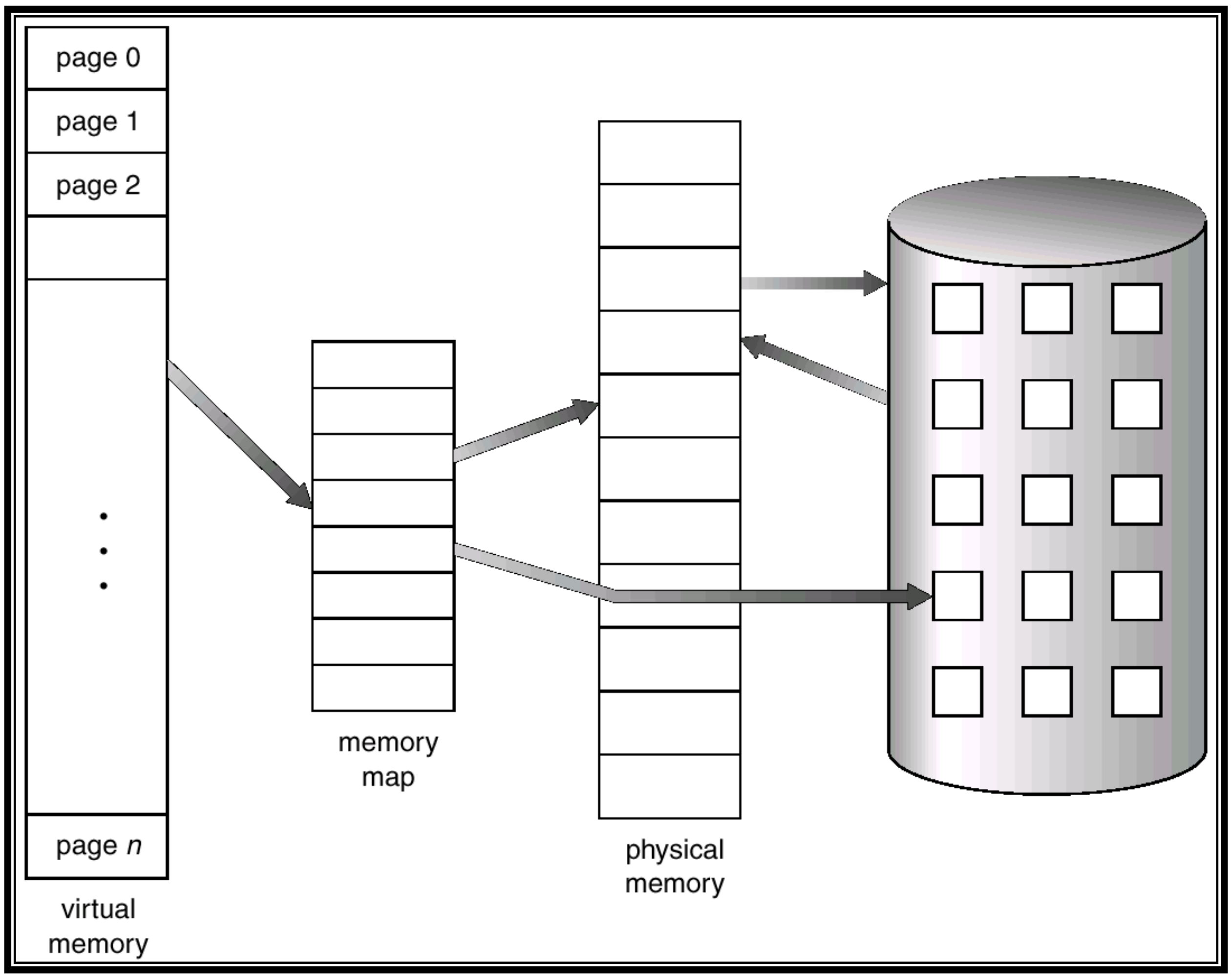
Virtual Memory

Basic Concept

- Usually, only part of the program needs to be in memory for execution
- Allow logical address space to be larger than physical memory size
- Bring only what is needed in memory when it is needed
- Virtual memory implementation
  - Demand paging
  - Demand segmentation

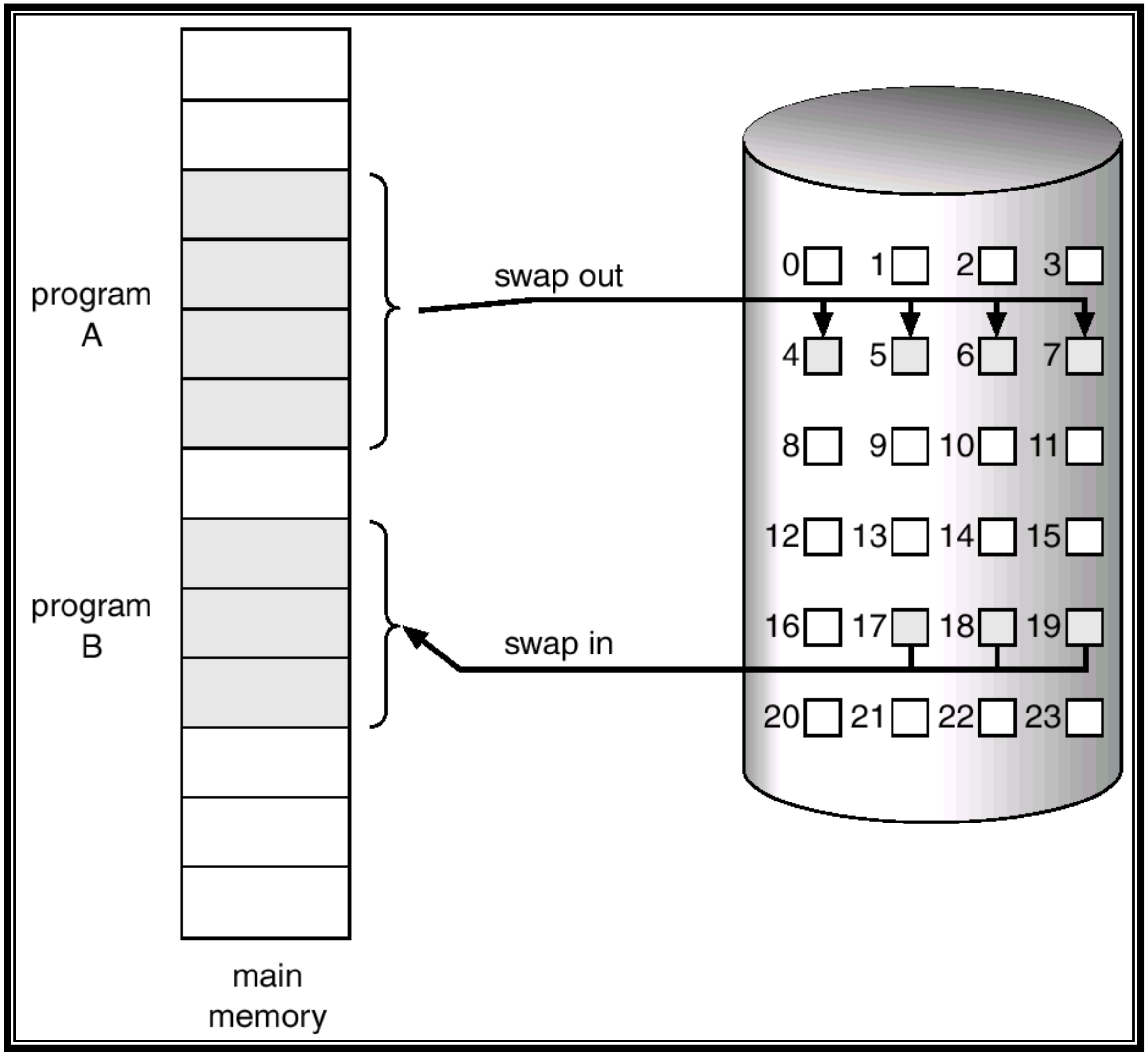
Virtual Memory That is Larger Than Physical Memory



Demand Paging

- Bring a page into memory only when it is needed (on demand)
  - Less I/O needed to start a process
  - 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

Transfer of a Paged Memory to Contiguous Disk Space



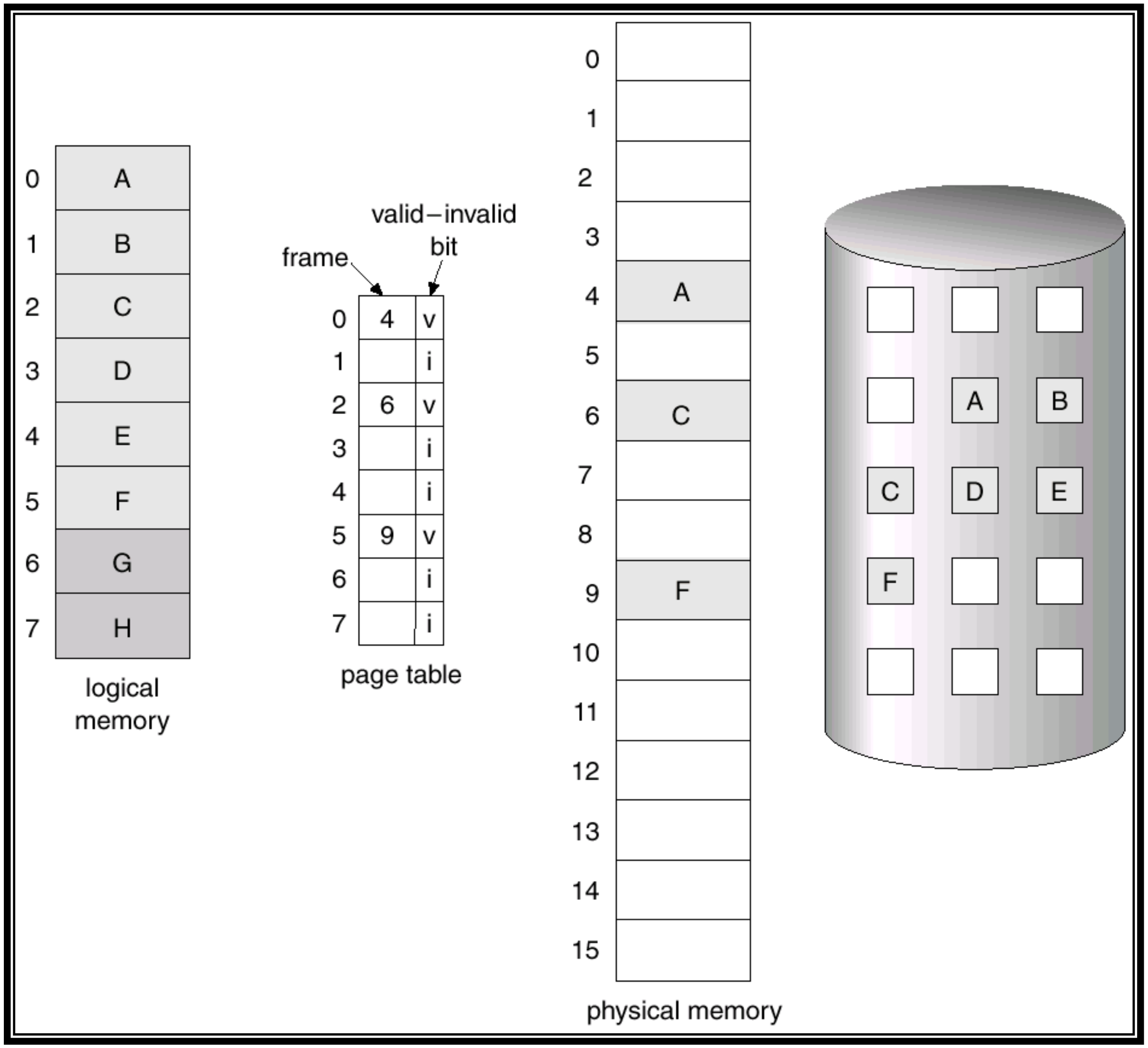
Some questions

- How to know if a page is in memory?
  - Valid bit
- If not present, what happens during reference to a logical address in that page?
  - Page fault
- If not present, where in disk is it?
- If a new page is brought to memory, where should it be placed?
  - Any free page frame
- What if there is no free page frame?
  - Page replacement policies
- Is it always necessary to copy a page back to disk on replacement?
  - Dirty/Modified bit

Valid-Invalid Bit

- With each page table entry a valid–invalid bit is associated
  (1 $\Rightarrow$ in-memory, 0 $\Rightarrow$ not-in-memory)
- Initially valid–invalid but is set to 0 on all entries
- Set to 1 when memory is allocated for a page (page brought into memory)
- Address translation steps:
  - Use page no. in address to index into page table
  - Check valid bit
  - If set, get page frame start address, add offset to get memory address, access memory
  - If not set, page fault

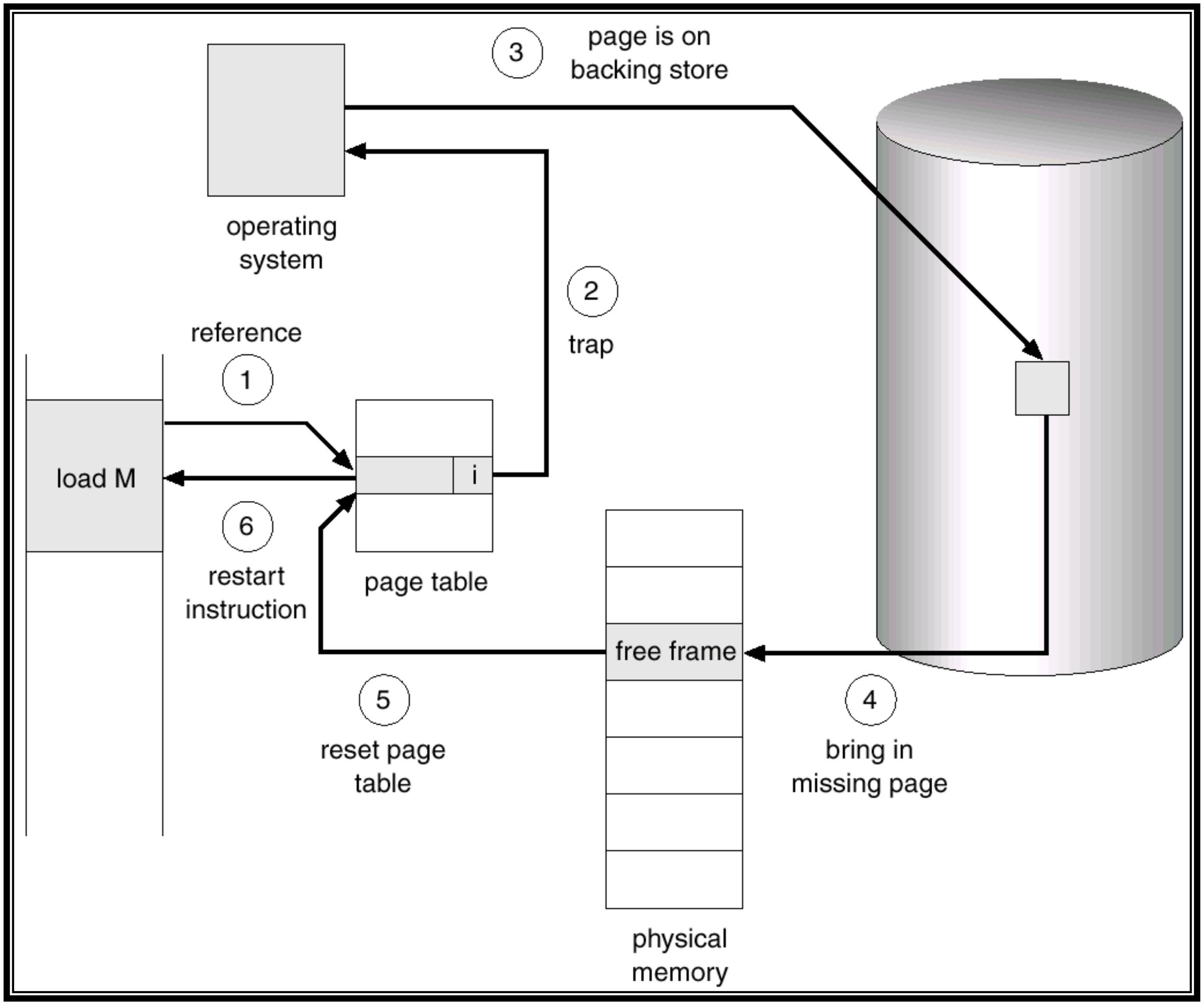
Page Table When Some Pages Are Not in Main Memory



Page Fault

- If there is ever a reference to a page, first reference will trap to OS $\Rightarrow$ page fault
- OS looks at another table to decide:
  - Invalid reference $\Rightarrow$ abort
  - Just not in memory, get from disk
- Get empty frame
- Get disk address of page (in PTE)
- Swap page into frame from disk (context switch for I/O wait)
- Modify PTE entry for page with frame no., set valid bit = 1.
- Restart instruction: Least Recently Used
  - block move
  - auto increment/decrement location

Steps in Handling a Page Fault



Performance of Demand Paging

- Page Fault Rate $0 \le p \le 1.0$
  - if $p = 0$ no page faults
  - if $p = 1$, every reference is a fault

- Effective Access Time (EAT)

$$
\begin{aligned}
\text{EAT} = (1 - p) \times \text{memory access} \\
+\, p\ (\text{page fault overhead} \\
+\, [\text{swap page out}\ ] \\
+\, \text{swap page in} \\
+\, \text{restart overhead})
\end{aligned}
$$

Demand Paging Example

- Memory access time = 150 ns
- 50% of the time the page that is being replaced has been modified and therefore needs to be swapped out.
- Swap Page Time = 10 msec = 10^7 ns

$$EAT = (1 - p) \times 150 + p\,(1.5 \times 10^7)$$
$$\approx 150 + p\,(1.5 \times 10^7) \text{ ns}$$

- EAT = 200 ns will need p = 0.0000033 !
- EAT = 165 ns (10% loss) will need p = 0.000001, or 1 fault in 1000000 accesses

Page in Disk

- Swap space: part of disk divided in page sized slots
  - First slot has swap space management info such as free slots etc.
- Pages can be swapped out from memory to a free page slot
  - Address of page <swap partition no., page slot no.>
  - Address stored in PTE
- Read-only pages can be read from the file system directly using memory-mapped files

What happens if there is no free frame?

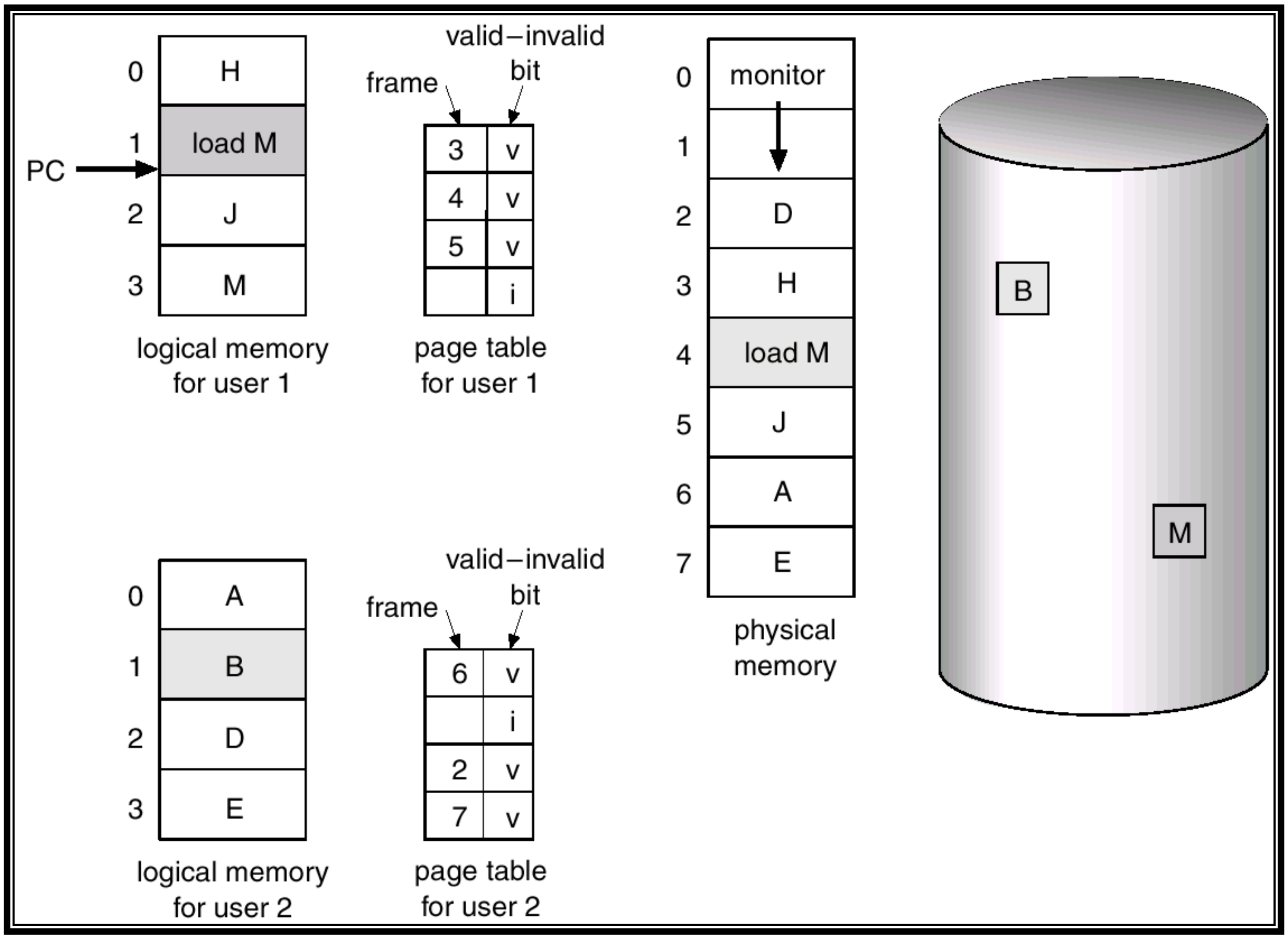
- Page replacement – find some page in memory, but not really in use, swap it out
  - algorithm
  - performance – want an algorithm which will result in minimum number of page faults.

- Same page may be brought into memory several times

Page Replacement

- Prevent over-allocation of memory by modifying page-fault service routine to include page replacement
- Use *modified* (*dirty*) *bit* to reduce overhead of page transfers
  - Set to 0 when a page is brought into memory
  - Set to 1 when some location in a page is changed
  - Copy page to disk only if bit set to 1
- Page replacement completes separation between logical memory and physical memory – large virtual memory can be provided on a smaller physical memory

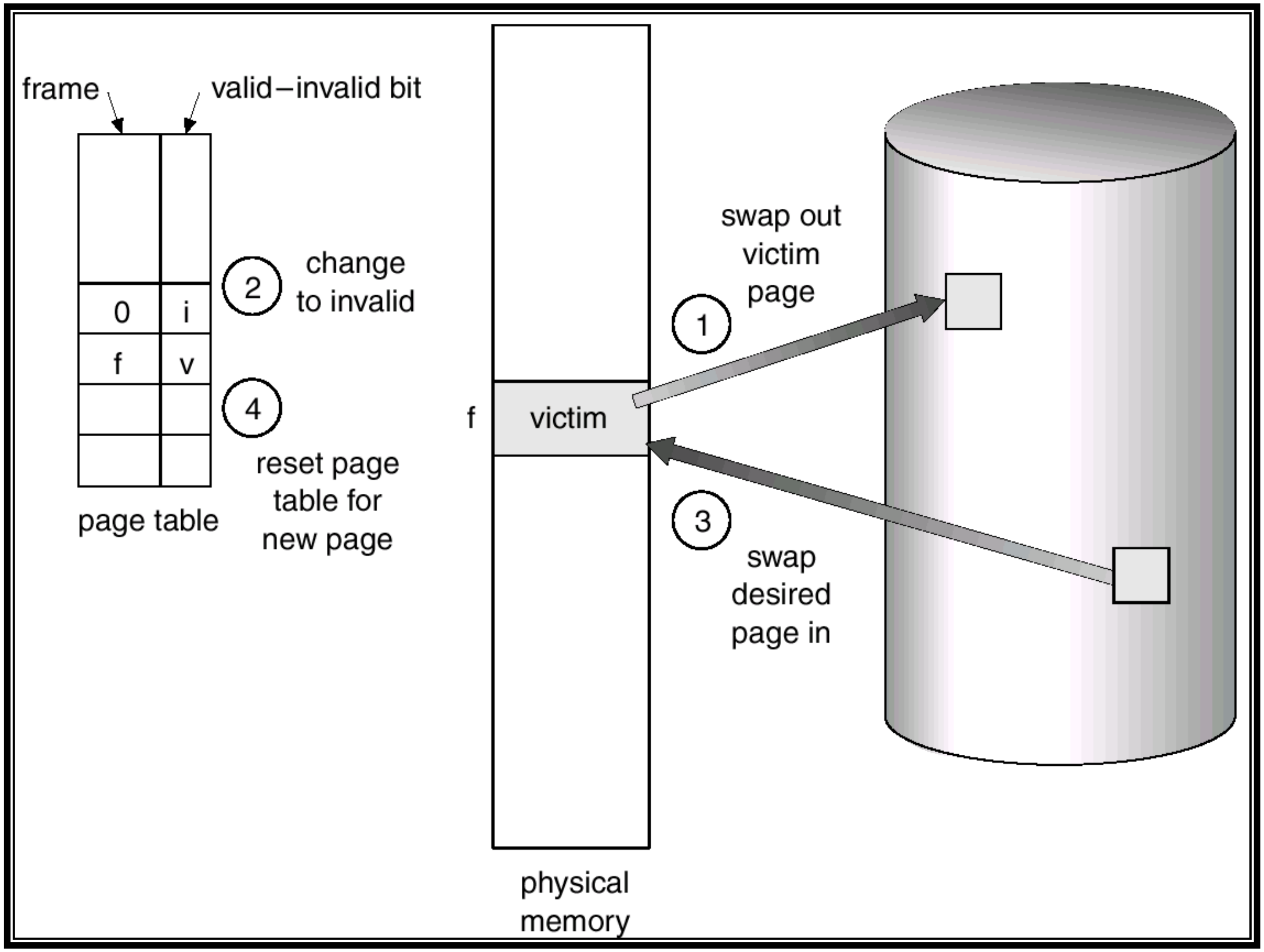
Need For Page Replacement



Basic Page Replacement

- Find the location of the desired page on disk
- Find a free frame
  - If there is a free frame, use it
  - If there is no free frame, use a page replacement algorithm to select a *victim* frame
- Copy the to-be replaced frame to disk if needed
- Read the desired page into the (newly) free frame. Update the page and frame tables.
- Restart the process

Page Replacement

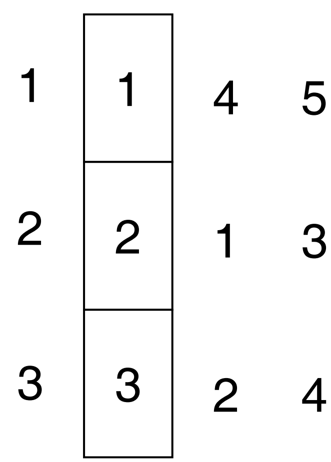


Page Replacement Algorithms

- Want lowest page-fault rate
- Evaluate algorithm by running it on a particular string of memory references (reference string of pages referenced) and computing the number of page faults on that string

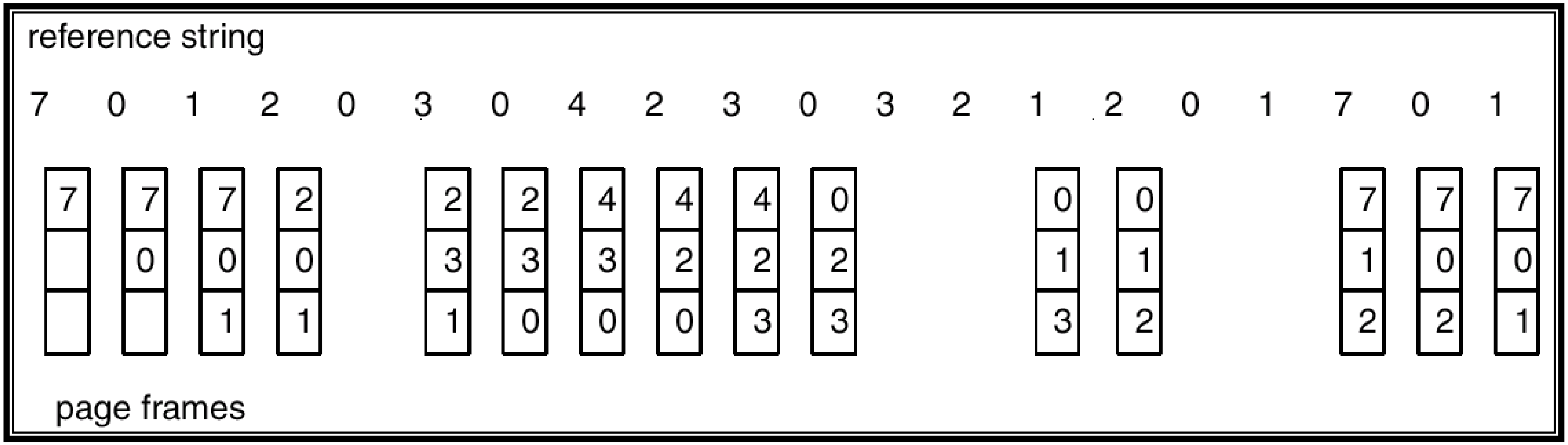
First-In-First-Out (FIFO) Algorithm

- Reference string: 1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5
- 3 frames (3 pages can be in memory at a time per process)



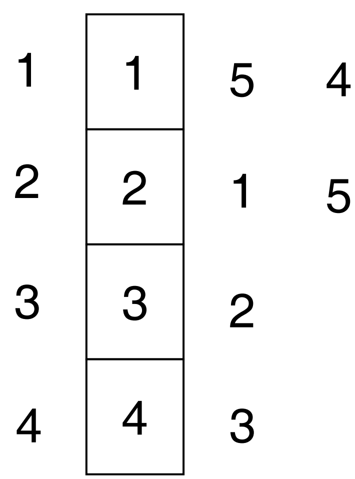
9 page faults

FIFO Page Replacement



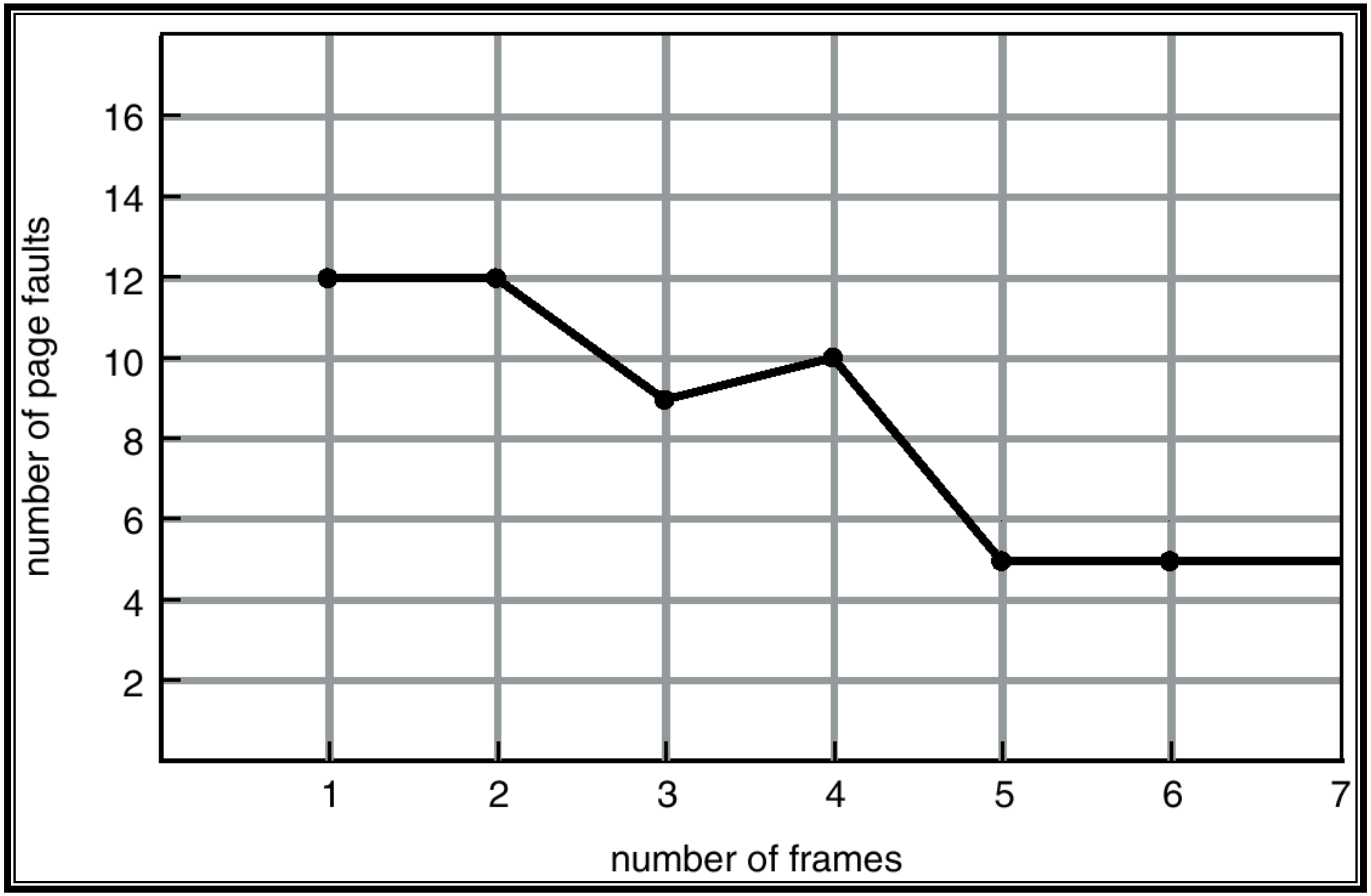
Belady’s Anamoly

- The number of page faults may increase with increase in number of page frames for FIFO
    - Counter-intuitive
- Consider 4 page frames



10 page faults

Belady’s Anamoly



Optimal Algorithm

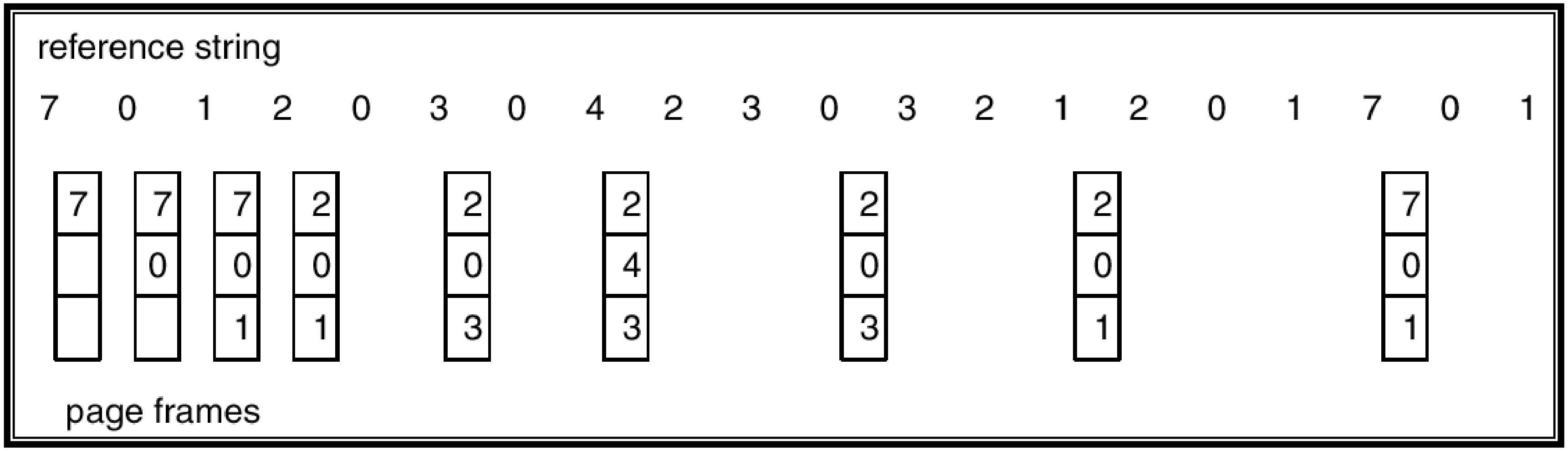
- Replace page that will not be used for longest period of time.
- 4 frames example: 1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5

1	4
2	
3	
4	5

6 page faults

- How do you know this?
- Used for measuring how well your algorithm performs

Optimal Page Replacement



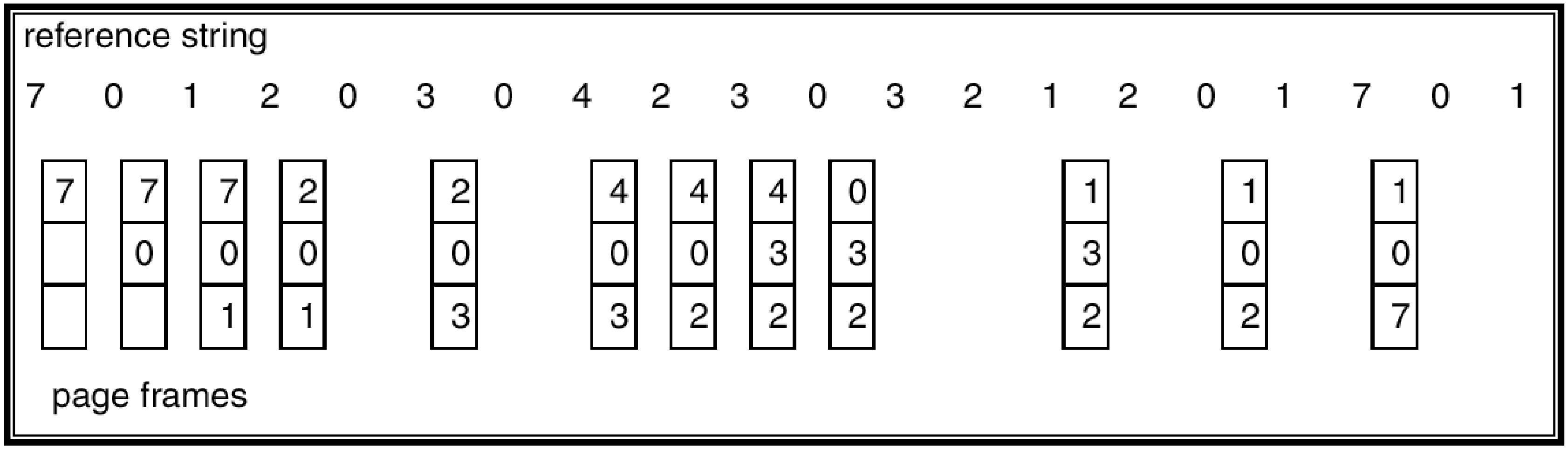
Least Recently Used (LRU) Algorithm

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

1	5	
2		
3	5	4
4	3	

- Counter implementation
  - Every page entry has a counter; every time page is referenced through this entry, copy the clock into the counter
  - When a page needs to be changed, look at the counters to determine which are to change

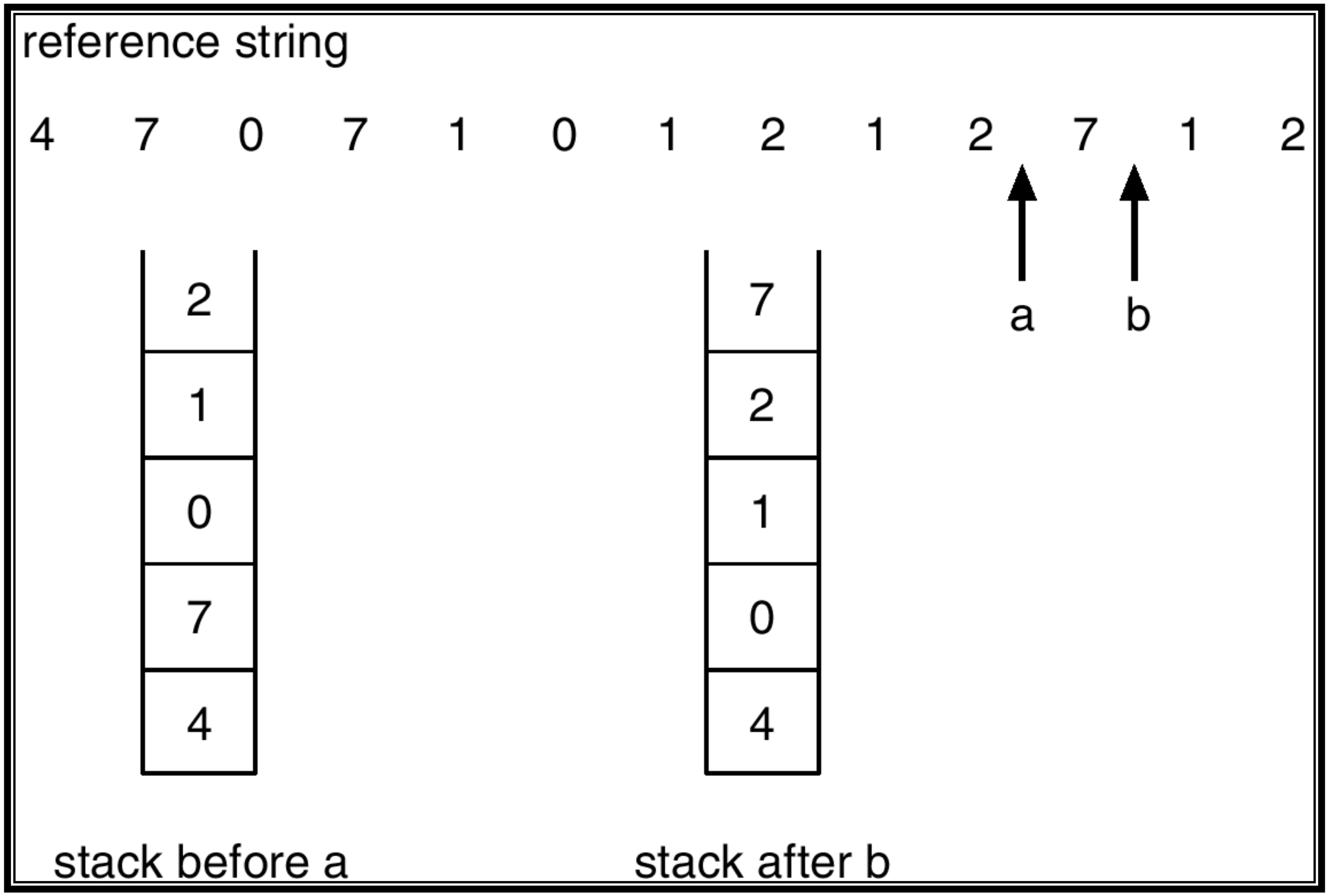
LRU Page Replacement



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

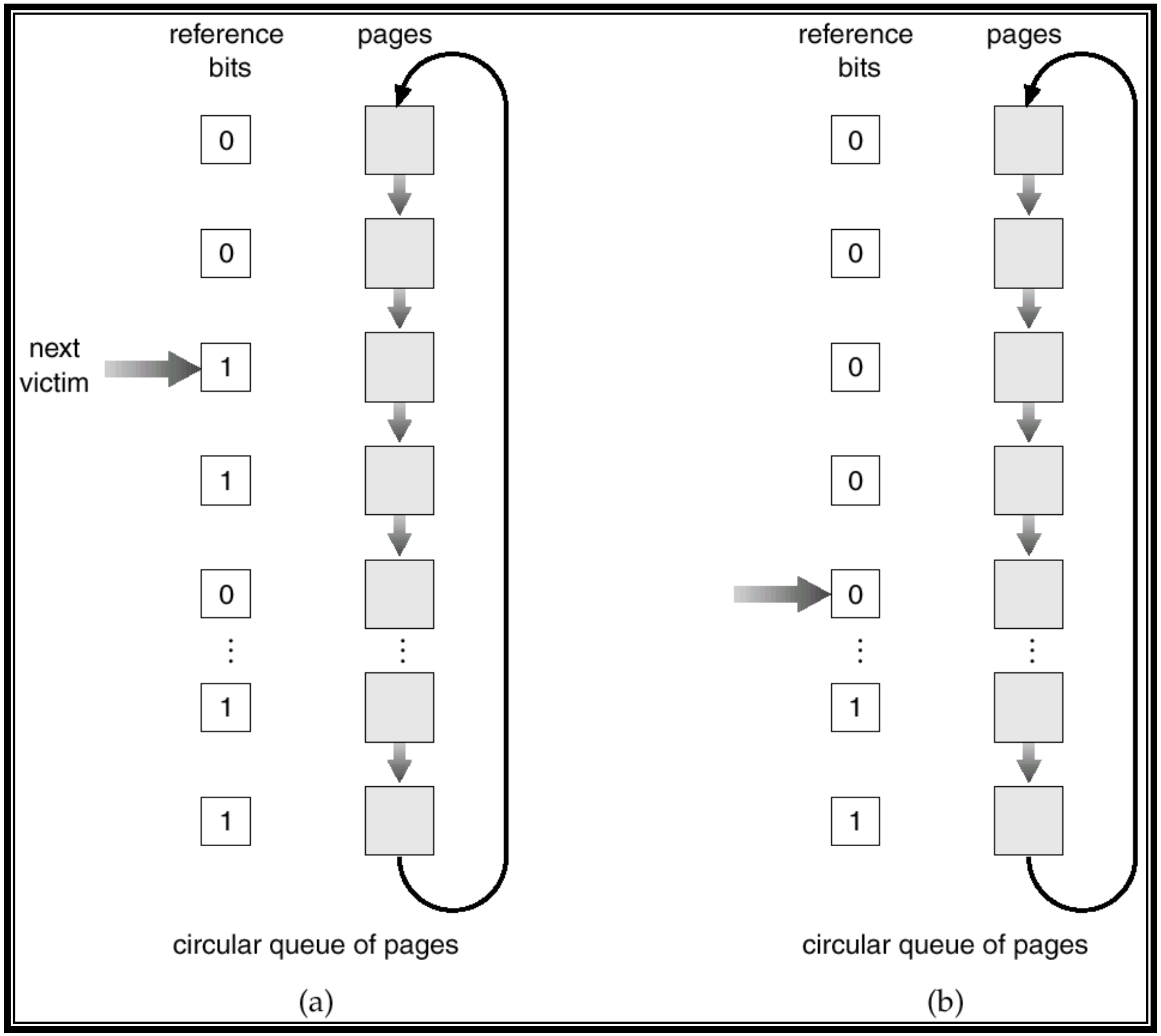


LRU Approximation Algorithms

- Reference bit
  - 1 bit per page frame, initially = 0
  - When page frame is referenced, bit set to 1
  - Periodically reset to 0
  - Replace the one which is 0 (if one exists). We do not know the order, however
- Additional Reference Bits algorithm
  - K bits marked 1 to K from MSB
  - ith bit indicates if page accessed in the ith most recent interval
  - At every interval (timer interrupt), right shift the bits (LSB drops off), shift reference bit to MSB, and reset reference bit to 0
  - To replace, find the frame with the smallest value of the K bits

- Second chance
  - Need reference bit
  - Clock replacement
  - If page to be replaced (in clock order) has reference bit = 1, then:
    - set reference bit 0
    - leave page in memory
    - replace next page (in clock order), subject to same rules

Second-Chance (clock) Page-Replacement Algorithm



Counting Algorithms

- Keep a counter of the number of references that have been made to each page
- LFU Algorithm: replaces page with smallest count
- 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 some minimum number of pages
- No process should use up nearly all page frames
- Two major allocation schemes
  - fixed allocation
  - priority allocation

Fixed Allocation

- Equal allocation – e.g., if 100 frames and 5 processes, give each 20 pages.
- Proportional allocation – Allocate according to the size of process

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.

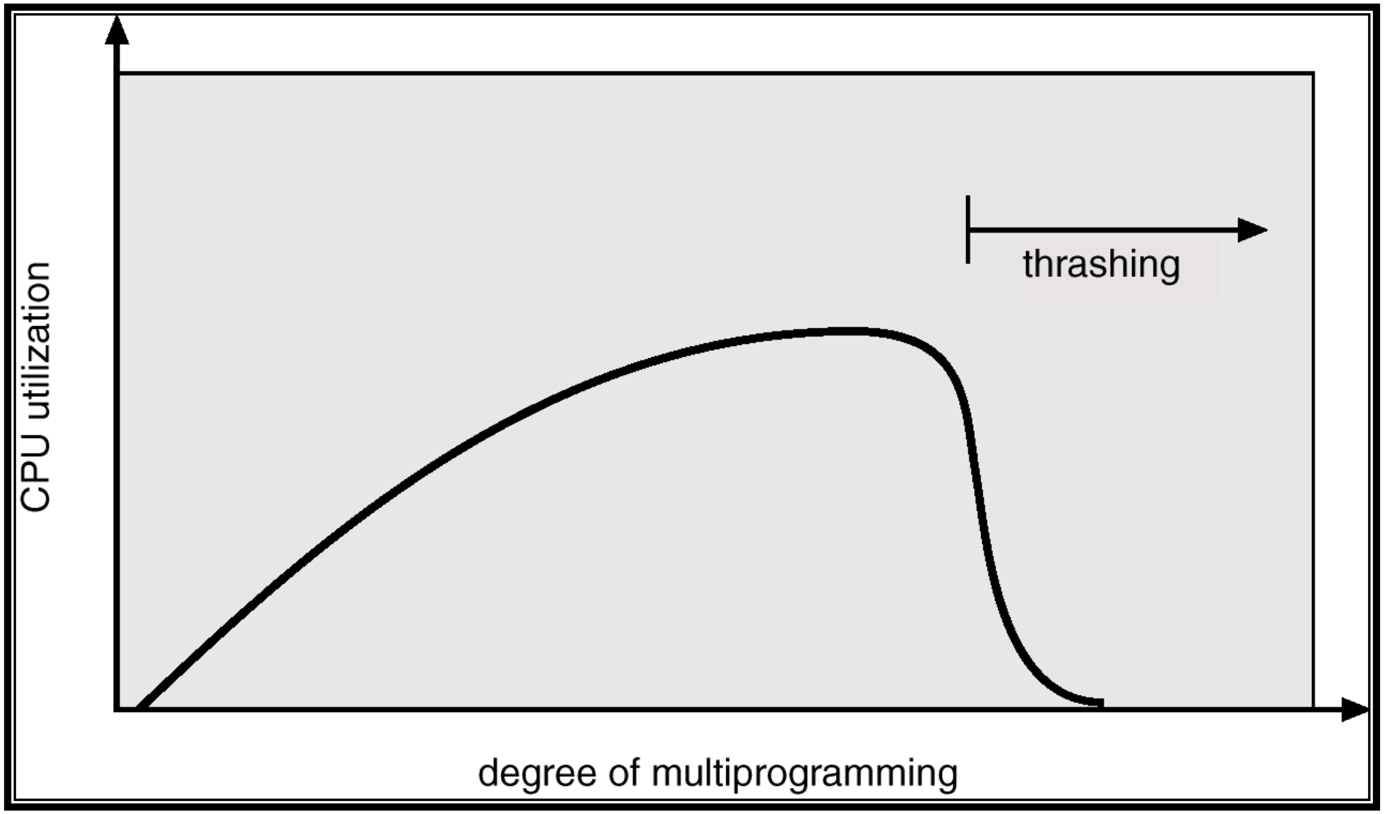
Why does paging work?

- Locality of reference
  - Processes tend to access locations which are close to each other (spatial locality) or which are accessed in the recent past (temporal locality)
- Locality of reference implies once a set of page is brought in for a process, less chance of page faults by the process for some time
- Also, TLB hit ratio will be high
- Working set – the set of pages currently needed by a process
- Working set of a process changes over time
  - But remains same for some time due to locality of reference

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
  - Adds to the problem as even less page frames are available for each process

- **Thrashing** ≡ a process is busy swapping pages in and out



- Thrashing occurs when

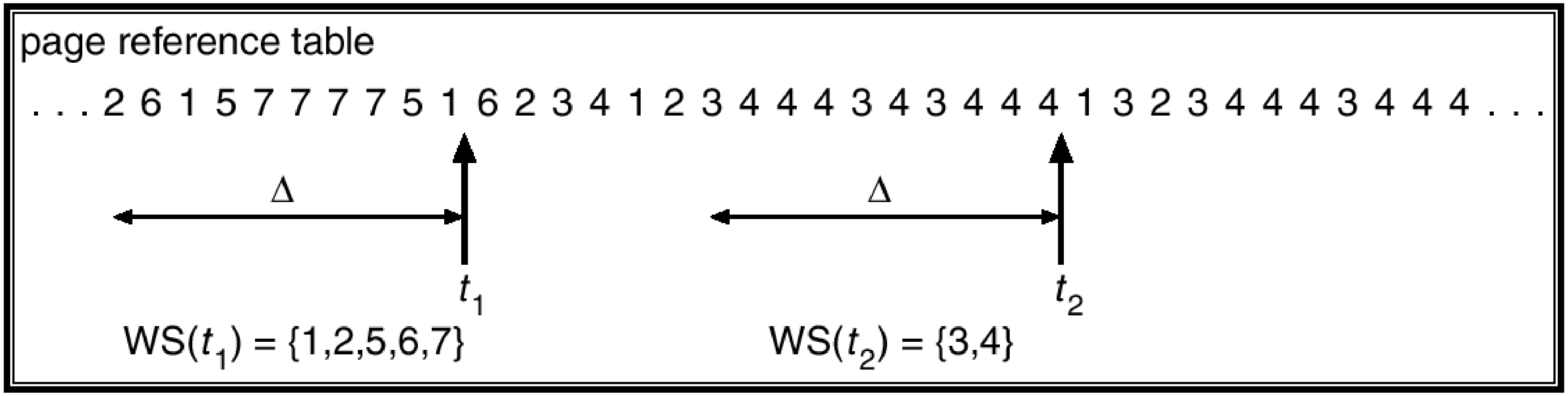
  Σ working set of all processes

  $>$ total memory size

Working-Set Model

- $\Delta \equiv$ working-set window $\equiv$ a fixed number of page references
  Example: 10,000 instruction
- WSS_i (working set of Process P_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 $D > m \Rightarrow$ Thrashing
- Policy if D > m, then suspend one of the processes.

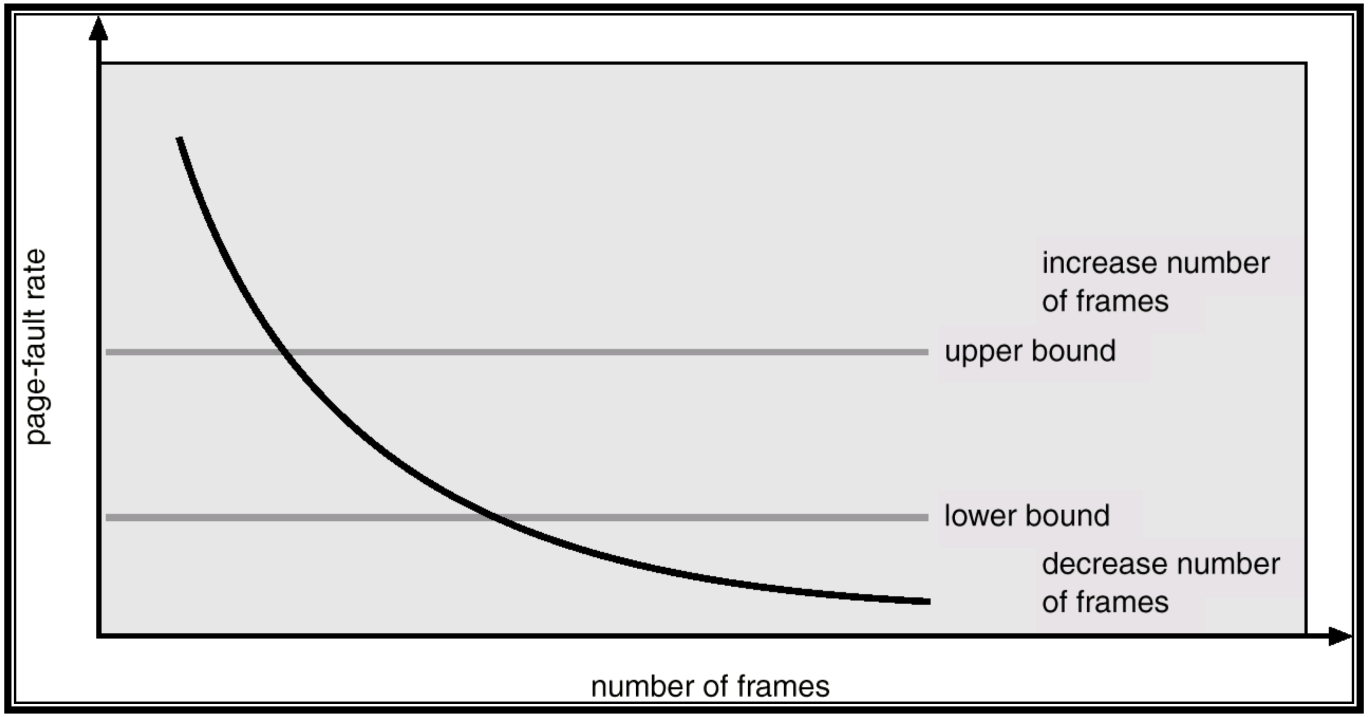
Working-set model



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

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

What should the page size be?

- Reduce internal fragmentation
  - Smaller is better
- Reduce table size
  - Larger is better
- Reduce I/O overhead
  - Smaller is better
- Capture locality
  - Larger is better
- Need to be chosen judiciously

Other Considerations

- Prepaging
  - Bring in pages not referenced yet
- 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 page faults

Other Considerations (Cont.)

- Program structure

```
int A[][] = new int[1024][1024];
Each row is stored in one page
```

Program 1

```
for (j = 0; j < A.length; j++)
    for (i = 0; i < A.length; i++)
        A[i,j] = 0;
```

1024 x 1024 page faults!!

Program 2

```
for (i = 0; i < A.length; i++)
    for (j = 0; j < A.length; j++)
        A[i,j] = 0;
```

1024 page faults

Other Considerations (Cont.)

- I/O Interlock – Pages must sometimes be locked into memory
  - Pages that are used for copying a file from a device must be locked from being selected for eviction by a page replacement algorithm
  - Some OS pages need to be I memory all the time
  - Use a lock bit to indicate if the page is locked and cannot be replaced