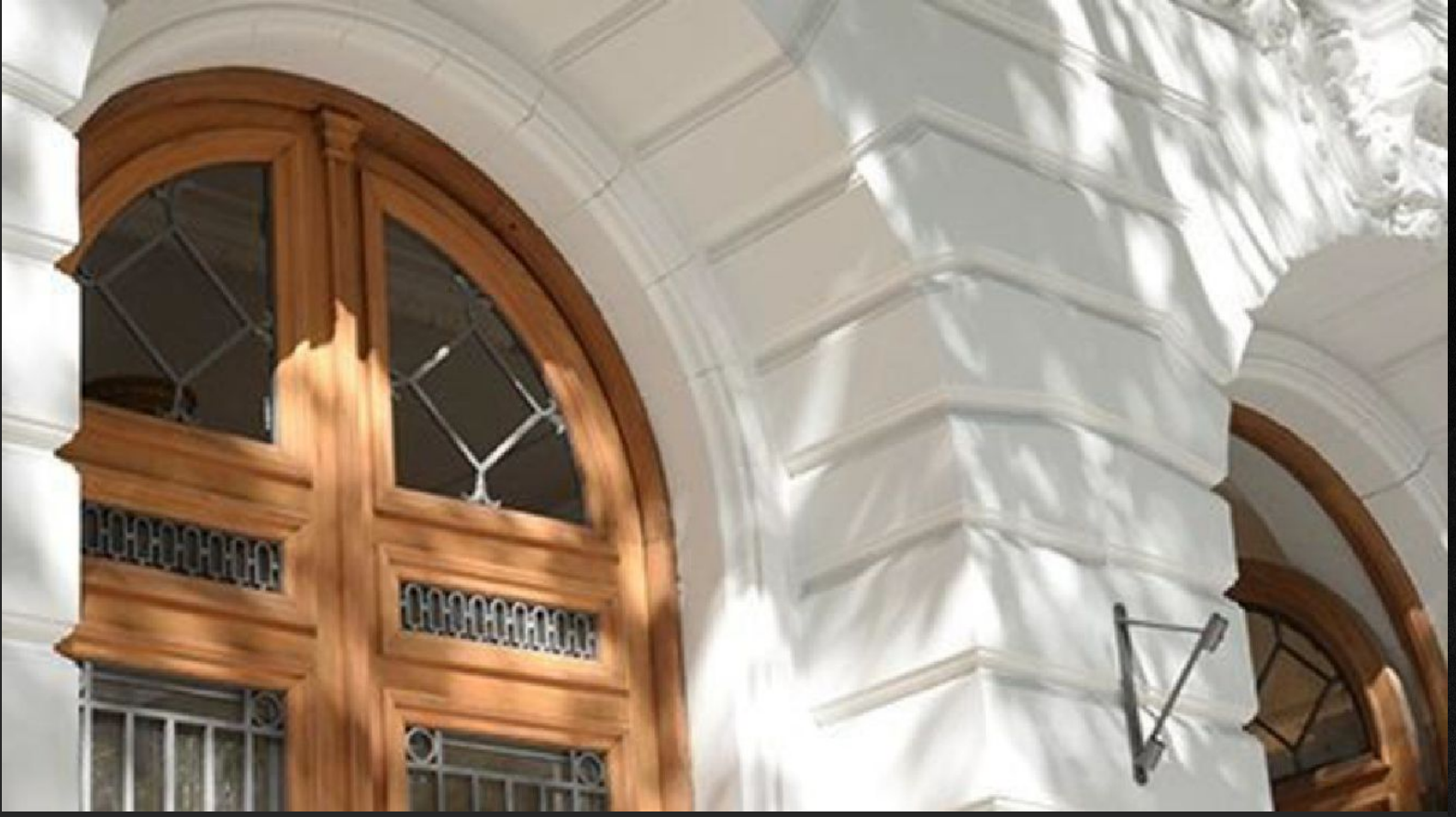


UTN.BA FACULTAD REGIONAL BUENOS AIRES

Arquitectura de Sistemas Operativos

Clase de repaso – Clases 6 a 9

Tecnicatura Universitaria en Programación

Agenda Repaso 2do Parcial

Clase 6: Planificación de Procesos

Clase 7: Gestión de Memoria

Clase 8: Memoria Virtual

Clase 9: Gestión de Entrada/Salida

¿Consultas?

Clase 6

- Tipos de planificación
- Criterios y políticas
- Algoritmos

Planificación de procesos

Objetivo – Tipos de Planificación de procesos

- El objetivo de la planificación es **asignar los procesos** al procesador para ser **ejecutados**.
- La planificación **afecta** el **rendimiento** general del sistema porque decide qué proceso ejecutará y qué procesos deberán esperar.

Planificación: Largo Plazo

Largo Plazo

Objetivo	Decisiones que toma	
controlar el grado de multiprogramación	¿Cuándo admitir un nuevo proceso al sistema?	¿Qué proceso admitir?
Procesos nuevos a cola de listo	Fin de un proceso, basado en métricas	FIFO, Prioridad, recursos, tiempo estimado
Procesos nuevos a listos/suspendidos		

Largo Plazo

🔴 **Largo plazo** → *Admisión de procesos nuevos*

- Decide **cuándo un proceso entra al sistema**.
- Controla **cuántos procesos hay en memoria** al mismo tiempo (grado de multiprogramación).
- Si hay recursos, pasa a la cola de listos. Si no, queda fuera o suspendido.

Planificación: Mediano Plazo

Medio Plazo

Objetivo

controlar el grado de multiprogramación

Es parte de la función de **swap**

Procesos suspendidos a bloqueado o listo

Decisiones que toma

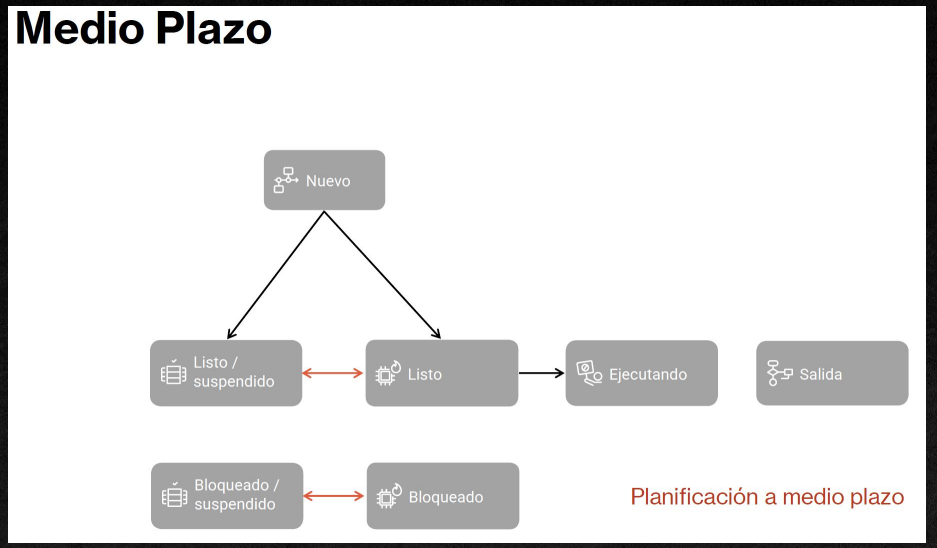
¿Cuándo traer un proceso a memoria principal?

Eventos

¿Qué proceso admitir?

Depende de la memoria disponible y requerida

Función **SWAP** – **Eventos** – Decisión **memoria** disponible y requerida



🟡 **Mediano plazo** → *Swap (suspensión/reanudación)*

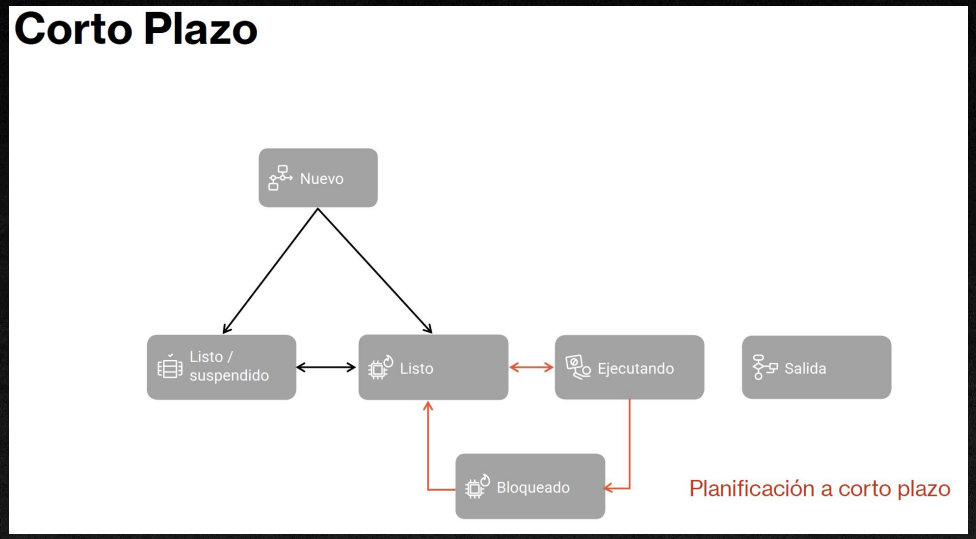
- Se activa cuando hay **falta o liberación de memoria**.
- Si hay mucha carga, **suspende procesos** y los lleva a disco.
- Cuando hay espacio, **los reactiva**. Esto mantiene el sistema ágil.

Planificación: Corto Plazo

Corto Plazo (dispatcher)

Objetivo	Decisiones que toma	
Optimizar el uso del sistema	¿Cuándo? Eventos: syscall, interrupciones, etc.	¿Qué proceso debe utilizar el procesador? Basado en algoritmos
Intercambia procesos		

Optimiza uso sistema – **intercambia** procesos – Decisión **CPU proceso con algoritmos**

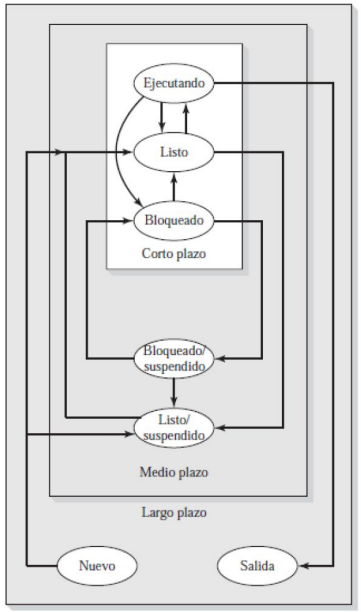


🟢 **Corto plazo** → *Dispatcher (planificador real)*

- Es el que actúa más seguido.
- Decide **qué proceso se ejecuta ahora mismo en el CPU**.
- Se activa en interrupciones, bloqueos o cuando un proceso termina su turno.
- Usa algoritmos como FCFS (First Come, First Served), Round Robin, etc.

Planificación:

Niveles de planificación



Modelo de 7 estados incluyendo el anidamiento de las funciones de planificación

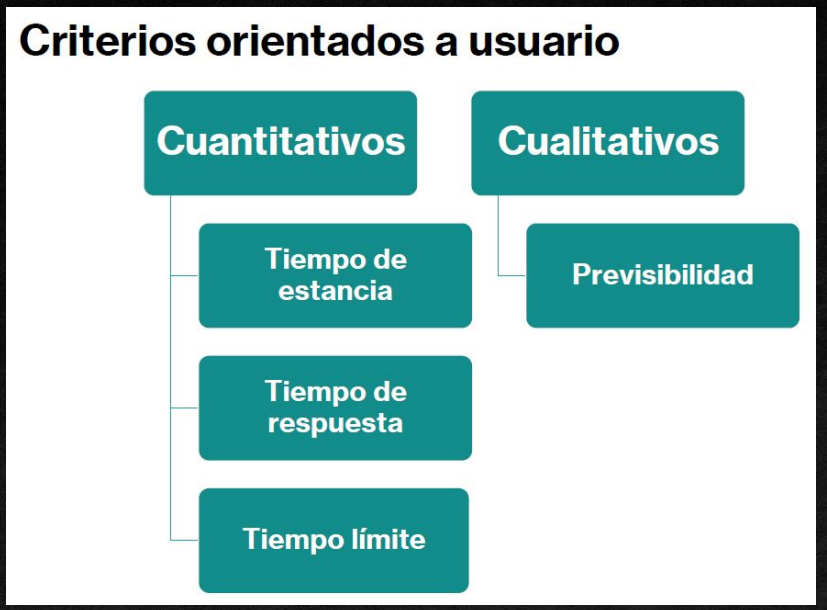
Tipos de planificación

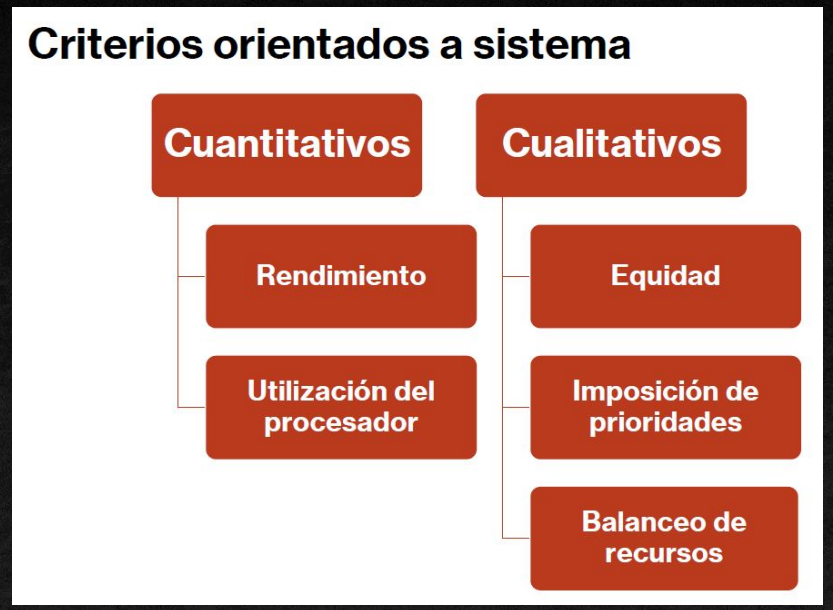
Corto Plazo	• Dispatcher
Medio Plazo	• Swap
Largo Plazo	• Nuevos procesos

📌 En resumen:

- El **de corto plazo** administra el uso del procesador.
- El **de mediano plazo** gestiona la carga de memoria.
- El **planificador de largo plazo** habilita el ingreso.

Criterios y políticas de planificación:





- Tiempo de respuesta limite.

- Rendimiento , equidad , prioridades.

- Políticas de exclusión, mas sobrecarga, mejora el rendimiento.
- Politicas de prioridad : Starvation – inanición. No se ejecuta.

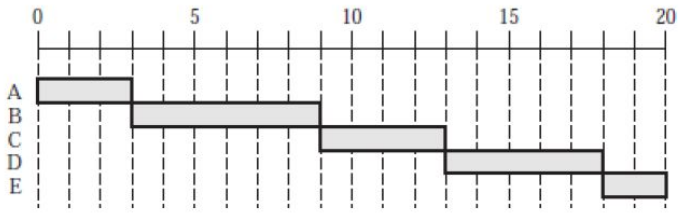
Algoritmos basados en la ejecución:

- **FCFS** – *First Come, First Served*
- **SPN** – *Shortest Process Next*
- **SRT** – *Shortest Remaining Time*
- **Round Robin** – *Round Robin Scheduling*
- **Feedback** – *Multilevel Feedback Queue*

FCFS (First Come – First Served) Atiende en el orden que llega

FCFS. Primero en llegar, primero en servirse

Proceso	Tiempo de llegada	Tiempo de servicio
A	0	3
B	2	6
C	4	4
D	6	5
E	8	2



El proceso E tiene el menor tiempo de servicio, pero tarda en ejecutarse porque debe esperar que finalicen procesos largos como el B o el D

FCFS. Primero en llegar, primero en servirse

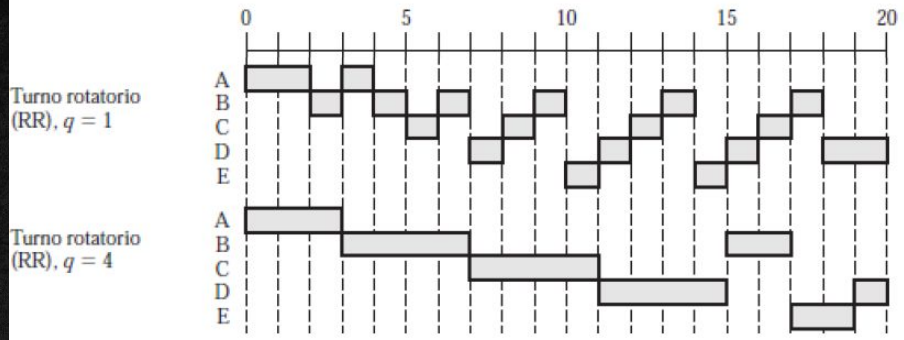
Modo de selección	• Sin expulsión
Tiempo de respuesta	• Puede ser alto especialmente si hay mucha diferencia entre los tiempos de ejecución de los procesos
Efecto sobre los procesos	• Penaliza procesos cortos; penaliza procesos con mucha E/S
Inanición	• No genera inanición
Sobrecarga	• Mínima

- El **primero** en llegar se **ejecuta**. Hay **espera.**

Round Robin. Cada proceso recibe un quantum de tiempo fijo.

Round Robin (turno rotatorio)

Proceso	Tiempo de llegada	Tiempo de servicio
A	0	3
B	2	6
C	4	4
D	6	5
E	8	2



- Pensado para reducir el castigo a los procesos cortos.
- Se define un quantum de tiempo (q).
- Si q es pequeño el proceso se mueve rápido, pero se genera sobrecarga de procesamiento.
- Si q es mayor que el proceso mas largo en ejecución, se vuelve a FCFS.

Round Robin (turno rotatorio)

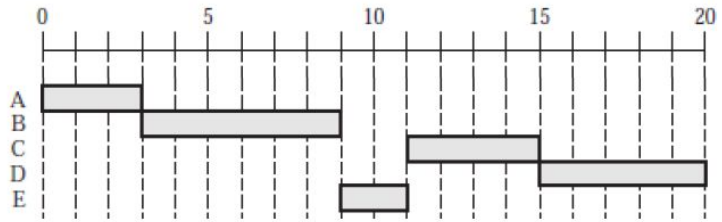
Modo de selección	• Con expulsión (por rodajas de tiempo)
Tiempo de respuesta	• Buen tiempo de respuesta para procesos cortos
Efecto sobre los procesos	• Tratamiento justo
Inanición	• No genera
Sobrecarga	• Mínima

- Le **toca** al siguiente proceso por **q tiempo**.

SPN (Shortest Process Next) Ejecuta el proceso con menor tiempo de ejecución estimado.

SPN. Primero el proceso mas corto

Proceso	Tiempo de llegada	Tiempo de servicio
A	0	3
B	2	6
C	4	4
D	6	5
E	8	2



Un proceso corto se situará a la cabeza de la cola, por delante de los procesos mas largos.

El tiempo de servicio debe ser informado por el programador, o se determina en base a estadísticas

- Se **evalua** el **tiempo** de servicio **primero.**

SPN. Primero el proceso mas corto

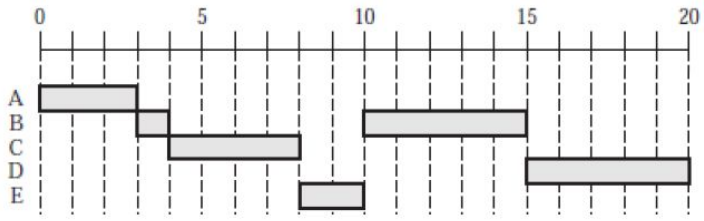
Modo de selección	• Sin expulsión
Tiempo de respuesta	• Buen tiempo de respuesta para procesos cortos
Efecto sobre los procesos	• Penaliza procesos largos
Inanición	• Posible
Sobrecarga	• Alta

- **Sin expulsion – Demoras procesos largos.**

SRT – Shortest Remaining Time – Primero el mas corto - Versión con expulsión de SPN.

SRT. Menor tiempo restante

Proceso	Tiempo de llegada	Tiempo de servicio
A	0	3
B	2	6
C	4	4
D	6	5
E	8	2



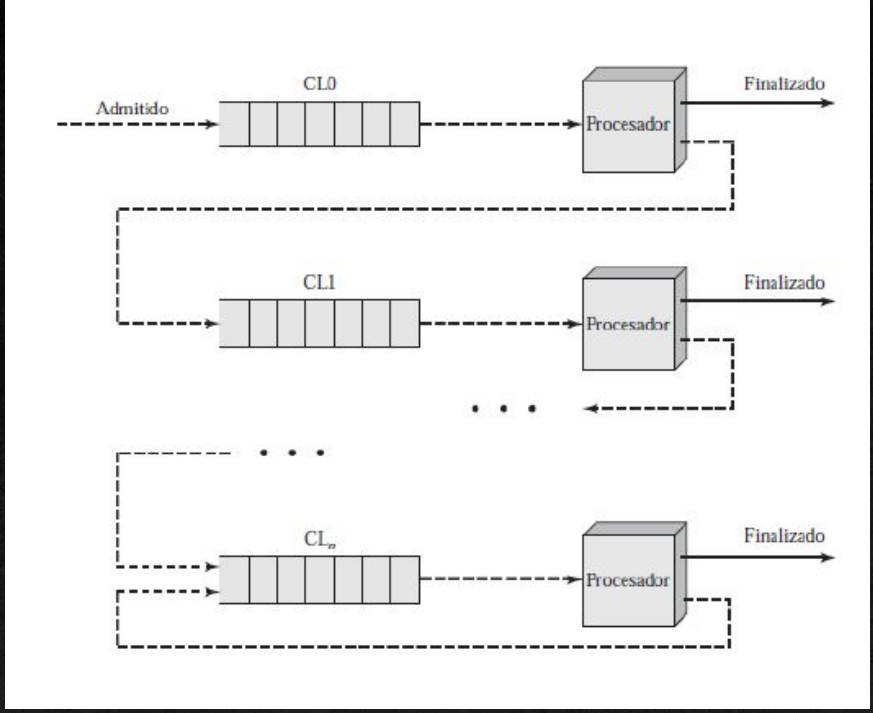
- SRT es una versión expulsiva de SPN.
- Si llega a la cola de listos un proceso con menor tiempo restante que el que se está ejecutando, el planificador podría expulsar al proceso actual.
- El tiempo de servicio debe ser informado por el programador, o se determina en base a estadísticas

SRT. Menor tiempo restante

Modo de selección	• Expulsiva (a la llegada)
Tiempo de respuesta	• Buen tiempo de respuesta.
Efecto sobre los procesos	• Penaliza procesos largos
Inanición	• Posible
Sobrecarga	• Alta

- Siempre se ejecuta el proceso con **el menor tiempo de servicio**.
- Si **llega un nuevo proceso** con menor tiempo restante que el actual, **interrumpe al que se está ejecutando.**
- Requiere conocer el **tiempo estimado de servicio** de cada proceso.

Retroalimentación - Feedback. Usa colas de prioridades dinámicas.



- Se usa **cuando no se conoce el tiempo de servicio** de los procesos.
- Utiliza **expulsión** y un esquema de **prioridades dinámicas**.
- Los procesos comienzan en la cola de mayor prioridad (CL0).
- Cada vez que un proceso es **expulsado**, se mueve a una **cola de menor prioridad**.
- Así, los procesos largos descienden gradualmente, favoreciendo a los procesos nuevos y cortos.
- Dentro de cada cola se usa FCFS (el primero que llega se ejecuta) , excepto en la de menor prioridad donde los procesos permanecen hasta finalizar.

Retroalimentacion - Feedback. Usa colas de prioridades dinámicas.

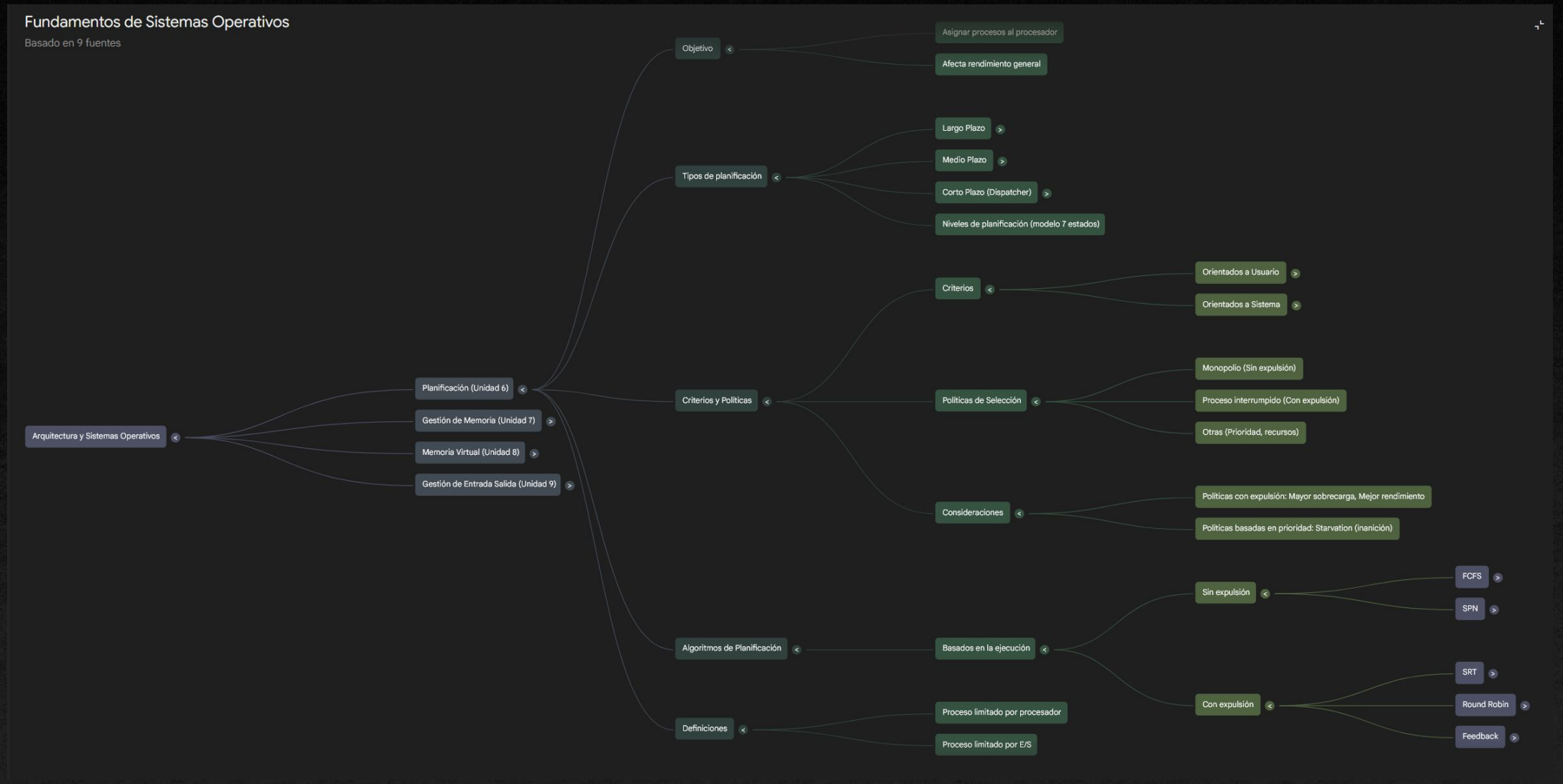
Retroalimentación (feedback)

Modo de selección	• Expulsiva (por rodaja de tiempo)
Tiempo de respuesta	• No especificado.
Efecto sobre los procesos	• Puede favorecer procesos con mucha E/S
Inanición	• Posible
Sobrecarga	• Alta

Resumen – Planificación de procesos

- Tipos:
  - Largo (Admisión)
  - Medio (Swap)
  - Corto (Dispatcher)
- Criterios:
  - Usuario (Tiempo respuesta Límite)
  - Sistema (Rendimiento, Equidad).
- Algoritmos:
  - FCFS (First Come – First Served) Atiende en el orden que llega.
  - SPN (Shortest Process Next) Ejecuta el proceso con menor tiempo de ejecución estimado.
  - SRT – Shortest Remaining Time – Versión con expulsión de SPN.
  - Round Robin. Cada proceso recibe un quantum de tiempo fijo.
  - Feedback. Usa colas de prioridades dinámicas.
  - Evaluación de sobrecarga e inanición.

Resumen – Planificación de procesos – Mapa Conceptual



Clase 7

- Requisitos
- Particionamiento
- Paginación y segmentación

Gestión de Memoria

Definiciones – Gestión de Memoria

Sistemas Monoprogramados	Sistemas Multiprogramados
La memoria se divide entre el SO y el programa en ejecución	La memoria se divide entre el SO y **los programas** en ejecución

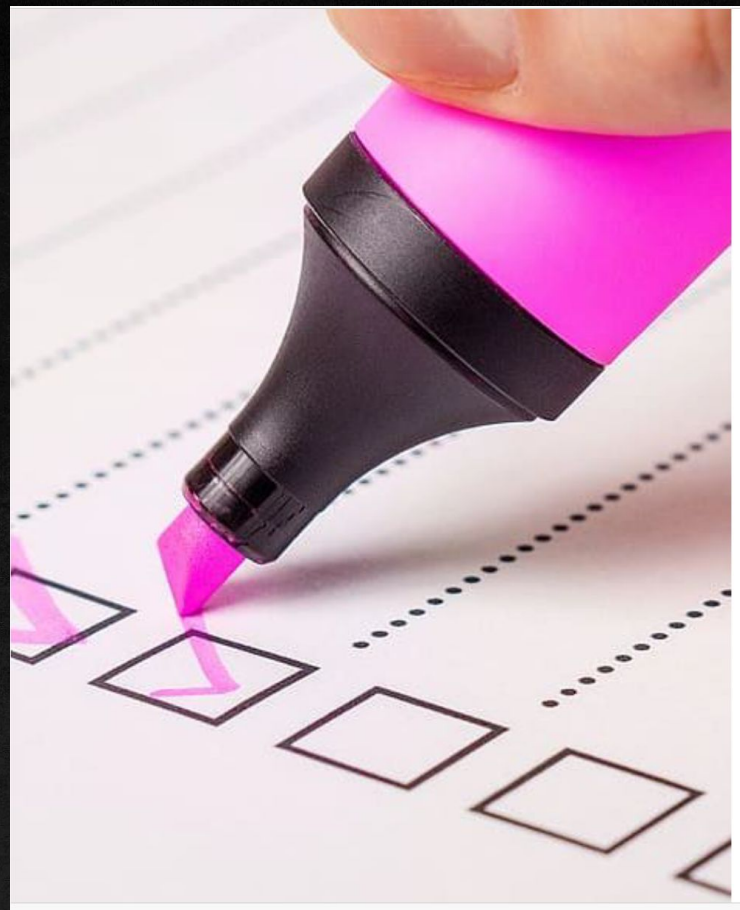
La gestión de la memoria se encarga de subdividir el área de memoria asignada a los programas de usuario

📌 **Resumen:**

Objetivo de la gestión de memoria

- Permitir que haya **suficientes procesos en memoria** para que el procesador **no esté ocioso**, maximizando el uso del sistema y evitando esperas innecesarias por E/S.

Gestión de Memoria – 5 Requisitos



🧠 Que se obtiene? 🤔

- **Reubicación** → Flexibilidad
- **Protección** → Seguridad
- **Compartición** → Cooperación
- **Organización lógica** → Modularidad
- **Organización física** → Eficiencia

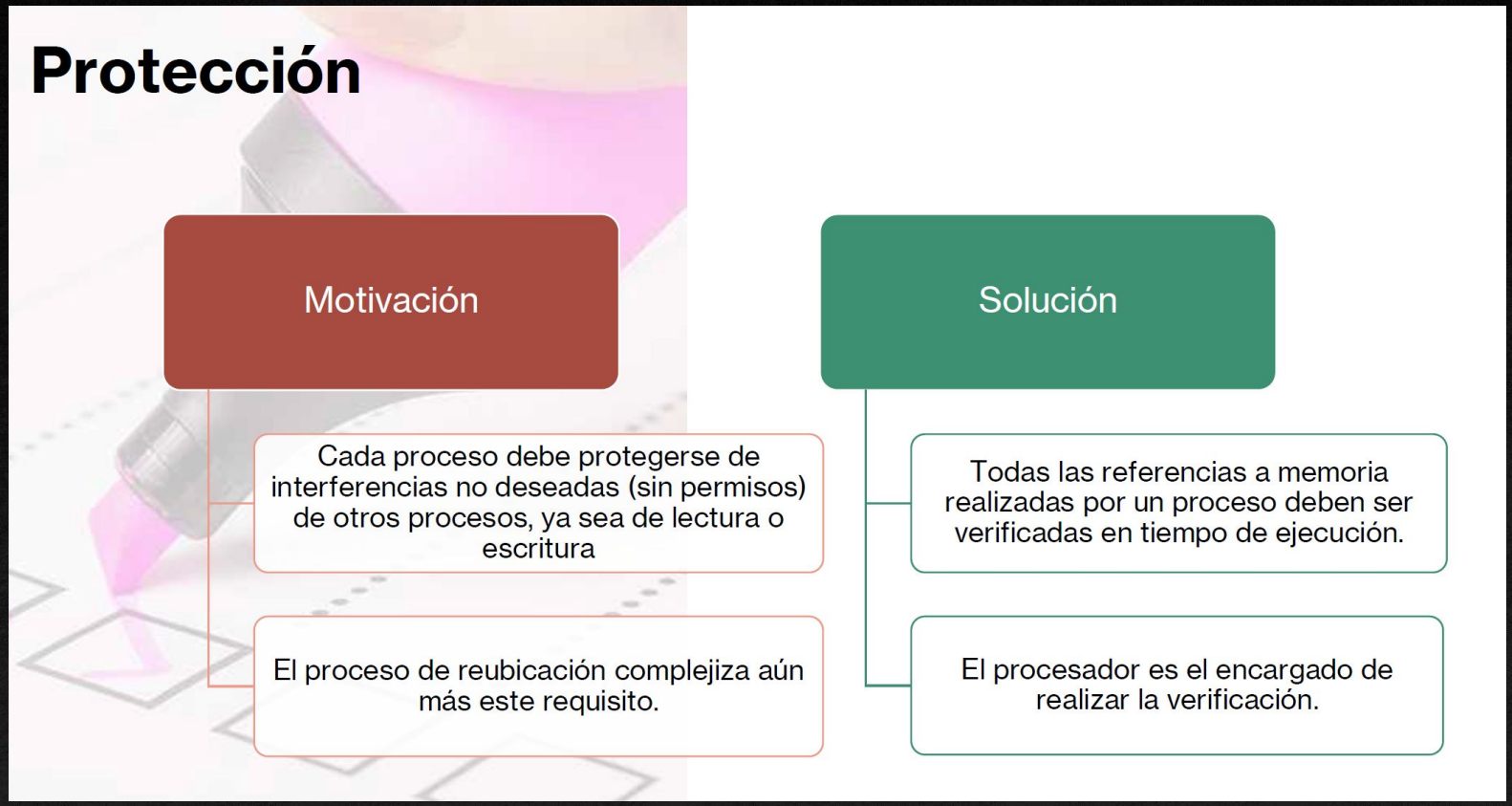
Gestión de Memoria – Reubicación - Flexibilidad

- La **reubicación** permite **ubicar procesos en memoria** aunque no se sepa su posición de antemano.
  Es clave para el **intercambio (swap)** y para traducir correctamente direcciones, usando la info del **PCB**.

- **PCB** : **Process Control Block**

(*Bloque de Control de Proceso*).

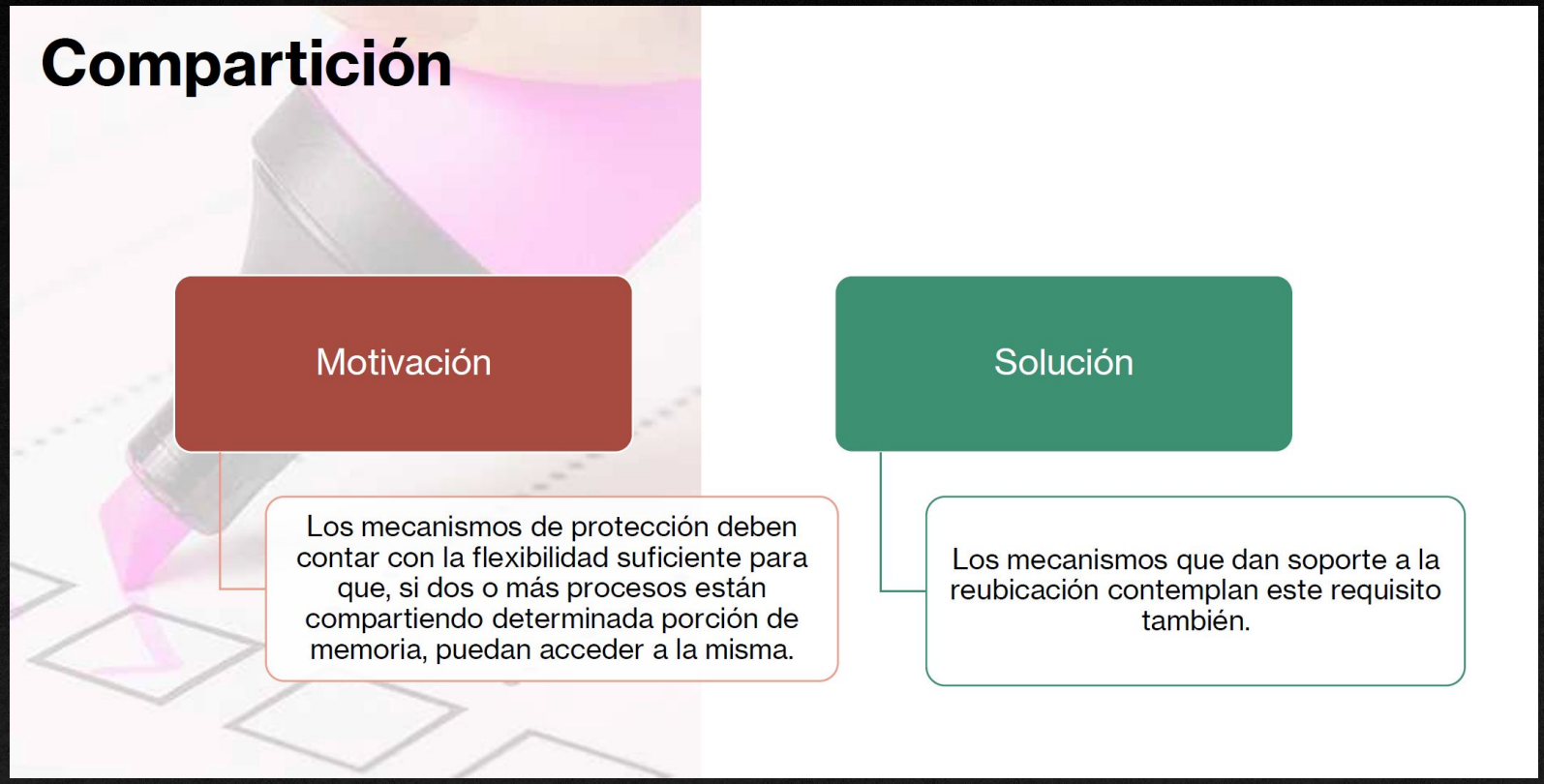
Gestión de Memoria – Protección - Seguridad



- Cada proceso debe estar protegido contra accesos no autorizados de otros procesos.

👉 El procesador verifica en tiempo real que **todas las referencias a memoria** sean válidas y seguras.

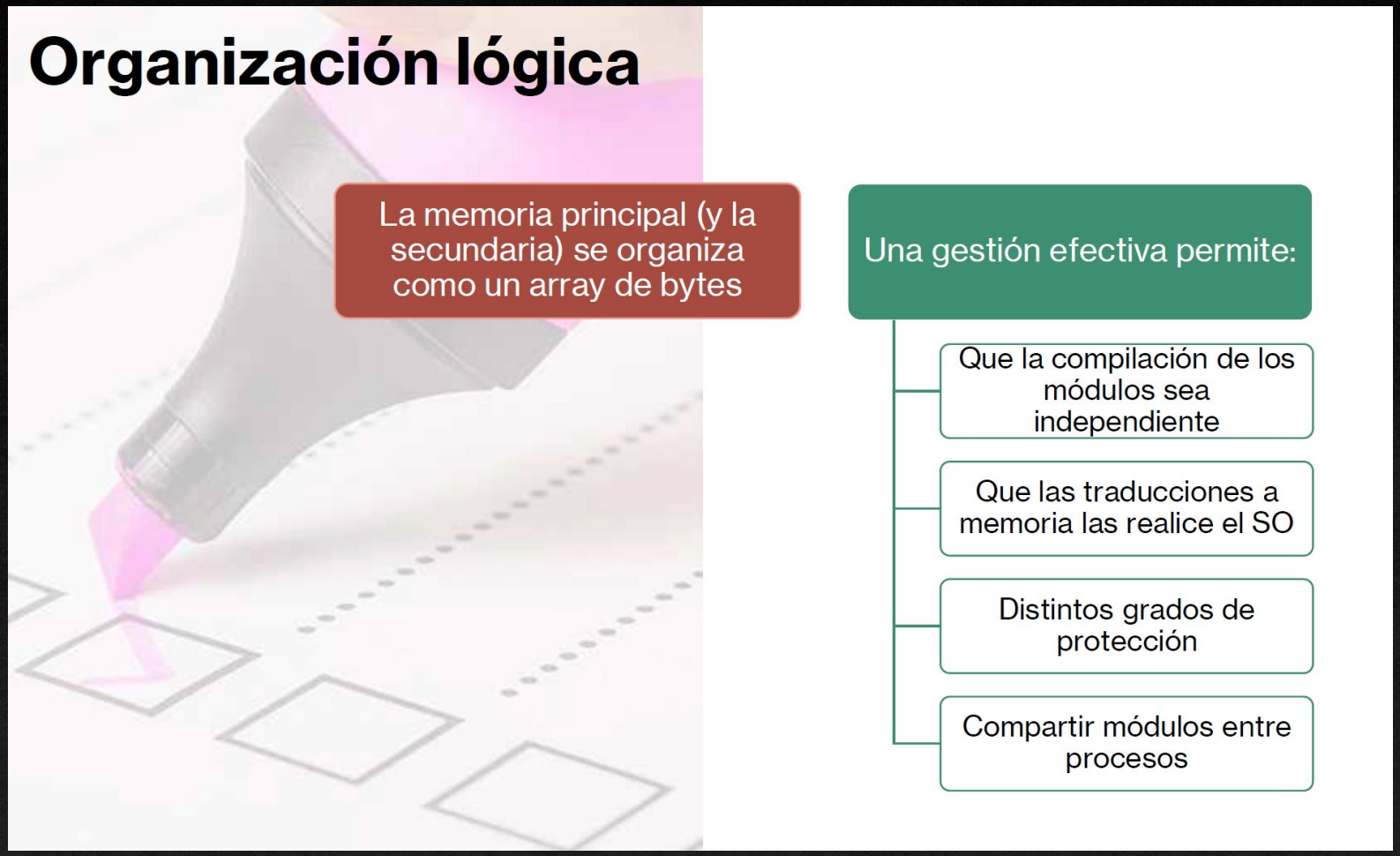
Gestión de Memoria – Compartición - Cooperación



- La memoria debe permitir que **varios procesos compartan áreas comunes** de forma segura y controlada.

👉 Se apoya en los **mismos mecanismos** que permiten la reubicación y protección.

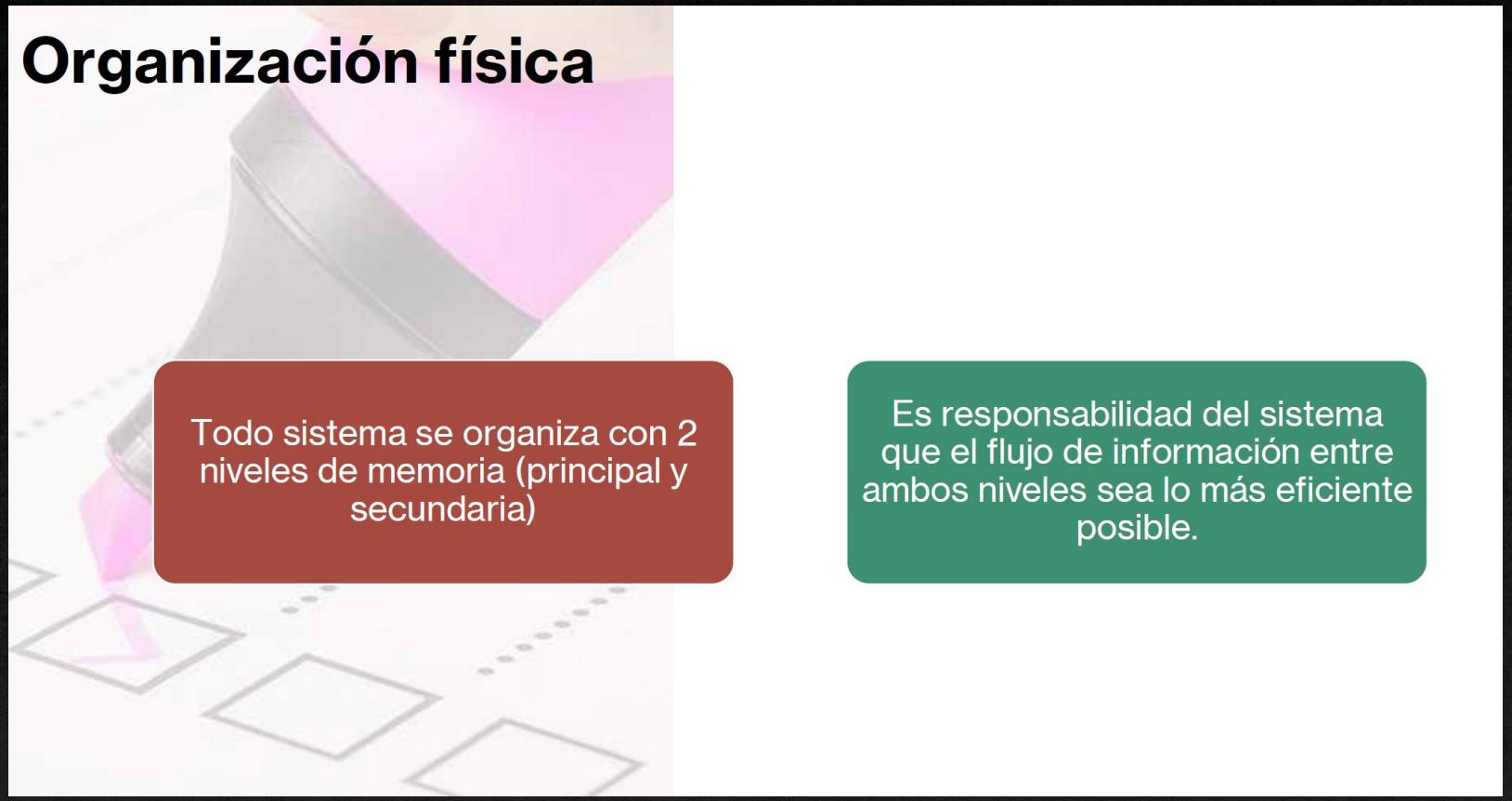
Gestión de Memoria – Organizacion logica - Modularidad



La memoria se organiza como un **array de bytes**, pero una gestión lógica permite:

- ✅ Compilar módulos por separado
- ✅ Que el SO traduzca direcciones
- ✅ Aplicar distintos niveles de protección
- ✅ Compartir módulos entre procesos

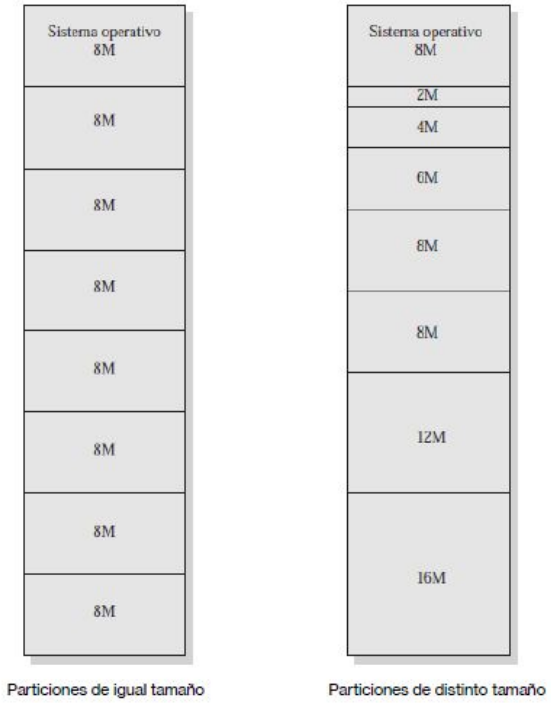
Gestión de Memoria – Organizacion física - Eficiencia



- El sistema usa **memoria principal y secundaria**, y debe **gestionar eficientemente el intercambio** de datos entre ambas.

Gestión de Memoria – Particionamiento Fijo

Particionamiento fijo. Dos alternativas.



- Particiones fijas: Un programa podría ser demasiado grande para caber en una partición. En este caso, el programa debe usar overlays, de forma que sólo se necesite una porción del programa en memoria principal en un momento determinado. Cuando se necesita un módulo que no está presente, el programa de usuario debe cargar dicho módulo en la partición del programa, superponiéndolo (overlaying) a cualquier programa o datos que haya allí.
- La utilización de la memoria principal es ineficiente. Cualquier programa ocupa una partición entera malgastando espacio interno de memoria. Este fenómeno se conoce como fragmentación interna.
- Ambos problemas se pueden mejorar, aunque no resolver, utilizando particiones de tamaño diferente.

Particionamiento fijo

- Divide la memoria en bloques fijos.
- Puede causar **fragmentación interna** (espacio desperdiciado).
- Si el programa es muy grande, requiere usar **overlays**.
- Se puede mejorar usando **particiones de distintos tamaños**, aunque no elimina los problemas por completo.

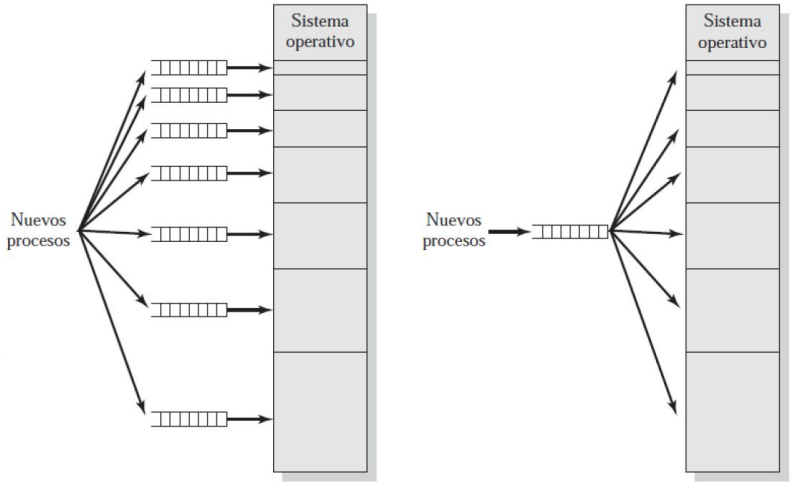
🧩 **¿Qué son los Overlays?**

- Los **overlays** son una técnica que permite **ejecutar programas más grandes que la partición de memoria disponible.**

Gestión de Memoria – Particionamiento de distinto tamaño

Particionamiento fijo de distinto tamaño. Asignación de memoria.

- La forma más sencilla consiste en asignar cada proceso a la partición más pequeña dentro de la cual cabe. En este caso, se necesita una cola de planificación para cada partición, que mantenga procesos en disco destinados a dicha partición.
- La ventaja de esta técnica es que los procesos siempre se asignan de tal forma que se minimiza la memoria malgastada dentro de una partición.
- Desventaja: Si todos los procesos ocupan menos memoria que las particiones mas grandes, estas no se utilizan.



Una cola de procesos por participación

Una única cola

- Una técnica óptima es emplear una única cola para todos los procesos. En el momento de cargar un proceso en la memoria principal, se selecciona la partición más pequeña disponible que puede albergar dicho proceso.
- Si todas las particiones están ocupadas, se debe llevar a cabo una decisión para enviar a swap a algún proceso.

Particionamiento fijo de distinto tamaño

- Se asigna cada proceso a la **partición más pequeña donde quepa**, para reducir desperdicio.
- Puede usarse una **cola por partición** o una **única cola general**.
- Mejora el uso de memoria, pero puede dejar **particiones grandes sin uso** si los procesos son chicos.

🧠**Cola de procesos por participación ¿Cómo funciona?**

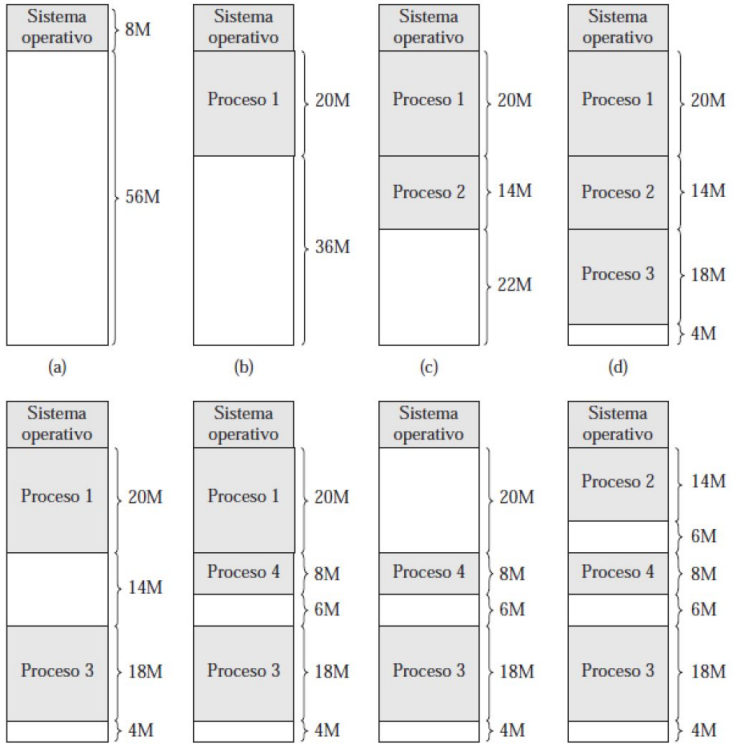
- Cada partición tiene su cola de espera.
- Cada proceso es enviado a la cola de la **partición más pequeña en la que quepa**.
- **Ejemplo:** Si un proceso necesita 4 MB, va a la cola de una partición de 4 MB o la siguiente más cercana.

🧠**Única cola de espera ¿Cómo funciona?**

- Cuando una partición se libera, el sistema busca en la cola el primer proceso que quepa en esa partición.
- Se elige la partición más pequeña disponible que pueda alojarlo..

Gestión de Memoria – Particionamiento dinámico

Particionamiento dinámico.



- Las particiones son de longitud y número variable. Cuando se lleva un proceso a la memoria principal, se le asigna exactamente tanta memoria como requiera.
- Como muestra este ejemplo, el método comienza correctamente, pero finalmente lleva a una situación en la cual existen muchos huecos en la memoria. Este fenómeno se conoce como fragmentación externa, indicando que la memoria que es externa a todas las particiones se fragmenta de forma incremental
- Una técnica para eliminar la fragmentación externa es la compactación: de vez en cuando, el sistema operativo desplaza los procesos en memoria, de forma que se encuentren contiguos y de este modo toda la memoria libre se encontrará unida en un bloque.
- La compactación consume tiempo y malgasta tiempo de procesador.

Particionamiento dinámico

- Asigna a cada proceso **solo la memoria que necesita** (tamaño variable).
- Genera **fragmentación externa** (espacios libres pequeños entre particiones).
- Para corregirlo, se puede usar **compactación** (mover procesos para juntar espacio libre).
- ⚠️ La compactación **consume tiempo del procesador**.

Gestión de Memoria – Particionamiento conclusiones

Particionamiento fijo. Conclusiones

Descripción	• La memoria principal se divide en particiones estáticas en tiempo de generación del sistema. Un proceso se puede cargar en una partición con igual o superior tamaño.
Virtudes	• Sencilla de implementar; poca sobrecarga para el sistema operativo.
Defectos	• Uso ineficiente de la memoria, debido a la fragmentación interna; debe fijarse el número máximo de procesos activos.

Particionamiento dinámico. Conclusiones

Descripción	• Las particiones se crean de forma dinámica, de tal forma que cada proceso se carga en una partición del mismo tamaño que el proceso.
Virtudes	• No existe fragmentación interna; uso mas eficiente de memoria principal.
Defectos	• Uso ineficiente del procesador, debido a la necesidad de compactación para evitar la fragmentación externa.

Resumen: Particionamiento fijo vs dinámico

Tipo	✅ Virtud	❌ Defecto
Fijo	Simple de implementar	Fragmentación interna y límite de procesos
Dinámico	Mejor uso de memoria (sin frag. int.)	Requiere compactación (consume CPU)

📌 Fijo: divide memoria en bloques estáticos.

📌 Dinámico: crea particiones según el tamaño de cada proceso.

Gestión de Memoria – Sistema Buddy – (estático y dinámico)

Sistema Buddy

- Este sistema surge como mejora al particionamiento estático y dinámico
- Lo que se hace es dividir los bloques de memoria a la mitad hasta obtener un bloque que se ajuste lo mejor posible a la petición de memoria
- El sistema Buddy propone que los bloques de memoria disponible son de tamaño 2^k , con $L \leq k \leq U$, donde:

  2^L = bloque de tamaño más pequeño

  2^U = bloque de tamaño máximo.

Algoritmo:

1. Antes de comenzar se asume que se tiene un único bloque de tamaño 2^U .
2. Si el proceso a cargar tiene un tamaño s, tal que $2^{U-1} < s < 2^U$ se asigna el bloque de tamaño 2^U entero.
1. Si la relación del paso anterior no se cumple, entonces el bloque se divide en 2 bloques de tamaño 2^{U-1}.
2. Si el proceso a cargar tiene un tamaño s, tal que $2^{U-2} < s < 2^{U-1}$ se asigna el bloque de tamaño 2^{U-1} entero.

En caso de que no se cumpla el paso 4, volver al paso 3.

El proceso se repite hasta encontrar el bloque de menor tamaño que pueda almacenar el proceso.

Sistema Buddy

- El **Sistema Buddy** es un método de gestión de memoria que **divide bloques en potencias de 2** para asignar espacio de forma más flexible y eficiente.
- Permite **reducir fragmentación** al dividir y unir bloques dinámicamente según las necesidades de los procesos.

Gestión de Memoria – Sistema Buddy

Sistema Buddy. Ejemplo

1	Bloque de 1 Mbyte	1 M					$2^U = 1\,M$
2	A Solicitar 100 K	A = 128 K	128 K	256 K	512 K		
3	B Solicitar 240 K	A = 128 K	128 K	B = 256 K	512 K		
4	C Solicitar 64 K	A = 128 K	C = 64 K	64 K	B = 256 K	512 K	
5	D solicitar 256 K	A = 128 K	C = 64 K	64 K	B = 256 K	D = 256 K	256 K
6	Liberar B	A = 128 K	C = 64 K	64 K	256 K	D = 256 K	256 K
7	Liberar A	128 K	C = 64 K	64 K	256 K	D = 256 K	256 K
8	E Solicitar 75 K	E = 128 K	C = 64 K	64 K	256 K	D = 256 K	256 K
9	Liberar C	E = 128 K	128 K	256 K	D = 256 K	256 K	
10	Liberar E	512 K			D = 256 K	256 K	
11	Liberar D	1 M					

Línea 1 – Bloque de 1 Mbyte

- Se parte de un único bloque de **1 MB** (2^20).

Línea 2 – A solicita 100K

- No hay bloque exacto de 100K, se usa el bloque **más pequeño que lo contiene**: 128K.
- El sistema divide 1MB en partes hasta lograrlo:
  - Se genera un bloque de **128K para A**, otro bloque igual y uno de **256K** y **512K** libres.

Línea 3 – B solicita 240K

- Se necesita al menos un bloque de **256K** para contenerlo (es la potencia de 2 más cercana).
- Se le asigna **uno de los bloques de 256K disponibles**.

Línea 4 – C solicita 64K

- Se busca el mínimo bloque que lo contenga: **64K**, potencia de 2.
- Se divide uno de los bloques de 128K restantes.
- El resultado: un bloque de **64K para C** y otro de 64K libre.

Línea 5 – D solicita 256K

- Se asigna **el bloque de 256K** que todavía está libre.

Línea 6 – Liberar B

- El bloque de 256K ocupado por B se libera.
- Ahora tenemos: A (128K), C (64K), B liberado (256K), D (256K), y bloques libres.

Línea 7 – Liberar A

- El bloque de 128K de A se libera.
- El sistema puede ahora verificar si **hay buddies** para fusionar.

Línea 8 – E solicita 75K

- Necesita al menos **128K** → se usa el bloque libre de 128K (antes de A).
- Se asigna a E.

Gestión de Memoria – Sistema Buddy – Continuación

Sistema Buddy. Ejemplo

1	Bloque de 1 Mbyte	1 M					2^U = 1 M
2	A Solicitar 100 K	A = 128 K	128 K	256 K	512 K		
3	B Solicitar 240 K	A = 128 K	128 K	B = 256 K	512 K		
4	C Solicitar 64 K	A = 128 K	C = 64 K	64 K	B = 256 K	512 K	
5	D solicitar 256 K	A = 128 K	C = 64 K	64 K	B = 256 K	D = 256 K	256 K
6	Liberar B	A = 128 K	C = 64 K	64 K	256 K	D = 256 K	256 K
7	Liberar A	128 K	C = 64 K	64 K	256 K	D = 256 K	256 K
8	E Solicitar 75 K	E = 128 K	C = 64 K	64 K	256 K	D = 256 K	256 K
9	Liberar C	E = 128 K	128 K	256 K	D = 256 K	256 K	
10	Liberar E	512 K			D = 256 K	256 K	
11	Liberar D	1 M					

- **Línea 8 – E solicita 75K**
  - Necesita al menos **128K** → se usa el bloque libre de 128K (antes de A).
  - Se asigna a E.
- **Línea 9 – Liberar C**
  - Se libera el bloque de 64K usado por C.
  - El sistema puede comprobar si ese 64K tiene su "buddy" libre para recombinarse en 128K.
- **Línea 10 – Liberar E**
  - Se libera el bloque de 128K que tenía E.
  - Ahora puede recombinarse con el bloque de 64K (si también está libre) → se ve que aparecen de nuevo 512K disponibles.
- **Línea 11 – Liberar D**
  - Al liberar D (256K), todos los bloques quedan libres y se **reconstruye el bloque de 1MB completo**.

Sistema Buddy

- Gestiona la memoria dividiéndola en **bloques de tamaño 2^n**.
  - Cuando un proceso solicita memoria, se le asigna el **bloque más pequeño posible** que lo contenga.
  - Si no hay un bloque exacto, se **divide uno mayor** hasta ajustarlo.
  - Al liberar memoria, si su "buddy" (bloque hermano) también está libre, **se combinan** para formar un bloque más grande.

Gestión de Memoria – Tipos de direcciones de memoria

Tipos de Direcciones de Memoria

Dirección **Lógica o Relativa**

- Es una referencia independiente de la ubicación real en memoria.
- **Debe ser traducida** antes de ser usada.
- Se define **respecto a una dirección base**, conocida por el procesador.

Dirección **Física o Absoluta**

- Es la **dirección real** en la memoria principal.

Traducción de direcciones (2 pasos)

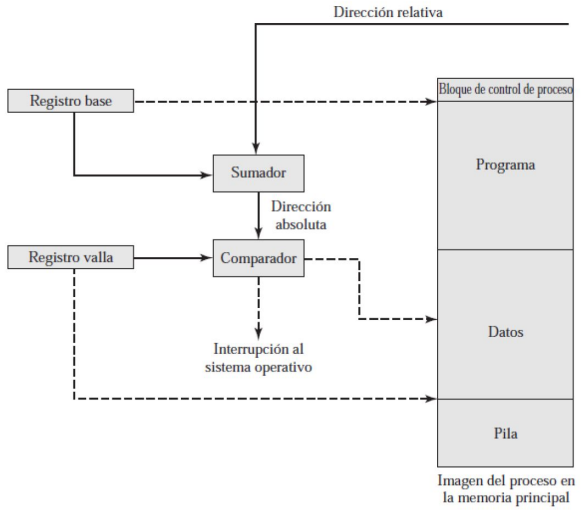
1. **El procesador** traduce la dirección lógica sumando la **dirección base**.
2. Verifica que esté **dentro de los límites permitidos** del proceso.

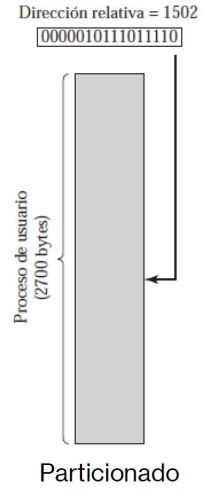
⚠ Si no lo está, se genera un **segmentation fault** (falla de segmento).

Paso a paso:

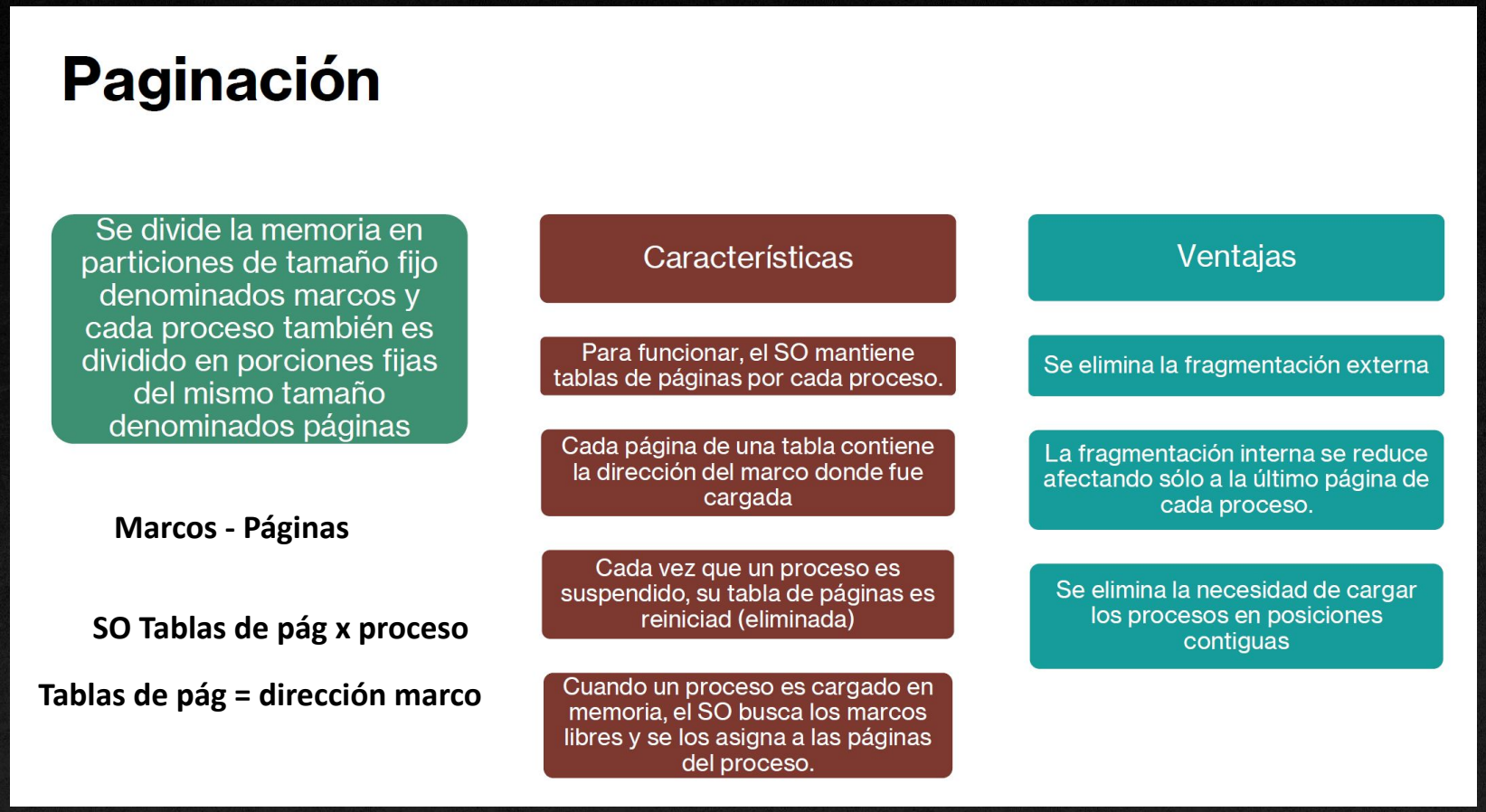
1. **El programa da una dirección lógica** (ej: 1502).
2. **El sistema le suma la base del proceso** (registro base).
   → Resultado: **dirección física** real en memoria.
3. **Se verifica** si está dentro del área asignada (registro valla).
4. ⚠ Si no lo está → **error de memoria (segmentation fault)**.

Traducción a dirección física





Gestión de Memoria – Paginación



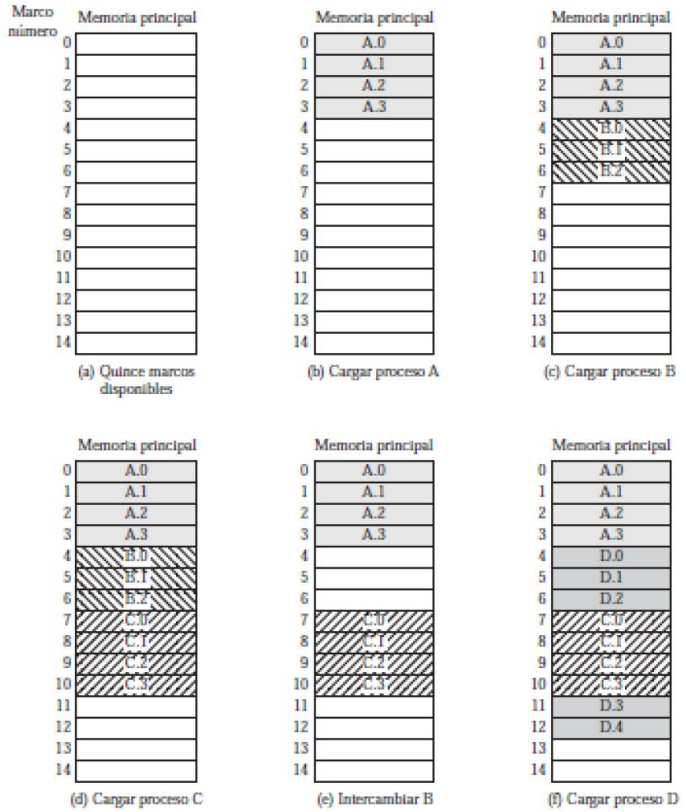
¿Qué es la paginación?

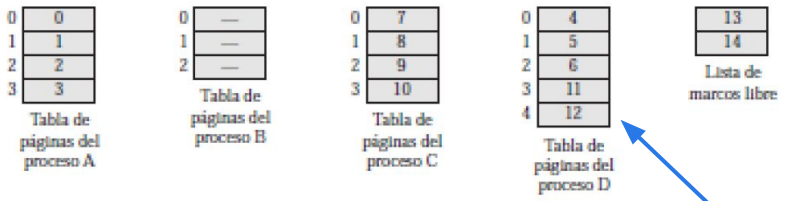
La paginación es una forma de administrar la memoria dividiéndola en bloques pequeños y fijos:

- La **memoria RAM** se divide en **marcos** (frames).
- Cada **proceso** se divide en **páginas** del mismo tamaño.
  🔁 Cada página del proceso se guarda en un marco libre de memoria.

Gestión de Memoria – Paginación

Paginación. Ejemplo





Intercambiar (swap out) proceso B

- Las páginas de B se eliminan de la memoria.
- Los marcos 4, 5 y 6 quedan **libres**.

Cargar proceso D

- D tiene 5 páginas: D.0 a D.4.
- El sistema **reutiliza** los marcos libres 4, 5 y 6 (dejados por B).
- También usa los marcos 11 y 12 para completar.

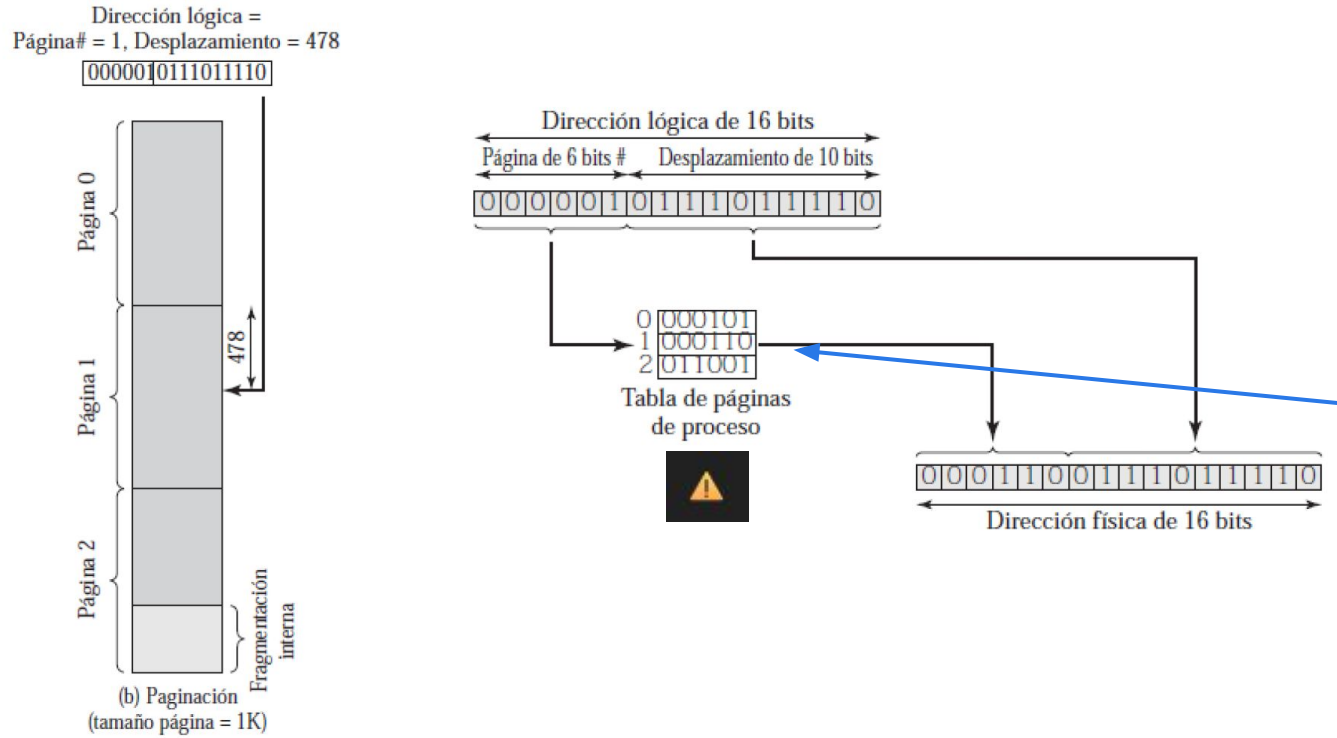
Tablas de páginas

Cada tabla indica en qué marco está cada página del proceso.
Por ejemplo:

- **Proceso A**: página 0 está en marco 0, página 1 en el 1, etc.
- **Proceso D**: sus páginas están distribuidas en marcos 4, 5, 6, 11 y 12.

Gestión de Memoria – Traducción direcciones físicas

Traducción a direcciones físicas



¿Qué muestra?

- Cómo se convierte una **dirección lógica (virtual)** en una **dirección física (real en memoria)** usando **paginación**.

- **Dirección lógica** de 16 bits = página `1`, desplazamiento `478`.
- Se separa en:
  - **6 bits** para número de página.
  - **10 bits** para desplazamiento dentro de la página.

Proceso:

1. **Página 1** → Se busca en la **tabla de páginas**.
2. Esa entrada da el **marco físico** donde está la página.
3. Se le suma el **desplazamiento** → se obtiene la **dirección física final**.

Resultado:

- Se accede correctamente al dato **en RAM**.
- El proceso **no necesita memoria contigua**.
- Se maneja de forma **segura y flexible**.

- Nro de página.
- El contenido de la dirección del número de página conforma la dirección física.

Gestión de Memoria – Segmentación

Segmentación

Es una técnica avanzada que divide la memoria principal y los procesos en porciones fijas

Direcciones mem conjunto =código + datos

Características

Ve al espacio de direcciones de memoria de un proceso como un conjunto de segmentos (código + datos)

Cada uno de estos segmentos es una unidad lógica que permite al programador organizar el código

Permite al SO ubicarlos en porciones de memoria distintas que no necesariamente deben ser contiguas.

Al igual que con la paginación, el SO crea tablas de segmentos por cada proceso y una lista de bloques de memoria libres .

Ventajas

Se elimina la fragmentación interna

Se elimina la necesidad de cargar los procesos en posiciones contiguas

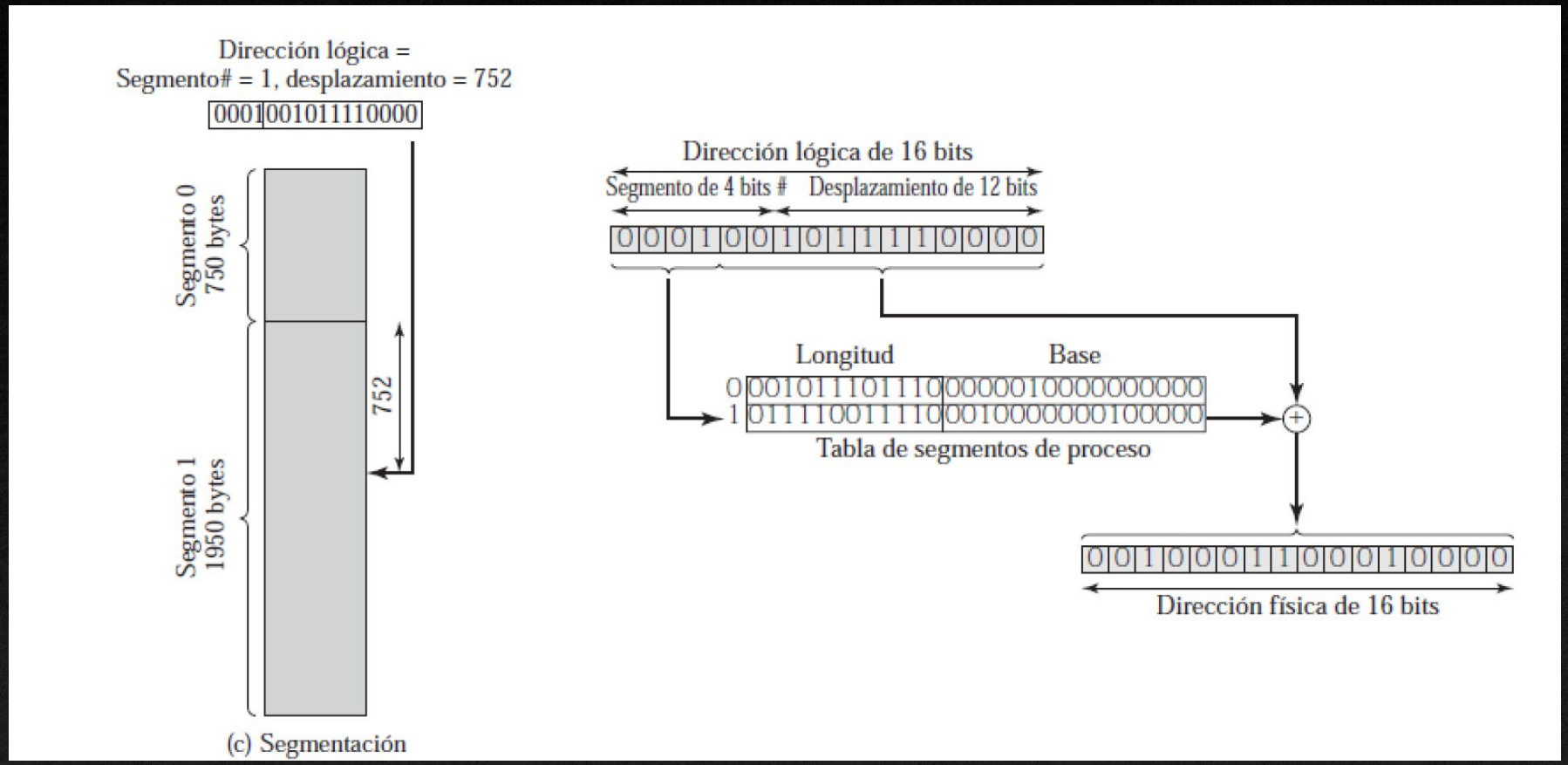
Posibilita que varios procesos compartan el segmento de código y el de datos.

- La **segmentación** divide la memoria y los procesos en **partes lógicas** llamadas **segmentos** (como código, datos, pila).
- Cada segmento representa una **unidad lógica del programa**.
- El sistema puede **ubicar segmentos en distintas partes** de la memoria.
- Se usan **tablas de segmentos** para acceder.

✅ **Ventajas**

- ❌ Se evita la **fragmentación interna**.
- 🧠 Permite una **organización más natural del código**.
- 🔄 Segmentos pueden ser **compartidos entre procesos**.

Gestión de Memoria – Traducción direcciones físicas

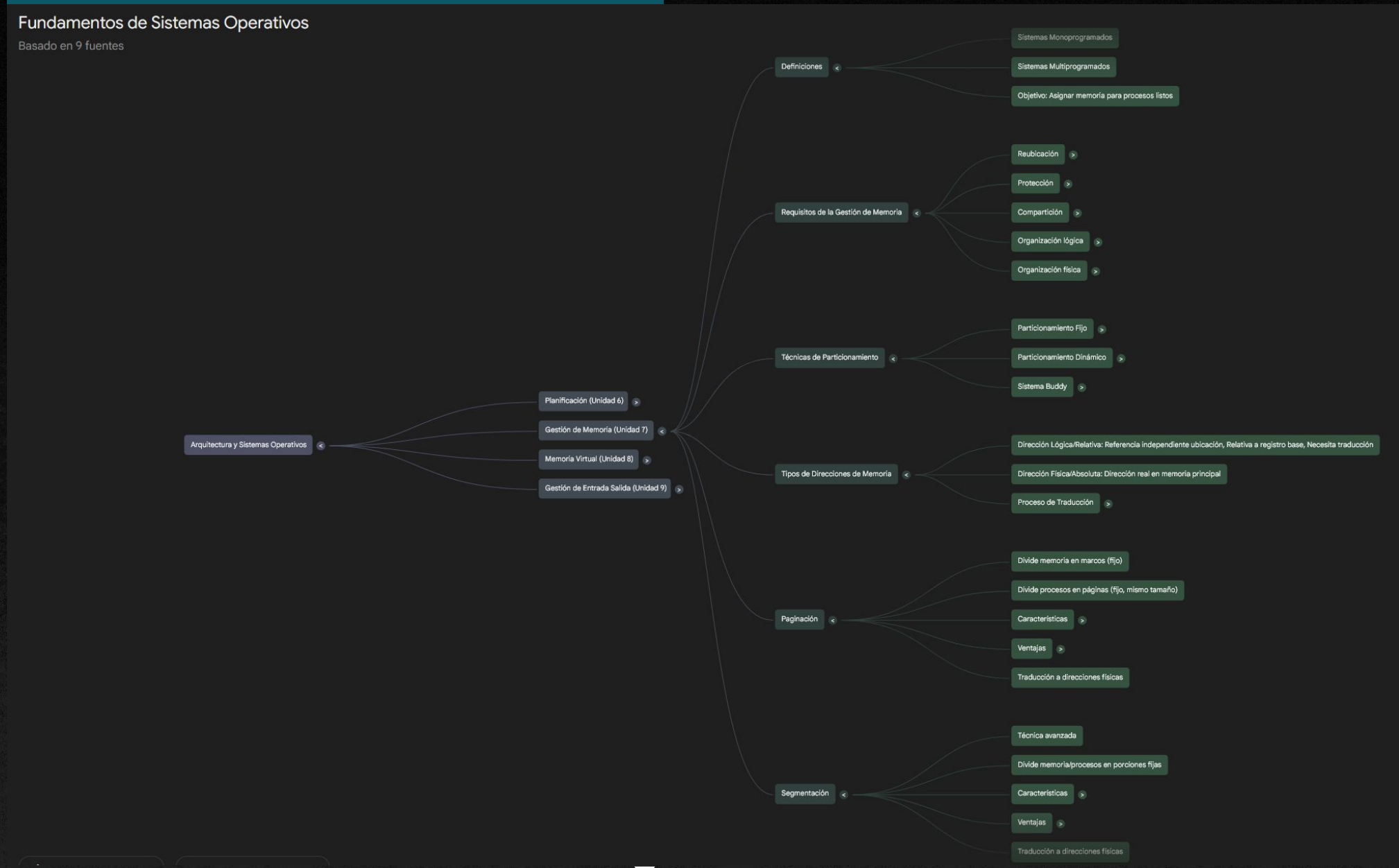


- Similar a la traducción por paginación, pero ahora por segmento.

Resumen – Gestión de Memoria

- Requisitos:
  - Reubicación
  - Protección
  - Compartición
  - Organización lógica/física.
- Particionamiento:
  - Fijo (Fragmentación Interna)
  - Dinámico (Externa).
  - Buddy System (Bloques 2^k).
- Direcciones:
  - Lógicas vs Físicas
  - Traducción necesaria.

Resumen – Gestion de Memoria – Mapa Conceptual



Clase 8

- Paginación
- Conjunto residente
- Políticas del Sistema Operativo

Memoria Virtual

Memoria Virtual - Definiciones

Características de la paginación y la segmentación

Direcciones lógicas	• Todas las referencias a memoria que realiza un proceso se realizan a direcciones lógicas
Traducción	• Las direcciones de memoria de un proceso se traducen en el momento en que el proceso es cargado.
Fragmentos	• Un proceso puede dividirse en páginas o segmentos
Ubicación	• Los fragmentos pueden ser cargados en posiciones no contiguas

⚠️ **Necesidad de la Memoria Virtual**

- Para ejecutar un proceso, **todas sus páginas o segmentos deben estar en RAM**.
- Si la RAM **no alcanza**, el sistema usa **memoria virtual** (por ejemplo, en disco).
- Esto permite **ejecutar programas más grandes** que la memoria física disponible.

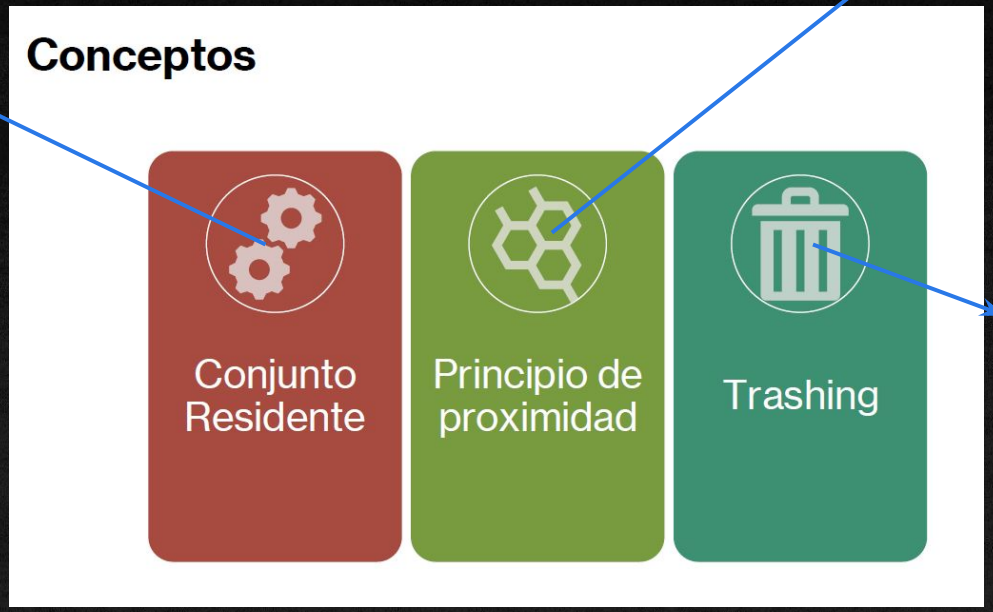
Memoria Virtual - Conceptos

⚙️ Conjunto Residente

Es el grupo de páginas o segmentos que **deben estar siempre en memoria** para que un proceso se ejecute correctamente.

✔️ **Beneficios:**

- Permite que un proceso sea **más grande que la RAM**.
- Mejora el grado de **multiprogramación**.



📍 Principio de Proximidad

Las referencias a memoria **tienden a agruparse** en zonas cercanas (ej. bucles, funciones).

✔️ **Implica que:**

- Solo se necesita tener en memoria **una parte del proceso** durante un tiempo corto.

🗑️ Trashing

Ocurre cuando el sistema **pasa más tiempo intercambiando páginas** que ejecutando instrucciones.

✔️ **Causado por:**

- Tener **muy poco conjunto residente**.
- Reemplazar constantemente páginas que serán **usadas nuevamente en breve**.

Memoria Virtual – Paginación Multinivel

Además de los procesos...

📚 Tabla de Páginas Multinivel

- Consiste en **2 o más niveles jerárquicos de tablas de páginas**. Cada nivel apunta al siguiente, hasta llegar a la **tabla final**, que contiene el **marco de memoria real**.

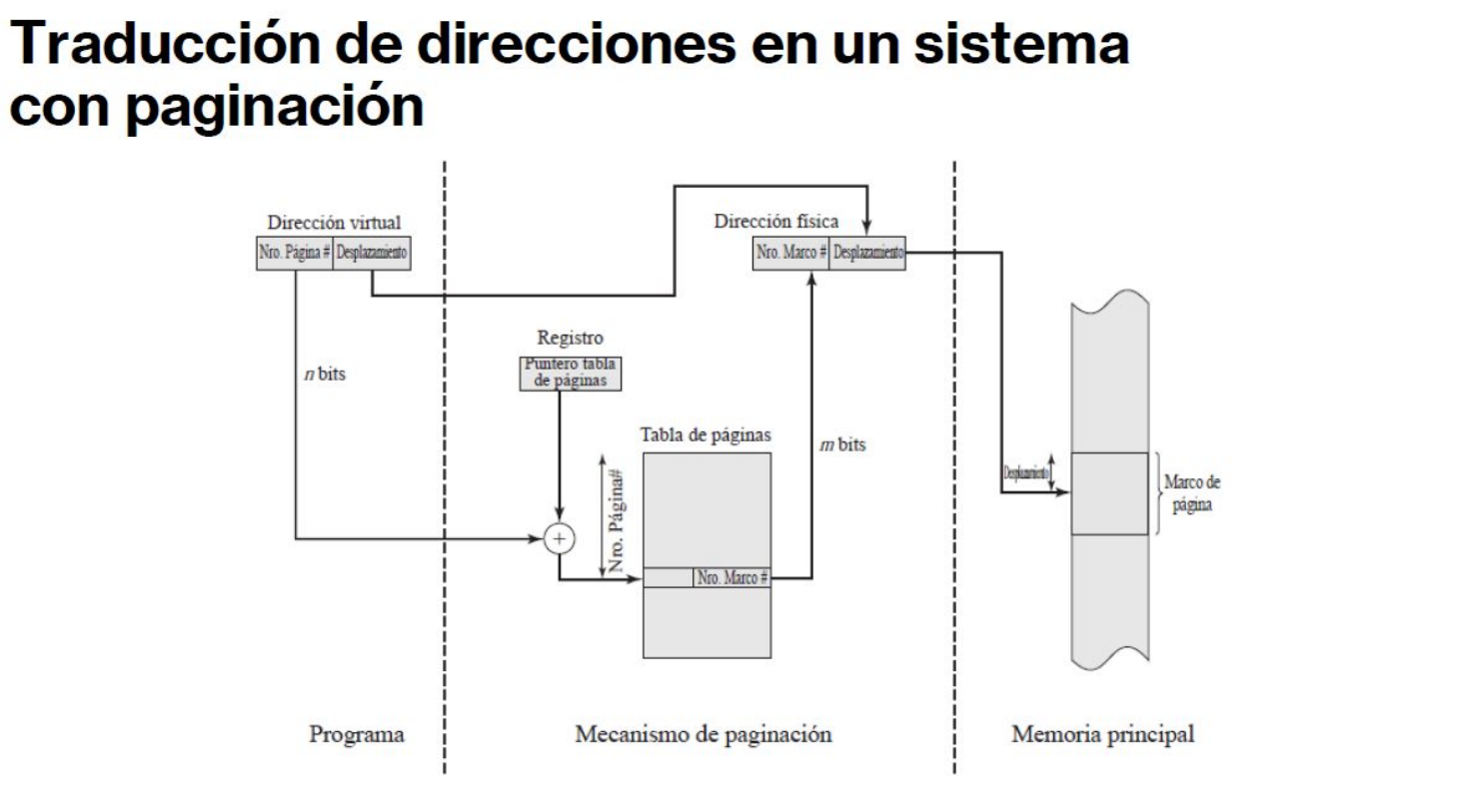
✅ Ventaja

- **Ahorra espacio**: la tabla puede caber en una sola página de memoria.

⚠️ Desventaja

- **Más accesos a memoria** → puede **afectar el rendimiento**.

Memoria Virtual – Paginación Multinivel



Se muestra el proceso mediante el cual una **dirección virtual generada por un programa** se convierte en una **dirección física en la memoria RAM**, usando **paginación**.

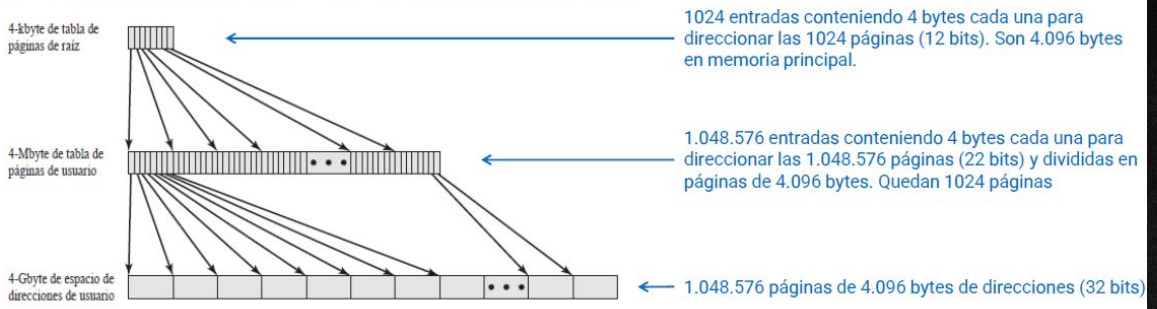
🧾 **Paso a paso:**

1. **Dirección virtual**
   Está compuesta por:
   1. **Número de página**
   2. **Desplazamiento** (offset dentro de la página)
2. **Mecanismo de paginación**
   1. Se utiliza el **número de página** para buscar en la **tabla de páginas**.
   2. La tabla de páginas indica en qué **marco de página** (frame) está esa página cargada.
3. **Dirección física**
   1. Se forma combinando el **número de marco** (obtenido de la tabla) con el **desplazamiento** original.
   2. Resultado: una dirección **real en memoria** donde está el dato.

Memoria Virtual – Paginación Multinivel

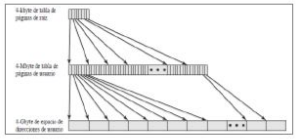
Ejemplo tabla de páginas jerárquicas de dos niveles

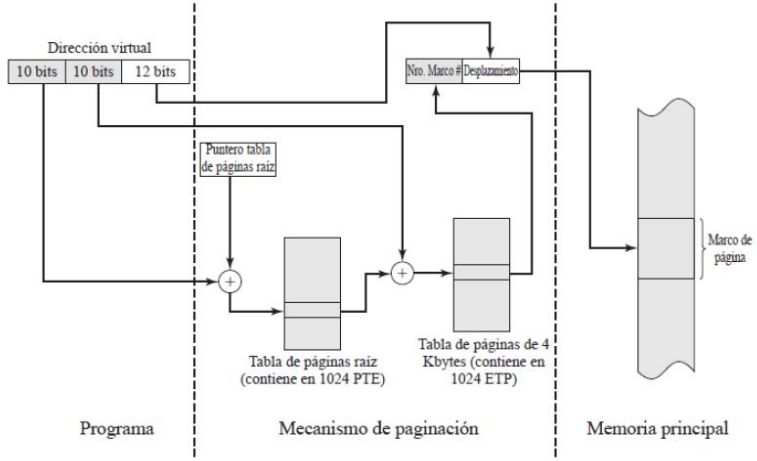
Ejemplo de un esquema de dos niveles que usa 32 bits para la dirección.



Asumimos un direccionamiento a nivel de byte y páginas de 4 Kbytes (2^{12}), por lo tanto, el espacio de direcciones virtuales de 4 Gbytes (2^{32}) se compone de 2^{20} páginas. Si cada una de estas páginas se referencia por medio de una entrada en la tabla de páginas (ETP) de 4-bytes, podemos crear una tabla de página de usuario con 2^{20}. La ETP requiere 4 Mbytes (2^{22} bytes). Esta tabla de páginas de usuario, que ocupa 2^{10} páginas, puede mantenerse en memoria virtual y hacerse referencia desde una tabla de páginas raíz con 2^{10} PTE que ocuparía 4 Kbytes (2^{12}) de memoria principal.

Traducción de direcciones para este ejemplo





La página raíz siempre se mantiene en la memoria principal. Los primeros 10 bits de la dirección virtual se pueden usar para indexar en la tabla de páginas raíz para encontrar la ETP para la página en la que está la tabla de páginas de usuario. Si la página no está en la memoria principal, se produce un fallo de página. Si la página está en la memoria principal, los siguientes 10 bits de la dirección virtual se usan para indexar la tabla de páginas de usuario para encontrar la ETP de la página a la cual se hace referencia desde la dirección virtual original.

La **paginación multinivel** organiza las tablas de páginas en forma **jerárquica** para reducir el uso de memoria.
Cada parte de la dirección virtual **actúa como puntero** que guía paso a paso por cada nivel de tabla hasta llegar al marco real en RAM.

✅ Ahorra espacio, pero requiere **más accesos y traducciones** para cada dirección virtual.

Memoria Virtual – Tabla de páginas invertida

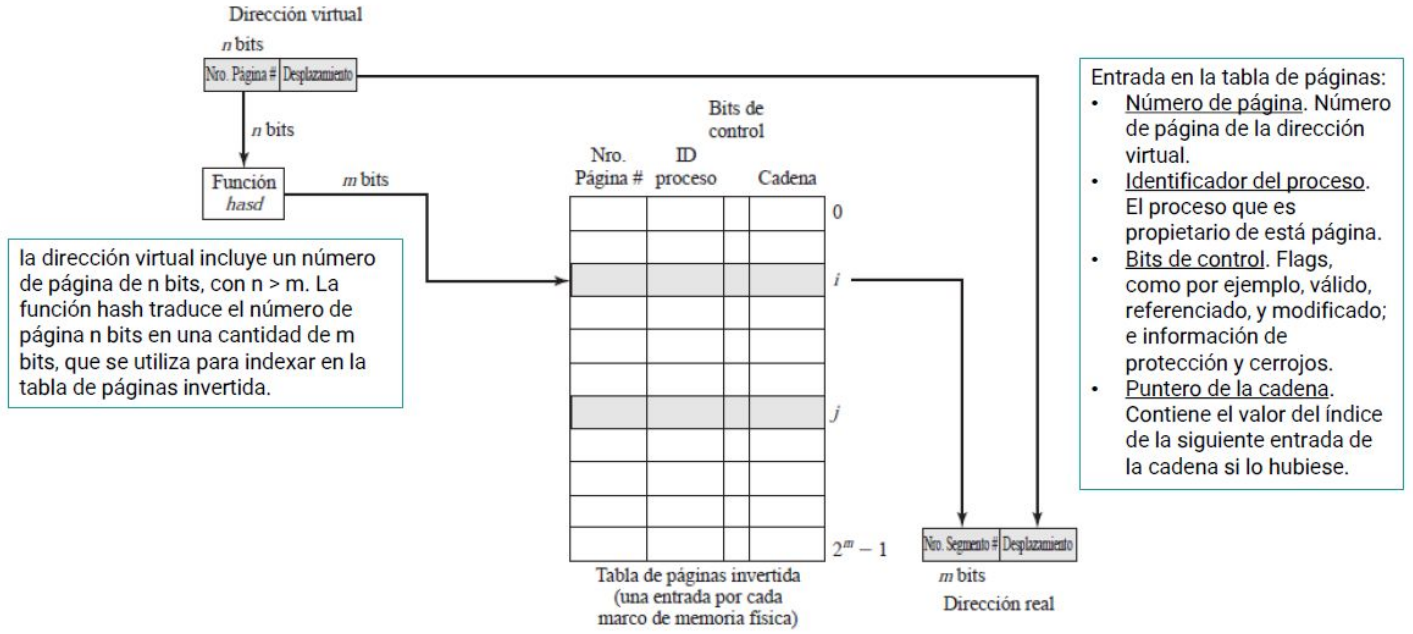
Tabla de páginas invertida

Consiste en una única tabla de páginas que posee una única entrada por cada marco de memoria.

La tabla posee Nro. de página, PID, bits de control, puntero de encadenamiento.

Ventajas	Desventajas
• Se requiere una porción fija de memoria para almacenar la tabla de páginas • Se reducen los accesos a memoria	• Las búsquedas son más complejas, y requieren más tiempo • Puede haber colisiones. Es más compleja la implementación de la memoria compartida

Estructura de página invertida



La tabla de páginas invertida tiene **una entrada por cada marco de memoria física** (no por cada página virtual).

✅ Usa **menos memoria** para la tabla, pero
⚠️ Las búsquedas son más **complejas y lentas** (usa función hash y puede haber colisiones).
Cada entrada incluye: **número de página**, **ID del proceso**, **bits de control** y un **puntero de encadenamiento** si hay más de una coincidencia.

Memoria Virtual – Buffer de traducción anticipada

Buffer de traducción anticipada (TLB)

Toda referencia a memoria virtual involucra 2 accesos a memoria, uno para buscar la entrada en la tabla de páginas y otro para buscar los datos.

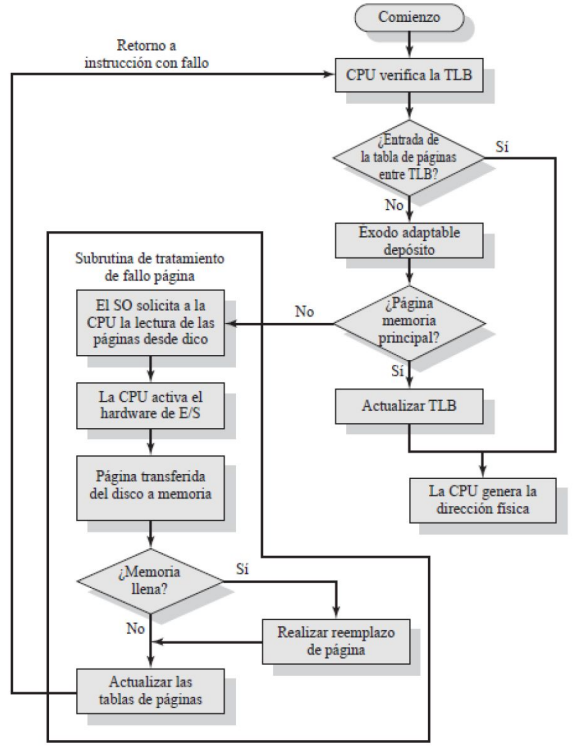
Para optimizar este proceso la mayoría de los esquemas de memoria virtual implementan una memoria cache especial de alta velocidad para las entradas de la tabla de página llamada TLB.

La TLB contiene aquellas entradas de la tabla de páginas que han sido usadas recientemente.

Operación de paginación y TBL

El proceso de traducción de una dirección lógica es como sigue:

1. El procesador busca la entrada en la tabla de páginas en la TLB.
   - En caso de encontrarse se recupera el número de marco y se construye la dirección real.
2. Si no se encuentra la entrada en la TLB (fallo de TLB), el procesador accede a la tabla de página. Se recupera el marco y se construye la dirección real.
   - La entrada de la tabla de páginas accedida será cargada en la TLB, para futuras referencias.
3. Si la referencia no se encuentra en memoria, se produce un fallo de página y el control es cedido al sistema operativo para que cargue la página solicitada.
   - La entrada de la tabla de páginas accedida será cargada en la TLB, para futuras referencias.



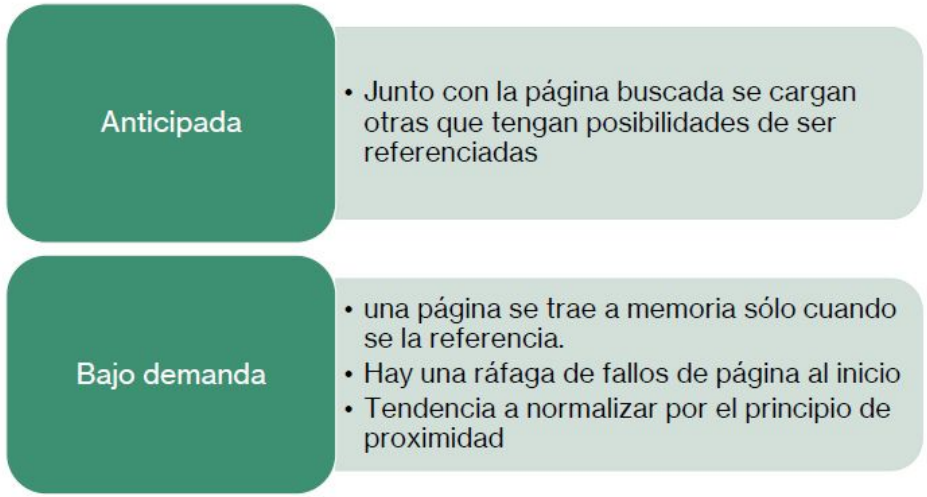
- **La TLB (Translation Lookaside Buffer)** es una memoria caché rápida que almacena temporalmente las últimas traducciones de direcciones virtuales a físicas.
- Guarda entradas recientes de la tabla de páginas y evita hacer dos accesos a memoria por cada referencia virtual, lo que mejora notablemente el rendimiento del sistema.

Cuando la CPU necesita acceder a memoria, primero busca la traducción en la TLB.

Si está, obtiene la dirección física directo.
Si no está (fallo de TLB), consulta la tabla de páginas.
Si la página no está cargada, ocurre un fallo de página y el sistema operativo trae la página desde disco.
Finalmente se actualiza la TLB y la tabla de páginas para futuras referencias.

Memoria Virtual – Políticas del Sistema Operativo

Políticas de recuperación de páginas



Políticas de reemplazo

Tienen como objetivo seleccionar la página que tiene menos posibilidades de ser referenciada en un tiempo corto.

- **Anticipada**
- Carga más de una página a la vez.
- Se basa en la probabilidad de que otras páginas cercanas también se necesiten.
- Ahorra tiempo si las predicciones son correctas

- **Bajo demanda**
- Solo se carga una página cuando es estrictamente necesaria.
- Puede causar muchos fallos al principio.
- Se estabiliza gracias al principio de localidad.

- **Políticas de reemplazo**
- El objetivo es liberar memoria eligiendo la página **menos probable de ser usada pronto.**
- Se aplican cuando ya no hay marcos libres disponibles.

Memoria Virtual – Políticas del Sistema Operativo

Políticas de reemplazo. Algoritmos básicos

Óptimo

- Reemplaza la página cuyo próximo instante de referencia se encuentre más alejado en el tiempo
- Es imposible de implementar.

Referencia Futura: mas alejado en el tiempo.

LRU

- Reemplaza la página que no se haya referenciado por mucho tiempo
- Presenta un buen rendimiento teórico pero es muy costoso de implementar.

Menos Usado: reemplaza la menos usada

FIFO

- Trata los marcos de página como si fuese un buffer circular reemplaza las páginas mediante una estrategia cíclica de tipo round robin
- Si bien es simple de implementar, presenta un muy bajo Rendimiento

Primero en entrar: saca la mas antigua

Reloj

- Similar a FIFO, sólo que agregar un "bit de usado" a las páginas.
- Se reemplaza la página cuyo bit se encuentre en 0 (cero)

Bit de uso: forma circular eficiente.

Reloj mejorado

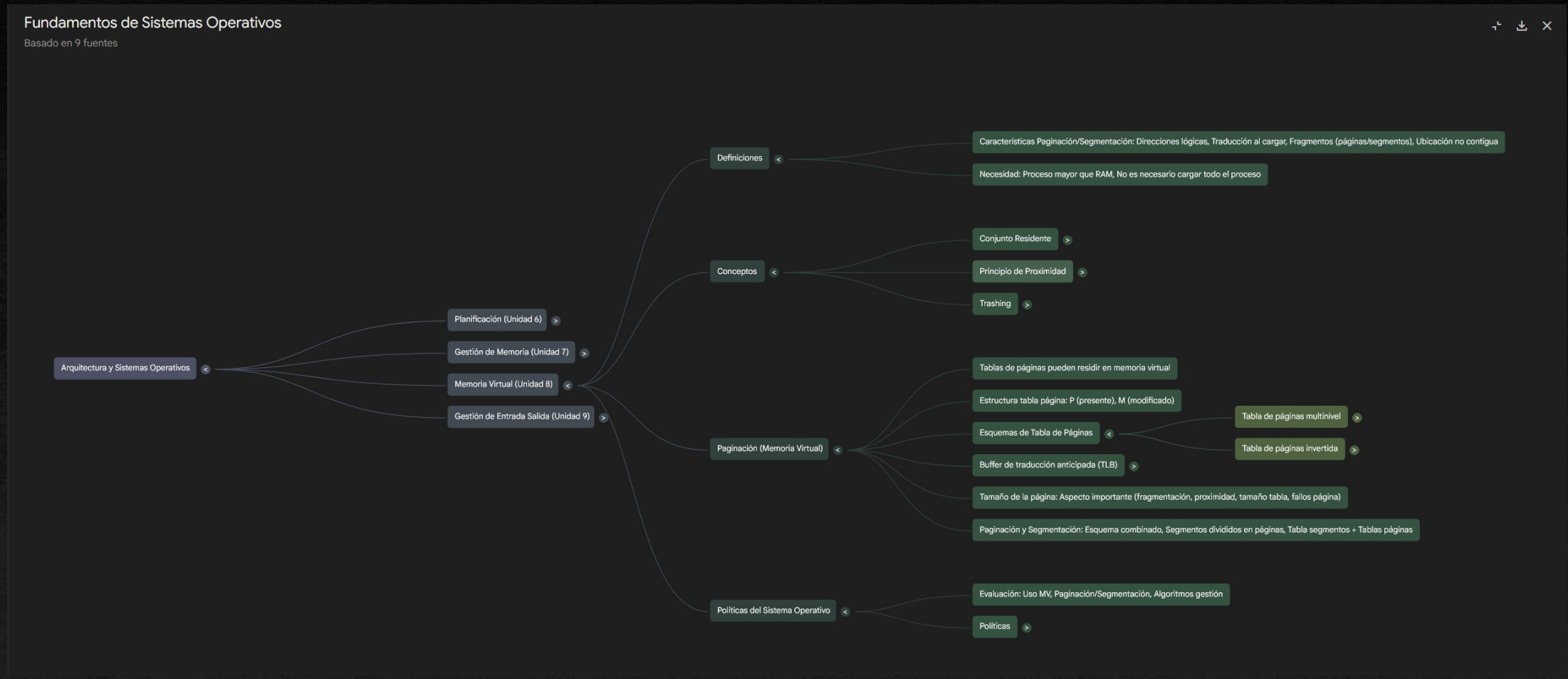
- Agrega un "bit de modificado" a las páginas. Con esto evita escribir la página a disco si no fueron modificadas

Bit de uso: + modificación

Resumen – Memoria Virtual

- Permite ejecutar procesos mayores que la RAM.
- Paginación:
  - Multinivel
  - Invertida
- TLB (Translation Lookaside Buffer): Cache de direcciones
- Políticas:
  - Anticipada / Bajo demanda
  - Remplazo:
    - Optimo
    - LRU (Least Recently Used)
    - FIFO (First in First Out)
    - Reloj

Resumen – Memoria Virtual – Mapa Conceptual



Clase 9

- Evolución de E/S
- Buffers
- Planificación disco
- RAID

Gestión de Entrada/Salida

Gestión de Entrada / Salida - Técnicas

Técnicas de E/S

E/S programada.	• El procesador envía un mandato de E/S a un módulo de E/S; a continuación, ese proceso realiza una espera activa hasta que se complete la operación antes de continuar.
E/S dirigida por interrupciones.	• El procesador emite un mandato de E/S y continúa ejecutando las instrucciones siguientes, siendo interrumpido por el módulo de E/S cuando éste ha completado su trabajo.
Acceso directo de memoria (DMA).	• Un módulo de DMA controla el intercambio de datos entre la memoria principal y un módulo de E/S. El procesador manda una petición de transferencia de un bloque de datos al módulo de DMA y resulta interrumpido sólo cuando se haya transferido el bloque completo.

E/S programada

- CPU espera activamente
- Solo continúa tras finalizar la operación

E/S por interrupciones

- CPU sigue ejecutando
- Se interrumpe al terminar la E/S

DMA (Acceso directo a memoria)

- Módulo externo maneja la transferencia
- CPU se interrumpe solo al final del bloque

Gestión de Entrada / Salida - Evolución

Evolución en el tiempo

1. El procesador controla directamente un dispositivo periférico. Esta situación se presenta en dispositivos simples controlados por un microprocesador.

2. Se añade un controlador o módulo de E/S. El procesador usa E/S programada sin interrupciones. Con este paso, el procesador se independiza de los detalles específicos de las interfaces de los dispositivos externos.

3. Se utiliza la misma configuración que en la etapa anterior, pero empleando interrupciones. El procesador no necesita gastar tiempo esperando a que se realice una operación de E/S, incrementando de esta manera la eficiencia.

4. Al módulo de E/S se le da control directo de la memoria mediante DMA. Con ello, puede mover un bloque de datos a la memoria sin involucrar el procesador, excepto al principio y al final de la transferencia.

5. Se mejora el módulo de E/S para convertirse en un procesador independiente. La CPU hace que el procesador ejecute un programa de E/S residente en la memoria principal. El procesador de E/S lee y ejecuta estas instrucciones sin la intervención del procesador.

6. El módulo de E/S tiene su propia memoria local. y se pueden controlar un gran conjunto de dispositivos de E/S, con una intervención mínima por parte del procesador.

1. CPU controla directamente el periférico (sistemas simples).

2. Se agrega módulo de E/S, pero con E/S programada.

3. Se incorpora interrupciones, la CPU ya no espera activamente.

4. DMA permite transferencias sin CPU, solo al inicio y final.

5. El módulo de E/S ejecuta programas propios (procesador autónomo).

6. Módulo de E/S con memoria local y mínima intervención del procesador.

Gestión de Entrada / Salida - Buffers

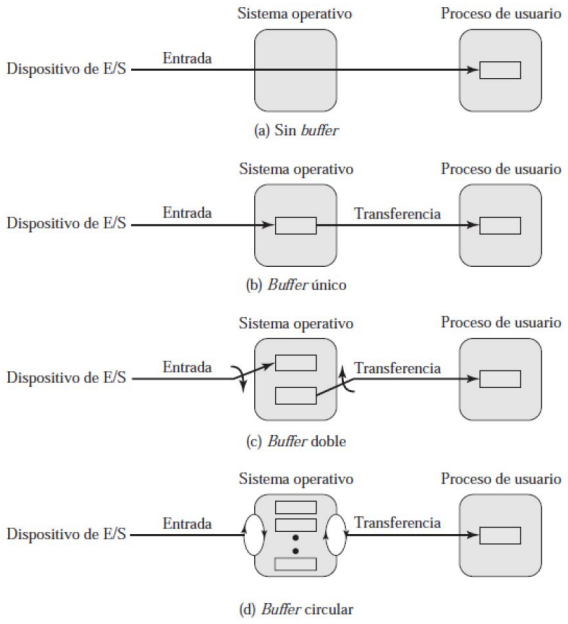
Situación hipotética

Un proceso debe leer datos de disco y escribirlos en una zona de datos del espacio de direcciones del proceso de usuario.

Dos opciones: espera activa, o suspender el proceso hasta que se complete la petición de E/S.

- Espera activa: hace un uso ineficiente del procesador
- Suspender el proceso: si se utiliza paginación de memoria, la página de la zona de datos del espacio de direcciones donde deben escribirse los datos debe estar en memoria. Si se suspende el proceso y la página se expulsa, se **genera un interbloqueo de un solo proceso.**

Tipos de buffers



- **Buffer Único**: Los datos se ingresan hasta llenar el buffer del SO. Luego son enviados al área de datos del proceso de usuario. Permite expulsar el proceso
- **Buffer Doble**: Los datos cargados en un buffer son consumidos por el proceso mientras el SO llena el otro buffer.
- **Buffer Circular:** Similar a buffer doble pero con más buffers. Esta técnica permite "suavizar" la demanda de E/S

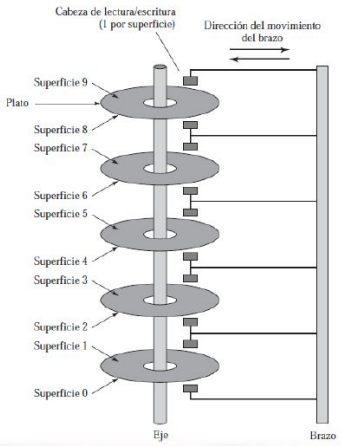
Uso de buffers

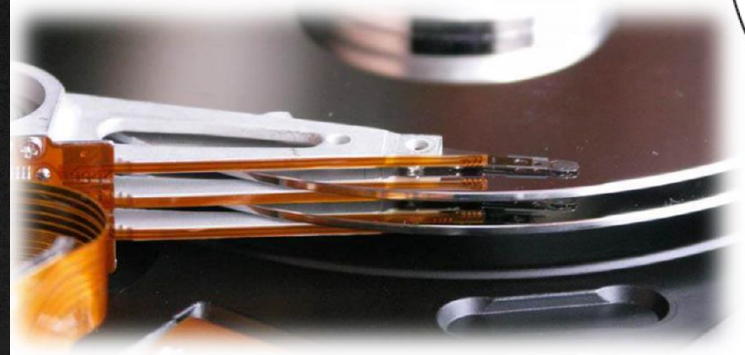
El uso de buffers es una técnica que si bien no permitirá que un dispositivo de E/S funcione a la velocidad del procesador o de la memoria principal, amortigua los picos en la demanda de E/S.

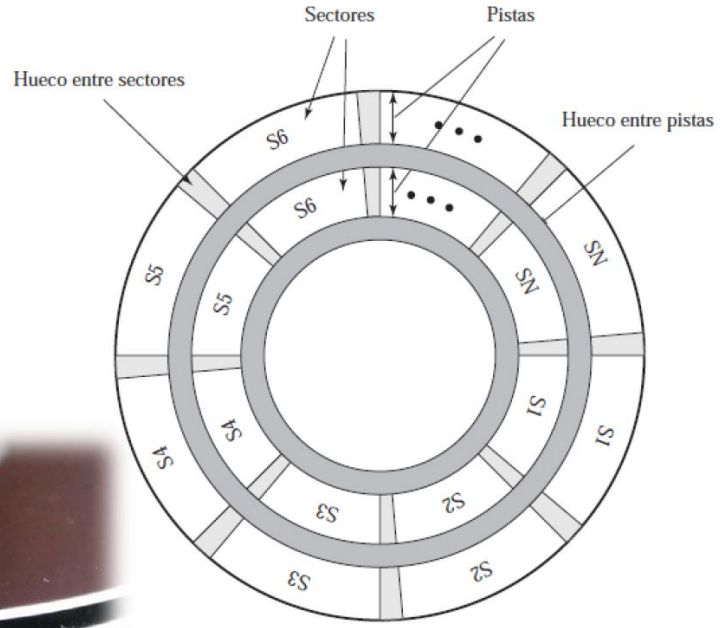
En un entorno multiprogramado, el uso de buffers es una técnica que puede incrementar la eficiencia del sistema operativo y el rendimiento de los procesos individuales.

Gestión de Entrada / Salida – Planificación de Disco

El disco.







Tiempo de búsqueda: el tiempo que se tarda en situar la cabeza en la pista

Retardo rotacional: el tiempo que se tarda en llegar al comienzo del sector

Tiempo de acceso: Tiempo de búsqueda + Retardo rotacional

Tiempo de transferencia: Tiempo que transcurre mientras que el sector se desplaza debajo de la cabeza de lectura/escritura

Gestión de E/ S – Políticas de Planificación

Políticas de planificación

Tienen como objetivo principal **minimizar el tiempo de búsqueda**, es decir reducir la cantidad de movimientos que realiza el cabezal de lectura.

FIFO *First In, First Out*	• Simple y equitativa. Rendimiento pobre
SSTF *Shortest Seek Time First*	• Primero el de tiempo de servicio más corto. Mejor rendimiento que FIFO. Posible starvation (Nunca se ejecuta)
SCAN *Elevator Algorithm*	• Algoritmo del ascensor. El brazo sólo se mueve en una dirección. No sufre starvation.
C-SCAN *Circular SCAN*	• Similar a SCAN, restringiendo el movimiento del brazo en una dirección. Reduce los tiempos de SCAN para algunas peticiones

Gestión de E/ S – RAID

RAID

"Redundant array of independent disks"
o array redundante de discos Independientes.

Es un conjunto de discos que a partir de distintas configuraciones proporcionan mejoras en cuanto a la velocidad de acceso, redundancia y consistencia de los datos.

Tres características fundamentales

1. El sistema operativo los percibe como un único dispositivo lógico.

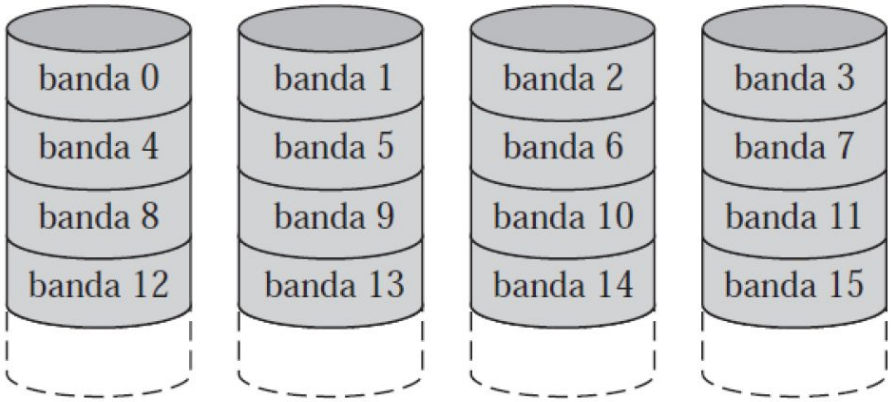
2. Los datos se distribuyen a lo largo de las unidades físicas de un vector

3. La capacidad de redundancia del disco se utiliza para almacenar información de paridad, que garantiza la recuperación de los datos en caso de falla.

Gestión de E/ S – RAID 0

Raid 0

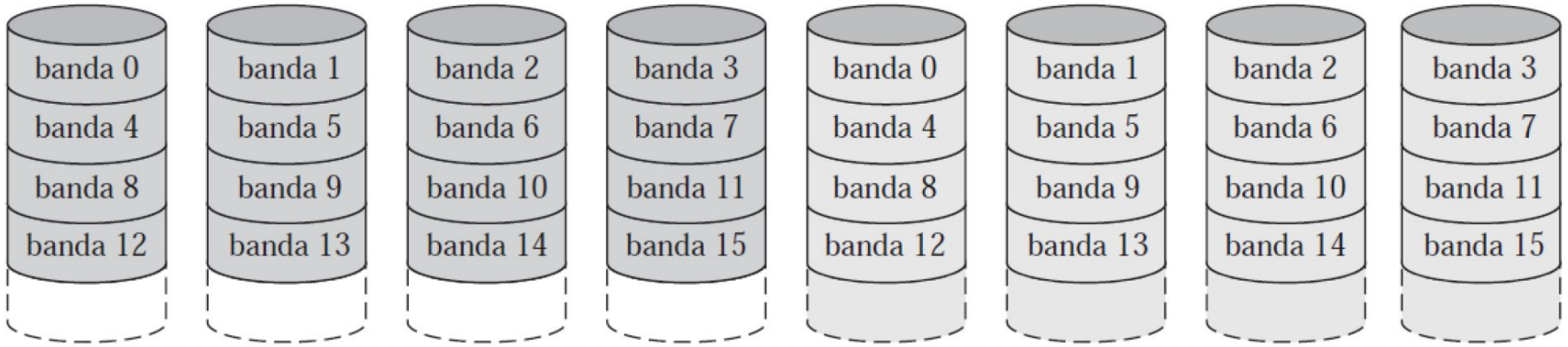
- No presenta redundancia
- Los datos están distribuidos a lo largo de todos los discos
- Facilita las lecturas en paralelo en el caso de que los datos no se encuentren en el mismo disco.



Gestión de E/ S – RAID 1

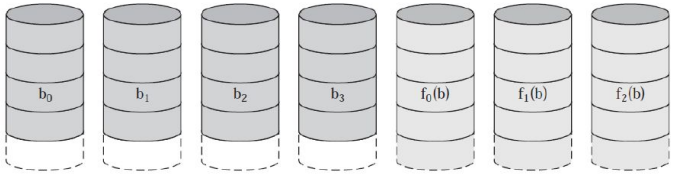
Raid 1 (mirroring)

- Una petición de lectura puede servirse de cualquiera de los 2 discos
- Una petición de escritura requiere actualizar ambas bandas.
- Proporciona tolerancia a fallos. Si un disco se rompe, se pueden recuperar los datos desde el otro disco.
- Requiere doble de espacio

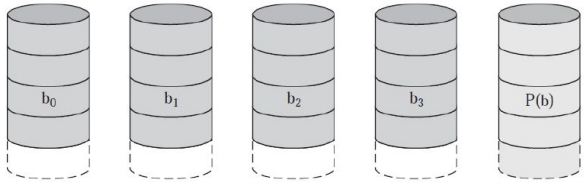


Gestión de E/ S – RAID 2 y 3

RAID 2 y RAID 3



RAID 2 – Corrección de errores mediante código Hamming



RAID 3 – Paridad a nivel bit

Diferencias entre RAID 2 y RAID 3

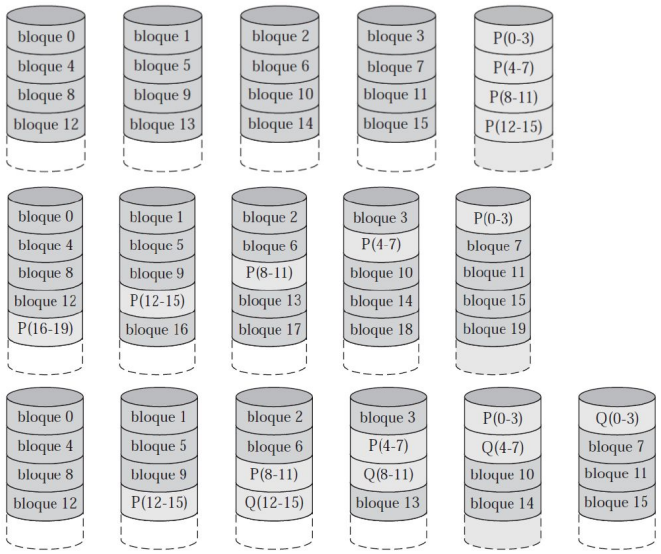
Característica	RAID 2	RAID 3
Método de redundancia	Corrección con **código Hamming**	**Paridad** a nivel bit
Discos de paridad	Varios discos (f0, f1, f2)	Solo **1 disco de paridad**
Lectura/escritura	Muy sincronizada	Sincronizada a nivel bit
Uso actual	Obsoleto	Poco común

RAID 2 necesita múltiples discos de corrección, mientras que RAID 3 usa solo uno con paridad simple.

El **código de Hamming** es un método de detección y **corrección de errores** que permite identificar **y corregir automáticamente** un solo bit incorrecto en los datos.
En RAID 2, se usan varios discos para guardar esos bits de corrección (f0, f1, f2), y así reconstruir cualquier dato si ocurre un error en un solo disco.

Gestión de E/ S – RAID 4 5 y 6

RAID 4 ,5 y 6



- RAID 4 – Paridad distribuida a nivel bloque
- RAID 5 – Paridad distribuida a nivel bloque
- RAID 6 – Redundancia dual

🧱 Diferencias entre **RAID 4, 5 y 6**

Nivel	Paridad	Distribución	Tolerancia a fallos
RAID 4	Paridad en 1 disco	**Centralizada**	Soporta **1 fallo**
RAID 5	Paridad en 1 disco	**Distribuida**	Soporta **1 fallo**
RAID 6	**2 discos de paridad**	**Distribuida**	Soporta **2 fallos**

📌 *RAID 5 mejora el rendimiento de RAID 4 evitando cuellos de botella en el disco de paridad. RAID 6 agrega **redundancia extra**, ideal para alta disponibilidad.*

Resumen – Gestión de Entrada / Salida

- Técnicas:
  - E/S programada
  - Interrupciones
  - DMA
- Buffers:
  - Único
  - Doble
  - Circular
- Planificación
  - FIFO (First in First Out)
  - SSTF (Shortest Seek Time First)
  - SCAN (Algoritmo del Ascensor)
  - C-SCAN (Circular SCAN)
- RAID
  - RAID 0 no redundante
  - RAID 1 espejo
  - RAID 5/6 paridad distribuida


UTN.BA
FACULTAD
REGIONAL
BUENOS AIRES
Muchas gracias