Main Memory (II)

Operating Systems

Yu-Ting Wu

(with slides borrowed from Prof. Jerry Chou)

Outline

- Paging (cont.)
- Segmentation
- Segmentation with Paging

Paging (cont.)

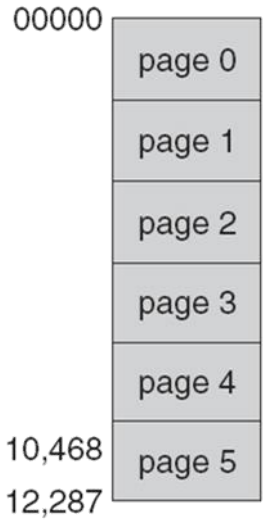
Memory Protection

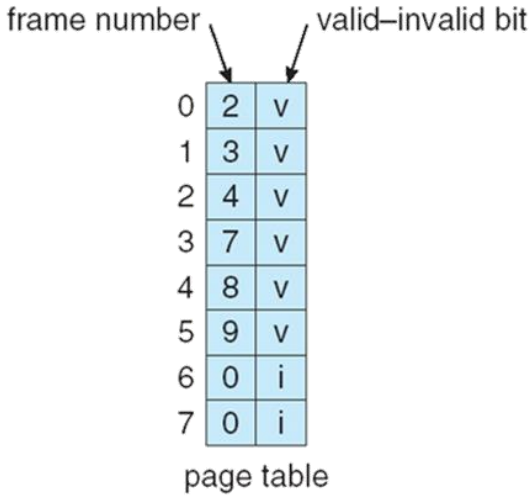
- Each page is associated with a set of **protection bit** in the page table
  - E.g., a bit to define read/write/execution permission

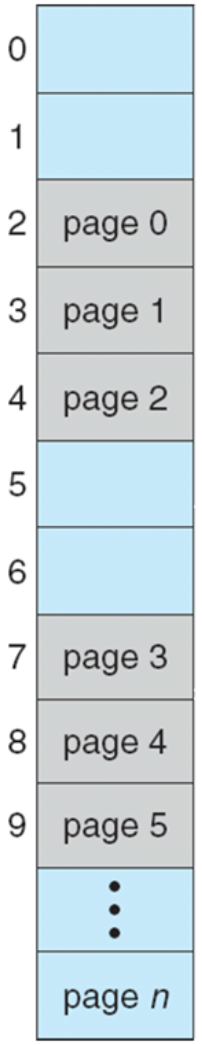
- Common use: **valid-invalid bit**
  - **Valid:** the page / frame is in the process' logical address space, and is thus a legal page
  - **Invalid:** the page / frame is not in the process' logical address space (e.g., virtual memory)

Valid-Invalid Bit Example

- Potential issues
  - Un-used page entry cause memory waste
    ➔ **use page table length register (PTLR)**
  - Process memory may not be on the boundary of a page
    ➔ memory limit register is still needed





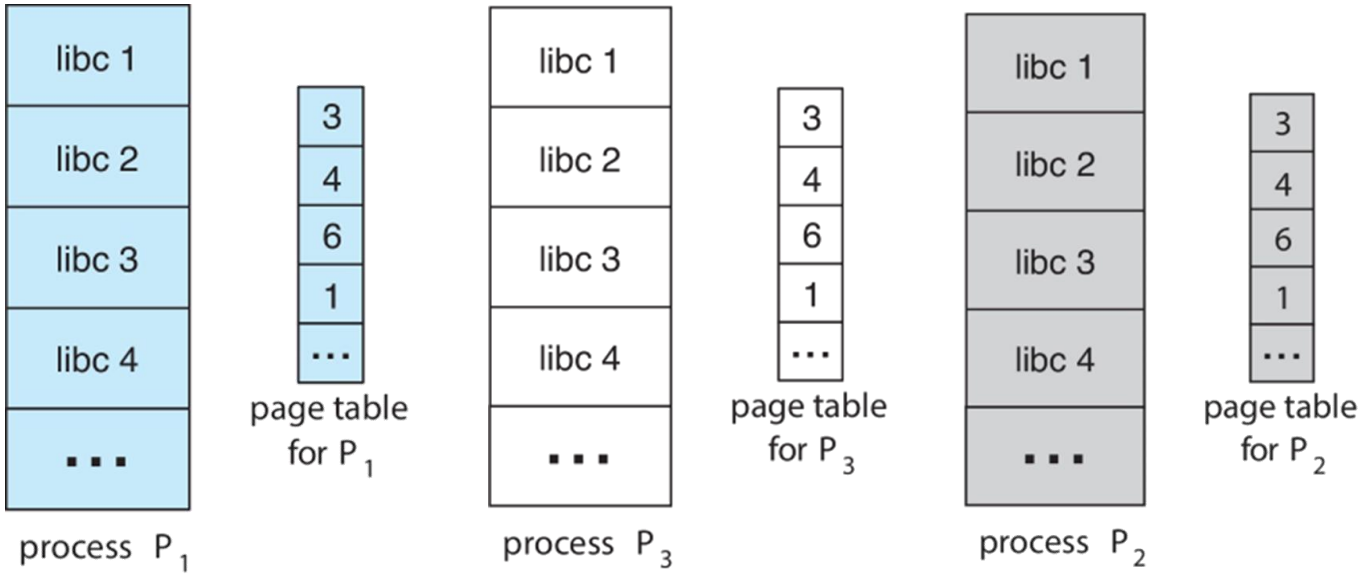


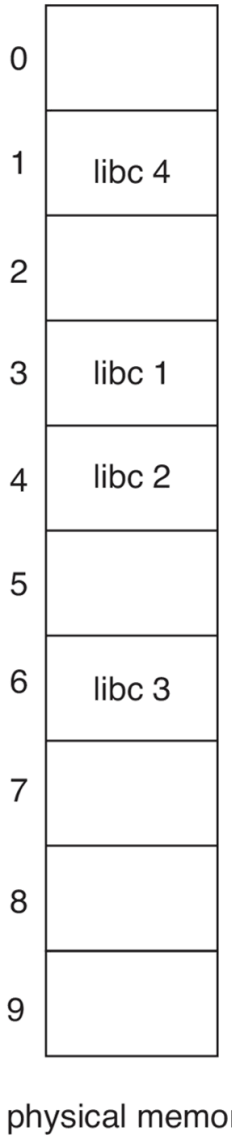
Shared Pages

- Paging allows processes **share** common code, which must be **reentrant**
- Reentrant code (pure code)
  - It never change during execution
  - Text editors, compilers, web servers, etc.
- **Only one copy** of the shared code needs to be kept in physical memory
- **Two (several) virtual addresses** are mapped to one physical address

Shared Pages by Page Table

- Shared code must appear in the same location in the logical address space of all processes





Page Table Memory Structure

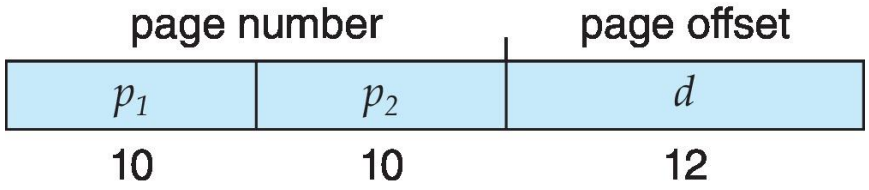
- Page table could be huge and difficult to be loaded
  - 4GB (2^{32}) logical address space with 4KB (2^{12}) page

  ➔ 1 million (2^{20}) page table entry
  - Assume each entry need 4 bytes (32 bits)

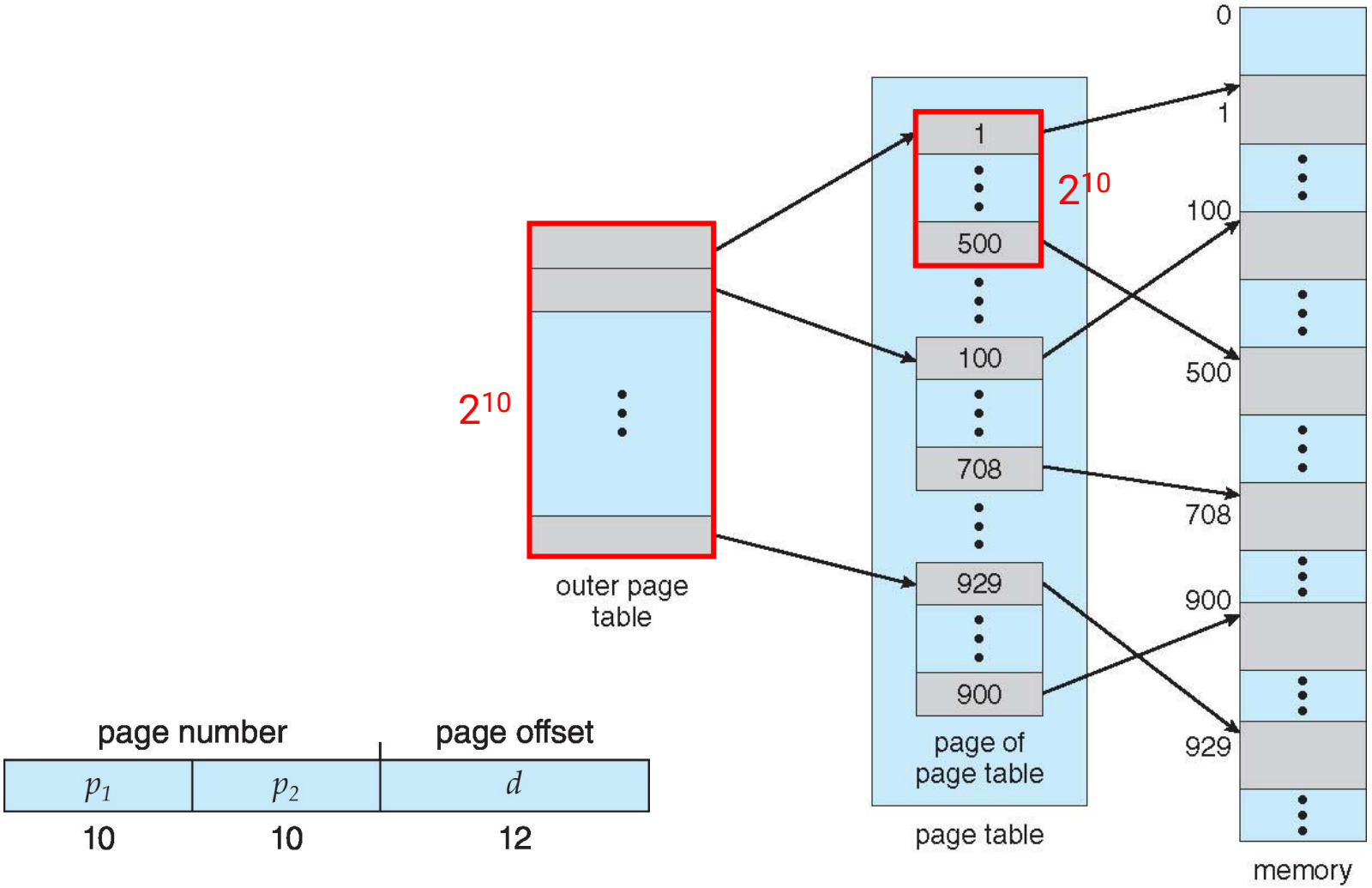
  ➔ Total size = 4MB
  - Need to break it into several smaller page tables, better within a single page size (i.e. 4KB)
  - Or reduce the total size of page table
- Solutions
  - Hierarchical paging
  - Hash page table
  - Inverted page table

Hierarchical Paging

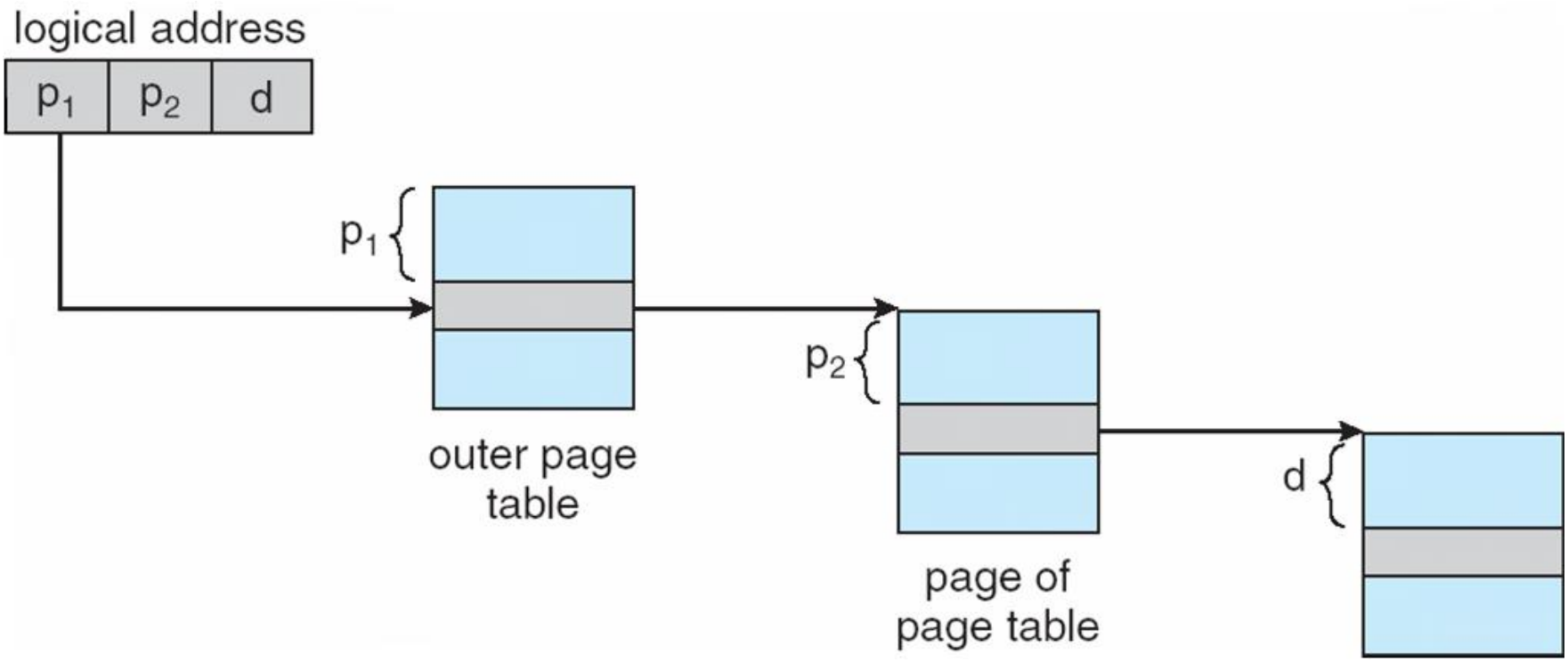
- Break up the logical address space into **multiple page tables**
  - Paged the page table
  - i.e. *n*-level page table
- Two-level paging (32-bit address with 4KB (2^{12}) page size)
  - 12-bit offset (d) ➔ 4KB (2^{12}) page size
  - 10-bit **outer** page number ➔ 1K (2^{10}) page table entries
  - 10-bit **inner** page number ➔ 1K (2^{10}) page table entries
  - 3 memory accesses



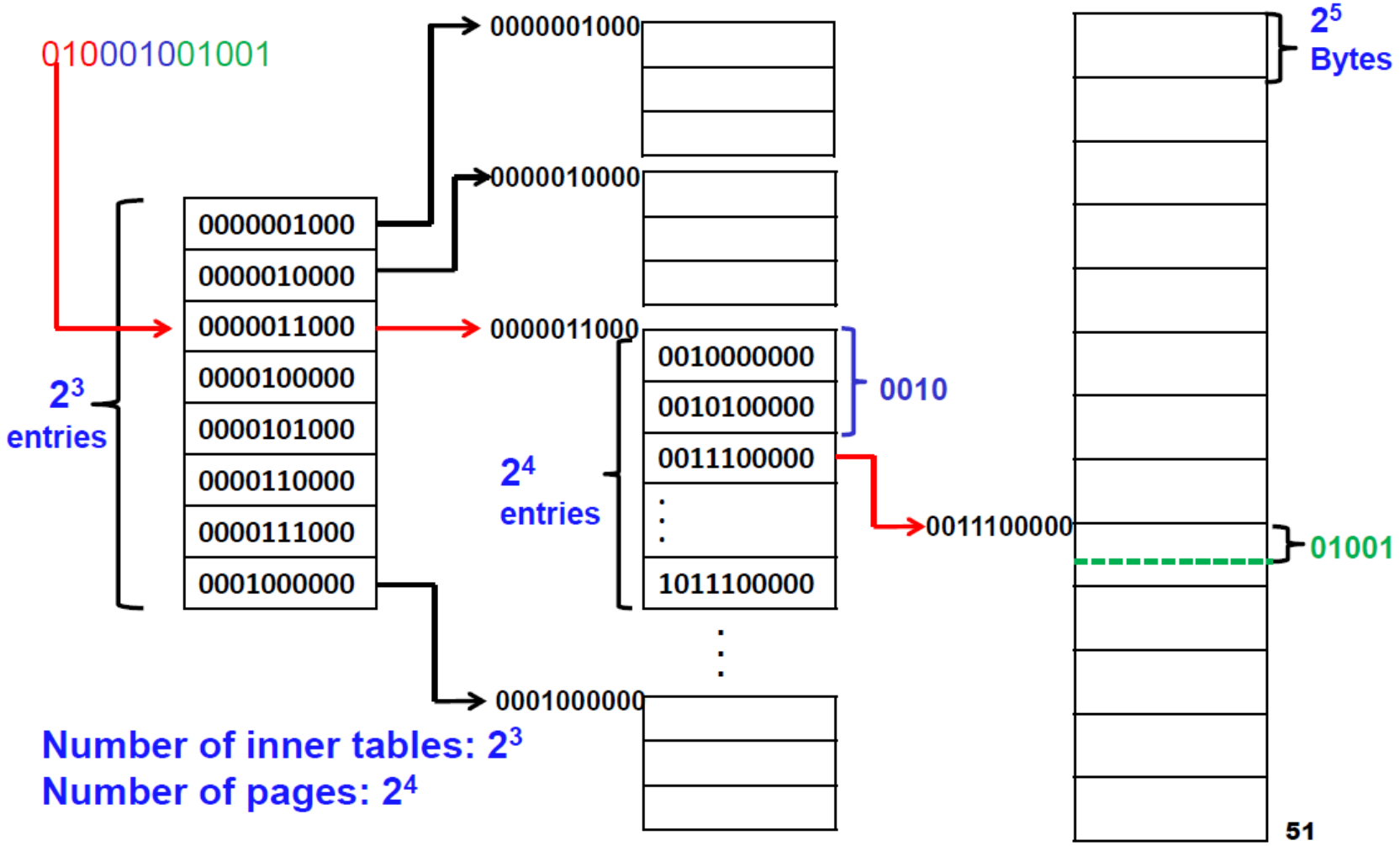
Two-Level Page Table Example



Two-Level Address Translation



Two-Level Address Translation Example



64-bit Address

- How about 64-bit address? (assume each entry needs 4 Bytes)
  - 42 (p1) + 10 (p2) + 12 (offset)

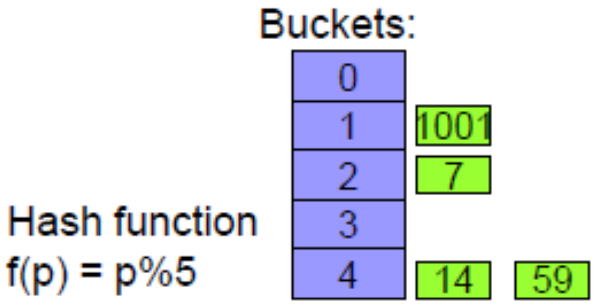
  ➔ outer table requires 2^{42} x 4B = **16TB** contiguous memory
  - 12 (p1) + 10 (p2) + 10 (p3) + 10 (p4) + 10 (p5) + 12 (offset)

  ➔ outer table requires 2^{12} x 4B = 16KB contiguous memory

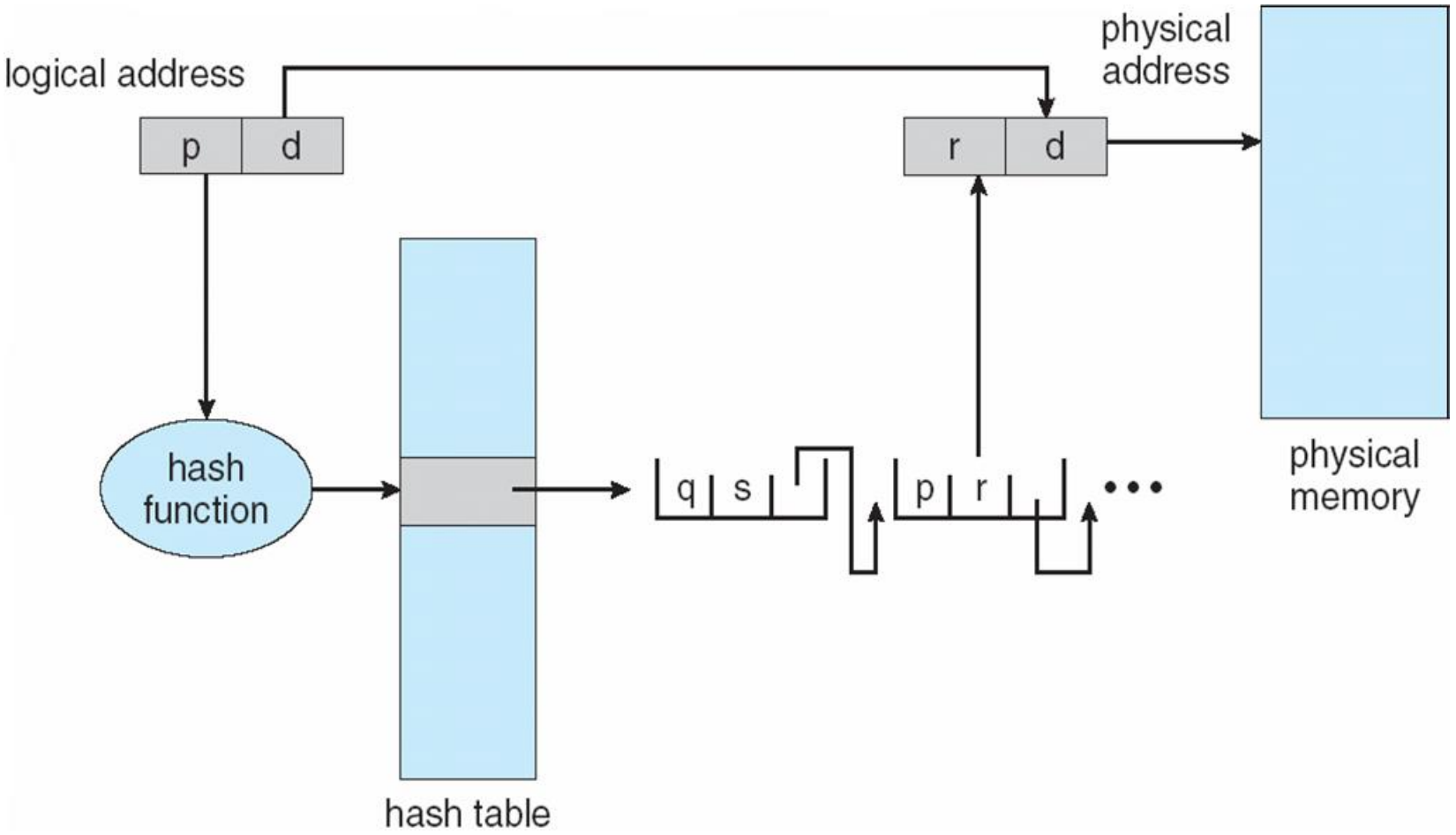
  ➔ **6** memory accesses !!!
- Examples
  - SPARC (32-bit) and Linux use 3-level paging
  - Motorola 68030 (32-bit) uses 4-level paging

Hashed Page Table

- **Commonly-used for address > 32 bits**
- Virtual page number is hashed into a hash table
- The size of the hash table varies
  - Larger hash table ➔ smaller chains in each entry
- Each entry in the hashed table contains
  - (Virtual Page Number, Frame Number, Next Pointer)
  - Pointers waste memory
  - Traverse linked list waste time and cause additional memory references



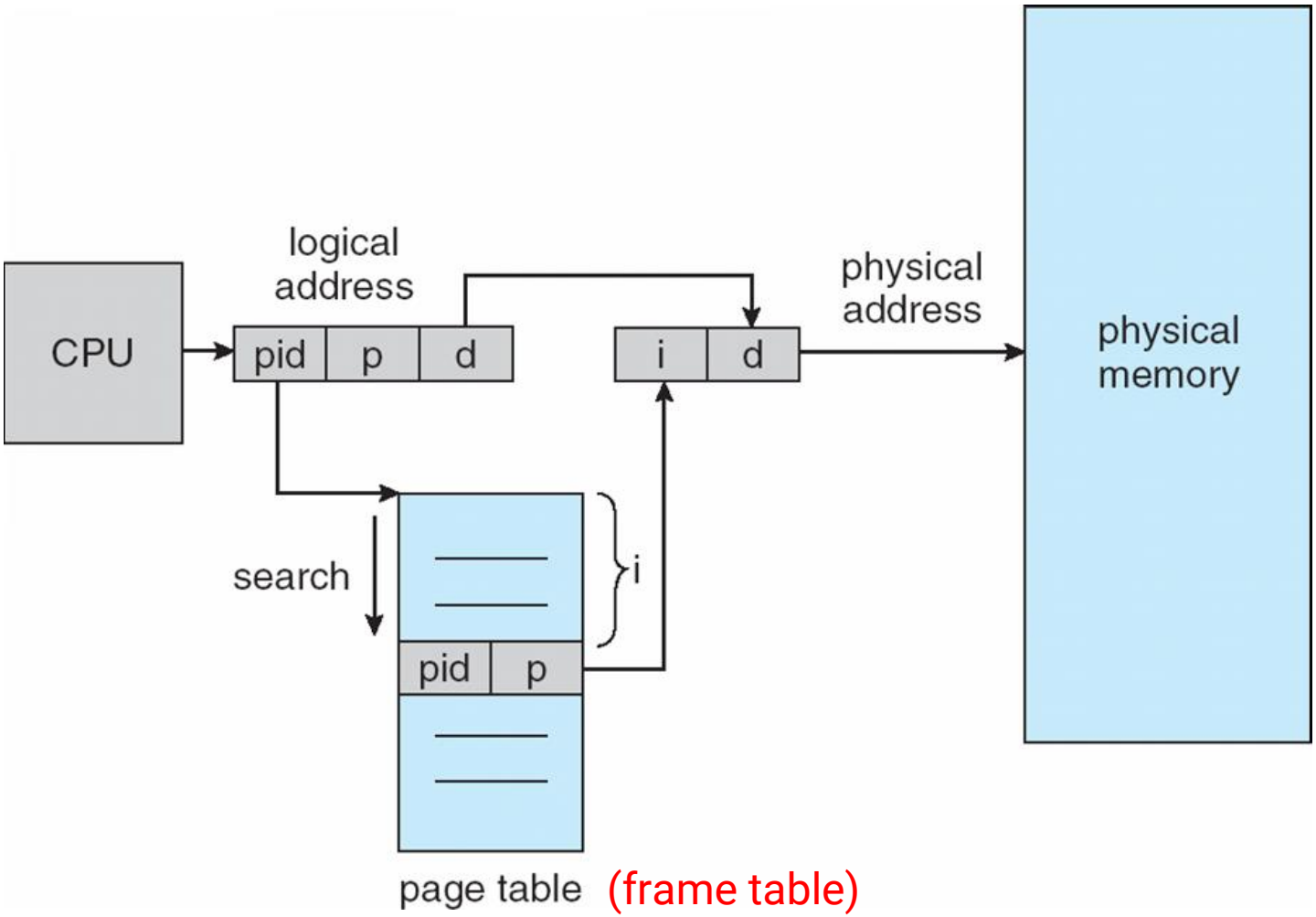
Hash Page Table Address Translation



Inverted Page Table (Frame Table)

- Maintains **no** page table for each process
- Maintains a **frame table** for the whole memory
  - One entry for each real frame of memory
- Each entry in the hashed table contains
  - (PID, Page Number)
- Eliminate the memory needed for page tables but **increase memory access time**
  - Each access needs to search the whole frame table
  - Solution: use hashing for the frame table
- **Hard to support shared page / memory**

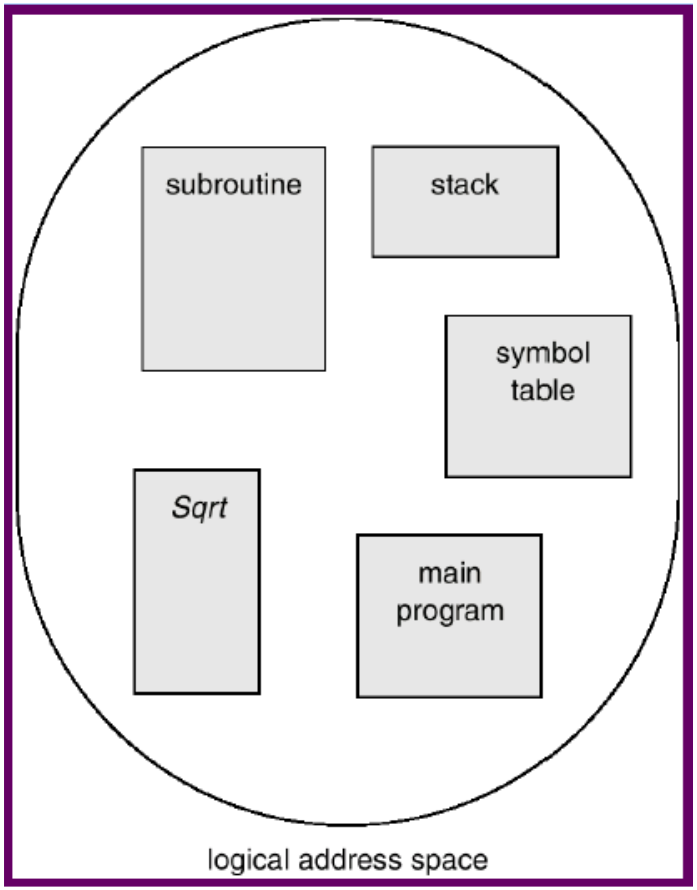
Inverted Page Table Address Translation



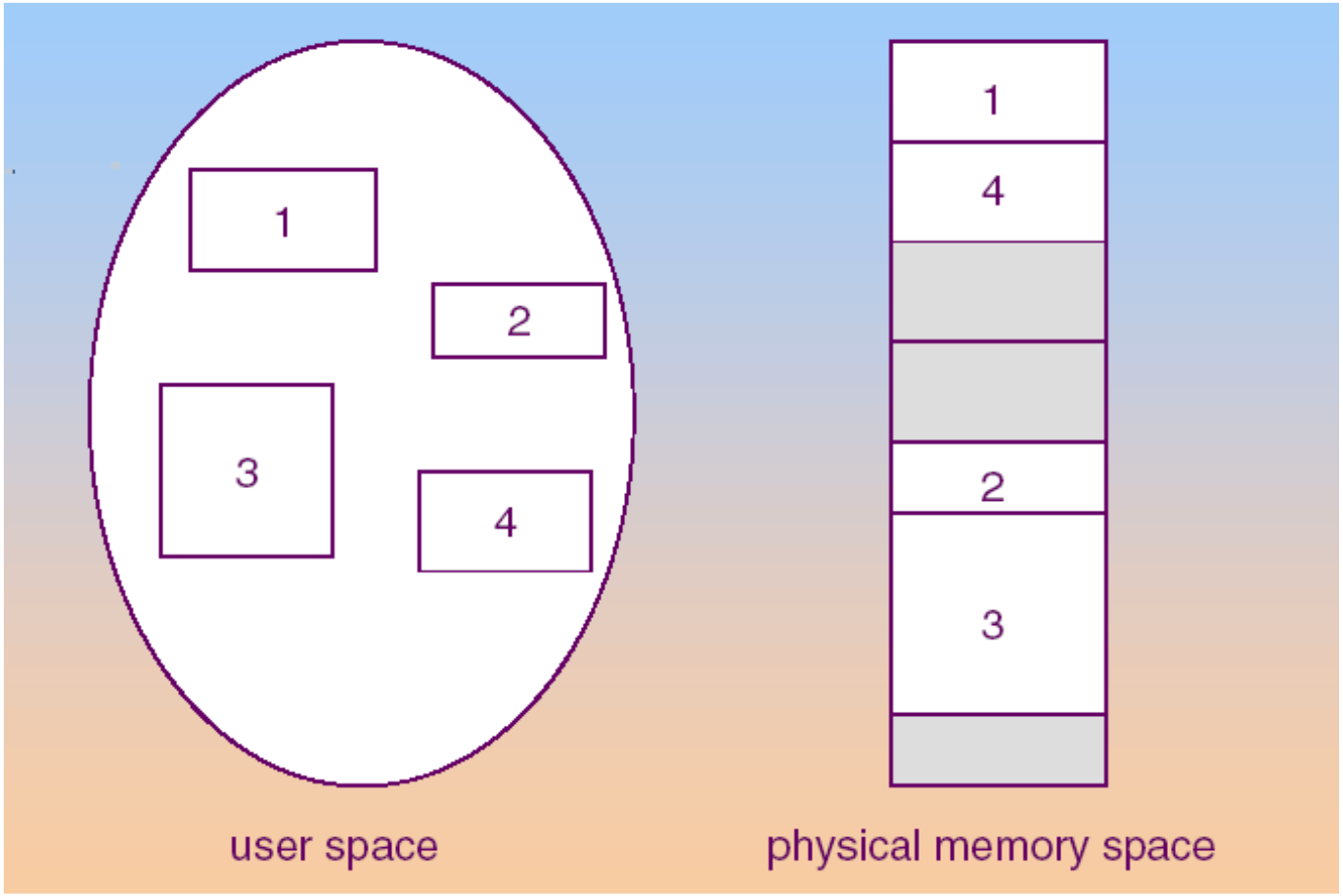
Segmentation

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
  - Function
  - Object
  - Local/global variables
  - Stack
  - Symbol table
  - Arrays



Logical View of Segmentation

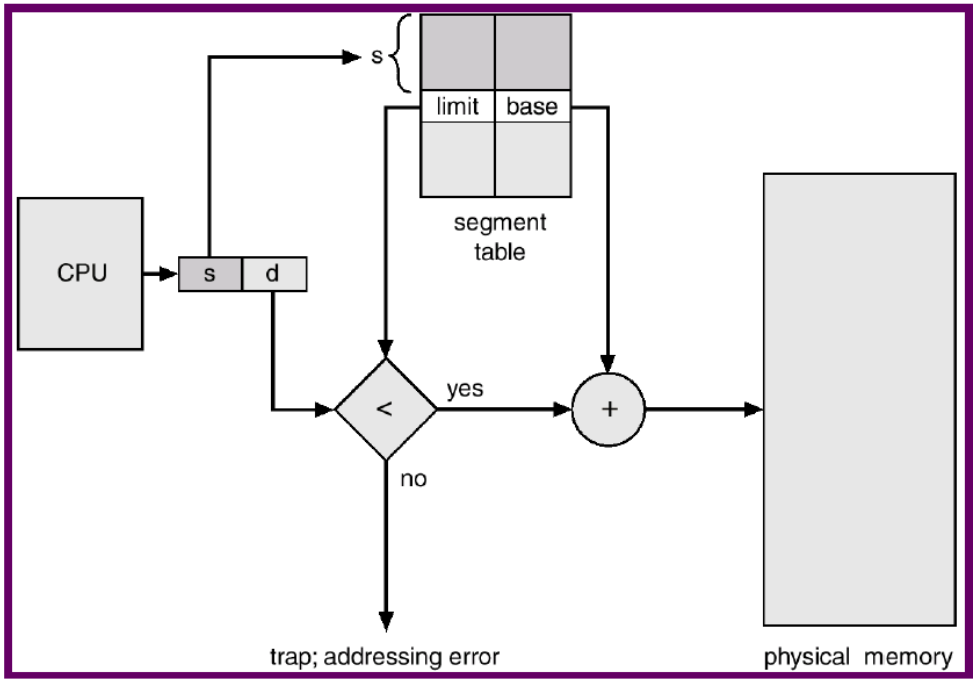


Segmentation Table

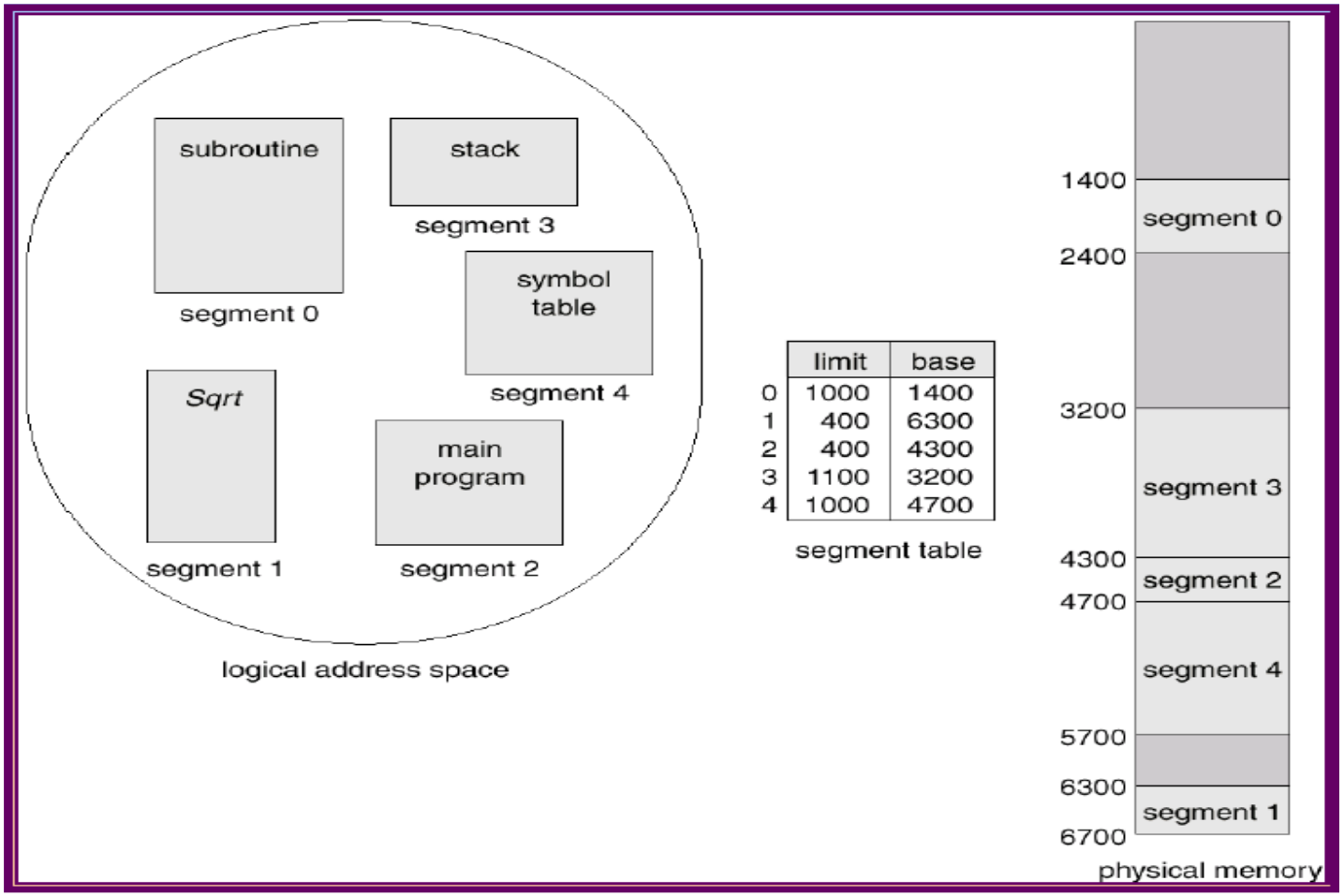
- Logical address: (seg#, offset)
  - Offset is only limited by physical address
- **Segmentation table**
  - Maps two-dimensional physical addresses
  - Each table entry has
    - **Base (4 bytes):** the start physical address
    - **Limit (4 bytes):** the length of the segment
- **Segment-table base register (STBR)**
  - The physical address of the segmentation table
- **Segment-table length register (STLR)**
  - The number of segments

Segmentation Hardware

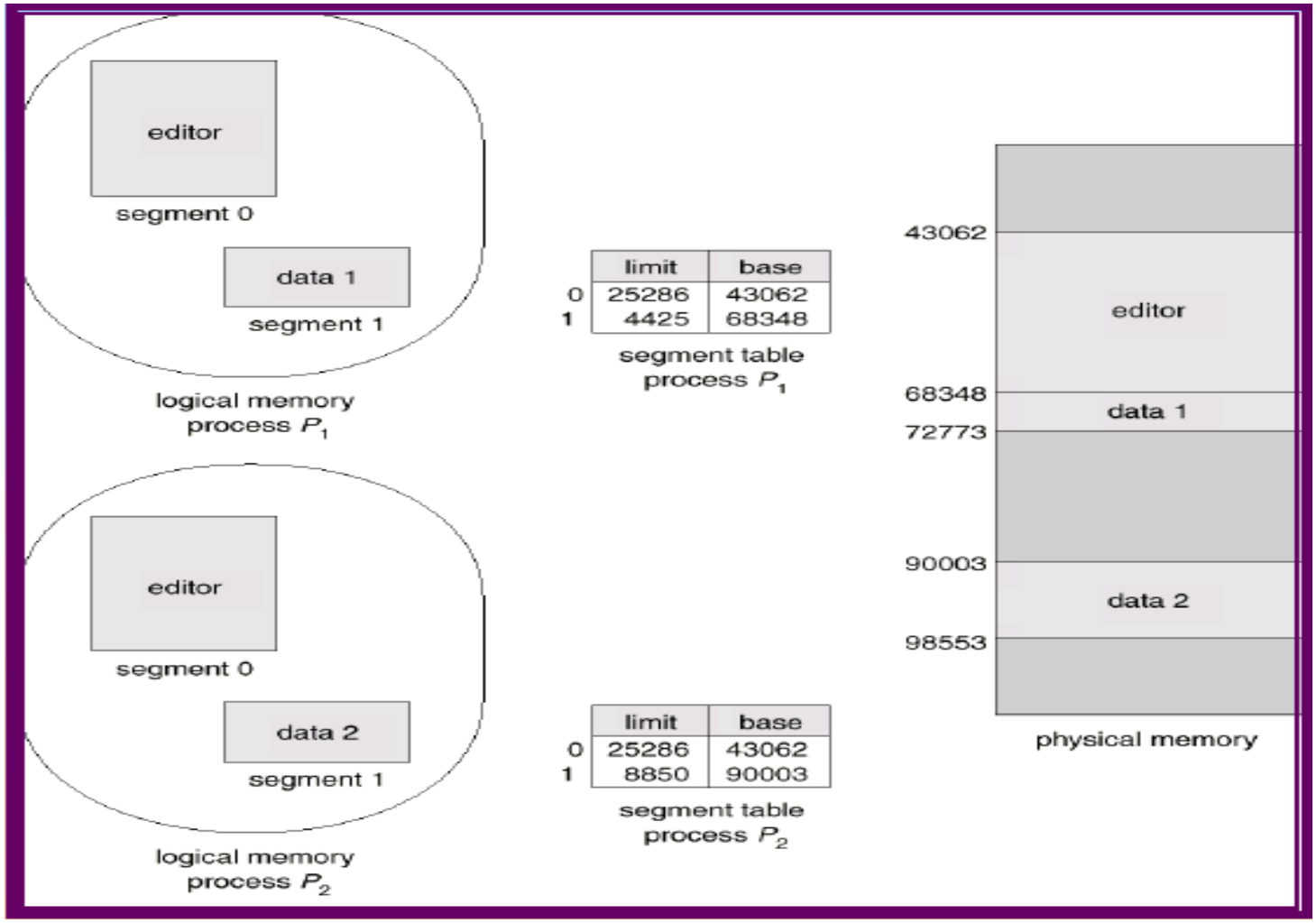
- Limit register is used to check offset length
- MMU allocate memory by assigning an appropriate **base address for each segment**
  - Physical address cannot overlap between segments



Example of Segmentation



Sharing of Segments



Protection and Sharing

- Protection bits associated with segments
  - Read-only segment (code)
  - Read-write segments (data, heap, stack)
- Code sharing occurs at **segment level**
  - Shared memory communication
  - Shared library
- Share segment by having same base in two segment tables

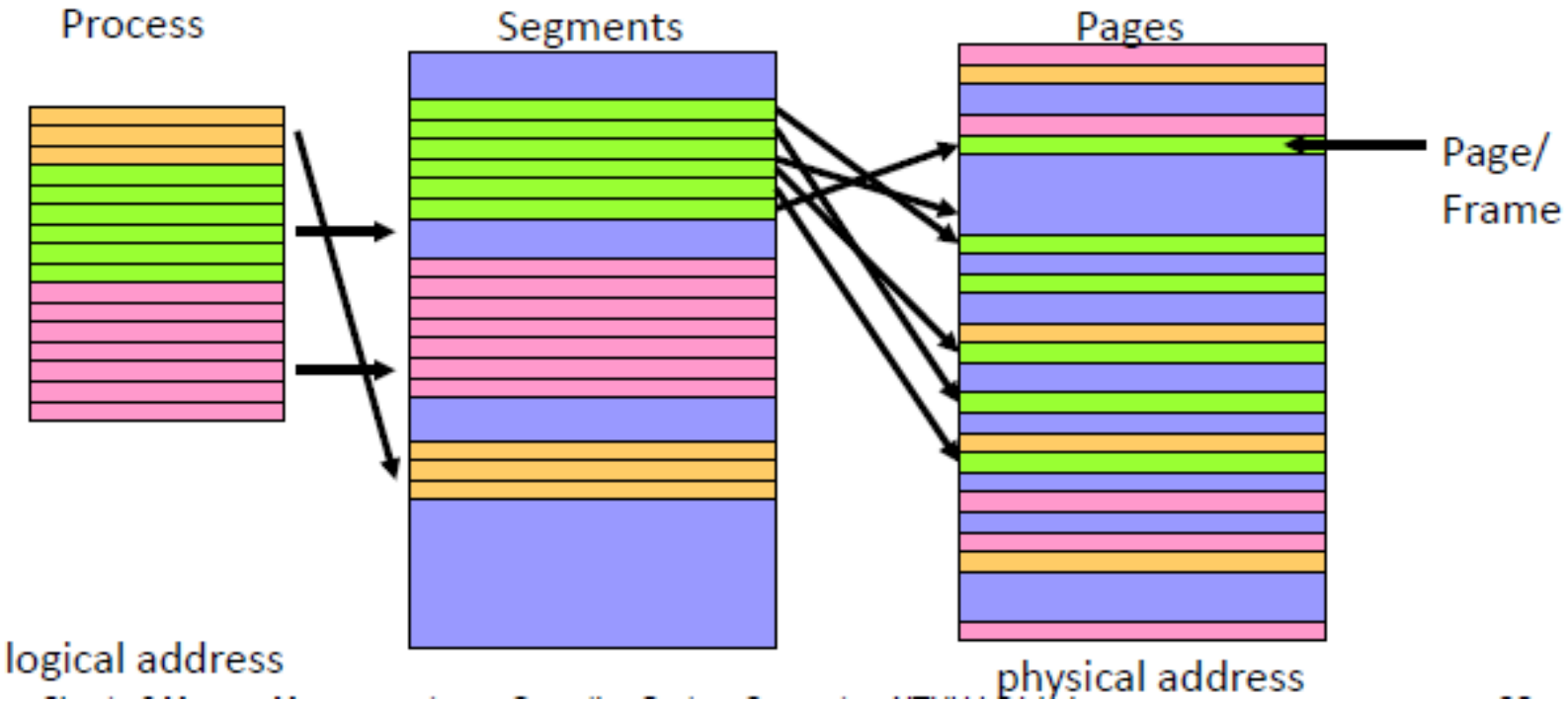
Segmentation v.s. Paging

	Paging	segmentation
Length	Fixed	Varied
Fragmentation	Internal	External
Table entry	Page number → frame number	Seg ID → (base addr, limit length)
View	Physical memory	User program

Segmentation with Paging

Basic Concept

- Apply **segmentation** in **logical address space**
- Apply **paging** in **physical address space**

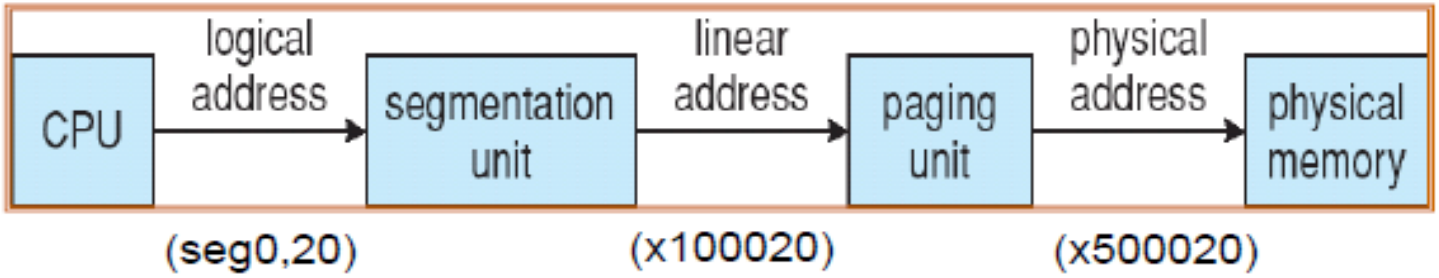


Address Translation

- CPU generates logical address
  - Given to **segmentation unit**

  ➔ Produces **linear addresses**
  - Linear address given to paging unit

  ➔ Generates **physical address** in main memory
- Segmentation and paging units form equivalent of MMU



Objective Review

- To provide a detailed description of various ways of organizing memory hardware

- To discuss various memory-management techniques