

ARCHITECTURAL CLASSIFICATION SCHEMES

Three Computer Architectural Classification Schemes are:

1. Flynn's Classification (1966)

- It is based on the **multiplicity of instruction stream and data streams** in a computer system
 - SISD, SIMD, MISD, MIMD

2. Feng's Classification (1972)

- It is based on **serial versus parallel processing**
 - WSBS, WPBS, WSBP, WPBS

3. Handler's Classification (1977)

- It is determined by the **degree of parallelism and pipelining** in various subsystems levels
 - PCU, ALU, BLC

FLYNN'S CLASSIFICATION

- Flynn's Classification is the most popular taxonomy of computer architecture, proposed by **Michael J. Flynn** in 1966 based on **number of instructions and data**
- It is based on the notion of **a stream of information**. Two types of information flow into a processor: **instructions** and **data**.
- The **Instruction Stream** is defined as the *sequence of instructions* executed by the processing unit
- The **Data Stream** is defined as *the sequence of data* including inputs, partial, or temporary results, called by the instruction stream

TYPES OF FLYNN'S TAXONOMY

- According to Flynn's classification, either of the **instruction or data streams can be single or multiple**. The Computer architecture can be classified into four categories:
 - ▣ Single-instruction stream Single-data Streams (SISD)
 - ▣ Single-instruction stream Multiple-data streams (SIMD)
 - ▣ Multiple-instruction stream Single-data streams (MISD)
 - ▣ Multiple-instruction stream Multiple-data streams (MIMD)

Instruction Streams

one

many

Data Streams

one

SISD

traditional von
Neumann single
CPU computer

MISD

May be pipelined
Computers

SIMD

Vector processors
fine grained data
Parallel computers

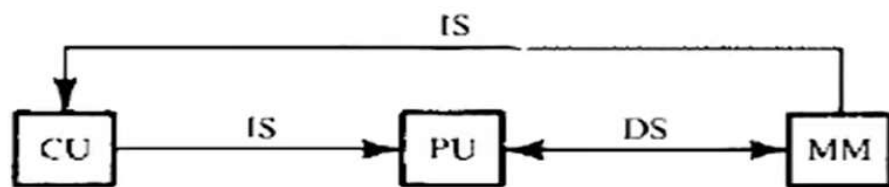
MIMD

Multi computers
Multiprocessors

many

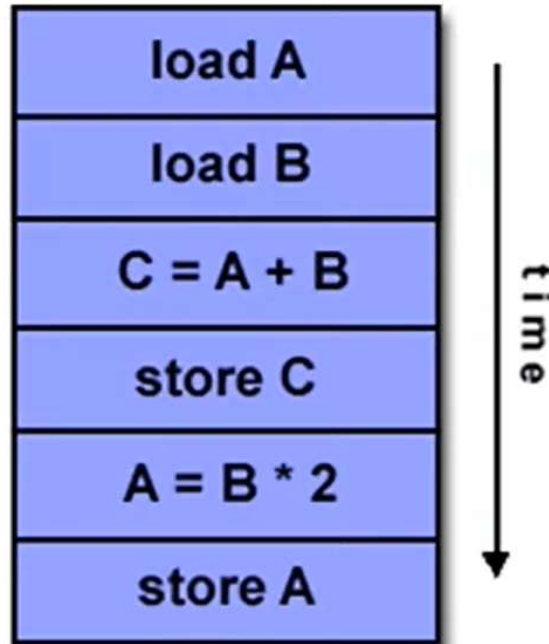
SISD Computers

- ❑ An 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 systems.
- ❑ It is a serial (non-parallel) computer
- ❑ Instructions are executed sequentially, but may be overlapped in their execution stages (Pipelining). Most SISD uni-processor systems are pipelined
- ❑ SISD computers may have more than one functional units, all under the supervision of control unit.
- ❑ **Examples:** Most PC's, single CPU workstations, minicomputers, mainframes. CDC-6600, VAX 11, IBM 7001 are SISD computers



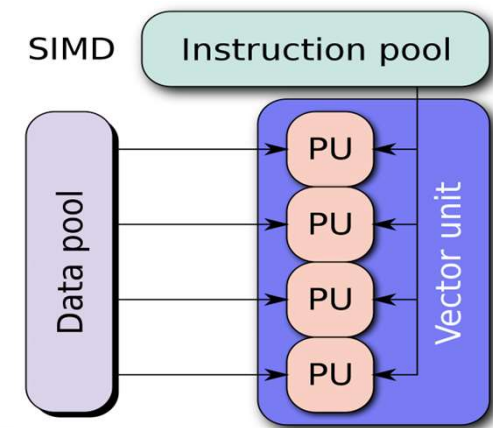
SISD Computers.....

Single Instruction, Single Data (SISD)



SIMD Computers

- ❑ An SIMD system is capable of executing the same instruction on all the CPUs but operating on different data streams.
- ❑ **Single instruction:** All processing units 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 sets simultaneously
- ❑ Well suited to scientific computing since they involve lots of vector and matrix operations.
- ❑ Example: Array Processor sand Vector Pipelines
 - ❑ Array processors: ILLIAC-IV, MPP
 - ❑ Vector Pipelines: IBM 9000, Cray X-MP, Y-MP & C90



Scalar Operation

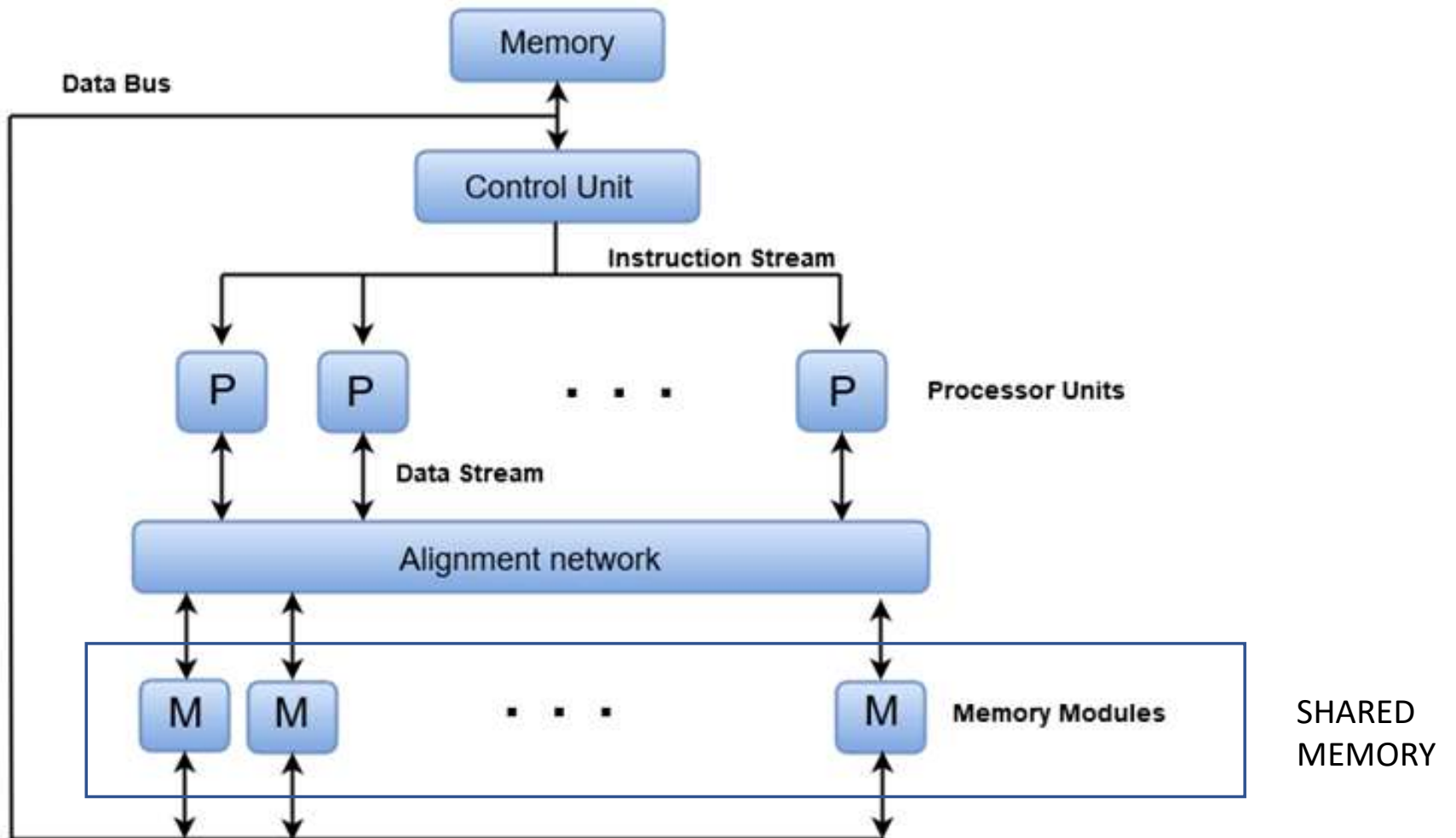
$$\begin{array}{rcl} A_x & + & B_x = C_x \\ A_y & + & B_y = C_y \\ A_z & + & B_z = C_z \\ A_w & + & B_w = C_w \end{array}$$

SIMD Operation of Vector Length 4

$$\begin{array}{c} A_x \\ A_y \\ A_z \\ A_w \end{array} + \begin{array}{c} B_x \\ B_y \\ B_z \\ B_w \end{array} = \begin{array}{c} C_x \\ C_y \\ C_z \\ C_w \end{array}$$

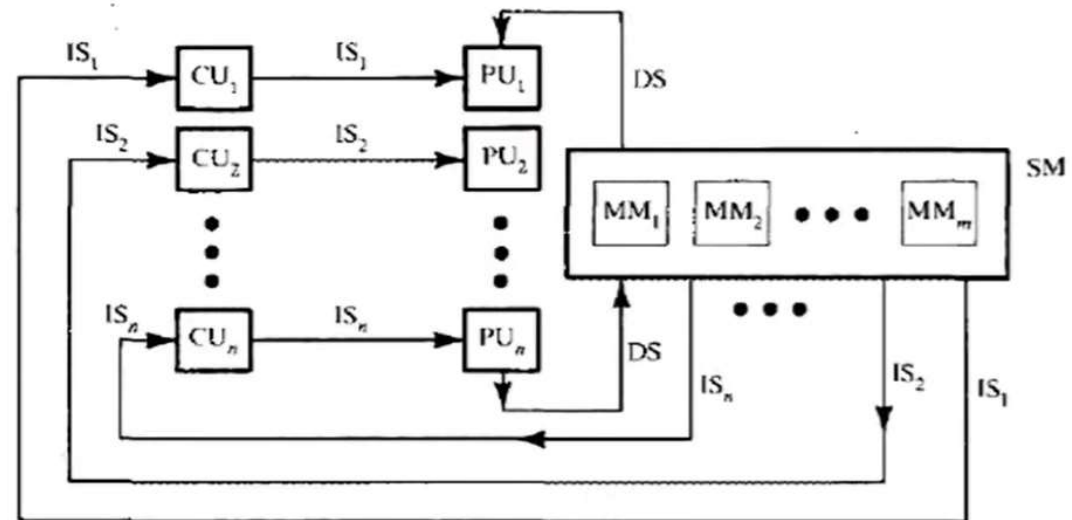
Intel® Architecture currently has SIMD operations of vector length 4, 8, 16

SIMD:



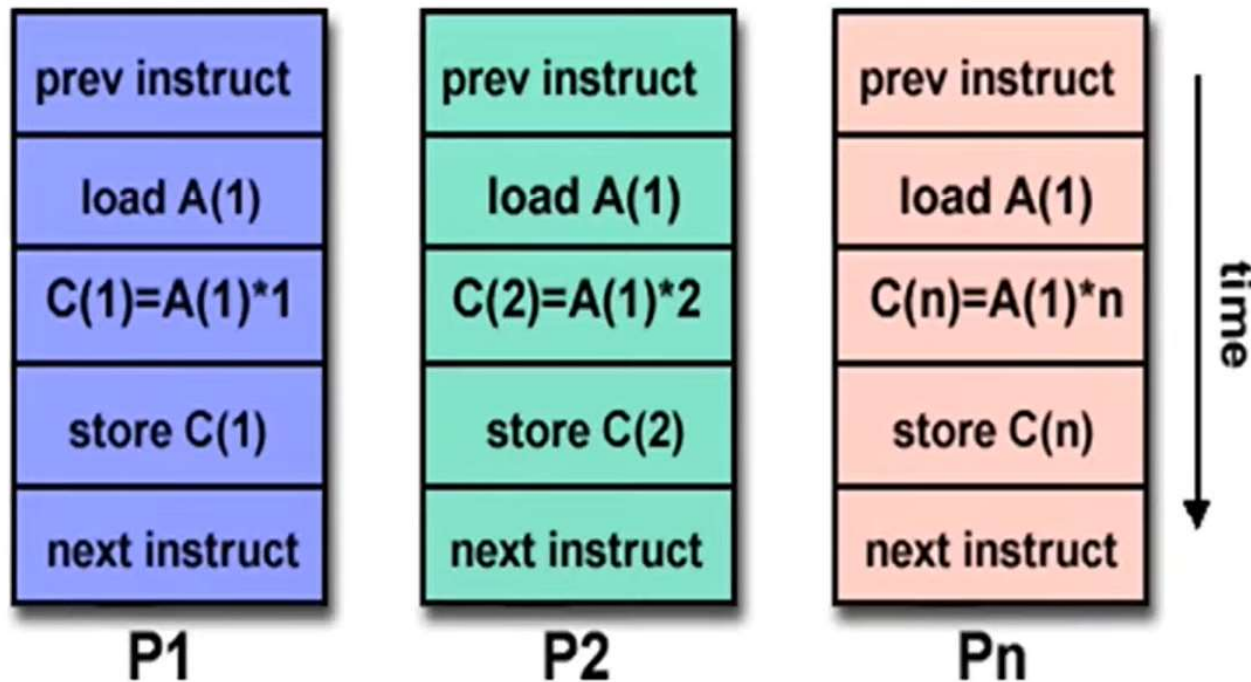
MISD Computers

- ❑ In MISD, multiple instructions operate on single data stream
- ❑ An MISD computing system is capable of executing different instructions on different PEs but all of them operating on the same dataset
- ❑ **Single instruction** : All processing units execute the same instruction at any given clock cycle
- ❑ **Multiple data**: Each processing unit can operate on a different 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 Arrays



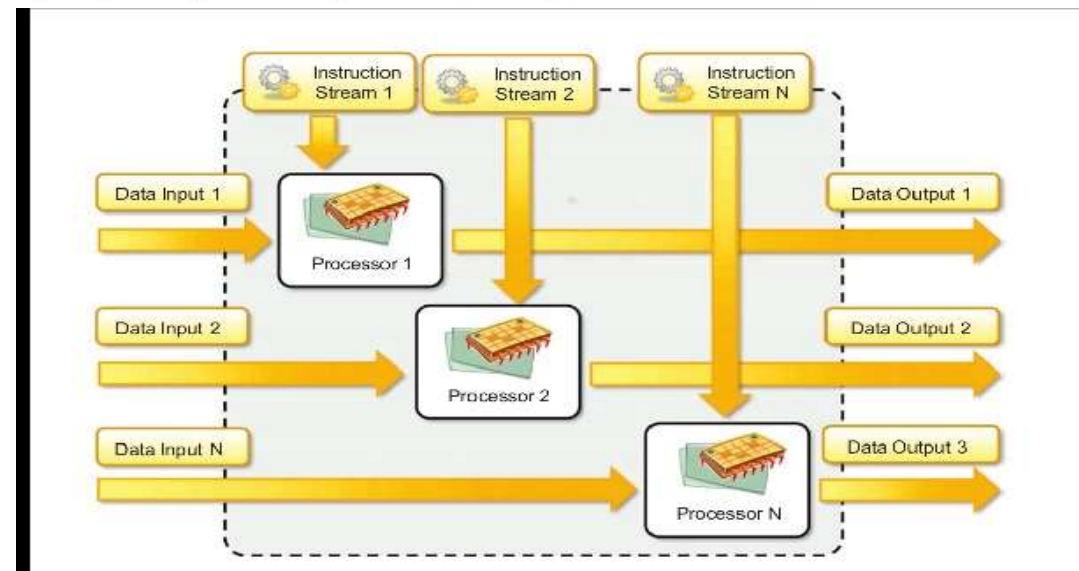
MISD Computers...

Multiple Instruction, Single Data (MISD)



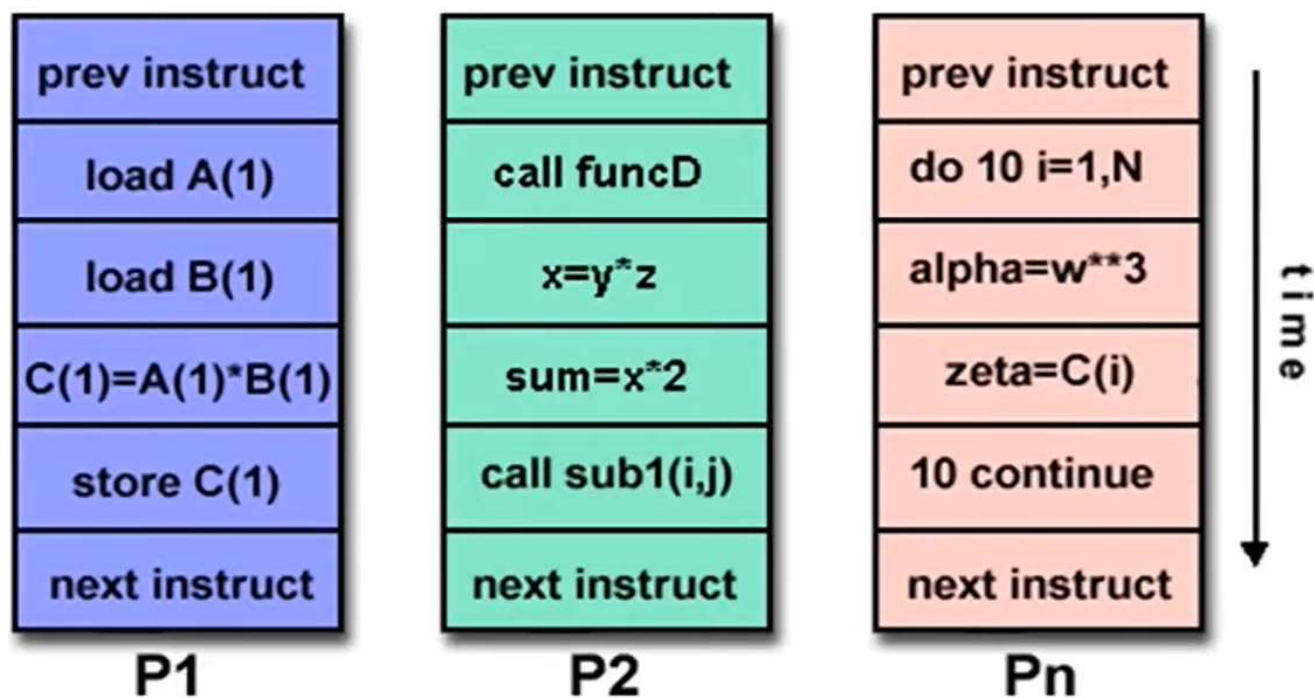
MIMD Computers

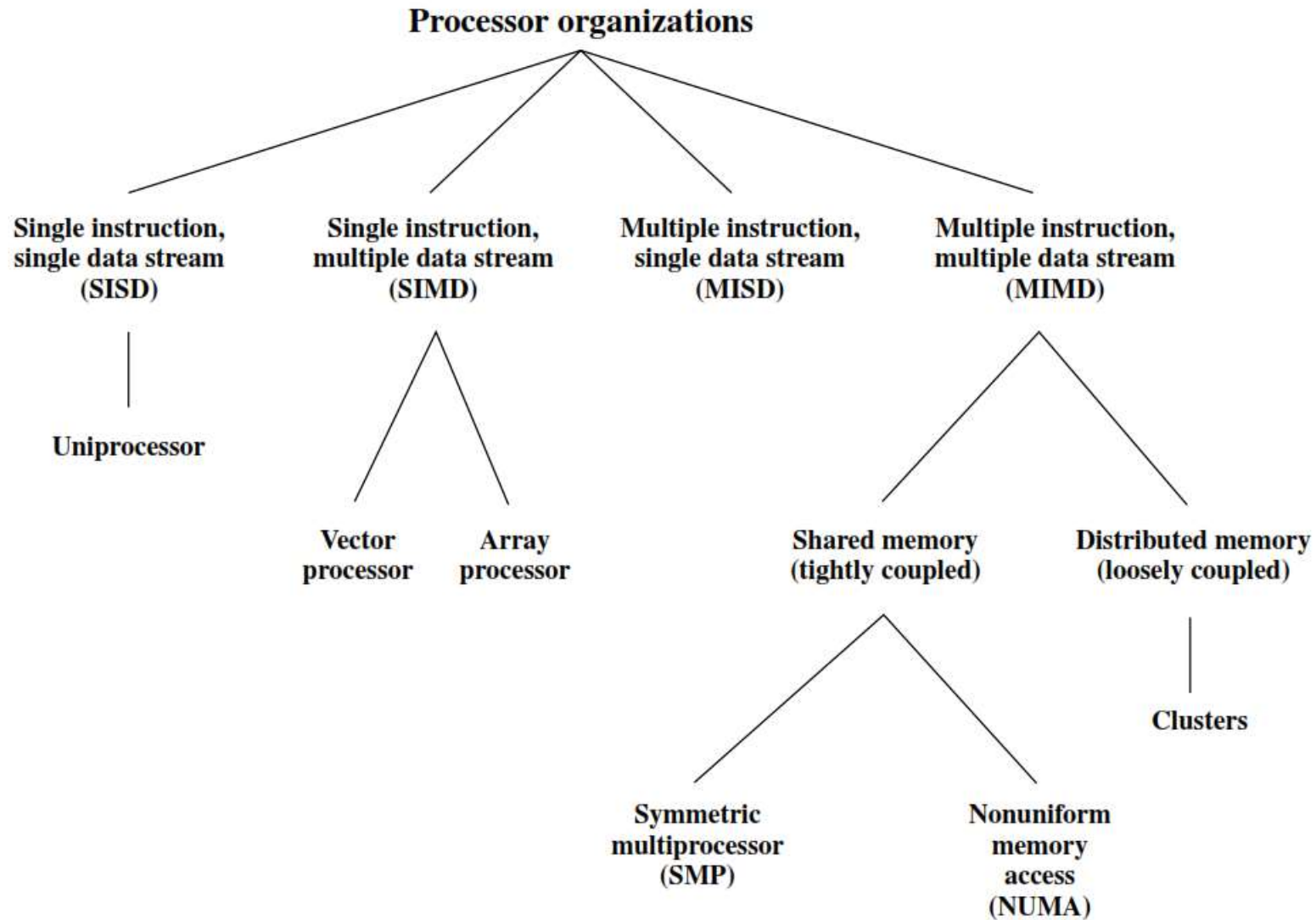
- ❑ An MIMD system is capable of executing multiple instructions on multiple data sets
- ❑ **Multiple Instruction:** Every processor may be executing a different instruction stream
- ❑ **Multiple Data:** Every processor may be working with a different data stream
- ❑ MIMD systems are parallel computers capable of processing several programs simultaneously
- ❑ Can be categorized as loosely coupled or tightly coupled depending on sharing of data and control
- ❑ Example:
 - ❑ most current supercomputers, networked parallel compute "grids" and multi-processor SMP computers - including some types of PCs
 - ❑ IBM-370, Cray-2, Cray X-MP, C.mmp, UNIVAC-1100/80



MIMD Computers...

Multiple Instruction, Multiple Data (MIMD)





A Taxonomy of Parallel Processor Architectures

Can Parallelism be applied to Single Processor based System??

A **Superscalar Processor** is a CPU that implements a form of parallelism called **instruction-level parallelism** within a single processor.

In contrast to a scalar processor, which can execute at most one single instruction per clock cycle, a superscalar processor can execute more than one instruction during a clock cycle

Superscalar processor

How it increases the overall throughput??

By simultaneously dispatching multiple instructions to different execution units on the processor. It therefore allows more throughput (the number of instructions that can be executed in a unit of time) than would otherwise be possible at a given clock rate. Each execution unit is not a separate processor (or a core if the processor is a multi-core processor), but an execution resource within a single CPU such as an arithmetic logic unit.

Superscalar processor

The superscalar technique is traditionally associated with several identifying characteristics (within a given CPU):

1. Instructions are issued from a sequential instruction stream
2. The CPU dynamically checks for data dependencies between instructions at run time.
3. The CPU can execute multiple instructions per clock cycle

Superscalar processor

The simplest processors are scalar processors. Each instruction executed by a scalar processor typically manipulates one or two data items at a time. By contrast, each instruction executed by a vector processor operates simultaneously on many data items. An analogy is the difference between scalar and vector arithmetic. A superscalar processor is a mixture of the two. Each instruction processes one data item, but there are multiple execution units within each CPU thus multiple instructions can be processing separate data items concurrently.

Superscalar CPU design emphasizes improving the instruction dispatcher accuracy and allowing it to always keep the multiple execution units in use.

Instruction-level parallelism

Instruction-level parallelism (ILP) is the parallel or simultaneous execution of a sequence of instructions in a computer program.

There are two approaches to instruction-level parallelism: hardware and software.

Hardware level works upon dynamic parallelism, whereas the software level works on static parallelism.

Dynamic parallelism means the processor decides at run time which instructions to execute in parallel, whereas static parallelism means the compiler decides which instructions to execute in parallel.

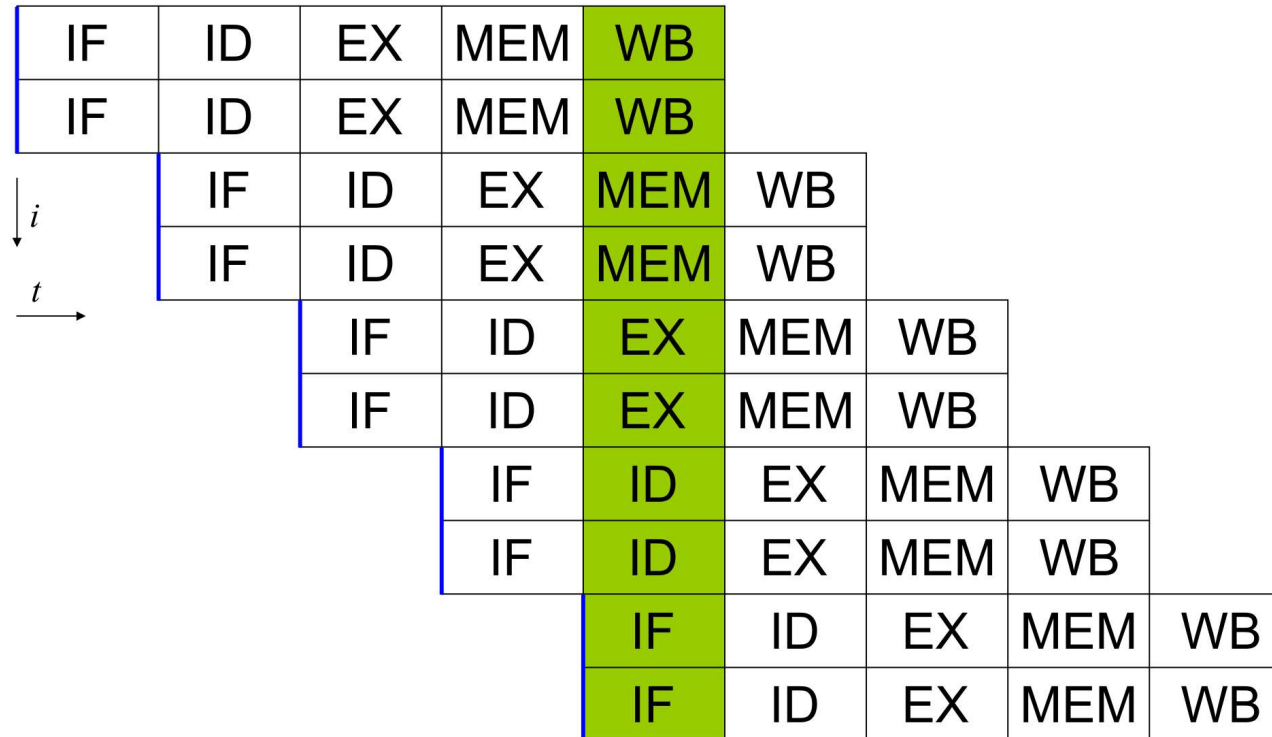
Instruction-level parallelism

Consider the following program:

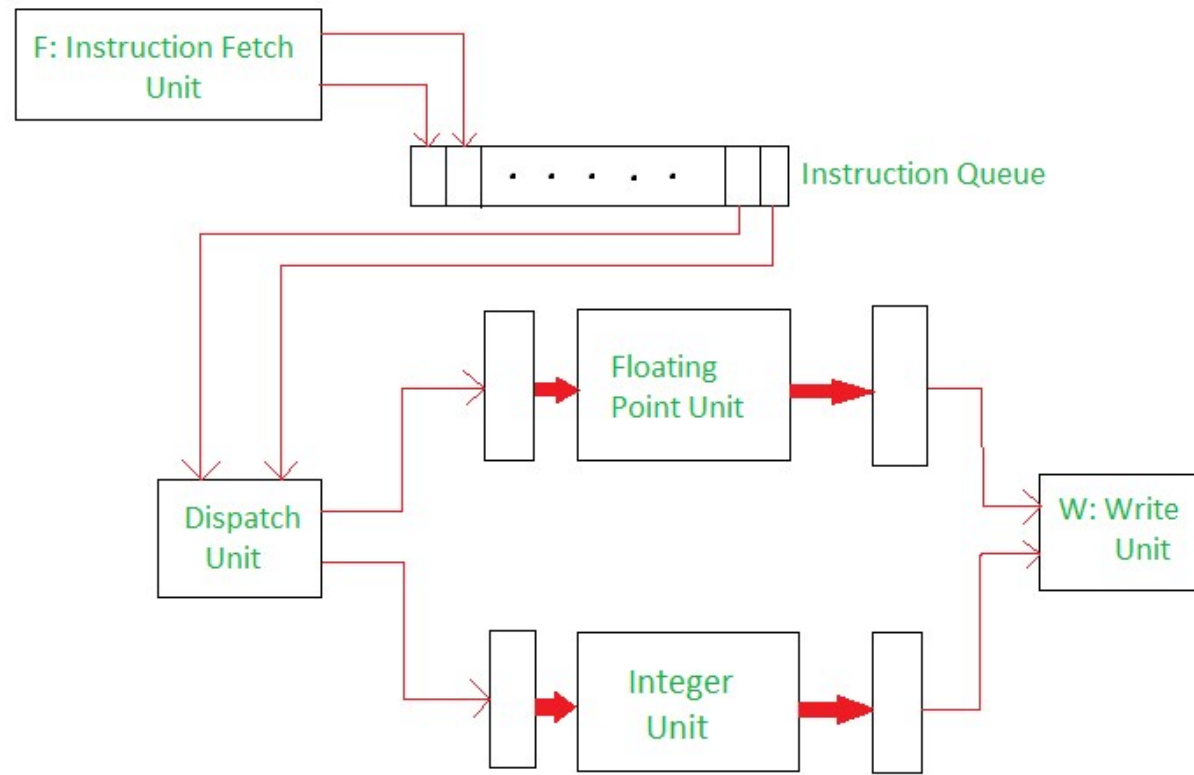
```
1  e = a + b
2  f = c + d
3  m = e * f
```

Operation 3 depends on the results of operations 1 and 2, so it cannot be calculated until both of them are completed. However, operations 1 and 2 do not depend on any other operation, so they can be calculated simultaneously. If we assume that each operation can be completed in one unit of time then these three instructions can be completed in a total of two units of time, giving an ILP of 3/2.

Instruction-level parallelism

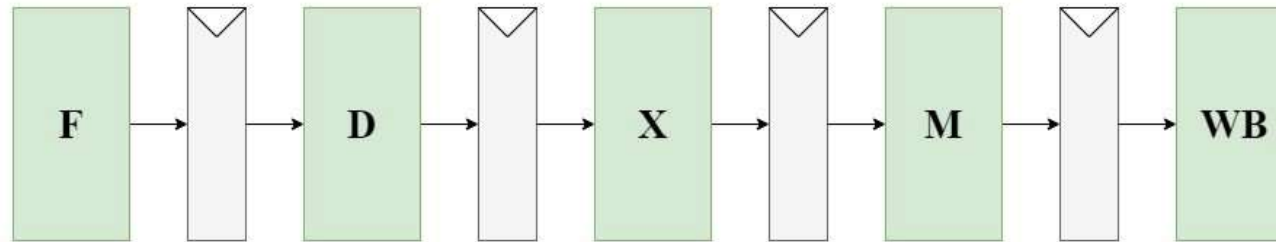


Simple superscalar pipeline. By fetching and dispatching two instructions at a time, a maximum of two instructions per cycle can be completed. (IF = instruction fetch, ID = instruction decode, EX = execute, MEM = memory access, WB = register write-back, i = instruction number, t = clock cycle [i.e. time])

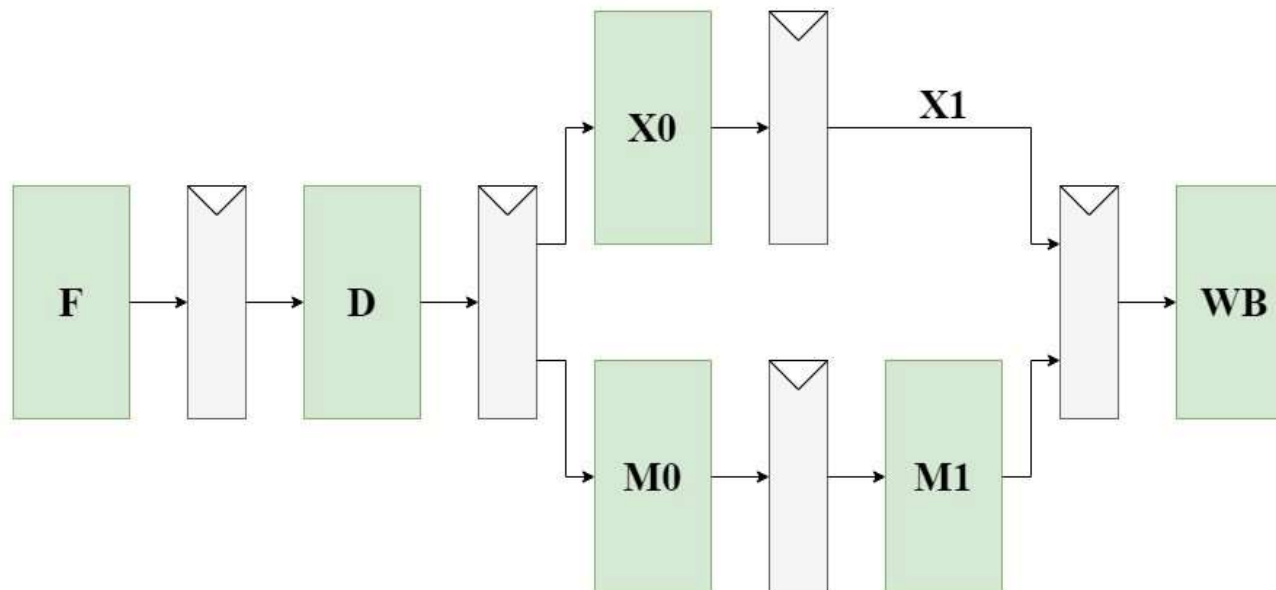


Processor with Two Execution Units

Scalar



Superscalar



What is the VLIW Architecture?

Very long instruction word (VLIW) refers to instruction set architectures designed to exploit instruction level parallelism (ILP). Whereas conventional central processing units (CPU, processor) mostly allow programs to specify instructions to execute in sequence only, a VLIW processor allows programs to explicitly specify instructions to execute in parallel.

The traditional means to improve performance in processors include dividing instructions into substeps so the instructions can be executed partly at the same time (termed pipelining), dispatching individual instructions to be executed independently, in different parts of the processor (superscalar architectures), and even executing instructions in an order different from the program (out-of-order execution). These methods all complicate hardware (larger circuits, higher cost and energy use) because the processor must make all of the decisions internally for these methods to work. In contrast, the VLIW method depends on the programs providing all the decisions regarding which instructions to execute simultaneously and how to resolve conflicts. As a practical matter, this means that the compiler (software used to create the final programs) becomes far more complex, but the hardware is simpler than in many other means of parallelism.

VLIW Architecture

In superscalar designs, the number of execution units is invisible to the instruction set. Each instruction encodes one operation only. For most superscalar designs, the instruction width is 32 bits or fewer.

In contrast, one VLIW instruction encodes multiple operations, at least one operation for each execution unit of a device. For example, if a VLIW device has five execution units, then a VLIW instruction for the device has five operation fields, each field specifying what operation should be done on that corresponding execution unit. To accommodate these operation fields, VLIW instructions are usually at least 64 bits wide, and far wider on some architectures.

For example, the following is an instruction for the Super Harvard Architecture Single-Chip Computer (SHARC). In one cycle, it does a floating-point multiply, a floating-point add, and two autoincrement loads. All of this fits in one 48-bit instruction:

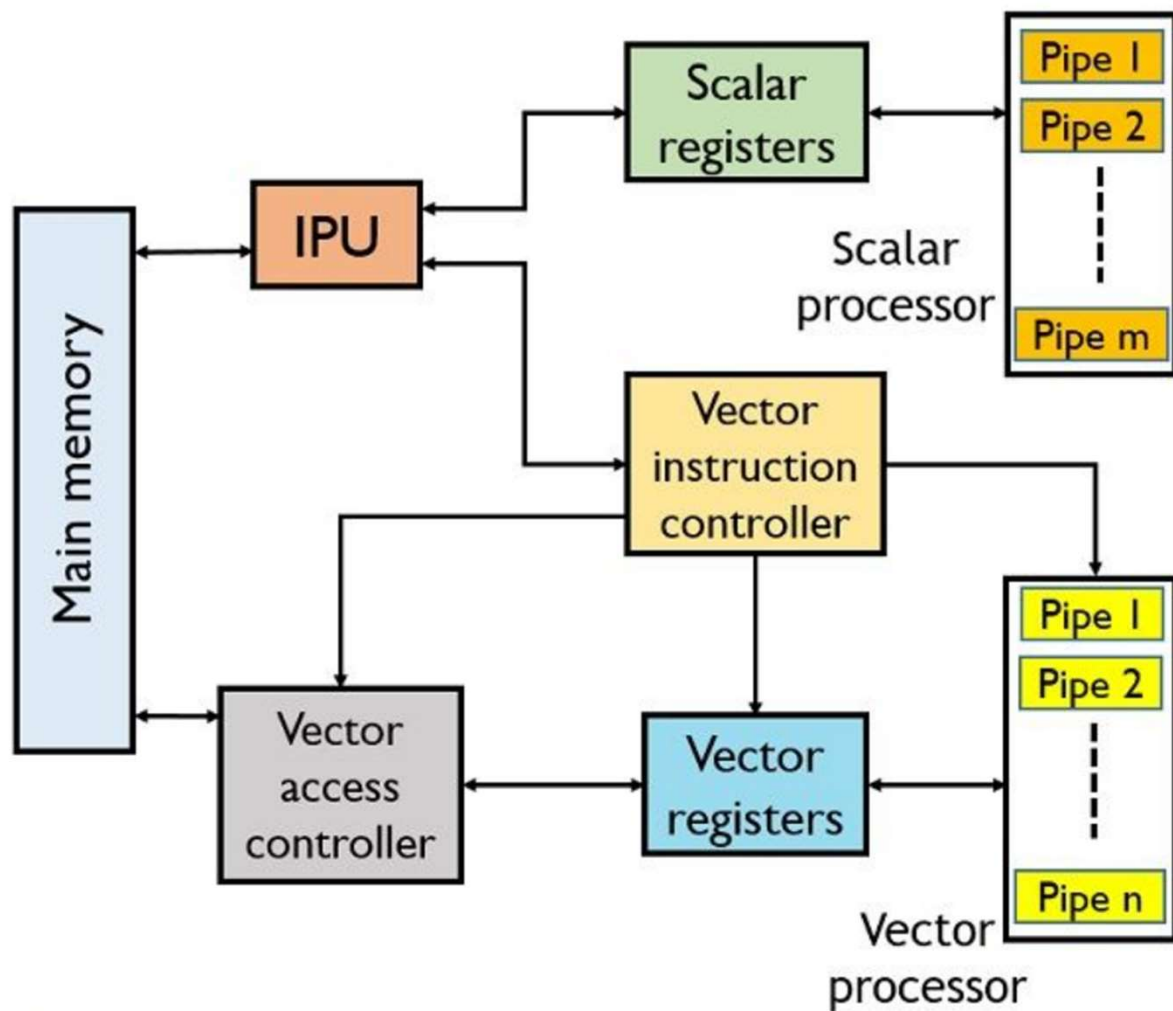
```
f12 = f0 * f4, f8 = f8 + f12, f0 = dm(i0, m3), f4 = pm(i8, m9);
```

Vector processor

A vector processor or array processor is a central processing unit (CPU) that implements an instruction set where its instructions are designed to operate efficiently and effectively on large one-dimensional arrays of data called vectors.

It holds a single control unit but has multiple execution units that perform the same operation on different data elements of the vector.

Unlike scalar processors that operate on only a single pair of data, a vector processor operates on multiple pair of data. However, one can convert a scalar code into vector code. This conversion process is known as vectorization. So, we can say vector processing allows operation on multiple data elements by the help of single instruction.



Functional Diagram of Vector Computer

Vector processor

Once the instruction is fetched then IPU determines either the fetched instruction is scalar or vector in nature. If it is scalar in nature, then the instruction is transferred to the scalar register and then further scalar processing is performed.

While, when the instruction is a vector in nature then it is fed to the vector instruction controller. This vector instruction controller first decodes the vector instruction then accordingly determines the address of the vector operand present in the memory.

Then it gives a signal to the vector access controller about the demand of the respective operand. This vector access controller then fetches the desired operand from the memory. Once the operand is fetched then it is provided to the instruction register so that it can be processed at the vector processor.

MIMD

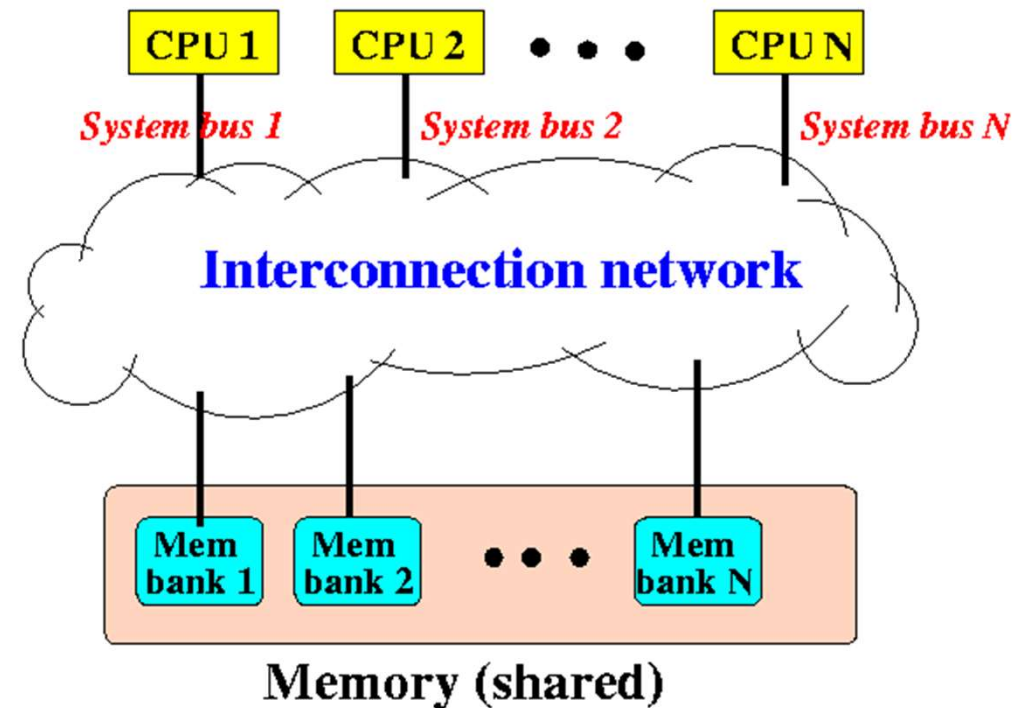
```
graph TD; A[MIMD] --- B[Multiple-instruction multiple-data streams]; A --- C[parallel architectures: made of multiple processors and multiple memory modules linked via some interconnection network.]; A --- D[two broad types: shared memory or distributed memory (message passing).];
```

Multiple-instruction multiple-data streams

parallel architectures: made of multiple processors and multiple memory modules linked via some interconnection network.

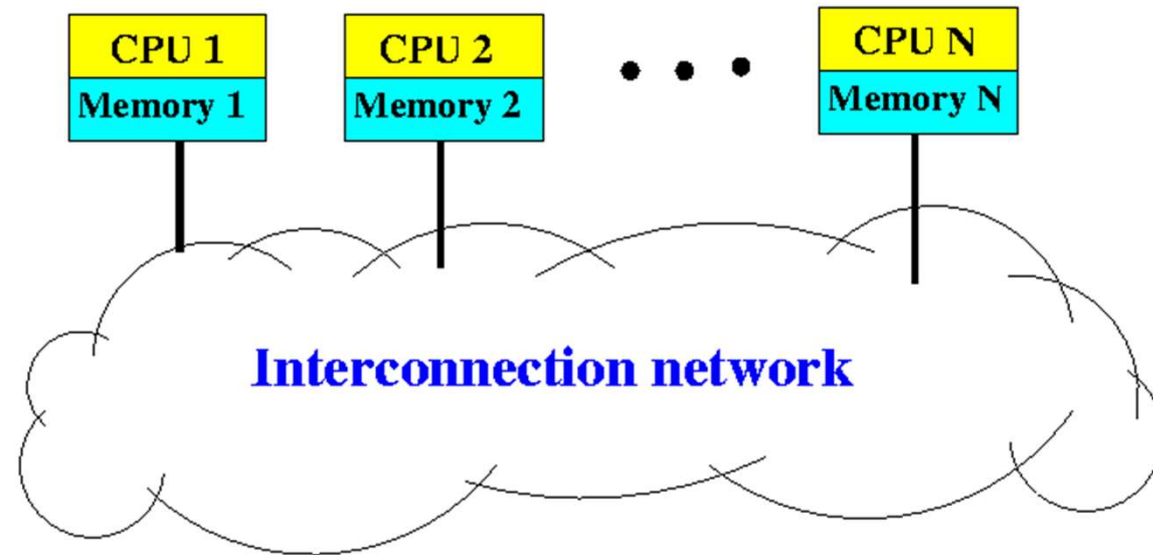
two broad types: shared memory or distributed memory (message passing).

Shared Memory



A shared-memory multiprocessor is an architecture consisting of several processors, all of which have direct (hardware) access to all the main memory in the system. This permits any of the system processors to access data that any of the other processors has created or will use.

Distributed Memory



In distributed memory MIMD machines, each processor has its own individual memory location. Each processor has no direct knowledge about other processor's memory. **For data to be shared, it must be passed from one processor to another as a message.**

- The data transferred in **shared-memory MIMD** computer is:
 - a **memory address** from a **CPU** to a **memory module** or
 - a **word (stored in memory)** from a **memory module** to a **CPU**

The amount of data transfer is **fixed size**

- The data transferred in **Message-passing MIMD** computer is:
 - a **message** from a **computer** to another **computer**

The amount of data transfer is **variable size**

Shared memory MIMD

```
graph LR; A[Shared memory MIMD] --> B[Uniform Memory Access (UMA) also known as Symmetric Multiprocessor (SMP)]; A --> C[Non Uniform Memory Access (NUMA)]; B --> D[Single Memory Controller is Used]; C --> E[Multiple Memory Controllers are used];
```

Uniform Memory Access (UMA) also known as Symmetric Multiprocessor (SMP)

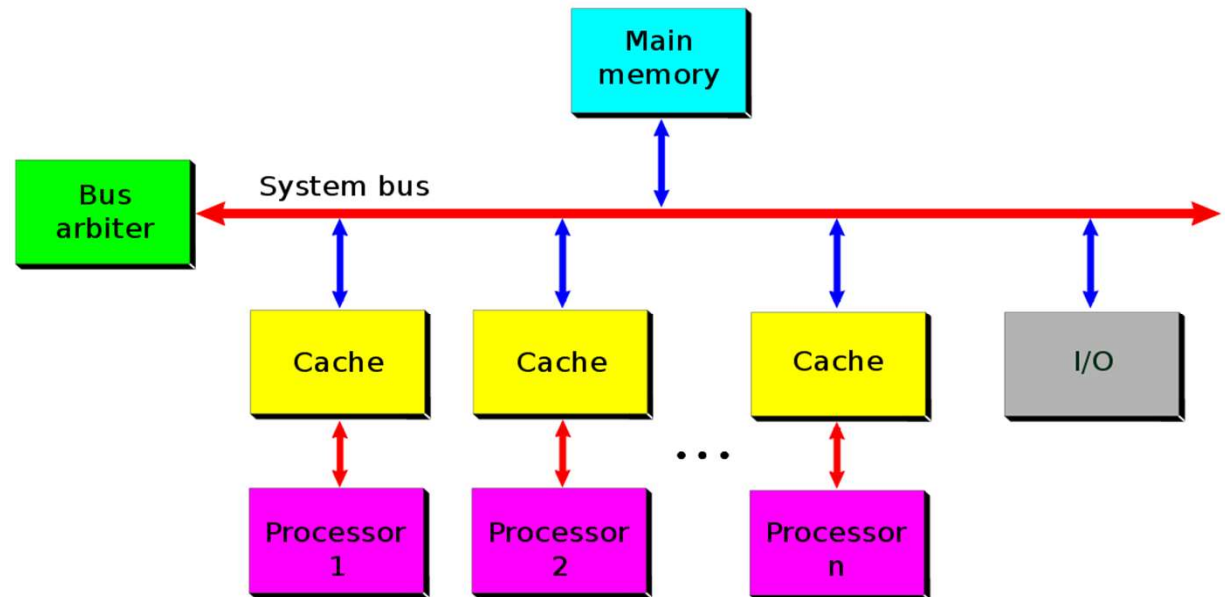
Single Memory Controller is Used

Non Uniform Memory Access (NUMA)

Multiple Memory Controllers are used

Symmetric Multiprocessing

In symmetric multiprocessing, multiple processors share a common memory and operating system. All of these processors work in tandem to execute processes. The operating system treats all the processors equally, and no processor is reserved for special purposes.



Symmetric Multiprocessing

- Symmetric multiprocessing is also known as tightly coupled multiprocessing as all the CPU's are connected at the bus level and have access to a shared memory.
- All the parallel processors in symmetric multiprocessing have their private cache memory to decrease system bus traffic and also reduce the data access time.
- Symmetric multiprocessing systems allow a processor to execute any process no matter where its data is located in memory. The only stipulation is that a process should not be executing on two or more processors at the same time.
- In general, the symmetric multiprocessing system does not exceed 16 processors as this amount can be comfortably handled by the operating system.

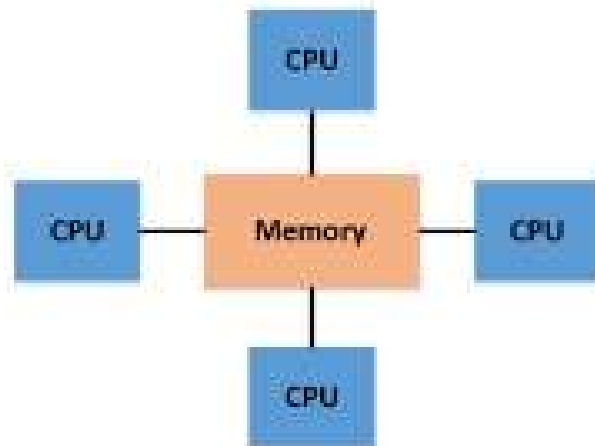
Symmetric Multiprocessing

Uses of Symmetric Multiprocessing

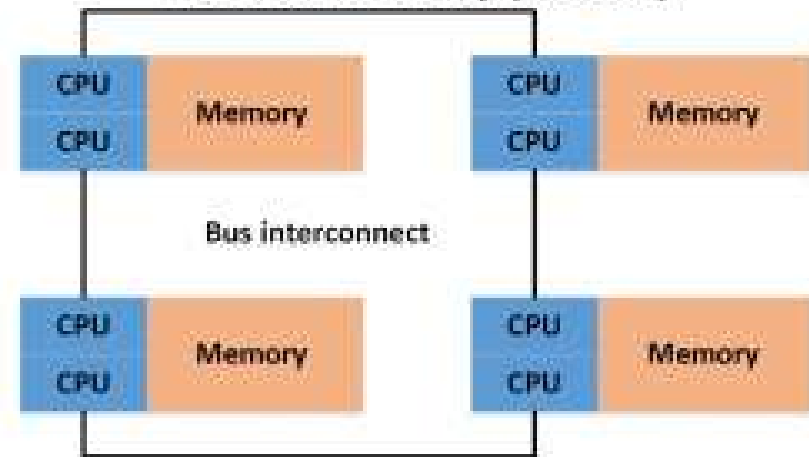
Some of the uses of symmetric multiprocessing are as follows –

- Symmetric multiprocessing is useful for time sharing systems as these have multiple processes running in parallel. So, these processes can be scheduled on parallel processors using symmetric multiprocessing.
- Symmetric processing is not that useful in personal computers unless multithreaded programming is taken into account. The multiple threads can be scheduled on the parallel processors.
- Time sharing systems that use multithreading programming can also make use of symmetric multiprogramming.

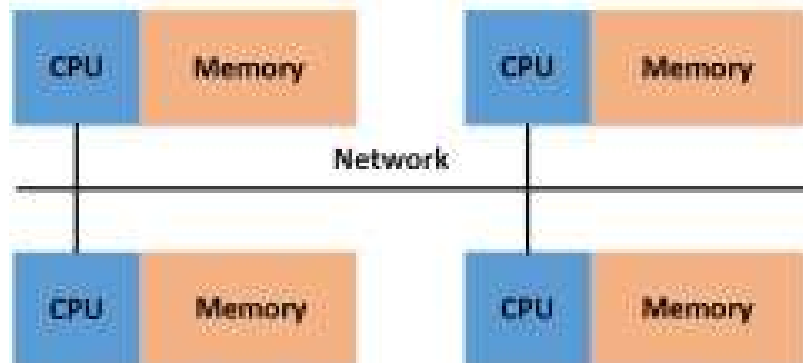
Shared memory (UMA)



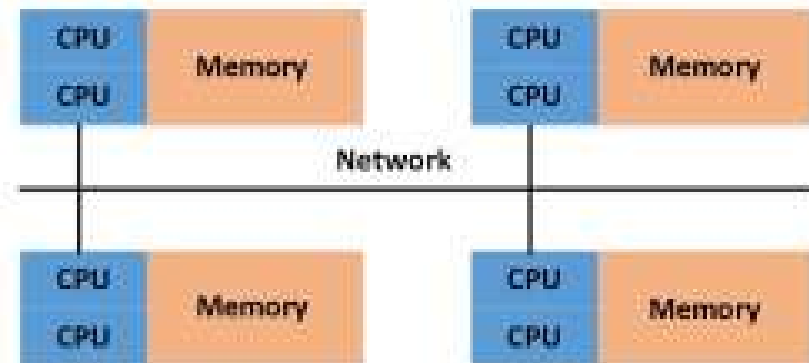
Shared memory (NUMA)



Distributed memory



Hybrid memory





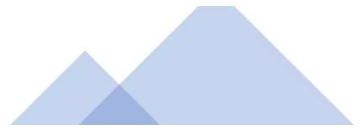
Advantages of Symmetric Multiprocessing


Some advantages of symmetric multiprocessing are –

- The throughput of the system is increased in symmetric multiprocessing. As there are multiple processors, more processes are executed.
- Symmetric multiprocessing systems are much more reliable than single processor systems. Even if a processor fails, the system still endures. Only its efficiency is decreased a little.

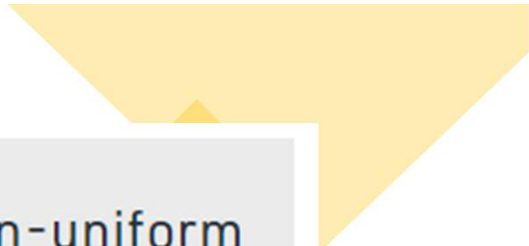
Disadvantages of Symmetric Multiprocessing

Some disadvantages of symmetric multiprocessing are –

- The operating system handles all the processors in symmetric multiprocessing system. This leads to a complicated operating system that is difficult to design and manage.
 - All the processors in symmetric multiprocessing system are connected to the same main memory. So a large main memory is required to accommodate all these processors.
- 



1. UMA stands for Uniform Memory Access.




NUMA stands for Non-uniform Memory Access.

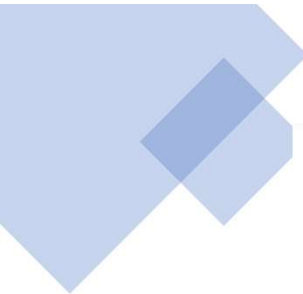
2. In Uniform Memory Access, Single memory controller is used.

In Non-uniform Memory Access, Different memory controller is used.

3. Uniform Memory Access is slower than non-uniform Memory Access.


Non-uniform Memory Access is faster than uniform Memory Access.





4. Uniform Memory Access has limited bandwidth.

Non-uniform Memory Access has more bandwidth than uniform Memory Access.




5. Uniform Memory Access is applicable for general purpose applications and time-sharing applications.

Non-uniform Memory Access is applicable for real-time applications and time-critical applications.

6. In uniform Memory Access, memory access time is balanced or equal.

In non-uniform Memory Access, memory access time is not equal.



7. There are 3 types of buses used in uniform Memory Access which are: Single, Multiple and Crossbar.

While in non-uniform Memory Access, There are 2 types of buses used which are: Tree and hierarchical.