# Chapter 4

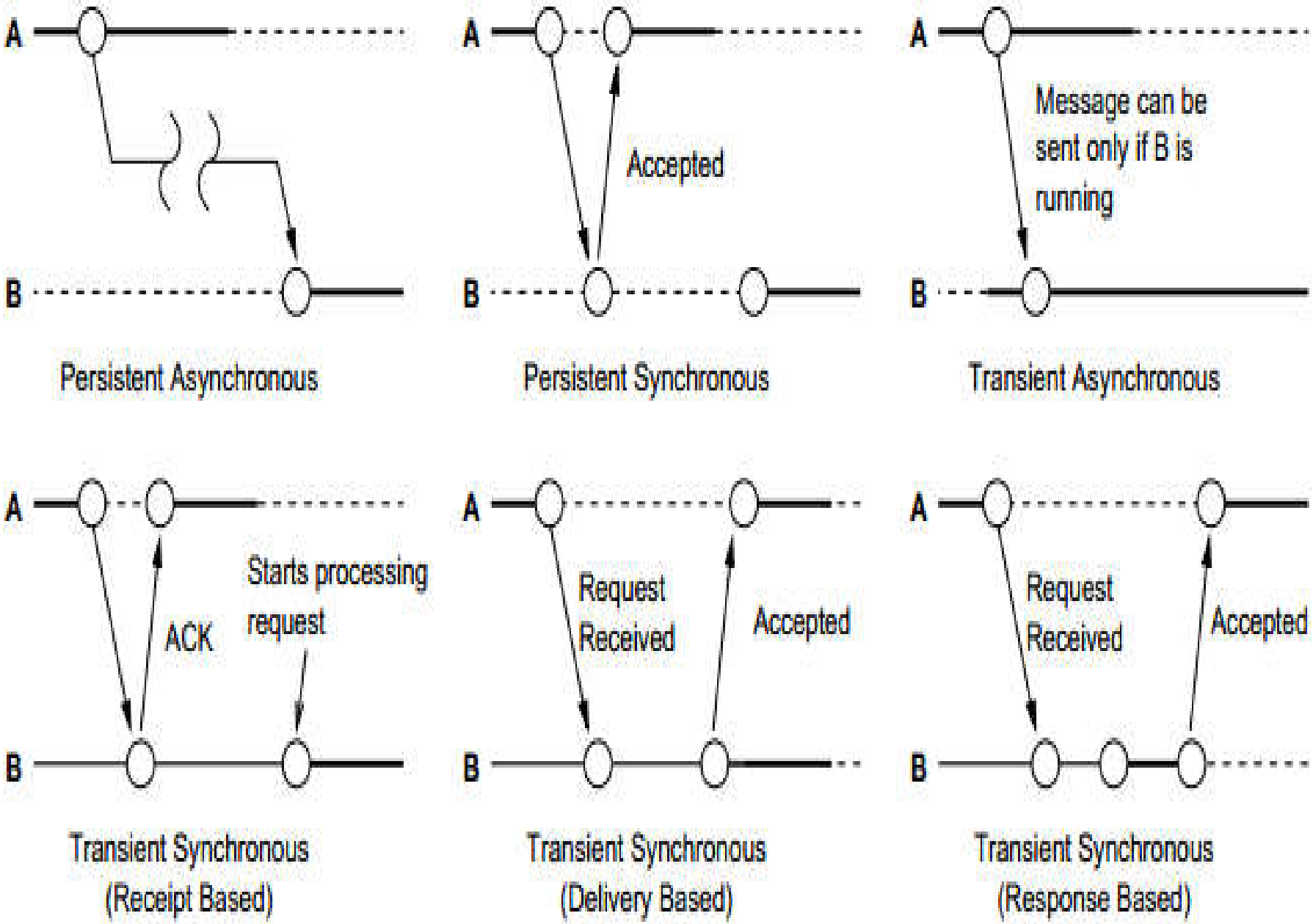
## COMMUNICATION IN DISTRIBUTED SYSTEMS

# Introduction

- What is communication and it purpose?

**Types of Communication**

| Comm. mode | Description |
|---|---|
| **Data –oriented communication** | - To exchange data b/n processes |
| **Control-oriented communication** | - Associates a transfer of control with every data transfer |
| **Synchronous communication** | - The sender blocks itself until the message has been received by the intended recipient. (Stronger)<br>- Reply is expected |
| **Asynchronous communication** | - Sender continues execution immediately after sending a message<br>- Buffering is required otherwise the message will be lost |
| **Transient communication** | - The message only delivered if the receiver is active |
| **Persistent communication** | - A message will be stored in the system until it can be delivered to the intended recipient |


A
B
Persistent Asynchronous
Accepted
Persistent Synchronous
Message can be sent only if B is running
Transient Asynchronous
ACK
Starts processing request
Transient Synchronous (Receipt Based)
Request Received
Accepted
Transient Synchronous (Delivery Based)
Request Received
Accepted
Transient Synchronous (Response Based)

# Layered Protocols

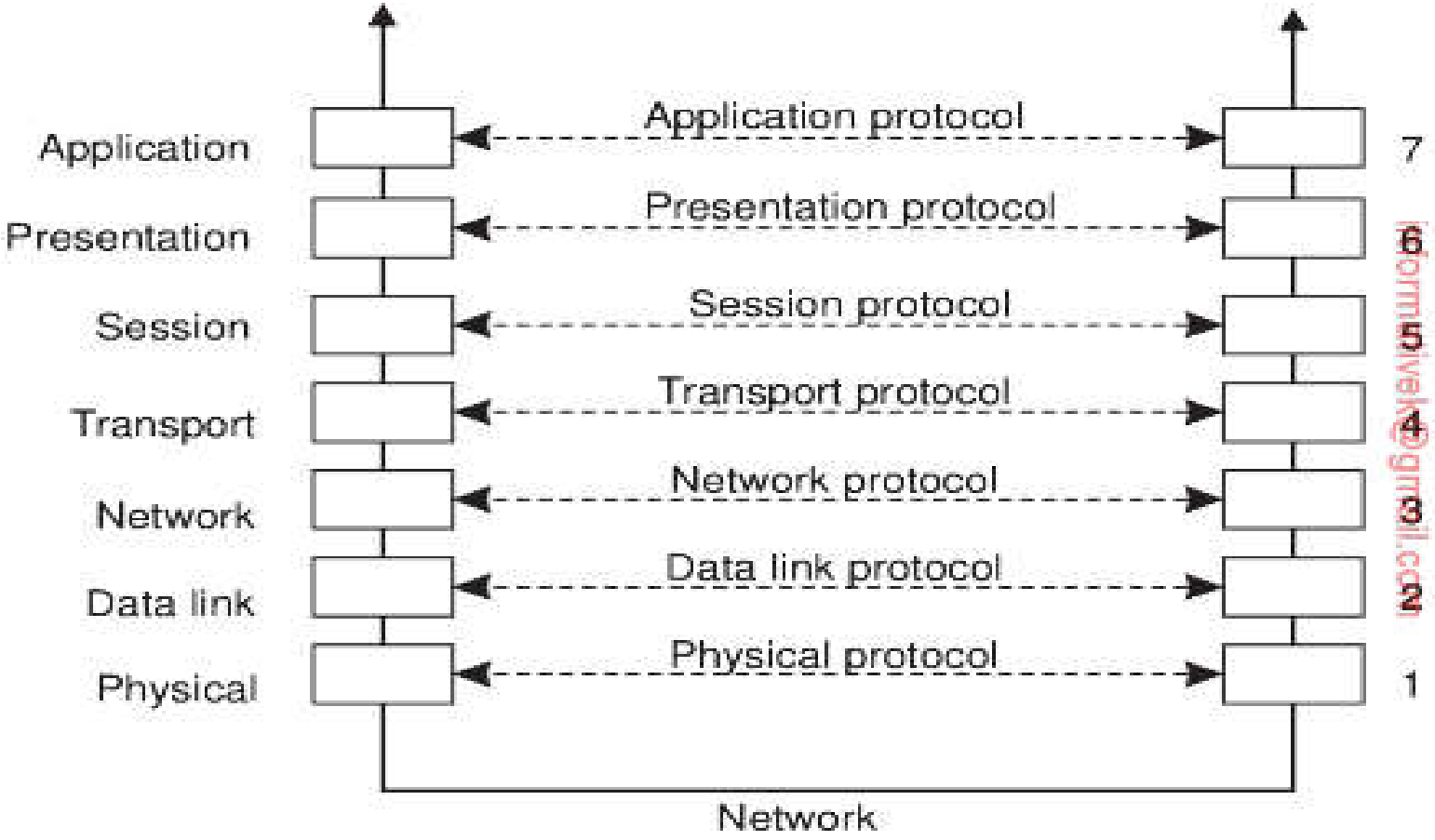



Figure 4.1 OSI reference model

**Physical layer** Deals with standardizing how two computers are connected and how 0s and 1s are represented.

**Data link layer** Provides the means to detect and possibly correct transmission errors, as well as protocols to keep a sender and receiver in the same pace.

**Network layer** Contains the protocols for routing a message through a computer network, as well as protocols for handling congestion.

**Transport layer** Mainly contains protocols for directly supporting applications, such as those that establish reliable communication, or support real-time streaming of data.

**Session layer** Provides support for sessions between applications.

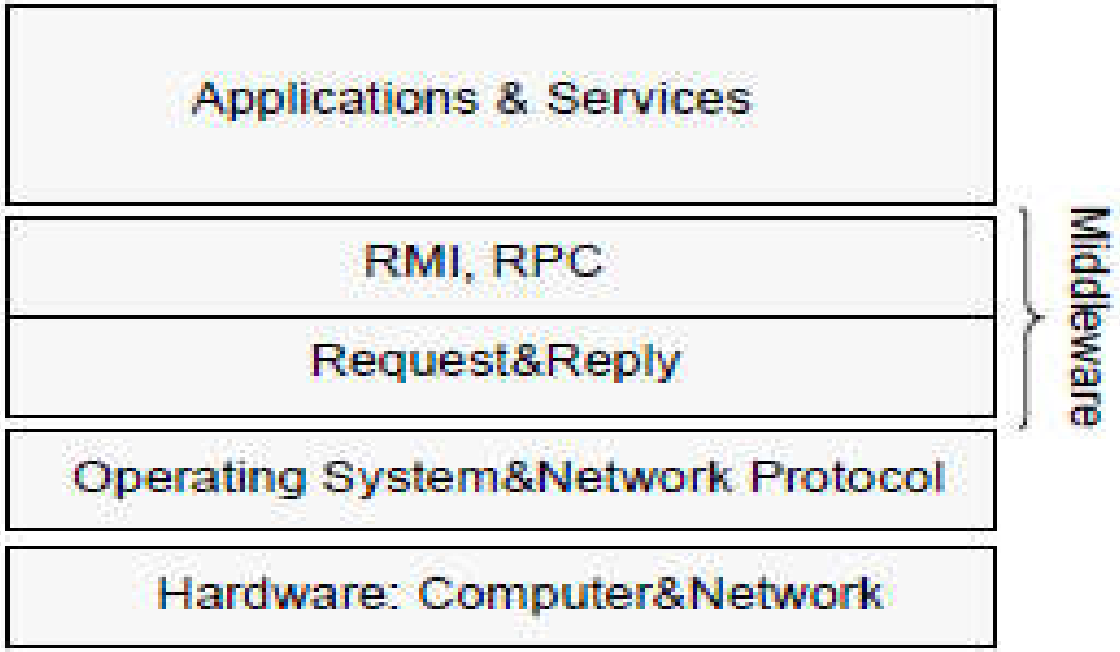
**Presentation layer** Prescribes how data is represented in a way that is independent of the hosts on which communicating applications are running.

**Application layer** Essentially, everything else: e-mail protocols, Web access protocols, file-transfer protocols, and so on.

# TCP/IP model

| Layer | New TCP Model |
|---|---|
| 5 | Application |
| 4 | Transport |
| 3 | Network |
| 2 | Data Link |
| 1 | Physical |

## Middleware protocols



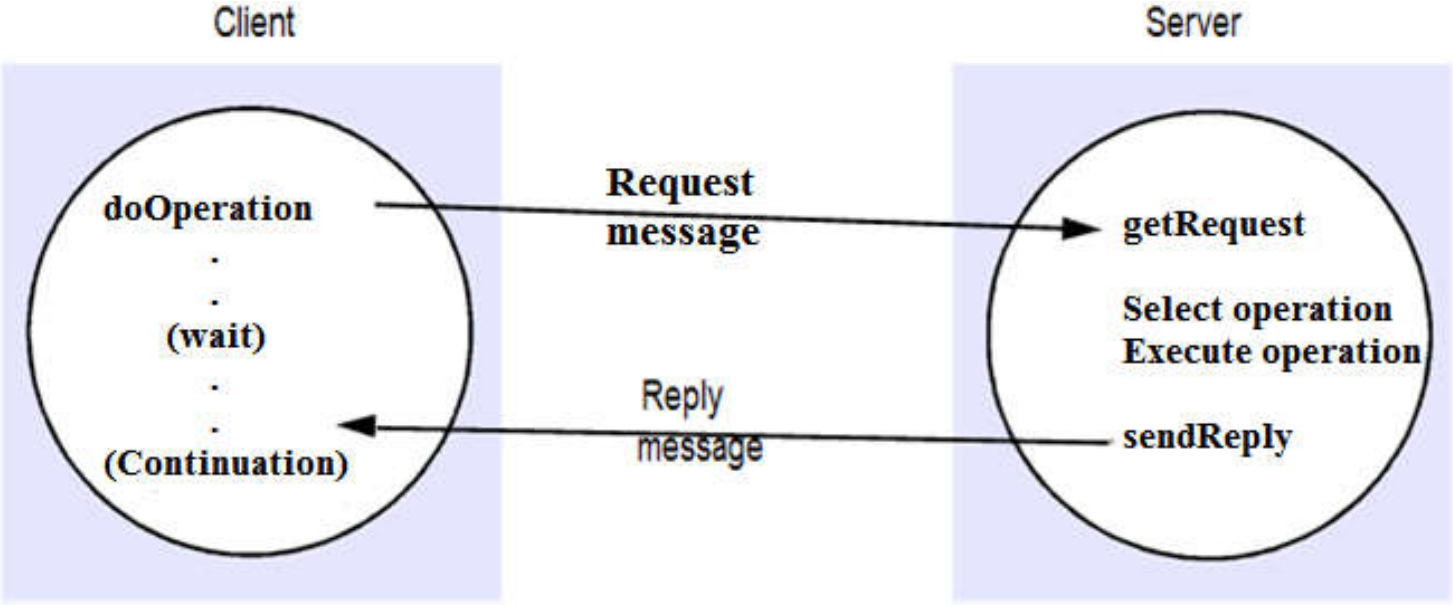

Middleware (Request-Replay, RPC, and RMI) and distributed applications have to be implemented on top of a network protocol

- *Communication between processes and objects in a distributed system is performed by **message passing.***
- *The following are methods/middleware frameworks that are used to establish communication between processes and objects of client and remote server in a distributed systems.*

**1. Request-Reply Protocol Communication in a Client-Server Model**

– The system *is structured as a group of processes (objects), called servers* that deliver services to *clients*.

Operations of the request-reply protocol




***public byte[] doOperation (RemoteRef s, int operationId, byte[] arguments);***

- Sends a request message to the remote server and receives the reply.
- The arguments specify the remote server, the operation to be invoked and the arguments of that operation.

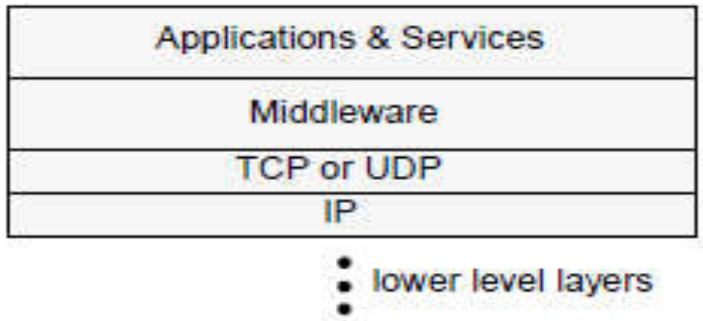
***public byte[] getRequest ();***

- Acquires a client request via the server port.

***public void sendReply (byte[] reply, InetAddress clientHost, int clientPort);***

- Sends the reply message to the client at its Internet address and port.

- **The client:**send (request) to server_reference;  receive(reply)
- **The server:**receive(request) from client-reference; *execute requested operation, s*end (reply) to client_reference;
- *Request* and *reply* are ***implemented based on the network protocol*** (e.g. TCP or UDP in case of the Internet)




**TCP is a reliable protocol**

- TCP implements additional mechanisms on top of IP to meet reliability guarantees.

**UDP is a protocol that does not guarantee reliable transmission.**

- UDP offers no guarantee of delivery.

- HTTP: An example of a request-reply protocol:
- HTTP is used by web browser clients to make requests to web servers and to receive replies from them.
- HTTP is implemented over TCP
- Each client- server interaction consisted of the following steps:
  - The client request and the server accept a connection at the default server port or at a port specified in the URL.
  - The client sends a request message to the server.
  - The server sends a reply message to the client.
  - The connection is closed.

- HTTP Request message

| *Method* | *URL or pathname* | *HTTP version* | *Headers* | *Message body* |
|---|---|---|---|---|
| GET | http://www.dcs.qmul.ac.uk/index.html | HTTP/ 1.1 | | |

- HTTP Reply message

| *HTTP version* | *Status code* | *Reason* | *Headers* | *Message body* |
|---|---|---|---|---|
| HTTP/1.1 | 200 | OK | | resource data |

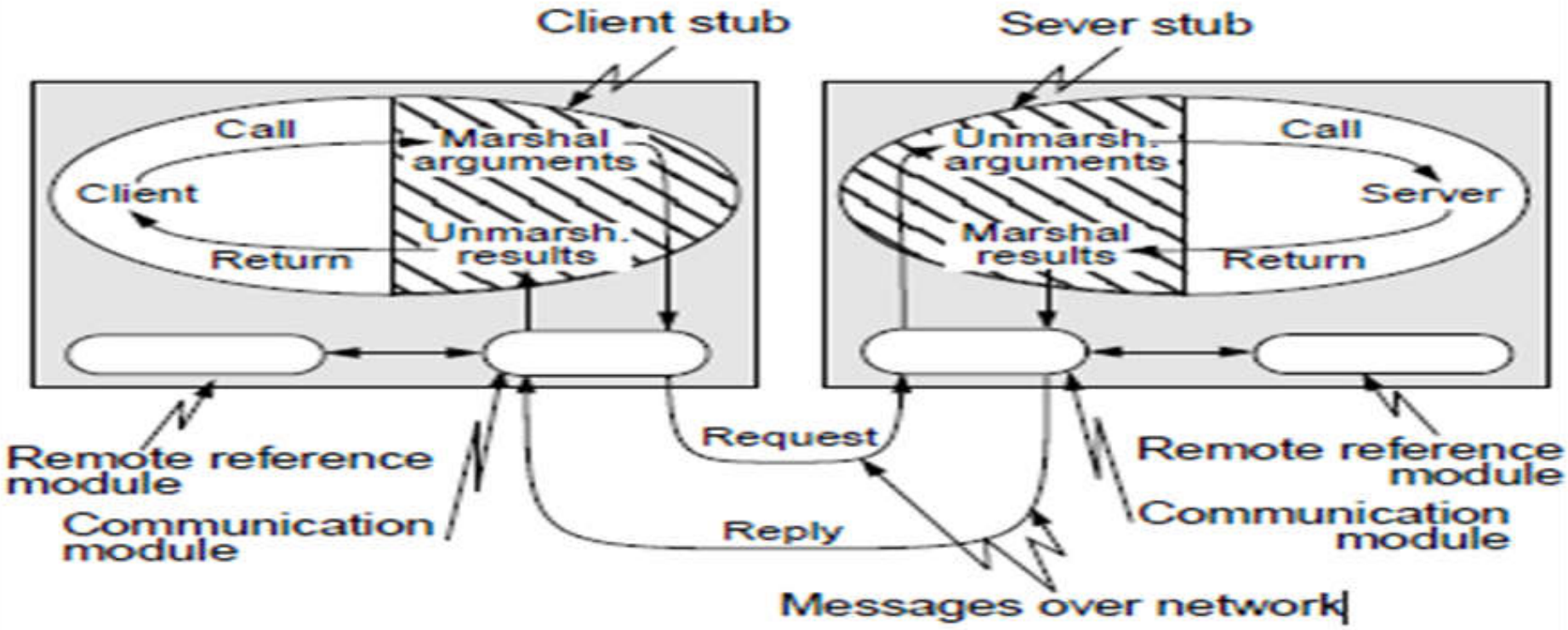
# 2.1. RPC(Remote Procedure Call)

- an inter-process communication that allows a computer program to cause a procedure to execute in another address space (commonly on another computer on a shared network)
- RPC is initiated by the client, by sending a request message to the remote server to execute a procedure with the supplied parameters.
- The remote server sends a response to the client, and the application continues its process.
- The client is blocked until the server processes the call

- **Difference b/n local and remote Procedure Call is:**
  - remote calls can fail because of unpredictable network problems
  - Callers must deal with such failures without knowing whether the remote procedure was actually invoked
- **Goals of RPC:**
  - Transparency: the calling object (procedure) is not aware that the called one is executing on a different machine, and vice versa
  - The originators of RPC, aimed to make remote procedure calls as much like local procedure calls as possible (no distinction in syntax)
  - RPC strives to offer at least location and access transparency, hiding the physical location of the remote procedure and also accessing local and remote procedures in the same way.

- **Sequence of events during an RPC**
  - The client calls the client stub.
    - Local call, parameters are passed
  - The client stub packs the parameters into a message and makes a system call to send the message.
    - Packing the parameters is called marshalling.
  - The client's local OS sends the message from the client machine to the server machine
  - The local OS on the server machine passes the incoming packets to the server stub.
  - The server stub unpacks the parameters from the message.
    - Unpacking the parameters is called unmarshalling.
  - Finally, the server stub calls the server procedure. The reply traces the same steps in the reverse direction.

# Remote Procedure Call



- Client Process
  - includes one *stub procedure* for each procedure in the service interface
  - The stub procedure marshals the procedure identifier and the arguments into a request message and sends it to the server via communication module.
  - When the reply message arrives, it unmarshals the results.
- Server Process
  - contains a dispatcher together with one server stub procedure and one service procedure for each procedure in the service interface

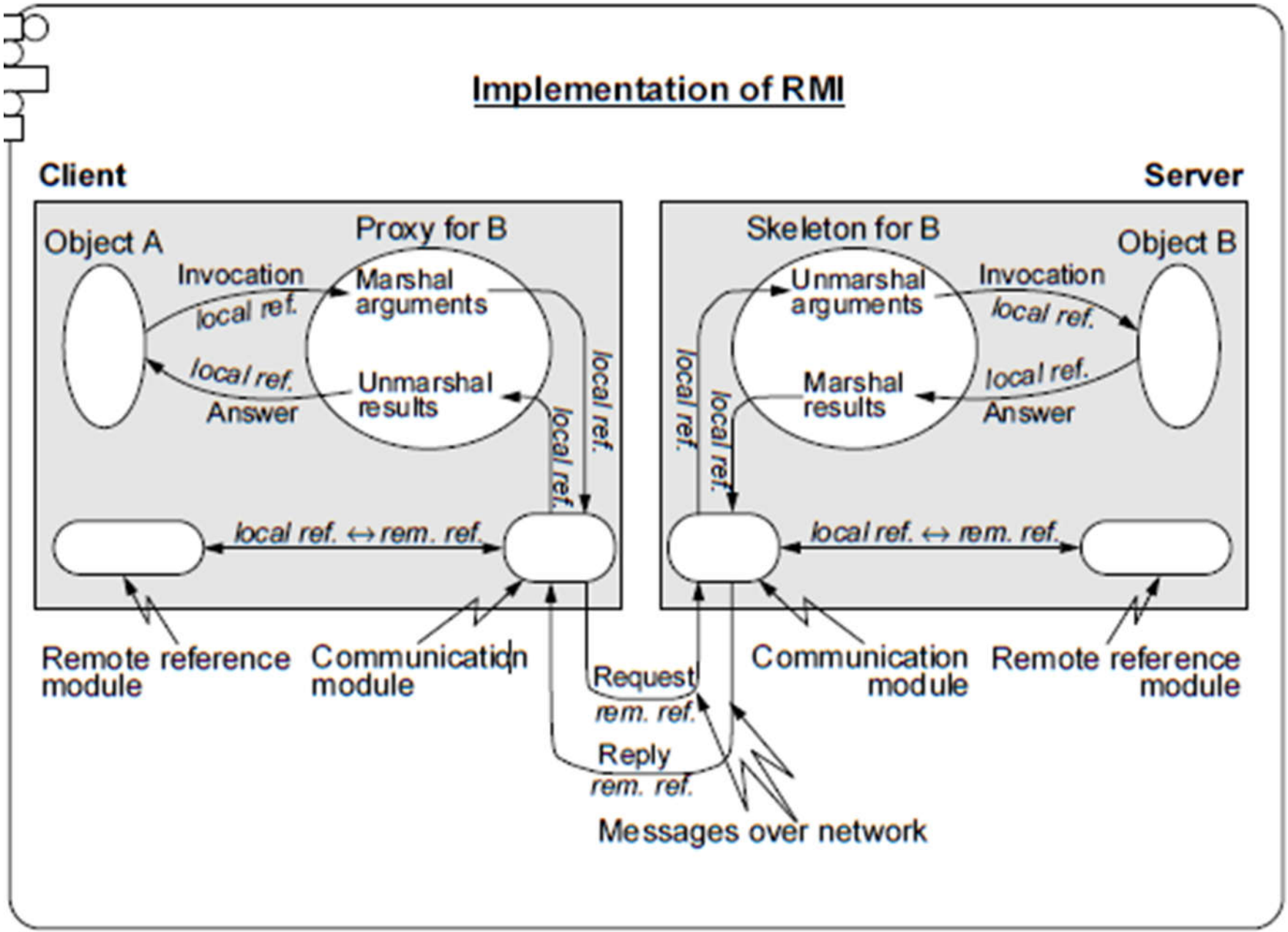
- The dispatcher selects *one of the server stub procedures* according to the *procedure identifier in the request message*.
- The server stub procedure then *unmarshals the arguments in the request message*, *calls the corresponding service procedure* and *marshals the return values for the reply message*.
- The service procedures implement the *procedures in the service interface*.
- The client and server stub procedures and the dispatcher can be *generated automatically by an interface compiler* from the *interface definition of the service*.

# 2.2. RMI (Remote Method Invocation)

- allows applications to
  - call object methods located remotely,
  - share resources and processing load across systems
- Unlike (RPC) which require that only simple data types or defined structures be passed to and from methods, RMI allows any Java object type to be used
- RMI allows both client and server to dynamically load new object types as required.

- Differences b/n RMI & RPC
  - In RMI, a calling object can invoke a method in a potentially remote object. In RPC, the underlying details are generally hidden from the user
  - The programmer is able to use the full expressive power of object-oriented programming in the development of DS, including the use of objects, classes and inheritance, and can also employ related object- oriented design methodologies and associated tools.
  - all objects in an RMI-based system have unique object references (whether they are local or remote), such object references can also be passed as parameters, thus offering significantly richer parameter-passing semantics than in RPC.

- RMI allows the programmer to pass parameters not only by value, as input or output parameters, but also by object reference.
- Passing references is particularly attractive if the underlying parameter is large or complex.
- The remote end, on receiving an object reference, can then access this object using remote method invocation, instead of having to transmit the object value across the network.


Implementation of RMI
Client
Server
Object A
Proxy for B
Skeleton for B
Object B
Invocation
local ref.
Marshal arguments
Unmarshal arguments
Invocation
local ref.
local ref.
Answer
Unmarshal results
Marshal results
local ref.
Answer
local ref.
local ref.
local ref.
local ref.
local ref. ↔ rem. ref.
local ref. ↔ rem. ref.
Remote reference module
Communication module
Request
rem. ref.
Reply
rem. ref.
Communication module
Remote reference module
Messages over network

- Who are the players?
  - Objects
    - Object A asks for a service
    - Object B delivers the service
  - The proxy for object B
    - If an object A holds a remote reference to a (remote) object B, there exists a proxy object for B on the machine which hosts A
    - created when the remote object reference is used for the first time.
    - is the local representative of the remote object (For each method in B there exists a corresponding method in the proxy)
    - *marshals* the arguments and builds the message to be sent, as a request, to the server.
    - *unmarshals* the received message and sends the results, in an answer, to the invoker.
  - **The skeleton for object B**
    - unmarshals the requested message and invokes the corresponding method in the remote object; it waits for the result and marshals it into the message to be sent with the reply

- Dispatcher
  - A part of the skeleton , it receives a request from the *communication module*, identifies the invoked method and directs the request to the corresponding method of the skeleton.
- Communication module
  - responsible of carrying out the exchange of messages which implement the request/reply protocol needed to execute the remote invocation.
- Remote reference module
  - translates between local and remote object references
  - The correspondence between them is recorded in a *remote object table*.
  - Remote object references are initially obtained by a client from a so called *binder* that is part of the global name service
  - Here servers register their remote objects and clients look up after services.

- Object A and Object B belong to the application.
- Remote reference module and communication module belong to the middleware.
- The proxy for B and the skeleton for B represent the so called *RMI software*.

**Steps of Client and server objects communication in RMI**

1. The calling sequence in the client object activates the method in the proxy corresponding to the invoked method in B.
2. The method in the proxy packs the arguments into a message (marshalling) and forwards it to the communication module.
3. Based on the remote reference obtained from the remote reference module, the communication module initiates the request/reply protocol over the network.
4. The communication module on the server's machine receives the request. Based on the local reference received from the remote reference module the corresponding method in the skeleton for B is activated.

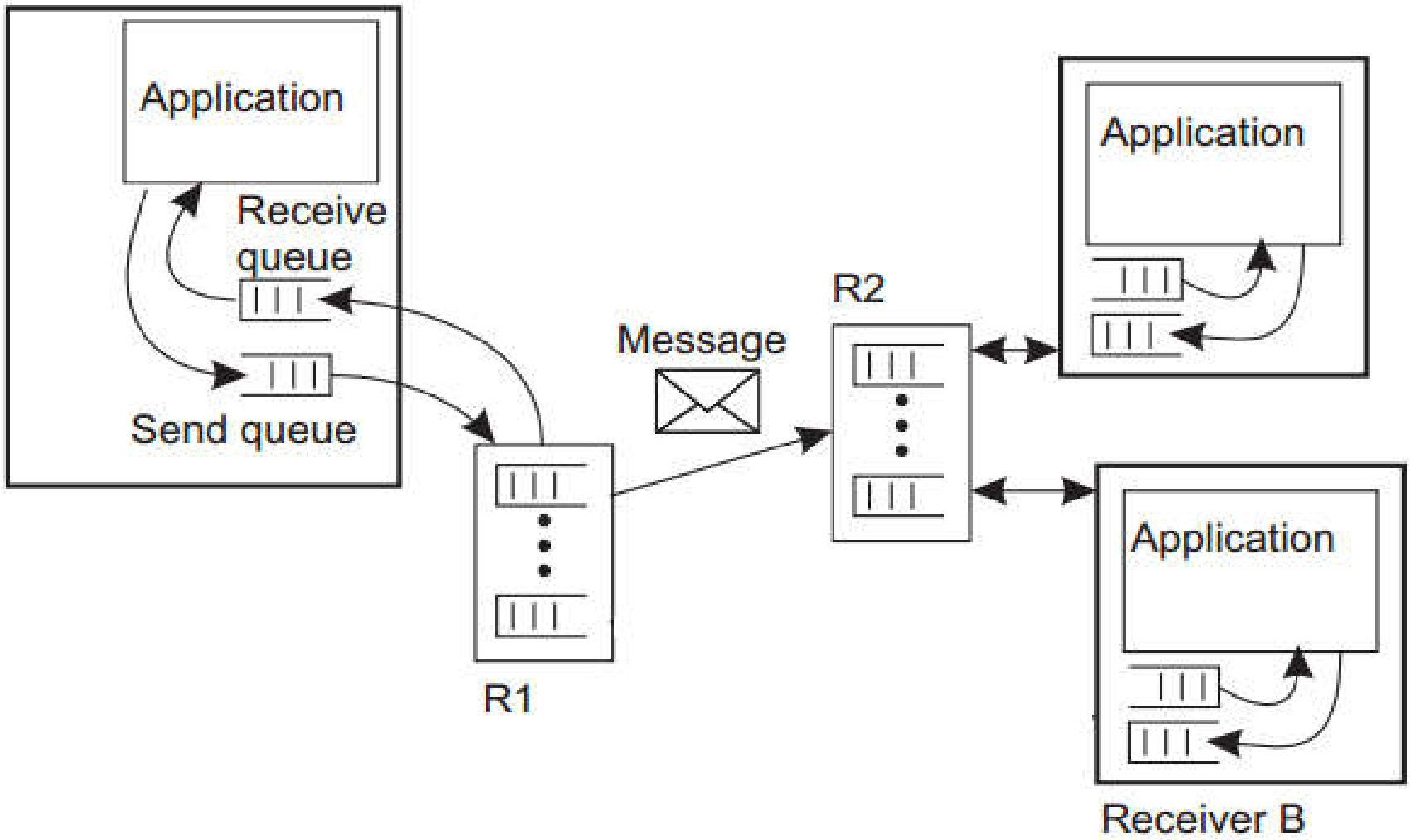
5. The skeleton method extracts the arguments from the received message (unmarshalling) and activates the corresponding method in the server object B.
6. After receiving the results from B, the method in the skeleton packs them into the message to be sent back (marshalling) and forwards this message to the communication module.
7. The communication module sends the reply, through the network, to the client's machine.
8. The communication module receives the reply and forwards it to the corresponding method in the proxy.
9. The proxy method extracts the results from the received message (unmarshalling) and forwards to the client

# Message-Oriented Communication

- Based on the model of processes sending messages to each other
- Can be synchronous/asynchronous/transient/persistent
- Provided by Message Oriented Middleware (MOM)
- MOM
  - provides **send()** and **receive()** primitives which abstracts the underlying OS and HW.
  - Allows programmers to use message passing without having to ware of about platforms, services provided in the platforms etc.
  - Provides marshalling services
  - **E.g**. 1. Message Passing Interface (MPI) – transient, designed for parallel computing, uses available network protocols

    2.MQ series from IBM-Provides persistent communication

    - Messages are sent to other processes by placing them in queues.




An example of a message queuing system

# Stream-Oriented Communication

- Deals with continuous media, data represented as a single stream of data rather than discrete chunks
- The main characteristic of continuous media
  - a spatial relationship (i.e., the ordering of the data)
  - temporal relationship between the data.
  - **Example**: Film is a good example of continuous media.

    - Frames of a film be played in the right order and played at the right time, otherwise the result will be incorrect.
- Stream of continuous are examples of isochronous communication (communication that has minimum and maximum end-to-end time delay requirements)
- When dealing with isochronous comm.,quality of service is an important issue.(i.e. timeliness and reliability)