CS6308- Java Programming

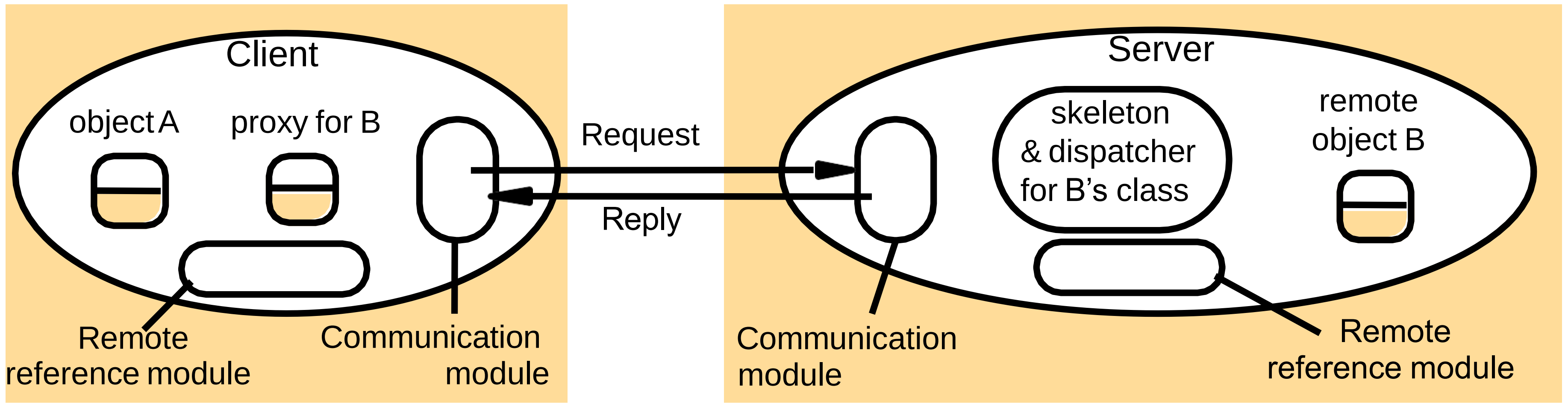
V P Jayachitra
Assistant Professor
Department of Computer Technology
MIT Campus
Anna University

Remote Method Invocation (RMI)

- RMI allows a Java object that executes on one machine to invoke a method of a Java object that executes on another machine.
- RMI is a java API that allows an application to invoke methods on an object that resides/executes in remote machine, as if the object is local.
- RMI abstracts the complexity of network communication and enables seamless interaction between objects in distributed applications.
- RMI is that it handles all the complex networking and communication details behind the scenes, making remote method calls appear as simple as local ones to the developer.

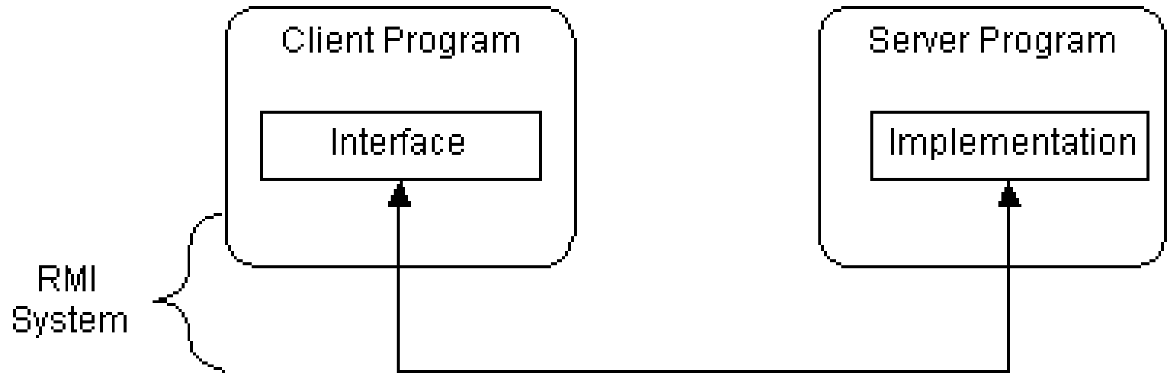
Why?

- Allows object to invoke methods on remote objects using local invocation.
- Supports communication between different VMs, potentially across the network.

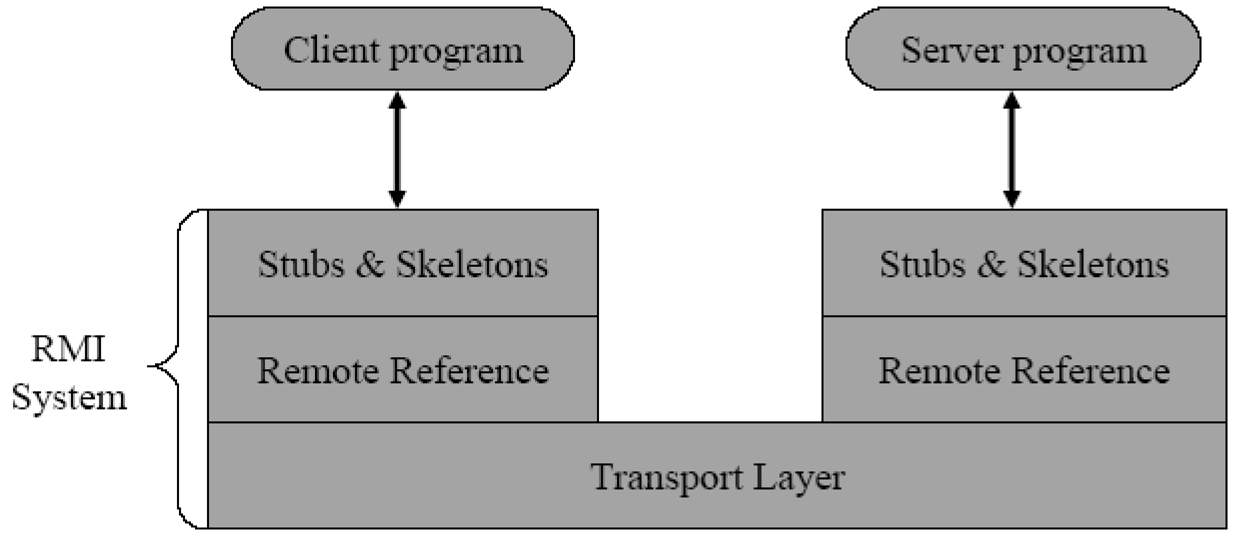


Principle of RMI

- RMI separates:
  - Definition of behaviour
  - Implementation of that behaviour
- Each of them is allowed to run on different JVMs
- Interfaces (define definition) resides on client side
- Classes (define implementation) resides on server machine



RMI architecture



Stub

- Represents the remote service implementation in the client (is a proxy)
- During communication between two machines through RPC or RMI, **parameters are packed into a message and then sent over the network**.
- This packing of parameters into a message is called marshalling. On the other side these packed parameters are unpacked from the message which is called unmarshalling.
- Marshalls parameters :
  - Encoding parameters
    - Primitive Type (integer, Byte, …) : copy by value
    - Reference Type (String, Object, …) : object copy
  - Information block from stub to skeleton
    - Remote object’s identifier
    - Parameters / the ID of method
- Unmarshalls return value or exception

Skeleton

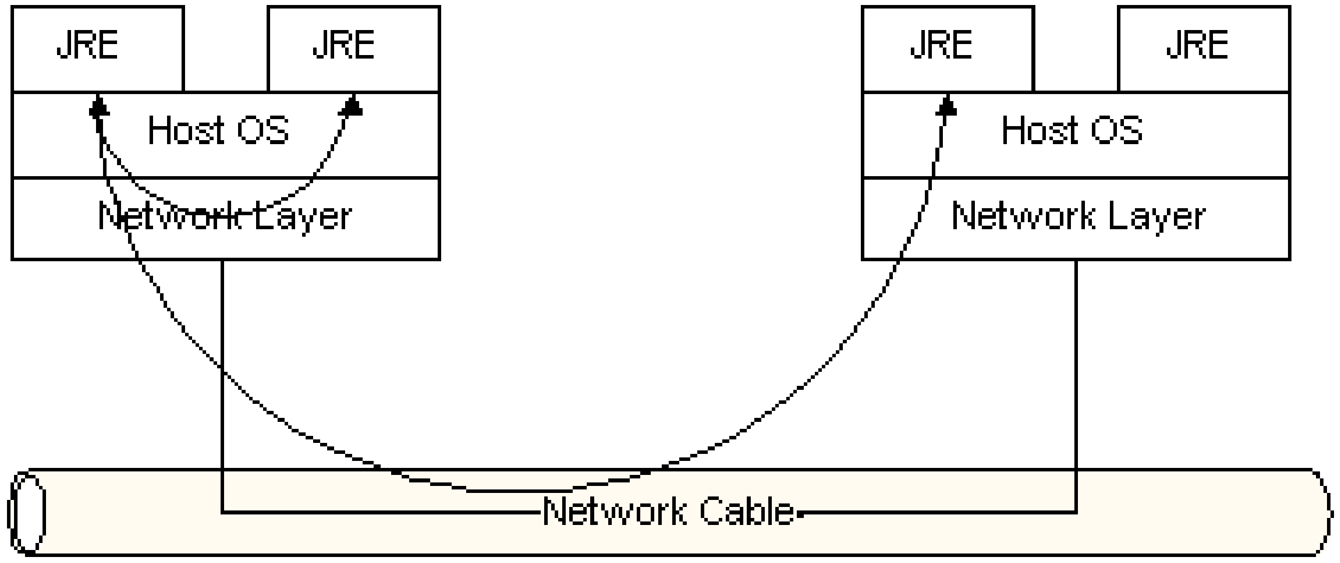
- Helper class on server
- Generated for RMI to use
- Communicates with stub across the link
- Reads parameters for the method call from the link
- Makes the call to the service object
- Accepts the return value, writes it back to the stub

Remote Reference Layer

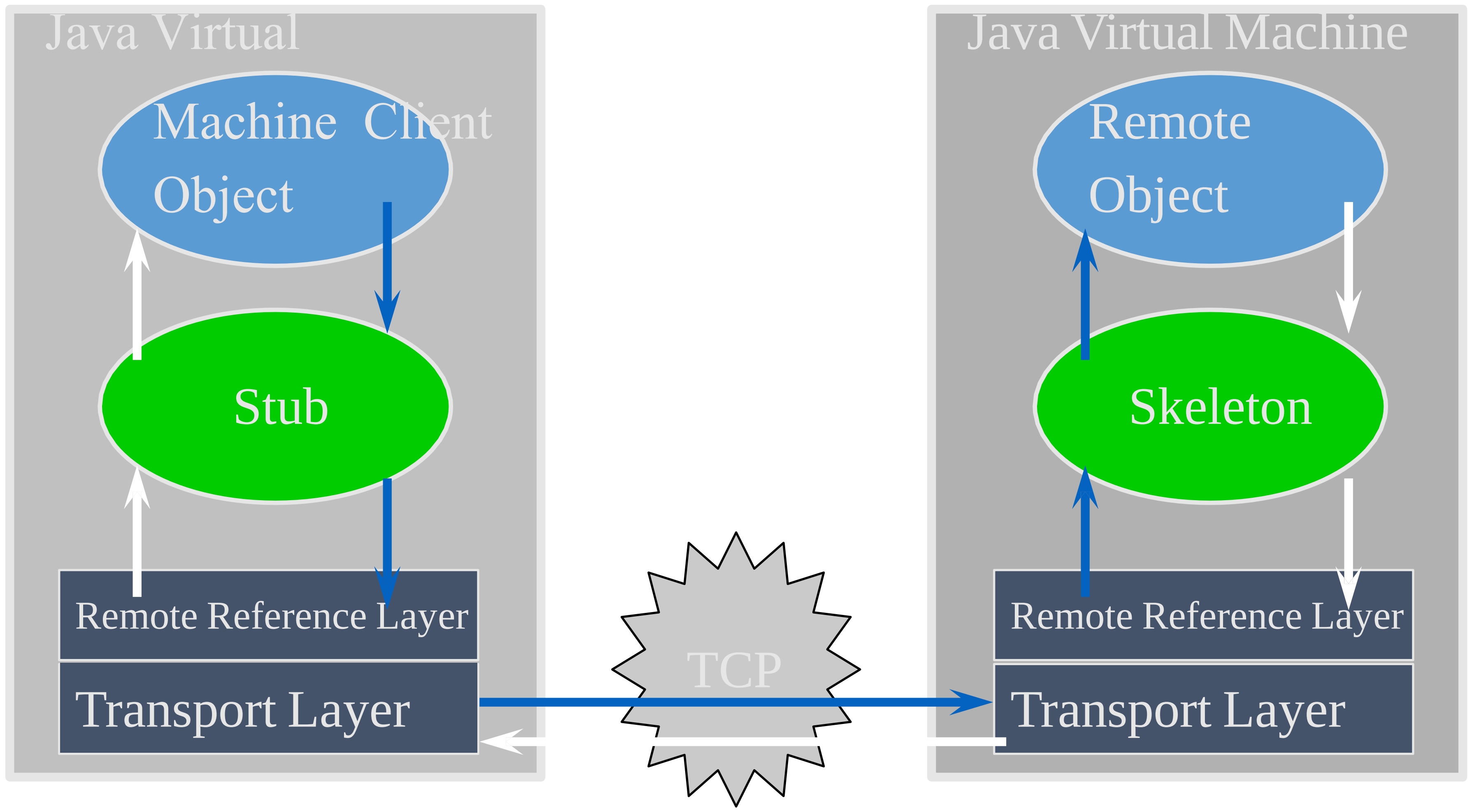
- Exists in both the RMI client and server
- Provides a constant interface to the stubs and skeletons
- Manages communication between stubs/skeleton
- Manages references to remote objects
  - Threading, garbage collection …
- Manages reconnection strategies if an object should become unavailable

Transport Layer

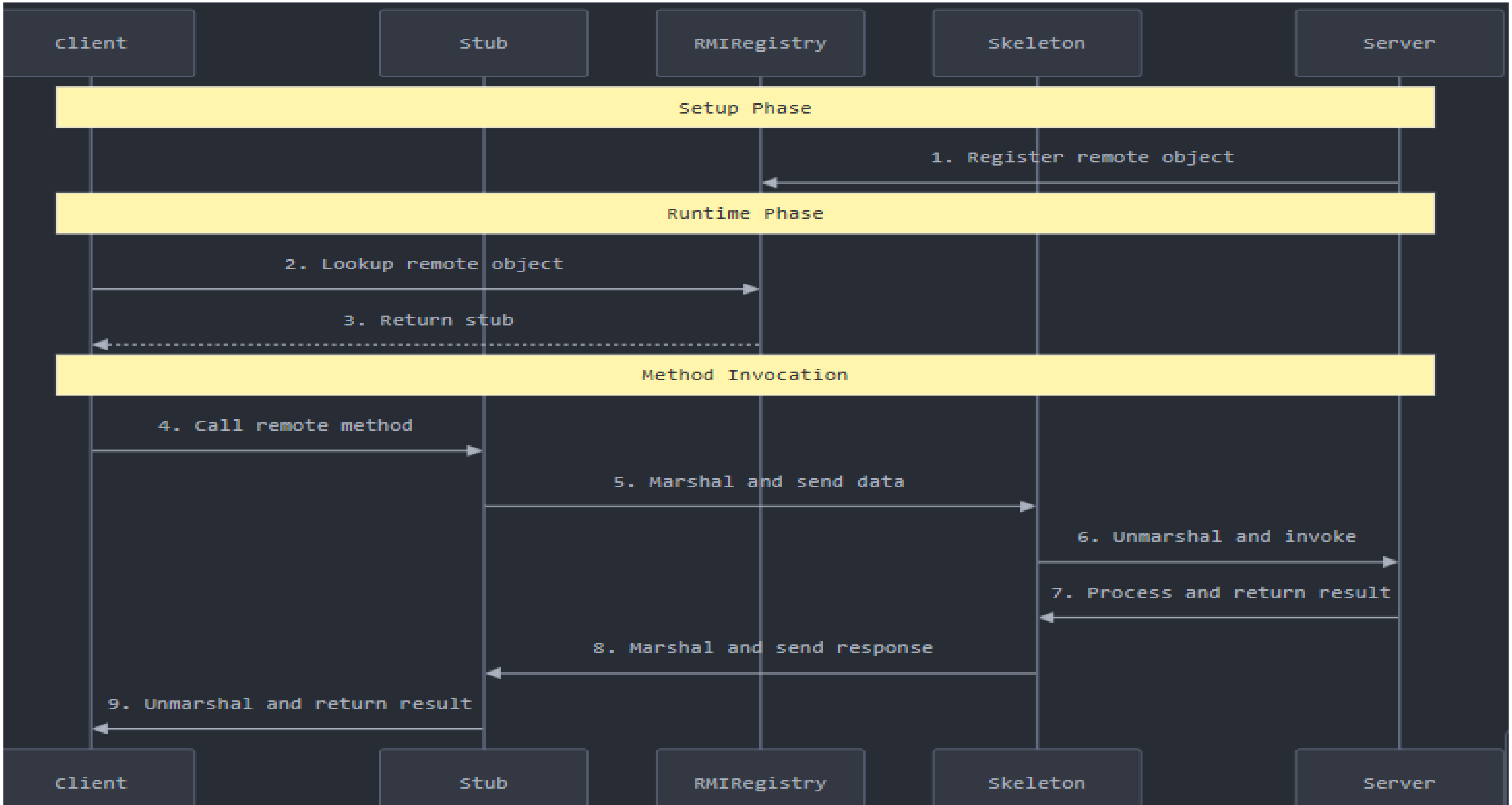
- Stream-based network connections that use TCP/IP
- Deals with communications
- For interoperability, RMI may use the OMG Internet Inter-ORB Protocol (IIOP)



RMI Layers


Java Virtual
Machine Client
Object
Stub
Remote Reference Layer
Transport Layer
TCP
Java Virtual Machine
Remote
Object
Skeleton
Remote Reference Layer
Transport Layer

RMI INTERACTION DIAGRAM

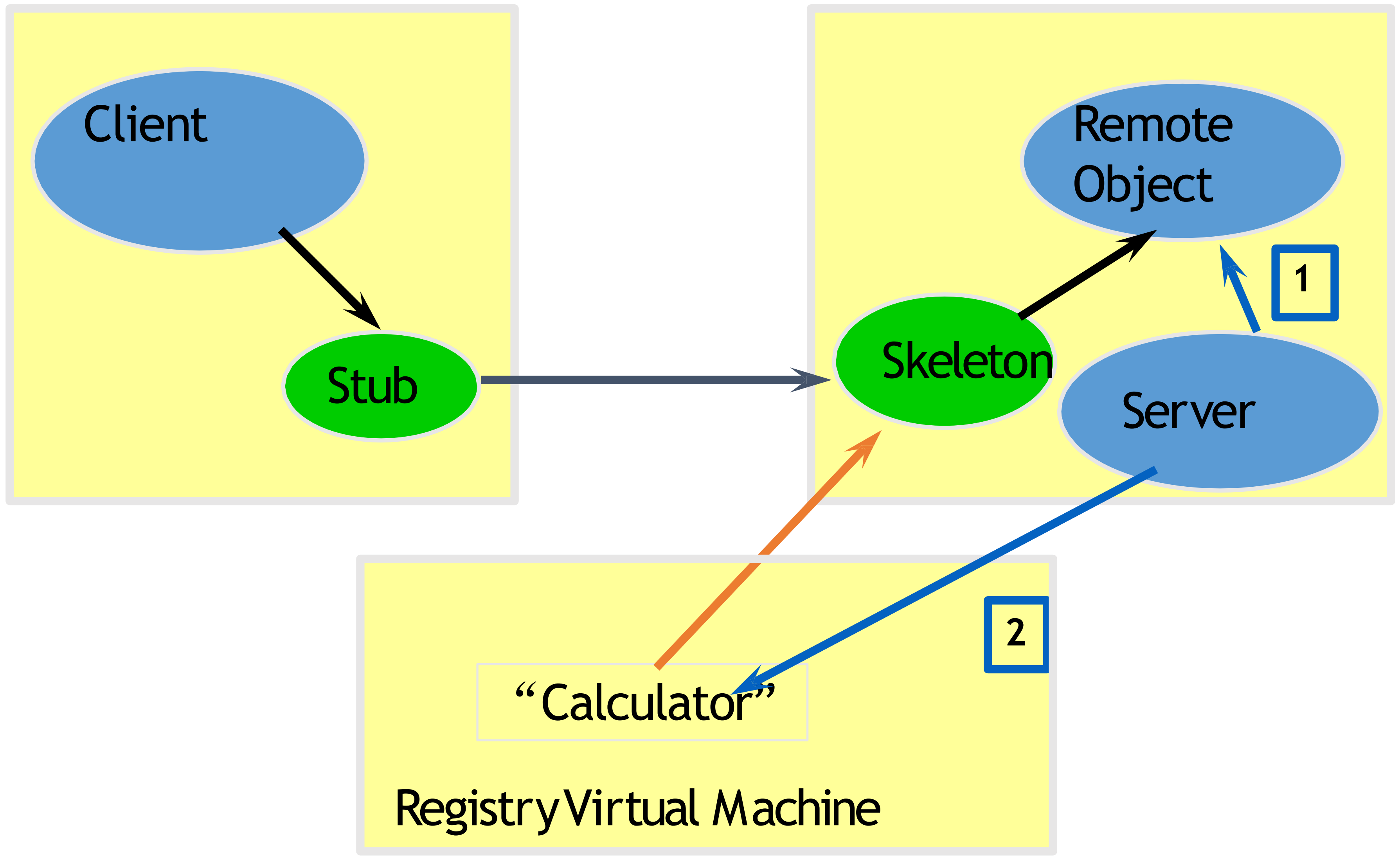


Naming Remote Objects

- How does a client find an RMI remote service?
  - Clients find remote services by using a naming or directory service, running on a well known host and port number
- RMI
  - can use different directory services, e.g. the Java Naming and Directory Service (JNDI)
  - includes simple service called RMI Registry (**rmiregistry**, default on port 1099)

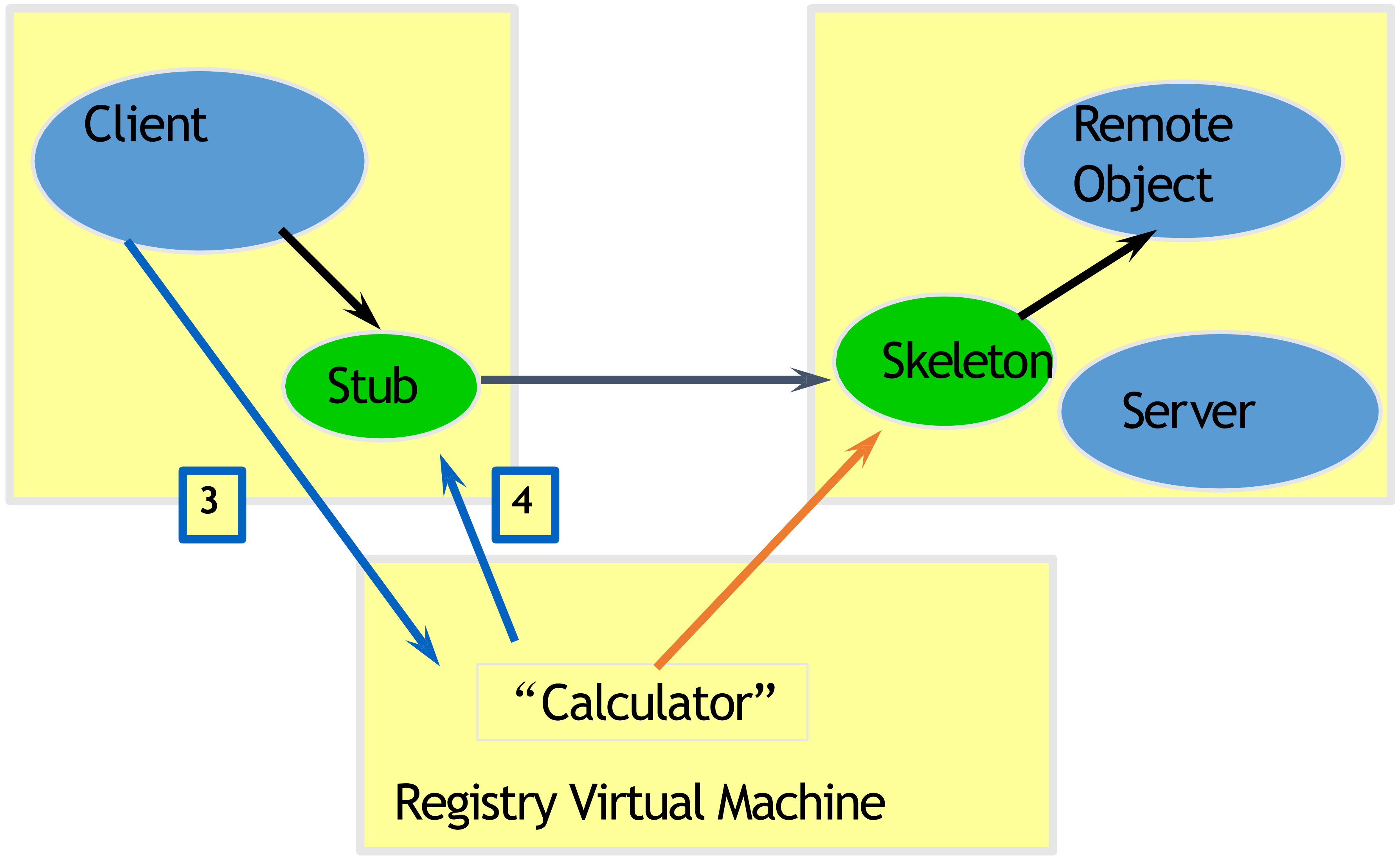
RMI Flow

1. Server Creates Remote Object
2. Server Registers Remote Object



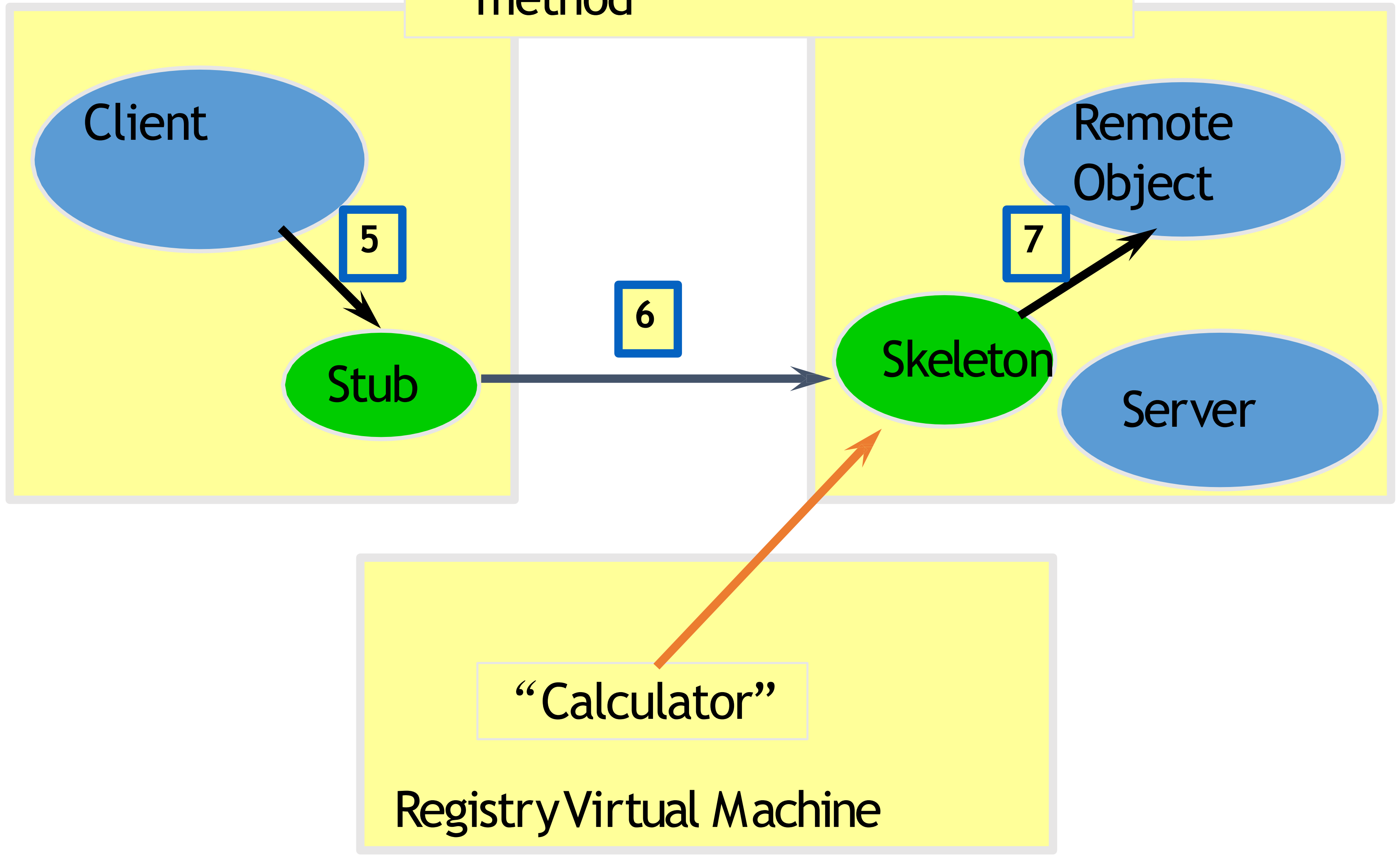
RMI Flow

3. Client requests object from Registry
4. Registry returns remote reference (and stub gets created)



RMI Flow

5. Client invokes stub method
6. Stub talks to skeleton
7. Skeleton invokes remote object method



Example code: step 1 Creating Remote Object

- Define a Remote Interface
    - extends java.rmi.Remote

```
interfaceAdder extends Remote
{
    public int add(int x, int y) throws RemoteException
}
```

Example code: step 1 Creating Remote Object

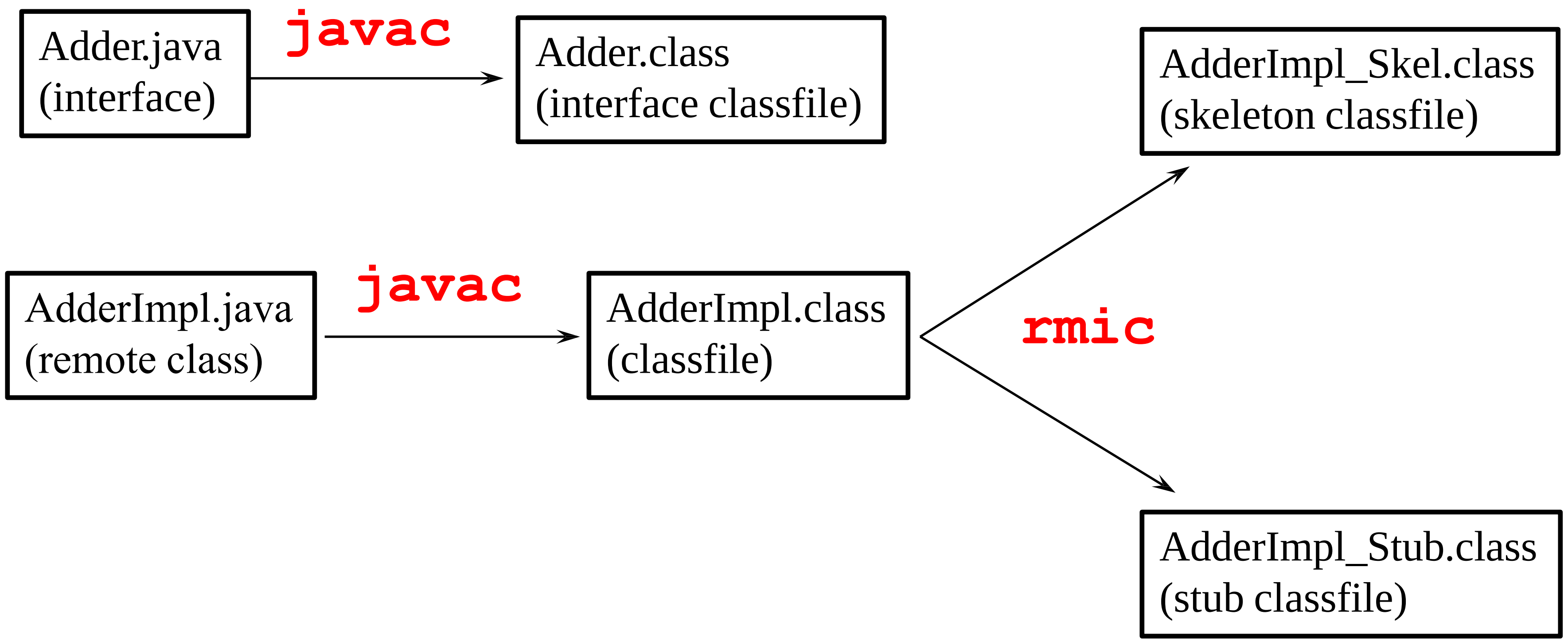
- Define a class that implements the Remote Interface
    - extends java.rmi.RemoteObject
    - or java.rmi.UnicastRemoteObject

```
classAdderImpl extends UnicastRemoteObject implementsAdder
{
  public AdderImpl() throws RemoteException
   {
     super();
   }
   public int add(int x, int y) throws RemoteException
   {
      return x + y;
   }
}
```

constructor calls super() which:

- Creates a random TCP port for this object
- Sets up networking infrastructure
- Creates server-side socket listeners

Compiling Remote Classes



Registering Remote Classes

- Start the registry
  - running process
- Unix:

```
rmiregistry &
```

- Windows:

```
start /m rmiregistry
```

- Remote object code in server

```
//Server
AdderImpl a1 = new AdderImpl(“Add”);
Naming.bind(“Add”, a1);
```

- Remote reference code in client

```
//Client
String url = “rmi://hostName/”;
Adder a = (Adder) Naming.lookup(url + “Add”);
```

RMI Client Example

```
String url = “rmi://hostName/”;
Adder a = (Adder) Naming.lookup(url + “Add”);

int sum = a.add(2,2);
System.out.println("2+2=" + sum);
```

RMI benefits

- Safe and Secure
  - RMI uses built-in Java security mechanisms
- Easy to Write/Easy to Use
  - A remote interface is an actual Java interface
- Distributed Garbage Collection
  - Collects remote server objects that are no longer referenced by any client in the network

Icalculator.java

```java
import java.rmi.Remote;
import java.rmi.RemoteException;
public interface ICalculator extends Remote{
    public int add(int a, int b) throws RemoteException;
    }
```

CalculatorImpl.java

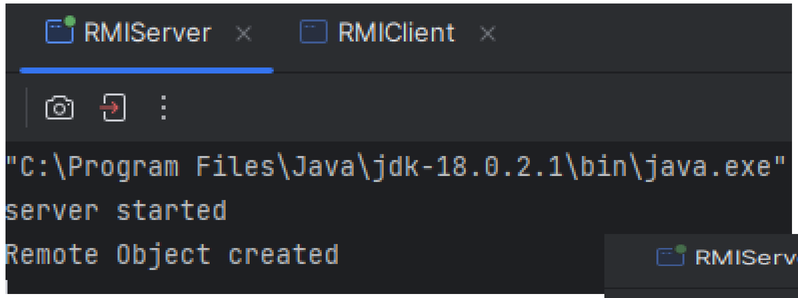
```java
import java.rmi.server.UnicastRemoteObject;
import java.rmi.Remote;
import java.rmi.RemoteException;
class CalculatorImpl extends UnicastRemoteObject implements
ICalculator {
    public CalculatorImpl() throws RemoteException{
        super();    }
    public int add(int a, int b) throws RemoteException{
        return a+b;    }
}
```

RMIClient.java

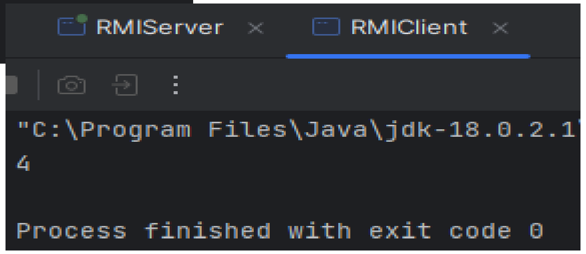
```java
import java.rmi.registry.LocateRegistry;
import java.rmi.registry.Registry;
public class RMIClient {
    public static void main(String[] args){
        try{
            Registry registry= LocateRegistry.getRegistry("localhost", 1099);
            ICalculator obj=(ICalculator) registry.lookup("cal");
            System.out.println(obj.add(2,2));
        }catch(Exception e){
            System.out.println(e.getMessage());        }
}}
```

RMIServer.java

```java
import java.rmi.registry.Registry;
import java.rmi.registry.LocateRegistry;
public class RMIServer {
    public static void main(String[] args){
        try{
            Registry registry=LocateRegistry.createRegistry(1099);
            System.out.println("server started");
            ICalculator  obj=new CalculatorImpl();
            registry.rebind("cal",obj );
            System.out.println("Remote Object created");
        }
        catch(Exception e){
            System.out.println(e.getMessage());
        }
    }
}
```



```
"C:\Program Files\Java\jdk-18.0.2.1\bin\java.exe"
server started
Remote Object created
```



```
"C:\Program Files\Java\jdk-18.0.2.1
4

Process finished with exit code 0
```

Client program

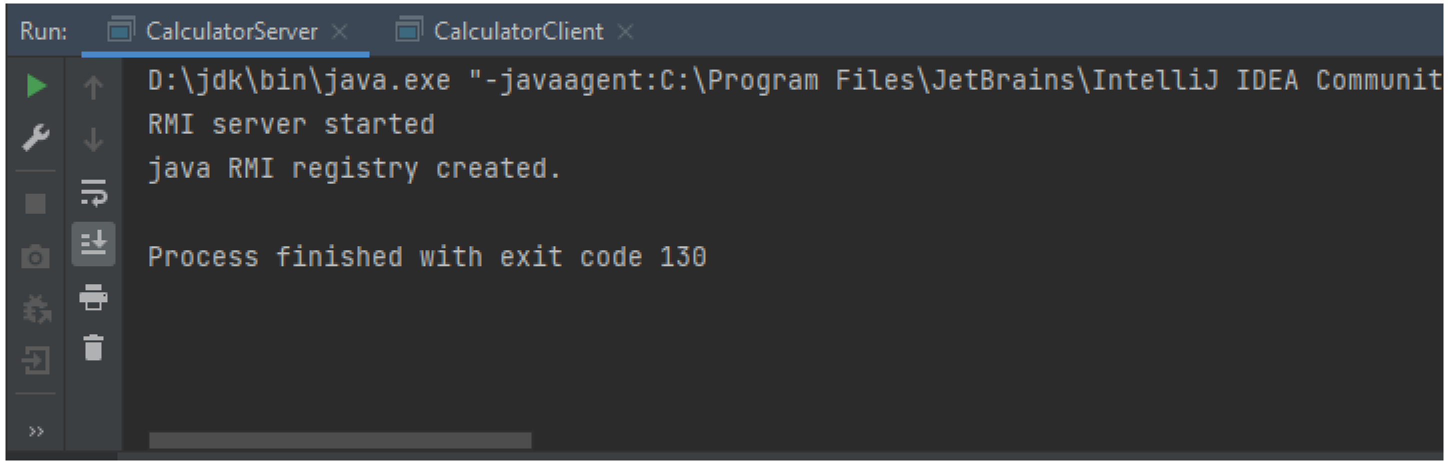
```java
import java.rmi.Naming;
import java.rmi.RemoteException;
import java.net.MalformedURLException;  import java.rmi.NotBoundException;

public class CalculatorClient {
    public static void main(String[] args) {
    try {
            Calculator c = (Calculator)Naming.lookup("rmi://localhost/CalculatorService");
            [illegible]
             System.out.println( c.mul(4, 6) );  System.out.println( c.div(12, 3) );
    }
   catch (MalformedURLException murle) {
             System.out.println("MalformedURLException");  System.out.println(murle);
                          }
                          catch (RemoteException re) { System.out.println(
                                  "RemoteException");  System.out.println(re);
                          }
                          catch (NotBoundException nbe) { System.out.println(
                                  "NotBoundException");  System.out.println(nbe);
                          }
                          catch (java.lang.ArithmeticException ae) {
                             System.out.println(java.lang.ArithmeticException"); System.out.println(ae);
                          }}}
```

Server program

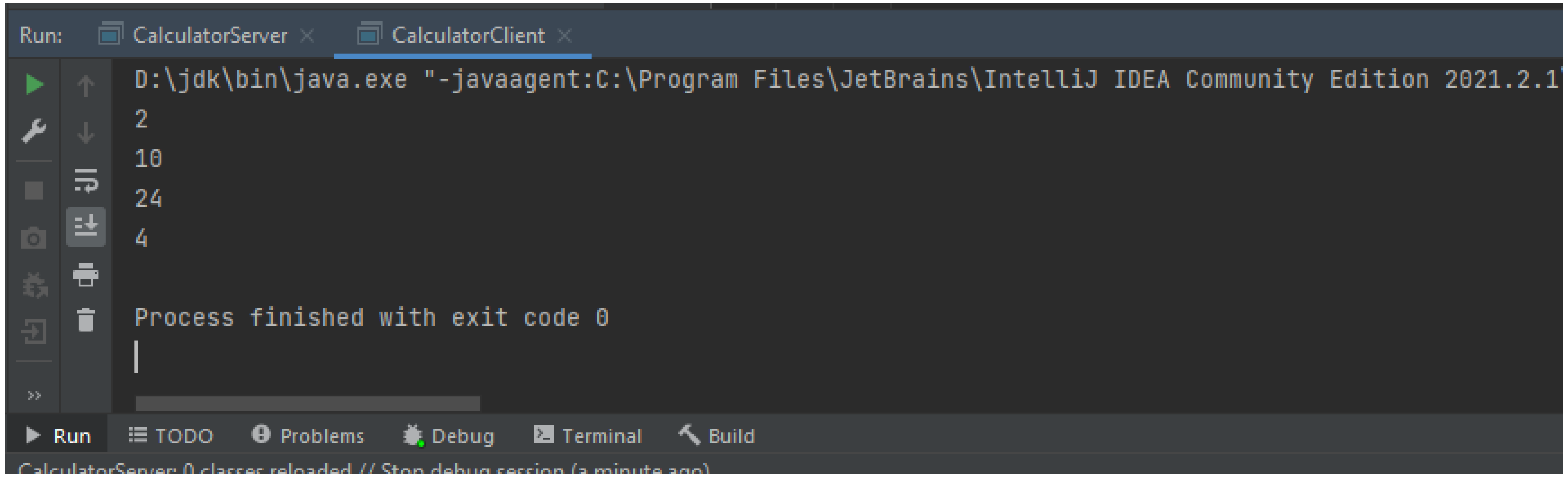
```java
import java.rmi.Naming;
import java.rmi.RMISecurityManager;
import java.rmi.RemoteException;
import java.rmi.registry.LocateRegistry;

public class CalculatorServer {

    public CalculatorServer() {
        System.out.println("RMI server started");

        try {
            LocateRegistry.createRegistry(1099);
            System.out.println("java RMI registry created.");
        } catch (RemoteException e) {
            e.printStackTrace();
        }

        try {
            Calculator c = new CalculatorImpl();
            Naming.rebind("rmi://localhost/CalculatorService", c);
        } catch (Exception e) {
            System.out.println("Trouble: " + e);
        }
    }

    public static void main(String args[]) {
        new CalculatorServer();
    }
}
```

Interface and Implementation program

```
public interface Calculator  extends java.rmi.Remote {
    public long add(long a, long b)throws java.rmi.RemoteException;
    public long sub(long a, long b)throws java.rmi.RemoteException;
    public long mul(long a, long b)throws java.rmi.RemoteException;
    public long div(long a, long b)throws java.rmi.RemoteException;
}
public class CalculatorImpl extends java.rmi.server.UnicastRemoteObject  implements Calculator {
public CalculatorImpl() throws java.rmi.RemoteException {
    super();
}
public long add(long a, long b) throws java.rmi.RemoteException {
    return a + b;
}
public long sub(long a, long b) throws java.rmi.RemoteException {
       return a - b;


 }
 public long mul(long a, long b) throws java.rmi.RemoteException {
     return a * b;
 }
 public long div(long a, long b)  throws java.rmi.RemoteException {
     return a / b;
 }
```



```
D:\jdk\bin\java.exe "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA Communit
RMI server started
java RMI registry created.

Process finished with exit code 130
```



```
D:\jdk\bin\java.exe "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA Community Edition 2021.2.1
2
10
24
4

Process finished with exit code 0
```

CalculatorServer: 0 classes reloaded // Stop debug session (a minute ago)