High Performance Computing

LECTURE #3

Agenda

- Parallel Computing Platform (Logical Organization)
- Flynn Taxonomy

1- Introduction

- A **computing platform** includes a hardware architecture and a software framework (including application frameworks), where the combination allows software to run.

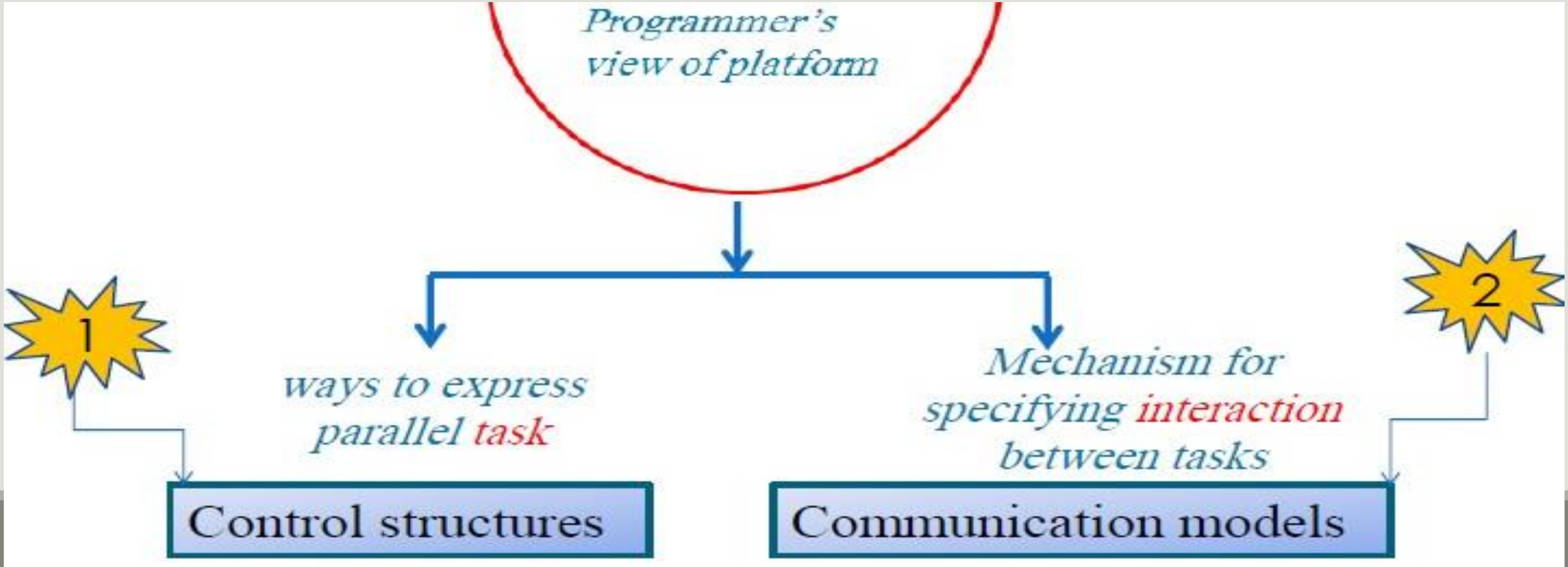
- Typical platforms include a
  - computer architecture,
  - operating system,
  - programming languages and
  - related user interface (run-time system libraries or graphical user interface).

Parallel Computing Platforms

Logical organization is split into **exactly two non-overlapping parts**

Control structures Communication models

An explicitly parallel program must specify accurately the interaction between concurrent subtasks..



- Parallelism can be expressed at various levels of granularity.
- **Computation / Communication Ratio:**
  - In parallel computing, granularity is a qualitative measure of the ratio of computation to communication.
  - Periods of computation are typically separated from periods of communication by synchronization events.

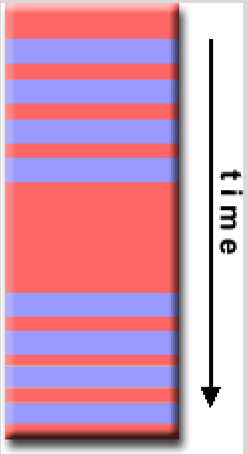
Fine-grain Parallelism:

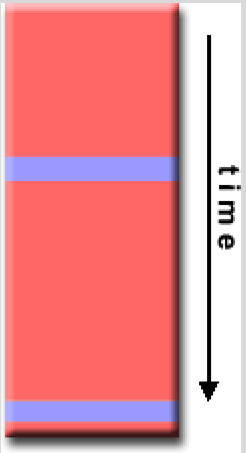
- Relatively small amounts of computational work are done between communication events
- Low computation to communication ratio.
- Facilitates **load balancing**.
- Implies high **communication overhead:**

If granularity is too fine it is possible that the overhead required for communications and synchronization between tasks takes longer than the computation.

Coarse-grain Parallelism:

- Relatively large amounts of computational work are done between communication/synchronization events
- High computation to communication ratio
- **Harder** to **load balance** efficiently.

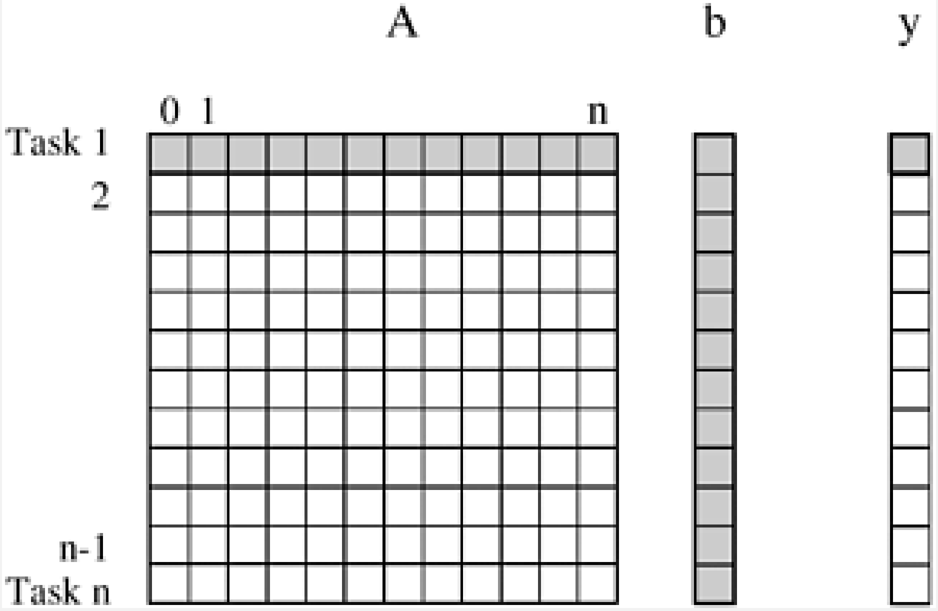


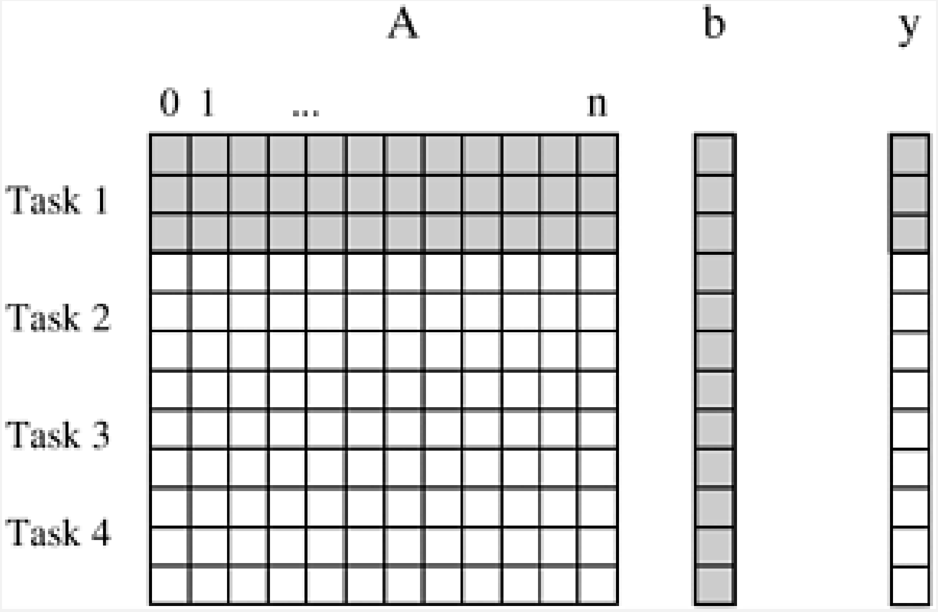


Fine-grain Parallelism:

Coarse-grain Parallelism:

- Example: Dense matrix multiplication





Which is Best?

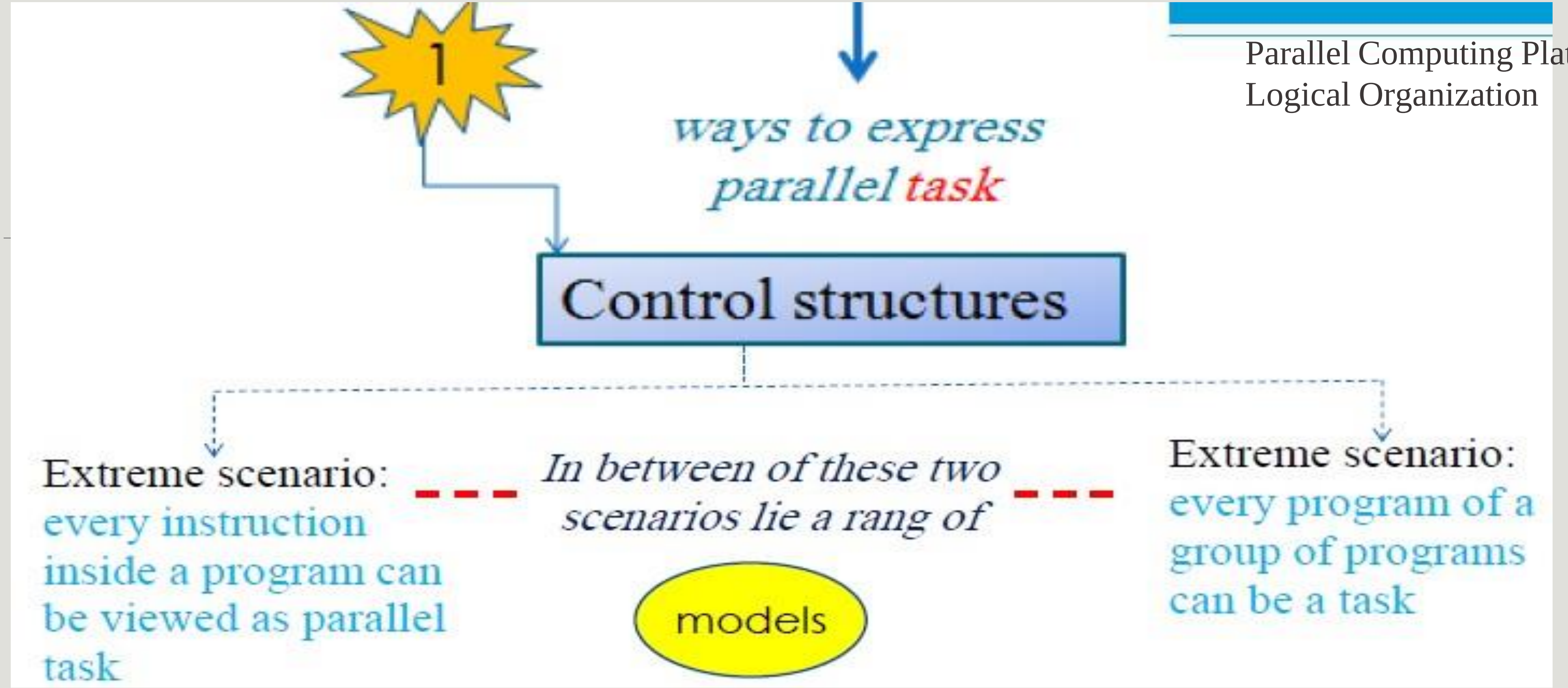
- The most efficient granularity is **dependent** on the algorithm and the hardware environment in which it runs.

- Use fine-grained granularity when the overhead of creating and managing tasks or threads is relatively low compared to the computational workload of each task.
- Fine-grained tasks are suitable when you need to maximize parallelism, and the tasks have limited data dependencies.
- Fine-grained granularity is appropriate when you want to keep all available CPU cores highly utilized.

Which is Best?

- The most efficient granularity is **dependent** on the algorithm and the hardware environment in which it runs.

  - Use coarse-grained granularity when the overhead of managing a large number of fine-grained tasks becomes a significant performance bottleneck.
  - Coarse-grained tasks are suitable when there are significant data dependencies or when managing fine-grained tasks becomes too complex.
  - Coarse-grained granularity is appropriate when the computational workload of each task is substantial, and you want to reduce the overhead of task creation and management.



Task A logically discrete section of computational work. A task is typically a program or program-like set of instructions that is executed by a processor.

Models:
Flynn's Classical Taxonomy

- Flynn’s classification scheme is based on the notion of a stream of information.

- Two types of information flow into a processor: instructions and data

- Each of these dimensions can have only one of two possible states: Single or Multiple.

- One of the more widely used classifications that classify parallel computers, use since 1966.

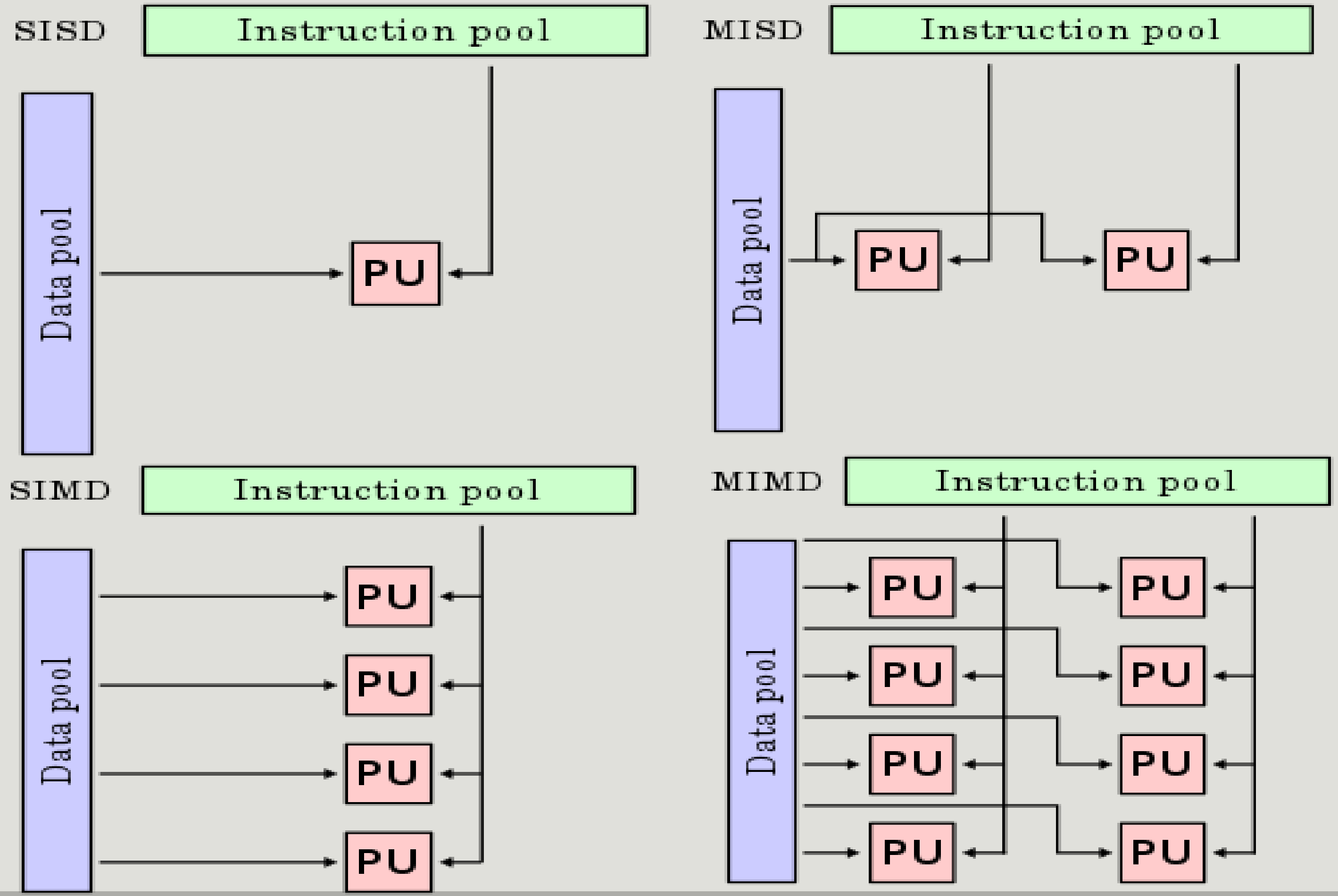
Flynn's Classical Taxonomy

The matrix below defines the 4 possible classifications according to Flynn:

models

SISD **Single Instruction, Single Data**	**SIMD** **Single Instruction, Multiple Data**
MISD Multiple Instruction, Single Data	MIMD Multiple Instruction, Multiple Data

Models:
Flynn's Classical Taxonomy



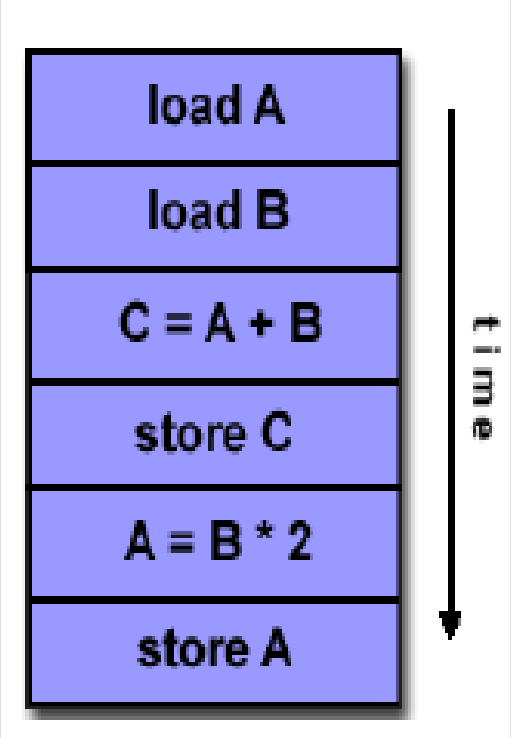
Flynn's Classical Taxonomy

The matrix below defines the 4 possible classifications according to Flynn:

~~SISD Single Instruction, Single Data~~	SIMD Single Instruction, Multiple Data
MISD Multiple Instruction, Single Data	MIMD Multiple Instruction, Multiple Data

Single Instruction, Single Data (SISD):

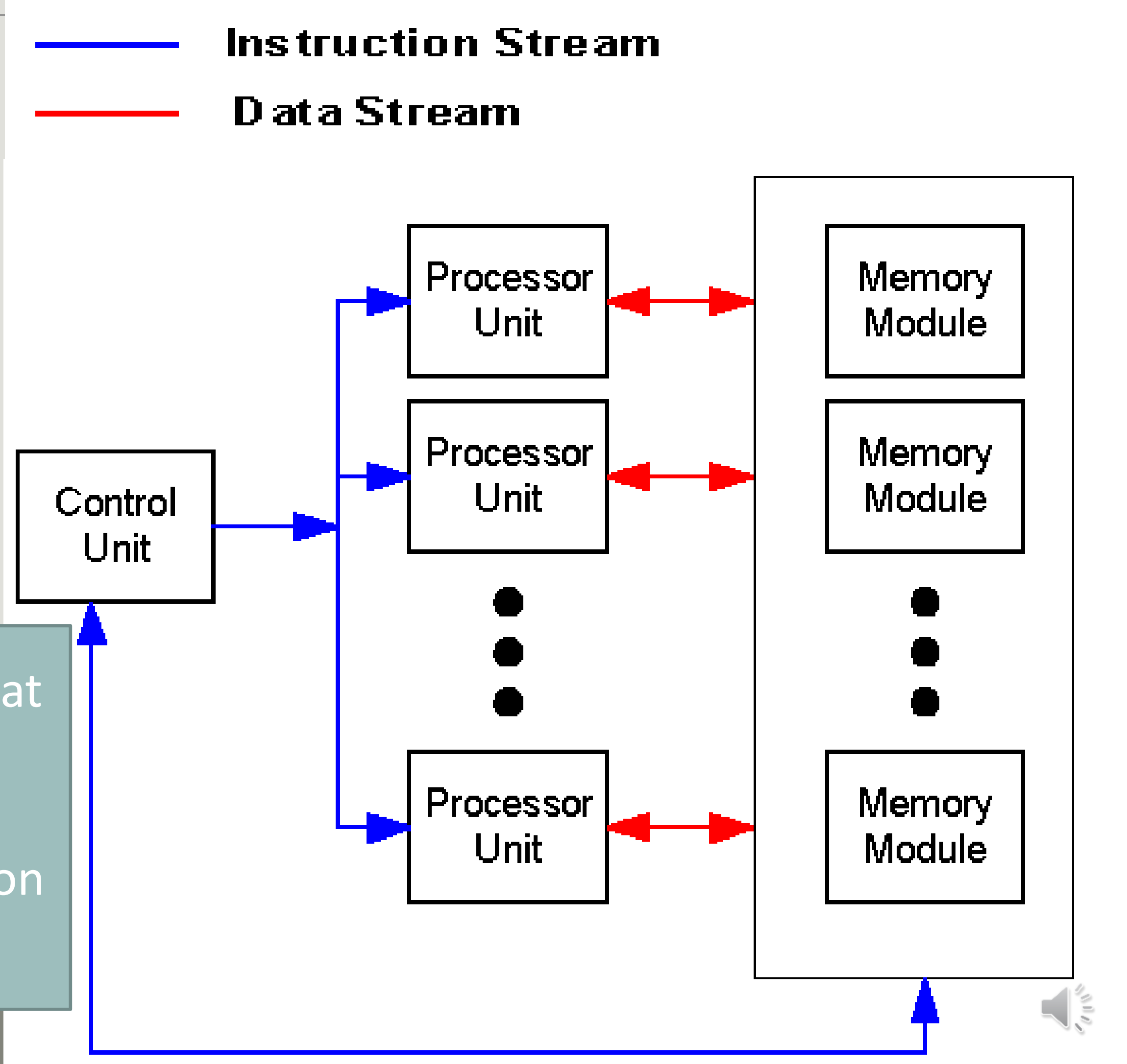
- A serial (non-parallel) computer
- Single instruction: only one instruction stream is being acted on by the CPU during any one clock cycle
- Single data: only one data stream is being used as input during any one clock cycle
- This is the oldest and even today, the most common type of computer
- Examples: older generation mainframes, minicomputers and workstations; most modern day PCs.

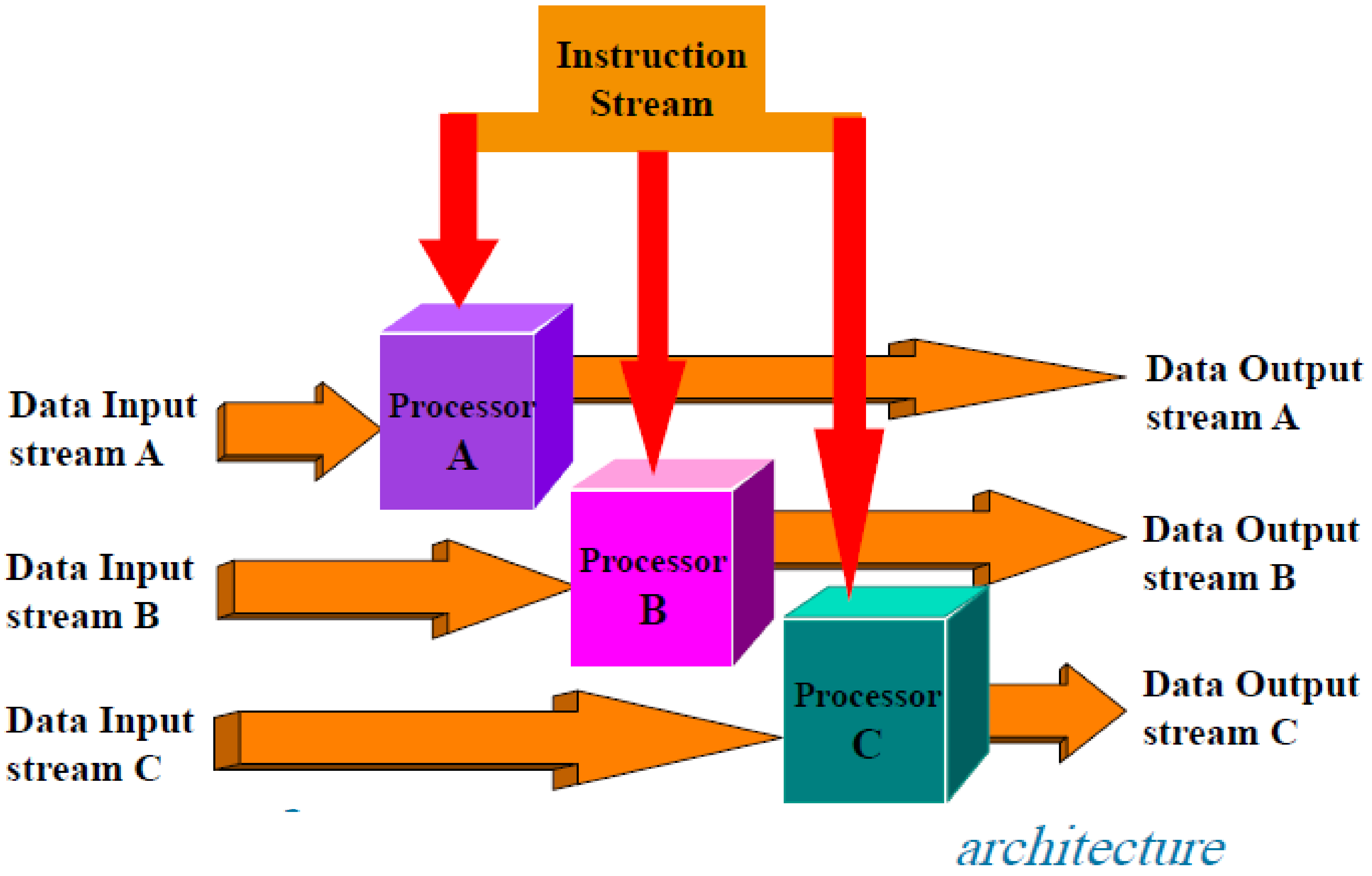


Single Instruction, Multiple Data (SIMD):

- 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

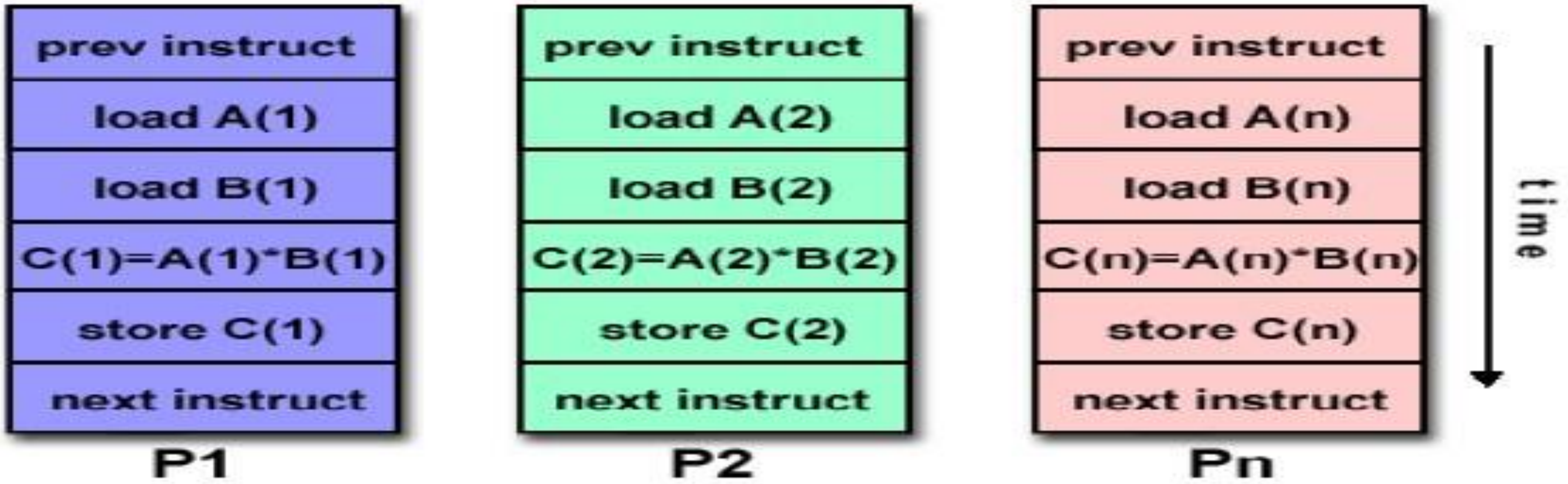
A single control unit that dispatches the same instruction to various processors (that work on different data)

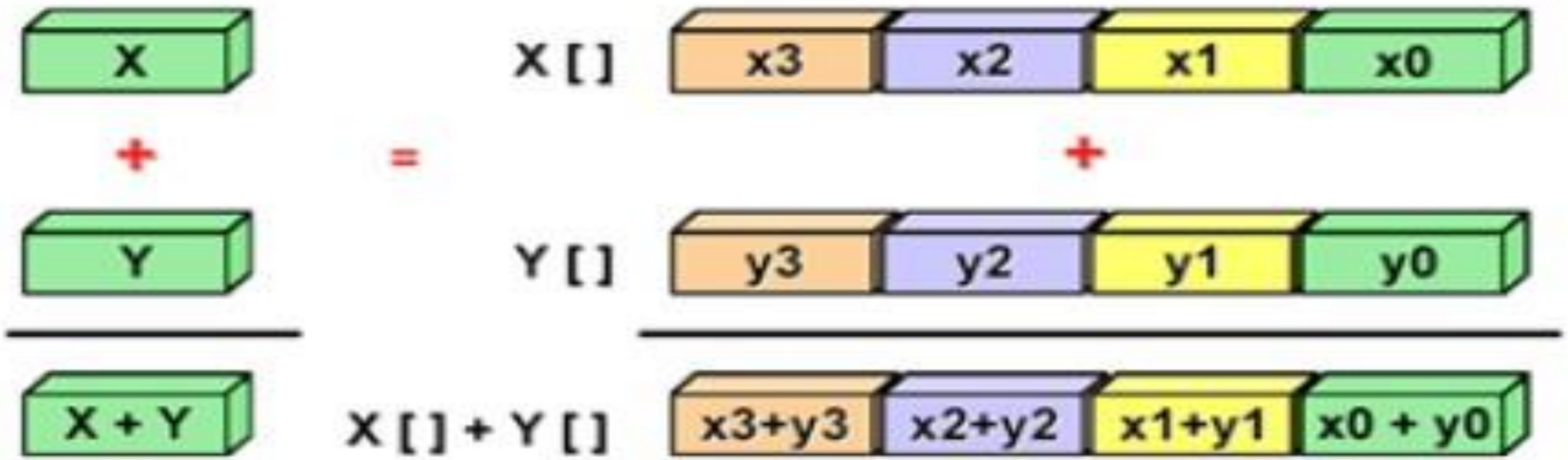



Instruction Stream
Data Input stream A
Processor A
Data Output stream A
Data Input stream B
Processor B
Data Output stream B
Data Input stream C
Processor C
Data Output stream C
architecture

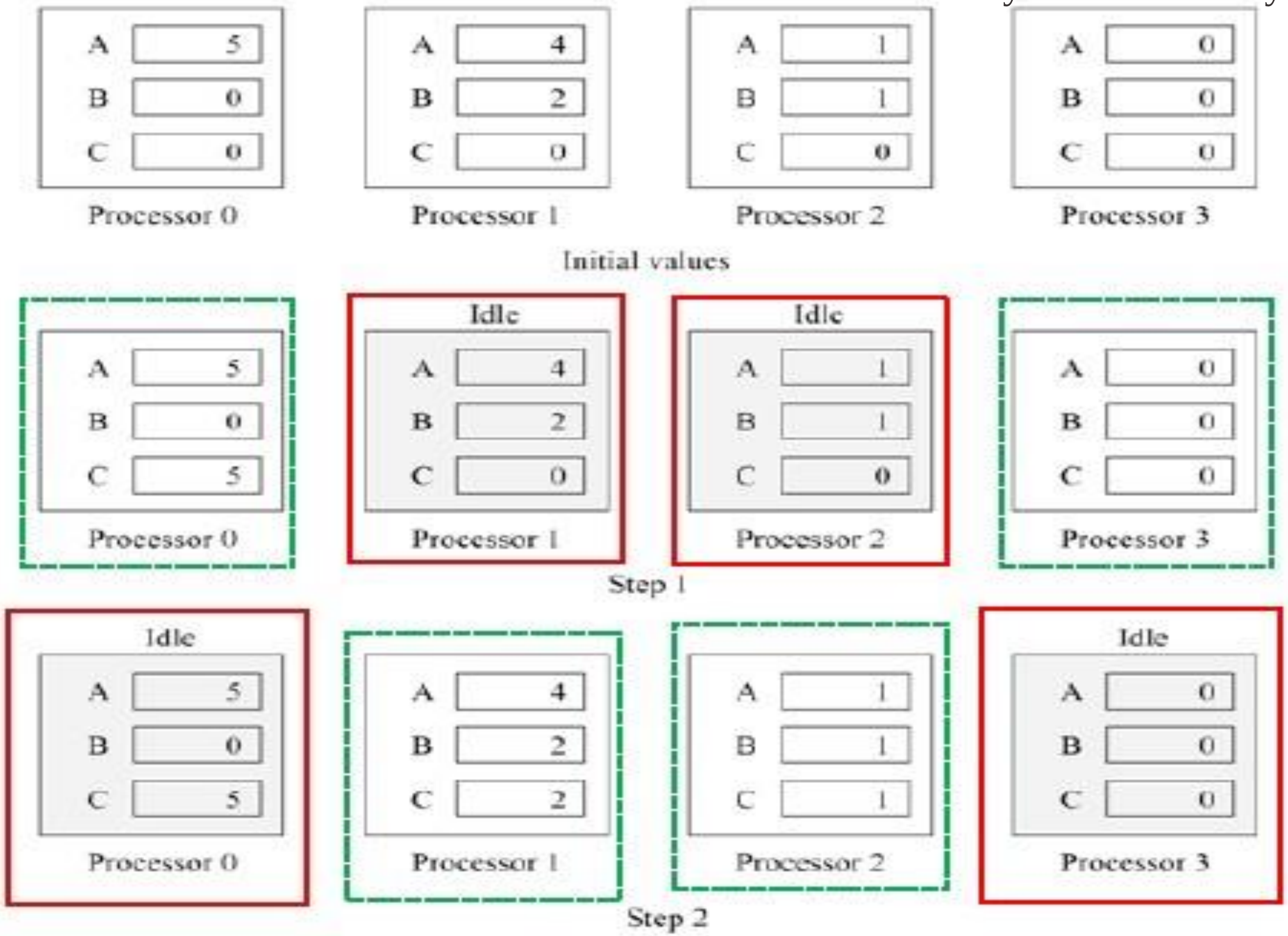
- Processor Arrays: ILLIAC IV, DAP Connection Machine CM-2, MasPar MP-1.
- Vector Pipelines: IBM 9000, Cray X-MP, Y-MP & C90, Fujitsu VP, NEC SX-2, Hitachi S820, ETA10
- Most modern computers, particularly those with graphics processor units (GPUs) employ SIMD instructions and execution units.
- Examples:

```
For (I = 0; i<1000; i++)
c[i] = a[i] * b[i];
```


prev instruct
load A(1)
load B(1)
C(1)=A(1)*B(1)
store C(1)
next instruct
P1
prev instruct
load A(2)
load B(2)
C(2)=A(2)*B(2)
store C(2)
next instruct
P2
prev instruct
load A(n)
load B(n)
C(n)=A(n)*B(n)
store C(n)
next instruct
Pn
time


X
+
Y
X + Y
=
X [ ]
x3
x2
x1
x0
+
Y [ ]
y3
y2
y1
y0
X [ ] + Y [ ]
x3+y3
x2+y2
x1+y1
x0 + y0

```
if (B == 0)
        C = A;
else
        C = A/B;
```



One problem of SIMD architectures

Executing a conditional statement on an SIMD computer with four processors: (a) the conditional statement; (b) the execution of the statement in two steps.

Single Instruction, Multiple Data (SIMD):

- Conditional Operations in SIMD: Divergence leading to idle states
- Possible Solutions:

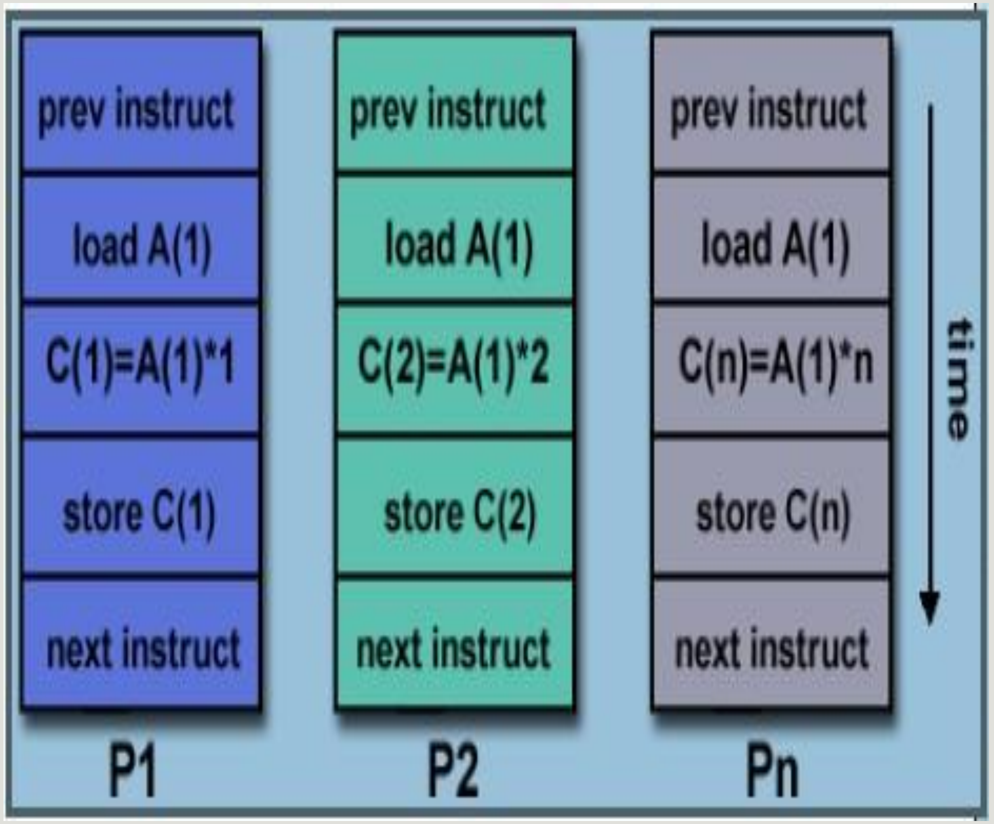
  - SIMD friendly algorithm
  - Prediction Hardware
  - Branchless code (Ternary operator)
  - Masked Operations (Intel AVX-512)
  - SIMD Compiler Directives

Your Turn !!!

Guess what are the SIMD drawbacks??!!

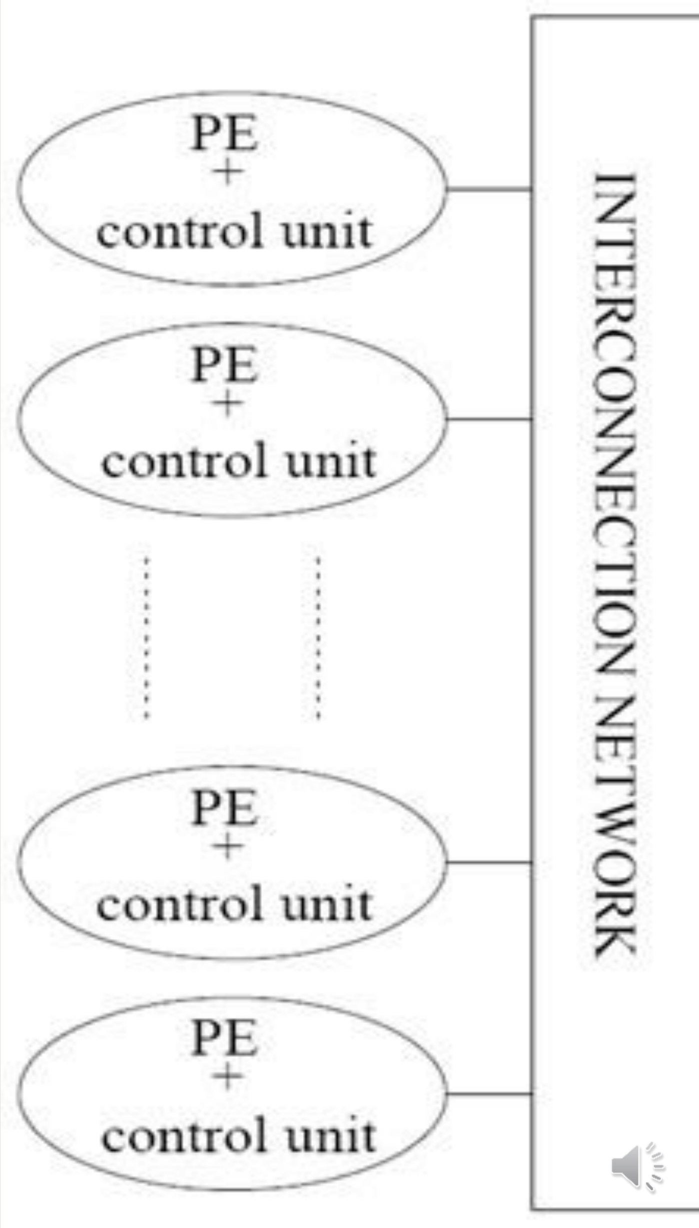
Multiple Instruction, Single Data (MISD):

- A single data stream is fed into multiple processing units.
- Each processing unit operates on the data independently via independent instruction streams.
- Few actual examples of this class of parallel computer have ever existed. One is the experimental Carnegie-Mellon C.mmp computer (1971).
- ex: Multiple cryptography algorithms attempting to crack a single coded message, fault tolerance.

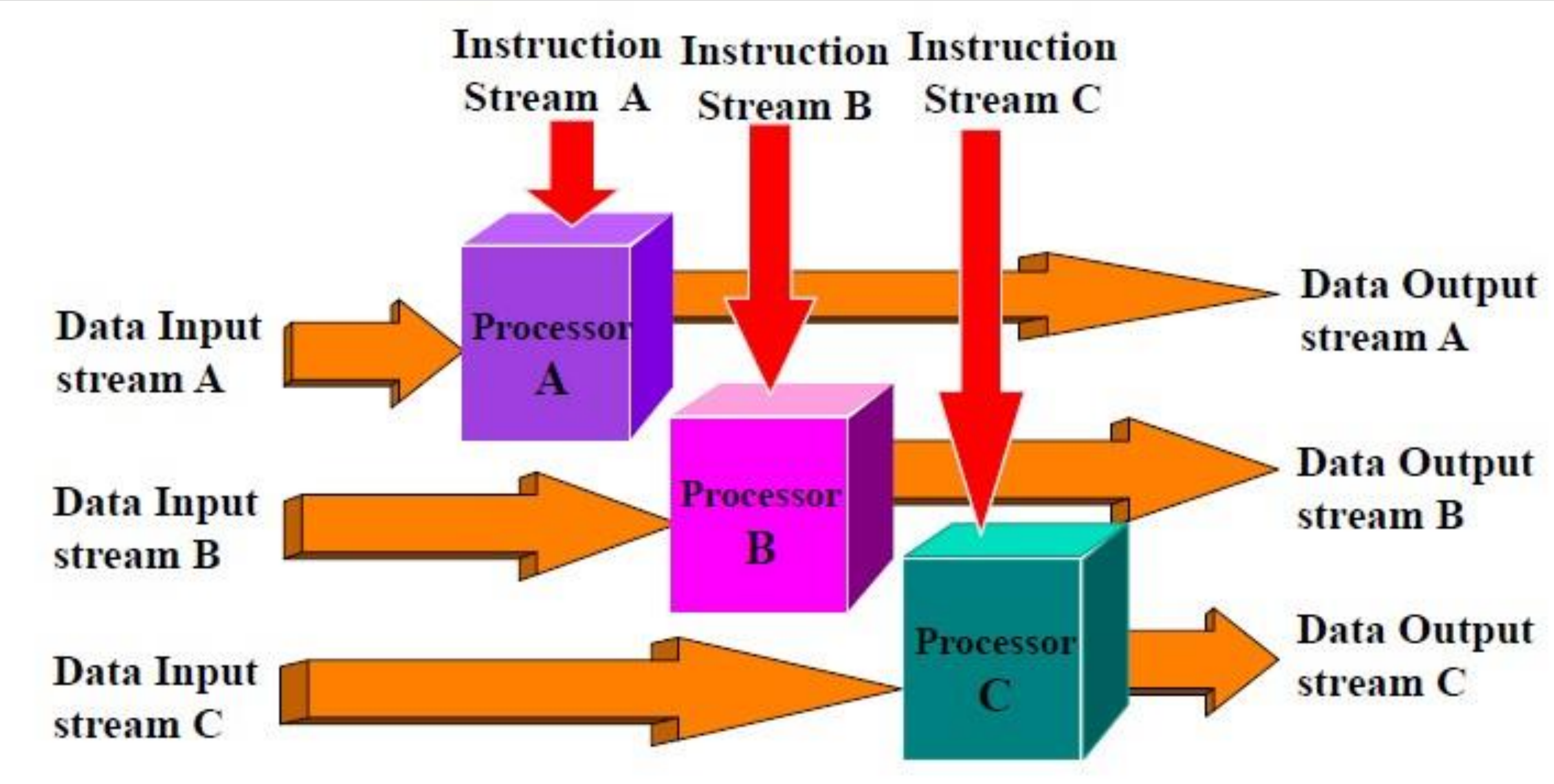


Multiple Instruction, Multiple Data (MIMD):

- Currently, the most common type of parallel computer. Most modern computers fall into this category.
- Multiple Instruction: every processor may be executing a different instruction stream.
- Multiple Data: every processor may be working with a different data stream.
- Examples: most current supercomputers, networked parallel computer clusters and "grids", multi-processor SMP computers, multi-core PCs.
- Note: many MIMD architectures also include SIMD execution sub-components



Multiple Instruction, Multiple Data (MIMD):



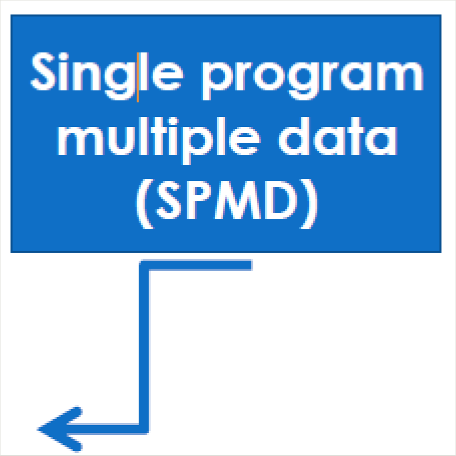
Multiple Instruction, Multiple Data (MIMD):

A simple variant of this model is

- Relies on multiple instance of the same program executing on different data

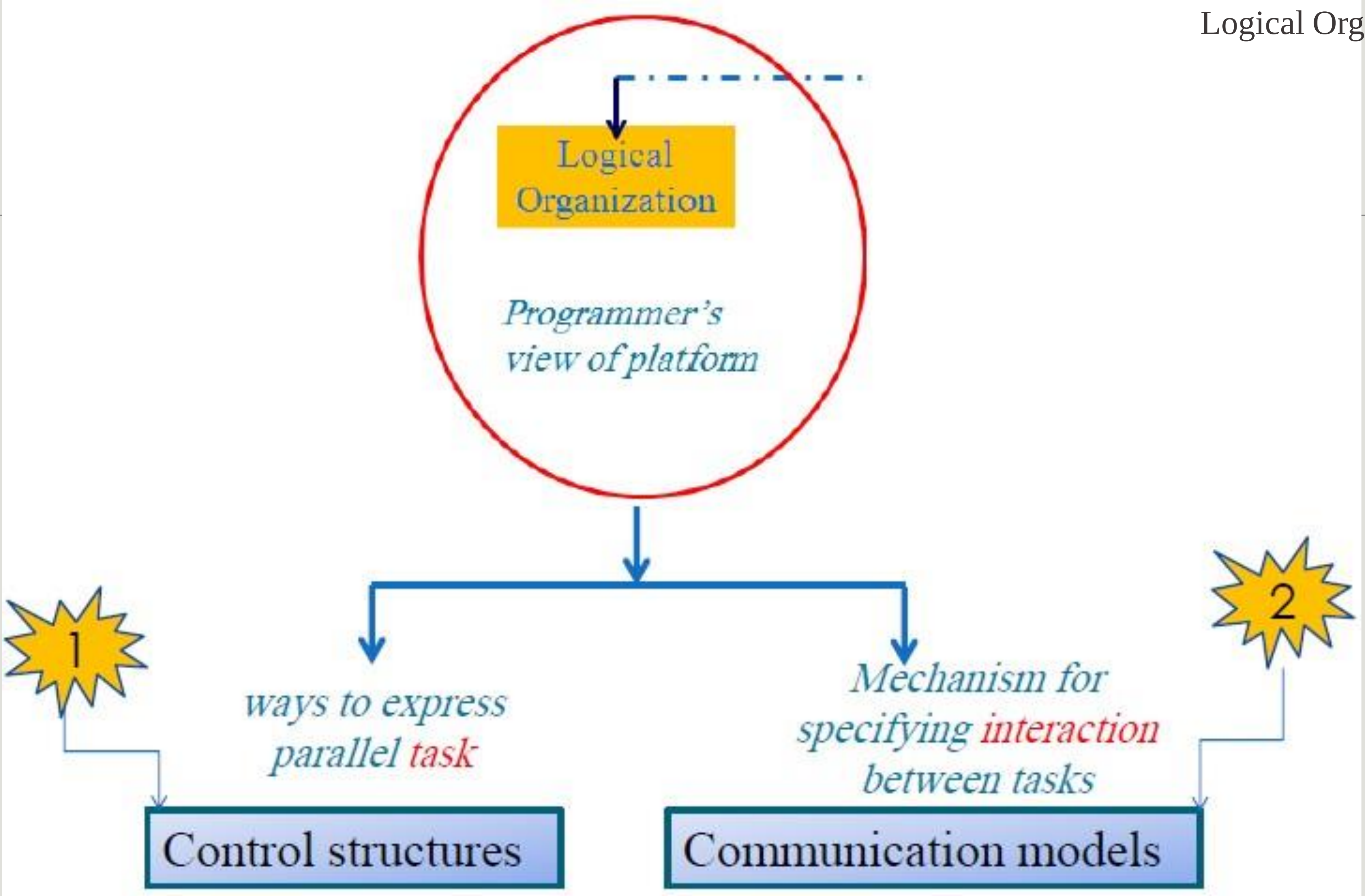
- Widely used by many parallel platforms and requires minimal architectural support

- Ex : Sun Ultra servers, multiprocessor PCs, workstation cluster & IBM SP



Your Turn

Compare between SIMD and MIMD


Logical
Organization
Programmer's
view of platform
1
2
ways to express
parallel task
Mechanism for
specifying interaction
between tasks
Control structures
Communication models

Communication Models

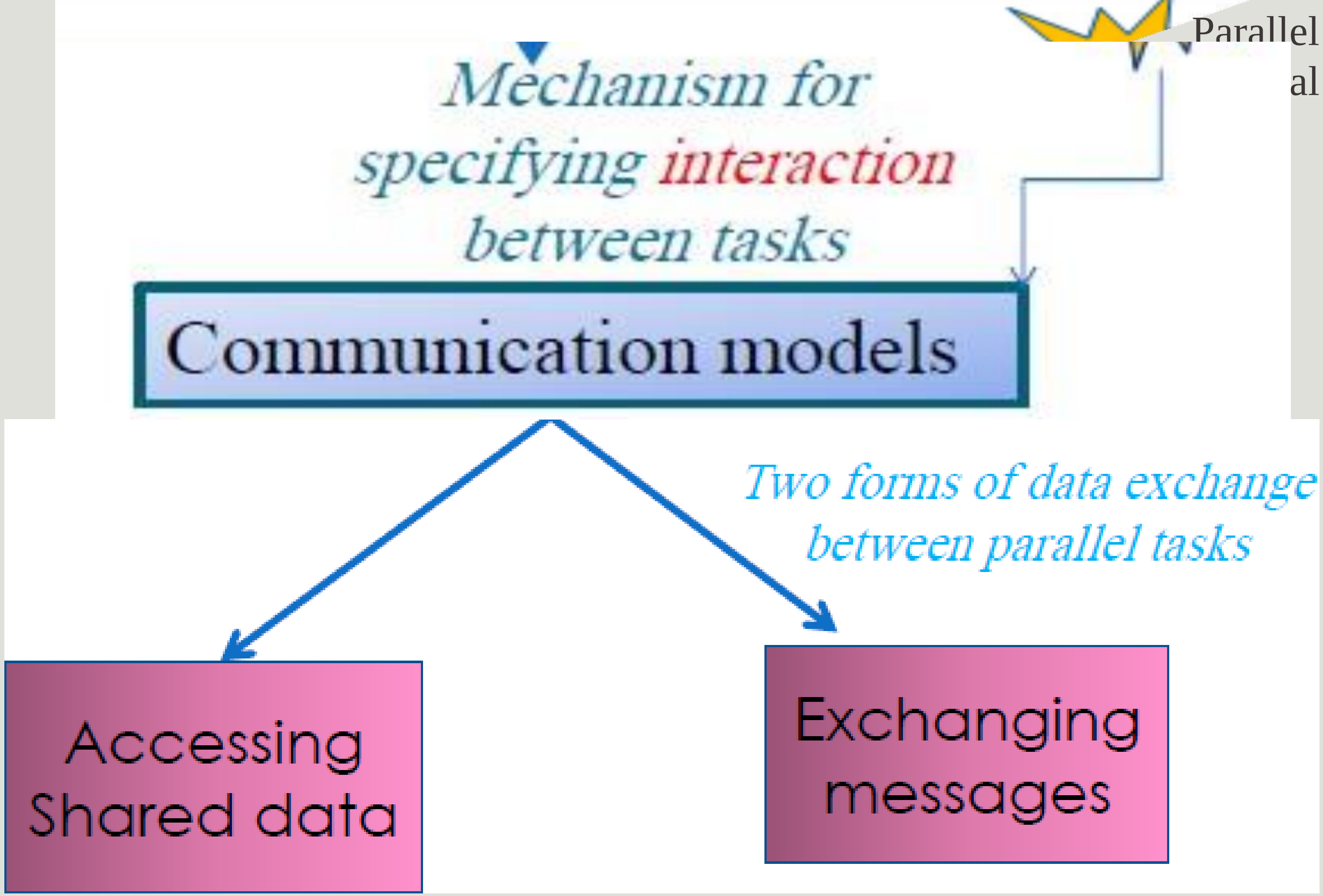
- Communication models in parallel computing define how processes or threads in a parallel system exchange data and information.

- These models specify the rules and mechanisms for communication among the components of a parallel system, which can include processes, threads, or even distributed nodes in a cluster or supercomputer.

- Different communication models are used depending on the architecture of the parallel system and the communication needs of the application.

Communication Models

- ❖ Shared Memory: Processes or threads communicate by reading and writing to a shared memory space..
- ❖ Distributed Memory: Each process has its own local memory, and communication between processes occurs by explicitly transferring data between them or message passing.
- ❖ Remote Procedure Call (RPC): Is a communication model where a process can invoke procedures or functions on a remote process as if they were local. It abstracts the communication details and provides a way for distributed processes to call functions on remote machines.
- ❖ Publish-Subscribe: Processes can subscribe to events or topics and publish messages. Subscribers receive messages published (e.g. MQTT).
- ❖ Shared Virtual Memory: An abstraction that allows distributed processes to access a common virtual address space [physically distributed](e.g. Tradmark)



Platforms that provide a shared data space are called shared-address-space machines or multiprocessors

Platforms that support messaging are called message passing platforms or multi-computers.