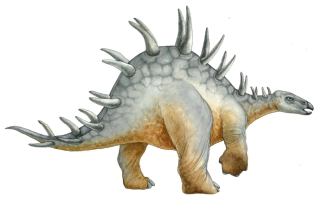
Capítulo 9: Memoria Virtual

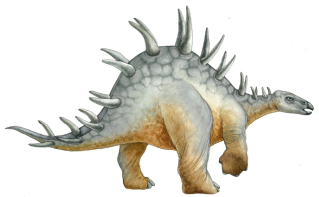
Capítulo 9: Memoria Virtual

- Antecedentes
- Paginación por demanda
- Copia-sobre-escritura
- Reemplazo de página
- Asignación de frames
- Thrashing
- Archivos mapeados a memoria
- Asignando memoria de kernel
- Otras consideraciones
- Ejemplos de sistemas operativos



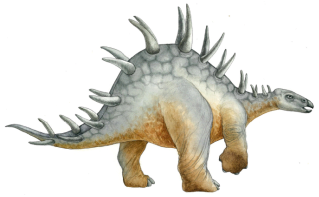
Objetivos

- Describir los beneficios de un sistema de memoria virtual
- Explicar los conceptos:
  - paginación por demanda,
  - algoritmos de reemplazo de página y
  - asignación de frames

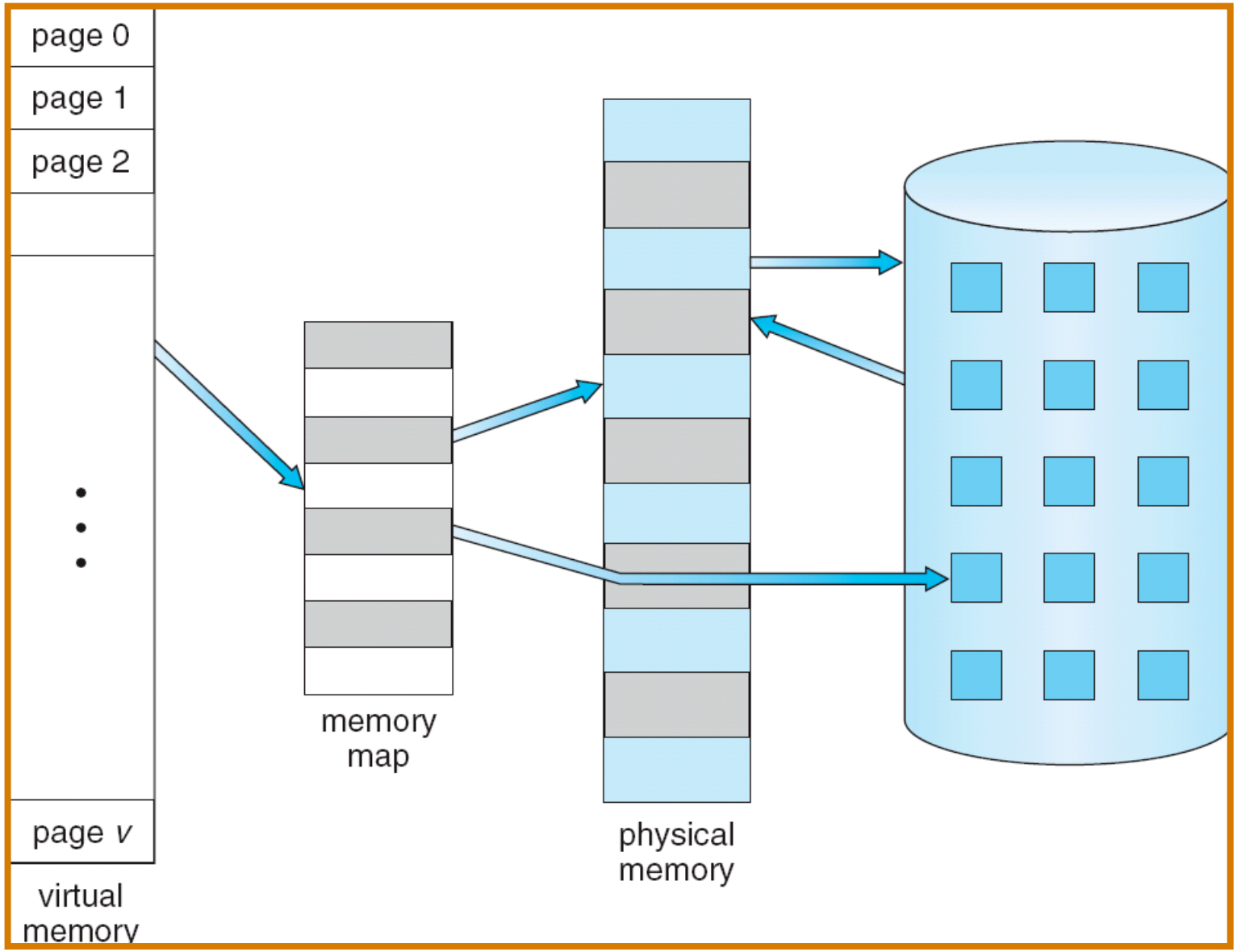


Antecedentes

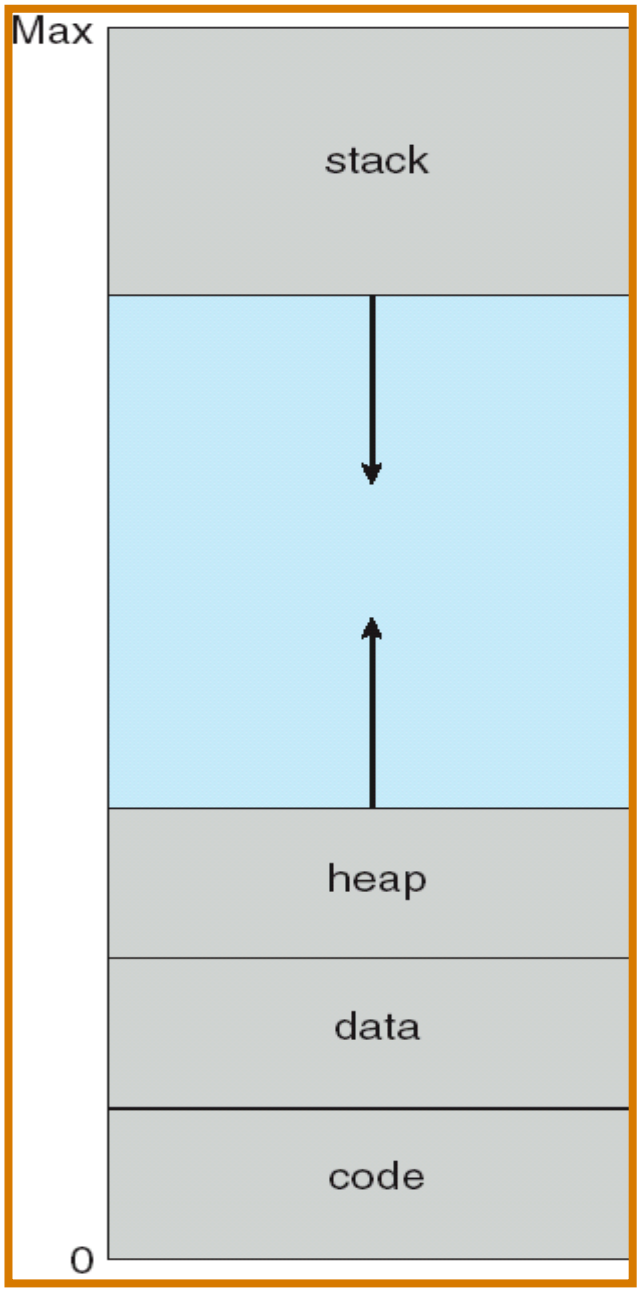
- Memoria Virtual – separación de memoria lógica de usuario de la memoria física.
  - Sólo una parte del programa debe estar en memoria para ejecución
  - Espacio de direcciones lógico puede ser más grande que la memoria física
  - Permite compartir espacios de direcciones entre varios procesos
  - Permite una creación más eficiente de procesos
- La memoria virtual puede implementarse a través de:
  - Paginación por demanda
  - Segmentación por demanda

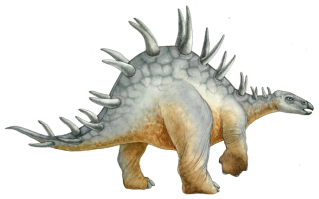


Memoria Virtual más grande que la Memoria Física

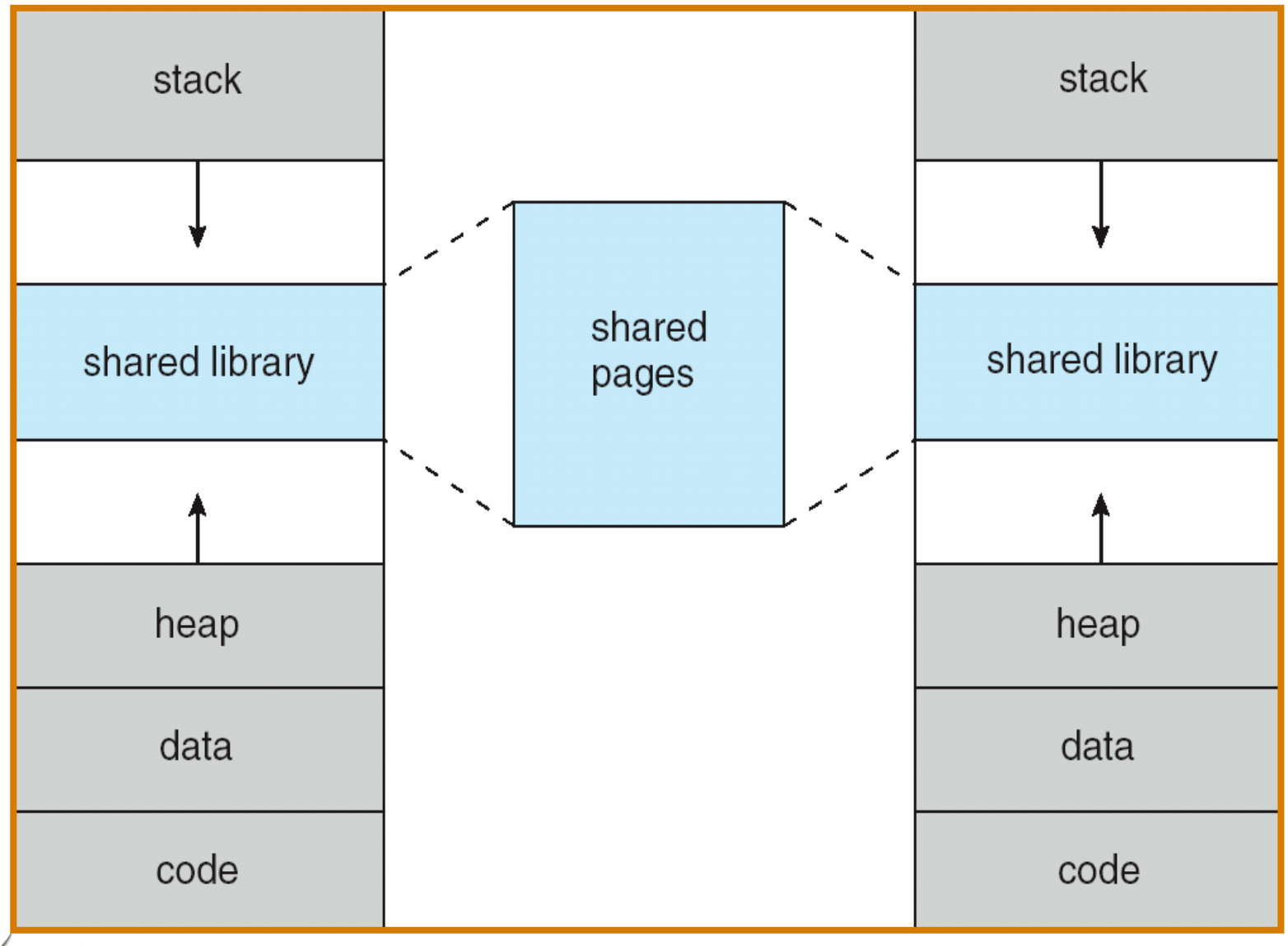


Espacio de direcciones virtual



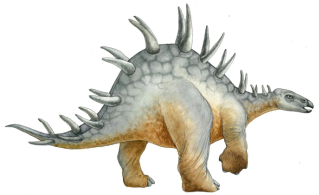


Biblioteca compartida utilizando memoria virtual

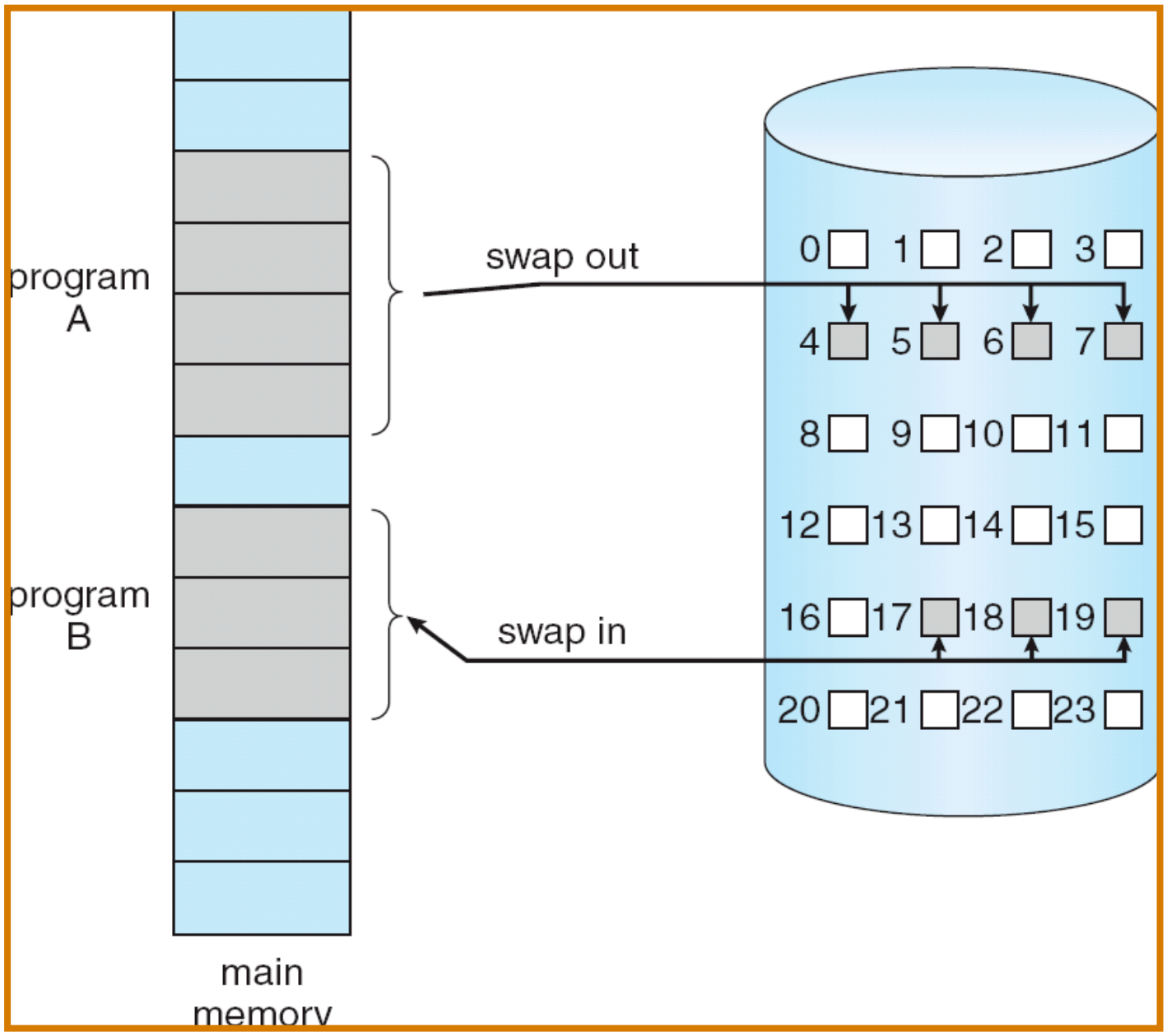


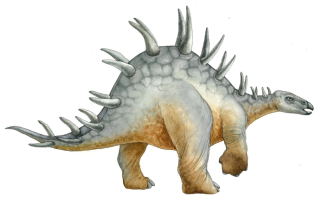
Paginación por demanda

- Traer una página a memoria cuando es requerida
  - Requiere menos E/S
  - Requiere menos memoria
  - Respuesta más rápida
  - Más usuarios
- Se requiere una página ⇒ referencia a ella
  - referencia inválida ⇒ abortar
  - no está en memoria ⇒ traer a memoria
- Swap perezoso – nunca mover una página a memoria a menos que sea requerida
  - Código que maneja páginas es un paginador



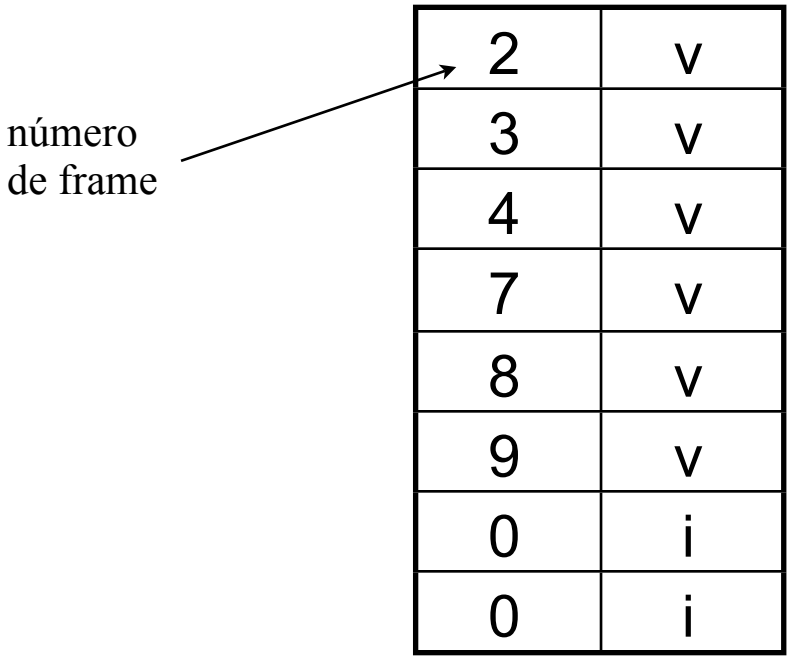
Transferir Memoria Paginada a espacio Disco Contiguo





Bit válido-inválido

- Se asocia un bit válido-inválido con cada página en la tabla (**v** ⇒ en-memoria, **i** ⇒ no-en-memoria)
- Inicialmente el bit vale **i** para toda entrada
- Ejemplo de una instantánea de la tabla de páginas:



- Durante la traducción de direcciones, si una entrada en la tabla de páginas tiene **i** ⇒ falta de página (*page fault*)

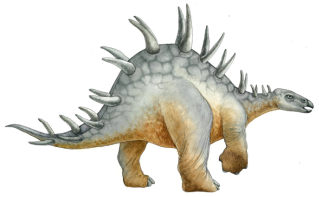
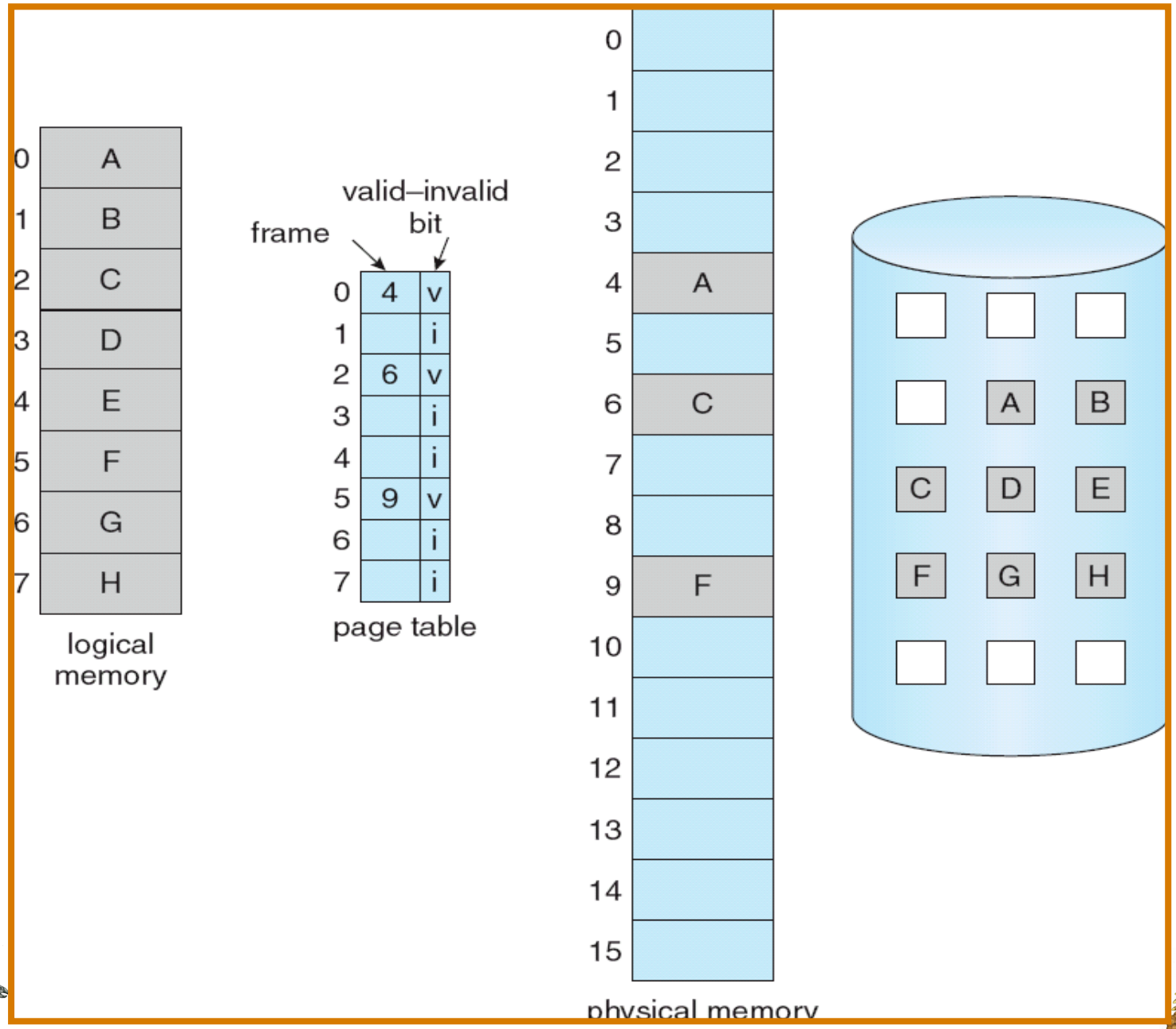


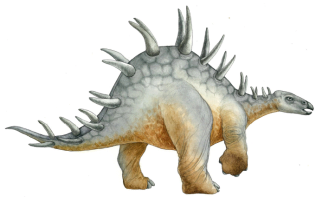
Tabla de páginas cuando algunas páginas NO están en Memoria Principal



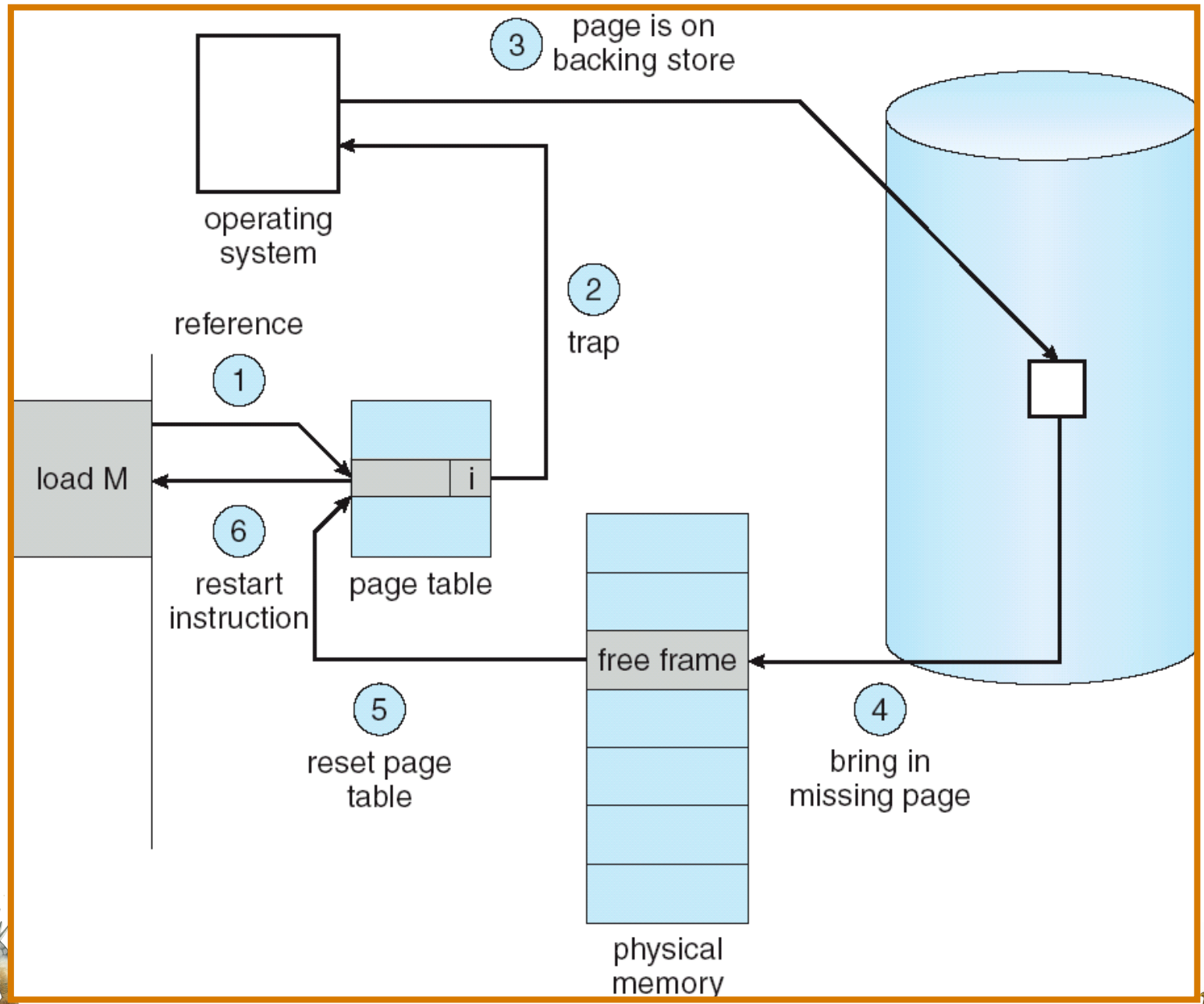
Falta de página

- Si hay una referencia a una página, la primera referencia a esa página genera trampa al SO:
  Falta de página

1. SO checa en otra tabla para decidir:
   - Referencia inválida ⇒ abortar
   - No está en memoria
2. Obtener un frame vacío
3. Mover la página al frame
4. Re-iniciar tablas
5. Asignar bit de validación = **v**
6. Re-iniciar la instrucción que ocasionó la falta



Pasos para manejar una falta de página

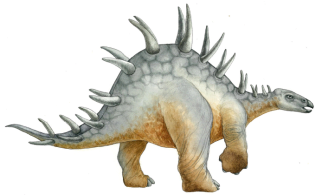


Rendimiento de Paginación por Demanda

- Ritmo de faltas de página $0 \le p \le 1.0$
  - si $p = 0$ no hay faltas de página
  - si $p = 1$, cada referencia es una falta

- Tiempo de Acceso Efectivo (EAT)

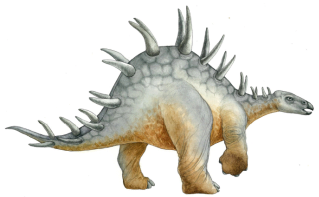
$$EAT = (1 - p) \times \text{acceso a memoria} + p\ (\text{overhead por falta de página} + \text{swap page out} + \text{swap page in} + \text{restart overhead})$$



Ejemplo de Paginación por Demanda

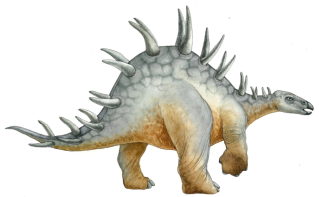
- Tiempo de acceso a memoria = 200 nanoseconds
- Tiempo promedio p/servicio por falta de página
  = 8 milliseconds
- EAT = (1 – p) x 200 + p (8 milliseconds)
  = (1 – p) x 200 + p x 8,000,000
  = 200 + p x 7,999,800
- Si uno de cada 1,000 accesos ocasiona una falta de página, entonces
  EAT = 8.2 microseconds

¡ Esto representa un factor de 40 de reducción !



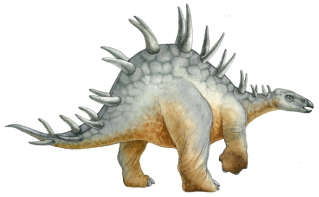
Creación de Procesos

- Memoria virtual permite otros beneficios durante la creación de procesos:
  - Copy-on-Write
  - Archivos Memory-Mapped

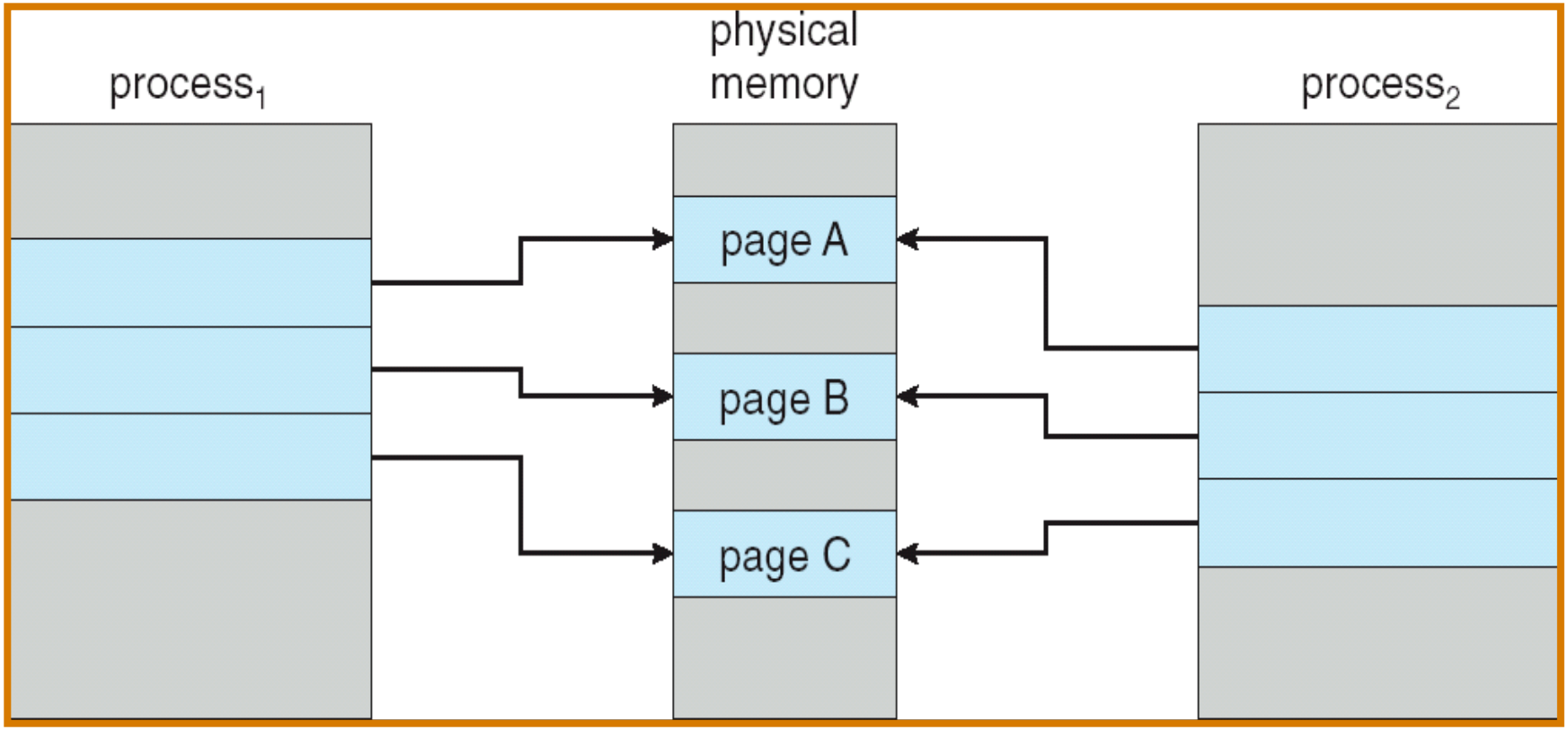


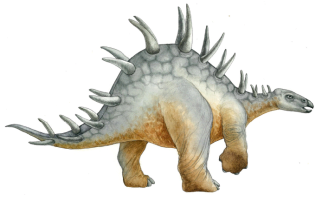
Copy-on-Write

- Copy-on-Write (COW) permite que tanto padre como hijo compartan *inicialmente* las mismas páginas en memoria
  - Si cualquier proceso modifica una página compartida, en ese momento se *copia*
- COW permite creación de procesos más eficiente, ya que sólo se copian las páginas modificadas
- Las páginas libres son asignadas de un *pool* de páginas limpias (zeroed-out)

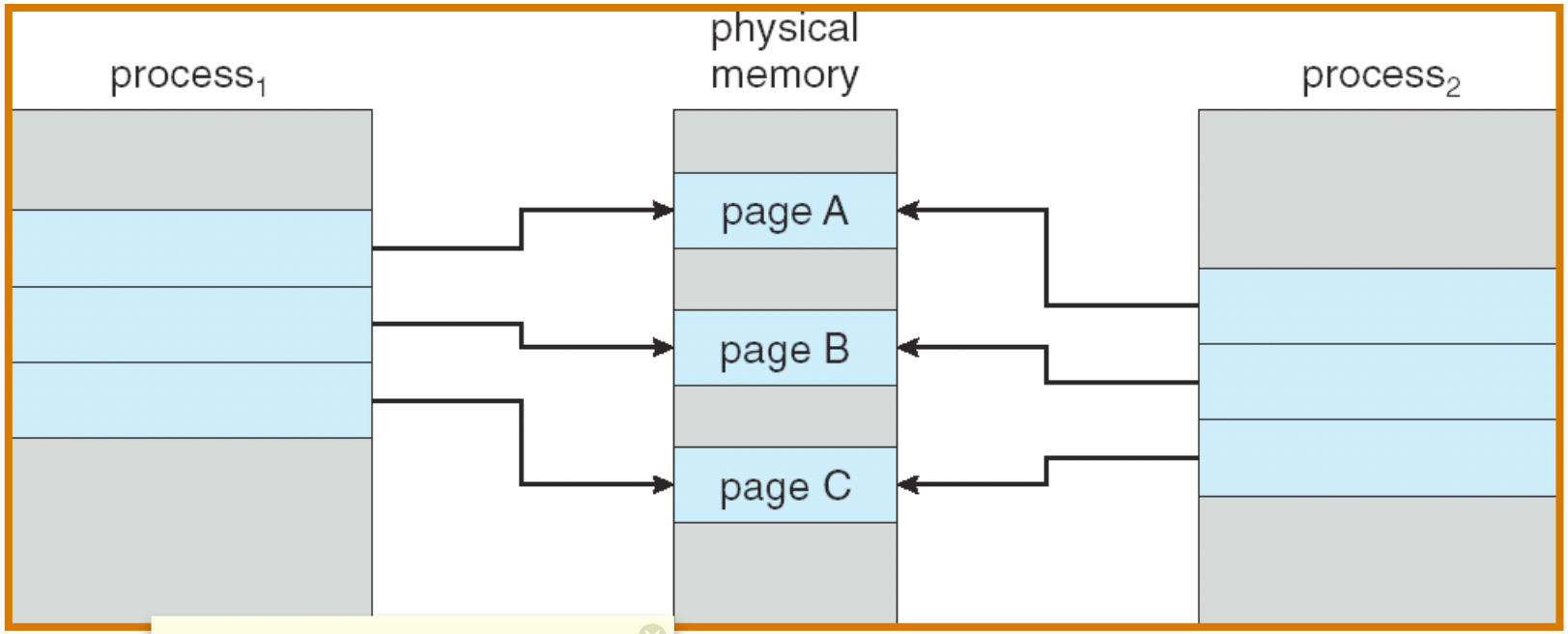


Antes que Proceso 1 modifique página C

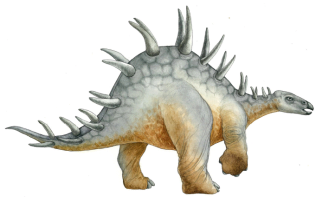




Después que Proceso 1 Modifica Página C

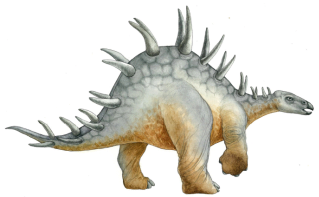


Esta imagen está mal. Deberíamos tener C y C' para el proceso 1.



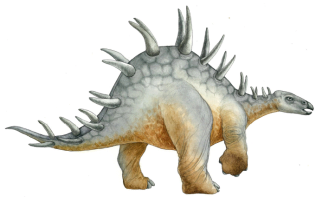
¿Qué sucede si no hay frames libres?

- Reemplazo de página – encuentra una página en memoria, que NO está en uso y sácala
  - algoritmo
  - rendimiento – queremos algoritmo que nos de el menor número de faltas de página
- Misma página puede ser traída a memoria muchas veces

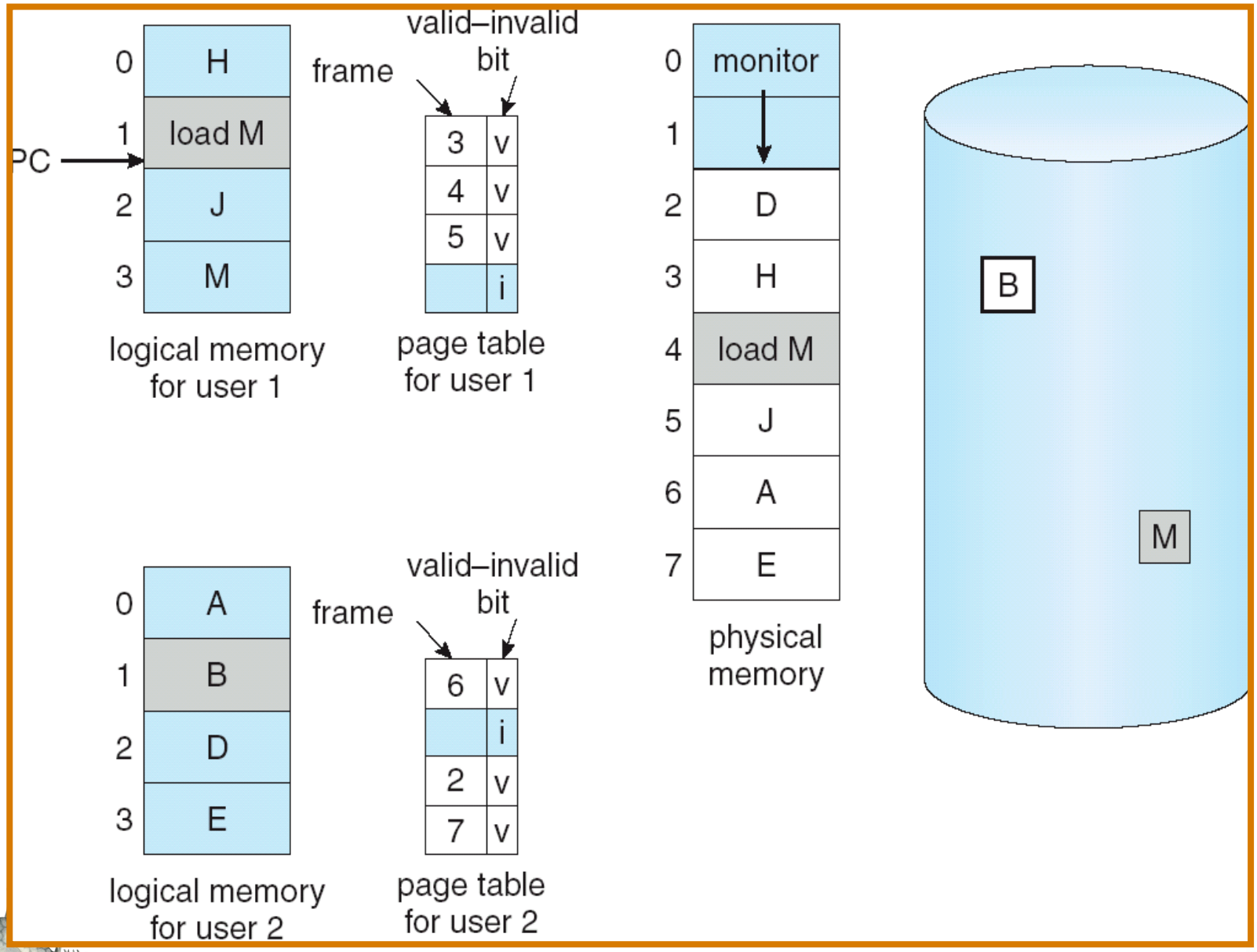


Reemplazo de página

- Prevenir sobre-asignación de memoria modificando rutina de servicio de falta-página para que incluya reemplazo de página
- Utiliza modify (dirty) bit para reducir la carga de transferencias de páginas – *sólo páginas modificadas son escritas a disco*
- Reemplazo de página completa la separación entre memoria lógica y física – memoria virtual grande se puede ofrecer en una memoria física pequeña.

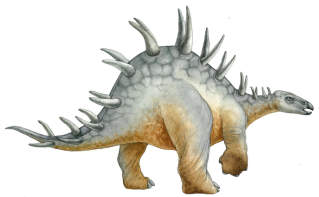


Reemplazo de página necesario

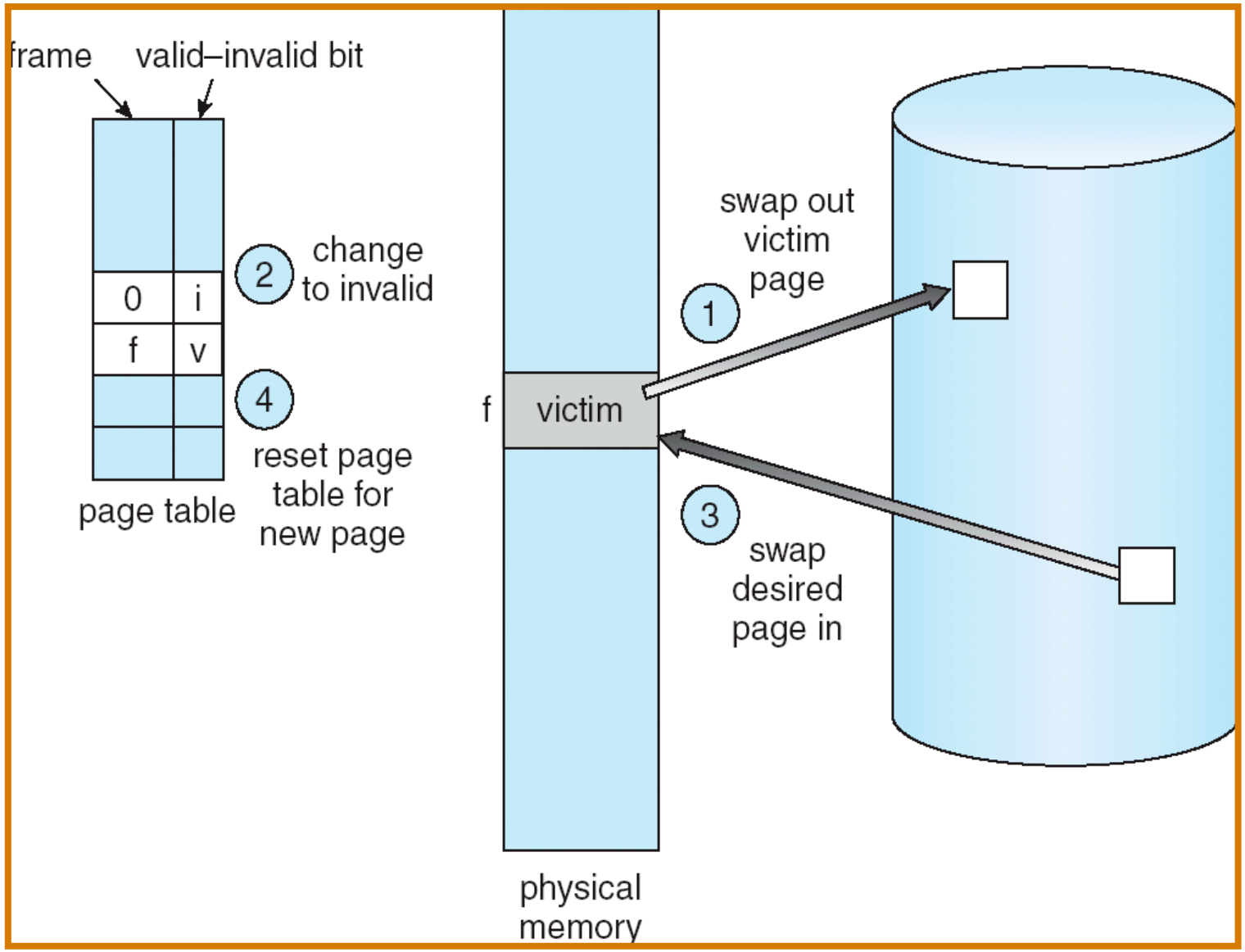


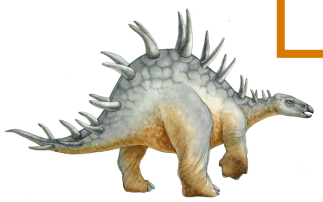
Reemplazo de página básico

1. Encontrar la localidad de la página deseada en disco
2. Encontrar un frame libre:
   i. Si existe un frame libre, úsalo
   ii. Si no hay frame libre, utiliza algoritmo de reemplazo para seleccionar frame víctima
3. Traer la página deseada al (recién creado) frame libre; actualiza las tablas de páginas y frame
4. Re-inicia el proceso



Reemplazo de página

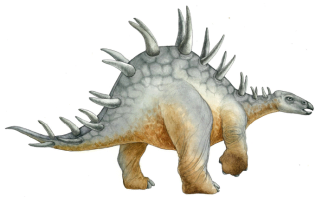




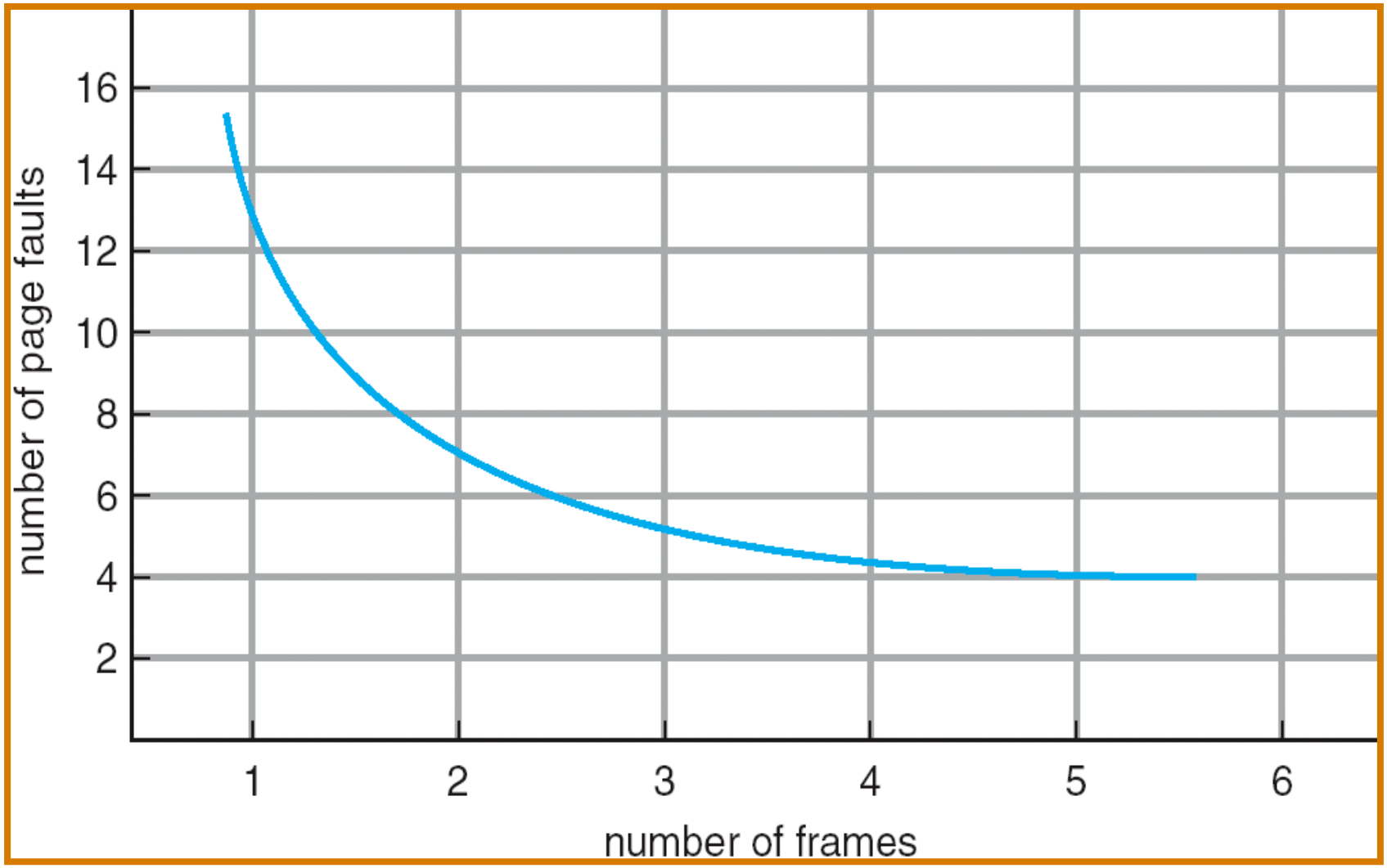
Algoritmos para reemplazo de páginas

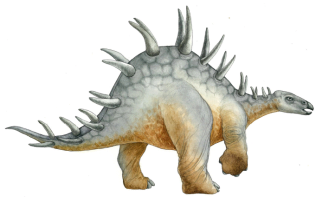
- Queremos ritmo más bajo de faltas
- Evaluar algoritmo ejecutándolo en una serie de referencias a memoria y calculando el número de faltas de página en esa serie
- En todos nuestros ejemplos, la serie de referencias es

  1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5



Gráfica de Faltas de Página Vs. Número de Frames





Algoritmo First-In-First-Out (FIFO)

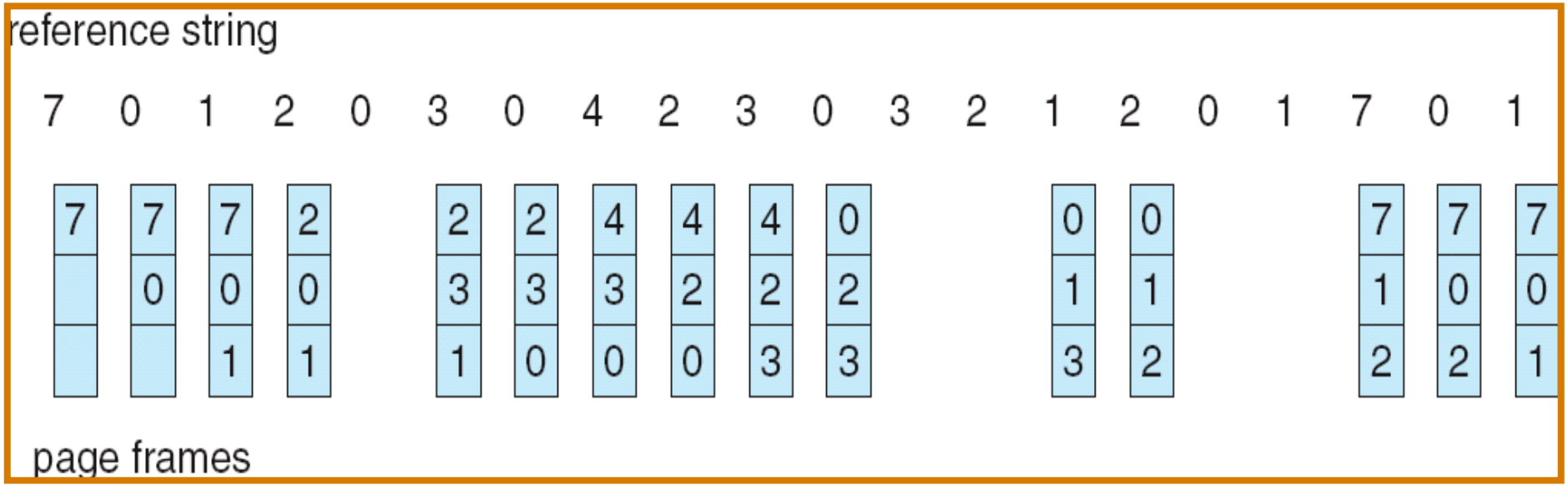
- Serie: 1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5
- 3 frames (3 páginas pueden estar en memoria)
- 4 frames

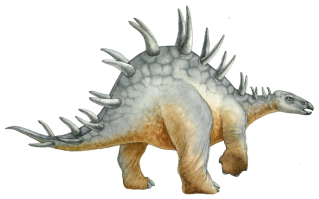
1	1	4	5	
2	2	1	3	9 faltas de página
3	3	2	4	

1	1	5	4	
2	2	1	5	10 faltas de página
3	3	2		
4	4	3		

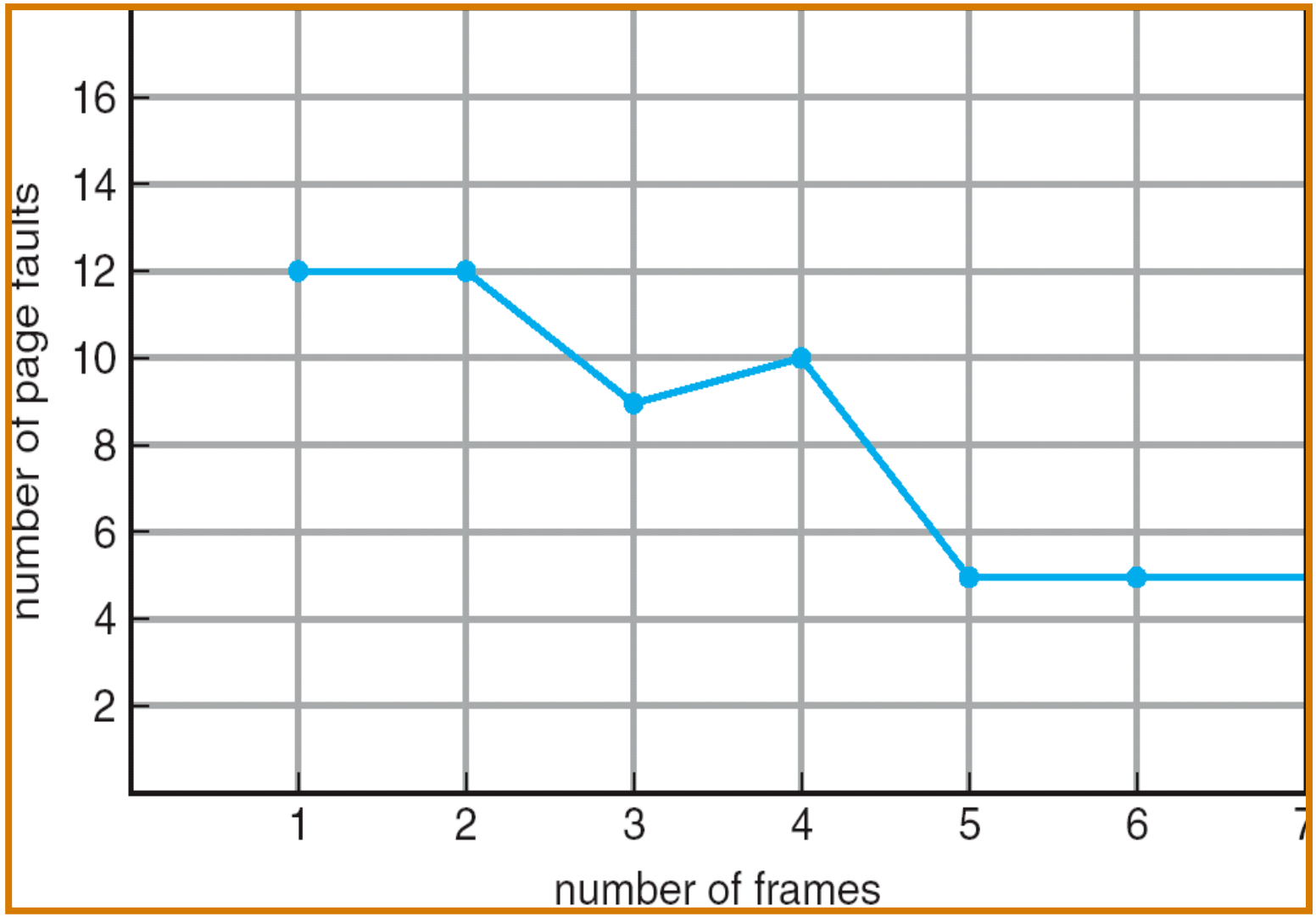
- Anomalía de Belady: más frames ⇒ más faltas de página

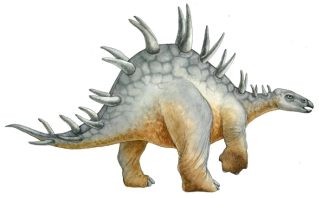
FIFO Reemplazo de Página





FIFO Ilustrando Anomalía de Belady





Algoritmo Óptimo

- Reemplazar página que NO será utilizada en el periodo más largo
  - Ejemplo con 4

1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5

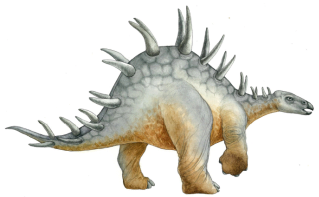
1
2
3
4

4

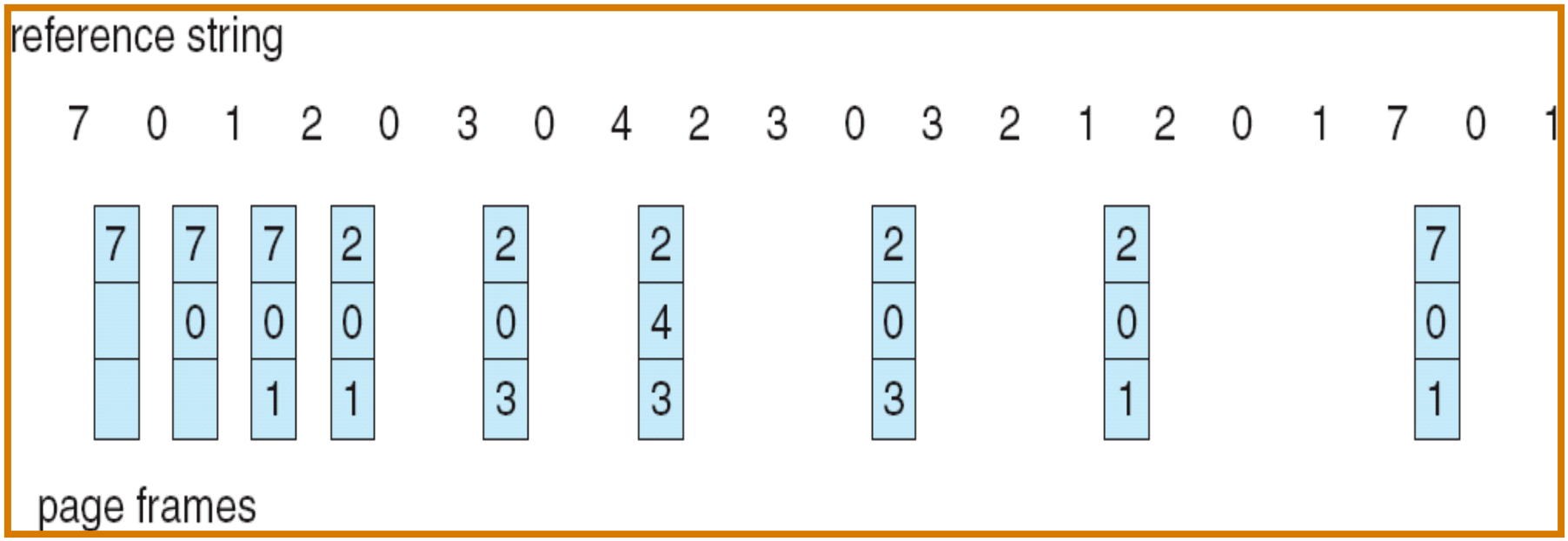
5

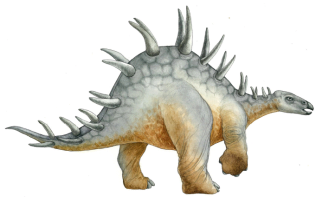
6 faltas de página

  - ¿Cómo sabemos esto? ;-)
  - Utilizado para saber qué tan bueno es tu algoritmo



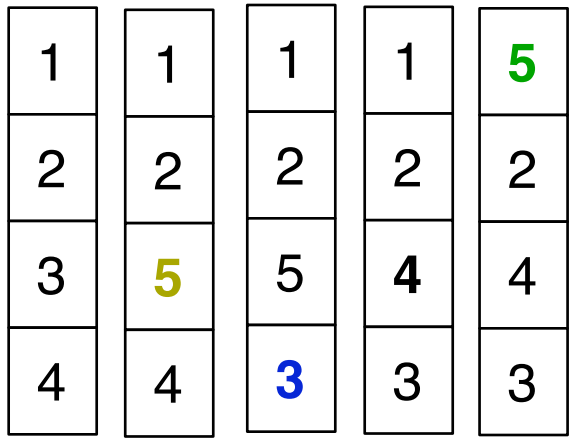
Reemplazo de página óptimo



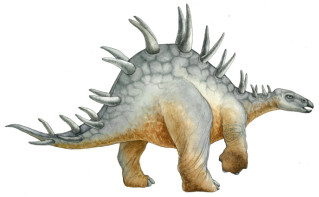


Algoritmo Menos Recientemente Utilizado (LRU)

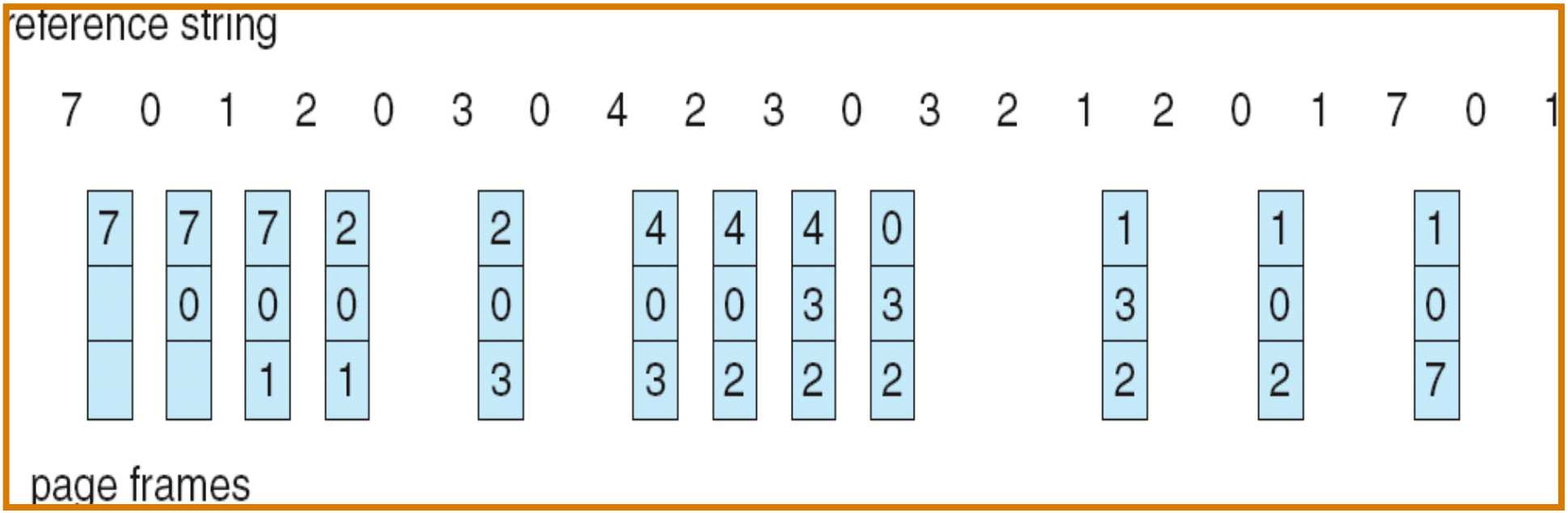
- Serie: 1, 2, 3, 4, 1, 2, **5**, 1, 2, **3**, **4**, **5**

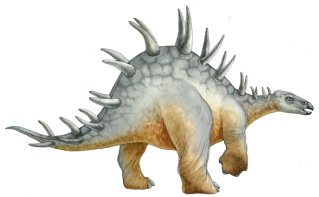


- Implementación con contador
  - Cada entrada de página tiene un contador; cada vez que es referida, copia el reloj al contador
  - Cuando una página requiere ser cambiada, revisa contadores para determinar cual cambiar



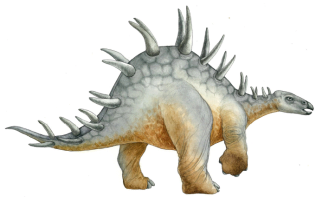
Reemplazo de página LRU



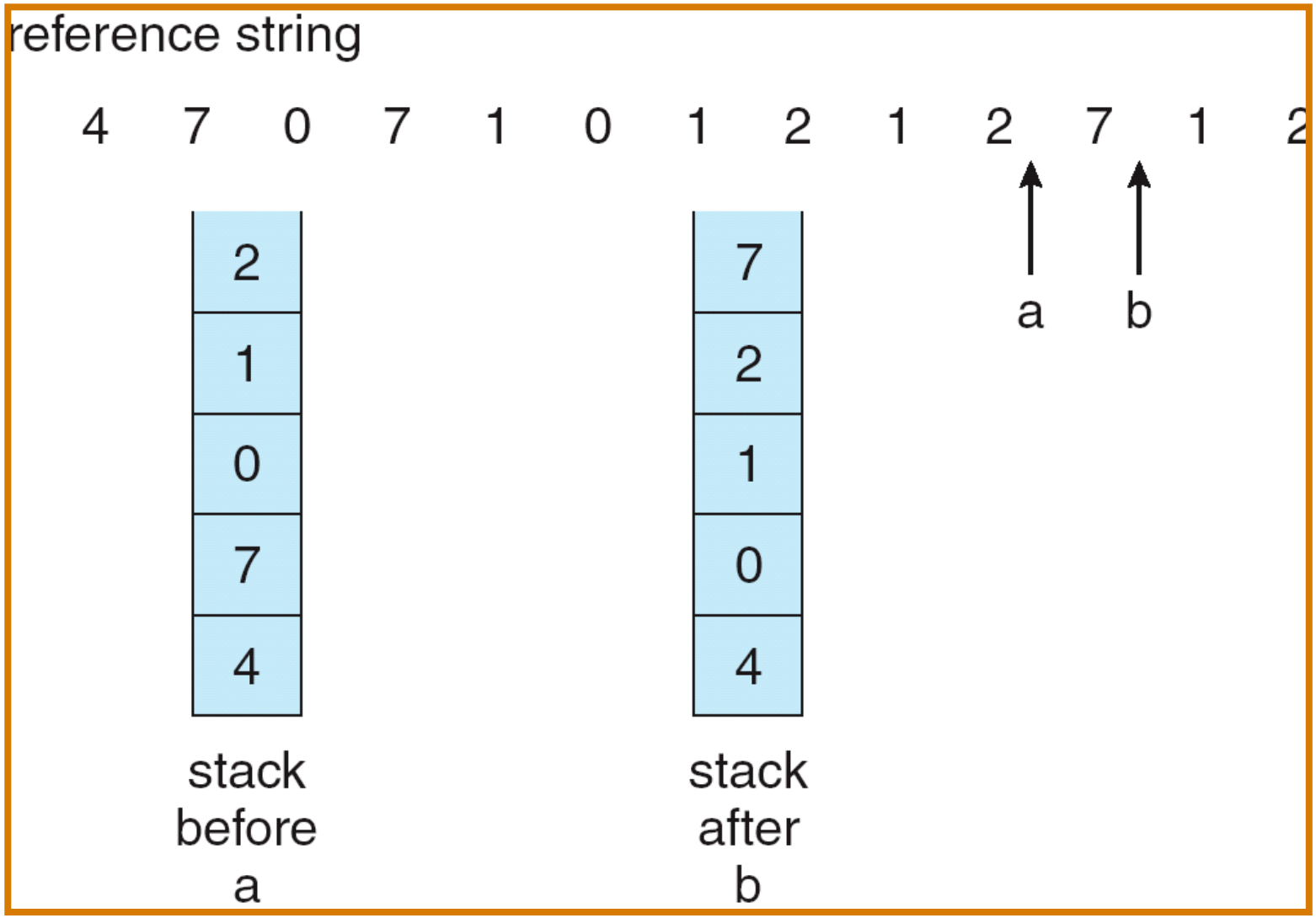


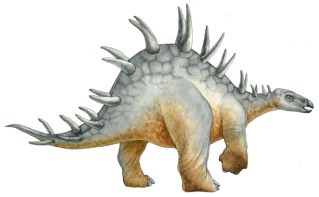
Algoritmo LRU (Cont.)

- Implementación con Stack – mantener un stack de números de página doblemente ligados:
  - Página referida:
    - Moverla al tope
    - Requiere cambiar 6 apuntadores
  - No hay búsqueda para el reemplazo



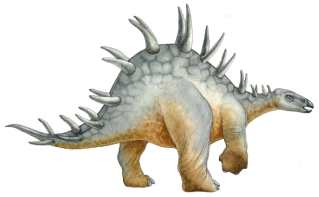
Utilizar un stack para grabar Referencias a la Página Más Reciente



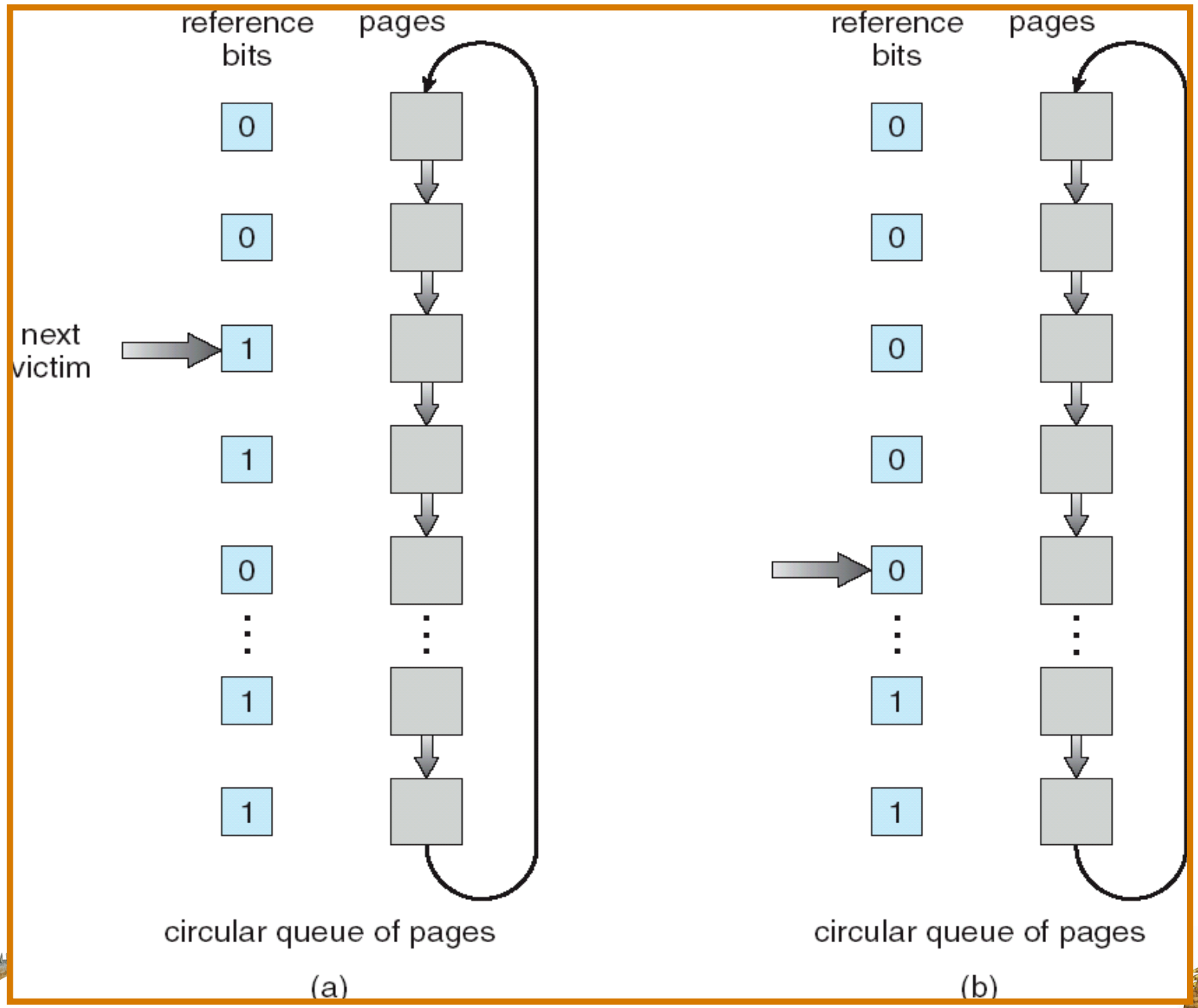


Algoritmos para aproximar LRU

- Bit de referencia
  - Asociar un bit con cada página, inicialmente = 0
  - Cuando hay referencia a la página asignar 1 al bit
  - Reemplazar el que es 0 (si existe)
    - Sin embargo, no conocemos el orden
- Segunda oportunidad
  - Requiere bit de referencia
  - Reemplazo de reloj
  - Si página a reemplazar (orden de reloj) tiene bit=1, entonces:
    - asignar 0 al bit de referencia
    - dejar página en memoria
    - reemplazar la siguiente página (orden de reloj), mismas reglas

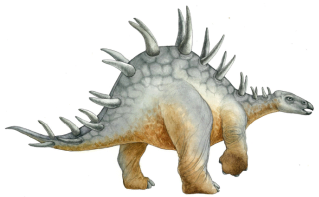


Segunda-oportunidad Algoritmo de Reemplazo-Página



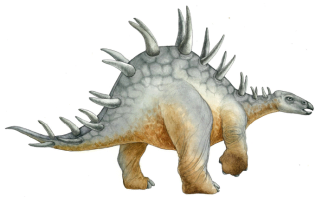
Algoritmo de cuenta

- Mantener un contador del número de referencias hechas a cada página
- Algoritmo LFU: reemplaza la página con la cuenta más pequeña
- Algoritmo MFU: basado en el argumento que una página con la cuenta más pequeña acaba de ser traída a memoria y se utilizará otra vez



Asignación de Frames

- Cada proceso requiere número *mínimo* de páginas
- Ejemplo: IBM 370 - 6 páginas para manejar instrucción SS MOVE:
  - instrucción tiene 6 bytes, puede ocupar 2 páginas
  - 2 páginas para manejar *from*
  - 2 páginas para manejar *to*
- Dos esquemas de asignación típicos
  - asignación fija
  - asignación por prioridad



Asignación fija

- Asignación igual – Por ejemplo, si hay 100 frames y 5 procesos, darle 20 frames a cada proceso.
- Asignación proporcional – Asignar de acuerdo al tamaño del proceso
  - s_i = size of process p_i
  - $S = \sum s_i$
  - m = total number of frames
  - a_i = allocation for $p_i = \frac{s_i}{S} \times m$

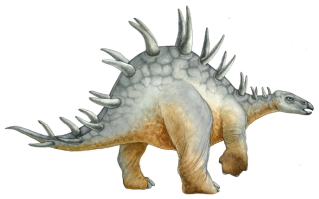
$$m = 64$$

$$s_i = 10$$

$$s_2 = 127$$

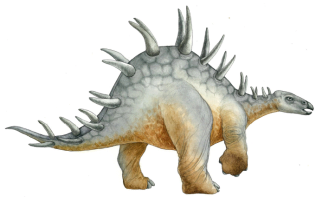
$$a_1 = \frac{10}{137} \times 64 \approx 5$$

$$a_2 = \frac{127}{137} \times 64 \approx 59$$



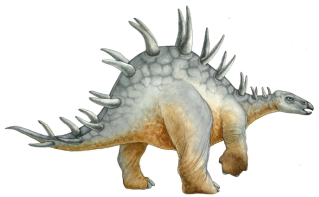
Asignación prioritaria

- Utilizar un esquema de asignación proporcional utilizando prioridades en lugar de tamaño
- Si el proceso P_i genera una falta de página,
  - selecciona para reemplazo una de sus frames
  - seleccionar para reemplazo un frame de un proceso de menor prioridad



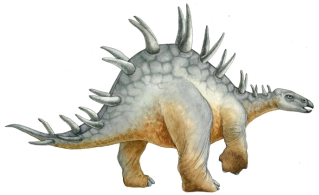
Asignación Global vs. Local

- Reemplazo Global – proceso selecciona un frame de reemplazo de todo el conjunto de frames; un proceso puede tomar un frame de otro
- Reemplazo Local – cada proceso selecciona de los frames asignados a él mismo

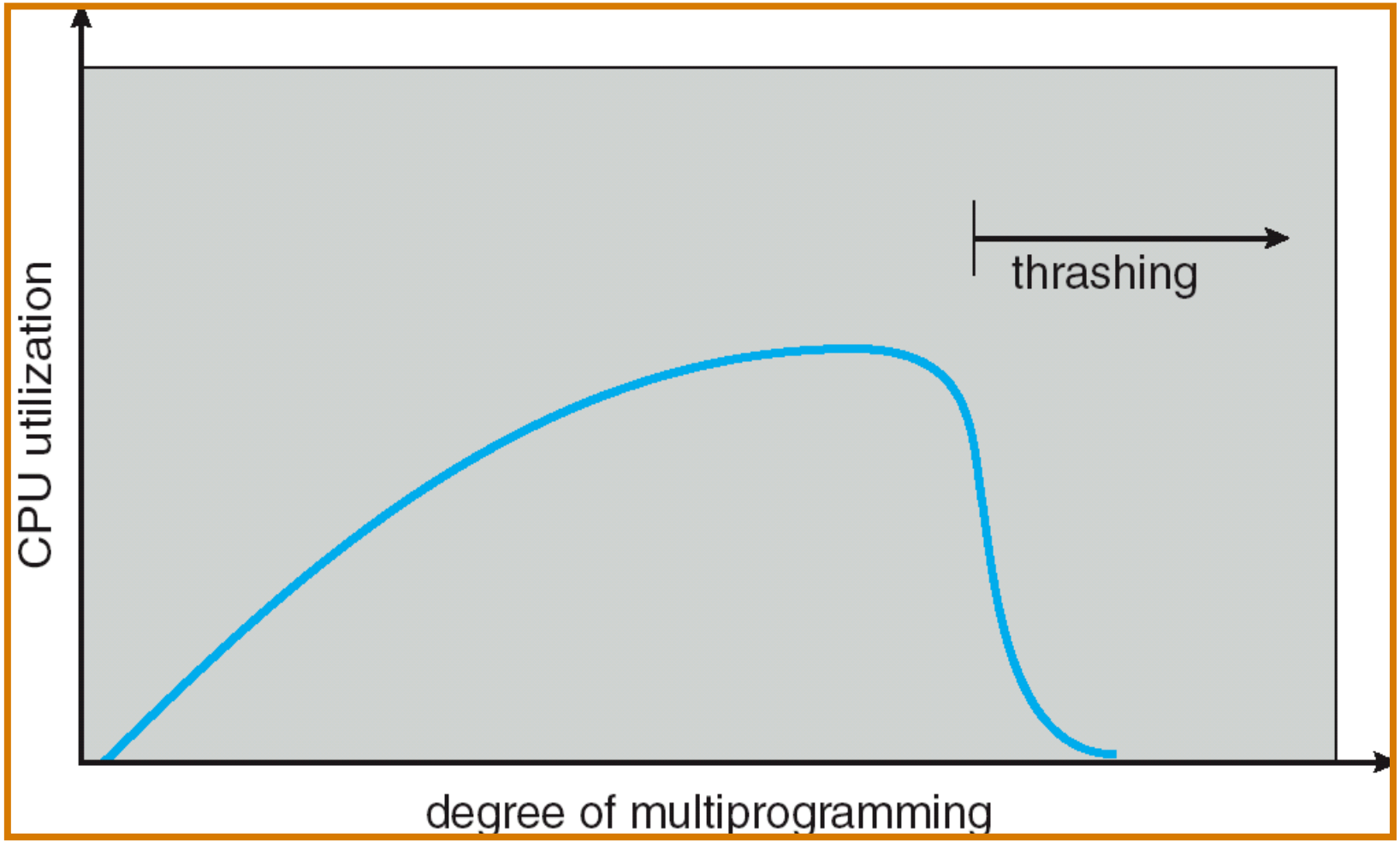


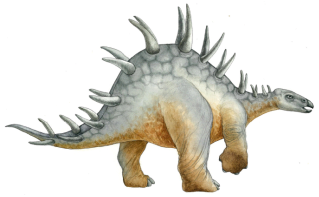
Thrashing

- Si un proceso no tiene “suficientes” páginas, el ritmo de faltas de página es muy alto. Esto produce:
  - Baja utilización de CPU
  - SO piensa que necesita incrementar el nivel de multi-programación
  - Se añade otro proceso al sistema
- Thrashing ≡ un proceso está ocupado *swapping* páginas de y hacia la memoria



Thrashing (Cont.)



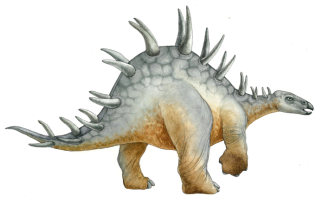


Paginación por Demanda y Thrashing

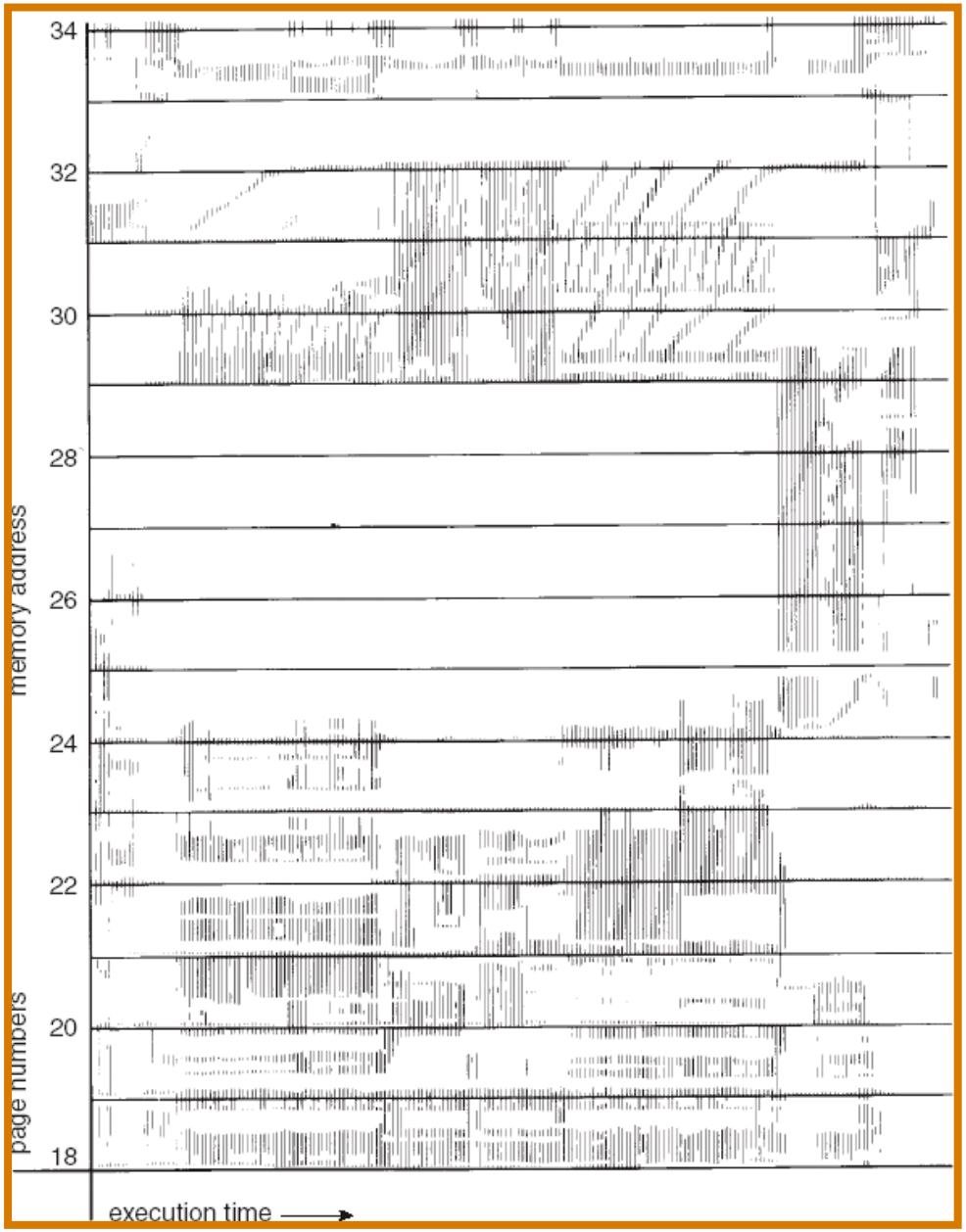
- ¿Por qué funciona paginación por demanda?

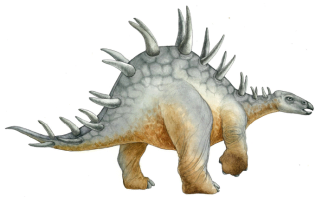
  Modelo de localidad
  - Proceso migra de una localidad a otra
  - Localidades pueden encimarse
- ¿Por qué ocurre thrashing?

  Σ tamaño localidad > tamaño memoria total



Localidad en Patrón de Referencia de Memoria

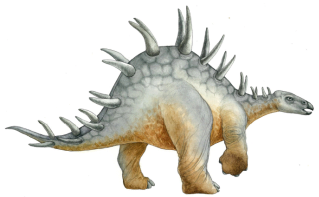




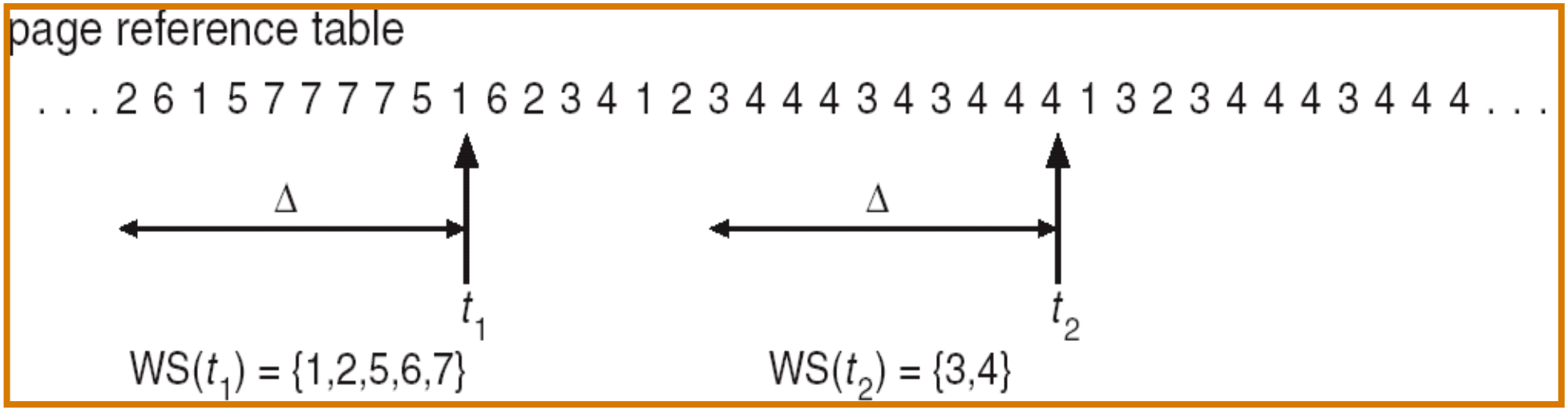
Modelo Conjunto de Trabajo

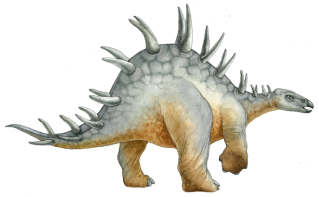
- $\Delta \equiv$ ventana conjunto-trabajo $\equiv$ un número fijo de referencias de página

  Ejemplo: 10,000 instrucciones
- WSS_i (conjunto de trabajo del proceso P_i) = número total de páginas referidas en la más reciente Δ (varia en el tiempo)
  - si Δ muy pequeña no contendrá toda la localidad
  - si Δ muy grande contendrá varias localidades
  - si $\Delta = \infty \Rightarrow$ contendrá todo el programa
- $D = \Sigma\ WSS_i \equiv$ demanda total de frames
- si $D > m \Rightarrow$ Thrashing
- Política si $D >$ m, suspender uno de los procesos



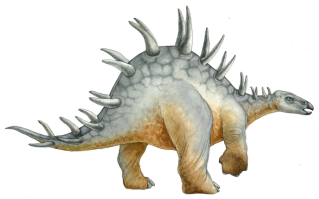
Modelo de Conjunto de Trabajo





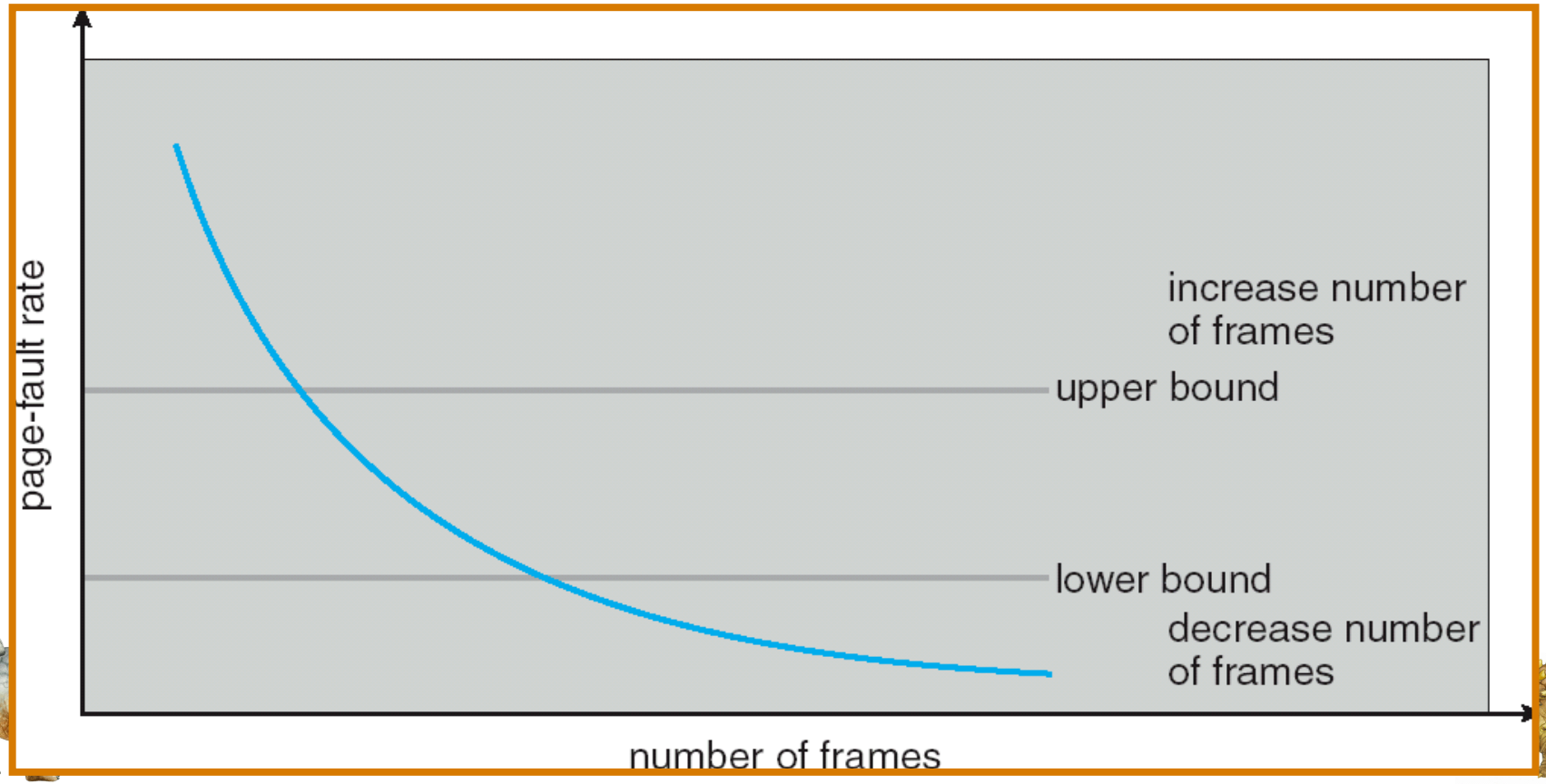
Seguimiento del Conjunto de Trabajo

- Aproximar con intervalo de tiempo + bit de referencia
- Ejemplo: Δ = 10,000
  - Temporizador interrumpe después de cada 5000 unidades de tiempo
  - Mantiene 2 bits en memoria por cada página
  - Cada vez que un temporizador interrumpe, copia y asigna los valores de todos los bits a 0
  - Si uno de los bits en memoria = 1 $\Rightarrow$ página está en el conjunto de trabajo
- Mejora = 10 bits con interrupciones cada 1000 unidades de tiempo



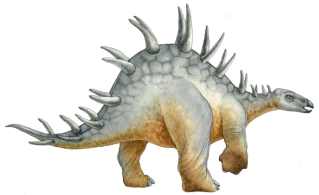
Esquema de frecuencia de faltas-página

- Establecer ritmo “aceptable” de faltas
  - Si el ritmo actual muy bajo, el proceso pierde un frame
  - Si demasiado alto, el proceso gana un frame

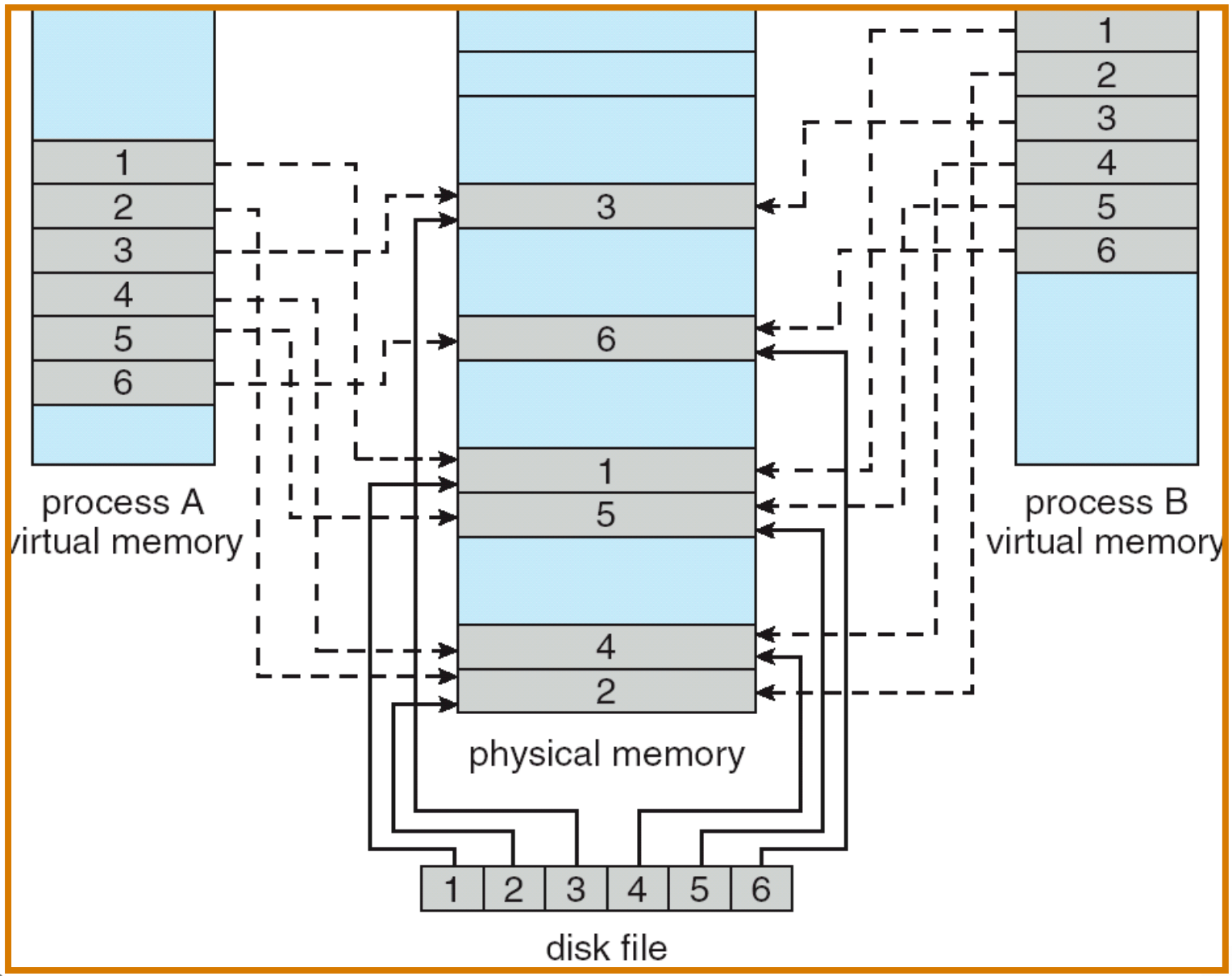


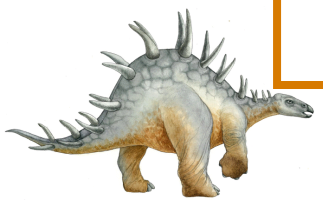
Archivos Mapeados a Memoria

- Archivos mapeados a memoria permite que un archivo E/S sea tratado como un acceso a memoria mapeando un bloque del disco a una página en memoria
- El archivo se lee inicialmente utilizando paginación por demanda. Una porción tamaño página del archivo se lee del sistema de archivos a la página física. Lecturas y escrituras subsecuentes de/hacia el archivo son tratadas como accesos a memoria ordinarios.
- Simplifica acceso tratando E/S al archivo a través de la memoria en lugar de llamadas sistema `read()` `write()`
- También permite que varios procesos mapeen el mismo archivo, permitiendo que las páginas en memoria sean compartidas

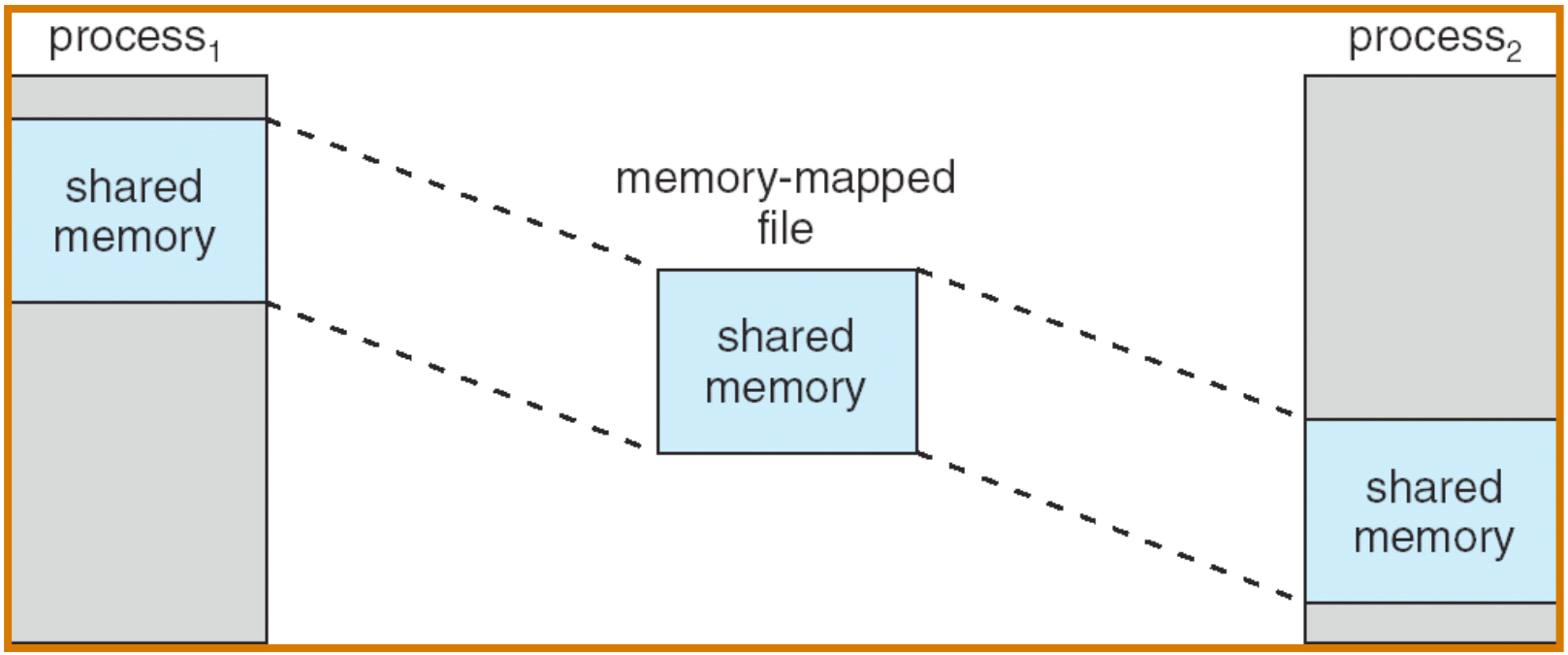


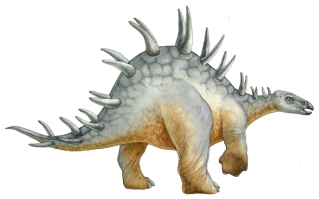
Archivos Mapeados a Memoria





Memoria Compartida Mapeada en Windows



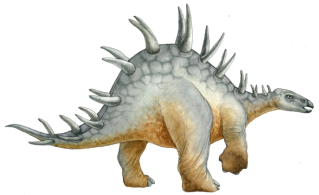


Archivos Mapeados a Memoria en Java

```
import java.io.*;
import java.nio.*;
import java.nio.channels.*;

public class MemoryMapReadOnly
{
   // Assume the page size is 4 KB
   public static final int PAGE_SIZE = 4096;

   public static void main(String args[]) throws IOException {
      RandomAccessFile inFile = new RandomAccessFile(args[0],"r");

      FileChannel in = inFile.getChannel();
      MappedByteBuffer mappedBuffer =
       in.map(FileChannel.MapMode.READ_ONLY, 0, in.size());
      long numPages = in.size() / (long)PAGE_SIZE;
      if (in.size() % PAGE_SIZE > 0)
         ++numPages;

      // we will "touch" the first byte of every page
      int position = 0;
      for (long i = 0; i < numPages; i++) {
         byte item = mappedBuffer.get(position);
         position += PAGE_SIZE;
      }

      in.close();
      inFile.close();
    }
}
```

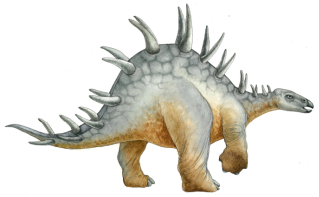
Asignando Memoria de Kernel

- Se maneja distinto a la memoria de usuario
- Usualmente asignada de un pool de memoria libre
  - Kernel solicita memoria para estructuras de distintos tamaños
  - Parte de la memoria del Kernel requiere ser contigua

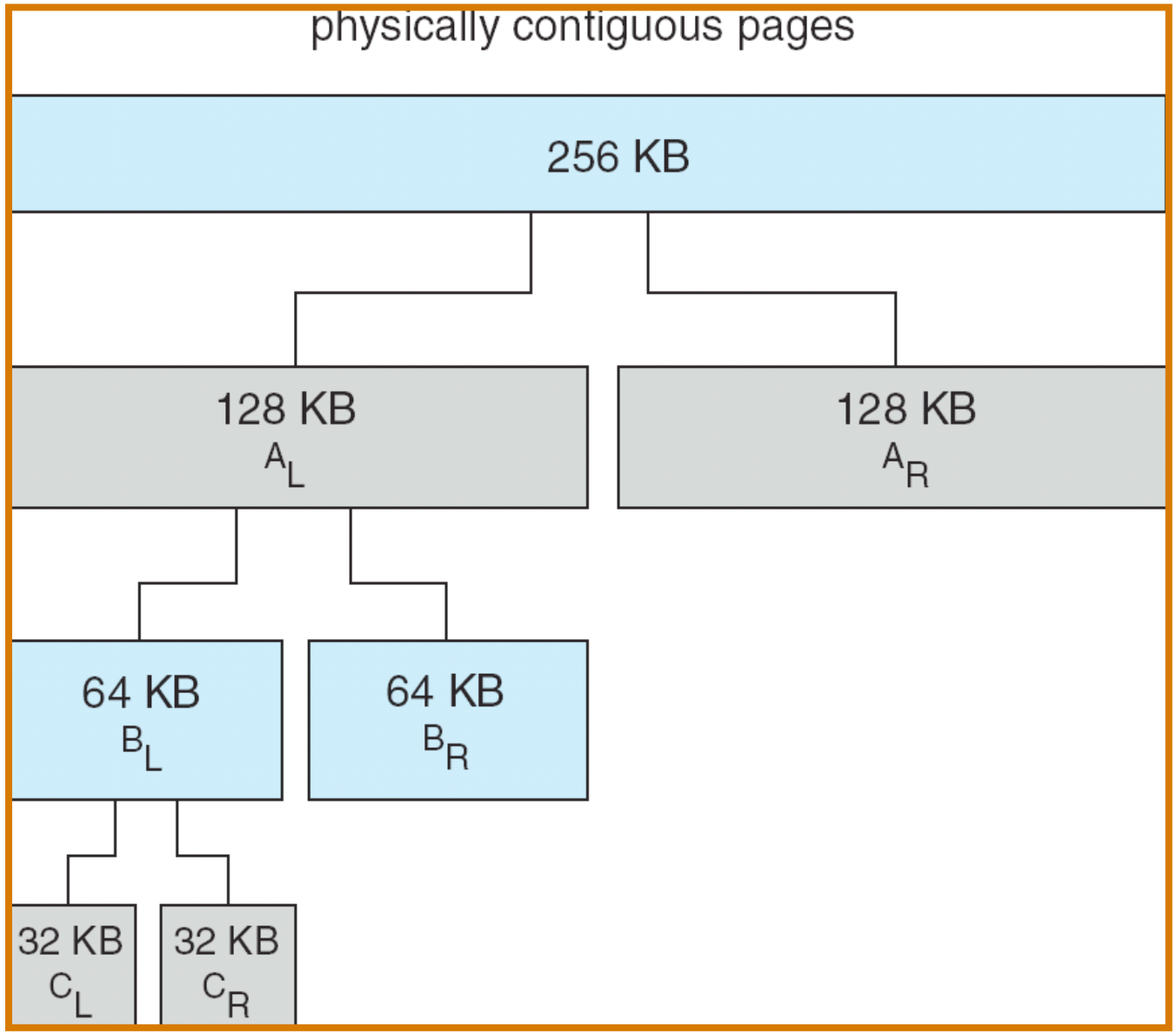


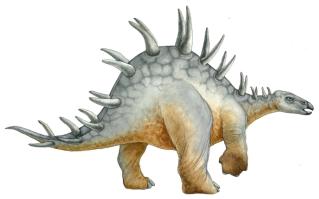
Sistema Amigo (Buddy)

- Asigna memoria de un segmento de tamaño fijo que consiste de páginas contiguas (físicamente)
- Memoria se asigna con **asignador potencia-de-2**
  - Satisfacer solicitudes en unidades con tamaños potencia de 2
  - Solicitud se redondea hasta la siguiente potencia de 2 mayor
  - Cuando se requiere asignación menor que la disponible, el pedazo actual se divide en dos de la potencia de 2 anterior
    - Continúa así hasta encontrar el tamaño apropiado



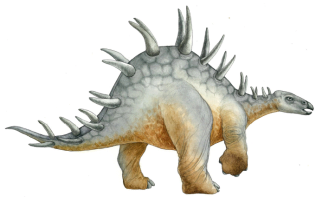
Asignador del sistema amigo



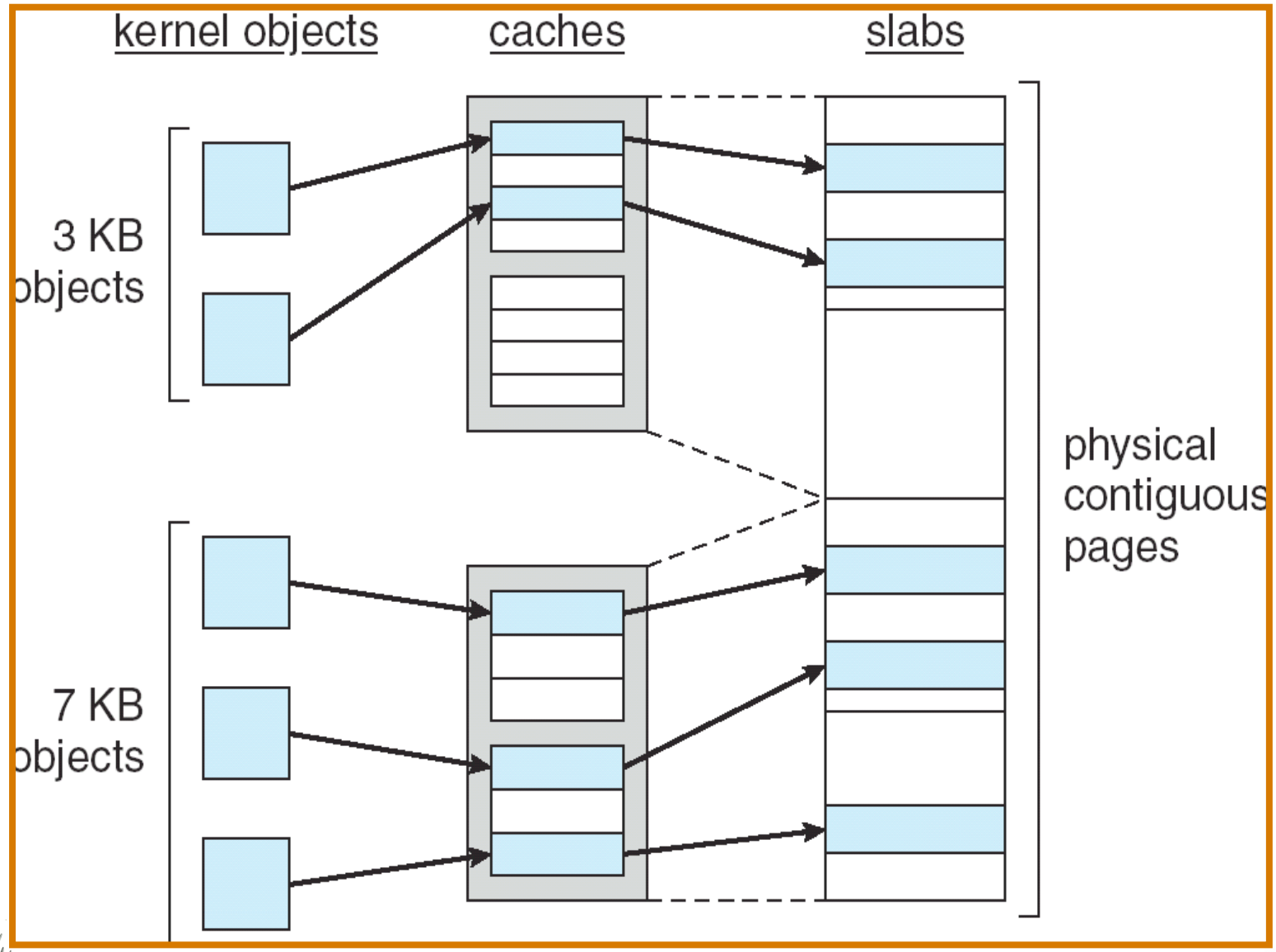


Asignador de losa (slab)

- Estrategia alternativa
- Losa es una o más páginas contiguas físicamente
- Cache consiste de una o más losas
- Un cache por cada estructura de datos del kernel
  - Cada cache se llena de objetos - instancias de la estructura de datos
- Cuando se crea el cache, se llena de objetos libres
- Cuando se almacenan estructuras, se marcan objetos como usados
- Si la losa está llena de objetos utilizados, el siguiente objeto se asigna desde una losa vacía
  - Si no hay losas vacías, se asigna una nueva losa
- Beneficios: no fragmentación y rápida satisfacción de solicitudes de memoria

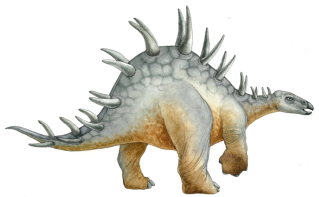


Asignación de losa



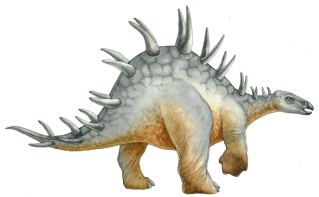
Otros asuntos -- Pre-paginación

- Pre-paginación
  - Para reducir el gran número de faltas de página que ocurren al inicio del proceso
  - Pre-paginar todas o algunas de las páginas que necesitará un proceso, antes de que sean referidas
  - *Pero* si las páginas pre-paginadas no se utilizan, se desperdicia E/S y memoria
  - Asume ***s*** páginas son *pre-paginadas* y ***α*** *utilizadas*
    - ¿El costo de ***s * α*** faltas de página ahorradas > o < que el costo de pre-paginar?
      *¿s * (1- α) páginas innecesarias?*
    - *α* cercano a cero ⇒ pre-paginar pierde



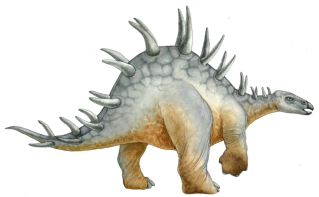
Otros asuntos – Tamaño de Página

- La selección del tamaño de página debe tomar en consideración:
  - fragmentación
  - tamaño de tabla
  - carga de E/S
  - localidad



Otros asuntos – Alcance de TLB

- Alcance TLB - Cantidad de memoria accesible desde TLB
- Alcance TLB = (Tamaño TLB) X (Tamaño de Página)
- Idealmente, el conjunto de trabajo de cada proceso se almacena en el TLB
  - De otra forma hay un grado alto de faltas de página
- Incrementar el Tamaño de Página
  - Puede aumentar la fragmentación, ya que no toda aplicación requiere un tamaño grande de página
- Proveer múltiples Tamaños de Página
  - Esto permite que aplicaciones utilizan un tamaño adecuado de página, sin aumentar la fragmentación



Otros asuntos – Estructura de Programa

- Estructura de Programa
  - `int[128,128] data;`
  - Cada renglón se almacena en una página
  - Programa 1

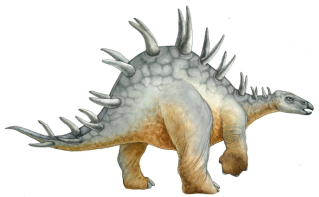
```
for (j = 0; j <128; j++)
    for (i = 0; i < 128; i++)
        data[i,j] = 0;
```

128 x 128 = 16,384 faltas de página

  - Programa 2

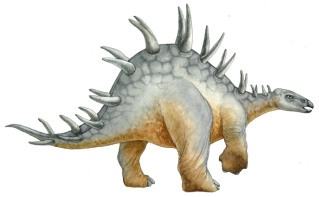
```
for (i = 0; i < 128; i++)
    for (j = 0; j < 128; j++)
        data[i,j] = 0;
```

128 faltas de página

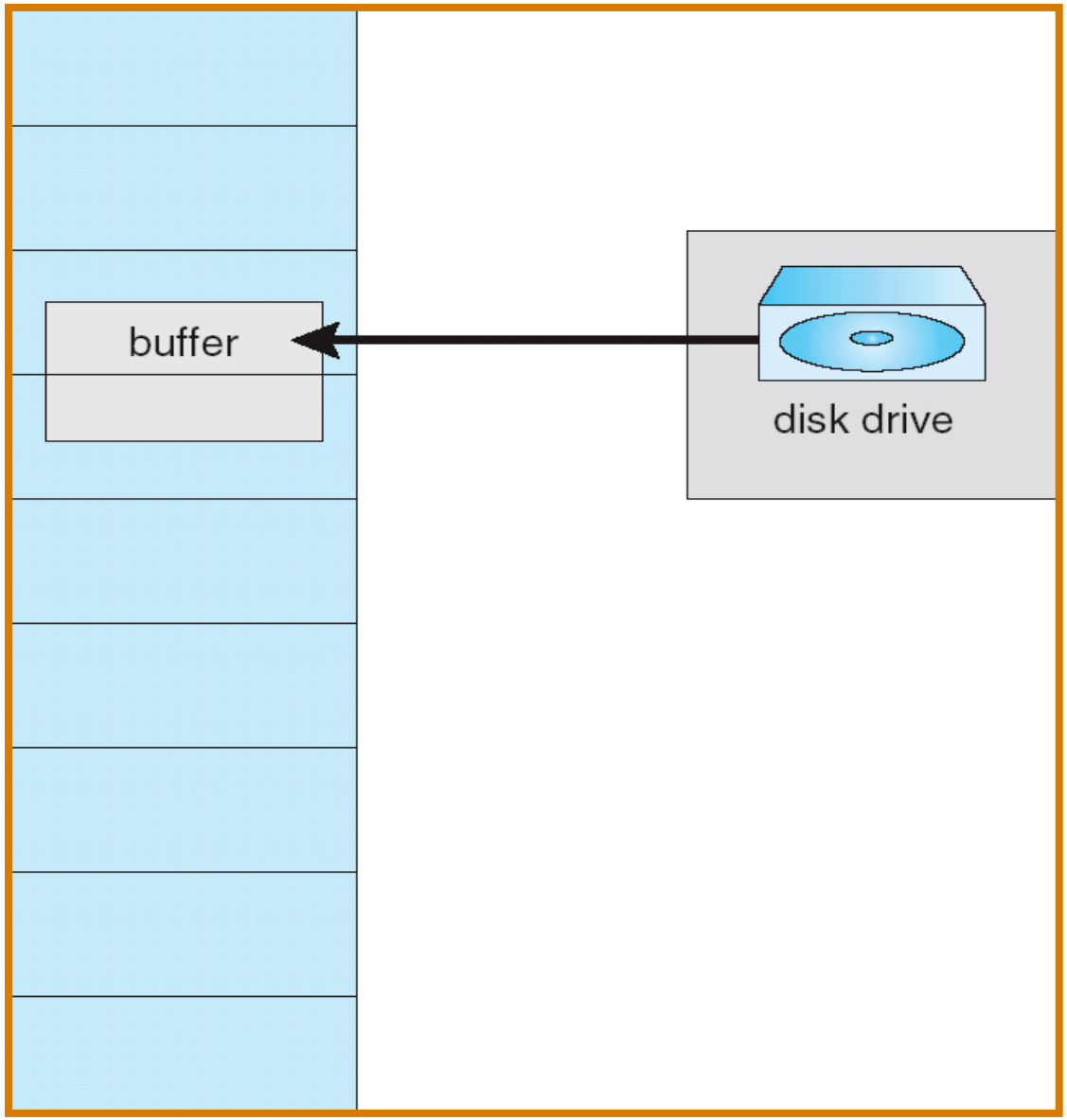


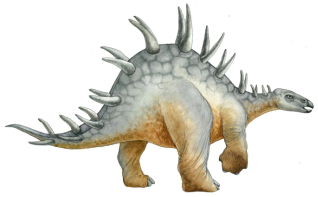
Otros asuntos – E/S interlock

- **E/S Interlock** – Algunas veces deben "bloquearse" en memoria algunas páginas
- Considerar E/S - Páginas que son utilizadas para copiar un archivo de un dispositivo deben bloquearse para evitar que sean "sacadas" por el algoritmo de reemplazo de páginas



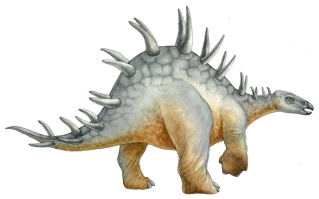
Razón p/Frames de E/S deben estar en Memoria





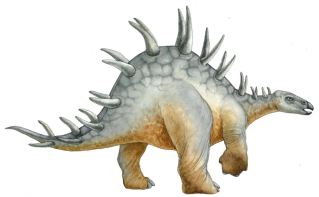
Ejemplos de Sistemas Operativos

- Windows XP
- Solaris



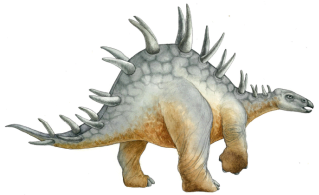
Windows XP

- Utiliza paginación por demanda para clustering. Clustering trae páginas alrededor de la que hace una falta.
- Se asigna un conjunto de trabajo mínimo a cada proceso y un conjunto de trabajo máximo
- Conjunto de trabajo mínimo es el mínimo número de páginas que el proceso tiene garantizadas en memoria
- Se pueden asignar tantas páginas a un proceso hasta llegar a su conjunto de trabajo máximo
- Cuando la cantidad de memoria libre cae por debajo de un umbral, recorte automático de conjunto de trabajo se realiza para recuperar memoria
- El recorte automático del conjunto de trabajo elimina todas las páginas “extras” de procesos que tengan más de sus mínimos

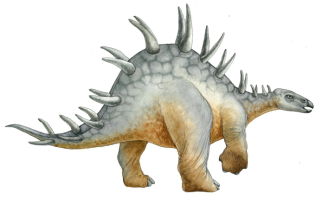


Solaris

- Mantiene una lista de páginas libres para asignar a procesos con faltas de páginas
- *Lotsfree* – parámetro umbral (cantidad de memoria disponible) para iniciar paginación
- *Desfree* – parámetro umbral para incrementar paginación
- *Minfree* – parámetro umbral para iniciar swapping
- Paginación se realiza por el proceso *pageout*
- Pageout busca/*escanea* páginas utilizando el algoritmo de reloj (modificado)
- *Scanrate es el ritmo a que se revisan las páginas*. Esto varía de *slowscan* a *fastscan*
- Pageout se llama más frecuentemente dependiendo de la cantidad de memoria libre disponible



Solaris 2 Escáner de Páginas



Final del Capítulo 9