Introduction to Parallel Processing

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.

Parallelism in Uniprocessor Systems

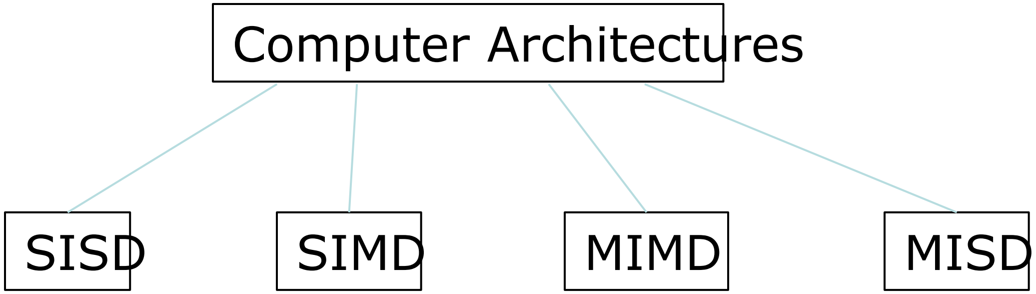
- It is possible to achieve parallelism with a uniprocessor system.
  - Some examples are the instruction pipeline, arithmetic pipeline, I/O processor.
- Note that a system that performs different operations on the same instruction is not considered parallel.
- Only if the system processes two different 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.
  - Classification based on the instruction and data streams

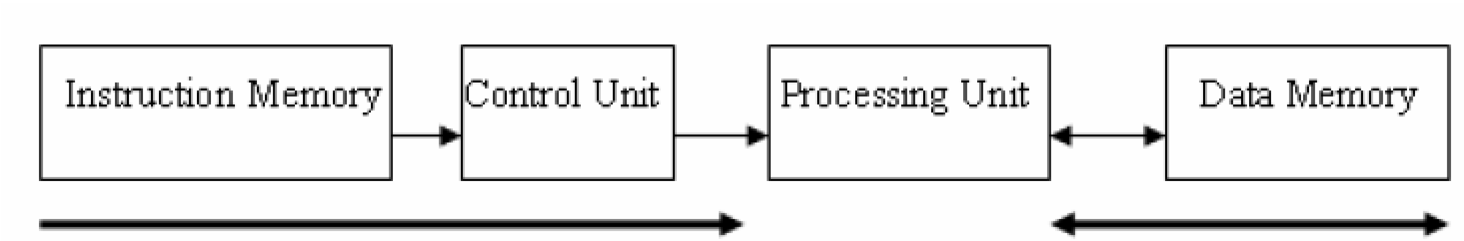
Flynn’s taxonomy

- Single Instruction Single Data (SISD)
  - Traditional sequential computing systems
- Single Instruction Multiple Data (SIMD)
- Multiple Instructions Multiple Data (MIMD)
- Multiple Instructions Single Data (MISD)



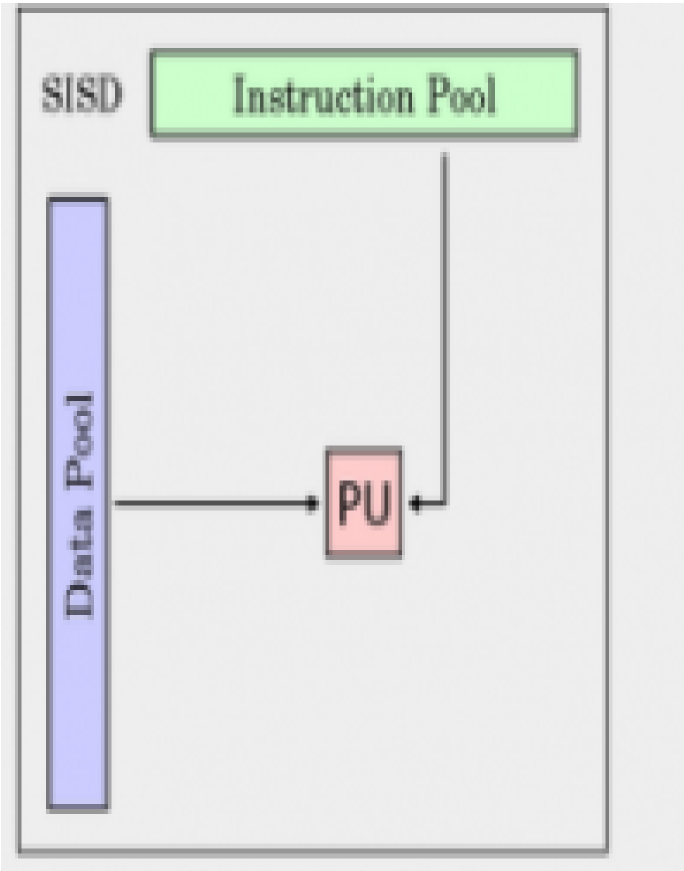
SISD

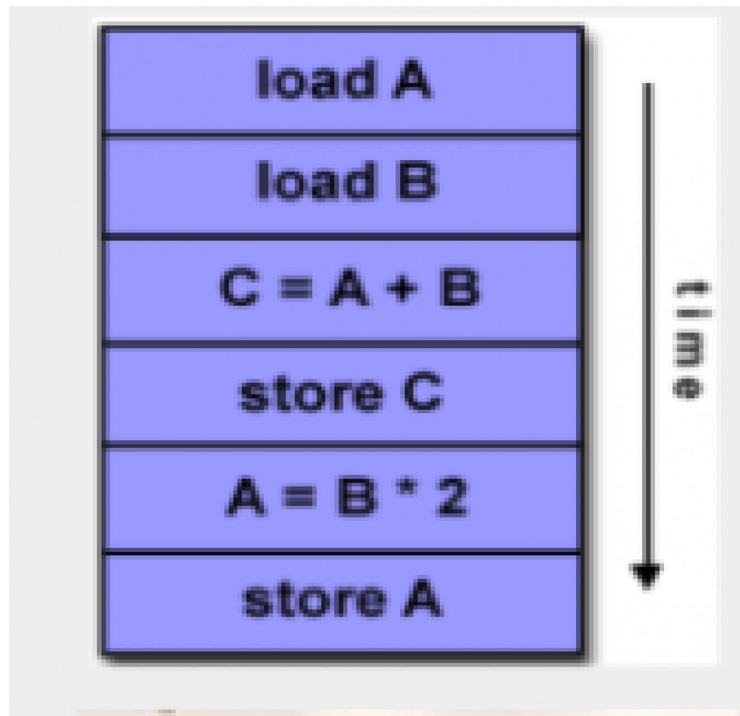
- A serial (non-parallel) computer
- Single Instruction: Only one instruction stream is being acted on by the CPU during any one clock cycle
- Single Data: Only one data stream is being used as input during any one clock cycle
- one stream of instructions and one stream of data. $I_s = D_s = 1$
- Deterministic execution
- This is the oldest and even today, the most common type of computer use it.
- Examples: older generation mainframes, minicomputers and work stations; most modern day PCs.



(a) SISD

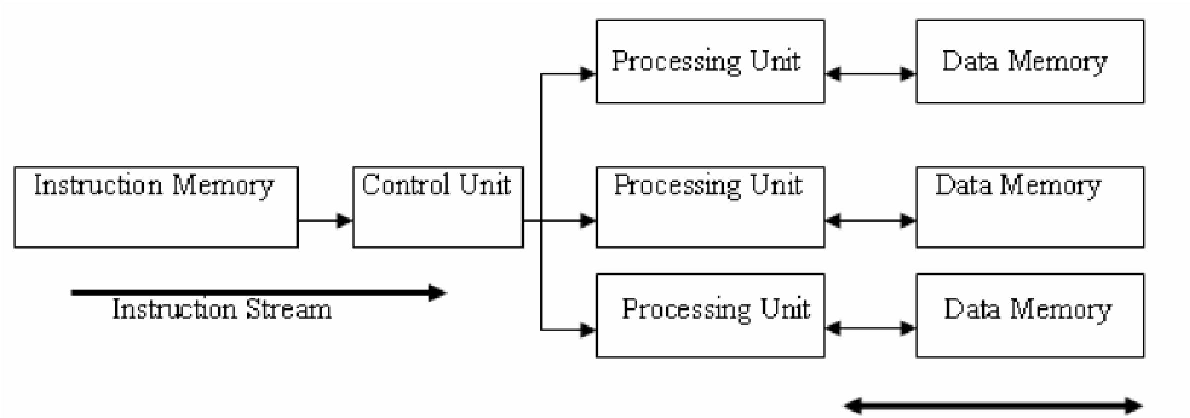
SISD


SISD
Instruction Pool
Data Pool
PU


load A
load B
C = A + B
store C
A = B * 2
store A
time

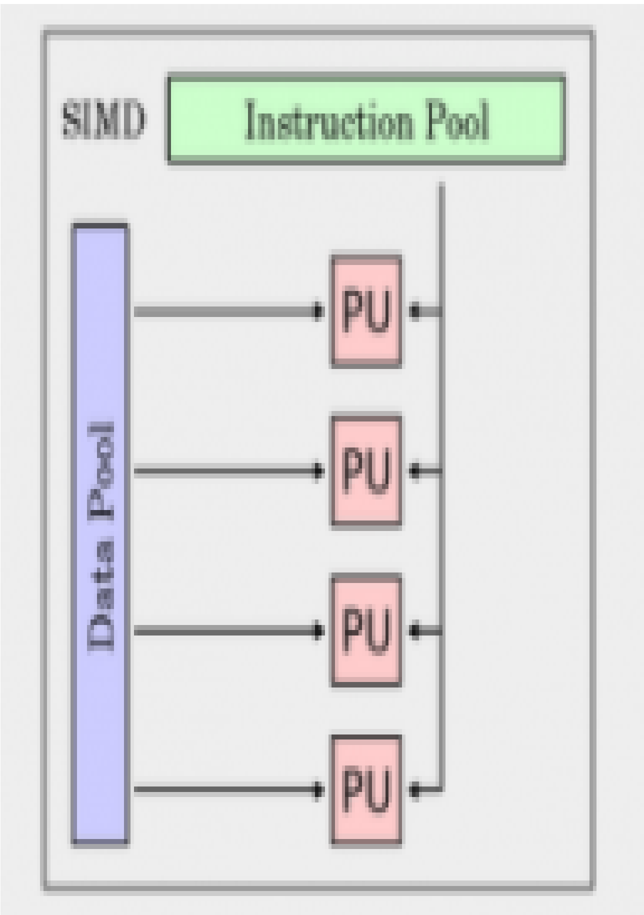
SIMD

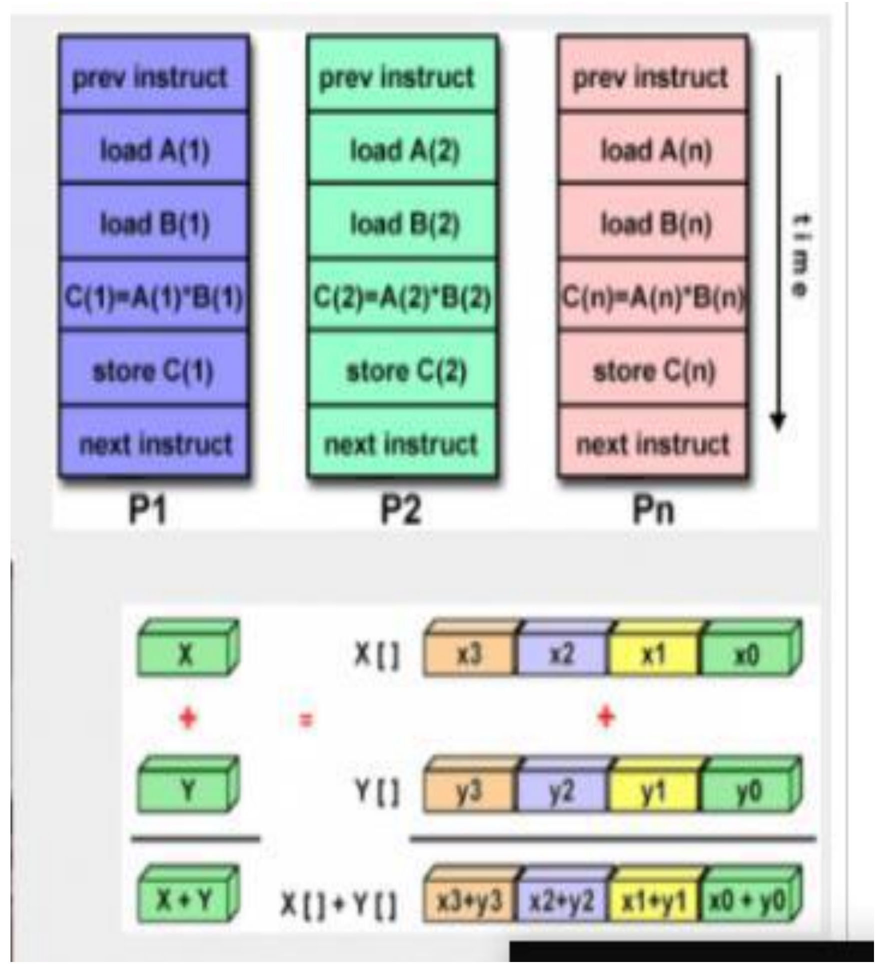
- A type of parallel computer
- Single Instruction: All processing units execute the same instruction at any given clock cycle
- Multiple Data: Each processing unit can operate on a different data element
- Best suited for specialized problems characterized by a high degree of regularity, such as graphics/image processing.
- Synchronous (lockstep) and deterministic execution
- Two varieties: Processor Arrays and Vector Pipelines
- $I_s = 1$ $Ds > 1$.



(b) SIMD

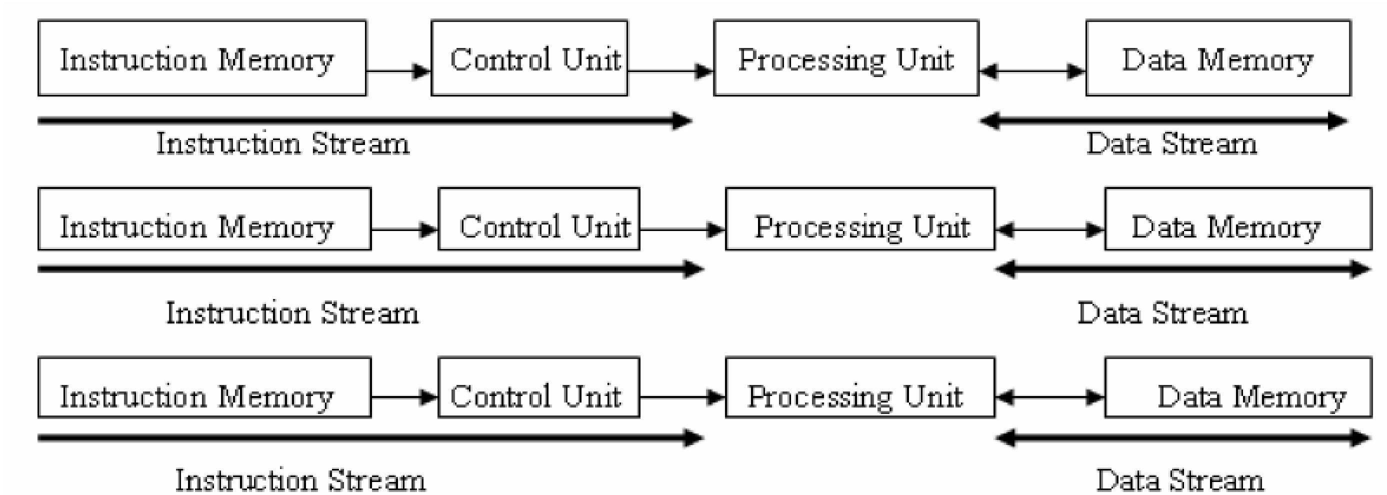
SIMD


SIMD
Instruction Pool
Data Pool
PU
PU
PU
PU

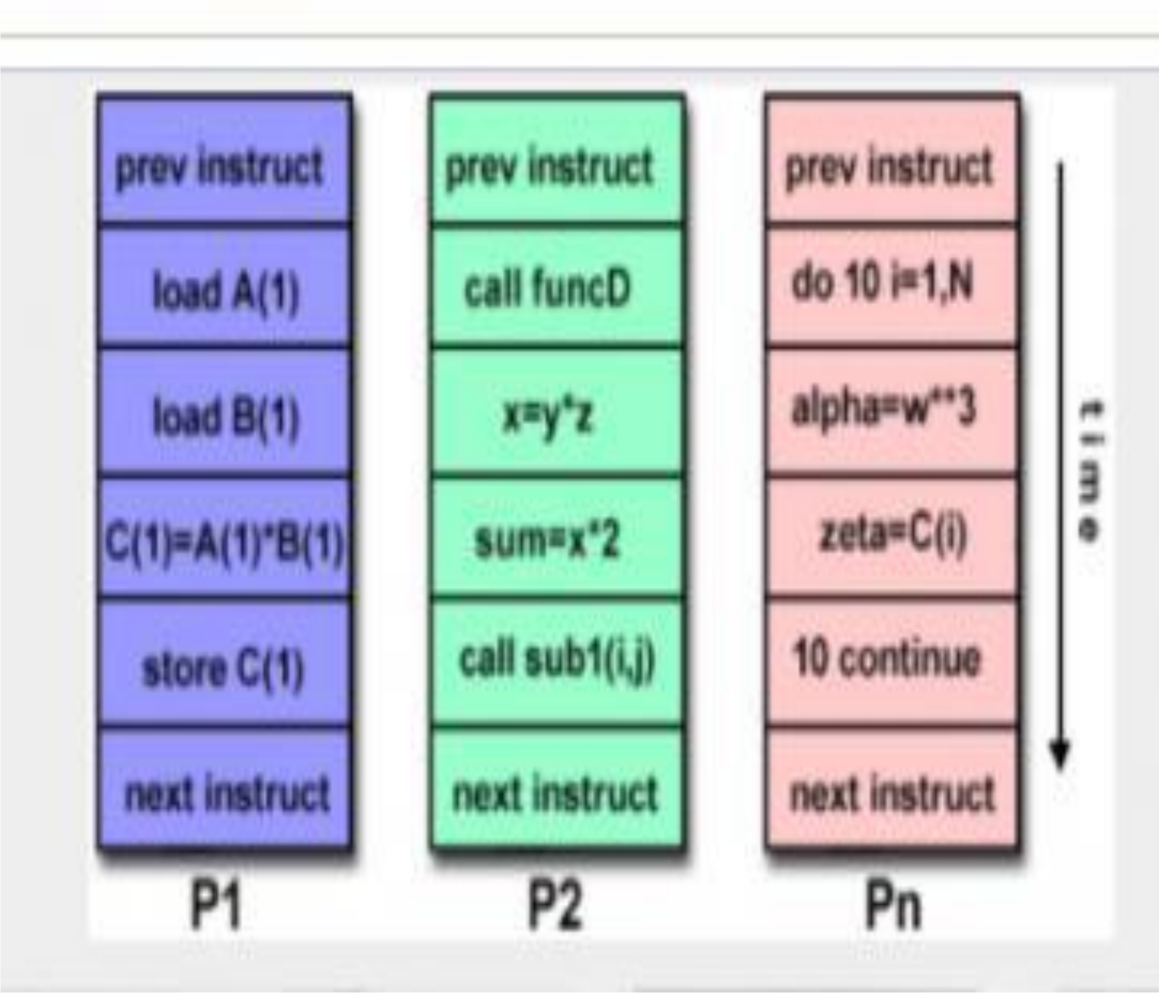

prev instruct
load A(1)
load B(1)
C(1)=A(1)*B(1)
store C(1)
next instruct
P1
prev instruct
load A(2)
load B(2)
C(2)=A(2)*B(2)
store C(2)
next instruct
P2
prev instruct
load A(n)
load B(n)
C(n)=A(n)*B(n)
store C(n)
next instruct
Pn
time
X
+
Y
X + Y
=
X [ ]
x3
x2
x1
x0
+
Y [ ]
y3
y2
y1
y0
X [ ] + Y [ ]
x3+y3
x2+y2
x1+y1
x0 + y0

MIMD

- type of parallel computer
- 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 non-deterministic
- Is > 1 Ds > 1
- Currently, the most common type of parallel computer - most modern super-computers fall into this category.
- Examples: most current supercomputers

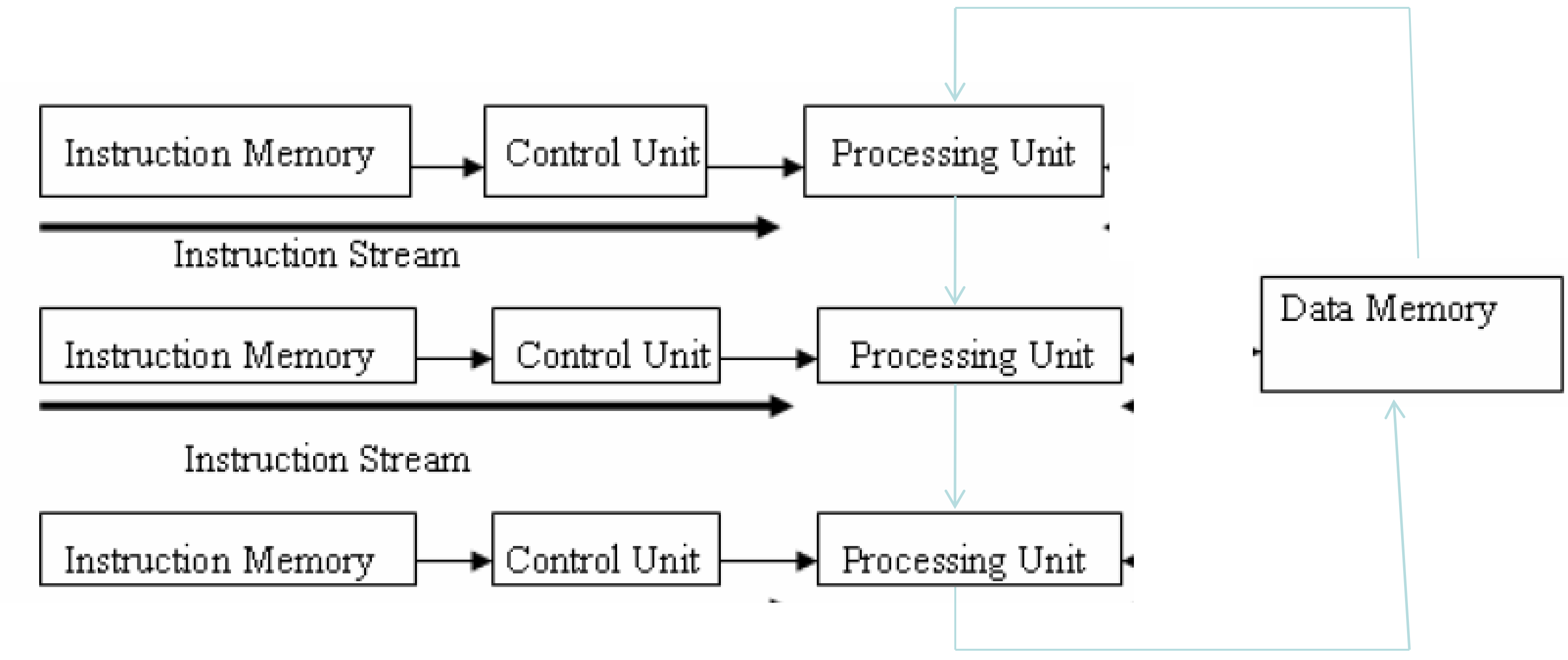


MIMD

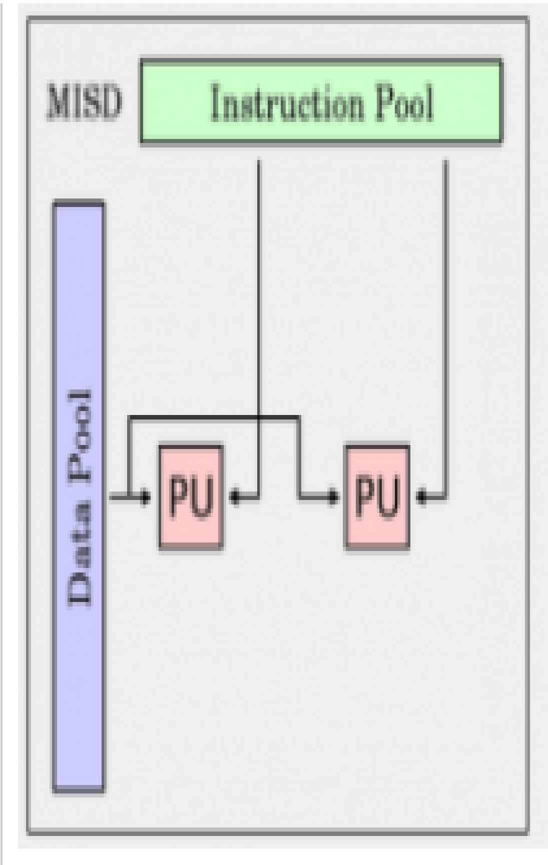

prev instruct
load A(1)
load B(1)
C(1)=A(1)*B(1)
store C(1)
next instruct
prev instruct
call funcD
x=y*z
sum=x*2
call sub1(i,j)
next instruct
prev instruct
do 10 i=1,N
alpha=w**3
zeta=C(i)
10 continue
next instruct
time
P1
P2
Pn

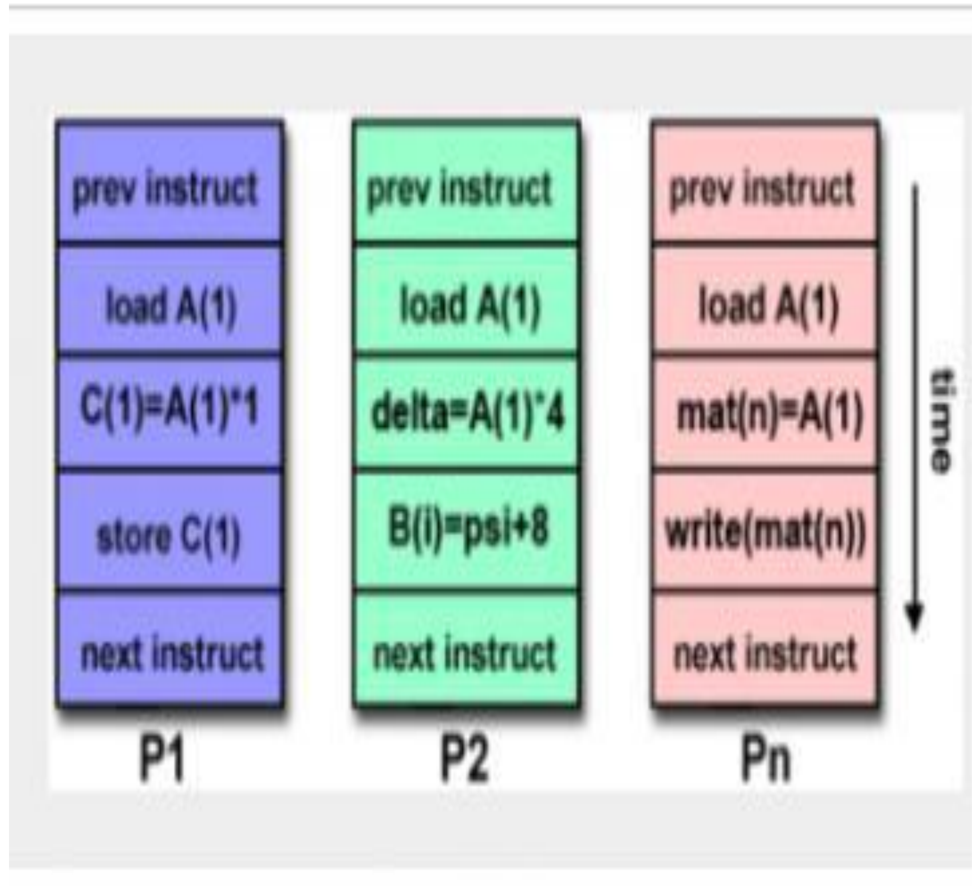
MISD

- Not commonly seen.
- Is > 1 Ds = 1
- Systolic array is one example of an MISD architecture.



MISD


MISD
Instruction Pool
Data Pool
PU
PU


prev instruct
load A(1)
C(1)=A(1)*1
store C(1)
next instruct
P1
prev instruct
load A(1)
delta=A(1)*4
B(i)=psi+8
next instruct
P2
prev instruct
load A(1)
mat(n)=A(1)
write(mat(n))
next instruct
Pn
time

State whether True or False for the following:

a) SISD computers can be characterized as $Is > 1$ and $Ds > 1$.
b) SIMD computers can be characterized as $Is > 1$ and $Ds = 1$
c) MISD computers can be characterized as $Is = 1$ and $Ds = 1$
d) MIMD computers can be characterized as $Is > 1$ and $Ds > 1$

Multithreading

Threads

- Threads are lightweight processes as the overhead of switching between threads is less
- Threads can be easily spawned

Why do we need threads?

- To enhance parallel processing
- To increase response to the user
- To utilize the idle time of the CPU
- Prioritize your work depending on priority

Example

- Consider a simple web server
- The web server listens for request and serves it
- If the web server was not multithreaded, the requests processing would be in a queue, thus increasing the response time and also might hang the server if there was a bad request.
- By implementing in a multithreaded environment, the web server can serve multiple request simultaneously thus improving response time

Multithreading on A Chip

- Multithreading – increase the **utilization of resources** on a chip by allowing multiple processes (threads) to share the **functional units of a single processor.**

- Processor must duplicate the state hardware for each thread – a separate **register file, PC, instruction buffer, and store buffer** for each thread
- The **caches, buffers** can be shared
- The **memory can be shared** through virtual memory mechanisms
- Hardware must support **efficient thread context switching**

Types of Multithreading

- **Fine-grained (Interleaved multithreading)**

  Cycle by cycle

- **Coarse-grained (Blocked multithreading)**

  Switch on event (e.g., cache miss)

  Switch on quantum/timeout

- **Simultaneous multithreading (SMT):**

  Instructions from multiple threads executed concurrently in the cycle

Types of Multithreading

- Fine-grain – switch threads on every instruction issue
  - Round-robin thread interleaving (skipping stalled threads)
  - Processor must be able to switch threads on every clock cycle
  - Advantage – can hide throughput losses that come from both short and long stalls
  - Disadvantage – slows down the execution of an individual thread since a thread that is ready to execute without stalls is delayed by instructions from other threads

Types of Multithreading

- Coarse-grain – switches threads only on costly stalls (e.g., L2 cache misses)

    Benefits:

    Simple, improved throughput (~30%), low cost .

    Thread priorities mostly avoid single-thread slowdown

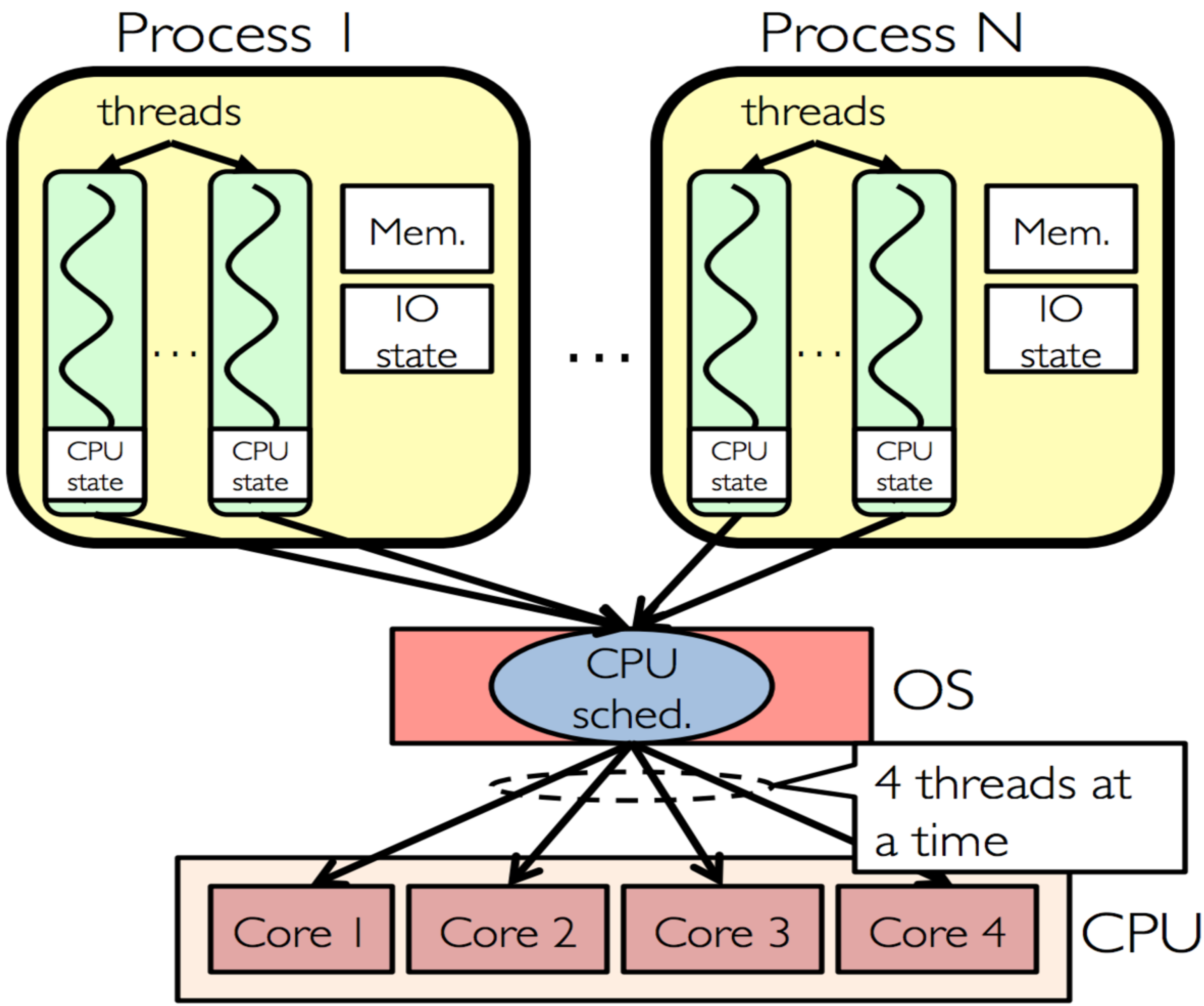
    Drawback:

        Nondeterministic, conflicts in shared caches

Simultaneous multithreading (SMT):

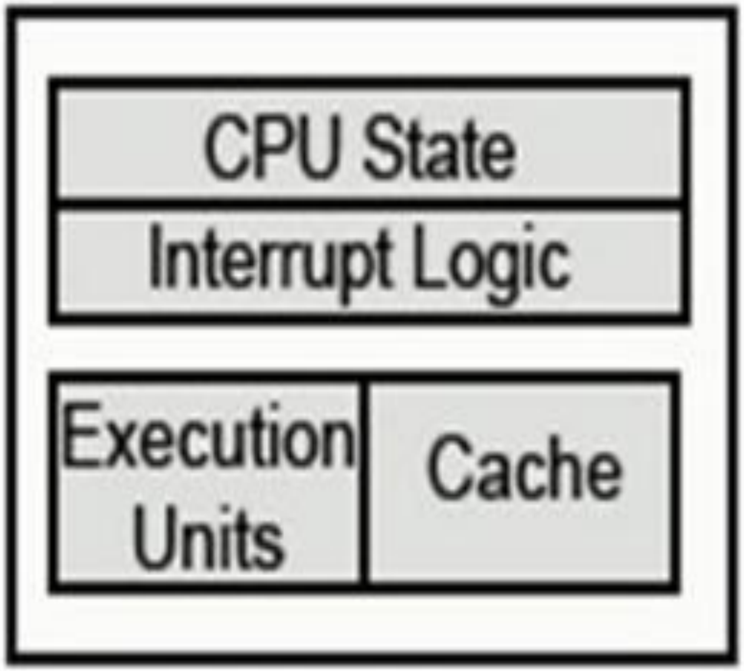
1. Instructions are simultaneously issued from multiple threads to the execution units of a superscalar processor.
2. This combines the wide superscalar instruction issue capability with the use of multiple thread contexts.

- Benefits:
  - Natural fit for superscalar
  - Improved throughput
  - Low incremental cost
- Drawbacks:
  - Additional complexity over superscalar
  - Cache conflicts



Process 1
Process N
threads
threads
Mem.
IO state
Mem.
IO state
CPU state
CPU state
CPU state
CPU state
CPU sched.
OS
4 threads at a time
Core 1
Core 2
Core 3
Core 4
CPU

Single Core



a) Single Core

Simultaneous **multithreading** (**SMT**)

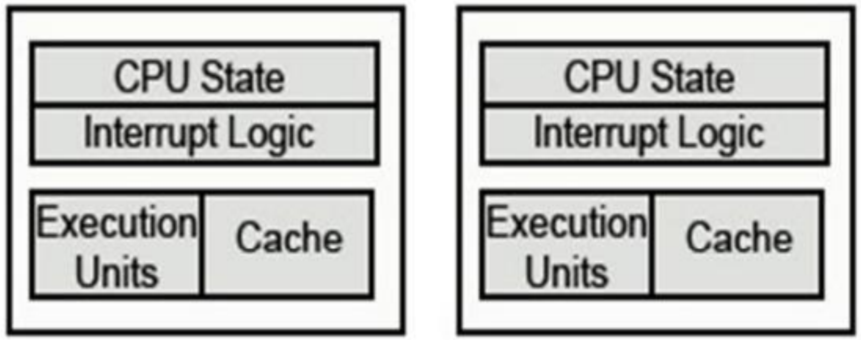
Parameter	Single-Core	Multi-Core
No of cores	One primary core	Two or more separate core
Processing	Sequential	Parallel
SMT	Not Possible	Possible
Power	Low	High
Speed	Slow	Fast
Efficiency	Low	High
Operation	One task at a time	Multitasking

Difference between MultiCore and MultiProcessor System

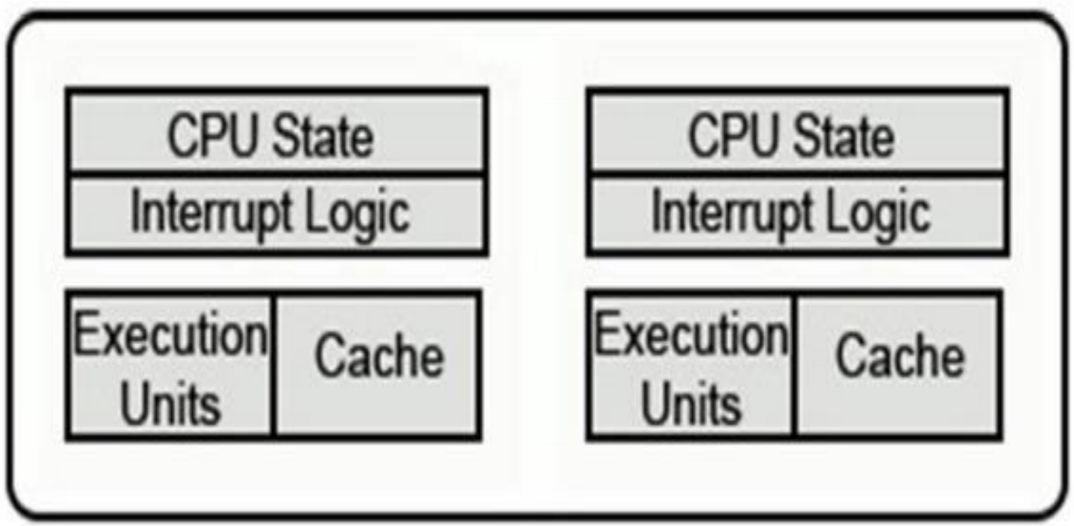
MultiCore	MultiProcessor
A single CPU or processor with two or more independent processing units called cores that are capable of reading and executing program instructions.	A system with two or more CPU’s that allows simultaneous processing of programs.
It executes single program faster.	It executes multiple programs Faster.
Not as reliable as multiprocessor.	More reliable since failure in one CPU will not affect other.

Difference between MultiCore and MultiProcessor System

MultiCore	MultiProcessor
It has less traffic.	It has more traffic.
It does not need to be configured.	It needs little complex configuration.
It’s very cheaper (single CPU that does not require multiple CPU support system).	It is Expensive (Multiple separate CPU’s that require system that supports multiple processors) as compared to MultiCore.



b) Multiprocessor



c) Multi-core