Unit-1 Introduction of Single-Processor Computing and Parallel Computing

MODULE

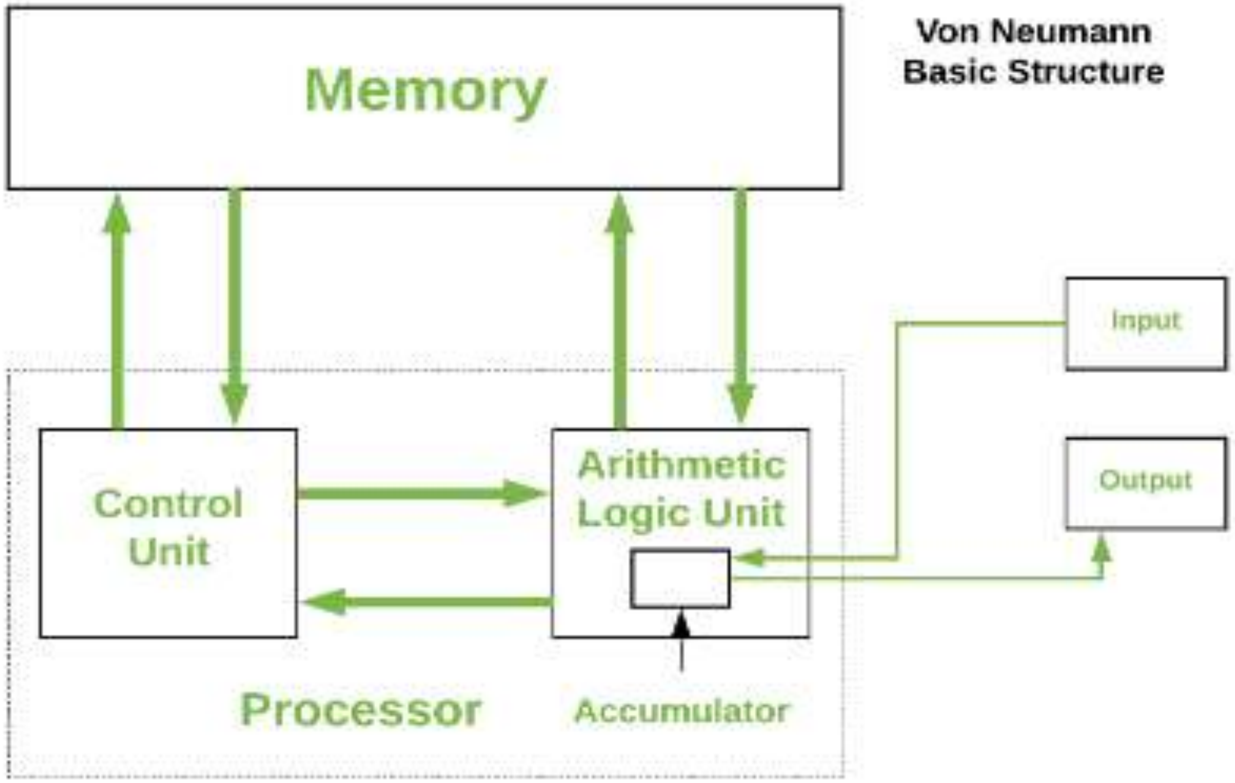
- Single-Processor Computing
- Parallel Computing

The Von Neumann architecture

- There have been 2 types of Computers:
- Fixed Program Computers – Their function is very specific and they couldn’t be reprogrammed, e.g. Calculators.
- Stored Program Computers – These can be programmed to carry out many different tasks, applications are stored on them, hence the name.

- Modern computers are based on a stored-program concept introduced by John Von Neumann.
- In this stored-program concept, programs and data are stored in a separate storage unit called memories and are treated the same.
- This novel idea meant that a computer built with this architecture would be much easier to reprogram.

Structure of Von Neumann architecture



- It is also known as **ISA** (Instruction set architecture) computer and is having three basic units:
  - The Central Processing Unit (CPU)
  - The Main Memory Unit
  - The Input/output Device
- Let’s consider them in detail.

- **Control Unit –**
  A control unit (CU) handles all processor control signals. It directs all input and output flow, fetches code for instructions, and controls how data moves around the system.
- **Arithmetic and Logic Unit (ALU) –**
  The arithmetic logic unit is that part of the CPU that handles all the calculations the CPU may need, e.g. Addition, Subtraction, Comparisons. It performs Logical Operations, Bit Shifting Operations, and Arithmetic operations.

- Registers – Registers refer to high-speed storage areas in the CPU. The data processed by the CPU are fetched from the registers.
- There are different types of registers used in architecture :-
  - Accumulator: Stores the results of calculations made by ALU. It holds the intermediate of arithmetic and logical operations. it act as a temporary storage location or device.
  - Program Counter (PC): Keeps track of the memory location of the next instructions to be dealt with. The PC then passes this next address to the Memory Address Register (MAR).

- Memory Address Register (MAR): It stores the memory locations of instructions that need to be fetched from memory or stored in memory.
- Memory Data Register (MDR): It stores instructions fetched from memory or any data that is to be transferred to, and stored in, memory.
- Current Instruction Register (CIR): It stores the most recently fetched instructions while it is waiting to be coded and executed.
- Instruction Buffer Register (IBR): The instruction that is not to be executed immediately is placed in the instruction buffer register IBR.

- **Buses –** Data is transmitted from one part of a computer to another, connecting all major internal components to the CPU and memory, by the means of Buses.
- Types:
  - **Data Bus:** It carries data among the memory unit, the I/O devices, and the processor.
  - **Address Bus:** It carries the address of data (not the actual data) between memory and processor.
  - **Control Bus:** It carries control commands from the CPU (and status signals from other devices) in order to control and coordinate all the activities within the computer.

- Input/output Devices – Program or data is read into main memory from the input device or secondary storage under the control of CPU input instruction.
- Output devices are used to output information from a computer.
- If some results are evaluated by the computer and it is stored in the computer, then with the help of output devices, we can present them to the user.

Memory Hierarchies

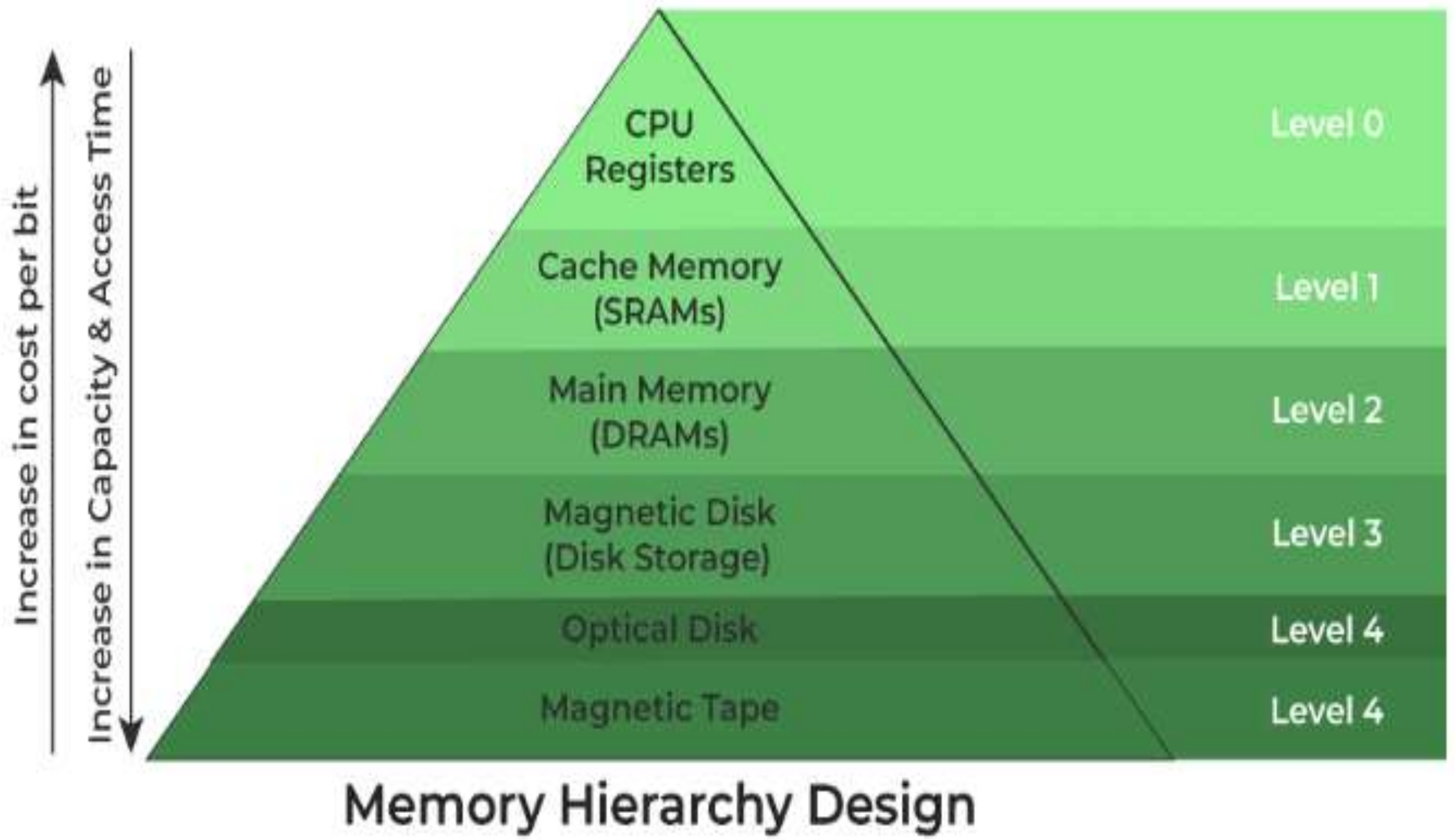
- In the Computer System Design, Memory Hierarchy is an enhancement to organize the memory such that it can minimize the access time.
- The Memory Hierarchy was developed based on a program behavior known as locality of references. The figure below clearly demonstrates the different levels of the memory hierarchy.

Why Memory Hierarchy is Required in the System?

- Memory Hierarchy is one of the most required things in Computer Memory as it helps in optimizing the memory available in the computer.
- There are multiple levels present in the memory, each one having a different size, different cost, etc.
- Some types of memory like cache, and main memory are faster as compared to other types of memory but they are having a little less size and are also costly whereas some memory has a little higher storage value, but they are a little slower.
- Accessing of data is not similar in all types of memory, some have faster access whereas some have slower access.

Types of Memory Hierarchy

- This Memory Hierarchy Design is divided into 2 main types:
- **External Memory or Secondary Memory:**
  Comprising of Magnetic Disk, Optical Disk, and Magnetic Tape i.e. peripheral storage devices which are accessible by the processor via an I/O Module.
- **Internal Memory or Primary Memory:**
  Comprising of Main Memory, Cache Memory & CPU Registers. This is directly accessible by the processor.



Memory Hierarchy Design

Memory Hierarchy Design

- **1. Registers**

  Registers are small, high-speed memory units located in the CPU. They are used to store the most frequently used data and instructions. Registers have the fastest access time and the smallest storage capacity, typically ranging from 16 to 64 bits.

- **2. Cache Memory**

  Cache memory is a small, fast memory unit located close to the CPU. It stores frequently used data and instructions that have been recently accessed from the main memory. Cache memory is designed to minimize the time it takes to access data by providing the CPU with quick access to frequently used data.

- **3. Main Memory**
  Main Memory, also known as RAM (Random Access Memory), is the primary memory of a computer system. It has a larger storage capacity than cache memory, but it is slower. Main memory is used to store data and instructions that are currently in use by the CPU.
- **Types of Main Memory**
  - **Static RAM:** Static RAM stores the binary information in flip flops and information remains valid until power is supplied. It has a faster access time and is used in implementing cache memory.
  - **Dynamic RAM:** It stores the binary information as a charge on the capacitor. It requires refreshing circuitry to maintain the charge on the capacitors after a few milliseconds. It contains more memory cells per unit area as compared to SRAM.

- **4. Secondary Storage**
  - Secondary storage, such as hard disk drives(HDD) and solid-state drives(SSD), is a non-volatile memory unit that has a larger storage capacity than main memory. It is used to store data and instructions that are not currently in use by the CPU. Secondary storage has the slowest access time and is typically the least expensive type of memory in the memory hierarchy.

- **5. Magnetic Disk**
  - Magnetic Disks are simply circular plates that are fabricated with either a metal or a plastic or a magnetized material. The Magnetic disks work at a high speed inside the computer and these are frequently used.
- **6. Magnetic Tape**
  - Magnetic Tape is simply a magnetic recording device that is covered with a plastic film. It is generally used for the backup of data. In the case of a magnetic tape, the access time for a computer is a little slower and therefore, it requires some amount of time for accessing the strip.

Characteristics of Memory Hierarchy

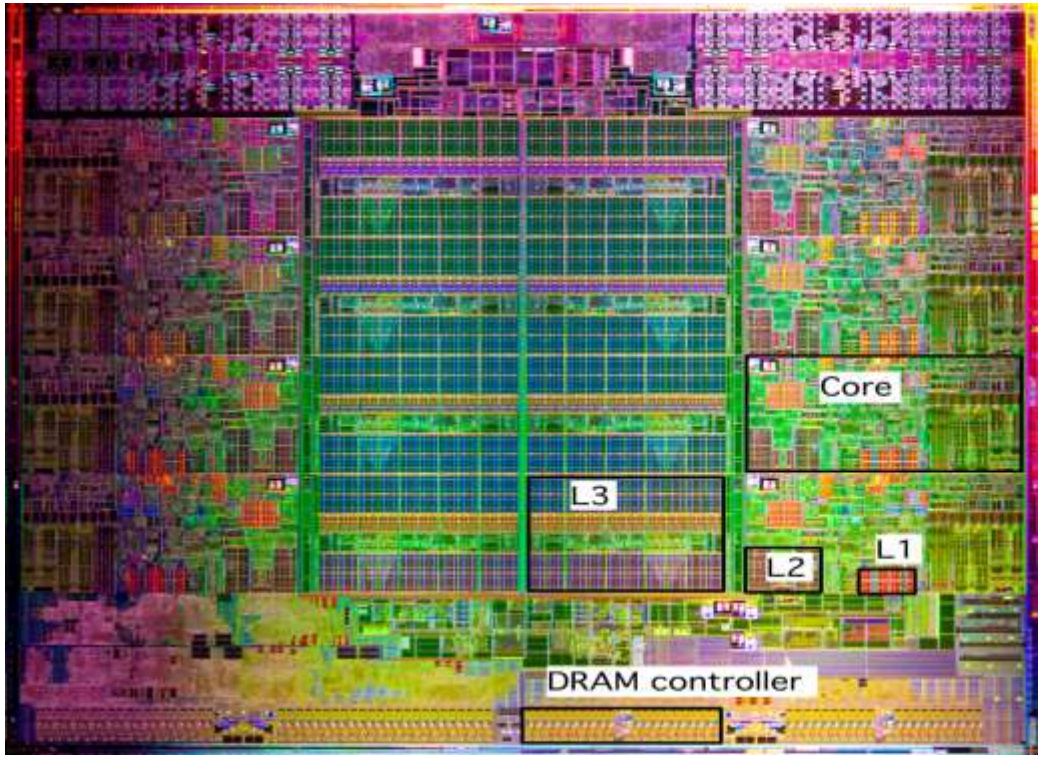
- **Capacity:** It is the global volume of information the memory can store. As we move from top to bottom in the Hierarchy, the capacity increases.
- **Access Time:** It is the time interval between the read/write request and the availability of the data. As we move from top to bottom in the Hierarchy, the access time increases.
- **Performance:** Earlier when the computer system was designed without a Memory Hierarchy design, the speed gap increased between the CPU registers and Main Memory due to a large difference in access time. This results in lower performance of the system and thus, enhancement was required. This enhancement was made in the form of Memory Hierarchy Design because of which the performance of the system increases. One of the most significant ways to increase system performance is minimizing how far down the memory hierarchy one has to go to manipulate data.
- **Cost Per Bit:** As we move from bottom to top in the Hierarchy, the cost per bit increases i.e. Internal Memory is costlier than External Memory.

Advantages of Memory Hierarchy

- It helps in removing some destruction, and managing the memory in a better way.
- It helps in spreading the data all over the computer system.
- It saves the consumer’s price and time.

Modern processors

- Modern processors are quite complicated, and in this section we will give a short tour of what their constituent parts.



- In the Von Neuman model there is a single entity that executes instructions.
- The Sandy Bridge pictured in figure has eight cores, each of which is an independent unit executing a stream of instructions.
- In this chapter we will mostly discuss aspects of a single core; in other section will discuss the integration aspects of the multiple cores.
- The Von Neumann model is also unrealistic in that it assumes that all instructions are executed strictly in sequence.

- Increasingly, over the last twenty years, processor have used out of- order instruction handling, where instructions can be processed in a different order than the user program specifies. Of course the processor is only allowed to re-order instructions if that leaves the result of the execution intact!
- In the block diagram (figure) you see various units that are concerned with instruction handling:
- This cleverness actually costs considerable energy, as well as sheer amount of transistors. For this reason, processors such as the Intel Xeon Phi use in-order instruction handling.

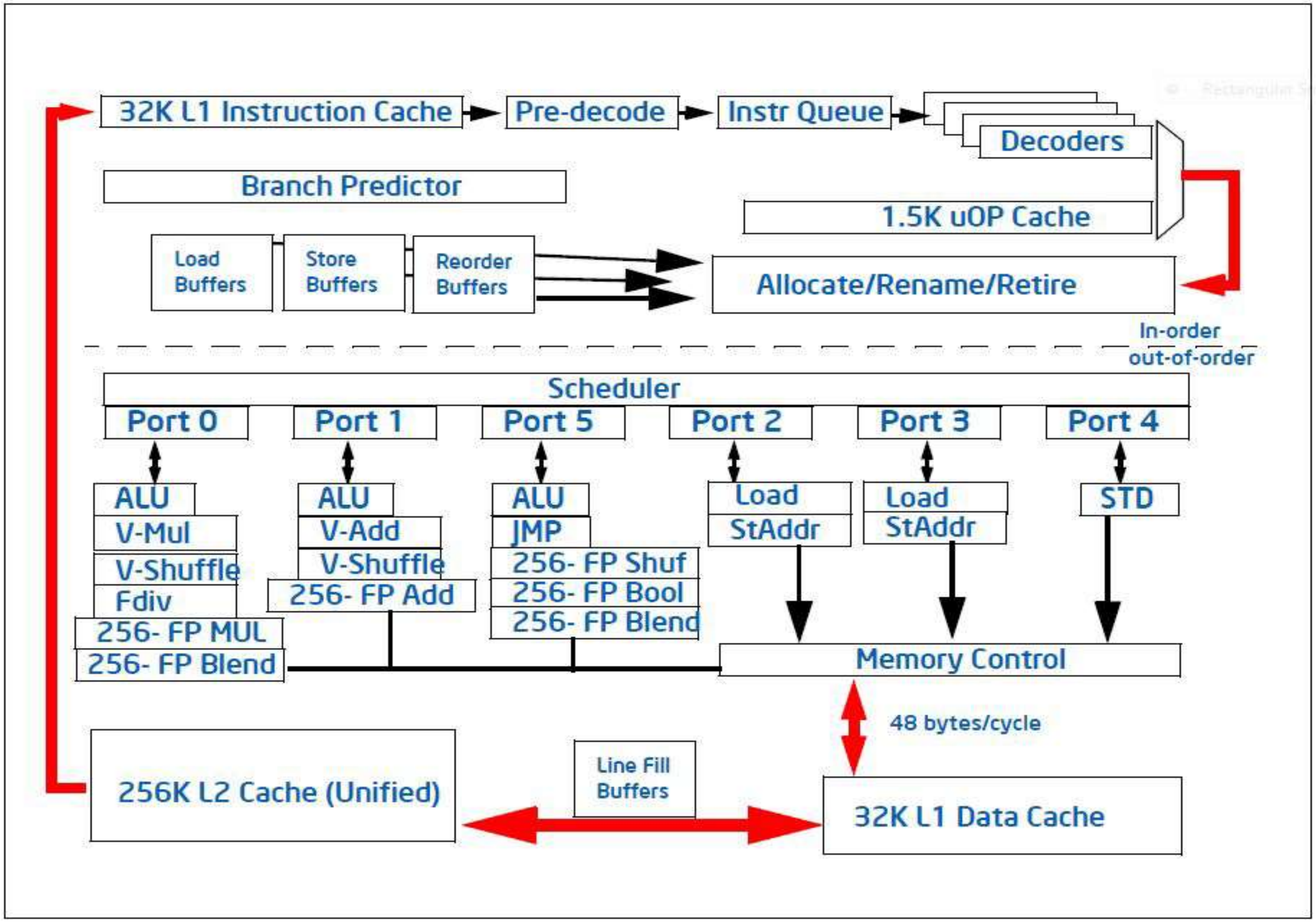


Figure 2-1. Intel microarchitecture code name Sandy Bridge Pipeline Functionality

Overview of Sandy Bridge:

- Sandy Bridge is Intel’s 32 nm micro architecture used in the second generation of Intel Core processors (Core i7, i5, i3).
- It succeeded the Nehalem and Westmere micro architectures.
- Sandy Bridge processors were first released in January 2011 under the Core brand.
- Notably, Sandy Bridge features a soldered contact between the die and the Integrated Heat Spreader (IHS), which differs from the subsequent generation, Ivy Bridge, that uses Thermal Interface Material (TIM)

CPU Core:

- Each Sandy Bridge chip contains 1 to 8 cores.
- L1 cache: 32 KB data + 32 KB instruction per core.
- L2 cache: 256 KB per core.
- Shared L3 cache: Includes the processor graphics (for LGA 1155 sockets).
- 64-byte cache line size.
- New µOP cache: Up to 1536-entry.

CPU Core:

- Improved integer ALU, vector ALU, and AGU per core.
- Two load/store operations per CPU cycle for each memory channel.
- Decoded micro-operation cache and an enlarged, optimized branch predictor.
- Retains four branch predictors found in Nehalem: the branch target buffer (BTB), indirect branch target array, loop detector, and renamed return stack buffer (RSB).
- Improved performance for transcendental mathematics, AES encryption, and SHA-1 hashing

- Graphics Unit (GPU):
  - Sandy Bridge integrates an on-die GPU (Intel HD Graphics).
  - The GPU operates at frequencies ranging from 650 MHz to 1350 MHz.
  - It shares the L3 cache with the CPU.
  - The GPU provides basic graphics capabilities for desktops and workstations.

- Memory Controller:
  - The memory controller manages communication between the CPU cores and system memory.
  - It ensures efficient data transfer and memory access.

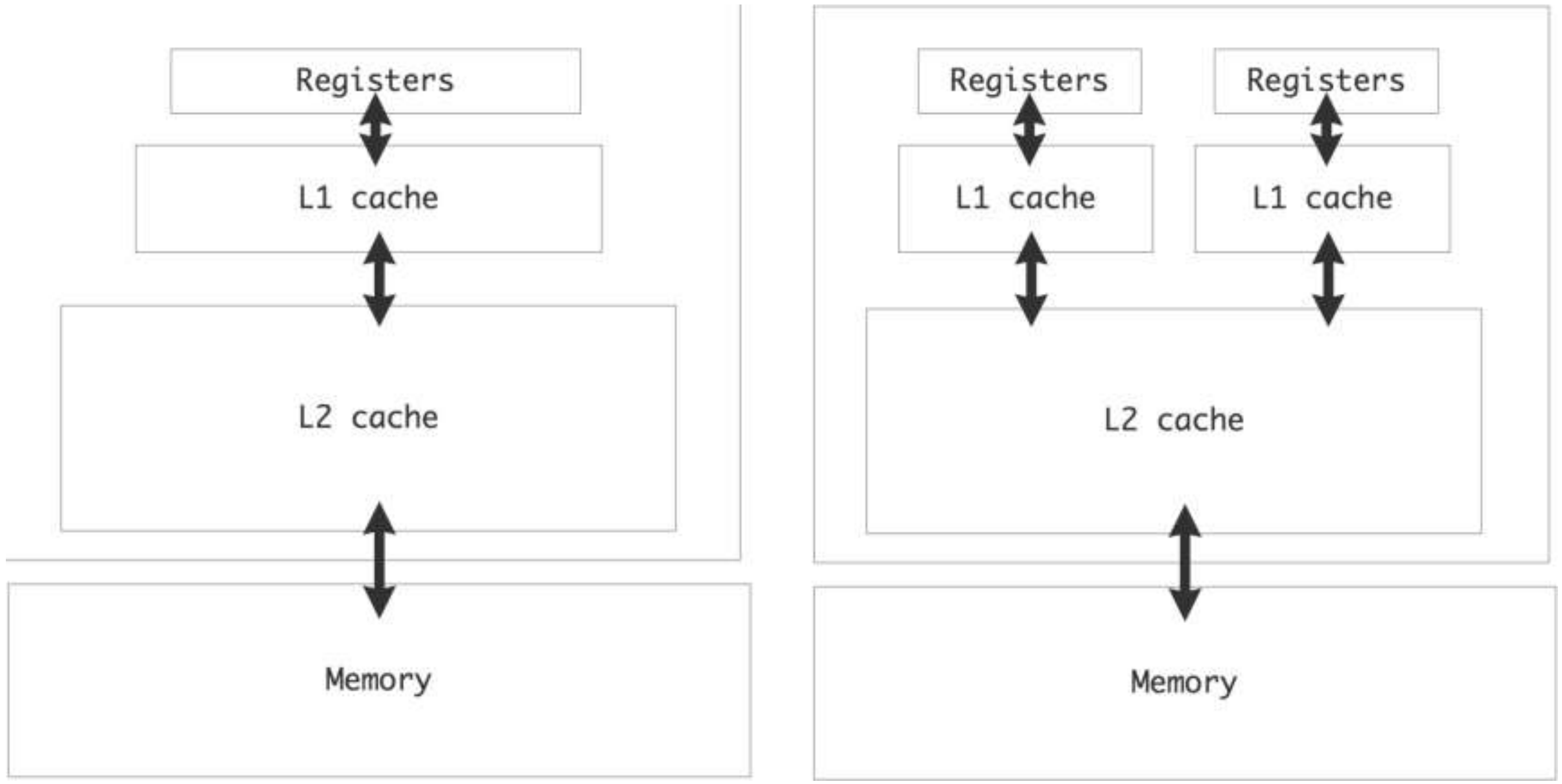
- PCI Express (PCIe) Interface:
  - ➢ Sandy Bridge supports PCIe for connecting peripheral devices such as GPUs, network cards, and storage controllers.

- Display Engine:
  - ➢ The display engine handles video output, including support for multiple displays, resolutions, and refresh rates.

- Power Management Control Blocks and Interconnect:
  - ➢ These components manage power states, clock gating, and dynamic frequency adjustments to optimize power consumption and performance.

Multicore Architecture

- A Multicore processor incorporates two or more separate processing units, called cores, into a single chip.
- Each core can execute instructions independently of the others, allowing the chip to run multiple programs or threads concurrently.
- These cores operate in parallel, enhancing overall system performance.



Cache hierarchy in a single-core and dual-core chip

Why Multicore?

- **Challenges with Single-Core Clock Frequencies**:
  - Increasing clock frequencies for single-core processors becomes difficult due to issues like heat, speed of light, and complex design.
  - Deeply pipelined circuits face heat problems and require expensive cooling solutions.
- **Shift Toward Parallelism**:
  - Many new applications are multithreaded emphasizing parallel execution.
  - Instruction-level parallelism (ILP) enabled rapid speed increases in processors over the last 15 years.
  - However, single-core superscalar processors struggle to fully exploit thread-level parallelism (TLP).
- Multi-core architectures address this challenge by explicitly exploiting TLP2

Characteristics of Multicore Architectures:

- **Multiple Cores on a Single Die**:
  - Cores fit on a single processor socket, also known as Chip Multi-Processor (CMP).
  - Each core runs in parallel, executing different threads.
- **Shared Memory Multiprocessor**:
  - All cores share the same memory.
  - Applications benefit from thread-level parallelism (TLP).
- **Applications Benefiting from Multicore**:
  - Database servers, web servers, compilers, multimedia apps, scientific applications, and more.
  - Examples include editing a photo while recording a TV show or downloading software while running an anti-virus program2.

Thread

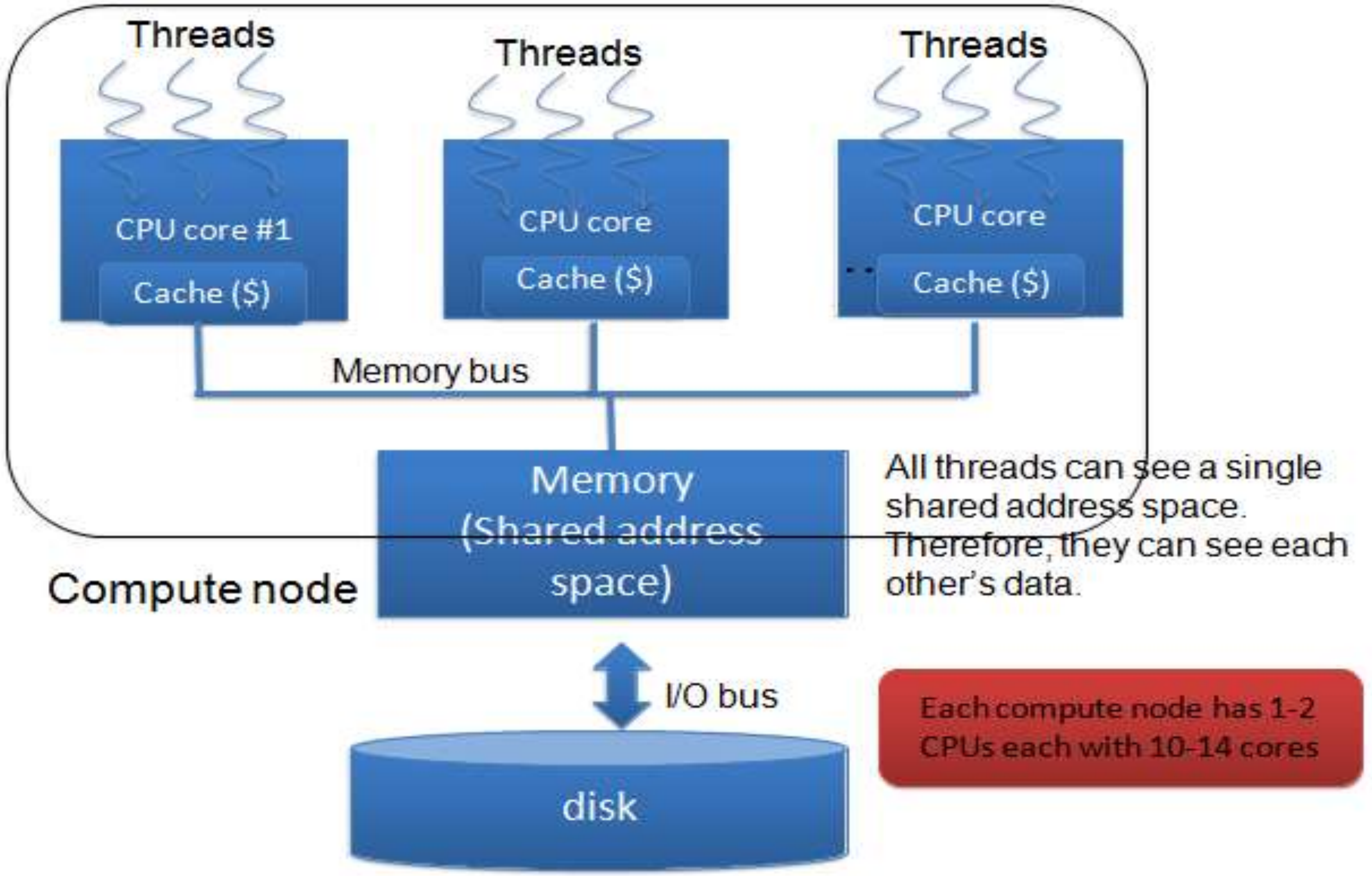
- Thread is a sequential flow of control within a process.
- A process can contain one or more threads.
- Threads have their own program counter and register values, but they share the memory space and other resources of the process.
- Thread is basically a lightweight process .

- Advantages:
  - It takes less time to create and terminate a new thread than to create, and terminate a process.
  - It takes less time to switch between two threads within the same process .
  - Less communication overheads.

Memory

- Two major classes of parallel programming models:
  - Shared Memory
  - Distributed Memory
- **Shared Memory**:
  - One large common shared memory accessible by all processors.
- **Distributed Memory**:
  - Each processor has its own local memory, not replicated elsewhere.
- Multi-core processors are a special kind of multiprocessor, with all cores on the same chip, sharing memory

Shared Memory Architecture

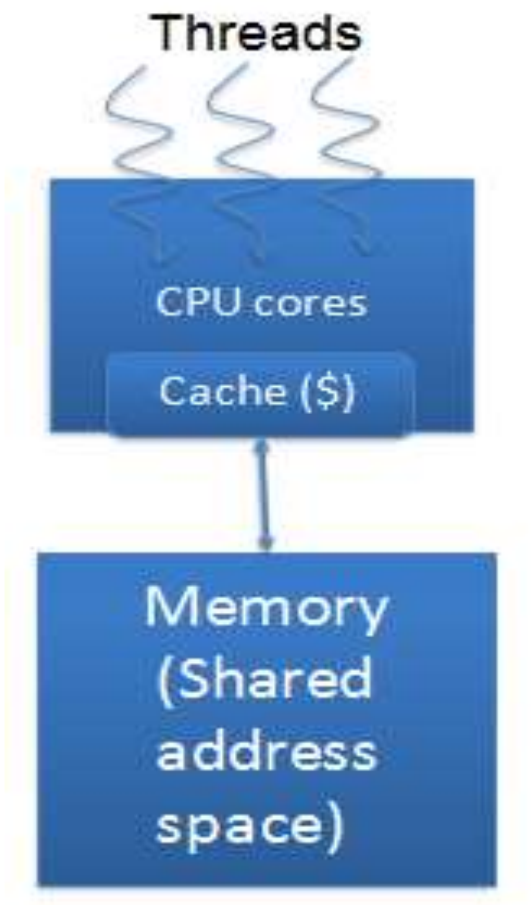


All threads can see a single shared address space. Therefore, they can see each other's data.

Each compute node has 1-2 CPUs each with 10-14 cores

Multi-Threading (for shared memory architectures)

- Threads are contained within processes
  - One process => multiple threads
- All threads of a process share the same address space (in memory).
- Threads have the capability to run concurrently (executing different instructions and accessing different pieces of data at the same time)
- But if the resource is occupied by another thread, they form a queue and wait.
  - For maximum throughput, it is ideal to map each thread to a unique/distinct core



Locality and data reuse

- By now it should be clear that there is more to the execution of an algorithm than counting the operations:
- the data transfer involved is important, and can in fact dominate the cost.
- Since we have caches and registers, the amount of data transfer can be minimized by programming in such a way that data stays as close to the processor as possible.
- Partly this is a matter of programming cleverly, but we can also look at the theoretical question: does the algorithm allow for it to begin with.
- It turns out that in scientific computing data often interacts mostly with data that is close by in some sense, which will lead to data locality.

Programming strategies for high performance

- **Parallelization**:
  - Break down your problem into smaller tasks that can be executed concurrently.
  - Utilize **multithreading**, **multiprocessing**, or **distributed computing** to leverage multiple cores or nodes.
  - Consider using libraries like **OpenMP**, **MPI**, or **CUDA** for parallel programming.

- **Algorithmic Optimization**:
  - Choose efficient algorithms that minimize computational complexity.
  - Optimize data structures (e.g., using hash tables, trees, or graphs) to reduce lookup times.
  - Use **approximation algorithms** when exact solutions are not necessary.
- **Memory Management**:
  - Minimize memory allocations and deallocations.
  - Select **stack memory** over **heap memory** where possible.
  - Optimize cache usage by accessing memory sequentially.

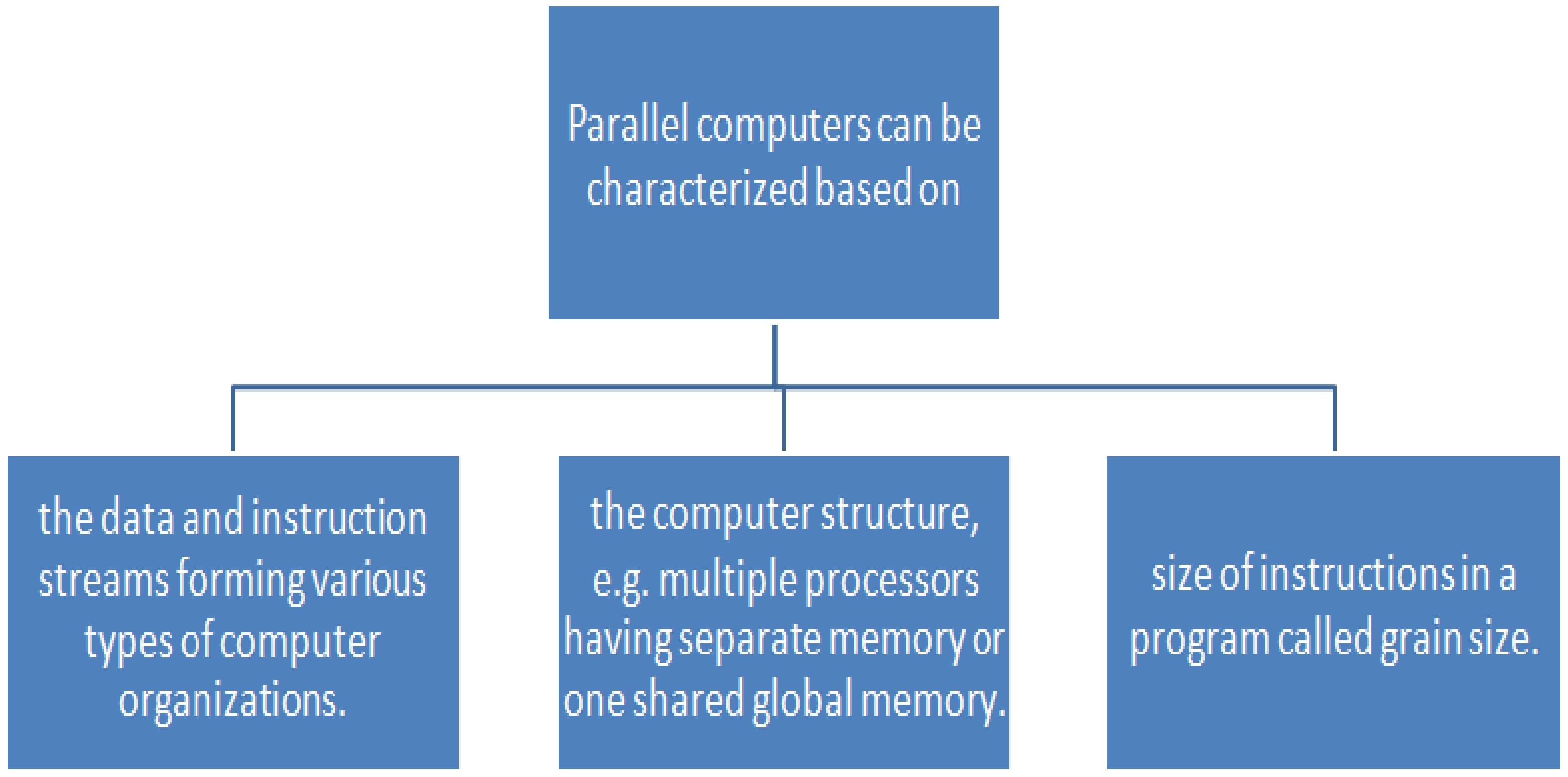
- **Vectorization**:
  - Utilize vector instructions (e.g., **SIMD** or **AVX**) to process multiple data elements simultaneously.
  - Rewrite loops to take advantage of vectorization.
- **I/O Optimization**:
  - Minimize I/O operations and use buffered I/O.
  - Optimize file formats and compression techniques.
  - Use asynchronous I/O when applicable.

- **Cache Optimization**:
  - Arrange data structures to improve cache locality.
  - Be aware of cache line sizes and alignment.
  - Use **prefetching** to reduce cache misses.
- **Memory Hierarchy Awareness**:
  - Understand the memory hierarchy (registers, cache, RAM, disk) and optimize data movement accordingly.
  - Use **NUMA-aware programming** for multicore systems.
- **Load Balancing**:
  - Distribute work evenly across cores or nodes.
  - Avoid situations where some cores are idle while others are overloaded.

- Study of concurrent and parallel executions is important due to following reasons:
  - i) Some problems are most naturally solved by using a set of co-operating processes.
  - ii) To reduce the execution time.
- **Concurrent execution** is the temporal behavior of the N-client 1-server model .
- **Parallel execution** is associated with the N-client N-server model. It allows the servicing of more than one client at the same time as the number of servers is more than one.

- **Granularity** refers to the **amount of computation done** in **parallel relative** to the **size** of the whole program.
- In parallel computing, granularity is a qualitative measure of the ratio of computation to communication.

Characteristics of Parallel computer


Parallel computers can be characterized based on
the data and instruction streams forming various types of computer organizations.
the computer structure, e.g. multiple processors having separate memory or one shared global memory.
size of instructions in a program called grain size.

TYPES OF CLASSIFICATION

- Classification based on the instruction and data streams
- Classification based on the structure of computers
- Classification based on how the memory is accessed
- Classification based on grain size

classification of parallel computers

- Flynn’s classification based on instruction and data streams
- The Structural classification based on different computer organizations;
- The Handler's classification based on three distinct levels of computer:
  - Processor control unit (PCU), Arithmetic logic unit (ALU), Bit-level circuit (BLC)
- describe the sub-tasks or instructions of a program that can be executed in parallel based on the grain size.

FLYNN'S CLASSIFICATION

- It is based on the notion of a stream of information. Two types of information flow into a processor: instruction and data.
- Introduced the concept of *instruction* and *data* streams for categorizing of computers.
- This classification is based on instruction and data streams
- Working of the instruction cycle.

- **Instruction Cycle**
- The instruction cycle consists of a sequence of steps needed for the execution of an instruction in a program

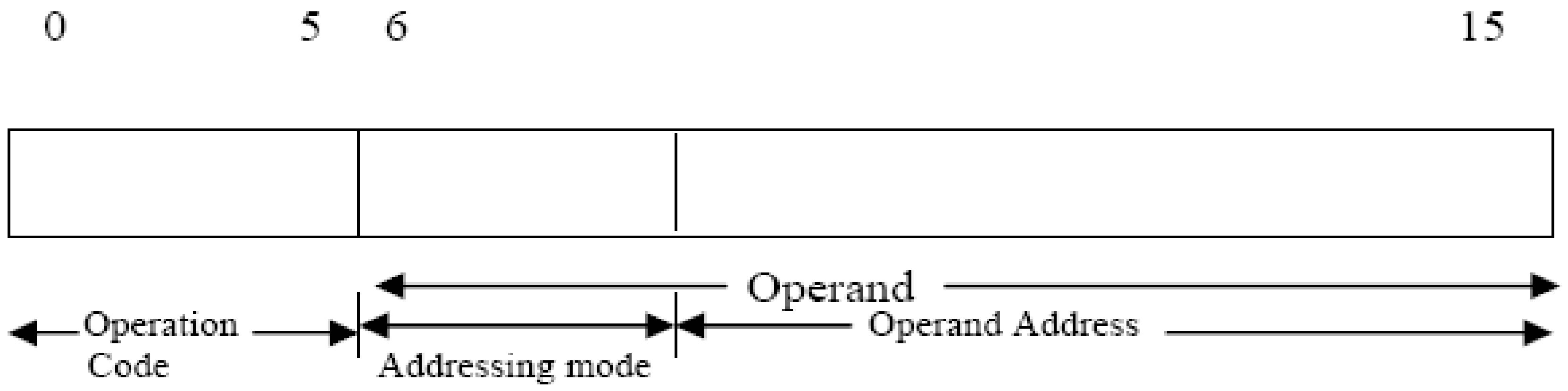


Figure 1: Opcode and Operand

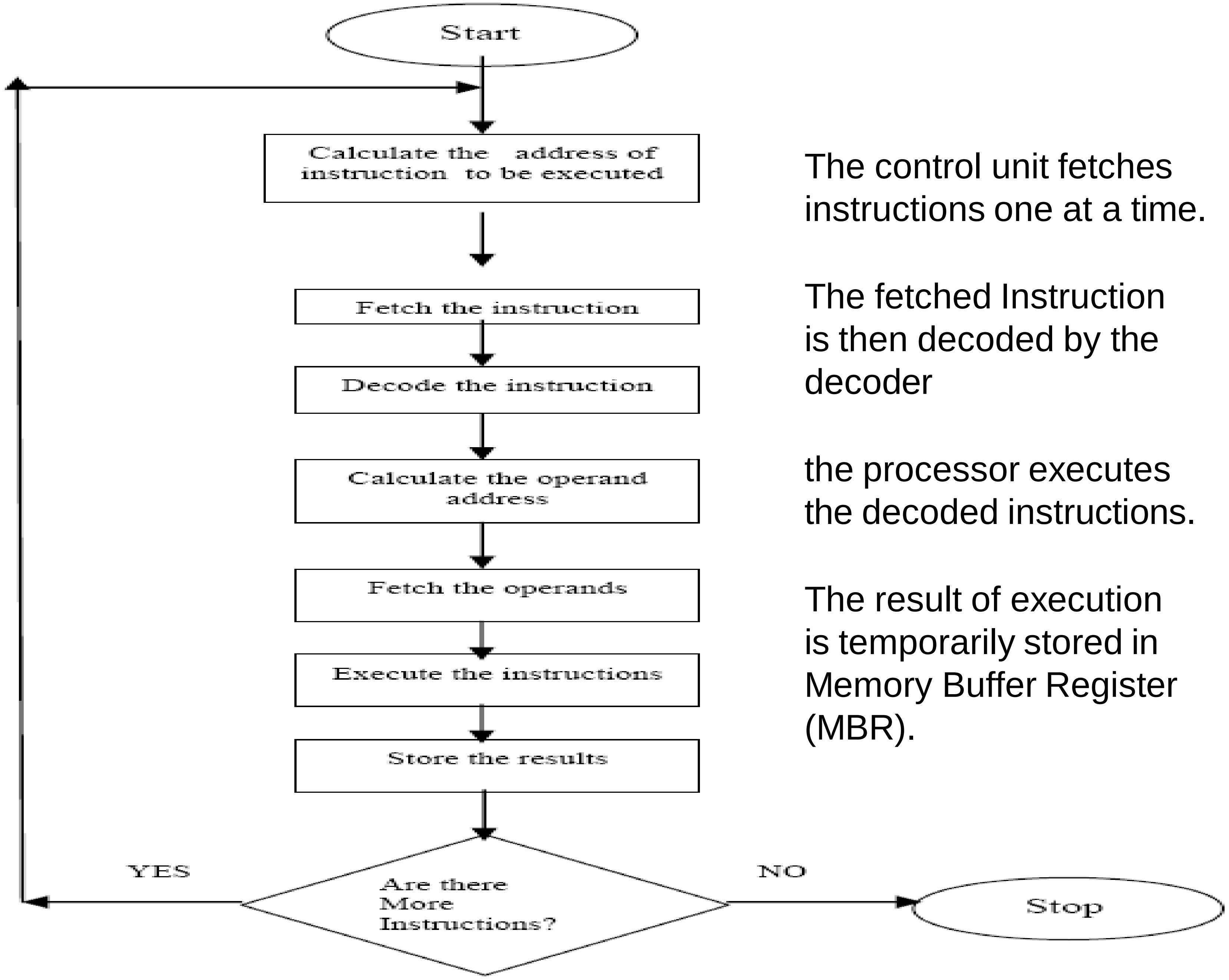


Figure 2: Instruction execution steps

The control unit fetches instructions one at a time.

The fetched Instruction is then decoded by the decoder

the processor executes the decoded instructions.

The result of execution is temporarily stored in Memory Buffer Register (MBR).

Instruction Stream and Data Stream

- flow of instructions is called ***instruction stream.***
- flow of operands between processor and memory is bi- directional. This flow of operands is called ***data stream***.

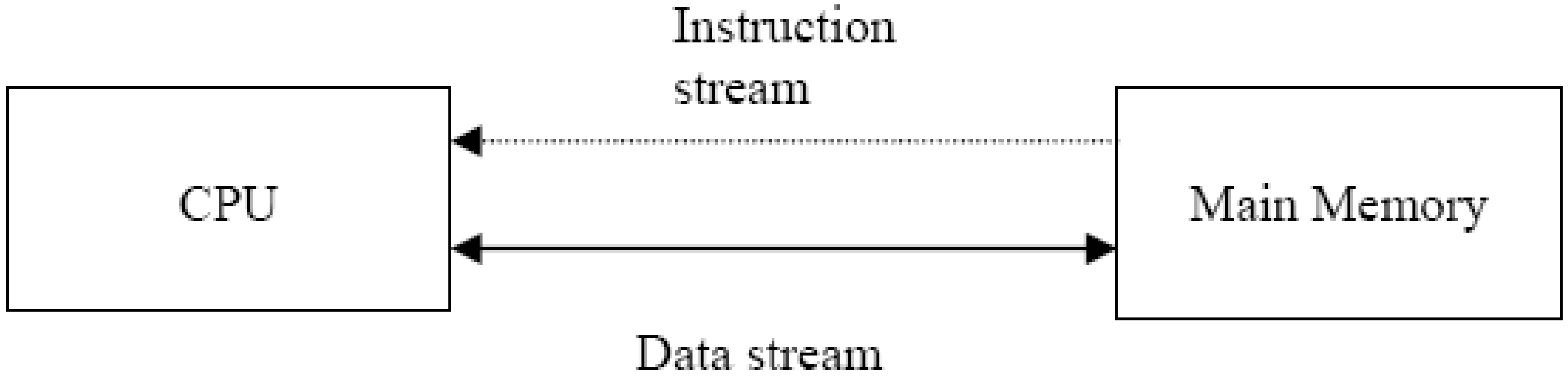


Figure 3: Instruction and data stream

Flynn’s Classification

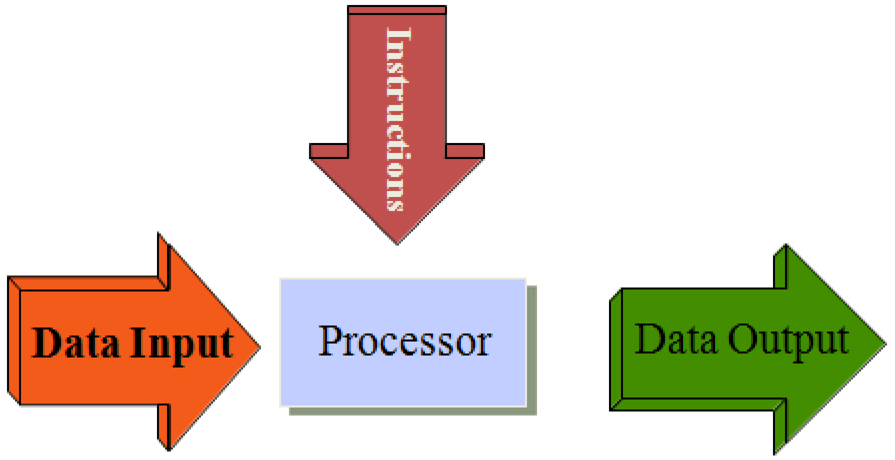
- Based on multiplicity of instruction streams and data streams observed by the CPU during program execution.

1) **Single Instruction and Single Data stream (SISD)**
2) **Single Instruction and multiple Data stream (SIMD)**
3) **Multiple Instruction and Single Data stream (MISD)**
4) **Multiple Instruction and Multiple Data stream (MIMD)**

Single Program Models: SIMD vs. MIMD

- **SP**: Single Program
  - Your parallel program is a single program that you execute on all threads (or processes)
- **SI**: Single Instruction
  - Each thread should be executing the same line of code at any given clock cycle.
- **MI**: Multiple Instruction
  - Each thread (or process) could be independently running a different line of your code (instruction) concurrently
- **MD**: Multiple Data
  - Each thread (or process) could be operating/accessing a different piece of the data from the memory concurrently

SISD : A Conventional Computer



➔Speed is limited by the rate at which computer can transfer information internally.

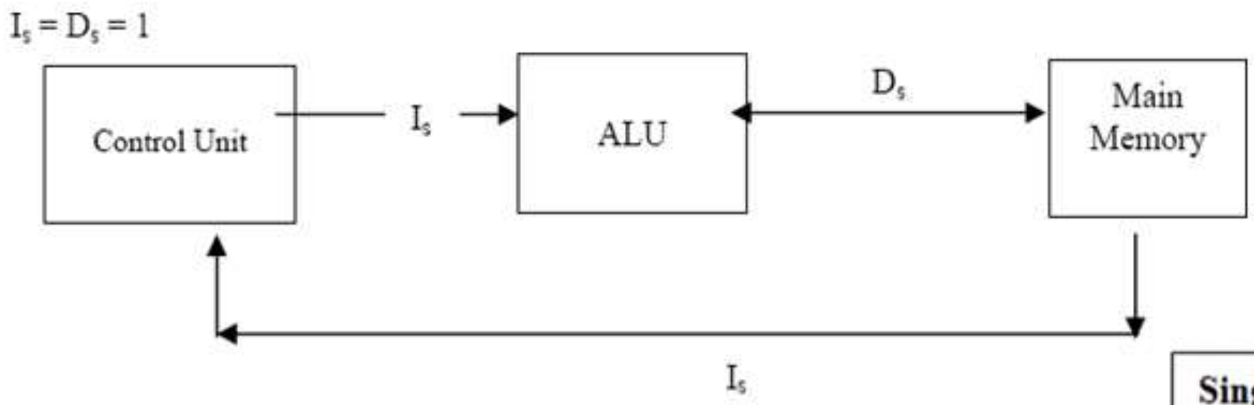
Ex: PCs, Workstations

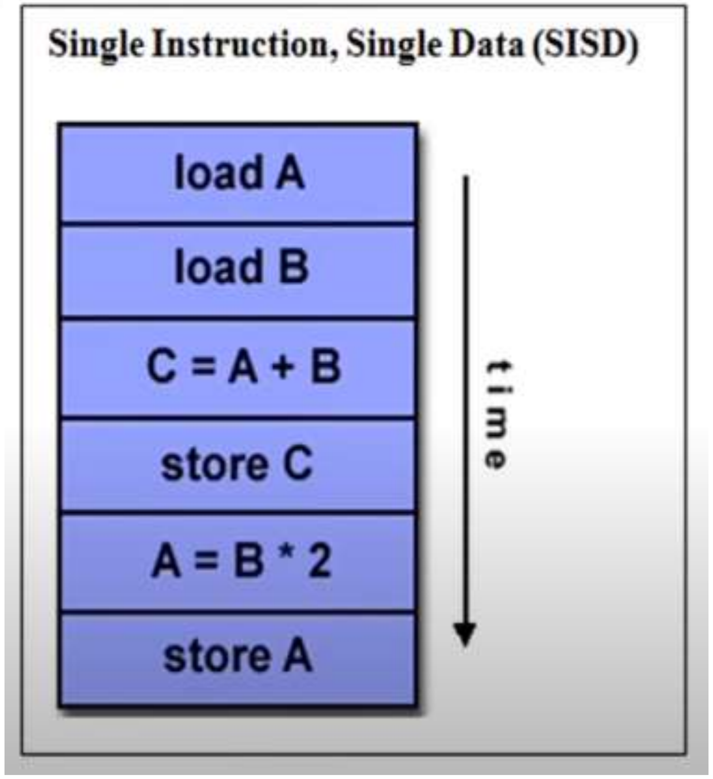
SISD Computers

- SISD computing system is a uni-processor machine which is capable of executing a single instruction, operating on a single data stream.
- **Single instruction:** only one instruction stream is being acted on by the CPU during any one clock cycle.
- **Single data:** only one data stream is being used as input during any one clock cycle

- Conventional single processor Von Neumann computers are classified as SISD system.
- It is a serial(non-parallel) computer.
- Instruction are executed sequentially, but may be overlapped in their execution stages (pipelining). Most SISD uni-processor system are pipelined
- SISD computers may have more than one functional units, all under the supervision of control unit.
- Examples: most PC’s, single CPU workstation, minicomputers, mainframes. CDC-6600, CDC 7600, VAX 11, IBM 7001 are SISD computers.

SISD Organization

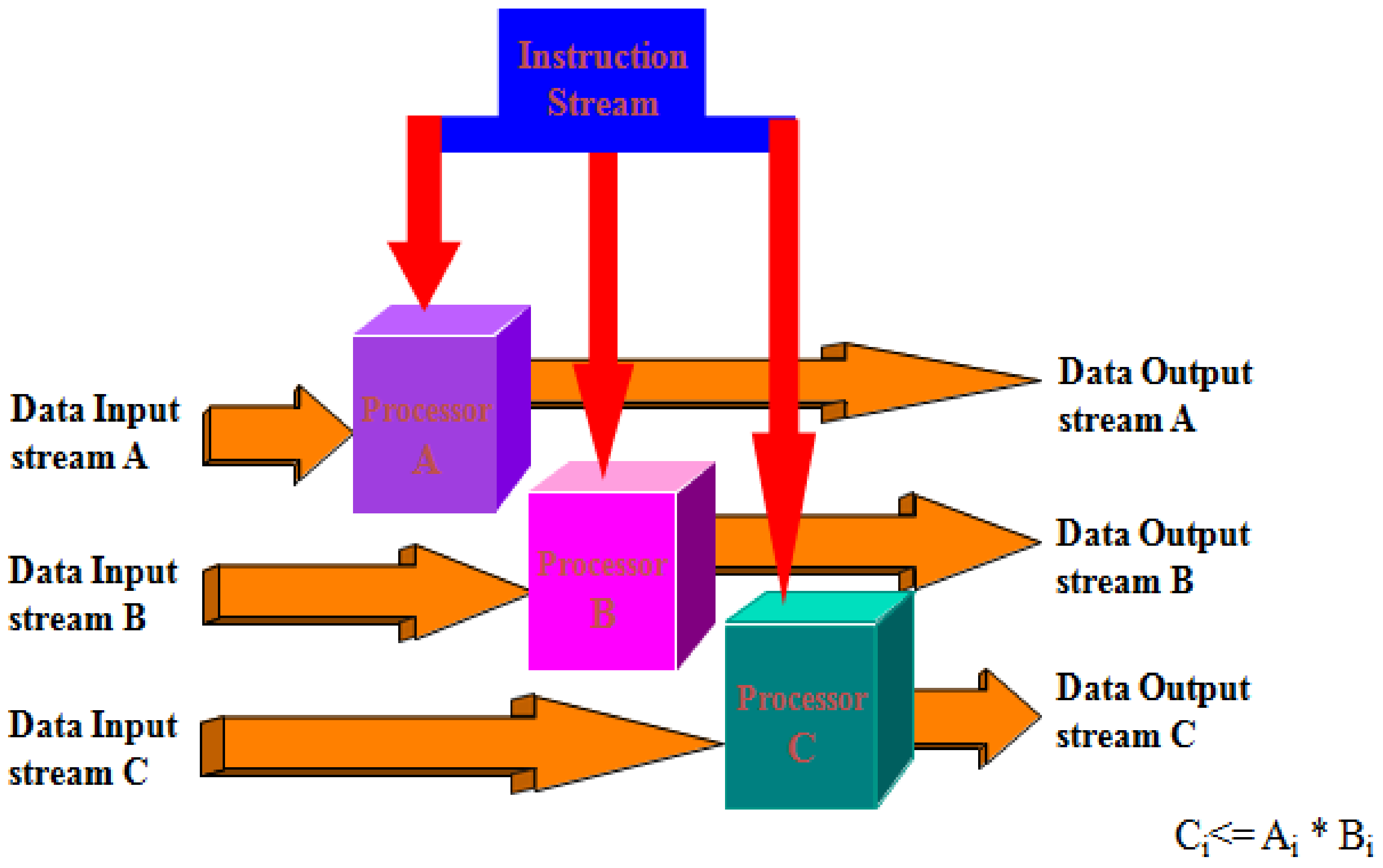




SIMD Computers

- SIMD system is capable of executing the same instruction on all the CPUs but operating on different data streams.
- **Single instruction:** All processing unit execute the same instruction at any given clock cycle
- **Multiple data:** Each processing unit can operate on a different data element

- SIMD computer has single control unit which issues one instruction at a time but it has multiple ALU’s or processing units to carry out on multiple data set simultaneously
- Well suited to scientific computing since they involve lots of vector and matrix operation.
- Example: Array processors and vector pipelines
- Array Processors: ILLIAC-IV, MPP
- Vector pipelines: IBM 9000, Cray X-MP, Y-MP & C90



Ex: CRAY machine vector processing, Intel MMX (multimedia support)

All processing elements are interacting with the common shared memory for the organization of single data stream

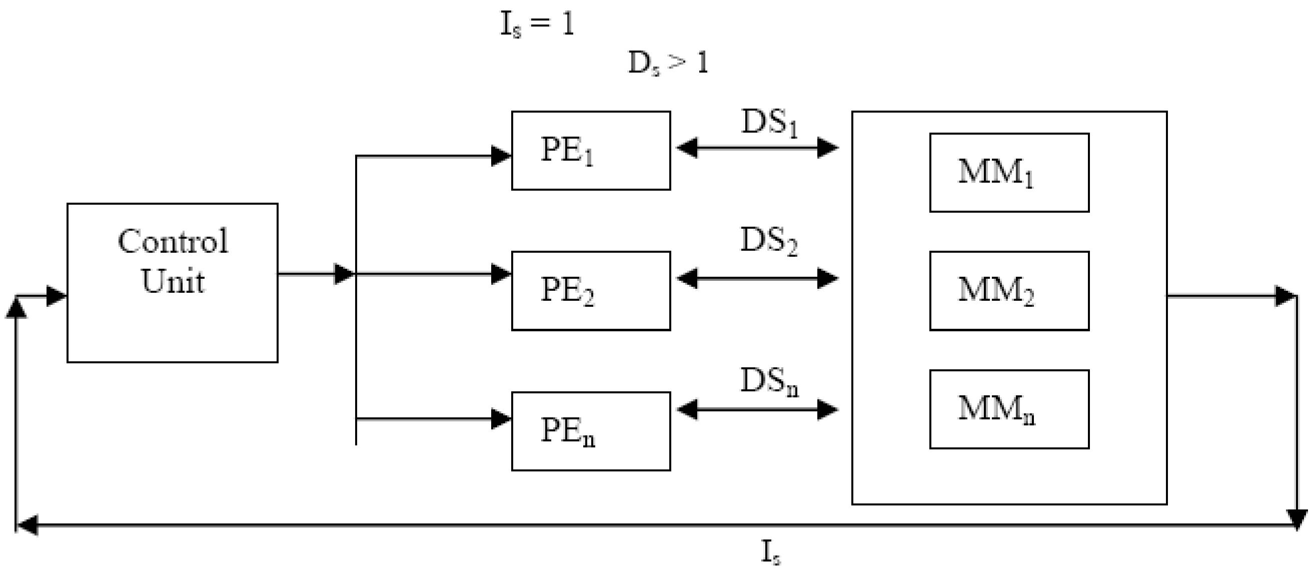
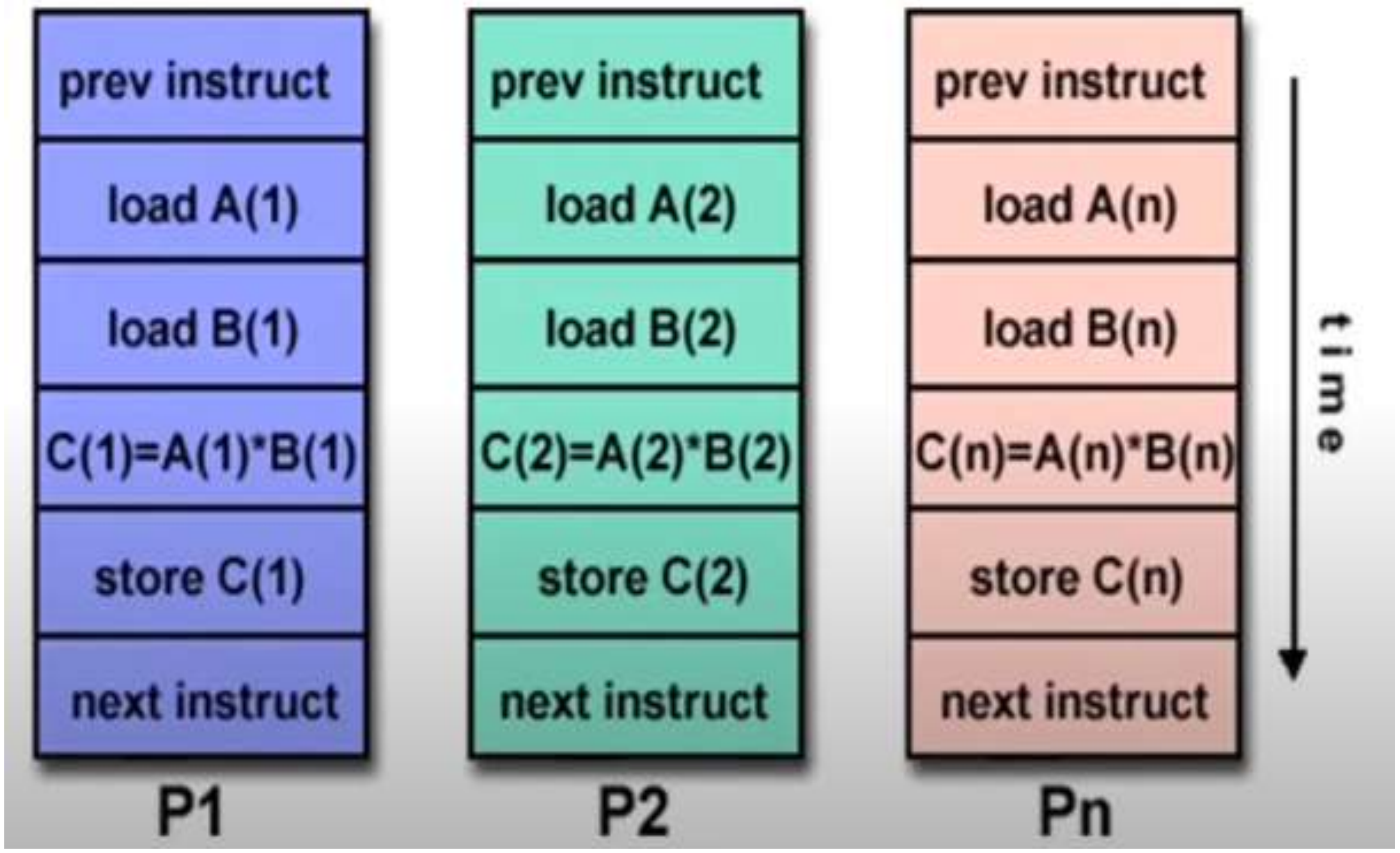


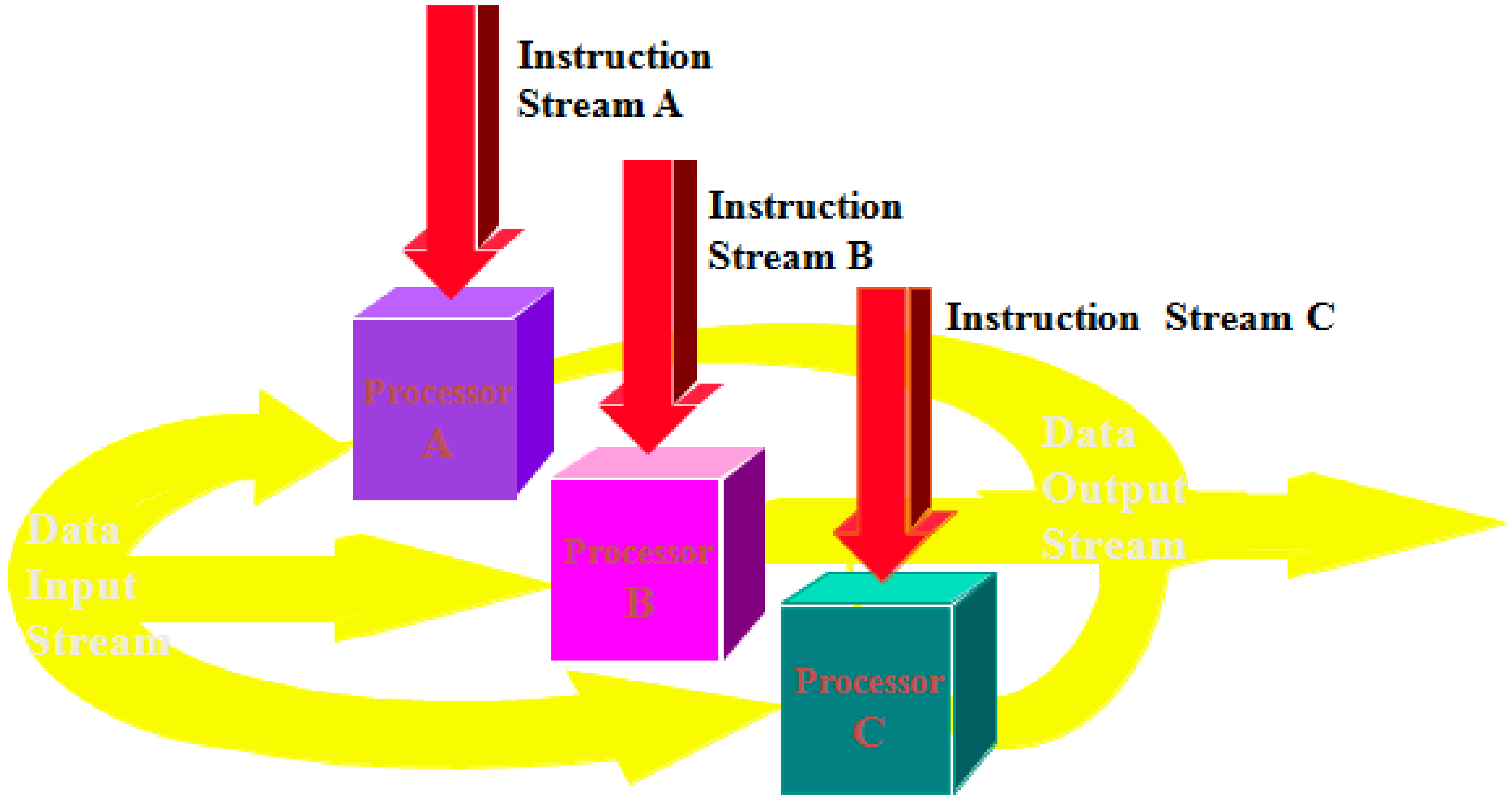
Figure 5: SIMD Organisation

Single Instruction, Multiple Data (SIMD)



MISD

- In MISD , multiple instruction operate on single data stream.
- MISD computing system is capable of executing different instruction on different PUs but all of them operating on the same dataset.
- Multiple instruction : Each processing unit may be executing a different instruction stream
- single data : every processing unit can operate on a same data element
- Machines built using the MISD model are practically not useful in most of the application, a few machines are built, but none of them are available commercially.
- Example : Systolic Array



➔ More of an intellectual exercise than a practical configuration. Few built, but commercially not available

$I_s > 1$
$D_s = 1$

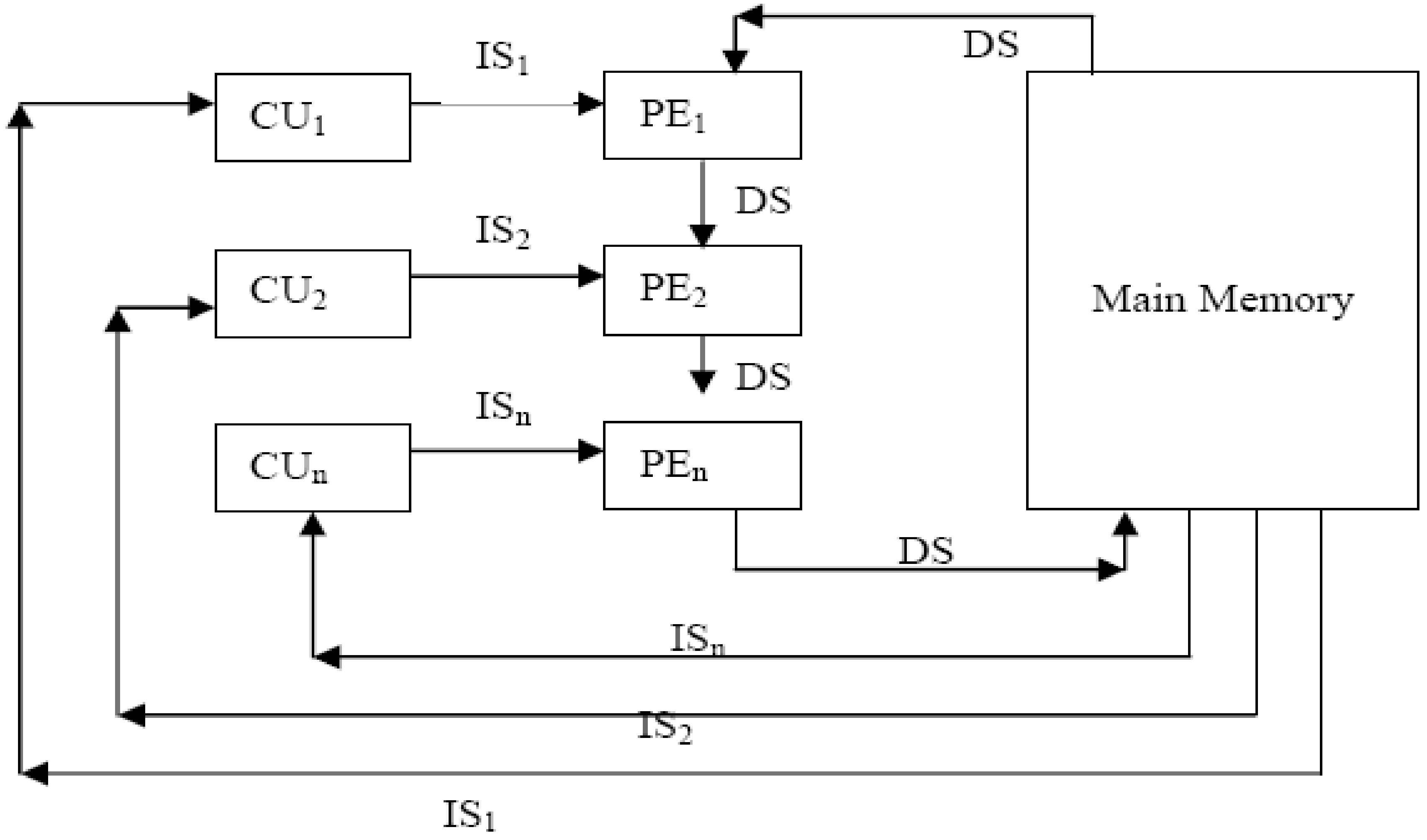
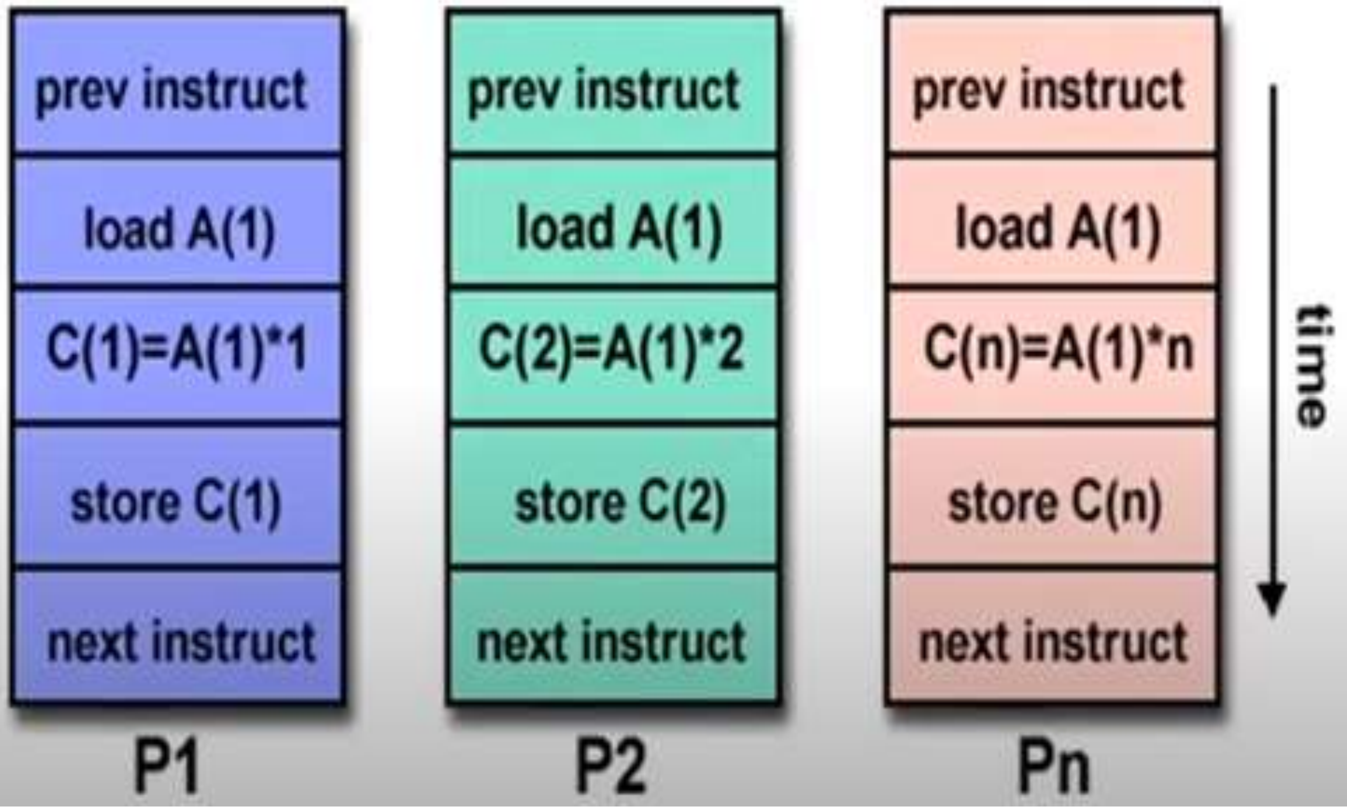


Figure 6: MISD Organisation

Multiple Instruction, Single Data (MISD)



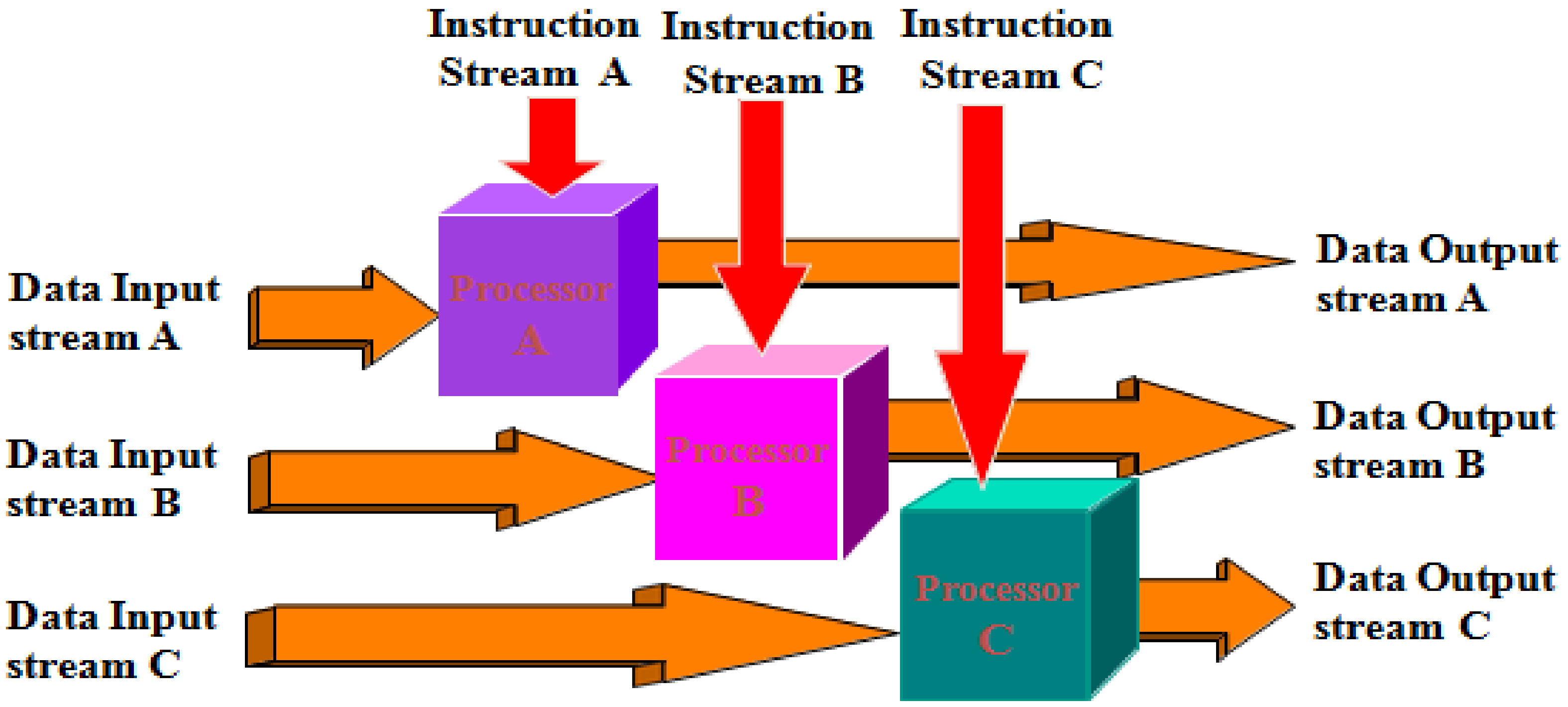
Advantages of MISD

– for the specialized applications like
  - Real time computers need to be fault tolerant where several processors execute the same data for producing the redundant data.
  - All these redundant data are compared as results which should be same otherwise faulty unit is replaced.
  - Thus MISD machines can be applied to fault tolerant real time computers.

MIMD Architecture

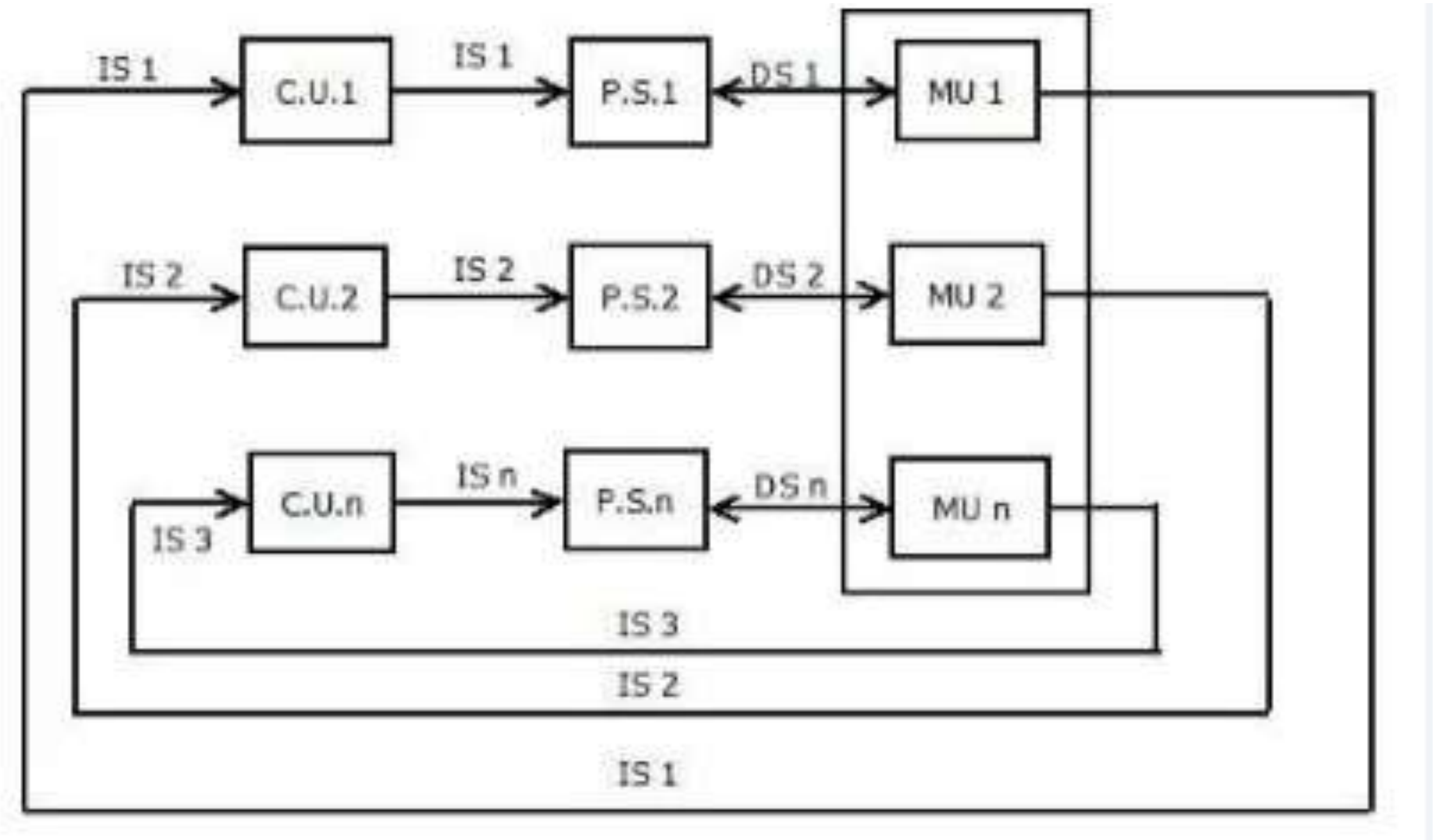
- MIMD system is capable of executing multiple instructions on multiple data sets.
- **Multiple instruction :** Each processing unit may be executing a different instruction stream
- **Multiple data:** Each processing unit can operate on a different data element
- MIMD systems are parallel computers capable of processing several programs simultaneously.
- Can be categorized as loosely coupled or tightly coupled depending on sharing of data and control.

- Examples :C.mmp, Cray-2, Cray X-MP, IBM 370/168 MP, Univac 1100/80, IBM 3081/3084.
- MIMD organization is the most popular for a parallel computer.
- In the real sense, parallel computers execute the instructions in MIMD mode

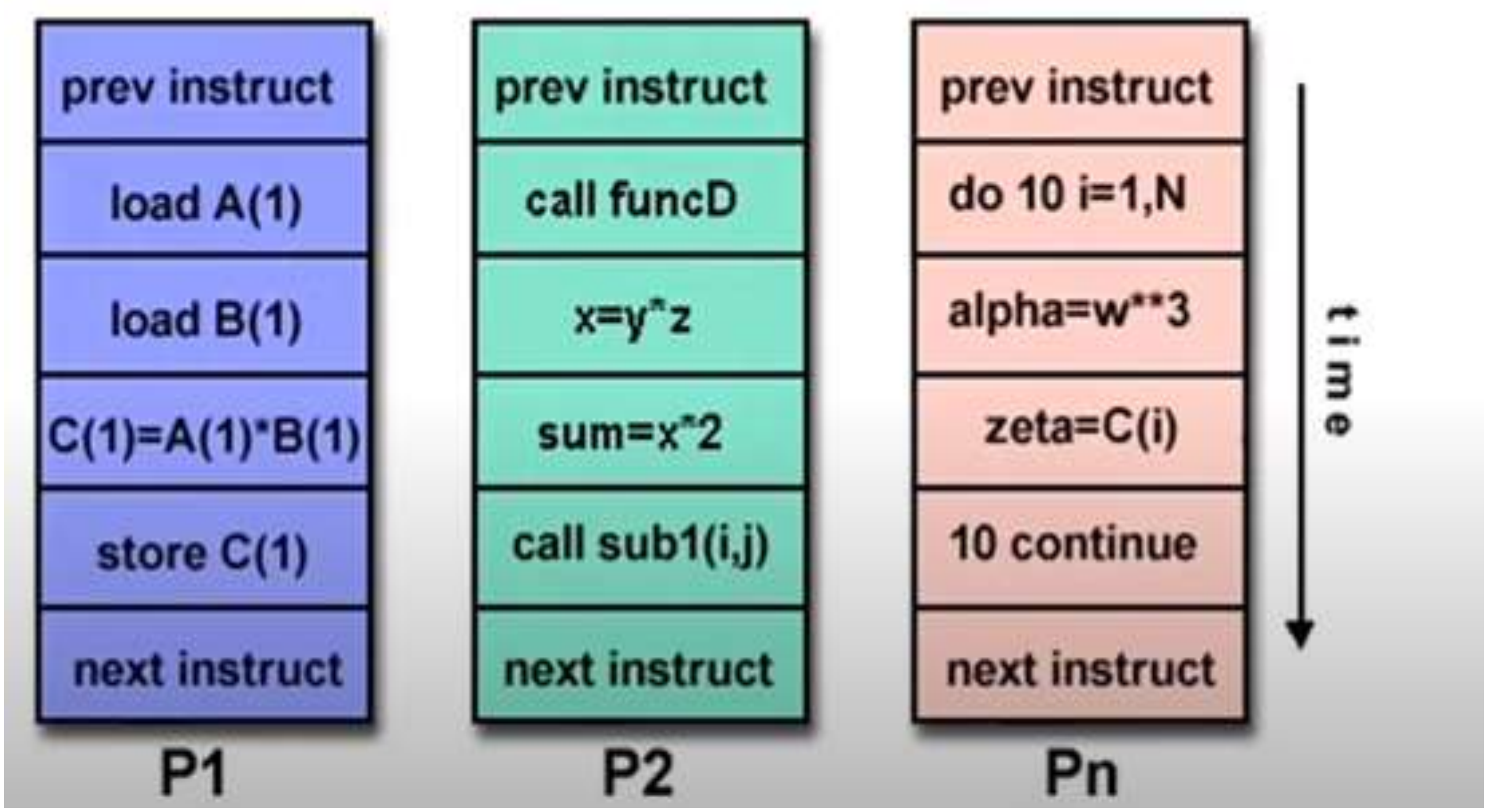


Unlike SISD, MISD, MIMD computer works asynchronously.

- Shared memory (tightly coupled) MIMD
- Distributed memory (loosely coupled) MIMD


IS 1
C.U.1
IS 1
P.S.1
DS 1
MU 1
IS 2
C.U.2
IS 2
P.S.2
DS 2
MU 2
C.U.n
IS n
P.S.n
DS n
MU n
IS 3
IS 3
IS 2
IS 1

Multiple Instruction, Multiple Data (MIMD)



HANDLER'S CLASSIFICATION

- In 1977, Handler proposed an elaborate notation for expressing the pipelining and parallelism of computers.
- Handler's classification addresses the computer at three distinct levels:
  - Processor control unit (PCU) corresponds to one processor or one CPU.
  - Arithmetic logic unit (ALU) equivalent to processing element(PE)
  - Bit-level circuit (BLC) corresponds to combinational logic circuit need to perform 1-bit operation in the ALU .

Way to describe a computer

- **Handler’s classification uses the three pairs of integers containing six independent entities to describe a computer**
- **Computer = (p * p', a * a', b * b')**
- Where p = number of PCUs
- p'= number of PCUs that can be pipelined
- a = number of ALUs controlled by each PCU
- a'= number of ALUs that can be pipelined
- b = number of bits in ALU or processing element (PE) word
- b'= number of pipeline segments on all ALUs or in a single PE

Relationship between various elements of the computer

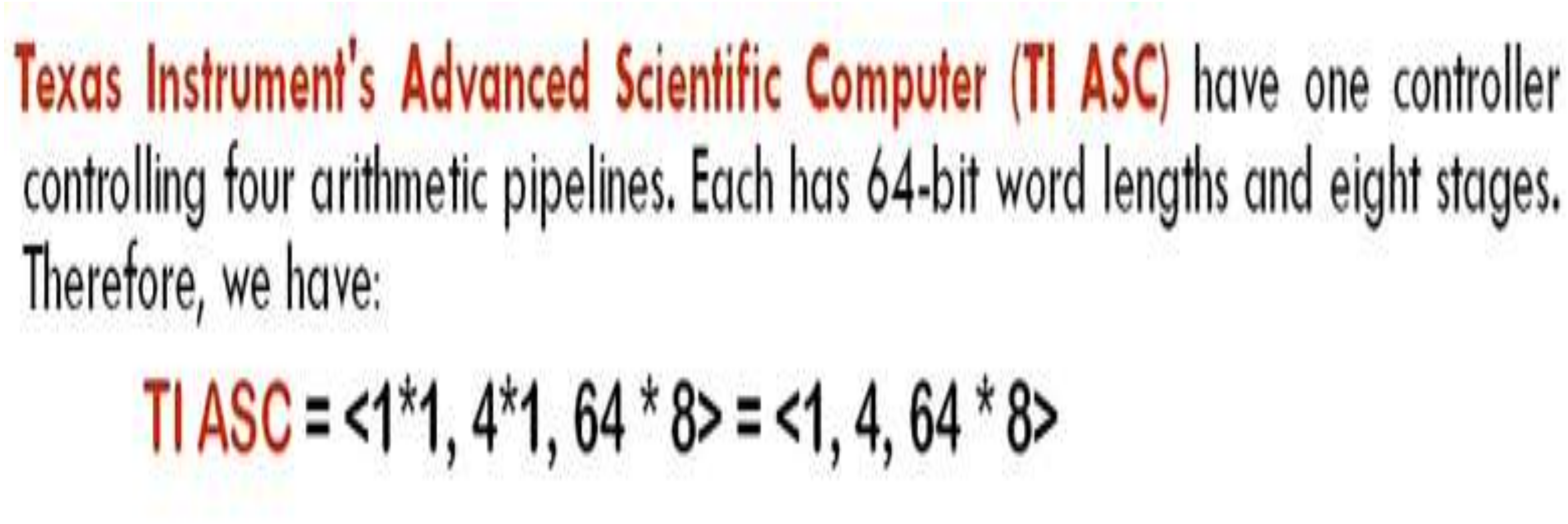
- The '*' operator is used to indicate that the units are pipelined or macro-pipelined with a stream of data running through all the units.

- The '+' operator is used to indicate that the units are no pipelined but work on independent streams of data.

- The 'v' operator is used to indicate that the computer hardware can work in one of several modes.

- The '~' symbol is used to indicate a range of values for any one of the parameters.

Example



Texas Instrument's Advanced Scientific Computer (TI ASC) have one controller controlling four arithmetic pipelines. Each has 64-bit word lengths and eight stages. Therefore, we have:

TI ASC = <1*1, 4*1, 64 * 8> = <1, 4, 64 * 8>

Example

- The CDC 6600 has a single main processor with an ALU that supported by 10 I/O processors. One control unit coordinates one ALU with a 60-bit word length. The ALU has 10 functional units which can be formed into a pipeline. The 10 peripheral I/O processors may work in parallel with each other and with the CPU. Each I/O processor contains one 12-bit word length with ALU. we specify CDC 6600 into two parts.

- CDC 6600 I/O Processor= (10*1, 1*1, 12*1)

- The description for the main(Central) processor is:
- CDC 6600main = (1*1, 1 * 10, 60*1)

- The main processor and the I/O processors can be regarded as forming a macro-pipeline so the '*' operator is used to combine the two structures:

- CDC 6600 = (I/O processors) * (central processor) = (10, 1, 12) * (1, 10, 60)

STRUCTURAL CLASSIFICATION

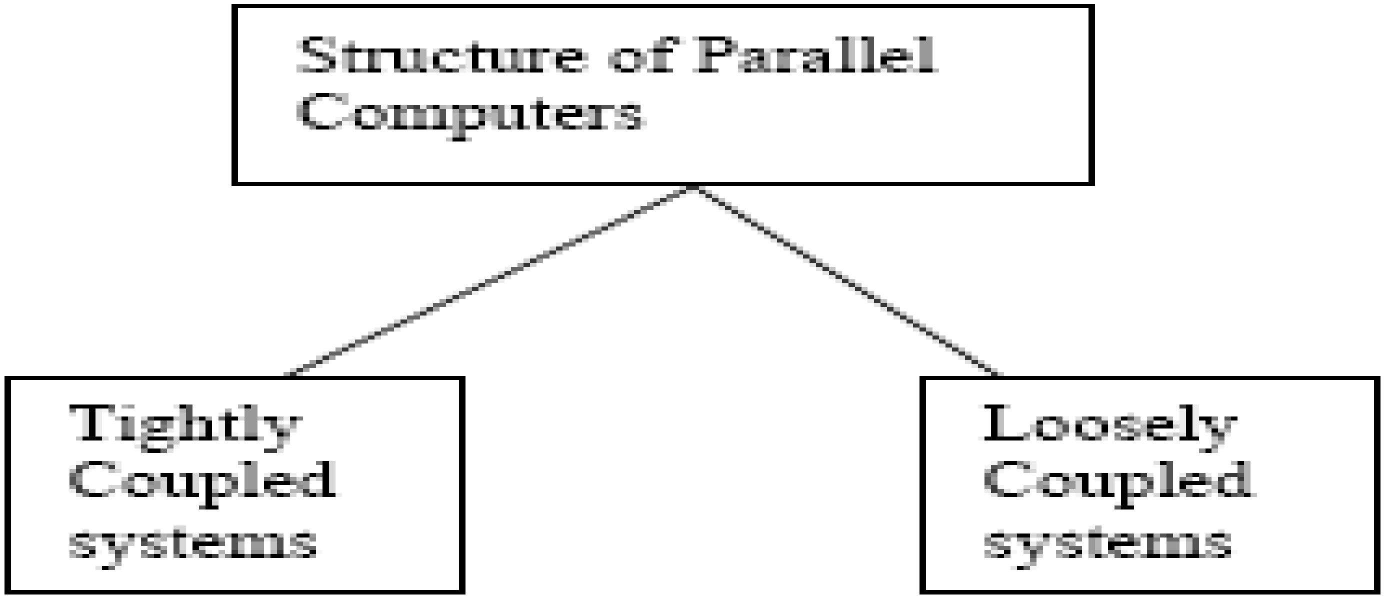


Figure 8: Structural classification

STRUCTURAL CLASSIFICATION

- A parallel computer (MIMD) can be characterized as a set of multiple processors and shared memory or memory modules communicating via an interconnection network.
- When multiprocessors communicate through the global shared memory modules then this organization is called **Shared memory computer** or **Tightly coupled systems**

- Shared memory multiprocessors have the following characteristics:
  - Every processor communicates through a shared global memory

  - For high speed real time processing, these systems are preferable as their throughput is high as compared to loosely coupled systems.

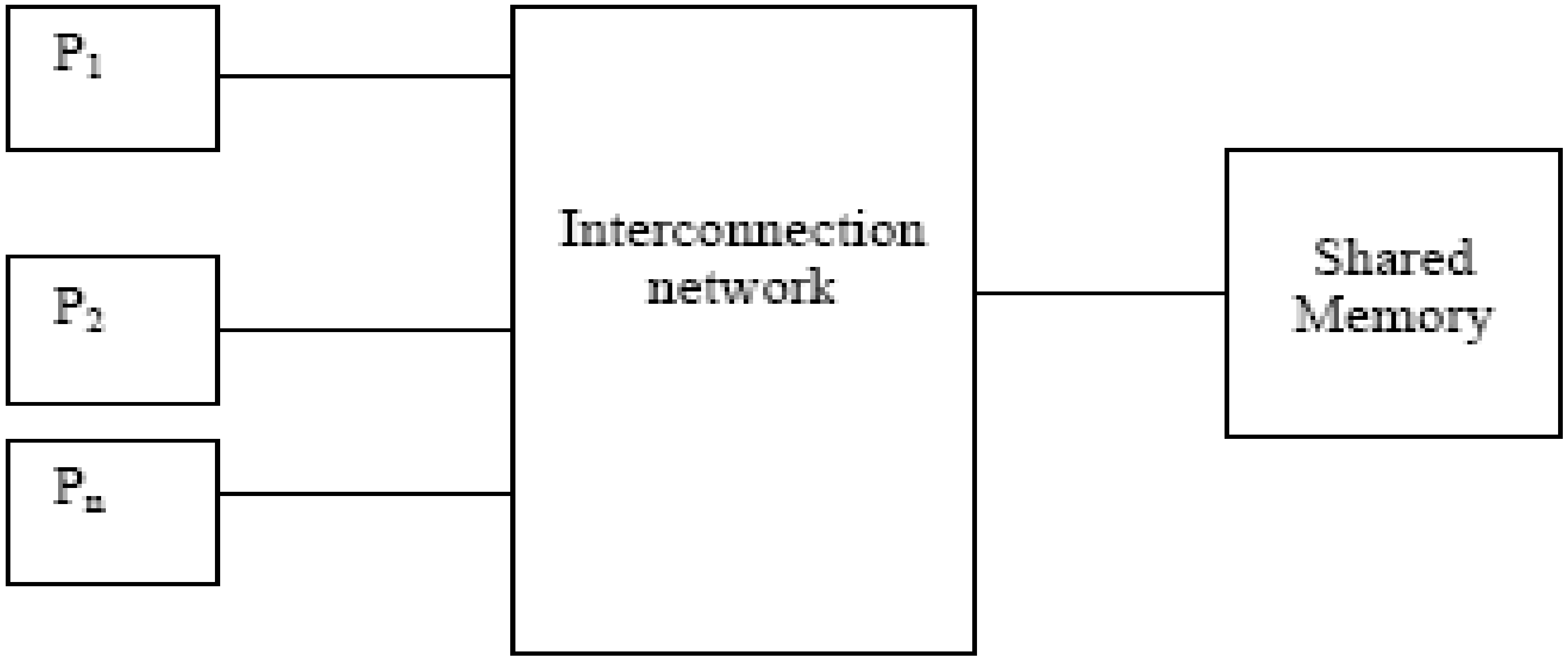


Figure 9: Tightly coupled system

- In tightly coupled system organization, multiple processors share a global main memory, which may have many modules.

- The processors have also access to I/O devices. The inter- communication between processors, memory, and other devices are implemented through various **interconnection networks,**

Types of Interconnection n/w

- **Processor-Memory Interconnection Network (PMIN)**
  - This is a switch that connects various processors to different memory modules.
- **Input-Output-Processor Interconnection Network (IOPIN)**
  - This interconnection network is used for communication between processors and I/O channels
- **Interrupt Signal Interconnection Network (ISIN)**
  - When a processor wants to send an interruption to another processor, then this interrupt first goes to ISIN, through which it is passed to the destination processor. In this way, synchronization between processor is implemented by ISIN.

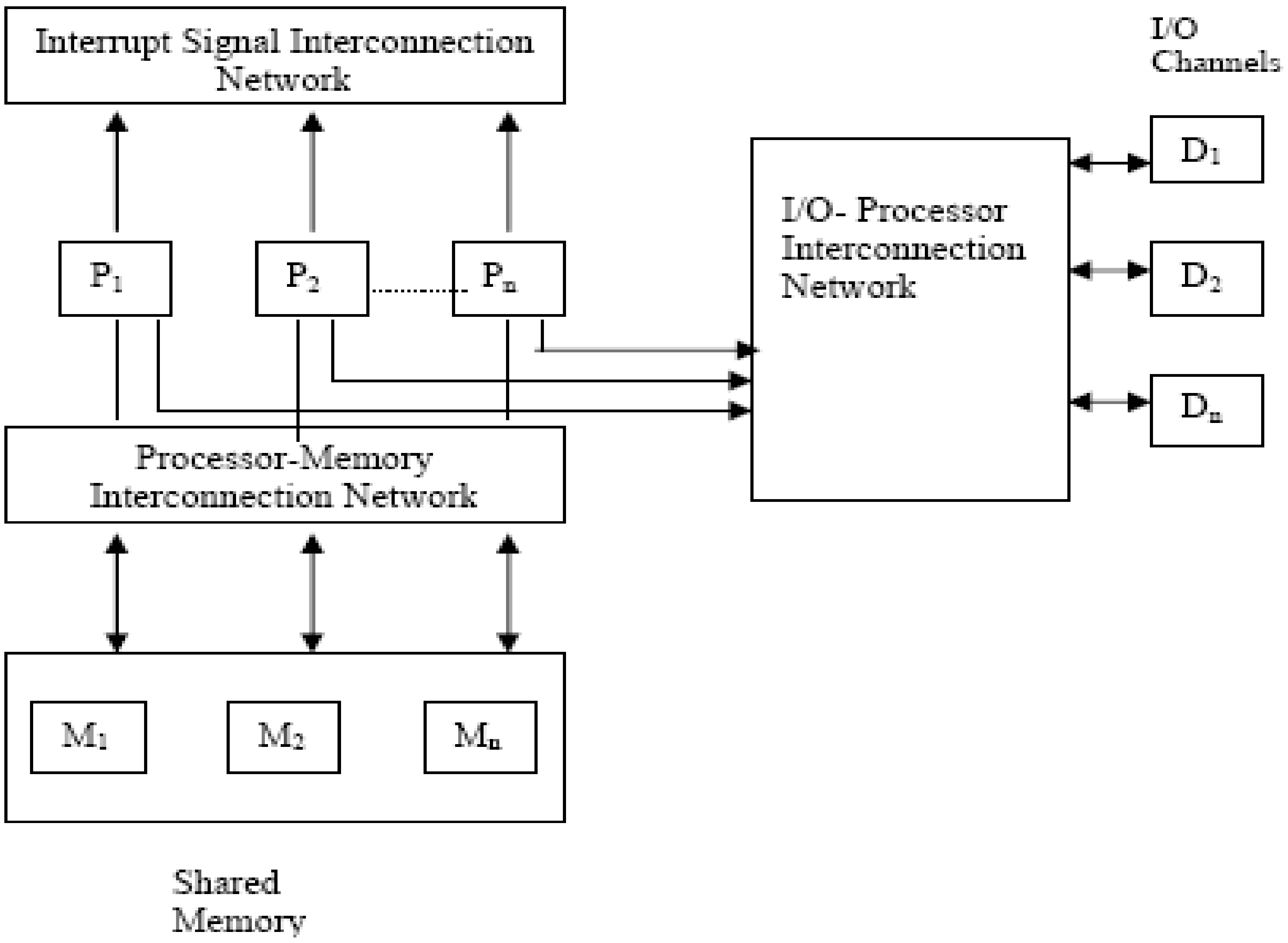
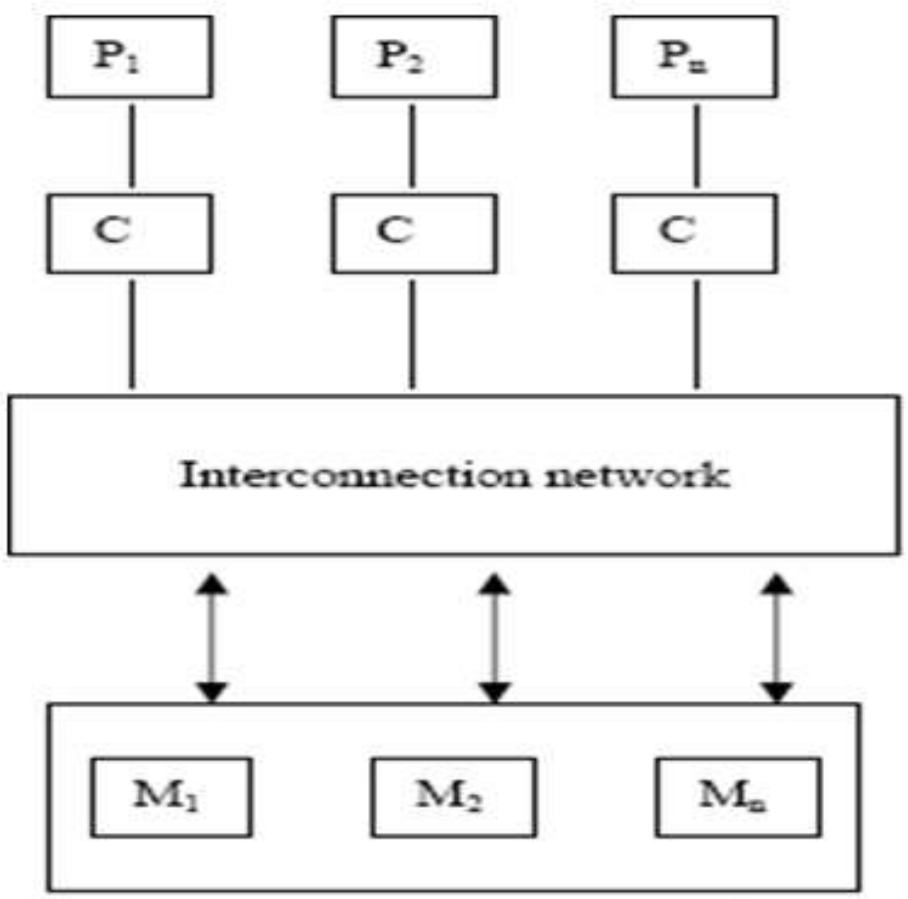

Interrupt Signal Interconnection Network
I/O Channels
P1
P2
Pn
I/O- Processor Interconnection Network
D1
D2
Dn
Processor-Memory Interconnection Network
M1
M2
Mn
Shared Memory

Figure 11: Tightly coupled system organization

- To reduce this delay, every processor may use cache memory for the frequent references made by the processor as



Tightly coupled systems with cache memory

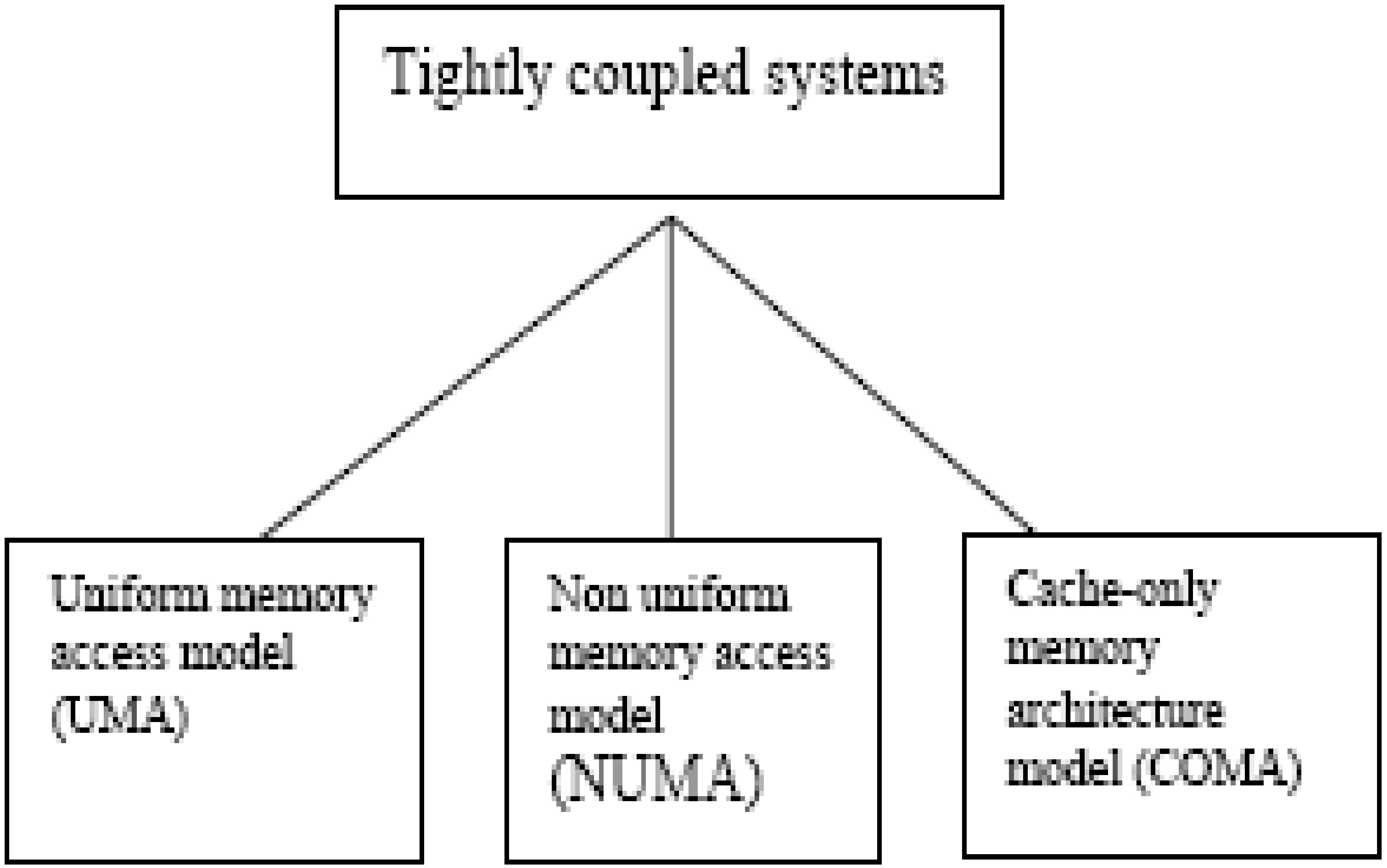
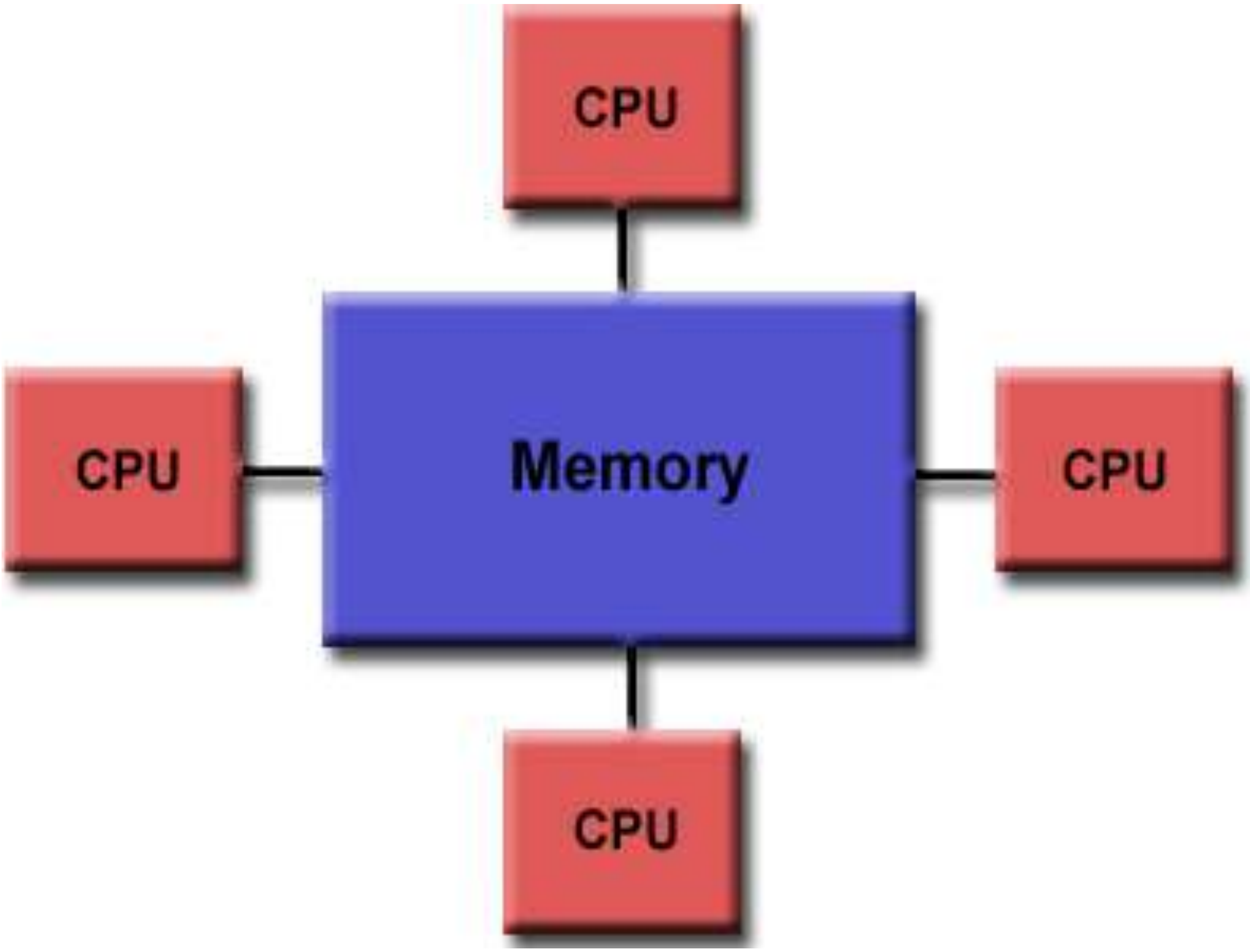

Tightly coupled systems
Uniform memory access model (UMA)
Non uniform memory access model (NUMA)
Cache-only memory architecture model (COMA)

Figure 13: Modes of Tightly coupled systems

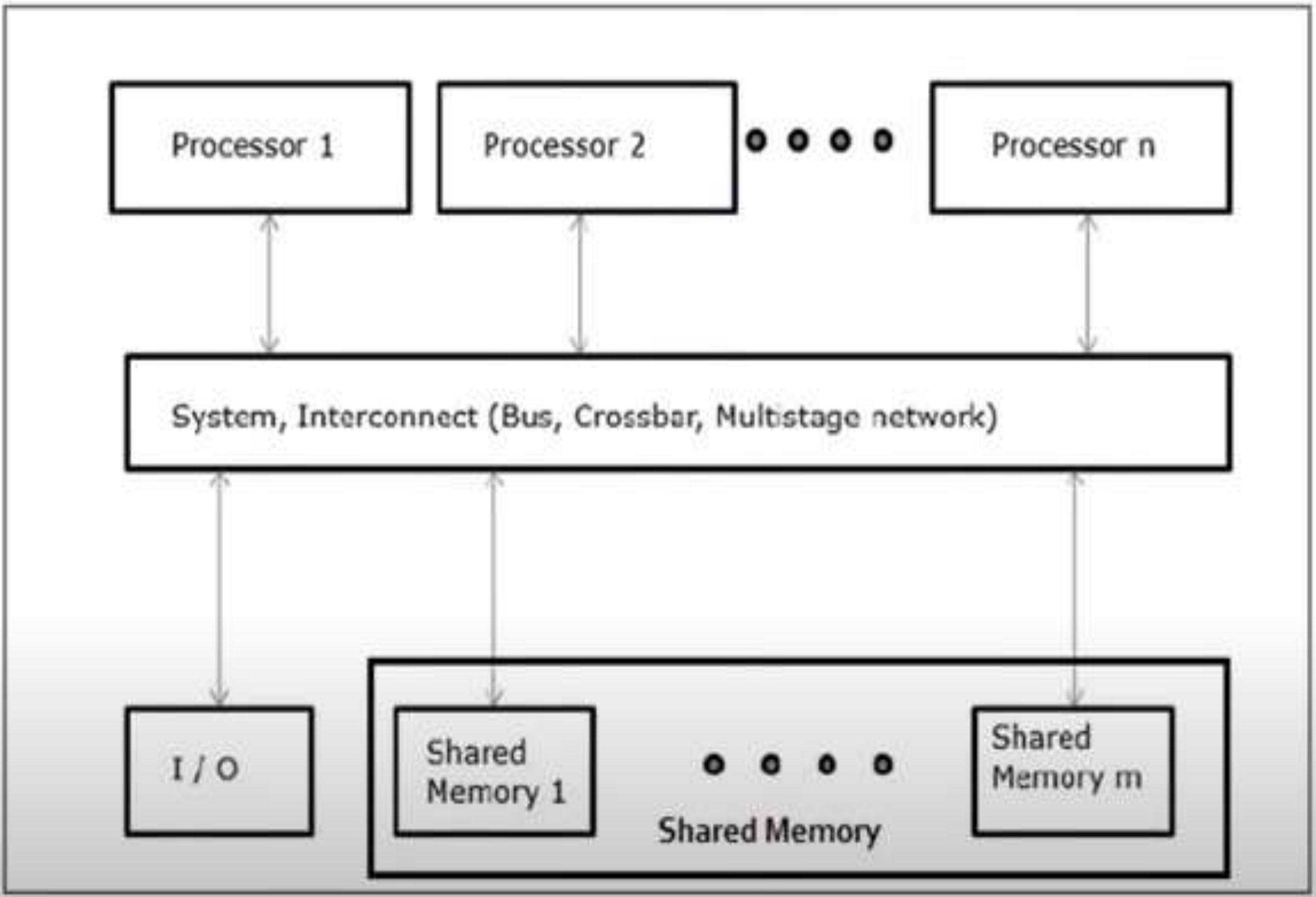
Uniform Memory Access Model (UMA)

- Uniform Memory Access(UMA) is a shared memory architecture used in parallel computer.
- In UMA model the physical memory is shared uniformly by all the processors.
- All the processors have equal access time to all memory location(which is why it is called Uniform Memory Access(UMA).

- Most commonly represented today by Symmetric Multiprocessor (SMP) machines
- Each processor may have private cache. Sometimes called **CC-UMA - Cache Coherent UMA**.
- **Cache coherent** means if one processor updates a location in shared memory, all the other processors know about the update. Cache coherency is accomplished at the hardware level.


CPU
CPU
Memory
CPU
CPU

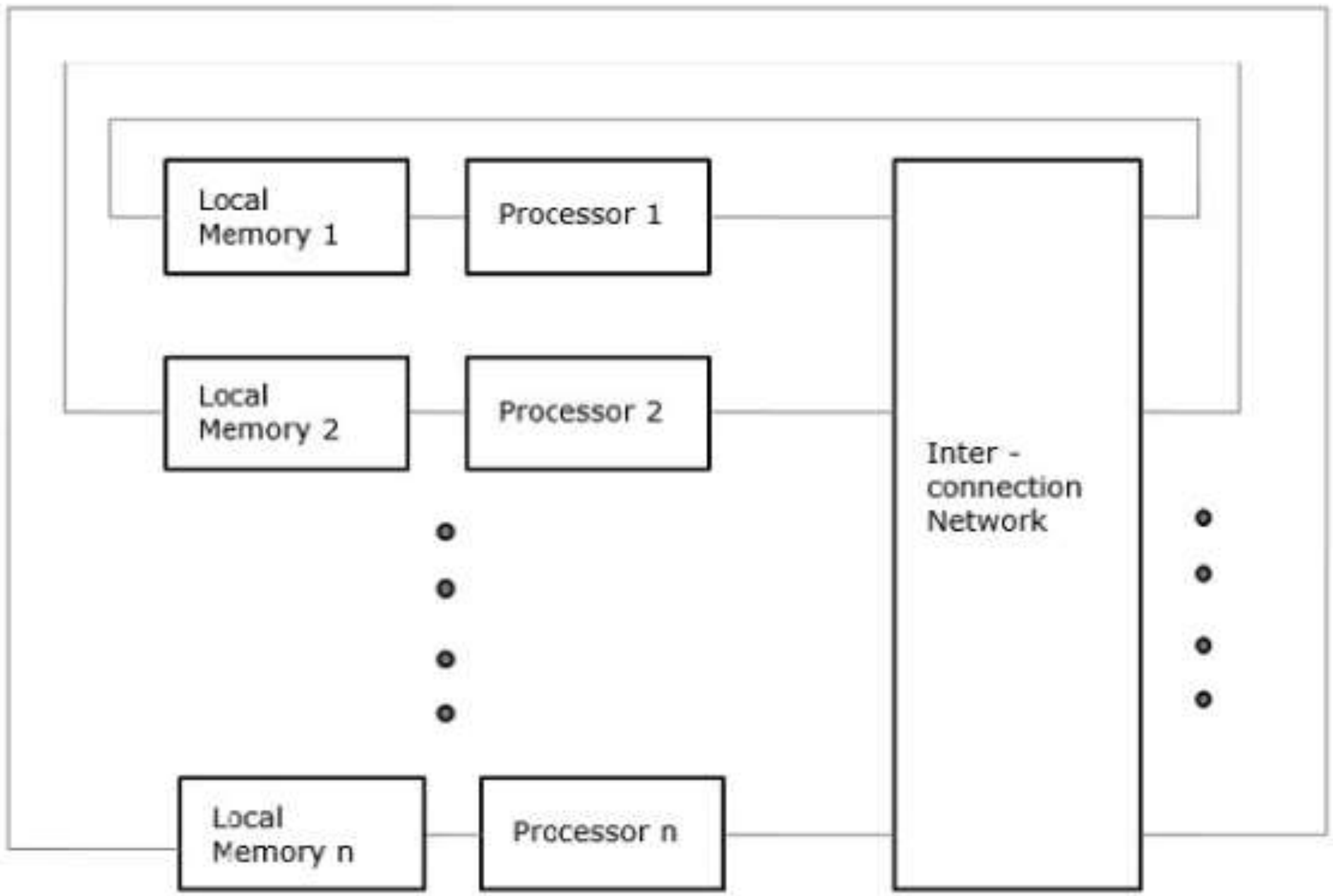
The UMA Multiprocessor model



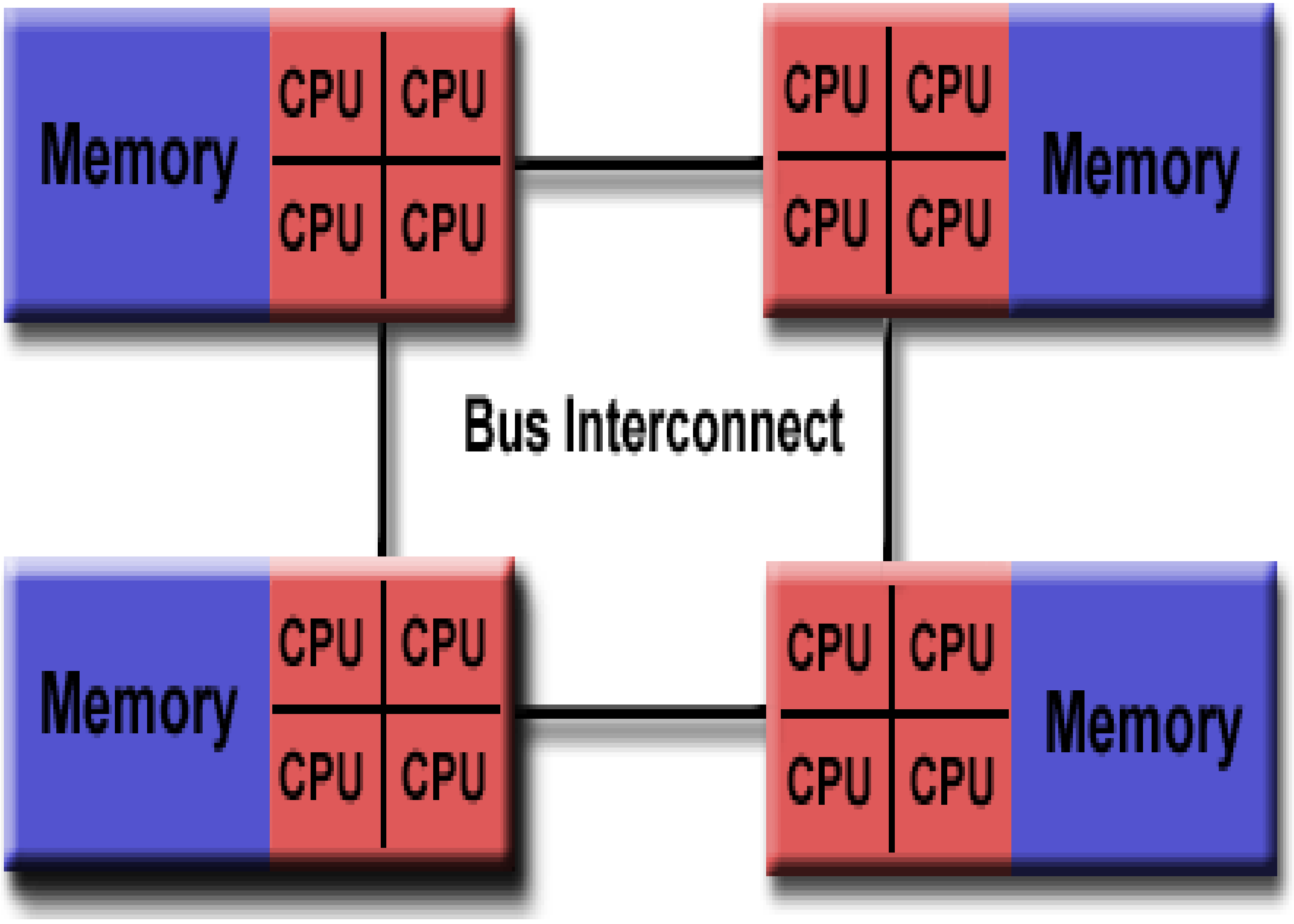
Non-Uniform Memory Access Model (NUMA)

- A NUMA Multiprocessor model shared memory system in which the access time varies with the location of the memory word.
- The shared memory is physically distributed among all the processors, called local memories. The collection of all local memories forms a global address space which can be accessed by all the processors.
- It is faster to access a local memory with a local processor.

- The access of remote memory attached to other processors is slower due to the added delay through the interconnection network.
- The system interconnect takes form of common bus, cross bar switch, multi stage network.

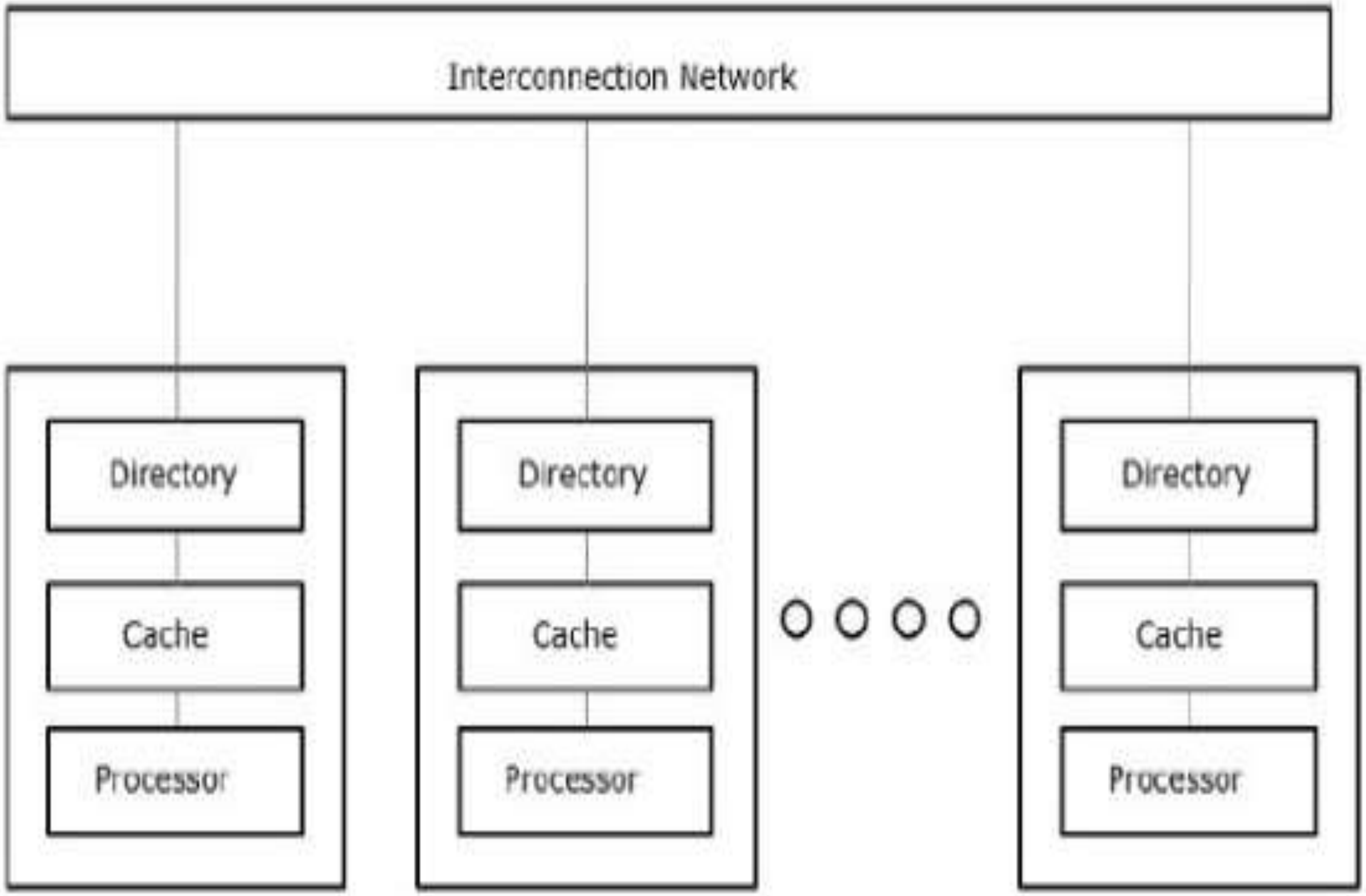

Local
Memory 1
Processor 1
Local
Memory 2
Processor 2
Inter -
connection
Network
Local
Memory n
Processor n

- Often made by physically linking two or more SMPs.
- One SMP can directly access memory of another SMP.
- Not all processors have equal access time to all memories (Which is why it is called Non-Uniform Memory Access)
- Memory access across link is slower.
- If cache coherency is maintained then may also be called CC-NUMA(cache coherent NUMA)


Memory
CPU
CPU
CPU
CPU
CPU
CPU
CPU
CPU
Memory
Bus Interconnect
Memory
CPU
CPU
CPU
CPU
CPU
CPU
CPU
CPU
Memory

Cache only memory architecture (COMA)

- The COMA model is a special case of NUMA machine in which the distributed main memories are converted to Caches.
- All caches form a global address space and there is no memory hierarchy at each processor node.
- Remote cache access is assisted by the distributed cache directories.


Interconnection Network
Directory
Cache
Processor
Directory
Cache
Processor
Directory
Cache
Processor

Advantages:

- Global address space provides a user-friendly programming perspective to memory
- Data sharing between tasks is both fast and uniform due to the proximity of memory to CPUs

Disadvantages:

- Primary disadvantage is the lack of scalability between memory and CPUs. Adding more CPUs can geometrically increases traffic on the shared memory-CPU path, and for cache coherent systems, geometrically increase traffic associated with cache/memory management.
- Programmer responsibility for synchronization constructs that ensure "correct" access of global memory.
- Expense: it becomes increasingly difficult and expensive to design and produce shared memory machines with ever increasing numbers of processors.

Loosely coupled system

- when every processor in a multiprocessor system, has its own local memory and the processors communicate via messages transmitted between their local memories, then this organization is called **Distributed memory computer** or **Loosely coupled system**

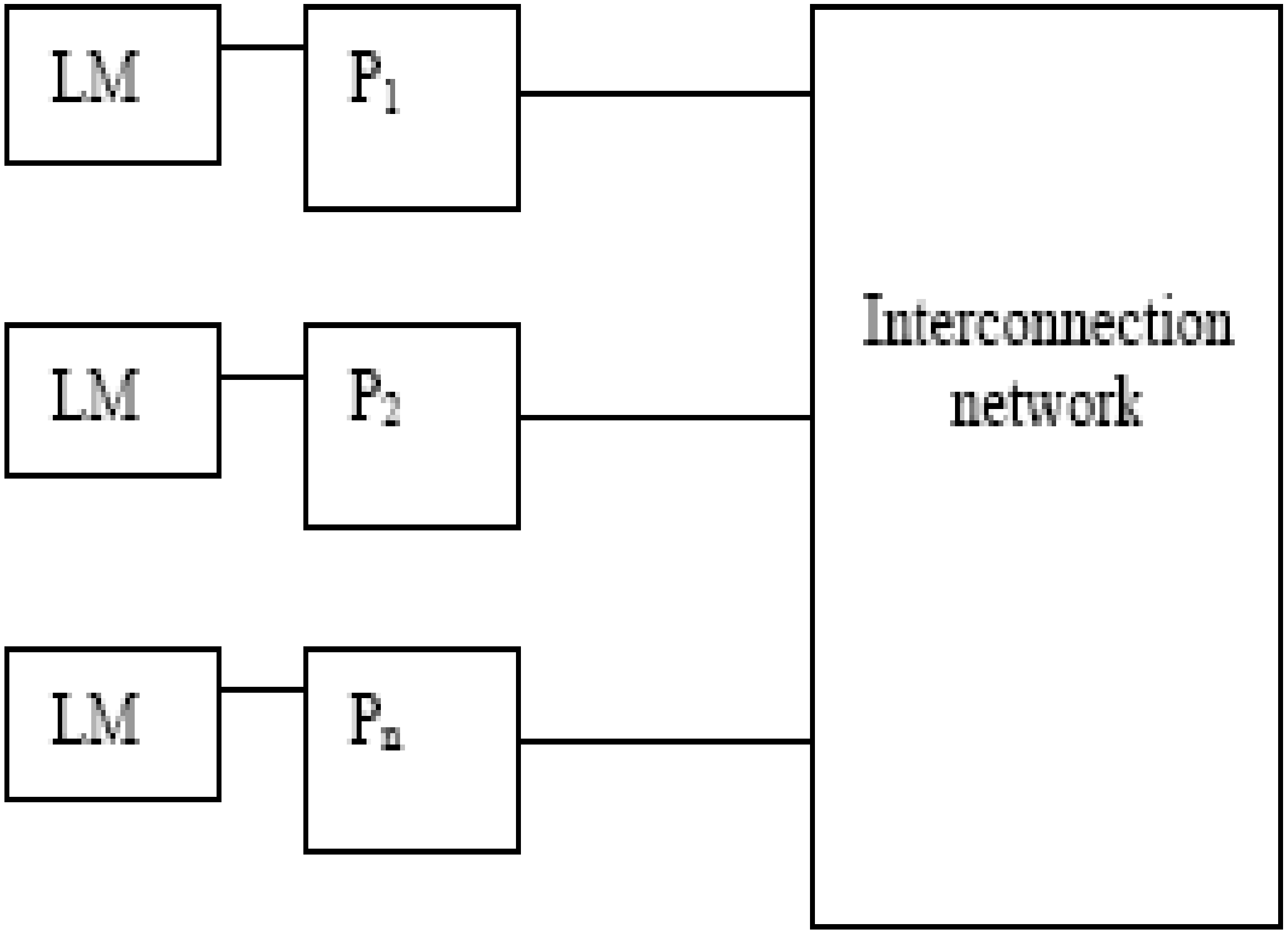
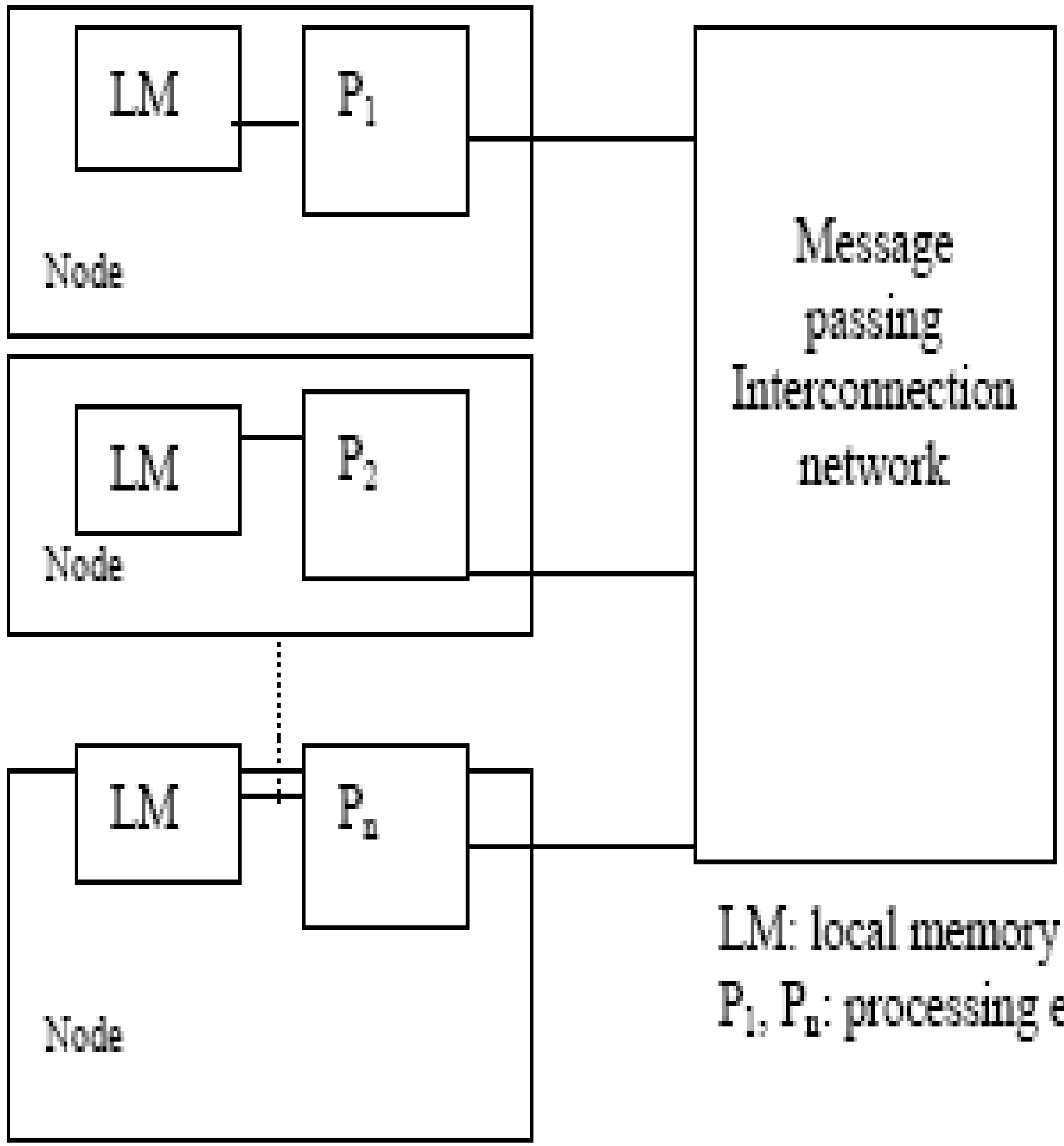


Figure 10 Loosely coupled system

- each processor in loosely coupled systems is having a large local memory (LM), which is not shared by any other processor.
- such systems have multiple processors with their own local memory and a set of I/O devices.
- This set of processor, memory and I/O devices makes a computer system.
- These systems are also called multi-computer systems.

- These computer systems are connected together via message passing interconnection network through which processes communicate by passing messages to one another.
- Also called as distributed multi computer system .



LM: local memory

P_1, P_n: processing elements

Figure 14: Loosely coupled system organisation

CLASSIFICATION BASED ON GRAIN SIZE

- This classification is based on recognizing the parallelism in a program to be executed on a multiprocessor system.
- The idea is to identify the sub-tasks or instructions in a program that can be executed in parallel

Factors affecting decision of parallelism

- Number and types of processors available, i.e. architectural features of host computer
  - Memory organization
  - Dependency of data, control and resources

Parallelism conditions

- The aim of parallel programmer is to exploit parallelism in a given program.
- To execute several program segments in parallel it is necessary that each segment is independent of other segment.
- So, before executing parallelism, all the condition of parallelism between the segments must be analyzed.

Types of dependencies

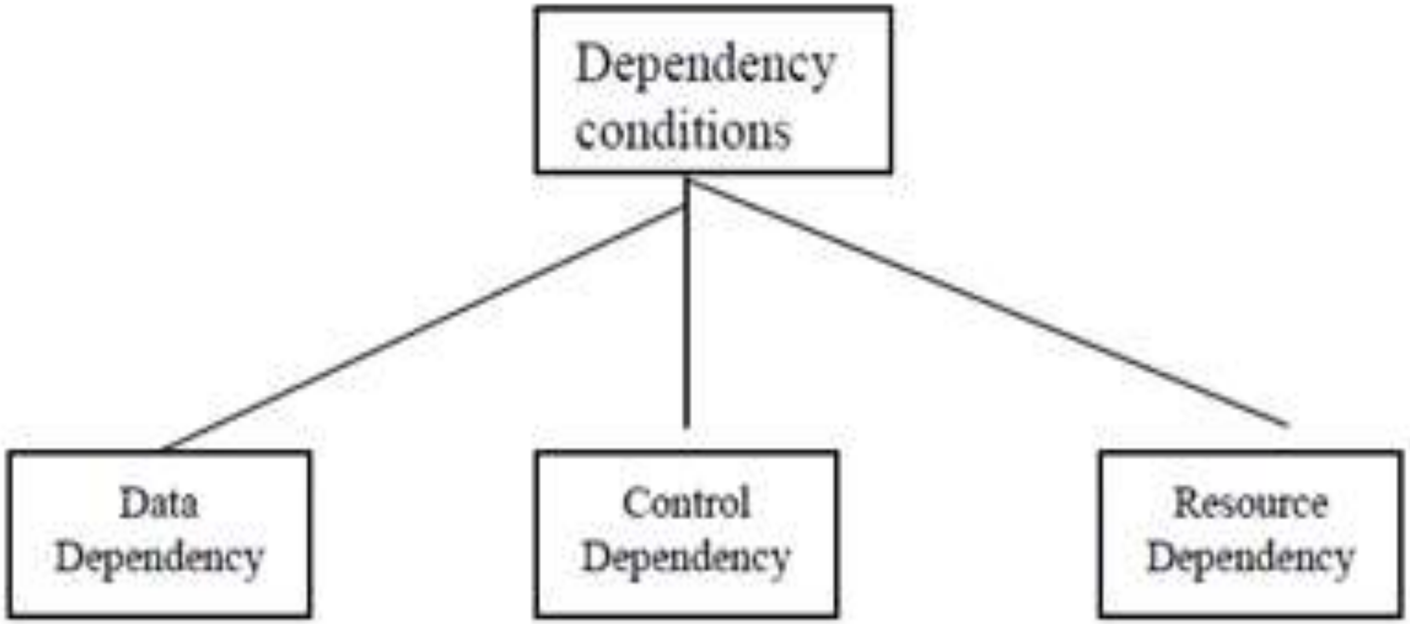


Figure 16: Dependency relations among the segments for parallelism

Dependence Graph

- A dependence graph is used to describe the dependence relationship between the statements.
  - Nodes corresponds to the program statement(instructions)
  - Directed edges with different labels shows the ordered relation among the statements.

Data Dependence

- It refers to the situation in which two or more instructions share same data.
- The instructions in a program can be arranged based on the relationship of data dependency
- how two instructions or segments are data dependent on each other

Types of data dependencies

- Flow Dependence : If instruction I2 follows I1 and output of I1 becomes input of I2, then I2 is said to be flow dependent on I1.
- Anti dependence : When instruction I2 follows I1 such that output of I2 overlaps with the input of I1 on the same data.

- Output dependence : When output of the two instructions I1 and I2 overlap on the same data, the instructions are said to be output dependent.
- I/O dependence : When read and write operations by two instructions are invoked on the same file, it is a situation of I/O dependence.

Control Dependence

- It arise when the order of execution of statements cannot be determined before runtime.
  - Ex. IF condition
- Control dependence often prohibits parallelism from being exploited.
- Compilers are used to eliminate this control dependence and exploit the parallelism

Resource Dependence

- The parallelism between the instructions may also be affected due to the shared resources.
- If two instructions are using the same shared resource then it is a resource dependency condition

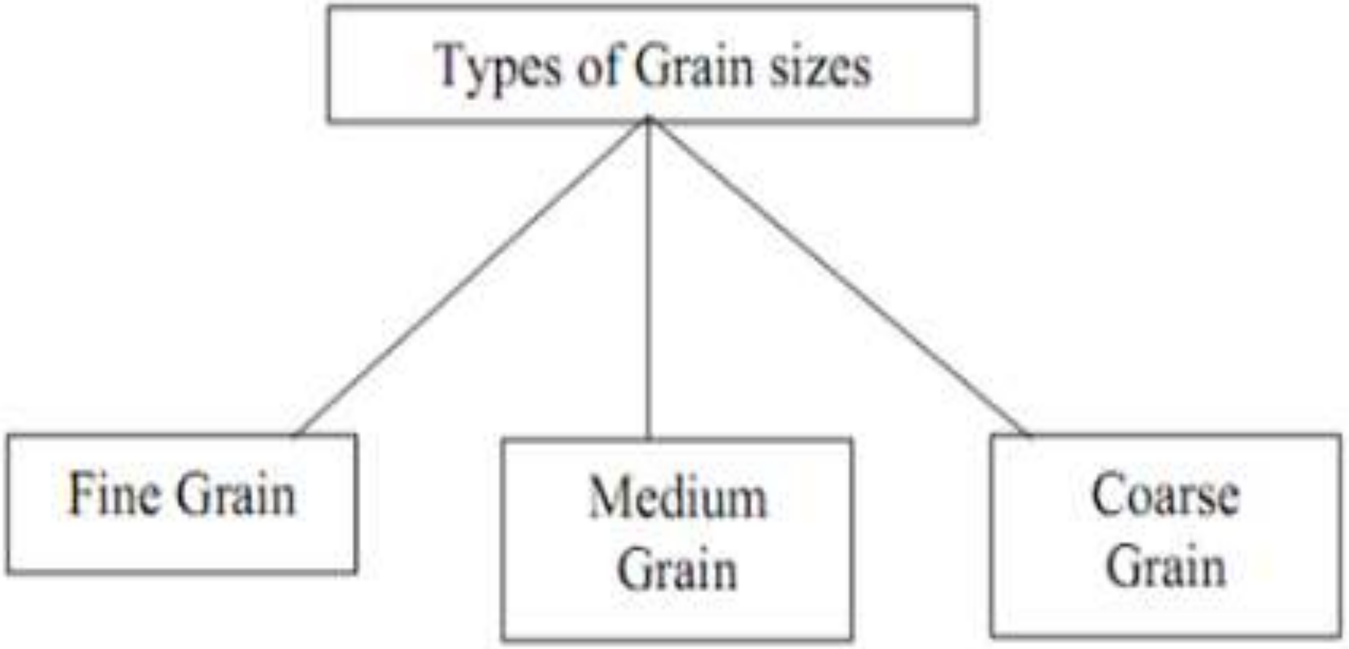
Bernstein Conditions for Detection of Parallelism

- For execution of instructions or block of instructions in parallel, The instructions should be independent of each other.
- These instructions can be data dependent / control dependent/ resource dependent on each other .

Parallelism based on Grain size

- **Grain size:** Grain size or Granularity is a measure which determines how much computation is involved in a process.

- Grain size is determined by counting the number of instructions in a program segment.


Types of Grain sizes
Fine Grain
Medium Grain
Coarse Grain

1. **Fine Grain:** This type contains approximately less than 20 instructions.

2. **Medium Grain:** This type contains approximately less than 500 instructions.

3. **Coarse Grain:** This type contains approximately greater than or equal to one thousand instructions.

LEVELS OF PARALLEL PROCESSING

- **Instruction Level**
- **Loop level**
- **Procedure level**
- **Program level**

Instruction level

- This is the lowest level and the degree of parallelism is highest at this level.
- The fine grain size is used at instruction level
- The fine grain size may vary according to the type of the program. For example, for scientific applications, the instruction level grain size may be higher.
- As the higher degree of parallelism can be achieved at this level, the overhead for a programmer will be more.

Loop Level

- This is another level of parallelism where iterative loop instructions can be parallelized.
- Fine grain size is used at this level also.
- Simple loops in a program are easy to parallelize whereas the recursive loops are difficult.
- This type of parallelism can be achieved through the compilers .

Procedure or Sub Program Level

- This level consists of procedures, subroutines or subprograms.
- Medium grain size is used at this level containing some thousands of instructions in a procedure.
- Multiprogramming is implemented at this level.

Program Level

- It is the **last level consisting** of independent programs for parallelism.
- Coarse grain size is used at this level containing tens of thousands of instructions.
- Time sharing is achieved at this level of parallelism

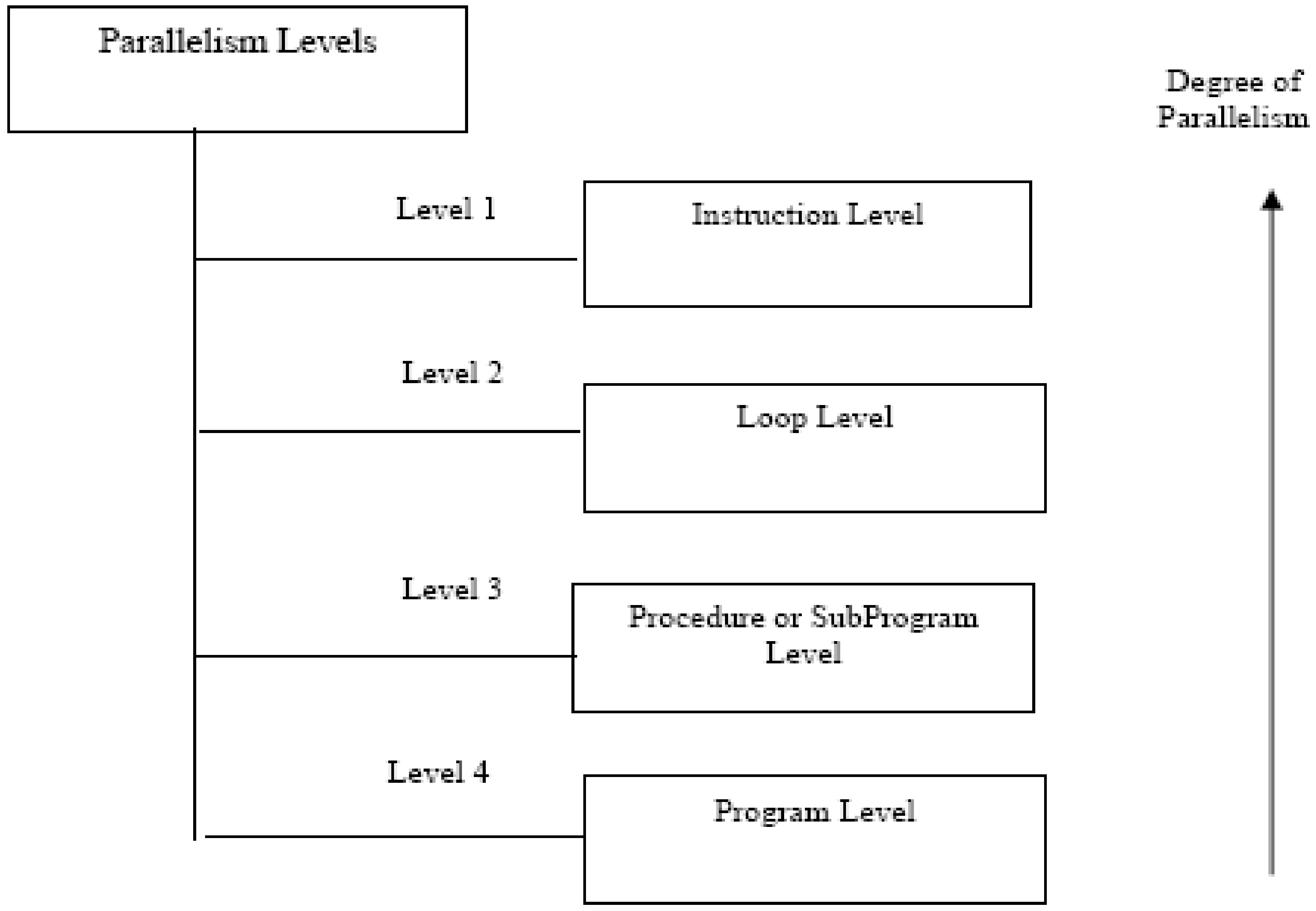


Figure 18: Parallelism Levels

Table 1: Relation between grain sizes and parallelism

Grain Size	**Parallelism Level**
Fine Grain	Instruction or Loop Level
Medium Grain	Procedure or SubProgram Level
Coarse Grain	Program Level