Virtual University

Dr. Muhammad Anwaar Saeed

Dr. Said Nabi

Ms. Hina Ishaq

CS621 Parallel and Distributed Computing

Concurrency Control

CS621 Parallel and Distributed Computing

Objectives

What is Concurrency?

Mechanisms for Concurrency Control.

Definition of Concurrency

"Concurrency is the task of running two or more computations over the same time interval. Two events are said to be concurrent if they occur within the same time interval."

What is Concurrency?

Concurrent doesn't necessarily mean at the same exact instant. For example, two tasks may occur concurrently within the same second but with each task executing within different fractions of the second.

Concurrent tasks can execute in a single or multiprocessing environment.

In a single processing environment, concurrent tasks exist at the same time and execute within the same time period by context switching.	In a multiprocessor environment, if enough processors are free, concurrent tasks may execute at the same instant over the same time period. The determining factor for what makes an acceptable time period for concurrency is relative to the application.

Concurrency Control

- There must be an implicit or explicit control over concurrency. It is both hazardous and unsafe when multiple flows of executions simultaneously operate in the same address space without any kind of agreement on ordered access. Two or more activities might access the same data and thus induce data corruption as well as inconsistent or invalid application state.
- Multiple activities that work jointly on a problem need an agreement on their common progress. Both issues represent fundamental challenges of concurrency and concurrent programming.

Concurrency Control Cont…

Synchronization and Coordination are two basic approaches to tackle this challenge:

- Synchronization is a mechanism that controls access on shared resources between multiple activities. It enforces exclusiveness and ordered access on the resource by different activities.
- Coordination aims at the orchestration of collaborating activities.

Benefits of Concurrency

- The overall time to perform the series of tasks is reduced.
- Concurrent processes can reduce duplication in code.
- The overall runtime of the algorithm can be significantly reduced.
- Concurrency control can also increase the scalability of parallel and distributed computing systems
- Redundancy can make systems more reliable.
- More real-world problems can be solved than with sequential algorithms alone.

Concurrency Control: Basic Approaches to Achieving Concurrency

CS621 Parallel and Distributed Computing

Objectives

Understanding of Concurrency Control.

Parallel Programming Technique.

Distributed Programming Technique.

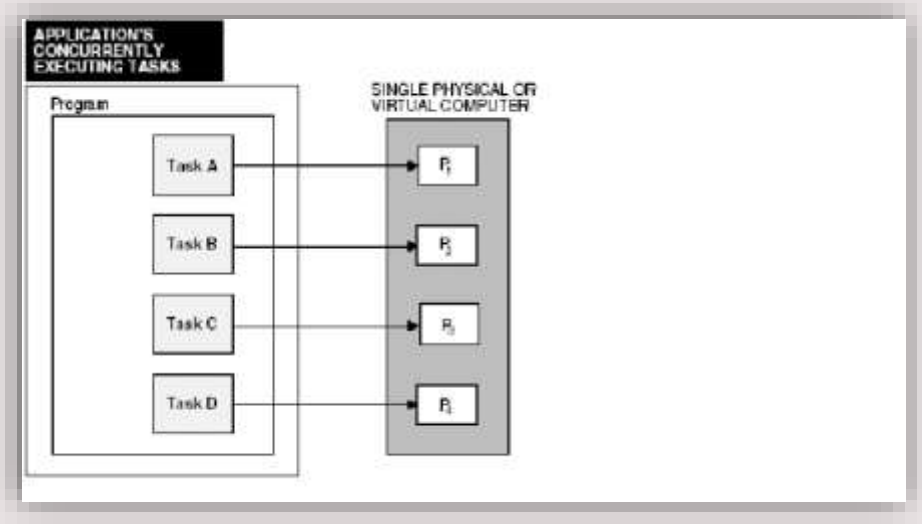
Achieving Concurrency

Parallel programming and distributed programming are two basic approaches for achieving concurrency:

- Parallel programming techniques assign the work a program has do to two or more processors within a single physical or a single virtual computer.
- Distributed programming techniques assign the work a program has to do to two or more processes where the processes may or may not exist on the same computer.

Achieving Concurrency: Parallel Programming Technique

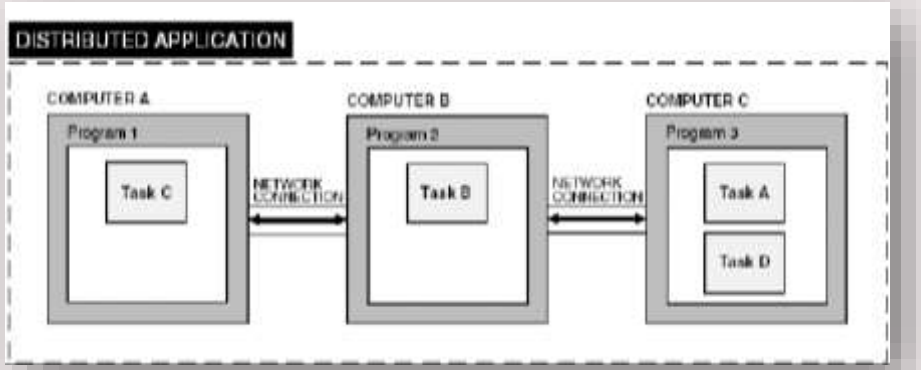
- The parallel application consists of one program divided into four tasks. Each task executes on a separate processor; therefore, each task may execute simultaneously. The tasks can be implemented by either a process or a thread.



Typical architecture for a parallel program.

Achieving Concurrency: Distributed Programming Technique

- The distributed application consists of three separate programs with each program executing on a separate computer. Program 3 consists of two separate parts that execute on the same computer. Although task A and D of Program 3 are on the same computer, they are distributed because they are implemented by two separate processes.



Typical architecture for a parallel and distributed program.

Concurrency Control: Models for Programming Concurrency

CS621 Parallel and Distributed Computing

Objectives

Models of Programming Concurrency.

Van Roy Approaches for Programming Concurrency.

Concurrency Control: Models for Programming Concurrency

Van Roy introduces four main approaches for programming concurrency:

- Sequential Programming.
- Declarative Concurrency.
- Message-passing Concurrency.
- Shared-state Concurrency.

Sequential Programming

In this deterministic programming model, no concurrency is used at all. In its strongest form, there is a total order of all operations of the program. Weaker forms still keep the deterministic behavior. However, they either make no guarantees on the exact execution order to the programmer a priori. Or they provide mechanisms for explicit preemption of the task currently active, as co-routines do, for instance.

Declarative Concurrency

Declarative programming is a programming model that favors implicit control flow of computations. Control flow is not described directly, it is rather a result of computational logic of the program. The declarative concurrency model extends the declarative programming model by allowing multiple flows of executions. It adds implicit concurrency that is based on a data-driven or a demand-driven approach. While this introduces some form of nondeterminism at runtime, the nondeterminism is generally not observable from the outside.

Message-passing Concurrency

This model is a programming style that allows concurrent activities to communicate via messages. Generally, this is the only allowed form of interaction between activities which are otherwise completely isolated. Message passing can be either synchronous or asynchronous resulting in different mechanisms and patterns for synchronization and coordination.

Shared-state Concurrency

Shared-state concurrency is an extended programming model where multiple activities are allowed to access contended resources and states. Sharing the exact same resources and states among different activities requires dedicated mechanisms for synchronization of access and coordination between activities. The general nondeterminism and missing invariants of this model would otherwise directly cause problems regarding consistency and state validity.

Memory Hierarchies

CS621 Parallel and Distributed Computing

Objectives

Introduction of Memory Hierarchy.

Characteristics of Memory Hierarchy.

Memory Hierarchies

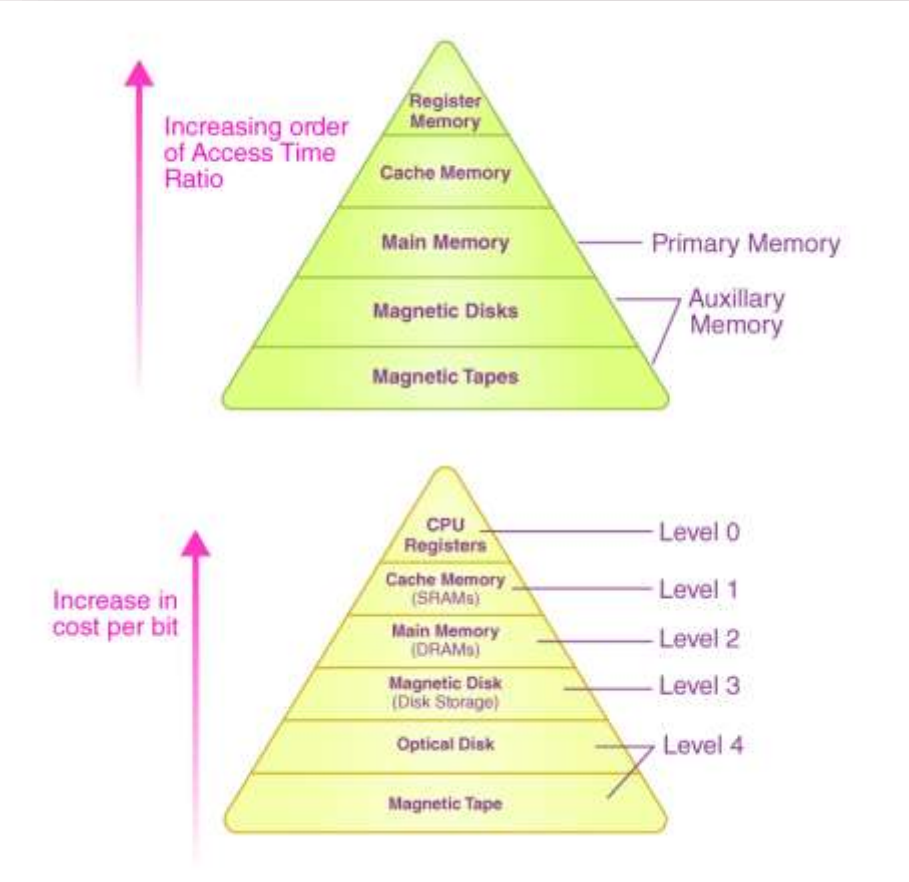
“Memory Hierarchy, in Computer System Design, is an enhancement that helps in organizing the memory so that it can minimize the access time. The development of the Memory Hierarchy occurred on a behavior of a program known as locality of references.”

Memory Hierarchies Cont…

We are concerned with five types of memory:

- **Registers:** are the fastest type of memory, which are located internal to a processor. These elements are primarily used for temporary storage of operands, small partitions of memory, etc., and are assumed to be one word (32 bits) in length in the MIPS architecture.
- **Cache:** is a very fast type of memory that can be external or internal to a given processor. Cache is used for temporary storage of blocks of data obtained from main memory (read operation) or created by the processor and eventually written to main memory (write operation).
- **Main Memory:** is modelled as a large, linear array of storage elements that is partitioned into static and dynamic storage. Main memory is used primarily for storage of data that a program produces during its execution, as well as for instruction storage.
- **Disk Storage:** is much slower than main memory, but also has much higher capacity than the preceding three types of memory.
- **Archival Storage:** is offline storage such as a CD-ROM jukebox or (in former years) rooms filled with racks containing magnetic tapes. This type of storage has a very long access time, in comparison with disk storage, and is also designed to be much less volatile than disk data.

Memory Hierarchies Cont…



Characteristics of Memory Hierarchy

Characteristics of a Memory Hierarchy can be inferred from the previous figure:

- **Capacity:** It refers to the total volume of data that a system's memory can store. The capacity increases moving from the top to the bottom in the Memory Hierarchy.
- **Access Time:** It refers to the time interval present between the request for read/write and the data availability. The access time increases as we move from the top to the bottom in the Memory Hierarchy.

Characteristics of Memory Hierarchy Cont..

Characteristics of a Memory Hierarchy can be inferred from the previous figure:

- **Performance:** When a computer system was designed earlier without the Memory Hierarchy Design, the gap in speed increased between the given CPU registers and the Main Memory due to a large difference in the system's access time. It ultimately resulted in the system's lower performance, and thus, enhancement was required. Such a kind of enhancement was introduced in the form of Memory Hierarchy Design, and because of this, the system's performance increased. One of the primary ways to increase the performance of a system is minimizing how much a memory hierarchy has to be done to manipulate data.
- **Cost per bit:** The cost per bit increases as one moves from the bottom to the top in the Memory Hierarchy, i.e. External Memory is cheaper than Internal Memory.

Limitations of Memory System Performance

CS621 Parallel and Distributed Computing

Objectives

Understanding of Limitations of Memory System Performance.

Memory Latency Example.

Limitations of Memory System Performance

Memory system, and not processor speed, is often the bottleneck for many applications.

Memory system performance is largely captured by two parameters, latency and bandwidth.

Latency: Is the time from the issue of a memory request to the time the data is available at the processor.	**Bandwidth:** Is the rate at which data can be pumped to the processor by the memory system.

Limitations of Memory System Performance Cont…

It is very important to understand the difference between latency and bandwidth.

Consider the example of a fire-hose. If the water comes out of the hose two seconds after the hydrant is turned on, the latency of the system is two seconds.

- Once the water starts flowing, if the hydrant delivers water at the rate of 5 gallons/second, the bandwidth of the system is 5 gallons/second.
- If you want immediate response from the hydrant, it is important to reduce latency.
- If you want to fight big fires, you want high bandwidth.

Memory Latency Example

Consider a processor operating at 1 GHz (1 ns clock) connected to a DRAM with a latency of 100 ns (no caches). Assume that the processor has two multiply-add units and is capable of executing four instructions in each cycle of 1 ns. The following observations follow:

- The peak processor rating is 4 GFLOPS.
- Since the memory latency is equal to 100 cycles and block size is one word, every time a memory request is made, the processor must wait 100 cycles before it can process the data.

Improving Effective Memory Latency Using Caches

CS621 Parallel and Distributed Computing

Objectives

Effect of Cache.

Effect of Cache with Example.

Improving Effective Memory Latency Using Caches

Caches are small and fast memory elements between the processor and DRAM.

This memory acts as a low-latency high-bandwidth storage.

If a piece of data is repeatedly used, the effective latency of this memory system can be reduced by the cache.

The fraction of data references satisfied by the cache is called the cache hit ratio of the computation on the system.

Cache hit ratio achieved by a code on a memory system often determines its performance.

Effect of Cache

Repeated references to the same data item correspond to temporal locality.

In our example, we had O(n2) data accesses and O(n3) computation. This asymptotic difference makes the above example particularly desirable for caches.

Reduce network congestion and improve overall performance.

Effect of Cache Example

Continue the previous example of memory latency, we introduce a cache of size 32 KB with a latency of 1 ns or one cycle. We use this setup to multiply two matrices A and B of dimensions 32 × 32(8KB or 1K words for each matrix). We have carefully chosen these numbers so that the cache is large enough to store matrices A and B, as well as the result matrix C.

- 1GHz processor, 4GFLOPS theoretical peak, 100ns memory Latency.
- Assume 1ns cache latency (full-speed cache)

Effect of Cache Example Cont…

The following observations can be made about the problem:

- Fetching the two matrices into the cache corresponds to fetching 2K words, which takes approximately 200 µs.
- Multiplying two n × n matrices takes 2n3 operations. For our problem, this corresponds to 64K operations, which can be performed in 16K cycles (or 16 µs) at four instructions per cycle.
- The total time for the computation is therefore approximately the sum of time for load/store operations and the time for the computation itself, i.e., 200 + 16 µs.
- This corresponds to a peak computation rate of 64K/216 or 303 MFLOPS.

Effect of Memory Bandwidth

CS621 Parallel and Distributed Computing

Objectives

Effect of Memory Bandwidth.

Effect of Memory Bandwidth Example.

Effect of Memory Bandwidth

Memory bandwidth is determined by the bandwidth of the memory bus as well as the memory units.

Memory bandwidth can be improved by increasing the size of memory blocks.

The performance of the CPU or GPU can also impact memory bandwidth.

Effect of Memory Bandwidth Example

Consider the same setup as in previous topic, except in this case, the block size is 4 words instead of 1 word. We repeat the dot-product computation in this scenario:

- Assuming that the vectors are laid out linearly in memory, eight FLOPs (four multiply-adds) can be performed in 200 cycles.
- This is because a single memory access fetches four consecutive words in the vector.
- Therefore, two accesses can fetch four elements of each of the vectors. This corresponds to a FLOP every 25 ns, for a peak speed of 40 MFLOPS.

Effect of Memory Bandwidth Example Cont…

It is important to note that increasing block size does not change latency of the system.

Physically, the scenario illustrated here can be viewed as a wide data bus (4 words or 128 bits) connected to multiple memory banks.

In practice, such wide buses are expensive to construct.

In a more practical system, consecutive words are sent on the memory bus on subsequent bus cycles after the first word is retrieved.

Effect of Memory Bandwidth Example Cont…

The above examples clearly illustrate how increased bandwidth results in higher peak computation rates.

The data layouts were assumed to be such that consecutive data words in memory were used by successive instructions (spatial locality of reference).

If we take a data-layout centric view, computations must be reordered to enhance spatial locality of reference.

Virtual University

Dr. Muhammad Anwaar Saeed

Dr. Said Nabi

Ms. Hina Ishaq

CS621 Parallel and Distributed Computing

Introduction to Load Balancing

CS621 Parallel and Distributed Computing

Objectives

Introduction of Load Balancing.

Issues in Load Balancing.

Introduction to Load Balancing

“The goal of partitioning is to distribute the computation load to each processing element such that all processing elements become neither overloaded nor idle and all processors can finish their computation at about the same time.”

Introduction to Load Balancing Cont…

Distributed computing system provides high performance environment that are able to provide high processing power.

Deals with distribution of processes among processors connected by a network.

For homogeneous parallel systems, the computation load is distributed as evenly as possible in a parallel computer.

For heterogeneous parallel system, the computation load is distributed according to the computing power of each processor.

Issues in Load Balancing

A load balancing strategy needs to resolve following issues:

- What load information (measurements) can be used.
- When to invoke balancing, i.e., conditions to balance.
- Which nodes make the load balancing decision.
- How should old data be handled.
- How load migrations are to be managed.

Mapping Techniques for Load Balancing

CS621 Parallel and Distributed Computing

Objectives

Understanding of Mapping Techniques for Load Balancing.

The need of Mapping Techniques for Load Balancing.

Mapping Techniques for Load Balancing

The computation domain is partitioned into several subdomains and then mapped onto processors of a parallel system with the objective that all tasks complete in the shortest amount of elapsed time.

In general, the number of subdomains equals to the number of processors in a parallel system.

In order to achieve a small execution time, the overheads of executing the tasks in parallel must be minimized.

Quality of Load Balancing Algorithm

The quality of load balancing algorithms can be measured by two factor:

- Number of steps: Needed to get the balanced state.
- Extent of loads: Moves over the link to which nodes are connected.

Mapping Techniques for Load Balancing

There are two key sources of overhead. The time spent in inter-process interaction is one source of overhead. Another important source of overhead is the time that some processes may spend being idle.

A good mapping of tasks onto processes must strive to achieve the twin objectives:

Reducing the amount of time processes spend in interacting with each other	Reducing the total amount of time some processes are idle while the others are engaged in performing some tasks.

Mapping Techniques for Load Balancing

Mapping techniques used in parallel algorithms can be broadly classified into two categories:

Static Mapping.	Dynamic Mapping.

The parallel programming paradigm and the characteristics of tasks and the interactions among them determine whether a static or a dynamic mapping is more suitable.

Static Mapping for Load Balancing

CS621 Parallel and Distributed Computing

Objectives

Understanding of Static Mapping for Load Balancing.

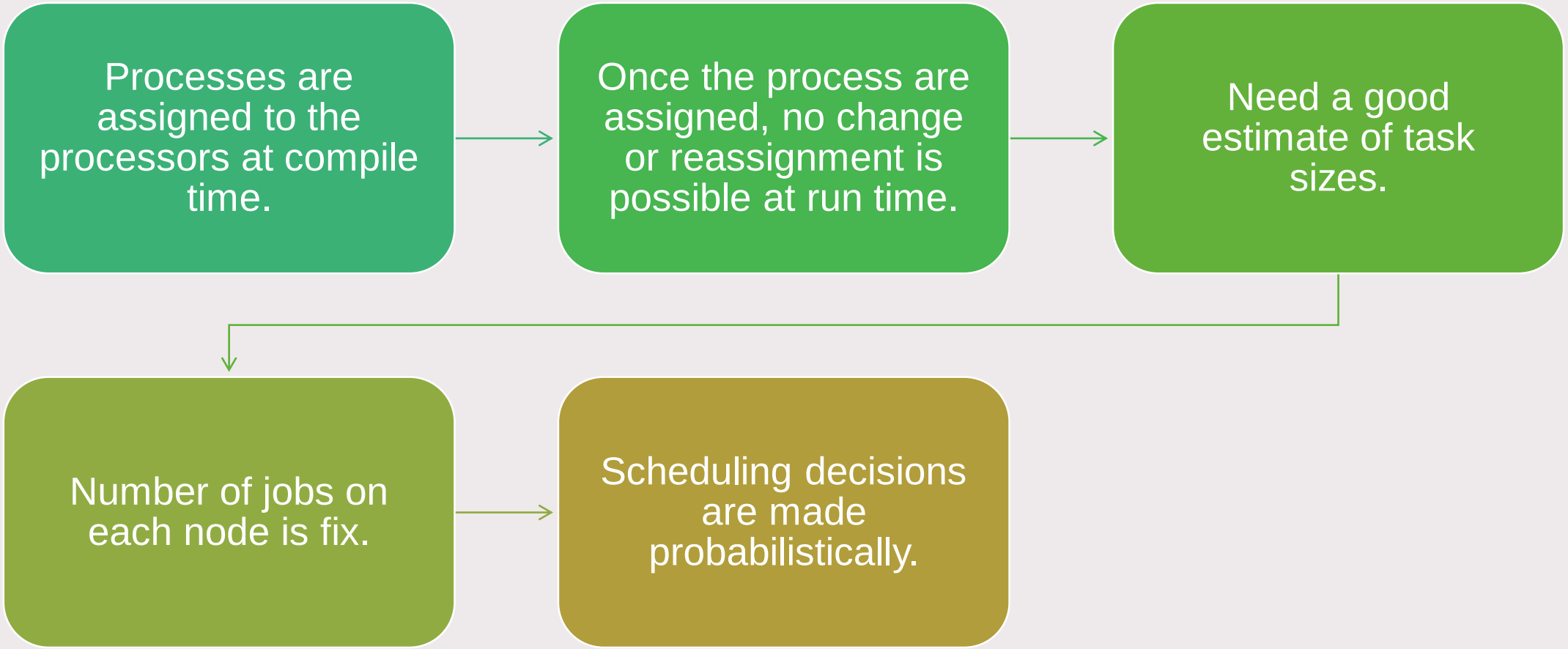
Advantages

Disadvantages

Static Mapping for Load Balancing

“Static mapping techniques distribute the tasks among processes prior to the execution of the algorithm.”

Static Mapping for Load Balancing Cont…



Advantages of Static Mapping

Algorithms that make use of static mapping are in general easier to design and program.

Since the mapping is fixed, there is no need for communication between processing nodes to determine task allocation. This reduces communication overhead and can improve performance.

Less network traffic due to load balancing related messages.

Disadvantages of Static Mapping

It is very difficult to compute a-priori execution time.

Lack of Fault Tolerance: Static mapping does not account for node failures, which can result in system downtime if a node fails.

The process allocation cannot be changed during execution.

Schemes for Static Mapping

CS621 Parallel and Distributed Computing

Objectives

Mappings Based on Data Partitioning.

Mappings Based on Task Partitioning.

Hierarchical Mappings.

Schemes for Static Mapping

Static mapping is often used in conjunction with a decomposition based on data partitioning.

Static mapping is also used for mapping certain problems that are expressed naturally by a static task-dependency graph. Mapping schemes based on data partitioning and task partitioning are:

- Mappings Based on Data Partitioning
- Mappings Based on Task Partitioning
- Hierarchical Partitioning

Mappings Based on Data Partitioning

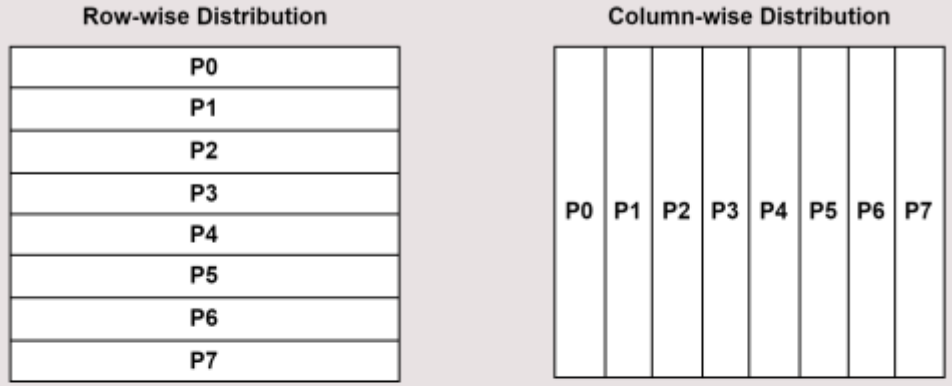
Mappings based on data partitioning's two of the most common ways of representing data in algorithms, namely, arrays and graphs.

- Block Distribution Schemes
  - Block Array Distributions
  - Cyclic and Block-Cyclic Distributions
  - Randomized Block Distributions

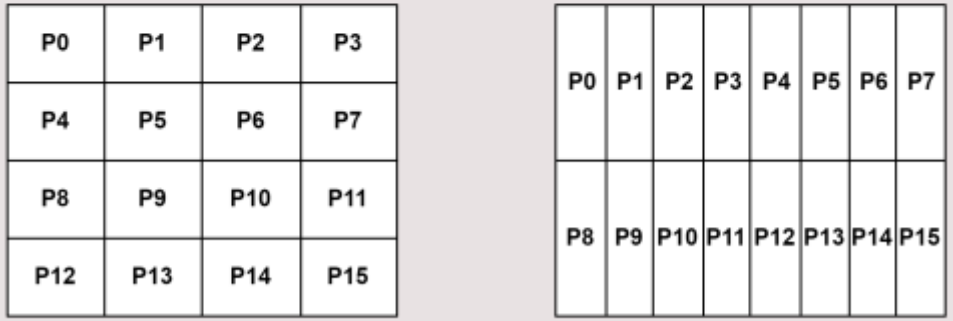
Mappings Based on Data Partitioning: Block Array Distribution

The data partitioning can be combined with the “owner-computes” rule to partition the computation into subtasks. The simplest data decomposition schemes for dense matrices are 1-D block distribution schemes.

In general, higher dimension decomposition allows the use of higher number of processes.



Examples of one-dimensional partitioning of an array among eight processes.



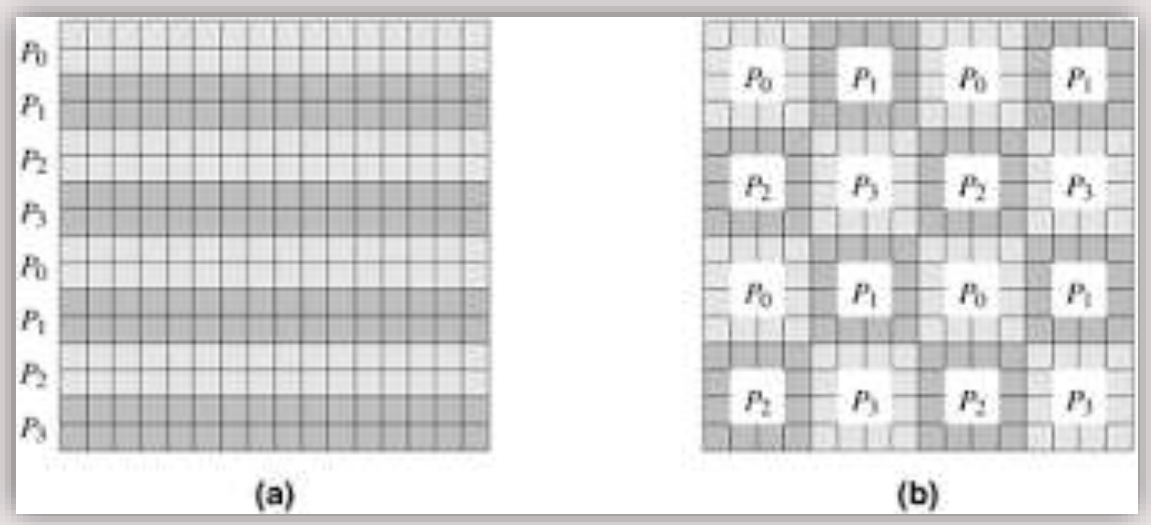
Examples of two-dimensional distributions of an array, (a) on a 4 x 4 process grid, and (b) on a 2 x 8 process grid.

Mappings Based on Data Partitioning: Cyclic and Block-Cyclic Distributions

The central idea behind a block-cyclic distribution is to partition an array into many more blocks than the number of available processes.

Then we assign the partitions (and the associated tasks) to processes in a round-robin manner so that each process gets several non-adjacent blocks.

More precisely, in a one-dimensional block-cyclic distribution of a matrix among p processes, the rows (columns) of an n x n matrix are divided into ap groups of n/(ap).



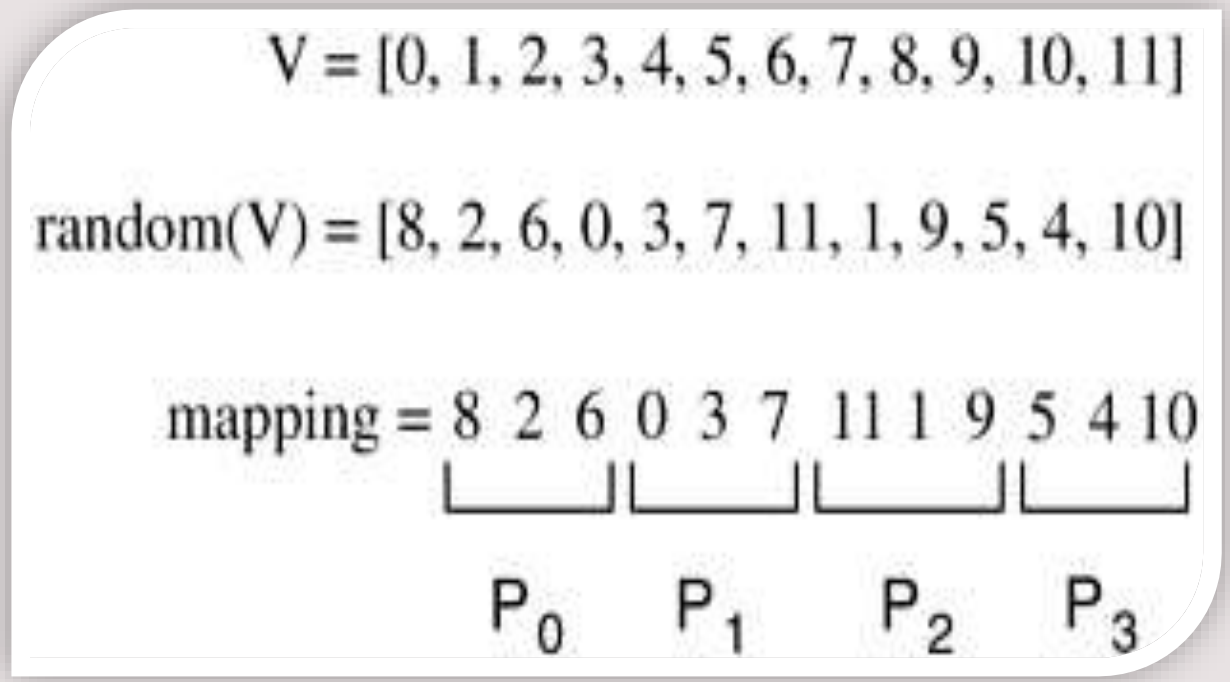
Examples of one- and two-dimensional block-cyclic distributions among four processes. (a) The rows of the array are grouped into blocks each consisting of two rows, resulting in eight blocks of rows. These blocks are distributed to four processes in a wraparound fashion. (b) The matrix is blocked into 16 blocks each of size 4 x 4, and it is mapped onto a 2 x 2 grid of processes in a wraparound fashion.

Mappings Based on Data Partitioning: Randomized Block Distribution

Just like a block-cyclic distribution, load balance is sought by partitioning the array into many more blocks than the number of available processes.

However, the blocks are uniformly and randomly distributed among the processes.

A one-dimensional randomized block mapping of 12 blocks onto four process (i.e., a = 3) is shown figure.



Schemes for Static Mapping Cont…

CS621 Parallel and Distributed Computing

Mappings Based on Task Partitioning

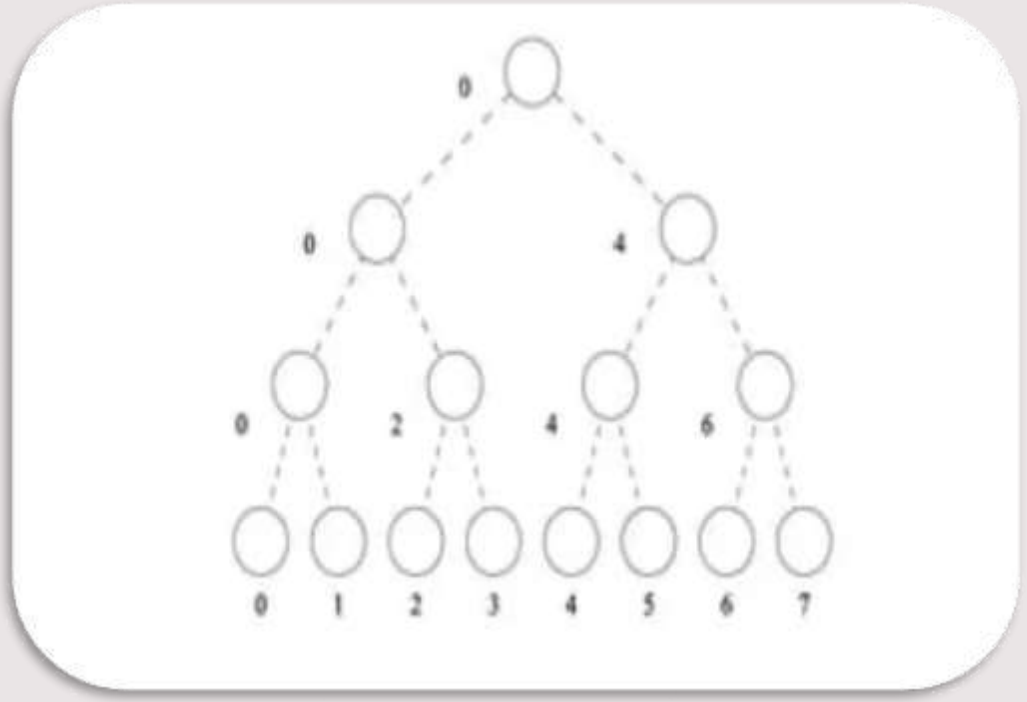
Partitioning a given task-dependency graph across processes.

Determining an optimal mapping for a general task-dependency graph is an NP-complete problem.

Excellent heuristics exist for structured graphs.

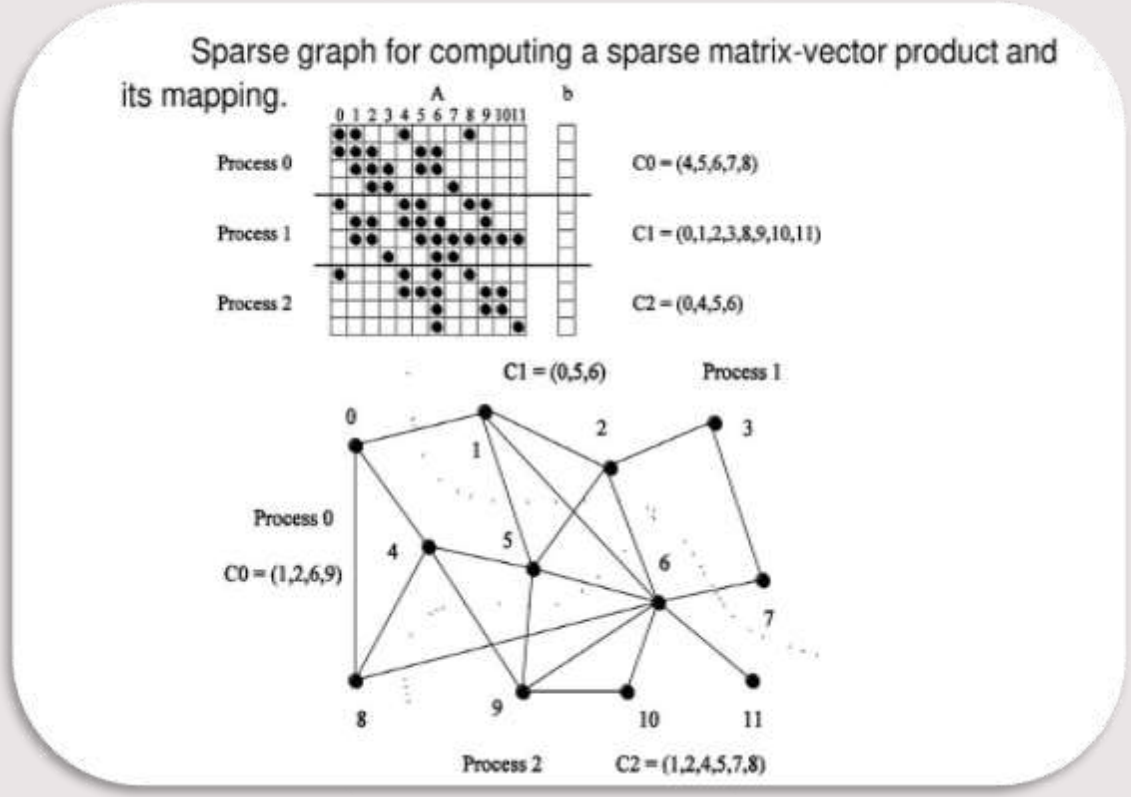
Mappings Based on Task Partitioning: Mapping a Binary Tree Dependency Graph

Example illustrates the dependency graph of one view of quick-sort and how it can be assigned to processes in a hypercube.



Mappings Based on Task Partitioning: Mapping a Sparse Graph

Reducing interaction overhead in sparse matrix-vector multiplication by partitioning the task-interaction graph.



Hierarchical Mappings

“A hierarchical mapping can have many layers and different decomposition and mapping techniques may be suitable for different layers.”

Hierarchical Mappings Cont…

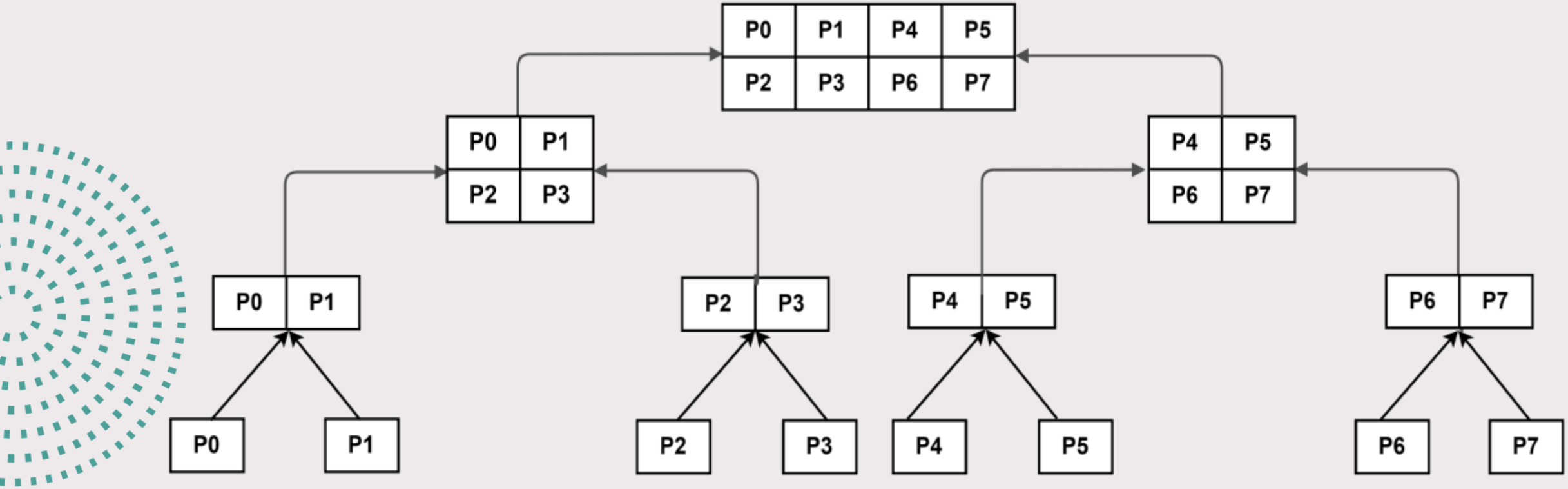
If the tasks are large enough, then a better mapping can be obtained by a further decomposition of the tasks into smaller subtasks.

A single mapping is inadequate.

Task mapping at the top layer and Data partitioning within each level.

Static mapping based on hierarchical mappings can be used in a variety of applications, including scientific simulations, machine learning, and image processing.

Hierarchical Mappings Cont…



An example of hierarchical mapping of a task-dependency graph. Each node represented by an array is a supertask. The partitioning of the arrays represents subtasks, which are mapped onto eight processes.

Dynamic Mapping for Load Balancing

CS621 Parallel and Distributed Computing

Objectives

Introduction of Dynamic Mapping.

Advantages of Dynamic Mapping.

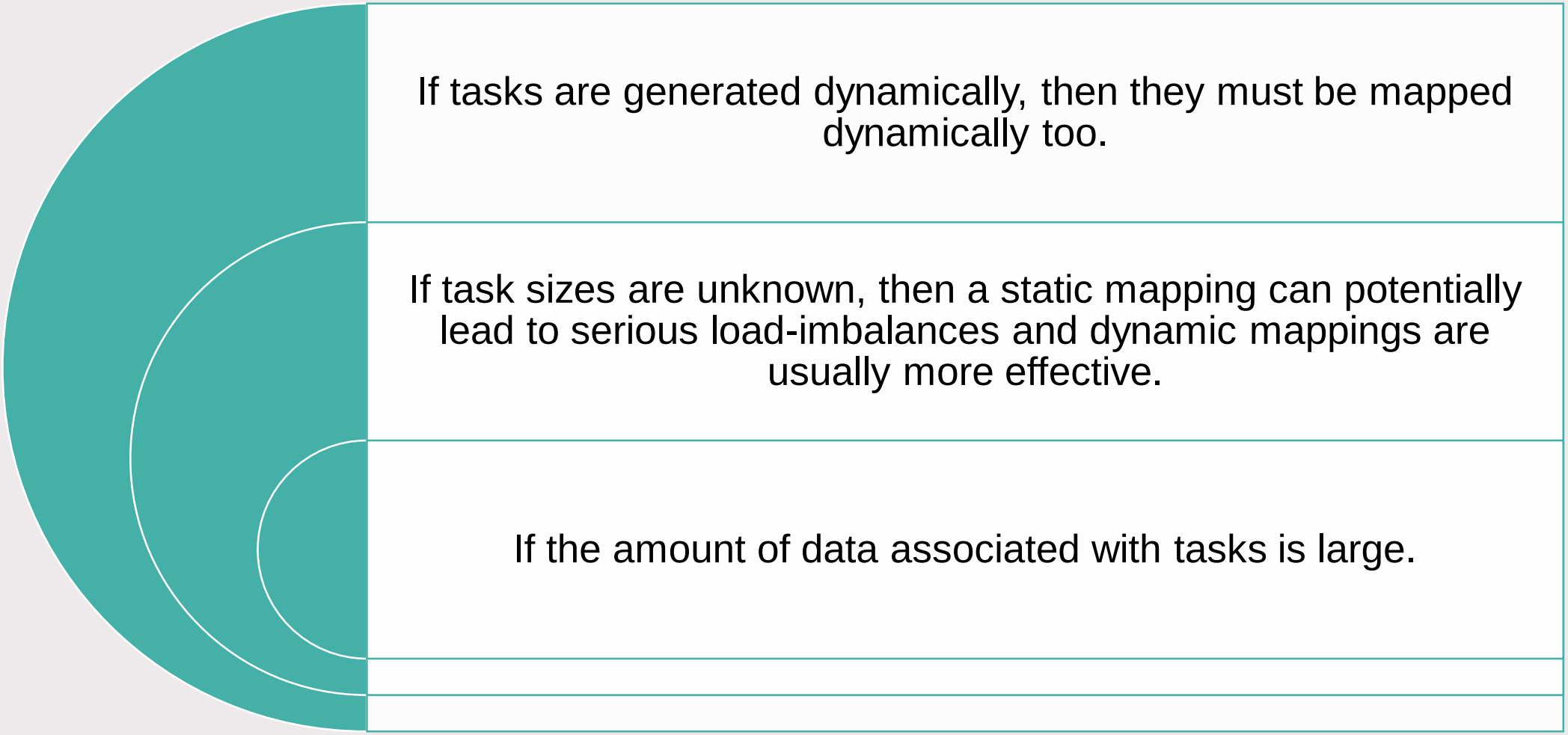
Disadvantages of Dynamic Mapping.

Dynamic Mapping for Load Balancing

“Dynamic mapping techniques distribute the work among processes

during the execution of the algorithm.”

Dynamic Mapping for Load Balancing Cont…



Advantages of Dynamic Mapping

- Greater Resource utilization.
- Enhanced Load Balancing
- The process allocation can be modified during execution if required.
- Scalability: Dynamic mapping can scale the system up or down based on the workload.

Disadvantages of Dynamic Mapping

Algorithms that require dynamic mapping are usually more complicated, particularly in the message-passing programming paradigm.

Greater overhead due to process redistribution.

Lack of determinism: Dynamic mapping introduces non-determinism into the system

Schemes for Dynamic Mapping

CS621 Parallel and Distributed Computing

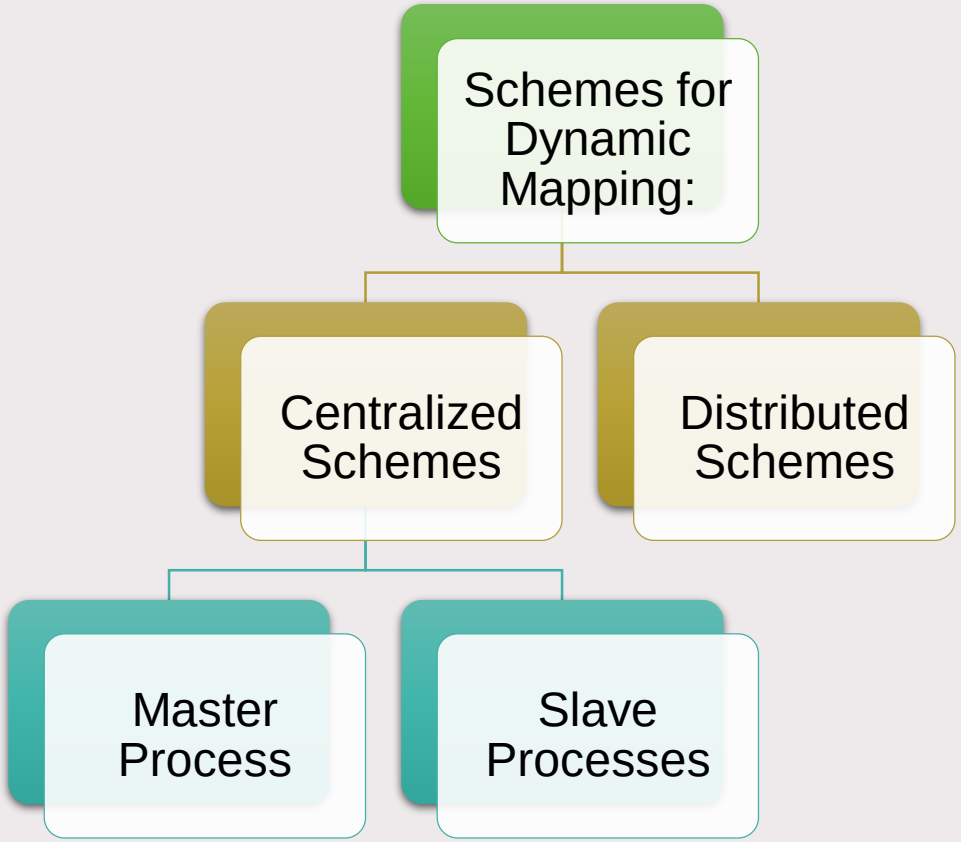
Objectives

Schemes for Dynamic Mapping.

Centralized Dynamic Mapping Scheme.

Distributed Dynamic Mapping Scheme.

Schemes for Dynamic Mapping



Centralized Dynamic Load Balancing Scheme

All executable tasks are maintained in a common central data structure, or they are maintained by a special process. If a special process is designated to manage the pool of available tasks, then it is often referred to as the master and the other processes that depend on the master to obtain work are referred to as slaves.

Whenever a process has no work, it takes a portion of available work from the central data structure or the master process.

When a new task is generated, it is added to this centralized data structure or reported to the master process.

Centralized Dynamic Load Balancing Scheme Cont…

Centralized load balancing schemes are usually easier to implement than distributed schemes.

If number of processes increases, master may become the bottleneck.

Chunk scheduling: A process picks up multiple tasks at once.

Large chunk sizes may lead to significant load imbalances as well.	Schemes to gradually decrease chunk size as the computation progresses.

Distributed Dynamic Load Balancing Scheme

“In a distributed dynamic load balancing scheme, the set of executable tasks are distributed among processes which exchange tasks at run time to balance work. Each process can send work to or receive work from any other process.”

Distributed Dynamic Load Balancing Scheme Cont…

Alleviates the bottleneck in centralized schemes. Critical parameters of a distributed load balancing scheme are as follows:

- How are the sending and receiving processes paired together?
- Is the work transfer initiated by the sender or the receiver?
- How much work is transferred in each exchange?
- When is the work transfer performed?

Virtual University

Dr. Muhammad Anwaar Saeed

Dr. Said Nabi

Ms. Hina Ishaq

CS621 Parallel and Distributed Computing

Introduction to Fault Tolerance

CS621 Parallel and Distributed Computing

Objectives

Introduction of Fault Tolerance.

Fault Classification.

Failure Classification.

Failure Masking.

Fault Tolerance

“A fault-tolerance system is one that continues to provide the required functionality in the presence of fault/failure.”

Fault Tolerance Cont…

A characteristic feature of distributed systems is the notion of partial failure:

- A partial failure may happen when one component in a distributed system fails.
- This failure may affect the proper operation of other components, while at the same time leaving yet other components totally unaffected.

An important goal in distributed systems design is to construct the system in such a way that it can automatically recover from partial failures without seriously affecting the overall performance.

Fault Classification

Faults are generally classified as transient, intermittent, or permanent:

- **Transient fault:** Occurs once and then disappear. If the operation is repeated, the fault goes away.
- **Intermittent fault:** Occurs, then vanishes of its own accord, then reappears, and so on. A loose contact on a connector will often cause an intermittent fault.
- **Permanent fault:** Is one that continues to exist until the faulty component is replaced. Burnt-out chips, software bugs, and disk head crashes are examples of permanent faults.

Failure Classification

Faults are generally classified into five categories:

- **Crash failure:** A server halts but working correctly until it halts.
- **Omission failure:** A server fails to respond to incoming requests.
- **Timing failure:** A server's response lies outside the specified time interval.
- **Response failure:** A server's response is incorrect.
- **Arbitrary failure:** A server may produce the arbitrary responses at arbitrary times.

Failure Masking

“Failure masking is a fault tolerance technique that hides occurrence of failures from other processes.”

Failure Masking Cont…

The most common approach to failure masking is redundancy which is categorized into three types:

- **Information redundancy:** Add extra bits to allow recovery from garbled bits.
- **Time redundancy:** Repeat an action if needed.
- **Physical redundancy:** Add extra equipment or processes so that the system can tolerate the loss or malfunctioning of some components.

Process Resilience

CS621 Parallel and Distributed Computing

Objectives

Introduction of Process Resilience.

Flat Groups versus Hierarchical Groups.

Failure Masking and Replications.

Approaches for Replications.

Process Resilience

“Process resilience incorporates techniques by which one or more processes can fail without seriously disturbing the rest of the system.”

Process Resilience Cont…

Related to this issue is reliable multicasting, by which message transmission to a collection of processes is guaranteed to succeed.

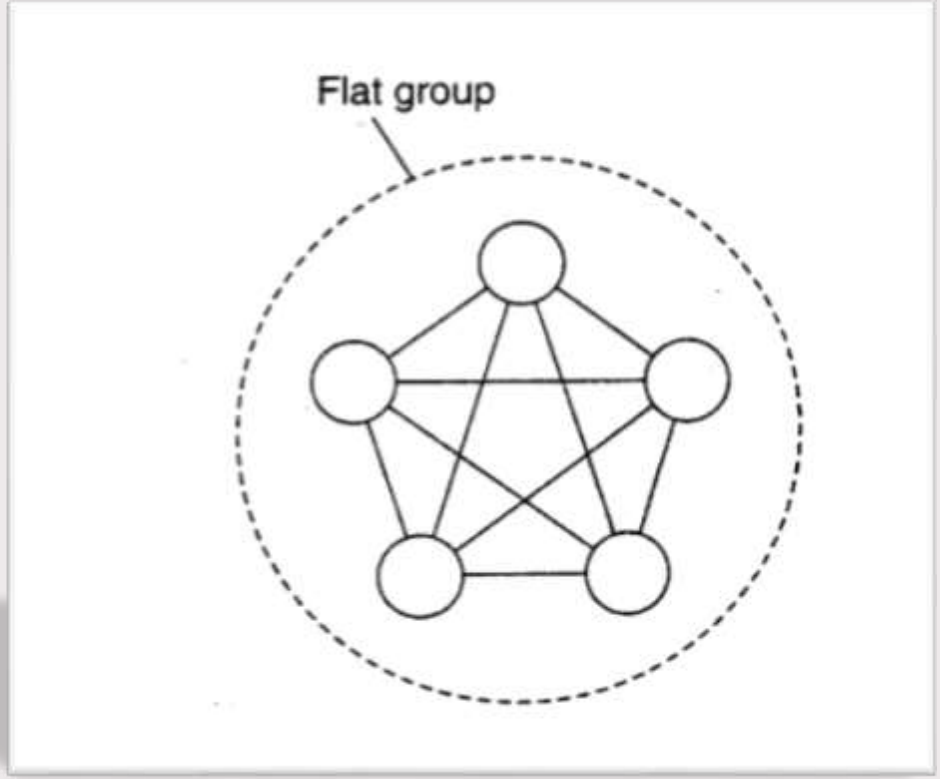
Reliable multicasting is often necessary to keep processes synchronized.

Protection against process failures can be achieved by process replication, organizing several identical processes into a group.

Groups are categorized into two categories: Flat Group and Hierarchy Group.

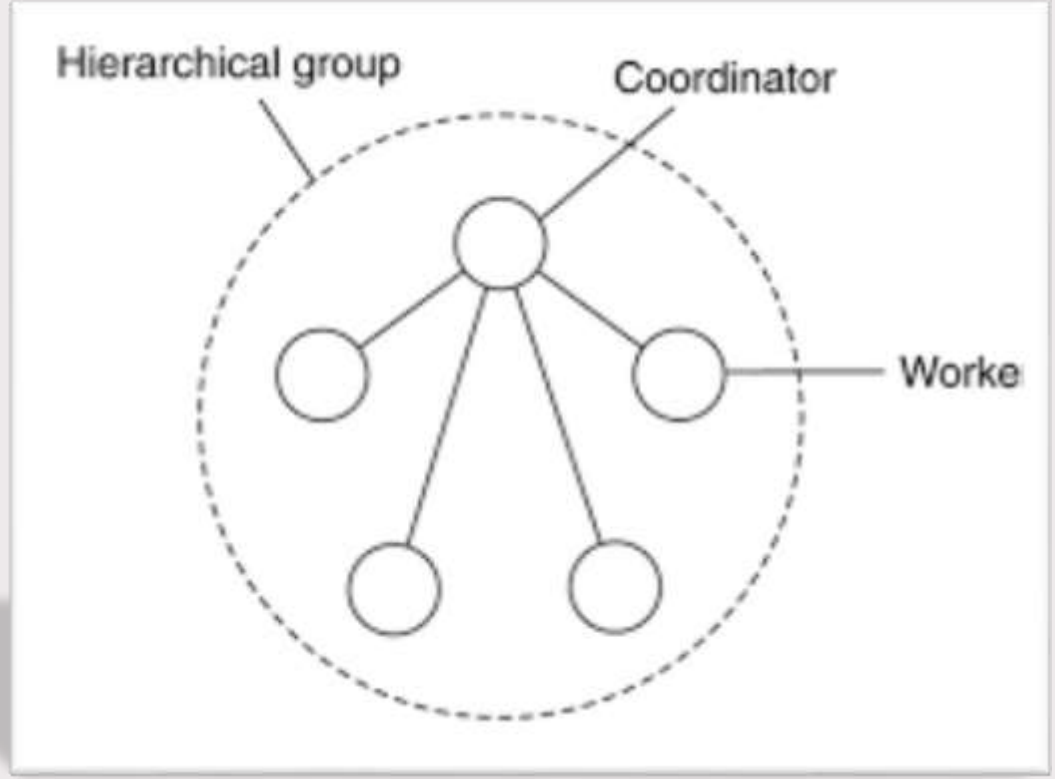
Flat Group

- All processes are equal.
- The processes make decisions collectively.
- No single point of failure, but decision making is more complicated as consensus is required.



Hierarchical Group

- A single coordinator makes all decisions.
- Single point-of failure, however: decisions are easily and quickly made by the coordinator without first having to get consensus.
- Group is transparent to its users; the whole group is dealt with as a single process.



Failure Masking and Replication

By organizing a fault tolerant group of processes , we can protect a single vulnerable process.

Two approaches to arranging the replication of the group are:

- Primary-base protocols and Replicated-write protocols.

Primary-Base Protocols

Appears in the form of a primary-backup protocol.

A group of processes is organized in a hierarchical fashion in which a primary coordinates all write operations.

When the primary crashes, the backups execute some election algorithm to choose a new primary.

Replicated-Write Protocols

Replicated-write protocols are used in the form of active replication, as well as by means of quorum-based protocols.

Solutions correspond to organizing a collection of identical processes into a flat group.

These groups have no single point of failure, at the cost of distributed coordination.

Reliable Client-Server Communication

CS621 Parallel and Distributed Computing

Objectives

Understanding of Reliable Client-Server Communication.

RPC Semantics in the Presence of Failures.

Reliable Client-Server Communication

Fault tolerance in distributed systems concentrates on faulty processes.

However, communication failures should also be considered .

A communication channel may exhibit crash, omission, timing, and arbitrary failures.

Peer to Peer Communication

Reliable point-to-point communication is established by making use of a reliable transport protocol, such as TCP.

TCP masks omission failures, which occur in the form of lost messages by using acknowledgments and retransmissions.

Crash failures of connections are not masked. The only way to mask such failures is to let the distributed system attempt to automatically set up a new connection.

RPC Semantics in the Presence of Failures

Remote Procedure Call (RPC) mechanism works well as long as both the client and server function perfectly.

Five classes of RPC failure can be identified:

The client is unable to locate the server.	The request message from the client to the server is lost.	The server crashes after receiving a request.	The reply message from the server to the client is lost.	The client crashes after sending a request.

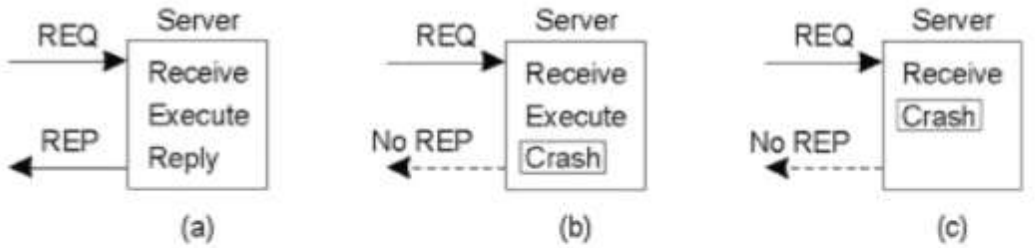
Server in Client-Server Communication

The sequence of events at a server is shown in Fig.

(a) A request arrives, is carried out, and a reply is sent.

(b) A request arrives and is carried out, just as before, but the server crashes before it can send the reply.

(c) Again, a request arrives, but this time the server crashes before it can even be carried out and no reply is sent back.



A server in client-server communication

a) Normal case

b) Crash after execution

c) Crash before execution

Server in Client-Server Communication Cont..

Server crashes are dealt with by implementing one of three possible implementation philosophies:

- **At least once semantics:** A guarantee is given that the RPC occurred at least once, but (also) possibly more that once.
- **At most once semantics:** A guarantee is given that the RPC occurred at most once, but possibly not at all.
- **No semantics:** Nothing is guaranteed, and client and servers take their chances.

Client in Client-Server Communication

When a client sends a request to a server and crashes before the server replies.At this point a computation is active and no parent is waiting for the result. Such an unwanted computation is called an orphan. Four orphan solutions have been proposed:

- **Extermination:** The orphan is simply killed-off.
- **Reincarnation:** Each client session has an epoch associated with it, making orphans easy to spot.
- **Gentle reincarnation:** When a new epoch is identified, an attempt is made to locate a requests owner, otherwise the orphan is killed.
- **Expiration:** If the RPC cannot be completed within a standard amount of time, it is assumed to have expired.

Reliable Group Communication

CS621 Parallel and Distributed Computing

Objectives

Understanding of Reliable Group Communication.

Reliable-Multicasting Schemes.

Reliable Group Communication

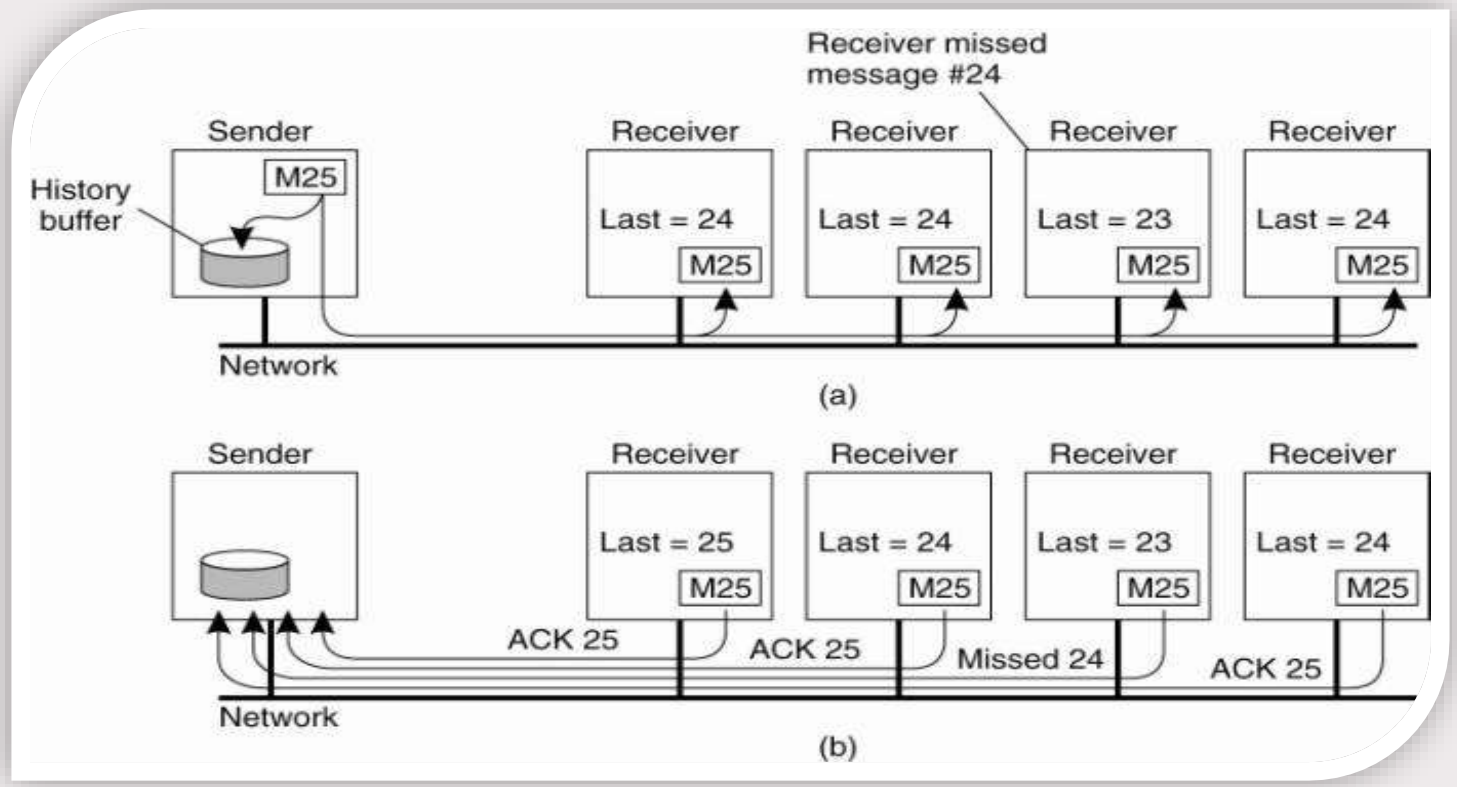
“Reliable multicast services guarantee that all messages are delivered to all members of a process group.”

Basic Reliable-Multicasting Schemes

A simple solution to reliable multicasting when all receivers are known and are assumed not to fail.

- The sending process assigns a sequence number to each message it multicasts.
- Assume that messages are received in the order they are sent.
- Each multicast message is stored locally in a history buffer at the sender.
- Assuming the receivers are known to the sender, the sender simply keeps the message in its history buffer until each receiver has returned an acknowledgment.
- If a receiver detects it is missing a message, it may return a negative acknowledgment, requesting the sender for a retransmission.

Basic Reliable-Multicasting Schemes Cont..



(a) Message transmission – note that the third receiver is expecting 24.

(b) Reporting feedback – the third receiver informs the sender.

Distributed Commit

CS621 Parallel and Distributed Computing

Objectives

Introduction of Distributed Commit.

Distributed Commit Protocol Phases.

Distributed Commit

“The distributed commit problem involves having an operation being performed by each member of a process group, or none at all.”

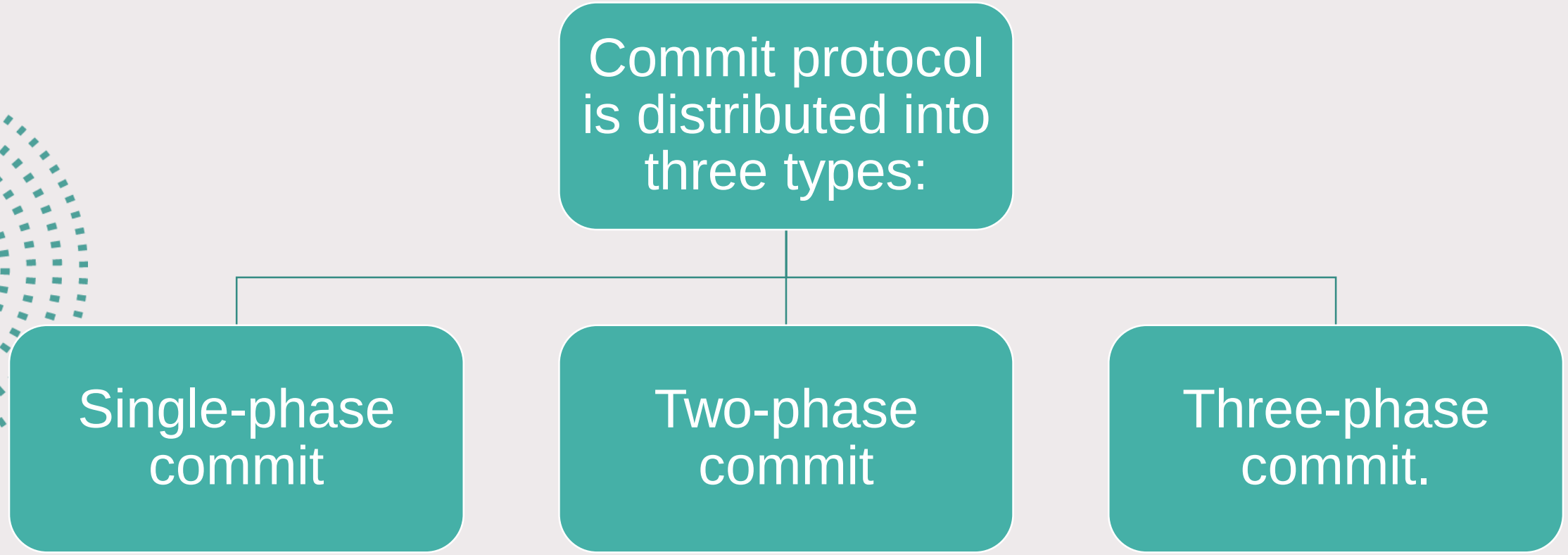
Distributed Commit Cont…

In the case of reliable multicasting, the operation is the delivery of a message.

With distributed transactions, the operation may be the commit of a transaction at a single site that takes part in the transaction.

Other examples of distributed commit, and how it can be solved are discussed in Tanisch (2000).

Distributed Commit Cont …



One-Phase Commit Protocol:

Coordinator tells all other processes that are also involved, called participants, whether to (locally) perform the operation in question.

If one of the participants cannot perform the operation, there is no way to tell the coordinator.

It cannot efficiently handle the failure of the coordinator.

The solutions: Two-Phase and Three-Phase Commit Protocols

Two-Phase Commit Protocol

“Assuming that no failures occur, the protocol consists of the following two phases, each consisting of two steps: The first phase is the voting phase, and the second phase is the decision phase.”

Two-Phase Commit Protocol Cont…

The coordinator sends a VOTE_REQUEST message to all participants.

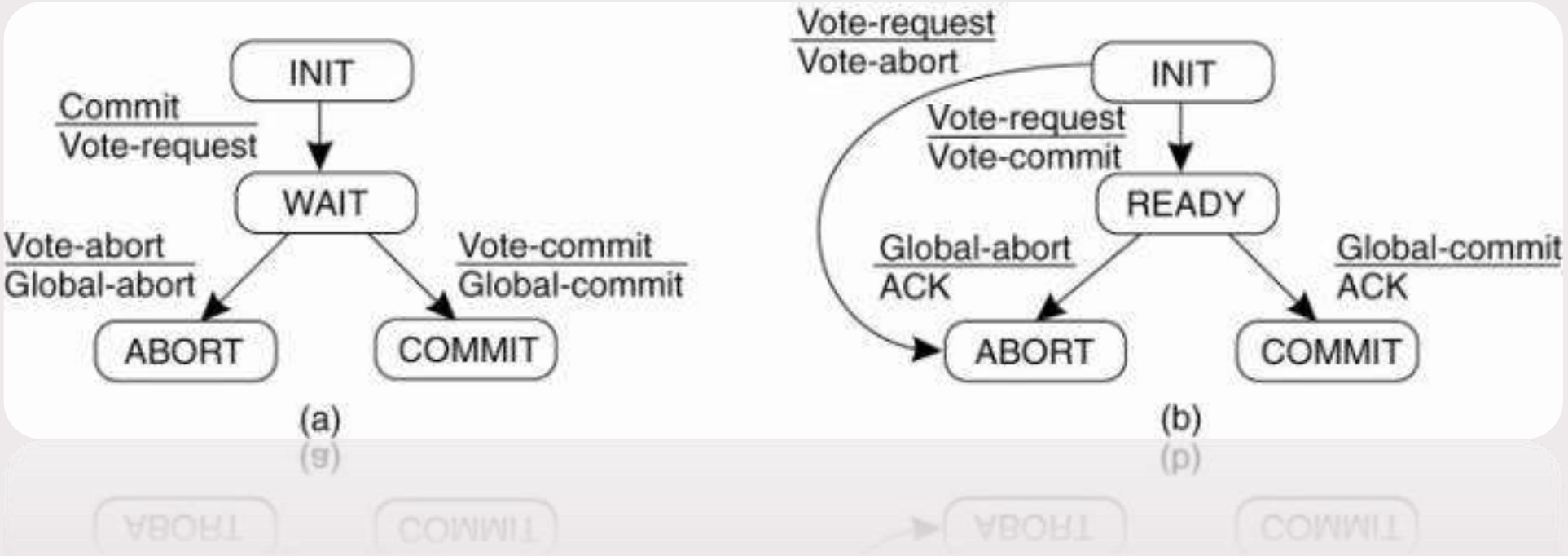
A group member returns VOTE_COMMIT if it can commit locally, otherwise VOTE_ABORT message.

All votes are collected by the coordinator.

- A GLOBAL_COMMIT is sent if all the group members voted to commit.
- If one group member voted to abort, a GLOBAL_ABORT is sent.

Group members then COMMIT or ABORT based on the last message received from the coordinator.

Two-Phase Commit Protocol Cont…



(a) The finite state machine for the coordinator in 2PC.

(b) The finite state machine for a participant.

Drawbacks of Two-Phase Commit Protocol

It can lead to both the coordinator and the participants blocking, which may lead to the dreaded deadlock.

If the coordinator crashes, the participants may not be able to reach a final decision, and they may, therefore, block until the coordinator recovers.

Two-Phase Commit is known as a blocking-commit protocol for this reason.

The solution: Three-Phase Commit Protocol

Three-Phase Commit Protocol

Skeen (1981) developed a variant of 2PC, called the three-phase commit protocol (3PC), that avoids blocking processes in the presence of fail-stop crashes.

The states of the coordinator and each participant satisfy the following two conditions:

- There is no single state from which it is possible to make a transition directly to either a COMMIT or an ABORT state.
- There is no state in which it is not possible to make a final decision, and from which a transition to a COMMIT state can be made.

Three-Phase Commit Protocol Cont…

(a) The finite state machine for the coordinator in 3PC.
(b) The finite state machine for a participant.

Recovery

CS621 Parallel and Distributed Computing

Objectives

Basic Concept of Recovery.

Types of Recovery.

Recovery

“The whole idea of error recovery is to replace an erroneous state with an error-free state. Once a failure has occurred, it is essential that the process where the failure happened recovers to a correct state.”

Recovery Cont…

Recovery from an error is fundamental to fault tolerance. Two main forms of recovery are:

- **Backward Recovery:** Return the system to some previous correct state (using checkpoints), then continue executing.
- **Forward Recovery:** When the system has entered an erroneous state, instead of moving back to a previous, checkpointed state, an attempt is made to bring the system in a correct new state from which it can continue to execute.

Backward Recovery

Advantages:

- Generally applicable method independent of any specific system or process.
- It can be integrated into (the middleware layer) of a distributed system as a general-purpose service.

Disadvantages:

- Restoring a system or process to a previous state is generally a relatively costly operation in terms of performance.
- Backward error recovery mechanisms are independent of the distributed application for which they are actually used, no guarantees can be given that once recovery has taken place, the same or similar failure will not happen again.

Forward Recovery

Advantages:

- Generally, have low overhead.

Disadvantages:

- It has to be known in advance which errors may occur. Only in that case is it possible to correct those errors and move to a new state.
- When an error occurs, the recovery mechanism then knows what to do to bring the system forward to a correct state.

Virtual University

Dr. Muhammad Anwaar Saeed

Dr. Said Nabi

Ms. Hina Ishaq

CS621 Parallel and Distributed Computing

Shared Memory

CS621 Parallel and Distributed Computing

Objectives

Introduction of Shared Memory.

Architecture of Shared Memory.

Shared Memory:

“Shared memory is a type of memory architecture that allows multiple processors or threads to access the same memory space. In the context of distributed and parallel computing, shared memory can be used to facilitate communication and synchronization between different processes or threads.”

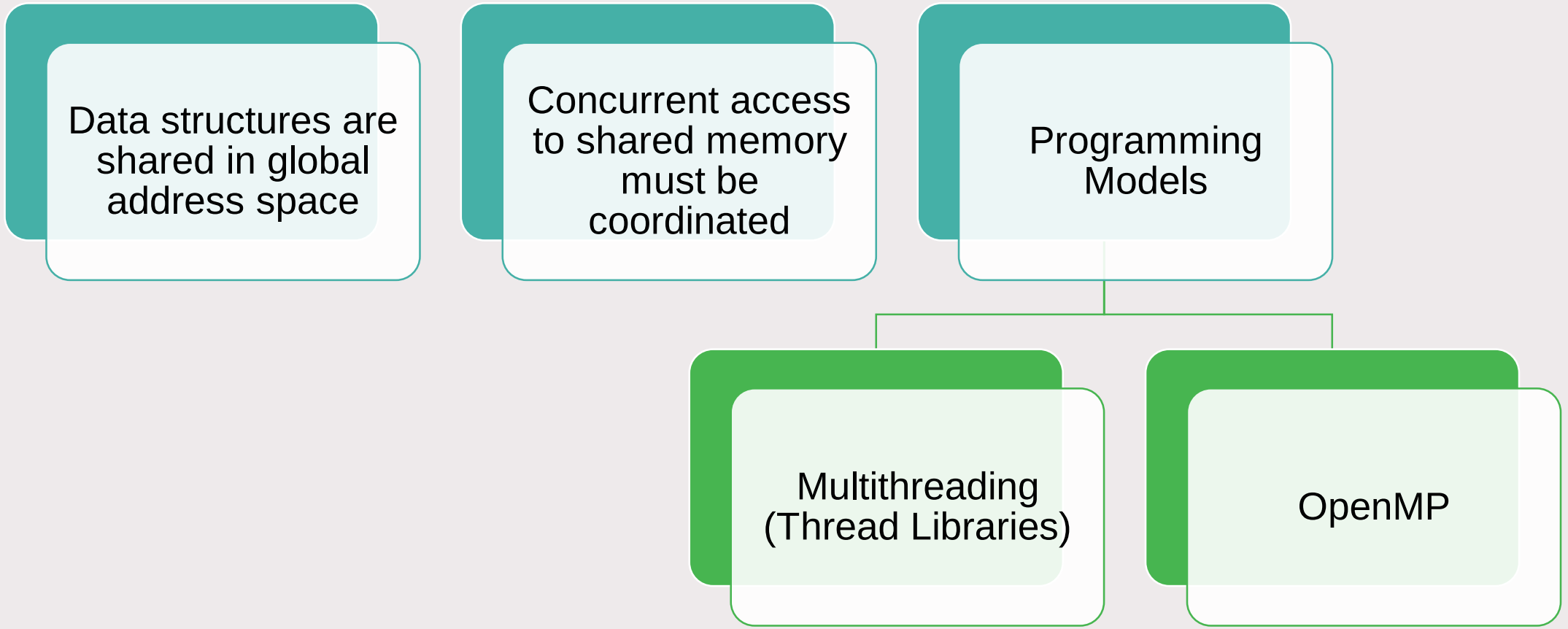
Shared Memory:

Processors have direct access to global memory and I/O through bus or fast switching network

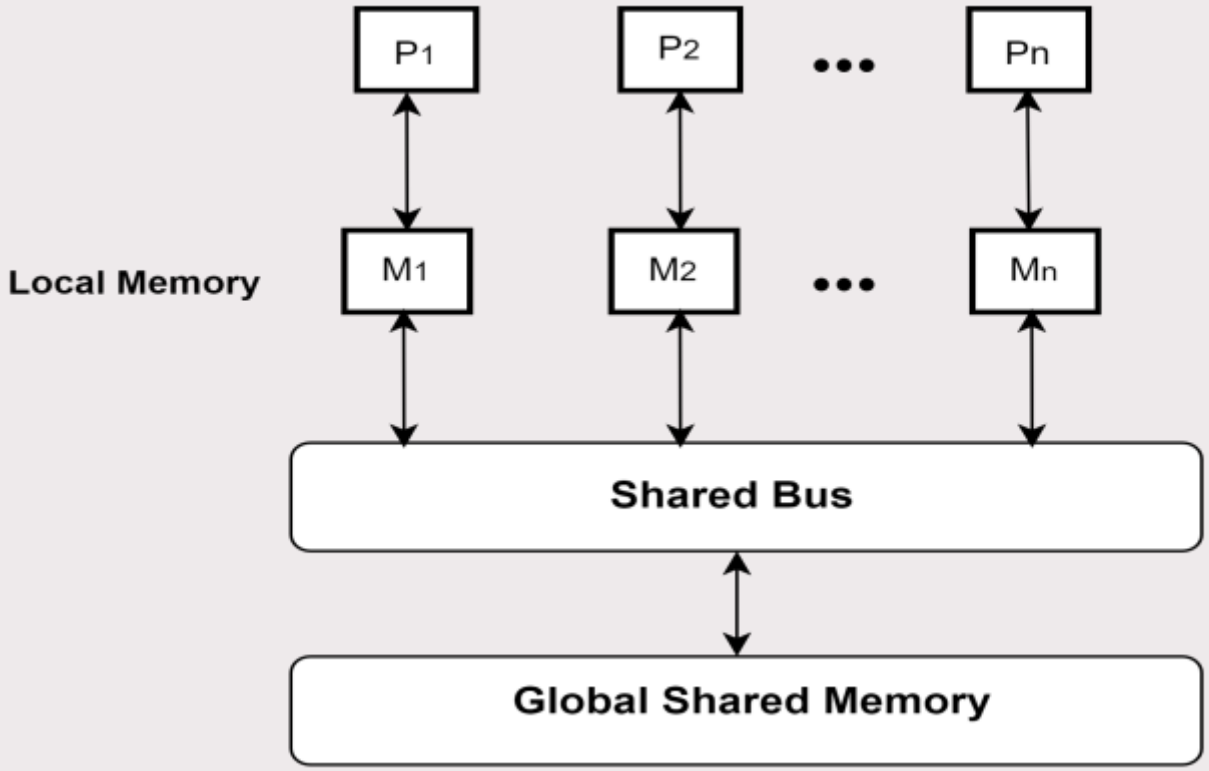
Cache Coherency Protocol guarantees consistency of memory and I/O accesses

Each processor also has its own memory (cache)

Shared Memory Cont…



Architecture of Shared Memory:



Distributed Memory

CS621 Parallel and Distributed Computing

Objectives

Introduction of Distributed Memory.

Architecture of Distributed Memory.

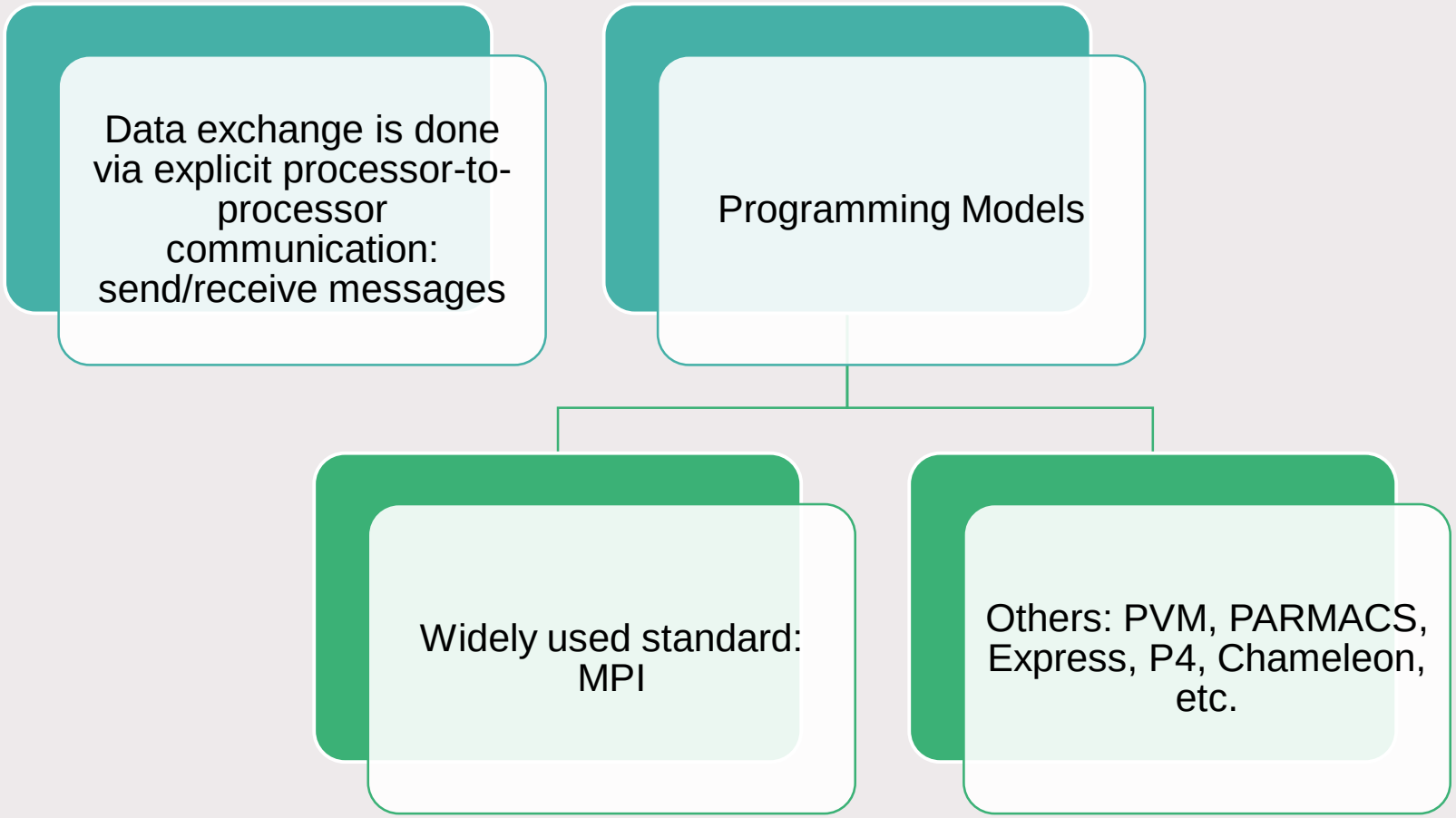
Distributed Memory:

“Distributed memory refers to a type of parallel computing architecture where each processor has its own private memory, and communication between processors happens through message passing. In this architecture, the memory of one processor is not directly accessible by other processors, and communication between processors occurs explicitly through messages that are sent and received.”

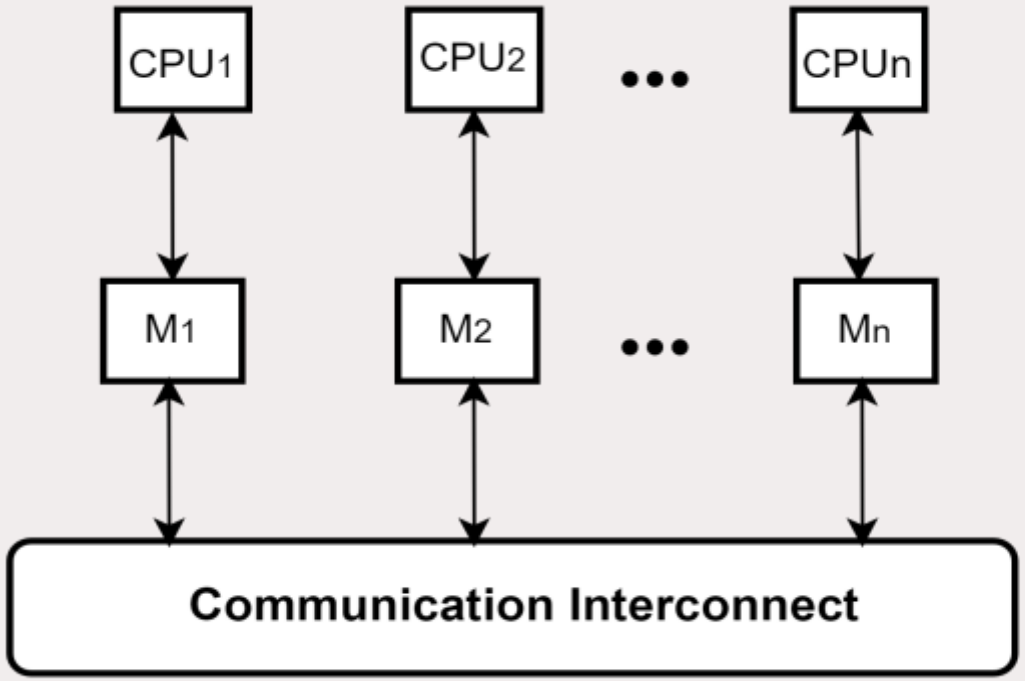
Distributed Memory:

- Each Processor has direct access only to its local memory
- Processors are connected via high-speed interconnect
- Data structures must be distributed

Distributed Memory Cont…



Architecture of Distributed Memory:



Flynn’s classification of computer architectures

CS621 Parallel and Distributed Computing

Objectives

What is Flynn’s classification of computer architectures?

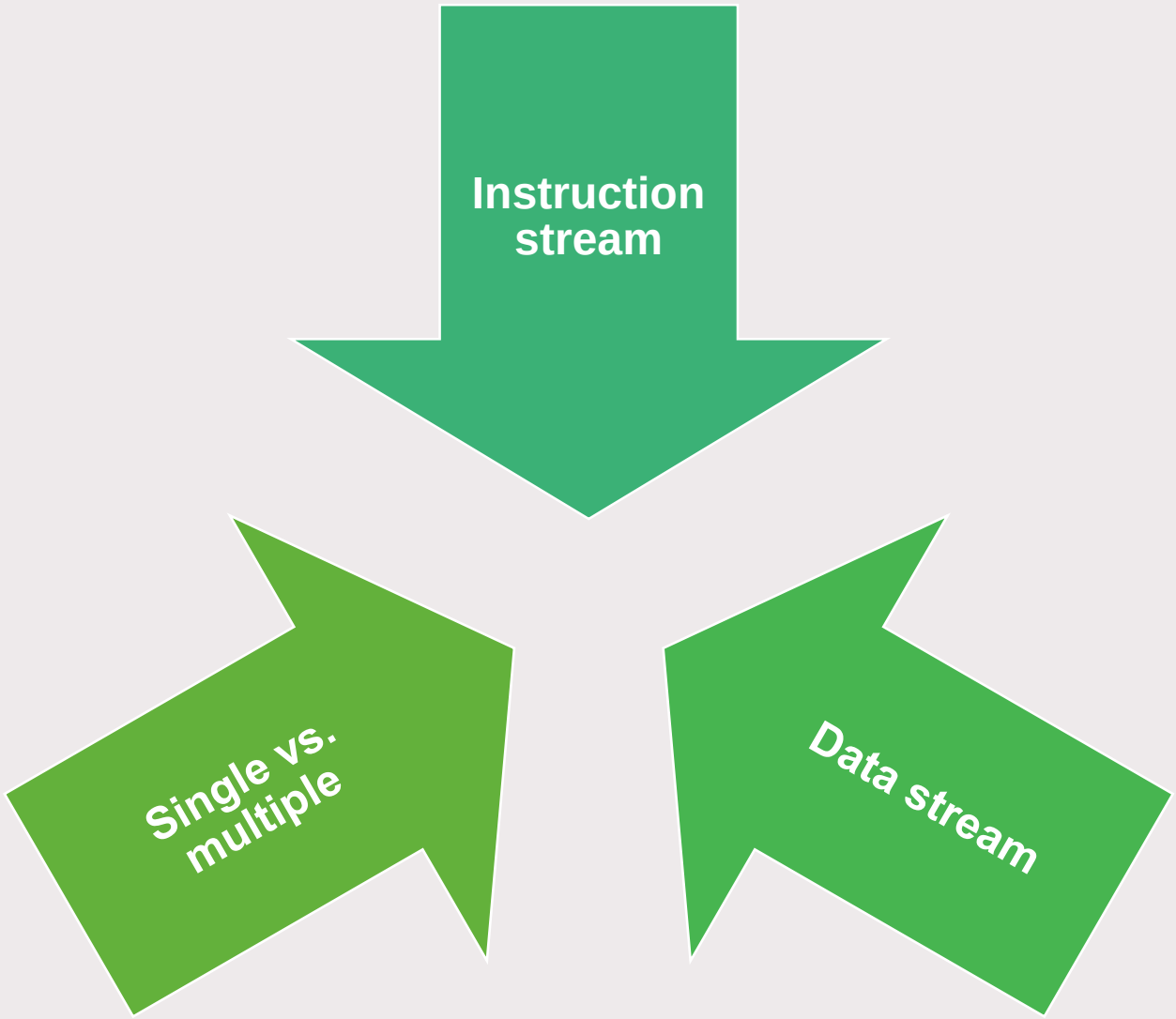
Basis for Flynn’s classification.

Types of Flynn’s classification.

Flynn’s classification of computer architectures (1966):

“Michael J Flynn classified computers on the basis of multiplicity of instruction stream and data streams in a computer system.”

Flynn’s classification of computer architectures Cont…



Flynn’s classification of computer architectures Cont…

The four classifications defined by Flynn are based upon the number of concurrent instruction (or control) and data streams available in the architecture.

- Single Instruction Single Data (SISD)
- Single Instruction Multiple Data (SIMD)
- Multiple Instruction Multiple Data (MIMD)
- Multiple Instruction Single Data (MISD)

Flynn’s classification of computer architectures Cont…

	Single instruction	**Multiple instruction**
Single data	SISD	MISD
Multiple data	SIMD	MIMD

SISD (Single-Instruction Single-Data)

CS621 Parallel and Distributed Computing

Objectives

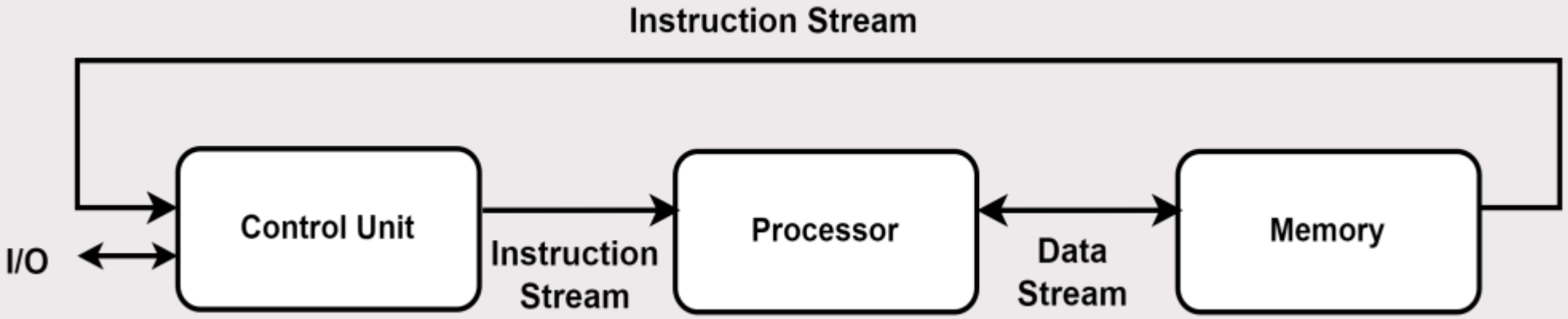
Introduction of SISD.

Architecture of SISD.

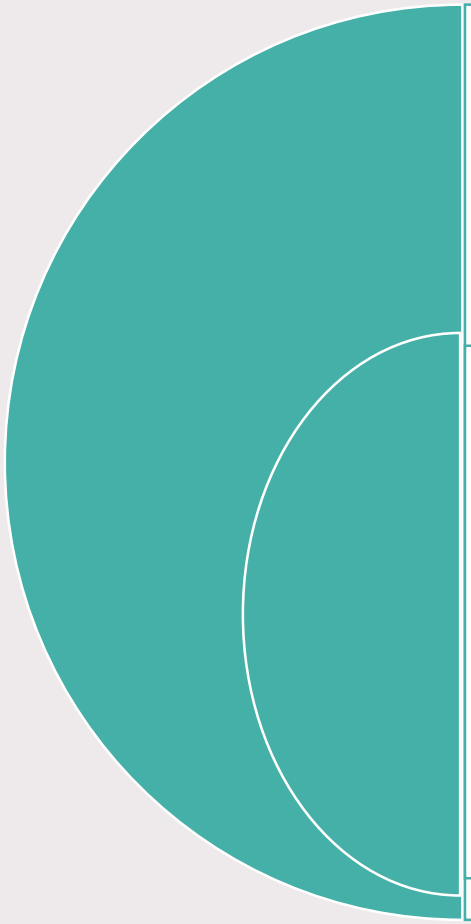
SISD (Single-Instruction Single-Data)

“Refers to the traditional von Neumann architecture where a single sequential processing element (PE) operates on a single stream of data.”

SISD (Single-Instruction Single-Data) Architecture



SISD (Single-Instruction Single-Data) Cont…



Conventional single-processor von Neumann computers are classified as SISD systems.

A typical non-pipelined architecture with general purpose registers as well as some dedicated special registers like:

- Program Counter (PC),
- Memory data registers (MDR),
- Memory address registers (MAR) and
- Instruction registers (IR).

SISD (Single-Instruction Single-Data) Cont…

Perform the same operation on multiple data operands concurrently.

Concurrency of processing rather than concurrency of execution.

Serial computer

Example: A personal computer processing instructions and data on single processor.

SIMD (Single-Instruction Multi-Data)

CS621 Parallel and Distributed Computing

Objectives

Introduction of SIMD.

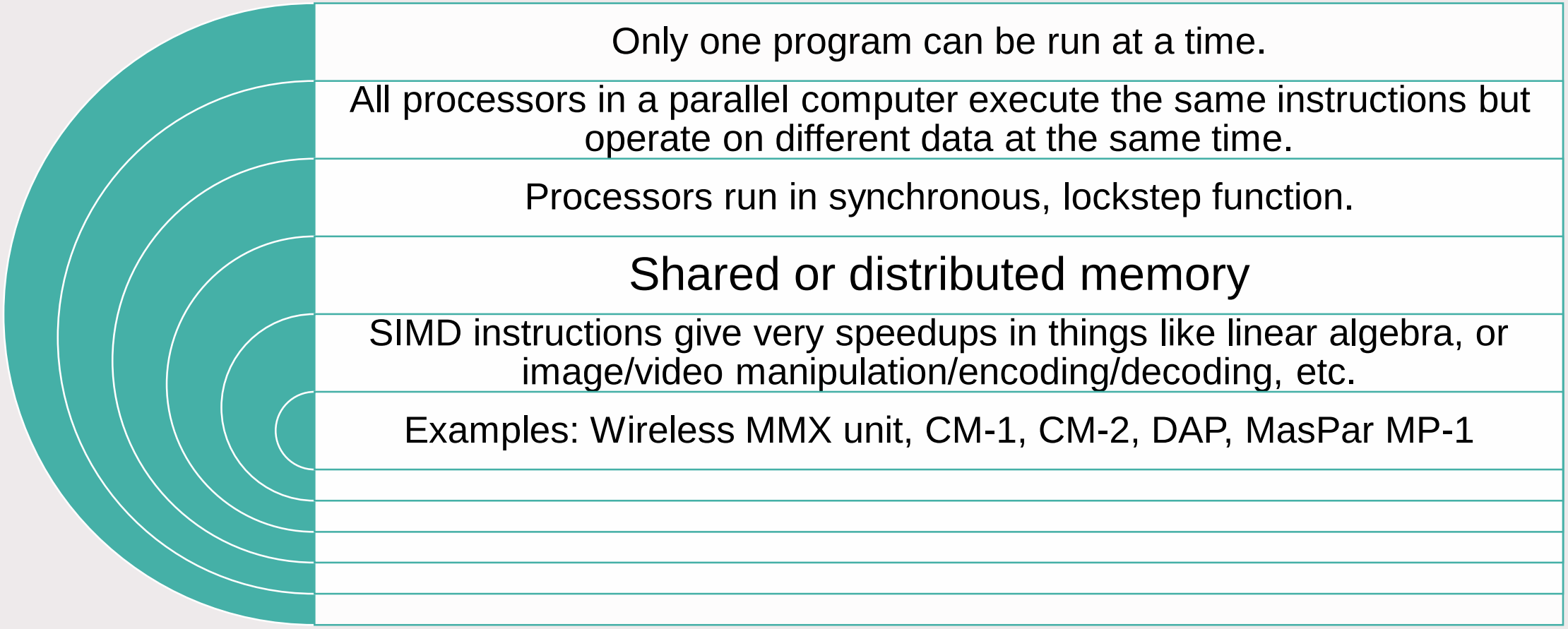
SIMD Architecture.

SIMD Schemes.

SIMD (Single-Instruction Multi-Data)

“SIMD is a multiple-processing system that performs one operation simultaneously on more than one piece of data.”

SIMD (Single-Instruction Multi-Data) Cont…

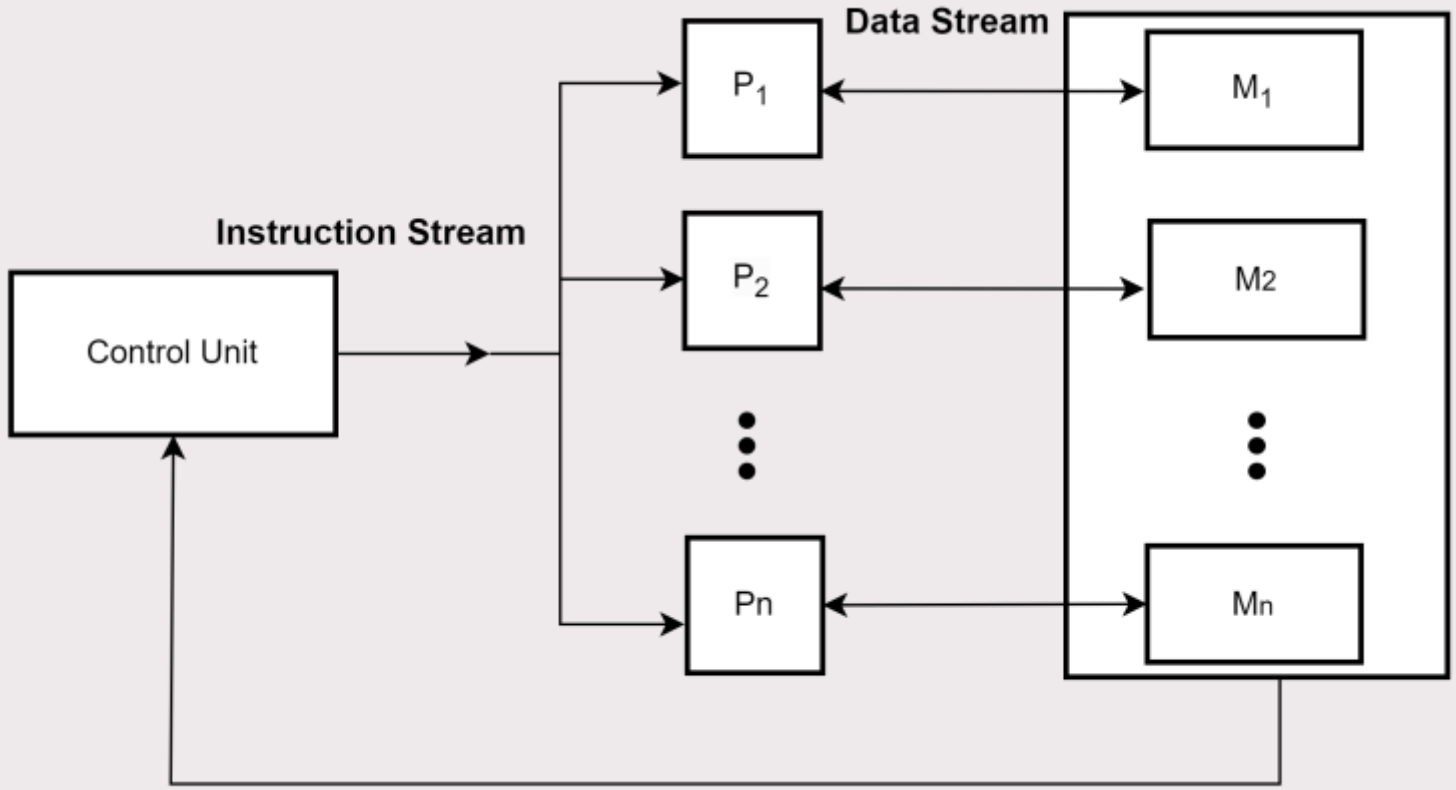


SIMD (Single-Instruction Multi-Data) Architecture

Consists of 2 parts:

- A front-end Von Neumann computer.
- A processor array: connected to the memory bus of the front end.

SIMD (Single-Instruction Multi-Data) Architecture Cont…



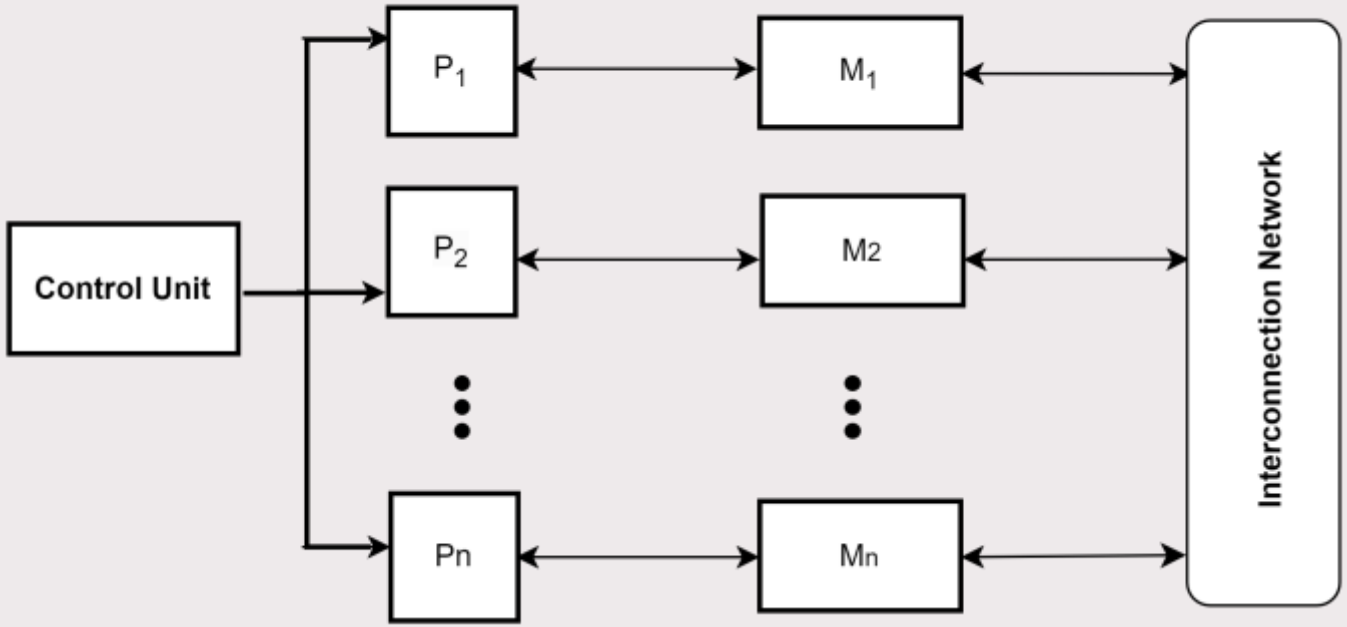
SIMD (Single-Instruction Multi-Data) Schemes

Classified into two configuration schemes

- Scheme 1 – Each processor has its own local memory.
- Scheme 2 – Processors and memory modules communicate with each other via interconnection network.

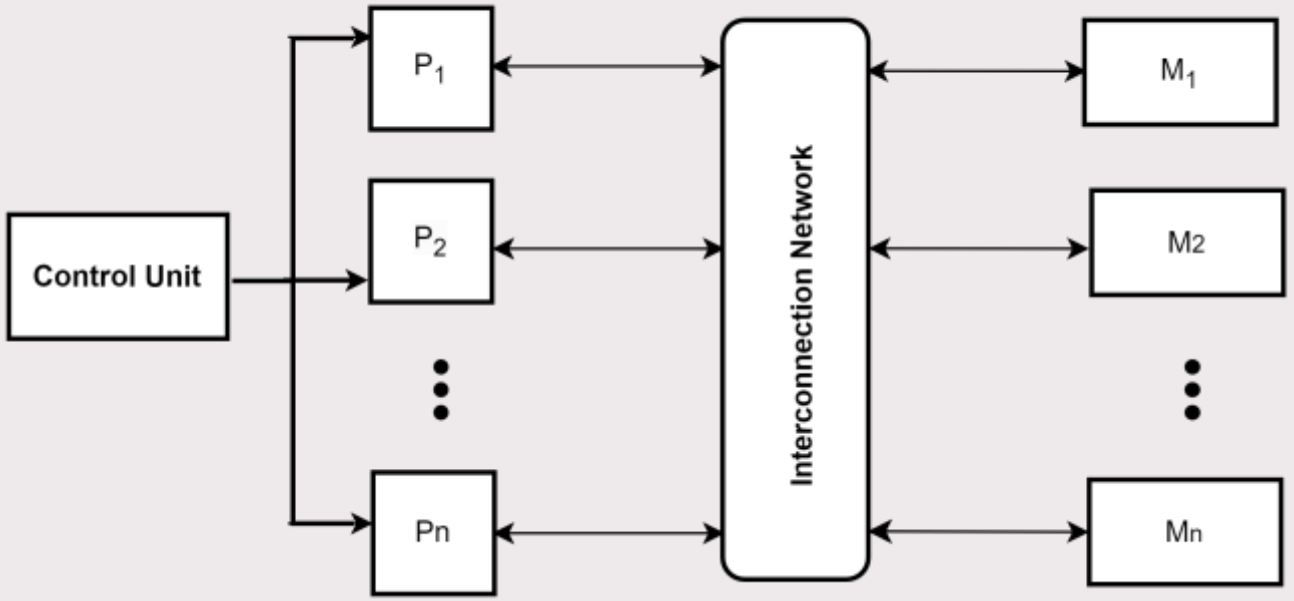
SIMD (Single-Instruction Multi-Data) Scheme Cont…

- SIMD Scheme 1



SIMD (Single-Instruction Multi-Data) Scheme Cont…

- SIMD Scheme 2



MISD (Multiple-Instruction Single-Data)

CS621 Parallel and Distributed Computing

Objectives

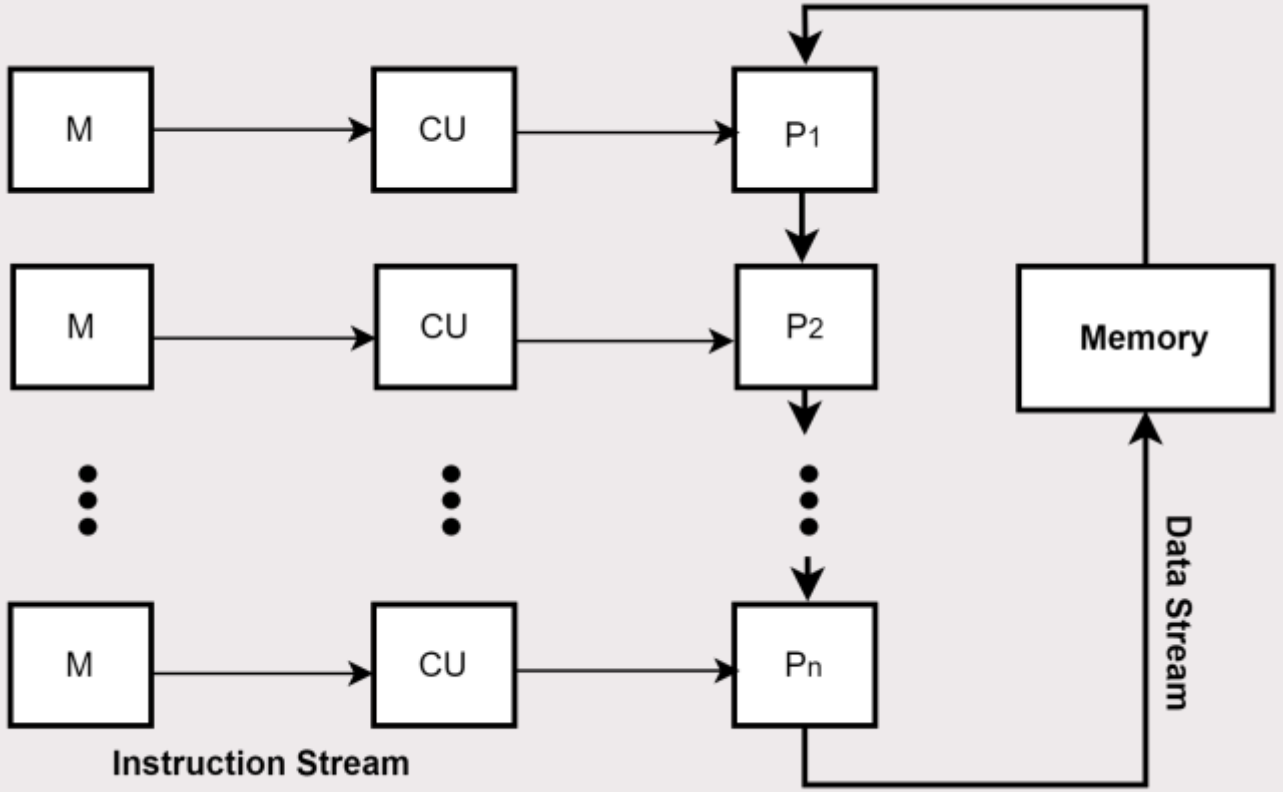
Introduction of MISD.

MISD Architecture.

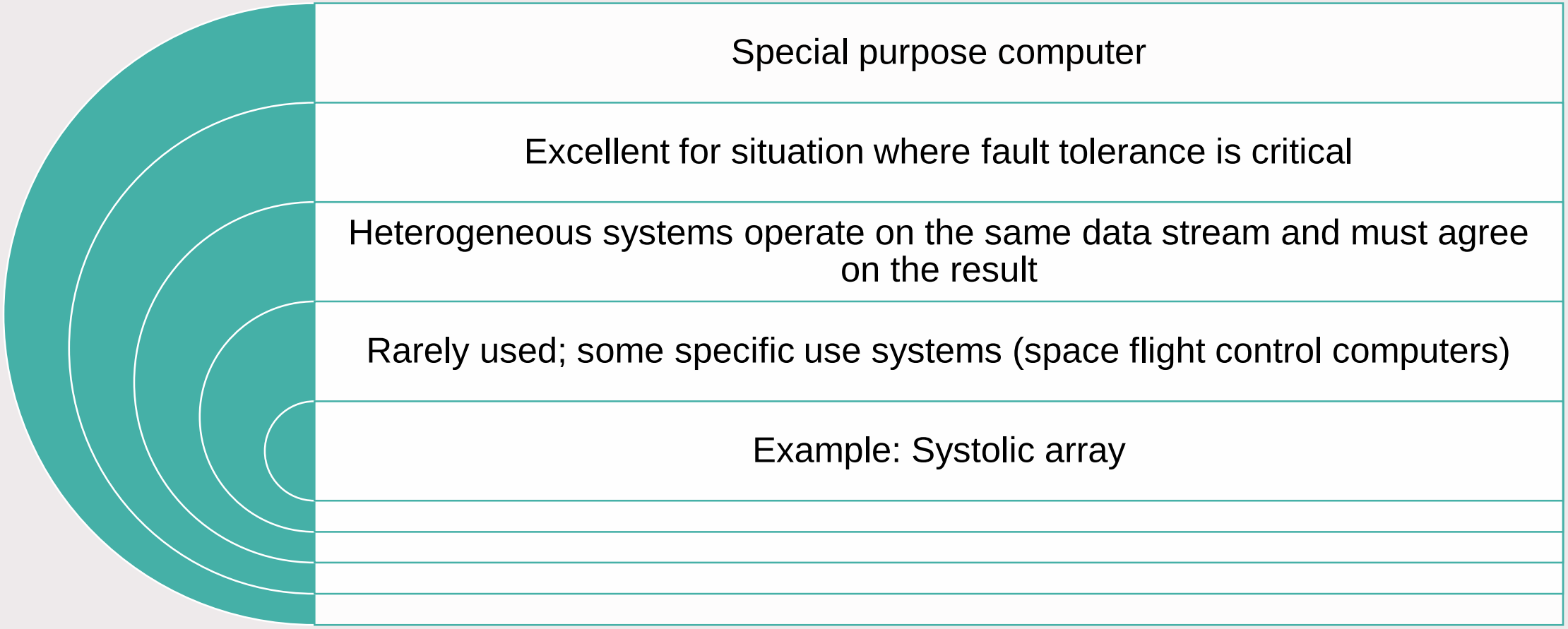
MISD (Multiple-Instruction Single-Data)

“A pipeline of multiple independently executing functional units operating on a single stream of data, forwarding results from one functional unit to the next."

MISD (Multiple-Instruction Single-Data) Architecture



MISD (Multiple-Instruction Single-Data) Cont…



MIMD (Multi-Instruction Multi-data)

CS621 Parallel and Distributed Computing

Objectives

Introduction of MIMD.

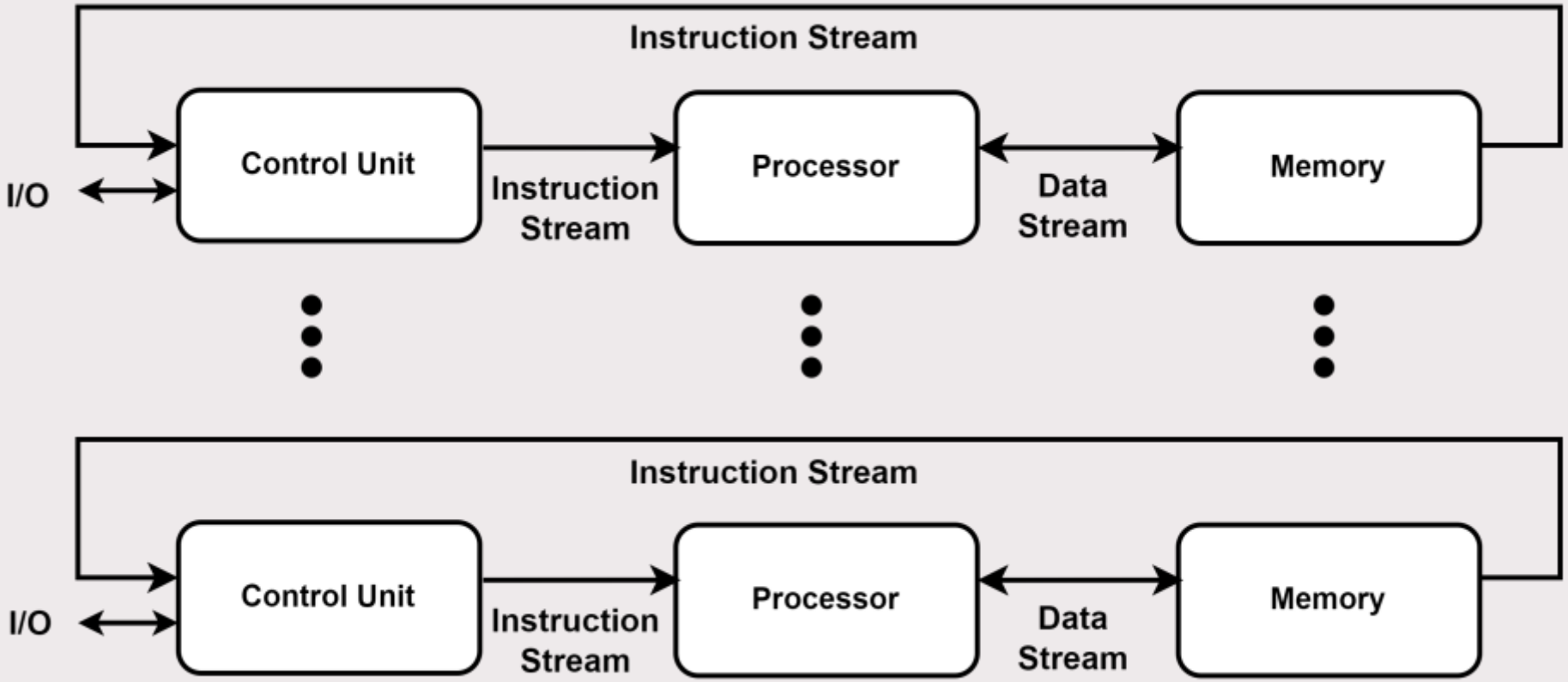
MIMD Architecture.

MIMD Shared Memory &
Message Passing Architectures.

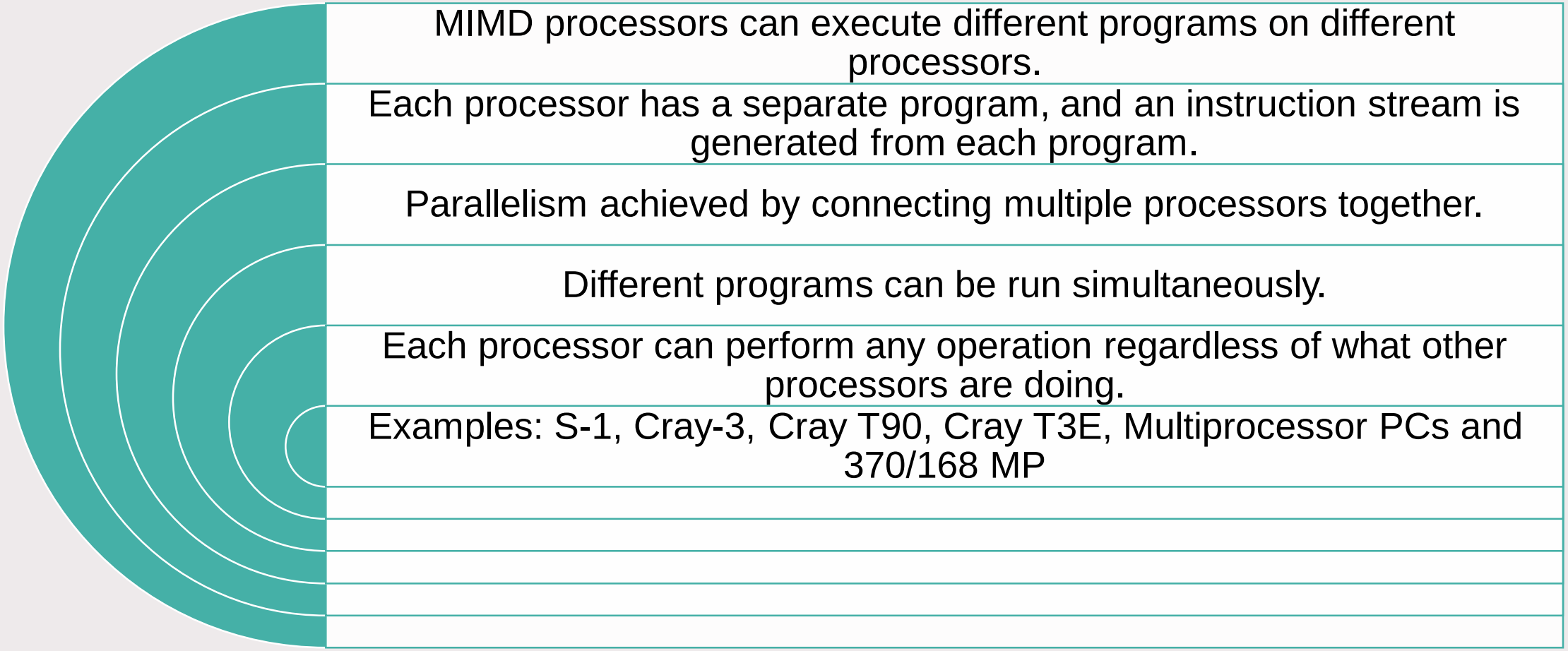
MIMD (Multi-Instruction Multi-data)

“In MIMD all processors in a parallel computer can execute different instructions and operate on various data at the same time.”

MIMD (Multi-Instruction Multi-data) Architecture



MIMD (Multi-Instruction Multi-data) Cont…



MIMD processors can execute different programs on different processors.
Each processor has a separate program, and an instruction stream is generated from each program.
Parallelism achieved by connecting multiple processors together.
Different programs can be run simultaneously.
Each processor can perform any operation regardless of what other processors are doing.
Examples: S-1, Cray-3, Cray T90, Cray T3E, Multiprocessor PCs and 370/168 MP

MIMD (Multi-Instruction Multi-data) Architecture Cont…

Special purpose computer

Shared or distributed memory

Made of multiple processors and multiple memory modules connected via some interconnection network.

Classified into two configuration schemes:

- Shared memory
- Message passing

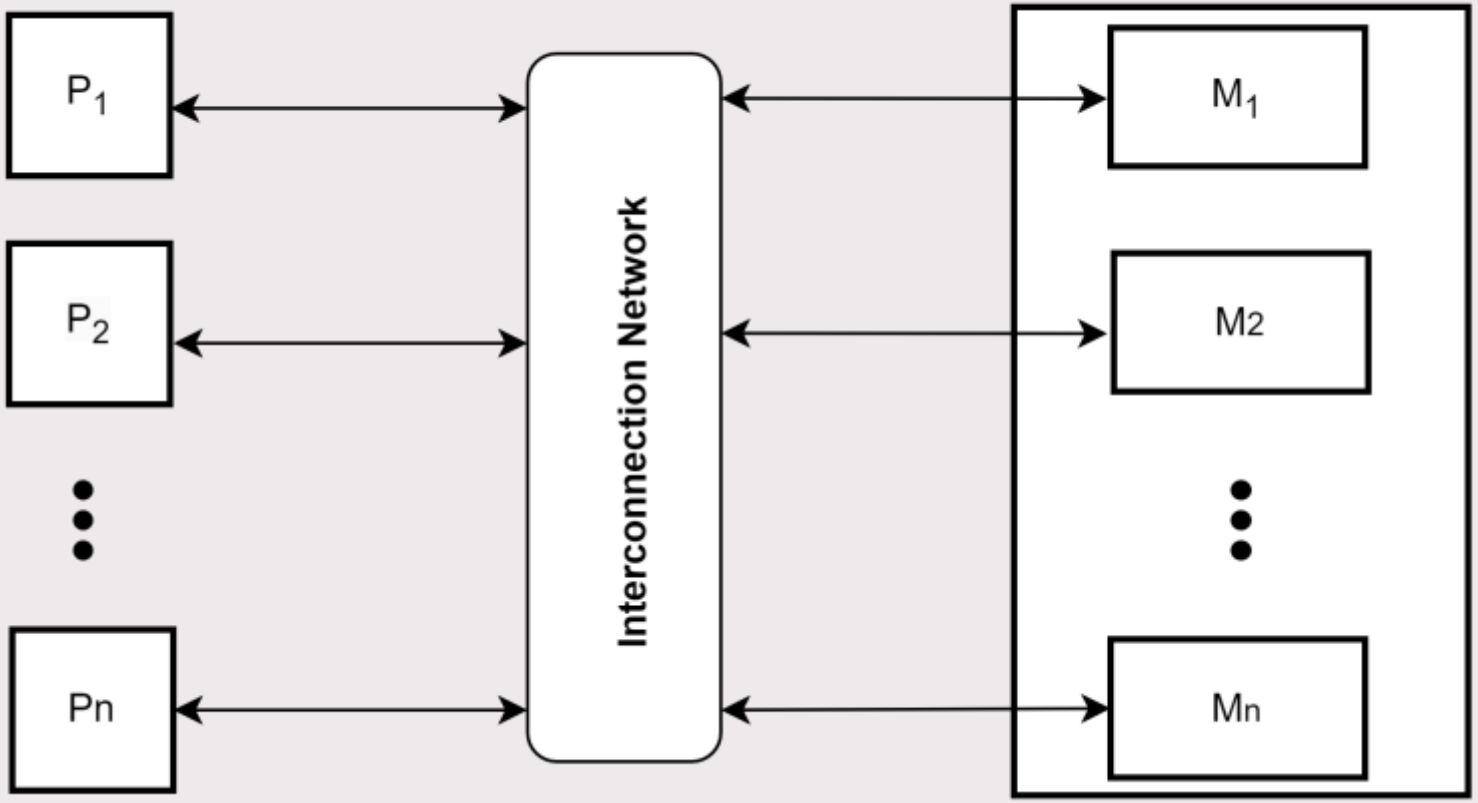
MIMD (Multi-Instruction Multi-data) Cont…

Shared Memory Organization

- Inter-processor coordination is accomplished by reading and writing in a global memory shared by all processors.
- Any processor can access then local memory of any other processor.
- Typically consists of servers that communicate through a bus and cache memory controller.

MIMD (Multi-Instruction Multi-data) Cont…

Shared Memory Architecture



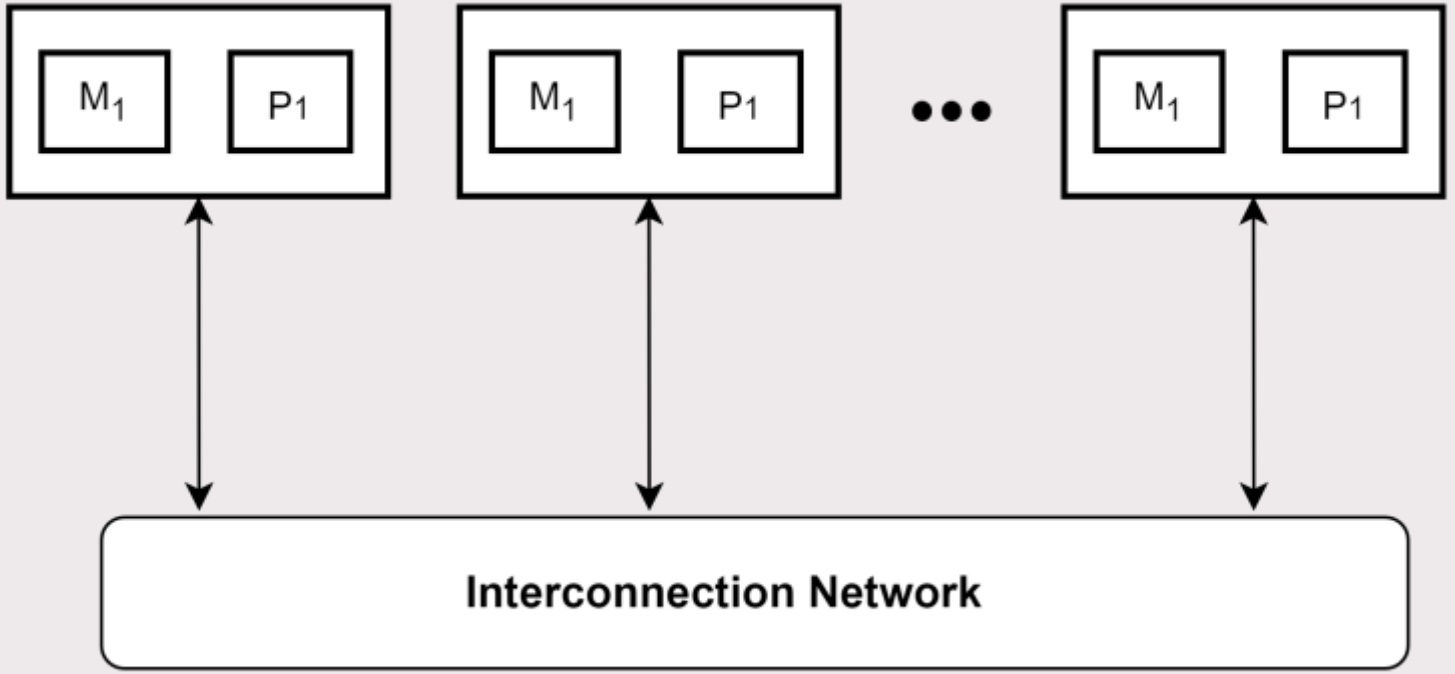
MIMD (Multi-Instruction Multi-data) Cont…

Message Passing Organization

- Point-to-Point
- Each processor has access to its own local memory.
- Communications are performed via send and receive operations.
- Message passing multiprocessors employ a variety of static networks in local communications.
- Must be synchronized among different processors.

MIMD (Multi-Instruction Multi-data) Cont…

Message Passing Architecture



SIMD-MIMD Comparison

CS621 Parallel and Distributed Computing

Objectives

Comparison of SIMD-MIMD.

SIMD-MIMD Comparison

SIMD computers require less hardware than MIMD computers (single control unit).

SIMD is less expensive as compared to MIMD.

SIMD follows synchronous processing where as, MIMD incorporates an asynchronous processing.

SIMD-MIMD Comparison Cont…

In MIMD each processing elements stores its individual copy of the program which increases the memory requirements. Conversely, SIMD requires less memory as it stores only one copy of the program.

MIMD is more complex and efficient as compared to SIMD.

SIMD-MIMD Comparison Cont…

Architecture	Single Instruction Multiple Data (SIMD)	Multiple Instruction Multiple Data (MIMD)
Type of processing	Same instruction on multiple data sets	Different instructions on multiple data sets
Flexibility	Limited	High
Memory access	Shared	Separate
Best suited for	Applications with regular data parallelism	Applications with irregular data dependencies or complex computations
Performance	High efficiency for parallelizable tasks	Flexible, but may be less efficient for parallelizable tasks

Virtual University

Dr. Muhammad Anwaar Saeed

Dr. Said Nabi

Ms. Hina Ishaq

CS621 Parallel and Distributed Computing

Introduction to Distributed Computing

CS621 Parallel and Distributed Computing

Objectives

Detailed explanation of distributed computing.

Introduction to Distributed Computing

“A distributed system is a collection of independent computers that appears to its users as a single coherent system.”

Introduction to Distributed Computing Cont.…

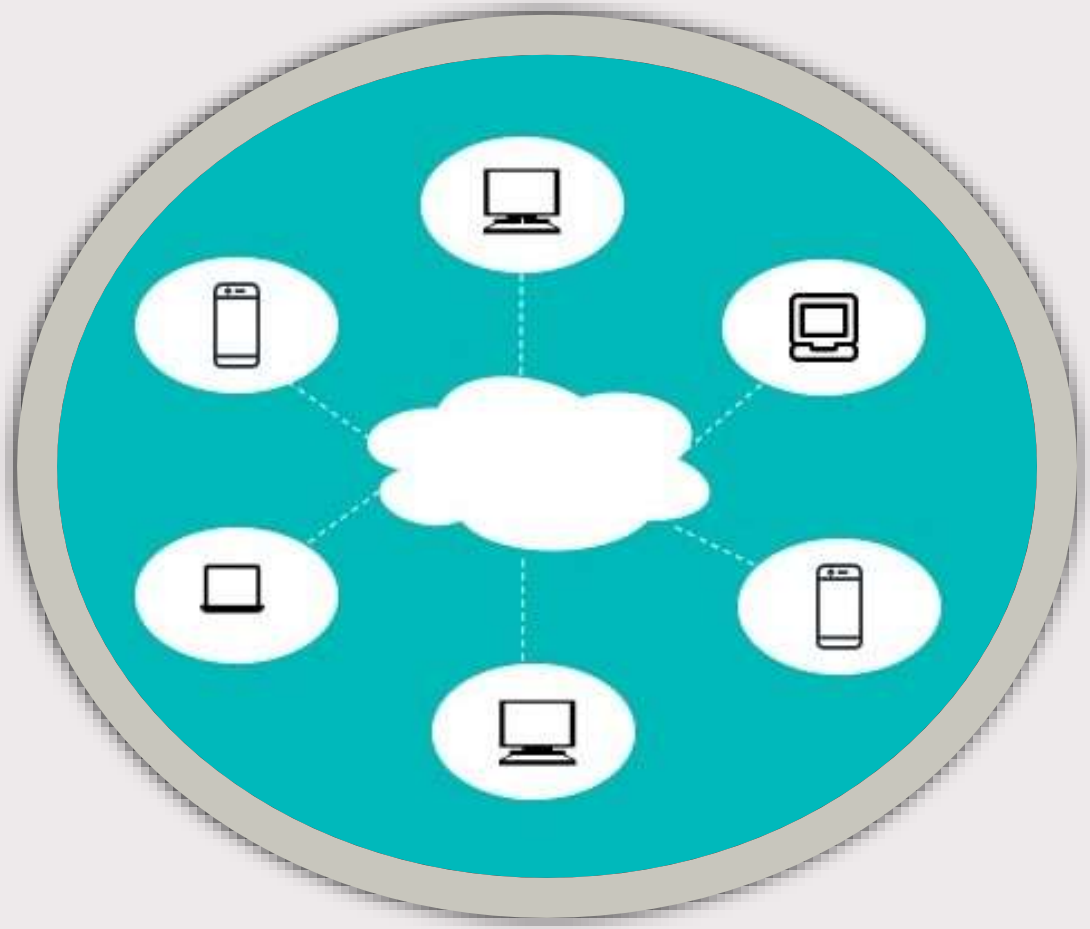
In distributed computing

- We have multiple autonomous computers which seems to the user as single system.
- There is no shared memory and computers communicate with each other through message passing.
- A single task is divided among different computers.

Uses or coordinates physically separate computing resources:

- Grid computing
- Cloud Computing

Introduction to Distributed Computing Cont.…



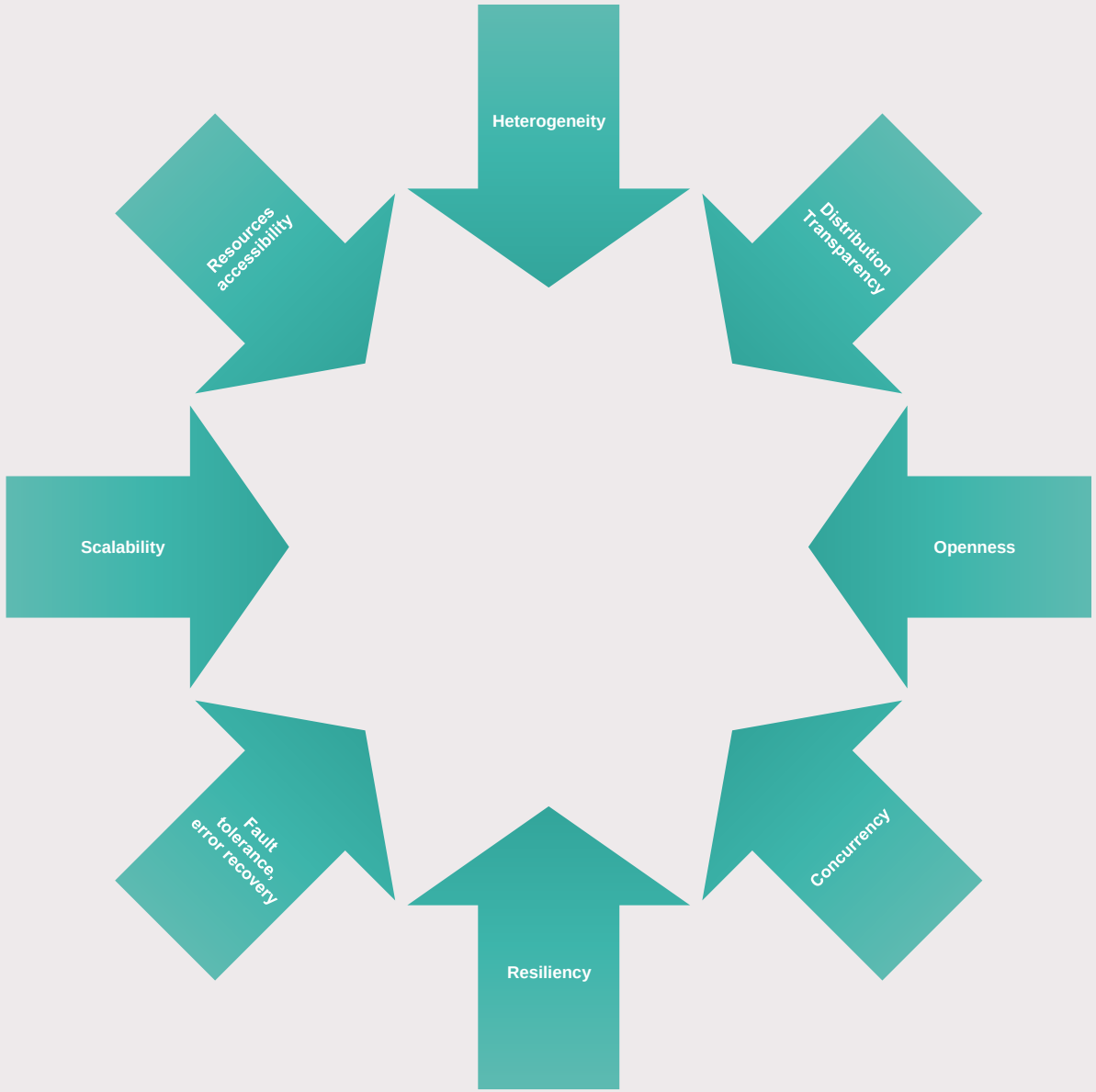
Why Use Distributed Computing?

CS621 Parallel and Distributed Computing

Objectives

Identifying the aspects that makes distributed computing more useful.

Why Use Distributed Computing



Why use Distributed Computing

Heterogeneity

The Internet enables users to access services and run applications over a heterogeneous collection of computers and networks.

Distribution Transparency

A distributed system that is able to present itself to users and applications as if it were only a single computer system is said to be transparent.

Openness

An open distributed system is a system that offers services according to standard rules that describe the syntax and semantics of those services.

Resiliency

With multiple computers, redundancies are implemented to ensure that a single failure doesn't equate to systems-wide failure.

Why use Distributed Computing Cont.…

Scalability

Scalability of a distributed system can be measured along at least three different dimensions: a system can be scalable with respect to its size, a geographically scalable system, a system can be administratively scalable.

Resources accessibility

The main goal of a distributed system is to make it easy for the users to access remote resources, and to share them in a controlled and efficient way.

Concurrency

Concurrency between programs sharing data is generally kept under control through synchronization mechanisms for mutual exclusion and transaction.

Fault tolerance, Error recovery

Fault tolerance is an important aspect in distributed systems design. A system is fault tolerant if it can continue to operate in the presence of failures.

Difference between Parallel and Distributed Computing

CS621 Parallel and Distributed Computing

Objectives

Identifying the key differences between parallel and distributed computing.

Difference between Parallel and Distributed Computing

Distributed computing is often used in tandem with parallel computing. Parallel computing on a single computer uses multiple processors to process tasks in parallel, whereas distributed parallel computing uses multiple computing devices to process those tasks.

Difference between Parallel and Distributed Computing Cont…

Sr.	Parallel Computing	Distributed Computing
1.	Many operations are performed simultaneously	System components are located at different locations
2.	Single computer is required	Uses multiple computers
3.	Multiple processors perform multiple operations	Multiple computers perform multiple operations
4.	Processors communicate with each other through bus	Computer communicate with each other through message passing.
5.	Improves the system performance	Improves system scalability, fault tolerance and resource sharing capabilities

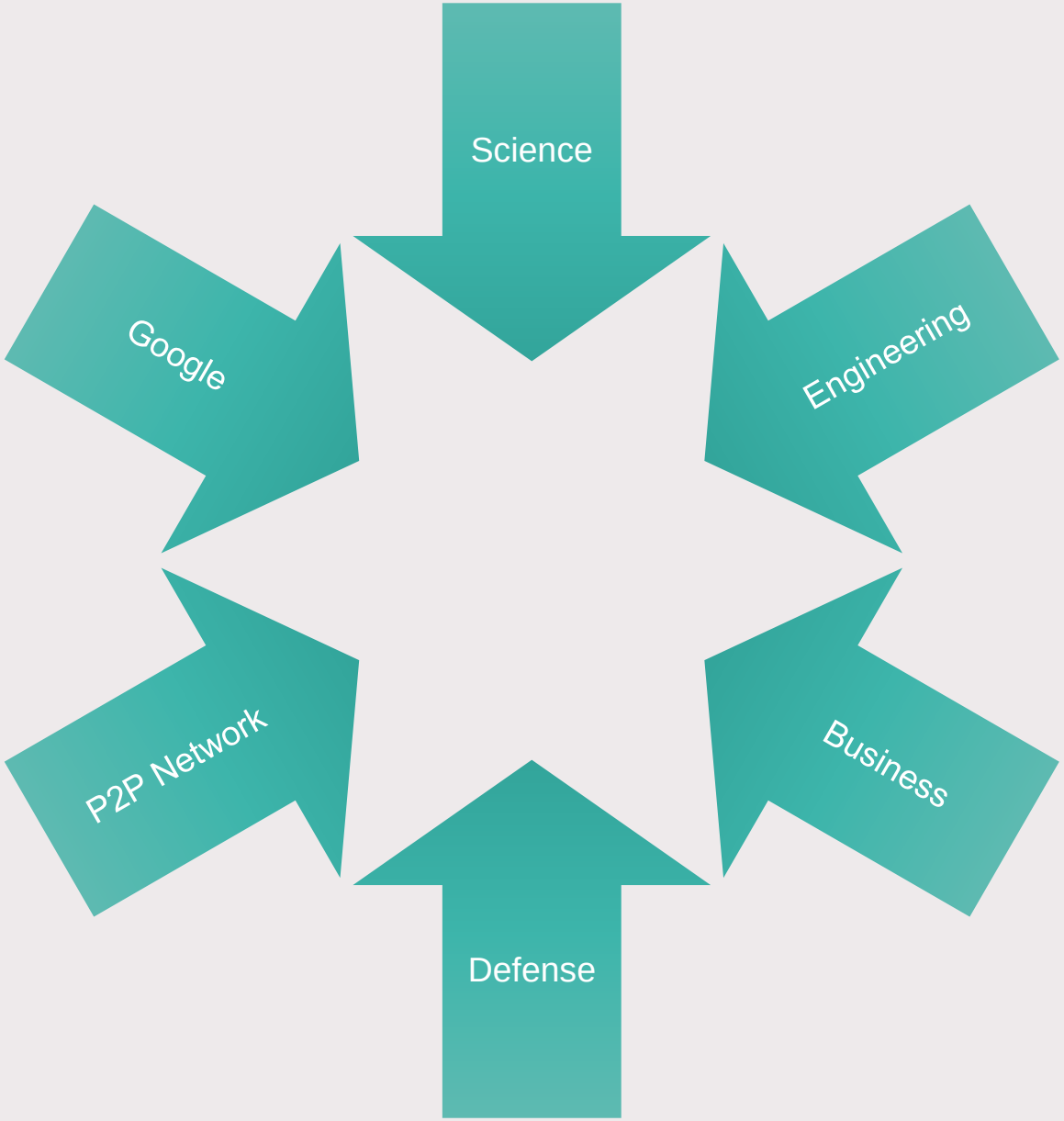
Applications of Parallel and Distributed Computing

CS621 Parallel and Distributed Computing

Objectives

Scope of parallel and distributed computing.

Applications of Parallel and Distributed Computing



Applications of Parallel and Distributed Computing

Science

- Global climate modeling
- Biology: genomics; protein folding; drug design
- Astrophysical modeling
- Computational Chemistry
- Computational Material Sciences and Nano sciences

Engineering

- Semiconductor design
- Earthquake and structural modeling
- Computation fluid dynamics (airplane design)
- Combustion (engine design)
- Crash simulation

Applications of Parallel and Distributed Computing Cont....

Business

- Financial and economic modeling
- Transaction processing, web services and search engines

Defense

- Nuclear weapons -- test by simulations
- Cryptography

Applications of Parallel and Distributed Computing Cont.…

P2P Network

- eDonkey, BitTorrent, Skype, …

Google

- 1500+ Linux machines behind Google search engine

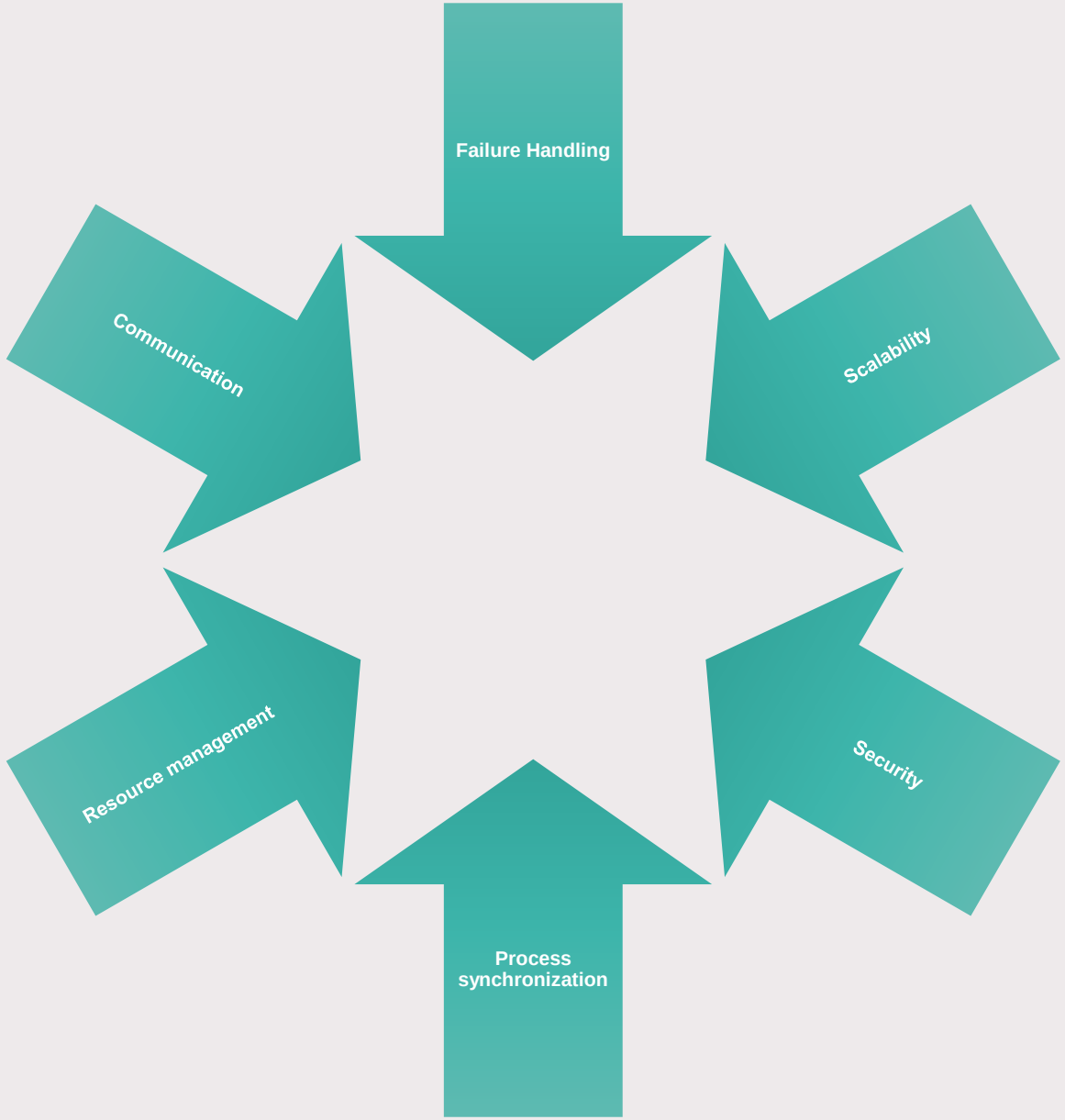
Issues in Parallel and Distributed Computing

CS621 Parallel and Distributed Computing

Objectives

Identifying the key challenges in parallel and distributed computing.

Issues in Parallel and Distributed Computing



Issues in Parallel and Distributed Computing

Failure Handling

Failures, like in any program, are a major problem. With so many processes and users, the consequences of failures are exacerbated.

Scalability

As a distributed system is scaled, several factors need to be taken into account: size, geography, and administration; and their associated problems like overloading, control and reliability.

Security

As connectivity and sharing increase, security is becoming increasingly important. Increased connectivity can also lead to unwanted communication, such as electronic junk mail, often called spam.

Issues in Parallel and Distributed Computing Cont....

Process synchronization

One of the most important issues that engineers of distributed systems are facing is synchronizing computations consisting of thousands of components. Current methods of synchronization like semaphores, monitors, barriers, remote procedure call, object method invocation, and message passing, do not scale well.

Resource management

In distributed systems, objects consisting of resources are located on different places. Routing is an issue at the network layer of the distributed system and at the application layer. Resource management in a distributed system will interact with its heterogeneous nature.

Communication and Latency

Distributed Systems have become more effective with the advent of Internet but there are certain requirements for performance, reliability etc. Effective approaches to communication should be used.

Parallel and Distributed Computing Efforts

CS621 Parallel and Distributed Computing

Objectives

Identifying the prerequisites for parallel and distributed computing.

Parallel and Distributed Computing Programming Design Efforts

Before a program is written or a piece of software is developed, it must first go through a design process.

For parallel and distributed programs, the design process will include three issues:

- Decomposition
- Communication
- Synchronization

Decomposition

Decomposition is the process of dividing up the problem and the solution into parts: logical areas and logical resources.

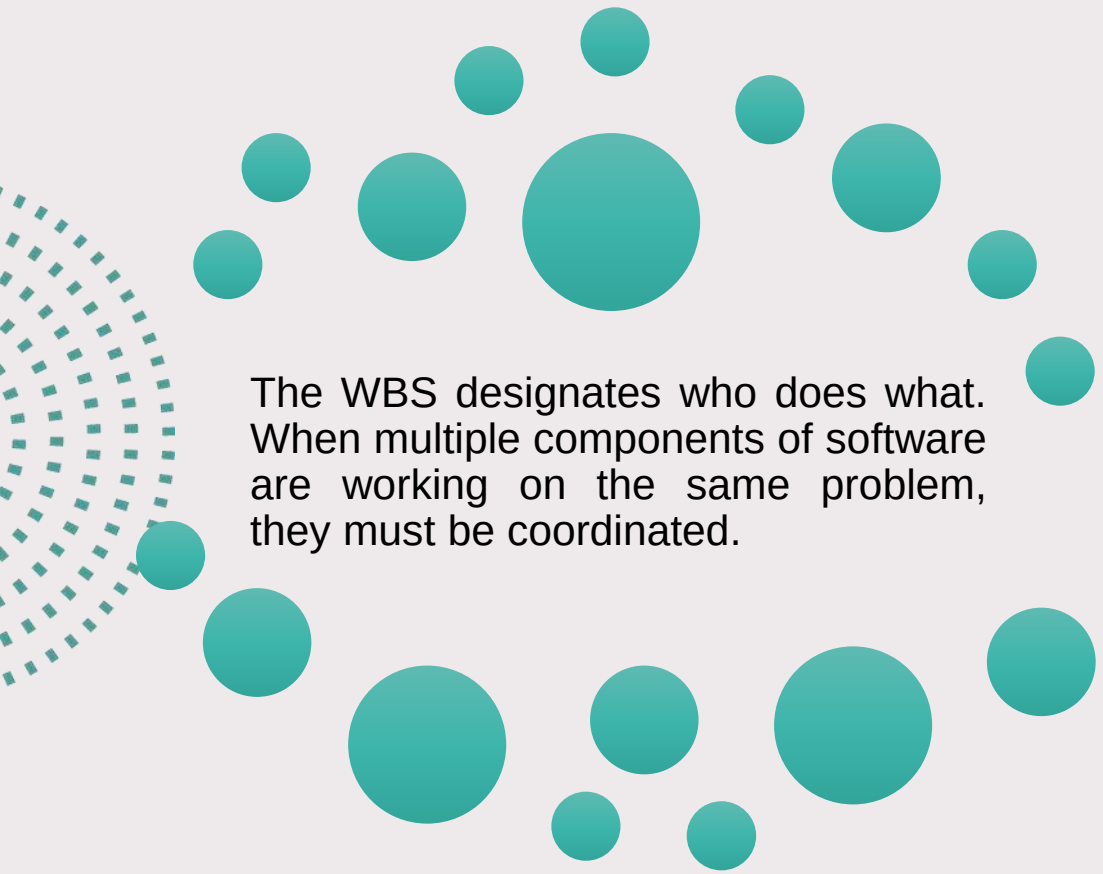
One of the primary issues of concurrent programming is identifying a natural WBS(Work breakdown structure) for the software solution at hand.

Communication

Following issues must be considered when designing parallel or distributed systems:
Important Questions:

- How will this communication be performed if the parts are in different processes or different computers?
- Do the different parts need to share any memory?
- How will one part of the software know when the other part is done?
- Which part starts first?
- How will one component know if another component has failed?

Synchronization



The WBS designates who does what. When multiple components of software are working on the same problem, they must be coordinated.

The components' order of execution must be coordinated. .
Do all the parts start at the same time or does some work while others wait?
What two or more components need access to the same resource?
Who gets it first? If some of the parts finish their work long before the other parts, should they be assigned new work?
Who assigns the new work in such cases?

Virtual University

Dr. Muhammad Anwaar Saeed

Dr. Said Nabi

Ms. Hina Ishaq

CS621 Parallel and Distributed Computing

What is Computing?

CS621 Parallel and Distributed Computing

Objectives

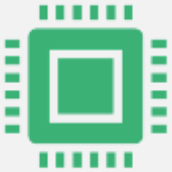
Introduction of Computing.

History of Computing.

What is Computing

“Computing is the process to complete a given goal-oriented task by using computer technology.”

What is Computing Cont.…



Computing may include the design and development of software and hardware systems for broad range of purposes.

Used for structuring, processing and managing any kind of information - to aid in the pursuit of scientific studies and making intelligent systems.

History of Computing

- Batch Era
- Time Sharing Era
- Desktop Era
- Network Era

History of Computing Cont..

Batch Era: Execution of series of programs on a computer without manual intervention.

Time Sharing: Sharing of computing resource among many users by means of multiprogramming and multi-tasking.

Desktop Era: A personal computer provides computing power to one user.

Network Era: Systems with shared memory and distributed memory.

Serial Vs. Parallel Computing

CS621 Parallel and Distributed Computing

Objectives

Serial Computing.

Parallel Computing.

Difference between serial and parallel computing

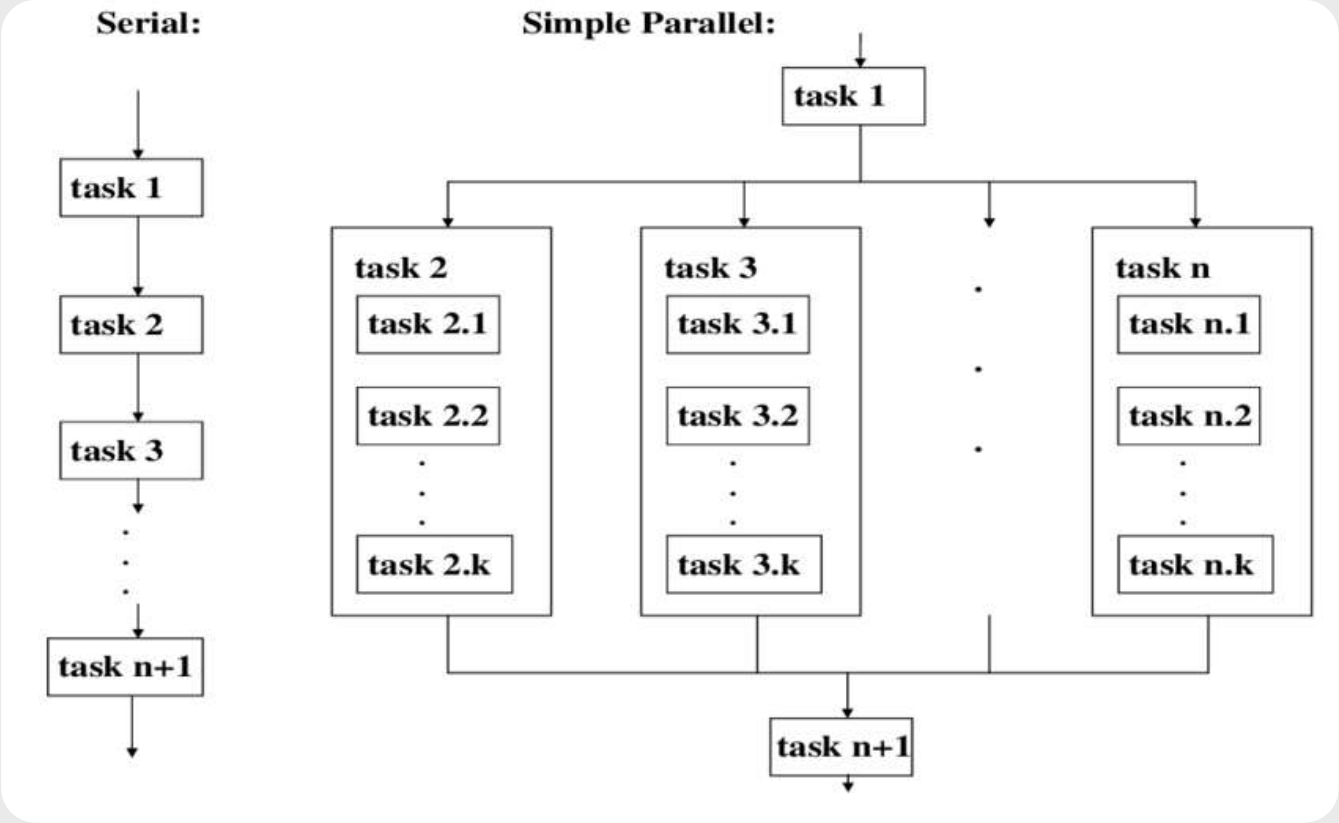
Serial Computing

“Serial computing is a type of processing in which one task is completed at a time and all the tasks are executed by the processor in a sequence.”

Parallel Computing

“Parallel computing is a type of computing architecture in which several processors simultaneously execute multiple, smaller calculations broken down from an overall larger, complex problem.”

Serial vs parallel Computing Cont..



Difference between Serial and parallel Computing

Serial Computing	Parallel Computing
Are uniprocessor systems.	Are multiprocessor systems.
Can execute one instruction at a time.	Can execute multiple instructions at a time.
Speed is limited.	No limitation on speed.
Lower performance.	Higher performance.
Examples: EDVAC, BINAC, and LGP-30.	Example: Window 7, 8 and 10.

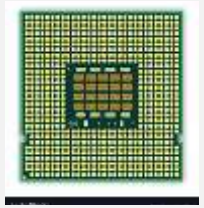
Introduction to Parallel Computing

CS621 Parallel and Distributed Computing

Objectives

Parallel Computing

Multi-Processer



Multi-Core

Introduction to Parallel Computing

“Parallel Computing is the simultaneous execution of the same task (split up and adapted) on multiple processors in order to obtain faster results.”

Introduction to Parallel Computing Cont.…

It is a kind of computing architecture where the large problems break into independent, smaller, usually similar parts that can be processed in one go. It is done by multiple CPUs communicating via shared memory, which combines results upon completion. It helps in performing large computations as it divides the large problem between more than one processor.

HPC: High Performance/Productivity Computing

Technical Computing

Cluster computing

Introduction to Parallel Computing Cont.…

The term parallel computing architecture sometimes used for a computer with more than one processor available for processing.

The recent multicore processor (chips with more than one processor core) are some commercial examples which bring parallel computing to the desktop.

Introduction to Parallel Computing Cont.…

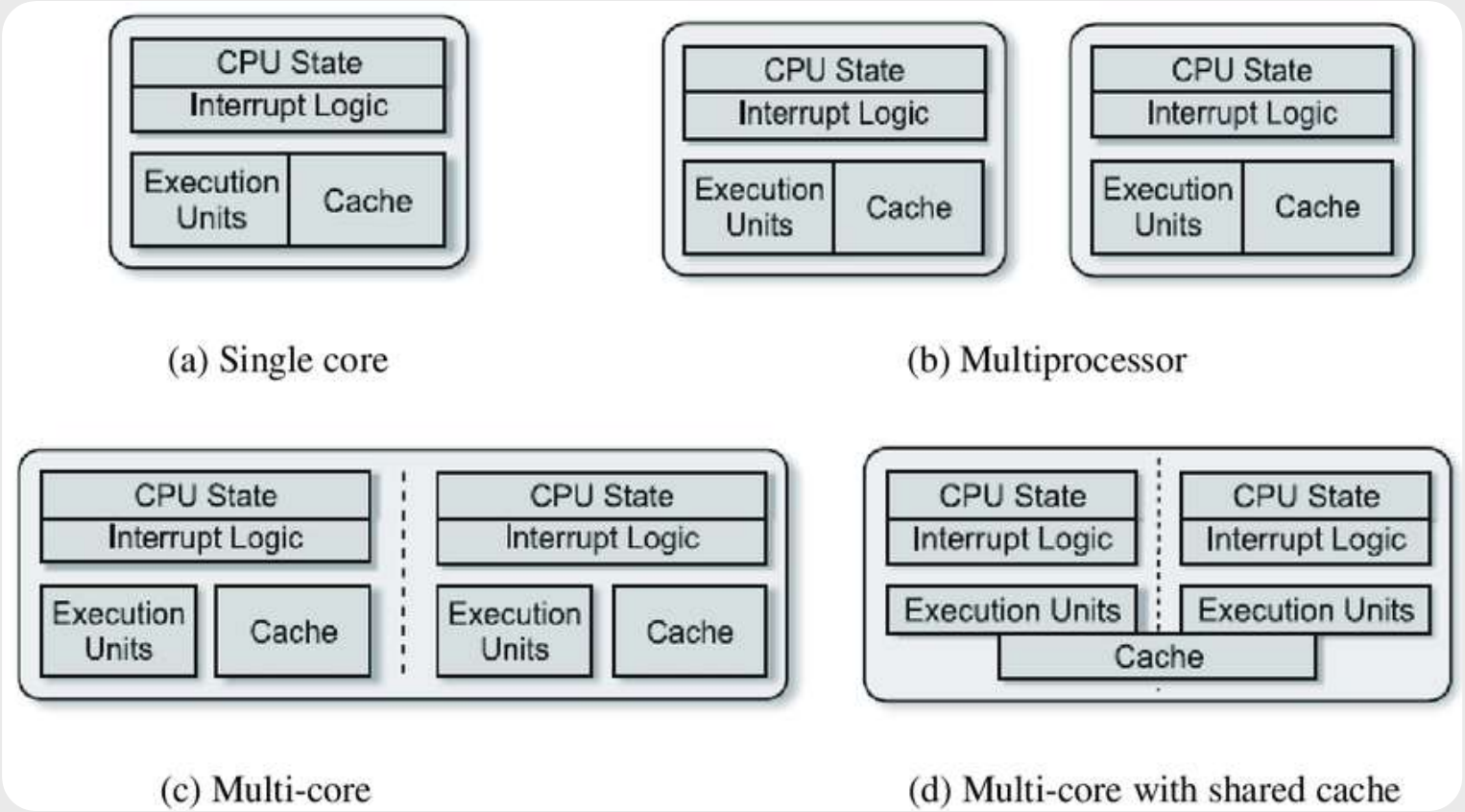
Multi-processor

More than one CPU works together to carry out computer instructions or programs.

Multi-core

Is a microprocessor on a single integrated circuit with two or more separate processing units, called cores, each of which reads and executes program instructions.

Introduction to Parallel Computing Cont…



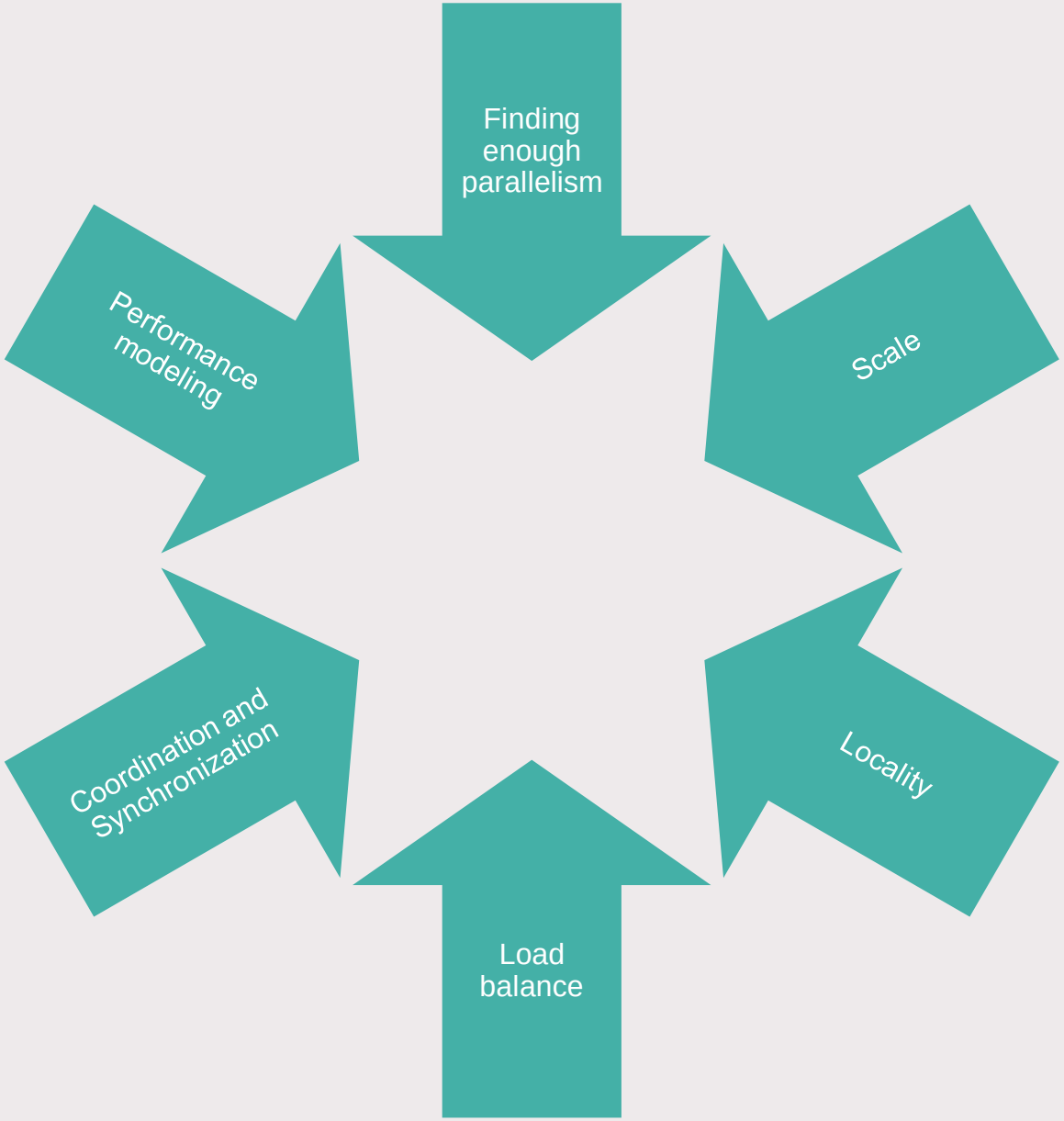
Principles of Parallel Computing

CS621 Parallel and Distributed Computing

Objectives

Principles of Parallel Computing.

Principles of Parallel Computing



Principles of Parallel Computing Cont…

Finding Enough Parallelism

Conventional architectures coarsely comprise of a processor, memory system, and the data-path. Each of these components' present significant performance bottlenecks. Parallelism addresses each of these components in significant ways.

Scale

Parallelism overhead includes cost of starting a head, accessing data, communicating shared data, synchronization and extra computation. Algorithms needs sufficiently large units of work to run fast in parallel.

Locality

Parallel processors collectively have large and fast cache. The memory addresses are distributed across the processors, a processor may have faster access to memory locations mapped locally than to memory locations mapped to other processors.

Principles of Parallel Computing Cont.…

Load Balance

Determines the workload, divide up evenly before staring in case of static load balancing but in dynamic load balance workload changes dynamically, need to rebalance dynamically.

Coordination and Synchronization

Several kind of synchronization is needed by processes cooperating to perform computation.

Performance Modeling

More efficient programming models and tools formulated for massively parallel supercomputers.

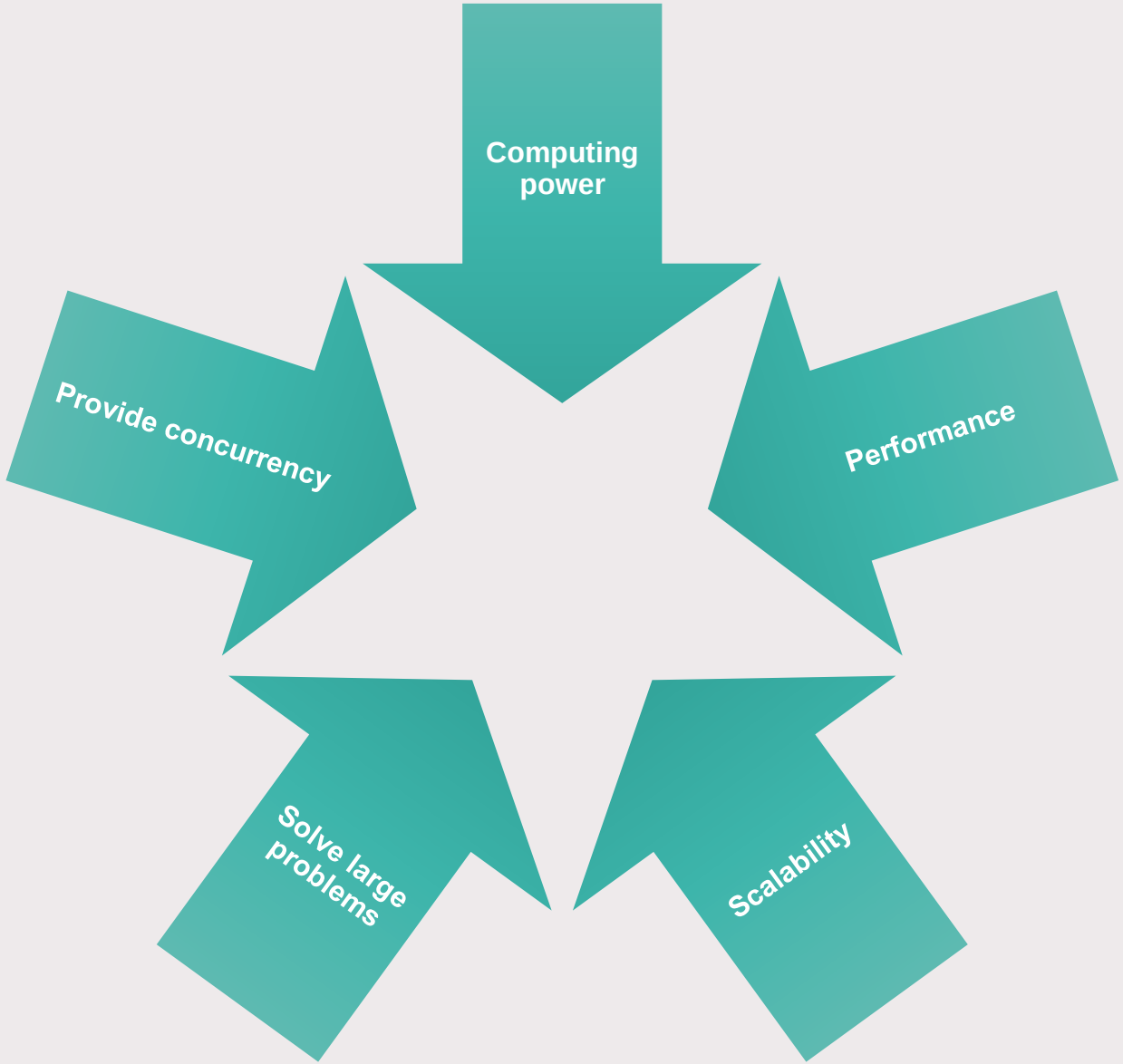
Why Use Parallel Computing?

CS621 Parallel and Distributed Computing

Objectives

Identifying the aspects that make parallel computing more useful.

Why Use Parallel Computing?



Why Use Parallel Computing?

Computing power

Modern consumer grade computing hardware comes equipped with multiple central processing units (CPUs) and/or graphics processing units (GPUs) that can process many sets of instructions simultaneously.

Performance

Theoretical performance steadily increased, due to the fact that performance is proportional to the product of the clock frequency and the number of cores.

Scalability

Problem can be scaled up from size to sizes that were out of reach with a serial application. The larger problem sizes are enabled by the larger amounts of main memory, disk storage, bandwidth over networks and to disk, and CPUs.

Why Use Parallel Computing Cont.…

Solve large problems

Solve large problems by breaking down larger problems into smaller, independent, often similar parts that can be executed simultaneously by multiple processors communicating via shared memory. Can solve large problems like Web search engines, processing millions of transaction per second, etc.

Cost

the cost of computation would reduce if we are to deploy parallel computation than sequential computation.

Provide concurrency

Parallelism leads naturally to Concurrency. For example, Several processes trying to print a file on a single printer.