# Memoria Virtual

## 95.57/75.03 Organización del computador

**Docentes:** Patricio Moreno y Adeodato Simó

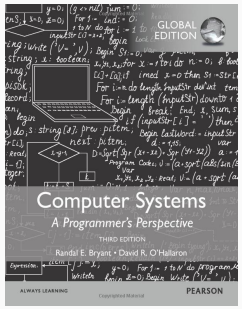
2.$^{do}$ cuatrimestre de 2020

Última modificación: Mon Jul 27 13:01:11 2020 -0300

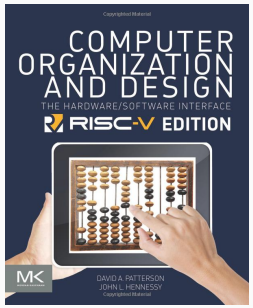
Facultad de Ingeniería (UBA)

# Créditos

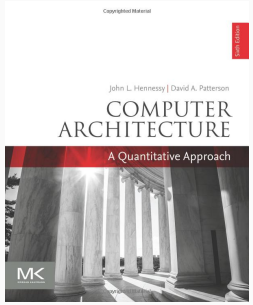
Para armar las presentaciones del curso nos basamos en:




R. E. Bryant and D. R. O'Hallaron, *Computer systems: a programmer's perspective*, Third edition, Global edition. Boston Columbus Hoboken Indianapolis New York San Francisco Cape Town: Pearson, 2015.




D. A. Patterson and J. L. Hennessy, *Computer organization and design: the hardware/software interface*, RISC-V edition. Cambridge, Massachusetts: Morgan Kaufmann Publishers, an imprint of Elsevier, 2017.

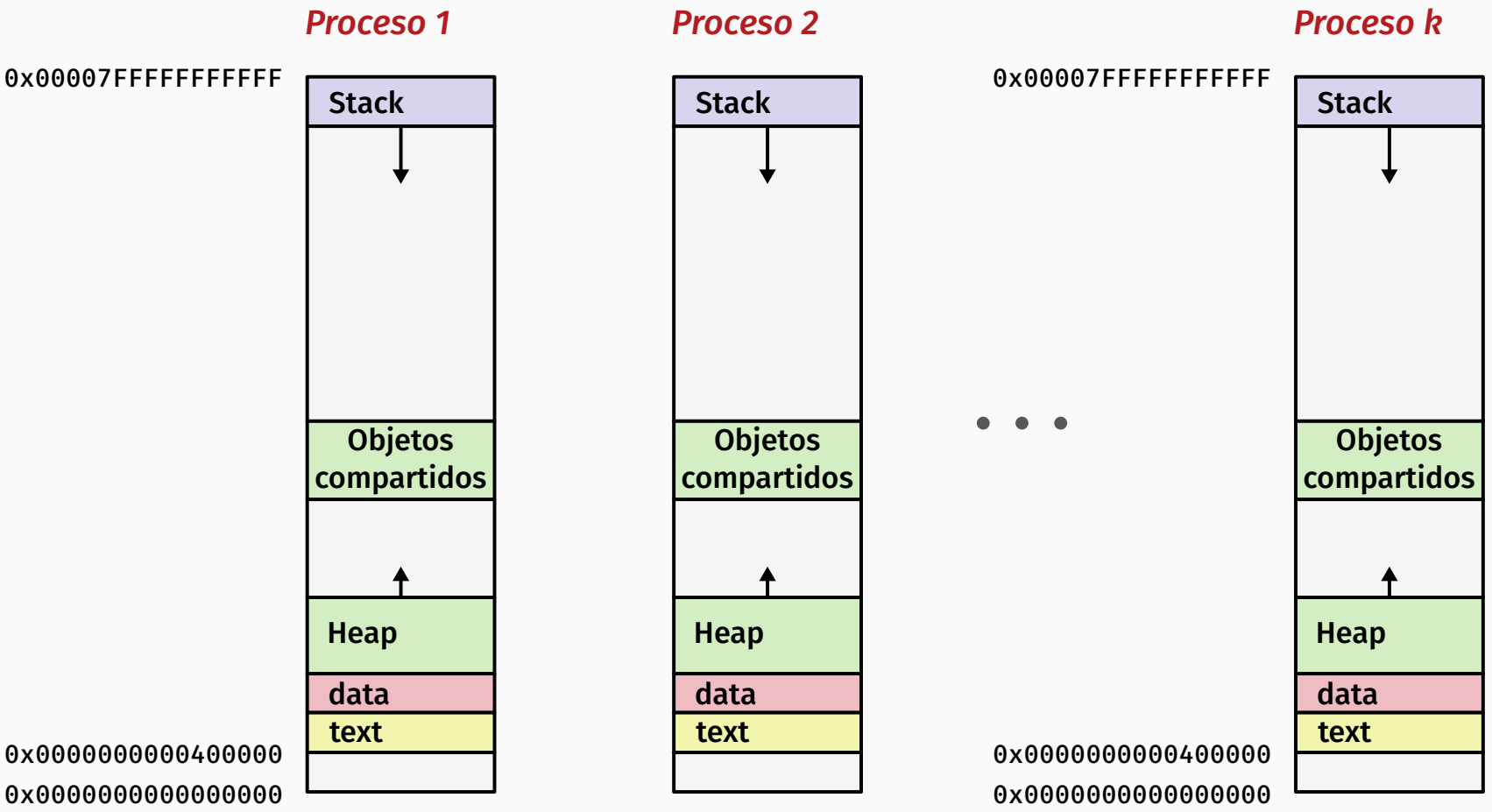



J. L. Hennessy and D. A. Patterson, *Computer architecture: a quantitative approach*. 2017.

# Tabla de contenidos

# ¿Cómo funciona la memoria?

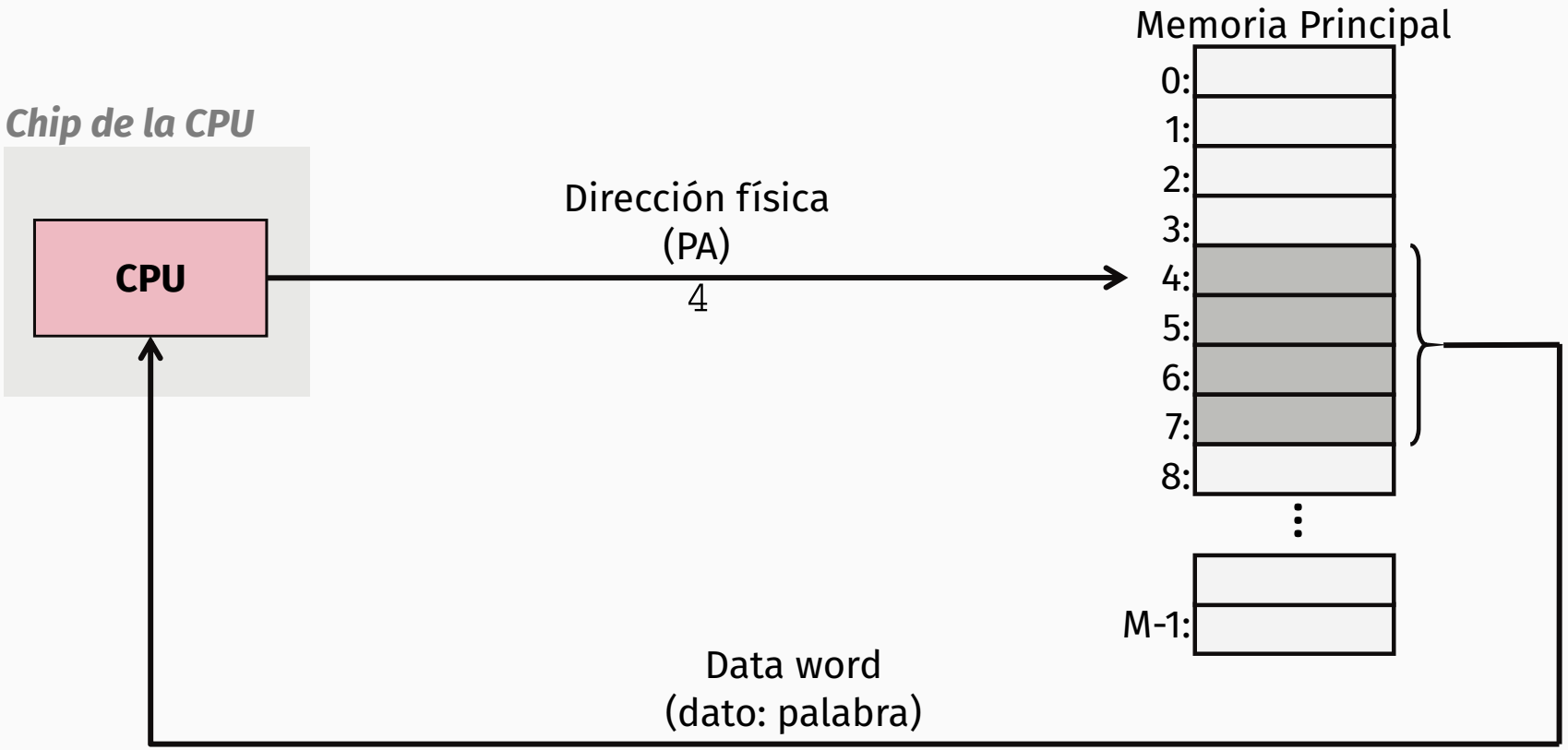
**¿por qué todos los procesos usan las mismas direcciones de memoria? (los mismos índices en nuestro arreglo de memoria)**




**Por el uso de memoria *¡virtual!***

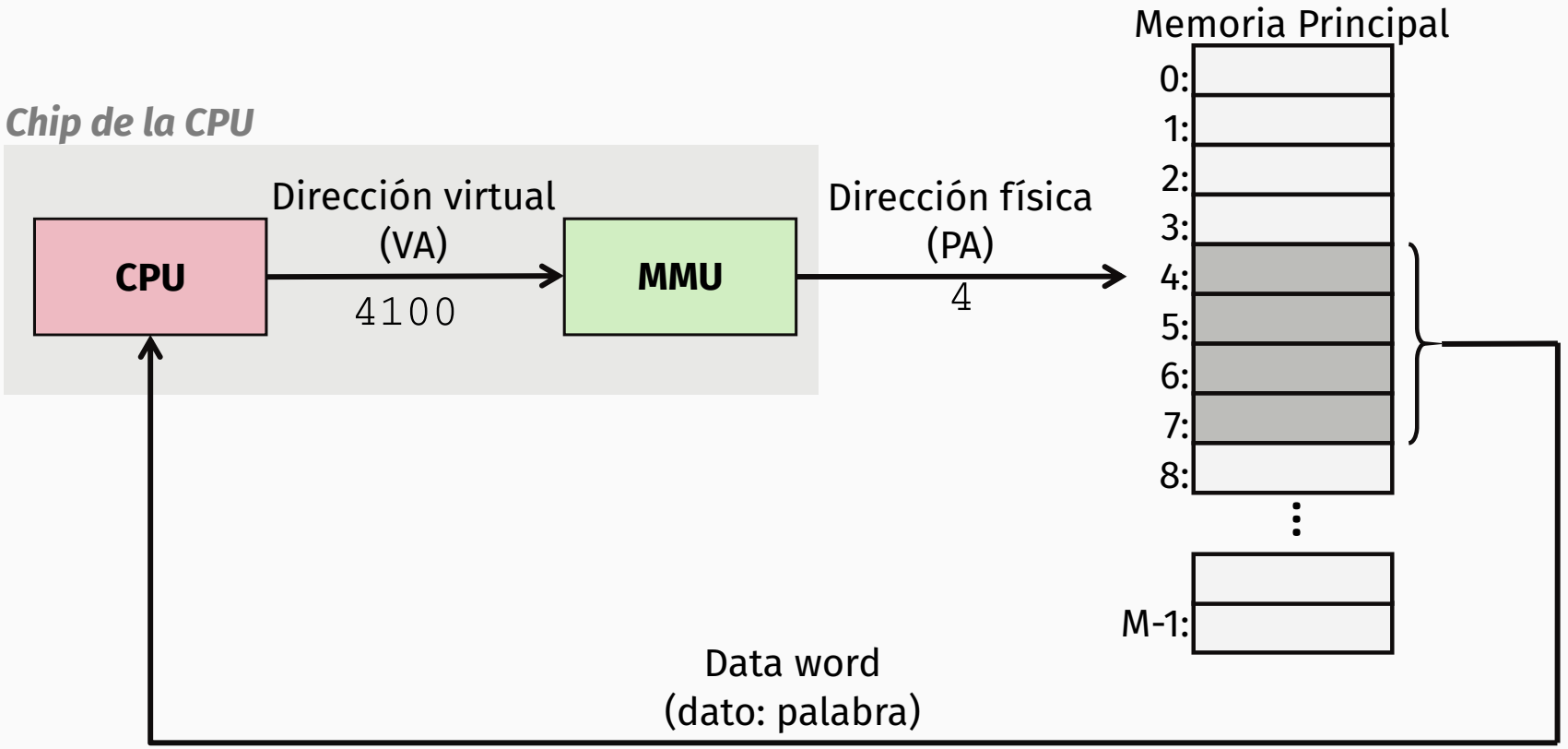
# Tabla de contenidos

# Direccionamiento físico




- Se usa en sistemas como los embebidos, e.g. autos, microondas, computadoras de vuelo, etc.

# Direccionamiento virtual




- Se usa en sistemas más complejos: servidores, computadoras, teléfonos inteligentes, televisores inteligentes, computadora de abordo de un auto.
- Una de las grandes ideas en computación.

# Espacios de direcciones

- **Espacio de direcciones lineal:** conjunto de direcciones dadas por una secuencia de números enteros no negativos:

$$\text{LAS} = \{0, 1, 2, 3, \ldots\}$$

- **Espacio de direcciones físicas:** conjunto de $M = 2^m$ direcciones físicas

$$\text{PAS} = \{0, 1, 2, 3, \ldots, M-1\}$$

- **Espacio de direcciones virtuales:** conjunto de $N = 2^n$ (típicamente $n > m$) direcciones virtuales

$$\text{VAS} = \{0, 1, 2, 3, \ldots, N-1\}$$

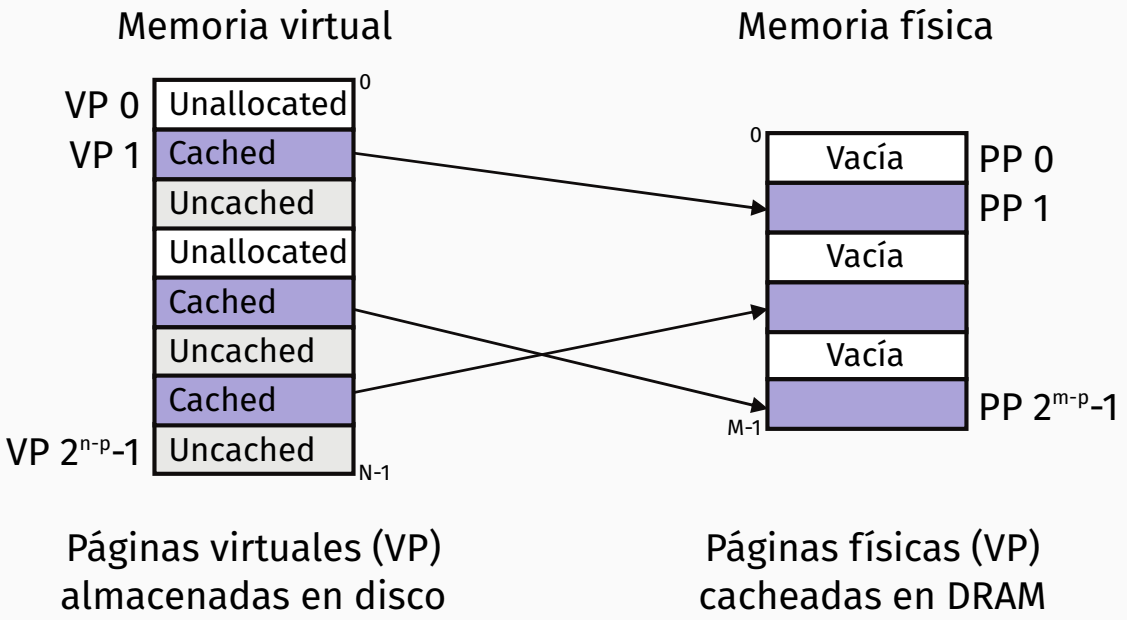
# ¿Por qué usar memoria virtual (VM)?

- **Uso eficiente de la memoria principal**
  - Se usa la DRAM como caché de partes del VAS
- **Simplifica la gestión de la memoria**
  - Cada proceso obtiene un espacio de direcciones lineal
- **Aisla los espacios de direcciones**
  - Un proceso "no" puede interferir con la memoria de otro
  - Un programa de usuario "no" puede acceder a información y código privilegiado del *kernel*

# Tabla de contenidos

# Memoria Virtual como herramienta de caché

- Conceptualmente, la **memoria virtual** es un arreglo de N bytes almacenados en forma continua en el disco.
- El contenido del arreglo en disco es *cacheado* en **memoria física** (**Cache DRAM**).
  - Los bloques de esta caché se llaman *páginas* (el tamaño es $P = 2^p$ bytes).

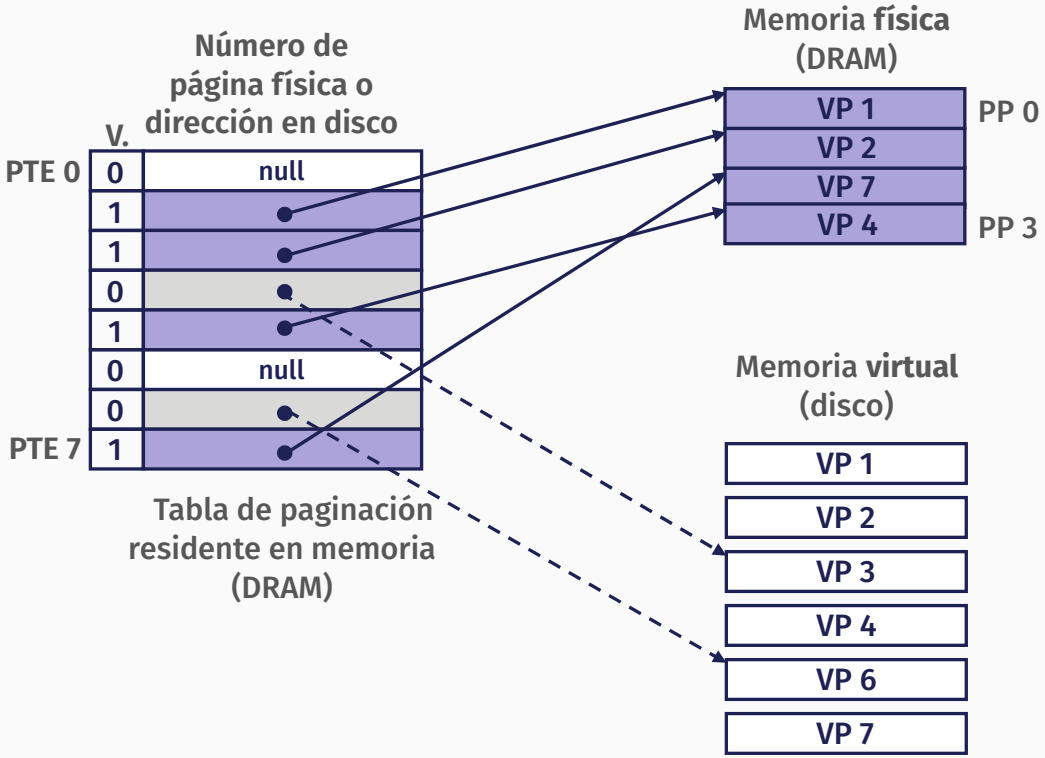


# Organización de la caché DRAM

- **Motivación:** la enorme penalización por fallos (*misses*)
  - la DRAM es 10 veces más lenta que la SRAM
  - los discos son 10000 veces más lentos que la DRAM
  - se tarda más de 1 ms en cargar un bloque del disco (más de 1 millón de ciclos de reloj)
    - la CPU puede trabajar un montón durante ese tiempo
- **Consecuencias:**
  - Tamaños de página grandes: típicamente 4 kB
    - Linux tiene páginas enormes de 2 MB (típicamente) a 1 GB
  - *Fully associative*
    - Cualquier página **virtual** se puede ubicar en cualquier página **física**
    - Requiere una función de mapeo grande
  - Algoritmos de reemplazo altamente sofisticados y costosos
  - ***write-back***

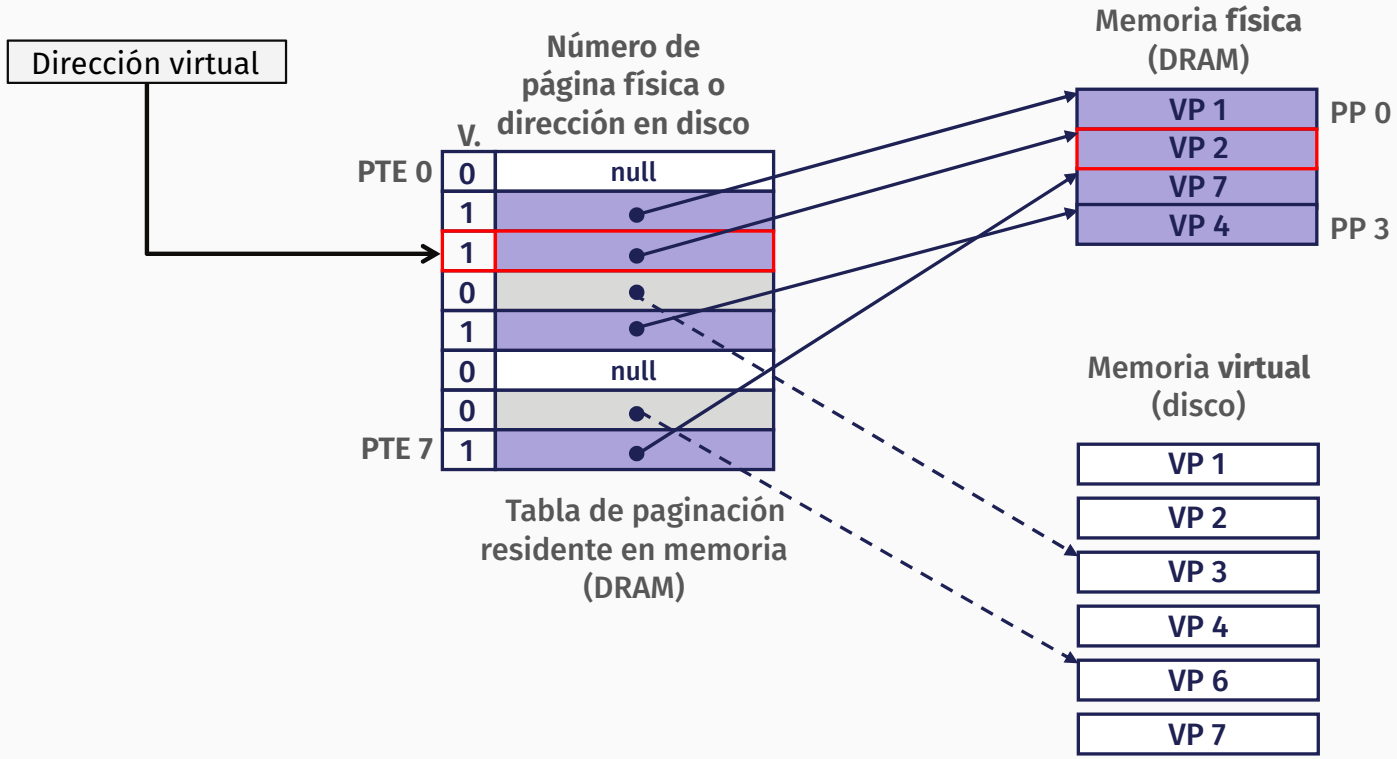
# Tablas de paginación

- Una **tabla de paginación** es está formada por entradas (PTEs, *page table entries*) que mapean páginas virtuales a páginas físicas.
  - es una estructura de datos que el kernel almacena por proceso en la DRAM



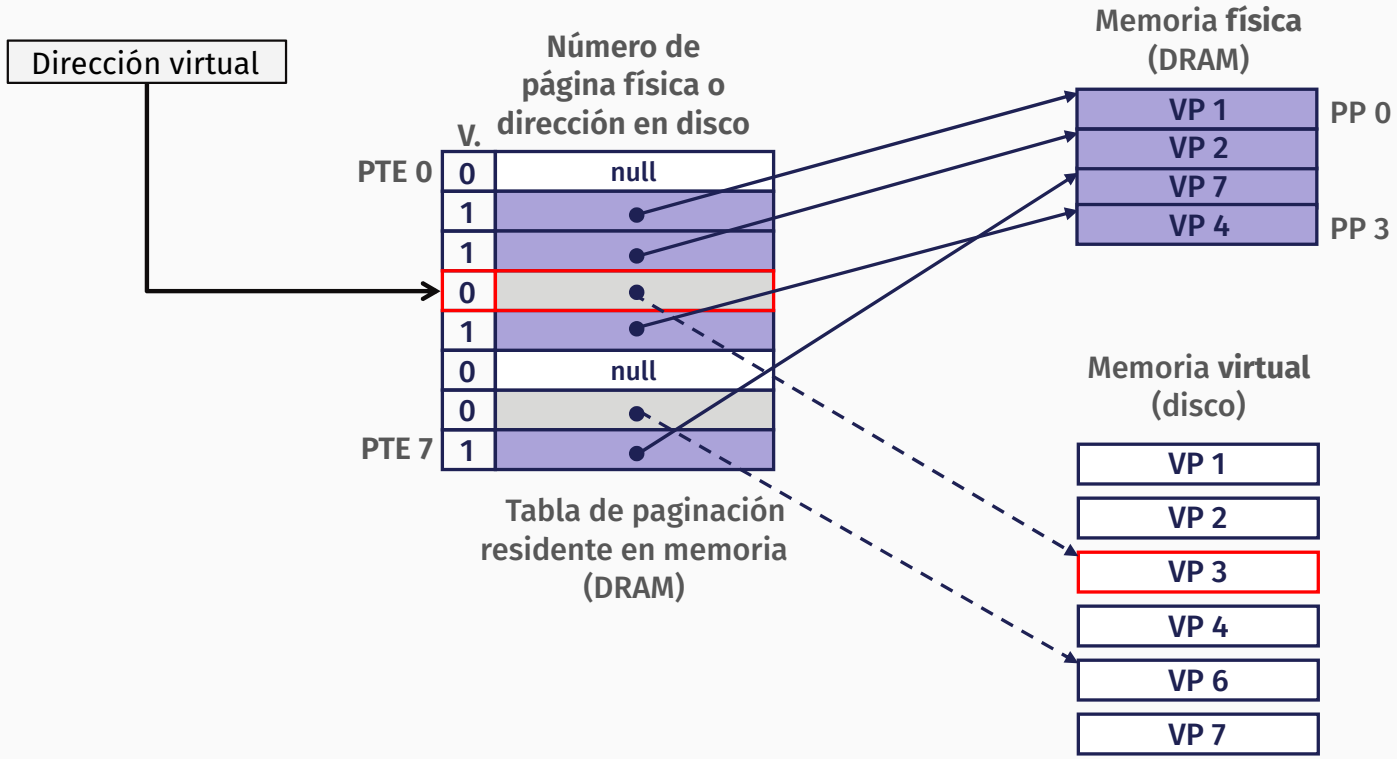
# Page hit

- se referencia un dato usando memoria virtual correspondiente a una página física que está en la memoria principal (un acierto en la caché DRAM)



# Page fault

- se referencia un dato usando memoria virtual correspondiente a una página física que **no** está en la memoria principal (un fallo/miss en la caché DRAM)

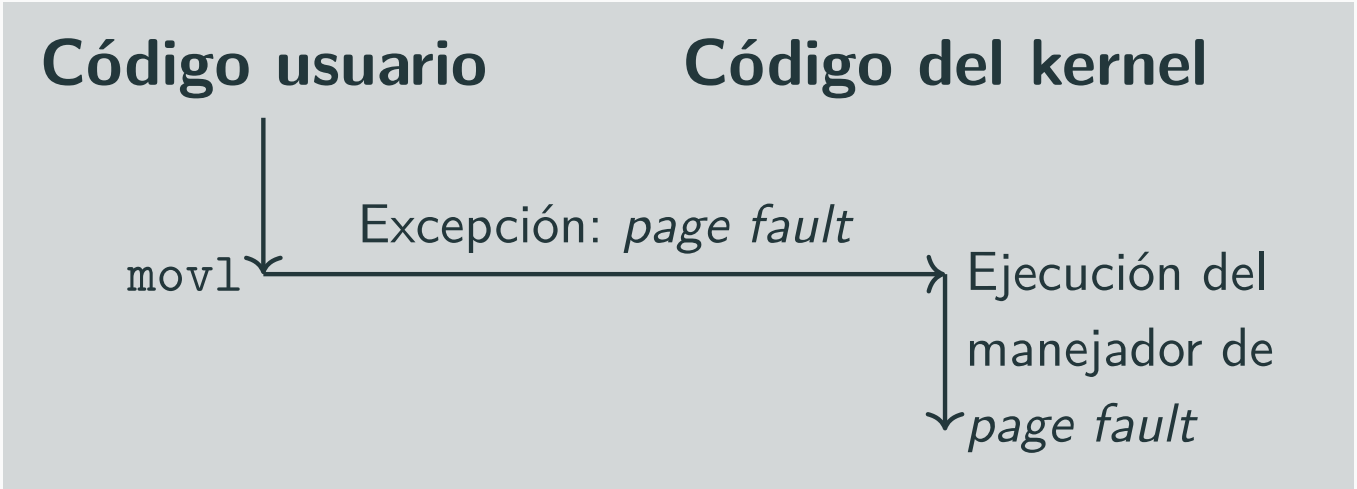


# Gestión de una *page fault*

- se escribe en una dirección de memoria

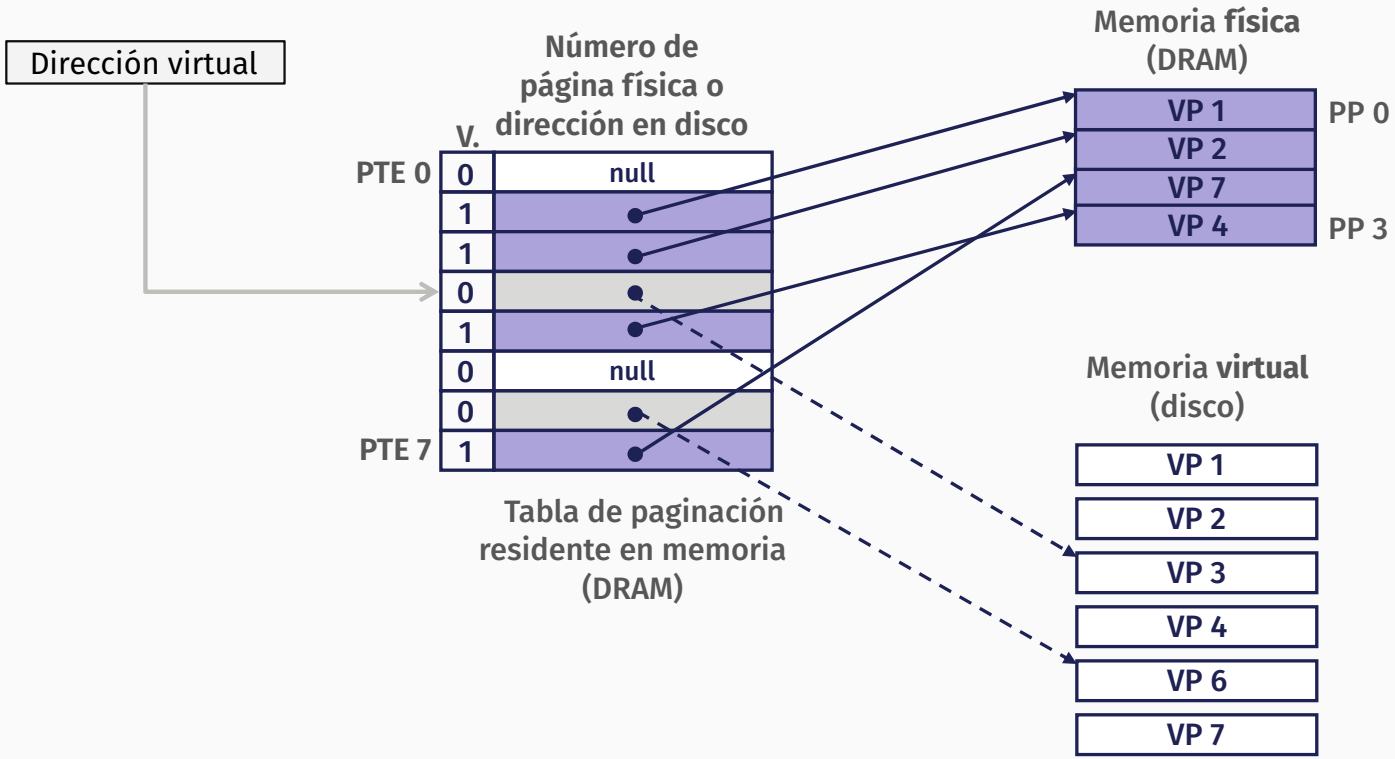
```
80483b7:    c7 05 10 9d 04 08 0d    movl    $0xd,0x8049d10
```

- esa dirección, porción, página de memoria está en el disco
- la **MMU** lanza una excepción por *page fault*
  - se elevan los privilegios al modo supervisor



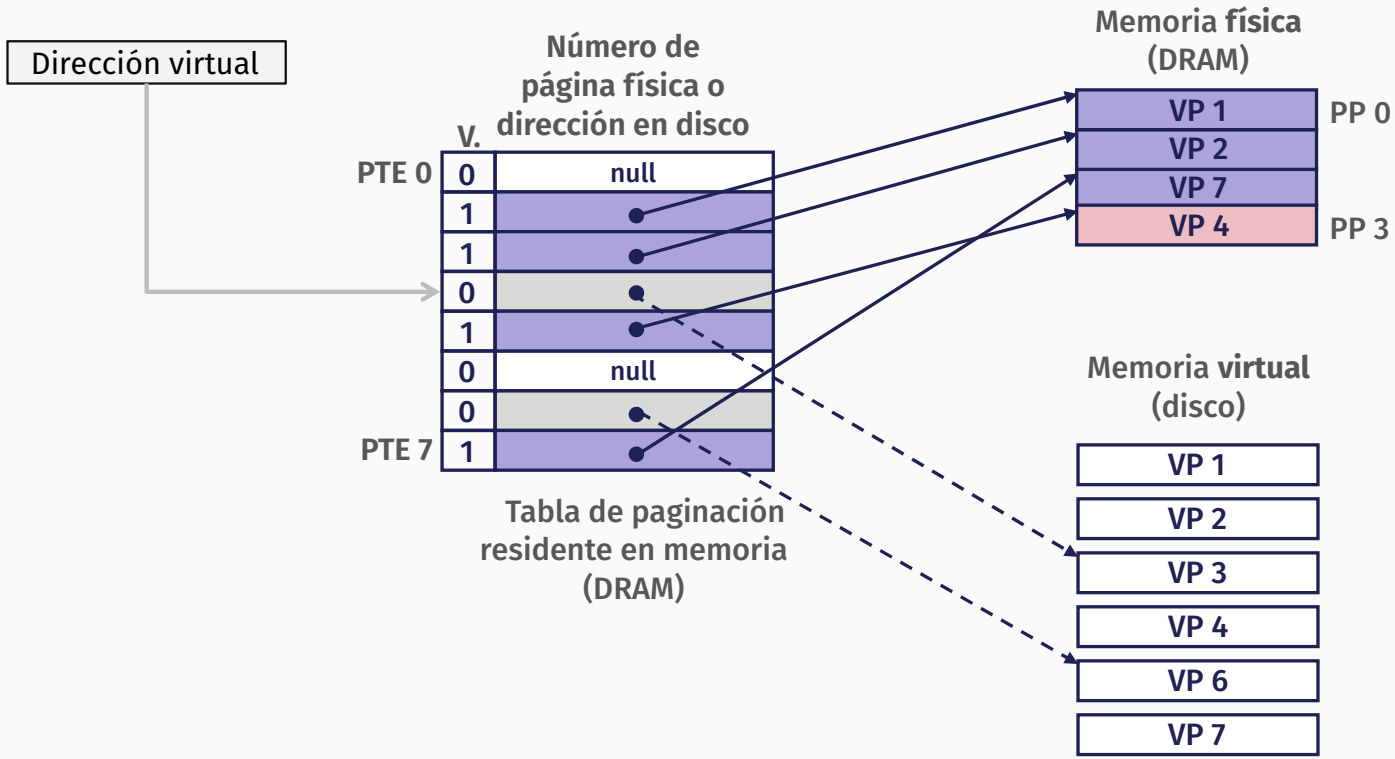
# Manejo del Page fault

- **page miss** causa la **excepción page fault**



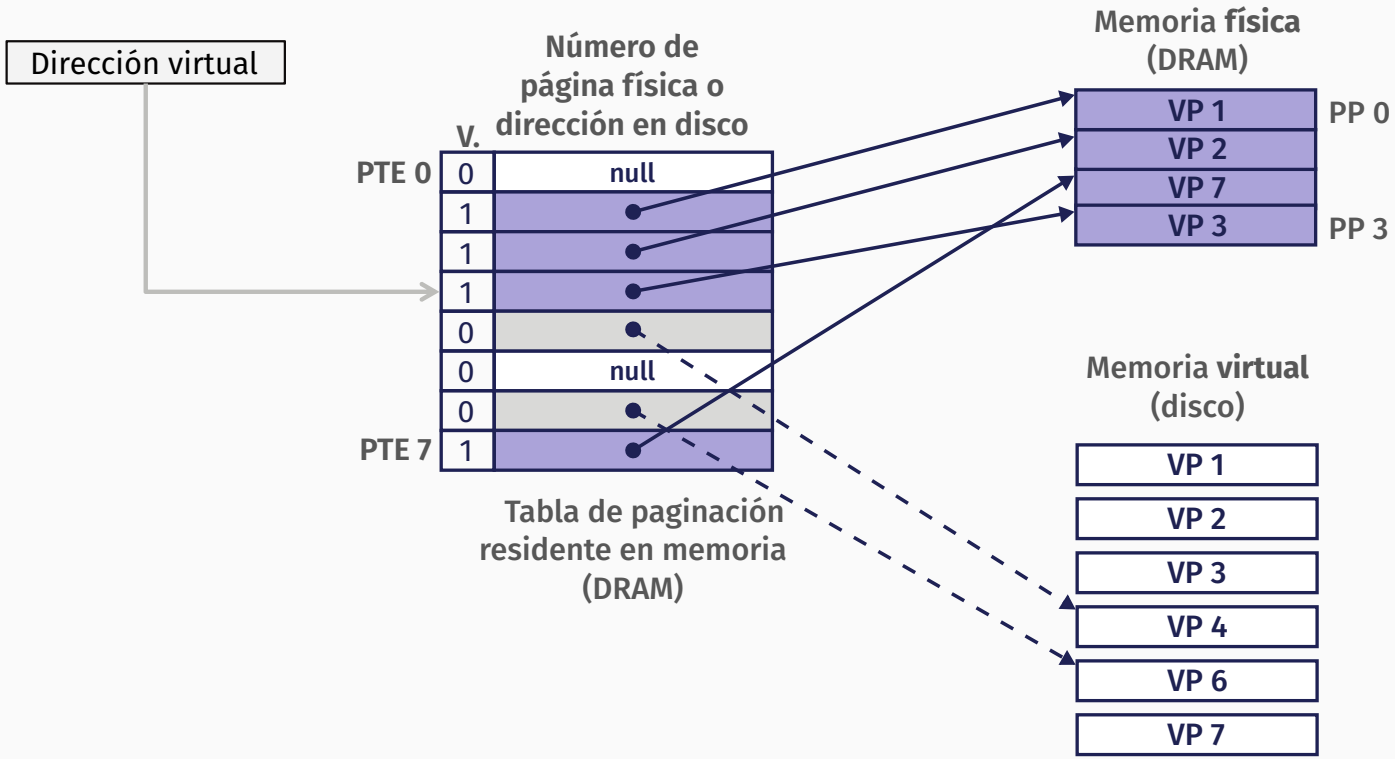
# Manejo del Page fault

- **page miss** causa la **excepción page fault**
- El *handler* de la excepción selecciona una víctima para desalojar (en ejemplo, es VP 4)



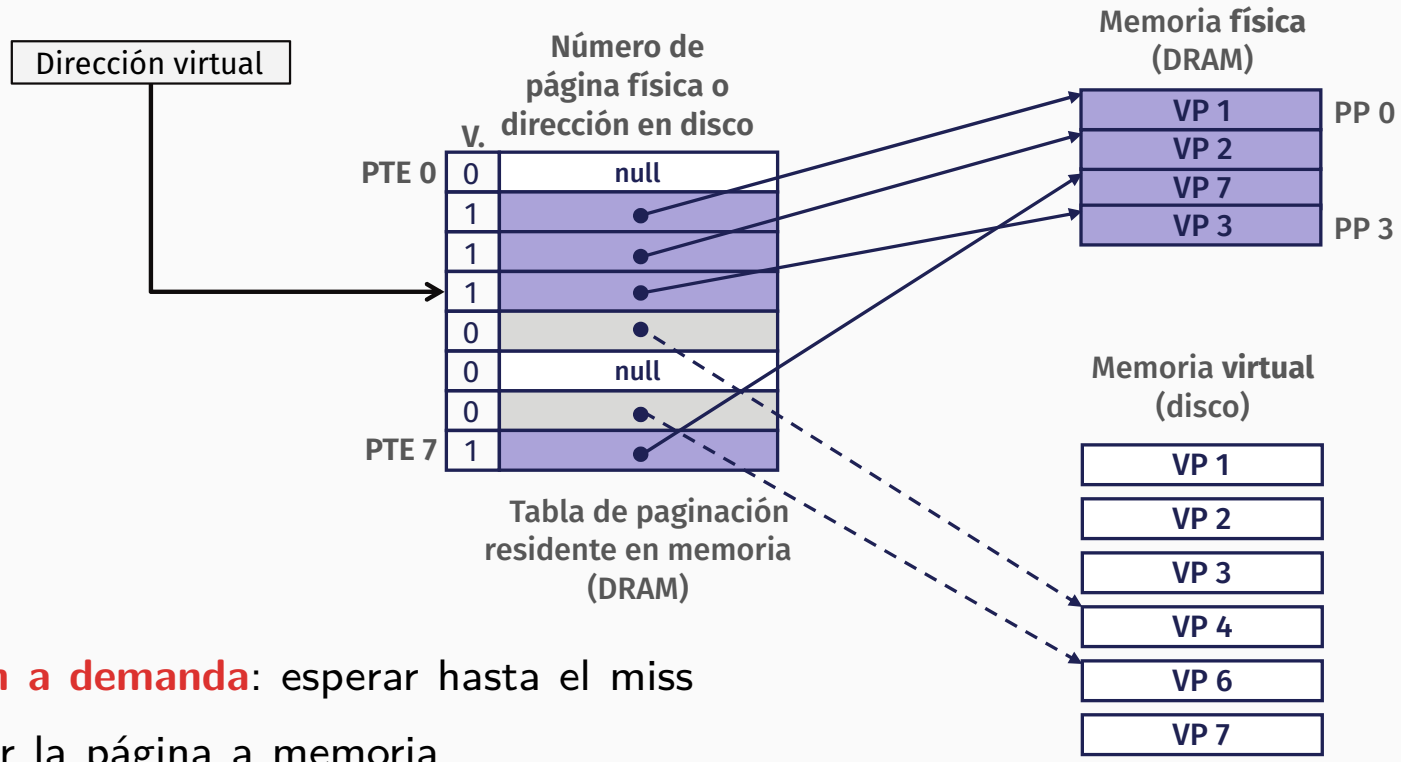
# Manejo del Page fault

- **page miss** causa la **excepción page fault**
- El *handler* de la excepción selecciona una víctima para desalojar (en ejemplo, es VP 4)



# Manejo del Page fault

- **page miss** causa la **excepción page fault**
- El *handler* de la excepción selecciona una víctima para desalojar (en ejemplo, es VP 4)
- La instrucción que causó la excepción es reiniciada: *page hit*

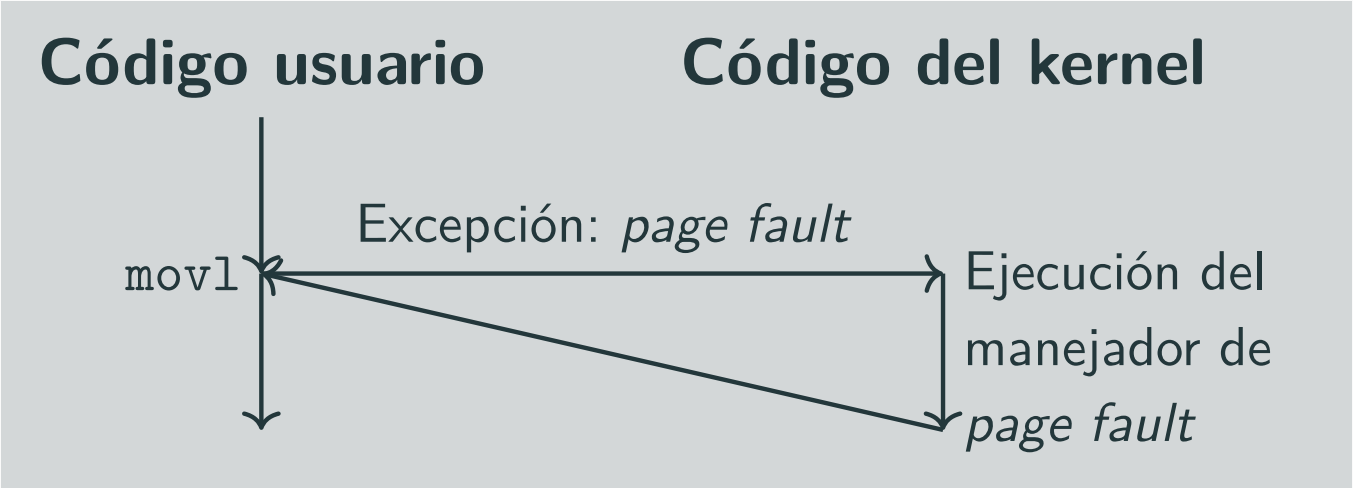



**paginación a demanda**: esperar hasta el miss para copiar la página a memoria

# Finalización de la excepción

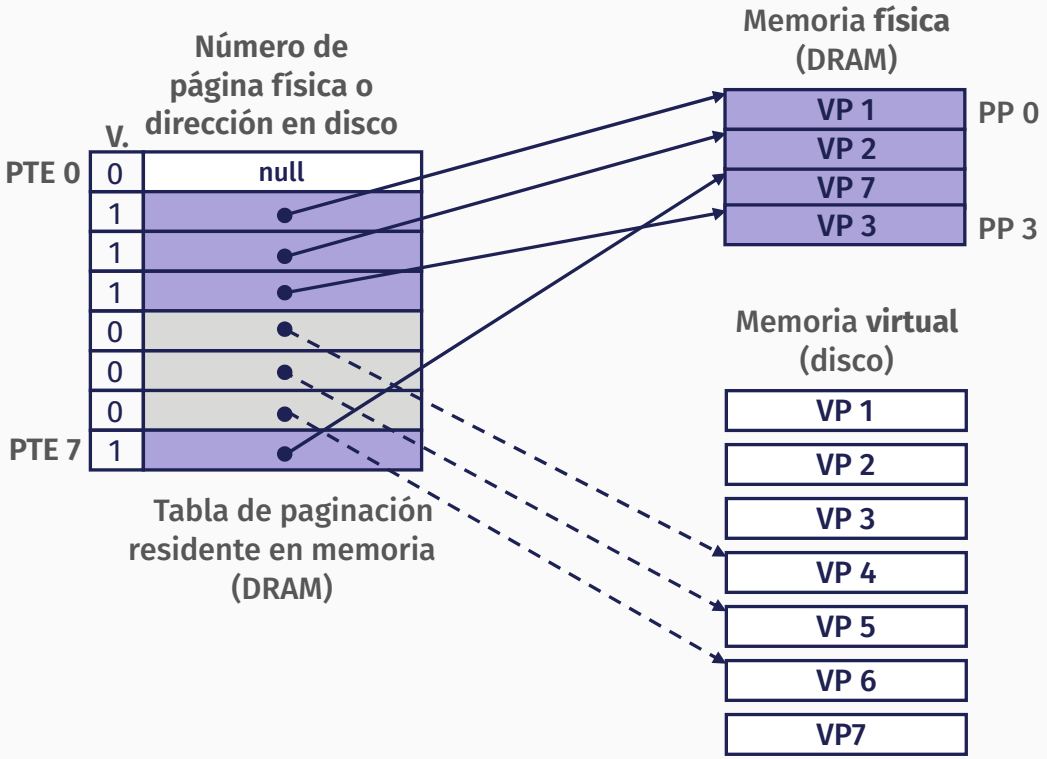
- el *handler* ejecuta la instrucción para retorno desde interrupciones (`iret`)
  - Es como `ret` pero restablece el nivel de privilegios
  - Retorna a la instrucción que causó la excepción
    - esta vez se ejecuta sin fallo

```
80483b7:    c7 05 10 9d 04 08 0d    movl    $0xd,0x8049d10
```



# Reserva de páginas

- reserva de nuevas páginas de memoria virtual (VP 5 en el ejemplo)




- el siguiente miss la carga en memoria

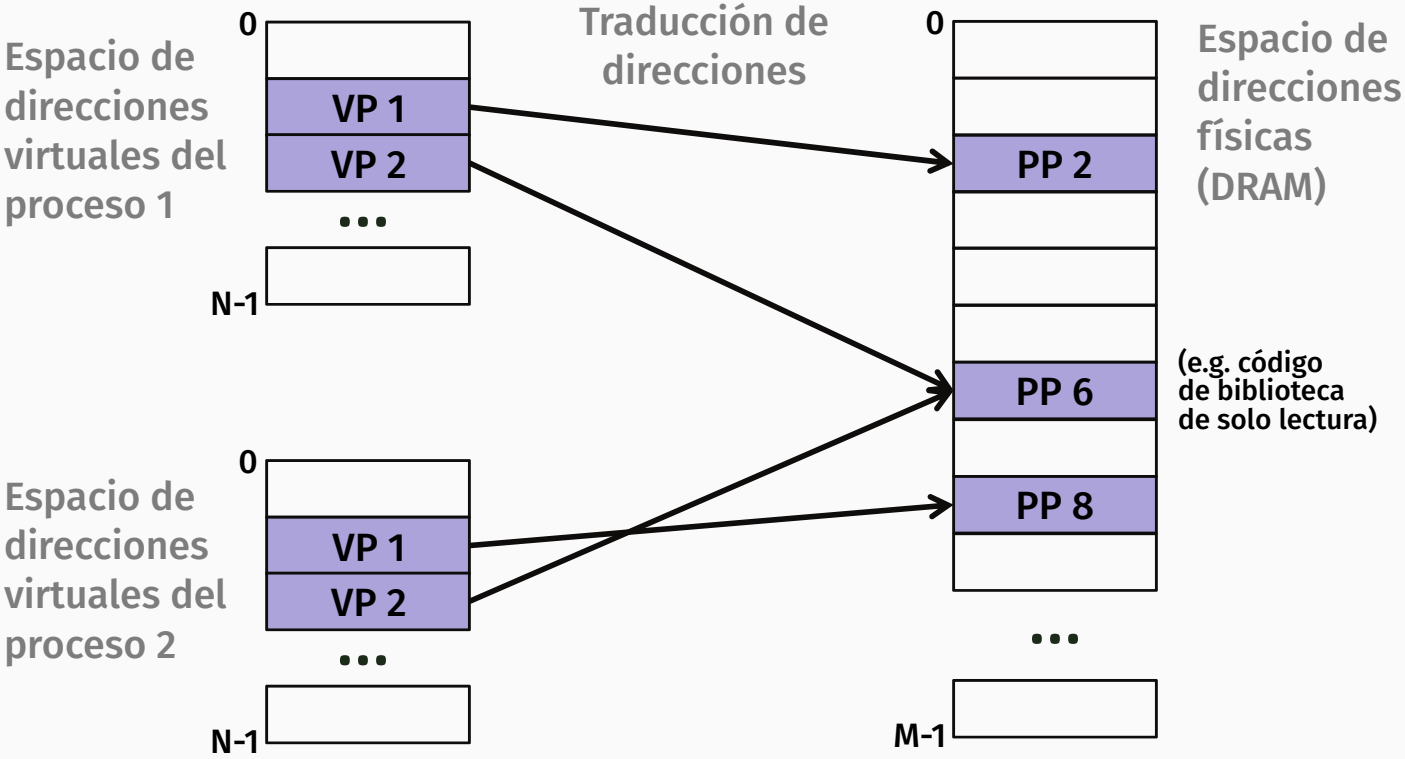
# Localidad al rescate

- La memoria virtual parece muy ineficiente, pero funciona por el principio de localidad
- Los programas tienden a acceder a un conjunto activo de páginas llamado **conjunto de trabajo** (**working set**)
  - los programas con buena localidad temporal mantienen conjuntos de trabajo reducidos
- `if (conjunto de trabajo < memoria principal)`
  - desempeño bueno para el proceso, después de los fallos en frío
- `if (conjunto de trabajo > memoria principal)`
  - *Thrashing*: el desempeño cae abruptamente (desaparece en la práctica) porque las páginas se copian continuamente entre disco y DRAM (se usa la memoria *swap*)
- Si se ejecutan varios procesos en simultáneo, el conjunto de trabajo total es la suma de los conjuntos.

# Tabla de contenidos

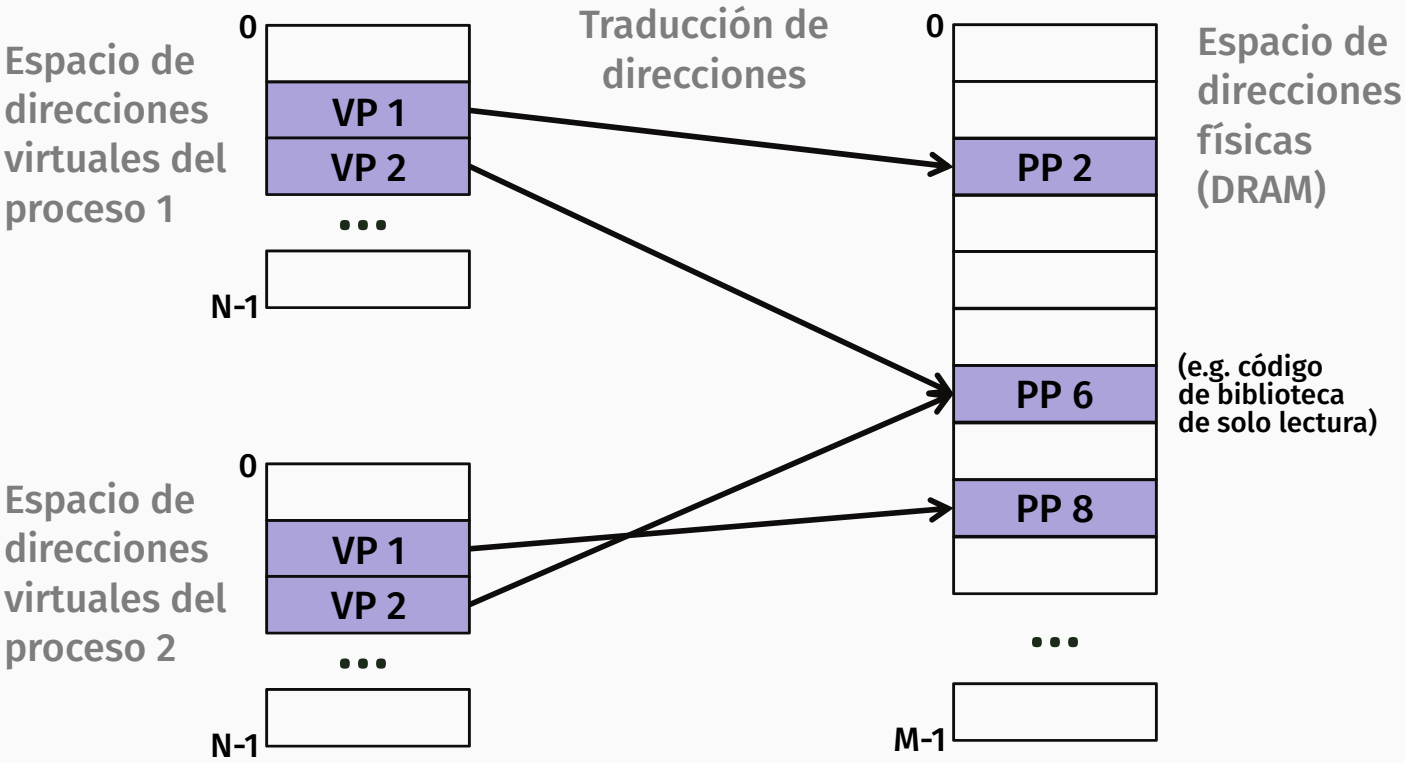
# Memoria Virtual para gestionar la memoria

- Cada proceso tiene su propio espacio de direcciones virtuales
  - permite ver la memoria como un arreglo de bytes
  - la función de mapeo se encarga de distribuir las direcciones en la memoria física
  - esta función influye en la localidad

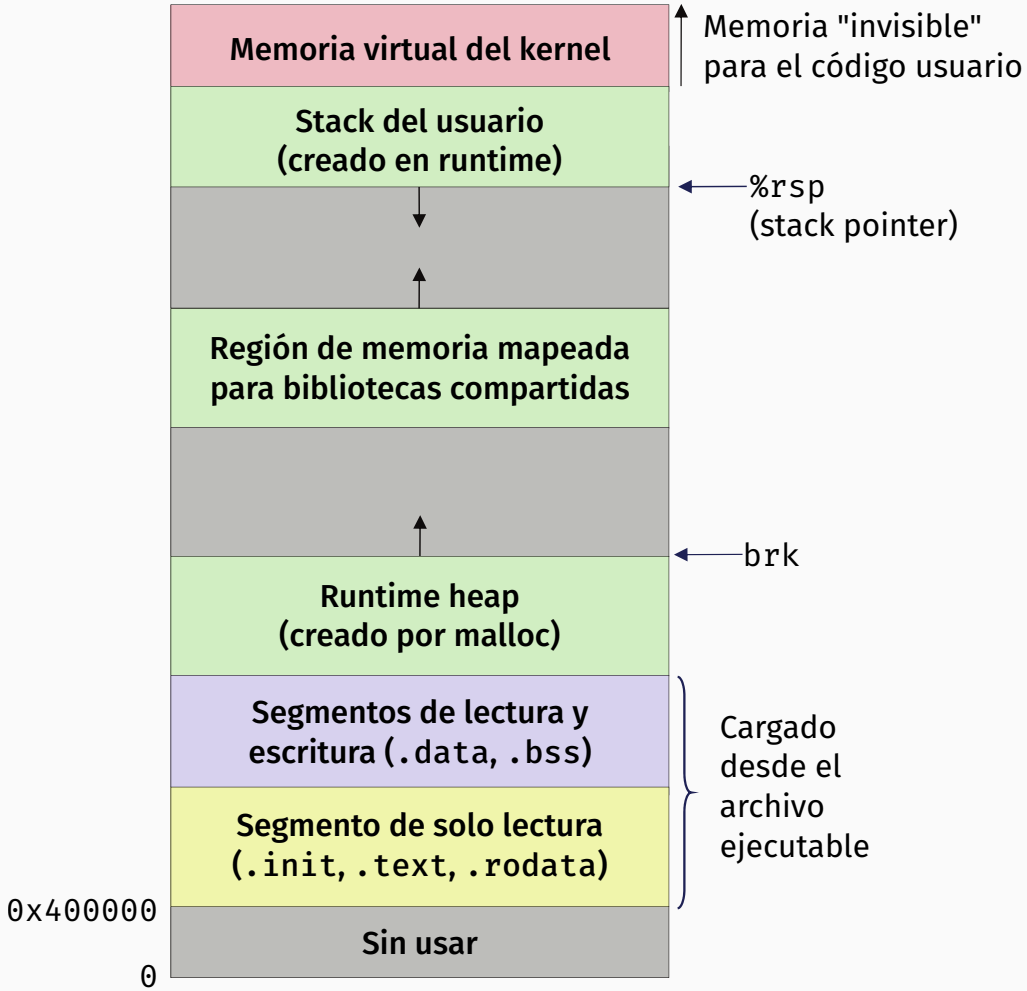


# Memoria Virtual para gestionar la memoria

- Simplifica la reserva de memoria
  - cada página virtual se puede mapear a cualquier página física
  - en diferentes momentos, una misma página virtual se puede almacenar en páginas físicas distintas
- Es simple compartir código y datos entre procesos
  - Mapea páginas virtuales distintas a una misma página física

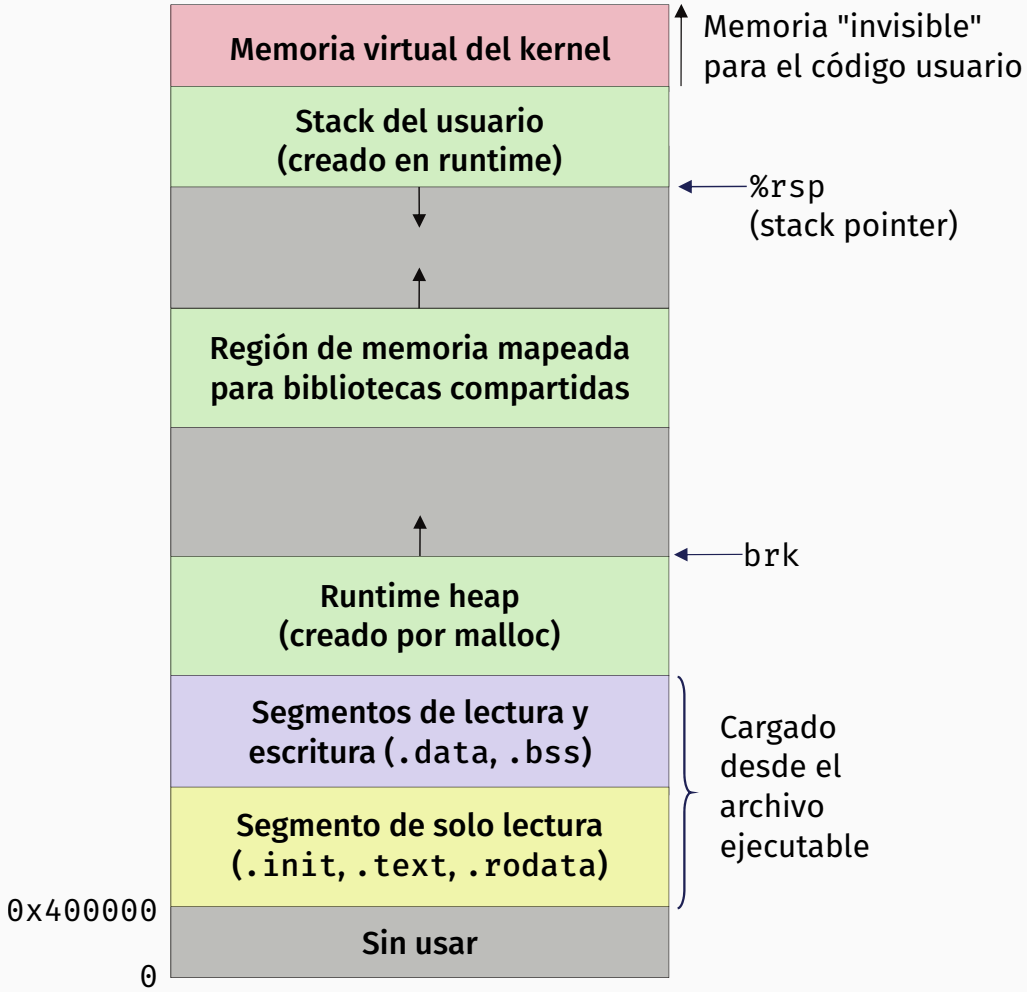


# Enlazado y carga



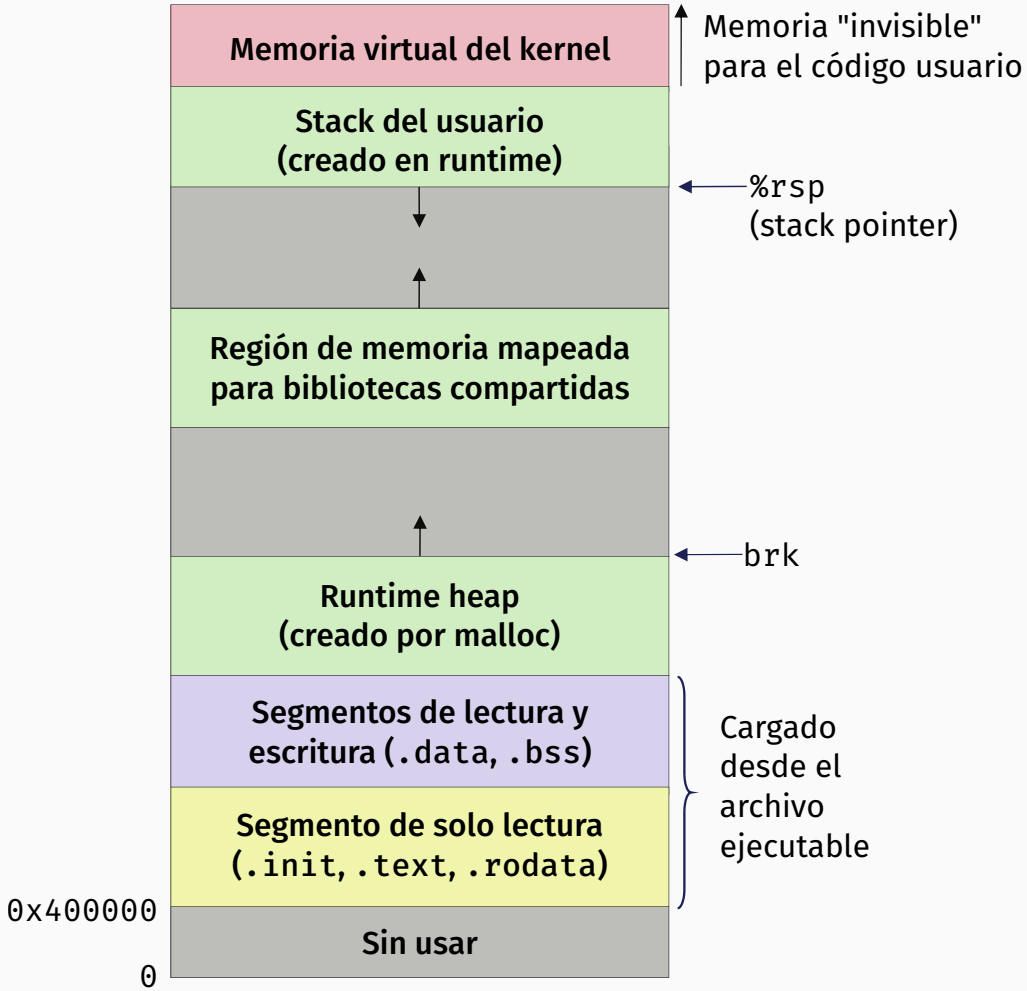
# Enlazado y carga

- **Enlazado**
  - Cada programa tiene un VAS similar al resto
  - Las secciones de código, datos, y el *heap* siempre comienzan en los mismos lugares



# Enlazado y carga

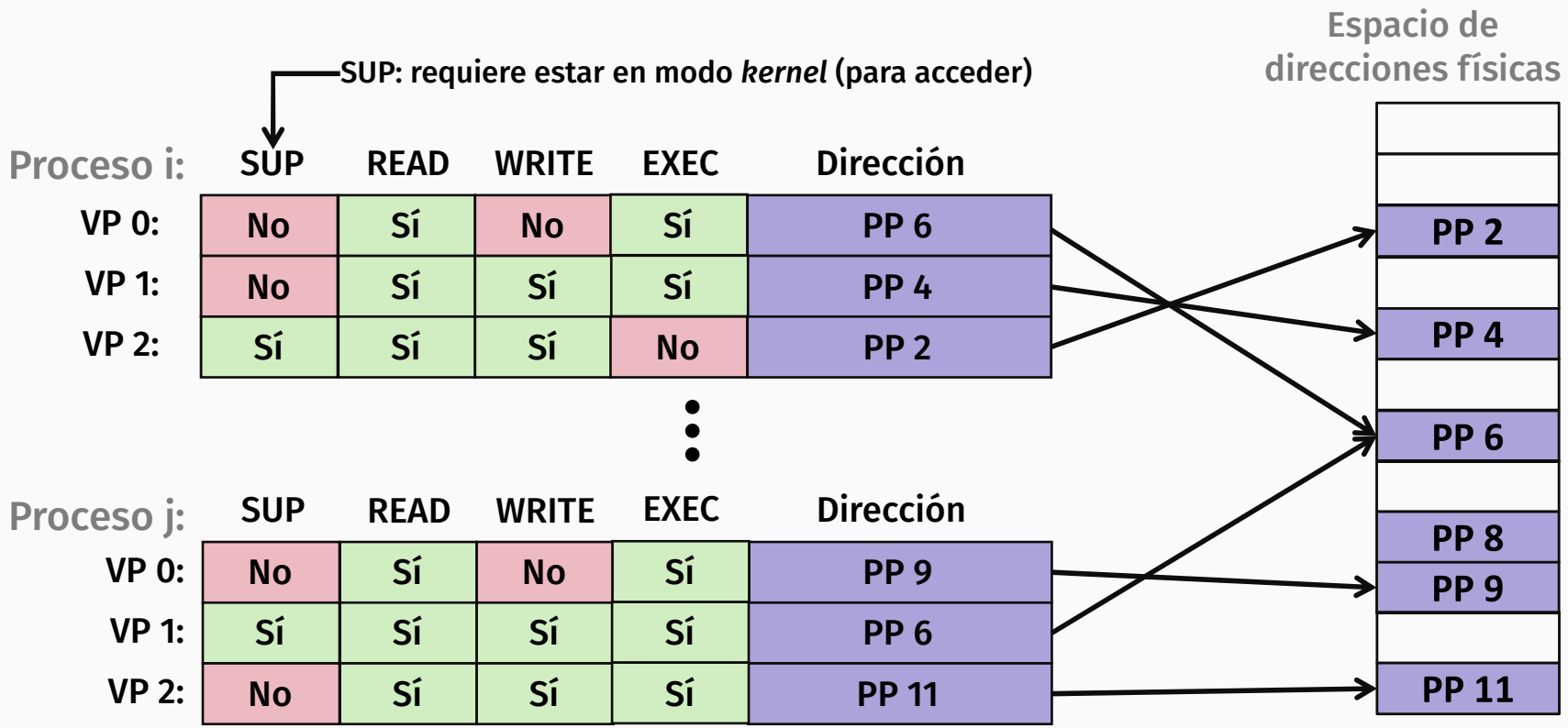
- **Enlazado**
  - Cada programa tiene un VAS similar al resto
  - Las secciones de código, datos, y el *heap* siempre comienzan en los mismos lugares
- **Carga**
  - La función `execve` reserva páginas virtuales para las secciones `.text` y `.data` y crea PTEs marcados como inválidos.
  - Las secciones `.text` y `.data` se copian, página por página, a demanda.



# Tabla de contenidos

# Memoria virtual como herramienta de protección

- Se extienden las PTEs como bits de permisos
- La MMU comprueba estos bits en cada acceso



# Tabla de contenidos

# Traducción de direcciones

- Espacio de direcciones virtuales:

$$\mathcal{V} = \{0, 1, \ldots, N-1\}$$

- Espacio de direcciones físicas:

$$\mathcal{P} = \{0, 1, \ldots, M-1\}$$

- Traducción de direcciones virtuales:

$$\text{MAPA} : \mathcal{V} \mapsto \mathcal{P} \cup \{\emptyset\}$$

Dada $a_v \in \mathcal{V}$, y $a_p = \text{MAP}(a_v)$, entonces se cumple:

  - $a_p \in \mathcal{P}$ si el dato referenciado por va está en la memoria física
  - $a_p = \emptyset$ si el dato no está cargado en memoria (ya sea inválida va o almacenado en disco)

# Símbolos para la traducción de direcciones

- **Parámetros básicos**

| | |
|---|---|
| **VAS** | espacio de direcciones virtuales |
| **PAS** | espacio de direcciones físicas |
| $\mathbf{N = 2^n}$ | cantidad de direcciones en el VAS |
| $\mathbf{M = 2^m}$ | cantidad de direcciones en el PAS |
| $\mathbf{P = 2^p}$ | tamaño de la página en bytes |

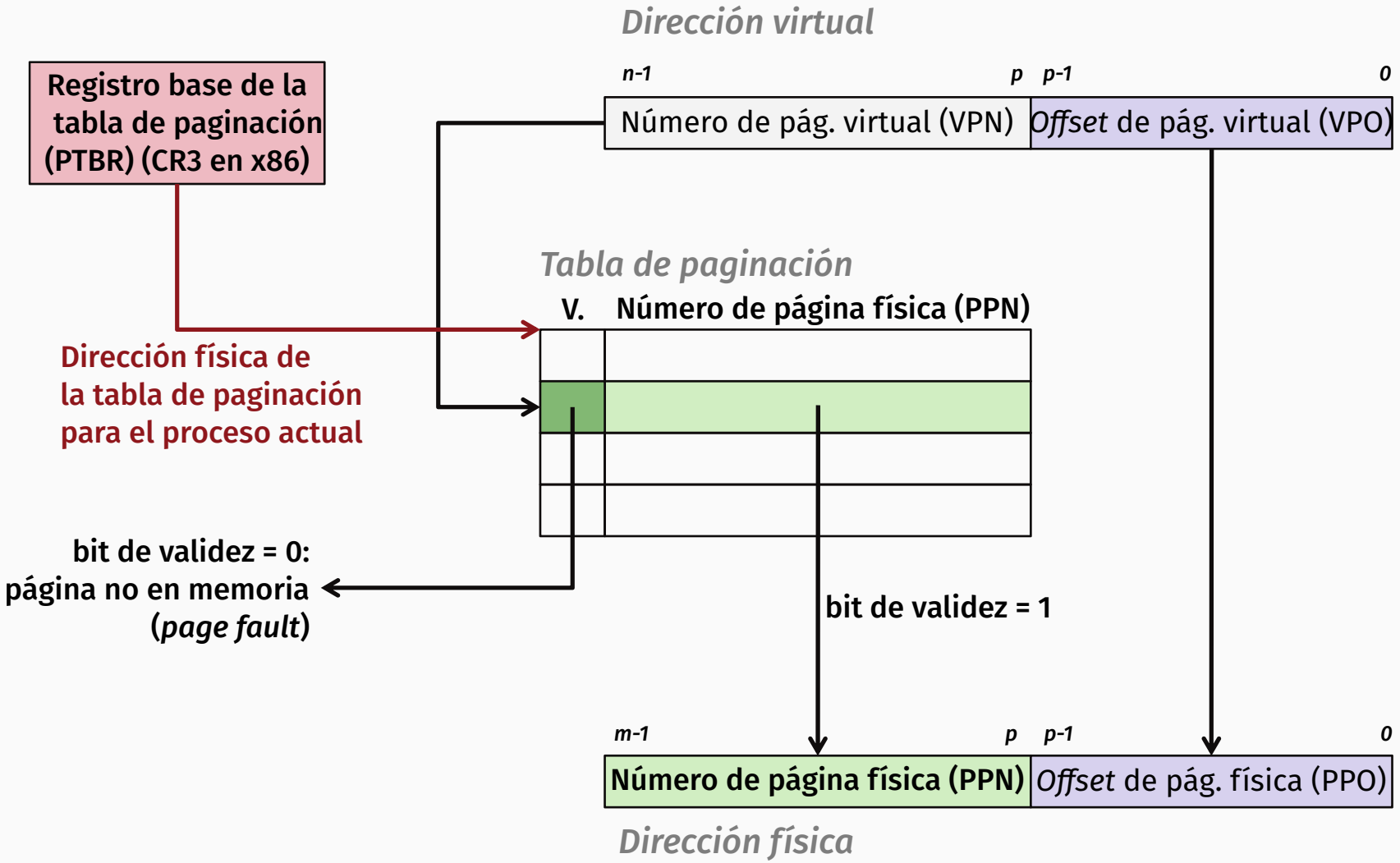
- **Componentes de una dirección virtual (VA)**

| | |
|---|---|
| **VPO** | offset de la página virtual (igual al PPO) |
| **VPN** | número de página virtual |

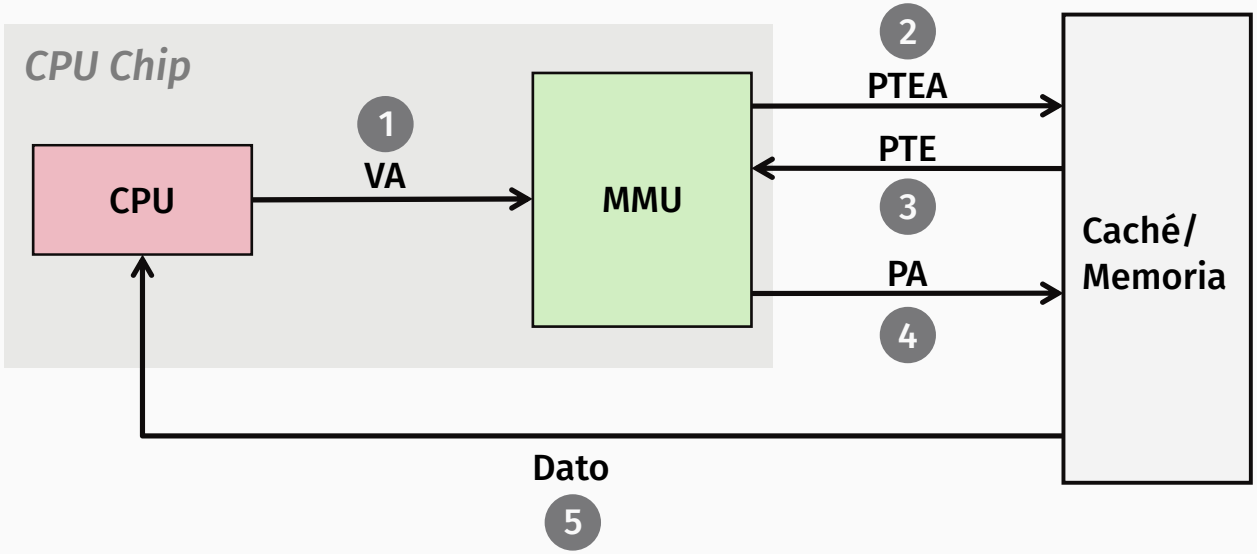
- **Componentes de una dirección física (PA)**

| | |
|---|---|
| **PPO** | offset de la página física (igual al VPO) |
| **PPN** | número de página física |

# Traducción de direcciones con tabla de paginación

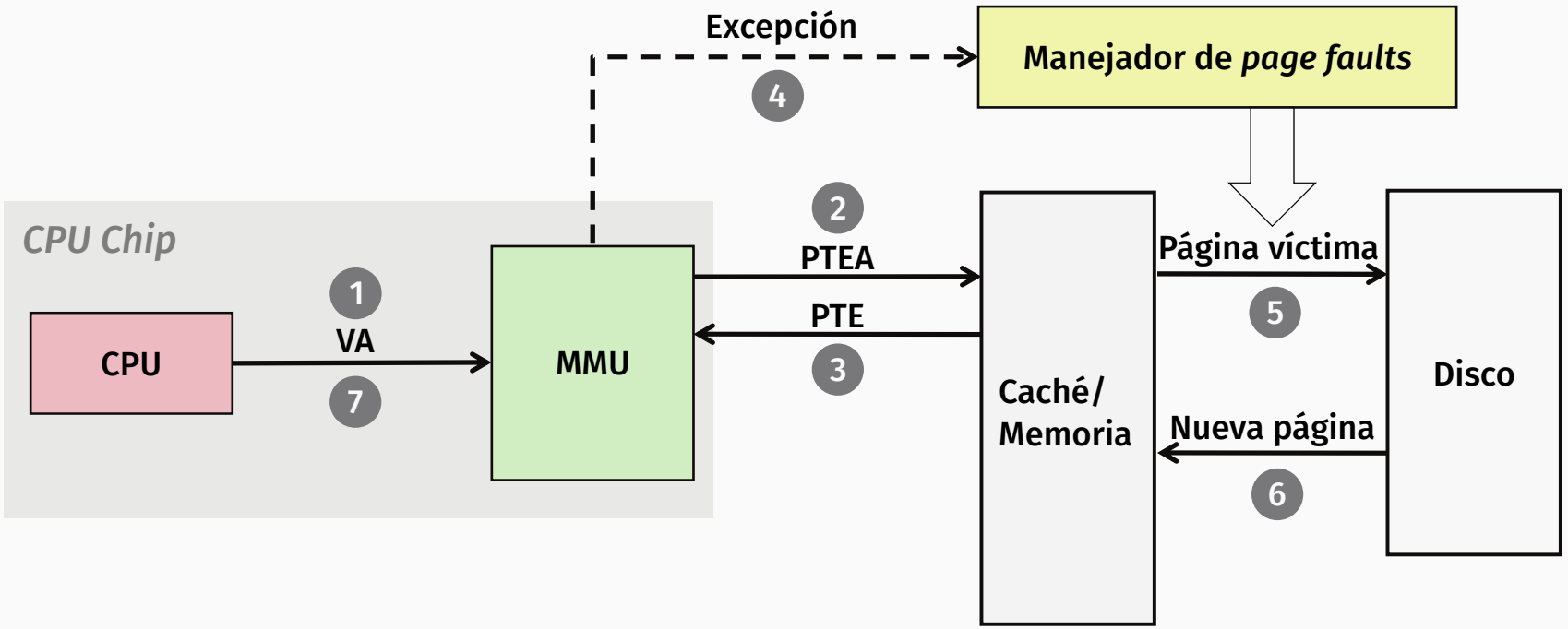


# Page hit



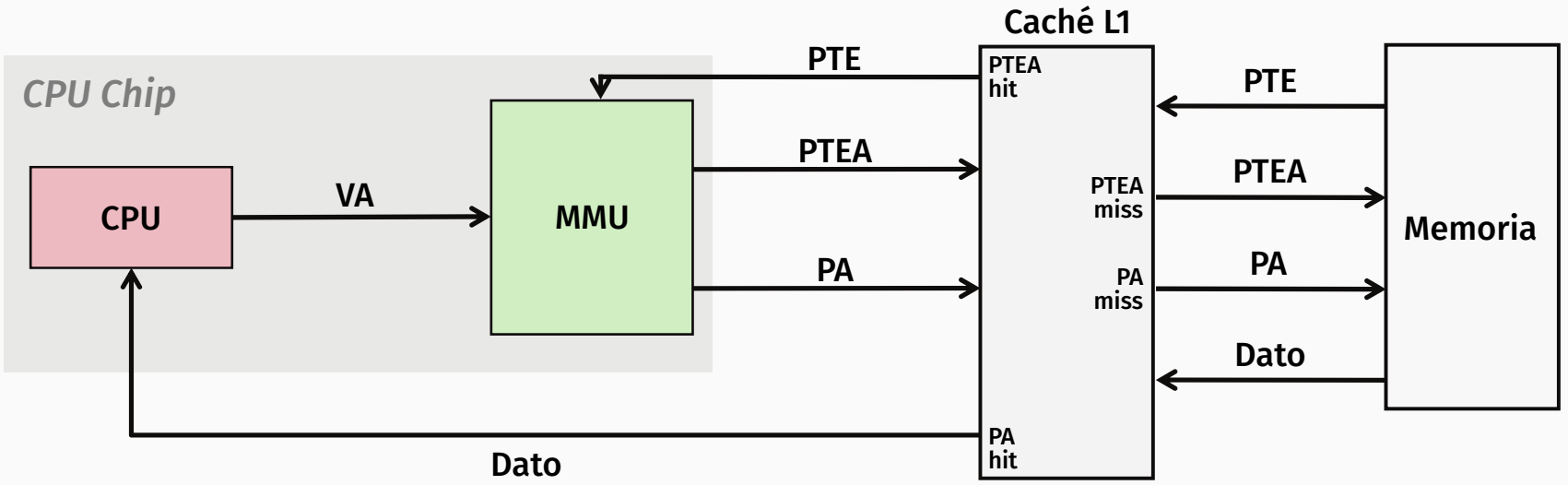

1. El procesador envía la dirección virtual, VA, a la MMU
2.-3. La MMU obtiene la PTE de la tabla de paginación en memoria
4. La MMU envía la dirección física a la memoria
5. La caché/memoria envía el dato al procesador

# Page fault




1. El procesador envía la dirección virtual, VA, a la MMU
2.-3. La MMU obtiene la PTE de la tabla de paginación en memoria
4. El bit de validez es cero y la MMU lanza la excepción
5. El *handler* elije la víctima (si es necesario, la escribe al disco)
6. El *handler* carga la nueva página y actualiza el PTE en memoria
7. El *handler* vuelve al proceso original y reinicia la instrucción

# Integración de la memoria caché y la memoria virtual




VA: dirección virtual PA: dirección física PTE: entrada de la tabla de paginación PTEA: dirección de una PTE

# Aceleración de la traducción usando un TLB

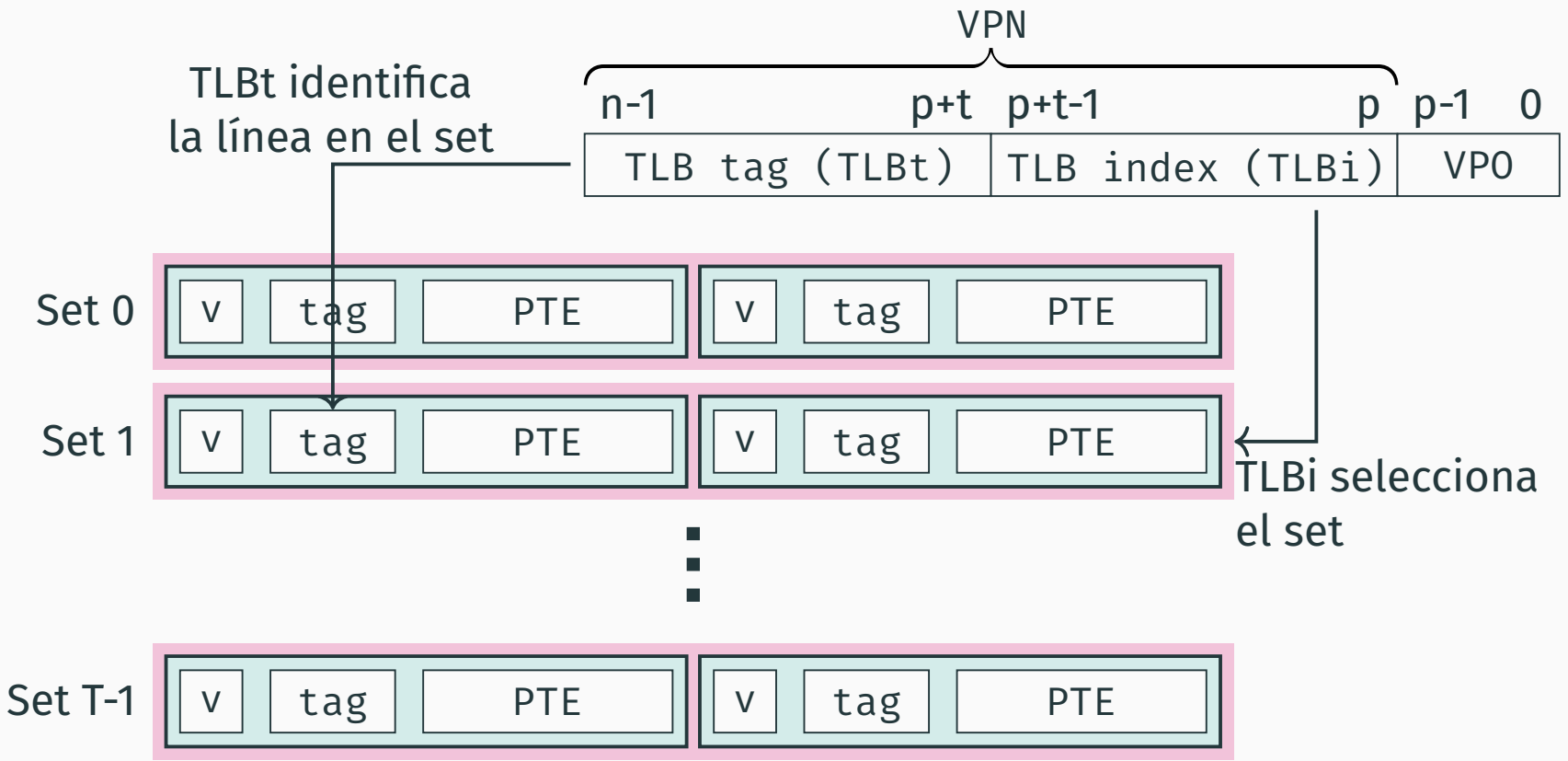
- Las entradas de la tabla de paginación (PTEs) se guardan en la caché L1 como cualquier otro dato
  - Las PTEs podrían ser desalojadas por cualquier otra referencia
  - Los hits todavía tienen el retardo de la memoria caché
- Solución: ***Translation Lookaside Buffer (TLB)***
  - Es una caché asociativa por conjuntos, en hardware, en la MMU
  - Mapea números de páginas virtuales con números de páginas físicas
  - Contiene PTEs completas para un subconjunto pequeño de páginas

# Símbolos para la traducción de direcciones

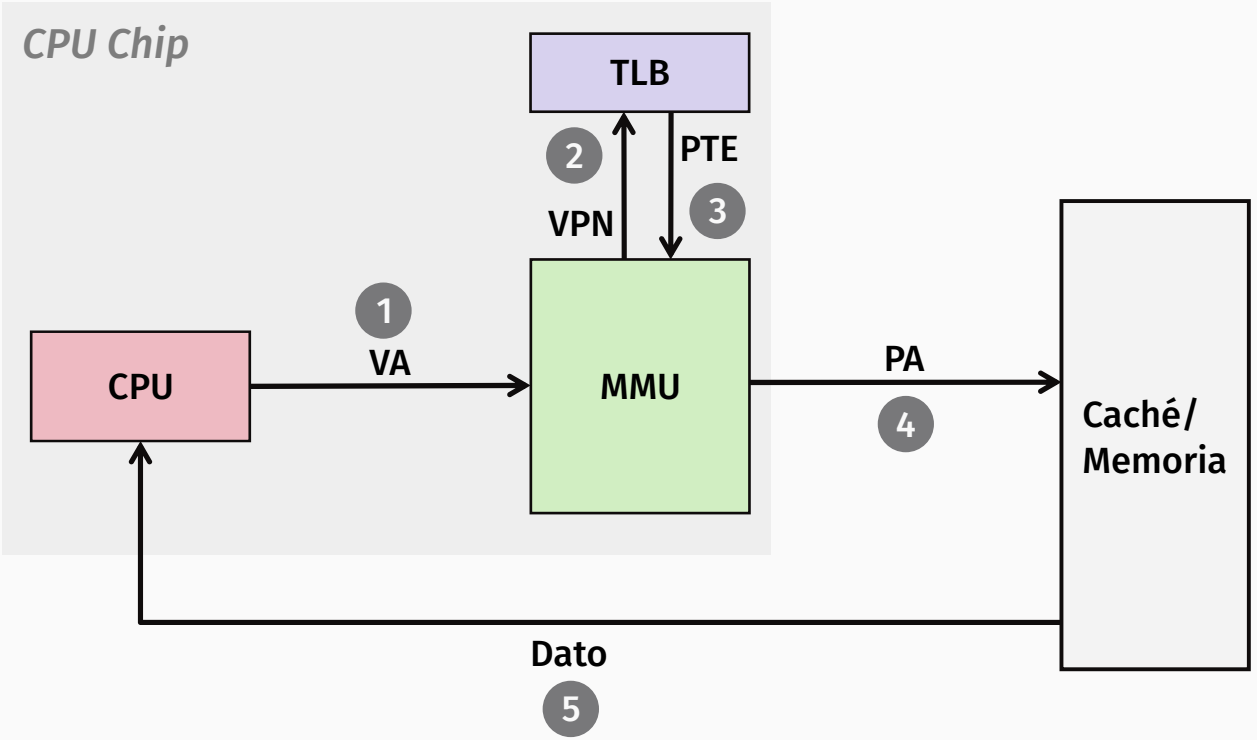
- **Parámetros básicos**
  - **VAS** espacio de direcciones virtuales
  - **PAS** espacio de direcciones físicas
  - **N** $= 2^n$ cantidad de direcciones en el VAS
  - **M** $= 2^m$ cantidad de direcciones en el PAS
  - **P** $= 2^p$ tamaño de la página en bytes
- **Componentes de una dirección virtual (VA)**
  - **TLBi** TLB index (¡nuevo!)
  - **TLBt** TLB tag (¡nuevo!)
  - **VPO** offset de la página virtual (igual al PPO)
  - **VPN** número de página virtual
- **Componentes de una dirección física (PA)**
  - **PPO** offset de la página física (igual al VPO)
  - **PPN** número de página física

# Acceso a la TLB

- La MMU usa la parte VPN de la VA para acceder a la TLB
  - Funciona como una caché de las ya vistas, pero no requiere del *cache offset* porque sólo almacena PTEs

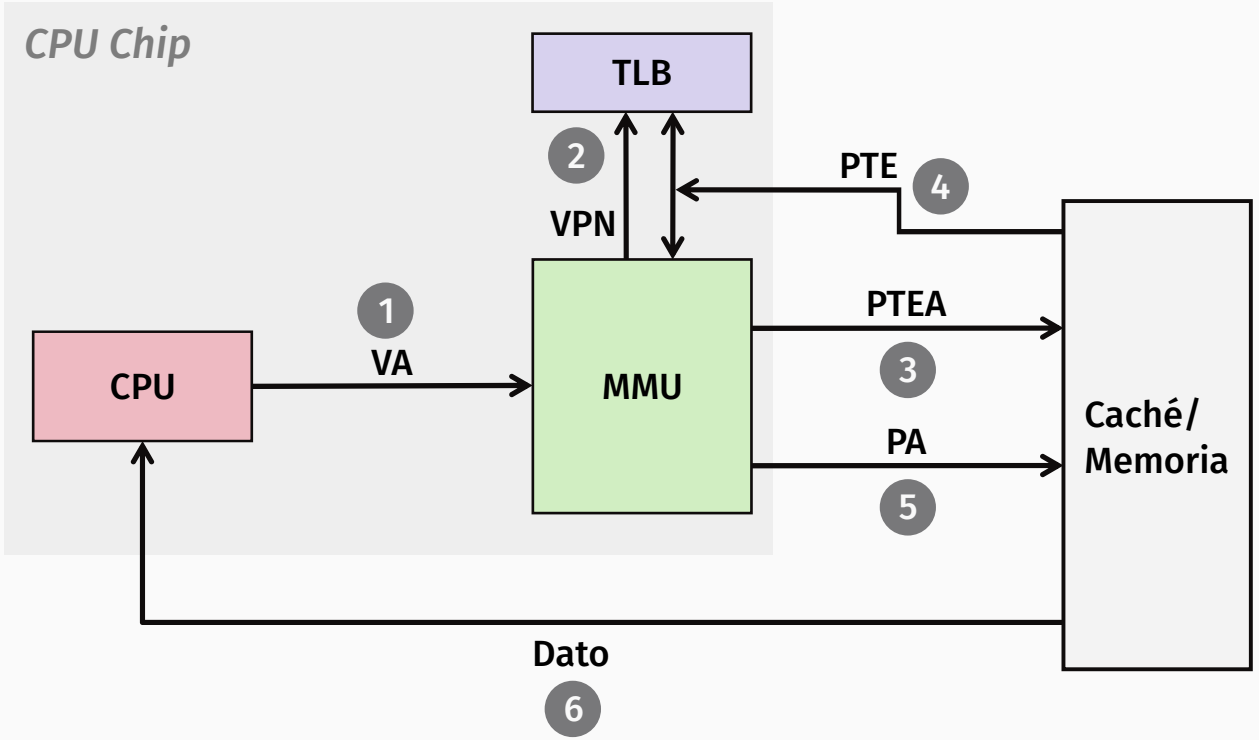


# TLB hit




Un acierto en la TLB **elimina** un acceso a memoria

# TLB miss

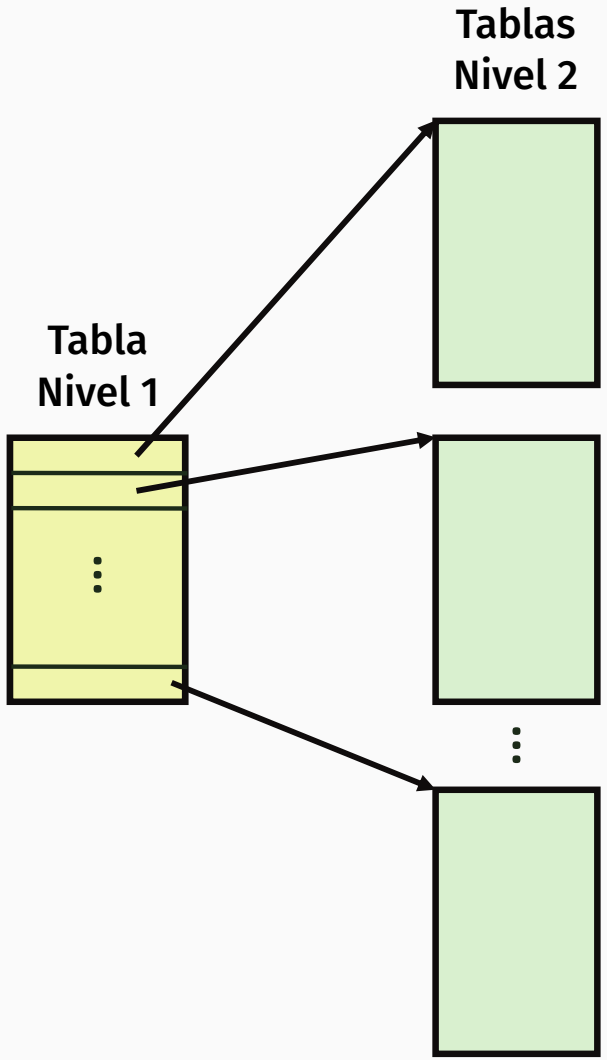



Un fallo/miss en la TLB **añade** un acceso a caché/memoria. Sin embargo, los fallos/misses en la TLB son raros.

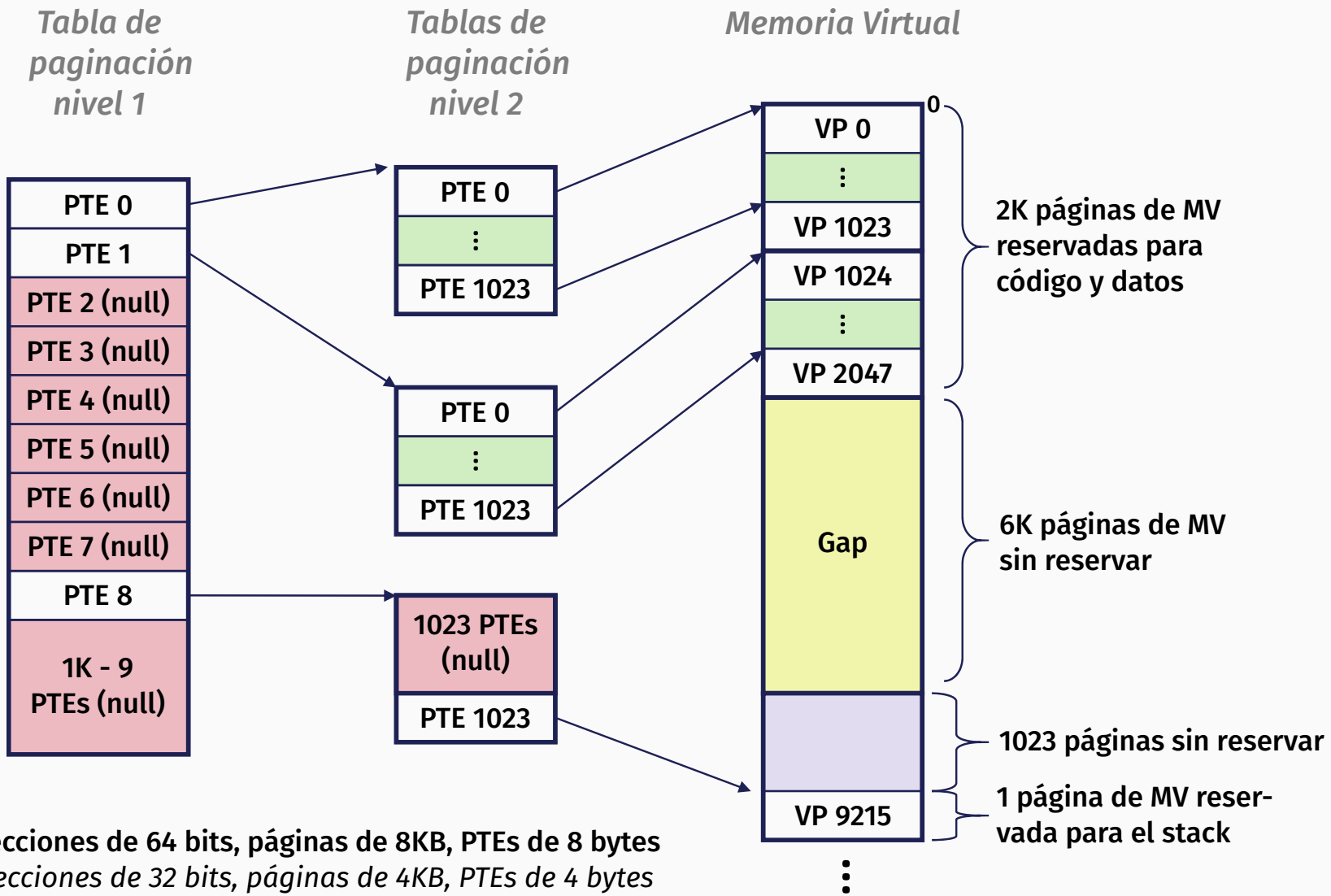
# Tabla de contenidos

# Tablas de paginación de múltiples niveles

- **Supongamos:**
  - espacio de direcciones de 48 bits
  - page size: 4 kB ($2^{12}$)
  - tamaño de las PTE: 8 bytes
- **Problema:**
  - Requiere una tabla de paginación de 512 GB
    - $2^{48} \cdot 2^{-12} \cdot 2^{3} = 2^{39}$ bytes
- **Solución común: tablas multi nivel**
- **Ejemplo:**
  - Tabla Nivel 1 (siempre en memoria): cada PTE apunta a una tabla nivel 2
  - Tabla Nivel 2 (se la puede guardar en disco): cada PTE apunta a una página (como vimos hasta ahora)



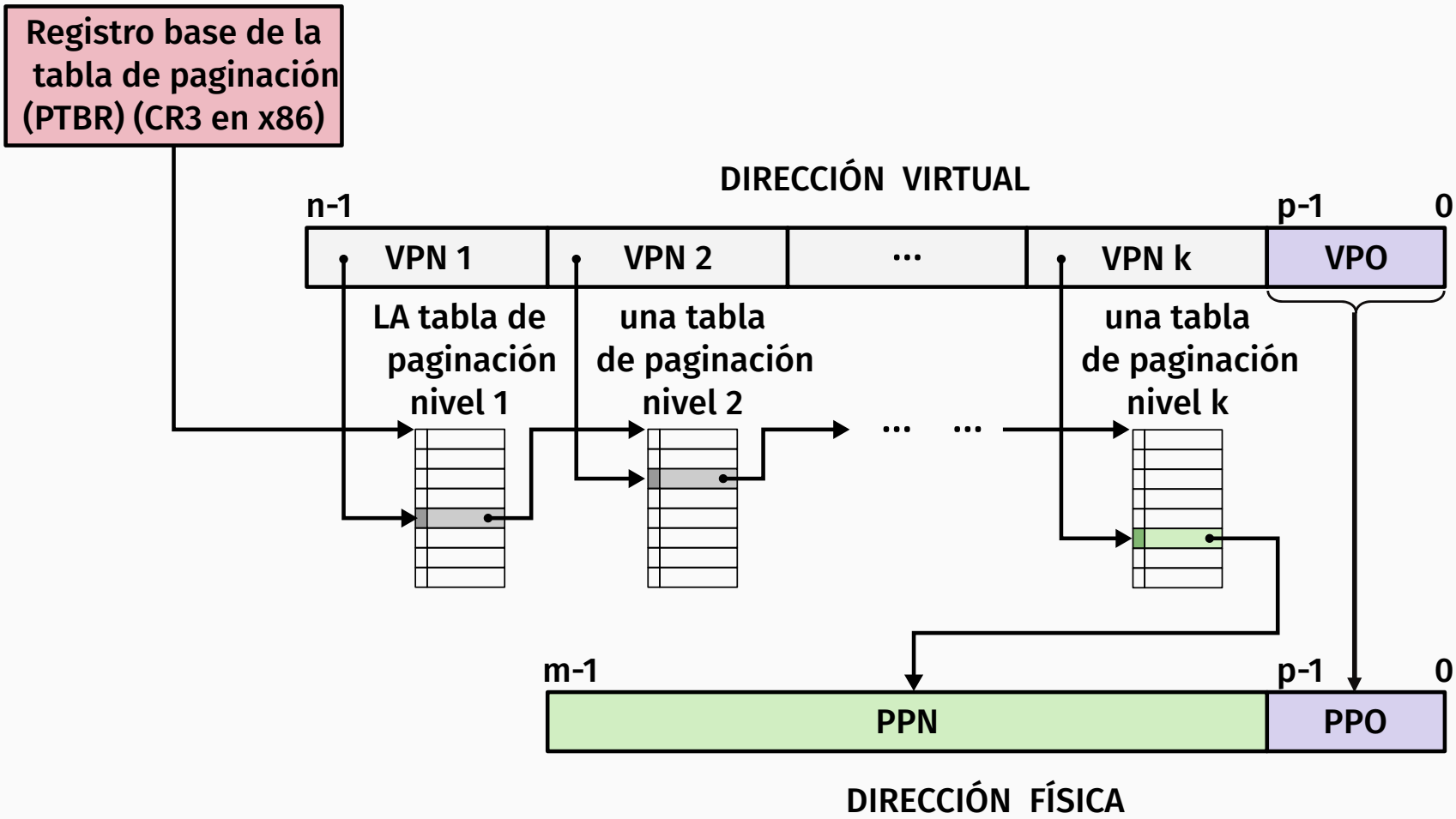
# Ejemplo: jerarquía de tabla de 2 niveles




**Direcciones de 64 bits, páginas de 8KB, PTEs de 8 bytes**

*Direcciones de 32 bits, páginas de 4KB, PTEs de 4 bytes*

# Traducciones con tablas de paginación de k niveles



# Resumen

- **Vista de desarrolladora de la memoria virtual:**
  - Cada proceso tiene su espacio de direcciones linea y privado
  - "No" puede ser corrupto por otro proceso
- **Vista del sistema de la memoria virtual:**
  - Usa la memoria de manera eficiente porque puede guardar en caché las páginas de memoria virtual
    - Eficiente sólo por la localidad; sin localidad, es malo
  - Simplifica la administración de la memoria y la programación
  - Simplifica la protección al agregar un punto de comprobación de permisos
- **Se implementa con una combinación de hardware y software**
  - MMU, TLB, y parte de la gestión de la excepción van por hardware
  - Gestión de la función de page fault, y gestión de la tabla va por software

# Ejercicio: memoria virtual

- **Datos: tabla de 2 niveles, páginas de 4 kB, tamaño de la memoria virtual y física 1 MB**

  Las direcciones, tanto física como virtual, son de 20 bits.

- **Si las tablas de paginación son iguales para cada nivel ¿cuántos bits se usan para cada parámetro/símbolo?**

  VPO
  VPN1
  VPN2
  PPO
  PPN

# Ejercicio: memoria virtual

- **Datos: tabla de 2 niveles, páginas de 4 kB, tamaño de la memoria virtual y física 1 MB**

  Las direcciones, tanto física como virtual, son de 20 bits.

- **Si las tablas de paginación son iguales para cada nivel ¿cuántos bits se usan para cada parámetro/símbolo?**

  | | |
  |---|---|
  | VPO | 12 bits |
  | VPN1 | 4 bits |
  | VPN2 | 4 bits |
  | PPO | 12 bits |
  | PPN | 8 bits |

# Ejercicio: memoria virtual

- **Datos: tabla de 2 niveles, páginas de 4 kB, tamaño de la memoria virtual y física 1 MB**

  Las direcciones, tanto física como virtual, son de 20 bits.

- **Si las tablas de paginación son iguales para cada nivel ¿cuántos bits se usan para cada parámetro/símbolo?**

  | | |
  |---|---|
  | VPO | 12 bits |
  | VPN1 | 4 bits |
  | VPN2 | 4 bits |
  | PPO | 12 bits |
  | PPN | 8 bits |

  ¿Tiene sentido virtualizar si los espacios de direcciones virtual y físico son del mismo tamaño?

  ¿Es un uso eficiente de tablas de paginación multi nivel?

# Ejercicio: memoria virtual

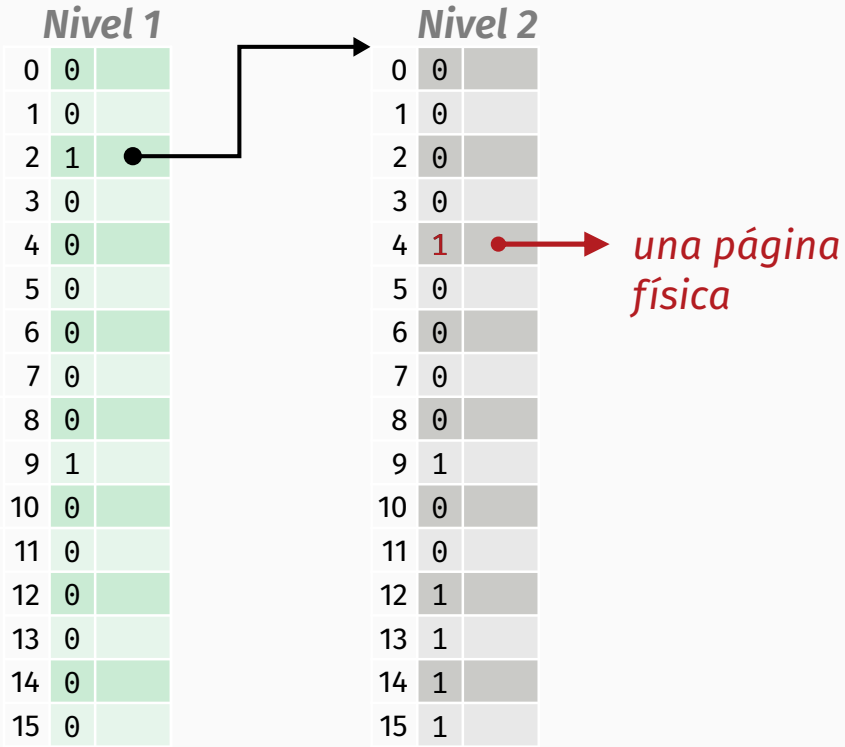
- **Datos: tabla de 2 niveles, páginas de 4 kB, tamaño de la memoria virtual y física 1 MB**

  Las direcciones, tanto física como virtual, son de 20 bits.

- **Para la siguiente tabla de paginación ¿qué pedidos son *page hits*, y cuáles son *page fault*?**

  `0x00000`
  `0x10A32`
  `0x15213`
  `0x20000`
  `0x2FFFF`
  `0x89999`
  `0x90210`

  ¿Cómo cambia la tabla si `malloc` reserva la página de la dirección `0x24000`?



# Ejercicio: memoria virtual

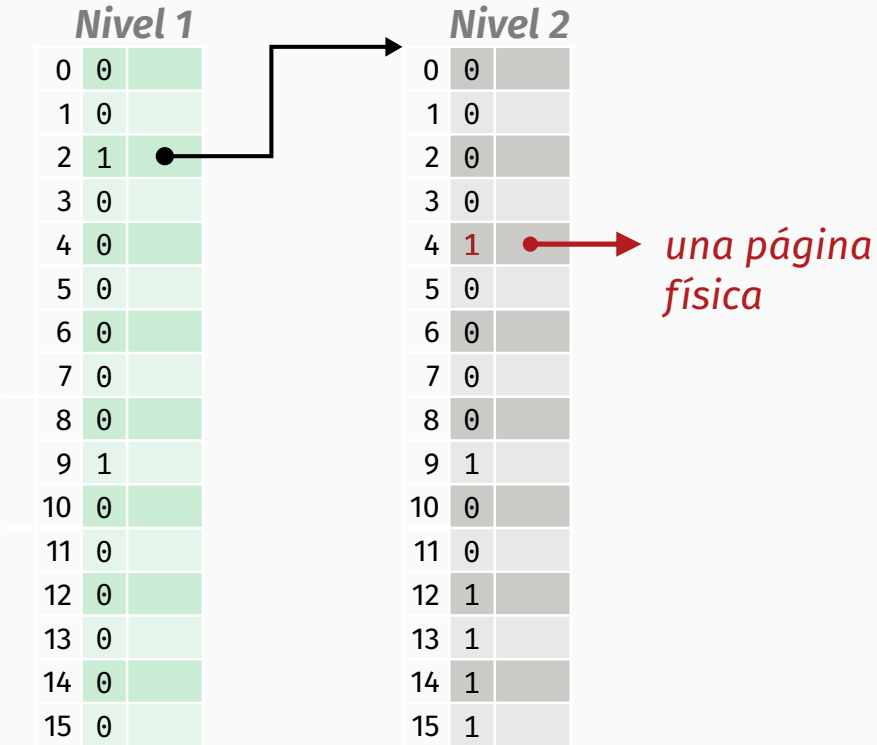
- **Datos: tabla de 2 niveles, páginas de 4 kB, tamaño de la memoria virtual y física 1 MB**

  Las direcciones, tanto física como virtual, son de 20 bits.

- **Para la siguiente tabla de paginación ¿qué pedidos son *page hits*, y cuáles son *page fault*?**

| | |
|---|---|
| `0x00000` | *page fault* |
| `0x10A32` | *page fault* |
| `0x15213` | *page fault* |
| `0x20000` | *page fault* |
| `0x2FFFF` | *page hit* |
| `0x89999` | *page fault* |
| `0x90210` | ?? |

¿Cómo cambia la tabla si `malloc` reserva la página de la dirección `0x24000`?



# Licencia del estilo de beamer

Obtén el código de este estilo y la presentación demo en

`github.com/pamoreno/mtheme`