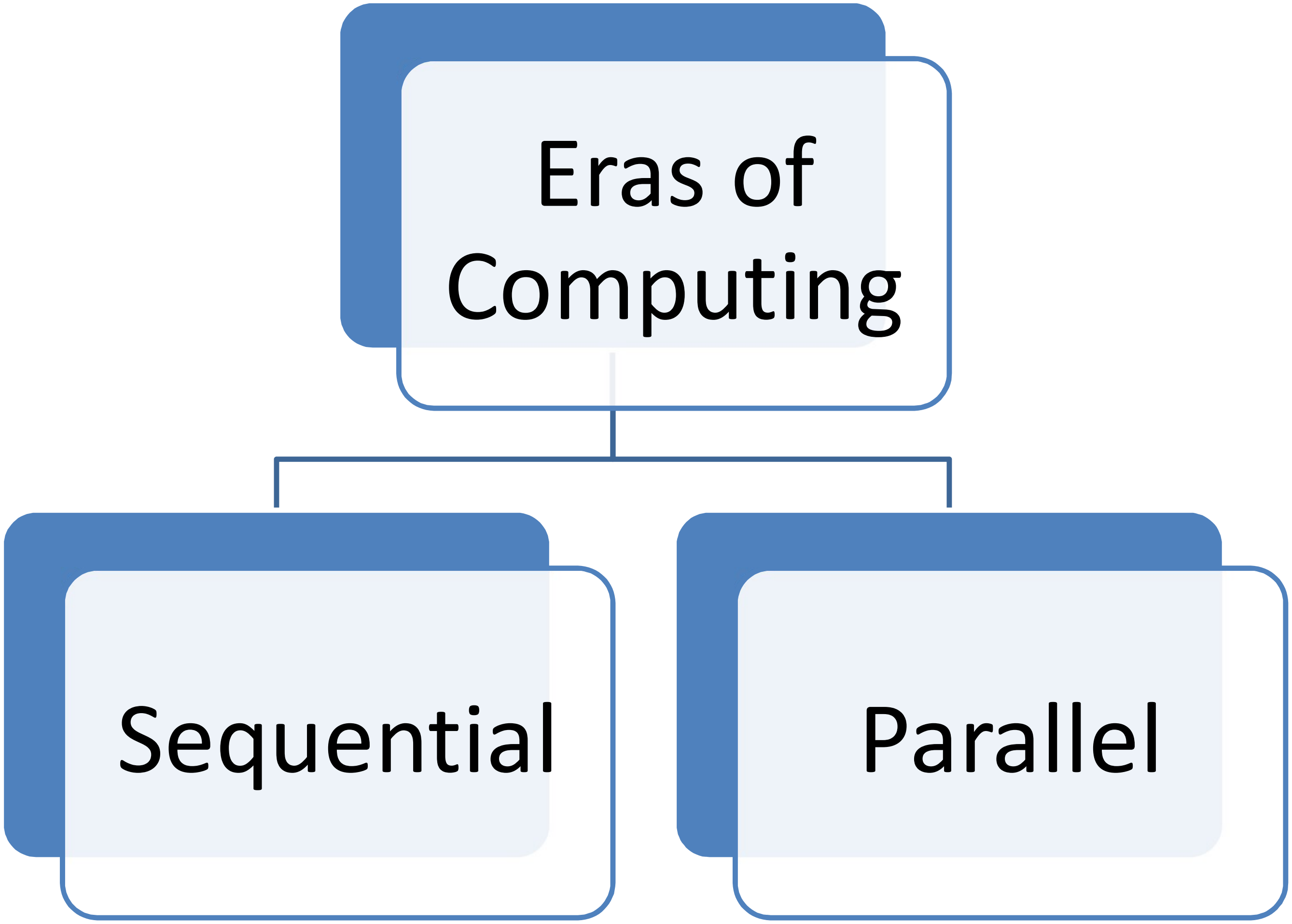
Eras of Computing

Types of computing models


Eras of Computing

Sequential

Parallel

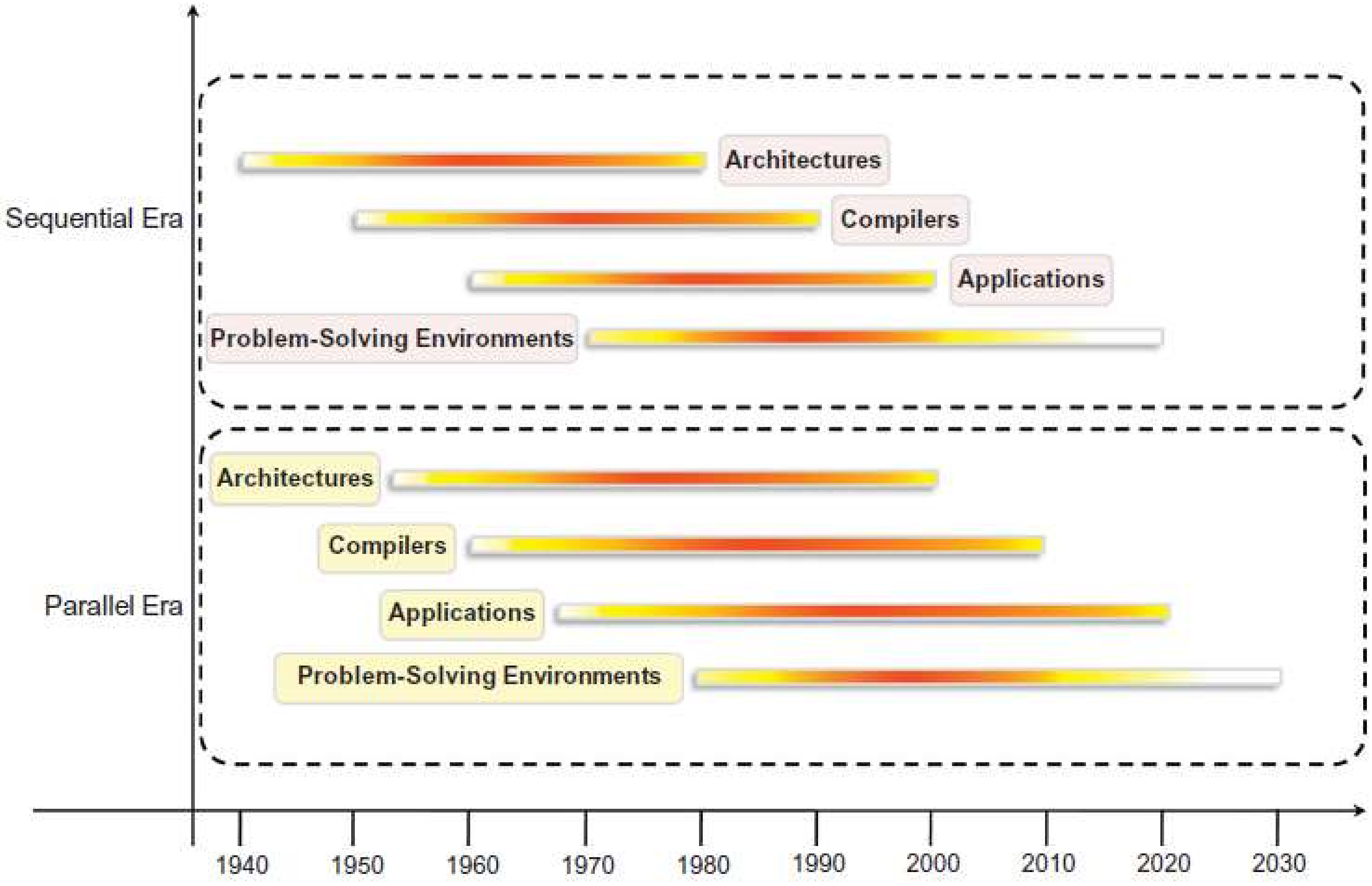


FIGURE 2.1

Eras of computing, 1940s–2030s.

Taken from Mastering cloud computing, Buyya R.

Parallel Vs. Distributed

- The term parallel implies a **tightly coupled** system.
- The term distributed refers to a wider class of system, including those that are tightly coupled.

Parallel Computing

- Computation is divided among several processors sharing the same memory.
- Often characterized by the homogeneity of components: Each processor is of the same type and it has the same capability as the others.
- 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 Computing

- Originally, multiple processor sharing the same physical memory in single computer considered parallel systems
- Now it 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.

Distributed Computing

- The term distributed computing encompasses any architecture or system that allows the computation to be broken down in to 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.
- Distributed computing includes a wider range of systems and applications than parallel computing and is often considered a more general term.

Distributed Computing

- It is not a rule, 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 .

Element Parallel Computing

- Processor chips are reaching their physical limits.
- Processing speed is constrained by the speed of light.
- Density of transistor packaged in processor is constrained by thermodynamic limitations.
- A viable solution is to connect multiple processors.

What is parallel processing?

- Processing of multiple tasks simultaneously on multiple processors is called parallel processing.
- The parallel program consists of multiple active processes (tasks) simultaneously solving a given problem.
- A given task is divided into multiple sub tasks using a divide-and-conquer technique, and each subtask is processed on a different central processing unit(CPU).
- Programming on a multiprocessor system using the divide-and-conquer technique is called parallel programming.

What is parallel processing?

- Applications require more computing power than a traditional sequential computer can offer.
- Parallel processing provides a cost-effective solution to this problem by increasing the number of CPUs in a computer and by adding an efficient communication system between them.
- The workload can then be shared between different processors.
- This setup results in higher computing power and performance than a single-processor system offers.

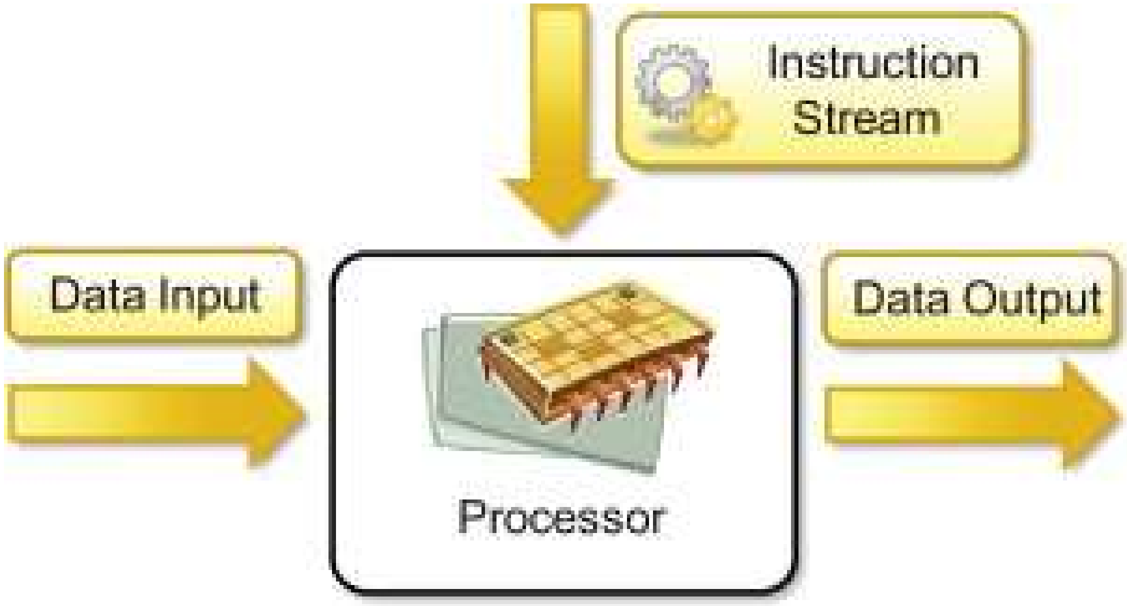
What is parallel processing?

- Influencing parameter for development of parallel processing:
  - Computational requirement
  - Sequential architecture reaching physical limitations
  - Improvement in hardware
  - Vector processing
  - Field is mature enough
  - Significant development in networking technology

Hardware Architecture for Parallel Processing

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

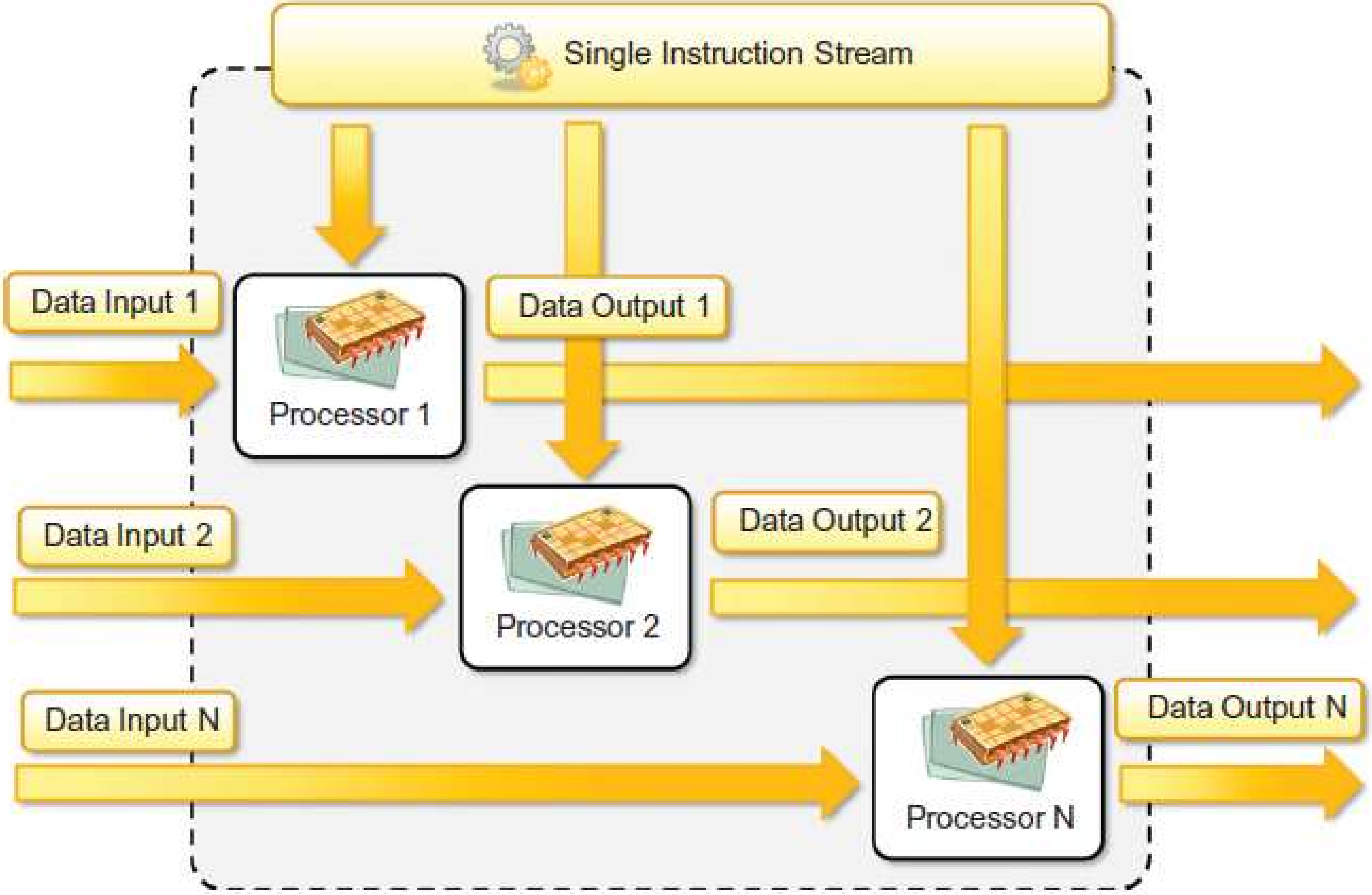
Single-instruction, single-data (SISD) systems



Single-instruction, single-data (SISD) systems

- Uniprocessor machine.
- Capable of executing single instruction operates on single data stream.
- Instruction processed sequentially.
- Hence called sequential computers.
- All instructions and data is to be processed have to be stored in primary memory.
- Speed is limited by transfer rate of information internally.
- i.e. IBM PC, Macintosh and Workstation.

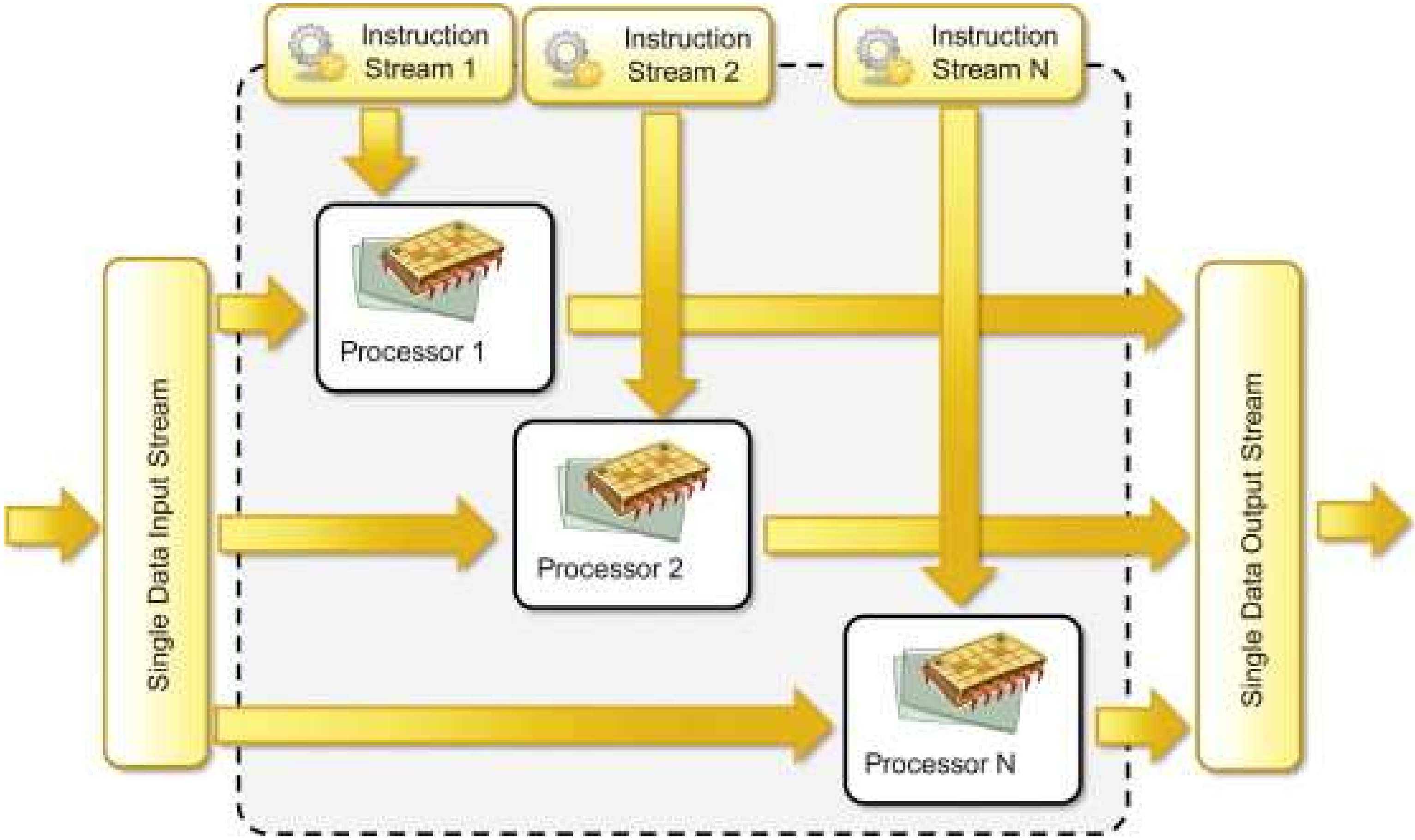
Single-instruction, multiple-data (SIMD) systems



Single-instruction, multiple-data (SIMD) systems

- An SIMD computing system is a multiprocessor machine capable of executing the same instruction on all the CPUs but operating on different data streams
- Machines based on an SIMD model are well suited to scientific computing since they involve lots of vector and matrix operations.
- For instance, statements such as $C_i = A_i * B_i$
- i.e. Cray's vector processing machine and Thinking Machines 'cm.

Multiple-instruction, single-data (MISD) systems



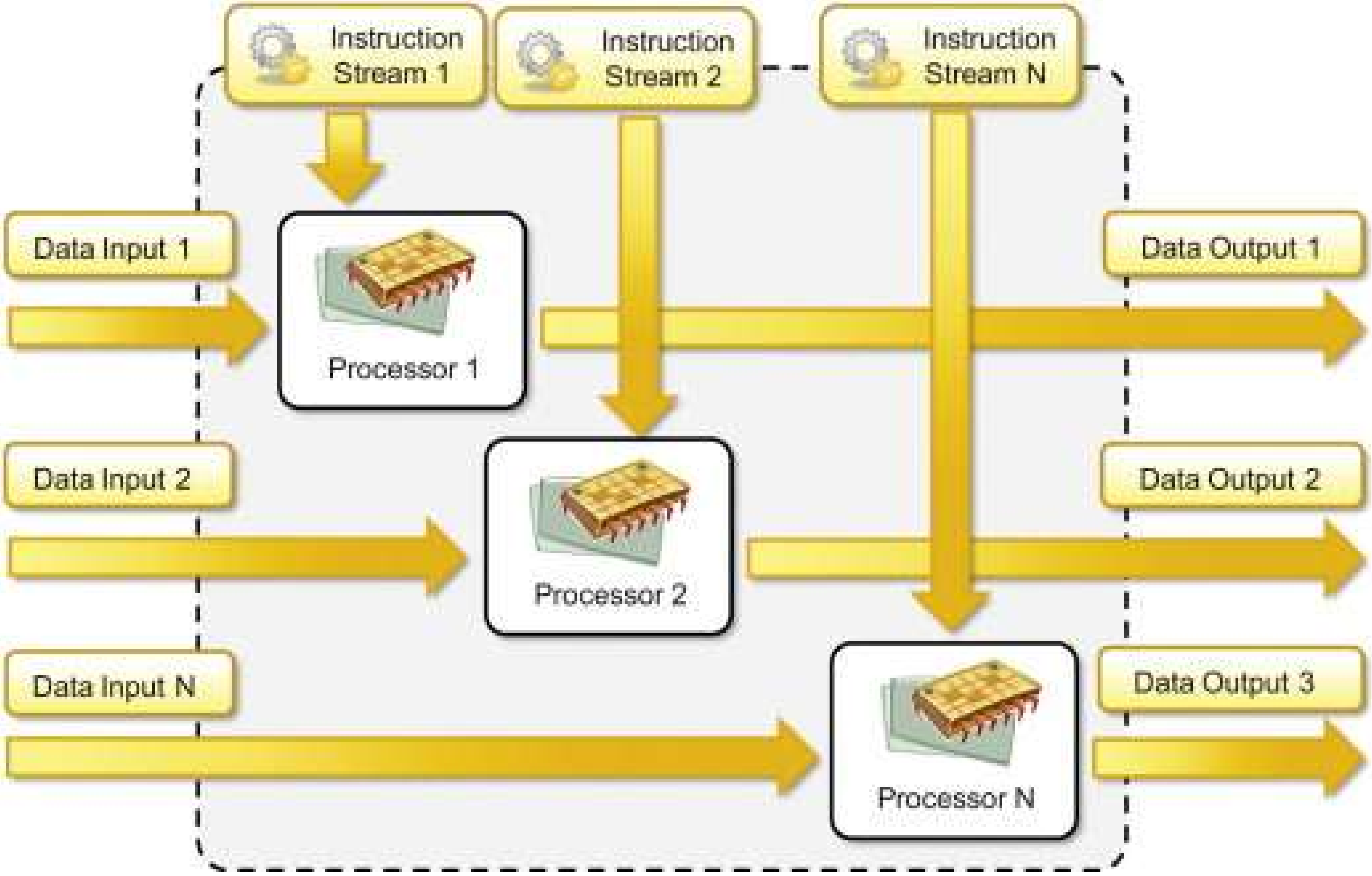
Multiple-instruction, single-data (MISD) systems

- An MISD computing system is a multiprocessor machine capable of executing different instructions on different PEs but all of them operating on the same data set.
- For instance, statements such as

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

- Machines built using the MISD model are not useful in most of the applications.

Multiple-instruction, multiple-data (MIMD) systems



Multiple-instruction, multiple-data (MIMD) systems

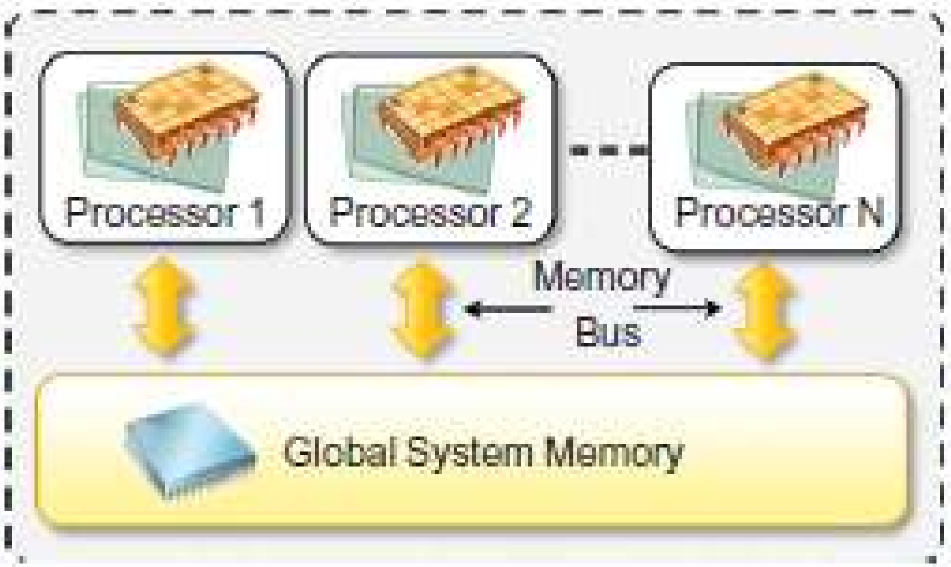
- An MIMD computing system is a multiprocessor machine capable of executing multiple instructions on multiple data sets.
- Each PE in the MIMD model has separate instruction and data streams; hence machines built using this model are well suited to any kind of application.
- Unlike SIMD and MISD machines, PEs in MIMD machines work asynchronously.

Multiple-instruction, multiple-data (MIMD) systems

- MIMD machines are broadly categorized based on the way PEs are coupled to the main memory:
  - Shared-memory MIMD
  - Distributed-memory MIMD

Shared Memory MIMD

- All PEs connected to single global memory – called tightly coupled multiprocessor systems.
- i.e. Silicon Graphics and Sun/IBM's SMP (Symmetric Multi-processing).

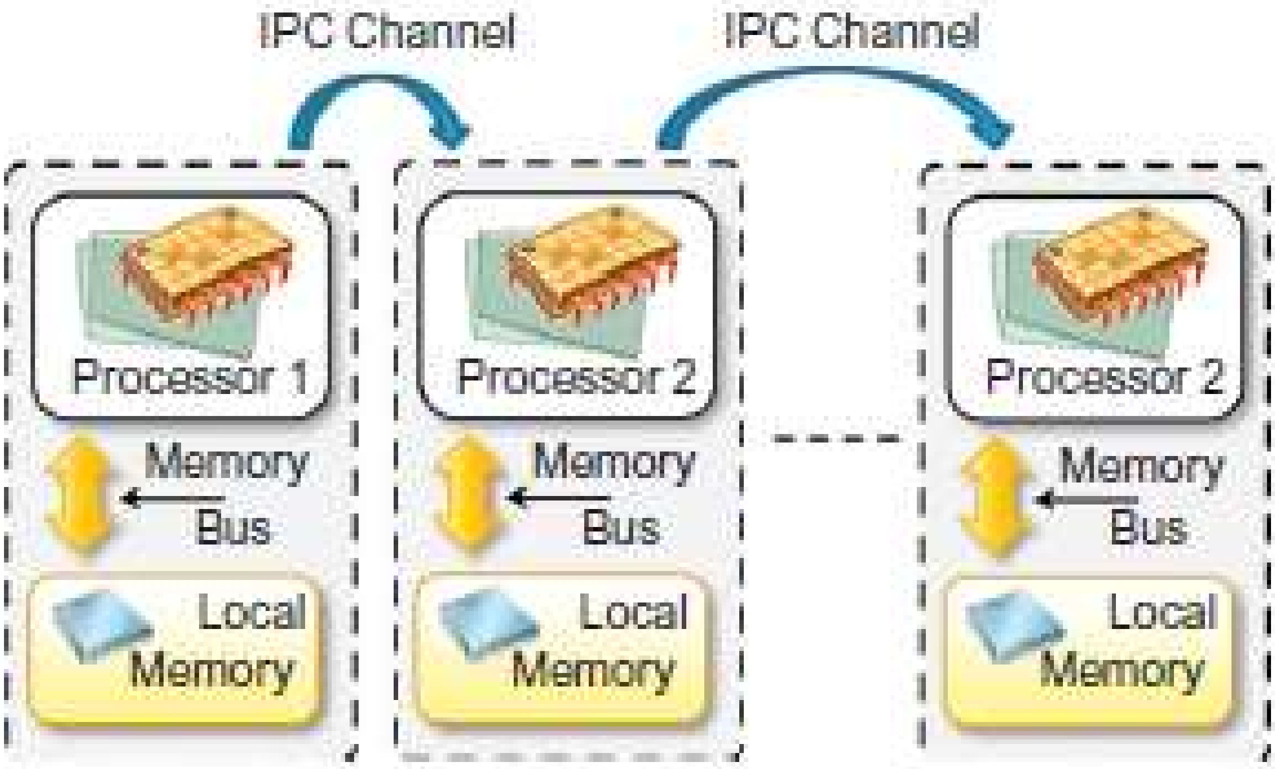


Distributed Memory MIMD

- A
  c
- C
  ir

(Int

- E



Comparison of MIMD

Shared MIMD

- Easier to program
- Less tolerant to failures
  - Failure affects entire system
- Harder to extend
  - Scaling (addition of more PEs) leads to memory contention.

Distributed MIMD

- Difficult to program wrt shared MIMD
- Tolerant to failures due to each PEs can be easily isolated.
- Easy to scale due to each PE has its own memory.

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.

Approaches to Parallel Programming

- Most Prominent Parallel Programming Approach
  - Data Parallelism
  - Process Parallelism
  - Farmer and Worker Model
- All these models are suitable for task level parallelism.
- Data Parallelism
  - Divide-and-conquer is used to split data into multiple sets which executed by each PEs using same instruction.
  - Suitable for SIMD

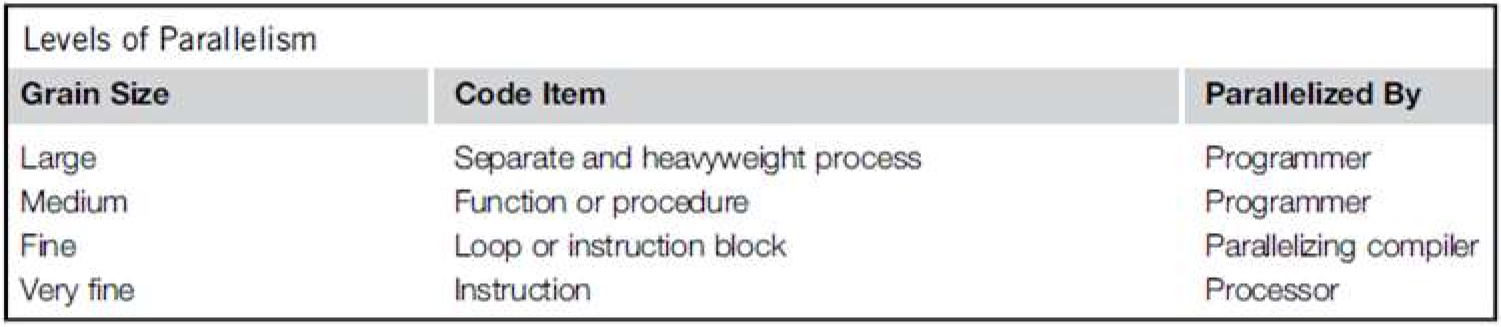
Approaches to Parallel Programming

- Process Parallelism
  - Given operation has multiple activities that can be processed on multiple processors.
- Farmer-and-worker model
  - Job distribution approach is used. One PE is master and all other remaining PEs are slaves.
  - Master assigns job to slaves. Slaves inform master on completion.

Data Parallelisms	Task Parallelisms
1. Same task are performed on different subsets of same data.	1. Different task are performed on the same or different data.
2. Synchronous computation is performed.	2. Asynchronous computation is performed.
3. As there is only one execution thread operating on all sets of data, so the speedup is more.	3. As each processor will execute a different thread or process on the same or different set of data, so speedup is less.
4. Amount of parallelization is proportional to the input size.	4. Amount of parallelization is proportional to the number of independent tasks is performed.
5. It is designed for optimum load balance on multiprocessor system.	5. Here, load balancing depends upon on the e availability of the hardware and scheduling algorithms like static and dynamic scheduling.

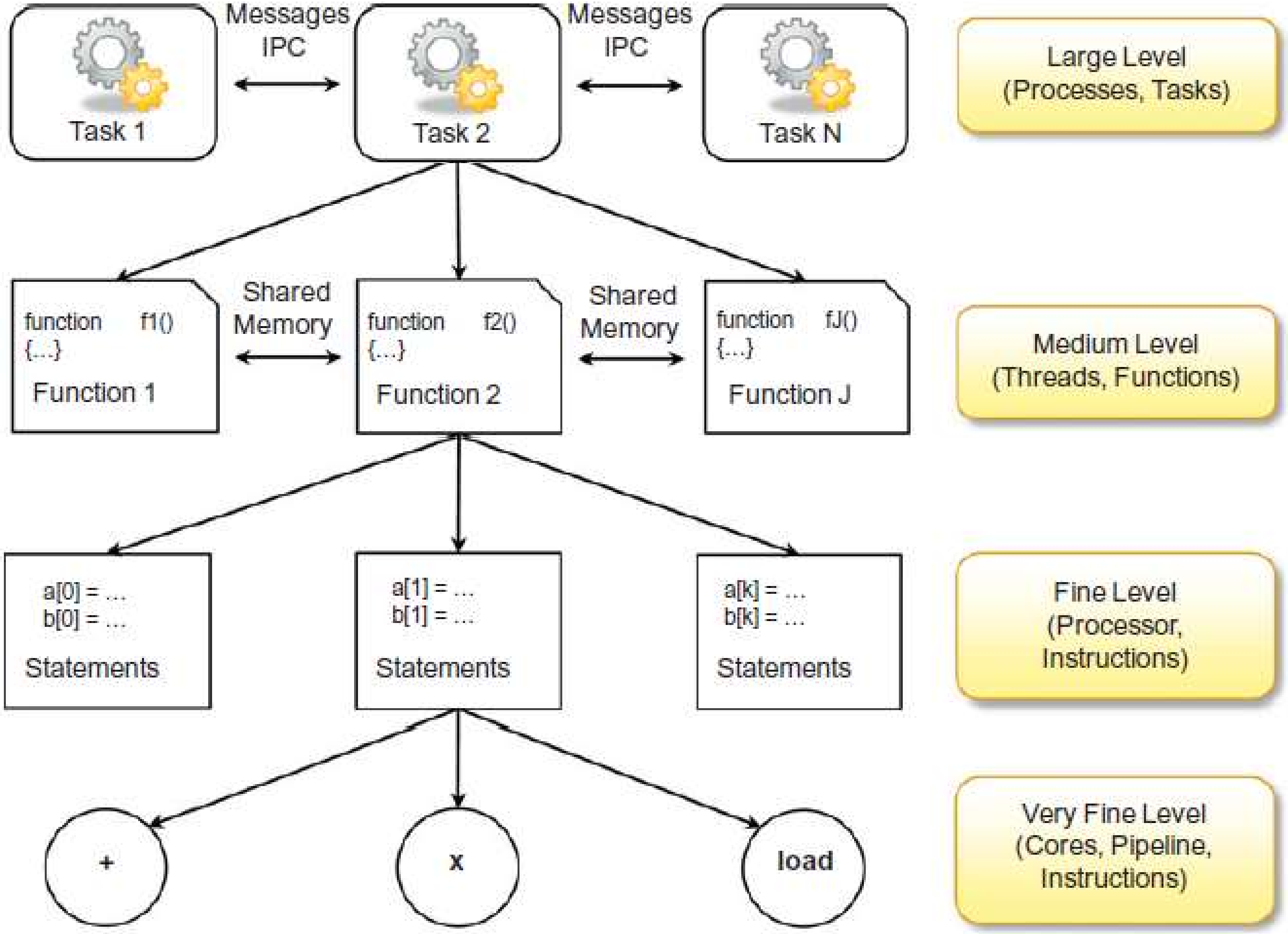
Level of Parallelism

- Levels of parallelism are decided based on the lumps of code (grain size) that can be a potential candidate for parallelism.
- All these approaches have a common goal: to boost processor efficiency by hiding ***latency***.



Levels of Parallelism

Grain Size	Code Item	Parallelized By
Large	Separate and heavyweight process	Programmer
Medium	Function or procedure	Programmer
Fine	Loop or instruction block	Parallelizing compiler
Very fine	Instruction	Processor



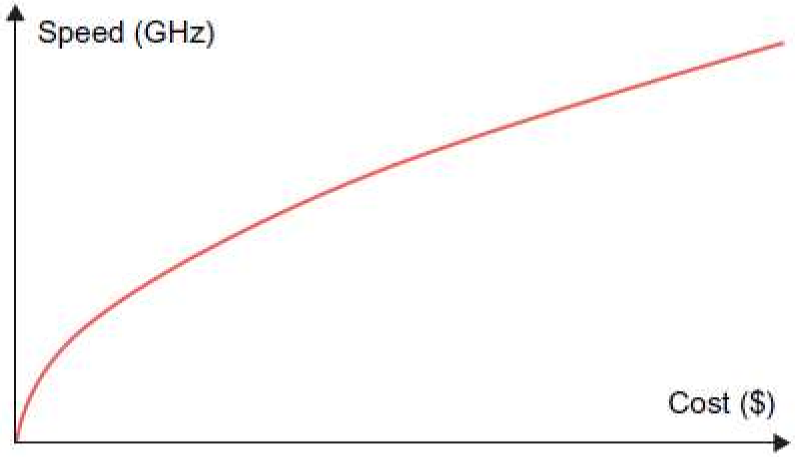
Messages IPC
Messages IPC
Task 1
Task 2
Task N
Large Level (Processes, Tasks)
Shared Memory
Shared Memory
function f1() {...}
Function 1
function f2() {...}
Function 2
function fJ() {...}
Function J
Medium Level (Threads, Functions)
a[0] = ...
b[0] = ...
Statements
a[1] = ...
b[1] = ...
Statements
a[k] = ...
b[k] = ...
Statements
Fine Level (Processor, Instructions)
+
x
load
Very Fine Level (Cores, Pipeline, Instructions)

Law of Caution

- 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.

Law of Caution

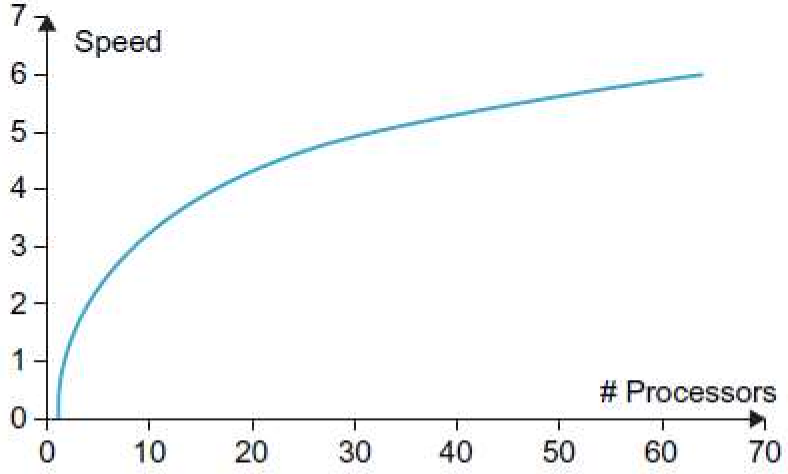
- Speed of computation is proportional to the square root of system cost; they never increase linearly.
- Therefore, the faster a system becomes, the more expensive it is to increase its speed



Law of Caution

- Speed by a parallel computer increases as the logarithm of the number of processors

$y = k*\log(N).$



Distributed Computing

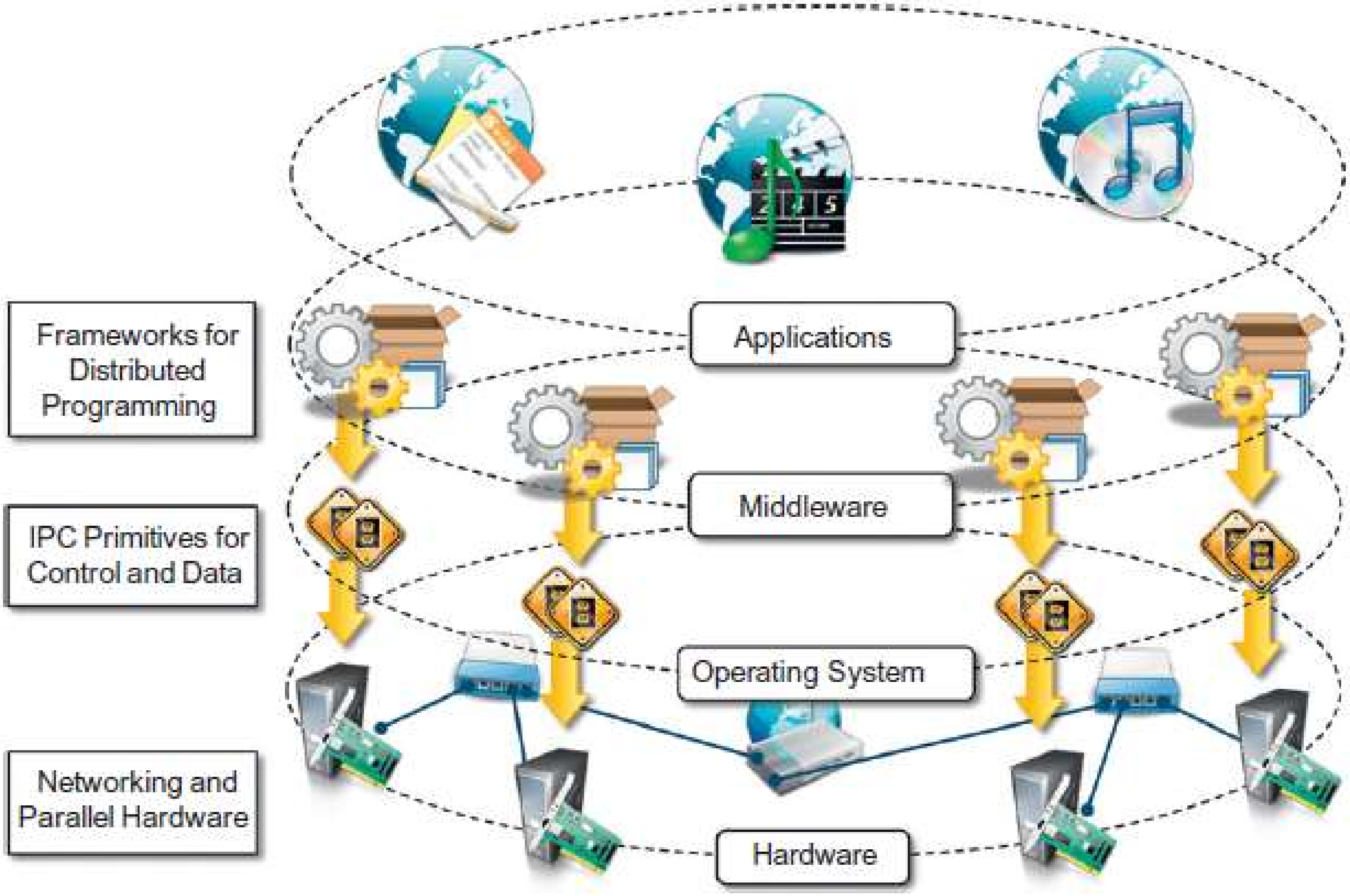
- Distributed computing studies the models, architectures, and algorithms used for building and managing distributed systems.
- **Definition:** A distributed system is a collection of independent computers that appears to its users as a single coherent system.

Distributed Computing

- Communication is another fundamental aspect of distributed computing.
- Since distributed systems are composed of **more than one computer** that **collaborate** together, it is necessary to provide some sort of **data and information exchange** between them, which generally occurs **through the network.**
- **Definition:** A distributed system is one in which components located at networked computers communicate and coordinate their actions only by passing messages.

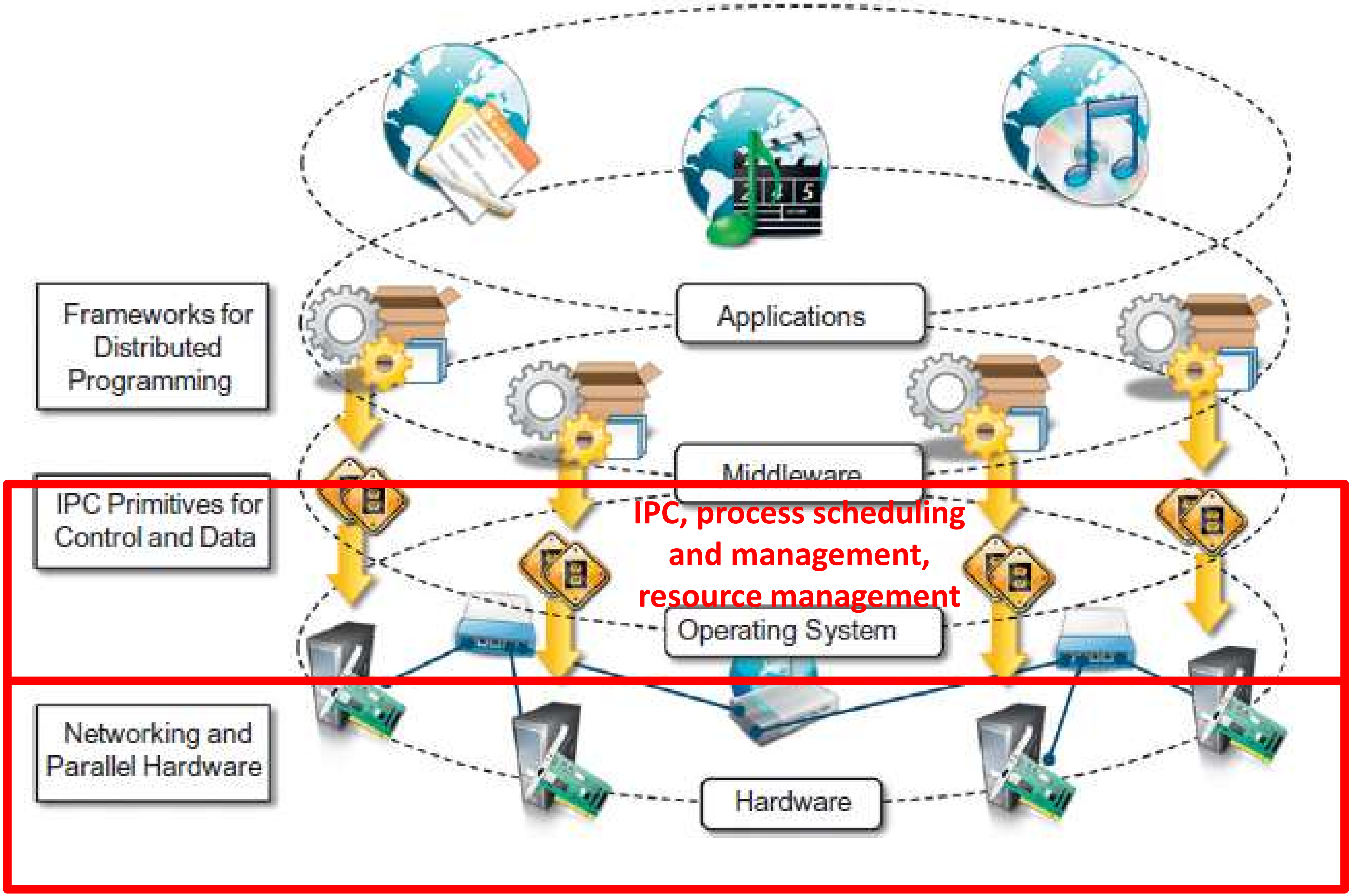
Components of Distributed System

- 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** .


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

Components of Distributed System

- At the very bottom layer, computer and network hardware constitute the physical infrastructure.
- Managed by operating system provide basic services for:
  - IPC
  - Process scheduling and management
  - Resource management of file system and local devices
- These two layers become the platform on top of specialized software makes distributed system.


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

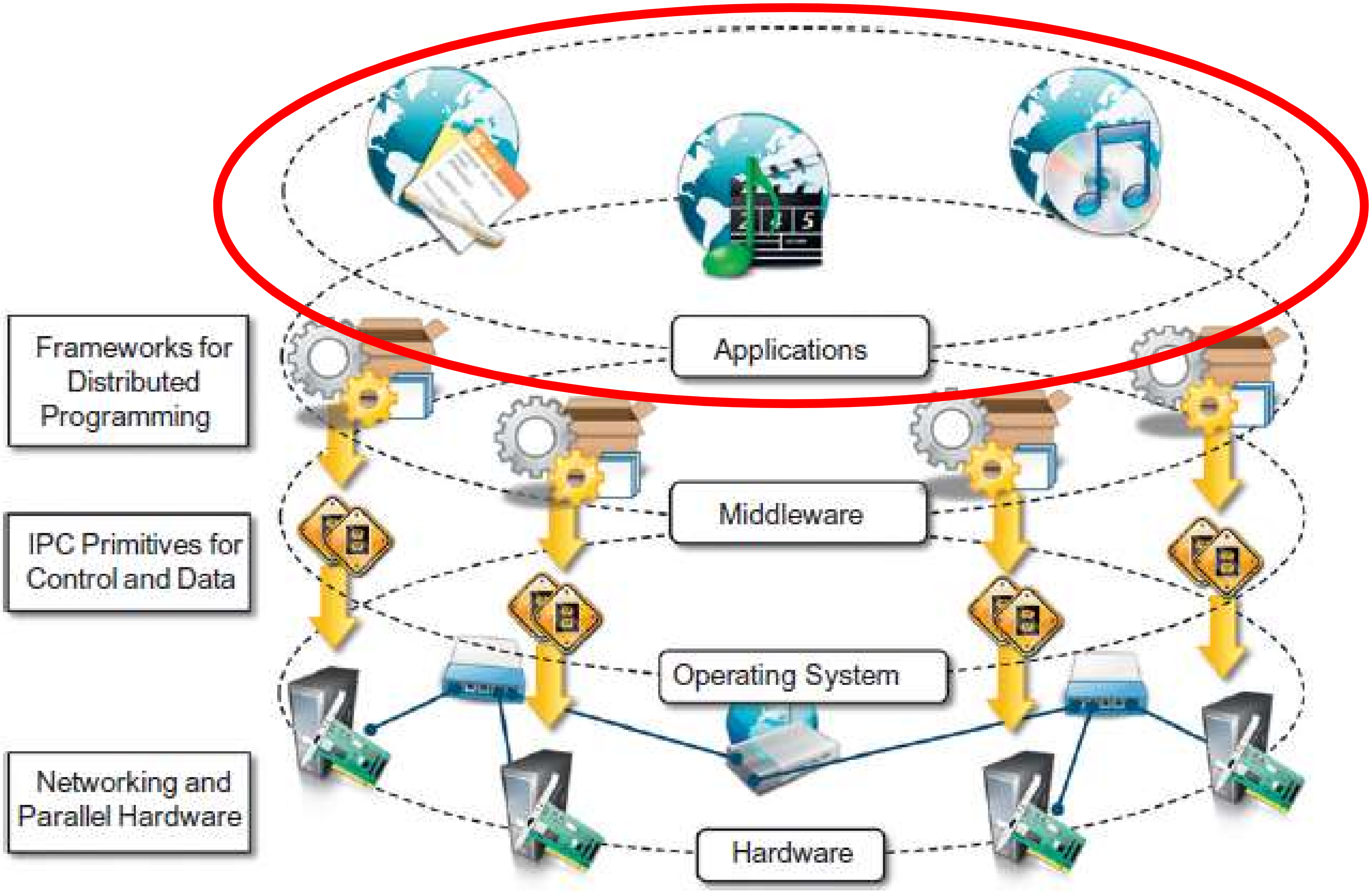
IPC, process scheduling and management, resource management

Components of Distributed System

- The middleware layer leverages these services for development and deployment of distributed applications.
- The middleware develop its own protocols, data formats and programing language or framework for distributed application development.

Components of Distributed System

- All of them constitute uniform interface to distributed applications developers.
- That is completely independent from the underlying operating system and hide all heterogeneities of bottom layers.
- At the top applications and services designed and developed to use the middleware.


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

- Design patterns represent the best practices used by experienced object-oriented software developers.
- Design patterns are solutions to general problems that software developers faced during software development.
- These solutions were obtained by trial and error by numerous software developers over quite a substantial period of time.

Architectural styles for distributed computing

- 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 that, taken together, constitute such an environment.

Architectural styles for distributed computing

- The interactions among these components and their responsibilities give structure to the middleware and characterize its type or, in other words, define its architecture.
- Architectural styles aid in understanding and classifying the organization of software systems in general and distributed computing in particular.

Architectural styles for distributed computing

- Architectural styles are mainly used to determine the **vocabulary of components and connectors** that are used as instances of the style together with a **set of constraints** on how they can be combined.
- Design patterns help in creating a common knowledge - how to structure the relations of components within an application and understand the internal organization of software applications.
- Architectural styles do the same for the overall architecture of software systems.

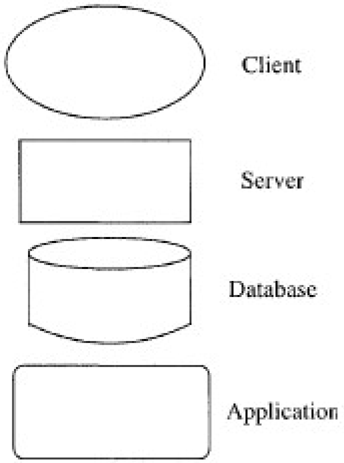
Architectural styles for distributed computing

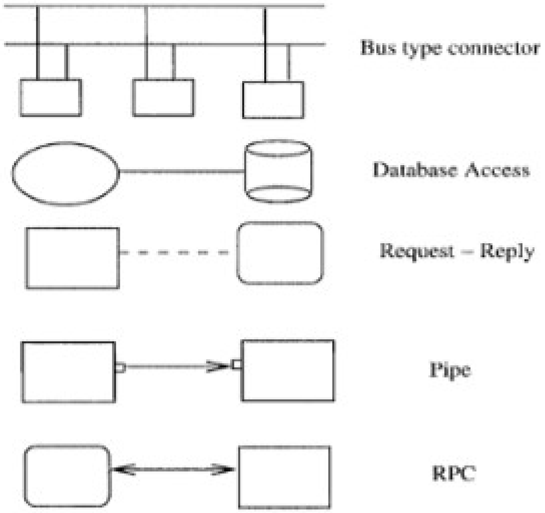
- Two major classes of architectural styles are:
  - Software architectural styles: relates to the logical organization of the software
  - System architectural styles: includes all those styles that describe the physical organization of distributed software systems in terms of their major components.

Component and connectors

- A component represents 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.
- Differently from components, connectors are not encapsulated in a single entity, but they are implemented in a distributed manner over many system components.

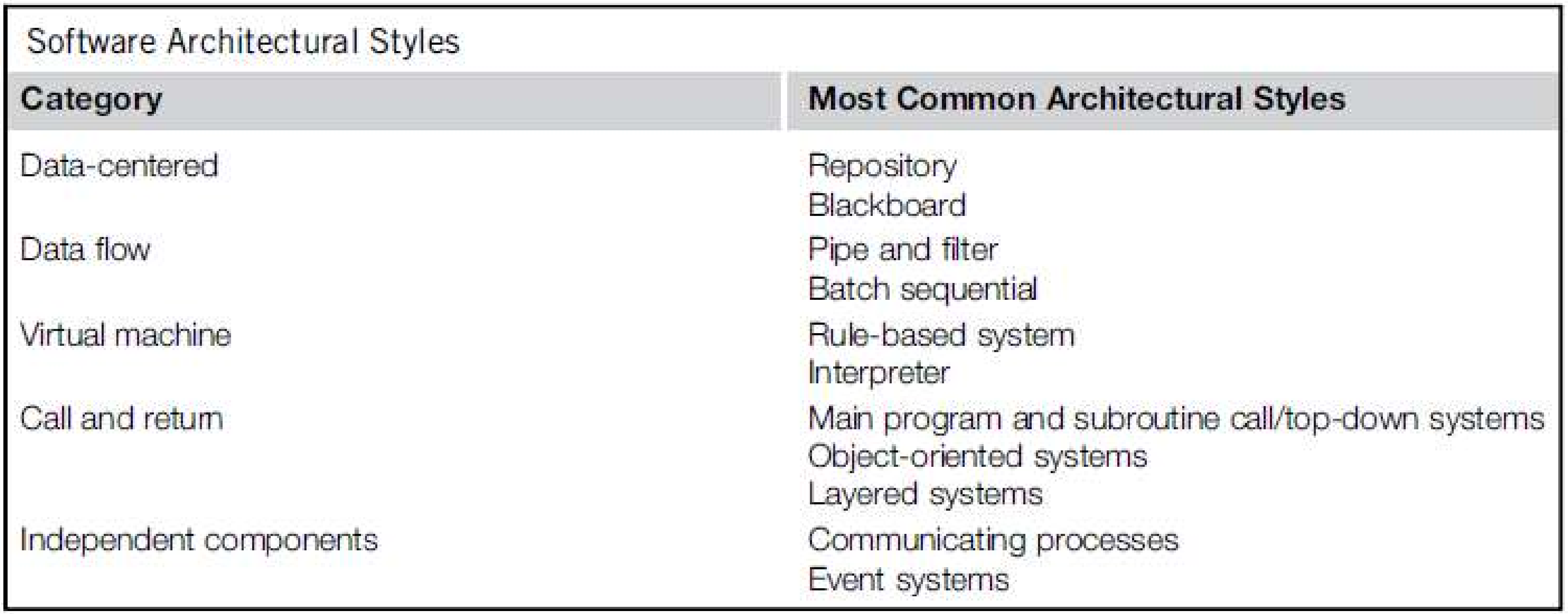
Component and connectors





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.
- They also identify the main abstractions that are used to shape the components of the system and the expected interaction patterns between them.



Software Architectural Styles

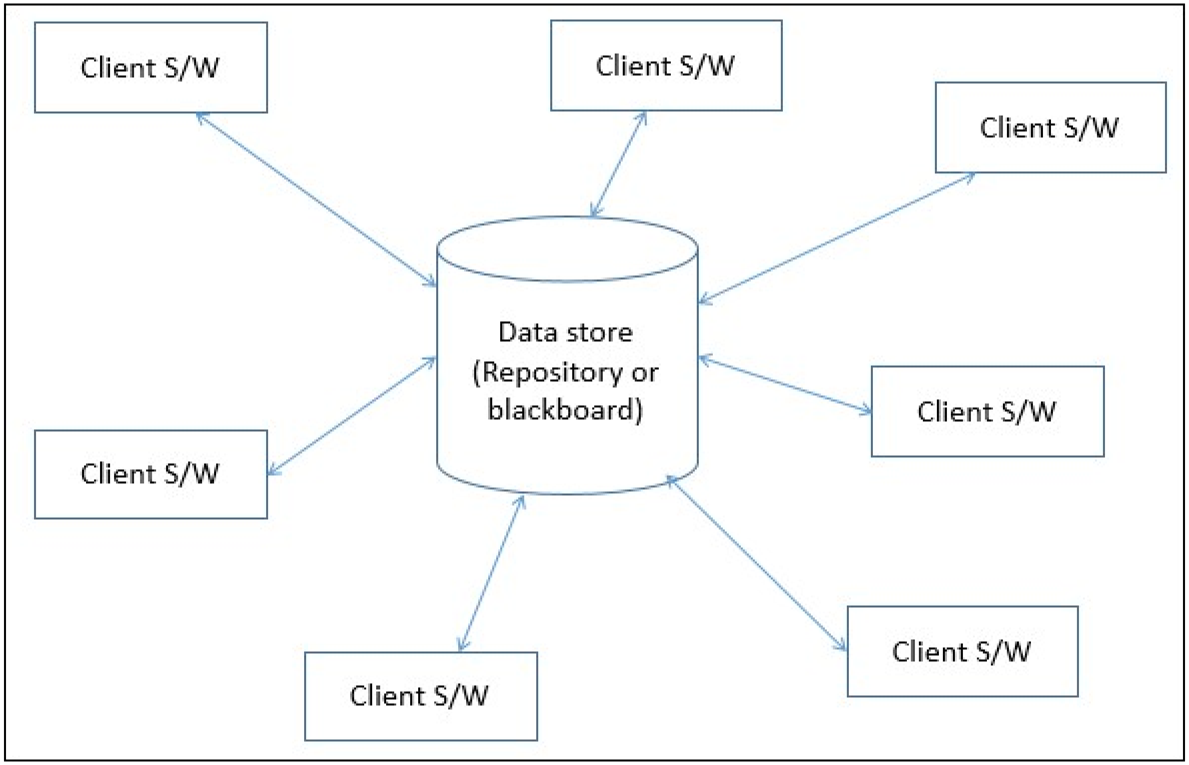
Category	Most Common Architectural Styles
Data-centered	Repository Blackboard
Data flow	Pipe and filter Batch sequential
Virtual machine	Rule-based system Interpreter
Call and return	Main program and subroutine call/top-down systems Object-oriented systems Layered systems
Independent components	Communicating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

Data Centered Architectures

- Data as the fundamental element of the software system, and access to shared data is the core characteristic of the data-centered architectures.
- Integrity of data is the overall goal for such systems.
  - Repository architectural style:
  - Blackboard architectural style

Repository architecture style

- The repository architectural style is characterized by two main components:
  - – The **central data structure**, which represents the current state of the system.
  - – A Data **accessor** or a collection of independent components, which operate on the central data.


Client S/W
Client S/W
Client S/W
Data store
(Repository or
blackboard)
Client S/W
Client S/W
Client S/W
Client S/W

Repository architecture style

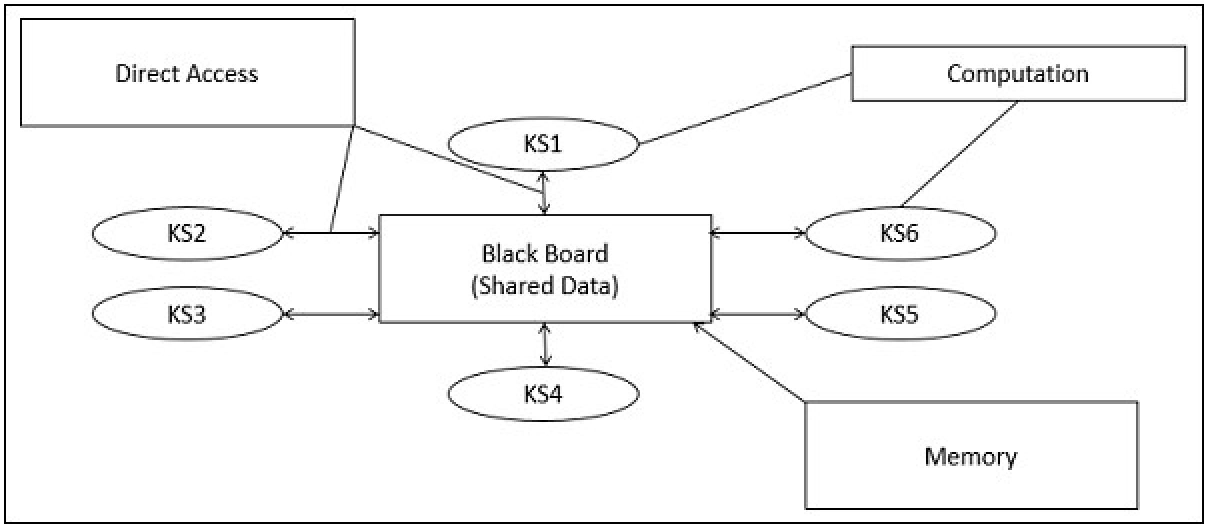
- The client sends a request to the system to perform actions (e.g. insert data).
- The computational processes are independent and triggered by incoming requests.
- If the types of transactions in an input stream of transactions trigger selection of processes to execute, then it is traditional database or repository architecture, or passive repository.
- This approach is widely used in DBMS, library information system, the interface repository in CORBA, compilers and CASE (computer aided software engineering) environments.

Repository architecture style

- **Advantages**
  - Provides data integrity, backup and restore features.
  - Provides scalability and reusability of agents as they do not have direct communication with each other.
  - Reduces overhead of transient data between software components.
- **Disadvantages**
  - It is more vulnerable to failure and data replication or duplication is possible.
  - High dependency between data structure of data store and its agents.
  - Changes in data structure highly affect the clients.
  - Evolution of data is difficult and expensive.
  - Cost of moving data on network for distributed data.

Blackboard Architecture Style

- The data store is active and its clients are passive.
- Therefore the logical flow is determined by the current data status in data store.
- It has a blackboard component, acting as a central data repository, and an internal representation is built and acted upon by different computational elements.


Direct Access
Computation
KS1
KS2
KS3
KS4
KS5
KS6
Black Board
(Shared Data)
Memory

Blackboard Architecture Style

- A number of components that act independently on the common data structure are stored in the blackboard.
- In this style, the components interact only through the blackboard. The data-store alerts the clients whenever there is a data-store change.
- The current state of the solution is stored in the blackboard and processing is triggered by the state of the blackboard.

Blackboard Architecture Style

- The system sends notifications known as **trigger** and data to the clients when changes occur in the data.
- This approach is found in certain AI applications and complex applications, such as speech recognition, image recognition, security system, and business resource management systems etc.

Blackboard Architecture Style

- If the current state of the central data structure is the main trigger of selecting processes to execute, the repository can be a blackboard and this shared data source is an active agent.
- A major difference with traditional database systems is that the invocation of computational elements in a blackboard architecture is triggered by the current state of the blackboard, and not by external inputs.

Blackboard Architecture Style

- The blackboard architectural style is characterized by three main components:
  - **Knowledge sources**: also known as **Listeners** or **Subscribers** are the entities that update the knowledge base that is maintained in the blackboard.
  - **Blackboard**: This represents the data structure that is shared among the knowledge sources and stores the knowledge base of the application.
  - **Control**: The control is the collection of triggers and procedures that govern the interaction with the blackboard and update the status of the knowledge base.

Blackboard Architecture Style

- **Advantages**
  - Provides scalability which provides easy to add or update knowledge source.
  - Provides concurrency that allows all knowledge sources to work in parallel as they are independent of each other.
  - Supports experimentation for hypotheses.
  - Supports reusability of knowledge source agents.
- **Disadvantages**
  - The structure change of blackboard may have a significant impact on all of its agents as close dependency exists between blackboard and knowledge source.
  - It can be difficult to decide when to terminate the reasoning as only approximate solution is expected.
  - Problems in synchronization of multiple agents.
  - Major challenges in designing and testing of system.

Data-flow architectures

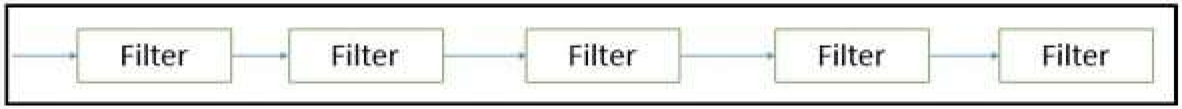
- Whole software system is seen as a series of transformations on consecutive pieces or set of input data, where data and operations are independent of each other.
- The data enters into the system and then flows from component to component – forms communication between them.
- The connections between the components or modules may be implemented as I/O stream, I/O buffers, piped, or other types of connections.

Data-flow architectures

- There are three types of execution sequences between modules–
  - Batch Sequential Style
  - Pipe-and-Filter Style
  - Process control

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.
- Typical application of this architecture includes business data processing such as banking and utility billing.
- It is very common to compose these phases using the batch- sequential style.

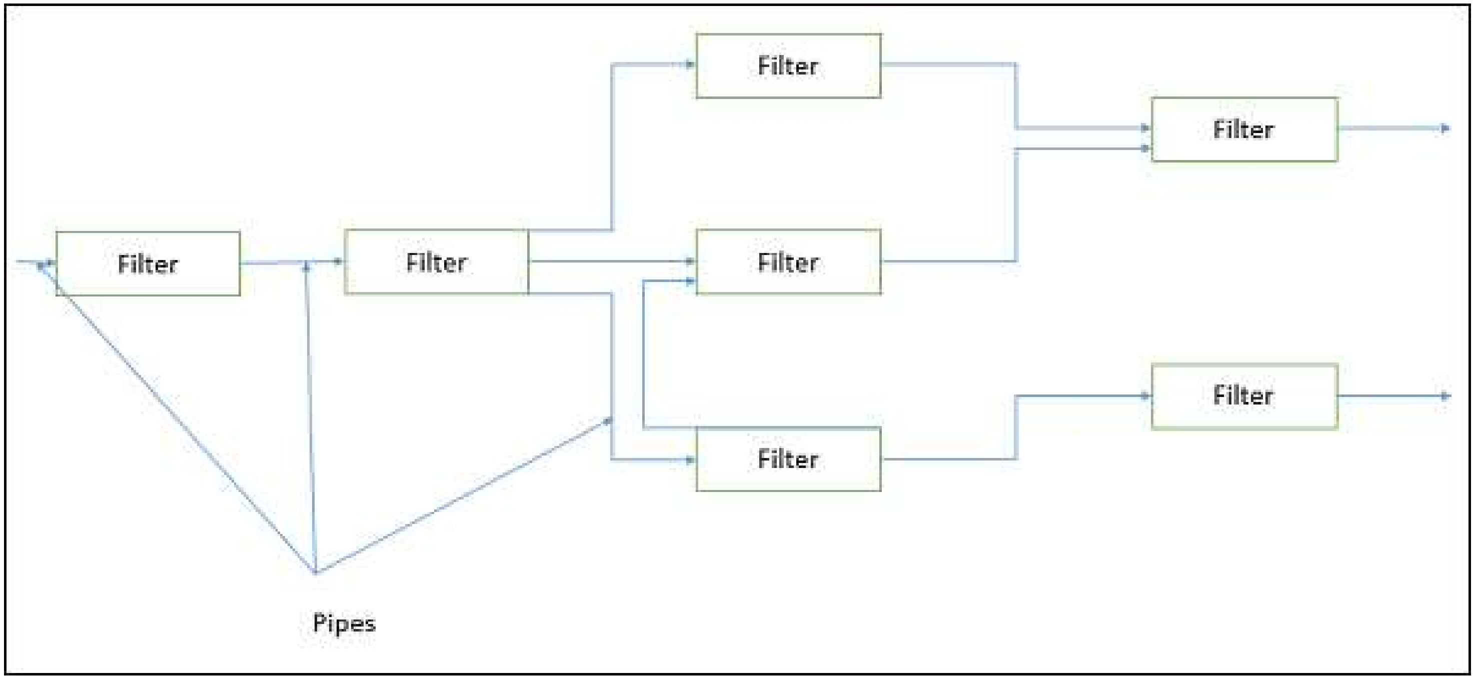


Batch Sequential Style

- **Advantages**
  - Provides simpler divisions on subsystems.
  - Each subsystem can be an independent program working on input data and producing output data.
- **Disadvantages**
  - Provides high latency and low throughput.
  - Does not provide concurrency and interactive interface.
  - External control is required for implementation.

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 the connection between one filter and the next is represented by a **data stream**.


Filter
Filter
Filter
Filter
Filter
Filter
Filter
Filter
Pipes

Pipe-and-Filter Style

- In the batch sequential style, data is processed incrementally and each filter processes the data as soon as it is available on the input stream.
- As soon as one filter produces a consumable amount of data, the next filter can start its processing.

Pipe-and-Filter Style

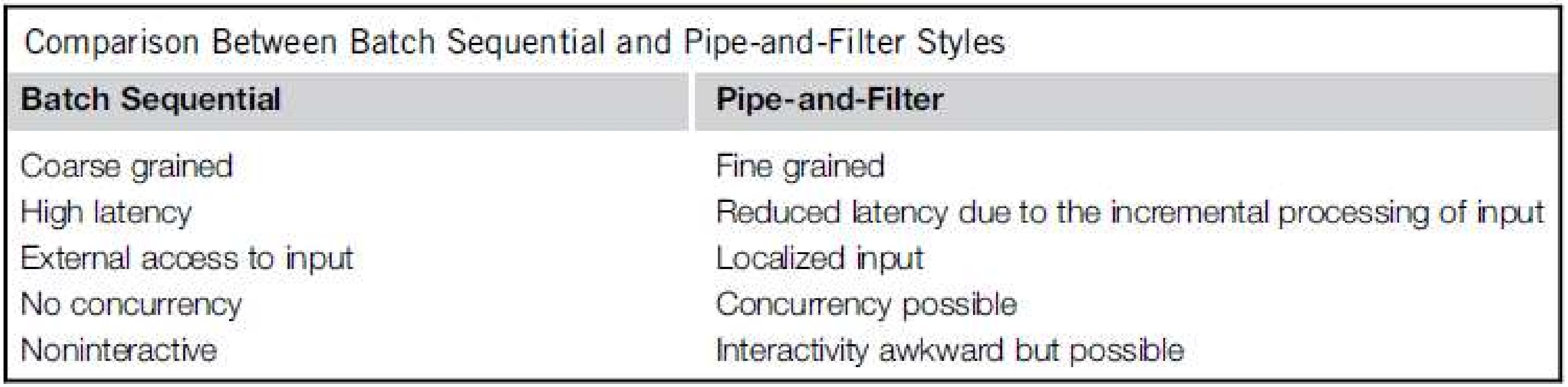
- Filters generally do not have state, know the identity of neither the previous nor the next filter, and they are connected with in-memory data structures such as first-in/first-out(FIFO) buffers or other structures.
- This particular sequencing is called pipelining and introduces concurrency in the execution of the filters.

Pipe-and-Filter Style

- A classic example of this architecture is the microprocessor pipeline, where by multiple instructions are executed at the same time by completing a different phase of each of them.
- Another example are the Unix shell pipes (i.e., cat filename| grep <pattern>| wc -l)
- Applications of this architecture can also be found in the compiler design, generation), image and signal processing, and voice and videostreaming.

Process Control

- It is a type of data flow architecture where data is neither batched sequential nor pipelined stream.
- The flow of data comes from a set of variables, which controls the execution of process.
- It decomposes the entire system into subsystems or modules and connects them.
- i.e. Real-time system software to control automobile anti-lock brakes, nuclear power plants, etc



Comparison Between Batch Sequential and Pipe-and-Filter Styles

Batch Sequential	Pipe-and-Filter
Coarse grained	Fine grained
High latency	Reduced latency due to the incremental processing of input
External access to input	Localized input
No concurrency	Concurrency possible
Noninteractive	Interactivity awkward but possible

Virtual machine architectures

- The virtual machine class of 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.

Virtual machine architectures

- 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.

Virtual machine architectures

- Virtual machine architectural styles are characterized by an indirection layer between applications and the hosting environment.
- This design has the major advantage of decoupling applications from the underlying hardware and software environment, but at the same time it introduces some disadvantages, such as a slowdown in performance.
- Other issues might be related to the fact that, by providing a virtual execution environment, specific features of the underlying system might not be accessible.

Rule-based Style

- This architecture is characterized by representing the abstract execution environment as an inference engine.
- Programs are expressed in the form of rules or predicates that hold true.
- The input data for applications is generally represented by a set of assertions or facts that the inference engine uses to activate rules or to apply predicates, thus transforming data.

Rule-based Style

- The output can either be the product of the rule activation or a set of assertions that holds true for the given input data.
- The set of rules or predicates identifies the knowledge base that can be queried to infer properties about the system.
- This approach is quite peculiar, since it allows expressing a system or a domain in terms of its behavior rather than in terms of the components.

Rule-based Style

- 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 PLCs and by activating alarms when specific conditions on the sensory data apply.
- 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.

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.
- The interpretation of the pseudo-program constitutes the execution of the program itself.
- Systems modeled according to this style exhibit four main components:
  - the interpretation engine that executes the core activity of this style,

Interpreter Style

  - an internal memory that contains the pseudo-code to be interpreted,
  - a representation of the current state of the engine, and
  - a representation of the current state of the program being executed.

- This model is quite useful in designing virtual machines for high level programming (Java, C#) and scripting languages (awk, PERL, and so on).

Call & return architectures

- This category identifies all systems that are organized into components mostly connected together by method calls.
- The activity of systems modeled in this way is characterized by a chain of method calls whose overall execution and composition identify the execution of one or more operations.
- The internal organization of components and their connections may vary.

Call & return architectures

- 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, and
  - layered style

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. T
- he 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.

Top-Down Style

- 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.

Object-oriented Style

- This architectural style encompasses a wide range of systems that have been designed and implemented by leveraging the abstractions of object-oriented programming (OOP).
- Systems are specified in terms of classes and implemented in terms of objects.
- Classes define the type of components by specifying the data that represent their state and the operations that can be done over these data.

Object-oriented Style

- 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.

Layered Style

- The layered system style allows the design and implementation of software systems in terms of layers, which provide a different level of abstraction of the system.
- Each layer generally operates with at most two layers: the one that provides a lower abstraction level and the one that provides a higher abstraction layer.
- Specific protocols and interfaces define how adjacent layers interact.

Layered Style

- It is possible to model such systems as a stack of layers, one for each level of abstraction.
- Therefore, the components are the layers and the connectors are the interfaces and protocols used between adjacent layers.
- A user or client generally interacts with the layer at the highest abstraction, which, in order to carry its activity, interacts and uses the services of the lower layer.
- This process is repeated (if necessary) until the lowest layer is reached.

Layered Style

- It is also possible to have the opposite behavior: events and callbacks from the lower layers can trigger the activity of the higher layer and propagate information up through the stack.
- The advantages of the layered style are that, as happens for the object-oriented style, it supports a modular design of systems and allows us to decompose the system according to different levels of abstractions by encapsulating together all the operations that belong to a specific level.
- Layers can be replaced as long as they are compliant with the expected protocols and interfaces, thus making the system flexible.

Layered Style

- The main disadvantage is constituted by the lack of extensibility, since it is not possible to add layers without changing the protocols and the interfaces between layers.
- This also makes it complex to add operations.
- Examples of layered architectures are the modern operating system kernels and the International Standards Organization/Open Systems Interconnection(ISO/OSI) or the TCP/IP stack.

Architectural styles based on independent components

- This class of architectural style models systems 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 .

Communicating Processes

- Components are represented by independent processes that leverage IPC facilities for coordination management.
- This is an abstraction that is quite suitable to modeling 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 leverage the services exposed by the other processes.

Communicating 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.
- Connectors are identified by IPC facilities used by these processes to communicate.

Event Systems

- In this architectural style, the components of the system are loosely coupled and connected.
- In addition to exposing operations for data and state manipulation, each component also 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.

Event Systems

- 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.

Event Systems

- The main advantage of such an architectural style is that it fosters the development of open systems: new modules can be added and easily integrated into the system as long as they have compliant interfaces for registering to the events.
- This architectural style solves some of the limitations observed for the top-down and object-oriented styles.
- First, the invocation pattern is implicit, and the connection between the caller and the callee is not hard-coded; this gives a lot of flexibility since addition or removal of a handler to events can be done without changes in the source code of applications.

Event Systems

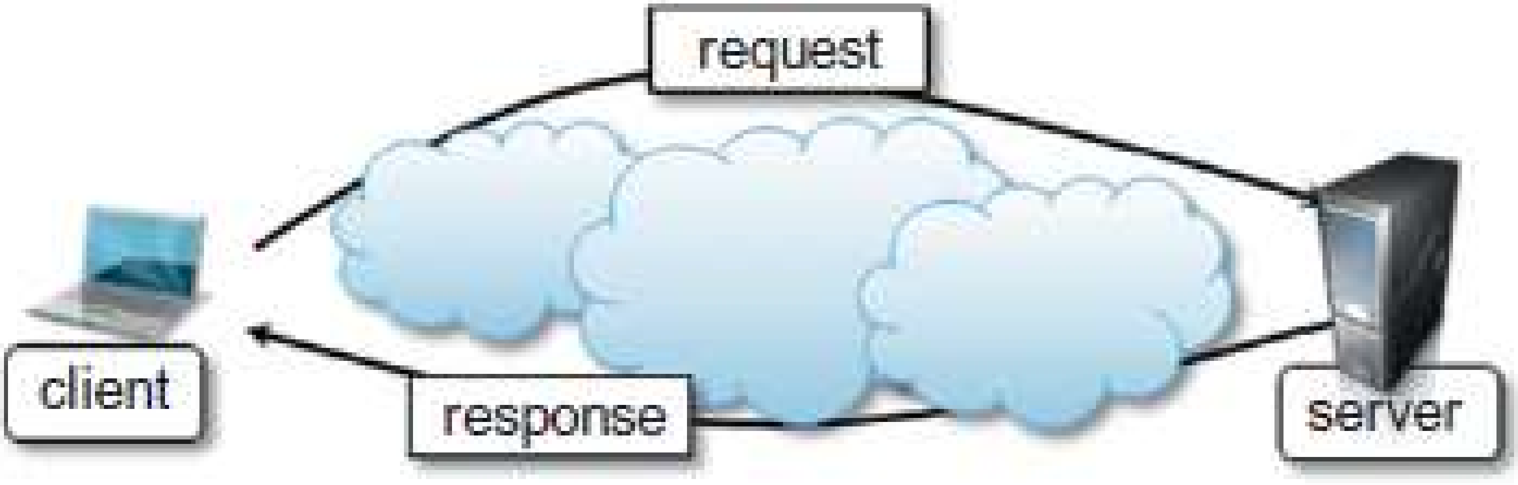
- Second, the event source does not need to know the identity of the event handler in order to invoke the callback.
- The disadvantage of such a style is that it relinquishes control over system computation.
- When a component triggers an event, it does not know how many event handlers will be invoked and whether there are any registered handlers.
- This information is available only at runtime and, from a static design point of view, becomes more complex to identify the connections among components and to reason about the correctness of the interactions.

System architectural styles

- System architectural styles cover the physical organization of components and processes over a distributed infrastructure.
- They provide a set of reference models for the deployment of such systems and help engineers not only have a common vocabulary in describing the physical layout of systems but also quickly identify the major advantages and drawbacks of a given deployment and whether it is applicable for a specific class of applications.
- Two fundamental reference styles are client/server and peer-to-peer.

Client/server

- This architecture is very popular in distributed computing and is suitable for a wide variety of applications.
- The client/server model features two major components: a server and a client.
- These two components interact with each other through a network connection using a given protocol.


request
client
response
server

Client/server

- The communication is unidirectional: The client issues a request to the server, and after processing the request the server returns a response.
- There could be multiple client components issuing requests to a server that is passively waiting for them.
- Hence, the important operations in the client-server paradigm are request, accept (client side), and listen and response (server side).

Client/server

- The client/server model is suitable in many-to-one scenarios, where the information and the services of interest can be centralized and accessed through a single access point: the server.
- In general, multiple clients are interested in such services and the server must be appropriately designed to efficiently serve requests coming from different clients.
- This consideration has implications on both client design and server design.

Client/server

- For the client design, we identify two major models: Thin-client model and Fat-client model.
- Thin-client model
  - The load of data processing and transformation is put on the server side
  - The client has a light implementation that is mostly concerned with retrieving and returning the data it is being asked for, with no considerable further processing.

Client/server

- Fat-client model
  - The client component is also responsible for processing and transforming the data before returning it to the user.
  - The server features a relatively light implementation that is mostly concerned with the management of access to the data.

Client/server

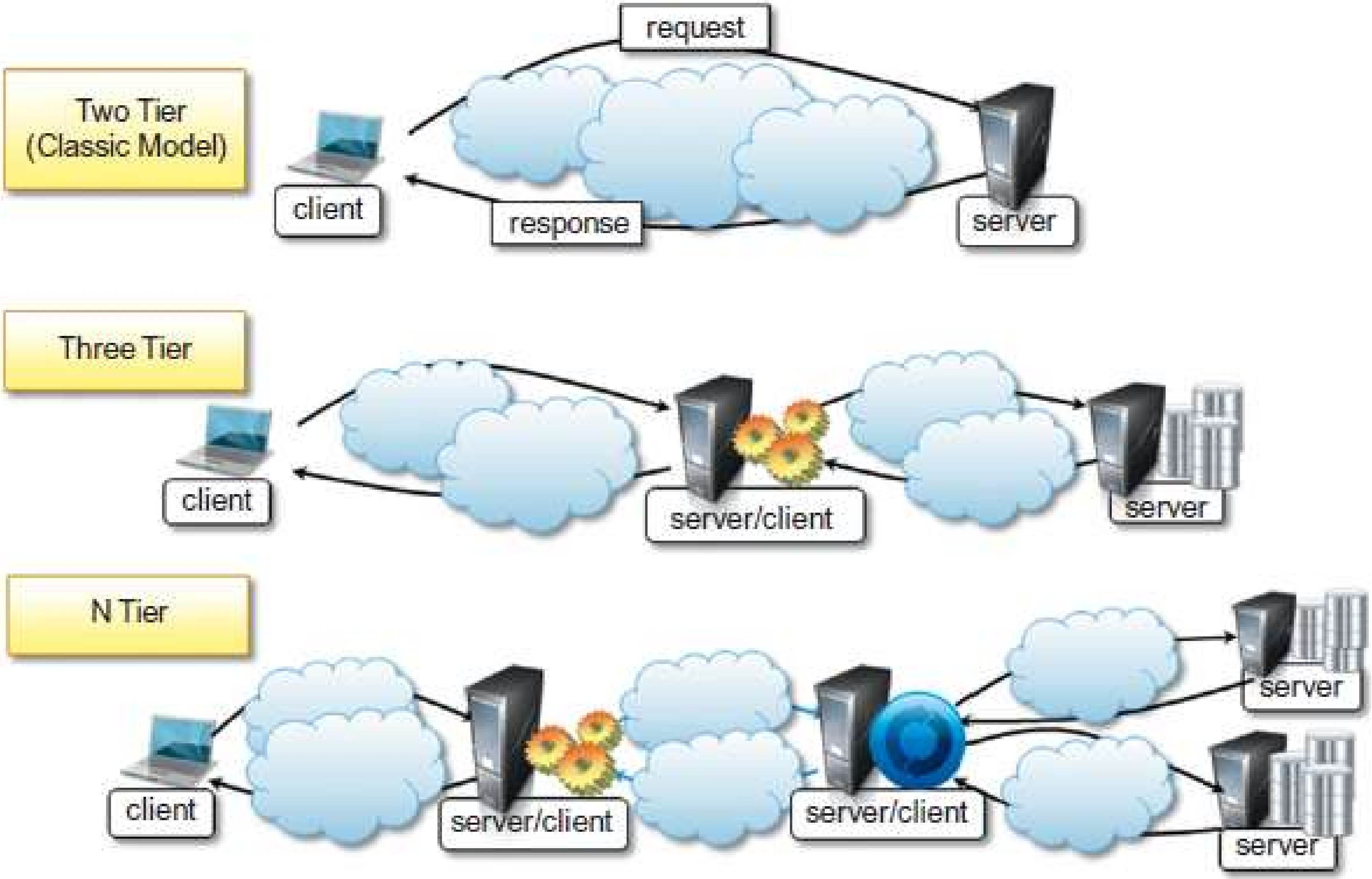
- The three major components in the client-server model:
  - Presentation
  - Application logic
  - Data storage

Client/server

- In the thin-client model, the client embodies only the presentation component, while the server absorbs the other two.
- In the fat-client model, the client encapsulates presentation and most of the application logic, and the server is principally responsible for the data storage and maintenance.

Client/server

- Presentation, application logic, and data maintenance can be seen as conceptual layers, which are more appropriately called tiers.
- The mapping between the conceptual layers and their physical implementation in modules and components allows differentiating among several types of architectures, which go under the name of multi-tiered architectures



Two Tier
(Classic Model)
request
client
response
server
Three Tier
client
server/client
server
N Tier
client
server/client
server/client
server
server

Two-tier architecture

- This architecture partitions the systems into two tiers, which are located one in the client component and the other on the server.
- The client is responsible for the presentation tier by providing a user interface; the server concentrates the application logic and the data store into a single tier.

Two-tier architecture

- The server component is generally deployed on a powerful machine that is capable of processing user requests, accessing data, and executing the application logic to provide a client with a response.
- This architecture is suitable for systems of limited size and suffers from scalability issues.

Two-tier architecture

- Performance decrease as the number of users increases.
- Another limitation is caused by the dimension of the data to maintain, manage, and access, which might be prohibitive for a single computation node or too large for serving the clients with satisfactory performance.

Three-tier/N-tier architecture

- The 3-tier architecture separates the presentation of data, the application logic, and the data storage into three tiers.
- This architecture is generalized into an N-tier model in case it is necessary to further divide the stages composing the application logic and storage tiers.
- More scalable than the two-tier because it is possible to distribute the tiers into several computing nodes, thus isolating the performance bottlenecks.

Three-tier/N-tier architecture

- More complex to understand and manage.
- A classic example of three-tier architecture is constituted by a medium-size Web application that relies on a relational database management system for storing its data.
  - Client component – Web browser (presentation tier)
  - Business logic – Application server (business logic tier)
  - Data storage – Database server (data tier)
- Application servers that rely on third-party (or external) services to satisfy client requests are examples of N-tiered architectures.

Client-server - Summary

- The client/server architecture has been the dominant reference model for designing and deploying distributed systems.
- It is generally suitable in the case of a many-to-one scenario
- The interaction is unidirectional.
- It suffers from scalability issues, and therefore it is **not appropriate in very large systems**.

Peer-to-peer

- A symmetric architecture in which all the components, called peers, play the same role and incorporate both client and server capabilities of the client/server model.
- Each peer acts as a server when it processes requests from other peers and as a client when it issues requests to other peers.

Peer-to-peer

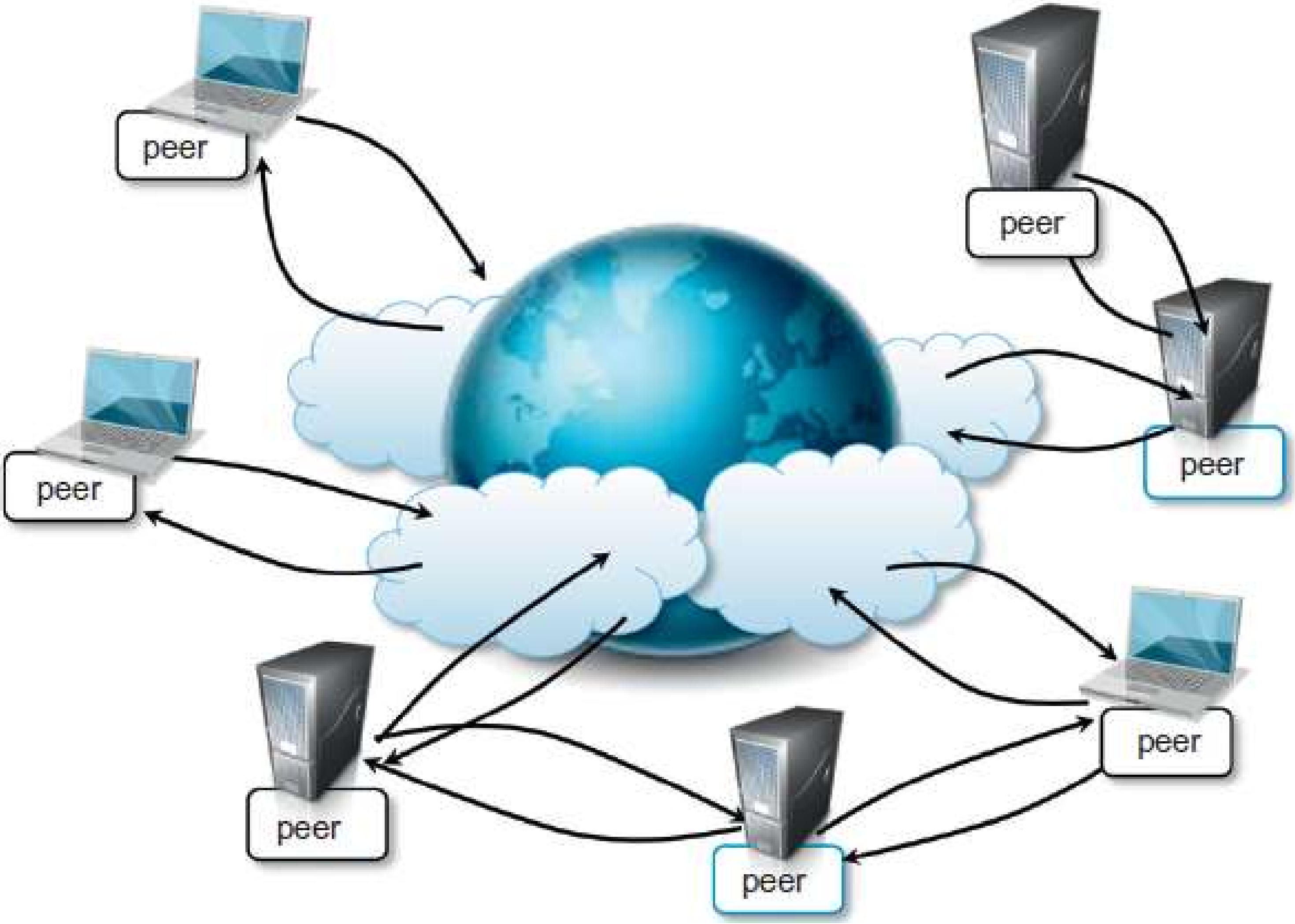
- With respect to the client/ server model that partitions the responsibilities of the IPC between server and clients, the peer-to- peer model attributes the same responsibilities to each component.
- Therefore, this model is quite suitable for highly decentralized architecture, which can scale better along the dimension of the number of peers.
- The disadvantage of this approach is that the management of the implementation of algorithms is more complex than in the client/server model

Peer-to-peer

- Examples: file-sharing applications such as Gnutella, Bit Torrent, and Kazaa.
- Despite the differences among these networks in coordinating nodes and sharing information on the files and their locations, all of them provide a user client that is at the same time a server providing files to other peers and a client downloading files from other peers.

Peer-to-peer

- To address an incredibly large number ofpeers, different architectures have been designed that divert slightly from the peer-to-peer model.
- For example, in Kazaa not all the peers have the same role, and some of them are used to group the accessibility information of a group of peers.
- Another interesting example of peer-to-peer architecture is represented by the Skype network.



peer
peer
peer
peer
peer
peer
peer
peer