistemas Operativos

Grupo C – Grados Ing. Computadores, Informática y Software
Departamento de Arquitectura de Computadores
Universidad de Málaga

v.20220512

Sources:

Kubiatowicz ©2010
Walpole ©2010
Silberschatz, Galvin and Gagne ©2009

Contenido

- Concepto de fichero
  - Tipos de fichero, atributos, acceso y operaciones
- Dispositivos de almacenamiento masivo
  - Discos duros: mecanismos, geometría, formato, estructura y rendimiento
- Implementación de sistemas de ficheros
  - Directorios
  - Métodos de Asignación:
    - contigua
    - enlazada (FAT)
    - indexada (i-node)
  - Gestión del espacio libre
  - Eficiencia y rendimiento

Concepto de fichero

Concepto de archivo o fichero

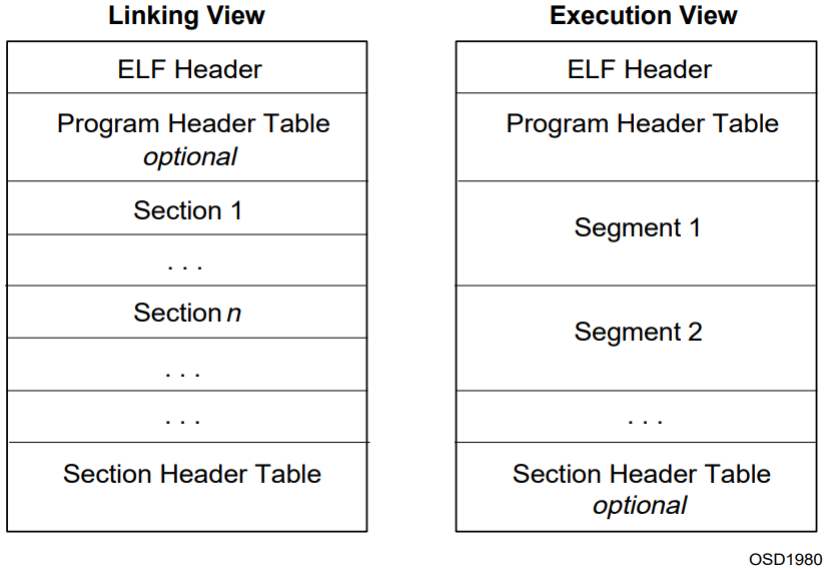
- El SO proporciona una visión lógica (abstracta) de la información almacenada en los dispositivos de almacenamiento secundario no volátil
- **Fichero**: espacio de direccionamiento lógico contiguo. Unidad lógica de almacenamiento secundario

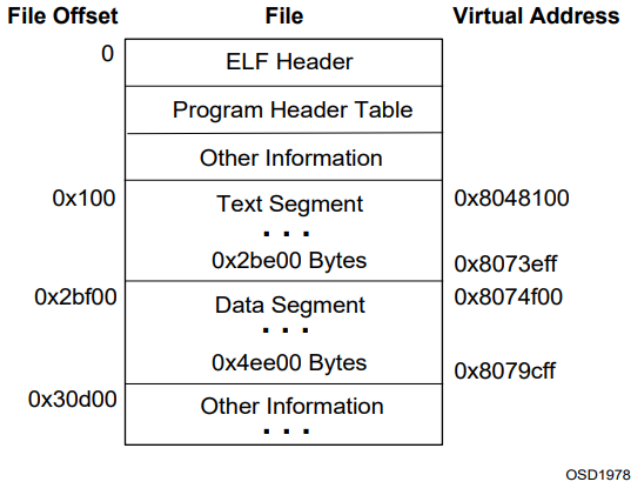
- Tipos de fichero:
  - De datos: caracteres o binarios
    - Los programas pueden estructurar los ficheros de datos como quieran
    - Los programas de usuario pueden imponer ciertas extensiones de archivos para indicar su tipo
    - Para el SO son sólo una secuencia de bytes
  - Ejecutables: entendibles por el SO

Extension	Meaning
file.bak	Backup file
file.c	C source program
file.gif	Compuserve Graphical Interchange Format image
file.hlp	Help file
file.html	World Wide Web HyperText Markup Language document
file.jpg	Still picture encoded with the JPEG standard
file.mp3	Music encoded in MPEG layer 3 audio format
file.mpg	Movie encoded with the MPEG standard
file.o	Object file (compiler output, not yet linked)
file.pdf	Portable Document Format file
file.ps	PostScript file
file.tex	Input for the TEX formatting program
file.txt	General text file
file.zip	Compressed archive

Concepto de archivo o fichero

- Ejecutables:
  - El SO (el cargador) debe conocer el formato de los archivos ejecutables
  - El **cargador** (syscall `exec()` en sistemas Unix) pone el programa y los datos en el espacio de direcciones del proceso y lo carga en memoria, valida permisos, copia los parámetros de entrada, …
  - Ej., Linux: `readelf -a` …





Executable and Linking Format (ELF). *Fuente:* https://refspecs.linuxfoundation.org/elf/elf.pdf

Concepto de archivo o fichero

- Atributos: metadatos asociados a cada fichero
  - Los atributos de los ficheros se mantienen en la estructuras del sistema de fichero o en el directorio (también en disco) que lo contiene
    - Nombre: información en forma legible por un humano
    - Identificador: número que identifica unívocamente al fichero en el sistema de ficheros
    - Localización: puntero a la localización del archivo en el dispositivo
    - Tamaño: tamaño actual del fichero
    - …
  - Al menos debe incluir el **nombre** y el identificador

Attribute	Meaning
Protection	Who can access the file and in what way
Password	Password needed to access the file
Creator	ID of the person who created the file
Owner	Current owner
Read-only flag	0 for read/write; 1 for read only
Hidden flag	0 for normal; 1 for do not display in listings
System flag	0 for normal files; 1 for system file
Archive flag	0 for has been backed up; 1 for needs to be backed up
ASCII/binary flag	0 for ASCII file; 1 for binary file
Random access flag	0 for sequential access only; 1 for random access
Temporary flag	0 for normal; 1 for delete file on process exit
Lock flags	0 for unlocked; nonzero for locked
Record length	Number of bytes in a record
Key position	Offset of the key within each record
Key length	Number of bytes in the key field
Creation time	Date and time the file was created
Time of last access	Date and time the file was last accessed
Time of last change	Date and time the file has last changed
Current size	Number of bytes in the file
Maximum size	Number of bytes the file may grow to

Concepto de archivo o fichero

- Acceso a ficheros:
  - Los ficheros se componen de bloques físicos de tamaño fijo (fragmentación interna), que son la unidad de transferencia entre disco y memoria
  - El SO operativo, generalmente, ofrece syscalls para acceder a los ficheros a nivel de byte (el mapeo de byte a bloque es relativamente sencillo)
  - Acceso secuencial (el más común):
    - read_next(): lee la posición actual del fichero y avanza a la siguiente posición
    - write_next(): como read_next() pero escribe en la posición actual
    - No se puede saltar aleatoriamente a una posición dada, pero se puede ir hacia atrás o hacia delante sin leer ni escribir
    - Basado en un modelo de fichero de dispositivo de cinta magnética
  - Acceso directo o aleatorio:
    - read(n)/write(n): lee/escribe el bloque/byte n
    - Esencial en sistemas de bases de datos
    - Basado en un modelo de fichero de dispositivo de disco

Concepto de archivo o fichero

- Operaciones con ficheros:
  - Creación: implica encontrar espacio en el sistema de ficheros para el mismo y crear su entrada en el directorio.
  - Borrado: liberar recursos (en el caso de hard links solo la entrada del directorio)
  - Abrir/Cerrar: para no especificar su nombre, sus permisos,… en cada operación con fichero
  - Leer/Escribir: n bytes desde la posición actual
  - Posicionamiento (seek): reposicionarse en el fichero
  - Leer/escribir atributos
- Ejemplo: syscalls de Linux para operar con ficheros
  - `fd = open (name, mode)`
  - `byte_count = read (fd, buffer, buffer_size)`
  - `byte_count = write (fd, buffer, num_bytes)`
  - `close (fd)`

Dispositivos de almacenamiento

Almacenamiento masivo o secundario

- Datos que deben sobrevivir a la terminación de un proceso o apagado de la máquina: **PERSISTENCIA**
- Necesidad de:
  - almacenar documentos, códigos de las aplicaciones, copias de seguridad, …
  - intercambiar datos por parte de los usuarios
  - acceso a información por múltiples procesos que se ejecutan concurrentemente
  - soporte para la memoria virtual
- **Un archivo o fichero** no es más que una secuencia de bits almacenados en un dispositivo (memoria secundaria)

Almacenamiento secundario

- Características físicas:
  - **No volátil**
  - En comparación con la RAM:
    - Barato
    - Grande
    - Lento
  - Heterogéneo (cintas y discos magnéticos, discos ópticos, discos SSD, tarjetas extraíbles, …)



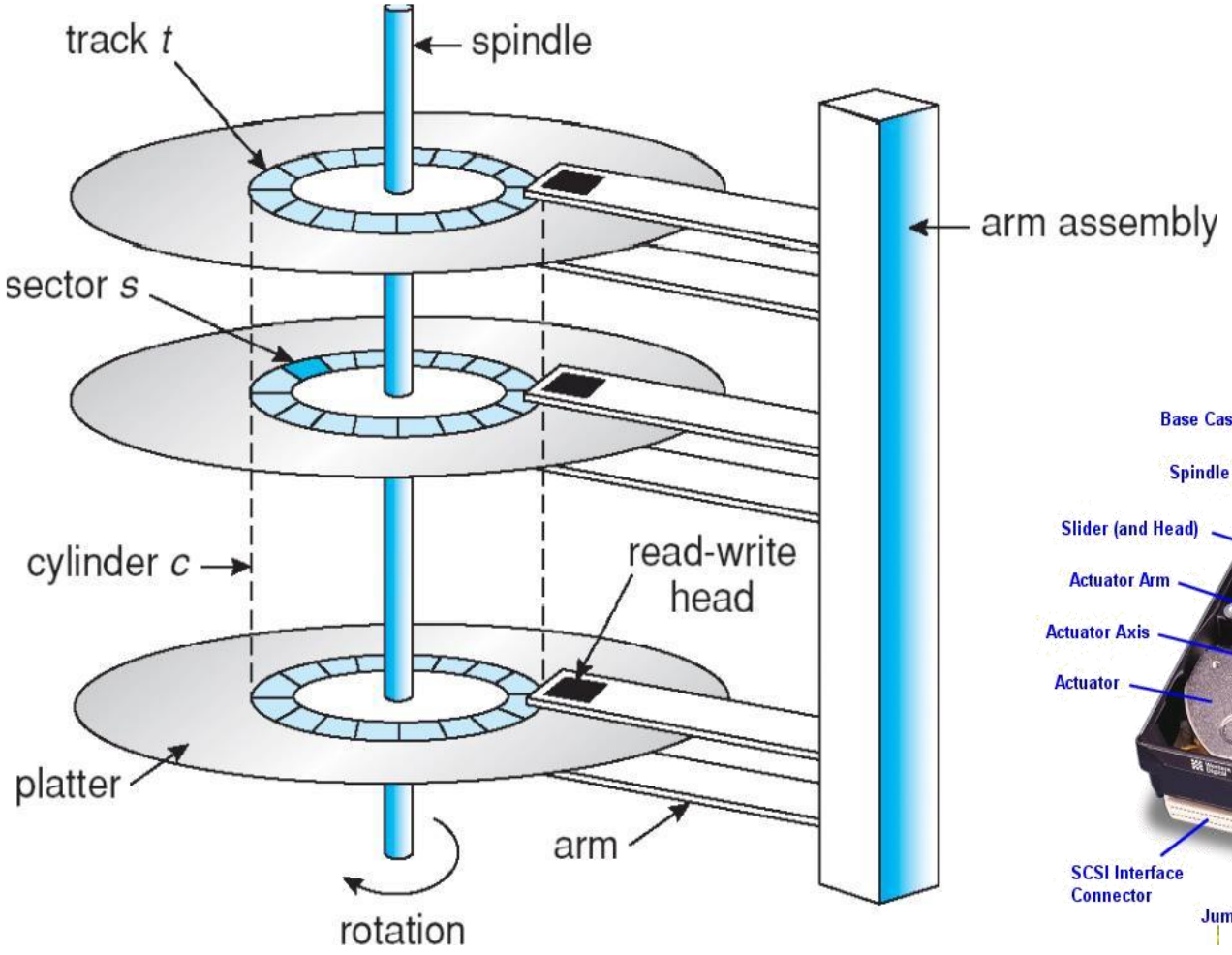

Almacenamiento secundario

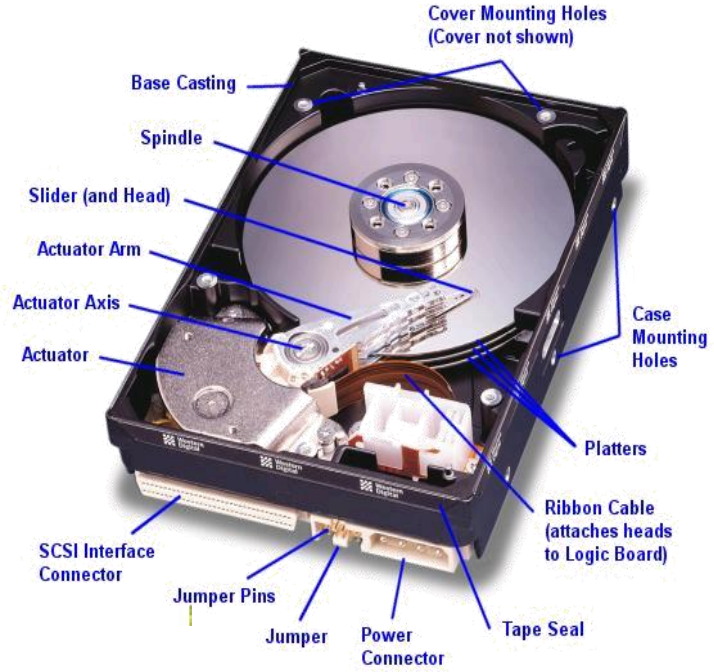
- **Cintas magnéticas**: utilizadas a principios de los 50 como dispositivos de almacenamiento secundario
  - Acceso secuencial lento (1000 veces más lento que un HDD)
  - En la actualidad usadas para backup, datos masivos accedidos con muy poca frecuencia y para transferir copias de seguridad de un sistema a otro
- **Discos magnéticos** (Hard Disk Drive – HDD)

  - Discos magnéticos con cabezas lectoras/escritoras
  - Acceso directo o aleatorio por bloques (sectores)
  - Su naturaleza mecánica los hace sensibles a golpes: la cabeza lectora puede hacer contacto con el disco rayando su superficie (head crash)
- **Discos de estado sólido** (Solid State Disk – SDD)
  - Tecnología Flash NVM (Non-Volatile Memory): transistores que atrapan carga
  - Acceso directo o aleatorio, rápido (pueden ser 100 veces más rápidos que los HDD)
  - Al no tener partes móviles (chips de transistores) son más fiables

- El dispositivo de almacenamiento se conecta al computador vía buses de E/S
  - Los buses varían a lo largo de la historia: **EIDE, ATA, SATA, USB, Fibre Channel, SCSI, …**
  - El controlador de E/S del host se comunica, vía bus, con el del disco (los discos suelen incorporar su controlador)

HDD: Geometría

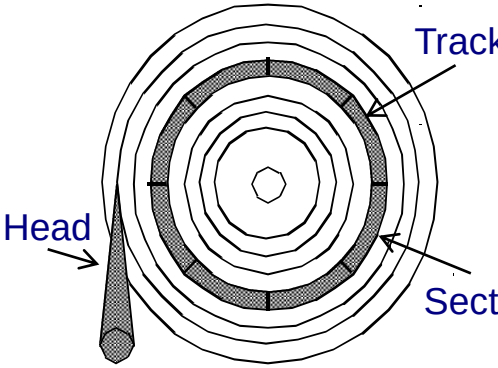




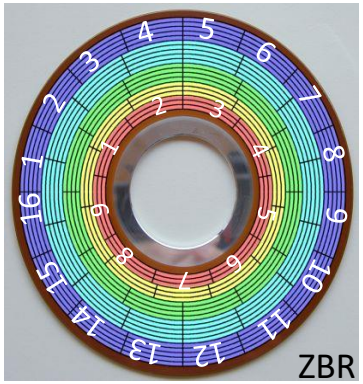
Western Digital Drive
http://www.storagereview.com/guide/

HDD: Geometría

- Propiedades:
  - La cabeza se mueve hasta localizar una pista ***(track)*** de información
  - La unidad básica de acceso es el **sector**
    - El sistema operativo trabaja con una unidad de transferencia que es múltiplo del sector: el **bloque** o ***cluster***
  - Todas las pistas de los diferentes platos que se pueden acceder sin mover la cabeza forman el **cilindro**



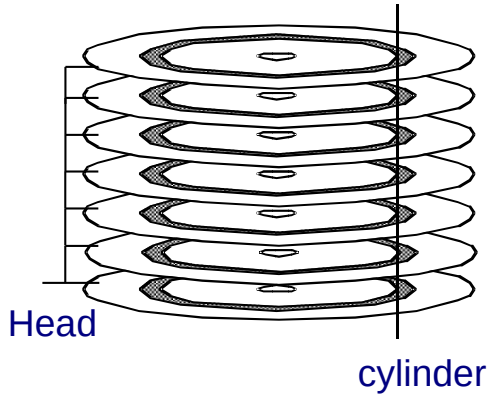
- *Zone bit recording* (ZBR):
  - La velocidad de giro del disco es constante
  - Como la densidad de bits por área es constante: más sectores en las pistas externa
  - La velocidad de lectura mayor en pistas externas



- Algunos números típicos:

  500 - 20,000 pistas por superficie

  32 - 800 sectores por pista



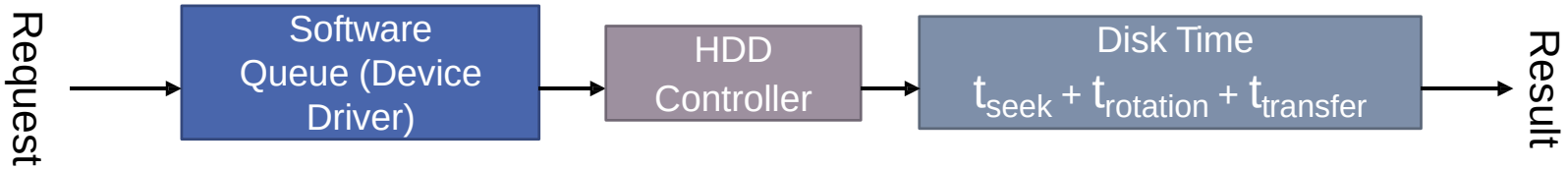
HDD: Formato físico

- Un disco magnético no es más que un conjunto de platos con una superficie continua donde se puede leer y escribir bits
- Se define el sector como la unidad mínima de transferencia gestionada por el controlador
- Un sector tiene la siguiente estructura:

- El preámbulo y la cola contienen metadatos usados por el controlador del disco, como el número de sector y un código de corrección de error (ECC)
- El fabricante elige el tamaño del sector (valores típicos de 512B a 4KB): se necesitan 100-1000 *bits* entre sectores para permitir al controlador medir la velocidad del disco y tolerar pequeños cambios (debidos a temperatura) en la longitud de la pista
  - Si el sector fuese de 1B: eficiencia espacial de sólo un 1% del disco (1% de espacio útil para almacenar datos); Si el sector fuese de 1KB: eficiencia espacial del 90%; Si el sector fuese de 1MB: eficiencia espacial de casi el 100% (pero aumenta la fragmentación interna).

HDD: Modelo de rendimiento

- Secuencia de lectura/escritura de disco:
  - Si el dispositivo está ocupado, la petición de E/S se encola: se pueden aplicar algoritmos de planificación de peticiones en la cola para optimizar el acceso a disco (ej. ordenarlas en orden creciente de cilindro)
  - El controlador de disco (hw) recibe la petición y realiza el acceso:
    - **Tiempo de posicionamiento**:
      - Posicionar las cabezas lectoras sobre el cilindro adecuado (*seek time*)
      - Espera el giro del disco hasta que el sector requerido se posicione bajo la cabeza lectora (*rotational latency*)
    - **Velocidad de transferencia:** bits por segundo a los que realiza la transferencia del sector bajo la cabeza (*transfer rate*)
  - Máximo ancho de banda: bloques consecutivos de la misma pista (mayor en las más externas)



Latencia de acceso a disco = $t_{queue} + t_{controller} + t_{seek} + t_{rotation} + t_{transfer}$

HDD: Modelo de rendimiento

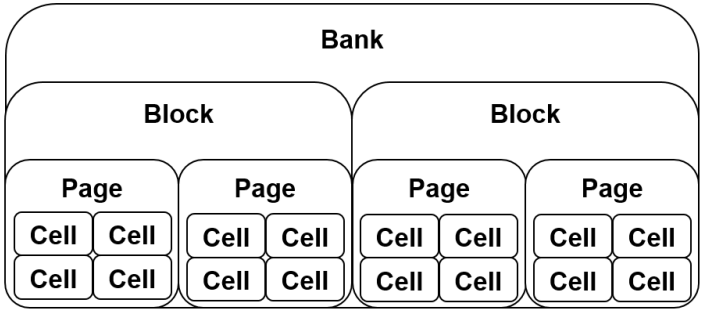
- Ejemplo de cálculo de latencia
  - No consideramos ni tiempo en cola ni del controlador
  - Tiempo medio de seek 5ms, retardo medio rotacional 4 ms
  - Tasa de transferencia de 4 MByte/s, para un sector de tamaño 1 Kbyte
- Acceso aleatorio en el disco
  - Tseek (5 ms) + Trot (4 ms) + Ttrans (0.25 ms)
  - Aproximadamente 10 ms leer/escribir los datos: 100 Kbytes/s
- Acceso aleatorio en el mismo cilindro
  - Tseek (0 ms) + Trot (4 ms) + Ttrans (0.25 ms)
  - Aproximadamente 5 ms leer/escribir los datos: 200 Kbytes/s
- Siguiente sector de la misma pista
  - Tseek (0 ms) + Trot (0 ms) + Ttrans (0.25 ms)
  - Aproximadamente 0.25 ms leer/escribir los datos: 4 Mbytes/s
- Aunque el HDD permite acceso directo/aleatorio se ve beneficiado del acceso secuencial a los ficheros (minimizar el tiempo de búsqueda, t_{Seek}, y rotación, $t_{Rotation}$, es importante)

Solid State Disk (SSD)

- ~2010 – Popularización de la memoria flash
  - Se codifican los bits mediante electrones atrapados en una celda
    - Single-level cell (SLC)
      - Un bit simple almacenado en un transistor
      - Más rápido, más duradera (50k -100k escrituras posibles)
    - Multi-level cell (MLC)
      - Se codifican dos ó mas bits con diferentes niveles de voltaje
      - Vida más limitada (1k-10k escrituras)

- Ventajas:
  - Más robustos que los HDDs: no tienen partes móviles (no sufren de *headcrash*)
  - Latencia de acceso más baja: no hay tiempo de posicionado de cabeza lectora, ni tiempo de rotación
  - Tiempo de acceso aleatorio uniforme (para lecturas, no para escrituras)
  - Consumen menos energía
- Desventajas:
  - El coste por byte es mayor que en HDD
  - La capacidad total es menor (aunque capacidad y coste mejoran día a día)
  - El tiempo de acceso depende de si se lee (más rápido) o se escribe (más lento)
  - El número de escrituras está limitado por la tecnología Flash

SSD: características



Fuente: Operating Systems: Storage Devices, Crooks & Joseph http://inst.eecs.Berkeley.edu/~cs162

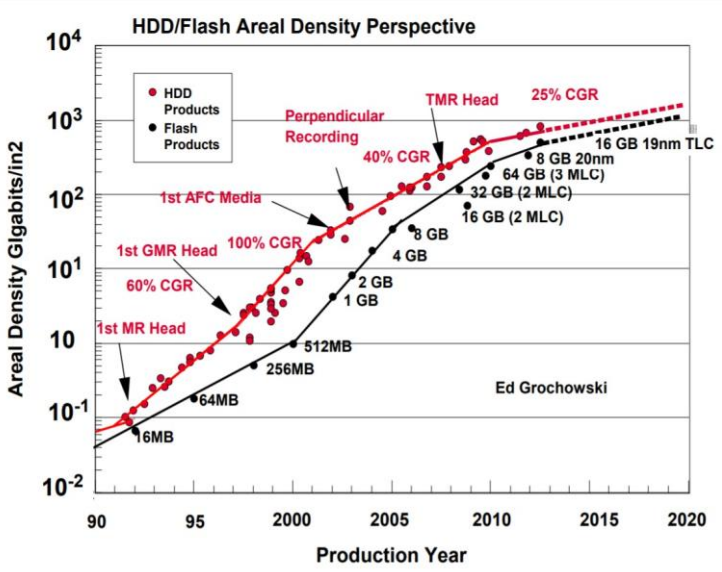
- Las celdas de los chips de memoria *flash* se organizan **páginas** y éstas en **bloques**
- La **unidad de lectura/escritura es la página** (equivalentes a los sectores en HDDs, 4KB-16KB)
- Las páginas no se pueden sobreescribir, hay que borrarlas primero (aplicarles tensión de borrado) y escribirlas (*programarlas*, con otro voltaje)
- El **borrado se realiza por bloque** que consta de múltiples páginas
- **El número de borrados por bloque es limitado** (~100k borrados): después la puerta Flash pierde su capacidad de retención
- El tiempo de vida del dispositivo se mide en escrituras del SSD entero por día (*Drive Writes Per Day* – DWPD)
- Ej. En un disco Flash NAND de 1TB con 5DWPD se pueden escribir 5TB por día durante el período de garantía (3/5 años)

SSD: controlador

- El controlador del SSD descarga al SO de las complicaciones de la escritura de páginas
- Escritura:
  - En página de bloque borrado: se escribe la página entera
  - En página ya escrita: se marca como inválida y se escribe en otra página borrada
  - Si no hay bloques borrados (memoria llena):
    - *Garbage collection*: el controlador puede reubicar páginas válidas para poder borrar un bloque (*write amplification*: una escritura puede producir muchas más)
    - *Overprovisioning*: hay ~20% de páginas ocultas para realizar este proceso
  - *Wear leveling*: el controlador cuenta el número de borrados de cada bloque e implementa un algoritmo para distribuir los borrados uniformemente
- Para llevar a cabo las **reubicaciones** el controlador mantiene una capa de traducción de bloque lógico a página física (*Flash Translation Layer* – FTL)

HDD, SSD: números

- HDDs: números
  - Posicionamiento: t_{seek} entre 3 ms - 12 ms
  - Rotación (RPM): 5400, 7200, 10000, 15000 rev/min
    - Rotation = 1/RPM min/rev * 60 s/min = 60/RPM s/rev → 11ms a 5400 rpm y 4 ms a 15000 rpm
    - $t_{Rotation}$ medio, para acceder a un dato que está en mitad del disco: ½ $t_{Rotation}$ → 5.5 ms a 5400 rpm y 2 ms a 15000 rpm
  - Tiempo medio de acceso: ~10 ms
  - *Endurance*: ~10^{16} ciclos de escritura por bit
- SSDs: números
  - Tiempo de acceso para lectura: < 0.1 ms (100x más rápido)
  - Tiempo de acceso para escritura: ~2 ms
  - Tiempo de borrado: ~5 ms
  - *Endurance*: ~10^{5} ciclos de escritura por bit



- Densidad de bits: superado el Terabit por pulgada cuadrada (>1Tb/in^2)

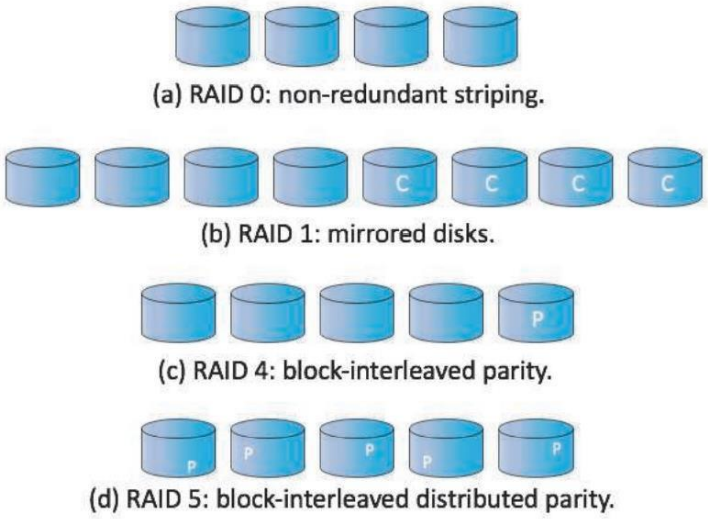
Implementación del sistema de ficheros

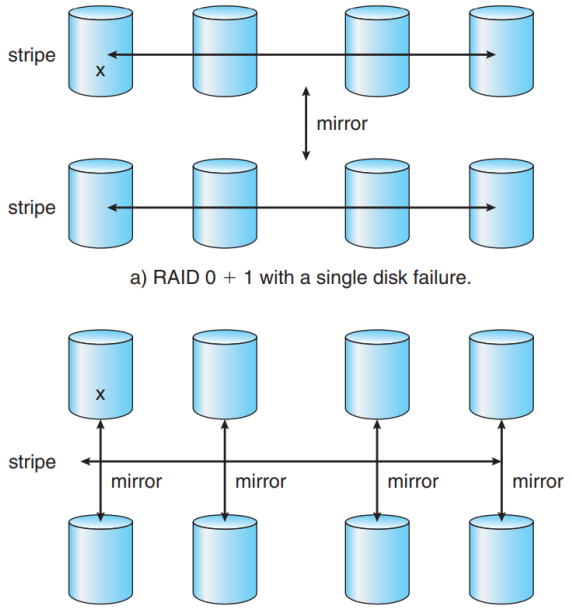
Organización del disco

- Los discos se subdividen en **particiones**
- Un disco o partición se puede usar:
  - ***Raw***: sin sistema de ficheros
  - **Formateado** con un sistema de ficheros (*filesystem*):
    - Unix file system (UFS)
    - Linux: *extended file system* (ext2, ext3, ext4) y otros (XFS, …)
    - Microsoft: FAT (FAT12, FAT16, vFAT, FAT32, exFAT), NTFS (Windows NT filesystem)
- La entidad que alberga un sistema de ficheros (disco o partición) se conoce como **volumen**
- Además de sistemas de ficheros de propósito general, hay sistemas de propósito específico (ejemplo /proc) coexistiendo en el mismo sistema operativo
- Los discos o particiones pueden organizarse en **RAIDs** (*Redundant Array of Independent Disks*) para mejorar prestaciones

Organización de discos

- Los discos se pueden agrupar en RAIDs que introducen redundancia (RAID = *Redundant Array of Independent Disks*)
- Con ello se puede mejorar:
  - La tolerancia a fallos con redundancia (RAID 1, RAID 4, RAID 5)
  - El rendimiento con paralelismo (RAID 0)



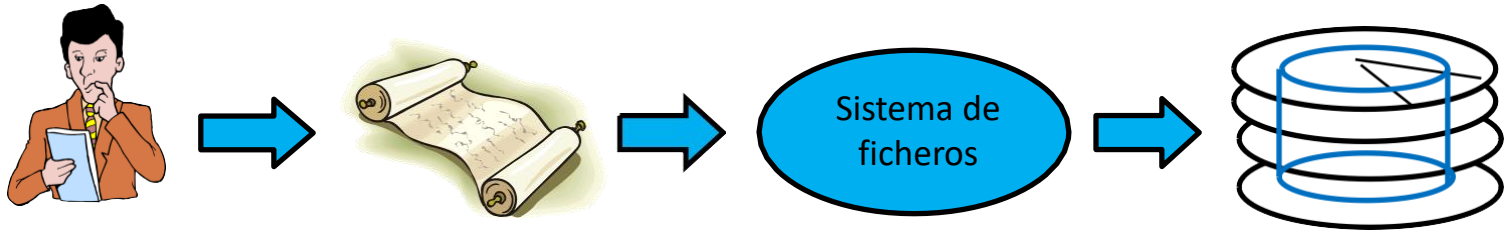


Direccionamiento y sistemas de ficheros

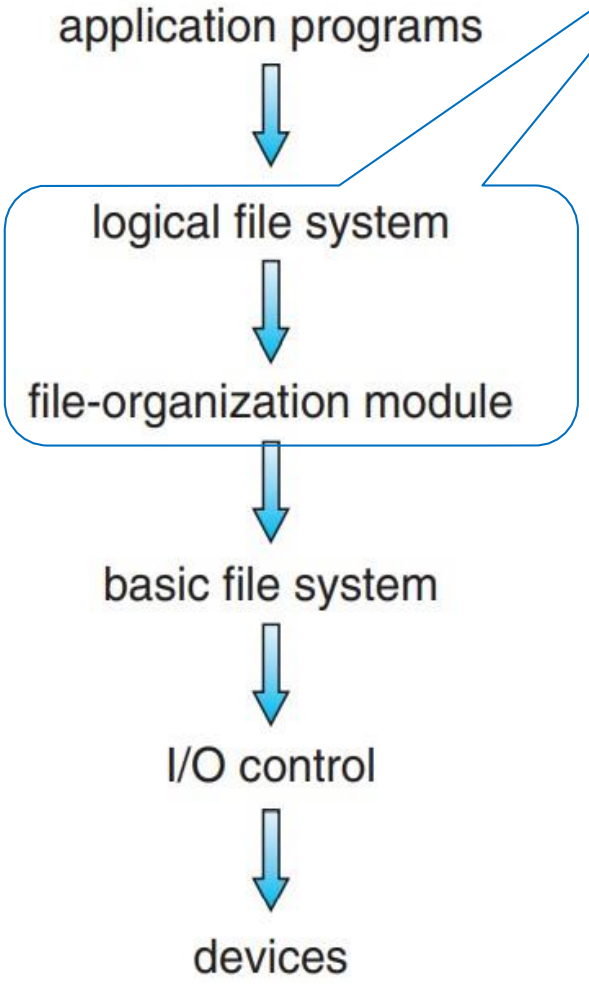
- *Logical Block Addressing* (**LBA**): El SO ve el disco como un *array unidimensional* de bloques lógicos, cada bloque tiene una dirección, desde 0 hasta el número de bloques en el disco/partición
- Cada **LBA** solicitada al dispositivo, se traduce en**:**
  - **HDD**:
    - sector, cilindro, cabeza (el bloque 0 suele ser el 1er sector de la 1ª pista del cilindro exterior)
  - **SDD**:
    - chip (banco), bloque, página (gestionado por la capa FTL)
- **Sistemas de ficheros**: capa del SO que abstrae la interfaz de dispositivos de bloques, como los discos duros, de manera que el usuario maneja **ficheros** y **directorios**
- La **unidad de asignación** del sistema de ficheros es el **bloque lógico** o **clúster** (uno o más sectores consecutivos (HDD) o páginas (SSD))

Implementación de sistemas de ficheros

- Traducción de la perspectiva de usuario a la del SO:
  - Si el usuario pide los bytes del 2 al 12:
    - El SO tiene que encontrar el bloque correspondiente a esos bytes en el dispositivo y devolver sólo los bytes pedidos
  - Si el usuario quiere escribir esos bytes:
    - El SO tiene que encontrar el bloque en disco y traerlo a memoria
    - Modificar los bytes correspondientes y escribir el bloque entero en el dispositivo
  - El sistema de ficheros trabaja a nivel de bloque:
    - Ej. Las funciones `getc()` y `putc()` de C leen y escriben un carácter de fichero, respectivamente → el SO accederá a un bloque entero y lo mantendrá en un búfer
  - En adelante, un fichero es una colección de bloques



Sistema de ficheros en capas

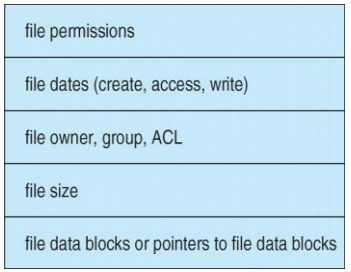


Diferentes para cada sistema de ficheros

- Un sistema de ficheros está, generalmente, dividido en diferentes **capas**:
  - Se encarga del nombrado, de manejar los metadatos (FCB: File Control Block) y de la protección de los ficheros
  - Maneja ficheros a nivel de bloque lógico (0 a N), LBA, y maneja el espacio libre (bloques no asignados)
  - Lanza comandos genéricos al dispositivo apropiado y maneja cachés y búferes para optimizar el rendimiento (*ej.* `retrieve block 123`)
  - Drivers de dispositivos y RTIs. Se comunica con los controladores de los dispositivos (*ej.* `read drive1, cylinder 72, track 2, sector 10, into mem location 1060`)

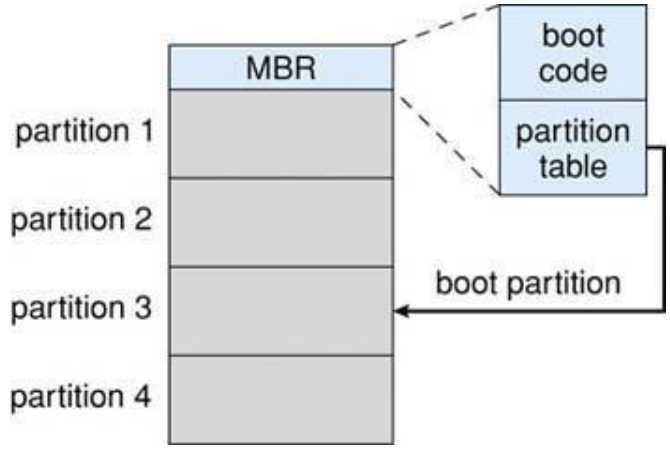
Implementación de sistemas de ficheros

- Estructuras almacenadas en el disco para la implementación de sistemas de ficheros:
  - ***Boot control block* (por volumen):** contiene información necesaria para arrancar un SO desde un volumen. El bloque puede estar vacío si no tiene un SO instalado:
    - En UFS (Unix File System) se llama Boot Block
    - En NTFS se llama Partition Boot Sector
    - En FAT se llama Master Boot Record (MBR)
  - ***Volume control block* (por volumen):** metadatos del volumen como el número de bloques, su tamaño, información de **los bloques libres**, punteros a los FCB, …
    - En UFS se llama superblock
    - En NTFS se llama Master File Table (MFT)
  - **Estructura de directorios (por sistema de ficheros):** usada para organizar los ficheros
  - ***File Control Block* (FCB) (por fichero):** contiene metadatos del fichero. Incluye un identificador único para su asociación a un directorio
    - En UFS se llama i-node
    - En NTFS se almacena en la MFT del volumen
    - En FAT los metadatos están en la entrada del directorio



Implementación de sistemas de ficheros

- Ejemplo (disco de PC, *legacy* BIOS):
  - *Sector 0: Master Boot Record (MBR)*
    - Contiene el código de arranque y la tabla de particiones
  - El resto del disco está divido en particiones
    - Partición: secuencia de sectores consecutivos
  - Cada partición puede empezar con un *boot control block*
    - Contiene un pequeño programa de arranque que lee el SO del sistema de archivos en esa partición
  - Inicio del SO:
    - La BIOS trae a memoria el MBR y lo ejecuta
    - A su vez, el código del MBR trae a memoria el *boot control block* de la partición de inicio que tendrá el cargador del SO



Implementación de sistemas de ficheros

- Estructuras almacenadas en memoria para la implementación de sistemas de ficheros:
  - Sirven para gestionar y agilizar (caching) los sistemas de ficheros
  - Estas estructuras se reservan cuando se monta el sistema de ficheros, se actualizan durante las operaciones sobre ficheros y se desechan al desmontarlo
    - En Linux: comandos `mount/umount`
  - Incluyen:
    - ***Mount table***: contiene información de los volúmenes montados
    - ***Directory structure cache***: mantiene entradas de directorios accedidos recientemente
    - ***System-wide open-file table***: contiene una copia de los FCBs de ficheros abiertos
    - ***Per-process open-file table***: contiene un puntero a la system-wide open-file table por cada fichero abierto de cada proceso
    - ***Buffers (buffer cache)*** que mantienen en memoria bloques de ficheros que están siendo leídos o escritos
      - En Linux: `free -h` (campo *buffer*/*cache*)

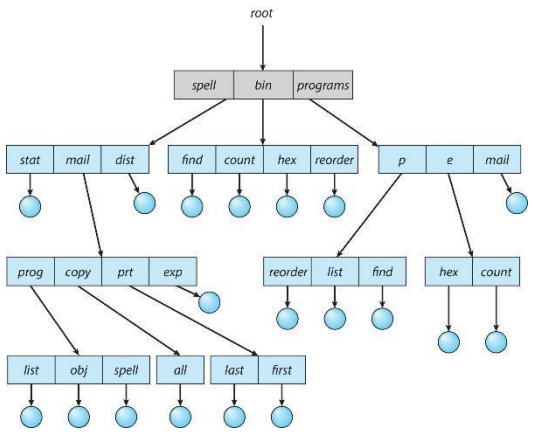
Estructura de directorios

- Los ficheros se organizan lógicamente en **directorios** (físicamente pueden estar en cualquier parte del disco) para obtener:
  - Eficiencia en la búsqueda de archivos
  - Nombrado conveniente para el usuario:
    - Dos usuarios pueden tener el mismo nombre para un archivo
    - El mismo archivo puede tener diferentes nombres (ej. hard links en Linux ext)
  - Agrupamiento: se pueden agrupar (lógicamente) archivos con las mismas propiedades (ej. todos los ejecutables, todos los juegos, …)
- Se suelen organizar como una estructura jerárquica (antes de los 70 no lo era)
- Los directorios son ficheros que se tratan de manera especial
- Sus entradas pueden ser ficheros u otros directorios (subdirectorios)
- Se modifican con syscalls: ej. `mkdir`, `rmdir`

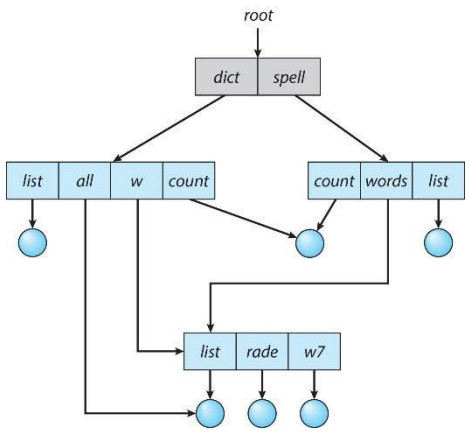
Estructura de directorios

- La estructura de directorios puede ser un a) árbol, un b) grafo acíclico, o incluso (c) un grafo general con ciclos (difícil de manejar)
- Grafo acíclico: más flexible, permite compartir subdirectorios y ficheros entre usuarios
  - Se puede implementar sobre una estructura árbol con *hard* y *soft links*
  - *Hard links*: nombres diferentes para el mismo fichero (múltiples entradas de directorios apuntan al mismo archivo)
  - *Soft links*: acceso directo (shortcut) que apunta a otro fichero/directorio

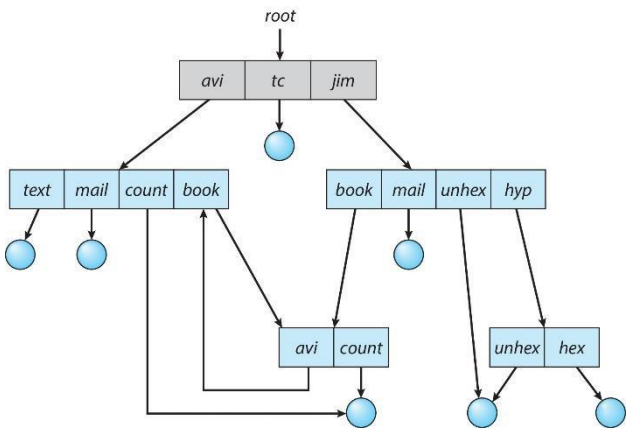
Fuente: Silberschatz, Global Edition



a) Árbol



b) Grafo acíclico



c) Grafo general

Estructura de directorios

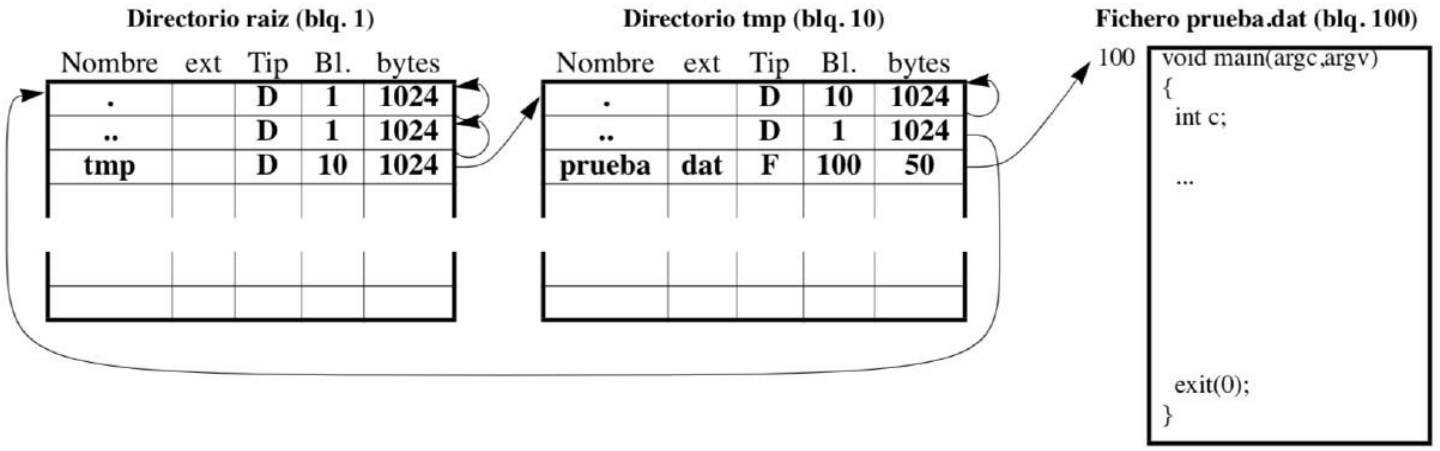
- Estructura de directorios: implementación
  - Lista de ficheros/subdirectorios:
    - Cada entrada de la lista contiene el nombre del fichero/subdirectorio y un puntero a los bloques de datos que lo componen (o puntero a metadatos)
    - Simple de programar
    - Ejecución lenta: ej. En la creación de un fichero hay que buscar el directorio, asegurarse de que no existe otro fichero con el mismo nombre y añadir una nueva entrada a la lista
    - Se puede acelerar con una lista enlazada o un árbol ordenando alfabéticamente las entradas
  - Tabla hash:
    - Se mantiene una lista de ficheros/subdirectorios para cada directorio y una estructura hash
    - Para buscar un fichero se codifica su nombre con una función hash que devuelve un número con el que se indexa la lista
    - Si hay una colisión (dos nombres dan el mismo hash) hay que resolverla
    - Funciona si las entradas son de tamaño fijo

Estructura de directorios

- Ejemplo: estructura **directorio FAT** (MS-DOS)
- Lista con nombre + atributos de los ficheros/subdirectorios
- Dos entradas especiales . y .. para el directorio actual y su padre

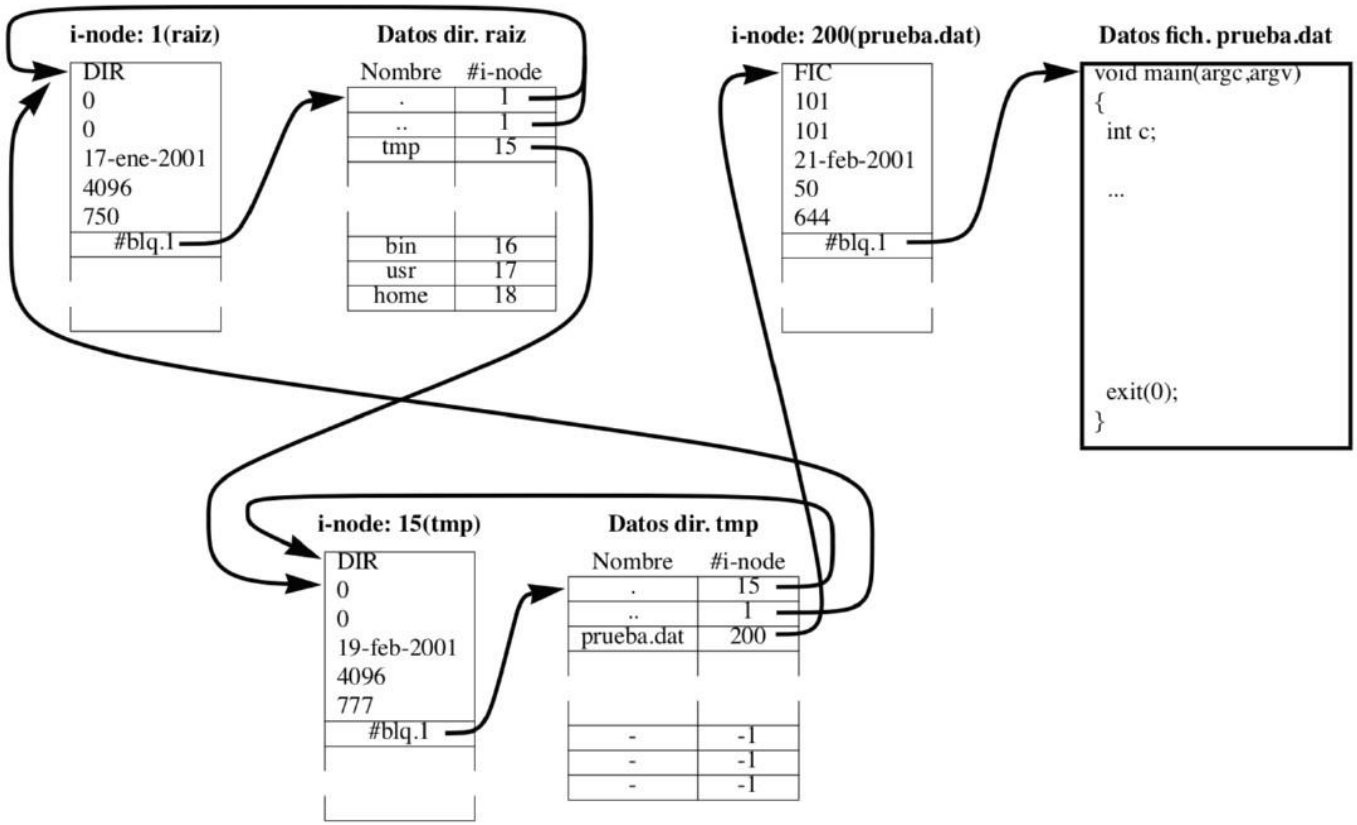
8	3	1	10	2	2	2	4
Nombre fichero	Extensión			Hora	Fecha	Primer bloque	Longitud
.				17:55	01/02/99	7300	512
..				17:55	01/02/99	40	512
Start	bat			17:58	01/02/99	7301	1034

- Ejemplo de archivo `C:\tmp\prueba.dat`:



Estructura de directorios

- Ejemplo: estructura **directorio UFS (Unix)**
  - Cada directorio es una lista con el nombre de fichero, un puntero al FCB (el denominado i-node), y los atributos
- Dos entradas especiales . y .. para el directorio actual y su padre
- Ejemplo `/tmp/prueba.dat`:



Estructura de directorios

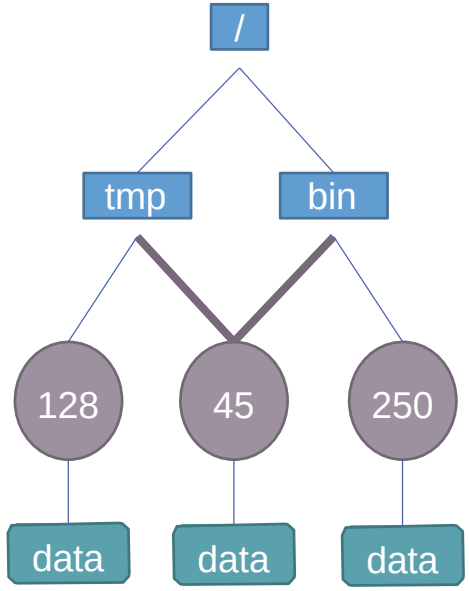
- ***Hard link***: crea entrada en el directorio apuntando al mismo i-node (incrementa el campo link count del i-node); borrar el último hard link borra el fichero físico; equivale a un fichero con múltiples nombres

Directorio /tmp

Nombre	*i-nod*
“data.tmp”	128
“hola.c”	45

Directorio /bin

Nombre	*i-nod*
“holamun.c”	45
“index.html”	250



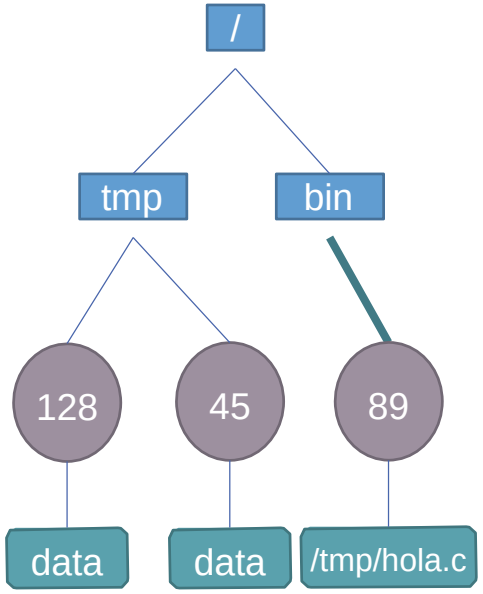
- ***Soft link***: crea entrada en el directorio; crea archivo especial con la ruta al fichero/directorio; el link puede tener una ruta inválida; borrar un soft link no puede borrar nunca un fichero físico

Directorio /tmp

Nombre	*i-nod*
“data.tmp”	128
“hola.c”	45

Directorio /bin

Nombre	*i-nod*
“holamun.c”	89

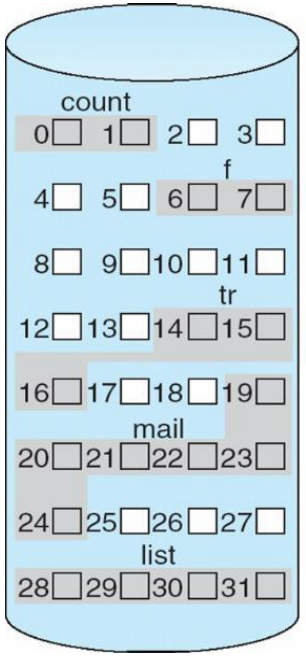


Sistemas de ficheros: asignación

- Objetivos:
  - Asignar bloques a ficheros de manera que se maximice el rendimiento en el acceso secuencial (el más común)
  - Facilitar el acceso aleatorio/directo
  - Facilitar las operaciones básicas con ficheros: borrado, creación, añadir/quitar datos, …
- Métodos:
  - Asignación **contigua**
  - Asignación **enlazada**
  - Asignación **indexada**

Sistemas de ficheros: asignación contigua

- Asignación contigua:
  - Cada fichero ocupa un **conjunto contiguo de bloques** en el disco
  - Un fichero se puede definir por la dirección del **primer bloque** y su **tamaño** en número de bloques
  - El acceso secuencial es rápido (en HDDs se minimiza t_{seek} y $t_{rotation}$)
  - El acceso aleatorio/directo es sencillo
  - El espacio libre se puede manejar con un vector de bits y una política de asignación *best fit* o *first fit*



directory

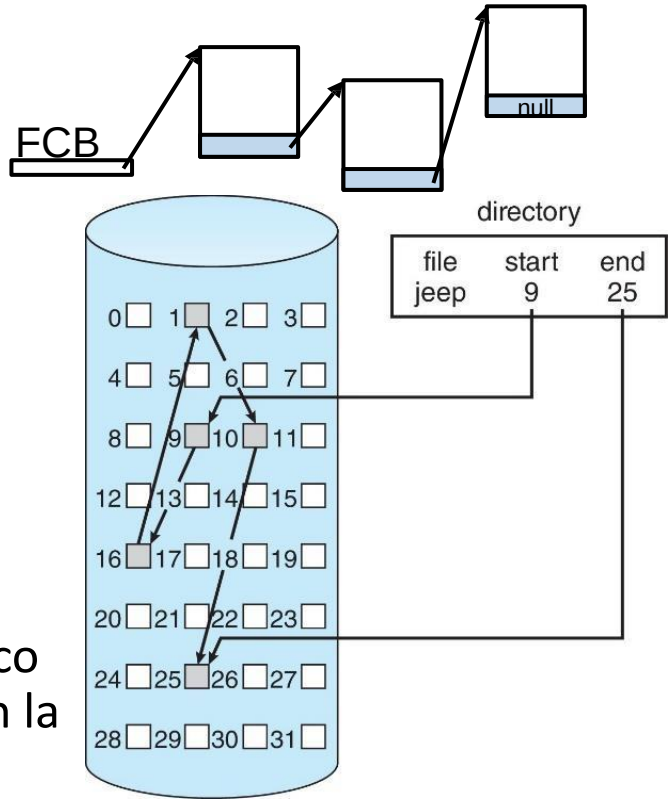
file	start	length
count	0	2
tr	14	3
mail	19	6
list	28	4
f	6	2

Sistemas de ficheros: asignación contigua

- Asignación contigua - inconvenientes:
  - Problema de **fragmentación externa**
    - La asignación dinámica fragmenta el espacio de almacenamiento al crear y borrar ficheros de distinto tamaño
    - Se crean huecos libres donde puede que no quepa un nuevo fichero
    - Solución: compactar o desfragmentar (mover ficheros para juntarlos) → costoso
  - ¿Cómo se determina el tamaño de un archivo?
    - `cp file1.txt file2.txt`: en este caso no hay problema
    - En otro caso: el usuario o el sistema tiene que indicar el tamaño en su creación
  - Si el tamaño reservado para el fichero no se ajusta al tamaño real:
    - No se completa el espacio reservado → fragmentación interna
    - Hace falta más espacio:
      - Buscar otro hueco para el fichero donde quepa, moverlo y borrar la antigua asignación (lento)
      - Otra opción: método de asignación contigua basado en *extents* (si el primer trozo contiguo de disco asignado al archivo no es suficiente se reserva otro en otra parte del disco manteniendo enlaces a *extents*

Sistemas de ficheros: asignación enlazada

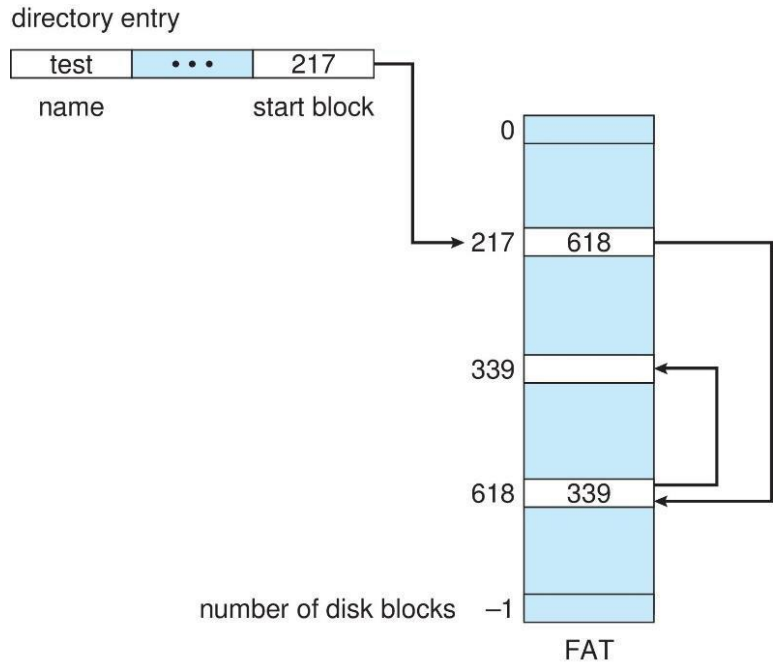
- Asignación enlazada:
  - Solventa los problemas de la asignación contigua
  - Cada **fichero es una lista enlazada de bloques** dispersos en el dispositivo de almacenamiento
  - El directorio contiene un puntero al primer y último bloque (añadir un bloque al final es sencillo)
  - Cada bloque tiene un puntero al siguiente (se pierde espacio para datos)
- Ventajas:
  - No tiene fragmentación externa
  - Los archivos pueden crecer hasta que se llene el disco
  - No es necesario especificar el tamaño del archivo en la creación (simplemente se crea una entrada en el directorio con un puntero a null)
- Desventajas:
  - Permite acceso secuencial (aunque se incrementa t_{seek} y $t_{rotation}$ en HDDs)
  - No es robusto: si se pierde un bloque, se pierde el resto del fichero a partir de ese bloque
  - Baja eficiencia en acceso aleatorio/directo



Fuente: Silberschatz, Global Edition

Sistemas de ficheros: asignación enlazada

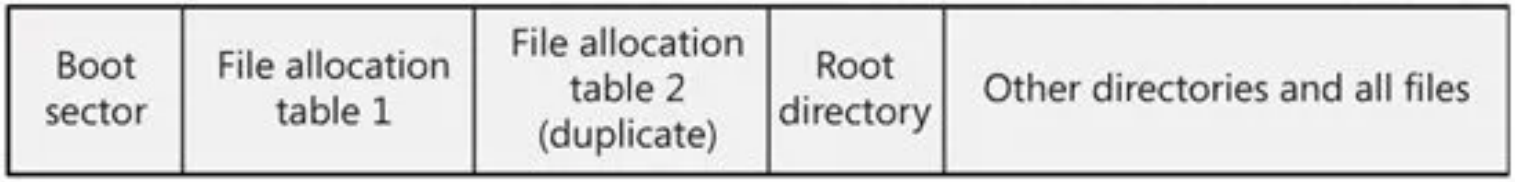
- Asignación enlazada: FAT
  - La variación **FAT** (*File Allocation Table*) del método de asignación enlazada mejora el acceso aleatorio/directo
  - Los enlaces no se mantienen en los bloques → se mantiene una estructura en el disco que contiene una tabla con una entrada por bloque del disco (FAT)
  - En cada entrada se almacena un puntero al siguiente bloque del fichero:
    - FAT##: ## es el número de bits de los punteros (FAT12, FAT16, FAT32, …)
    - El tamaño de la FAT puede ser un problema → clusters
    - Tamaño máximo posible de fichero: $2^{\#\#}$ * cluster size
      - Realmente, limitado por el campo *length* de la entrada del directorio



  - En el directorio se almacena el número del primer bloque del fichero
  - Si la FAT no está cacheada en memoria se incrementan los accesos a disco: Acceso a la FAT + acceso al bloque del fichero

Sistemas de ficheros: asignación enlazada

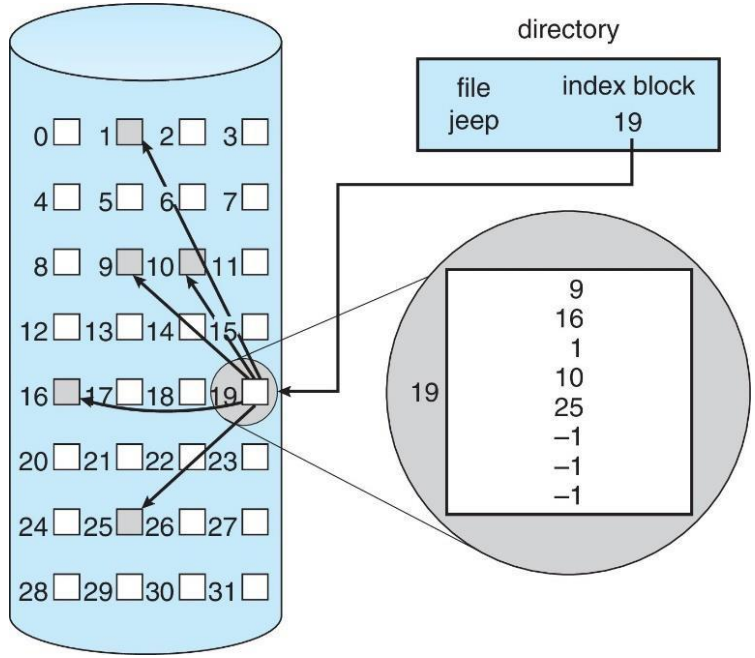
- Estructura de un volumen FAT:



- Códigos especiales para una entrada FAT:
  - *Cluster* libre: 0x0000
  - *Cluster* defectuoso: 0xFFF7
  - Último *cluster* de un archivo (rango -1 a -8):
    - 0xFF8-0xFFF para FAT12
    - 0xFFF8-0xFFFF para FAT16
    - 0xFFFFFF8-0xFFFFFFFF para FAT32

Sistemas de ficheros: asignación indexada

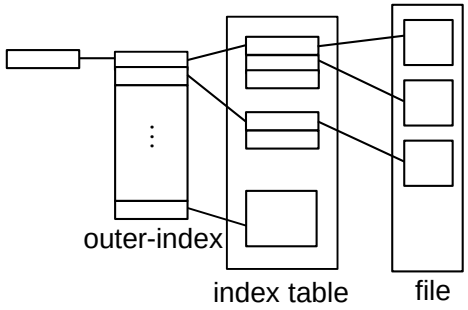
- Asignación indexada
  - Los punteros a los bloques del fichero se mantienen en el FCB: Todos los punteros juntos en un bloque/tabla de índices (index block)
  - Facilita el acceso aleatorio/dinámico
  - No tiene fragmentación externa
  - Los archivos pueden crecer fácilmente, mientras haya entradas en la tabla de índices
  - Desventajas:
    - Riesgo de fragmentación interna si la tabla de índices no se llena (por ejemplo, muchos ficheros pequeños)
    - Los bloques dispersos por el disco pueden ralentizar el acceso en HDDs
    - Para ficheros grandes, el tamaño de la tabla de índices puede no ser suficiente ¿cómo hacerla más grande?



Fuente: Silberschatz, Global Edition

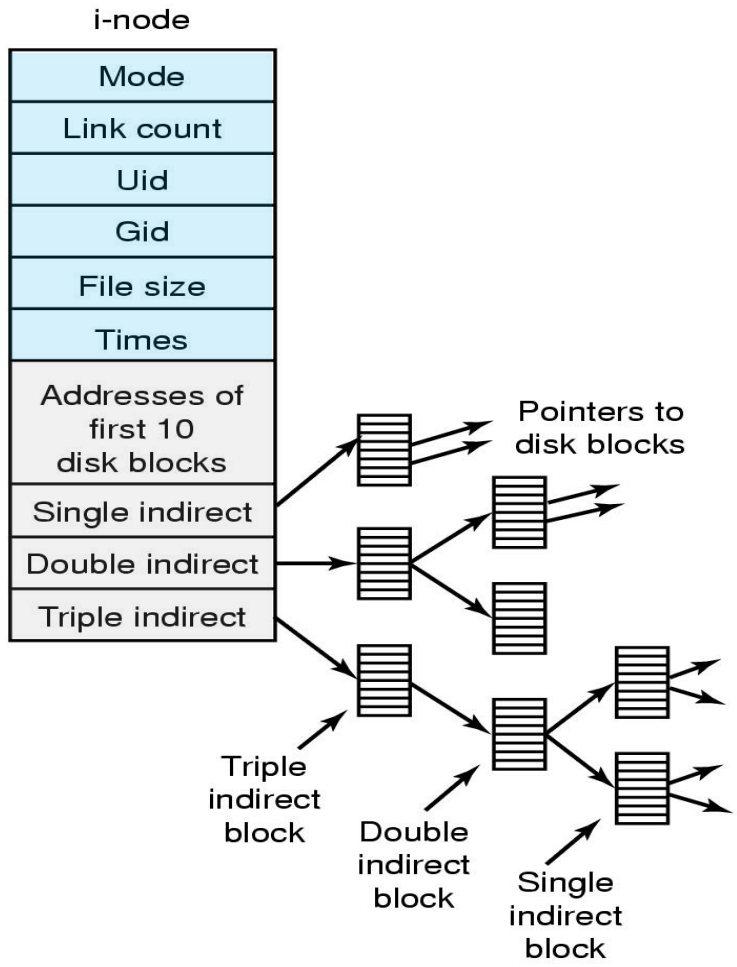
Sistemas de ficheros: asignación indexada

- Asignación indexada: Mecanismos para agrandar la tabla de índices
  - Enlazado:
    - Reservar la última entrada de la tabla de índices para que apunte a otro bloque con la extensión de la tabla de índices (o null si no hay)
  - Multinivel:
    - Las entradas de la tabla de índices del primer nivel apuntan a bloques con tablas de índices de segundo nivel. Las entradas de estas tablas apuntan a bloques de datos del fichero
    - Se puede generalizar a 3, 4, o más dependiendo del tamaño máximo de fichero deseado
  - Combinado:
    - Una parte de la table de índices contiene índices directos, es decir, que apuntan a los primeros bloques de datos del fichero
    - Las últimas entradas de la tabla de índices se usan para mantener indirecciones de 1, 2, 3, … niveles
    - Ej. Unix i-node



Sistemas de ficheros: asignación

- Asignación indexada: UNIX *i-node*
  - Unix implementa un sistema de ficheros con asignación indexada combinada
  - Eficiente para ficheros pequeños
  - Permite ficheros grandes
  - El FCB se llama i-node y contiene una tabla de índices combinada (entre 11 y 15).
  - Por ejemplo, con 13 índices:
    - Los primeros 10 punteros son directos a bloques de datos
    - El puntero 11º apunta a un bloque indirecto
    - El puntero 12º apunta a un bloque doblemente indirecto
    - El puntero 13º aputna a un bloque triplemente indirecto

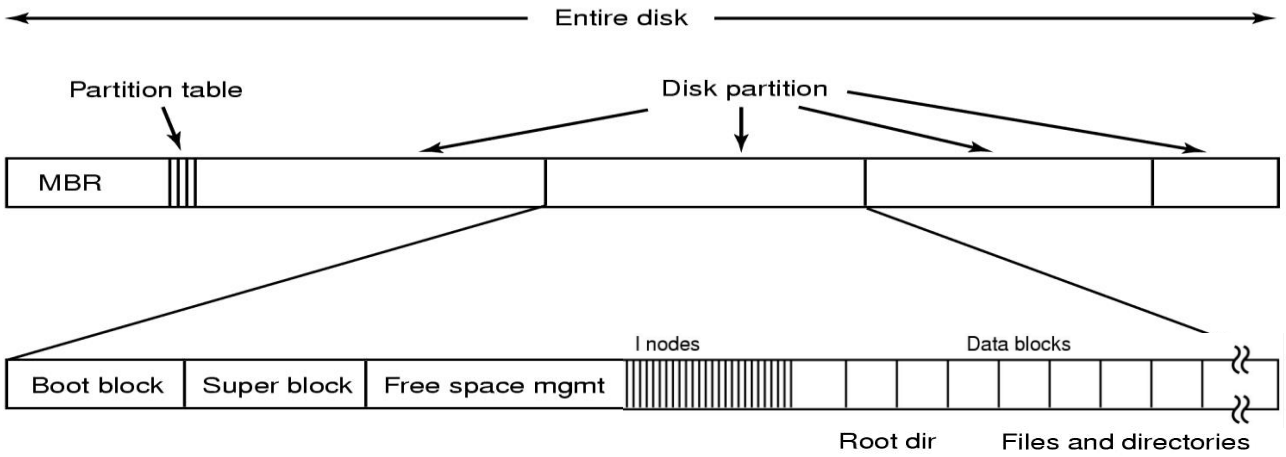


Sistemas de ficheros: asignación indexada

- Campos de un i-node con tabla de índices de 13 entradas

Field	Bytes	Description
Mode	2	File type, protection bits, setuid, setgid bits
Nlinks	2	Number of directory entries pointing to this i-node
Uid	2	UID of the file owner
Gid	2	GID of the file owner
Size	4	File size in bytes
Addr	39	Address of first 10 disk blocks, then 3 indirect blocks
Gen	1	Generation number (incremented every time i-node is reused)
Atime	4	Time the file was last accessed
Mtime	4	Time the file was last modified
Ctime	4	Time the i-node was last changed (except the other times)

- *Layout* del disco (para sistemas UNIX antiguos)



Sistemas de ficheros: asignación indexada

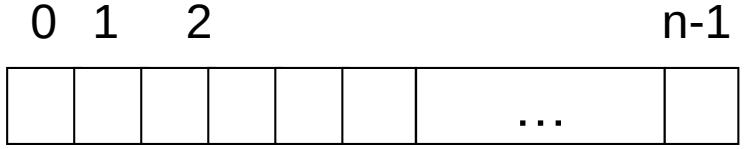
- UNIX *i-nodes*:
  - Ventajas: Sencillo de implementar
    - Los archivos pueden expandirse fácilmente rellenando las tablas de índices (hasta un punto máximo)
    - Los ficheros pequeños se implementan con facilidad y se acceden con rapidez (acceso directo)
  - Inconvenientes: muchas búsquedas (t_{seek} en HDDs) para archivos grandes
    - Ficheros muy grandes pueden requerir hasta 4 accesos a bloques (3 indirecciones)

Ejemplo: en un sistema de ficheros con asignación indexada combinada con 13 entradas para la tabla de índices, punteros de 32 bits y bloques de 1024 bytes

- ¿Cuántos accesos se necesitan para obtener el bloque 44 del disco? (Se supone que el i-node del fichero se ha accedido en la apertura) → Dos: uno para obtener la tabla de índices apuntada por el puntero 11; otro para obtener el bloque de datos apuntado por el puntero 34 de la tabla de índices obtenida en el paso anterior (En la que hay 1024B/4B = 256 entradas)
- ¿Y el bloque 5? → Uno: para obtener el bloque apuntado directamente por el puntero 5
- ¿Y el 340? → Tres: 340 > 10 + 256 → Hay que acceder al puntero 12 que contiene dos indirecciones y obtener la tabla de índices, obtener el índice de la siguiente tabla de índices y finalmente acceder al bloque
- ¿Cuál es el tamaño máximo de fichero permitido? 10*1024 + 256*1024 + 256*256*1024 + 256*256*256*1024 ≈ 16GB

Sistemas de ficheros: gestión del espacio libre

- Gestión del espacio libre: vectores de bits



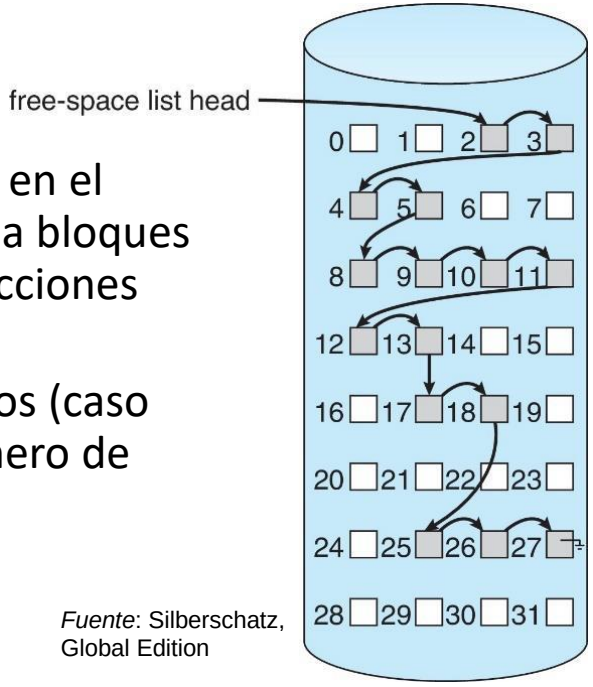
$$\text{bit}[i] = \begin{cases} 0 \Rightarrow \text{block i-ésimo libre} \\ 1 \Rightarrow \text{block i-esimo en uso} \end{cases}$$

- El vector de bits se puede mantener en memoria y acceder por palabras:
  - Muchas CPUs ofrecen instrucciones para hacer operaciones a nivel de bit (bitwise) de manera eficiente
  - Cálculo del número de bloque a partir del vector de bits, leyéndolo en palabras de n bits = n * (nº de palabras a 0) + offset del primer bit a 1 de la primera palabra que no está a 0
- Facilita encontrar espacio libre contiguo para ficheros
- Desventajas:
  - El vector de bits puede ocupar mucho espacio: Ej. Un disco de 1 TB con bloques de 4 KB requerirá 32 MB de vector de bits (240/221 = 218 bits = 225 bytes = 25 MB)
    - Escala mal con el tamaño de disco
  - Para que sea **eficiente** tiene que estar en **memoria**, pero hay que mantener copias en disco cada cierto tiempo (coherencia)
    - No se puede permitir que bit[i] valga 0 en disco y 1 en memoria

Sistemas de ficheros: gestión del espacio libre

- Gestión del espacio libre: lista enlazada
  - Sólo hace falta mantener en memoria el puntero al primer bloque libre
  - En FAT, se mantiene la lista de bloques libres en la propia FAT
  - Desventajas:
    - No es eficiente: caminar por la lista de bloques libres requiere muchos accesos a disco
    - Obtener espacio libre contiguo es complicado

- Variantes de lista enlazada:
  - *Grouping*: mantiene los punteros a n bloques libres en el primer bloque; los primeros n-1 punteros apuntan a bloques libres; el último apunta a otro bloque libre con direcciones de bloques libres;…
  - *Counting*: para grupos de bloques libres consecutivos (caso común) se mantiene el puntero al primero y el número de bloques libres consecutivos



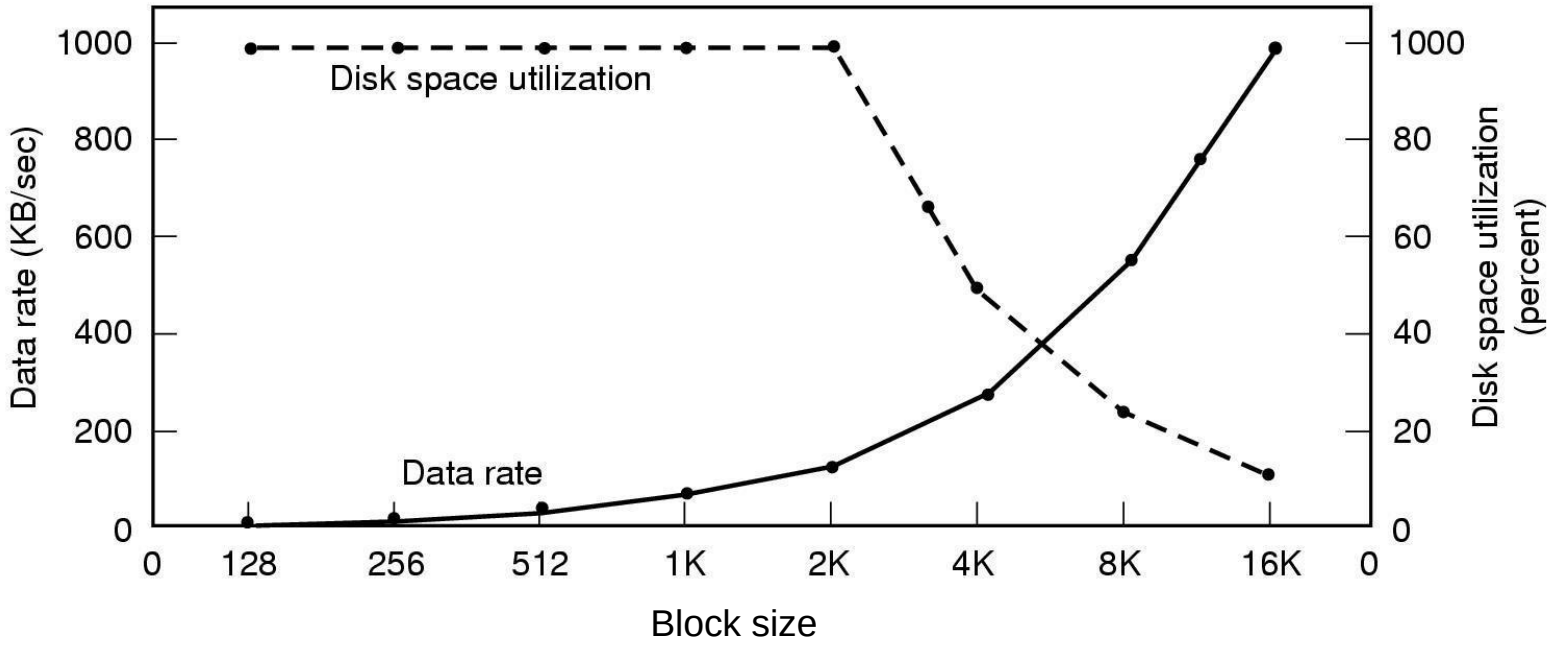
Fuente: Silberschatz, Global Edition

Sistemas de ficheros: prestaciones

- Eficiencia y rendimiento: tamaño de bloque (cluster)
  - La elección del tamaño del bloque mínimo que se accederá en cada transacción de disco puede determinar la eficiencia y el rendimiento del sistema de ficheros (¿Tamaño de página de MV?¿Tamaño de sector en HDDs? …)
  - Tamaño de bloque grande:
    - Más fragmentación interna
      - Si hay muchos ficheros pequeños se desperdicia mucho espacio
      - Baja utilización del espacio de disco (disk space utilization)
    - En HDDs, menor tiempo de búsqueda, mejor rendimiento
  - Tamaño de bloque pequeño:
    - En HDDs, más búsquedas, peor rendimiento
    - Estructuras de datos más grandes (ej. FAT)
    - Menor fragmentación interna
      - Mejor utilización del espacio de disco

Sistemas de ficheros: prestaciones

- El tamaño de bloque (*cluster*) tiene gran influencia en las prestaciones
- Debe elegirse un valor de compromiso



En esta gráfica, se asume un HDD todos los ficheros de 2 KB, un t_{seek} = 10 ms, $t_{rotation}$=8.33 ms * ½ y velocidad de tranferencia = 15 MB/s