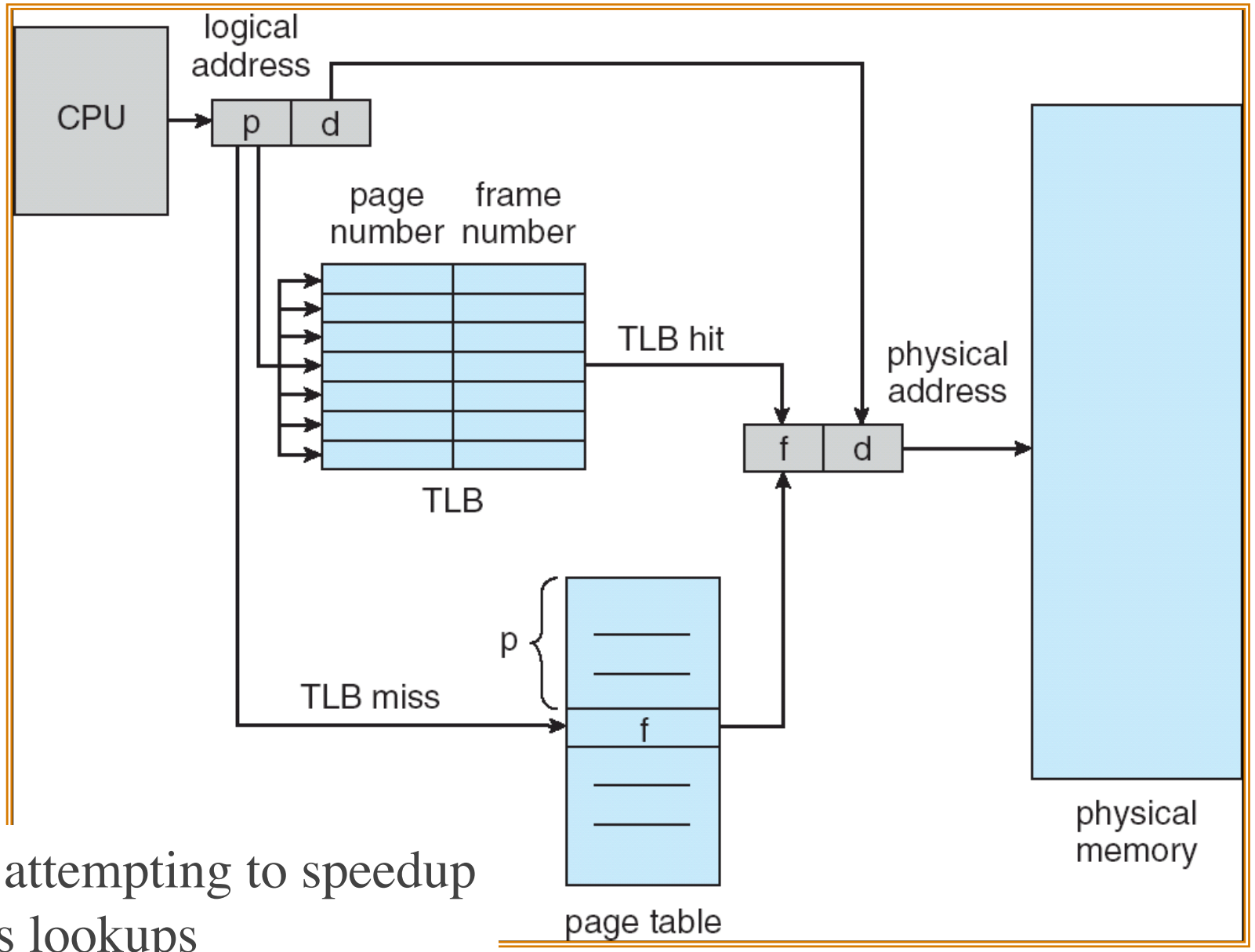
# Paging Hardware With TLB




We are attempting to speedup address lookups

# Effective Access Time

- Associative Lookup = $\varepsilon$ time unit
- Assume memory cycle time is 1 microsecond
- Hit ratio – percentage of times that a page number is found in the associative registers; ratio related to number of associative registers
- Hit ratio = $\alpha$
- **Effective Access Time** (EAT)

$$EAT = (1 + \varepsilon)\,\alpha + (2 + \varepsilon)(1 - \alpha)$$
$$= 2 + \varepsilon - \alpha$$

# TLB

- Some TLBs support address-space ID
  - OS loans a unique value per process
  - If current process ASID != TLB ASID, then don't use it
- Otherwise, TLBs are flushed at context switch

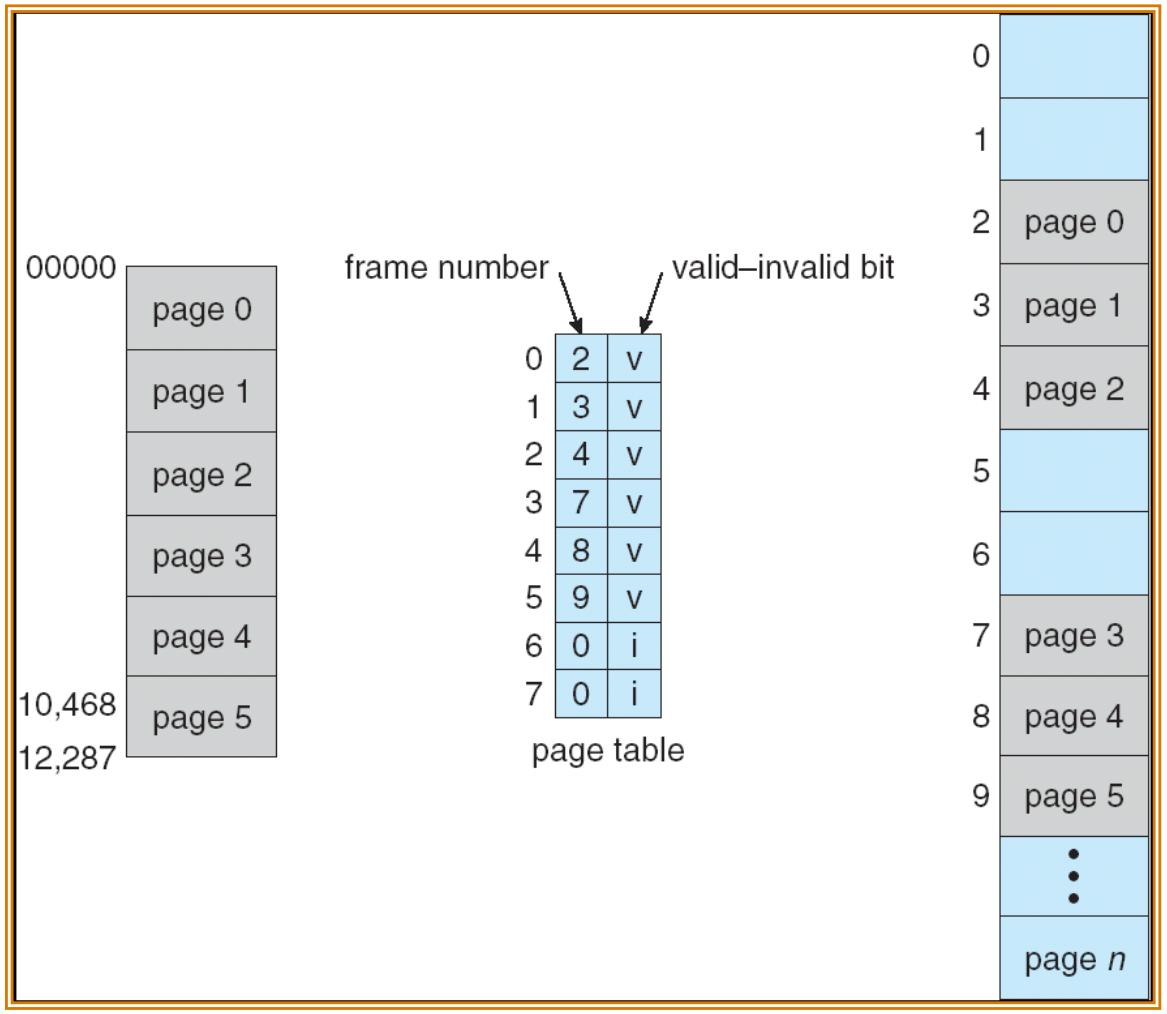
- Question: what affects TLB hit ratio?
  - For code?
  - For data?

# 8.4.3: Memory Protection

- Need some mechanism to identify that a page is not allocated to a process (even though the page table will have an entry for this logical page)

- **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




Next chapter, we will see more uses for this bit

# 8.5: Page Table Structure

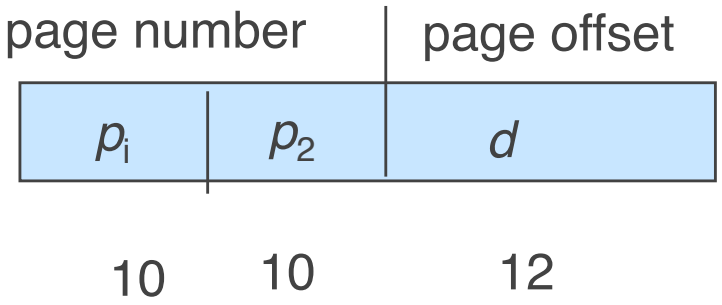
- Problem is that page tables are per-process structure and they can be large
  - Consider 64 bit address space and page size of 8 KB
    - Page table size = $2^{51}$ or $2*10^{15}$ entries
- Hierarchical Paging
- Hashed Page Tables
- Inverted Page Tables

# Hierarchical Page Tables

- Break up the logical address space into multiple page tables
- A simple technique is a two-level page table

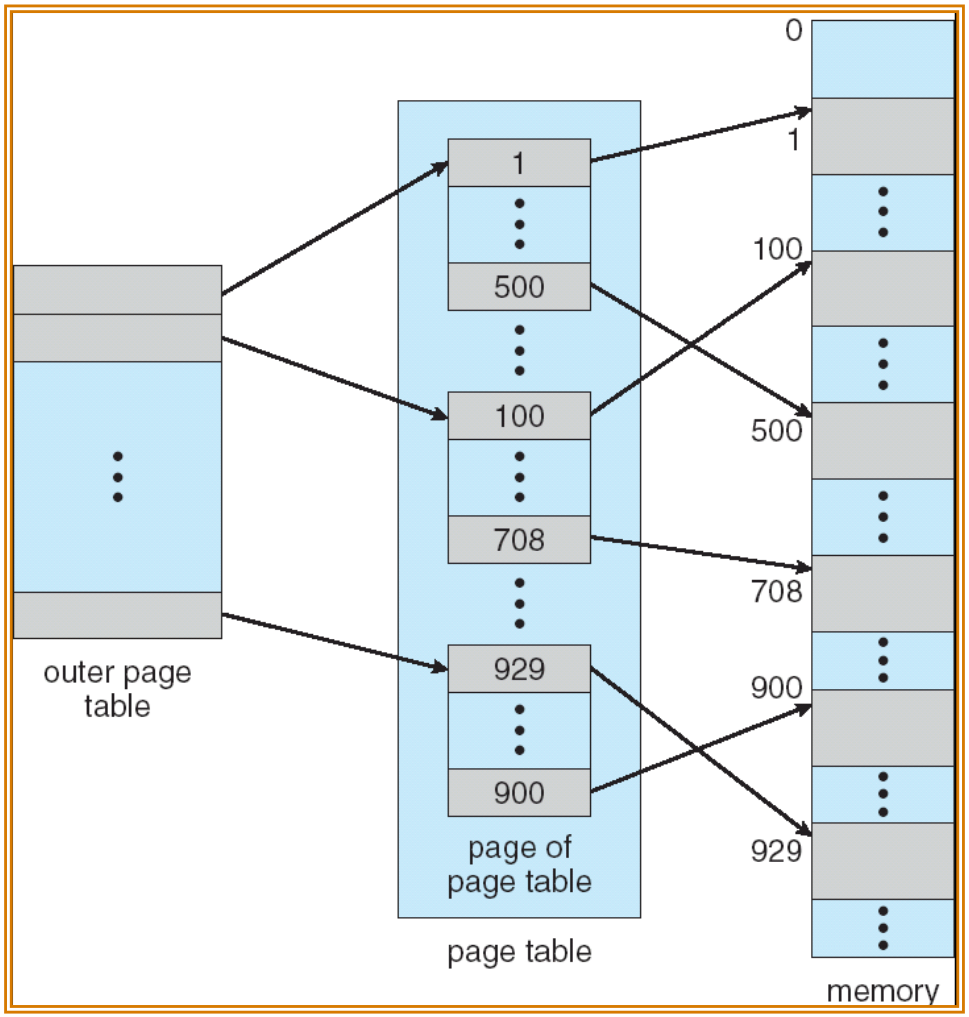
# Two-Level Paging Example

- A logical address (on 32-bit machine with 4K page size) is divided into:
  - a page number consisting of 20 bits
  - a page offset consisting of 12 bits
- Since the page table is paged, the page number is further divided into:
  - a 10-bit page number
  - a 10-bit page offset
- Thus, a logical address is as follows:



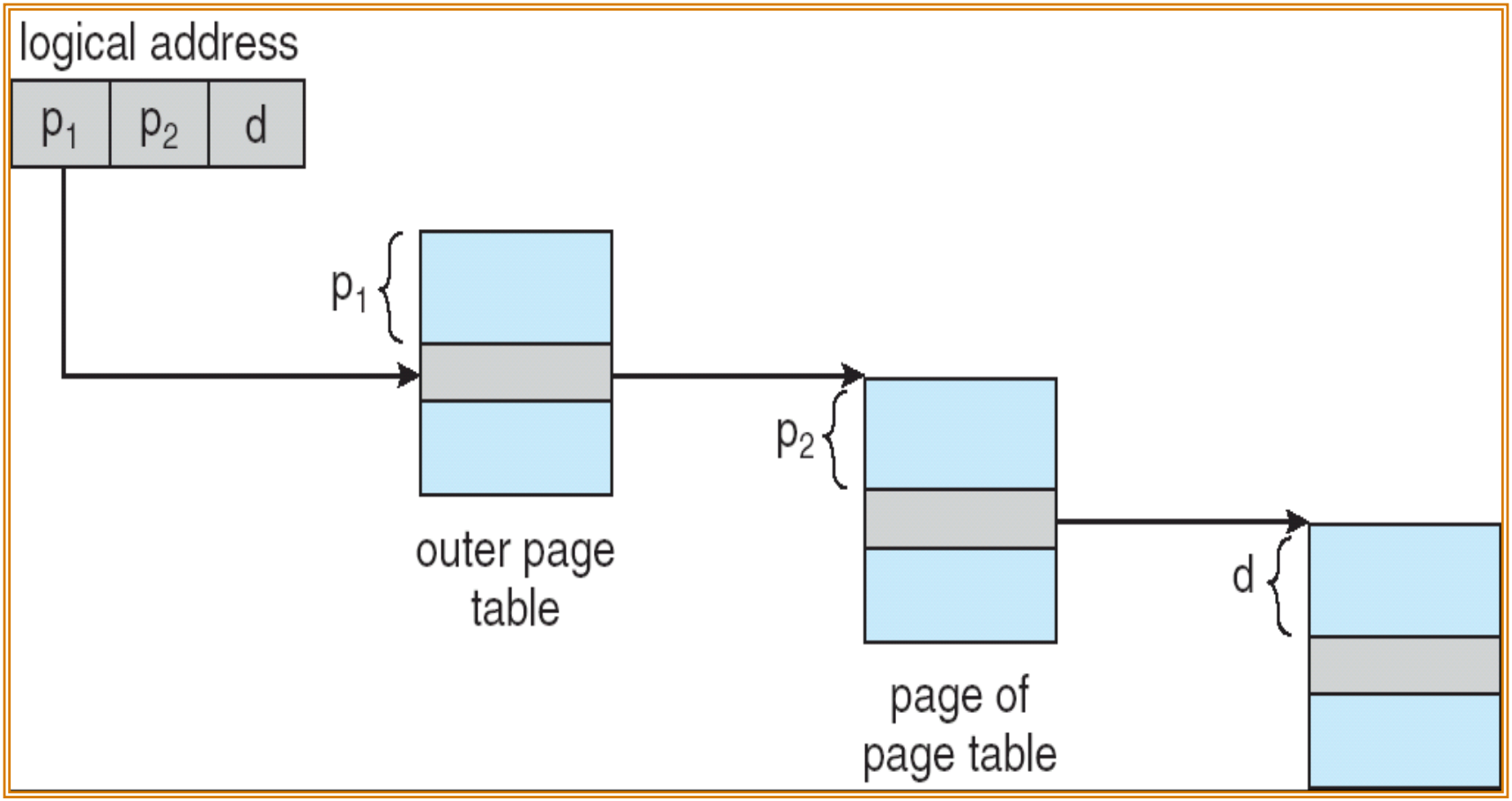

where $p_i$ is an index into the outer page table, and $p_2$ is the displacement within the page of the outer page table

# Two-Level Page-Table Scheme




# Address-Translation Scheme

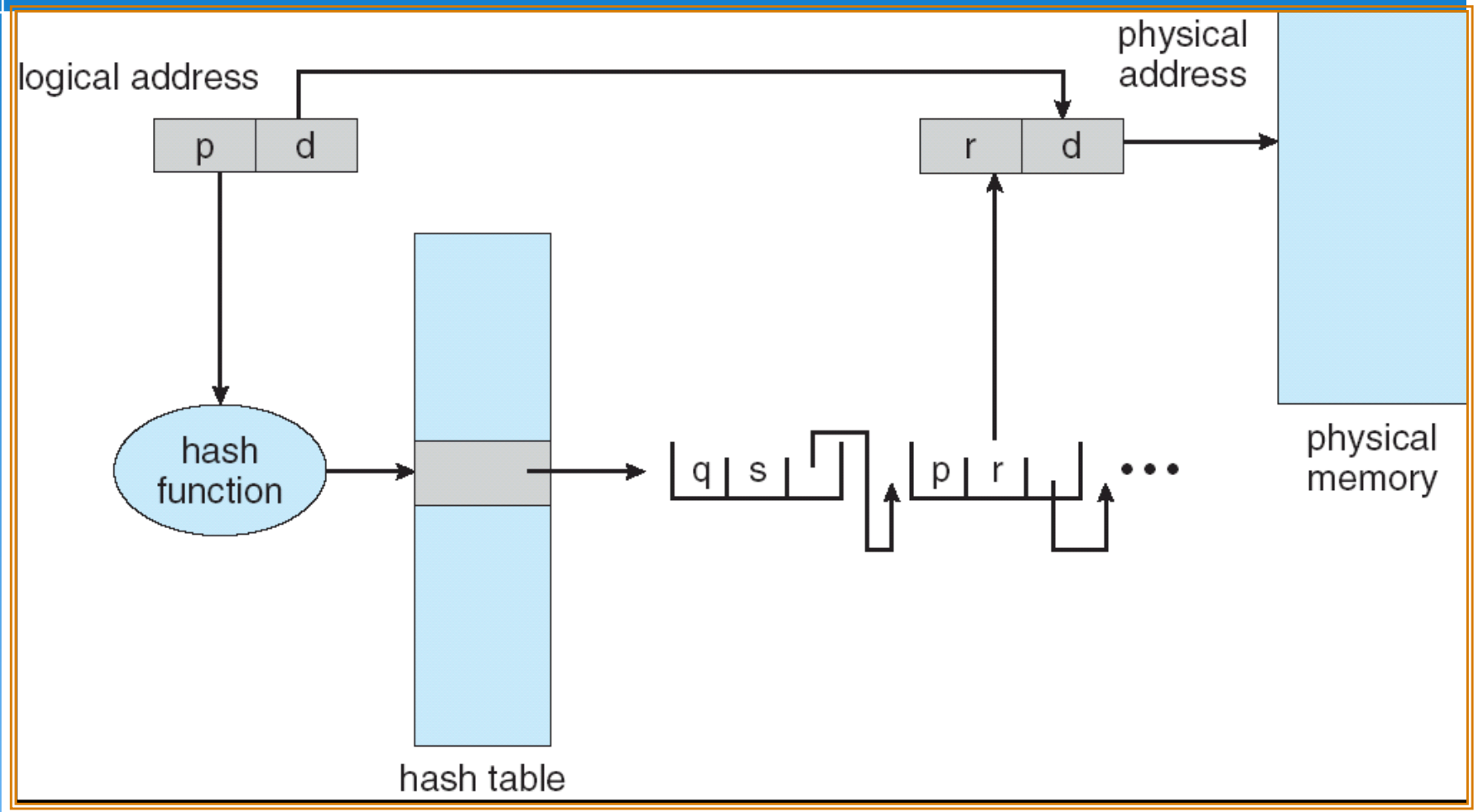
- Address-translation scheme for a two-level 32-bit paging architecture




# Hashed Page Tables

- Common in address spaces > 32 bits
- The virtual page number is hashed into a page table. This page table contains a chain of elements hashing to the same location.
- Virtual page numbers are compared in this chain searching for a match. If a match is found, the corresponding physical frame is extracted.

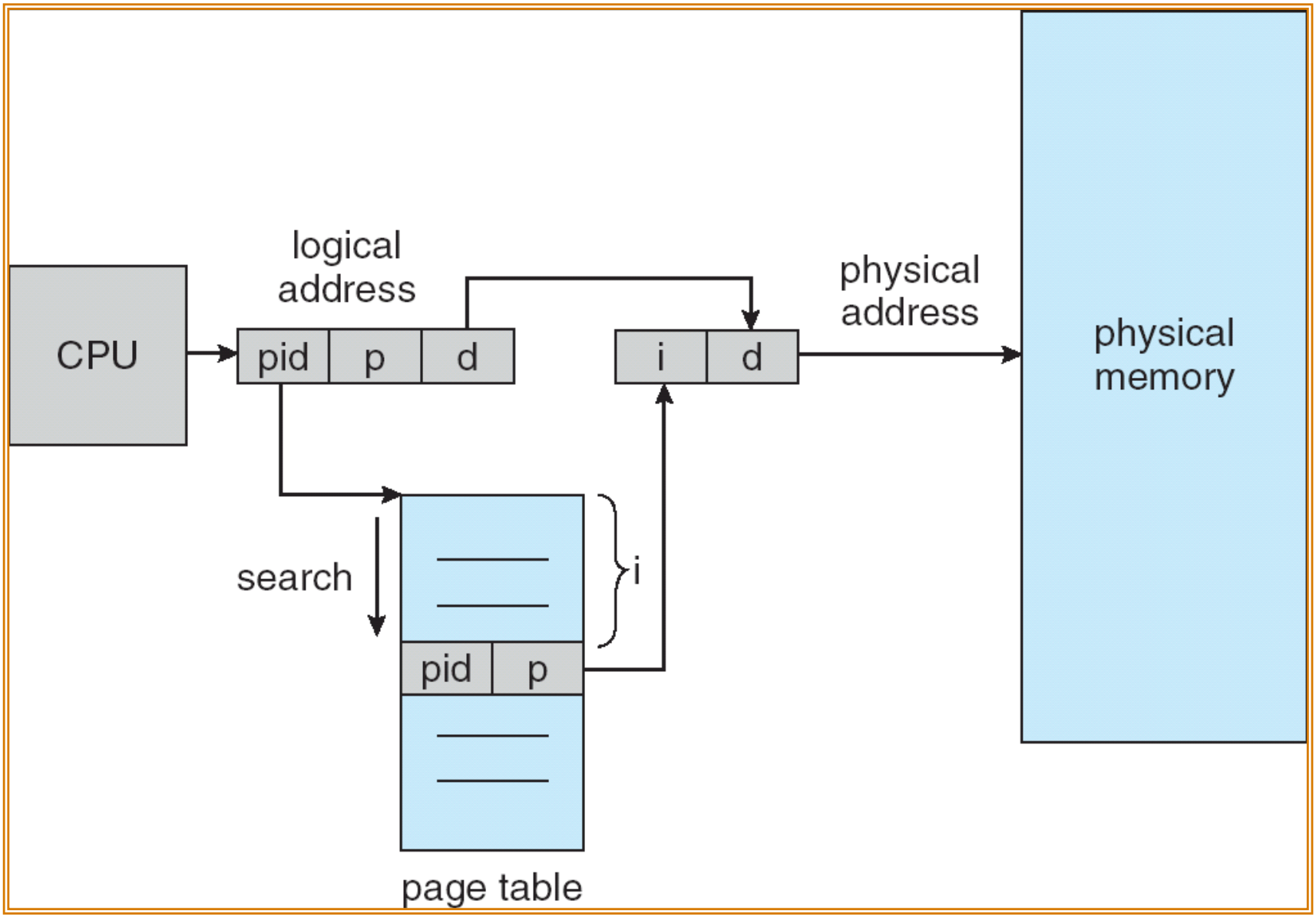
# Hashed Page Table




# Inverted Page Table

- One entry for each real frame of memory
- Entry consists of the virtual address of the page stored in that real memory location, with information about the process that owns that page
- Decreases memory needed to store each page table, but increases time needed to search the table when a page reference occurs
- Use hash table to limit the search to one — or at most a few — page-table entries

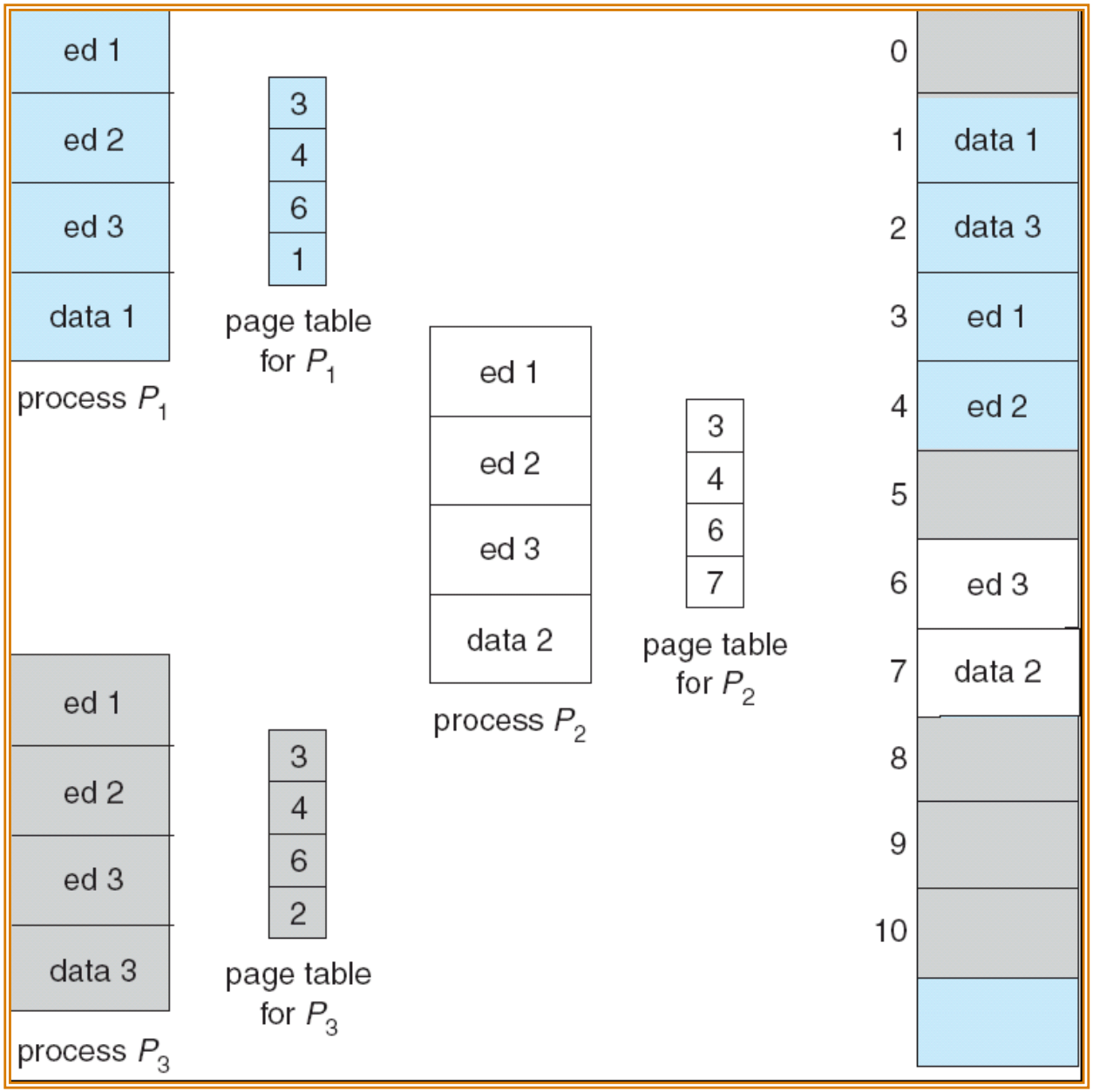
# Inverted Page Table Architecture




# 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

# Shared Pages Example



# 8.6: 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,

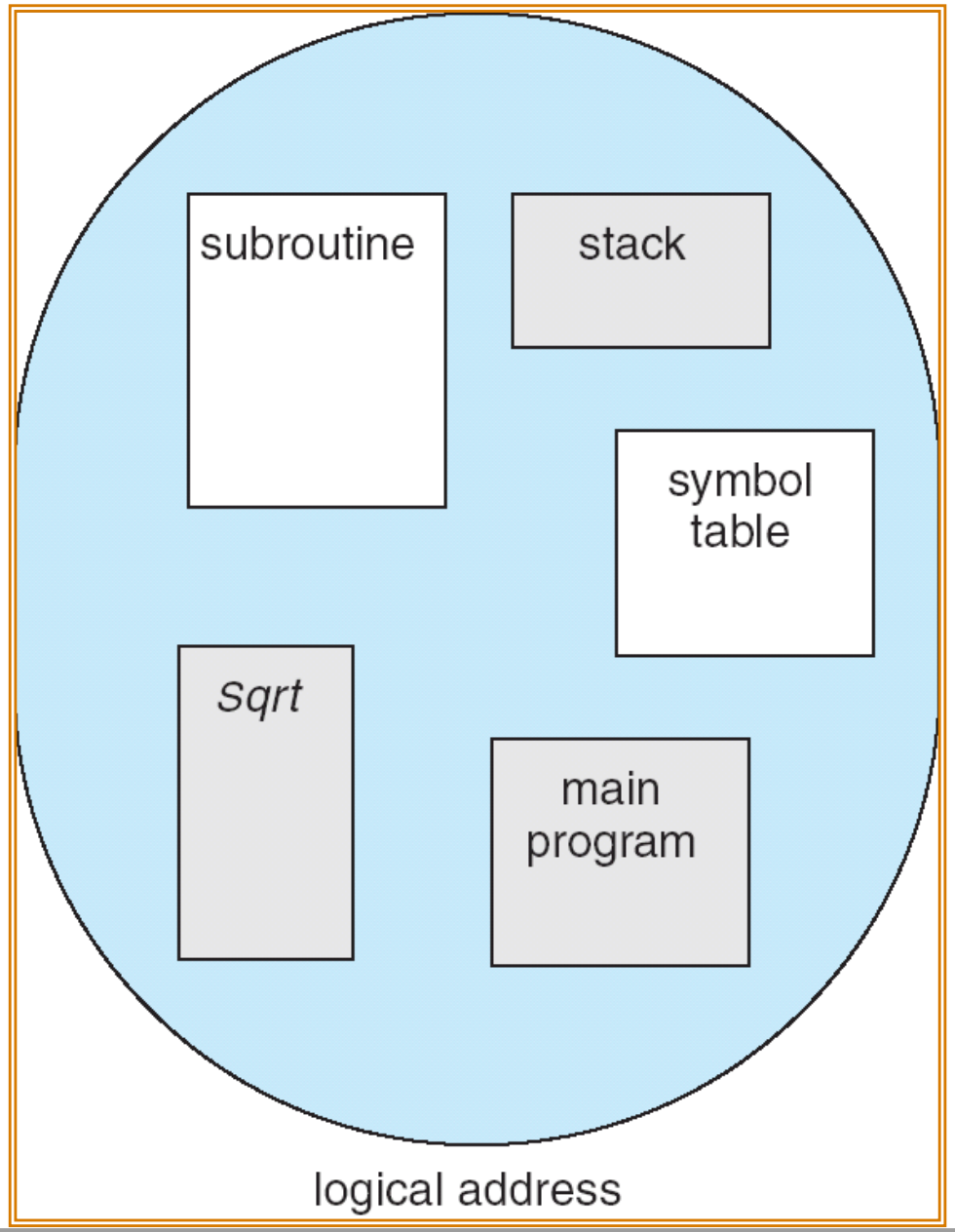
  local variables, global variables,

  common block,

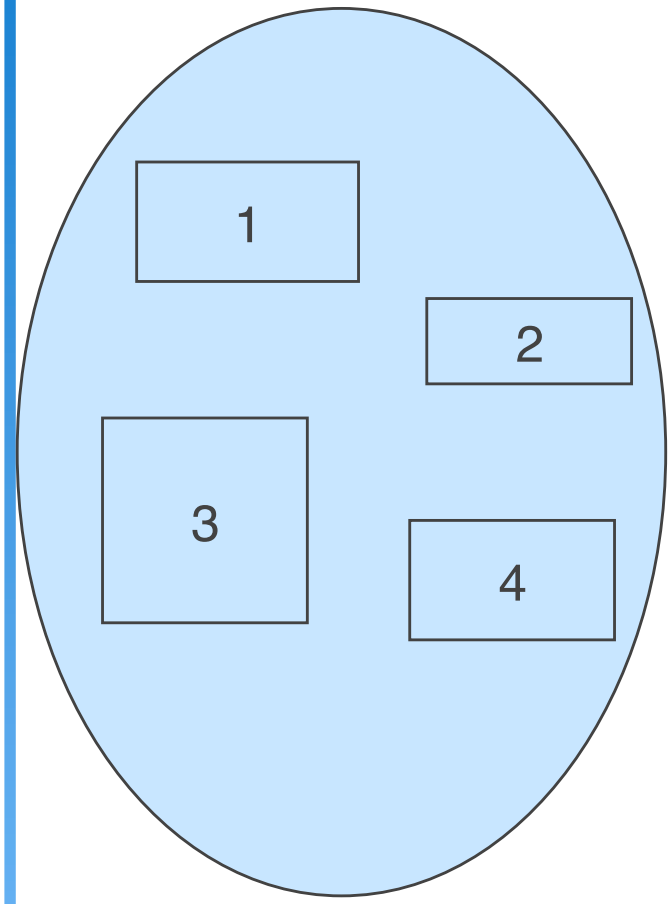
  stack,

  symbol table, arrays

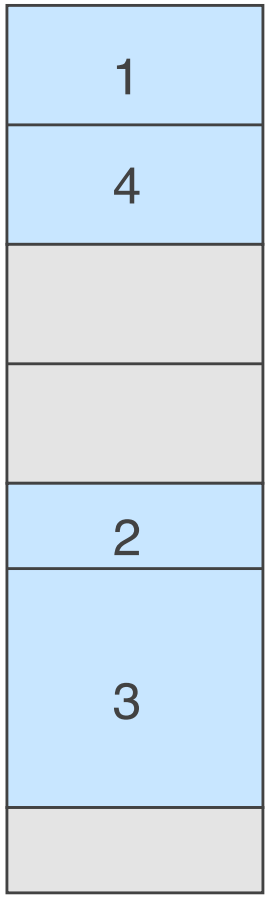
# User's View of a Program




# Logical View of Segmentation




user space




physical memory 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* < STLR

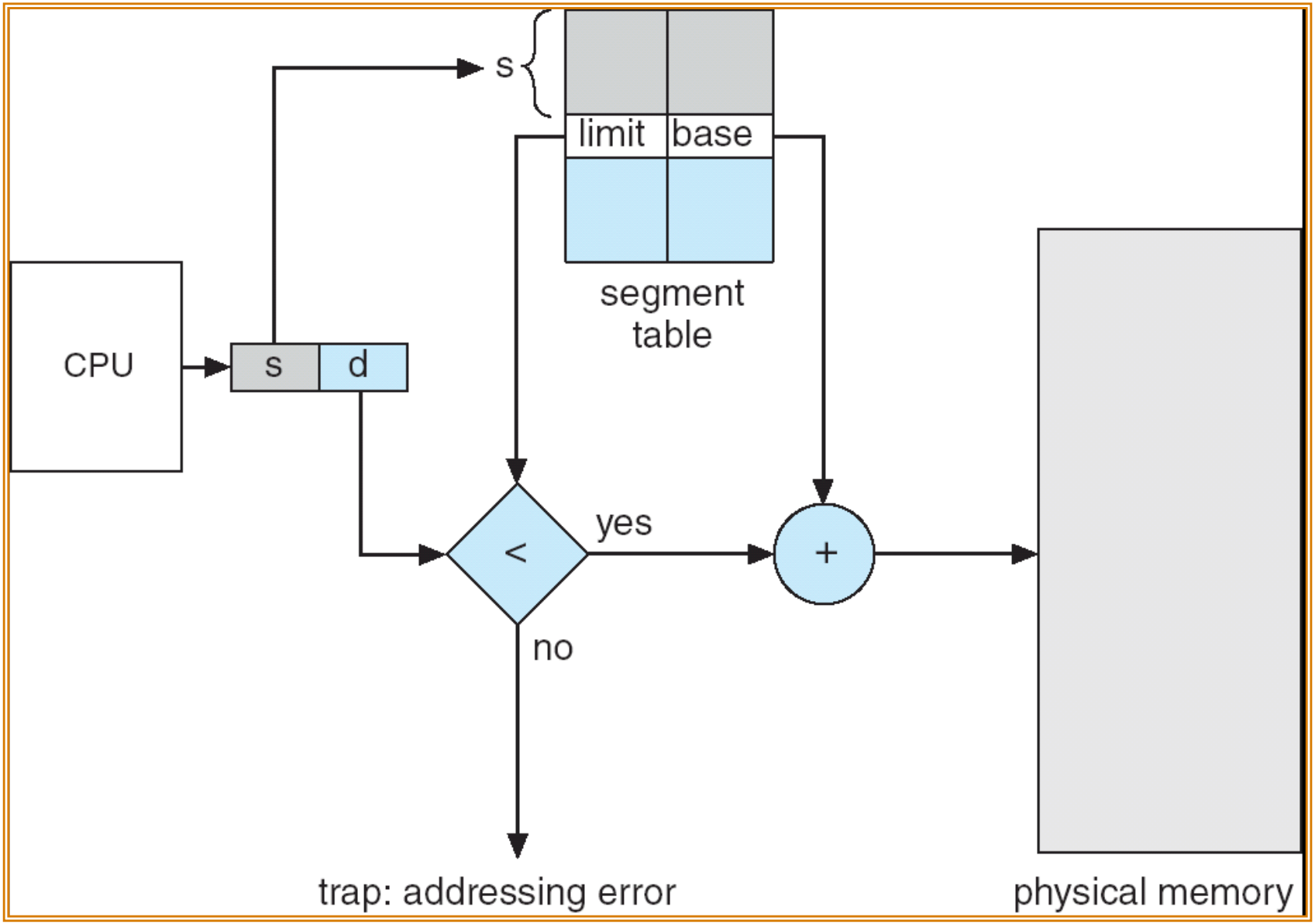
# Segmentation Architecture (Cont.)

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

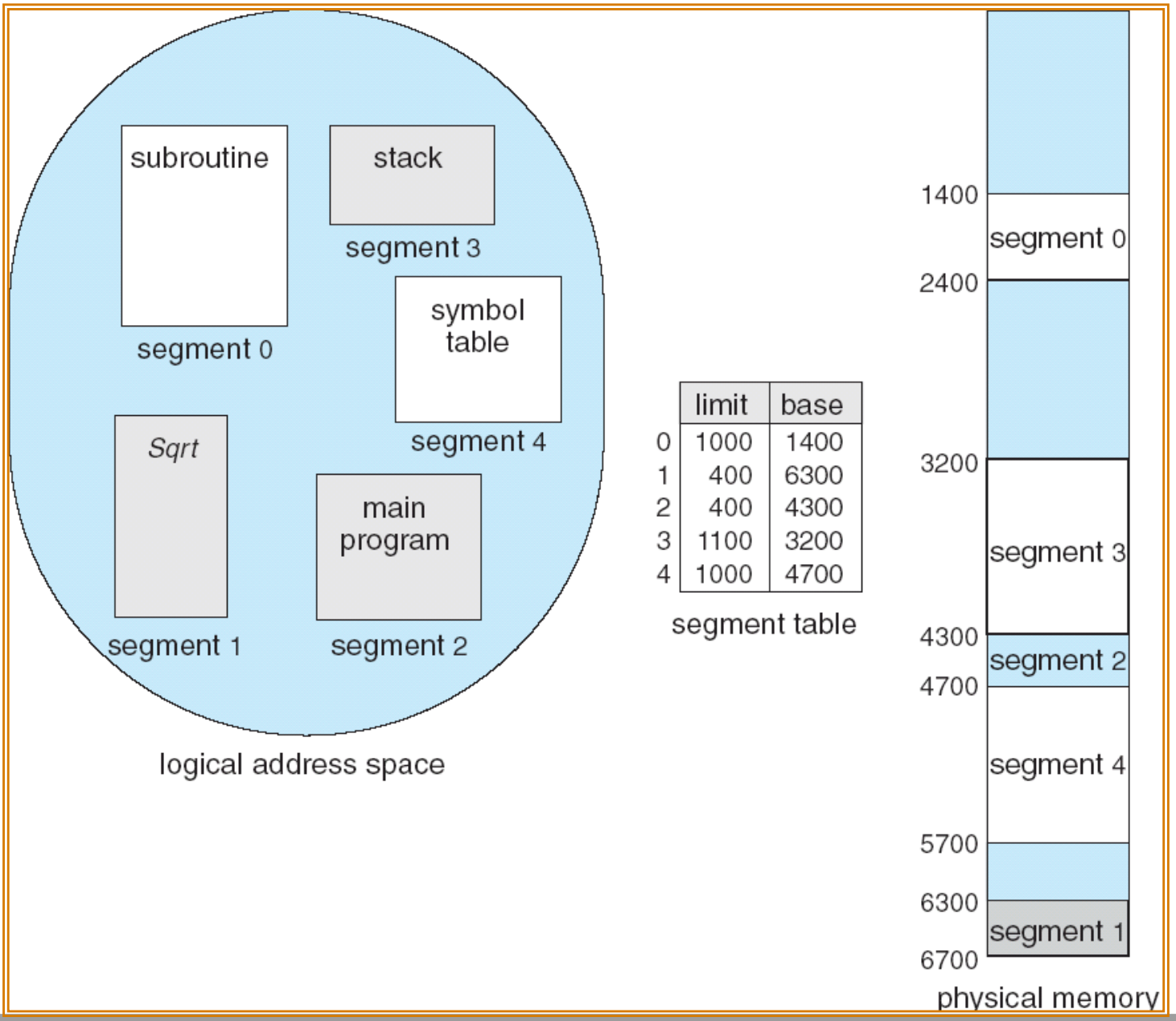
# Segmentation Architecture (Cont.)

- 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
- A segmentation example is shown in the following diagram

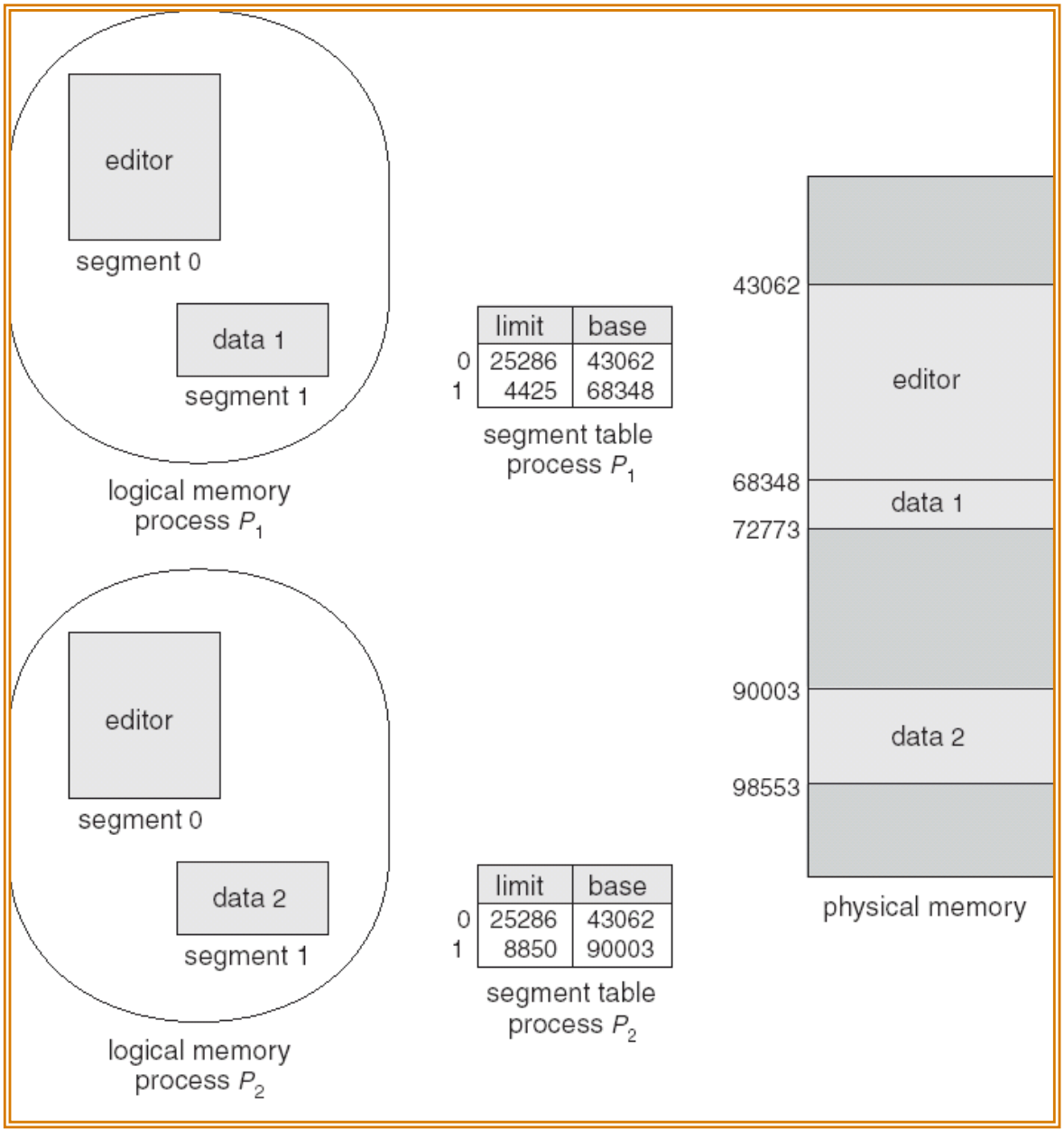
# Address Translation Architecture




# Example of Segmentation



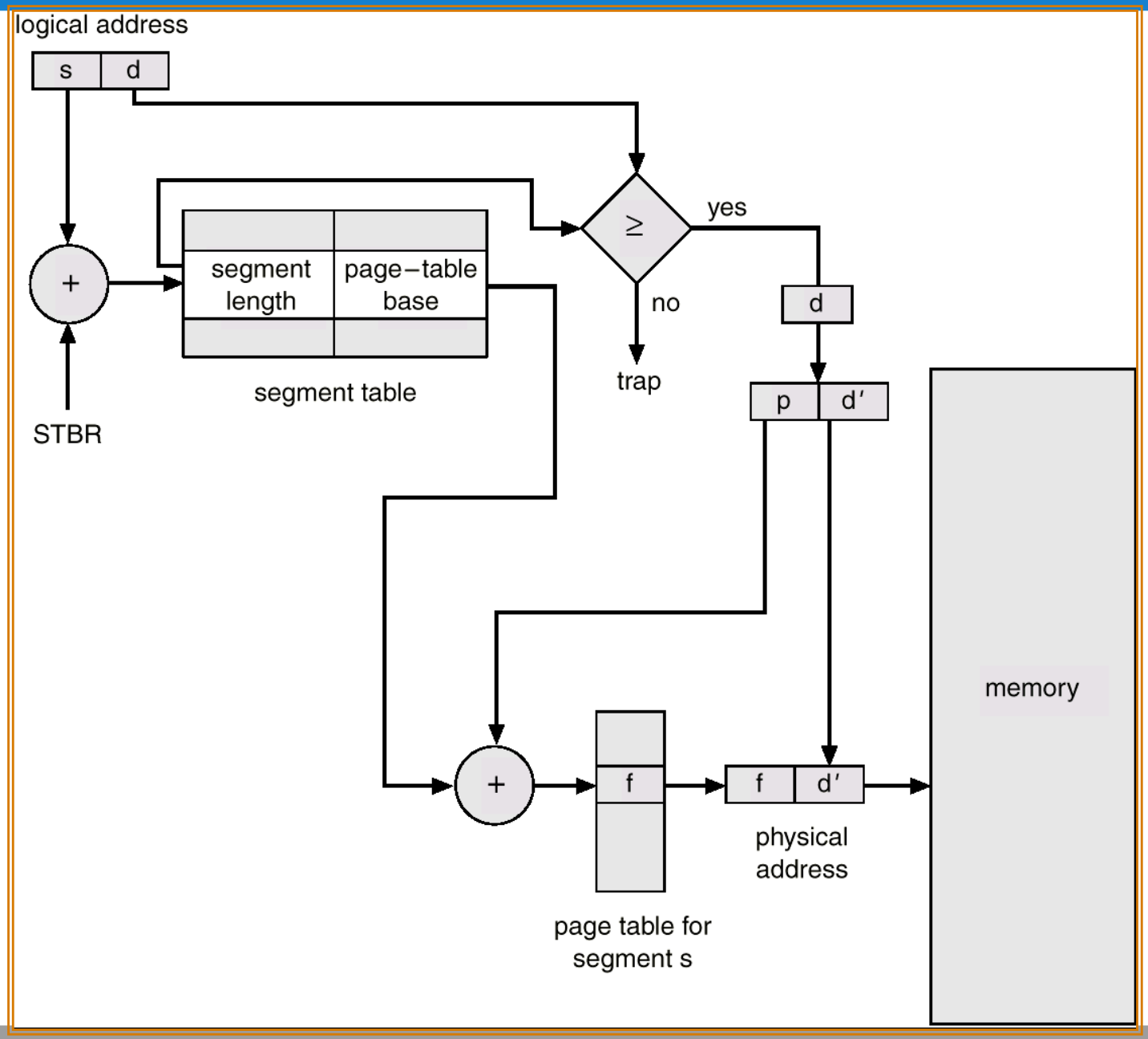

# Sharing of Segments




# Segmentation with Paging – MULTICS

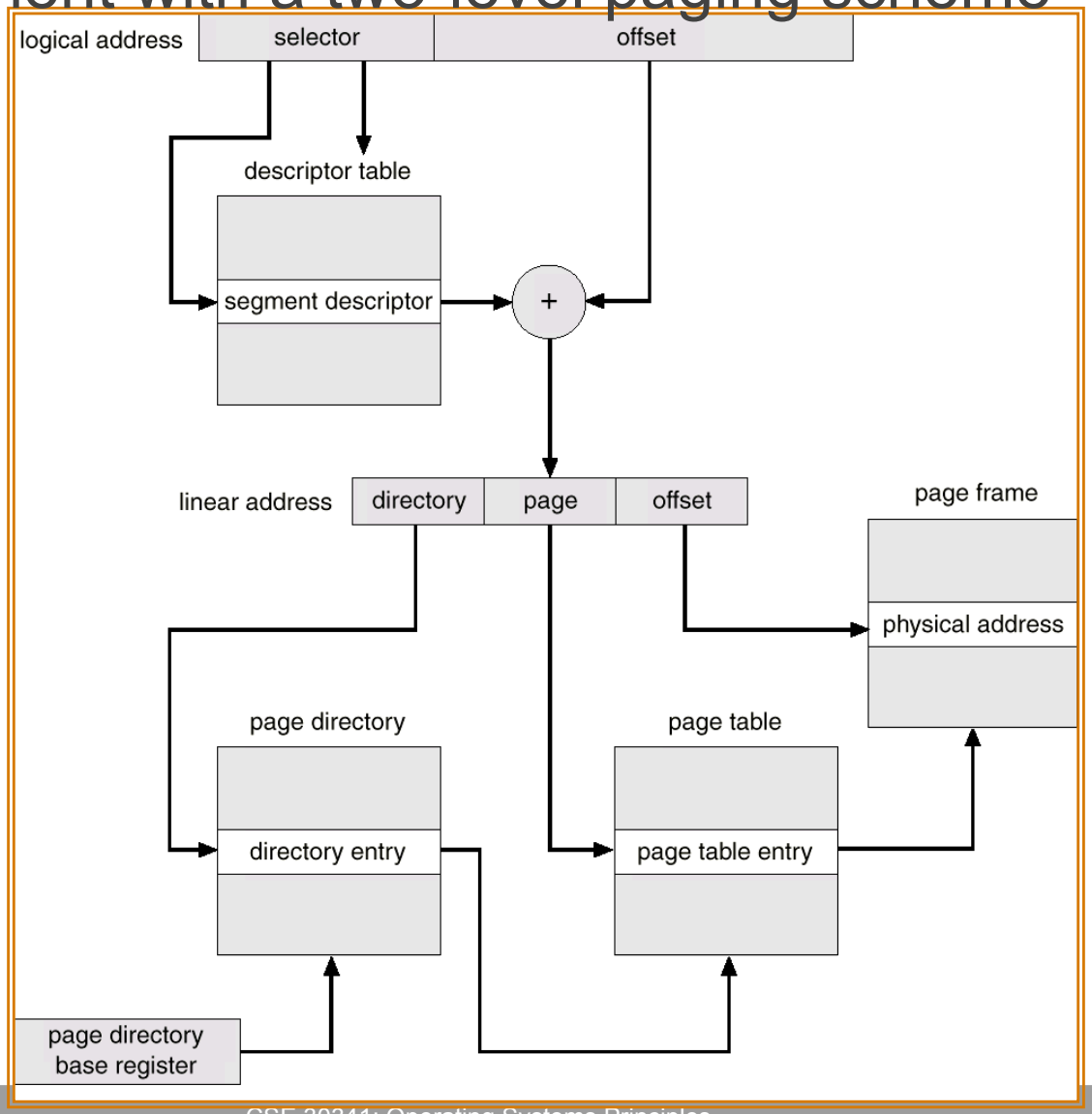
- The MULTICS system solved problems of external fragmentation and lengthy search times by paging the segments
- Solution differs from pure segmentation in that the segment-table entry contains not the base address of the segment, but rather the base address of a page table for this segment

# MULTICS Address Translation Scheme



# 8.7: Intel 30386 Address Translation

- segmentation with paging for memory management with a two-level paging scheme




# Linux on Intel 80x86

- Uses minimal segmentation to keep memory management implementation more portable
- Uses 6 segments:
  - Kernel code
  - Kernel data
  - User code (shared by all user processes, using logical addresses)
  - User data (likewise shared)
  - Task-state (per-process hardware context)
  - LDT
- Uses 2 protection levels:
  - Kernel mode
  - User mode