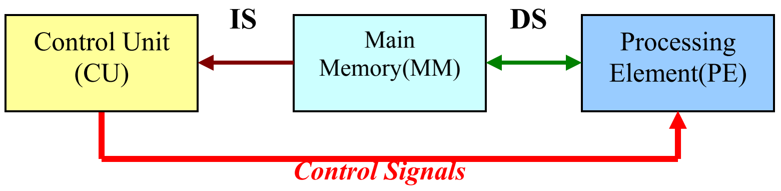
Flynn's Taxonomy

The taxonomy of computer systems proposed by M. J. Flynn in 1966 has remained the focal point in the field. This is based on the notion of instruction and data streams that can be simultaneously manipulated by the machine. A stream is just a sequence of items (instruction or data).

1. SISD

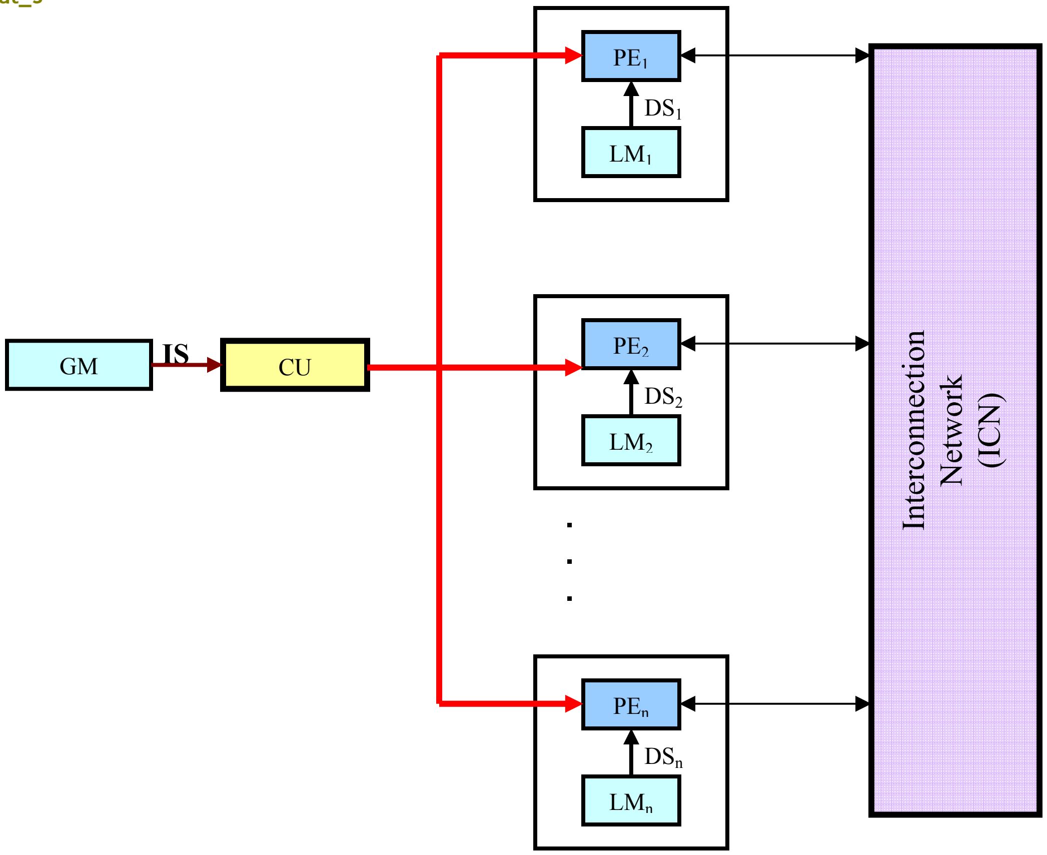
- Single Instruction Stream, Single Data Stream
- Conventional sequential computer (von Neumann architecture) i.e. uniprocessor
- Single control unit (CU) fetches single Instruction Stream (IS) from memory
- The CU then generates appropriate control signals to direct single processing element (PE) to operate on single Data Stream (DS) i.e. one operation at a time



  - In parallel processing literature the term PE is more common for ALU. A PE, unlike CU, has no instruction decoding capability.

2. SIMD

- Single Instruction Stream, Multiple Data Stream
- Single CU and multiple PEs
- CU fetches an instruction from memory and after decoding, broadcasts control signals to all PEs
- A global clock drives all PEs
- That is, at any given time, all PEs are **synchronously** executing the **same instruction** but on different sets of data; hence the name **SIMD**
- Analogy
  A drill sergeant gives orders to a platoon of soldiers. The sergeant barks out a single order, "Attention", and all soldiers execute it in parallel, each one using his or her own arms, legs, and body. The next order, say "Present Arms", is not issued until the sergeant is satisfied that the previous one has been completed. Again, the soldiers all carry out this next order simultaneously, with each one presenting his or her own weapon.
- Control (e.g. branch) and scalar instructions are executed by the CU
- Arithmetic and logic instructions are executed by PEs.
- Activation/deactivation of PEs is done by CU
- ICN for PE-PE communication
- SIMD exploits data / spatial/synchronous parallelism



- The synchronous model of parallelism was a good match with the types of problems which occurred most often in the early days of parallel processing, namely large vector and matrix computations. In these types of problems, we typically carry out exactly the same sequence of operations on each element in the vector or matrix. For example, in the array/vector multiplication operation **Y = A * X** (where X, Y are N-element vectors and A is an N x N matrix), the identical sequence of operations are done for each element of the result vector Y:

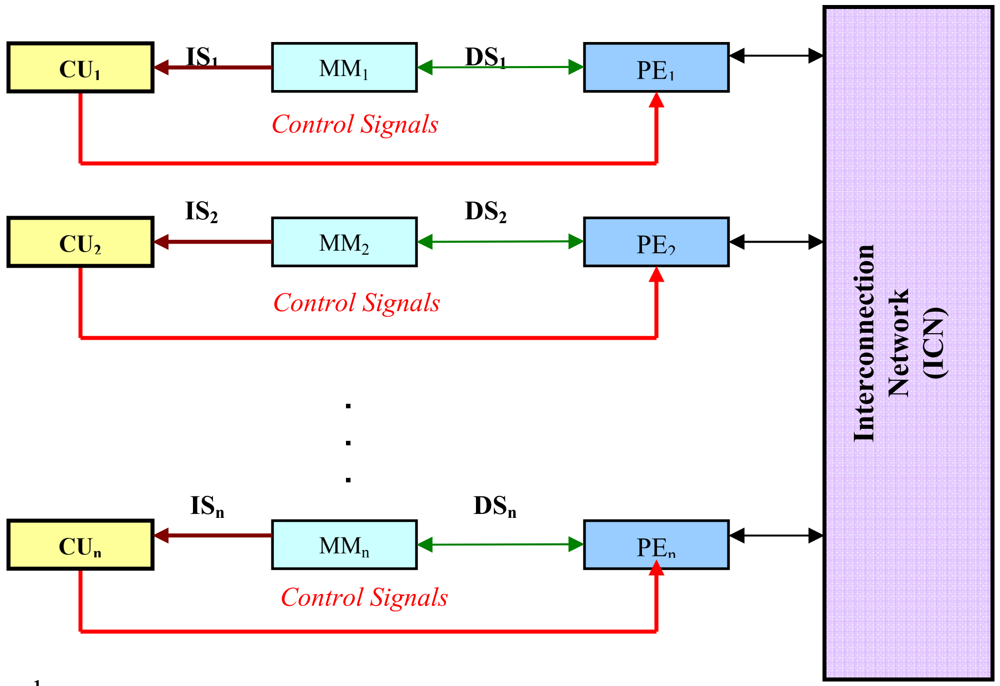
$$y_i = \sum_{j=1}^{N} a_{ij} * x_j \text{ for all i = 1, 2, 3, . . ., N}$$

  If we have N processing elements, P_1, ... P_N, we could give P_i a copy of row i of matrix A and vector X and have them all synchronously execute this summation operation on their local data items. This produces the result vector Y in N time steps rather than the N^2 required by a sequential implementation. Many other vector and array operations are also easily implemented using this approach.
- SIMD machines are also called array /vector processor as they are employed for processing complex operations on large arrays or vectors
- A SIMD is a special purpose machine.
- ILLIAC-IV is an example of SIMD of historic interest
- Thinking Machine Corporation's CM -1 and 2 (CM: Connection Machine) are commercially available SIMDs

- More recently, variants of this concept have found use in co-processing units such as the MMX units in Intel processors and DSP chips such as the SHARC.
- The Intel Pentium 3 with its SSE (Streaming SIMD Extensions) and the Pentium 4 with SSE2 provide a number of instructions that execute a single instruction on multiple data items.
- These architectural enhancements rely on the highly structured (regular) nature of the underlying computations, for example, in image processing and graphics to deliver improved performance

3. MIMD

- Multiple Instruction Stream, Multiple Data Stream
- Multiple processors each executing its own instruction stream to process the data stream allocated to it.



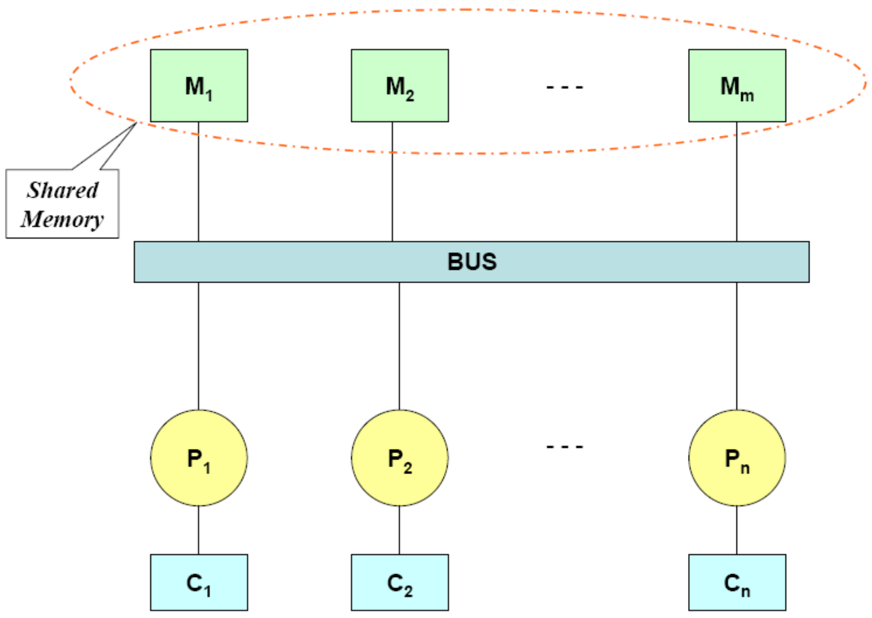
- <u>Analogy</u>
  A good analogy to asynchronous parallelism is the way that different craftspersons work to build a house. Carpenters, masons, electricians, and plumbers all work independently on their own tasks and at their own rate. Occasionally, they need to communicate with each other to find out if a specific task has been completed or to provide someone else with information about what you are doing.
- A MIMD is a true multiprocessor
- In contrast to SIMD, a MIMD is a general-purpose machine.
- MIMD exploits asynchronous parallelism
- ICN is provided for processor-processor and processor-memory interaction.
- When all the processors in MIMD are running the same program, we call it Single Program Multiple Data (SPMD) computation.
- The SPMD model is widely used by many parallel platforms and requires minimal architectural support.
- Examples of such platforms include the Sun Ultra Servers, multiprocessors PCs, workstation clusters, and IBM SP.

Classification of MIMD Machines

MIMD machines are classified on the basis of how processors communicate with each other. Accordingly, there're two categories:

Shared Memory Multiprocessors

- Single address space shared by all processors
- By single address space we mean that same address generated by two or more processors refers to the same memory element in one of the memory modules.
- Shared Memory multiprocessors are usually connected by a shared BUS

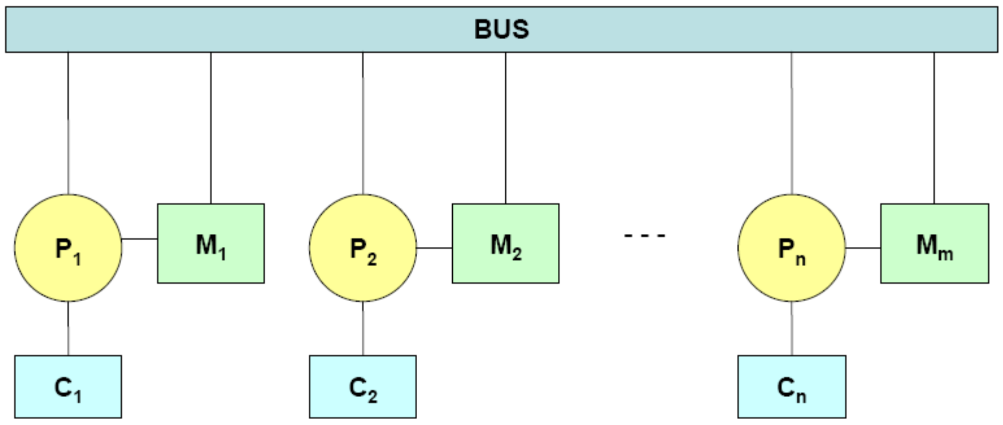


- References generated by processors are first searched in their respective caches. A cache miss necessitates memory access.
- Every memory access requires bus access.
- Once a processor gains bus mastership, it can access any memory module in uniform (i.e. equal) time. Hence, these architectures are called **U**niform **M**emory **A**ccess (UMA) architectures.
- Shared memory architectures are also called **tightly coupled multiprocessors** due to close integration of modules (processors, memory, I/O)
- Shared memory architectures are commonly shipped as **S**ymmetric **M**ulti **P**rocessors (SMP). An SMP is characterized by:
  - Processors of identical capabilities
  - Equal access of resources to all processors
  - OS Kernel can run on any processor
- In contrast, asymmetric multiprocessors are characterized by master-slave relationship, where a master processor assigns job to slave processors.

- Shared Memory multiprocessors suffer performance loss due to bus contention and therefore can't be used for larger number of processors.

Distributed Shared Memory (DSM)

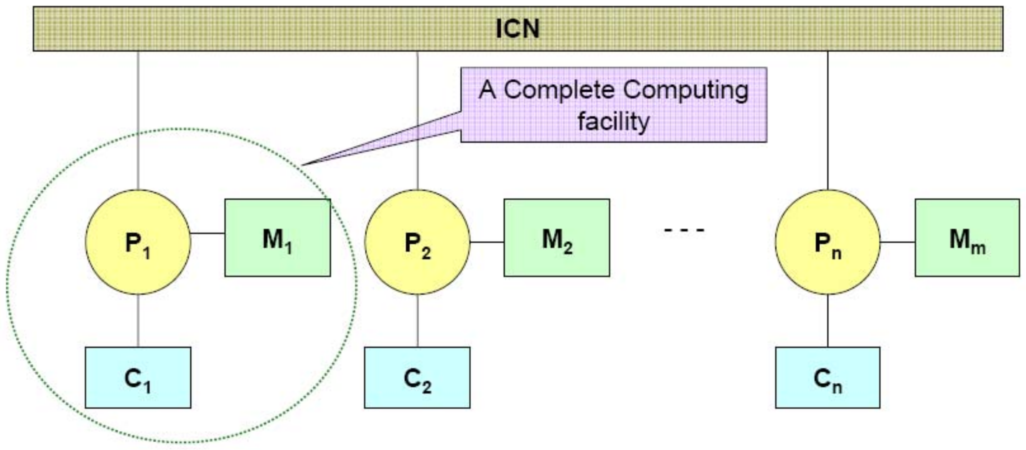
- DSM reduces bus contention problem of shared memory architectures. Again, by shared we mean a single address space shared by all processors.
- However, a memory module is dedicated to each processor as shown in the following diagram.



- As can be seen, a processor P_K can access its memory module M_K without accessing bus. Careful address mapping by OS can reduce the need for accessing memory modules over bus.
- A processor P_K can access its memory module M_K in lesser time than it takes to access any other module M_J ($J \neq K$). Hence, these architectures are called **N**on **U**niform **M**emory **A**ccess (NUMA) architectures.
- Be it shared memory, or DSM, inter-processor communication takes place using shared variables.
- Examples of such machines include the **SGI Origin** and the **Sun Enterprise servers**.

Distributed Memory Multiprocessors

- Separate address space for every processor i.e. same address generated by two or more processors refers to different memory element.



- Each node is a complete computer system (for this reason we also call it multicomputer model in contrast to shared memory and DSM architectures which are commonly referred to as multiprocessor systems)
- Examples of multicomputer models are COW (Cluster of Workstations) and NOW (Network of Workstations).
- No processor is allowed to access memory module of other processor. Hence, these systems are called **NO R**emote **M**emory **A**ccess (NORMA) architectures.
- Distributed memory architectures are also called **loosely coupled multiprocessors** due to not as close integration of modules (processors, memory, I/O) as in shared memory multiprocessors
- Processors communicate with each other using message passing over ICN.

Comparison

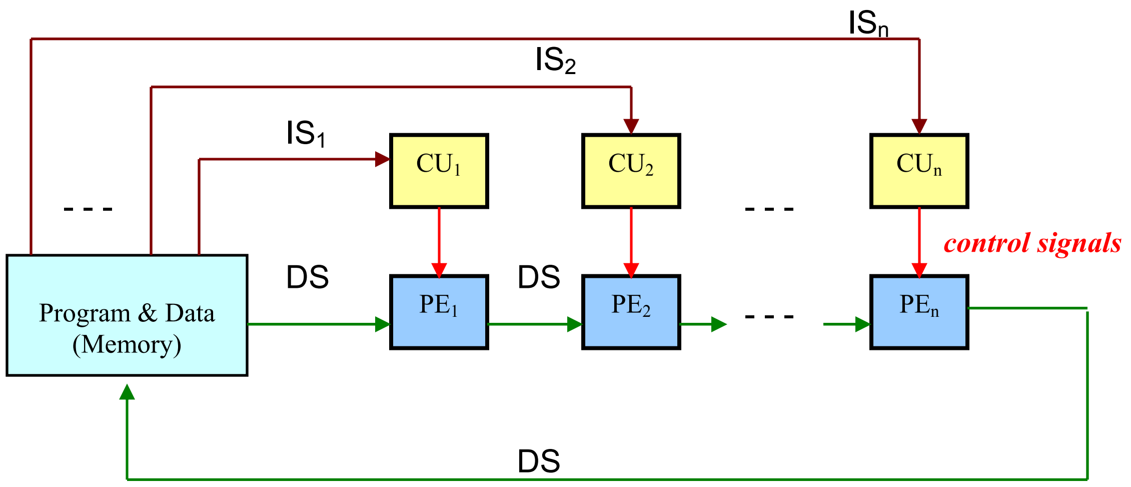
- Programmability
  - Shared memory architectures are easy to program as they provide a single address space as found in uniprocessor environment.
  - Distributed memory machines pose difficulty due to multiple address spaces.
- Reliability
  - NORMA is more reliable than shared memory architectures because a fault in any node remains "local" in most of the cases.
  - In shared memory, any failing module (processor or memory) can cause the whole system to fail due to close integration.
- Scalability
  - This is difficult to add new modules (processors in particular) in shared memory architectures on account of network (bus) contention and above all, due to tight coupling of resources. That is they scale poorly. On the hand, distributed memory architectures offer excellent scalability.

Comparison of SIMD and MIMD

- SIMD computers require less hardware than MIMD computers because they have only one control unit.
- SIMD computers require less memory because only one copy of the program needs to be stored. In contrast, MIMD computers store the program and operating system at each processor.
- However, the relative unpopularity of SIMD processors as general purpose compute engines can be attributed to their specialized hardware architectures, economic factors, design constraints, product life-cycle, and application characteristics.
- In contrast, platforms supporting the SPMD paradigm can be built from inexpensive off-the-shelf components with relatively little effort in a short amount of time.
- SIMD computers require extensive design effort resulting in larger product development times. Since the underlying serial processors change so rapidly, SIMD computers suffer from fast obsolescence. The irregular nature of many applications also makes SIMD architectures less suitable

4. MISD

- Multiple Instruction Stream, Single Data Stream
- Some experts do not place any machine in this category. Some place pipelined computations under this class.
- However, the concept is useful in computations where the same input is to be subjected to several different computations.
- Some conceivable examples might be:
  - Multiple frequency filters operating on a single signal stream.
  - Multiple cryptography algorithms attempting to crack a single coded message.


IS$_n$
IS$_2$
IS$_1$
CU$_1$
CU$_2$
CU$_n$
control signals
Program & Data (Memory)
DS
PE$_1$
DS
PE$_2$
PE$_n$
DS