Information Technology Institute

Introduction to Operating Systems

Prepared by:
Zeyad Ashraf

Memory Management

Difference between 32-bit and 64-bit operating systems

- There are two types of processors existing, i.e., **32-bit** and **64-bit** processors.

- **A 32-bit system** can access2^{32} different memory addresses, i.e 4 GB of RAM or physical memory.

- **A 64-bit system** can access 2^{64} different memory addresses, i.e actually 18-Quintillion bytes (18 exabytes) of RAM.

- A computer with a 64-bit processor can have a 64-bit or 32-bit version of an operating system installed. However, with a 32-bit operating system, the 64-bit processor would not run at its full capability.

3.1 Need of Memory Management

- In systems with multiple programs running in parallel, there could be **many processes in memory at the same time**, and each process may have specific memory needs.
- Processes may need memory for various reasons:
  - **First**, the executable itself may need to be loaded into memory for execution.
  - The **second** item would be the data part of the executable. These could be *hardcoded* strings, text, and variables that are referenced by the process.
  - The **third** type of memory requirement could arise from runtime requests for memory. These could be needed from the *stack/heap* for the program to perform its execution.
- The **OS** and the kernel **components** may also need to be loaded in memory. Additionally.

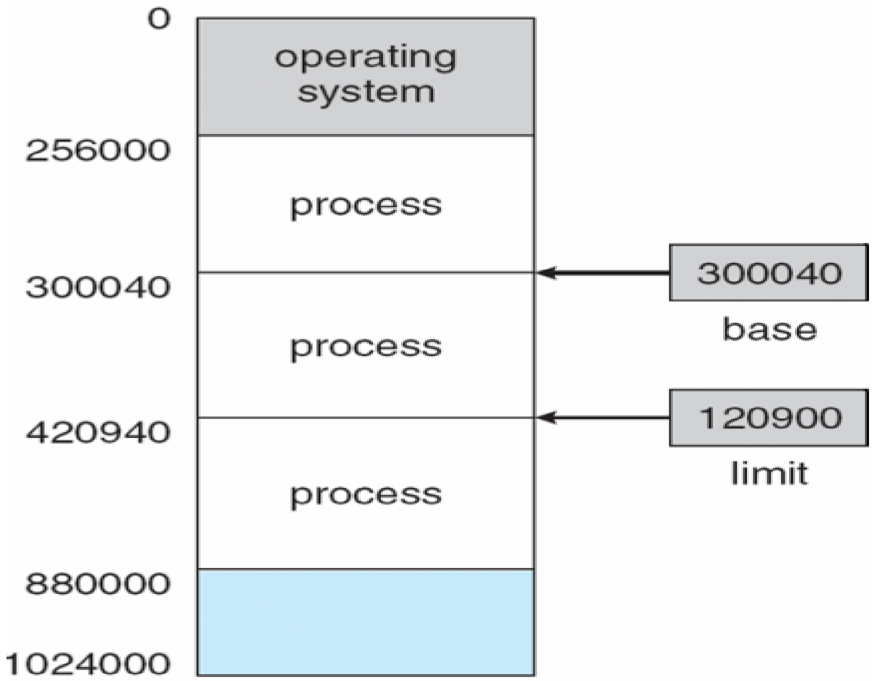
Background

- 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
- 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

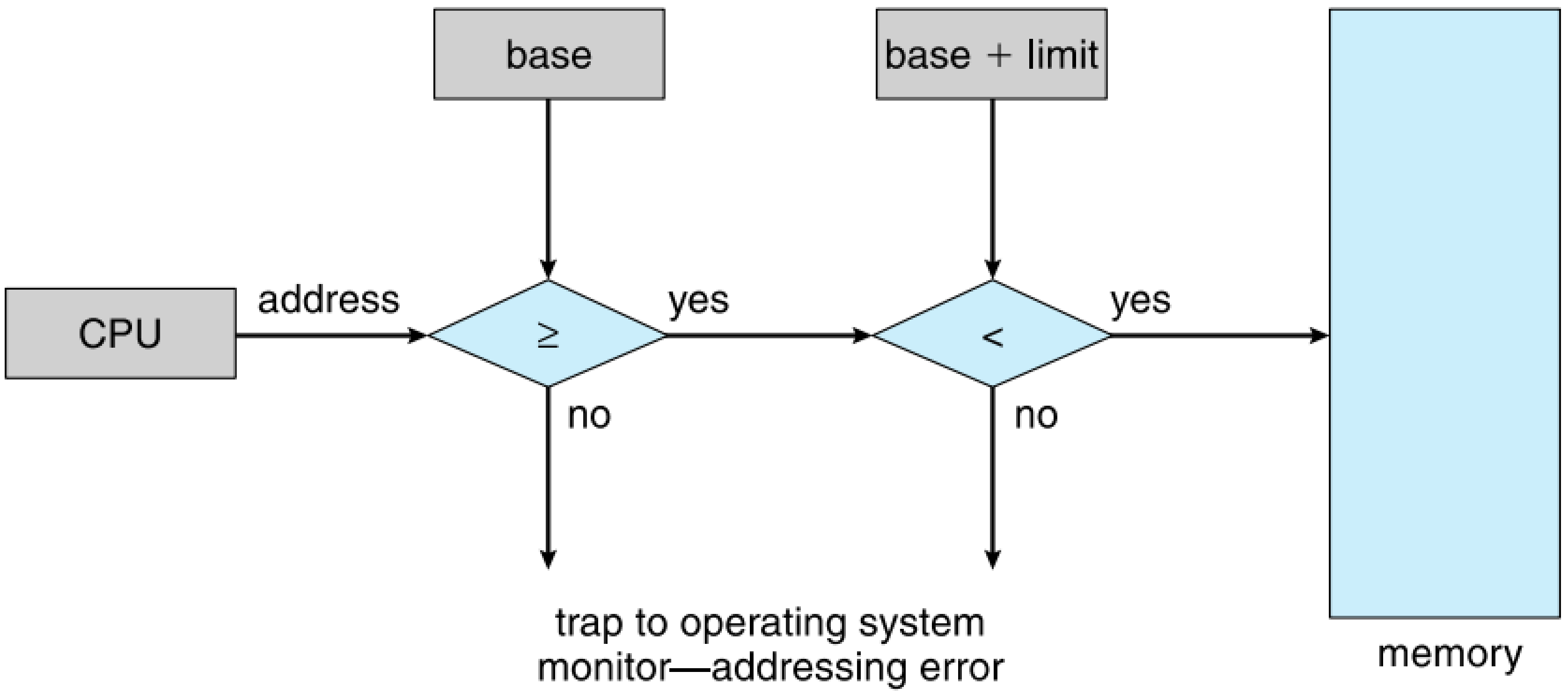
• Base and Limit Registers

- A pair of base and limit registers define the logical address space
- CPU must check every memory access generated in user mode to be sure it is between base and limit for that user

Location = base + limit



• Hardware Address Protection



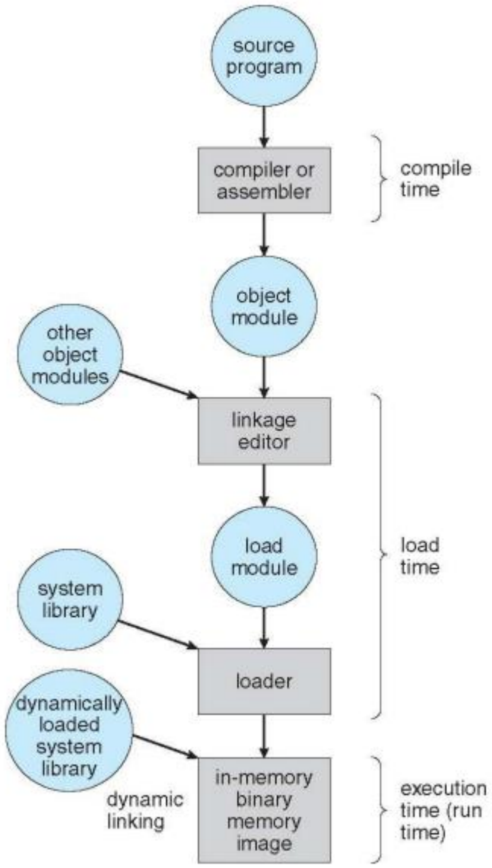
• Address Binding

- Programs on disk, ready to be brought into memory to execute form an **input queue**
- Without support, must be loaded into address 0000 .
- Compiled code addresses bind to **relocatable** addresses.

• 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)

• Multistep Processing of a User Program



• Logical vs. Physical Address Space

Aspect	Logical Address	Physical Address
Definition	The address generated by the CPU while executing a program.	The actual address in the main memory (RAM) where data and instructions are physically stored.
Also Known	Virtual Address	Real Address
Used By	Program (process) and programmer	Operating System and memory hardware
Physical Existence	Does not physically exist in memory; it is only a logical representation	Physically exists in main memory
User Visibility	Visible to the user/programmer (in the code)	Not directly visible to the user
When They Are Identical	Identical only in Compile-Time and Load-Time binding	Different in Execution-Time binding
Used In	Virtual Memory, Paging, Segmentation	Hardware memory access
Memory Management	Managed by the OS and MMU for address translation	Managed by physical hardware

• Memory-Management Unit (MMU)

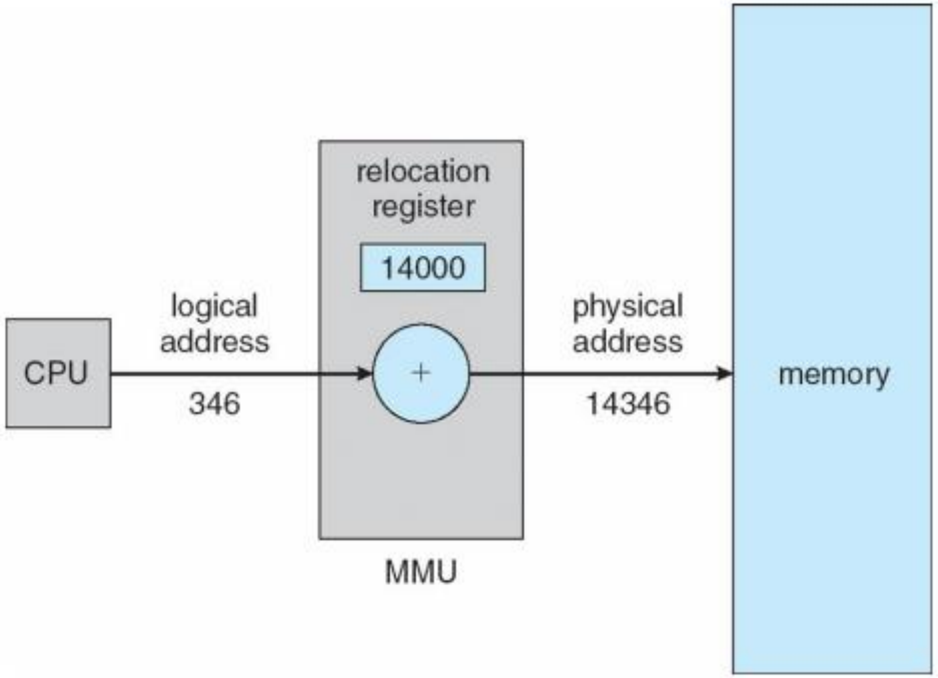
- Hardware device that at run time **maps virtual to physical** address
- Many methods possible, covered in the rest of this chapter

- Base register now called **relocation register**

    ex MS-DOS on Intel 80x86 used 4 relocation registers
- User program deals with logical addresses; it never sees the real physical addresses
- Logical address bound to physical addresses

Dynamic relocation using a relocation register

- 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



• Dynamic Linking

Aspect	Static Linking	Dynamic Linking
Linking Time	Before execution (Compile time or Load time)	During execution (Run time)
Who Performs the Linking	Loader or Linker	Loader + Operating System + Stub
Libraries	Fully included inside the executable program	Loaded only when needed
Executable File Size (EXE)	Large, because everything is embedded	Smaller, since libraries are not included
Memory Usage	High (duplicate library copies in each program)	Low (shared libraries among programs)
Updating / Patching	Difficult — requires rebuilding the program	Easy — just update the shared library
Execution Speed	Faster, since everything is pre-linked	Slightly slower at first load (due to runtime linking)
Sharing Between Programs	Not possible (each program has its own copy)	Possible (multiple programs share the same library)
Examples	`.exe` file contains all the code	Programs using `.dll` in Windows or `.so` in Linux

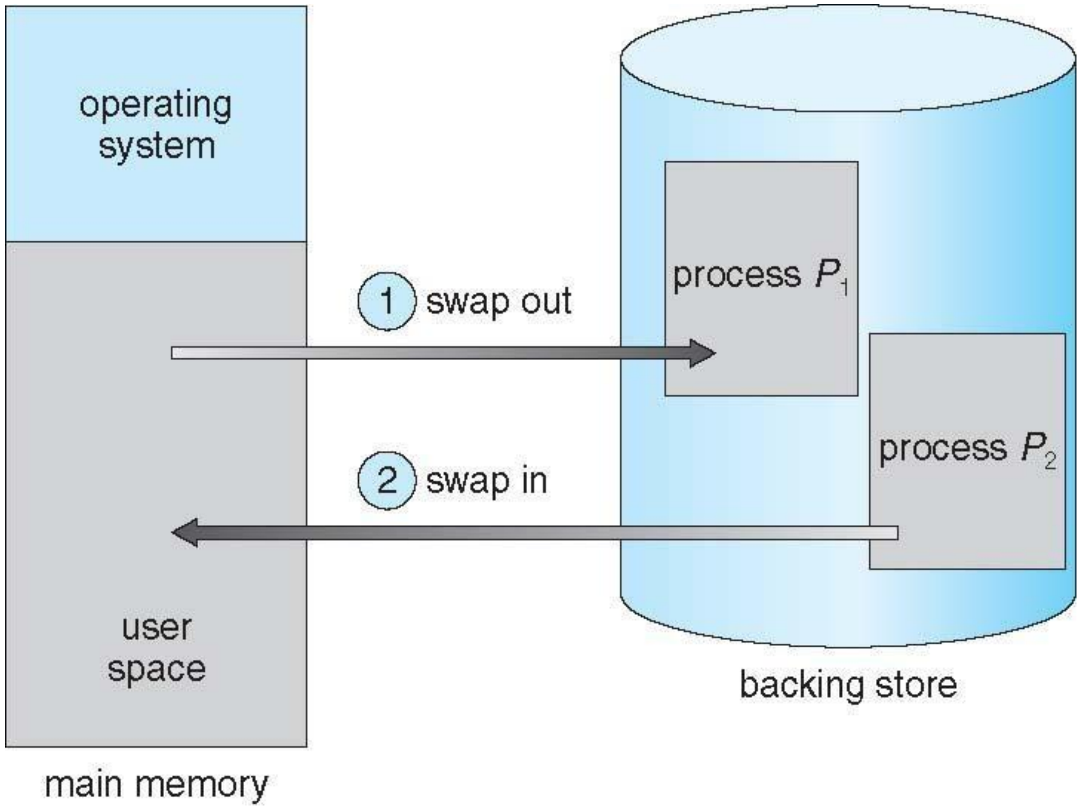
• Swapping

- A process can be **swapped** temporarily out of **memory** to a backing store, and then brought back into 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
- System maintains a **ready queue** of ready-to-run processes which have memory images on disk

• Swapping (Cont.)

- Does the swapped out process need to swap back in to 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)

• Schematic View of Swapping



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
  - Plus swap in of same sized process
  - Total context switch swapping component time of 4000ms (4 seconds)

• 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 on Mobile Systems

- ***Not typically supported (***Flash memory based***)***
    - Small amount of space
    - Limited number of write cycles
    - Poor throughput between flash memory and CPU on mobile platform
- Instead use other methods to free memory if low
    - iOS ***asks*** apps to voluntarily relinquish allocated memory
        - Read-only data thrown out and reloaded from flash if needed
        - Failure to free can result in termination
    - **Android** terminates apps if low free memory, but first writes application state to flash for fast restart
    - Both OSes support paging as discussed below

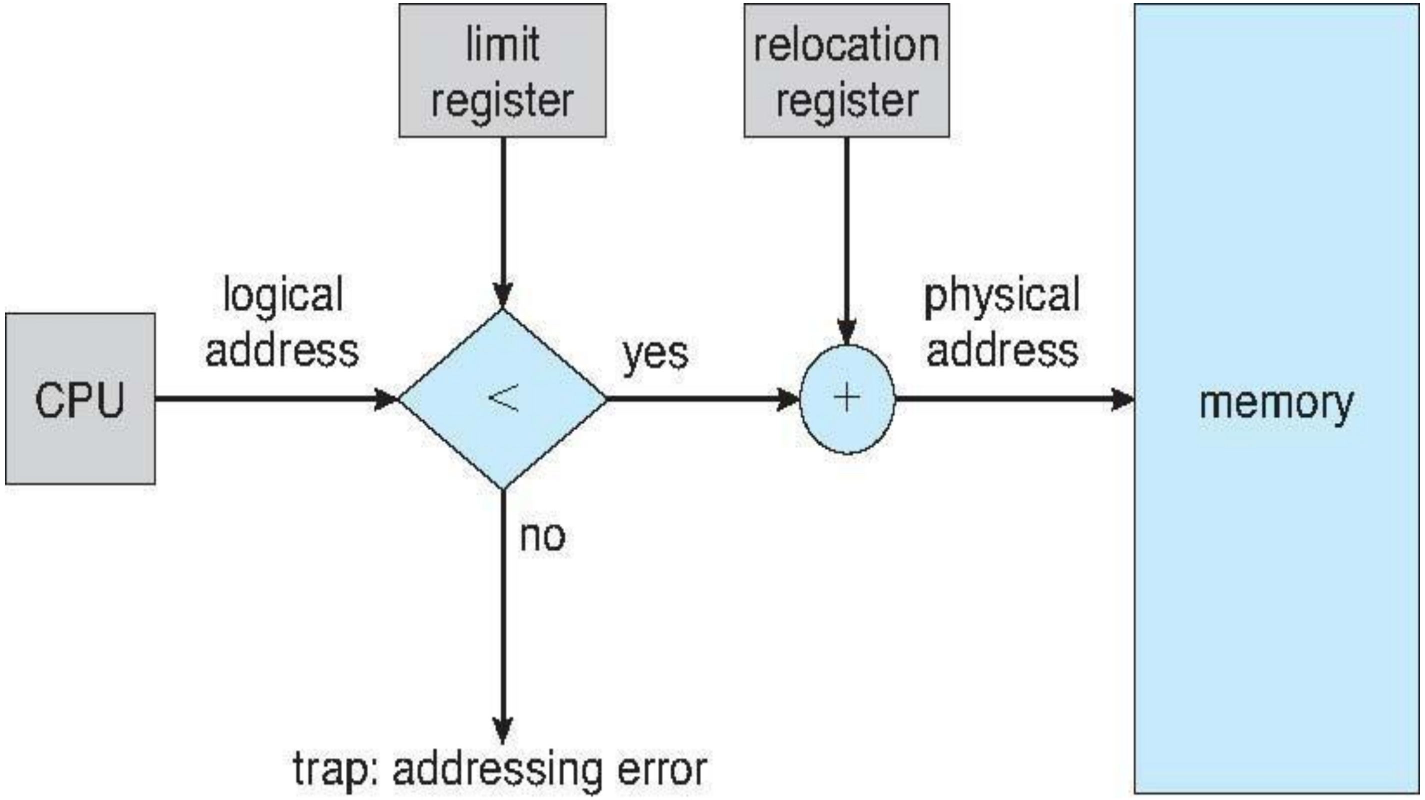
• Contiguous Allocation

- Main memory must support both OS and user processes
- Limited resource, must allocate efficiently
- Contiguous allocation is one early method
- Main memory usually into two **partitions**:
  - **Resident OS**, usually held in low memory with interrupt vector
  - **User processes** then held in high memory
  - Each process contained in single contiguous section of memory

• Contiguous Allocation (Cont.)

- **Relocation registers** used to protect user processes from each other, and from changing operating-system code and data
  - Base register contains value of smallest physical address
  - Limit register contains range of logical addresses – each logical address must be less than the limit register
  - MMU maps logical address *dynamically to physical*
  - Can then allow actions such as **kernel changing size**

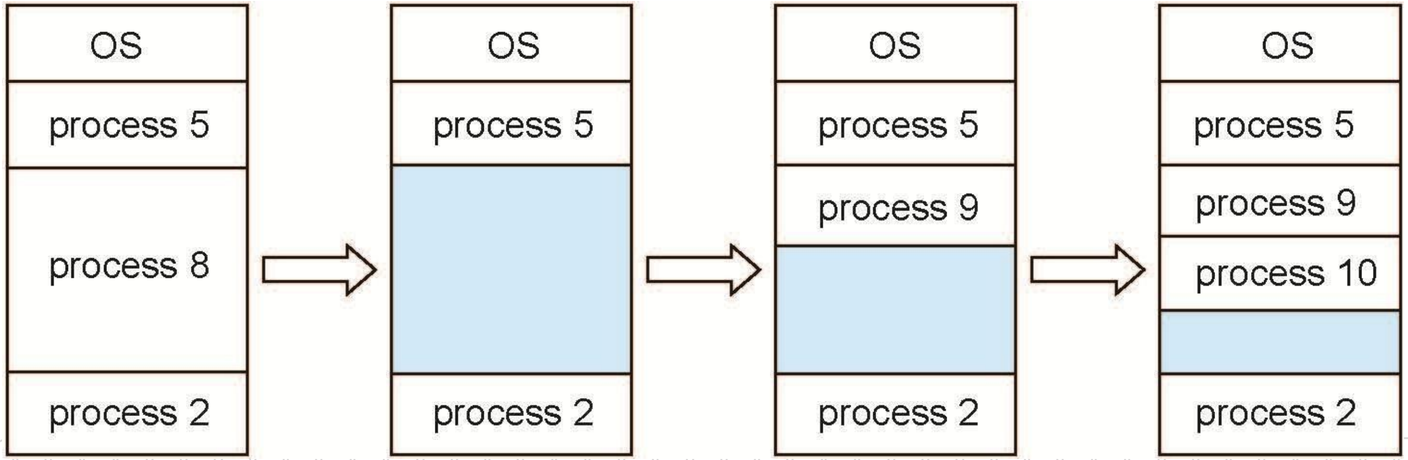
• Hardware Support for Relocation and Limit Registers



• Multiple-partition allocation

- Multiple-partition allocation
  - Degree of multiprogramming limited by number of **partitions**
  - **Hole** – block of available memory; holes of various size are scattered throughout memory
  - When a process arrives, **it is allocated memory from a hole large enough** to accommodate it
  - Process exiting frees its partition, adjacent free partitions combined
  - Operating system maintains information about:

    a) allocated partitions   b) free partitions (hole)



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 and best-fit better than worst-fit in terms of speed and storage utilization

• 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**

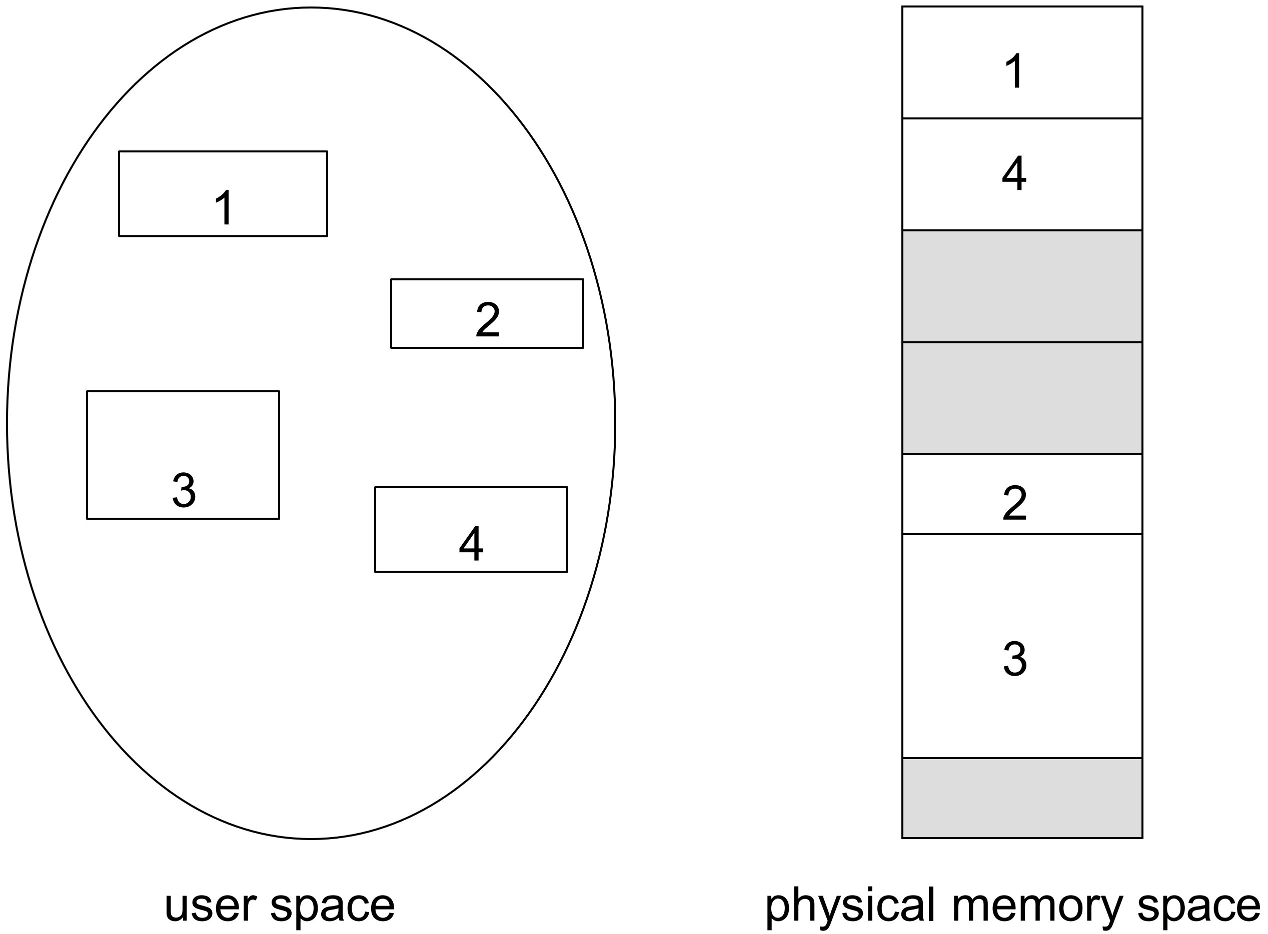
Fragmentation (Cont.)

- 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

• 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
    - Common Block
    - Stack
    - Symbol Table
    - Arrays

Logical View of Segmentation



Segmentation Architecture

- Logical address consists of a two tuple:

  <segment-number, offset>,

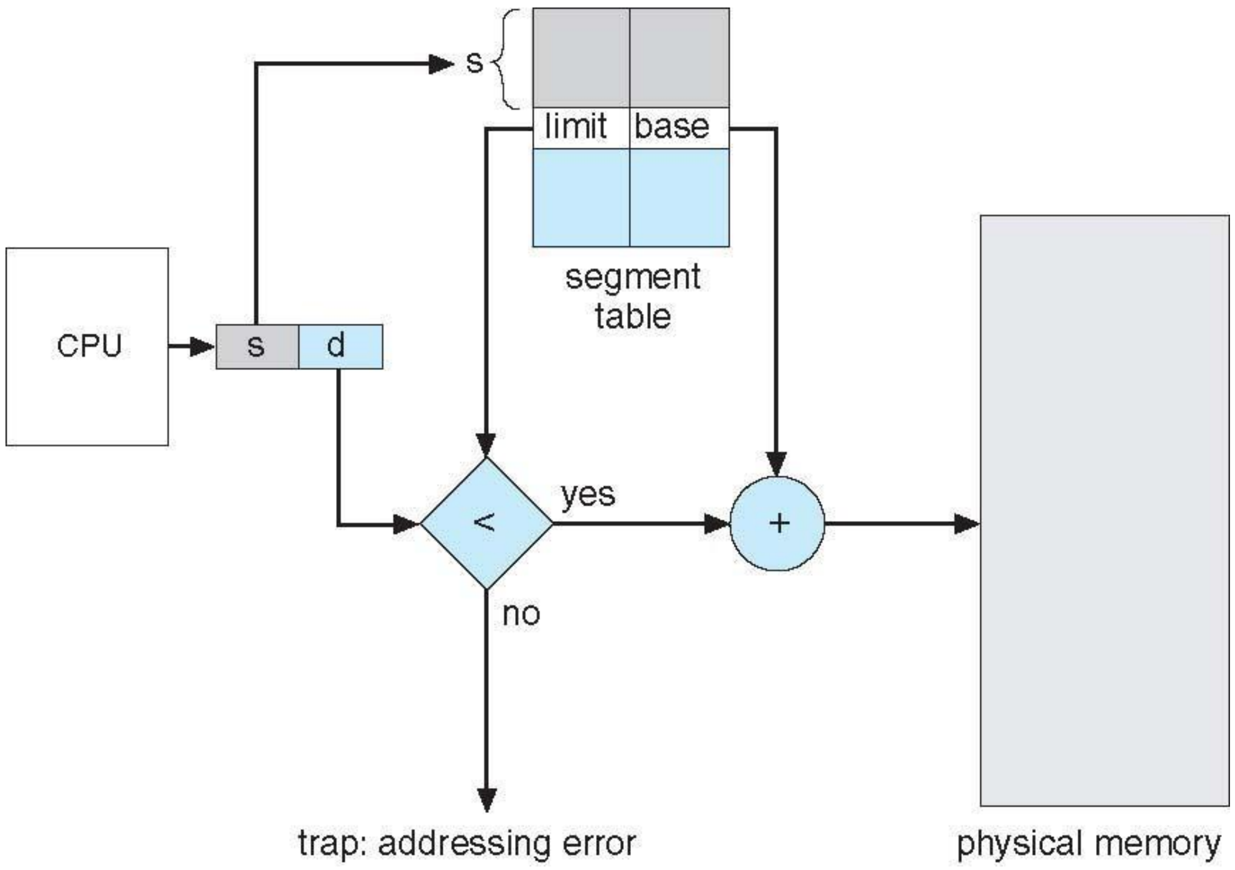
- **Segment table** – maps two-dimensional physical addresses; each table entry has:
  - **base** – contains the starting physical address where the segments reside in memory
  - **limit** – specifies the length of the segment
- **Segment-table base register (STBR)** points to the segment table's location in memory
- **Segment-table length register (STLR)** indicates number of segments used by a program;

  segment number ***s*** is legal if ***s*** < **STLR**

Segmentation Architecture (Cont.)

- Protection
  - With each entry in segment table associate:
    - validation bit = 0 $\Rightarrow$ illegal segment
    - read/write/execute privileges
- Protection bits associated with segments; code sharing occurs at segment level
- Since segments vary in length, memory allocation is a dynamic storage-allocation problem
- A segmentation example is shown in the following diagram

Segmentation Hardware



Paging

- Physical address space of a process can be **noncontiguous**; process is allocated physical memory whenever the latter is available
  - Avoids external fragmentation
- 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**
- 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
- Still have Internal fragmentation

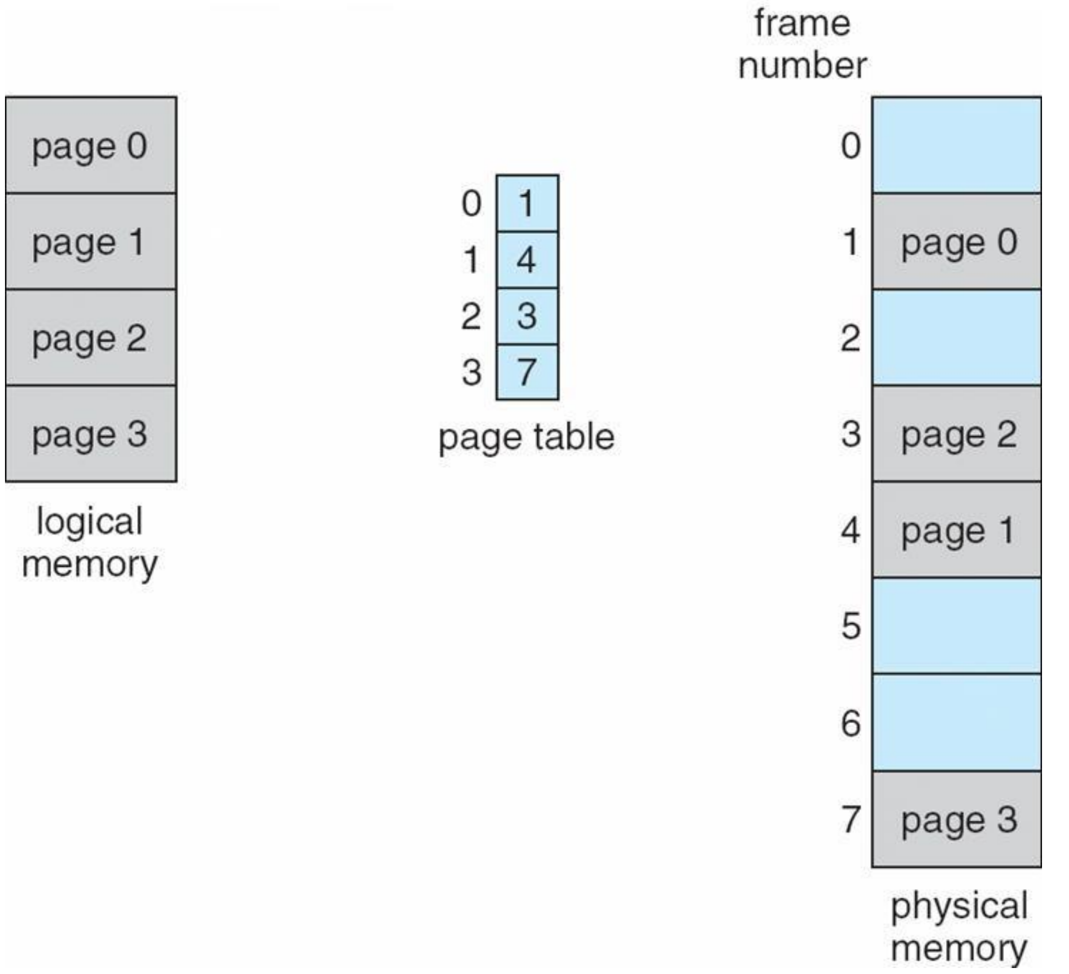
• Address Translation Scheme

- 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

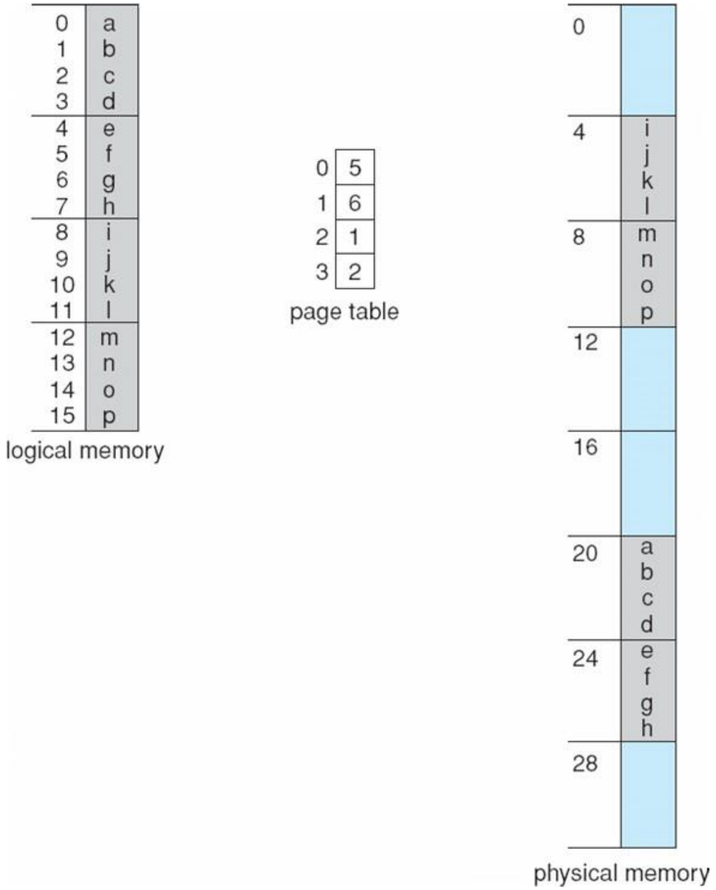
page number	page offset
p	d
m -n	n

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

• Paging Model of Logical and Physical Memory

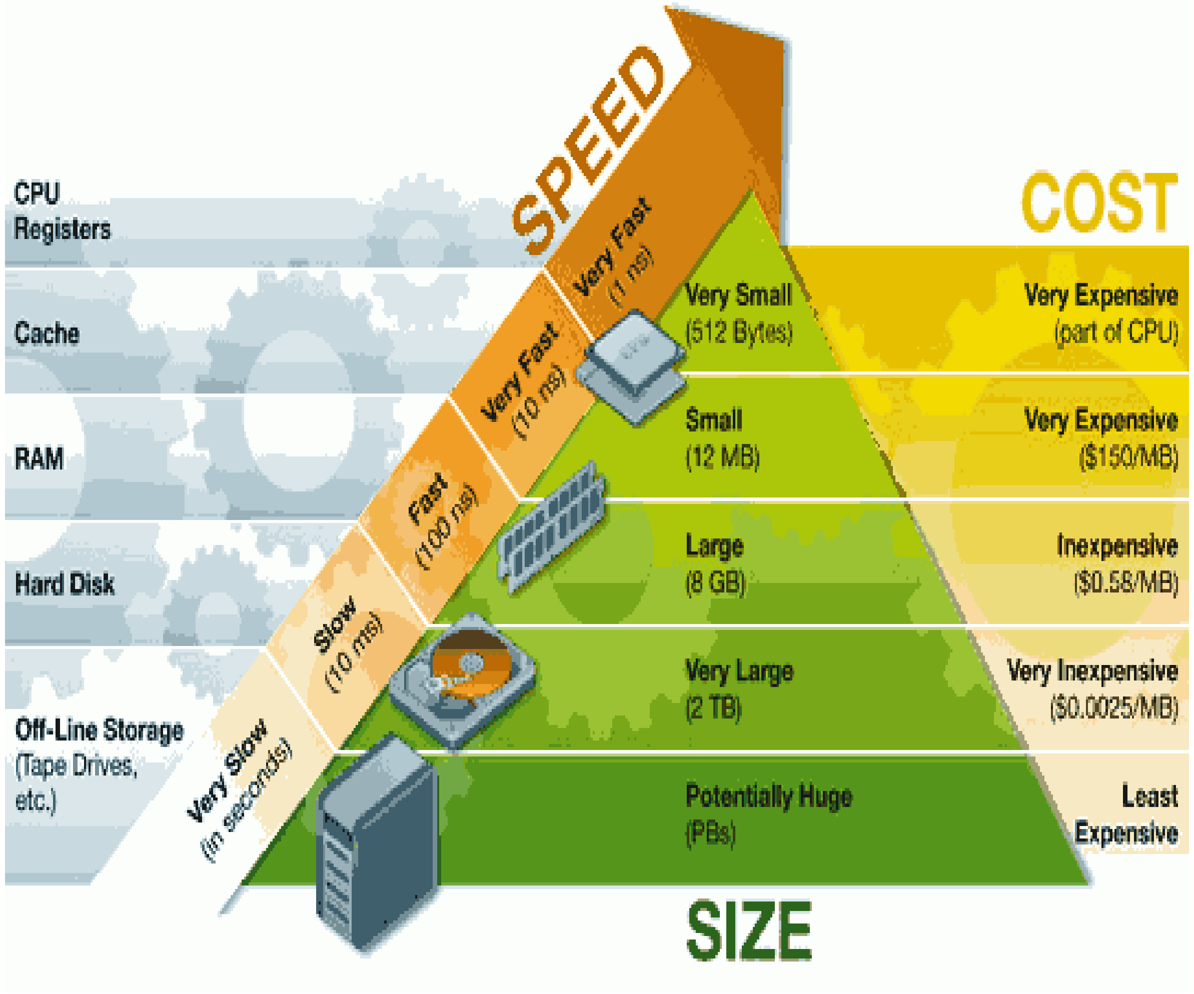


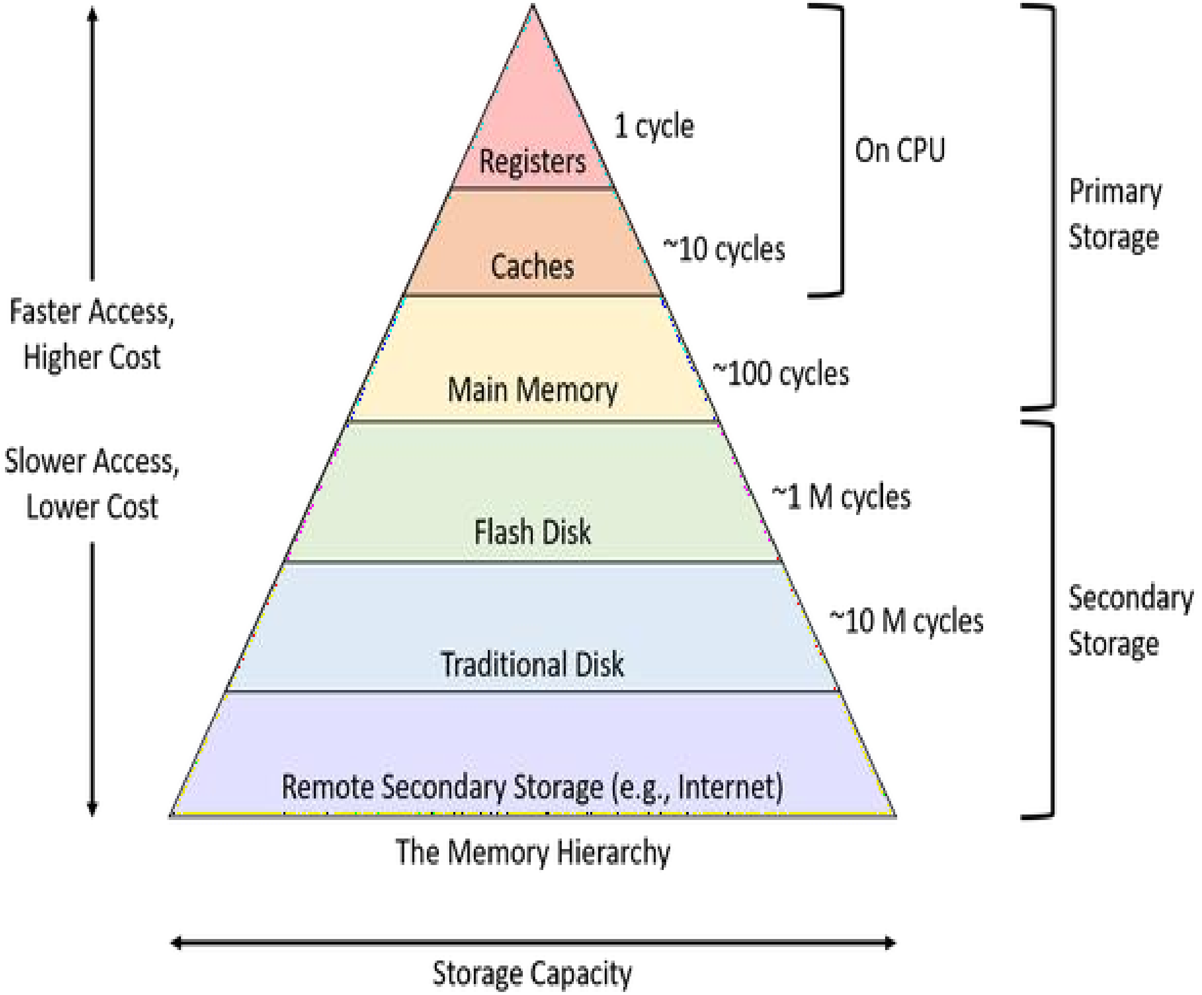
Paging Example



n=2 and m=4   32-byte memory and 4-byte pages

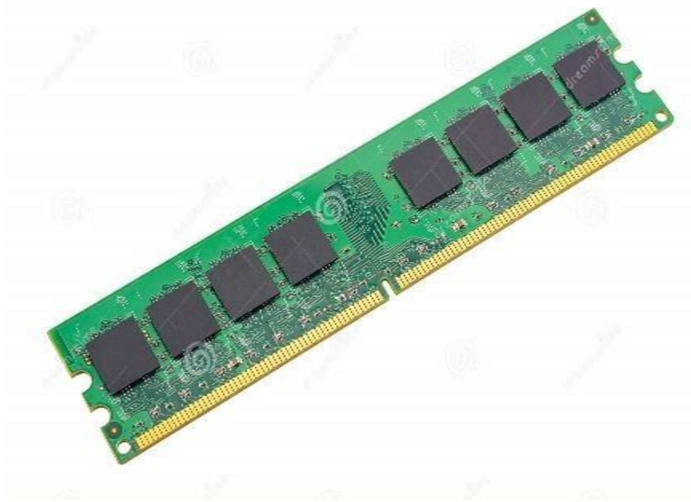
Storage-Device (memory) Hierarchy





• RAM

- Is the main memory in a computer consists of arranged cells.
- Array of memory words
- Each byte has an address
- Memory Address:
  - Physical
  - Logical
- CPU Instructions:
  - Load: moves a word from main memory to CPU register
  - Store: move a word from CPU register to main memory



• Memory Unit

- The memory unit consists of **RAM** (Random Access Memory) and **ROM** (Read Only Memory), sometimes referred to as primary or main memory .
- Unlike a hard drive (secondary memory), this memory is fast and also directly accessible by the CPU.
- RAM is split into partitions (bytes).
- Each partition consists of an **address** and its **contents** .
- The address will uniquely identify every location (byte) in the memory.
- Loading data from permanent memory (secondary storage or hard drive), into the faster and directly accessible temporary memory (RAM), allows the CPU to operate much quicker.

• Need of Memory Management

- **In systems with multiple programs running in parallel**, there could be **many processes in memory at the same time**, and **each process may have specific memory needs**.

- **The OS (kernel component) may also need to be loaded in memory**. Additionally, there may be a specific portion of memory needed for specific devices (Ex: printer spooling).

• Spooling in Operating System

- the CPU executes the instructions and deal with many processes, and we know that the time taken in the I/O operation is very large compared to the time taken by the CPU for the execution of the instructions.

- SPOOLING: is used for the purpose of copying data between different devices. In modern systems it is usually used for mediating between a computer application and a slow peripheral, such as a printer.

• Printer spooling

- the CPU executes the instructions and deal with many processes, and we know that the time taken in the I/O operation is very large compared to the time taken by the CPU for the execution of the instructions.
- SPOOLING: is used for the purpose of copying data between different devices. In modern systems it is usually used for mediating between a computer application and a slow peripheral, such as a printer.

- **Spooling refers to** the **time that it takes to move information from a program or document to your printer**, which must have been turned on and connected

- The spooling process usually happens in the background, and can sometimes **take several minutes** if you're **printing** an especially **large document** or image.

• Inter-process Communication

A process can be of two types:

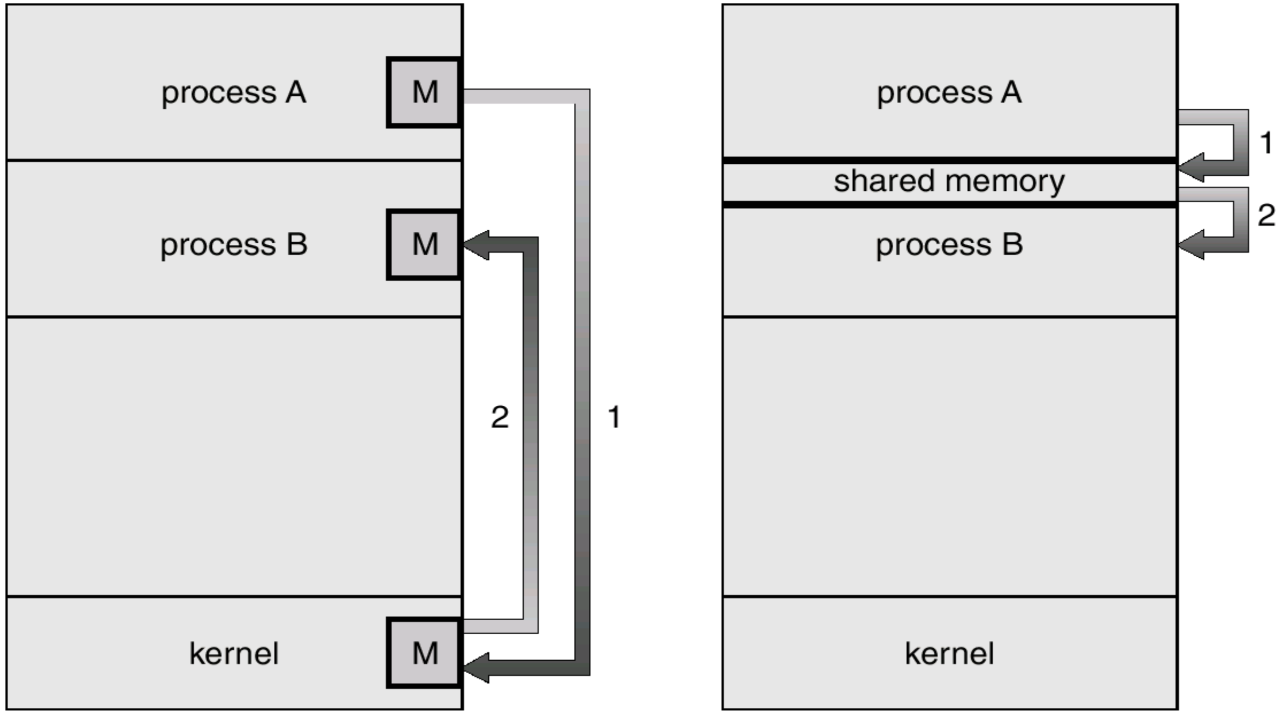
1. An independent process is not affected by the execution of other processes
2. co-operating process can be affected by other executing processes.

Inter-process communication (IPC) is a mechanism that allows processes to communicate with each other and synchronize their actions.

• Inter-process Communication

- The communication between these processes can be seen as a method of co-operation between them. Processes can communicate with each other through both:

- Shared Memory
- Message passing



• Shared Memory Method

- When two or more processes need to communicate with each other**, they may create a shared memory** area that is accessible by both processes.
- Then, one of the processes may act as the ***producer*** *of data*, while the other could act as the ***consumer*** *of data*.
- The ***memory acts as the communication buffer*** between these two processes.

Note: this method need a way of management when the two processes need to save in the shared memory at the same time, it is called ***Synchronization***

- **Race condition** solve is mutex and semaphores .

• Message Passing Method

- In this method, processes communicate with each other without using any kind of shared memory. they proceed as follows:
- **For examples** – chat programs, ***TCP/IP communication,*** print server

- Establish a communication link (if a link already exists, no need to establish it again.)
- Start exchanging messages using basic primitives.
  We need at least two primitives:
  – **send**(message, destination) or **send**(message)
  – **receive**(message, host) or **receive**(message)

• Message Passing Method

Message Passing through Communication Link.

In Direct message passing:

The process which wants to communicate must explicitly name the recipient or sender of the communication.

e.g. **send(p1, message)** means send the message to p1.

Similarly, **receive(p2, message)** means to receive the message from p2.

In indirect Communication:

Is done via a shared mailbox (port), which consists of a queue of messages. The sender keeps the message in mailbox and the receiver picks them up.

The standard primitives used are: **send(A, message)** which means send the message to mailbox A. The primitive for the receiving the message also works in the same way e.g. **received (A, message)**.

• Message Passing Method (chat program)

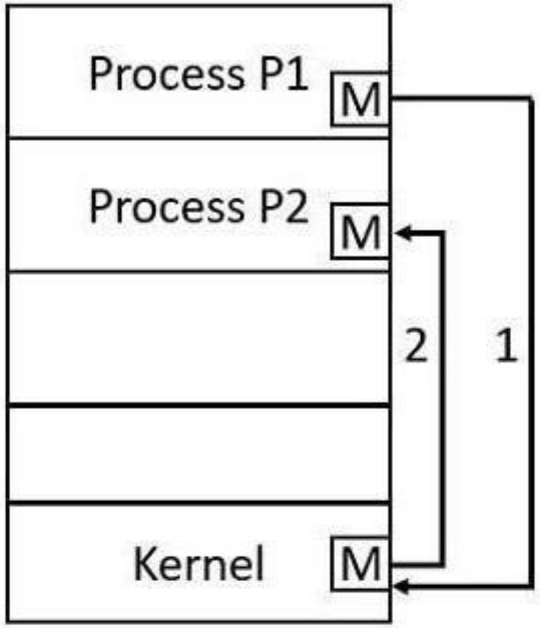
- **Step 1** – Message passing provides two operations which are as follows –
- Send message
- Receive message
- Messages sent by a process can be either fixed or variable size.

- **Step 2** – For fixed size messages the system level implementation is straight forward. It makes the task of programming more difficult.

- **Step 3** – The variable sized messages require a more system level implementation but the programming task becomes simpler.

- **Step 4** – If process P1 and P2 want to communicate they need to send a message to and receive a message from each other that means here a communication link exists between them.

- **Step 5** – Methods for logically implementing a link and the send() and receive() operations.



Message Passing System

Thank You

With My Best Wishes

Zeyad ashraf