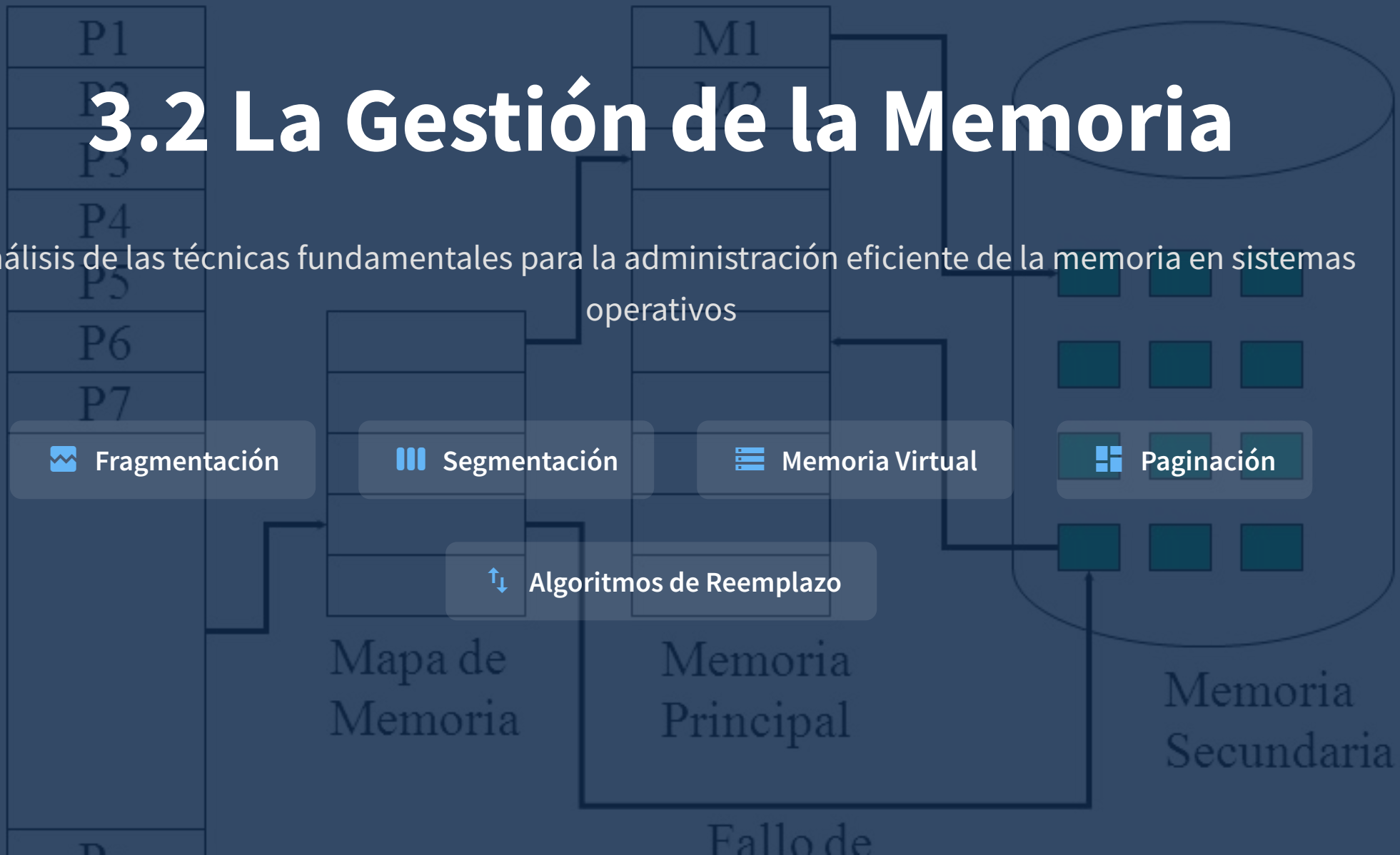


Gestión de Memoria Virtual

3.2 La Gestión de la Memoria

Análisis de las técnicas fundamentales para la administración eficiente de la memoria en sistemas operativos



Índice de Contenidos



3.2.1 Fragmentación

Tipos de fragmentación ([interna](#) y [externa](#)) y criterios de evaluación del gestor de memoria



3.2.2 Segmentación

Esquema de gestión con bloques no contiguos, ventajas, compactación y mecanismos de protección



3.2.3 Memoria Virtual

Técnica para ejecutar programas [más grandes que la memoria física](#), ventajas y archivo de paginación



3.2.4 Paginación

División de procesos en páginas, marcos de página y traducción de direcciones virtuales a físicas



3.2.5 Algoritmos de Reemplazo




Estrategias para gestionar fallos de página: [FIFO](#), [LRU](#) y selección del algoritmo óptimo

3.2.1 Fragmentación




Concepto

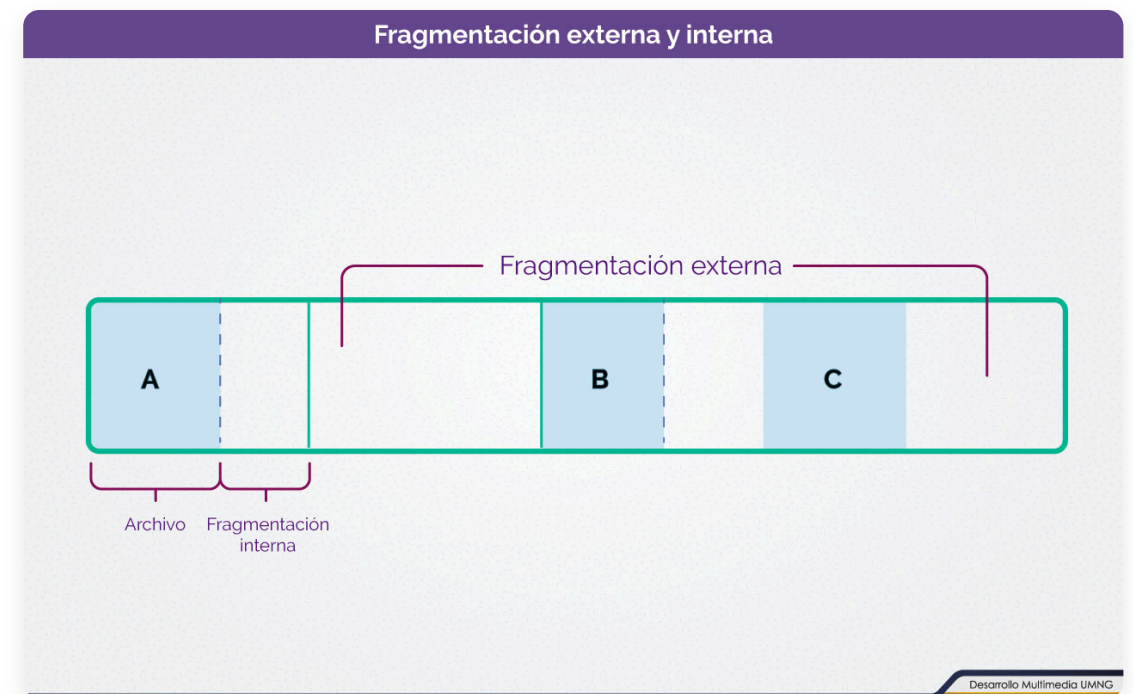
Cantidad de **memoria desaprovechada** por el gestor de memoria.
Principal desafío en la gestión eficiente.

Tipos de fragmentación

-  **Interna**
Asignación de más memoria de la necesaria. Típica en particiones fijas.
-  **Externa**
Espacios libres suficientes pero dispersos, imposibilitando asignación contigua.
-  **Temporal**
Relacionada con gestión dinámica a lo largo del tiempo.

Criterios de evaluación

-  **Memoria desaprovechada:** Cantidad de memoria perdida en asignación
-  **Complejidad en el tiempo:** Tiempo perdido en acceso a memoria
-  **Protección y uso compartido:** Seguridad y control de accesos



3.2.2 Segmentación

i Concepto

Esquema avanzado de gestión basado en **particiones variables**.
Permite que bloques de un proceso estén en áreas **no contiguas** de memoria.

★ Ventajas

III Bloques no contiguos

≡ Compactación de memoria

🛡️ Protección por segmento

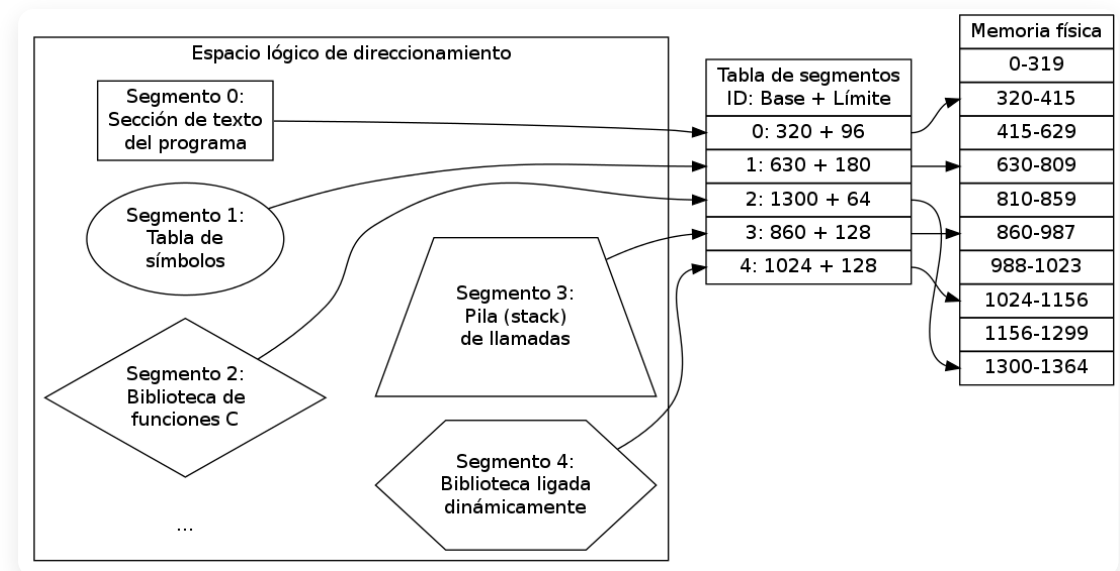
🔗 Uso compartido de memoria

≡ Compactación de memoria

- 1 Identificar bloques libres dispersos
- 2 Mover bloques ocupados para crear espacio contiguo
- 3 Actualizar tablas de segmentación

🛡️ Protección y uso compartido

- ✓ Derechos de acceso por segmento (lectura, escritura, ejecución)
- 👥 Compartición de bibliotecas y memoria entre procesos



3.2.3 Memoria Virtual

i Concepto y definición

Técnica desarrollada en 1961 por Fotheringham. Permite que **el programa, datos y pila combinados sean mayores que la memoria física disponible**. Solo se cargan en memoria las partes de uso corriente.

★ Ventajas

↕ **Programas más grandes** que la memoria física

■ **Más programas** cargados simultáneamente

🔄 **Inicio más rápido** de los programas

↔ **Menor frecuencia** de intercambio de procesos

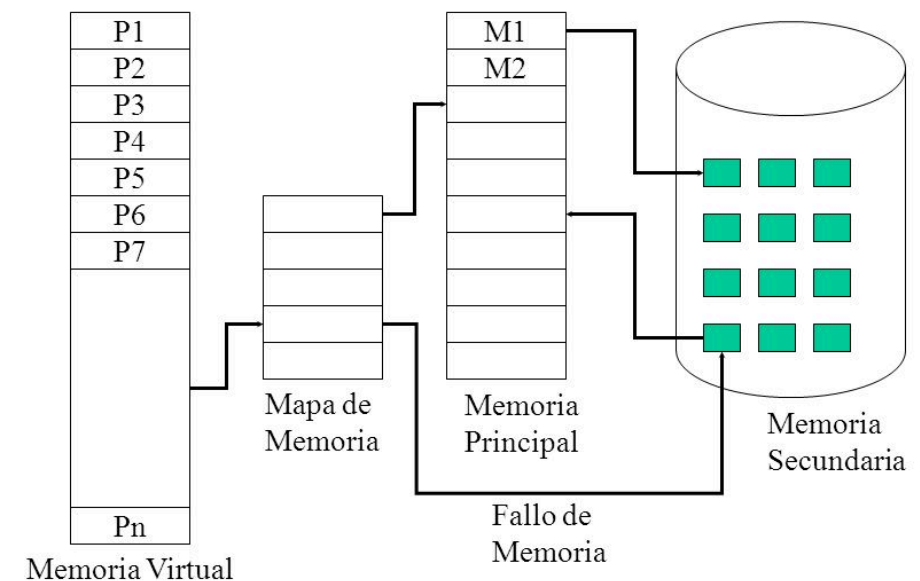
⚙️ Funcionamiento y gestión

- 1 Sistema operativo decide qué partes cargar en memoria
- 2 Determina cuándo cargarlas y dónde ubicarlas
- 3 Un proceso puede usar CPU mientras otro realiza swapping

📄 Archivo de paginación

Área en disco duro utilizada como si fuese RAM. Permite ejecutar más programas de los que cabrían en memoria física.

Gestión de Memoria Virtual





3.2.4 Paginación


Concepto


Técnica que **divide el proceso en páginas** (bloques de datos) del mismo tamaño. El sistema operativo divide la memoria en **marcos de página** y carga únicamente las páginas que se van a utilizar.

Marcos y bloques de datos


 **Marcos de página**
Bloques de memoria física de tamaño fijo


 **Páginas**
Bloques de datos del proceso del mismo tamaño

 **Carga no ordenada**
Las páginas pueden cargarse en marcos no consecutivos

 **Fallo de página**
Ocurre al acceder a una página no cargada en memoria

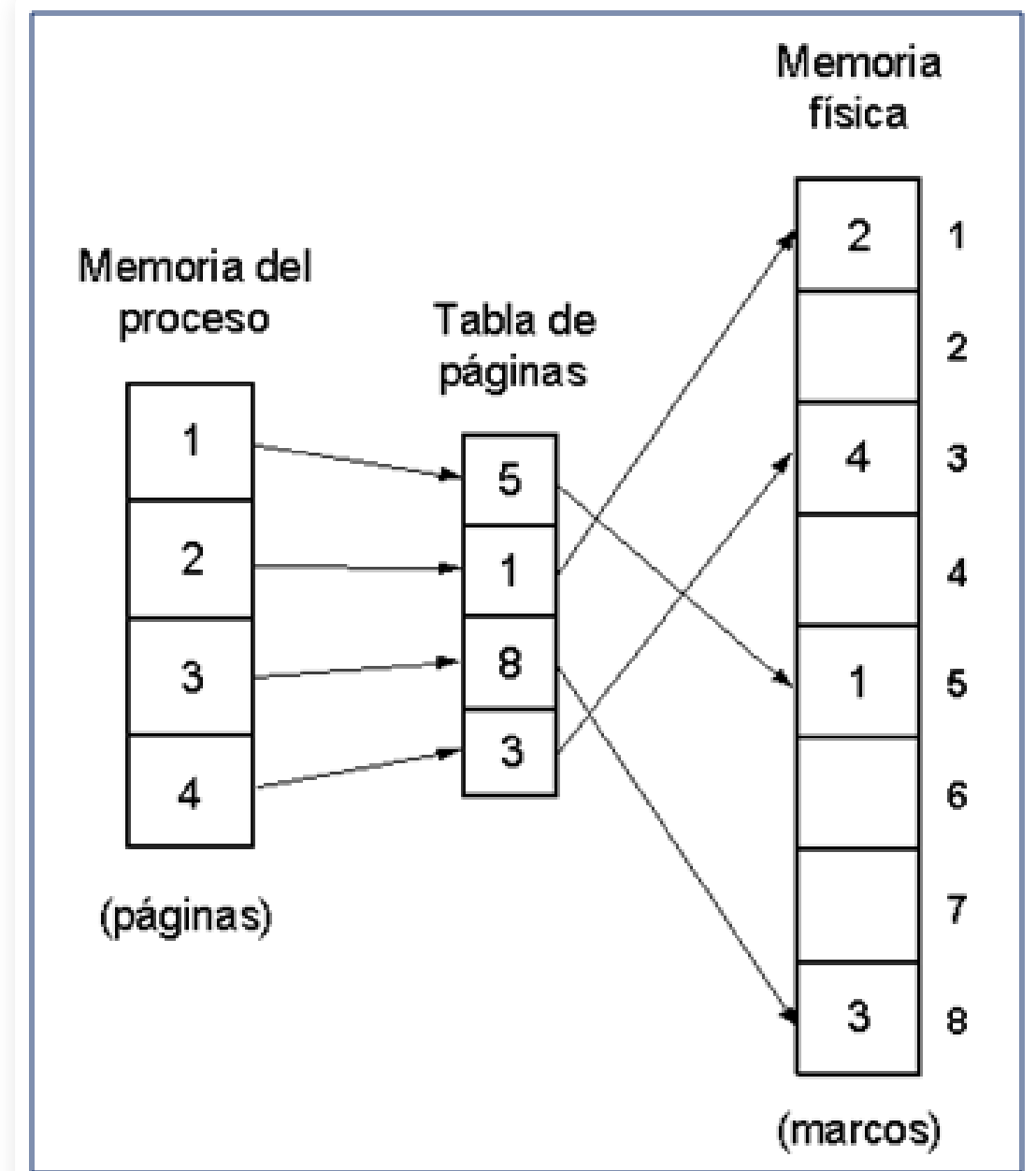
Dirección virtual vs. física

 **Virtual**
Espacio de direcciones continuo visto por el proceso

 **Física**
Ubicación real en memoria RAM, puede ser no contigua

Funcionamiento detallado

- 1 Verificar si la página está en memoria mediante tabla de páginas
- 2 Si está presente, traducir dirección virtual a física
- 3 Si no está presente (fallo de página), cargar desde disco



3.2.5 Algoritmos de Reemplazo de Páginas

↔ Algoritmos principales



FIFO (First Input First Output)

Elige la página que **lleva más tiempo en memoria**. Simple de implementar pero puede presentar anomalía de Belady.



LRU (Least Recently Used)

Elige la página **usada menos recientemente**. Basado en el principio de localidad temporal. Más eficiente pero más complejo.

🔍 Comparación de algoritmos



FIFO: Simple pero ineficiente en ciertos casos



LRU: Mejor rendimiento pero mayor complejidad



Óptimo: Teóricamente el mejor, no implementable



Clock: Variante aproximada de LRU, balanceada

⚙ Selección del algoritmo óptimo

🔗 **Complejidad de implementación:** Equilibrio entre eficiencia y simplicidad

🔧 **Recursos disponibles:** Hardware especializado para algoritmos complejos

📈 **Patrones de acceso:** Comportamiento típico de los programas

	FIFO con 3 páginas		FIFO con 4 páginas	
Página referenciada	Páginas residentes	Falla de página	Páginas residentes	Falla de página
A	A	F	A	F
B	A B	F	A B	F
C	A B C	F	A B C	F
D	B C D	F	A B C D	F
A	C D A	F	A B C D	
B	D A B	F	A B C D	
E	A B E	F	B C D E	F
A	A B E		C D E A	F
B	A B E		D E A B	F
C	B E C	F	E A B C	F
D	E C D	F	A B C D	F
E	E C D		B C D E	F
	(9 fallas)		(10 fallas)	

Comparación de Algoritmos de Reemplazo



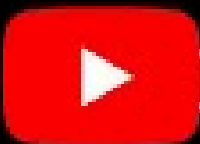
FIFO

- ✓ **Simple implementación** - Fácil de programar
- ✓ **Bajo overhead** - Requiere mínima estructura de datos
- ✗ **Anomalía de Belady** - Más marcos pueden aumentar fallos
- ✗ **No considera uso reciente** - Puede reemplazar páginas activas



LRU

- ✓ **Principio de localidad** - Mejor aprovechamiento de patrones
- ✓ **Menor tasa de fallos** - Generalmente superior a FIFO
- ✗ **Mayor complejidad** - Requiere hardware o software adicional
- ✗ **Mayor overhead** - Necesita actualizar información de uso



FNE Profesor

OTERIAS DE LA COMPUTACION

SISTEMAS OPERATIVOS

ALGORITMO DE REEMPLAZO DE PAGINAS FIFO vs LRU

LRU

LRU		0	1	2	3	4	5	6	7	8
Memoria Principal		0	1	2	3	4	5	6	7	8
Marco 0	0	0	0	0	0	0	0	0	0	0
Marco 1		1	1	1	1	1	1	1	1	1
Marco 2			2	2	2	2	2	2	2	2
Marco 3					3	3	3	3	3	3
	E	E	E	A	E	E	E	E	E	E

FIFO		0	1	2	3	4	5	6	7	8
Memoria Principal		0	1	2	3	4	5	6	7	8
Marco 0	0	0	0	0	0	0	0	0	0	0
Marco 1		1	1	1	1	1	1	1	1	1
Marco 2			2	2	2	2	2	2	2	2
Marco 3					3	3	3	3	3	3
	E	E	E	A	E	E	E	A	E	E

FIFO

💡 Cuándo utilizar cada algoritmo

⚙️ **FIFO**: Sistemas con recursos limitados, implementación simple

📁 **Clock**: Equilibrio entre eficiencia y complejidad

🔗 **LRU**: Sistemas con alto rendimiento, acceso localizado

⚙️ **Híbridos**: Sistemas modernos combinan múltiples estrategias

Ejemplo Práctico: Fallo de Página y Reemplazo

❗ Proceso de fallo de página

1 Detección

CPU intenta acceder a una dirección virtual **no presente** en memoria física

2 Interrupción

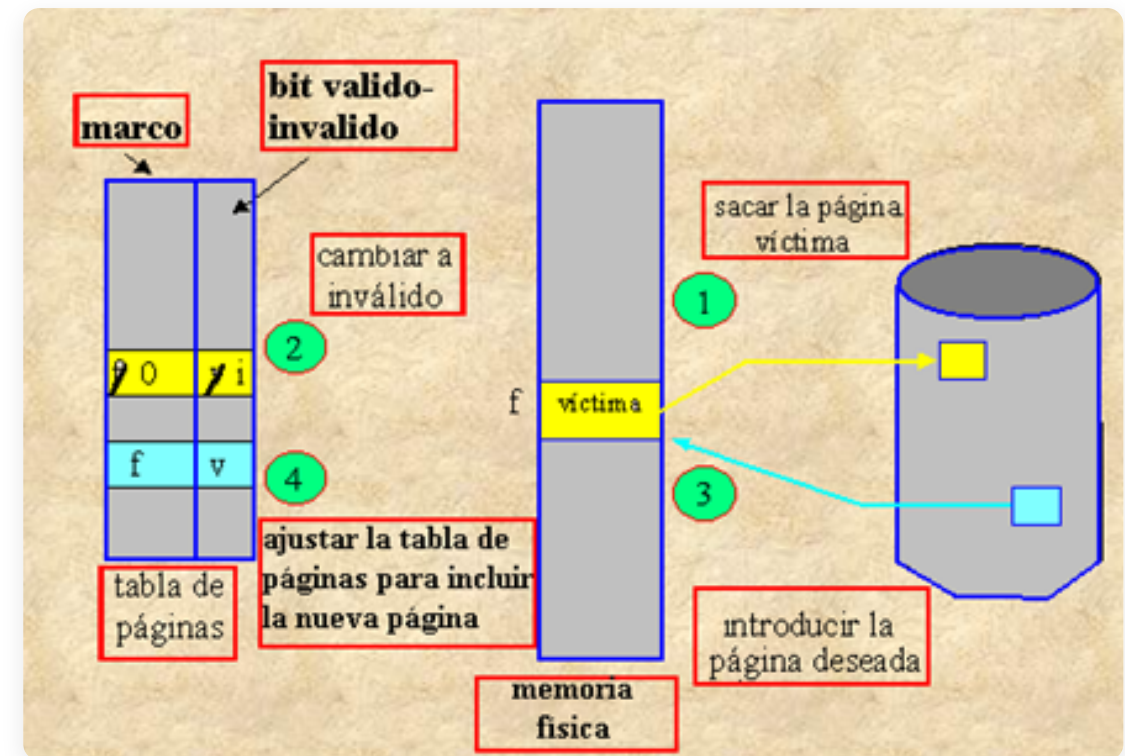
Se genera una **trampa** que transfiere control al sistema operativo

3 Selección de víctima

SO elige una página para reemplazar según el algoritmo activo (FIFO, LRU, etc.)

4 Carga y actualización

Se carga la página solicitada desde disco y se **actualiza la tabla de páginas**



💡 Aspectos clave

- 🕒 **Latencia:** El proceso de manejo de fallo implica acceso a disco, mucho más lento que RAM
- 🔄 **Swapping:** Si la página víctima fue modificada, debe escribirse en disco antes de reemplazarla
- 🛡️ **Protección:** Durante todo el proceso se mantienen los mecanismos de protección de memoria

🧠 Impacto en el rendimiento

Un **alto número de fallos de página** (thrashing) degrada significativamente el rendimiento del sistema. La selección adecuada del algoritmo de reemplazo y el tamaño del conjunto de trabajo son cruciales para minimizar este efecto.

Resumen: Gestión de Memoria

Conceptos clave



Fragmentación

Interna y **externa**: memoria desaprovechada que afecta la eficiencia del sistema



Segmentación

Permite **bloques no contiguos** con protección y uso compartido



Memoria Virtual

Ejecutar programas **más grandes** que la memoria física disponible



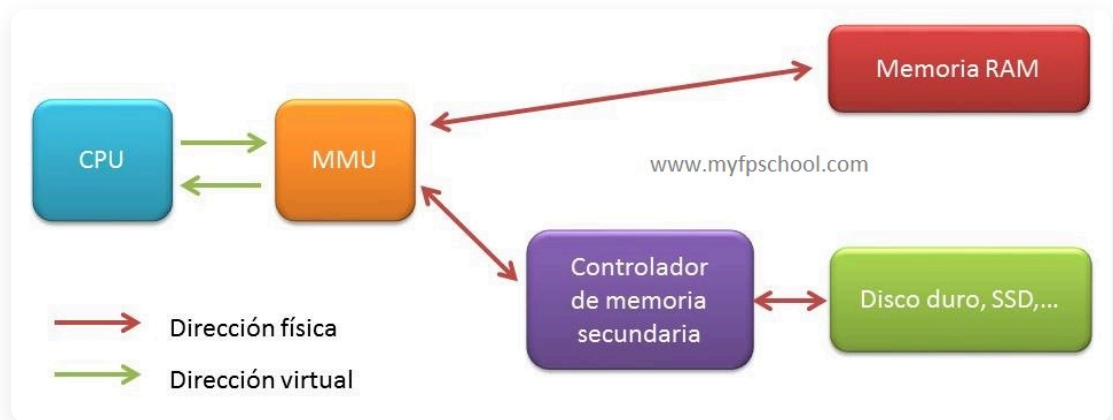
Paginación

División en **páginas** y **marcos** con traducción de direcciones



Algoritmos de Reemplazo

FIFO y **LRU** para gestionar fallos de página eficientemente



! Importancia en sistemas modernos



Rendimiento: Optimiza el uso de recursos



Protección: Aísla procesos entre sí



Multiprogramación: Permite ejecución concurrente



Eficiencia: Minimiza desperdicio de memoria