# Promoción 2

## Memoria Virtual

Hay partes de un proceso que no necesitan estar disponibles en la memoria en todo momento, como por ejemplo una sección de programa que se ejecuta una o ninguna vez, o regiones de memoria que se alocan dinámicamente y luego se liberan. Por lo tanto no es necesario que todo el  proceso se cargue a memoria.

Se traen partes del proceso a medida que se las necesita. Estas partes se llaman **conjunto residente (working set).** El hardware detecta cuando se necesita una parte del proceso que no está en el working set. Si no lo está, se carga para continuar su ejecución.

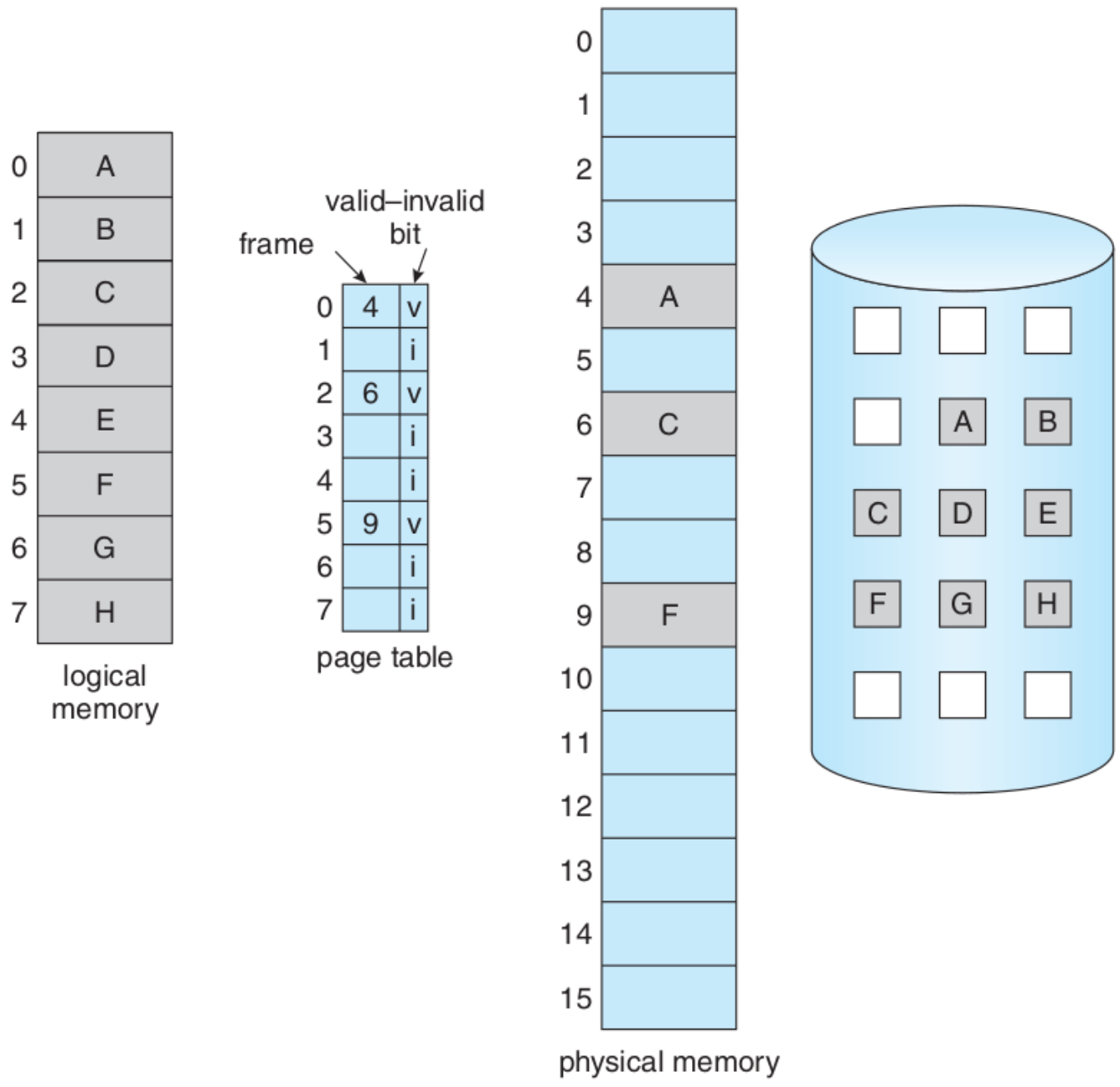
- Más procesos en memoria.
- Un proceso puede ser más grande que la memoria principal.
- El programador y usuario no se preocupan por la memoria utilizada.

La limitación de la memoria la impone el Hardware y el bus de direcciones.

### Requerimientos

- El hardware debe soportar paginación/segmentación bajo demanda.
- Memoria secundaria para almacenar las secciones de los procesos no alocados en memoria principal.
- Soporte del sistema operativo para manejar páginas o segmentos entre memoria principal y secundaria.

## Paginación (bajo demanda) con Memoria Virtual




Bajo este esquema, cada proceso tiene su propia tabla de páginas, donde cada entrada hace referencia al frame en el que se encuentra la página en la memoria principal. Además, tienen bits de control: el bit **válido/inválido** indica si la página está en la memoria (escribe SO, lee HW) y el **bit modificado**, que indica si la página fue o no modificada (lee SO, escribe HW). El tamaño de la tabla de páginas depende del espacio de direcciones que ocupe el proceso, y puede alcanzar un tamaño considerable.

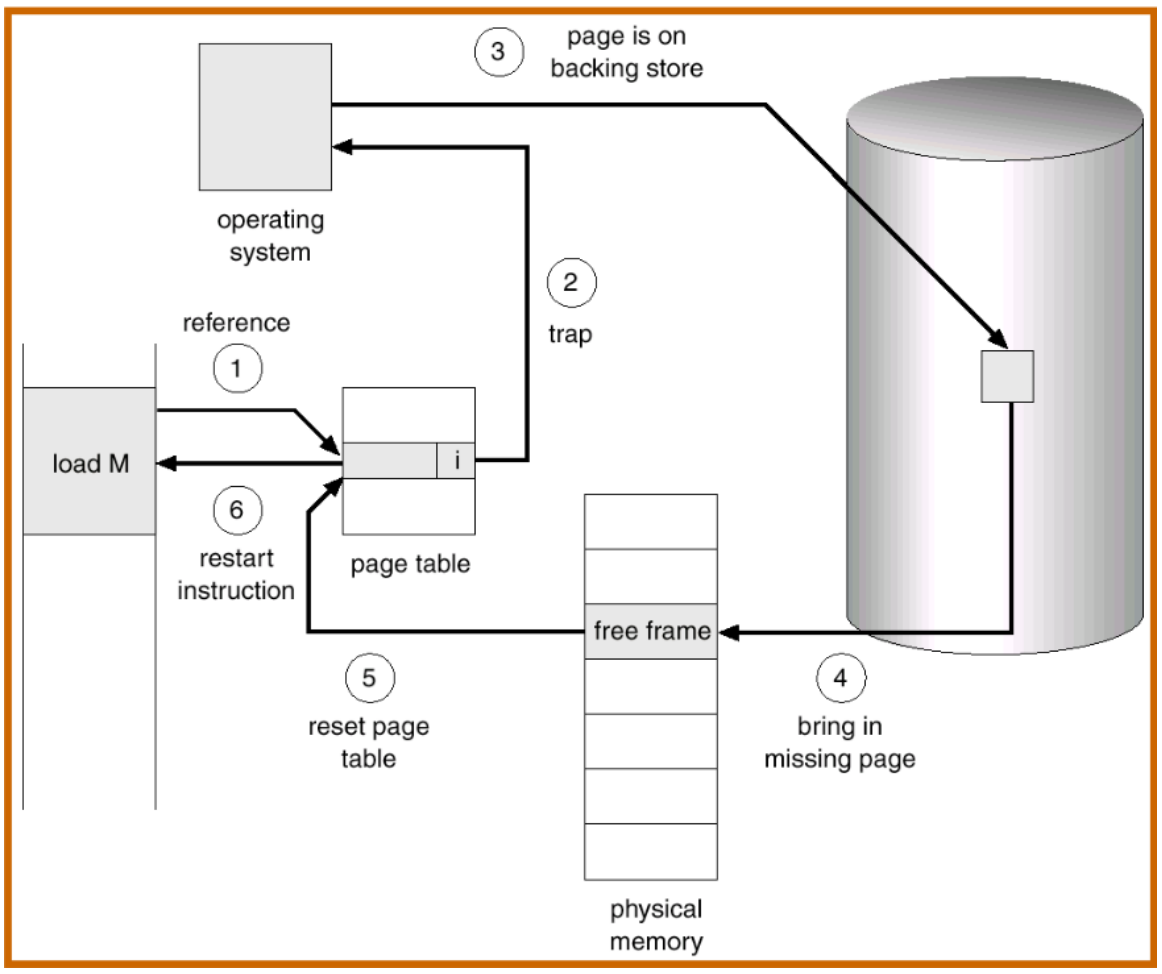
# Page Fault (con MV)

Ocurre cuando un proceso intenta acceder a una dirección que no se encuentra en el working set. Cuando se da, el hardware lo detecta y genera un trap al sistema operativo, el cual pone al proceso en estado de blocked y gestiona la carga de la página requerida.

Para ello busca un frame libre en memoria, y hace una operación de entrada/salida al disco para copiar la página el el frame. Mientras esto ocurre, se le cede CPU a otro proceso hasta que la operación termine (ocurre un context switch). Una vez terminada, el sistema operativo actualiza la tabla de páginas poniendo un 1 en el bit válido y la dirección base del frame donde se aloca la página.

> 💡 La tasa de page faults tiende a aumentar cuando el tamaño de página es **intermedio**

El proceso que generó el fault vuelve a estado ready, y cuando se ejecute lo hará desde la instrucción que generó el fault.




La idea de la paginación es alocar la mayor cantidad de páginas posibles para aumentar el grado de multiprogramación. Cada vez que se aloca una página en un marco, se da el page fault. Si se necesita alocar una página y no hay frames liberados, el sistema operativo selecciona una página víctima mediante alguno de los tantos algoritmos.

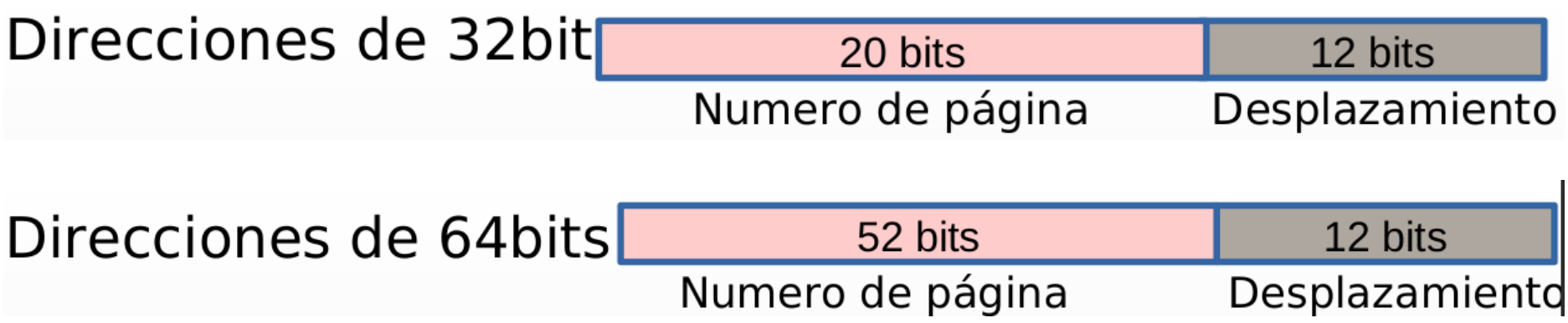
La performance de estos algoritmos se mide en la cantidad de page faults que producen. Mayor performance implica menor cantidad de page faults, y vice-versa.

# Tabla de Páginas

Utiliza correspondencia directa. Existen tres formas de organizarla: tabla única lineal (un nivel), tabla multinivel y tabla de hashing.

> 💡 La tabla de páginas es parte del **contexto**

## Tabla Única Lineal



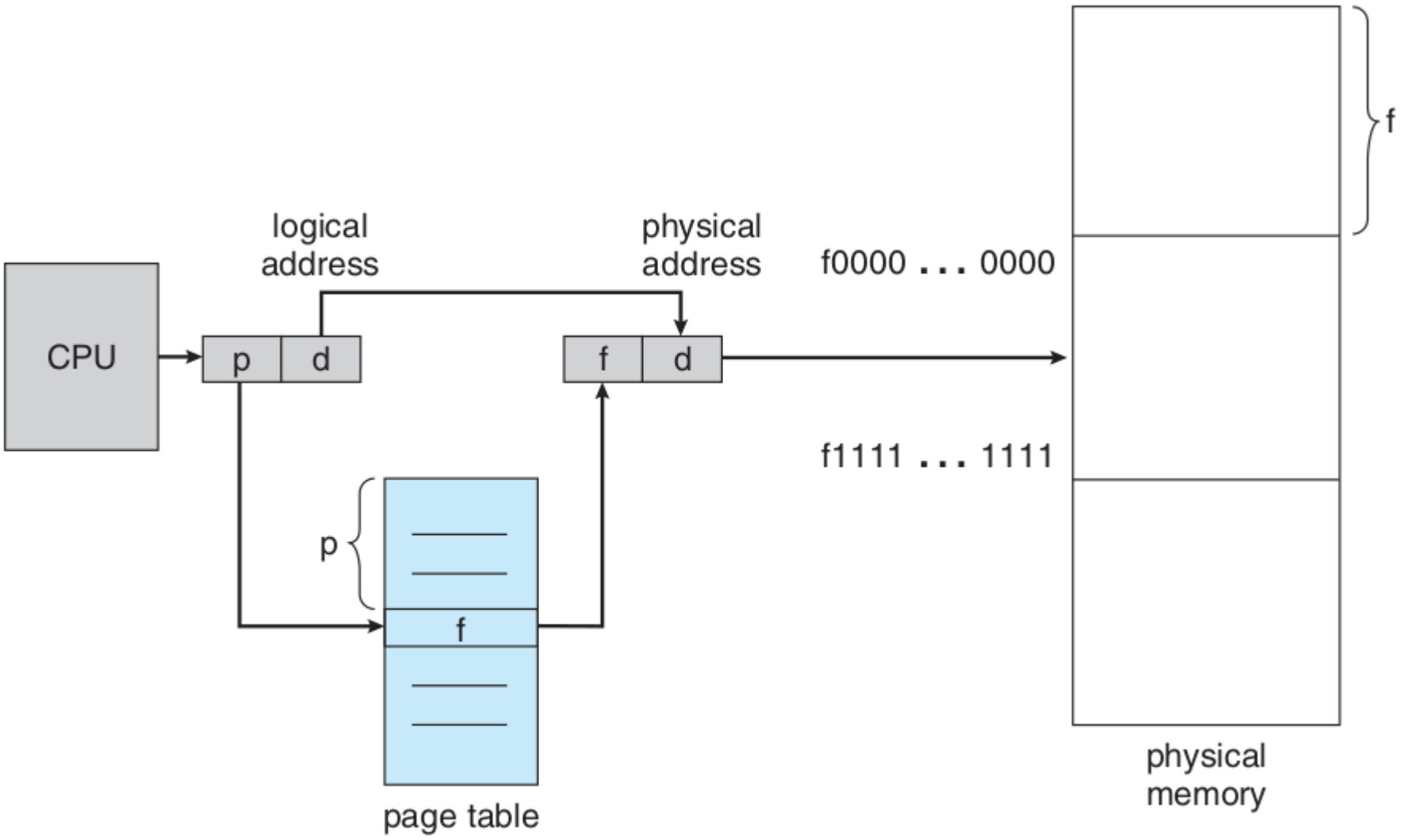



**Figure 8.10** Paging hardware.

Las tablas lineales tienen la desventaja de utilizar demasiada memoria, pero son de rápido acceso.

## Tabla Multinivel

Alocar una tabla única contiguamente en memoria es mala idea, por lo que surge la idea de dividir la tabla de páginas en otras tablas, y así paginar la tabla de páginas.

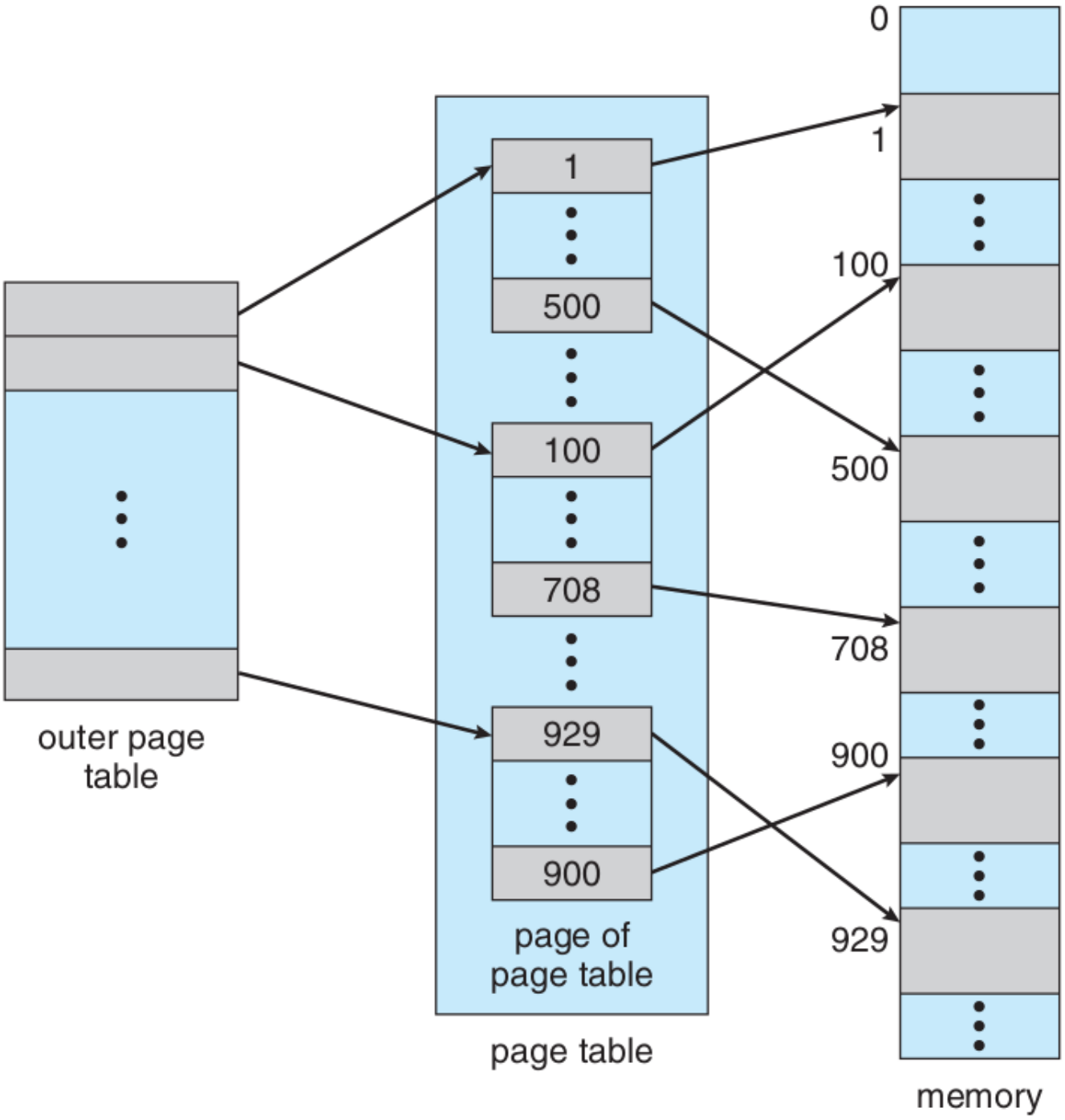
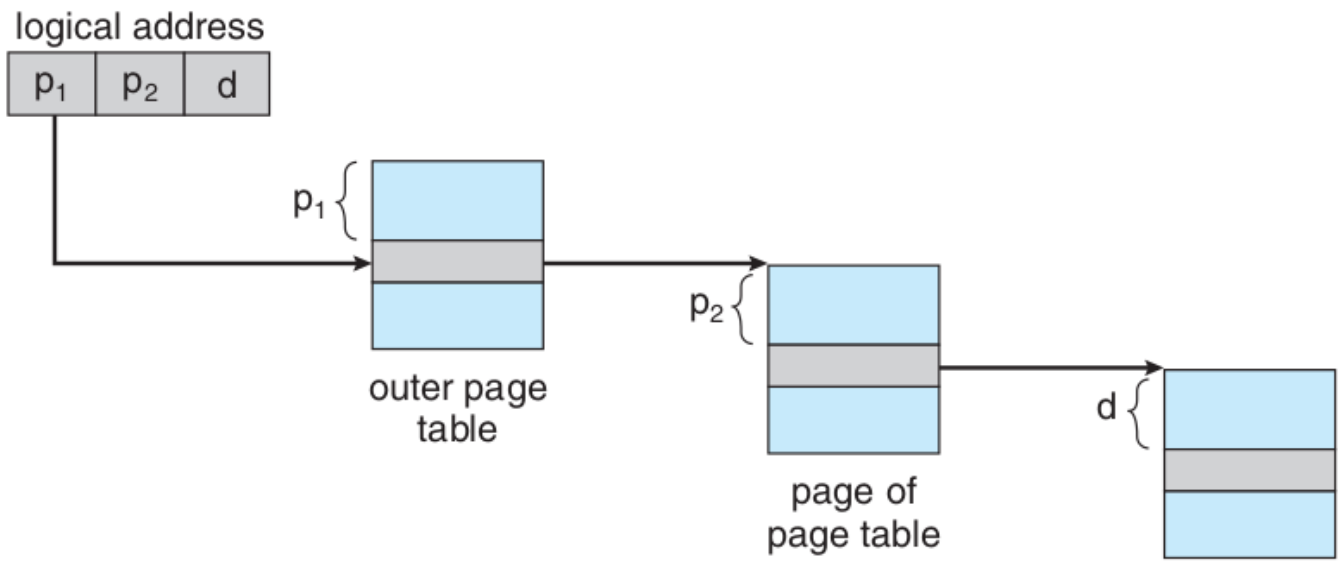



Tabla de dos niveles

La dirección de memoria tendrá un nivel más.

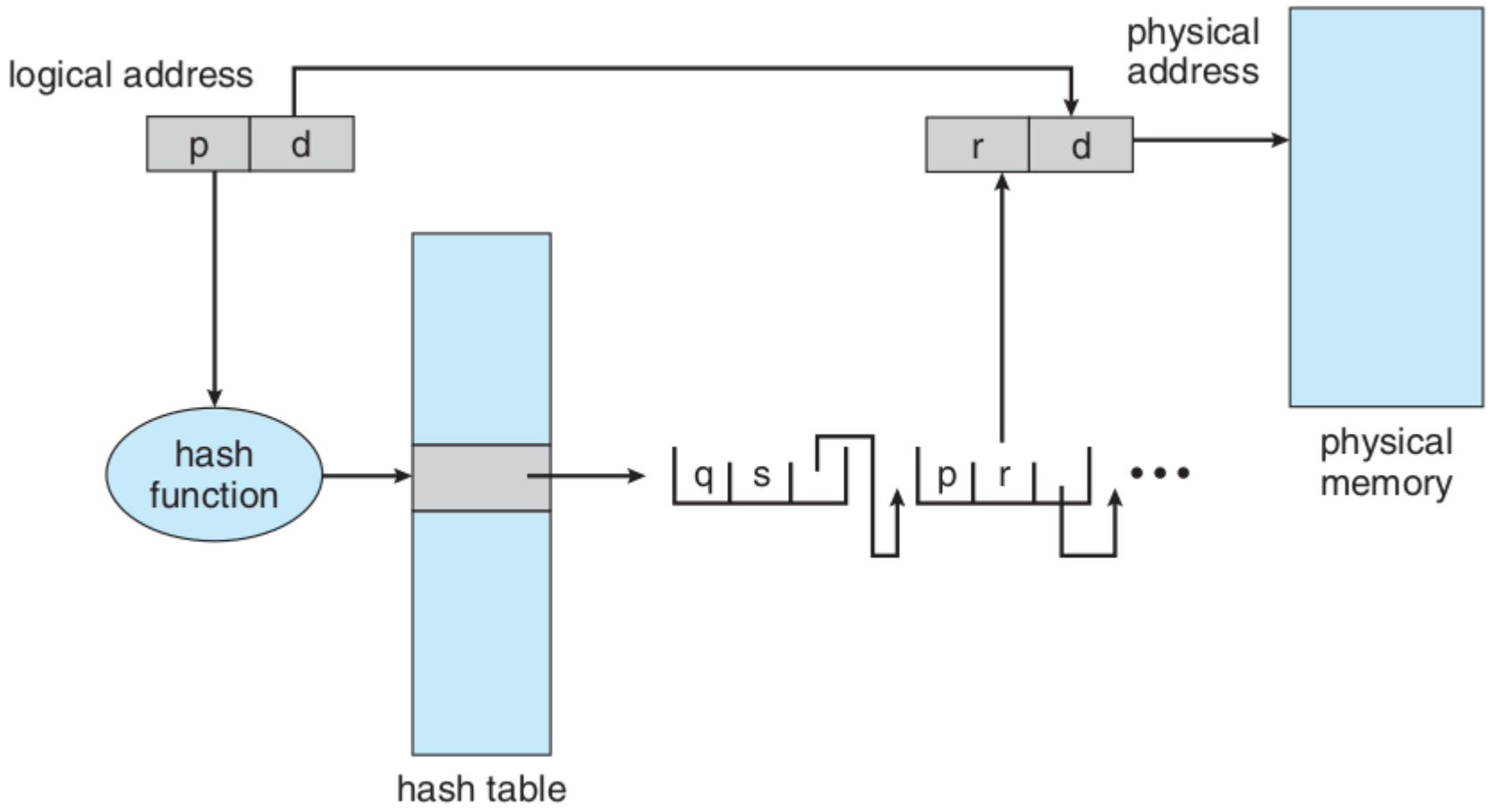



Las tablas multinivel se consideran inapropiadas ya que para reducir de forma considerable el espacio de las páginas y la tabla, hay que dividir en muchos niveles. Esto hace más complejo el acceso, ya que habría que ir traduciendo la dirección nivel por nivel.

## Tabla de Hashing

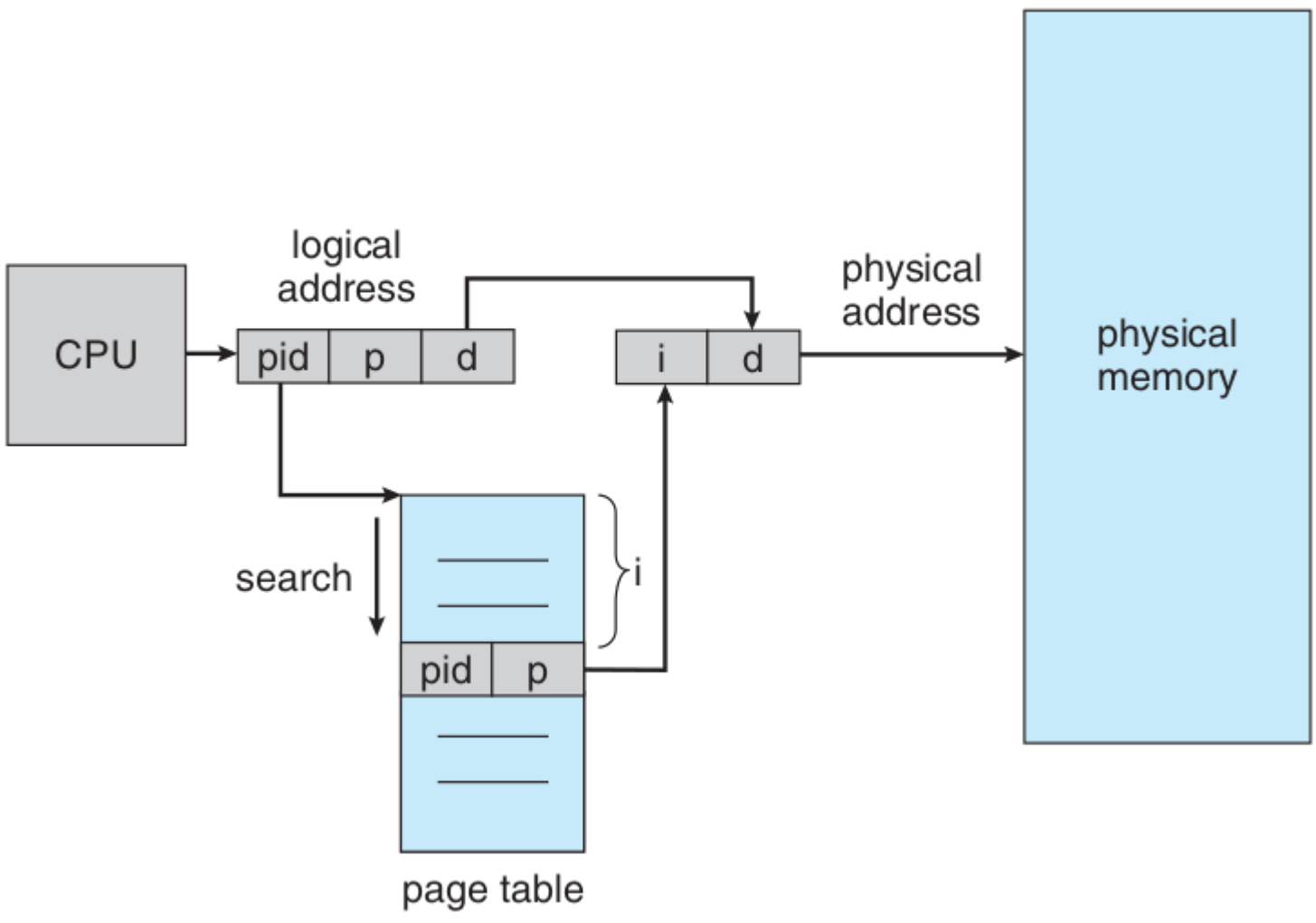
El procedimiento con tabla de hashing consiste en asociar una página a un valor de tabla cuyo índice se obtiene con la función de hashing.

> 💡 Si la tabla no provee una buena dispersión, la resolución de una dirección podría requerir de más de un acceso a memoria.

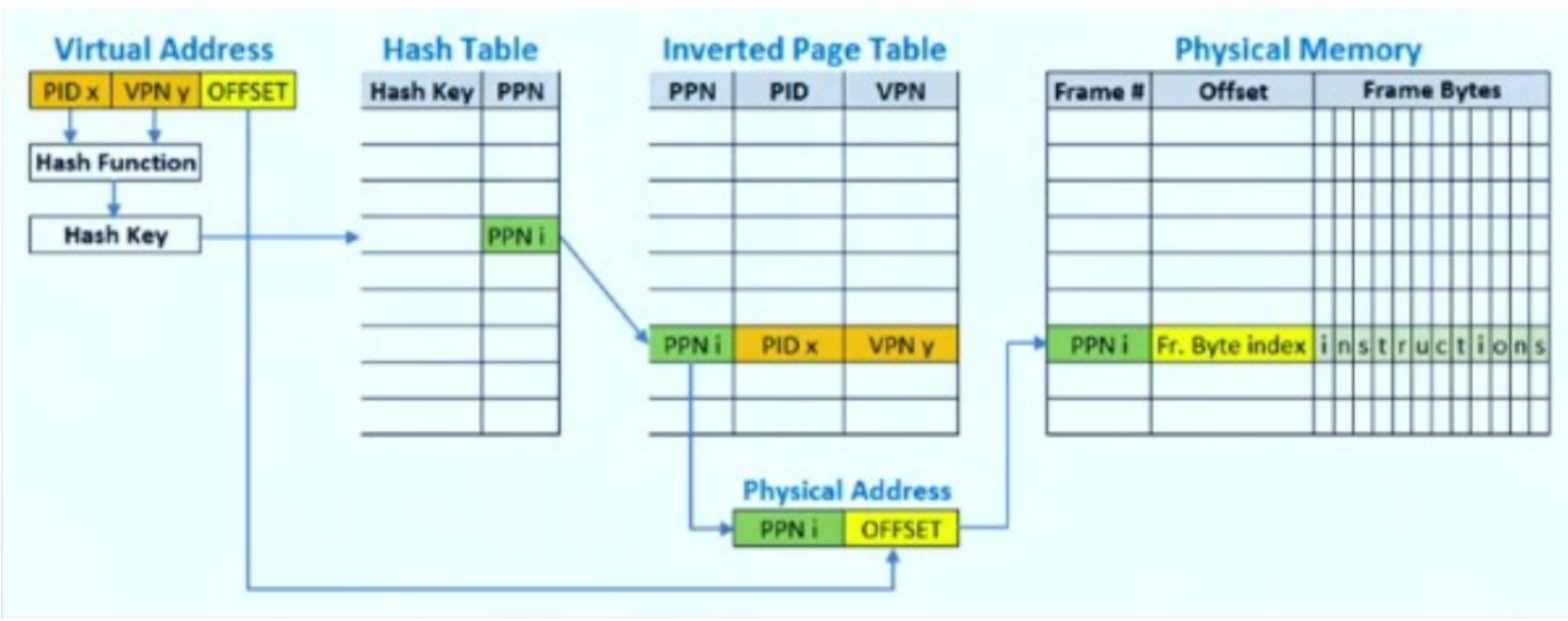



## Tabla Invertida

Una sola tabla donde hay una entrada por marco existente en la memoria real. Esto quiere decir que por cada espacio de direcciones en el disco físico, habrá una entrada en la tabla.




La tabla invertida se puede aplicar con hashing, donde el número de página se transforma en un valor de hash y así hallar el marco asociado. También se implementa un mecanismo de encadenamiento para solucionar colisiones (mismo hash para 2 direcciones virtuales).




| | Tabla Normal | Tabla Invertida |
|---|---|---|
| Entrada | Una entrada por cada por cada página del proceso en memoria virtual | Una entrada por cada frame en la memoria física |
| Mapeo | Cada entrada contiene información sobre la página correspondiente en memoria virtual y su frame. Traduce dirección virtual a física. | Cada entrada contiene información sobre la página en memoria virtual, mapeada al frame correspondiente. Traduce de física a virtual. |
| Uso de memoria | Proporcional al tamaño del espacio virtual. Más memoria, más larga. | Proporcional a la cantidad de frames en memoria física, causando que menos memoria sea utilizada. |

# Tamaño de las Páginas

**Páginas pequeñas:**

- Menor fragmentación interna.
- Más páginas requeridas por proceso.
- Tablas de páginas más grandes.
- Mayor cantidad de páginas en memoria.

**Páginas grandes:**

- Mayor fragmentación interna.
- Mayor velocidad de transferencia de páginas entre memoria secundaria y principal (la memoria secundaria está diseñada para transferir bloques grandes de manera eficiente).

# Translation Look-aside Buffer

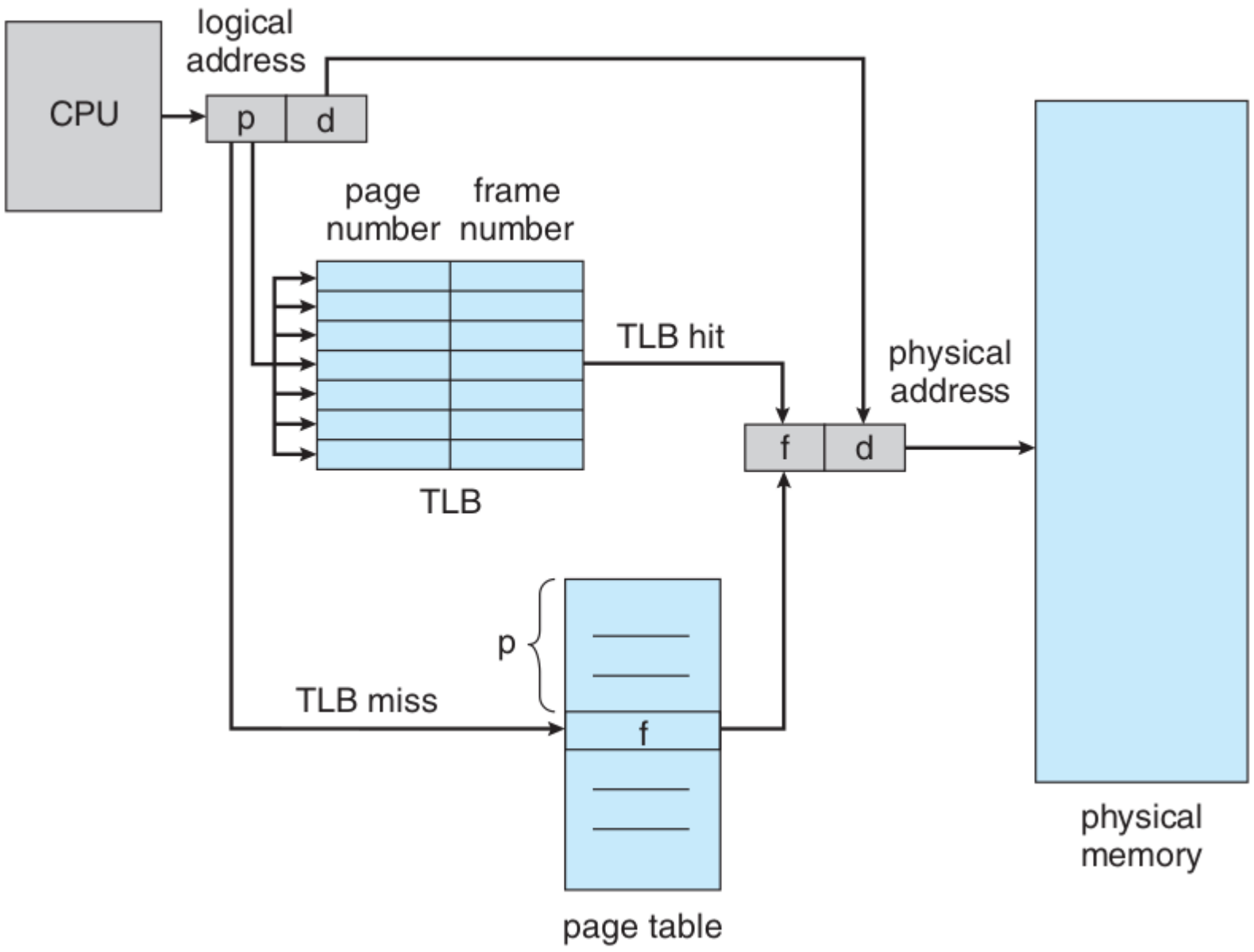
Para acceder a una dirección de memoria, debemos fijarnos en la tabla de páginas (un acceso) y luego ir hacia la dirección que la tabla nos indica (otro acceso), lo que causa un tiempo de acceso muy largo. La solución a este problema es una memoria caché rápida llamada Translation Look-aside Buffer (TLB).

Cada entrada a este buffer funciona como diccionario (key-value). Cuando al TLB le piden un valor, la key es comparada con todas las keys existentes simultáneamente y al hallar la correspondiente la retorna.

Para mantener la eficiencia, el TLB debe mantenerse reducido. Muchos sistemas actuales mantienen muchos niveles de TLBs.

Puede ocurrir que la key requerida no esté en el TLB (TLB miss), por lo que se produce un page fault. Además, podemos incorporar la página que se necesita al buffer mediante hardware o un trap al sistema operativo (depende del tipo de CPU) para reducir el tiempo de accesos futuros.

💡 El acceso al TLB es mediante el Hardware




# Asignación de Marcos

La asignación de marcos puede ser **dinámica** o **fija**. Si es dinámica, el número de marcos por procesos varía en tiempo de ejecución. Si es fija, no varía.

La asignación fija puede ser equitativa (se dividen entre la cantidad de procesos los marcos disponibles) o proporcional (se asignan los marcos acorde al tamaño de cada proceso).

# Reemplazo de Páginas

Hay varios algoritmos para seleccionar una página víctima. El reemplazo puede ser global o local.

Con **reemplazo global**, el fallo de página de un proceso puede reemplazar la página de cualquier otro, aumentando la cantidad de frames asignados a él. Si un proceso es de alta prioridad, puede tomar los frames de otro proceso con menor prioridad. El sistema operativo no controla la tasa de page faults por proceso.

Con **reemplazo local**, el fault de un proceso solo reemplaza sus marcos asignados (cantidad de frames constantes). Puede tener frames asignados que no utiliza, malgastando memoria. Con reemplazo local el sistema operativo puede saber cuál es la tasa de faults por proceso.

# Algoritmos de Reemplazo

- **Óptimo:** teórico.
- **FIFO:** first in first out, más sencillo de aplicar.
- **LRU:** soporte del hardware para mantener timestamps de acceso a las páginas. Favorece a las páginas más recientemente accedidas.
- **Second Chance:** FIFO con second chance bit, favorece a las páginas más referenciadas.
- **Non Recently Used:** utiliza bits R y M, favorece a las páginas más recientemente referenciadas.

# Thrashing

Un proceso hace thrashing si pasa más tiempo paginando que ejecutándose.

In fact, look at any process that does not have "enough" frames. If the process does not have the number of frames it needs to support pages in active use, it will quickly page-fault. At this point, it must replace some page. However, since all its pages are in active use, it must replace a page that will be needed again right away. Consequently, it quickly faults again, and again, and again, replacing pages that it must bring back in immediately.

Es posible controlar el thrashing mediante algoritmos de reemplazo locales, para que no robe frames a otros procesos. Implementando un algoritmo local bajamos el rendimiento del sistema pero podemos controlar este fenómeno.

# Modelo de Localidad

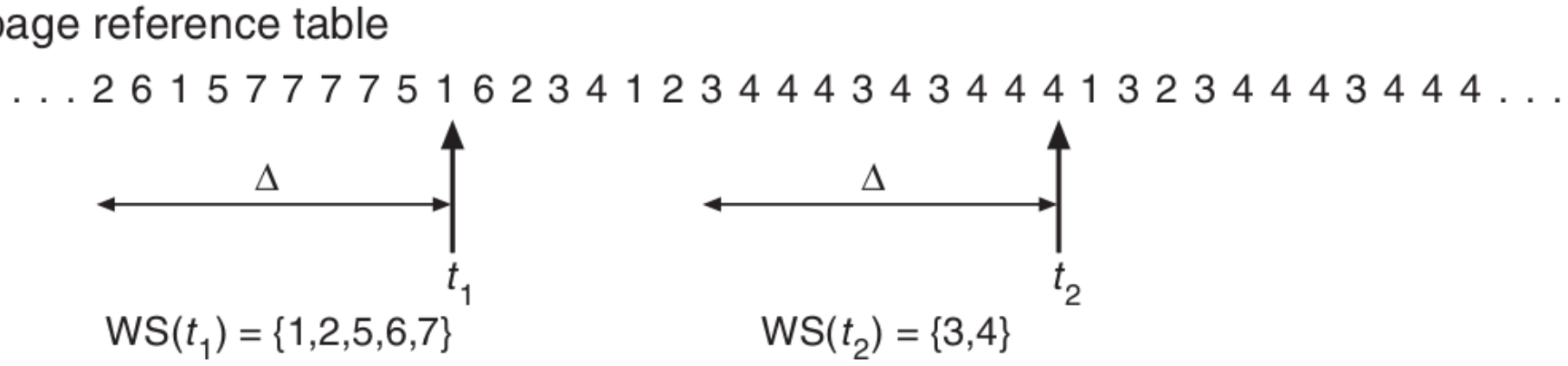
El modelo de localidad nos dice que mientras un proceso se ejecuta, se mueve entre localidades. Una localidad es un conjunto de páginas que se utilizan de manera conjunta por el proceso, y se pueden superponer.

For example, when a function is called, it defines a new locality. In this locality, memory references are made to the instructions of the function call, its local variables, and a subset of the global variables. When we exit the function, the process leaves this locality, since the local variables and instructions of the function are no longer in active use. We may return to this locality later.

Un proceso hará fallos de página hasta que la localidad que necesite usar esté completamente en memoria, y cuando la esté, no habrán más fallos por un tiempo. Si no se puede traer la localidad completa a memoria, provocará thrashing.

# Working Set ($\Delta$)

El modelo de working set se basa en el principio de localidad. Es el conjunto de páginas que tienen las más recientes $\Delta$ referencias a páginas. Uno de los aspectos más importantes del working set es el tamaño, es decir, la cantidad de páginas que puede albergar.

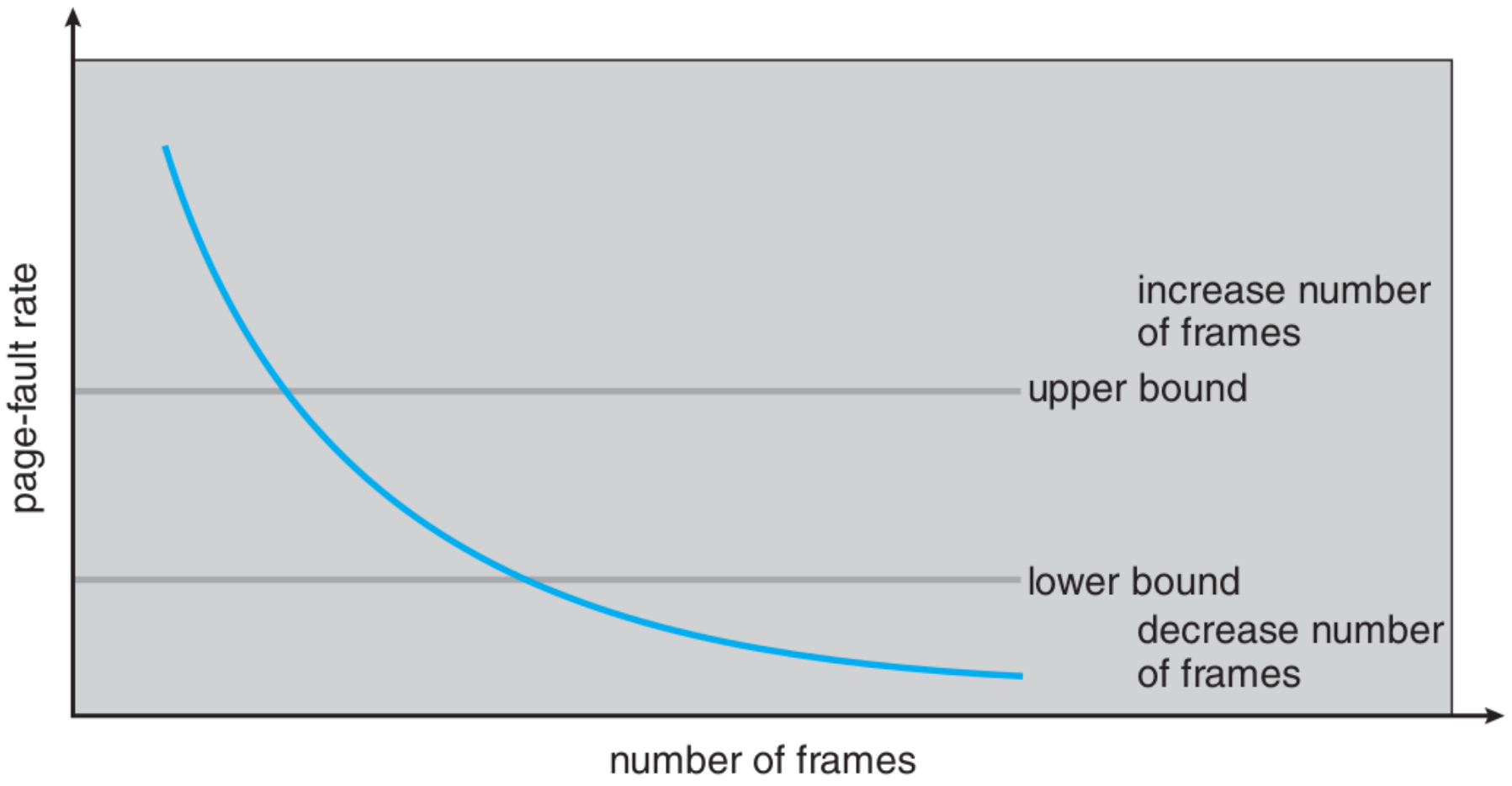



$\Delta$ = 10

En cuanto a la elección del delta, si tomamos uno chico no llegará a cubrir una localidad entera, y si tomamos uno muy grande cubrirá más de una localidad. Uno de los principales problemas del working set es tener noción de su ubicación en la memoria, ya que la ventana es móvil.

# Page Fault Frequency (PFF)

Esta técnica de prevención de thrashing utiliza la tasa de fallos de página para estimar si el tamaño del conjunto residente es el adecuado. Si la PFF es alta se necesitan más frames, y si es baja los procesos tienen frames demás.




# Demonio de Paginación

El demonio de paginación es un proceso creado por el sistema operativo durante el arranque, y su objetivo es ayudar en la administración de memoria. Se ejecuta cuando hay poca memoria libre o demasiada memoria modificada.

- Limpia páginas modificadas para reducir el tiempo del próximo swapeo y hacer más rápido el tiempo de transferencia al sincronizar páginas contiguas.
- Mantiene algunos marcos libres en la memoria.
- Demora la liberación de una página hasta que realmente haga falta.

# Memoria Compartida

Los procesos pueden compartir un marco de memoria. Para ello, este marco debe estar asociado a una página en la tabla de páginas de cada proceso, y este número no debe ser necesariamente el mismo.

Los procesos también comparten una copia de código read-only, sin embargo los datos de cada uno son privados y se encuentran en páginas no compartidas.

# Mapeo de Archivos en Memoria

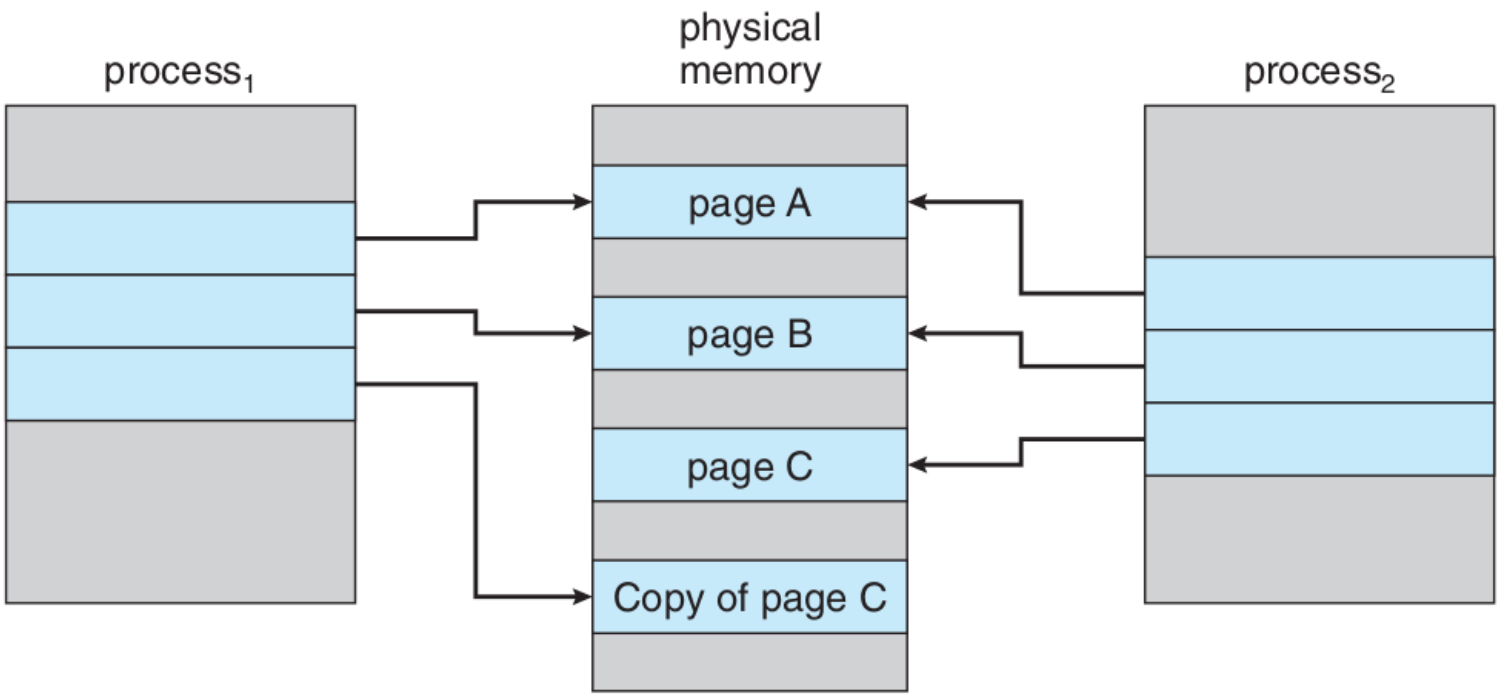
Permite a un proceso asociar el contenido de un archivo a una región de su espacio de direcciones virtual. El contenido del archivo que se sube tiene que ser acorde al tamaño de página (se pueden subir partes del archivo que ocupen más de una página, pero se comprometerán mas espacios de memoria).

La ventaja de esta técnica es que no es necesario realizar operaciones a un archivo mediante un syscall, sino que se toman como accesos a memoria rutinarios, causando que la operación en archivos sea más rápida.

# Copy-on-Write

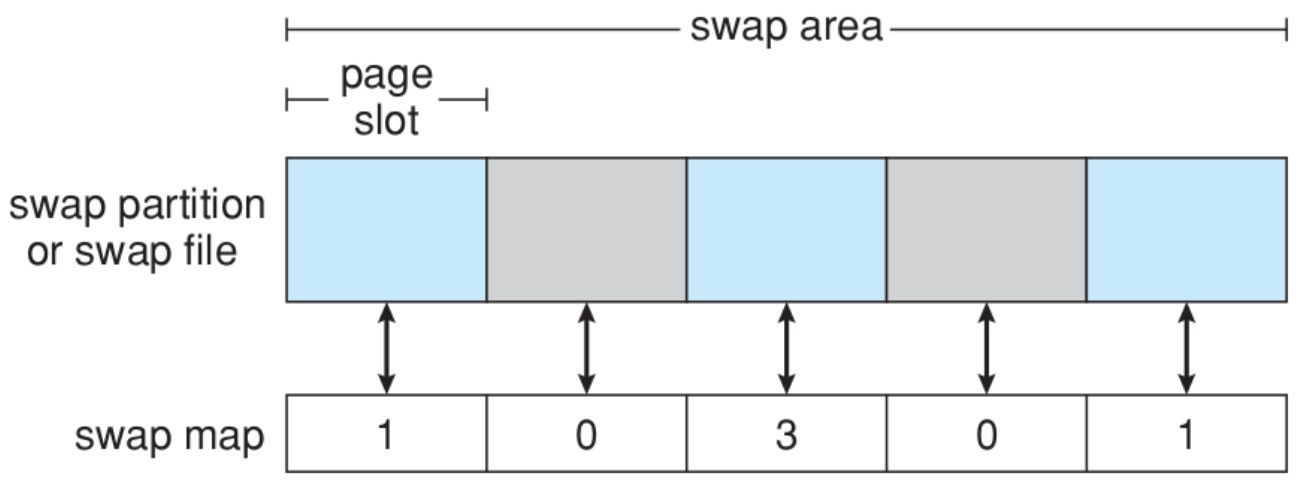
Esta técnica agiliza la creación de nuevos procesos en calls como el fork. Básicamente crea una copia de un archivo solamente cuando es modificado, sino se implementa como un archivo compartido.

Recall that the `fork()` system call creates a child process that is a duplicate of its parent. Traditionally, `fork()` worked by creating a copy of the parent's address space for the child, duplicating the pages belonging to the parent. However, considering that many child processes invoke the `exec()` system call immediately after creation, the copying of the parent's address space may be unnecessary. Instead, we can use a technique known as **copy-on-write**, which works by allowing the parent and child processes initially to share the same pages. These shared pages are marked as copy-on-write pages, meaning that if either process writes to a shared page, a copy of the shared page is created. Copy-on-write is illustrated in Figures 9.7 and 9.8, which show the




## Área de Swap

memory—that is, memory not backed by any file. Linux allows one or more swap areas to be established. A swap area may be in either a swap file on a regular file system or a dedicated swap partition. Each swap area consists of a series of 4-KB **page slots**, which are used to hold swapped pages. Associated with each swap area is a **swap map**—an array of integer counters, each corresponding to a page slot in the swap area. If the value of a counter is 0, the corresponding page slot is available. Values greater than 0 indicate that the page slot is occupied by a swapped page. The value of the counter indicates the number of mappings to the swapped page. For example, a value of 3 indicates that the swapped page is mapped to three different processes (which can occur if the swapped page is storing a region of memory shared by three processes). The data structures for swapping on Linux systems are shown in Figure 10.10.



💡 En las áreas de swap se guardan páginas de **datos** (variables) y páginas de los **stacks**.

---

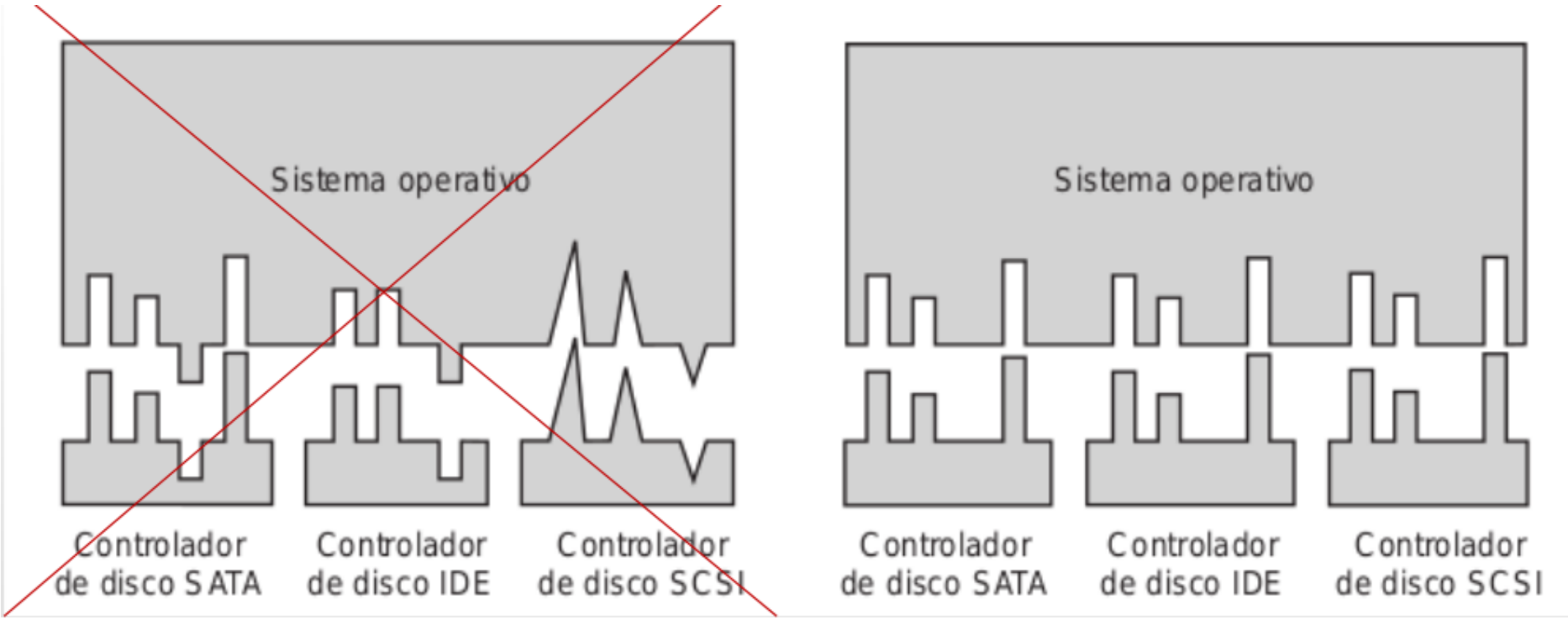
# Entrada/Salida

El sistema de entrada salida de una computadora se encarga de controlar los dispositivos conectados a ella y proporcionar una interfaz de utilización. Para ello tiene que lidiar con ciertos problemas, como por ejemplo las particularidades de cada dispositivo.

El sistema operativo tiene que tener en cuenta los siguientes aspectos:

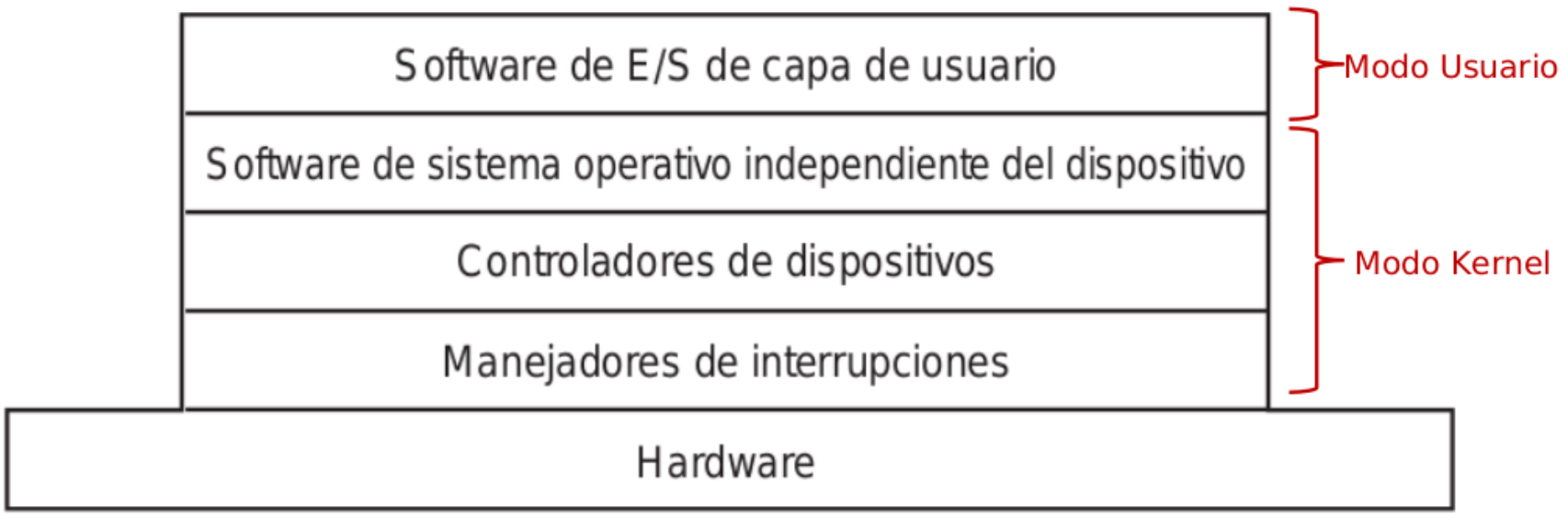
- **Unidad de transferencia:** si es un dispositivo que envía byte por byte (character-stream) o manda directamente un bloque de bytes como unidad (block).
- **Forma de acceso:** si el dispositivo es secuencial, es decir, recibe los datos en un orden fijo determinado por él, o es aleatorio, donde el dispositivo puede acceder a cualquier lugar de la memoria.
- **Tipo de acceso:** el dispositivo puede ser de acceso compartido, lo que quiere decir que varios procesos lo pueden usar de forma concurrente, o puede ser dedicado, donde un dispositivo lo usa a la vez.
- **Dirección de datos:** algunos dispositivos realizan entrada/salida, pero otros solo se manejan en una dirección.

La entrada salida busca ocultar los detalles y particularidades de cada dispositivo al CPU, para que así sean más fáciles de manejar. Los procesos deben ver a los dispositivos en términos de operaciones comunes (read, write, open, etc).




- **Eficiencia:** como los procesos de entrada/salida pueden resultar muy tediosos, la forma que tendremos de acelerarlos es con la multi-programación, para que mientras un proceso espera por la E/S, otro se pueda ejecutar.
- **Planificación:** organizar los requerimientos a los dispositivos para minimizar el tiempo de acceso.
- **Buffering:** almacenamiento de datos en memoria mientras son transferidos, para acelerar la velocidad de transferencia y problemas de tamaño y forma de los datos.
- **Caching:** mantener en una caché copias de los datos de reciente acceso.
- **Spooling:** administración de la cola de requerimientos a un dispositivo.
- **Manejo de errores:** administra errores (de lectura, escritura, dispositivo no disponible, etc), retorna código de error y genera los logs de informes.
- **Bloqueante o no bloqueante:** si la forma de entrada/salida es bloqueante, el proceso se suspende hasta que el requerimiento de e/s se complete. Si es no bloqueante, el resultado de la operación es recibido en cuanto sea posible.

# Diseño




En la **capa de usuario** se encuentran las librerías de funciones, que permiten el acceso a syscalls e implementan servicios que no dependen del Kernel, y los procesos de apoyo como por ejemplo el spooling daemon.

El **software de sistema operativo independiente del dispositivo** brinda los principales servicios de entrada salida como interfaz uniforme, manejo de errores, buffering, asignación de recursos y planificación, entre otros. Dentro de este software tenemos la información de estado de cada dispositivo (archivos abiertos, conexiones de red, etc) y otras estructuras que representan buffers, utilización de memoria, disco, etc.

Los **drivers (controladores)** brindan el código independiente de cada dispositivo. Este código traduce los requerimientos de los procesos en comandos puntuales para el dispositivo (y vice-versa), como por ejemplo escribir sobre los registros del controlador, acceder a la memoria mapeada y encolar requerimientos. Generalmente las interrupciones generadas por los dispositivos son atendidas por funciones que tiene el driver.

Los drivers proveen una interfaz entre el sistema operativo y el hardware, y los códigos están dentro del espacio de memoria del Kernel. Una ventaja de los drivers es que para agregar nuevo hardware al sistema solo basta indicar el driver correspondiente, sin necesidad de modificar el Kernel.

El **gestor de interrupciones** se encarga de atender a las interrupciones solicitadas por los dispositivos e/s, y deriva al driver correspondiente. También resguarda información necesaria, y funciona independientemente del driver en cuestión.

# Optimización

Las operaciones de entrada y salida son las que más afectan la performance del sistema, debido a varios factores como los context switches, la utilización de CPU para ejecutar drivers y código de I/O, y utilización excesiva de los buses para copiar datos.

Podemos darle una solución a esto:

- Reduciendo los context switches y copias de datos.
- Reduciendo la frecuencia de interrupciones mediante transferencias de grandes cantidades de datos, controladores más eficaces y polling.
- Utilizando DMA (direct memory access).

---

# File System

Los archivos nos permiten almacenar grandes cantidades de datos a largo plazo, y facilitan el acceso compartido al mismo conjunto de información.  Un archivo como tal es una entidad abstracta con nombre, que posee un espacio lógico continuo y direccionable. Les permite a los programas leer y escribir datos.

El sistema de manejo de archivos son pequeñas unidades de software que proveen los servicios necesarios para utilizar archivos, como por ejemplo crear, borrar, buscar, escribir, etc. lo que nos facilita como programadores el trabajo con ellos (nos evitamos programar la administración a bajo nivel). También facilita el acceso de las aplicaciones a ellos.

El sistema operativo debe:

- Garantizar la integridad del contenido de los archivos.
- Dar soporte de e/s a los dispositivos.
- Brindar interfaces de e/s para tratamiento de archivos.

# Aspectos de un Archivo

Los posibles tipos de un archivo son

- Archivos regulares: texto plano como source files, o archivos binarios (como los ejecutables).
- Directorios: archivos que ayudan a mantener la estructura en el FileSystem.

Atributos:

- Nombre
- Identificador
- Tipo
- Localización
- Tamaño
- Protección, seguridad y monitoreo: permisos, registros de acceso.

> 💡 El chequeo acerca de los permisos de un usuario sobre un archivo se hace en el **open**

# Directorios

Contienen información de los archivos y directorios que contiene. Son archivos. Los directorios pueden realizar operaciones como buscar, borrar, listar y renombrar archivos, entre otras.

La implementación de directorios ayuda a que la localización de un archivo sea más eficiente, nos permite utilizar el mismo nombre para dos archivos distintos (varían en el path) y podemos agrupar archivos por propiedades o funciones.

# Compartir Archivos

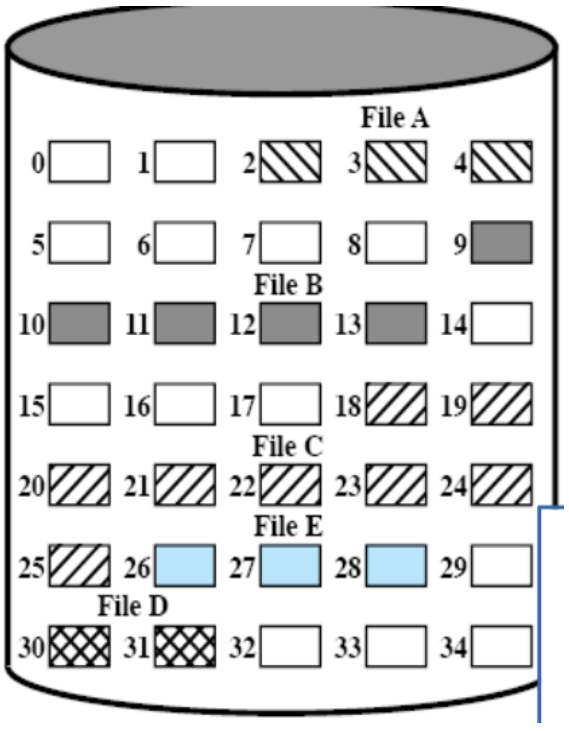
Como Linux es multiusuario debemos poder compartir archivos entre varios usuarios, y para ello necesitamos implementar un esquema de protección que permita manejar los accesos simultáneos y los derechos de acceso, donde el propietario pueda modificar los permisos (qué puede hacer un determinado usuario).

Los distintos derechos de acceso son:

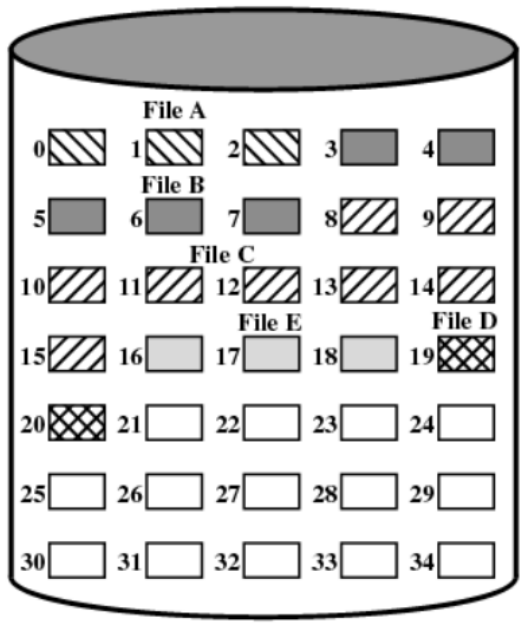
- Execution
- Reading
- Appending
- Updating
- Changing protection
- Deletion

# Asignación Continua

Los bloques utilizados se hallan de forma continua, para ello se debe conocer el tamaño del archivo durante su creación (pre-asignación). El FAT (file allocation table) es simple, ya que solo se requiere una entrada para el bloque de inicio y otra para la longitud. Tiene también la ventaja de que puede ser leído con una única operación.

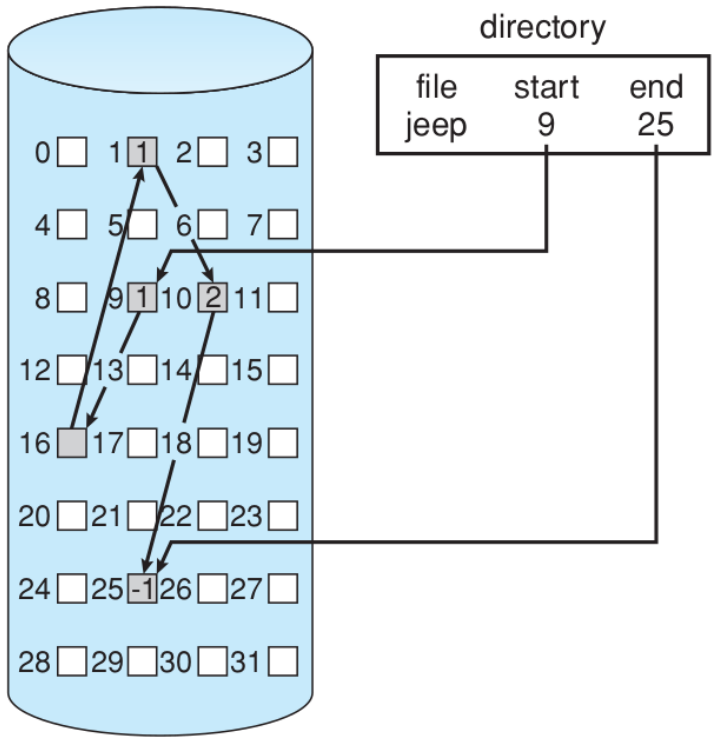



Las desventajas de esta técnica es que es más difícil encontrar bloques libres, y el tamaño de los archivos se puede ver incrementado. Además, puede generar fragmentación externa. Sin embargo puede ser solucionada con la compactación:



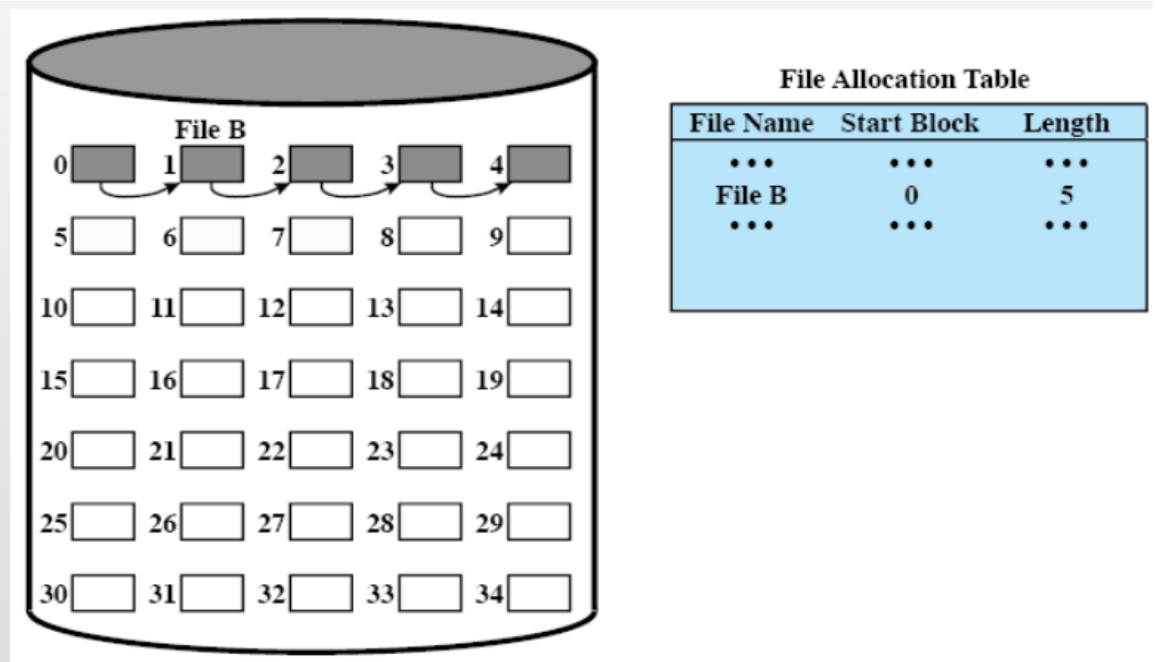

## Asignación Encadenada

La asignación encadenada soluciona todos los problemas de la continua, ya que los archivos son una **lista enlazada** de bloques, donde cada bloque puede estar en cualquier lugar del disco.



El directorio contiene un puntero hacia el primer y último bloque del archivo, y cada bloque un puntero hacia el próximo. Es particularmente útil para el acceso secuencial, y los archivos pueden crecer bajo demanda.

Otra cosa que puede pasar es que se consoliden los bloques en base al archivo, para garantizar la cercanía de los bloques pertenecientes al mismo.




# Asignación Indexada

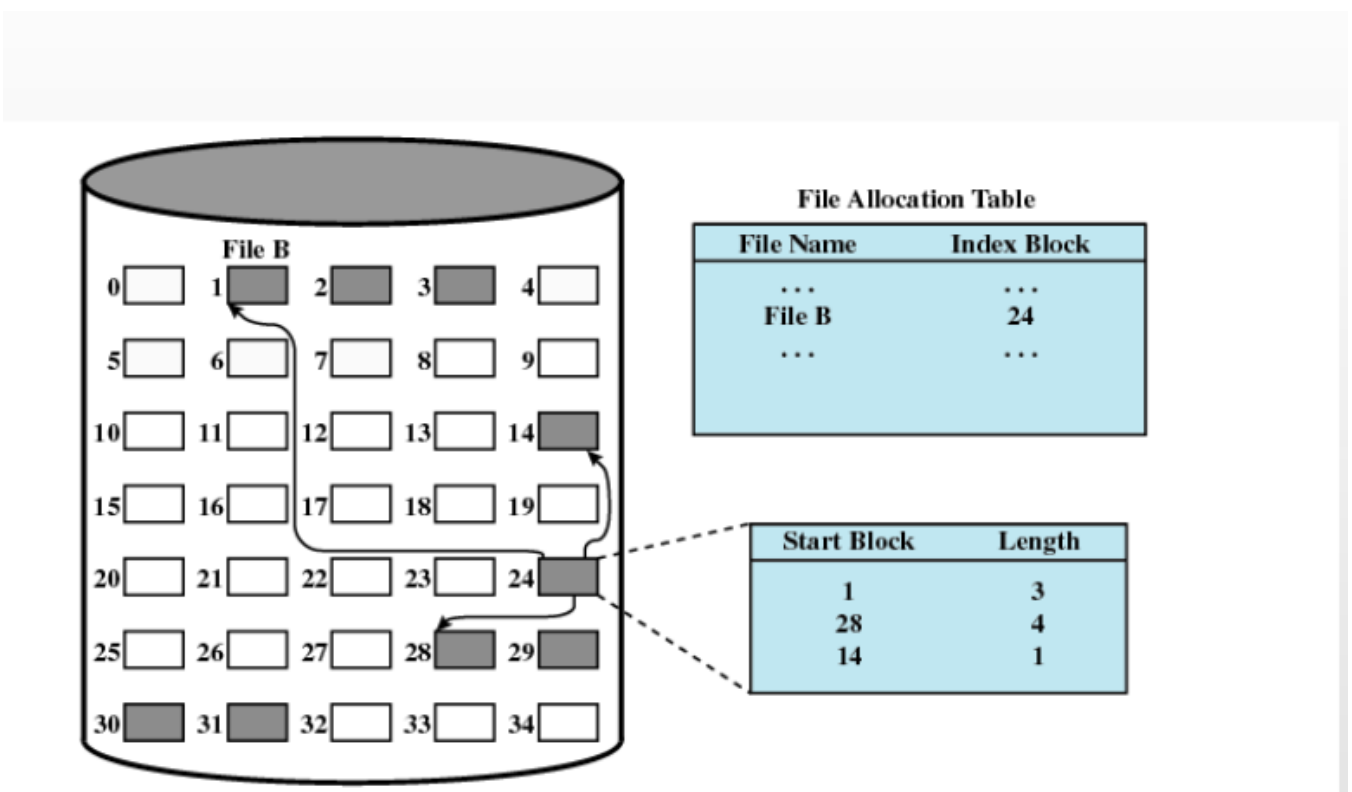
La asignación indexada soluciona el problema del acceso secuencial obligatorio que nos impone el acceso encadenado juntando todos los punteros en una misma ubicación llamada **bloque de índices.**

Cada archivo tiene su bloque de índices, que es un vector de direcciones a los bloques donde la i-ésima entrada al vector apunta al i-ésimo bloque.

Esta técnica elimina la fragmentación externa, hace eficiente el acceso random y simplifica el FAT, ya que la única entrada que tiene es la dirección del bloque de índices.

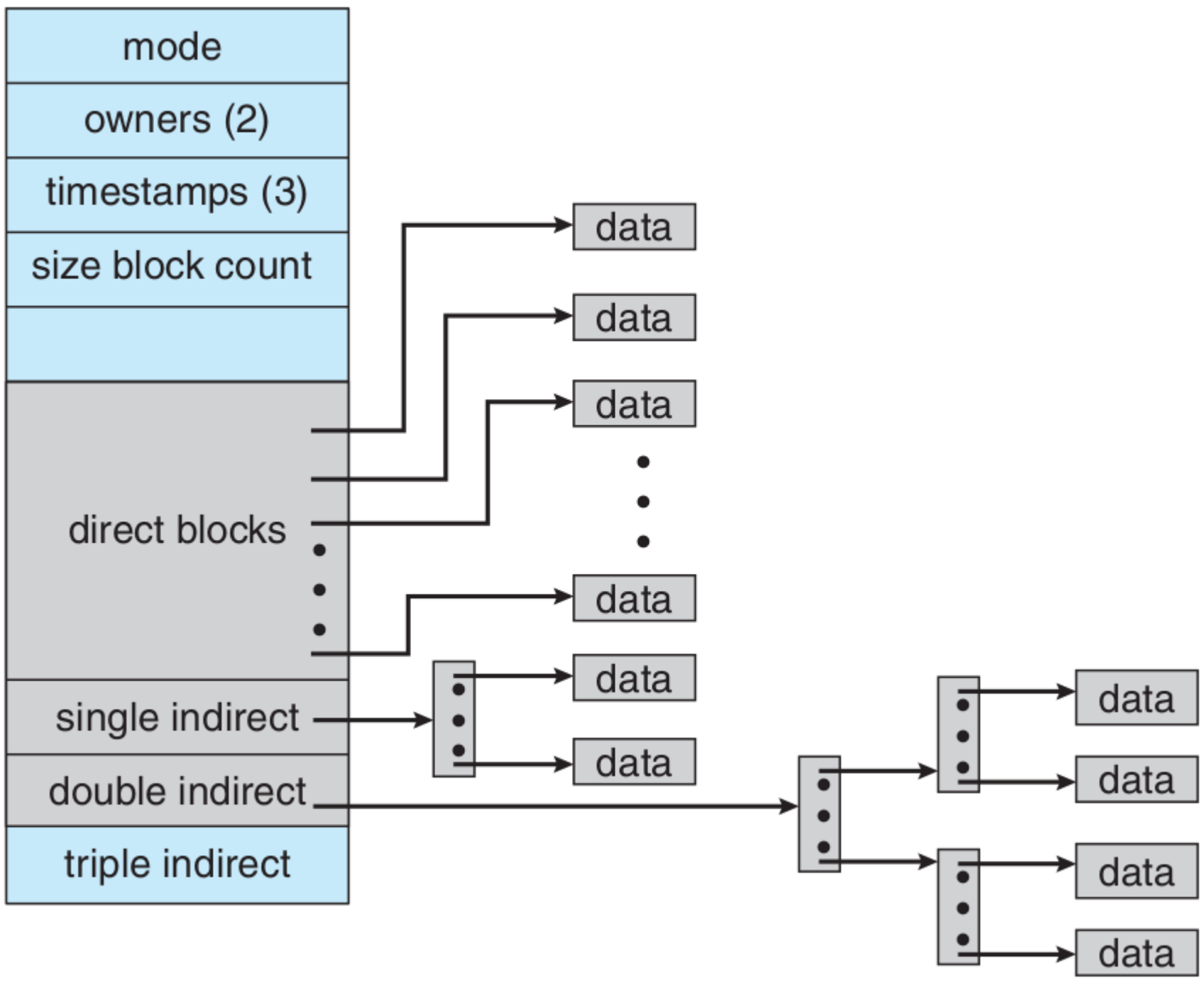
## Variante: Asignación por Secciones

Una modificación a la asignación Indexada, donde a cada entrada del bloque de índices se le agrega el campo longitud. El índice apunta al primer bloque de un conjunto almacenado de manera continua. Es un merge entre la indexada y la continua.




## Variante: Niveles de Indirección

Técnica que combina la encadenada con la indexada (i-nodos). Consiste en que el ante-penúltimo bloque es indirecto, es decir, contiene direcciones a otros bloques que contienen datos. El penúltimo direcciones a bloques con más direcciones, y el último sigue el mismo patrón.




## Gestión de Espacio Libre

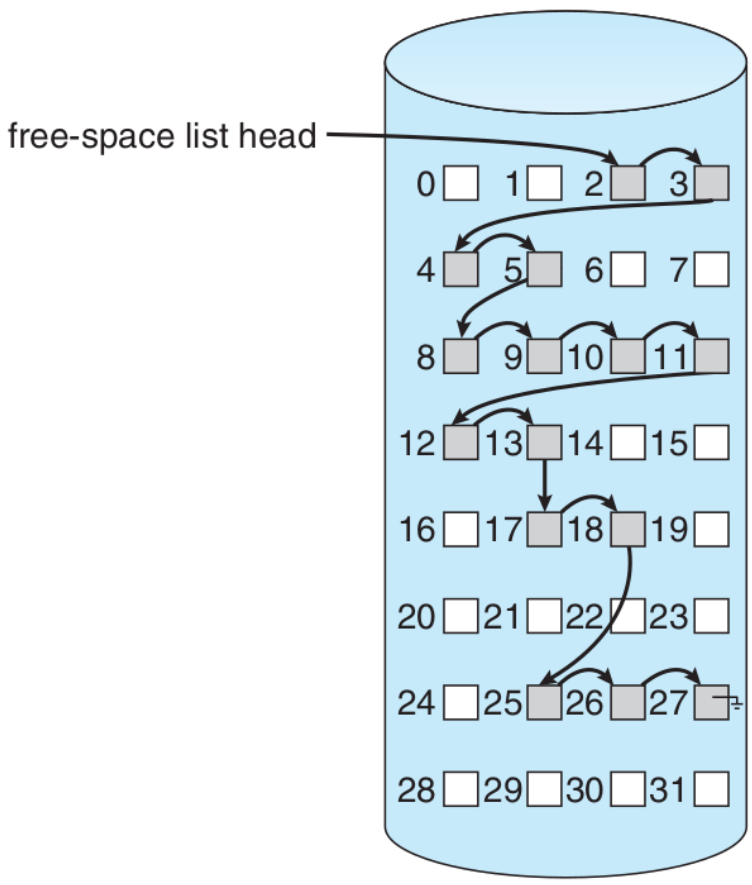
El sistema operativo debe llevar un control de qué bloques de disco están disponibles.

## Tablas de Bits

Tabla con un bit por cada bloque de disco, donde un 0 significa que está libre y un 1 que está en uso. Es fácil de encontrar un bloque o grupo de bloques libres, pero el tamaño puede ser demasiado grande con un disco no tan extenso.

## Lista Enlazada

El sistema operativo contiene una lista enlazada de todos los bloques libres, donde el primero contiene un puntero al segundo, y así sucesivamente. No es eficiente, ya que para recorrer la lista debemos acceder secuencialmente a cada uno de los bloques. Sin embargo, esto no es frecuente porque generalmente el sistema operativo necesita simplemente un bloque para almacenar datos, por lo que agarra el primero que encuentra.




### Agrupamiento

Variación de la lista enlazada. El primer bloque libre contiene direcciones de N bloques libres, donde las N - 1 primeras direcciones son bloques libres y la N-ésima dirección referencia a otro bloque con N direcciones de bloques libres.

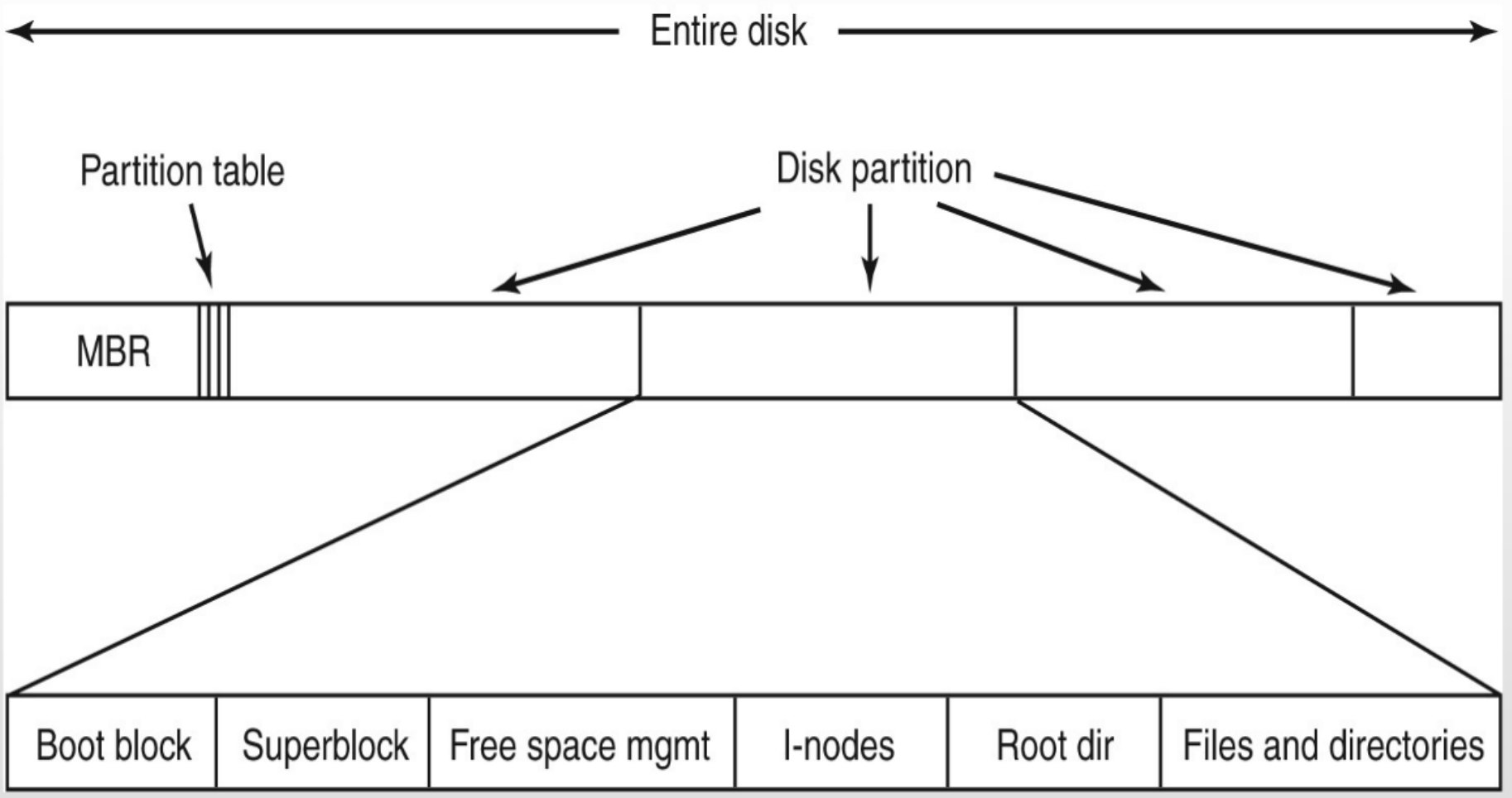
## Tipos de Archivos

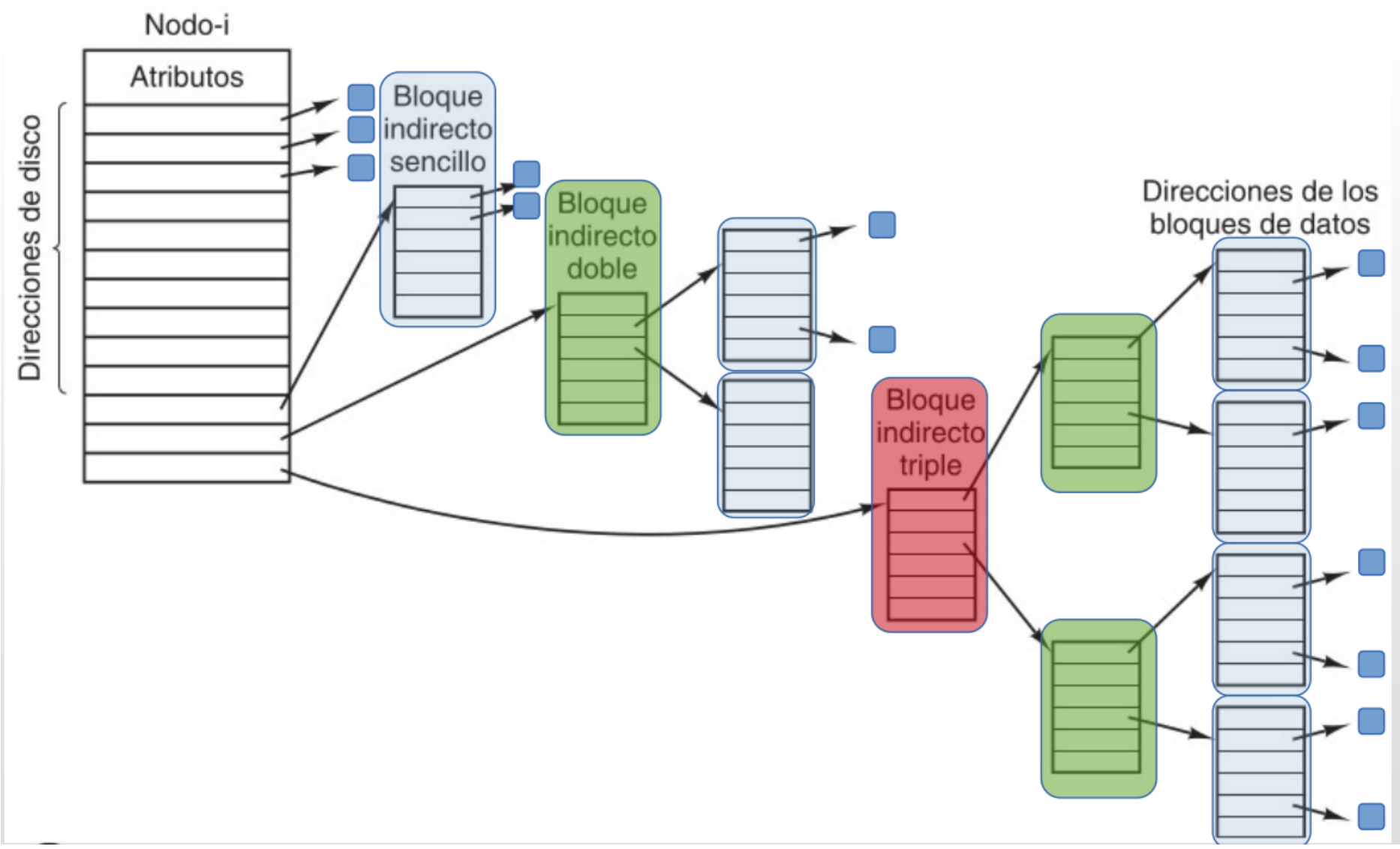
- Archivo común
- Directorio
- Archivos especiales (por ejemplo, los dispositivos)
- Named pipes (comunicación entre procesos)
- Links
- Links simólicos

## Estructura del Volumen

Cada disco físico puede ser dividido en uno o más volúmenes, donde cada volumen contiene un sistema de archivos. Los sistemas de archivos contienen:

- **Boot Block:** código para bootear el sistema operativo.
- **Superblock:** atributos sobre el FileSystem (bloques libres, etc).
- **I-Node table:** tabla donde están todos los i-nodos, donde un i-nodo es una estructura de control con información clave de un archivo.
- **Data blocks:** bloques de datos sobre los archivos en el volumen.



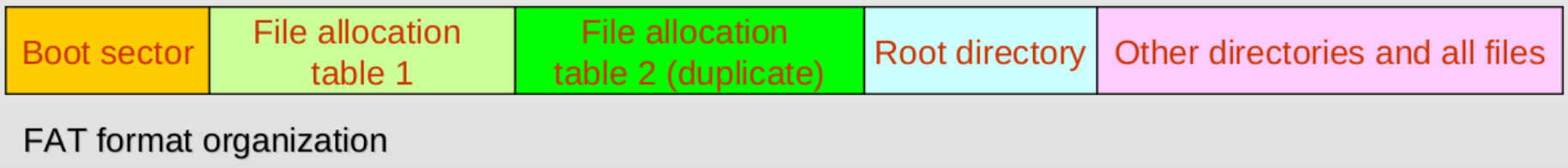

# File Systems de Windows

Windows soporta los siguientes FileSystems:

## FAT

FAT tiene la ventaja de que es compatible con varios sistemas multiboot, permiten upgrades desde versiones anteriores y soporta dispositivos como diskettes. Las distintas versiones de la gorda se diferencian por un número que indica la cantidad de bits que se usan para identificar los diferentes bloques o clusters (FAT 12, FAT 16 y FAT 32).

FAT como tal es un mapa de bloques del sistema de archivos que tiene tantas entradas como bloques. Este mapa o tabla se almacena en los primeros sectores de la partición.




FAT format organization

FAT utiliza un esquema de asignación encadenada, pero el puntero al próximo bloque lo tiene la gorda, no los bloques.

Cuanto más chico es el FAT (por ejemplo, FAT 12) menos son la cantidad de bloques que pueden haber en el disco, al igual que el tamaño del volumen. Si el FAT es más grande, (FAT 32) mayores son los bloques direccionables así como el tamaño máximo del volumen.

## NTFS

NTFS es el sistema nativo de Windows, el cual utiliza 64 bits para referenciar sectores, y permite tener volúmenes de hasta 16 Exabytes (16 billones de GB).

Utilizamos NTFS en vez de FAT ya que, a pesar de que FAT es simple y más rápida para algunas operaciones, NTFS soporta:

- Discos y archivos más grandes.
- Mejor performance en discos grandes.
- Nombres de archivos hasta 255 chars.
- Atributos de seguridad.
- Transaccional.

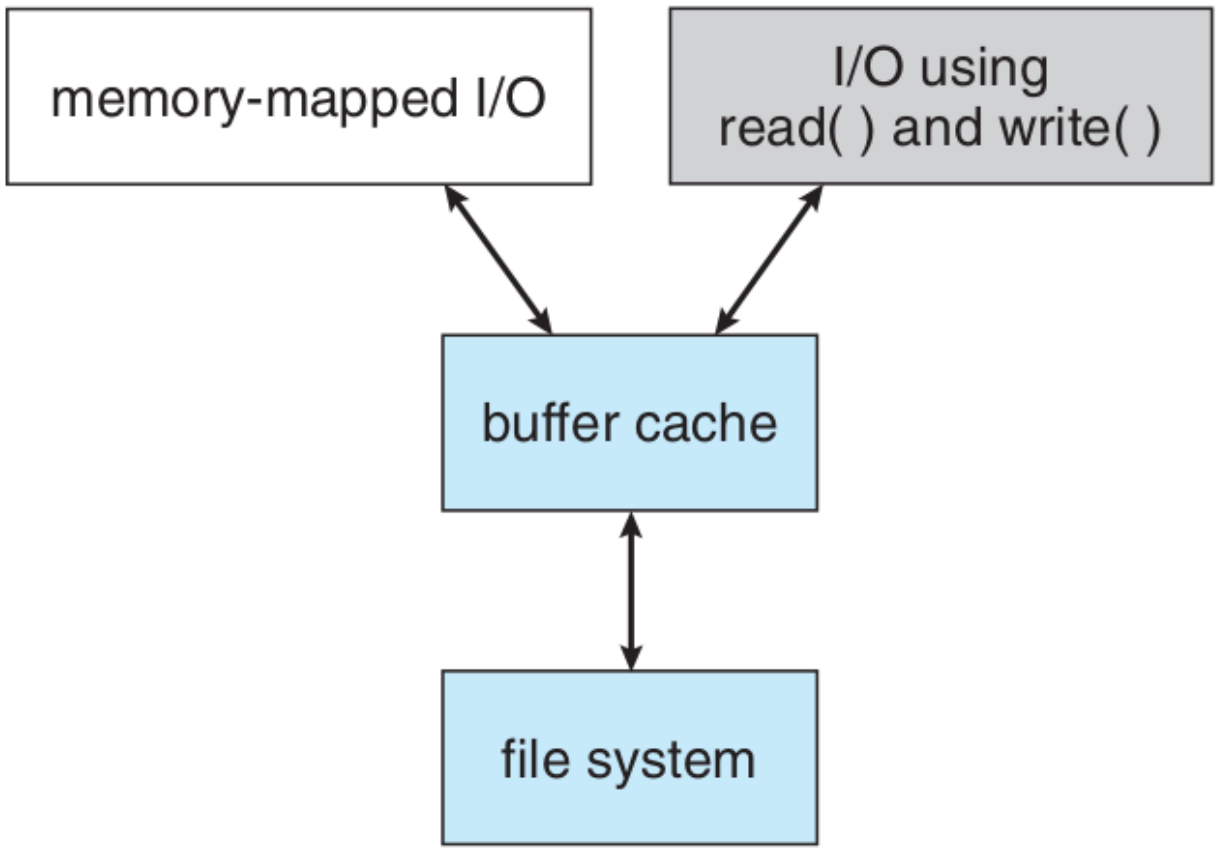
# Disk Caché

Son buffers en memoria principal que contienen bloques de disco, para minimizar el acceso a ellos. Si un proceso quiere acceder a este espacio de memoria, puede copiarlo a su espacio de direcciones (no permite el acceso compartido) o se trabaja directamente como memoria compartida.

La capacidad de almacenamiento de la caché no es extensa, por lo que se deben implementar algoritmos de reemplazo. Para ello se utiliza una lista donde el último es el usado más recientemente (se implementa con Least Recently Used). Cuando un bloque entra, se mete en el final de la lista. La ventaja de implementarlo con una lista es que no hace falta mover los bloques, sino que directamente cambiamos hacia donde apuntan los punteros.

# Buffer Caché

Es una estructura formada por muchos disk cachés.




El buffer caché tiene dos partes: el **header**, que contiene información relacionada a los datos (bloque, número de bloque, estado, relación con otros buffers, ...) y el **buffer** en sí, que es el lugar donde se almacenan los bloques traídos desde memoria.

Esta estructura es independiente del sistema de archivos, y es un servicio del sistema operativo.

## Header

El header identifica el número de dispositivo a donde pertenece el bloque almacenado, y el número de bloque. También contiene el estado y punteros hacia:

- 2 punteros para la hash queue.
- 2 punteros para la free list.
- 1 puntero al bloque de memoria.

> 💡 Podemos modificar el header sin modificar el archivo, y vice-versa.

Los estados de los buffers pueden ser los siguientes:

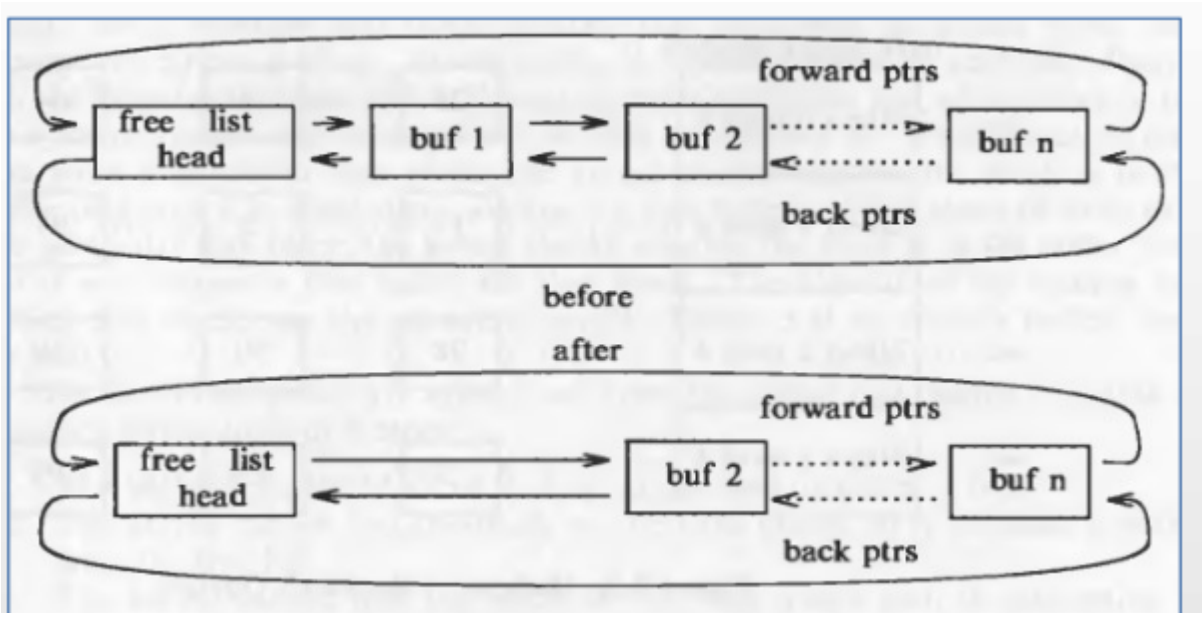
- Free.
- Busy.
- Escribiendo/leyendo.
- Delayed write: bloques que se encuentran modificados en el buffer pero no están actualizados en el disco.

> 💡 Un buffer puede estar busy y delayed write a la vez.

## Free List

Es una lista encadenada utilizada para encadenar los buffers disponibles para almacenar un bloque de disco. Esto no significa que esté vacío, sino que el proceso que lo estaba utilizando pudo haber terminado, o estar en delayed write sin que haya una instrucción posterior.

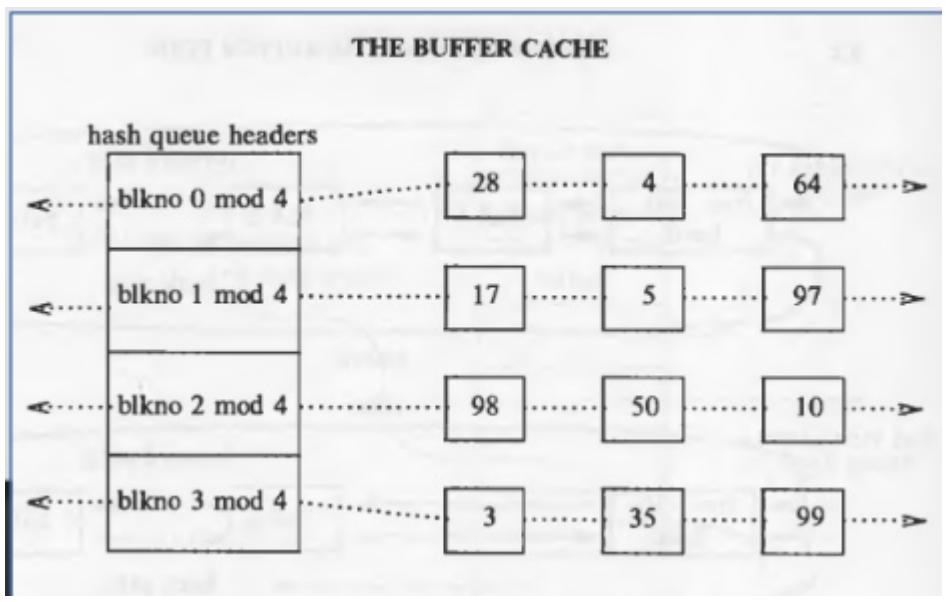
La free list se ordena con LRU.



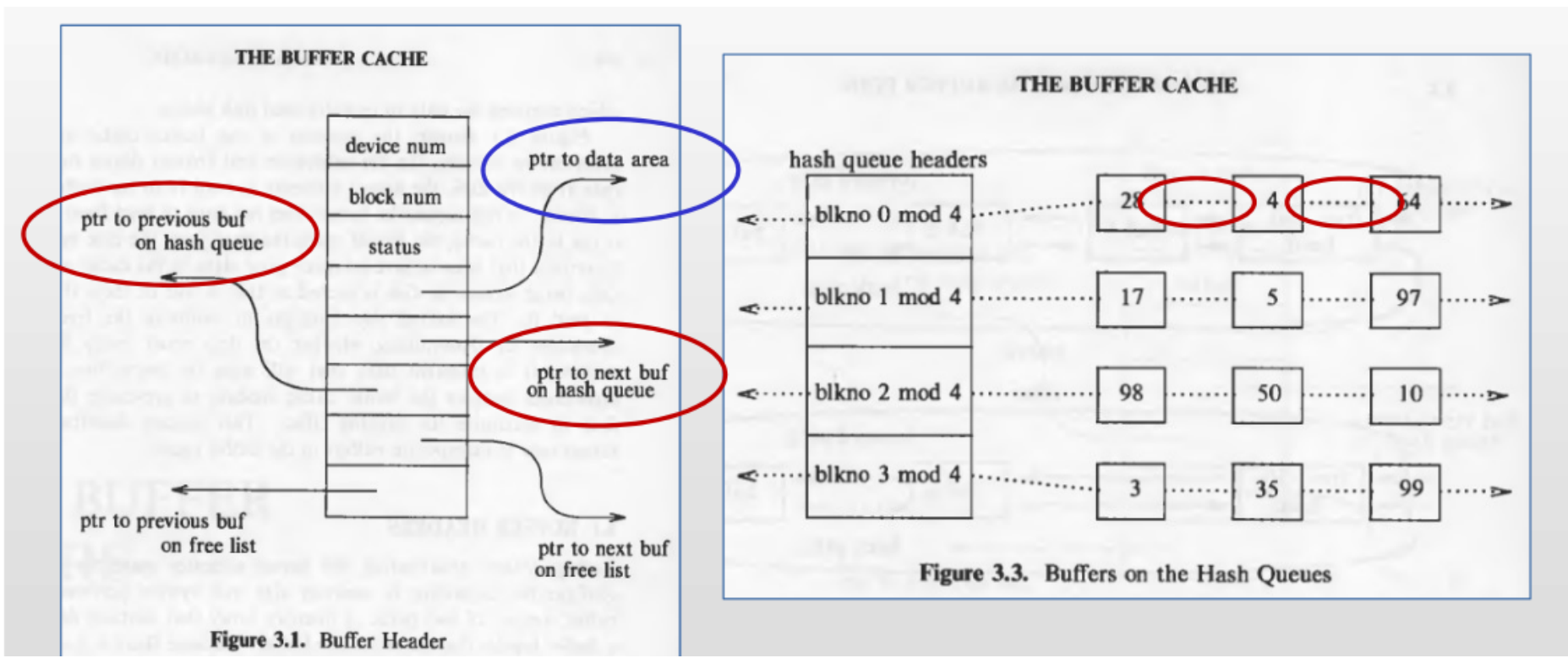

## Hash Queue

Son colas utilizadas para optimizar la búsqueda de un bloque en particular. En la cola se encuentran los headers de los buffers. Al número de bloque se le aplica una función de hashing para agrupar los buffers cuyo resultado haya dado igual, para hacer más eficaces las búsquedas.

Para que las colas no sean demasiado extensas, la función de hash provee de alta dispersión.




Los bloques son agrupados mediante los 2 punteros almacenados en el header, donde uno apunta hacia el anterior buffer y otro al siguiente de la hash queue.




# Funcionamiento del Buffer Caché

Si un proceso quiere acceder a un archivo, mediante su i-nodo busca la ubicación del bloque al que pertenece, y se fija si lo puede mandar al buffer caché. El buffer evalúa si puede satisfacer el requerimiento o si debe ejecutar una operación de

entrada/salida. Se pueden dar los siguientes escenarios:

- **Bloque en Hash Queue y Free List:** el bloque se encuentra almacenado en un buffer, y está disponible para utilizar. Se remueve el buffer buscado de la free list, pasa a estado busy y se reacomodan los punteros de la free list.
- **Bloque ausente en la Hash Queue:** el bloque no está en un buffer, por lo tanto se toma el primer buffer de la free list (depende del algoritmo de reemplazo), se lee de disco el bloque requerido y se incorpora el buffer en la hash queue que corresponda (solo se cambian los punteros, no se modifica la ubicación en memoria).
- **Bloque ausente en la Hash Queue y el bloque libre en DW:** si el bloque libre que necesitamos está en estado de delayed write, se asigna el bloque que siga. Cuando los que estaban en DW sean escritos a disco, estos bloques serán ubicados al principio de la Free List.
- **Bloque ausente en la Hash Queue y Free List vacía:** como no hay bloques libres, el proceso queda en estado de bloqueado hasta que haya algún buffer disponible. Cuando el proceso despierte, hay que verificar nuevamente que el bloque no esté tomado (osea, que no esté en la hash queue).
- **Bloque en la hash queue, pero busy:** si el bloque se encuentra en la hash queue pero está ocupado, el proceso que lo necesita debe esperar a que el buffer cambie a estado disponible.

---

https://prod-files-secure.s3.us-west-2.amazonaws.com/74a75ccf-02b9-4642-8519-1c39785d9574/9fa9a1f9-727f-4ac4-934a-6b48db99d339/preguntas-respondidas.pdf

https://prod-files-secure.s3.us-west-2.amazonaws.com/74a75ccf-02b9-4642-8519-1c39785d9574/96d71cc0-2f6e-4398-b17e-d748d711a58d/Segundo_Examen_de_Promocin.pdf

https://prod-files-secure.s3.us-west-2.amazonaws.com/74a75ccf-02b9-4642-8519-1c39785d9574/575d9174-fa2f-4a4b-ad43-768a910e44fe/Tercer_Examen_de_Promocin.pdf