

# **Network Programming**

## **[CAC355]**

### **BCA 6<sup>th</sup> Sem**

Er. Sital Prasad Mandal

(Email : [info.sitalmandal@gmail.com](mailto:info.sitalmandal@gmail.com))  
Bhadrapur, Jhapa, Nepal

<https://networkprogam-mmc.blogspot.com/>

# Unit-12

## RMI

1	RMI Service Interface
2	Implementing RMI Service Interface
3	Creating RMI Server
4	Creating RMI Client
5	Running RMI System

## An Overview of RMI Applications

### Distributed Object Model

**Remote Object** - an object whose methods can be invoked from another Virtual Machine.

**Remote Interface** - a Java interface that declares the methods of a remote object.

**Remote Method Invocation (RMI)** - the action of invoking a method of a remote interface on a remote object.

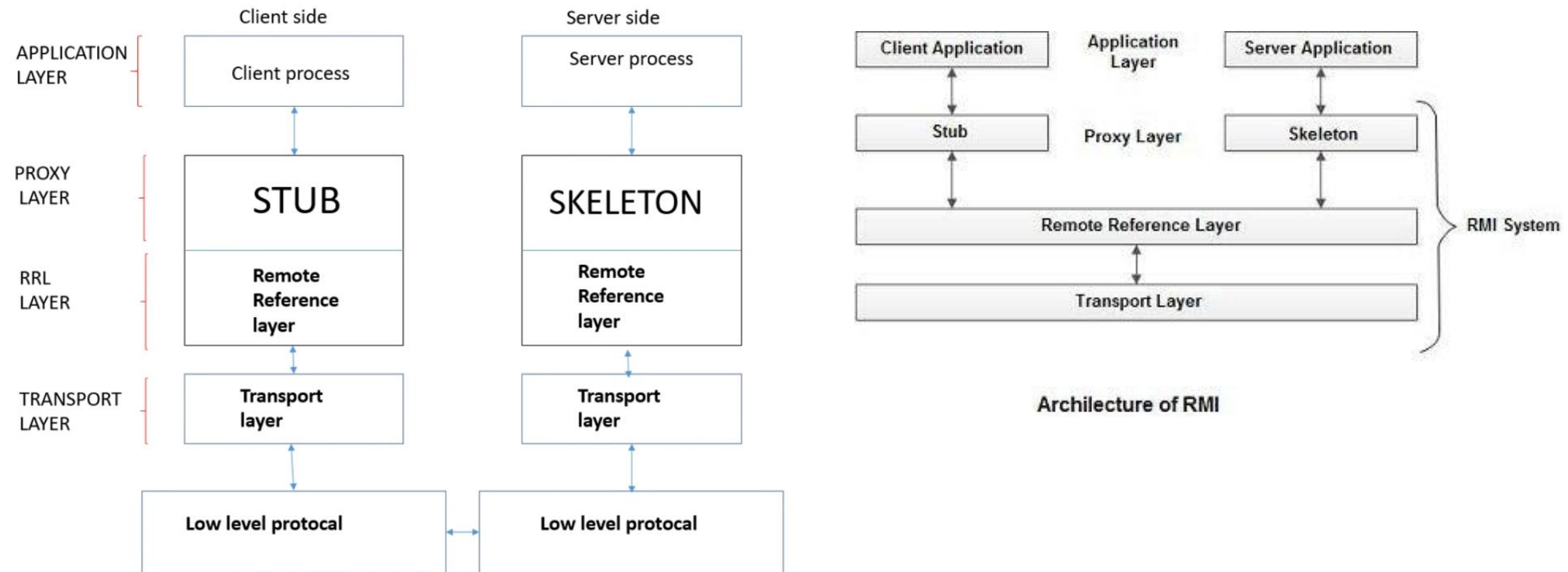
# Unit-12

## RMI

- RMI stands for **Remote Method Invocation**.
- It is an **API** that allows an object residing in one system (JVM) to **access/invoke** an object running on another JVM.
- RMI is used to **build distributed** applications;
- it provides **remote communication** between Java programs using two objects ***stub*** and ***skeleton***.
- It is provided in the package **java.rmi**.

# Unit-12

## RMI Architecture



**Transport Layer** – This layer connects the client and the server. It manages the existing connection and also sets up new connections.

**Stub** – A stub is a representation (proxy) of the remote object at client. It resides in the client system; it acts as a gateway for the client program.

**Skeleton** – This is the object which resides on the server side. **stub** communicates with this skeleton to pass request to the remote object.

**RRL(Remote Reference Layer)** – It is the layer which manages the references made by the client to the remote object.

# Unit-12

## RMI

### APPLICATION LAYER

- Responsible for running client and server applications
- Here client application invokes methods defined by server application
- When client invokes a method then request is passed to proxy layer

### PROXY LAYER

- Responsible for creating client stub at client side by packing the request msgs sent by client process
- Also responsible for creating skeleton by packing response msgs sent by server
- Once stub and skeleton are created they are passed to RRL

### REMOTE REFERENCE LAYER (RRL)

- Checks semantics and remote references used by client process using remote reference protocol
- Finally RRL transmits msgs and data to RMI transport layer

### TRANSPORT LAYER

- Responsible for establishing and maintaining stream oriented connection between client and server.
- Also responsible for managing send/receive of request/reply msgs between client and server



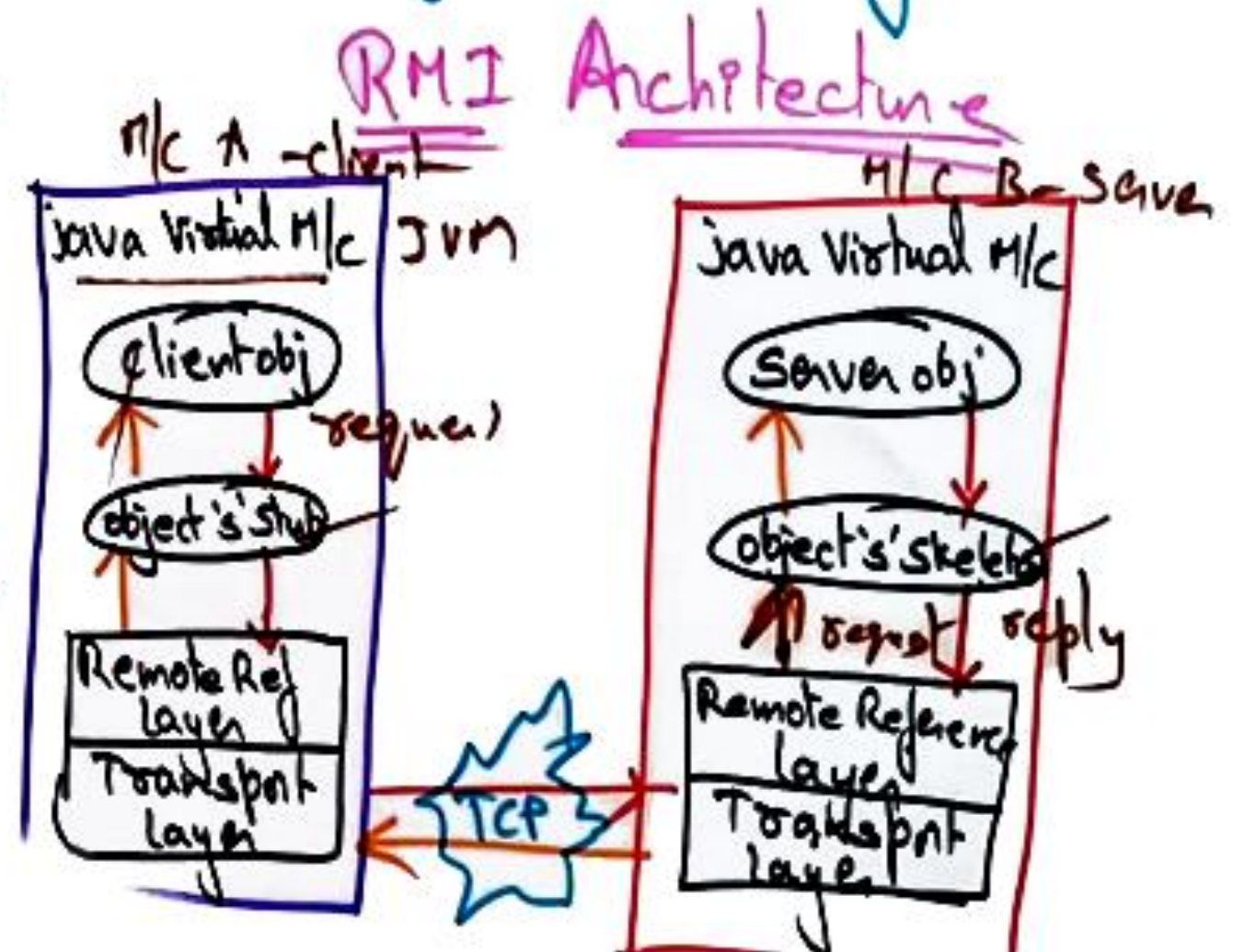
## Unit-12

# RMI Remote Method Invocation

The RMI system consists of three layers in its architecture

- Stub / Skeleton layer
- Remote Reference layer
- Transport layer

3 layers



## RMI architecture

# An Overview of RMI Applications

RMI uses stub and skeleton object for communication with the remote objects.

\* Remote object is an object whose method can be invoked from another JVM.

Stub :-  $\Rightarrow$  client-side

- It is an object, acts as a gateway for the client side.  
All outgoing requests are routed through it.
- It resides at the client side and represents the remote object.

client  $\xrightarrow{\text{invoke()}}$  server



## An Overview of RMI Applications

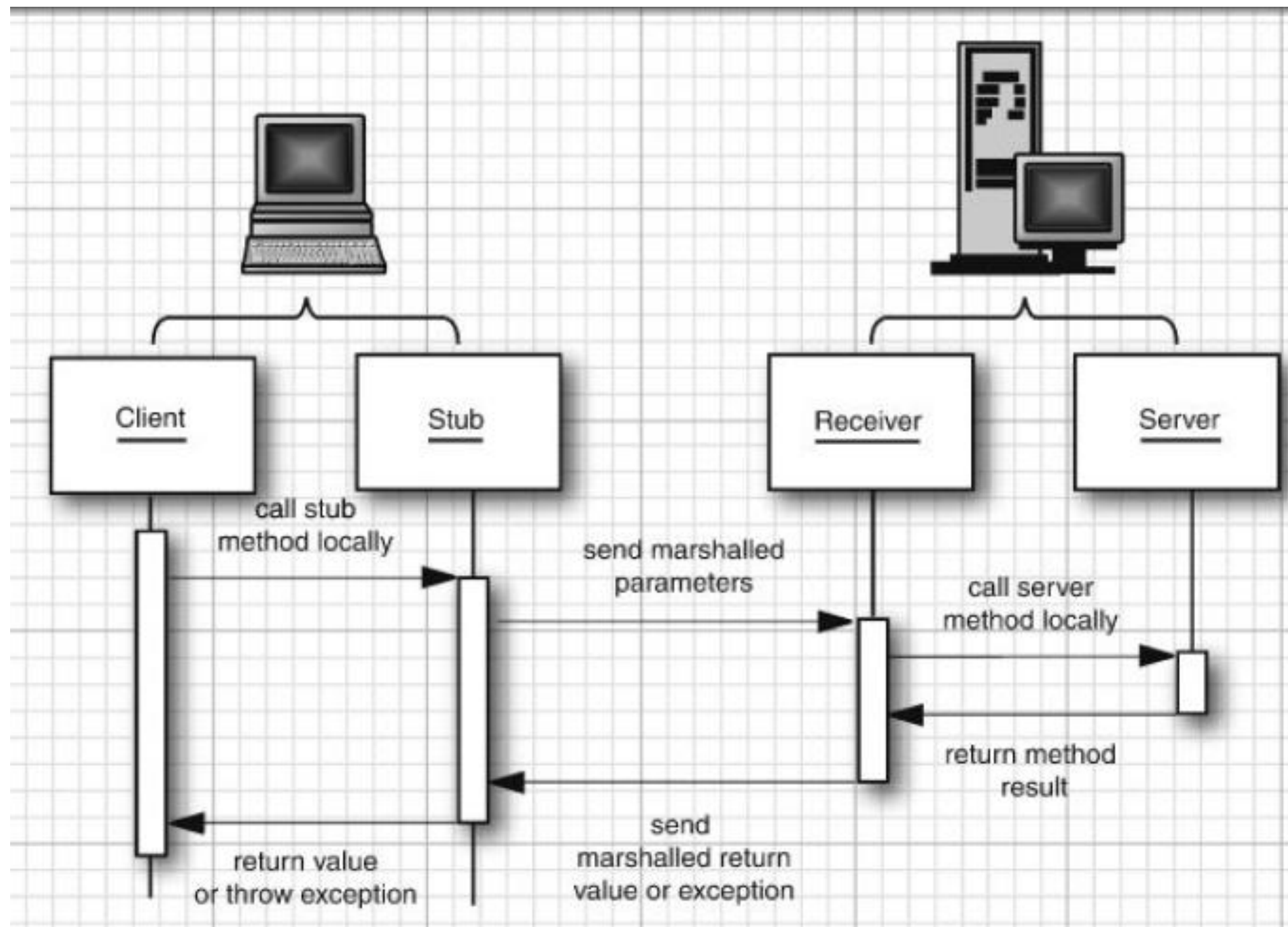
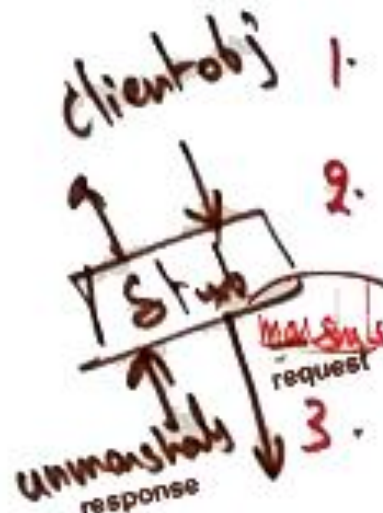


Figure. Parameter Marshalling

## Unit-12

# An Overview of RMI Applications

→ when a caller invokes method on the stub object, it does,

- 
- The diagram shows a box labeled 'Stub' with an arrow pointing to it from 'clientobj'. An arrow labeled 'marshals request' points from the 'Stub' box to the right. An arrow labeled 'unmarshals response' points from the right back to the 'Stub' box.
1. It initiates a connection with JVM.
  2. It writes and transmits (marshals) the parameters to the remote virtual m/c (JVM)
  3. It waits for result. { client waits for result }
  4. It reads (unmarshals) the return value or exception
  5. It finally, returns the value to the caller.



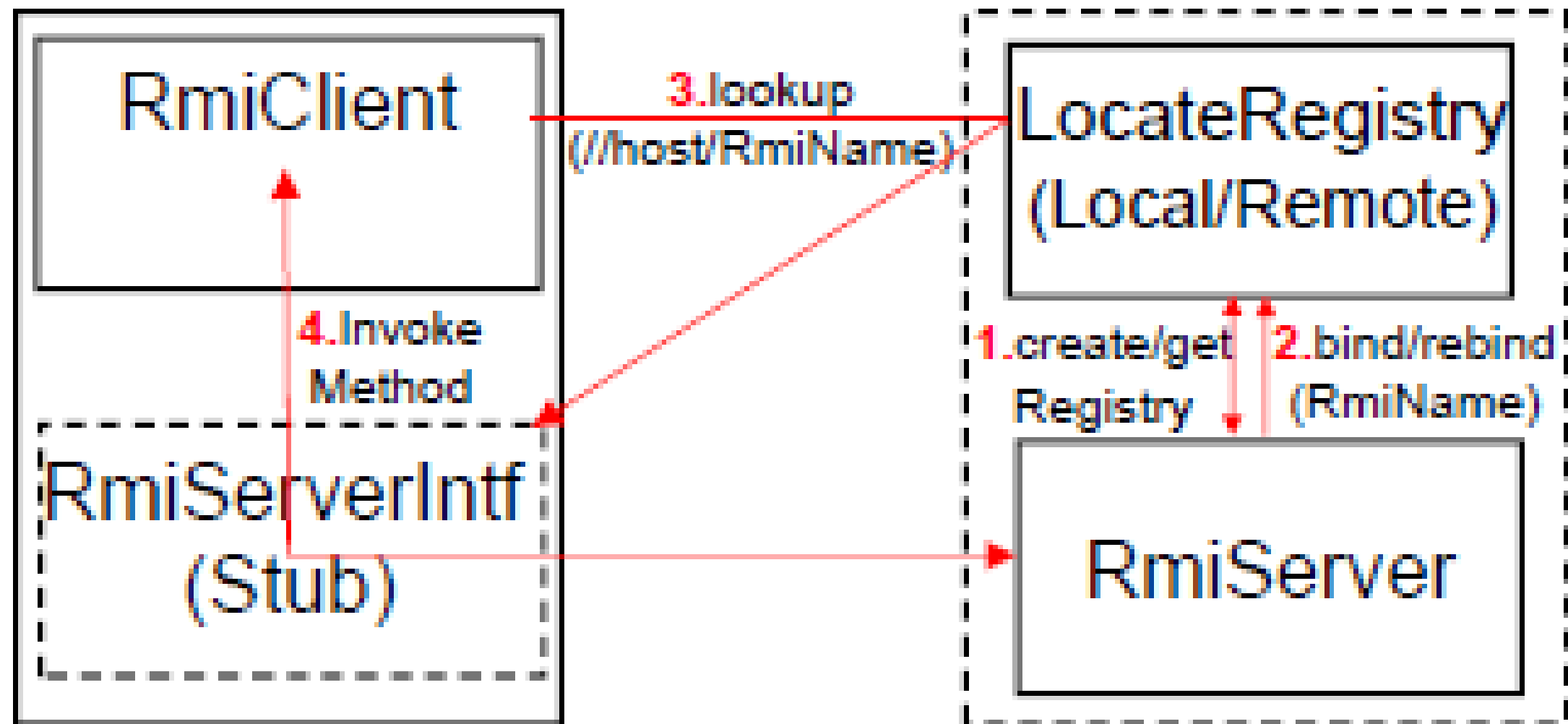
# An Overview of RMI Applications

→ Skeleton :-

- It is an object, acts as a gateway for the server side of object.
- All the incoming requests are routed through it.
- When skeleton receives the incoming request, it does the following tasks:
  1. It reads the parameter for the remote obj
  2. It invokes the method on the actual remote object &
  3. It writes and transmits (marshals) the result to the caller.

## Unit-12

# An Overview of RMI Applications





# Unit-12

## RMI

### 1. Interface

```
import java.rmi.*;
public interface Rem extends Remote{
    public int addNum(int a,int b)throws RemoteException;
    public int subNum(int a,int b)throws RemoteException;
}
```

### 2. Rem\_impt

```
import java.rmi.*;
import java.rmi.server.*;
public class Rem_impt extends UnicastRemoteObject implements Rem{
    Rem_impt()throws RemoteException{}

    public int addNum(int a,int b){return a+b;}
    public int subNum(int a,int b){return a-b;}
}
```

# Unit-12

## RMI

### 3. Server

```
import java.rmi.*;
import java.rmi.registry.*;

public class Server{

    public static void main(String args[]){
        try{
            Rem_impt locobj = new Rem_impt();
            Registry rgsty = LocateRegistry.createRegistry(1888);
            rgsty.rebind("Rem", locobj);
        }catch(RemoteException re){System.out.println(re);}
    }
}
```

# Unit-12

## RMI

### 4. Client

```
import java.rmi.*;
import java.util.*;
import java.rmi.registry.*;

public class Client{
    public static void main(String args[]){
        Scanner sc = new Scanner(System.in);
        System.out.println("Enter first number");
        int a = sc.nextInt();
        System.out.println("Enter first number");
        int b = sc.nextInt();
        try{
            Rem_impt remObj = new Rem_impt();
            Registry rgsty = LocateRegistry.getRegistry(1888);
            rgsty.rebind("Rem", remObj);
            System.out.println("Sum = " + remObj.addNum(a,b));
            System.out.println("Sum = " + remObj.subNum(a,b));
        }catch (RemoteException re){System.out.println(re);}
    }
}
```

# Unit-12

## RMI

RMI	CORBA
1. RMI stands for remote method invocation	1. CORBA stands for Common Object Request Broker Architecture
2. It uses Java interface for implementation.	2. It uses Interface Definition Language (IDL) to separate interface from implementation.
3. CORBA passes objects by value.	3. CORBA passes objects by reference.
4. Java RMI is a server-centric model.	4. CORBA is a peer-to-peer system.
5. RMI uses the Java Remote Method Protocol as its underlying remote communication protocol.	5. CORBA use Internet Inter- ORB Protocol as its underlying remote communication protocol.
6. RMI is slow in execution than CORBA.	6. CORBA is fast in execution than RMI.