Chapter 9: Virtual Memory

Chapter 9: Virtual Memory

- Background
- Demand Paging
- Copy-on-Write
- Translation Lookaside Buffer
- Page Replacement
- Allocation of Frames
- Thrashing

Objectives

- To describe the benefits of a virtual memory system
  - One of the most important of all concepts related to Memory Management is Virtual Memory.
- To explain the concepts of demand paging, page-replacement algorithms, and allocation of page frames
- To discuss the principle of the working-set model

Background

- Virtual memory – separation of user logical memory from physical memory.
  - Virtual Memory refers to the concept whereby a process with a larger size than available memory can be loaded and executed by loading the process in parts.
  - The program memory is divided into ***pages*** and the available physical memory into ***frames***.
  - If a process attempts to access a page that is not available in the main memory and the information of which does not exist in its page table, a ***page fault occurs***.
  - The Operating System now takes care of ***swapping*** this page in to the main memory from the backing store.
  - Only 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

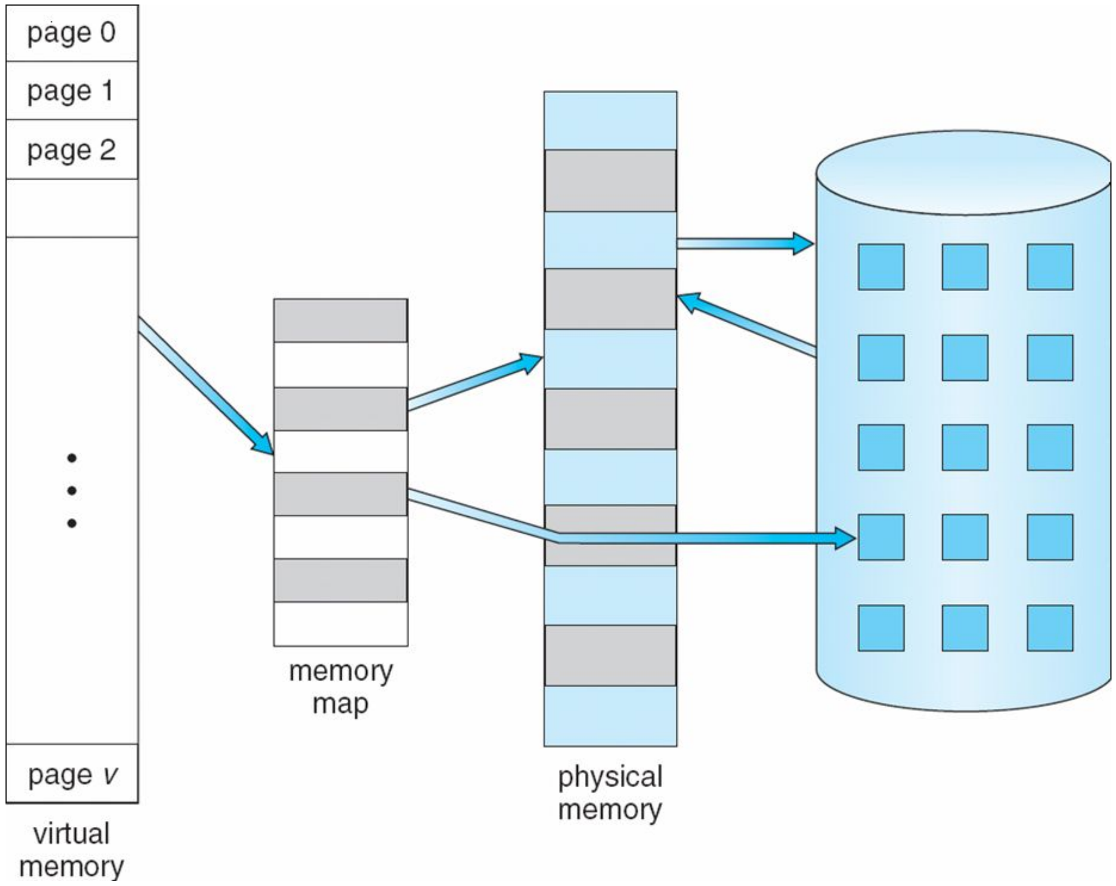
- Virtual memory can be implemented via:
  - Demand paging
    - Demand Paging refers to *loading a page of program code* from disk into memory *as and when it is required* by the program.
  - Demand segmentation

Virtual memory

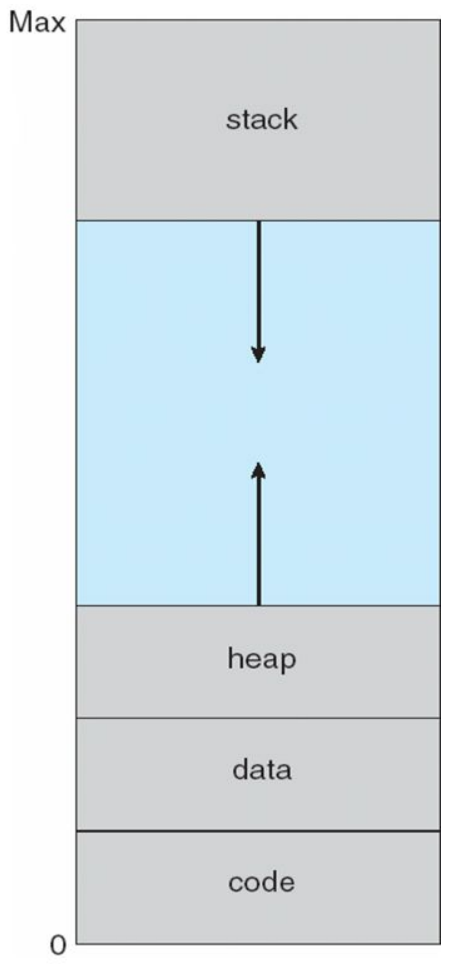
- The data and the instructions are read from the main memory and then after the execution of the instruction is over,
  - they are written back onto the memory.
- But what happens when the program in consideration is larger than the size of the available memory?
  - ***Virtual Memory*** is a concept that addresses this issue by allowing a program than is even larger than the size of the available free memory to be loaded and executed and eliminate the chances of *external fragmentation.*
  - The Operating System maps the programmer's virtual addresses to real hardware storage addresses.
  - Mapping implies the correspondence between the virtual addresses and the physical addresses using virtual translation mechanisms as decided by the Operating System.

- The program memory is divided into pages and the available physical memory into frames. The page size is always equal to the frame size.
  - The page size is generally in a **power of** 2 to avoid the calculation involved to get the page number and the offset from the CPU generated address.
- In the virtual memory systems, the addresses that the application programs deal with are known as virtual addresses.
  - These virtual addresses used by the application program are mapped to physical addresses by translation of these virtual addresses.
  - This is taken care of by the virtual memory system's address translation mechanism by mapping these virtual addresses to frame addresses using the Page Map Tables.
  - This is what we call the physical address in the main memory that can be used to refer to the contents from the memory.
  - The process address space implies the number of unique addresses needed to hold both the process and its data. The virtual address space refers to the memory space of the virtual addresses.

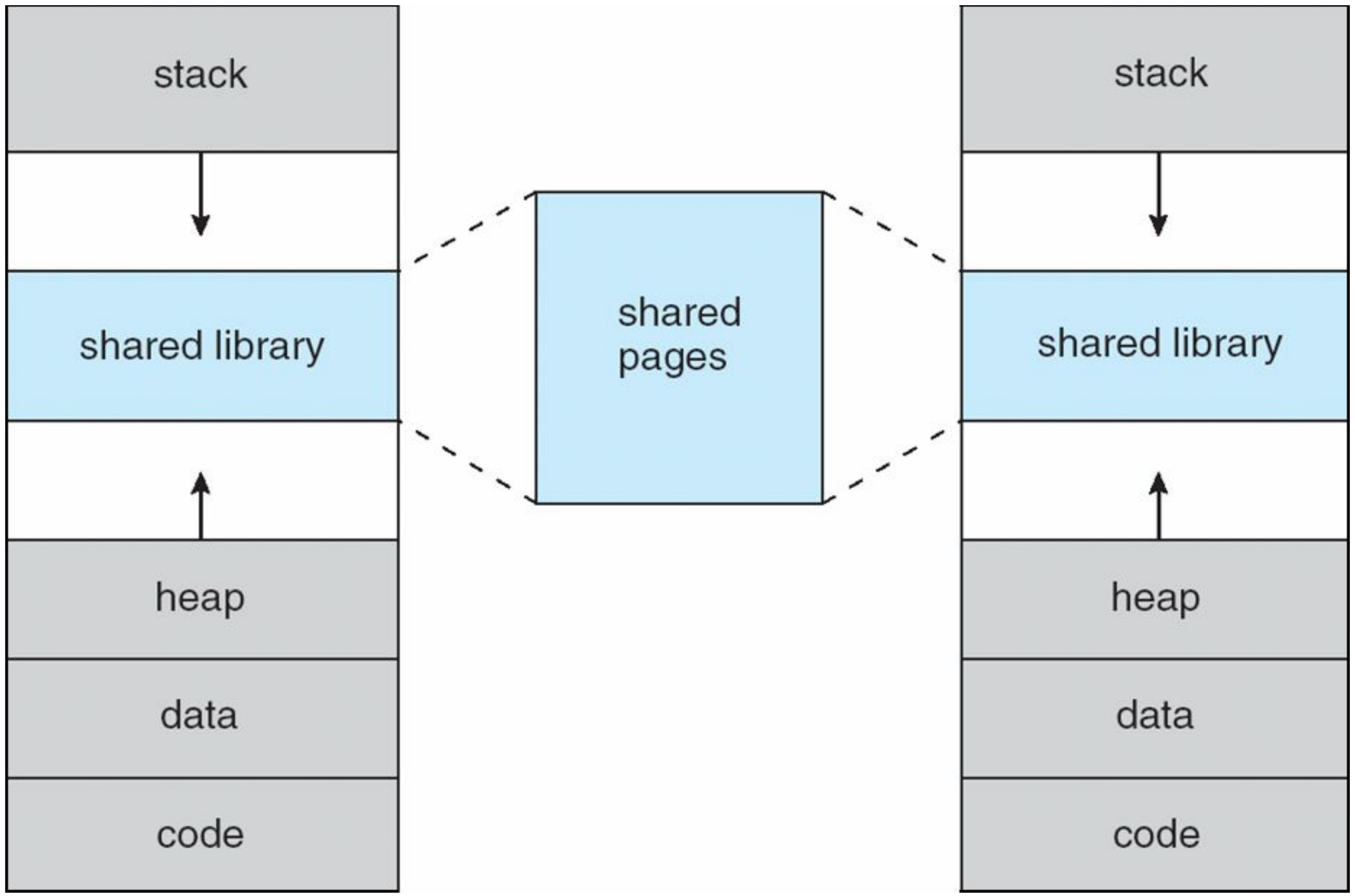
Virtual Memory That is Larger Than Physical Memory



Virtual-address Space



Shared Library Using Virtual Memory



Advantages and Disadvantages of Virtual Memory Systems

§ The *primary advantage* or objective of Virtual Memory systems is the ability to ***load and execute a process*** that requires a ***larger amount of memory*** than what is available by loading the ***process in parts*** and then executing them.

§ The *disadvantage* is that Virtual Memory systems tend to be ***slow and require additional support*** from the system's hardware for ***address translations***.

§ It can be said that ***the execution speed*** of a process in a Virtual Memory system can ***equal***, but ***never exceed***, the execution speed of the same process with Virtual Memory *turned off*.

§ Hence, ***we do not have an advantage with respect to the execution speed of the process.***

§ The ***advantage*** lies in the ability of the system to ***eliminate external fragmentation***.

§ The other disadvantage of Virtual Memory systems is the possibility of ***Thrashing*** *due to excessive Paging and Page faults*.

§ In may be noted that **Trash Point** is a point after which the execution of a process comes ***to a halt***;

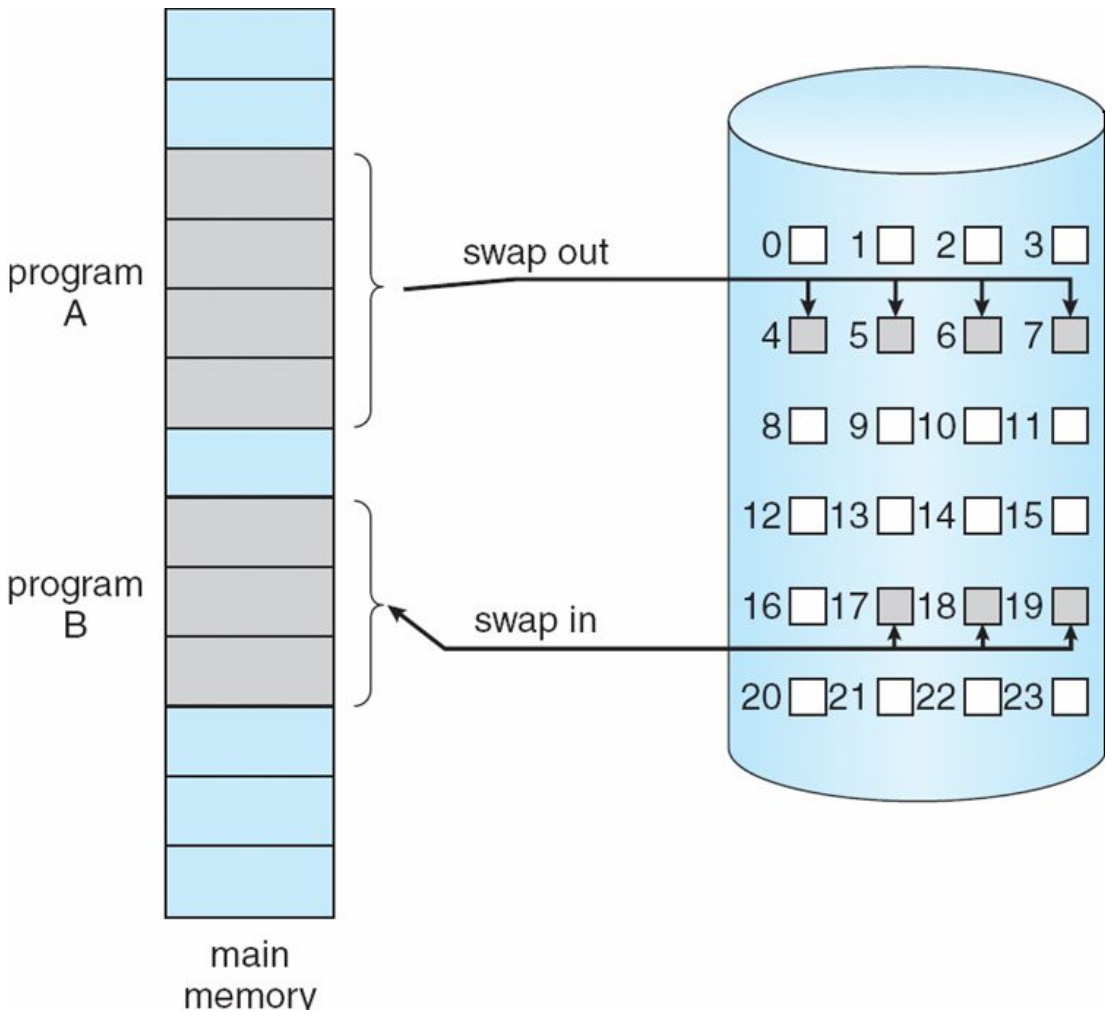
§ the system is busier paging pages in and out of the memory than executing them.

Demand Paging

- In *Virtual Memory Systems* the pages are not loaded in the memory until they are "***demanded***" by a process; hence the name, ***Demand Paging***.
- Demand paging allows the various parts of a process to be brought into physical memory as the process needs them to execute.
- In virtual memory systems, demand paging is a type of swapping in which pages of data are not copied from disk to RAM until they are needed.
- Bring a page into memory only when it is needed
  - Less I/O needed
  - Less memory needed
  - Faster response
  - More users
- Page is needed ⇒ reference to it
  - invalid reference ⇒ abort
  - not-in-memory ⇒ bring to memory

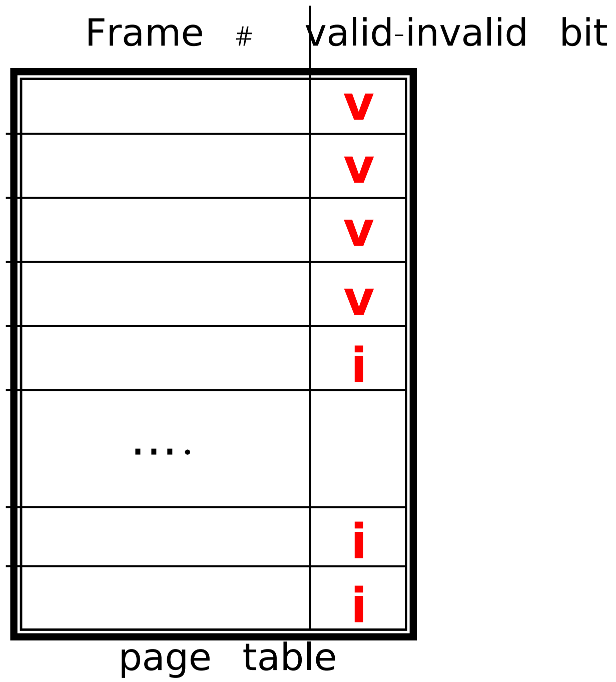
- **Lazy swapper** – never swaps a page into memory unless page will be needed
  - Swapper that deals with pages is a **pager**
- **How Demand Paging Works**
- When the CPU executes an instruction that is not available in the memory, a *Page Fault* occurs.
- This means a page is being referenced,
- The Operating System is now responsible for bringing the appropriate page into memory from the disk.
- The total turnaround time of a process is divided into CPU time and I/O time.
- Disk I/O takes a longer time than CPU and the process must wait until the page has been fetched from the disk.
- A module of the Operating System called the Page Fault Handler is given the control.
    - The information also includes the disk address of the page in the backing storethe information needed to bring the required page into the main memory from the backing store. The information also includes the disk address of the page in the backing store.

Transfer of a Paged Memory to Contiguous Disk Space



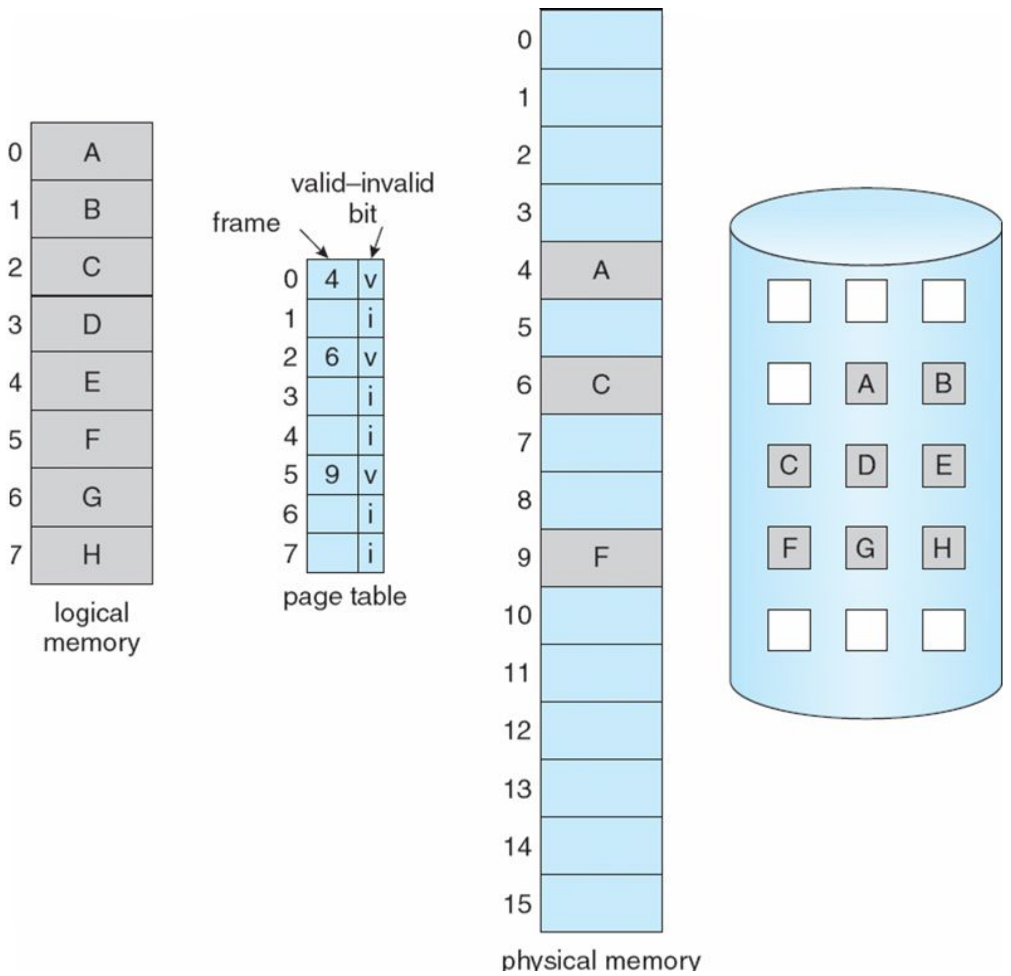
Valid-Invalid Bit

- 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

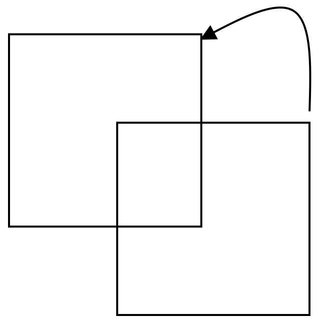


Page Fault

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

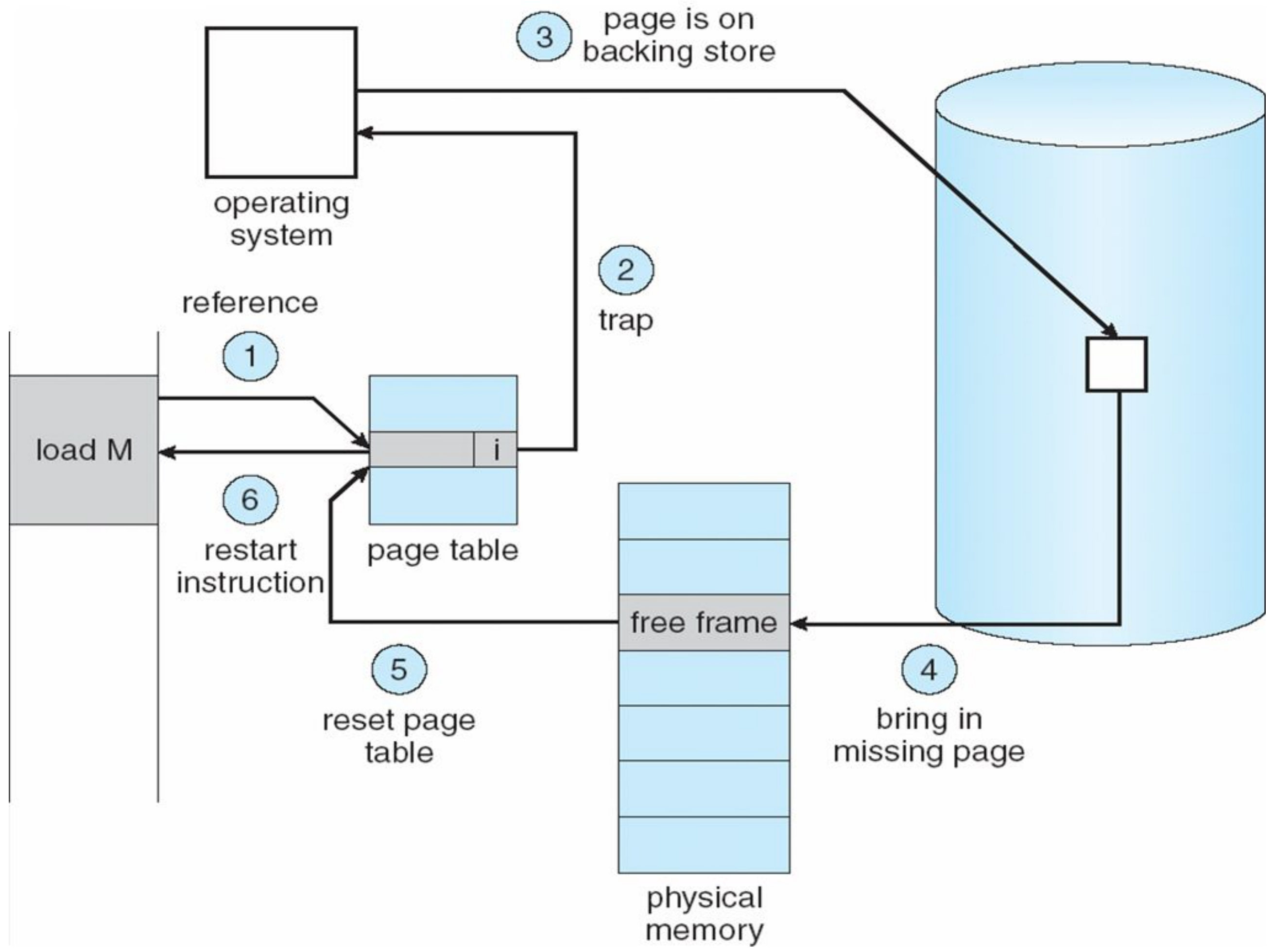
Page Fault (Cont.)

- Restart instruction
  - block move



  - auto increment/decrement location

Steps in Handling a Page Fault



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)

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

)

Process Creation

Virtual memory allows other benefits during process creation:

- Copy-on-Write
- Memory-Mapped Files (later)

Copy-on-Write

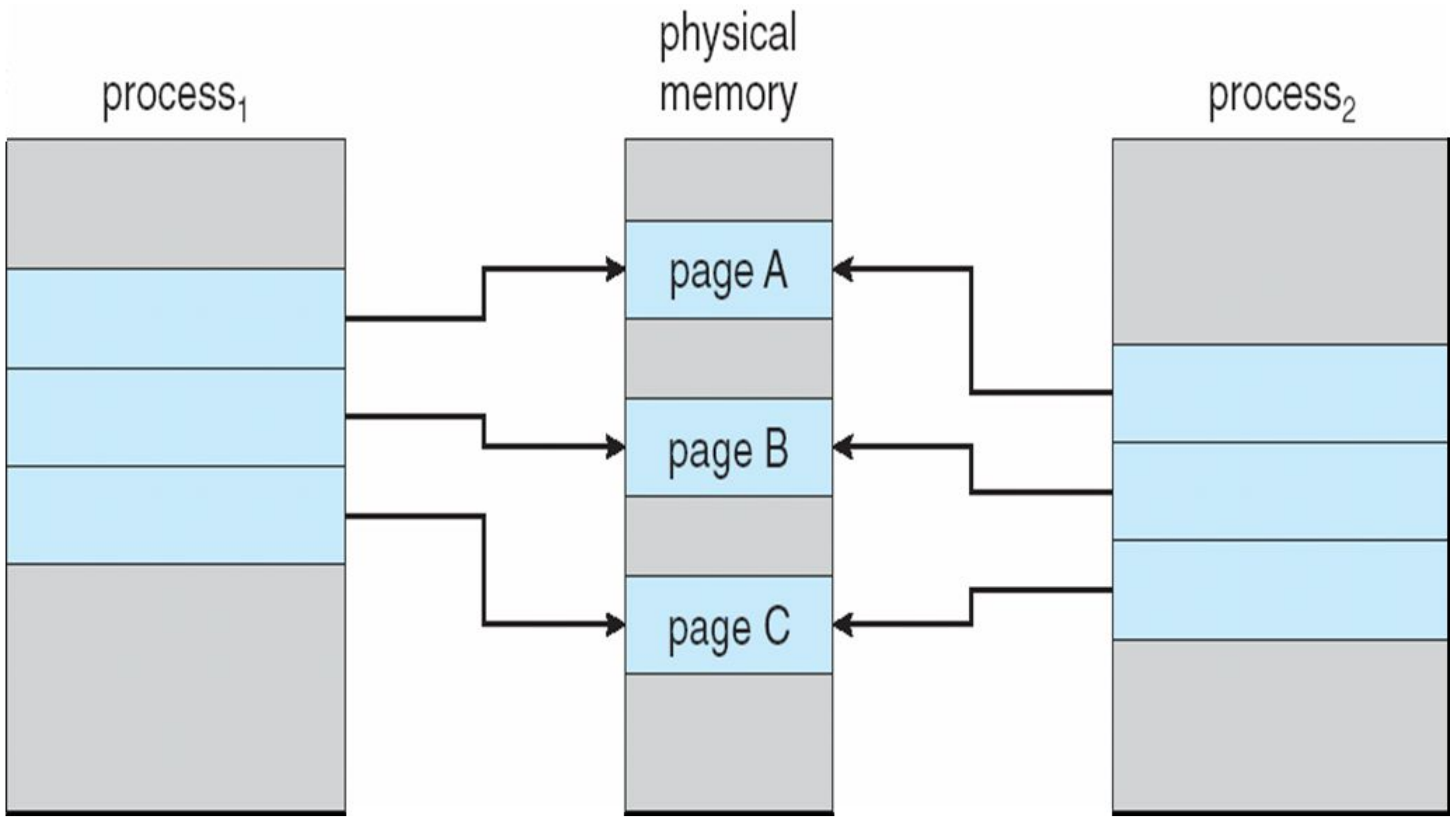
- Copy-on-Write (COW) allows both parent and child processes to initially *share* the same pages in memory

  If either process modifies a shared page, only then the is page copied

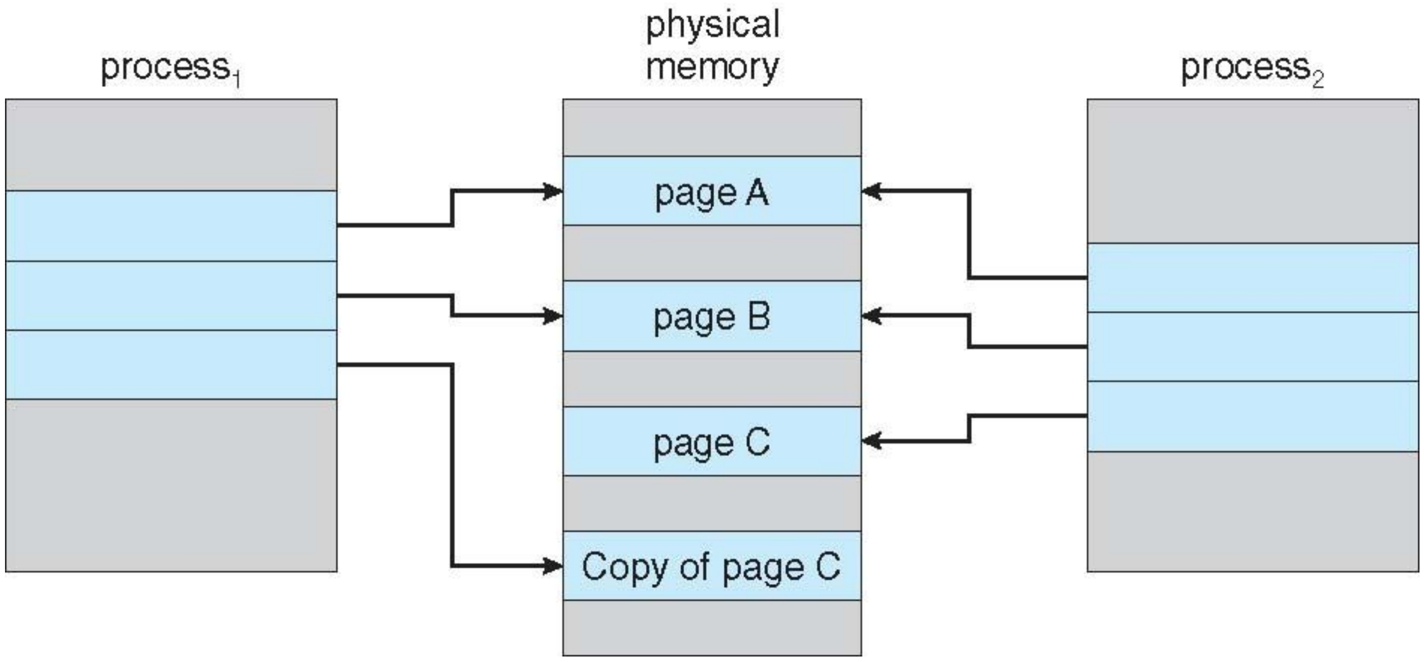
- COW allows more efficient process creation as only modified pages are copied

- Free pages are allocated from a **pool** of zeroed-out pages

Before Process 1 Modifies Page C



After Process 1 Modifies Page C



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

Translation Lookaside Buffer (TLB)

- Each virtual memory reference can cause two physical memory accesses
  - One to fetch the page table
  - One to fetch the data
- To overcome this problem a high-speed cache is set up for page table entries
  - Called a Translation Lookaside Buffer (TLB)
- Contains page table entries that have been most recently used

Translation Lookaside Buffer

- Given a virtual address, processor examines the TLB
- If page table entry is present (TLB hit), the frame number is retrieved and the real address is formed
- If page table entry is not found in the TLB (TLB miss), the page number is used to index the process page table
- First checks if page is already in main memory
  - If not in main memory, a page fault is issued
- The TLB is updated to include the new page entry

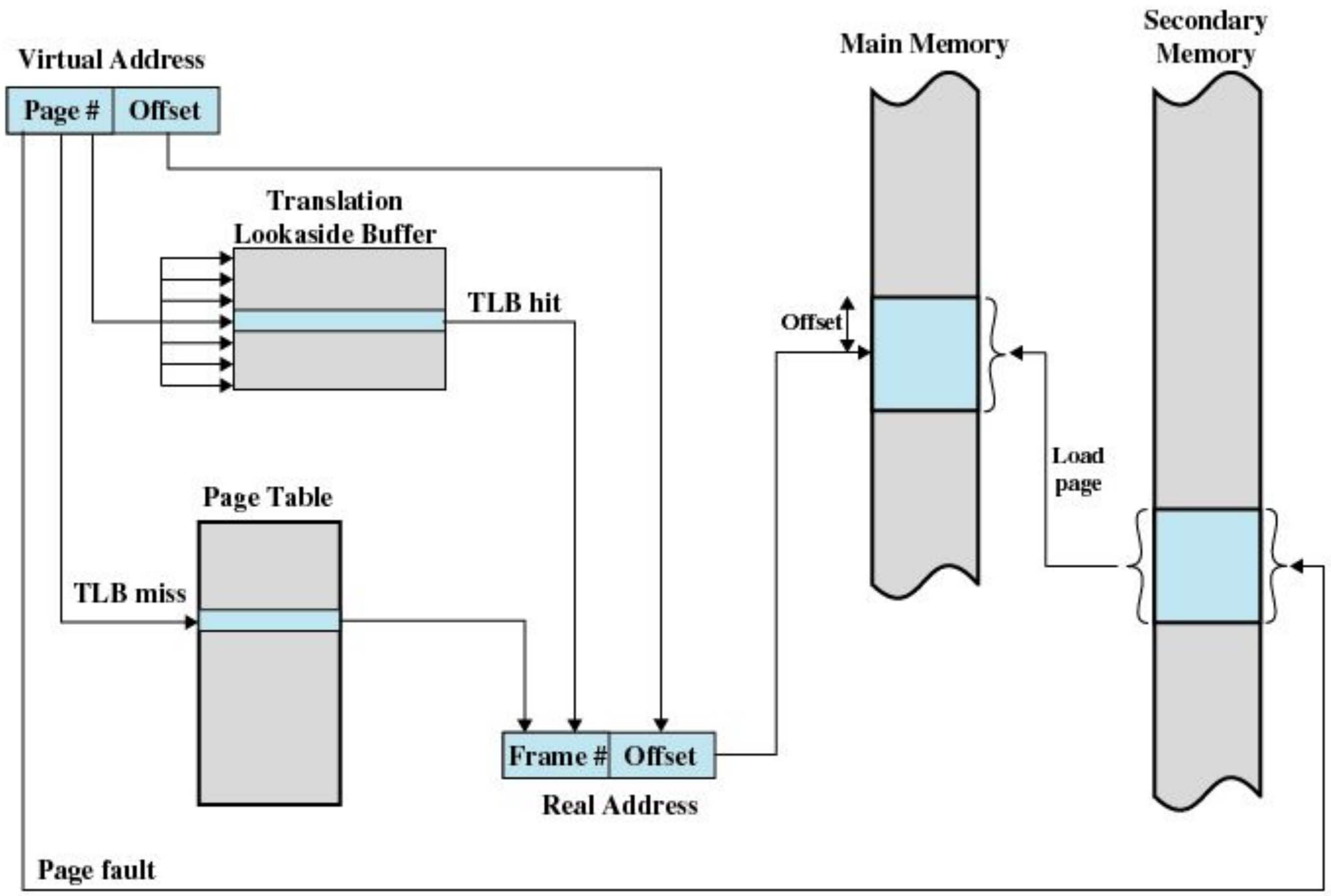


Figure 8.7 Use of a Translation Lookaside Buffer

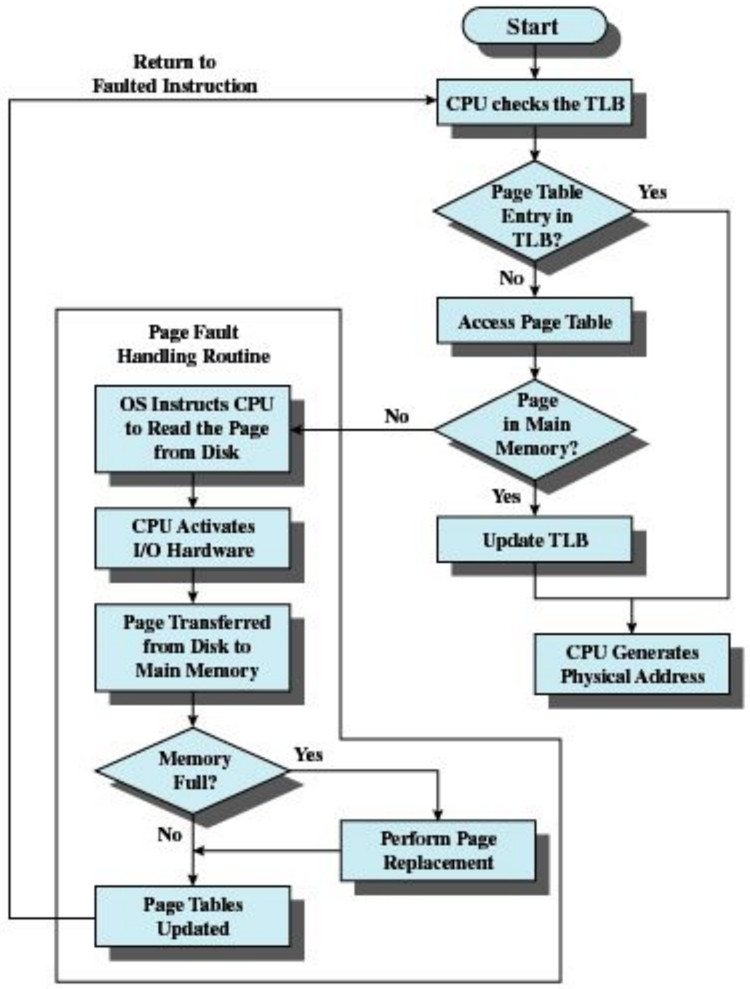


Figure 8.8 Operation of Paging and Translation Lookaside Buffer (TLB) [FURH87]

29

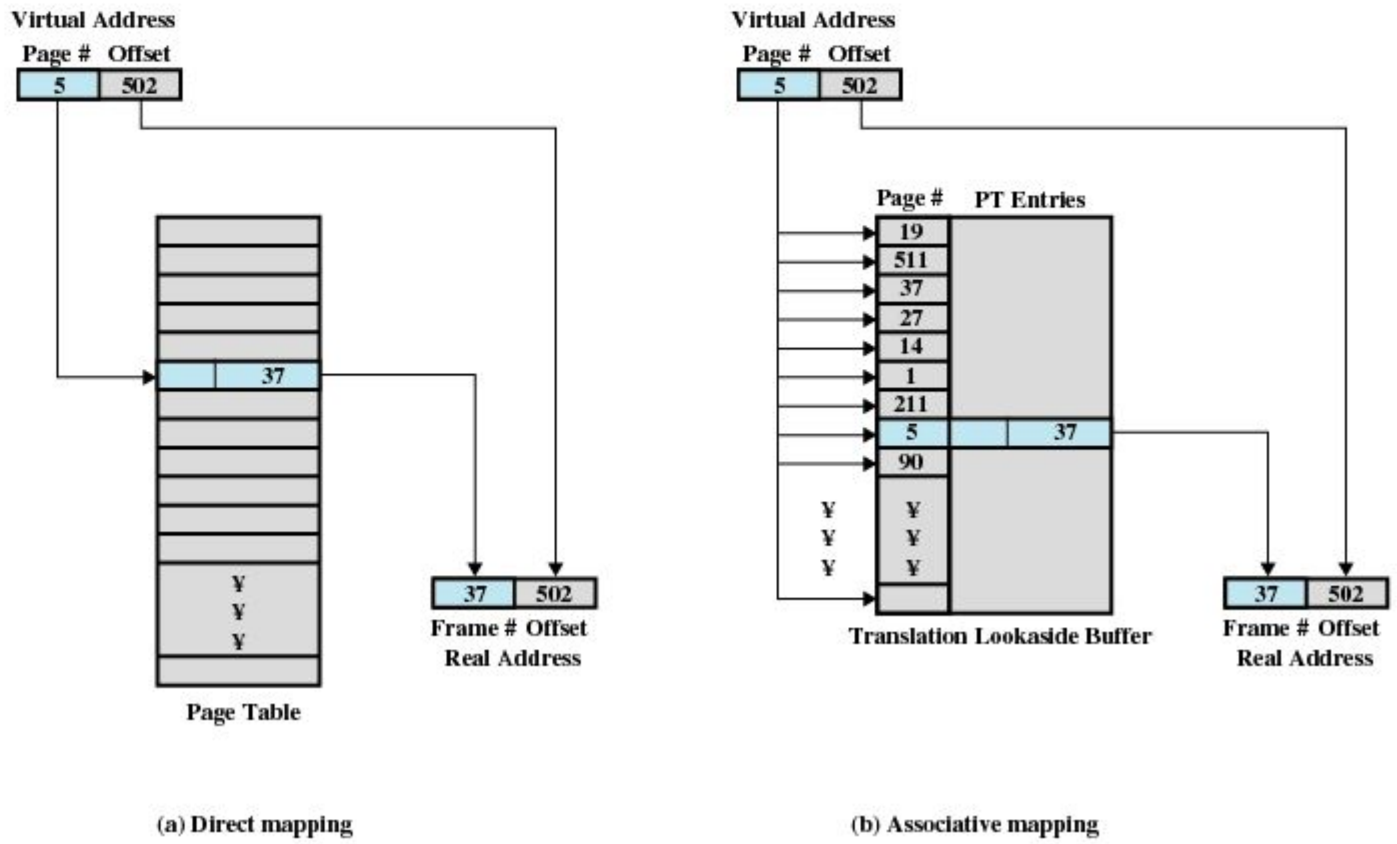


Figure 8.9 Direct Versus Associative Lookup for Page Table Entries

30

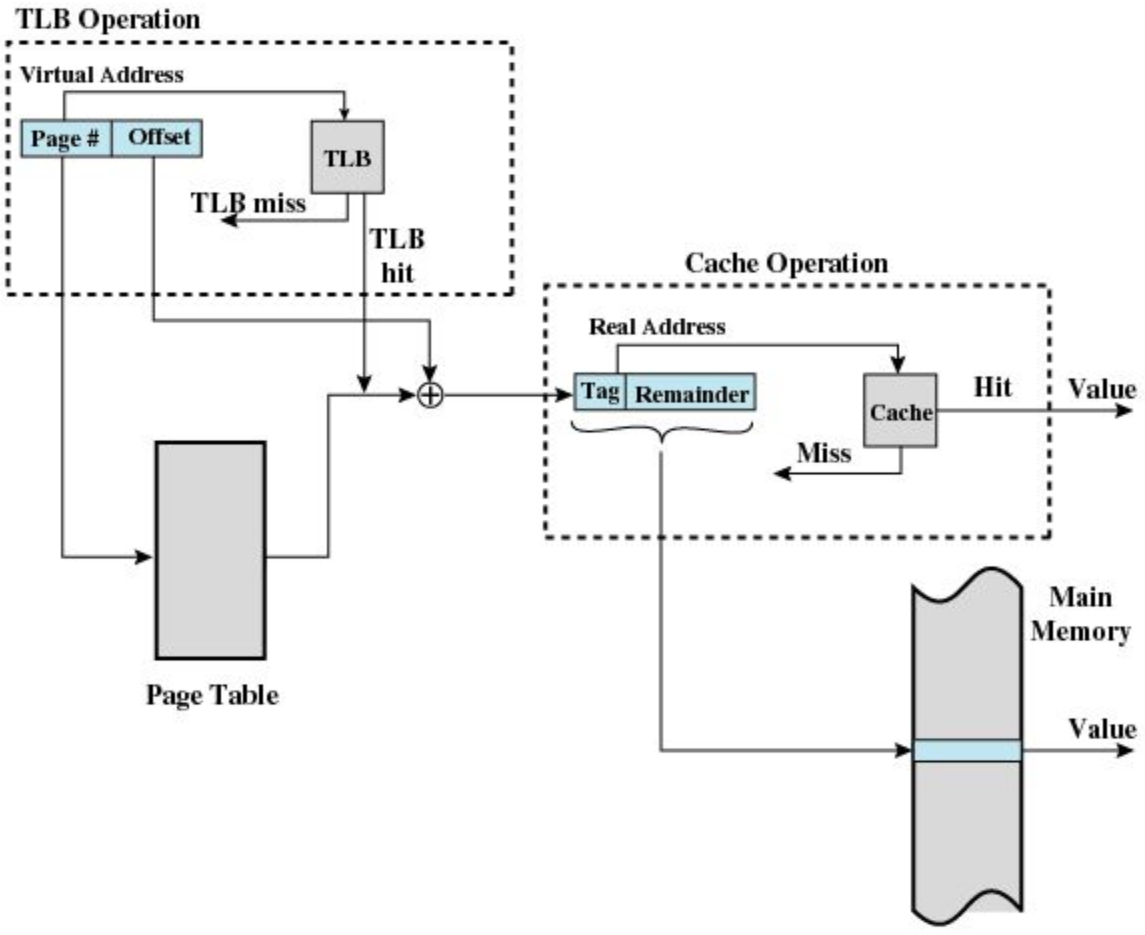


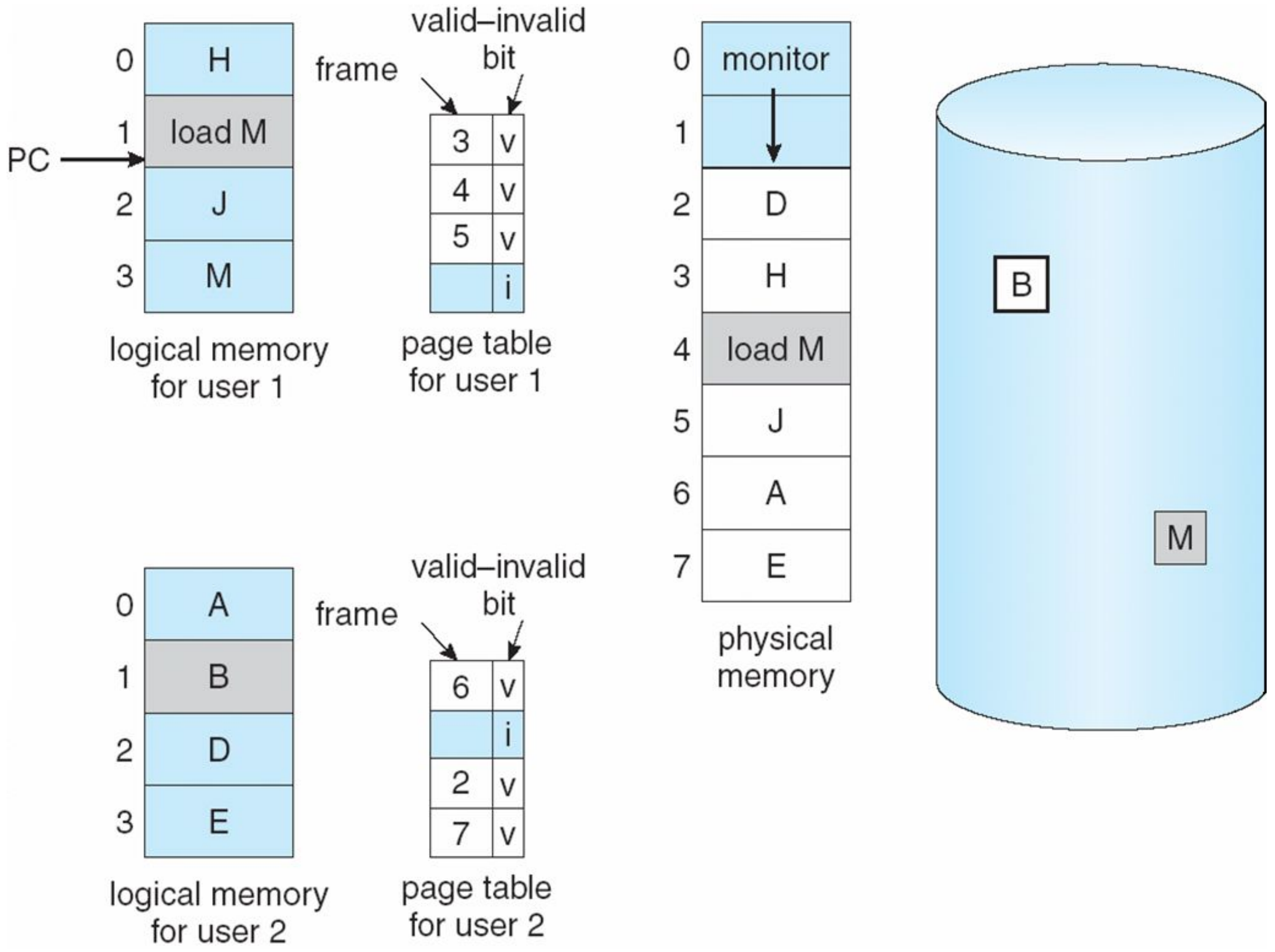
Figure 8.10 Translation Lookaside Buffer and Cache Operation

31

Page Replacement

- Prevent over-allocation of memory by modifying page-fault service routine to include page replacement
- 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

Need For Page Replacement



Basic Page Replacement

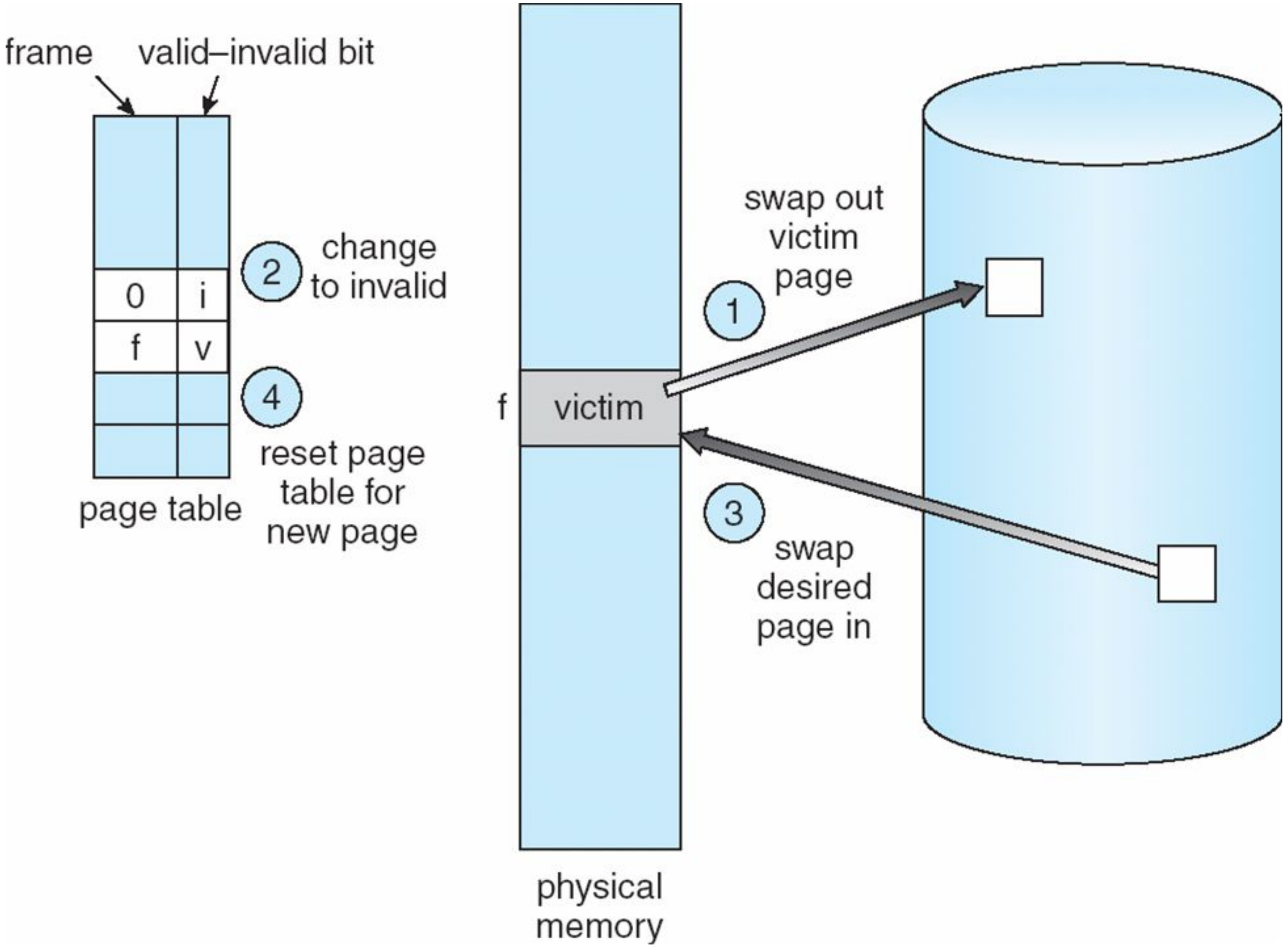
1. Find the location of the desired page on disk

2. Find a free frame:
   - If there is a free frame, use it
   - If there is no free frame, use a page replacement algorithm to select a **victim** frame

3. Bring the desired page into the (newly) free frame; update the page and frame tables

4. Restart the process

Page Replacement

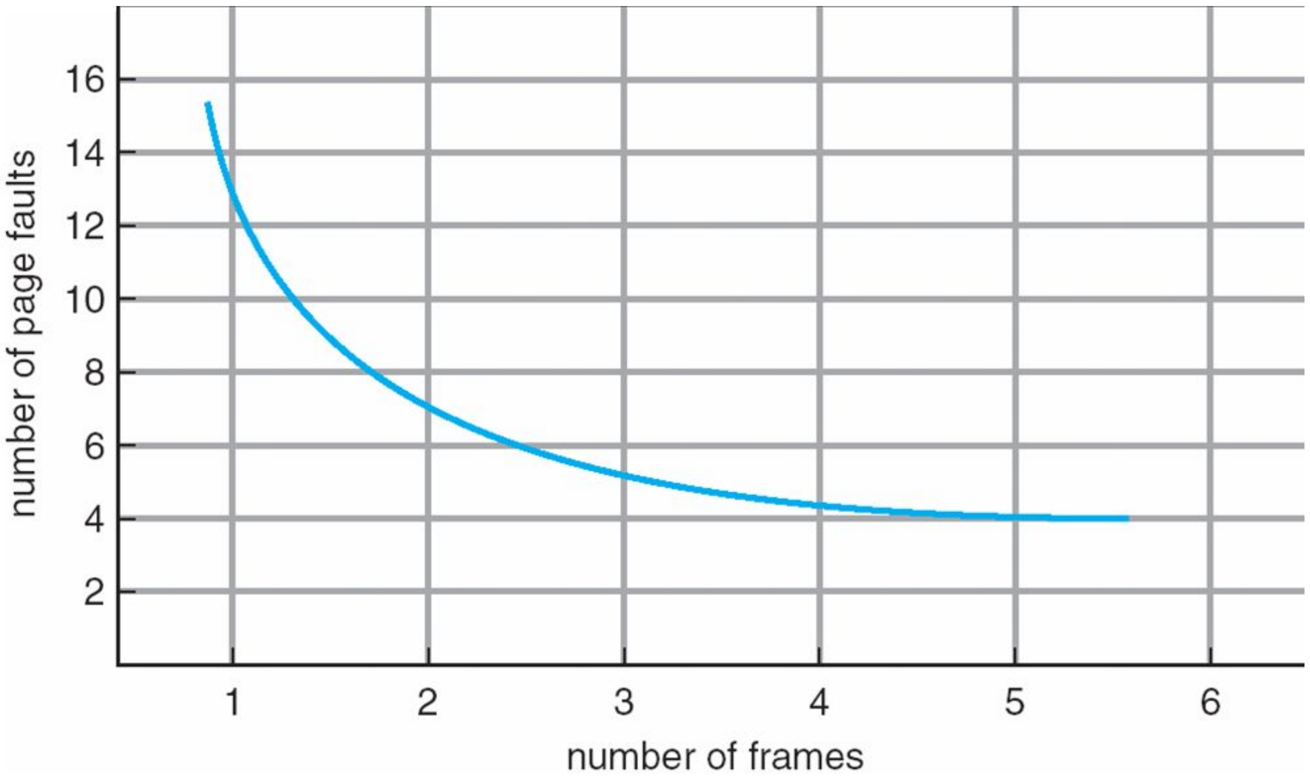


Page Replacement Algorithms

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

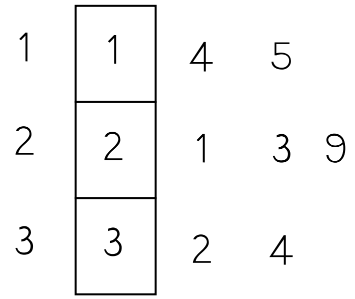
  1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5

Graph of Page Faults Versus The Number of Frames



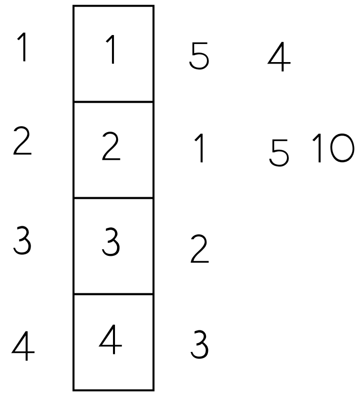
First-In-First-Out (FIFO) Algorithm

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



9 page faults

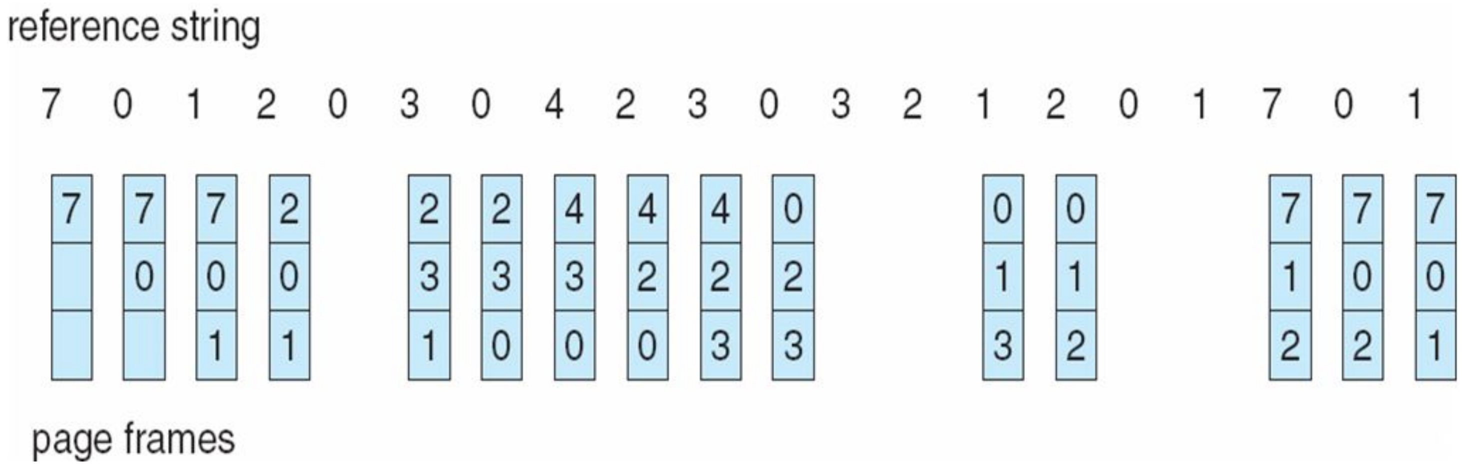
- 4 frames



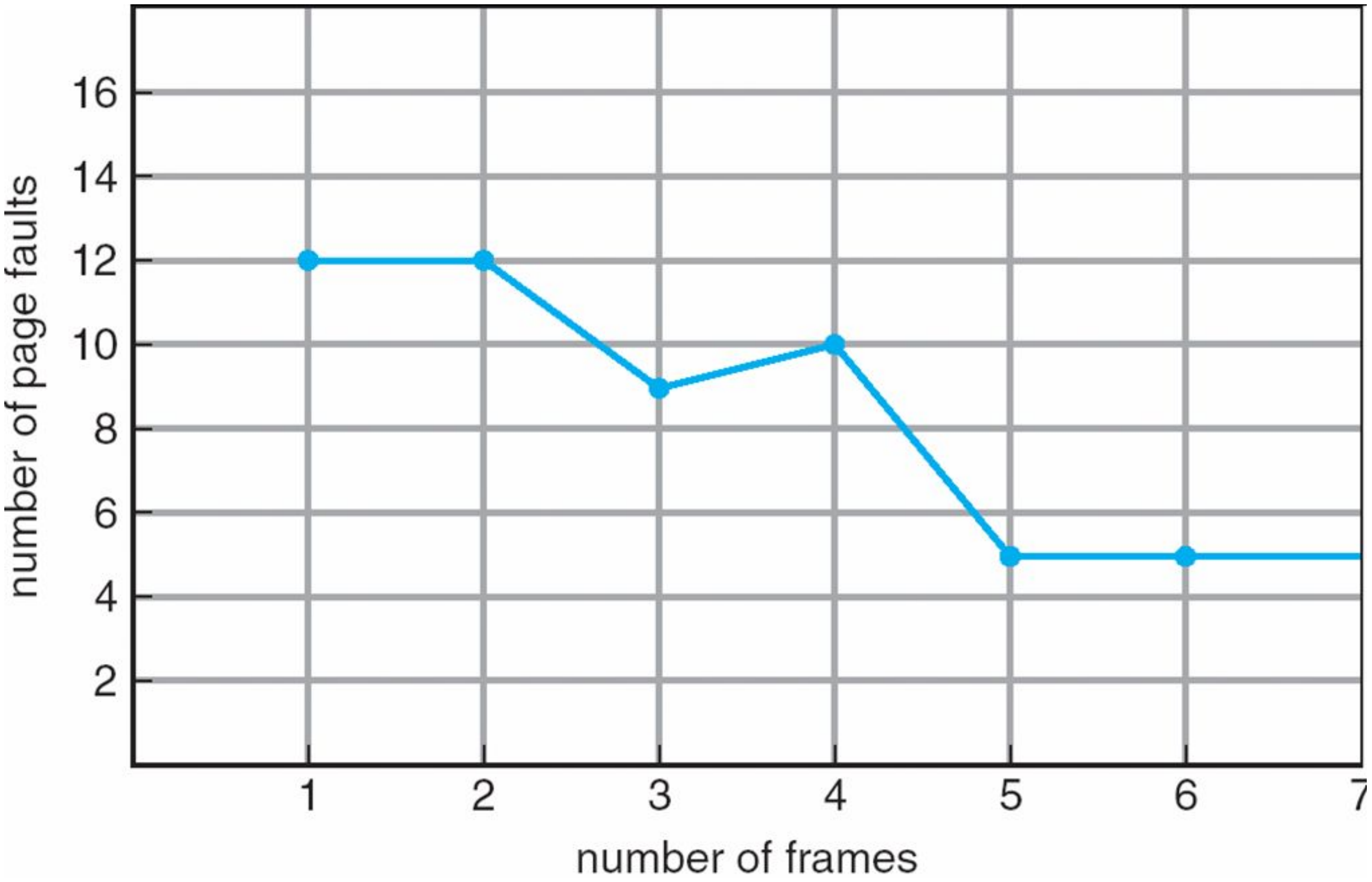
10 page faults

- Belady's Anomaly: more frames ⇒ more page faults

FIFO Page Replacement



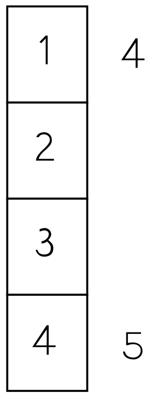
FIFO Illustrating Belady's Anomaly



Optimal Algorithm

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

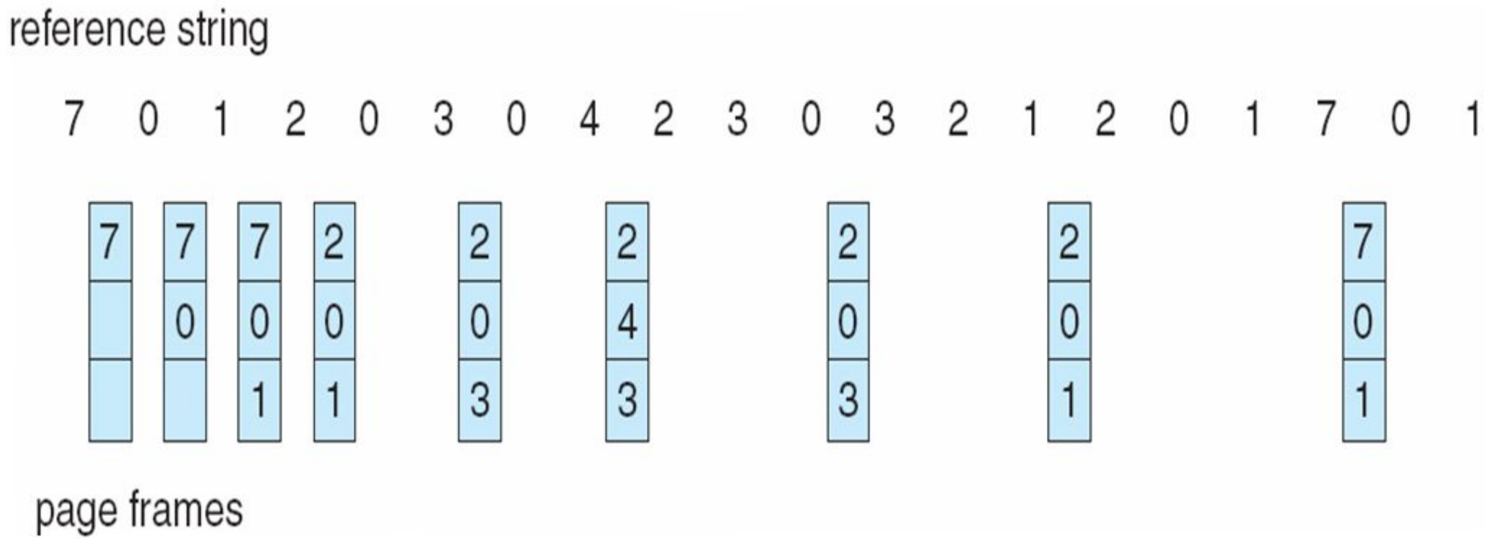
1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5



6 page faults

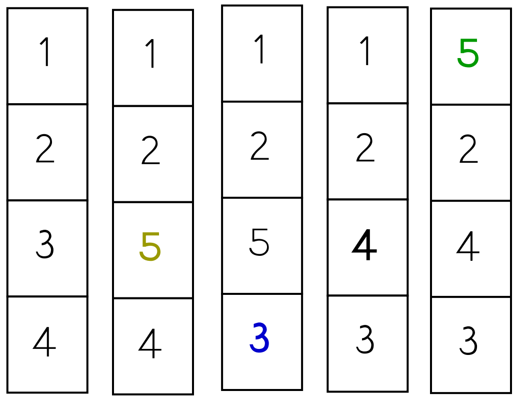
- How do you know this?
- Used for measuring how well your algorithm performs

Optimal Page Replacement



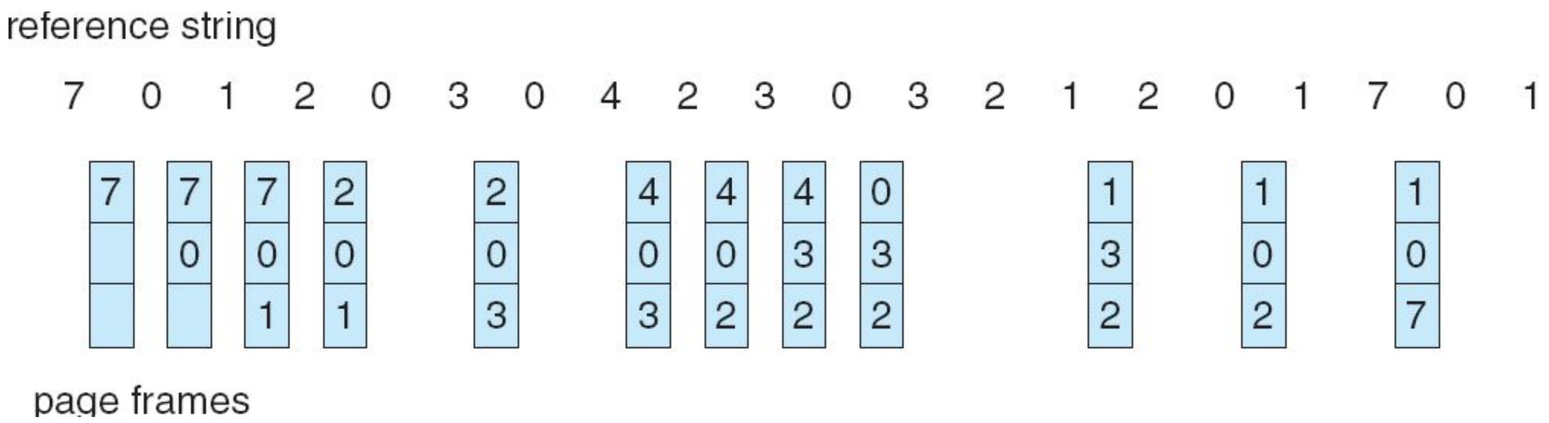
Least Recently Used (LRU) Algorithm

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



- 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

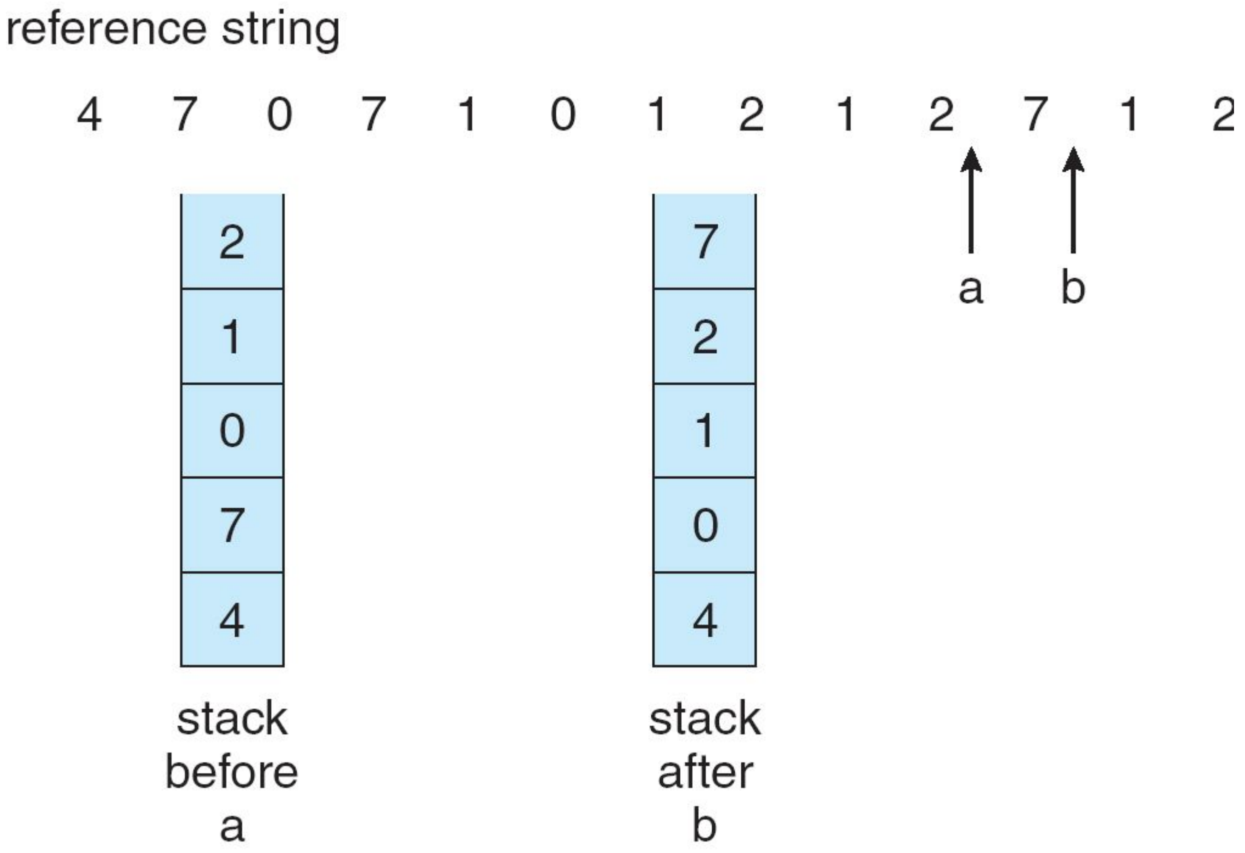
LRU Page Replacement



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
  - 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, however
- 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

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

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 size of process p_i
  - $S \quad s_i$
  - m total number of frames
  - a_i allocation for $p_i \quad \frac{s_i}{S} \quad m$

$$m \quad 64$$
$$s_i \quad 10$$
$$s_2 \quad 127$$
$$a_1 \quad \frac{10}{137} \quad 64 \quad 5$$
$$a_2 \quad \frac{127}{137} \quad 64 \quad 59$$

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

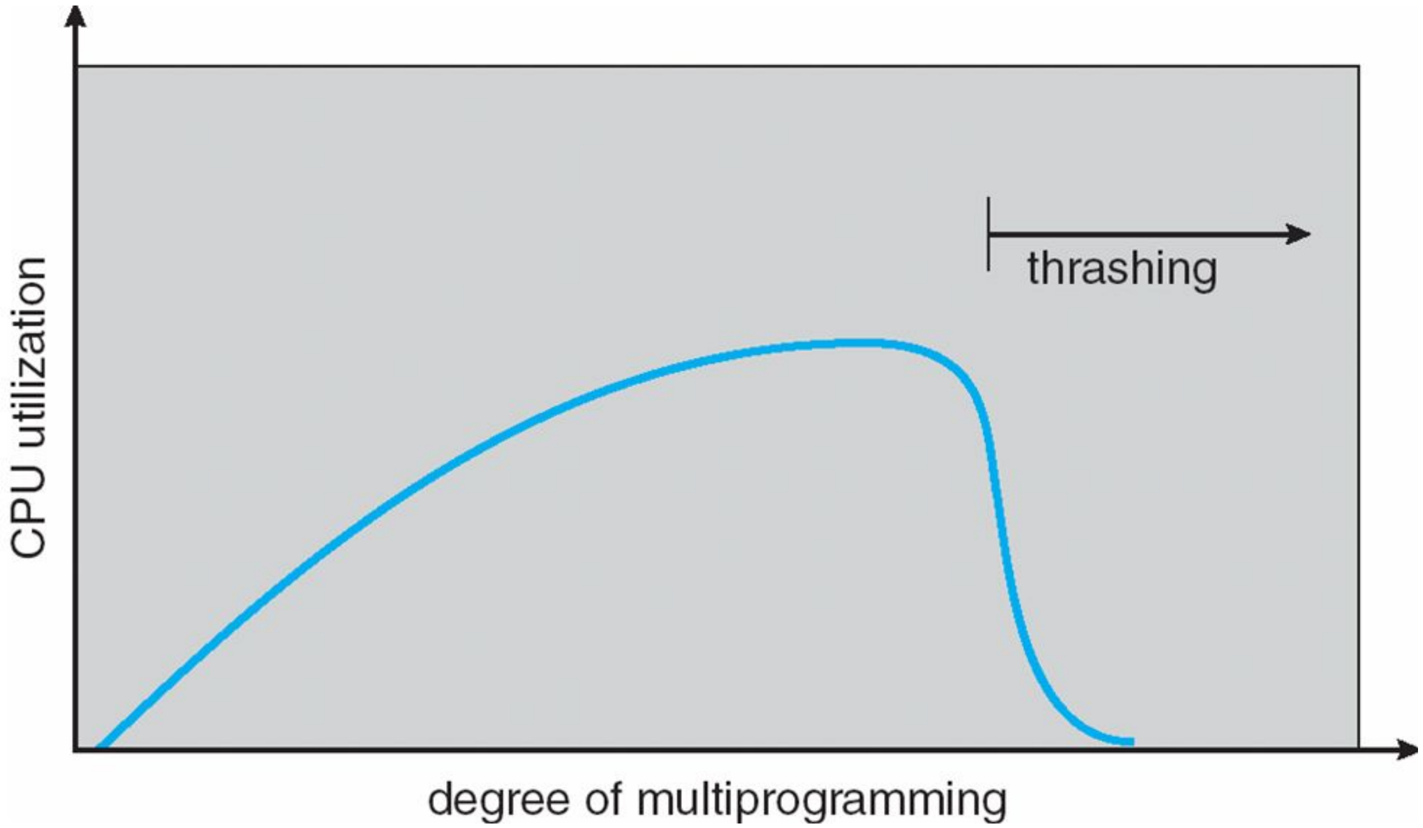
- **Global replacement** – process selects a replacement frame from the set of all frames; one process can take a frame from another
- **Local replacement** – each process selects from only its own set of allocated frames

Thrashing

- If a process does not have "enough" 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

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

Thrashing (Cont.)


thrashing
CPU utilization
degree of multiprogramming