Contents

- Remote procedure call.
- Remote object invocation.

Introduction

- Communication is the way of exchanging messages and information by various processes running on different machines.
- The inter-process communication is the form of communication that makes interaction possible among multiple processes running on different machines over the network.
- There are different communication techniques provided by the distributed system to run various distributed applications.
- There are different communication models like remote procedure call, remote object invocation, message-oriented communication and stream-oriented communication.

Remote Procedure Call (RPC)

- In the distributed system, multiple processes running on the different machines can communicate with each other by sending and receiving explicit messages using the low-level message format.
- The basic and most commonly used mode of communication in the distributed system is the RPC.
- RPC is a powerful technique for constructing the distributed, client-server based applications.
- The RPC allows processes on the sender's machine to call the procedures or subroutines on the receiver's machine, where the call is generated at the sender's local address space and the procedure is executed at the receiver's address space remotely.
- Therefore, in RPC, although the procedure is called and executed remotely, yet it appears like a local to the sender's process.

Remote Procedure Call (RPC)

- The RPC uses conventional inter-process communication (IPC) to call the procedures on the remote machine, which are exposed by the interfaces.
- The interfaces in RPC provide signatures for a set of methods without specifying their implementation.
- Each client in RPC has at least one stub procedure in the service interface, which is responsible for marshaling and unmarshaling of the data and passing the message to the communication module.
- The server in RPC has a dispatcher, which uses server stub to fetch the appropriate procedure as per request message and process it along with the user's data.

Remote Procedure Call (RPC)

- **Client and Server Stub**
- The RPC is intended to make communication between the processes and procedures, where the parameters are passed by the local process to the remote procedure using the local system call.
- The remote procedure executes the request and sends result back to the client process using remote system calls.
- Therefore, to make RPC more transparent, the calling process at the client side packs parameters and data into a message called the client stub.
- This message transforms digital data into wire format.
- The packing of the data is also called the marshaling process.
- The RPC transport sends client stub across the network to the server side.

Remote Procedure Call (RPC)

- The server stub is generated at the server side, which unpacks client stub and passes parameters to the called procedure that generates the result.
- The server stub is also responsible for packing the result into the message and sending across the network to the client, where the client stub unpacks it and sends the result back to the client's process.
- The unpacking of the data is also called the unmarshaling process.

- **Working of the Remote Procedure Call**
- In RPC, the client is always an invoker of method who uses request primitive, while the server responds to the client's request for the execution of procedure.

Remote Procedure Call (RPC)

- The RPC process consists of 10 steps depicted in figure 3 and explained as follows.

1. The client process invokes a procedure that is running on the server machine.
2. The client stub is generated using the local system call by the local operating system, where it packs the request message; the client will be blocked till it gets the response from the server.
3. The local operating system kernel passes the client stub to the RPC transport.
4. The client side RPC transport sends the client stub to the server side RPC transport.
5. The remote kernel at the server side passes the client stub to the server stub.
6. The server stub unpacks the client stub and forwards the parameters to an appropriate called procedure.

Remote Procedure Call (RPC)

7. The server procedure processes the client's request and packs the result into the server stub.
8. The server operating system sends the server stub to the client's local operating system using RPC transport.
9. The client's local operating system passes the server stub to the client stub.
10. Finally, the client stub unpacks the server stub and returns the result back to the called client process.

- **Parameter Passing in RPC**
- In RPC, the parameters are passed in two ways - pass by value and pass by reference.
- In pass by value, the actual parameters and their data types are copied into the stack and passed to the called procedure.

Remote Procedure Call (RPC)

- In pass by reference, the pointer to the data is passed instead of value to the called procedure at the server side.
- However, it is very difficult to implement as the server needs to keep the track of the pointer to the data at the client's address space.

- **Extended RPC Models**
- The extended RPC models are the system which uses RPC for their communication.
- However, the basic pattern of interaction will be same.
- The two extended RPC models – doors and asynchronous RPC.

Remote Procedure Call (RPC)

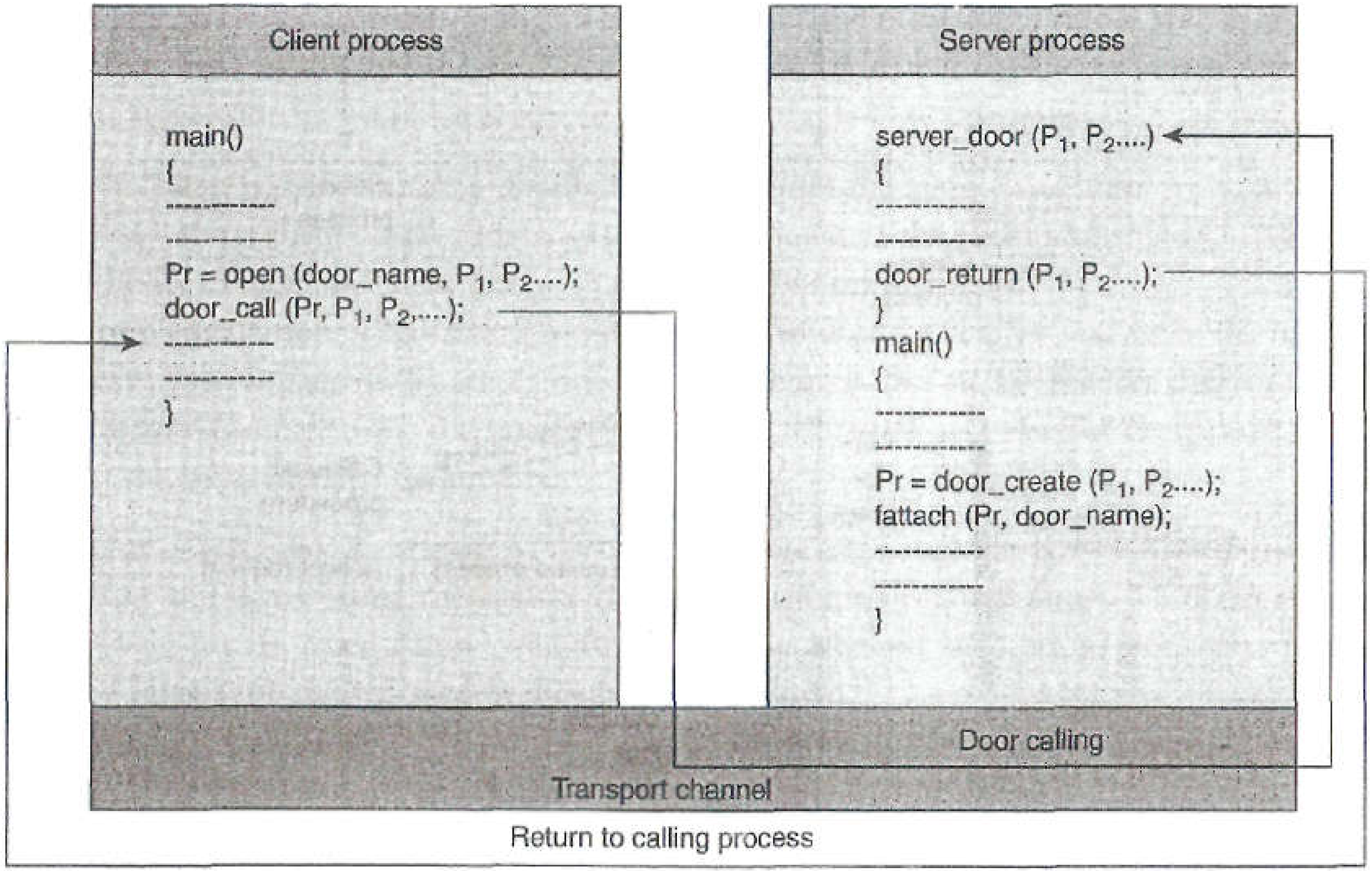
- **Doors**
- The doors were developed by the Sun Microsystems for their Spring Operating System.
- They allow server processes to create the door and the client processes to the call doors.
- The door communication model uses IPC to call a subroutine or a procedure on the remote host, which works similar to the RPC.
- The doors are created by the server using door_create() method, while the client invokes different procedures on the server using door_call() method.
- As a response, the server process calls door_return() method to return the result back to the called process.
- It also allows the client and the server to get and manage the communication information about each other.

Remote Procedure Call (RPC)

- The server machine has multiple processes that are responsible for creating and managing the pool of shared threads, while the client has at least one process that calls door_call() method.
- The only disadvantage of the door is that the developer has to keep a track of all the processes and threads that are creating and running on the doors at the remote server.
- Figure 4 shows the door mechanism.

Remote Procedure Call (RPC)

- Figure 5.4 Door mechanism.

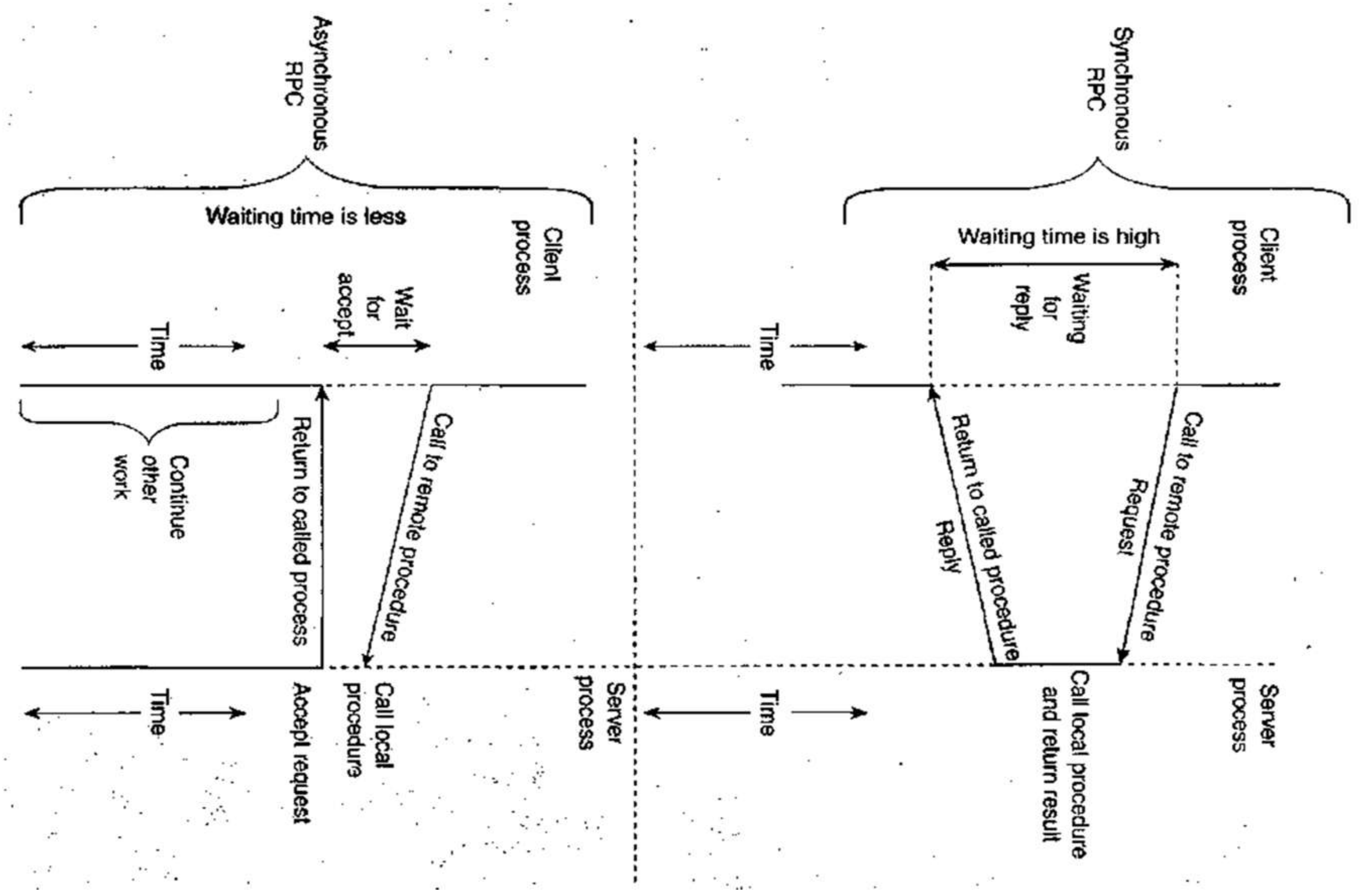


Remote Procedure Call (RPC)

- **Asynchronous RPC**
- The traditional RPC is synchronous in nature where once the client sends request to the server, it waits for an indefinite time until the server responds with the reply.
- If there is no response or result from the server, the client is unnecessary blocked for certain amount of time during which the client would have done other useful work.
- Therefore, to avoid unwanted and unnecessary blocking of the client, the RPC provides an alternative called asynchronous RPC.
- Figure 5 illustrates synchronous and asynchronous RPC.
- In asynchronous RPC, the client process can continue its work with the other processes immediately-after issuing a request to the server.

Remote Procedure Call (RPC)

- Figure 5.5 Synchronous and asynchronous RPC.



Remote Procedure Call (RPC)

- It never allows client process to wait for the response from the server.
- Thus, unwanted blocking is avoided in asynchronous RPC.
- The server process also sends response message to the client immediately after receiving the request message.
- The disadvantage of asynchronous RPC is the lack of reliability and it is very difficult to implement too.

Remote Procedure Call (RPC)

- **Example of RPC**
- RPC is a widely used communication model in most of the distributed applications.
- The distributed computing environment (DCE) is one of the systems, which follows the RPC model.
- It is broadly used in most of the modern distributed applications.
- The DCE is a client-server model used in the middleware applications that allow the client machine to access various services provided by the server machine over the network using the abstraction layer.
- The common services provided by DCE are thread service, security service, directory service, file service and time service.

Remote Object Invocation

- The remote object invocation is a common object model used in most of the distributed system applications.
- It provides distinct features as compared to the existing RPC.
- In the remote object invocation, the objects are distributed across the network that encapsulates the data and operations.
- It is called remote object because call is initiated at local machine and uses objects from the remote machine.
- The distributed objects can be accessed through a common interface at the client side as well as at the server side.
- Therefore, we can say that the remote object invocation uses interfaces to locate and fetch the object at the server side, whereas execution takes place at the remote side only.

Remote Object Invocation

- When the client wants to access a distributed object from the remote machine, it packs the operation into proxy at the client's local address space that is similar to the client stub of RPC.
- When proxy reaches to the server side, the client proxy is unmarshaled or unpacked and then sever skeleton locates the object implementation.
- The server procedure then executes the request and finds the result.
- The server skeleton is analogous to the server stub in RPC which is also responsible for packing the result into the skeleton.
- The skeleton is ultimately unpacked by the proxy to return the result back to the client process which has invoked the method.

Remote Object Invocation

- During the object invocation process, the object adaptor plays an important role for creating, locating, activating and deactivating the remote objects from the objects repository.
- CORBA (Common Object Request Broker Architecture) and RMI (Remote Method Invocation) are the two popular object-based invocation models used in the distributed system.

1 Types of Object and Binding

- The objects in the distributed system are either created at the compilation time or at the run time dynamically.
- If the client has compiled the time knowledge and interfaces of the objects, then the object binding is called static binding and

Remote Object Invocation

- if the objects ate initialized and created at runtime where the client has no compiled time knowledge about implementation of objects and their interfaces, then the object binding is called dynamic binding.
- Therefore, the objects used in the static binding are called static objects, while the objects used in the dynamic binding are called dynamic objects.
- There are two more types of object binding - implicit binding and explicit binding.
- In the implicit binding, the client process directly invokes the method and binds the object directly without using any special function.
- In the explicit binding, a special intermediate function is needed to bind the object of the invoking method. It uses pointer to the client proxy for remote calling of the objects.

Remote Object Invocation

- The objects can be persistent if they are permanently stored on the secondary storage so that some processes can use and load them in their own address space anytime.
- Conversely, the transient objects have short life span and they are temporal in nature; they exist until server process uses them.
- Once the process exits, the objects associated with the server process are automatically removed from the servers address space.

Remote Method Invocation (RMI)

- The RMI is a remote object invocation technique used to locate and fetch the objects at the remote side using object reference.
- In RMI, the client machine is always an invoker of methods, which initiates the remote method call by using the local process.
- RMI allows programmers to develop distributed programs with the same syntax and semantics as used for the standalone programs.

1 Architecture of Remote Method Invocation

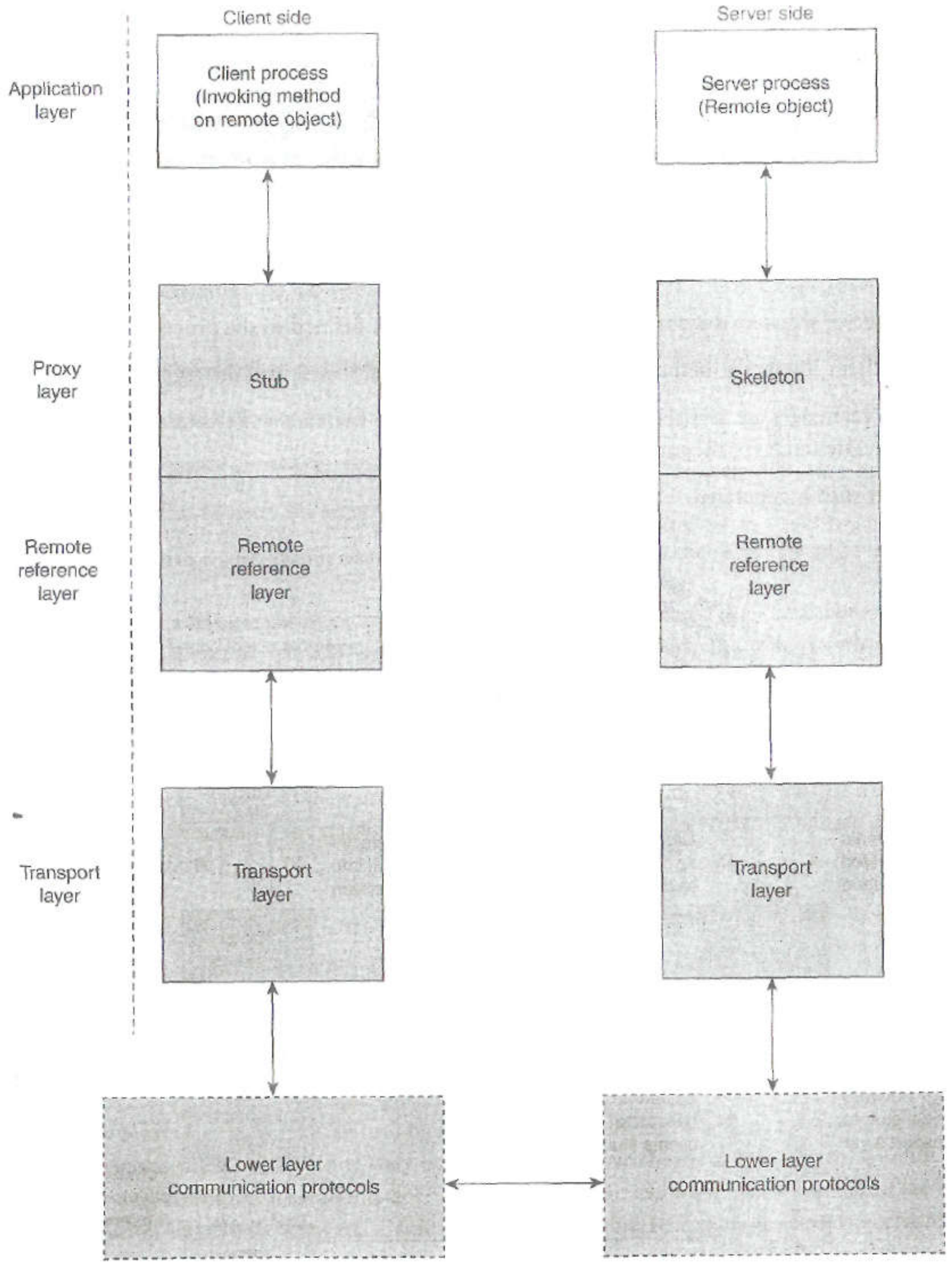
- The RMI architecture has four layers - application layer, proxy layer, remote reference layer, and transport layer as depicted in figure 6.

Remote Method Invocation (RMI)

- The application layer is responsible for running the client and server applications, where the client application invokes methods defined by the server application.
- When the client invokes a method, then the request is passed to the proxy layer.
- The proxy layer is responsible for creating client stub at the client side by packing the request message sent by the client process and creating the skeleton by packing the response message sent by the server.
- Once the stubs and skeletons are created, they are passed to the remote reference layer that checks the semantics and remote references used by the client process using the remote reference protocol.

RMI

- Finally, the remote reference layer transmits the messages and data to the RMI transport layer that is responsible for establishing and maintaining a stream-oriented connection between the client and the server.
- The transport layer is also responsible for managing the send/receive of the request/reply messages between the client and the server.



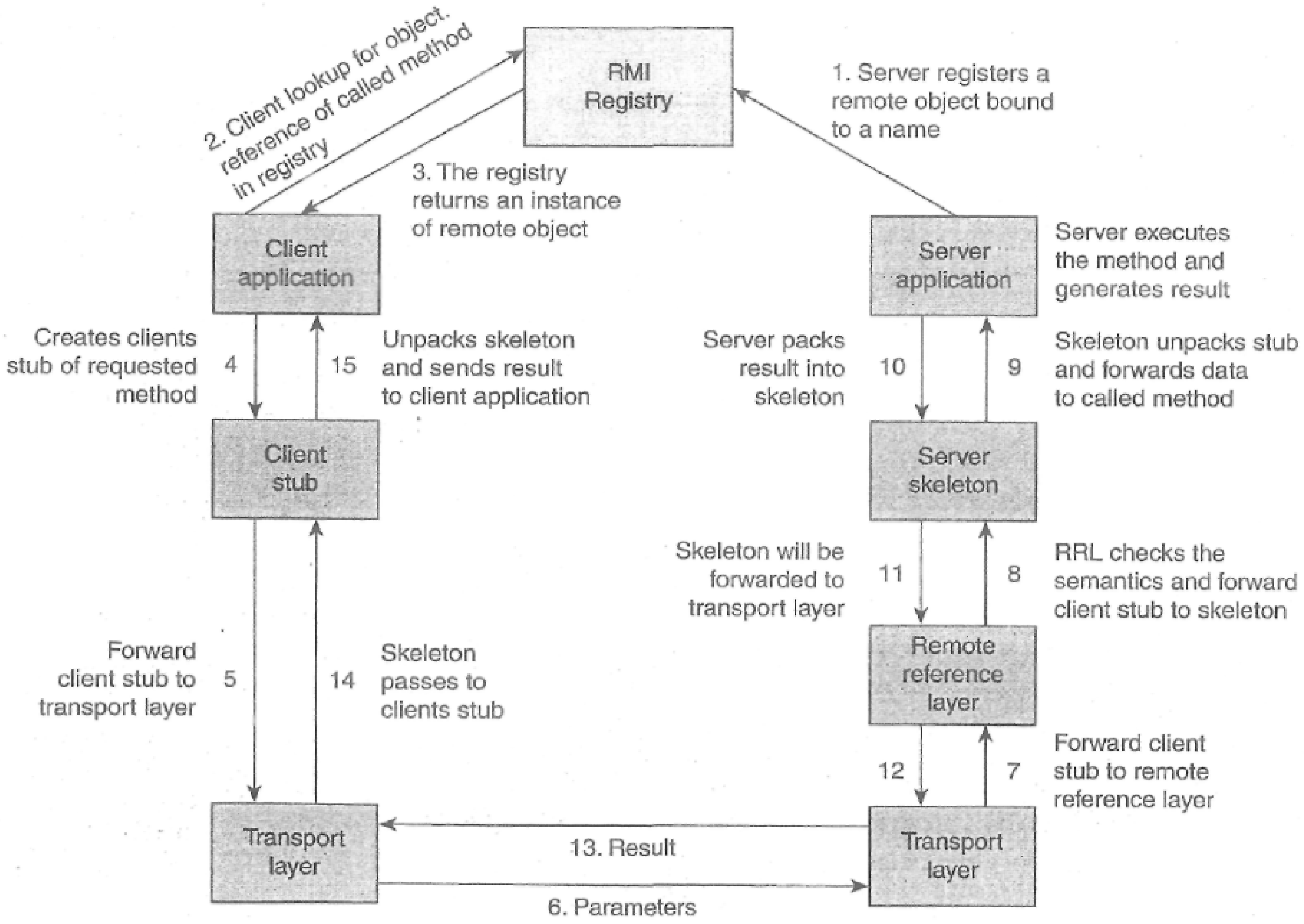
Remote Method Invocation (RMI)

2 Remote Method Invocation Process

- The RMI is designed to provide remote execution environments to the distributed applications so that any machine in the network can invoke any method denned on the remote server and execute it like a local method.
- In RMI, the registry plays an important role in storing and providing the object references from the object repository so that whenever the client invokes any method, it checks the registry to get the object references.
- The steps involved in the RMI execution process are depicted in figure 7 and are explained as follows:

1. The server registers the object references of all methods defined in the procedure.
2. The client invokes a method and searches the object references into the registry.

Remote Method Invocation (RMI) Process


RMI Registry
1. Server registers a remote object bound to a name
2. Client lookup for object. reference of called method in registry
3. The registry returns an instance of remote object
Client application
Server application
Server executes the method and generates result
Creates clients stub of requested method
4
15
Unpacks skeleton and sends result to client application
Server packs result into skeleton
10
9
Skeleton unpacks stub and forwards data to called method
Client stub
Server skeleton
Skeleton will be forwarded to transport layer
11
8
RRL checks the semantics and forward client stub to skeleton
Forward client stub to transport layer
5
14
Skeleton passes to clients stub
Remote reference layer
12
7
Forward client stub to remote reference layer
Transport layer
13. Result
6. Parameters
Transport layer

Remote Method Invocation (RMI)

3. If the references are available, only then the client packs the data and creates the client stub; else, it generates an error message.
4. Client stub is generated.
5. The client stub is passed to the RPC transport at client side.
6. RPC transport at client side forwards it to server side.
7. Transport layer at server side forwards client stub to remote reference layer.
8. The remote reference layer receives the client stub and after checking semantics, forwards it to the server skeleton.
9. The server skeleton unpacks the client stub and locates the invoked method.
10. The server executes the client's invoked method, generates the result and packs it into the skeleton.

Remote Method Invocation (RMI)

11. The skeleton is passed to the Remote Reference Layer (RRL).
12. RRL forwards skeleton to transport layer.
13. The RPC transport forwards the skeleton to the client stub.
14. The client stub receives the server skeleton and unpacks it.
15. The client stub returns result back to the client obtained from the server through the skeleton.

- The invocation in RMI can be static or dynamic.
- In static invocation, the client compiles the time knowledge about all interfaces and implementations associated with the objects.
- However, in the dynamic invocation, the interfaces are defined and objects are created dynamically at the run time.

Remote Method Invocation (RMI)

- The two popular RMI models are Distributed Computing Environment (DCE) and Java RMI.
- In DCE, the interface definition language is used at the server side to make a remote object appear like a local.
- The distributed named objects are stored at the server side, which are located and fetched using their name by the client using the naming service.
- The DCE mostly uses binding handle to identify the interfaces of an objects.