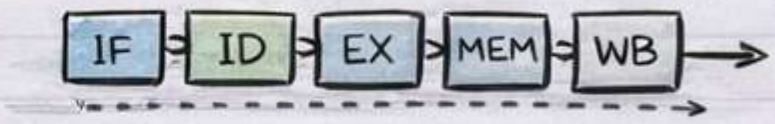
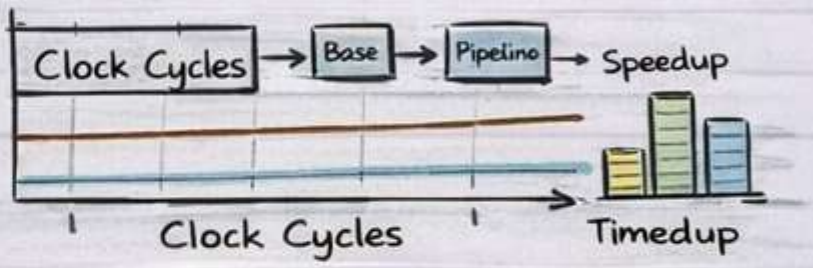
COA
Module-3 Notes

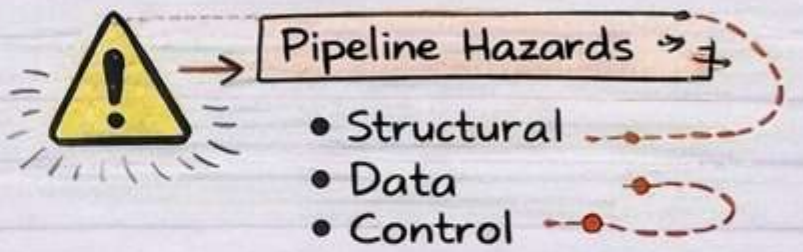
by pyqfort.com

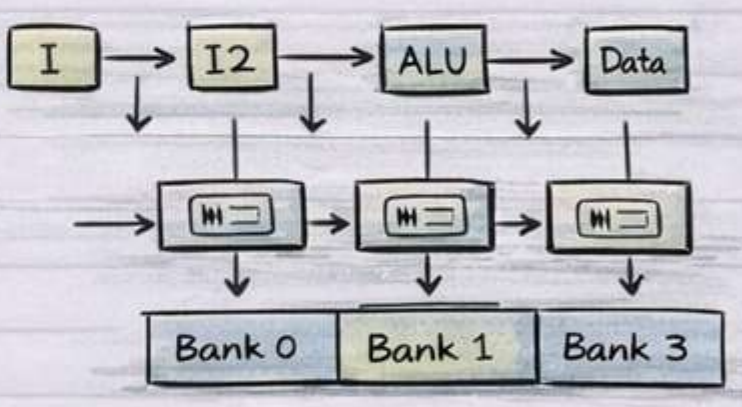
Contents Covered:

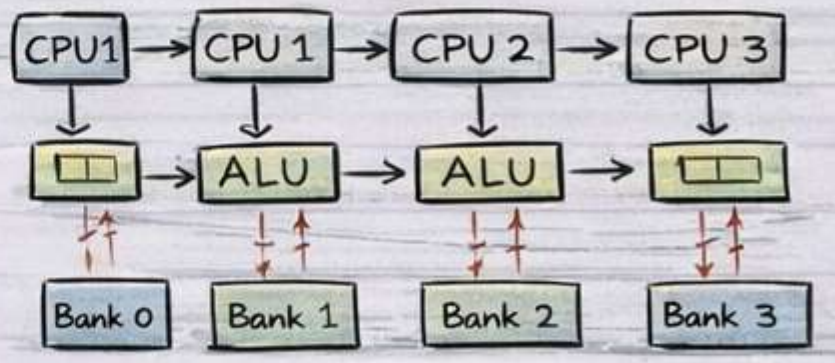
- Intro to Pipelining
- Throughput and Speedup
- Pipeline Hazards
- Intro to Parallel Processing
- SISD
- SIMD
- MISD
- MIMD
- Concurrent Access to Memory
- Cache Coherency

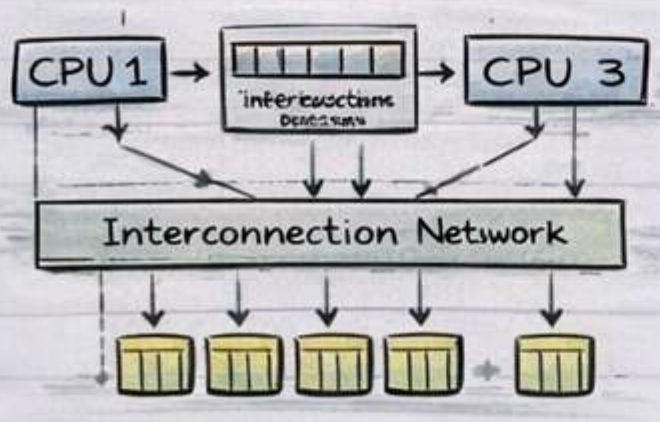


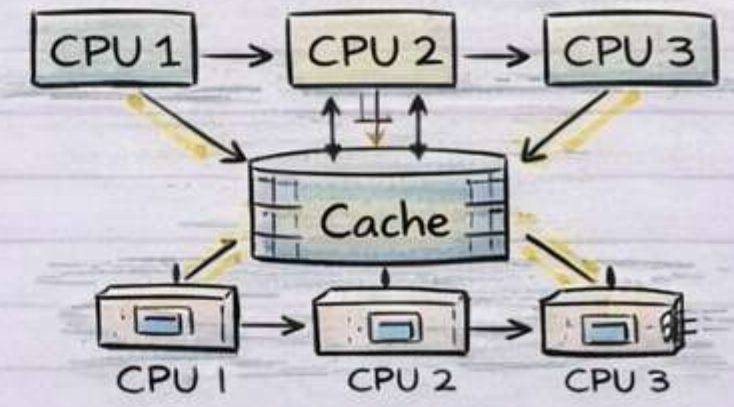






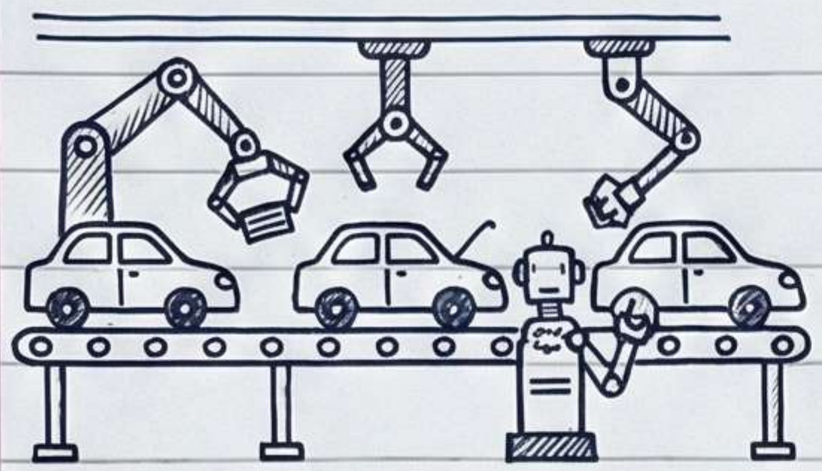






Module-3: Pipelining

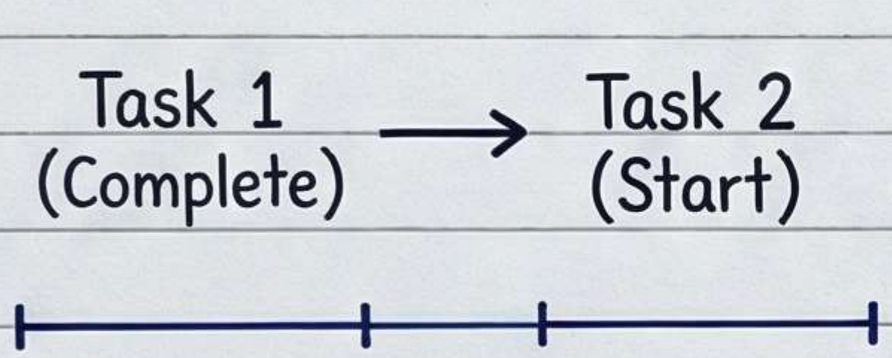
Basic Concept



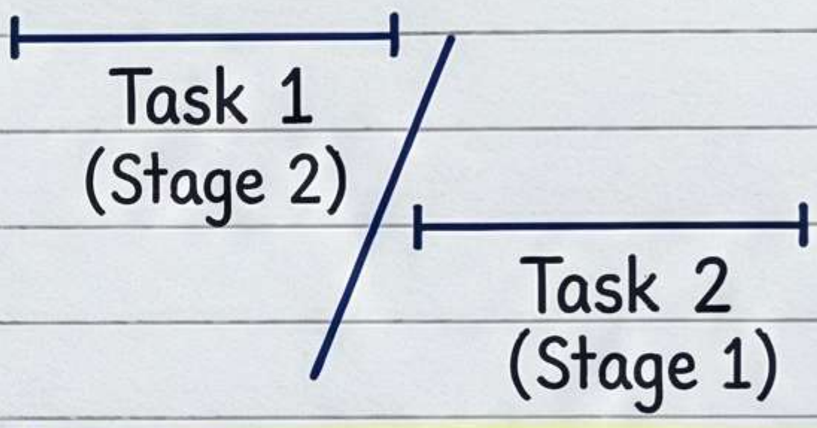
Think of an ASSEMBLY LINE for cars. Different stations (stages) work on different cars at the same time.

Pipelining divides instruction execution into multiple STAGES (segments) for concurrent processing.

Execution Types



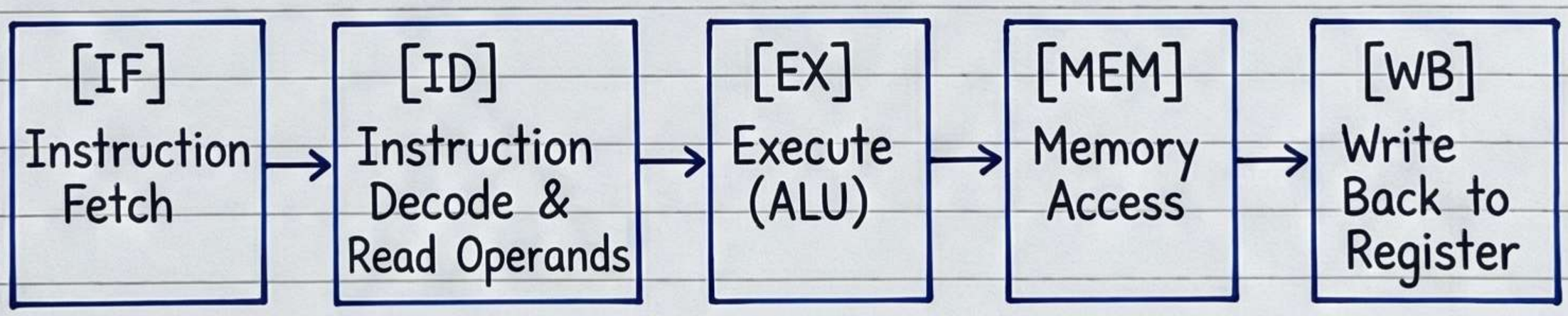
Non-Overlapped

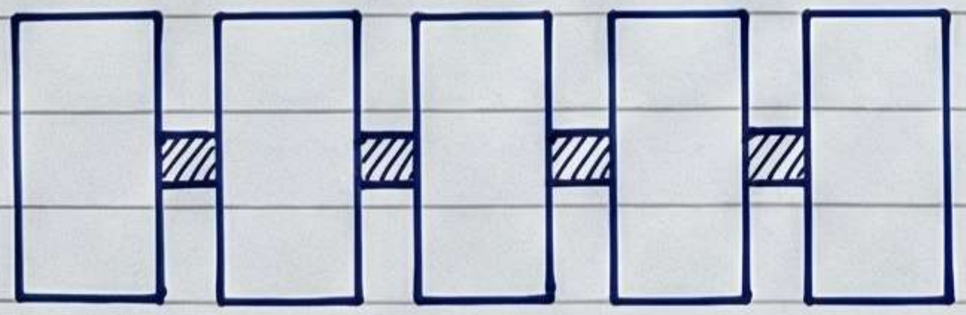


Overlapped (Pipelining)

↳ Higher THROUGHPUT!

The 5-Stage Pipeline





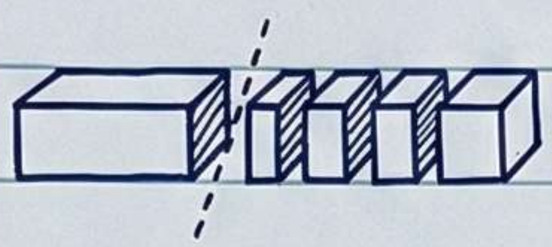
REGISTERS between stages hold intermediate results.

Characteristics of Pipeline

1. **Overlapping Operations:**

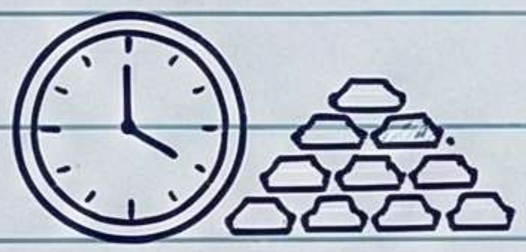
Different stages of multiple instructions are processed SIMULTANEOUSLY. Like a multi-lane road.

2. **Multiple Stages:**



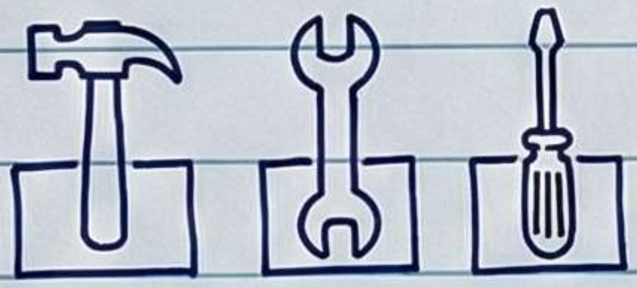
An instruction's execution is broken down into a sequence of smaller, INDEPENDENT stages.

3. **Increased Throughput:**



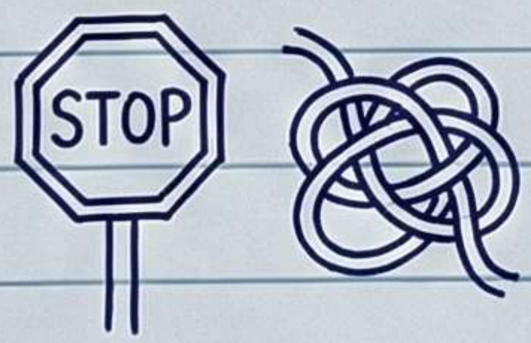
MORE instructions can be completed per unit of time (though single instruction latency might slightly increase).

4. **Specialized Hardware:**



Each stage often has DEDICATED hardware for its specific task.

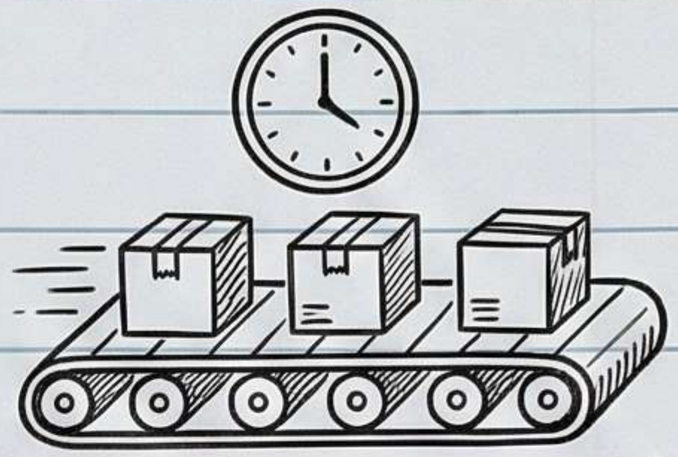
5. **Potential Hazards:**



Issues like DATA dependencies, CONTROL dependencies (branches), and STRUCTURAL conflicts can STALL the pipeline and reduce efficiency.

'Throughput and Speedup'

Throughput:

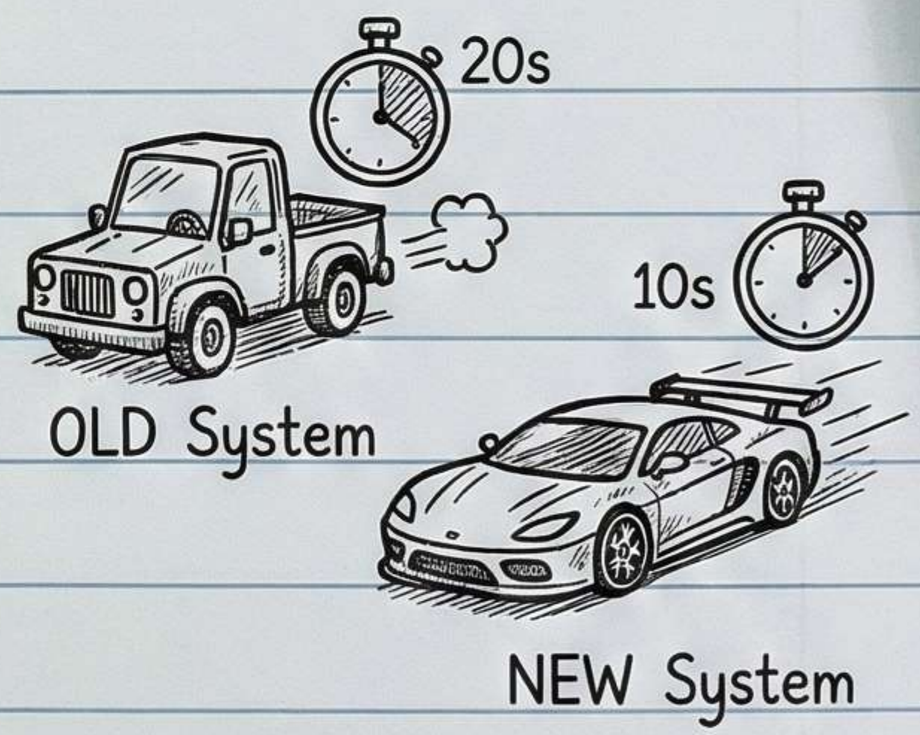


This measures how much WORK a system can do in a given amount of TIME.

Formula: Throughput = $\frac{\text{Amount of work done}}{\text{Time taken}}$

Example: If a system processes 1000 tasks in 10 seconds, Throughput = 100 tasks/second.

Speedup:



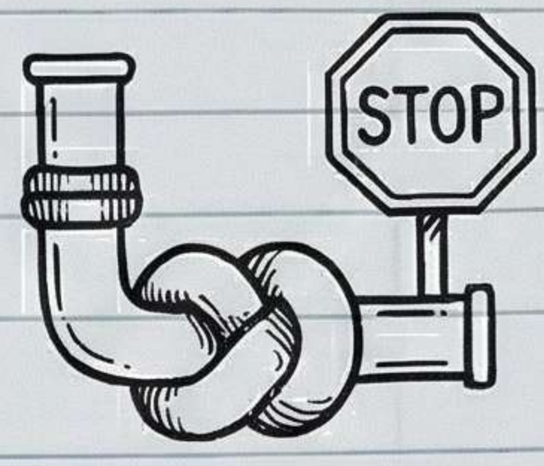
This measures how much FASTER a NEW system is compared to an OLD system for the same task.

Formula: Speedup = $\frac{\text{Time taken by old system}}{\text{Time taken by new system}}$

Example: If old system took 20 seconds and new system takes 10 seconds, Speedup = 20 / 10 = 2 (meaning it's 2 times faster).

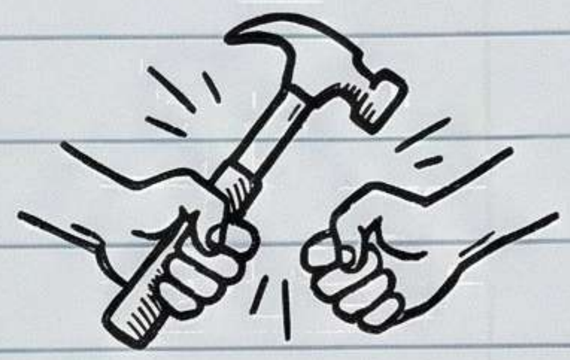
Pipeline Hazards

Conditions that OBSTRUCT the smooth execution flow, preventing the next instruction from executing in its designated clock cycle.



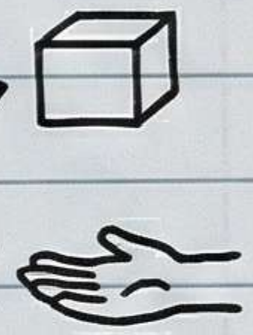
Arise from CONFLICTS and DEPENDENCIES between overlapping instructions.

1. Structural Hazards:



Two instructions need the SAME HARDWARE RESOURCE at the same time (e.g., single memory access unit).
Causes RESOURCE CONTENTION, requires STALLING.

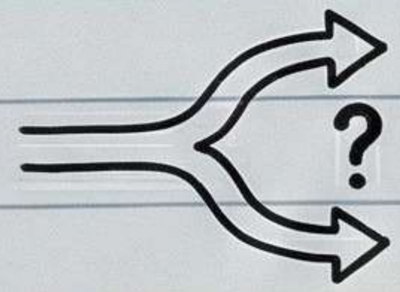
2. Data Hazards:



An instruction DEPENDS on the result of a PREVIOUS, incomplete instruction.
Using STALE DATA leads to errors.

- Types: RAW (Read After Write), WAW (Write After Write), WAR (Write After Read).
- Solutions: FORWARDING (bypassing) or pipeline STALLS.

3. Control Hazards (Branch Hazards):

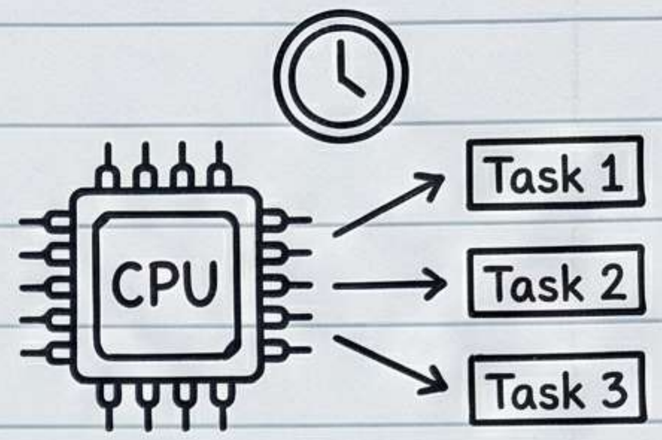


Caused by BRANCH instructions that ALTER the flow. Outcome/target isn't known immediately.
Can fetch from the WRONG PATH, requiring a FLUSH (discarding instructions) and lost cycles.

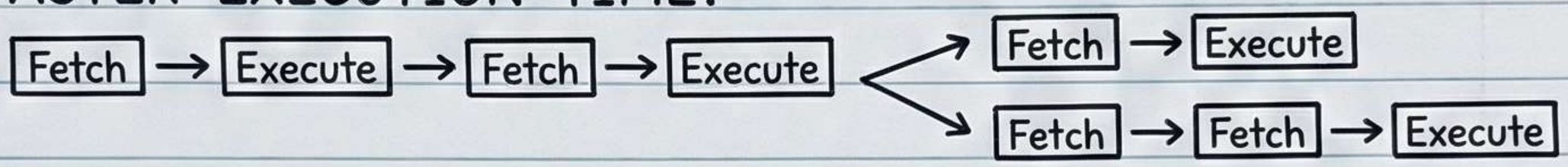
Solution: BRANCH PREDICTION.

Parallel Processors: Introduction

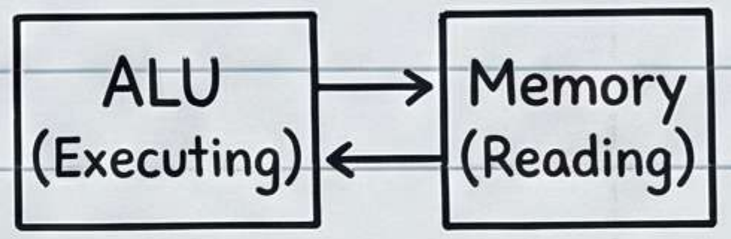
Definition:
Parallel processing is a term for techniques providing SIMULTANEOUS data-processing tasks to increase COMPUTATIONAL SPEED.



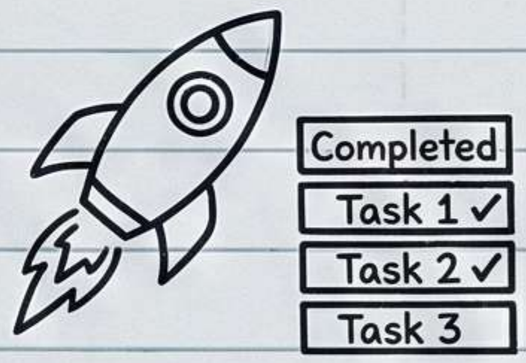
Concept: Instead of sequential processing, it performs CONCURRENT data processing to achieve FASTER EXECUTION TIME.



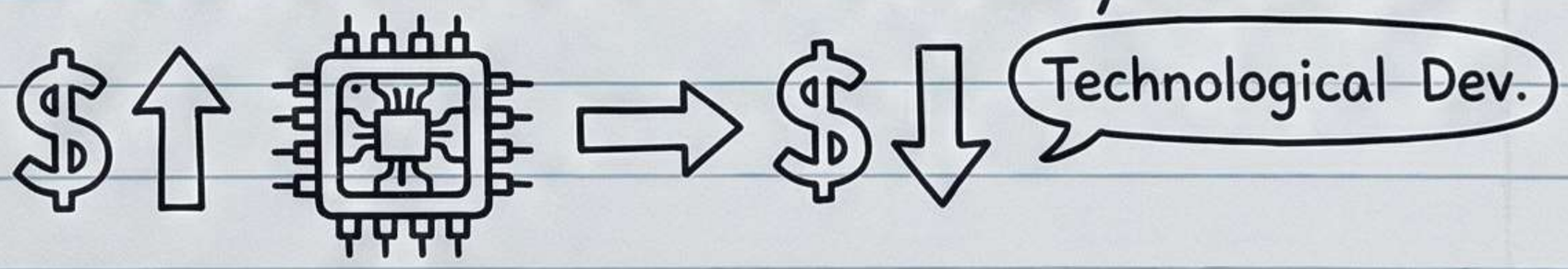
Example:
While one instruction is executed in the ALU, the NEXT instruction can be read from memory.



Purpose:
To SPEED UP computer processing capability and INCREASE THROUGHPUT (amount of processing in a given time).



Hardware & Cost:
Increases hardware and cost, but TECHNOLOGICAL DEVELOPMENTS have made it economically feasible.



Classification (Flynn):
Based on instruction and data streams: SISD, SIMD, MISD, MIMD.

SISD | SIMD
MISD | MIMD

SISD (Single Instruction stream, Single Data stream)

Definition:

In SISD, a single processor executes a single INSTRUCTION STREAM on a single DATA STREAM. It's a SEQUENTIAL processing model, meaning the processor performs ONE operation at a time.

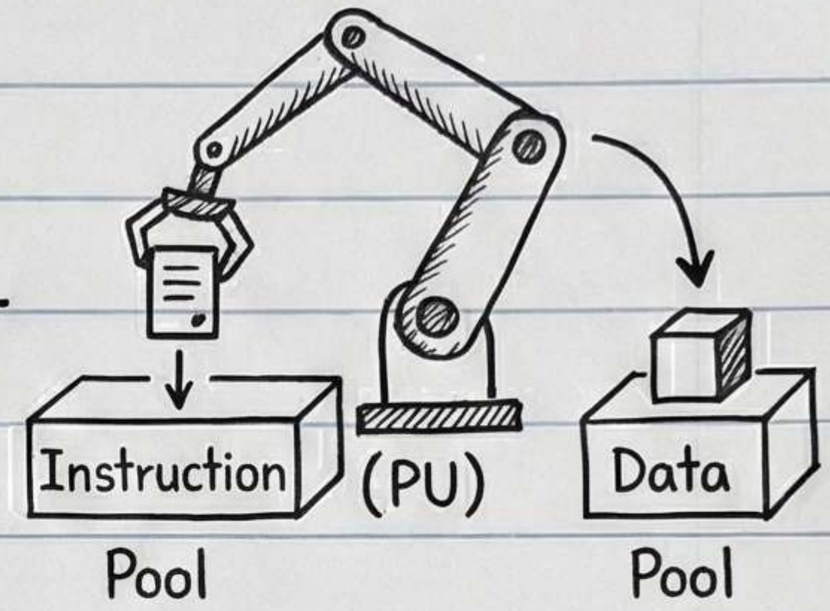
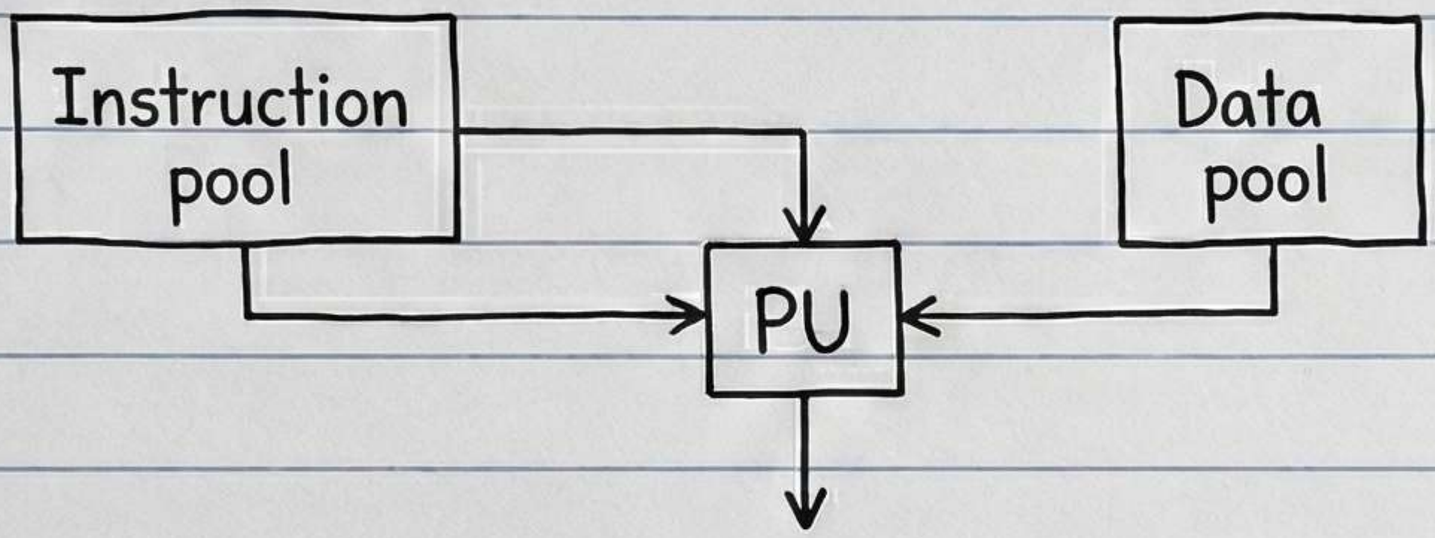
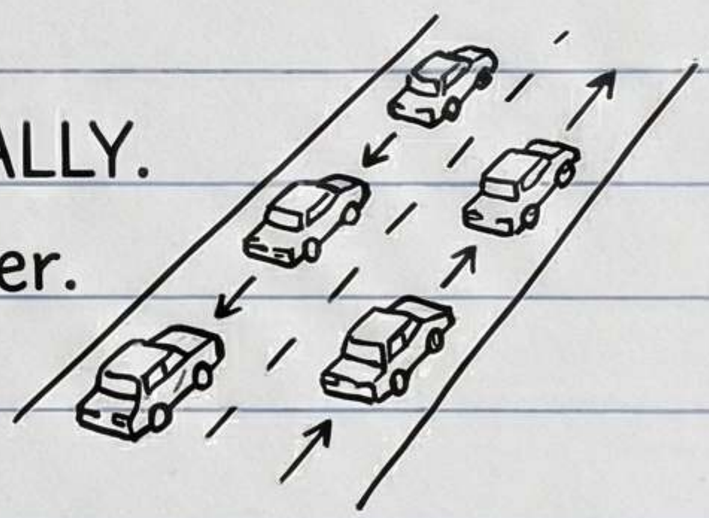


Diagram:



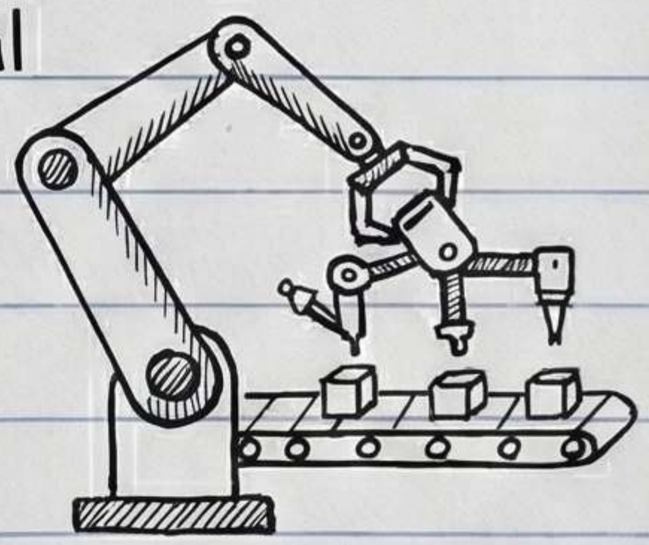
Execution:

Instructions are executed SEQUENTIALLY.
A single lane moving one after another.



Parallelism in SISD:

The system may or may not have internal parallel processing capabilities.
Parallel processing can be achieved by MULTIPLE FUNCTIONAL UNITS or by PIPELINE processing.



SIMD (Single Instruction stream, Multiple Data streams)

Definition: In SIMD, a single INSTRUCTION STREAM is applied to MULTIPLE DATA STREAMS simultaneously. This means the SAME INSTRUCTION is executed on multiple data points at once. It is a type of PARALLEL COMPUTING.

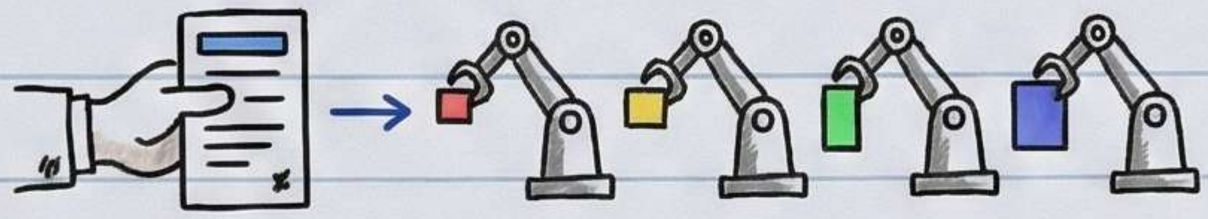
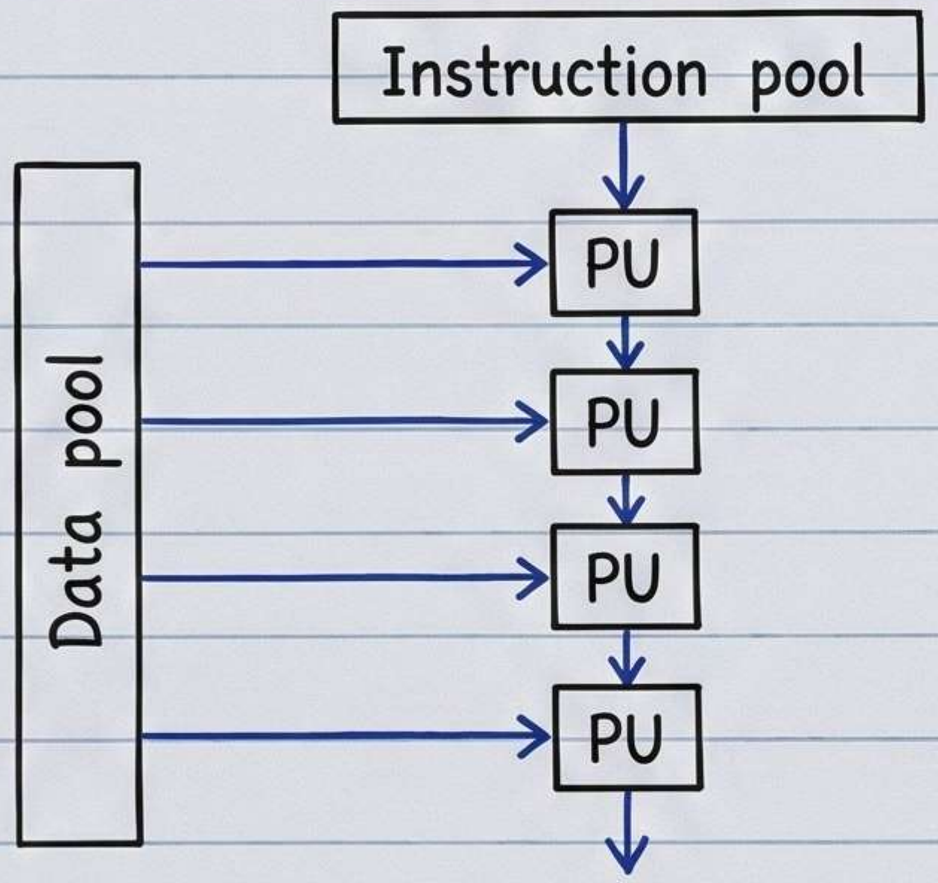
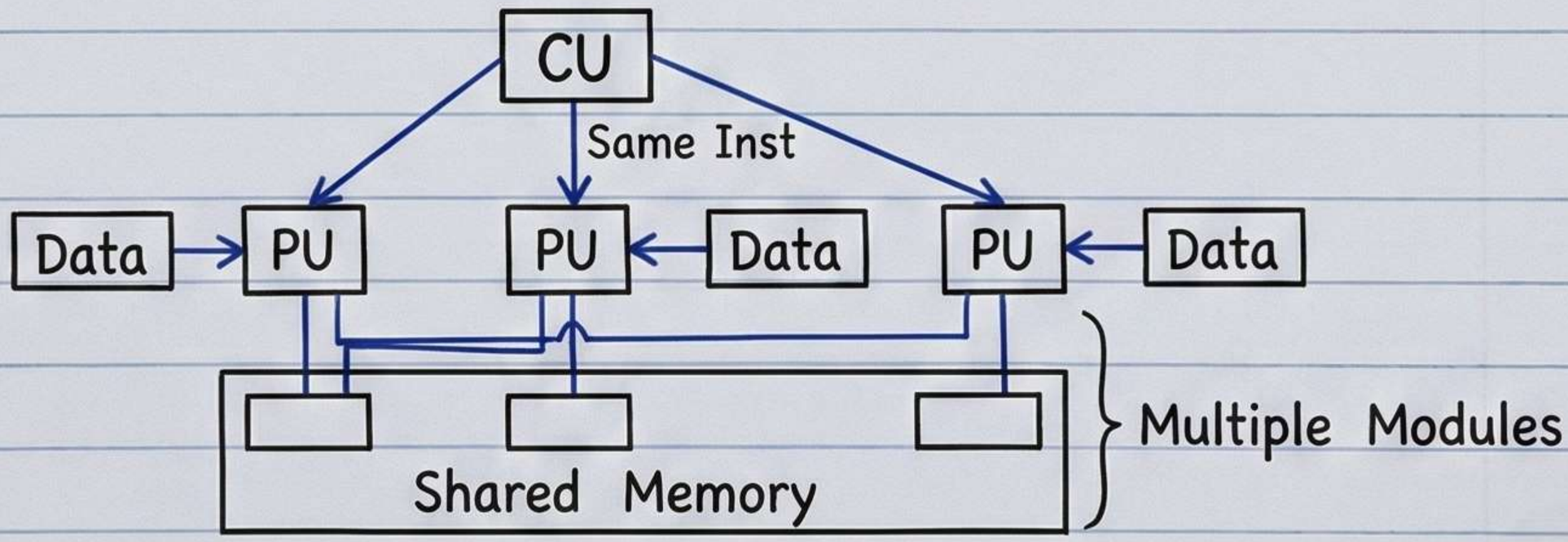


Diagram:



Key Characteristics:

All processors receive the SAME INSTRUCTION from the control unit but operate on DIFFERENT ITEMS OF DATA. The SHARED MEMORY unit must contain MULTIPLE MODULES to communicate with all processors simultaneously.



MISD (Multiple Instruction streams, Single Data stream)

Definition:

In MISD, MULTIPLE INSTRUCTION STREAMS operate on the SAME DATA STREAM simultaneously. This means different operations are performed on the same data.

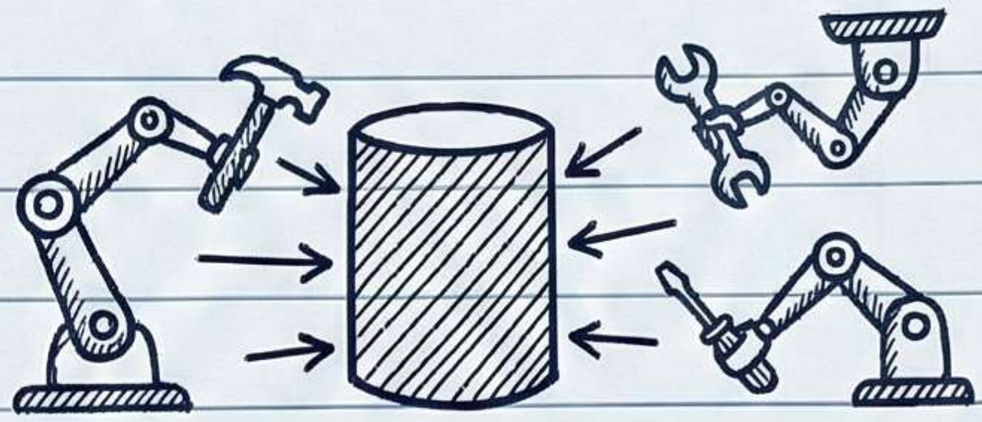
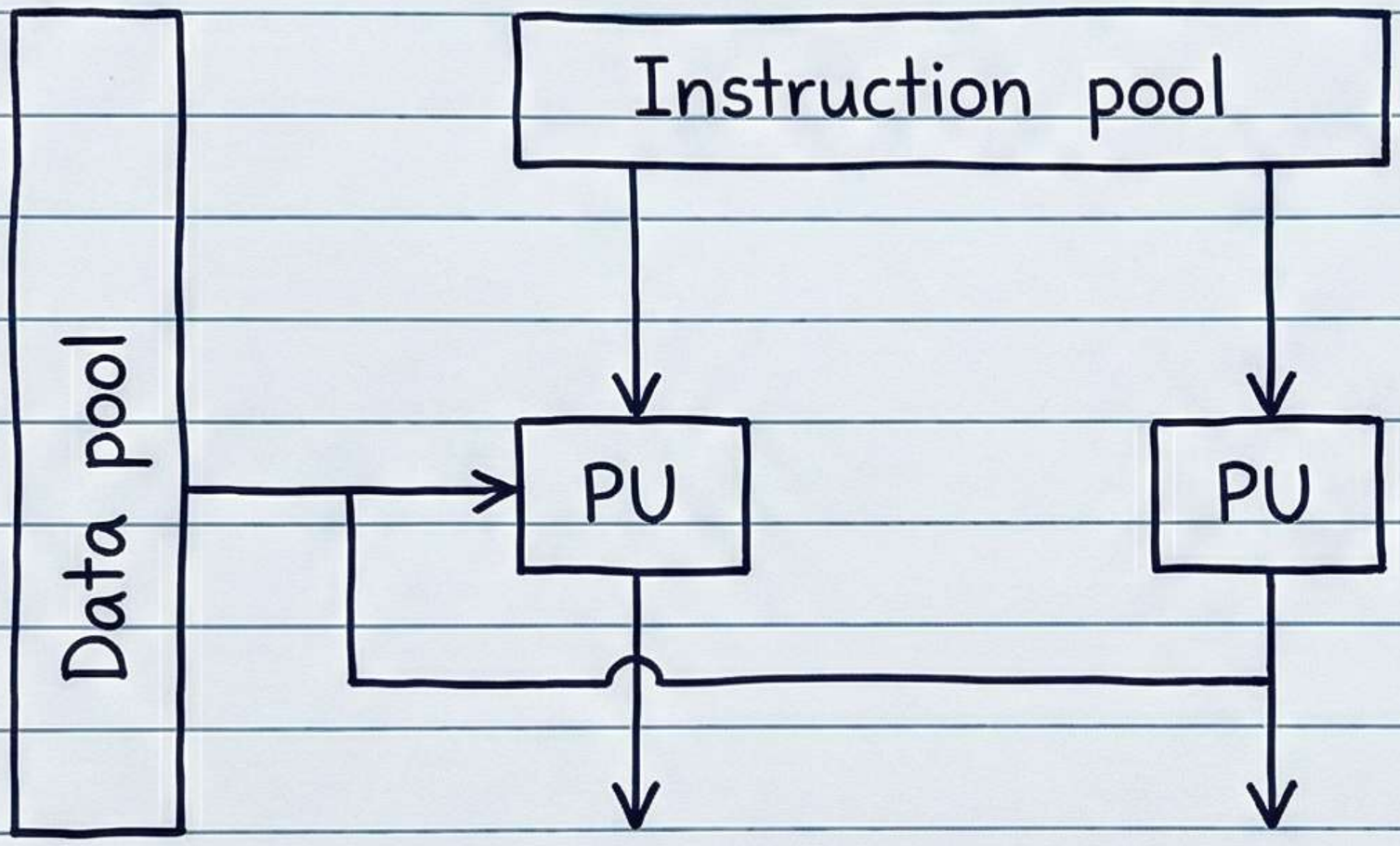


Diagram:



Key Characteristics:

This architecture is quite RARE and not commonly used in practical systems. It's mostly THEORETICAL or used in specialized systems like FAULT-TOLERANT COMPUTING (e.g., for redundancy and error checking).

MIMD (Multiple Instruction streams, Multiple Data streams)

Definition:
In MIMD, multiple processors execute DIFFERENT INSTRUCTION STREAMS on DIFFERENT DATA STREAMS simultaneously. This is the most GENERAL and FLEXIBLE form of parallel processing, where each processor can perform DIFFERENT TASKS simultaneously.

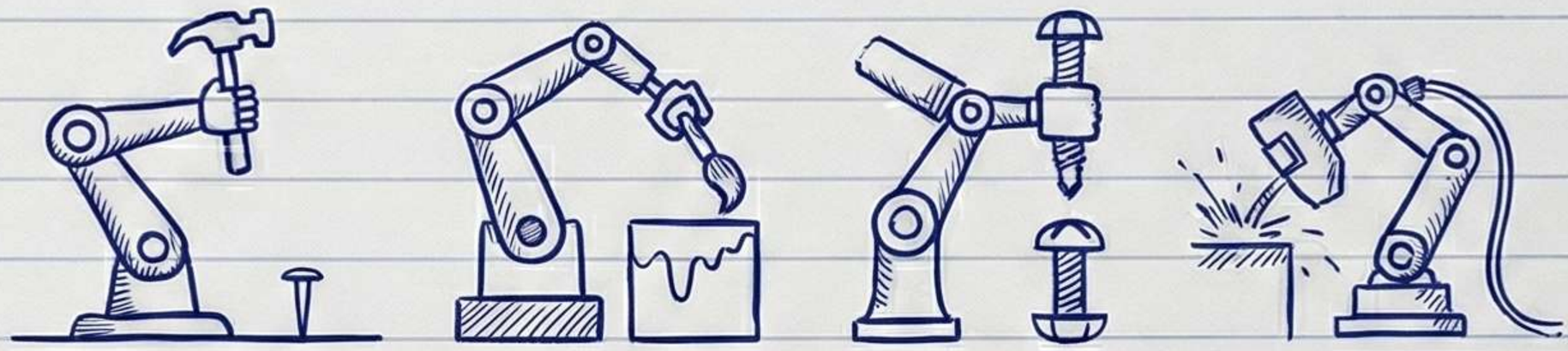
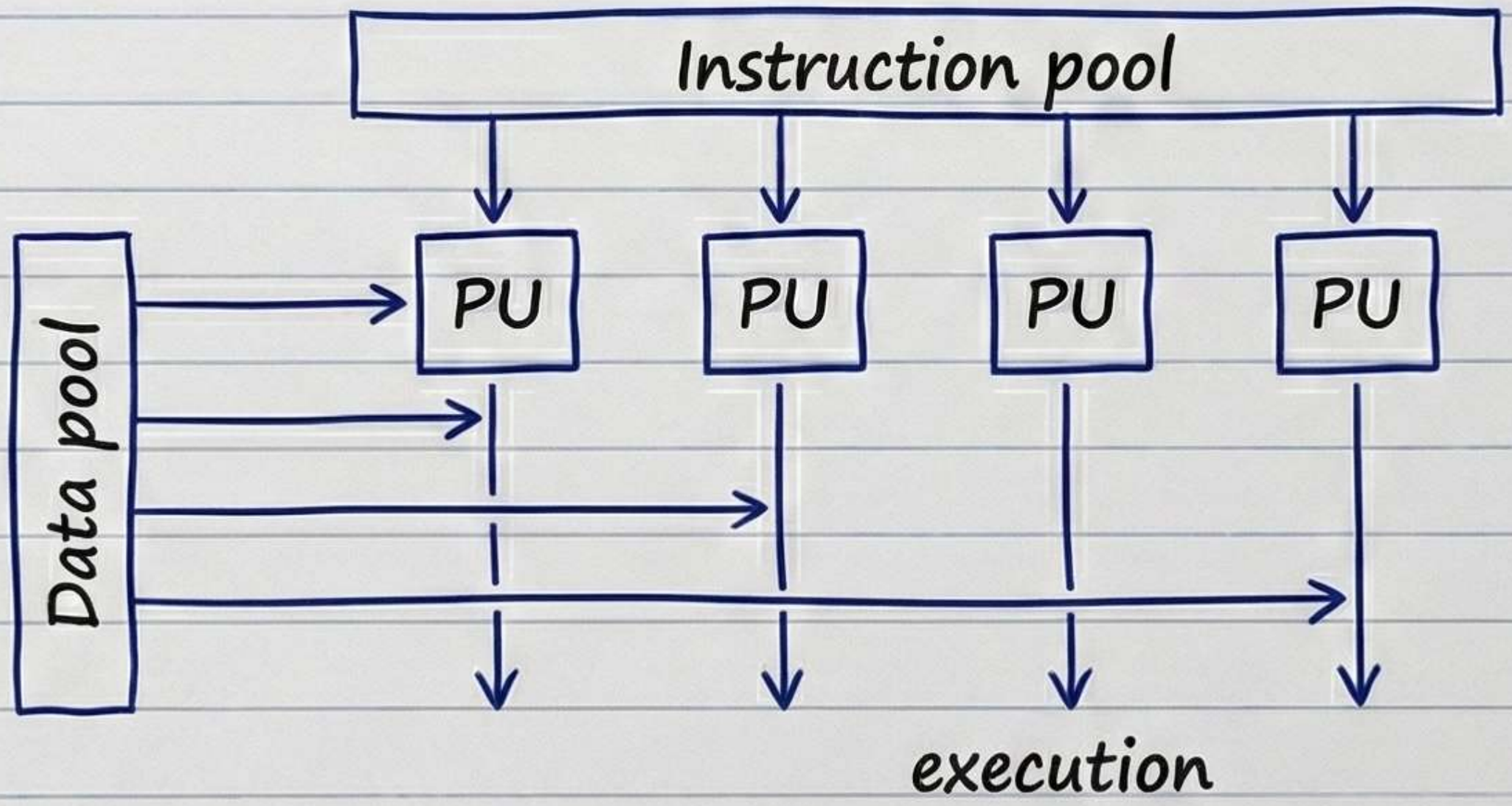
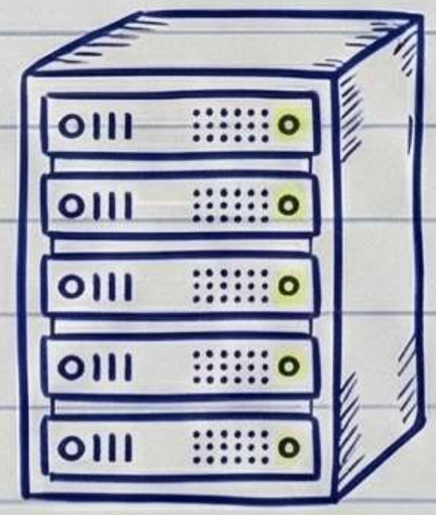


Diagram:



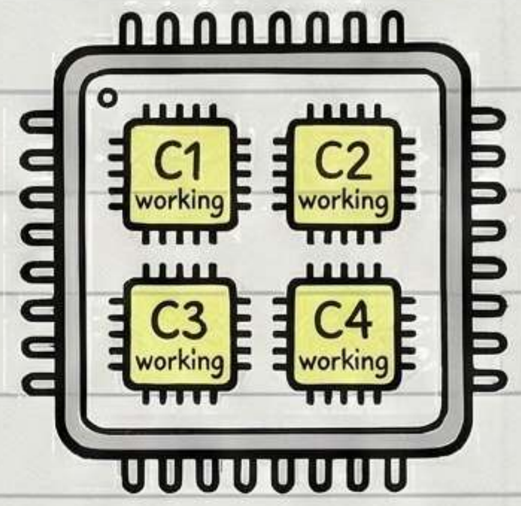
Key Characteristics:
Most MULTIPROCESSOR and MULTI-COMPUTER SYSTEMS can be classified in this category.



Concurrent Access to Memory

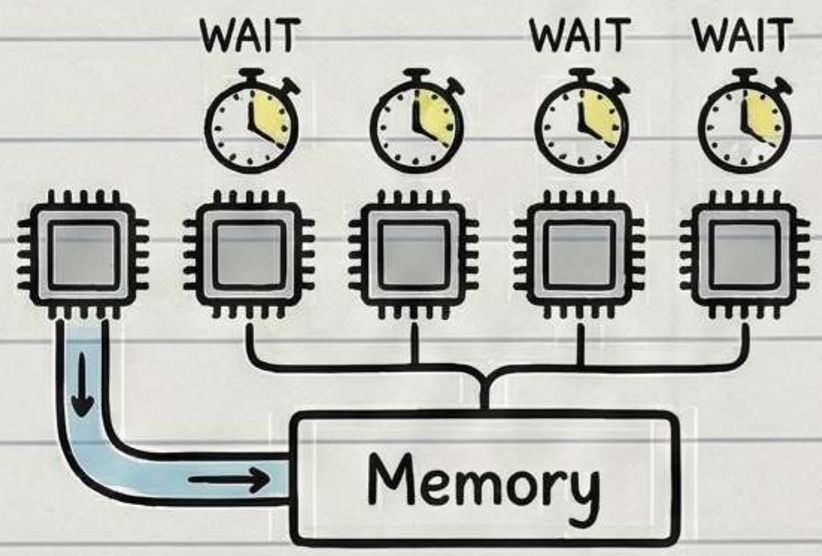
The Need:

Parallel processors have MULTIPLE PROCESSING UNITS (CORES) working at the same time, often on different tasks.



The Problem (Bottleneck):

If processors can't access memory CONCURRENTLY, they must WAIT for each other, creating a BOTTLENECK.

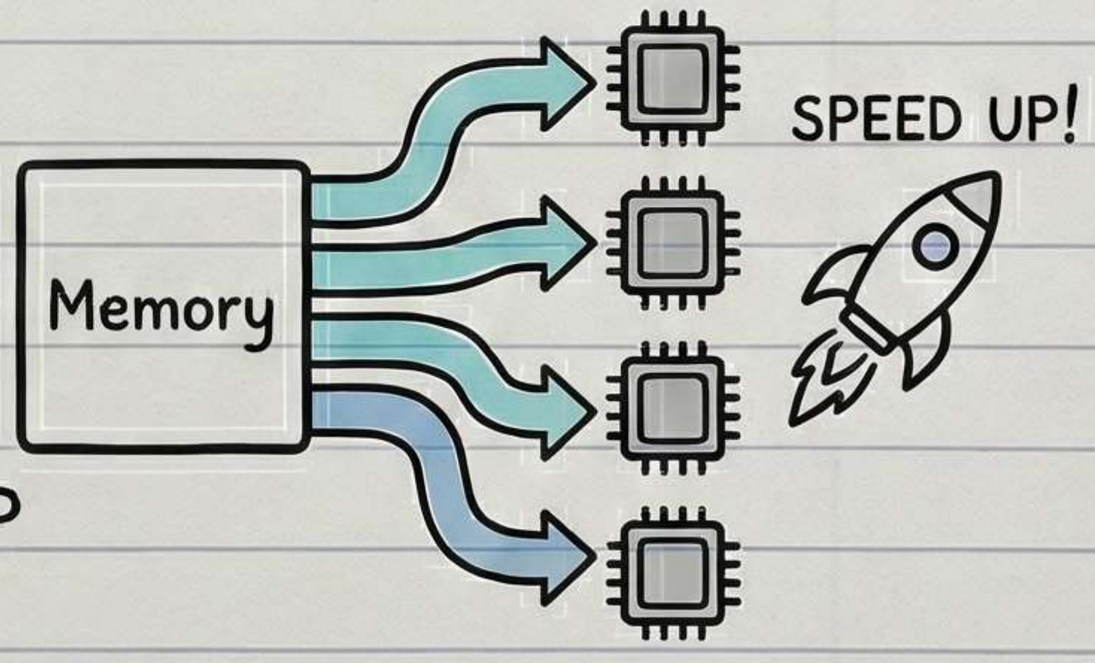


Analogy:

Like many WORKERS sharing a SINGLE TOOLBOX and taking turns.

The Solution:

Concurrent memory access allows multiple processors to READ and WRITE data SIMULTANEOUSLY.
This is crucial for TRUE PARALLELISM and SPEEDING UP computations.



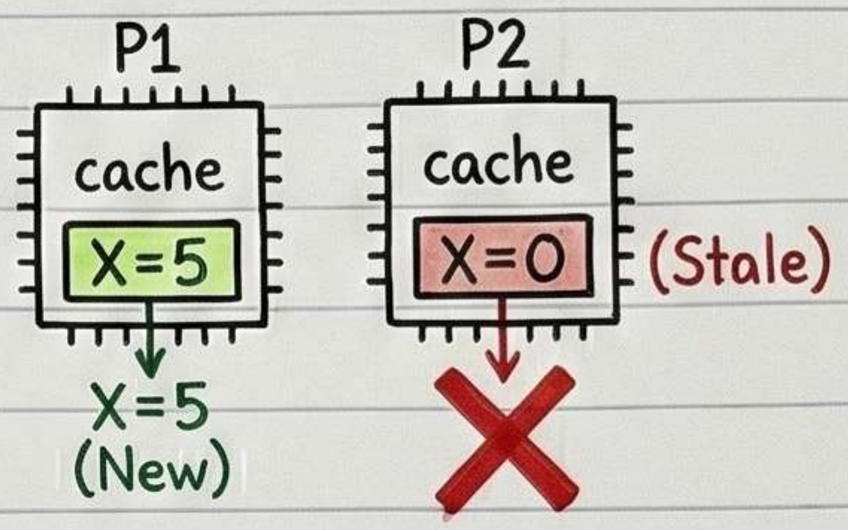
Impact:

Without it, the benefit of multiple processors is greatly DIMINISHED.

Cache Coherency

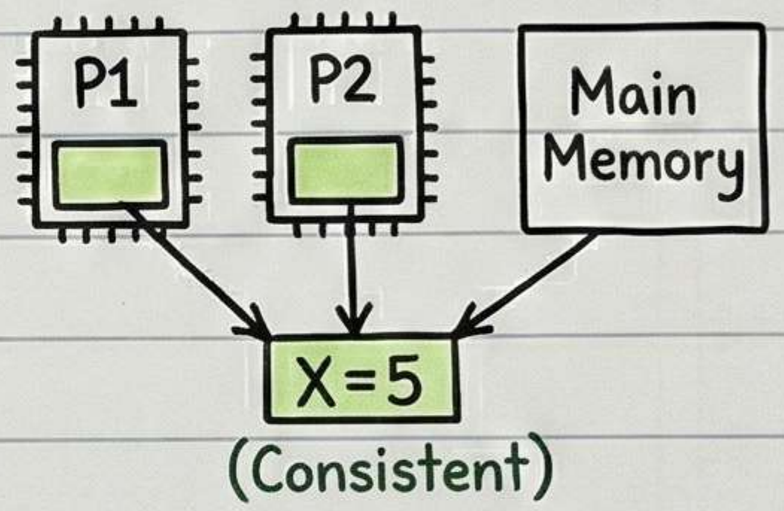
The Problem:

In a multi-processor system, each processor has its own local cache. If P1 updates a variable 'X' in its cache, P2's cache might still have the OLD, STALE value.



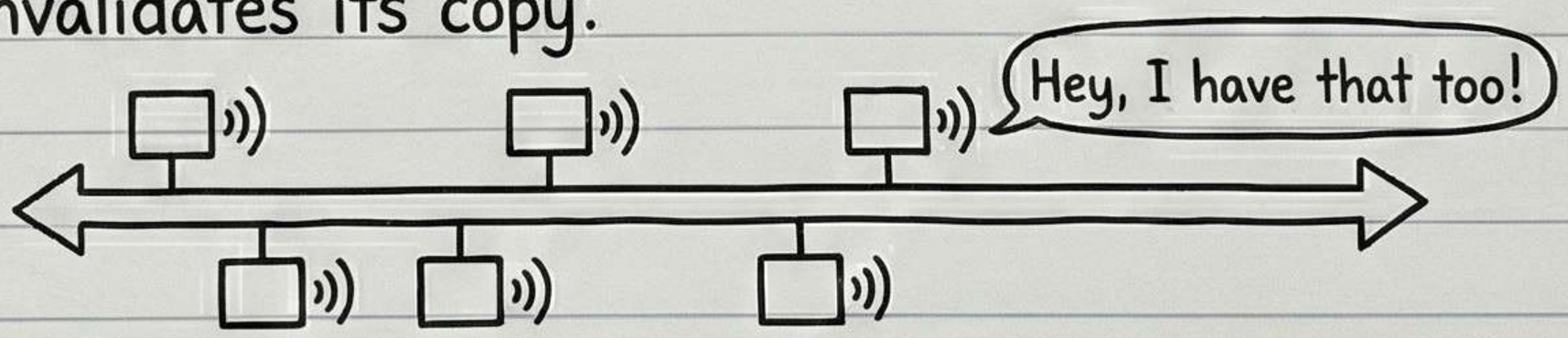
The Solution (Goal):

Mechanisms are needed to ensure all caches and main memory see a CONSISTENT view of data



Two Main Approaches:

1. **Snooping Protocols:**
   Each cache "SNOOPS" (monitors) the system bus. If it sees a write to data it holds, it updates or invalidates its copy.



2. **Directory-Based Protocols:**
   A central "DIRECTORY" keeps track of who has which memory blocks. Processors check with the directory before writing. Scales better for many processors.