Virtual Memory(2)

Dept. of Computer Science

Hanyang University

Allocation of Frames

- How to allocate page frames to multiple processes?

- Each process needs minimum number of pages
  - HW 측면: IBM 370 – 6 pages to handle SS MOVE instruction:
    - instruction is 6 bytes, might span 2 pages
    - 2 pages to handle **from**
    - 2 pages to handle **to**
  - SW 측면:
    - Loop 내의 page 는 한꺼번에 allocate 되는 것이 유리함
    - 그렇지 않으면 매 loop 마다 page fault – CPU/disk load 심한 불균형

- Two major allocation schemes
  - fixed allocation
  - priority allocation

Fixed Allocation

- Equal allocation
  - Allocate the same number of frames to each process
  - e.g., (100 frames, 5 processes) 20 pages each
- Proportional allocation
  - Allocate according to the size of process

– $s_i =$ size of process p_i

– $S = \sum s_i$

– $m =$ total number of frames

– $a_i =$ allocation for $p_i = \frac{s_i}{S} \times m$

$$m = 64$$

$$s_i = 10$$

$$s_2 = 127$$

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

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

Priority Allocation

- Use a proportional allocation scheme using priorities rather than size
- High priority process
  - Give more memory so that it can finish early
  - (decreased I/O ➔ decreased time in ‘waiting’ state ➔ early finish)
- If process P_i generates a page fault,
  - select a victim from one of its frames
  - select a victim from lower priority processes’ frames

Global vs. Local Replacement

- Global replacement – process selects a replacement frame from the set of all frames; one process can take a frame from another.
- Local replacement – each process selects from only its own set of allocated frames.

Thrashing

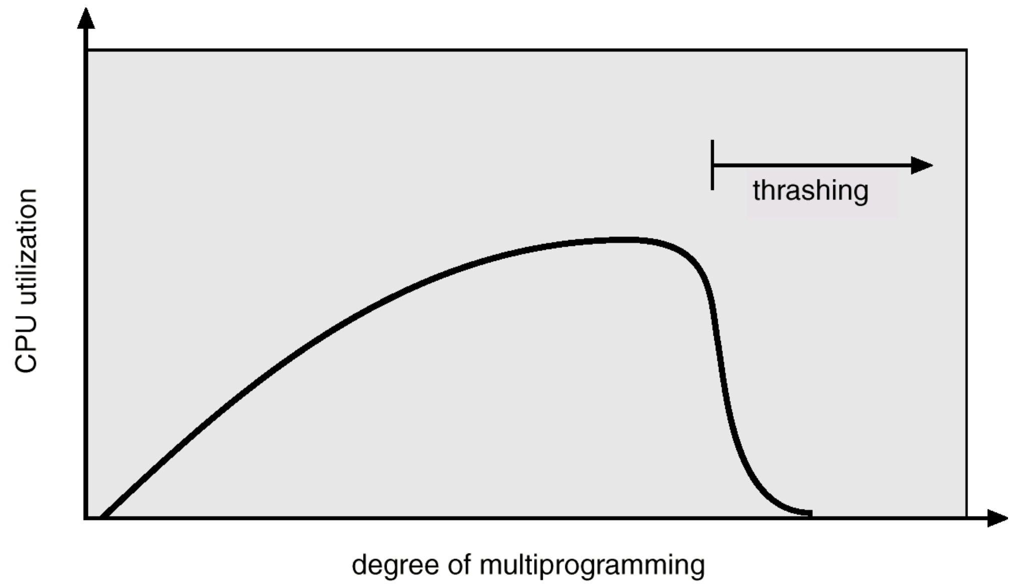
- If a process does not have “enough” pages, the page-fault rate is very high. This leads to:
  - Low CPU utilization
  - Operating system thinks that it needs to increase the degree of multiprogramming
  - Another process added to the system (higher MPD)
- Thrashing
  - A process is busy swapping pages in and out
  - CPU is idle most of the time – low throughput

main()

{ for (I=1, 100000) { A = B + X }}

A
main()
X
B

Thrashing Diagram



- Why does paging work?
  Locality model
  - Process migrates from one locality to another
  - Localities may overlap
- Why does thrashing occur?
  (Σ size of locality) > (total allocated memory size)

Locality

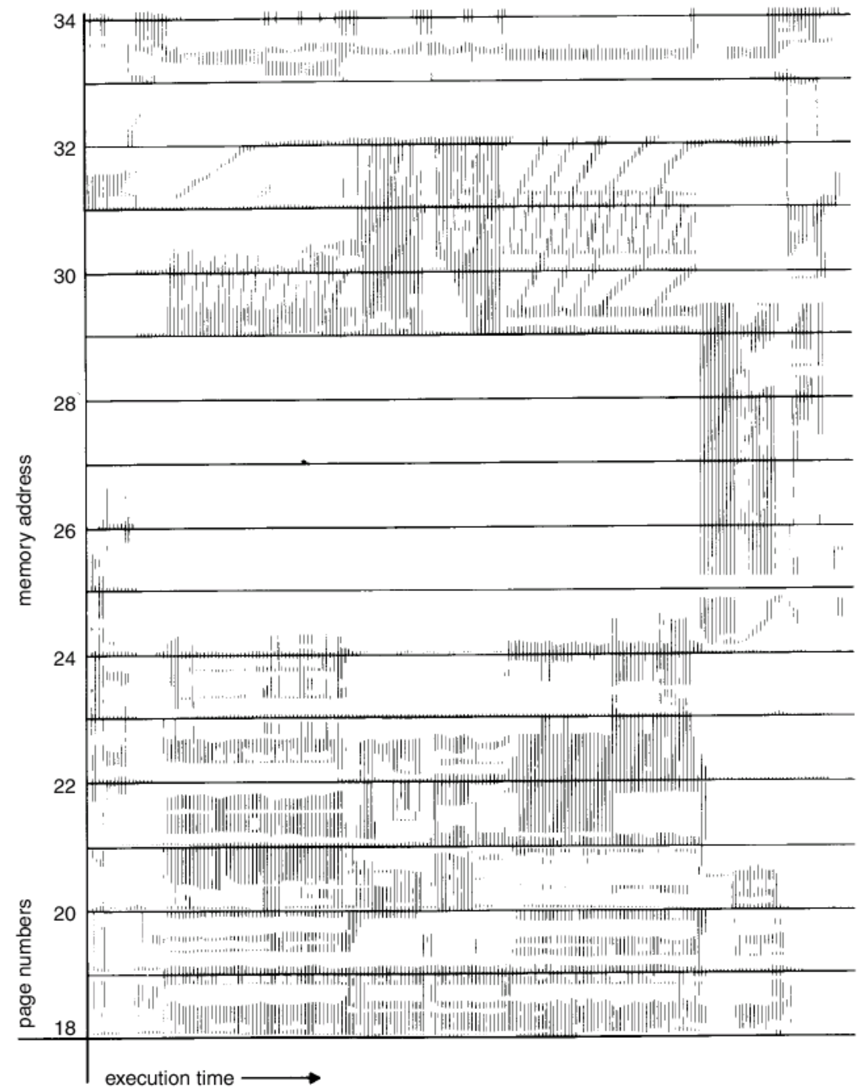
- 프로그램의 메모리 참조는 고도의 지역성을 가짐

- 임의 시간 Δt 내에 프로그램의 일부분만을 집중적으로 참조

  - 시간 지역성 (Temporal Locality) : 현재 참조된 메모리가 가까운 미래에도 참조될 가능성이 높음
    ex) loop, subroutine, stack

  - 공간 지역성 (Spatial Locality) : 하나의 메모리가 참조되면 주변의 메모리가 계속 참조될 가능성이 높음
    ex) Array Traversal, 명령의 순차 실행

Locality In a Memory-Reference Pattern

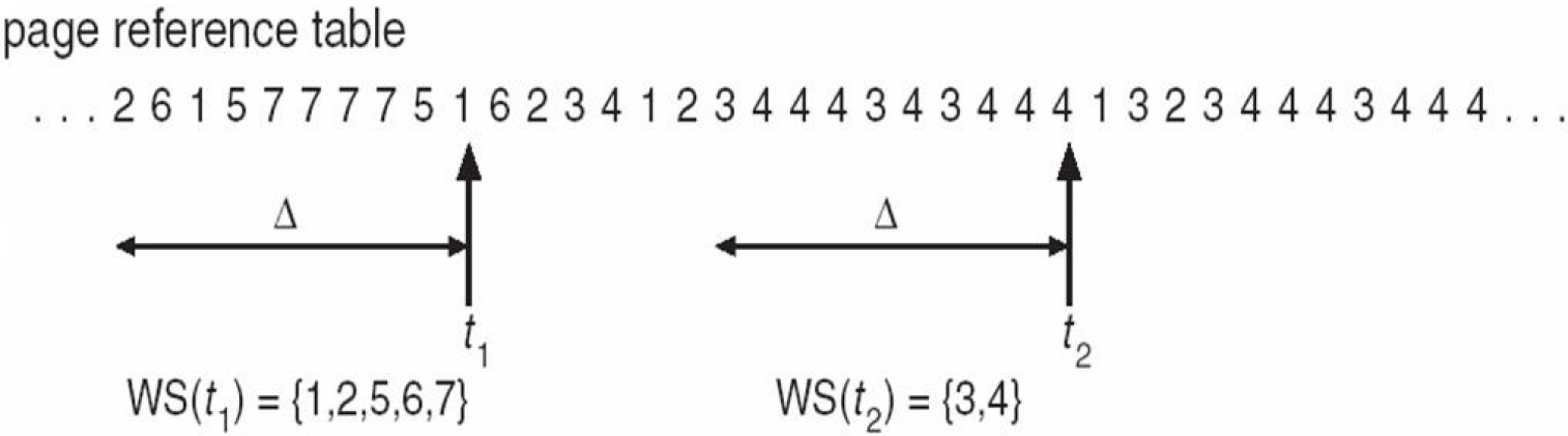


Working-Set Model

- $\Delta \equiv$ working-set window $\equiv$ a fixed number of page references (e.g., 10,000 instructions)
- $WS(t_i) = \{$ pages referenced in $[\, t_i, \;\; t_i - \Delta \,] \}$
  - if Δ too small, will not encompass entire locality
  - if Δ too large, will encompass several localities
  - if $\Delta = \infty \Rightarrow$ will encompass entire program

    123123123248024802480248033666666633666666336666633666
- WSS_i = Working set size for process P_i
- $D = \Sigma\, WSS_i \equiv$ total demand frames
- If $D > m \Rightarrow$ *Thrashing* (m is total number of available frames)
- Policy: if D > m, then suspend one of the processes
- Working set model
  - If a process is not allocated memory greater than its WSS, choose a process to be suspended ➔ Determines MPD
  - 즉 (WSS 전체) or (suspended) 중 택일

Working-Set model



Working-Set Model

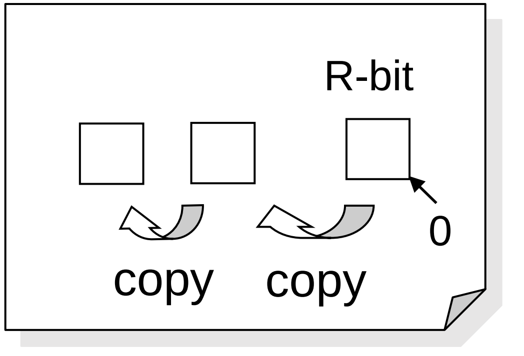
- $WS(t_i)$ = { pages referenced in [$t_{i,}$, $t_i - \Delta$] }
- 만일 페이지 P 가 t_i에 $WS(t_i)$에 속하였으면 keep in memory
  안 속하였으면 out of memory
- 이 원칙에 따라 replace, allocate 를 결정
- 따라서 working set model은 allocate/replace 를 같이 결정함
- 시간에 따라 allocation size 가 달라질 수 있음
- $WS(t_i)$ 가 모두 보장되어야만 run, 아니면 suspend

Keeping Track of the Working Set

- 구현: 매 ref 마다 각 page 들의 최근 reference time 을 Δ 와 비교?

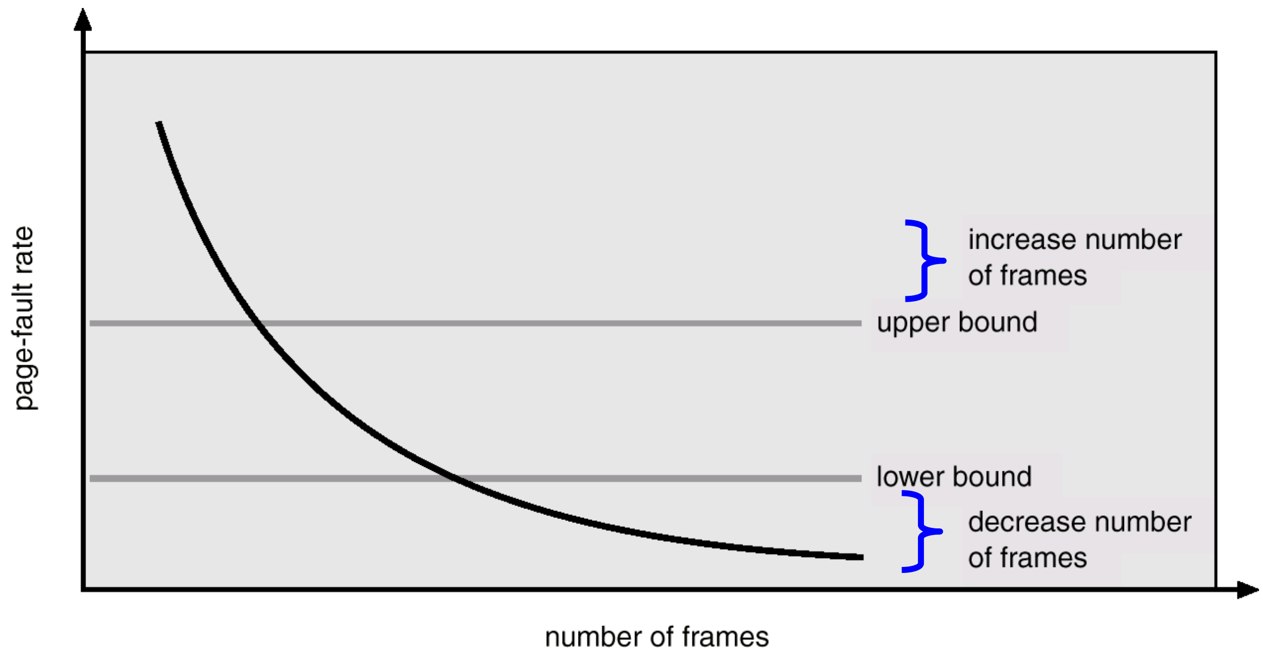
  → too expensive (space for ref-time field + time for comparison)

- Approximate with interval timer + a reference bit
- Example: Δ = 10K,
  - Timer interrupts: every 5K time units.
  - Keep in memory 2 bits for each page.
  - timer interrupts → copy and resets all reference bits
  - If one of the bits in memory = 1 ⇒ page belongs to the working set
- Why is this not completely accurate?
- Improvement = 10 bits and interrupt every 1000 time units



- Q: How do you decide window size?

Page-Fault Frequency Scheme



- Establish “acceptable” page-fault rate
  - If actual rate too low, process loses frame
  - If actual rate too high, process gains frame
- Working set 기법은 page 참조 시마다 페이지 집합을 수정
- PFF 기법은 page fault 발생 시에만 페이지 집합을 수정

Other Benefits of VM: Process Creation

- Virtual memory allows other benefits during process creation:

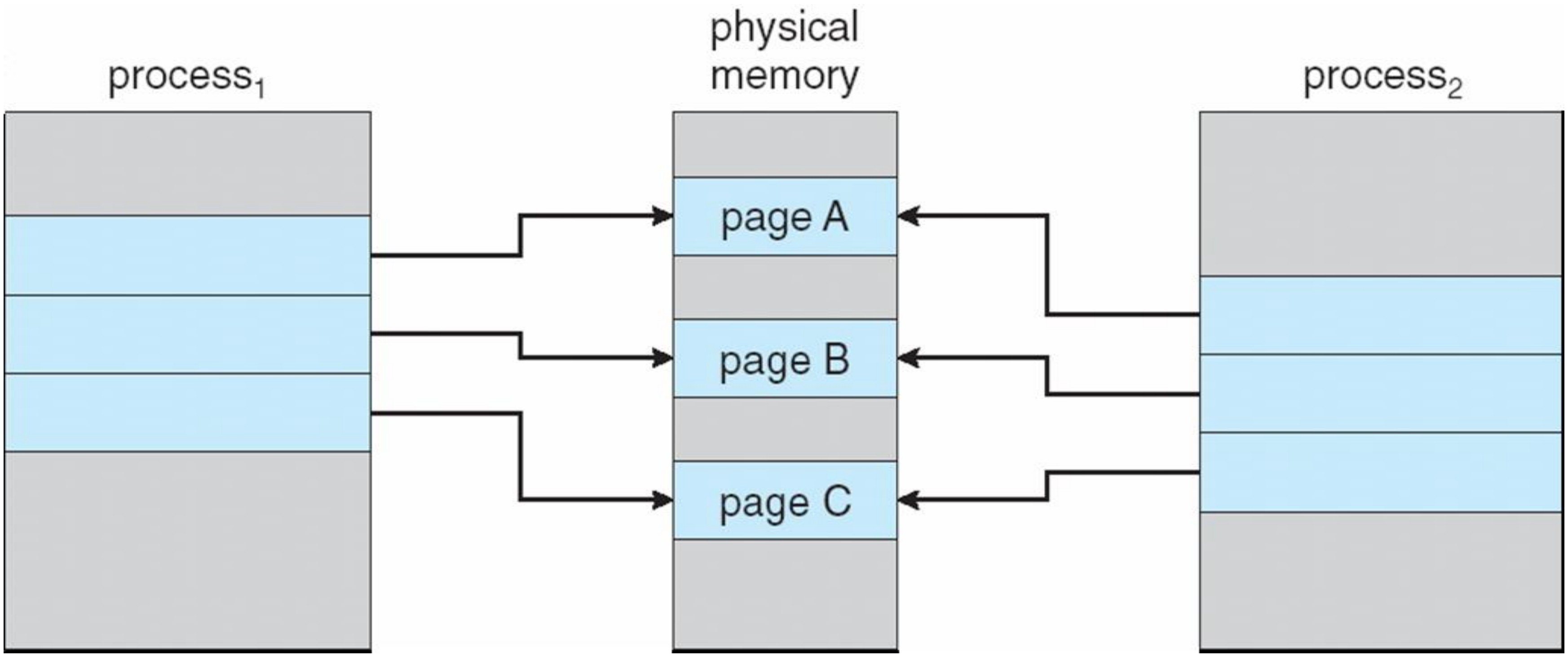
  - Copy-on-Write

  - Memory-Mapped Files

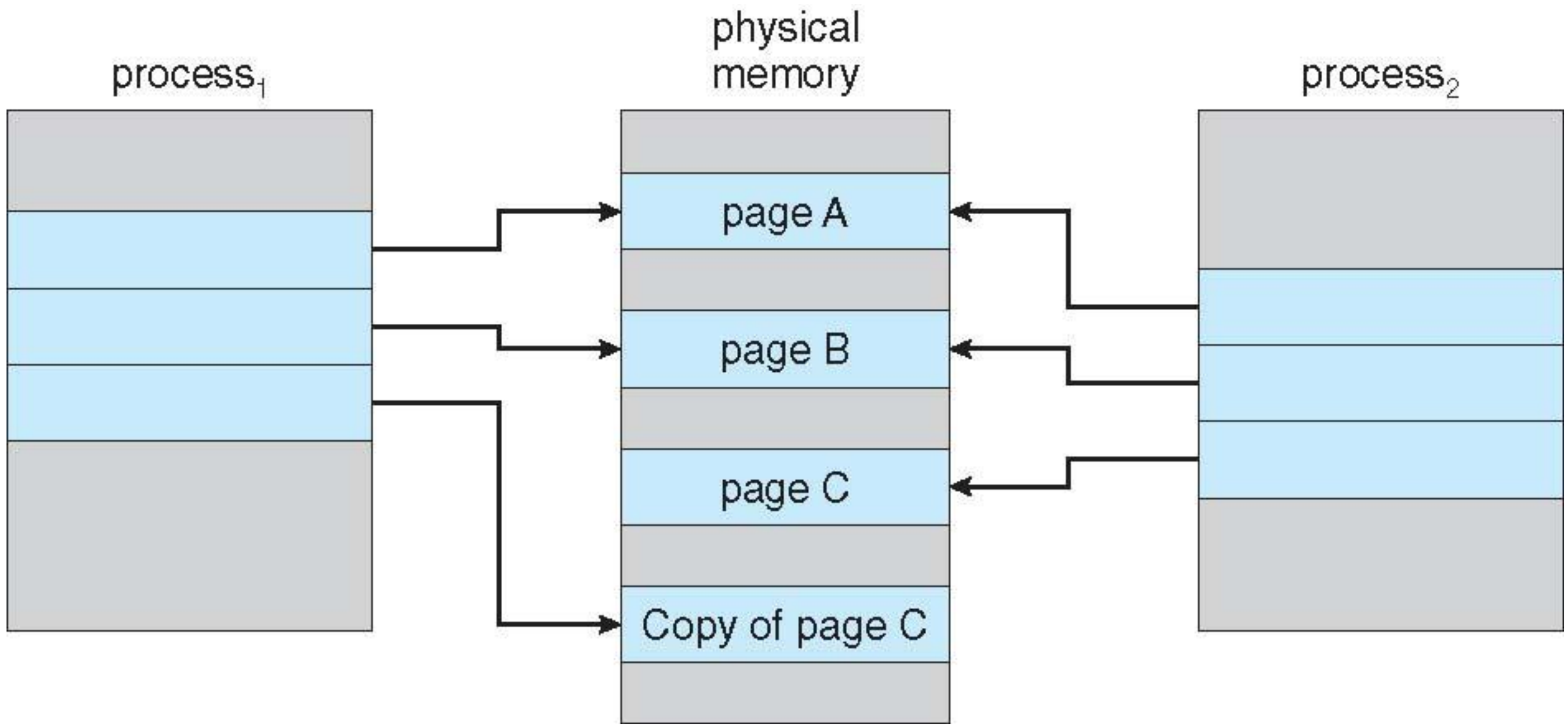
Copy-on-Write

- Copy-on-Write (COW)
  - Allows both parent and child processes to initially *share* the same pages in memory
  - If either process modifies a shared page, only then is the page copied
- COW allows more efficient process creation as only modified pages are copied

Before Process 1 Modifies Page C



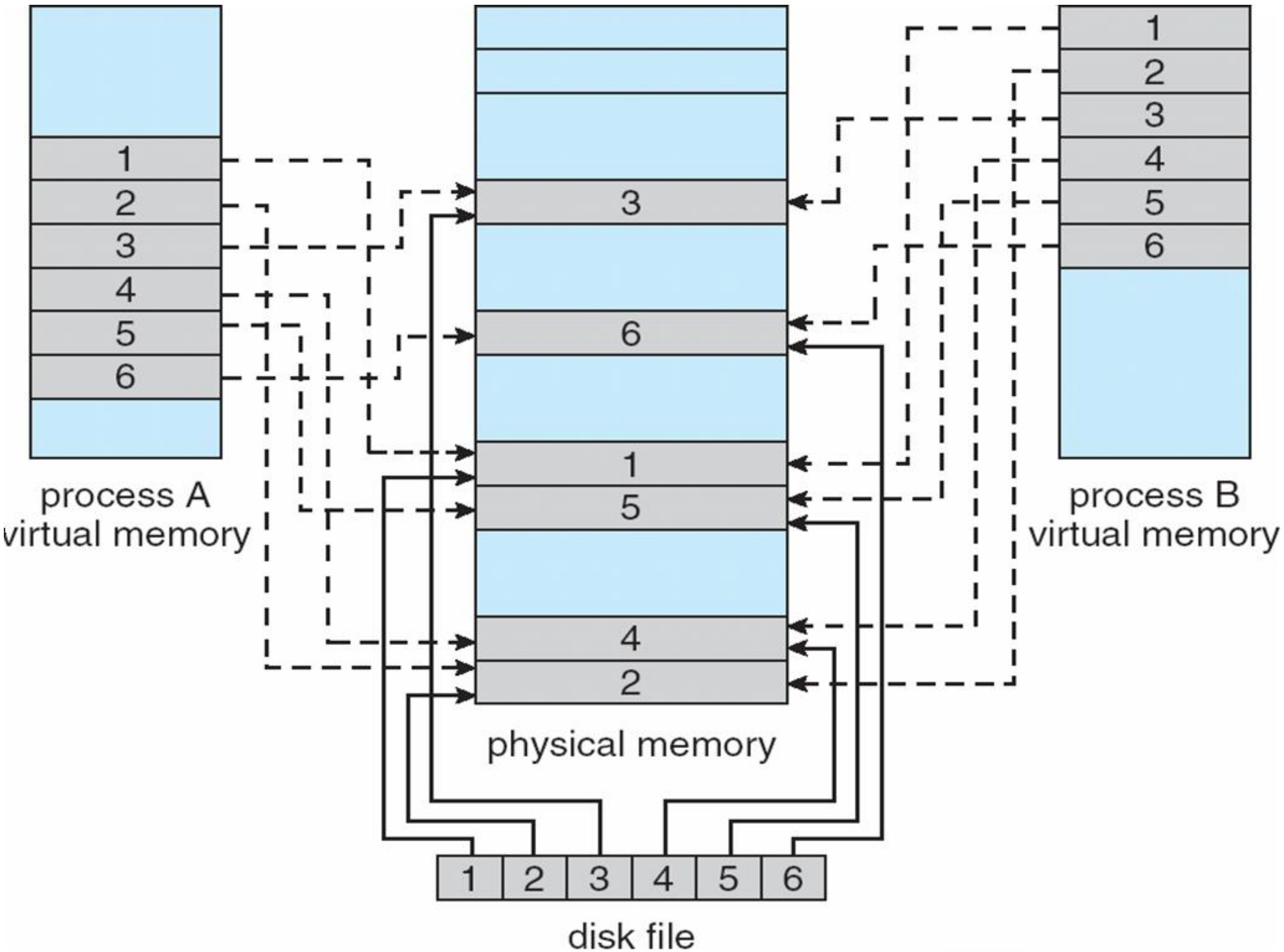
After Process 1 Modifies Page C



Memory-Mapped Files

- Memory-mapped file I/O
  - Allows file I/O to be treated as routine memory access
  - by **mapping** a disk block to a page in memory
- A file is initially read using demand paging
  - A page-sized portion of the file is read from the file system into a physical page
  - Subsequent reads/writes to/from the file are treated as ordinary memory accesses
- Simplifies file accesses
  - Treats file I/O through memory rather than **read(), write()** system calls
- Also allows several processes to map the same file allowing the pages in memory to be shared

Memory Mapped Files



Other Issues – Prepaging

- Prepaging
  - To reduce the large number of page faults that occurs at process startup
  - Prepage all or some of the pages a process will need, before they are referenced
  - But if prepaged pages are unused, I/O and memory was wasted
  - Assume s pages are prepaged and α of the pages is used
    - Is cost of $s * \alpha$ saved pages faults > or < than the cost of prepaging $s * (1-\alpha)$ unnecessary pages?
    - α near zero $\Rightarrow$ prepaging loses

Other Issues – Page size

- Considerations for page size selection
  - Internal fragmentation
  - Page table size
  - Disk transfer efficiency – seek/rotation vs. transfer
  - Frequency of I/O operations
  - Improved Locality
    - Smaller page size isolate only needed info within page
  - Trend
    - Larger page size
    - CPU speed, memory capacity – improves faster than disk speed
    - Page fault (relative) penalty is becoming more costly these days

Other Issues – TLB Reach

- TLB Reach
  - The amount of memory accessible from the TLB
  - TLB Reach = (TLB Size) X (Page Size)
  - Ideally, the working set of each process is stored in the TLB
    - Otherwise there is a high degree of TLB misses
  - Increase the Page Size
    - This may lead to an increase in fragmentation as not all applications require a large page size
  - Provide Multiple Page Sizes
    - This allows applications that require larger page sizes the opportunity to use them without an increase in fragmentation

Other Issues – Program Structure

- Program structure
  - `int data[128][128];`
  - Each row is stored in one page
  - Assume that the # of free frames for data < 128
  - Program 1

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

128 x 128 = 16,384 page faults

  - Program 2

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

128 page faults (even when there is only one page frame for data)

Other Issues – I/O interlock

- **I/O Interlock** – Pages must sometimes be locked into memory

- Consider I/O - Pages that are used for copying a file from a device must be locked from being selected for eviction by a page replacement algorithm