Introduction to Cloud Computing

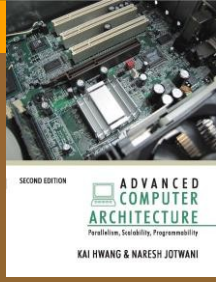
Parallel Processing I

15-319, spring 2010

7th Lecture, Feb 2nd

Majd F. Sakr

Mc Graw Hill Education



ADVANCE COMPUTER ARCHITECTURE

Second Edition

Kai Hwang & Naresh Jotwani

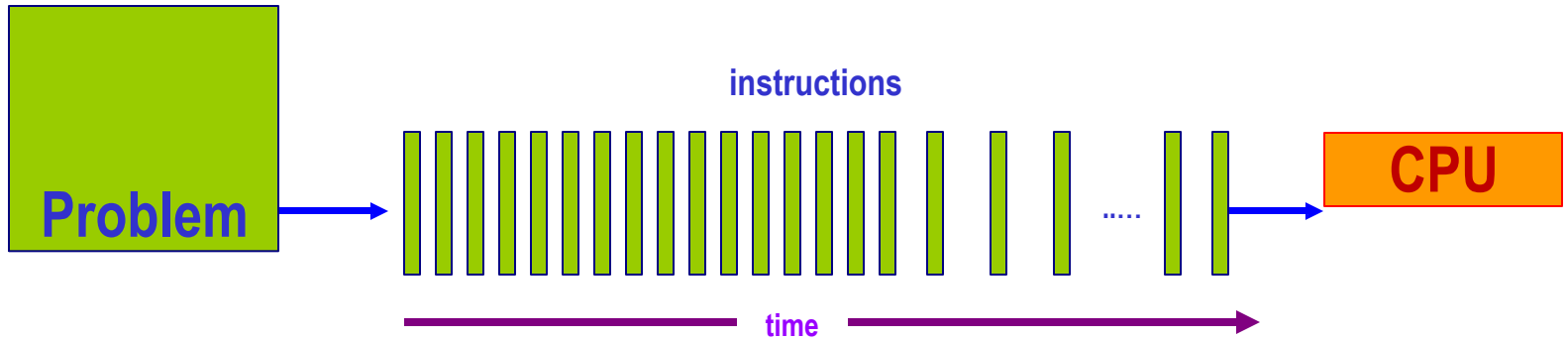
Lecture Outline

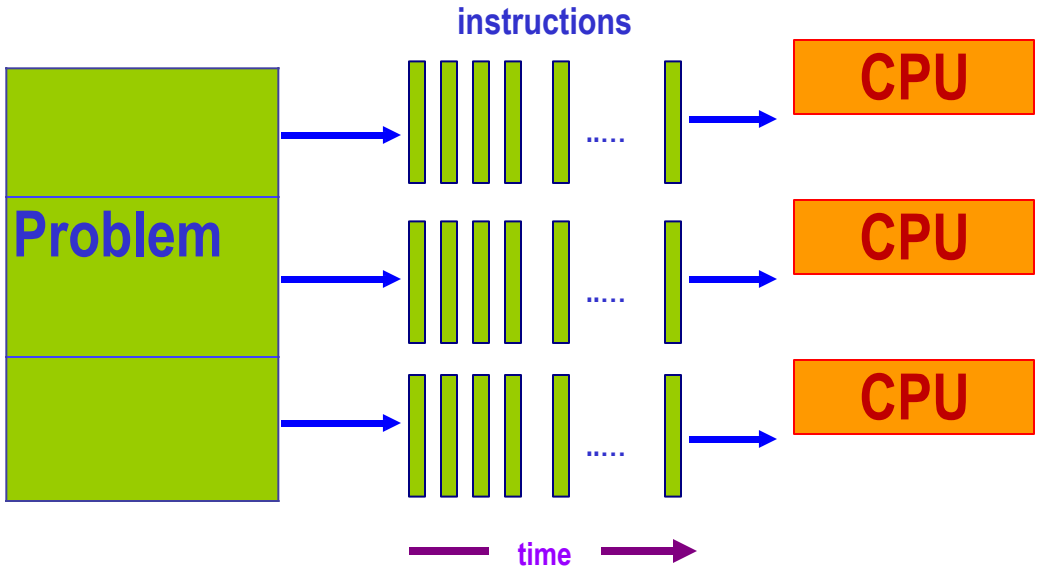
- What is Parallel Computing?
- Motivation for Parallel Computing
- Parallelization Levels - granularities
- Parallel Computers Classification
- Parallel Computing Memory Architecture
- Parallel Programming Models

Lecture Motivation

- Concurrency and why?
- Different flavors of parallel computing
- Get the basic idea of the benefits of concurrency and the challenges in achieving concurrent execution.

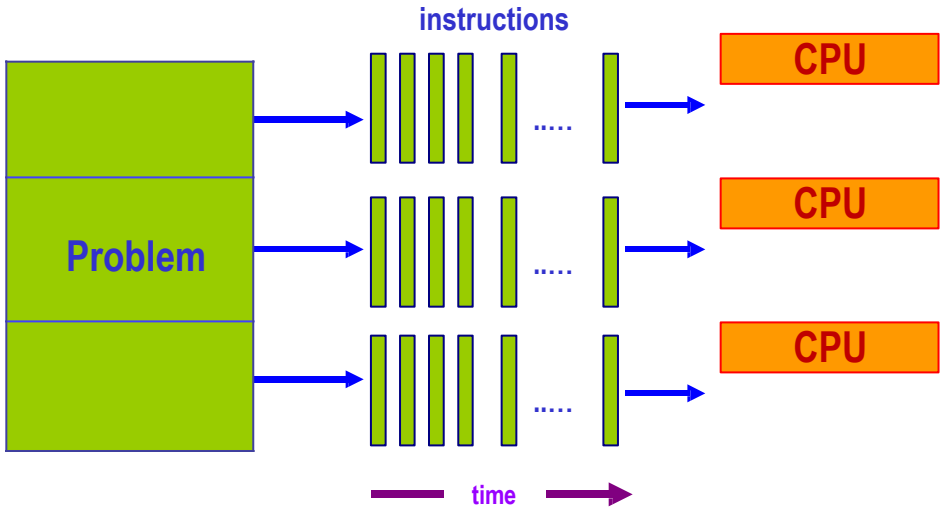
What is Parallel Computing?





Parallel Computing Resources

- Multiple processors on the same computer
- Multiple computers connected by a network
- Combination of both



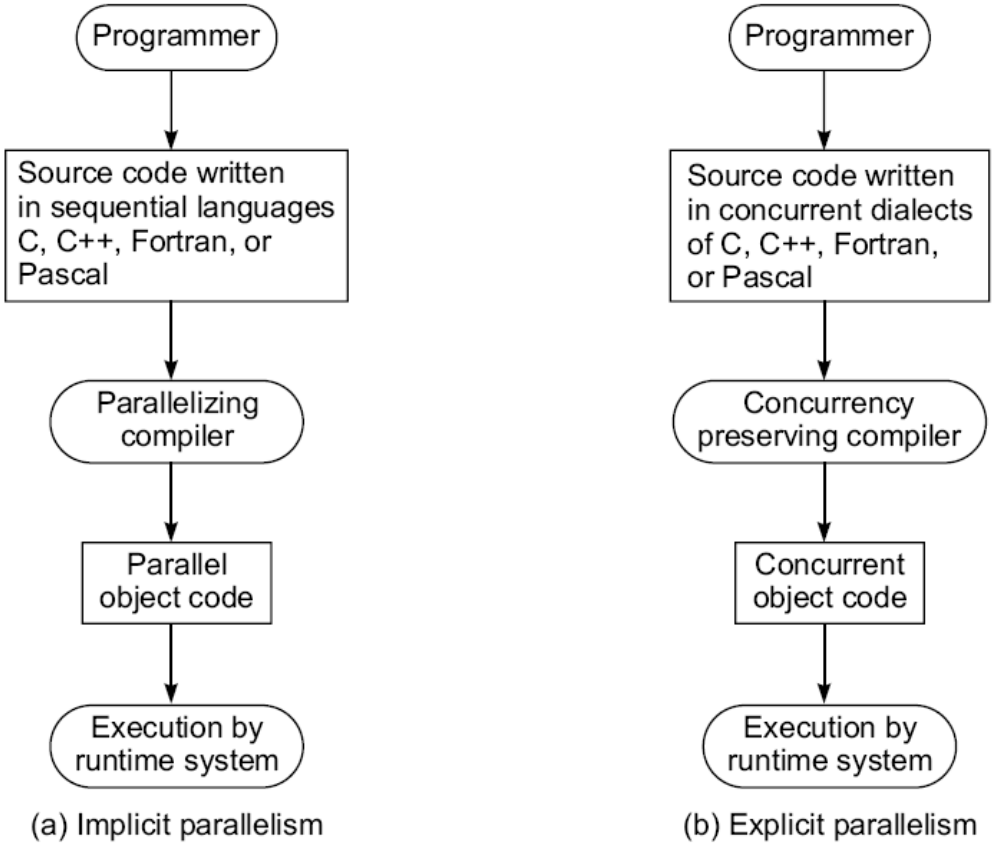


Fig. 1.5 Two approaches to parallel programming (Courtesy of Charles Seitz; adapted with permission from "Concurrent Architectures", p. 51 and p. 53, *VLSI and Parallel Computation*, edited by Suaya and Birtwistle, Morgan Kaufmann Publishers, 1990)

Parallel Random Access Machine

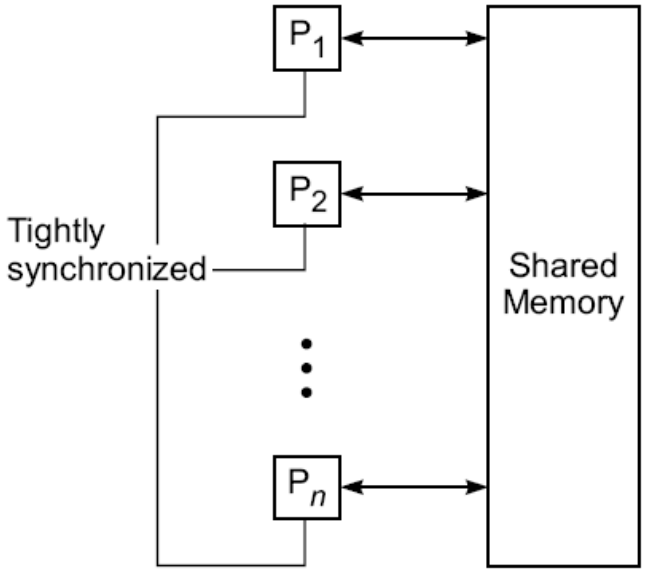


Fig. 1.14 PRAM model of a multiprocessor system with shared memory, on which all *n* processors operate in lockstep in memory access and program execution operations. Each processor can access any memory location in unit time

- Ideal machine
- Assumes
  - Synchronization
  - Scalability
  - No Time Delay
  - No Concurrency Problem
  - No Cache Coherence

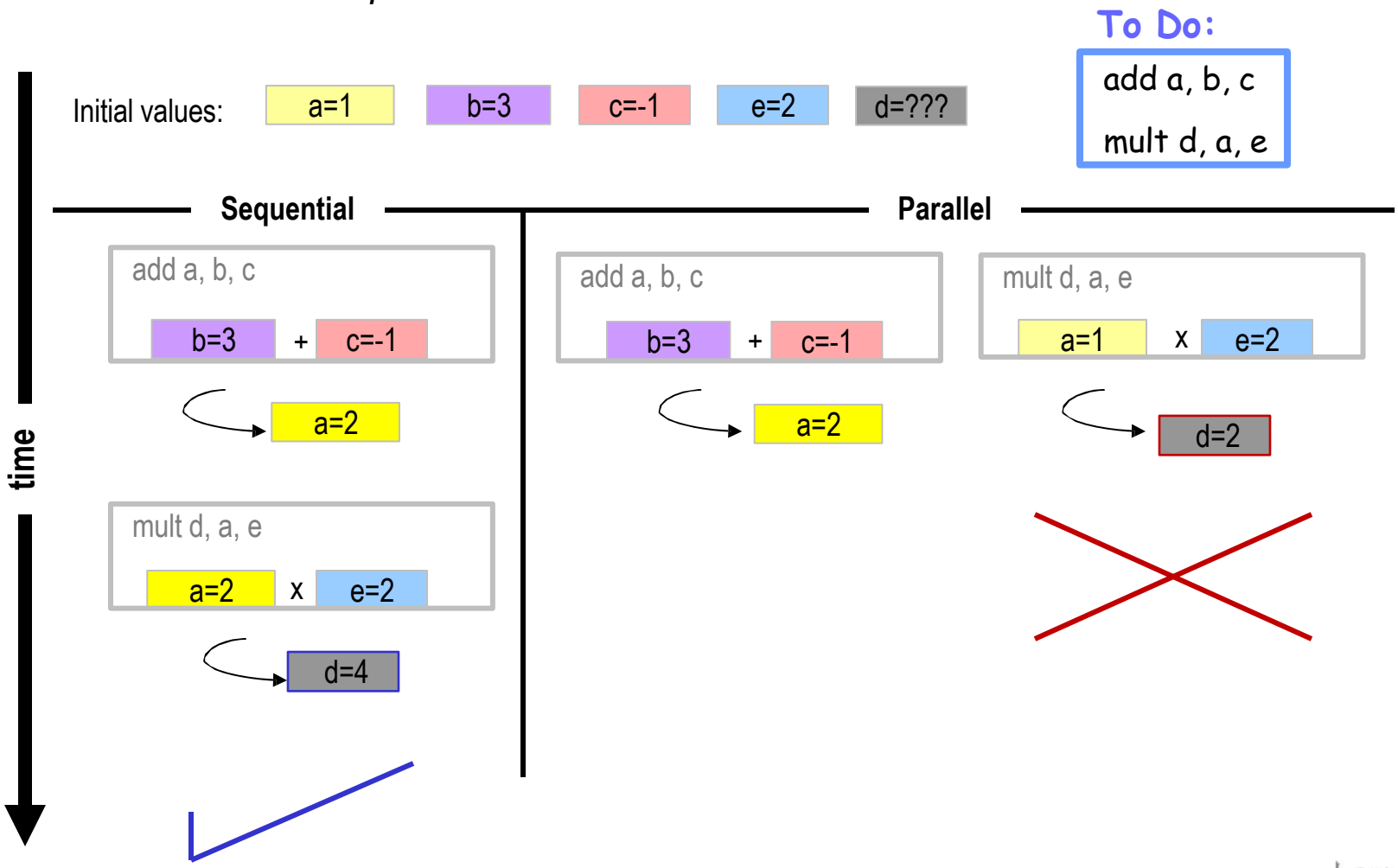
When can a computation be Parallelized? (1/3)

- **When using multiple compute resources to solve the problem saves it more time than using single resource**

- **At any moment, we can execute multiple program chunks**
  - Dependencies
    - Data Dependence
      - Flow Dependence
      - Anti Dependence
      - Output Dependence
      - IO Dependence
    - Resource Dependence
    - Control Dependence

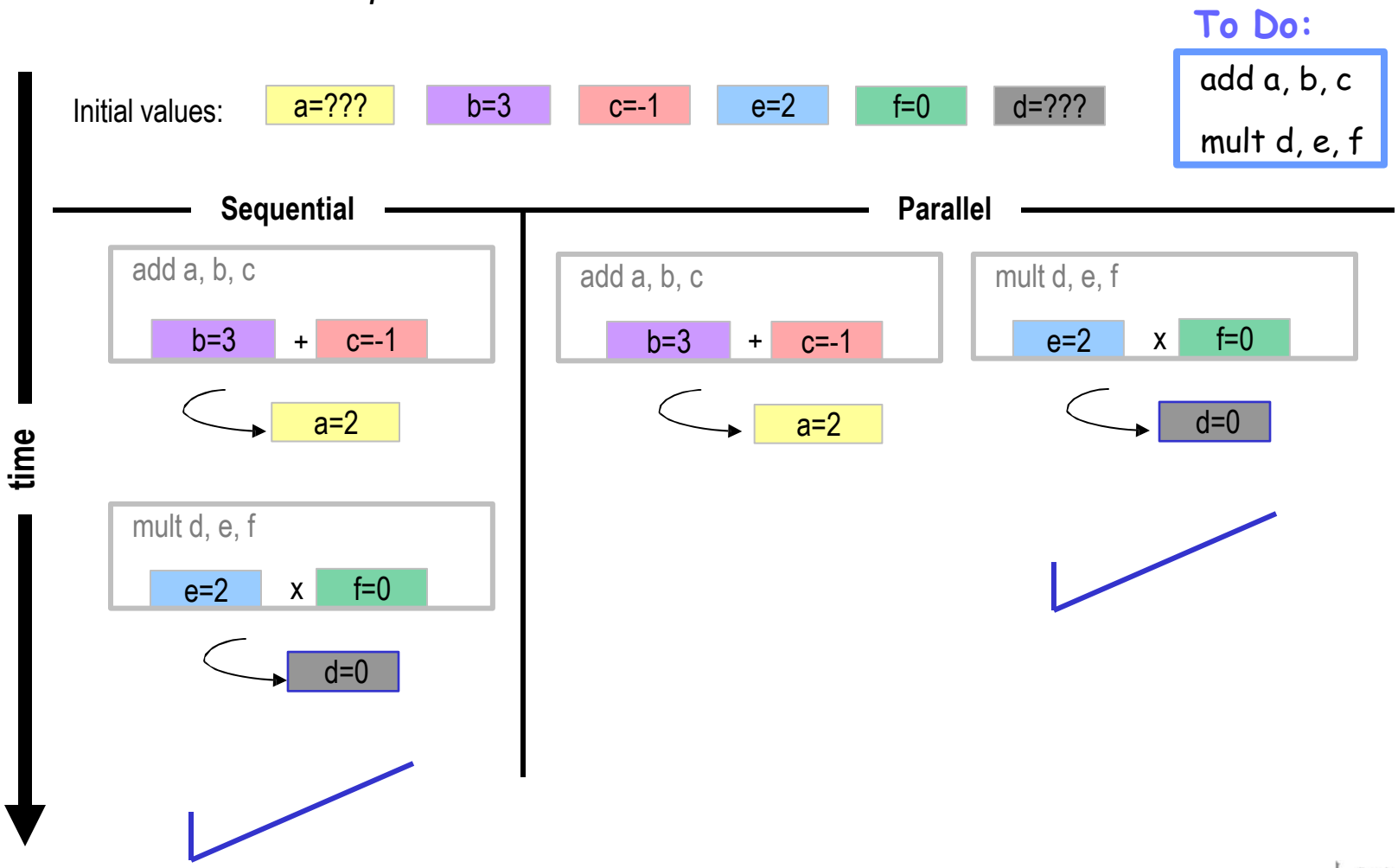
When can a computation be Parallelized? (2/3)

- **Problem's ability to be broken into discrete pieces to be solved simultaneously**
  - *Check data dependencies*



When can a computation be Parallelized? (3/3)

- **Problem's ability to be broken into discrete pieces to be solved simultaneously**
  - *Check data dependencies*



Dependencies

- **At any moment, we can execute multiple program chunks**
  - Dependencies
    - Data Dependence
      - Flow Dependence (Output of S1 overlaps with input of S2)
        - Load R1, A; Add R2, R1
      - Anti Dependence (Output of S2 overlaps with input of S1)
        - Add R2, R1; Move R1, R3
      - Output Dependence (Add R2, R1; Move R2, R3)
      - IO Dependence
    - Resource Dependence
    - Control Dependence
    - Unknown Dependence
      - Subscript is itself subscribed
      - Subscript does not have loop index variable
      - Non linear subscript
      - Subscript appears more than once with subscripts having different coefficients

Assignment 2

Give an example of the Unknown dependence. Use any suitable example for your illustration.

Bernstein's Conditions - 1

- Bernstein's revealed a set of conditions which must exist if two processes can execute in parallel.
- Notation
  - *Let P1 and P2 be two processes .*
  - I_i is the set of all input variables for a process P_i.
  - O_i is the set of all output variables for a process P_i.
- If P_1 and P_2 can execute in parallel (which is written as $P_1 \parallel P_2$), then:

$$I_1 \cap O_2 = \varnothing$$

$$I_2 \cap O_1 = \varnothing$$

$$O_1 \cap O_2 = \varnothing$$

Bernstein's Conditions: An Example

- For the following instructions P_1, P_2, P_3, P_4, P_5 in program order and
  - Instructions are in program order
  - Each instruction requires one step to execute
  - Two adders are available

$P_1 : C = D \times E$

$P_2 : M = G + C$

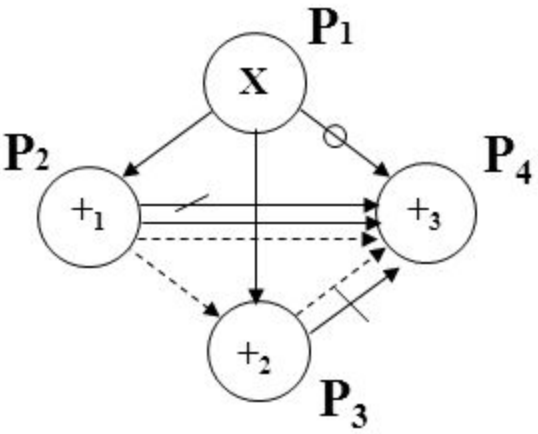
$P_3 : A = B + C$

$P_4 : C = L + M$

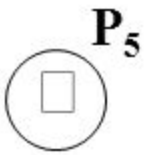
$P_5 : F = G \div E$

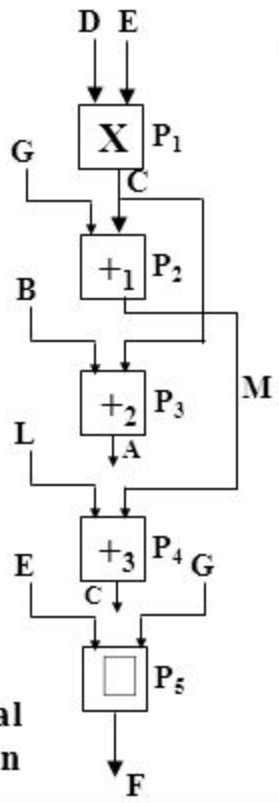
Using Bernstein's Conditions after checking statement pairs:

$P_1 \parallel P_5$, $P_2 \parallel P_3$, $P_2 \parallel P_5$, $P_5 \parallel P_3$, $P_4 \parallel P_5$



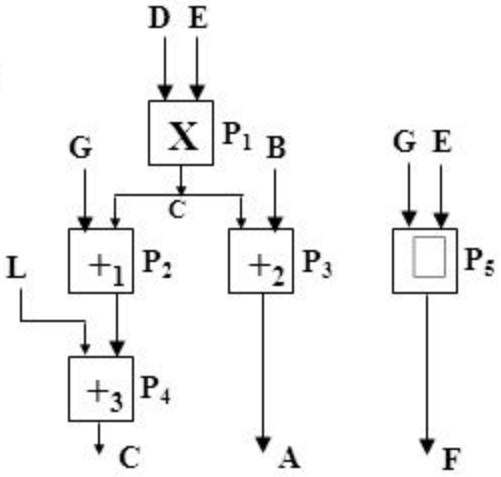
Dependence graph:
Data dependence (solid lines)
Resource dependence (dashed lines)





Sequential execution

Time



Parallel execution in three steps assuming two adders are available per step

For the previous example calculate the sequential time of execution and parallel time of execution considering:

- Addition takes 4 clock cycles
- Multiplication takes 12 clock cycles and
- Division takes 18 clock cycles

Now, Calculate the Speedup in each case

Lecture Outline

- What is Parallel Computing?
- Motivation for Parallel Computing
- Parallelization Levels
- Parallel Computers Classification
- Parallel Computing Memory Architecture
- Parallel Programming Models

Why Parallel Computing? (1/2)

- **Why not simply build faster serial computer??**

"In pioneer days they used oxen for heavy pulling, and when one ox couldn't budge a log, they didn't try to grow a larger ox. We shouldn't be trying for bigger computers, but for more systems of computers."

—Admiral Grace Hopper

Why Parallel Computing? (1/2)

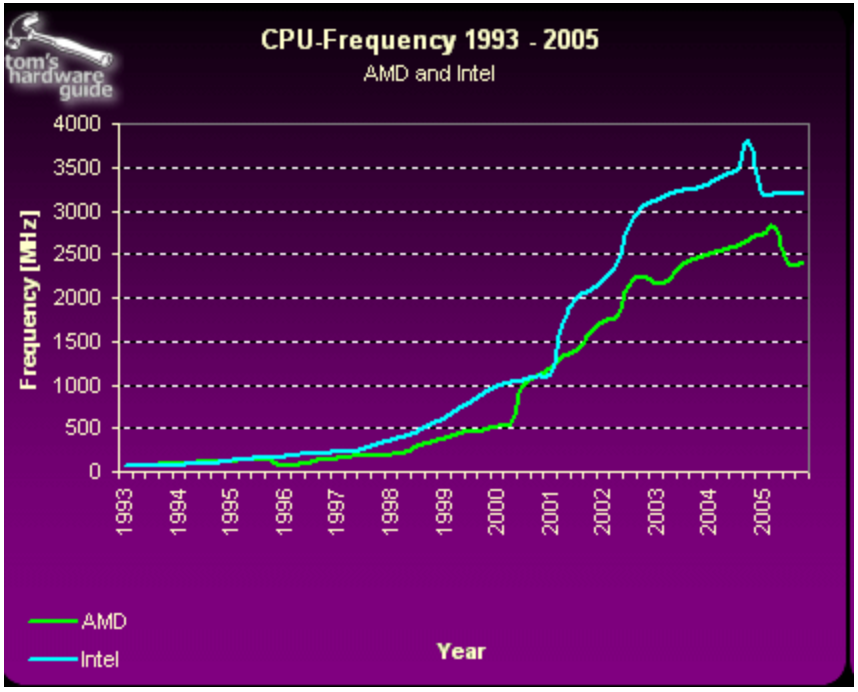
- **Why not simply build faster serial computer??**
  - The speed at which data can move through hardware determines the speed of a serial computer. So we need to increase proximity of data.
  - Limits to miniaturization: even though
  - Economic Limitations: cheaper to use multiple commodity processors to achieve fast performance than to build a single fast processor.

Why Parallel Computing? (2/2)

- Saving time
- Solving large complex problems
- Take advantage of concurrency

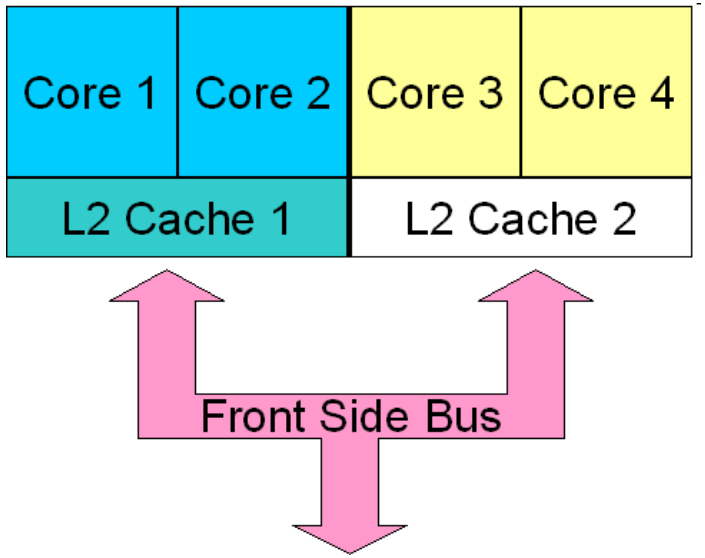
How much faster can CPUs get?

- Towards the early 2000's we hit a "frequency wall" for processors.
- Processor frequencies topped out at 4 GHz due to the thermal limit of Silicon.



Multicore CPUs

- Have fully functioning CPU "cores" in a processor.
- Have individual L1 and shared L2 caches.
- OS and applications see each core as an individual processor.
- Applications have to be specifically rewritten for performance.



Source: hardwaresecrets.com

Graphics Processing Units (GPUs)

- Massively Parallel Computational engines for Graphics.
- More ALUs, less cache – Crunches lots of numbers in parallel but can't move data very fast.
- Is an "accelerator", cannot work without a CPU.
- Custom Programming Model and API.

© NVIDIA

© NVIDIA

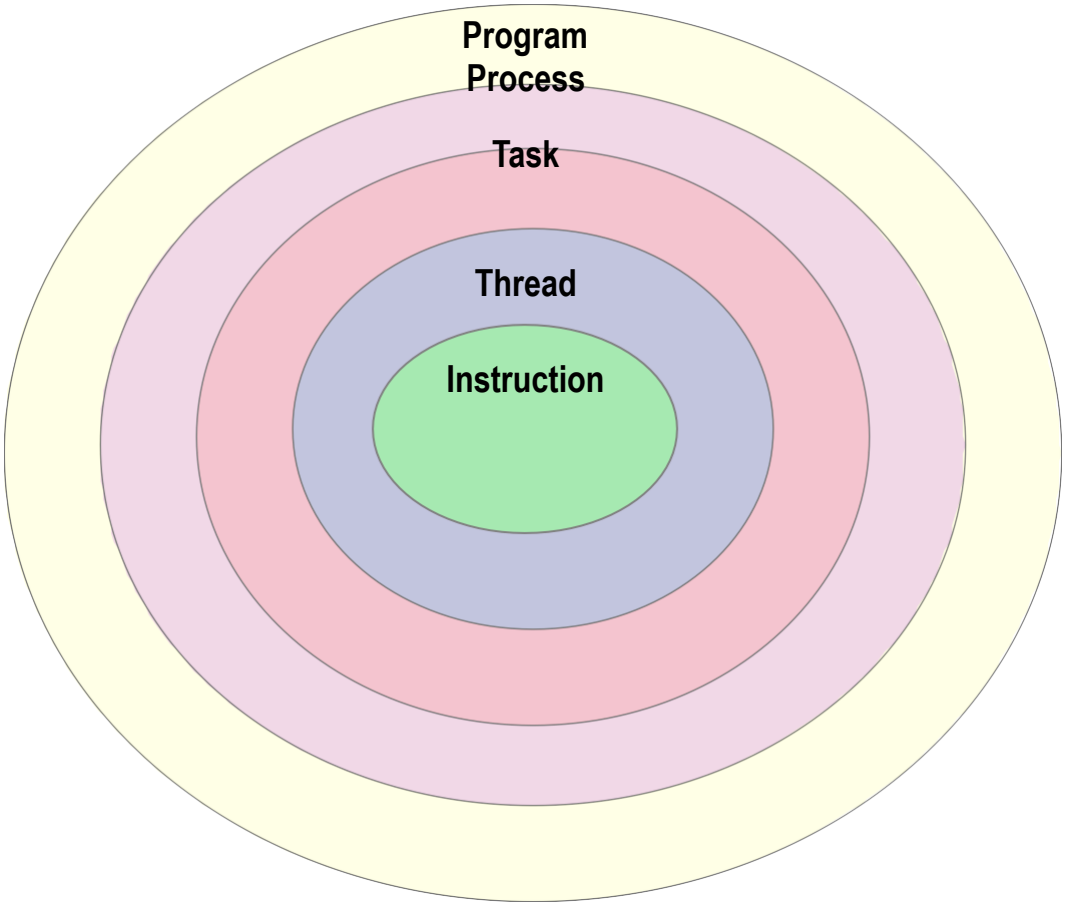
Lecture Outline

- What is Parallel Computing?
- Motivation of Parallel Computing
- **Parallelization Levels**
- **Parallel Computers Classification**
- **Parallel Computing Memory Architecture**
- **Parallel Programming Models**

Terms

- **Program**: an executable file with one or multiple tasks.
- **Process**: instance of a program in execution. It has its own address space, and interacts with other processes only through communication mechanisms managed by the OS.
- **Task**: execution path through the address space that has many instructions. (Some times task and process are used interchangeably).
- **Thread**: stream of execution used to run a task. It's a coding construct that does not affect the architecture of an application. A process might contain one or multiple threads all sharing the same address space and interacting directly.
- **Instruction**: a single operation of a processor. A thread has one or multiple instructions.

Program Execution Levels

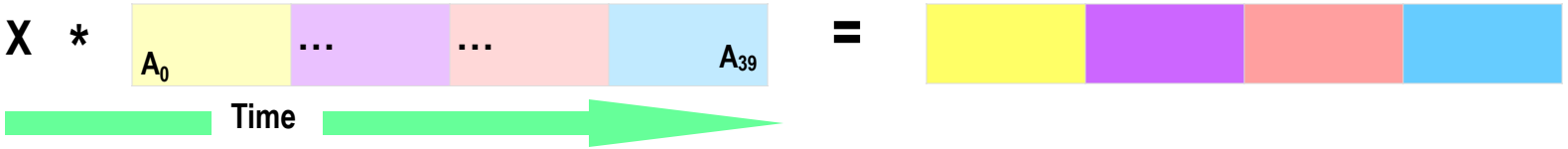


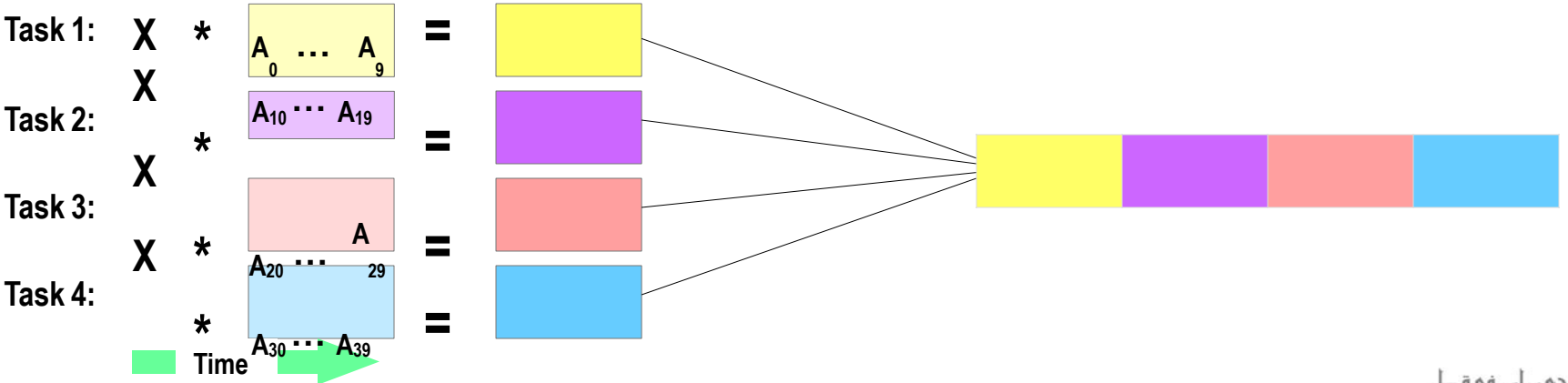
Parallelization levels (1/8)

- Data-level parallelism
- Task-level parallelism
- Instruction-level parallelism
- Bit-level parallelism

Parallelization levels (2/8)

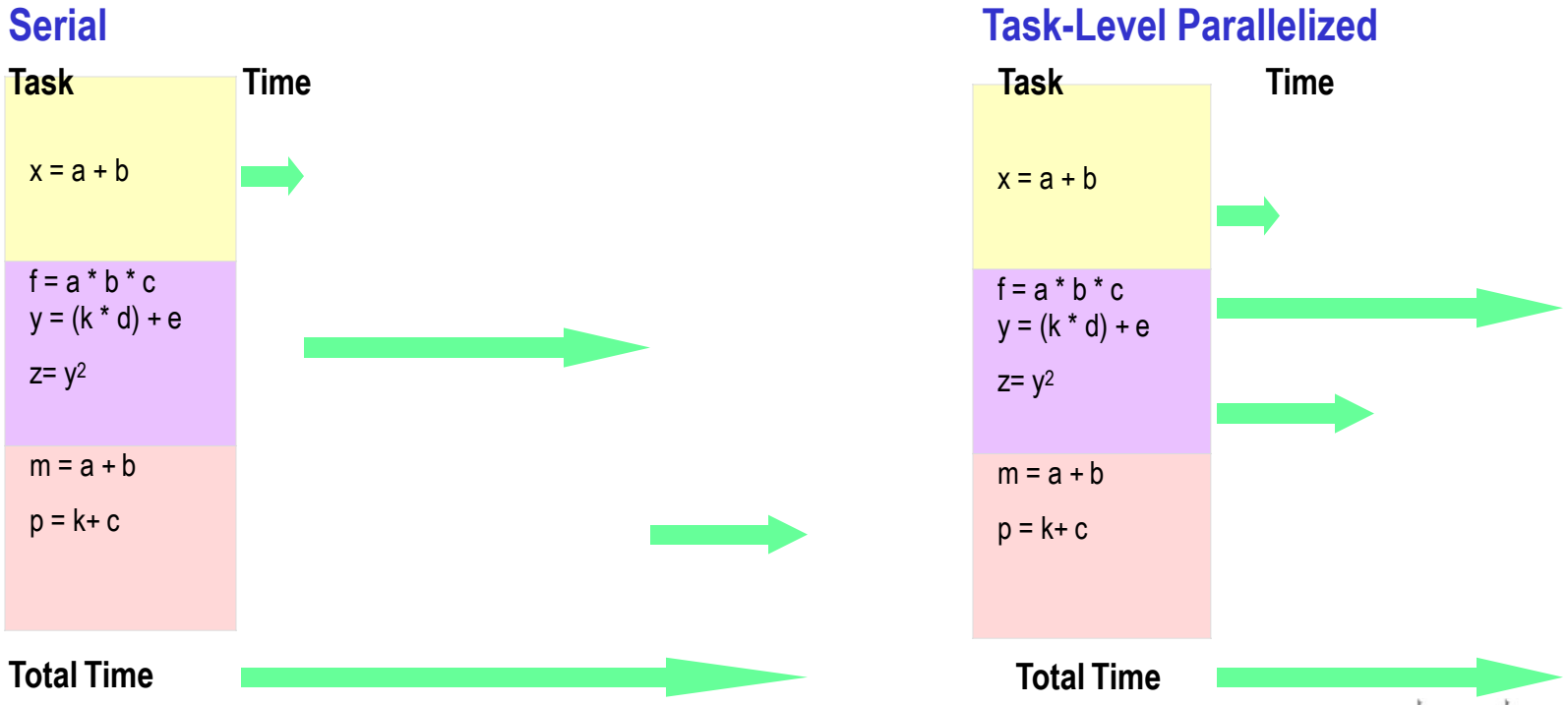
- **Data-level Parallelism**
  - Associated with loops.
  - Same operation is being performed on different partitions of the same data structure.
  - Each processor performs the task of its part of the data.
    - Ex: add x to all the elements of an array
  - But, can all loops can be parallelized?
    - Loop carried dependencies: if each iteration of the loop depends on results from the previous one, loop can not be parallelized.





Parallelization levels (3/8)

- **Task-level Parallelism**
  - Completely different operations are performed on the same or different sets of data.
  - Each processor is given a different task.
  - As each processor works in its own task, it communicates with other processors to pass and get data.



Parallelization levels (4/8)

- **Instruction-level parallelism (ILP)**
  - Reordering the instructions and combining them into groups so the computer can execute them simultaneously.
  - Example:

```
x = a + b
y = c + d
z = x + y
```

Serial

- If each instruction takes 1 time unit, sequential performance requires 3 time units.

Instruction-level Parallelized

- The first two instructions are independent, so they can performed simultaneously.
- The third instruction depends on them so it must be performed afterwards.
- Totally, this will take 2 time units.

Parallelization levels (5/8)

- **Micro-architectural techniques that make use of ILP include**

Instructions pipelining:

- Increasing the instruction throughput by splitting the instruction into different stages.
- Each pipeline stage corresponds to a different action the processor performs on that instruction in that stage.
- Machine with N pipelines can have up to N different instructions at different stages of completion.
- Processing rate of each step takes less time than processing the whole instruction (all stages) at once. A part of each instruction can be done before the next clock signal. This reduces the delay of waiting for the signal.

Instruction #	Pipeline Stage						
1	IF	ID	EX	MEM	WB		
2		IF	ID	EX	MEM	WB	
3			IF	ID	EX	MEM	WB
4				IF	ID	EX	MEM
5					IF	ID	EX
Clock Cycle	1	2	3	4	5	6	7

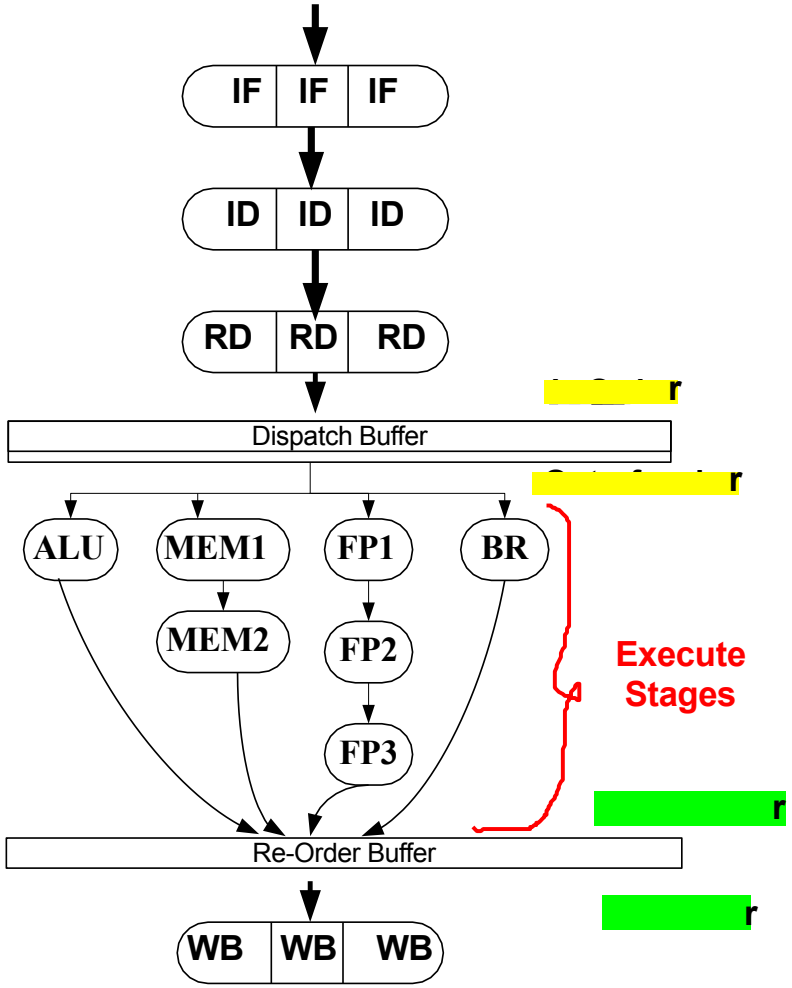
IF: Instruction Fetch
ID: Instruction Decode
EX: Execute
MEM: Memory Access
WB: Write Back to Reister

Parallelization levels (6/8)

- Instruction-level parallelism (ILP)

Order of superscalars

- Superscalar of a core is divided at levels:
- A level-x core is the one with x functional units.
- The more functional units:
  - the more parallelism the system can do &
  - the less is the time spent on execution.



Parallelization levels (7/8)

- **Micro-architectural techniques that make use of ILP include**

Superscalar processors:

  - Implements ILP within a single processor so that a processor can execute more than one instruction at a time (per clock cycle).
- It does that by sending multiple instructions to redundant functional units on the processor simultaneously. These functional unit are not separate processors! They are execution resources (ALU, multiplier, …) within a single processor. This enables the processor to accept multiple instructions per clock cycle.

2 instructions are fetched and sent at a time, so 2 instructions can be completed per clock cycle!

Instruction #	Pipeline Stage						
1	IF	ID	EX	MEM	WB		
2	IF	ID	EX	MEM	WB		
3		IF	ID	EX	MEM	WB	
4		IF	ID	EX	MEM	WB	
5			IF	ID	EX	MEM	WB
6			IF	ID	EX	MEM	WB
7				IF	ID	EX	MEM
8				IF	ID	EX	MEM
9					IF	ID	EX
10					IF	ID	EX
Clock Cycle	1	2	3	4	5	6	7

Parallelization levels (8/8)

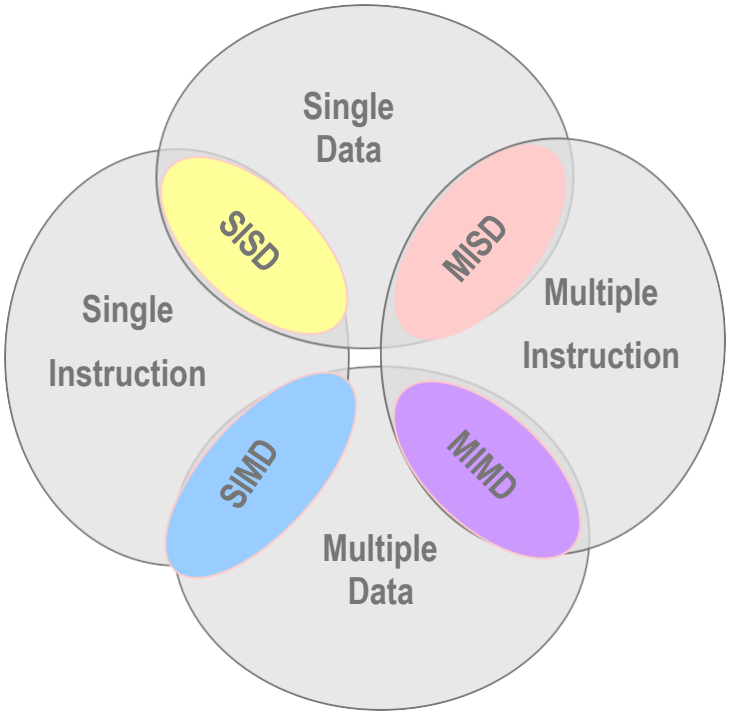
- **Bit-level parallelism**
  - Based on increasing the word size of the processor.
  - Reduces the number of instructions required to perform an operation on variables whose sizes are bigger than the processor word size.
  - Example:
    - Adding 2 16-bit integers with an 8-bit processor requires 2 instructions
    - Adding them with a 16-bit processor required one instruction

Lecture Outline

- What is Parallel Computing?
- Motivation of Parallel Computing
- Parallelization Levels
- **Parallel Computers Classification**
- **Parallel Computing Memory Architecture**
- **Parallel Programming Models**

Parallel Computers Classification (1/5)

- **Flynn's Classical Taxonomy**
  - Differentiate between the architectures of multi-processor computers based on two independent aspects: **Instructions** streaming to a processor and **Data** the processor uses.

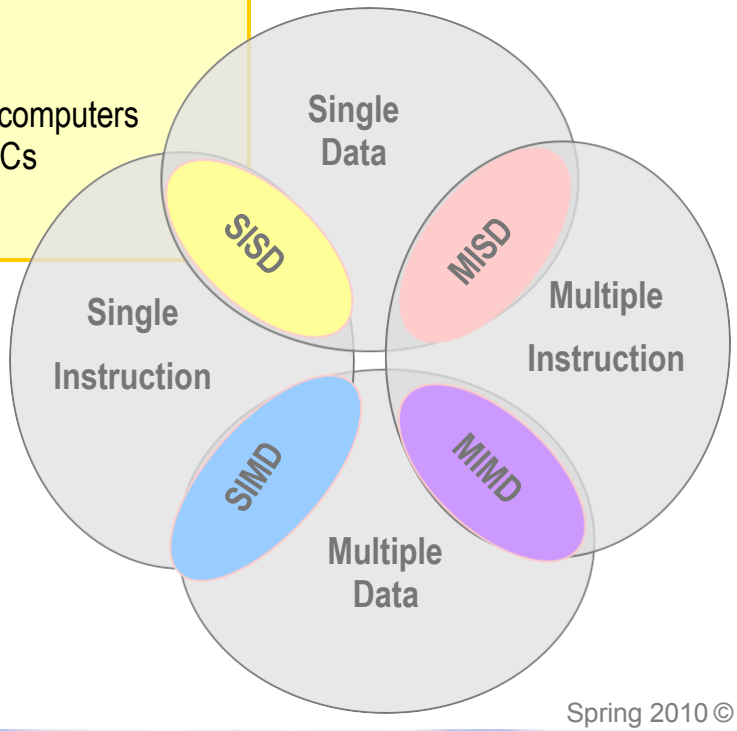


Parallel Computers Classification (2/5)

- **Flynn's Classical Taxonomy**
  - Differentiate between the architectures of multi-processor computers based on two independent aspects: **Instructions** streaming to a processor and **Data** the processor uses.

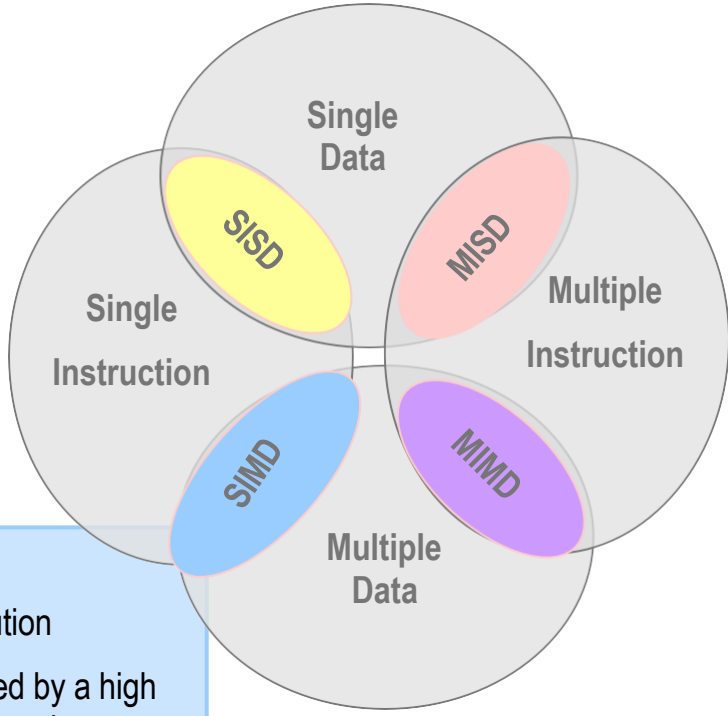
SISD

- Serial (non-parallel) computer
- Deterministic execution
- Older generation mainframes, minicomputers & workstations, most modern day PCs



Parallel Computers Classification (3/5)

- **Flynn's Classical Taxonomy**
  - Differentiate between the architectures of multi-processor computers based on two independent aspects: **Instructions** streaming to a processor and **Data** the processor uses.



SIMD

- A type of parallel computer
- Synchronous (lockstep) and deterministic execution
- Best suited for specialized problems characterized by a high degree of regularity, such as graphics/image processing.
- Most modern computers, particularly those with graphics processor units (GPUs)

vn an abstract model of SIMD computers having a sing
operational model of SIMD computers is presented belo
Implementation models and case studies of SIMD machi

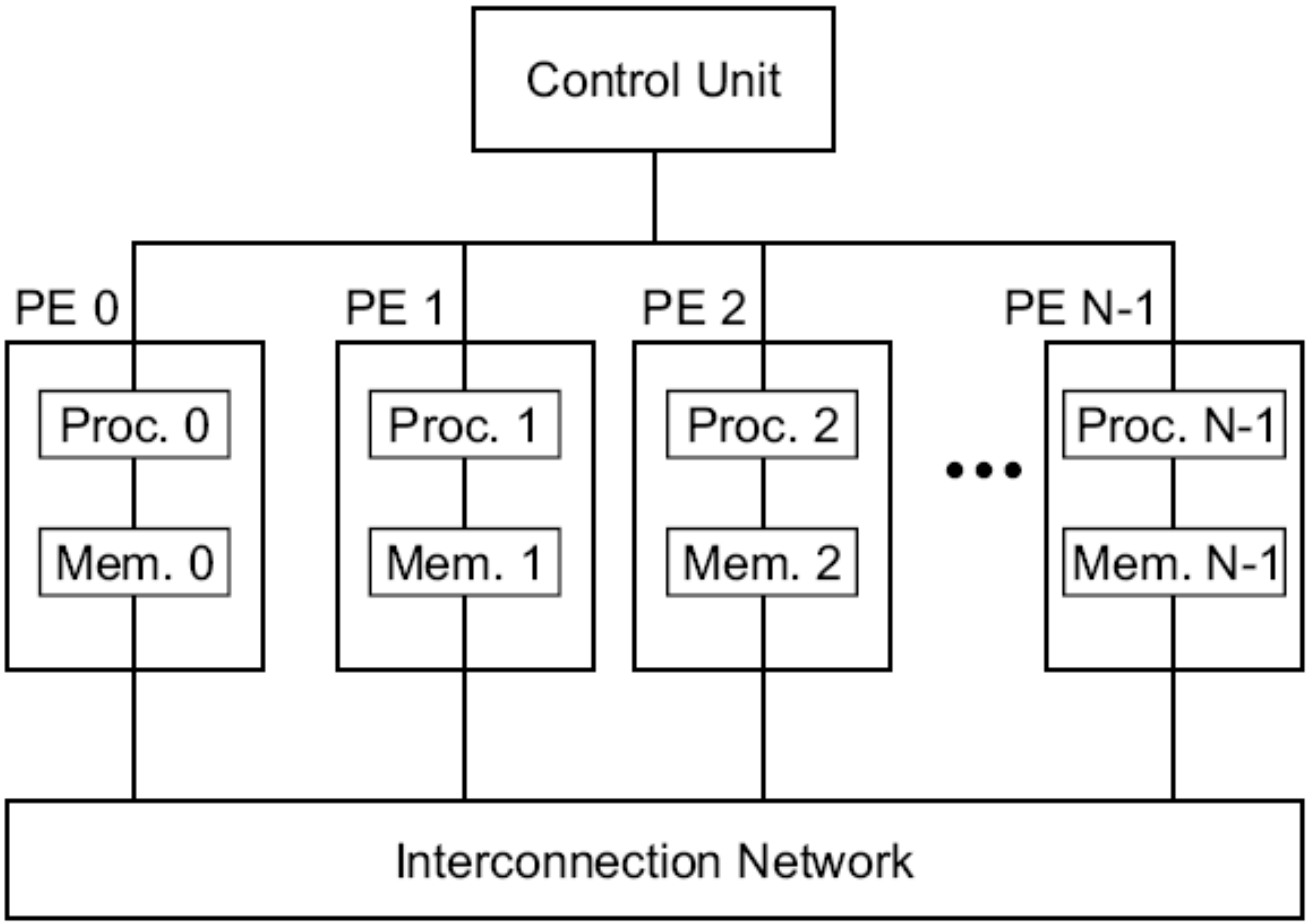


Fig. 1.12 Operational model of SIMD computers

Mc Graw Hill Education

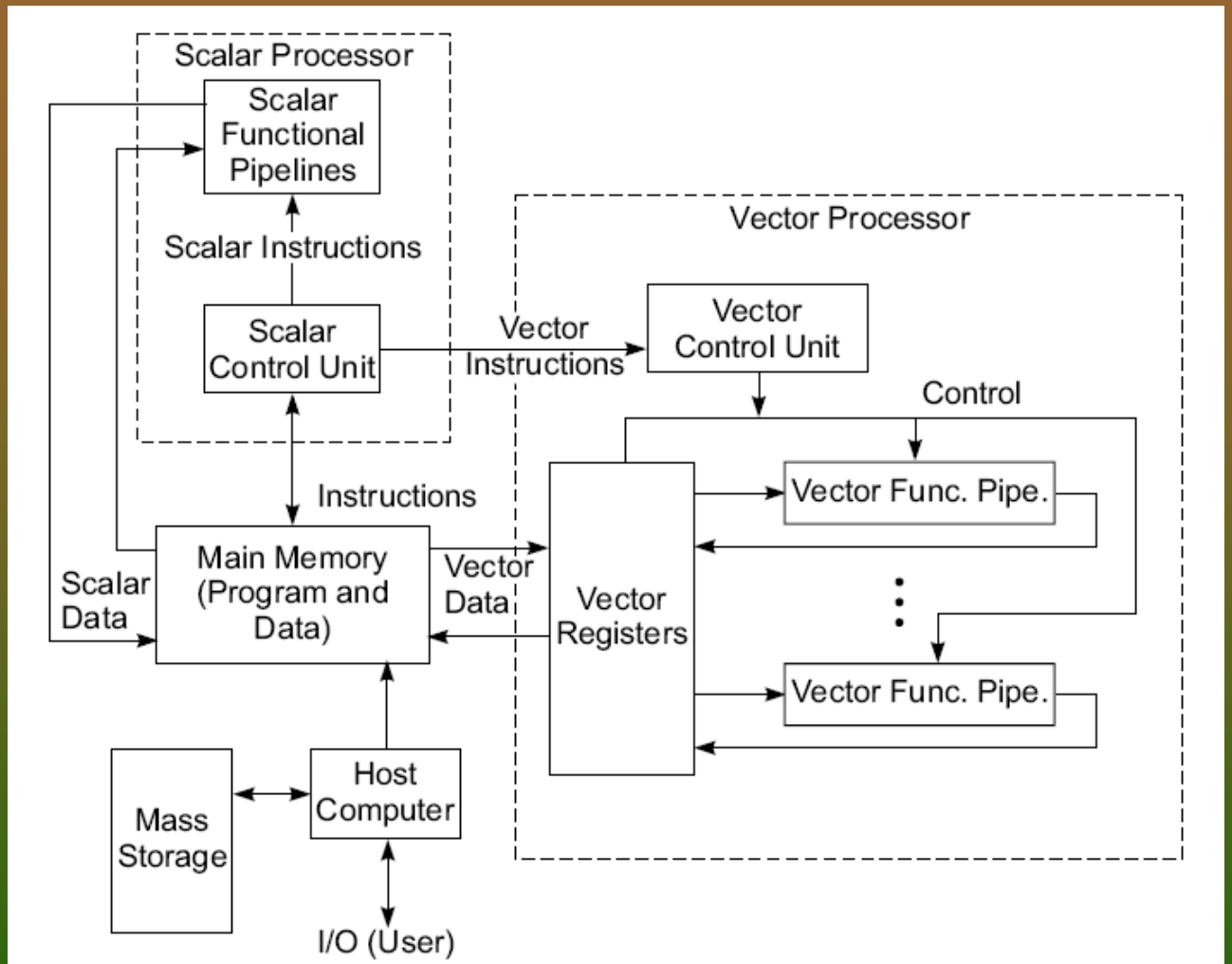


Fig. 1.11 The architecture of a vector supercomputer

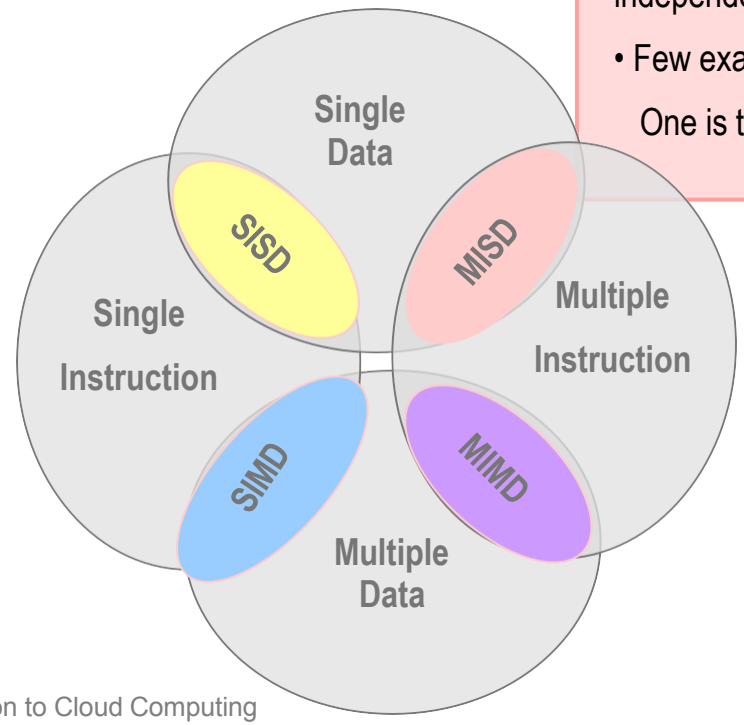
Parallel Computers Classification (4/5)

- **Flynn's Classical Taxonomy**
  - Differentiate between multi-processor computer architectures based on two independent dimensions: **Instructions** streaming to the processor and **Data** the processor uses.

MISD

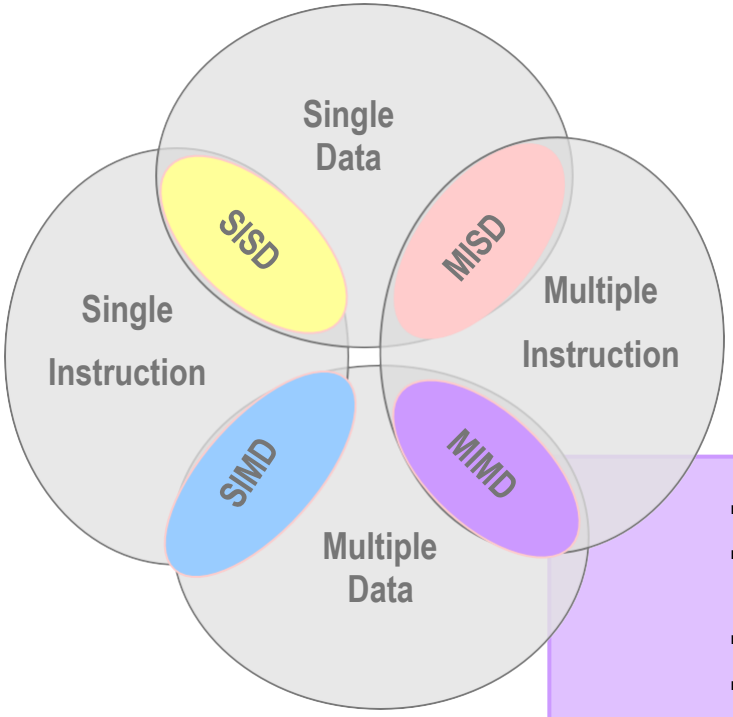
- Single data stream is fed into multiple processing units.
- Each processing unit operates on the data independently using independent streams of instructions.
- Few examples exist:

  One is the experimental Carnegie-Mellon C.mmp computer (1971).



Parallel Computers Classification (5/5)

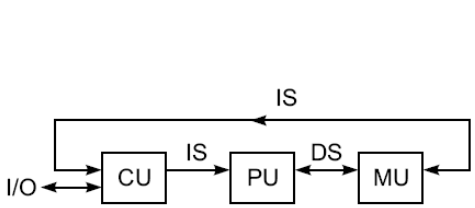
- **Flynn's Classical Taxonomy**
  - Differentiate between the architectures of multi-processor computers based on two independent aspects: **Instructions** streaming to a processor and **Data** the processor uses.



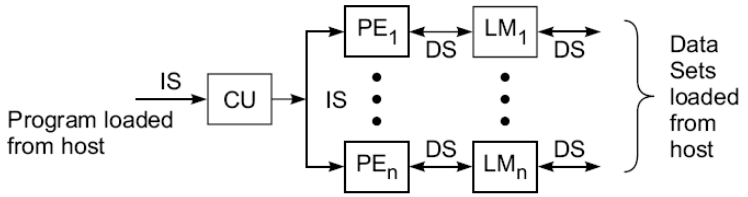
MIMD

- The most common type of parallel computing.
- Execution can by synchronous or asynchronous, deterministic or non-deterministic
- Many MIMD architecture include SIMD execution sub-components
- Examples: Most current supercomputers, networked parallel computer clusters and grids, multi-processor SMP computers, multi core PCs

Mc Graw Hill Education



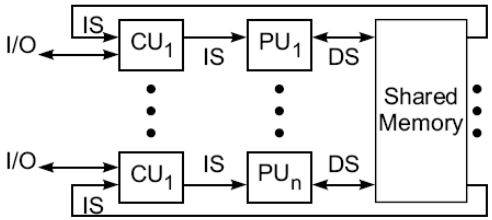
(a) SISD uniprocessor architecture



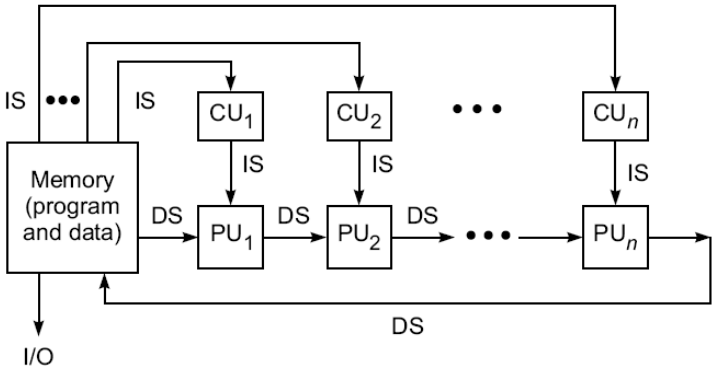
(b) SIMD architecture (with distributed memory)

Captions:

CU = Control Unit
PU = Processing Unit
MU = Memory Unit
IS = Instruction Stream
DS = Data Stream
PE = Processing Element
LM = Local Memory



(c) MIMD architecture (with shared memory)



(d) MISD architecture (the systolic array)

Fig. 1.3 Flynn's classification of computer architectures (Derived from Michael Flynn, 1972)

Lecture Outline

- What is Parallel Computing?
- Motivation of Parallel Computing
- Parallelization Levels
- Parallel Computers Classification
- **Parallel Computing Memory Architecture**
- **Parallel Programming Models**

Parallel Computing Memory Architecture (1/9)

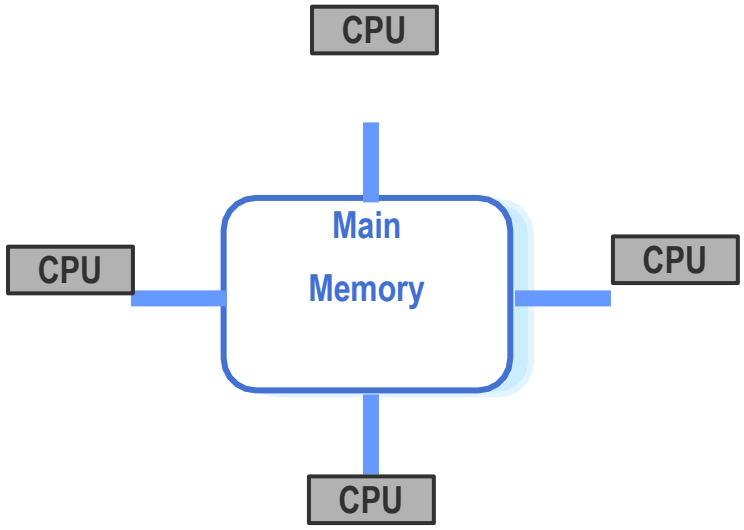
- Shared Memory Architecture
- Distributed Memory Architecture
- Hybrid Distributed-Shared Memory Architecture

Parallel Computing Memory Architecture (2/9)

- **Shared Memory Architecture**
  - **Processors perform their operations independently but they share the same resources since they have access to all the memory as global address space.**
  - **When a processor changes the value of a location in the memory, the effect is observed by all other processors.**
  - **Based on memory access times: Shared memory machines can be divided into two main classes UMA and NUMA.**

Parallel Computing Memory Architecture (3/9)

- **Shared Memory Architecture**
  - **Uniform Memory Access:**
    - **Equal access rights and access times to memory.**
    - **Represented by Symmetric Multiprocessor (SMP).**
    - **Sometimes called Cache Coherent UMA (CC-UMA).**



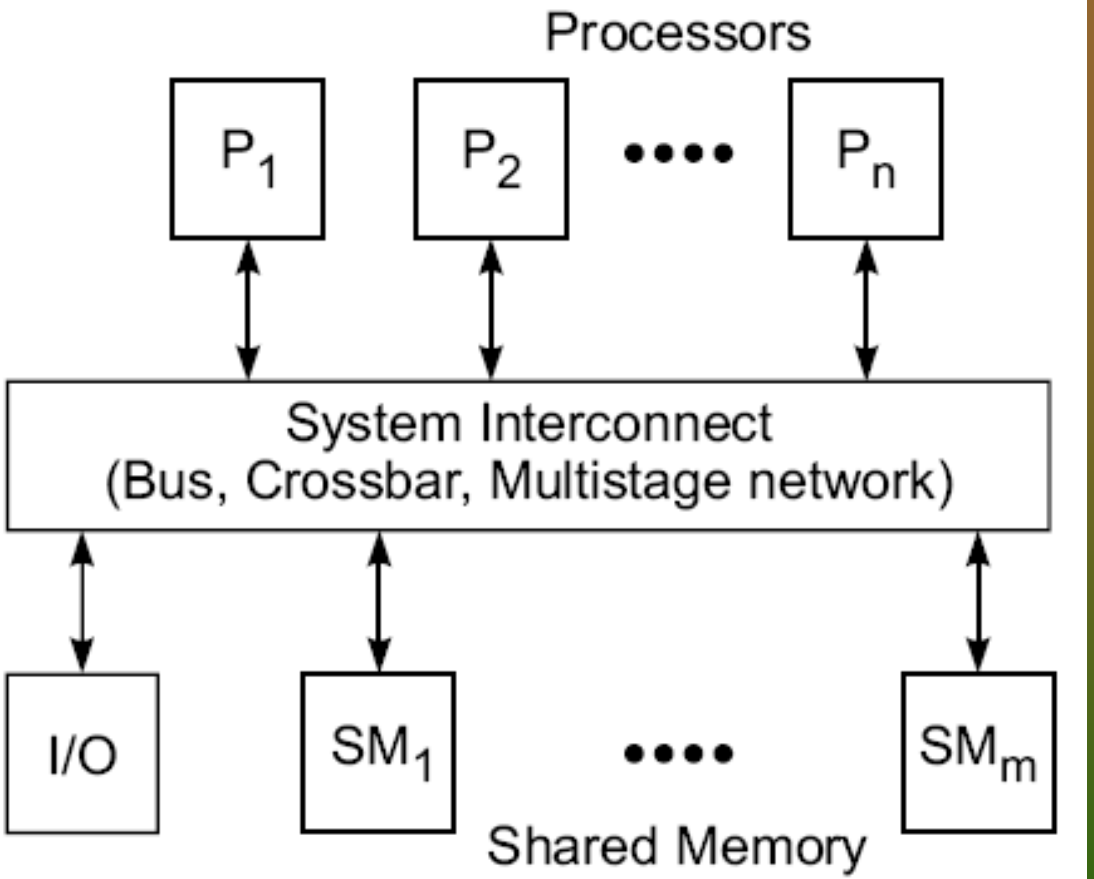
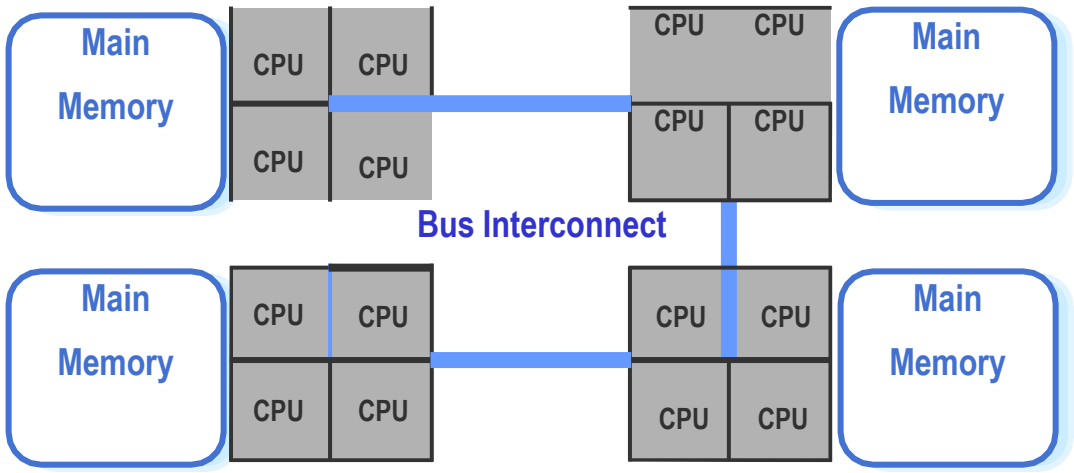


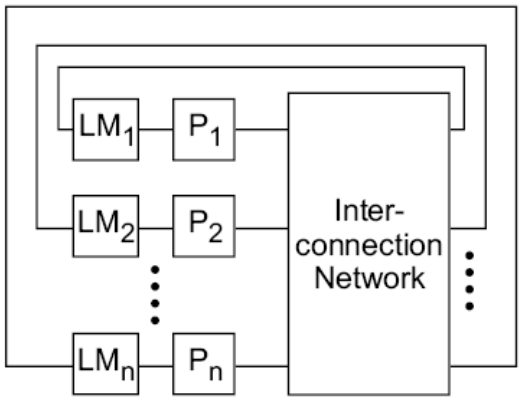
Fig. 1.6 The UMA multiprocessor model

Parallel Computing Memory Architecture (4/9)

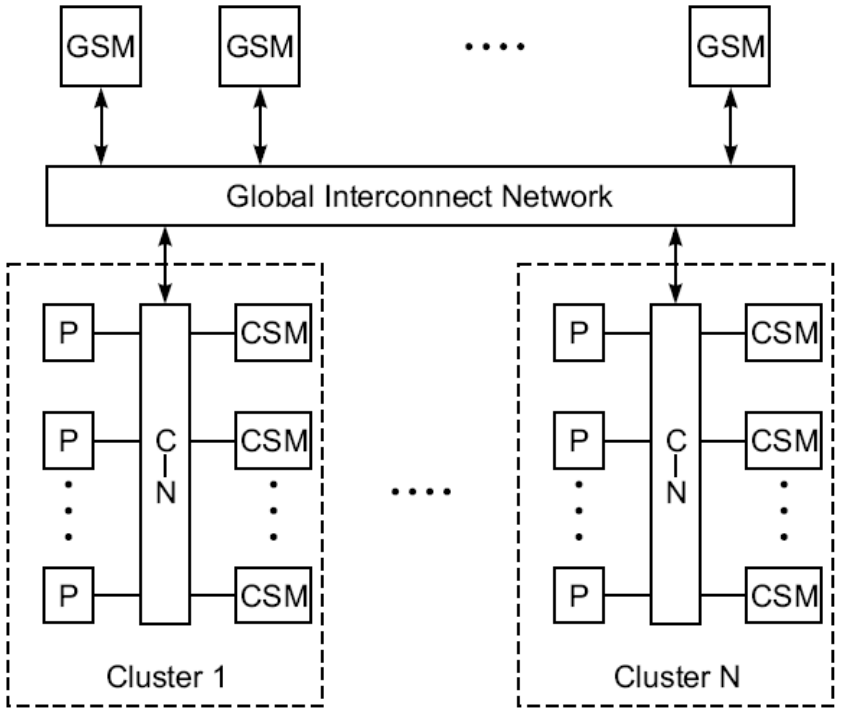
- **Shared Memory Architecture**
  - **Non-Uniform Memory Access:**
    - Usually delivered by physically linking two or more SMPs so they can access the memory of each other directly.
    - Not all processors have equal access time to all memories.
    - Memory access across the physical link is slow.
    - Called Cache Coherent NUMA (CC-NUMA) if cache coherency is maintained.



Mc Graw Hill Education

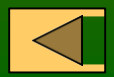


(a) Shared local memories (e.g. the N Butterfly)



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

Fig. 1.7 Two NUMA models for multiprocessor systems



Parallel Computing Memory Architecture (5/9)

Shared Memory Architecture

- **Advantages**
  - Program development can be simplified. Global address space makes referencing data stored in memory similar to traditional single-processor programs.
  - Memory is proximate to CPUs which makes data sharing between tasks fast and unifor m.

- **Disadvantages**
  - Lack of scalability between memory and CPUs.
    - Adding more CPUs can increases on the path between shared memory and CPUs.
  - Maintaining data integrity is complex.
    - Synchronization is required to ensure correct access of memory.

Mc Graw Hill Education

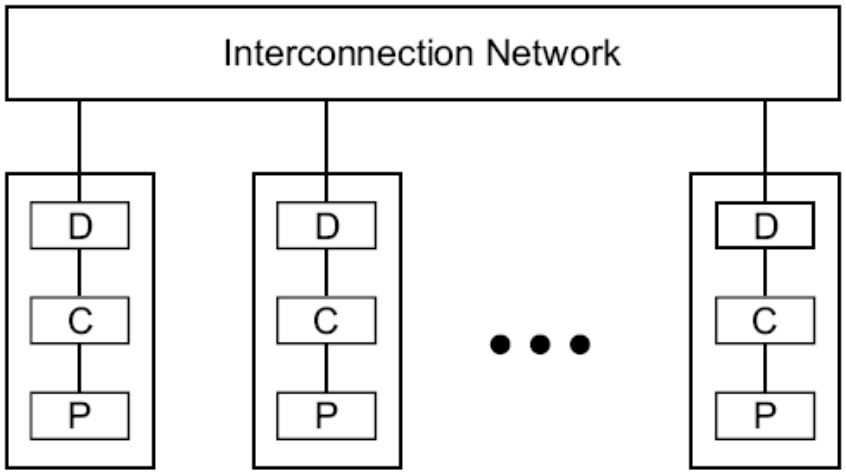
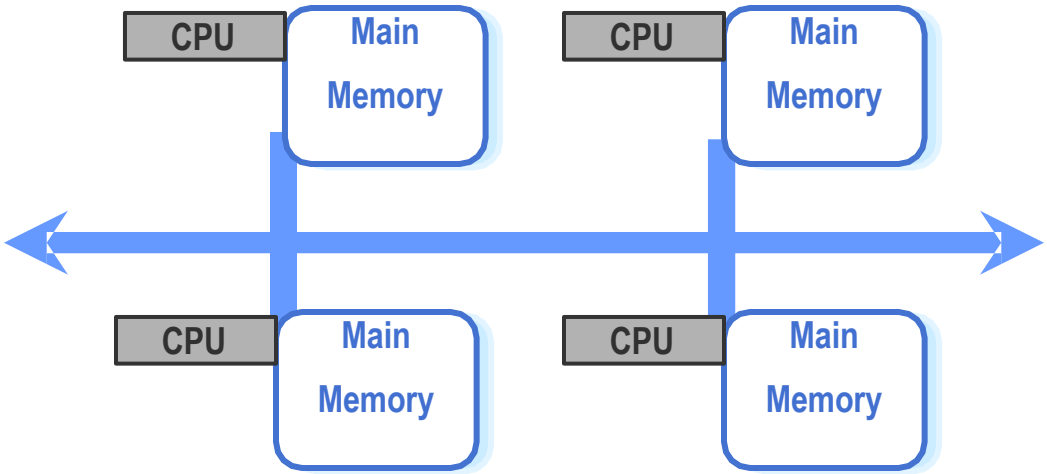


Fig. 1.8 The COMA model of a multiprocessor (P: Processor, C: Cache, D: Directory; e.g. the KSR-1)

Parallel Computing Memory Architecture (6/9)

- **Distributed Memory Architecture**
  - Each processor has its own address space (local memory). Changes done by each processor is not visible by others.
  - Processors are connected to each other over a communication network.
  - The program must define a way to transfer data when it is required between processors.



Parallel Computing Memory Architecture (7/9)

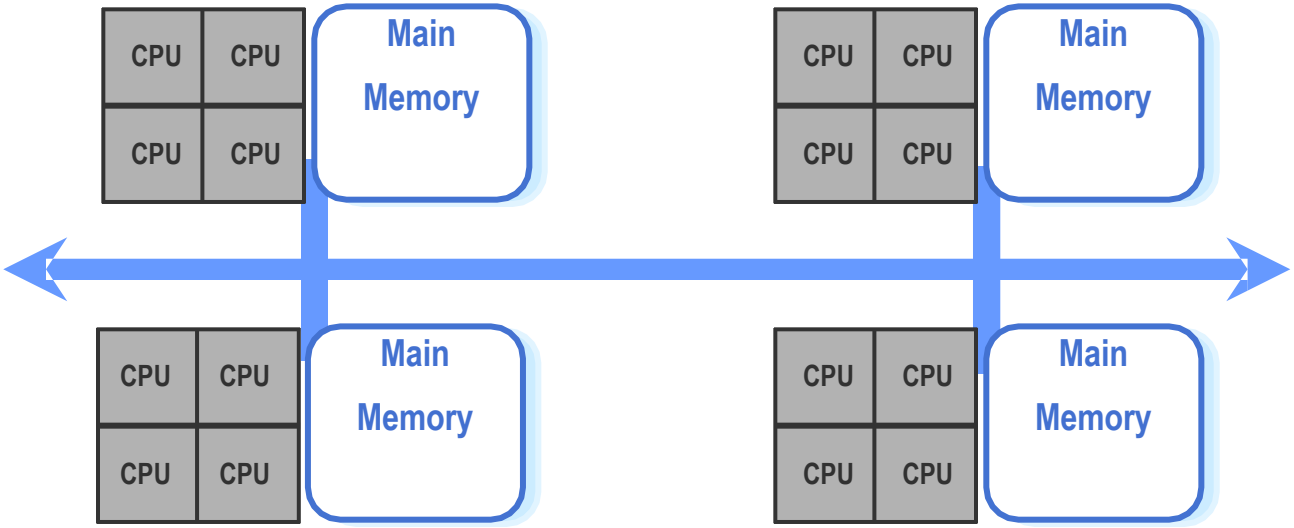
Distributed Memory Architecture

- **Advantages**
  - Scalability in memory size is along number of processors.
  - Each processor has fast access to its own memory with no interference or overhead.
  - Cost effectiveness: uses multiple low cost processors and networking.

- **Disadvantages**
  - Data communication between processors is the programmer's responsibility.
  - It's difficult to map existing data structures, based on global memory, to such organization.
  - Challenge: How to distribute a task over multiple processors (each with its own memory), and then reassemble the results from each processors into one solution?

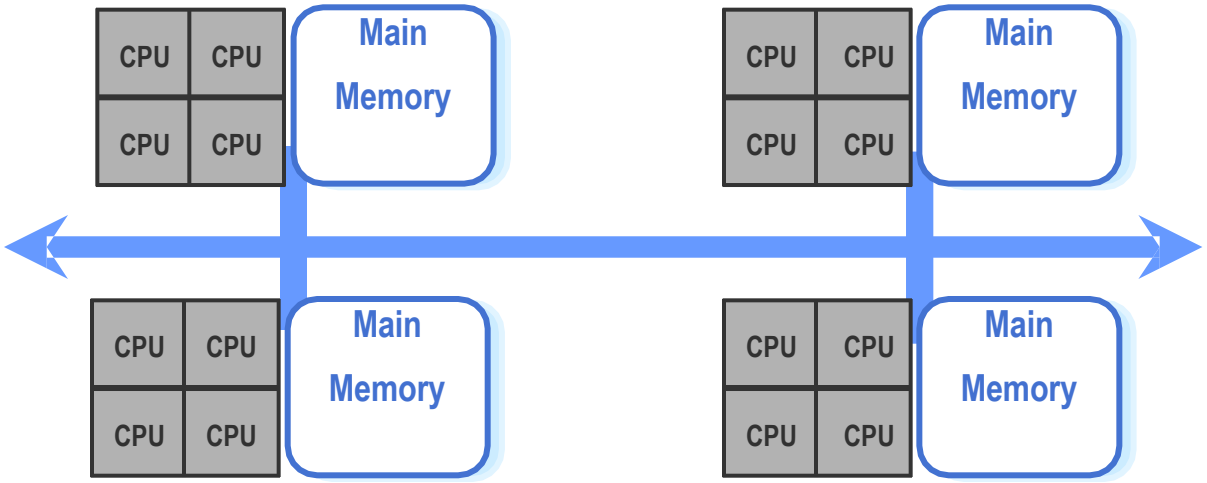
Parallel Computing Memory Architecture (8/9)

- **Hybrid Distributed-Shared Memory Architecture**
  - Used in most of today's fast and large parallel computers
  - The shared memory components are SMP nodes.
  - The distributed memory component is a network of SMP nodes.
  - Advantages and disadvantages are the common points between the two architectures.



Parallel Computing Memory Architecture (9/9)

- **Hybrid Distributed-Shared Memory Architecture**
  - Processors on an SMP machine address the machine's memory as global.
  - Each SMP knows only about the SMP's own memory, but not the memory on another SMP.
  - Therefore, network communications are required to move data from one SMP to another.



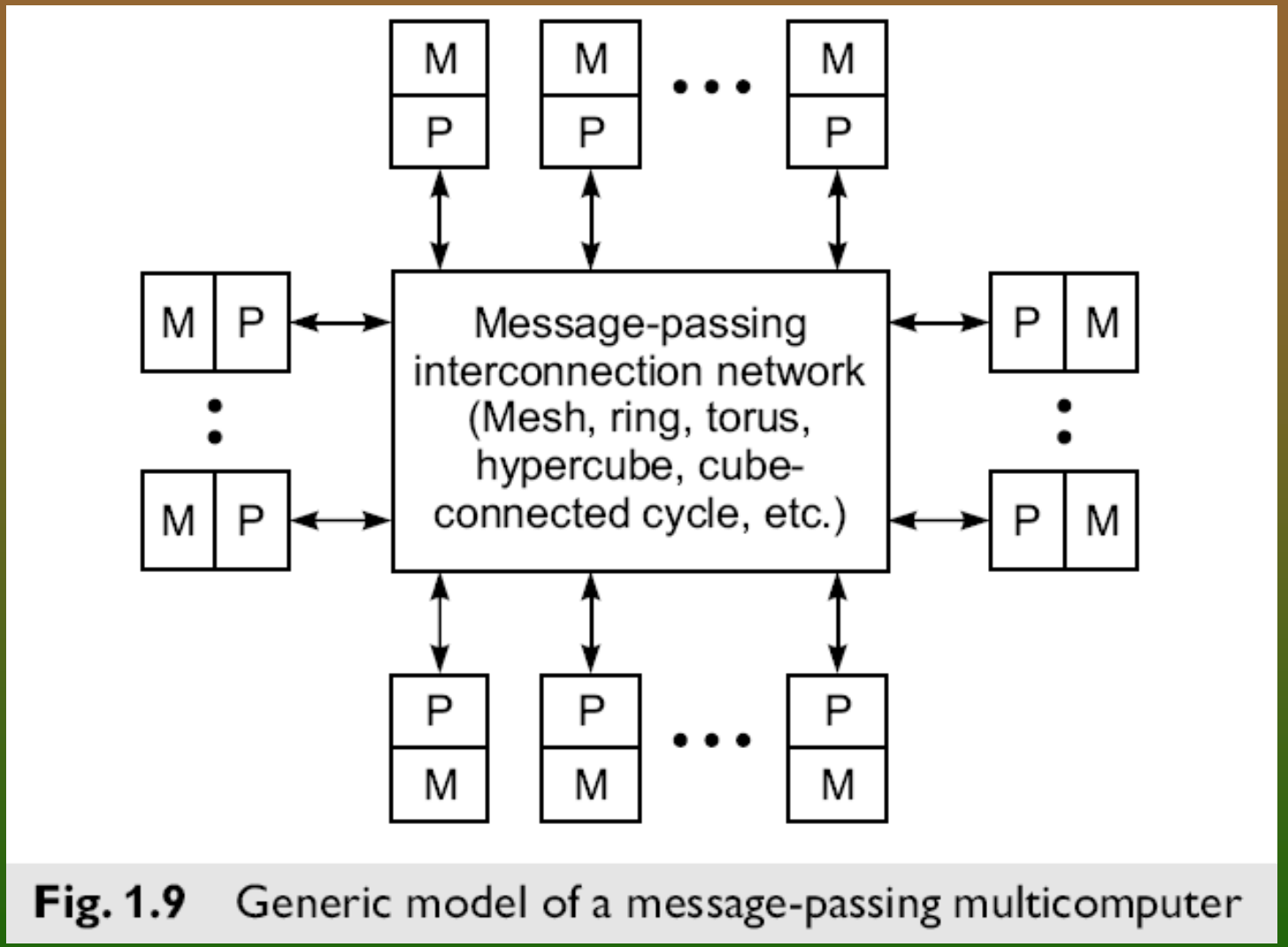


Fig. 1.9 Generic model of a message-passing multicomputer

Lecture Outline

- What is Parallel Computing?
- Motivation of Parallel Computing
- Parallelization Levels
- Parallel Computers Classification
- Parallel Computing Memory Architecture
- **Parallel Programming Models**

Parallel Programming Models (1/12)

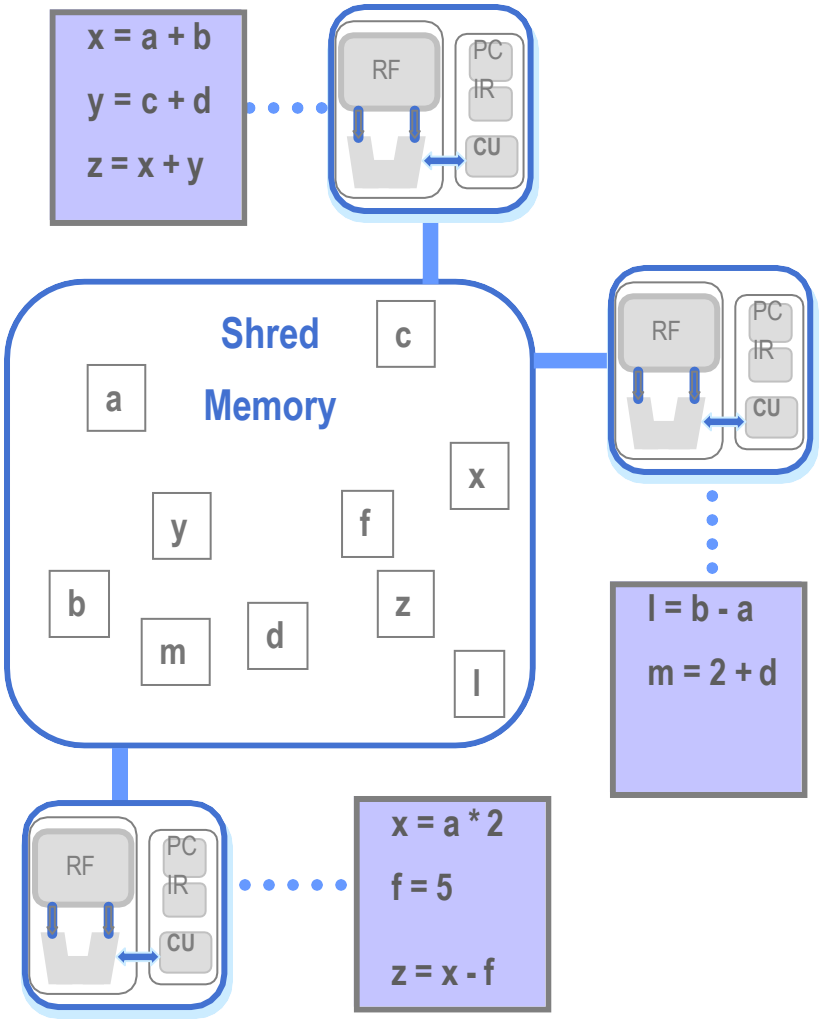
- **What is Programming Model?**
  - A programming model presents an abstraction of the computer system.
  - A programming model enables the expression of ideas in a specific way. A programming language is then used to put these ideas in practice.
- **Parallel Programming Model:**
  - Software technologies that are used as an abstraction above hardware and memory architectures to express parallel algorithms or to match algorithms with parallel systems.

Parallel Programming Models (2/12)

- Shared Memory Model
- Threads Model
- Data Parallel Model
- Message Passing Model
- Others…
  - SPMD
  - MPMD

Parallel Programming Models (3/12)

- **Shared Memory Model**
  - Processors read from and write to variables stored in a shared address space asynchronously.
  - Mechanisms such as locks and semaphores are used to control access to the shared memory.

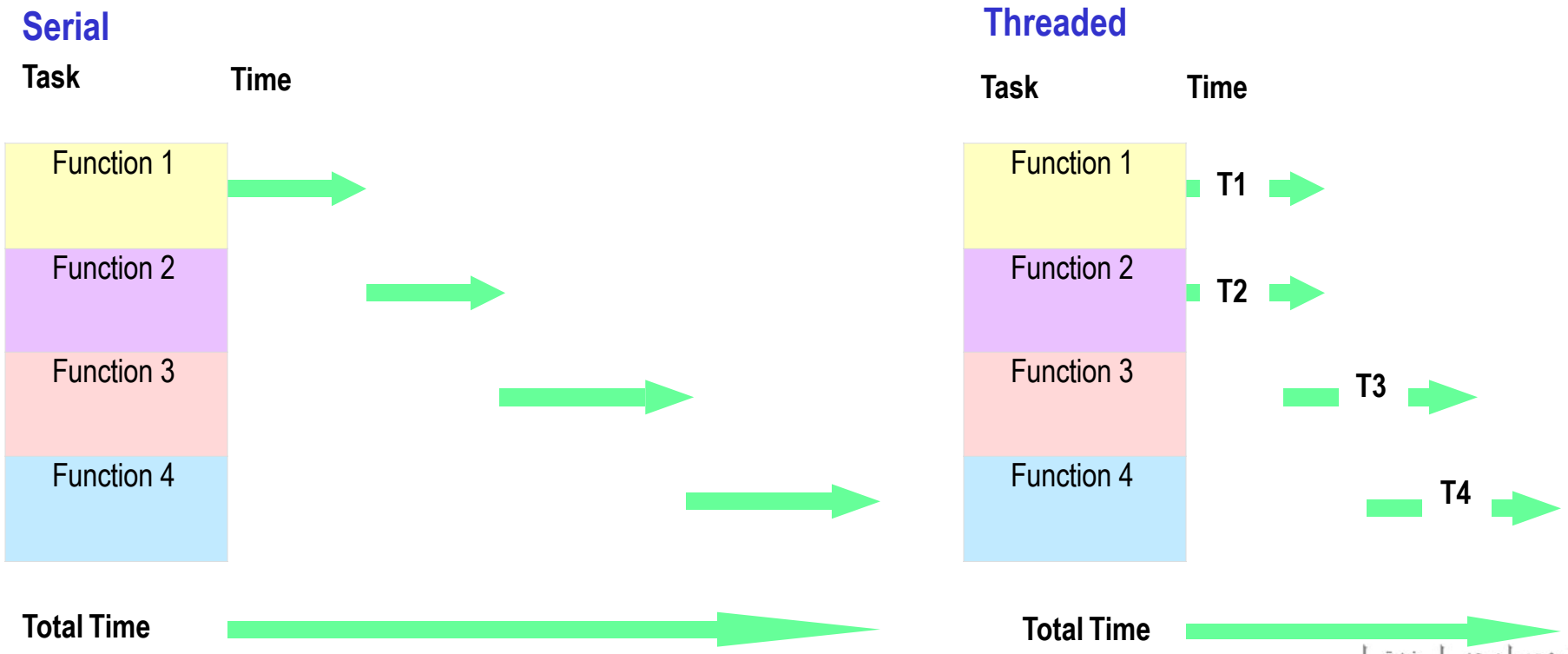


Parallel Programming Models (4/12)

- **Shared Memory Model**
  - **Advantage:** Program development can be simplified since no process owns the data stored in memory.
    - Because all processors can access the same variables, referencing data stored in memory is similar to traditional single-processor programs.
  - **Disadvantage:** it's difficult to understand and manage data locality.
    - Keeping data local to the processor that is working on it conserves memory access to this processor. This causes bus traffic when multiple processors are trying to access the same data.

Parallel Programming Models (5/12)

- **Threads Model**
  - A single process is divided into multiple, concurrent execution paths, each path is a thread.
  - Process execution time is reduced because threads are distributed and executed on different processors concurrently.



Parallel Programming Models (6/12)

- **Threads Model: How does it work?**
  - Each thread has local data that is specific to this thread and not replicated on other threads.
  - But also, all threads share and communicate through the global memory.
  - Results of each threads execution can then be combined to form the result of the main process.
  - Threads are associated with shared memory architectures.
  - Implementations: *POSIX Threads* and *OpenMP*

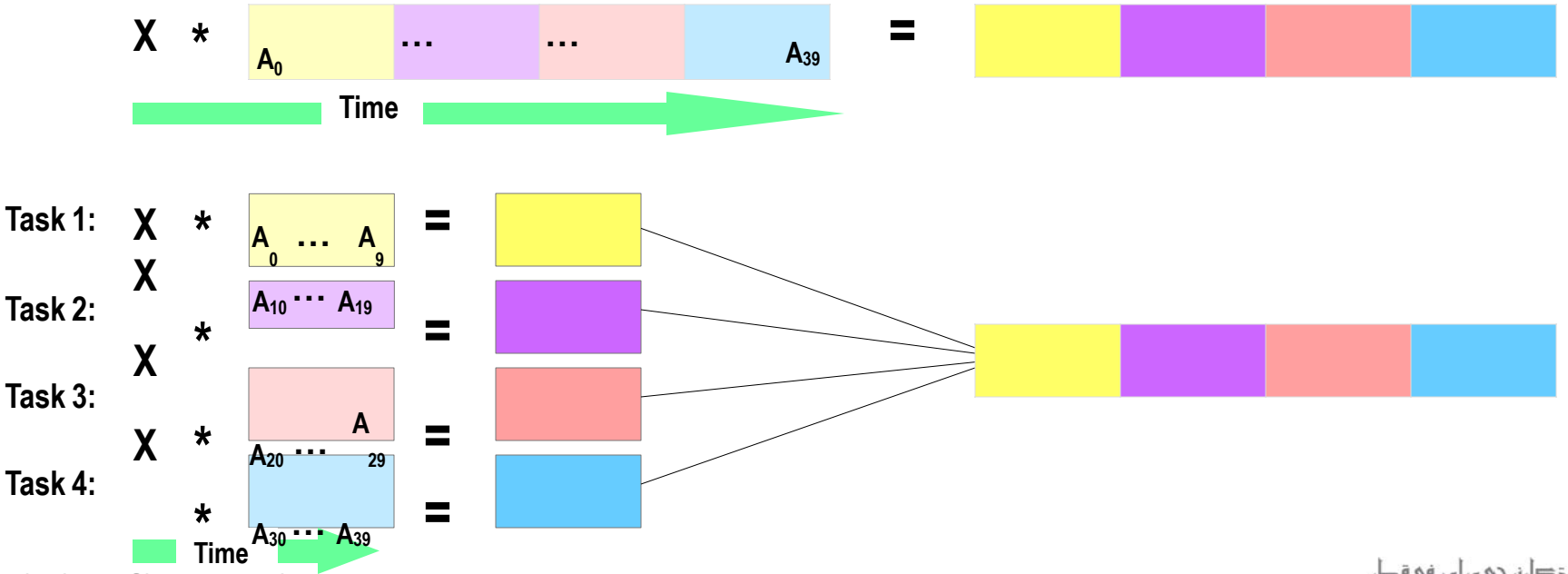
Parallel Programming Models (7/12)

- **Threads Model: Challenges?**
  - An issue is load balancing between threads.
  - What happens when a thread finishes with a region of code? Wait for other threads? Help other threads with the load?
  - **Solution:** Different scheduling algorithms (for example in loop context):
    - Static Scheduling: each threads execute *n* iterations and then wait for the other threads to finish *n* iterations.
    - Dynamic scheduling: *n* iterations of the remaining iterations are dynamically assigned to threads that are idle.
    - Guided scheduling: each time a thread finishes executing it is assigned some iterations:

$$\#\ iterations\ assigned \cong \frac{\#\ \text{remaining iterations}}{\#\ threads}$$

Parallel Programming Models (8/12)

- **Data Parallel Model**
  - A data set (ex: an array) is divided into chunks, operations are performed on each chunk concurrently.
  - A set of tasks are carried out on the same data structure. But, each task is performed on a different part of this data structure.
  - Tasks that are carried out on the same part of the data structure do the same operations on each instance of this data. (ex: multiply each element of the array by 2).

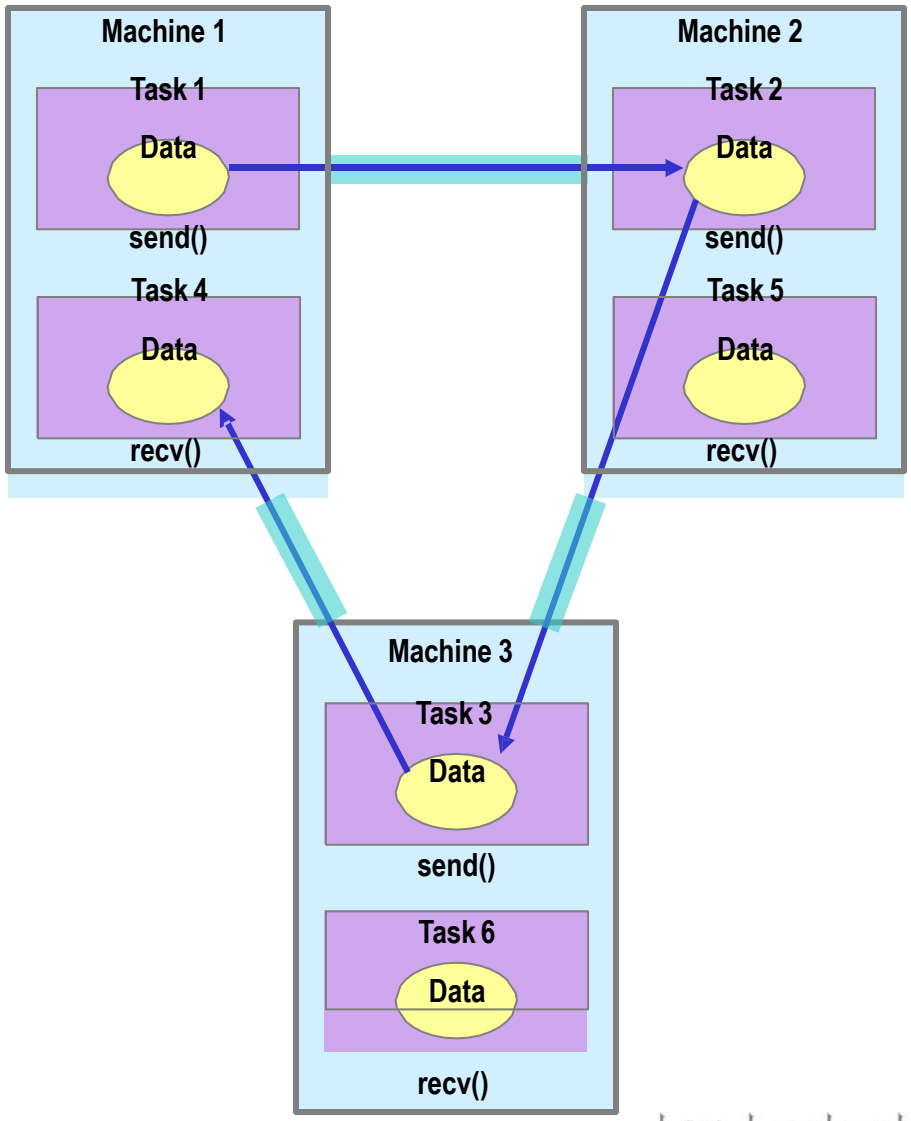


Parallel Programming Models (9/12)

- **Data Parallel Model Implementation**
  - Data-parallel compilers need the programmer to provide information that specifies how data is to be partitioned into tasks and how the data is distributed over the processors.
  - On shared memory architectures, all tasks can have access to the data structure through global memory so the data are not actually divided.
  - On distributed memory architectures, the data structure is divided into chunks and each chunk resides in the local memory of each task.

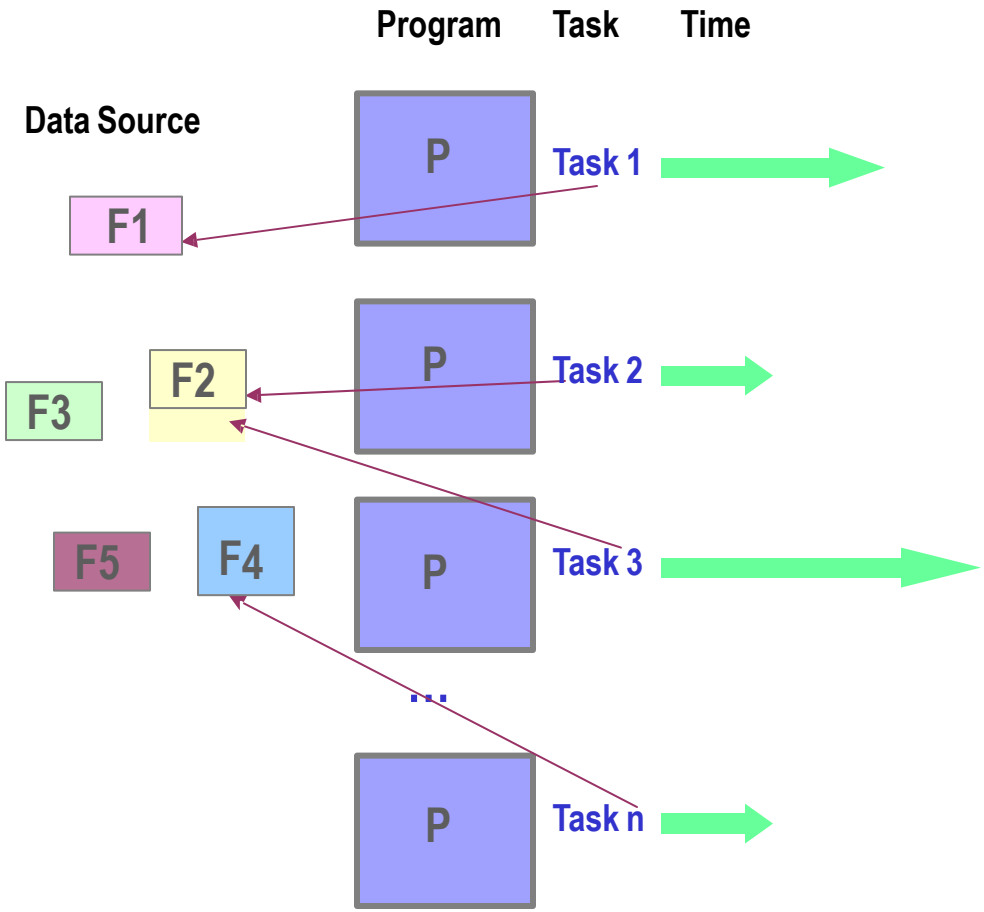
Parallel Programming Models (10/12)

- **Message Passing Model**
  - Tasks use their own local memory.
  - Tasks can be on the same machine or across multiple machines.
  - Data are exchanges between tasks by sending and receiving messages.
  - Data transfer requires cooperative operations between processes (ex: send followed by a matching receive).



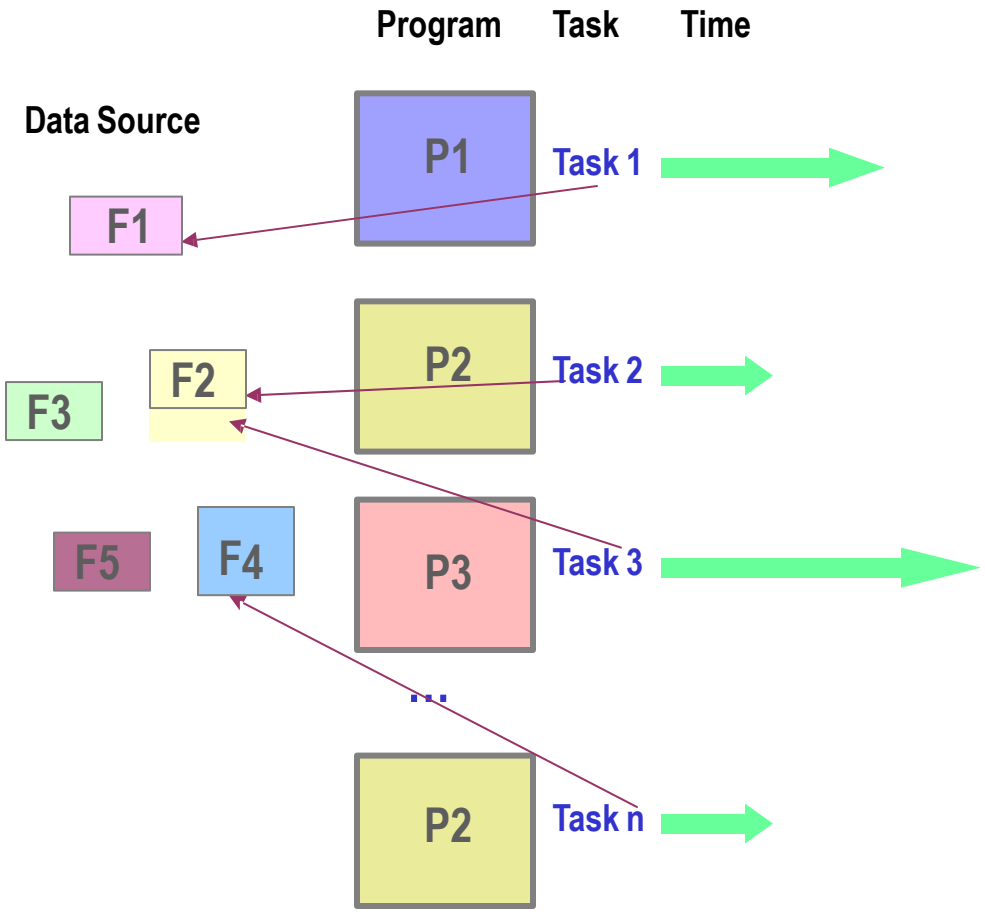
Parallel Programming Models (11/12)

- **Single Program Multiple Data (SPMD):**
  - Could be built by combining any of the mentioned parallel programming models. All tasks may use different data.
  - All tasks work simultaneously to execute one single program.
  - At any point during execution time, tasks can be executing the same or different instructions of the same program using different data resources.
  - Tasks do not always execute all of the programs, sometimes only parts of it.

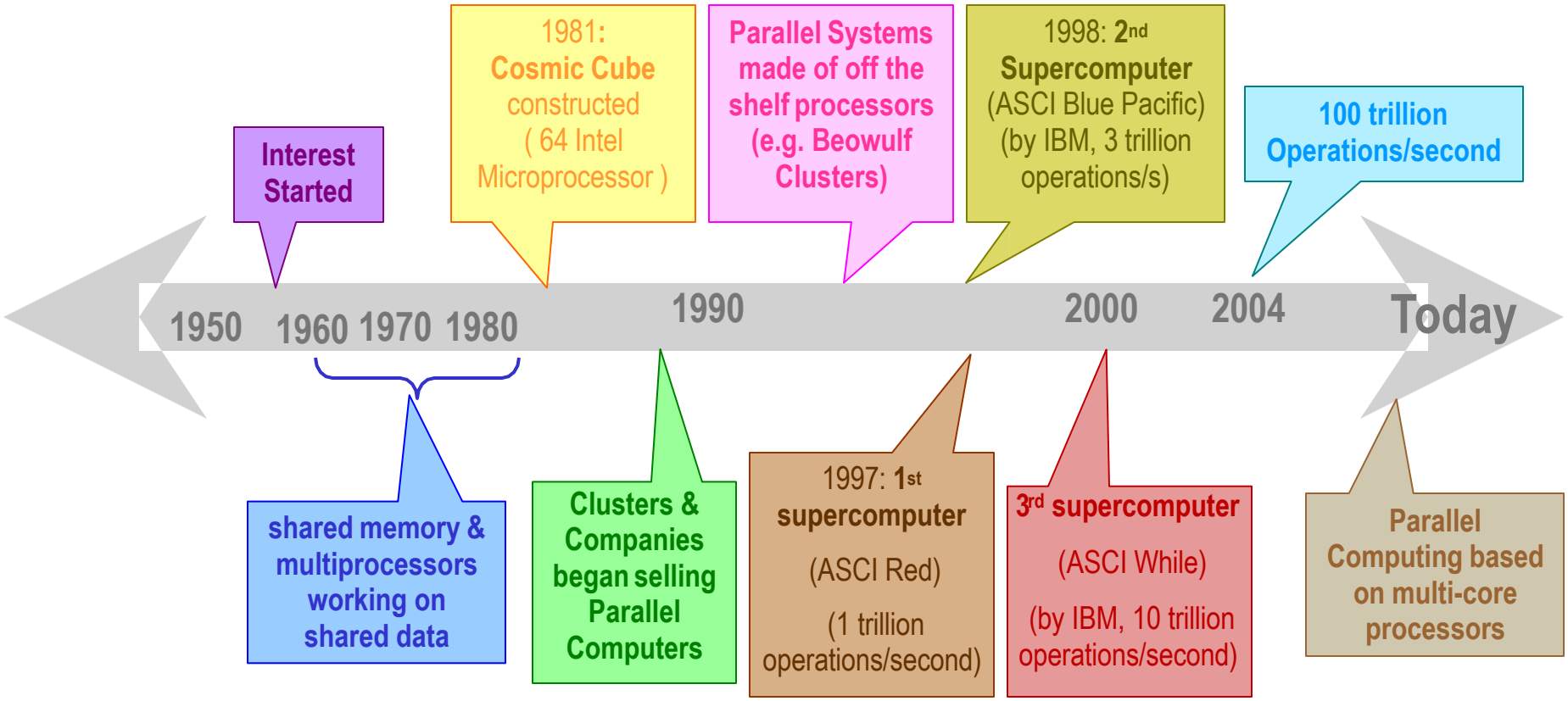


Parallel Programming Models (12/12)

- **Multiple Program Multiple Data (MPMD):**
  - Could be built by combining any of the mentioned parallel programming models.
  - Has many programs. The programs runs in parallel, but each task could be performing the same or different part of the program that other tasks perform.
  - All tasks may use different data

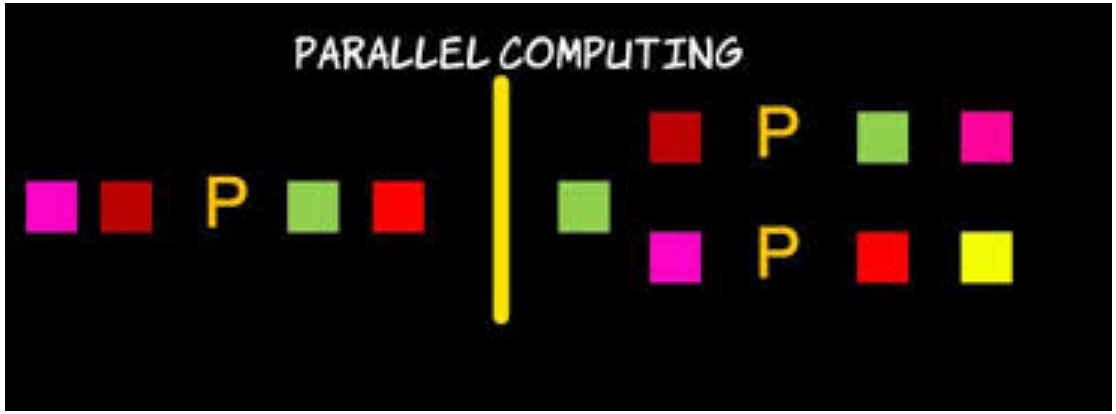


History of Parallel Computing



References

- http://en.wikipedia.org/wiki/Parallel_programming_model
- http://www.buyya.com/cluster/v2chap1.pdf
- https://computing.llnl.gov/tutorials/parallel_comp/
- http://www.acm.org/crossroads/xrds8-3/programming.html
- http://researchcomp.stanford.edu/hpc/archives/HPCparallel.pdf
- http://www.mcs.anl.gov/~itf/dbpp/text/node9.html
- http://en.wikipedia.org/wiki/Parallel_computing
- http://www.azalisaudi.com/para/Para-Week2-TypesOfPara.pdf



Thanks!