OPERATING SYSTEMS

Memory Management

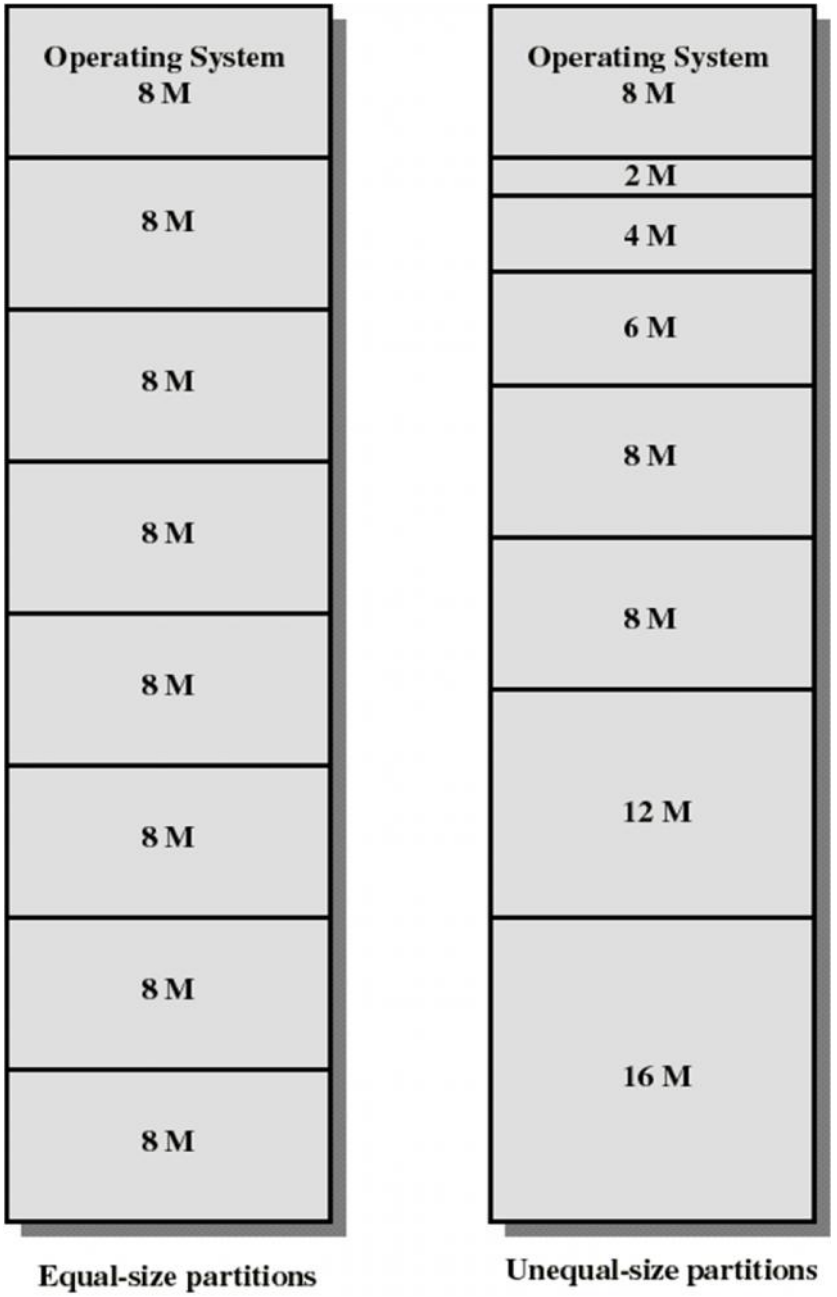
- Introduction
- Fixed Partitioning
- Variable Partitioning
- Memory Hole/Allocation management
- Problems

Introduction

- Techniques to manage Main memory efficiently
- Provides multitasking facility
- Responsible for memory allocation, de-allocation and keep track for each location
- Basic techniques:
  - Fixed/Static Partitioning
  - Variable/Dynamic Partitioning

Fixed Partitioning

- Partition of main memory into a set of non-overlapping memory regions called partitions
- Fixed partitions can be of equal or unequal sizes
- Leftover space in partition after program assignment is called internal fragmentation
- Uses overlaying technique



Types of Fixed Partitions

Equal-size partitions

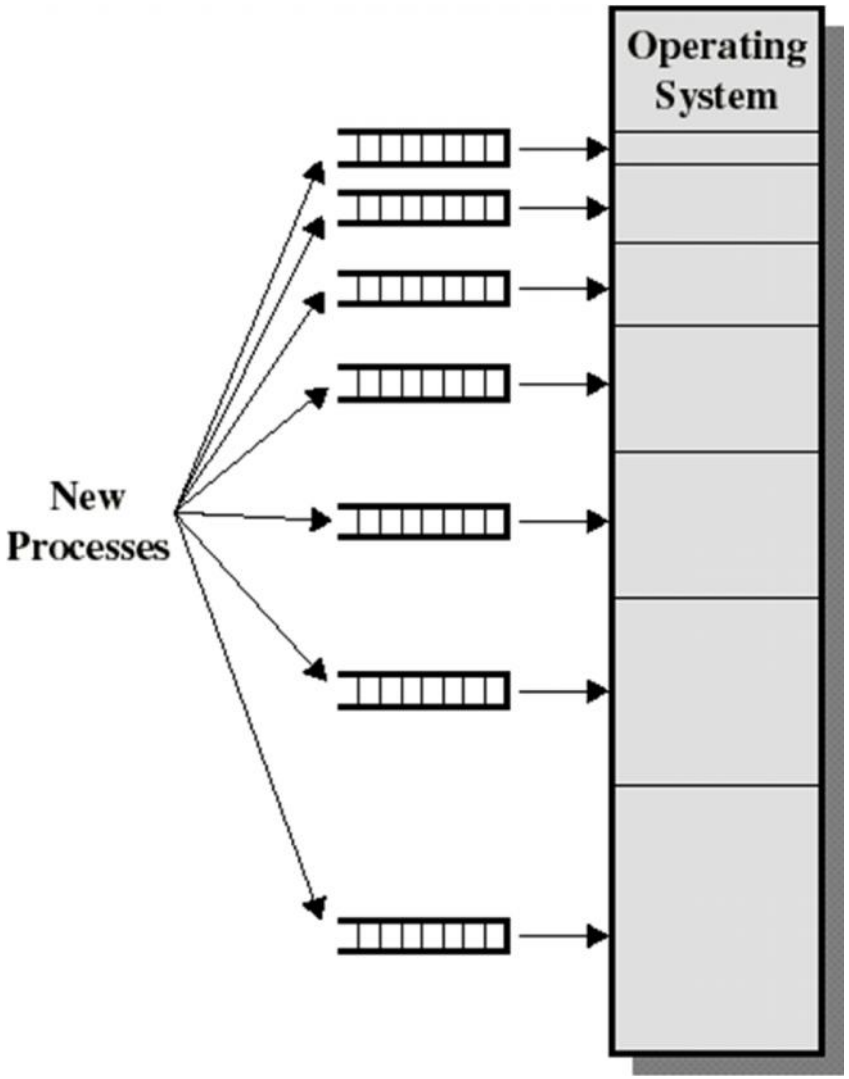
- If there is an available partition, a process can be loaded into that partition –
  - because all partitions are of equal size, it does not matter which partition is used.
- Doesn’t require any allocation policy
- Fast and easy
- Result internal fragmentation

Types of Fixed Partitions

Unequal-size partitions

(Multiple queues)

- assign each process to the smallest partition within which it will fit.
- a queue exists for each partition size.
- tries to minimize internal fragmentation.
- Some queues might be free while other might be loaded



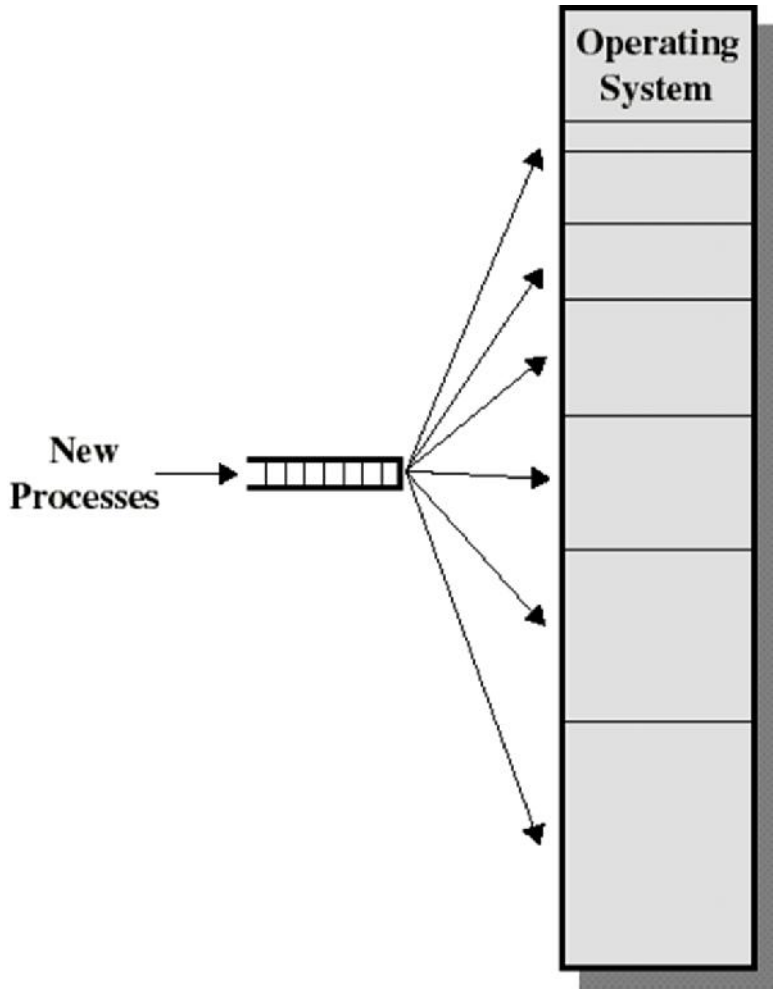
Continued

Types of Fixed Partitions

Unequal-size partitions

(single queue)

- The smallest available partition that will hold the process is selected.
- increases the level of multiprogramming at the expense of internal fragmentation.



Continued

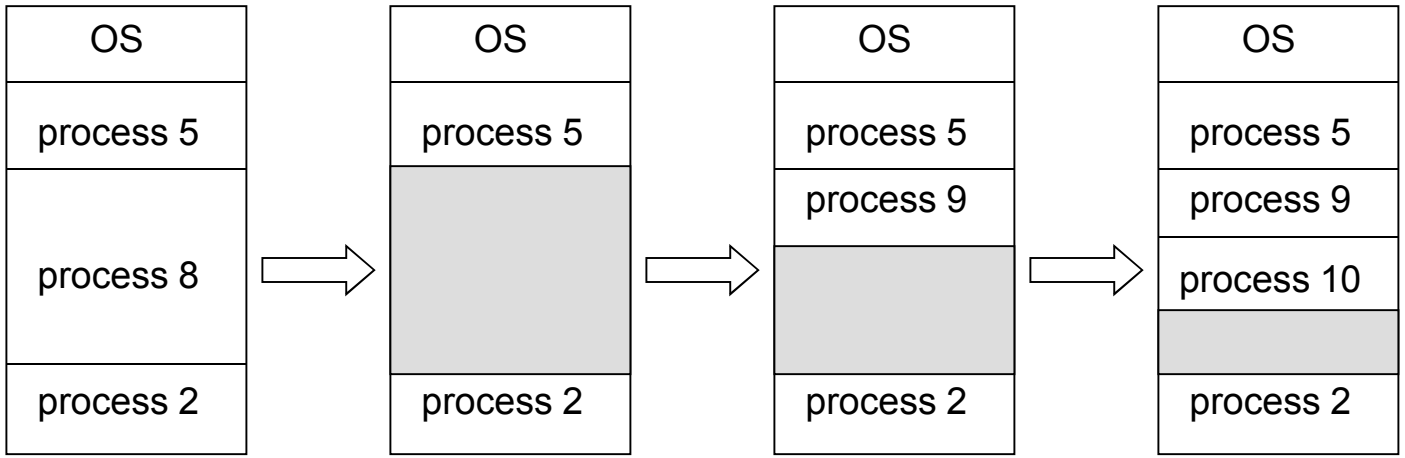
Problems in Fixed Partitioning

- Main memory use is inefficient.
- A program occupies an entire partition which can cause internal fragmentation.
- Unequal-size partitions reduces the problem little bit
- Equal-size partitions was used in early IBM's OS/MFT (Multiprogramming with a Fixed number of Tasks).

Continued

Variable Partitioning

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



Internal/External Fragmentation

There are really two types of fragmentation:

1. **Internal Fragmentation**
   - Allocated memory may be slightly larger than requested memory
   - The size difference in memory internal to a partition, but not being used.
1. **External Fragmentation**
   - Memory space left between two or more allocated processes
   - External fragmentation exist in dynamic memory divisions

Reducing External Fragmentation

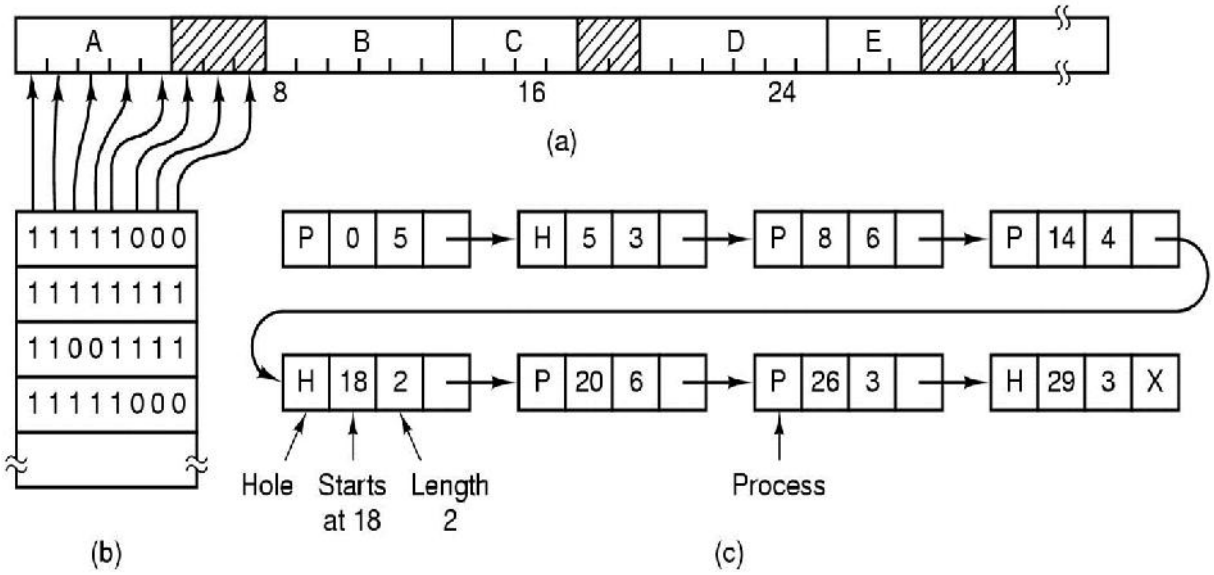
Compaction

- Shuffle memory contents to place all free memory together in one large block (or possibly a few large ones).
- Only if relocation is dynamic and is done at execution time.
- Problem:
  - Disturbs processes those are communicating with I/O
  - Addresses should be resolved after the compaction

Memory Hole/Allocation Management

- It is required to keep track of used and unused memory spaces
- Techniques:
  - **Bit-map:** It uses a table where 0 represent empty spaces & 1 represents used spaces.
  - **Linked List:** Every node contains 4 fields, 1st to show Process (P) or Memory-Hole (H), 2nd to show starting address of P/H, 3rd to present length of P/H and 4th field contains address of next node

- Figure a shows the used/unused memory spaces
- Figure b shows swapping management using Bit-map
- Figure c shows swapping management using Linked List

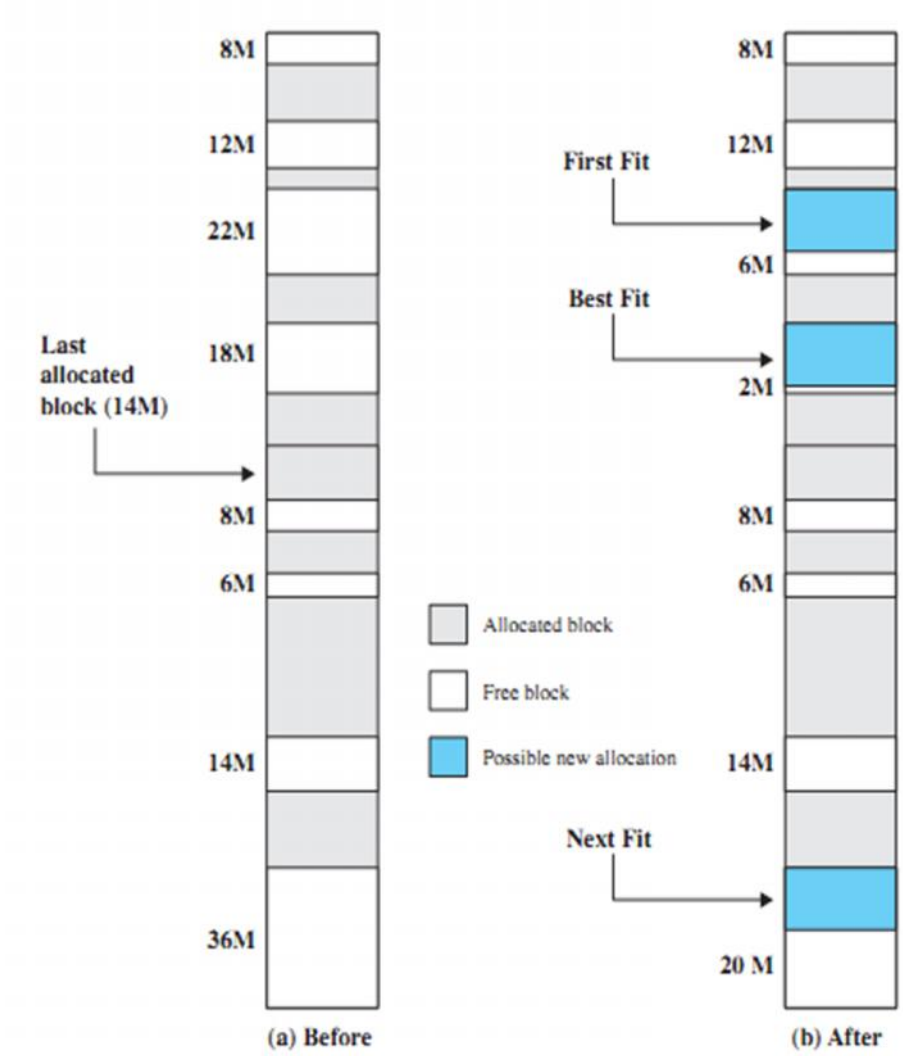


Placement Algorithms

- Methods for process allocation in fixed (unequal) and dynamic memory management techniques:
  - First-fit: It favors allocation near the beginning where a process can accommodate.
  - Next-fit: It works like first fit, but it starts searching of memory-hole from last point it has used.
  - Best-fit: It searches whole list & find the smallest memory-hole where process can fit.
  - Worst-fit: It searches whole list & fine the largest memory-hole to allocate a process.
  - Buddy System: dynamic memory allocation w.r.t requested memory size

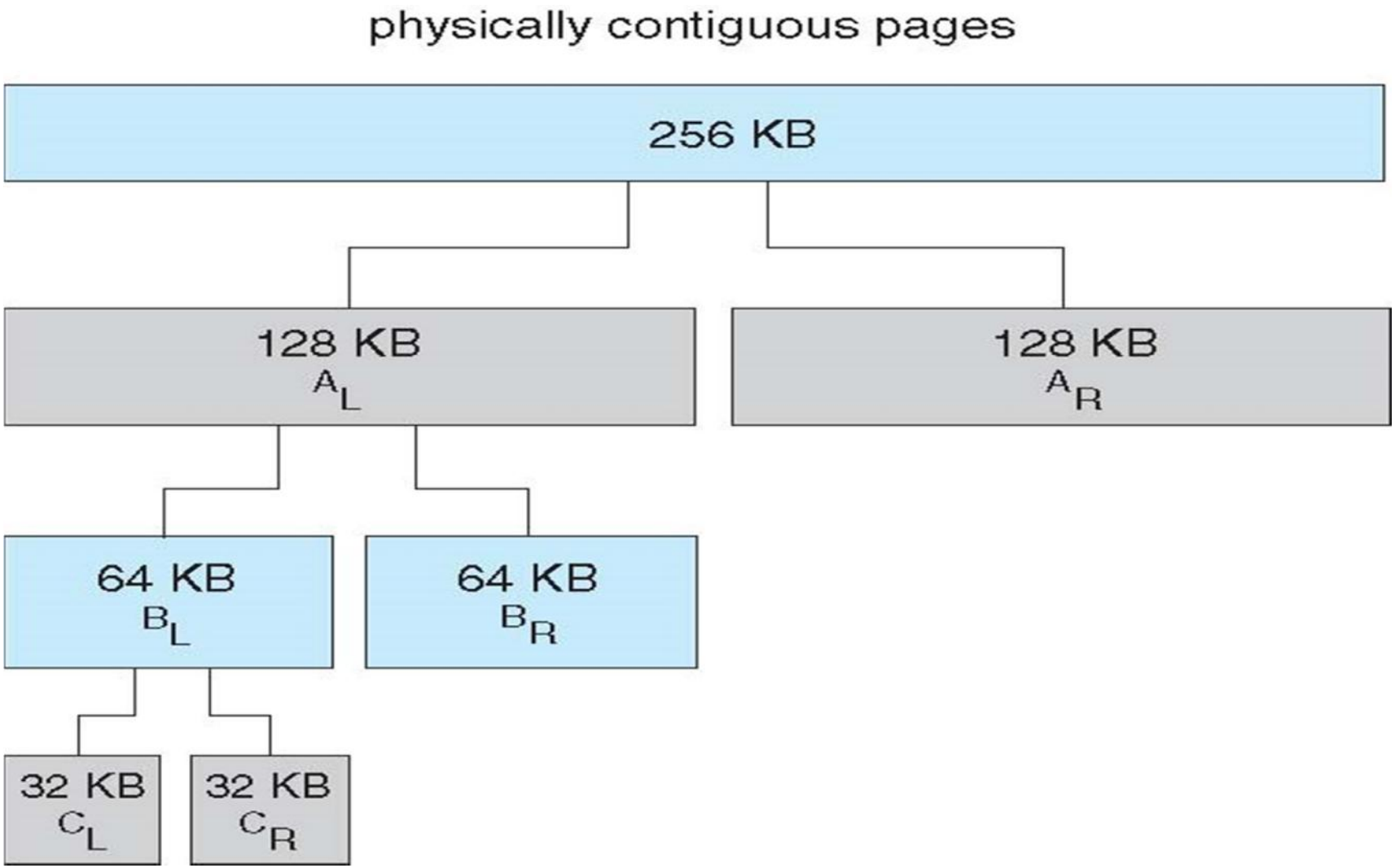
Placement Algorithms

- Used to decide which free block to allocate to a process of 16MB.
- Goal: reduce usage of compaction procedure (its time consuming).
- Example algorithms:
  - First-fit
  - Next-fit
  - Best-fit
  - Worst-fit

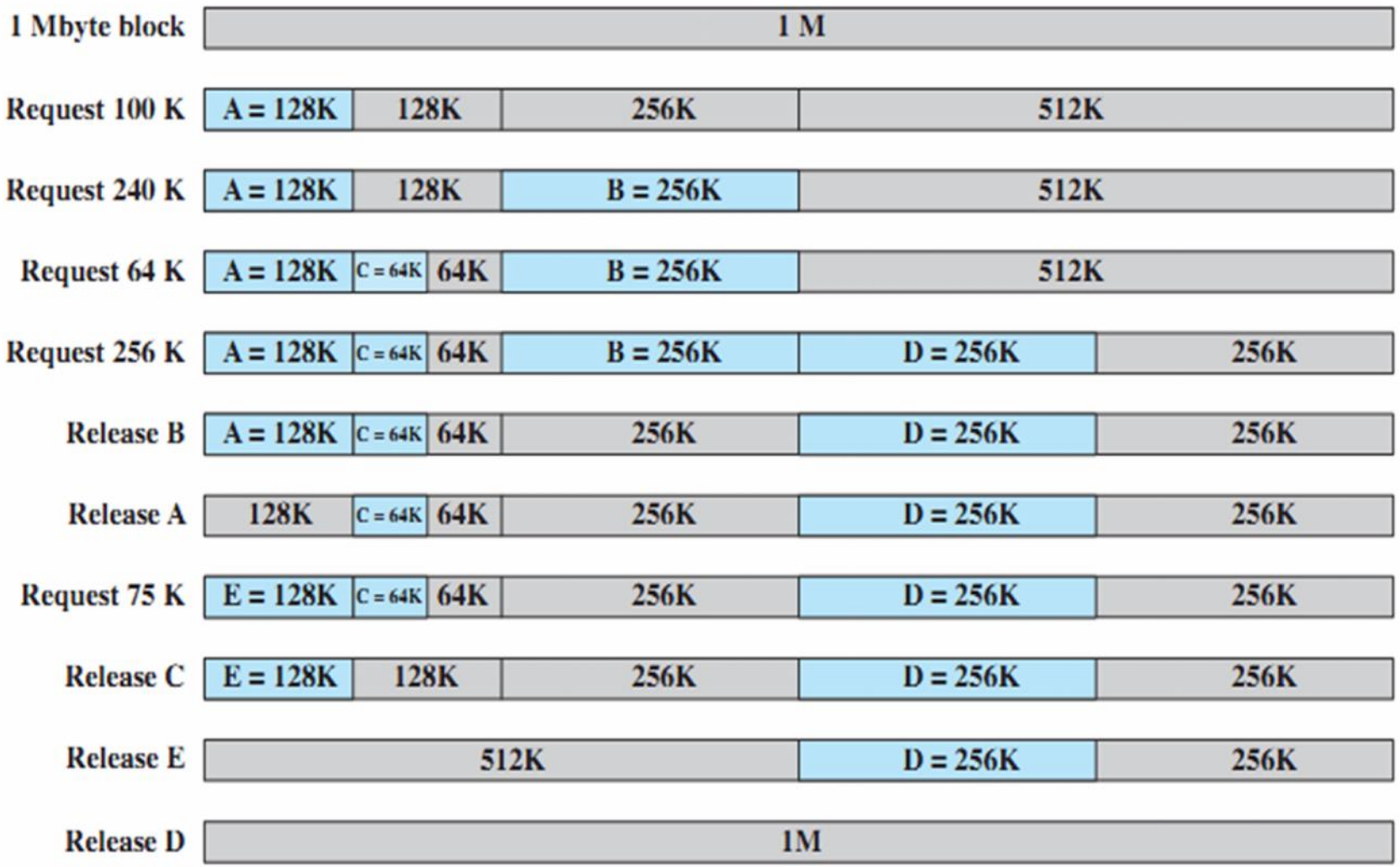


Continued

Buddy System Allocation



Example of Buddy System



Problems

Question # 1

Consider a swapping system in which memory consists of the following hole sizes in memory order: 10K, 4K, 20K, 18K, 7K, 9K, 12K and 15K. Which hole is taken for successive segment requests of (a) 12K, (b) 10K, (c) 9K for first fit? Repeat the question for best fit, worst fit and next fit.

(a) 1st fit

12K, 10K and 9K will be allocated in Block # 5, 4, 3 respectively

(b) Best fit

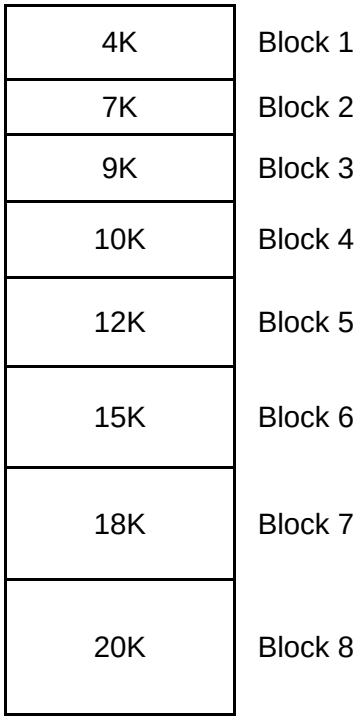
12K, 10K and 9K will be allocated in Block # 5, 4, 3 respectively

(c) Worst fit

12K, 10K and 9K will be allocated in Block # 8, 7, 6 respectively

(d) Next fit

12K, 10K and 9K will be allocated in Block # 5, 6, 7 respectively



Problems

Question # 2

A computer uses the buddy system for memory management. Initially it has one block of 256K at address 0. After successive requests for 5K, 25K, 35K and 20K come in, how many blocks are left and what are their sizes and addresses?

Continued

Problems

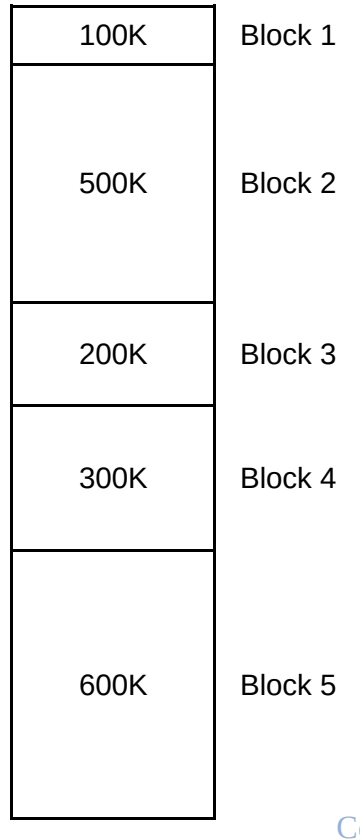
Question # 3

Given 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 624K (in order)? Which algorithm makes the most efficient use of memory?

First fit

Best fit

Worst fit



Continued

Questions

OPERATING SYSTEMS

Simple/Basic Paging

Introduction

Paging mechanism

Translation Look-aside buffer

Sharing & Protection

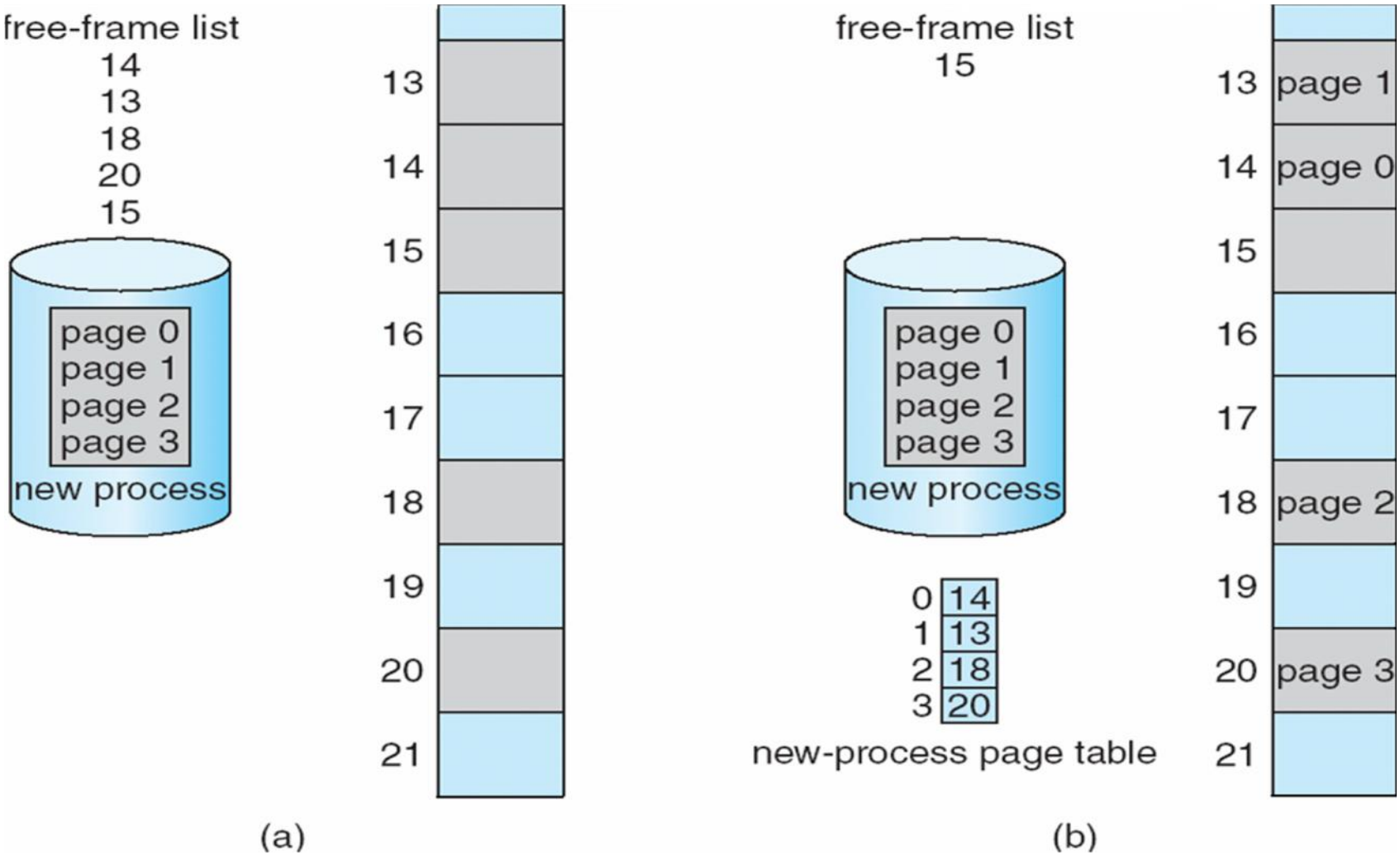
Simple/Basic Paging

- Divide physical memory into fixed-sized chunks/blocks called **frames**.
- Divide logical memory into blocks of same size **pages**.
- The process pages can be assigned to any free frames in main memory.
- Pages can be allocated in noncontiguous frames

Simple/Basic Paging Requirements

- Need to keep track of all free frames.
- To run a program of size n pages, need to find n free frames and load program.
- Need to set up a page table to translate logical to physical pages/addresses.
- Internal fragmentation possible only for the last page
- No external fragmentation possible

Page-Table & Free-Frame list

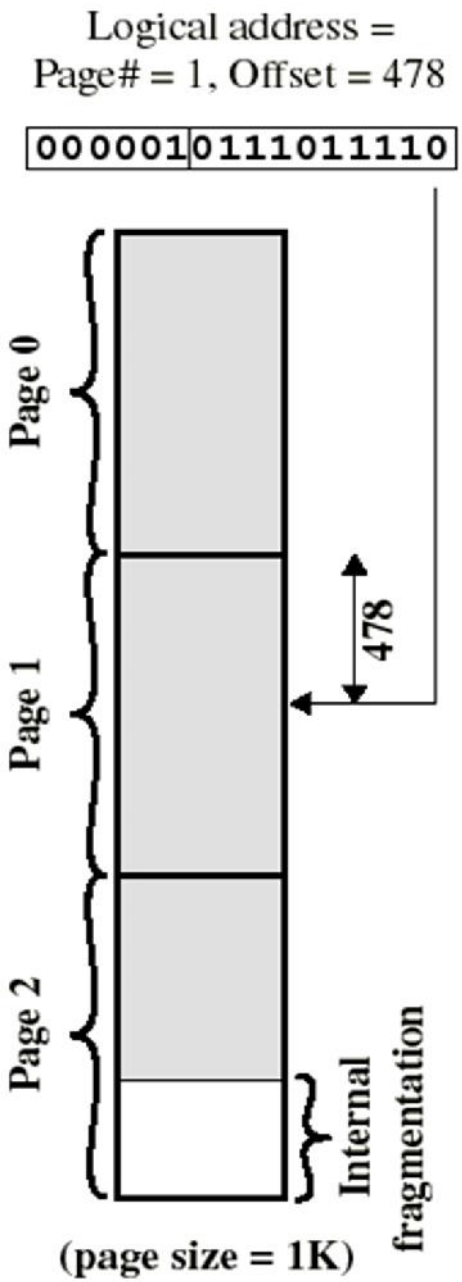


Before allocation

After allocation

Logical address in paging

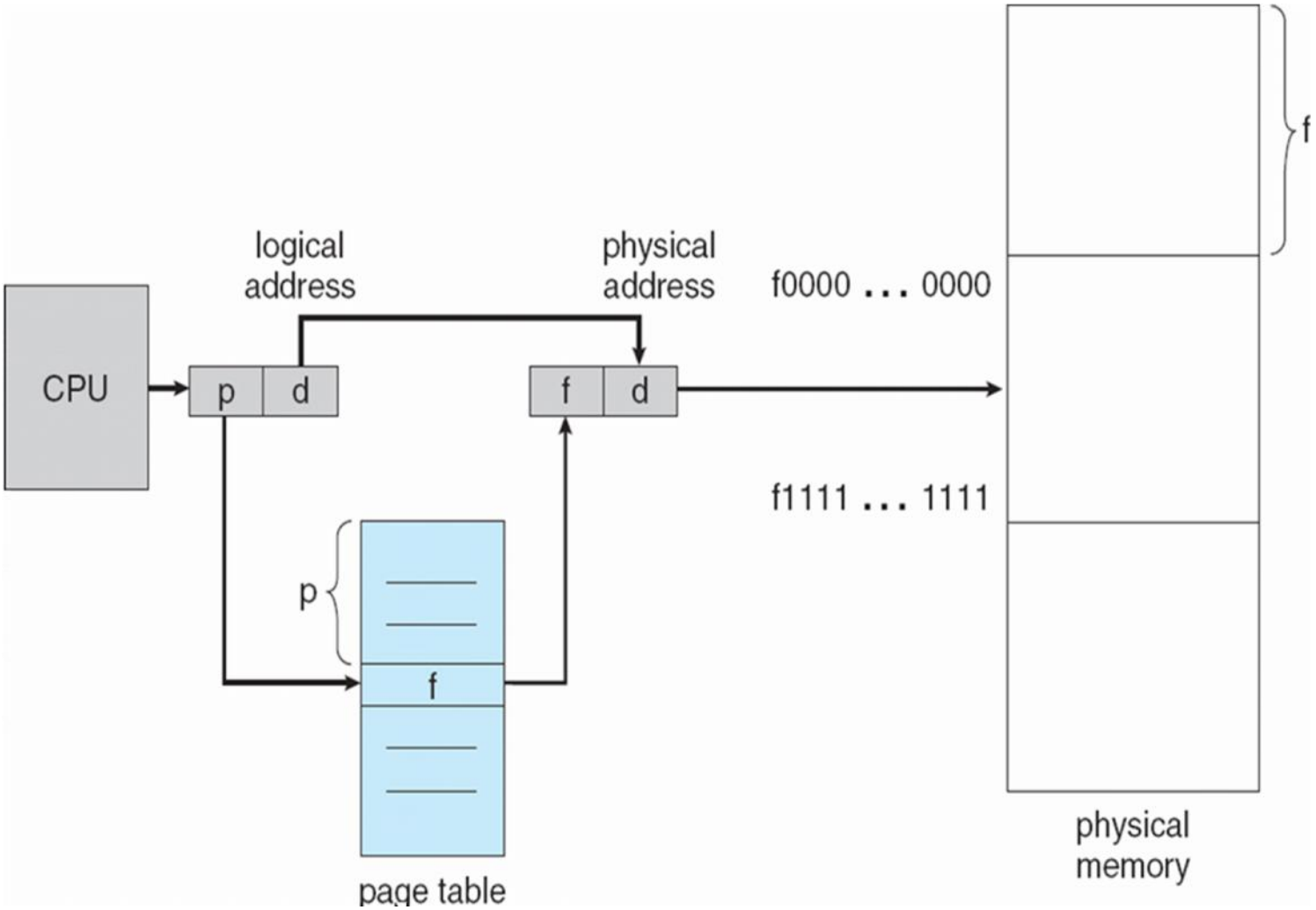
- The logical address becomes a relative address.
- If16 bits addresses are used with page size = 1K, then 10 bits for offset and 6 bits available for page number.
- Address translation from logical to physical is done with help of page table
- page table helps to obtain the physical address (frame number, offset).



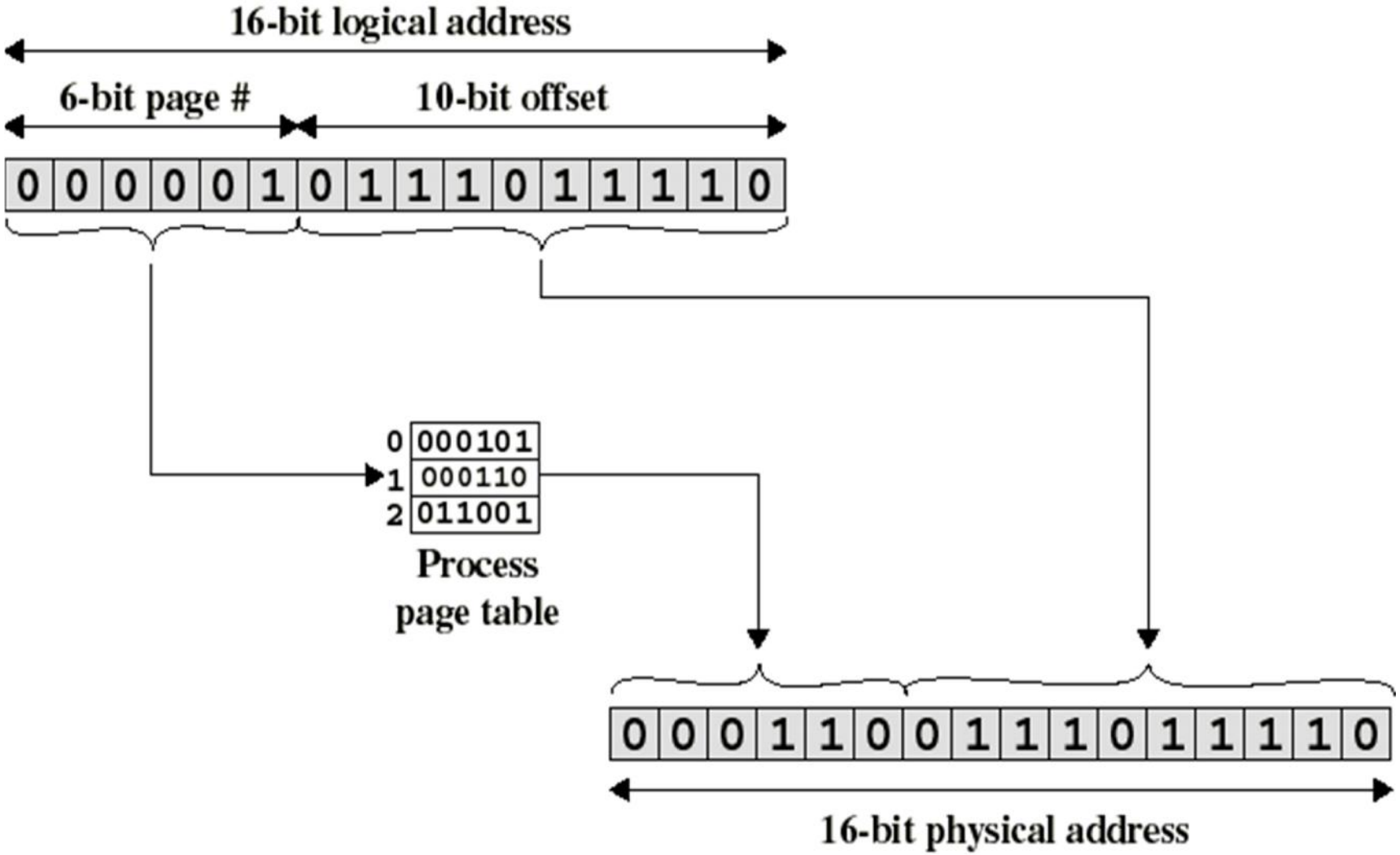
Address Translation Scheme

- Logical address generated by CPU is divided into two parts:
  - *Page number (p)* – used as an index into a *page table* which contains the base address of each page in physical memory.
  - *Page offset/displacement (d)* – combined with base address to define the physical memory address
- By using a page size of a power of 2, the pages are invisible to the programmer, compiler/assembler, and the linker.
- Address translation at run-time is then easy to implement in hardware

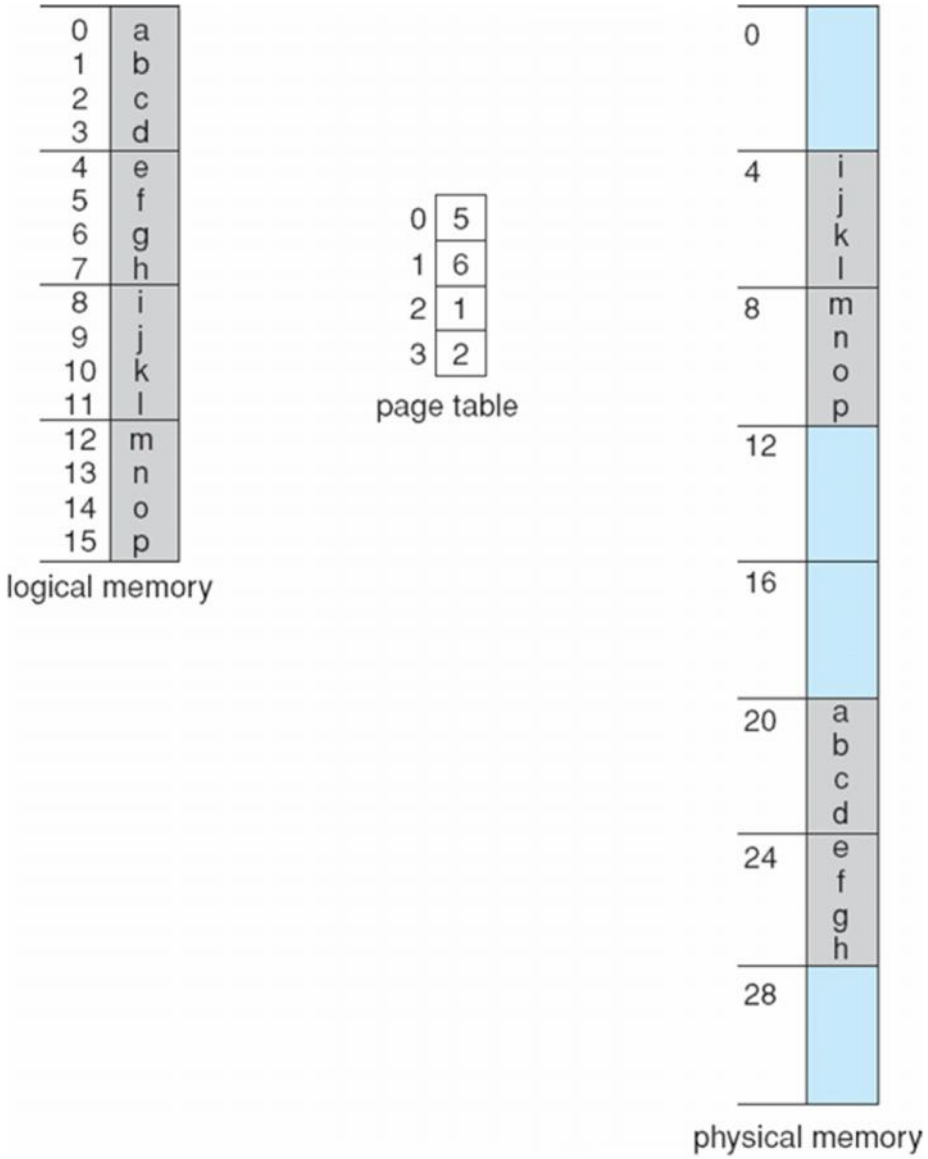
Address Translation Architecture



Logical-to-Physical Address Translation in Paging



Paging Example



Implementing Page Table

1. Keep Page Table in main memory:
   - Page-table base register (PTBR) points to the page table.
   - Page-table length register (PTLR) indicates size of the page table.
   - Every data/instruction access requires two memory accesses
     - one for the page table
     - one for the data/instruction.
2. Keep Page Table in hardware (in MMU)
   - Expensive if page table is large
   - Faster but costly

Implementing Page Table

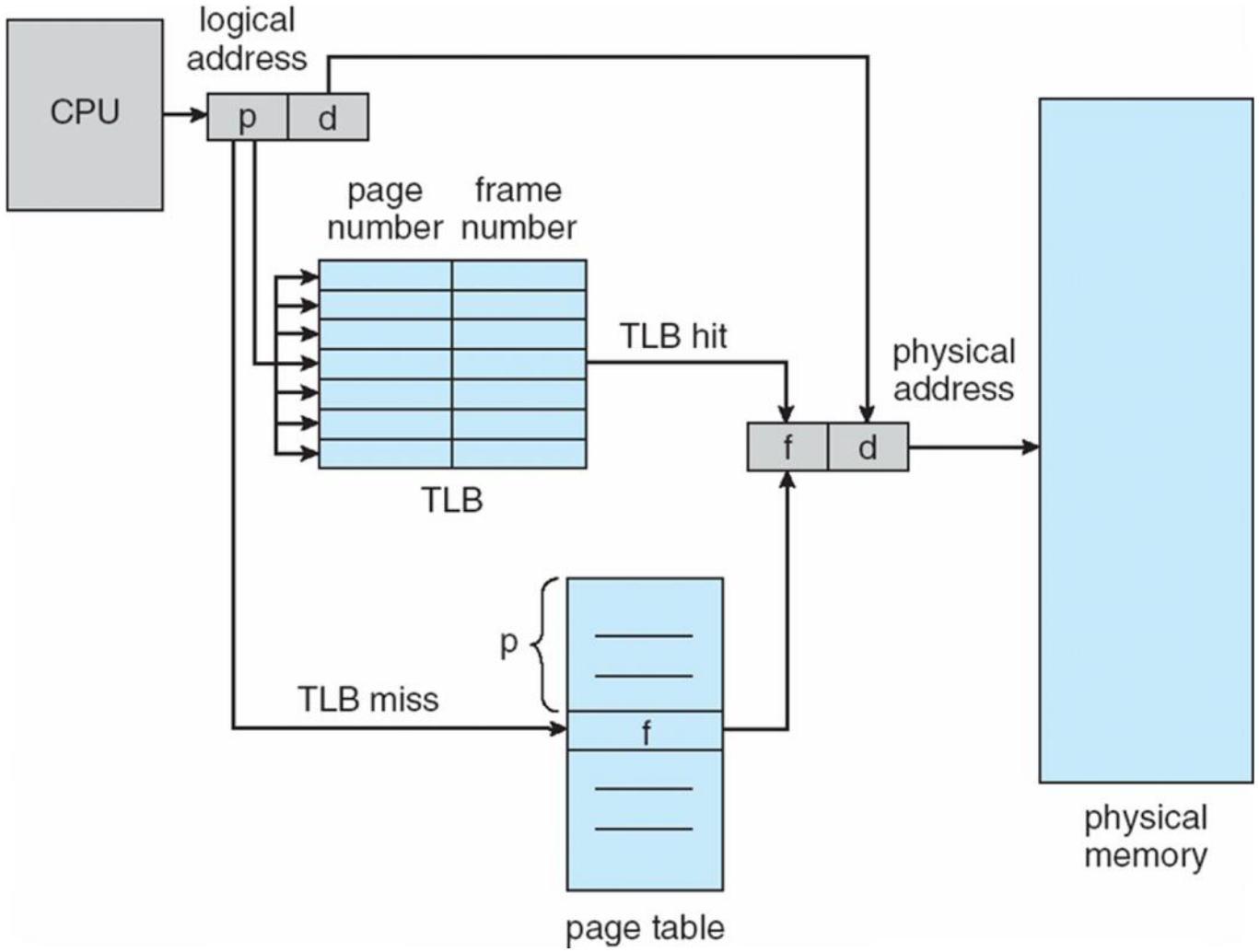
3. Combination:
   - Special fast-lookup hardware cache called *Associative Memory* (*Registers*) or *Translation Look-aside Buffer (TLB)*
   - enables fast parallel search:

Page #	Frame #

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

Continued

Paging Hardware With TLB



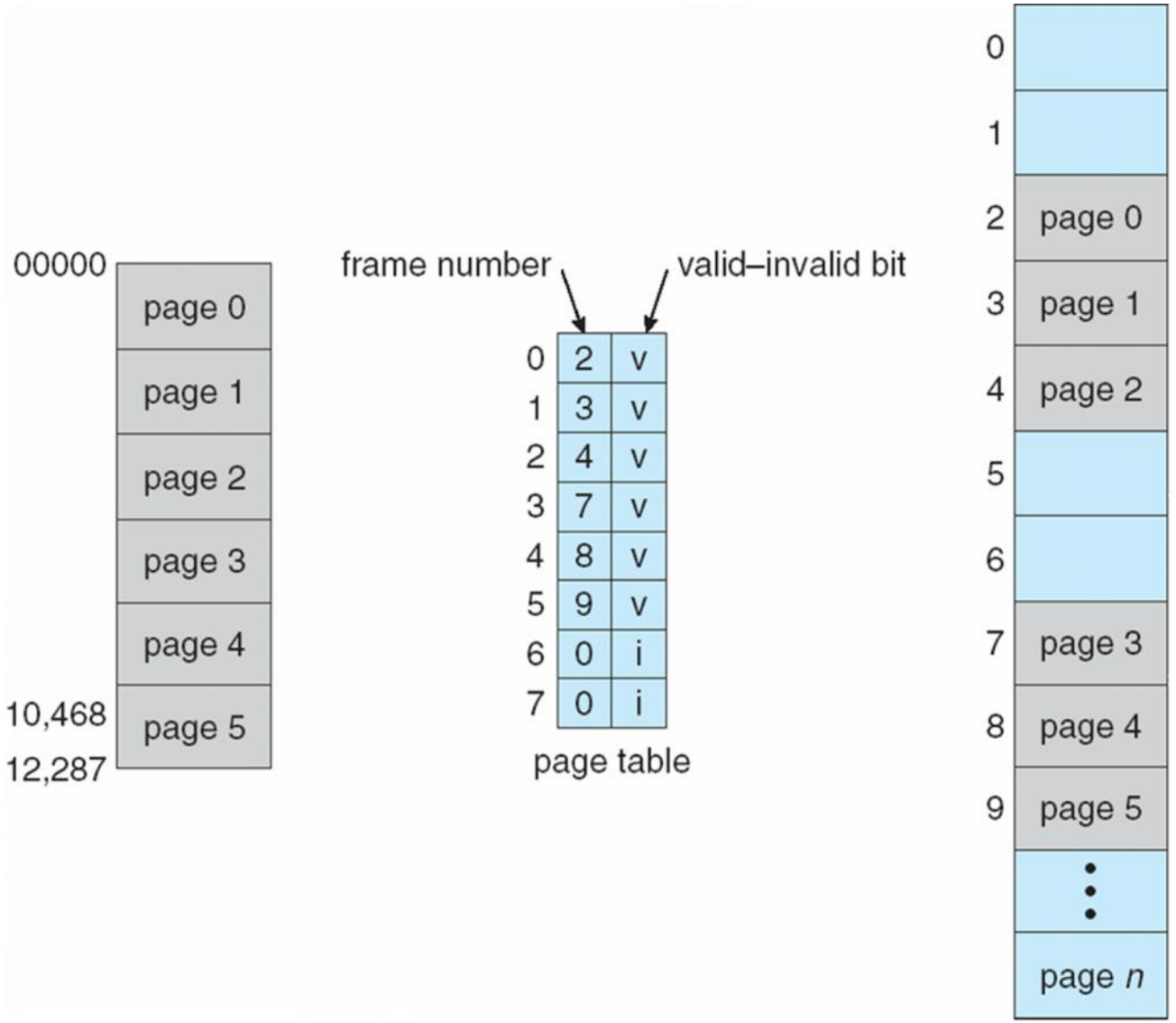
Importance of TLB

- TLB takes advantage of the Locality Principle.
- Associative mapping hardware to simultaneously interrogate all TLB entries to find a match/hit on page number.
- TLB hit rates are 90+%.
- TLB must be flushed each time a new process enters the running state.
- Requires to keep/load TLB information in/from process context.

Memory Protection

- Memory protection implemented by associating a 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.

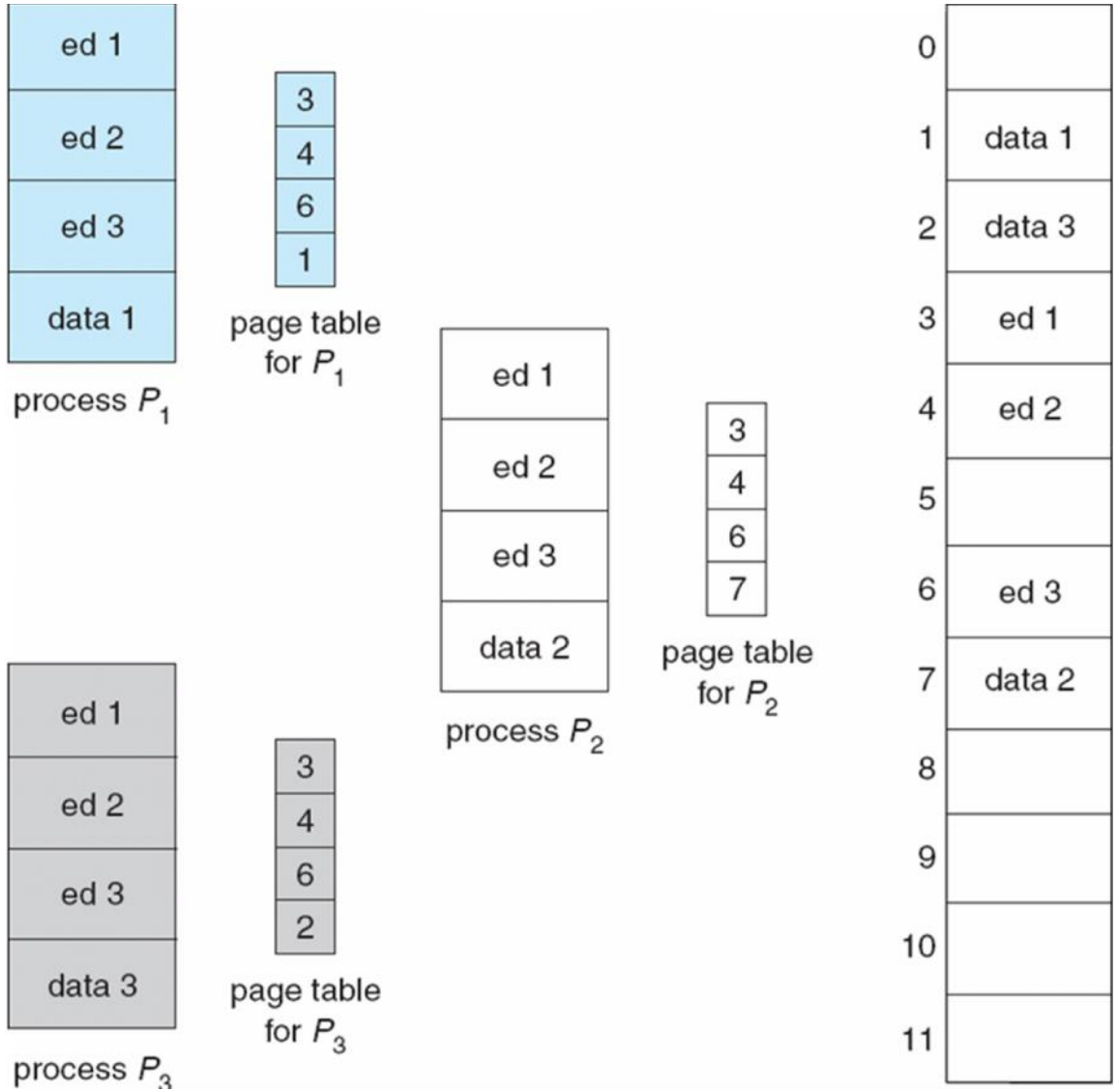
Valid (v) or Invalid (i) Bit in a Page Table



Shared Pages

- Shared code:
  - One copy of read-only 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 separate copy of code and data.
  - The pages for the private code and data can in the logical address space.

Shared Pages Example



Continued

Questions

OPERATING SYSTEMS

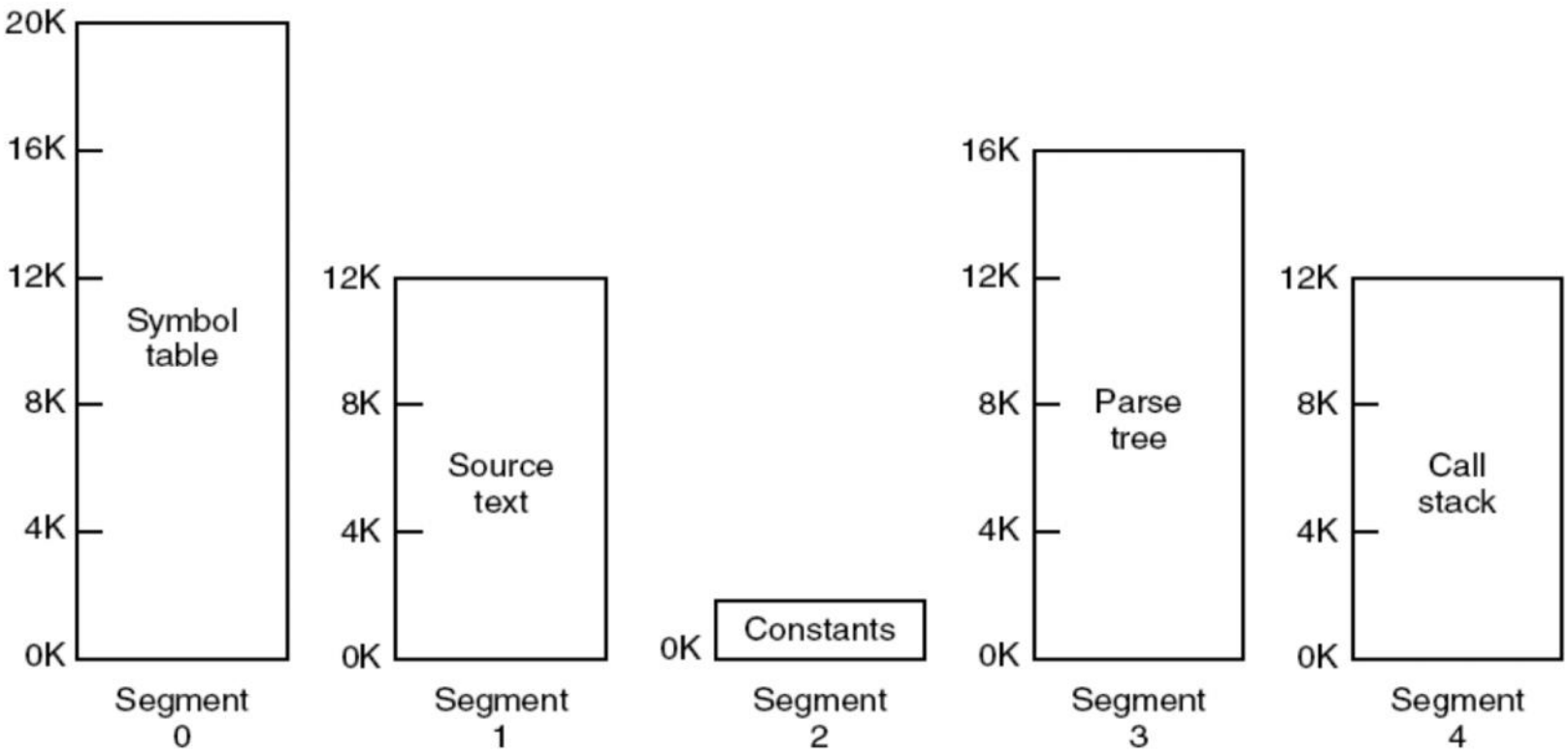
Simple/Basic Segmentation

- Introduction
- Segmentation mechanism
- Sharing & Protection
- Comparison with paging

Simple/Basic Segmentation

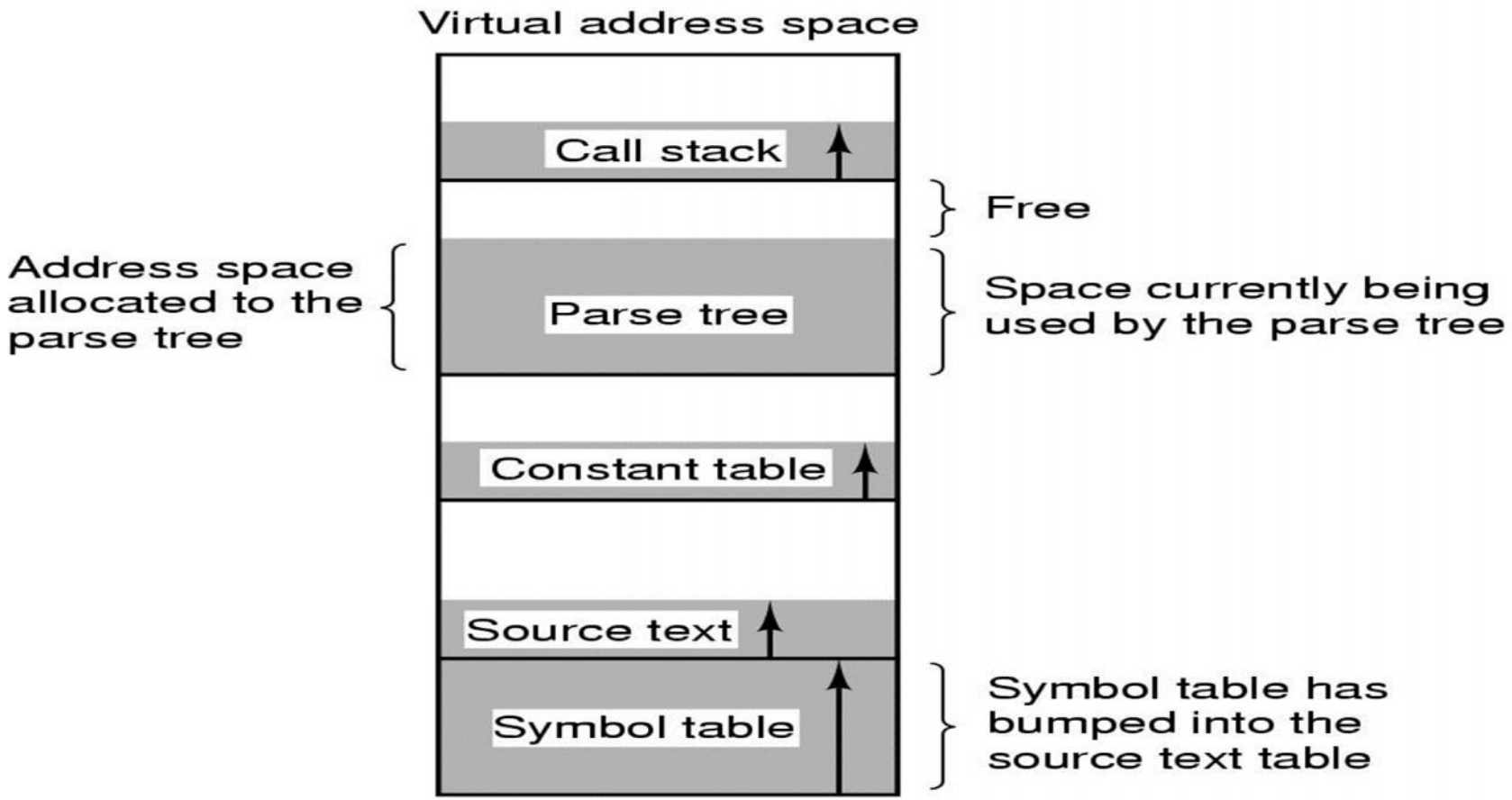
- Paging division is arbitrary; no natural/logical boundaries for protection/sharing.
- Segmentation supports user's view of a program.
- A program is a collection of segments (logical units)
- A compiler has many tables that are built up as compilation proceeds, possibly including:
  - The source text (Code segment)
  - The symbol table – the names and attributes of variables.
  - The table containing integer, floating-point constants used.
  - The parse tree, the syntactic analysis of the program.
  - The stack used for procedure calls within the compiler.

Segmentation Feature



Segmentation allows each table to grow or shrink independently of the other tables.

Segmentation solution

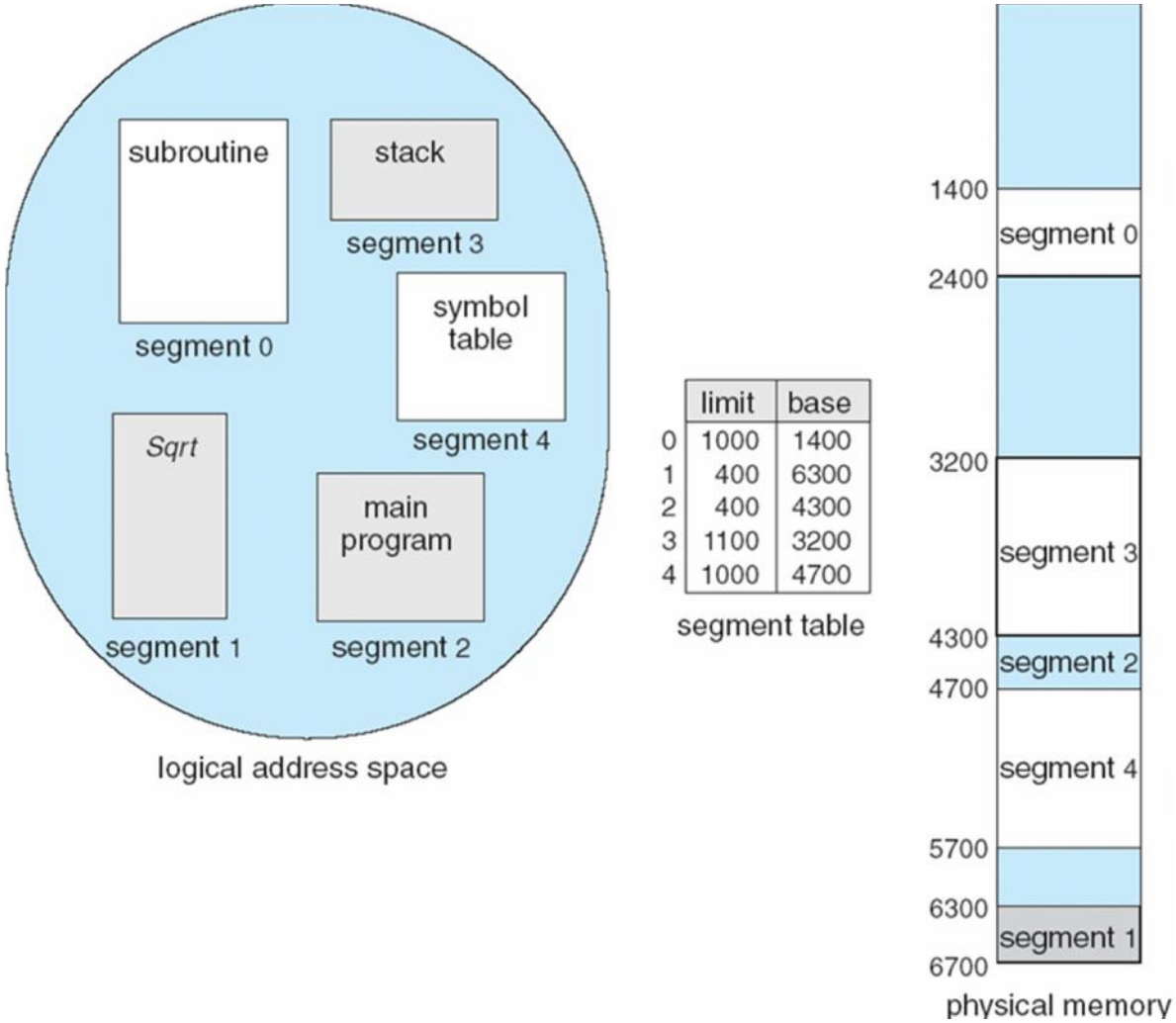


In a one-dimensional address space with growing tables, one table may bump into another.

Simple Segmentation properties

- Each program is subdivided into blocks of non-equal size called segments.
- When a process gets loaded into main memory, its different segments can be located anywhere.
- Each segment is fully packed with instructions/data; no internal fragmentation.
- There is external fragmentation; it is reduced when using small segments.
- In contrast with paging, segmentation is visible to the programmer:
  - provided as a convenience to organize logically programs (example: data in one segment, code in another segment).
  - must be aware of segment size limit.
- The OS maintains a segment table for each process. Each entry contains:
  - the starting physical addresses of that segment.
  - the length of that segment (for protection).

Example of Segmentation

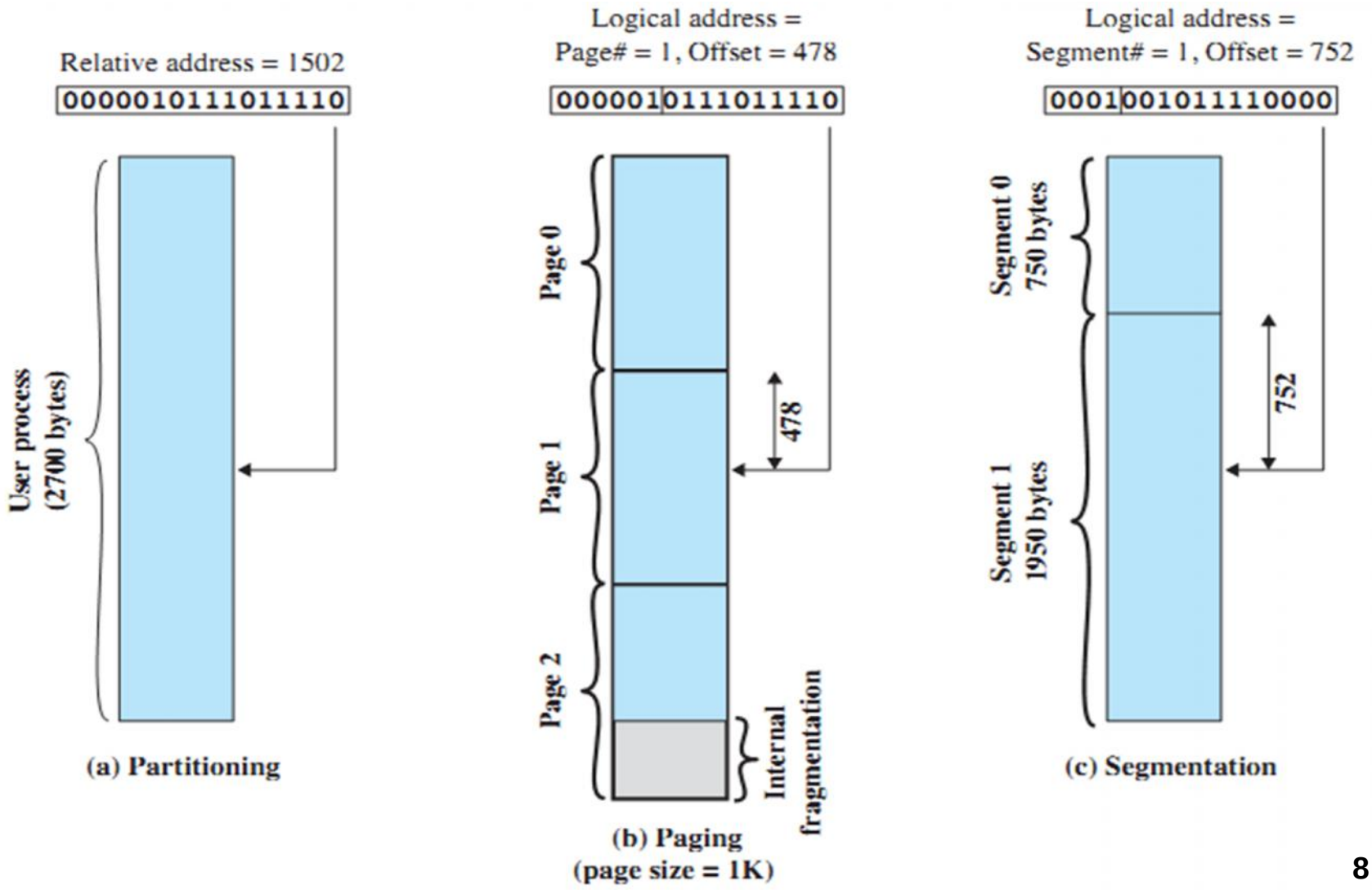


Implementing Segmentation

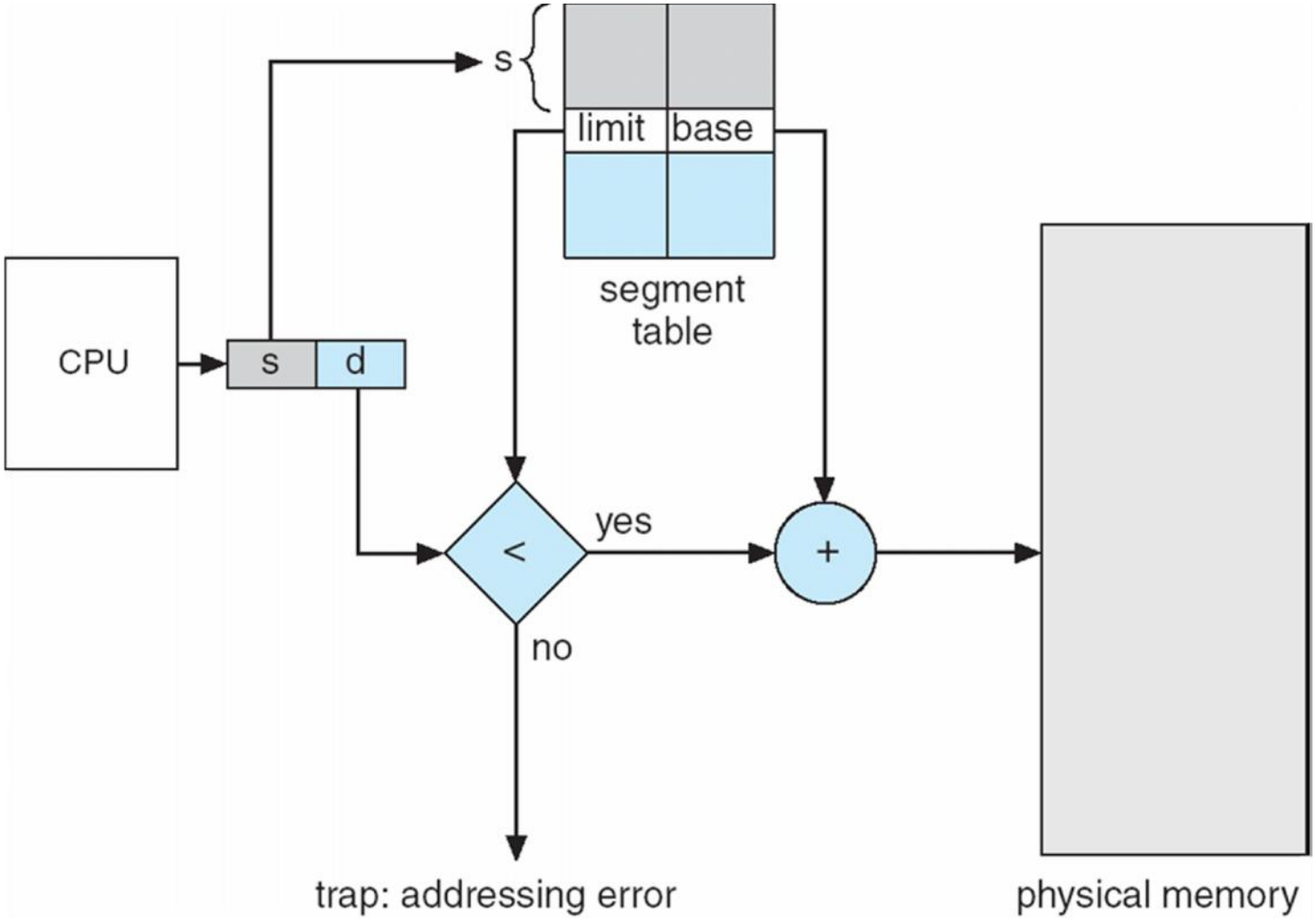
- Logical address consists of a two fields:

  <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* < STLR.

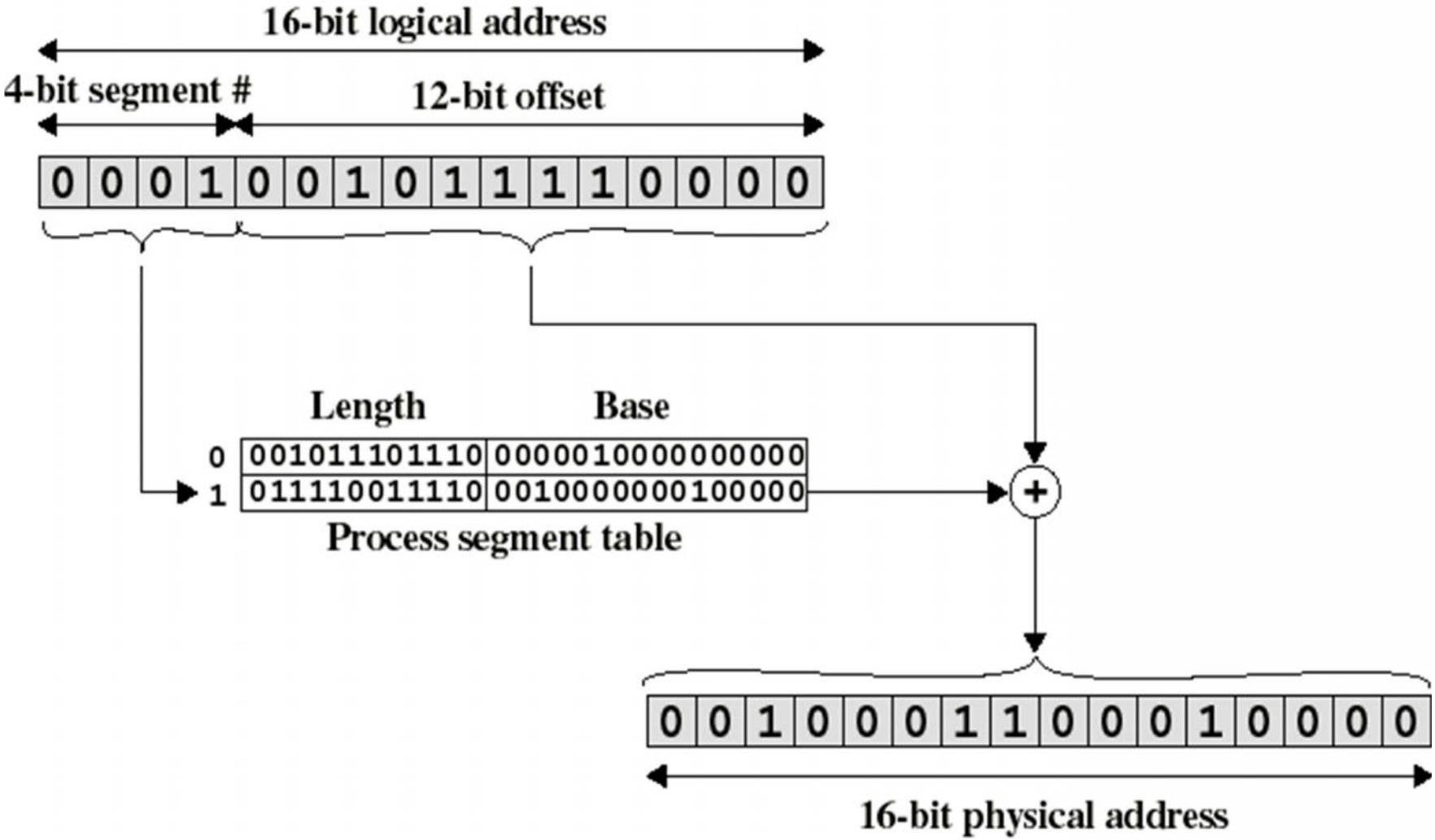
Logical address in segmentation



Address Translation Architecture



Logical-to-Physical Address Translation in segmentation



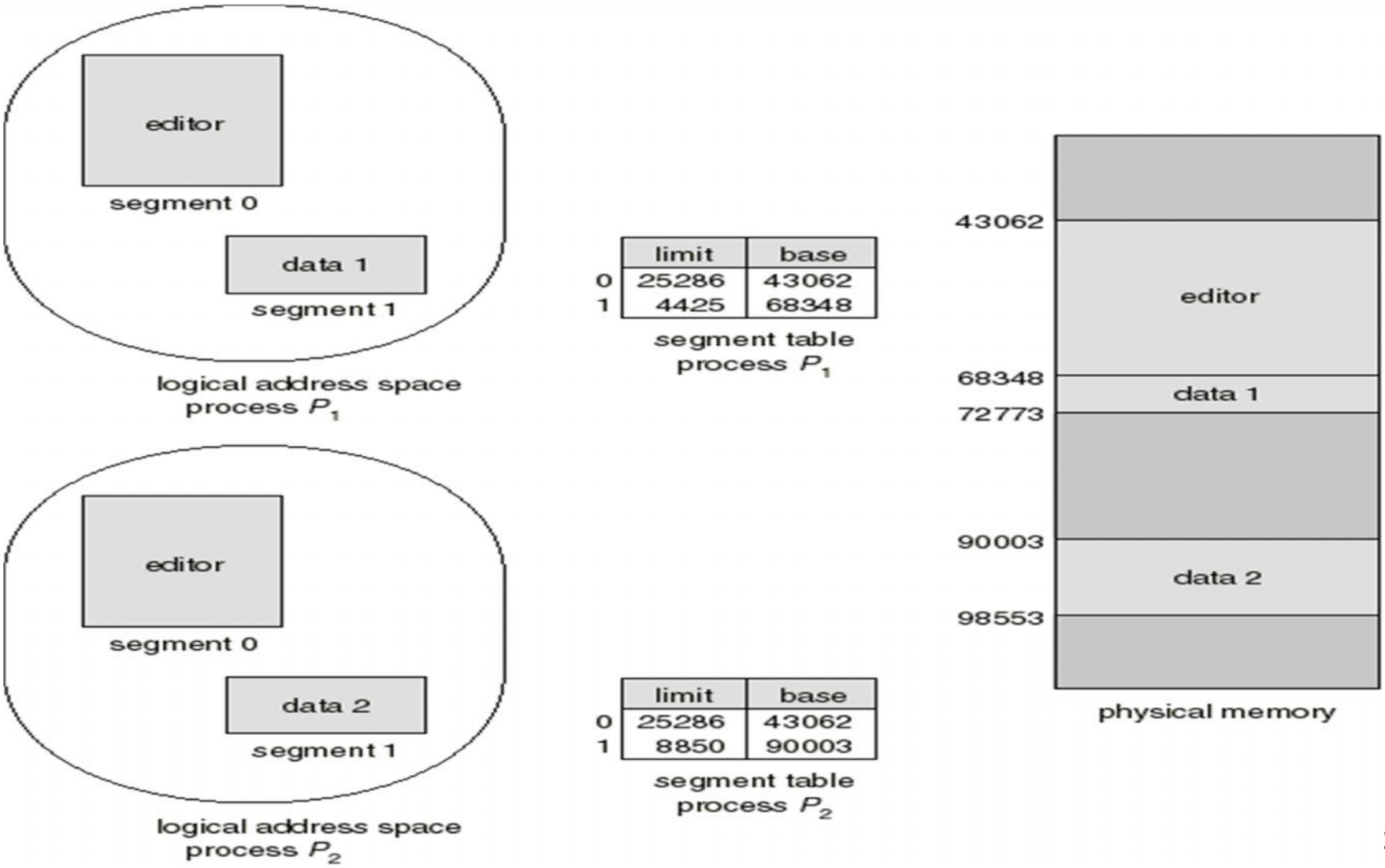
Protection in Segmentation

- Protection – with each entry in segment table associate:
  - validation bit = 0 ⇒ 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.

Sharing in Segmentation Systems

- Segments are shared when entries in the segment tables of 2 different processes point to the same physical locations.
- Example: the same code of a text editor can be shared by many users:
  - Only one copy is kept in main memory.
- But each user would still need to have its own private data segment.

Shared Segments Example



Simple segmentation/paging comparison

- Segmentation is visible to the programmer whereas paging is transparent.
- Naturally supports protection/sharing.
- Supports logical organization of program into segments while using different kinds of protection (example: execute-only for code but read-write for data).
- Segments are variable-size; Pages are fixed-size.
- Segmentation requires more complicated hardware for address translation than paging.
- Segmentation suffers from external fragmentation. Paging only yields a small internal fragmentation.
- Combine Segmentation and Paging for suitable outcome

Questions