CS4172: Parallel and Distributed Computing

Lecture 1

Spring 24

INFS SST

RP: Nauman Ahmad

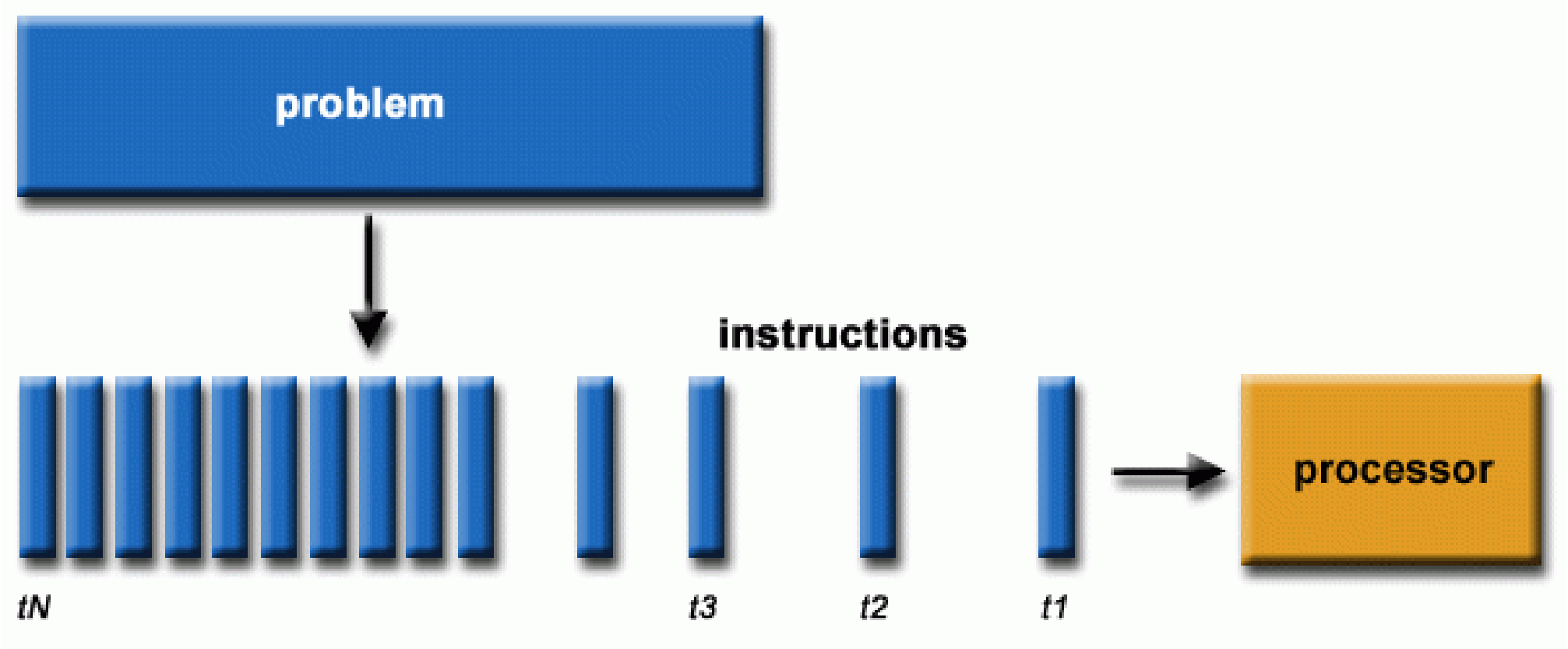
What Is Parallel Computing?

Serial Computing

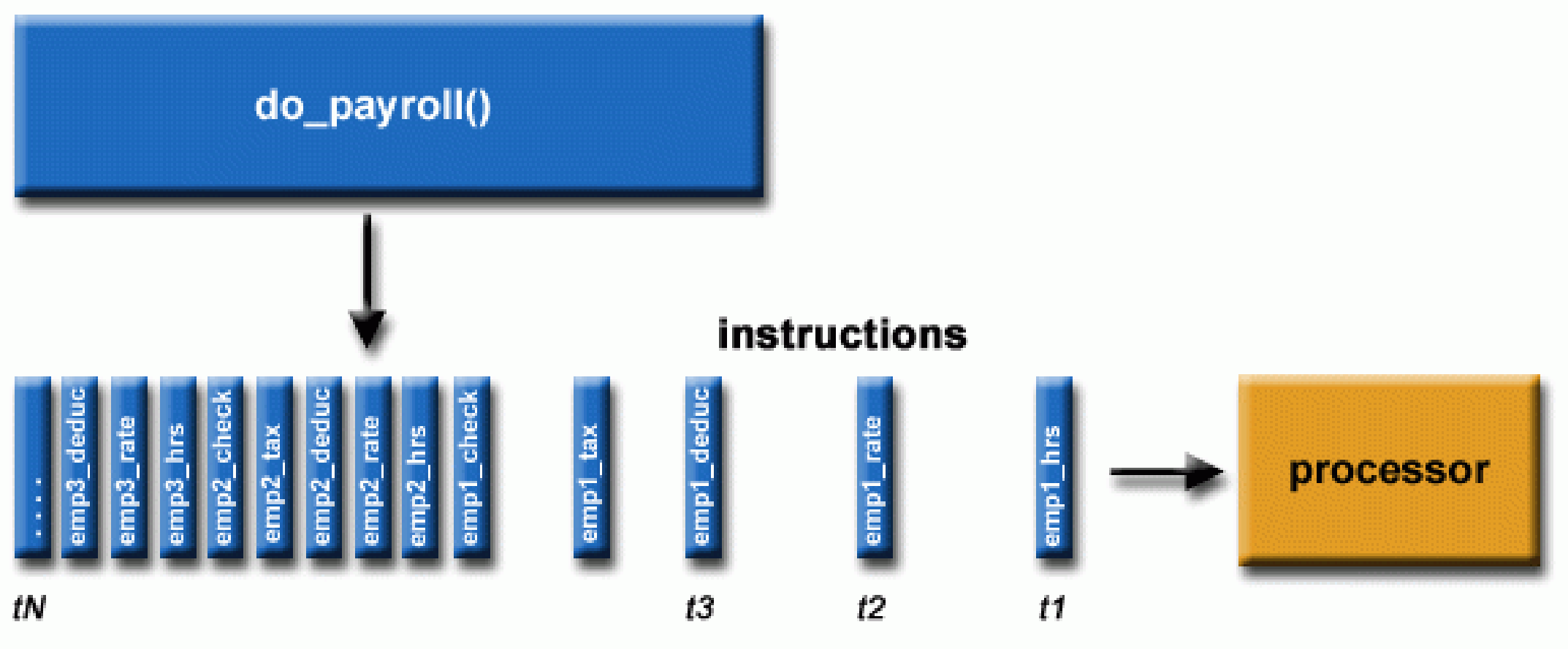
Traditionally, software has been written for ***serial*** computation:

- A problem is broken into a discrete series of instructions
- Instructions are executed sequentially one after another
- Executed on a single processor
- Only one instruction may execute at any moment in time

Serial computing generic example



Serial computing example of processing payroll

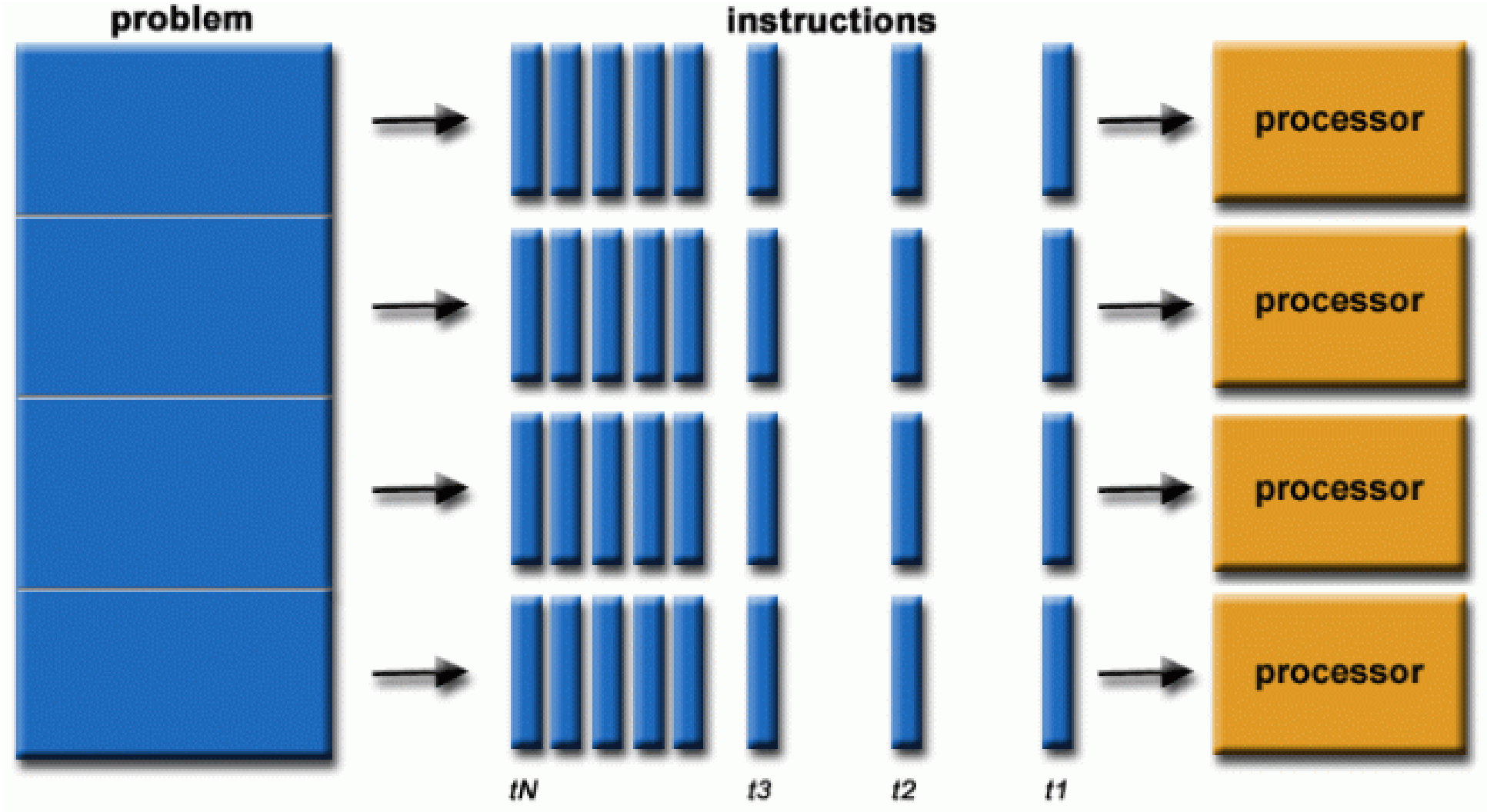


Parallel Computing

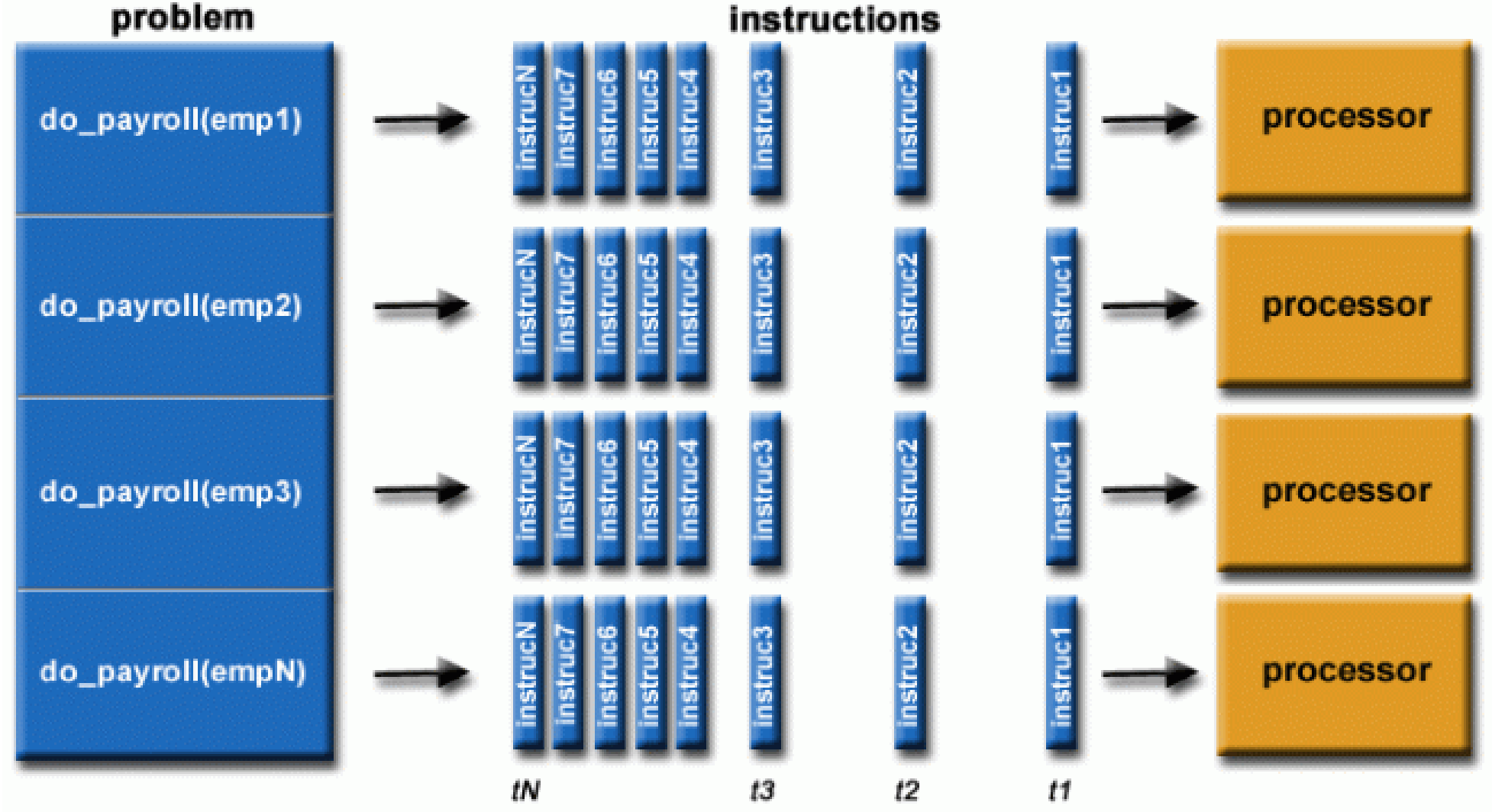
In the simplest sense, ***parallel computing*** is the simultaneous use of multiple computer resources to solve a computational problem:

- A problem is broken into discrete parts that can be solved concurrently
- Each part is further broken down to a series of instructions
- Instructions from each part execute simultaneously on different processors
- An overall control/coordination mechanism is employed

Parallel computing generic example



Parallel computing example of processing payroll



Requirements

- The computational problem should be able to:
  - Be broken apart into discrete pieces of work that can be solved simultaneously;
  - Execute multiple program instructions at any moment in time;
  - Be solved in less time with multiple compute resources than with a single compute resource.
- The compute resources are typically:
  - A single computer with multiple processors/cores
  - An arbitrary number of such computers connected by a network

Parallel Computers

- Virtually all stand-alone computers today are parallel from a hardware perspective:
  - Multiple functional units (L1 cache, L2 cache, branch, pre-fetch, decode, floating-point, graphics processing (GPU), integer, etc.)
  - Multiple execution units/cores

Thread

- A smaller unit of a process.
- It represents a sequence of instructions (Fetch, Decode, Execute and Store) that can be scheduled and executed independently by a Process Unit (PU).
- Threads within a process share the same code section and data section, but they have their own registers and stack space.
- **Hardware Support:** Thread-Level Parallelism (TLP) involves the concurrent execution of multiple threads, either within a single processor core (e.g., through hyper-threading) or across multiple cores.

Instruction

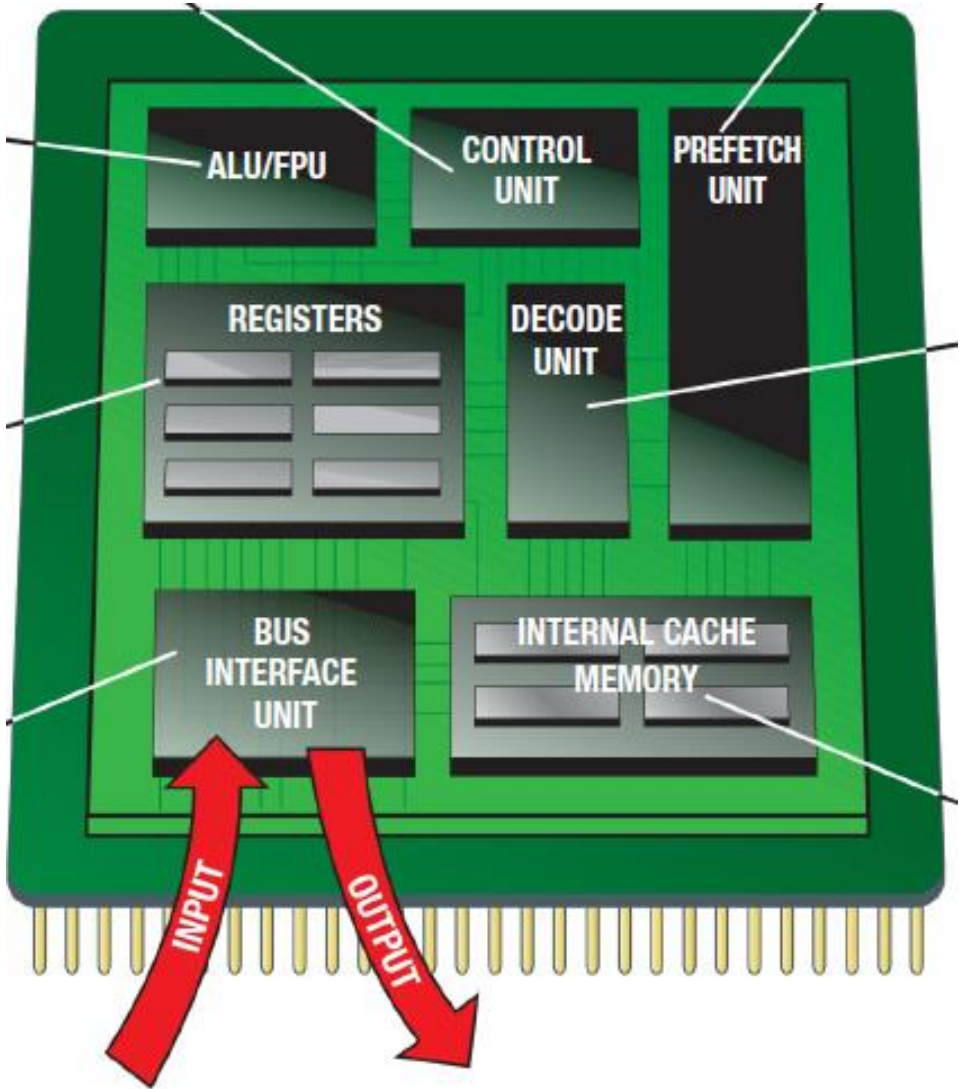
- An instruction is a fundamental operation that a computer's central processing unit (CPU) can execute. Instructions are part of the machine code that represents a program.
- **Instruction-Level Parallelism (ILP)** refers to the potential parallel execution of multiple instructions within a single processor core.
- Hardware Support: This can involve techniques like pipelining, instruction reordering, or the simultaneous execution of multiple instructions using different execution units.

Software and Hardware Parallelization Techniques Support each other

- The effectiveness of software parallelization techniques such as Thread-Level Parallelism (TLP) and Instruction-Level Parallelism (IPL) can indeed be enhanced by underlying Hardware features like pipelining or other forms of parallelism.

Inside a CPU Core

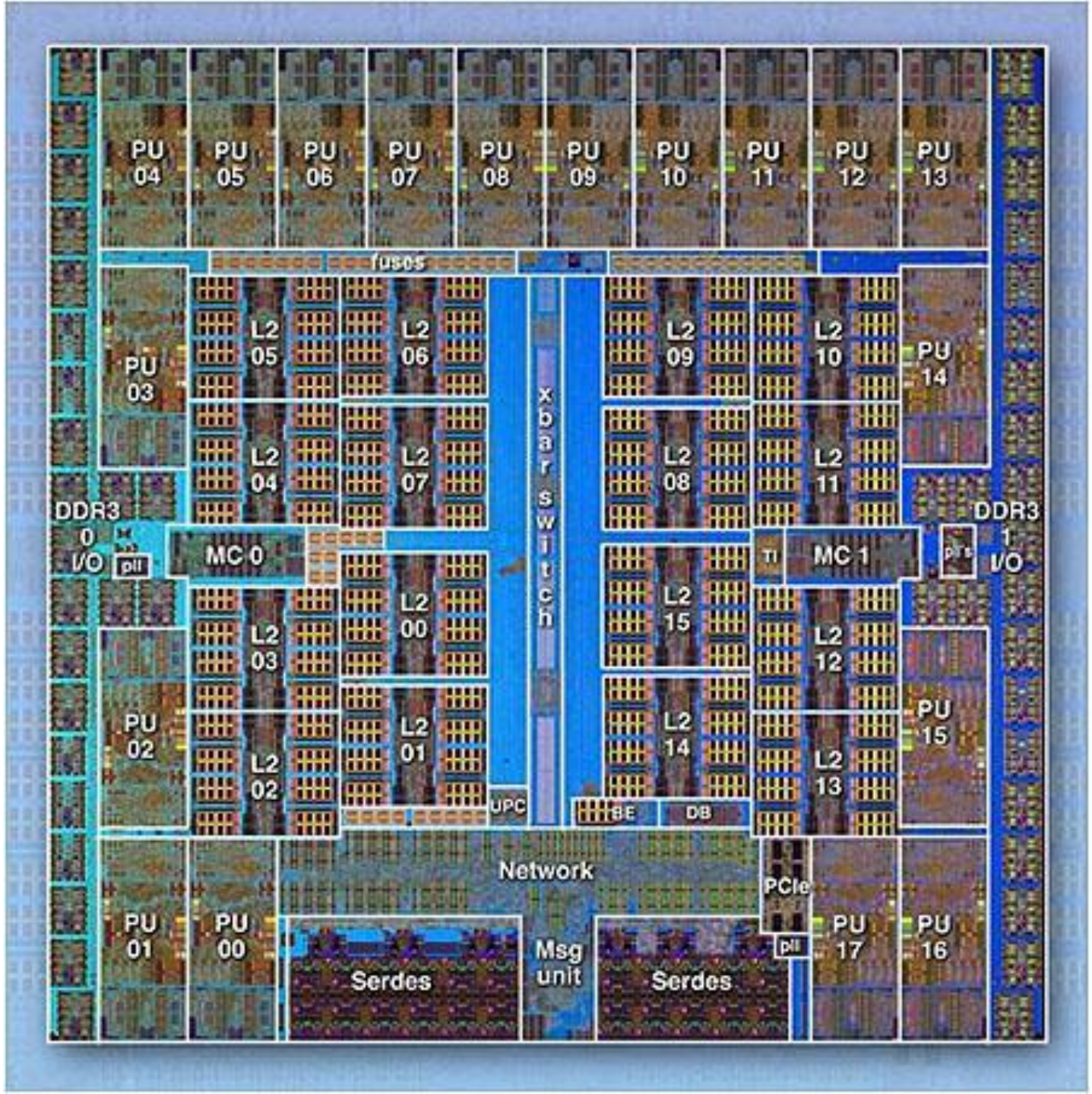
- **Core: a physical Process Unit (PU) that is capable of executing its own set of instructions independently.**
- **Now a days, a CPU contains many Cores.**



- **A Core handling multiple Threads simultaneously/c on-currently is called Hyper-Threading.**

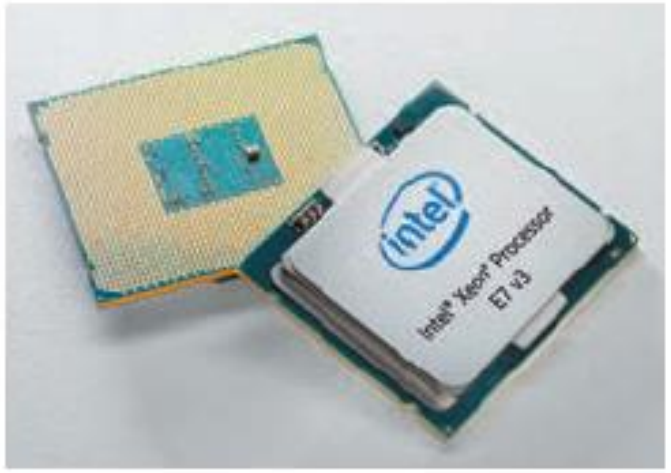
Socket (from hardware parallelization concept)

- Cores/PU can be arranged in one or more sockets with memory sharing.

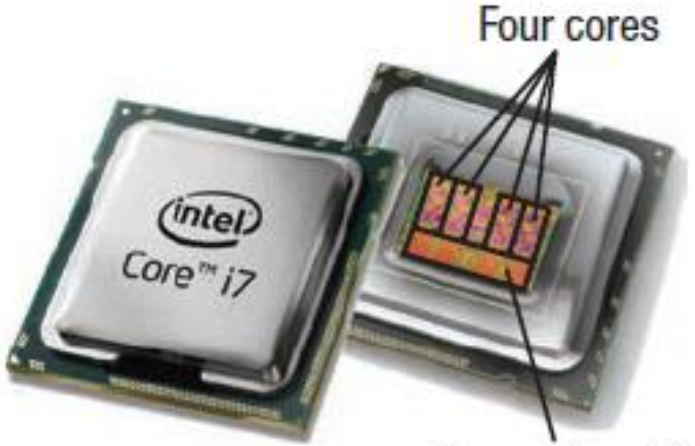


IBM BG/Q Compute Chip with 18 cores (PU) and 16 L2 Cache units (L2)

CPU Examples



SERVER PROCESSORS



DESKTOP PROCESSORS

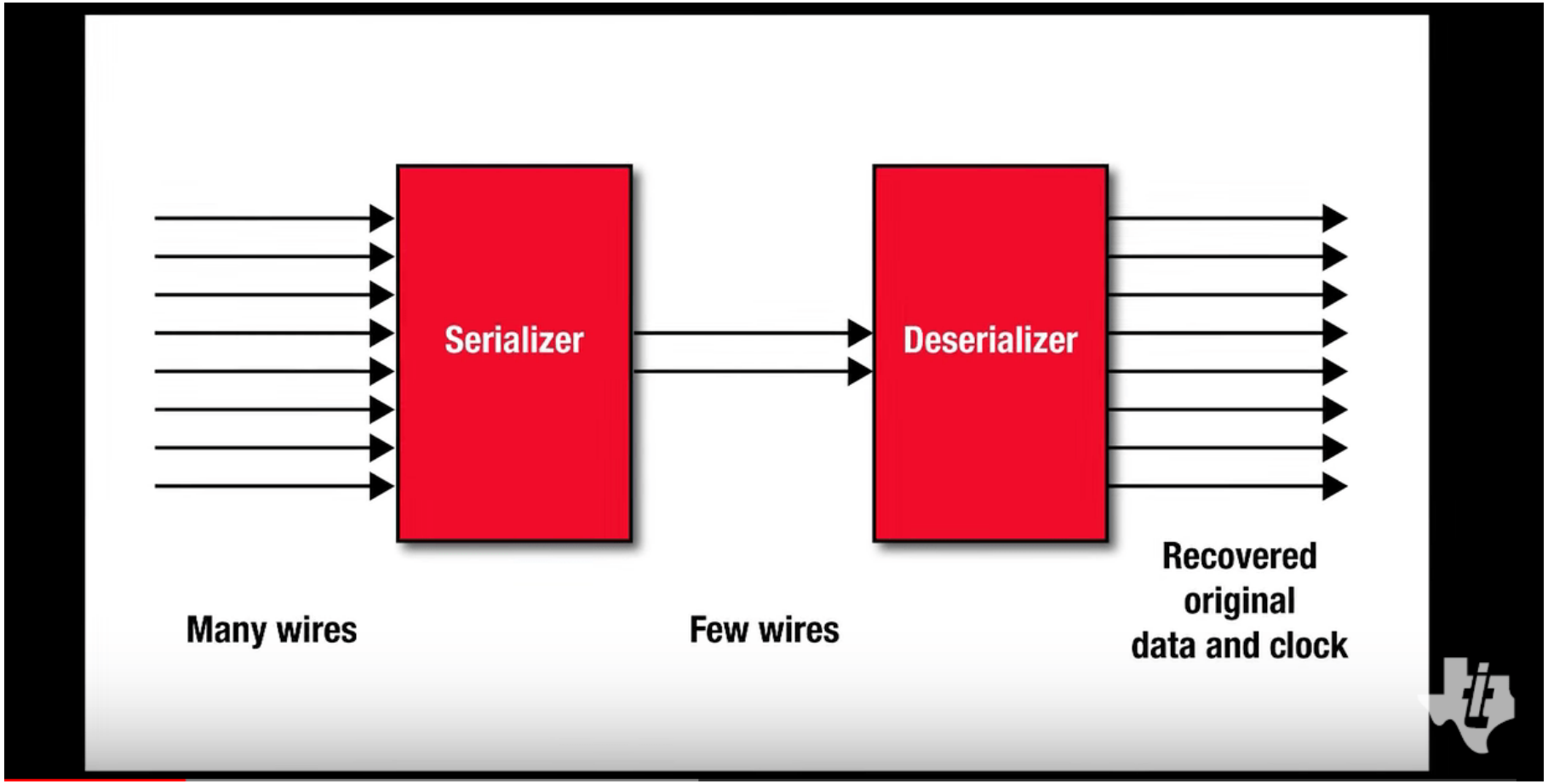
MOBILE PROCESSORS

CPU Examples

TYPE OF CPU	NAME	NUMBER OF CORES
SERVER	Intel Xeon (E7 family)	4–18
	AMD Opteron (6300 series)	4–16
DESKTOP	Intel Core i7 (6th gen)	4–8
	AMD FX	4–8
MOBILE (NOTEBOOKS)	Intel Core M	2
	Intel Atom x7	4
	Intel Celeron	1–4
MOBILE (MOBILE DEVICES)	Intel Atom x5	4
	ARM Cortex-A17	4
	NVIDIA Tegra 4*	4

*Based on ARM Cortex-A15

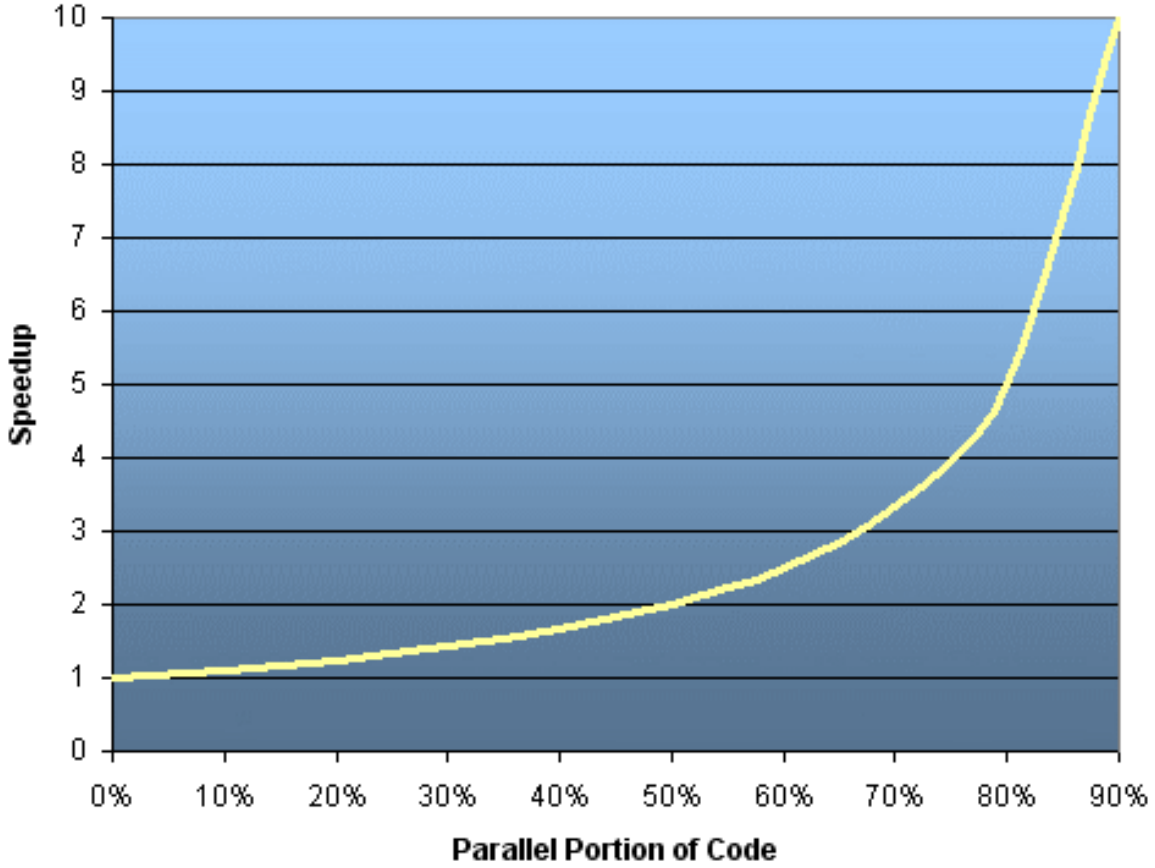
Serdes

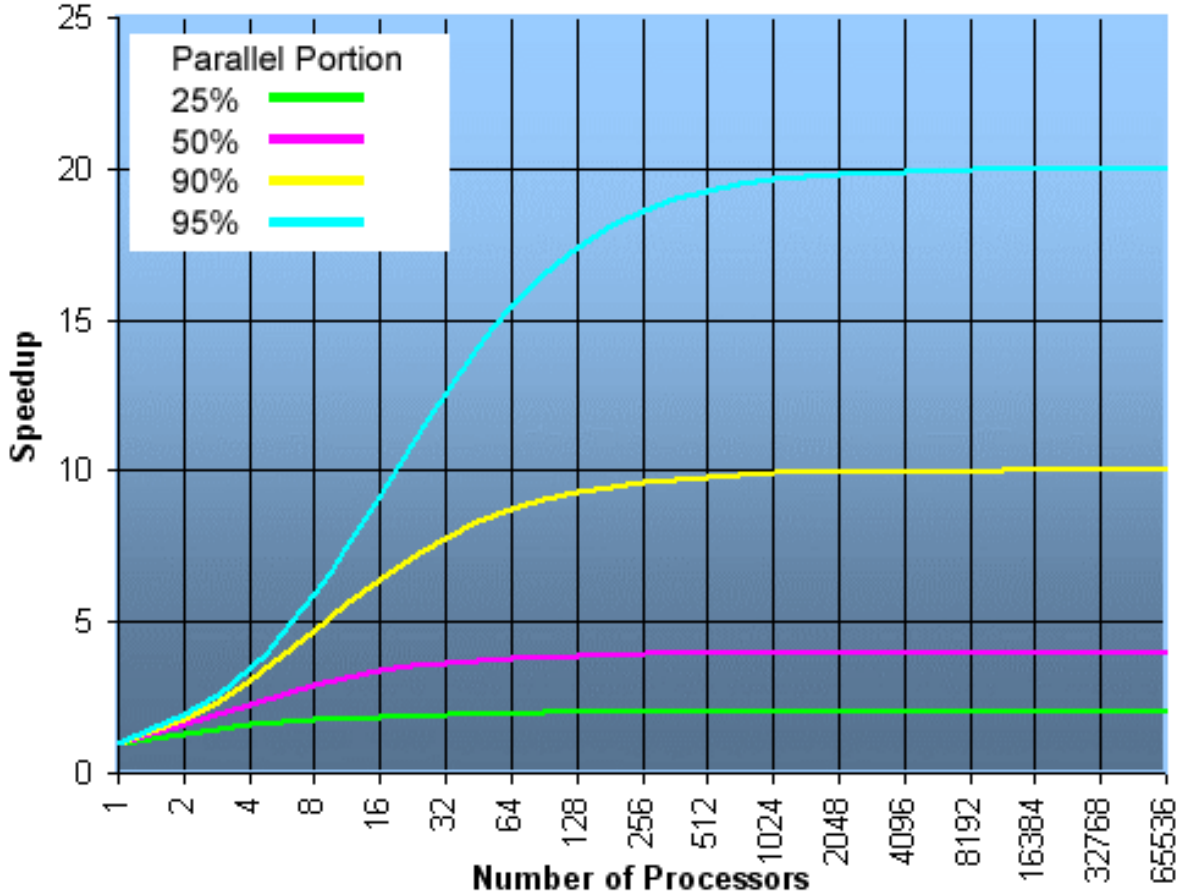

Serializer
Deserializer
Many wires
Few wires
Recovered original data and clock

Speed Up

- Clock Rate: a system clock runs at 200 MHz. CPU clock speed is 2 GHz. It means CPU clock essentially ticks 10 times during each system clock tick.
- Memory Size
  - SRAM (Cache) and DRAM (Main memory)
- No. of Cores (PU)
- Instruction Type
  - Serial
  - Parallel
- There are Scalability problems: the hidden challenges of growing a system.

Speed Up: Amdahl's law





Speed Up: Amdahl's law

$$\text{speedup} = \frac{1}{1 - P}$$

- (P): the fraction of code that can be parallelized.
- If none of the code can be parallelized, P = 0 and the speedup = 1 (no speedup).
- If all of the code is parallelized, P = 1 and the speedup is infinite (in theory).
- If 50% of the code can be parallelized, maximum speedup = 2, meaning the code will run twice as fast.

Speed Up: No. of Processors

- Introducing the number of processors performing the parallel fraction of work, the relationship can be modeled by:

$$\text{speedup} = \frac{1}{\frac{P}{N} + S}$$

- where P = parallel fraction, N = number of processors and S = serial fraction.

Speed Up: No. of Processors (contd.)

- It soon becomes obvious that there are limits to the scalability of parallelism. For example:

	speedup			
N	P = .50	P = .90	P = .95	P = .99
10	1.82	5.26	6.89	9.17
100	1.98	9.17	16.80	50.25
1,000	1.99	9.91	19.62	90.99
10,000	1.99	9.91	19.96	99.02
100,000	1.99	9.99	19.99	99.90

- "Famous" quote: *You can spend a lifetime getting 95% of your code to be parallel, and never achieve better than 20x speedup no matter how many processors you throw at it!*

Architectural Classifications

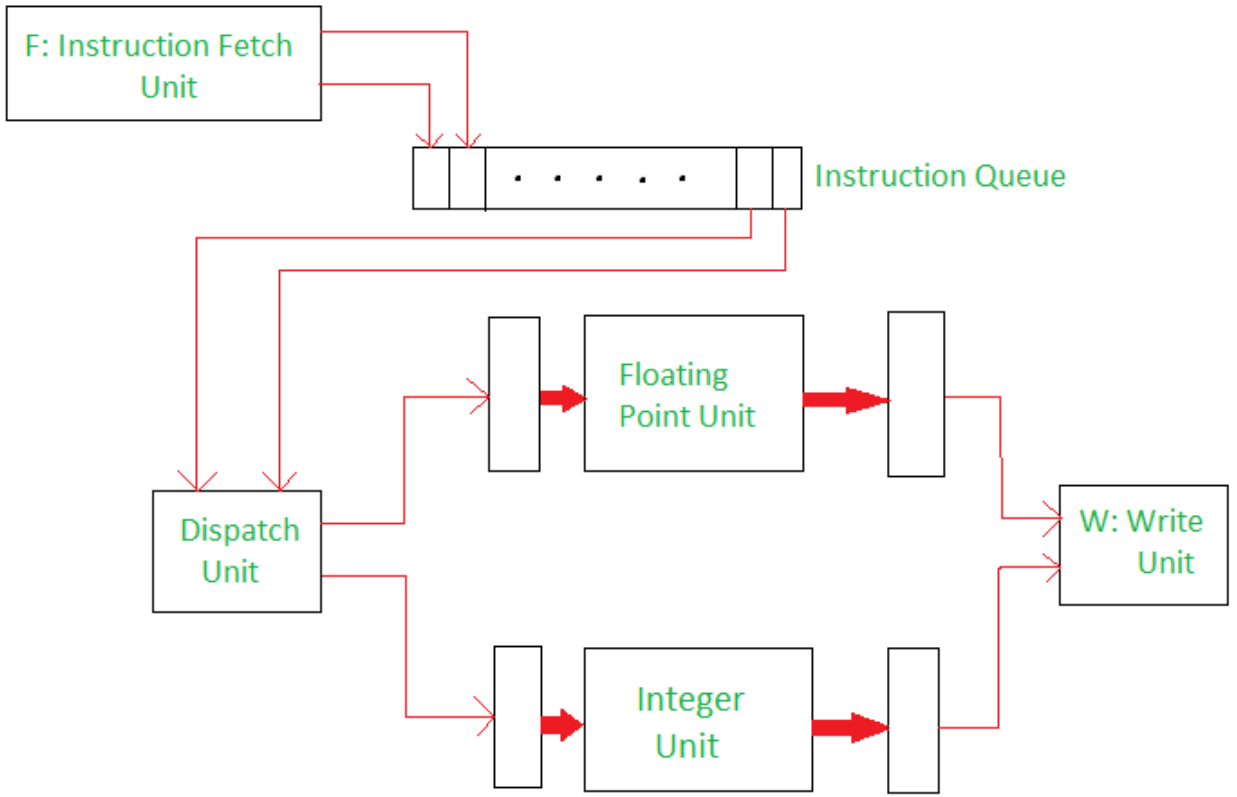
Hardware Parallel computing can be broadly classified as:

- Superscalar and Multicore architectures are real-world parallel processing architectures that exist and have been widely implemented in modern computing systems.
- These classifications are concerned with the internal design and capabilities of a single processor or a collection of processors within a system.

Superscalar Architecture

- **Superscalar architecture** refers to a processor design that allows the execution of multiple instructions in parallel during a single clock cycle.
- It involves multiple execution units within a single core, enabling the processor to simultaneously execute multiple instructions from a single instruction stream.
- **Multiple Execution Units:** Superscalar processors have multiple functional units, such as arithmetic logic units (ALUs), floating-point units (FPUs), and others, allowing the simultaneous execution of different types of instructions.
- **Dynamic Instruction Scheduling:** Superscalar processors often employ techniques like out-of-order execution and dynamic scheduling to maximize instruction throughput.

Superscalar Architecture



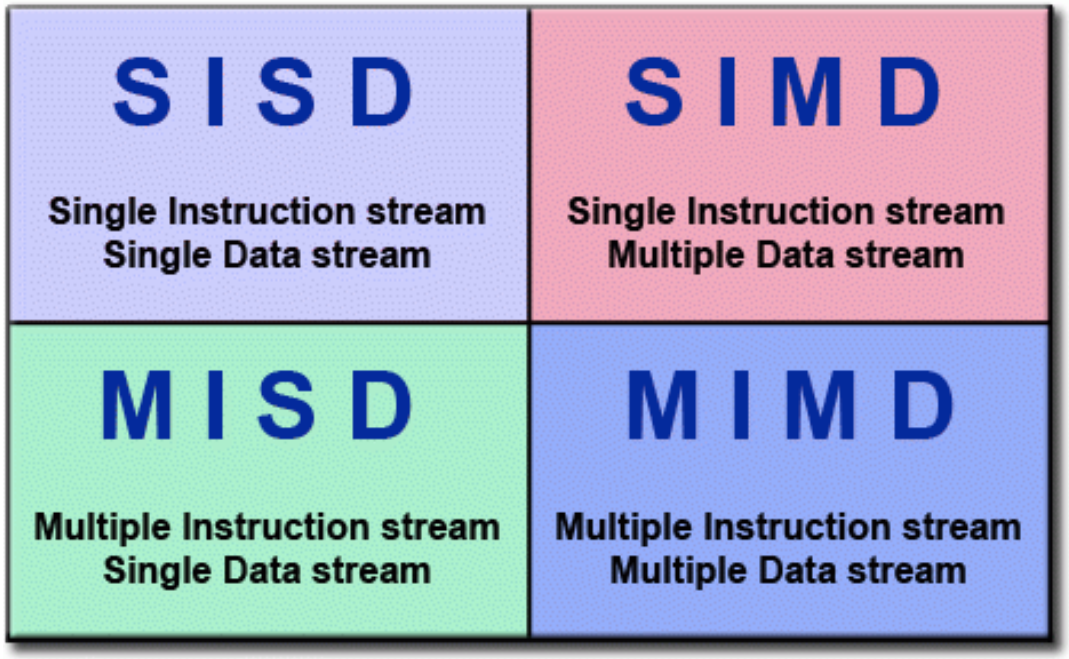
Processor with Two Execution Units

Multicore Architecture

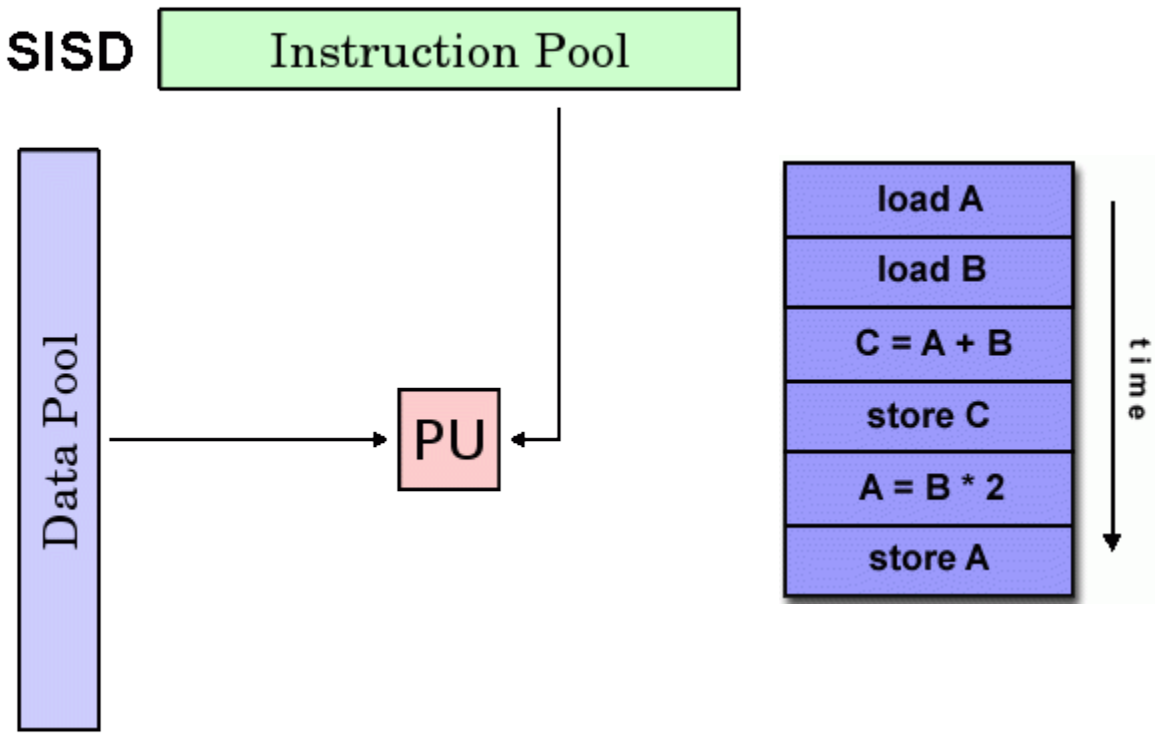
- **Multicore architecture** involves integrating 2 or more individual processor cores onto a single chip. Each core operates independently, with its own set of registers, execution units, and cache memory.
- **True Parallelism:** Multicore architecture achieves true parallelism by allowing multiple cores to execute different tasks simultaneously.
- **Parallelism Between Cores:** It exploits parallelism by allowing multiple cores to work on different tasks concurrently, enabling better overall system performance.

Flynn's Taxonomy (hardware-oriented classification system for parallel computing architectures)

- Flynn's Taxonomy is a classification system proposed by Michael J. Flynn in 1966 to categorize computer architectures based on the number of instruction streams (I) and data streams (D) that a computer can process simultaneously.

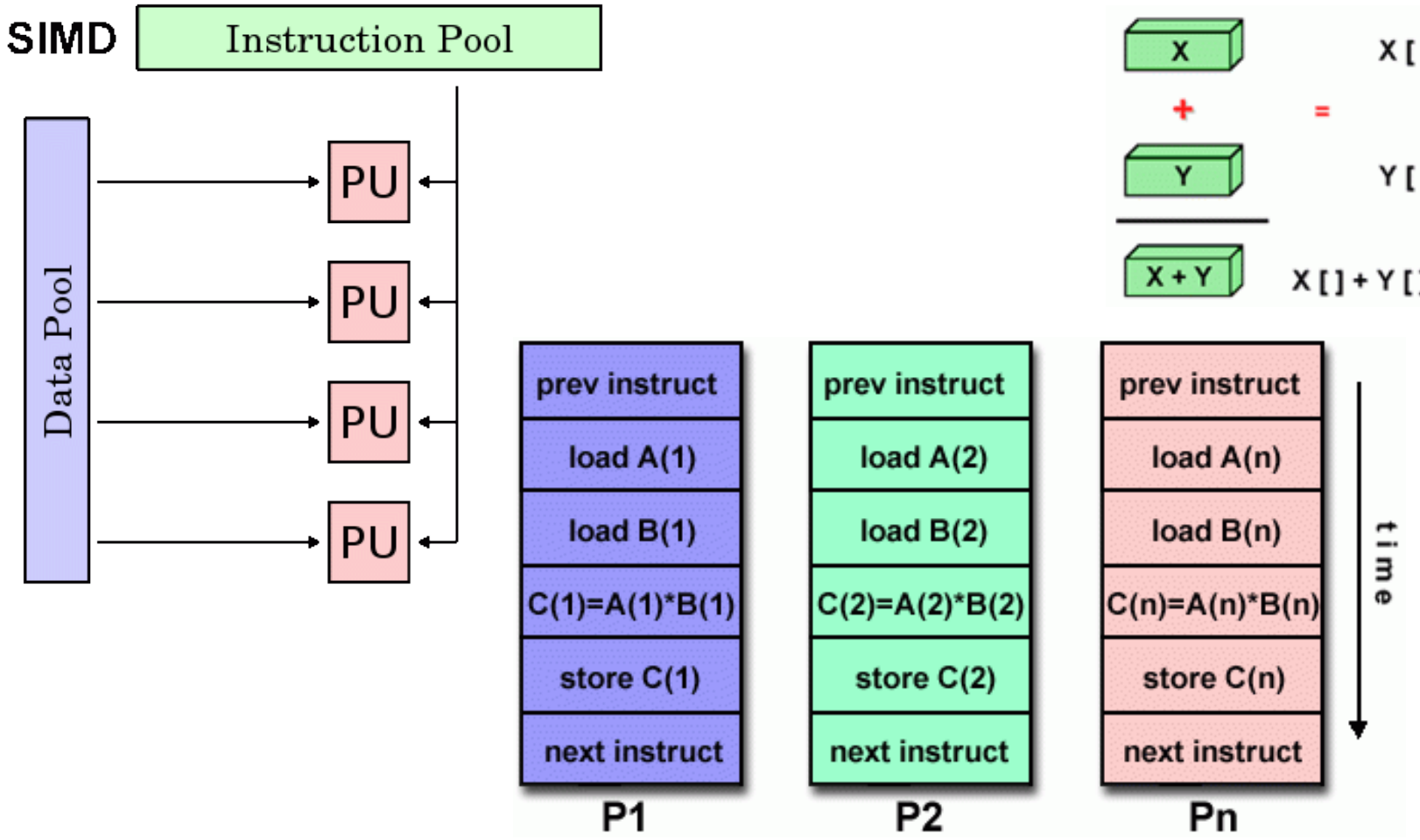


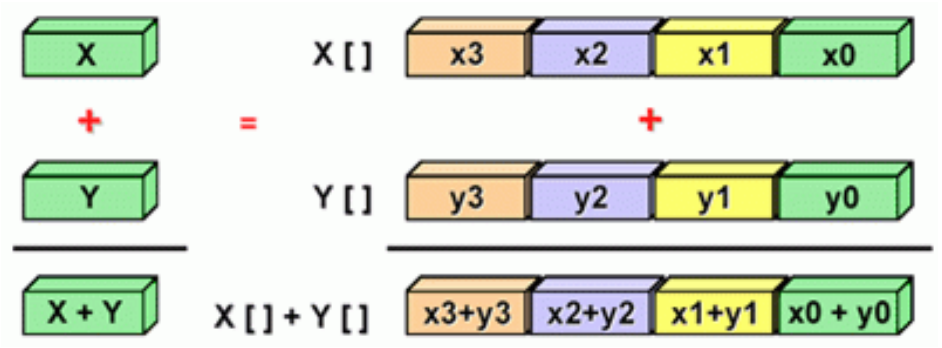
Single Instruction, Single Data (SISD)



- **Superscalar architecture aligns with the SISD category.**
- **A serial (non-parallel) computer (some researchers think)**
- **Examples: older generation mainframes, minicomputers, workstations and single processor/core PCs.**

Single Instruction, Multiple Data (SIMD)

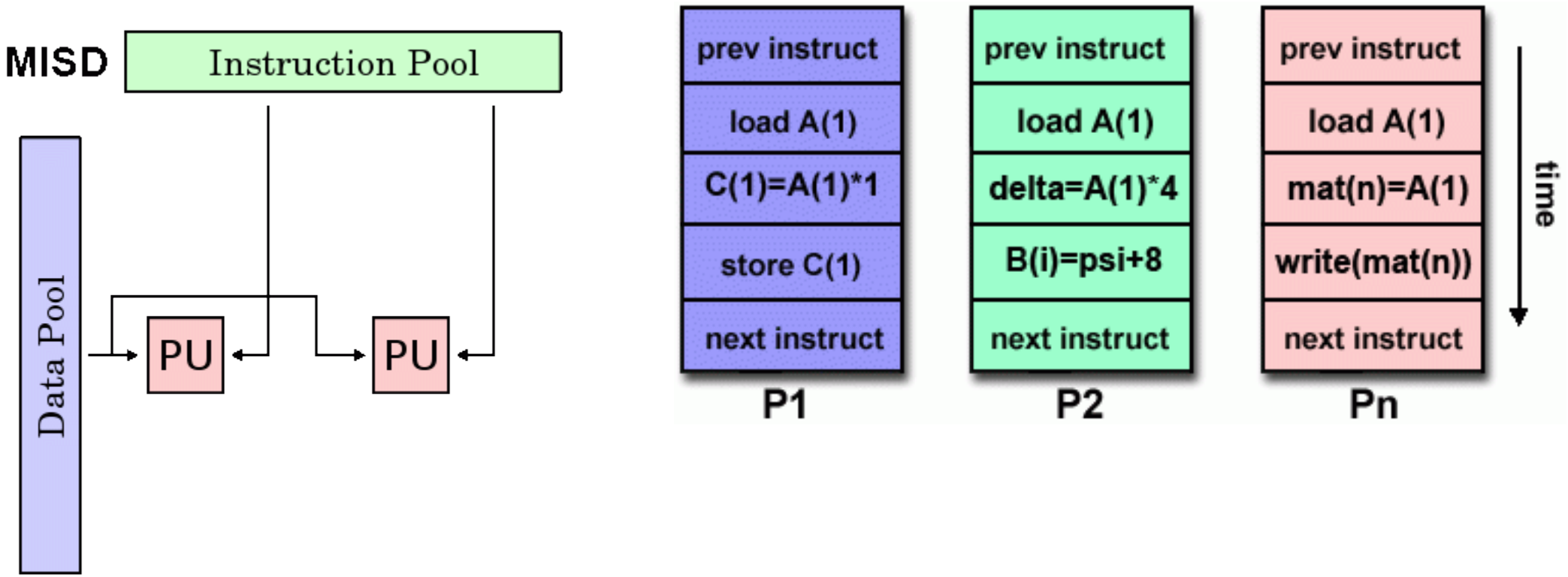




Single Instruction, Multiple Data (SIMD)

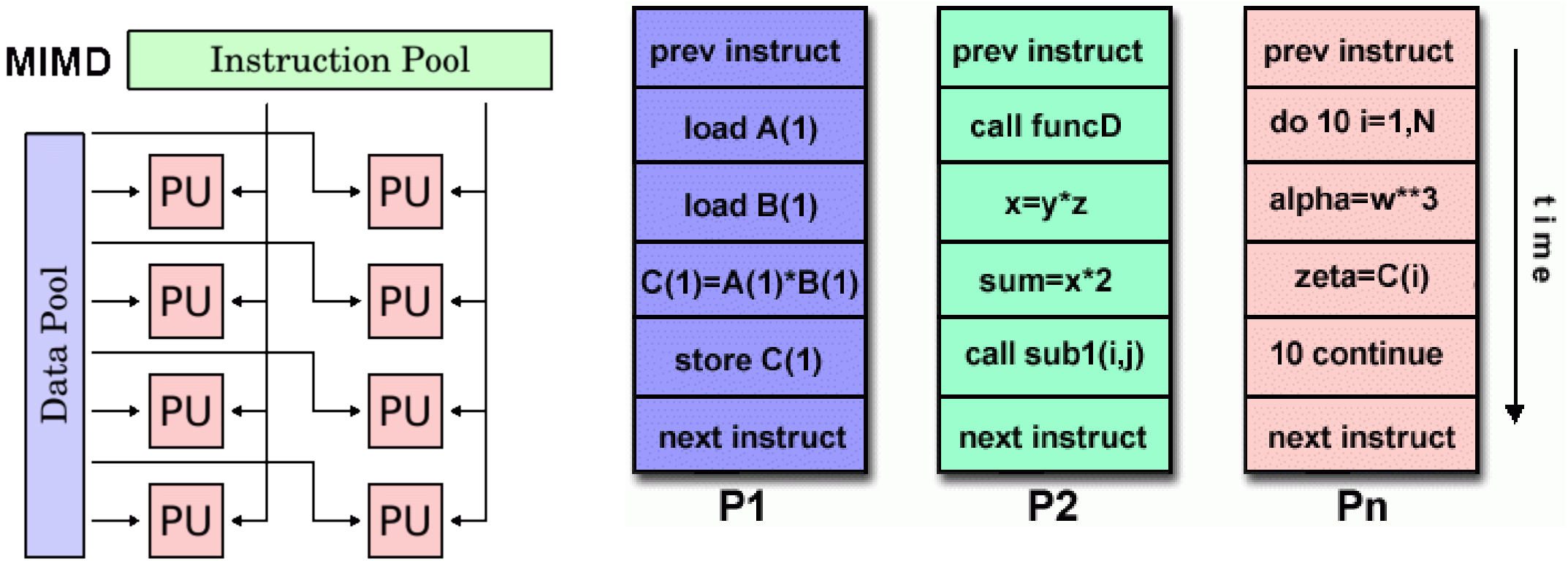
- A type of parallel computer
- **Single Instruction:** All processing units execute the same instruction at any given clock cycle
- **Multiple Data:** Each processing unit can operate on a different data element
- Best suited for specialized problems characterized by a high degree of regularity, such as graphics/image processing.
- Synchronous (lockstep) and deterministic execution
- Two varieties: Processor Arrays and Vector Pipelines
- Examples:
  - Processor Arrays: Thinking Machines CM-2, MasPar MP-1 & MP-2, ILLIAC IV
  - 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) and FPGAs employ SIMD instructions and execution units.

Multiple Instruction, Single Data (MISD)



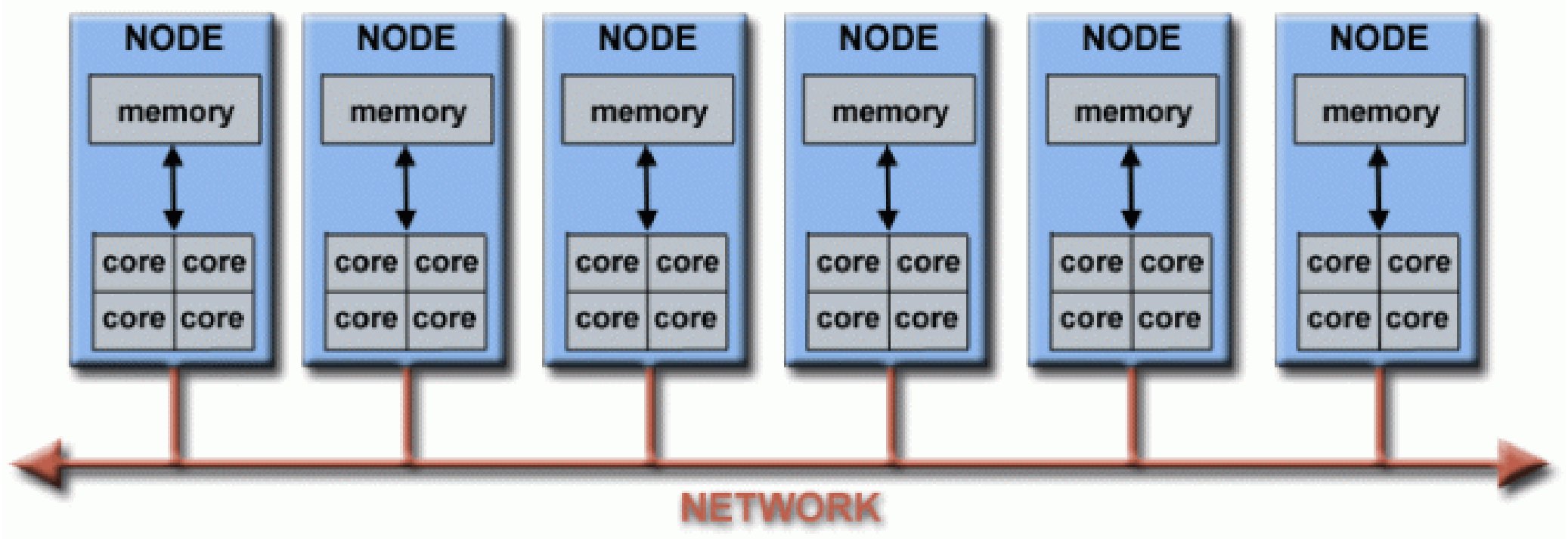
It is a Hypothetical architecture.

Multiple Instruction, Multiple Data (MIMD)



- **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.**

Network connections – Distributed Computing



- Networks connect multiple stand-alone computers (nodes) to make larger parallel computer clusters.

Key Characteristics of Distributed Computing

- **Parallel Processing:** Nodes in a distributed computing system can work on different parts of a problem simultaneously, allowing for parallel processing and potentially reducing the overall computation time.
- **Resource Sharing:** Distributed computing systems often involve the sharing of resources, such as computational power, memory, or storage, across multiple nodes. This enables more efficient utilization of resources.
- **Fault Tolerance:** Distributed systems can be designed to be fault-tolerant. If one node fails, the overall system can continue to function with the remaining nodes. This enhances reliability and availability.
- **Scalability:** Distributed systems can be easily scaled by adding more nodes to the network. This scalability is essential for handling larger workloads or accommodating increased demand.
- **Decentralized Control:** Unlike centralized systems where a single entity controls the entire system, distributed systems often have decentralized control. Each node can make decisions independently based on local information.

Examples of Distributed Computing Systems

- Grid Computing: geographically dispersed computers are connected to work on a common task.
- Cluster Computing: involves the use of interconnected computers (nodes) that work together closely within a specific location.
- Cloud Computing: extends the idea of distributed computing by providing on-demand access to a shared pool of computing resources over the internet.

Course Statistics (expect changes as per cond)

- Midterm Exam 20 - 25
- Final Exam 40
- Assignments 15
- Quizzes 15
- Project 10