OPERATING SYSTEMS

Memory Management

Likitha P

Department of Computer Science

OPERATING SYSTEMS

Memory Allocation (Partitioning, Relocation), Fragmentation

Likitha P

Department of Computer Science

Slides Credits for all PPTs of this course

- The slides/diagrams in this course are an **adaptation**, **combination**, and **enhancement** of material from the following resources and persons:

1. Prof. Venkatesh Prasad, Department of CSE, PES University.
2. Prof. Chandravva Hebbi, Department of CSE, PES University.
3. Slides of Operating System Concepts, Abraham Silberschatz, Peter Baer Galvin, Greg Gagne - 9th edition 2013 and some slides from 10th edition 2018.
4. Some conceptual text and diagram from Operating Systems - Internals and Design Principles, William Stallings, 9th edition 2018.
5. Some presentation transcripts from A. Frank – P. Weisberg.
6. Some conceptual text from Operating Systems: Three Easy Pieces, Remzi Arpaci-Dusseau, Andrea Arpaci Dusseau.

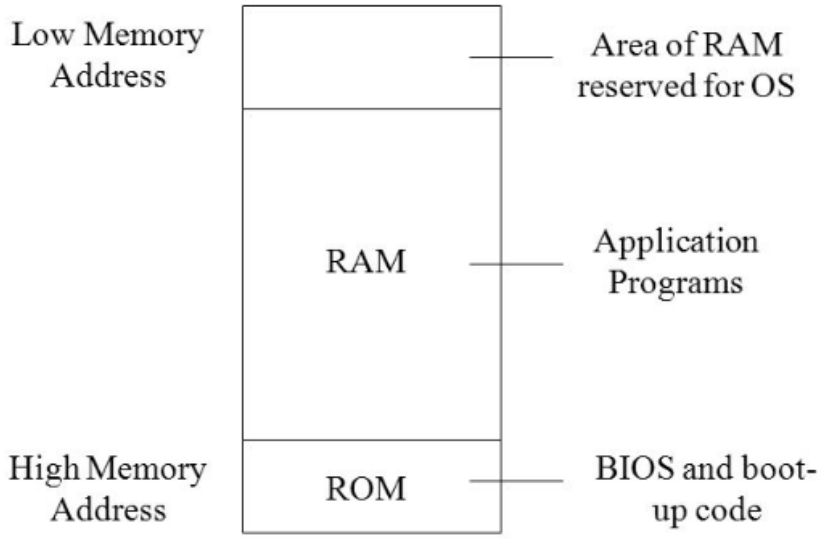
Memory Allocation Techniques Overview

- The common memory allocation techniques are:
  - Single Contiguous Allocation – simplest method. All memory, except that reserved for OS, is available to a process.
  - Partitioned Allocation - Memory is divided into different blocks and each process is allocated according to need.
  - Segmented Memory Management – Memory is divided into different segments (logical grouping of data and code).
  - Paged Memory management – Memory is divided into fixed sized units called frames, used in virtual memory environment.

  (Most OS use Segmentation with Paging.)

Contiguous Allocation – Kernel and User space

- Main memory must support both OS and user processes.
- Limited resource, must be allocated efficiently amongst OS and user processes.
- Contiguous allocation is an early method of organizing the main memory.
- Main memory is usually organized into two **partitions**:
  - Resident OS, usually held in low memory with interrupt vector also in low memory.
  - User processes are held in high memory.
  - Each process is contained in a single section of memory that is contiguous to the section containing the next process.



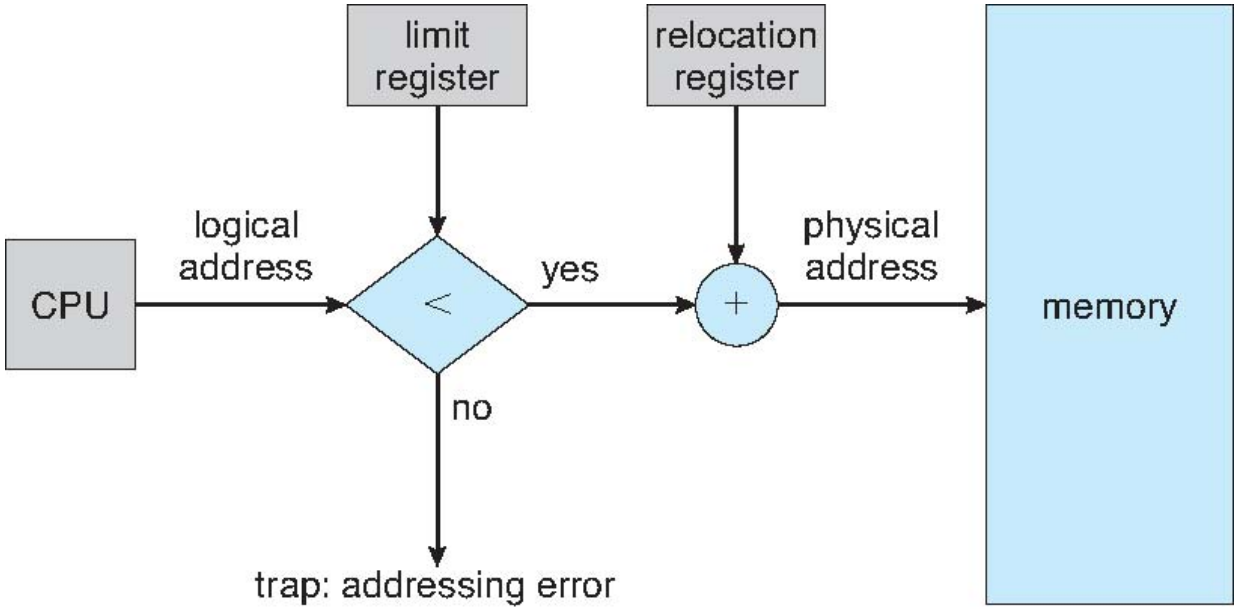
Memory Protection

- Every address generated by a CPU is checked, so it is possible to protect both the operating system and the other users' programs and data from being modified by other running processes. Implemented by base and limit registers. When a process is scheduled, the dispatcher loads these registers with the relevant values.
- The relocation-register scheme provides an effective way to allow the operating system's size to change dynamically.
  - This flexibility is desirable in many situations. For example, the operating system contains **code** and **buffer space** for **device drivers**. If a device driver is not currently in use, it makes little sense to keep it in memory; instead, it can be loaded into memory only when it is needed. Likewise, when the device driver is no longer needed, it can be removed and its memory allocated for other needs.

Contiguous Allocation (Cont.)

- Relocation registers used to protect user processes from each other, and from changing operating-system code and data
  - Base register contains value of smallest **physical** address
  - Limit register contains range of **logical** addresses – each logical address must be less than the limit register
  - MMU maps logical address *dynamically*
  - Can then allow actions such as kernel code being **transient** (it comes and goes as needed) and kernel/OS changing size during program execution.

Hardware Support for Relocation and Limit Registers



When the CPU scheduler selects a process for execution, the dispatcher loads the relocation and limit registers with the correct values as part of the context switch.

Advantages and Disadvantages of Fixed Partition

Advantages:

- Easy to implement.
- Minimal OS overhead.

Disadvantages:

- Internal Fragmentation.
- External Fragmentation.
- Limited process size.
- Limitation on Degree of Multiprogramming.

Multiple-partition allocation

- Multiple-partition allocation
  - Divide memory into several **fixed-size partitions**.
  - Each partition may contain exactly one process.
  - Degree of multiprogramming is bounded by number of partitions.

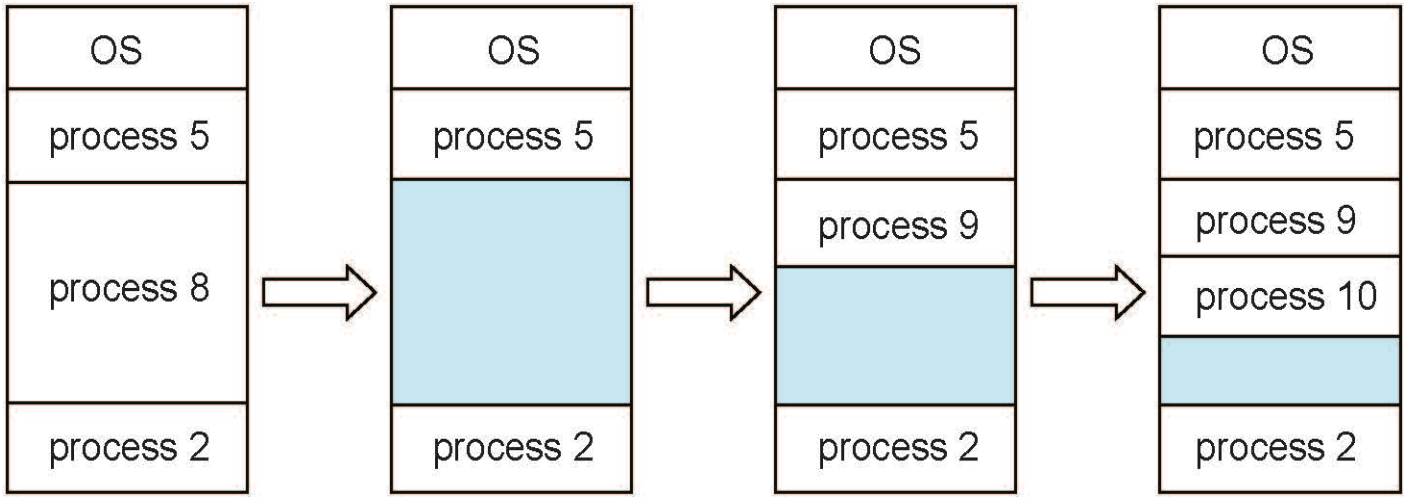
Variable partition sizes for efficiency. This scheme keeps a table indicating which parts of memory are available/occupied.

Hole – block of available memory; holes of various size are scattered throughout memory.

Also known as variable partition and dynamic partition allocation.

Multiple-partition allocation (Cont.)

- When a process arrives, it is allocated memory from a hole large enough to accommodate it.
- Process exiting frees its partition, adjacent free partitions are combined.
- Operating system maintains information about:
  a) allocated partitions b) free partitions (hole)



Multiple Partition-Allocation Strategies

How to satisfy a request of size ***n*** from a list of free holes?

- **First-fit**: Allocate the ***first*** hole that is big enough (generally faster).
- **Best-fit**: Allocate the ***smallest*** hole that is big enough; must search entire list, unless ordered by size
  - Produces the smallest leftover hole
- **Next-fit**: Similar to first fit but it will search for first available fit from the last allocation point.
- **Worst-fit**: Allocate the ***largest*** hole; must also search entire list
  - Produces the largest leftover hole

First-fit and best-fit are better than worst-fit in terms of speed and storage utilization

Advantages and Disadvantages of Multiple Partition

Advantages:

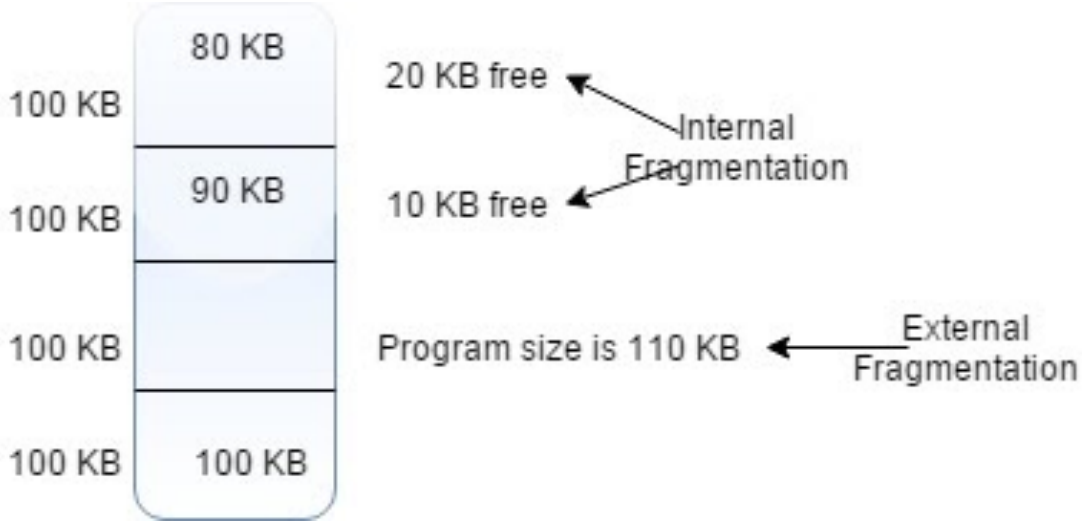
- No Internal Fragmentation: Allocate the *first* hole that is big enough (generally faster)
- No restriction on degree of multiprogramming
- No limitation on size of process

Disadvantages:

- Difficult to implement
- External Fragmentation

Fragmentation

- **Internal Fragmentation** – allocated memory may be slightly larger than requested memory; this size difference is internal to a partition, but not being used.
- **External Fragmentation** – total memory space exists to satisfy a request, but it is not contiguous.
  - First fit statistical analysis reveals that given *N* allocated blocks (66% for example), another 0.5 *N* blocks (33% for example) are lost to fragmentation.
  - 1/3 of memory may be unusable -> **50-percent rule**
    - Unusable memory = (0.5N)/(N+0.5N)= 1/3



Fragmentation (Cont.)

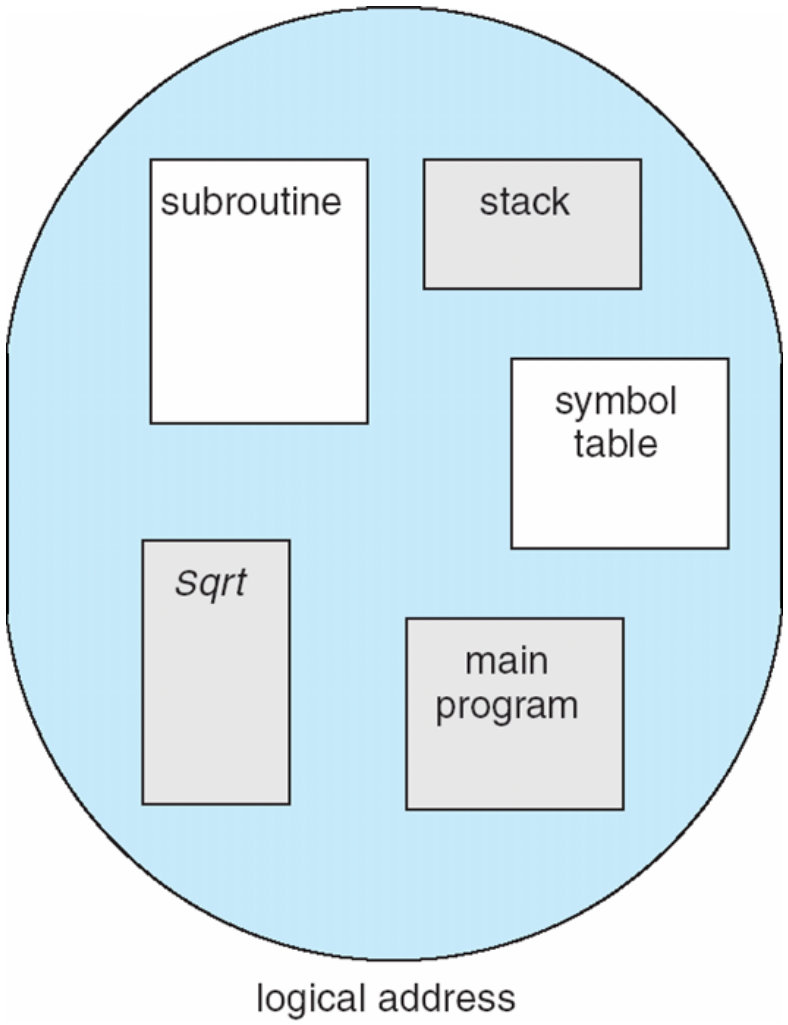
- Reduce external fragmentation by **compaction**
  - Shuffle memory contents to place all free memory together in one large block.
  - Compaction is possible *only* if relocation is dynamic, and is done at execution time.
  - I/O problem
    - Latch job in memory while it is involved in I/O
    - Do I/O only into OS buffers
- Now consider that backing store has same fragmentation problems.
- Read about **Buddy System Memory allocation** and **Memory Compression** techniques.

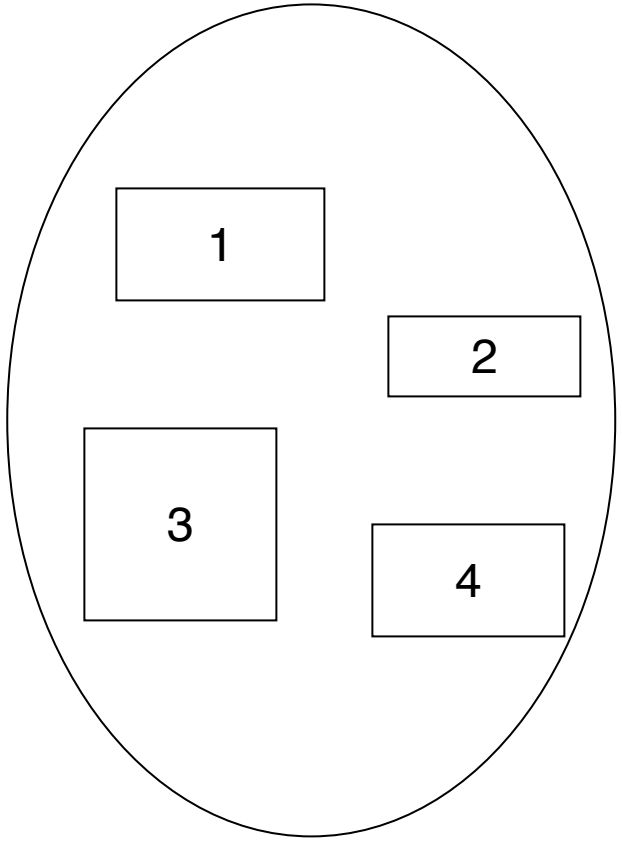
Segmentation

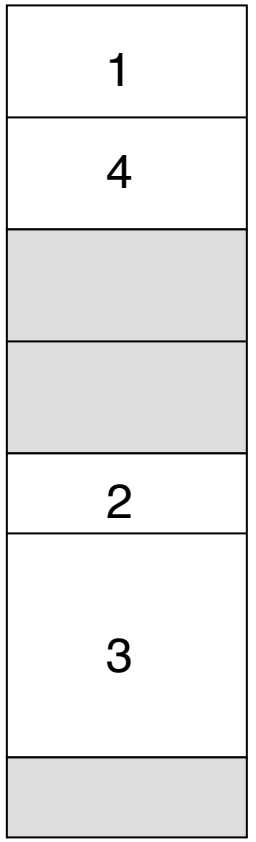
- Memory-management scheme that supports programmer's view of memory.
- A program is a collection of segments, a logical grouping of data and code.
  - A segment is a logical unit such as: main program, procedure, function/method, object, local variables, global variables, common block, stack, symbol table, arrays, etc.
  - Variable sized segments and no ordering in the segments.
  - Each segment has a name and a length.
  - Addresses specify segment name + offset within the segment.

OPERATING SYSTEMS

User's View of a Program







Segmentation Basics

- Segmentation is a technique for breaking memory up into logical pieces
- Each “piece” is a grouping of related information
  - data segments for each process
  - code segments for each process
  - data segments for the OS, etc.
- Segmentation permits the physical address space of a process to be non-contiguous.
- Like paging, use virtual addresses and use disk to make memory look bigger than it really is.
- Segmentation can be implemented with or without paging.

Segmentation Architecture

- Logical address for a segment consists of the tuple:

  <segment-number s, offset d>

- **Segment table** – maps two-dimensional logical addresses to one-dimensional physical addresses; each table entry has:
  - **Segment base** – contains the starting physical address where the segments reside in memory.
  - **Segment 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*** < **STLR**

Segmentation Architecture (Cont.)

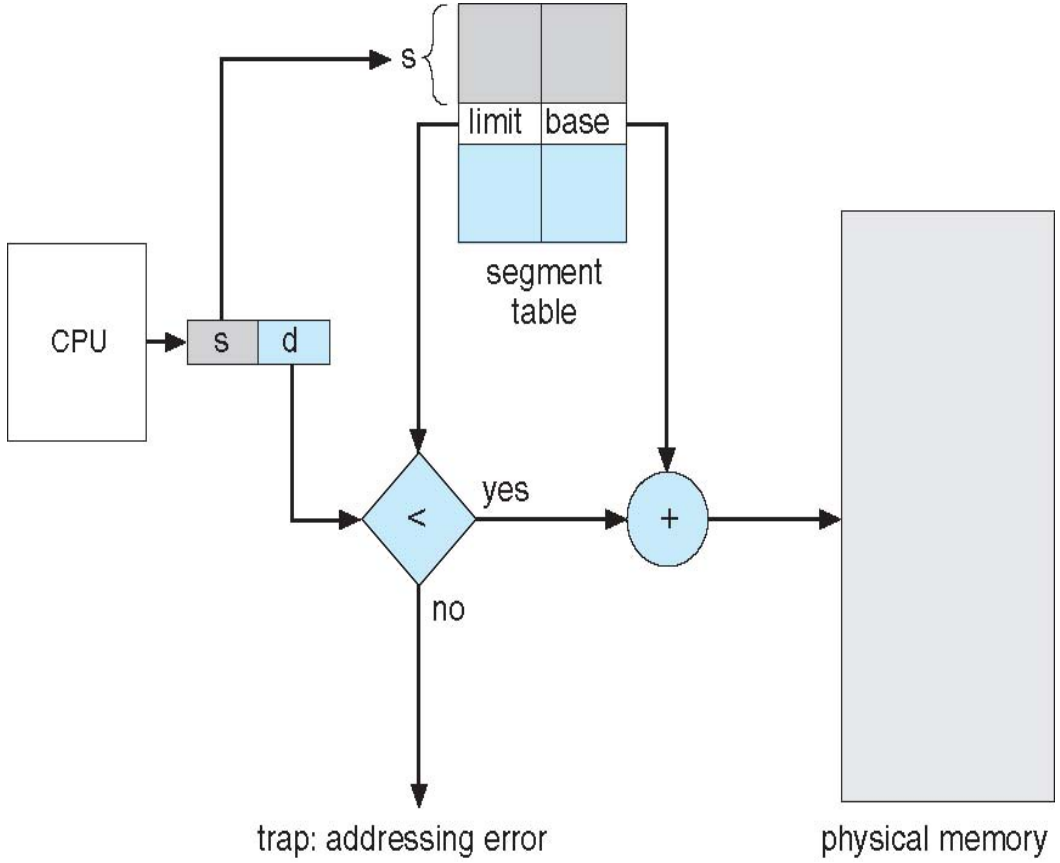
- Protection
  - With each entry in segment table associate:
    - validation bit = 0 $\Rightarrow$ illegal segment
    - read/write/execute privileges
- Protection bits associated with segments; code sharing occurs at segment level
- Since segments vary in length, memory allocation is a dynamic storage-allocation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

Segmentation Hardware

- Sounds very similar to paging
- Big difference – segments can be variable in size
- As with paging, to be effective hardware must be used to translate logical address
- Most systems provide segment registers
- If a reference isn't found in one of the segment registers
  - trap to operating system
  - OS does lookup in segment table and loads new segment descriptor into the register
  - return control to the user and resume

OPERATING SYSTEMS
Segmentation Hardware (Cont.)

- A logical address consists of two parts: a segment number, s, and an offset into that segment, d.
- The segment number is used as an index to the segment table. The offset d of the logical address must be between 0 and the segment limit. If it is not, we trap to the operating system (logical addressing attempt beyond end of segment).
- When an offset is legal, it is added to the segment base to produce the address in physical memory of the desired byte. The segment table is thus essentially an array of base–limit register pairs.

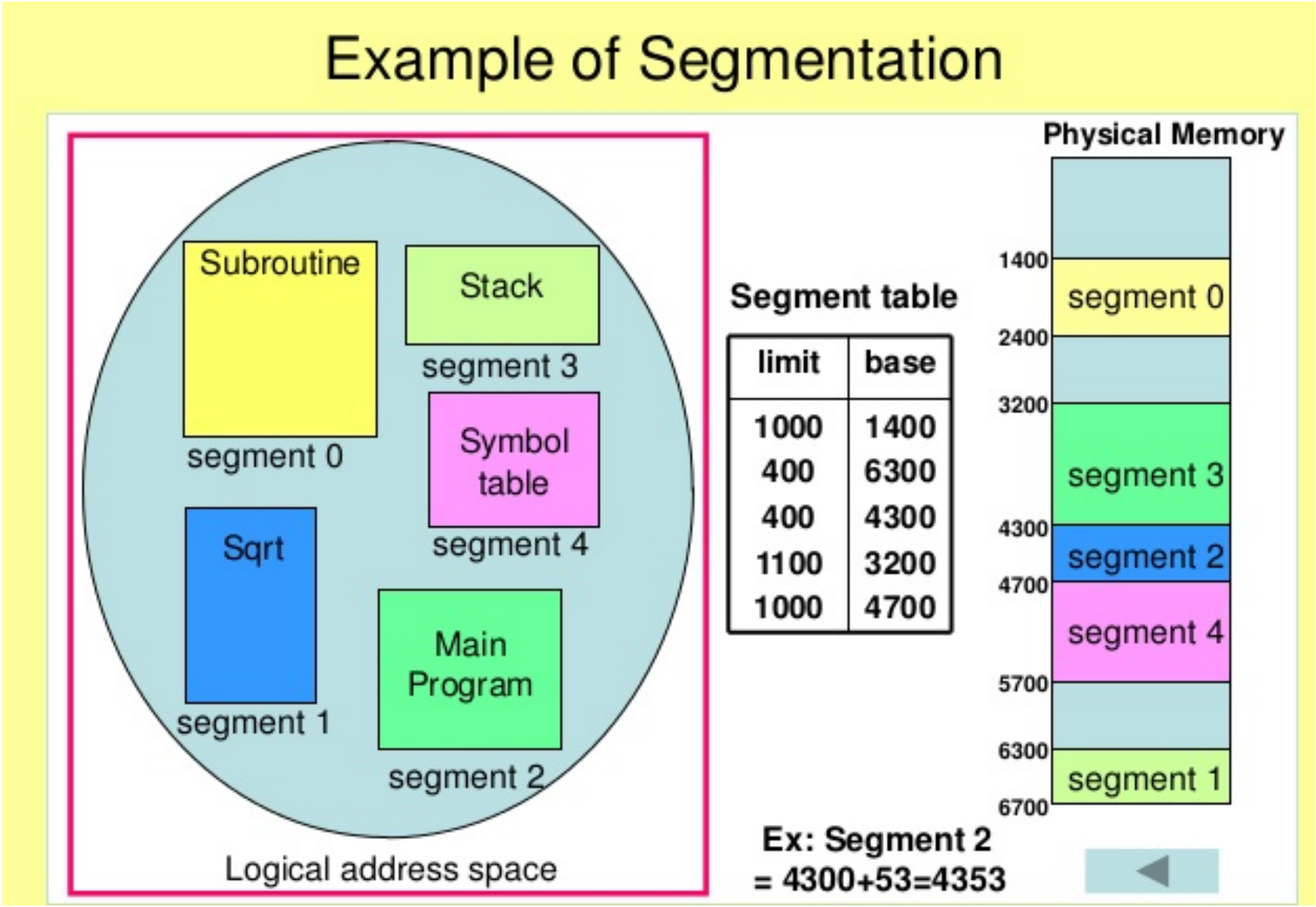


Dynamics of Segmentation

- Typically, each process has its own segment table.

- Similar to paging, each segment table entry contains a present (valid-invalid) bit and a modified bit.
- If the segment is in main memory, the entry contains the starting address and the length of that segment.
- Other control bits may be present if protection and sharing is managed at the segment level.
- Logical to physical address translation is similar to paging except that the offset is added to the starting address (instead of appended).

Segmentation Example



Segmentation Example (Cont.)

- We have five segments numbered from 0 through 4. The segments are stored in physical memory as shown.
- The segment table has a separate entry for each segment, giving the beginning address of the segment in physical memory (or base) and the length of that segment (or limit).
- For example, segment 2 is 400 bytes long and begins at location 4300.
  - Thus, a reference to byte 53 of segment 2 is mapped onto location 4300 + 53 = 4353.
  - A reference to segment 3, byte 852, is mapped to 3200 (the base of segment 3) + 852 = 4052.
  - A reference to byte 1222 of segment 0 would result in a trap to the operating system, as this segment is only 1,000 bytes long.

Segmentation Summary

- In each segment table entry, we have both the starting address and length of the segment; the segment can thus dynamically grow or shrink as needed.
- However variable length segments introduce external fragmentation and are more difficult to swap in and out.
- It is natural to provide protection and sharing at the segment level since segments are visible to the programmer (pages are not).
- Useful protection bits in segment table entry:
  - read-only/read-write bit
  - Kernel/User bit

Segmentation Advantages and Disadvantages

Advantages:

- No internal fragmentation.
- Less Overhead compared paging.
- Easier to relocate segments.
- Segment table size is lesser compared to page table.

Disadvantages:

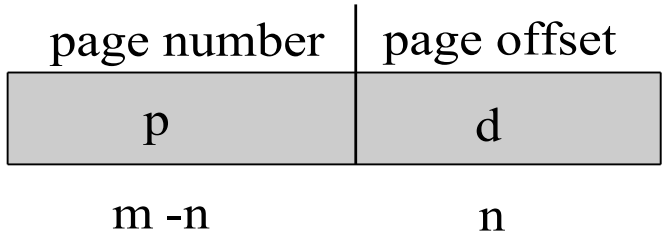
- Variable sized segments leads to external fragmentation in memory.
- Must find a space big enough to place new segments, difficult to allocate contiguous memory to variable sized partition.
- Costly memory management algorithm.

Paging

- Physical address space of a process can be noncontiguous; process is allocated physical memory whenever the latter is available.
  - Avoids external fragmentation
  - Avoids problem of varying sized memory chunks
- Divide physical memory into fixed-sized blocks called **frames.**
  - Size is power of 2, between 512 bytes and 16 Mbytes.
- Divide logical memory into blocks of same size called **pages.**
- **Page size and frame size are defined by the hardware.**
- Keep track of all free frames in free frame list.
- 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.
- Will still have Internal fragmentation.

Address Translation Scheme

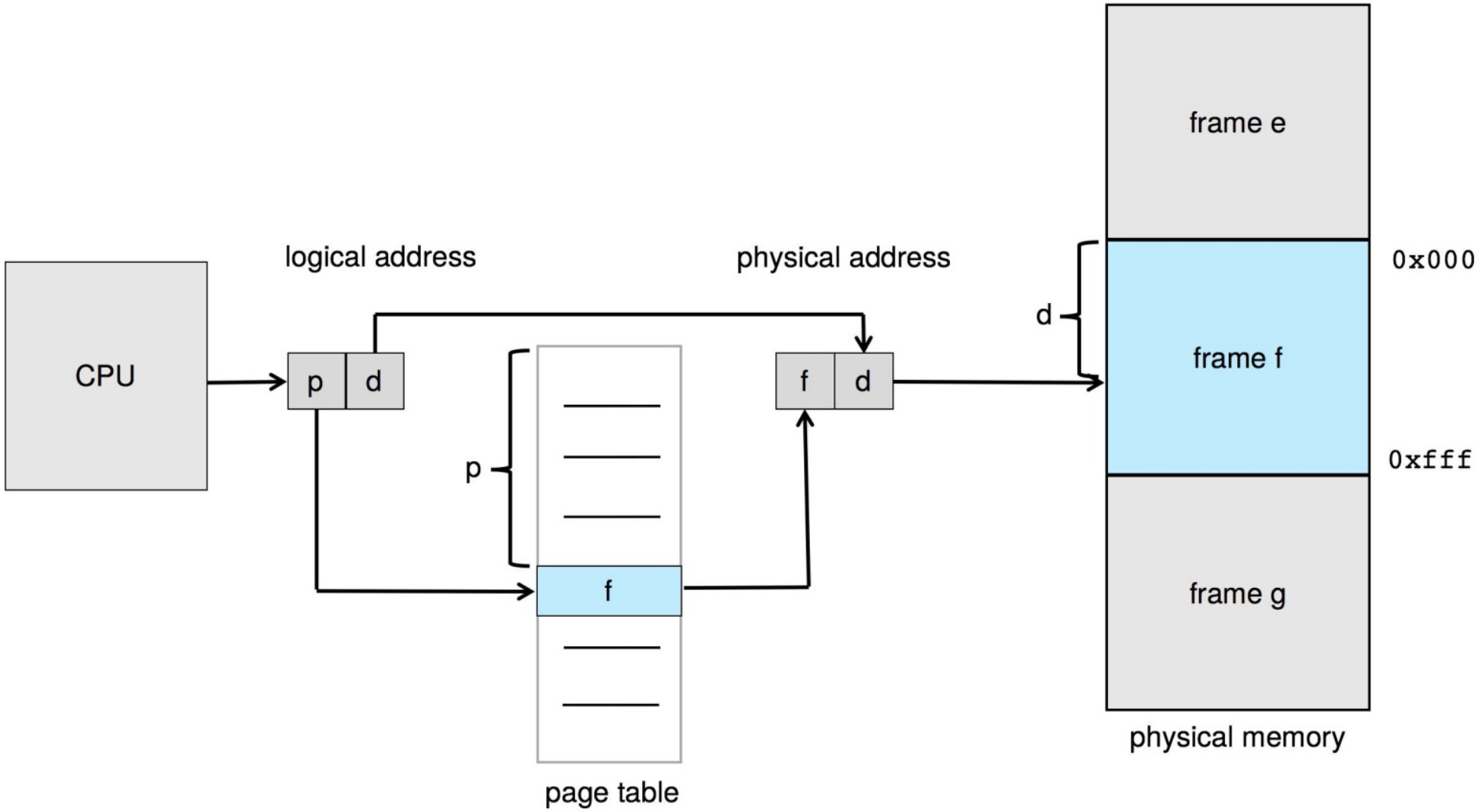
- Address generated by CPU is divided into:
  - **Page number** (***p***) – used as an index into a **page table** which contains base address of each page in physical memory
  - **Page offset** (***d***) – combined with base address to define the physical memory address that is sent to the memory unit



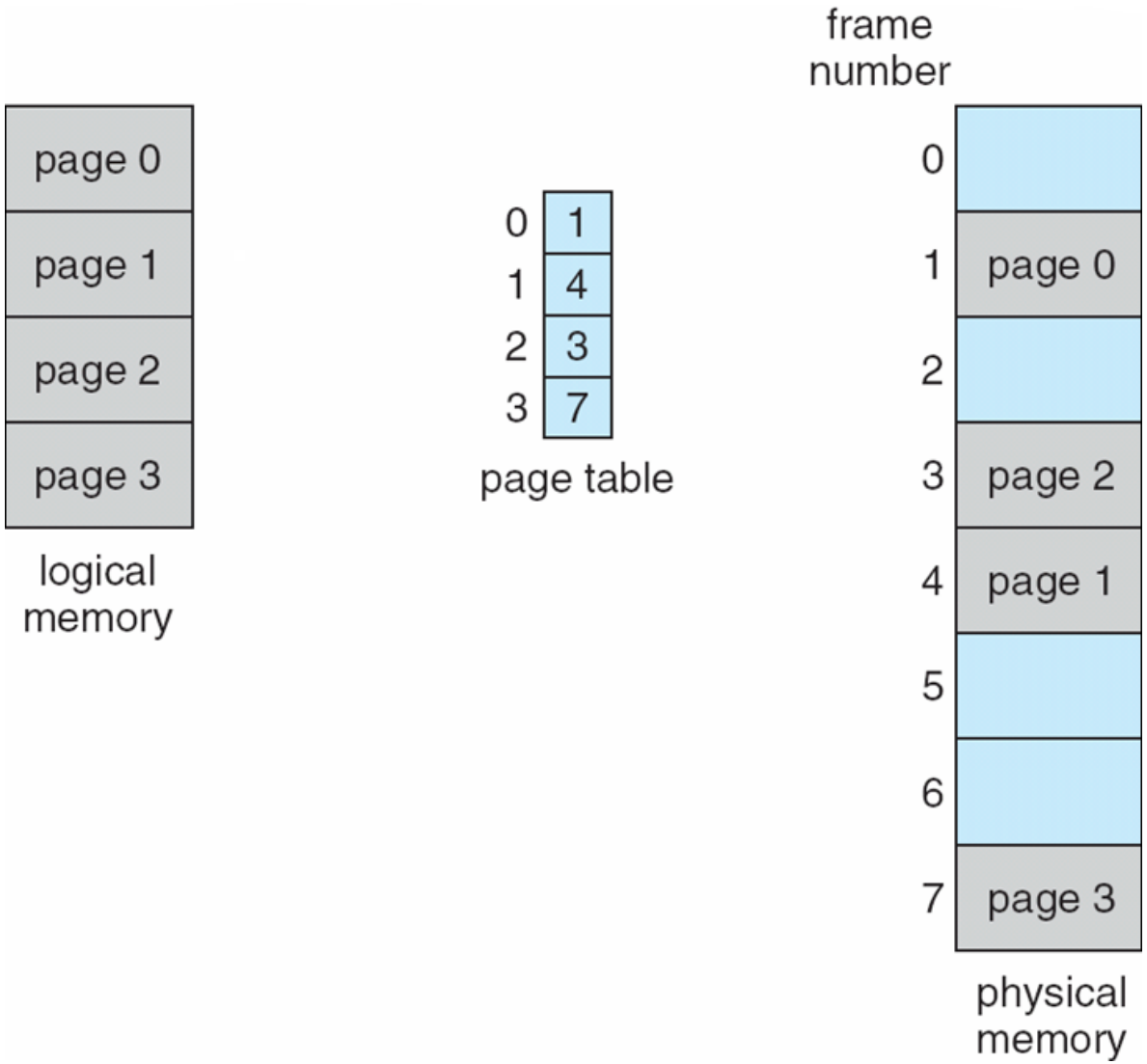
  - For given logical address space 2^m and page size 2^n

OPERATING SYSTEMS

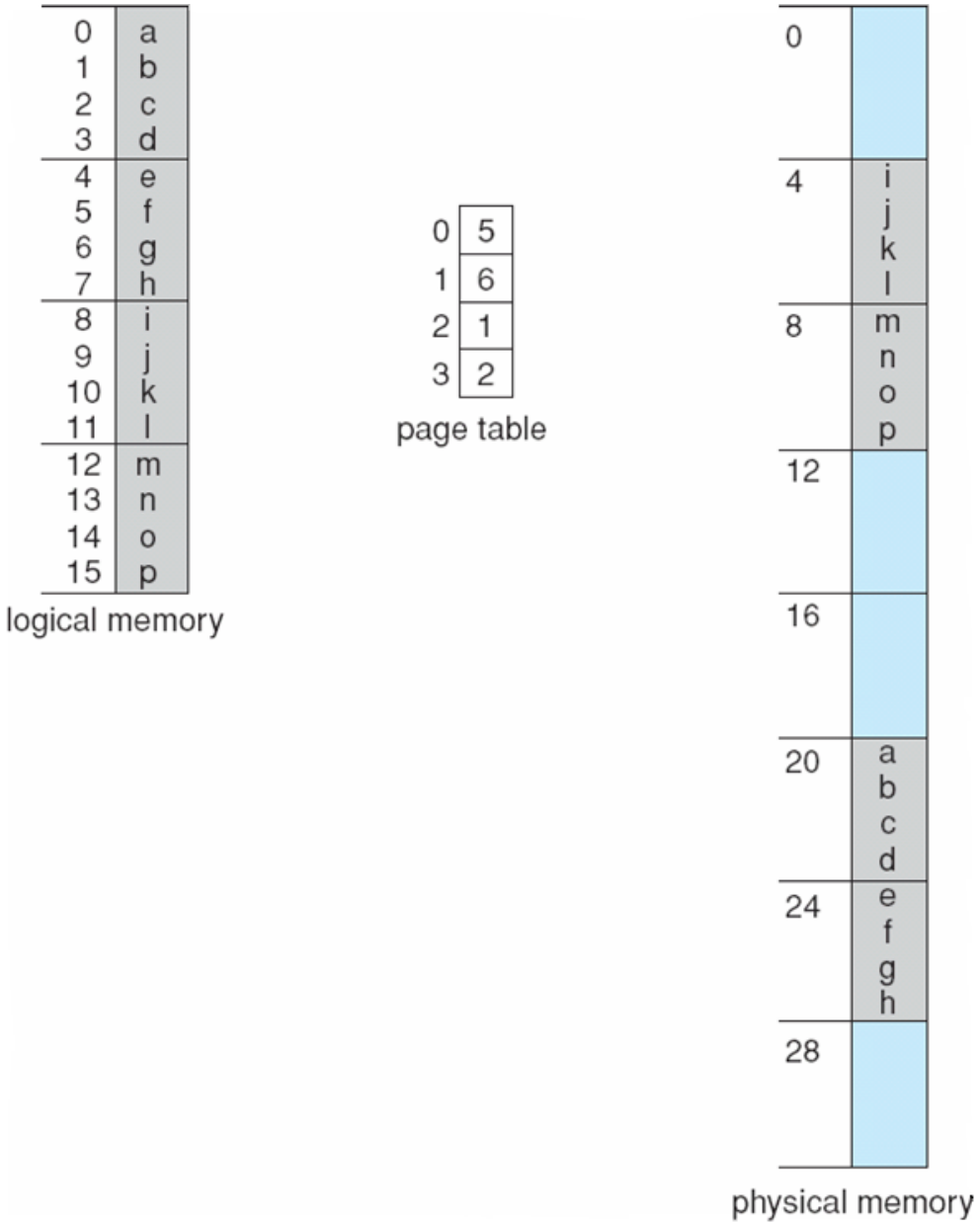
Paging Hardware



Paging Model of Logical and Physical Memory



OPERATING SYSTEMS
Paging Example



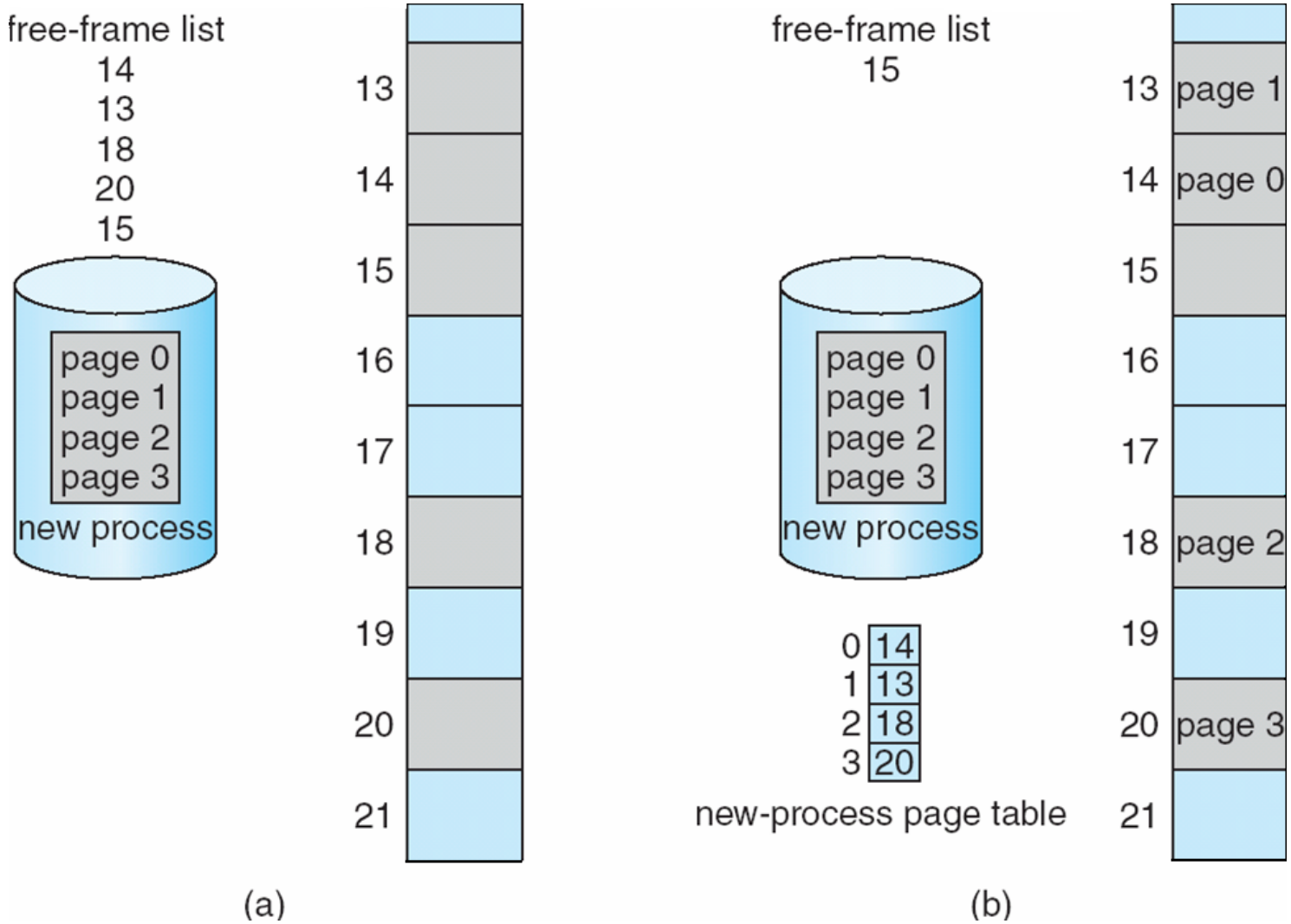
- Here, we are using a page size of 4 bytes and a physical memory of 32 bytes (8 pages).
- Programmer's view of memory is mapped to physical memory.
- Logical address 0 is mapped to physical address 20 [=(5X4) + 0].
- Logical address 1 is mapped to physical address 21 [=(5X4) + 1].
- Similarly, LA 2 and 3 are mapped to PA 22 and 23 respectively.
- Page 0 is mapped to frame 5.

Paging (Cont.)

- Calculating internal fragmentation
  - Page size = 2,048 bytes
  - Process size = 72,766 bytes
  - 35 pages + 1,086 bytes
  - **Internal fragmentation of 2,048 - 1,086 = 962 bytes**
  - Worst case fragmentation = **1 frame – 1 byte**
  - Best case fragmentation = **1 byte**
  - On average, fragmentation = 1 / 2 frame size
  - So are small frame sizes desirable?
  - However, each page table entry takes memory to track.
  - Page table size grows over time.

OPERATING SYSTEMS

Free Frames



Before allocation

After allocation

Implementation of Page Table

- Page table is stored in main memory.
- **Page-table base register** (**PTBR**) points to the page table.
- **Page-table length register** (**PTLR**) 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**).

Translation Look-Aside Buffer

- Some TLBs store **address-space identifiers** (**ASIDs**) in each TLB entry – uniquely identifies each process to provide address-space protection for that process.
  - Otherwise need to flush at every context switch.
- TLBs are typically small (64 to 1,024 entries).
- On a TLB miss, value is loaded into the TLB for faster access next time.
  - Replacement policies must be considered.
  - Some entries can be **wired down** for permanent fast access.

Hardware

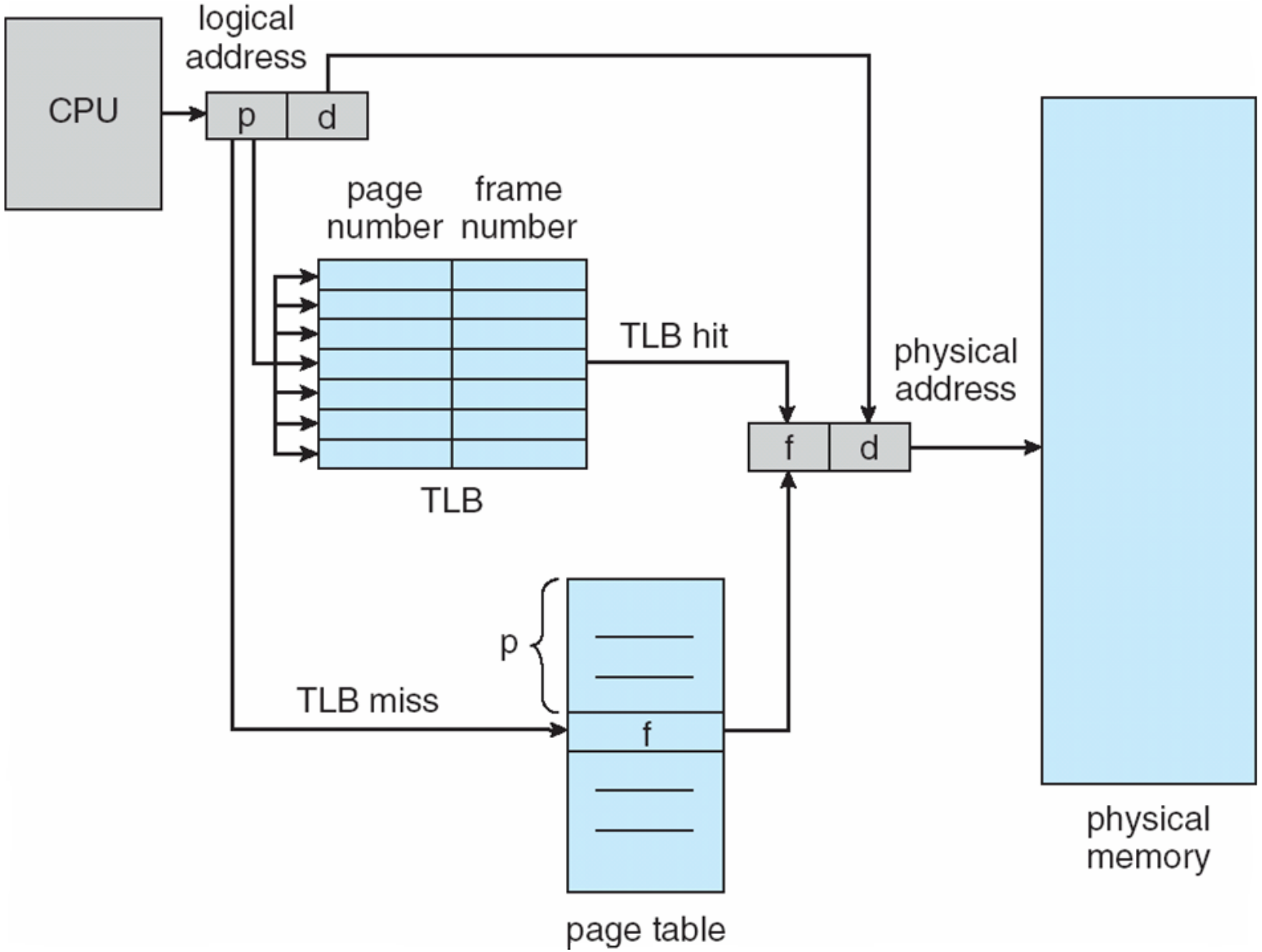
- Associative memory – parallel search

Page #	Frame #

- Address translation (p, d)
  - If p is in associative register, get frame # out
  - Otherwise get frame # from page table in memory

OPERATING SYSTEMS

Paging Hardware With TLB



Effective Access Time

- Hit ratio – percentage of times that a page number is found in the TLB
- An 80% hit ratio means that we find the desired page number in the TLB 80% of the time.
- Suppose that 10 nanoseconds is required to access memory.
  - If we find the desired page in TLB then a mapped-memory access takes 10 ns.
  - Otherwise we need two memory accesses, and access time is 20 ns.
- **Effective Access Time** (**EAT**)

EAT = (TLB_hit_ratio X TLB_hit_time) + (TLB_miss_ratio X TLB_miss_time)

EAT = 0.80 x 10 + 0.20 x 20 = 12 nanoseconds

implying 20% slowdown in access time.

- Consider a more realistic hit ratio of 99%,

EAT = 0.99 x 10 + 0.01 x 20 = 10.1ns

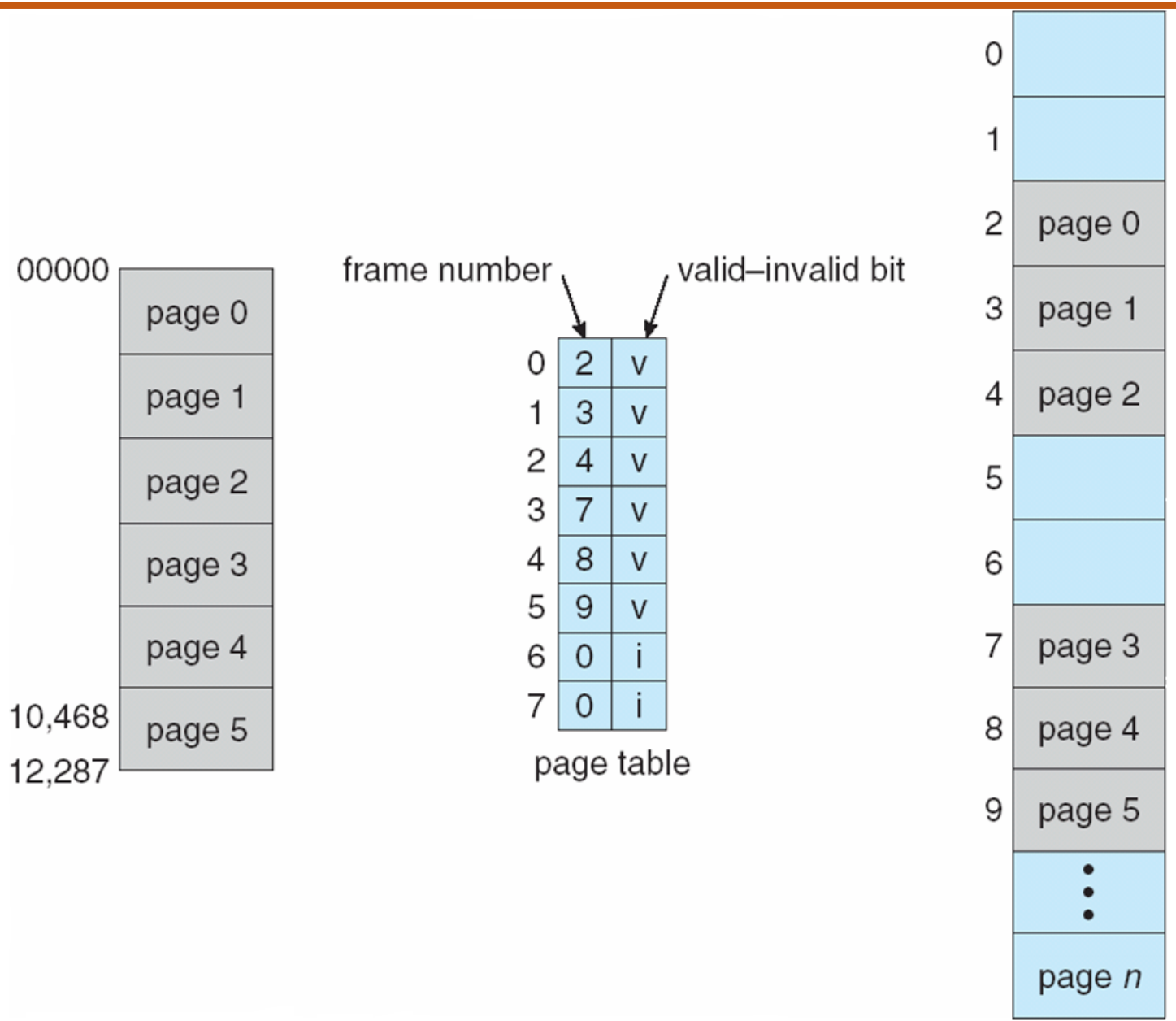
implying only 1% slowdown in access time.

Memory Protection

- Memory protection is implemented by associating protection bit with each frame to indicate if read-only or read-write access is allowed.
  - Can also add more bits to indicate page execute-only, and so on.
- **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.
  - Or use **page-table length register** (**PTLR**).
- Any violations result in a trap to the kernel.

OPERATING SYSTEMS

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



Shared Pages

Shared code

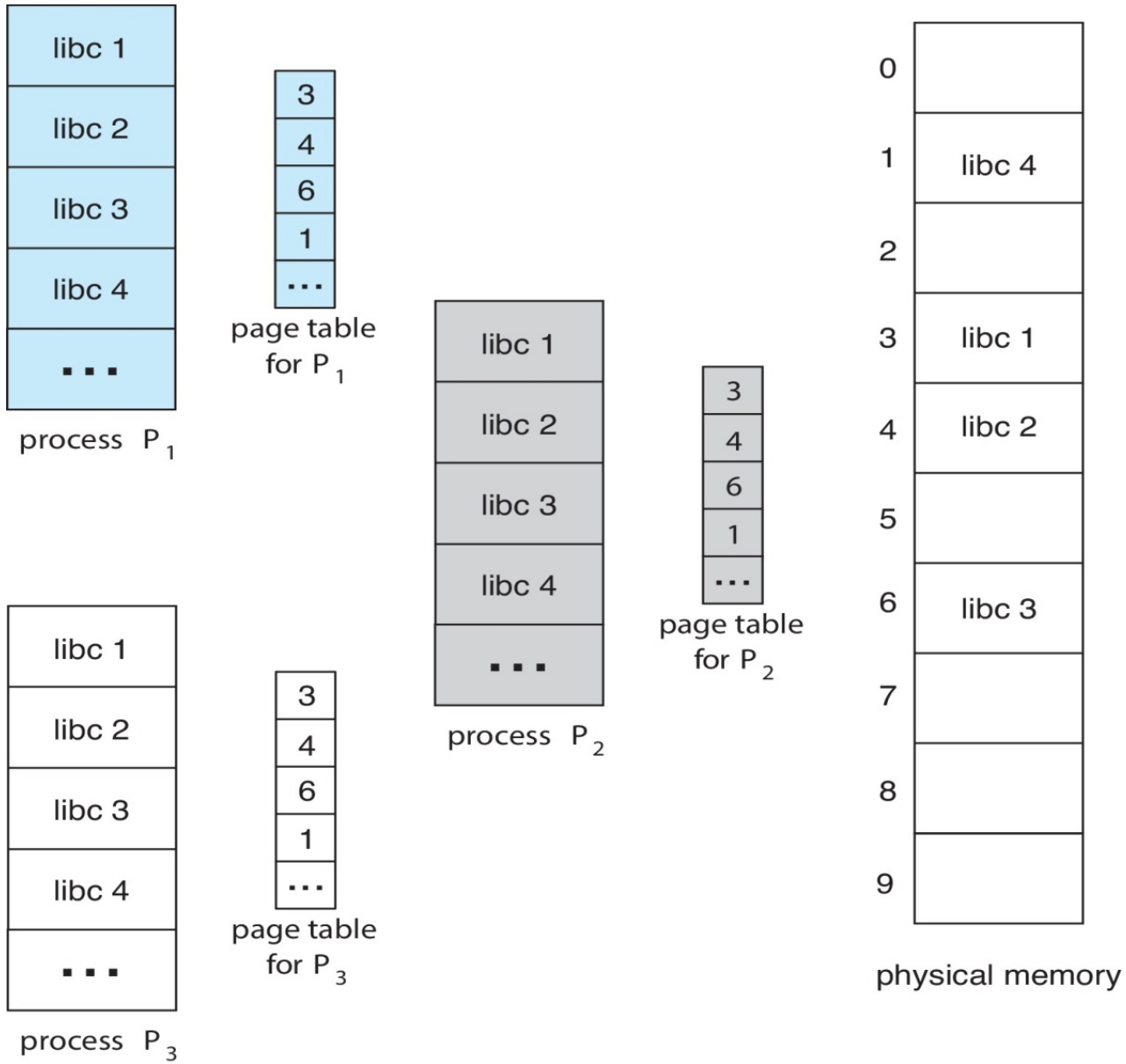
- One copy of read-only (**reentrant**) code shared among processes (i.e., text editors, compilers, window systems)
- Similar to multiple threads sharing the same process space
- Also useful for interprocess communication if sharing of read-write pages is allowed

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

OPERATING SYSTEMS

Shared Pages Example



Paging Advantages and Disadvantages

Advantages:

- No external fragmentation.
- Frames can be allocated in a non-contiguous manner.
- Swapping is easier as pages are equal sized.
- Segment table size is lesser compared to page table.

Disadvantages:

- Internal fragmentation.
- Longer memory access times.
- Page table size grows with increasing sizes of programs.
- Complex memory management technique.

THANK YOU

Likitha P
Department of Computer Science Engineering

likithap@pes.edu