Principles of Parallel & Distributed Computing

Risala Tasin Khan

Professor

IIT, JU

Parallel Vs Distributed Computing

- The terms parallel computing and distributed computing are often used interchangeably, even though they mean slightly different things.
- The term parallel implies a tightly coupled system, whereas distributed refers to a wider class of system, including those that are tightly coupled.
- More precisely, the term parallel computing refers to a model in which the computation is divided among several processors sharing the same memory.
- The architecture of a parallel computing system is often characterized by the homogeneity of components: each processor is of the same type and it has the same capability as the others.
- The shared memory has a single address space, which is accessible to all the processors.
- Parallel programs are then broken down into several units of execution that can be allocated to different processors and can communicate with each other by means of the shared memory.

Parallel Vs Distributed Computing(Cont..)

- Originally we considered parallel systems only those architectures that featured multiple processors sharing the same physical memory and that were considered a single computer.
- Over time, these restrictions have been relaxed, and parallel systems now include all architectures that are based on the concept of shared memory, whether this is physically present or created with the support of libraries, specific hardware, and a highly efficient networking infrastructure.
- For example, a cluster of which the nodes are connected through an InfiniBand network and configured with a distributed shared memory system can be considered a parallel system

Parallel Vs Distributed Computing(Cont..)

- The term **distributed computing** encompasses any architecture or system that allows the computation to be broken down into units and executed concurrently on different computing elements, whether these are processors on different nodes, processors on the same computer, or cores within the same processor.
- Therefore, distributed computing includes a wider range of systems and applications than parallel computing and is often considered a more general term.
- The term **distributed** often implies that the locations of the computing elements are not the same and such elements might be heterogeneous in terms of hardware and software features.
- Classic examples of distributed computing systems are computing grids or Internet computing systems, which combine together the biggest variety of architectures, systems, and applications in the world.

Reason of the popularity of Parallel Processing in Distributed Environment

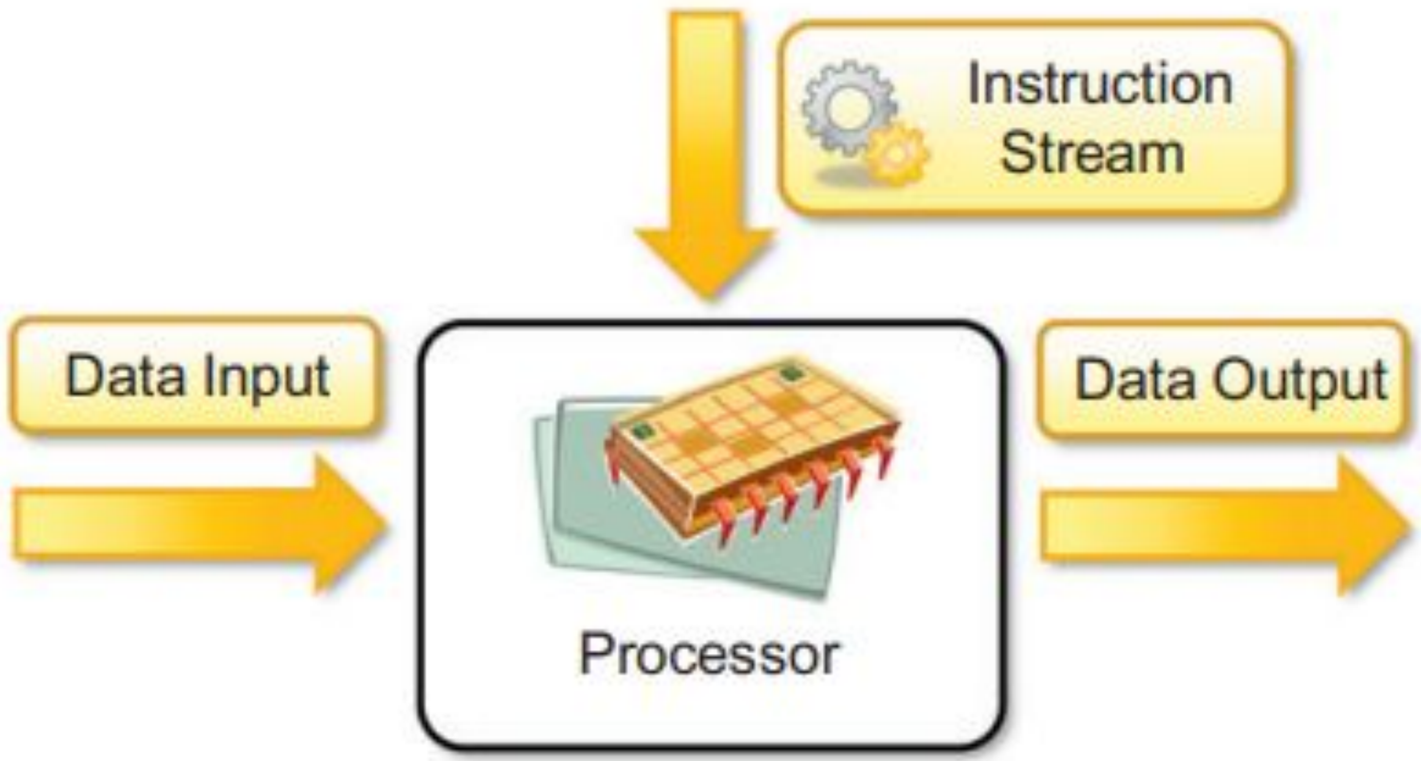
- The development of parallel processing is being influenced by many factors. The prominent among them include the following:
    - Computational requirements are ever increasing in the areas of both scientific and business computing. The technical computing problems (for example aerospace analysis, GIS, mechanical design etc.) require high-speed computational power.
    - Sequential architectures are reaching physical limitations as they are constrained by the speed of light and thermodynamics laws. The speed at which sequential CPUs can operate is reaching saturation point (no more vertical growth), and hence an alternative way to get high computational speed is to connect multiple CPUs (opportunity for horizontal growth).
    - The technology of parallel processing is mature and can be exploited commercially; there is already significant R&D work on development tools and environments.
    - Significant development in networking technology is paving the way for heterogeneous computing.

Hardware Architecture of Parallel Processing

- The core elements of parallel processing are CPUs.
- Based on the number of instruction and data streams that can be processed simultaneously, computing systems are classified into the following four categories:
  - Single-instruction, single-data (SISD) systems
  - Single-instruction, multiple-data (SIMD) systems
  - Multiple-instruction, single-data (MISD) systems
  - Multiple-instruction, multiple-data (MIMD) systems

Single Instruction Single Data System (SISD)

- An SISD computing system is a uniprocessor machine capable of executing a single instruction, which operates on a single data stream (see Figure 2.2).
- In SISD, machine instructions are processed sequentially; hence computers adopting this model are popularly called sequential computers.
- Most conventional computers are built using the SISD model.
- All the instructions and data to be processed have to be stored in primary memory.
- The speed of the processing element in the SISD model is limited by the rate at which the computer can transfer information internally.
- Dominant representative SISD systems are IBM PC, Macintosh, and workstations


Instruction Stream
Data Input
Processor
Data Output

Single Instruction Multiple Data System (SIMD)

- An SIMD computing system is a multiprocessor machine capable of executing the same instruction on all the CPUs but operating on different data streams (see Figure 2.3).
- Machines based on an SIMD model are well suited to scientific computing since they involve lots of vector and matrix operations.
- An example of Single Instruction Multiple Data (SIMD) is found in graphics processing units (GPUs). For instance, when processing a 3D image, a single instruction—such as a mathematical operation like addition or multiplication—can be applied simultaneously to multiple pixels in the image. This allows for parallel processing, significantly speeding up tasks like image rendering, video processing, or simulations.
- SIMD is also utilized in scientific computing, where large datasets require repetitive computations on multiple data points.

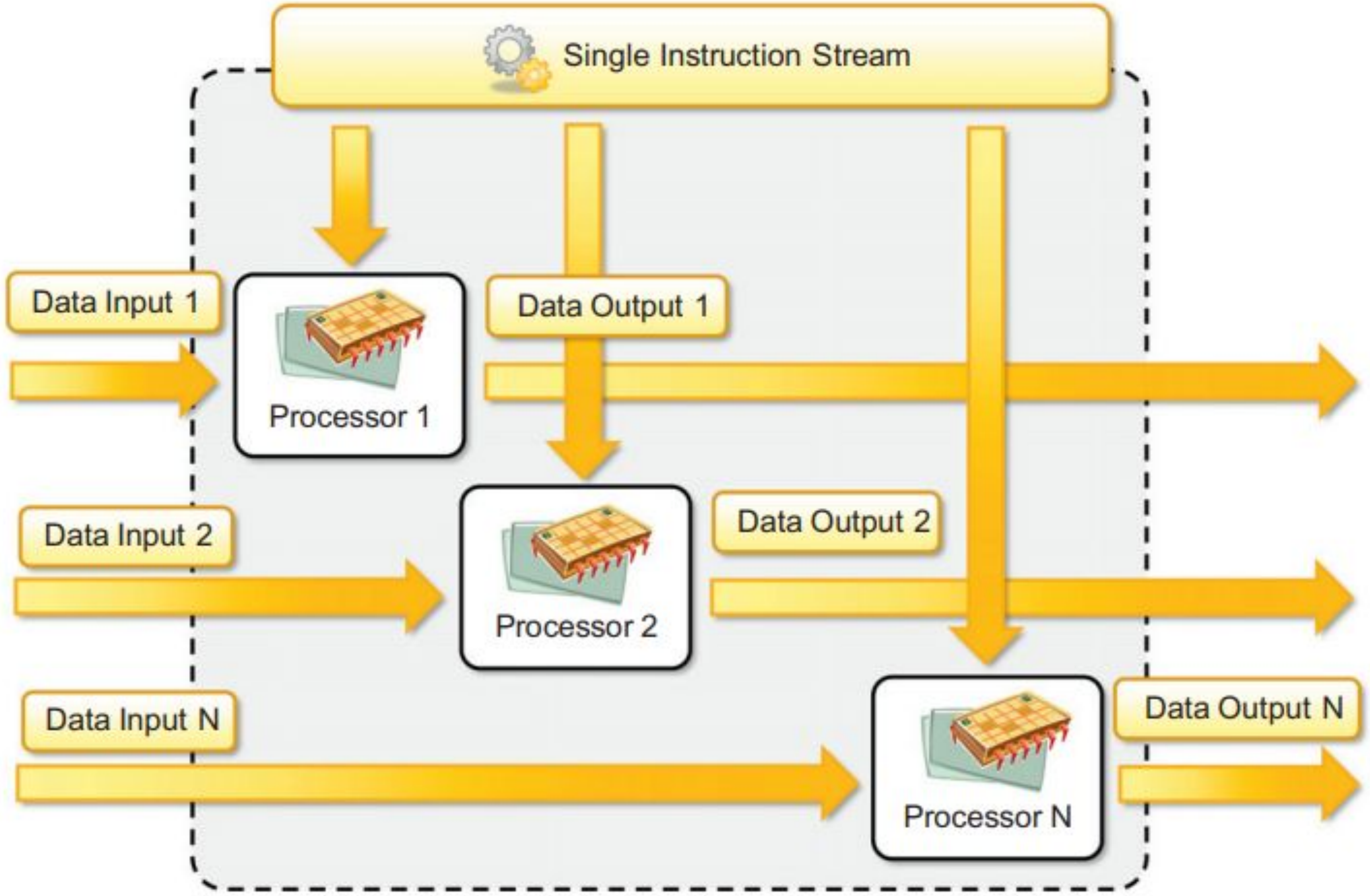


FIGURE 2.3

Single-instruction, multiple-data (SIMD) architecture.

Multiple Instruction Single Data System (MISD)

- An MISD computing system is a multiprocessor machine capable of executing different instructions on different PEs(Processing Elements) but all of them operating on the same data set (see Figure 2.4). For instance, statements such

$$y = \sin(x) + \cos(x) + \tan(x)$$

perform different operations on the same data set.

- Machines built using the MISD model are not useful in most of the applications; a few machines are built, but none of them are available commercially.
- They became more of an intellectual exercise than a practical configuration
- An example of an MISD (Multiple Instruction, Single Data) system would be a fault-tolerant computer system. In this type of system, multiple processors execute different instructions on the same data stream to ensure reliability and detect errors.
- For instance, in avionics or space exploration, where high reliability is critical, systems like the Space Shuttle's flight control computers operate in an MISD configuration to cross-check results and ensure accuracy.

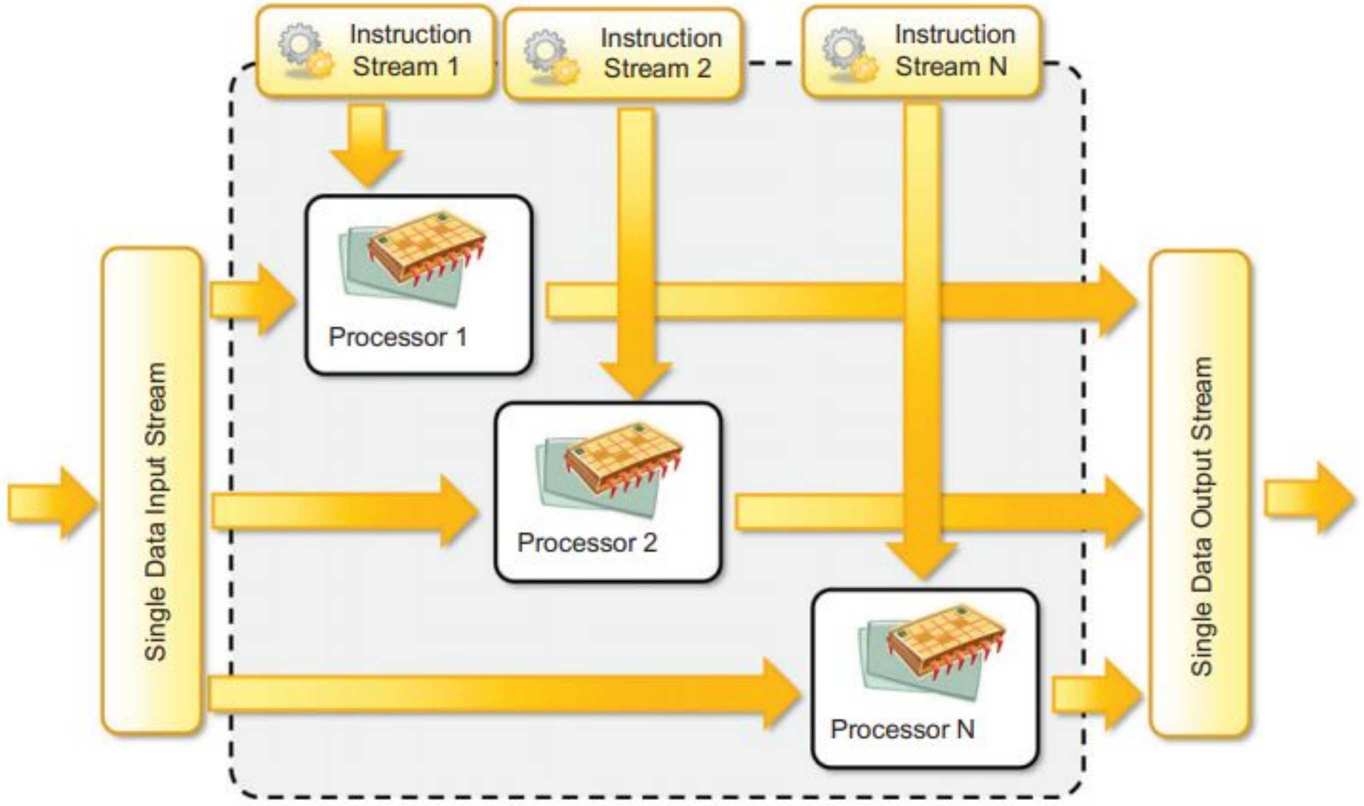


FIGURE 2.4

Multiple-instruction, single-data (MISD) architecture.

Multiple Instruction Multiple Data System (MIMD)

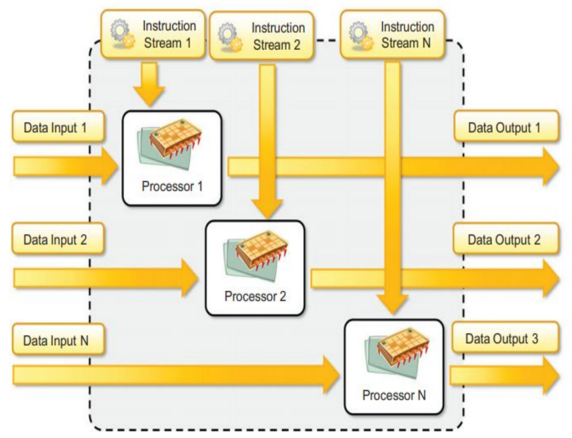


FIGURE 2.5

Multiple-instructions, multiple-data (MIMD) architecture.

- An MIMD computing system is a multiprocessor machine capable of executing multiple instructions on multiple data sets (see Figure 2.5).
- Each PE in the MIMD model has separate instruction and data streams; hence machines built using this model are well suited to any kind of application.
- Unlike SIMD and MISD machines, PEs in MIMD machines work asynchronously.
- MIMD machines are broadly categorized into **shared-memory MIMD** and **distributed-memory MIMD** based on the way PEs are coupled to the main memory

Shared Memory MIMD Machine

- In the shared memory MIMD model, all the PEs are connected to a single global memory and they all have access to it (see Figure 2.6).
- Systems based on this model are also called tightly coupled multiprocessor systems.
- The communication between PEs in this model takes place through the shared memory; modification of the data stored in the global memory by one PE is visible to all other PEs.
- Dominant representative shared memory MIMD systems are Silicon Graphics machines and Sun/IBM’s SMP (Symmetric Multi-Processing).

Distributed Memory MIMD Machines

- In the distributed memory MIMD model, all PEs have a local memory.
- Systems based on this model are also called loosely coupled multiprocessor systems.
- The communication between PEs in this model takes place through the interconnection network (the interprocess communication channel, or IPC).
- The network connecting PEs can be configured to tree, mesh, cube, and so on.
- Each PE operates asynchronously, and if communication/synchronization among tasks is necessary, they can do so by exchanging messages between them.

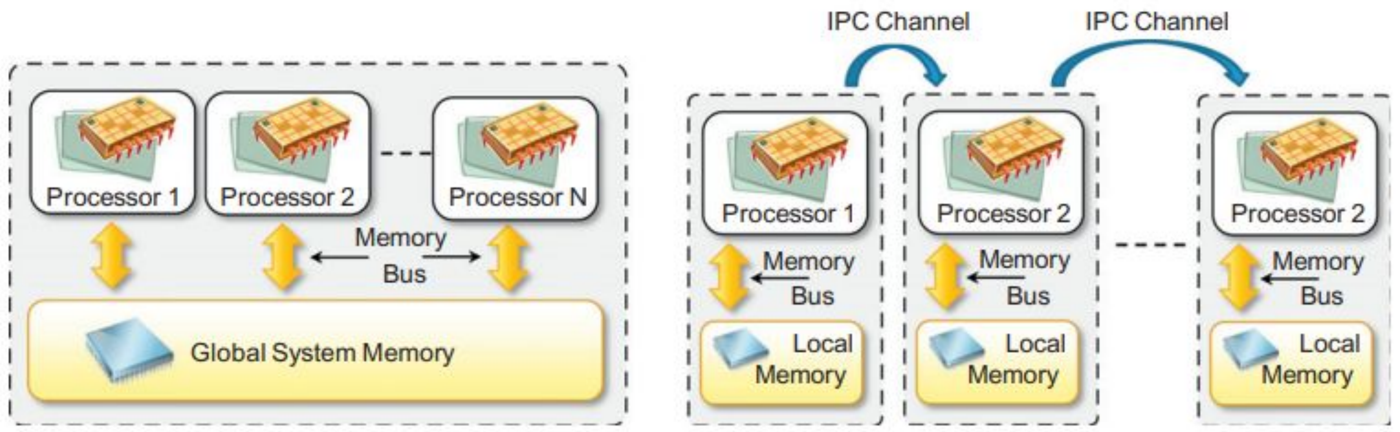


FIGURE 2.6

Shared (left) and distributed (right) memory MIMD architecture.

Approaches to parallel programming

- A sequential program is one that runs on a single processor and has a single line of control.
- To make many processors collectively work on a single program, the program must be divided into smaller independent chunks so that each processor can work on separate chunks of the problem.
- The program decomposed in this way is a parallel program.
- A wide variety of parallel programming approaches are available.
- The most prominent among them are the following:
- Data parallelism
- Process parallelism
- Farmer-and-worker model

These three models are all suitable for task-level parallelism.

Cont…

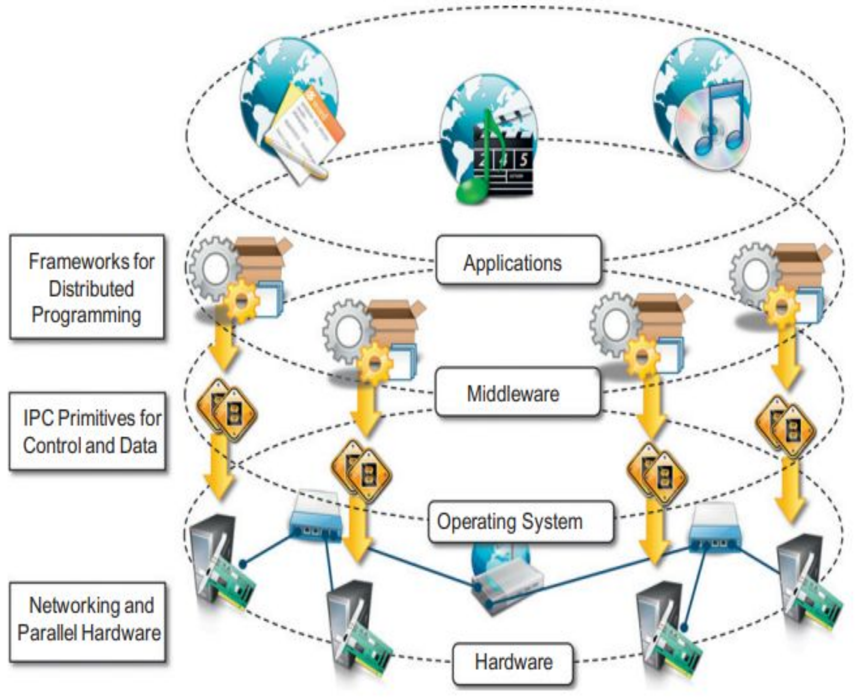
- In the case of data parallelism, the **divide-and-conquer technique** is used to split data into multiple sets, and each data set is processed on different PEs using the same instruction.
- This approach is highly suitable for processing on machines based on the SIMD model.
- In the case of process parallelism, a given operation has multiple (but distinct) activities that can be processed on multiple processors.
- In the case of the farmer-and-worker model, a job distribution approach is used:
  - one processor is configured as master and all other remaining PEs are designated as slaves;
  - the master assigns jobs to slave PEs and, on completion, they inform the master, which in turn collects results.
  - These approaches can be utilized in different levels of parallelism.

Laws of Caution

- **How much benefits an application or software can gain from parallelism?**
  - Parallelism is used to perform multiple activities together so that the system can increase its throughput or its speed.
  - But the relations that control the increment of speed are not linear.
  - For example, for a given *n* processors, the user expects speed to be increased by *n* times. This is an ideal situation, but it rarely happens because of the communication overhead
  - Here are two important guidelines to take into account:
    - Speed of computation is proportional to the square root of system cost; they never increase linearly. Therefore, the faster a system becomes, the more expensive it is to increase its speed
    - Speed by a parallel computer increases as the logarithm of the number of processors (i.e., $y = k\log(N)$).

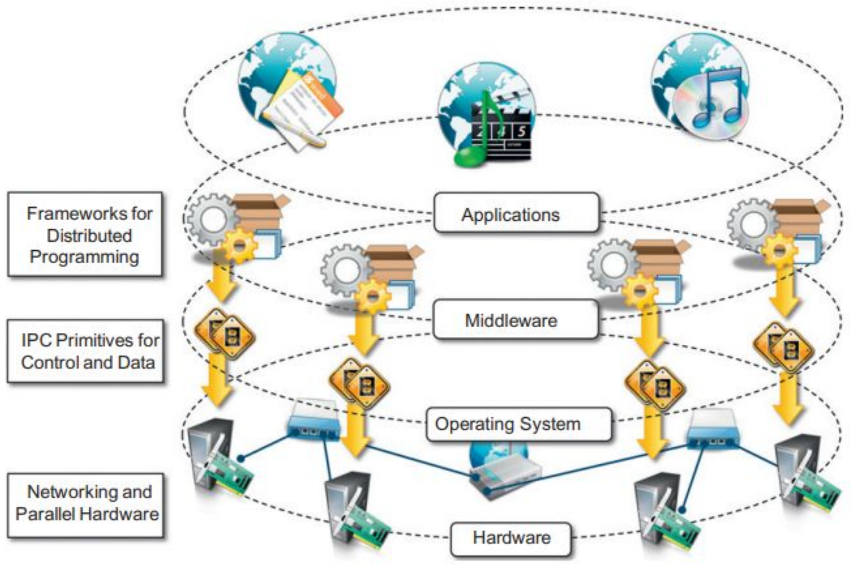
Components of a Distributed System

- A distributed system is the result of the interaction of several components that traverse the entire computing stack from hardware to software.
- It emerges from the collaboration of several elements that—by working together—give users the illusion of a single coherent system.
- Figure provides an overview of the different layers that are involved in providing the services of a distributed system.


Frameworks for Distributed Programming
IPC Primitives for Control and Data
Networking and Parallel Hardware
Applications
Middleware
Operating System
Hardware

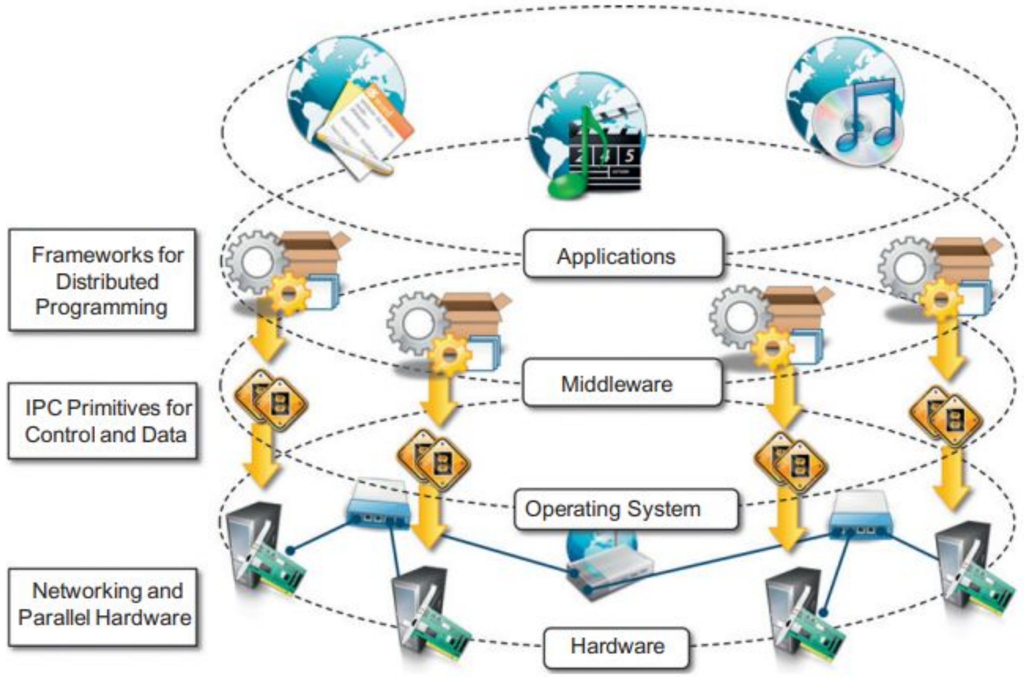
Cont….

- At the very bottom layer, computer and network hardware constitute the physical infrastructure;
- these components are directly managed by the operating system, which provides the basic services for inter-process communication (IPC), process scheduling and management, and resource management in terms of file system and local devices.
- Taken together these two layers become the platform on top of which specialized software is deployed to turn a set of networked computers into a distributed system


Frameworks for Distributed Programming
Applications
Middleware
IPC Primitives for Control and Data
Operating System
Networking and Parallel Hardware
Hardware

Cont…

- The middleware layer provides services to build a uniform environment for the development and deployment of distributed applications.
- This layer supports the programming paradigms for distributed systems.
- By relying on the services offered by the operating system, the middleware develops its own protocols, data formats, and programming language or frameworks for the development of distributed applications.
- All of them constitute a uniform interface to distributed application developers that is completely independent from the underlying operating system and hides all the heterogeneities of the bottom layers


Frameworks for Distributed Programming
IPC Primitives for Control and Data
Networking and Parallel Hardware
Applications
Middleware
Operating System
Hardware

Cont…

- The top of the distributed system stack is represented by the applications and services designed and developed to use the middleware.
- These can serve several purposes and often expose their features in the form of graphical user interfaces (GUIs) accessible locally or through the Internet via a Web browser

Architectural styles for distributed computing

- Although a distributed system comprises the interaction of several layers, the **middleware layer** is the one that enables distributed computing, because it provides a coherent and uniform runtime environment for applications.
- There are many different ways to organize the components of distributed systems.
- The interactions among these components and their responsibilities defines the architecture
- An architectural model of a distributed system is concerned with the placement of its parts and the relationships between them.
- Examples include:
    - Client-Server model
    - Peer-to-Peer model
-

Cont…

- The client-server model can be modified by:
  - The partition of data or replication at cooperative servers
  - The caching of data by proxy servers and clients
  - The use of mobile code and mobile agents

Cont…

Major concerns in the architectural models are to make the system:

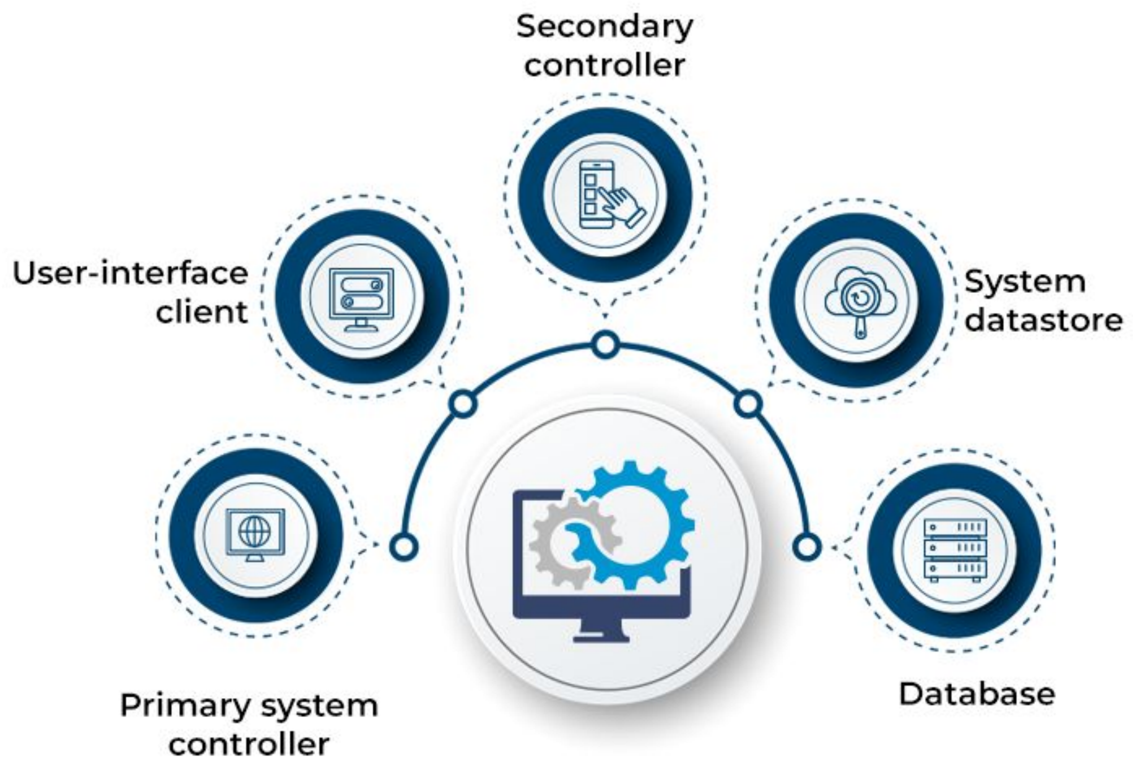
- ❖ Reliable
- ❖ Manageable
- ❖ Adaptable
- ❖ Cost-effective

- We organize the architectural styles into two major classes:
- **Software architectural styles**
- **System architectural styles**

Component and Connectors

- A **component** represents a unit of software that encapsulates a function or a feature of the system.
    - Examples of components can be programs, objects, processes, pipes, and filters.
- A **connector** is a communication mechanism that allows cooperation and coordination among components.
- Connectors are not encapsulated in a single entity, but they are implemented in a distributed manner over many system components

KEY COMPONENTS OF A DISTRIBUTED SYSTEM



Software architectural styles

- Software architectural styles are based on the logical arrangement of software components.
- They are helpful because they provide an intuitive view of the whole system, despite its physical deployment.
- According to Garlan and Shaw [105], architectural styles are classified as shown in Table 2.2.

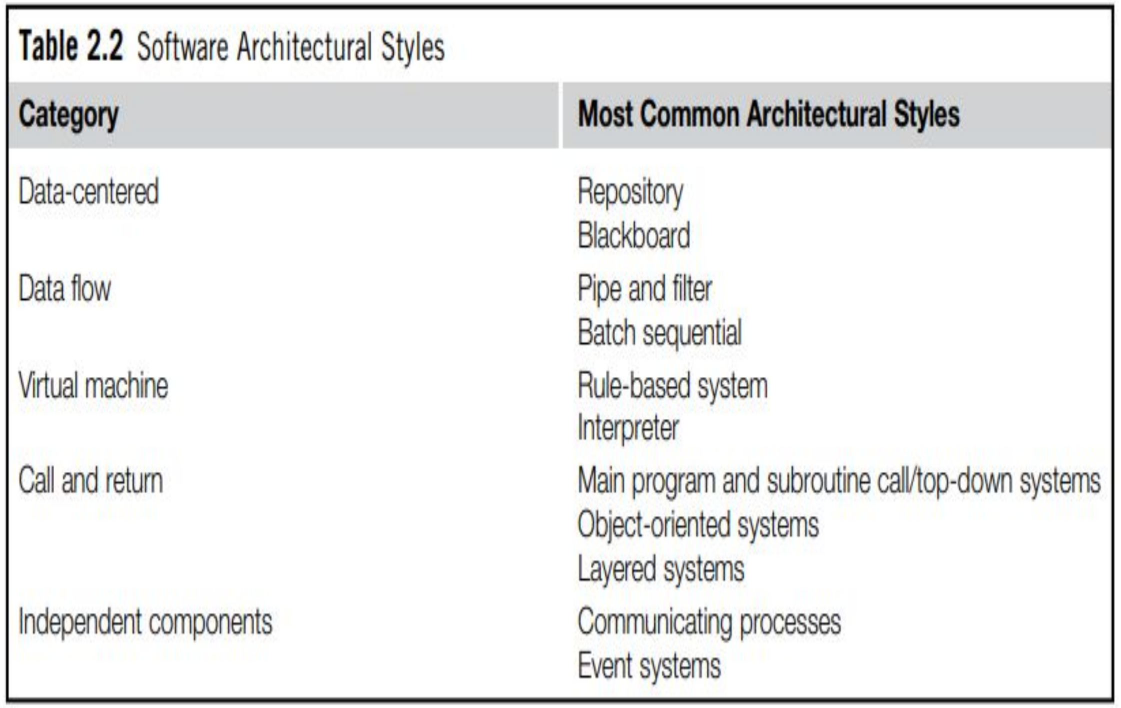
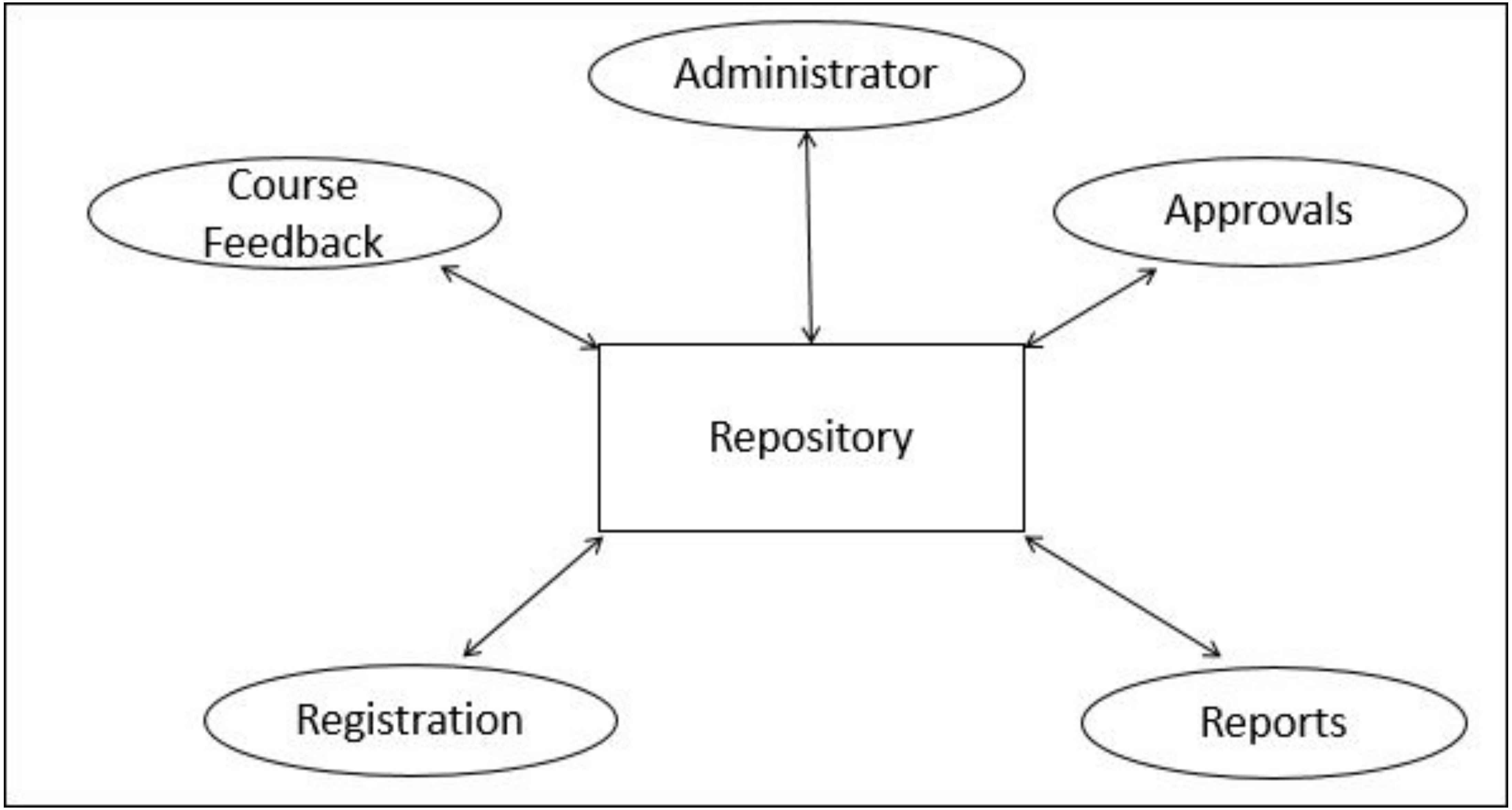


Table 2.2 Software Architectural Styles

Category	Most Common Architectural Styles
Data-centered	Repository Blackboard
Data flow	Pipe and filter Batch sequential
Virtual machine	Rule-based system Interpreter
Call and return	Main program and subroutine call/top-down systems Object-oriented systems Layered systems
Independent components	Communicating processes Event systems

Data Centred Architectures

- These architectures identify the data as the fundamental element of the software system, and access to shared data is the core characteristic of the data-centred architectures.
- Therefore, especially within the context of distributed and parallel computing systems, integrity of data is the overall goal for such systems.
- The **repository architectural style** is the most relevant reference model in this category.
- It is characterized by two main components:
  - **the central data structure**, which represents the current state of the system,
  - and **a collection of independent components**, which operate on the central data.
- The ways in which the independent components interact with the central data structure can be very heterogeneous.


Administrator
Course Feedback
Approvals
Repository
Registration
Reports

Data-flow architectures

- Data Flow Architecture is transformed input data by a series of computational or manipulative components into output data.
- With respect to the data-centred styles, in which the access to data is the core feature, data-flow styles explicitly incorporate the pattern of data flow, since their design is determined by an orderly motion of data from component to component, which is the form of communication between them.

Data-flow architectures(Cont..)

- **Batch Sequential Style:**
  - The batch sequential style is characterized by an ordered sequence of separate programs executing one after the other.
  - These programs are chained together by providing as input for the next program the output generated by the last program after its completion, which is most likely in the form of a file.
  - This design was very popular in the mainframe era of computing and still finds applications today.
  - For example, many distributed applications for scientific computing are defined by jobs expressed as sequences of programs that, for example, pre-filter, analyze, and post-process data.
  - The main disadvantage of batch sequential architecture is that, it does not provide concurrency and interactive interface. It provides high latency and low throughput.

Data-flow architectures(Cont..)

- **Pipe-and-Filter Style:**
  - The pipe-and-filter style is a variation of the previous style for expressing the activity of a software system as sequence of data transformations.
  - Each component of the processing chain is called a **filter**, and Pipe is a connector which passes the data from one filter to the next.
  - Pipe is a directional stream of data implemented by a data buffer to store all data, until the next filter has time to process it.
  - As soon as one filter produces a consumable amount of data, the next filter can start its processing.
  - Filters generally do not have state, know the identity of neither the previous nor the next filter, and they are connected with in-memory data structures such as first-in/first-out (FIFO) buffers or other structures.
  - This particular sequencing is called **pipelining** and introduces concurrency in the execution of the filters.

What is meant by Pipe?

- Pipe is a connector which passes the data from one filter to the next.
- Pipe is a directional stream of data implemented by a data buffer to store all data, until the next filter has time to process it.
- It transfers the data from one data source to one data sink.
- Pipes are the stateless data stream.
- The figure shows the pipe-filter sequence.
- All filters are the processes that run at the same time, it means that they can run as different threads, coroutines or be located on different machines entirely.
- Filter reads the data from its input pipes and performs its function on this data and places the result on all output pipes.
- If there is insufficient data in the input pipes, the filter simply waits.

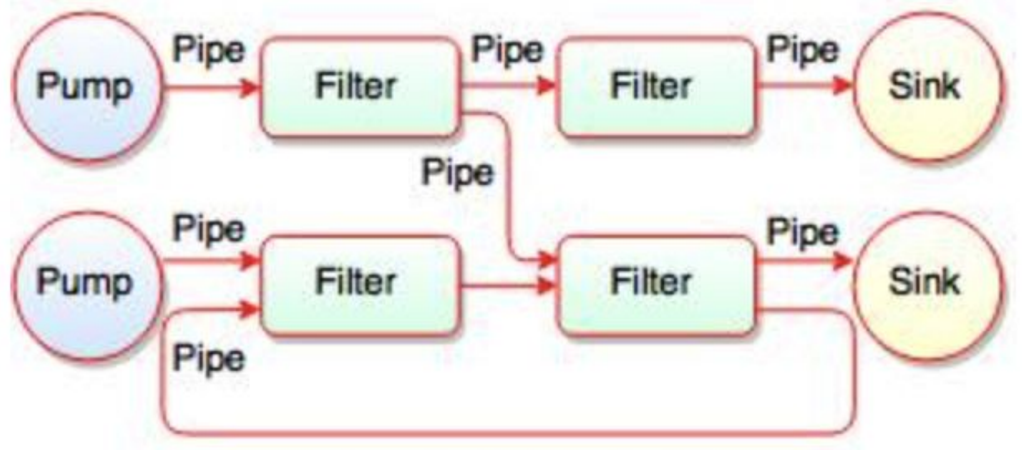


Fig. Pipes and Filters

Pros and Cons of Pipe and Filter Model

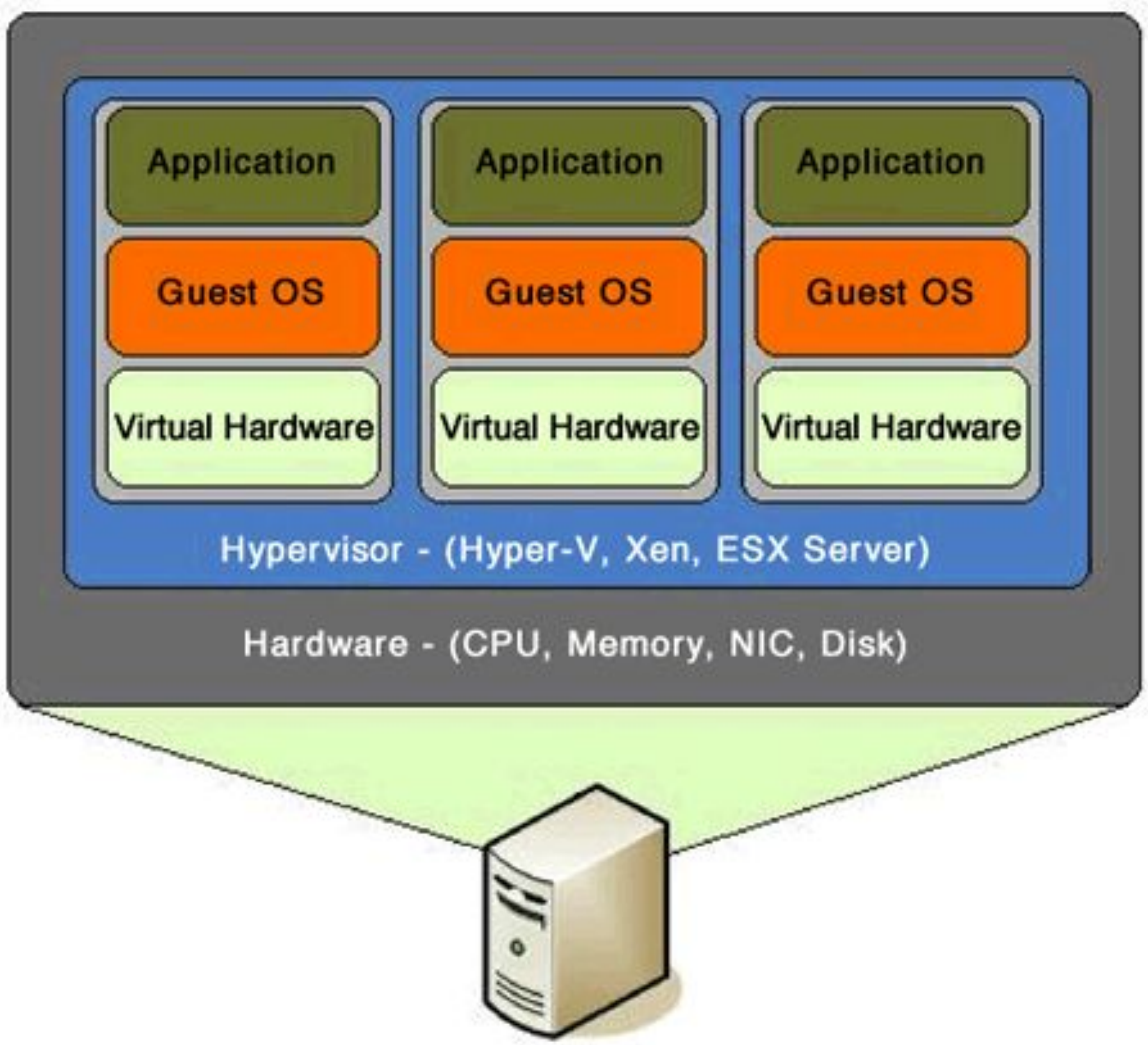
- **Advantages of Pipes and Filters**
  - Pipe-filter provides concurrency and high throughput for excessive data processing.
  - It simplifies the system maintenance and provides reusability.
- **Disadvantages of Pipe and Filter**
  - Pipe and Filter are not suitable for dynamic interactions.
  - It is difficult to configure Pipe-filter architecture dynamically.

Data-flow architecture (Cont…)

- **Process Control Architecture**
  - Process Control Architecture is a type of Data Flow Architecture, where data is neither batch sequential nor pipe stream.
  - In process control architecture, the flow of data comes from a set of variables which controls the execution of process.
  - This architecture decomposes the entire system into subsystems or modules and connects them.
  - Process control architecture is suitable in the embedded system software design, where the system is manipulated by process control variable data
  - This architecture is applicable for building temperature control system.

Virtual machine architectures

- The virtual machine architectural styles is characterized by the presence of an abstract execution environment (generally referred as a virtual machine) that simulates features that are not available in the hardware or software.
- Applications and systems are implemented on top of this layer and become portable over different hardware and software environments as long as there is an implementation of the virtual machine they interface with.
- The general interaction flow for systems implementing this pattern is the following:
    - the program (or the application) defines its operations and state in an abstract format, which is interpreted by the virtual machine engine.
    - The interpretation of a program constitutes its execution.
    - It is quite common in this scenario that the engine maintains an internal representation of the program state.
- Very popular examples within this category are rule-based systems, interpreters, and command-language processors.


Application
Application
Application
Guest OS
Guest OS
Guest OS
Virtual Hardware
Virtual Hardware
Virtual Hardware
Hypervisor - (Hyper-V, Xen, ESX Server)
Hardware - (CPU, Memory, NIC, Disk)

Virtual machine architectures (Cont…)

- **Rule-Based Style:**
- Rule-based virtual machine architecture operates by using predefined rules to manage and execute tasks within the virtual machine environment. These rules dictate how the virtual machine interprets and processes instructions, ensuring consistency and efficiency.
- For example, in distributed virtual machines, rule-based services can intercept application code and modify it dynamically to provide additional functionality. This approach reduces resource demands on client systems and centralizes administration, enhancing security and manageability.

- Rule-based systems are very popular in the field of artificial intelligence.
- Practical applications can be found in the field of process control, where rule-based systems are used to monitor the status of physical devices by being fed from the sensory data collected and processed by PLCs1 and by activating alarms when specific conditions on the sensory data apply.
- Another interesting use of rule-based systems can be found in the networking domain: network intrusion detection systems (NIDS) often rely on a set of rules to identify abnormal behaviors connected to possible intrusions in computing systems.

Virtual machine architectures (Cont…)

- **Interpreter Style:**
- The core feature of the interpreter style is the presence of an engine that is used to interpret a pseudo-program expressed in a format acceptable for the interpreter.
- Systems modelled according to this style exhibit four main components: <u>the interpretation engine</u> that executes the core activity of this style, <u>an internal memory</u> that contains the pseudo-code to be interpreted, <u>a representation of the current state of the engine</u>, and <u>a representation of the current state of the program being executed</u>.
- This model is quite useful in designing virtual machines for high-level programming (Java, C#) and scripting languages (Awk, PERL, and so on).
- Within this scenario the virtual machine closes the gap between the end-user abstractions and the software/hardware environment

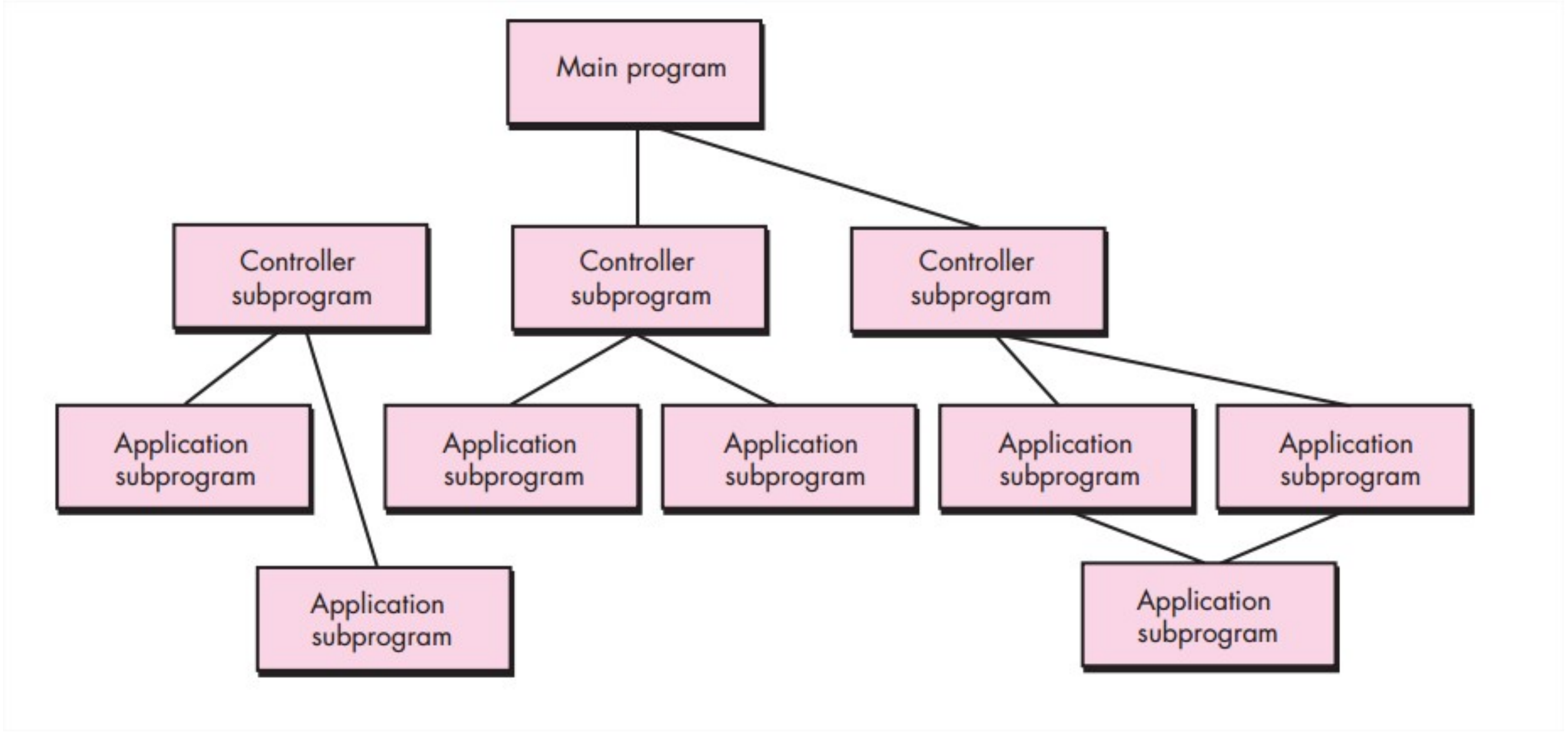
Call & return architectures

- This category identifies all systems that are organised into components mostly connected together by method calls.
- The activity of systems designed in this way is characterized by a chain of method calls whose overall execution identify the execution of one or more operations.
- The internal organization of components and their connections may vary.
- Nonetheless, it is possible to identify three major subcategories, which differentiate by the way the system is structured and how methods are invoked:
- top-down style,
- object-oriented style,

Call & return architectures(Cont..)

- **Top-Down Style:**
- This architectural style is quite representative of systems developed with imperative programming, which leads to a divide-and-conquer approach to problem resolution.
- Systems developed according to this style are composed of one large main program that accomplishes its tasks by invoking subprograms or procedures.
- The components in this style are procedures and subprograms, and connections are method calls or invocation.
- The calling program passes information with parameters and receives data from return values or parameters.
- Method calls can also extend beyond the boundary of a single process by leveraging techniques for remote method invocation, such as remote procedure call (RPC) and all its descendants.
- The overall structure of the program execution at any point in time is characterized by a tree, the root of which constitutes the main function of the principal program.
- This architectural style is quite intuitive from a design point of view but hard to maintain and manage in large systems

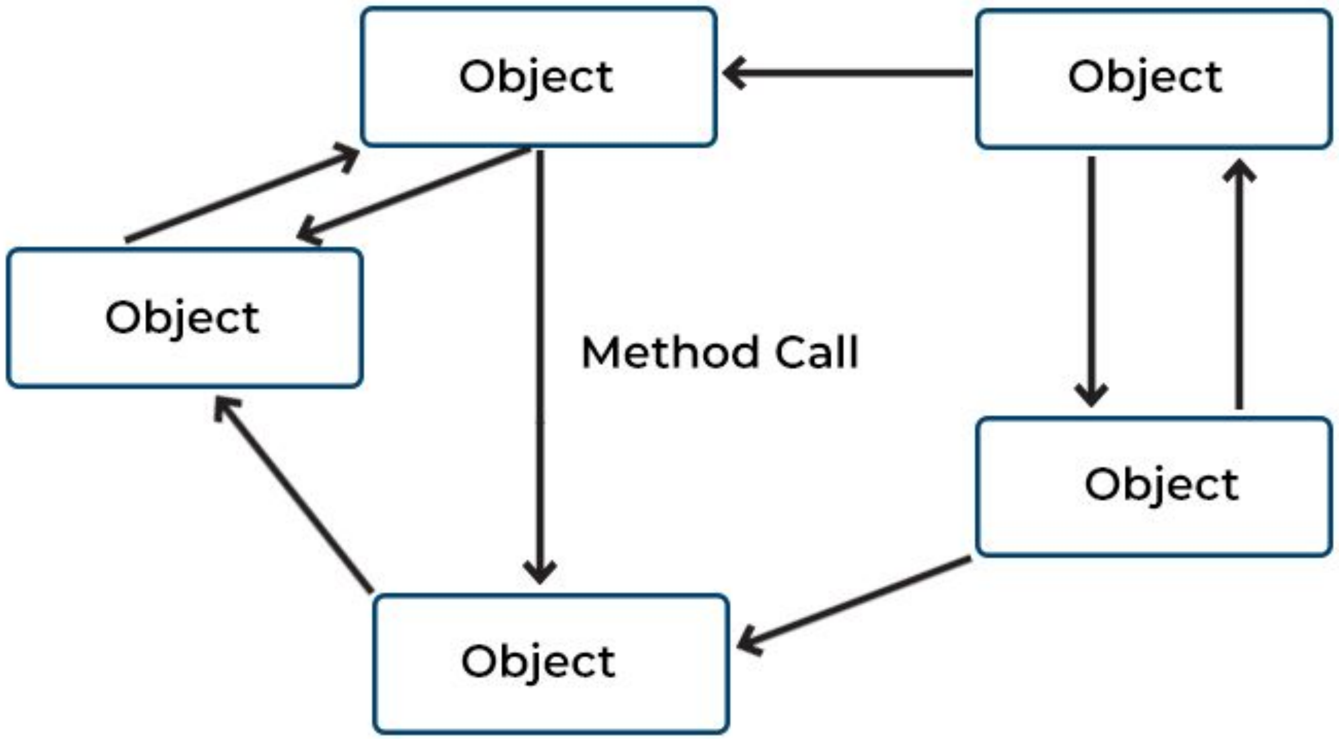
Top-Down Style



Call & return architectures(Cont…)

- **Object-Oriented Style:**
- This architectural style encompasses a wide range of systems that have been designed and implemented by using the concept of the abstractions of object-oriented programming (OOP)
- Systems are specified in terms of classes and implemented in terms of objects.
- One of the main advantages over the top-down style is that there is a coupling between data and operations used to manipulate them.
- Object instances become responsible for hiding their internal state representation and for protecting its integrity while providing operations to other components.
- This leads to a better decomposition process and more manageable systems.
- Disadvantages of this style are mainly two:
  - each object needs to know the identity of an object if it wants to invoke operations on it,
  - and shared objects need to be carefully designed in order to ensure the consistency of their state.

OBJECT-BASED ARCHITECTURE



Architectural styles based on independent components

- This class of architectural style designs a system in terms of independent components that have their own life cycles, which interact with each other to perform their activities.
- There are two major categories within this class—
- communicating processes and
- event systems

—which differentiate in the way the interaction among components is managed

Architectural styles based on independent components(Cont…)

- **Communicating Processes:**
  - In this architectural style, components are represented by independent processes that use IPC facilities for coordination management.
  - This is an abstraction that is quite suitable for modelling distributed systems that, being distributed over a network of computing nodes, are necessarily composed of several concurrent processes.
  - Each of the processes provides other processes with services and can use the services exposed by the other processes.
  - The conceptual organization of these processes and the way in which the communication happens vary according to the specific model used, either peer-to-peer or client/server.

Architectural styles based on independent components(Cont..)

- **Event Systems:**
  - In this architectural style, the components of the system are loosely coupled and connected.
  - In event-based architecture, the entire communication is through events. When an event occurs, the system gets the notification.
  - This means that anyone who receives this event will also be notified and has access to information. Sometimes, these events are data, and at other times they are URLs to resources.
  - As such, the receiver can process what information they receive and act accordingly.
  - Each component publishes (or announces) a collection of events with which other components can register.
  - In general, other components provide a callback that will be executed when the event is activated.
  - During the activity of a component, a specific runtime condition can activate one of the exposed events, thus triggering the execution of the callbacks registered with it.
  - Event activation may be accompanied by contextual information that can be used in the callback to handle the event.
  - This information can be passed as an argument to the callback or by using some shared repository between components.
  - Event-based systems have become quite popular, and support for their implementation is provided either at the API level or the programming language level.