Operating Systems Concepts

Main Memory: Address Translation

CS 4375, Fall 2025

Instructor: MD Armanuzzaman (***Arman***)

marmanuzzaman@utep.edu

October 6, 2025

Summary

- Address Space
- Virtual memory
- Linux Memory APIs
    - malloc(), free()
- Example Code

Agenda

- Main Memory:
  - Fixed and Dynamic Memory Allocation
  - External and Internal Fragmentation
  - Address Binding
  - Hardware Address Protection
  - Paging
  - Segmentation

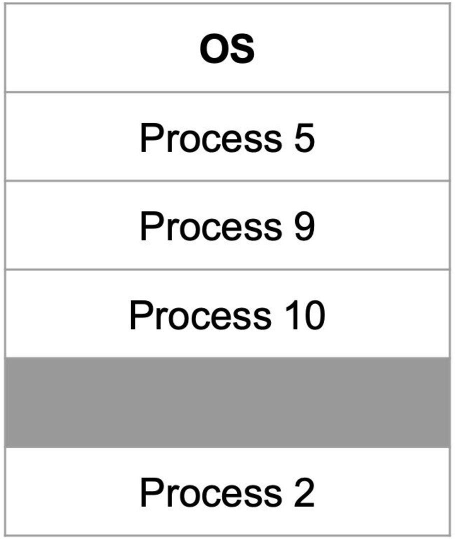
Memory Management

- Memory needs to be subdivided to accommodate multiple processes
- Memory management is an optimization task under constraints
- Movement between levels of storage hierarchy:

Level	1	2	3	4	5
Name	registers	cache	main memory	solid state disk	magnetic disk
Typical size	< 1 KB	< 16MB	< 64GB	< 1 TB	< 10 TB
Implementation technology	custom memory with multiple ports CMOS	on-chip or off-chip CMOS SRAM	CMOS SRAM	flash memory	magnetic disk
Access time (ns)	0.25 - 0.5	0.5 - 25	80 - 250	25,000 - 50,000	5,000,000
Bandwidth (MB/sec)	20,000 - 100,000	5,000 - 10,000	1,000 - 5,000	500	20 - 150
Managed by	compiler	hardware	operating system	operating system	operating system
Backed by	cache	main memory	disk	disk	disk or tape

Memory Allocation

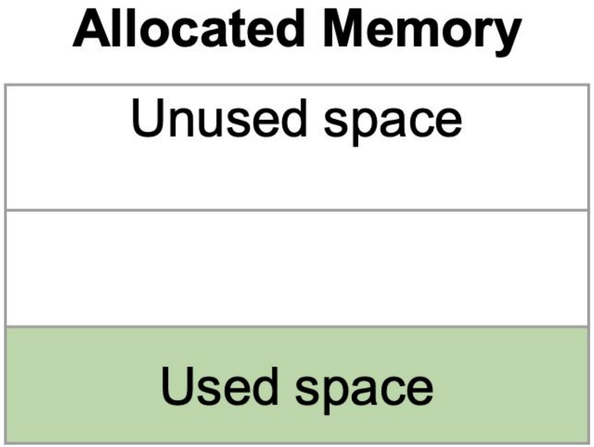
- Fixed-partition allocation
  - Divide memory into fixed-size partitions
  - Each partition contains exactly one process
  - The degree of multiprogramming is bound by the number of partitions
  - When a process terminates, the partition becomes available for other processes



- May lead to Internal Fragmentation– allocated memory may be slightly larger than **requested** memory; this size difference is memory **internal** to a partition, but not being **used**

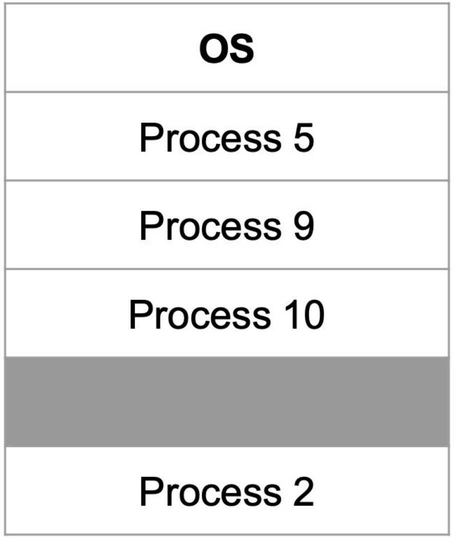
Memory Allocation

- May lead to Internal Fragmentation– allocated memory may be slightly larger than **requested** memory; this size difference is memory **internal** to a partition, but not being **used**



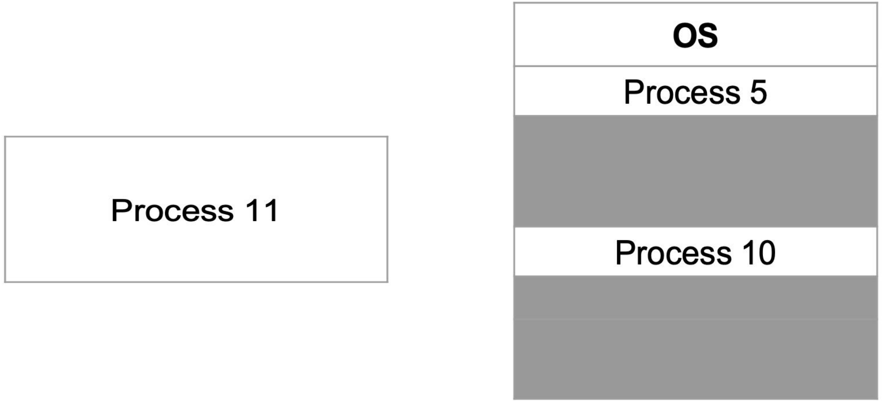
Memory Allocation

- Variable-partition Scheme (Dynamic)
  - When a process arrives, search for a hole large enough for this process
  - **Hole** – block of available memory; holes of various size are scattered throughout memory
  - Allocate only as much memory as needed
  - Operating system maintains information about:
    - allocated partitions
    - free partitions (hole)



Memory Allocation

- Can lead to External Fragmentation– total memory space exists to satisfy a request, but it is not contiguous (in average ~50% lost)



- Reduce external fragmentation by compaction
  - Shuffle memory contents to place all free memory together in one large block
  - Compaction is possible only if relocation is dynamic, and is done at execution time

Dynamic Storage-Allocation Problem

- How to satisfy a request of size n from a list of free holes?
    - **First-fit:** Allocate the first hole that is big enough
    - **Best-fit:** Allocate the smallest hole that is big enough; must search entire list, unless ordered by size. Produces the smallest leftover hole
    - **Worst-fit:** Allocate the largest hole; must also search entire list. Produces the largest leftover hole

- First-fit is faster
- Best-fit is better in terms of storage utilization
- Worst-fit may lead to less fragmentation

Dynamic Storage-Allocation Example

- Given five memory partitions of 100 KB, 500 KB, 200 KB, 300 KB, and 600 KB (in order), how would each of the first-fit, best-fit, and worst-fit algorithms place processes of 212 KB, 417 KB, 112 KB, and 426 KB (in order)?
- Which algorithm makes the most-efficient use of memory?

Dynamic Storage-Allocation Example

- Given five memory partitions of 100 KB, 500 KB, 200 KB, 300 KB, and 600 KB (in order), how would each of the first-fit, best-fit, and worst-fit algorithms place processes of 212 KB, 417 KB, 112 KB, and 426 KB (in order)?
- Which algorithm makes the most-efficient use of memory?

Available partitions after first fit: [100, 176, 200, 300, 183] KB [426 KB!!]

Available partitions after best fit: [100, 83, 88, 88, 174] KB

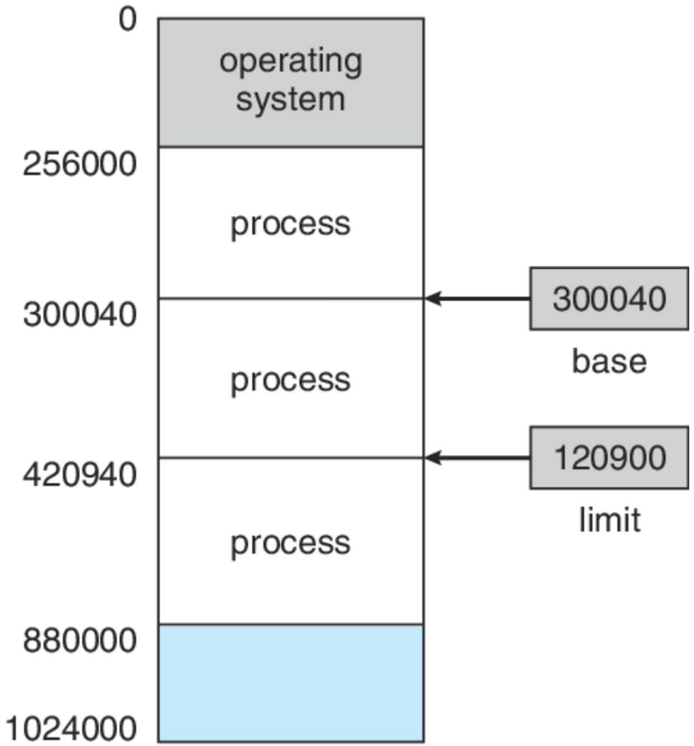
Available partitions after worst fit: [100, 83, 200, 300, 276] KB [426 KB!!]

Address Binding

- Addresses in a source program are generally symbolic
    - e.g. int count;
- A compiler binds these symbolic addresses to relocatable addresses
    - e.g. 100 bytes from the beginning of this module
- The linkage editor or loader will in turn bind the relocatable addresses to absolute addresses
    - e.g. 74014
- Each binding is mapping from one address space to another

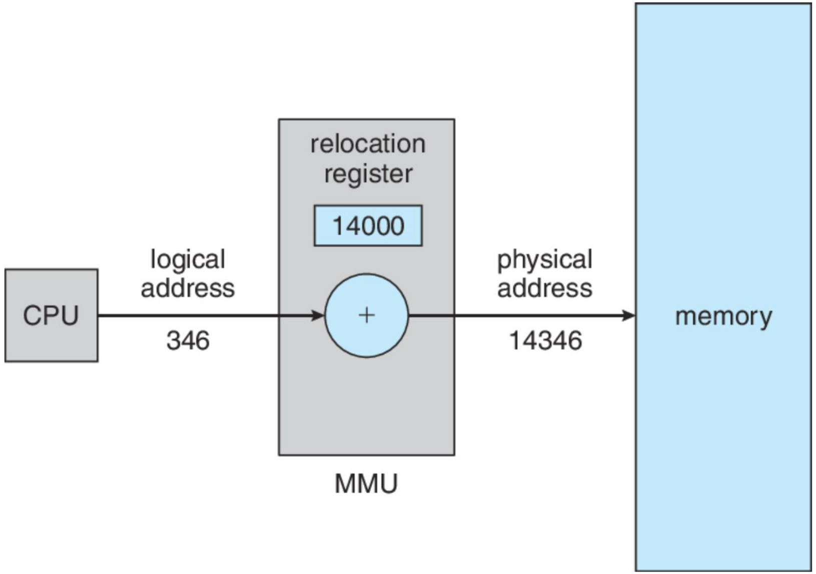
Logical Address Space

- Each process has a separate memory space
- Two registers provide address protection between processes:
  - Base register: smallest legal address space
  - Limit register: size of the legal range



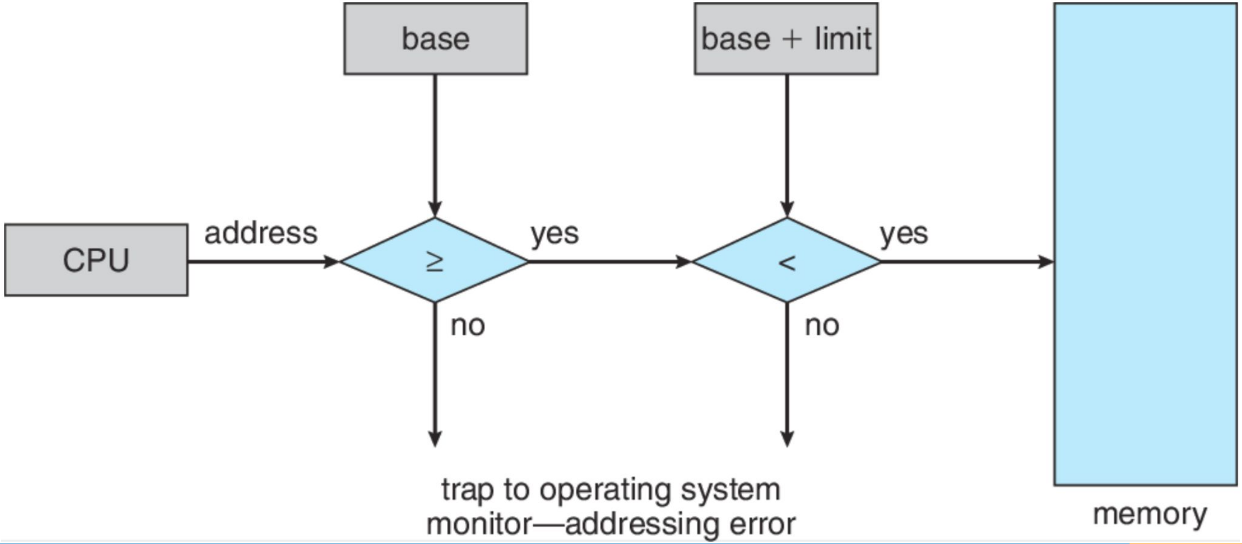
Memory-Management Unit (MMU)

- Hardware device that maps logical to physical address
- In MMU scheme, the value in the relocation register (base register) is added to every address generated by a user process at the time it is sent to memory
- The user program deals with logical addresses; it never sees the real physical addresses



Hardware Address Protection

- CPU hardware compares every address generated in user mode with the registers
- Any attempt to access other processes' memory will be trapped and cause a fatal error



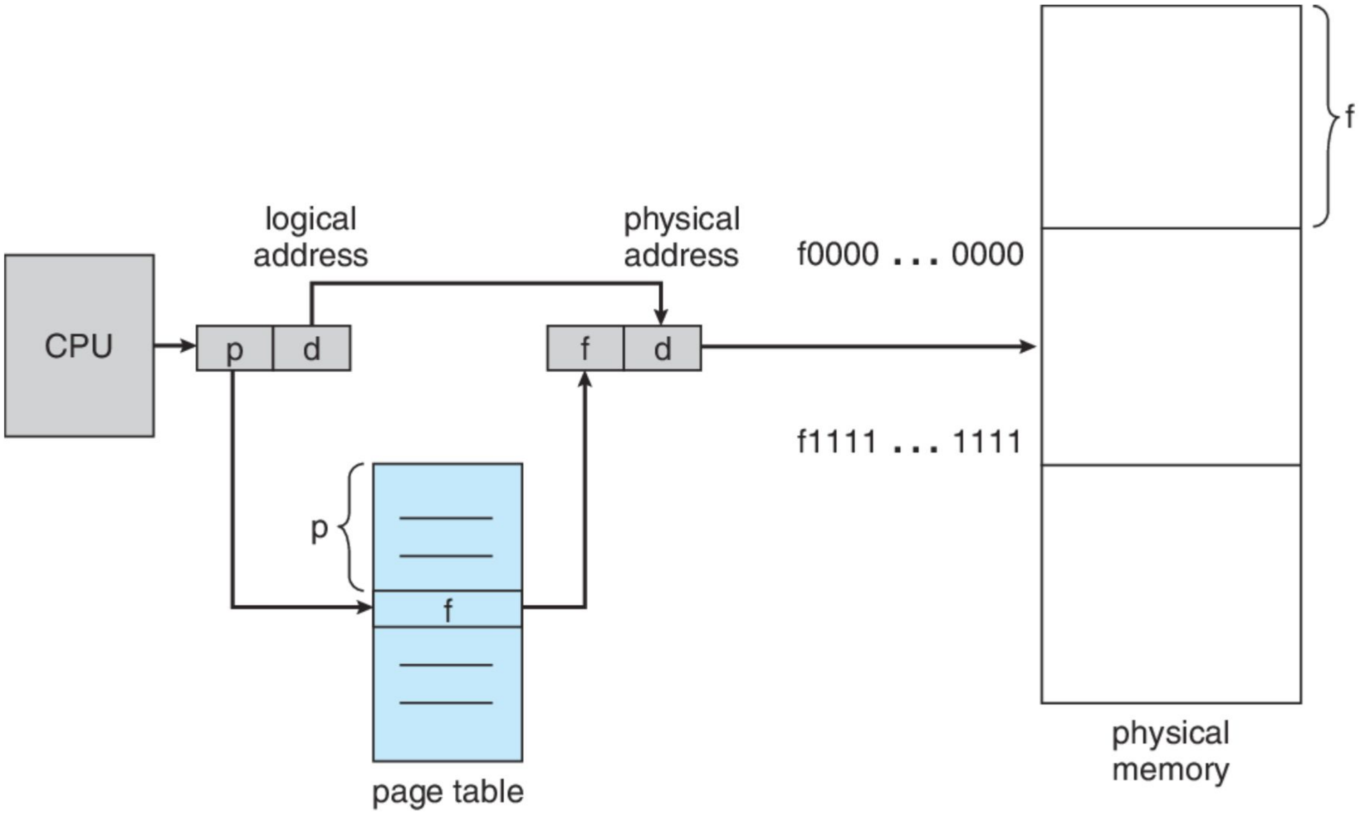
Paging - noncontiguous

- Physical address space of a process can be noncontiguous
- Divide physical memory into **fixed-sized blocks** called frames (size is power of 2, between 512 bytes and 16 megabytes)
- Divide logical memory into blocks of same size called pages
- 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

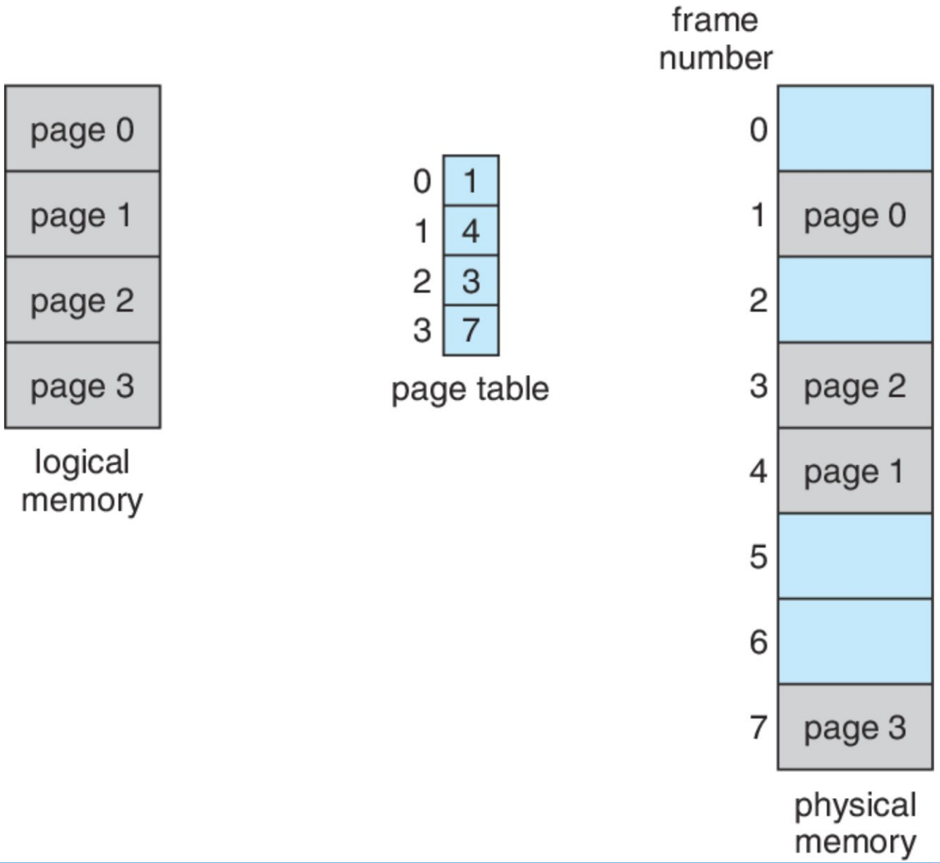
Paging Address Translation Scheme

- 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

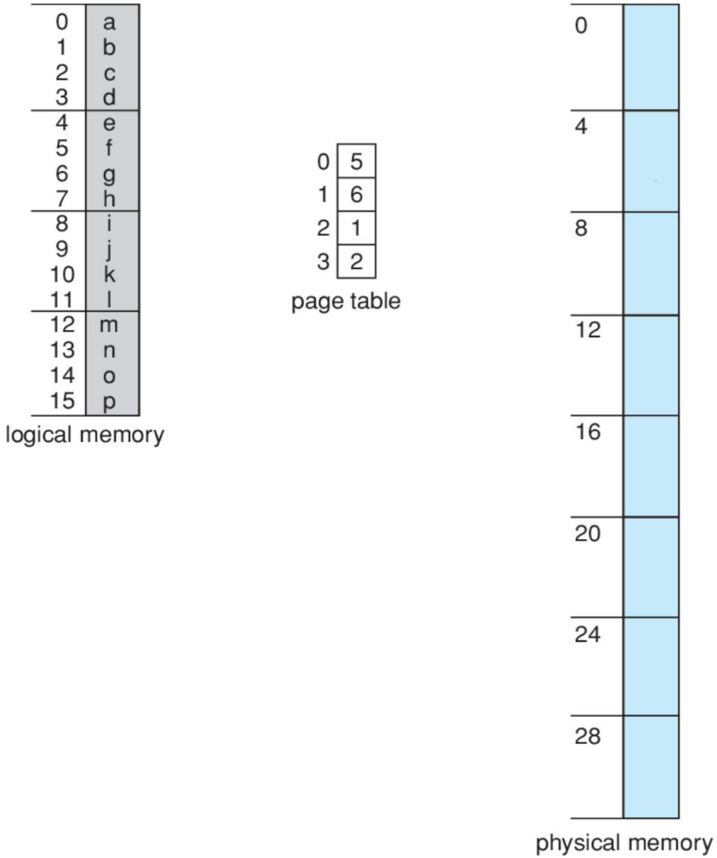
Address Translation Architecture



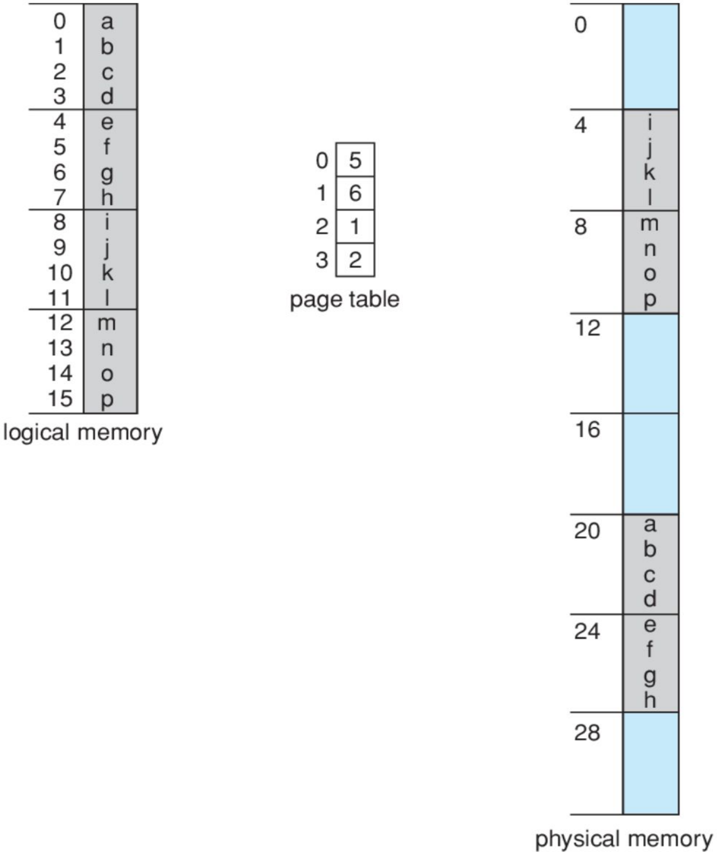
Paging Example



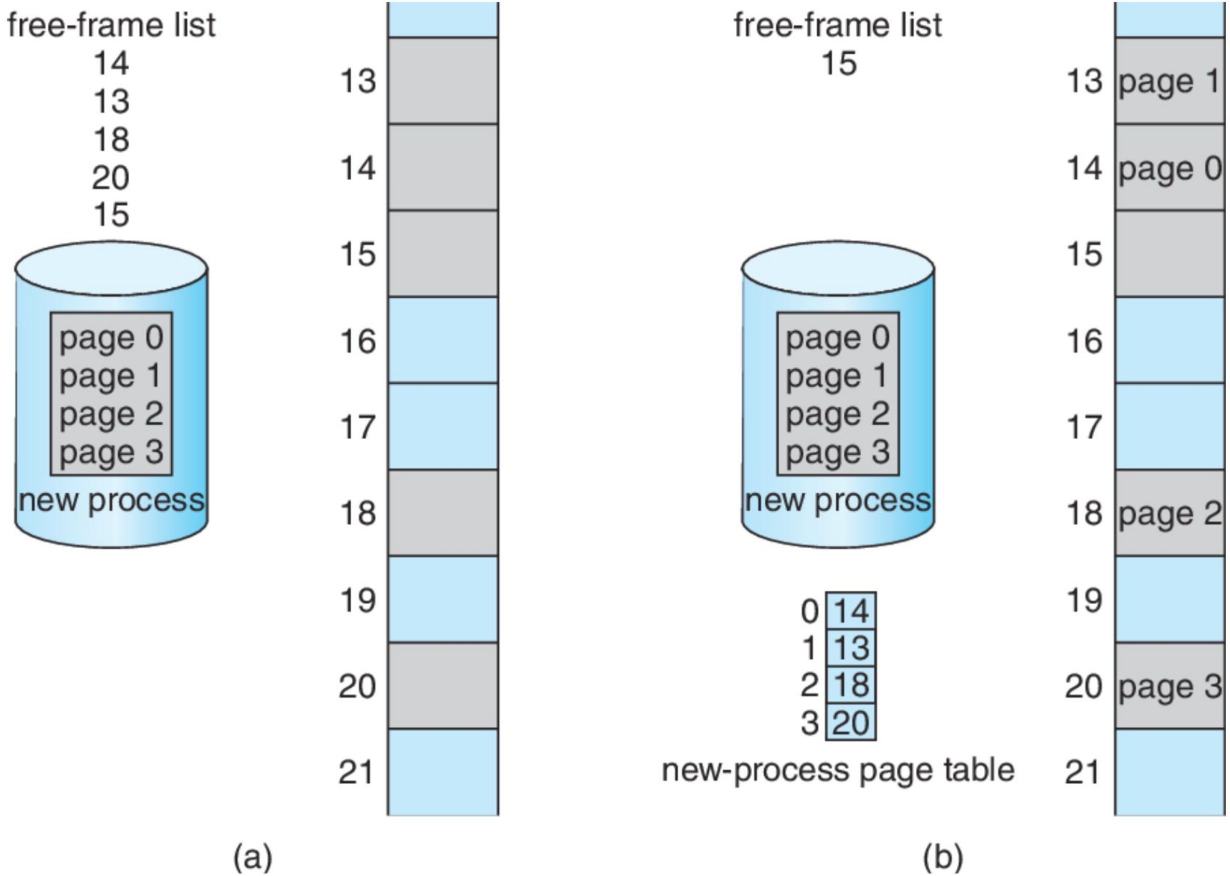
Paging Example



Paging Example



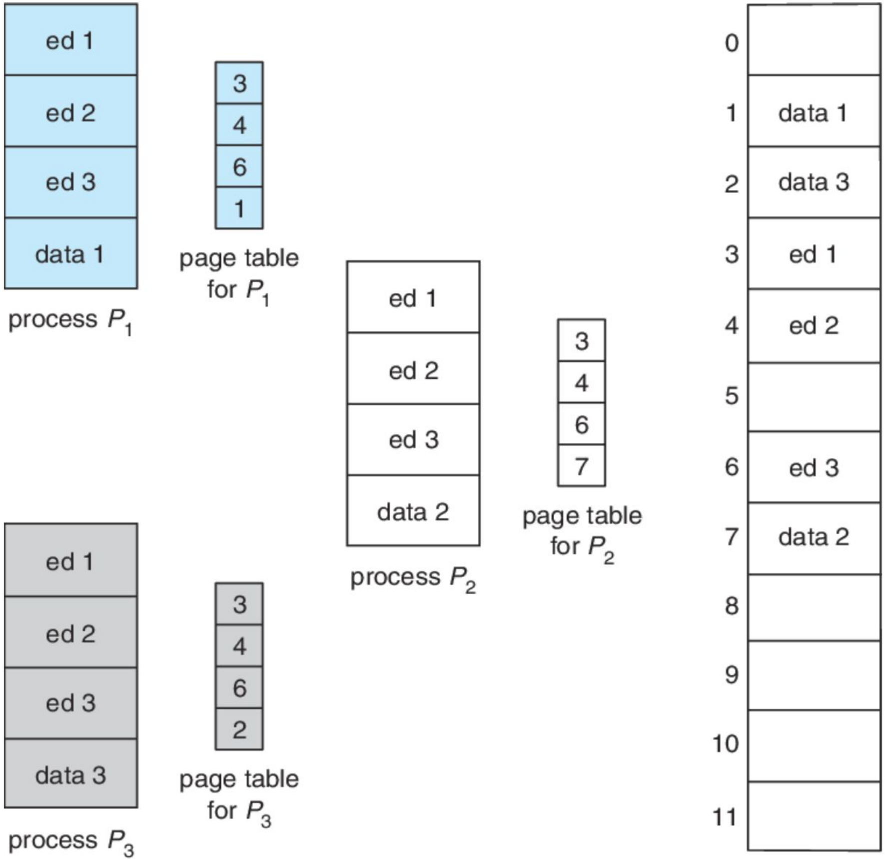
Free Frames


free-frame list
14
13
18
20
15
page 0
page 1
page 2
page 3
new process
13
14
15
16
17
18
19
20
21
(a)
free-frame list
15
page 0
page 1
page 2
page 3
new process
0 14
1 13
2 18
3 20
new-process page table
13 page 1
14 page 0
15
16
17
18 page 2
19
20 page 3
21
(b)

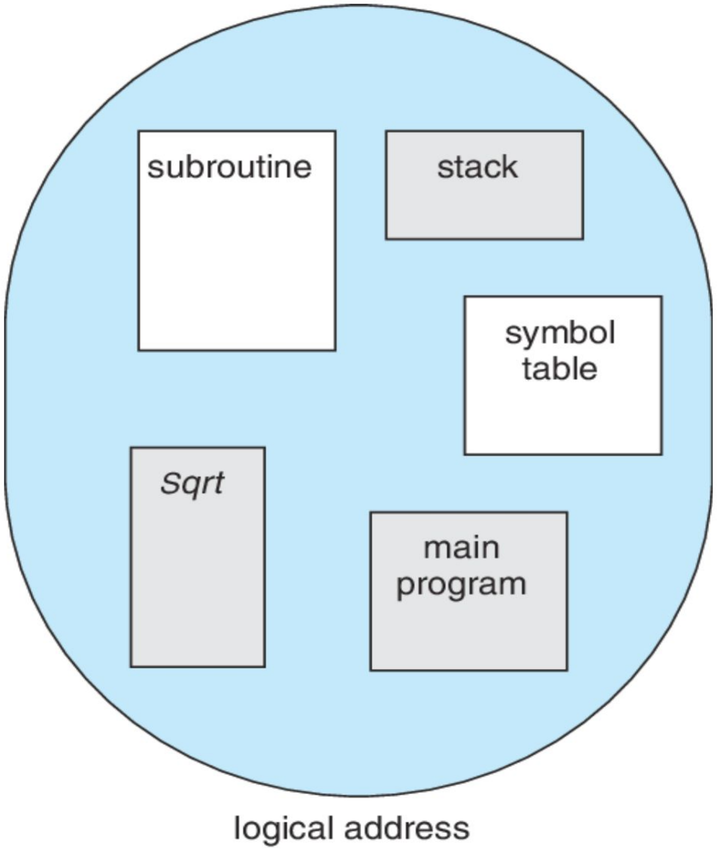
Shared Pages

- **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



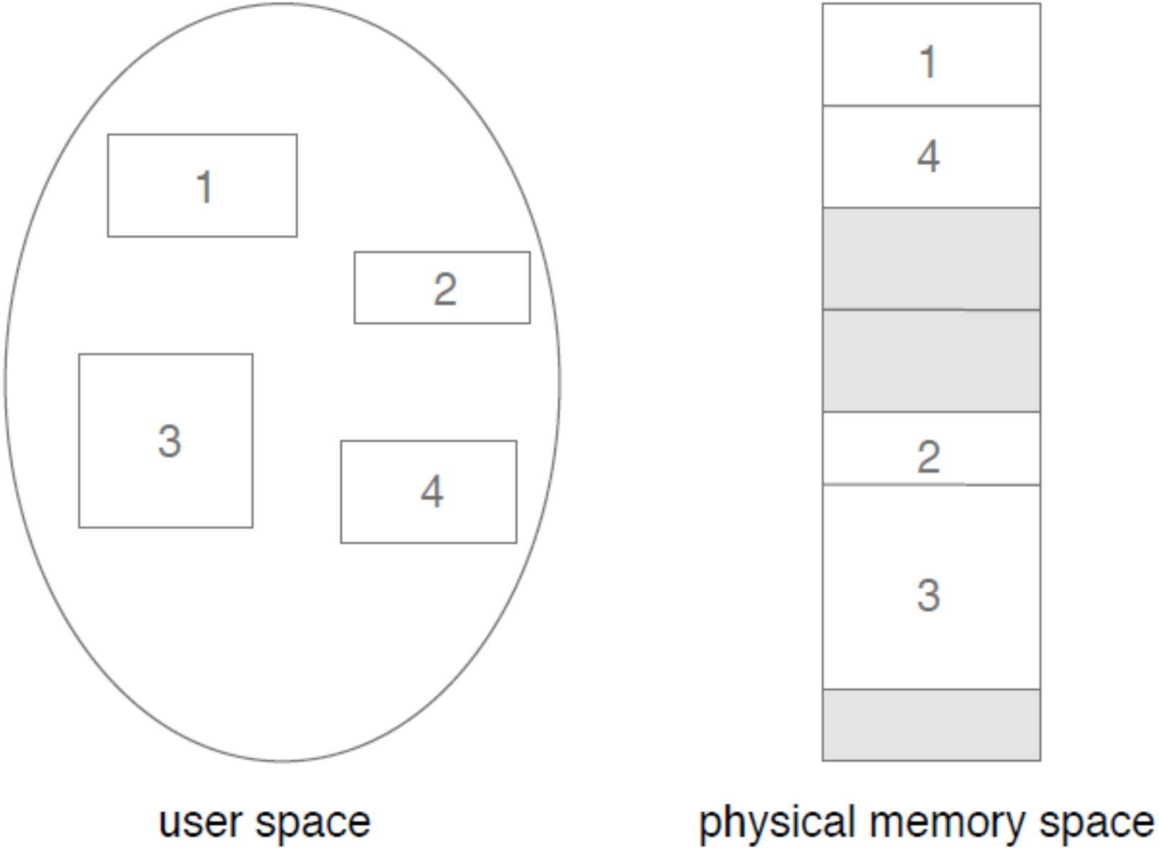
User's View of a Program



Segmentation

- Memory-management scheme that supports user view of memory
- A program is a collection of **variable sized** segments. A segment is a logical unit such as:
  - main program,
  - procedure,
  - function,
  - method,
  - object,
  - local variables, global variables,
  - common block,
  - symbol table, arrays

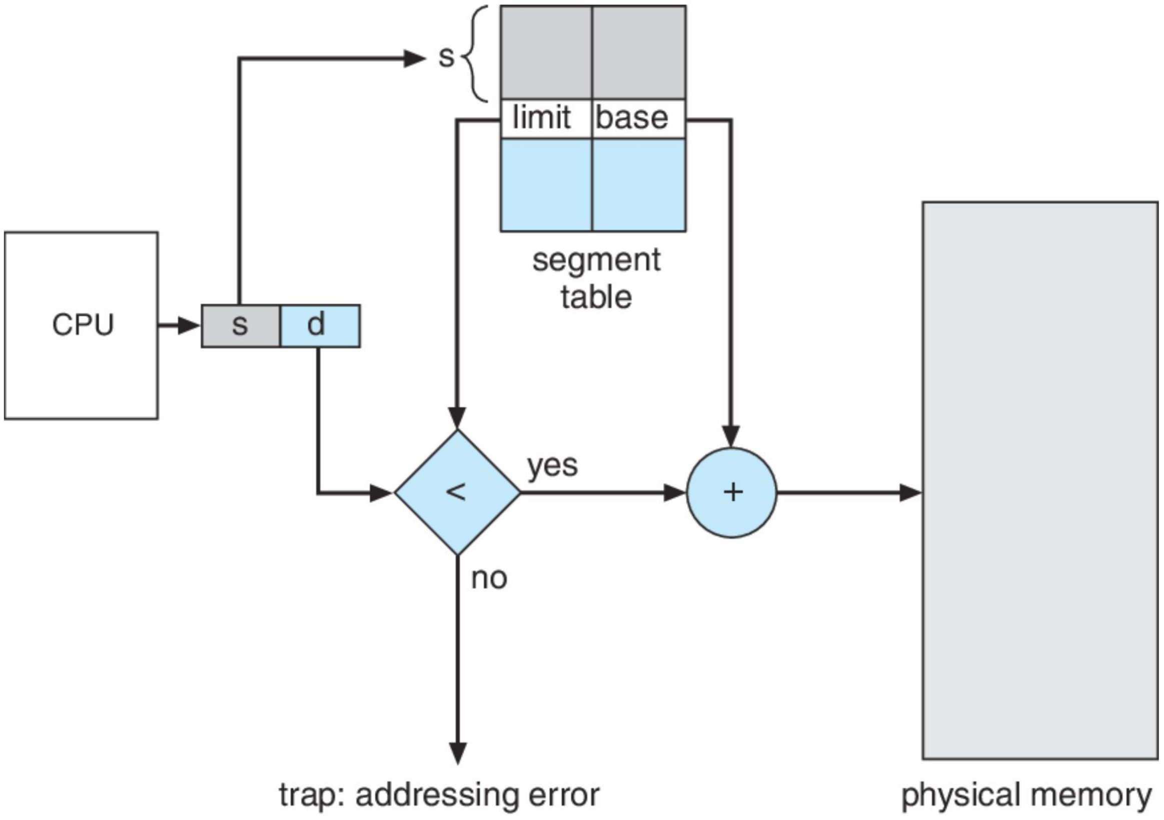
Segmentation: Logical View



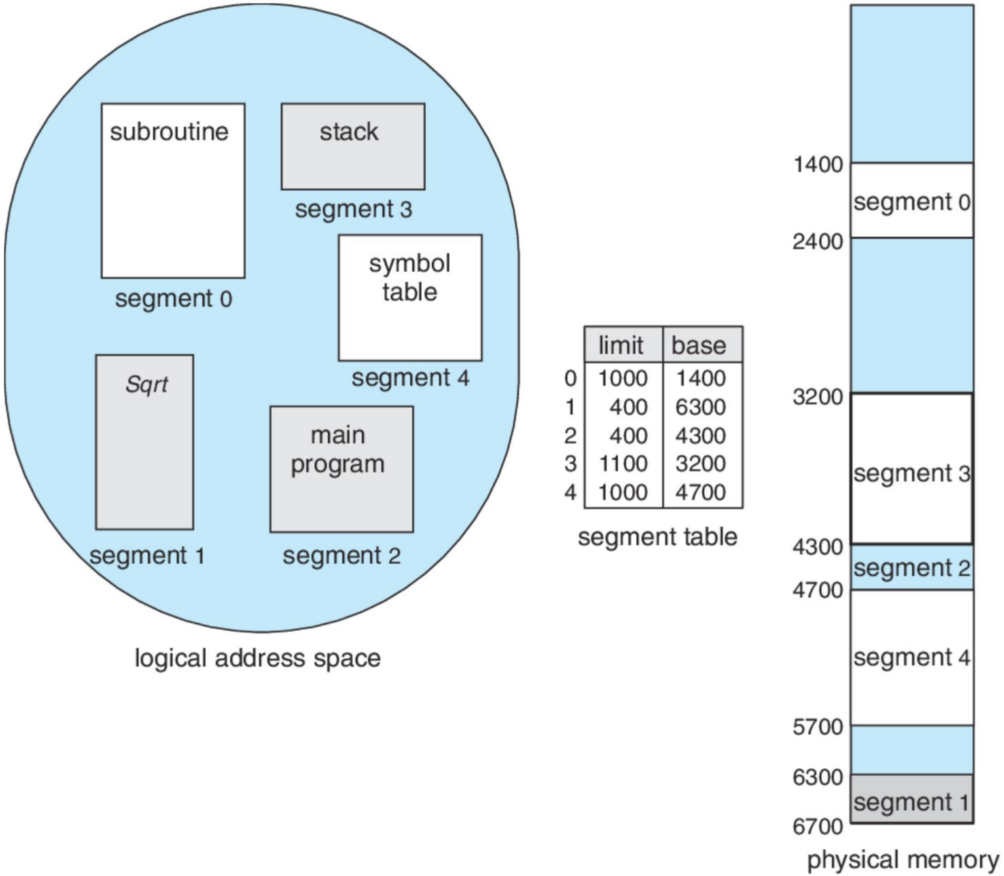
Segmentation: Architecture

- 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 the length (limit) of the segment
- Segment addressing is **d (offset) < STLR**

Segmentation: Address Translation Architecture



Segmentation: Example



Exercise

- Consider the following segment table:

Segment	Base	Length
0	219	600
1	2300	14
2	90	100
3	1327	580
4	1952	96

1) What are the physical addresses for the following logical addresses?

   a) 1, 100

   b) 2, 0

   c) 3, 580

Exercise

- Consider the following segment table:

Segment	Base	Length
0	219	600
1	2300	14
2	90	100
3	1327	580
4	1952	96

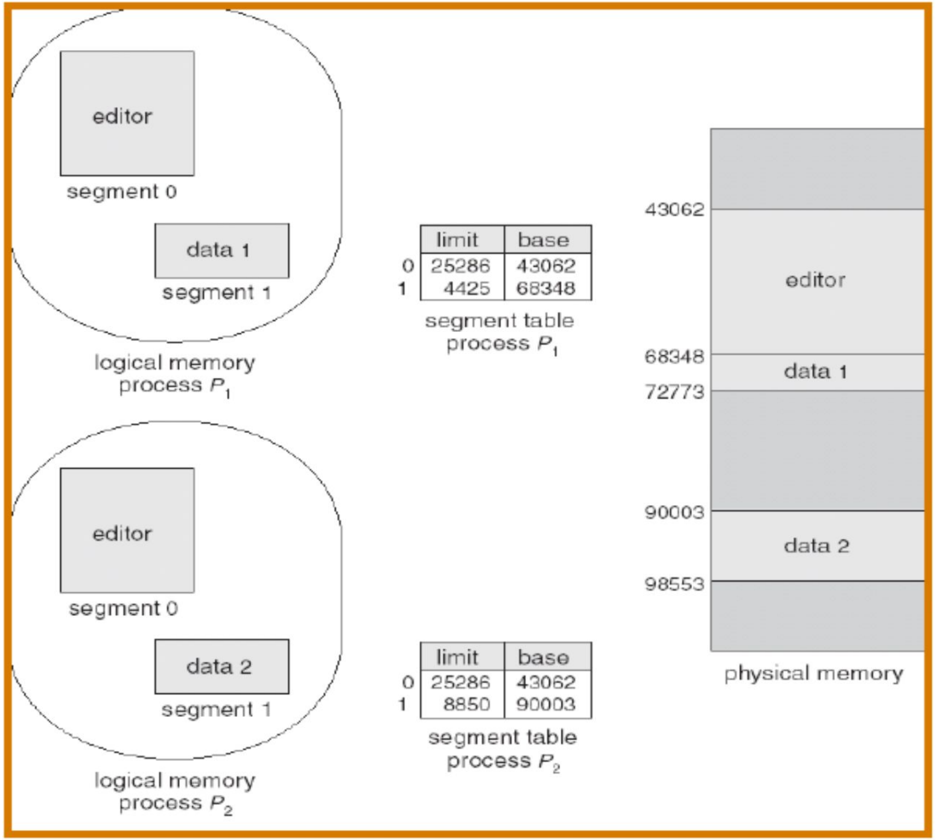
1) What are the physical addresses for the following logical addresses?

   a) 1, 100 — illegal reference, not within segment limits [0..13]

   b) 2, 0 — physical address = 90 + 0 = 90

   c) 3, 580 — illegal reference, not within segment limits [0..579]

Sharing of Segments



Paging vs Segmentation

Paging	Segmentation
Fixed-size division	Variable-size division
Managed by OS	Managed by compiler
Fragmentation: Internal	Fragmentation: External
Speed: Faster	Speed: Slower
Unit size by hardware	Unit size by user/programmer
Invisible to the user	Visible to the user

Modern OSes implement a hybrid approach called "**segmentation with paging**"

Acknowledgements

- “Operating Systems Concepts” book and supplementary material by A. Silberschatz, P. Galvin and G. Gagne
- “Operating Systems: Internals and Design Principles” book and supplementary material by W. Stallings
- “Modern Operating Systems” book and supplementary material by A. Tanenbaum
- R. Doursat and M. Yuksel from University of Nevada, Reno
- Farshad Ghanei from Illinois Tech
- T. Kosar and K. Dantu from University at Buffalo

Announcement

- Next class
  - Quiz 2: Could be in class or take home
  - xv6 scheduling: IMPORTANT FOR HOMEWORK 3
- Homework 2
  - Due on Today October 6th, 11.59 PM