Introduction to Cloud Computing

Dr. Prasenjit Chanak
Assistant Professor

Department of Computer Science and Engineering
Indian Institute of Technology (BHU), Varanasi-221005

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
- The figure provides an overview of the different layers that are involved in providing the services of a distributed system

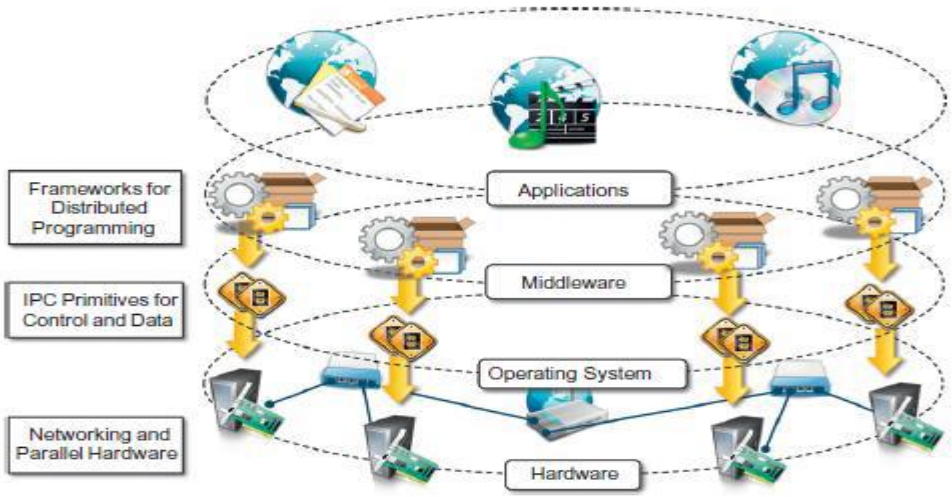


FIGURE 2.10
A layered view of a distributed system.

Components of Distributed System

- 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

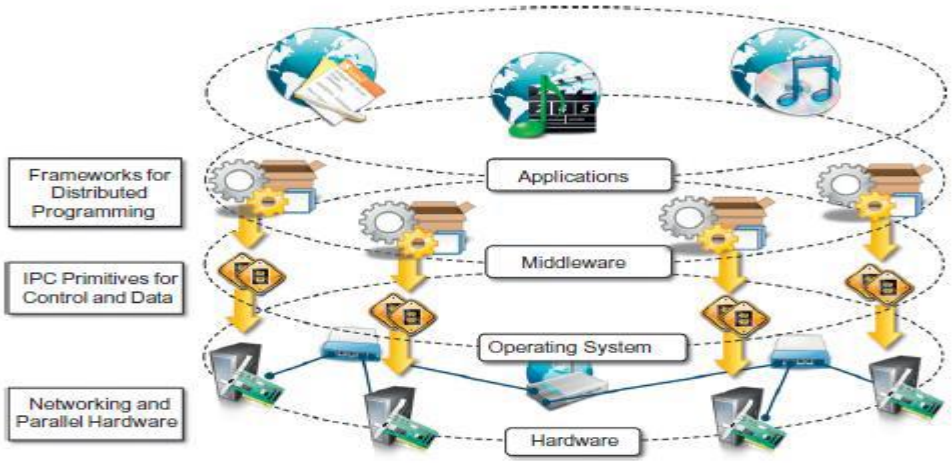


FIGURE 2.10
A layered view of a distributed system.

Components in a Distributed System

- ***Workstations:*** Computers used by end-users to perform computing
- ***Server Systems:*** Computers which provide resources and services
- ***Personal Assistance Devices****:* Handheld computers connected to the system via a wireless communication link

Common Properties of Distributed Computing

- Fault tolerance
  - When one or some nodes fails, the whole system can still work fine except performance.
  - Need to check the status of each node
- Each node play partial role
  - Each computer has only a limited incomplete view of the system
  - Each computer may know only one part of the input
- Resource sharing
  - Each user can share the computing power and storage resource in the system with other users
- Load sharing
  - Dispatching several tasks to each nodes can help share loading to the whole system.
- Easy to expand
  - We expect to use few time when adding nodes. Hope to spend no time.
- Performance
  - Parallel computing can be considered a subset of distributed computing

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 that, taken together, constitute such an environment
- 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 the classifying the organization of the software systems in general and distributed computing in particular

Architectural Styles for Distributed Computing

- 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
- Software architectural styles are classified as
  - Data Centered
  - Data Flow
  - Virtual Machine
  - Call & Return
  - Independent Components
- System Architecture Styles
  - System architectural styles cover the physical organization of components and processes over a distributed infrastructure
- System architectural styles are classified as
  - Client-Server
  - Peer-to-Peer

Data Centered Architectures

- These architectures identify the data as the fundamental element of the software system, and access to shared data is the core characteristics of the data-centered architectures
- Within the context of distributed and parallel computing systems, integrity of data is 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 component, which operate on the central data
- The ways in which the independent components interact with the central data structure can be very heterogeneous

Data Centered Architectures

- In particular repository based architectures differentiate and specialize further into subcategories according to the choice of control discipline to apply for the shared data structure. Of particular interest are databases and blackboard systems
- In the repository systems, the dynamics of the system is controlled by independent components, which by issuing an operation on the central repository, trigger the selection of specific processes that operate on data

Data Flow Architectures

- **Batch Sequential:** 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 sequence of programs that, for example, pre-filter, analyze, and post process data. It is very common to compose these phases using the batch sequential style.
- **Pipe-and-Filter Style:** It 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

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
- The general interaction flow for systems implementing this pattern is – 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

Call and Return architectures

- This 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 one or more operations
- There are three categories in this
  - Top down Style : developed with imperative programming
  - Object Oriented Style: Object programming models
  - Layered Style: provides the implementation in different levels of abstraction of the system

Client / Server

- The information and the services of interest can be centralized and accessed through a single access point : the server.
- Multiple clients are interested in such services and the server must be appropriately designed to efficiently serve requests coming from different clients

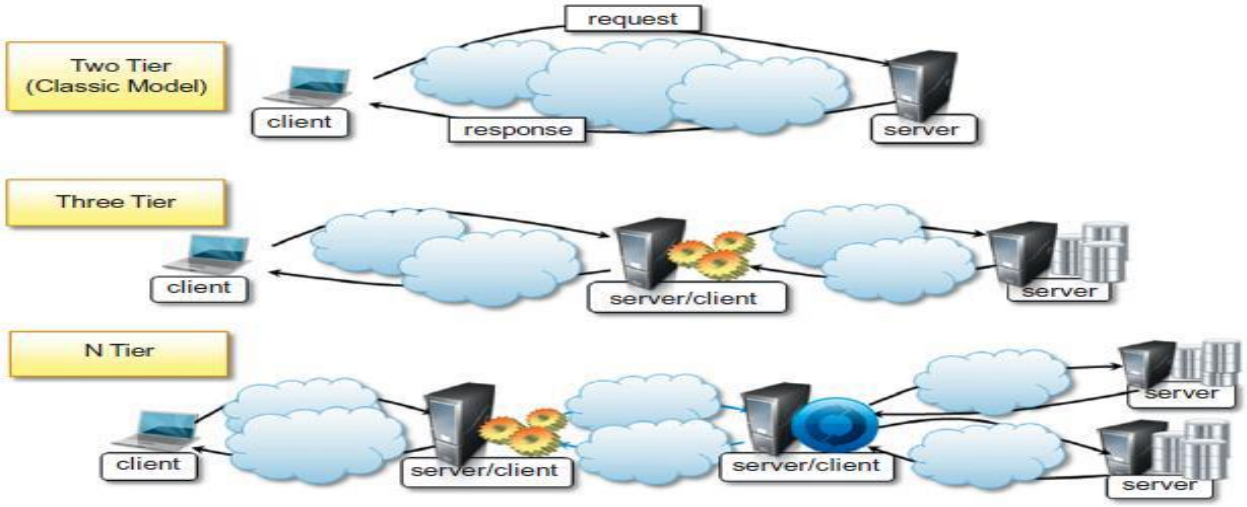


FIGURE 2.12
Client/server architectural styles.

Peer- to – Peer

- Symmetric architectures 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

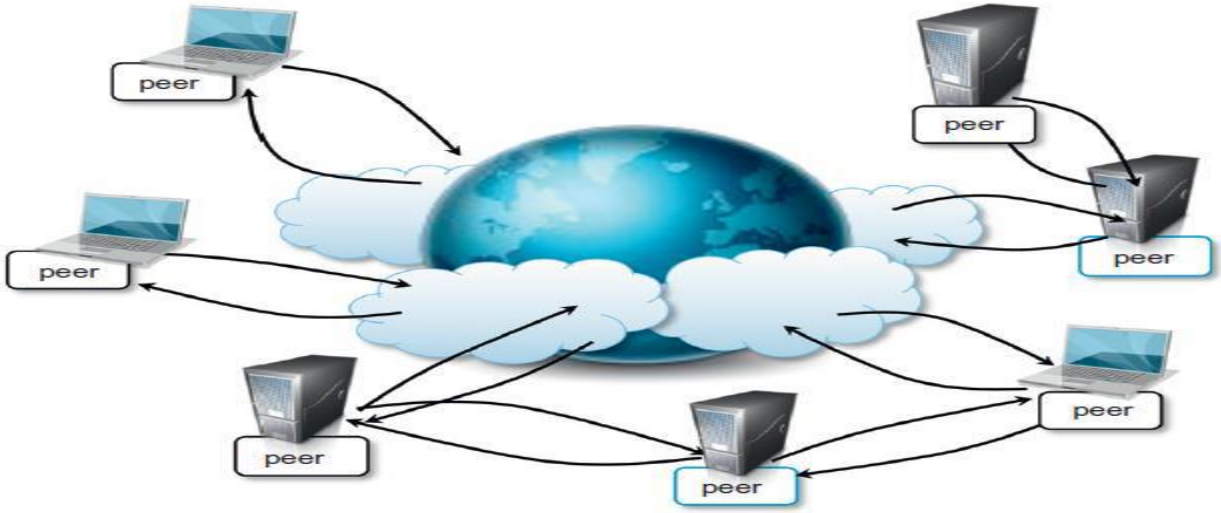


FIGURE 2.13
Peer-to-peer architectural style.

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
- 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
- IPC is what ties together the different components of a distributed system, thus making them act as a single system
- There are several different models in which processes can interact with each other – these maps to different abstractions for IPC
- Among the most relevant that we can mention are shared memory, remote procedure call (RPC), and message passing
- At lower level, IPC is realized through the fundamental tools of network programming
- Sockets are the most popular IPC primitive for implementing communication channels between distributed processes

Message-based communication

- The abstraction of message has played an important role in the evolution of the model and technologies enabling distributed computing
- The definition of distributed computing – is the one in which components located at networked computers communicate and coordinate their actions only by passing messages. The term messages, in this case, identifies any discrete amount of information that is passed from one entity to another. It encompasses any form of data representation that is limited in size and time, where as this is an invocation to a remote procedure or a serialized object instance or a generic message
- The term message-based communication model can be used to refer to any model for IPC

Message-based communication contd…

- Several distributed programming paradigms eventually use message-based communication despite the abstractions that are presented to developers for programming the interactions of distributed components
- Here are some of the most popular and important:
- **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. The structure and the content of a message vary according to the model. Examples of this model are the Message-Passing-Interface (MPI) and openMP

Message-based communication contd…

- **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. Examples of distributed object infrastructures are Common Object Request Broker Architecture (CORBA), Component Object Model (COM, DCOM, and COM+), Java Remote Method Invocation (RMI), and .NET Remoting

Message-based communication contd…

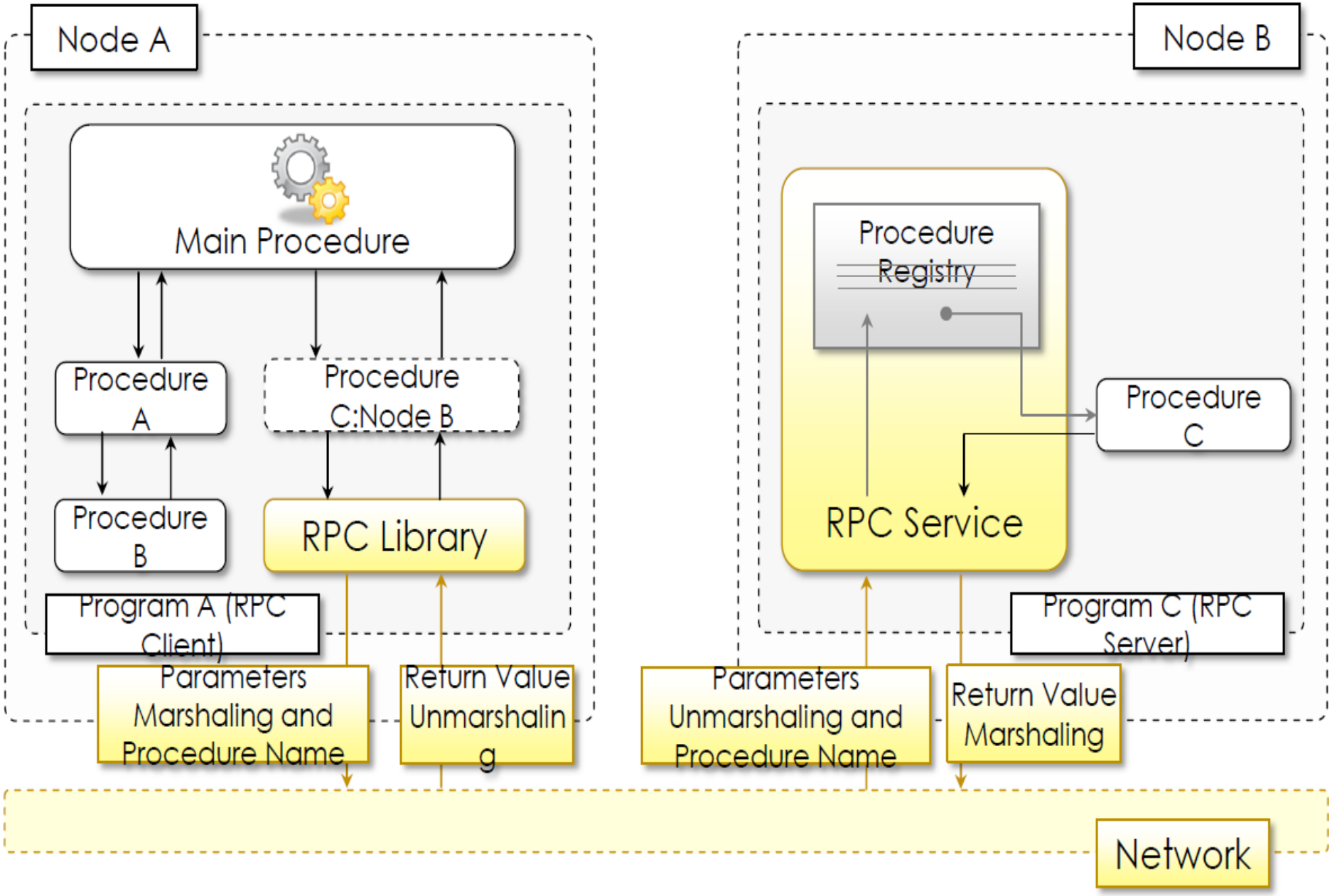
- **Distributed agents and active Objects:** Programming paradigms based on agents and active objects involve by definition the presence of instances, whether they are agents of objects, despite the existence of requests
- **Web Service:** An implementation of the RPC concept over HTTP; thus allowing the interaction of components that are developed with different technologies. A Web service is exposed as a remote object hosted on a Web Server, and method invocation are transformed in HTTP requests, using specific protocols such as Simple Object Access Protocol (SOAP) or Representational State Transfer (REST).

Models for message-based communication

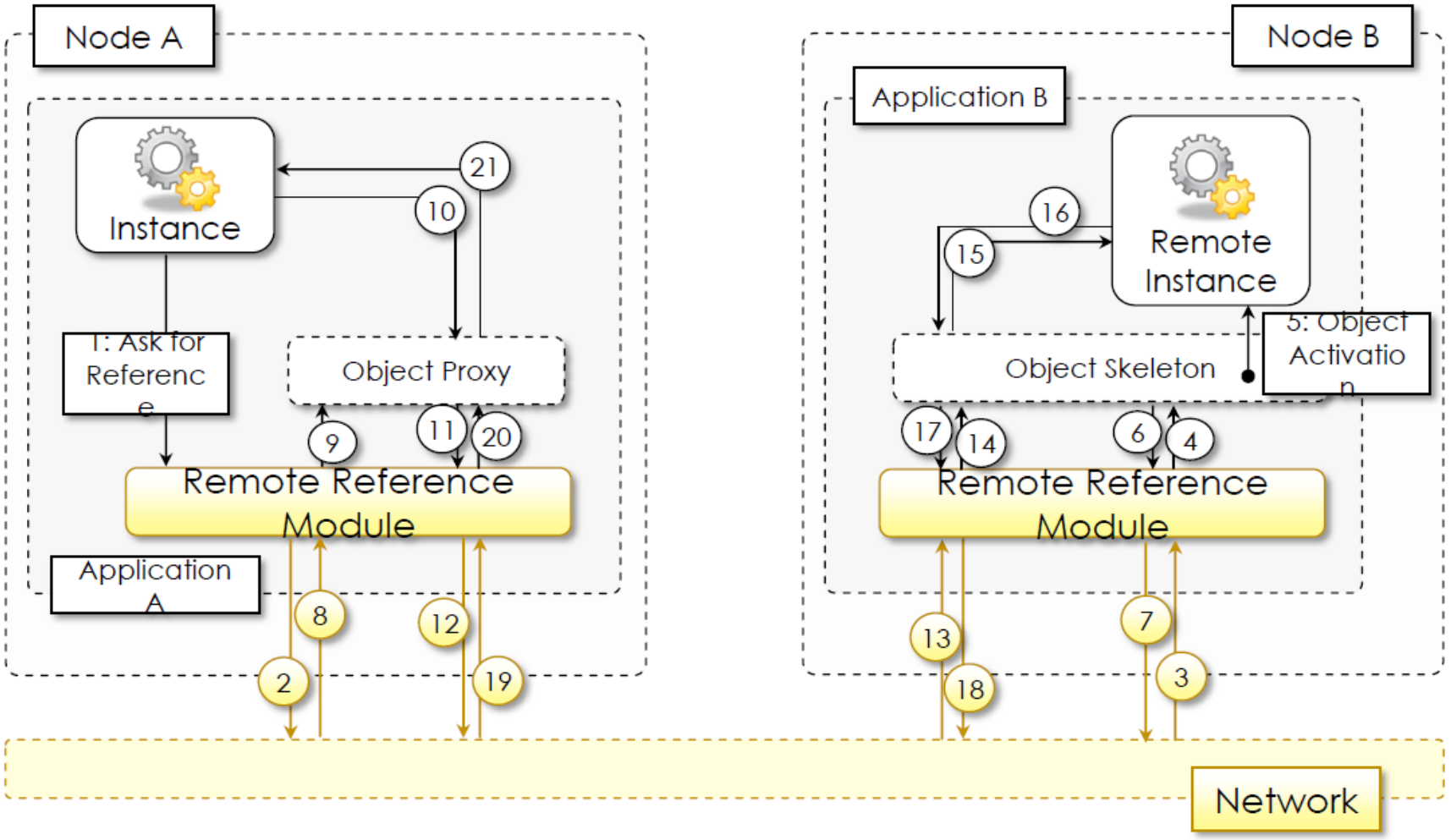
- Point-to-Point message model :
- Publish – and – Subscribe message model
    - Push Strategy
    - Pull Strategy
- Request- reply message model

Technologies for distributed computing

- Remote Procedure Call (RPC)
  - RPC is the fundamental abstraction enabling the execution procedures on clients' 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
  - The important aspect of RPC is marshalling and unmarshalling
- 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 be coherently act as though they were in the same address space


Node A
Main Procedure
Procedure A
Procedure C:Node B
Procedure B
RPC Library
Program A (RPC Client)
Parameters Marshaling and Procedure Name
Return Value Unmarshaling
Node B
Procedure Registry
Procedure C
RPC Service
Program C (RPC Server)
Parameters Unmarshaling and Procedure Name
Return Value Marshaling
Network

Distributed Object



Examples of distributed Object frameworks

- Common Object Request Broker Architecture (CORBA): cross platform and cross language interoperability among distributed components.
- Distributed Component Object Model (DCOM/COM+) : Microsoft technology for distributed object programming before the introduction of .NET technology.
- Java Remote Method Invocation (RMI): technology provided by Java for enabling RPC among distributed Java objects.
- .NET Remoting: IPC among .NET applications, a uniform platform for accessing remote objects from within any application developed in any of the languages supported by .NET.