VIT®
BHOPAL
www.vitbhopal.ac.in

CSE3009 - Parallel and Distributed Computing

Course Type: LTP Credits: 4

Prepared by
Dr Komarasamy G
Senior Associate Professor
School of Computing Science and Engineering
VIT Bhopal University

Unit-2

Parallel Algorithm and Design - Preliminaries – Decomposition Techniques – Mapping Techniques for Load balancing.

Synchronous Parallel Processing – Introduction, Example - SIMD Architecture and Programming Principles.

Synchronous Parallel Processing

Parallel Computing :

- It is the use of multiple processing elements **simultaneously for solving any problem.**
- Problems are broken down into instructions and are solved concurrently as each resource that has been applied to work is working at the same time.

Advantages of Parallel Computing over Serial Computing are as follows:

- It saves time and money as many resources working together will reduce the time and cut potential costs.
- It can be impractical to solve larger problems on Serial Computing.
- It can take advantage of non-local resources when the local resources are finite.
- Serial Computing 'wastes' the potential computing power, thus Parallel Computing makes better work of the hardware.

Synchronous Parallel Processing

Why parallel computing?

- The whole real-world runs in dynamic nature i.e. many things happen at a certain time but at different places concurrently.
- This data is extensively huge to manage.
- Real-world data needs more dynamic simulation and modeling, and for achieving the same, parallel computing is the key.
- Parallel computing provides **concurrency and saves time and money.**
- Complex, large datasets, and their management can be organized only and only using parallel computing approach.
- Ensures the effective utilization of the resources. The hardware is guaranteed to be used effectively whereas in serial computation only some part of the hardware was used and the rest rendered idle.
- Also, it is impractical to implement real-time systems using serial computing.

Synchronous Parallel Processing

Parallel processing is particularly useful when running programs that perform complex computations, and it provides a viable option to the quest for cheaper computing alternatives. Supercomputers commonly have hundreds of thousands of microprocessors for this purpose. Parallel processing should not be confused with concurrency, which refers to multiple tasks that run simultaneously.

Each of the processing units has its own local memory unit to store both data and instructions. In SIMD computers, processors need to communicate among themselves. This is done by **shared memory** or by **interconnection network**.

While some of the processors execute a set of instructions, the remaining processors wait for their next set of instructions. Instructions from the control unit decides which processor will be **active** (execute instructions) or **inactive** (wait for next instruction).

Synchronous Parallel Processing

Parallel Processing

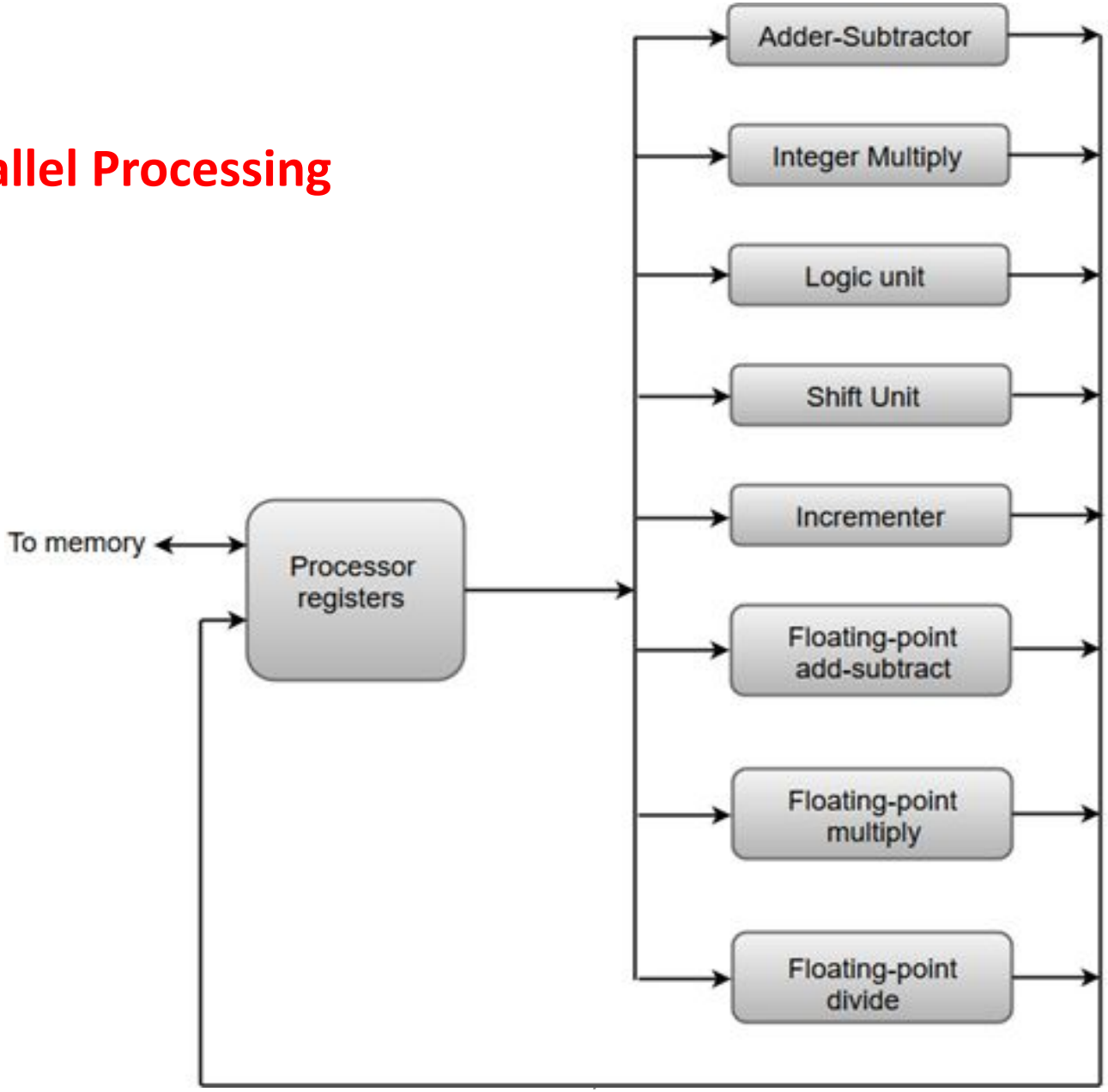
- Parallel processing can be described as a **class of techniques which enables the system to achieve simultaneous data-processing tasks to increase the computational speed of a computer system.**
- A parallel processing system can carry out **simultaneous data-processing to achieve faster execution time.**
- For instance, while an instruction is being processed in the ALU component of the CPU, the next instruction can be read from memory.
- The primary purpose of parallel processing is to enhance the computer processing capability and increase its **throughput**, i.e. the **amount of processing that can be accomplished during a given interval of time.**

Synchronous Parallel Processing

- A parallel processing system can be achieved by having a **multiplicity of functional units** that perform identical or different operations simultaneously.
- The data can be distributed among various multiple functional units.
- The diagram shows one possible way of separating the execution unit into eight functional units operating in parallel.

Synchronous Parallel Processing

Parallel Processing



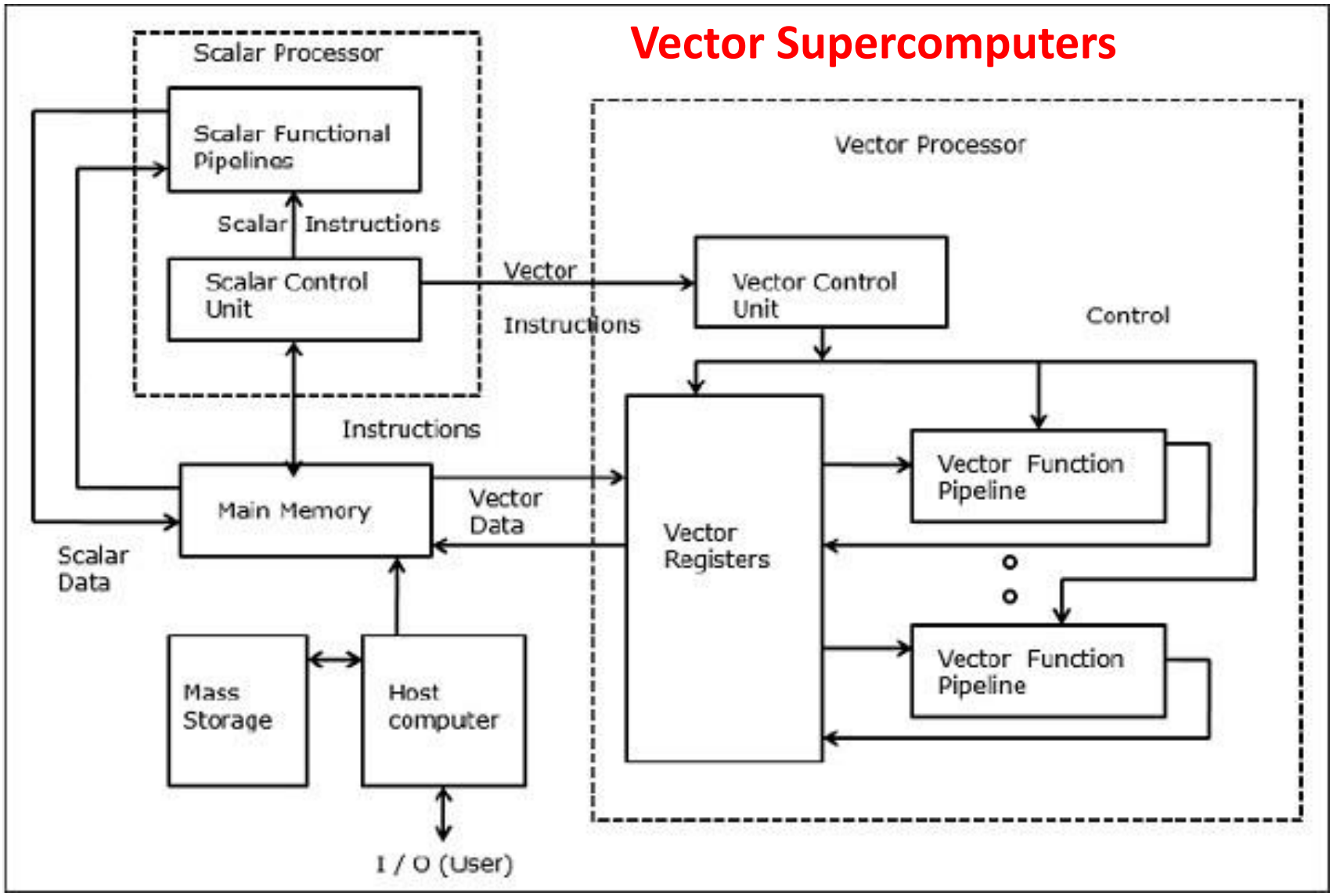
Synchronous Parallel Processing

- The adder and integer multiplier performs the arithmetic operation with integer numbers.
- The **floating-point operations** are separated into three circuits operating in parallel.
- The logic, shift, and increment operations can be performed concurrently on different data.
- All units are independent of each other, so one number can be shifted while another number is being incremented.

Introduction about SIMD

Single instruction, multiple data (SIMD) is a class of parallel computers in Flynn's taxonomy It describes computers with multiple processing elements that perform the same operation on multiple data points simultaneously. Such machines exploit data level parallelism, but not concurrency: there are simultaneous (parallel) computations, but only a single process (instruction) at a given moment. SIMD is particularly applicable to common tasks such as adjusting the contrast in a digital image or adjusting the volume of digital audio. Most modern CPU designs include SIMD instructions to improve the performance of multimedia use.

Introduction about SIMD

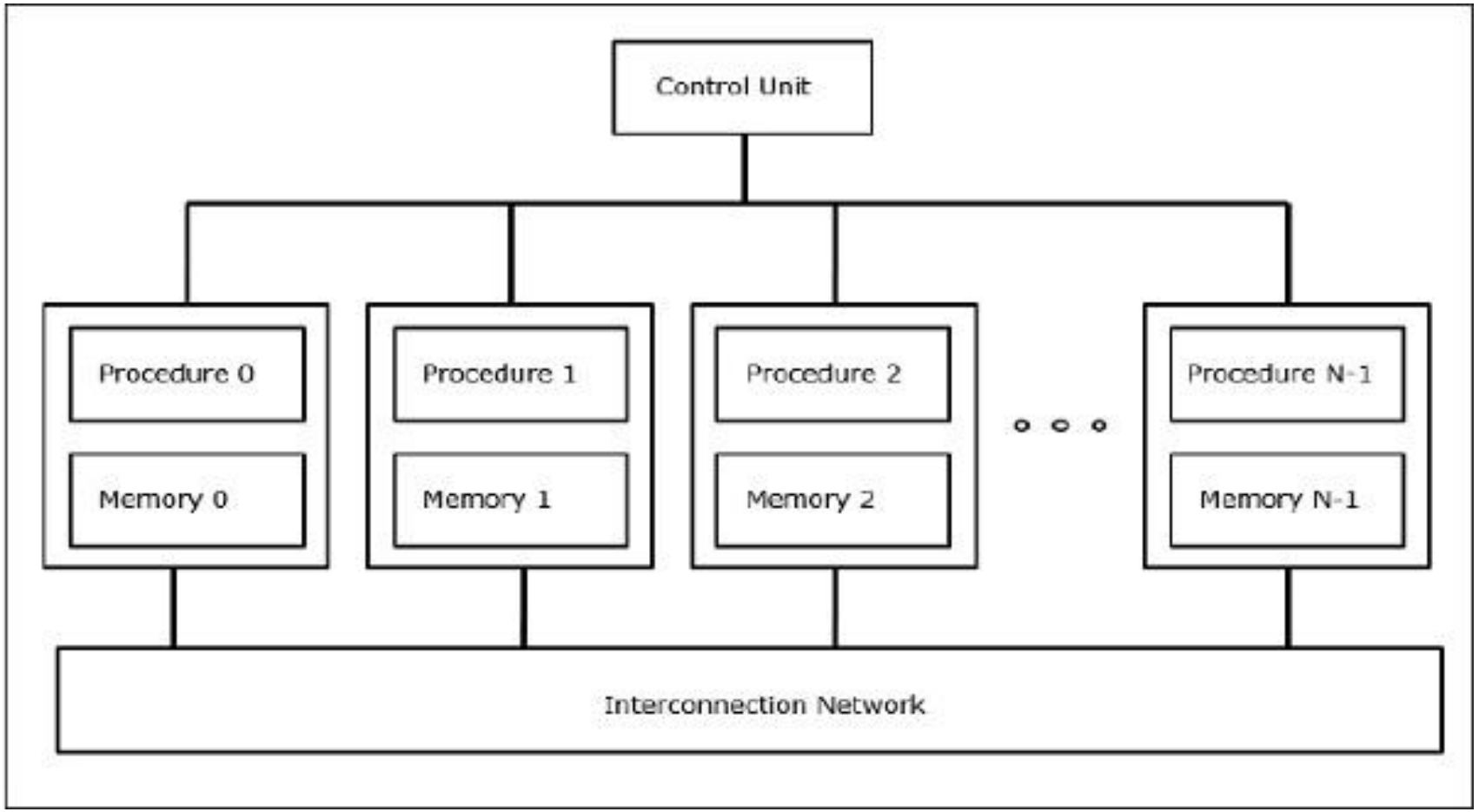


Introduction about SIMD

- **Multivector and SIMD Computers**
- Supercomputers and parallel processors for vector processing and data parallelism.
- **Vector Supercomputers**
- In a vector computer, a vector processor is attached to the scalar processor as an **optional feature.**
- The host computer **first loads program and data to the main memory.** Then the scalar control unit **decodes all the instructions**.
- If the decoded instructions are scalar operations or program operations, the scalar processor executes those operations using **scalar functional pipelines.**
- On the other hand, if the decoded instructions are vector operations then the instructions will be sent to **vector control unit.**

Introduction about SIMD

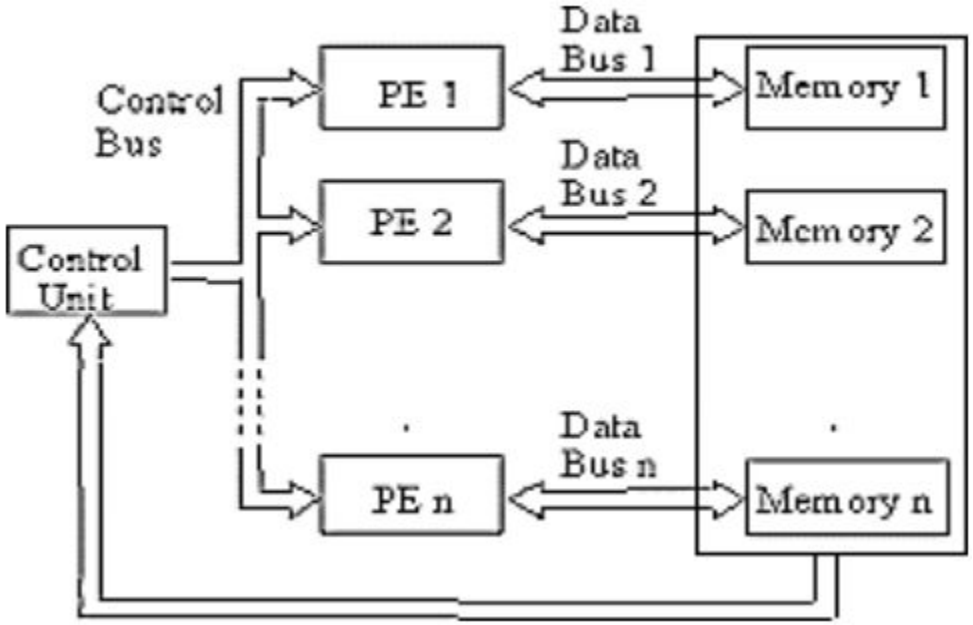
- **SIMD Supercomputers**
- In SIMD computers, ‘N’ number of processors are connected to a control unit and all the processors have their individual memory units. **All the processors are connected by an interconnection network.**



SIMD Architecture (Single instruction Multiple Data)

- Single instruction is applied to a multiple data item to produce the same output.
- Master instruction work on vector of operand.
- No of processors running the same instruction one clock cycle by the strict lock approach.
- It is type of Instruction level parallelism
- Communication network allow parallel synchronous communication between several Processing Element / Memory modules.

SIMD Processor Architecture

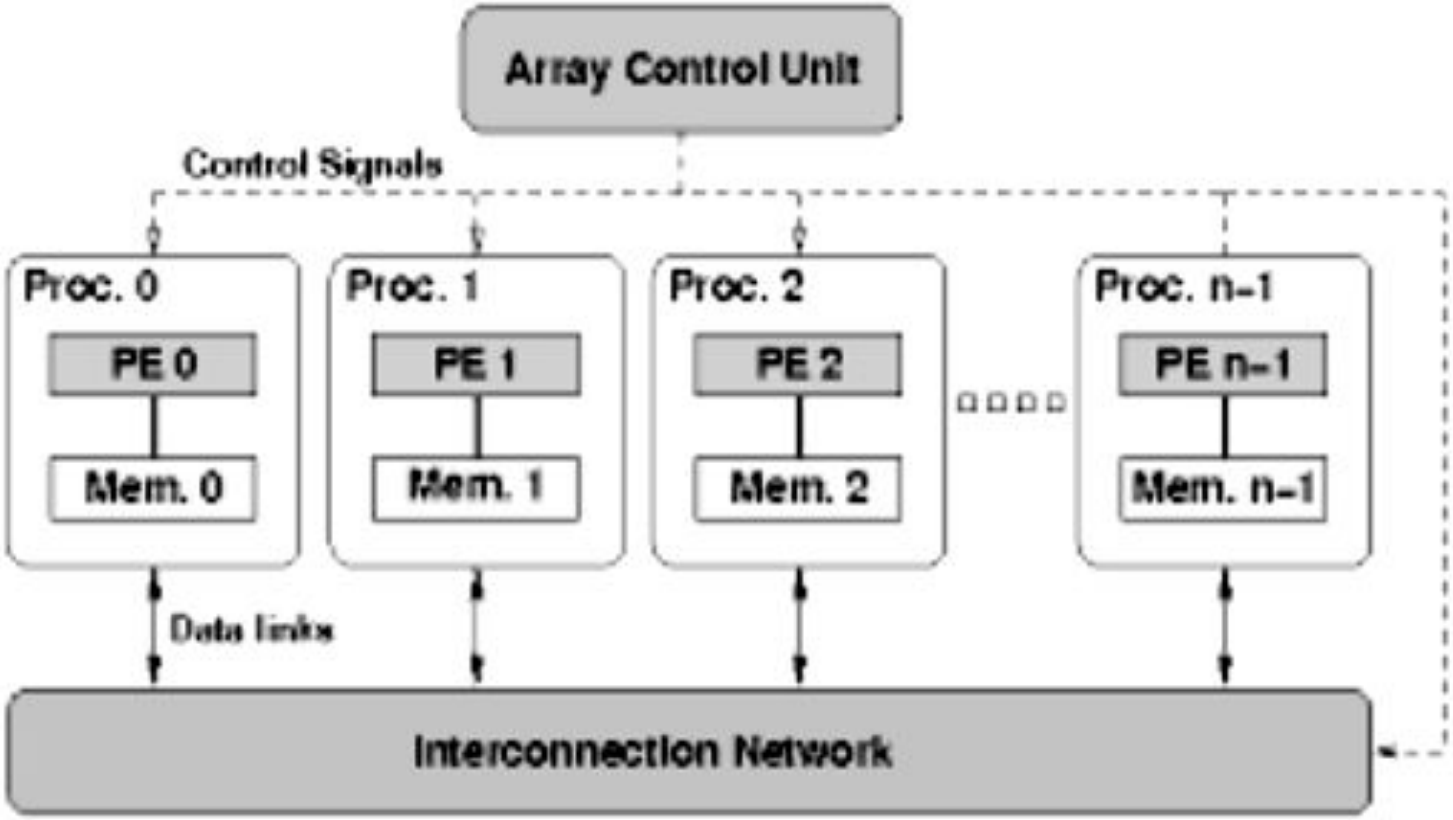


SIMD Architecture (Single instruction Multiple Data)

- **Following two SIMD architectures depict fundamentally different approaches to the parallel processing**

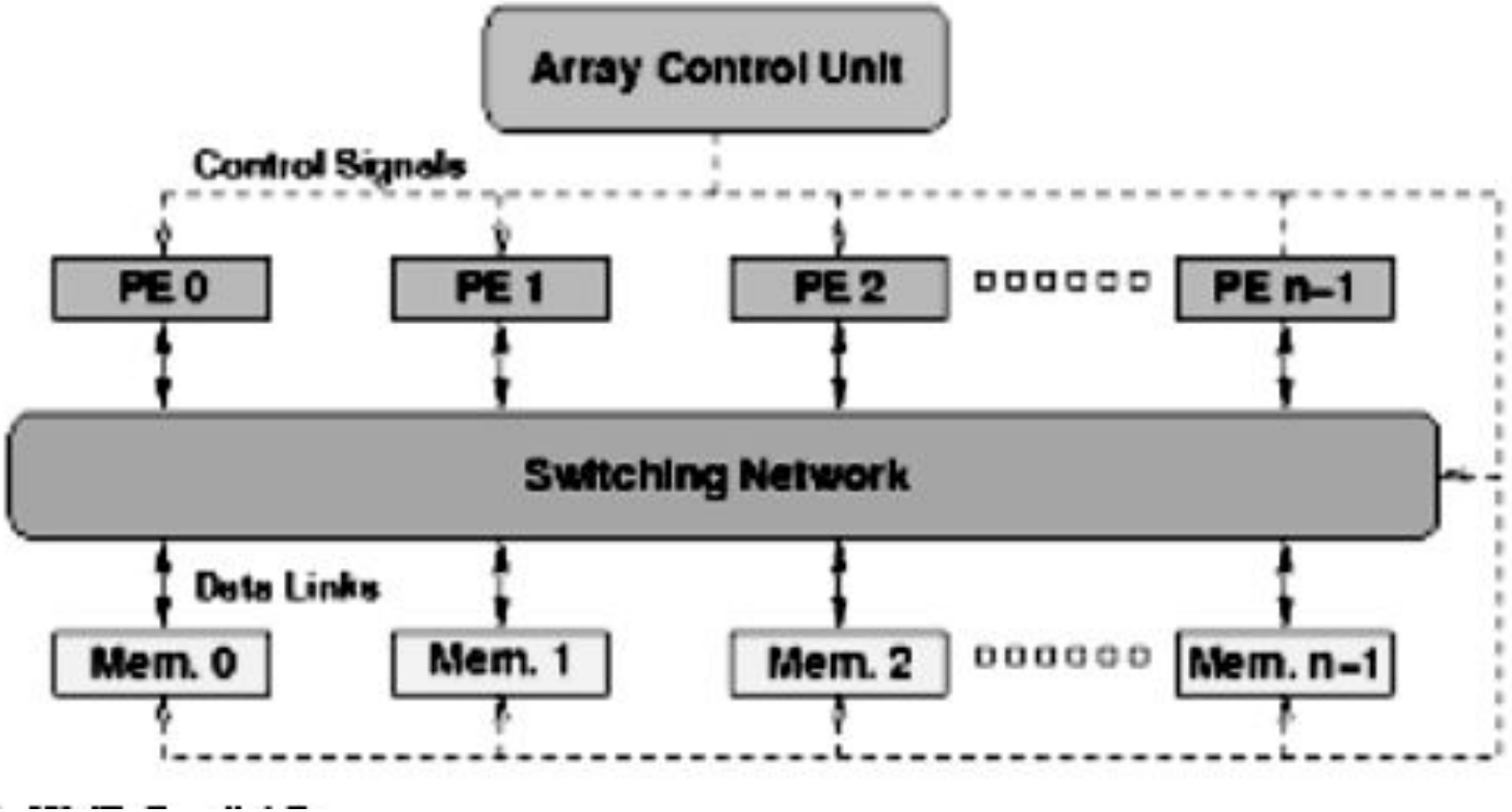
Data Communication based on message passing paradigm:

- Here the memory is part of PE and thus it communicates through the interconnection network for passing the data.



SIMD Architecture (Single instruction Multiple Data)

- **Shared memory between processors:**
- Here **memories are not local** and the data is read and aligned by the alignment network that aligns the data between PEs and Memory modules



SIMD Parallel Process:

- During the execution of program, it is often required to mask of a PE from doing processing, which is equivalent to having some autonomous control within a PE.
- PE has a mask bit which can be masked during processing of an instruction.
- When a mask in PE is set it receives instruction from Control Unit as No operation.
- Executes instruction when mask bit is reset.
- **Each PE has one or more index registers added to global addresses supplied by the CU Instruction.**
- The arithmetic logic unit has few general purpose registers and pointer registers to support data and address manipulation.

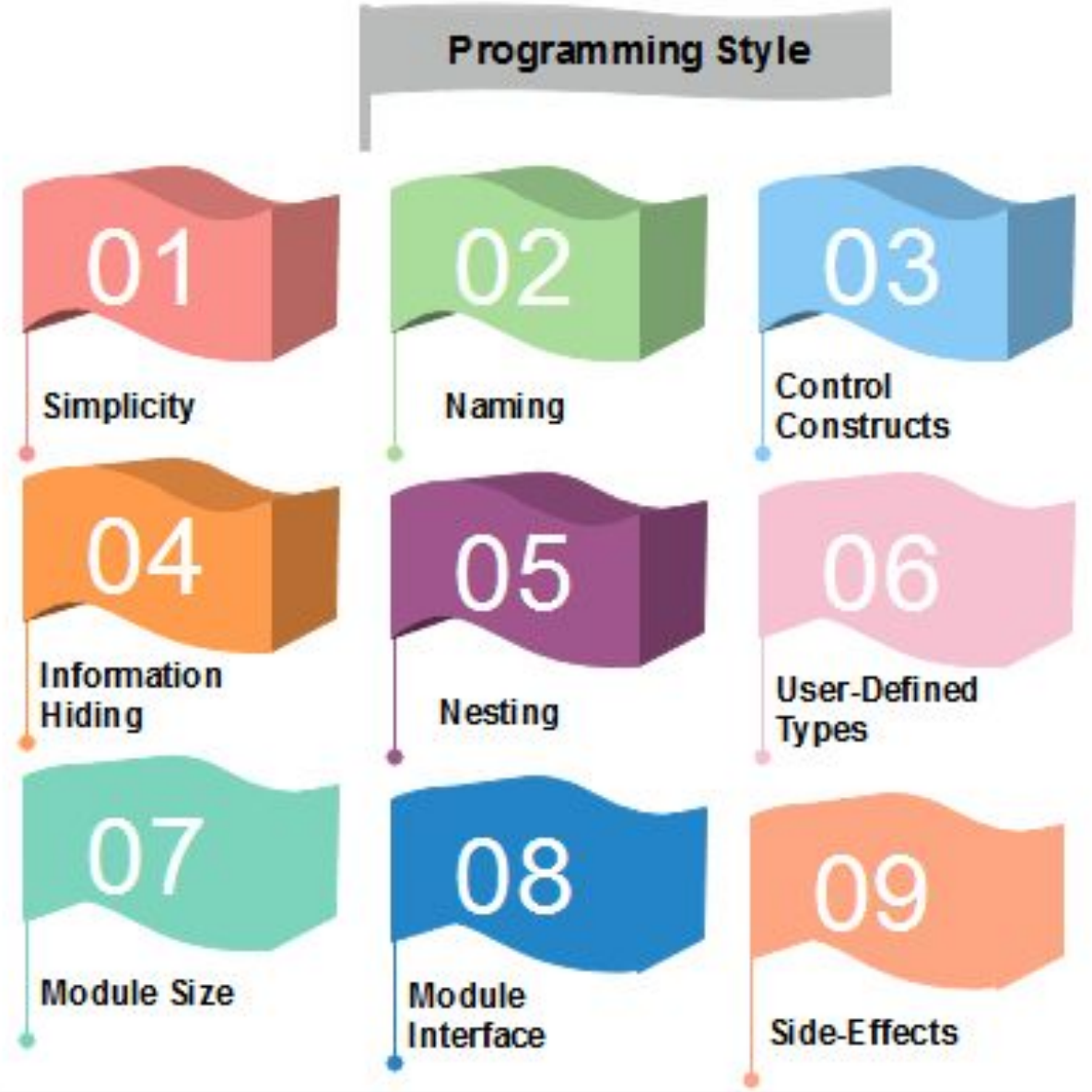
SIMD mesh connected architecture:

- Here we are dealing with the mesh Connected architecture which has been built using the mesh connected architecture.
- **Each node of such machine will have four ports- Top port, left port, right port and bottom port.**
- The instruction set belongs to CU with PEs executing some of instructions that are prefixed with P to indicate that these shall be executed on PEs in parallel.
- Each PE also has four bidirectional ports for communication to four neighbors.

Programming Principles

- Programming style refers to the technique used in **writing the source code for a computer program.**
- Most programming styles are designed to help programmers quickly read and understands the program as well as avoid making errors. **(Older programming styles also focused on conserving screen space.)**
- A good coding style can overcome the many deficiencies of a first programming language, while poor style can defeat the intent of an excellent language.
- The goal of good programming style is to **provide understandable, straightforward, elegant code.**
- The programming style used in a various program may be derived from the coding standards or code conventions of a company or other computing organization, as well as the preferences of the actual programmer.

- **Some general rules or guidelines in respect of programming style:**



- **1. Clarity and simplicity of Expression:** The programs should be designed in such a manner so that the objectives of the program is clear.
- **2. Naming:** In a program, you are required to name the module, processes, and variable, and so on. Care should be taken that the naming style should not be cryptic and non-representative.
- **For Example:** a = 3.14 * r * r

  area of circle = 3.14 * radius * radius;
- **3. Control Constructs:** It is desirable that as much as a possible single entry and single exit constructs used.
- **4. Information hiding:** The information secure in the data structures should be hidden from the rest of the system where possible. Information hiding can decrease the coupling between modules and make the system more maintainable.

- **5. Nesting:** Deep nesting of loops and conditions greatly harm the static and dynamic behavior of a program. It also becomes difficult to understand the program logic, so it is desirable to avoid deep nesting.
- **6. User-defined types:** Make heavy use of user-defined data types like enum, class, structure, and union. These data types make your program code easy to write and easy to understand.
- **7. Module size:** The module size should be uniform. The size of the module should not be too big or too small. If the module size is too large, it is not generally functionally cohesive. If the module size is too small, it leads to unnecessary overheads.
- **8. Module Interface:** A module with a complex interface should be carefully examined.
- **9. Side-effects:** When a module is invoked, it sometimes has a side effect of modifying the program state. Such side-effect should be avoided where as possible.

Common Programming Principles

Programming principles are guidelines and best practices that help developers write clean, maintainable, and efficient code. Here are 7 common programming principles

1. **KISS (Keep It Simple, Stupid)**
2. **DRY (Don't Repeat Yourself)**
3. **YAGNI (You Aren't Gonna Need It)**
4. **SOLID**
5. **Separation of Concerns (SoC)**
6. **Avoid Premature Optimization**
7. **Law of Demeter**

Common Programming Principles

- **1. KISS (Keep It Simple, Stupid)**
- Nobody in programming loves to debug, maintain, or make changes in complex code. It is very first principle, acronym stands for **Keep It Simple, Stupid**. The other famous alternate acronyms are
- Each unit should have only limited knowledge about other units: only units “closely” related to the current unit.
- Each unit should only talk to its friends; don’t talk to strangers.
- ***“Keep It Simple, Stupid (KISS)“*** states that most systems work best if they are kept simple rather than making it complex, so when you are writing code your solution should not be complicated that takes a lot of time and effort to understand. If your code is simple then other developers won’t face any problem understanding the code logic and they can easily proceed further with your code.
- **Example of KISS (Keep It Simple, Stupid)**

```
function mul ( a, b ) {
return a*b
}
```

- **2. DRY (*Don't Repeat Yourself)***
- Duplication of data, logic, or function in code not only makes your code lengthy but also wastes a lot of time when it comes to maintaining, debug or modify the code. If you need to make a small change in your code then you need to do it at several places.
- ***"Don't Repeat Yourself (DRY)"*** principal goal is to **reduce the repetition of code.** It states that a piece of code should be implemented in just one place in the source code.
- The opposite of the DRY principle is WET ("write everything twice" or "waste everyone's time") which breaks the DRY principle if you are writing the same logic at several places.
- **You can create a common function or abstract your code to avoid the repetition in your code.**

Common Programming Principles

- **Example of DRY (*Don't Repeat Yourself)***
- **//Convert Two Temperatures From Fahrenheit to Celsius before applying DRY principal**

```
input far1 from user
input far2 from user
calculate cel1 = (fah1-32) *5/9
calculate cel2= (fah2-32)*5/9
print cel1
print cel2
```

//Convert Two Temperatures From Fahrenheit to Celsius after applying DRY principal

```
input far1 from user
input far2 from user
calculate cel1 = cal ( f1)
calculate cel2= cal (f2)
print cel1
print cel2
function cal ( fah ) {
cel= (fah1-32) *5/9
return cel
}
```

- **3. YAGNI (*You Aren't Gonna Need It)***
- Your software or program can become larger and complex if you are writing some code which you may need in the future but not at the moment.
- ***"You Aren't Gonna Need It (YAGNI)"*** principle states that "**don't implement something until it is necessary**" because in most of the cases you are not going to use that piece of code in future.
- Most of the programmers while implementing software think about the future possibility and add some code or logic for some other features which they don't need at present.
- They add all the unnecessary class and functionality which they might never use in the future. Doing this is completely wrong and you will eventually end up in writing bloated code also your project becomes complicated and difficult to maintain.
- We recommend all the programmers to avoid this mistake to save a lot of time and effort.

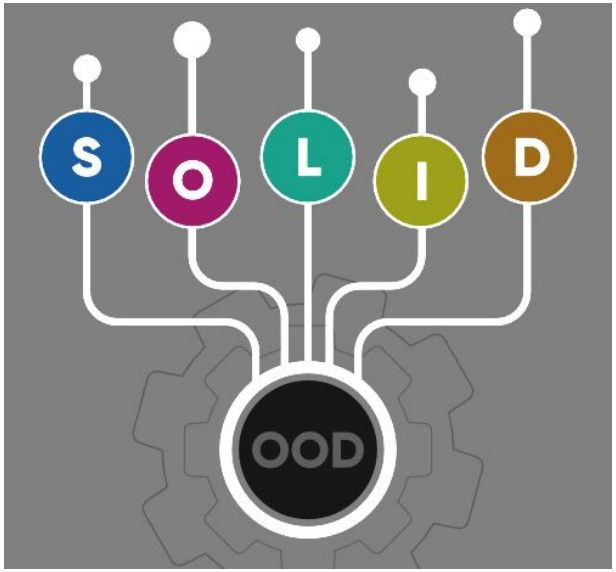
- **Example of YAGNI**
- **// If Programmer asked to write a Program to add 7 and 3**

```
function (x,y){
return x+y
}
```

- instead of this write what you have to told

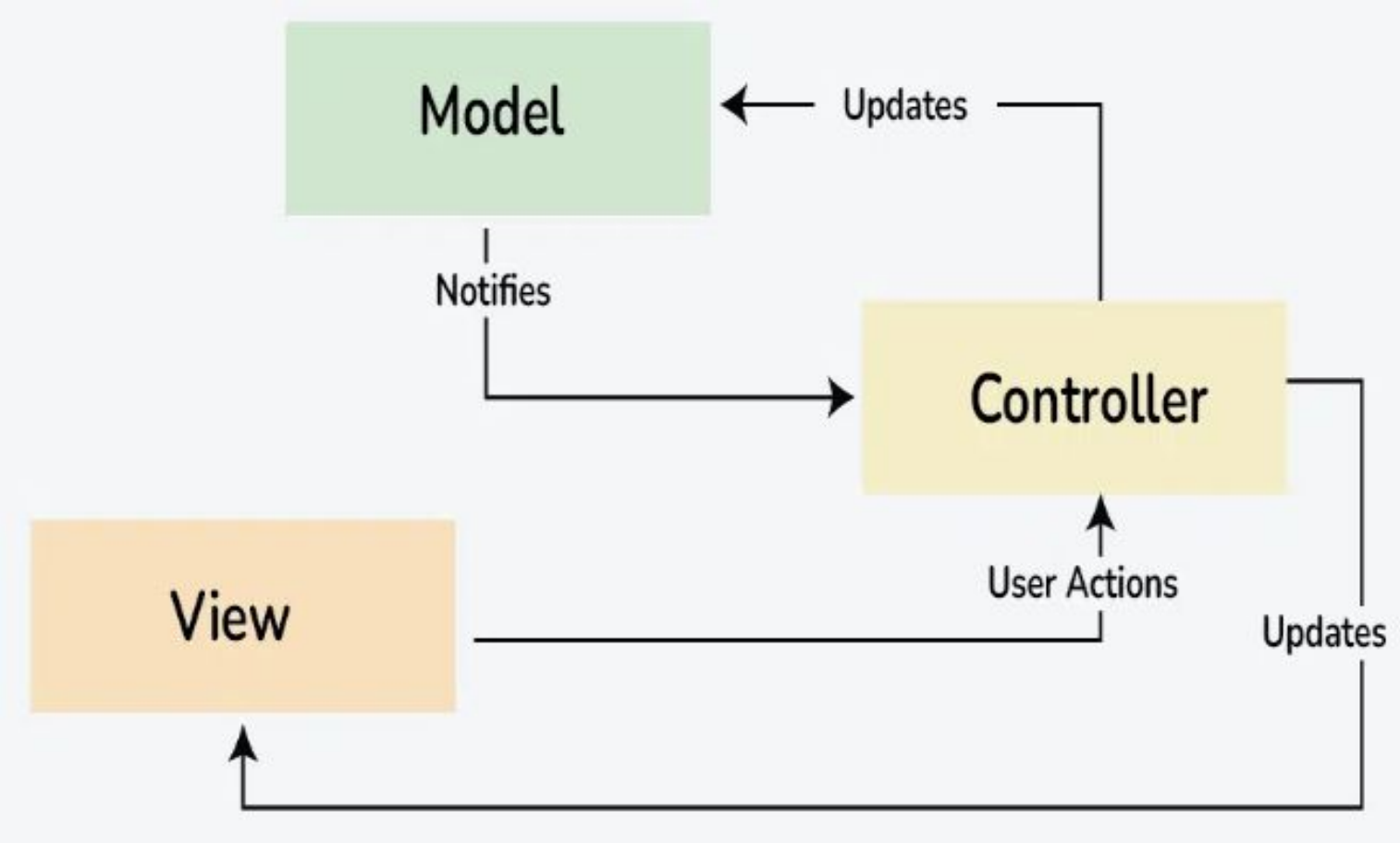
```
function ( 7, 3 ){
return 7+3
}
```

- **4. SOLID**
- The SOLID principle stands for five principles which are Single responsibility, Open-closed, Liskov substitution, Interface Segregation, and Dependency inversion.
- These principles are given by ***Robert C. Martin*** and you can check about these SOLID principle in detail.
- **This principle is an acronym of the five principles which is given below.**
- Single Responsibility Principle (SRP)
- Open/Closed Principle
- Liskov's Substitution Principle (LSP)
- Interface Segregation Principle (ISP)
- Dependency Inversion Principle (DIP)



- **5. Separation of Concerns (SoC):**
- Separation of Concerns Principle **partition a complicated application into different sections or domains**. Each section or domain addresses a separate concern or has a specific job.
- Each section is independent of each other and that's why each section can be tackled independently also it becomes easier to maintain, update, and reuse the code.
- **Example of SOC**
- **business logic** (the content of the webpage) in an application is a different concern and user interface is a different concern in a web application program.
- One of the good examples of SoC is the MVC pattern where data ("model"), the logic ("controller"), and what the end-user sees ("view") divided into three different sections and each part is handled independently. Saving of data to a database has nothing to do with rendering the data on the web.

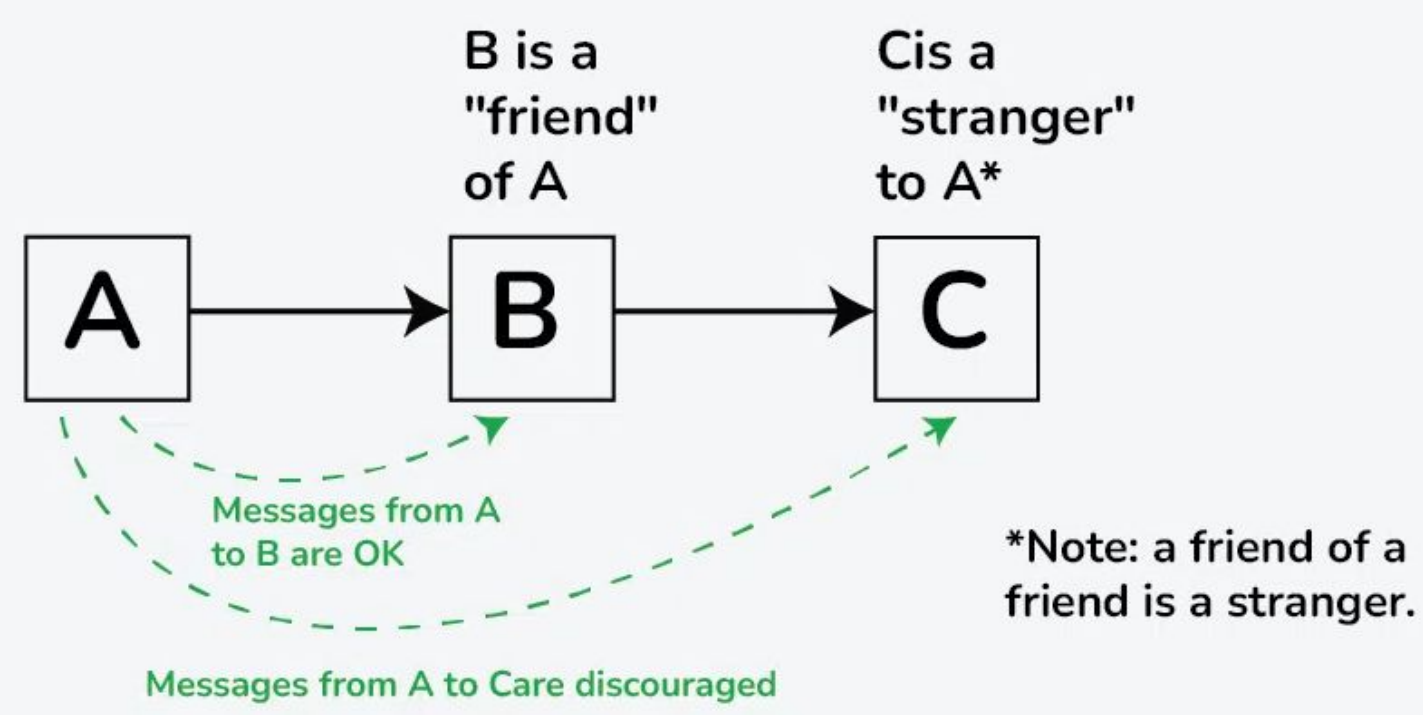
- **5. Separation of Concerns (SoC):**



- **6. Avoid Premature Optimization**
- Optimization indeed helps in **speeding up the program or algorithm** but according to this principle you don't need to **optimize your algorithm at an early stage of development.**
- If you do premature optimization you won't be able to know where a program's bottlenecks will be and maintenance will become harder for you.
- If you optimize your code in the beginning and case if the requirement may change than your efforts will be wasted and your code will go to the garbage.
- So it's better to optimize the **algorithm at the right time to get the right benefit of it.**
- Premature optimization is the root of all evil in programming. **–Donald Knuth**

- **7. Law of Demeter**
- This principle was first introduced by **Ian Holland** in 1987 at Northeastern University.
- It is also known as the ***principle of least knowledge*.**
- This principle divides the responsibility between classes or different units and it can be summarized in three points.
- Each unit should have only limited knowledge about other units: only units “closely” related to the current unit.
- Each unit should only talk to its friends; don’t talk to stranger (**not familiar person**).
- Only talk to your immediate friends.

- **7. Law of Demeter**



The Law of Demeter helps in maintaining independent classes and makes your code less which is very important in software development to make your application **flexible, stable, maintainable and understandable.**

More details

- https://pubhtml5.com/kcvf/ptcf/basic/51-100
- https://www.geeksforgeeks.org/7-common-programming-principles-that-every-developer-must-follow/