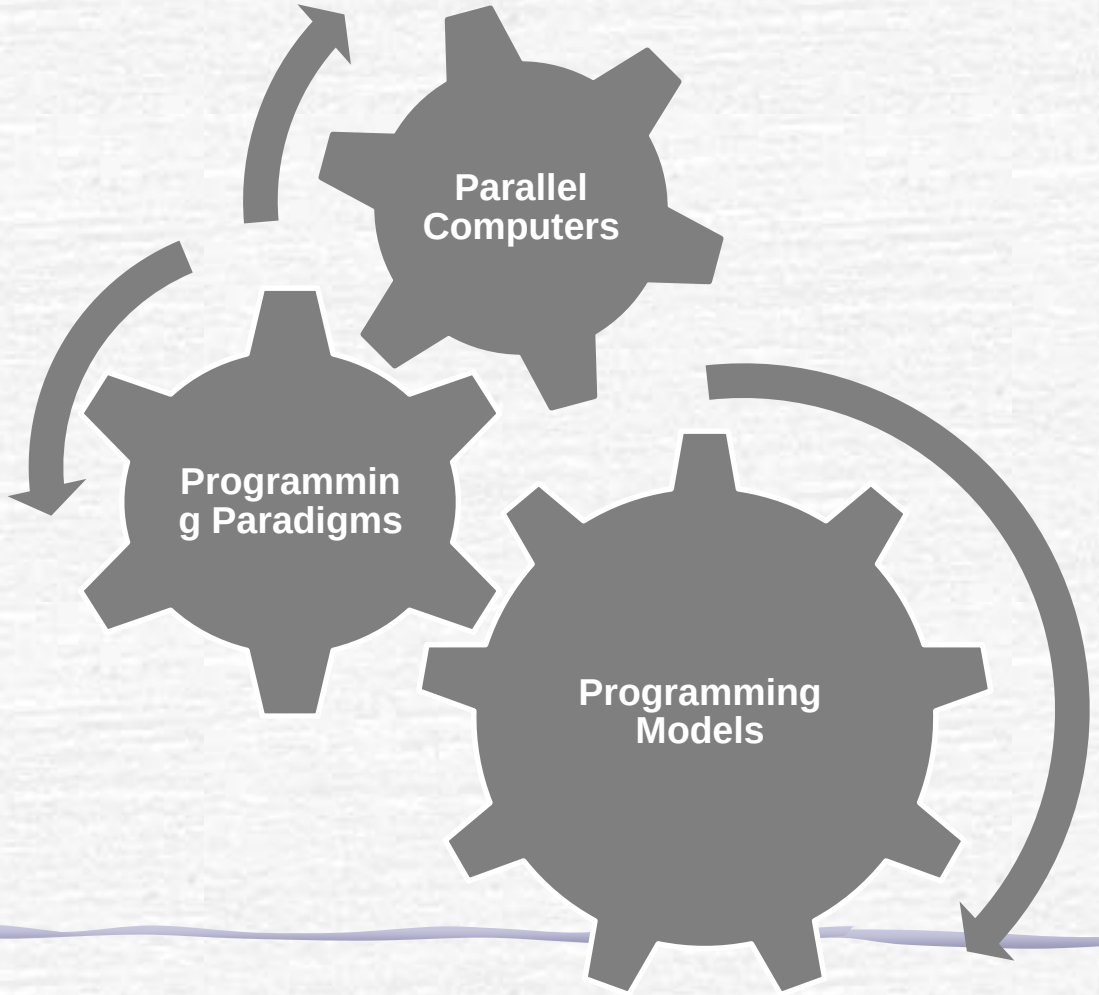
Department of Computer Science

Understanding
Parallel Computers - Paradigms and Programming Models

Sofien GANNOUNI
Computer Science
E-mail: gnnosf@ksu.edu.sa ; gansof@yahoo.com

Introduction

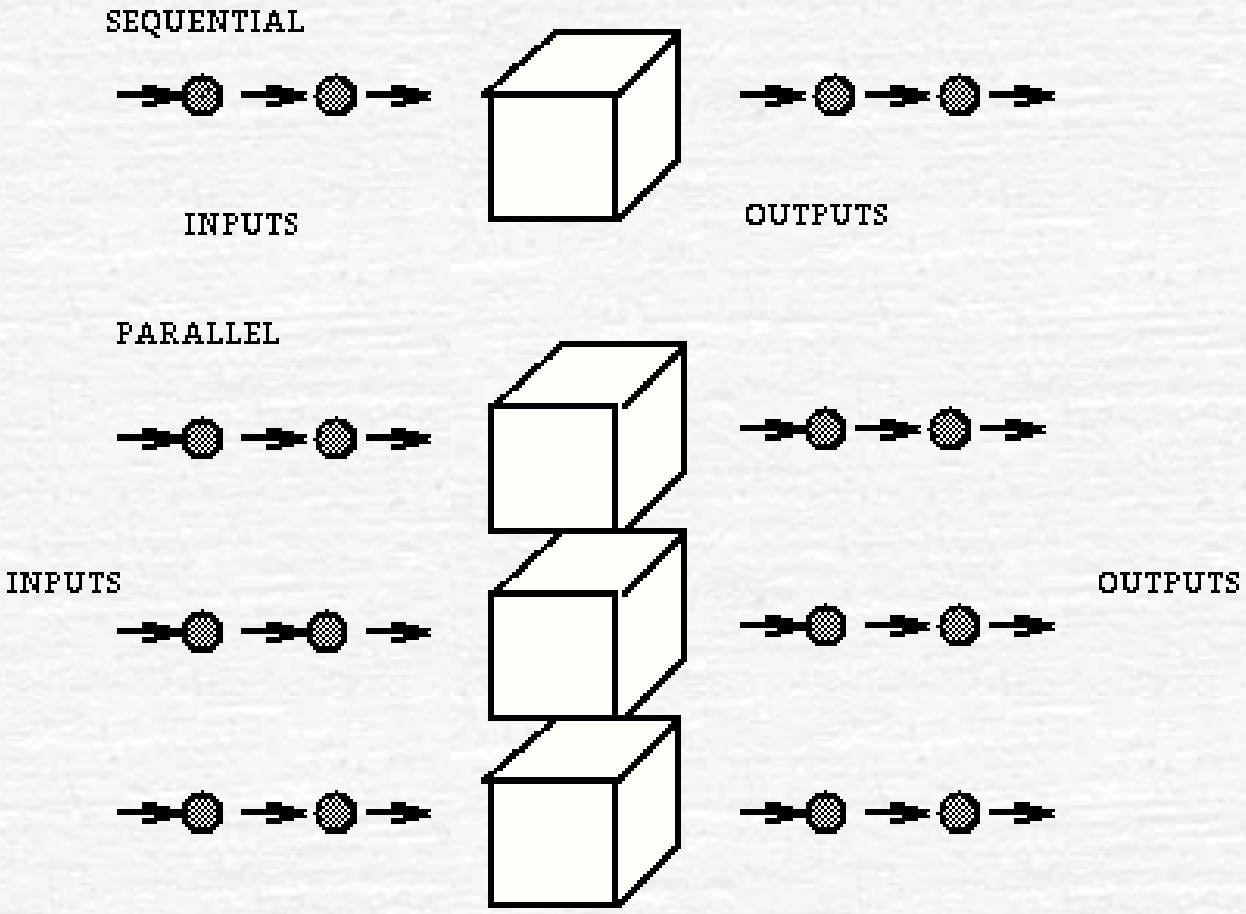


Classification of Architectures – Flynn's classification

- Single Instruction Single Data (SISD): Serial Computers
- Single Instruction Multiple Data (SIMD)
  - Processor arrays
- Multiple Instruction Single Data (MISD): Not popular
- Multiple Instruction Multiple Data (MIMD)
  - Most popular
  - most supercomputers, clusters, computational Grids etc

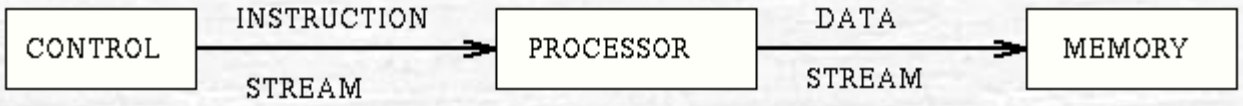
	Single Data	Multiple Data
Single Instruction	SISD	SIMD
Multiple Instruction	MISD	MIMD

Sequential vs. Parallel Processing



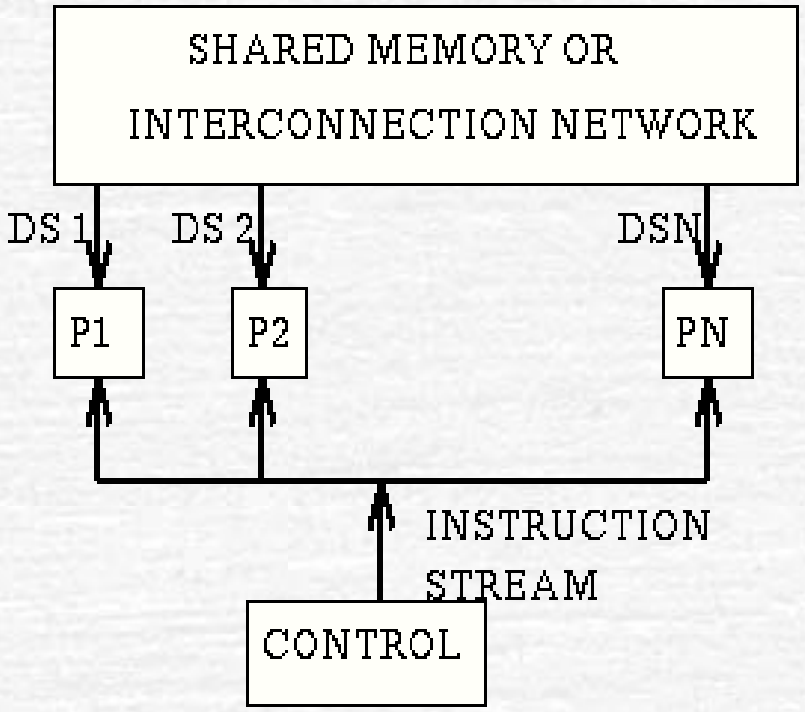
SISD Computers

- **a stream of** instructions **(the algorithm) tells the computer what to do.**
- **a stream of** data **(the input) is affected by these instructions.**



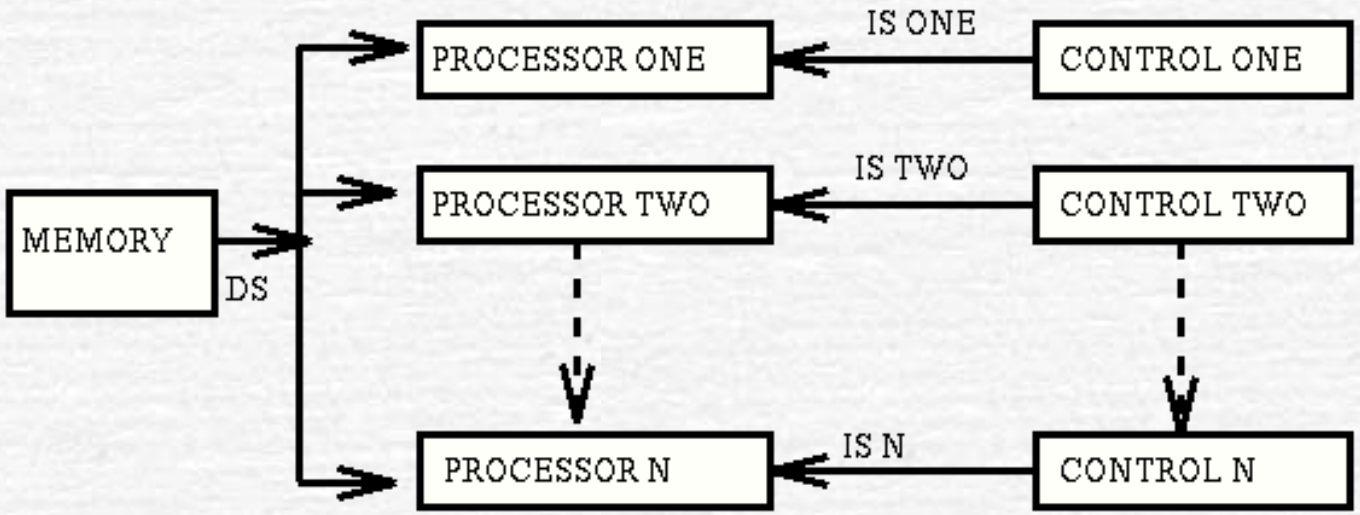
SIMD Computers

- A single instruction stream is broadcasted to multiple processors, each having its own data stream



P = PROCESSOR
DS = DATA STREAM

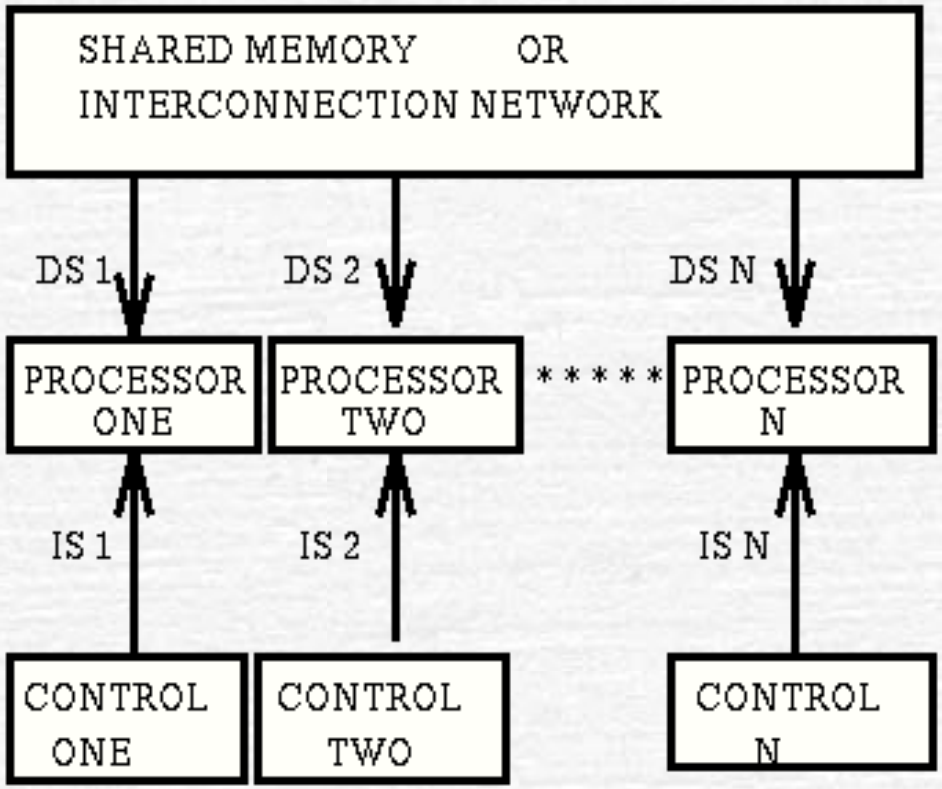
MISD Computers



IS = INSTRUCTION STREAM

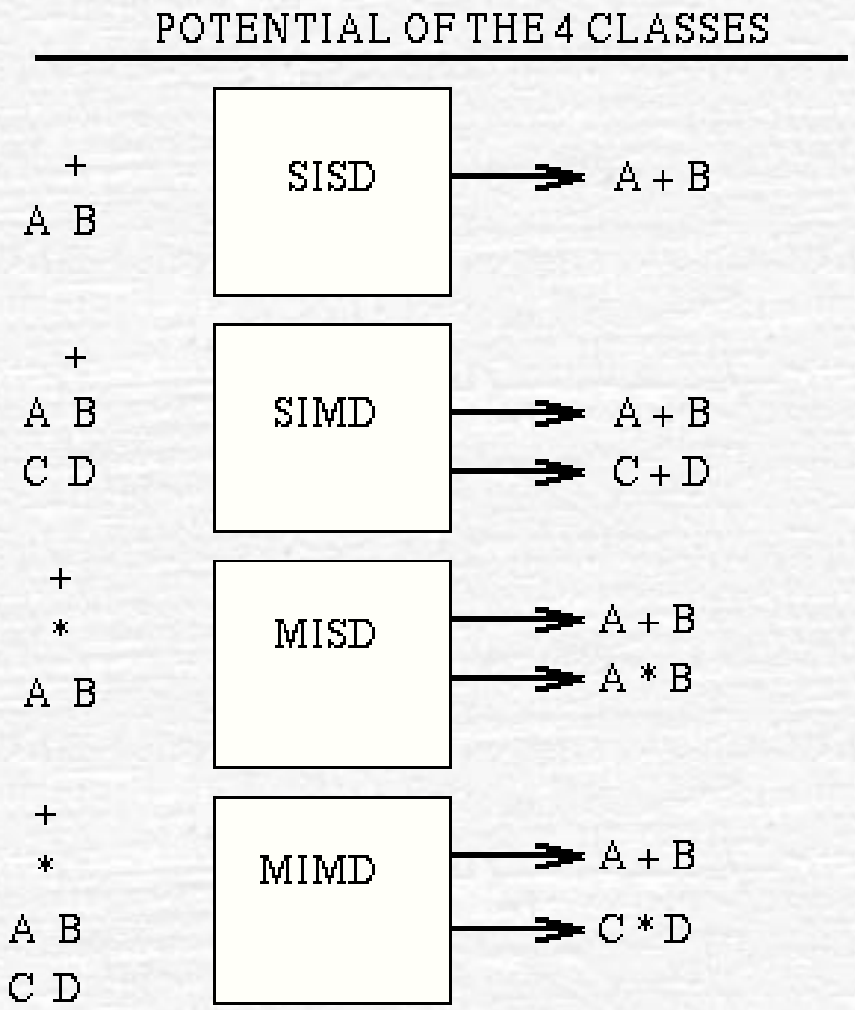
DS = DATA STREAM

MIMD (multiprocessors / multicomputers)



DS = DATA STREAM IS = INSTRUCTION STREAM

Potential of the 4 Classes



Memory reference Mechanisms

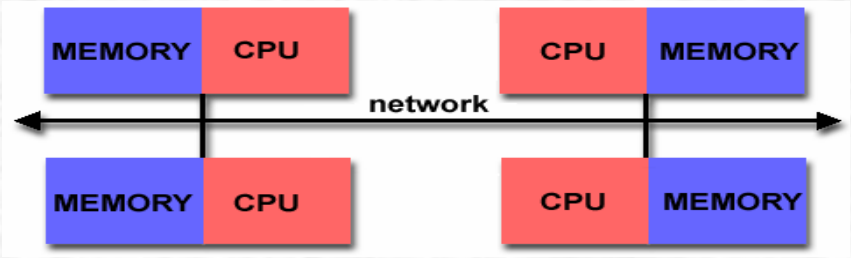
- **Shared Memory**
- **One sided Communication**
  - Supports a single shared address space
  - All threads reference all memory locations
  - It does not attempt to keep memory coherent
  - put() and get() to respectively store or load data
- **Message passing**
  - Two-sided communication mechanism
  - Requires collaboration between processors
  - send and receive methods

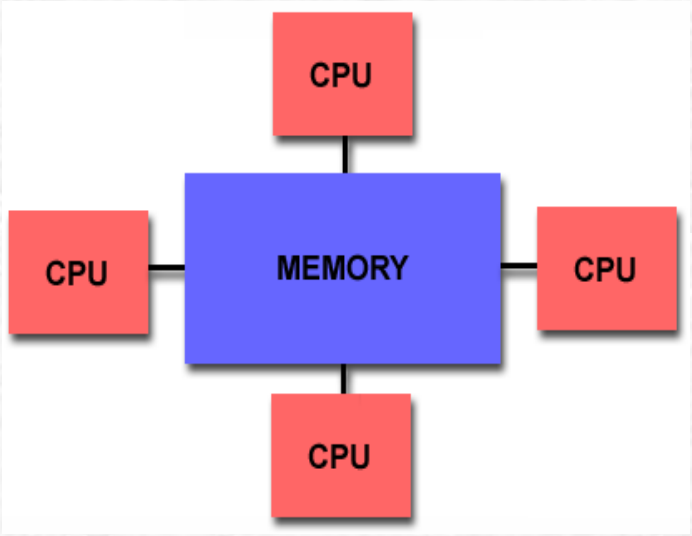
Parallel Paradigms

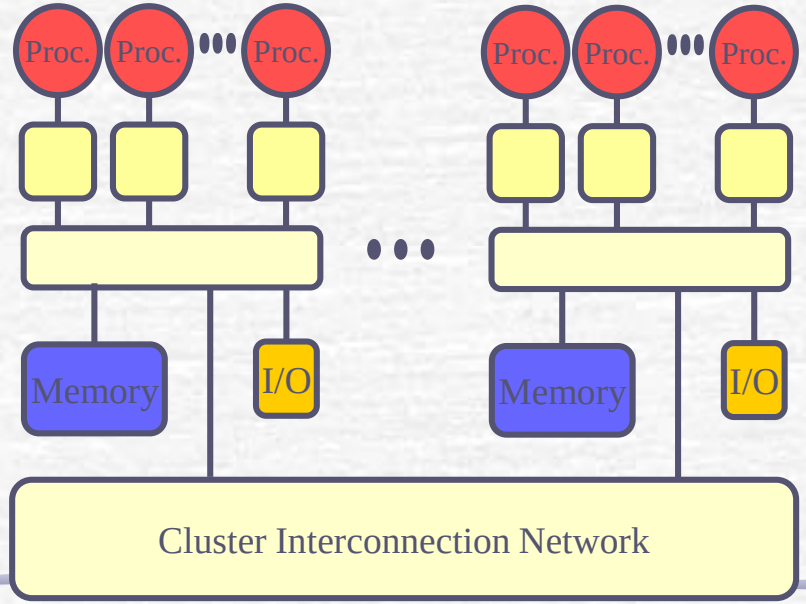
- Shared Memory Paradigm
- Message Passing
- Multi-threading

Shared Memory Paradigm

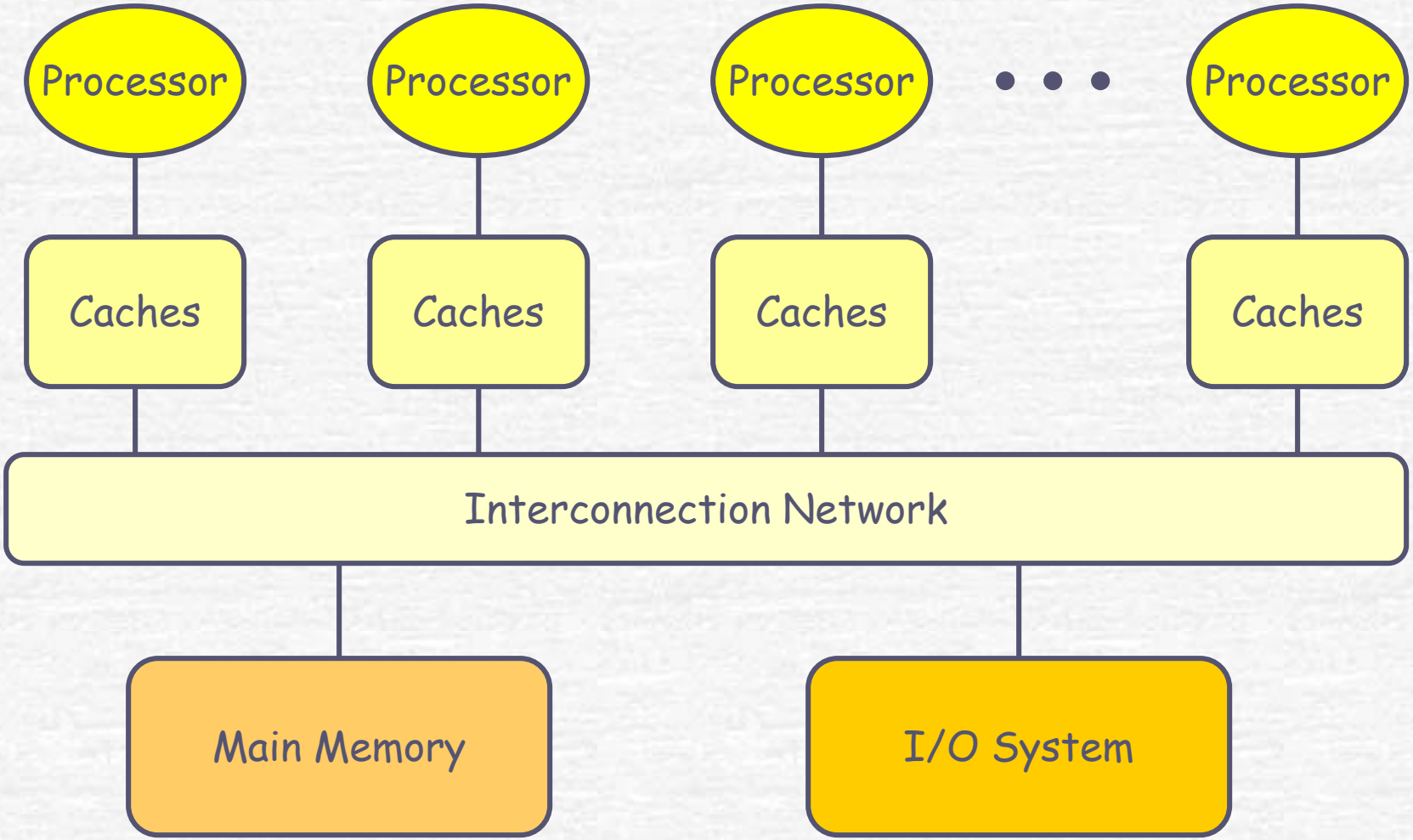
- Shared memory
- Distributed memory
- Hybrid systems



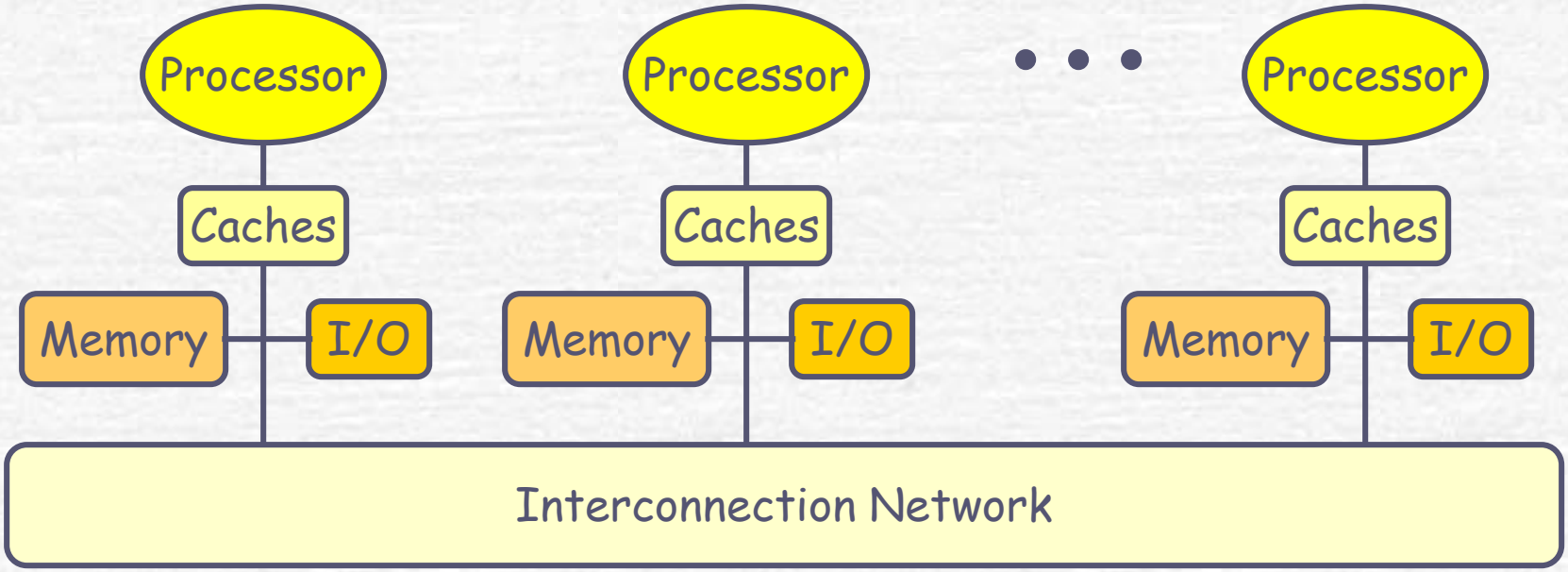




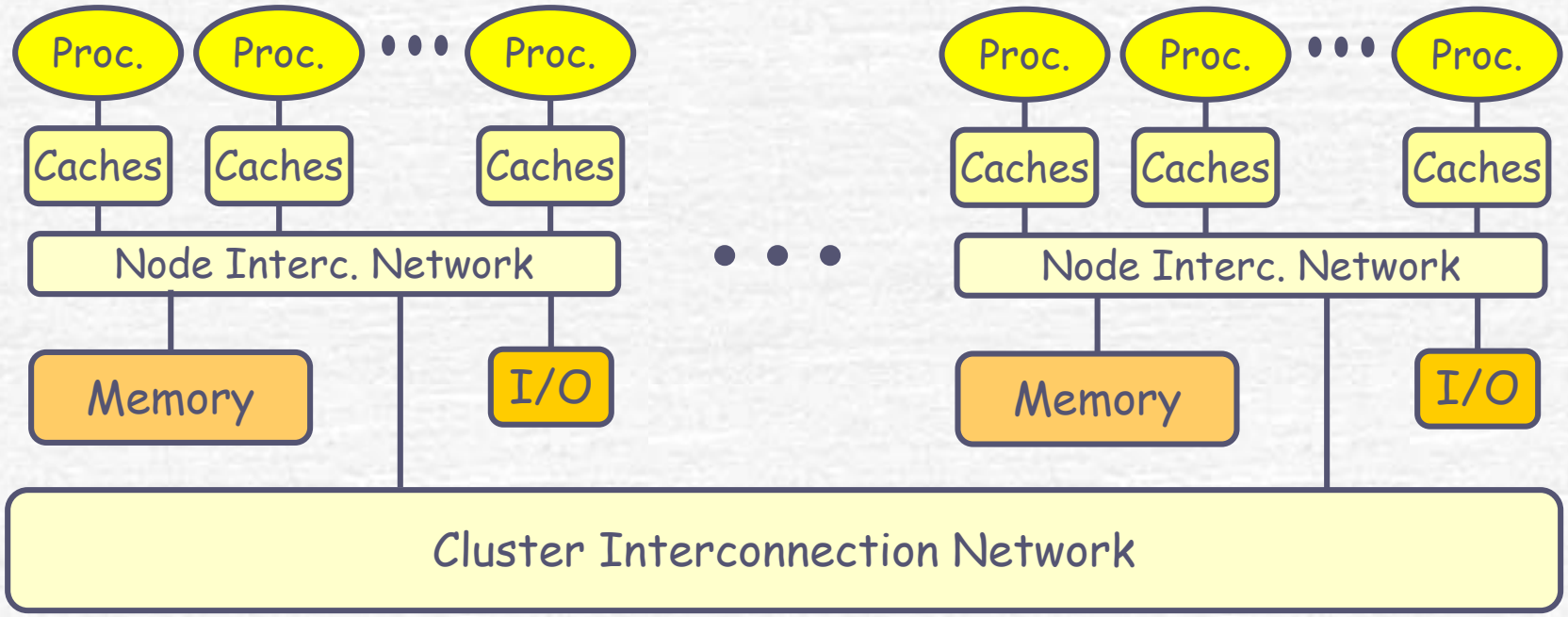
Centralized Shared Memory



Distributed Memory Architecture



Distributed Shared Memory



Message Passing Paradigm

- Allows for communication between a set of processors.
- Each processor is required to have a local memory, no global memory is required.
- The whole address space of the system consists of multiple private address spaces.
- Communication occurs between processors by sending and receiving messages.

Basic Operations

- **Blocking non-buffered send/ receive**
  - the sender issues a *send* operation and cannot proceed until a matching receive at the receiver's side is encountered and the operation is complete.
- **Blocking buffered send/ receive**
  - when the sender process reaches a *send* operation it copies the data into the buffer on its side and can proceed without waiting.
  - At the receiver side it is not necessary that the received data will be stored directly at the designated location.
  - When the receiver process encounters a receive operations it checks the buffer for data.

Basic Operations

- **Non-Blocking non-buffered send/ receive**
  - the process needs not to be idle but instead can do useful computations while waiting for the send / receive operation to complete.
- **Non-Blocking buffered send/ receive**
  - the sender issues a direct memory access operation (DMA) to the buffer. DMA operations can be carried out without processor being involved.
  - The sender can proceed with its computations.
  - At the receiver side, when a receive operation is encountered the data is transferred from the buffer to memory location.

- **Broadcast**
  - This function allows one process (called the root) to send the same data to all communicator members
- **Scatter**
  - llows one process to give out the content of its send buffer to all processes in a communicator.
- **Gather**
  - Each process gives out the data in its send buffer to the root process which stores them according to their ranks.

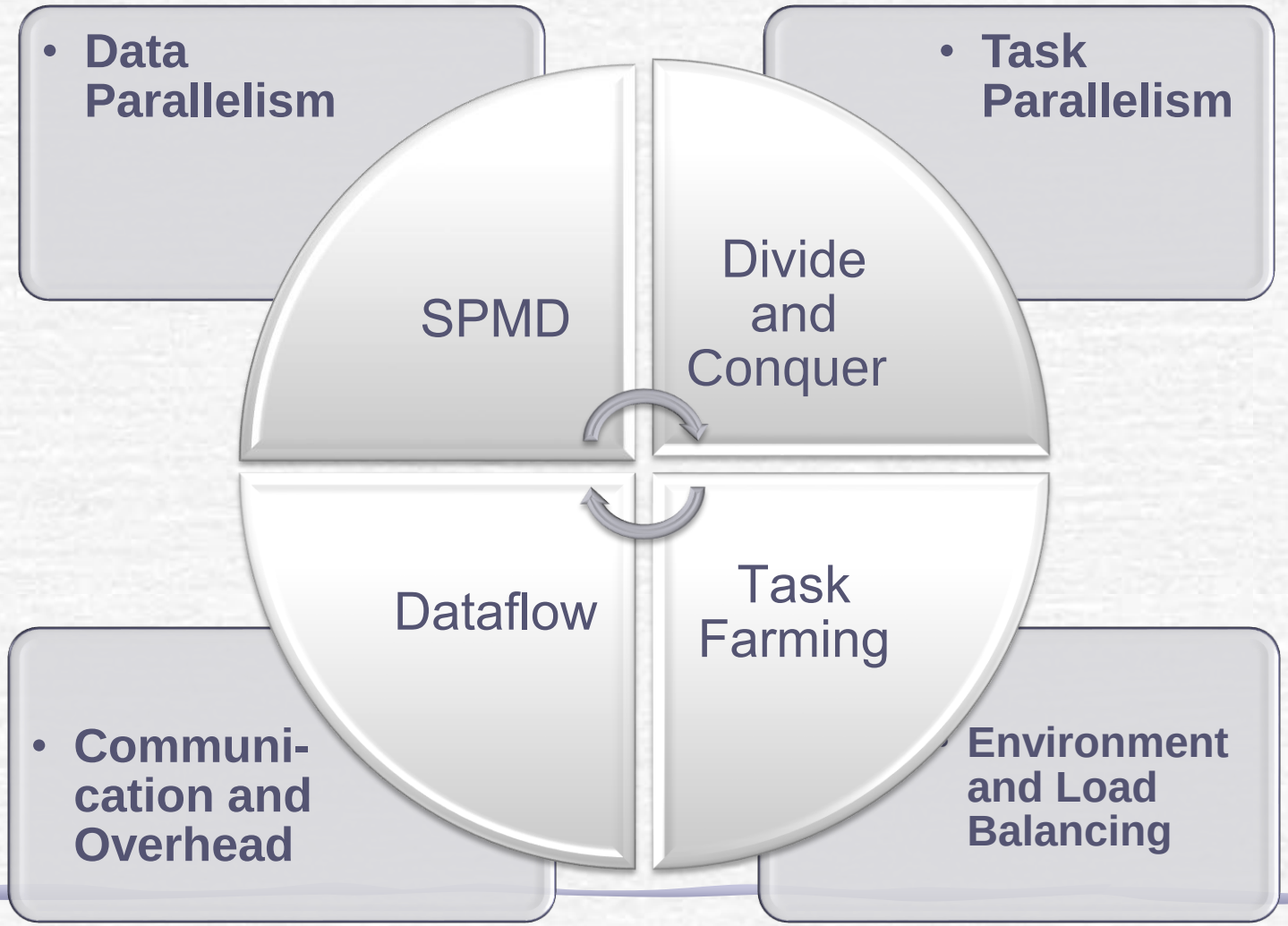
Multi-threading Paradigm

- **In a single-core (superscalar) system,**
  - we can define multithreading as the ability of the processor's hardware to run two or more threads in an overlapping fashion by allowing them to share the functional units of that processor.
- **in a multi-core system,**
  - we can define multithreading as the ability of two or more processors to run two or more threads simultaneously (in parallel) where each thread run on a separate processor
- **Modern systems combine both multithreading approaches.**

Parallel Programming Models

- SPMD
- Task Farming
- Divide and Conquer
- Dataflow

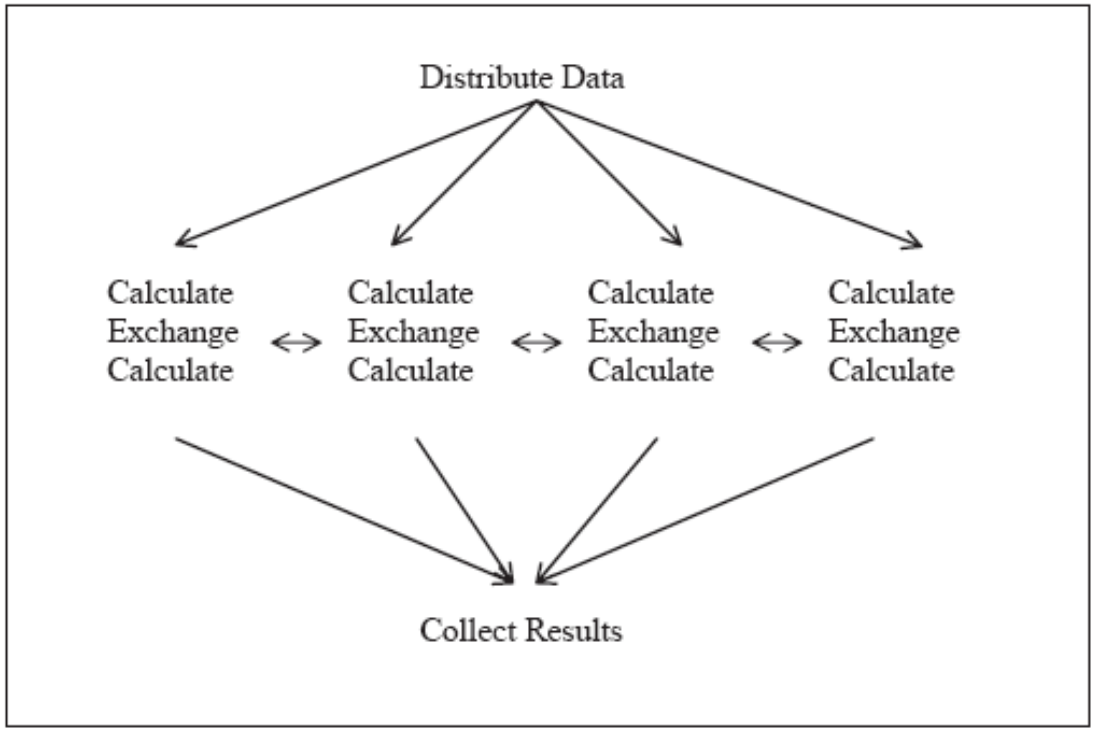
Models and Design Considerations



Single Program Multiple Data (SPMD)

- SPMD model is the dominant pattern to structure parallel programs.
- A single program is loaded into each node of a parallel system.
- Node are executing the code independently but act on multiple data sets including "private" and " shared " data.
- Nodes cooperating in the execution of the program are assigned unique IDs, allowed to self-schedule themselves, and dynamically get assigned to the required tasks under this cooperative execution.

SPMD: The Concept

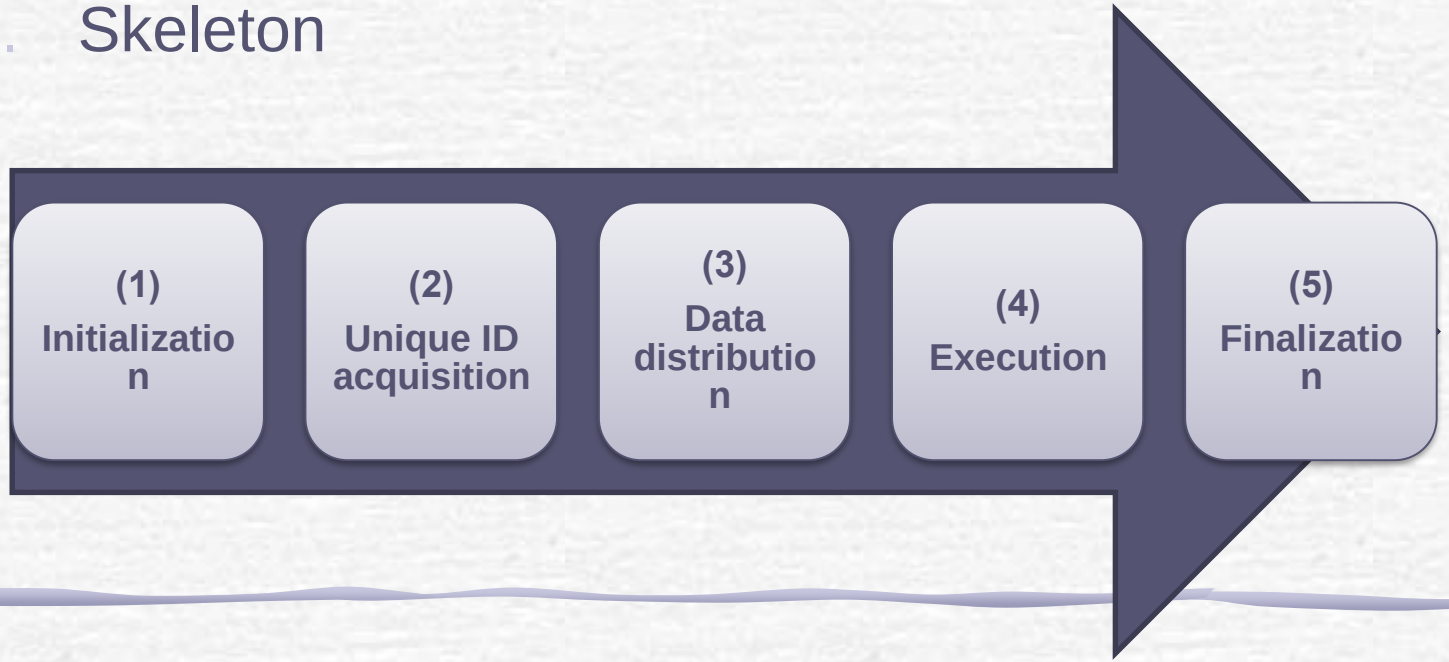


Basic structure of SPMD program

SPMD-Style Program Components

Program components:

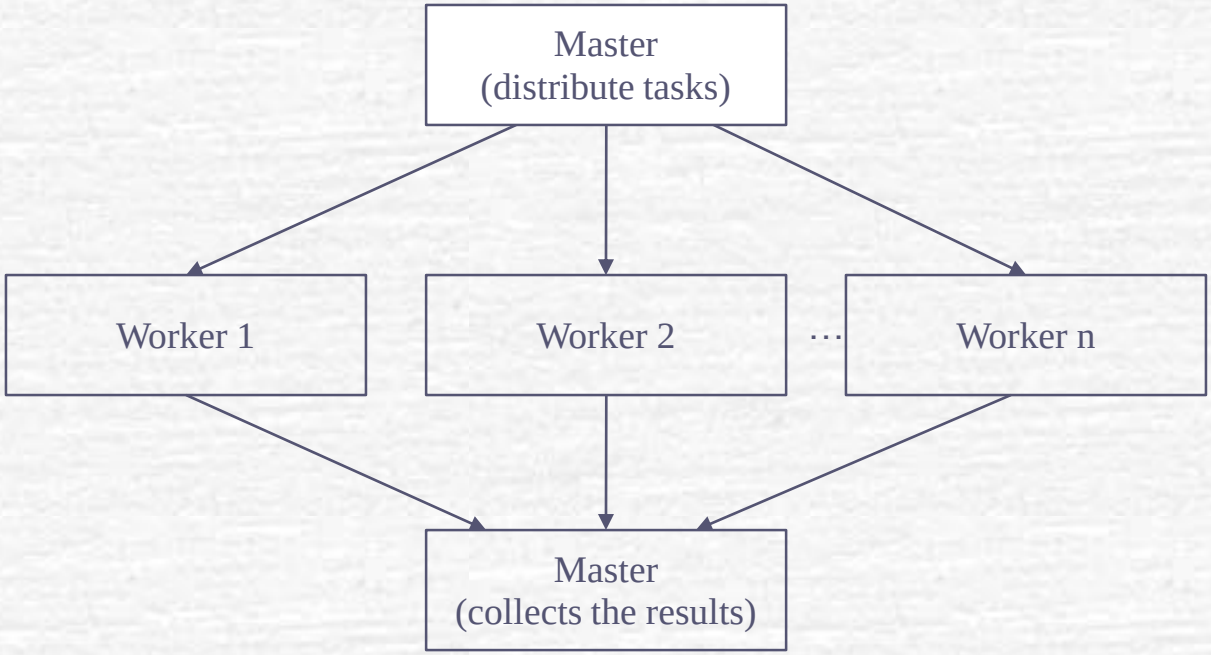
1. Source code
2. Load Balancing Strategy
3. Skeleton



Task Farming Model

- **Task farming (or Master-Worker or Master-Slave or Manager-Worker) consists of two entities:**
  - the master
  - and many workers.
- **The master:**
  - decomposes the problem into small tasks,
  - then distributes/farms these tasks among a farm of workers
  - and finally collects the partial results to produce the final result of the computation.
- **The worker:**
  - receives a task, process it and send the result back to the master.

Basic Task Farming Structure



Simple Task Farming Algorithm

Initialization

For task = 1 to N

PartialResult = + Function (task) ← Worker Tasks

End

act_on_tasks_complete() ← Master Tasks

Generalized Task Farming Algorithm

Initialization

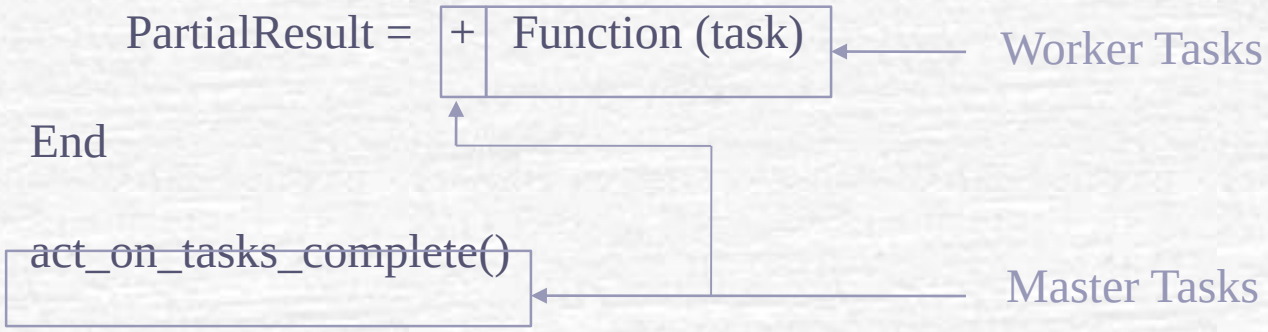
Do

For task = 1 to N

PartialResult = + Function (task)

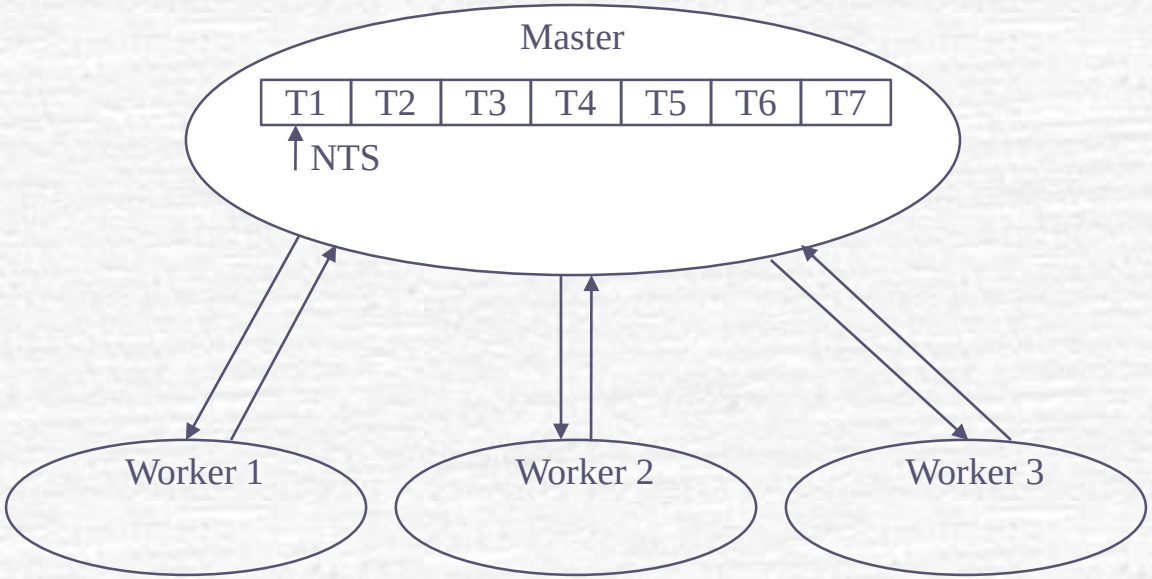
End

act_on_tasks_complete()

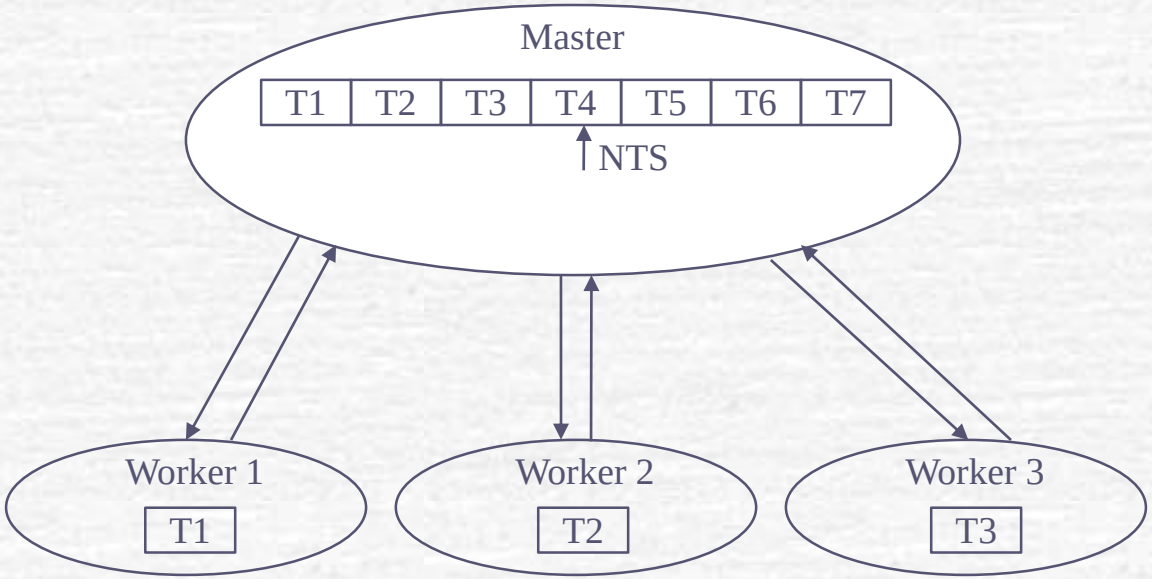


While (end condition not met)

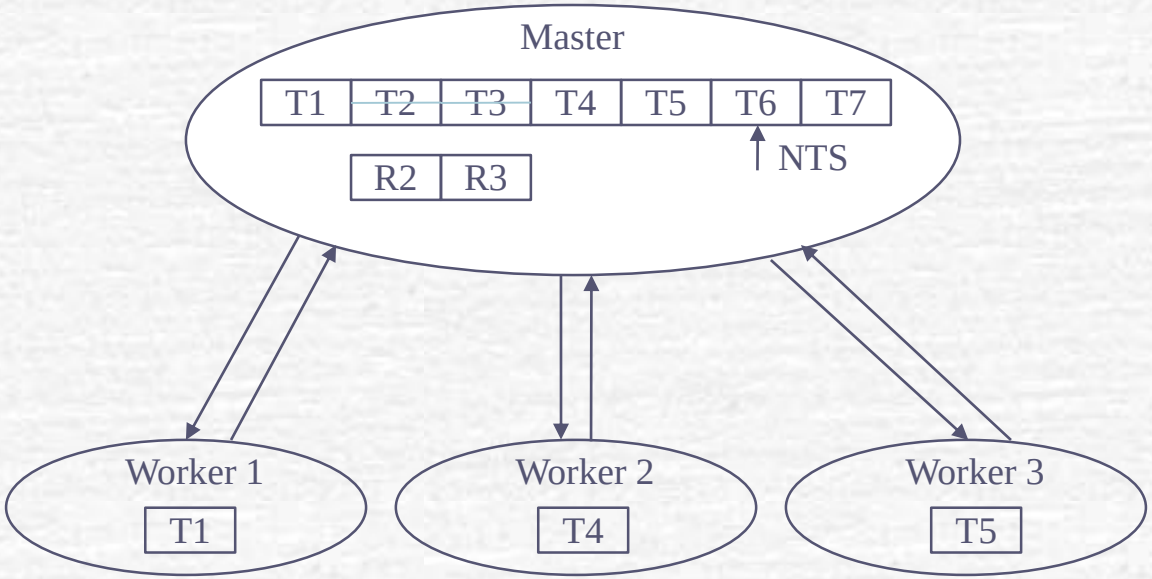
Simple Task Farming Example (a)



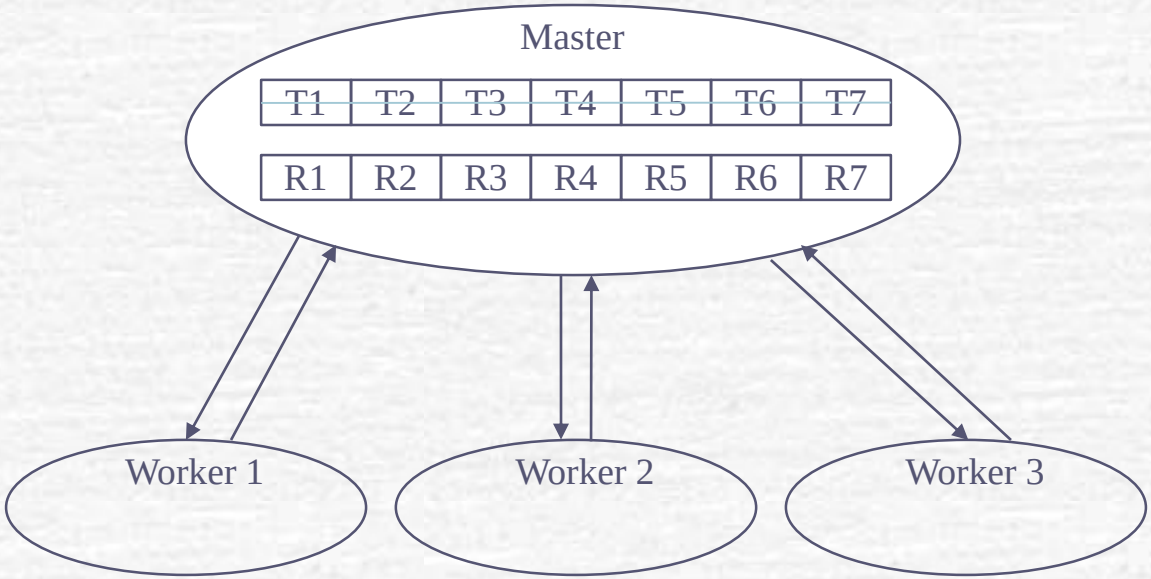
Simple Task Farming Example (b)



Simple Task Farming Example (c)



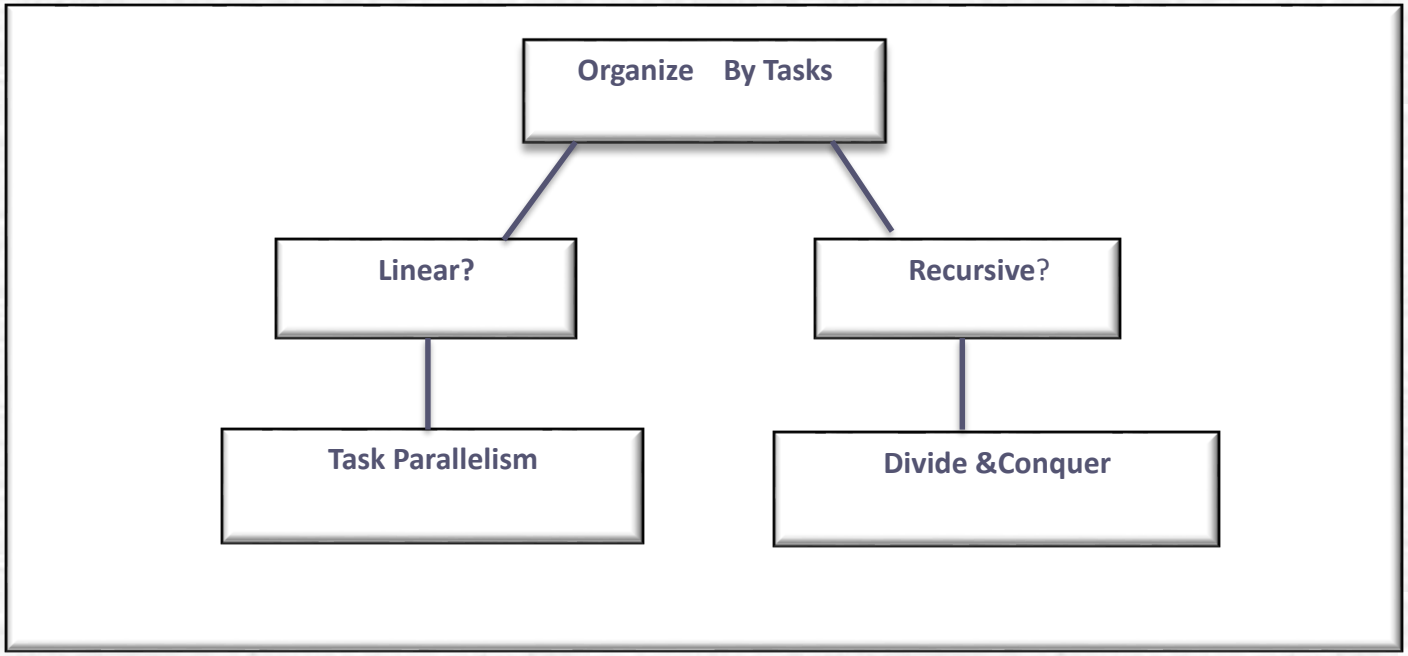
Simple Task Farming Example (d)



Divide and Conquer Model

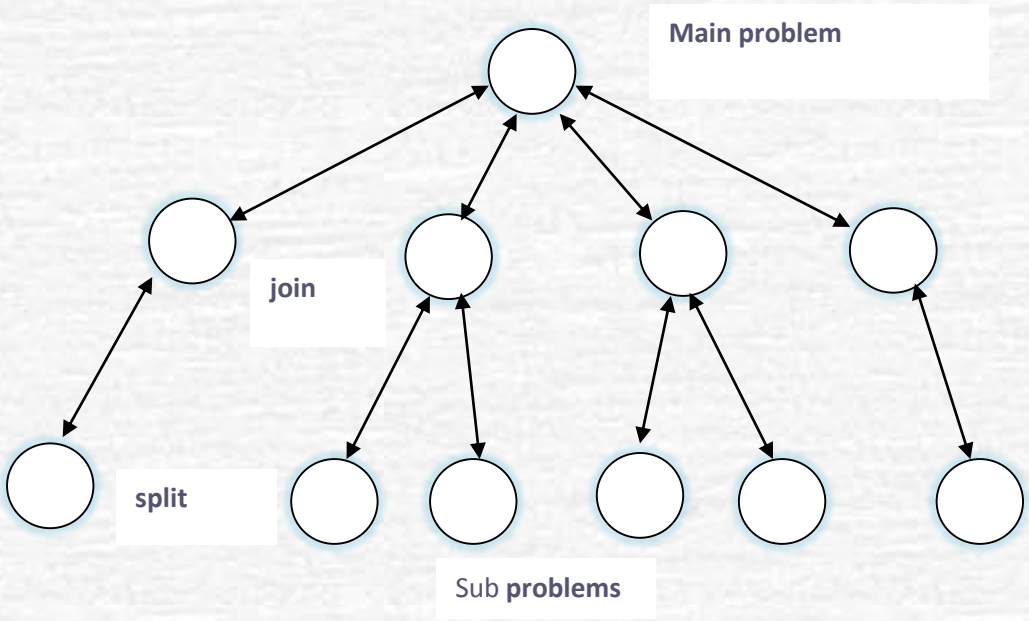
- This model solves a complex problem by splitting it into smaller easier sub-problems.
- The sub-problems have the same nature as the original one and could be solved by recursively applying the same algorithm.
- The recursion stops at the base case in which the sub-problem is solved directly.
- The results of all sub-problems are then joined to produce the final overall solution.
- The sub-problems are usually solved independently and concurrently

Task Organization



- The key point in Divide and Conquer is that the same task is recursively performed on different data.

Task Decomposition

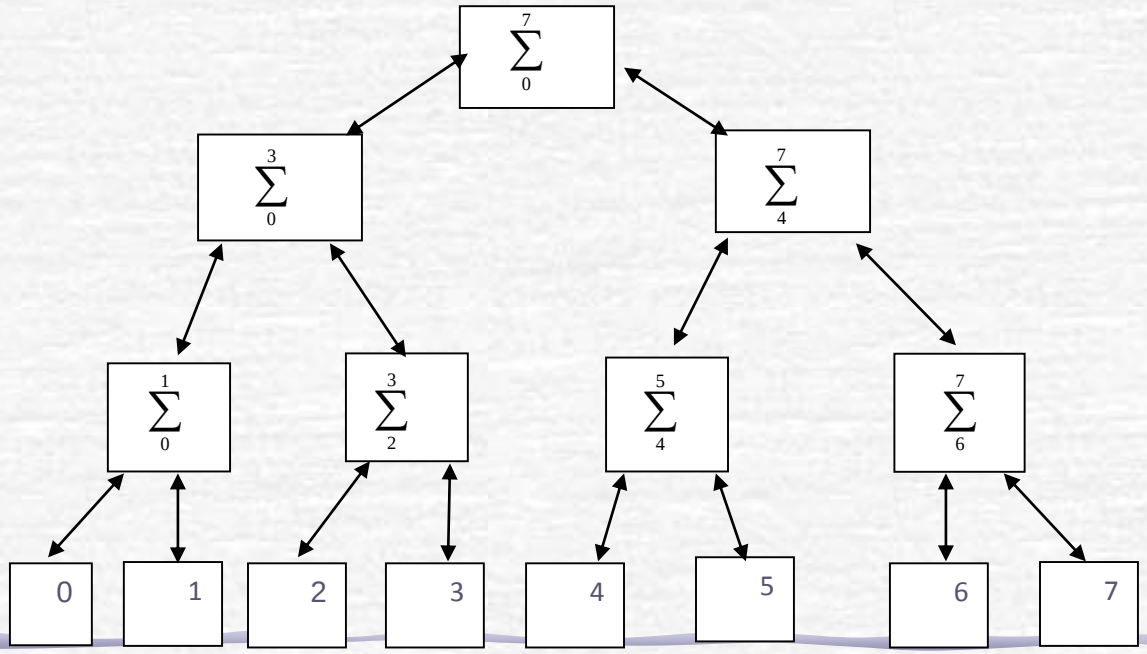


Decomposition of Tasks is dynamic because not all tasks are known in advance.

Parallel Divide and Conquer Example

- Problem : Summing N numbers .
- Solution : Divide into two subproblems each of size N/2

  Using the feature $\sum_{i=0}^{2^n-1} = \sum_{i=0}^{2^{n-1}-1} + \sum_{i=2^{n-1}}^{2^n-1}$

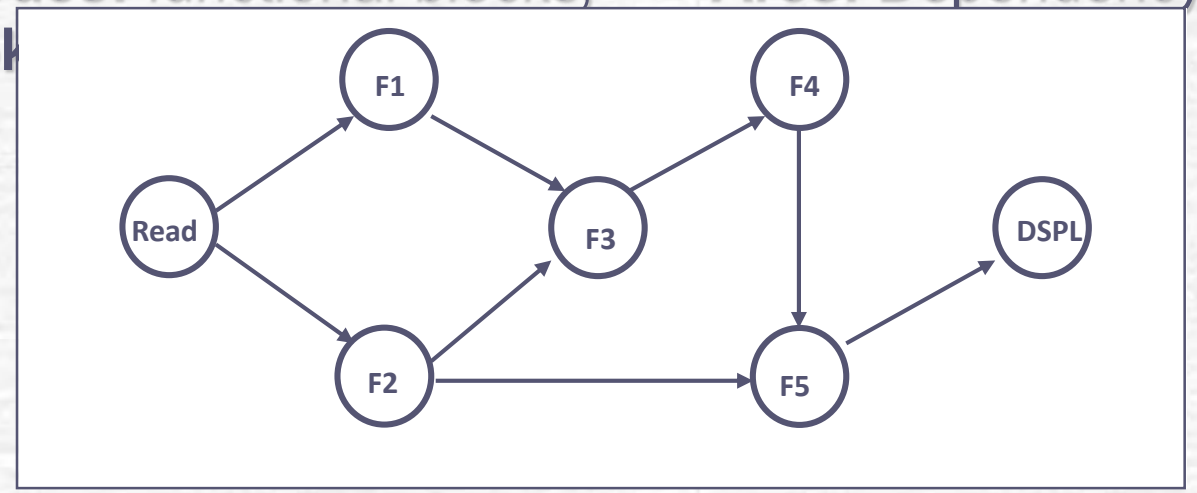


Dataflow

- **The main concept of dataflow programming is to divide the computation problem into multiple disjoint functional blocks.**
- **Each block solves part of the problem.**
- **Blocks are connected to each other to :**
  - show the dependency between them.
  - *express the logical execution flow*, and they can be used to easily express parallelism.
- **Data-pipelining is a specialized case of dataflow model.**

Principles of Dataflow Model

- **In dataflow model, computation is modeled as a directed graph.**
  - **Nodes:** functional blocks, **Arcs:** Dependency, **Tok**



There may be many nodes that are ready to fire (execute) at a given time.