Operating Systems

Nguyen Tri Thanh
ntthanh@vnu.edu.vn

Question

What is incorrect about overlays?

A. overlays allows a large program to run in a smaller MEM
B. Overlays only loads codes on demand (when they are used)
C. Programmers need to split the program into modules
D. Overlays is supported in all high level programming languages

Question

What is incorrect about swapping?

A. swapping is the same as overlays
B. swapping uses hard disk as the *backing store*
C. swapping allows many processes whose size is even larger than MEM to run
D. a lower priority process is rolled out for a higher priority one to run (when needed)

Review

Which is incorrect about non-contiguous MEM allocation?

A. split logical memory into parts
B. utilize MEM more effectively in comparison with contiguous allocation method
C. need a Memory Management Unit
D. only suitable for some types of processes

Review

Which is correct about MMU of paging and segmentation allocation methods?

A. they are the same
B. MMU of paging needs more information than that of segmentation
C. they use different resolution methods
D. MMU of segmentation is faster than that of paging

Question

Suppose a process in contiguous allocation:

- the base address is 10400
- the limit register is 1200
- the reference is 246;

Which of the following is the correct physical address of the reference ?

A. 10154

B. 10646

C. 1446

D. 954

Question

A system uses paging

- the frame size of 2KB;
- the address register is 32 bits

Which of the following is correct about register segmentation?

A. (page:offset) = (19:13)
B. (page:offset) = (21:11)
C. (page:offset) = (22:10)
D. (page:offset) = (20:12)

Question

Frame
56
120
3

A system uses paging

- the frame size of 4KB;
- the address register is 32 bits

- Which of the following is the correct physical address of the reference (2,1296)?
  A. 560*4096+1296
  B. 120*4096+1296
  C. 3*4096+1296
  D. 120*1024+1296

Virtual Memory

Paging on demand

Page replacement

Frame allocation

Thrashing

Objectives

- Introduce paging method
- Introduce segmentation method

Reference

- Chapter 9 of Operating System Concepts

Virtual memory

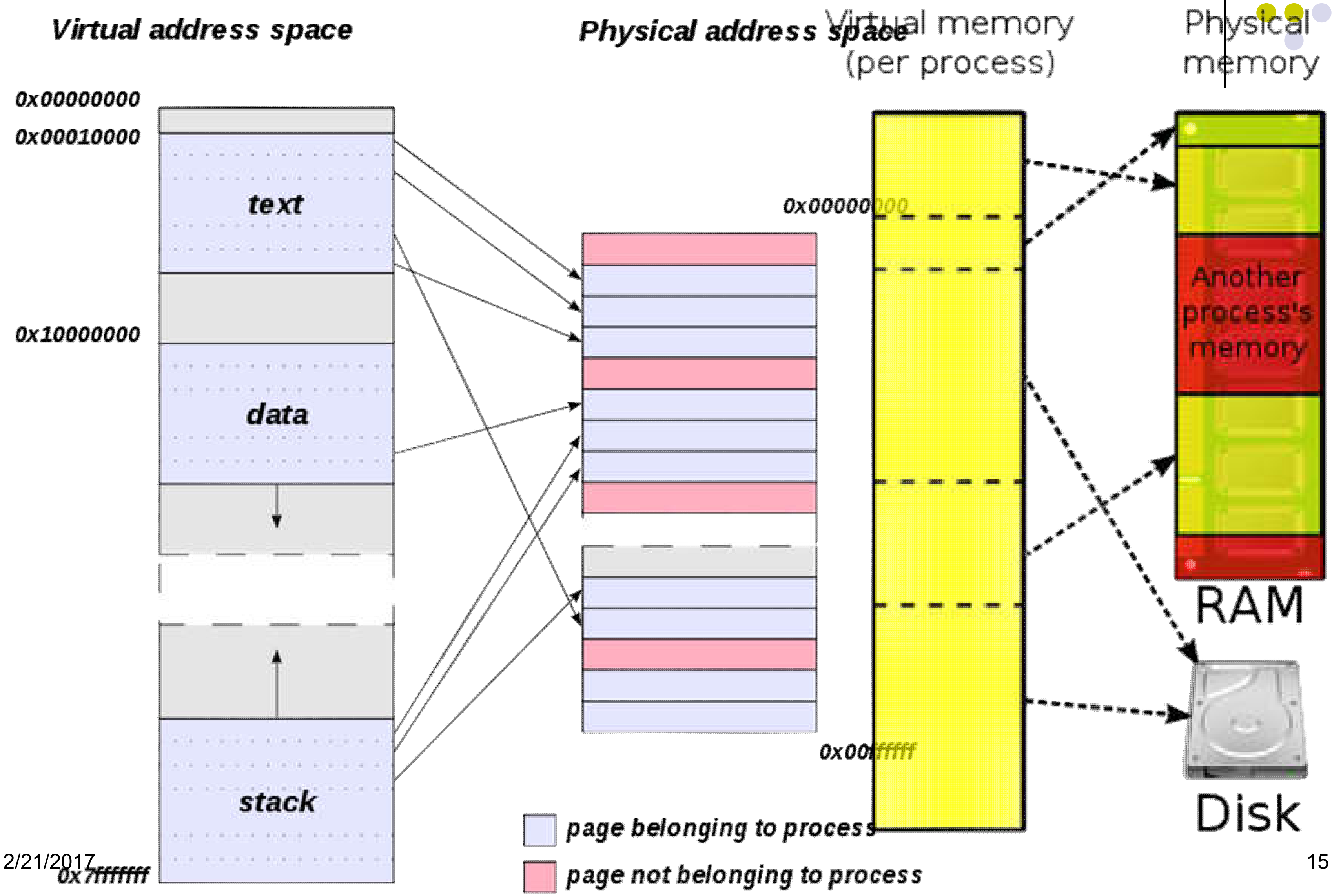
Virtual memory

- **S**eparation of user logical memory from physical memory.
  - Only a part of the program needs to be in memory for execution
  - Logical address space can therefore be much larger than physical address space
  - Allows address spaces to be shared by several processes
  - Allows for more efficient process creation
- Virtual memory can be implemented via
  - Paging on demand
  - Segmentation on demand

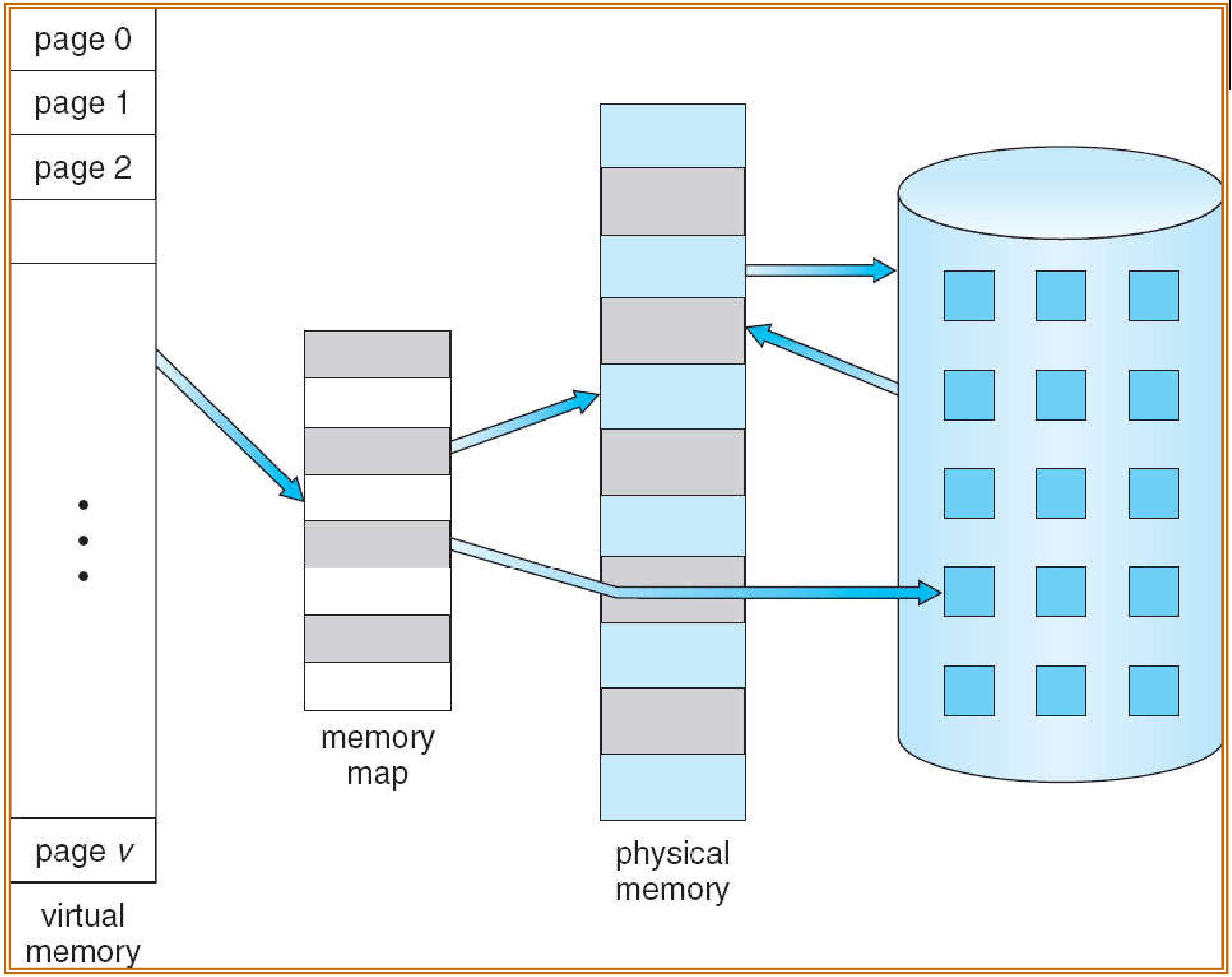
Virtual memory

- Linux is, of course, a virtual memory system, meaning that the addresses seen by user programs do not directly correspond to the physical addresses used by the hardware. Virtual memory introduces a layer of indirection that allows a number of nice things. With virtual memory, programs running on the system can allocate far more memory than is physically available; indeed, even a single process can have a virtual address space larger than the system's physical memory. Virtual memory also allows the program to play a number of tricks with the process's address space, including mapping the program's memory to device memory.
- http://www.makelinux.net/ldd3/chp-15-sect-1

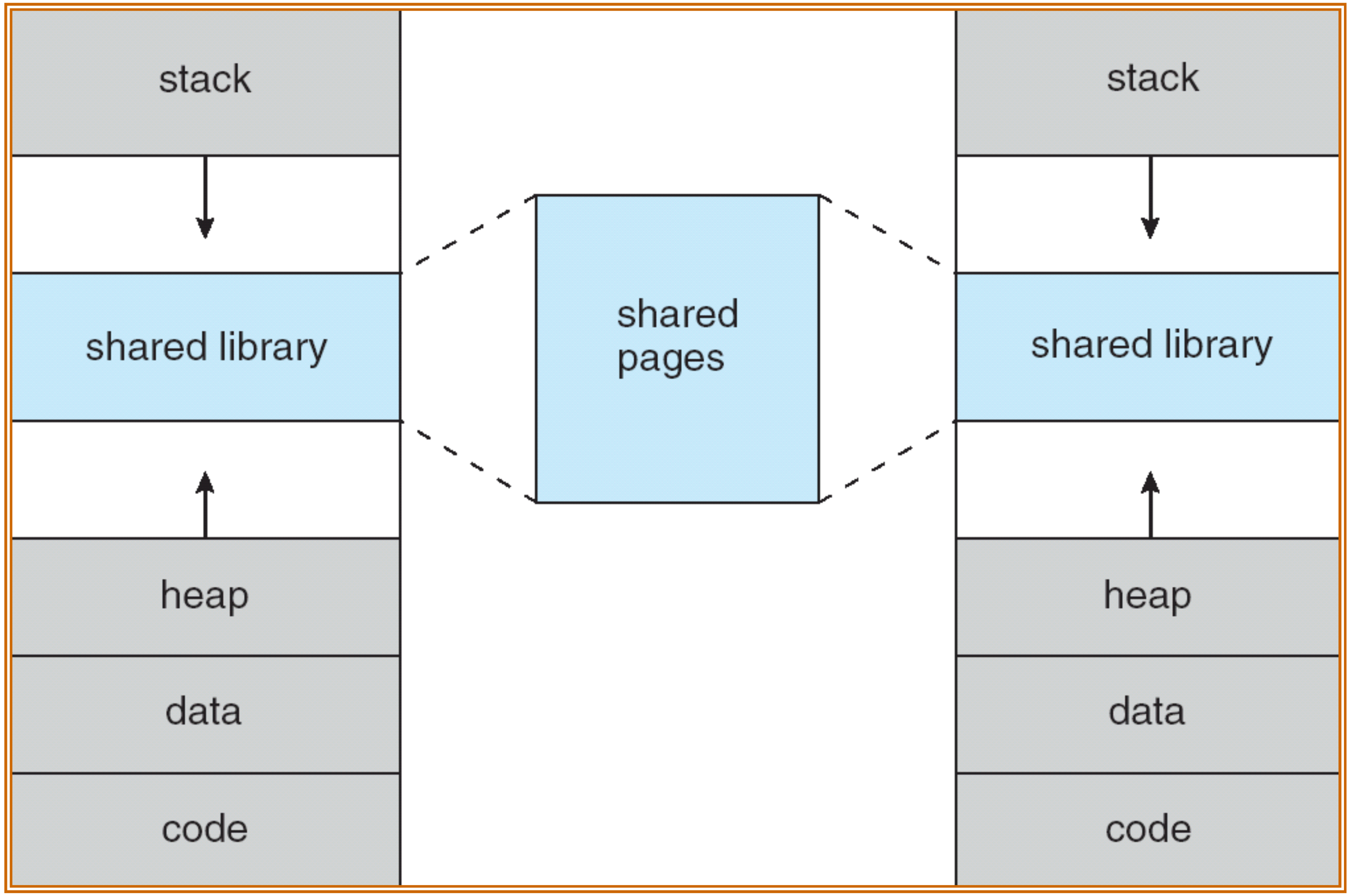
Virtual-address Space



Virtual Memory That is Larger Than Physical Memory



Shared Library Using Virtual Memory



Process Creation

- Virtual memory allows other benefits
  - Copy-on-Write during process creation
  - Memory-Mapped Files

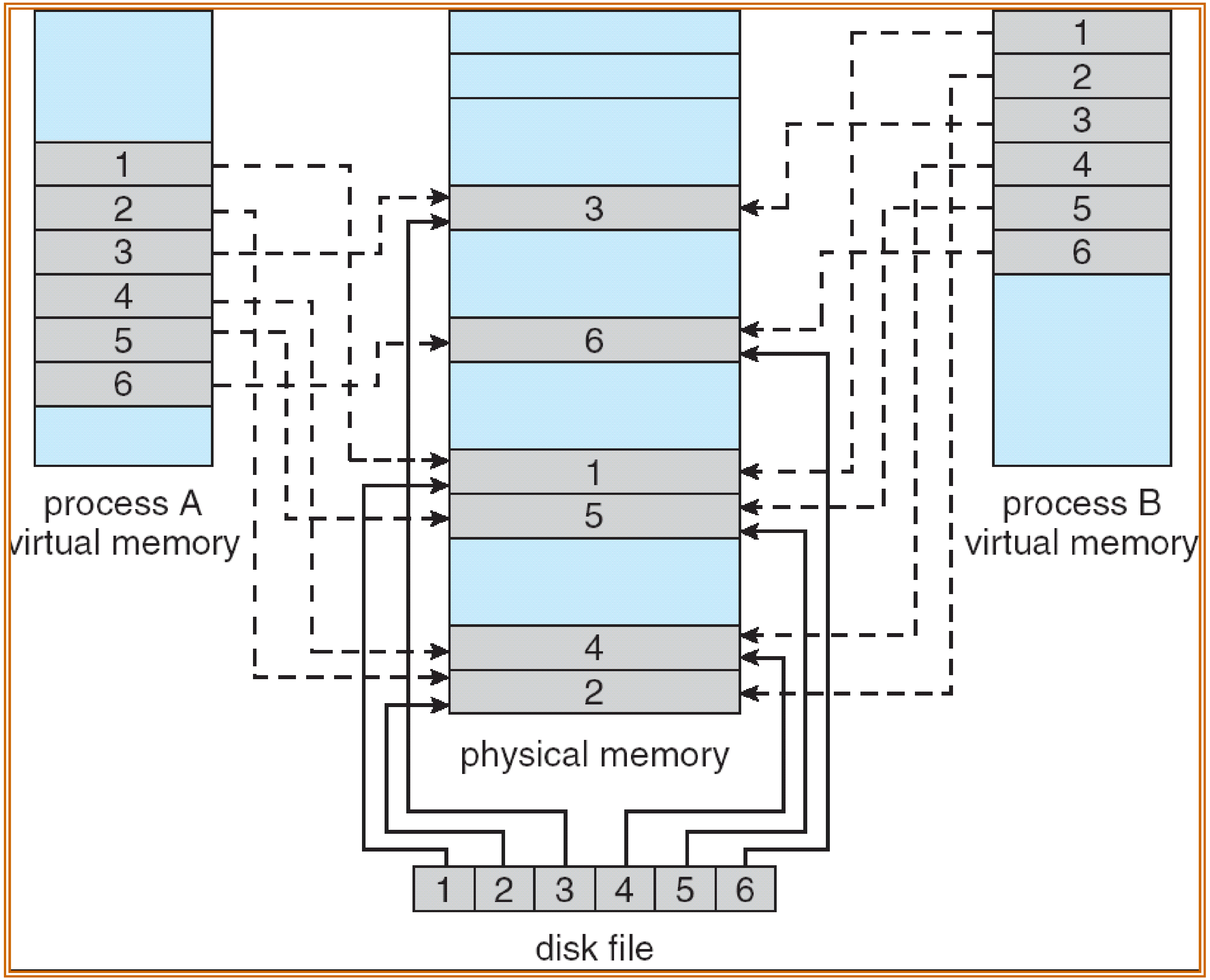
Copy-on-Write

- 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
  - refer to Sect. 2, Chapter 3 of “Lập trình C/C++ …”
- COW allows more efficient process creation as only modified pages are copied

Memory Mapped Files

- A file is considered as a memory segment
- Read/write operations are performed via memory
  - not read/write file system calls
- Allow multiple processes shared a file
- refer to Sect. 5, Chapter 4 of “Lập trình /C++ trên Linux”

Memory Mapped Files



Question

- What is the correct advantage of memory mapped file?
  A. reduces the task of the system's OS
  B. treats as the buffer for manipulating the file
  C. allows programmers to organize the file
  D. uses as shared resource among processes

Question

- Which of the following is incorrect about virtual memory?
  A. it is separated from physical memory
  B. it is mapped into physical memory during process execution
  C. it gives additional benefits, e.g., COW, file mapping
  D. an address in virtual memory is preserved when mapped into physical memory

Dynamic loading

- Routine is not loaded until it is called
- Better memory-space utilization
  - unused routine is never loaded
- Useful when
  - large amounts of code are needed to handle infrequently occurring cases
- No special support from the operating system
  - refer to Section 3.4, Chapter 7 of “Lập trình C/C++ …”

Dynamic linking and shared library

- Linking postponed until execution time
- Small piece of code, *stub*, used to locate the appropriate memory-resident library routine
  - Stub replaces itself with the address of the routine, and executes the routine
  - Operating system needed to check if routine is in processes’ memory address
- Dynamic linking is particularly useful for libraries
- Also known as **shared libraries** in Linux
  - refer to Sect. 3, Chapter 7 of “Lập trình C/C++ trên Linux”

Paging on demand

Food order


One more ice cream
One more steak
In a minute
art_links@yahoo.com.vn

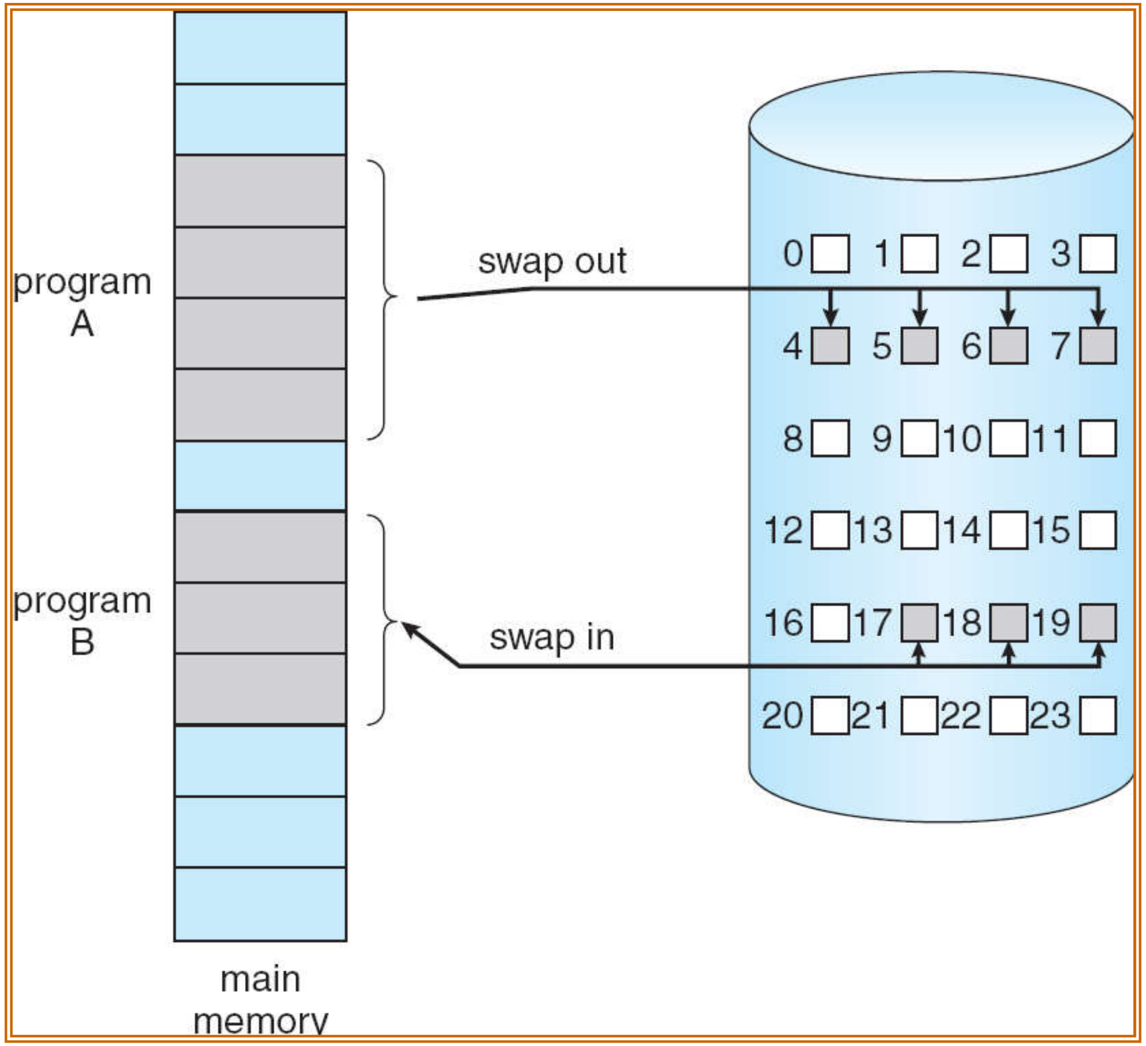
Paging on demand

- Bring a page into memory only when needed
  - Less I/O needed
  - Less memory needed
  - Faster response
  - More users
- Page is needed $\Rightarrow$ reference to it
  - invalid reference $\Rightarrow$ abort
  - not-in-memory $\Rightarrow$ bring to memory
- **Lazy swapper** – never swaps a page into memory unless page is needed
  - Swapper that deals with pages is a **pager**

Question

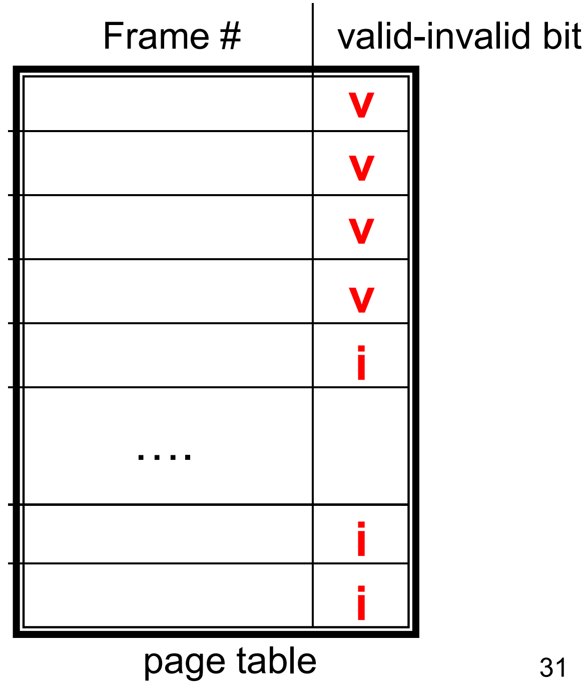
- Why it is possible when only a part of a process is loaded into MEM?
  A. Because instructions of a process are independent
  B. Because we can indicate which instructions to run
  C. Because only one instruction is executed at a time
  D. Because related instructions are always in the same group

Transfer of a Paged Memory to Contiguous Disk Space



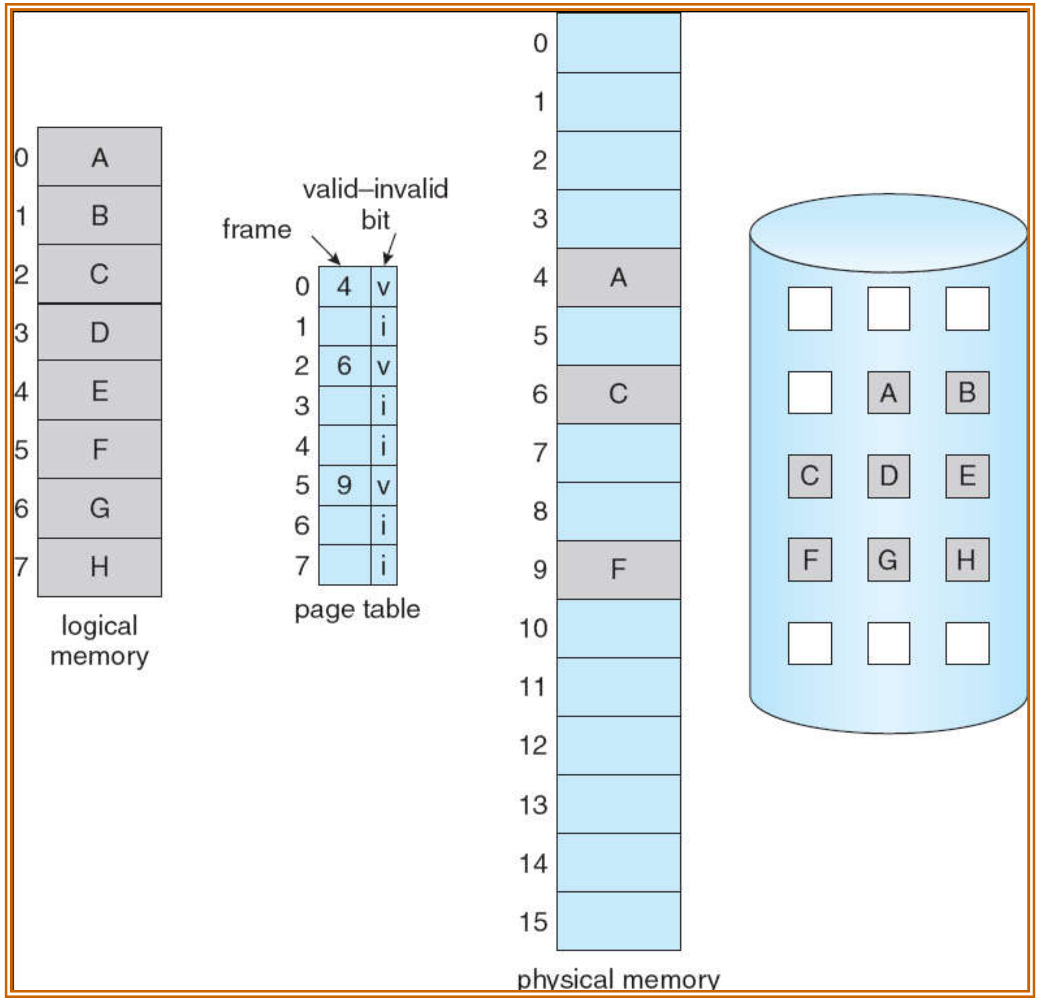
Valid-Invalid Bit

- With each page table entry a valid–invalid bit is associated
  - (**v** ⇒ in-memory, **i** ⇒ not-in-memory)
- Initially valid–invalid bit is set to **i** on all entries
- Example of a page table snapshot:



- During address translation, if valid–invalid bit in page table entry is **I** ⇒ page fault

Page Table When Some Pages Are Not in Main Memory

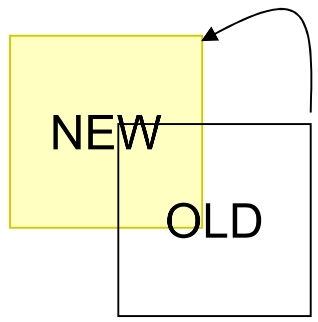


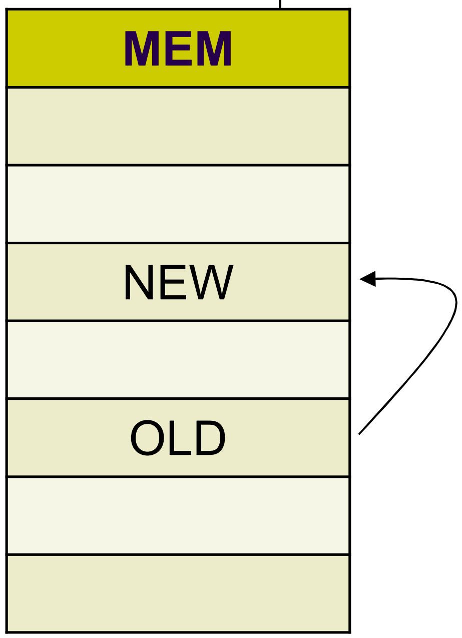
Page Fault

- If there is a reference to a page,
  - first reference to that page will trap to operating system: *page fault*
  1. Operating system looks at another table to decide:
     - Invalid reference ⇒ abort
     - Just not in memory
  2. Get empty frame
  3. Swap page into frame
  4. Update the page table
  5. Set validation bit = **v**
  6. Restart the instruction that caused the page fault

Page Fault (Cont.)

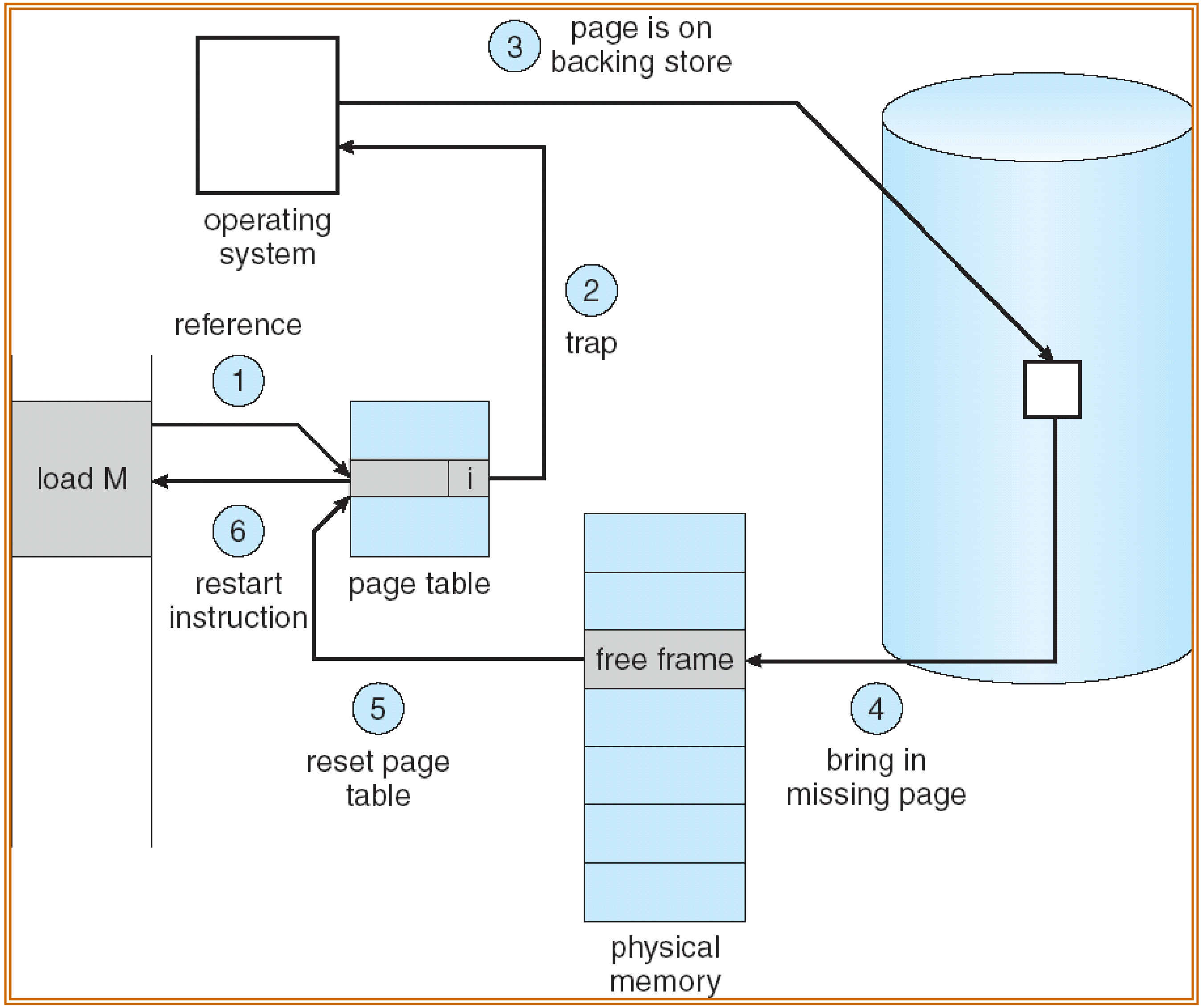
- Restart instruction
  - block move



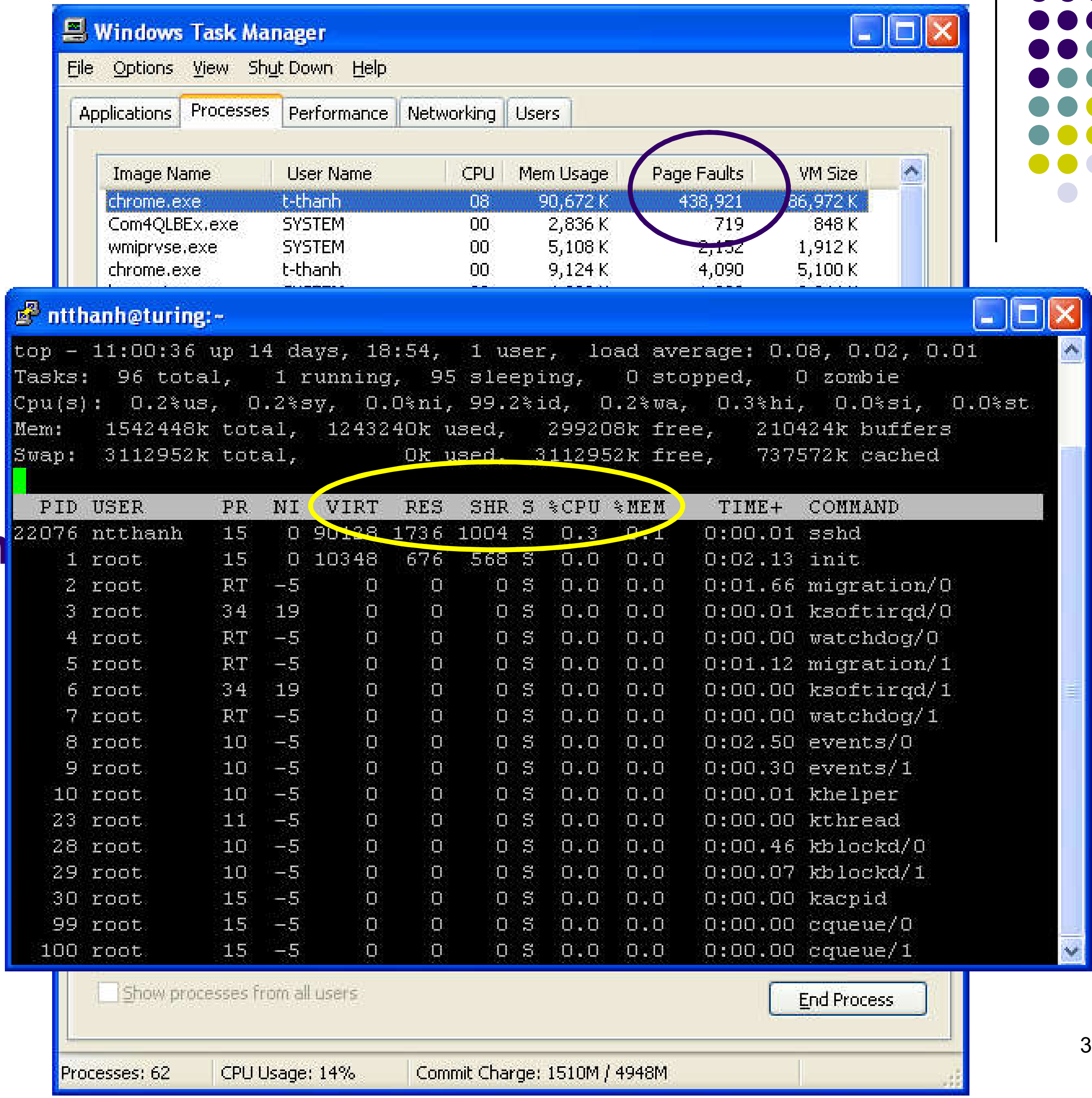


  - auto increment/decrement location

Steps in Handling a Page Fault



Process information



Question

- Which of the following is incorrect about a page fault?
  A. it happens in paging on demand
  B. it happens when a reference to a page that is not in MEM
  C. when a page fault occurs the corresponding process will be terminated
  D. a page fault handler is called whenever it occurs

Question

- Which of the following is incorrect order of steps in handling a page fault?
  A. check if the valid bit is invalid ⇒ raise a page fault
  B. a page fault is raised ⇒ find the page in backing store
  C. a page fault is raised ⇒ find a free frame
  D. load the page into memory ⇒ restart the instruction

If no free frame available

- Call page replacement procedure
  - swap out an unused page from MEM
- Algorithms
  - FIFO, Optimal, LRU, LRU-approximation
- Performance of the algorithm
  - page-fault rate
  - which algorithm is better?

Performance of paging on demand

- Page Fault Rate $0 \leq p \leq 1.0$
  - if $p = 0$ no page faults
  - if $p = 1$, every reference is a page fault
- Effective Access Time (EAT)

EAT = $(1 – p)$ * memory access
+ p (page fault overhead
+ swap page out
+ swap page in
+ restart overhead)

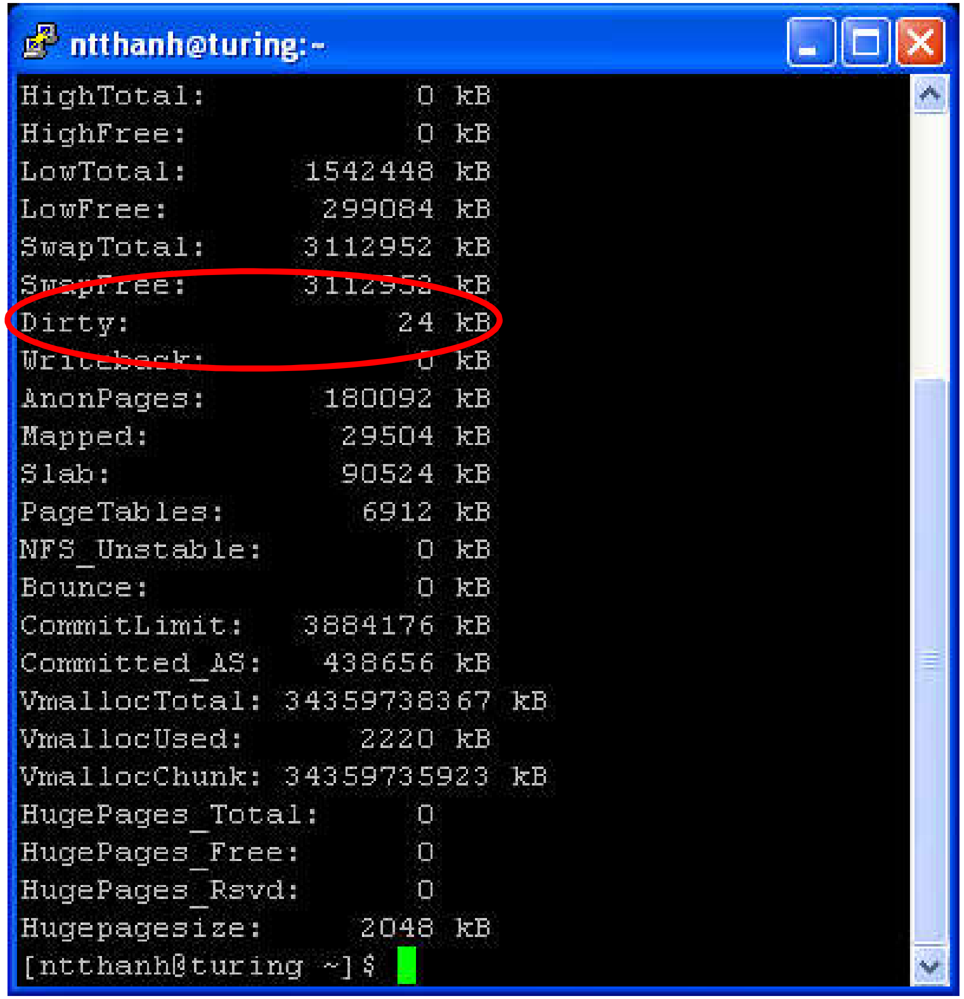
Paging on Demand Example

- Memory access time = 200 nanoseconds
- Average page-fault service time = 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
- If one access out of 1,000 causes a page fault
  - EAT = 8.2 microseconds.
  - slowdown by a factor of 40!!

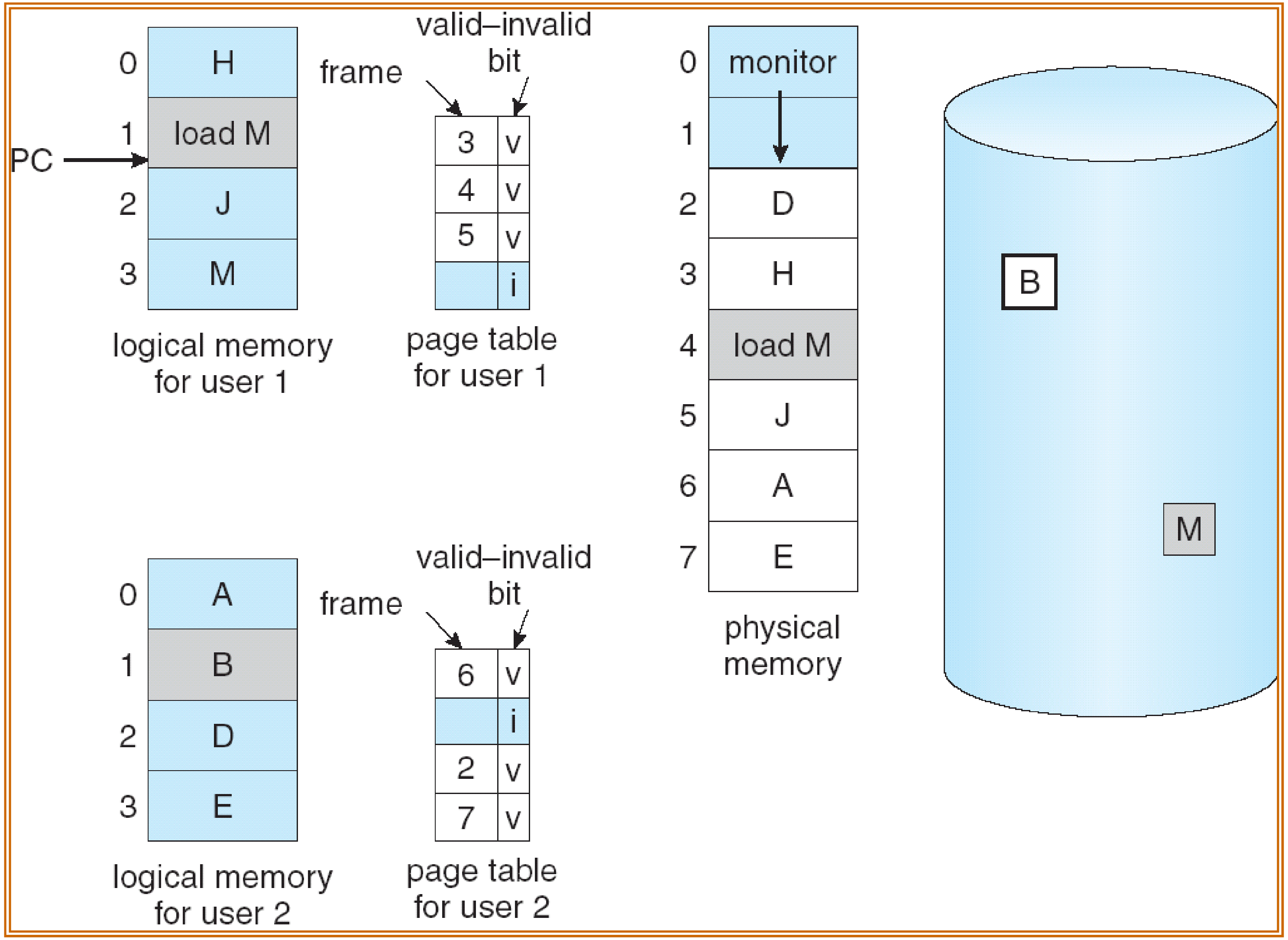
Page Replacement

- Prevent over-allocation of memory
  - include page replacement in page-fault service routine
- Use **modify (dirty) bit** to reduce overhead of page transfers
  - only modified pages are written to disk
- Page replacement completes separation between logical memory and physical memory
  - large virtual memory can be provided on a smaller physical memory

Page Replacement



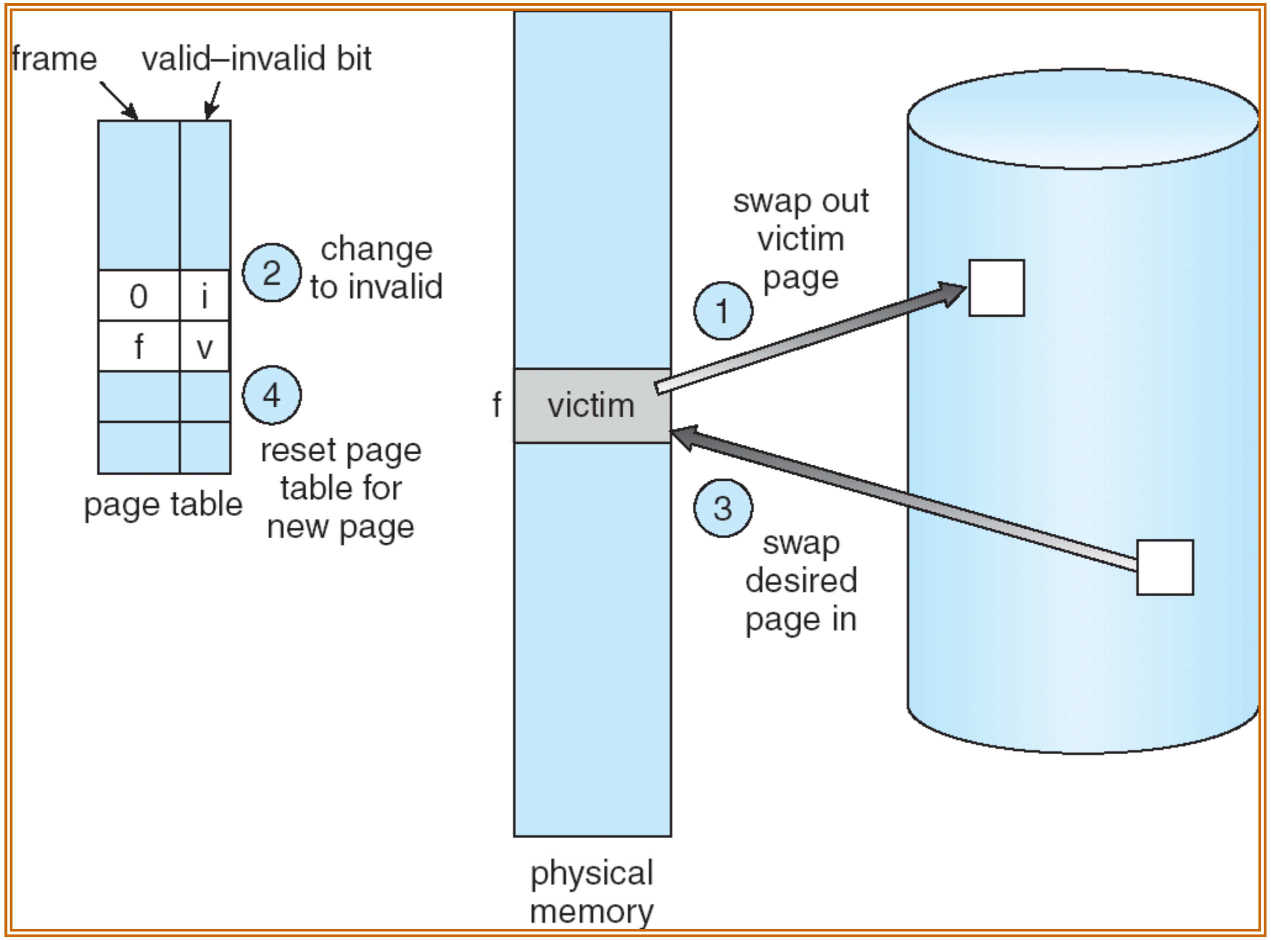
Need For Page Replacement



Basic Page Replacement

1. Find the location of the desired page on disk
2. Find a free frame
   - If there is a free frame, use it
   - Else use a page replacement algorithm to select a **victim** frame
3. Bring the desired page into the (newly) free frame; update the page table
4. Resume the process

Page Replacement



Question

Which of the following is incorrect about page replacement?

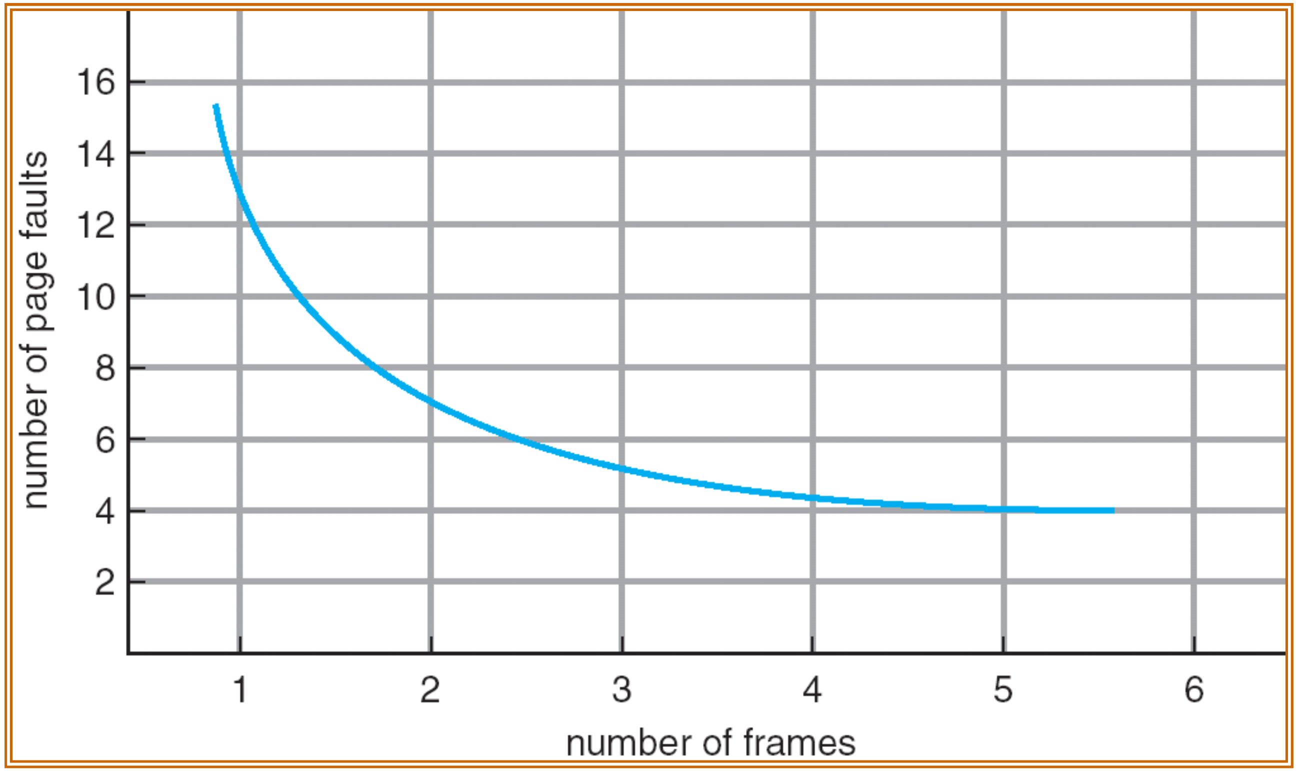
A. a victim frame is selected to be swapped out
B. the page table which is the victim will be updated
C. the victim frame is always written into the backing store
D. the victim frame is only written into the backing store if it is dirty

Page Replacement Algorithms

- Want lowest page-fault rate
- Evaluate algorithm
  - run it on a particular string of memory references (reference string)
  - compute the number of page faults on that string
- In all our examples, the reference string is

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

Graph of Page Faults Versus The Number of Frames



First-In-First-Out (FIFO) Algorithm

- Reference string: 1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5

- 3 frames

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

9 page faults

- 4 frames

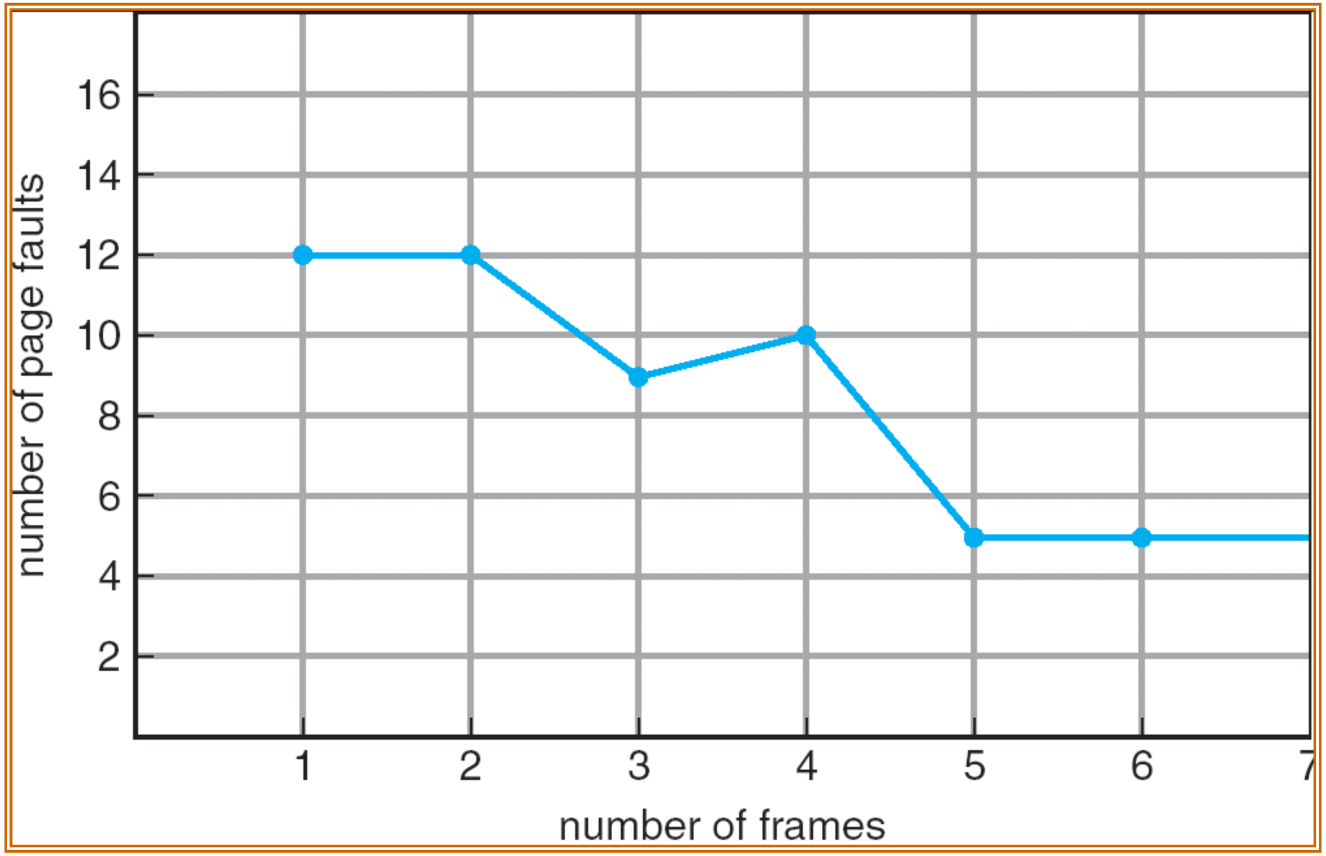
1	1	5	4
2	2	1	5
3	3	2	
4	4	3	

10 page faults

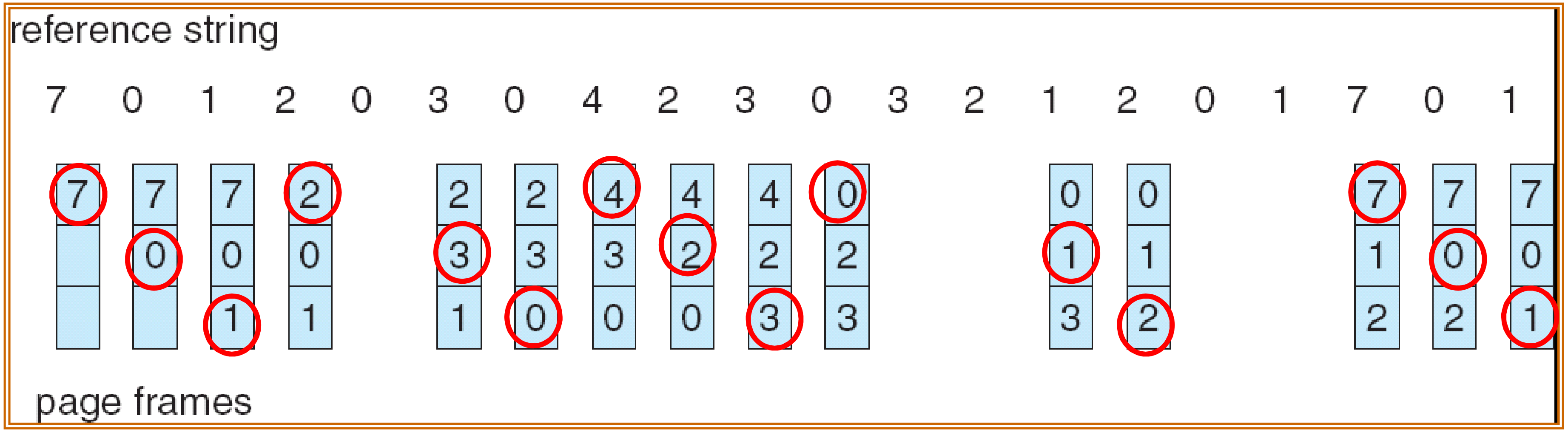
$\Rightarrow$ Belady's anomaly

Belady's Anomaly: more frames $\Rightarrow$ more page faults

FIFO Illustrating Belady's Anomaly



FIFO Page Replacement



Question

- A reference string 7 0 1 2 0 3 0 4 2 3 0 3 2 1 2 0 1 7 0 1, FIFO is used with 3 frames. Which of the following is the correct order of swapped out pages?
  A. 7 0 1 2 3 0 4 2 3 0 1 2
  B. 7 0 1 2 3 0 4 3 2 0 1 2
  C. 7 0 1 3 2 0 4 2 3 0 1 2
  D. 7 0 1 2 3 0 4 2 3 1 0 2

Question

- A reference string 7 0 1 2 0 3 0 4 2 3 0 3 2 1 2 0 1 7 0 1, FIFO is used with 3 frames. Which of the following is the correct number of page faults?
  A. 13
  B. 14
  C. 15
  D. 16

Optimal Algorithm

- Replace page that will not be used for longest period of time
- 4 frames example

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

How do you know this?

- Used for measuring how well the algorithm performs

1	4
2	
3	
4	5

6 page faults

Question

- A reference string 7 0 1 2 0 3 0 4 2 3 0 3 2 1 2 0 1 7 0 1, Optimal algorithm is used with 3 frames. Which of the following is the correct order of swapped out pages?
  A. 7 1 0 3 4 2
  B. 7 0 1 4 3 2
  C. 7 1 0 4 3 2
  D. 7 1 4 1 3 2

Question

- A reference string 7 0 1 2 0 3 0 4 2 3 0 3 2 1 2 0 1 7 0 1, Optimal algorithm is used with 3 frames. Which of the following is the correct number of page faults?
  A. 8
  B. 9
  C. 10
  D. 11

Least Recently Used (LRU) Algorithm

- Least recently used page is swapped out first
  - Reference string: 1, 2, 3, 4, 1, 2, **5**, 1, 2, **3**, **4**, **5**
  - 4 frames

1	1	1	1	**5**
2	2	2	2	2
3	**5**	5	**4**	4
4	4	**3**	3	3

Question

- A reference string 7 0 1 2 0 3 0 4 2 3 0 3 2 1 2 0 1 7 0 1, LRU is used with 3 frames.
- Which of the following is the correct order of swapped out pages?
  A. 7 1 2 3 1 4 1 3 2
  B. 7 2 1 3 0 4 2 3 2
  C. 7 1 2 3 0 4 1 2 3
  D. 7 1 2 3 0 4 0 3 2

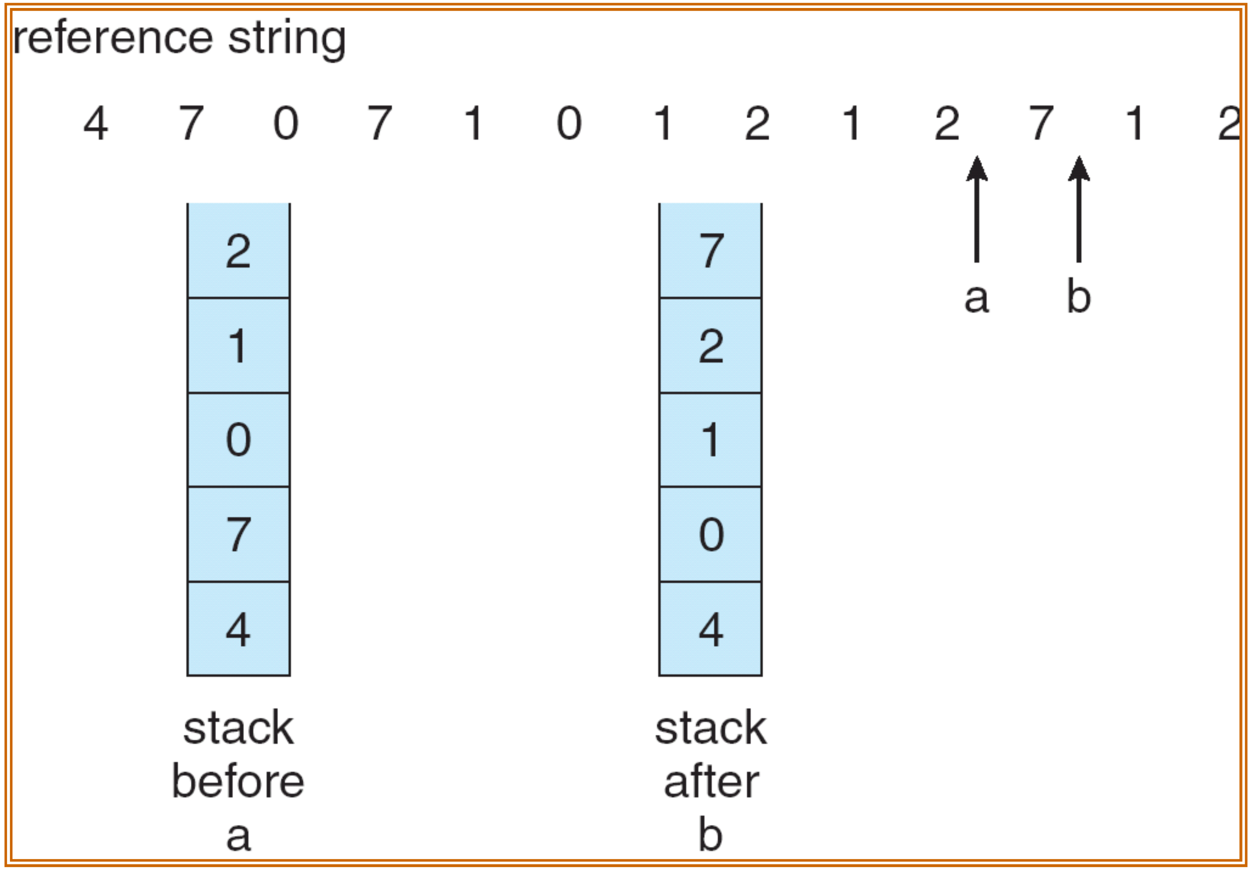
Question

- A reference string 7 0 1 2 0 3 0 4 2 3 0 3 2 1 2 0 1 7 0 1, LRU is used with 3 frames.
- Which of the following is the correct number of page faults?
  A. 13
  B. 12
  C. 11
  D. 10

LRU Algorithm (Cont.)

- Stack implementation
  - keep a stack of page numbers in a double link form
  - Page referenced:
    - move it to the top
    - requires 6 pointers to be changed
  - No search for replacement

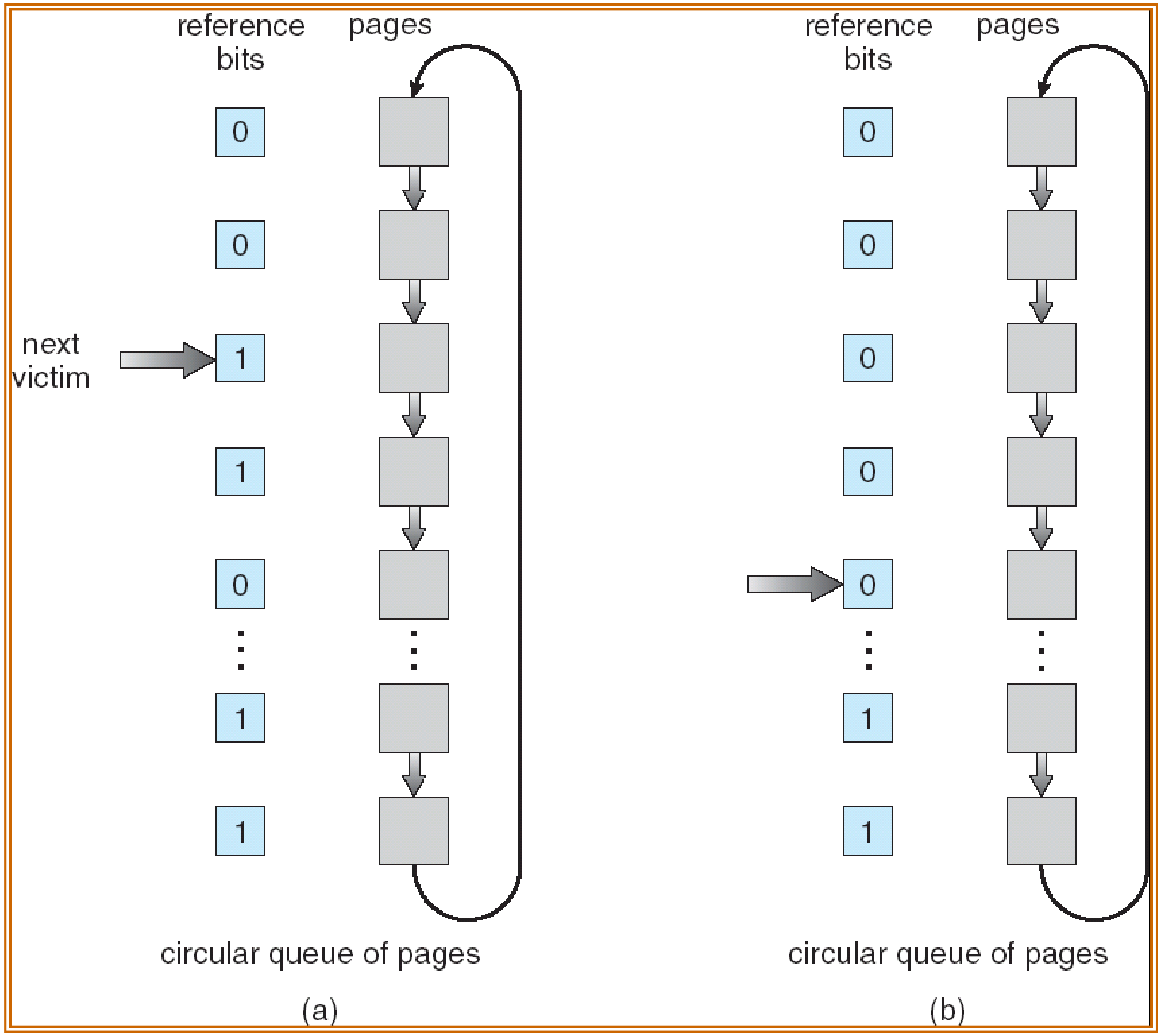
Use Of A Stack to Record The Most Recent Page References



LRU Approximation Algorithms

- Reference bit
  - With each page associate a bit, initially = 0
  - When page is referenced bit set to 1
  - Replace the one which is 0 (if one exists)
    - We do not know the order
- Second chance (follow clock order)
  - Need reference bit
  - Clock replacement
  - If page to be replaced has reference bit = 1 then:
    - set reference bit 0
    - leave page in memory
    - replace next page, subject to same rules

Second-Chance (clock) Page-Replacement Algorithm



Question

- Suppose the second chance is used;
  - the reference bits of frames are: 1 1 0 1 1 0
  - the head is at second frame
- Which of the following are the reference bits after a page replacement is done
  A. 0 0 0 1 1 0
  B. 1 0 1 1 1 0
  C. 1 0 0 1 1 0
  D. 1 0 1 0 1 0

Least frequently Used (LFU) Algorithm

- Counter implementation
  - Every page entry has a counter;
    - every time page is referenced, copy the clock into the counter
  - When a page needs to be swapped
    - look at the counters to determine

Counting Algorithms

- Keep a counter of the number of references that have been made to each page
- **Least Frequently Used (LFU) Algorithm**
  - replaces page with smallest count
- **Most Frequently Used (MFU) Algorithm**
  - based on the argument that the page with the smallest count was probably just brought in and has yet to be used

Allocation of Frames

- Each process needs *minimum* number of pages
- Example: 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*
- Two major allocation schemes
  - fixed allocation
  - priority allocation

Fixed Allocation

- Equal allocation
  - For example, if there are 100 frames and 5 processes, give each process 20 frames.
- Proportional allocation
  - Allocate according to the size of process

$$- s_i = \text{size of process } p_i$$

$$- S = \sum s_i$$

$$- m = \text{total number of frames}$$

$$- a_i = \text{allocation for } a_i = \frac{s_i}{S} \times m$$

$$m = 64$$

$$s_1 = 10$$

$$s_2 = 127$$

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

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

Question

- A system uses proportional allocation and has
  - 90 frames x 2KB
  - 3 processes with size of (138KB, 96KB, 164KB)
- Which of the following is the correct number of allocated frames of (P_1, P_2, P_3)
  A. 32, 21, 37
  B. 31, 22, 37
  C. 30, 22, 38
  D. 33, 22, 35

Priority Allocation

- Use a proportional allocation scheme using priorities rather than size
- If process P_i generates a page fault,
  - select for replacement one of its frames
  - select for replacement a frame from a process with lower priority number

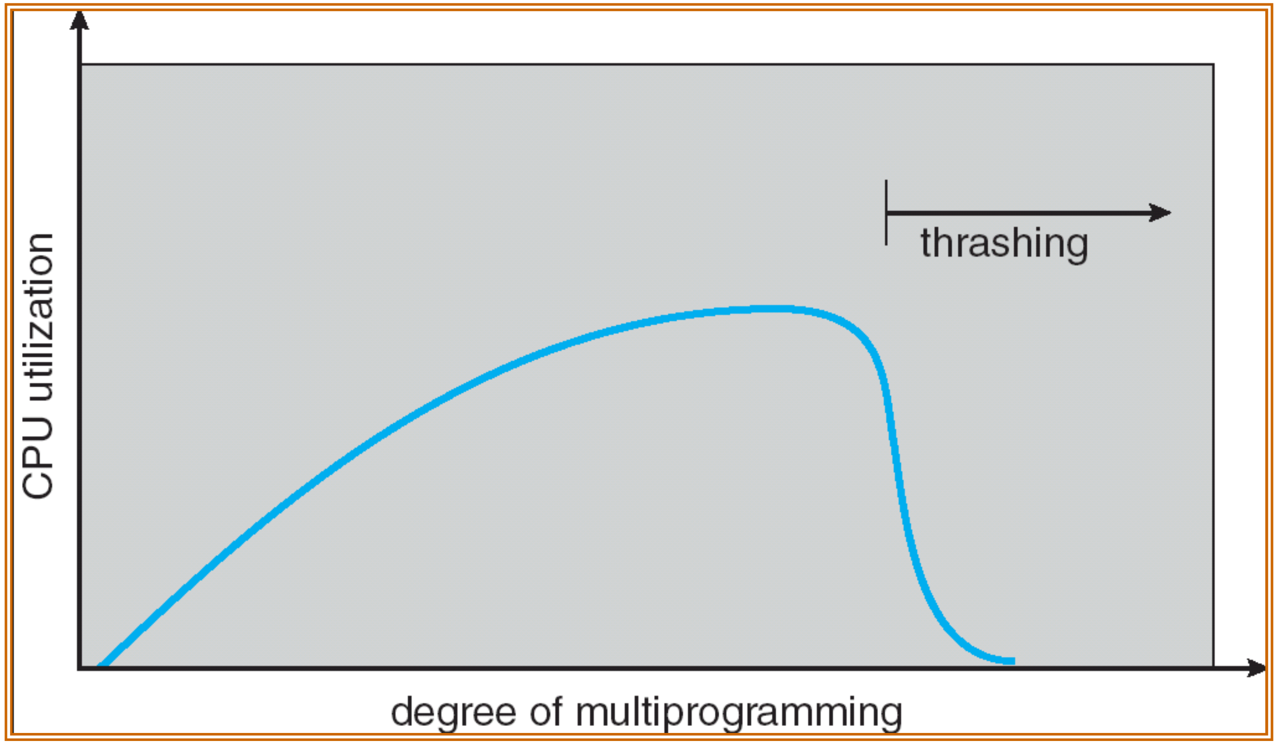
Global vs. Local Allocation

- **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” frames, the page-fault rate is very high.
- **Thrashing**
  - a process is busy swapping pages in and out
- This caused by:
  - low CPU utilization
  - operating system thinks that it needs to increase the degree of multiprogramming
  - another process added to the system

Thrashing (Cont.)



Solutions to Thrashing

- Use local allocation
- Use priority allocation
  - not good solution
- Working set model
  - A suitable solution

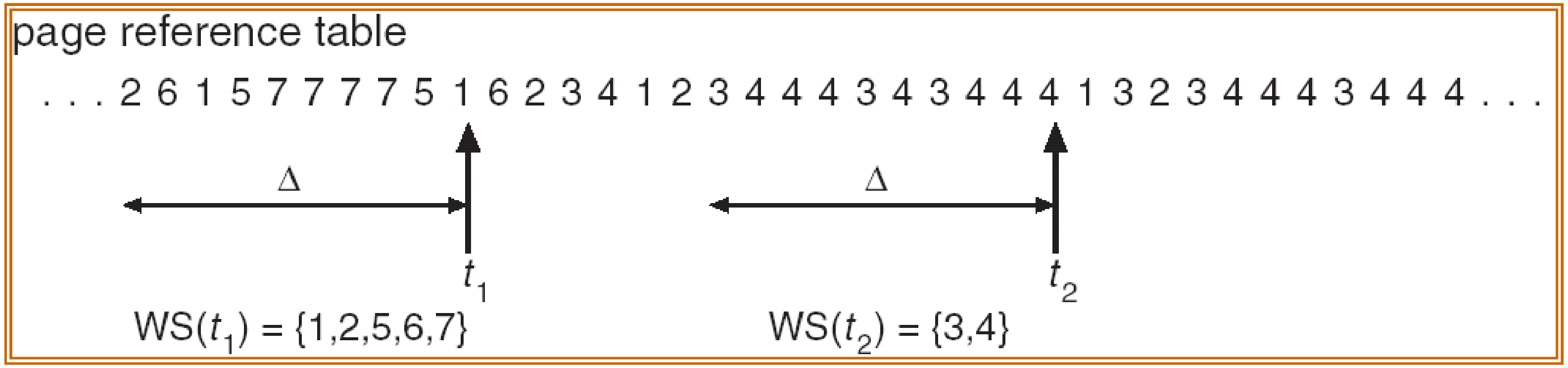
Question

- Which of the following is incorrect about priority allocation?
  A. higher priority process is allocated first
  B. it prevents thrashing from happening
  C. frames are allocated globally
  D. it does not prevent thrashing from happening

Working-Set Model

- $\Delta \equiv$ working-set window
  - a number of page references, e.g. 10,000
- Working set of Process P_i
  - WSS_i =total number of pages referenced in the most recent Δ (varies in time)
  - 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
- $D = \Sigma\ WSS_i \equiv$ total demand frames
- if $D > m$ (total of frames)$\Rightarrow$ Thrashing
- Policy if D > m, then suspend one process

Working-set model



Keeping Track of the Working Set

- Approximate with interval timer + a reference bit
- Example: $\Delta = 10{,}000$
  - Timer interrupts after every 5000 time units
  - Keep in memory 2 bits for each page
  - Whenever a timer interrupts copy and sets the values of all reference bits to 0
  - If one of the bits in memory = 1 $\Rightarrow$ page in working set
- Why is this not completely accurate?
- Improvement = 10 bits and interrupt every 1000 time units

Question

- Suppose a delta =10; reference string
  - 2 6 1 5 7 7 7 7 5 1 6 2 3 4 4 4 3 4 4 4 1 3 2 3 4 4 4 3 4 4 4 …
- Which of the following is the correct WSS at 20th reference?

A. {2 3 4 6}

B. {2 3 4 5 6}

C. {1 2 3 4 6}

D. {7 1 2 3 4 6}


Please wait...
Microsoft
Microsoft
Windows 2000
Advanced Server
Windows is shutting down...

Question?