Unit - IV

Parallel and Distributed Computing

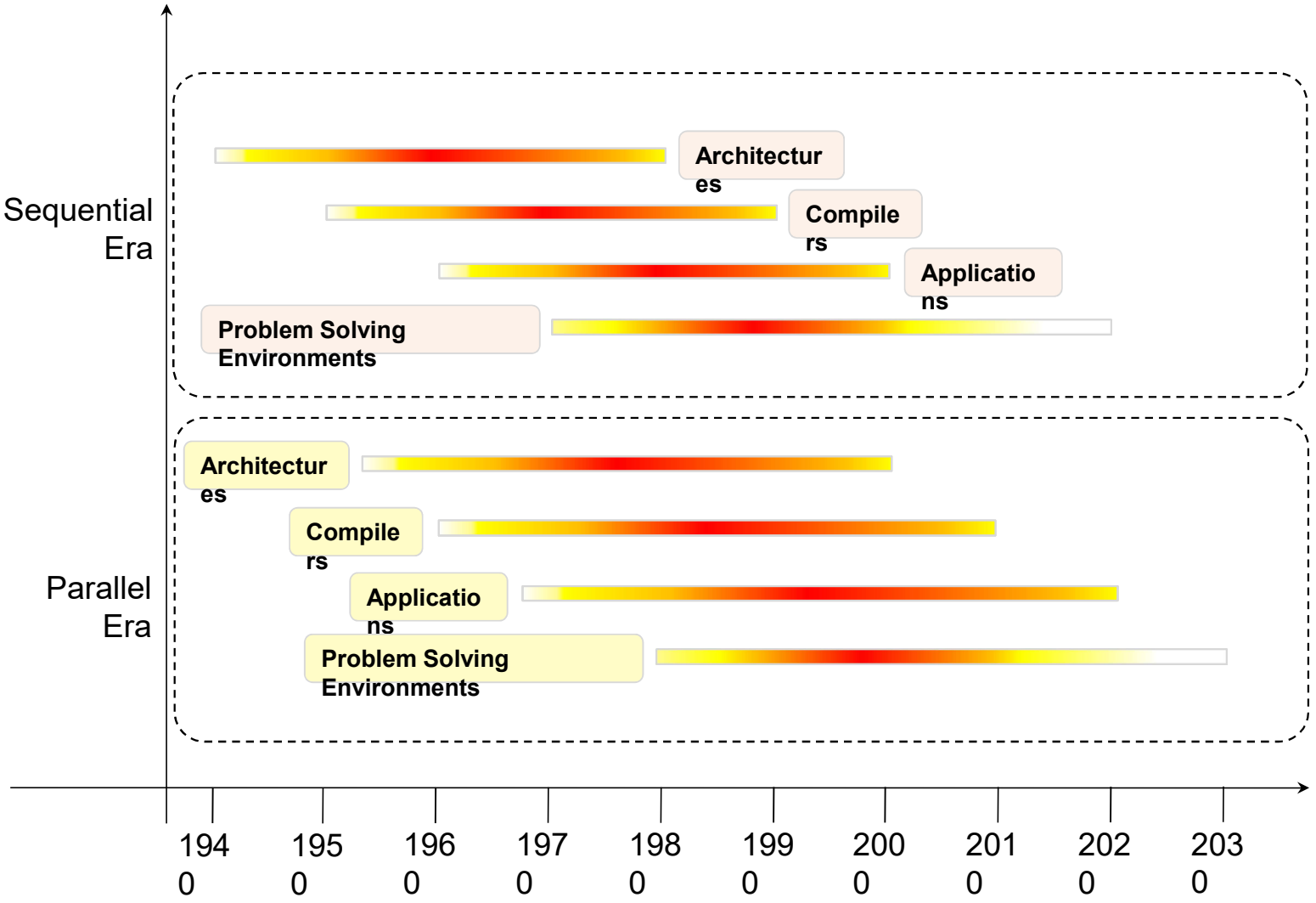
Eras of Computing, Parallel vs. Distributed Computing, Elements of Parallel Computing, What is Parallel Processing?, Hardware Architectures for Parallel Processing, Approaches to Parallel Programming, Levels of Parallelism, Laws of Caution, Elements of Distributed Computing, General Concepts and Definitions, Components of a Distributed System, Architectural Styles for Distributed Computing, Models for Inter-Process Communication, Technologies for Distributed Computing, Remote Procedure Call, Distributed Object Frameworks, Service Oriented Computing

Principles of Parallel and Distributed Computing

- Cloud computing is a new technological trend that supports better utilization of IT infrastructures, services, and applications.
- It adopts a service delivery model based on a pay-per-use approach, in which users do not own infrastructure, platform, or applications but use them for the time they need them.

Eras of computing

- The two fundamental and dominant models of computing are sequential and parallel.
- The sequential computing era began in the 1940s; the parallel (and distributed) computing era followed it within a decade.


Sequential Era
Architectures
Compilers
Applications
Problem Solving Environments
Parallel Era
Architectures
Compilers
Applications
Problem Solving Environments
1940
1950
1960
1970
1980
1990
2000
2010
2020
2030

Parallel processing

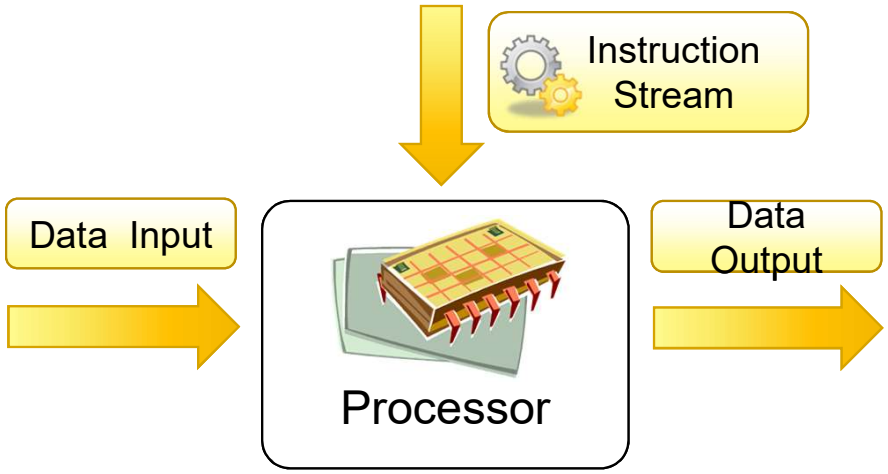
- Processing of multiple tasks simultaneously on multiple processors is called parallel processing.
- The parallel program consists of multiple active processes (tasks) simultaneously solving a given problem.
- A given task is divided into multiple subtasks using a divide-and-conquer technique, and each 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.

Hardware architectures for parallel processing

- Single-instruction, single-data (SISD) systems
- Single-instruction, multiple-data (SIMD) systems
- Multiple-instruction, single-data (MISD) systems
- Multiple-instruction, multiple-data (MIMD) systems

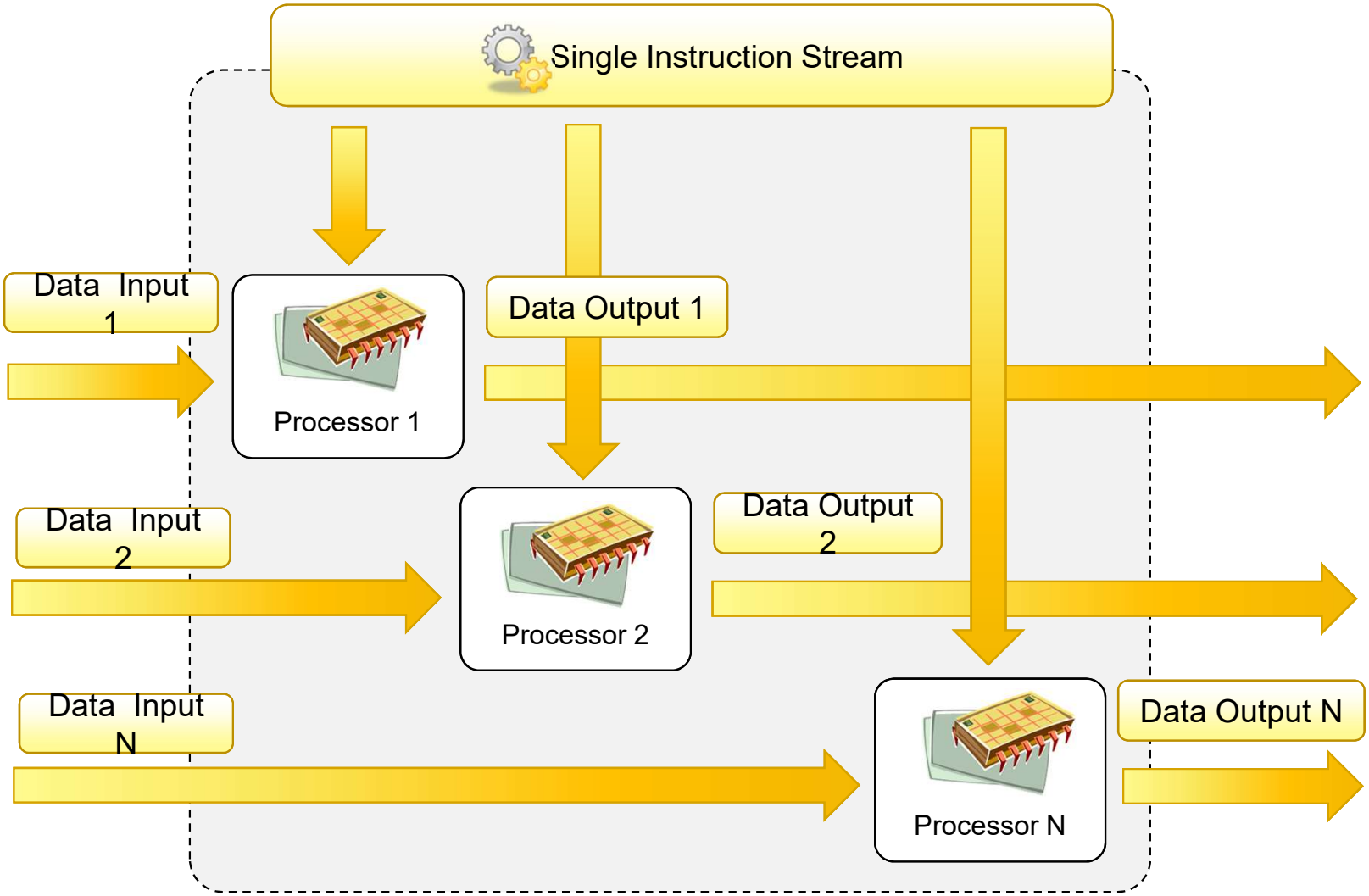
Single-instruction, single-data (SISD) systems

- An SISD computing system is a uniprocessor machine capable of executing a single instruction, which operates on a single data stream



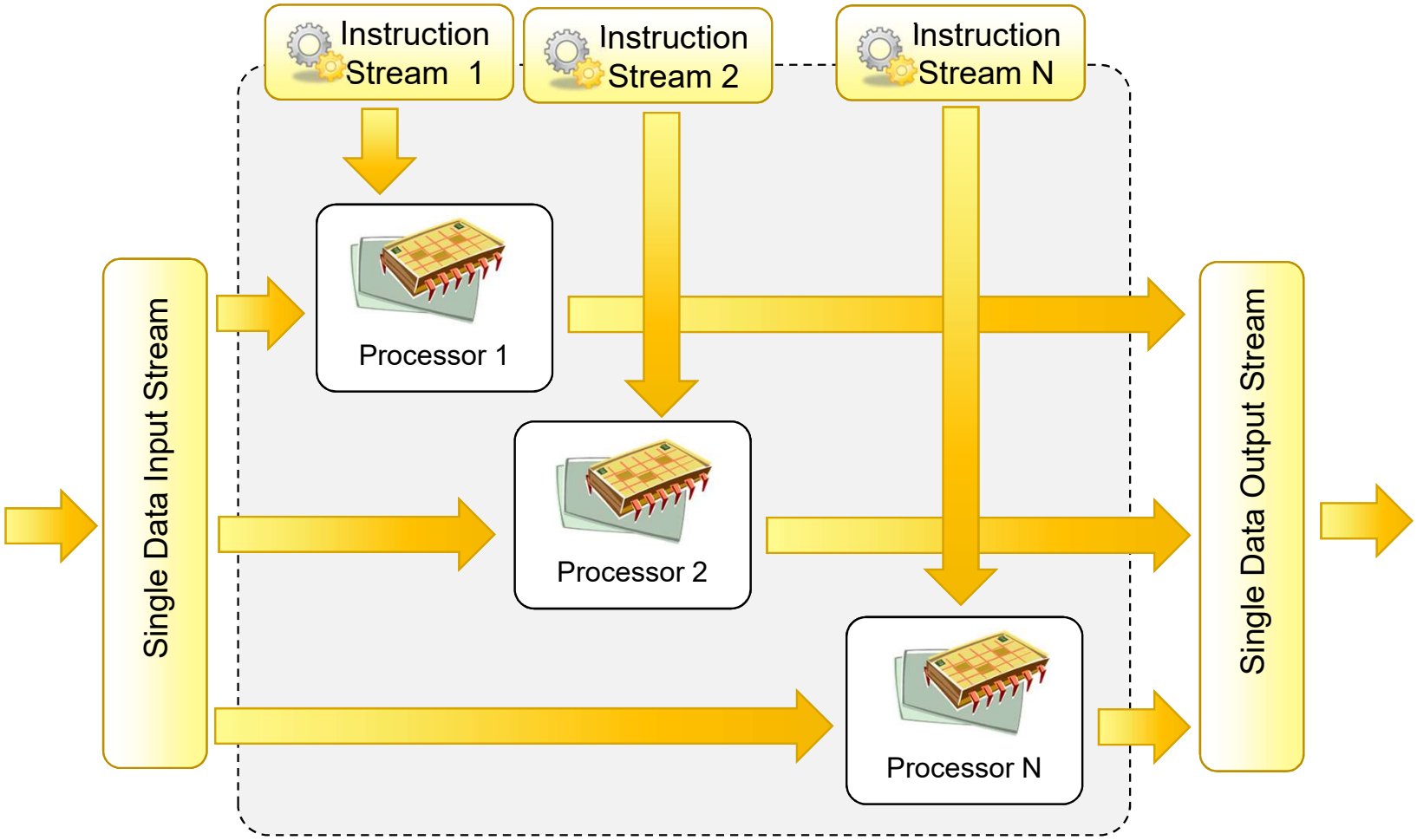
Single-instruction, multiple-data (SIMD) systems

- An SIMD computing system is a multiprocessor machine capable of executing the same instruction on all the CPUs but operating on different data streams


Single Instruction Stream
Data Input 1
Processor 1
Data Output 1
Data Input 2
Processor 2
Data Output 2
Data Input N
Processor N
Data Output N

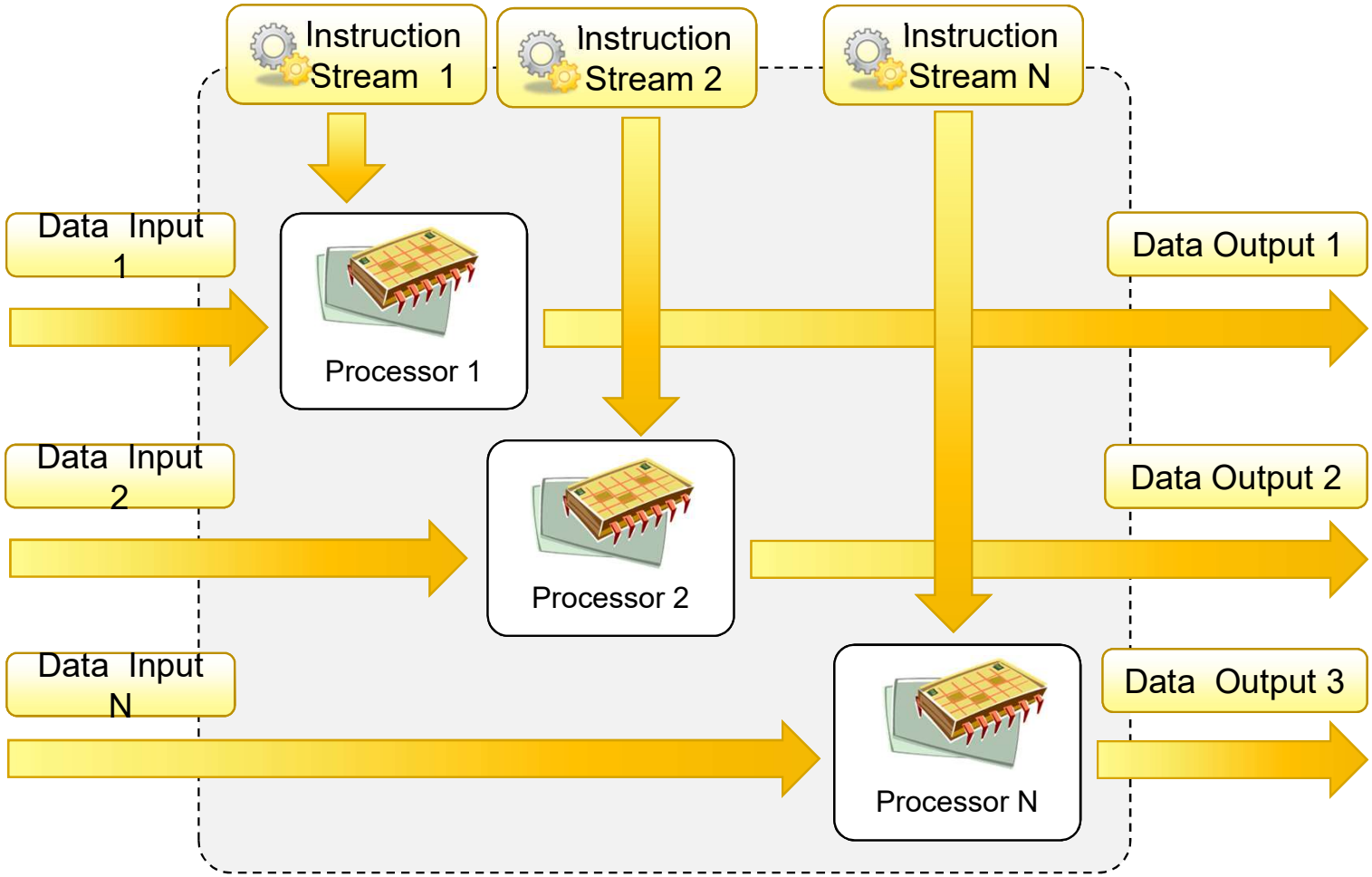
Multiple-instruction, single-data (MISD) systems

- An MISD computing system is a multiprocessor machine capable of executing different instructions on different processing elements PEs but all of them operating on the same data set.


Instruction Stream 1
Instruction Stream 2
Instruction Stream N
Single Data Input Stream
Processor 1
Processor 2
Processor N
Single Data Output Stream

Multiple-instruction, multiple-data (MIMD) systems

- An MIMD computing system is a multiprocessor machine capable of executing multiple instructions on multiple data sets.
- MIMD machines are broadly categorized into shared-memory MIMD and distributed-memory MIMD based on the way PEs are coupled to the main memory.

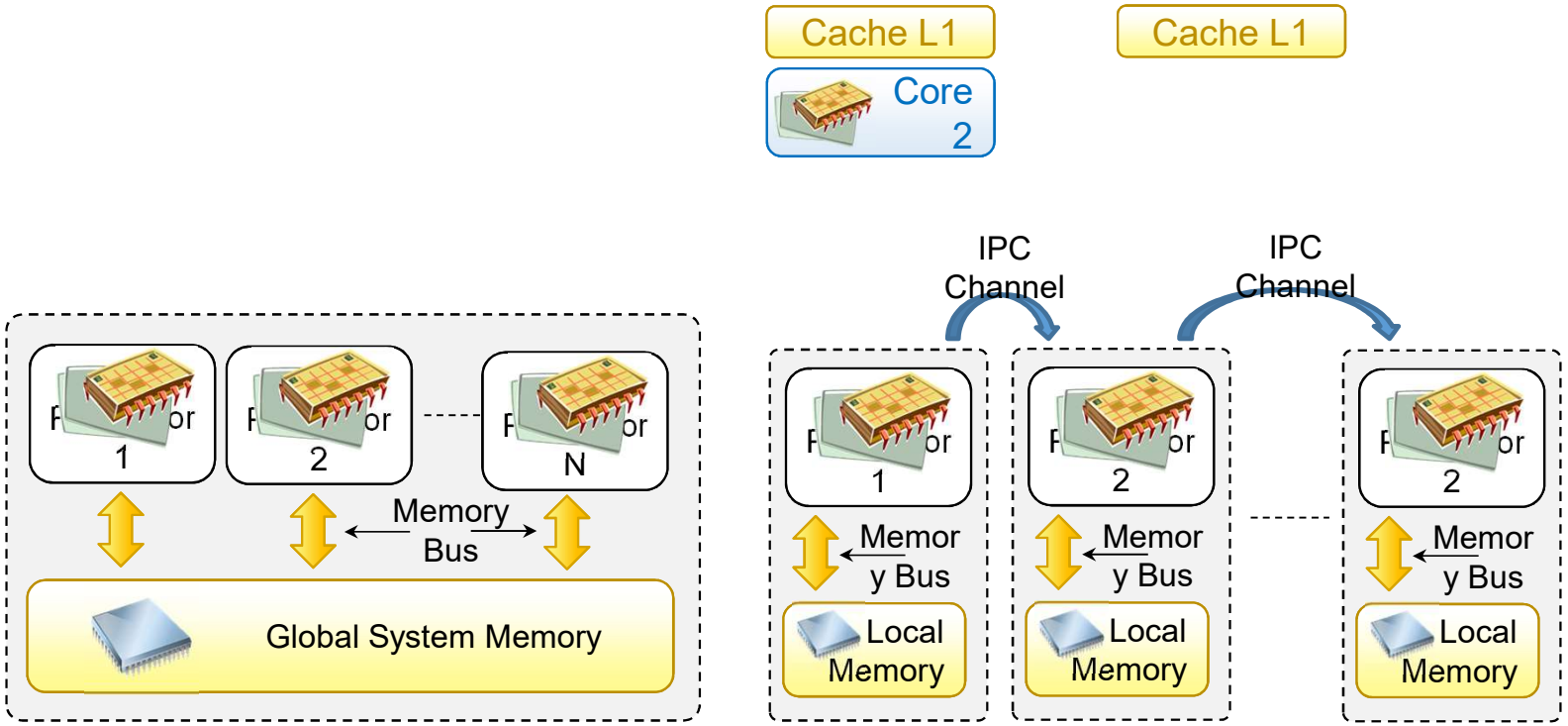

Instruction Stream 1
Instruction Stream 2
Instruction Stream N
Data Input 1
Data Input 2
Data Input N
Processor 1
Processor 2
Processor N
Data Output 1
Data Output 2
Data Output 3

Shared memory MIMD machines

- In the shared memory MIMD model, all the PE's are connected to a single global memory and they all have access to it.
- Systems based on this model are also called tightly coupled multiprocessor systems.

Distributed memory MIMD machines

- In the distributed memory MIMD model, all PE's have a local memory.
- Systems based on this model are also called loosely coupled multiprocessor systems.



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

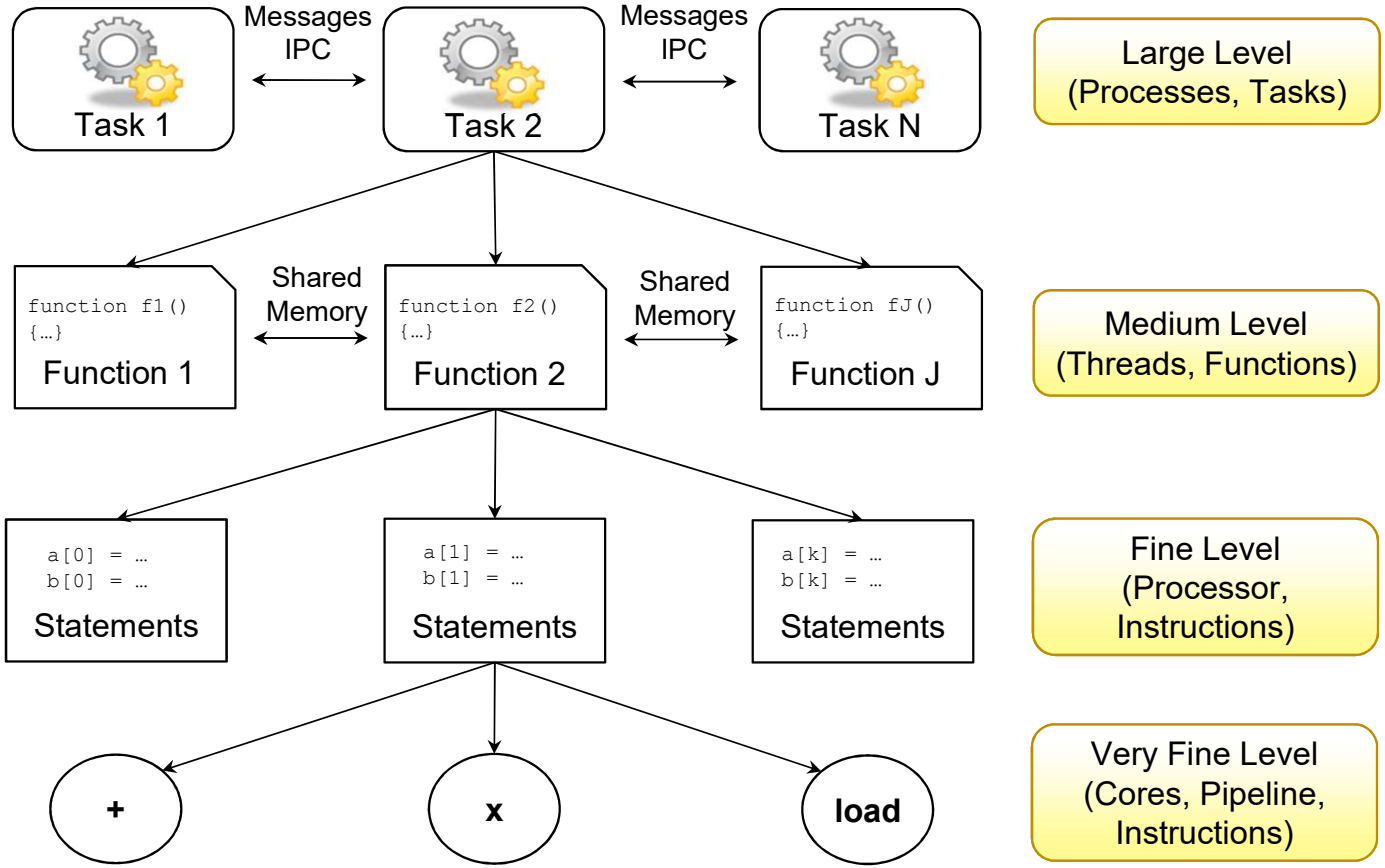
Approaches to parallel programming

- A sequential program is one that runs on a single processor and has a single line of control.
- 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

- 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 to 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.

Levels of parallelism

- Levels of parallelism are decided based on the lumps of code (grain size) that can be a potential candidate for parallelism.
- Parallelism within an application can be detected at several levels:
  - Large grain (or task level)
  - Medium grain (or control level)
  - Fine grain (data level)
  - Very fine grain (multiple-instruction issue)


Task 1
Messages
IPC
Task 2
Messages
IPC
Task N
Large Level
(Processes, Tasks)
function f1()
{…}
Function 1
Shared
Memory
function f2()
{…}
Function 2
Shared
Memory
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)

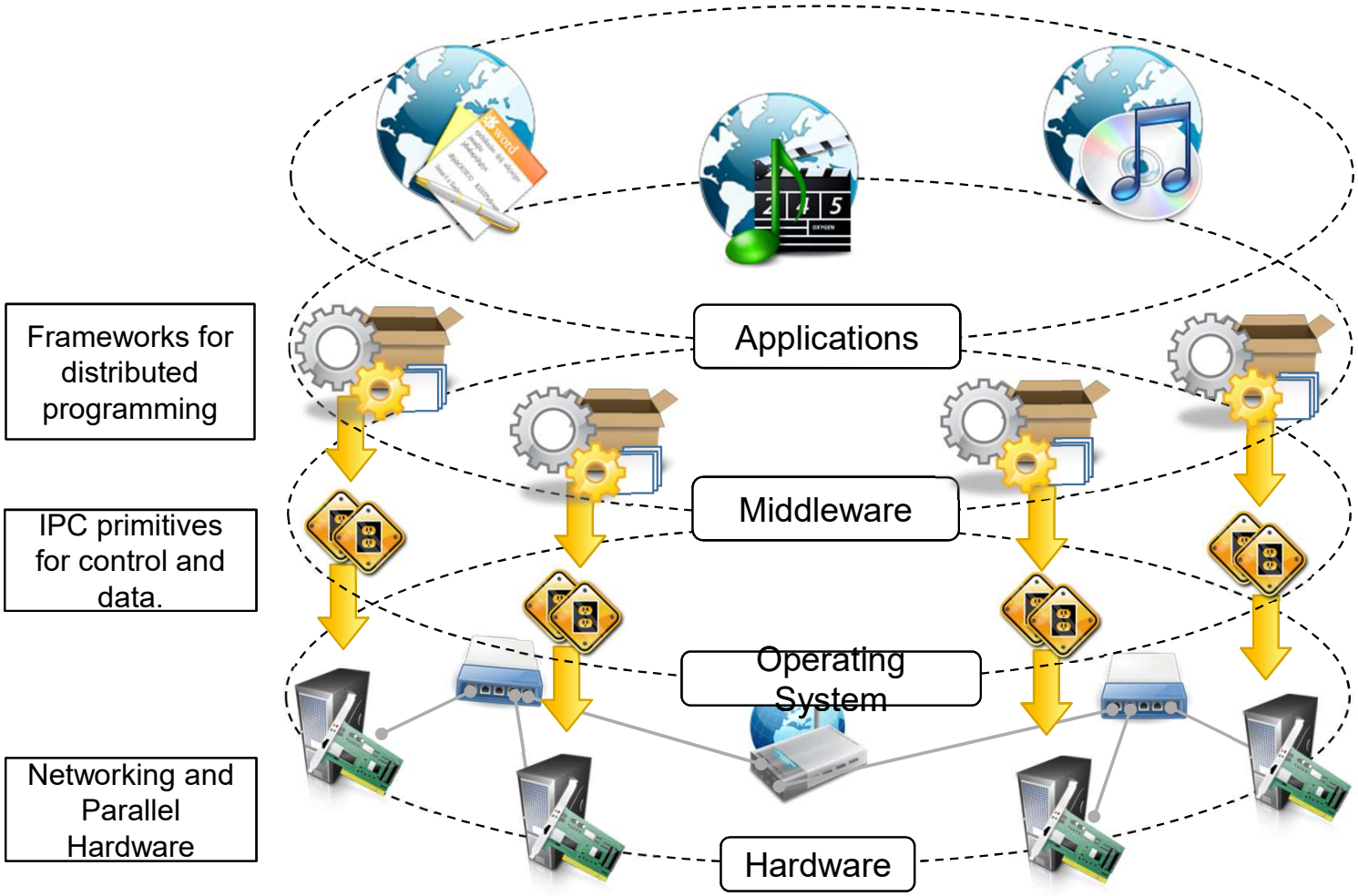
Distributed computing

- A distributed system is a collection of independent computers that appears to its users as a single coherent system.
- A distributed system is one in which components located at networked computers communicate and coordinate their actions only by passing messages.

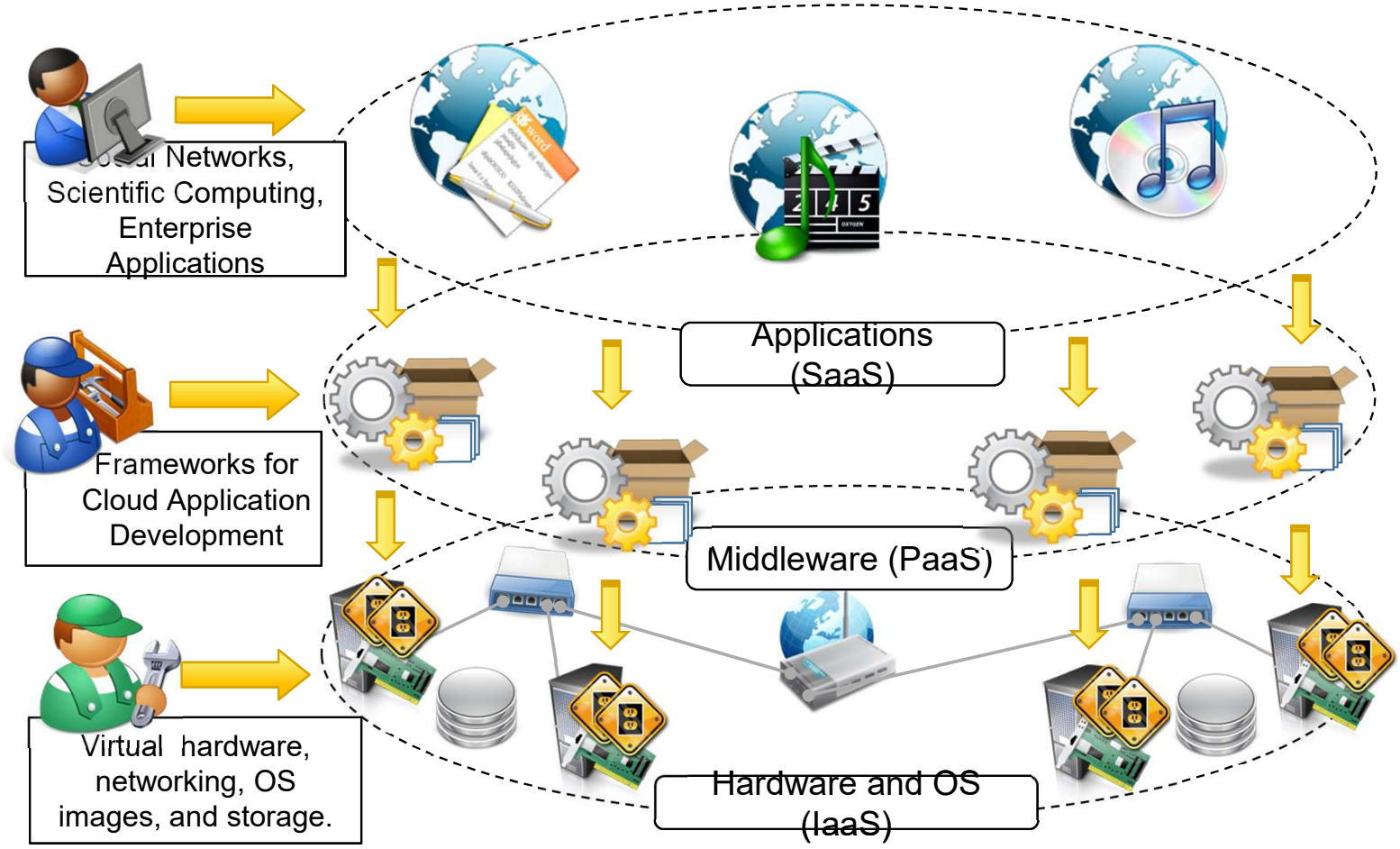
Components of a distributed system

- 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.

A layered view of a distributed system



A cloud computing distributed system



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 within the community of software engineers and developers as to how to structure the relations of components within an application and understand the internal organization of software applications.
- 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.

Software architectural styles

- Software architectural styles are based on the logical arrangement of software components.
- Software 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

- 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-centered architectures.

Repository architectural style

- 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.

Blackboard architectural style

- In blackboard systems, the central data structure is the main trigger for selecting the processes to execute.
- The blackboard architectural style is characterized by three main components:
  - Knowledge sources: These 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.

Data-flow architectures

- Data-flow architectures, it is the availability of data that controls the computation.

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

Pipe-and-Filter Style

- The pipe-and-filter style is a variation of the previous style for expressing the activity of as of 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.
- Data-flow architectures are optimal when the system to be designed embodies a multistage process, which can be clearly identified into a collection of separate components that need to be orchestrated together.

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.

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. The output can either be the product of the rule activation or a set of assertions that holds true for the given input data.
- Rule-based systems are very popular in the field of artificial intelligence.

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.

Call & return architectures

- This category identifies all systems that are organised 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

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

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.

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.

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:** In this architectural style, components are represented by independent processes that leverage IPC facilities for coordination management.
- **Event Systems:** In this architectural style, the components of the system are loosely coupled and connected.

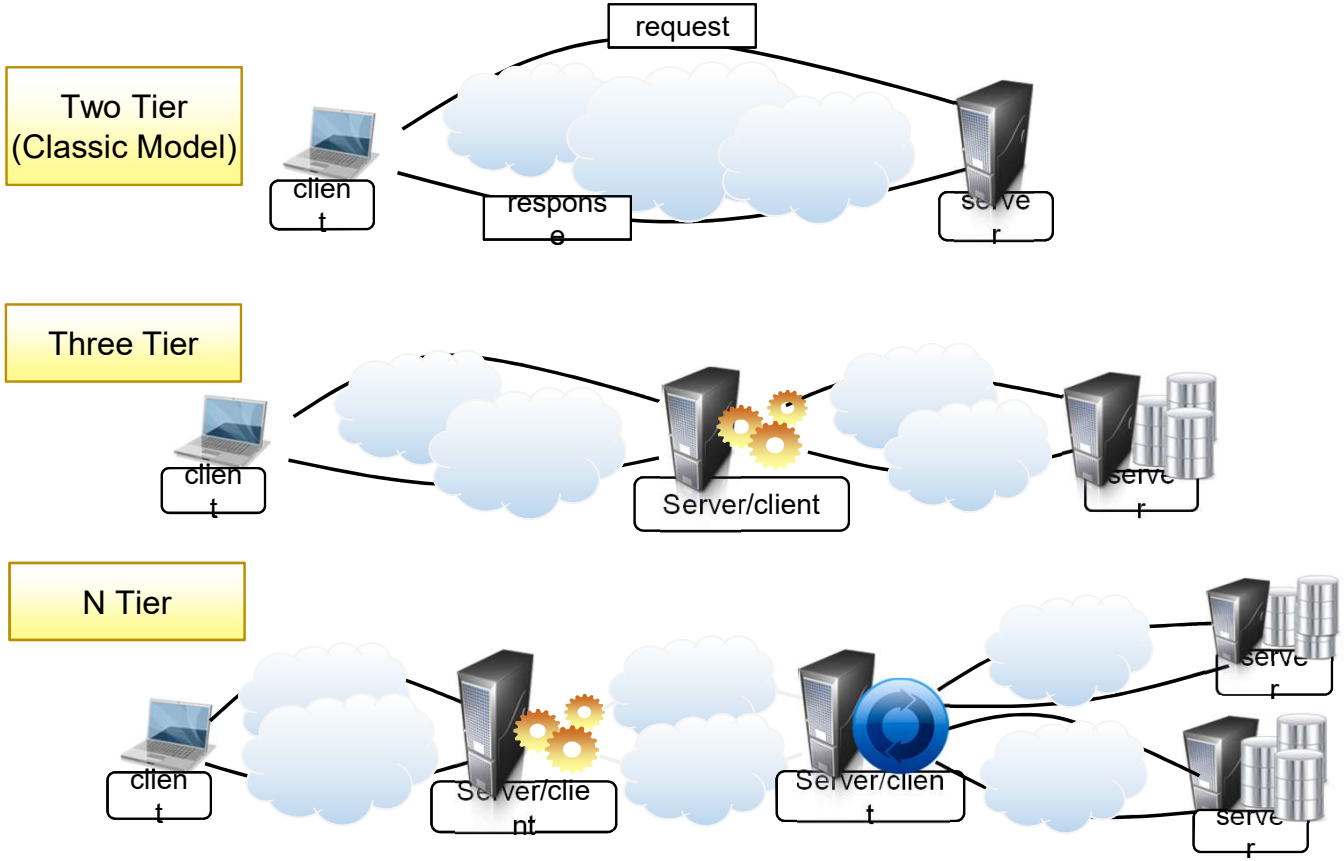
System architectural styles

- System architectural styles cover the physical organization of components and processes over a distributed infrastructure.
- There are two fundamental reference styles: 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.
- The important operations in the client-server paradigm are request, accept (client side),and listen and response (server side).

Client/server architectural styles



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.

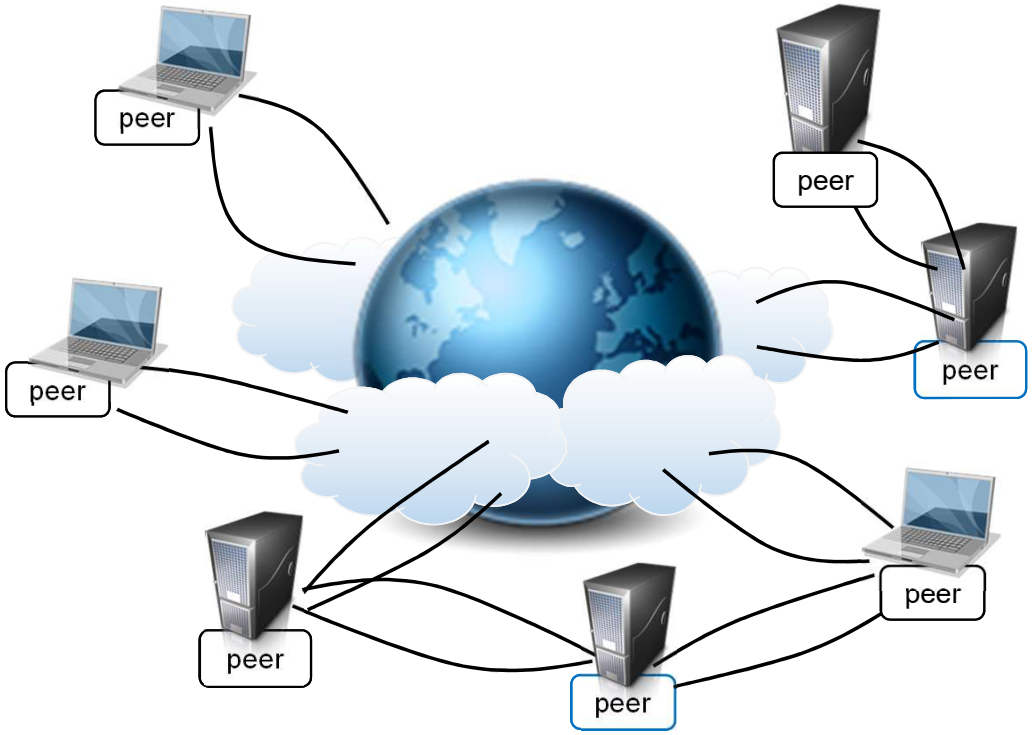
Three-tier architecture/N-tier architecture

- The three-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.
- This model is generally more scalable than the two-tier one because it is possible to distribute the tiers into several computing nodes, thus isolating the performance bottle necks.

Peer-to-peer

- The peer-to-peer model, introduces asymmetric architecture in which all the components , called peers, play the same role and incorporate both client and server capabilities of the client/server model.
- More precisely, each peer acts as a server when it processes requests from other peers and as a client when it issues requests to other peers.
- 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 architectural style



Models for inter process communication

- Distributed systems are composed of a collection of concurrent processes interacting with each other by means of a network connection.
- Therefore, IPC is a fundamental aspect of distributed systems design and implementation.
- IPC is used to either exchange data and information or coordinate the activity of processes.

Message-based communication

- **Message passing:** This paradigm introduces the concept of a message as the main abstraction of the model. The entities exchanging information explicitly encode in the form of a message the data to be exchanged. Examples of this model are the Message-Passing Interface(MPI) and Open MP.
- **Remote procedure call (RPC):** This paradigm extends the concept of procedure call beyond the boundaries of a single process, thus triggering the execution of code in remote processes.
- **Distributed objects:** This is an implementation of the RPC model for the object-oriented paradigm and contextualizes this feature for the remote invocation of methods exposed by objects. Each process registers a set of interfaces that are accessible remotely. Examples of distributed object infrastructures are Common Object Request Broker Architecture(CORBA), Component Object Model (COM, DCOM, and COM1), Java Remote Method Invocation(RMI), and .NET Remoting

- **Distributed agents and active objects:** Programming paradigms based on agents and active objects involve by definition the presence of instances, whether they are agents of objects, despite the existence of requests.
- **Web services:** Web service technology provides an implementation of the RPC concept over HTTP, thus allowing the interaction of components that are developed with different technologies.

Models for message-based communication

- **Point-to-point message model :** This model organizes the communication among single components. Each message is sent from one component to another, and there is a direct addressing to identify the message receiver.
- **Publish-and-subscribe message model :** This model introduces a different strategy, one that is based on notification among components. There are two major roles: the publisher and the subscriber. The publish-and-subscribe model is very suitable for implementing systems based on the one- to-many communication model and simplifies the implementation of indirect communication pat- terns.

- **Request-reply message model :** The request-reply message model identifies all communication models in which, for each message sent by a process, there is a reply. Publish- and-subscribe models are less likely to be based on request-reply since they rely on notifications.

Technologies for distributed computing

- Remote procedure call (RPC),
- Distributed object frameworks,
- Service-oriented computing.

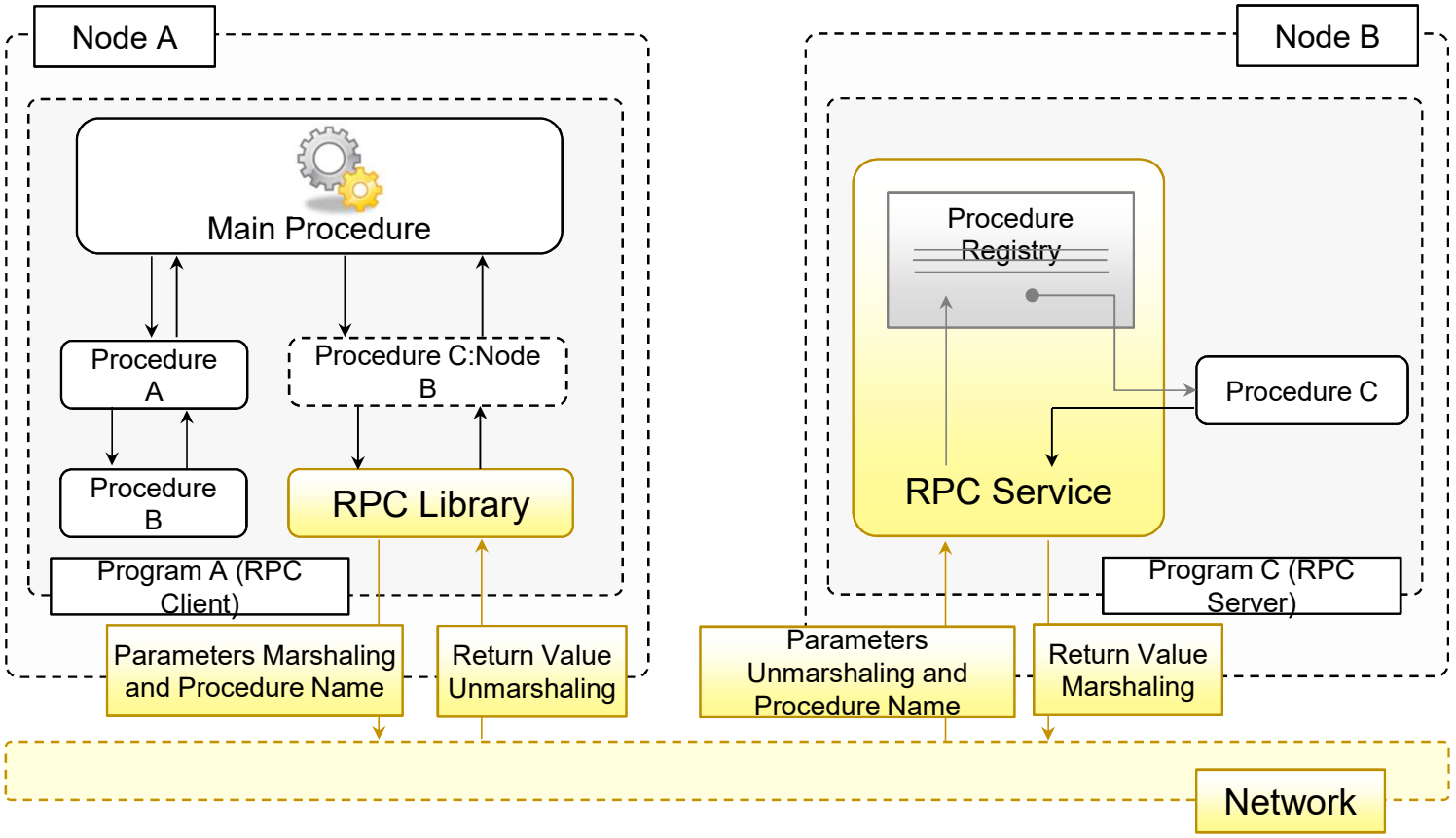
Remote procedure call

- RPC is the fundamental abstraction enabling the execution of procedures on client's request.
- RPC allows extending the concept of a procedure call beyond the boundaries of a process and a single memory address space.
- The called procedure and calling procedure may be on the same system or they may be on different systems in a network.
- An important aspect of RPC is marshaling, which identifies the process of converting parameter and return values into a form that is more suitable to be transported over a network through a sequence of bytes. The term unmarshaling refers to the opposite procedure.

Developing a system leveraging RPC for IPC consists of the following steps:

- Design and implementation of the server procedures that will be exposed for remote invocation.
- Registration of remote procedures with the RPC server on the node where they will be made available.
- Design and implementation of the client code that invokes the remote procedure(s).

The RPC reference model



Distributed object frameworks

- Distributed object frameworks extend object-oriented programming systems by allowing objects to be distributed across a heterogeneous network and provide facilities so that they can coherently act as though they were in the same address space.
- Distributed object frameworks give the illusion of interaction with a local instance while invoking remote methods. This is done by a mechanism called a proxy skeleton.

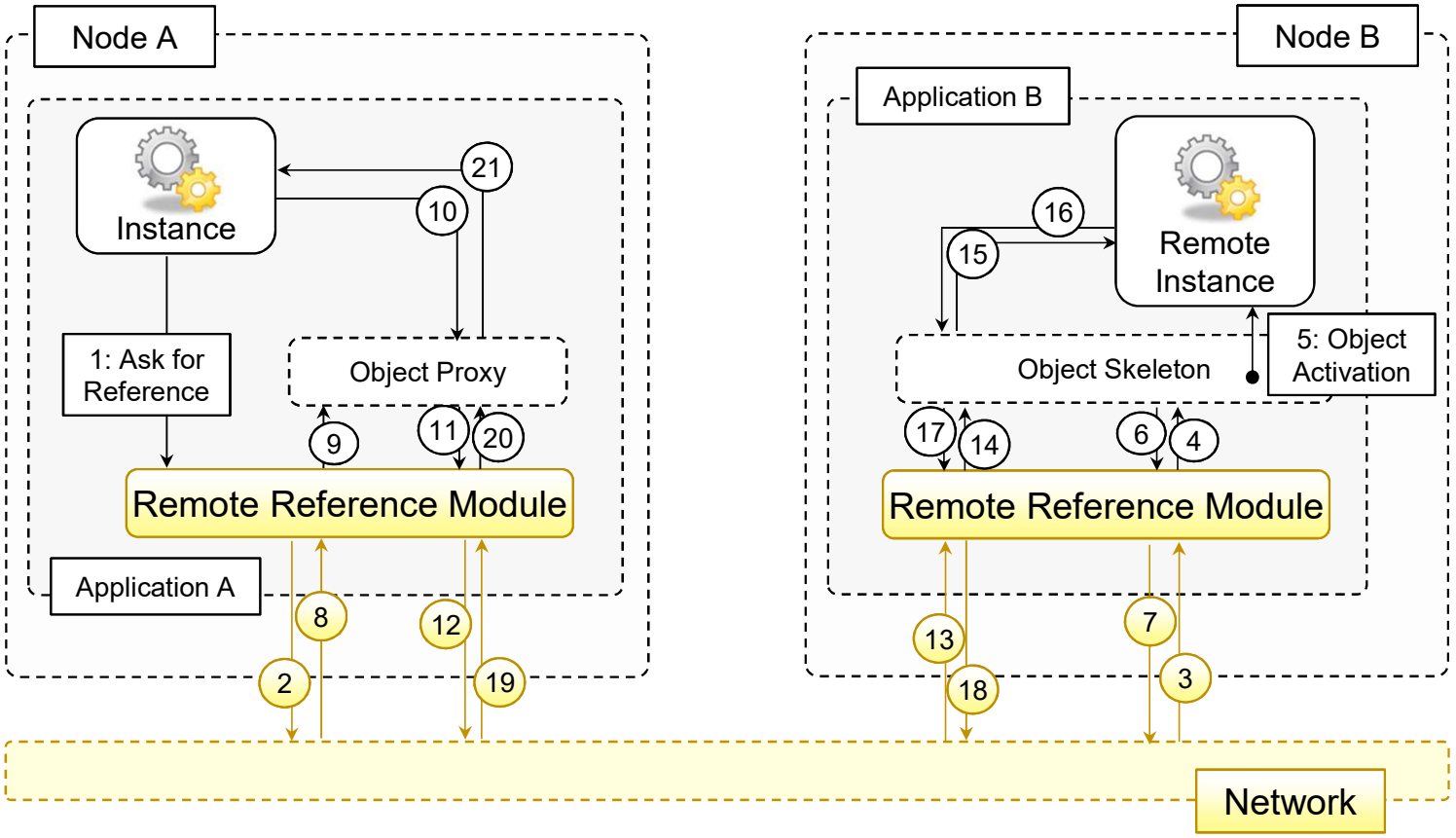
The common interaction pattern is the following:

1.The server process maintains a registry of active objects that are made available to other processes. According to the specific implementation, active objects can be published using interface definitions or class definitions.

2.The client process, by using a given addressing scheme, obtains a reference to the active remote object. This reference is represented by a pointer to an instance that is of a shared type of interface and class definition.

3.The client process invokes the methods on the active object by calling them through the reference previously obtained. Parameters and return values are marshaled as happens in the case of RPC.

The distributed object programming model



Examples of distributed object frameworks

- **Common object request broker architecture(CORBA):** CORBA is a specification introduced by the Object Management Group(OMG) for providing cross- platform and cross-l.
- **Distributed component object model (DCOM/COM1):** DCOM, later integrated and evolved into COM1, is the solution provided by Microsoft for distributed object programming before the introduction of .NET technology. DCOM introduces a set of features allowing the use of COM components beyond the process boundaries.

- **Java remote method invocation (RMI):** Java RMI is a standard technology provided by Java for enabling RPC among distributed Java objects. RMI defines an infrastructure allowing the invocation of methods on objects that are located on different Java Virtual Machines (JVMs) residing either on the local node or on a remote one.
- **.NET remoting :** Remoting is the technology allowing for IPC among .NET applications. It provides developers with a uniform platform for accessing remote objects from within any application developed in any of the languages supported by.NET.

Service-oriented computing

- Service-oriented computing organizes distributed systems in terms of services, which represent the major abstraction for building systems.
- Service orientation expresses applications and software systems as aggregations of services that are coordinated within a service-oriented architecture(SOA).

- Boundaries are explicit. A service-oriented application is generally composed of services that are spread across different domains, trust authorities, and execution environments.
- Services are autonomous. Services are components that exist to offer functionality and are aggregated and coordinated to build more complex system. They are not designed to be part of a specific system, but they can be integrated in several software systems, even at the same time

- Services share schema and contracts, not class or interface definitions. Services are not expressed in terms of classes or interfaces, as happens in object-oriented systems, but they define them selves in terms of schemas and contracts.
- Services compatibility is determined based on policy.
- Service orientation separates structural compatibility from semantic compatibility.
- Structural compatibility is based on contracts and schema and can be validated or enforced by machine-based techniques.

Service-oriented architecture (SOA)

- SOA is an architectural style supporting service orientation.
- It organizes a software system into a collection of interacting services.
- SOA encompasses a set of design principles that structure system development and provide means for integrating components in to a coherent and decentralized system.

Features

- **Standardized service contract:** Services adhere to a given communication agreement, which is specified through one or more service description documents.
- **Loose coupling:** Services are designed as self-contained components, maintain relationships that minimize dependencies on other services, and only require being aware of each other.
- **Abstraction:** A service is completely defined by service contracts and description documents. They hide their logic, which is encapsulated within their implementation.

- **Reusability:** Designed as components, services can be reused more effectively, thus reducing development time and the associated costs. Reusability allows for a more agile design and cost-effective system implementation and deployment
- **Autonomy:** Services have control over the logic they encapsulate and, from a service consumer point of view, there is no need to know about their implementation
- **Lack of state:** By providing a state less interaction pattern (at least in principle), services increase the chance of being reused and aggregated, especially in a scenario in which a single service is used by multiple consumers that belong to different administrative and business domains.

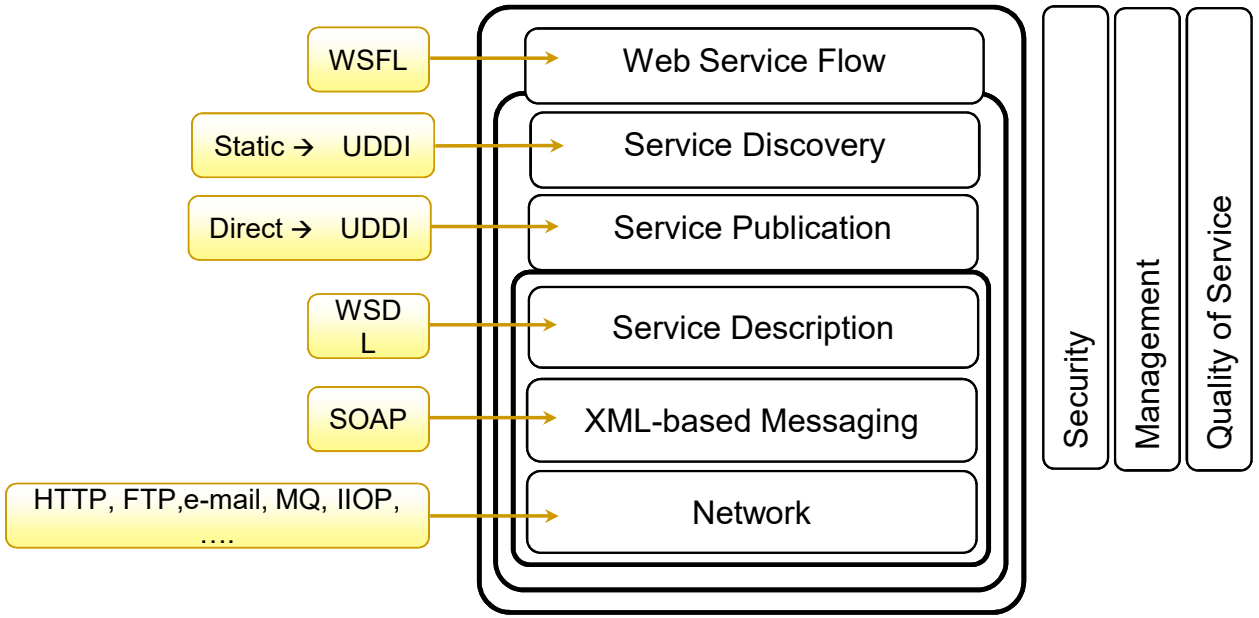
Discoverability: Services are defined by description documents that constitute supplemental metadata through which they can be effectively discovered. Service discovery provides an effective means for utilizing third-party resources.

Composability: Using services as building blocks, sophisticated and complex operations can be implemented. Service orchestration and choreography provide a solid support for composing services and achieving business goals.

Web services

- Web services are the prominent technology for implementing SOA systems and applications. They leverage Internet technologies and standards for building distributed systems.
- The concept behind a Web service is very simple.
- Using as a basis the object-oriented abstraction, a Web service exposes a set of operations that can be invoked by leveraging Internet-based protocols.
- System architects develop a Web service with their technology of choice and deploy it in compatible Web or application servers.

WS technology Stack



- Simple Object Access Protocol (SOAP), an XML-based language for exchanging structured information in a platform-independent manner, constitutes the protocol used for Web service method invocation.

SOAP Messages

```
POST /InStock HTTP/1.1
Host: www.stocks.com
Content-Type: application/soap+xml; charset=utf-8
Content-Length: <Size>

<?xml version="1.0">
<soap:Envelope xmlns:soap="http//www.w3.org/2001/12/soap-envelope"
 soap:encondingStyle="http//www.w3.org/2001/12/soap-encoding">
<soap:Header></soap:Header>
<soap:Body xmlns:m=http://www.stocks.org/stock>
 <m:GetStockPrice>
  <m:StockName>IBM<m:StockName>
 </m:GetStockPrice>
</soap:Body>
</soap:Envelope>
```

SOAP Messages

```
POST /InStock HTTP/1.1
Host: www.stocks.com
Content-Type: application/soap+xml; charset=utf-8
Content-Length: <Size>

<?xml version="1.0">
<soap:Envelope xmlns:soap="http//www.w3.org/2001/12/soap-envelope"
 soap:encondingStyle="http//www.w3.org/2001/12/soap-encoding">
 <soap:Header></soap:Header>
 <soap:Body xmlns:m=http://www.stocks.org/stock>
  <m:GetStockPriceResponse>
   <m:Price>34.5<m:Price>
  </m:GetStockPriceResponse>
 </soap:Body>
</soap:Envelope>
```

Envelope

Header: Metadata & Assertions

Body: Execution Result

Service orientation and cloud computing

- Web services and Web 2.0-related technologies constitute a fundamental building block for cloud computing systems and applications.
- Web 2.0 applications are the front end of cloud computing systems, which deliver services either via Web service or provide a profitable interaction with AJAX-based clients.

Assignment 4

- Evaluate Parallel vs. Distributed Computing with suitable example.
- Evaluate Elements of Parallel Computing with suitable example.
- Evaluate Hardware Architectures for Parallel Processing with suitable example.
- Evaluate Approaches to Parallel Programming with suitable example.
- Evaluate Levels of Parallelism with suitable example.
- Evaluate Laws of Caution with suitable example.
- Evaluate Elements of Distributed Computing with suitable example.
- Evaluate Architectural Styles for Distributed Computing with suitable example.
- Evaluate Models for Inter-Process Communication with suitable example.
- Evaluate Technologies for Distributed Computing with suitable example.
- Evaluate Remote Procedure Call with suitable example.
- Evaluate Distributed Object Frameworks with suitable example.
- Evaluate Service Oriented Computing with suitable example.

Submission link: https://forms.gle/f8vGUapeDMowSL7n9

Presentation IV (Choose any one topic)

- Case study on Parallel Computing
- Case study on Distributed

Submission link: https://forms.gle/f8vGUapeDMowSL7n9

Lab 4

To demonstrate the Virtual Machine Migration from one node to the other.

Submission link: https://forms.gle/f8vGUapeDMowSL7n9