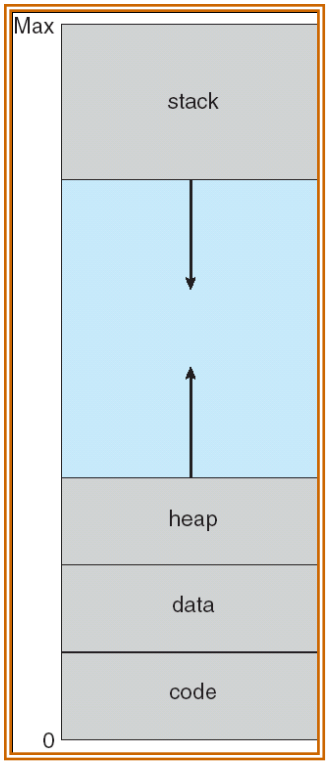
9: Memoria Virtual

Sistemas Operativos 1

Ing. Alejandro León Liu

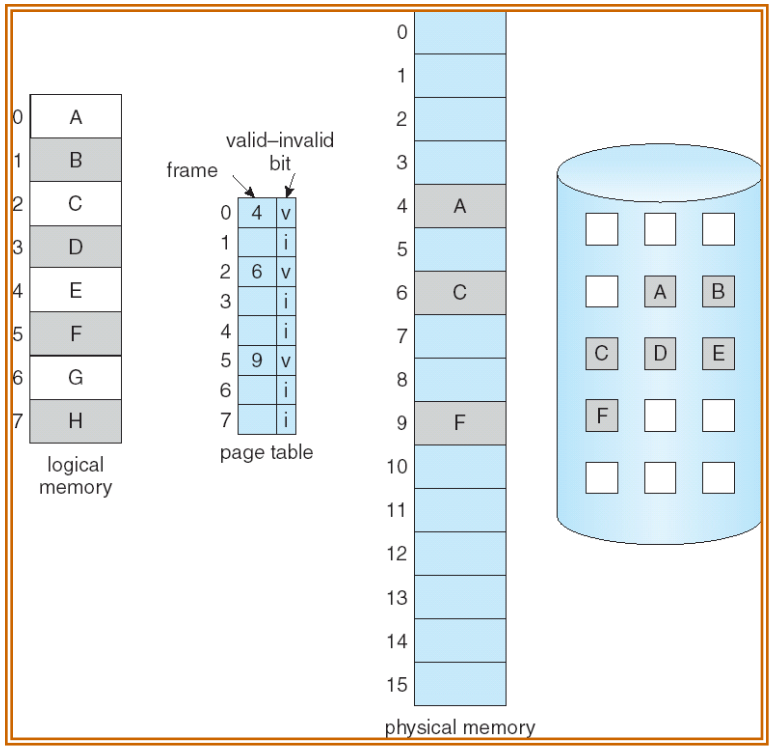
- Instrucción a ejecutar debe estar en memoria
  - Acercamiento anterior: todo el programa debe estar en memoria
    - Únicamente una parte es utilizada
      - Código siendo ejecutado (una instrucción a la vez)
      - Datos utilizados
  - Dynamic loading
    - Requiere acciones por desarrollador

- Espacio de memoria virtual
- Ejecución de un programa parcialmente en memoria
  - El programa puede ser mayor a la memoria física
  - Cada programa utilizará menor memoria
    - Permite tener más programas en memoria
      - Podría aumentar utilización de CPU
        - Menor número de procesos en input queue
    - Menor I/O para swap in/out cada programa
  - Compartir bibliotecas entre procesos
  - Compartir memoria (comunicación)
  - Fork más eficiente

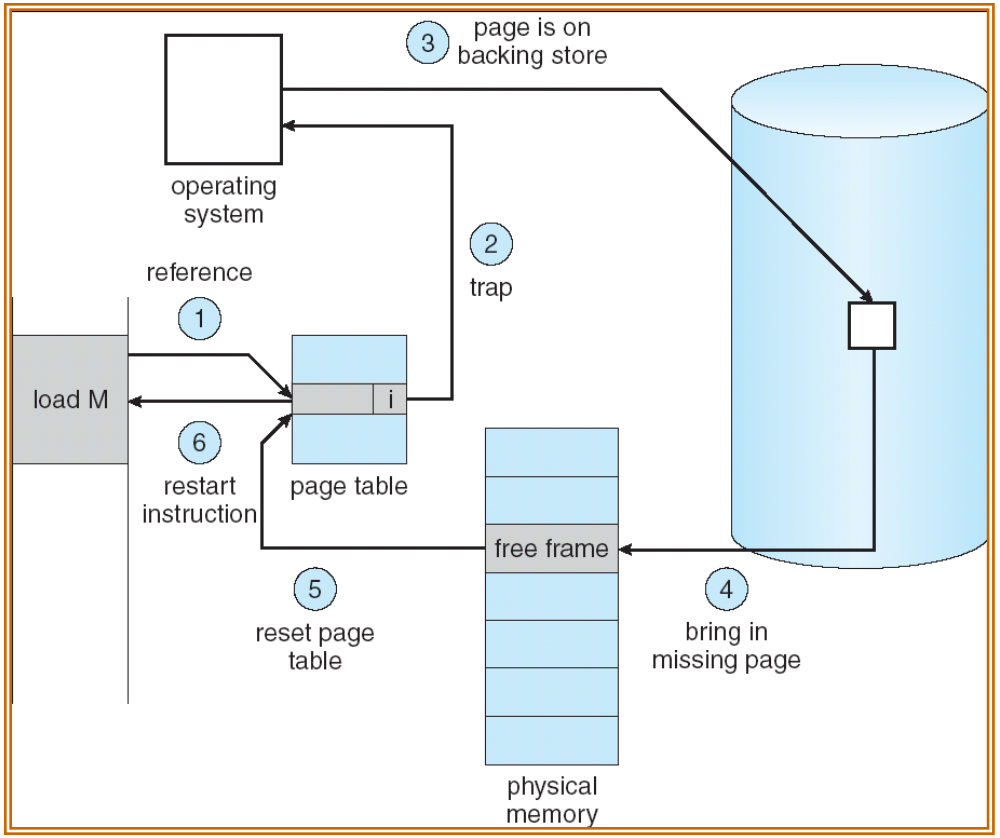


DEMAND PAGING

- Cargar únicamente las páginas que son necesitadas
- Páginas no accedidas nunca se cargan
- Swapper: Carga proceso completo
- Pager: Carga páginas
- Page Table
  - Registra si página está en memoria o disco
  - Si página está en disco
    - Page fault



- Page fault
  - Página es válida?
    - Inválida: cancelar
    - Válida: continuar
  - Obtener frame vacía
  - Traer página a memoria
  - Actualizar PCB
    - Ubicación en disco de página
  - Bit válido
  - Reiniciar instrucción



- Pure demand paging
  - Iniciar ejecución de proceso sin ninguna página
  - Peor de los casos, por cada instrucción
    - Page fault al ejecutar instrucción
    - Varias page faults para traer parámetros
    - Estadísticas muestran que es poco probable
- Archivos binarios
  - No cambian (código)
  - Al realizar swap out, se crea copia exacta a archivo original
  - Utilizar el mismo file system para swap out

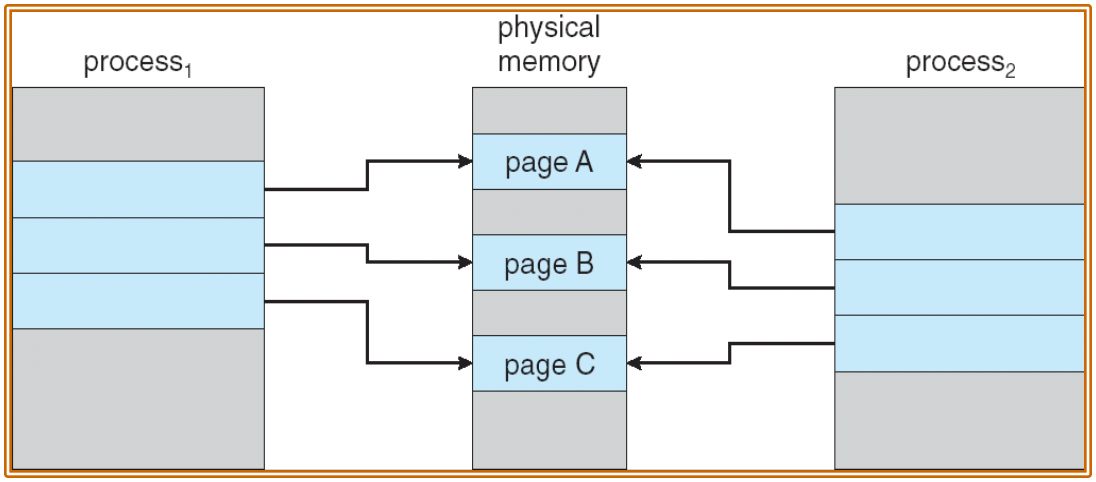
- Hardware
  - Tabla de paginación. Bits que indica si página está en memoria
  - Almacenamiento secundario (swap space)
  - Reiniciar instrucciones
    - Peor de los casos se ejecuta instrucción dos veces
    - MOVS en intel
      - Revisar que ambas direcciones estén en memoria
      - Registros temporales

- Proceso completo
  1. Page fault: Trap al S.O.
  2. Almacenar registros y estado de proceso
  3. Determinar que interrupción fue page fault (interrupt table)
  4. Revisar si página es válida y ubicación en disco
  5. Solicitar lectura a disco
     1. Esperar en cola hasta que disco procese la lectura
     2. Esperar seek y latency time
     3. Transferir página a frame disponible
  6. Mientras se espera, asignar CPU a otro proceso
  7. Recibir interrupción de disco (I/O terminada)
  8. Almacenar registros y estado de otro proceso
  9. Determinar que interrupción fue de disco (interrupt table)
  10. Actualizar page table
  11. Esperar a que el CPU vuelva a asignarse a el proceso
  12. Restaurar registros estado y page table del proceso
  13. Reiniciar instrucción

- Desempeño
  - p = probabilidad de page fault
  - Acceso a memoria efectivo
    - ema = [(1 – p) x ma] + [p x page fault time]
  - ma = 200 ns (Tiempo acceso a memoria)
  - Si p = 0, mismo tiempo de acceso a memoria
  - page fault time = 8,000,000 ns (lectura disco)
  - ema = 200 + 7,999,800 x p
- Si p = 1/1000
  - ema = 8,200 ns (40 veces el ma) !!!!
- No queremos que desempeño baje 10%
  - ema < 1.1 x ma
  - 220 > 200 + 7,999,800 x p
  - p = 1 / 399,990

COPY ON WRITE

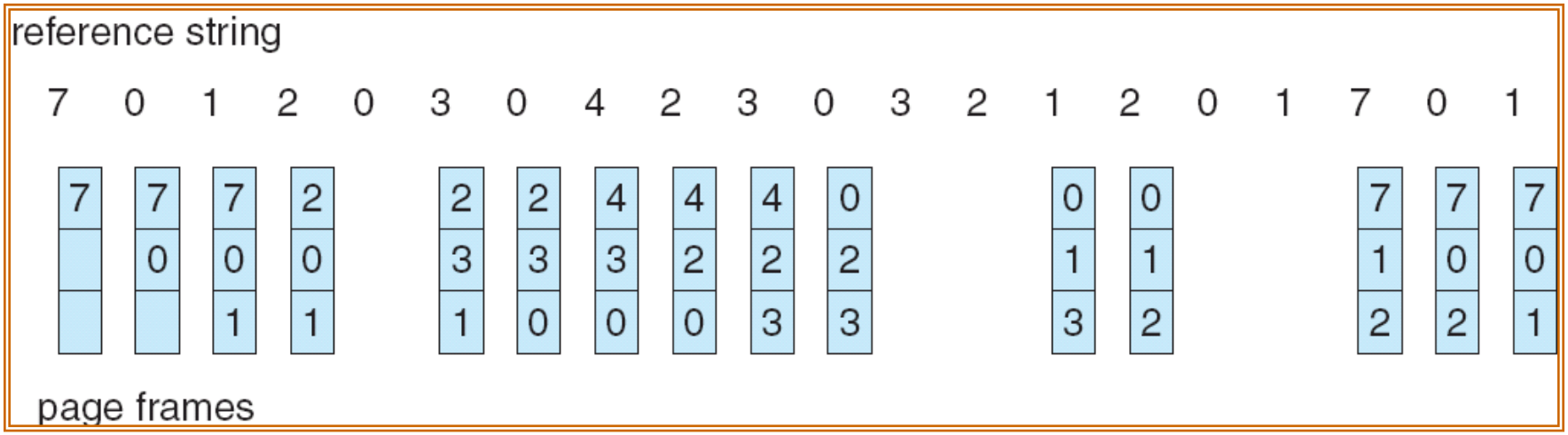
- Fork() crea proceso duplicado
  - Duplica páginas
- Exec() carga programa en proceso
  - Copia inicial fue innecesaria
- Copy on write: inicialmente no copiar (duplicar)
  - Al realizar primer escritura, duplicar página



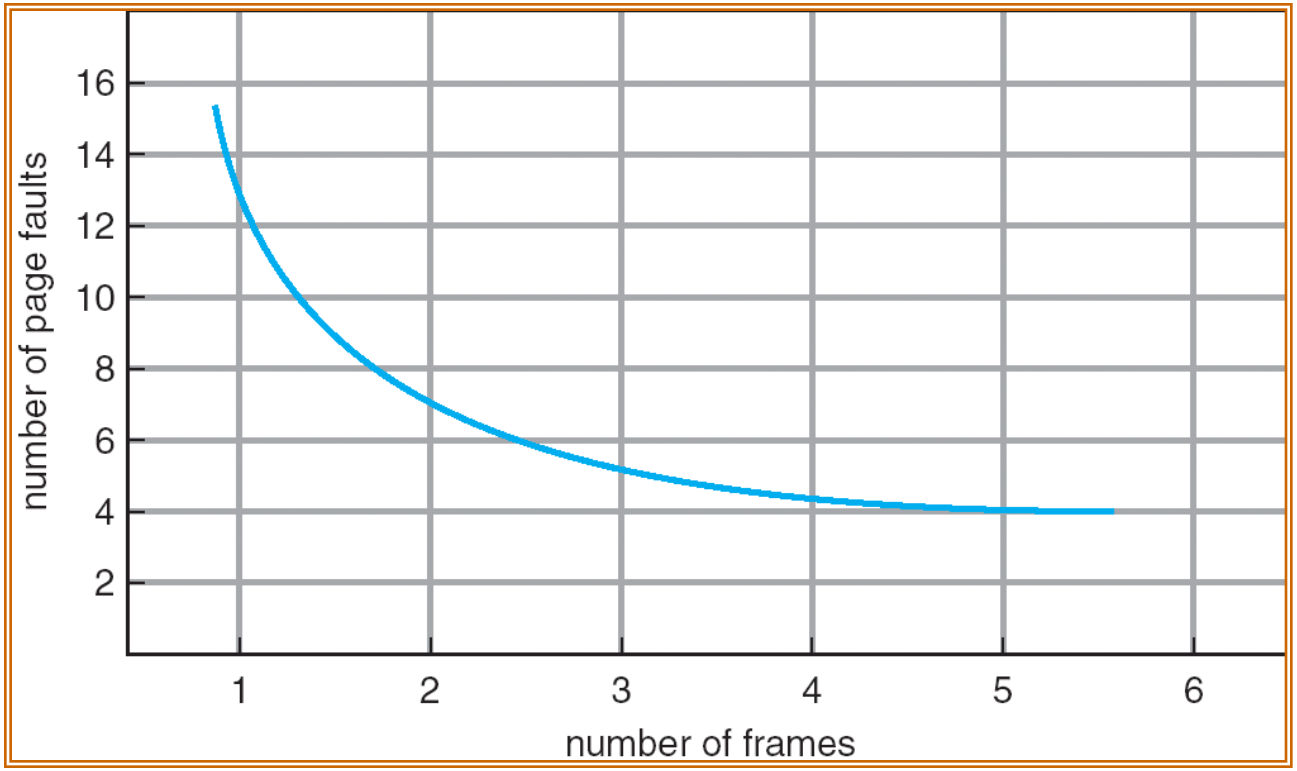
REEMPLAZO DE PÁGINAS

- Encontrar frame libre
  - Si existe frame libre, utilizarla
  - Si no hay frames libres, utilizar algoritmo de reemplazo de página para seleccionar una frame
  - Swap out de la frame
  - Actualizar frame y page tables
- Modify bit
  - Indica si una página ha sido modificada desde que se hizo swap in
  - Si la página ha sido seleccionada para ser reemplazada y no ha sido modificada, no es necesario swap-out
- Algoritmo de reemplazo de páginas

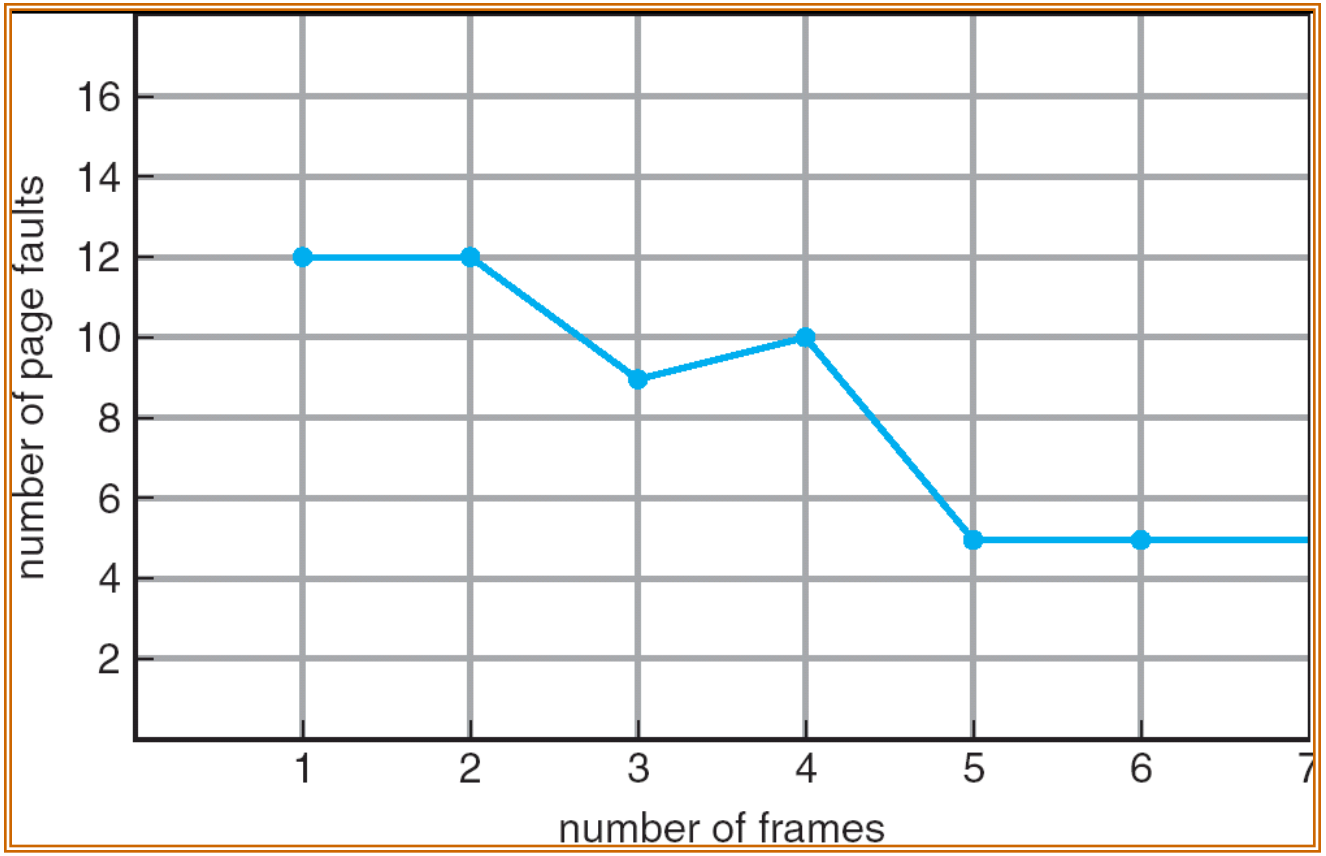
- FIFO
  - Reemplazar la página más antigua
  - Fácil de implementar
  - Probablemente ese código ya se utilizó y no volverá a utilizarse. Ej: menú inicial



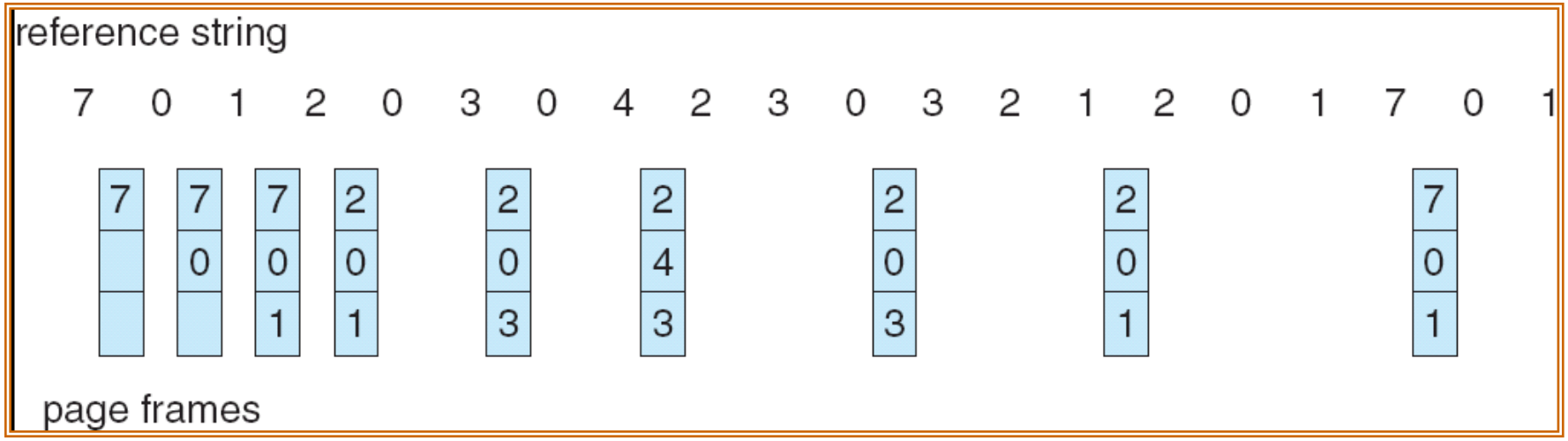
- Páginas consultadas
  - 1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5



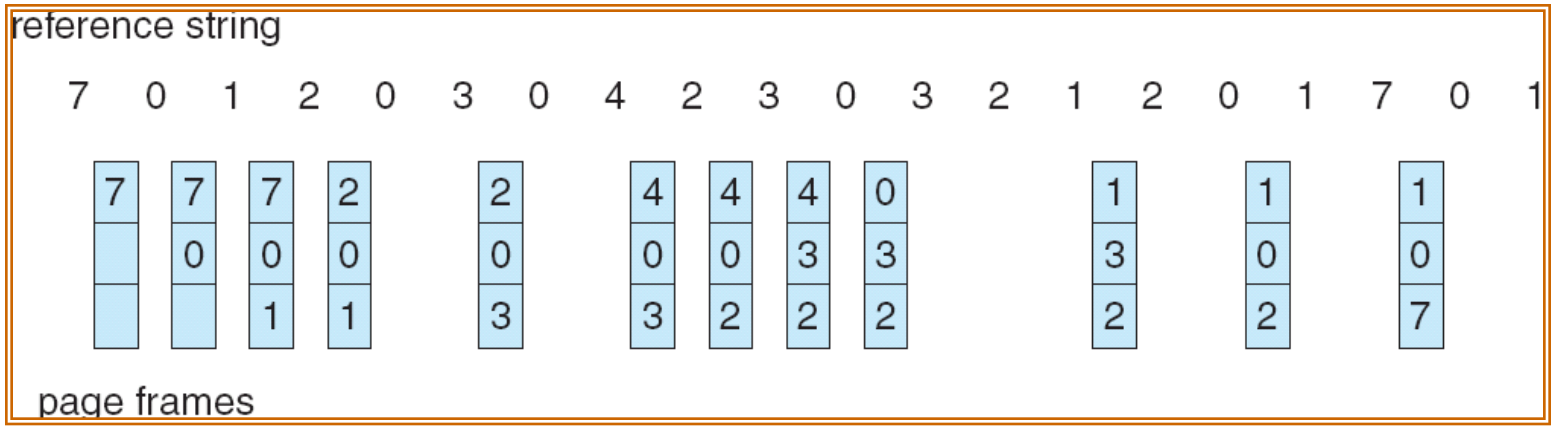
- Belody's anomaly
  - Más frames → más page faults



- Optimal page replacement
  - Número más bajo de page faults
  - No puede suceder Belady's anomaly
  - Reemplazar la página que no será utilizada por el período más largo
  - Difícil saber si una página será utilizada en un futuro



- Least Recently Used (LRU)
  - Swap out página menos utilizada recientemente
  - Misma idea que Optimal page replacement, vista al pasado



- Implementación
  - Contadores
    - En cada entrada de page table, agregar #tick
    - Reemplazar página con #tick más bajo
    - Búsqueda completa
  - Pila
    - Cada vez que una pila es referenciada, se coloca en el tope
    - El tope tendrá la página utilizada más recientemente
    - El fondo tendrá la página utilizada más antiguamente
    - Implementada como una lista doblemente encadenada
    - No hay búsqueda
    - Actualizar pila es costoso
- Debe existir soporte en hardware (eficiencia)

- Aproximaciones de LRU
  - Reference bit por cada página
  - Inicialmente todos están en 0
  - Set en 1 cuando la página es referenciada.
  - Páginas referenciadas en período de tiempo tendrán 1
  - Byte de referencia
    - 8 bits de referencia
    - Al terminar período, hacer shift derecha, despreciar bit menos significativo
    - Byte puede ser interpretado como número entero
    - Número no único, pero si indica tiempo

- Considerar el reference bit, modify bit
  - (0,0): ni usada, ni modificada: swap
  - (0,1): no usada, pero modificada. Tendrá que ser escrita a disco.
  - (1,0): utilizada, pero no modificada. Probablemente será utilizada de nuevo.
  - (1,1): utilizada y modificada. Probablemente será utilizada de nuevo y tendremos que escribir a disco

- Second Chance
  - Combinación de FIFO y reference bit
  - Revisar si la página ha sido referenciada
    - No ha sido referenciada: Swap out
    - Si ha sido referenciada: Borrar reference bit y reingresar en cola

- Reemplazo de página basado en número de referencias
  - Mantener contador de referencias de cada página
  - Least frequently used (LFU)
    - Reemplazar página menos utilizada
    - Páginas utilizadas únicamente al inicio
      - Aging: restar número de referencias con el tiempo
  - Most frequently used (MFU)
    - LFU, número de referencias es bajo porque se acaba de swap in y se utilizará a continuación
  - Ninguno se aproxima al algoritmo óptimo

ASIGNACIÓN DE FRAMES

- Asignar un número máximo de frames por proceso
- Mantener algunas frames libres para nuevos procesos.
- Número mínimo de páginas depende de arquitectura
  - 1 por instrucción
  - 1 por cada operando de memoria (x número de indirecciones)
- Número máximo definido por tamaño de RAM
- Algoritmos
  - Asignación equitativa
    - Procesos grandes y pequeños tendrán mismo número de páginas
  - Asignación proporcional
    - Posible tomar en cuenta prioridad

- Asignación local
  - Reemplazo de una página asignada a este proceso
  - Desperdicio de páginas no utilizadas (de otros procesos)
- Asignación global
  - Reemplazo seleccionado de todas las páginas
  - Un proceso no puede controlar su número de page faults
  - Mejor throughtput
  - Mayormente utilizadado

THRASHING

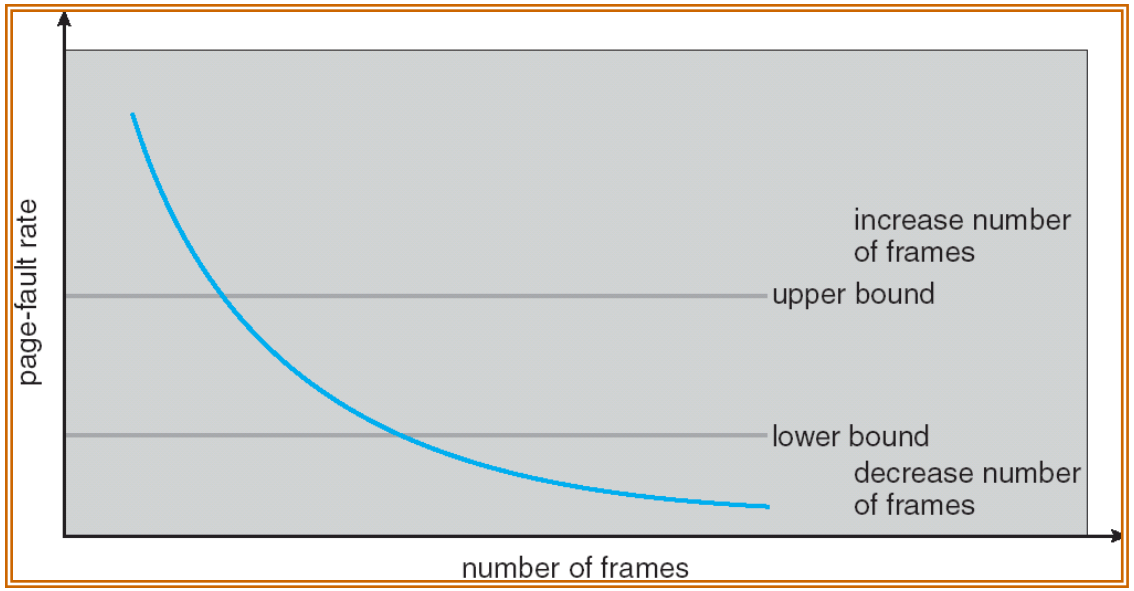
- Proceso con frecuentes page faults.
- Transcurre más tiempo paginando que ejecutándose.

- Escenario
  - Varios procesos entran al sistema
  - Un proceso necesita más y más frames,
    - Varios page faults
    - Quitar frames de otros procesos
  - Los demás procesos tambien necesitan frames
  - Ready queue se vacía
  - Baja la utilización de CPU

- Soluciones
  - Únicamente permitir asignación local
    - 1 Proceso no afecta desempeño de otros procesos
    - Proceso continuará con varios page faults

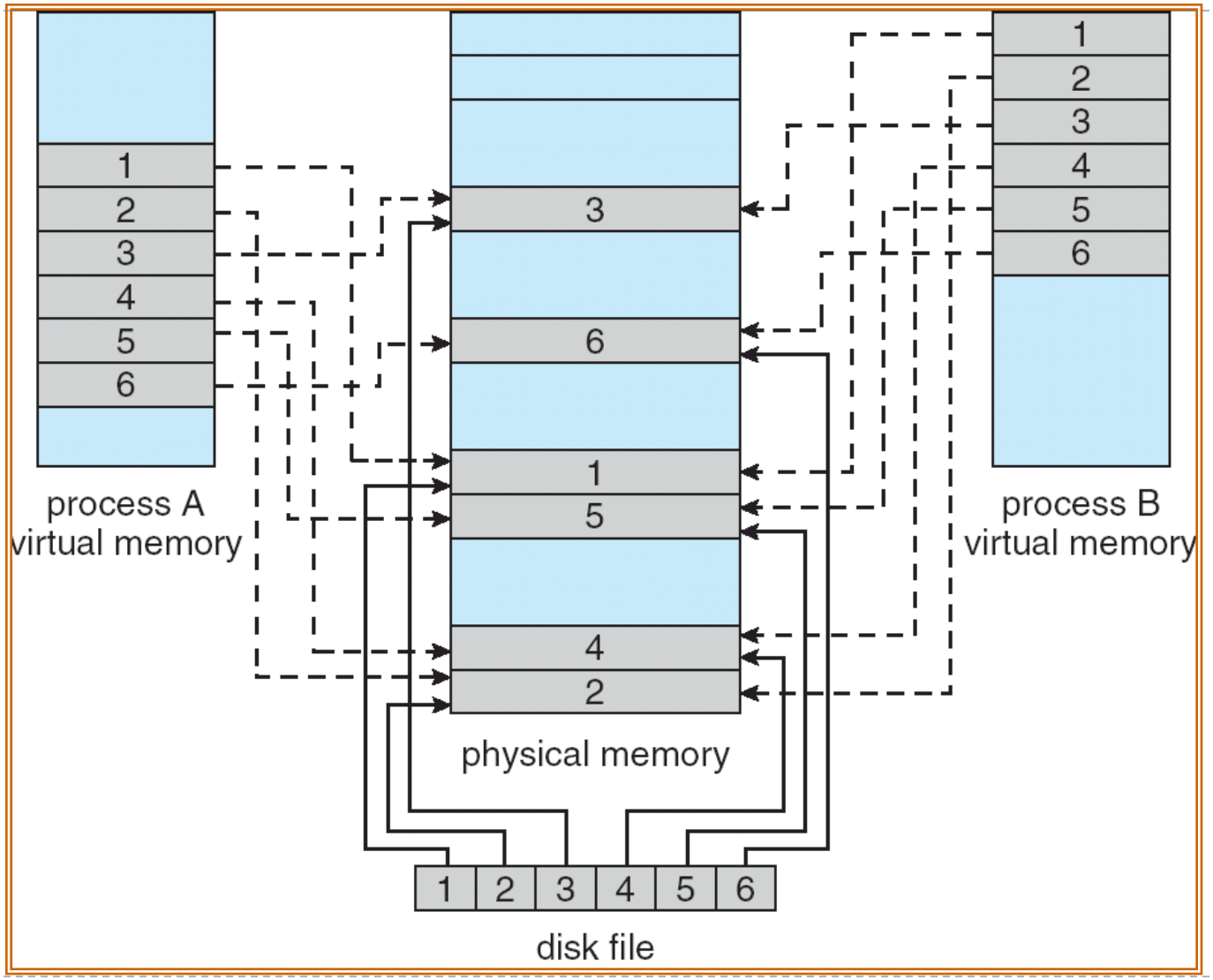
- Working set model
  - Se basa en que un proceso se mueve entre localidades
  - Una localidad es un conjunto de páginas que están activas a la vez.
  - Definir el número de referencias a tomar en cuenta
    - Bajo número
      - No definirá la localidad completa
    - Alto número
      - Tendrá todas las páginas referenciadas
  - Si la localidad aumenta, y no hay frames disponibles
    - Suspender otro proceso
    - Hacer swap out a todas sus páginas
  - Al cambiar de localidad: varios page faults
  - Implementación
    - Varios bits de referencia
    - Interrupción de intervalos de tiempo

- Page Fault Frequency
  - Controlar la frecuencia de page faults
    - Page Fault bajo: el proceso tiene muchas frames
    - Page Fault alto: el proceso tiene pocas frames
  - Establecer límites (inferior y superior) de #frames
  - Tambien puede ser necesario suspender procesos



MEMORY MAPPED FILES

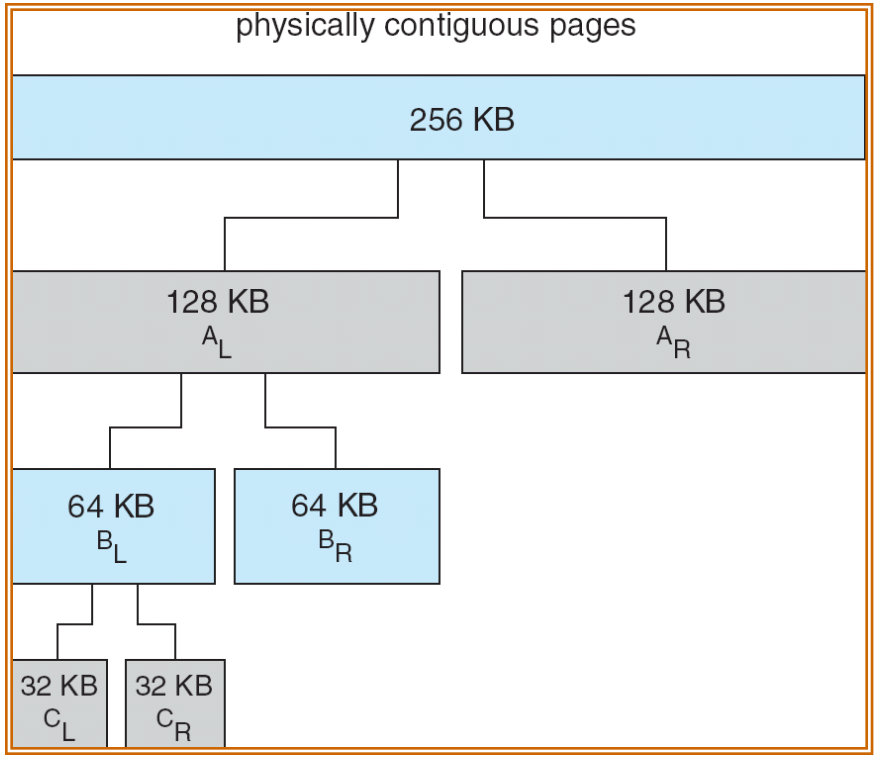
- Open(), read(), write() normales en disco
- Aprovechar mecanismo de memoria virtual
- Mapear bloques de disco a páginas
- Demand paging
- Reads y Writes en memoria
  - No necesariamente inmediatos
- Aplica tambien a I/O
  - Despliegue de pantalla
  - Comunicación con puertos serial/paralelo


1
2
3
4
5
6
process A
virtual memory
3
6
1
5
4
2
physical memory
1
2
3
4
5
6
process B
virtual memory
1
2
3
4
5
6
disk file

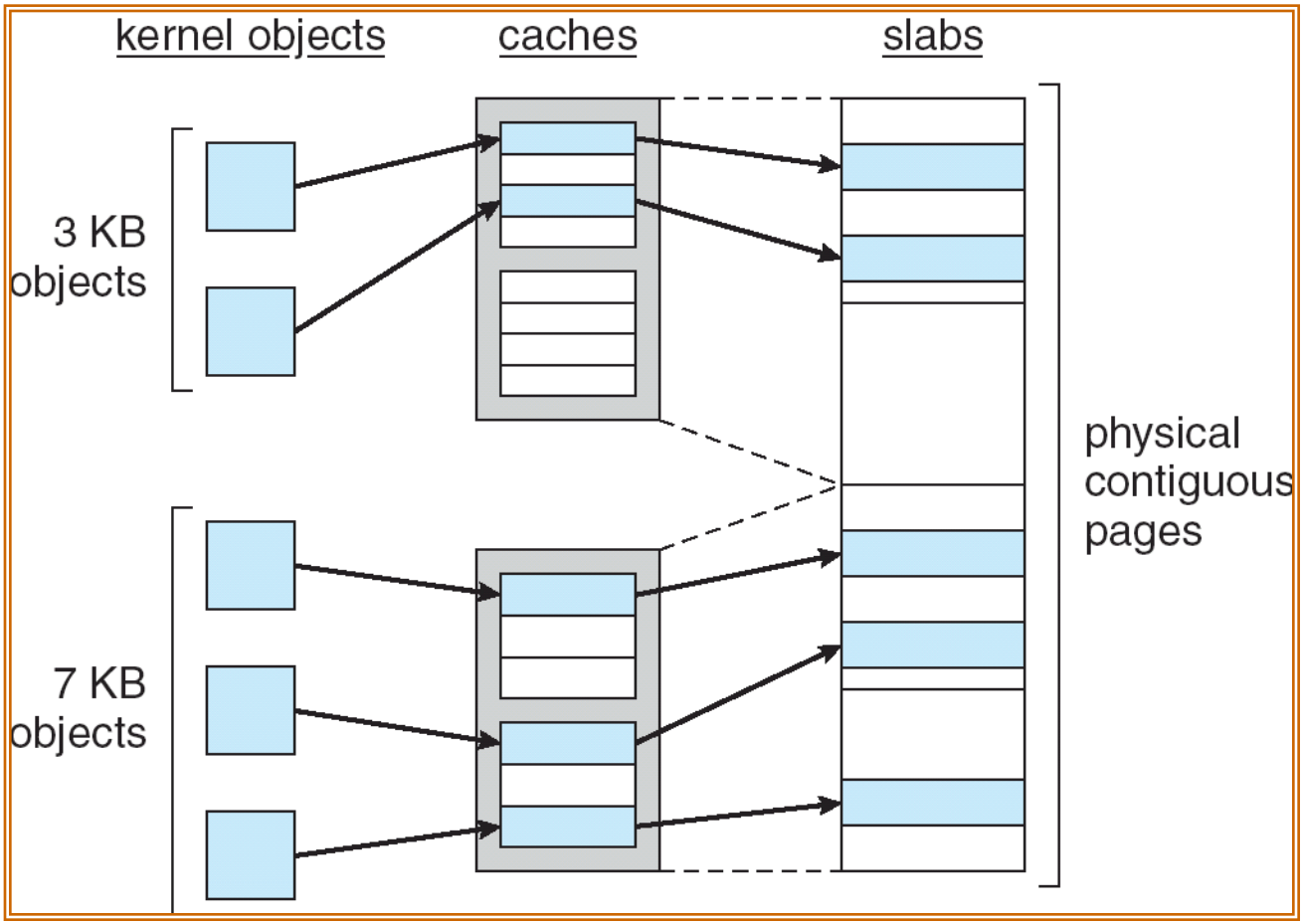
KERNEL MEMORY

- Manejado separado a memoria de procesos
  - Datos de kernel no son sujetos a paginación
  - Objetos creados y destruidos frecuentemente
  - Asignaciones pequeñas (estructuras de datos)
    - Fragmentación interna
  - Dispositivos I/O interactúan directamente con memoria
    - Requieren que memoria sea continua

- Buddy system
  - Dividir memoria en potencias de 2
  - Fácil combinar buddies
  - Fragmentación interna



- Slabs
  - Slab: páginas continuas
  - Cache: uno para cada tipo de estructura de datos
    - Físicamente compuesto por varios slabs
    - Lógicamente compuesto por varios objetos del mismo tipo
      - Semáforos, PCB, Archivos, etc…
  - Creación rápida
  - Para destruir objeto, solamente se marca como libre
  - No hay fragmentación
  - Utilizado en Solaris, Linux, BSD


kernel objects
caches
slabs
3 KB objects
7 KB objects
physical contiguous pages

TAMAÑO DE PÁGINAS

- Tamaño de la tabla de paginación
  - Mayor tamaño de página: menor tabla
- Tiempo para swap in/out
  - Latency y seek time: 28 ms (fijo)
  - Tiempo de transferencia: 2Mb/sec (512bytes/0.2ms)
  - Tamaño de página grande
- Fragmentación interna por proceso: 50% de tamaño de página.

- Localidades más precisas con páginas pequeñas
  - Aislar únicamente memoria utilizada
- Frecuencia de page faults
  - Menor en páginas grandes
- Tomar en cuenta tamaño de sectores en disco
- Históricamente tamaño de páginas va creciendo

DESARROLLADORES / COMPILADORES

- Program 1

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

- Program 2

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

- Peor de los casos
  - Program 1: 128x128 = 16,384 page faults
  - Program 2: 128 page faults

- Estructuras de datos
  - Localidades más precisas
    - Pilas,
  - Localidades poco precisas
    - Hash tables. Los datos son dispersos
  - Velocidad de búsqueda, número de referencias
- Compiladores
  - Separar código y datos para tener código reentrante (no se modifica)
  - Mantener una función completamente en una página
  - Funciones que se llaman unas a otras se pueden agrupar
- Objetos
  - Punteros: memoria más dispersa
  - Localidades menos precisas

WINDOWS

- Demand paging.
- Clustering: traer páginas adyacentes
- Working set minimum / maximum
- Tamaño de páginas: 4KB
- Se baja del número de frames disponibles
  - Remover páginas de procesos. (Sin llegar al mínimo)