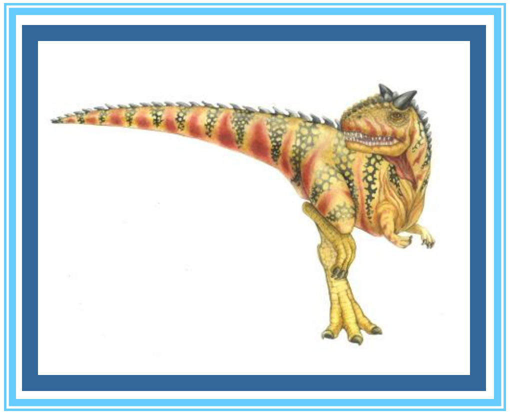
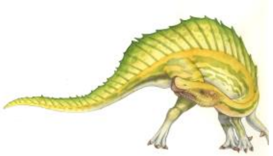
Memory Management (contd.)

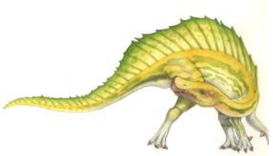


- **Slides mostly borrowed from Silberschatz & Galvin**
  - With occasional modifications from us



Memory Management: Topics

- Background
- Swapping
- Contiguous Memory Allocation
- Segmentation
- Paging
- Structure of the Page Table



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

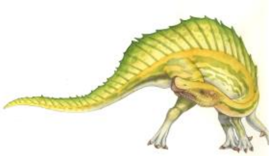
    method

    object

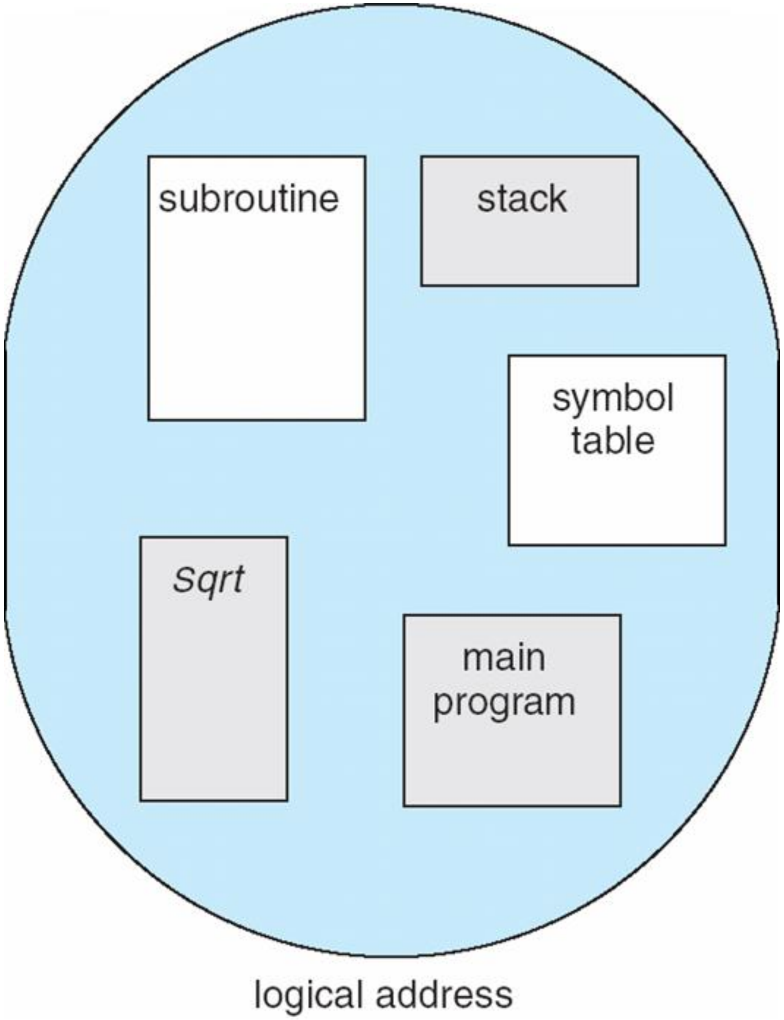
    local variables, global variables

    stack

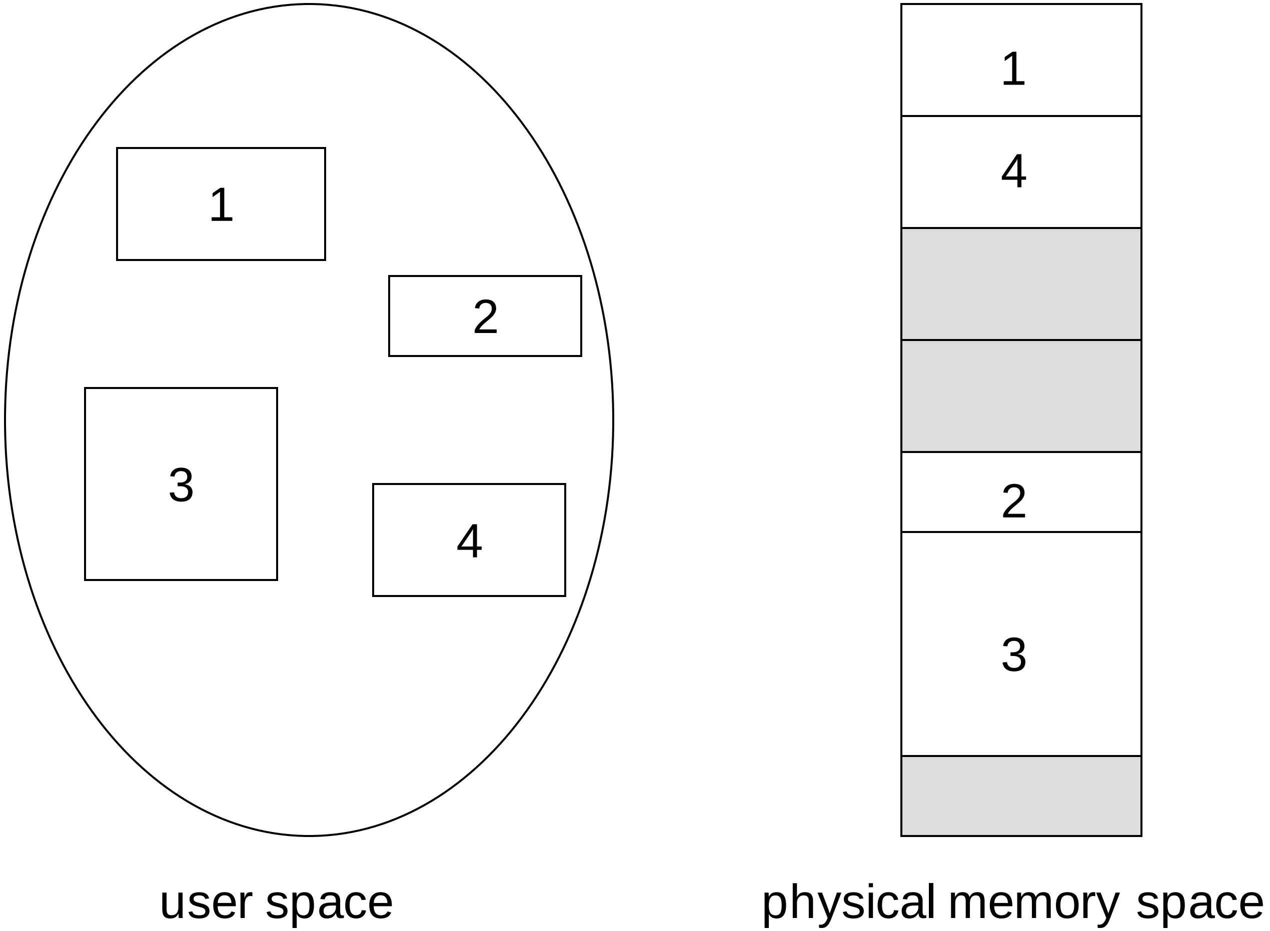
    arrays



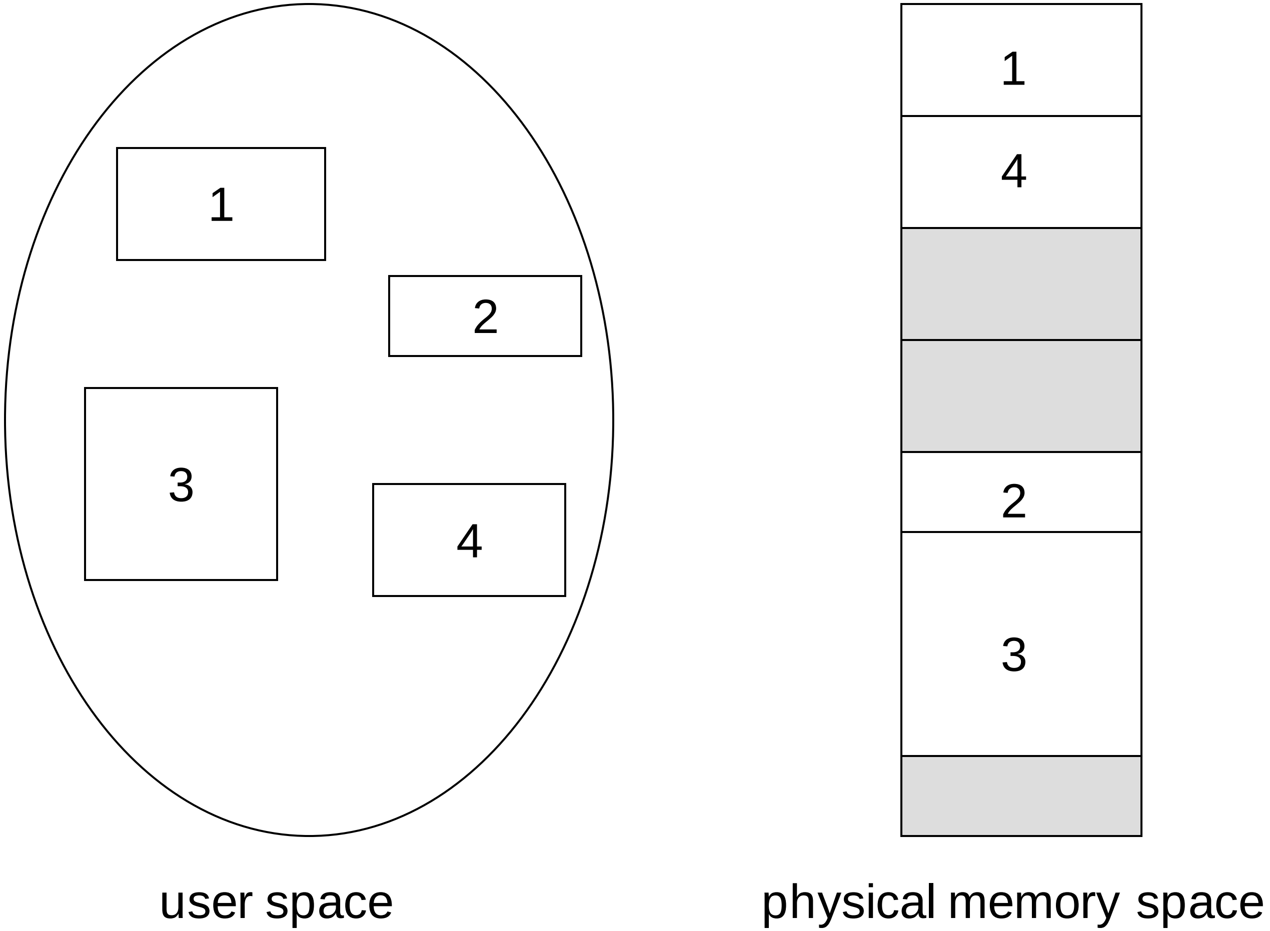
User’ s View of a Program



Logical View of Segmentation



Logical View of Segmentation



Note: we are shifting to non-contiguous memory allocation

Segmentation Architecture

- Logical address consists of a two-tuple:

  <segment-number, offset>

- **Segment table** – used to map logical addresses to physical addresses

- Segment table has one entry for each segment of this process
- Each segment table entry has:
  - **base** – contains the starting physical address where the segment resides in memory
  - **limit** – specifies the length of the segment
  - Two more fields – see next slide

Segmentation Architecture (Cont.)

- Protection
  - With each entry in segment table associate:
    - valid bit -- 0 indicates presently invalid segment
    - read/write/execute access privileges of the segment
  - Protection bits associated with segments -- code sharing occurs at segment level
- Since segments vary in length, memory allocation is a dynamic storage-allocation problem

Segment Table

Base	Limit	Valid / Invalid	Access Privileges

Segmentation Architecture

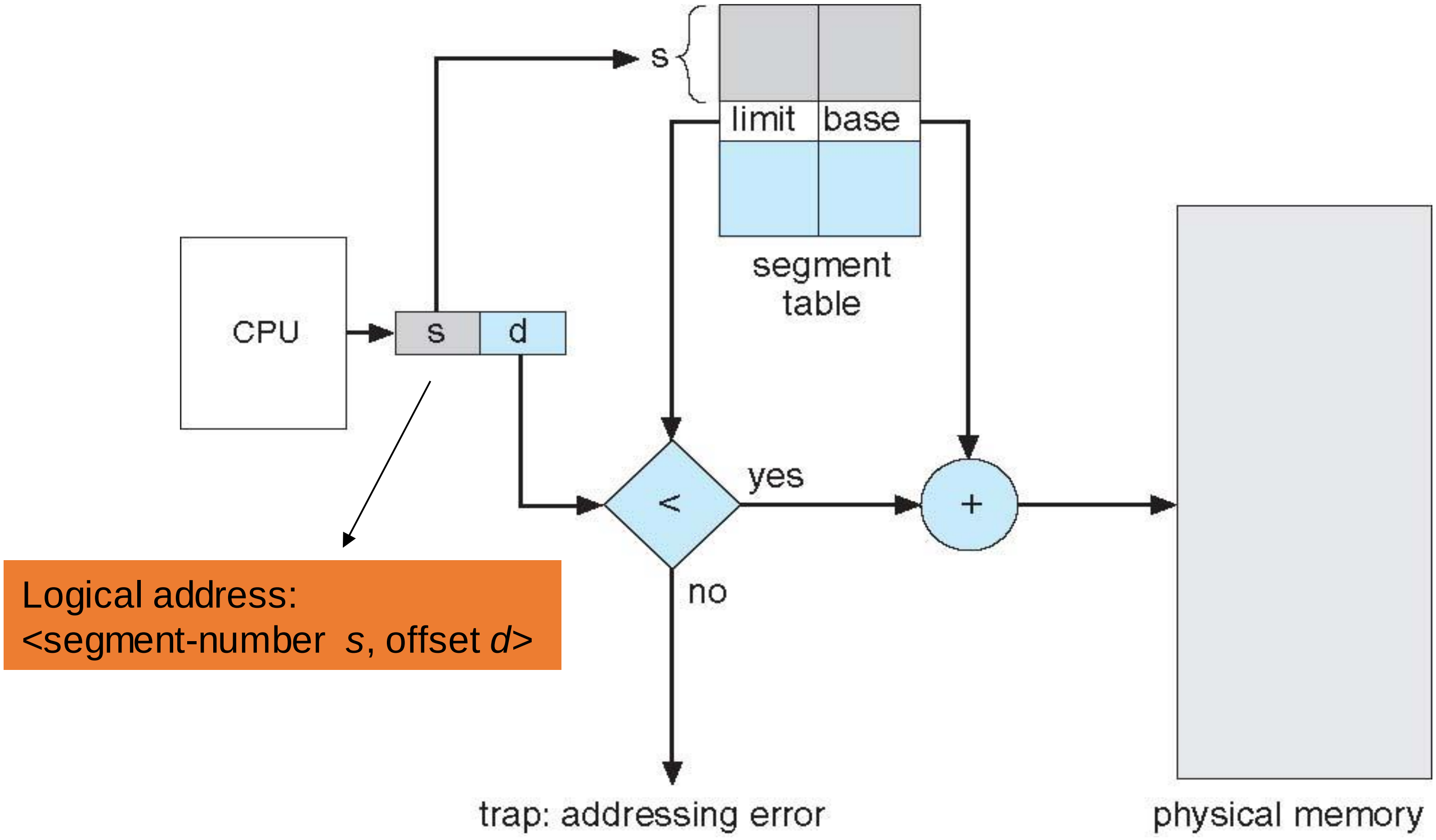
- Logical address consists of a two-tuple: <segment-number *s*, offset *d*>

- **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**

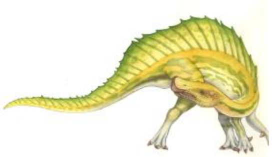
Segmentation Hardware



Logical address:
<segment-number *s*, offset *d*>

Memory Management: Topics

- Background
- Swapping
- Contiguous Memory Allocation
- Segmentation
- Paging
- Structure of the Page Table

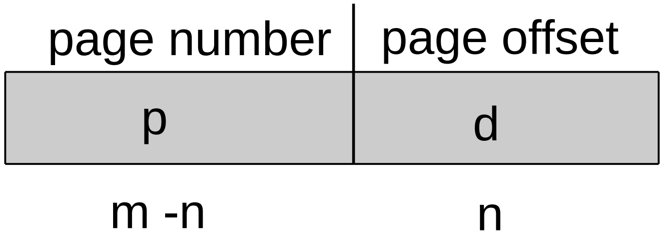


Paging

- Divide physical memory into fixed-sized blocks called **frames**
  - Size is power of 2, between 512 bytes and 16 MB
  - Typical size: 32KB, 4 MB
- Divide logical memory into blocks of same size called **pages**
- Physical address space of a process can be noncontiguous; process is allocated physical memory in units of a page wherever available
  - Avoids external fragmentation
  - Can still have Internal fragmentation
  - Avoids problem of varying sized memory chunks
- OS keeps track of all free frames
- To run a program of size ***N*** pages, need to find ***N*** free frames and load program (frames need not be contiguous)
- Set up a **page table** to translate logical to physical addresses

Address Translation Scheme

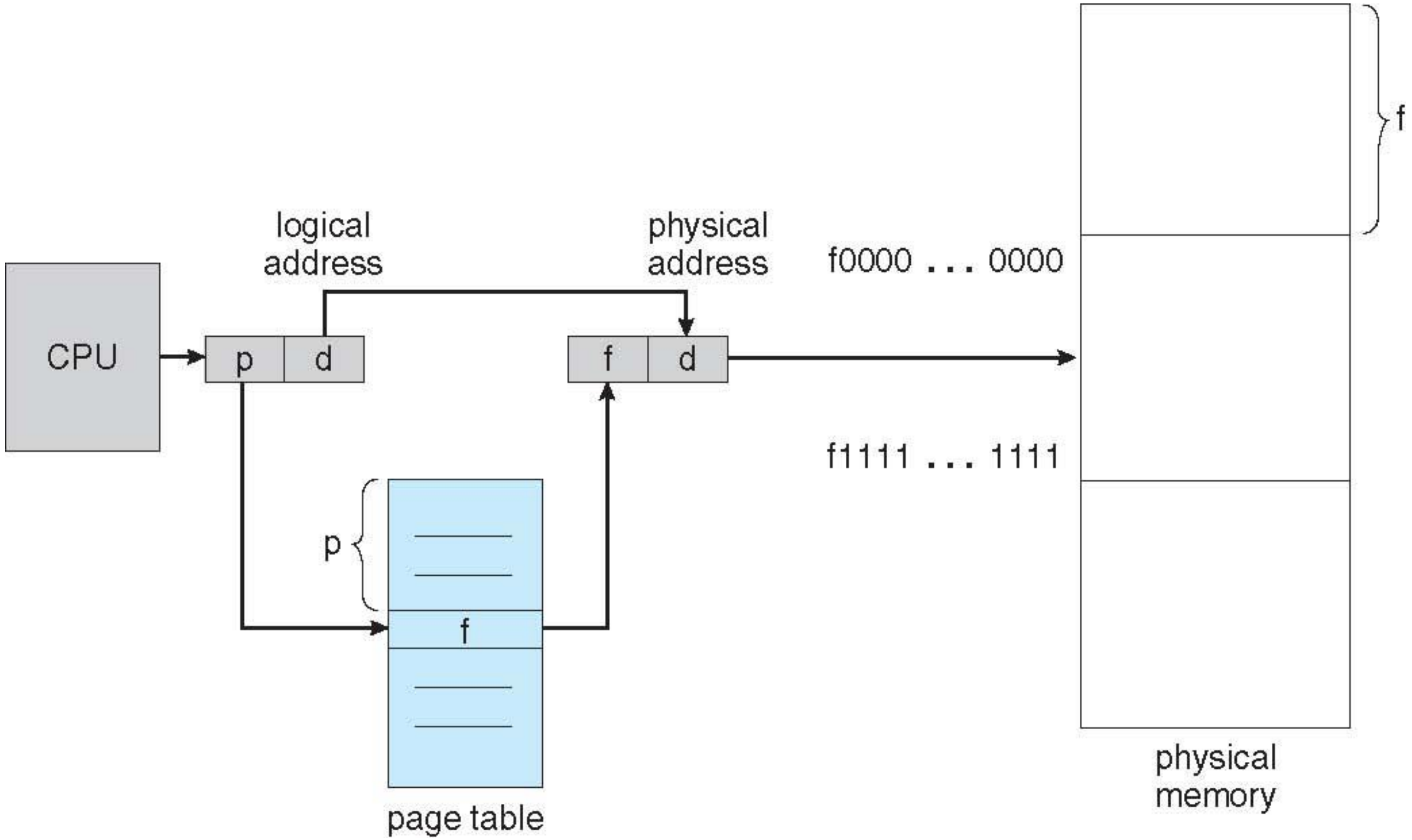
- Address generated by CPU is divided into:
  - **Page number** (*p*) – used as an index into a **page table** which contains base address of each page in physical memory
  - **Page offset** (*d*) – combined with base address to define the physical memory address that is sent to the memory unit



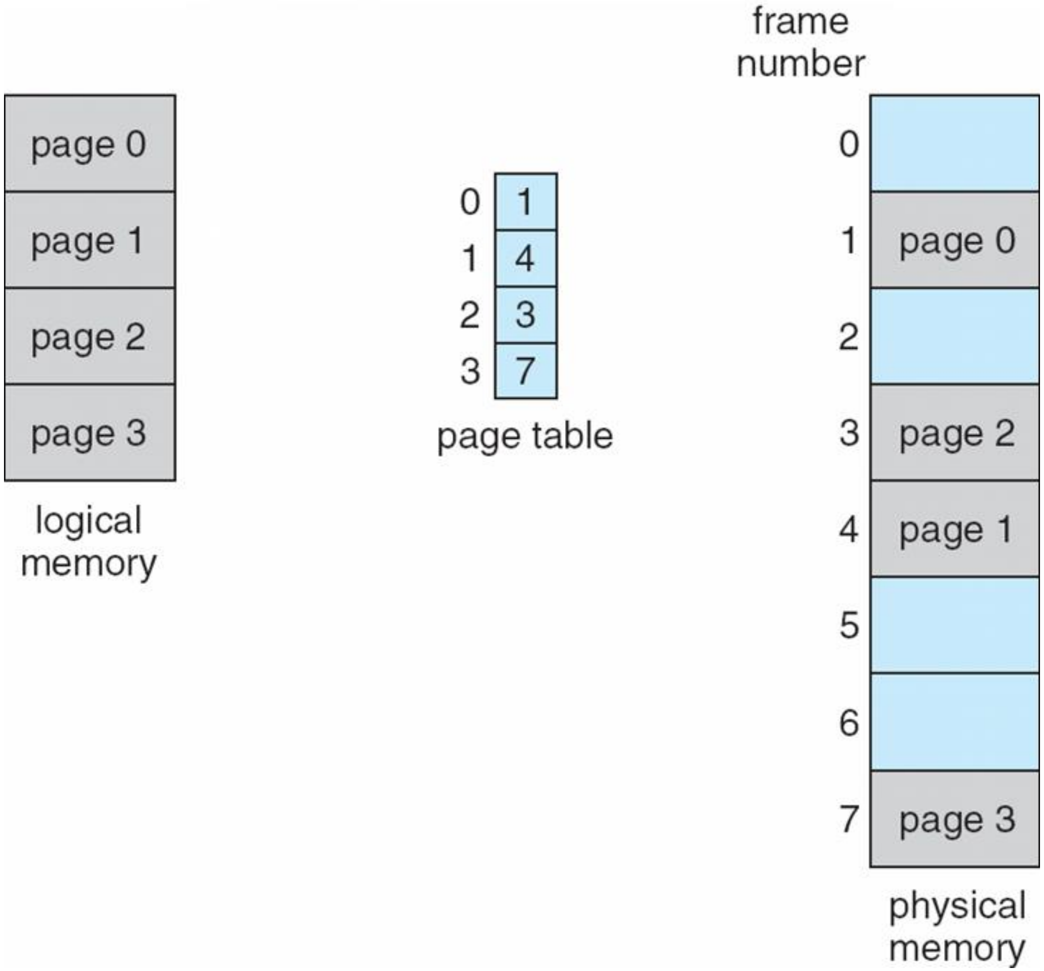
page number	page offset
p	d
m -n	n

  - For given logical address space 2^m and page size 2^n

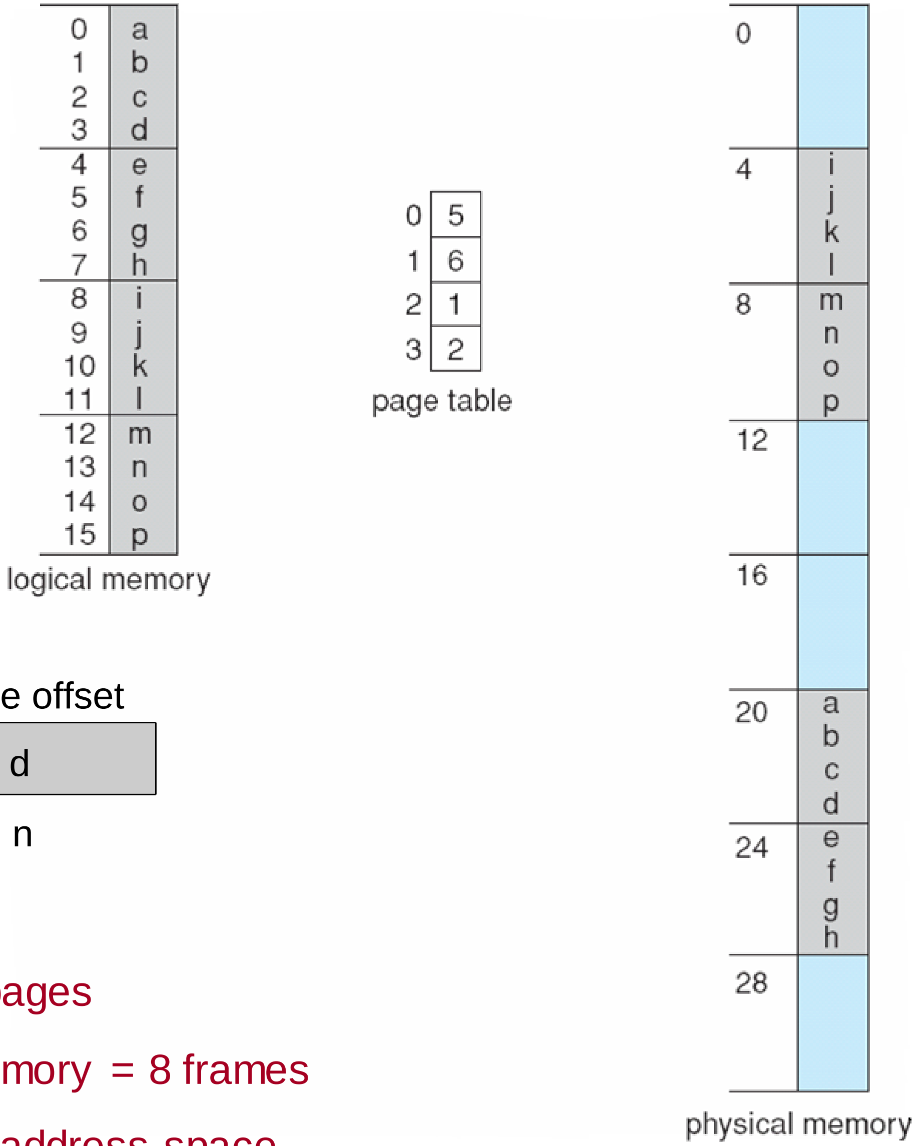
Paging Hardware



Paging Model of Logical and Physical Memory



Paging Example



page number	page offset
p	d
m -n	n

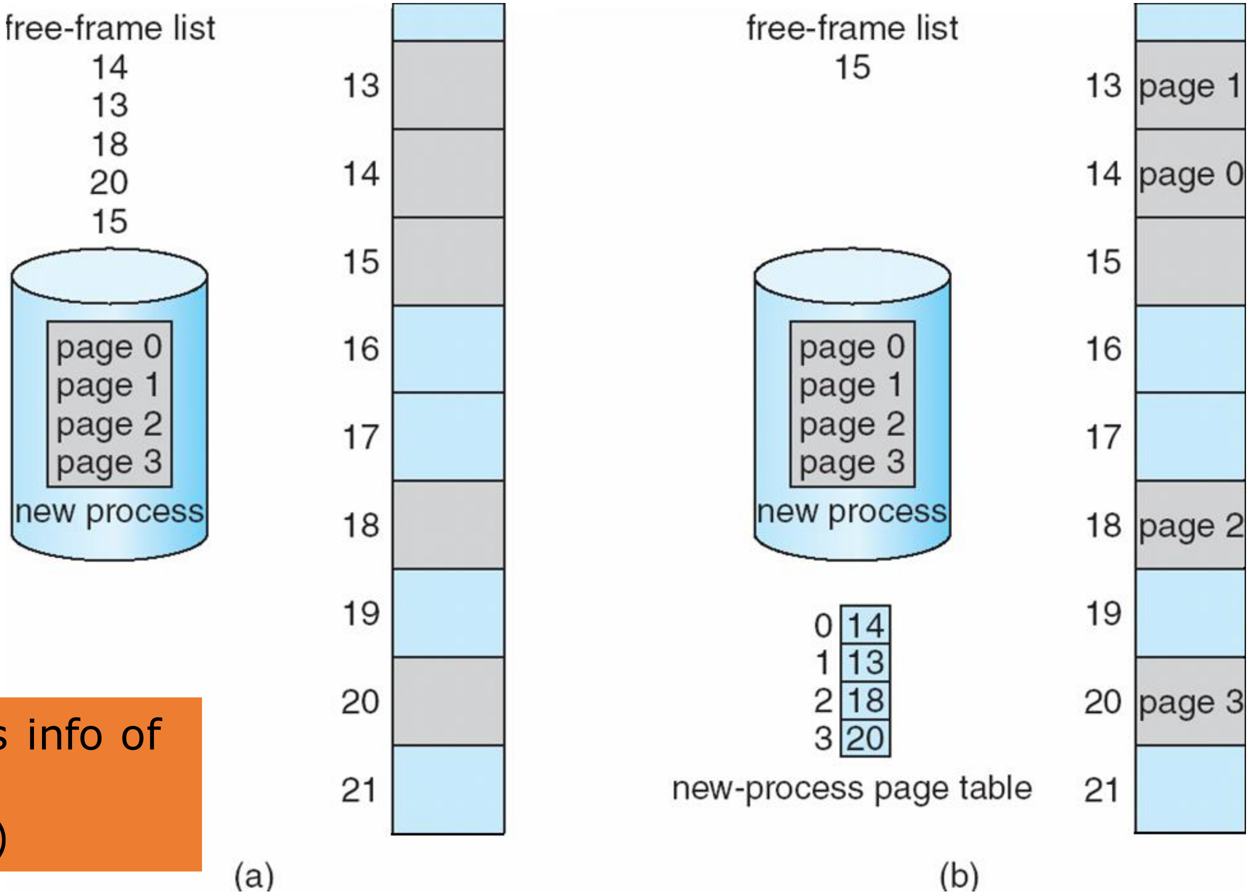
n=2 and *m*=4

n=2 means 4-byte pages

32-byte physical memory = 8 frames

2^4 = 16-byte logical address space

Free Frames



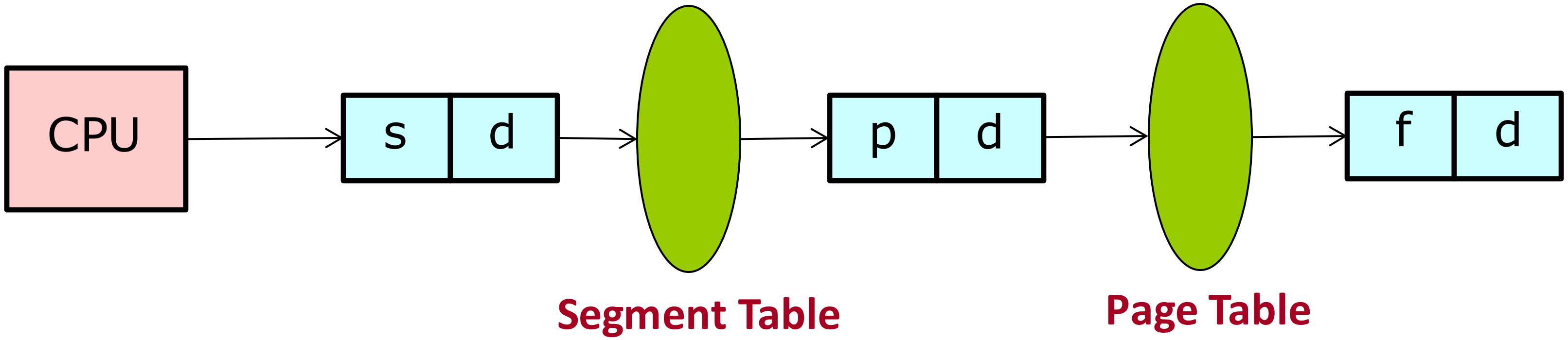
OS maintains info of free frames (frame table)

Before allocation

After allocation

Segmentation and Paging Combined

- A program is divided into segments; each segment divided into pages.



Advantages of Paging

- Clear separation between the programmer's view of memory and the actual physical memory

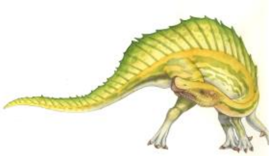
- Programmer's view of memory of his/her process
  - A single contiguous space, starting at (logical) address 0
  - Assume entire physical memory contains only this process

- Actual physical memory
  - Non-contiguous, scattered in pages across the physical memory
  - Occupies only a part of the physical memory

- Logical addresses mapped to physical addresses by OS, using specialized hardware support (to be discussed)

How to decide page / frame size?

- Calculating internal fragmentation
  - Page size = 2048 bytes
  - Process size = 72,766 bytes (35 * 2048 + 1086)
  - 35 pages + 1086 bytes
  - Internal fragmentation of 2048 - 1086 = 962 bytes
- How to decide frame or page size?
  - Worst case fragmentation = 1 frame – 1 byte
  - Average fragmentation = 1/2 of frame size
  - So small frame sizes desirable?
  - But each page table entry takes memory to track

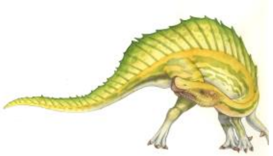


Optimum Page Size

- Assume that:
  - Average process segment size = S
  - Page table entry size = K bytes
  - Page size = P bytes
- Average internal fragmentation per segment = P / 2
- Average number of pages per segment = S / P (actually ceiling)
- Thus, total overhead $V = KS / P + P / 2$
- To find the value of P that minimizes overhead, set $dV/dP = 0$
  - $- K S / P^2 + 1/2 = 0$
  - Thus, $P = \sqrt{(2 S K)}$

Small P implies less internal fragmentation

Large P implies smaller page table (lower overhead)



Memory Management: Topics covered

- Background
- Swapping
- Contiguous Memory Allocation
- Segmentation
- Paging
- Structure of the Page Table (and variants of the page table) - to be covered next