# Main Memory: Paging and Segmentation

CSSE 332

Operating Systems

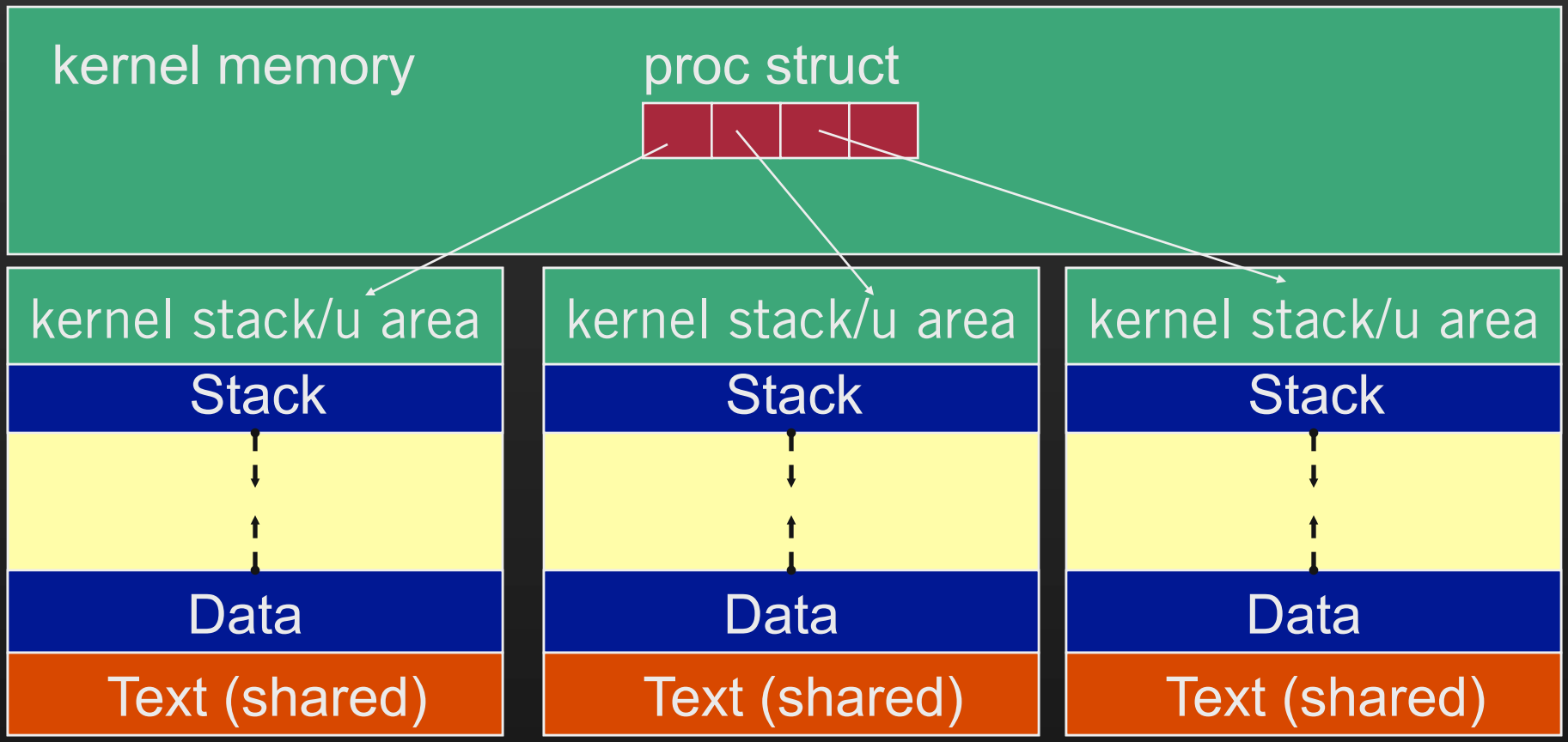
Rose-Hulman Institute of Technology

ROSE-HULMAN INSTITUTE OF TECHNOLOGY

# Background

- Program must be brought (from disk) into memory and placed within a process for it to be run
- Main memory, cache, and registers are ONLY storage CPU can access directly
- Register access in one CPU clock (or less)
- Main memory can take many cycles
- **Cache** sits between main memory and CPU registers
- Protection of memory required to ensure correct operation

# Big picture



# Paging

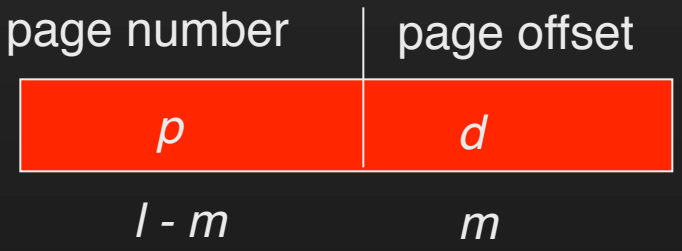
- Logical address space of a process can be noncontiguous
- Process is allocated physical memory wherever it is available
- Divide physical memory into fixed-sized blocks called **frames** (size is power of 2, between 512 bytes and 8,192 bytes)
- Divide logical memory into blocks of same size called **pages**

# Paging (cont.)

- 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 is minimal

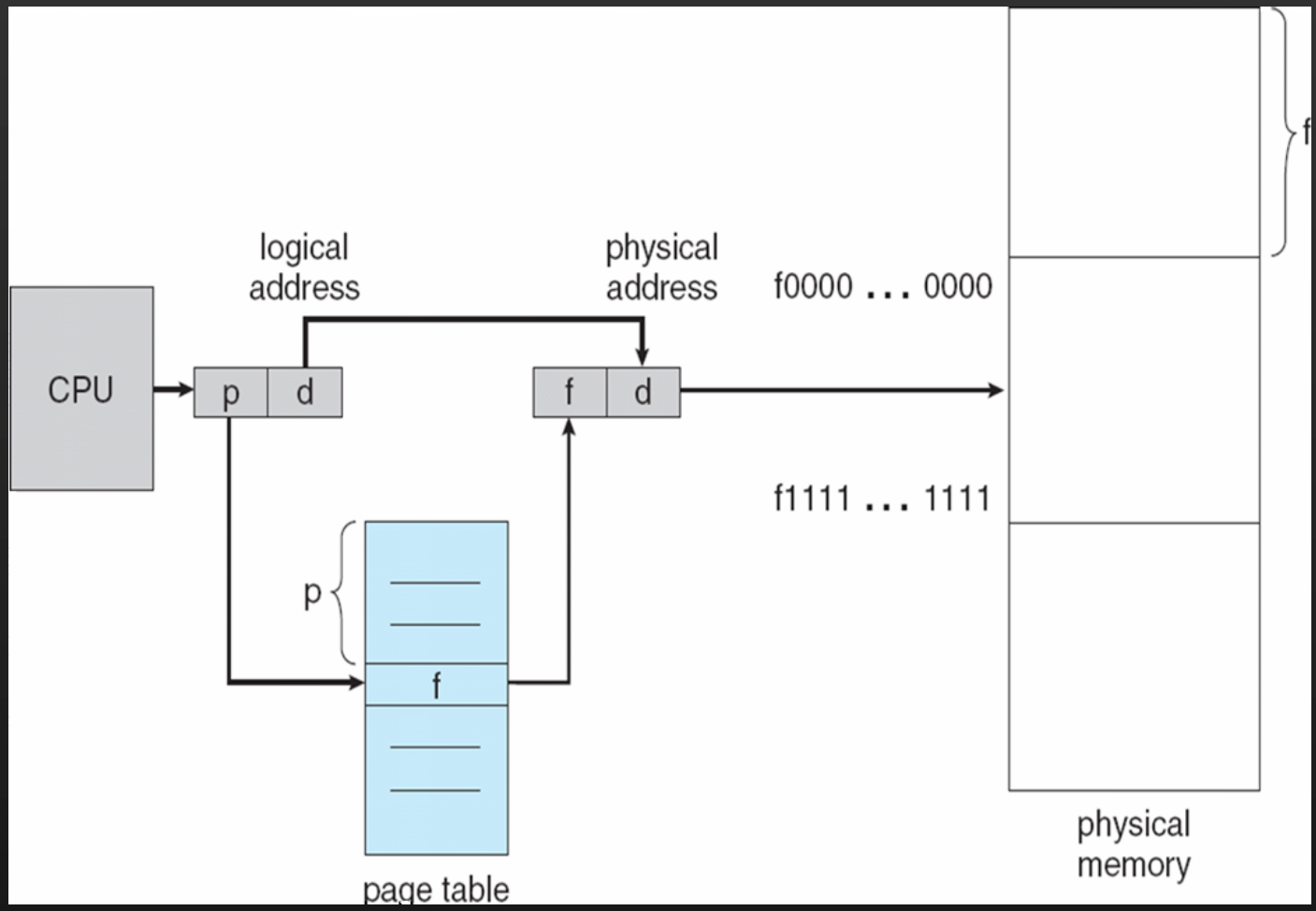
# Address translation scheme

- 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 by the MMU

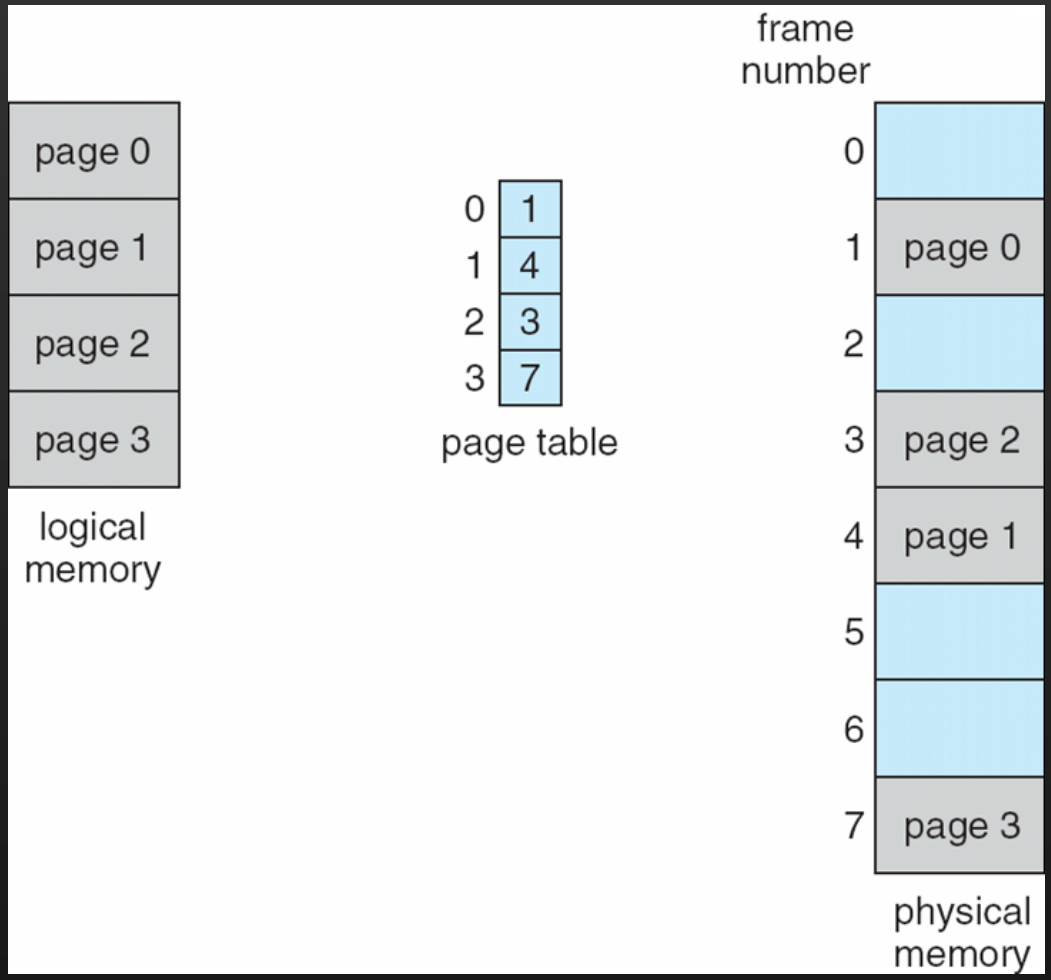



  - For given logical address space $2^l$ *and page size* $2^m$

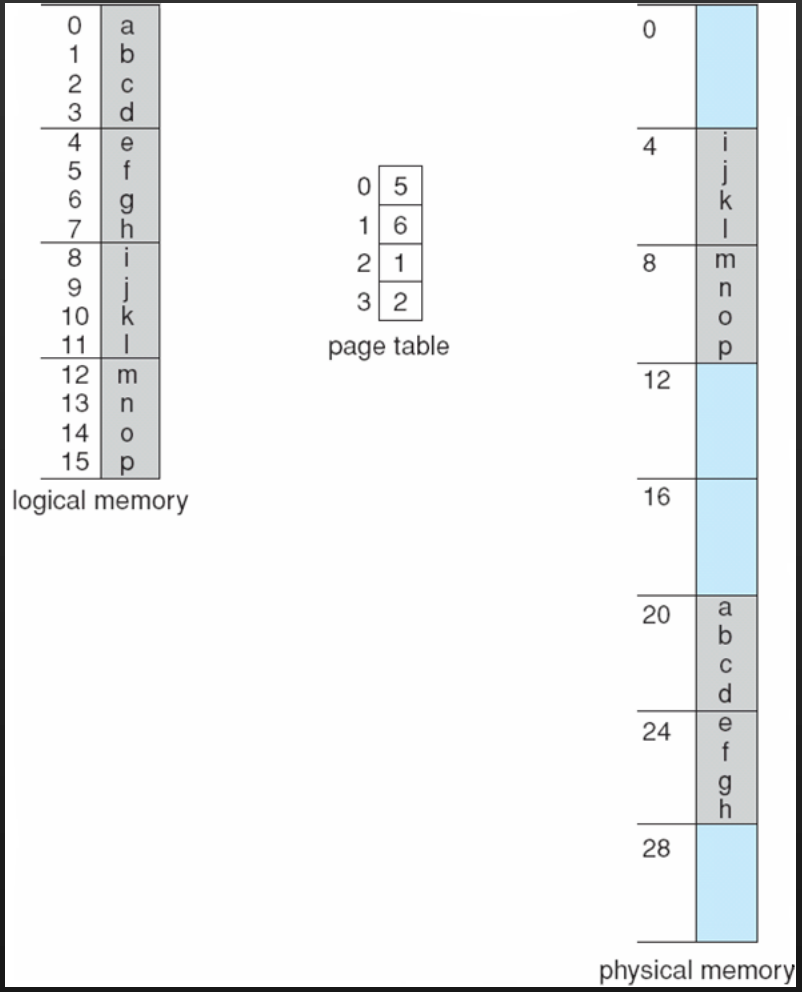
# Paging hardware



# Paging model: logical vs physical memory

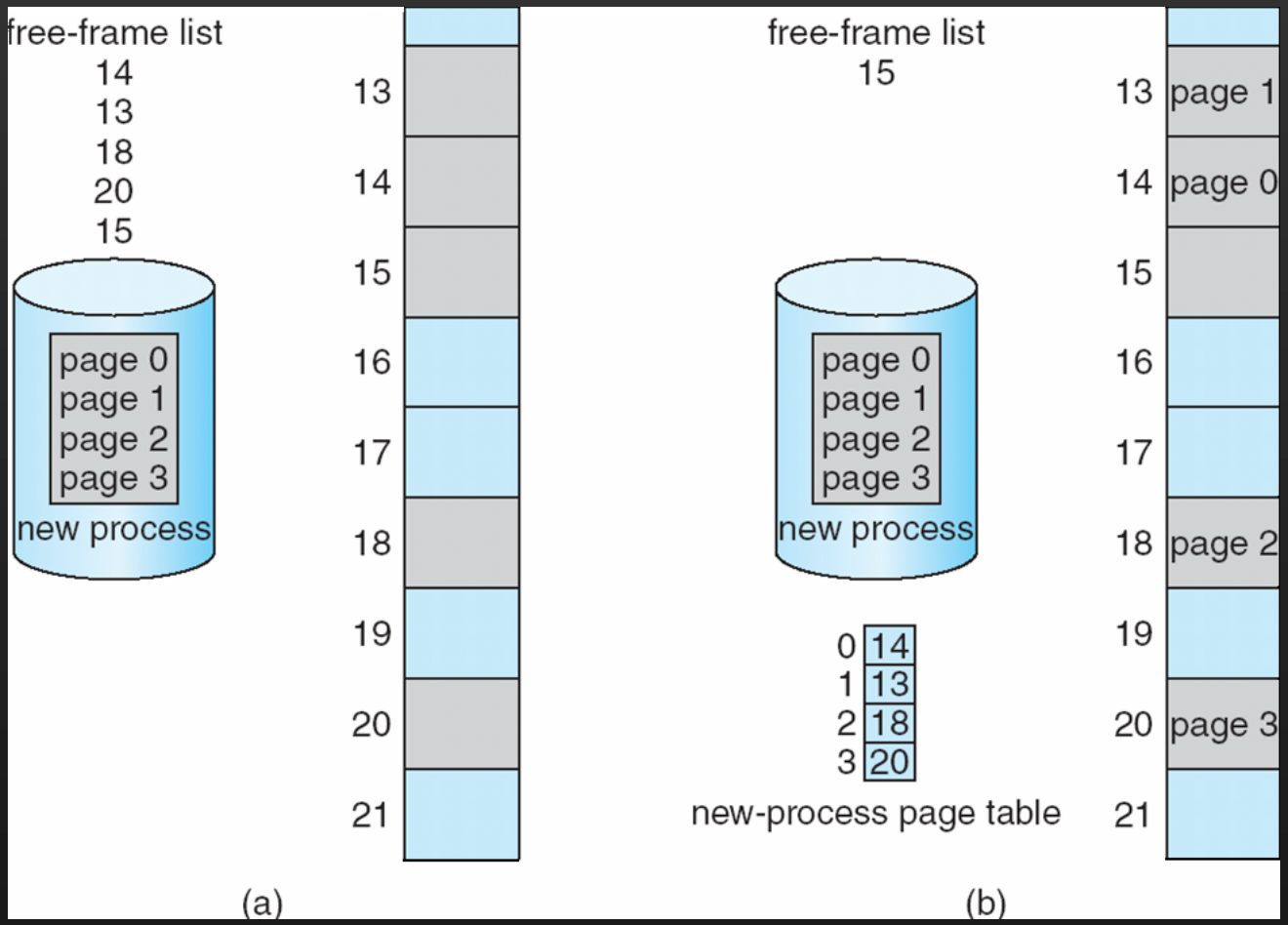


# Paging example




32-byte memory and 4-byte pages

# Free frames




Before allocation

After allocation

ROSE-HULMAN INSTITUTE OF TECHNOLOGY

# Implementation of page table

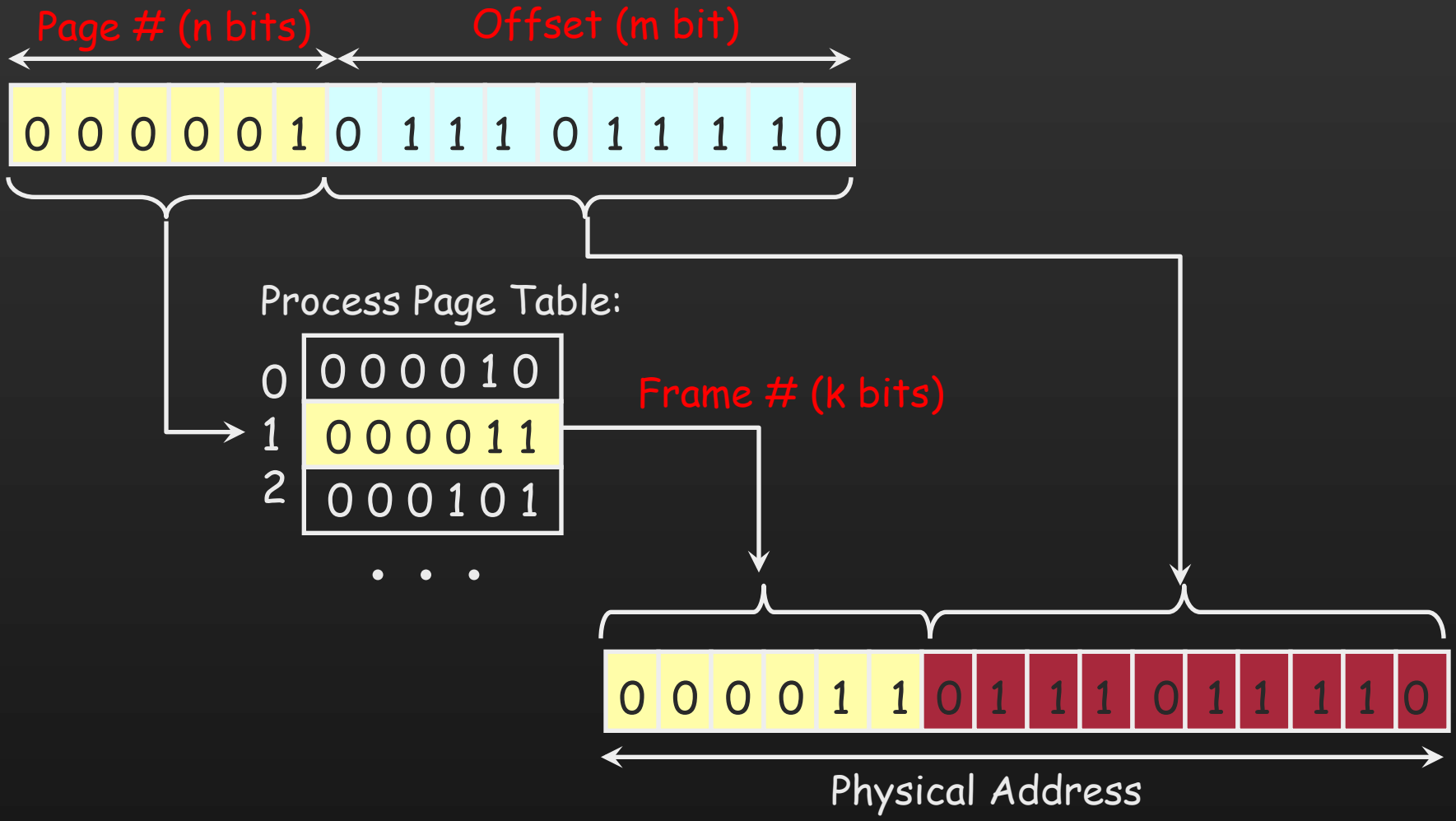
- Page table is kept in main memory
- **Page-table base register (PTBR)** points to the page table
- **Page-table length register (PRLR)** indicates size of the page table
- In this scheme every data/instruction access requires two memory accesses.
- The two memory access problem can be solved by the use of a special fast-lookup hardware cache called **associative memory** or **translation look-aside buffers (TLBs)**

# Address translation in paging

- To make it convenient, size of a page is a power of 2
- Given a (page #, offset), find (frame #, offset)
- $2^n$ > number of pages in process =>
  - n bits to hold the page number
- $2^k$ > number of frames in memory =>
  - k bits to hold the frame number
- $2^m$ = number of bytes in a page =>
  - m bits to determine the offset within a page
- **Logical address** is **nm**
- **Physical address** is simply **km** where k is frame that stores page "n" (from the page table)

ROSE-HULMAN INSTITUTE OF TECHNOLOGY

# Address translation in paging



# Address translation *e.g.*

| Page # | Logical Address | Value |
|---|---|---|
| 0 | 0 | A |
| 0 | 1 | B |
| 0 | 2 | C |
| 0 | 3 | D |
| 1 | 4 | E |
| 1 | 5 | F |
| 1 | 6 | G |
| 1 | 7 | H |
| 2 | 8 | I |
| 2 | 9 | J |
| 2 | 10 | K |
| 2 | 11 | L |
| 3 | 12 | M |
| 3 | 13 | N |
| 3 | 14 | O |
| 3 | 15 | P |

**Process**

| Page # | Frame # |
|---|---|
| 0 | 2 |
| 1 | 0 |
| 2 | 5 |
| 3 | 4 |

**Page table**

**Main memory**

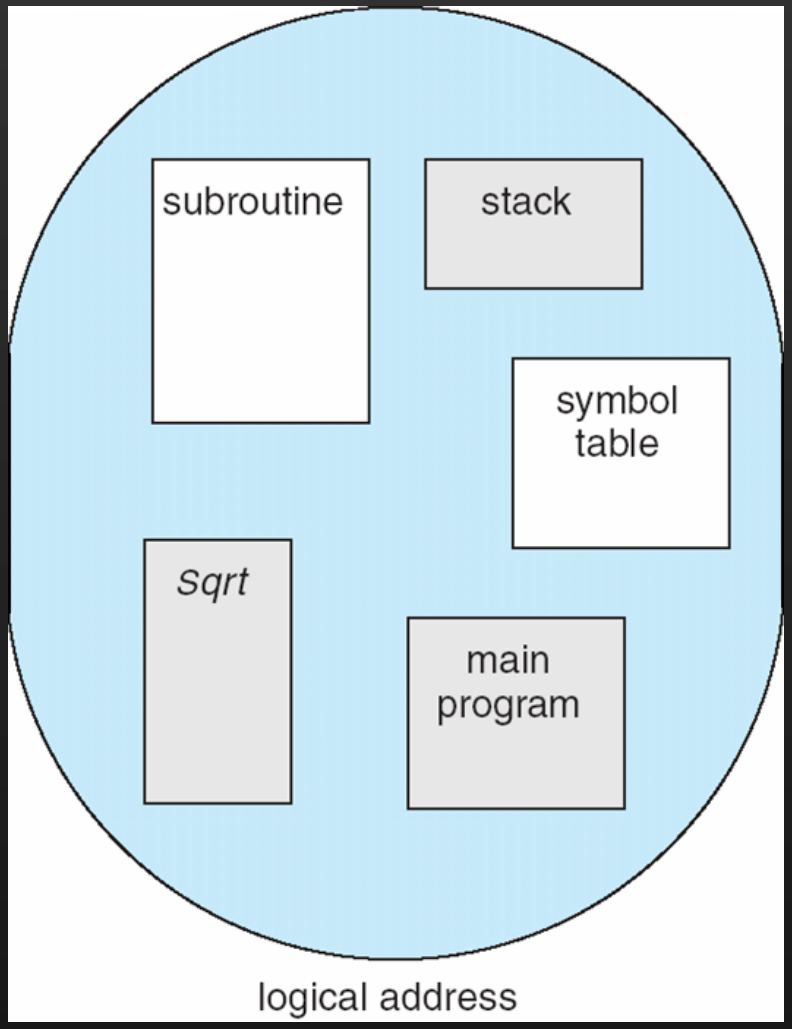
| Frame # | Physical address | Value |
|---|---|---|
| 0 | 0 | E |
| 0 | 1 | F |
| 0 | 2 | G |
| 0 | 3 | H |
| 1 | 4 | |
| 1 | 7 | |
| 2 | 8 | A |
| 2 | 9 | B |
| 2 | 10 | C |
| 2 | 11 | D |
| 3 | 12 | |
| 3 | 15 | |
| 4 | 16 | M |
| 4 | 17 | N |
| 4 | 18 | O |
| 4 | 19 | P |
| 5 | 20 | I |
| 5 | 21 | J |
| 5 | 22 | K |
| 5 | 23 | L |

ROSE-HULMAN INSTITUTE OF TECHNOLOGY

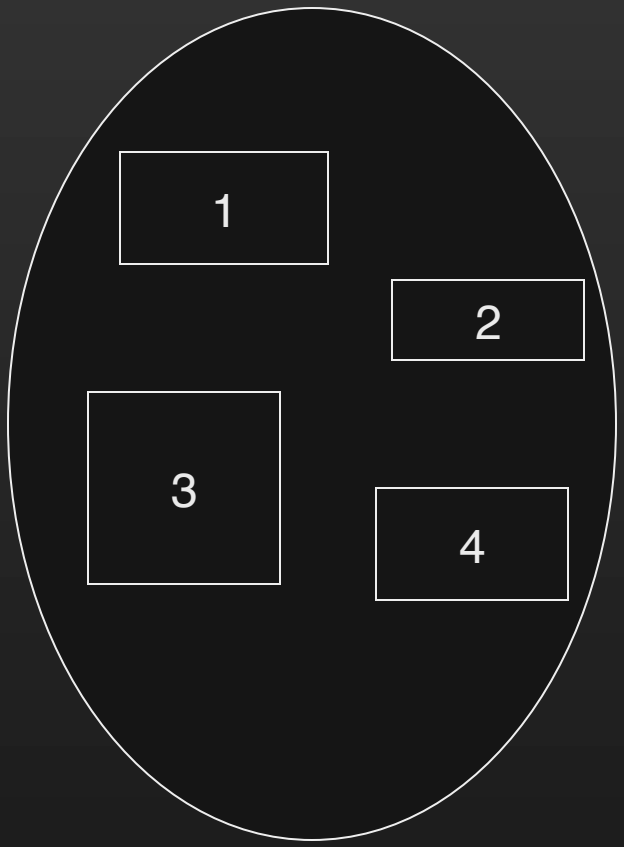
# Segmentation

- 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
  - object
  - local variables, global variables
  - arrays

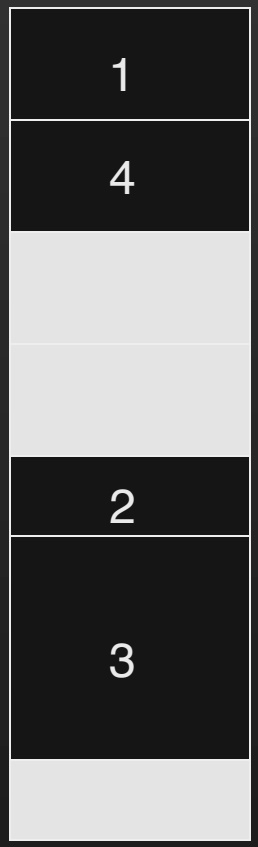
# User's view of a program



# Logical view of segmentation




user space

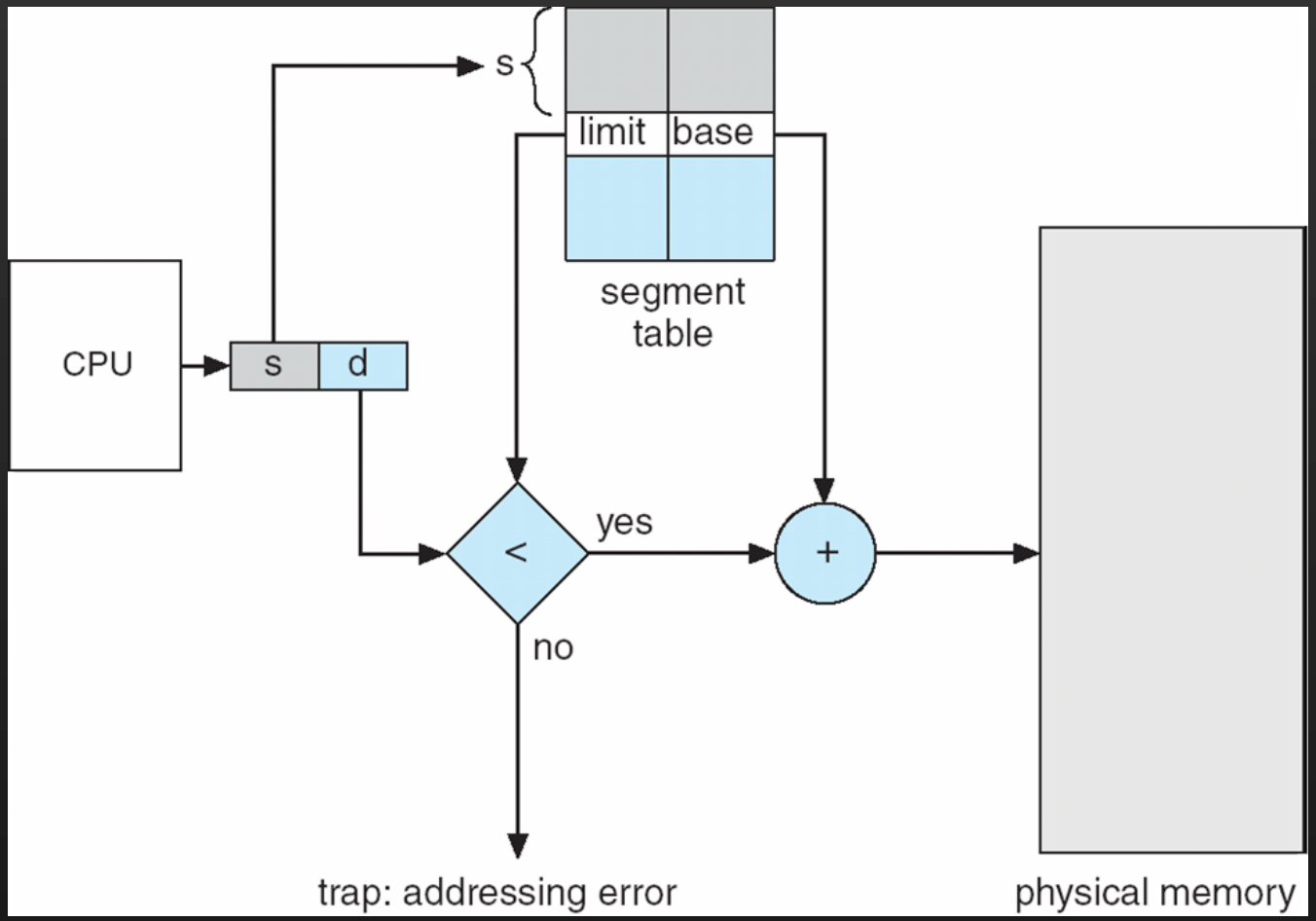



physical memory space

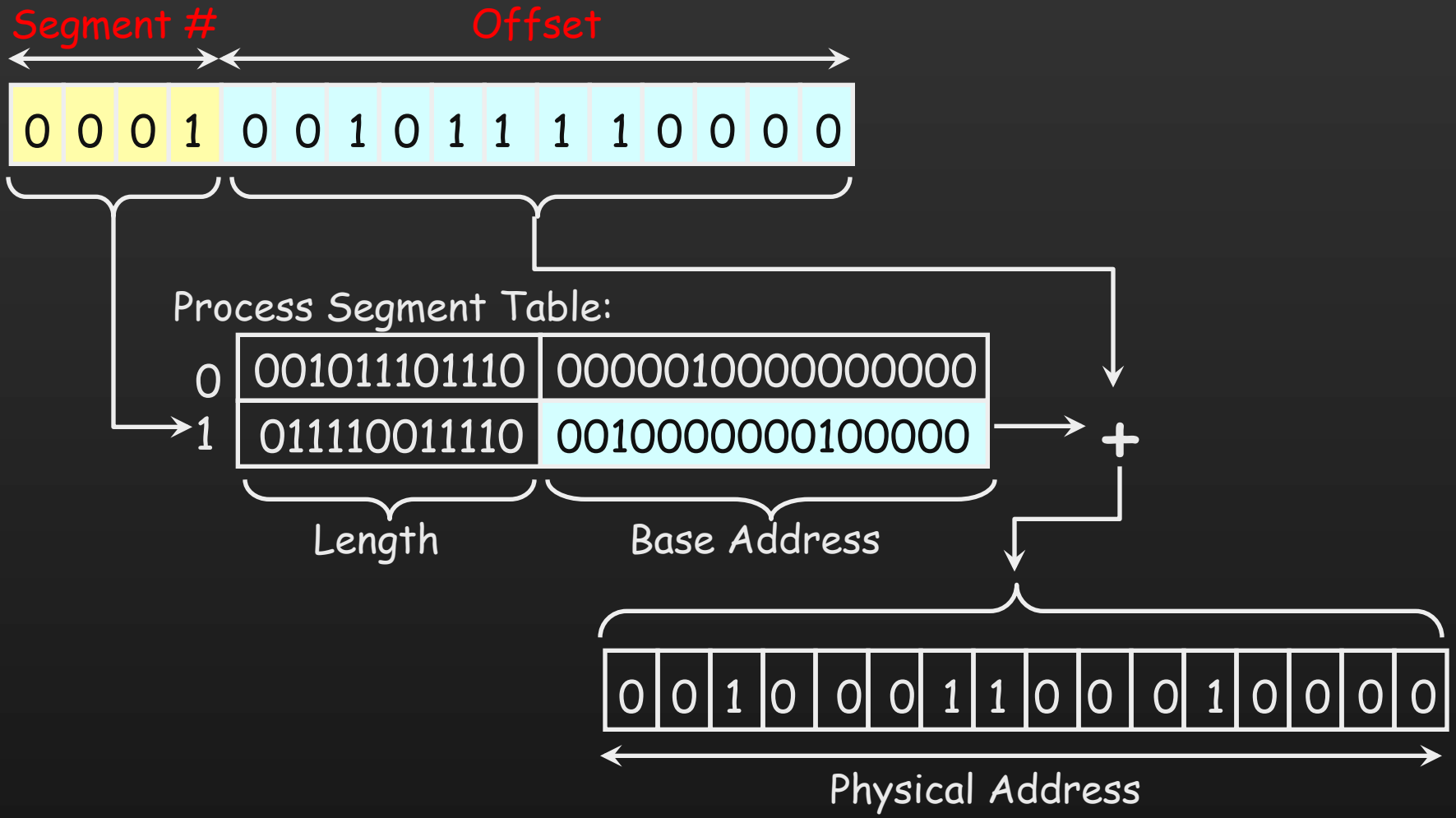
# Segmentation architecture

- 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 hardware



# Address translation in segmentation



# Example of segmentation