VIT-AP UNIVERSITY

Computer Organization & Architecture

ECE 2002

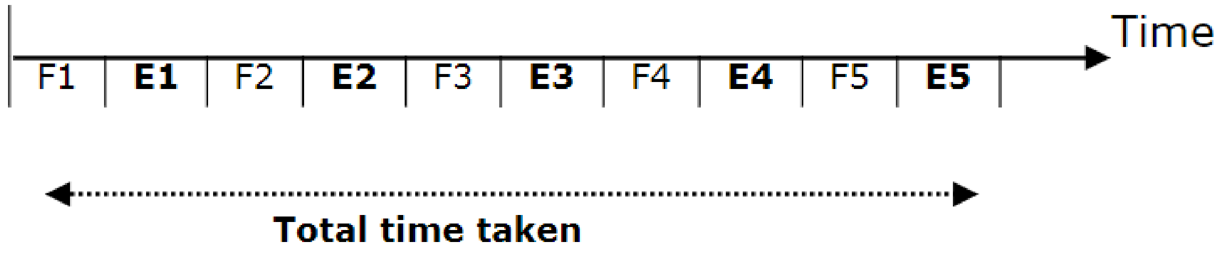
Overlapping different stages of an instruction is called pipelining.

An instruction requires several steps which mainly involve fetching, decoding and execution.
If these steps are performed one after the other, they will take a long time.
As processors became faster, several of these steps started to get overlapped, resulting in faster processing. This is done by a mechanism called pipelining.

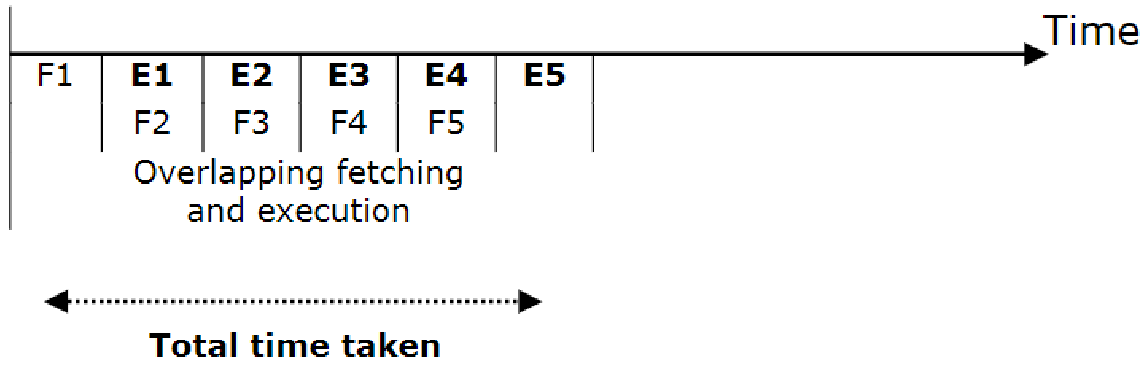
2 Stage Pipelining - 8086

Here the instruction process in divided into two stages of fetching and execution.
Fetching of the next instruction takes place while the current instruction is being executed. Hence two instructions are being processed at any point of time.

NON-PIPELINED PROCESSOR EG: 8085

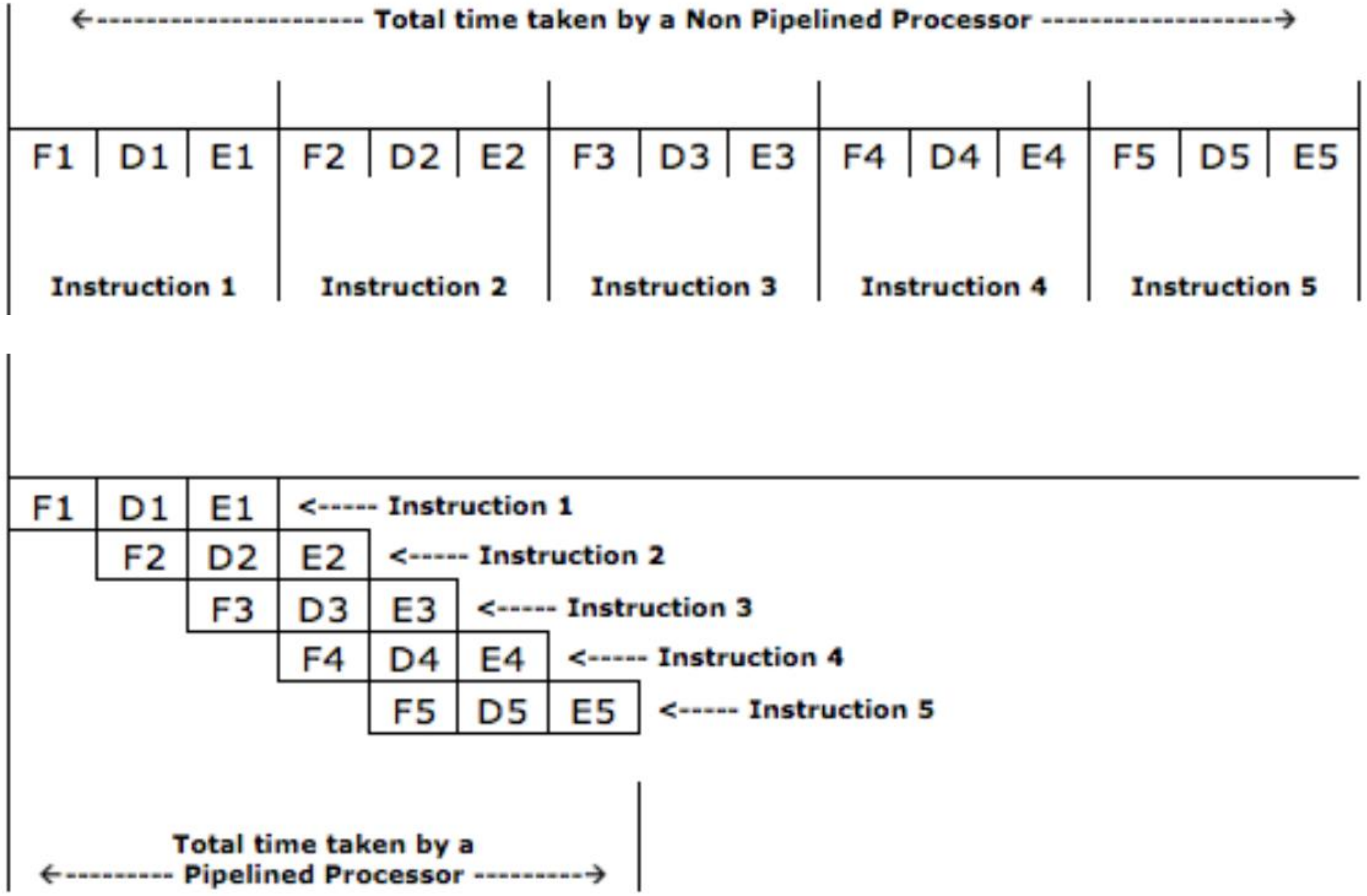


PIPELINED PROCESSOR EG: 8086



3 Stage Pipelining – 80386/ ARM 7

Here the instruction process in divided into three stages of **fetching, decoding and execution** and are overlapped. Hence **three instructions** are being processed at any point of time.

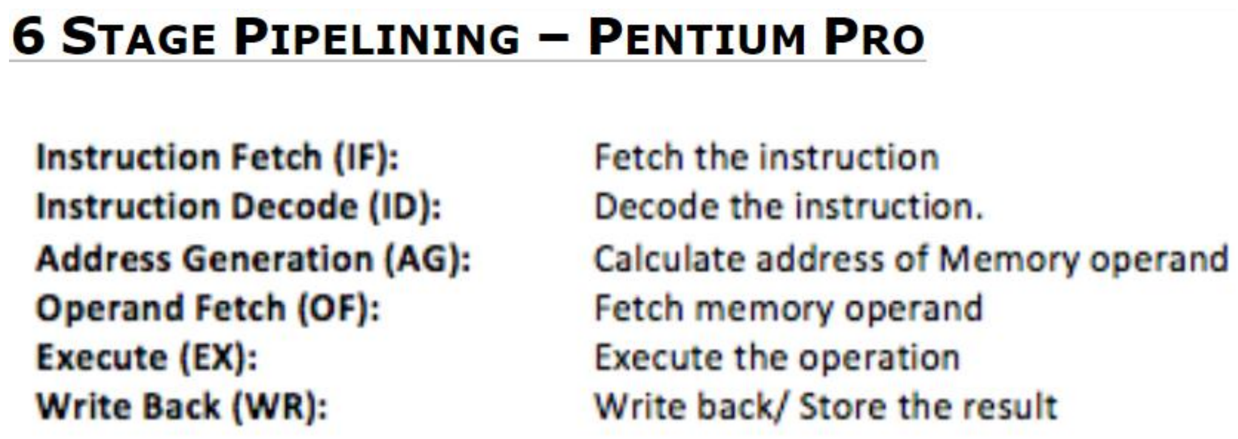

←---------------------- Total time taken by a Non Pipelined Processor ------------------→
F1 D1 E1 F2 D2 E2 F3 D3 E3 F4 D4 E4 F5 D5 E5
Instruction 1
Instruction 2
Instruction 3
Instruction 4
Instruction 5
F1 D1 E1 <----- Instruction 1
F2 D2 E2 <----- Instruction 2
F3 D3 E3 <----- Instruction 3
F4 D4 E4 <----- Instruction 4
F5 D5 E5 <----- Instruction 5
Total time taken by a
←--------- Pipelined Processor ---------→

4 Stage Pipelining

Fetch, Decode, Execute, Store

5 Stage Pipelining - Pentium

Fetch, Decode, Address Generation, Execute, Store



6 Stage Pipelining – Pentium Pro

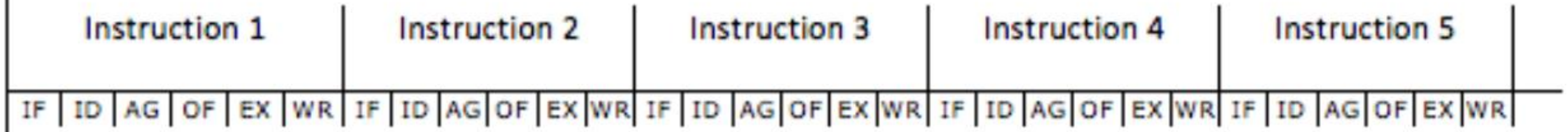
Instruction Fetch (IF):	Fetch the instruction
Instruction Decode (ID):	Decode the instruction.
Address Generation (AG):	Calculate address of Memory operand
Operand Fetch (OF):	Fetch memory operand
Execute (EX):	Execute the operation
Write Back (WR):	Write back/ Store the result

Non Pipelined Processor

K = No of stages = 6.

N = No of Instructions = 5.

Total cycles = K x N = 6 x 5 = 30 cycles.

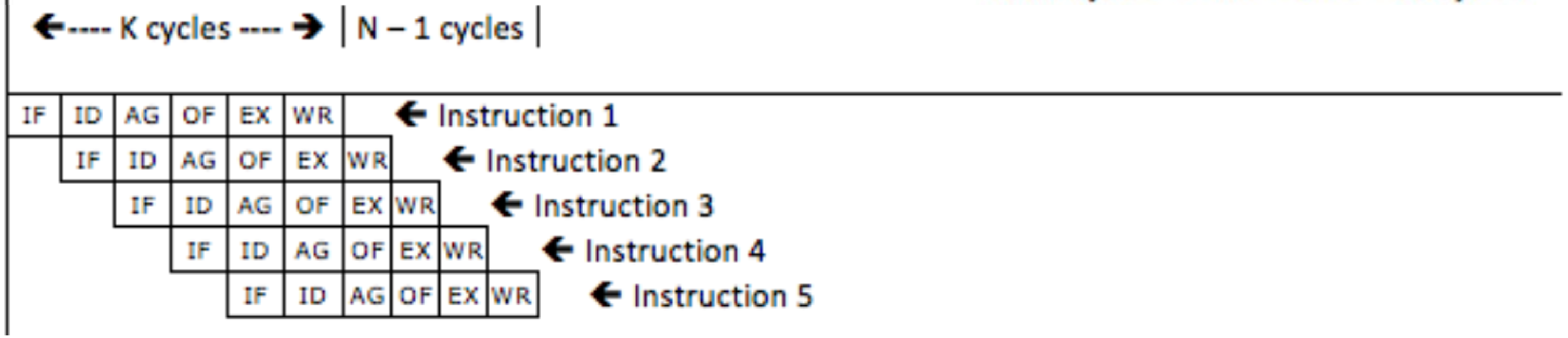

Instruction 1
Instruction 2
Instruction 3
Instruction 4
Instruction 5
IF ID AG OF EX WR IF ID AG OF EX WR IF ID AG OF EX WR IF ID AG OF EX WR IF ID AG OF EX WR

Non Pipelined Processor

K = No of stages = 6.

N = No of Instructions = 5.

Total cycles = K x N = 6 x 5 = 30 cycles.


←---- K cycles ---- →
N – 1 cycles
IF ID AG OF EX WR ← Instruction 1
IF ID AG OF EX WR ← Instruction 2
IF ID AG OF EX WR ← Instruction 3
IF ID AG OF EX WR ← Instruction 4
IF ID AG OF EX WR ← Instruction 5

Drawbacks/ Hazards of Pipelining

There are various hazards of pipelining, which **cause a dip** in the performance of the processor. These hazards become even **more prominent** as the **number of pipeline stages increase**. They may occur due to the following reasons.

1) Data Hazard/ Data Dependency Hazard

Data Hazard is caused when **the result (destination) of one instruction becomes the operand (source) of the next instruction**.

Consider two instructions I1 and I2 (I1 being the first).

Assume I1: INC [4000H]
Assume I2: MOV BL , [4000H]

Clearly in I2, BL should get the incremented value of location [4000H].
But this can only happen once I1 has completely finished execution and also written back the result at [4000H].
In a multistage pipeline, I2 may reach execution stage before I1 has finished storing the result at location [4000H], and hence get a wrong value of data.
This is called **data dependency hazard**.
It is solved by inserting NOP (No operation) instructions between such data dependent instructions.

2) Control Hazard/ Code Hazard

Pipelining assumes that the program will always flow in a sequential manner.
Hence, it performs various stages of the forthcoming instructions before-hand, while the current instruction is still being executed.
While programs are sequential most of the times, it is not true always.
Sometimes, branches do occur in programs.
In such an event, all the forthcoming instructions that have been fetched/ decoded etc have to be flushed/ discarded, and the process has to start all over again, from the branch address. This causes pipeline bubbles, which simply means time of the processor is wasted.

Consider the following set of instructions:

```
Start:
        JMP Down
        INC BL
        MOV CL, DL
        ADD AL, BL
        ...
        ...
        ...

Down:   DEC CH
```

JMP Down is a branch instruction.
After this instruction, program should jump to the location "Down" and continue with DEC CH instruction.

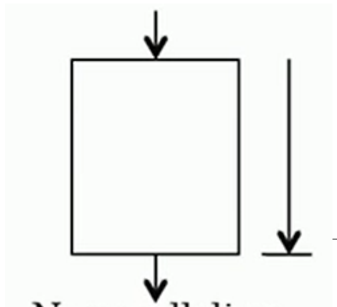
But, in a multistage pipeline processor, the sequentially next instructions after JMP Down have already been fetched and decoded. These instructions will now have to be discarded and fetching will begin all over again from DEC CH. This will keep several units of the architecture idle for some time. This is called a pipeline bubble.

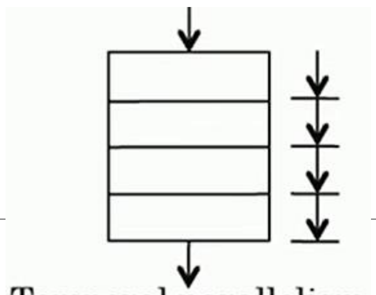
The **problem of branching is solved** in higher processors by a method called "**Branch Prediction Algorithm**". It was introduced by **Pentium** processor. It relies on the **previous history** of the instruction as most programs are repetitive in nature. It then **makes a prediction** whether branch will be **taken or not** and hence puts the correct instructions in the pipelines.

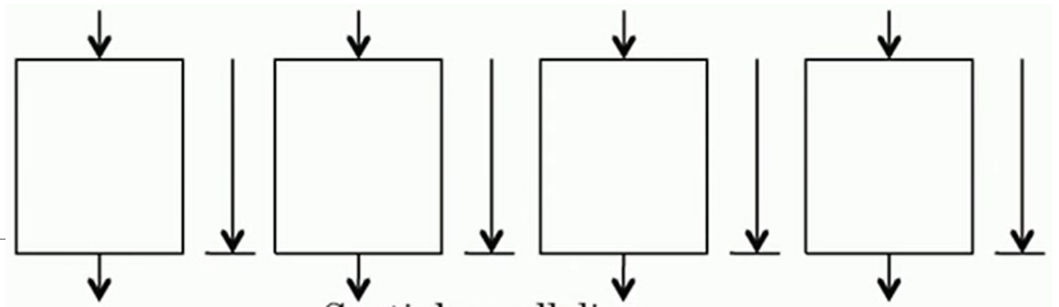
3) STRUCTURAL HAZARD

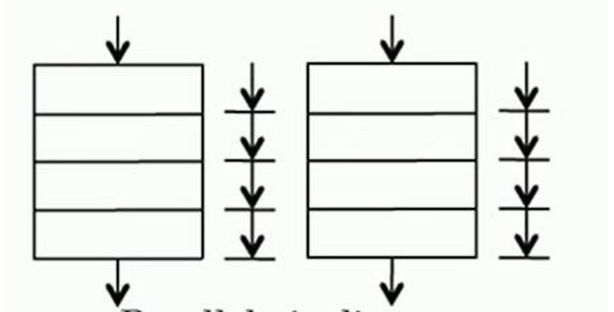
Structural hazards are caused by **physical constraints in the architecture like the buses**. Even in the most basic form of pipelining, we want to execute one instruction and fetch the next one. Now as long as execution only involves registers, pipelining is possible. But **if execution requires to read/ write data from the memory, then it will make use of the buses, which means fetching cannot take place at the same time**. So the fetching unit will have to wait and hence a pipeline bubble is caused. This problem is solved in complex Harvard architecture processors, which use separate memories and separate buses for programs and data. This means fetching and execution can actually happen at the same time without any interference with each other.
E.g.: PIC 18 Microcontroller.

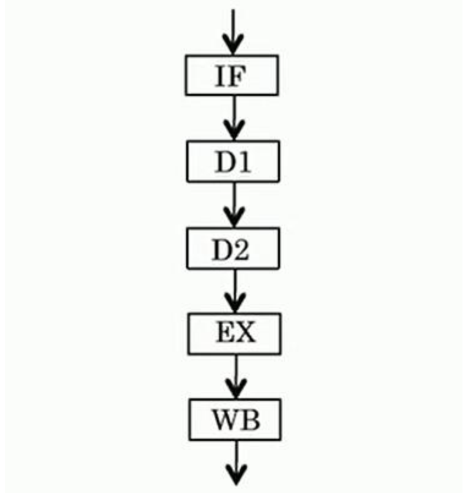
Scalar to Superscalar pipeline


No parallelism

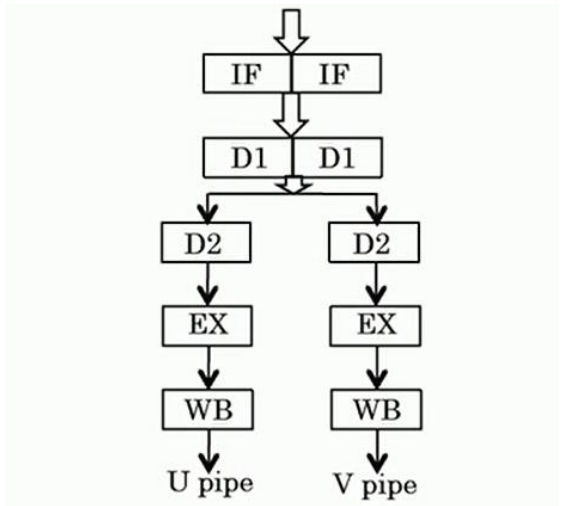

Temporal parallelism


Spatial parallelism

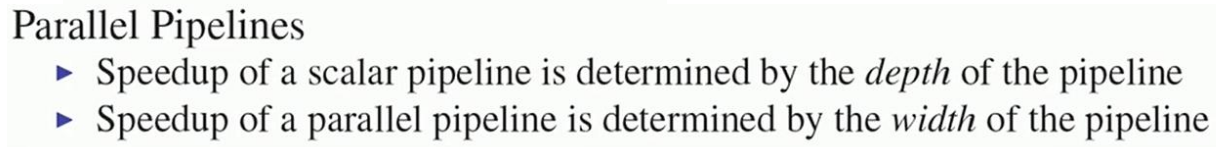

Parallel pipeline



5-stage i486 scalar pipeline

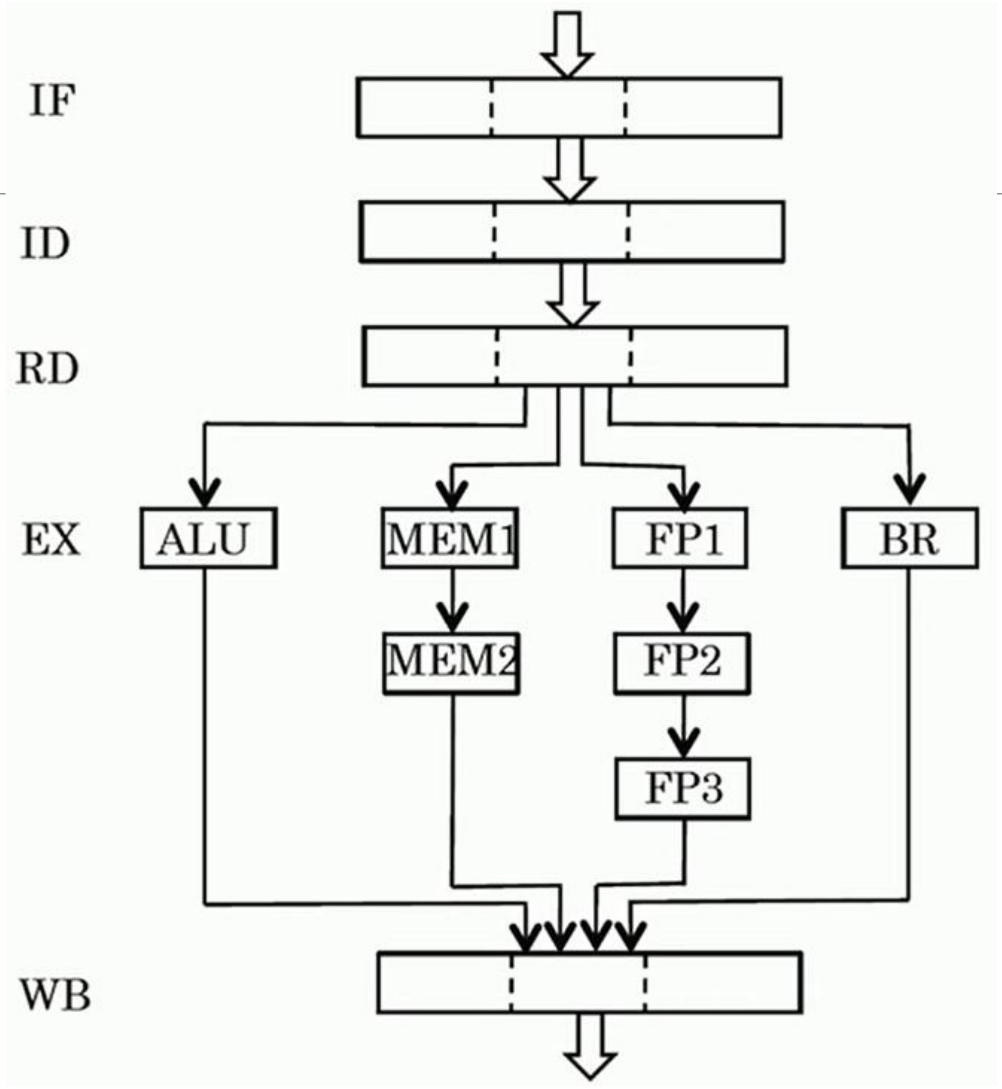


5-stage Pentium parallel pipeline

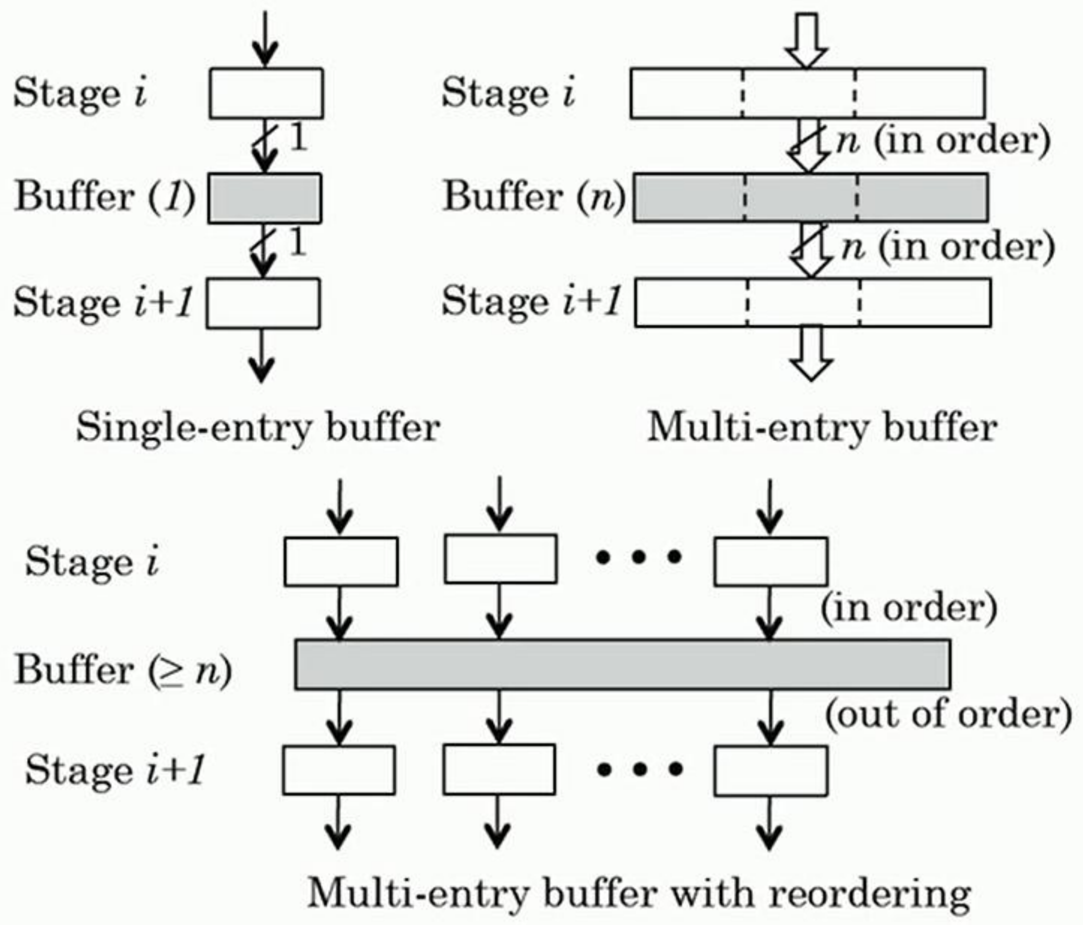

Parallel Pipelines

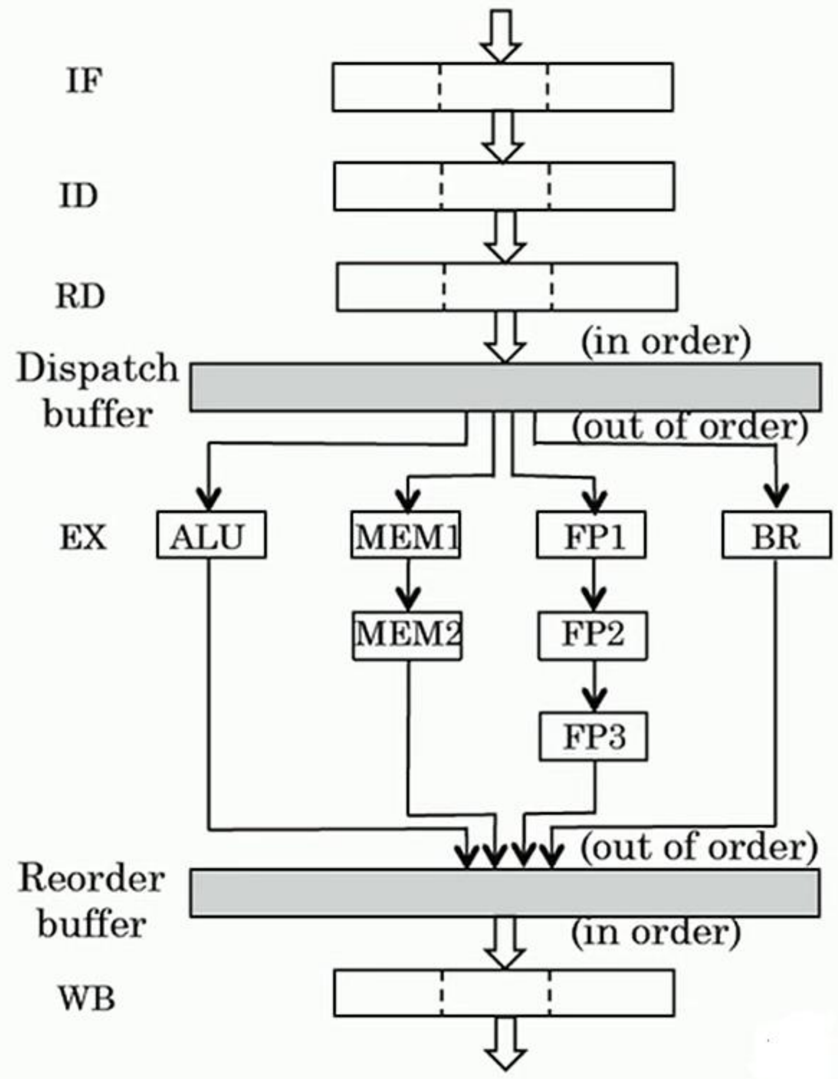
- Speedup of a scalar pipeline is determined by the *depth* of the pipeline
- Speedup of a parallel pipeline is determined by the *width* of the pipeline

Diversified Pipeline



Dynamic pipelines

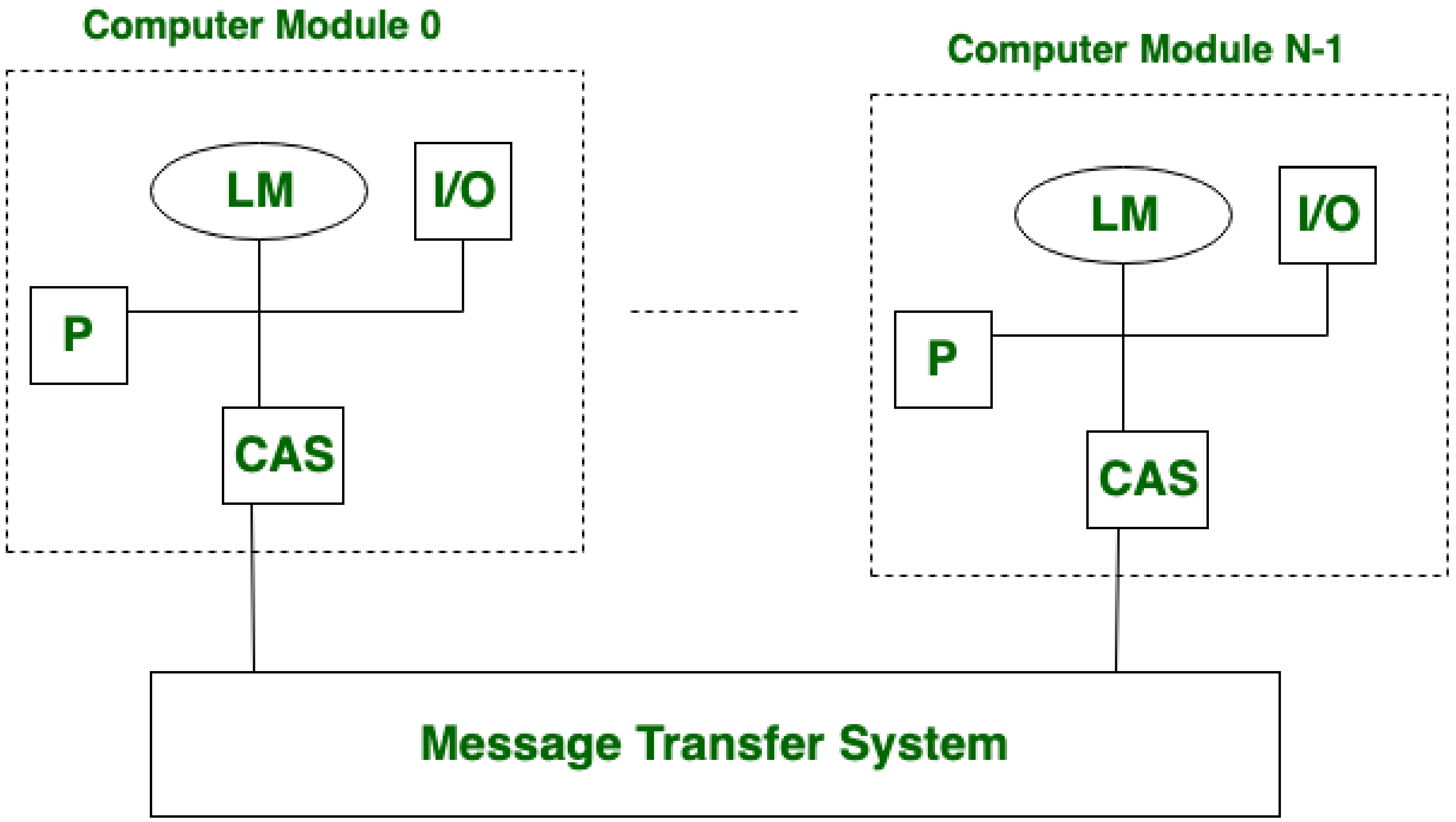




Loosely Coupled Multiprocessor system

- It is a type of multiprocessing system in which, There is distributed memory instead of shared memory.
- In loosely coupled multiprocessor system, data rate is low rather than tightly coupled multiprocessor system.
- In loosely coupled multiprocessor system, modules are connected through MTS (Message transfer system) network.

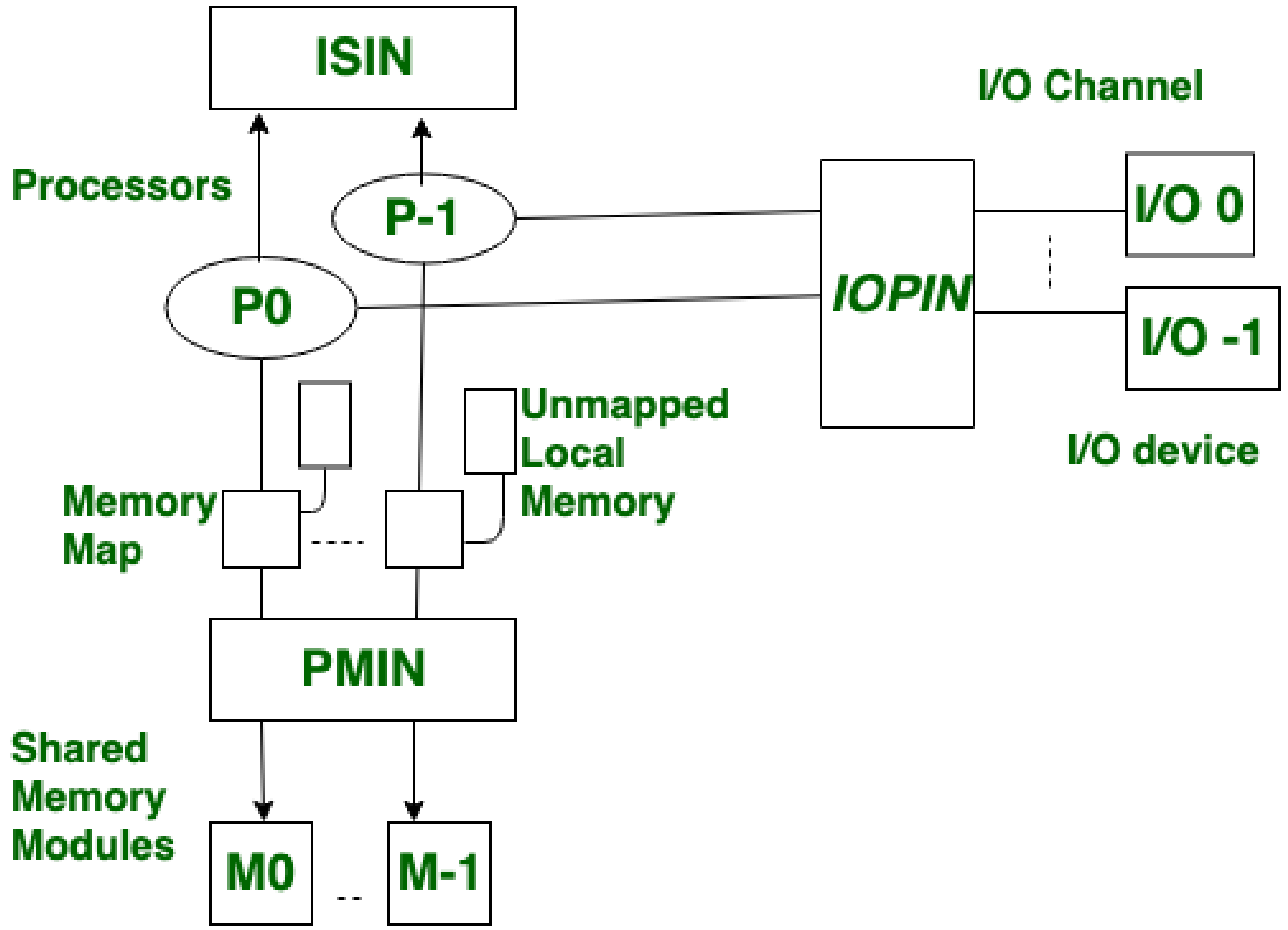
Loosely Coupled Multiprocessor system



Tightly Coupled Multiprocessor system

- It is a type of multiprocessing system in which there is shared memory.
- In tightly coupled multiprocessor system, data rate is high rather than loosely coupled multiprocessor system.
- In tightly coupled multiprocessor system, modules are connected through PMIN, IOPIN and ISIN networks.

Tightly Coupled Multiprocessor system



Differences

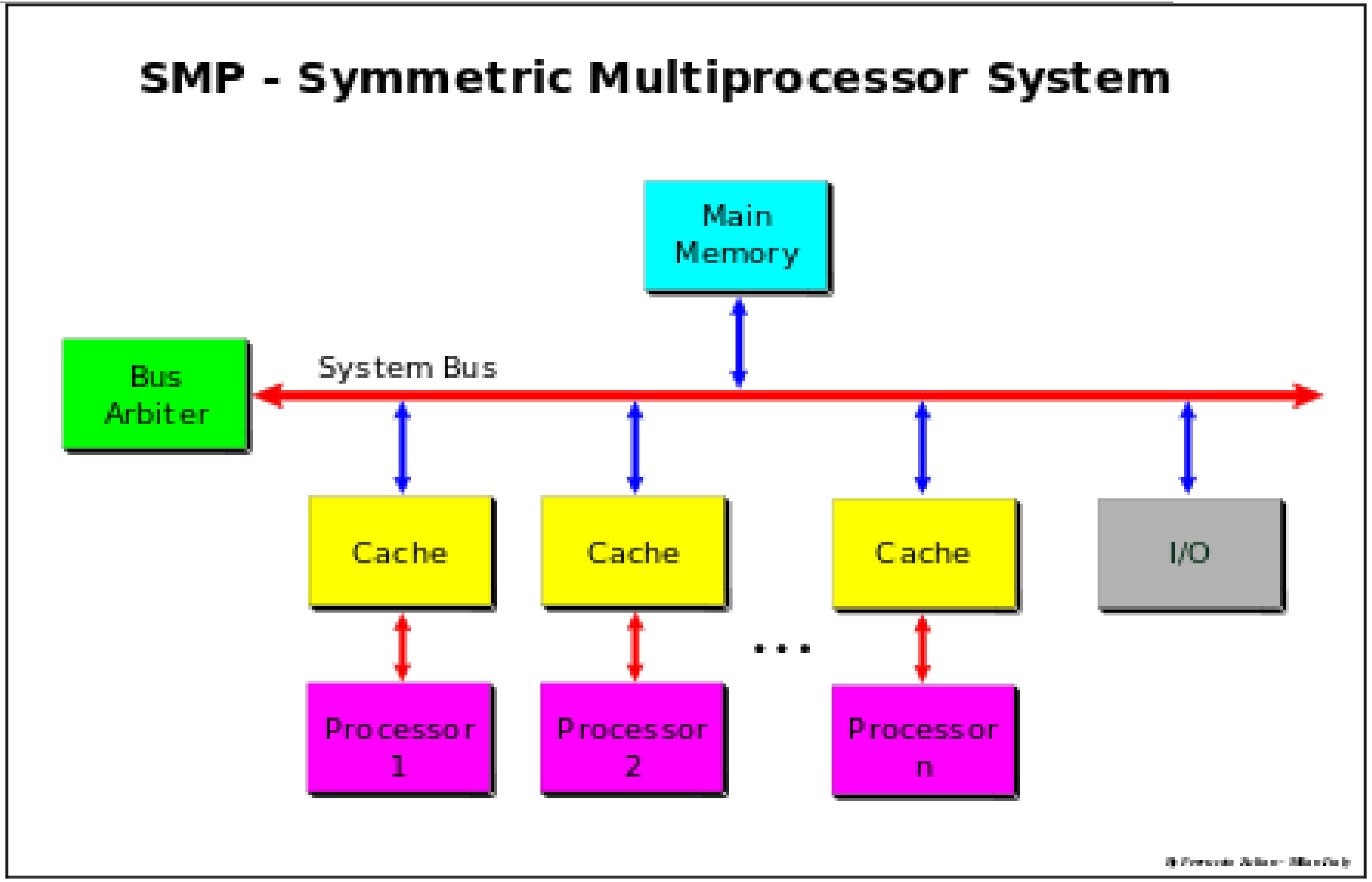
Loosely coupled	Tightly coupled
There is distributed memory in loosely coupled multiprocessor system	There is shared memory, in tightly coupled multiprocessor system
Has low data rate	Has high data rate
The cost of this system is less	It is more costly
Modules are connected through Message transfer system network	While there is PMIN, IOPIN and ISIN networks
Memory conflicts don’t take place	This system have memory conflicts
It has low degree of interaction between tasks	It has high degree of interaction between tasks
there is direct connection between processor and I/O devices	IOPIN helps connection between processor and I/O devices
Applications of loosely coupled multiprocessor are in distributed computing systems	Applications of tightly coupled multiprocessor are in parallel processing systems

Symmetric Multiprocessor system

- SMP systems have centralized shared memory called *main memory* (MM) operating under a single operating system with two or more homogeneous processors.
- Usually each processor has an associated private high-speed memory known as cache memory (or cache) to speed up the main memory data access and to reduce the system bus traffic.
- Processors may be interconnected using buses, crossbar switches or on-chip mesh networks.
- The bottleneck in the scalability of SMP using buses or crossbar switches is the bandwidth and power consumption of the interconnect among the various processors, the memory, and the disk arrays.
- Mesh architectures avoid these bottlenecks, and provide nearly linear scalability to much higher processor counts at the sacrifice of programmability

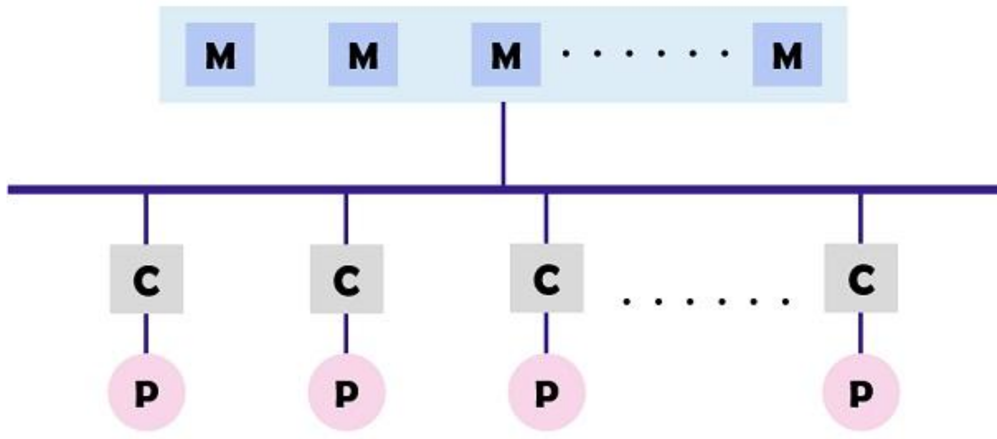
Symmetric Multiprocessor system

- SMP systems allow any processor to work on any task no matter where the data for that task is located in memory, provided that each task in the system is not in execution on two or more processors at the same time.
- With proper operating system support, SMP systems can easily move tasks between processors to balance the workload efficiently.



UMA (Uniform memory access)

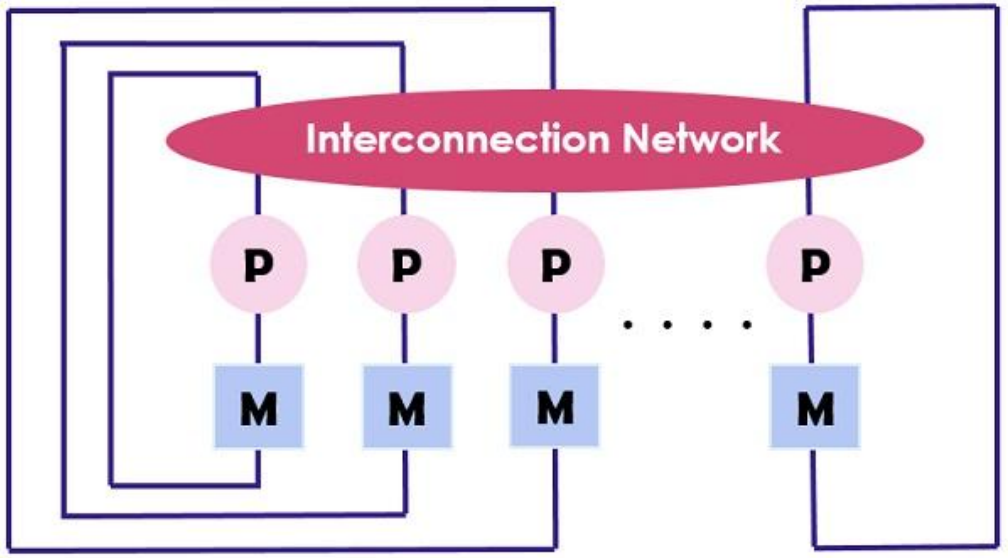
- UMA stands for Uniform Memory Access; it is a shared memory architecture for the multiprocessors.
- Single memory controller is used and accessed by all the processors with the help of the interconnection network.
- Each processor has equal memory accessing time (latency) and access speed.
- It can employ either of the single bus, multiple bus or crossbar switch.
- As it provides balanced shared memory access, it is also known as **SMP (Symmetric multiprocessor)** systems.
- Uniform Memory Access is slower than non-uniform Memory Access.
- Uniform Memory Access has limited bandwidth.
- Uniform Memory Access is applicable for general purpose applications and time-sharing applications.
- In uniform Memory Access, memory access time is balanced or equal.



Bus-based UMA (SMP) Shared Memory

NUMA (Non-uniform memory access)

- NUMA stands for Non-Uniform Memory Access; it is a multiprocessor model in which each processor is connected with a dedicated memory.
- However, these small parts of the memory combine to make a single address space.
- Unlike UMA, the access time of the memory relies on the distance where the processor is placed → which means varying memory access time.
- It allows access to any of the memory location by using the physical address.
- NUMA is intended to increase the available bandwidth to the memory and for which it uses multiple memory controllers.
- It combines numerous machine cores into "**nodes**" where each core has a memory controller.
- To access the local memory in a NUMA machine the core retrieves the memory managed by the memory controller by its node.
- While to access the remote memory which is handled by the other memory controller, the core sends the memory request through the interconnection links.



NUMA Shared Memory System