CSE2008: Operating Systems

L23 L24 L25 L26 & L27 : Memory Management

Dr. Subrata Tikadar
SCOPE, VIT-AP University

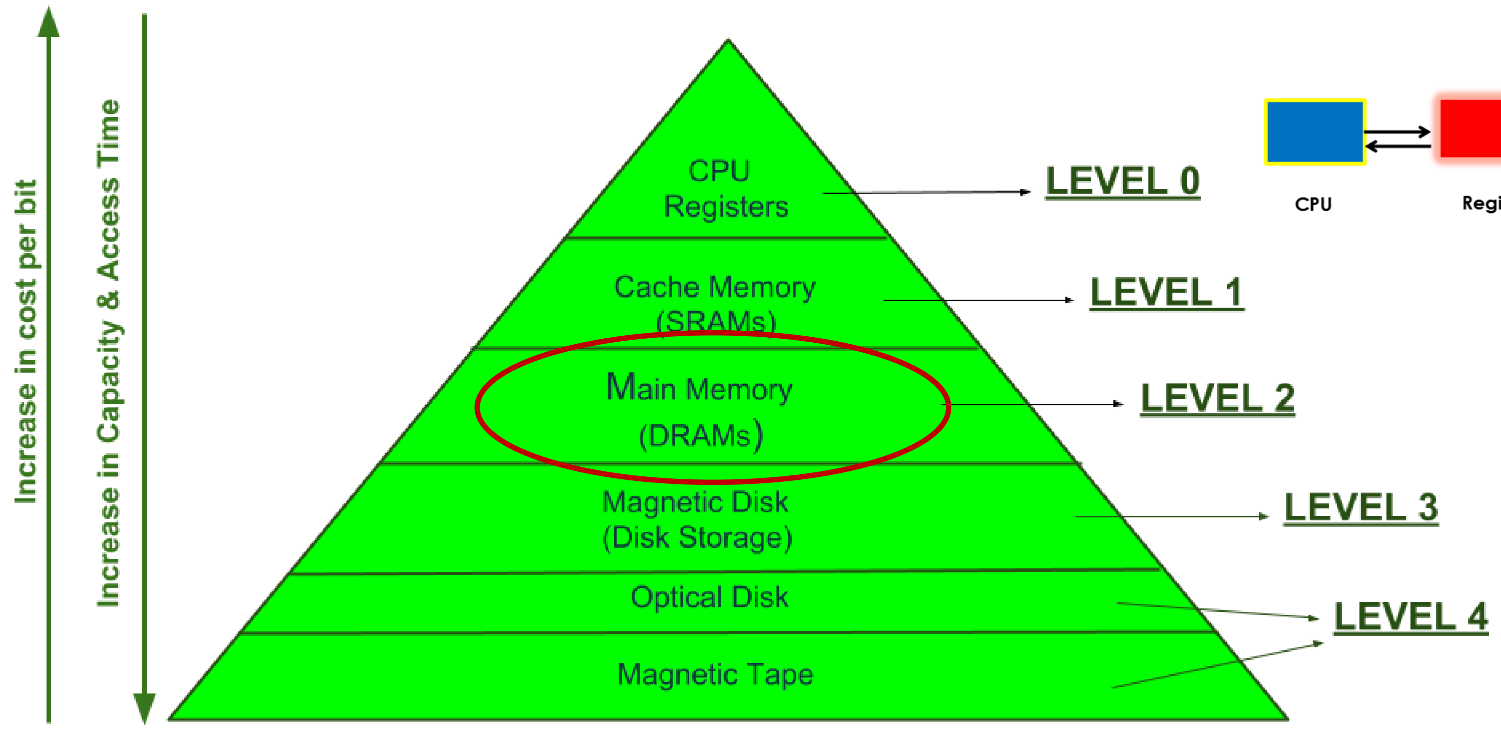
Recap

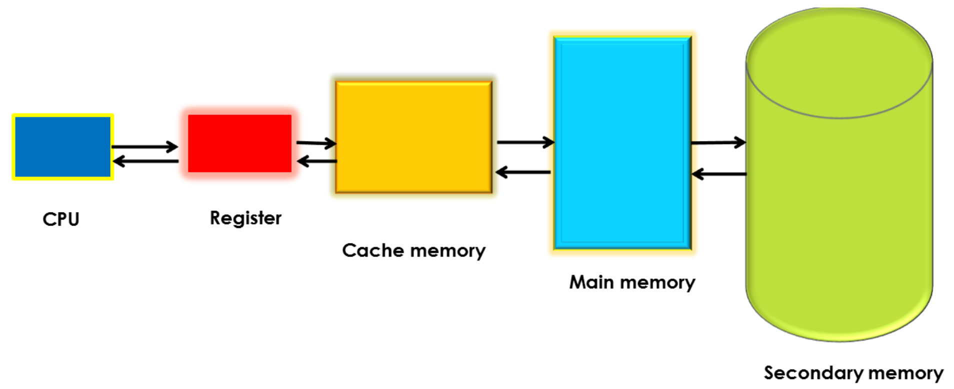
- Introductory Concepts
- Process Fundamentals
- IPC
- CPU Scheduling Algorithms
- Multithreading Concepts
- Synchronization
- Deadlock

Outline

- Memory Hierarchy
- Address Space
- Contiguous Memory Allocation
- Paging
- Swapping

Memory Hierarchy





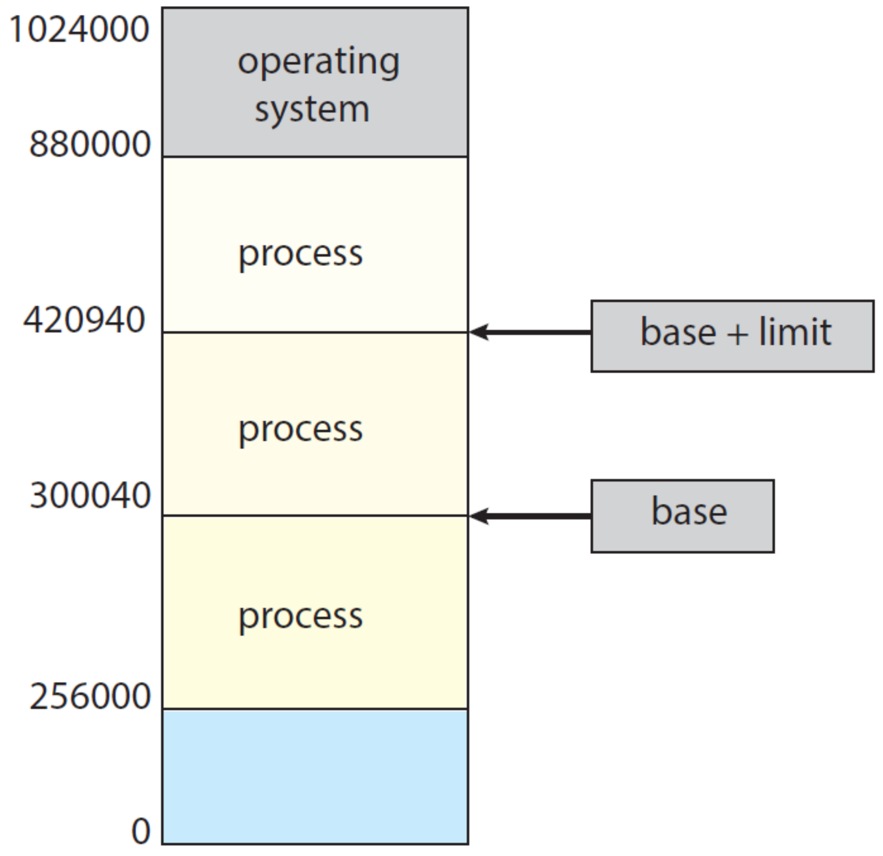
MEMORY HIERARCHY DESIGN

Address Space

- Basic Hardware
- Address Binding
- Logical Versus Physical Address Space
- Dynamic Loading
- Dynamic Linking and Shared Libraries

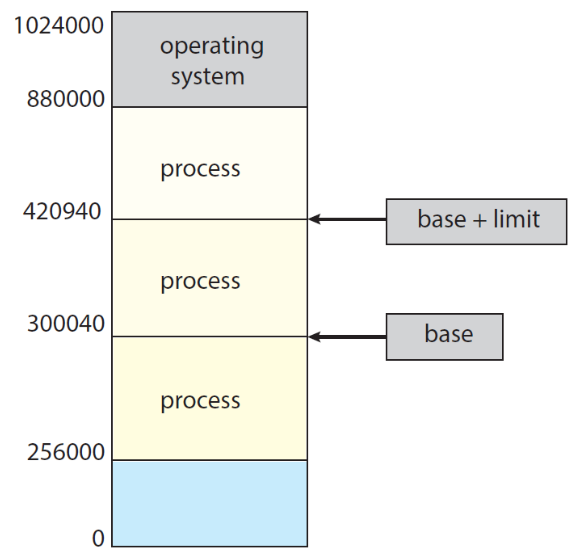
Address Space

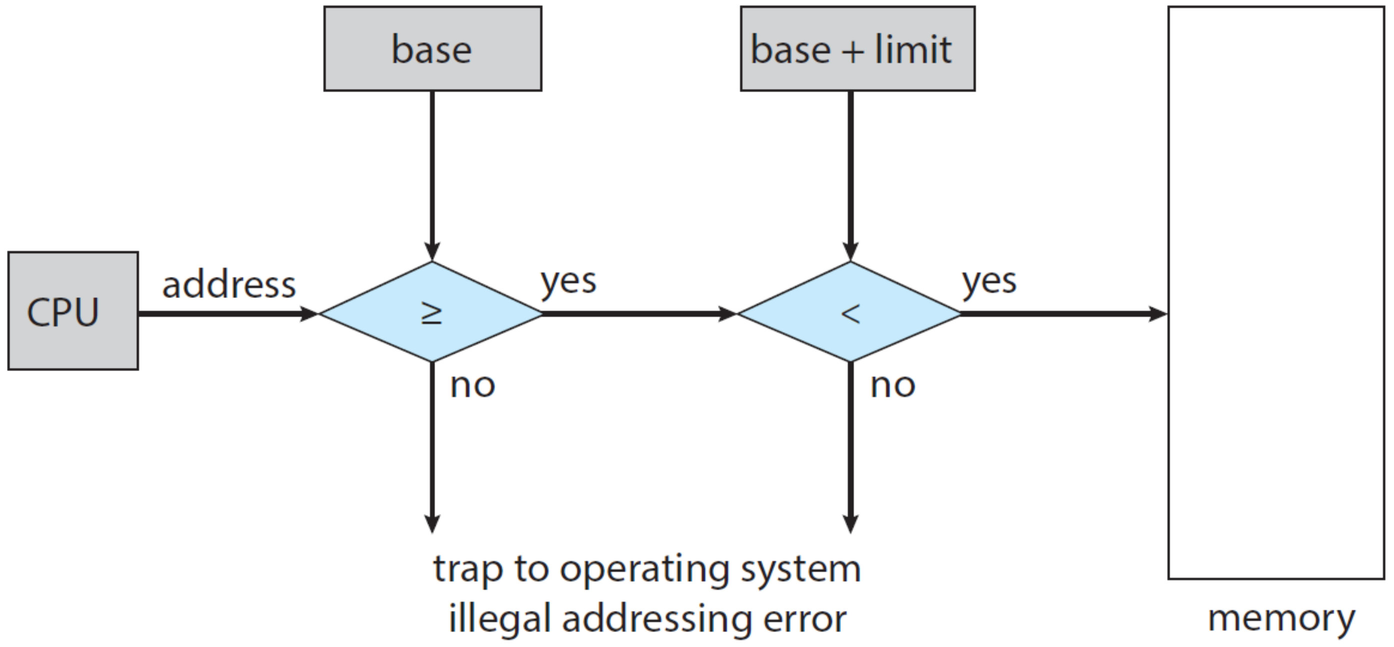
- Basic Hardware



Address Space

- Basic Hardware





Address Space

- Address Binding
  - Binding Steps
    - Compile time

      If you know at compile time where the process will reside in memory, then absolute code can be generated.

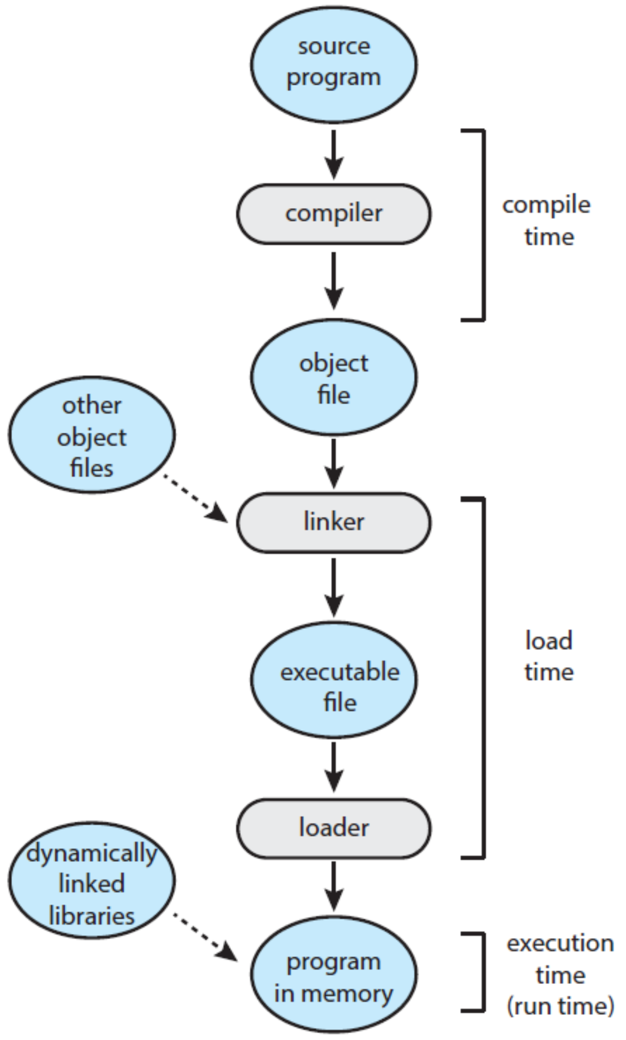
        Example: If you know that a user process will reside starting at location R, then the generated compiler code will start at that location and extend up from there. If, at some later time, the starting location changes, then it will be necessary to recompile this code.
    - Load time

      If it is not known at compile time where the process will reside in memory, then the compiler must generate relocatable code.

        In this case, final binding is delayed until load time. If the starting address changes, we need only reload the user code to incorporate this changed value.
    - Execution time

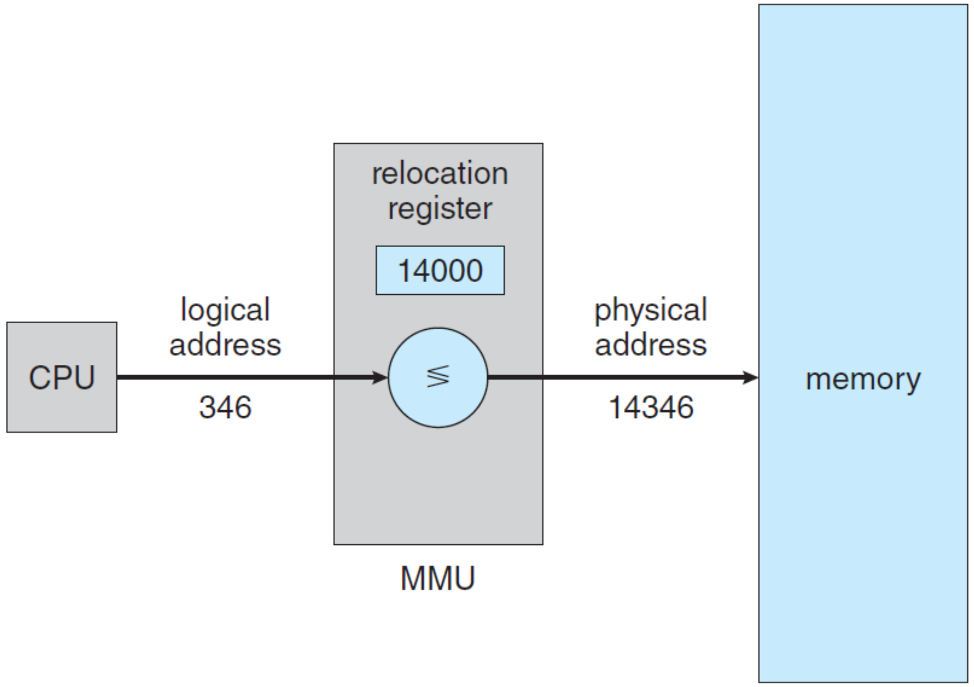
      If the process can be moved during its execution from one memory segment to another, then binding must be delayed until run time.

        Special hardware must be available for this scheme to work. Most operating systems use this method.



Address Space

- Logical Versus Physical Address Space
  - An address generated by the CPU is commonly referred to as **a logical address**, whereas an address seen by the memory unit—that is, the one loaded into the memory-address register of the memory—is commonly referred to as a **physical address**.
  - The set of all logical addresses generated by a program is a **logical address space**. The set of all physical addresses corresponding to these logical addresses is **a physical address space** (in the execution-time address-binding scheme, the logical and physical address spaces differ).



Address Space

- Dynamic Loading
  - A routine is not loaded until it is called
    - All routines are kept on disk in a relocatable load format. The main program is loaded into memory and is executed. When a routine needs to call another routine, the calling routine first checks to see whether the other routine has been loaded. If it has not, the relocatable linking loader is called to load the desired routine into memory and to update the program's address tables to reflect this change. Then control is passed to the newly loaded routine.

Address Space

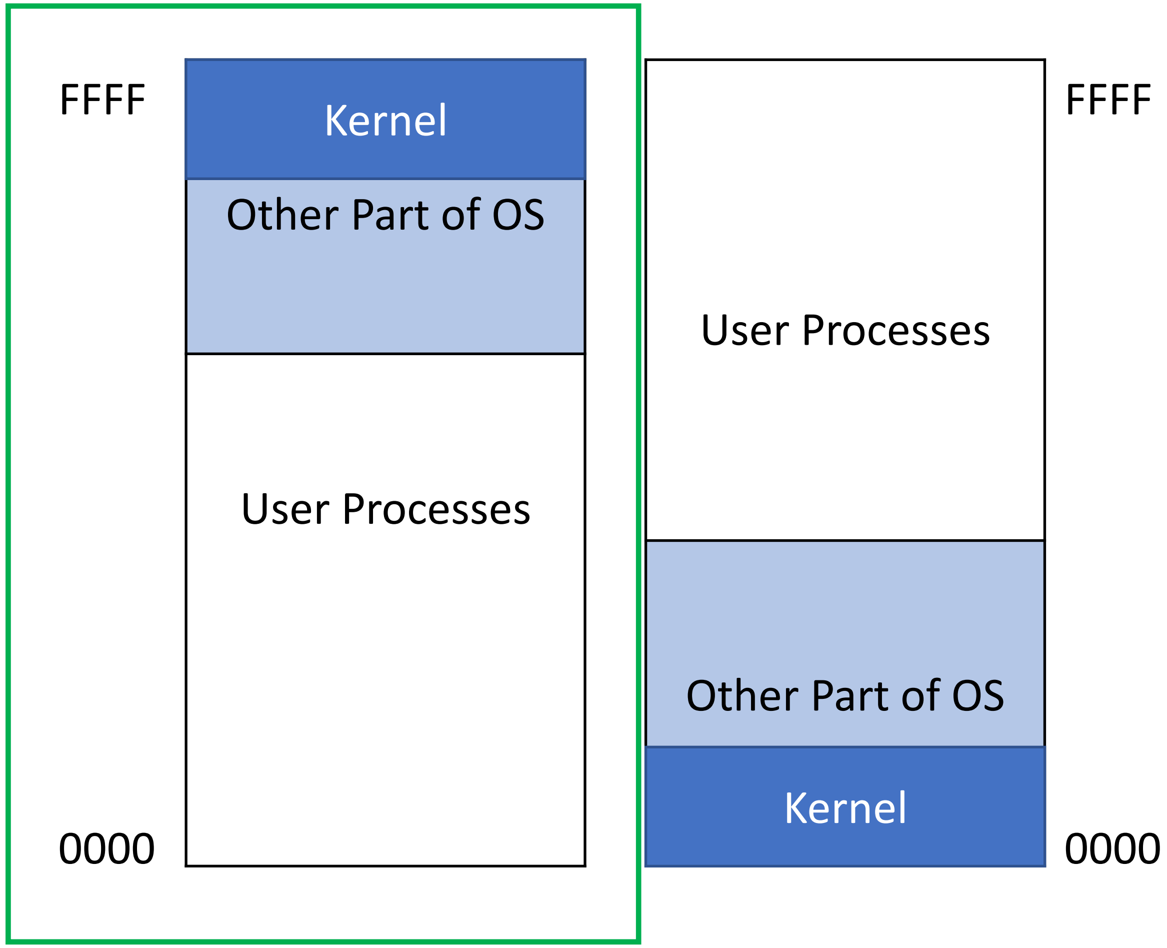
- Dynamic Linking and Shared Libraries
  - Dynamically linked libraries (**DLLs**) are system libraries that are linked to user programs when the programs are run.
    - Advantages:
      1. Helps in avoiding increasing size of the process and waste of memory.
      2. These libraries can be shared among multiple processes, so that only one instance of the DLL in main memory

Contiguous Memory Allocation

- Two contiguous partitions
  - One for OS
  - Another for user processes

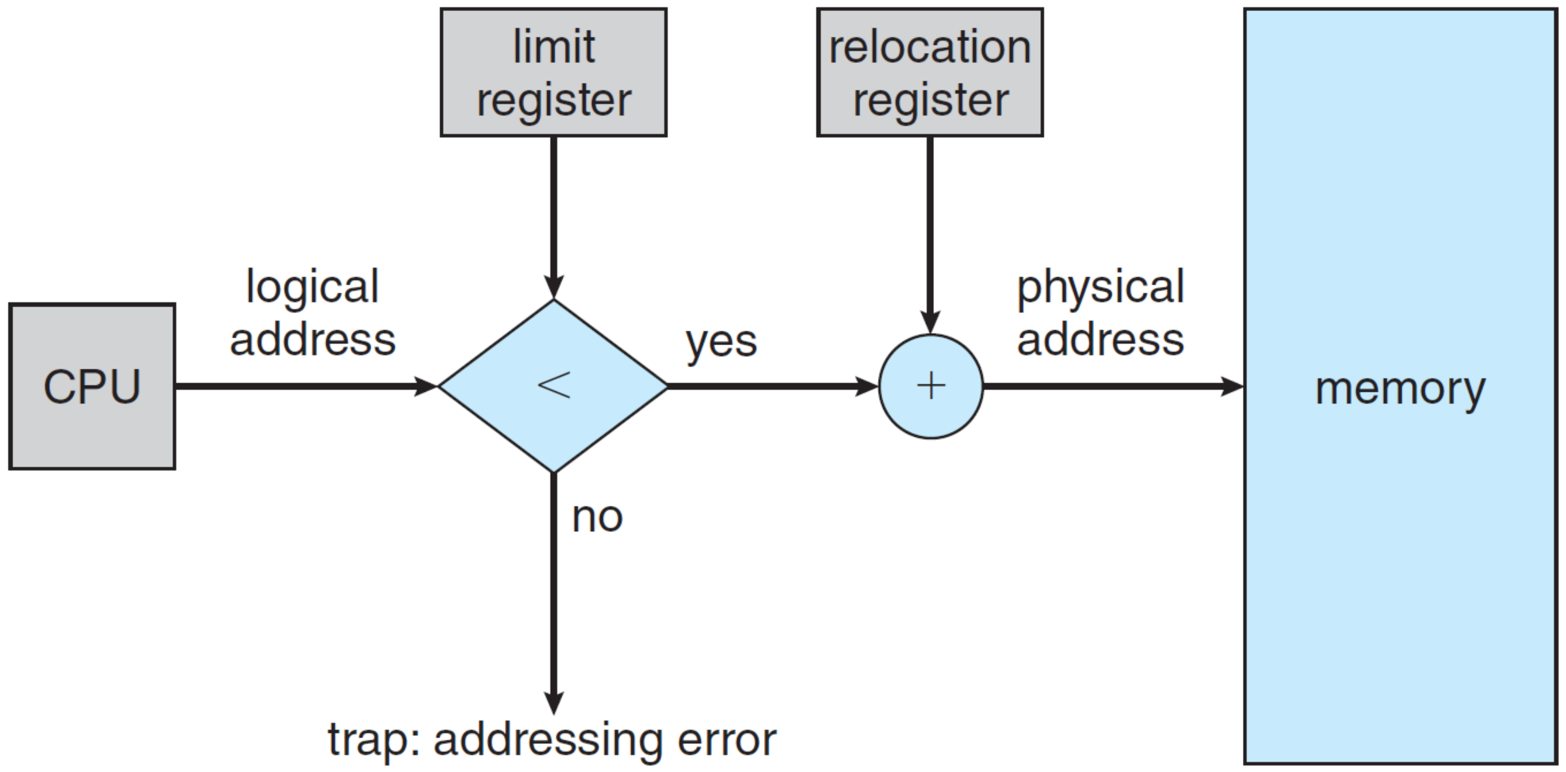
Main Concerns:

- Memory Protection
- Memory Allocation



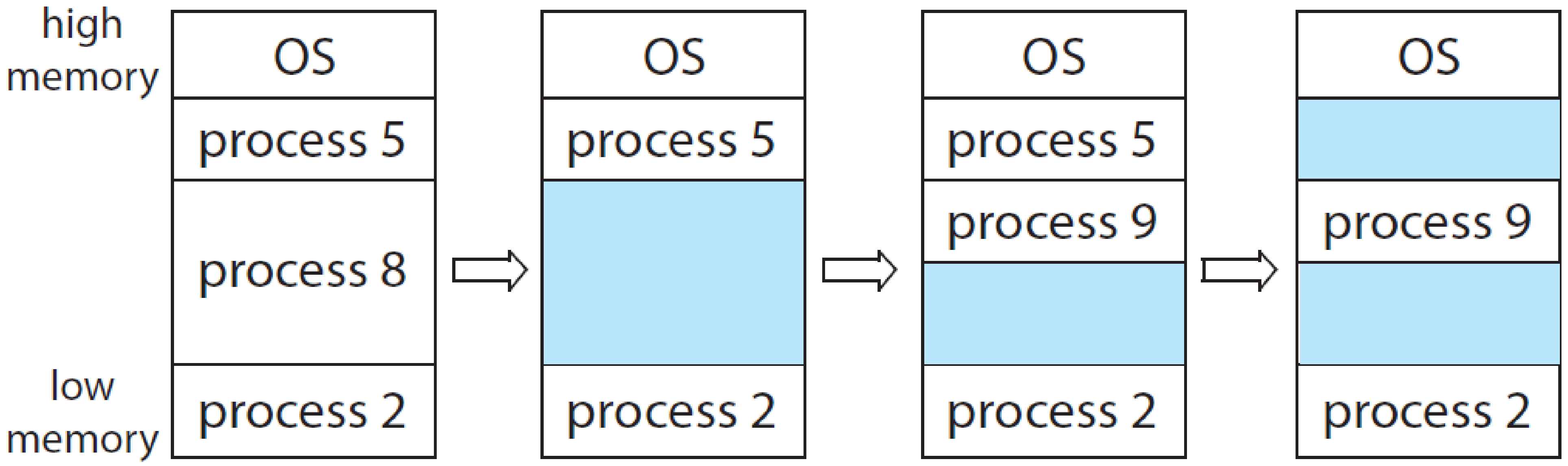
Contiguous Memory Allocation

- Memory Protection



Contiguous Memory Allocation

- Memory Allocation



Contiguous Memory Allocation

- Solution of Dynamic Storage Allocation Problem
    - **First fit.** Allocate the first hole that is big enough. Searching can start either at the beginning of the set of holes or at the location where the previous first-fit search ended. We can stop searching as soon as we find a free hole that is large enough.
    - **Best fit.** Allocate the smallest hole that is big enough. We must search the entire list, unless the list is ordered by size. This strategy produces the smallest leftover hole.
    - **Worst fit.** Allocate the largest hole. Again, we must search the entire list, unless it is sorted by size. This strategy produces the largest leftover hole, which may be more useful than the smaller leftover hole from a best-fit approach.

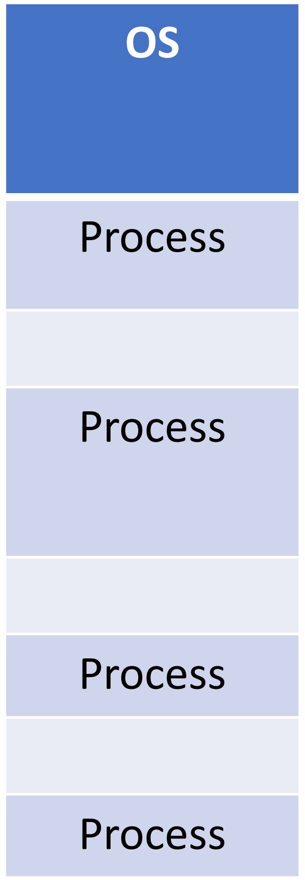
Contiguous Memory Allocation

- Solution of Dynamic Storage Allocation Problem
  - Exercise:

    Given six memory partitions of 100 MB, 170 MB, 40 MB, 205 MB, 300 MB, and 185 MB (in order), how would the first-fit, best-fit, and worst-fit algorithms place processes of size 200 MB, 15 MB, 185 MB, 75 MB, 175 MB, and 80 MB (in order)? Indicate which—if any—requests cannot be satisfied. Comment on how efficiently each of the algorithms manages memory.

Contiguous Memory Allocation

- Fragmentation – Problem in these Solutions (First-Fit/Best-Fit)



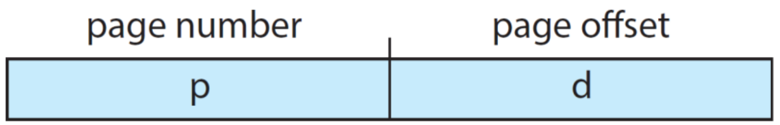
  - Solution
    - **Compaction**
    - **Paging**

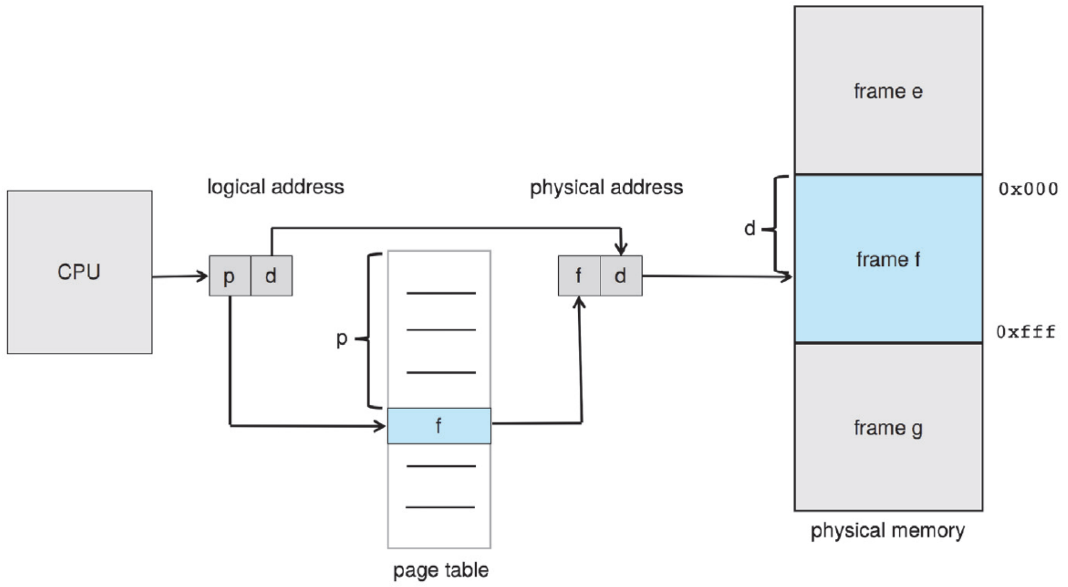
Paging

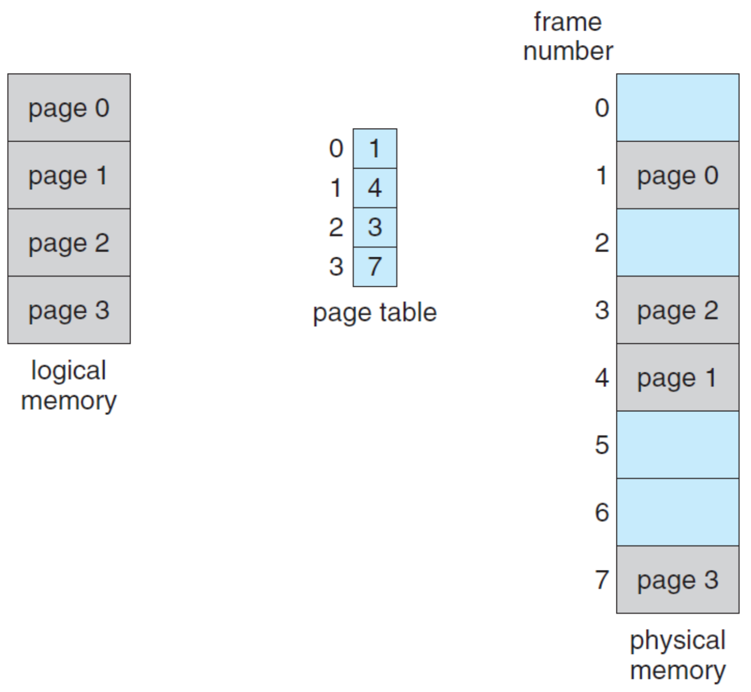
- Basic Method
  - Breaks
    - physical memory into fixed-sized blocks called **frames**
    - logical memory into blocks of the same size called **pages**

Paging

- Basic Method (cont…)
  - Every address generated by the CPU is divided into two parts
    - A page number **(p)**
    - A page offset **(d)**

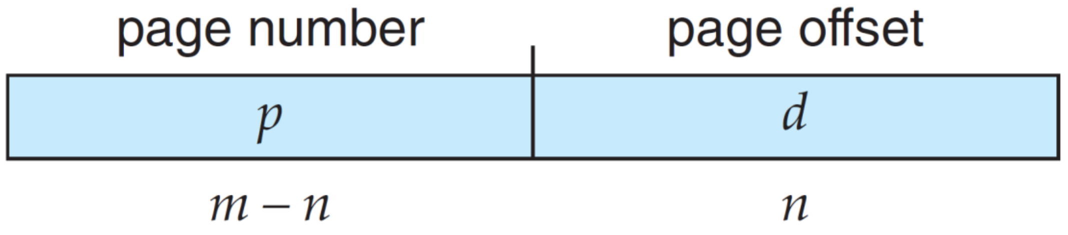






Paging

- Basic Method (cont…)
  - Steps taken by the MMU to translate a logical address generated by the CPU to a physical address:
    1. Extract the page number p and use it as an index into the page table.
    2. Extract the corresponding frame number f from the page table.
    3. Replace the page number p in the logical address with the frame number f .



Paging

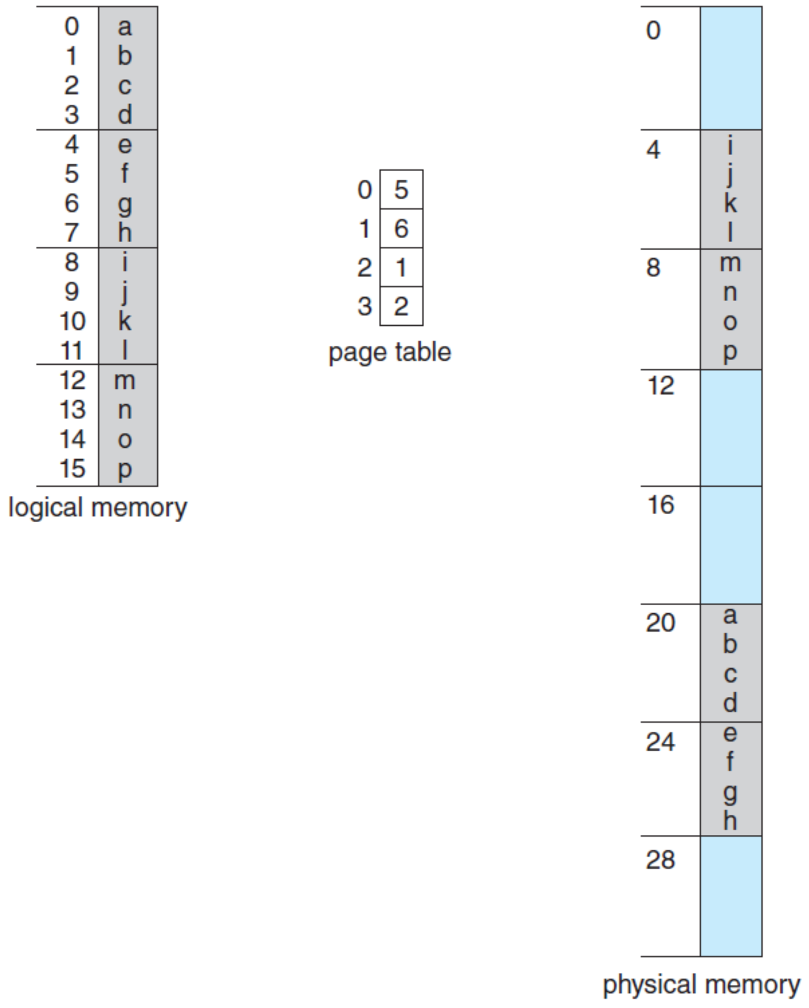
- Basic Method (cont…)
  - Example:
    - Consider the memory in this figure.

      Here, in the logical address, n = 2 and m = 4. Using a page size of 4 bytes and a physical memory of 32 bytes (8 pages). Therefore, the programmer's view of memory can be mapped into physical memory:

      ➢ Logical address 0 is page 0, offset 0. Indexing into the page table, we find that page 0 is in frame 5.

        Logical address 0 maps to physical address 20 [= (5 × 4) + 0]. Logical address 3 (page 0, offset 3) maps to physical address 23 [= (5 × 4) + 3].

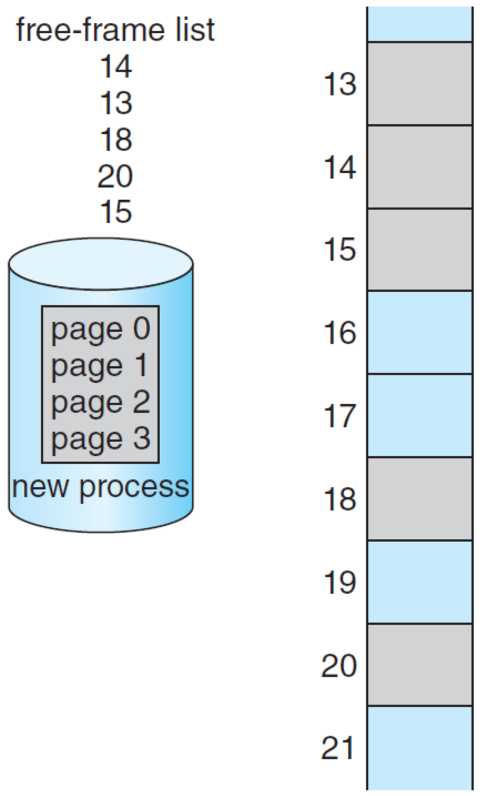
      ➢ Logical address 4 is page 1, offset 0; according to the page table, page 1 is mapped to frame 6. Thus, logical address 4 maps to physical address 24 [= (6× 4) + 0].

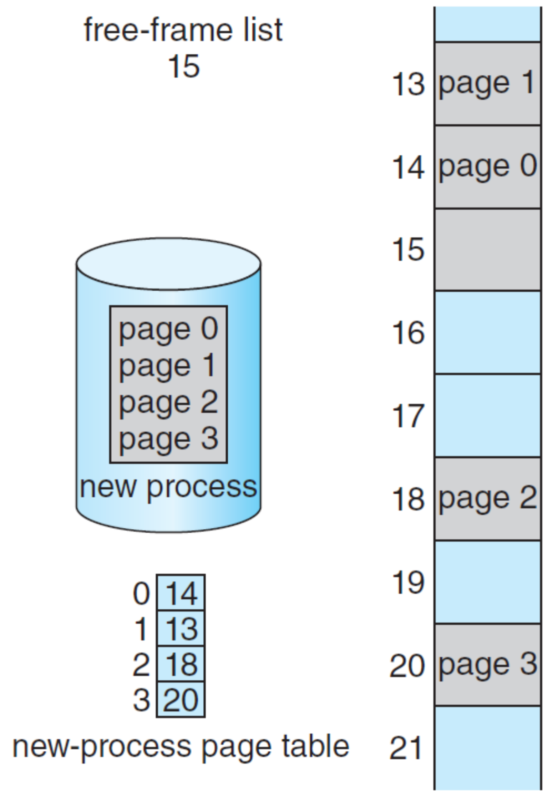


Paging

- Basic Method (cont…)

Free frames before allocation





Free frames after allocation

Paging

- Hardware Support
  - Use of hardware registers
  - Use of main memory (PTBR)

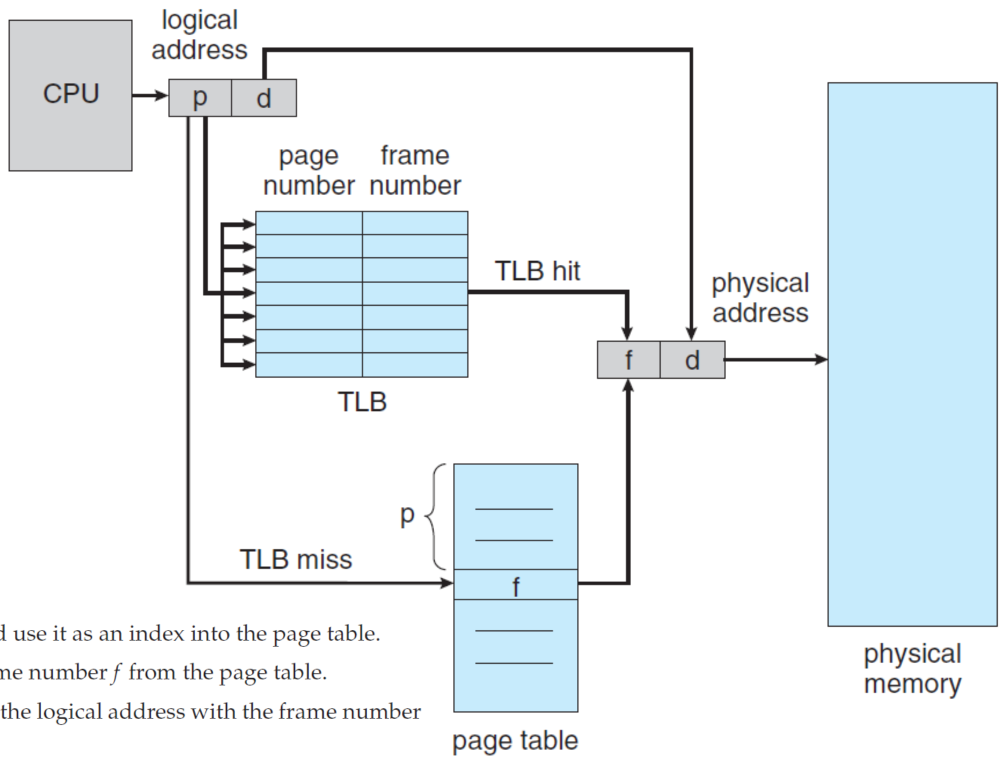
Slower Memory Access Times

Solution:

Translation Look-Aside Buffer

Paging

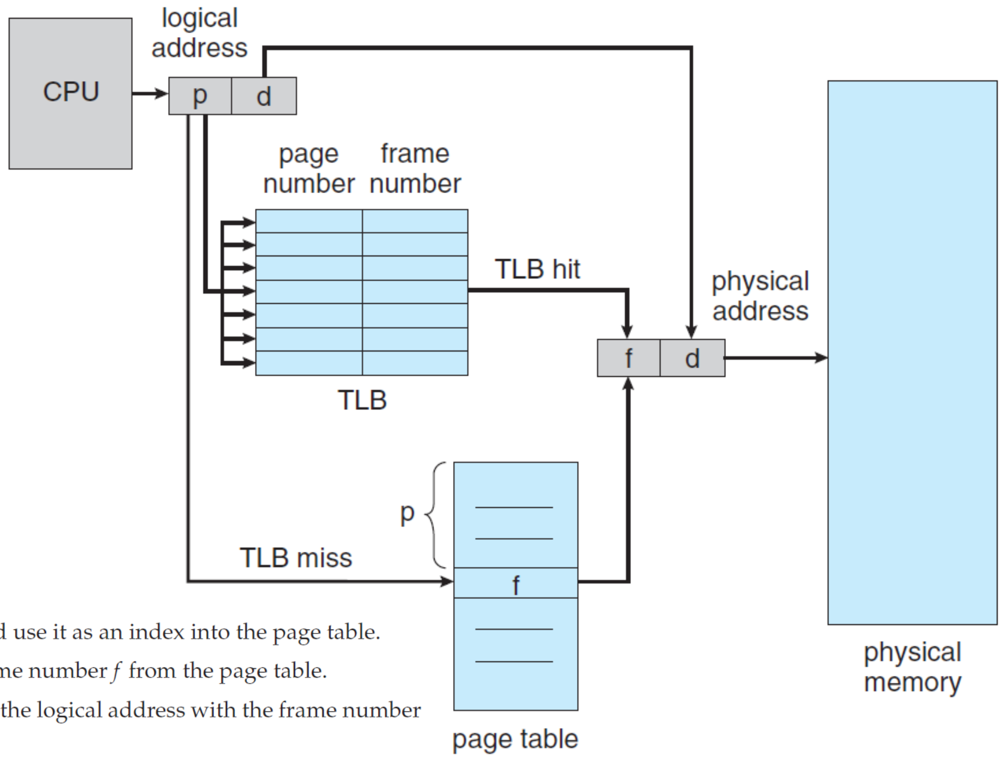
- Hardware Support
  - Translation Look-Aside Buffer (TLB)



1. Extract the page number *p* and use it as an index into the page table.
2. Extract the corresponding frame number *f* from the page table.
3. Replace the page number *p* in the logical address with the frame number *f*.

Paging

- Hardware Support
  - Translation Look-Aside Buffer (TLB)



1. Extract the page number *p* and use it as an index into the page table.
2. Extract the corresponding frame number *f* from the page table.
3. Replace the page number *p* in the logical address with the frame number *f*.

Hit Ratio & effective memory-access time:

The percentage of times that the page number of interest is found in the TLB is called the hit ratio. **Example**: An 80-percent hit ratio means that we find the desired page number in the TLB 80 percent of the time.

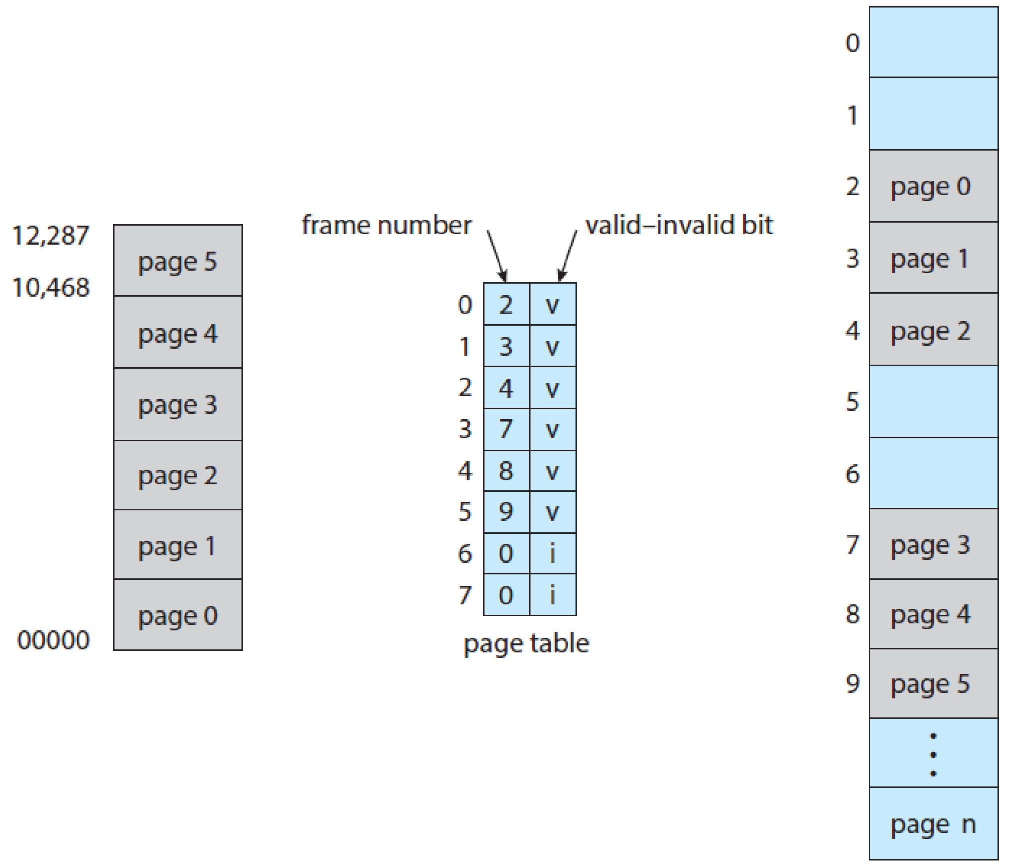
$$\begin{aligned}\text{effective access time} &= 0.80 \times 10 + 0.20 \times 20 \\ &= 12 \text{ nanoseconds}\end{aligned}$$

For a 99-percent hit ratio,

$$\begin{aligned}\text{effective access time} &= 0.99 \times 10 + 0.01 \times 20 \\ &= 10.1 \text{ nanoseconds}\end{aligned}$$

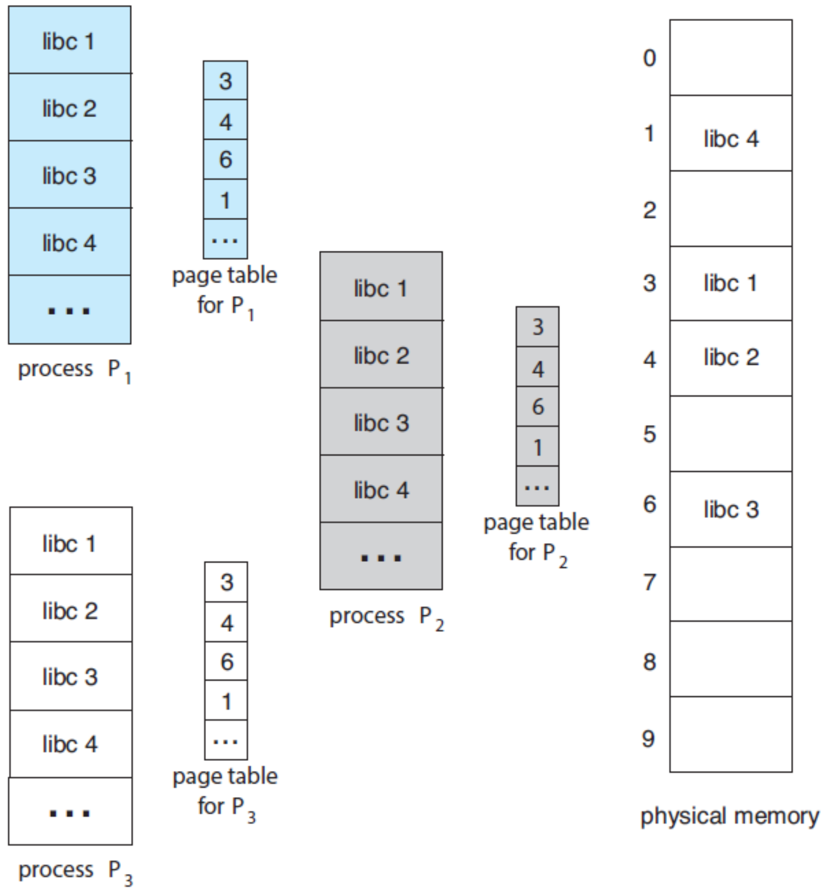
Paging

- Hardware Support
  - Protection



Paging

- Hardware Support
  - Shared Pages

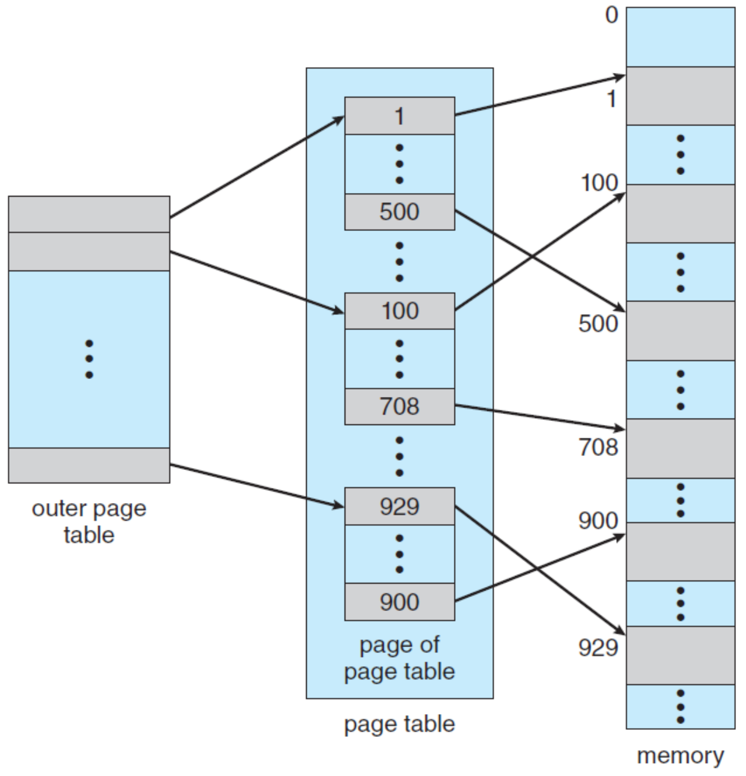


Paging

- Structure of the Page Table
  - Hierarchical Paging
  - Hashed Page Tables
  - Inverted Page Tables

Paging

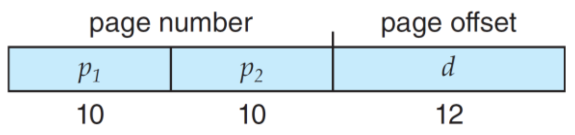
- Structure of the Page Table
  - Hierarchical Paging



Example:

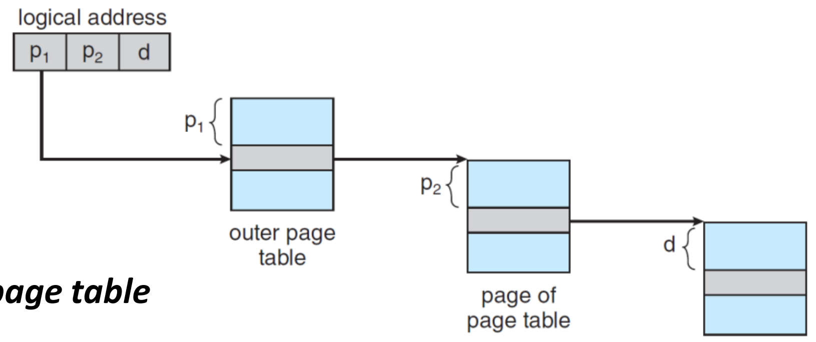
Consider a system with a 32-bit logical address space and a page size of 4 KB. A logical address is divided into a page number consisting of 20 bits and a page offset consisting of 12 bits. Because we page the page table, the page number is further divided into a 10-bit page number and a 10-bit page offset.

Thus, a logical address is as follows:



where p1 is an index into the outer page table and p2 is the displacement within the page of the inner page table.

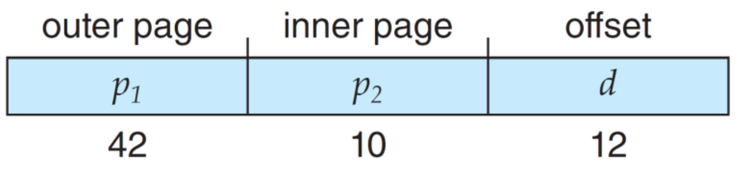
➢ *The address-translation method:*



A forward-mapped page table

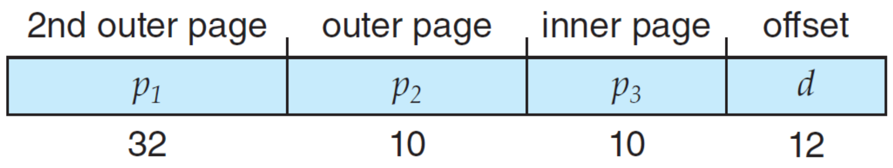
Paging

- Structure of the Page Table
  - Hierarchical Paging (cont)
    - For the page size of 4 KB (2^{12}) in such a system:
      - The page table consists of up to 2^{52} entries. If we use a two-level paging scheme, then the inner page tables can conveniently be one page long, or contain 2^{10} 4-byte entries. The addresses look like this:



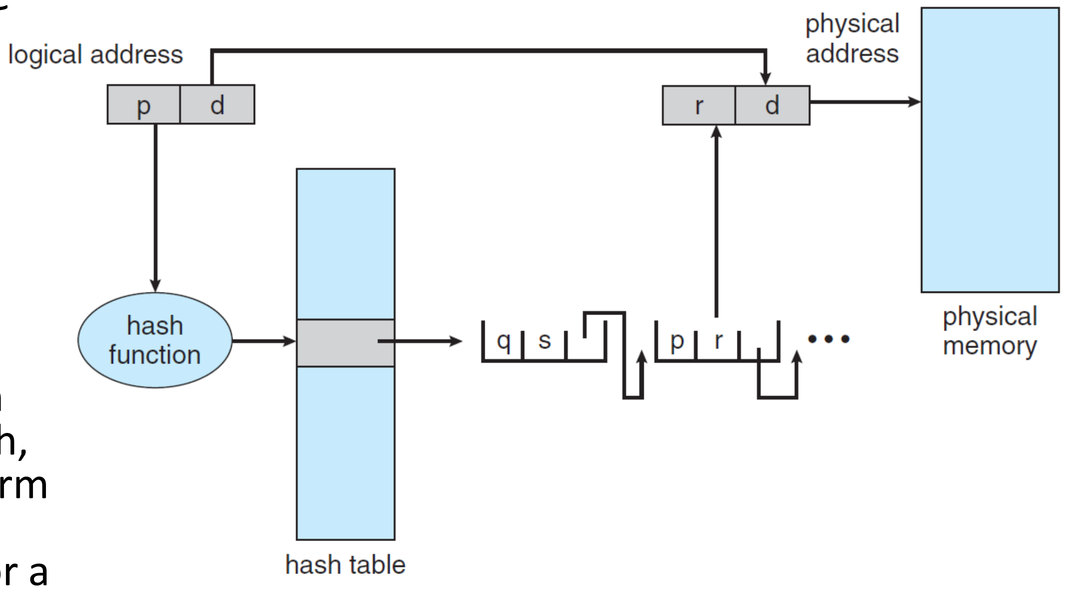
      - The outer page table consists of 2^{42} entries, or 2^{44} bytes → a very big table

      - Way to avoid such a large table is obviously to **divide the outer page table** into smaller pieces



Paging

- Structure of the Page Table
  - Hashed Page Tables
    - Each entry in the hash table contains a linked list of elements that hash to the same location (to handle collisions).
    - Each element consists of three fields:
      1. the virtual page number
      2. the value of the mapped page frame
      3. a pointer to the next element in the linked list.
    - The algorithm:
      - The virtual page number is compared with field 1 in the first element in the linked list. If there is a match, the corresponding page frame (field 2) is used to form the desired physical address. If there is no match, subsequent entries in the linked list are searched for a matching virtual page number.



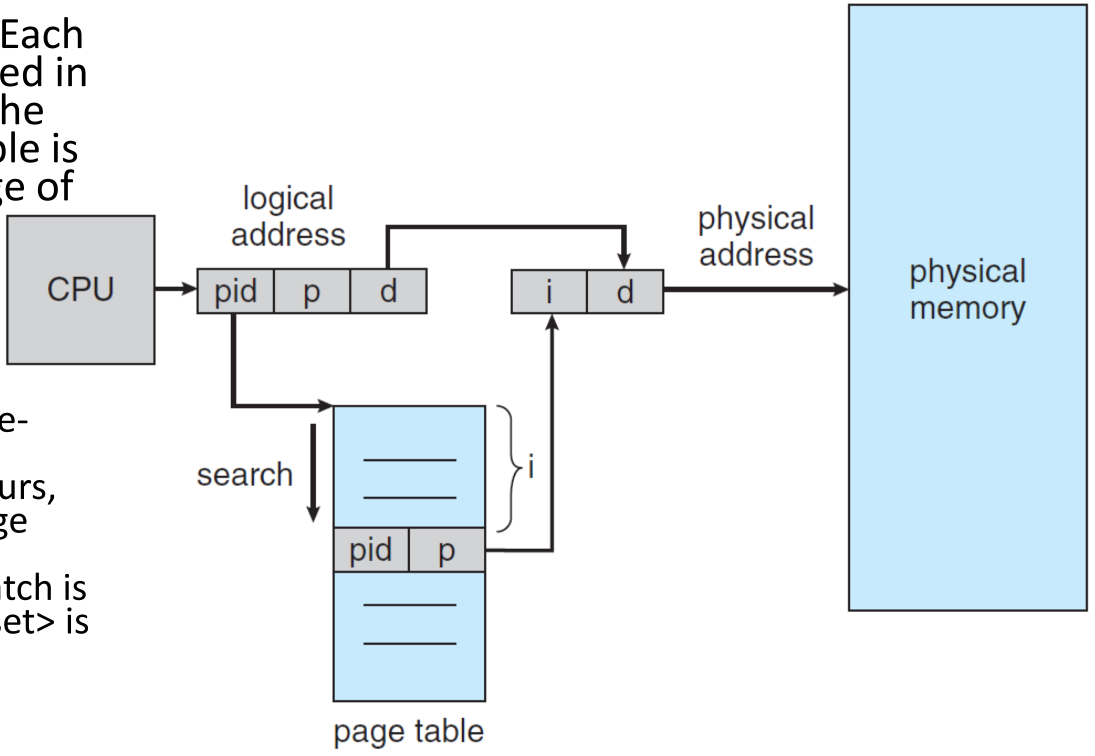
Paging

- Structure of the Page Table
  - Inverted Page Tables
    - One entry for each real page (or frame) of memory. Each entry consists of the virtual address of the page stored in that real memory location, with information about the process that owns the page. Thus, only one page table is in the system, and it has only one entry for each page of physical memory.
    - **Example (IBM RT)**:

      Each virtual address in the system consists of a triple:

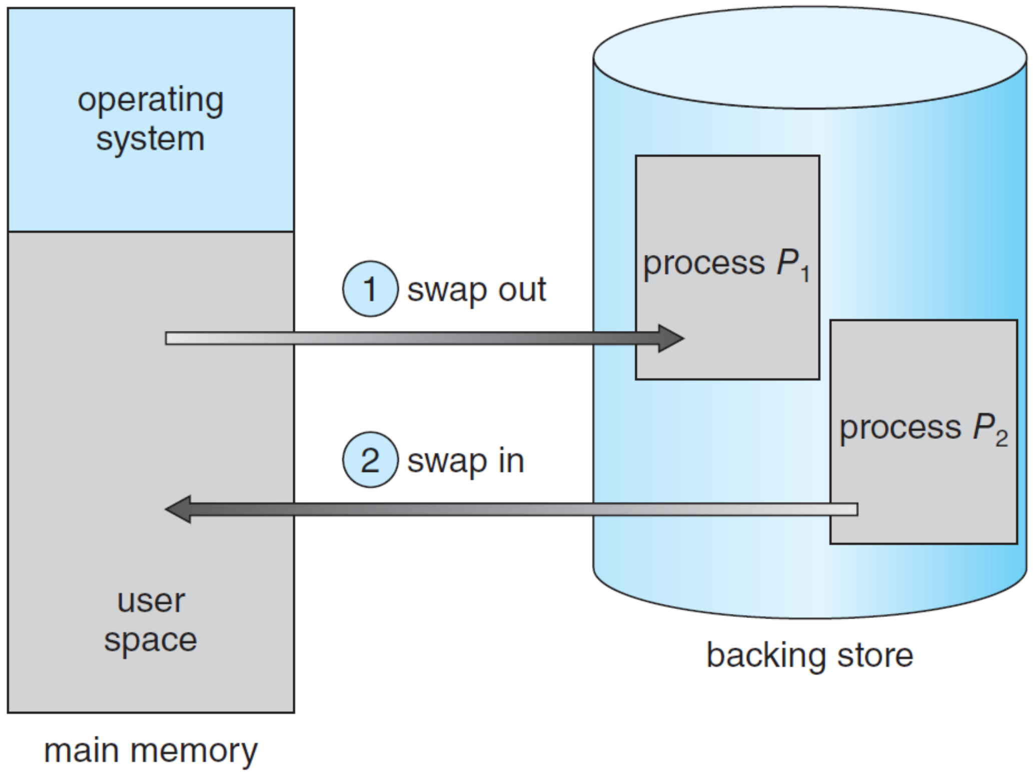
      <process-id, page-number, offset>.

      Each inverted page-table entry is a pair <process-id, page-number> where the process-id assumes the role of the address-space identifier. When a memory reference occurs, part of the virtual address, consisting of <process-id, page number>, is presented to the memory subsystem. The inverted page table is then searched for a match. If a match is found—say, at entry i—then the physical address <i, offset> is generated. If no match is found, then an illegal address access has been attempted.



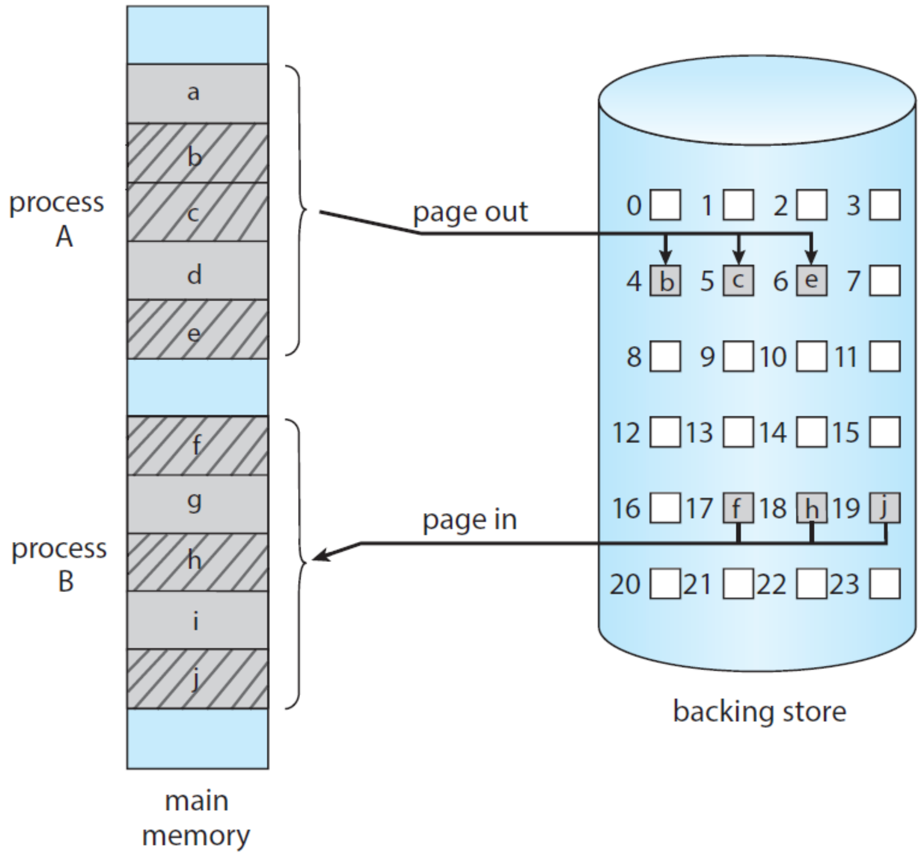
Swapping

- Example of **standard swapping** of two processes using a disk as a backing store.



Swapping

- Swapping with Paging



Swapping

- Swapping on Mobile Systems
  - Apple’s iOS asks applications to voluntarily relinquish allocated memory. Read-only data (such as code) are removed from main memory and later reloaded from flash memory if necessary. Data that have been modified (such as the stack) are never removed. However, any applications that fail to free up sufficient memory may be terminated by the operating system.
  - Android adopts a strategy similar to that used by iOS. It may terminate a process if insufficient free memory is available. However, before terminating a process, Android writes its application state to flash memory so that it can be quickly restarted.

Self-Study

- Try to relate all the discussed topics in this set of lectures with the standard hardware architecture, e.g.,
  - Intel 32- and 64-bit Architectures
  - ARMv8 Architecture

Reference

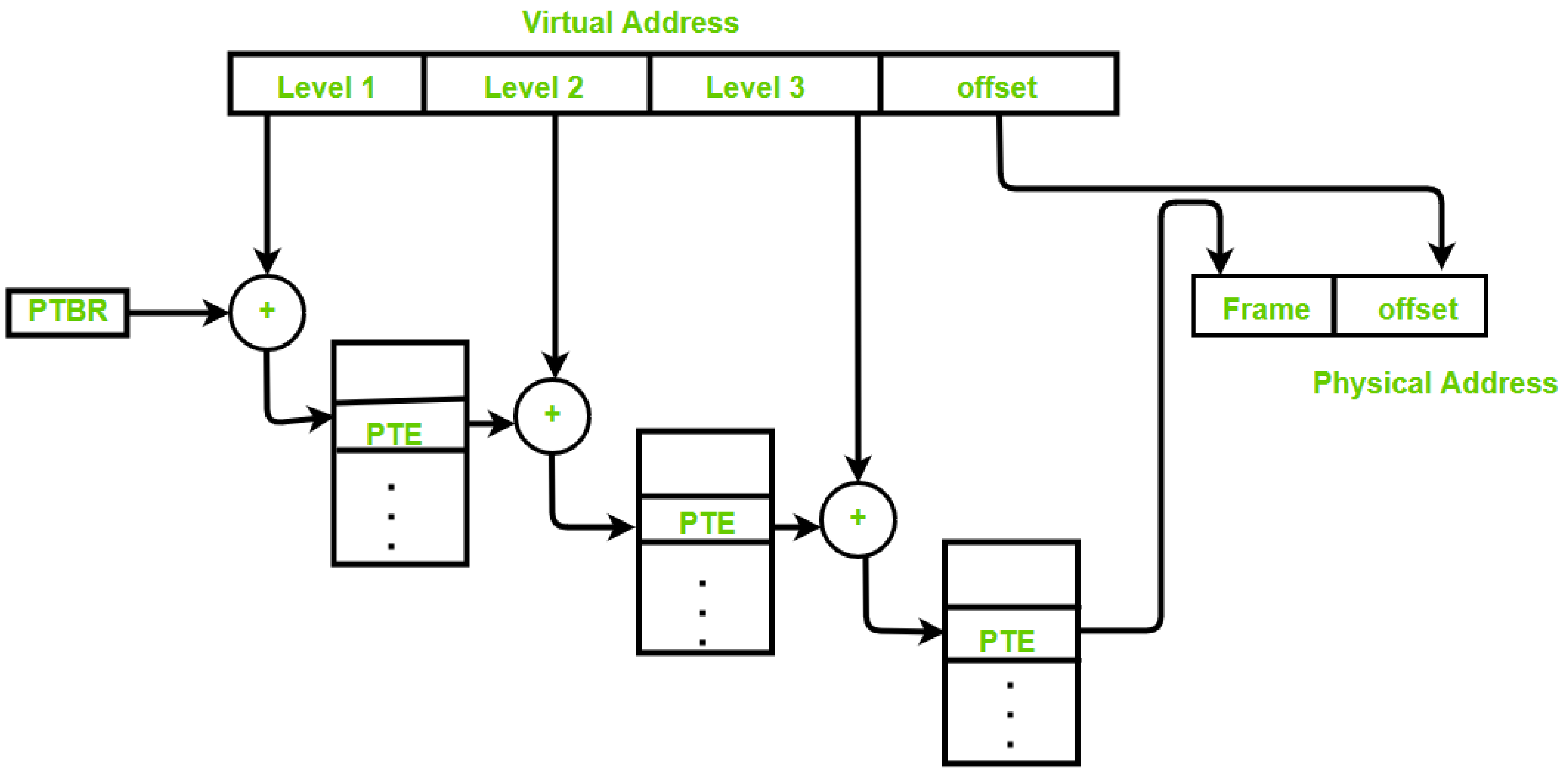
- Abraham Silberschatz , Peter B. Galvin, Greg Gagne, “Operating System Concepts”, Addison Wesley, 10th edition, 2018
  - Chapter 9: Section 9.1 – 9.5

Next

- **Module 5:** Virtual Memory and File Management

Thank You

Appendix



3 Level paging system