CS 3006 Parallel and Distributed Computer

Fall 2022

1. Learn about parallel and distributed computer architectures.(1)
2. Implement different parallel and distributed programming paradigms and algorithms using Message-Passing Interface (MPI) and OpenMP.(4)
3. Perform analytical modelling, dependence, and performance analysis of parallel algorithms and programs.(2)
4. Use Hadoop or MapReduce programming model to write bigdata applications.(5)

Week # 1 – Lecture # 1, 2, 3

22nd,23rd, 24th August 2022

23rd, 24th, 25th Muḥarram ul Haram, 1444

Dr. Nadeem Kafi Khan

Lecture # 1 - Topics

- Introduction
- Definition and Architecture block diagram
  - Shared Memory Systems
  - Distributed Memory Systems

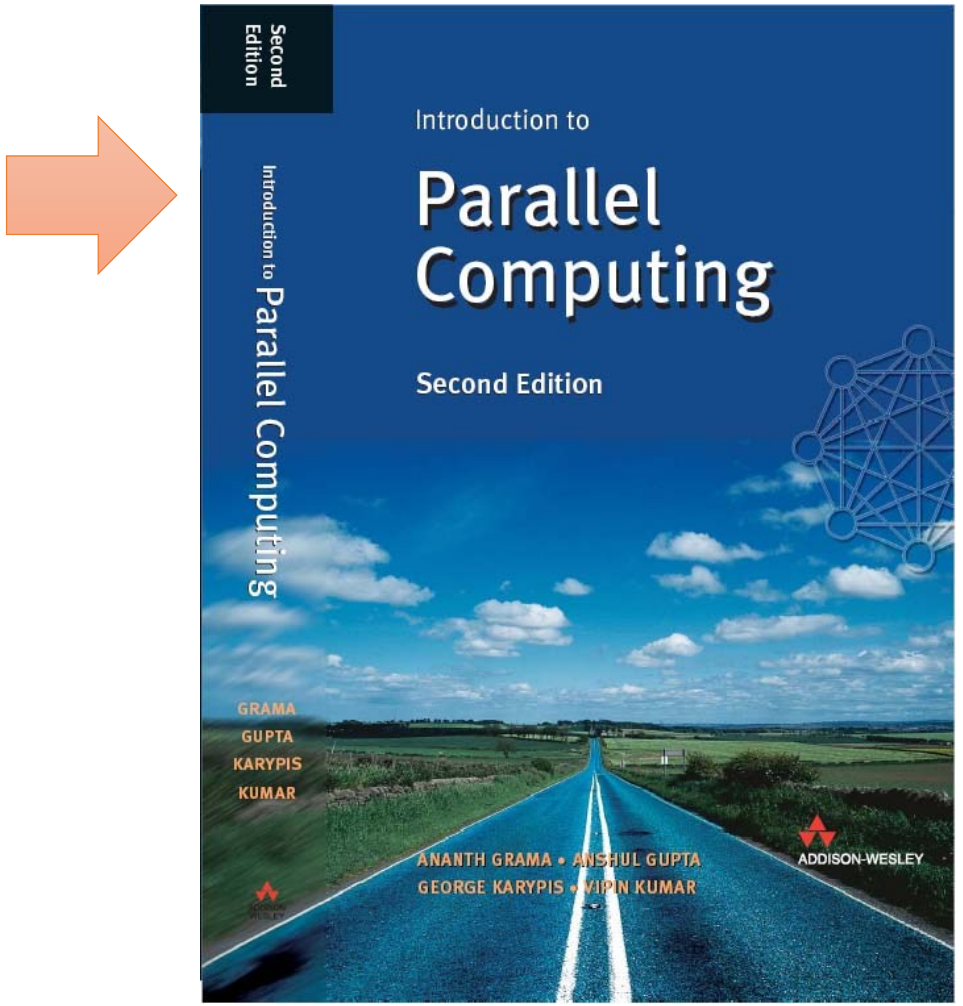
CS 3006

- Key course for BS (CS)
  - Most of the computation parallel and distributed and large-scale storage is now distributed.
- Course Instructor:
  - Dr. Nadeem Kafi Khan, Assistant Professor (CS)
  - Office: Main Campus, CS Block. Ext. 131.
  - Email: nadeem.kafi@nu.edu.pk (pls. send email from your @nu.edu.pk)
  - Please pay attention to my emails and Google classroom posts.
- Course slides and other materials will be posted on Google Classroom.
- Participation in class and on Google Classroom

CS 3006

- Textbook
  - Introductin to Parallel Computing 2nd Ed.
    By Ananth Grama, Anshul Gupta, George Karypis, Vipin Kumar

- Reference Materials
  - *Will be posted on Google classroom*



CS 3006

- Contact Hours
  - Lecture — See timetable
  - Consultancy Hours — Will be posted later
  - Interactions — Google Classroom and/or Email

- Course Pre-requisites
  - Programming, data structures and Operating Systems
  - Computer Organization and Assembly Language

Cornell Notetaking Method

Cue Column

2.5 Inches

- Main Ideas
- Questions that connect points
- Diagrams
- Study prompts

When?
After class
During review

Notes Column

6 Inches

- Main lecture notes here
- Use concise sentences
- Use shorthand symbols
- Use abbreviations
- Use lists
- Put space between points

When?
During class

Summary Column

- For top level, main ideas
- Use as a quick reference area

When?
After class
During review

2"

CS 3006

- Evaluation Criteria

- Active reading of textbook REQUIRED.
- Plagiarism will be marked as Zero.
- Late submissions are not allowed.
- Required Attendancee 80%

- Assignments **(14%) – Lab based** **(Last two will constitute Semester Project)**
- Quiz **(6%)**
- Mid Term **(15+15=30%)**
- Final **(50%)**

Marginal Improvement **Huge Impact**

	'04-'05	'06-'10	'11-'16
Rank	100+	3	1
Prize Money	$0.3m	$5m	$14m
% **Matches** Won	49%	79%	90%
% **Points** Won	49%	52%	55%

Theory

Topics: Shared Memory and Distributed Memory systems, Parallel Execution terms, Scalability, Parallel Overhead, Flynn's Taxonomy

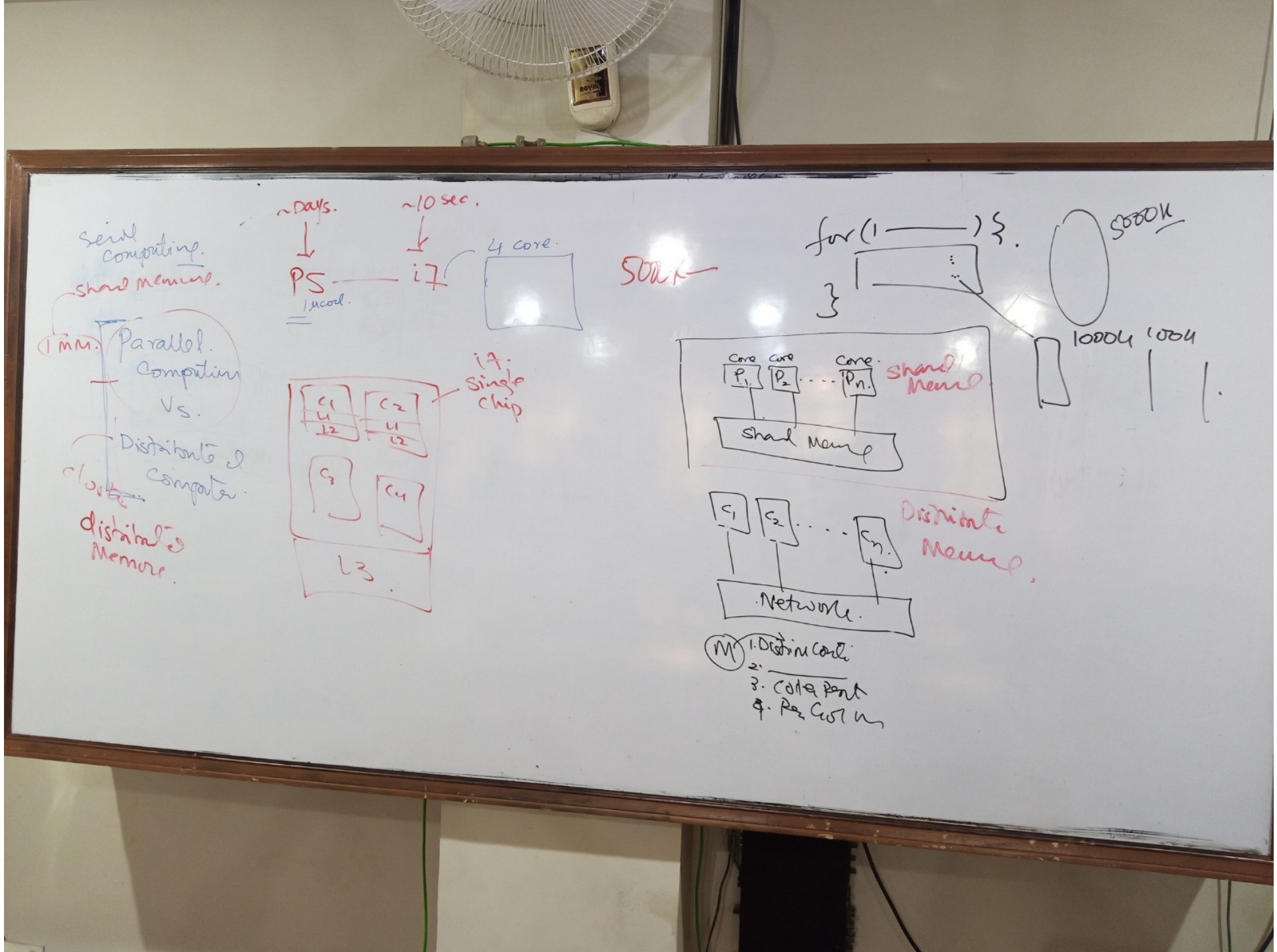
Learning Goals for Week # 1 based on topics

- Motivation: How to speed up a large computational problem. Executes using multiple threads (OpenMP) and distributed computation on cluster [multiple computers connected using an external network] (MPI). Why do we need parallel and distributed computing?
- What changes are required in the hardware architecture and software tools stack? i) Definitions of Parallel and Distributed Computing with basic hardware architecture. ii) Software tools and techniques for parallel and distributed platforms. OpenMP, MPI
- Two key issues: i) What type of overhead to expect in both types of PDC? How far can we scale parallel and distributed computing paradigms? iii) cost of both types of PDC: hardware infrastructure (costly), developing new tools, trained personnel with an understanding of parallel hardware and tools.
- Flynn's Taxonomy: Identify CPU, Memory, Network, How data is stored and distributed to different processors?

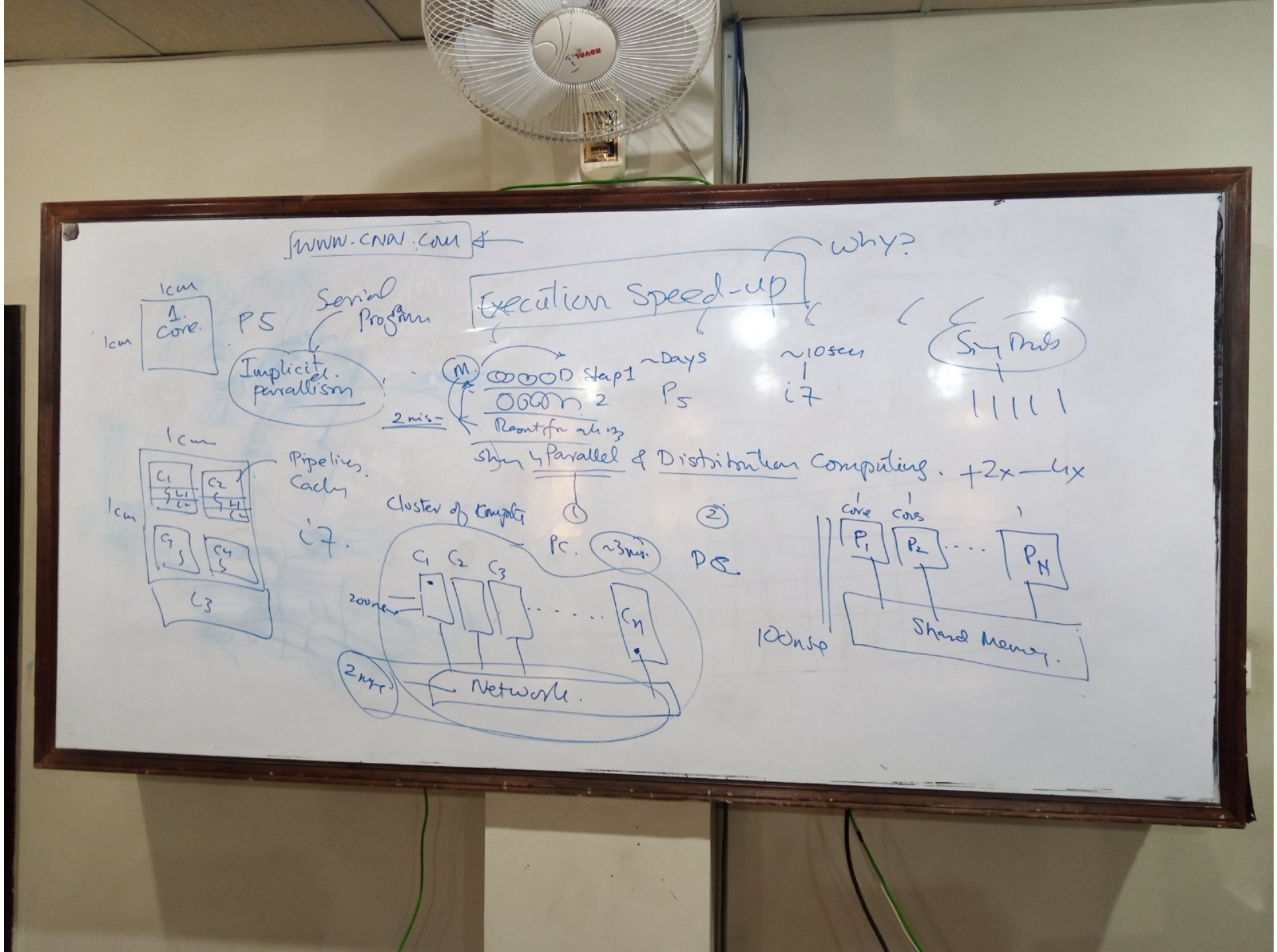
PDC topics Discussed in Lecture # 1

- Motivation for Parallel and Distributed Computing
- Why we need PDC? …Real world example(s).
- Parallel Computing paradigm
  - Shared Memory architecture exploited by multi-threaded.
- Distributed Memory paradigm
  - Distributed memory architecture exploited by multi-processes.
  - The computational task submitted to a master process, which distributed work (execution of code or processing of data) to other slave processes running on different computers of the cluster. The slave processes will execute the task in parallel and send results to master which is responsible to display results.
  - Why a cluster of 60 computers is a distributed system?

BSC-5C


Serial computing.
Shard memory.
1 MM.
Parallel Computing
Vs.
Distributed Computer.
distributed Memory.
~Days.
~10 sec.
P5
1 core.
i7
4 core.
i7.
Single chip
C1
C2
C3
C4
L1
L2
L3
50000
for(1 ———){.
50000
10000
1000
Core
Core
Core
P1
P2
Pn.
Shard Memory
Shard Memory
C1
C2
Cn.
Distributed Memory.
Network.
M
1. Distrim Code
2:
3. Code Part
4. Res Collection

BSC-5E


www.CNAI.com
Execution Speed-up
Why?
Serial Program
Implicit parallism
P5
i7
~Days
~10sec
Sys Bus
Pipelining
Cache
Cluster of Computers
Network
Shared Memory
Parallel & Distributed Computing
+2x — 4x

Lecture # 2 - Topics

- Parallel Execution Terms and their definitions
- Scalability

PDC CLOs as per FAST-NU official document

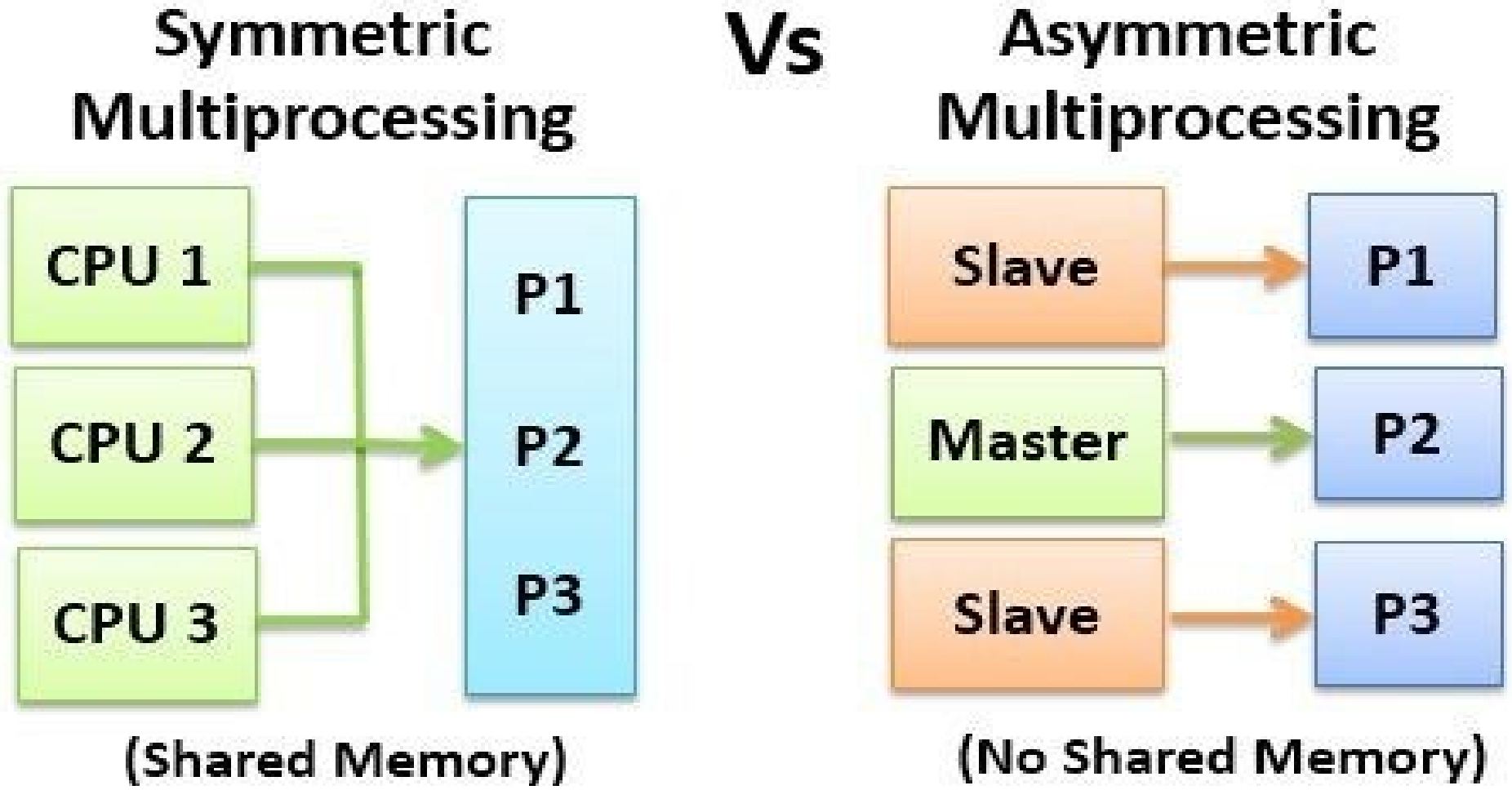
1. Learn about parallel and distributed computer architectures.(1)
2. Implement different parallel and distributed programming paradigms and algorithms using Message-Passing Interface (MPI) and OpenMP.(4)
3. Perform analytical modelling, dependence, and performance analysis of parallel algorithms and programs.(2)
4. Use Hadoop or MapReduce programming model to write bigdata applications.(5)

Distributed Computing	Parallel Computing
In distributed computing, a number of unified computers work towards a common task while communicating with each other with the help of message passing	In parallel computing, a task is divided into multiple sub-task which are then allotted to different processors on the same computer system.
Number of Computer Systems Involved	
Multiple physical computer systems are present in the same computer system.	A single physical computer system hosts multiple processors.
Dependency Between Processes	
There might not be much dependency between the processes.	There is more dependency between the process. Output of one might be the input of another.
Scalability	
The systems are easily scalable as there is no limitation on how many systems can be added to a network.	The systems that implement parallel computing have limited scalability.
Resource Sharing	
Computers have their own memory and processors.	All the processors share the same memory.
Synchronization	
The computers in the network have to implement synchronization algorithms.	All processors use the same master clock for synchronization.
Usage	
Generally preferred in places requiring high scalability.	Generally preferred in places requiring faster speed and better performance.

Some General Parallel Terminology

- **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.
- **Parallel Task**
  - A task that can be executed by multiple processors safely (yields correct results)
- **Serial Execution**
  - Execution of a program sequentially, one statement at a time. In the simplest sense, this is what happens on a one processor machine. However, virtually all parallel tasks will have sections of a parallel program that must be executed serially.

Symmetric vs. Asymmetric Multiprocessing Architecture

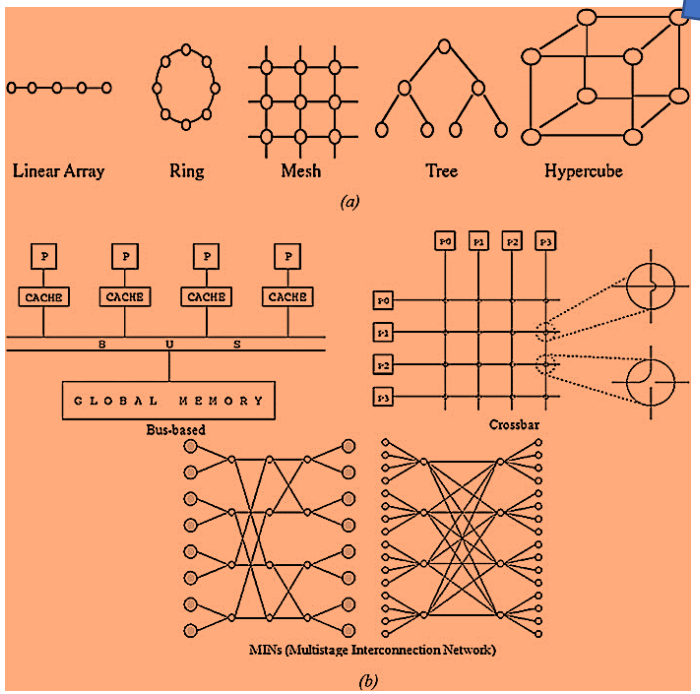


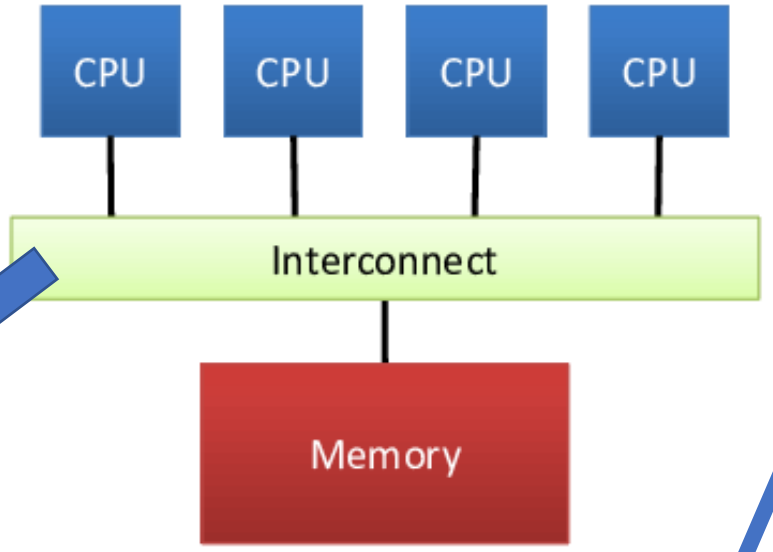
- Same type of processing elements vs Different type of processors element used in computations.
- Same type of Computation vs Different type of computations done of same processing elements.

Some General Parallel Terminology

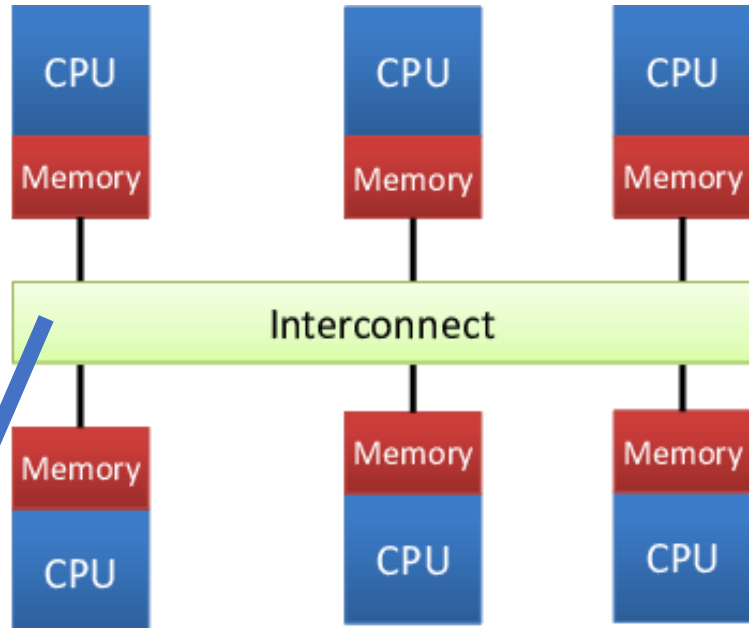
- **Parallel Execution**
  - Execution of a program by more than one task, with each task being able to execute the same or different statement at the same moment in time.
- **Shared Memory**
  - From a strictly hardware point of view, describes a computer architecture where all processors have direct (usually bus based) access to common physical memory. In a programming sense, it describes a model where parallel tasks all have the same "picture" of memory and can directly address and access the same logical memory locations regardless of where the physical memory actually exists.
- **Distributed Memory**
  - In hardware, refers to network based memory access for physical memory that is not common. As a programming model, tasks can only logically "see" local machine memory and must use communications to access memory on other machines where other tasks are executing.

Shared Memory vs. Distributed Memory





a) Shared Memory



b) Distributed Memory

This network interconnect is either very high-speed Ethernet switch ~10Gbit or even higher Infiniband or other switches

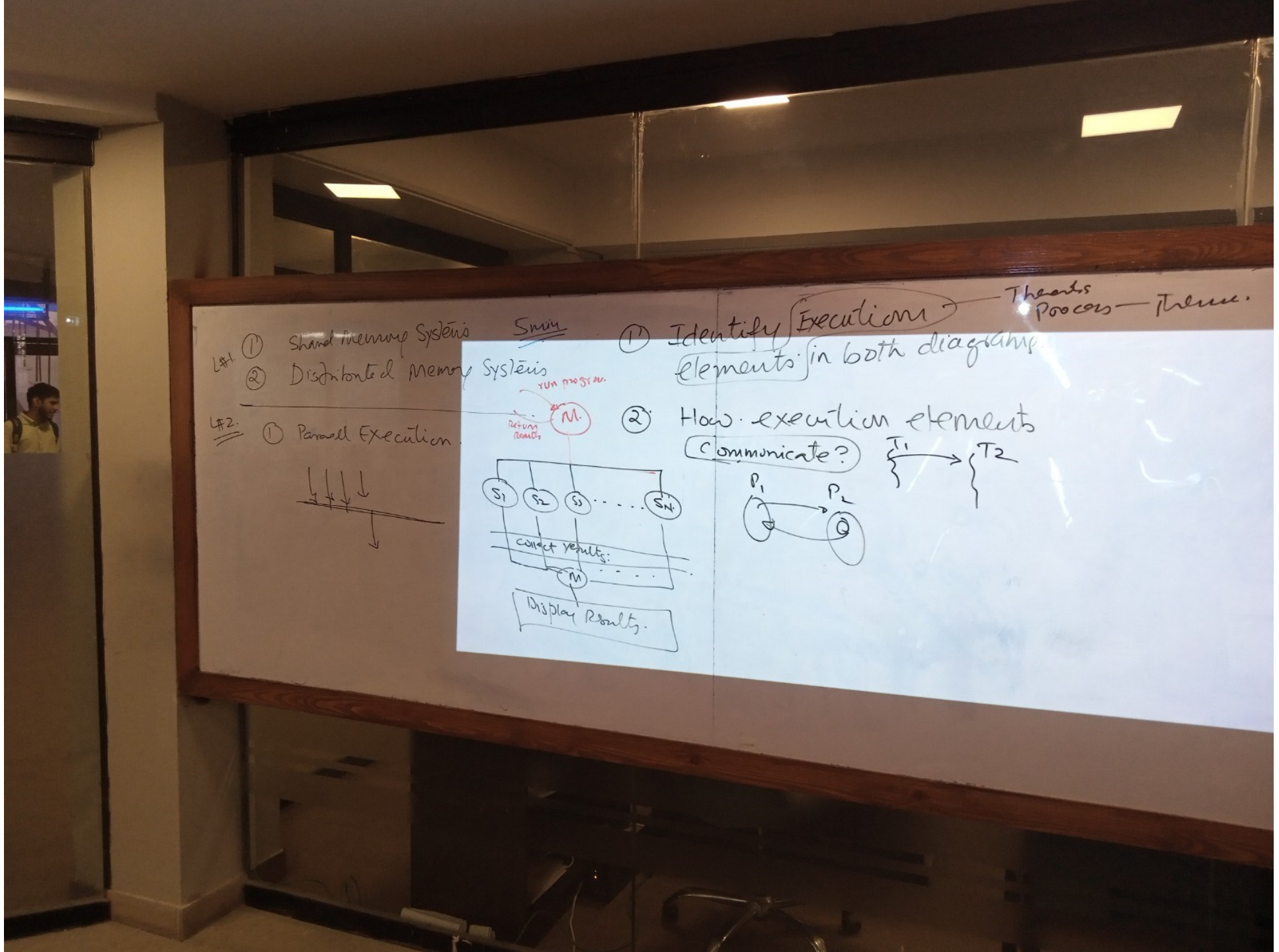
Some General Parallel Terminology

- **Communications**
  - Parallel tasks typically need to exchange data. There are several ways this can be accomplished, such as through a shared memory bus or over a network, however the actual event of data exchange is commonly referred to as communications regardless of the method employed.
- **Synchronization**
  - The coordination of parallel tasks in real time, very often associated with communications. Often implemented by establishing a synchronization point within an application where a task may not proceed further until another task(s) reaches the same or logically equivalent point.
  - Synchronization usually involves waiting by at least one task, and can therefore cause a parallel application's wall clock execution time to increase.

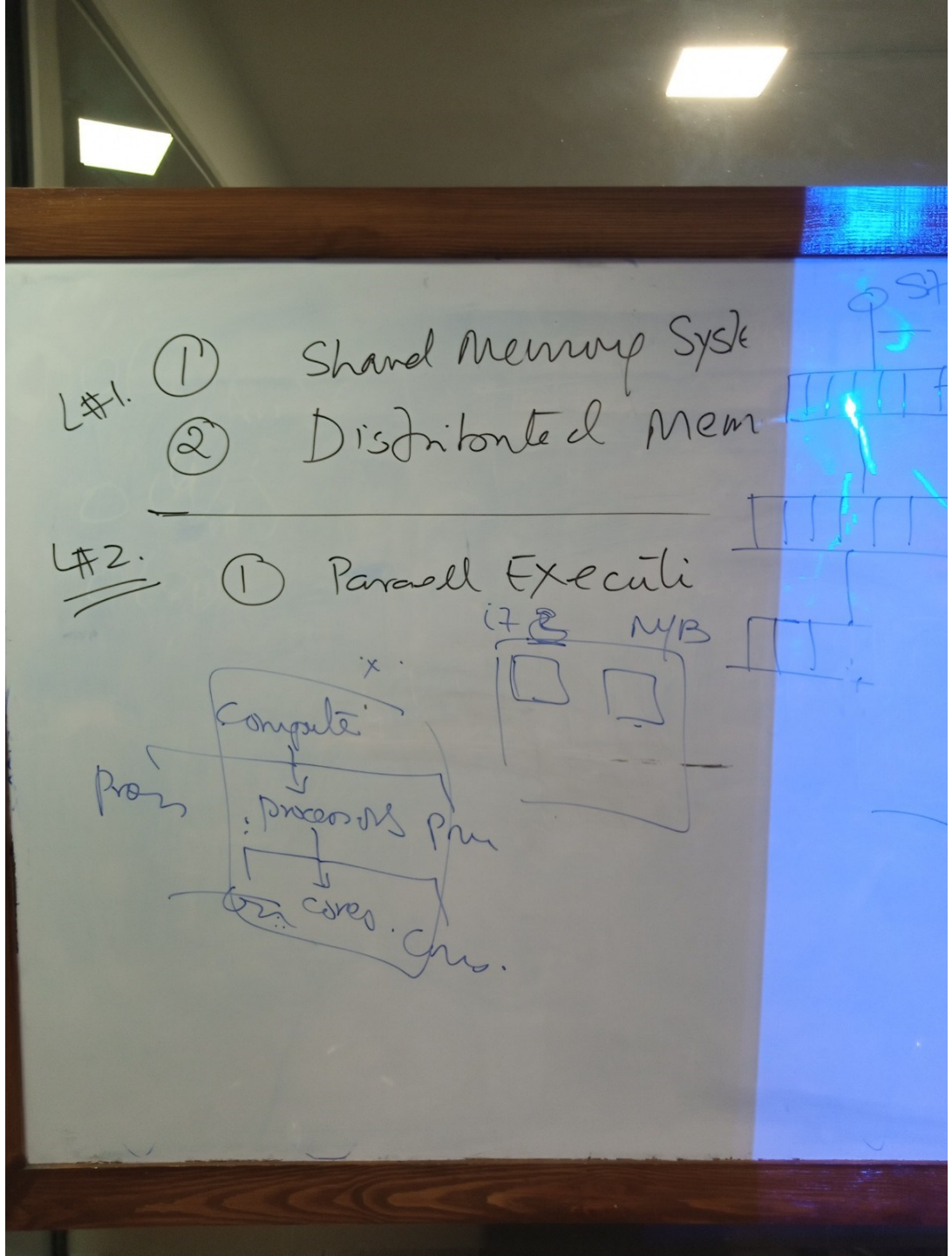
Some General Parallel Terminology

- **Scalability**
  - Refers to a parallel system's (hardware and/or software) ability to demonstrate a proportionate increase in parallel speedup with the addition of more processors. Factors that contribute to scalability include:
    - Hardware - particularly memory-cpu bandwidths and network communications
    - Application algorithm
    - Parallel overhead related
    - Characteristics of your specific application and coding

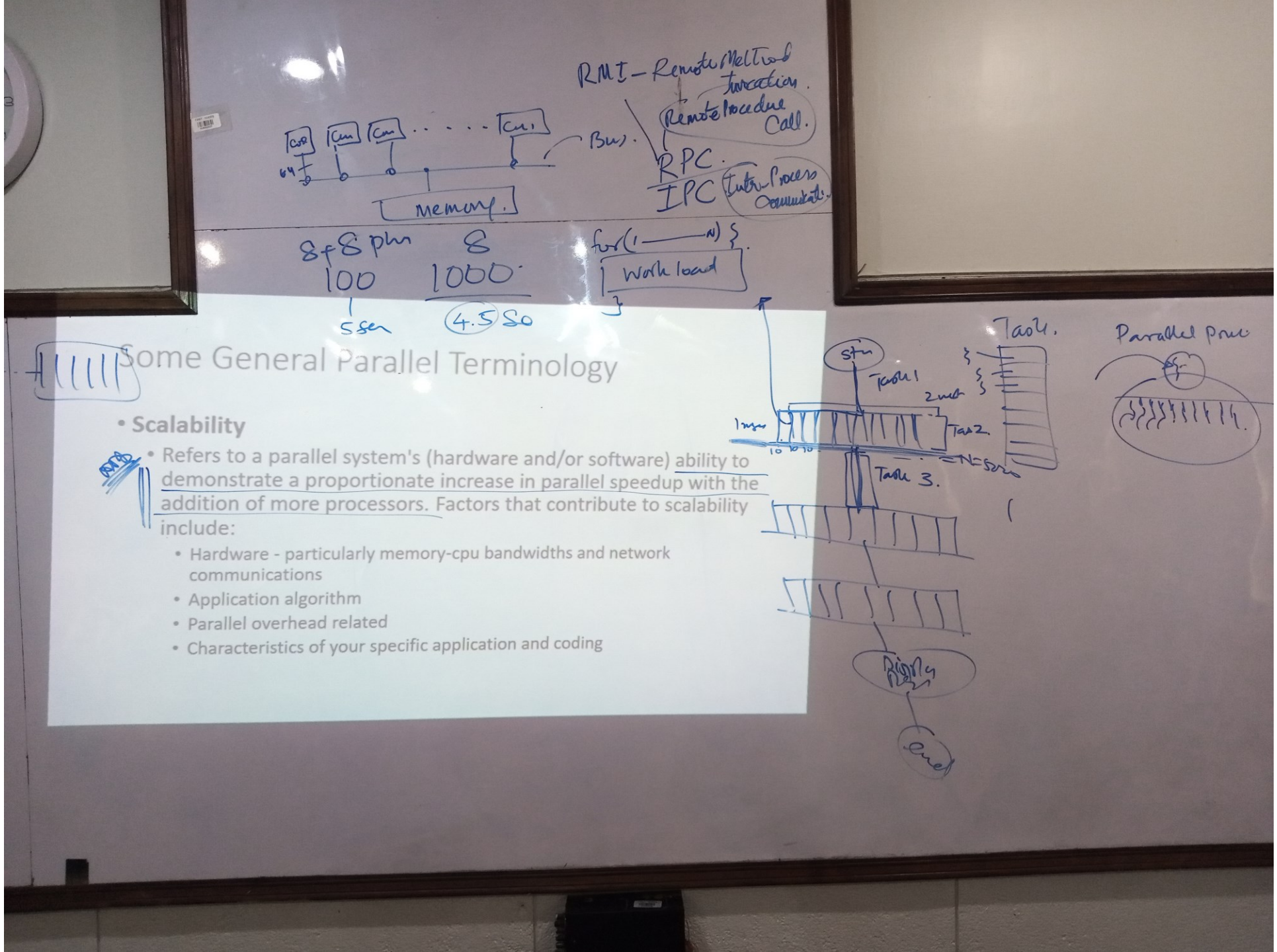
BSC-5E


L#1.
1. Shared memory Systems
2. Distributed Memory Systems
L#2.
1. Parallel Execution
Smiy
run program
M.
return results
S1
S2
S3
SN
collect results
M
Display Results.
1. Identify Execution elements in both diagrams
Threads
Process
Thread.
2. How execution elements communicate?
P1
P2
T1
T2

BSC-5E


L#1.
1
Shared Memory Syste
2
Distributed Mem
L#2.
1
Parallel Executi
i7
N/B
x.
Compute
Proc
processors Pro
cores. Cpu.

BSC-5C


RMI – Remote Method Invocation
Remote Procedure Call.
RPC.
IPC
Inter-Process Communication
Bus.
Memory.
8 of 8 plus
100
5 sec
8
1000
4.5 sec
for(1 ——— N) {
Work load
}
Some General Parallel Terminology
• Scalability
• Refers to a parallel system's (hardware and/or software) ability to demonstrate a proportionate increase in parallel speedup with the addition of more processors. Factors that contribute to scalability include:
• Hardware - particularly memory-cpu bandwidths and network communications
• Application algorithm
• Parallel overhead related
• Characteristics of your specific application and coding
Task.
Parallel proc
Task 1
Task 2
Task 3.

Lecture # 3 - Topics

- Overhead in Parallel and Distributed Computing
- Speed-up and Amdahl Law
- Flynn’s Taxonomy
- Granularity

Some General Parallel Terminology

- **Parallel Overhead**
  - The amount of time required to coordinate parallel tasks, as opposed to doing useful work. Parallel overhead can include factors such as:
    - Task start-up time
    - Synchronizations
    - Data communications
    - Software overhead imposed by parallel compilers, libraries, tools, operating system, etc.
    - Task termination time
- **Massively Parallel**
  - Refers to the hardware that comprises a given parallel system - having many processors. The meaning of many keeps increasing, but currently IBM Blue Gene/L pushes this number to 6 digits.

Some General Parallel Terminology

- **Observed Speedup**
  - Observed speedup of a code which has been parallelized, defined as:

$$\frac{\text{wall-clock time of serial execution}}{\text{wall-clock time of parallel execution}}$$

  - One of the simplest and most widely used indicators for a parallel program's performance.

The Amdahl's Law formula is

$$overall\ speedup = \frac{1}{(1-P) + \frac{P}{S}}$$

- P is the time proportion of the algorithm that can be parallelized.
- S is the speedup factor for that portion of the algorithm due to parallelization.

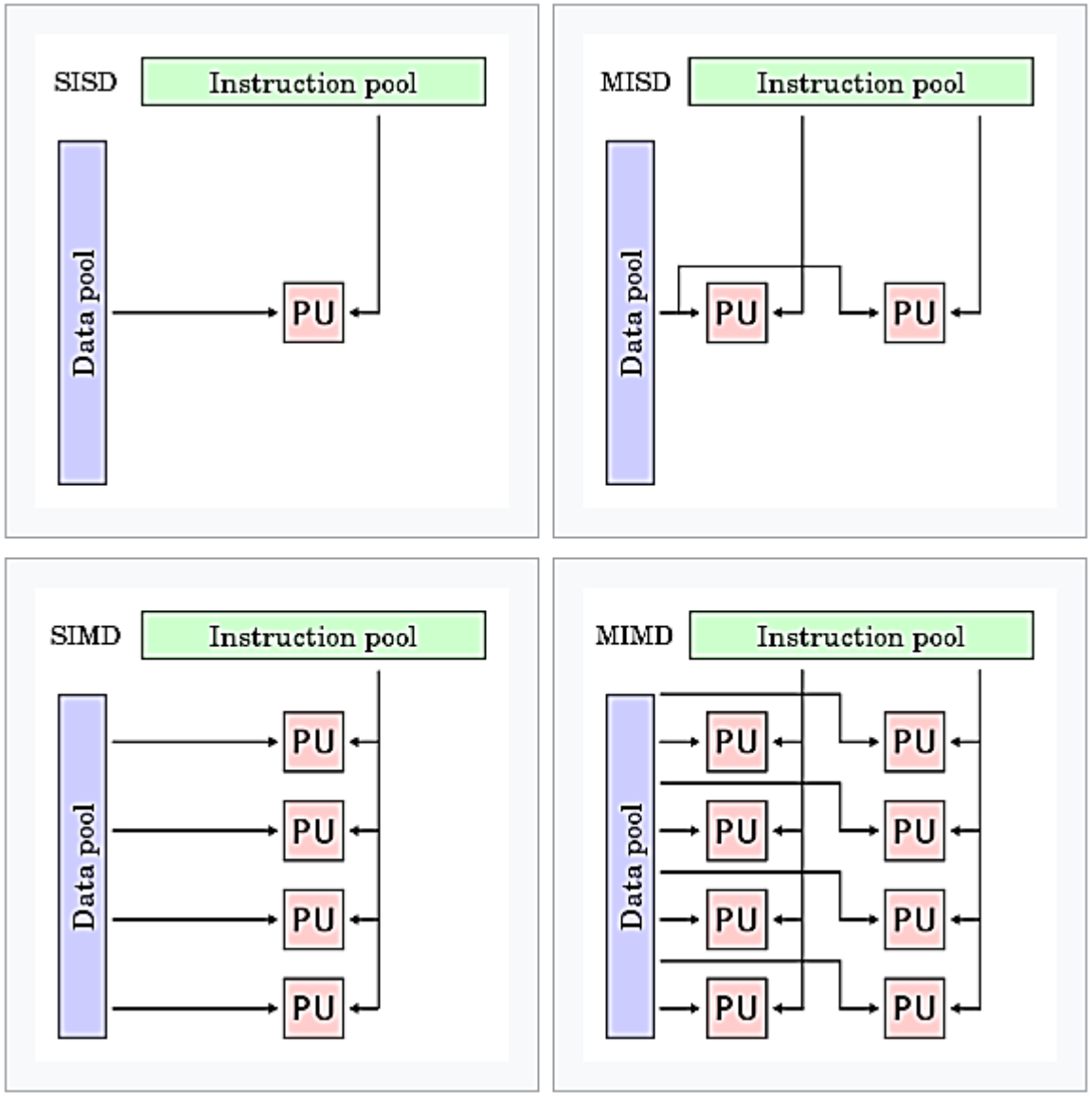
For example, suppose that we use our strategy to search for primes using 4 processors, and that 90% of the running time is spent checking 2k-digit random numbers for primality (after an initial 10% of the running time computing a list of k-digit primes). Then P = .90 and S = 4 (for 4-fold speedup). According to Amdahl's Law,

$$overall\ speedup = \frac{1}{(1-0.90) + \frac{0.90}{4}} = \frac{0.10}{0.225} = 3.077$$

This estimates that we will obtain about 3-fold speedup by using 4-fold parallelism.

Flynn's Taxonomy

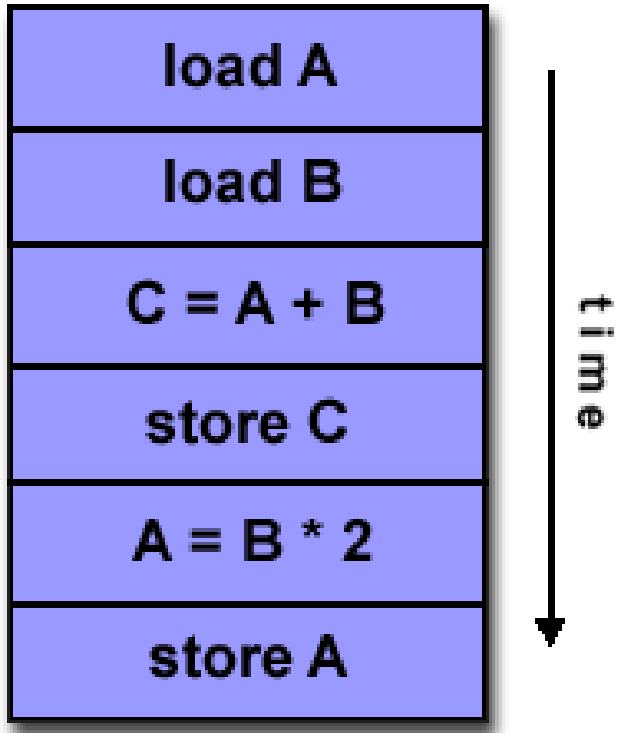
Data Streams \ Instruction Streams	one	many
one	**SISD** traditional von Neumann single CPU computer	**MISD** May be pipelined Computers
many	**SIMD** Vector processors fine grained data Parallel computers	**MIMD** Multi computers Multiprocessors



PU = Processing Unit

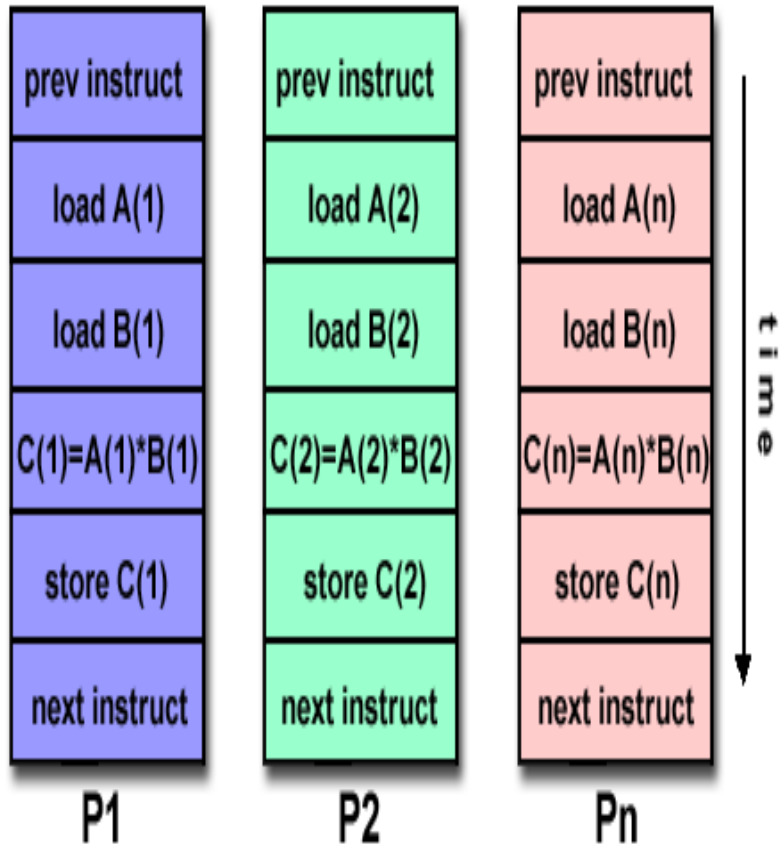
Single Instruction, Single Data (SISD)

- 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
- Deterministic execution
- This is the oldest and until recently, the most prevalent form of computer
- Examples: most PCs, single CPU workstations and mainframes



Single Instruction, Multiple Data (SIMD)

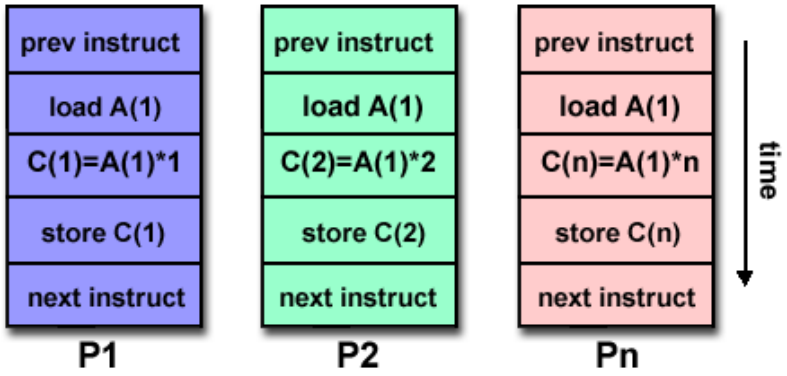
- A type of parallel computer
- **Single instruction**: All processing units execute the same instruction at any given clock cycle
- **Multiple data:** Each processing unit can operate on a different data element
- This type of machine typically has an instruction dispatcher, a very high-bandwidth internal network, and a very large array of very small-capacity instruction units.
- Best suited for specialized problems characterized by a high degree of regularity, such as image processing.
- Synchronous (lockstep) and deterministic execution
- Two varieties: Processor Arrays and Vector Pipelines



- Examples: Vectorization is a prime example of SIMD in which the same instruction is performed across multiple data. A variant of SIMD is single instruction, multi-thread (SIMT), which is commonly used to describe GPU workgroups.

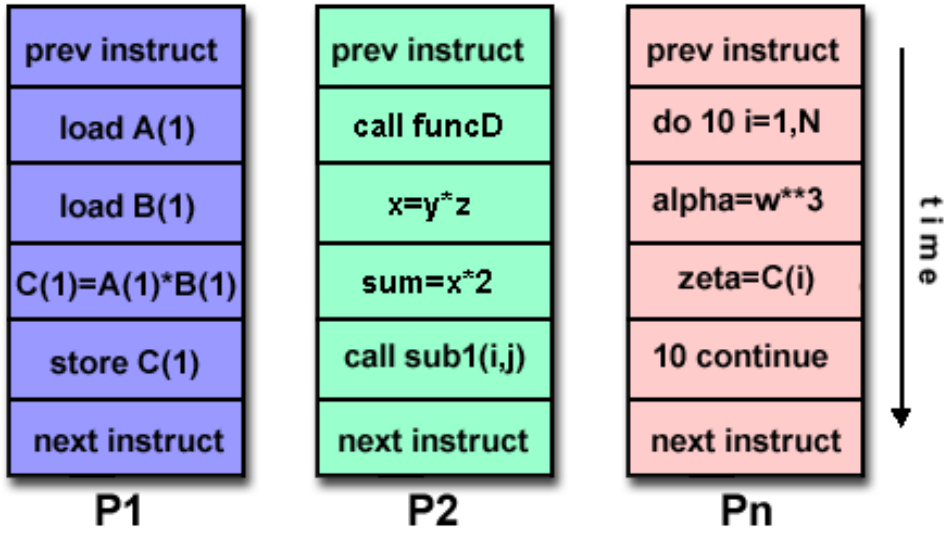
Multiple Instruction, Single Data (MISD)

- A single data stream is fed into multiple processing units.
- Each processing unit operates on the data independently via independent instruction streams. This is not a common architecture.
- Few actual examples of this class of parallel computer have ever existed. One is the expérimental Carnegie-Mellon C.mmp computer (1971).
- Some conceivable uses might be:
  - multiple frequency filters operating on a single signal stream
  - multiple cryptography algorithms attempting to crack a single coded message.
  - **Redundant computation on the same data. This is used in highly fault-tolerant approaches such as spacecraft controllers. Because spacecraft are in high radiation environments, these often run two copies of each calculation and compare the output of the two.**



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
- Execution can be synchronous or asynchronous, deterministic or n deterministic
- Examples: most current supercomputers, networked parallel com "grids" and multi-processor SMP computers - including some type PCs.
- The final category has parallelization in both instructions and data and is referred to as MIMD. This category describes multi-core parallel architectures that comprise the majority of large parallel systems.



Granularity

Granularity is 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.

Facilitates load balancing and Implies high communication overhead and less opportunity for performance enhancement

Coarse-grain Parallelism: Relatively large amounts of computational work are done between communication/synchronization events.
Harder to load balance efficiently