# CSE5351: Parallel Processing

# Part 1A

# Assignment 1 (Deadline Sept. 16)

- Your name
- Title of the project (supercomputer #)
- 3-page description in your own words (no cut and paste) single line, font size 11
- References of papers
- Names (and emails) of researchers
- Attach PDf files, not links
  1. IEEE
  2. ACM
  3. Springer
  4. Elsevier

**Google search Top 500 supercomputers.**

https://www.top500.org/

# Don't Do's

- Write No more than a small paragraph about the hardware of the supercomputer
- Not Too much history
- No Link
- Email the Word file along with two pdf files.

- Reference
- I. Ahmad and D. Andrew, "The fastest algorithm for sorting," *Journal of Supercomputing, Jan. 2020, PP. 31-45, Springer.*

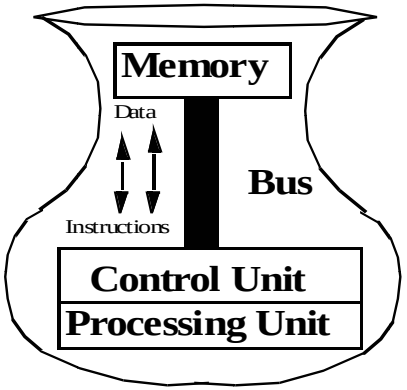
# Evolution of Computer Architecture

- Computers have gone through 70 years of development.
- There have been five generations of development,
- These generation differ in technology, architecture, software, application, and representative systems
- Each generation has improved from the previous generations in hardware and software technologies used and in application levels.

# Why Parallel Computers?

## Fast is never fast enough!

One processor, no matter how fast, is not enough because of the fundamental bottleneck flaw in the Von Neumann architecture.




## Are modern computers fast enough to solve all of the complex problems?

No, because the complexity of problems is increasing at rate faster than the speed of computers.

## History of computing indicates that demand for computing power is always greater than available hardware

# Computationally Intensive Applications

- There is a very large number of applications which are computationally intensive and are crucial to the advancement of technology.
- These complex problems belong to science and engineering and require large memory and a Petaflop (PFLOPS stands for 1000 trillions of floating point operations) computer.

# Grand Challenge Problems

**Prediction of weather, climate, and global changes**

**Challenges in materials sciences**

**Design of drugs**

**Human genome**

**Astronomy**

**Challenges in transportation**

**Vehicle dynamics**

**Nuclear fusion**

**Enhanced oil and gas recovery**

**Computational ocean sciences**

**Vision**

**Visualization and graphics**

# What do we mean by parallel?

*Parallelism* can be achieved through the use of a *parallel computer*.

*Parallel Processing* is information processing that emphasizes the concurrent manipulation of data elements belonging to two or more processes solving a single problem.

A *parallel computer* is a multiple-processors computer capable of parallel processing. It is a computer with many processing units, or processors interconnected by a network.

Given a problem to be solved, it is broken into a number of sub-problems. All of these sub-problems are now solved simultaneously, each on a different processor. The results are then combined to produce an answer to the original problem.

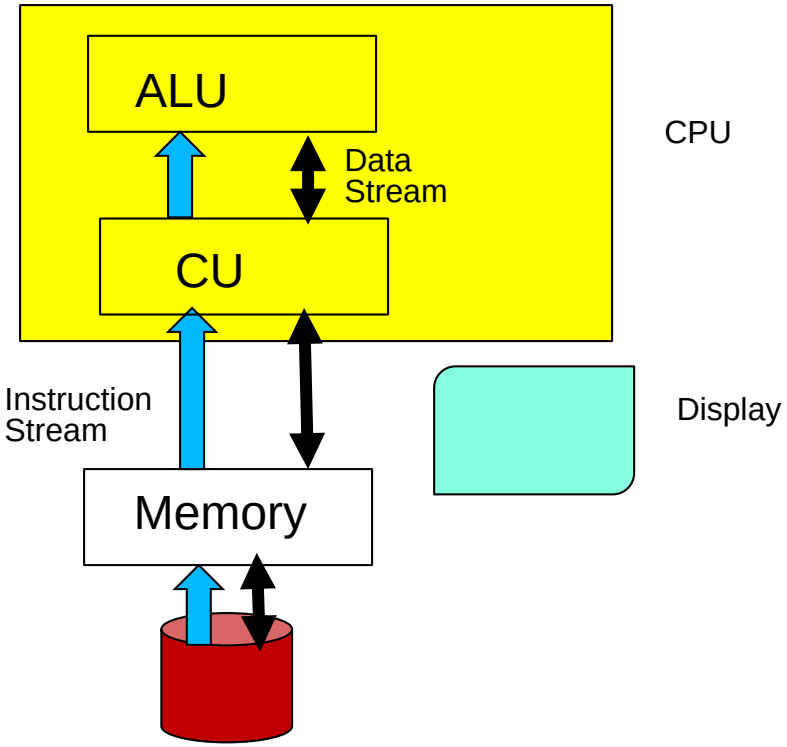
# Supercomputers

- A *supercomputer* is a general-purpose computer capable of solving individual problems at extremely high computational speeds, compared with other computers built during the same time. By definition supercomputers have always existed.
- All contemporary supercomputers are parallel computers with the number of processors varying from 2 to millions.

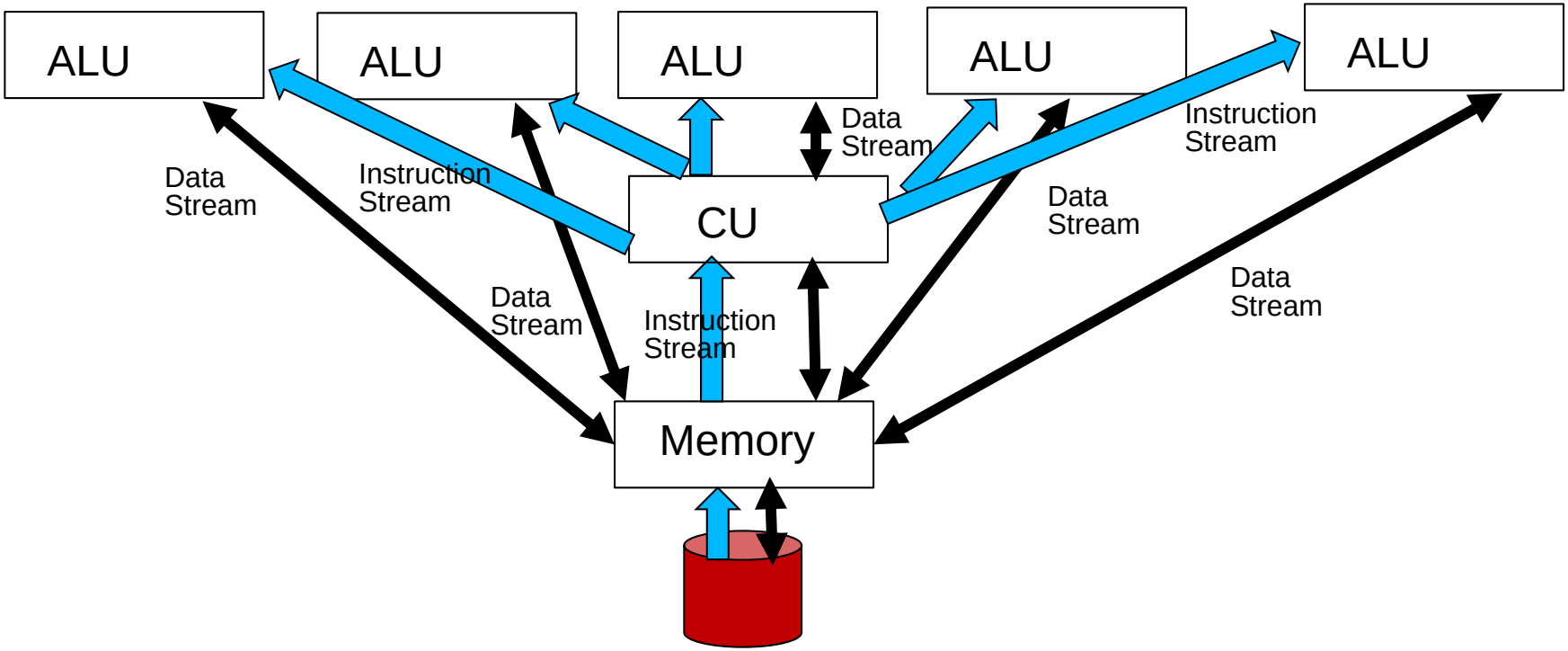
# Classification of Parallel Computers

- Based on how instructions and data are divided (Flynn's Taxonomy)
- Based on how memory is organized.
- Based on Synchronization
- Based on how processors are connected with each other.

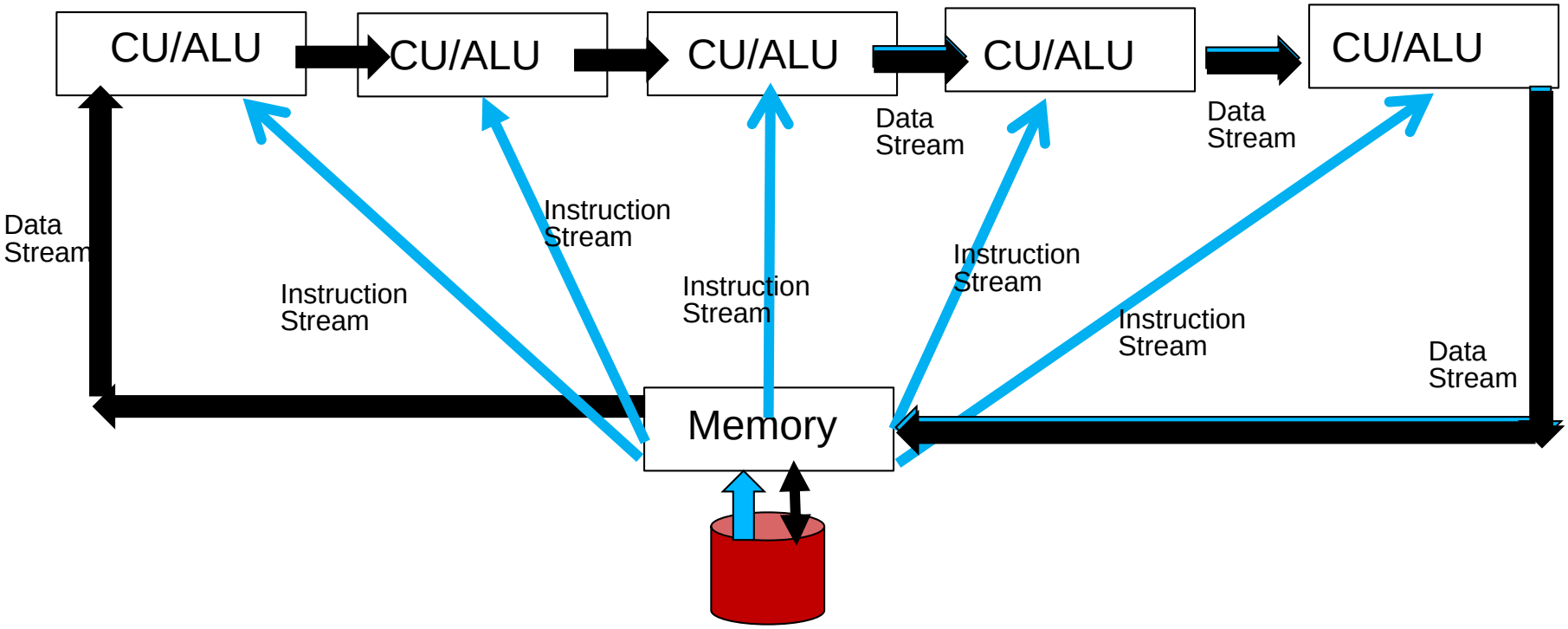
# Single Instruction Stream Single Data Stream (SISD)



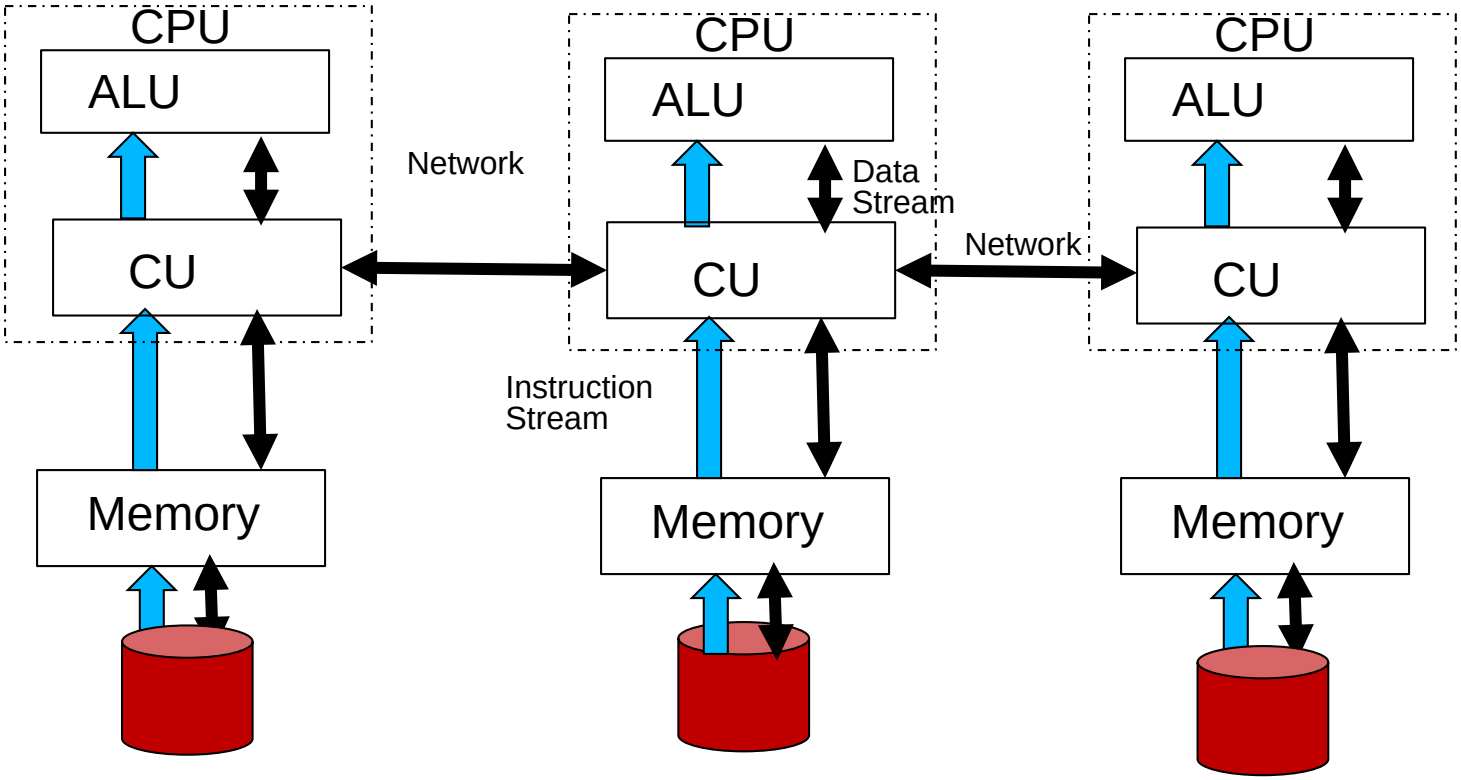
# Single Instruction Stream Multiple Data Stream (SIMD)



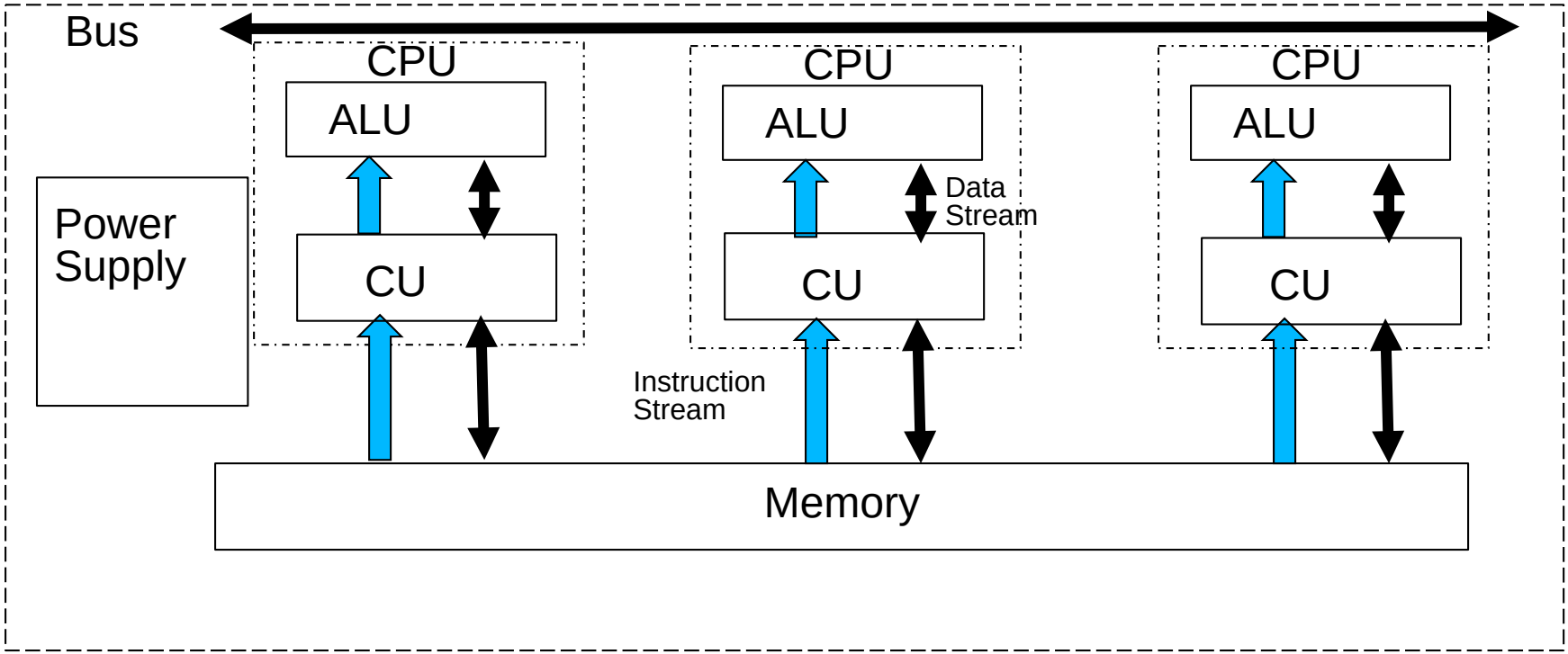
# Multiple Instruction Stream Single Data Stream (MISD)



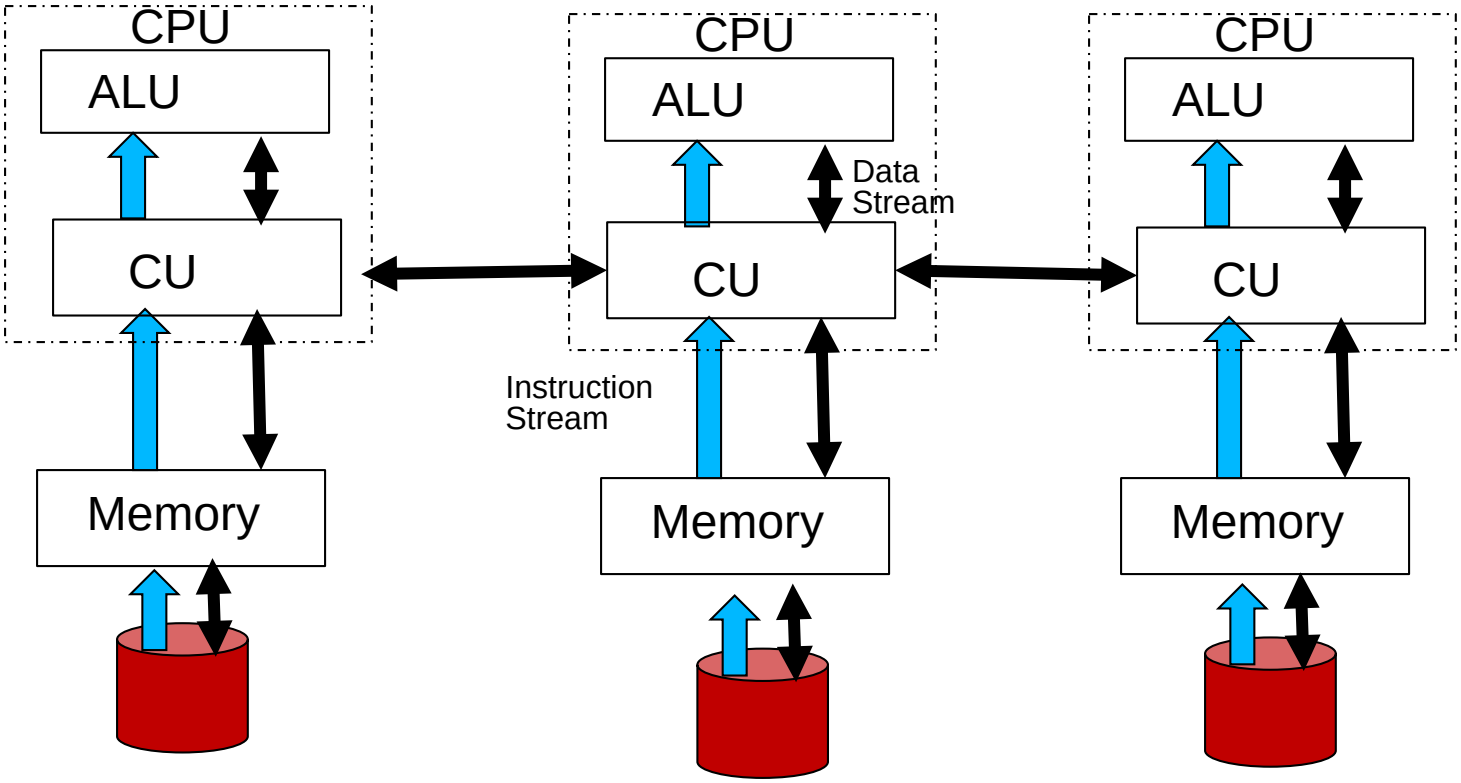
# Multiple Instruction Stream Multiple Data Stream (MIMD)



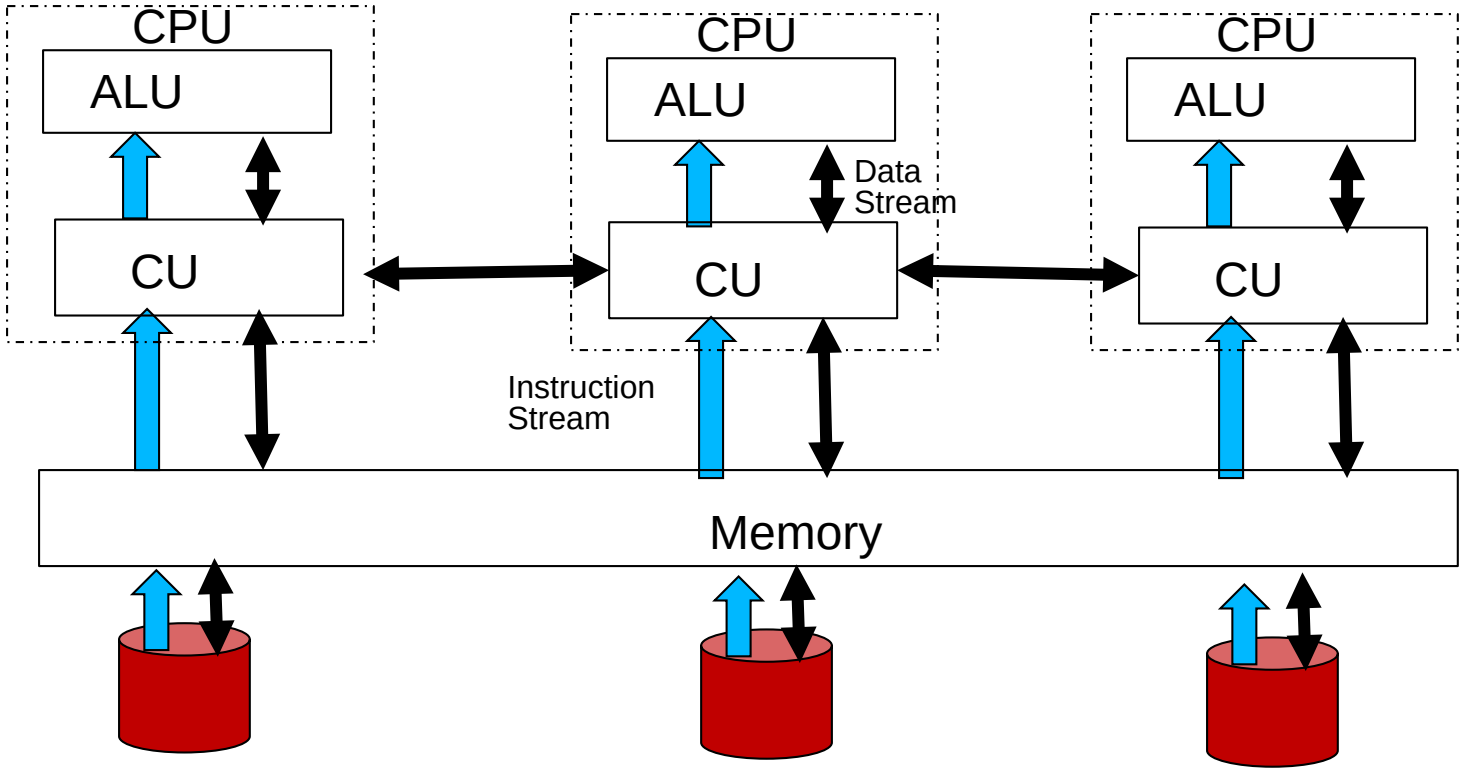
# Multiple Instruction Stream Multiple Data Stream (MIMD) (multicore chip)



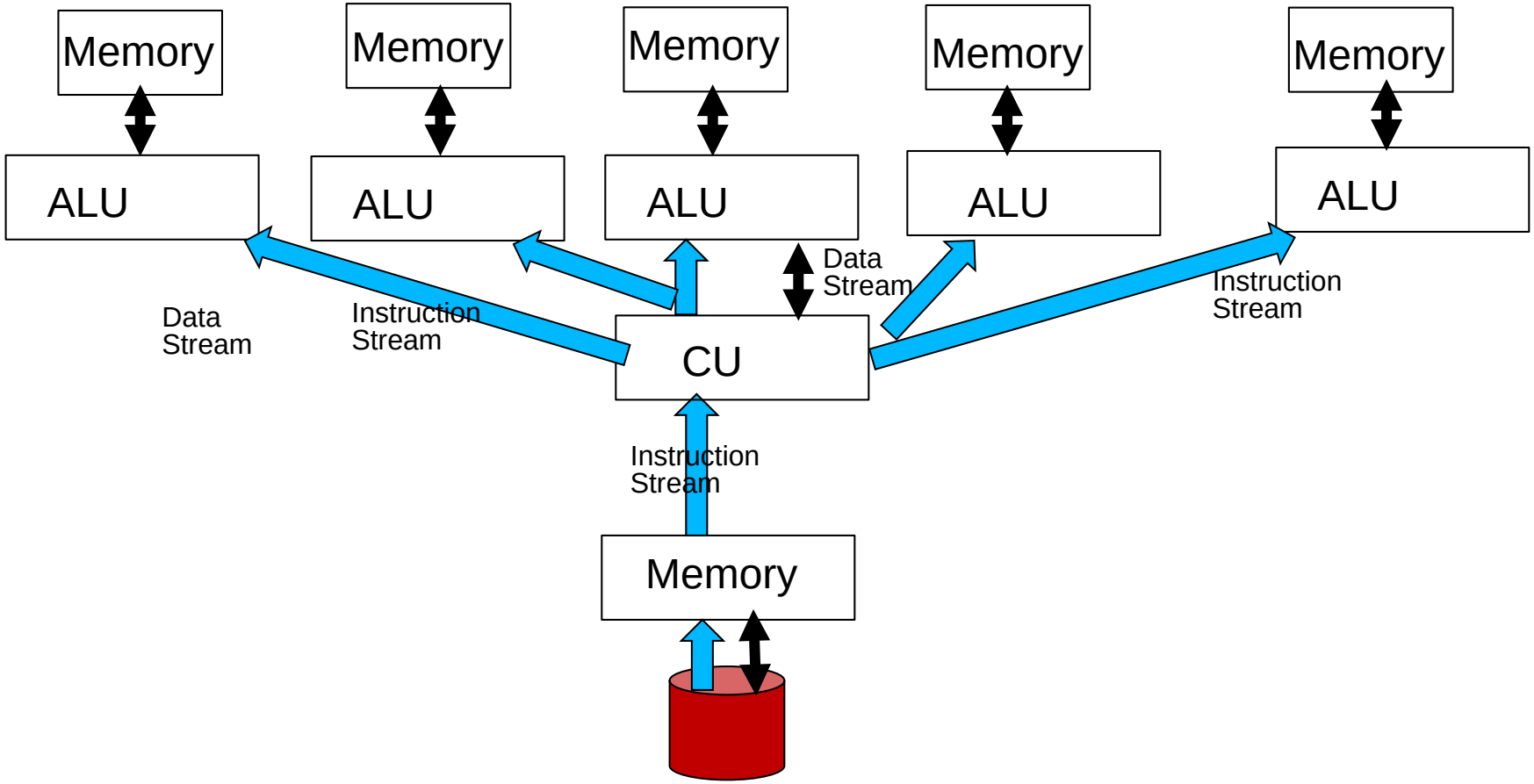
# Multiple Instruction Stream Multiple Data Stream (MIMD) Distributed Memory



# Multiple Instruction Stream Multiple Data Stream (MIMD) shared Memory



# A Practical SIMD Machine (GPU)



# Instructions and Data Distribution

- Computer architectures can be classified with respect to *instruction stream* and *data stream* (Flynn's Taxonomy).
- An instruction stream is a sequence of operations performed by the computer, and a data stream is a sequence of items on which the instructions operate.
- There have been a few extensions and modifications to this taxonomy. But is still widely used and computers are classified according to it.

# Flynn's Taxonomy

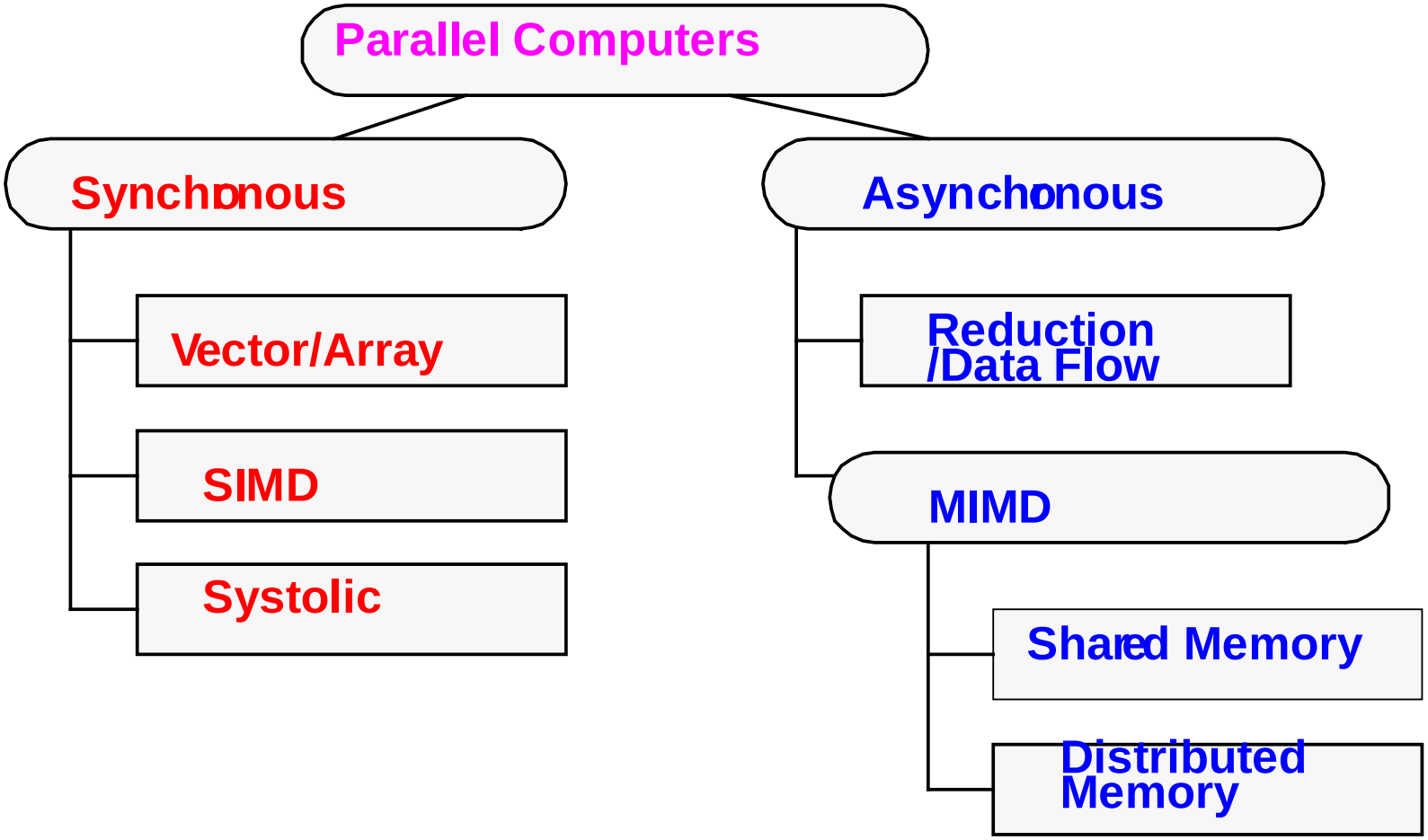
**SISD: With 1 processors, the RAM model is a *single-instruction* (stream) *single-data* (stream) (SISD) machine, which models conventional sequential computers.**

***MIMD:* multiple-instruction (streams) *multiple-data* (streams) machine. A special case of MIMD is the *single-program multiple-data* (SPMD) computation, where all the processes execute the same program, parameterized by the process index.**

**SIMD: *single-instruction* (stream) *multiple-data* (streams) machine. At each cycle all processors must execute the same instruction, and there is only one instruction stream.**

**Difference between SIMD and SPMD: in an SPMD computation, different instructions can be executed at the same cycle.**

**MISD:?**



# Vector / Array Parallelism

- When the problem size is small for each processors, we say the workload is fine-grained.
- When the problem size is large for each processors, we say the workload is coarse-grained.
- The key to obtaining speedup in this manner is overlapping fine-grained tasks.
- This type of parallelism is good for handling batches of data when the operations can be broken down into small stages.

# SIMD Computers

The efficiency of SIMD depends on how many exam sheets are processed in parallel.

It is suitable if there are large number of operations that can be done in parallel.

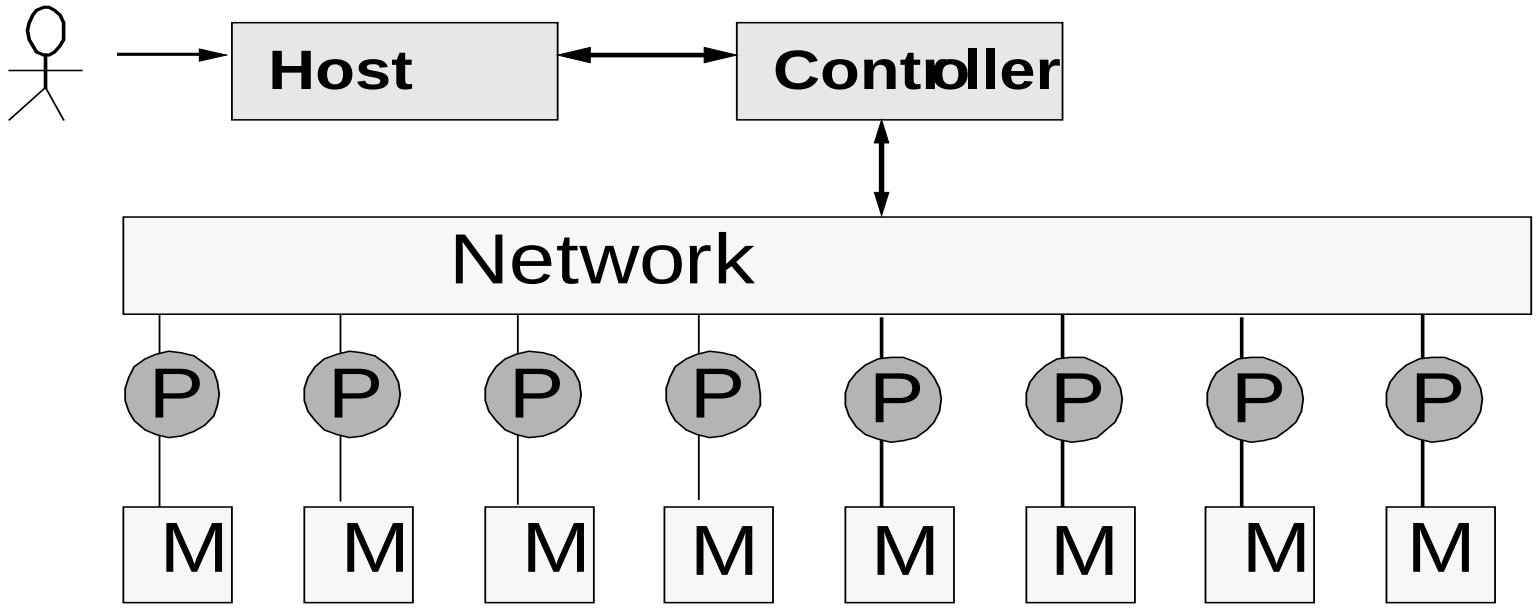
The Control unit has its own memory for storing system and user programs and the processors have their own memories.

Usually the number of processing elements (PEs) is very large.

Serial instructions are executed in the control unit while the parallel instructions are broadcast to the PEs to be executed in a lock step fashion.

The vector data is distributed to the local memories before parallel execution starts.

# SIMD Computers




A SIMD parallel computer
(all processors work under the control of the controller for each instruction; all processors either execute the same instruction or sit idle)

# MIMD Paradigm

**MIMD paradigm means that processors can be simultaneously executing different instructions on different data.**

**There is no supervisor and it is asynchronous.**

# Problems in MIMD

**Approach 2:**

**Keep all exam sheets in the data vault and place a lock on it**

**When a grader has finished its grading, it unlocks the exam sheet.**

**Multiple graders can read but only one grader can write at a time**

**Approach 3:**

**Divide the exam sheets among grader**

**Each grader works on its own pile of exam sheets**

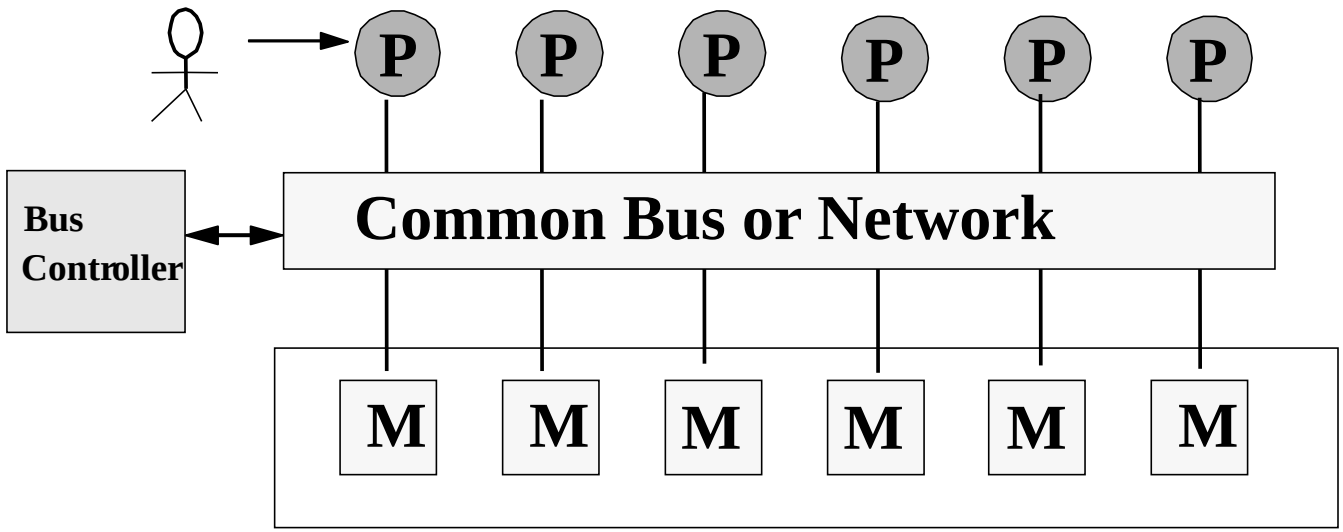
**If a grader needs an exam sheet, whose folder is with some other grader, it needs to send a message to that grader and get a copy of the exam sheet**

# Shared versus Distributed Memory

- **MIMD machines are further classified as shared versus distributed memory.**
- **The *address space* of a process is the set of memory locations accessible by the process.**
- ***Single address space*, from the programmer's viewpoint.**
- ***Multiple address spaces*.**

# Shared Memory Computers

- In a shared-memory parallel computer, there is a single global memory space which is accessed by all processors.
- The processors are connected to the memory usually by a common bus or a switch based multistage interconnection network.
- Programming shared memory machines is relatively easier.
- Graders work independently.

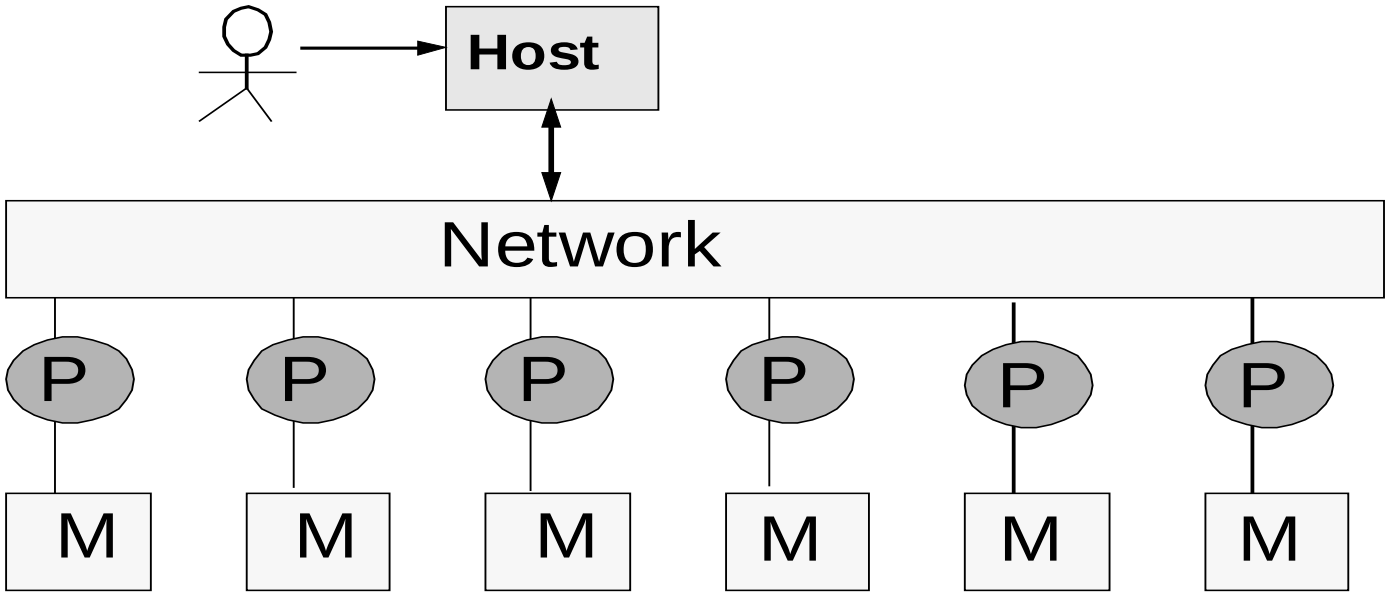



**A shared-memory parallel computer (usually users login into one of the processors)**

# Distributed-Memory Computers

- On the other hand, in a distributed-memory parallel computer, processors have their own individual memories.
- The processors are connected to each other by a communication network
- Each processor executes its own program and accesses its own memory (therefore variables in a program are local).
- To get the data from the memory of some other processors, message- passing through the network has to be employed.
- Distributed-memory computers are also called message-passing systems because interprocessor communication is done by message- passing.

# Distributed-Memory Computer




**A distributed-memory MIMD parallel computer (host is responsible for I/O and user access but all processors execute their individual programs)**

# Distributed-Memory MIMD Grader

- Most importantly, the distributed-memory model can be easily (relatively!) implemented on a network of multiple independent computers (e.g. workstations).
- There may or may not be host. In a distributed-memory MIMD bank both the customers and their exam sheets need to be divided among the graders.
- The graders have their own small “trays” of exam sheets.

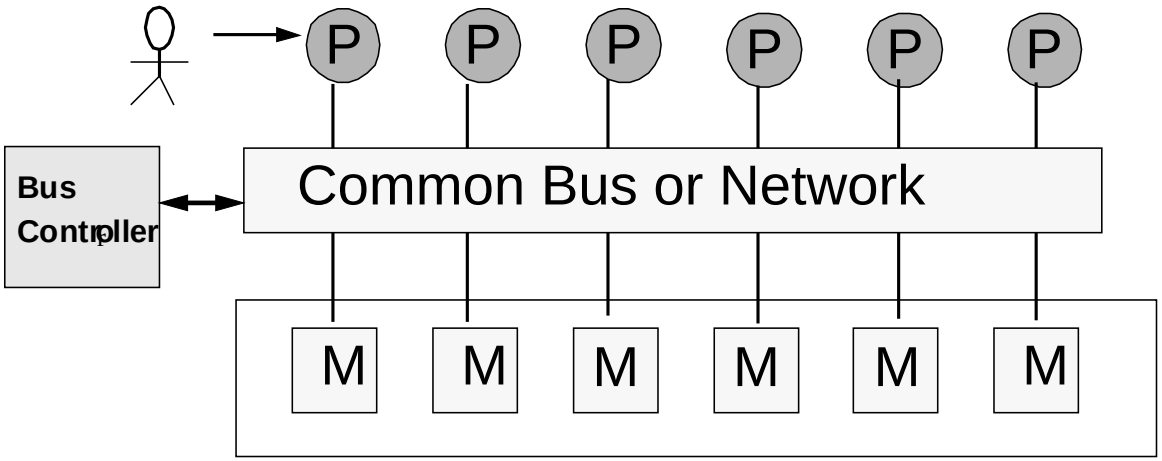
# Shared-Memory Computers

**In a shared-memory parallel computer, there is a single global memory space which is accessed by all processors.**

**The processors are connected to the memory usually by a common bus or a switch based multistage interconnection network.**

**This attribute specifies how the machine model handles shared- memory access conflicts.**

- ***Exclusive read exclusive write* (EREW) rule**
- **C*oncurrent read exclusive write* (CREW)**
- **C*oncurrent read concurrent write* (CRCW)**
- **Programming shared memory machines is relatively easier**




**A shared-memory parallel computer (usually users login into one of the processors)**

# Advantages of SIMD Computers

SIMD computers are very well suited for regular problems with large data sizes. That is, it is suitable if there are large number of operations that need can be done in parallel.

The efficiency of SIMD depends on how many exam sheets are processed in parallel.

There is not need for explicit synchronization because processors are implicitly synchronized.

Programs are easier to write and debug.

Generally less memory is need because of a single program.

# Advantages of MIMD Computers

- More general and flexible in parallelizing an application.
- MIMD computers do not need the added cost associated with a global control unit.
- They are very well suited for coarse-grained problems.
- A MIMD computer can simulate SIMD but the converse is not true.

# Algorithmic Models of Parallel Computers

- A *parallel machine model* (also known as *programming model*, *type architecture*, *conceptual model*, or *idealized model*) is an abstract parallel computer from the programmer's viewpoint, analogous to the von Neumann model for sequential computing.
- The abstraction should capture implicitly the relative costs of parallel computation.
- A model should be precise enough about performance without being too explicit about the implementation details.
- An abstract model provides many benefits to computer architects, software developers, programmers, and algorithm designers.
- Every parallel computer has a *native model* that closely reflects its own architecture.

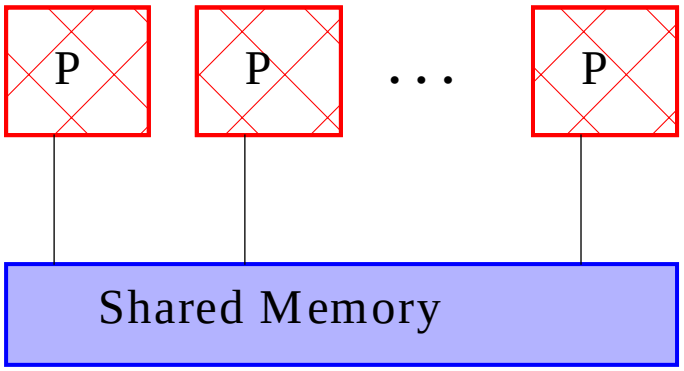
# The PRAM Model

- A *parallel random-access machine* (PRAM) of size *n* consists of *n* processors, all accessing a shared memory.
- A parallel program on a PRAM consists of *n* processes, where the *i*-th process resides in the *i*-th processor and is composed of a sequence of instructions.
- Each processor executes one instruction at each basic time step (called a *cycle*). The instructions include data transfer, arithmetic/ logic, control flow, and I/O instructions, as can be found in a typical sequential computer.

# Shared-Memory Acess

- One important feature of the theoretical PRAM model is that all processes have equal access time to all memory locations. We say that such a machine has *uniform memory access* (UMA).
- On the other hand, if it takes different amounts of time for processes to access different word locations, we say the machine has *nonuniform memory access* (NUMA).

# The PRAM Model




MIMD
Fine grain
Tightly synchronous
Zero overhead
Shared variable

# Scalable Computer Architectures

- A common feature of modern computers is parallelism.
- Even within a single processor, parallelism has been exploited in many ways.
- The concept of S*calability* is important, which includes parallelism.
- The scalability concept is central to modern computer design.
- A computer system is called *scalable* if it can *scale up* (i.e., improve its resources) to accommodate ever-increasing performance and functionality demand and/or *scale down* (i.e., decrease its resources) to reduce cost.

# Attributes of scalable Architecture

**Scalability involves the ability of the system to scale up or down. Saying that a system is scalable implies the following:**

***Functionality and Performance*: The scaled-up system should provide more functionality or better performance. The total computing power of the system should increase proportionally to the increase in resources.**

***Scaling in Cost*: The cost paid for scaling up must be reasonable. A rule of thumb is that scaling up *n* times should incur a cost of no more than *n* or *n* log *n* times.**

***Compatibility*: The same components, including hardware, software, and application software, should still be usable with little change.**

**The main motivation for a scalable system is to provide a flexible, cost-effective information processing tool.**

# Scalability Across All Computer Classes

**The scalability concept has several advantages:**

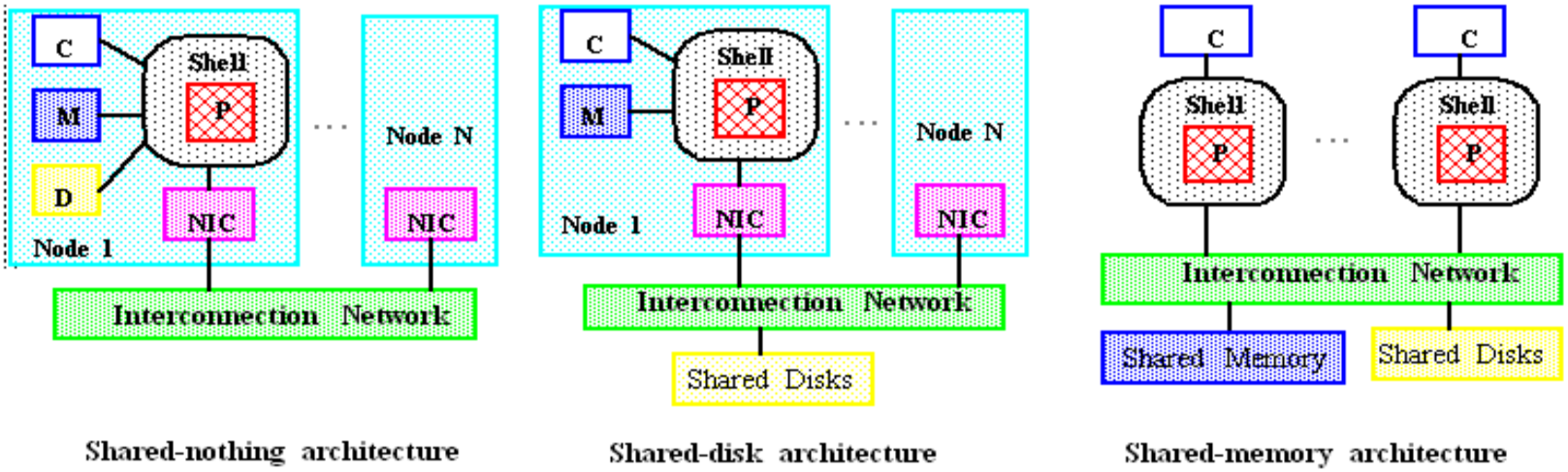
- **It satisfies different performance and cost requirements. For instance, a user could first purchase a low-end system. When his performance requirement increases, he could scale up the system, such that the original software and hardware components can still be used.**

- **High-end machines may use low-end components to cut cost. For example, with high volume, PCs have low-cost, off-the-shelf components. With a scalable architecture, supercomputers can use such components to reduce system cost.**

- **The cutting-edge technology developed for high-end systems may eventually migrate down the pyramid to improve the performance of low-end systems, if cost-effectiveness can be increased with improved production technology.**

# Converging System Architectures

**Scalable parallel computers are roughly of three architectures, which share much commonality with increasing levels of resources sharing**




Shared-nothing architecture | Shared-disk architecture | Shared-memory architecture

C: Cache D: Disk M:Memory
NIC: Network interface circuitry P:Processor

## *Shared-nothing* architecture

**Consists of a number of *nodes* connected by an *interconnection network*. The node usually follows a *shell* architecture, including a (board-level) cache, a local memory, a *network interface circuitry* (NIC), and a disk.**

## *Shared-disk* architecture

**Disk modules are moved out of the nodes to be shared among the nodes.**

## *Shared-memory* architecture,

**The main memory is shared, that is, it has a single address space.**

# Dimensions of Scalability

**There are different ways of scaling (improving or reducing system resources).**

## (i) Size Scalability

**Gaining higher performance or functionality by increasing the *machine size* (i.e., the number of processors), investing in more storage (cache, main memory, disks), improving the software, etc.**

**Size scalability measures the maximum number of processors a system can accommodate. Not all parallel computers are equally size-scalable.**

**The communication subsystem, including interconnect, interface, and communication software, may also need to be improved.**

## (ii) Scaling Up in Other Resources

**One can keep the same number of processors, but invest in more memory, bigger off-chip caches, bigger disks, etc.**

## (iii) Software Scalability

**The software of a scalable computer system can be improved in many ways, as suggested below:**

- **A newer version of the operating system, with more functionalities**
- **A better compiler with more efficient optimizations**
- **More efficient mathematical and engineering libraries**
- **More efficient and easy-to-use applications software**
- **More user-friendly programming environment**

# (iv) Application Scalability

**The same program should run with proportionally better performance on a scaled-up system. There are three points worth noting when studying application scalability:**

**a) Many practical parallel applications have built-in limitations regarding machine size and problem size. These limitations cannot be exceeded by simply increasing machine resources. The program has to be significantly modified to handle more processors.**

**b) We should consider an application on a specific machine as a combination. This application/machine pair is sometimes referred to as a *system*.**

**c) Application scalability does not depend on just machine size and problem size. It also depends on the memory capacity, I/O capability, and communication capability of the machine. All these factors jointly affect the scalability.**

## (v) Technology Scalability

***Technology scalability*** **applies to a scalable system that can adapt to changes in technology. It can be further divided into three categories: *generation scalability*, *space scalability*, and *heterogeneity scalability*.**

## (vi) Generation (Time) Scalability

**A system can be scaled up using the next-generation components, such as a faster processor, a faster memory, a newer version of operating system, a more powerful compiler, etc.**

**The fastest evolving component of a computer system is the processor, which is expected to double its performance approximately every 18 months to 2 years.**

**The slowest evolving is probably programming languages.**

## (vii) Space Scalability

**This refers to the ability of a system to scale from multiple processors in a box, in a room, or in a building to multiple buildings and geographic regions.**

## (viii) Heterogeneity Scalability

**This property refers to how well a system can scale up by integrating hardware and software components supplied from different designers or vendors.**

# Performance Attributes

| Terminology | Notation | Unit |
|---|---|---|
| Machine size (cores, nodes) | $n$ | Dimensionless |
| Clock rate | $f$ | GHz |
| Workload | $W$ | Gflop, Tflop, Pflop |
| Sequential execution time | $T_1$ | s |
| Parallel execution time | $T_n$ | s |
| Speed | $P_n = W / T_n$ | Gflop/s, Tflop/s, Pflop/s |
| Speedup | $S_n = T_1 / T_n$ | Dimensionless |
| Efficiency | $E_n = S_n / n$ | Dimensionless |
| Utilization | $U_n = P_n / (nP_{peak})$ | Dimensionless |
| Startup time | $t_0$ | μs |
| Asymptotic bandwidth | $r_\infty$ | MB/s, GB/s |

# Parallelism Overhead

**The parallelism and the interaction operations are the sources of *overhead.* These overheads can be divided into the following four types:**

- ***Parallelism overhead* caused by process management**
- ***Communication overhead* caused by processors exchanging information**
  - ***point-to-point* communication overhead**
  - ***Collective communication* overhead.**
- ***Synchronization overhead* in executing synchronization operations**
- ***Load imbalance overhead* incurred, when some processors are idle while the others are busy**

# Modelling Communication Overhead

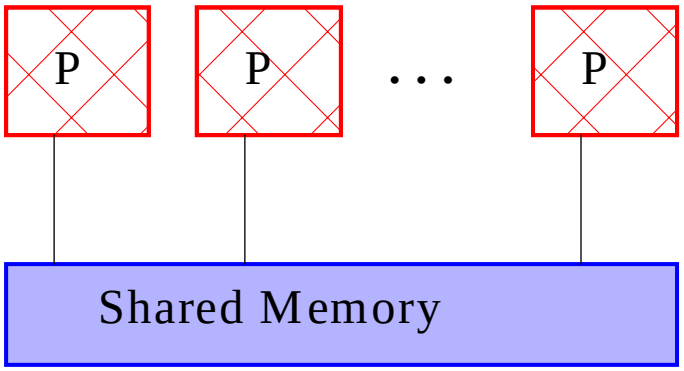
**The *start-up time*, denoted by $t_0$, is the time (in μs) to communicate a 0-Byte or a short (e.g., one-word) message. The start-up time is also known as the *communication latency*, or simply *latency*.**

**The *asymptotic bandwidth*, denoted by $r_\infty$, is the rate (in MB/s) for communicating a long message.**

# The Performance of PRAM Model

- **The machine size *n* can be arbitrarily large.**
- **The basic time step is called a *cycle*.**
- **Within a cycle, each processor executes exactly one instruction. The instruction could be a *null* instruction (i.e., do nothing), in which case we say the processor is idle at that cycle.**
- **All processors implicitly synchronize at each cycle, and the synchronization overhead is assumed to be zero. Communication is done through reading and writing of shared variables.**
- **The communication and parallelism overheads are ignored. Thus the only overhead accounted for in a PRAM program is the load imbalance overhead.**
- **An instruction can be any *random-access machine* instruction.**

# The PRAM Model




MIMD
Fine grain
Tightly synchronous
Zero overhead
Shared variable

# Example of PRAM Operation

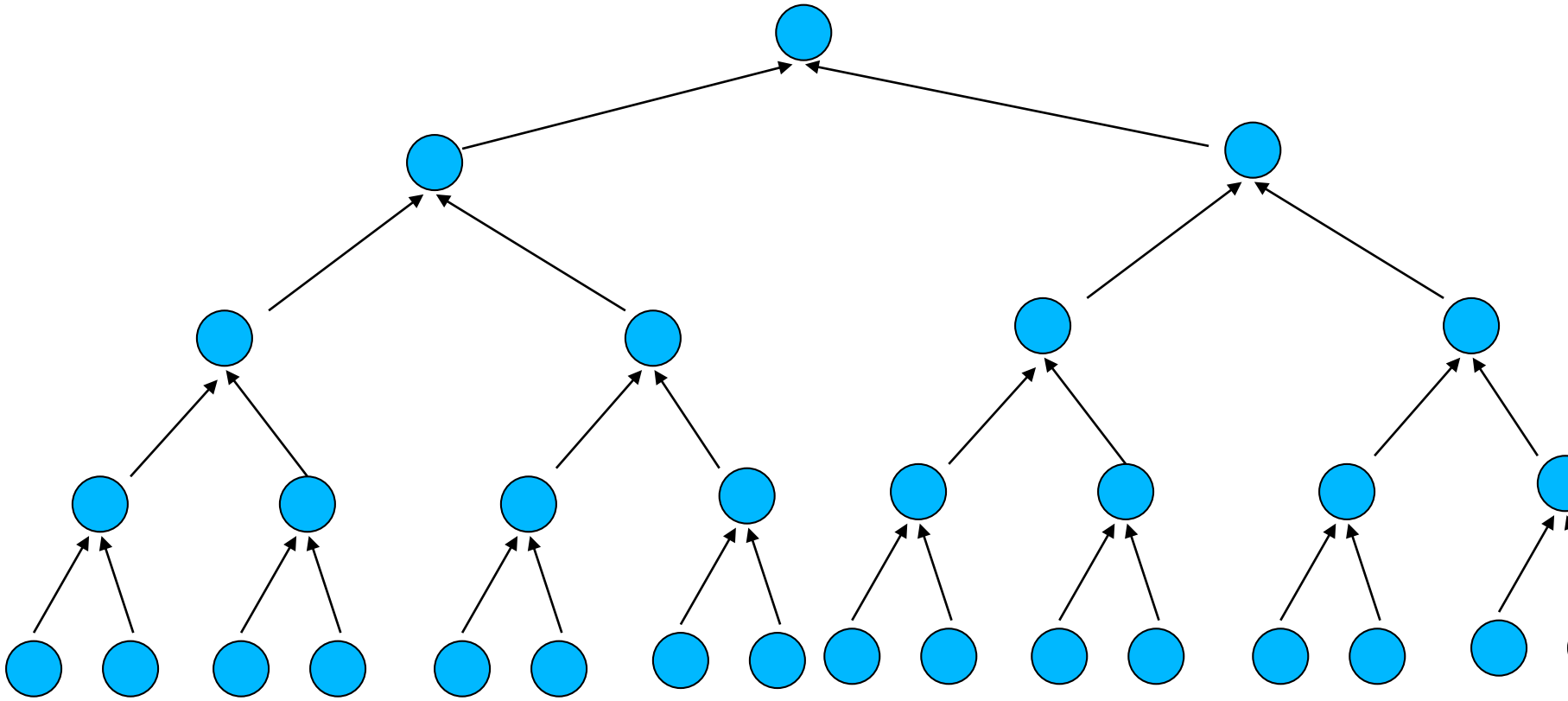
**To compute the inner product *s* of two *N*-dimensional vectors *A* and *B* on an *n*-processor EREW PRAM computer**

**One can assign each processor to perform 2*N*/*n* additions and multiplications to generate a local result in 2*N*/*n* cycles, and then one can add all the *n* local sums to a final sum *s* by a treelike reduction method in $\log n$ cycles.**

**The total execution time is $2N/n + \log n$ cycles. Compared with the sequential algorithm which takes 2*N* cycles, the parallel PRAM algorithm has a speedup of $n/\{1 + \lceil n/(2N) \rceil \log n\} \rightarrow n$ , when $N \gg n$ .**

**The time complexity of most PRAM algorithms is expressed as a function of *N* and *n*. The PRAM model does account for the load imbalance overhead.**

**Due to its simplicity and clean semantics, the PRAM model has been favored by many computer scientists. This model is also widely used in analyzing the complexity of parallel algorithms.**



# Shortcoming of the PRAM Model

- **Unrealistic assumptions of zero communication overhead and instruction-level synchrony.**
- **The time complexity of a PRAM algorithm is often expressed in the big-*O* notation, which is often misleading**

# Computational Complexity in PRAM steps

**Suppose three PRAM algorithms *A*, *B*, and *C* have time complexities of 7*n*, (*n* log *n*)/4, and *n* log log *n*, respectively, when executing on an *n*-processor PRAM computer.**

**According to the big-*O* notation, algorithm *A* is the fastest (*O*(*n*)), followed by *C* (*O*(*n* log log *n*)), with *B* being the slowest (*O*(*n* log *n*)).**

**In reality, on machines with no more than 1024 processors, we have**

$$\log n \leq \log 1024 = 10$$

**and**

$$\log \log n \leq \log \log 1024 < 4$$

**Thus the fastest algorithm is really *B*, followed by *C*, while *A* is the slowest for systems with less than 1024 processors.**

**With unrealistic assumptions, the PRAM model has not been used as a machine model for *real-life* parallel computers.**

# The BSP Model

**The *bulk synchronous parallel* (BSP) model aims to overcome the shortcomings of the PRAM model, while keeping its simplicity.**

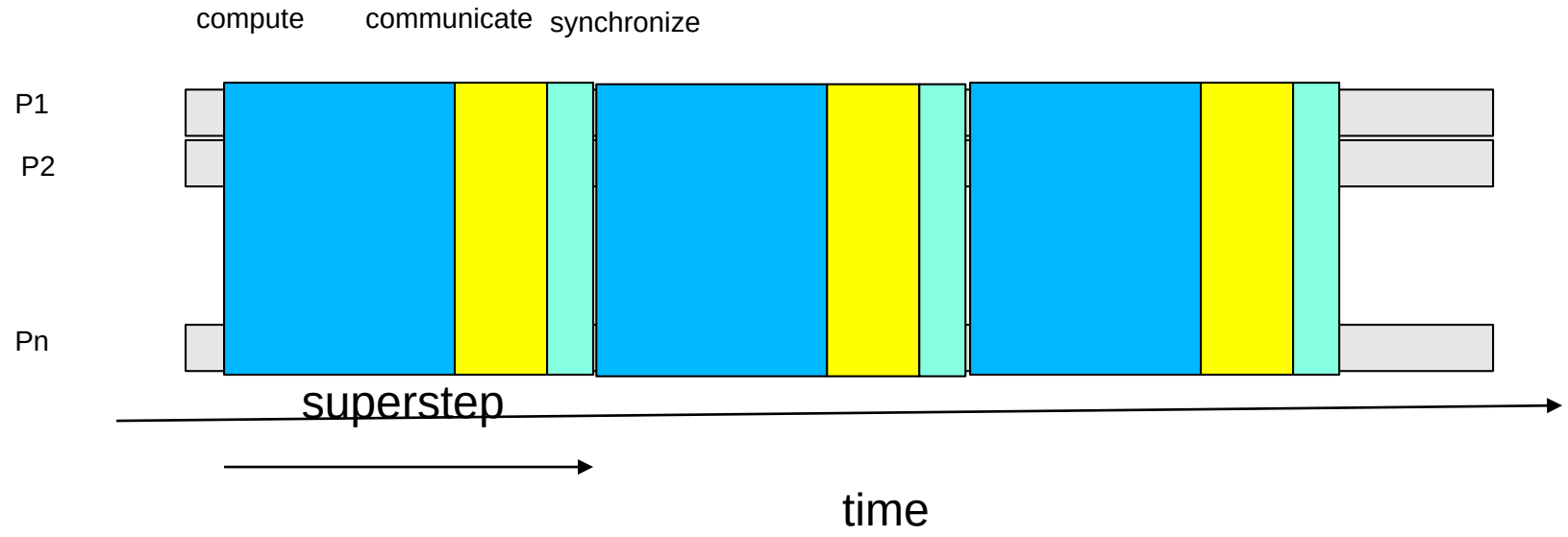
**A *BSP computer* consists of a set of *n* processor/memory pairs (*nodes*) that are interconnected by a communication network.**

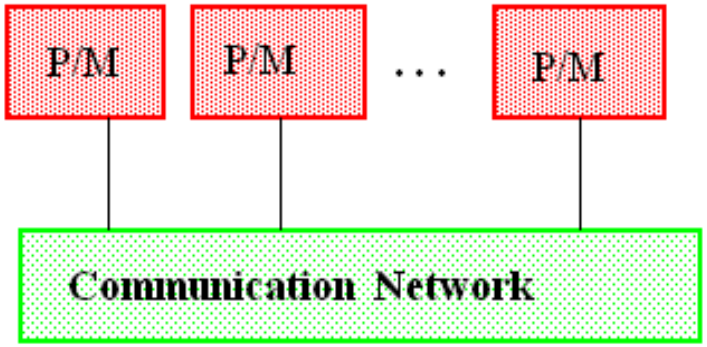
**A BSP program has *n* processes, each residing on a node.**

**The basic time unit is a *cycle* (or *time step*).**

**The program executes as a strict sequence of *supersteps*. In each superstep, a process executes the computation operations in at most *w* cycles, a communication operation that takes *gh* cycles, and a *barrier synchronization* that takes *l* cycles.**

**The barrier forces the processes to wait so that all processes have to finish the current superstep before any of them can begin the next superstep.**






*w*: Maximum computation time within each superstep
*l*: barrier synchronization overhead
*g*: *h* relation coefficient

MIMD
Superstep:
- Computation
- Communication
- Barrier

Variable grain
Loosely synchronous
Nonzero overhead
Message passing or shared variable

The BSP computer is an MIMD system.

It is loosely synchronous at the superstep level, compared to the instruction-level tight synchrony in the PRAM model.

Within a superstep, different processes execute asynchronously at their own paces.

**The BSP model does not require any specific memory interaction mechanism**

**Within a superstep, each computation operation uses only data in its local memory.**

**These data are put into the local memory either at the program start-up time or by the communication operations of previous supersteps.**

**A communication is always realized in a point-to-point manner. Thus it is not allowed for multiple processes to read or write the same memory location in the same cycle.**

**All memory and communication operations in a superstep must completely finish before any operation of the next superstep begins.**

# Communication model of BSP

- **The BSP model abstracts the communication operations in a BSP superstep by the *h* relation concept.**
- **An *h relation* is an abstraction of any communication operation, where each node sends at most *h* words to various nodes and each node receives at most *h* words.**
- **On a BSP computer, the time to realize any *h* relation is no more than *gh* cycles, where g is a constant decided by the machine platform.**

# BSP Versus PRAM

**The BSP model is more realistic than the PRAM model because it accounts for all overheads except the *parallelism overhead* for process management. The execution time of a superstep is determined as follows:**

- **To account for load imbalance, the computation time *w* is the maximum number of cycles spent on computation operations by any processor.**
- **The synchronization overhead is *l*, which has a lower bound of the communication network latency (i.e., the time for a word to propagate through the physical network) and is always greater than zero.**
- **The communication overhead is *gh* cycles, where *g* is the proportional coefficient for realizing an *h* relation. The value of *g* is platform-dependent, but independent of the communication pattern. In other words, *gh* is the time to execute the most time-consuming *h* relation.**
- **The time for a superstep is estimated by the sum *w* + *gh* + *l*.**

- **The BSP model allows the overlapping of the computation, the communication, and the synchronization operations within a superstep. If all three types of operations are fully overlapped, the time for a superstep becomes max(*w*, *gh*, *l*). However, we will use only the more conservative *w* + *gh* + *l*.**

# Parallel execution using the BSP machine model

**Algorithm for inner-product using 8-processor BSP computer in 4 supersteps:**

**Superstep 1**

***Computation*: Each processor computes its local sum in $w = 2N/8$ cycles.**

***Communication*: Processors 0, 2, 4, 6 send their local sums to processors 1, 3, 5, 7. Apply 1 relation here.**

***Barrier synchronization*.**

**Superstep 2**

***Computation*: Processors 1, 3, 5, 7 each perform one addition ($w = 1$).**

***Communication*: Processors 1 and 5 send their intermediate results to processors 3 and 7. A 1 relation is applied here.**

***Barrier synchronization*.**

**Superstep 3**

***Computation*: Processors 3 and 7 each perform one addition. ($w = 1$).**

***Communication*: Processor 3 sends its intermediate result to processor 7. Apply 1 relation here.**

***Barrier synchronization*.**

**Superstep 4**

***Computation*: Processor 7 performs one addition ($w = 1$) to generate the final sum.**

**No more communication or synchronization is needed.**

**The total execution time is $2N/8 + 3g + 3l + 3$ cycles.**

**In general, the execution time is $2N/n + \log n(g + l + 1)$ cycles on an *n*-processor BSP. This is in contrast to the $2N/n + \log n$ time on a PRAM computer.**

**The two extra terms, $\log n$ and $l\log n$ correspond to *communication* and *synchronization overheads*, respectively.**

# Physical Machine Models

**Large-scale computer systems are generally classified into six practical machine models:**

- *Single-instruction multiple-data* (SIMD), i.e. GPUs, machines
- *Parallel vector processor* (PVP), MISD
- *Symmetric multiprocessor* (SMP, Multicore Chip), MIMD (shared memory)
- *Massively parallel processor* (MPP), MIMD (distributed memory)
- Cluster of workstations (COW), and the *distributed shared memory* (DSM) multiprocessors, MIMD

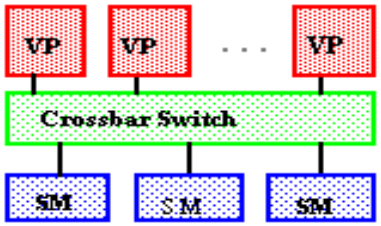
**SIMD computers are used mostly for special-purpose applications. The remaining models are all MIMD machines.**

**Most modern parallel computers are built with commercially available, off-the-shelf commodity hardware and software components. The only exception is the PVP machines, where many building blocks are custom-made.**

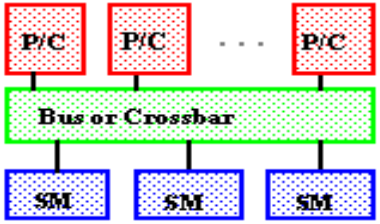
# Semantic Attributes of Parallel Machine Models

| Attributes | PRAM | PVP/SMP/Multicore | DSM | MPP/COW |
|---|---|---|---|---|
| Homogeneity | MIMD | MIMD | MIMD | MIMD |
| Synchrony | Instruction level synchronous | Asynchronous or loosely synchronous | Asynchronous or loosely synchronous | Asynchronous or loosely synchronous |
| Interaction mechanism | Shared Variable | Shared Variable | Shared Variable | Message passing |
| Address space | Single | Single | Single | Multiple |
| Access cost | UMA | UMA | NUMA | NORMA |
| Memory model | EREW, CREW or CRCW | Sequential consistency | Sequential or weak ordering | N/A |
| Example machines | Theoretical model | IBM R50, Cray T-90, multicore | Stanford DASH, SGI Origin 2000 | Stampede, Summit Berkeley NOW |

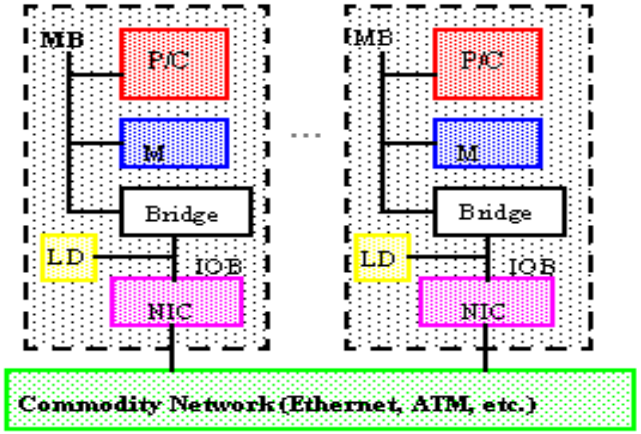
# Examples



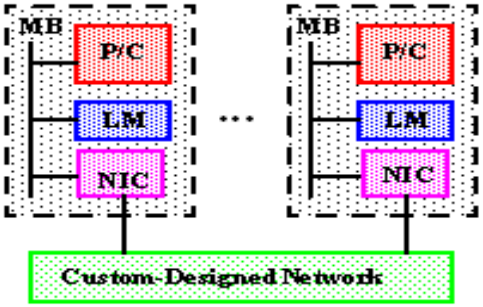

Parallel vector processor



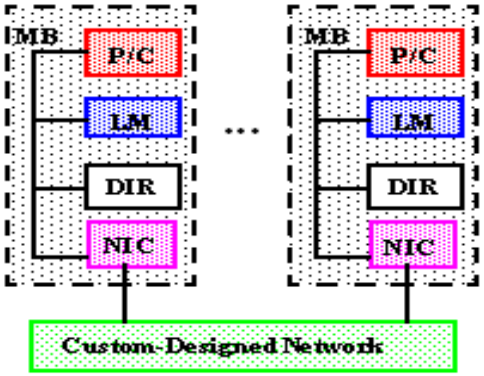

Symmetric multiprocessor




Cluster of workstations




Massively Parallel Processor




Distributed shared memory machine

| | | |
|---|---|---|
| Bridge: Interface between memory bus and I/O bus | | |
| IOB: I/O bus | LD: Local disk | LM:Local memory |
| MB: Memory bus | NIC: Network interface circuitry | |
| P/C: Microprocessor and cache | SM: Shared memory | VP:Vector processor |

# Symmetric Multiprocessors

- **Examples include the IBM R50, the SGI Power Challenge, and the DEC Alpha server 8400.**
- **An SMP system uses commodity microprocessors with on-chip and off-chip caches.**
- **The processors are connected to a shared memory through a high-speed snoopy bus or a switch.**
- **It is important for the system to be *symmetric*, in that every processor has equal access to the shared memory, the I/O devices, and the operating system services.**
- **SMP systems are heavily used in commercial applications, such as databases, on-line transaction systems, and data warehouses.**
- **Most PVPs and SMPs have at most 64 processors, such as the Sun Ultra Enterprise 10000. The limitation is mainly caused by using a centralized shared memory and a bus or crossbar system.**

# Massively Parallel Processors

**The term MPP (*massively parallel processor*) generally refers to a very large-scale computer system having the following features:**

- **Commodity microprocessors in processing nodes.**
- **Physically distributed memory over processing nodes.**
- **An interconnect with high communication bandwidth and low latency.**
- **Can be scaled up to hundreds or thousands of processors. Like the multiprocessor model, it is an asynchronous MIMD machine.**
- **Processes are synchronized through blocking message-passing operations, not shared-variable synchronization operations.**
- **The program consists of multiple processes, each having its private address space. Processes interact by passing messages.**

**Examples are Tianhe-2, Stampede.**

# Distributed Shared-Memory (DSM) Machines

- **Examples are the Stanford DASH, and Cray T3D.**
- ***Cache directory*** **(DIR) is used to support distributed coherent caches.**
- **The main difference between DSM machines and SMP is that the memory is physically distributed among different nodes. However, the system hardware and software create an illusion of a single address space to application users.**
- **A DSM machine can be also implemented with software extensions on a network of workstations such as the TreadMarks.**

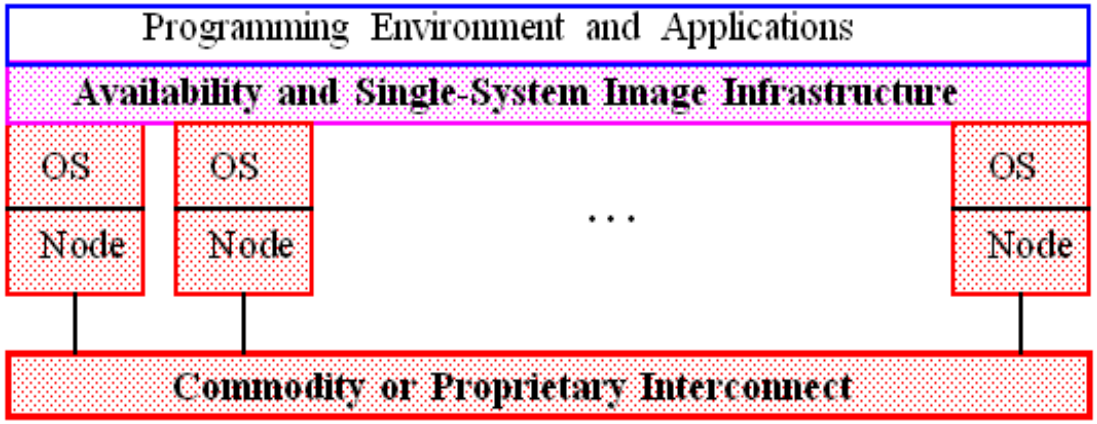
# Basic Concepts of Clustering

## Cluster Characteristics

**A *cluster* is a collection of complete computers (nodes), that are interconnected by a high-performance network or a LAN.**

**Typically, each computer node is an SMP server, a workstation, or a personal computer.**

**All cluster nodes must be able to work together collectively as a single, integrated computing resource.**



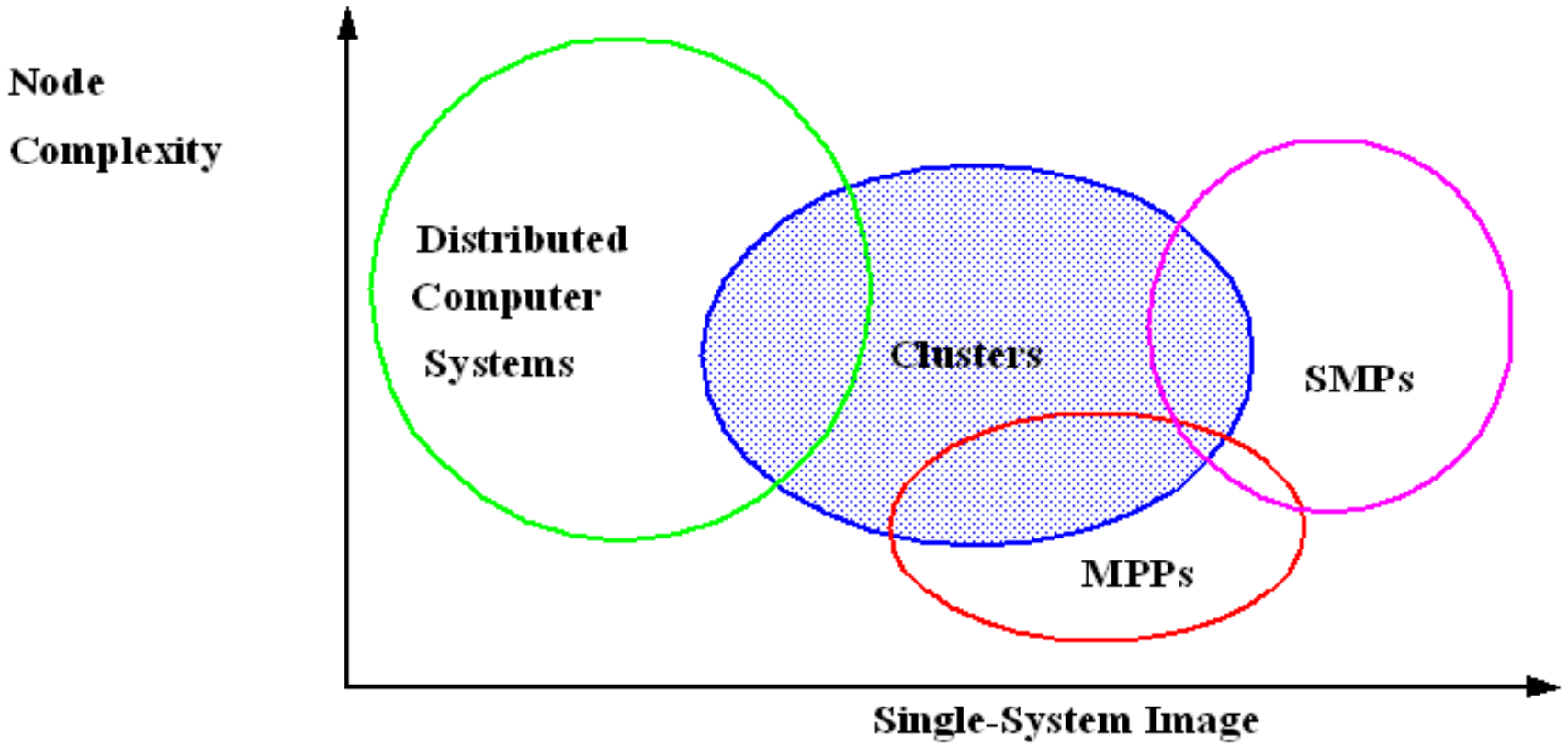

Typical architecture of a cluster of multiple computers.

# The five architectural concepts of clusters

- *Cluster Nodes*: Each node is a complete computer with cache, memory, disk, and some I/O adapters. Furthermore, a complete, standard operating system resides on each node. A node can have more than one processor, but only one copy of the OS image.
- Single-System Image: A cluster is a single computing resource. This is in contrast to a *distributed system* (e.g., a local computer network), where the nodes are used only as individual resources. SSI makes a cluster easier to use and manage.
- *Internode Connection*: The nodes of a cluster are usually connected through a commodity network, such as Ethernet, FDDI, Fiber-Channel, and ATM switch.
- *Enhanced Availability*: Clustering offers a cost-effective way to enhance the availability of a system, which is the percentage of time a system remains available to the user.
- *Better Performance*: A cluster should offer higher performance in a number of service areas. One area is to treat a cluster as a *superserver*. If each node of an *n*-node cluster can serve *m* clients, the cluster as a whole can serve *mn* clients simultaneously.

# Architectural Comparisons

**Clusters, SMPs (multicores), MPPs, and distributed systems are four overlapped architectural concepts.**



# Comparison of Cluster, MPP, Multicores, and Distributed Systems

| System Characteristic | MPP (supercomputers) | Multicore | Cluster | Distributed System |
|---|---|---|---|---|
| Number of nodes (N) | O(100) - O(1000,000) | O(10) or less | O(100) or less | O(10)-O(1000) |
| Node complexity | Fine or medium grain | Medium or coarse grain | Medium grain | Wide range |
| Internode communication | Message passing or shared variables (MPI and OpenMP) | Shared Memory (OpenMP) | Message Passing (MPI) | Shared files, RPC, Hadoop, message passing |
| Job scheduling | Single run queue at host | Single run queue | Multiple queues but coordinated | Independent multiple queues |
| SSI support | Partially | Always | Desired | No |
| Node OS copies and Type | N (microkernel) and 1 host OS (monolithic) | One (monolithic) | N (homogeneous desired) | N (heterogeneous) |
| Address space | Multiple (single if DSM) | Single | Multiple | Multiple |
| Internode security | Unnecessary | Unnecessary | Required if exposed | Required |
| Ownership | One organization | One organization | One or more organizations | Many organizations |

# Comparison of Cluster, MPP, SMP, and Distributed Systems

| System Characteristic | MPP | SMP | Cluster | Distributed System |
|---|---|---|---|---|
| Network protocol | Nonstandard | Nonstandard | Standard or nonstandard | Standard |
| System availability | Low to medium | Often low | Highly available or fault-tolerant | Medium |
| Performance metric | Throughput and turnaround time | Turnaround time | Throughput and turnaround time | Response time |
| SSI: Single-system image, RPC: Remote procedure call, DSM: Distributed shared memory | | | | |

# Benefits and Difficulties of Clusters

**The cluster concept brings many benefits as well as challenges. Among them the most important ones are**

- **usability**
- **availability**
- **scalability**
- **available utilization**
- **performance/cost ratio**

# Project Search (5 papers, combination of journal and conference papers)

**Where to search and how to search project papers**

1. Google
2. Google scholar
3. Citeceer
4. UTA Library
    1. IEEE Xplore Database
    2. Elsevier database, Science Direct
    3. ACM database
    4. Springer

**Rules**:

1. No paper older than 5 years
2. No dot com papers or links (actual pdf must be sent)
3. Must be computer science journal from IEEE, ACM, Elsevier or Springer
4. Must be IEEE or ACM conferences

# Journals and conference

## Journals

1. IEEE Transactions on parallel and distributed systems
2. IEEE Transactions on pattern analysis and machine intelligence
3. Journal of Supercomputing
4. Journal of parallel and distributed computing
5. Parallel computing

## Conferences

Supercomputing conference (HPC conference)

International conference on parallel processing

International parallel and distributed symposium