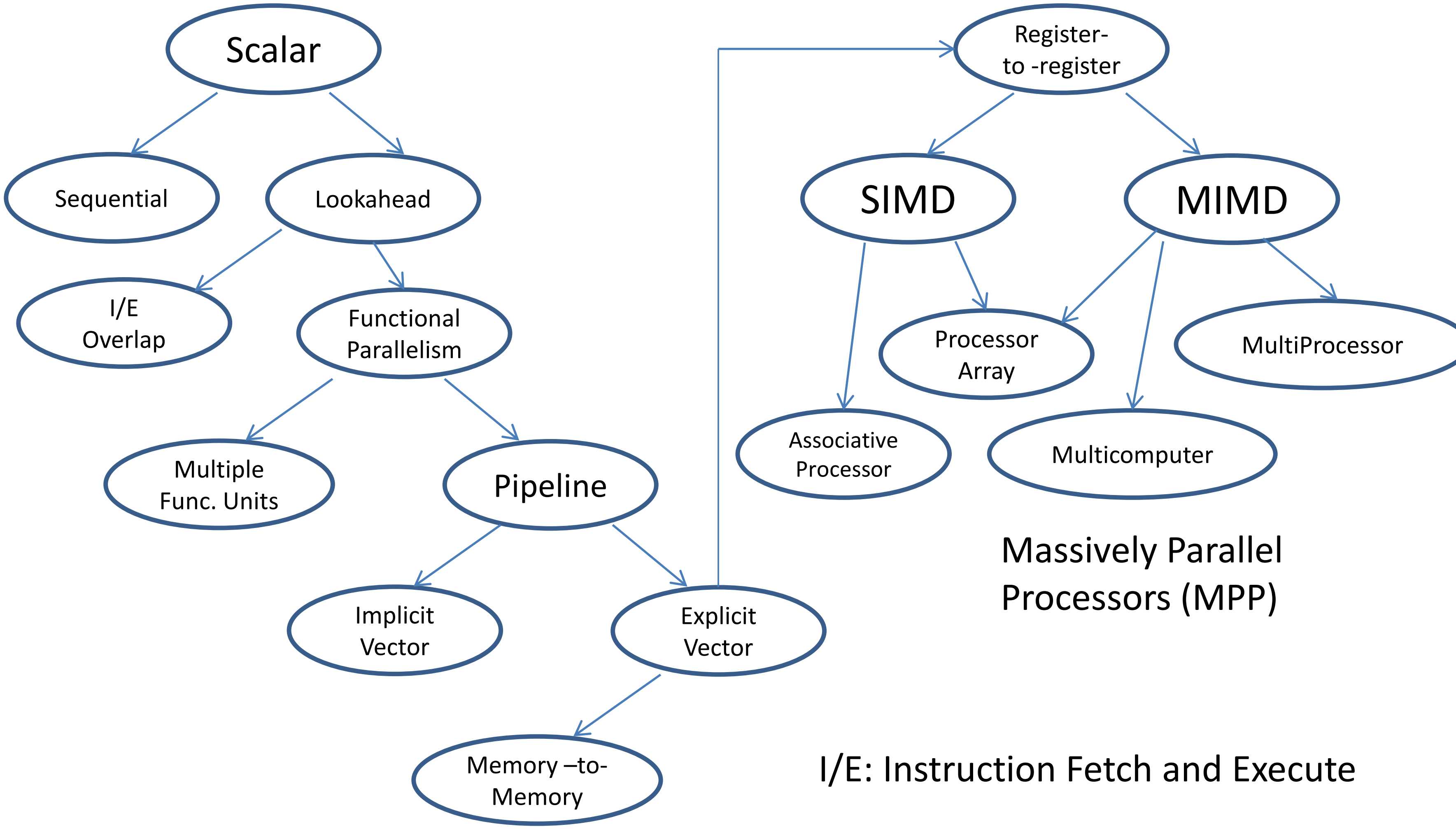
Evolution of Computer Architecture

Dr. Mohammed Abdulridha Hussain

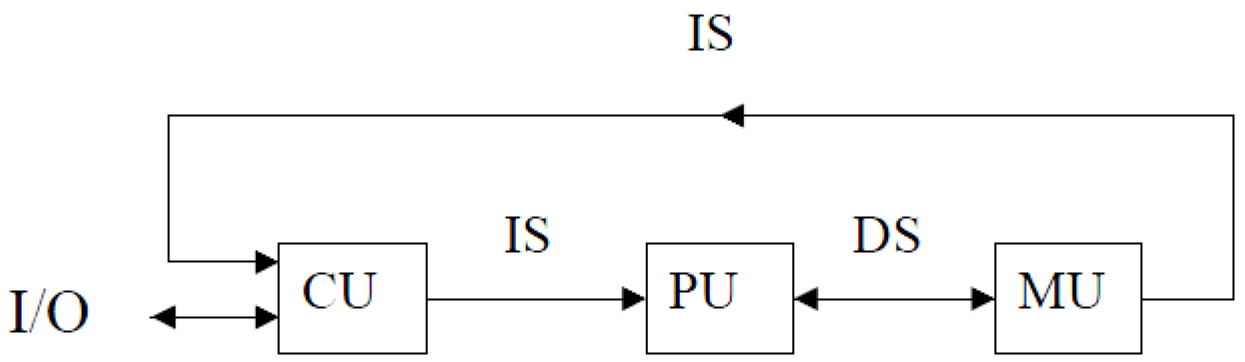
Flynn’s Classification



Massively Parallel Processors (MPP)

I/E: Instruction Fetch and Execute

SISD

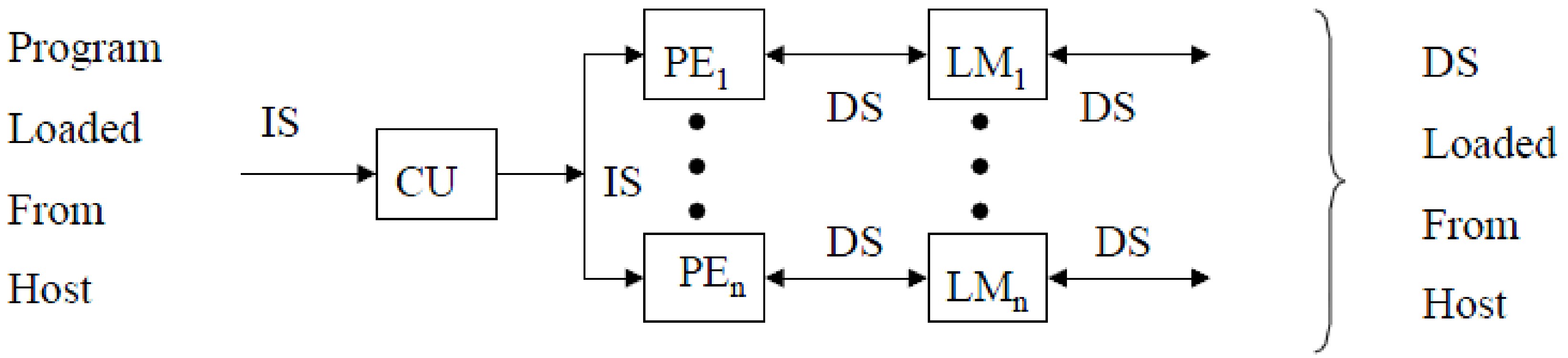


(a) SISD Uniprocessor Architecture

Captions:

CU - Control Unit	**;**	**PU – Processing Unit**
MU – Memory Unit	**;**	**IS – Instruction Stream**
DS – Date Stream		

SIMD

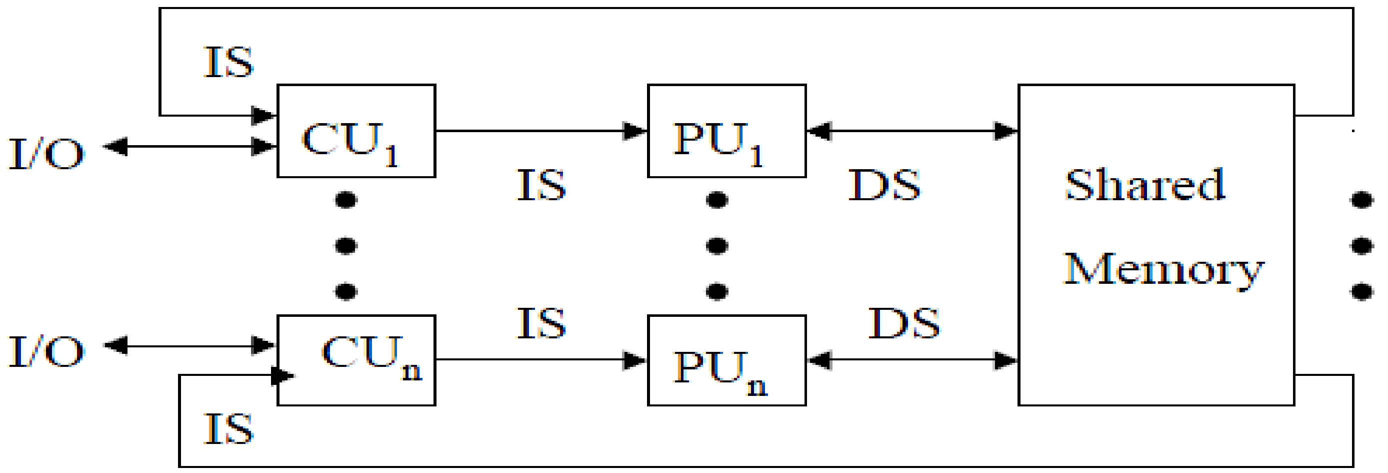


(b) SIMD Architecture (with Distributed Memory)

Captions:

CU - Control Unit	**;**	**PU - Processing Unit**
MU - Memory Unit	**;**	**IS - Instruction Stream**
DS - Date Stream	**;**	**PE – Processing Element**
LM – Local Memory		

MIMD

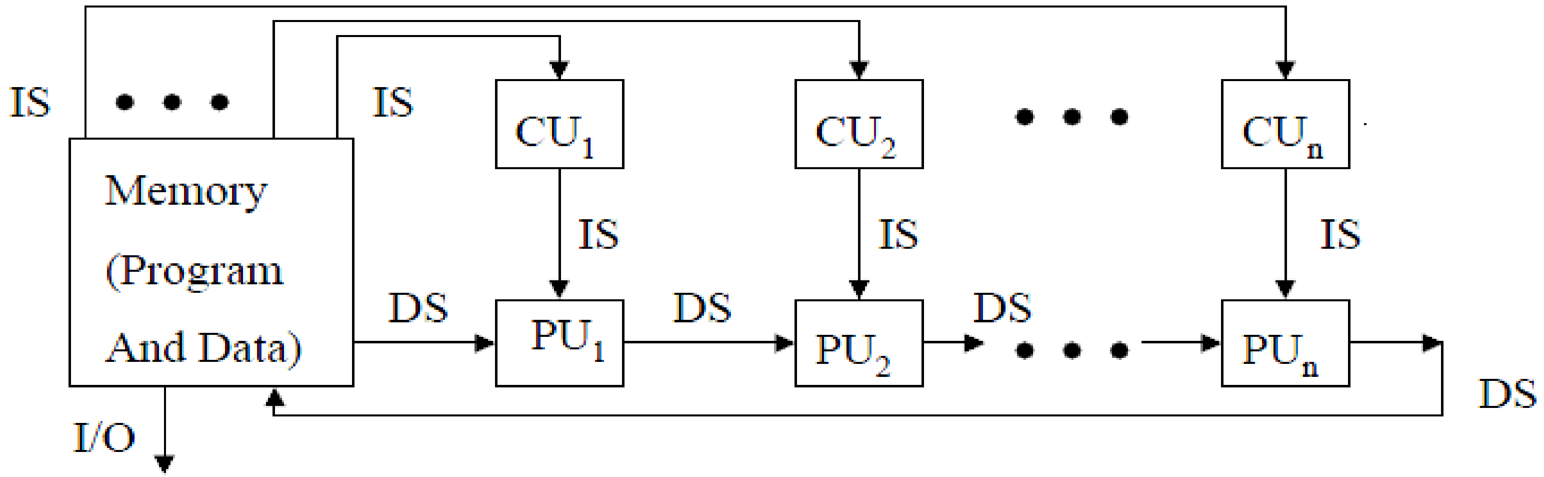


(c) MIMD Architecture (with Shared Memory)

Captions:

CU - Control Unit	;	**PU - Processing Unit**
MU - Memory Unit	;	**IS - Instruction Stream**
DS - Date Stream	;	**PE – Processing Element**
LM – Local Memory		

MISD



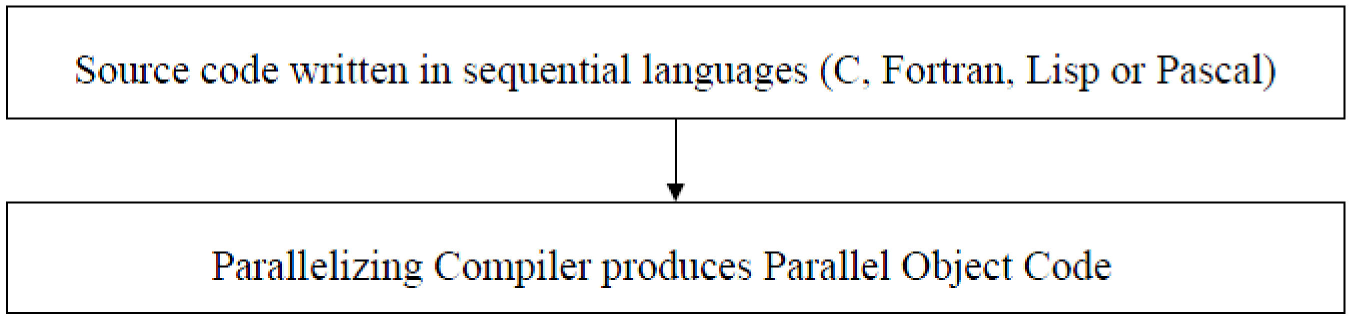
(d) MISD Architecture (the Systolic Array)

Captions:

CU - Control Unit	**;**	**PU - Processing Unit**
MU - Memory Unit	**;**	**IS - Instruction Stream**
DS - Date Stream	**;**	**PE – Processing Element**
LM – Local Memory		

Two Approaches to Parallel Programming

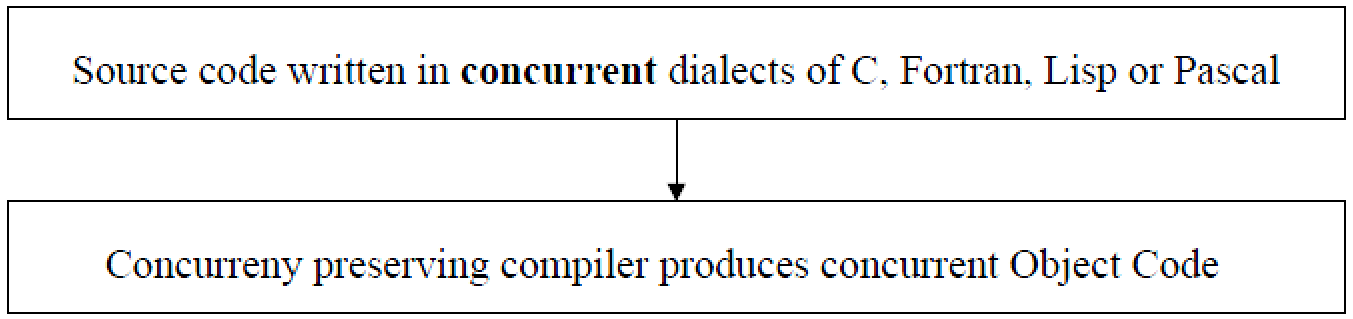
a) Implicit Parallelism



Source code written in sequential languages (C, Fortran, Lisp or Pascal)

Parallelizing Compiler produces Parallel Object Code

b) Explicit Parallelism



Source code written in **concurrent** dialects of C, Fortran, Lisp or Pascal

Concurreny preserving compiler produces concurrent Object Code

Two Categories of Parallel Computers

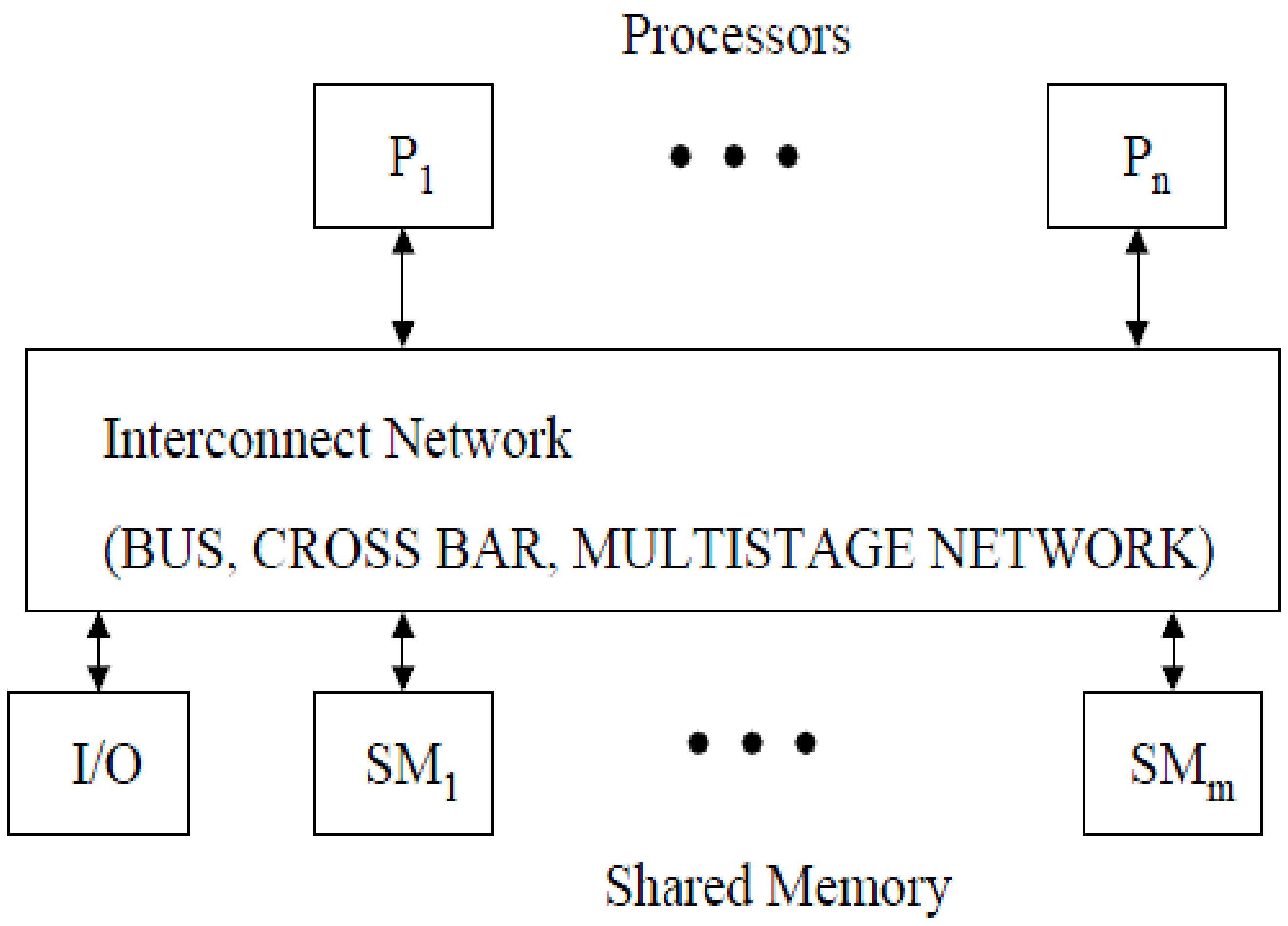
1. Shared Memory Multiprocessors (tightly coupled systems
2. Message Passing Multicomputers

SHARED MEMORY MULTIPROCESSOR MODELS:

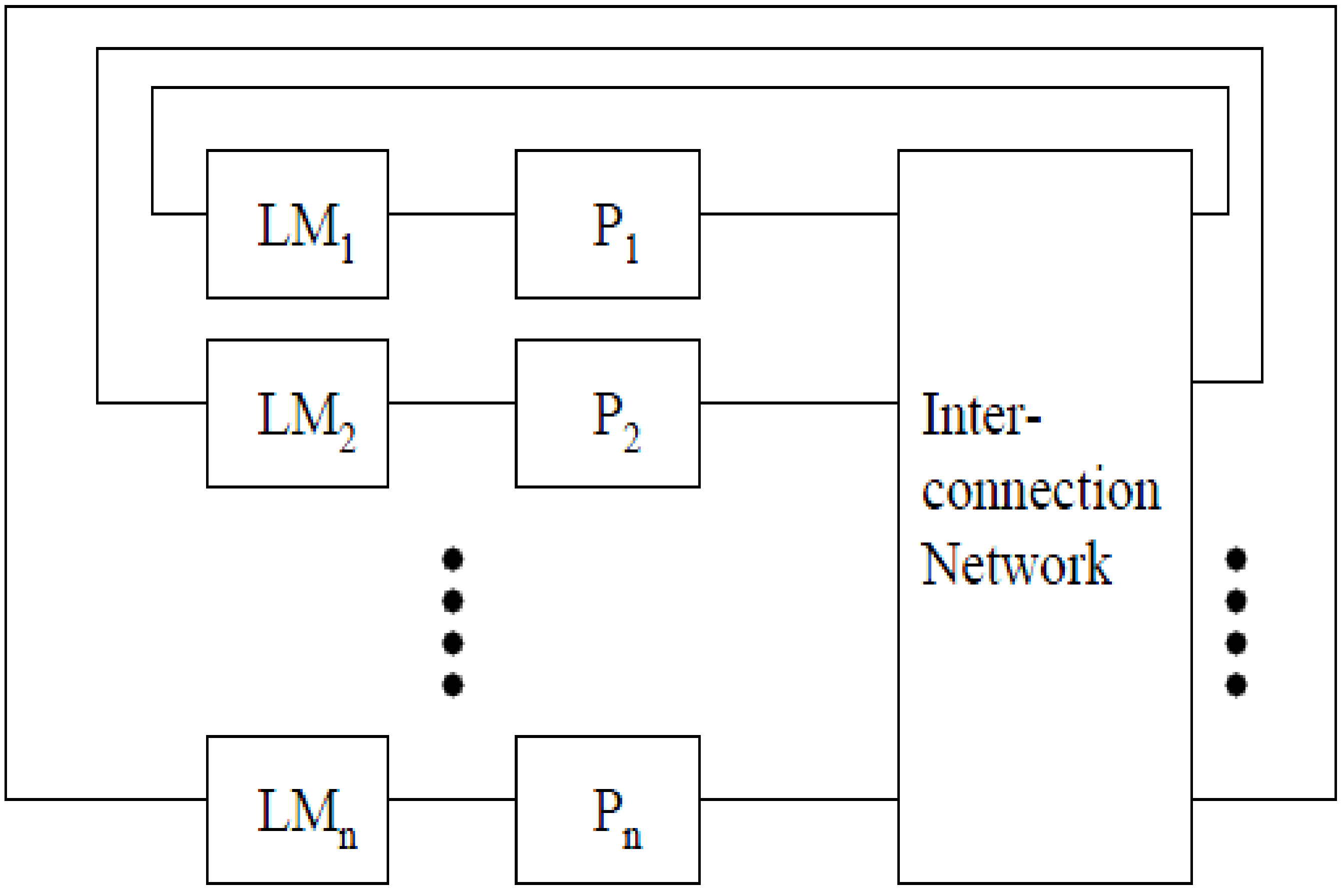
a. Uniform Memory Access (UMA)

b. Non-Uniform Memory Access (NUMA)

c. Cache-Only Memory Architecture (COMA)

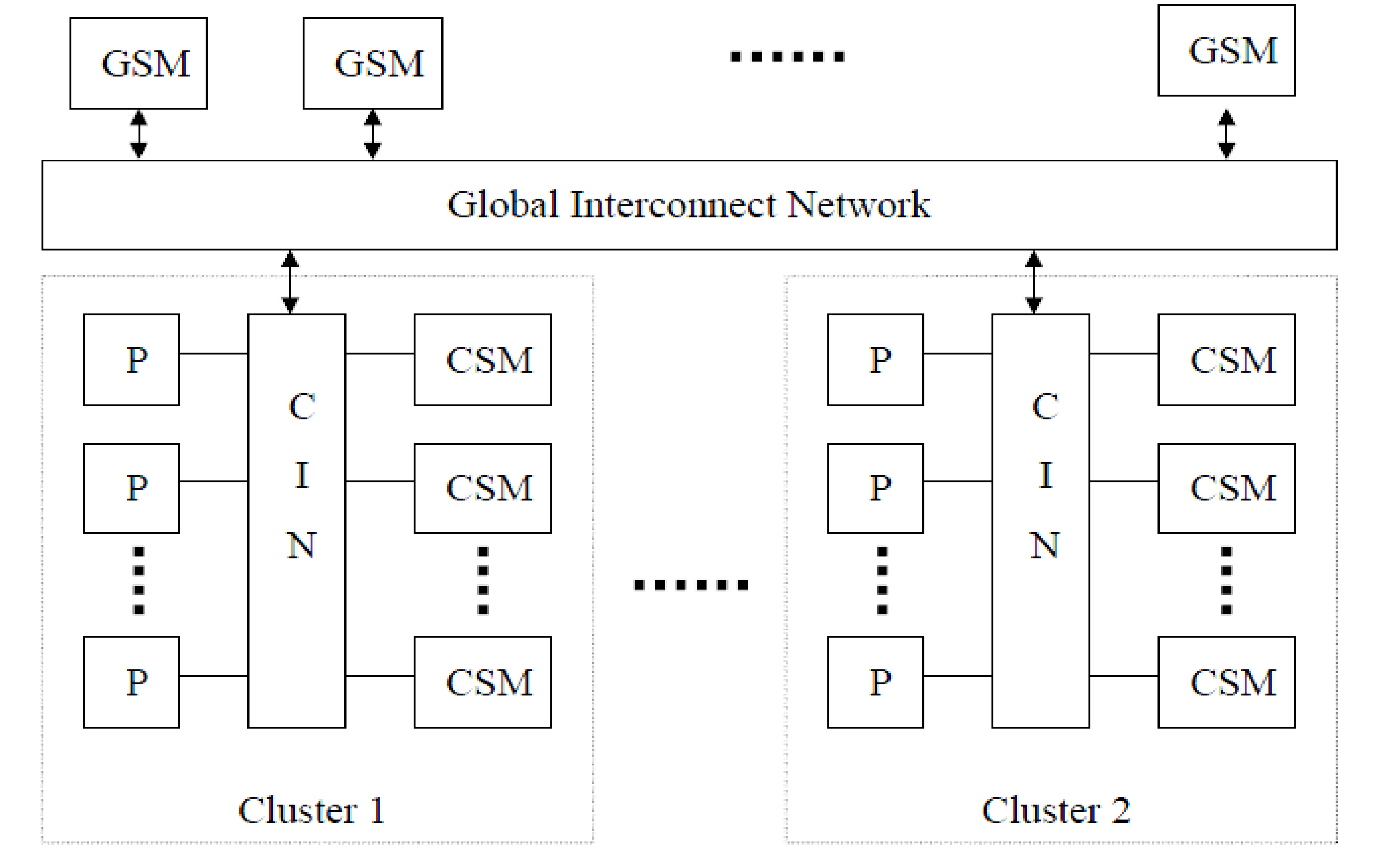


The **UMA multiprocessor model** (e.g., the Sequent Symmetry S-81)



(a) Shared local Memories (e.g., the BBN Butterfly)

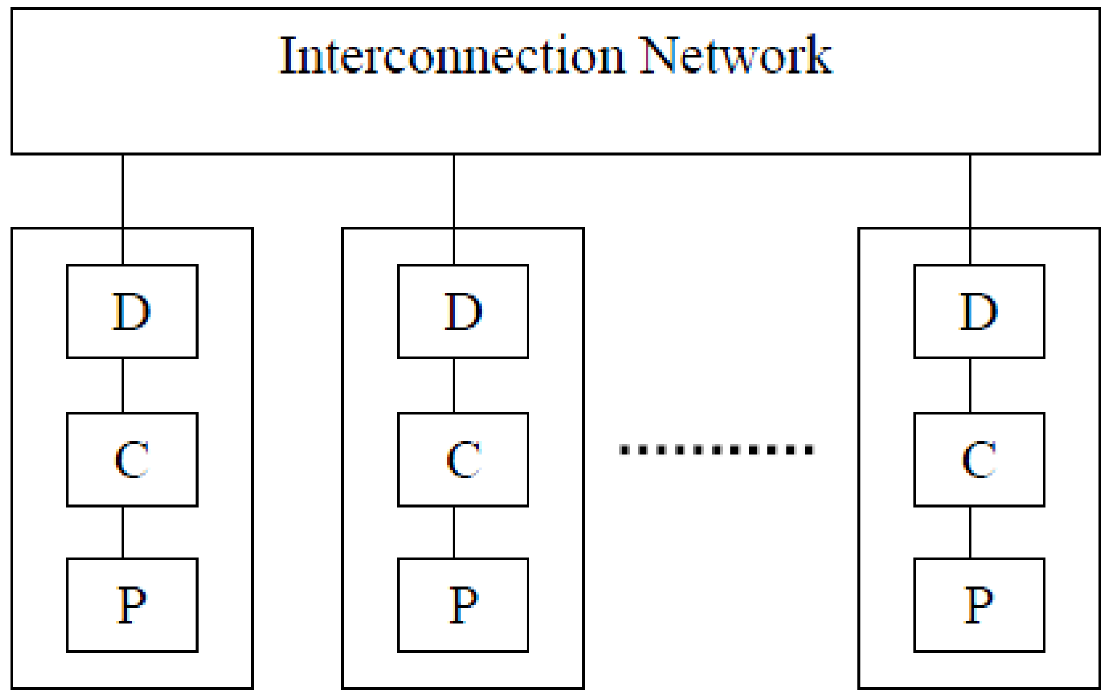
NUMA Models for Multiprocessor Systems



(b) A hierarchical cluster model (e.g., the Cedar system at the University of Illinois)

NUMA Models for Multiprocessor Systems

P: Processor, CSM: Cluster Shared Memory, GSM: Global Shared Memory
CIN: Cluster Interconnection Network



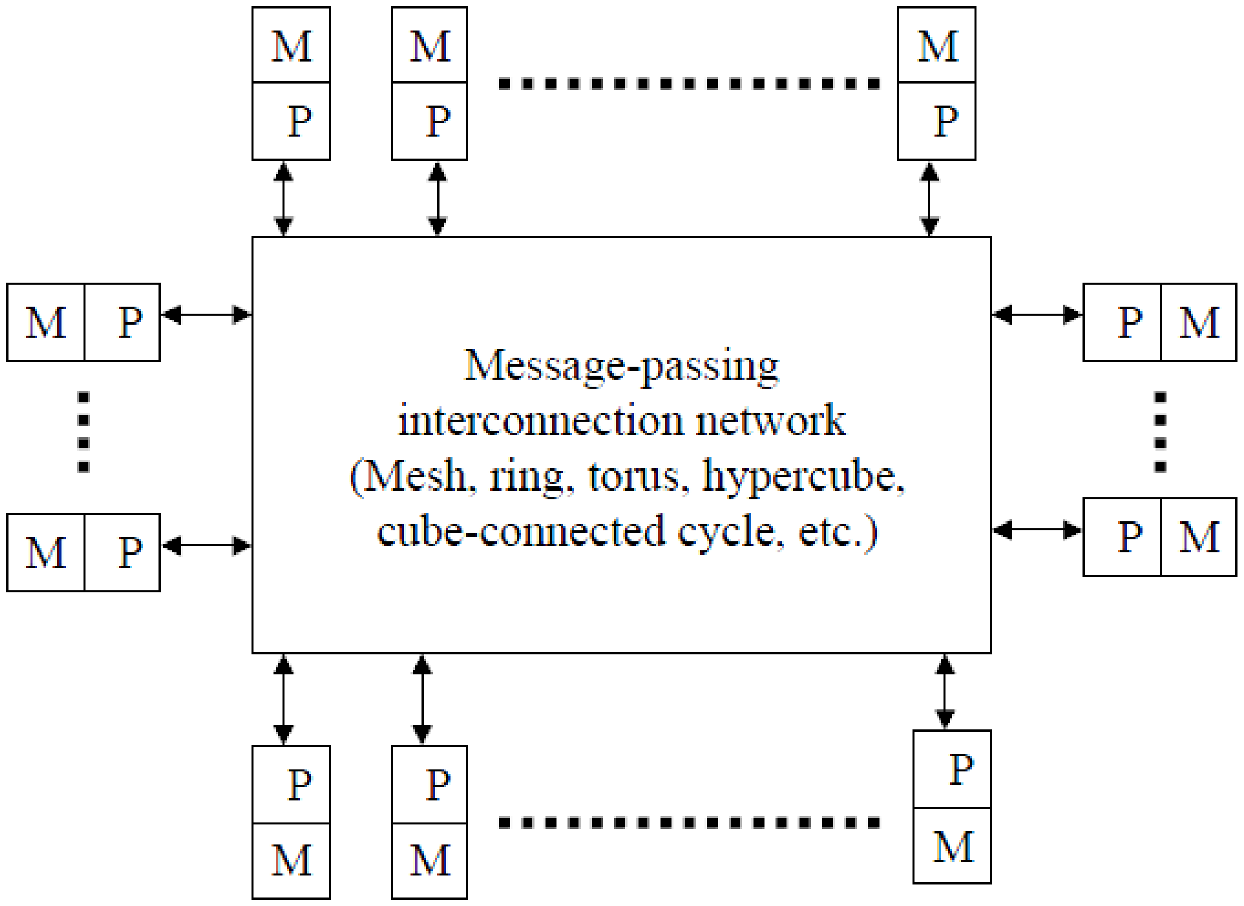
P : Processor

C : Cache

D : Directory

The COMA Model of a multiprocessor (e.g., the KSR-1)

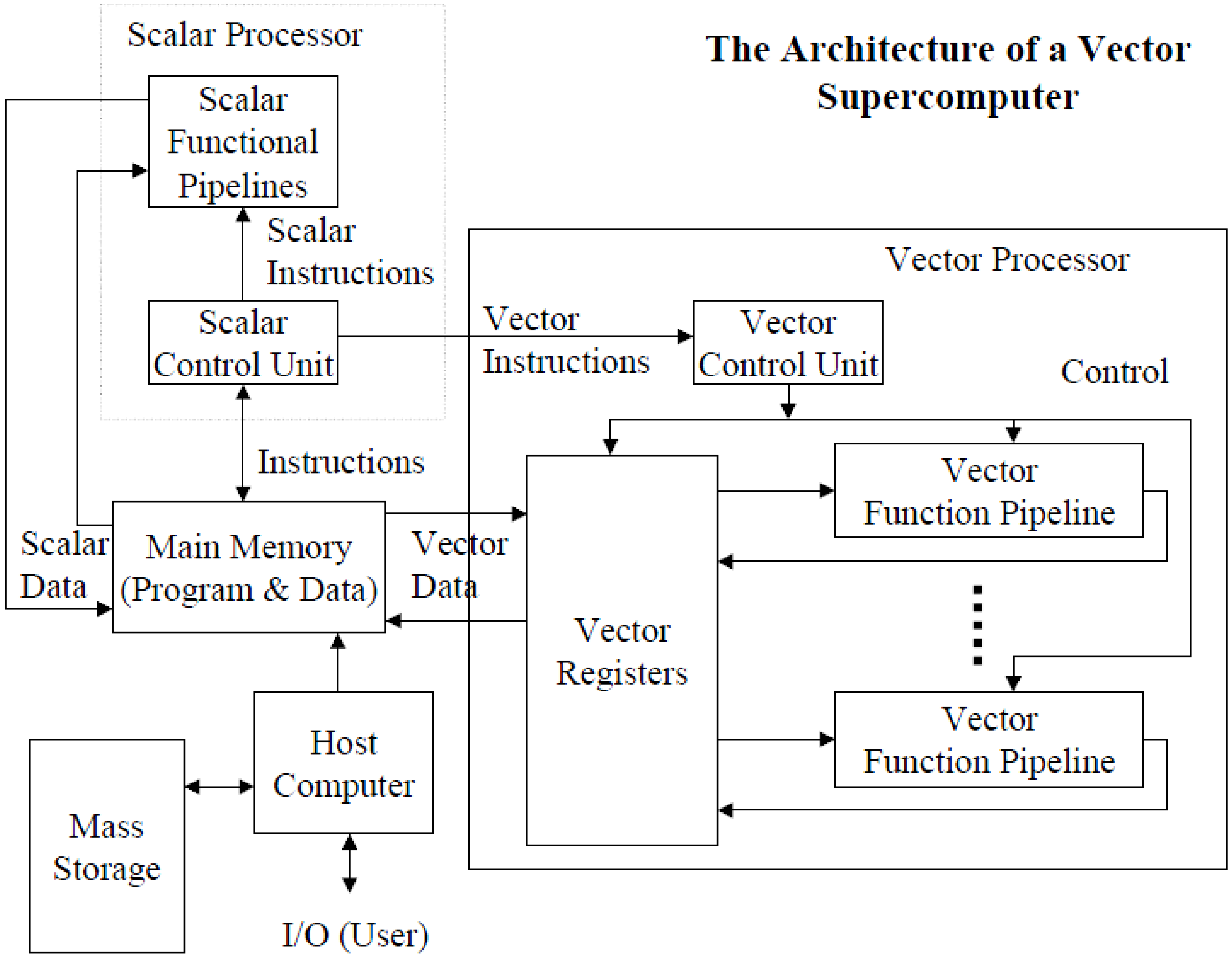
Generic Model of a message-passing multicomputer



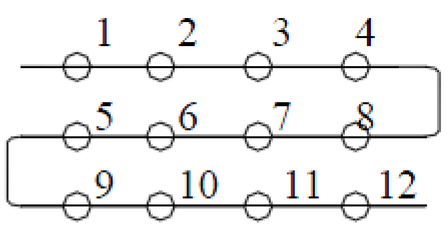
e.g., Intel Paragon, nCUBE/2

Important issues: Message Routing Scheme, Network flow control strategies, dead lock avoidance, virtual channels, message-passing primitives, program decomposition techniques.

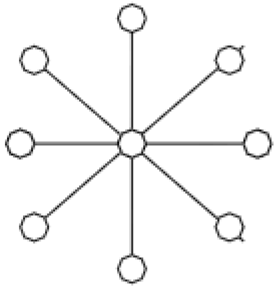
The Architecture of a Vector Supercomputer



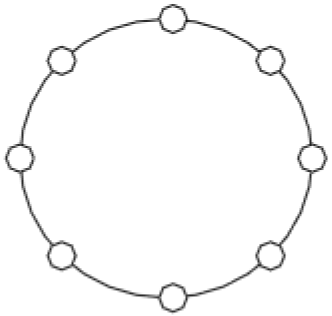
STATIC Connection Networks



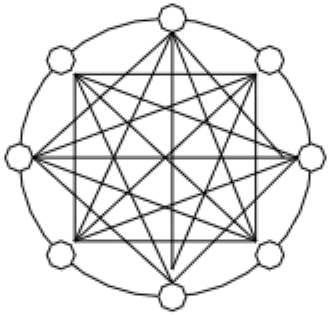
Linear Array



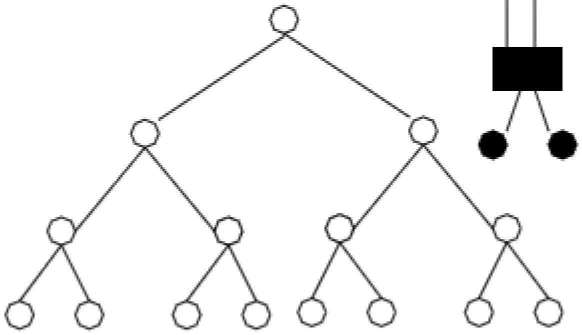
Star



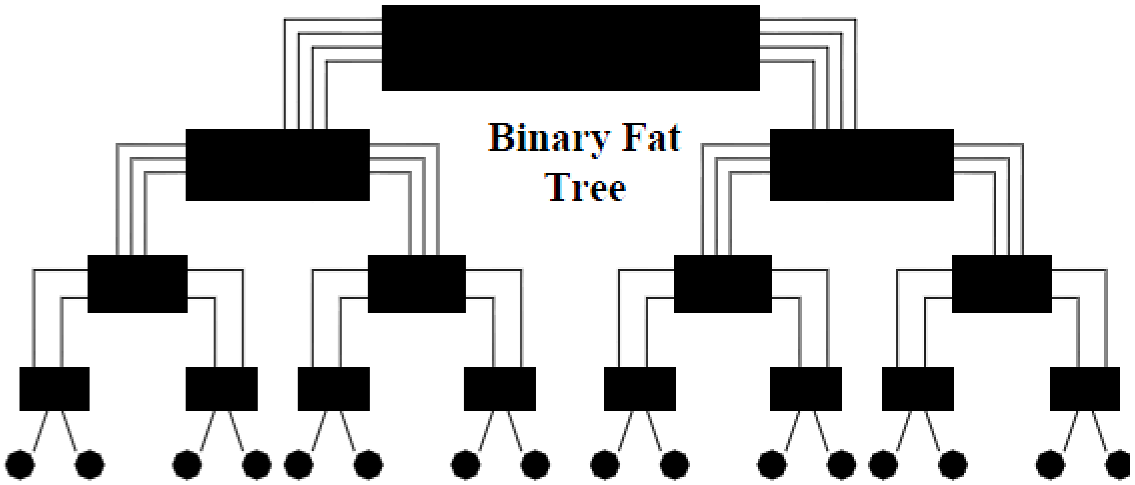
Ring



Fully connected Ring

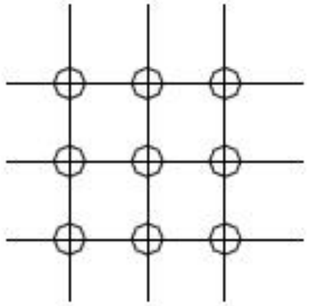


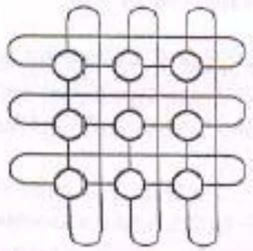
Binary Tree

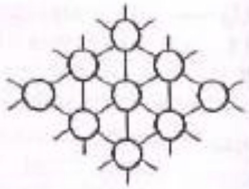


Binary Fat Tree

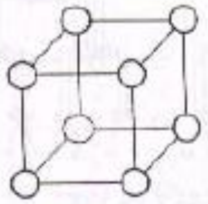
The Channel width of Fat Tree increases as we ascend from leaves to root. This concept is used in CM5 connection Machine.

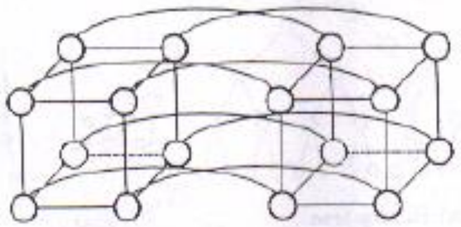

Mesh


Torus


Systolic Array

Degree = t


3-cube


A 4 dimentional cube formed with 3D cubes

Pipelining

Dr. Mohammed Abdulridha Hussain

Introduction

- Linear Pipeline Processors

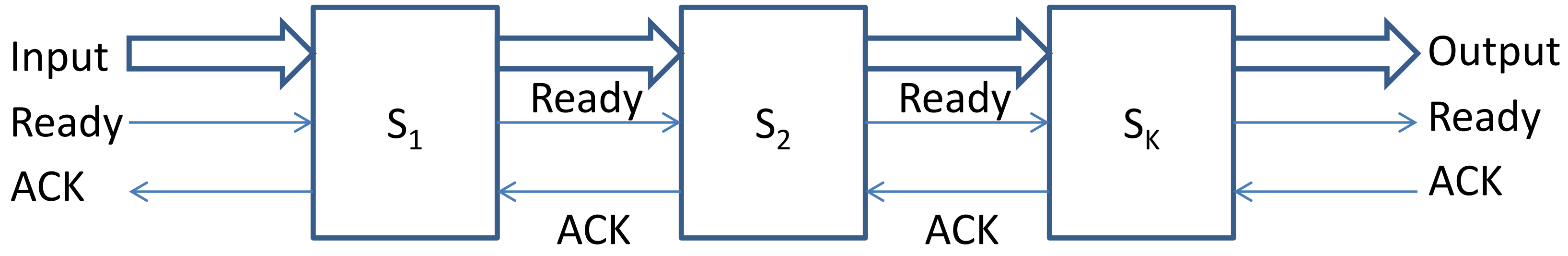
  A linear pipeline Processor is a cascade of processing stages which are linearly connected to perform a fixed function over a stream of data flowing from one end to the other.

  Linear pipelines are applied for instruction execution, arithmetic computation, and memory access operations.

Asynchronous Pipeline

- Data flow between adjacent stages in an asynchronous pipeline is controlled by a handshaking protocol.
- When stage Si is ready to transmit, it sends a ready signal to stage Si+1. After Si+1 receives the incoming data, it returns as acknowledge signal to Si.
- Asynchronous pipelines are useful in designing communication channels in messagepassing multicomputers.

Asynchronous Pipeline



Total time required

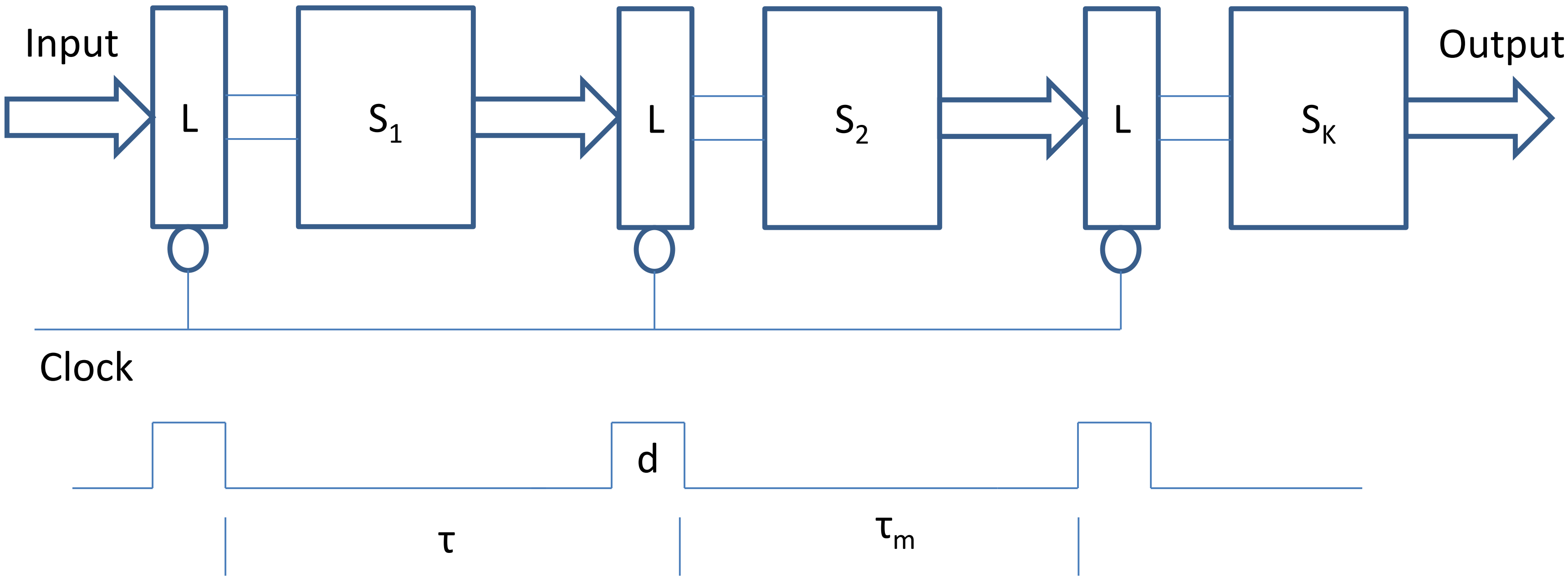
$T_k = [\ K + (n - 1\ )]\ \tau$

Synchronous Pipeline

- Clocked latches are used to interface between stages. The latches are made with master-slave flip-flop, which can isolate inputs from outputs. Upon the arrival of a clock pulse, all latches transfer data to the next stage simultaneously.
- The pipeline stages are combinational logic circuits. It is desired to have approximately equal delays in all stages.

Synchronous Pipeline

- These delays determine the clock period and thus the speed of the pipeline.



Four stage pipeline

	1	**2**	**3**	**4**
S_1	X			
S_2		X		
S_3			X	
S_4				X

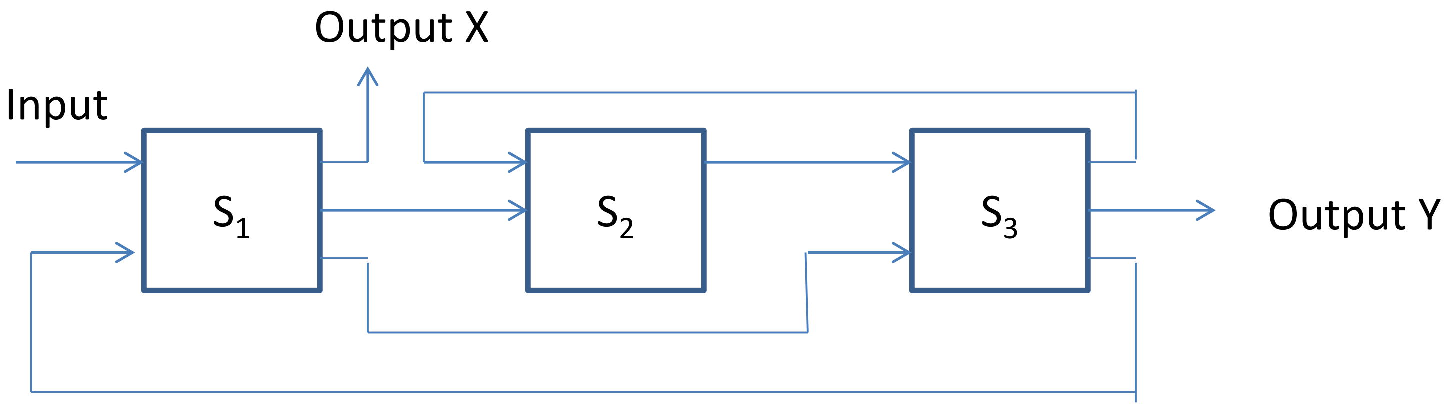
S_i = stage I
L = Latch
τ = Clock period

τ_m = Maximum stage delay
d = Latch delay
ACK = Acknowledge signal

Nonlinear Pipeline Processors

- A dynamic pipeline can be reconfigured to perform variable functions at different times.
- Multiple reservation tables can be generated for the evaluation of different functions.
- Each reservation table displays the time-space flow of data through the pipeline for one function evaluation. There is a many-to-many mapping between various pipeline configurations and different reservation tables.

Nonlinear Pipeline Processors



A three stage pipeline

Latency and: is the number of time units (clock cycles) between two initiations of a pipeline

Must be Collision free Scheduling

	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**
S_1	X					X		X
S_2		X		X				
S_3			X		X		X	

Reservation table for function X

	1	**2**	**3**	**4**	**5**	**6**
S_1	Y				Y	
S_2			Y			
S_3		Y		Y		Y

Reservation table for function Y

Pipeline

Dr. Mohammed Abdulridha Hussain

Pipeline Performance measures

- In the following analysis, we provide three performance measures for the goodness of a pipeline. These are the Speed-up S(n), Throughput U(n), and Efficiency E(n). It should be noted that in this analysis we assume that the unit time T = t units.

1. Speed-up S(n)

- Consider the execution of m tasks (instructions) using n-stages (units) pipeline. As can be seen, n + m - 1 time units are required to complete m tasks.
- $Speed - up\ S(n) = \frac{Time\ using\ sequential\ processing}{Time\ using\ pipeline\ processing}$

$$= \frac{m \times n \times t}{(n+m-1) \times t}$$

$$= \frac{m \times n}{n+m-1}$$

- $\lim_{m \to \infty} S(n) = n$

2. Throughput U(n)

- $Theoughput\ U(n) = no.of\ tasks\ executed\ per\ unit\ time = \frac{m}{(n+m-1)\times t}$
- $\lim_{m\to\infty} U(n) = 1\ assuming\ that\ t = 1\ unit\ time$

3. Efficiency E(n)

- $Efficiency\ E(n) = Ratio\ of\ the\ actual\ speed - up\ to\ the\ maximum\ speed - up = \frac{Speed-up}{n} = \frac{m}{n+m-1}$

- $\lim_{m\to\infty} E(n) = 1$

Pipeline “Stall” Due to Instruction Dependency

- Instruction dependency refers to the case whereby fetching of an instruction depends on the results of executing a previous instruction. Instruction dependency manifests itself in the execution of a conditional branch instruction. the next instruction to fetch will not be known until the result of executing instruction is known.

- **Example 1 (PDF page 205)**

Pipeline "Stall" Due to Data Dependency

- Data dependency in a pipeline occurs when a source operand of instruction I_i depends on the results of executing a preceding instruction, I_j, $i > j$. It should be noted that although instruction I_i can be fetched, its operand(s) may not be available until the results of instruction I_j are stored.

- **Example 2 (PDF page 206)**
- **Example 3(PDF page 208)**

Methods Used to Prevent Fetching the Wrong Instruction or Operand

- Use of NOP (No Operation) This method can be used in order to prevent the fetching of the wrong instruction, in case of instruction dependency, or fetching the wrong operand, in case of data dependency.

- **Example 4 (PDF page 210)**

Methods Used to Reduce Pipeline Stall

- Swap instructions
- One important condition that must be satisfied to produce correct results is that the set of instructions that are swapped with the branch instruction hold no data and/or instruction dependency relationship among them.