

Unit 4: Memory Management (Galvin – chap 8,9)

Contents

- Contiguous and non-contiguous memory,
- Swapping
- Paging, Segmentation
- Virtual Memory, demand Paging
- Page replacement algorithms- FIFO, LRU,
Optimal
- Allocation of frames and Trashing

Background

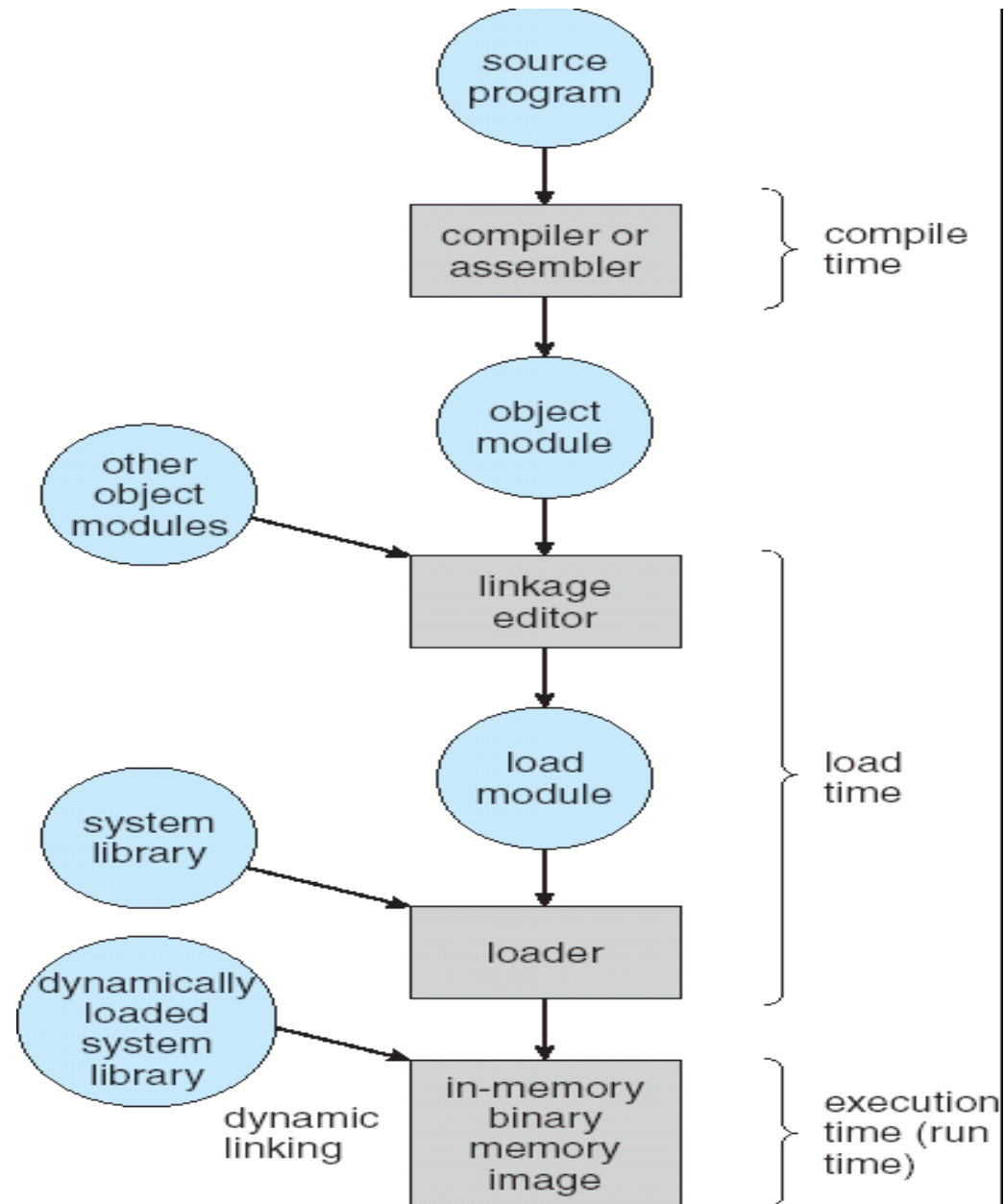
- Program must be brought into memory and placed within a process for it to be run
- **Input queue or job queue** – collection of processes on the disk that are waiting to be brought into memory to run the program"
- User programs go through several steps before being run.

Binding of Instructions and Data to Memory

Address binding of instructions and data to memory addresses can happen at three different stages

- **Compile time:** If memory location known a priori, absolute code can be generated; must recompile code if starting location changes
- **Load time:** Must generate relocatable code if memory location is not known at compile time.
- **Execution time:** Binding delayed until run time if the process can be moved during its execution from one memory segment to another. Need hardware support for address maps (e.g., base and limit registers).

Multistep Processing of a User Program



Logical vs. Physical Address Space

The concept of a logical address space that is bound to a separate physical address space is central to proper memory management"

- **Logical address** – generated by the CPU; also referred to as virtual address

- **Physical address** – address seen by the memory unit

Logical and physical addresses are the same in compile-time and

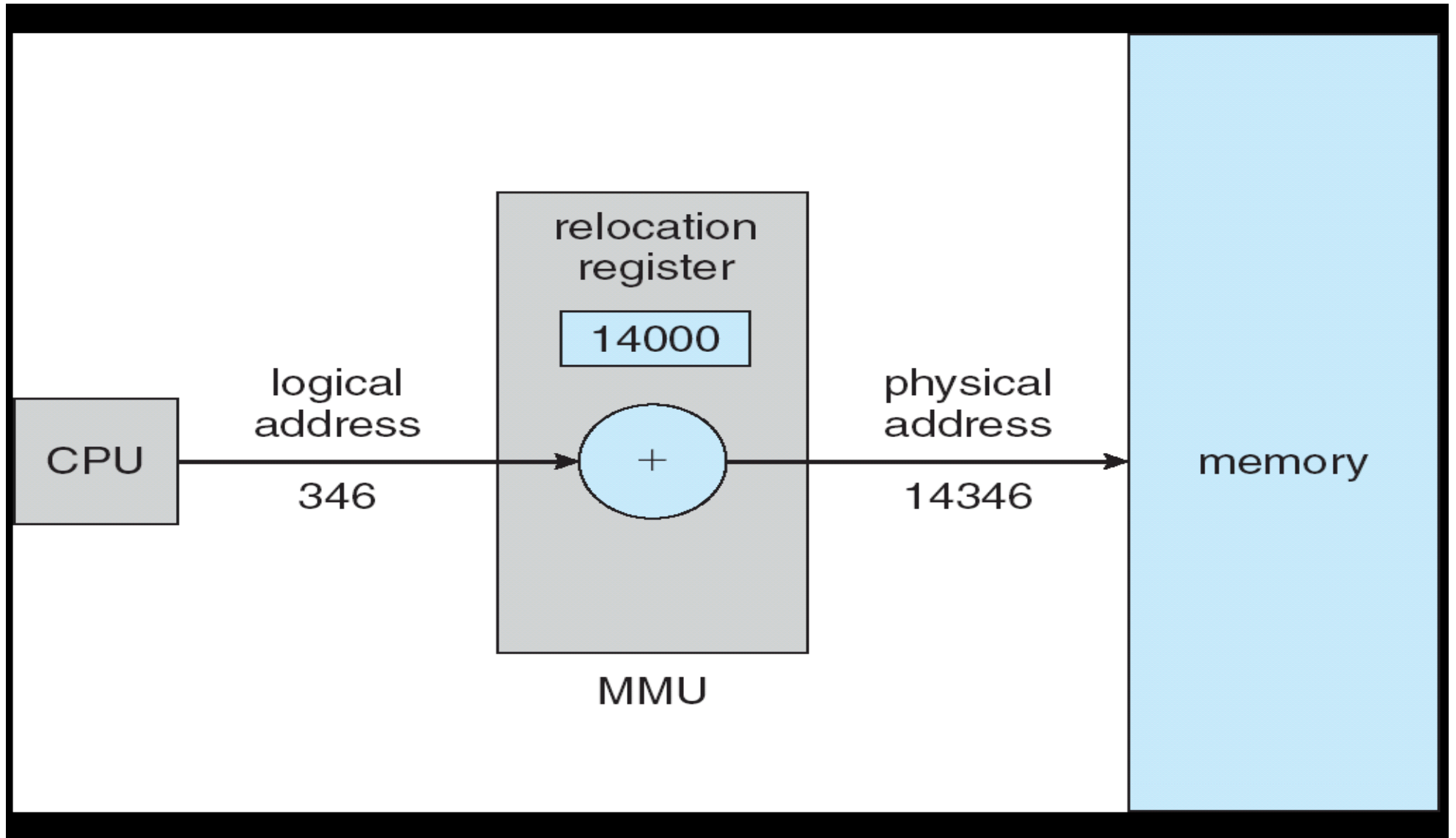
load-time address-binding schemes; logical (virtual) and physical

addresses differ in execution-time address-binding scheme

Memory-Management Unit (MMU)

- Hardware device that maps virtual to physical address
- In MMU scheme, the value in the relocation register is added to every address generated by a user process at the time it is sent to memory
- The user program deals with logical addresses; it never sees the real physical addresses

Dynamic relocation using a relocation register



Swapping

A process can be swapped temporarily out of memory to a backing store, and then brought back into memory for continued execution

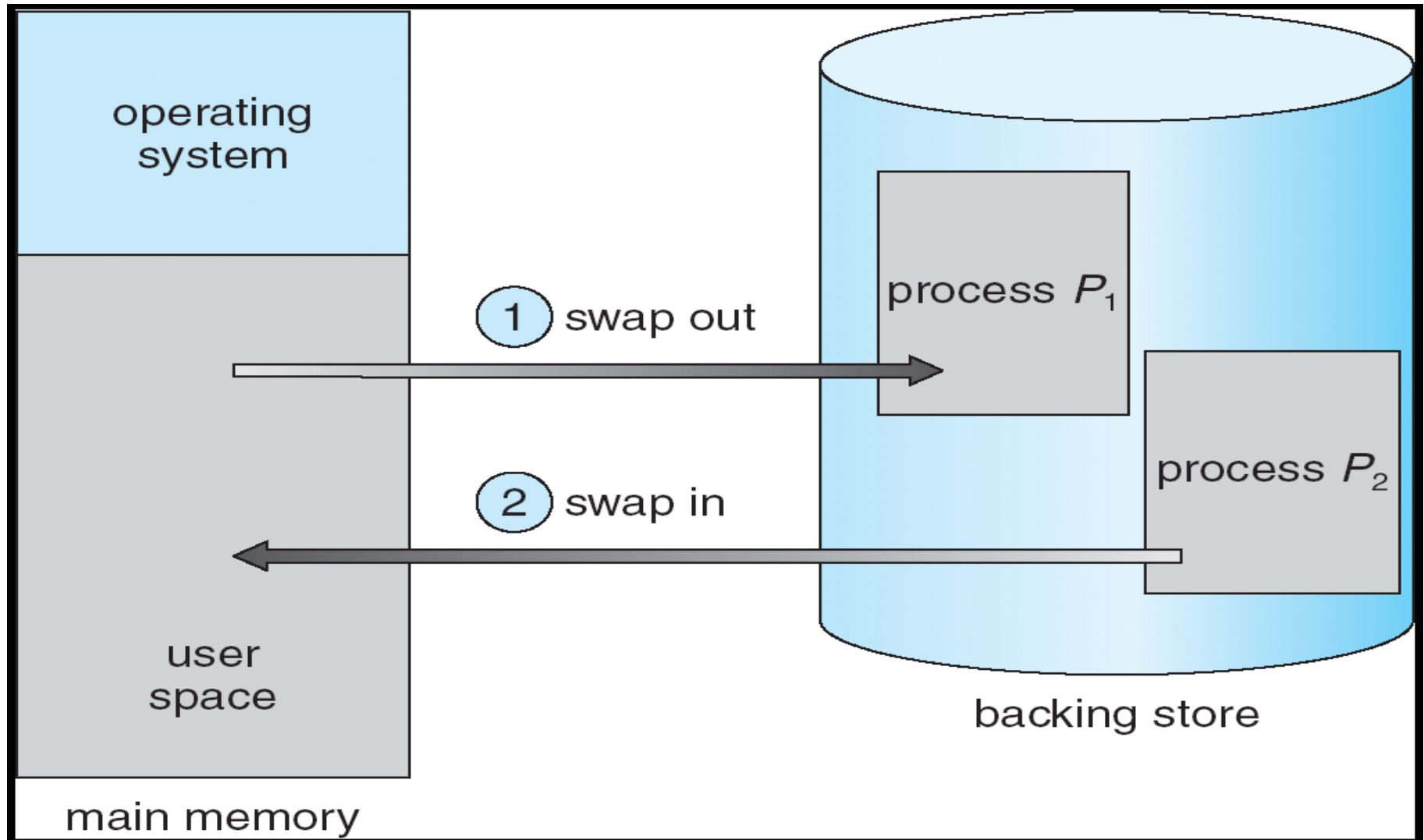
- **Backing store** – fast disk large enough to accommodate copies of all memory images for all users; must provide direct access to these memory images

- **Roll out (swap-out), Roll in (Swap-in)** – swapping variant used for priority-based scheduling algorithms; lower-priority process is swapped out so higher-priority process can be loaded and executed

- Major part of swap time is transfer time; total transfer time is directly proportional to the amount of memory swapped

- Modified versions of swapping are found on many systems (i.e., UNIX, Linux, and Windows)"

Schematic View of Swapping



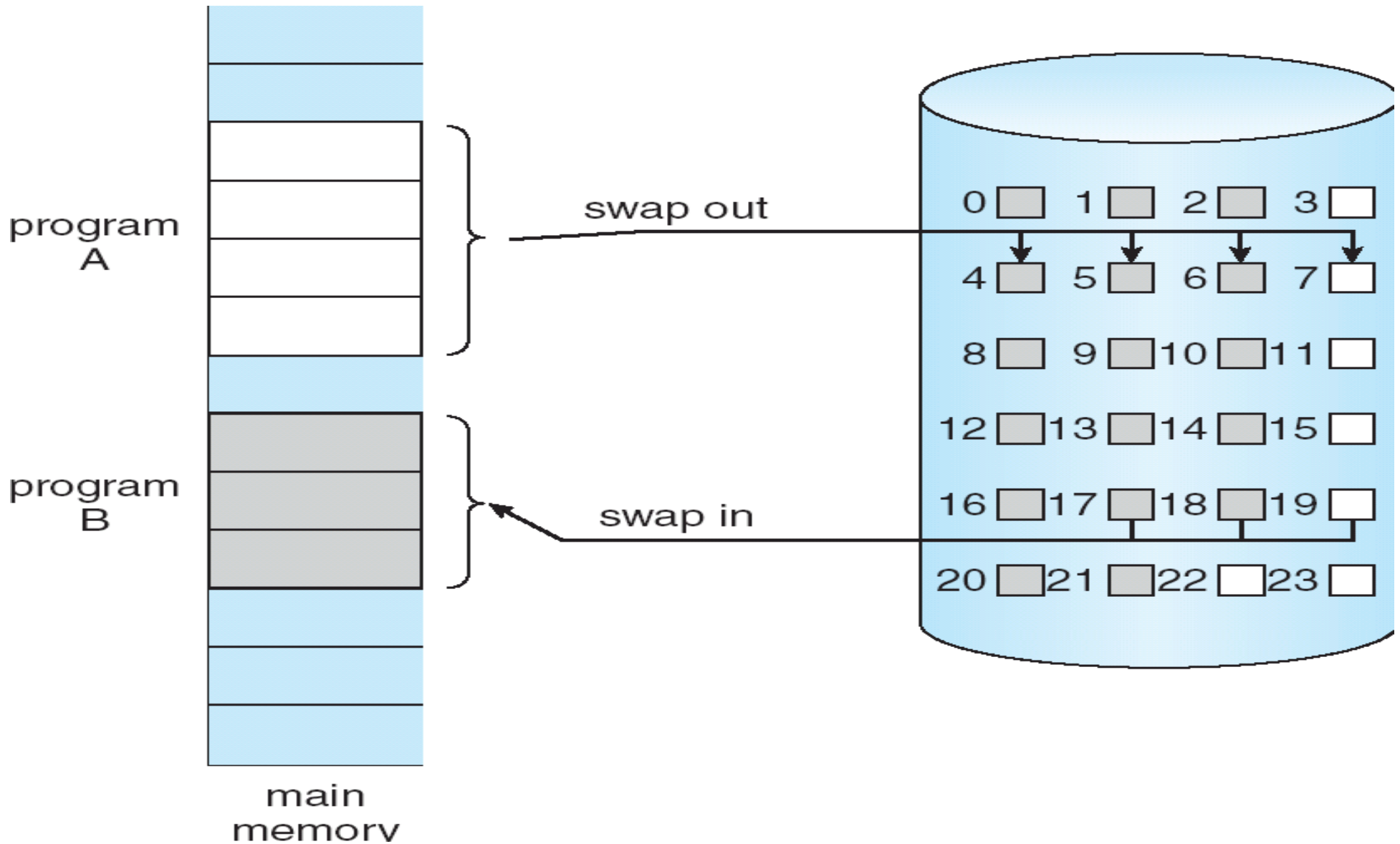
Demand Paging

Bring a page into memory only when it is needed"

1. Less I/O needed
 2. Less memory needed
 3. Faster response
 4. More users
- Page is needed \Rightarrow reference to it
 1. invalid reference \Rightarrow abort
 2. not-in-memory \Rightarrow bring to memory

Lazy swapper – never swaps a page into memory unless page will be needed Swapper that deals with pages is a **pager**

Transfer of a Paged Memory to Contiguous Disk Space



Valid-Invalid Bit

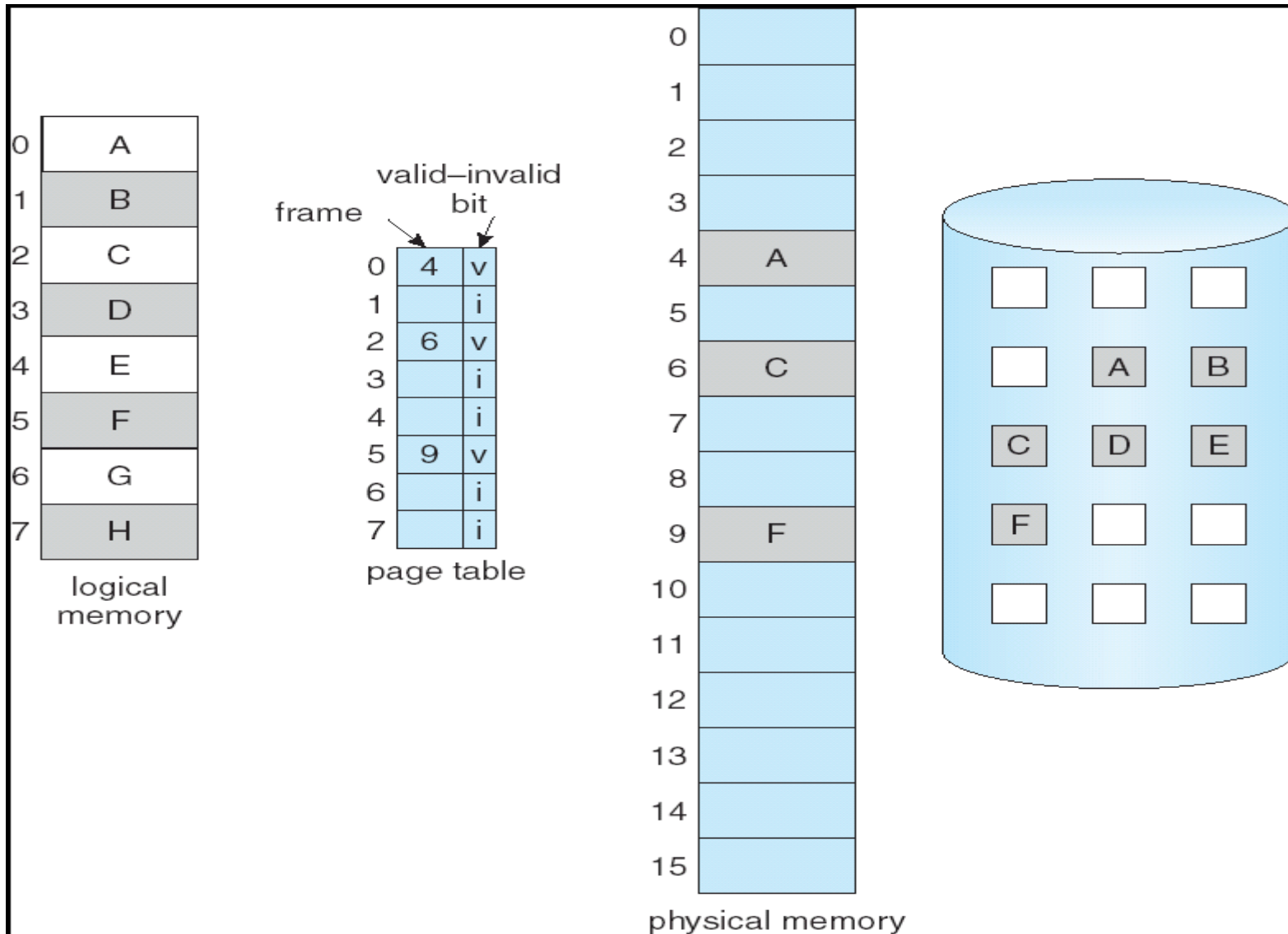
- With each page table entry a valid–invalid bit is associated (1 ⇒ in-memory, 0 ⇒ not-in-memory)"
- Initially valid–invalid but is set to 0 on all entries"
- Example of a page table snapshot:

Frame #	Valid-invalid bit
	1
	1
⋮	
	0
	1

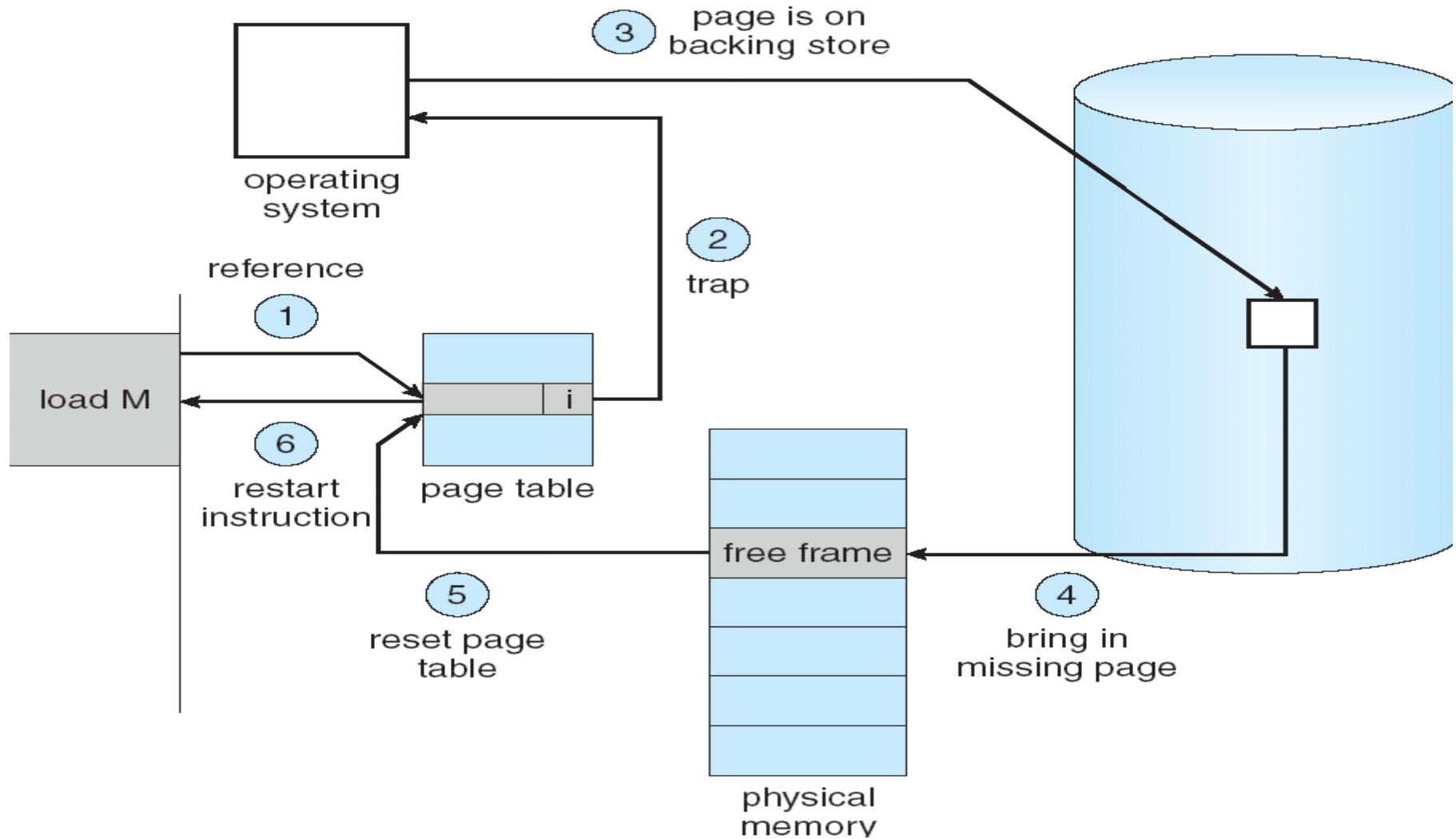
page table

- During address translation, if valid–invalid bit in page table entry is 0 ⇒ page fault
-

Page Table When Some Pages Are Not in Main Memory



Steps in Handling a Page Fault



What happens if there is no free frame?

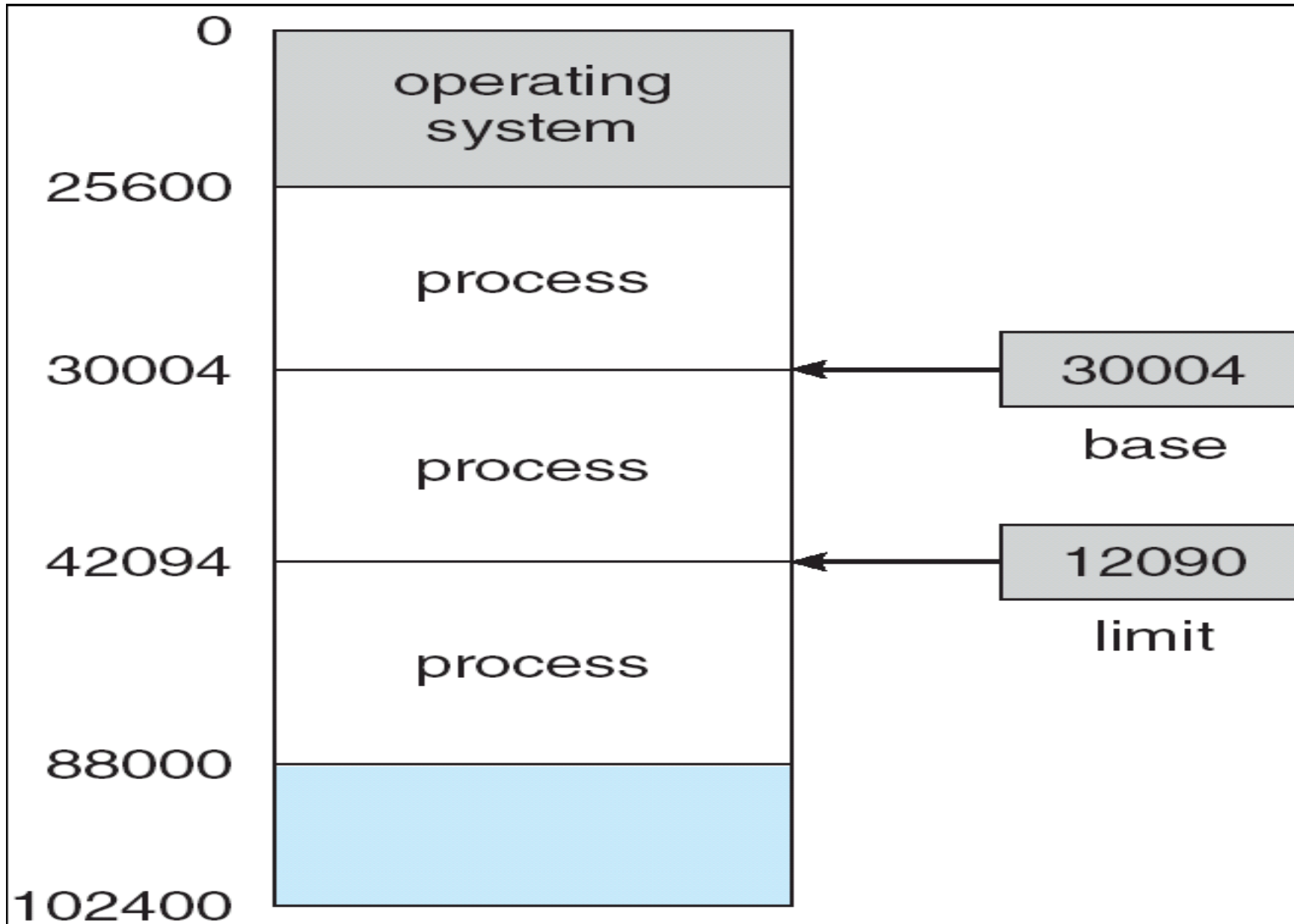
- Page replacement – find some page in memory, but not really in use, swap it out
 - i. **algorithm**
 - ii. **performance** – want an algorithm which will result in minimum number of page faults
- Same page may be brought into memory several times

Memory Allocations

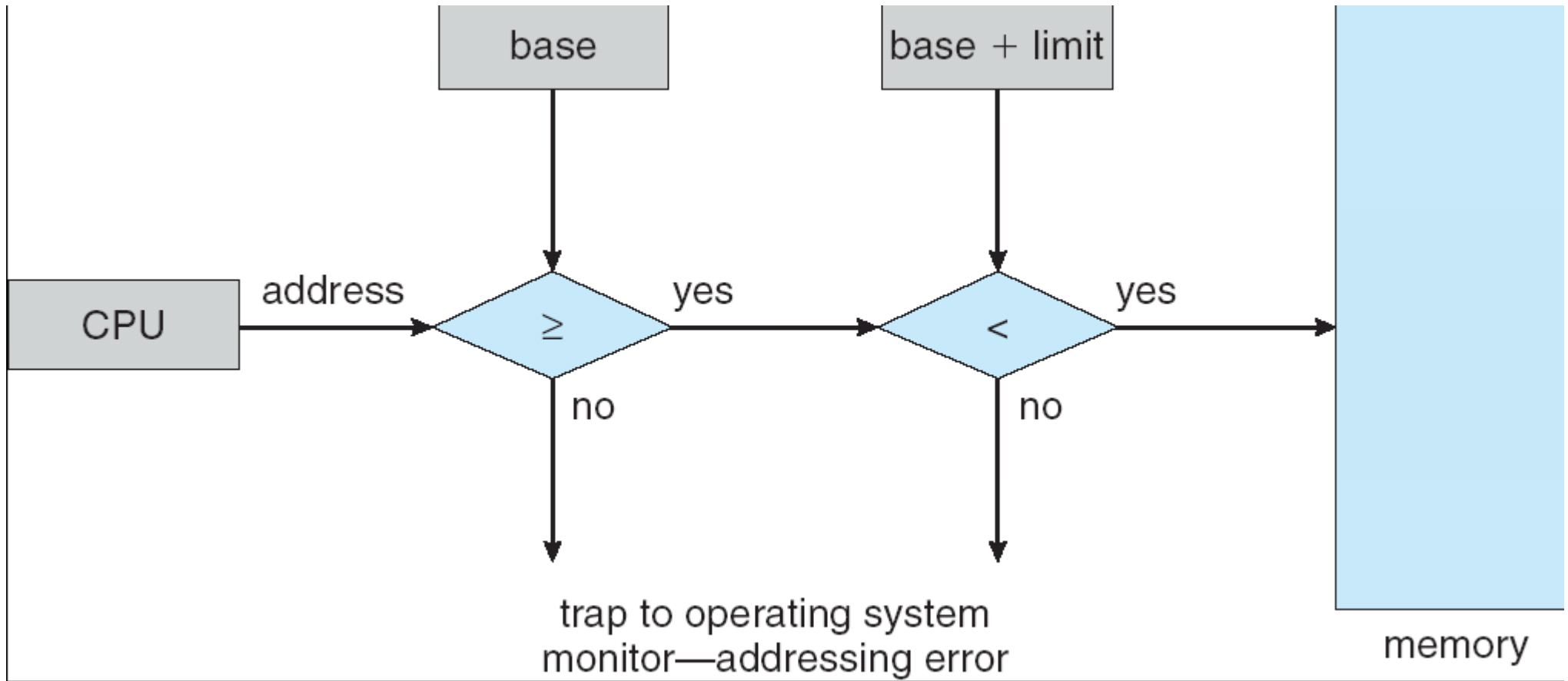
Contiguous Allocation

- Main memory usually into **two partitions**:
 1. **Resident operating system**, usually held in low memory with interrupt vector
 2. User processes then held in high memory
- **Each (Single-partition) allocation**
 - Relocation-register scheme used to protect user processes from each other, and from changing operating-system code and data
 - Relocation register contains value of smallest physical address; limit register contains range of logical addresses – each logical address must be less than the limit register

A base and a limit register define a logical address space



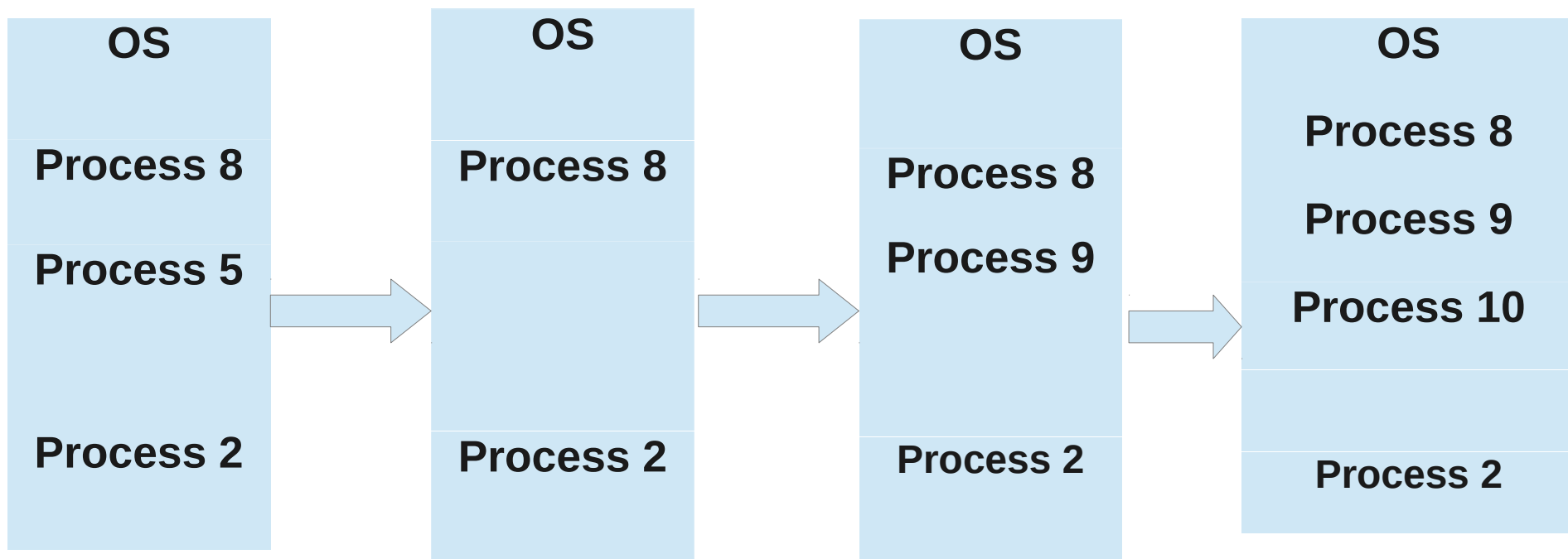
HW address protection with base and limit registers



Contiguous Allocation (Cont.)

Multiple-partition allocation

- Hole – block of available memory; holes of various size are scattered throughout memory
- When a process arrives, it is allocated memory from a hole large enough to accommodate it
- Operating system maintains information about:
 - a) allocated partitions
 - b) free partitions (hole)



Dynamic Storage-Allocation Problem

- How to satisfy a request of size n from a list of free holes
 - First-fit: Allocate the first hole that is big enough
 - Best-fit: Allocate the smallest hole that is big enough; must search entire list, unless ordered by size. Produces the smallest leftover hole.
 - Worst-fit: Allocate the largest hole; must also search entire list. Produces the largest leftover hole.
- First-fit and best-fit better than worst-fit in terms of speed and storage utilization

- Example 1:
 - Given free memory partitions of 100K, 500K, 200K, 300K, and 600K (in order), how would each of the
- First-fit, Best-fit, and Worst-fit algorithms place processes of 212K, 417K, 112K, and 426K (in order)?
 - Which algorithm makes the most efficient use of memory?
-

- Example 2:
 - Given free memory partitions of 100K, 500K, 200K, 300K, and 600K (in order), how would each of the
- First-fit, Best-fit, and Worst-fit algorithms place processes of 317K, 226K, 423K, and 116K (in order)?
 - Which algorithm makes the most efficient use of memory?

- **Fragmentation**

- Wastage of memory.

- External Fragmentation – Gaps between allocated contiguous memory - total memory space exists to satisfy a request, but it is not contiguous.

- Internal Fragmentation – allocated memory may be slightly larger than requested memory; this size difference is memory internal to a partition, but not being used

- Reduce external fragmentation by [compaction](#).

- Shuffle memory contents to place all free memory together in one large block

- Compaction is possible only if relocation is dynamic, and is done at execution time

Define Garbage collection

Paging

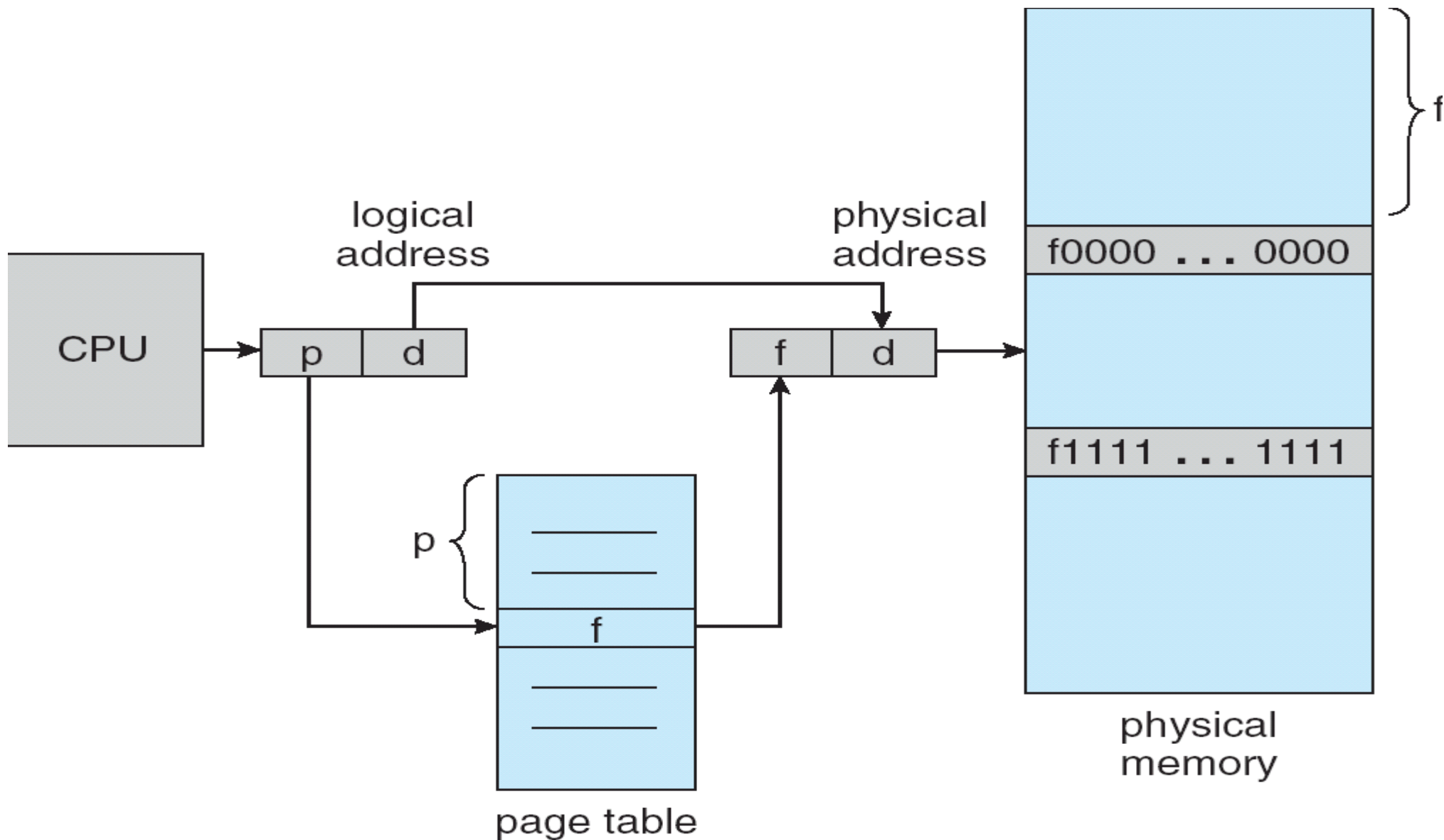
- Logical address space of a process can be noncontiguous; process is allocated physical memory whenever the latter is available
 - Divide physical memory into fixed-sized blocks called frames (size is power of 2, between 512 bytes and 8192 bytes)
 - Divide logical memory into blocks of same size called pages.
 - Keep track of all free frames
 - To run a program of size n pages, need to find n free frames and load program
 - Set up a page table to translate logical to physical addresses
 - Internal fragmentation

- **Address Translation Scheme**

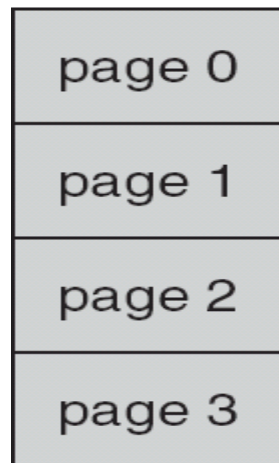
- Address generated by CPU is divided into:

1. **Page number (p)** – used as an index into a page table which contains base address of each page in physical memory
2. **Page offset (d)** – combined with base address to define the physical memory address that is sent to the memory unit

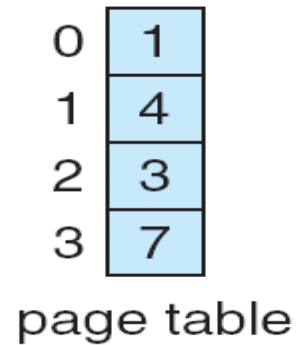
Address Translation Architecture



Paging Example

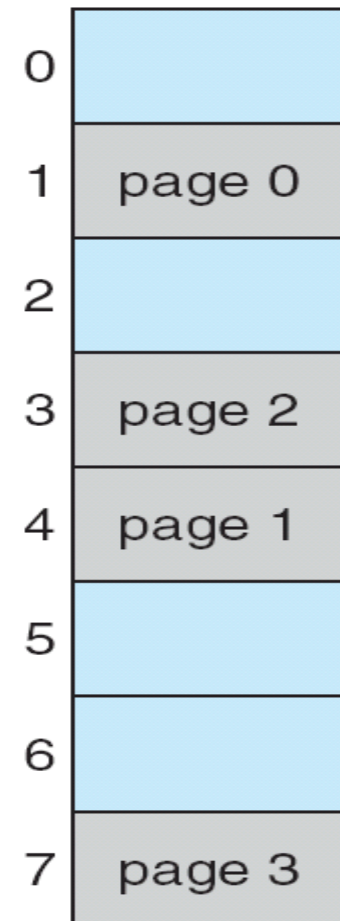


logical
memory



page table

frame
number

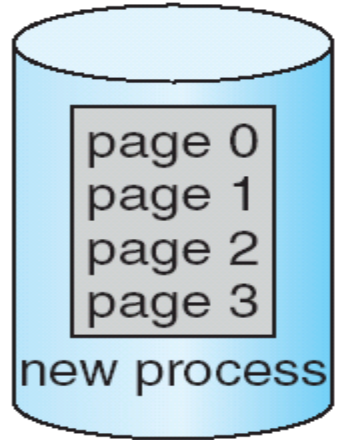


physical
memory

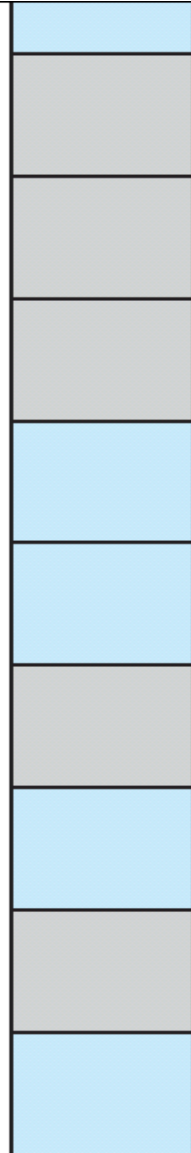
Free Frames

free-frame list

14
13
18
20
15

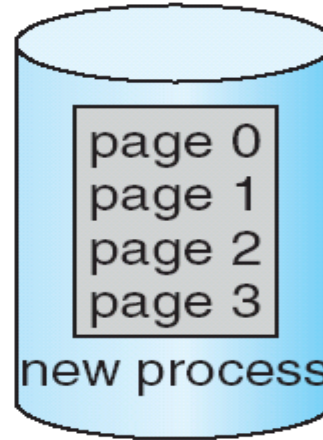


13
14
15
16
17
18
19
20
21



free-frame list

15



0	14
1	13
2	18
3	20

new-process page table

13 page 1
14 page 0
15
16
17
18 page 2
19
20 page 3
21



(a)

(b)

- **Implementation of Page Table**

- Page table is kept in main memory

- **Page-table base register (PTBR)** points to the page table

- **Page-table length register (PRLR)** indicates size of the page

table

- In this scheme every data/instruction access requires two memory accesses. One for the page table and one for the data/instruction.

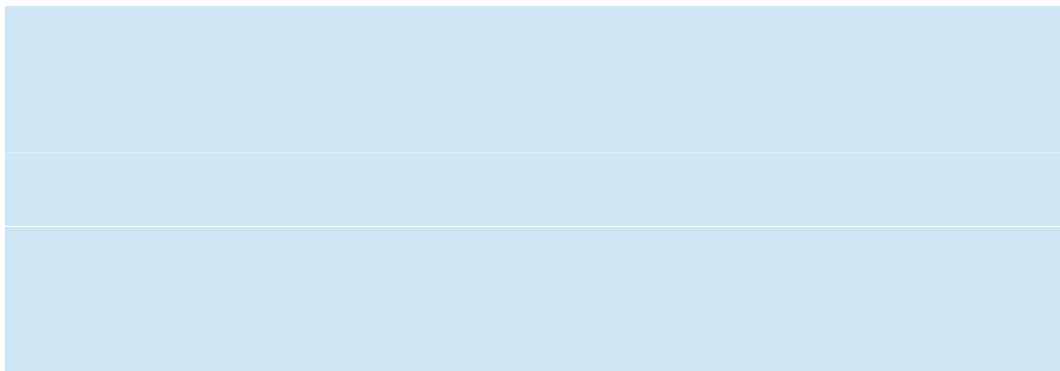
- The two memory access problem can be solved by the use of a special fast-lookup hardware **cache called associative memory or translation look-aside buffers (TLBs)**

Associative Memory

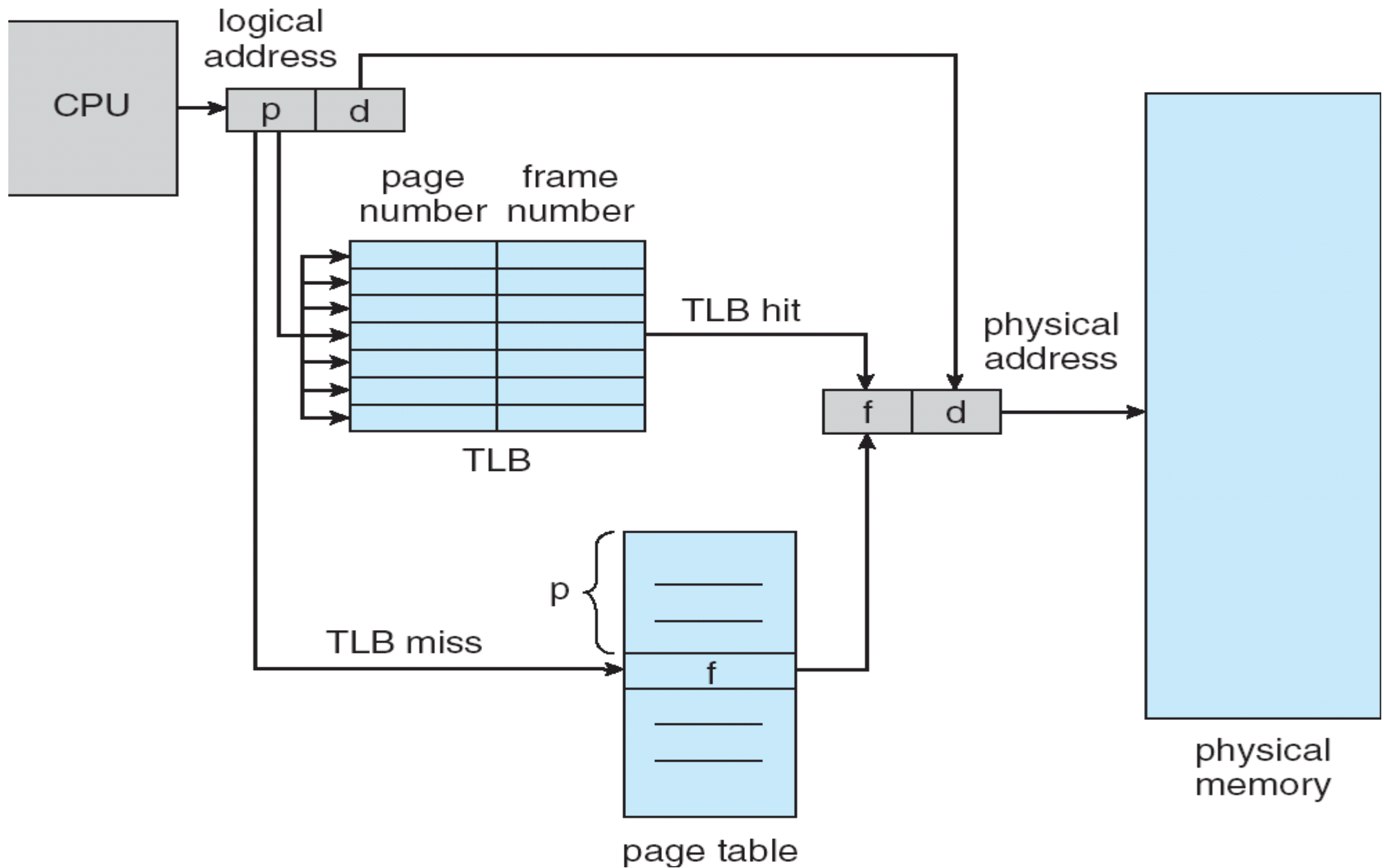
- Associative memory – parallel search

Page #

Frame #



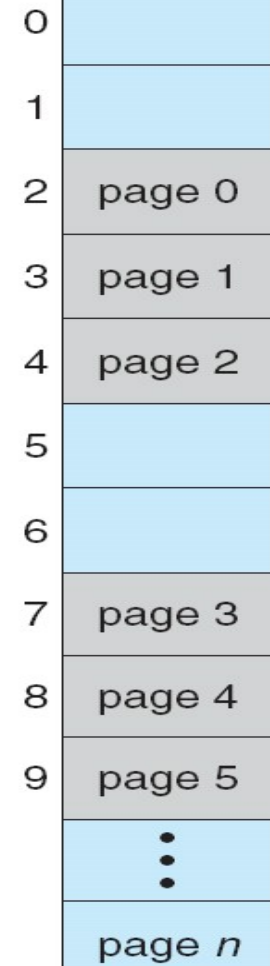
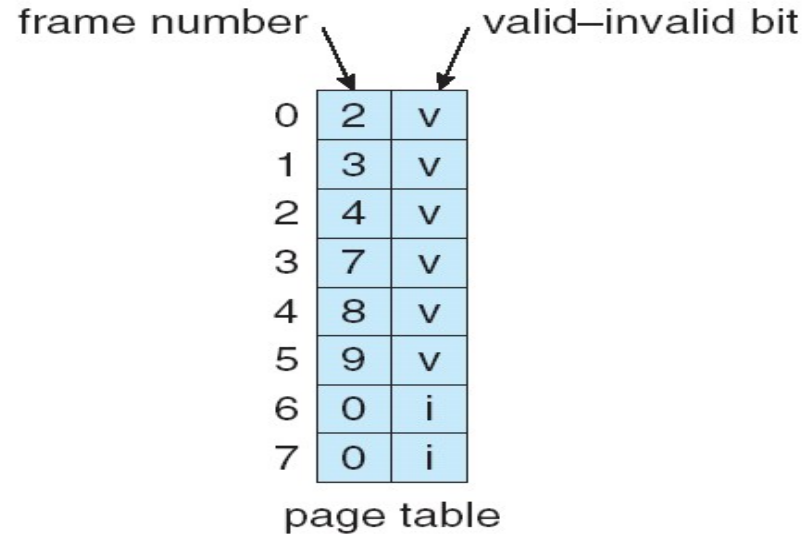
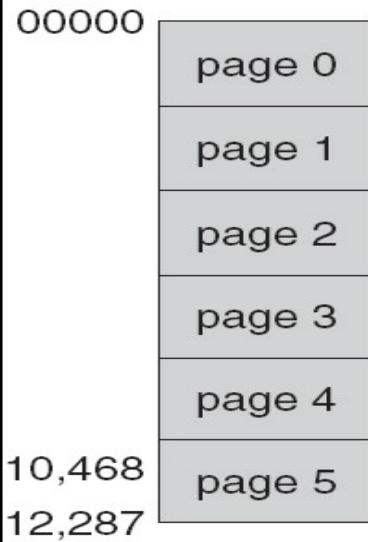
Paging Hardware With TLB



Memory Protection

- Memory protection implemented by associating protection bit with each frame
- Valid-invalid bit attached to each entry in the page table:
 - “valid” indicates that the associated page is in the process’ logical address space, and is thus a legal page
 - “invalid” indicates that the page is not in the process’ logical address space

Valid (v) or Invalid (i) Bit In A Page Table



Virtual Memory

- Elusion to expand memory view onto the secondary storage.
- Moving pages – frames between memory and HD
- Demand paging

Shared Pages

- **Shared code**

- One copy of read-only (reentrant) code shared among processes (i.e., text editors, compilers, window systems).
- Shared code must appear in same location in the logical address space of all processes

- **Private code and data**

- Each process keeps a separate copy of the code and data
- The pages for the private code and data can appear anywhere in the logical address space

Segmentation

- Memory-management scheme that supports user view of memory "
- A program is a collection of segments. A segment is a logical unit such as:

main program,

procedure,

function,

method,

object,

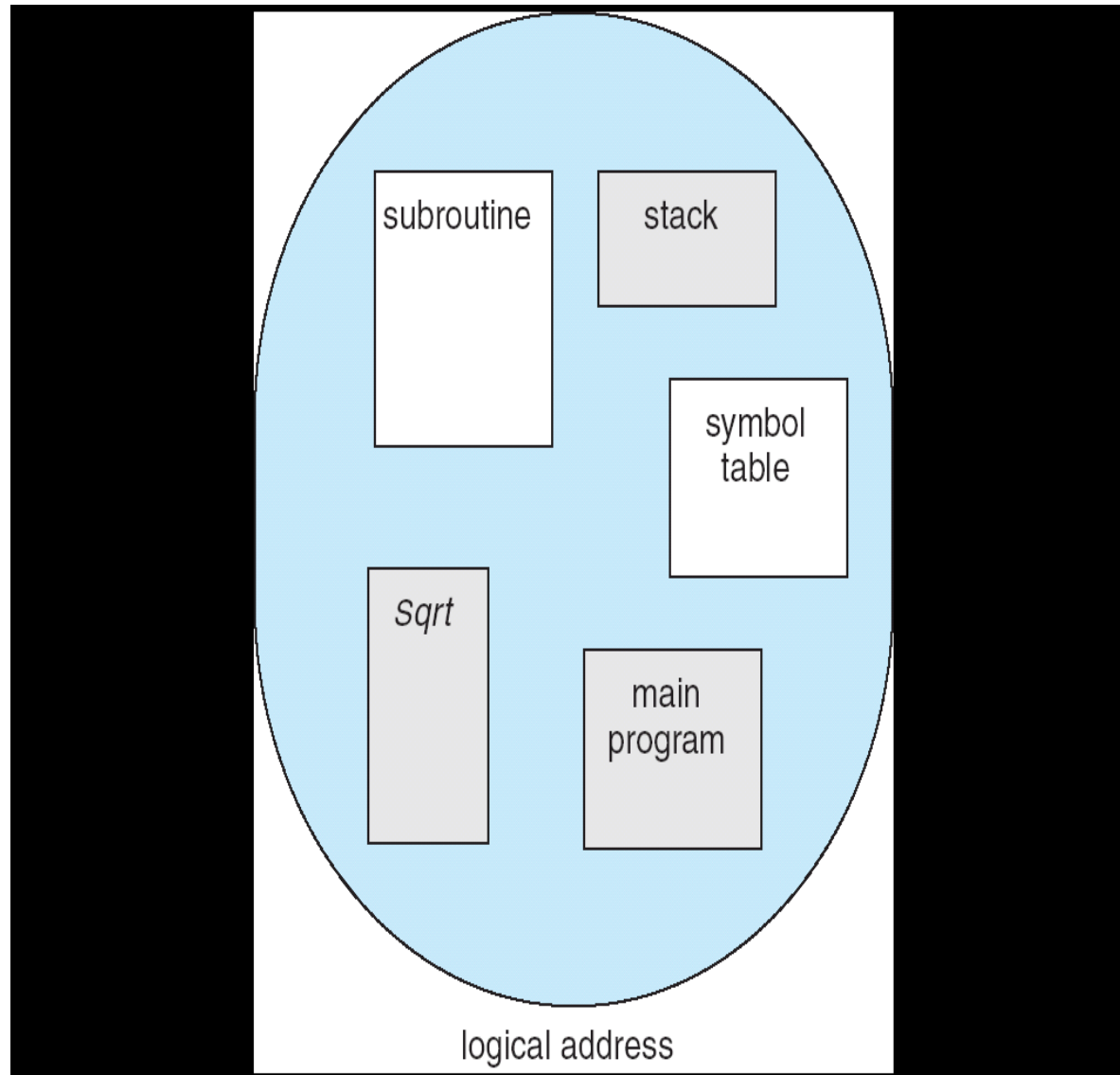
local variables, global variables,

common block,

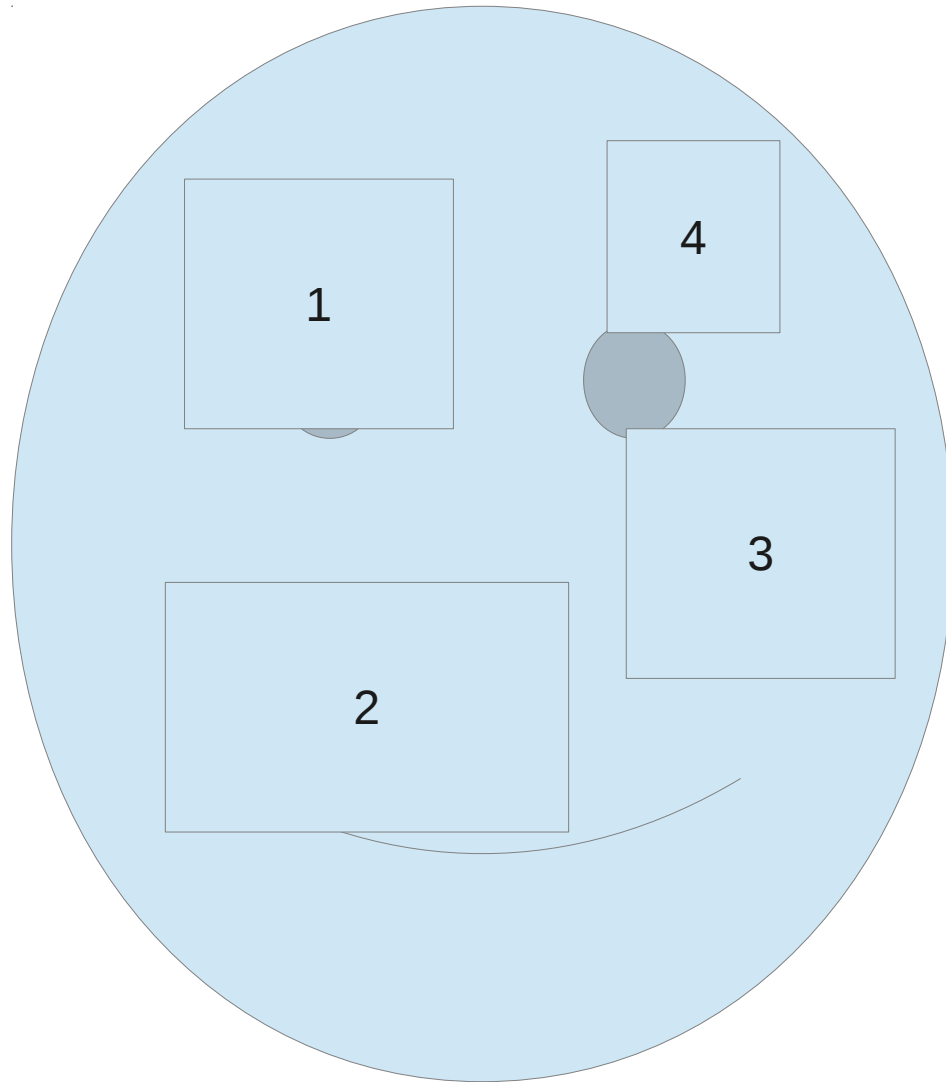
stack,

symbol table, arrays

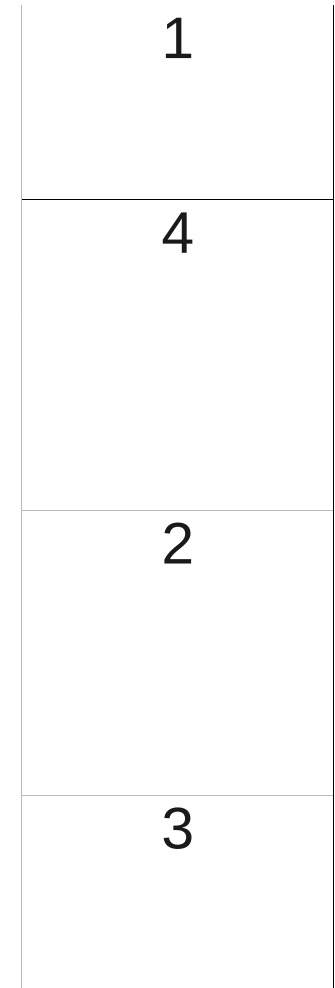
User's View of a Program



Logical View of Segmentation



User space



Physical address space

Segmentation Architecture

- Logical address consists of a two tuple:
 - <segment-number, offset>,
- ■ Segment table – maps two-dimensional physical addresses;
- each table entry has:
 - base – contains the starting physical address where the segments reside in memory
 - limit – specifies the length of the segment
- Segment-table base register (STBR) points to the segment table's location in memory
- Segment-table length register (STLR) indicates number of segments used by a program;

segment number s is legal if $s < \text{STLR}$

Segmentation Architecture (Cont.)

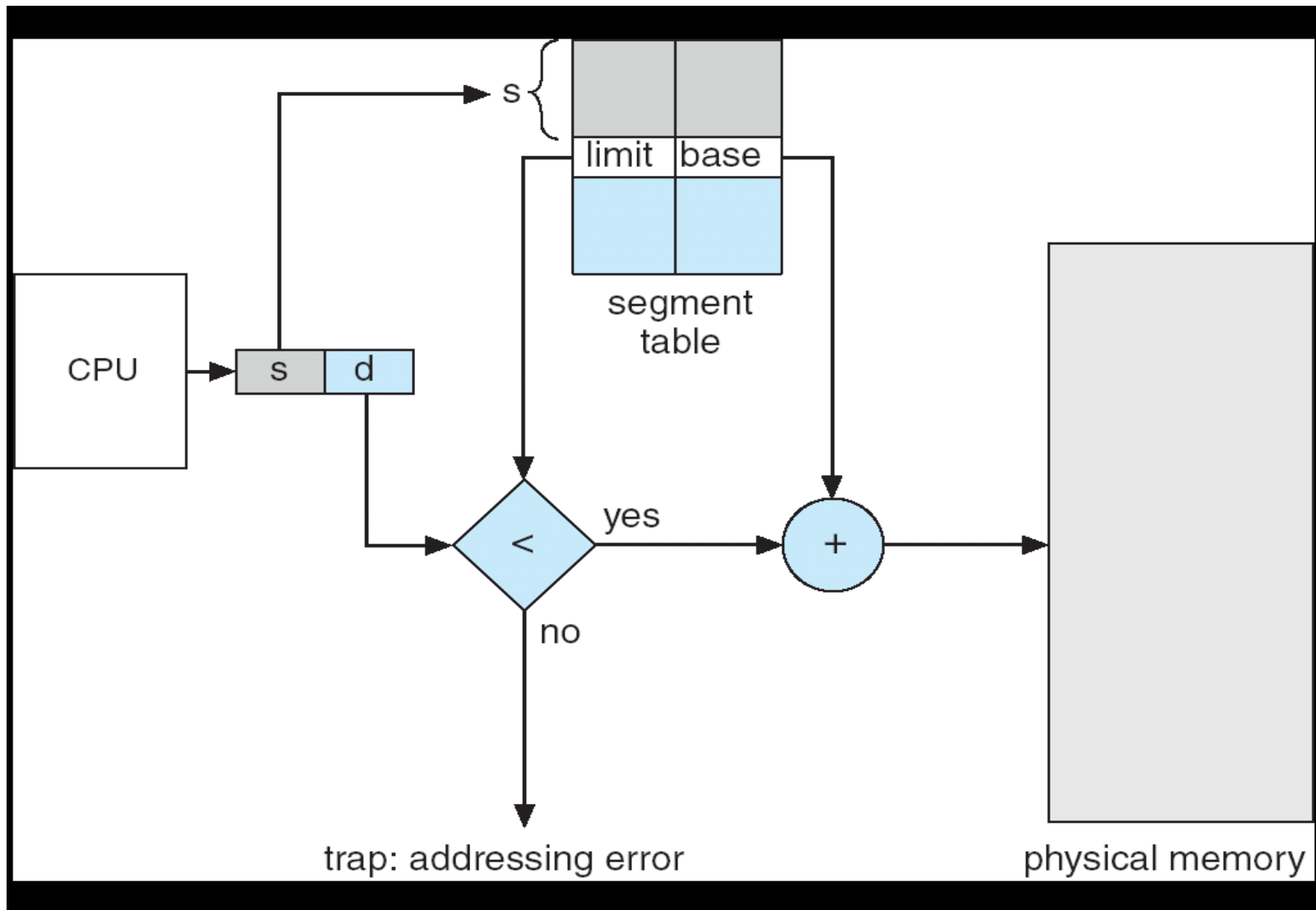
- Relocation.
 - dynamic
 - by segment table
- Sharing.
 - shared segments
 - same segment number
- Allocation.
 - first fit/best fit
 - external fragmentation

Difference between paging and segmentation

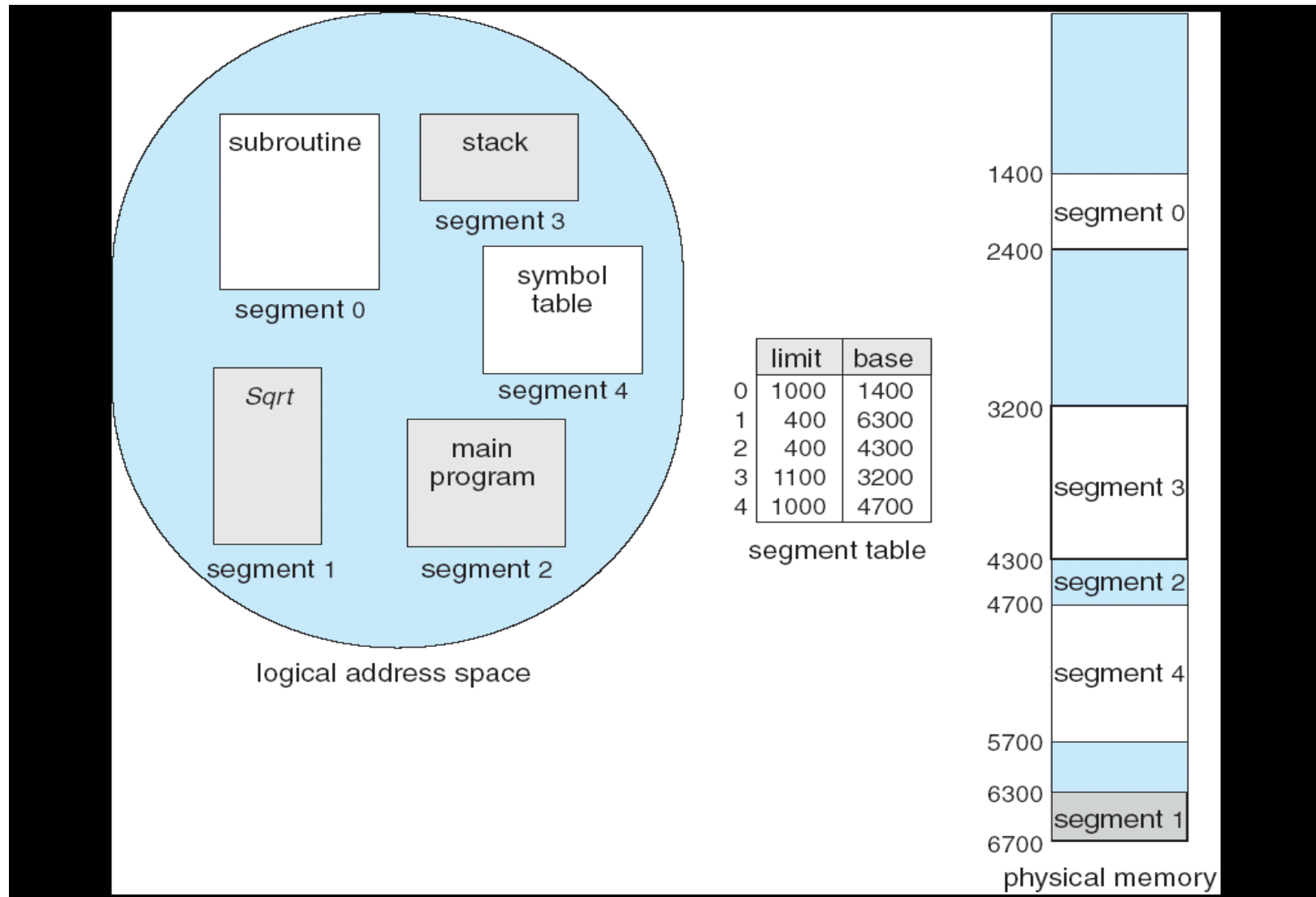
Segmentation Architecture (Cont.)

- Protection. With each entry in segment table associate:
 - validation bit = 0 \Rightarrow illegal segment
 - read/write/execute privileges
- Protection bits associated with segments; code sharing occurs at segment level
- Since segments vary in length, memory allocation is a dynamic storage-allocation problem
- A segmentation example is shown in the following diagram

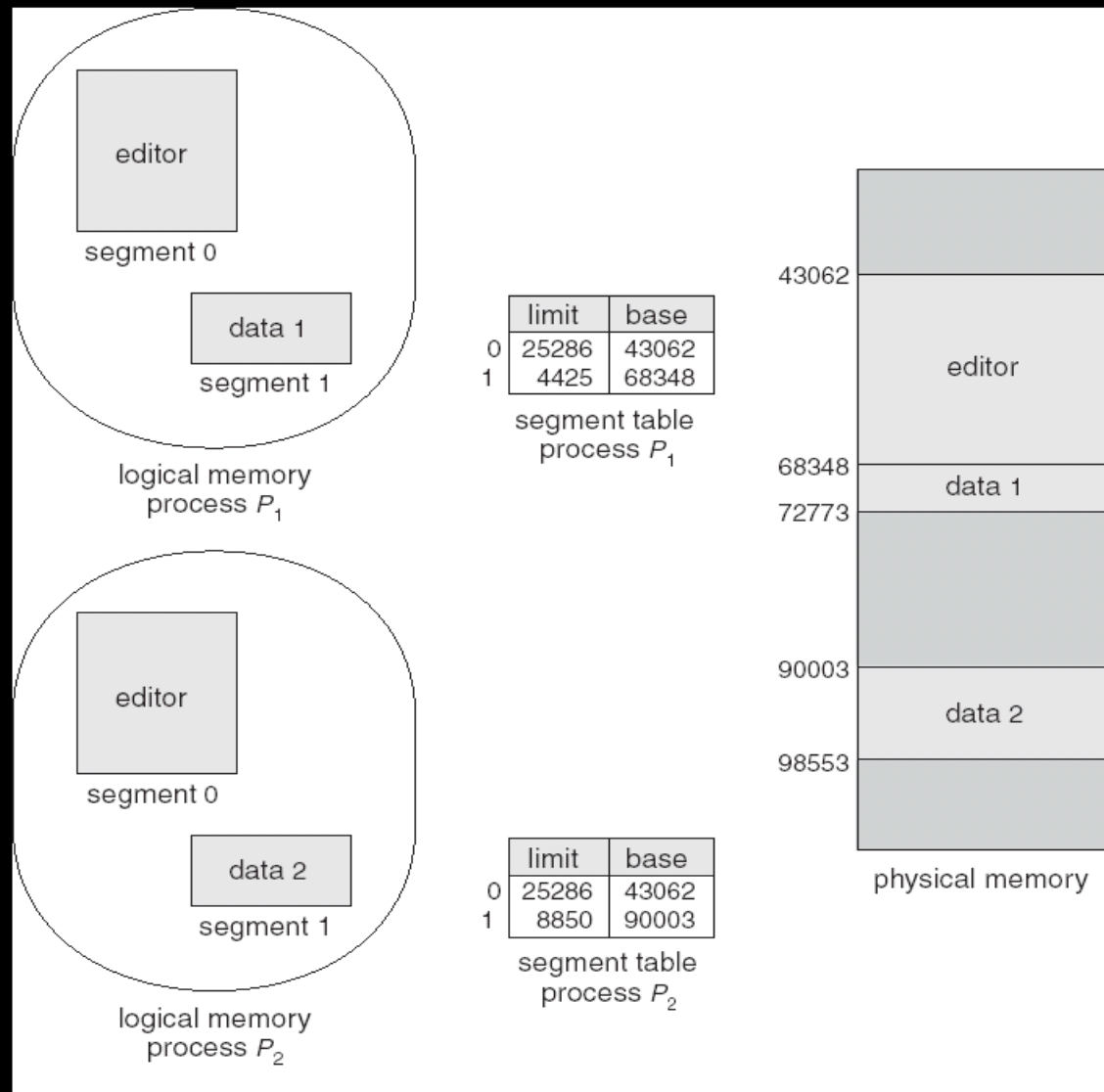
Address Translation Architecture



Example of Segmentation



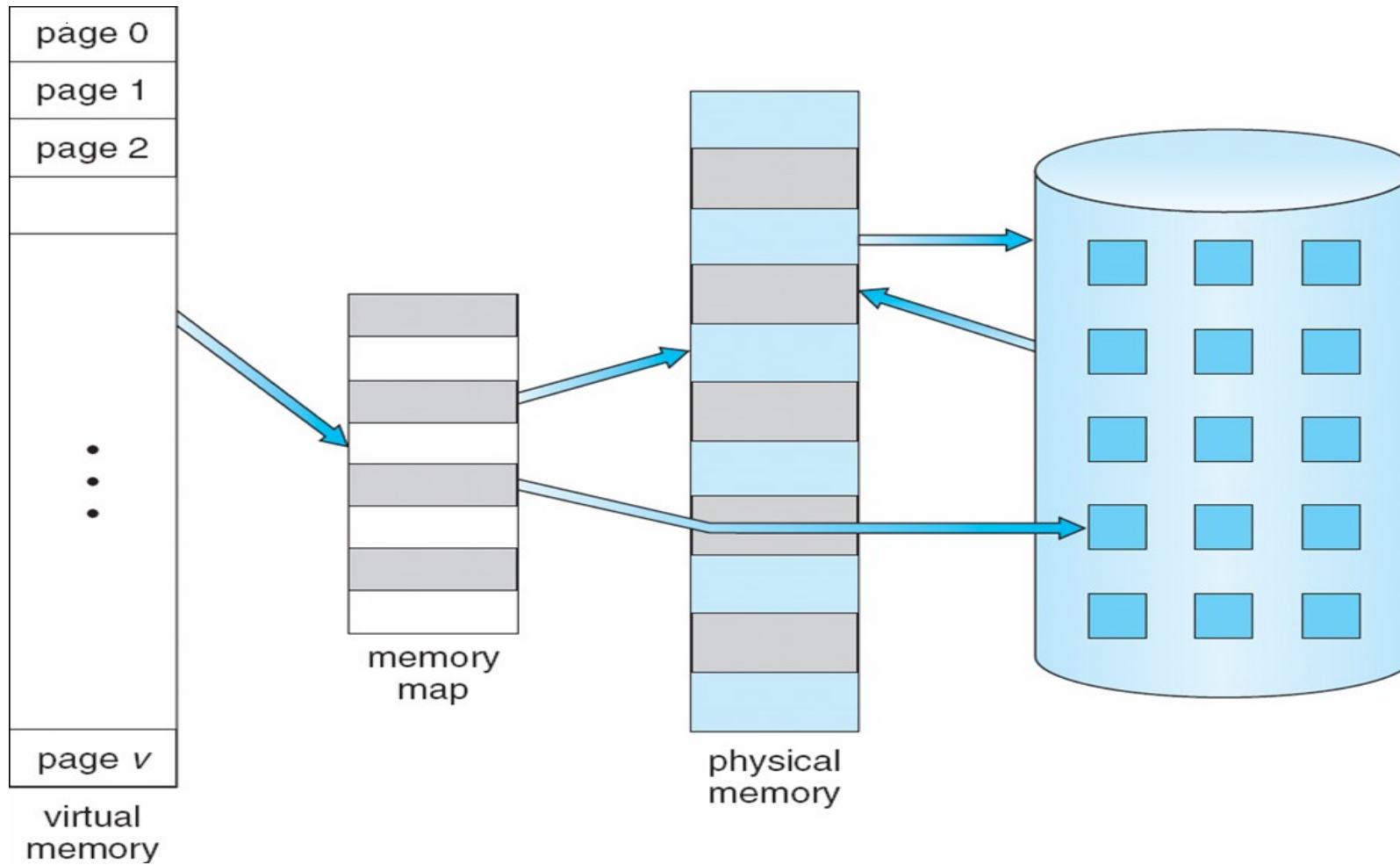
Sharing of Segments



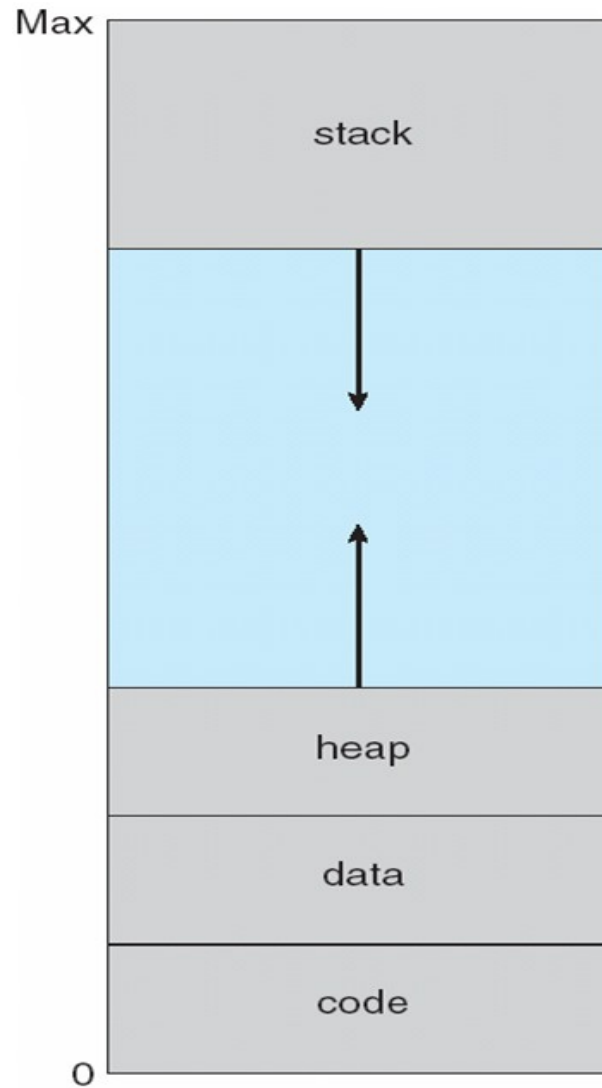
Virtual memory

- separation of user logical memory from physical memory.
- 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:
 1. Demand paging
 2. Demand segmentation

Virtual Memory That is Larger Than Physical Memory



Virtual-address Space



Page Faults

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 \Rightarrow abort

Just not in memory

Get empty frame

Swap page into frame

Reset tables

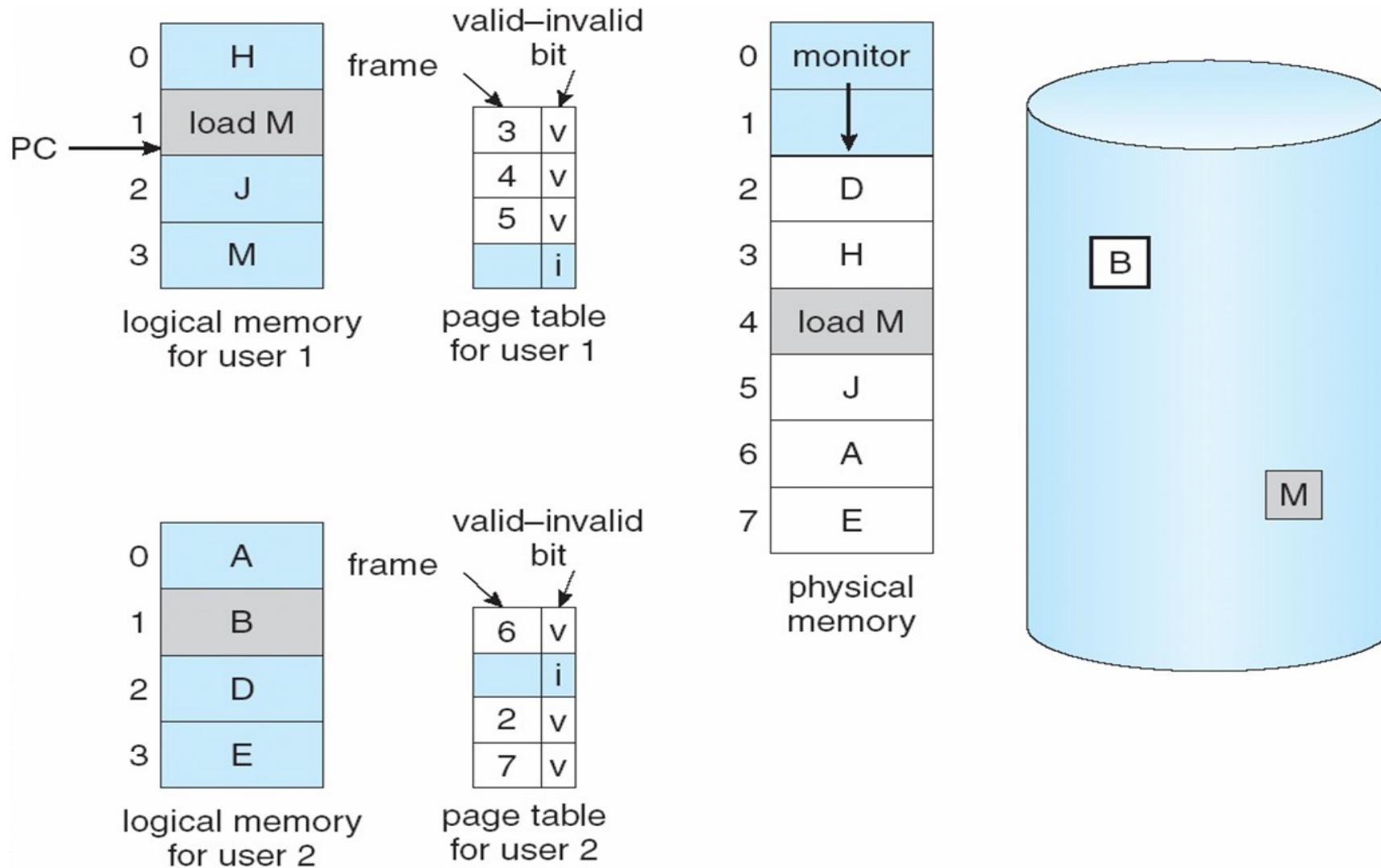
Set validation bit = **v**

Restart the instruction that caused the page fault

Page 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

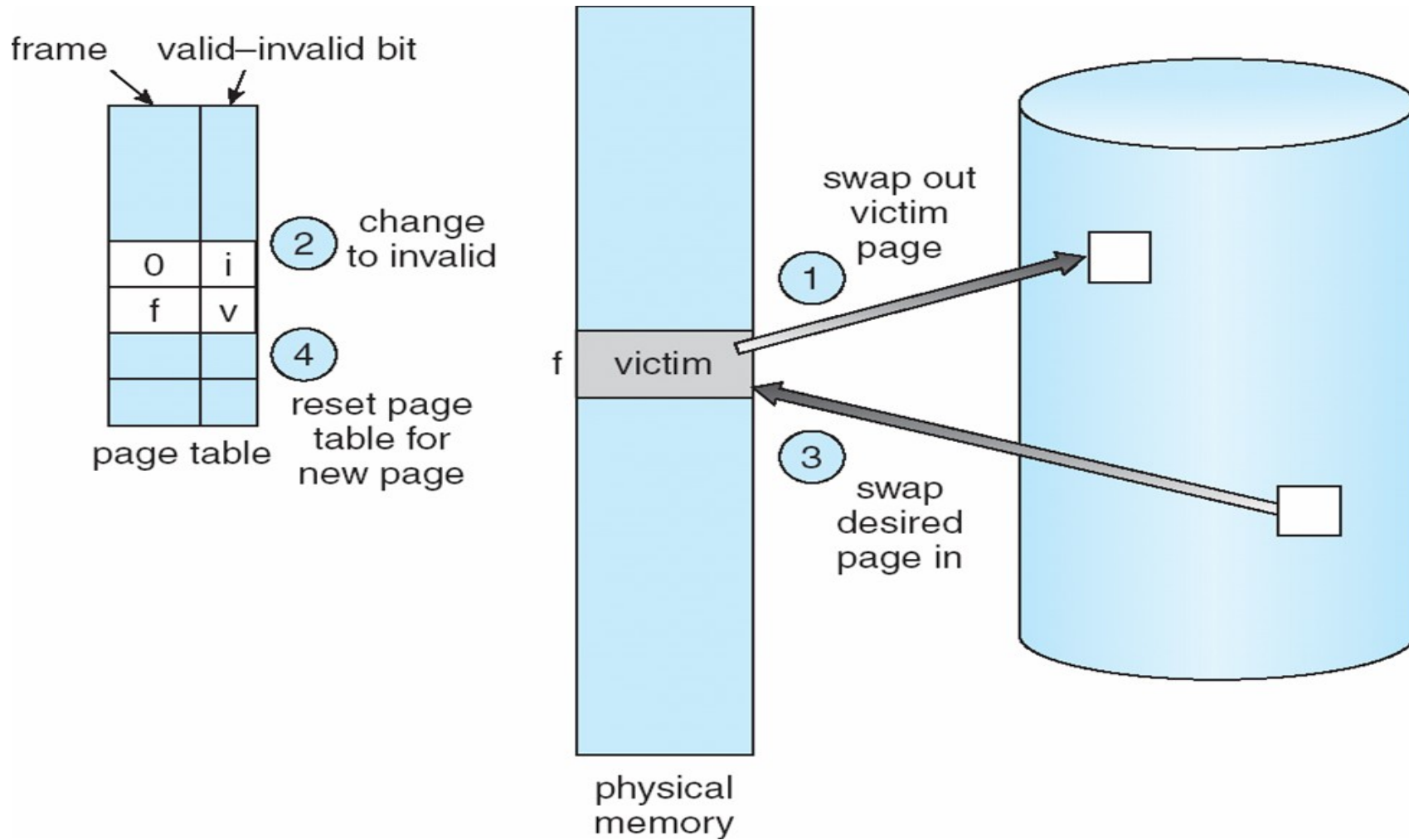
Need For Page Replacement



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

- We 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 :

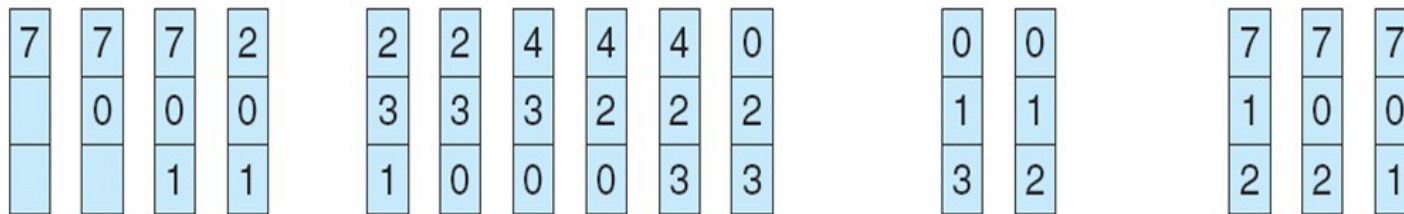
1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5

First-In-First-Out (FIFO) Algorithm

- Suffers from Belady's Anomaly: more frames \Rightarrow more page faults.
- Example:

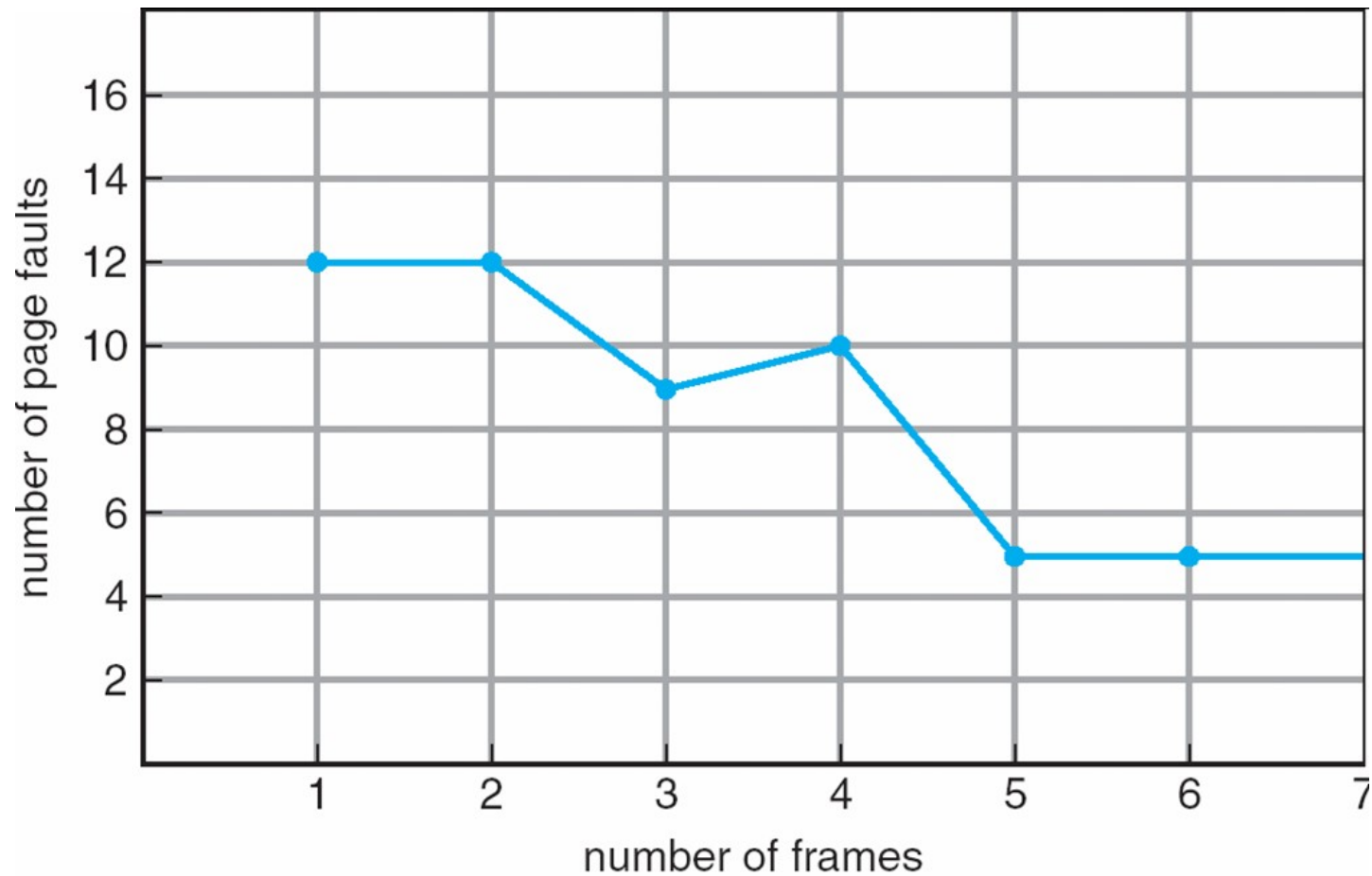
reference string

7 0 1 2 0 3 0 4 2 3 0 3 2 1 2 0 1 7 0 1



page frames

FIFO Illustrating Belady's Anomaly

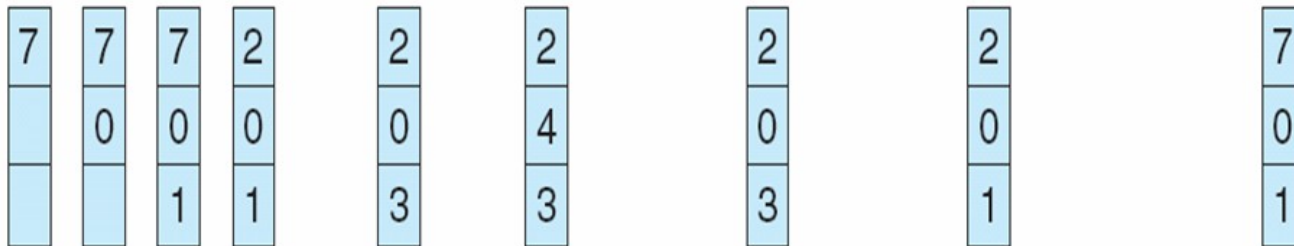


Optimal Algorithm

Replace page that will not be used for longest period of time

reference string

7 0 1 2 0 3 0 4 2 3 0 3 2 1 2 0 1 7 0 1



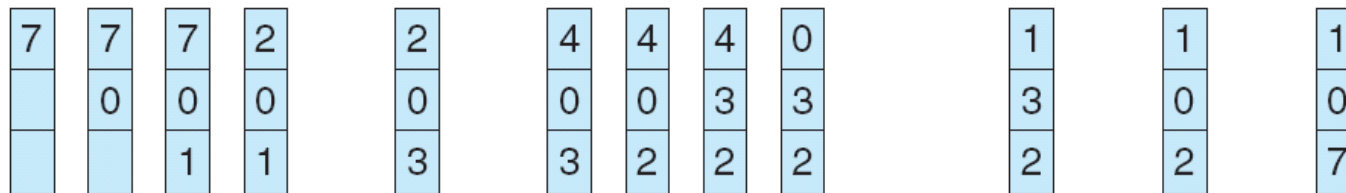
page frames

Least Recently Used (LRU) Algorithm

- Replace the page which is least recently used in past.
-

reference string

7 0 1 2 0 3 0 4 2 3 0 3 2 1 2 0 1 7 0 1



page frames

Allocation of Frames

- Each process needs minimum number of pages
- Two major allocation schemes :
 - 1. 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
 - 2. 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 multi-programming another process added to the system

Thrashing \equiv a process is busy swapping pages in and out

Thrashing (Cont.)

