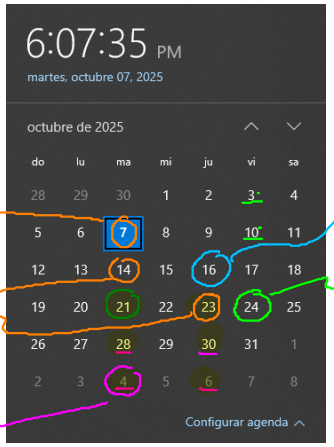
DZ 07/10/2025 - Sistemas Operativos (Udea)

0. Proximos eventos

- Parcial 1 (Modulos 1 y 2)
  - Oct 28
  - Oct 30
  - Nov 4 (9/14)
  - Nov 6



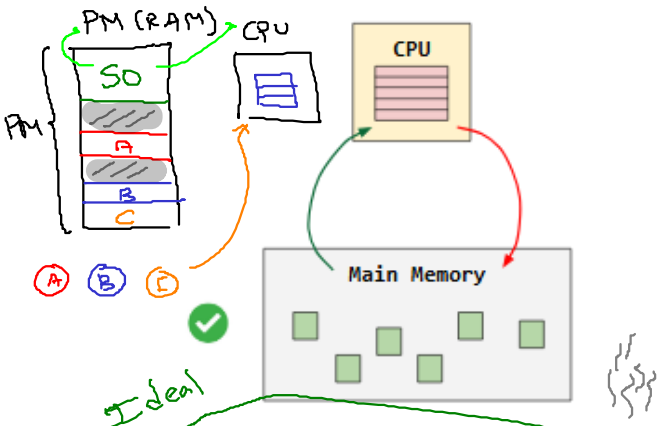
- Bonus: (x2)
  - 1. Taller V. CPU (0.4)
  - 2. Taller V. Mem (0.4) *
  - 3. Pasteles (0.4) *

https://github.com/udea-so/pasteles_SO

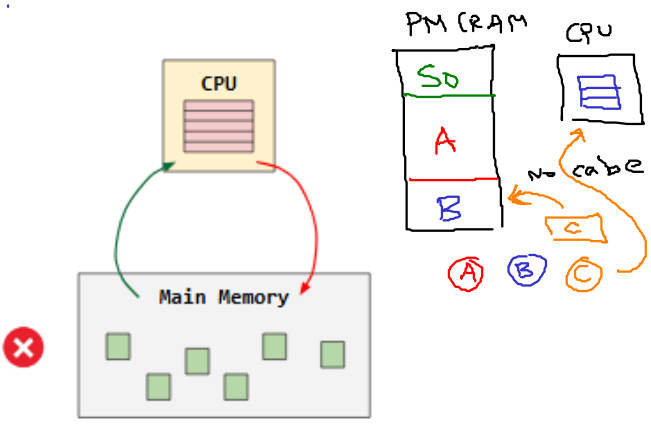
https://www.redhat.com/en/services/training/red-hat-academy

1. Contextualizacion

Caso Feliz



Realidad



Ideal

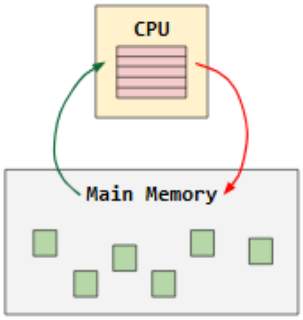
$size(\sum AS(P_i)) < size(PM)$

$\uparrow size(AS(A)+AS(B)+AS(C)) < size(PM)$ — 16GB

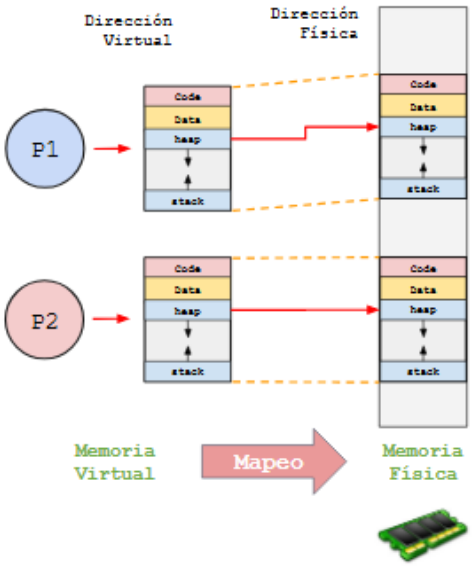
Realidad: Los recursos Fisicos son Finitos

En resumen

1. El espacio de direcciones de un proceso es pequeño y cabe en memoria física.
2. El espacio de direcciones debido al workload (todos los procesos en ejecución) cabe en memoria.



No tenemos suficiente Mem. Fisica

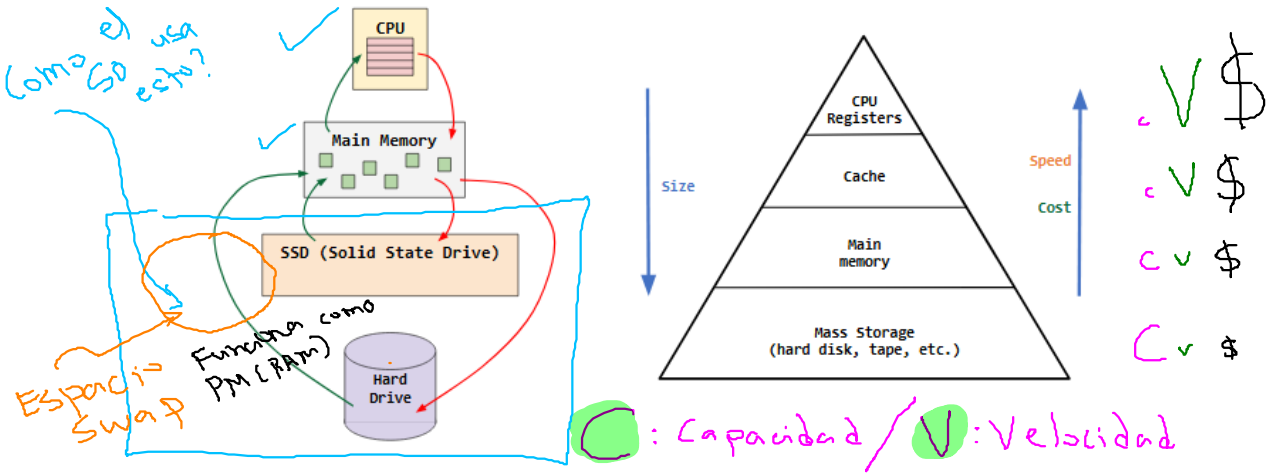


Pregunta clave

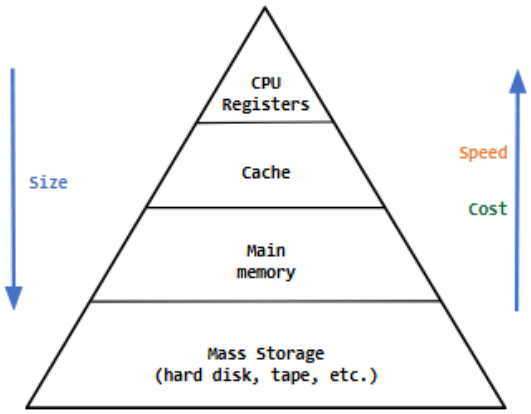
¿Cómo ir más allá de la memoria física?

¿Cómo puede el sistema operativo usar un dispositivo más grande y lento para proporcionar de forma transparente la ilusión de un gran espacio de direcciones virtuales?

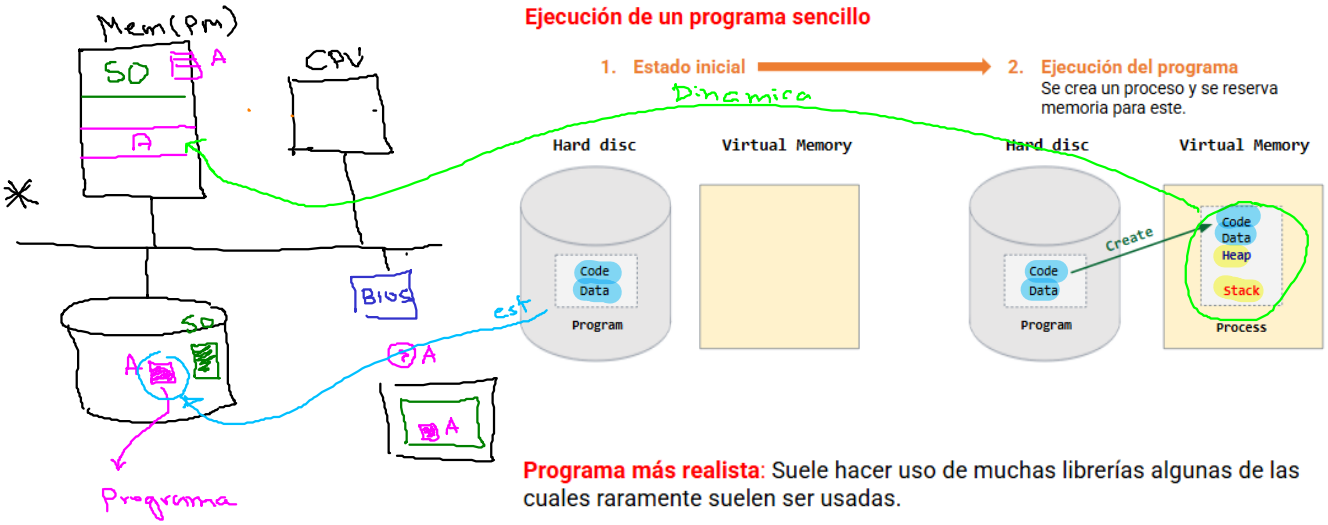
2. Jerarquía de Memoria



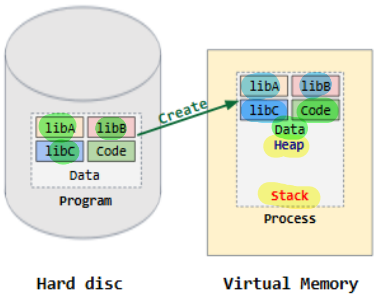
Cada capa actúa como un **espacio de almacenamiento de respaldo (backing store)** de la capa superior



3. Procesos y Memoria



Programa más realista: Suele hacer uso de muchas librerías algunas de las cuales raramente suelen ser usadas.



Escenario

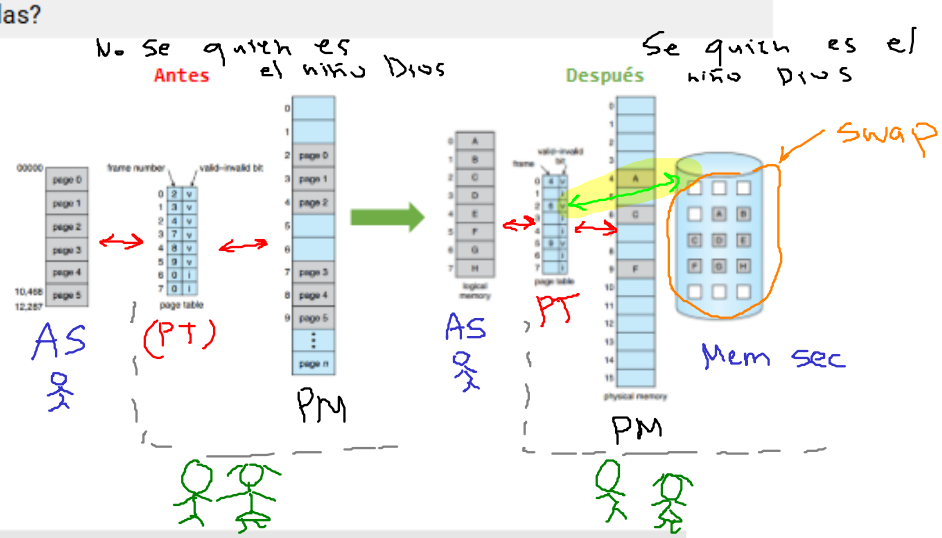
- El programa usa varias librerías que son referenciadas en memoria virtual.
- De acuerdo a lo visto hasta ahora, las páginas virtuales tienen que estar mapeadas en memoria física a través de la **tabla de página (PT)**.

Problema: Gasto de memoria

Escenario

¿Cómo evitar el desperdicio de páginas físicas (frames) empleados para respaldar páginas virtuales (pages) poco usadas?

- Se requiere un espacio (usualmente en el disco duro) en la jerarquía de memoria.
- Enviar a otro nivel porciones de espacio de direccionamiento que **no son muy demandadas**.



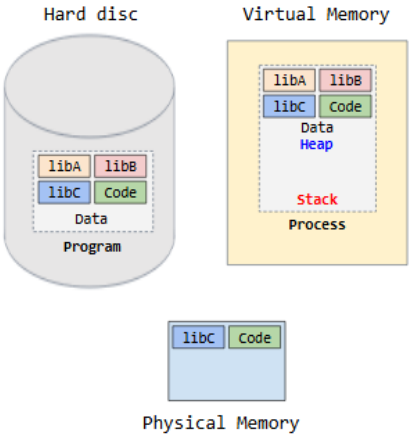
4. Swapping

¿Cómo ir más allá de la memoria física?

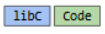
¿Cómo puede el sistema operativo usar un dispositivo más grande y lento para proporcionar de forma transparente la ilusión de un gran espacio de direcciones virtuales?

Mejora: Swapping

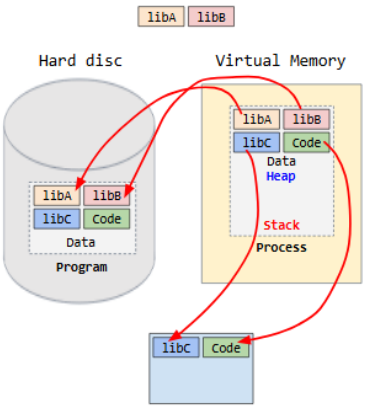
1. **Ejecución del programa**
   Se reserva la memoria virtual del proceso creado.



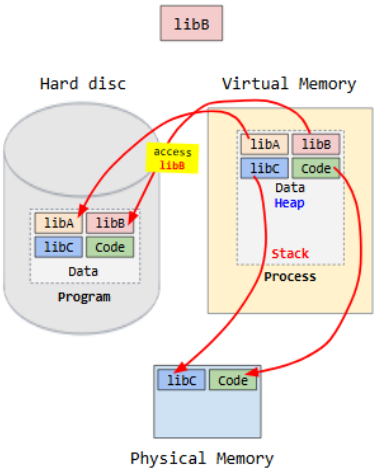
2. **Se crean las referencias**
   - Memoria virtual → Memoria física



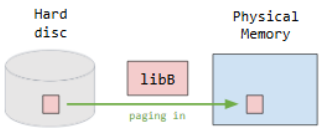
   - Memoria virtual → Disco duro



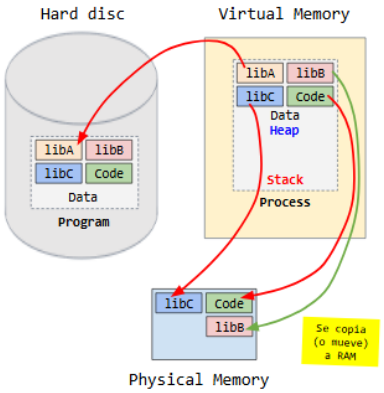
3. **Se accede a la librería B**



4. **Se copia (o mueve) la página a la memoria principal**

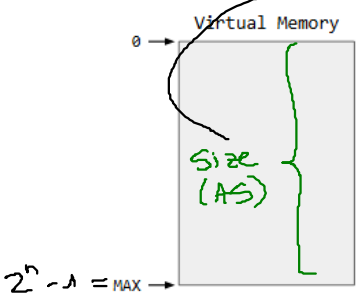


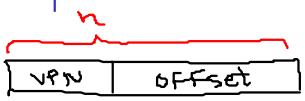
La referencia para `libB` se actualiza.



5. Problema de los grandes espacios de direccionamiento

Escenario: Veamos un caso real:





Bits de Mem. virtual	Tamaño de la Mem. virtual
32	4*(2^30) = 4 GB
64	16*(2^60) = 16 EB (**Valor muy grande**)

Conclusiones:

- El espacio de memoria virtual es muy grande, pudiendo incluso superar el tamaño de la memoria física (real). size (PM) = RAM
- Bajo estas condiciones, puede que ni siquiera quepa un proceso (y mucho menos varios).

$size(AS) = 2^n \quad [0, 2^{n-1}]$

$n = 32 \longrightarrow size(AS) = ?$

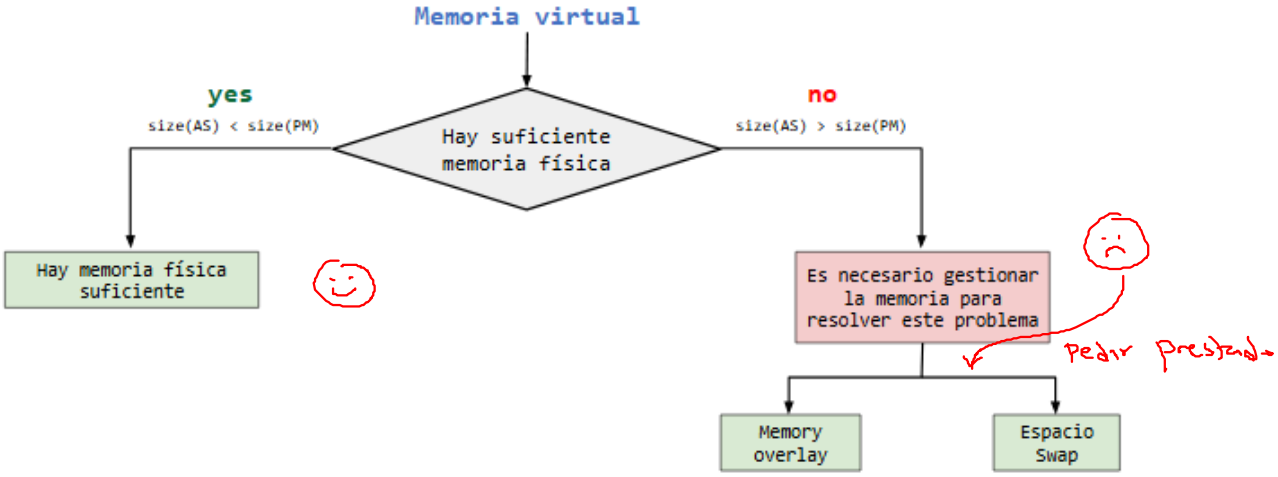
$size(AS) = 2^{32} = 2^2 2^{30} = 4G$ (1GB)

$n = 64 \longrightarrow size(AS) = ?$

$size(AS) = 2^{64} = 2^4 2^{60}$ (1E)

$= 16\ ET$

¿Como manejar este problema?

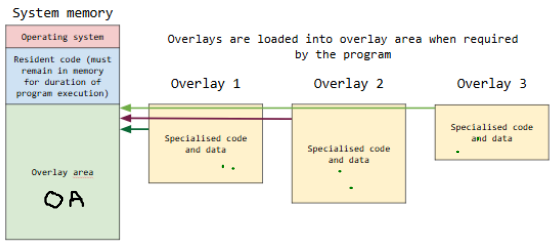


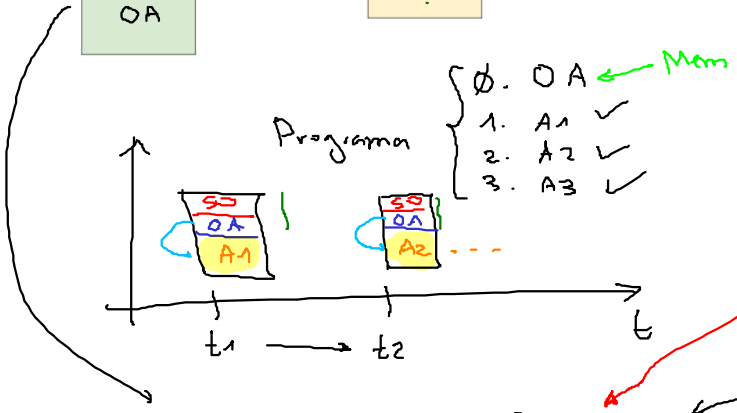
. vs .

6. Memory Overlay (*)

El programador divide el programa en un número de secciones lógicas:

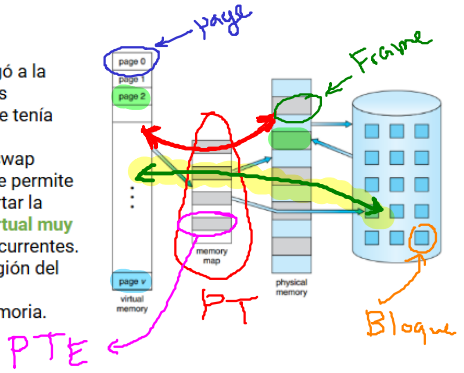
- **Código residente**: Sección lógica que permanece siempre memoria.
- **Overlay area**: Secciones del programa que solo se cargan cuando son requeridas.





Espacio Swap

- Con la llegada de la **multiprogramación**, se llegó a la necesidad de soportar mas memoria fisica de la que se tenía disponible.
- La adición de un espacio swap (espacio de intercambio) le permite al sistema operativo soportar la ilusión de una memoria virtual muy grande para procesos concurrentes.
- El espacio swap es una región del disco duro empleada para almacenar páginas de memoria.



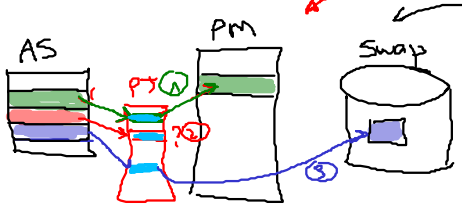
Pregunta:

¿Cómo el sistema Operativo identifica la localización de cada página en el espacio de direcciones?

Respuesta:

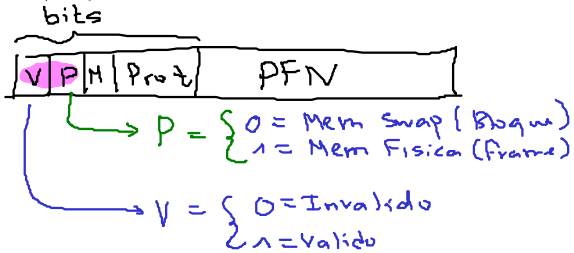
Cada página del espacio de direcciones tiene tres posibilidades de mapeo:

1. Memoria física (V = 1, P = 1)
2. Memoria secundaria (V = 1, P = 0)
3. No se mapea - free (V = 0)

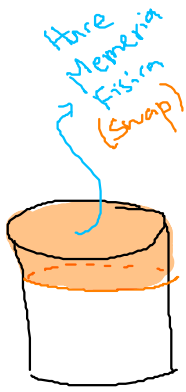


① V=1, P=1
② V=0, P=X
③ V=1, P=0

PTE
bits+PFN



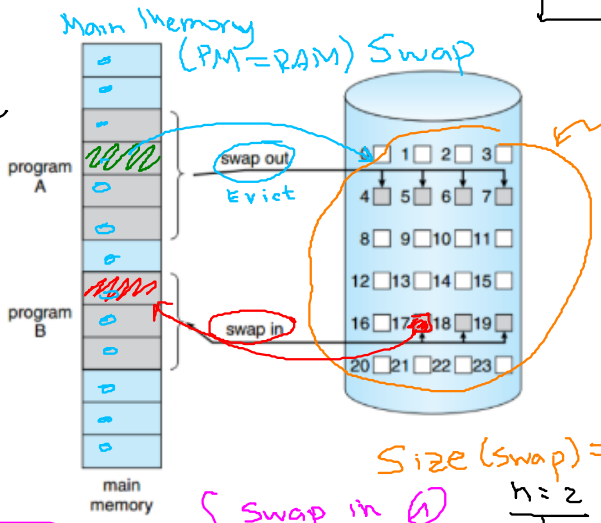
6. Espacio Swap



Introducción

Espacio reservado en el disco duro para almacenar páginas:

- El SO divide el espacio swap en páginas.
- El tamaño del espacio swap determina el número de páginas que el sistema puede utilizar en un momento dado.
- Operaciones de transferencia entre páginas:
  - Swap out - Page out
  - Swap in - Page in



Size(swap) = n * size(PM)

n = 2

↳ size(PM) = 4GB

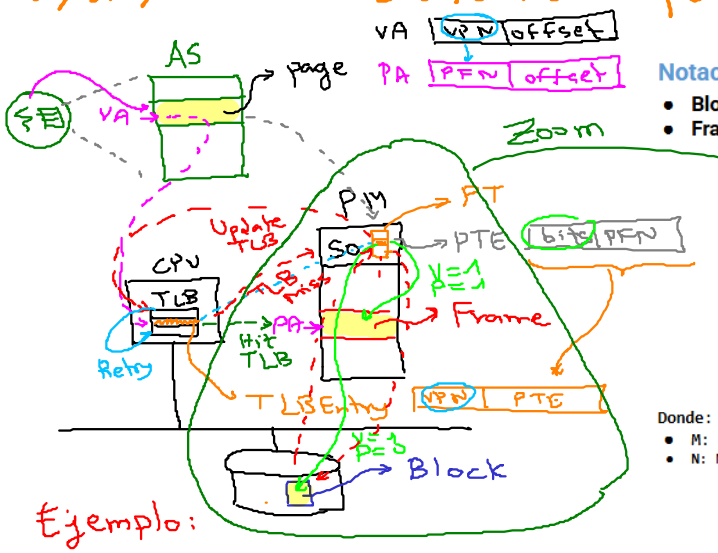
↓

size(swap) = 8GB

Next = Como se hace el intercambio

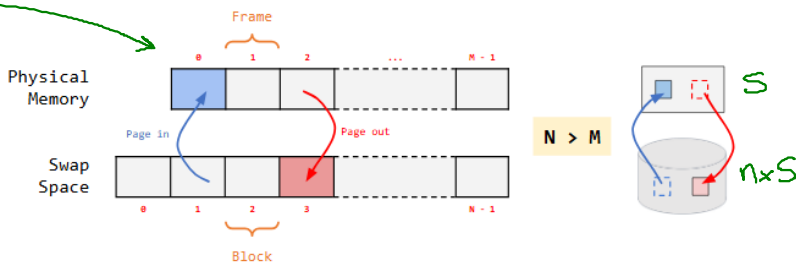
- Swap in (1)
- Swap out (2)

09/10/2025 - Sistemas Operativos (Vde@)



Notación

- **Bloque (block)**: Espacio de memoria (página) en espacio swap.
- **Frame (frame)**: Espacio de memoria (página) en memoria física.



Donde:

- M: Número de frames.
- N: Número de bloques.

Ejemplo:

- Memoria física de 4 páginas (frames).
- Espacio swap de 8 páginas (blocks).
- Cuatro procesos: P0, P1, P2 y P3.

Las siguientes tablas detallan el mapeo ~~para cada una de las entradas~~ de la tabla de página cuyo bit de validez (V) es 1.

P0 (Proc 0)

VPN	PFN	Block
VNP0	PFN0	---
VNP1	---	Block0
VNP2	---	Block1
...	...	...

P1 (Proc 1)

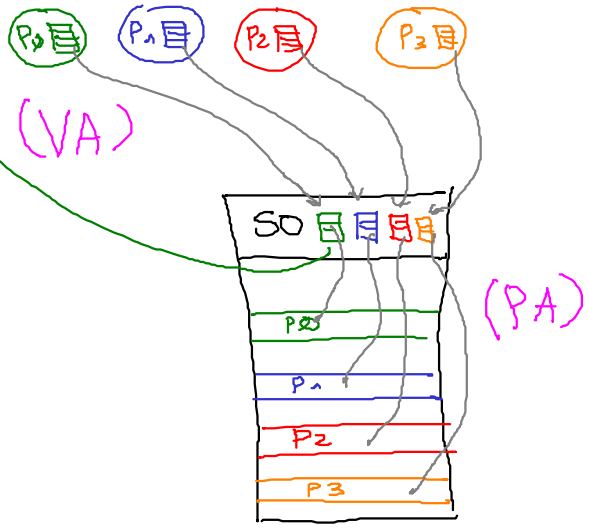
VPN	PFN	Block
VNP0	---	Block3
VNP1	---	Block4
VNP2	PFN1	---
VPN3	PFN2	---

P2 (Proc 2)

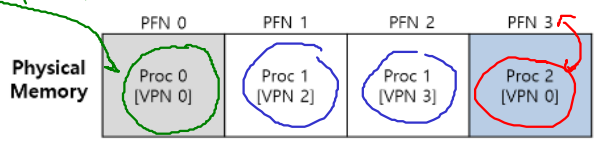
VPN	PFN	Block
VNP0	PFN3	---
VNP1	---	Block6
...	...	...

P3 (Proc 3)

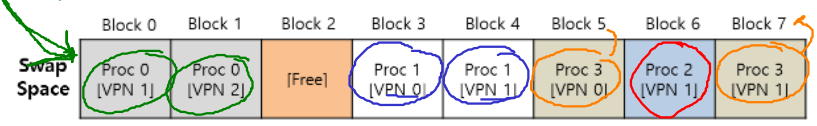
VPN	PFN	Block
VNP0	---	Block5
VNP1	---	Block7
...	...	...



V=1 P=1



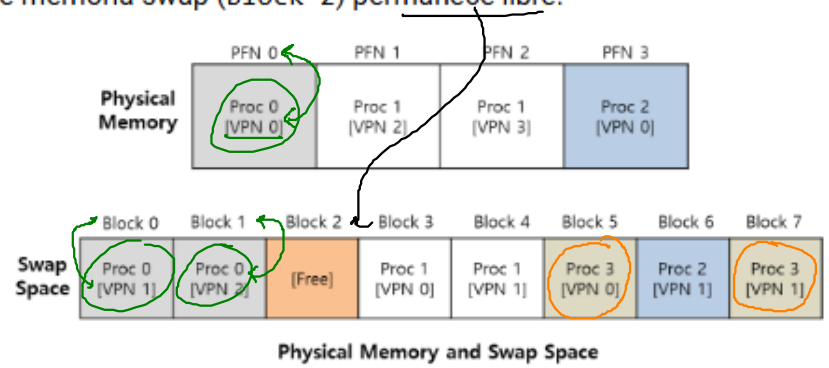
V=1 P=0



Physical Memory and Swap Space

Observaciones importantes:

1. P0, P1, P2 y P3 están compartiendo activamente memoria.
2. De las páginas válidas de estos procesos, sólo algunas están en memoria; el resto se encuentra en espacio swap en el disco.
3. Todas las páginas del proceso P3 han sido (**swap out**) sacadas de memoria física y llevadas al disco (Por lo que este proceso no se encuentra actualmente en ejecución).
4. Un bloque de memoria swap (`Block 2`) permanece libre.

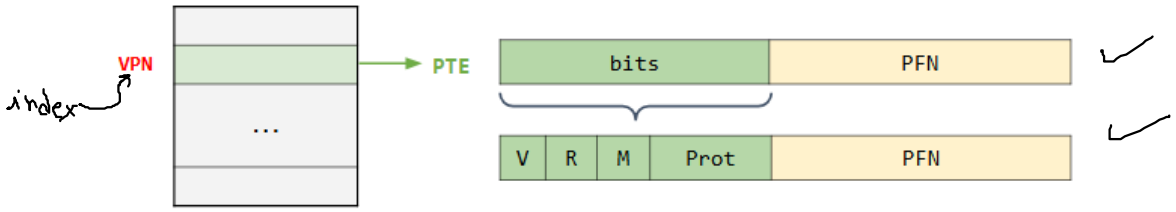


Physical Memory and Swap Space

7. Volviendo a los Bits de la PTE.

VA → (mapeo) → PA

Page Table (PT)



Estructura de una PTE (Page Table Entry)

- **Bit de validez (V):** Indica si la traducción es válida.
- **Bits de protección (Prot):** read - write - execute.
- **Bit de presencia (P):** Indica si la página está en la mem. física.
- **Bit sucio (M):** Indica si la página ha sido modificada.
- **Bit de referencia (R):** Indica si la página ha sido accedida.

V = { 1 ; OK / 0 ; ~~OK~~ }

P = { 1 ; Phys Mem / 0 ; Swap }

V	R	P	M	Prot	PFN

Bit present

- Se requiere un mecanismo para soportar el **intercambio de páginas** desde (page in) y hacía (page out) el disco duro.
- Cuando el hardware analiza el **PTE**, puede suceder que la pagina requerida no este presente en memoria (a pesar de ser una acceso legal).
- El bit de presencia (P) permite conocer dónde se encuentra la página

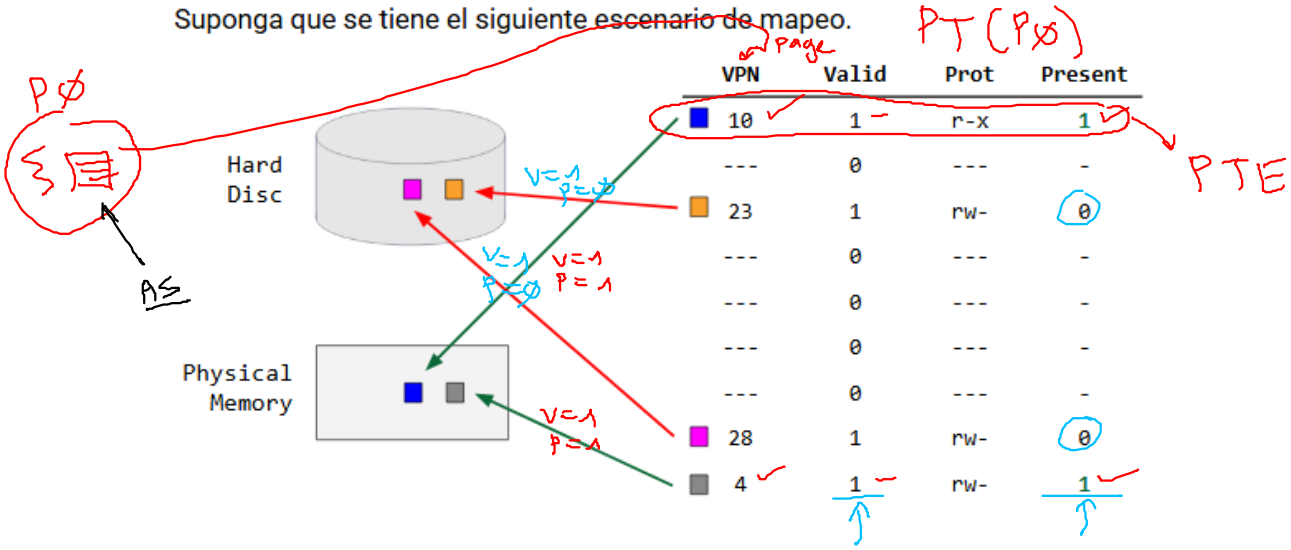
V	R	P	M	Prot	PFN

V	P	Significado
1	1	La página está en la memoria física.
1	0	La página se encuentra en el espacio de intercambio

Valor	Significado
1	La página requerida está presente en la memoria física.
0	La página requerida no está presente en la memoria física, pero sí en el disco

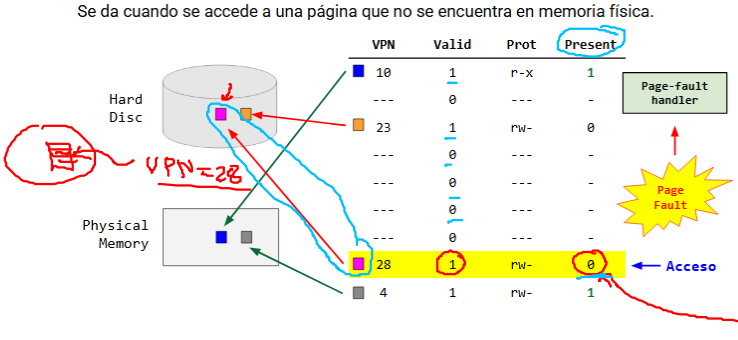
Ejemplo

Suponga que se tiene el siguiente escenario de mapeo.

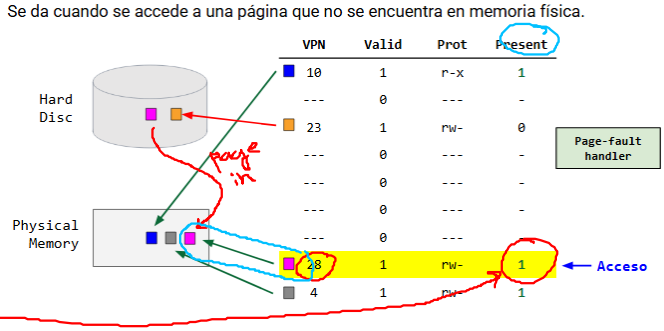


① Caso de fallo de página (hay espacio en memoria física)

Se da cuando se accede a una página que no se encuentra en memoria física.

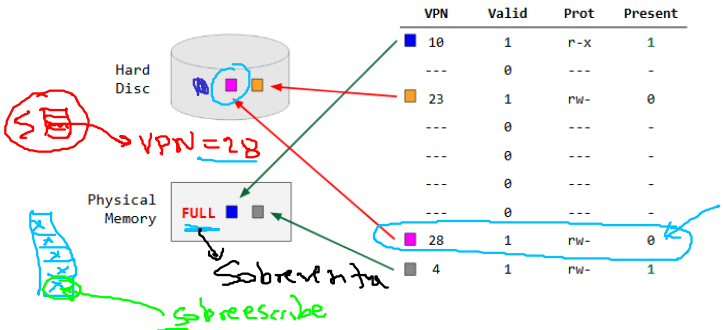


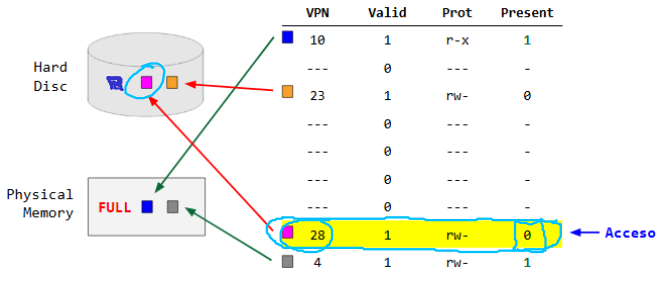
Se da cuando se accede a una página que no se encuentra en memoria física.

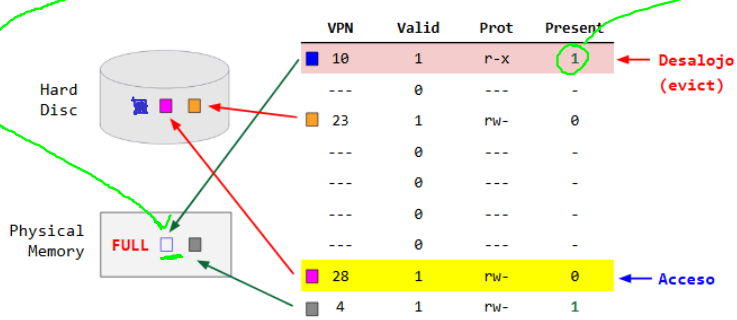


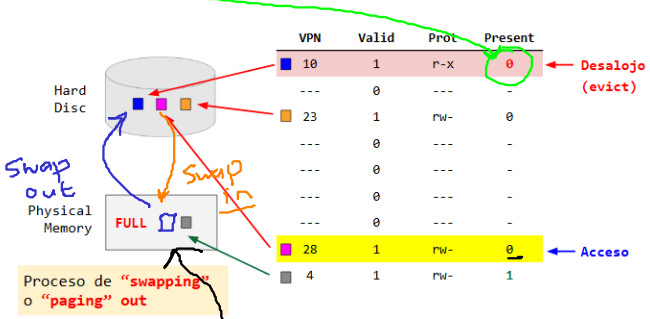
① Caso de fallo de página (Memoria Física FULL)

- El sistema operativo retira las páginas de memoria física (**page out**) para hacer espacio para las nuevas páginas de memoria requerida usando una **page-replacement policy**.

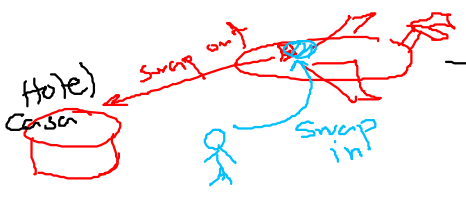


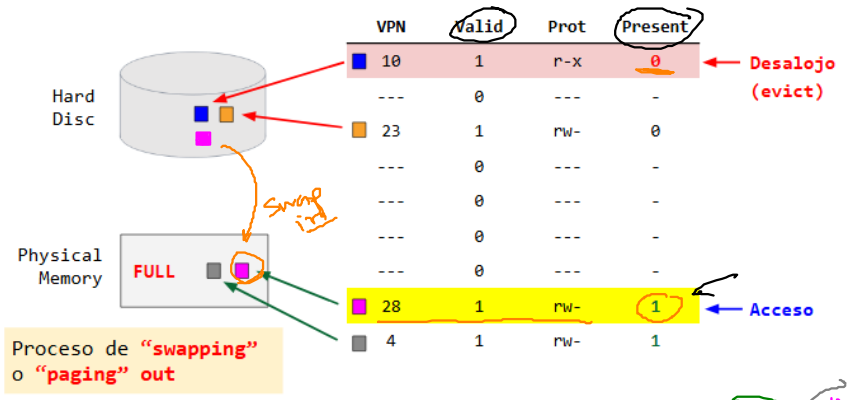




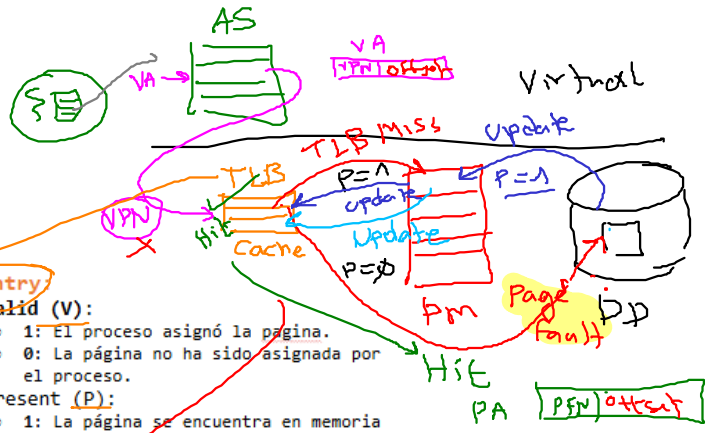


Sobreventa





8. Traducción de direcciones



Rol de los bits V (Valid) y P (Present)

TLB Entry: VPN+PTE

- **Valid (V):** Dice si la entrada es una traducción válida o no.

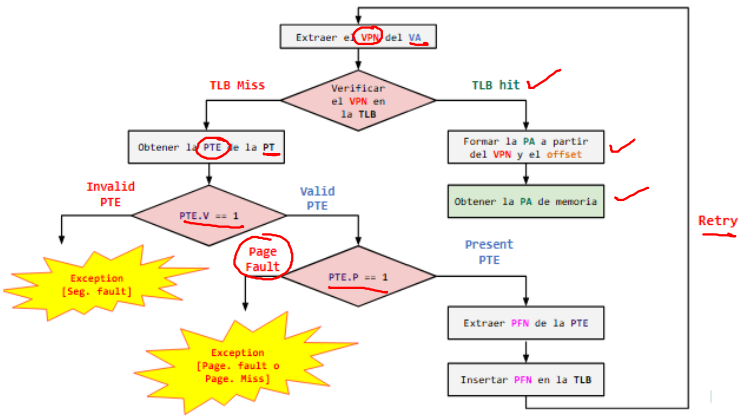
VPN	PFN	Valid	Prot	ASID

PT Entry:

- **Valid (V):**
  - **1:** El proceso asignó la página.
  - **0:** La página no ha sido asignada por el proceso.
- **Present (P):**
  - **1:** La página se encuentra en memoria física.
  - **0:** La página se encuentra en el disco

PFN	Valid	Prot	Present

Algoritmo




```
1  VPN = (VirtualAddress & VPN_MASK) >> SHIFT
2  (Success, TlbEntry) = TLB_Lookup(VPN)
3  if (Success == True) // TLB Hit
4    if (CanAccess(TlbEntry.ProtectBits) == True)
5      Offset = VirtualAddress & OFFSET_MASK
6      PhysAddr = (TlbEntry.PFN << SHIFT) | Offset
7      Register = AccessMemory(PhysAddr)
8    else RaiseException(PROTECTION_FAULT)
9      else // TLB Miss
10       PTEAddr = PTBR + (VPN * sizeof(PTE))
11       PTE = AccessMemory(PTEAddr)
12       if (PTE.Valid == False)
13         RaiseException(SEGMENTATION_FAULT)
14       else
15         if (CanAccess(PTE.ProtectBits) == False)
16           RaiseException(PROTECTION_FAULT)
17         else if (PTE.Present == True)
18           // assuming hardware-managed TLB
19           TLB_Insert(VPN, PTE.PFN, PTE.ProtectBits)
20           RetryInstruction()
21         else if (PTE.Present == False)
22           RaiseException(PAGE_FAULT)
```

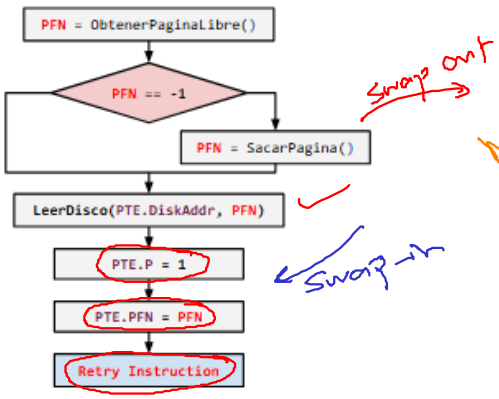
Page-Fault Control Flow Algorithm (Hardware)

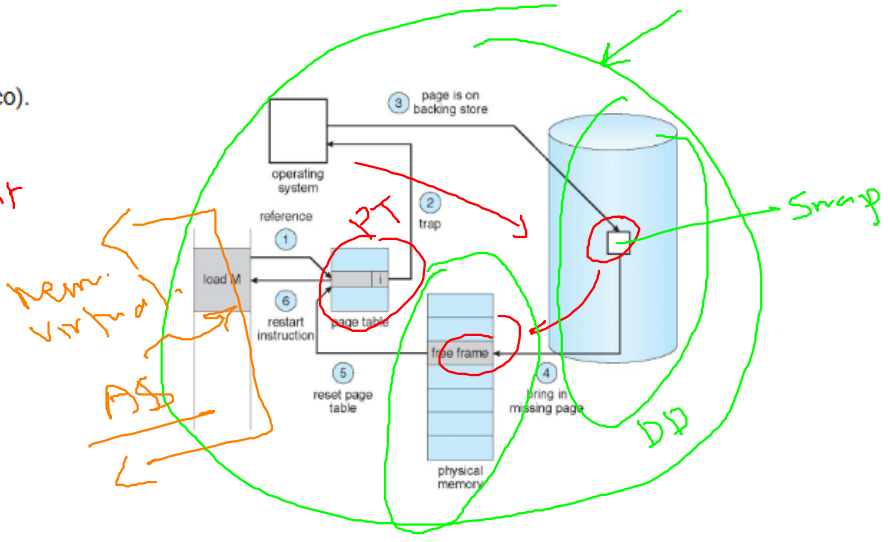
* Fallo de pagina.

Fallo de página

Cuando ocurre un fallo de página:

- El SO accede a la PTE.
- Obtiene la dirección de la página (en el disco).
- Lanza el requerimiento al disco





Cuando hay un fallo de página

- El sistema operativo debe encontrar el frame (dentro de la memoria física) que será asociado a la página que generó el fallo.
- Si no se encuentra la página en memoria física (frame), es necesario esperar a que se ejecute el algoritmo de reemplazo con el fin de liberar espacio en memoria física sacando algunas páginas de esta.

Page fault Handler

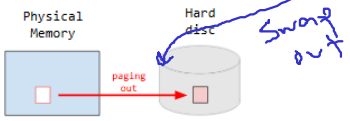
```
1  PFN = FindFreePhysicalPage()
2  if (PFN == -1)                  // no free page found
3    PFN = EvictPage()             // run replacement algorithm
4  DiskRead(PTE.DiskAddr, pfn)     // sleep (waiting for I/O)
5  PTE.present = True              // update page table with present
6  PTE.PFN = PFN                   // bit and translation (PFN)
7  RetryInstruction()              // retry instruction
```

Page-Fault Control Flow Algorithm (Software)

¿Cual página sacar? → **Política de reemplazo (page-replacement policy)**

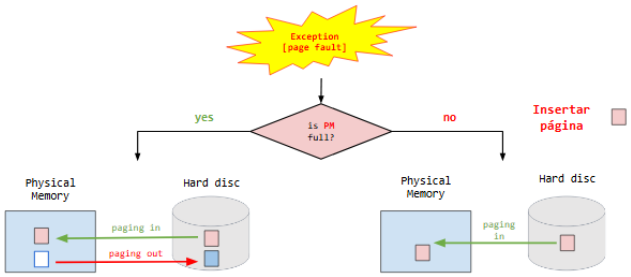
La política de reemplazo es el proceso de selección de una pagina para ser retirada de memoria física.

```
1  PFN = FindFreePhysicalPage()
2  if (PFN == -1)                  // no free page found
3    PFN = EvictPage()             // run replacement algorithm
4  DiskRead(PTE.DiskAddr, pfn)     // sleep (waiting for I/O)
5  PTE.present = True              // update page table with present
6  PTE.PFN = PFN                   // bit and translation (PFN)
7  RetryInstruction()              // retry instruction
```



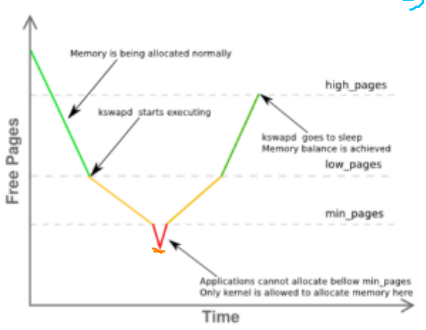
Proceso inicial (el que habíamos dicho al principio)

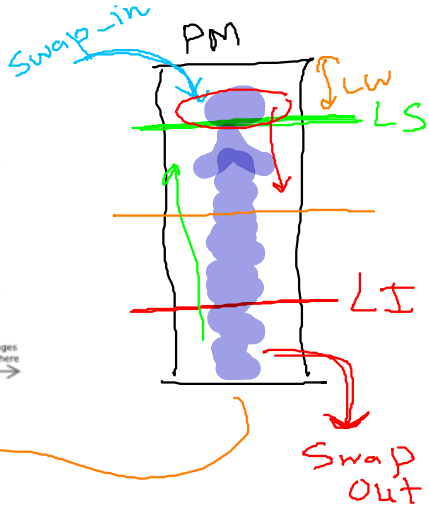
Cuando la memoria está llena, el sistema operativo retira (page out) páginas de memoria (frames) para hacer espacio para nuevas páginas. Sin embargo, esta aproximación **no es realista**.



Proceso real

- El sistema operativo busca mantener una pequeña porción de memoria libre.
- Se manejan dos umbrales:
  - HW (High watermark).
  - LW (Low watermark).
- Un **background thread** llamado **swap daemon (page daemon)** es responsable de liberar memoria.





Reemplazo

Proceso real