Introduction to Parallel Computing: Flynn's Classification, Instruction Pipelining

(Chapter 4-5-6)

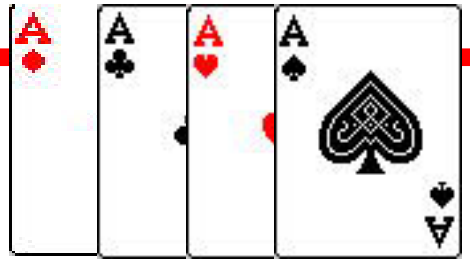
Topics Covered

- An Overview of Parallel Processing
- Parallelism in Uniprocessor Systems
- Organization of Multiprocessor
  - Flynn's Classification
  - System Topologies
  - MIMD System Architecture
  - Pipelining
  - Error Detection & Correction- Hamming Code
  - Distributed System & IoT

An Overview of Parallel Processing

- What is parallel processing?
  - Parallel processing is a method to improve computer system performance by executing two or more instructions simultaneously.
- The goals of parallel processing.
  - One goal is to reduce the “wall-clock” time or the amount of real time that you need to wait for a problem to be solved.
  - Another goal is to solve bigger problems that might not fit in the limited memory of a single CPU.

An Analogy of Parallelism



The task of ordering a shuffled deck of cards by suit and then by rank can be done faster if the task is carried out by two or more people. By splitting up the decks and performing the instructions simultaneously, then at the end combining the partial solutions you have performed parallel processing.

Another Analogy of Parallelism

Another analogy is having several students grade quizzes simultaneously. Quizzes are distributed to a few students and different problems are graded by each student at the same time. After they are completed, the graded quizzes are then gathered and the scores are recorded.

Parallelism in Uniprocessor Systems

- It is possible to achieve parallelism with a uniprocessor system.
  - Some examples are the instruction pipeline, arithmetic pipeline, I/O processor.
- Note that a system that performs different operations on the same instruction is not considered parallel.
- Only if the system processes two different instructions simultaneously can it be considered parallel.

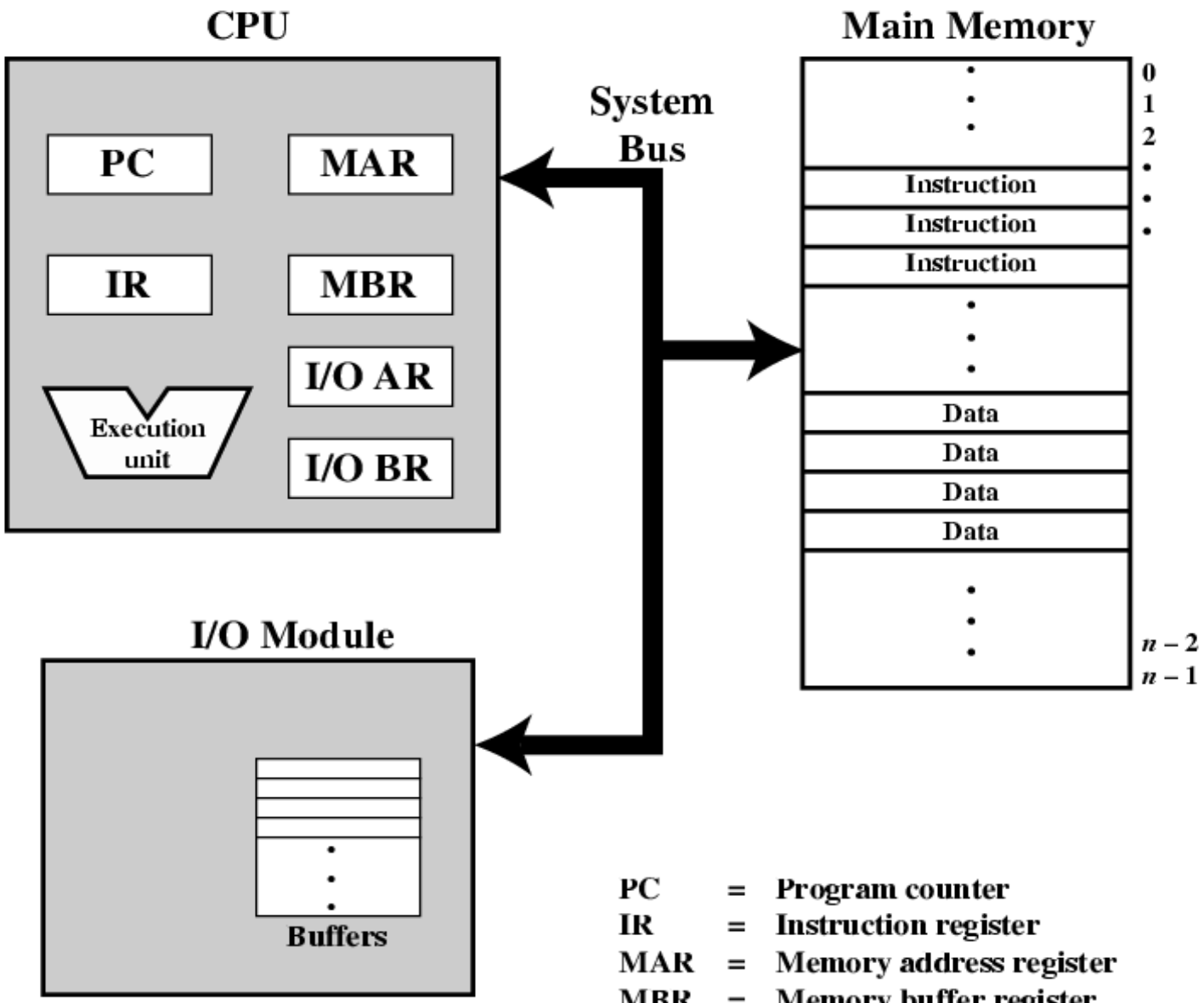
Organization of Multiprocessor Systems

- Flynn's Classification
  - Was proposed by researcher Michael J. Flynn in 1966.
  - It is the most commonly accepted taxonomy of computer organization.
  - In this classification, computers are classified by whether it processes a single instruction at a time or multiple instructions simultaneously, and whether it operates on one or multiple data sets.

Computer Components

- The Control Unit and the Arithmetic and Logic Unit constitute the
  - **Central Processing Unit**
- Data and instructions need to get into the system and results out
  - **Input/output**
- Temporary storage of code and results, is needed
  - **Main memory**

Computer Components:
Top Level View



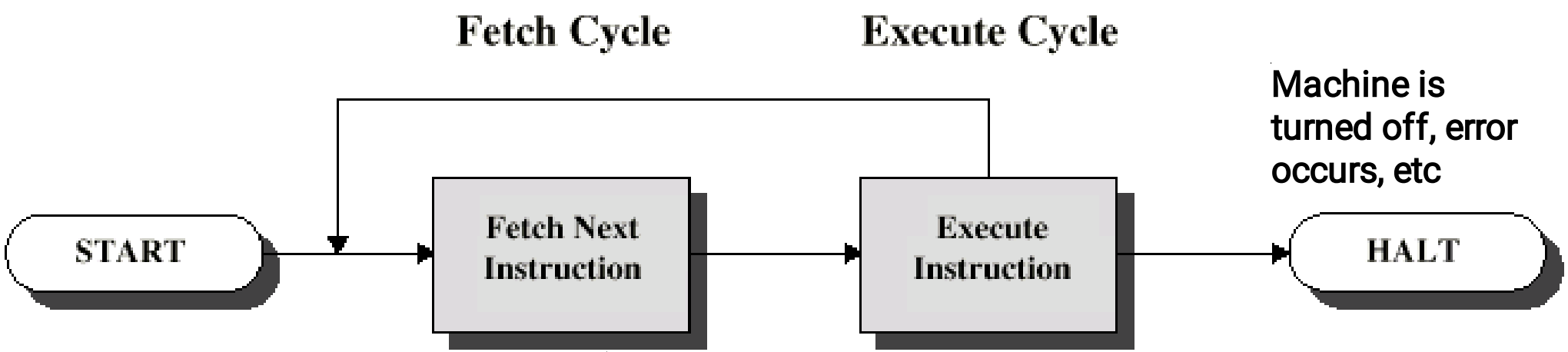
PC = Program counter
IR = Instruction register
MAR = Memory address register
MBR = Memory buffer register
I/O AR = Input/output address register
I/O BR = Input/output buffer register

Computer Function- Instruction Cycle

- basic function is execution of a program, which consists of a set of instructions stored in memory
- each instruction processing (Inst. cycle) consists of two steps:
  - 1. Fetch Instruction (Fetch Cycle)

    - from memory using PC reg. that holds address of inst. to be fetched next inc PC after each inst. fetch, for in sequence execution of inst., fetched inst. loaded into IR reg.,
  - 2. Execute this Instruction (Execute Cycle)

    - involve several operations, depends on nature of instructions



Instruction Cycle

1. Fetch Cycle

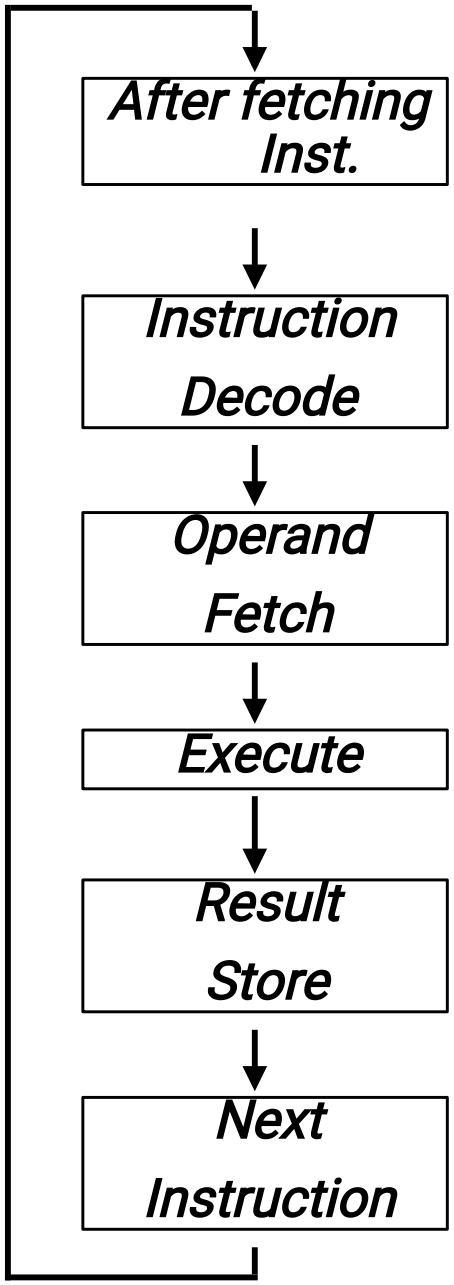
- At beginning, Processor fetches instruction from memory location pointed to by PC
- Program Counter (PC) holds address of next instruction to fetch
- Increment PC, so that it will fetch next instruction in sequence
- Instruction loaded into Instruction Register (IR)
- Processor interprets instruction and performs required actions (that fall-into 4 categories)

2. Execute Cycle

The processor interprets inst. & performs required actions, that fall into four categories:

- **Processor-memory**
  - Data transfer between CPU and main memory
- **Processor I/O**
  - Data transfer between CPU and I/O module
- **Data processing**
  - Some arithmetic or logical operation on data
- **Control**
  - Alteration of sequence of operations
  - e.g. jump, i.e. fetch an inst from location 49, which specifies that next inst. be from 82, by setting PC to 82, so next inst is from 82 rather than 50 location

Execution Cycle- An Example



Obtain instruction from program storage

Determine required actions and instruction size

Locate and obtain operand data

Compute result value or status

Deposit results in storage for later use

Determine successor instruction;

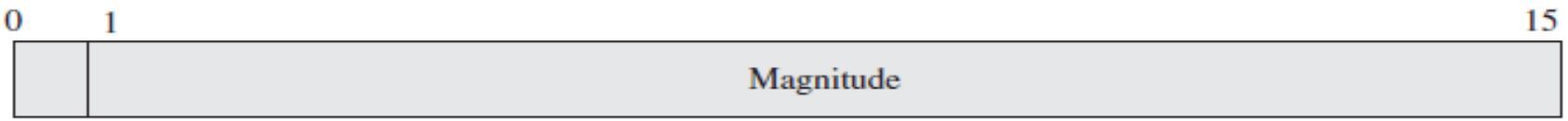
e.g. to add two no.

X = Y + Z;

Example of Program Execution

The program fragment shown adds the contents of the memory word at address 940 to the contents of the memory word at address 941 and stores the result in the latter location:

(a) Instruction format



(b) Integer format

Program counter (PC) = Address of instruction
Instruction register (IR) = Instruction being executed
Accumulator (AC) = Temporary storage

(c) Internal CPU registers

0001 = Load AC from memory
0010 = Store AC to memory
0101 = Add to AC from memory

(d) Partial list of opcodes

Example of Program Execution

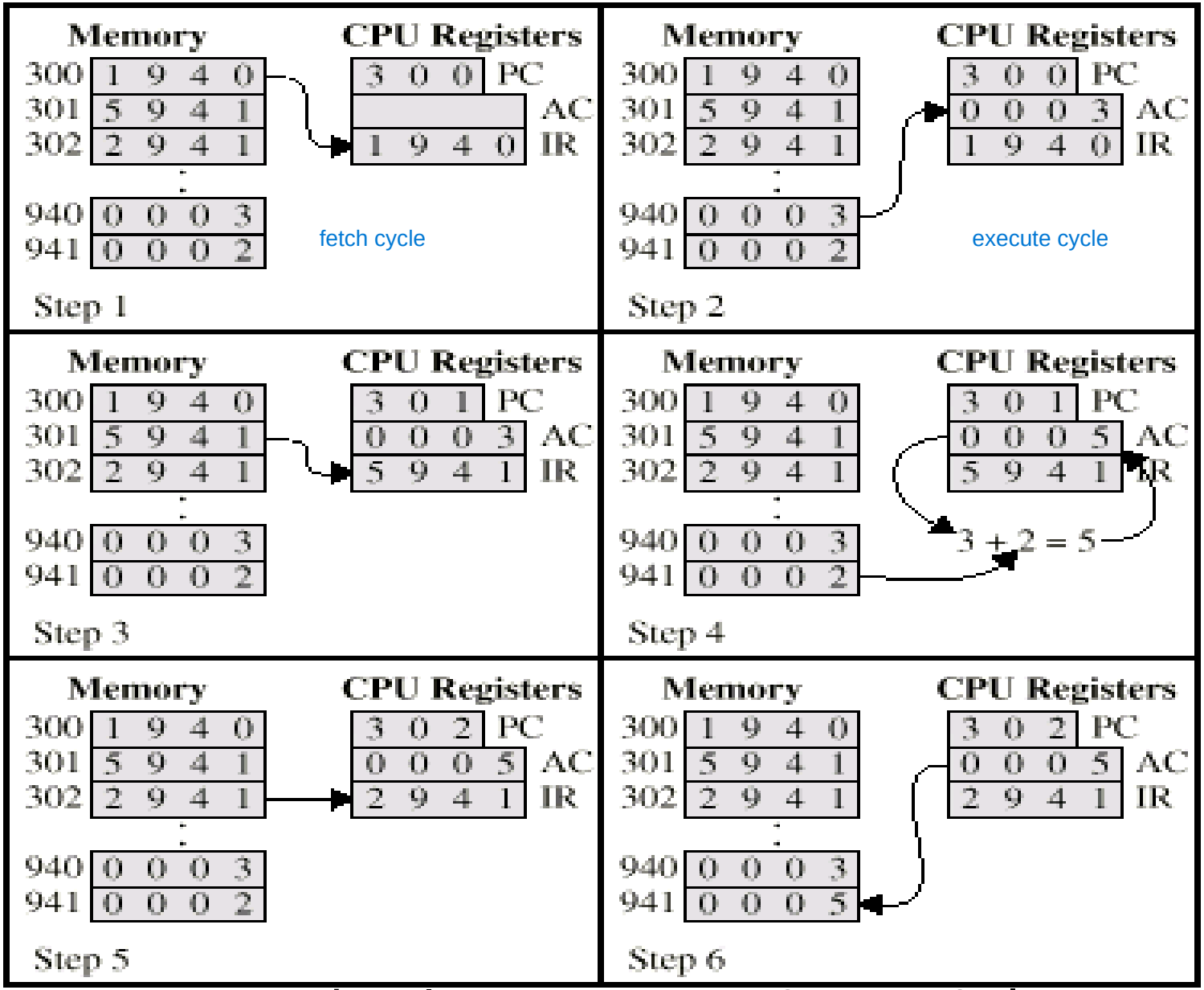
instructions

Data

Instruction 1

Instruction 2

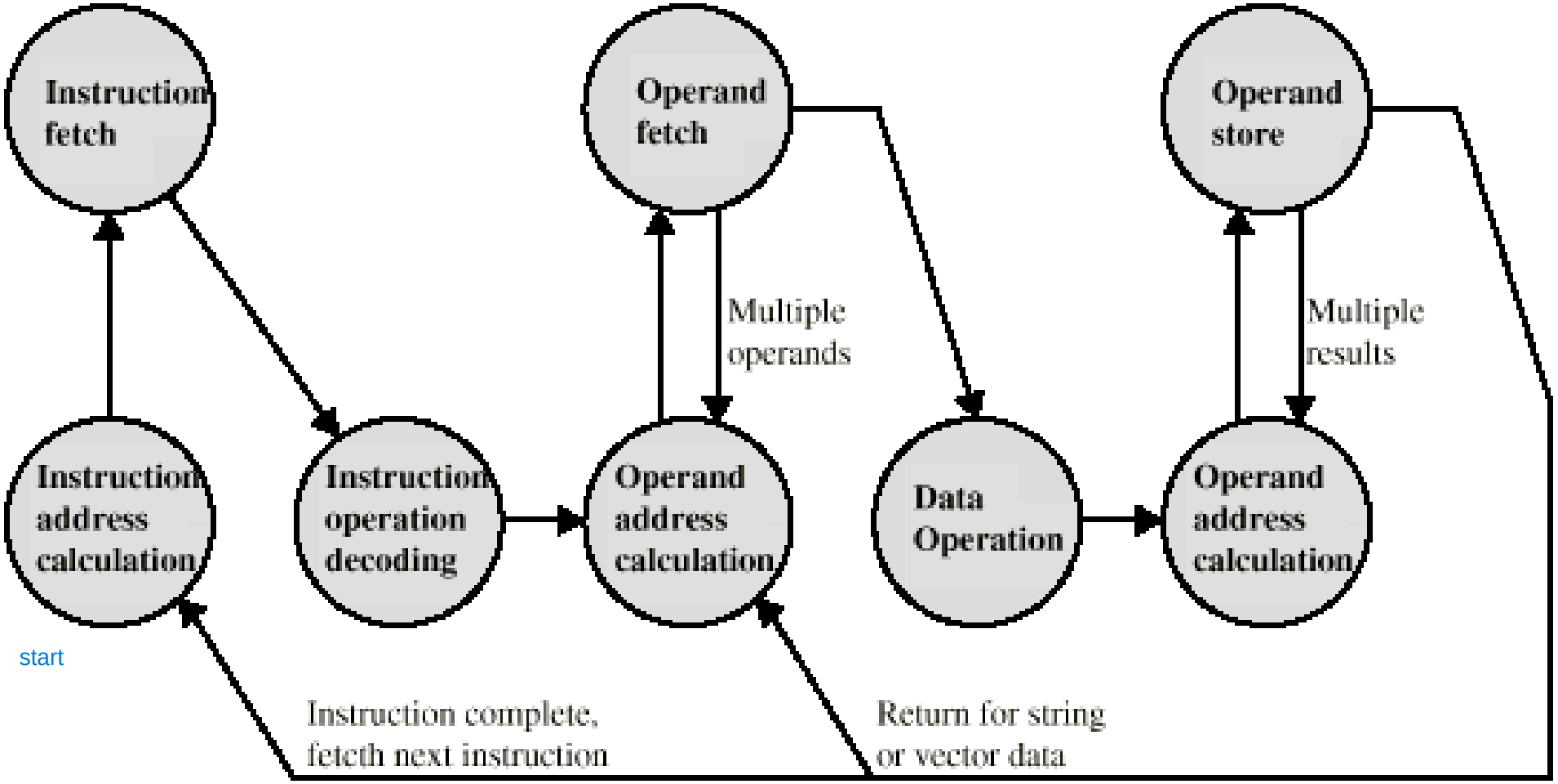
Instruction 3



3 Fetch Cycles

3 Execute Cycles

Instruction Cycle - State Diagram



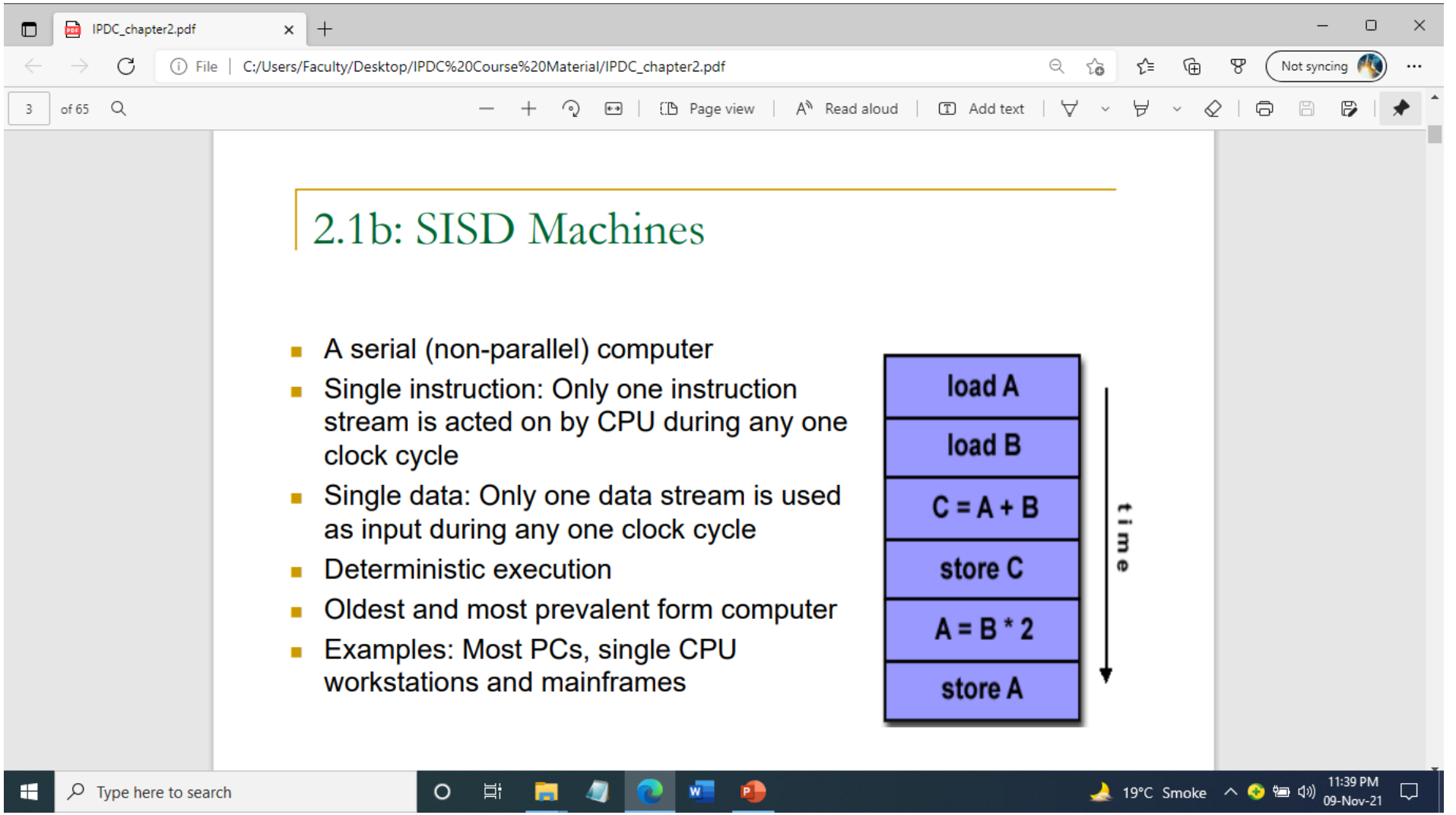
FLYNN'S CLASSIFICATION

- 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

Single Instruction, Single Data (SISD)

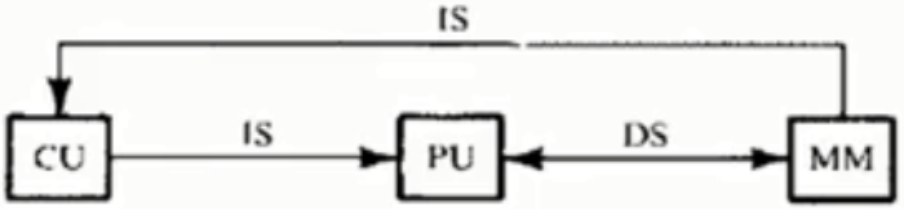
- SISD machines executes a single instruction on individual data values using a single processor.
- Based on traditional Von Neumann uniprocessor architecture, instructions are executed sequentially or serially, one step after the next.
- Until most recently, most computers are of SISD type.

SISD


IPDC_chapter2.pdf
File | C:/Users/Faculty/Desktop/IPDC%20Course%20Material/IPDC_chapter2.pdf
Not syncing
3 of 65
Page view
Read aloud
Add text
2.1b: SISD Machines
A serial (non-parallel) computer
Single instruction: Only one instruction stream is acted on by CPU during any one clock cycle
Single data: Only one data stream is used as input during any one clock cycle
Deterministic execution
Oldest and most prevalent form computer
Examples: Most PCs, single CPU workstations and mainframes
load A
load B
C = A + B
store C
A = B * 2
store A
time
Type here to search
19°C Smoke
11:39 PM
09-Nov-21

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



Single Instruction, Multiple Data (SIMD)

2.2a: SIMD Machines (I)

- A type of parallel computers
- Single instruction: All processor units execute the same instruction at any give clock cycle
- Multiple data: Each processing unit can operate on a different data element
- It typically has an instruction dispatcher, a very high-bandwidth internal network, and a very large array of very small-capacity instruction units
- Best suitable for specialized problems characterized by a high degree of regularity, e.g., image processing
- Two varieties: Processor Arrays and Vector Pipelines
- Examples: Connection Machines, MasPar-1, MasPar-2;
- IBM 9000, Cray C90, Fujitsu VP, etc

Single Instruction, Multiple Data (SIMD)

- An SIMD machine executes a single instruction on multiple data values simultaneously using many processors.
- Since there is only one instruction, each processor does not have to fetch and decode each instruction. Instead, a single control unit does the fetch and decoding for all processors.
- SIMD architectures include array processors.
- The control unit (CU) of the central processing unit (CPU) regulates & integrates the operations of the computers. It selects & retrieves instructions from the main memory in proper sequence & interprets them so as to activate the other functional elements of the system at the appropriate moment….

Single Instruction, Multiple Data (SIMD)

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

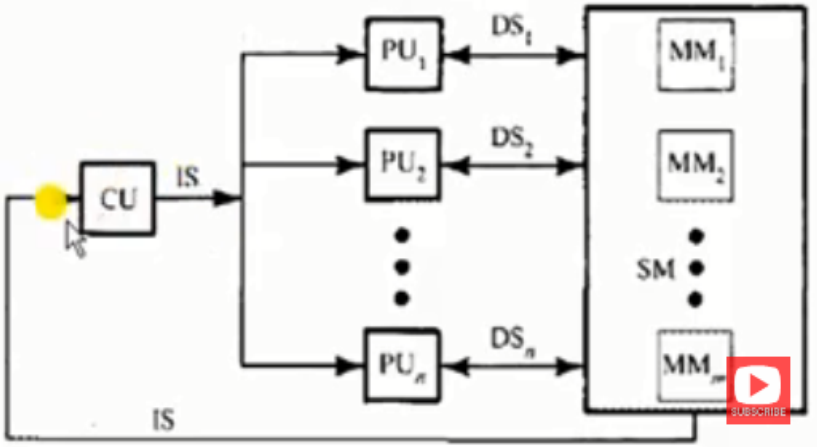
Multiple data: Each processing unit can operate on a different data element

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

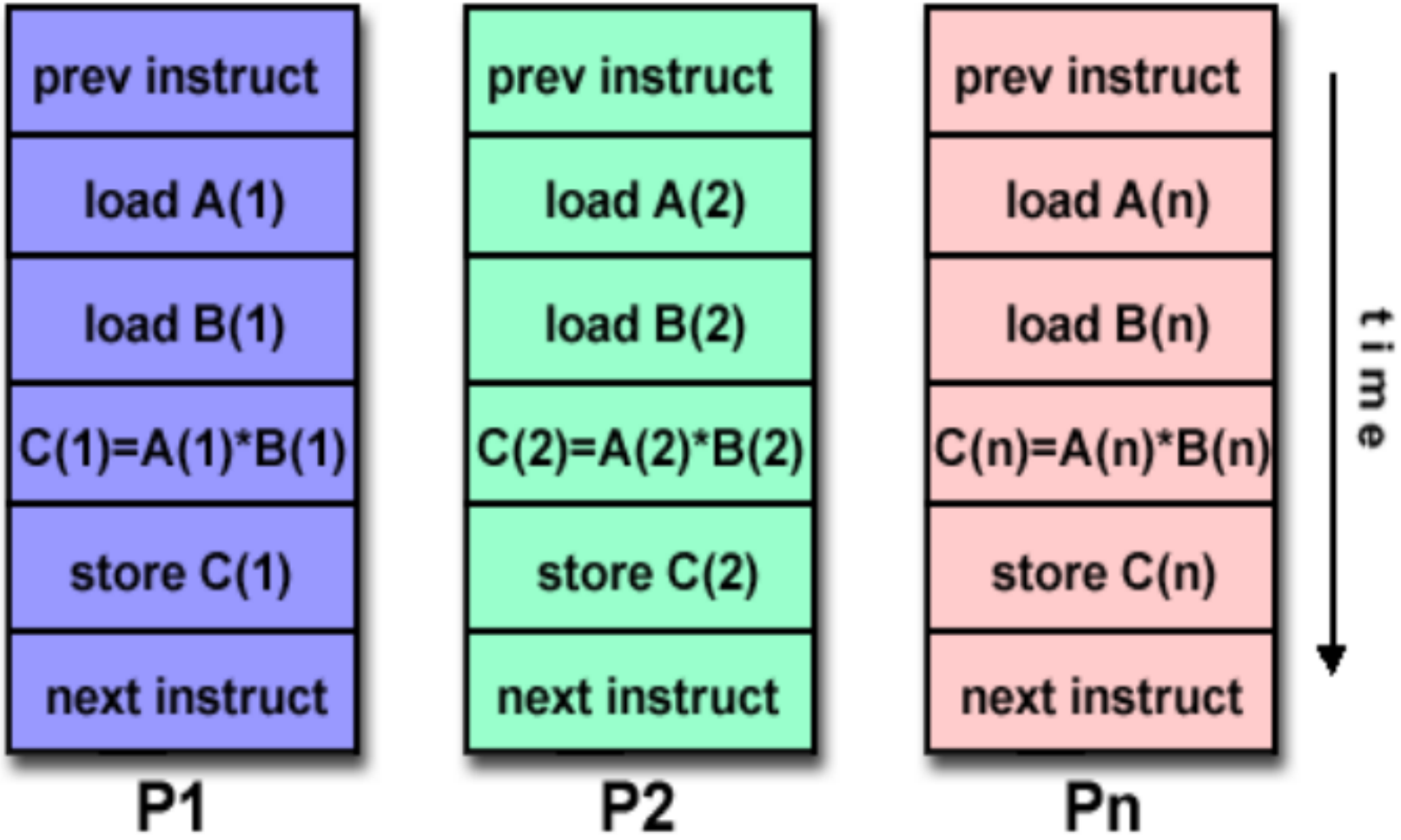
Well suited to scientific computing since they involve lots of vector and matrix operations.

Example: Array Processor sand Vector Pipelines

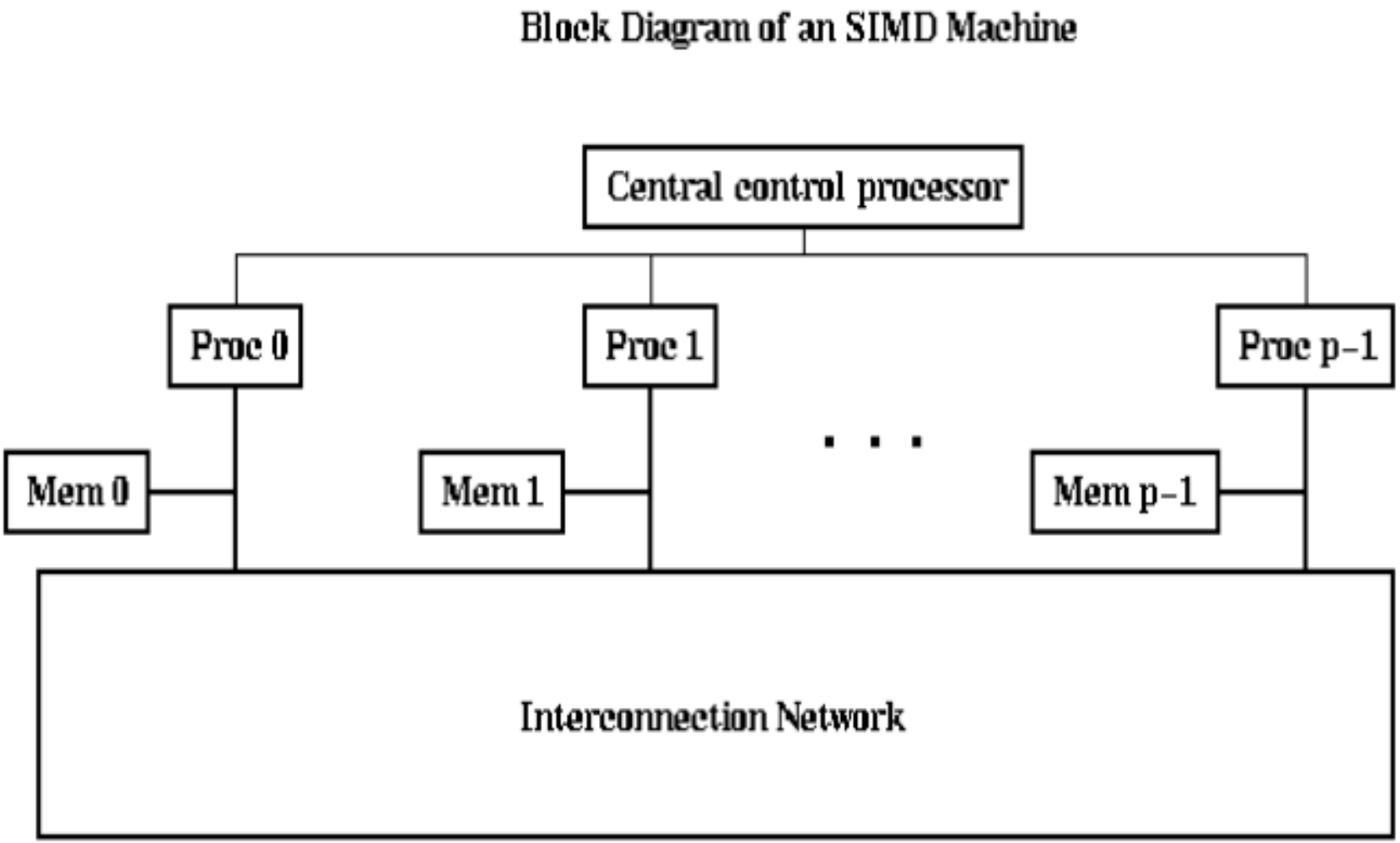
- Array processors: ILLIAC-IV, MPP
- Vector Pipelines: IBM 9000, Cray X-MP, Y-MP & C90



2.2b: SIMD Machines (II)



2.2c: SIMD Machine (III)



2.3a: MISD Machines (I)

- A single data stream is fed into multiple processing units
- Each processing unit operates on the data independently via independent instruction streams
- Very few actual machines: CMU's C.mmp computer (1971)
- Possible use: multiple frequency filters operating on a single signal stream

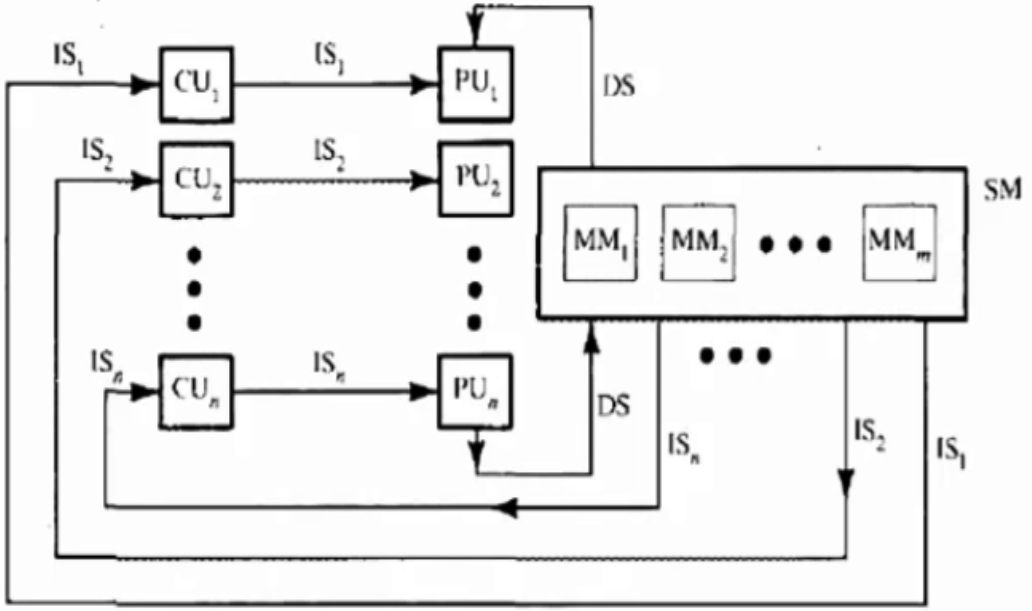
physics kei applications mei hoti use

Multiple Instruction, Single Data (MISD)

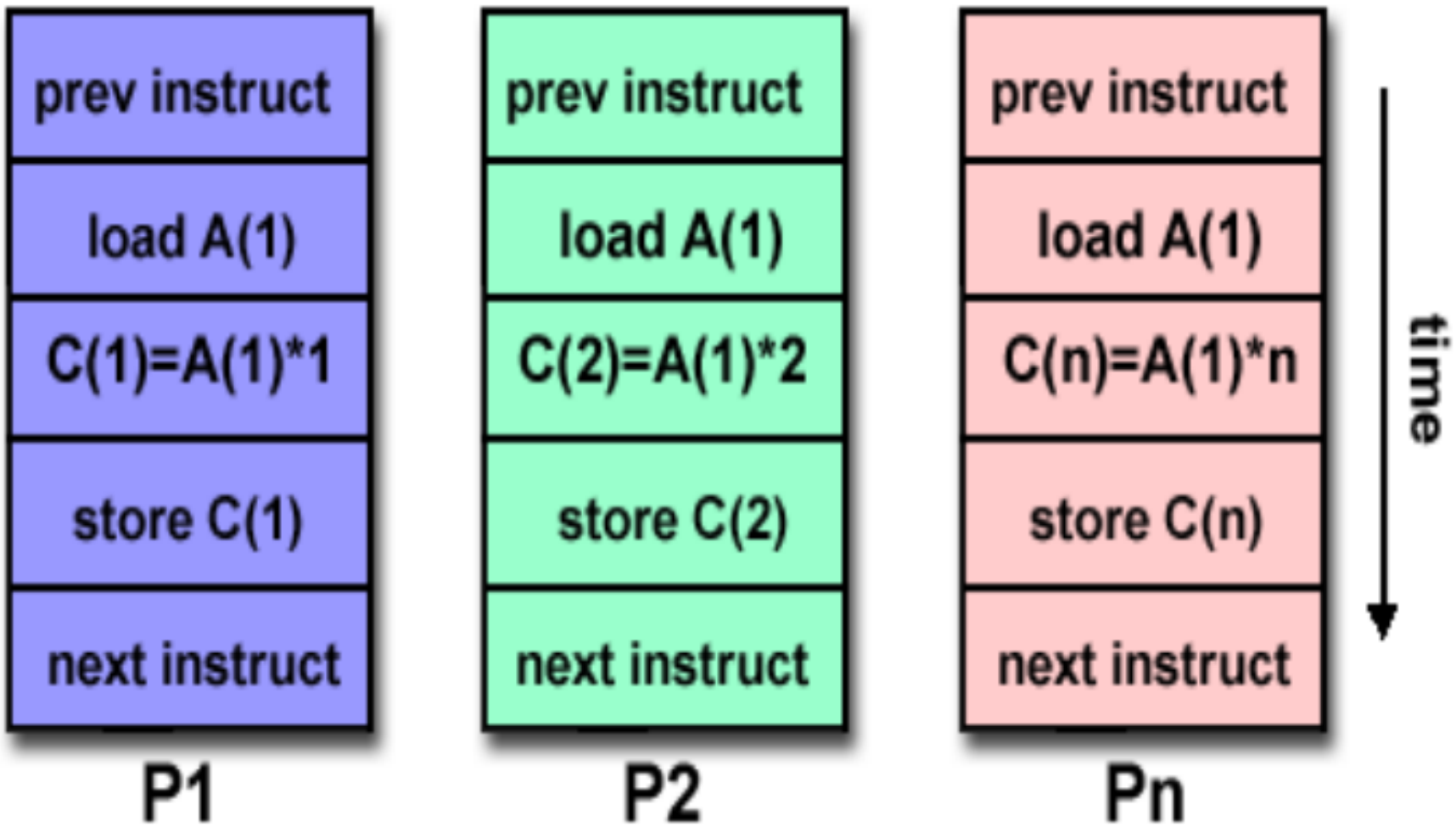
- This category does not actually exist. This category was included in the taxonomy for the sake of completeness.

MISD Computers

- In MISD, multiple instructions operate on single data stream
- An MISD computing system is capable of executing different instructions 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 Arrays



2.3b: MISD Machines (II)



Multiple Instruction, Multiple Data (MIMD)

- MIMD machines are usually referred to as multiprocessors or multicomputers.
- It may execute multiple instructions simultaneously, contrary to SIMD machines.
- Each processor must include its own control unit that will assign to the processors parts of a task or a separate task.
- It has two subclasses: Shared memory and distributed memory

2.4a: MIMD Machines (I)

- Multiple instruction: Every processor may execute a different instruction stream
- Multiple data: Every processor may work with a different data stream
- Execution can be synchronous or asynchronous, deterministic or non-deterministic
- Examples: most current supercomputers, grids, networked parallel computers, multiprocessor SMP computer

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

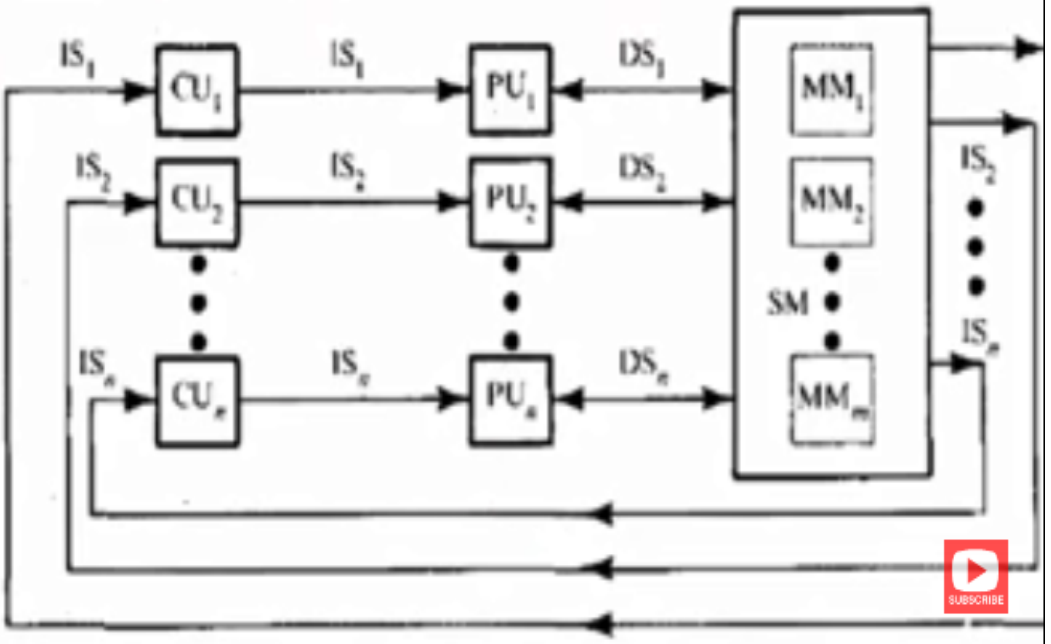
Multiple Data: Every processor may be working with a different data stream

MIMD systems are parallel computers capable of processing several programs simultaneously

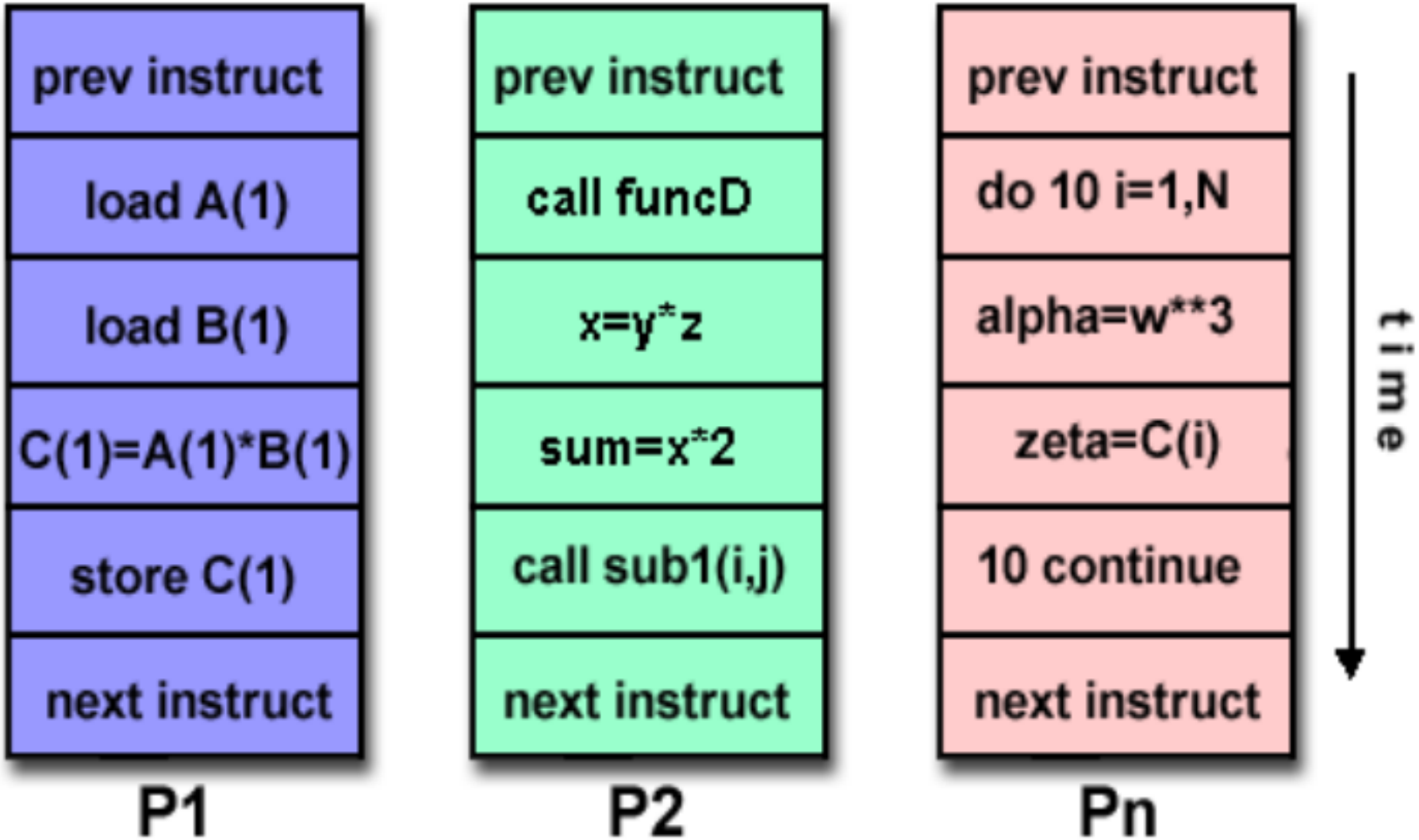
Can be categorized as loosely coupled or tightly coupled depending on sharing of data and control

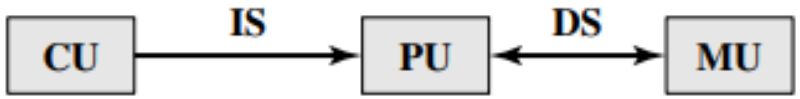
Example:

- 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

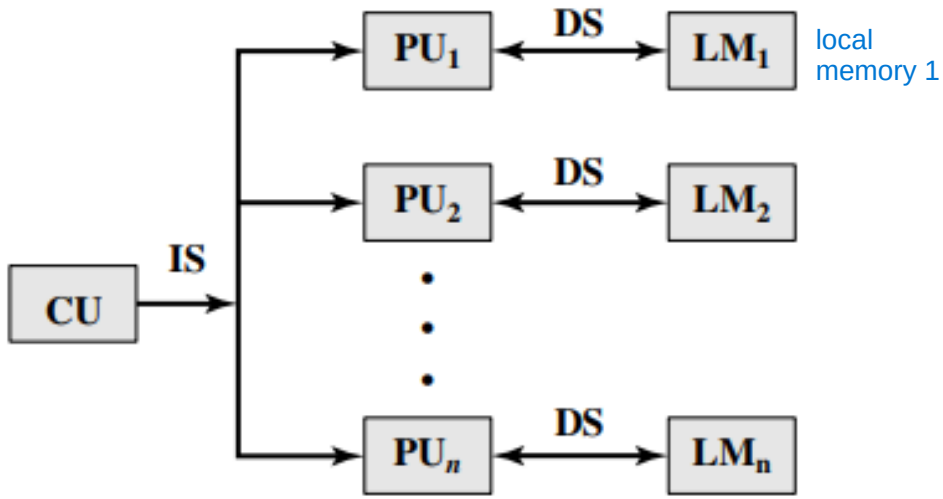


2.4b: MIMD Machines (II)



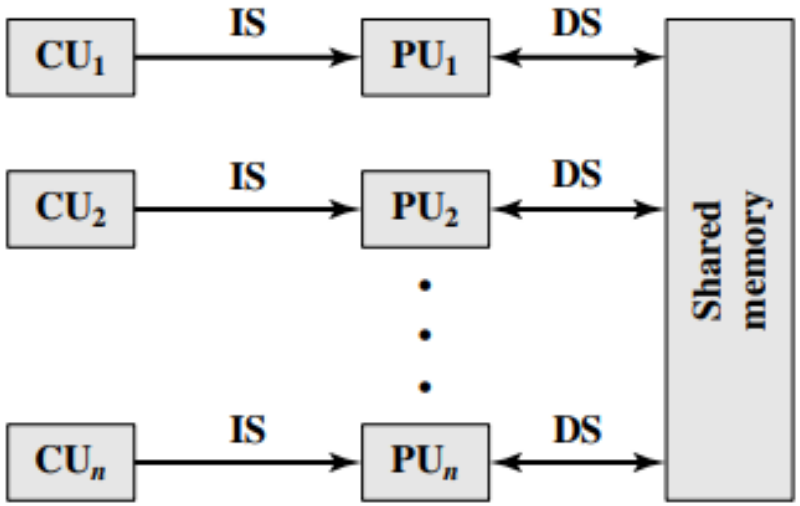


(a) SISD

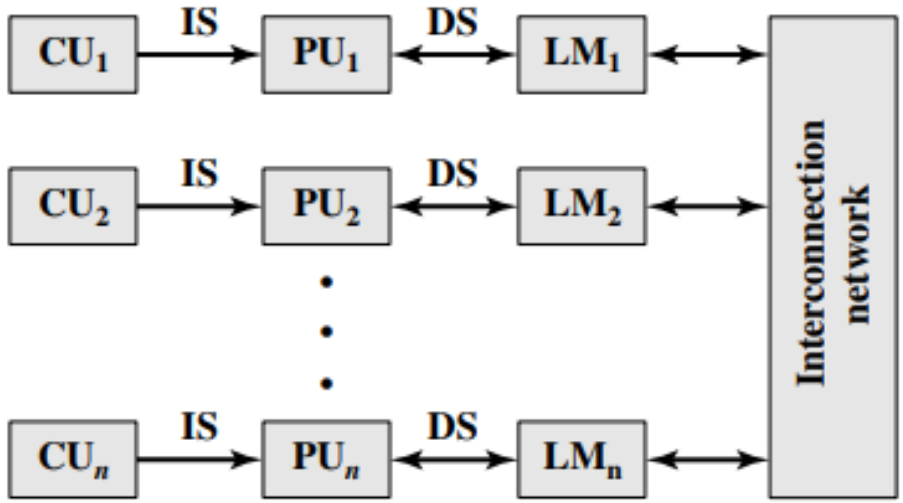


SIMD is restricted to use distributed memory

(b) SIMD (with distributed memory)



(c) MIMD (with shared memory)



(d) MIMD (with distributed memory)

CU = Control unit
IS = Instruction stream
PU = Processing unit
DS = Data stream
MU = Memory unit
LM = Local memory

SISD = Single instruction, single data stream
SIMD = Single instruction, multiple data stream
MIMD = Multiple instruction, multiple data stream

Figure 17.2 Alternative Computer Organizations

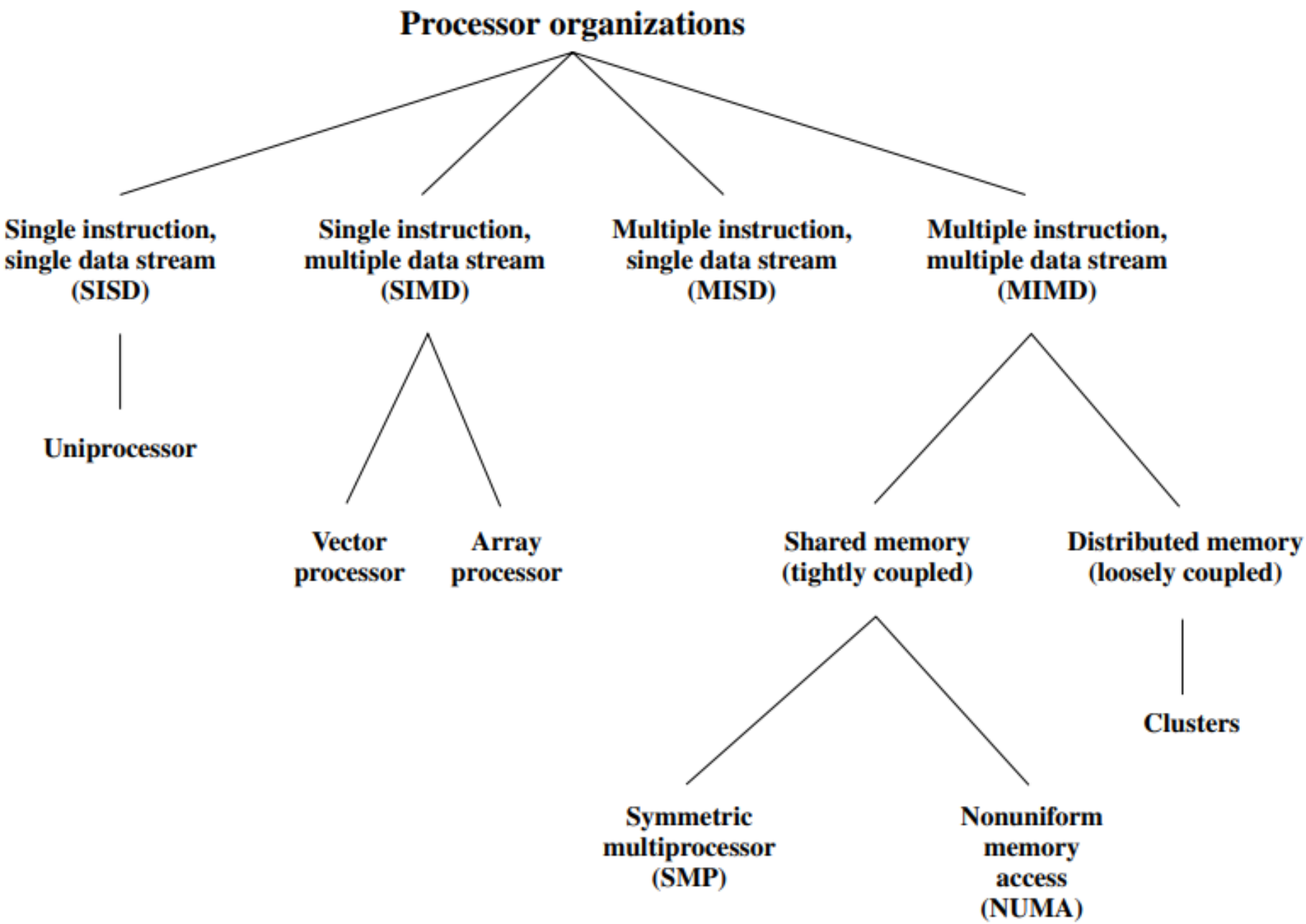
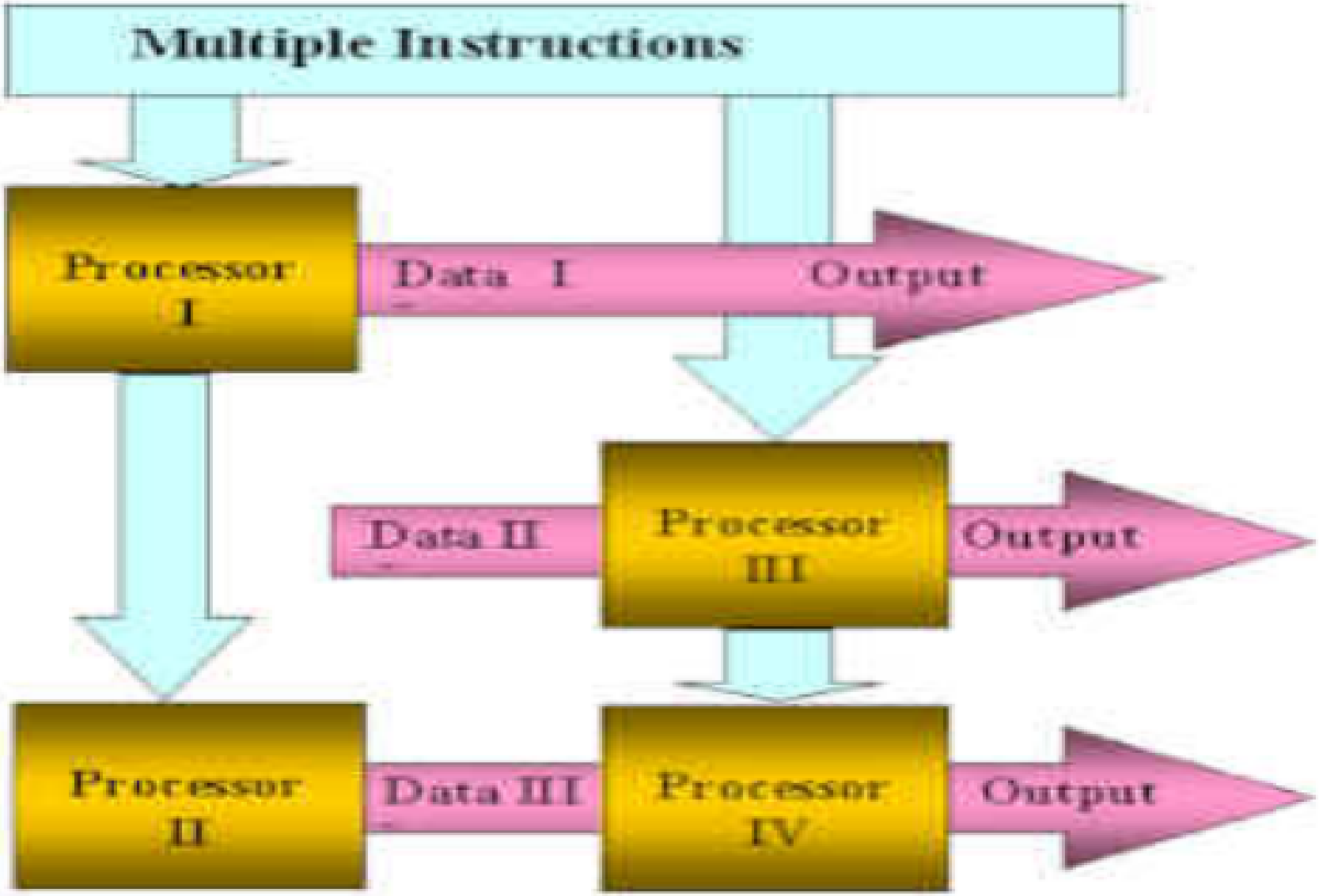
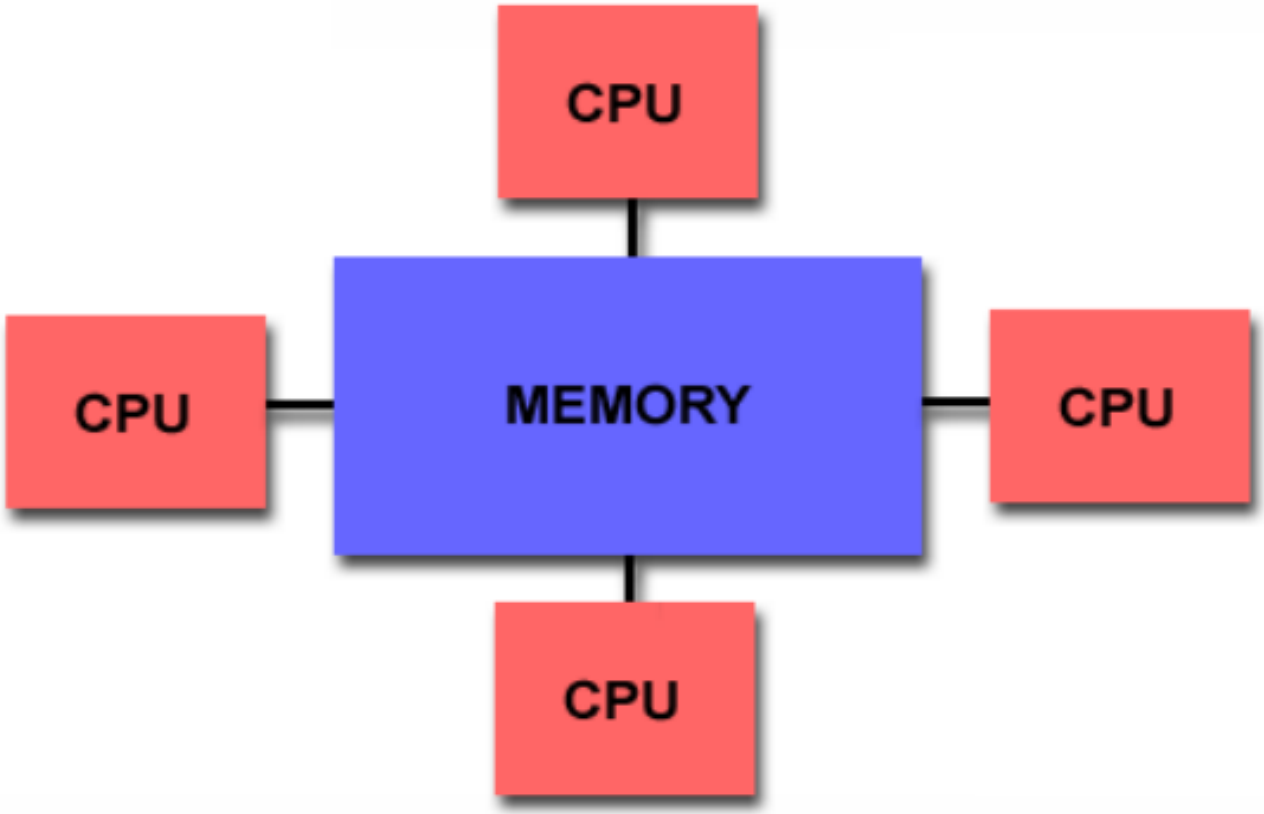


Figure 17.1 A Taxonomy of Parallel Processor Architectures

MIMD Machine



2.6b: Shared Memory Architecture



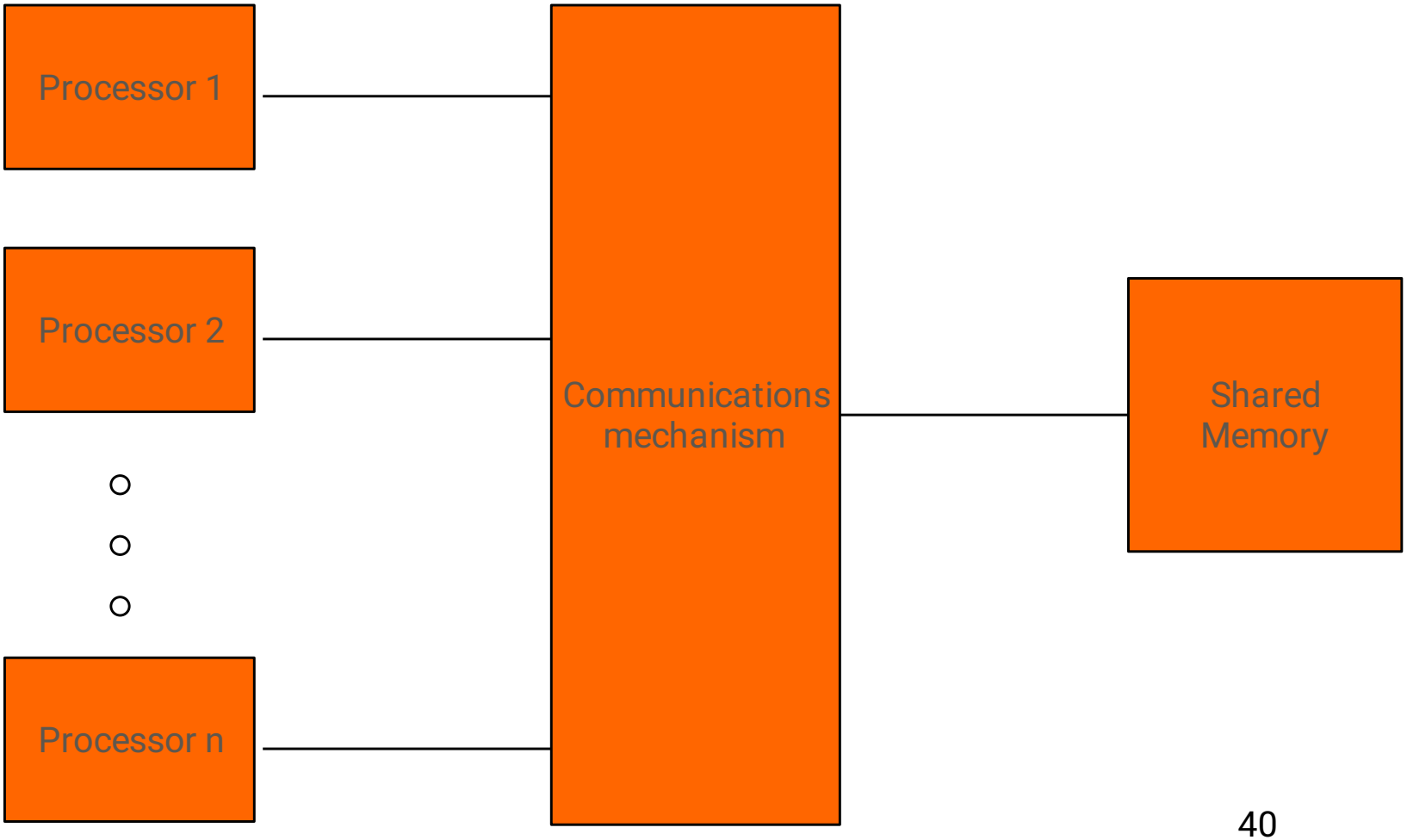
MIMD System Architectures

- Finally, the architecture of a MIMD system, contrast to its topology, refers to its connections to its system memory.
- A systems may also be classified by their architectures. Two of these are:
  - Uniform memory access (UMA)
  - Nonuniform memory access (NUMA)

Uniform memory access (UMA)

- The UMA is a type of symmetric multiprocessor, or SMP, that has two or more processors that perform symmetric functions. UMA gives all CPUs equal (uniform) access to all memory locations in shared memory. They interact with shared memory by some communications mechanism like a simple bus or a complex multistage interconnection network.

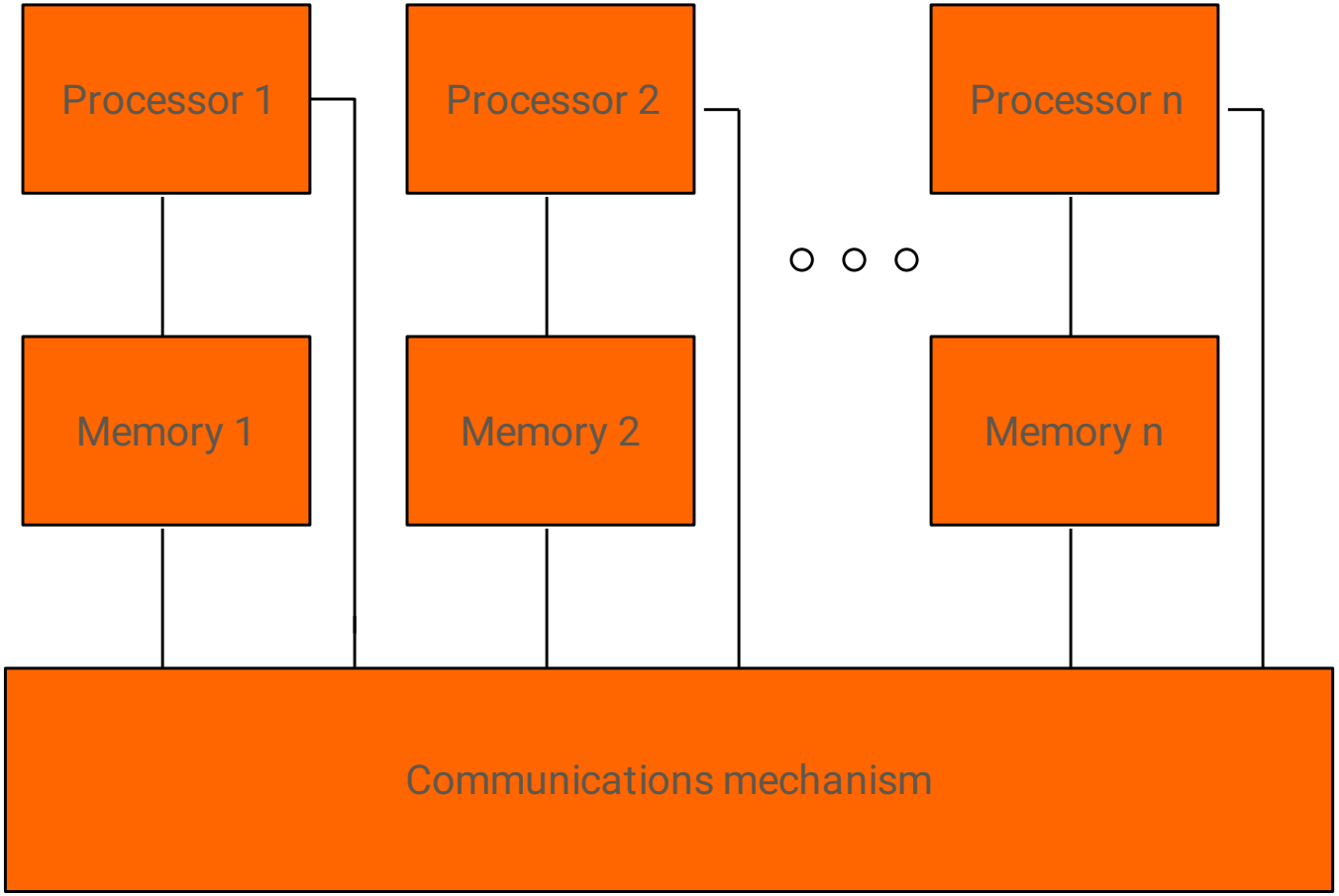
Uniform memory access (UMA) Architecture



Nonuniform memory access (NUMA)

- NUMA architectures, unlike UMA architectures do not allow uniform access to all shared memory locations. This architecture still allows all processors to access all shared memory locations but in a nonuniform way, each processor can access its local shared memory more quickly than the other memory modules not next to it.

Nonuniform memory access (NUMA) Architecture



2.6i: Advantages of Shared Memory Machines

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

2.6j: Disadvantages of Shared Memory Machines

- Lack of scalability between memory and CPUs: Adding processors can geometrically increase traffic on the shared memory-CPU path and for cache coherence management
- Programmer's responsibility for synchronization constructs (correct access to memory)
- Expensive to design shared memory computers with increasing numbers of processors

System Topologies

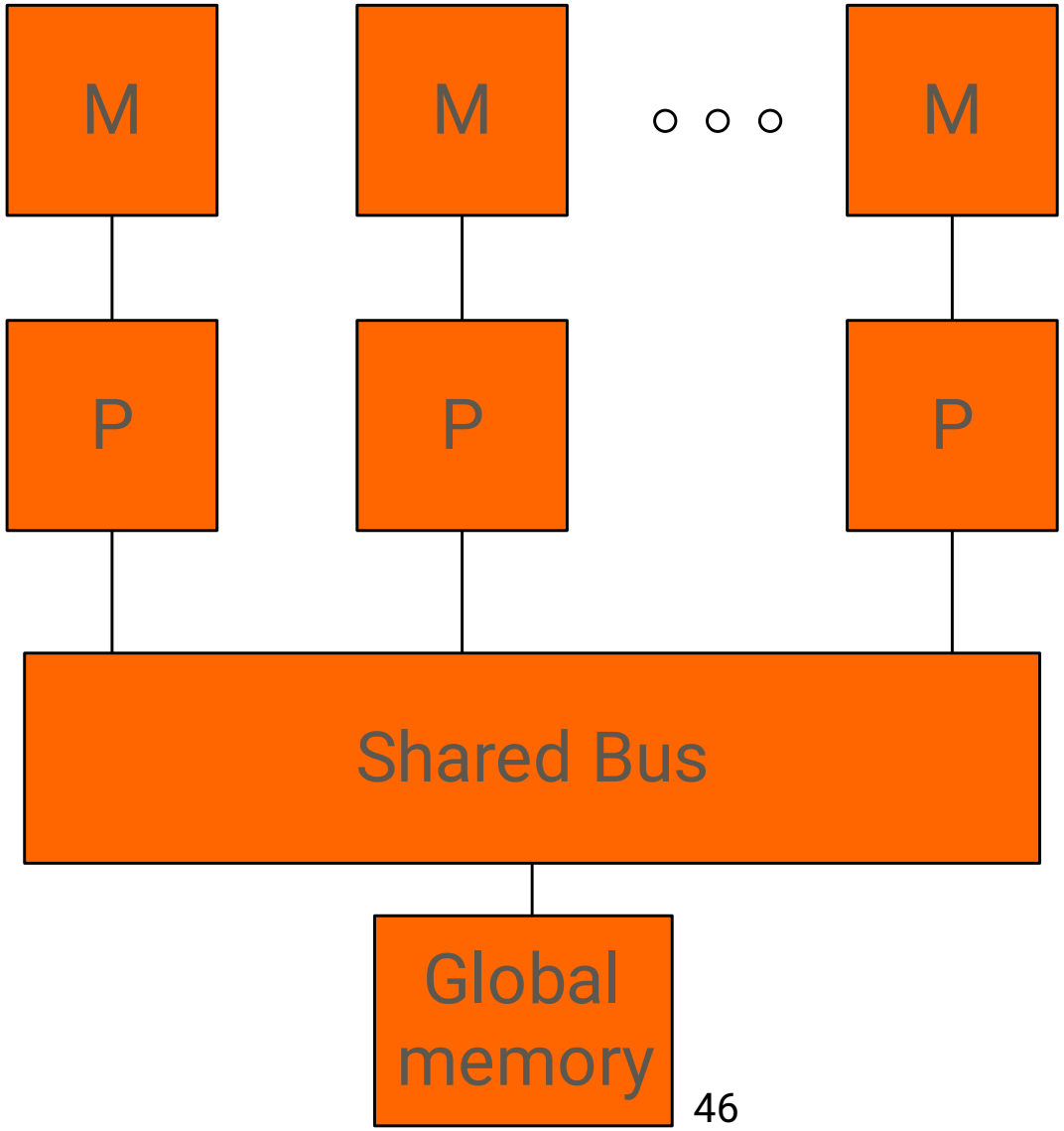
- Topologies
  - A system may also be classified by its topology.
  - A topology is the pattern of connections between processors.
  - The cost-performance tradeoff determines which topologies to use for a multiprocessor system.

Topology Classification

- A topology is characterized by its diameter, total bandwidth, and bisection bandwidth
  - Diameter – the maximum distance between two processors in the computer system.
  - Total bandwidth – the capacity of a communications link multiplied by the number of such links in the system

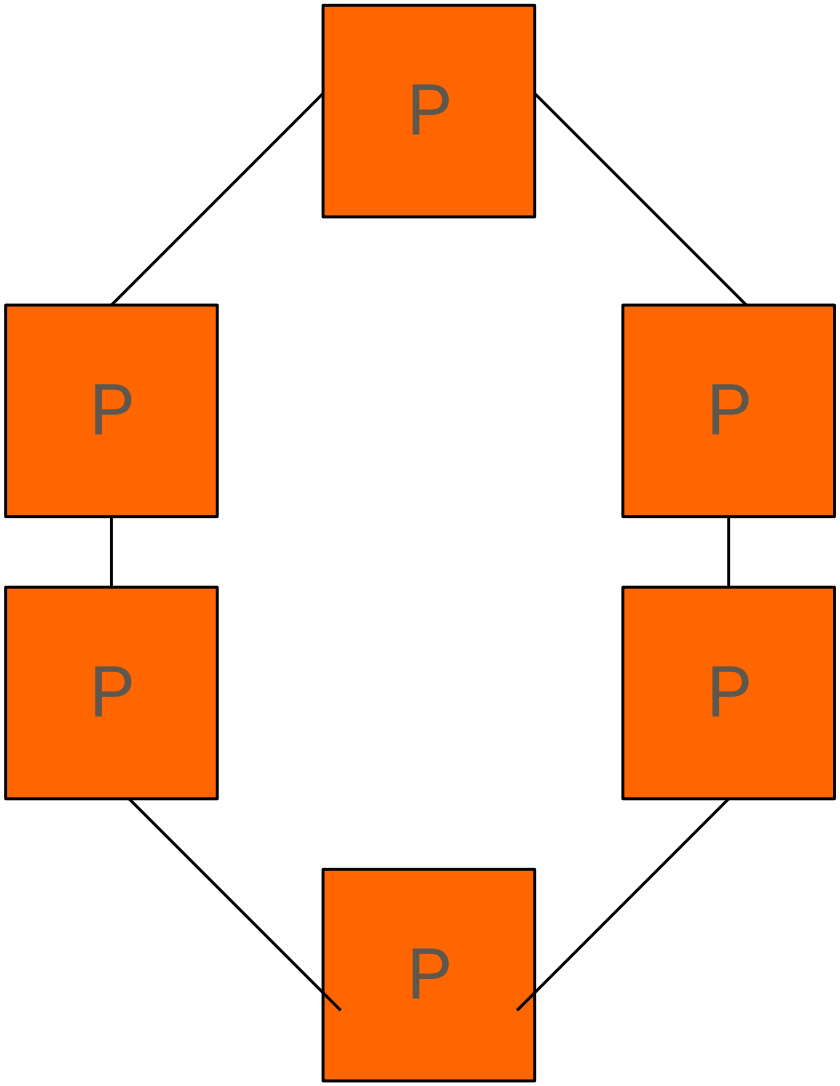
System Topologies

- Shared Bus Topology
  - Processors communicate with each other via a single bus that can only handle one data transmissions at a time.
  - In most shared buses, processors directly communicate with their own local memory.



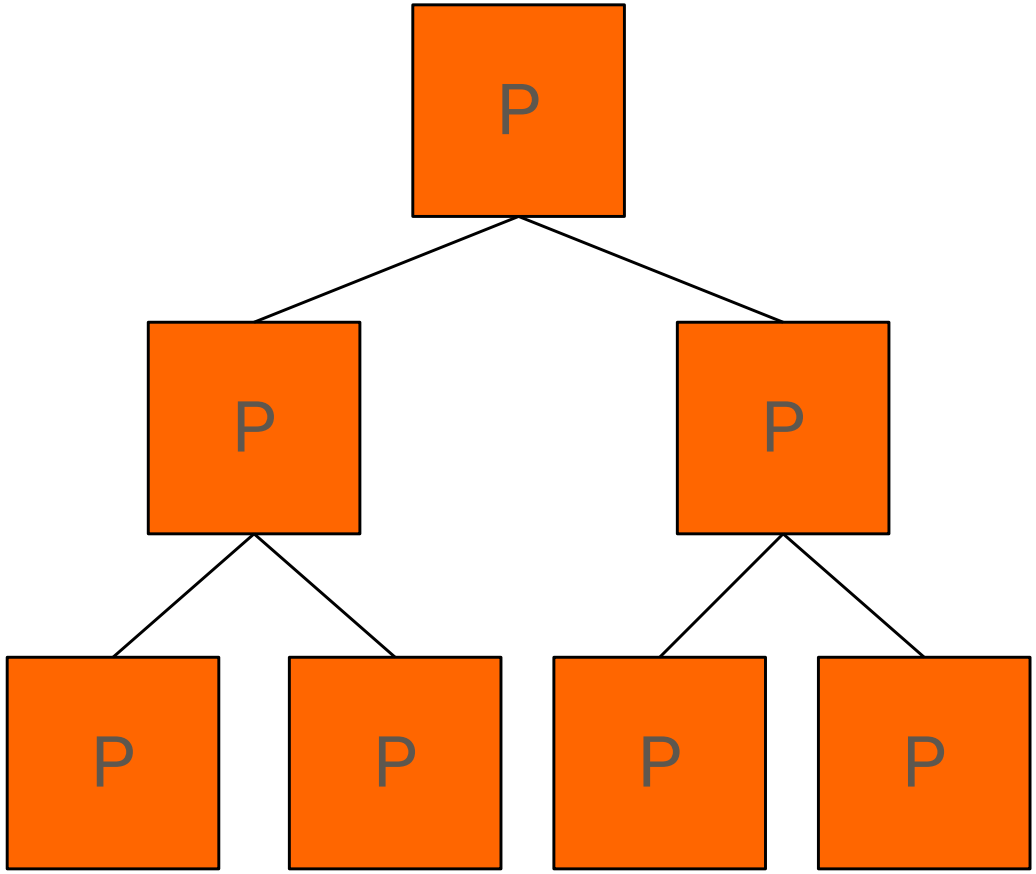
System Topologies

- Ring Topology
  - Uses direct connections between processors instead of a shared bus.
  - Allows communication links to be active simultaneously but data may have to travel through several processors to reach its destination.



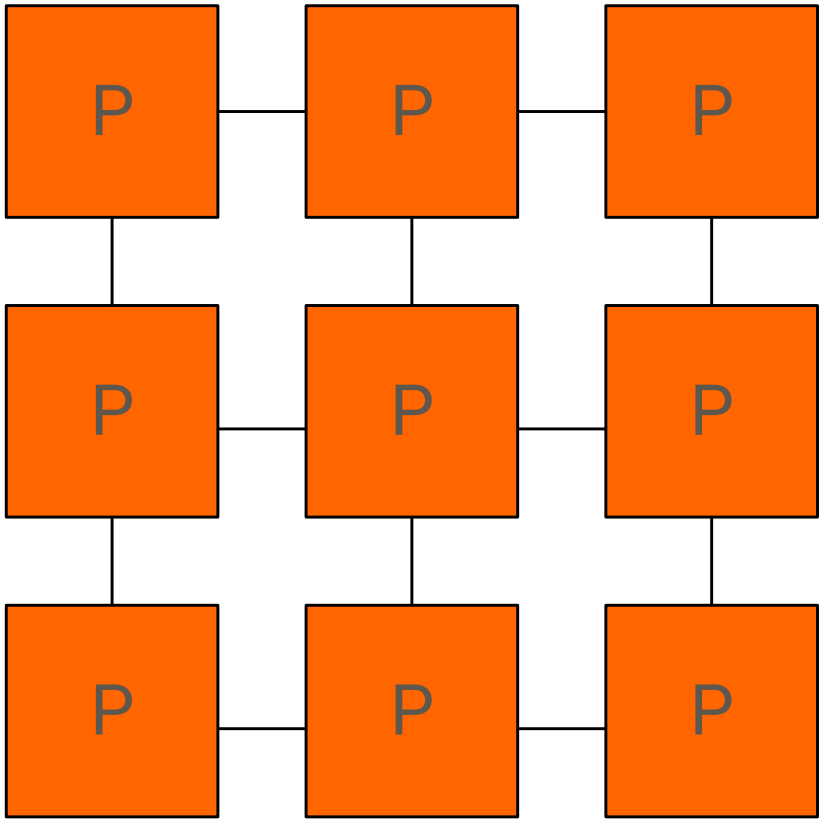
System Topologies

- Tree Topology
  - Uses direct connections between processors; each having three connections.
  - There is only one unique path between any pair of processors.



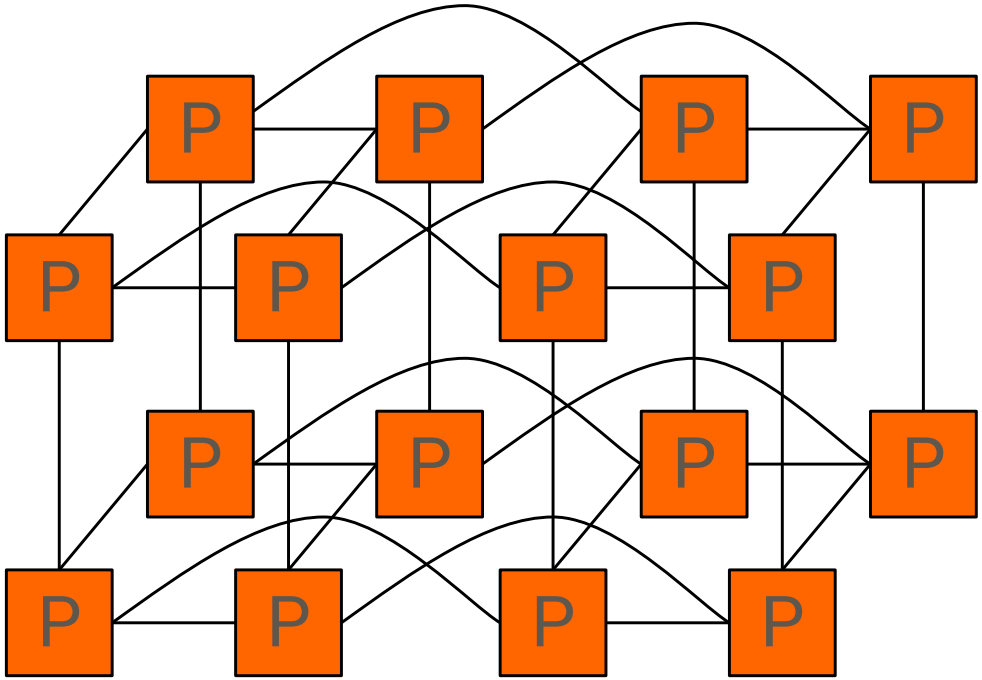
Systems Topologies

- Mesh Topology
  - In the mesh topology, every processor connects to the processors above and below it, and to its right and left.



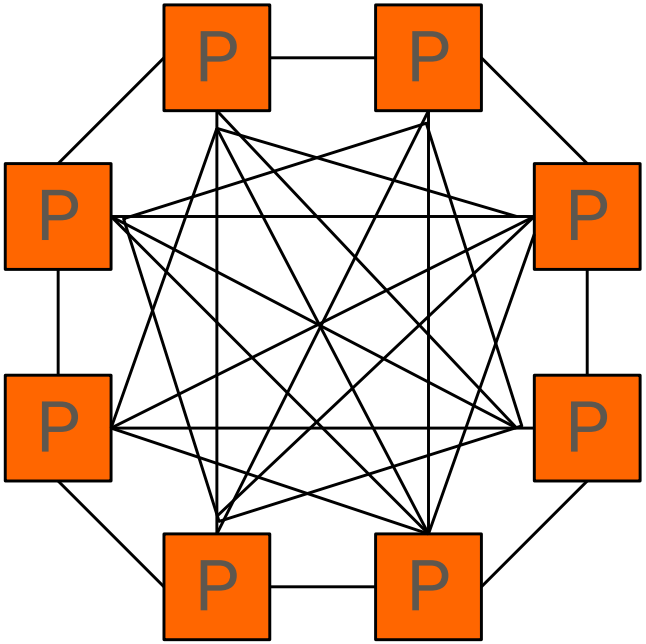
System Topologies

- Hypercube Topology
  - Is a multiple mesh topology.
  - Each processor connects to all other processors whose binary values differ by one bit. For example, processor 0(0000) connects to 1(0001) or 2(0010).



System Topologies

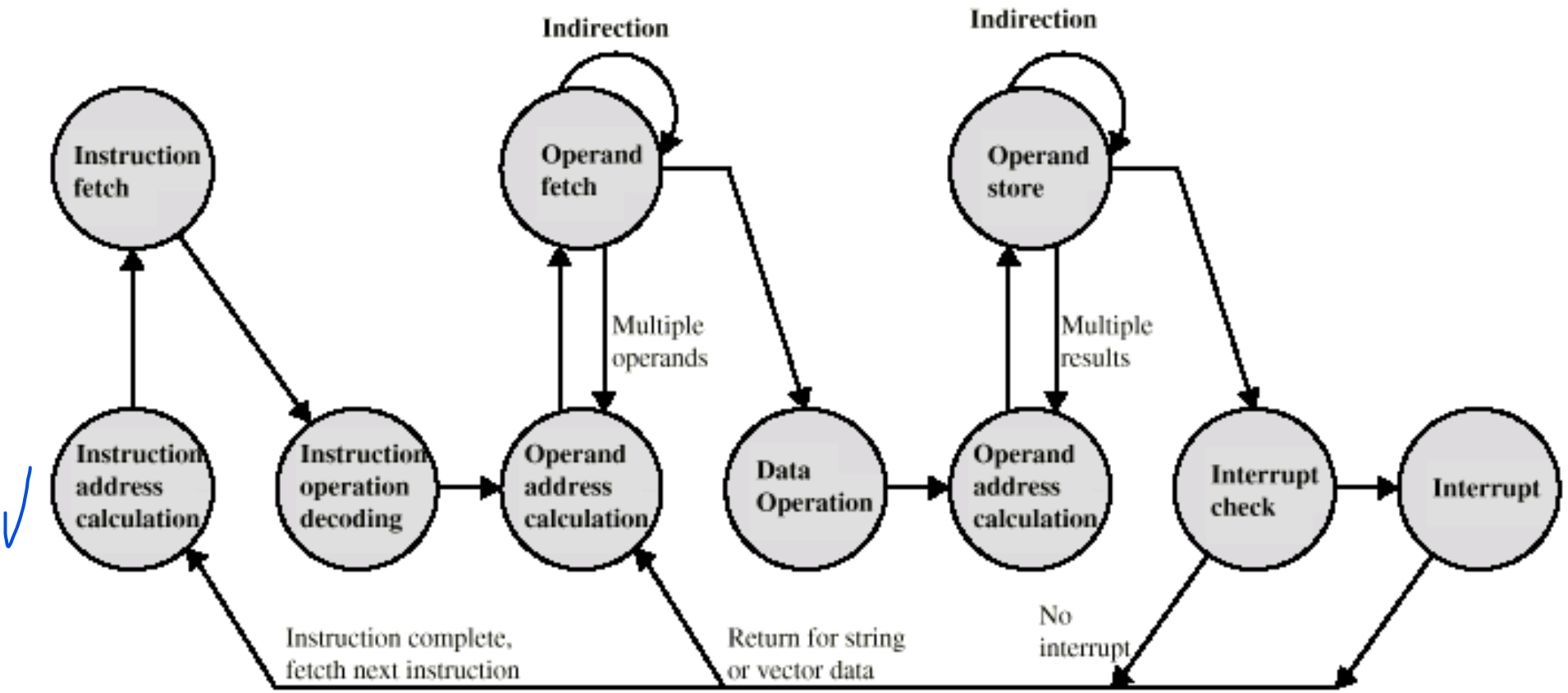
- Completely Connected Topology
  - Every processor has n-1 connections, one to each of the other processors.
  - There is an increase in complexity as the system grows but this offers maximum communication capabilities.



Instruction Pipelining

- Instruction pipelining is similar to the use of an assembly line in a manufacturing plant. An assembly line takes advantage of the fact that a product goes through various stages of production. By laying the production process out in an assembly line, products at various stages can be worked on simultaneously. This process is also referred to as pipelining,
- because, as in a pipeline, new inputs are accepted at one end before previously accepted inputs appear as output at the other end.

Pipelining- Instruction Cycle State Diagram



To apply this concept to instruction execution: in fact an instruction has a number of stages.

As above fig breaks inst cycle up into 10 tasks, which occur in sequence. Clearly, there should be some opportunity for pipelining.

Instruction Prefetch

- As a simple approach, consider subdividing inst processing into two stages: fetch instruction and execute instruction
- The *pipeline* has two independent stages. The first stage fetches an inst and buffers it. When the second stage is free, the first stage passes it the buffered instruction. While the second stage is executing the inst., the first stage takes advantage of any unused memory cycles to fetch and buffer the next inst.
- This is called instruction prefetch or fetch overlap.
- It should be clear that this process will speed up inst execution. If the fetch and execute stages were of equal duration, the inst cycle time would be halved.

Instruction Pipelining: Factors Affected

Execution rate is unlikely (varies) for two reasons:

- The execution time will generally be longer than the fetch time. Execution will involve reading and storing operands and the performance of some operation. Thus, the fetch stage may have to wait for some time before it can empty its buffer.
- A conditional branch inst makes the address of the next inst to be fetched unknown. Thus, the fetch stage must wait until it receives the next inst address from the execute stage. The execute stage may then have to wait while the next inst is fetched.

Instruction Pipelining Strategy

- *Guessing* can reduce the time loss from the second reason
- *A simple rule is the following:* When a conditional branch inst is passed on from the fetch to the execute stage, the fetch stage fetches the next inst in memory after the branch inst. Then, if the branch is not taken, no time is lost. If the branch is taken, the fetched inst must be discarded and a new inst fetched.

*instruction

Instruction Pipelining Strategy

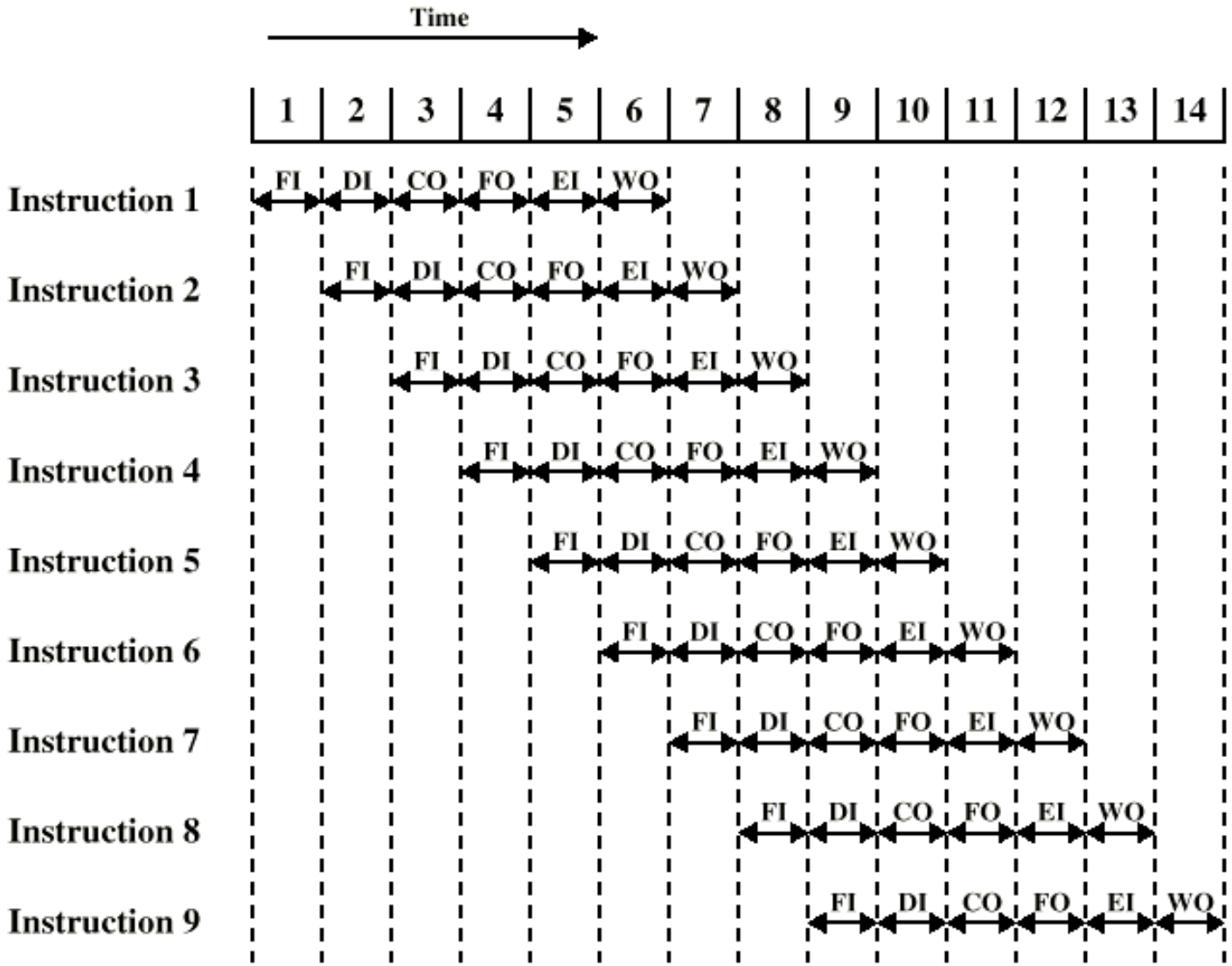
Following are the decomposition of the inst processing

- Fetch instruction (FI)
- Decode instruction (DI)
- Calculate operands (CO)
- Fetch operands (FO)
- Execute instructions (EI)
- Write operand or result (WO)

Ins 1 = 6 s
7th second = Ins 2
8th = Ins3

Timing of Pipeline (Example)

without parallelism: 9 * 6 = 54s
with parallelism: 14s



The diagram assumes that each inst goes through all six stages of the pipeline.

This will not always the case, e.g. a load inst don't need the WO stage. Several other factors serve to limit performance enhancement.

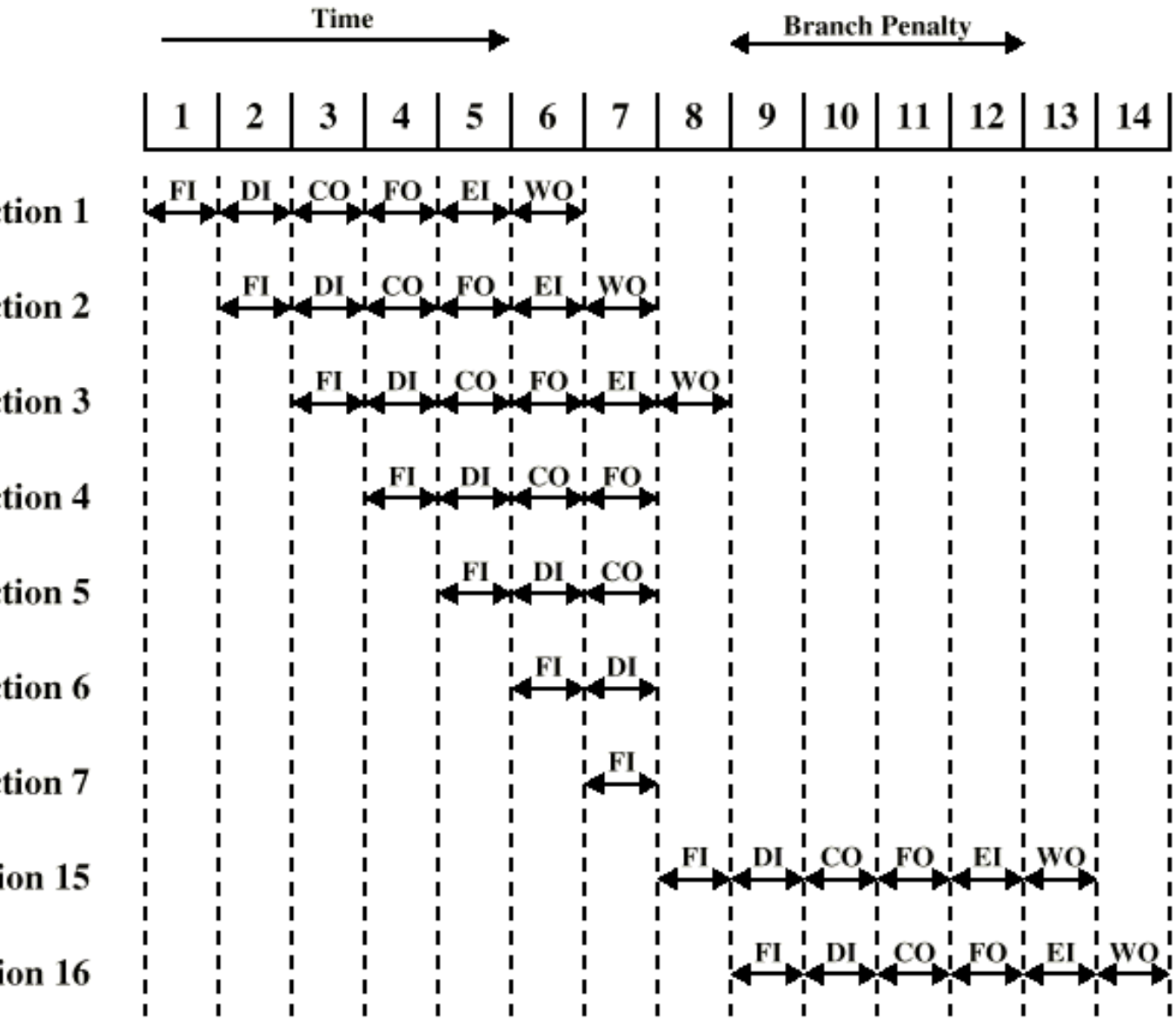
If the six stages are not of equal duration, there will be some waiting involved at various pipeline stages.

Another difficulty is conditional branch inst, which can invalidate several instruction fetches.

Figure shows that a six-stage pipeline can reduce the execution time for 9 instructions from 54 time units to 14 time units.

Branch in a Pipeline

first Ins = 6s
7th second mei 2nd Ins....just aik branch statement sai itna gap
13th second mei 3rd Ins



Assume that inst 3 is a conditional branch to inst 15.

Until the inst is executed, there is no way of knowing which inst will come next.

Pipeline, in this example simply loads next inst in sequence (inst 4) and proceeds.

In previous example branch is not taken and we get full performance. But here branch is taken.

The pipeline must be cleared of inst that are not useful. During time unit 8, inst 15 enters pipeline.

No inst complete during time unit 9 through 12;
(i.e. performance penalty)

Figure shows the effect of a Conditional Branch on inst Pipeline Operation

Six Stage Instruction Pipeline

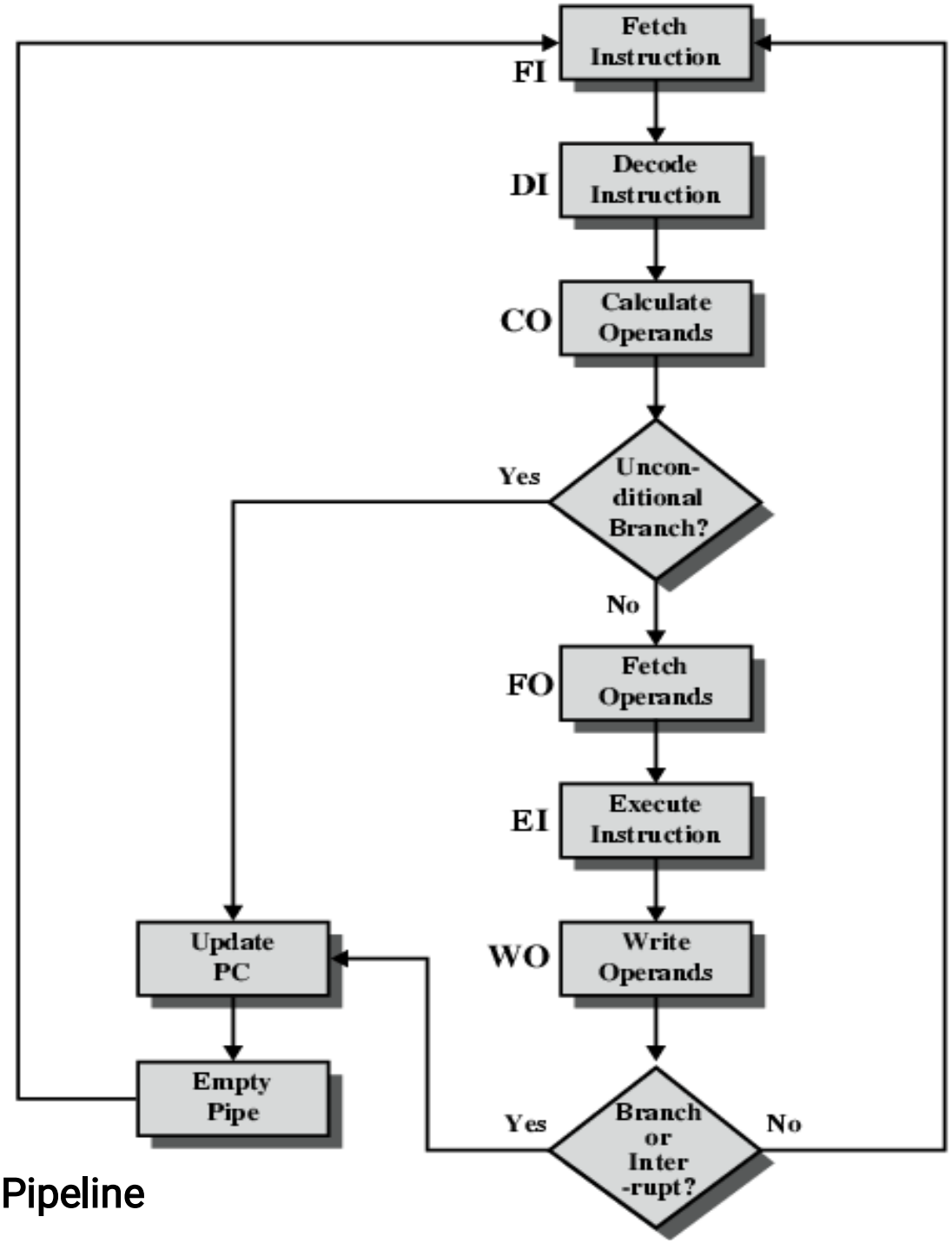


Figure shows the Six-Stage CPU Instruction Pipeline

Alternative Pipeline Depiction

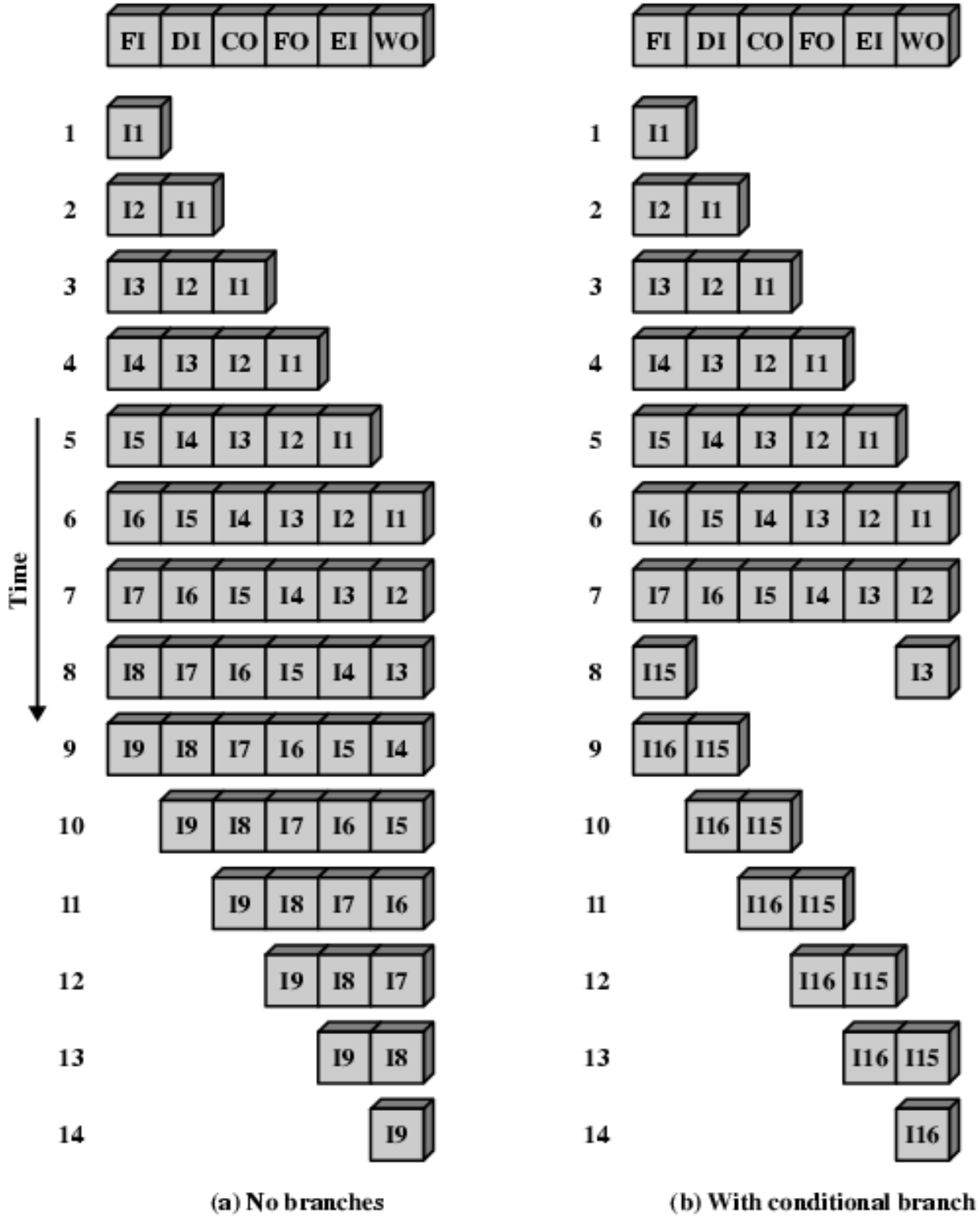
Fig (a) shows that pipeline is full at time 6, with 6 different inst in various stages of execution, and remains full through time 9. I-9 is the last inst to be executed.

Fig (b) shows that pipeline is full at times 6 and 7.

At time 7, I-3 is in execute stage and execute a branch to I-15. At this point inst I-4 through I-7 are f lushed from pipeline, so that at time 8, only two inst are in pipeline, I-3 and I-15.

It appears that greater number of stages in pipeline, faster execution rate.

Inst Pipelining is a power full technique for enhancing performance but requires careful design to achieve optimum results with reasonable complexity.



(a) No branches (b) With conditional branch

Pipelining: Improved Performance

Pipelining sai performance improve hoti hei but aisa nie hei keh performance double ho jai

- But not doubled:
  - Fetch usually shorter than execution
    - Prefetch more than one instruction?
  - Any jump or branch means that prefetched instructions are not the required instructions
- Add more stages to improve performance

generally execution cycle took more time than fetch cycle.

branch statement mean that we need to discard pre-fetch instructions.

more stages => less dependability => less wait => No deadlock => More efficiency

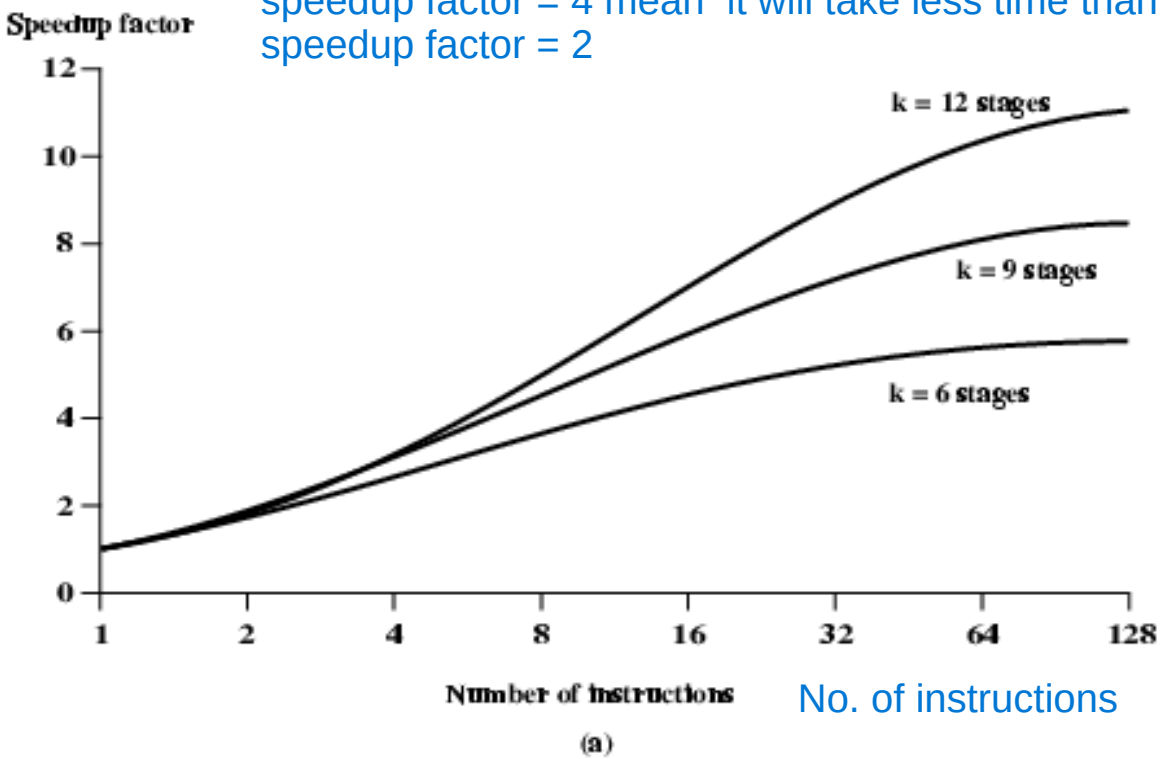
Speedup Factors with Instruction Pipelining

total instructions = 16
stages = 6
speed up factor = 4

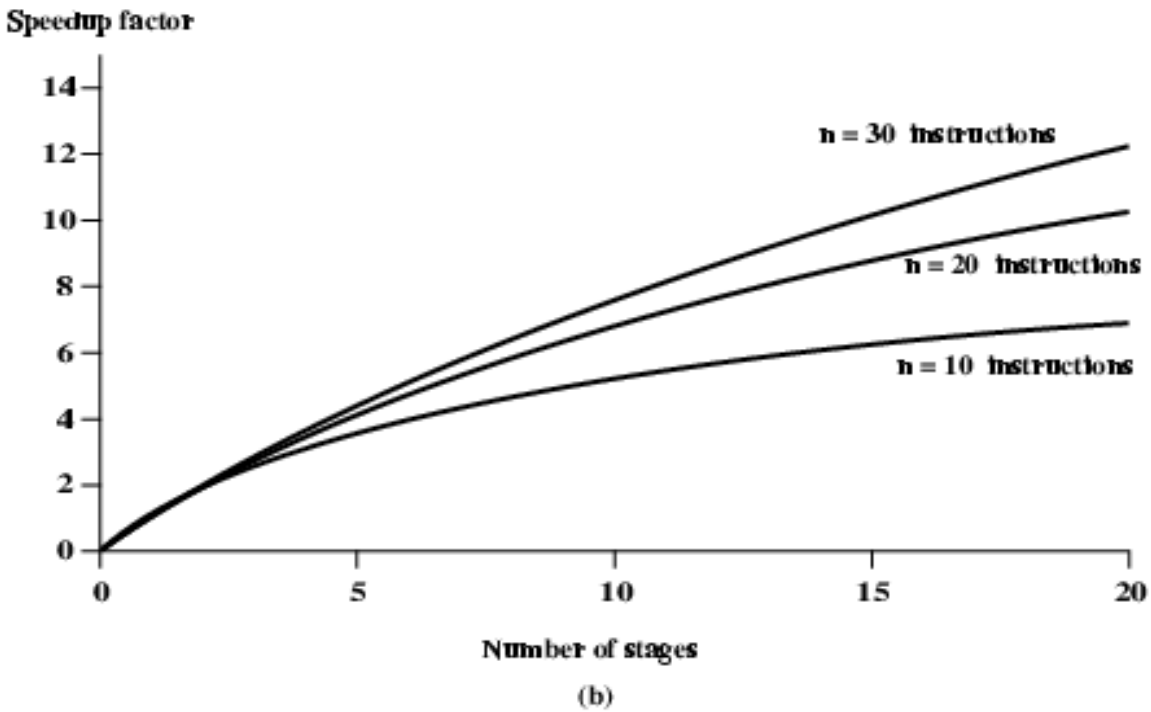
stages = 9
speed up factor = 5

More stages => More Speed Up factor
less time consumption

speedup factor = 4 mean it will take less time than speedup factor = 2



No. of instructions



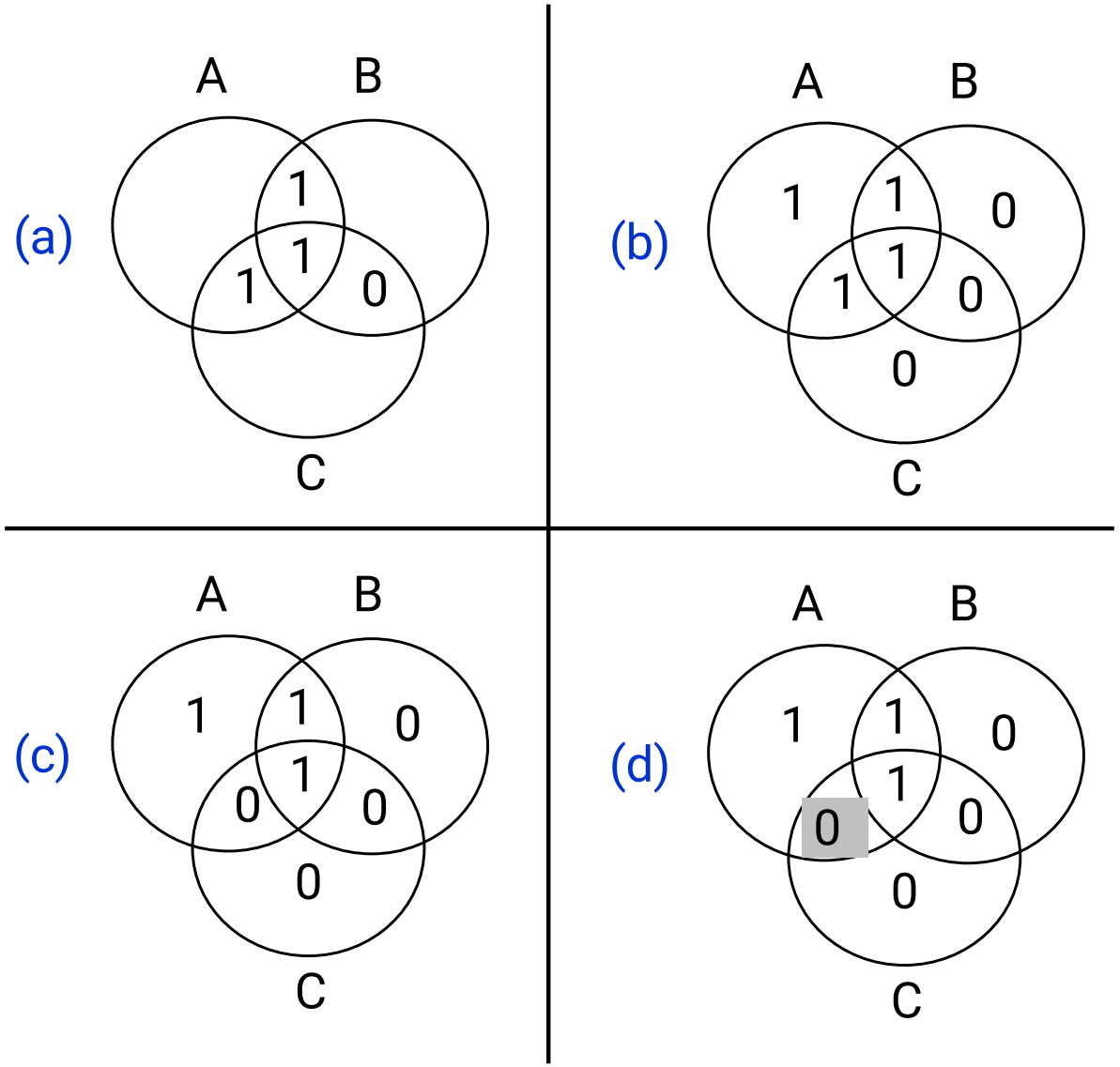
Error Correction

- Hard Failure
  - Permanent defect
- Soft Error
  - is a Random, non-destructive event alters contents of one or more memory cells without damaging memory , caused by power supply, or alpha particles in materials
- Detected using Hamming error correcting code

- When data are to be read into memory, a calculation is performed on data to produce a code. Both code and data are stored, thus, if an M -bit word of data is to be stored, and the code is of length K bits, then actual size of stored word is (M + K) bits

- When previously stored word is read out, code is used to detect error & then correct the errors

Error detection & correction using Hamming code



-Hamming Code on 4-bit words (M = 4), with three intersecting circle, & seven compartments, 4 data bits are placed at inner comp Fig(a), & other are f lled with parity bits

-Each parity bit is chosen so that total no. of 1s in its circle is even Fig(b), As circle A includes three data 1s, parity bit in that circle is set to 1

-If an error changes one of data bits Fig(c), it is easily found, By checking parity bits, discrepancies are found in circle A & C but not in B, i.e. only one comp is in A&C but not B, correcting error by changing that bit

e.g., Hamming Code for 8-bit words

- For Hamming Code how long the code must be. An error could occur on any of M data bits or K check bits, we must have $(2^k - 1) >= (M + K)$

- E.g., for a word of 8 data bits (M = 8), we have

- $K = 3$: $2^3 - 1 < 8+3$

- $K = 4$: $2^4 - 1 > 8+4$, (thus 8-data bits require 4-check bits, total of 12-bits word)

$$C1 = D1 * D2 * D4 * D5 * D7$$
$$C2 = D1 * D3 * D4 * D6 * D7$$
$$C4 = D2 * D3 * D4 * D8$$
$$C8 = D5 * D6 * D7 * D8$$

Check bits are calculated as

Bit position	12	11	10	9	8	7	6	5	4	3	2	1
Position no	1100	1011	1010	1001	1000	0111	0110	0101	0100	0011	0010	0001
Data bit	D8	D7	D6	D5		D4	D3	D2		D1		
Check bit					C8				C4		C2	C1

e.g. C1 = D1 (0011) * D2 (0101) etc select all where first binary digit is 1 out of 4-

e.g., Hamming Code for 8-bit words

C1 = D1 , D2 , D4 , D5 , D7 (3, 5, 7, 9, 11 bit positions check with parity bit C1)

C2 = D1 , D3 , D4 , D6 , D7 (3, 6, 7, 10, 11 check with parity bit C2)

C4 = D2 , D3 , D4 , D8 (5, 6, 7, 8 with C4)

C8 = D5 , D6 , D7 , D8 ; we have an 8-bit word is

D8	D7	D6	D5	D4	D3	D2	D1
0	0	1	1	1	0	0	1

Originally Sent

C1 = 1 * 0 * 1 * 1 * 0 = 1

C2 = 1 * 0 * 1 * 1 * 0 = 1

C4 = 0 * 0 * 1 * 0 = 1

C8 = 1 * 1 * 0 * 0 = 0

Received As

C1 = 1 * 0 * 1 * 1 * 0 = 1

C2 = 1 * 1 * 1 * 1 * 0 = 0

C4 = 0 * 1 * 1 * 0 = 0

C8 = 1 * 1 * 0 * 0 = 0

Suppose data bit 3 sustain an error, changed from 0 to 1, when check bits are recalculated,

When new check bits are compared with old check bits

Resultant word is formed:

i.e. result is 0 1 1 0 indicating that bit position 6, which contains data bit 3, is in error

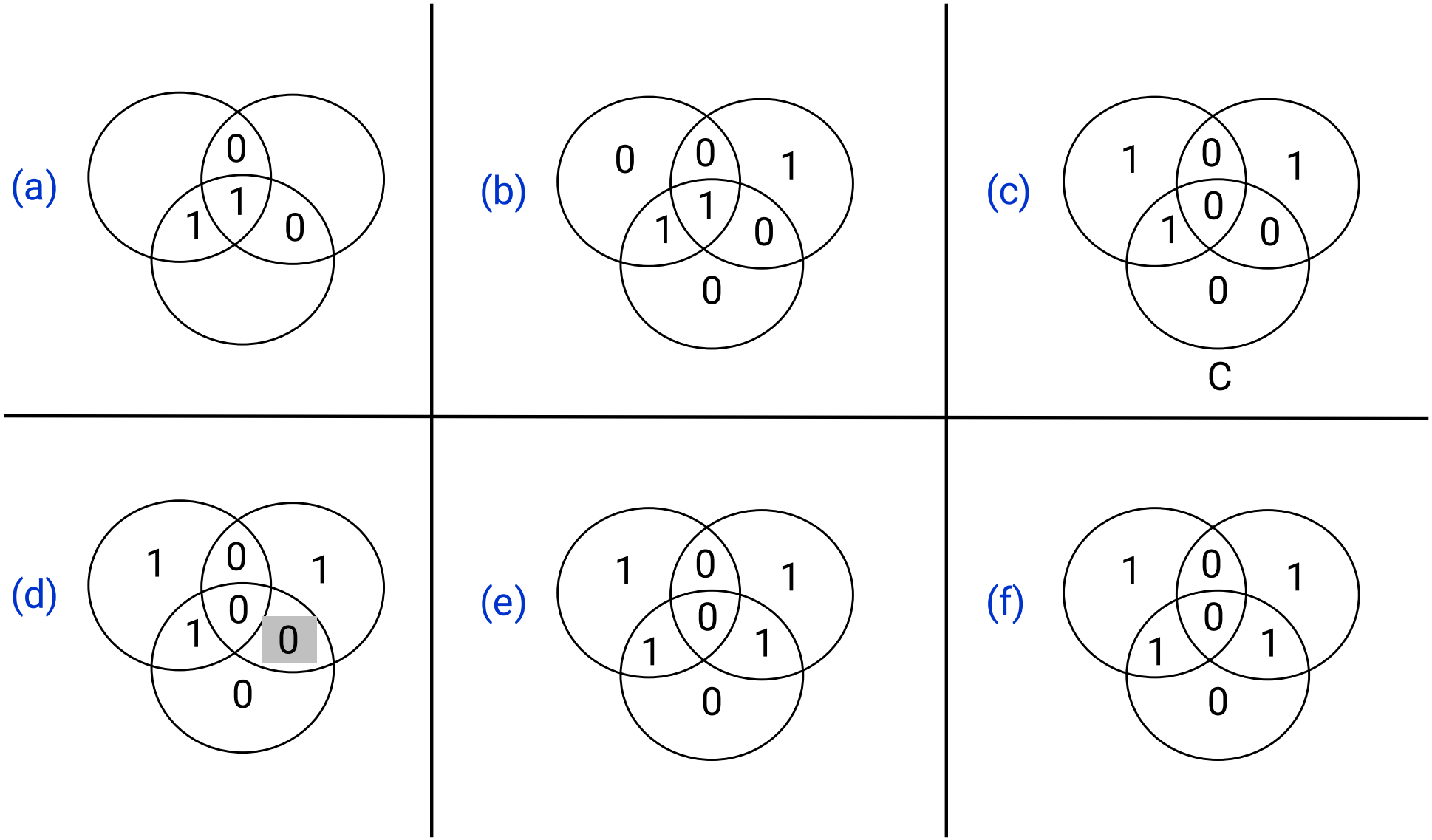
C8	C4	C2	C1	
0	1	1	1	
0	0	0	1	*
0	1	1	0	

e.g., Hamming Code for 8-bit words

Bit position	12	11	10	9	8	7	6	5	4	3	2	1
Position number	1100	1011	1010	1001	1000	0111	0110	0101	0100	0011	0010	0001
Data bit	D8	D7	D6	D5		D4	D3	D2		D1		
Check bit					C8				C4		C2	C1
Word stored as	0	0	1	1	0	1	0	0	1	1	1	1
Word fetched as	0	0	1	1	0	1	1	0	1	1	1	1
Position number	1100	1011	1010	1001	1000	0111	0110	0101	0100	0011	0010	0001
Check bit					0				0		0	1

Table : Single-error-correcting (SEC) & Check Bit Calculation

Hamming (DED) Code



Increase in Word length with Error Correction

	Single-Error Correction		Single-Error Correction/ Double-Error Detection	
Data Bits	Check Bits	% Increase (Overhead)	Check Bits	% Increase (Overhead)
8	4	50	5	62.5
16	5	31.25	6	37.5
32	6	18.75	7	21.875
64	7	10.94	8	12.5
128	8	6.25	9	7.03
256	9	3.52	10	3.91

Distribute Systems & Internet of Things

Def nition: An ***Internet of Things*** is a Network of physical objects that are connected to the Internet allowing them to send, receive and exchange data.

Def nition: The Internet of Things as simply an interaction between the physical and digital worlds. The digital world interacts with the physical world using a plethora of sensors and actuators. Internet of Things refers a new kind of world where almost all th devices and appliances that we use are connected to a network.

Internet of Things (Introduction)

Internet of Things

Internet of Things (Introduction)

Sensors and *actuators* are devices, which help in interacting with the physical environment. The data collected by the sensors has to be stored and processed intelligently in order to derive useful inferences/decisions from it.

There are various types of sensors for different purposes. The sensors have the capacity to take measurements such as temperature, air quality, speed, humidity, pressure, f bw, movement and electricity etc.

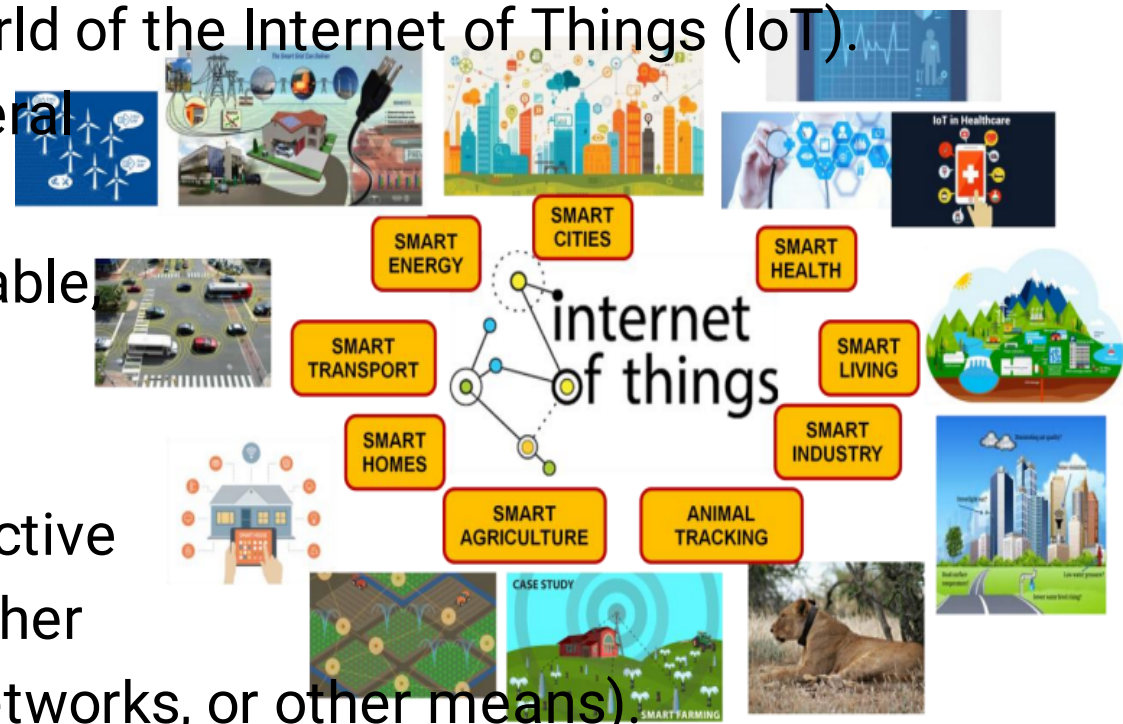
An actuator is a device that is used to effect a change in the environment such as the temperature controller of an air conditioner.

Sensors turn a physical input into an electrical output, and *actuators* do the opposite. They take electrical signals from control modules and turn them into physical outputs. ... The actuator receives the input of an electrical signal and turns it

Internet of Things (IoT) applications

Today's physical world contains about 1.5 trillion types of "things." When 99% of these objects are connected, a new era will be born. Imagine a world where billions of objects can sense, communicate and share information, all interconnected over public or private Internet Protocol (IP) networks. These interconnected objects have data regularly collected, analyzed and used to initiate action, providing a wealth of intelligence for planning, management and decision making. This is the world of the Internet of Things (IoT).

Internet of Things refers to the general idea of things, especially everyday objects, that are readable, recognizable, locatable, addressable through information sensing device and/or controllable via the Internet, irrespective of the communication means (whether via RFID, wireless LAN, wide area networks, or other means).



What is the Internet of Things?

The storage and processing of data can be done on the edge of the network itself or in a remote server.

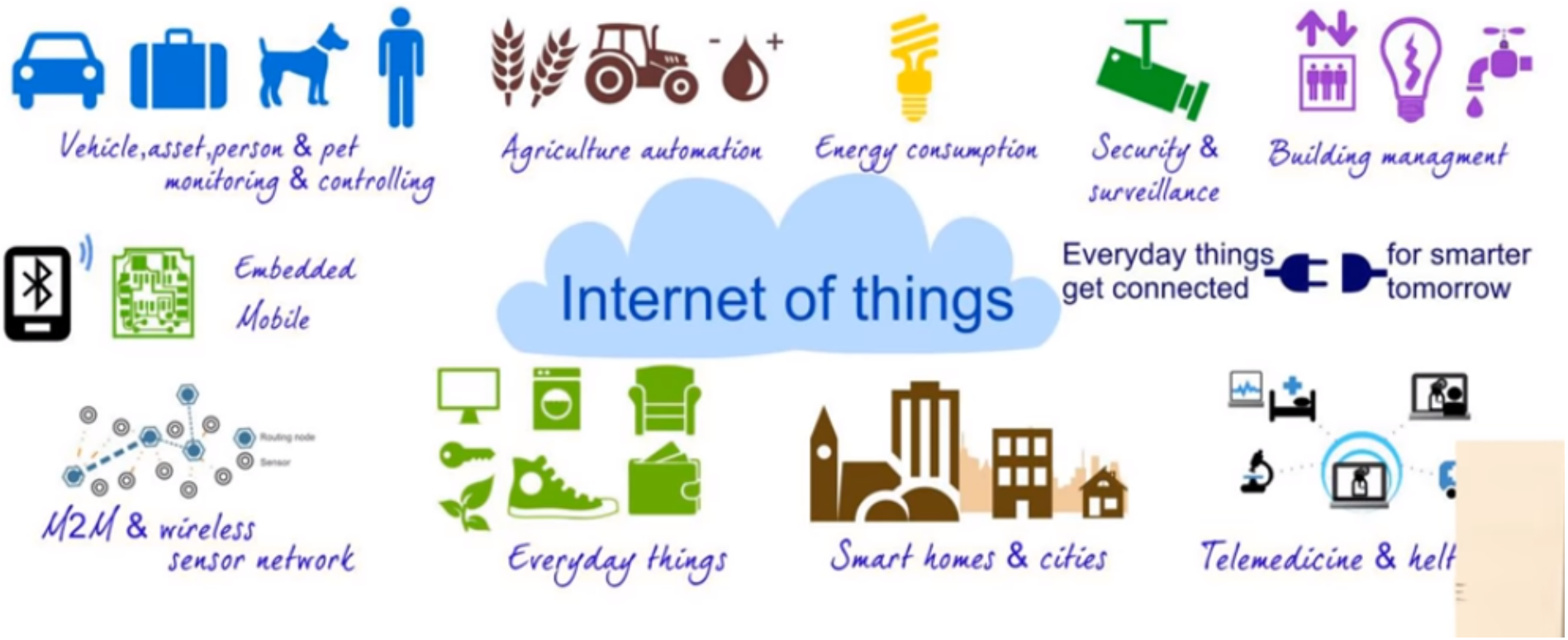
If any pre-processing of data is possible, then it is typically done at either the sensor or some other proximate device.

The processed data is then typically sent to a remote server.

The storage and processing capabilities of an IoT object are also restricted by the resources available, which are often very constrained due to limitations of size, energy, power, and computational capability. The communication between IoT devices is mainly wireless because they are generally installed at geographically dispersed locations.

What is the Internet of Things?

The Internet of Things finds various applications in health care, fitness, education, entertainment, social life, energy conservation, environment monitoring, home automation, and transport systems.



What is the Internet of Things?

Sensors: are small devices, which measures some kind of data/values, usually have analog input, meaning they measure temp, wind speed or something from nature & convert it into digital data, do no processing or very little processing, consume low-power, need to work a lot of time,

Local processing & Local Storage?

Second component/layer in the Internet of things is some local processing (by a mini computer or micro-controller) or local storage of data. Get data from sensors, process the data make decision locally and send some necessary/relevant data to Edge/Fog computing devices (edge of cloud),

Network and Internet?

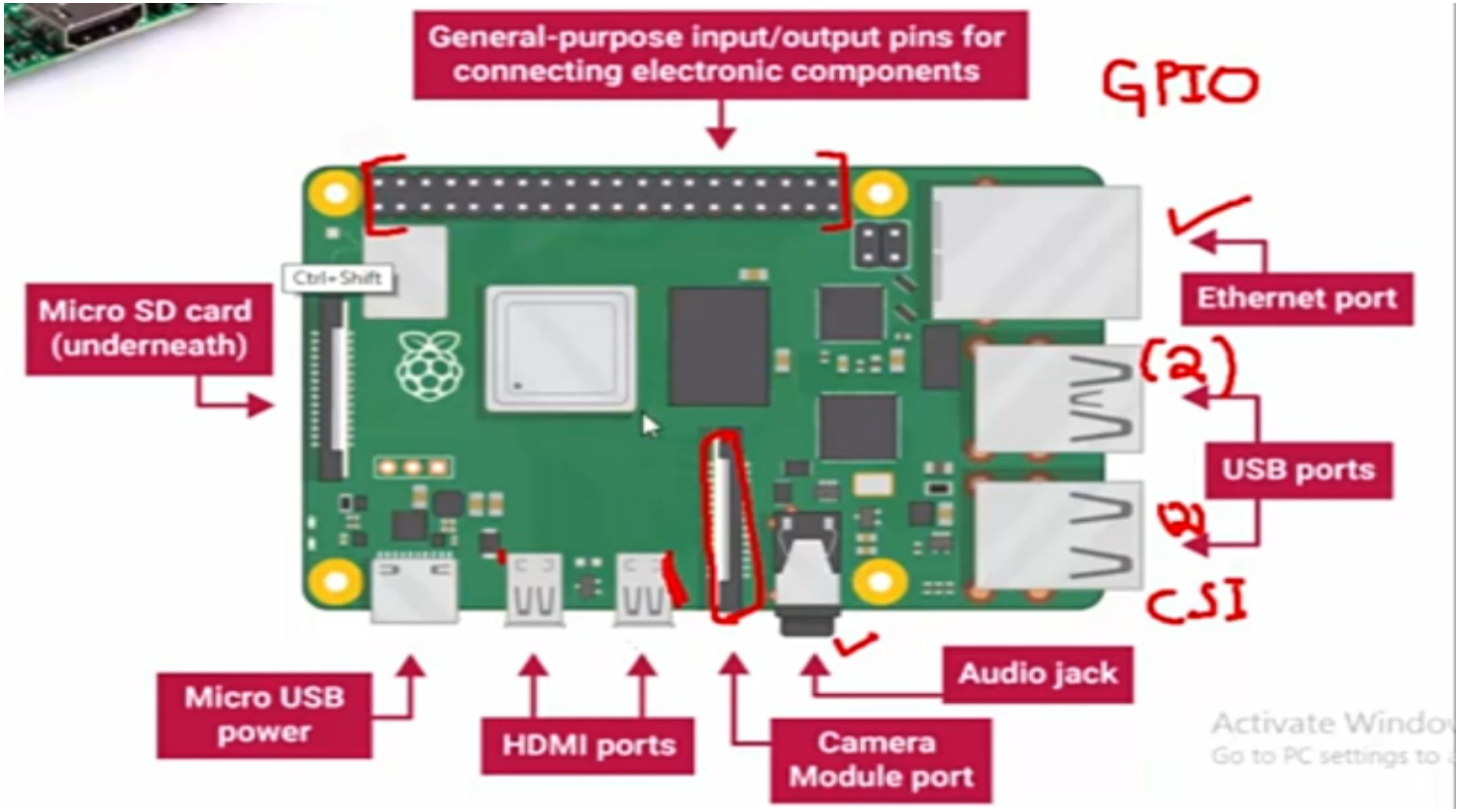
Edge/Fog computing devices provide the connectivity for Internet of things (first we discussed things & now internet), Gateways/ISP, it connects the local processing device to the cloud to send the most relevant data. For IoT, we have different protocols, CoAP & MQTT are designed for M2M communication (less secure) & others are HTTP, XMPP.

Cloud processing and storage

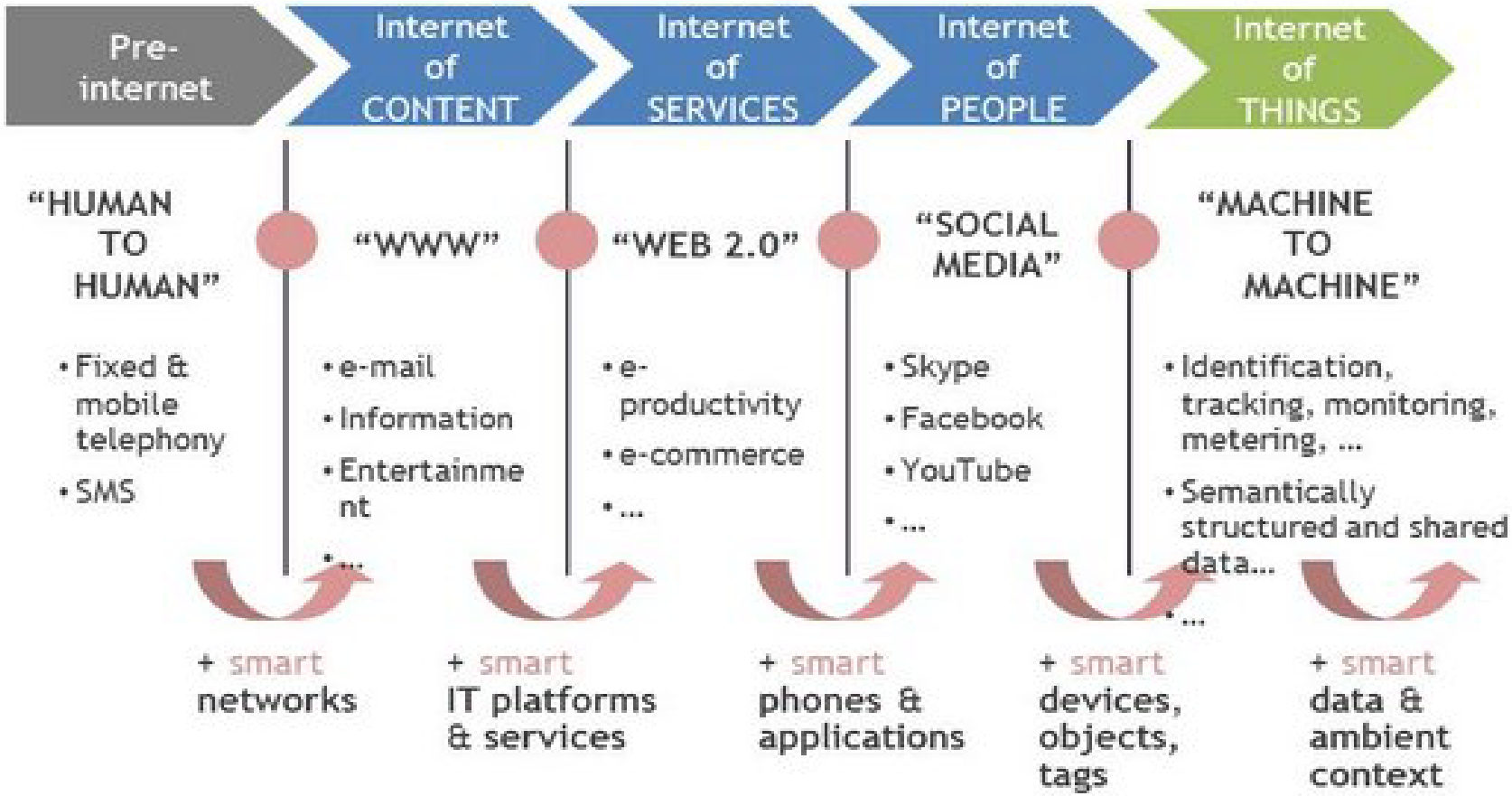
Cloud job is to aggregate this data received from gateways/fog computing devices, make inferences/conclusion and store data for the long term.

Raspberry Pi

A computer, runs linux, more software-oriented programming, full networking system. 900MHz ARM (Advanced RISC Machine), GPU, 1 GB RAM



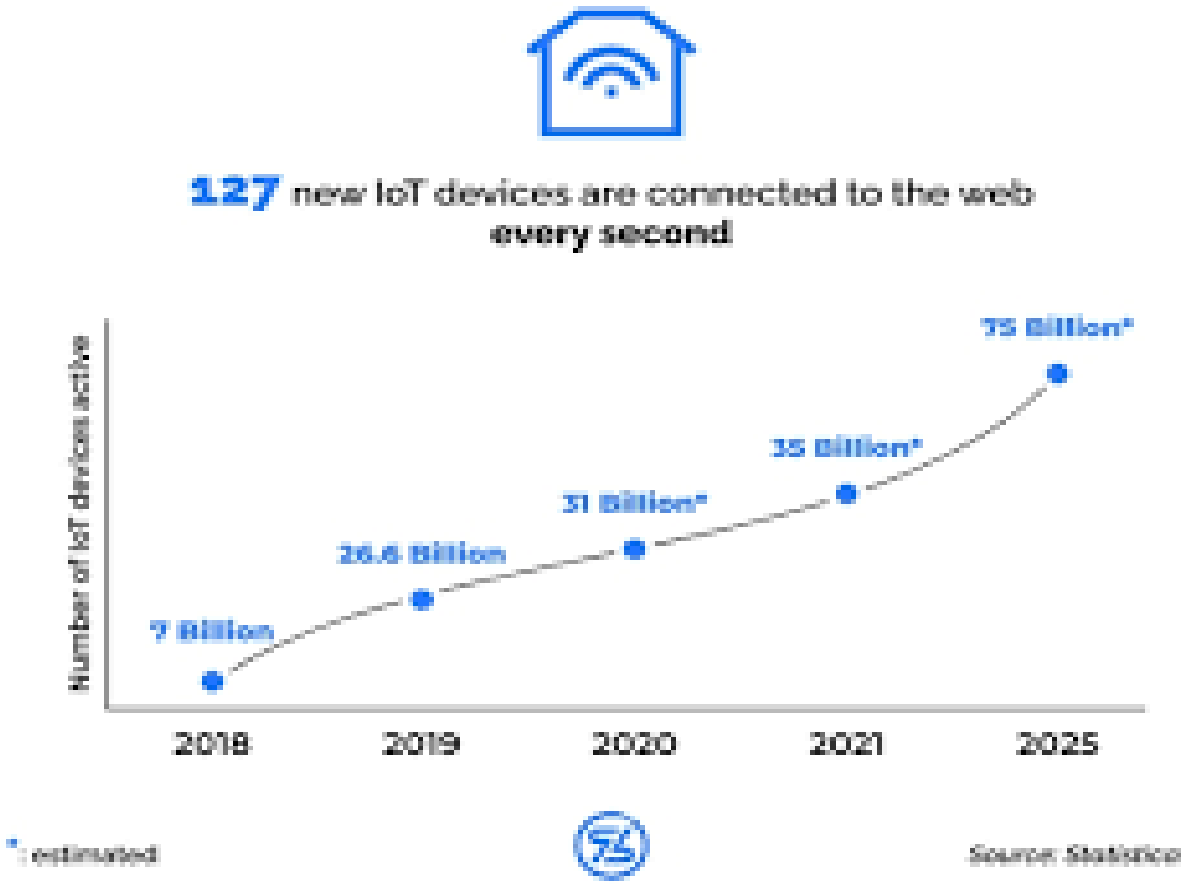
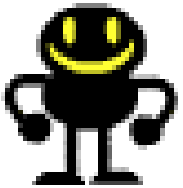
Evolution of Internet of Things



© ETSI 2014. All rights reserved

Evolution of Internet of Things

The Internet of Things (IoT) is everywhere. It provides advanced data collection, connectivity, and analysis of information collected by computers everywhere—taking the concepts of Machine-to-Machine communication farther than ever be



THE END