THE UNIVERSITY *of* EDINBURGH
informatics

Operating Systems (INFR10079)

2022/2023 Semester 2

Memory Management

(Non-contiguous Allocation)

abarbala@inf.ed.ac.uk

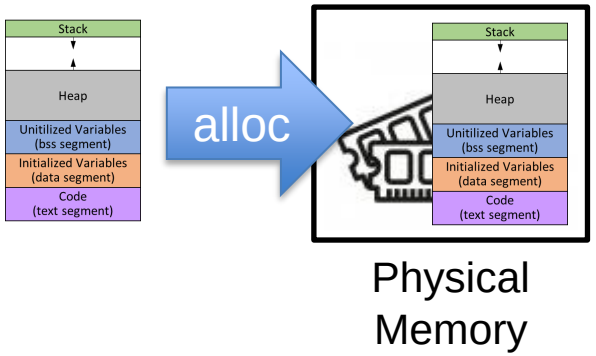
Chapter 9.3, 9.4, 9.6

Overview

- Non-contiguous memory allocation
- Segmentation
  - Basics
- Paging
  - Basics
  - Page Tables
  - TLB
  - Memory Protection
  - Shared Pages
  - Hierarchical Pages
  - Hashed Pages
  - Inverted Pages
  - Use cases

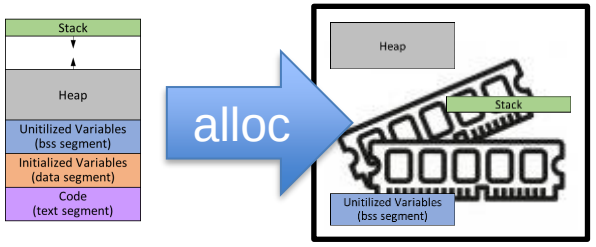
Memory Allocation

- **Contiguous**
  - Allocation granularity is entire logical address space
  - *Request physical space* for the **entire program at once**
    - **Contiguous** in logical memory and physical memory
  - Issues
    - Fragmentation
    - Long compaction times (ext. Frag.)
    - Long swap times (ext. Frag.)



Physical Memory

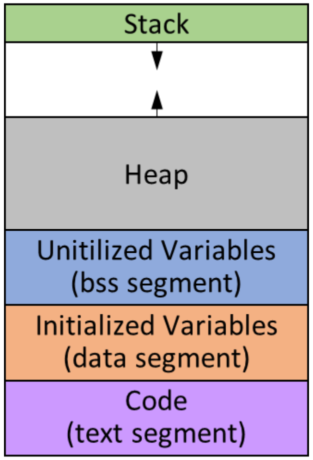
- **Non-contiguous**
  - Split logical address space in chunks
  - *Request physical space* for each **program's chunk at the time**
    - **Contiguous** in logical memory, **non-contiguous** in physical
  - Key mechanisms
    - Segmentation
    - Paging



Physical Memory

Segmentation

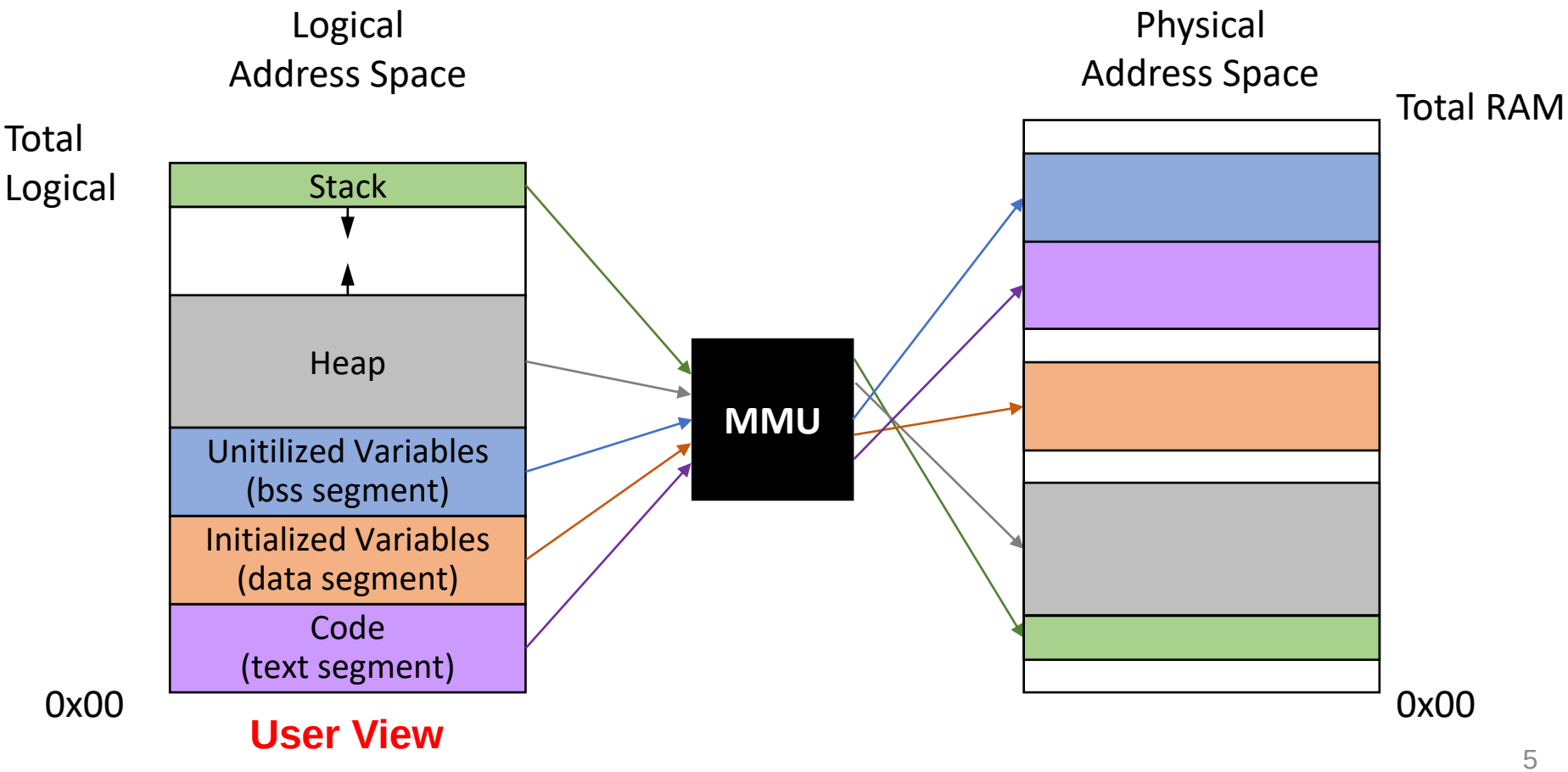
- Segmentation
  - Partition an address space into **variable size** chunks/units
  - *Logical units*
    - Stack, heap, data, code, subroutines, …
    - With associated segment #
  - A **logical address** is <segment #, offset>



- Facilitates sharing and reuse
  - Segment is a natural unit of sharing
    - e.g., shared library

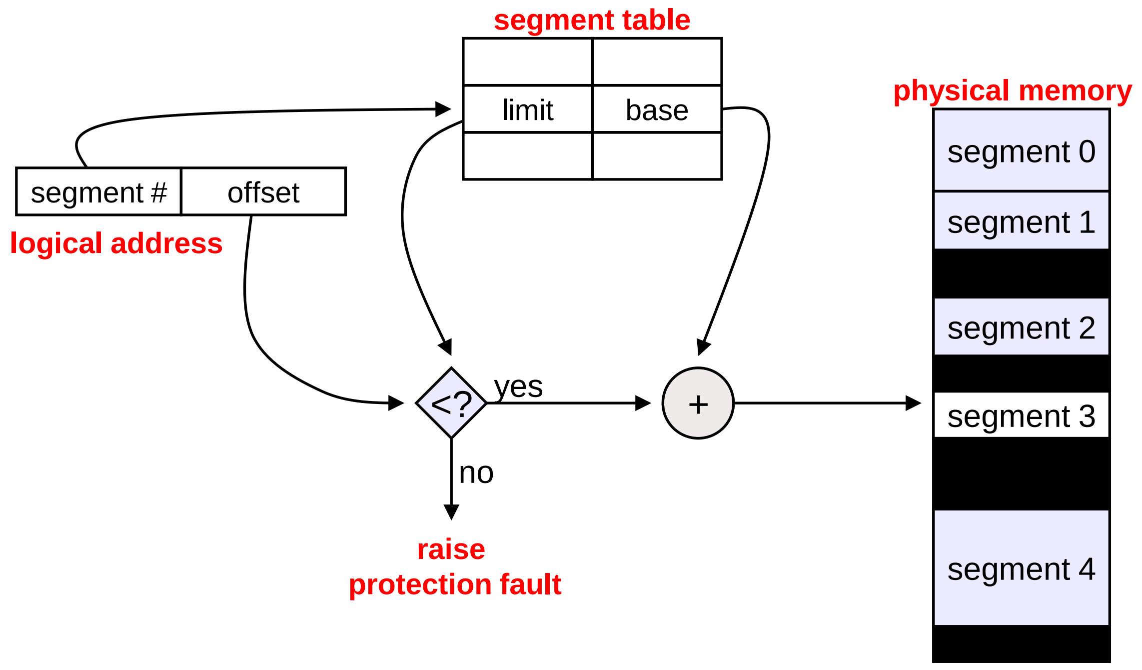
Segmentation

- *Logical address* space divided in **variable size** chunks



Hardware Support

- Segment table
  - Multiple base/limit pairs, one per segment
  - Segments named by **segment #**, used as index into table
    - A logical address is <segment #, offset>
  - Physical address = logical address offset + segment base address



Pros and cons

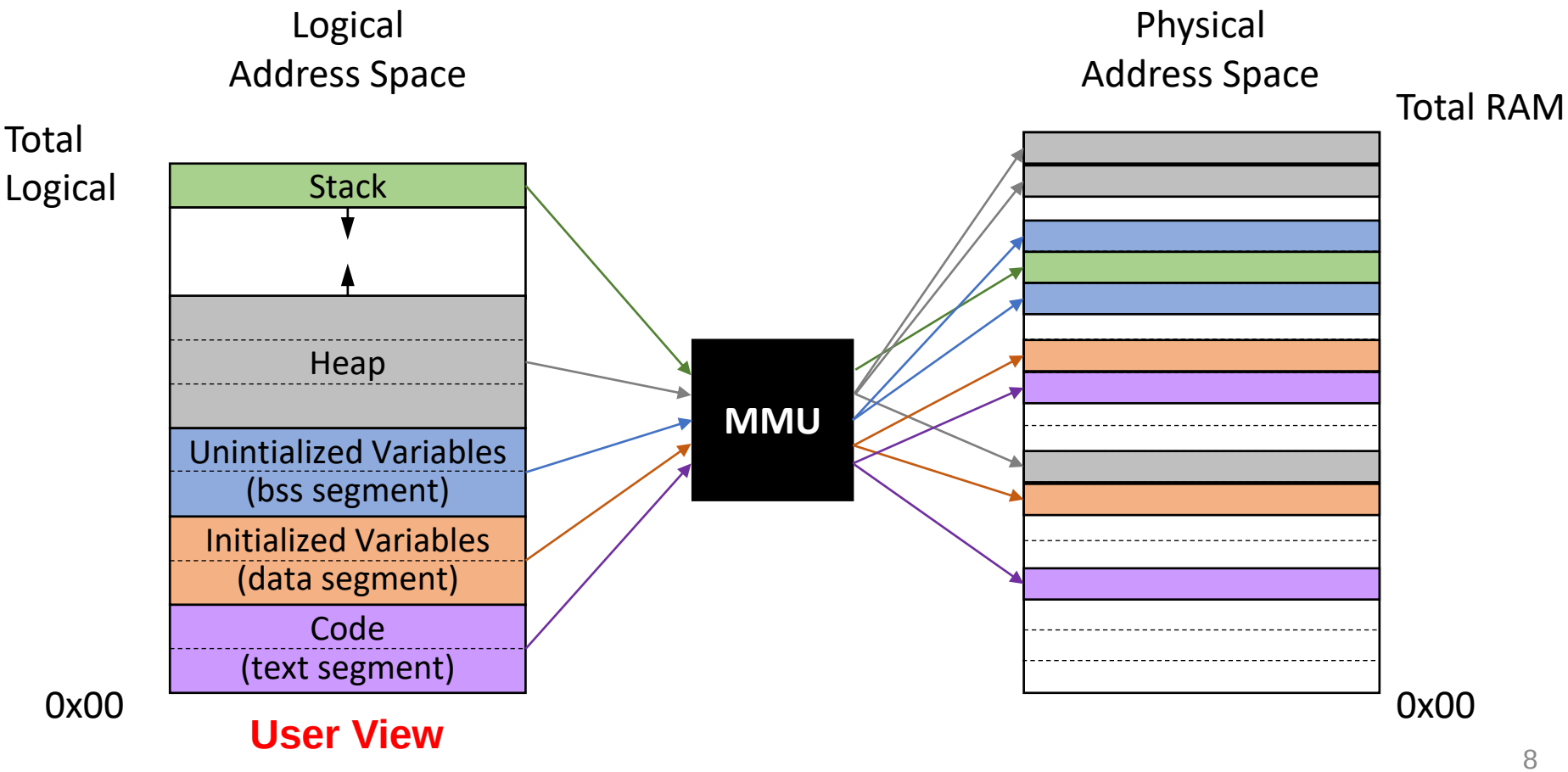
- Allows non-contiguous physical addresses
  - Allocated “chunks” are smaller than entire program address space
  - Reduce fragmentation by exploiting varying sized holes
- Enables sharing
  - Same segment can be shared across processes

- Process view and physical memory **very different**
- By implementation, process can **only access** its own memory

- Segmentation rarely used today
  - x86 CPUs doesn’t support it on 64bit models

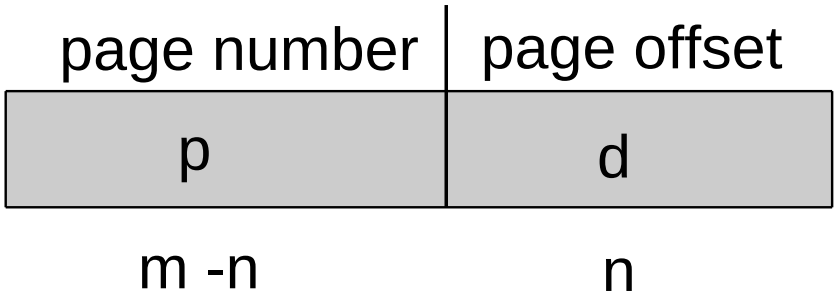
Paging

- *Logical address* space divided in **fixed size** chunks
  - *Physical address* space divided in **fixed size** chunks, *same size*



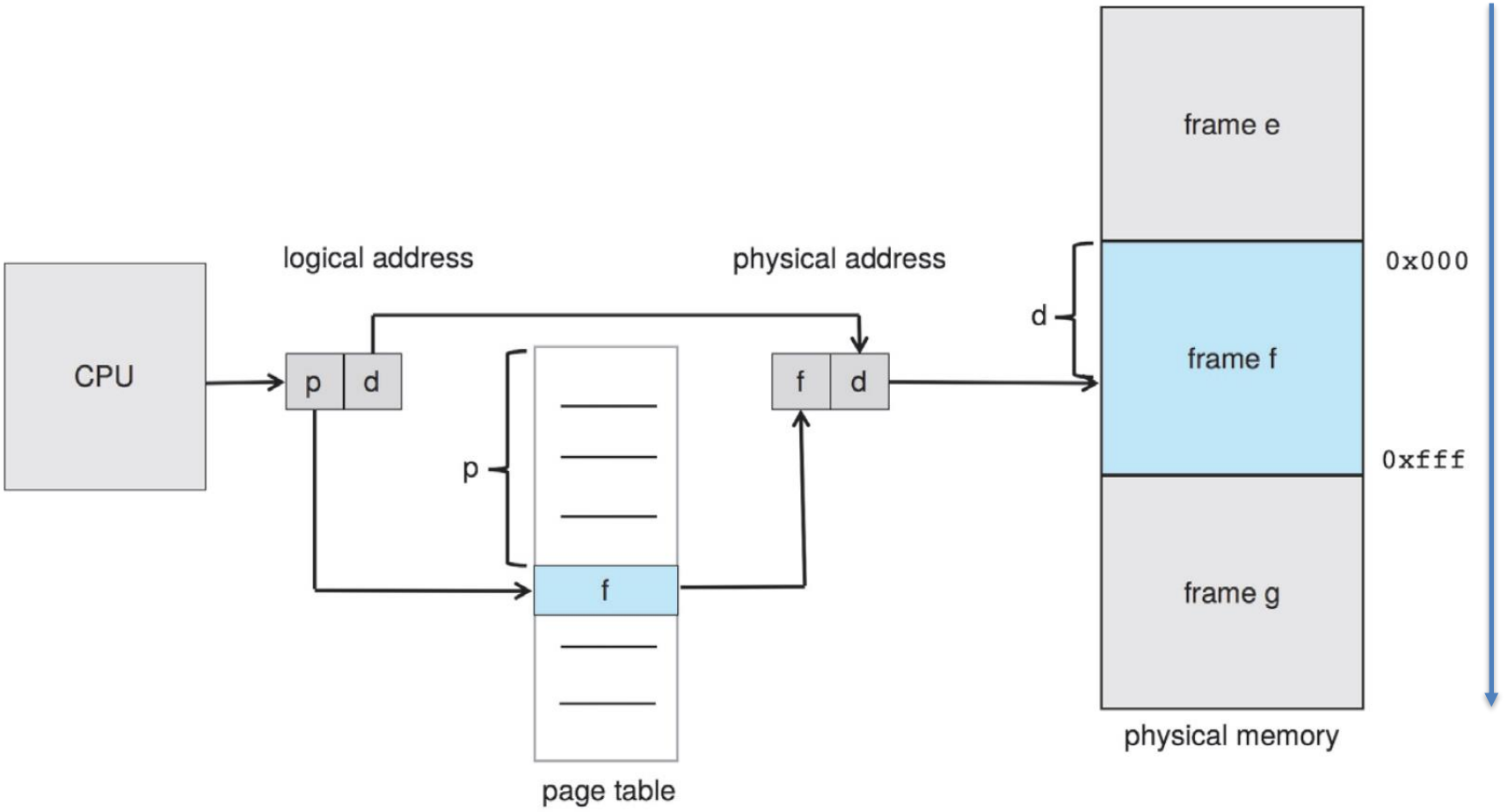
Address Translation Scheme

- Map **logical** chunks (pages) to **physical** chunks (frames)
  - **Page** size **== frame** size
  - Chunks are at a **predefined fixed** (logical and physical) address
- **Logical address** is divided into
  - **Page Number** (***p***)
    - Index into a **page table** which contains **frames base address**
  - **Page Offset** (***d***)
    - Summed to **frame base address** to make **physical** memory address



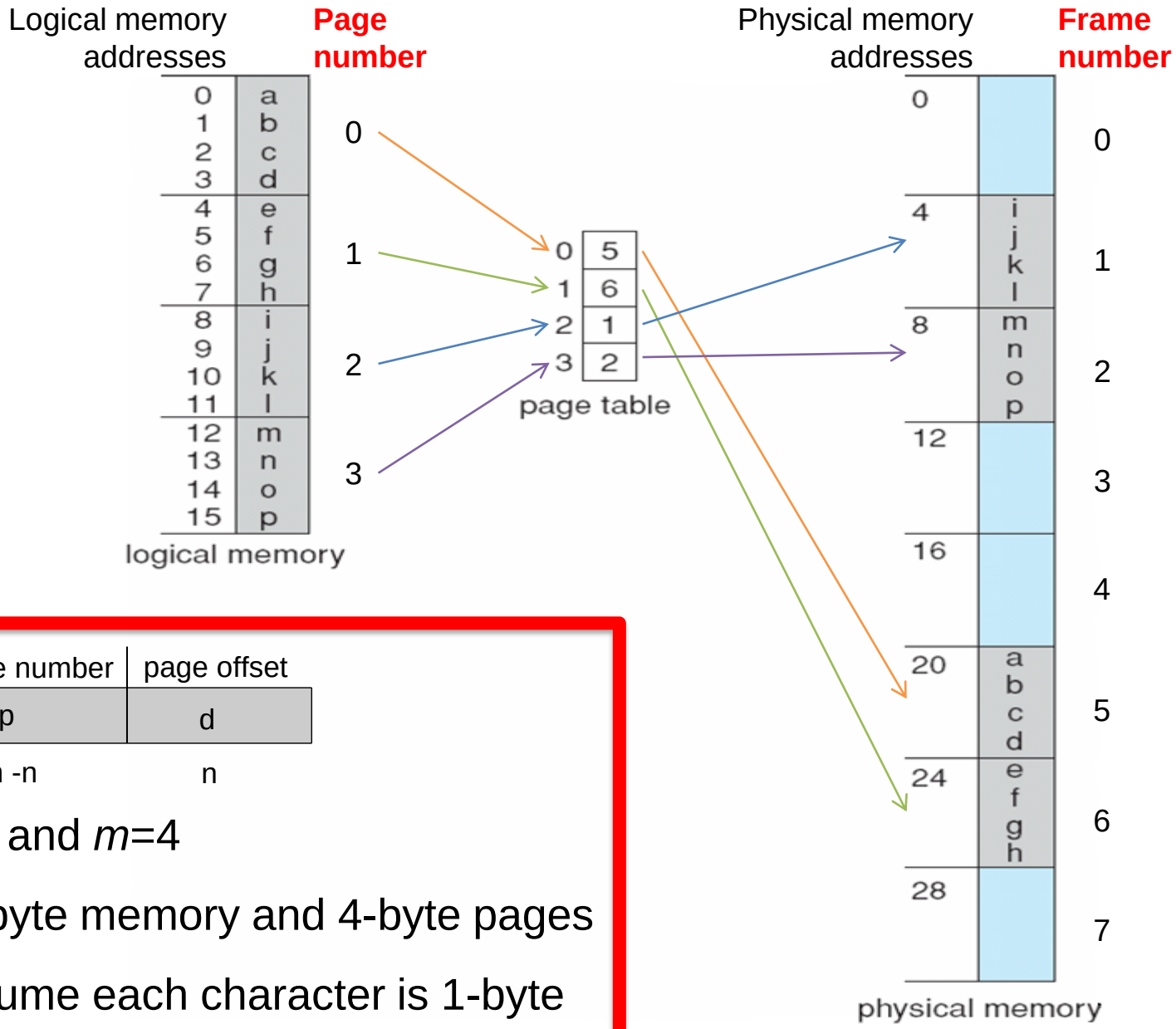
  - Logical address space size 2^m **bytes**
  - Page and frame size 2^n **bytes**

Paging Hardware



Page table: array of **page table entries**
Page table entry (PTE): frame base address and bits/flags

Paging Example



page number	page offset
p	d
m -n	n

n=2 and *m*=4

32-byte memory and 4-byte pages

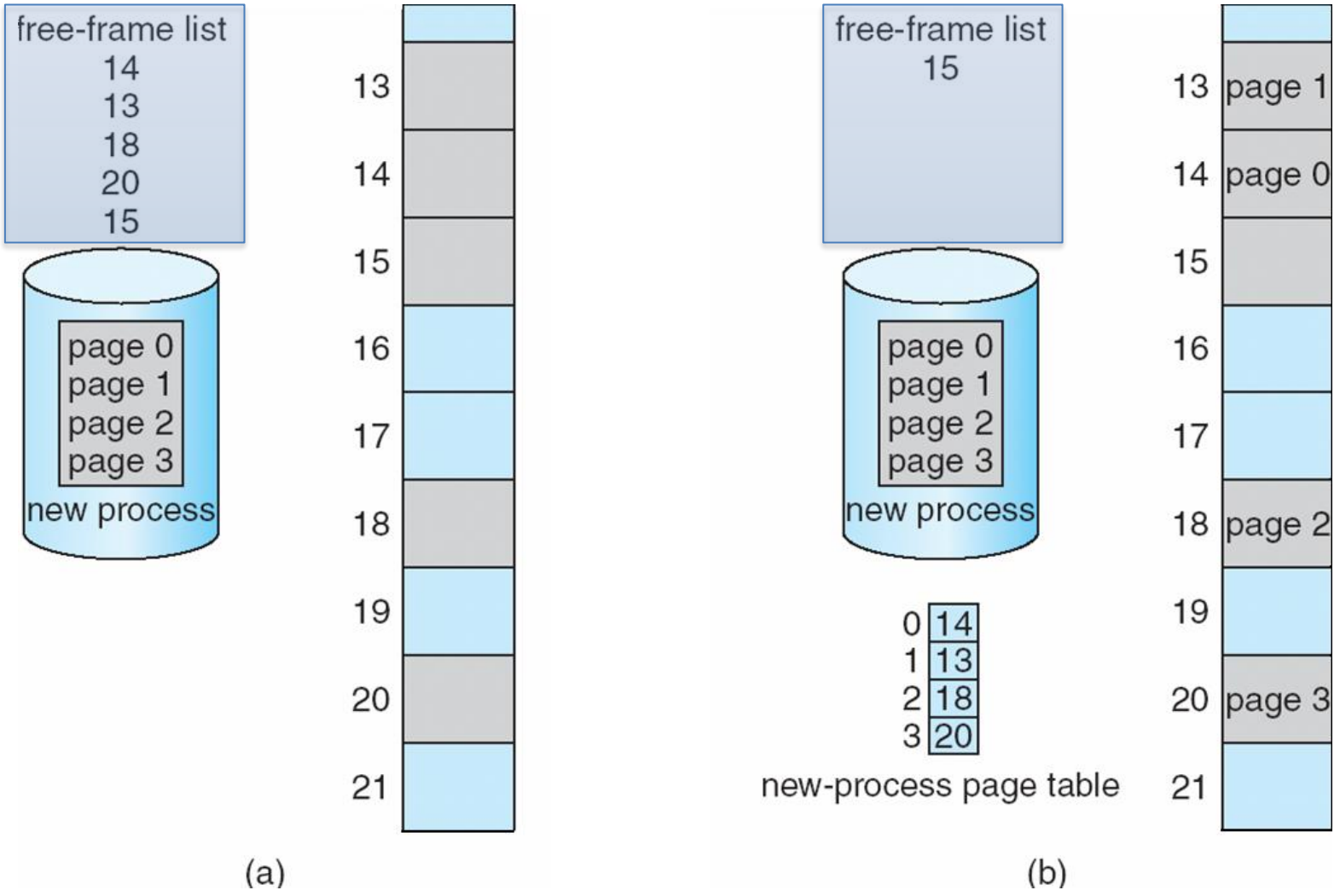
Assume each character is 1-byte

Advantages with Paging

- **No external** fragmentation
- **Internal** fragmentation **depends on page size**
  - Page size = 2,048 bytes, process size = 72,766 bytes
  - 35 pages (71,680 bytes) + 1,086 bytes
  - Internal fragmentation of 2,048 - 1,086 = 962 bytes
  - *Worst case* fragmentation = 1 frame – 1 byte
  - *On average* fragmentation = .5 frame size
  - *Are small frames desirable?*
    - Translations may require memory accesses (costly)
    - Different page sizes available on every system

- Process view and physical memory **very different**
- By implementation, process can **only access** its own memory

Free Frames



Before allocation

After allocation

Paging: Many Processes