Operating System Concepts

Course Code: CSC 2209 Course Title: Operating Systems

Dept. of Computer Science
Faculty of Science and Technology

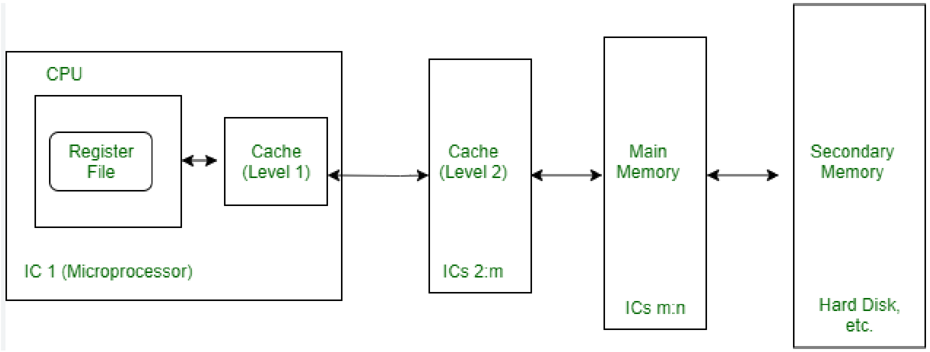
Lecturer No:		**Week No:**		**Semester:**	

Lecture Outline

1. Background
2. Logical and Physical Address Space
3. Static and Dynamic Loading
4. Swapping
5. Fragmentation
6. Paging
7. Virtual Memory

Background

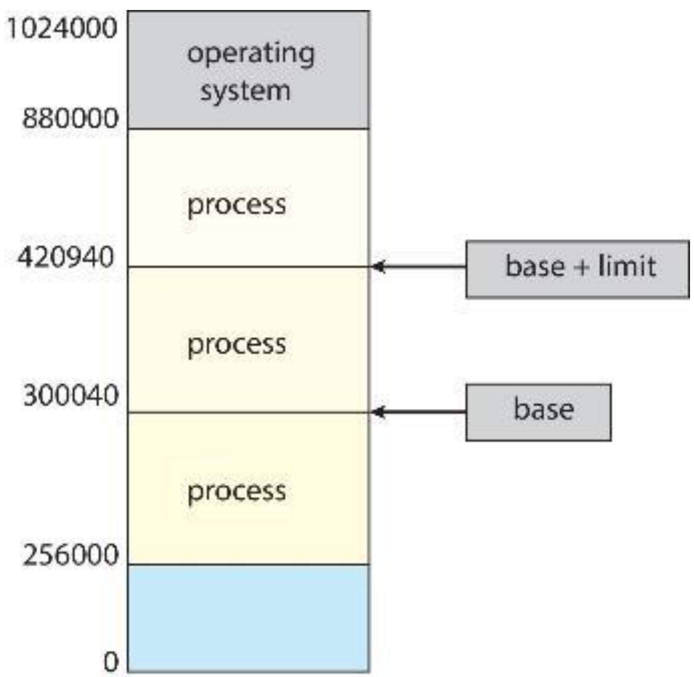
- Program must be brought (from disk) into memory and placed within a process for it to be run
- Main memory and registers are only storage CPU can access directly
- Memory unit only sees a stream of:
  - addresses + read requests, or
  - address + data and write requests
- Register access is done in one CPU clock (or less)
- Main memory can take many cycles, causing a **stall**
- **Cache** sits between main memory and CPU registers

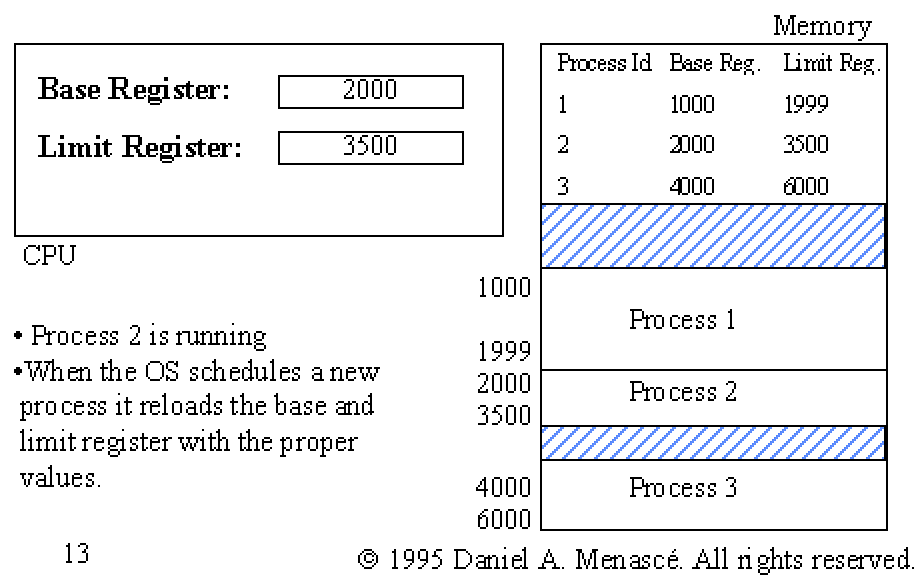


- Protection of memory required to ensure correct operation

Protection

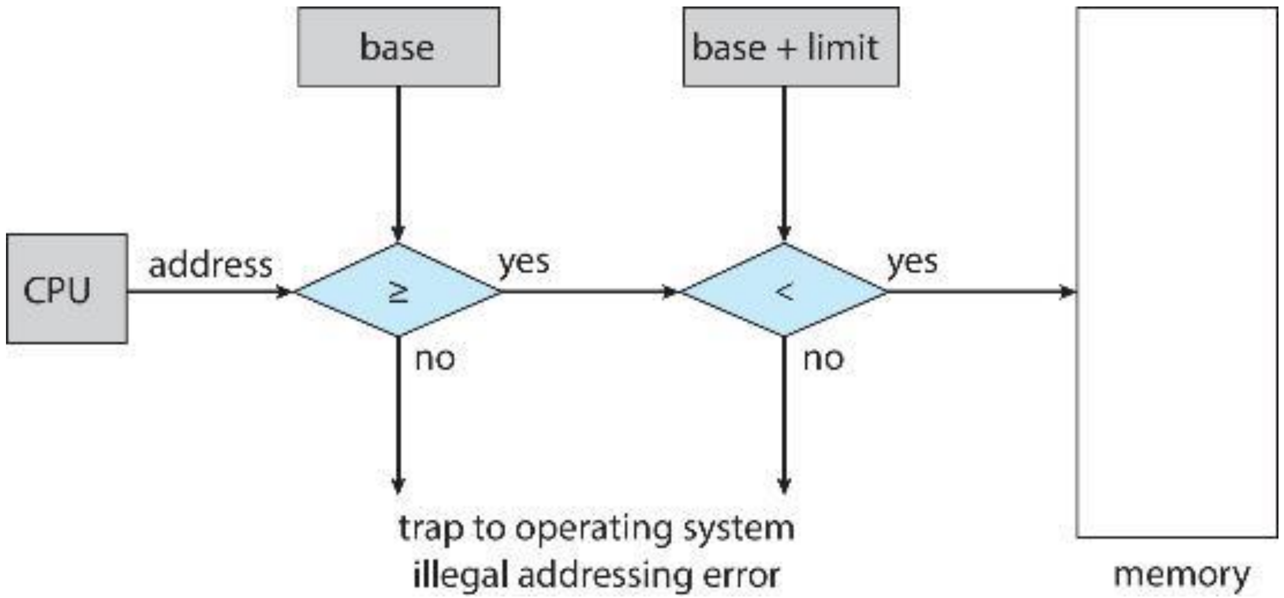
- Need to ensure that a process can **access only** those addresses in its address space.
- We can provide this protection by using a pair of **base** and **limit registers** define the logical address space of a process





Hardware Address Protection

- CPU must check every memory access generated in user mode to be sure it is between **base** and **limit** for that user



- The instructions to loading the base and limit registers are privileged

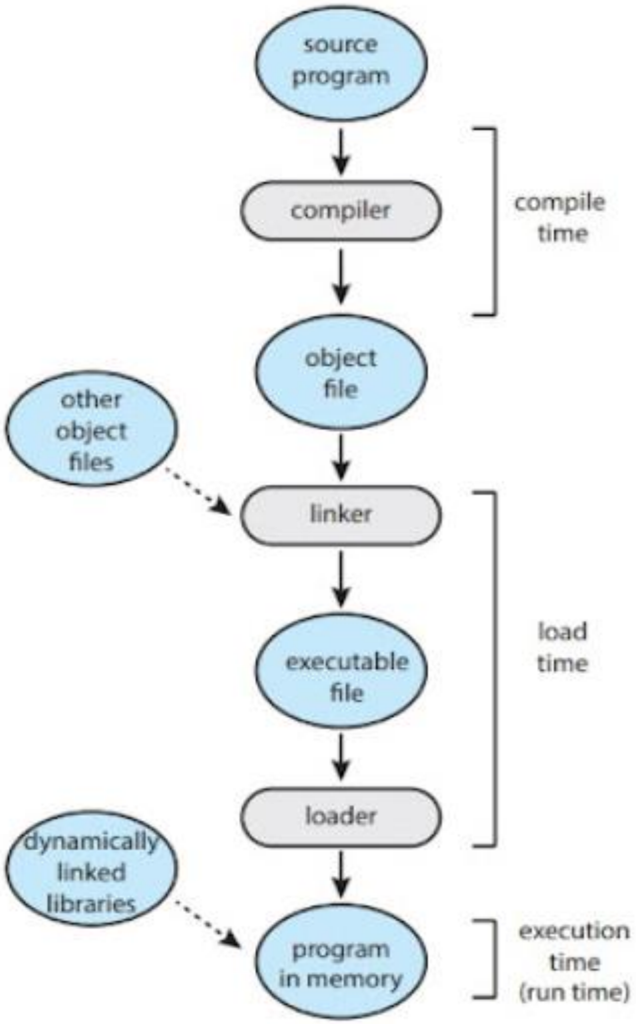
Address Binding

- Programs on disk, ready to be brought into memory to execute from an **input queue**
  - Without support, must be loaded into address 0000
- Inconvenient to have first user process physical address always at 0000
  - How can it not be?
- Addresses represented in different ways at different stages of a program's life
  - **Source code** addresses usually symbolic address
  - **Compiled code** addresses **bind** to relocatable addresses
    - i.e., "14 bytes from beginning of this module"
  - **Linker or loader** will **bind** relocatable addresses to absolute addresses
    - i.e., 74014
  - Each binding **maps** one address space to another

Binding of Instructions and Data to Memory

Address binding of instructions and data to memory addresses can happen at three different stages

- **Compile time**: If memory location known a priori, **absolute code** can be generated; must recompile code if starting location changes
- **Load time**: Must generate **relocatable code** if memory location is not known at compile time
- **Execution time**: Binding delayed until run time if the process can be moved during its execution from one memory segment to another
  - Need hardware support for address maps (e.g., base and limit registers)

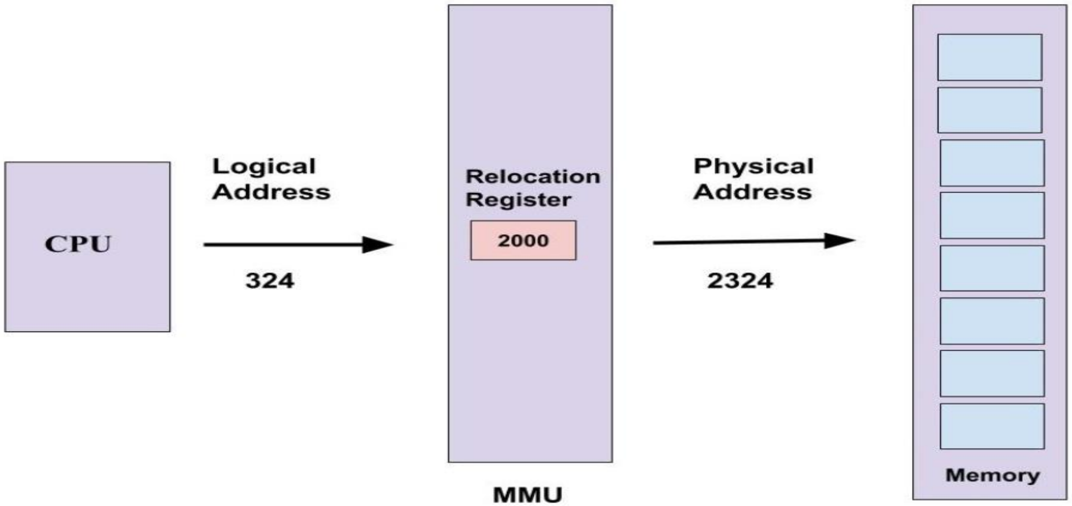


Dynamic Loading

- The entire program does not need to be in memory to execute
- Routine is not loaded until it is called
- Better memory-space utilization; unused routine is never loaded
- All routines kept on disk in relocatable load format
- Useful when large amounts of code are needed to handle infrequently occurring cases
- No special support from the operating system is required
  - Implemented through program design
  - OS can help by providing libraries to implement dynamic loading

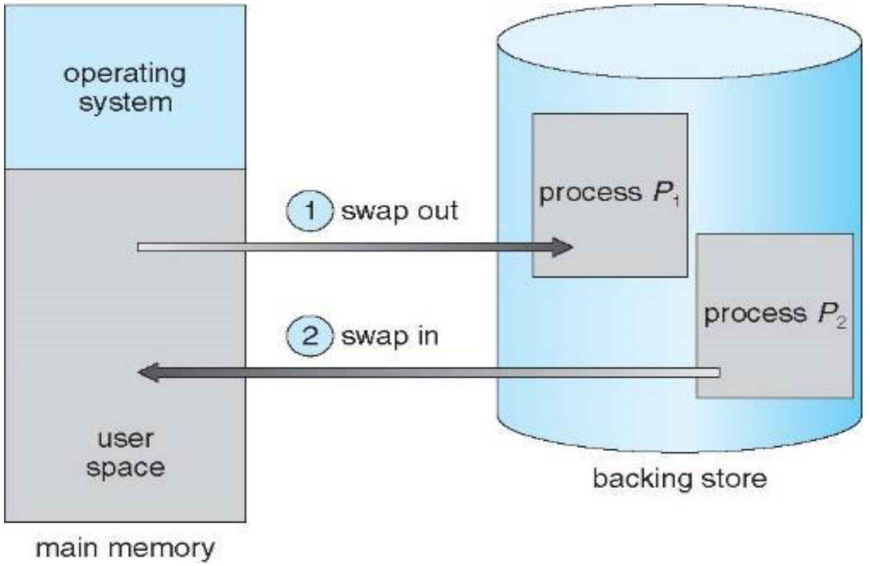
Logical vs. Physical Address Space

- The concept of a logical address space that is bound to a separate **physical address space** is central to proper memory management
  - **Logical address** – generated by the CPU; also referred to as **virtual address**
  - **Physical address** – address seen by the memory unit
- Logical and physical addresses are the same in compile-time and load-time address-binding schemes; logical (virtual) and physical addresses differ in execution-time address-binding scheme
- **Logical address space** is the set of all logical addresses generated by a program
- **Physical address space** is the set of all physical addresses generated by a program



Swapping

- A process can be **swapped** temporarily out of memory to a **backing store**, and then brought **back** into memory (main memory) for continued execution
    - Total physical memory space of processes can exceed physical memory
- **Backing store – fast disk**
    - large enough to accommodate copies of all memory images for all users;
    - must provide direct access to these memory images
- **Roll out, roll in – swapping variant**
    - used for priority-based scheduling algorithms;
    - lower-priority process is swapped out so higher-priority process can be loaded and executed
- Major part of swap time is transfer time;
    - total transfer time is directly proportional to the amount of memory swapped
- System maintains a **ready queue**
    - ready-to-run processes which have memory images on disk



Swapping (Cont.)

- Does the swapped-out process need to swap back into same physical addresses?

- Depends on address binding method
  - Plus consider pending I/O to/from process memory space

- Modified versions of swapping are found on many systems (i.e., UNIX, Linux, and Windows)
  - Swapping normally disabled
  - Started if more than threshold amount of memory allocated
  - Disabled again once memory demand reduced below threshold

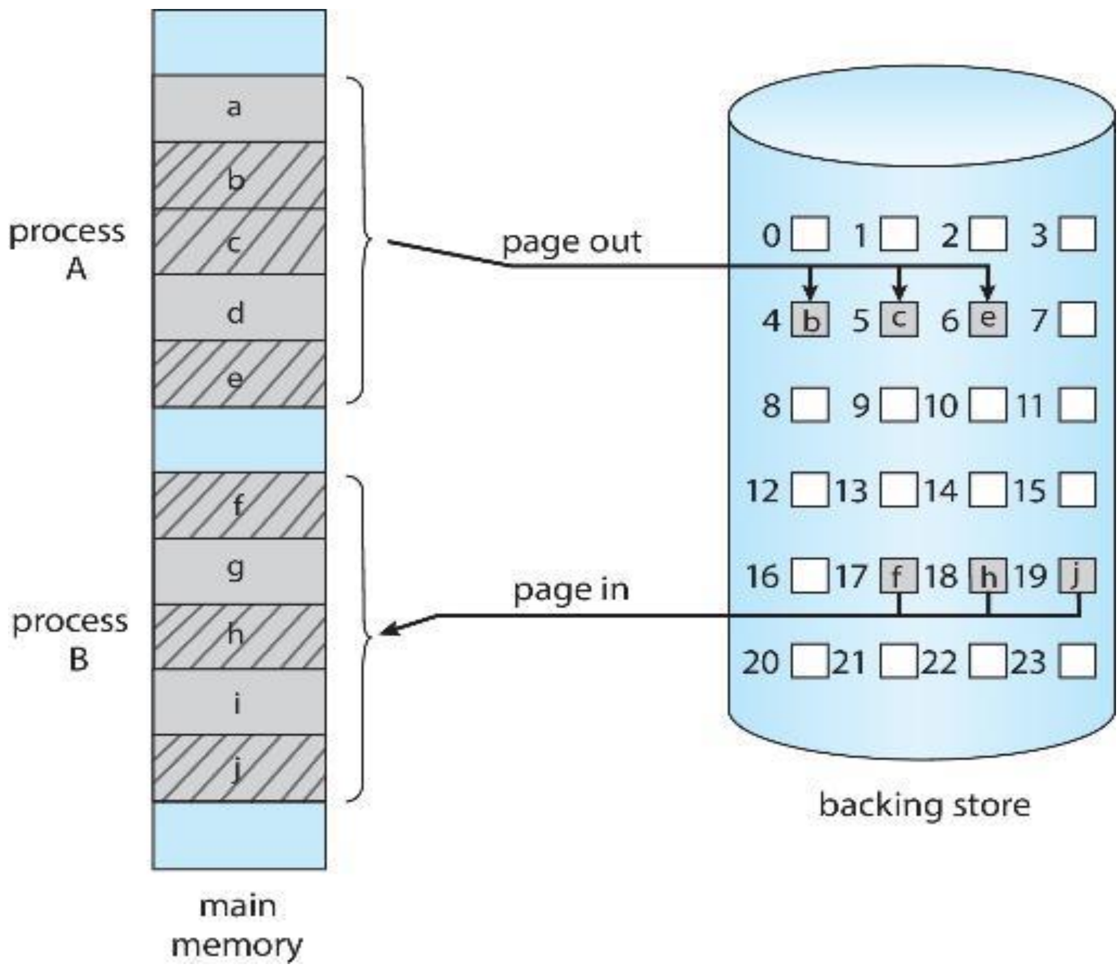
Context Switch Time including Swapping

- If next processes to be put on CPU is not in memory, need to swap out a process and swap in target process
- Context switch time can then be very high
- 100MB process swapping to hard disk with transfer rate of 50MB/sec
  - Swap out time of 2000 ms or 2 sec
  - Swap in of same sized process required same time (2 sec)
  - Total context switch swapping component time of 4000ms (4 seconds)
- Can reduce if reduce size of memory swapped – by knowing how much memory really being used
  - System calls to inform OS of memory use via request_memory() and release_memory()

Context Switch Time and Swapping (Cont.)

- Other constraints as well on swapping
  - Pending I/O – can't swap out as I/O would occur to wrong process
  - Or always transfer I/O to kernel space, then to I/O device
    - Known as **double buffering**, adds overhead
- Standard swapping not used in modern operating systems
  - But modified version common
    - Swap only when free memory extremely low

Swapping with Paging



physical frame numbers

Fragmentation

- **External Fragmentation** – total memory space exists to satisfy a request, but it is not contiguous
- **Internal Fragmentation** – allocated memory may be slightly larger than requested memory; this size difference is memory internal to a partition, but not being used
- **First fit** analysis reveals that given *N* blocks allocated, 0.5 *N* blocks lost to fragmentation
  - 1/3 may be unusable -> **50-percent rule**

Fragmentation (Cont.)

- 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
  - I/O problem
    - Latch job in memory while it is involved in I/O
    - Do I/O only into OS buffers
- Now consider that backing store has same fragmentation problems

Paging

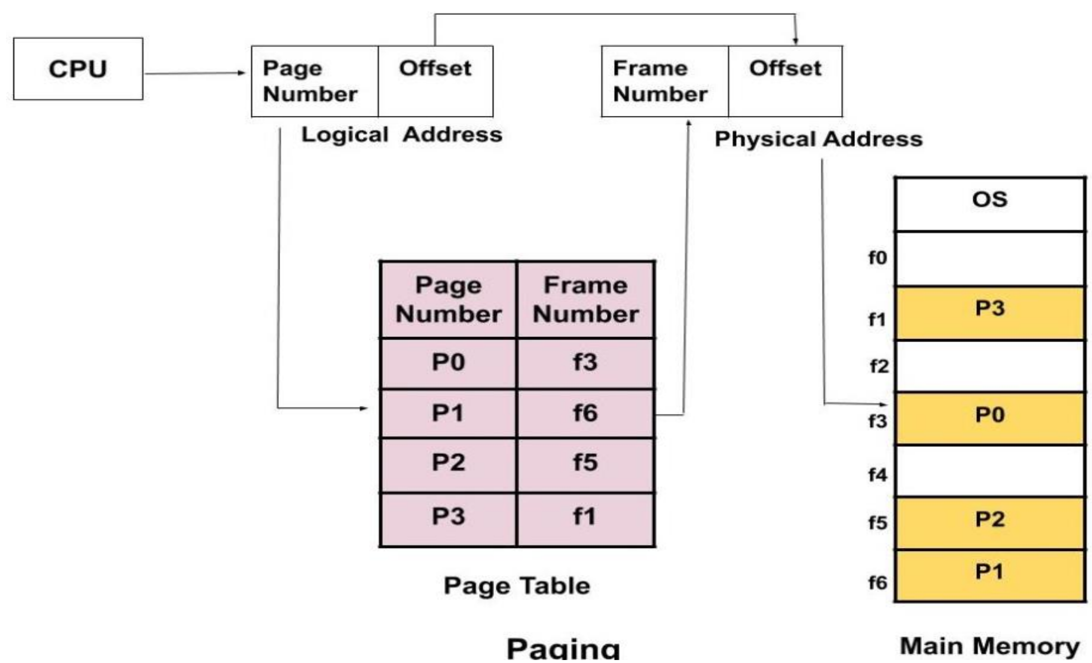
- Physical address space of a process can be noncontiguous; process is allocated physical memory whenever the latter is available
  - Avoids external fragmentation
  - Avoids problem of varying sized memory chunks
- Divide physical memory into fixed-sized blocks called **frames**
  - Size is power of 2, between 512 bytes and 16 Mbytes
- 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
- Backing store likewise split into pages
- Still have Internal fragmentation

Address Translation Scheme

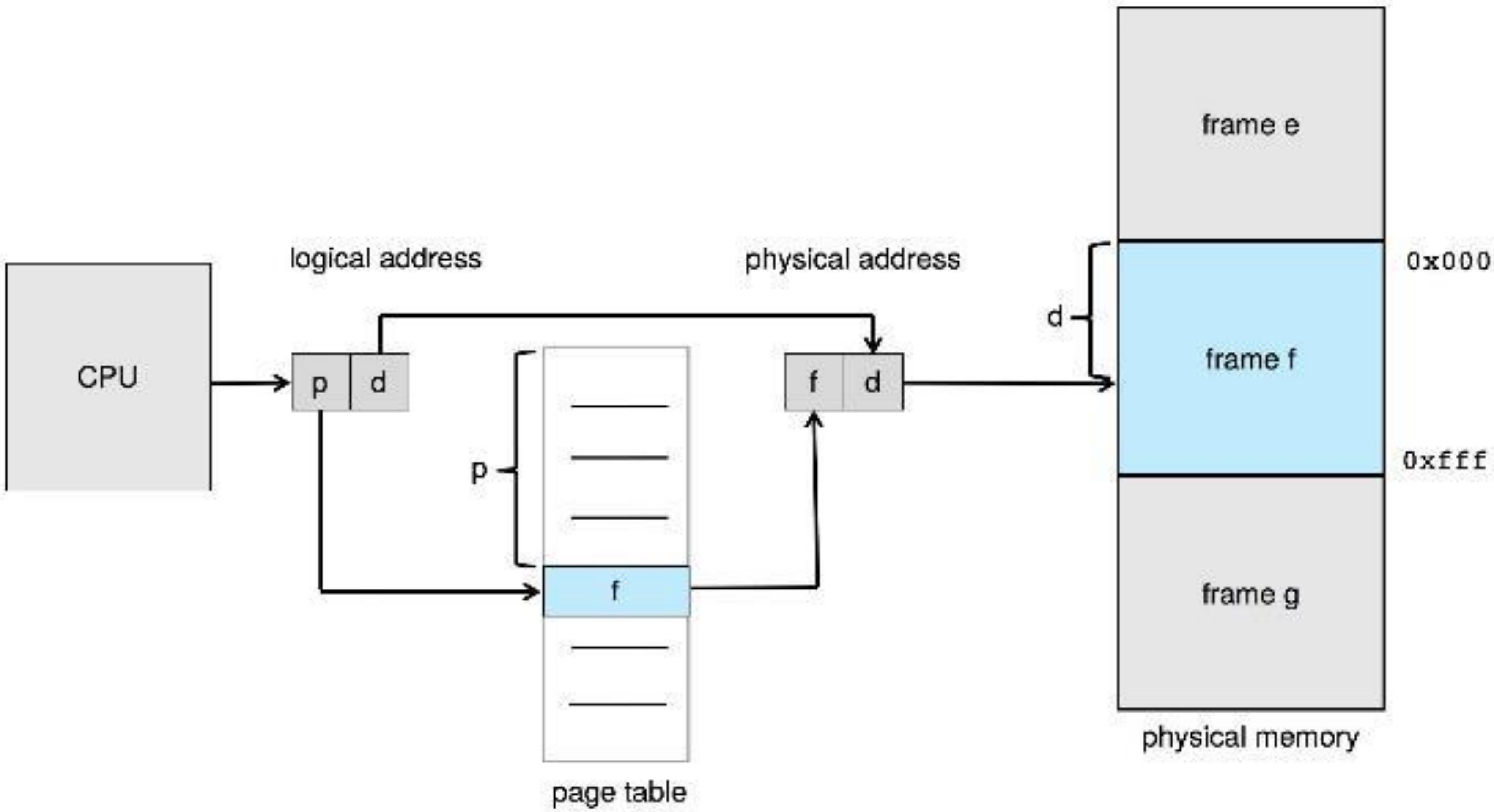
- Address generated by CPU is divided into:
  - **Page number** (***p***) – used as **an index** into a **page table** which contains base address of each page in physical memory
  - **Page offset** (***d***) – combined with base address to define the physical memory address that is sent to the memory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

page number	page offset
p	d
m -n	n

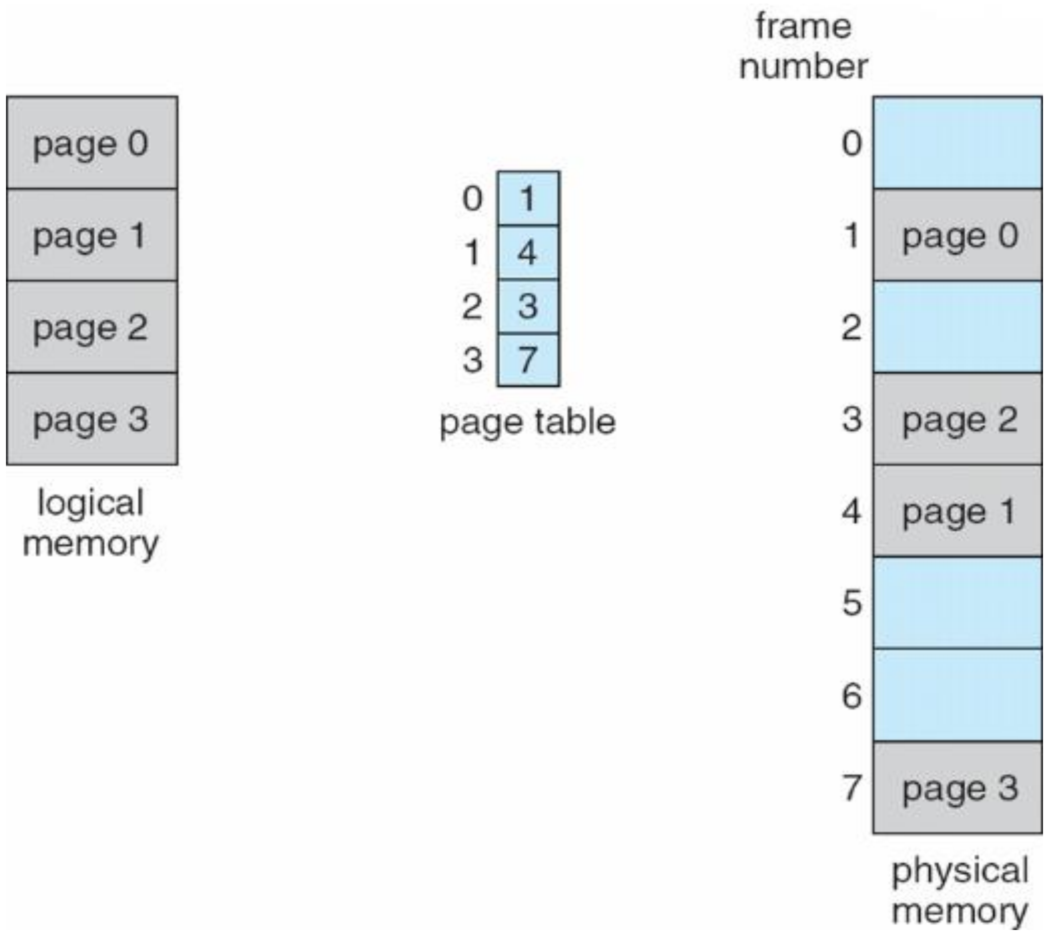
- For given logical address space 2^m and page size 2^n



Paging Hardware

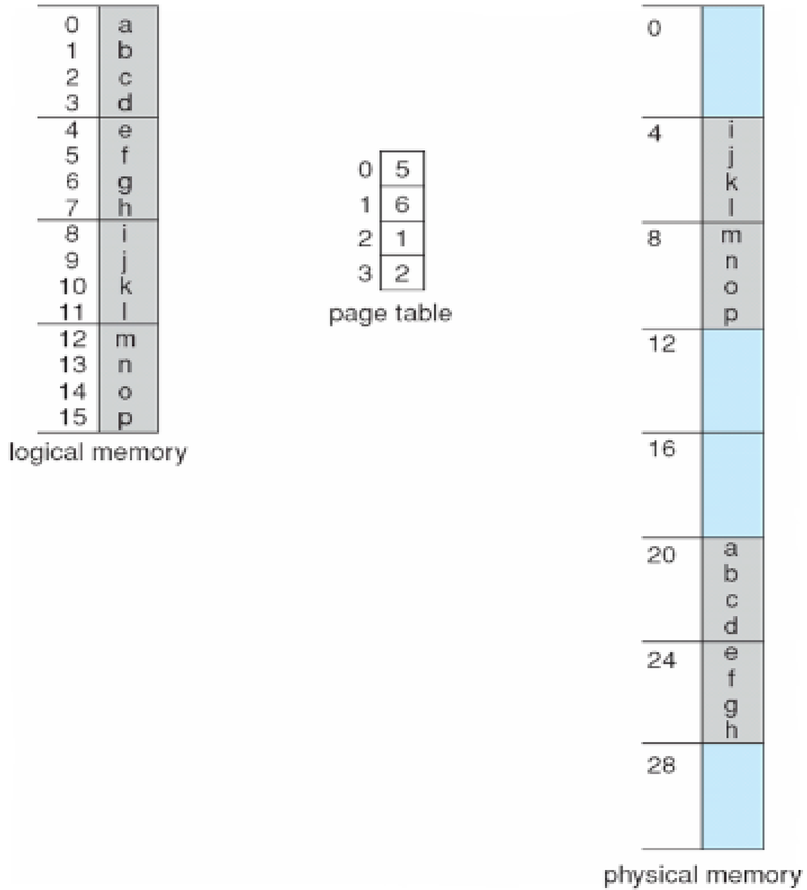


Paging Model of Logical and Physical Memory



Paging Example

- Logical address: n = 2 and m = 4. Using a page size of 4 bytes and a physical memory of 32 bytes (8 pages)



Virtual memory

- **Virtual memory** – separation of user logical memory from physical memory
  - Only part of the program needs to be in memory for execution
  - Logical address space can therefore be much larger than physical address space
  - Allows address spaces to be shared by several processes
  - Allows for more efficient process creation
  - More programs running concurrently
  - Less I/O needed to load or swap processes

Virtual memory (Cont.)

- **Virtual address space** – logical view of how process is stored in memory
  - Usually start at address 0, contiguous addresses until end of space
  - Meanwhile, physical memory organized in page frames
  - MMU must map logical to physical
- Virtual memory can be implemented via:
  - Demand paging
  - Demand segmentation

Books

- Operating Systems Concept
  - Written by Galvin and Silberschatz
  - Edition: 9^{th}

References

- Operating Systems Concept
  - Written by Galvin and Silberschatz
  - Edition: 9^{th}