II EVOLUTION OF OPERATING SYSTEM

1. EVOLUTION

CPU : The hardware that executes instructions. Processor

: A physical chip that contains one or more CPUs.

Core : The basic computation unit of the CPU.

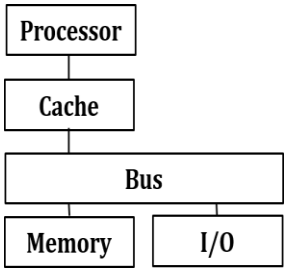
Multicore : Including multiple computing cores on the same CPU.

Multiprocessor : Including multiple processors.

According to the number of general-purpose processors used it is divided into

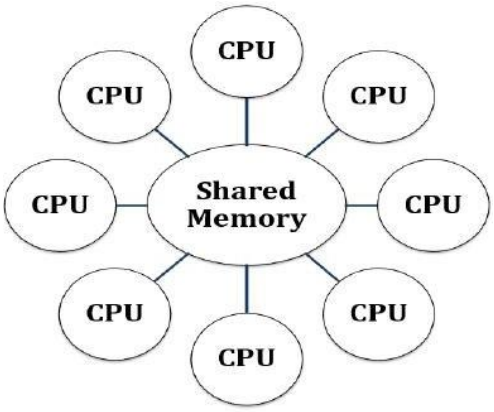
- Single Processor Systems
- Multiprocessor Systems
- Clustered Systems

❖ **Single Processor Systems**



On a single-processor system, there is one main CPU capable of executing the instructions. Most computer systems use a single processor containing one CPU with a single processing core. The **core** is the component that executes instructions and registers for storing data locally. The one main CPU with its core is capable of executing a general-purpose instruction set, including instructions from processes.

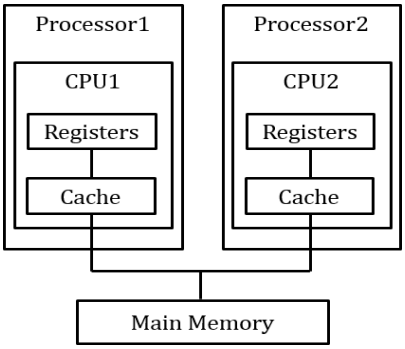
❖ **Multiprocessor Systems**



They have more processors, each with a single-core CPU. The processors share the computer bus and sometimes the clock, memory, and peripheral devices. The main advantages are

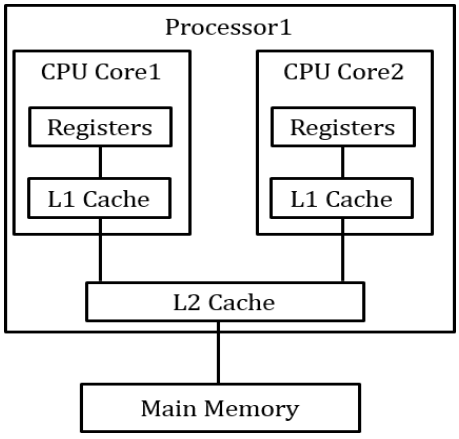
- Increased throughput
- Economy of scale
- Increased reliability – graceful degradation or fault tolerance

The advantage of multiprocessor systems is increased throughput. By increasing the number of processors, more work will be done in less time. The most common multiprocessor systems use **symmetric multiprocessing** (**SMP**), in which each peer CPU processor performs all tasks, including



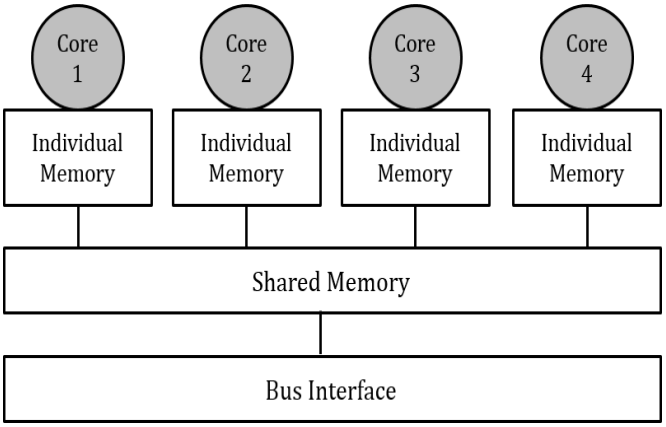
operating-system functions and user processes.

The Figure illustrates a typical SMP architecture with two processors, each with its own CPU. Each CPU processor has its own set of registers, and local cache. Main Memory is shared. The benefit is that many processes can run simultaneously *N* processes can run if there are *N* CPUs without causing performance to deteriorate significantly. However, since the CPUs are separate, one may be sitting idle while another is overloaded, resulting in inefficiencies. These inefficiencies can be avoided if the processors share certain data structures. The definition of **multiprocessor** has evolved over time and now includes



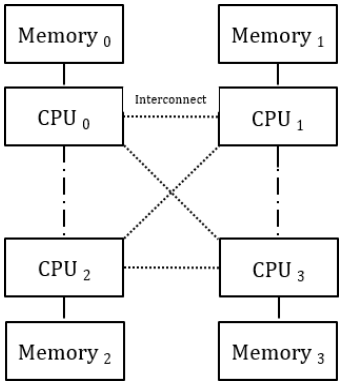
multicore systems, in which multiple computing cores reside on a single chip. Multicore systems can be more efficient than multiple chips with single cores.

In a dual-core design with two cores on the same processor chip. In this design, each core has its own register set, as well as its own local cache, often known as a level 1, or L1, cache. Notice, too, that a level 2 (L2) cache is local to the chip but is shared by the two processing cores.



NUMA Multiprocessing Architecture

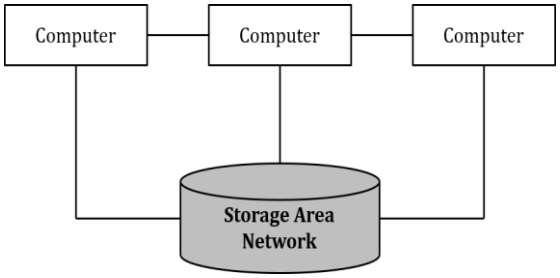
Adding additional CPUs to a multiprocessor system will increase computing power; and adding too many CPUs, becomes a bottleneck and performance begins to degrade. Instead to provide each CPU (or group of CPUs) with its own local memory that is accessed via a small, fast local bus. The CPUs are connected by a **shared system interconnect**, so that all CPUs share one physical address space. This approach known as **Non-Uniform Memory Access**, or **NUMA**



The advantage is that, when a CPU accesses its local memory, it is fast, and no contention over the system interconnect. Thus, NUMA systems can scale more effectively as more processors are added. A drawback is **increased latency**. For example, CPU0 cannot access the local memory of CPU3 as quickly as it can access its own local memory, slowing down performance.

Because NUMA systems can scale to accommodate a large number of processors, they are becoming increasingly popular on servers as well as high-performance computing systems.

❖ **Clustered Systems**

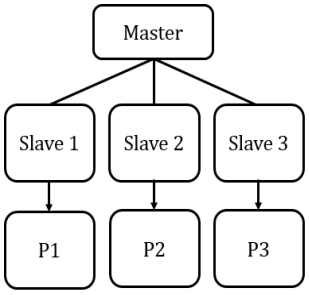


Clustered computers share storage and are closely linked via a local-area network LAN or a faster interconnect. Clustering is usually used to provide **high availability service** that is, service that will continue even if one or more systems in the cluster fail.

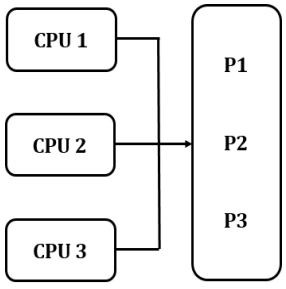
A layer of cluster software runs on the cluster nodes. Each node can monitor one or more of the others (over the network). If the monitored machine fails, the monitoring machine can take ownership of its storage and restart the applications that were running on the failed machine. High availability provides increased reliability, which is crucial in many applications. The ability to continue providing service proportional to the level of surviving hardware is called **graceful degradation**. Some systems go beyond graceful degradation and are called **fault tolerant**, because they can suffer a failure of any single component and still continue operation. Fault tolerance requires a mechanism to allow the failure to be detected, diagnosed, and, if possible, corrected.

Clustering can be structured asymmetrically or symmetrically.

In **Asymmetric Clustering**, one machine is in hot standby mode while the other is running the applications. The hot standby host machine does nothing but monitor the active server. If that server fails, the hot standby host becomes the active server.



Asymmetric Clustering



Symmetric Clustering

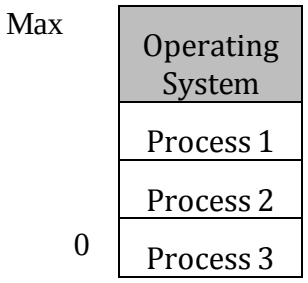
In **symmetric clustering**, two or more hosts are running applications and are monitoring each other. This structure is obviously more efficient, as it uses all of the available hardware.

2. OPERATING SYSTEM OPERATIONS

- Multiprogramming and Multitasking
- Dual-Mode and Multimode Operation
- Timer

❖ Multiprogramming

Multiprogramming increases CPU utilization, so that the CPU always has one to execute. In a multiprogrammed system, a program in execution is termed as **process**.



Memory Layout of a Multiprogramming System

The operating system picks and begins to execute one of these processes. Eventually, the process may have to wait for some task, such as an I/O operation, to complete. In a non- multiprogrammed System, the CPU would sit idle. In a multiprogrammed system, the operating system simply switches to, and executes, another process. When **that** process needs to wait, the CPU switches to **another** process, and so on. Eventually, the first process finishes waiting and gets the CPU back. As long as at least one process needs to execute, the CPU is never idle.

❖ Multitasking

CPU switches jobs so frequently that users can interact with each job while it is running, creating **interactive** computing

- **Response time** should be < 1 second
- Each user has at least one program executing in memory
- If several jobs ready to run at the same time [**CPU scheduling]**
- If processes don't fit in memory, **swapping** moves them in and out to run
- **Virtual memory** allows execution of processes not completely in memory

❖ **Dual-Mode and Multimode Operation**

The following are the modes

- **User Mode:**

Operating system running a user application such as handling a text editor. The transition from user mode to kernel mode occurs when the application requests the help of operating system or an interrupt or a system call occurs. The mode bit is set to 1 in the user mode. It is changed from 1 to 0 when switching from user mode to kernel mode.

- **Kernel Mode**

The system starts in kernel mode when it boots and after the operating system is loaded, it executes applications in user mode. There are some privileged instructions that can only be executed in kernel mode. These are interrupt instructions, input output management etc. If the privileged instructions are executed in user mode, it is illegal and a trap is generated. The mode bit is set to 0 in the kernel mode. It is changed from 0 to 1 when switching from kernel mode to user mode.

The concept of modes of operation in operating system can be extended beyond the dual mode. Known as the **multimode** system. In those cases more than 1 bit is used by the CPU to set and handle the mode.

❖ **Timer**

A user program cannot get stuck in an infinite loop or to fail to call system services and never return control to the operating system. A timer is used to accomplish this goal. A timer can be set to interrupt the computer after a specified period. A **variable timer** is generally implemented by a fixed-rate clock and a counter. The operating system sets the counter. Every time the clock ticks, the counter is decremented. When the counter reaches 0, an interrupt occurs.