# Administración de Memoria

## Sistemas Operativos

Gonzalo Guillamon

Departamento de Computación
Facultad de Ciencias Exactas y Naturales
Universidad de Buenos Aires

14 de septiembre de 2017

# Administración de memoria

- Administrador de Memoria (Memory Manager):

  Componente del sistema operativo encargado de administrar la memoria.

# Administración de memoria

- Administrador de Memoria (Memory Manager):

  Componente del sistema operativo encargado de administrar la memoria.

  ¿Más específico?

# Administración de memoria

- Administrador de Memoria (Memory Manager):

  Componente del sistema operativo encargado de administrar la memoria.

  ¿Más específico?

  Se encarga de (entre otras cosas):

  1. Asegurar la disponibilidad de memoria
  2. Asignar y liberar memoria.
  3. Organizar la memoria disponible.
  4. Asegurar la protección de la memoria.
  5. Permitir acceso a memoria compartida.

# 1. Asegurar la disponibilidad de memoria

- Memoria Virtual: Hacerle creer al proceso que dispone de más memoria de la que realmente tiene.

# 1. Asegurar la disponibilidad de memoria

- Memoria Virtual: Hacerle creer al proceso que dispone de más memoria de la que realmente tiene.

Entonces... ¿Cuánta memoria tiene mi sistema?

# 1. Asegurar la disponibilidad de memoria

- Memoria Virtual: Hacerle creer al proceso que dispone de más memoria de la que realmente tiene.

Entonces... ¿Cuánta memoria tiene mi sistema?

- Cantidad de bytes de memoria física = MEM_SIZE.
- Cantidad de celdas de memoria física = MEM_SIZE / DIR_UNIT.
- Cantidad de celdas de memoria virtual = $2^{DIR\_BITS}$.
- Cantidad de bytes de memoria virtual = $2^{DIR\_BITS}$*DIR_UNIT.

# 1. Asegurar la disponibilidad de memoria

- Memoria Virtual: Hacerle creer al proceso que dispone de más memoria de la que realmente tiene.

Entonces... ¿Cuánta memoria tiene mi sistema?

- Cantidad de bytes de memoria física = MEM_SIZE.
- Cantidad de celdas de memoria física = MEM_SIZE / DIR_UNIT.
- Cantidad de celdas de memoria virtual = $2^{DIR\_BITS}$.
- Cantidad de bytes de memoria virtual = $2^{DIR\_BITS}$*DIR_UNIT.

Ejercicios:

# 1. Asegurar la disponibilidad de memoria

- Memoria Virtual: Hacerle creer al proceso que dispone de más memoria de la que realmente tiene.

Entonces... ¿Cuánta memoria tiene mi sistema?

- Cantidad de bytes de memoria física = MEM_SIZE.
- Cantidad de celdas de memoria física = MEM_SIZE / DIR_UNIT.
- Cantidad de celdas de memoria virtual = $2^{DIR\_BITS}$.
- Cantidad de bytes de memoria virtual = $2^{DIR\_BITS}$*DIR_UNIT.

Ejercicios:

- Tengo direcciones de 16 bits. ¿Cuánta memoria virtual puedo tener direccionando a byte? ¿Cuánta física?

# 1. Asegurar la disponibilidad de memoria

- Memoria Virtual: Hacerle creer al proceso que dispone de más memoria de la que realmente tiene.

Entonces... ¿Cuánta memoria tiene mi sistema?

- Cantidad de bytes de memoria física = MEM_SIZE.
- Cantidad de celdas de memoria física = MEM_SIZE / DIR_UNIT.
- Cantidad de celdas de memoria virtual = $2^{DIR\_BITS}$.
- Cantidad de bytes de memoria virtual = $2^{DIR\_BITS}$*DIR_UNIT.

Ejercicios:

- Tengo direcciones de 16 bits. ¿Cuánta memoria virtual puedo tener direccionando a byte? ¿Cuánta física?
- Tengo 65536 ($2^{16}$) bytes de memoria física dividia en celdas de 16 bits. ¿Cuántos bits necesito para direccionar?

# 2. Asignar y liberar memoria

- ~~Alocar~~ Asignar = reservar una porción de memoria para un proceso.
  - La porción de memoria pasa a estar ocupada por el proceso que la solicitó.
  - Tenemos que saber quién es el dueño de esa porción de memoria.
- Liberar = un porción de la memoria vuelve a estar disponible para cualquier proceso.

# 2. Asignar y liberar memoria

- ~~Alocar~~ Asignar = reservar una porción de memoria para un proceso.
  - La porción de memoria pasa a estar ocupada por el proceso que la solicitó.
  - Tenemos que saber quién es el dueño de esa porción de memoria.
- Liberar = un porción de la memoria vuelve a estar disponible para cualquier proceso.

Cuando hablamos de *memoria*, ¿nos referimos a memoria virtual o física?

# 3. Organizar la memoria disponible

Pensemos en la memoria como una colección de celdas.

El MM debe elegir qué porción de memoria (qué celdas) asignar.

# 3. Organizar la memoria disponible

Pensemos en la memoria como una colección de celdas.

El MM debe elegir qué porción de memoria (qué celdas) asignar.

- ¿Cómo agrupamos la memoria?
  - De a celdas.
  - En Bloques de tamaño fijo.
  - En Bloques de tamaño variable.

# 3. Organizar la memoria disponible

Pensemos en la memoria como una colección de celdas.

El MM debe elegir qué porción de memoria (qué celdas) asignar.

- ¿Cómo agrupamos la memoria?
  - De a celdas.
  - En Bloques de tamaño fijo.
  - En Bloques de tamaño variable.
- ¿Cómo organizamos la memoria libre?
  - Con un mapa de bits.
  - Con una lista enlazada.

# 3. Organizar la memoria disponible

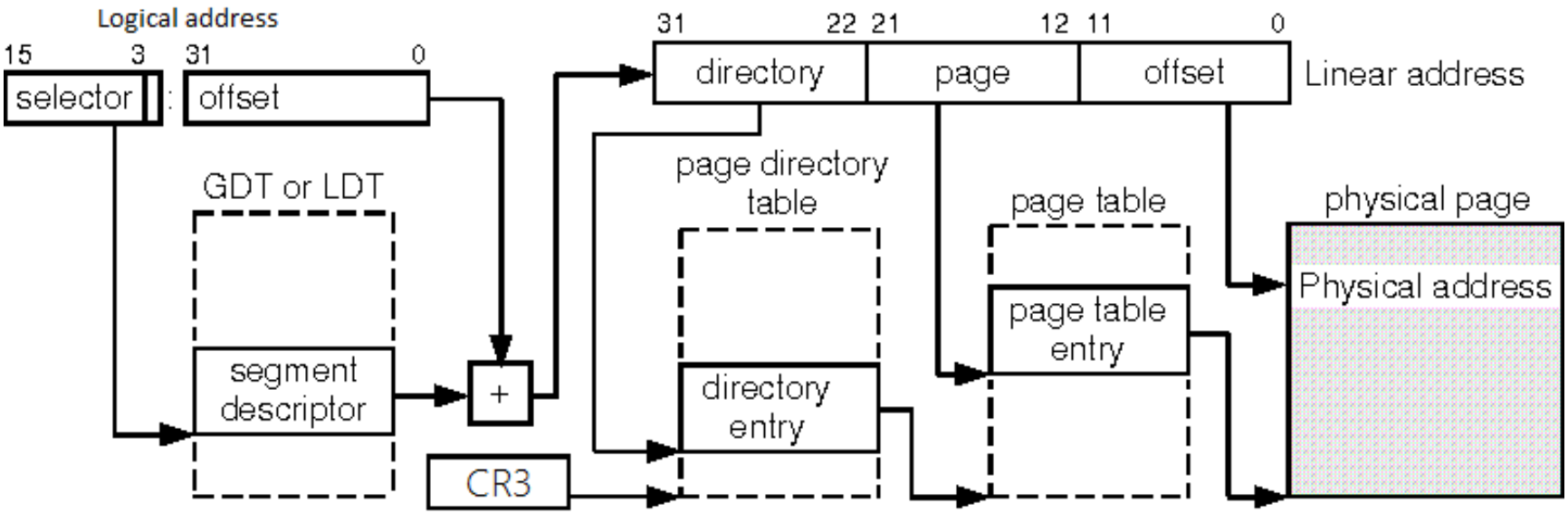
Pensemos en la memoria como una colección de celdas.

El MM debe elegir qué porción de memoria (qué celdas) asignar.

- ¿Cómo agrupamos la memoria?
  - De a celdas.
  - En Bloques de tamaño fijo.
  - En Bloques de tamaño variable.
- ¿Cómo organizamos la memoria libre?
  - Con un mapa de bits.
  - Con una lista enlazada.
- Mecanismos más sofisticados:
  - Segmentación.
  - Paginación.
  - Segmentación + paginación.

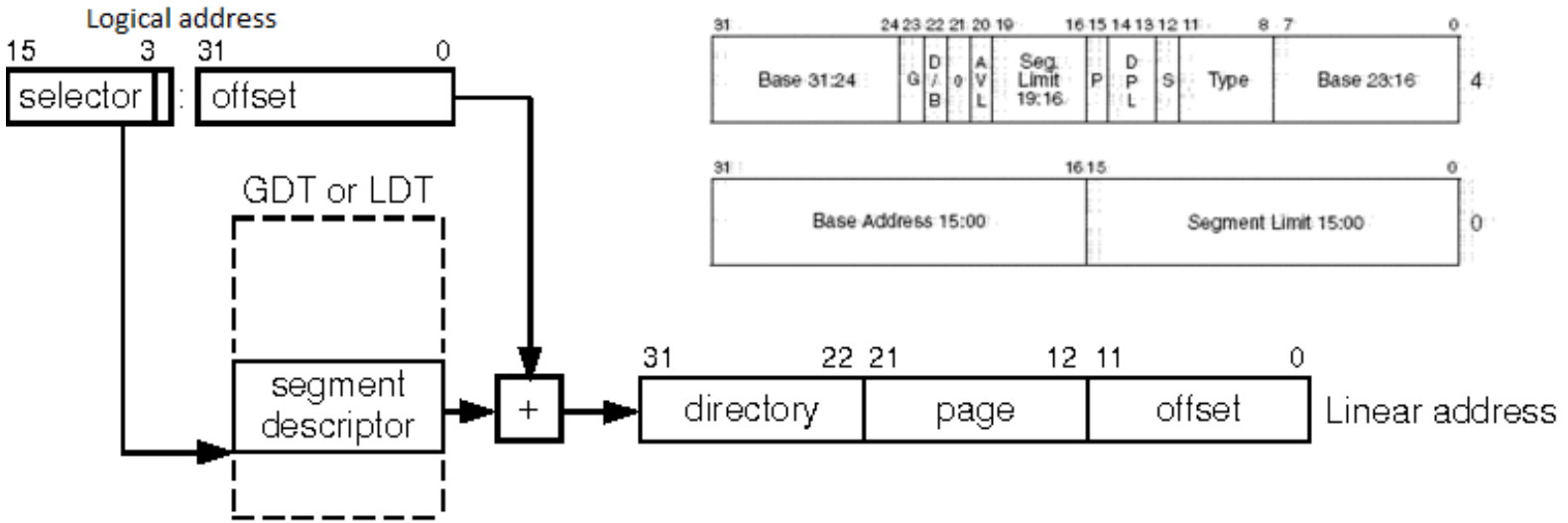
# Repaso de Orga II

- MMU
  - Unidad de Gestión de Memoria (Memory Management Unit)
  - Permite traducir direcciones virtuales a físicas.
- Memoria virtual!



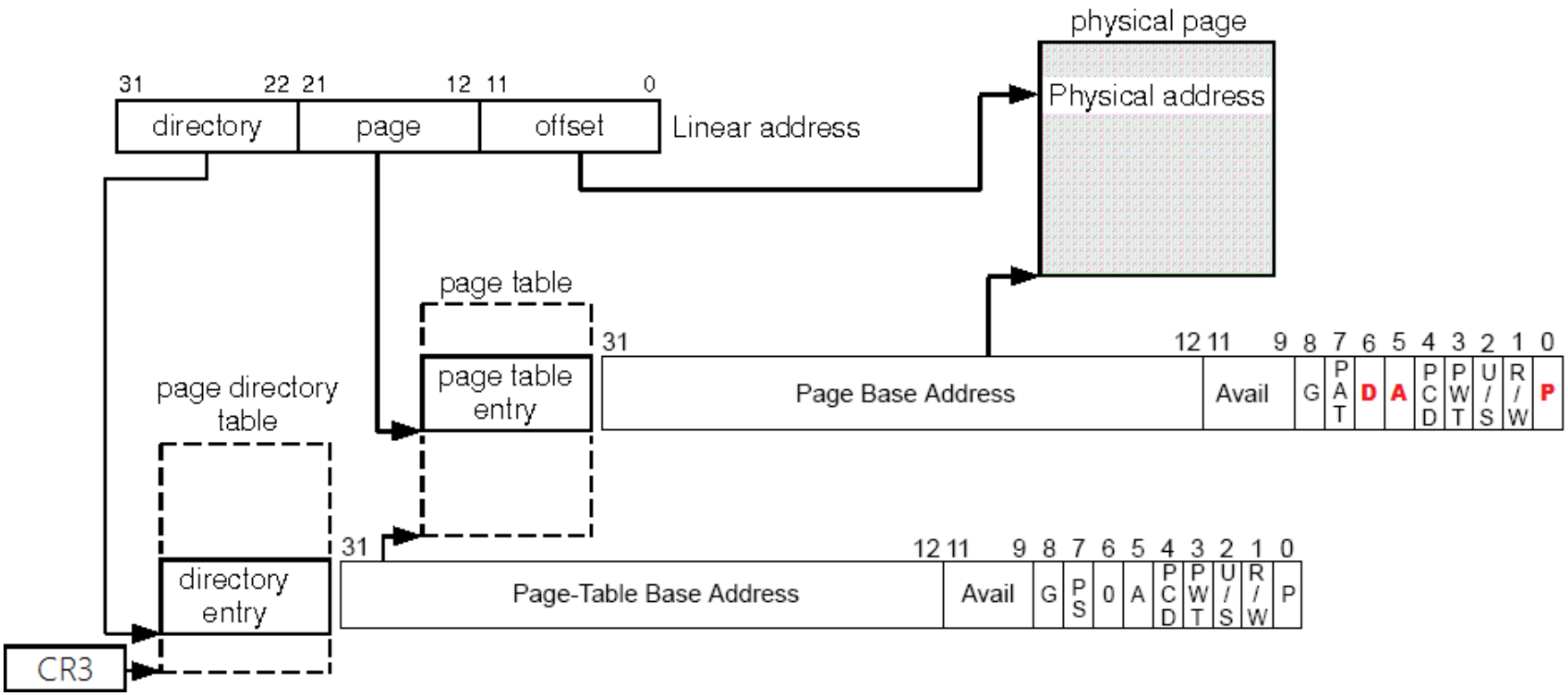
# Segmentación

- Segmentación = Separar la memoria en segmentos.
- Un segmento generalmente se define con una base (donde empieza) y un límite (hasta donde llega).
- Se acceden mediante **direcciones lógicas**.
- Primeros bits de la dirección indexan el **Descriptor de Segmento** en la tabla de descriptores.
- El resto es el offset dentro del segmento.
- Los descriptores almacenan información sobre los segmentos (inicio en memoria física, tamaño, atributos, ...).



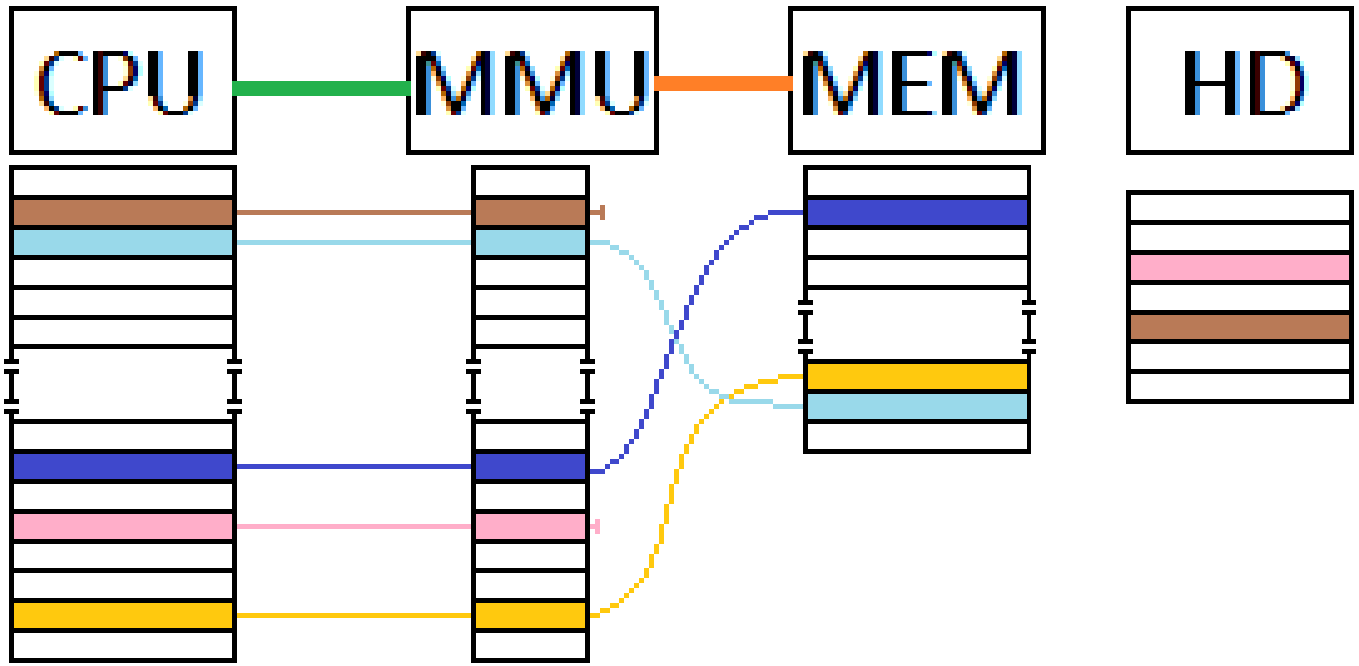
# Paginación

- Paginación = Dividir la memoria en páginas (muchas páginas, no necesariamente contiguas).
- Los conjuntos de bits indexan en las sucesivas tablas de páginas. Los últimos son el offset en la página.



# Direcciones lógicas vs direcciones físicas

- Memoria física: Una celda dentro de la memoria del sistema. El tamaño está determinado por el hardware.
  - Un **marco de página** es una porción de memoria física.
- Memoria virtual: Una representación de la información almacenada. El tamaño depende de la unidad de direccionamiento y la cantidad de bits de direccionamiento.
  - Una **página** es una porción de memoria virtual.



# Más sobre paginación

- Paginación nos permite mapear mucha más memoria de la que realmente tiene el sistema, Si una página no está cargada en ningún marco de página, el MM se encarga de ir a buscarla al disco y cargarla en memoria.

# Más sobre paginación

- Paginación nos permite mapear mucha más memoria de la que realmente tiene el sistema, Si una página no está cargada en ningún marco de página, el MM se encarga de ir a buscarla al disco y cargarla en memoria.

  Pero... ¿Qué pasa si no hay lugar en la memoria? (por lo general, MEM_VIRTUAL $>$ MEM_FISICA).

# Más sobre paginación

- Paginación nos permite mapear mucha más memoria de la que realmente tiene el sistema, Si una página no está cargada en ningún marco de página, el MM se encarga de ir a buscarla al disco y cargarla en memoria.

  Pero... ¿Qué pasa si no hay lugar en la memoria? (por lo general, MEM_VIRTUAL $>$ MEM_FISICA).
- Algoritmos de remoción:
  - FIFO
  - LRU
  - Segunda oportunidad
  - Not Recently Used

# Más sobre paginación

- Paginación nos permite mapear mucha más memoria de la que realmente tiene el sistema, Si una página no está cargada en ningún marco de página, el MM se encarga de ir a buscarla al disco y cargarla en memoria.

  Pero... ¿Qué pasa si no hay lugar en la memoria? (por lo general, MEM_VIRTUAL $>$ MEM_FISICA).
- Algoritmos de remoción:
  - FIFO
    La clásica de siempre.
  - LRU
    Desalojo la página que hace más tiempo que no se usa.
  - Segunda oportunidad
    Si fue referenciada, le doy otra chance.
  - Not Recently Used
    Primero desalojo las ni referenciadas ni modificadas. Después las referenciadas y por último las modificadas.

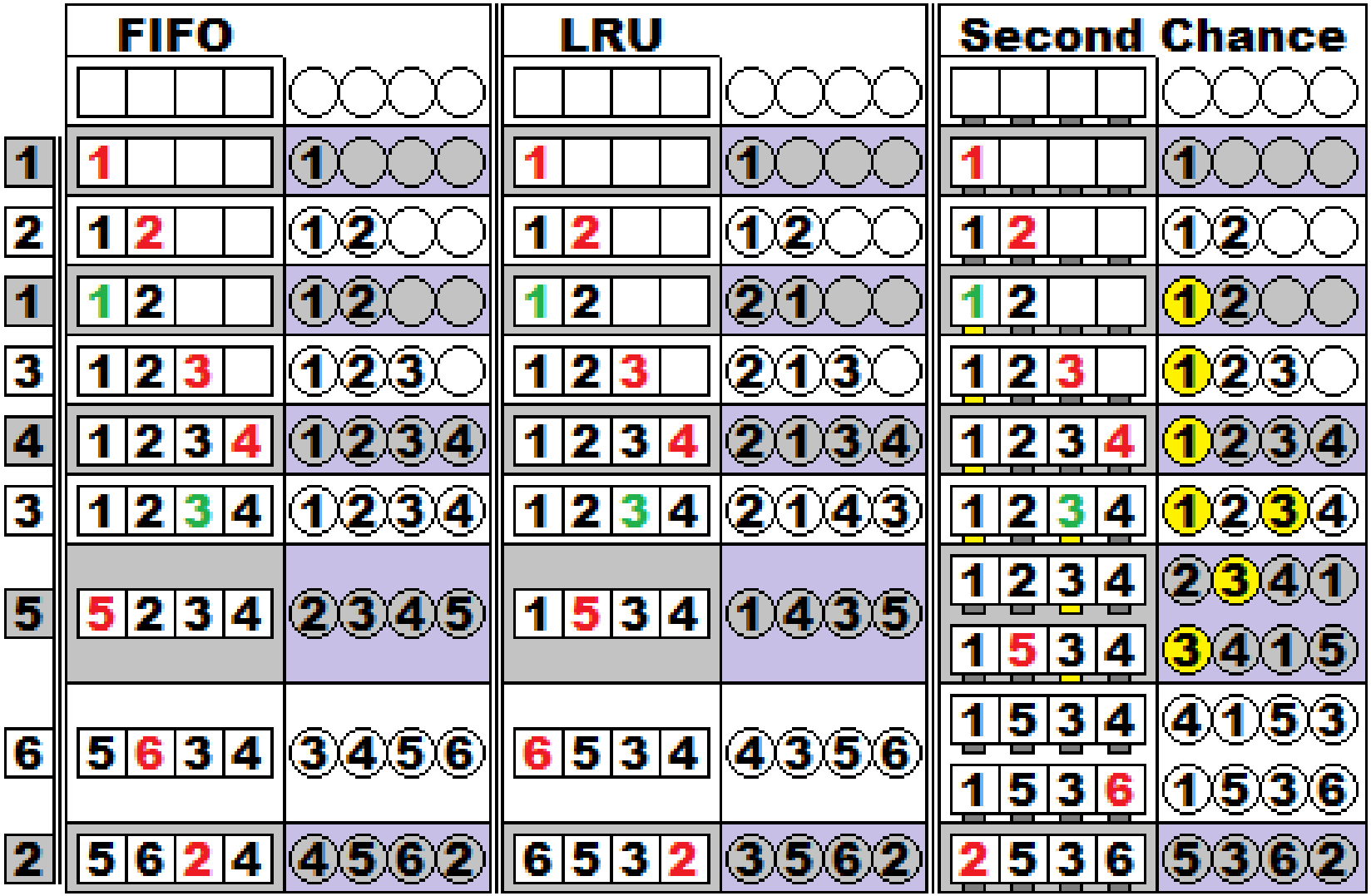
# Ejercicio: Algoritmos de remoción

- Tengo un sistema con 6 páginas y sólo 4 marcos de página. La memoria comienza vacía.

  Llegan los siguientes pedidos de memoria (número de página) en ese orden:

  1, 2, 1, 3, 4, 3, 5, 6, 2
- Indique qué página se desaloja tras cada pedido utilizando los algoritmos FIFO, LRU y Second Chance y calcule el *hit-rate* en cada caso.
- Hit-Rate= Páginas qué pedí y ya estaban cargadas en memoria/páginas totales pedidas.

# Solución



# Solución

- Hit-Rate (FIFO) = 2 / 9
- Hit-Rate (LRU) = 2 / 9
- Hit-Rate (SC) = 2 / 9

# 4. Asegurar la protección de la memoria

Un proceso no debería poder usar memoria que no reservó.

# 4. Asegurar la protección de la memoria

Un proceso no debería poder usar memoria que no reservó.

- Paginación provee una solución para esto:

  Cada proceso tiene su propia tabla de páginas.

# 4. Asegurar la protección de la memoria

Un proceso no debería poder usar memoria que no reservó.

- Paginación provee una solución para esto:

  Cada proceso tiene su propia tabla de páginas.
- De forma similar, puede utilizarse segmentación para lo mismo:

  Cada proceso tiene su propia tabla de segmentos.

# 4. Asegurar la protección de la memoria

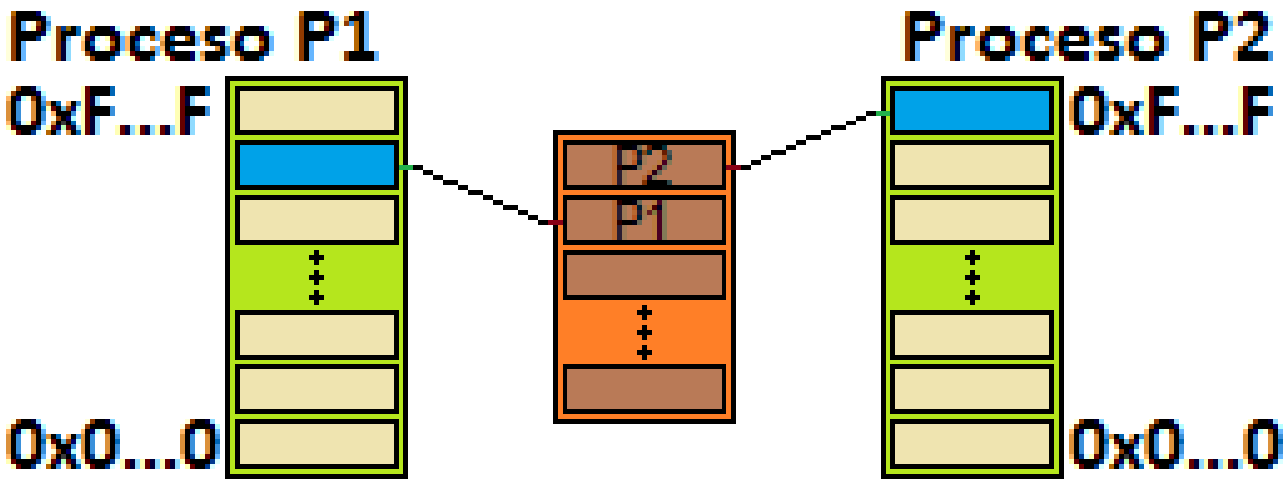
Un proceso no debería poder usar memoria que no reservó.

- Paginación provee una solución para esto:

  Cada proceso tiene su propia tabla de páginas.

- De forma similar, puede utilizarse segmentación para lo mismo:

  Cada proceso tiene su propia tabla de segmentos.



# 5. Permitir acceso a memoria compartida

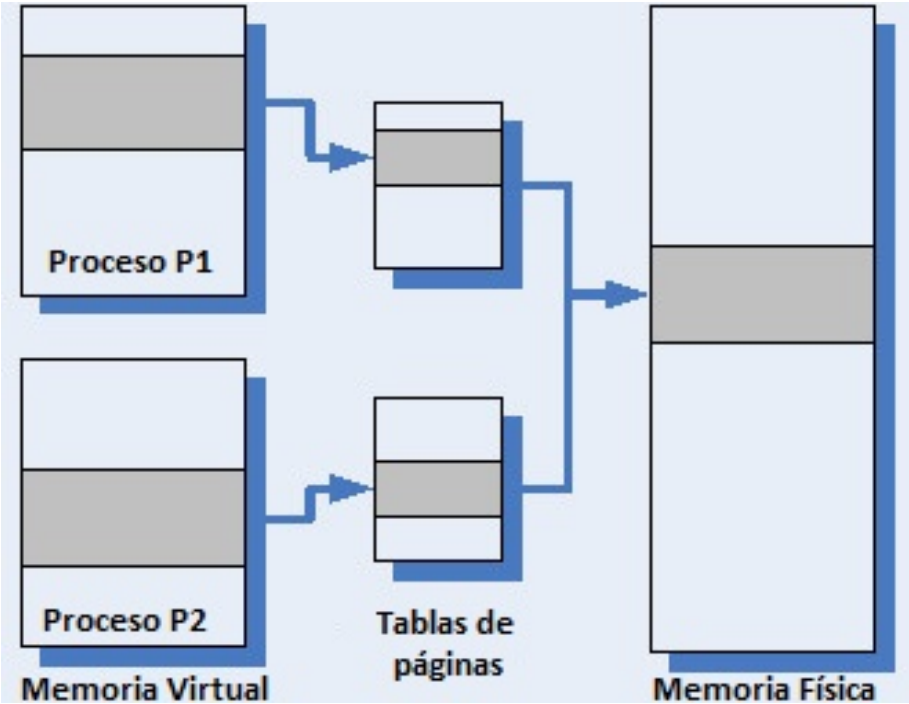
Queremos que dos procesos lean y escriban sobre una misma variable.

# 5. Permitir acceso a memoria compartida

Queremos que dos procesos lean y escriban sobre una misma variable.

- Una vez más, paginación es la respuesta.

  Podemos mapear dos páginas al mismo marco de página.



# ¿Qué es `malloc`?

# ¿Qué es `malloc`?

Opciones:

# ¿Qué es `malloc`?

Opciones:

- Lo contrario de `buennoc` (HINT: no).

# ¿Qué es `malloc`?

Opciones:

- Lo contrario de `buennoc` (HINT: no).
- Una syscall.

# ¿Qué es `malloc`?

Opciones:

- Lo contrario de `buennoc` (HINT: no).
- Una syscall.
- ~~Un alocador de memoria~~.

# ¿Qué es `malloc`?

Opciones:

- Lo contrario de `buennoc` (HINT: no).
- Una syscall.
- ~~Un alocador de memoria~~.
- Una función provista por una lib.

# ¿Qué es `malloc`?

Opciones:

- Lo contrario de `buennoc` (HINT: no).
- Una syscall.
- ~~Un alocador de memoria~~.
- Una función provista por una lib.

Más en las man-pages...

- man 2 syscalls (`http://goo.gl/aOwOJk`)
- man malloc (`http://goo.gl/OAoxAs`)

# ¿Qué es `malloc`?

Opciones:

- Lo contrario de `buennoc` (HINT: no).
- Una syscall.
- ~~Un alocador de memoria~~.
- Una función provista por una lib.

Más en las man-pages...

- man 2 syscalls (`http://goo.gl/aOwOJk`)
- man malloc (`http://goo.gl/OAoxAs`)

Entonces..

# ¿Qué es `malloc`?

Opciones:

- Lo contrario de `buennoc` (HINT: no).
- Una syscall.
- ~~Un alocador de memoria~~.
- Una función provista por una lib.

Más en las man-pages...

- man 2 syscalls (`http://goo.gl/aOwOJk`)
- man malloc (`http://goo.gl/OAoxAs`)

Entonces..

- `malloc` no es syscall sino un "memory manager" provisto por la lib-C. Ergo, `malloc` vive y se ejecuta en el espacio de usuario.

# ¿Qué es `malloc`?

Opciones:

- Lo contrario de `buennoc` (HINT: no).
- Una syscall.
- ~~Un alocador de memoria~~.
- Una función provista por una lib.

Más en las man-pages...

- man 2 syscalls (`http://goo.gl/aOwOJk`)
- man malloc (`http://goo.gl/OAoxAs`)

Entonces..

- `malloc` no es syscall sino un "memory manager" provisto por la lib-C. Ergo, `malloc` vive y se ejecuta en el espacio de usuario.
- Pertenece a una familia de funciones integrada por `malloc`, *`calloc`*, *`realloc`* y *`free`*.
- El standard de C sólo define el comportamiento de estas funciones. Cómo se implementan depende de cada implementación y del SO

# `malloc` en Linux

En Linux, la implementación más común de malloc lo que hace es pedir memoria al kernel por medio de:

- `brk()` y `sbrk()`: cambian el límite del heap del proceso
- `mmap()`: mapea una porción grande de memoria (una página) en el espacio de direcciones del proceso.

# ¿Cómo pide más memoria `malloc`? (1)

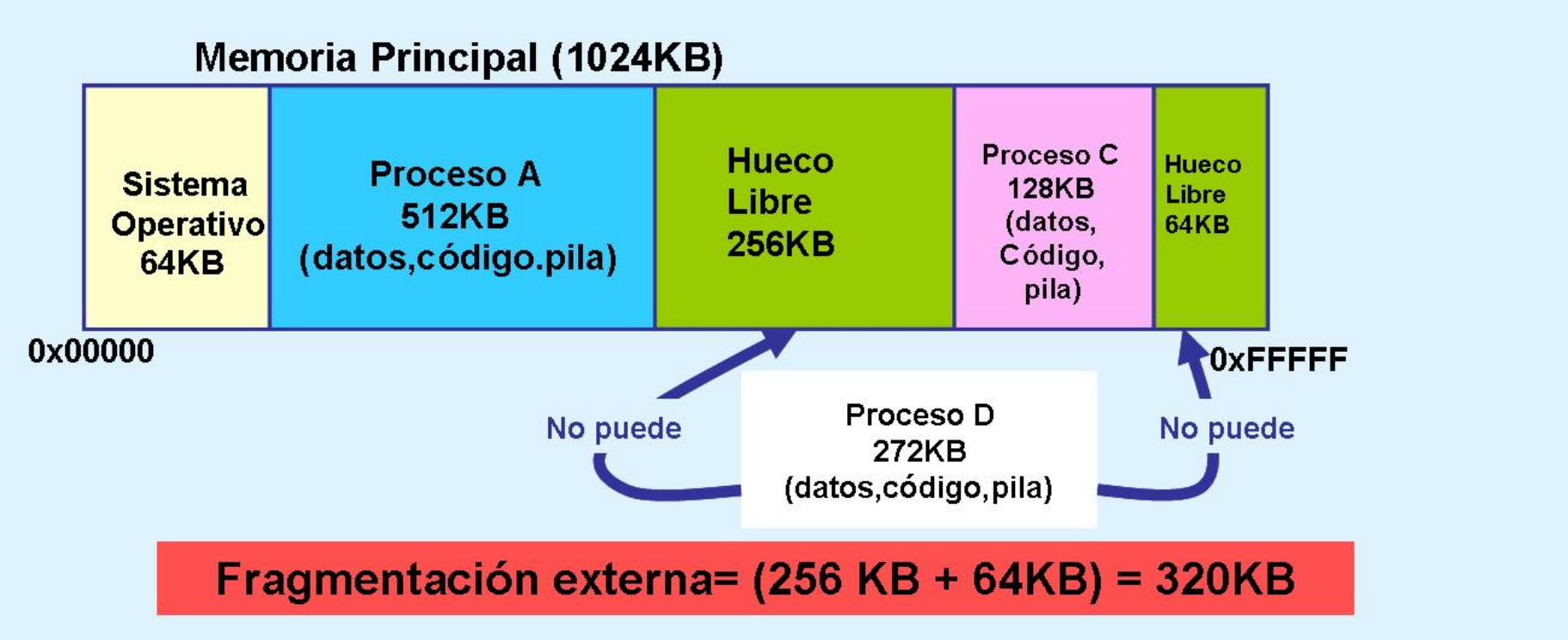
Frente a un nuevo pedido de memoria `malloc` hace lo siguiente:

1. `malloc` busca entre las porciones libres de memoria del *Heap* uno o más buckets contiguos que puedan satisfacer el tamaño pedido de memoria.
2. Si eso falla, entonces o bien ocupamos toda la memoria del *Heap*, o bien (lo más probable) la distribución de memoria fue tal que el *Heap* quedó fragmentado y no hay un espacio de memoria contiguo que podamos ofrecer. En cualquier caso, `malloc` intentará extender el *Heap* del proceso a través `brk()`/`sbrk()` (o de mmap() en algunos casos).

# ¿Cómo pide más memoria `malloc`? (2)

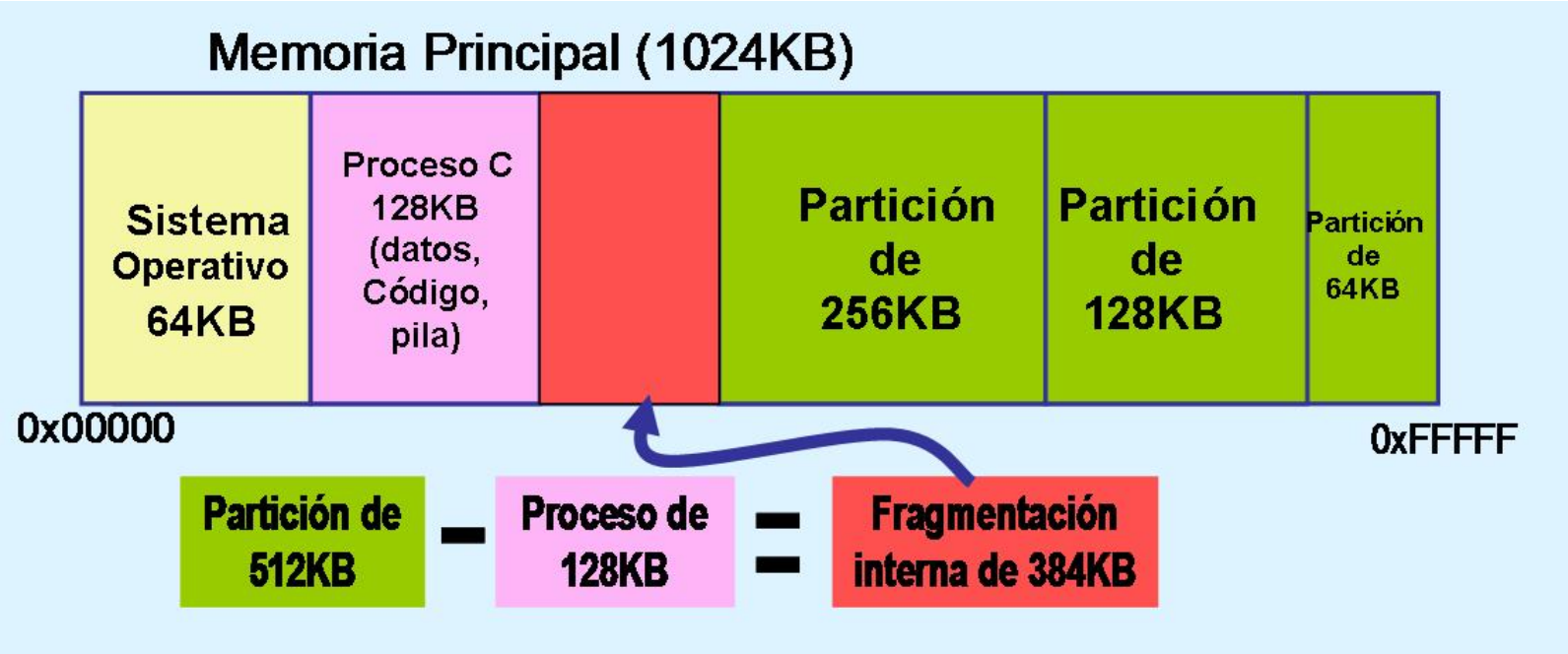
3. La llamadas `brk()`/`sbrk()` al Kernel ajustarán, dentro del *mm_struct* del proceso, el tope del *Heap* para que éste sea más extenso. Esto no aumenta la memoria directamente, ya que no se está mapeando memoria física al proceso, sino que lo que se hace es ampliar el espacio de direcciones reservado para el *Heap*.
4. Finalmente, cuando alguna de esas posiciones de memorias no mapeadas es accedida (generalmente por una lectura/escritura de la implementación de `malloc`) se produce una excepción de *Page Fault*. Dicha excepción será atrapada por el Kernel y producirá una invocación al administrador de páginas para obtener una nueva página de memoria física para el frame que generó la excepción.

# Fragmentación Externa




Bloques *pequeños* de memoria no contiguos.

# Fragmentación Interna




Memoria desperdiciada dentro de una partición (un bloque o página).

# Asignacion de memoria

¿Qué porción de memoria me conviene asignar?

# Asignacion de memoria

¿Qué porción de memoria me conviene asignar?

- ¿Cómo agrupamos la memoria?
  - De a celdas.
  - En Bloques de tamaño fijo.
  - En Bloques de tamaño variable.
- ¿Cómo organizamos la memoria libre?
  - Con un mapa de bits.
  - Con una lista enlazada.

# Asignacion de memoria

¿Qué porción de memoria me conviene asignar?

¿Cómo agrupamos la memoria?

De a celdas.

En Bloques de tamaño fijo.

En Bloques de tamaño variable.

¿Cómo organizamos la memoria libre?

Con un mapa de bits.

Con una lista enlazada.

- Algoritmos de elección de bloque libre:
  - First fit
  - Best fit
  - Worst fit
  - Quick fit

# Asignacion de memoria

¿Qué porción de memoria me conviene asignar?

¿Cómo agrupamos la memoria?

De a celdas.

En Bloques de tamaño fijo.

En Bloques de tamaño variable.

¿Cómo organizamos la memoria libre?

Con un mapa de bits.

Con una lista enlazada.

- Algoritmos de elección de bloque libre:
  - First fit
    La primera sección de memoria contigua del tamaño necesario.
  - Best fit
    De todas las secciones de tamaño mayor o igual al tamaño necesario, tomo la más chica.
  - Worst fit
    Mejor tomo la más grande.
  - Quick fit
    Además me guardo listas de bloques de determinados tamaños.

# Ejercicio: Algoritmos de elección de bloque libre

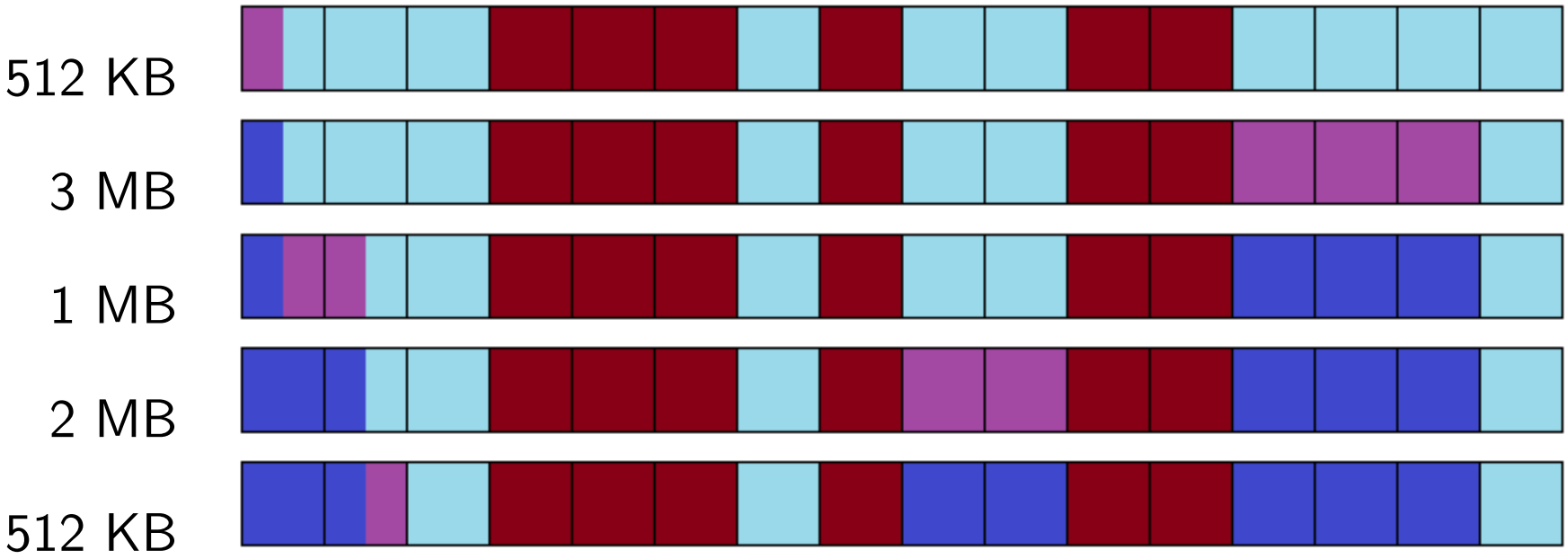
- Tengo un sistema con 16 MB de memoria **sin particionar** que direcciona a byte. El estado actual de la memoria es el siguiente (cuadrado= 1MB):

Llegan los siguientes pedidos de memoria en ese orden:

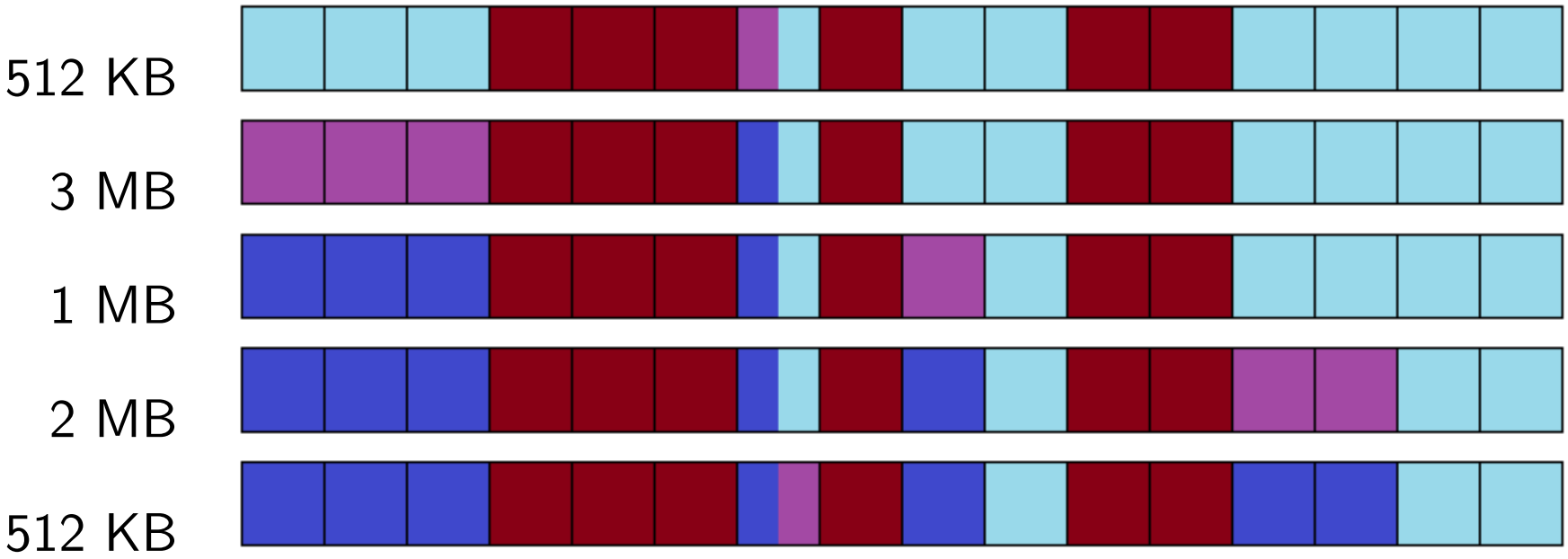
512 KB, 3 MB, 1 MB, 2MB, 512 KB.

- Indique qué bloques se asignan para cada pedido utilizando first-fit, best-fit y worst-fit.

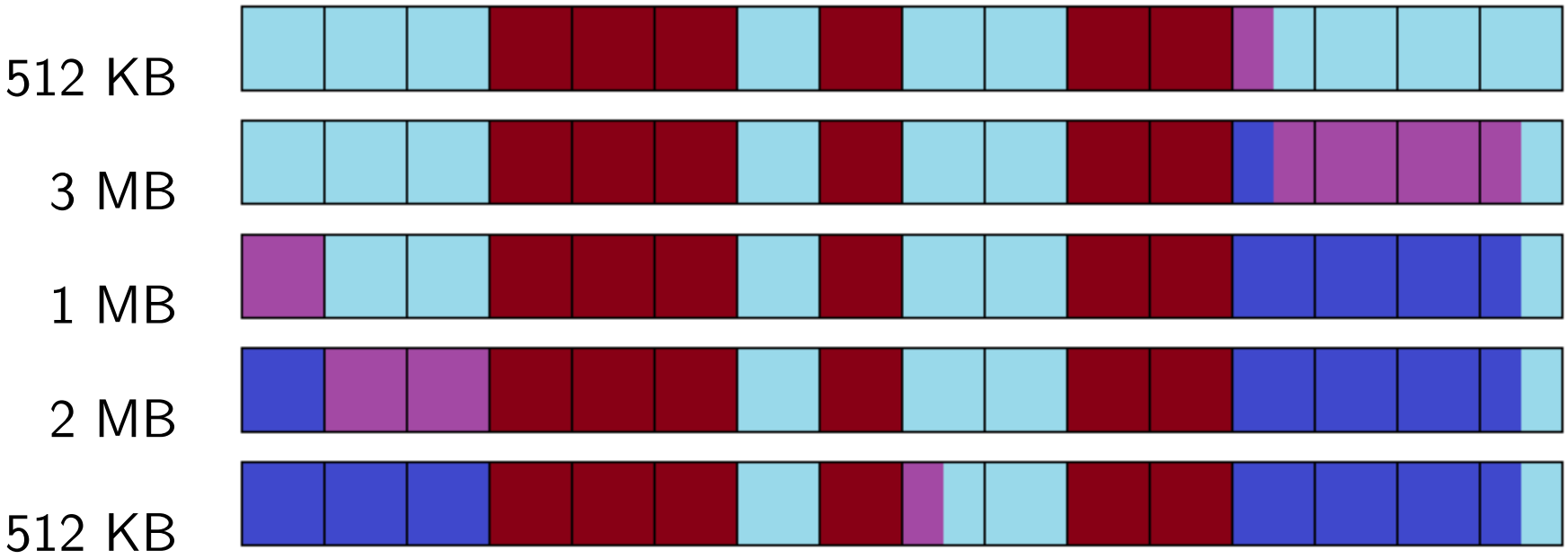
# Solución First-Fit



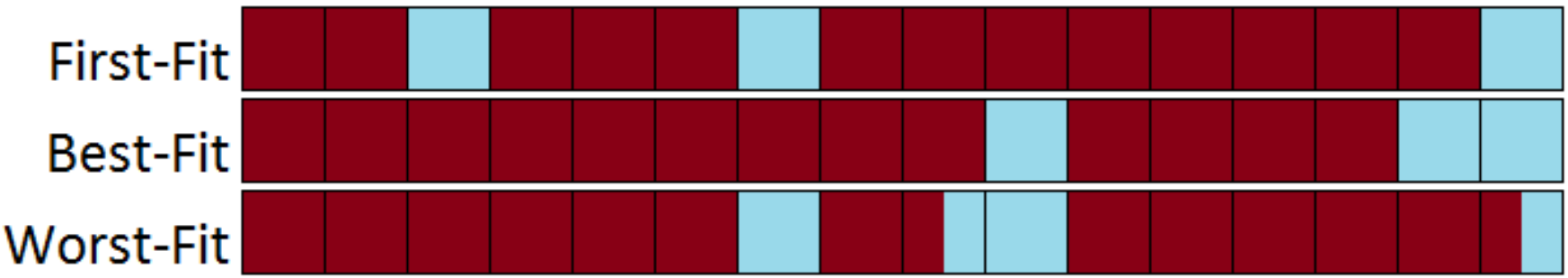
# Solución Best-Fit



# Solución Worst-Fit



# Entonces, ¿Cuál es mejor?



# Page Fault

- ¿Qué pasa cuando una página no está cargada en la memoria?

# Page Fault

- ¿Qué pasa cuando una página no está cargada en la memoria?

1. Un proceso accede a una dirección (virtual) de memoria.
2. La MMU traduce la dirección virtual a dirección física (accede a la entrada en la última tabla de páginas).
3. Lee el atributo correspondiente a presencia en la memoria.
4. Si es negativo, se produce la interrupción Page Fault.
5. Se ejecuta la **RAI** correspondiente.
6. Si la memoria está llena, se ejecuta el algoritmo de remoción.
7. Si la página que se va a desalojar fue modificada, hay que bajarla al disco.
8. Se carga en el lugar liberado la página solicitada.
9. Se vuelve a ejecutar la instrucción del proceso que accede a la dirección solicitada.

# Thrashing

- Situación en la que el SO pasa más tiempo cargando páginas que ejecutando procesos.

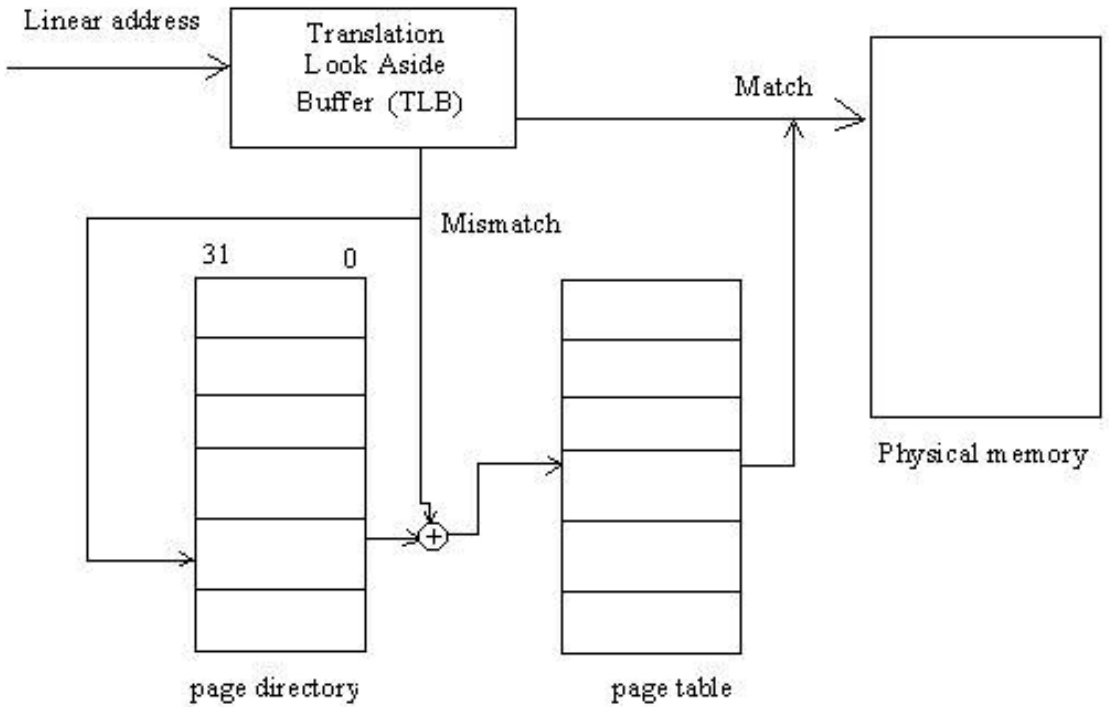
  Supongan que tenemos 2 procesos y un solo marco de página disponible.

  Cada proceso usa una sóla página, pero cada vez que ejecute, su página no va a estar cargada.
- Siempre va a estar cargada la página del otro.

# TLB (Translation Lookaside Buffer)

Buffer de Traducción Adelantada.

- Es una caché que guarda 'traducciones'.
- Paginación de 4 niveles: Cuatro accesos a memoria (1 por cada tabla) más uno para leer la página.




- ¿Entonces para qué me guardo todas las tablas?

# Copy-on-Write

- Dijimos que al crear un nuevo proceso se duplica toda su memoria.

  Sabemos que, en general después de un fork() viene un exec().

  exec() inutiliza todas las páginas de memoria, entonces ¿para qué nos gastamos en duplicar todo?
- Copy-on-Write: Sólo duplico (copy) cuando alguno de los procesos escribe (write).

# FIN

¿Preguntas?