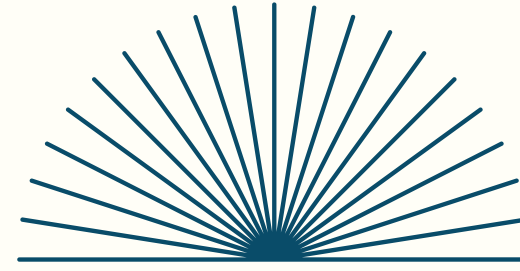




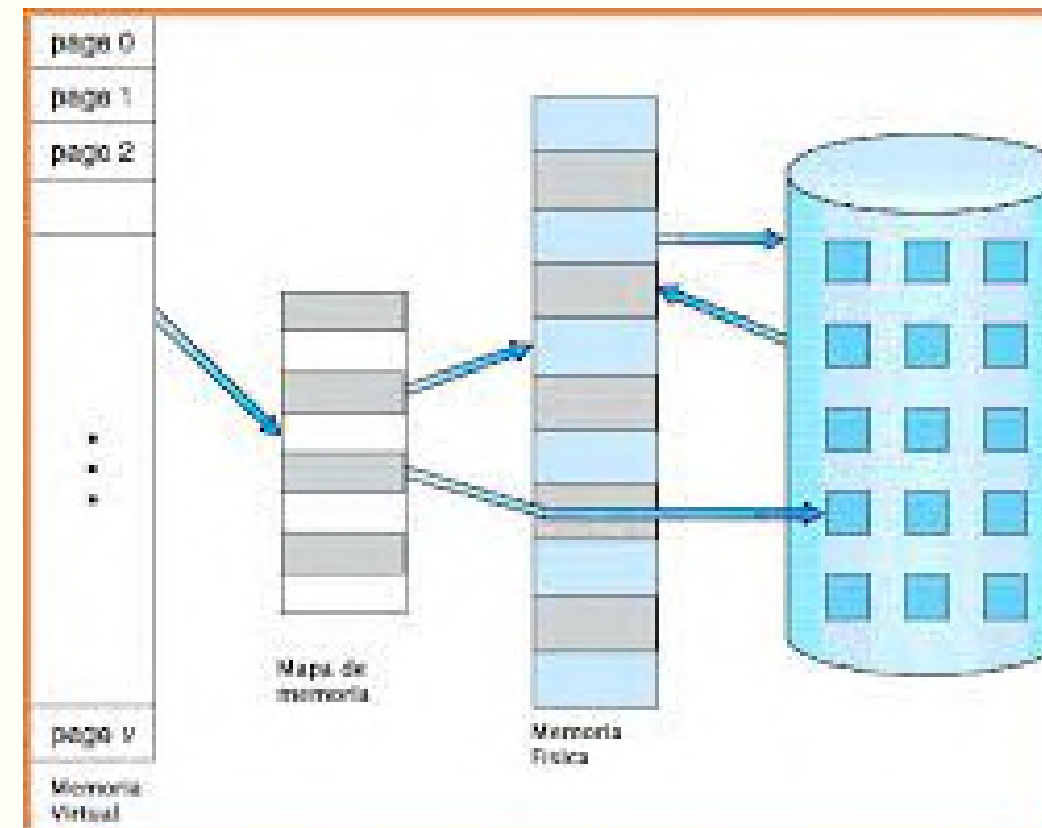
# MEMORIA VIRTUAL

*EQUIPO 8 - SISTEMAS OPERATIVOS (4CM4)*





# SUSTITUCIÓN DE PÁGINAS



# OPTIMIZACIÓN DE MEMORIA

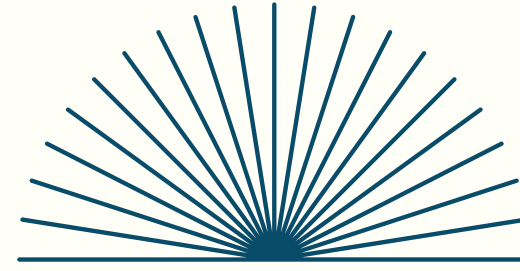
- Si un procesos de 10 paginas solo utiliza la mitad, se ahorra en operaciones E/S. Como solo se usan 5 páginas, se puede ejecutar el doble de procesos.
- Si cada proceso requiere 10 marcos y se tienen 40 marcos, se podrían ejecutar 8 procesos en lugar de 4.
- Esta optimización de memoria también tiene un posible error que es la sobreasignación de memoria.

# Sobreasignación

Al sobre asignar, por ejemplo teniendo la ejecución de 6 procesos donde solo se utilicen 5 paginas como en el caso anterior, quedarían 10 paginas libres.

Pero en el caso de que esos procesos intenten usar sus 10 paginas, entonces harían falta 60 marcos teniendo solo 40 disponibles.

Esta sobreasignación de memoria se dará en el momento en que al ejecutarse un proceso y hallar un fallo de página por no haber marcos libres, debido a que toda la memoria esta siendo usada.



# SUSTITUCIÓN BÁSICA DE PÁGINAS

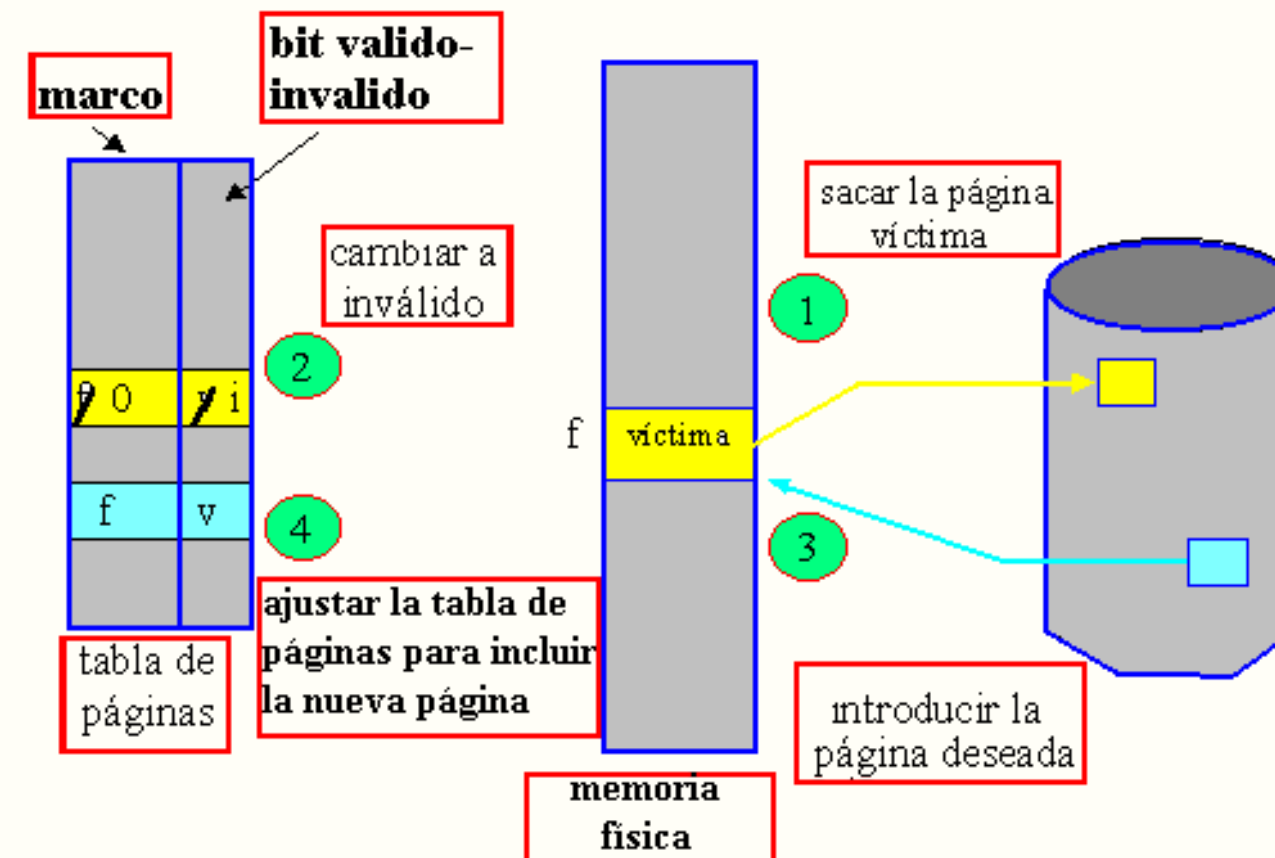


Figura 7.3. Reemplazo de página

# FUNCIONAMIENTO

- Si no hay marcos libres, se localiza uno que no este siendo usado y se libera.
  - El contenido del marco va a la zona de intercambio y se modifican las tablas de paginas para indicar que ya no esta en memoria, de tal forma que el marco se uso para la página que provoco el fallo en el proceso.
1. Hallar la ubicación de la página en el disco.
  2. Localizar un marco libre.
- Si esta libre usarlo, sino usar un marco victima y escribirlo en el disco.
3. Leer página deseada y cargarla en el marco liberado.
  4. Reiniciar el proceso usuario.

# BIT DE MODIFICACIÓN

Usando un bit de modificación se reduce la carga de trabajo adicional.

Ese bit esta asociado a una página o marco, solo se activa si la página ha sido modificada.

Si una página debe sustituirse, se lee este bit para saber si escribir la página en el disco (si es que esta activo) o si no es necesario (el bit no esta activo).

Este método reduce a la mitad del tiempo la asignación de E/S si la página no ha sido modificada,

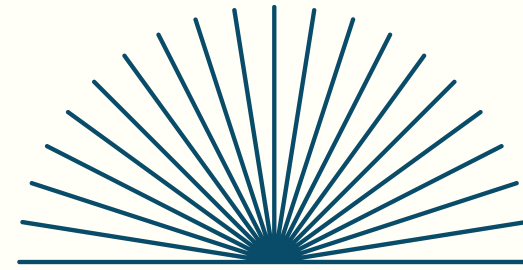


## Paginación bajo demanda

La sustitución de páginas es ideal aplicada al mecanismo de paginación bajo demanda, pues se puede proporcionar memoria virtual a partir de una memoria física pequeña.

Si tenemos un proceso de 20 paginas se puede ejecutar en diez marcos usando paginación.





# SUSTITUCIÓN DE PÁGINAS FIFO





## Sustitución FIFO

Asocia cada página en el instante en que se cargo a la memoria. Es el primero en entrar, primero en salir.

Si se sustituye se elige la página mas antigua.

Por ello puedo no ser bueno, si el ultimo dato debe salir y es importante para el proceso, nos causara problemas




# SUSTITUCIÓN ÓPTIMA DE PÁGINAS





# DESCRIPCIÓN DEL ALGORITMO ÓPTIMO:

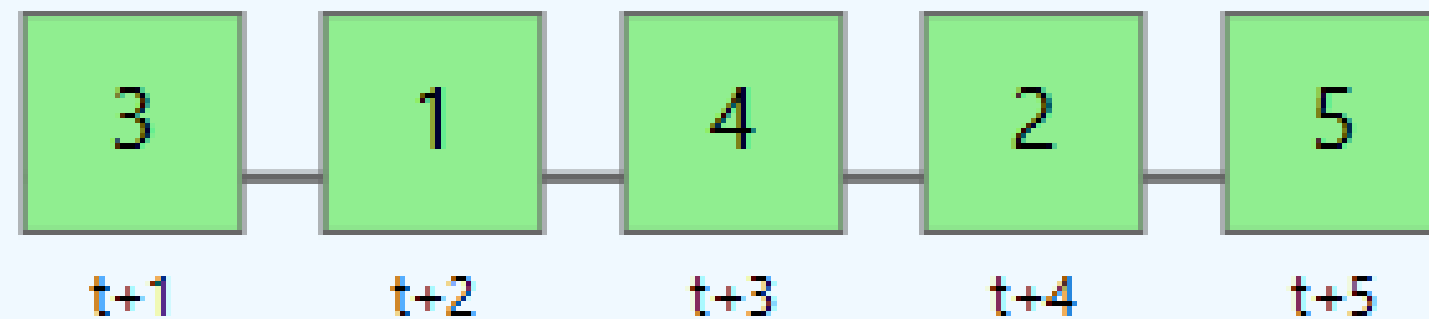


El algoritmo de sustitución de páginas óptimo (OPT o MIN) reemplaza la página que no se volverá a utilizar durante el periodo de tiempo más largo. Este enfoque asegura la menor tasa posible de fallos de página para un número fijo de marcos.

# FUNCIONAMIENTO:

## Escenario Teórico (Ideal)

Referencias Futuras Conocidas:



Decisión Óptima:

Reemplazar página 5 (próximo uso más lejano)

Este algoritmo revisa las futuras referencias de página y selecciona para su reemplazo aquella página que estará sin usarse por más tiempo en el futuro. Esto garantiza la mínima cantidad de fallos de página, pero requiere un conocimiento previo de la secuencia de acceso, lo que lo hace impráctico en la mayoría de los casos reales.

# EJEMPLO DE APLICACIÓN:

Con una secuencia de referencia de ejemplo, el algoritmo óptimo resulta en 9 fallos de página, mientras que un algoritmo FIFO (primero en entrar, primero en salir) podría tener hasta 15 fallos para la misma secuencia, si no se consideran los primeros tres fallos iniciales para llenar los marcos de memoria. Esto muestra que el algoritmo óptimo es mucho más eficiente que FIFO.

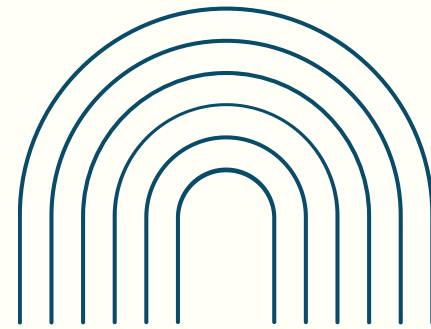


# LIMITACIÓN:

La principal desventaja es que este algoritmo es complicado de implementar en sistemas reales porque requiere conocimiento del futuro, lo cual generalmente no es posible.

# PROPÓSITO DE COMPARACIÓN:

Aunque es teóricamente ideal, el algoritmo óptimo se utiliza principalmente como una referencia comparativa para evaluar la eficiencia de otros algoritmos de reemplazo.



# SUSTITUCIÓN DE PÁGINAS LRU (LEAST RECENTLY USED)

---



# DESCRIPCIÓN DEL ALGORITMO LRU:

El algoritmo LRU reemplaza la página que no ha sido utilizada en el periodo de tiempo más largo hacia atrás. Es una aproximación práctica al algoritmo óptimo, que, en lugar de ver hacia adelante, observa el pasado reciente.

cadena de referencia

7 0 1 2 0 3 0 4 2 3 0 3 2 1 2 0 1 7 0 1



marcos de página

Figura 9.14 Algoritmo óptimo de sustitución de páginas.

# DIFERENCIA ENTRE FIFO, OPT Y LRU:

- FIFO: Reemplaza la página más antigua (la que fue cargada en primer lugar).
- OPT: Reemplaza la página que se usará más lejos en el futuro.
- LRU: Reemplaza la página que no ha sido usada en el mayor tiempo hacia atrás.

# VENTAJA DE LRU:

Al utilizar el pasado como indicador, LRU ofrece una solución práctica y aproximada al algoritmo óptimo sin la necesidad de conocer el futuro. Es aplicable en situaciones reales donde el algoritmo óptimo no es factible.

## EJEMPLO DE APLICACIÓN DE LRU:

En el mismo ejemplo de referencia, LRU produce 12 fallos de página. Aunque es más que los 9 fallos de página del óptimo, sigue siendo una mejora significativa sobre el FIFO, que daría 15 fallos de página.

# IMPLEMENTACIÓN DE LRU:

- Método de Contadores: Se asocia un contador o reloj lógico con cada página para registrar el instante de su último uso. Este método requiere que se actualice constantemente el contador cada vez que se accede a una página.
- Método de Pila: Se mantiene una pila de páginas, donde cada referencia mueve la página correspondiente al tope de la pila. Esto permite que la página menos recientemente utilizada quede en la base de la pila, lista para ser reemplazada cuando sea necesario.

# DESVENTAJAS DE LRU:

- Ambos métodos de implementación de LRU tienen sus inconvenientes:
- Los contadores requieren mantenimiento constante de registros de tiempo, lo cual puede ser costoso en cuanto a recursos.
- El uso de una pila requiere reorganizar continuamente las referencias, lo cual también es costoso en términos de tiempo de procesamiento.

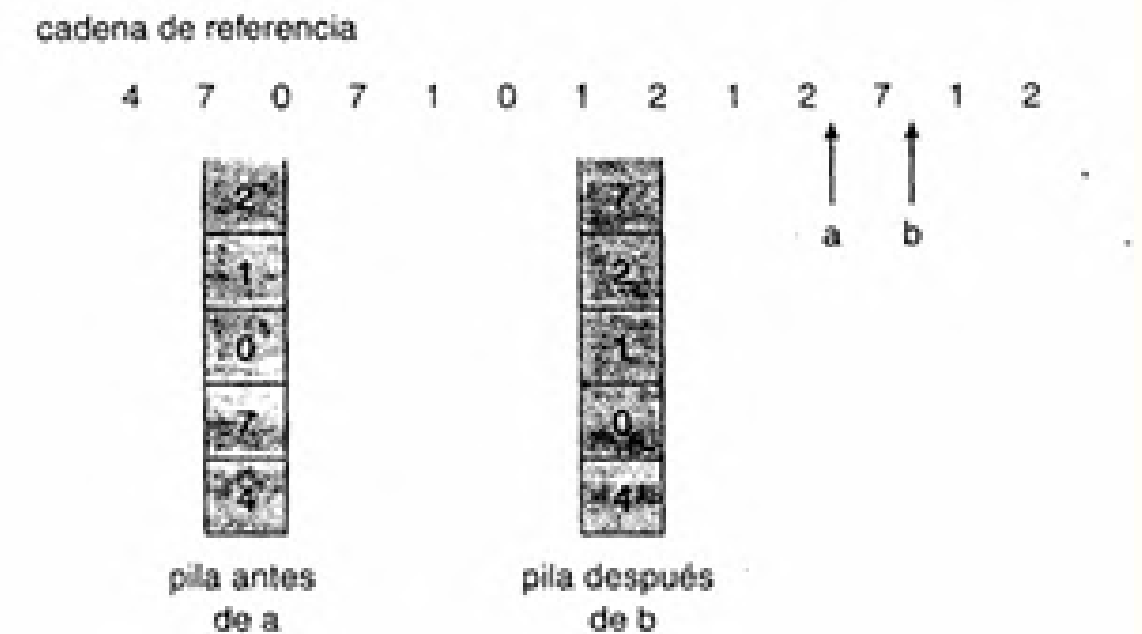


# SUSTITUCIÓN DE PÁGINAS MEDIANTE APROXIMACIÓN LRU



# CONCEPTO PRINCIPAL:

La sustitución de páginas mediante aproximación LRU es una solución que surge debido a las limitaciones del hardware en los sistemas informáticos. Es importante porque no todos los sistemas tienen la capacidad para implementar algoritmos de sustitución de páginas de manera completa.



# LIMITACIONES DE HARDWARE:

- La mayoría de los sistemas informáticos no cuentan con el soporte hardware suficiente para implementar algoritmos verdaderos de sustitución de páginas
- Algunos sistemas directamente no proporcionan ningún soporte hardware
- Esta limitación lleva a la necesidad de buscar alternativas y aproximaciones



# SOLUCIÓN INTERMEDIA:

- Como alternativa, muchos sistemas proporcionan un bit de referencia por página
- Este bit funciona como una ayuda básica para el manejo de páginas
- Es una solución menos compleja que un sistema completo de sustitución

# FUNCIONAMIENTO DEL BIT DE REFERENCIA:

- Al inicio del proceso, todos los bits son desactivados (valor 0) por el sistema operativo
- El bit se activa por el hardware cuando:
  - Se accede a la página
  - Se lee la página
  - Se escribe cualquier byte en dicha página

# PROCESO DE USUARIO:

- Cuando se ejecuta un proceso de usuario:
  - El bit asociado con cada página se activa
  - Esta activación ocurre cuando las páginas son referenciadas
- El sistema puede determinar qué páginas se han usado y cuáles no examinando los bits de referencia

# IMPORTANCIA DEL SISTEMA:

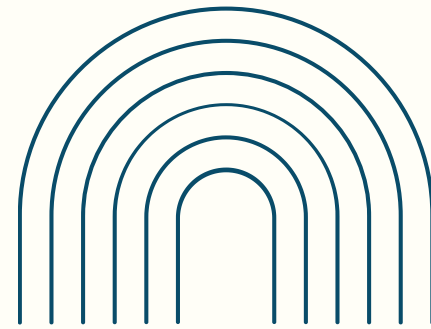
- Esta información constituye la base para muchos algoritmos de sustitución de páginas
- Permite una aproximación práctica al problema de gestión de memoria
- Aunque no es tan preciso como un sistema LRU completo, ofrece una solución viable

# VENTAJAS:

- Es más simple de implementar que un sistema LRU completo
- Requiere menos recursos de hardware
- Proporciona una solución práctica al problema de sustitución de páginas

# LIMITACIONES:

- No puede determinar el orden exacto de utilización de las páginas
- Es una aproximación, no una solución exacta
- La precisión es menor que en un sistema LRU verdadero



# ALGORITMO DE LOS BITS DE REFERENCIA ADICIONALES

---

## CONCEPTO PRINCIPAL:

Este algoritmo es una mejora del sistema básico de bits de referencia, que proporciona información más detallada sobre el uso de las páginas en memoria mediante el registro de bits de referencia a intervalos regulares.

# ESTRUCTURA DE ALMACENAMIENTO:

- Se utiliza un byte (8 bits) para cada página
- Se mantiene en una tabla de memoria específica
- Los intervalos de registro son regulares (por ejemplo, cada 100 milisegundos)



# FUNCIONAMIENTO DEL SISTEMA OPERATIVO:

- El sistema operativo maneja el desplazamiento de bits periódicamente
- En cada intervalo:
  - Desplaza el bit de referencia de cada página
  - Lo mueve a la posición de mayor peso
  - Descarta el bit de menor peso previo

# INTERPRETACIÓN DE LOS REGISTROS:

- Registro 00000000:
  - Indica que la página no ha sido utilizada
  - Representa inactividad durante 8 períodos temporales
- Registro 11111111:
  - Indica uso frecuente de la página
  - La página ha sido referenciada en todos los períodos

# ANÁLISIS DE PATRONES:

- Se pueden interpretar los bytes como números enteros
- Una página con valor 11110000:
  - Ha sido utilizada recientemente
  - Luego no ha tenido actividad
- Una página con valor 00001111:
  - No ha sido usada recientemente
  - Tuvo actividad en períodos anteriores

# CRITERIOS DE SUSTITUCIÓN:

- Se pueden sustituir páginas basándose en estos valores
- No se garantiza la unicidad de los números
- Es necesario sustituir todas las páginas con el valor más pequeño

# VENTAJAS:

- Proporciona un historial de uso más detallado
- Permite decisiones más informadas sobre sustitución
- Mejora la precisión respecto al bit único de referencia

# PARTICULARIDADES DEL MÉTODO:

- El número de bits del registro puede variarse
- La selección depende del hardware disponible
- Busca optimizar la velocidad de actualización

# ALGORITMO DE SEGUNDA OPORTUNIDAD



## CONCEPTO PRINCIPAL:

El algoritmo de segunda oportunidad es una variación del algoritmo FIFO básico que añade un nivel adicional de consideración antes de sustituir una página, mejorando así la eficiencia en la gestión de memoria.



# BASE DEL ALGORITMO:

- Es fundamentalmente un algoritmo de sustitución FIFO
- Incorpora una verificación adicional del bit de referencia
- Proporciona una "segunda oportunidad" a las páginas utilizadas



# PROCESO DE FUNCIONAMIENTO:

- Cuando se selecciona una página para sustitución:
  - Se verifica su bit de referencia
  - Si el bit es 0: se sustituye la página inmediatamente
  - Si el bit es 1:
    - Se le da una segunda oportunidad
    - Se pasa a la siguiente página en la lista FIFO
    - Se borra el bit de referencia

# IMPLEMENTACIÓN MEDIANTE COLA CIRCULAR:

- Utiliza una estructura de cola circular
- Emplea un puntero para seguimiento
- El puntero indica la siguiente página a examinar

## COMPORTAMIENTO DEL PUNTERO:

- Avanza por la cola buscando páginas
- Se detiene cuando encuentra una página con bit de referencia 0
- Si no encuentra ninguna, da la vuelta completa

# CASOS DE USO:

- Caso óptimo:
  - Encuentra rápidamente una página con bit 0
  - Realiza la sustitución inmediatamente
- Peor caso:
  - Recorre toda la cola
  - Todas las páginas tienen bit de referencia activo
  - Debe dar segunda oportunidad a todas

# CARACTERÍSTICAS ESPECIALES:

- Una página muy utilizada mantiene su bit de referencia en 1
- Permanece activa en memoria más tiempo que en FIFO puro
- Las páginas poco utilizadas son candidatas más probables para sustitución

# CONSIDERACIONES DE IMPLEMENTACIÓN:

- El bit de referencia se borra al dar segunda oportunidad
- La página conserva su posición en la cola
- El algoritmo combina aspectos de FIFO y LRU

## VENTAJAS:

- Mejor rendimiento que FIFO simple
- Protege páginas frecuentemente utilizadas
- Implementación relativamente sencilla

# DESVENTAJAS:

- Puede requerir un recorrido completo de la cola
- Mayor overhead que FIFO puro
- No garantiza la mejor selección posible

# VARIACIONES Y ADAPTACIONES:

- Puede modificarse el número de "oportunidades"
- Se puede ajustar el criterio de segunda oportunidad
- Permite personalización según necesidades específicas

# ALGORITMO MEJORADO DE SEGUNDA OPORTUNIDAD

Es posible mejorar el algoritmo de segunda oportunidad al considerar los bits de referencia y modificación como un par ordenado. Al hacer esta consideración, se presentan cuatro casos posibles.

- (0,0). No hay ninguna modificación reciente; es la mejor página para sustitución.
- (0,1). No se ha usado recientemente pero sí se ha modificado; se requiere escribir la página antes de hacer la sustitución.
- (1,0). Se usó recientemente pero está limpia. Es probable que sea utilizada pronto.
- (1,1). Se ha utilizado y modificado recientemente; hay probabilidad de que se vuelva a utilizar y es necesario escribirla en disco antes de efectuar la sustitución.



- Toda página pertenece a una de estas cuatro clases mencionadas.
- Si hace falta sustituir una página se utiliza el mismo esquema que en el algoritmo del reloj, con la diferencia de que aquí se examina la clase a la que pertenece la página en cuestión.
- Es por ello que se sustituye la primera página encontrada perteneciente a la clase no vacía más baja.
- Es posible tener que recorrer la cola circular en múltiples ocasiones antes de encontrar una página que se pueda sustituir.
- La principal diferencia entre el algoritmo en cuestión y el del reloj se encuentra en el hecho de que aquí se da la preferencia a todas aquellas páginas que no se han modificado con la finalidad de reducir el número de operaciones de entrada y salida efectuadas.



# SUSTITUCIÓN DE PÁGINAS BASADA EN CONTADOR

Para efectuar la sustitución de páginas puede mantenerse un contador de la cantidad de referencias que se han hecho a cada página y desarrollar alguno de los siguientes esquemas.

- **Last frequently used.** Requiere sustituir la página con el valor de contador más bajo, debido a que las páginas más utilizadas deben tener un valor mayor de referencias. La desventaja es que si hay una página inicialmente muy utilizada, posteriormente quedará en desuso al tener un valor de contador muy grande.
- **Most frequently used.** Considera que la página con el valor de contador más bajo se cargó en la memoria recientemente y por ende, aún debe ser utilizada.

La implementación de estos algoritmos resulta bastante costosa, además de que no proporcionan buenas aproximaciones al algoritmo de sustitución OPT.

# ALGORITMOS DE BÚFER DE PÁGINAS

Los sistemas suelen mantener un conjunto compartido de marcos libres. Al producirse un fallo de página, se selecciona un marco víctima como antes.

La página se lee en un marco libre extraído de dicho conjunto, antes de que se escriba en disco la víctima, lo que permite que el proceso se reinicie lo antes posible sin esperar a que se descargue la página víctima; al descargarse, su marco se agrega al conjunto compartido de marcos libres.

Este concepto puede ampliarse si se mantiene un conjunto compartido de marcos libres y recordando qué página se ha alojado en cada marco. Dado que el contenido de un marco no se modifica al escribirlo en el disco, es posible reutilizar la página antigua a partir del conjunto de marcos libres si fuese necesario y si dicho marco no se ha reutilizado.

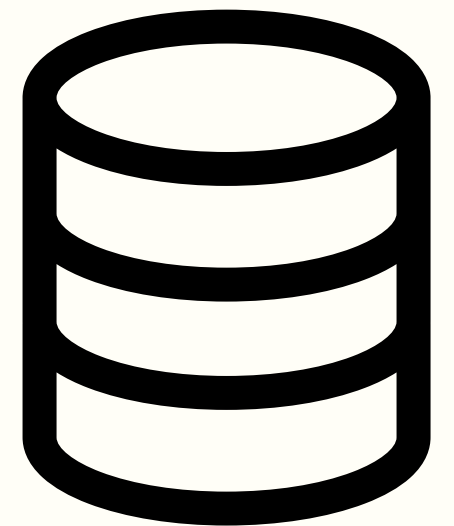
# APLICACIONES Y SUSTITUCIÓN DE PÁGINAS

Algunas aplicaciones que acceden a los datos a través de la memoria virtual del sistema operativo tienen un rendimiento peor que si el sistema operativo no proporcionara ningún mecanismo de búfer.

## **Ejemplo: una base de datos.**

Esta proporciona sus propios mecanismos de gestión de memoria y búfers de entrada y salida. Estas aplicaciones conocen su uso de la memoria y el disco mejor que el sistema operativo, el cual implementa algoritmos de propósito general.

Si el sistema operativo proporciona mecanismos de búfers de entrada y salida, se utiliza el doble de memoria para realizar operaciones de entrada y salida.



### **Ejemplo: almacenes de datos.**

En estos sistemas se realizan frecuentemente lecturas de disco secuenciales masivas, seguidas de cálculos intensivos y estructuras.

Un algoritmo LRU se acercaría a eliminar las páginas antiguas y conservar las nuevas; es por ello que para este caso, el algoritmo MFU sería más eficiente.

Dados estos problemas, algunos sistemas operativos dan a ciertos programas especiales la capacidad de utilizar una partición del disco a modo de una gran matriz secuencial (disco sin formato) de bloques lógicos, sin ninguna estructura de datos propia de los sistemas de archivos.

Las operaciones de entrada y salida efectuadas sobre esta matriz se consideran entradas y salidas sin formato; estas puentean todos los servicios del sistema de archivos, como la paginación bajo demanda para la entrada y salida de archivos, bloqueo de archivos, la preextracción, la asignación de espacio, nombres de archivo y los directorios.



# CAUSA DE LA SOBREPAGINACIÓN

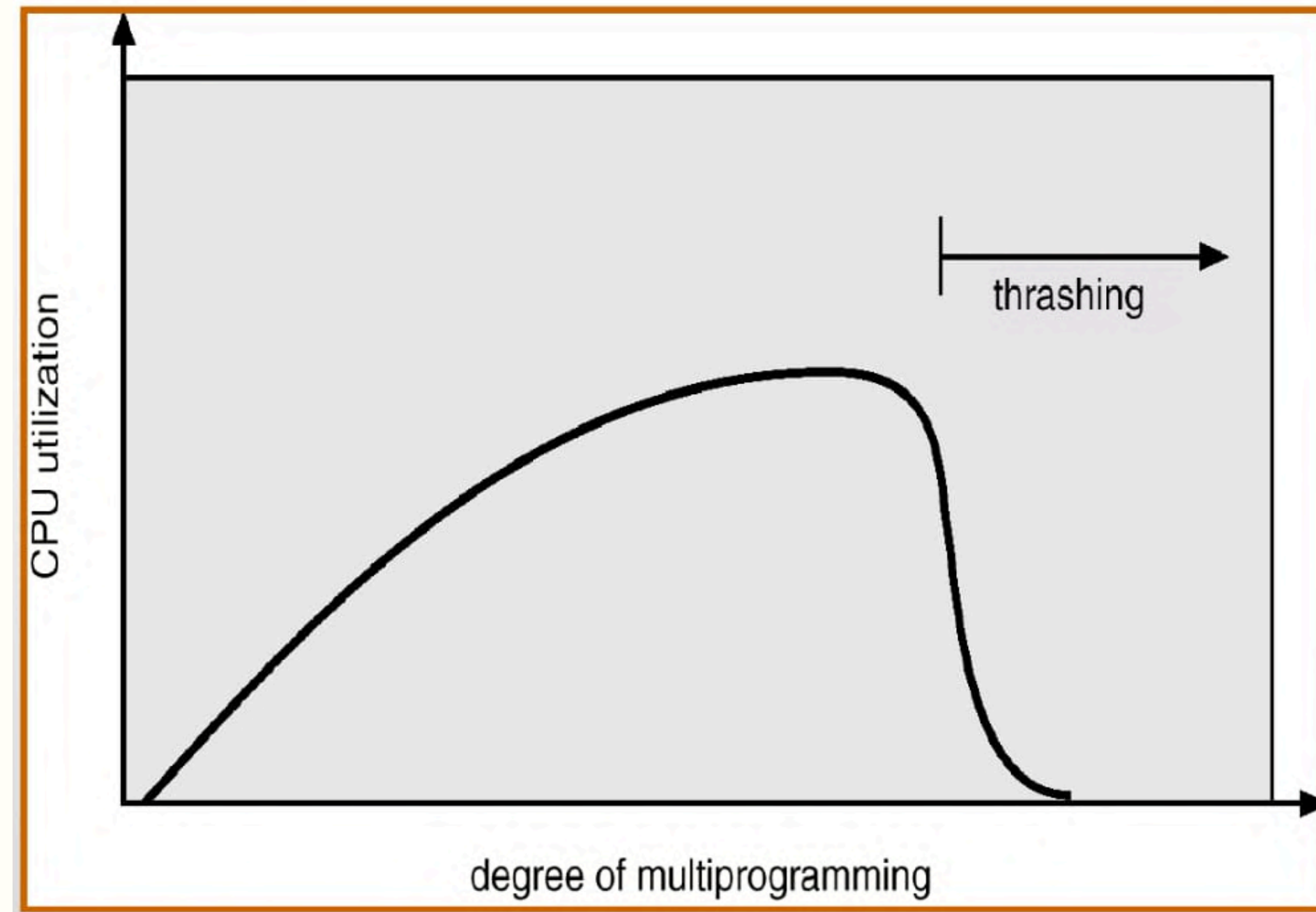
La sobrepaginación provoca graves problemas de rendimiento.

El sistema operativo gestiona la utilización de la CPU. Si la tasa de utilización es baja, el grado de multiprogramación aumenta al introducir un nuevo proceso en el sistema. Se utiliza un algoritmo de sustitución global, el cual no considera a qué proceso pertenece una página para efectuar la sustitución.

El planificador de la CPU, al notar que la tasa de utilización desciende, incrementa el grado de multiprogramación. El nuevo proceso buscará iniciarse quitando marcos a los procesos que se están ejecutando, provocando más fallos de página y que la cola del dispositivo de paginación crezca.

Esta cuestión trae como consecuencia que la utilización de la CPU caiga gradualmente y que el planificador de la CPU trate de aumentar el grado de multiprogramación en mayor medida. Se produce una sobrepaginación y la tasa de procesamiento del sistema desciende considerablemente, incrementándose a su vez la tasa de fallos de página.

La gráfica en cuestión muestra la tasa de utilización de la CPU en función del grado de multiprogramación.





Se observa que a medida que el grado de multiprogramación aumenta, lo hace también la tasa de utilización de la CPU, aunque de una manera más lenta, hasta alcanzar un valor máximo.

Cuando el grado de multiprogramación crezca aún más, se presentará la sobrepaginación y la tasa de utilización de la CPU caerá abruptamente.

Es posible reducir los efectos de la sobrepaginación mediante un algoritmo de sustitución local (o basado en prioridades). Si uno de los procesos entra en sobrepaginación, es incapaz de robar marcos de otro proceso y hacer que este también entre en sobrepaginación. Esto por supuesto que no resuelve el problema por completo.

Para prevenir la sobrepaginación, se requiere proporcionar a cada procesos tantos marcos como necesiten, para lo cual existen diferentes técnicas; la estrategia basada en conjunto de trabajo comienza por examinar la cantidad de marcos que un proceso está utilizando realmente. Dicha técnica define el modelo de localidad de ejecución del proceso.



# MODELO DEL CONJUNTO DE TRABAJO

El modelo del conjunto de trabajo se basa en la suposición de la localidad de ejecución de los programas. Para su implementación, se utiliza un parámetro Delta para definir la ventana del conjunto de trabajo.

La idea es examinar las Delta referencias de página más recientes; el conjunto de dichas referencias es el conjunto de trabajo. Si una página está siendo usada con frecuencia, aparecerá dentro del conjunto de trabajo; si no se utiliza, esta será eliminada del mismo, Delta unidades después de la última referencia realizada a la página.

El sistema operativo monitorea el conjunto de trabajo de cada proceso y asigna a este conjunto los suficientes marcos que lo satisfagan en cuanto al tamaño requerido. Si hay los suficientes marcos, puede iniciarse un proceso; cuando la suma de los tamaños de los conjuntos de trabajo excede el total de marcos disponibles, el propio sistema operativo selecciona un proceso para suspenderlo, evitando la sobrepaginación.



# ASIGNACIÓN DE MARCOS

Existen varias estrategias para asignar marcos de memoria. Una opción es que el sistema operativo asigne sus búferes y tablas desde la lista de marcos libres, de modo que, si no los utiliza, puedan destinarse a paginación para el usuario. Otra opción es reservar 3 marcos libres para garantizar disponibilidad en caso de fallos de página. La **estrategia básica** consiste en **asignar todos los marcos libres al proceso de usuario**.

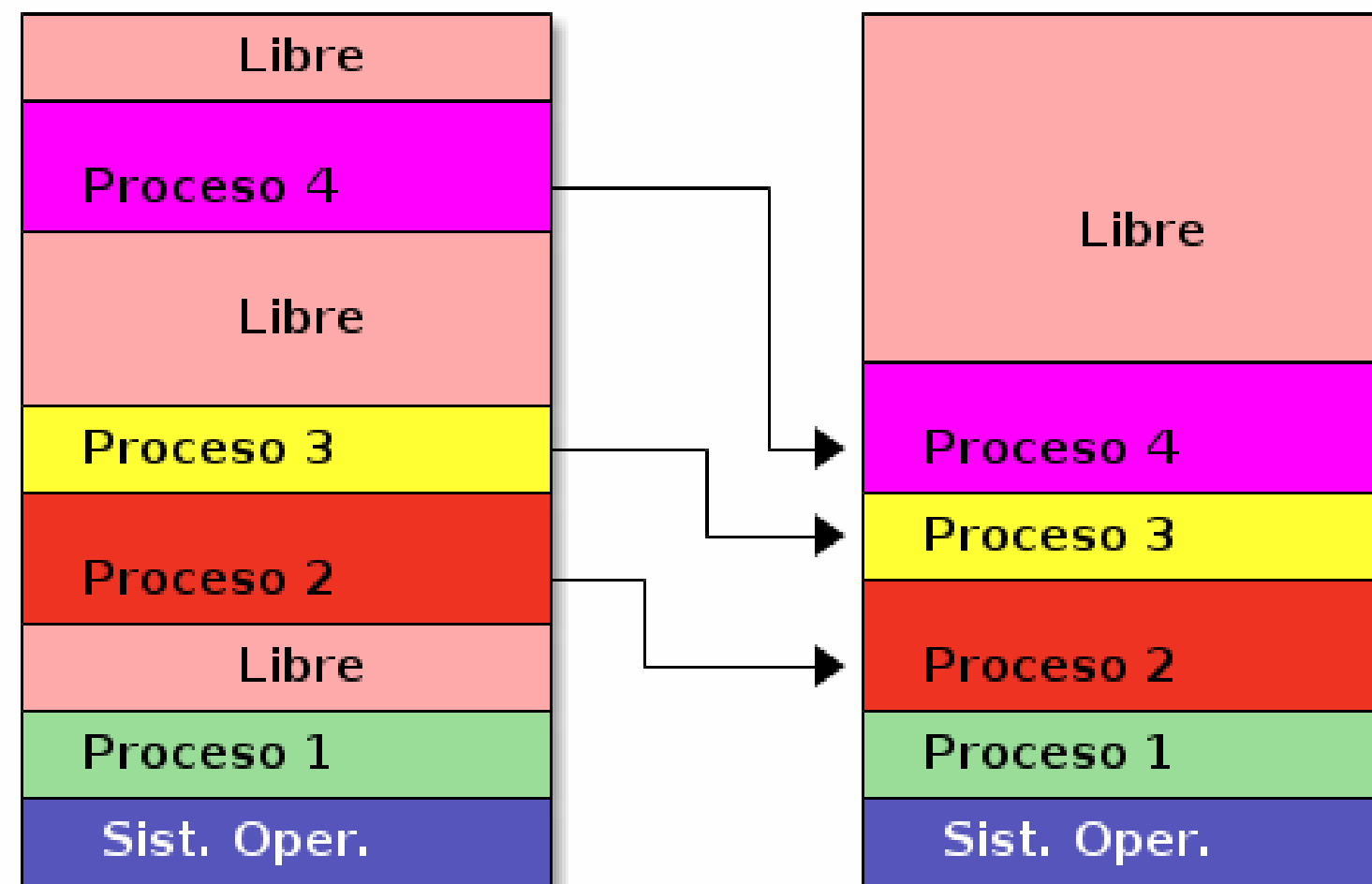
Por ejemplo, para asignar marcos de memoria, planteamos cómo distribuir la memoria libre entre varios procesos. Supongamos que hay 93 marcos disponibles y dos procesos. En un ejemplo simple de sistema monousuario con 128 KB de memoria dividida en páginas de 1 KB, el sistema operativo usa 35 marcos, dejando 93 libres para el proceso del usuario. Con paginación bajo demanda, estos 93 marcos se ubican primero en la lista de marcos libres. Cuando se agoten, un algoritmo de sustitución elegirá una de las 93 páginas en uso para reemplazarla con la siguiente. Al finalizar el proceso, los 93 marcos se devolverán a la lista de disponibles.



# NÚMERO MÍNIMO DE MARCOS

Es esencial no asignar más marcos de memoria que los disponibles y asegurar un número mínimo de marcos para cada proceso. Esto es crucial para el rendimiento, ya que menos marcos asignados aumentan la tasa de fallos de página, ralentizando el proceso. Además, si ocurre un fallo de página antes de completar una instrucción, esta debe reiniciarse. Por ello, es necesario contar con suficientes marcos para alojar todas las páginas que una instrucción pueda requerir.

El **número mínimo** de marcos está **definido** por la **arquitectura informática**. mientras que el **número máximo** está **definido** por la cantidad de **memoria física** disponible.



### **Ejemplo:**

Imaginemos una máquina en la que cada instrucción que accede a la memoria utiliza solo una dirección de memoria. En este caso, al menos un marco sería necesario para la instrucción y otro para la dirección de memoria referenciada. Además, si se permite una dirección indirecta (por ejemplo, una instrucción de carga en la página 16 que apunte a una dirección en la página 0, la cual a su vez referencia a la página 23), entonces el sistema de paginación necesitará al menos tres marcos por proceso para funcionar correctamente.

## **PEOR ESCENARIO**

Se produce en aquellas arquitecturas informáticas que permiten múltiples niveles de indirección.

### **Ejemplo:**

En una arquitectura con palabras de 16 bits, donde 15 bits representan una dirección y 1 bit indica indirección, una instrucción de carga puede apuntar sucesivamente a direcciones indirectas.

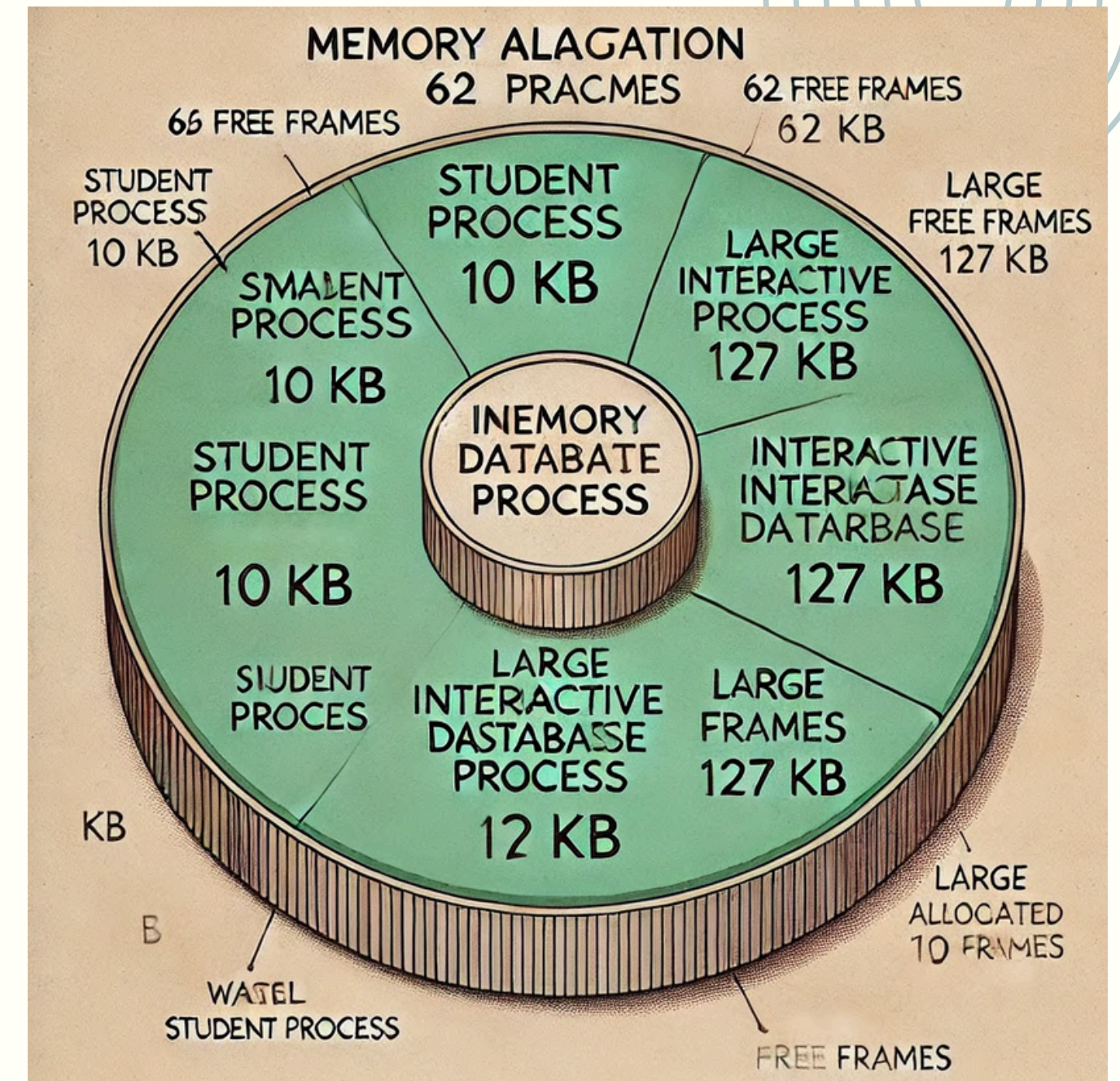
En el peor de los casos, esto requeriría que toda la memoria virtual esté cargada en la memoria física.

**Para resolver una dificultad sobre el peor de los casos, debemos de imponer un límite en el número de niveles de dirección.**



# ALGORITMOS DE ASIGNACIÓN

- La **asignación equitativa** es un método eficiente para repartir  $m$  marcos entre  $n$  procesos, dando a cada proceso un número igual de marcos,  $m/n$ . Por ejemplo, si hay 93 marcos y 5 procesos, cada uno recibe 18 marcos, y los 3 marcos restantes se utilizan como un conjunto compartido de marcos libres.
- Otra alternativa es utilizar una **asignación proporcional**. La **asignación proporcional distribuye la memoria disponible entre los procesos según su tamaño**. Por ejemplo, en un sistema con 62 marcos libres y dos procesos (uno de 10 KB y otro de 127 KB), asignar 31 marcos a cada proceso no sería eficiente, ya que el proceso pequeño solo necesita 10 marcos. Con la asignación proporcional, se asigna memoria de manera más eficiente, distribuyendo los marcos según las necesidades de cada proceso.





Para el ejemplo anterior usando **asignación proporcional**. Sea  $S_i$ , el tamaño de la memoria virtual para el proceso  $p_i$ , definimos  $S = \sum S_i$

Entonces, si el número total de marcos disponibles es  $m$ , asignaremos  $a_i$  marcos al proceso donde  $a_i$ , es aproximadamente:  $a_i = S_i / S * m$ .

Utilizando un mecanismo de asignación proporcional, repartiríamos 62 marcos entre dos procesos, uno de 10 páginas y otro de 127 páginas, asignando 4 marcos y 57 marcos, respectivamente a los dos procesos, ya que:  $(10/137) * 62 \approx 4$  y  $(127/137) * 62 \approx 57$ .

De esta forma, ambos procesos compartirán los marcos disponibles de acuerdo con sus necesidades, en lugar de repartir los marcos equitativamente.

La asignación de memoria, tanto equitativa como proporcional, puede variar según el nivel de multiprogramación. Si este nivel aumenta, cada proceso pierde marcos de memoria para dar espacio a nuevos procesos. Si el nivel disminuye, los marcos de los procesos terminados se redistribuyen entre los procesos restantes.

También hay que notar que ambos mecanismos, se trata por igual a los procesos de mayor prioridad que a los de baja prioridad

# ASIGNACIÓN GLOBAL Y LOCAL

Si hay multiples procesos que compiten por los marcos , podemos clasificar los algoritmos de sustitución de páginas en dos categorias: **sustitución global** y **sustitución local**

- **Sustitución global.** Permite a un proceso seleccionar y quitar un marco de memoria a otro proceso, incluso si ese marco está actualmente asignado a dicho proceso.
- **Sustitución local.** Requiere que cada proceso sólo efectúe esa selección entre su propio conjunto de marcos asignado .

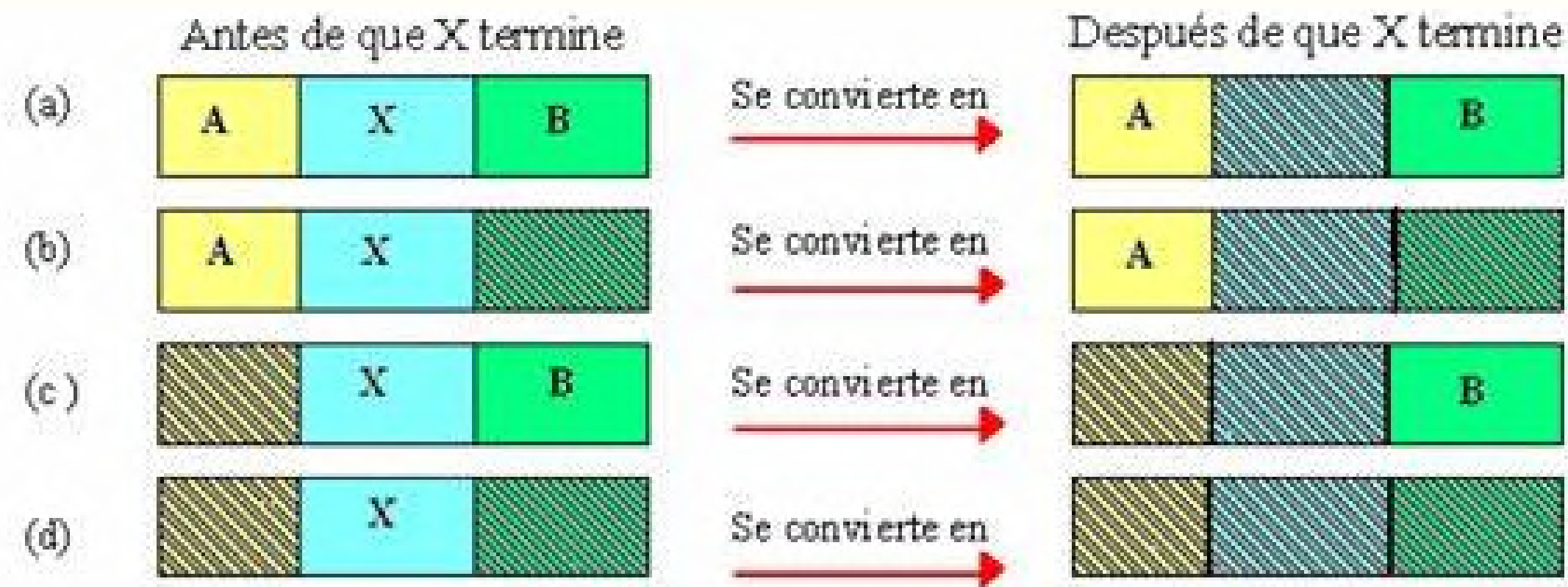


Figura 6.7. Cuatro combinaciones de vecinos para el proceso X que concluye

## EJEMPLO

Considere un esquema de asignación en el que permitamos a los procesos de alta prioridad seleccionar marcos de los procesos de baja prioridad para la sustitución. Un proceso podrá seleccionar un marco de sustitución de entre sus propios marcos o de entre los marcos de todos los procesos de menor prioridad. Esta técnica permite que un proceso de alta prioridad incremente su tasa de asignación de marcos a expensas de algún proceso de baja prioridad.

Con una estrategia de **sustitución local**, la cantidad de marcos asignados a un proceso permanece constante. En cambio, con la estrategia de **sustitución global**, un proceso puede seleccionar marcos de otros procesos, lo que aumenta su propio número de marcos asignados, siempre y cuando otros procesos no elijan sus marcos para sustitución.

## Problemas de sustitución local y global

**Sustitución global.** Un proceso no puede comprobar su propia tasa de fallos de página. El conjunto de páginas en memoria de un proceso depende tanto de su propio comportamiento de paginación como del de los demás procesos, lo que puede hacer que su comportamiento varíe debido a factores externos.

**Sustitución local.** Puede resultar perjudicial para un proceso, al no tener a su disposición otras páginas de memoria menos utilizadas. Lo cual puede generar una mayor tasa de procesamiento del sistema.



# SOBREPAGINACIÓN

Si el número de marcos asignados a un proceso de baja prioridad cae por debajo del número mínimo requerido por la arquitectura de la máquina, deberemos suspender la ejecución de dicho proceso y a continuación descargar de memoria todas sus restantes páginas, liberando así todos los marcos que estuvieran asignados. Este mecanismo introduce un nivel intermedio de planificación de la CPU, basado en la carga y la descarga de páginas .

Si un proceso no tiene suficientes marcos para las páginas que usa activamente, generará constantes fallos de página. Esto lo obliga a sustituir páginas que necesitará de nuevo pronto, produciendo una serie continua de fallos de página al tener que recargar inmediatamente las mismas páginas.

Esta alta tasa de actividad de paginación, se denomina **sobrepaginación**.

Un proceso entrará en sobrepaginación si invierte más tiempo implementando los mecanismos de paginación que en la propia ejecución del proceso.



# FRECUENCIA DE FALLOS DE PÁGINA

Para controlar la sobrepaginación, hay una estrategia más directa que está basada en la **frecuencia de fallos de página**. Cuando se entra en la sobrepaginación, se produce una alta tasa de fallos de página, por ende se busca controlar esa tasa. Si es demasiada alta, el proceso necesitará más marcos y a la inversa. Entonces, establecemos límites(superior e inferior), si la tasa real de fallos de página excede el límite superior, asignamos el proceso a otro marco, en caso contrario, eliminamos un marco del proceso. Esto nos permite medir y controlar directamente la tasa de fallos de página para evitar la sobrepaginación.