Tema 4:
Sistemas de Ficheros

Grado en Ing. Informática

Curso 2024/25

Profesor: Manuel E. Acacio Sánchez

Índice

4.1. Introducción

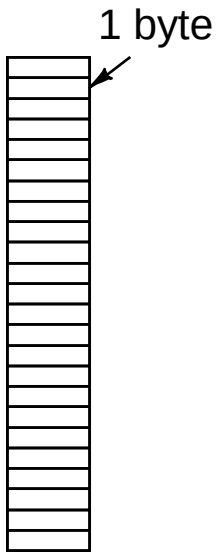
- El **almacenamiento secundario** es necesario para:
  - Almacenar gran cantidad de datos que no caben en memoria ppal
  - Preservar información para que no desaparezca aunque termine el proceso (o aunque se apague el sistema)
  - Compartir información como programas, documentos,... entre varios procesos
- Los dispositivos de almacenamiento secundario guardan la información en un cjto de bloques sin más estructura
  - El SO debe proporcionar una *interfaz sencilla* para acceder a dichos dispositivos
- Solución: **Sistema de Ficheros (SF)**

  ⇒ Estructura de datos basada en **ficheros** (unidades que almacenan información) y **directorios** (permiten organizar y agrupar los ficheros)
- Primero veremos el **SF** desde el punto de vista del **usuario** y después desde el de la implementación por parte del **SO**

Índice

4.2 Ficheros

- Un fichero es la unidad lógica de almacenamiento
  - Para guardar y recuperar información de un dispositivo de almacenamiento secundario se hace uso de un fichero
- Estructura de un fichero: En los SO modernos, secuencia de bytes, el significado lo da el programa que accede al mismo
  - Flexibilidad: cada programa puede definir la estructura de datos más conveniente para organizar la información en el fichero
- Cada fichero se identifica mediante un nombre
  - El sistema de ficheros determina cómo puede ser el nombre (estructurado o no, longitud, caracteres válidos, distinción entre mayúsculas y minúsculas…)



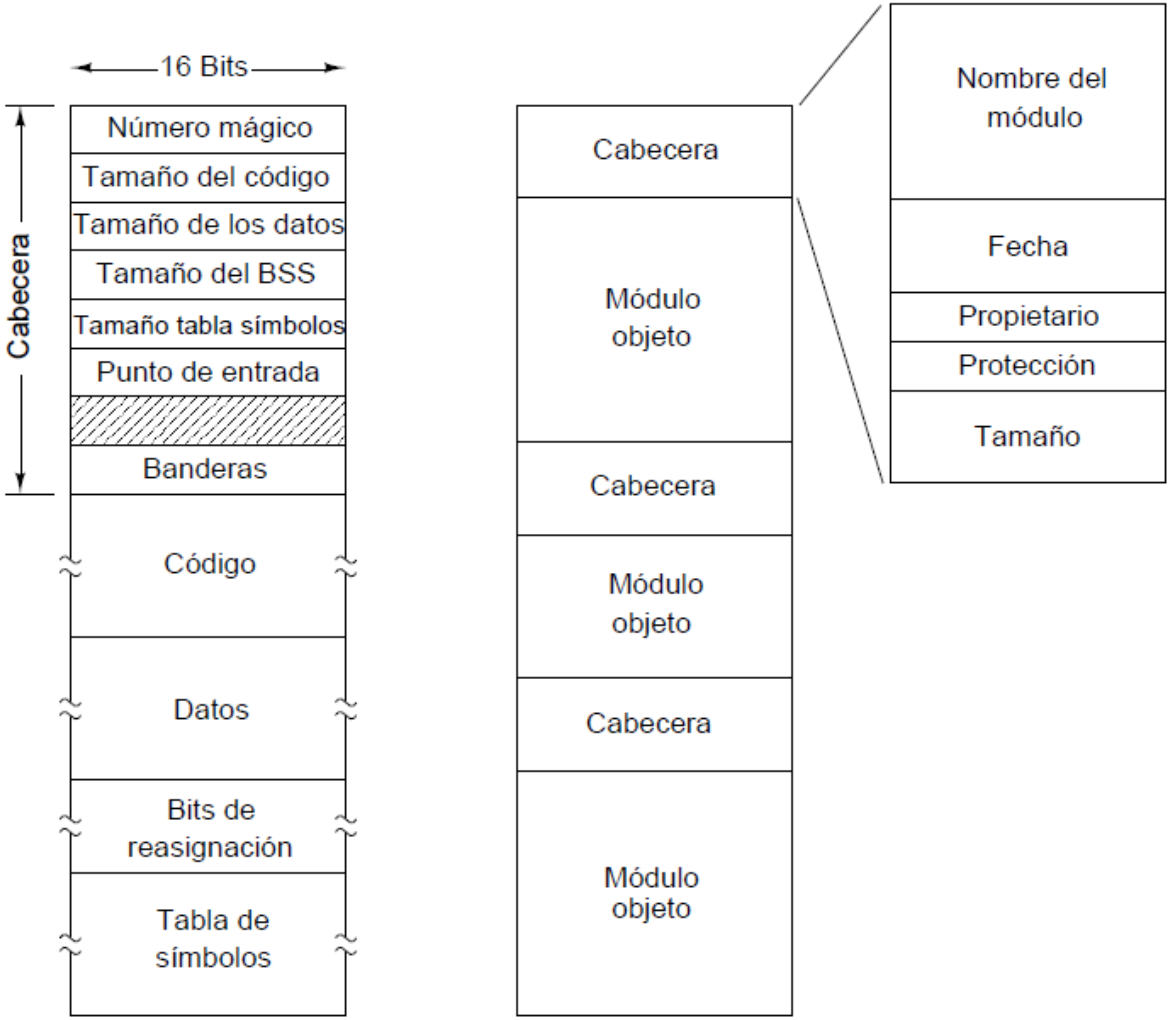
ristra
de bytes

4.2.1 Tipos de ficheros

- **Tipos de ficheros**
  - Ficheros regulares: contienen la información del usuario
    - Ficheros de texto: pueden verse tal cual (líneas de texto) y modificarse usando un editor de texto
    - Ficheros binarios: no pueden verse tal cual, tienen estructura interna que depende del programa que los usa o, si son ejecutables, del SO
  - Directorios: ficheros gestionados por el SO para organizar ficheros
  - Ficheros especiales de dispositivo:
    - Ficheros especiales de caracteres: para acceder a dispositivos de E/S serie (como impresoras, redes, …)
    - Ficheros especiales de bloques: para acceder a dispositivos de almacenamiento secundario con acceso aleatorio (discos)
  - Ficheros enlaces simbólicos: almacenan la ruta de acceso a ficheros, con lo que permiten acceder a un mismo fichero desde distintos directorios

4.2.1 Tipos de ficheros

- **Ejemplos de ficheros binarios (UNIX):**



4.2.1 Tipos de ficheros

- **Extensiones:** pueden decirle a un explorador de ficheros qué acción llevar a cabo sobre el fichero. Ejemplos:

Extensión	Significado
fichero.bak	Copia de seguridad
fichero.c	Código fuente en C
fichero.gif	Imagen en formato GIF
fichero.hlp	Fichero de ayuda
fichero.html	Código fuente en HTML
fichero.jpg	Imagen fija en formato JPEG
fichero.mp3	Sonido codificado en MP3
fichero.mpg	Vídeo codificado en MPEG
fichero.o	Fichero objeto (salida del compilador, todavía no enlazada)
fichero.pdf	Fichero PDF
fichero.ps	Fichero PostScript
fichero.tex	Fichero fuente de TEX
fichero.txt	Fichero de texto genérico
fichero.zip	Archivo comprimido con formato ZIP

4.2.2 Acceso a un fichero

- Acceso secuencial
  - Todos los bytes del fichero se leen en orden, desde el principio
  - Condicionado por el medio de almacenamiento (por ejemplo, un teclado o cinta)
  - Impuesto por el propio Sistema Operativo (por ejemplo, una tubería)

- Acceso aleatorio
  - Se pueden leer los bytes en cualquier orden (llamadas al sistema que permiten posicionamiento)
  - También admite acceso secuencial

4.2.3 Atributos de un fichero

- Un fichero tiene elementos adicionales llamados atributos

	Campo	Significado
Protección	Protección	Quién debe tener acceso y de qué forma
	Contraseña	Contraseña necesaria para tener acceso al fichero
	Creador	Identificador de la persona que creó el fichero
	Propietario	Propietario actual
Ciertas propiedades	Bandera de sólo lectura	0 Lectura/escritura, 1 para lectura exclusivamente
	Bandera de ocultación	0 normal, 1 para no exhibirse en listas
	Bandera del sistema	0 fichero normal, 1 fichero de sistema
	Bandera de biblioteca	0 ya se ha respaldado, 1 necesita respaldo
	Bandera texto/binario	0 fichero de texto, 1 fichero binario
	Bandera de acceso aleatorio	0 sólo acceso secuencial, 1 acceso aleatorio
	Bandera temporal	0 normal, 1 eliminar al salir del proceso
	Banderas de cerradura	0 no bloqueado, $\neq 0$ bloqueado
Tiempo	Tiempo de creación	Fecha y hora de creación del fichero
	Tiempo del último acceso	Fecha y hora del último acceso al fichero
	Tiempo de la última modificación	Fecha y hora de la última modificación del fichero
Tamaños	Tamaño actual	Número de bytes en el fichero
	Tamaño máximo	Tamaño máximo al que puede crecer el fichero

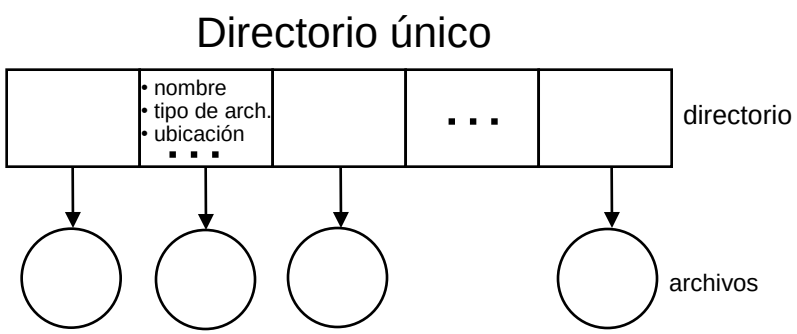
4.2.4 Operaciones con ficheros

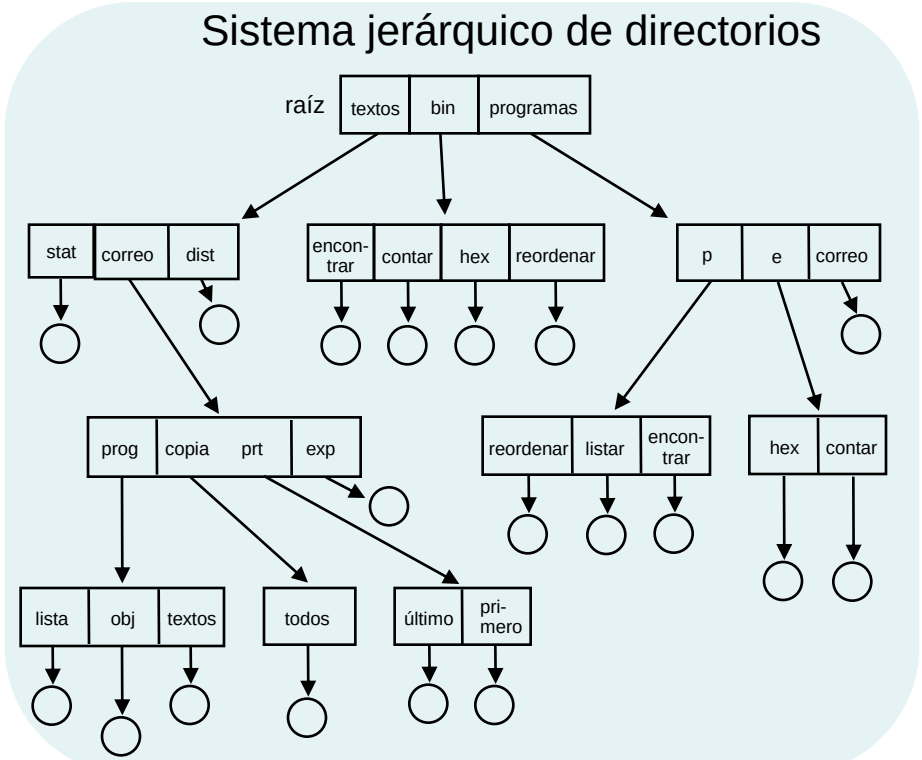
- Algunas **llamadas al sistema** para el manejo de los ficheros:
  - **Create**: crea un fichero vacío (se inician algunos atributos)
  - **Delete**: elimina un fichero (libera espacio)
  - **Open**: antes de usar un fichero, el proceso debe abrirlo
  - **Close**: cierra un fichero abierto
  - **Read/Write**: lee/escribe datos de/en un fichero
  - **Append**: escribe datos al final del fichero (caso especial de **Write**)
  - **Seek**: especifica el punto de lectura/escritura de datos en un fichero de acceso aleatorio
  - **Get/Set attributes**: obtiene/establece los atributos asociados a un fichero
  - **Rename**: cambia el nombre de un fichero
  - **Truncate**: elimina el contenido de un fichero a partir de una posición dada

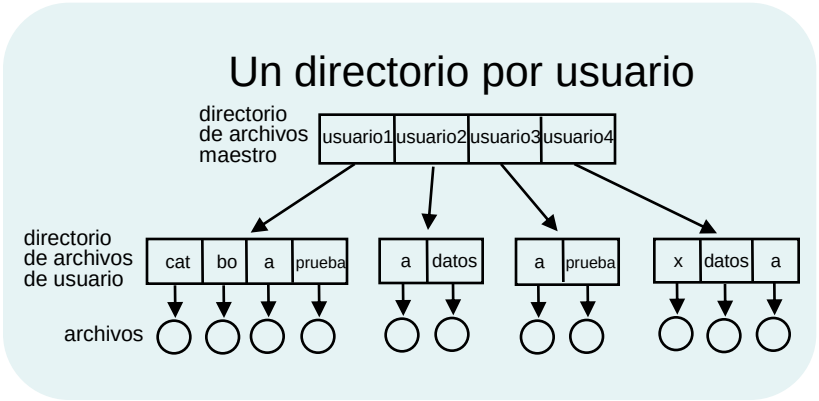
Índice

4.3.1 Directorios

- Son necesarios para organizar los ficheros
- Suelen ser ficheros que almacenan información sobre otros ficheros (nombre, atributos, etc.)
- Son posibles distintas organizaciones:







4.3.2 Nombre de la ruta de acceso

- Los SS.OO. con un sistema jerárquico de directorio suelen tener dos formas básicas para indicar la ruta de acceso o nombre de un fichero:

  - Ruta absoluta:
    - Especifica el camino desde el directorio raíz hasta el fichero
    - El primer carácter de la ruta es el separador («/» en Unix, «\» en Windows). Ejemplo: `/usr/bin/firefox`

  - Ruta relativa:
    - Asociada al concepto de directorio actual
    - No empieza por el carácter separador y dependen del directorio actual. Ejemplo: `bin/firefox` si el directorio actual es `/usr`

  - Dos directorios especiales:
    - Directorio «.»: directorio actual
    - Directorio«..»: directorio padre

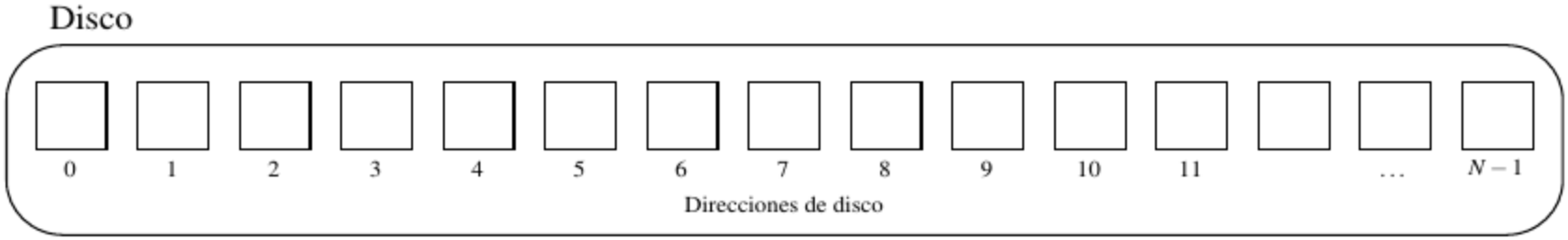
4.3.3 Operaciones con directorios

- Algunas llamadas al sistema para el manejo de directorios (varían más de un SO a otro que las de ficheros):
  - **Create**: crea un directorio vacío (tendrá las entradas «.» y «.. »)
  - **Delete**: borra un directorio vacío
  - **Opendir**: abre un directorio para recorrerlo
  - **Closedir**: cierra un directorio abierto
  - **Readdir**: devuelve la siguiente entrada de directorio abierto
  - **Rename**: cambia de nombre a un directorio
  - **Link**: permite que un fichero aparezca en varios directorios
  - **Unlink**: elimina entrada de directorio (si es la única del fichero, elimina el fichero)
- Preguntas:
  - Si un directorio es un fichero ¿por qué se usa `readdir` y no `read`?
  - ¿Por qué no existe `writedir`?

Índice

4.4 Implementación del sistema de ficheros

- En lo que sigue vamos a representar al dispositivo de almacenamiento como un array lineal de bloques*:*
  - Supondremos bloques con un tamaño potencia de 2 (512, 1024, 4096, etc. bytes)
  - Los bloques se numerarán desde 0 hasta N-1, siendo N el tamaño del disp. de almacenamiento, en bloques
  - Cada bloque tiene una posición que lo identifica: dirección de disco del bloque (o dirección)



4.4 Implementación del sistema de ficheros

- El objetivo del sistema de ficheros es usar dicha estructura de datos para almacenar en ella información usando ficheros y directorios
- Se intenta agrupar la información relacionada en bloques secuenciales
- Veremos la implementación de los elementos individuales (ficheros, directorios, etc.)

4.4.1 Implementación de ficheros

- Todo fichero tiene asociado un conjunto de bloques de disco
- Implementación de ficheros: manera de llevar un registro de qué bloques pertenecen a un fichero
- Hay varias posibilidades:

  - Asignación contigua

  -Asignación mediante lista ligada

  -Asignación mediante lista ligada e índice

  -Nodos-i

4.4.1 Implementación de ficheros

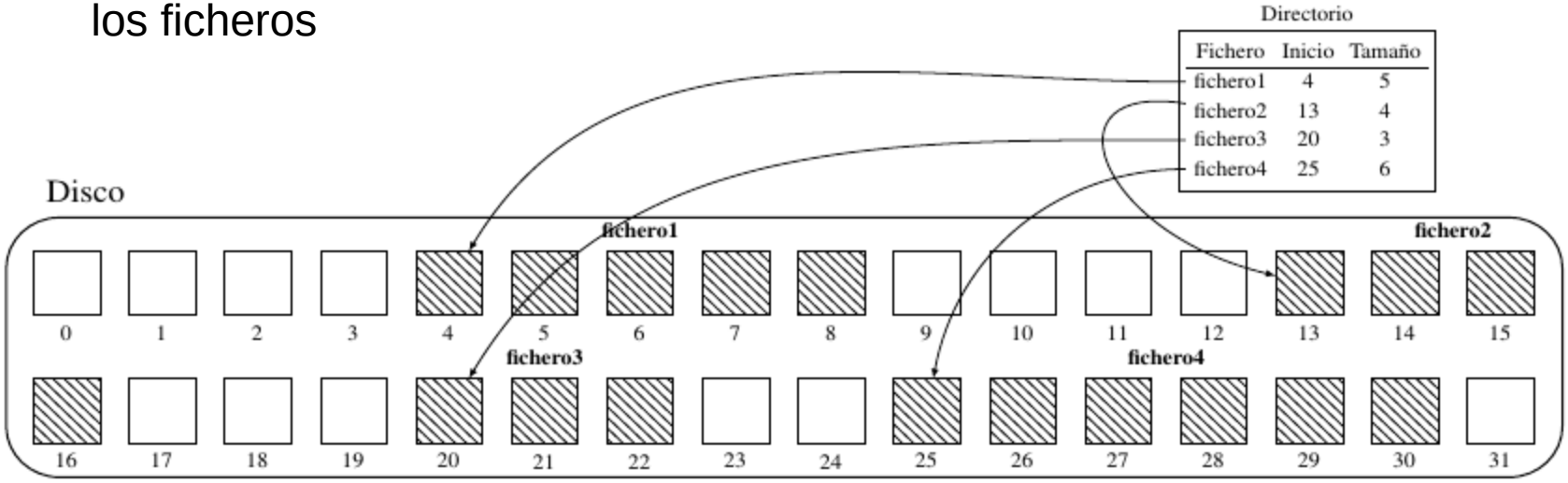
- Asignación contigua:

- Esquema más sencillo
- Todos los bloques de un mismo fichero están contiguos

Pros: 1) Fácil de implementar (a partir de la dirección del primer bloque sabremos las demás); 2) Excelente rendimiento (lecturas secuenciales)

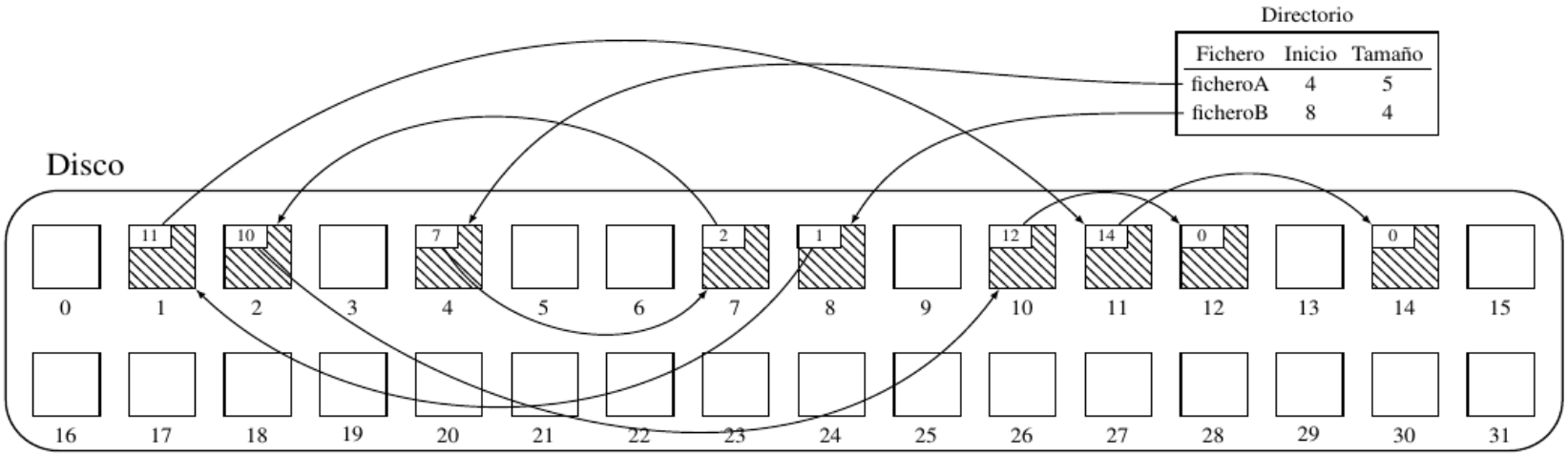
Cons: 1) Irrealizable de forma eficiente en un sistema de ficheros de propósito general (no se sabe a priori el tamaño máximo de cada fichero); 2) fragmentación externa (bloques libres repartidos en grupos pequeños)

- Útil en DVDs o CD-ROMs, ya que se conoce de antemano el tamaño de los ficheros



4.4.1 Implementación de ficheros

- **Asignación con lista enlazada:**



- Cada bloque contiene un puntero (nº de bloque) al bloque siguiente

Pros: 1) no hay fragmentación externa; 2) Necesita solo la dirección del primer bloque

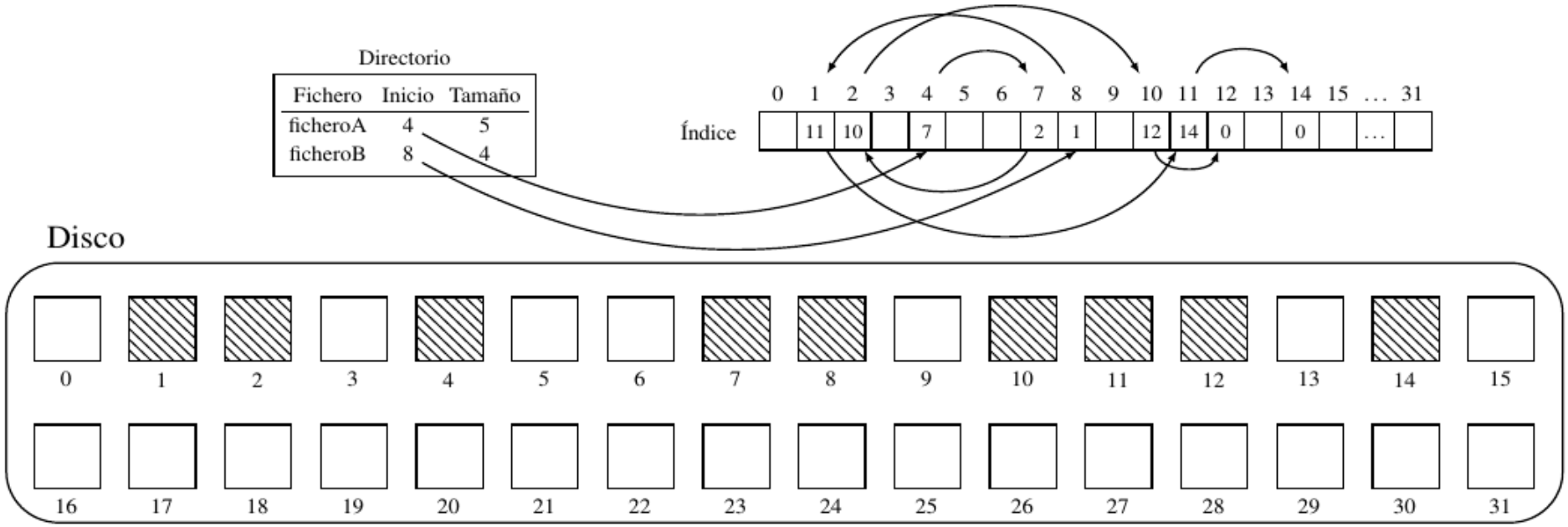
Cons: 1) acceso aleatorio lento; 2) El tamaño del espacio para datos en el bloque ya no es potencia de 2

4.4.1 Implementación de ficheros

- **Asignación con lista enlazada e índice:**
  - Misma idea que antes, pero los punteros se almacenan en una estructura aparte (**índice**) que se almacena en disco, se trae a memoria cuando se usa el sistema de ficheros y se escribe de nuevo en disco si se modifica

Pros: elimina los dos inconvenientes anteriores

Cons: 1) Índice tiene que estar en memoria y con discos de gran capacidad consumiría mucha (aumentar el tamaño de bloque no resuelve el problema, lo postpone, y crea fragmentación interna); 2) Pueden surgir inconsistencias si las modificaciones no se guardan en disco

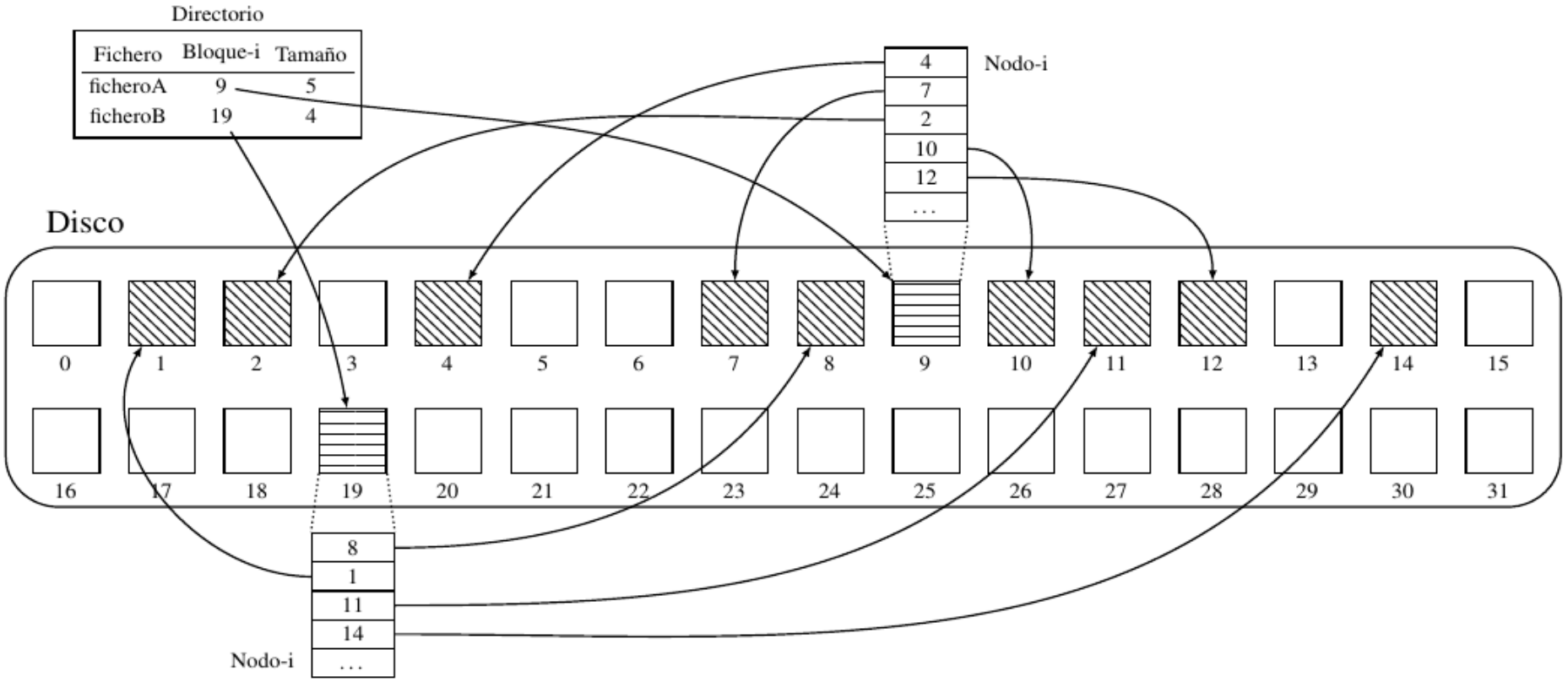


4.4.1 Implementación de ficheros

- **Asignación con nodos-i:**
  - **Idea**: cada fichero, además de los bloques de datos, tiene asociado un bloque *índice* (nodo-i)
  - **Nodo-i**: bloque de disco con la lista de números de bloque, en orden, que componen el fichero
  - **Problema 1**: la mayoría de los ficheros son pequeños, con lo que se desperdiciarían muchos bloques de discos que estarían medio vacíos (fragmentación interna)
  - **Solución 1**: nodos-i pequeños y varios nodos-i por bloque
  - El nodo-i contiene los atributos de un fichero y las direcciones de sus bloques. Junto con cada nombre de fichero se guarda la posición de su nodo-i en la zona reservada para nodos-i en disco
  - El nodo-i se lee cuando se abre el fichero y sólo tiene que estar en memoria mientras el fichero correspondiente esté abierto (la memoria necesaria es independiente del tamaño del disco)

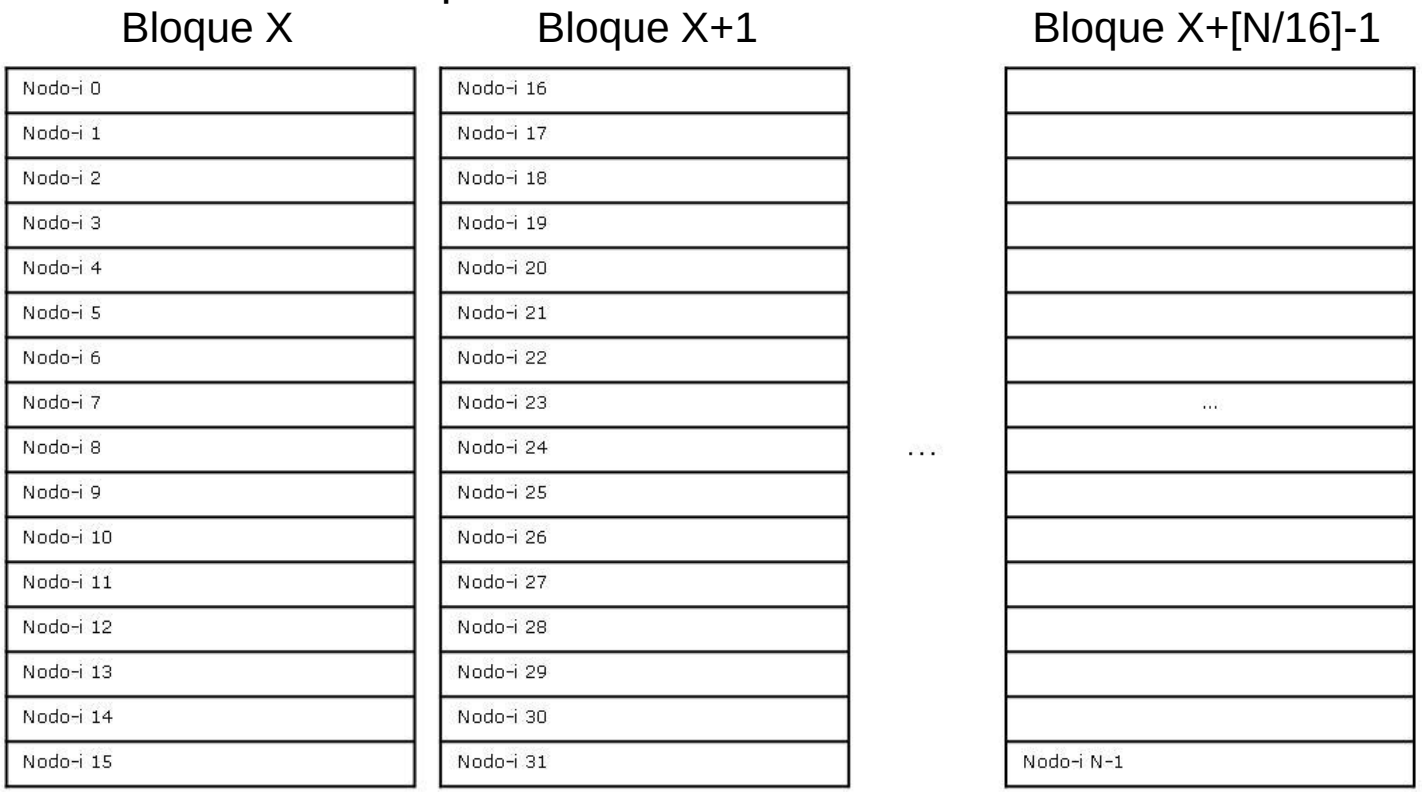
4.4.1 Implementación de ficheros

- **Asignación con nodos-i:**



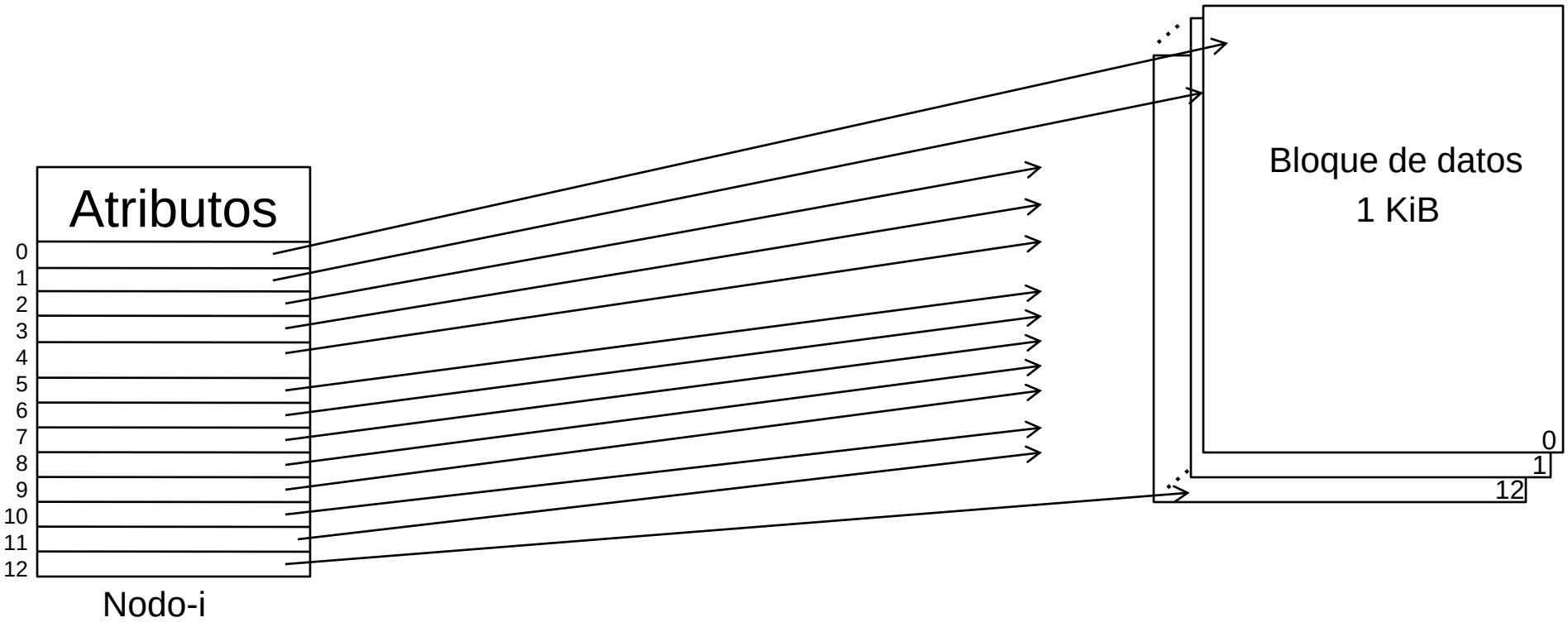
4.4.1 Implementación de ficheros

- **Asignación con nodos-i:**
  - **Solución 1**: nodos-i pequeños y varios nodos-i por bloque
    - Ej. Tamaño de bloque 1024 bytes y nodos-i de 64 bytes → 16 nodos-i por bloque.
    - Tabla nodos-i → Bloques consecutivos en disco con nodos-i



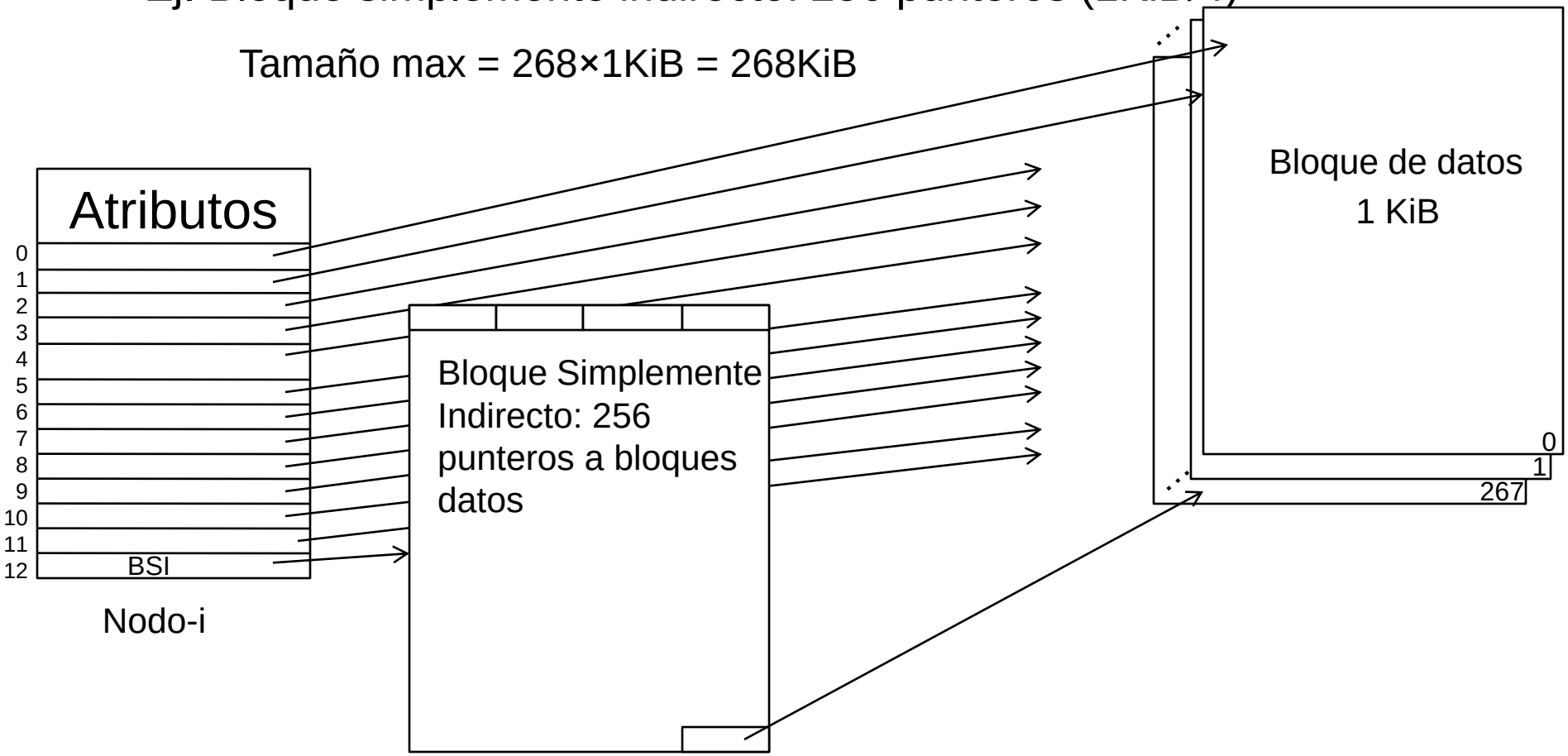
4.4.1 Implementación de ficheros

- **Ejemplo nodo-i de 64 bytes y tamaño de bloque de 1KiB:**
  - En cada bloque caben 16 nodos-i (=1024/64)
  - Supongamos direcciones de 4 bytes y 13 direcciones por nodo-i (resto de bytes para atributos)
  - ¿Tamaño máximo de fichero soportado? 13×1 KiB = 13 KiB



4.4.1 Implementación de ficheros

- **Ejemplo nodo-i de 64 bytes y tamaño de bloque de 1KiB:**
  - Bloques indirectos permiten tamaños de fichero mayores
  - Ej. Bloque simplemente indirecto: 256 punteros (1KiB/4)

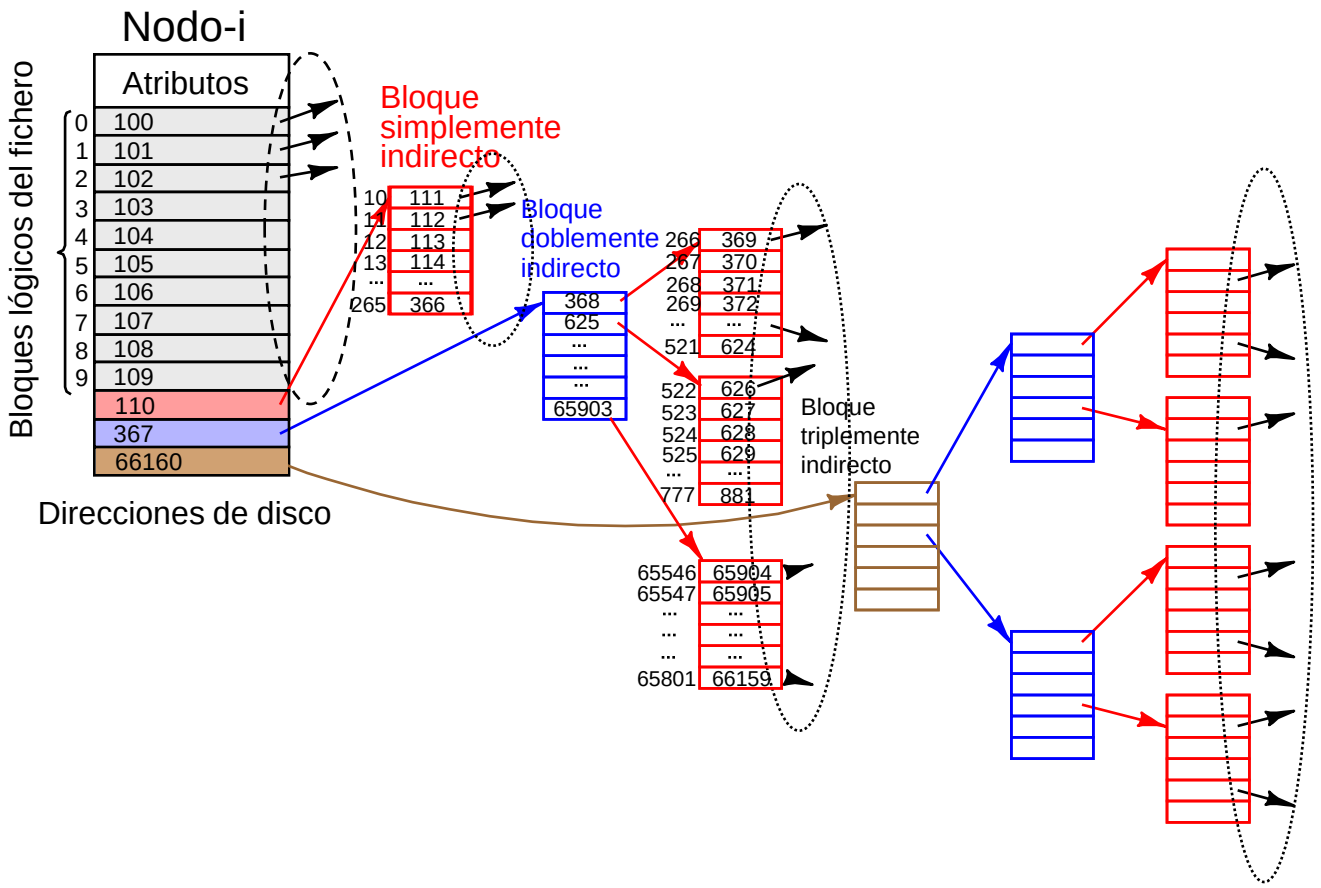


4.4.1 Implementación de ficheros

- **Ejemplo nodo-i de 64 bytes y tamaño de bloque de 1KiB:**
  - Bloques indirectos permiten tamaños de fichero mayores
  - BSI: 256 punteros a bloques de datos del fichero
  - ¿Ficheros aún más grandes? Bloque Doblemente Indirecto (BDI): 256 punteros a BSI (cada uno 256 punteros a bloques de datos) = 256 × 256 = 65.536 punteros a bloques de datos
  - ¿Ficheros MUCHO más grandes? Bloque Triplemente Indirecto (BTI): 256 punteros a BDI, cada uno con 256 punteros a BSI, cada uno con 256 a bloques de datos = 256 × 256 × 256 = 16.777.216 punteros a bloques de datos en total

4.4.1 Implementación de ficheros

- **Asignación con nodos-i**
  - ¿Ficheros grandes? Se usan *bloques indirectos*, que no almacenan datos, sino más direcciones de bloques

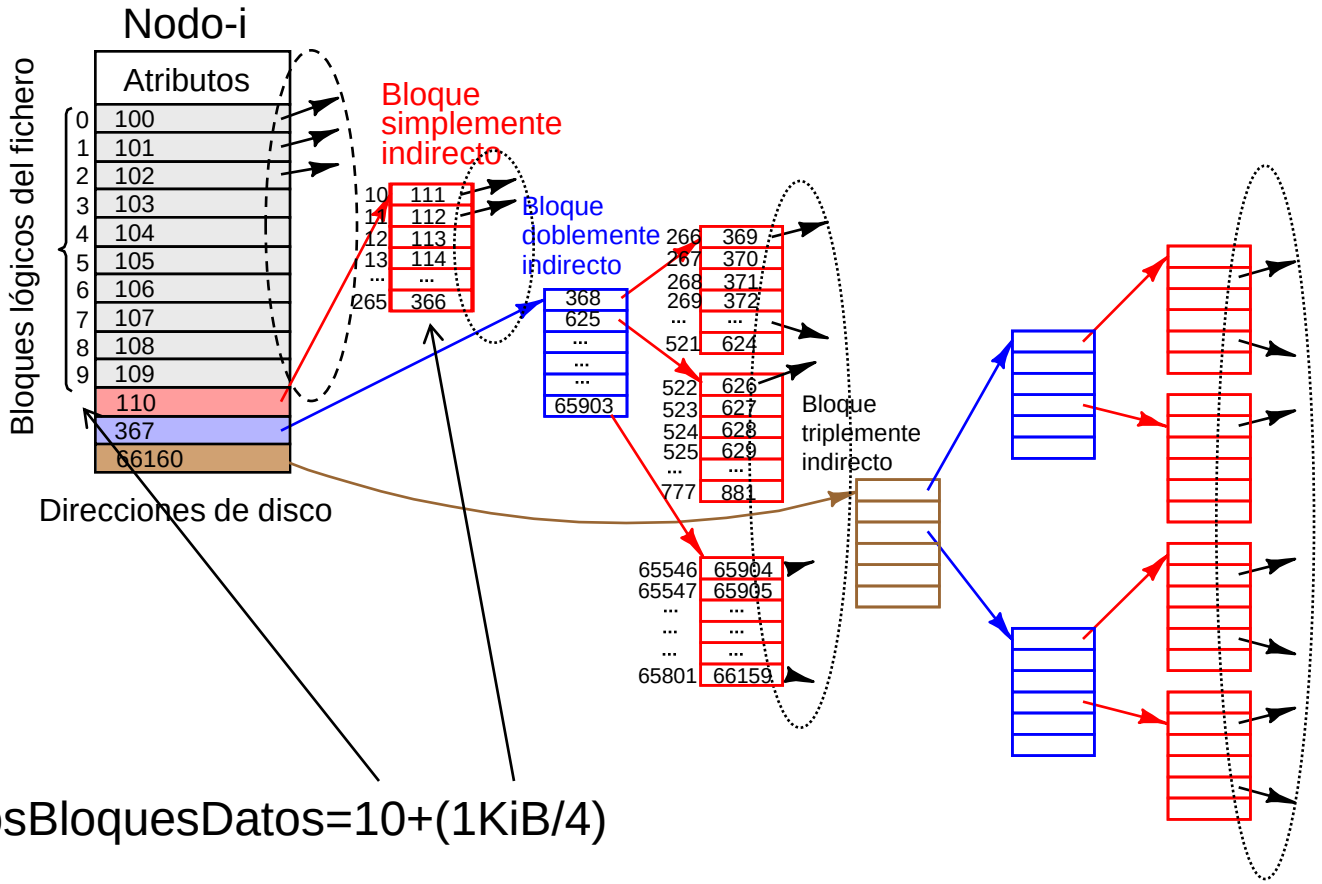


4.4.1 Implementación de ficheros

- Esta es la aproximación usada en Unix
- Ventajas con respecto a la lista enlazada con índice:
  - Cada nodo-i está en memoria solo cuando el fichero es abierto
  - Entrada en directorio de un fichero: indica el número de nodo-i
  - Acceder a cualquier posición: a la sumo leer 4 bloques: **BTI**-**BDI**-**BSI**-bloque_de_datos
  - Por ejemplo, fichero de hasta 16GB con nodo-i (bloques 1 KiB, Direcciones de 4 bytes, Nodo-i con 10+3 direcciones)
    - En lista enlazada precisa 64MB de FAT en RAM para él solo (16×2^{20} bloques x 4 bytes de dirección por bloque)

4.4.1 Implementación de ficheros

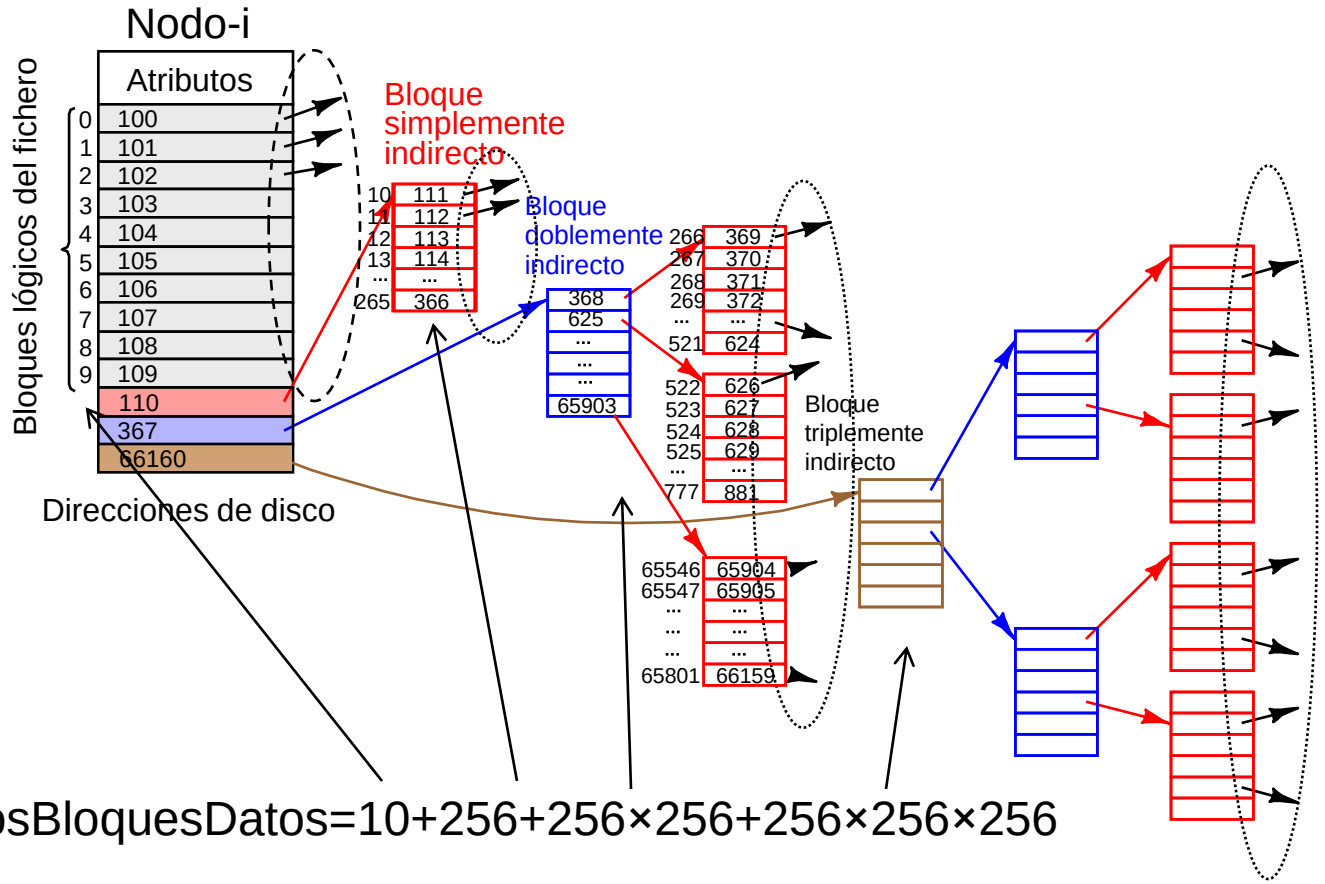
- **Ejemplo:**
  - Bloques de 1KiB, direcciones de 4 bytes, nodo-i con 13 entradas
  - Tamaño máximo de fichero = NºPunterosBloquesDatos×2^{10}≈16GB



NºPunterosBloquesDatos=10+(1KiB/4)

4.4.1 Implementación de ficheros

- **Ejemplo:**
  - Bloques de 1KiB, direcciones de 4 bytes, nodo-i con 13 entradas
  - Tamaño máximo de fichero = NºPunterosBloquesDatos×2^{10}≈16GB



NºPunterosBloquesDatos=10+256+256×256+256×256×256

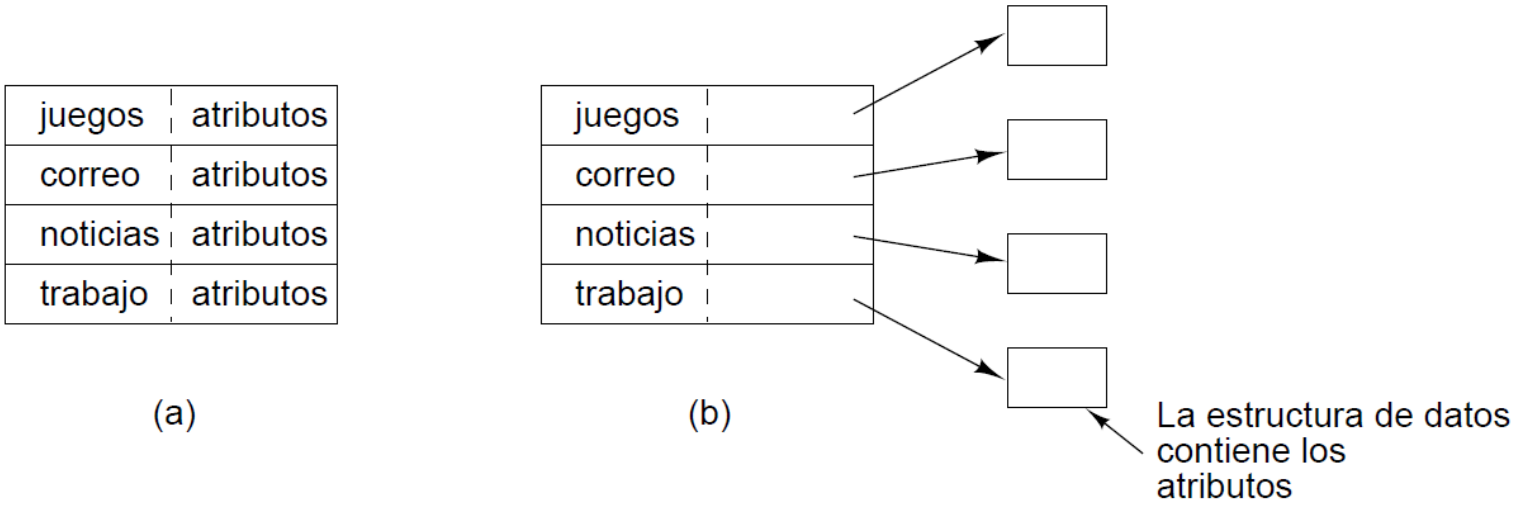
4.4.1 Implementación de ficheros

- Esta es la aproximación usada en Unix
- Ventajas con respecto a la lista enlazada con índice:
  - Cada nodo-i está en memoria solo cuando el fichero es abierto
  - Entrada en directorio de un fichero: indica el número de nodo-i
  - Acceder a cualquier posición: a la sumo leer 4 bloques: **BTI**-**BDI**-**BSI**-bloque_de_datos
  - Por ejemplo, fichero de hasta 16GB con nodo-i (bloques 1 KiB, Direcciones de 4 bytes, Nodo-i con 10+3 direcciones)
    - En lista enlazada precisa 64MB de FAT en RAM para él solo (16×2^{20} bloques x 4 bytes de dirección por bloque)

- Sistema de ficheros Ext4 :
  - Tamaño de bloque de 4 KiB
  - Nodo-i con 15 punteros (12 directos + 1 **BSI** + 1 **BDI** + 1 **BTI**) y 256 bytes de tamaño
  - Hasta 4×2^{30} ficheros y de hasta 16 TiB
  - Tamaño máximo del sistema de fichero 1 EiB

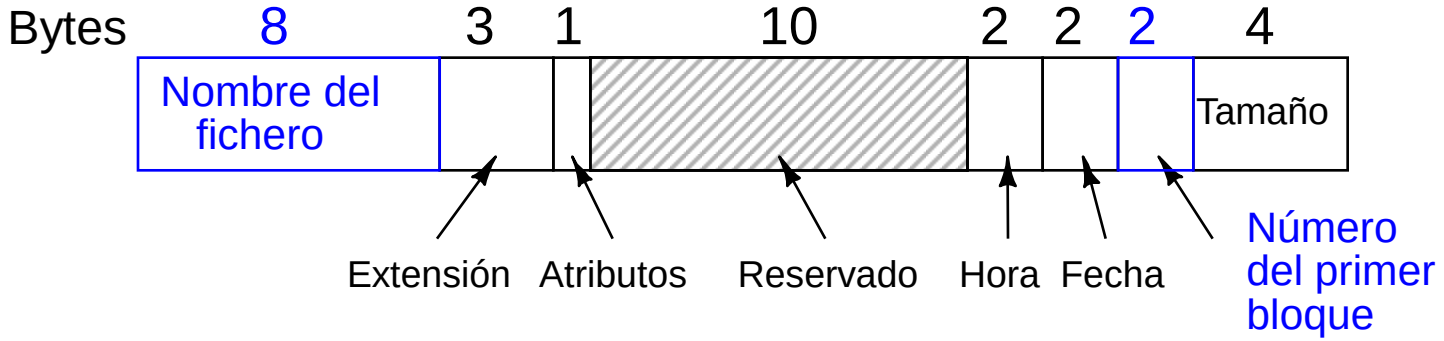
4.4.2 Implementación de directorios

- Principal función de los directorios: asociar un nombre de fichero con la información para localizar los datos del propio fichero
  - Al abrir un fichero el SO usa la ruta para localizar su entrada de directorio y obtener su información (atributos, bloques de datos, …)
- Un aspecto estrechamente relacionado con el anterior es dónde se guardan los atributos del fichero. Dos posibilidades:
  - En la propia entrada del directorio **(a)**
  - En una estructura aparte apuntada por la entrada del directorio **(b)**



4.4.2 Implementación de directorios

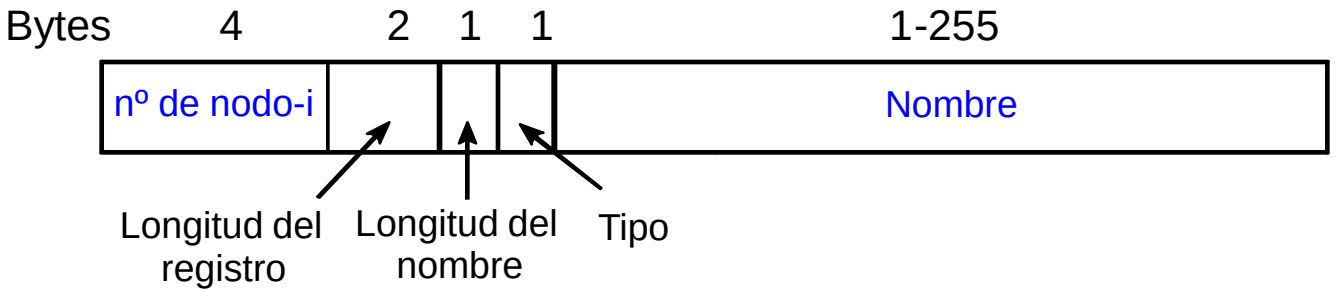
- Directorios en MS-DOS:
  - Los directorios son ficheros que almacenan una lista desordenada de entradas (o registros) de 32 bytes



  - Un bit de los atributos de la entrada distingue a un directorio de un fichero normal. Un directorio puede tener subdirectorios formando un árbol de directorios
  - El directorio raíz es una excepción: no se implementa como un fichero y ocupa unos bloques fijos en el disco (tiene un tamaño máximo establecido)
  - A día de hoy se usa en sistemas de ficheros como vFAT o FAT32

4.4.2 Implementación de directorios

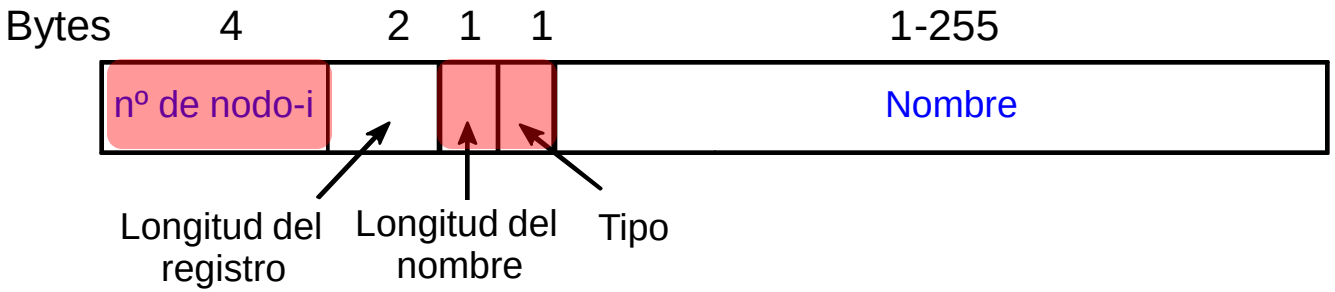
- Directorios en UNIX
  - También es posible crear un árbol de directorios
    - Uno de los atributos (1 bit) diferencia entre ficheros normales y directorios
  - Todos los directorios (incluido el raíz) son ficheros que almacenan una lista desordenada de entradas de longitud variable (suele ser múltiplo de 4)



  - Las entradas no almacenan atributos de ficheros sino nombres y números de nodos-i asociados. El tipo de fichero es una excepción y se utiliza para acelerar los listados de directorios
  - Toda la información sobre tipo, tamaño, tiempos, propiedad y bloques en disco se almacena en el nodo-i

4.4.2 Implementación de directorios

- Directorios en UNIX
  - ¿Para qué necesitamos el campo “Longitud del registro”?
    - A partir del resto de campos podemos saber el tamaño total

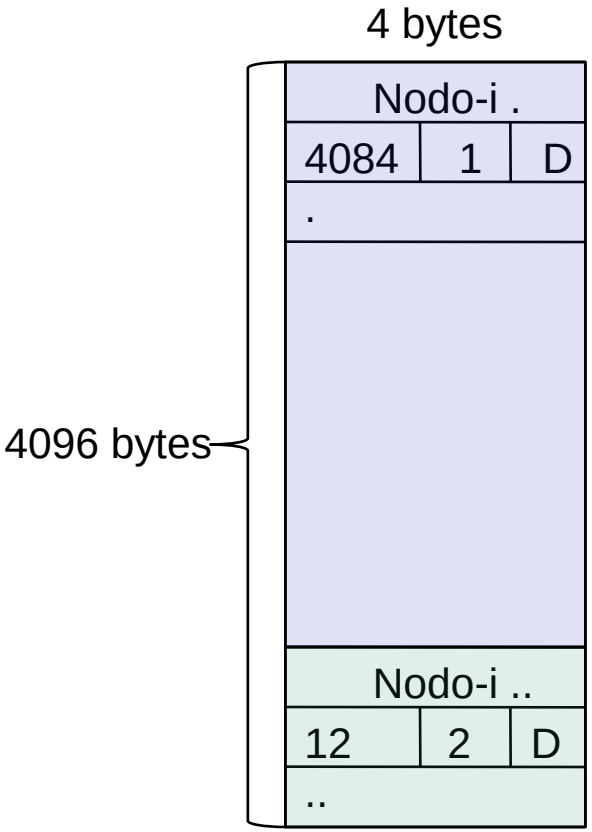


  - Facilita el renombrado de las entradas
  - Para gestionar el espacio libre: al crear un directorio, el registro de la entrada “.” se queda con todo el espacio libre del bloque. Cuando se crear una nueva entrada, se busca un registro con tamaño suficiente
  - Para hacer más fácil el borrado de entradas: se añade el espacio que se libera a la entrada anterior
  - Cada entrada de directorio estará de forma completa en un solo bloque (no puede estar parcialmente en dos bloques)

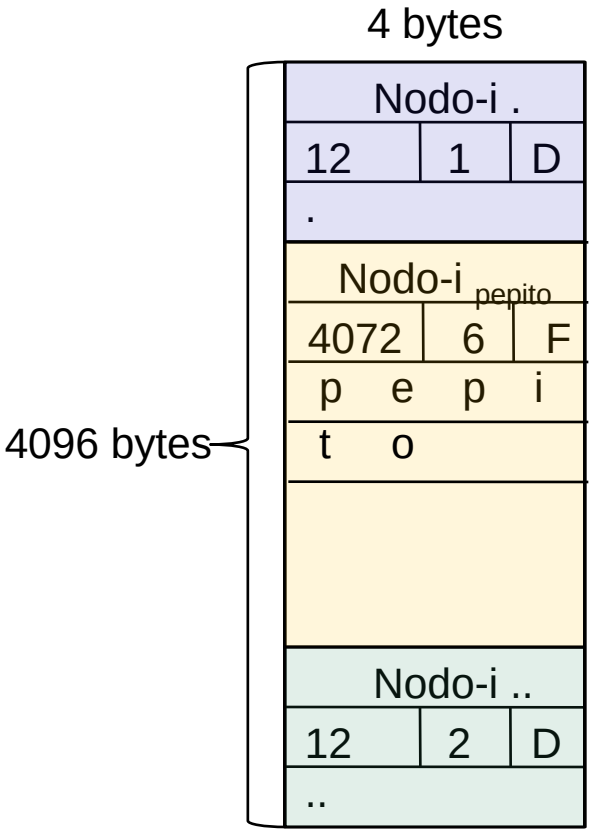
4.4.2 Implementación de directorios

- Directorios en UNIX
  - Ejemplo: Creación de un nuevo directorio y de un fichero dentro de él

1. Creación del directorio



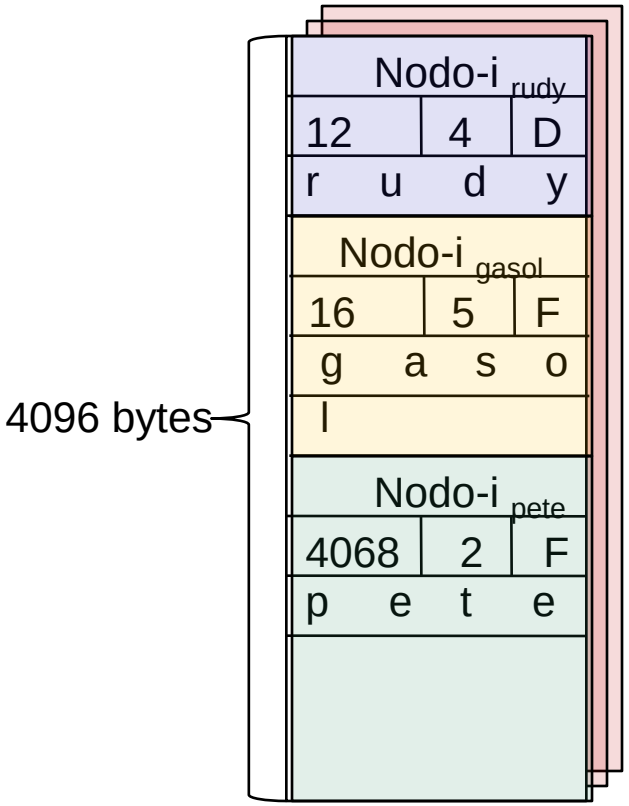
2. Creación del fichero “pepito”



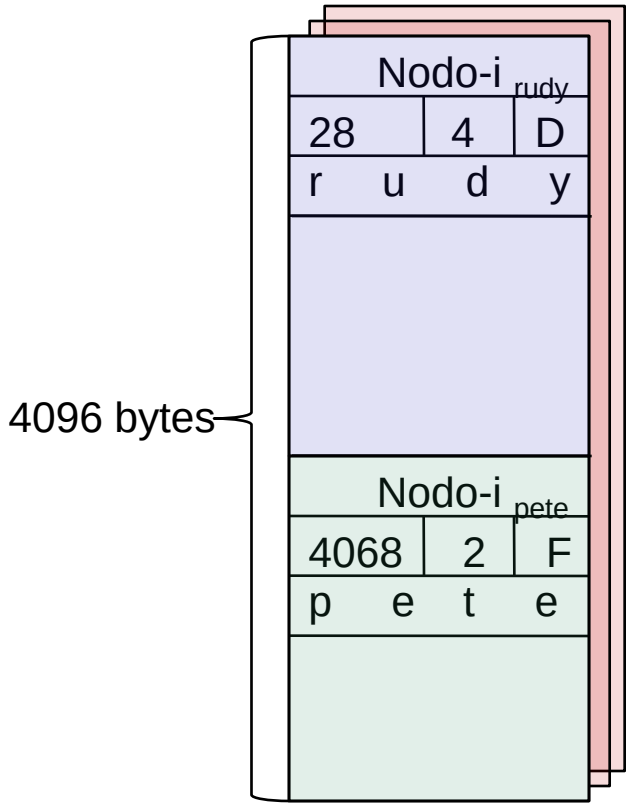
4.4.2 Implementación de directorios

- Directorios en UNIX
  - Ejemplo: Eliminación de un fichero

1. Último bloque del directorio
4 bytes

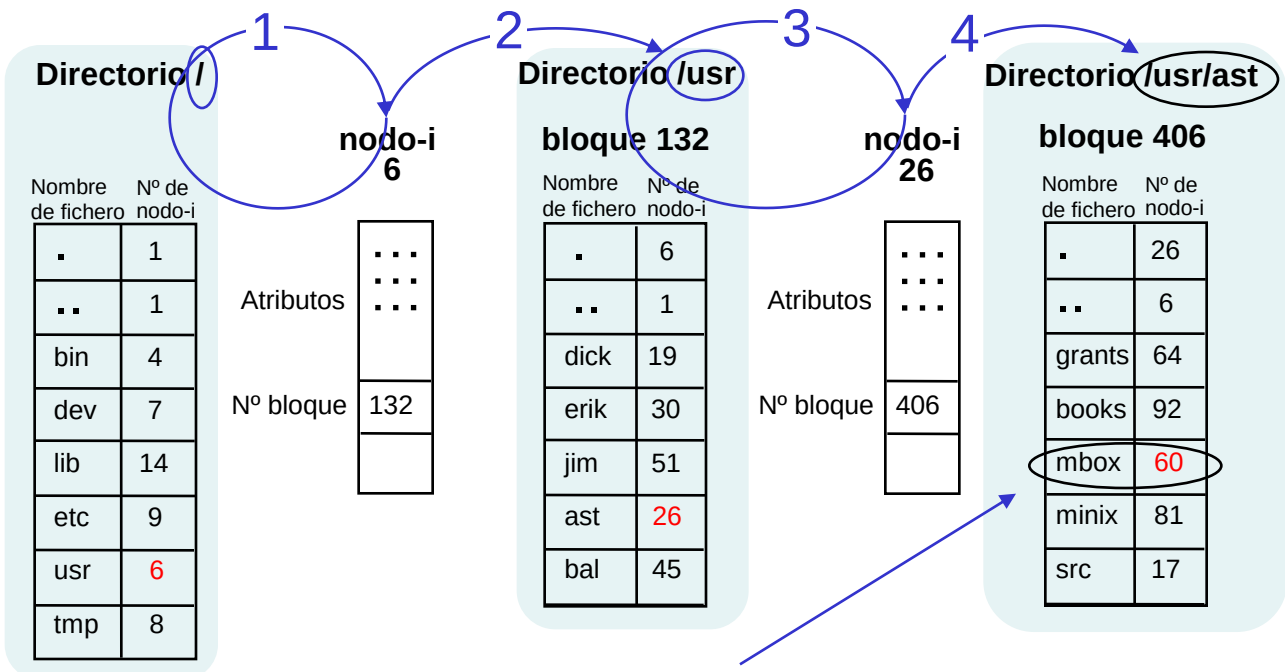


2. Eliminación del fichero “gasol”
4 bytes



4.4.3 Resolución de rutas

- ¿Cómo se resuelve una ruta?
  - Vamos a verlo para la implementación de directorios de Unix (el mecanismo es similar para otros casos)
  - ¿Cómo llegamos al nº de nodo-i de un fichero a partir de su ruta?
  - Por ejemplo: fichero `/usr/ast/mbox`



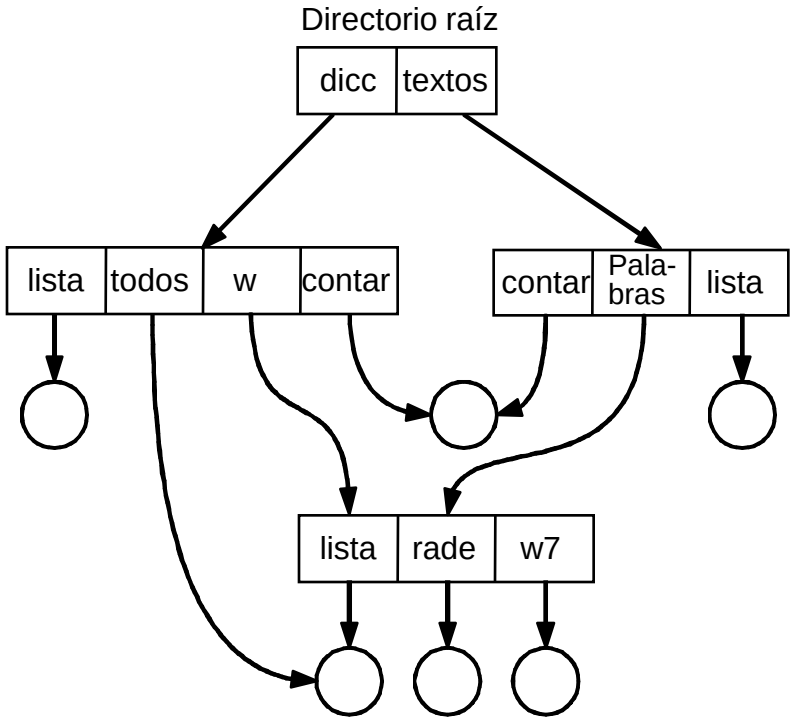
El directorio /usr/ast/mbox tiene asignado el nodo-i 60 (4 accesos a disco, suponiendo que el nodo-i y los bloques del directorio raíz ya están en memoria)

4.4.3 Resolución de rutas

- ¿Cómo se resuelve una ruta?
  - Dado que la resolución de rutas es algo muy frecuente y se suelen repetir las rutas, el SO usa una «caché» para guardar el resultado de la resolución y así acelerar el proceso (en Linux: *dentry cache*)
  - Las rutas relativas se procesan de la misma manera salvo que se comienza en el directorio actual
    - Los directorios «.» y «..» no tienen tratamiento especial

4.4.4 Ficheros compartidos

- Son ficheros con más de una entrada de directorio, dentro de un mismo directorio (nombres distintos) o en directorios distintos
  - *Enlace*: conexión entre un directorio y un fichero compartido
- 2 implementaciones básicas:
  - Enlaces físicos (*hard links*)
    - Necesitan estructuras tipo nodo-i
    - Dos entradas de directorio apuntan a un mismo nodo-i
    - Se necesita un contador de enlaces en el nodo-i
  - Enlaces simbólicos (*soft links*)
    - Ficheros especiales de tipo «link» con el nombre de ruta
    - Pueden ser más costosos (en tiempo y espacio) pero son más flexibles
    - Permiten enlazar directorios y ficheros en ≠ sistemas de ficheros



- Problema: peligro de duplicidad de datos (2 o más rutas para el mismo fichero)

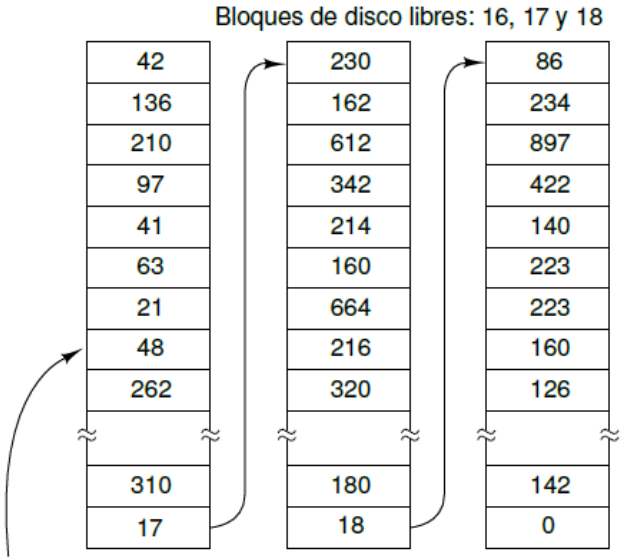
¿Qué ocurre cuando se borra el fichero?

4.4.5 Administración del espacio en disco

- Disco = dispositivo de bloques en el que la unidad mínima de lectura y escritura es el *bloque físico* (sector)
- El SO suele agrupar sectores para formar *bloques lógicos* mayores
- ¿Qué **tamaño de bloque lógico** usar?
  - Suele ser múltiplo del tamaño del bloque físico
  - A la hora de elegir un tamaño de bloque lógico hay que buscar un equilibrio razonable entre:
    - **Eficiencia en el uso del espacio**: a mayor tamaño de bloque, más espacio de disco desperdiciado, principalmente, por fragmentación interna. A menor tamaño, mayor número total de bloques, y por tanto, más punteros (más bloques indirectos) y más grandes
    - **Rendimiento:** A mayor tamaño de bloque, mayor tasa de transferencia en las lecturas/escrituras de disco, y menos bloques indirectos (menos accesos a disco)

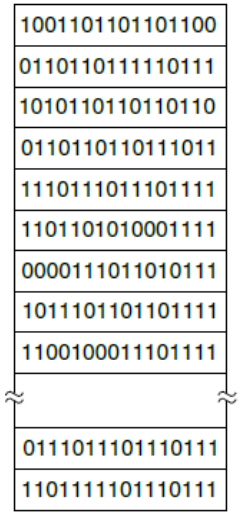
4.4.5 Administración del espacio en disco

- **Registro de bloques libres**:
  - Debemos saber qué bloques lógicos del disco están libres y cuáles no:
    - **Lista ligada de bloques** (a): cada bloque de la lista tiene nº de bloques libres (tantos como pueda) y un puntero al siguiente bloque de la lista libres agrupados
      - Los bloques que contienen la lista son bloques libres (no asignados) y pueden usarse tan pronto como dejen de contener información útil



4.4.5 Administración del espacio en disco

- **Registro de bloques libres**:
  - Debemos saber qué bloques lógicos del disco están libres y cuáles no:
    - **Mapa de bits** (b): se almacena en disco un mapa de N bits (1 por bloque). Valor 1 → Bloque libre; Valor 0 → Bloque en uso
      - Suele ocupar menos espacio que la opción (a) y permite buscar de forma rápida grupos de bloques libres consecutivos en disco
      - Alternativa empleada en sistemas de ficheros actuales como Ext4 y NTFS



Un mapa de bits

(b)

Índice

4.5 Caché de disco

- Los dispositivos sobre los que se construyen los SSFF (p.e. discos) suelen ser mucho más lentos que la memoria principal
- Para mejorar su rendimiento, la solución más común es utilizar una porción de la memoria principal para almacenar bloques de disco, conocida como caché de disco (o, también, «caché de bloques» o «caché de buffers »)
  - Funcionamiento similar al de cualquier caché *writeback*
    - Se busca primero en la caché de disco. Si está, la solicitud se completa sin acceder a disco. Si no, se accede a disco y se inserta el bloque en caché
  - Algoritmo de reemplazo LRU con listas ligadas: en la cola de la lista siempre están los elementos que hace más que no se usan
    - No es buena idea usar LRU puro de cara a la consistencia del SFF ya que no todos los bloques son iguales (por ejemplo, nodos-i)
    - La consistencia del sistema de ficheros se compromete si, por ejemplo, se apaga este y hay nodos-i en caché de disco que han sido modificados

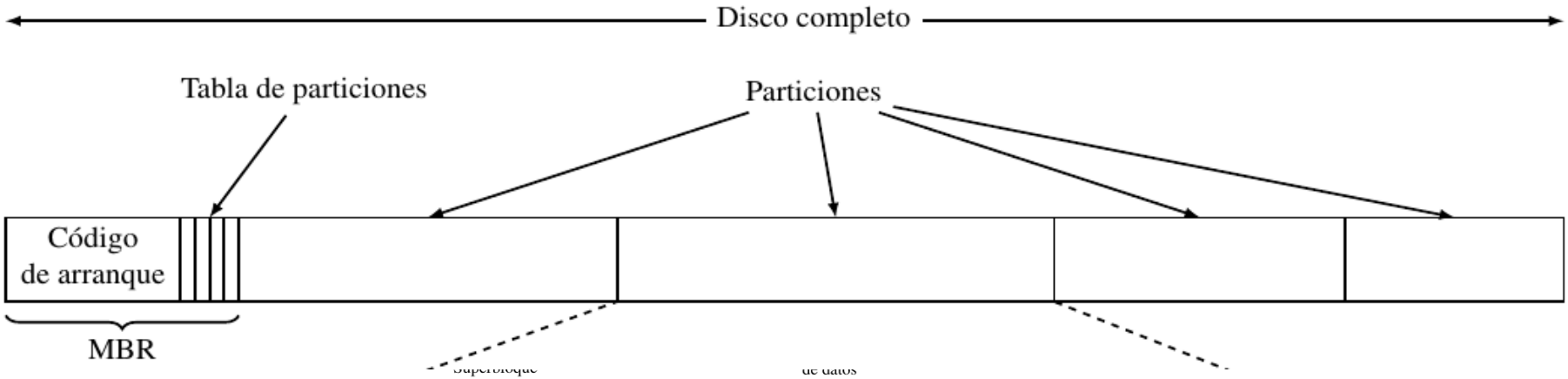
4.5 Caché de disco

- **Solución**: LRU modificado que tiene en cuenta dos factores independientes:
  - ¿Es probable que el bloque se vuelva a necesitar pronto?
    - Si **no** es así (por ejemplo, bloque indirecto), colocar directamente el bloque al final de la lista LRU con el fin de que se recicle rápido
  - ¿Es esencial el bloque para la consistencia del SFF?
    - Si **sí** es así (por ejemplo, nodo-i o bloque del mapa de bits) y ha sido modificado, escribir en disco lo antes posible sin importar su posición en la lista (en Unix, antes de 5 seg)
    - Tampoco es recomendable tener bloques de datos mucho tiempo sin escribir (en Unix, cualquier bloque de datos modificado se escribe en disco a lo más tardar en 30 seg)
    - A través de la orden `sync` de Unix el usuario puede forzar escrituras a disco de todos los bloques modificados

Índice

4.6 Discos y sistemas de ficheros

- Es habitual que un disco contenga varios sistemas de ficheros
- Los SSOO crean el concepto de partición para hacer esto posible:
  - **Partición**: porción de bloques consecutivos en un disco
  - Son manejadas por el SO que las "ve" también como un array lineal de bloques lógicos (empieza por el 0 y acaba por el N-1)
  - **Tabla de particiones**: contienen el número y tamaño de las particiones existente, junto con su bloque de inicio y fin
    - Se suele almacenar en el bloque 0 del disco (*MBR*)



4.6 Discos y sistemas de ficheros

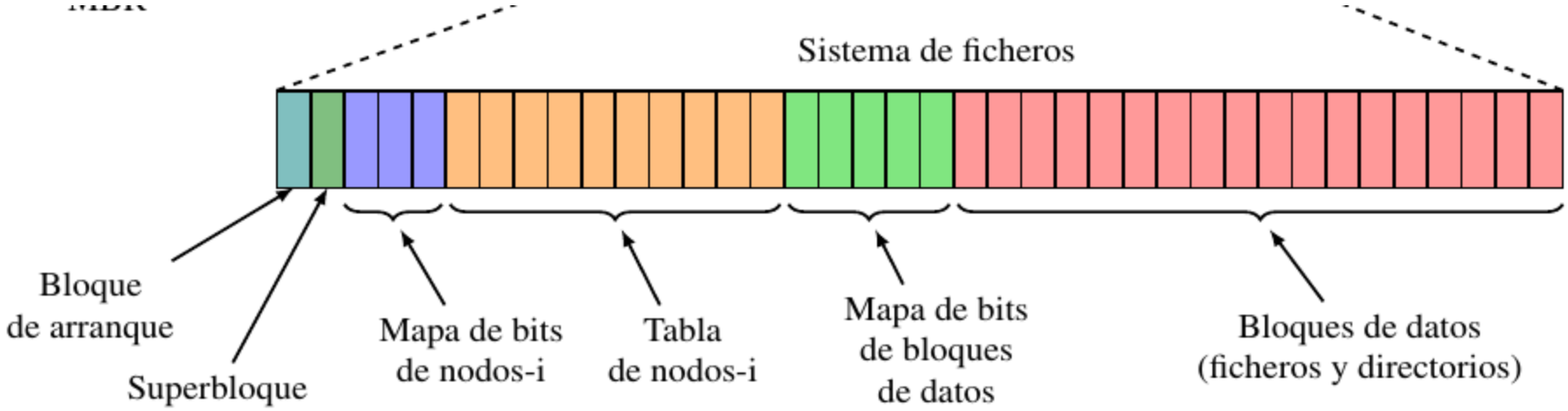
Particiones. Razones uso:

- Varios Sistemas Operativos

- Varios Sistemas de Ficheros
  - Sistema de ficheros raíz ext4
  - Partición de intercambio con formato especial

- Diferentes usos:
  - Sistema de ficheros raíz
  - Sistema de ficheros para datos de usuario
  - Partición de intercambios

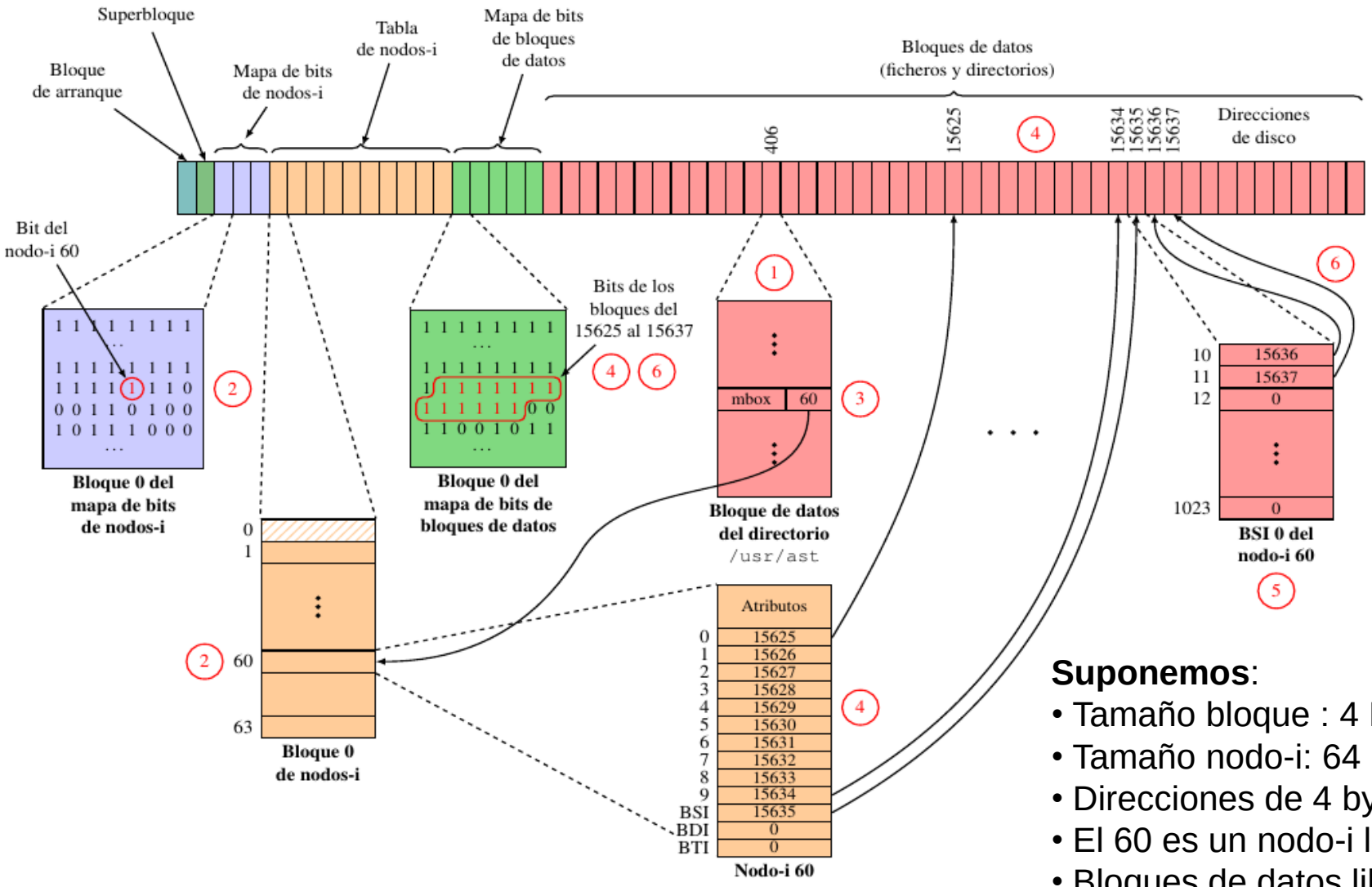
4.6 Discos y sistemas de ficheros



Bloque arranque: Si se usa, código de arranque del SO (bloque 0 de la partición) Dado que el SFF no lo usa, 0 queda libre como valor no válido
Superbloque: información organización del SF (nº de bloques lógicos, nº nodos-i, tamaño mapas de bits, …) (bloque 1 de la partición y copias en otros)
Mapa de bits de bloques: $B / 8 \times T_B$ bloques
Mapa de bits de nodos-i: $I / 8 \times T_B$ bloques
Nodos-i: $I \times T_I / T_B$ bloques
Resto de bloques: bloques datos (ficheros)+BLOQUES INDIRECTOS

4.6 Discos y sistemas de ficheros

Ejemplo: Creación del fichero `/usr/ast/mbox` de tamaño 45KiB



Suponemos:

- Tamaño bloque : 4 KiB
- Tamaño nodo-i: 64 bytes
- Direcciones de 4 bytes
- El 60 es un nodo-i libre
- Bloques de datos libres entre el 15625 y el 15639