Lecture 4 – RPC/RMI

Topics

- Introduction to Distributed Objects and Remote Invocation
- Remote Procedure Call
- Java RMI

Two main ways to do DC (apart from socket programming)

- Remote Method Invocation (RMI)
  - Local object invokes methods of an object residing on a remote computer
  - Invocation as if it was a local method call
- Event-based Distributed Programming
  - Objects receive asynchronous notifications of events happening on remote computers/processes

Remote Procedure Call (RPC)

- Objects that can receive remote method invocations are called remote objects and they implement a remote interface.
- Programming models for distributed applications are:
  - Remote Procedure Call (RPC)
    - Client calls a procedure implemented and executing on a remote computer
    - Call as if it was a local procedure

RPC Interfaces

- Interfaces for RPC
  - An explicit interface is defined for each module.
  - An Interface hides all implementation details.
  - Accesses the variables in a module can only occur through methods specified in interface.

Remote Procedure Call (RPC)

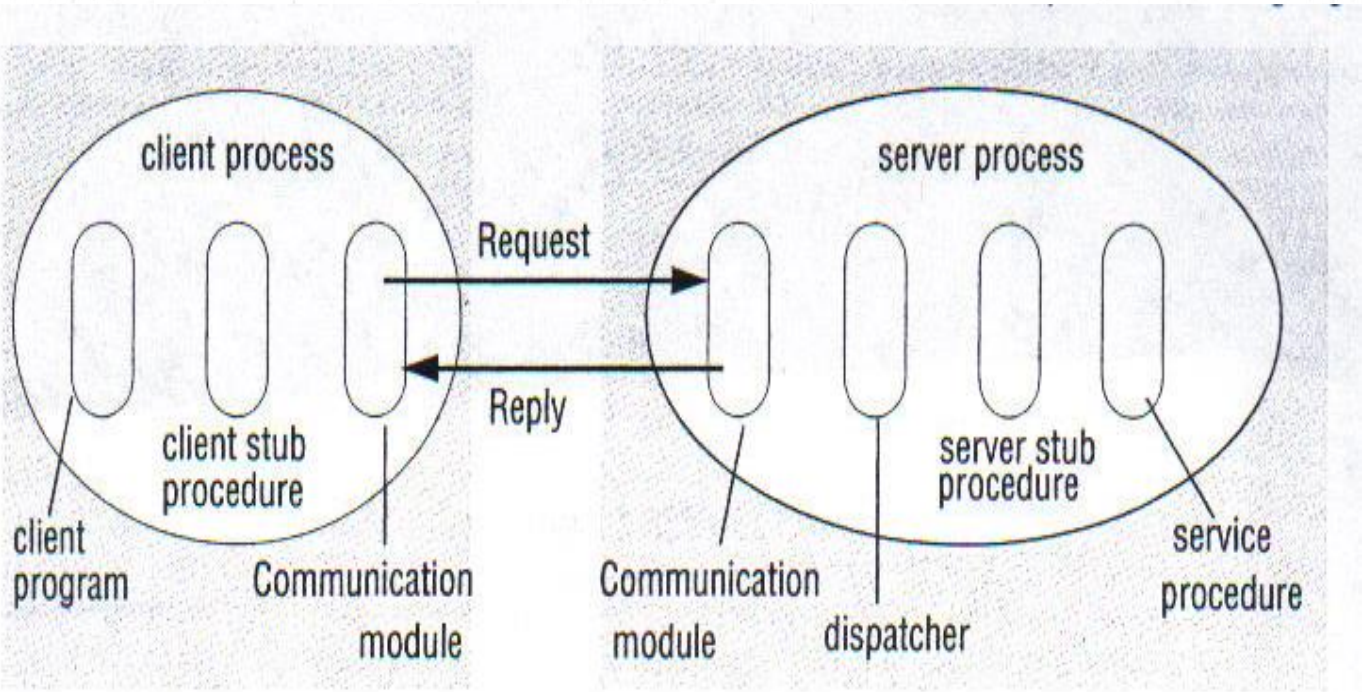


Figure 3. Role of client and server stub procedures in RPC in the context of a procedural language

Couloris,Dollimore and Kindberg Distributed Systems: Concepts & Design Edn. 4 , Pearson Education 2005

SUN ONC RPC

- RPC only addresses procedure calls.
- RPC is not concerned with objects and object references.
- A client that accesses a server includes one stub procedure for each procedure in the service interface.
- A client stub procedure is similar to a proxy method of RMI (discussed later).
- A server stub procedure is similar to a skeleton method of RMI (discussed later).

Strength and Weaknesses of RPC

- RPC (or even RMI) is not well suited for adhoc query processing. (e.g. SQL queries)
- It is not suited for transaction processing without special modification.
- A separate special mode of querying is proposed – Remote Data Access (RDA).
- RDA is specially suited for DBMS.
- In a general client_server environment both RPC and RDA are needed.

RMI

Java Remote Method Invocation

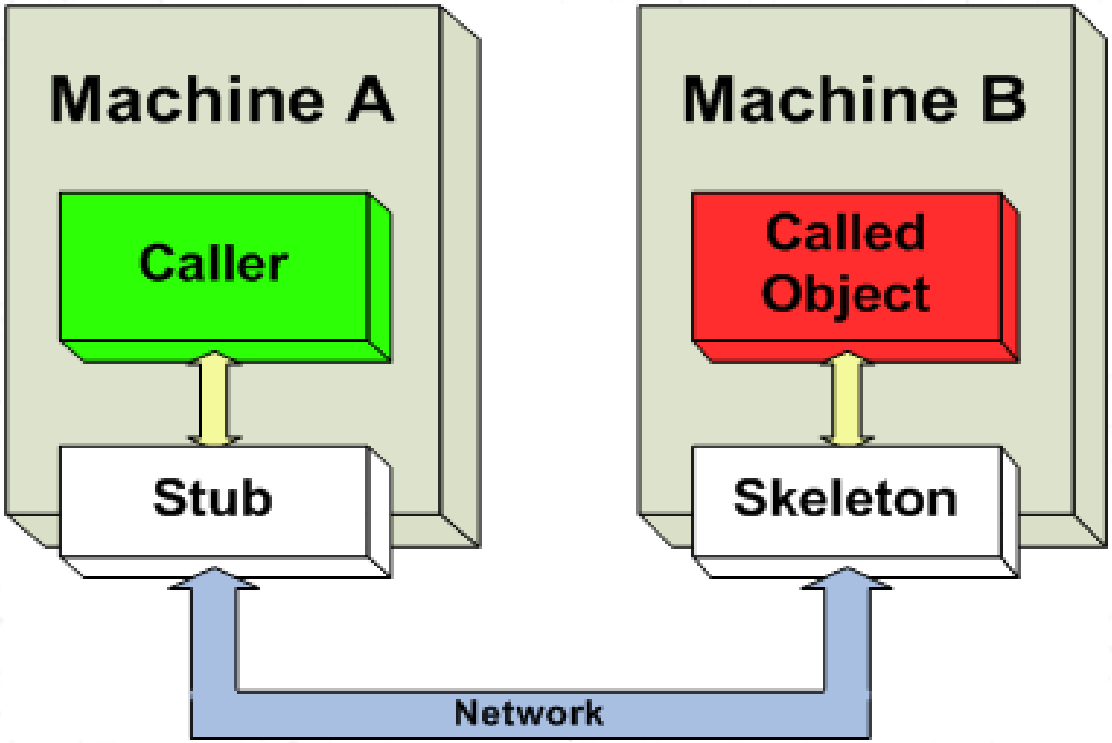


Fig: Distributed Object Technology

Java Remote Method Invocation

- RMI Server, client, interface, stubs, skeletons
- RMI Registry
- Objects + RPC = RMI
- Method Invocation between different JVMs
- Java RMI API
- JRMP (Java Remote Method Protocol)
- Java object serialization
- Parameter Marshalling

RMI System Architecture

Lets divide into two perspectives:

- Layered Structure
- Working Principles

RMI Layered Structure

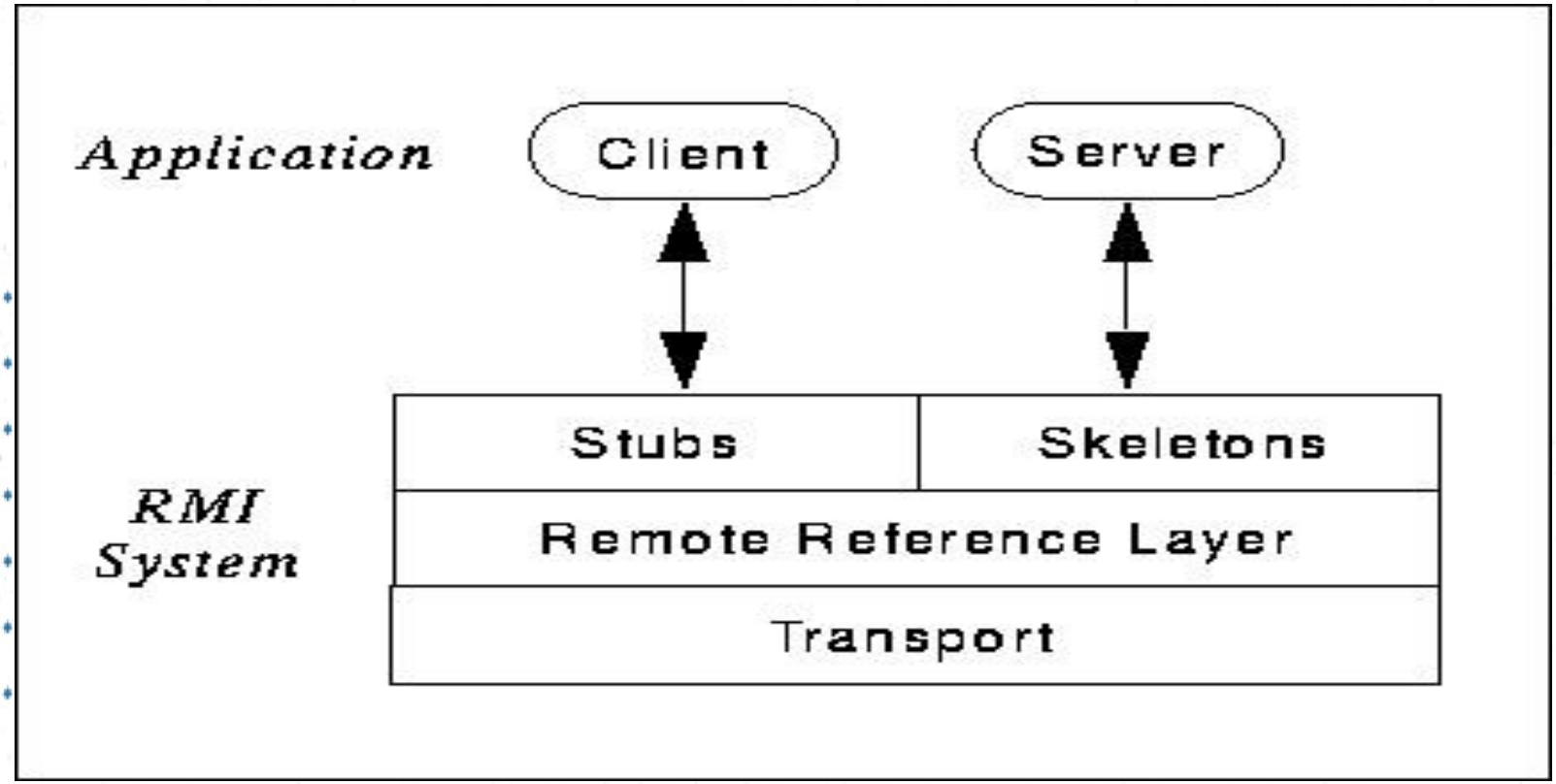


Fig: RMI Layered Structure

RMI Layered Structure

- Application layer: Server, Client
- Interface: Client stub, Server skeleton
- Remote Reference layer: RMI registry
- Transport layer: TCP/IP

RMI Working Principles

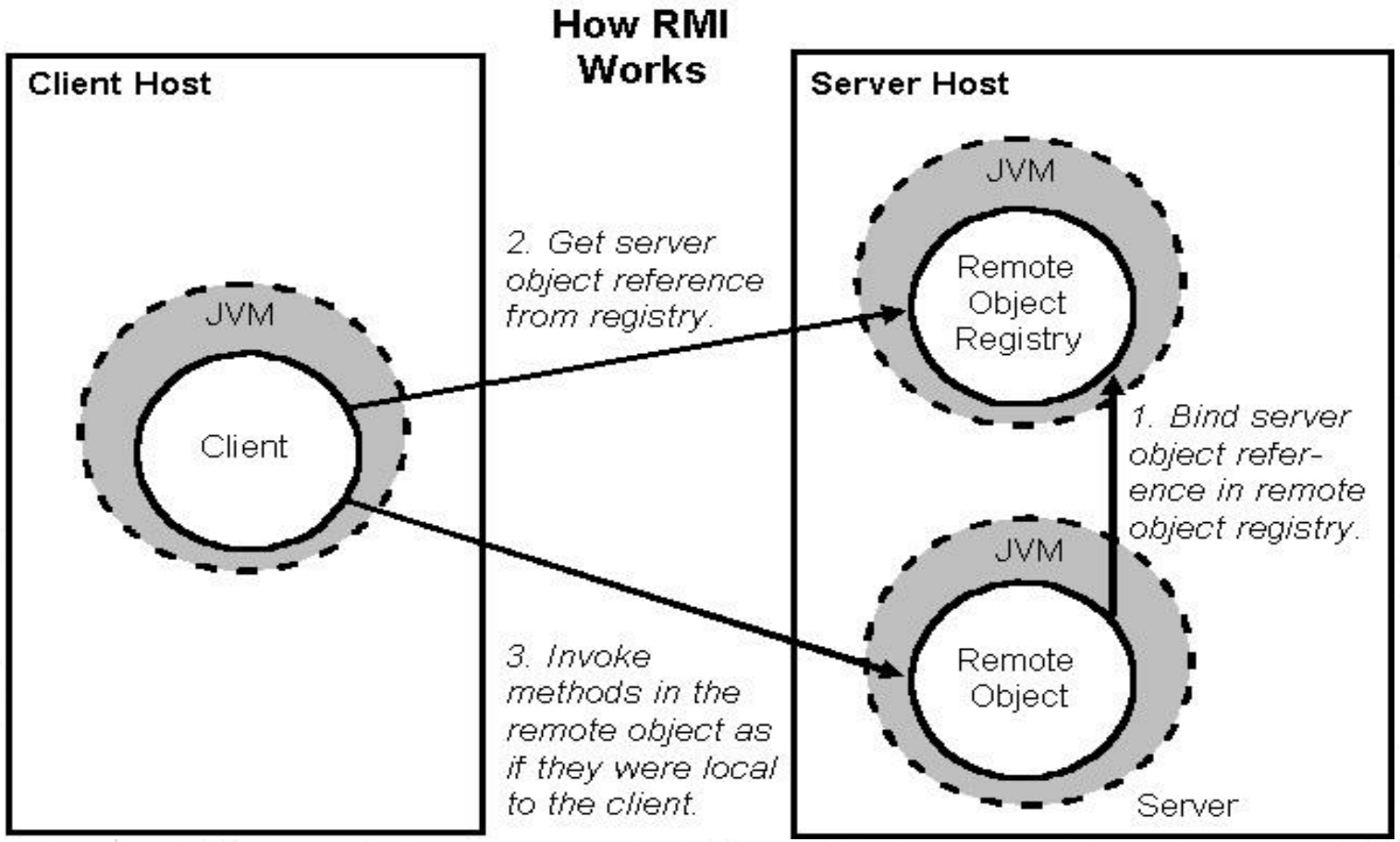
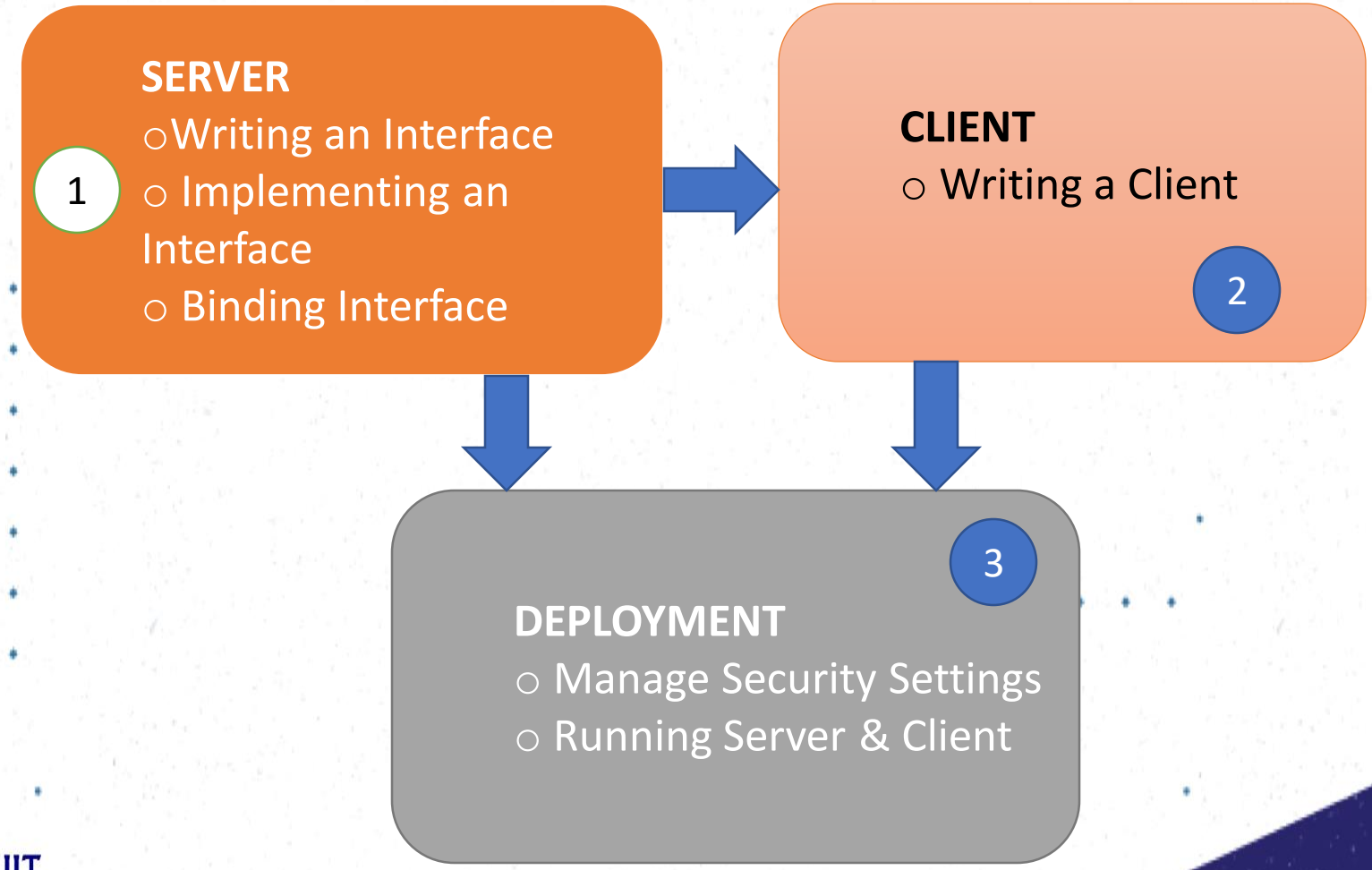


Fig: RMI Working principles

Ready to Develop One?

A Simple RMI Application



Service Interface: An Agreement Between Server & Client

- Factorial Operation

```
public long factorial(int number) throws
    RemoteException;
```

- Check Prime Operation

```
public boolean checkPrime(int number) throws
    RemoteException;
```

```
public BigInteger square(int number) throws
    RemoteException;
```

Server Application: Writing a Service Interface

```
//interface between RMI client and server

import java.math.BigInteger;
import java.rmi.*;

public interface MathService extends Remote {

  // every method associated with RemoteException
  // calculates factorial of a number
  public long factorial(int number) throws
       RemoteException;

  // check if a number is prime or not
  public boolean checkPrime(int number) throws
       RemoteException;

  //calculate the square of a number and returns
       BigInteger
  public BigInteger square(int number) throws
       RemoteException;


}
```

Fig: *MathService* Interface

Server Application: Implementing the Service Interface

```
//MathService Server or Provider

import java.awt.font.NumericShaper;
import java.math.BigInteger;
import java.rmi.*;
import java.rmi.registry.LocateRegistry;
import java.rmi.server.UnicastRemoteObject;

public class MathServiceProvider extends UnicastRemoteObject implements
    MathService {

  // MathServiceProvider implements all the methods of MathService interface
  // service constructor
  public MathServiceProvider() throws RemoteException {
    super();
  }

  // implementation of factorial
  public long factorial(int number) {
    // returning factorial
    if (number == 1)
      return 1;
    return number * factorial(number - 1);
  }
```

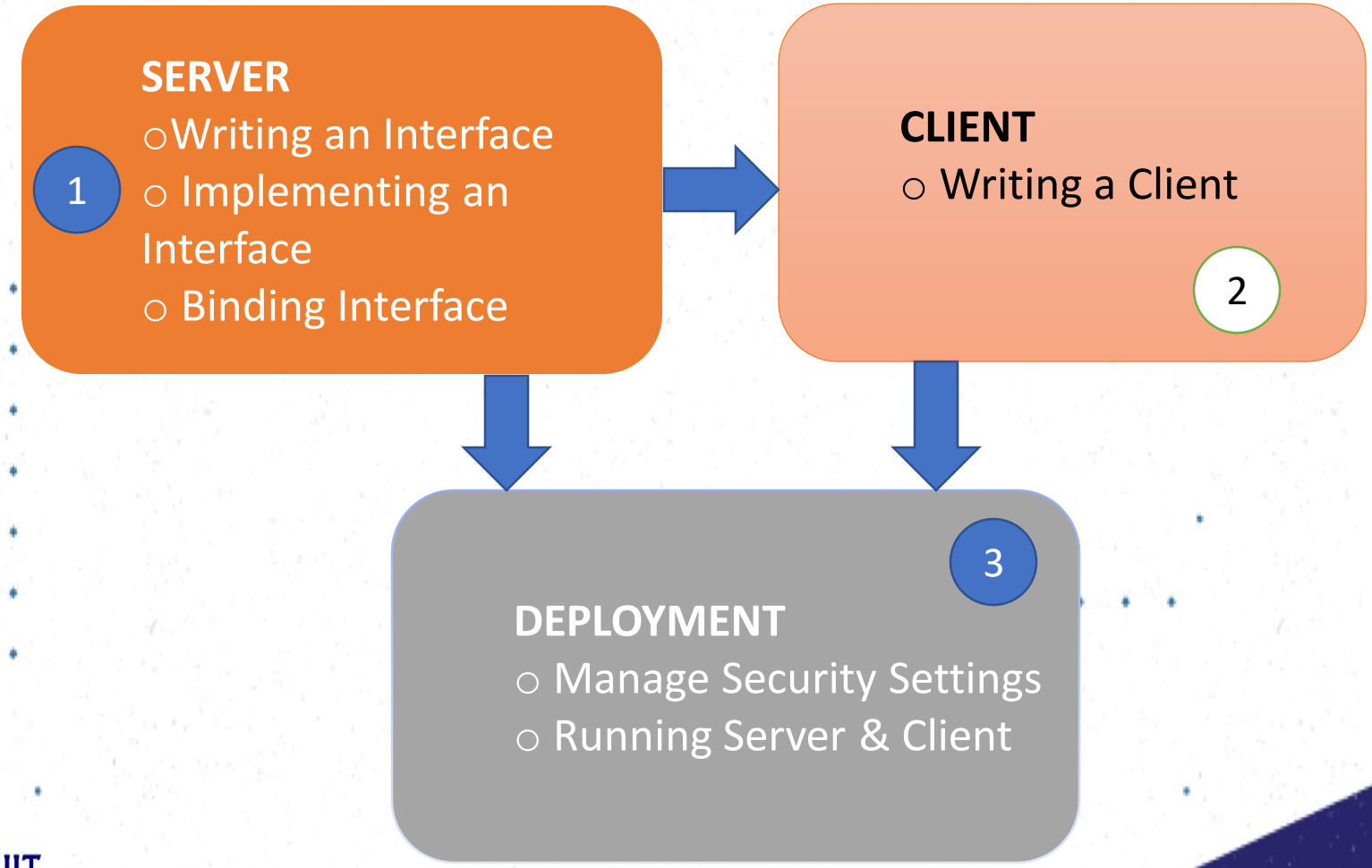
Server Application: Instantiating & Binding the Service

```
public static void main(String args[]) {
    try {
      // setting RMI security manager
      if (System.getSecurityManager() == null) {
        System.setSecurityManager(new
            RMISecurityManager());
      }

      // creating server instance
      MathServiceProvider provider = new
          MathServiceProvider();
      // binding the service with the registry
      LocateRegistry.getRegistry().bind("
          MathService", provider);
      System.out.println("Service is bound to RMI
           registry");
    } catch (Exception exc) {
      // showing exception
      System.out.println("Cant bind the service:
          " + exc.getMessage());
      exc.printStackTrace();

    }
  }
```

A Simple RMI Application



Client Application: Service Lookup

```
 // Assign security manager
if (System.getSecurityManager() == null) {
  System.setSecurityManager(new
      RMISecurityManager());
}

// Accessing RMI registry for MathService
String hostName = "localhost"; //this can
    be any host
MathService service = (MathService) Naming.
    lookup("//" + hostName
    + "/MathService");
```

Fig: Client locating *MathService* service

Client Application: Accessing Service

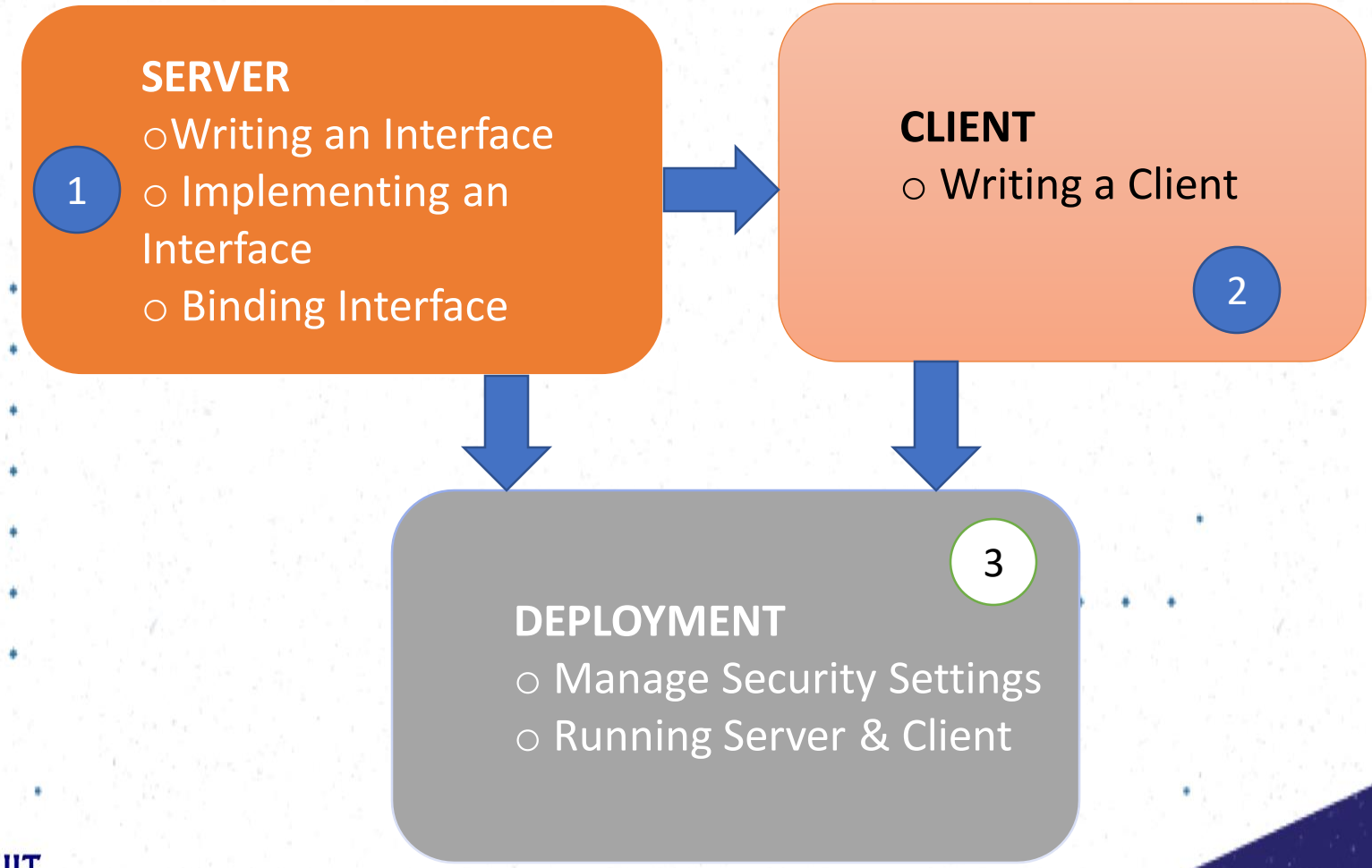
```
    // Call to factorial method
System.out.println("The factorial of " + number +
     "=" + service.factorial(number));

     // Call to checkPrime method
boolean isprime=service.checkPrime(number);

    //Call to square method
BigInteger squareObj=service.square(number);
```

Fig: Client accessing *MathService* service

A Simple RMI Application



Server Deployment: Start RMI Registry

- To start RMI registry on windows

```
$ start rmiregistry
```

- To start RMI registry on Unix

```
$ rmiregistry \&
```

Server Deployment: Compile the Server

- Compile both MathService interface and MathServiceProvider class

```
$ javac MathService.java MathServiceProvider.
    java
```

Security Deployment: Create Security Policy file (Both Client & Server)

- Create a security policy file called *no.policy* with the following content and add it to CLASSPATH
- This step implies for both server and client

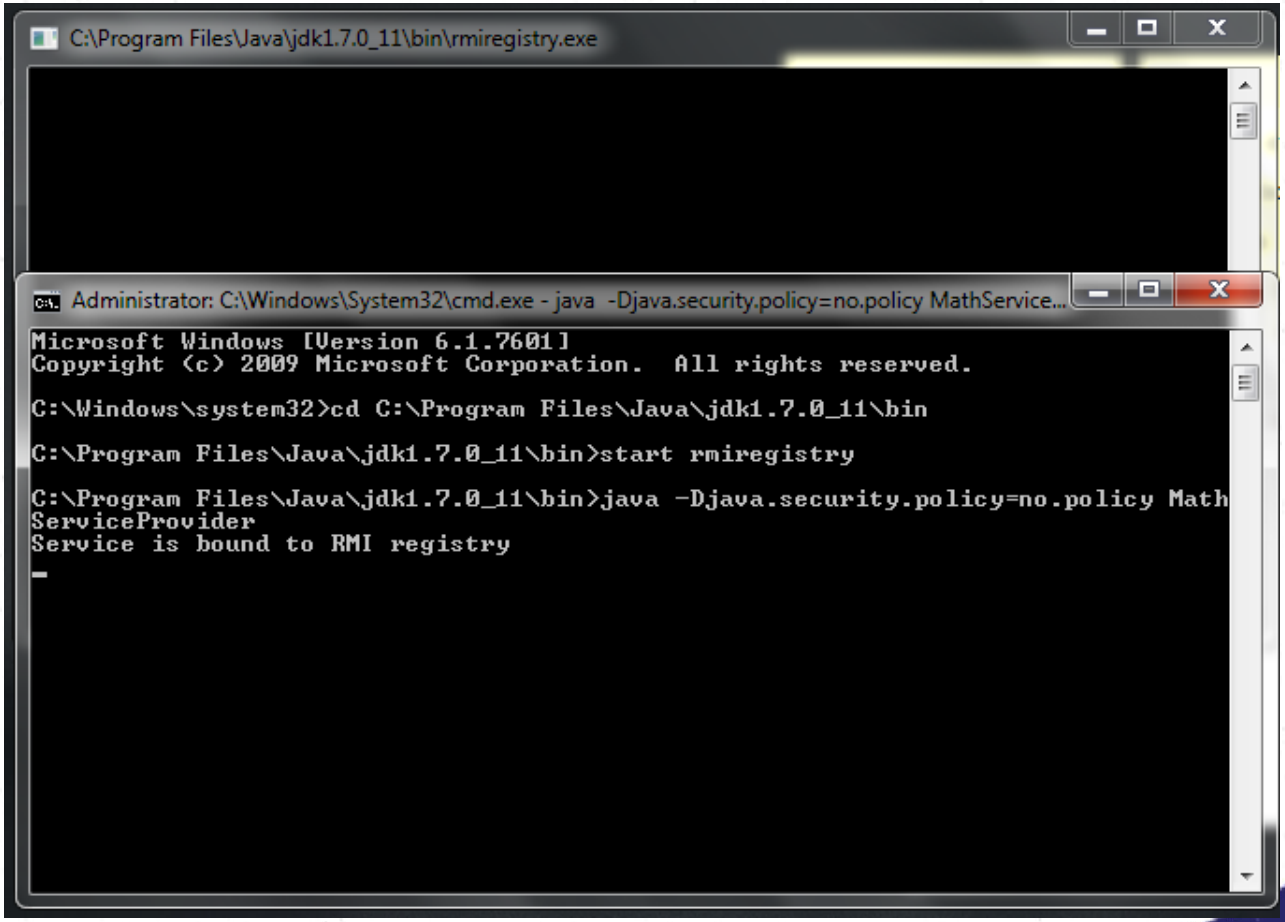
```
grant {
permission java.security.AllPermission;
};
```

Start the Server

- Execute the command to run server

```
$ java -Djava.security.policy=no.policy
    MathServiceProvider
```

Server Running

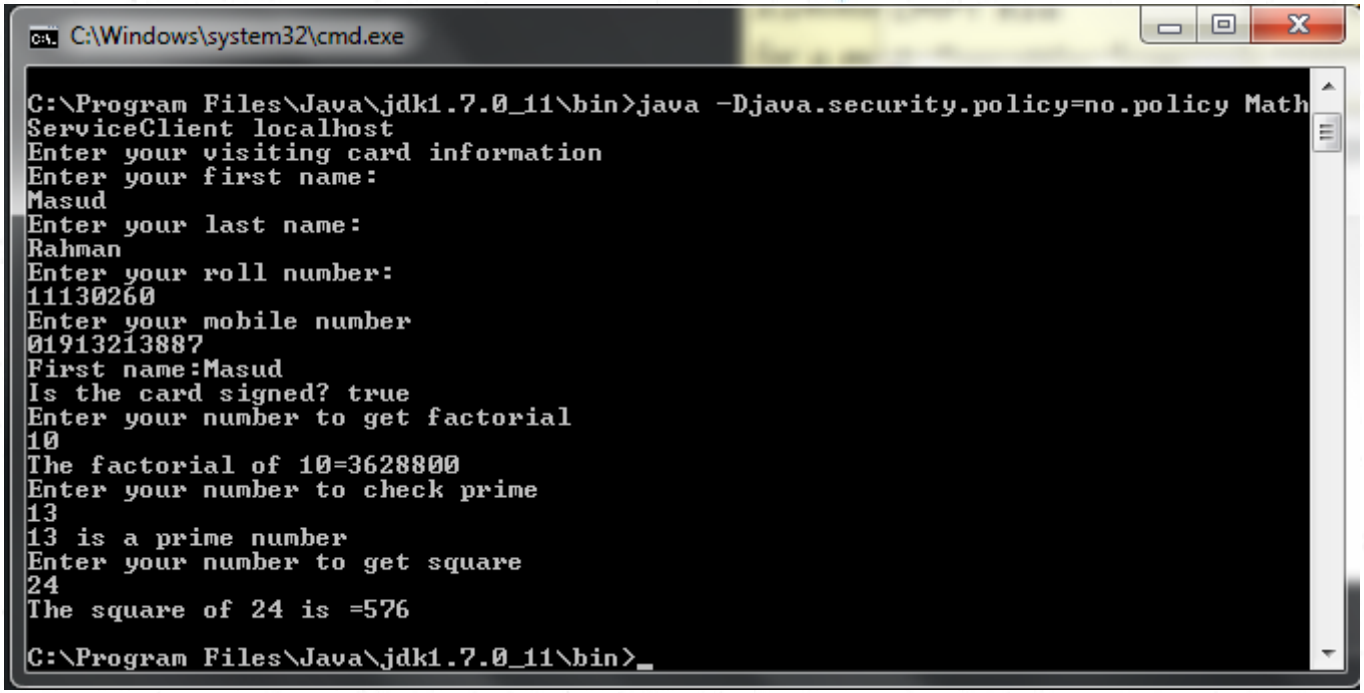


Start the Client

- Execute the command to run client

```
$ java -Djava.security.policy=no.policy
    MathServiceClient localhost
```

Client Interface



Java RMI notes

- Java Object Serialization
- Parameter Marshalling & Unmarshalling
- Object Activation
  - Singleton
  - Per client
  - Per call

Strength Of Java RMI

- *Object Oriented*: Can pass complex object rather than only primitive types
- *Mobile Behavior*: Change of roles between client and server easily
- *Design Patterns*: Encourages OO design patterns as objects are transferred
- *Safe & Secure*: The security settings of Java framework used
- *Easy to Write /Easy to Use*: Requires very little coding to access service

Strengths Of Java RMI

- *Connects to Legacy Systems*: JNI & JDBC facilitate access.
- *Write Once, Run Anywhere*: 100% portable, run on any machine having JVM
- *Distributed Garbage Collection*: Same principle like memory garbage collection
- *Parallel Computing*: Through multi-threading RMI server can serve numerous clients
- *Interoperable between different Java versions*: Available from JDK 1.1, can communicate between all versions of JDKs

Weaknesses of Java RMI

- *Tied to Java System*: Purely Java-centric technology, does not have good support for legacy system written in C, C++, Ada etc.
- *Performance Issue* : Only good for large-grain computation
- *Security Restrictions & Complexities*: Threats during downloading objects from server, malicious client request, added security complexity in policy file.

.NET Remoting

- .NET equivalent of Java RMI
- Uses Windows registry as the registry service
- Java RMI supports more communication protocols and .NET remoting
- .NET remoting creates proxies at runtime similar to RMI

Conclusion

- RMI/RPC makes it easier to build distributed systems with programmers not having to worry about network comm. (sockets, etc)
- .NET Remoting
- No interoperability (Java only due to binary messages used)