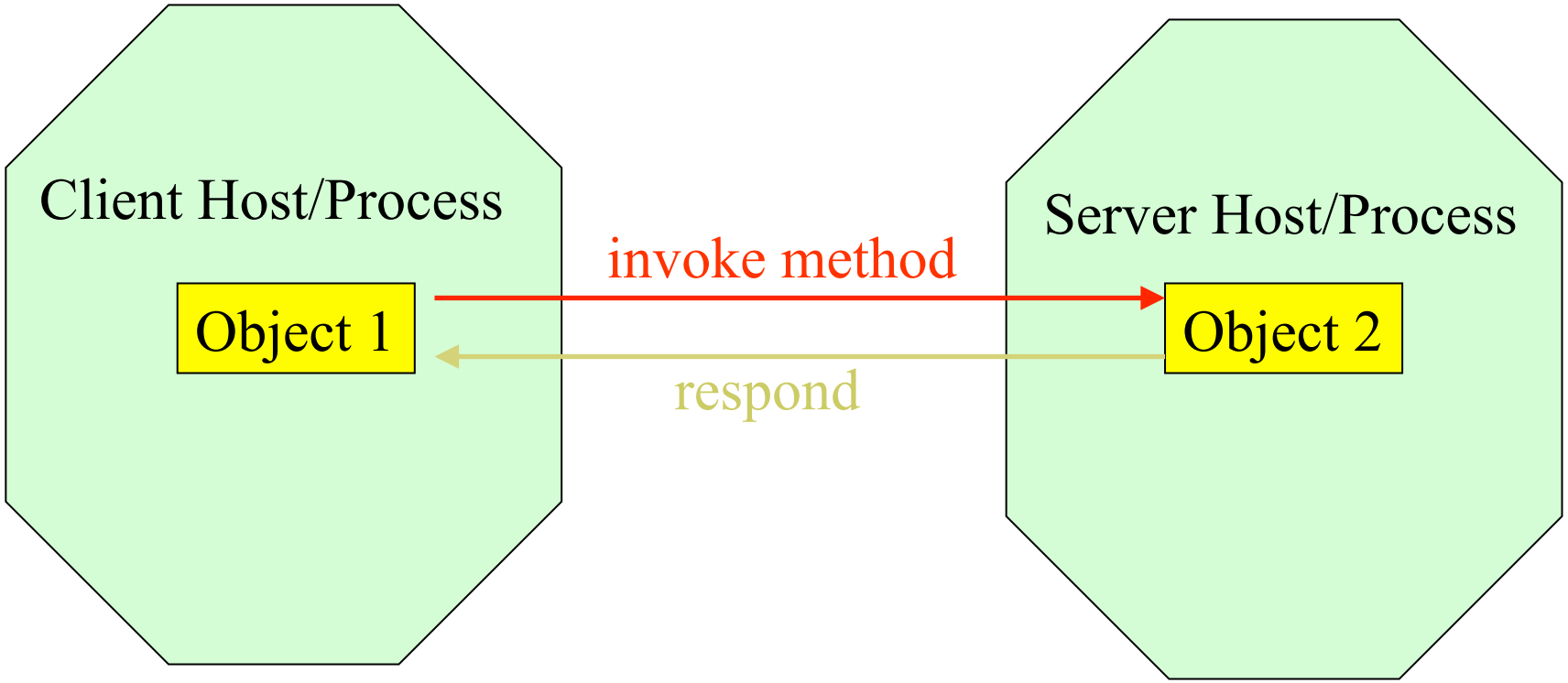
Distributed Objects

Remote Method Invokation

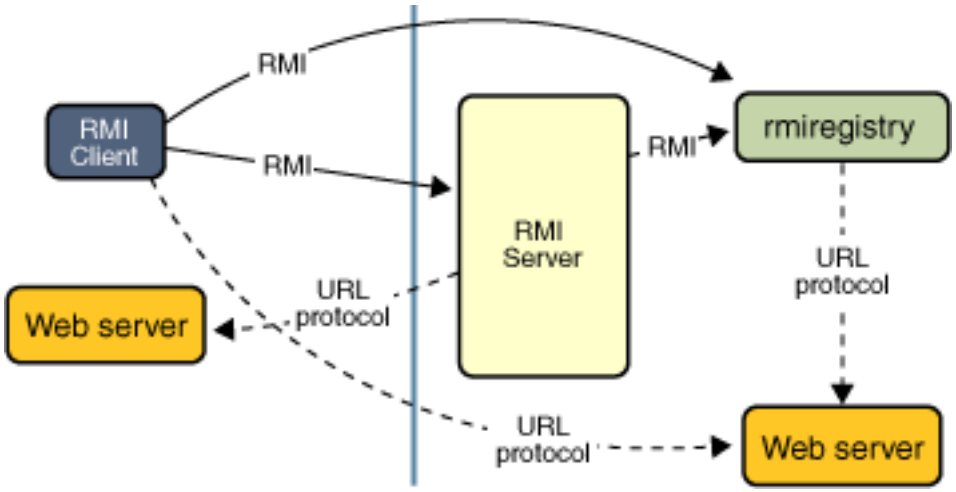
Distributed Systems

Object Oriented Paradigm

Distributed Object Oriented Paradigm



The RMI model



Distributed object applications need to:

- Locate remote objects. Applications can use various mechanisms to obtain references to remote objects. For example, an application can register its remote objects with RMI's simple naming facility, the RMI registry. Alternatively, an application can pass and return remote object references as part of other remote invocations.
- Communicate with remote objects. Details of communication between remote objects are handled by RMI. To the programmer, remote communication looks similar to regular Java method invocations.
- Load class definitions for objects that are passed around. Because RMI enables objects to be passed back and forth, it provides mechanisms for loading an object's class definitions as well as for transmitting an object's data.

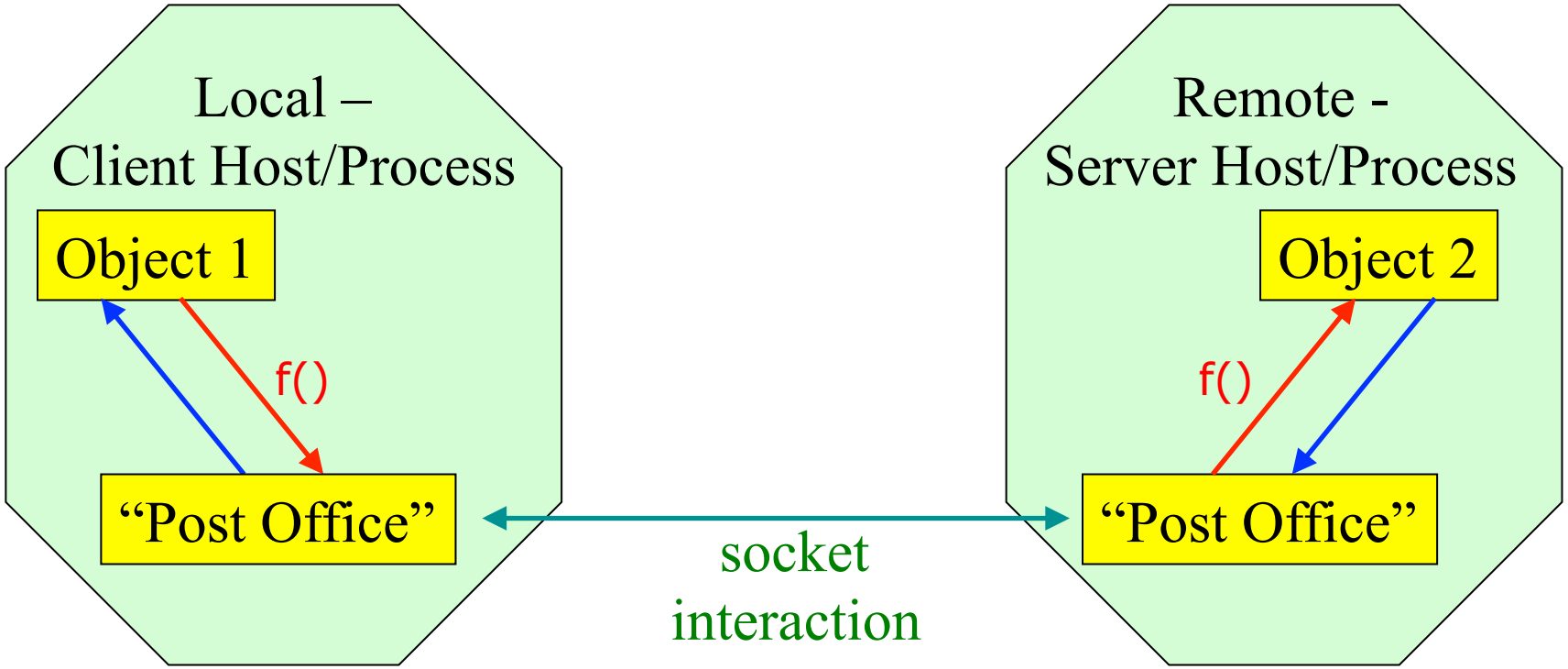
Distributed Objects

Remote Method Invokation:

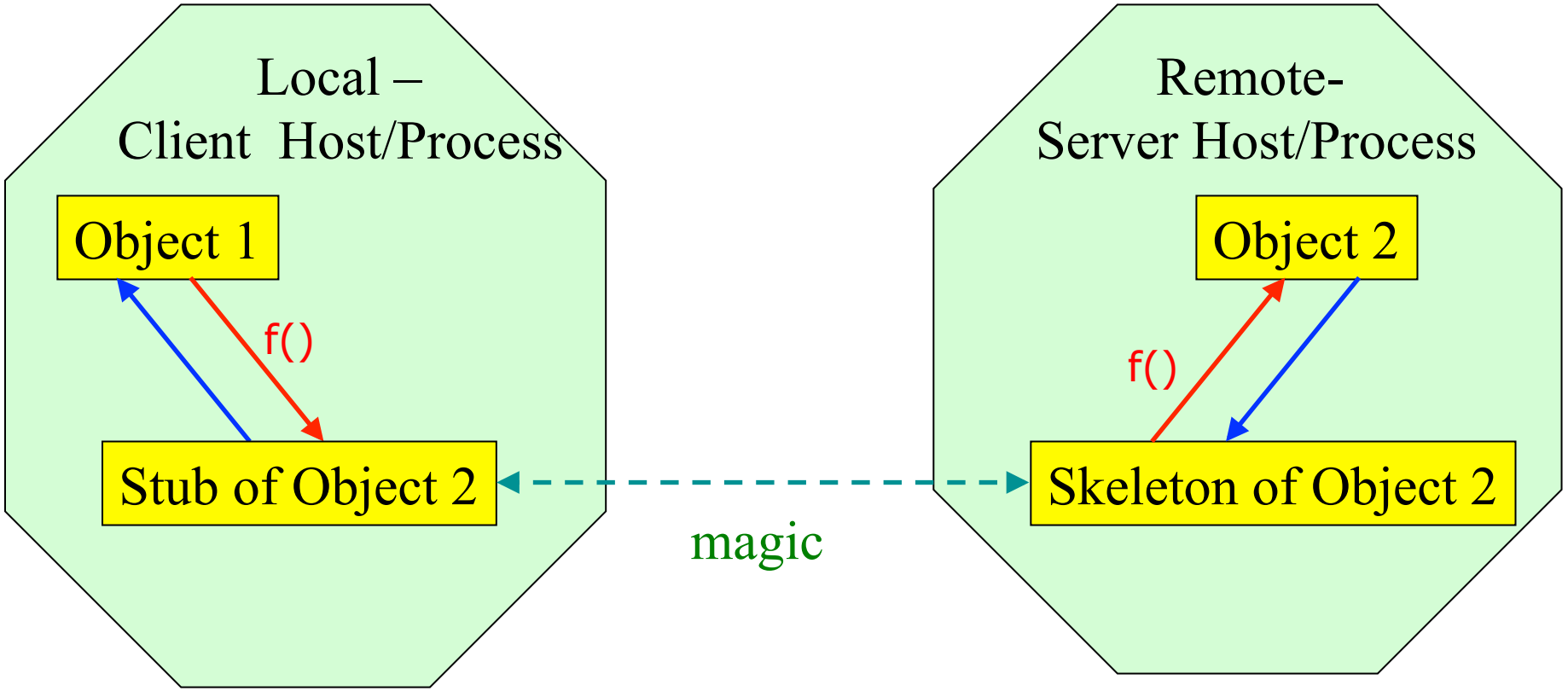
How does it work?

The conceptual model

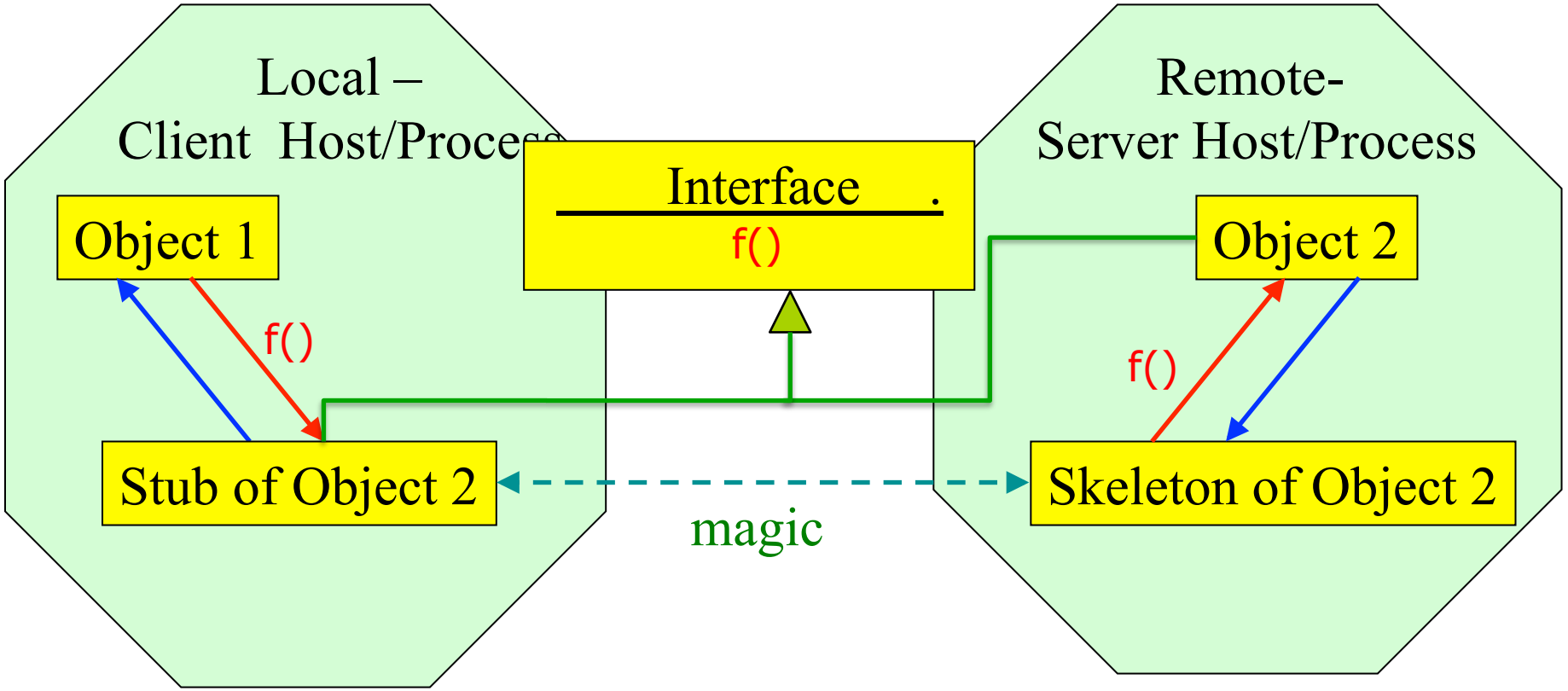
Distributed Object Oriented: implementation



Distributed Object Oriented: RMI paradigm



Distributed Object Oriented: RMI paradigm



Distributed Objects

A “do it yourself” implementation

A “do it yourself” implementation

1. Person: the interface

```
package distributedobjectdemo;

public interface Person {
  public int getAge() throws Throwable;
  public String getName() throws Throwable;
}
```

A “do it yourself” implementation

2. Person: The class

```
package distributedobjectdemo;

public class PersonServer implements Person{
  int age;
  String name;
  public PersonServer(String name,int age){
    this.age=age;
    this.name=name;
  }
  public int getAge(){
    return age;
  }
  public String getName(){
    return name;
  }
 public static void main(String a[]) {
    PersonServer person = new PersonServer("Marko", 45);
    Person_Skeleton skel = new Person_Skeleton(person);
    skel.start();
    System.out.println("server started");
  }
}
```

A “do it yourself” implementation

3. Person: the skeleton

```
package distributedobjectdemo;
import java.net.Socket;
import java.net.ServerSocket;
import java.io.*;

public class Person_Skeleton extends Thread {
  PersonServer myServer;
  int port=9000;

  public Person_Skeleton(PersonServer server) {
    this.myServer=server;
  }
// la classe continua...
```

A “do it yourself” implementation

3. Person: the skeleton

```
public void run(){
   Socket socket = null;
   ServerSocket serverSocket=null;
   try {
     serverSocket=new ServerSocket(port);
   }
   catch (IOException ex) {
    System.err.println("error while creating serverSocket");
    ex.printStackTrace(System.err);  System.exit(1);
   }

   while (true) {
    try {
     socket=serverSocket.accept();
     System.out.println("Client opened connection");
    }
    catch (IOException ex) {
     System.err.println("error accepting on serverSocket");
     ex.printStackTrace(System.err);  System.exit(1);
    }
    // il metodo continua...
```

A “do it yourself” implementation

3. Person: the skeleton

```
try {
  while (socket!=null){
    ObjectInputStream instream=
              new ObjectInputStream(socket.getInputStream());
    String method=(String)instream.readObject();
    if (method.equals("age")) {
      int age=myServer.getAge();
      ObjectOutputStream outstream=
              new ObjectOutputStream(socket.getOutputStream());
      outstream.writeInt(age);
      outstream.flush();
    } else if (method.equals("name")) {
      String name=myServer.getName();
      ObjectOutputStream outstream=
              new ObjectOutputStream(socket.getOutputStream());
      outstream.writeObject(name);
      outstream.flush();
    }
  }
//prosegue con il catch...
```

A “do it yourself” implementation

3. Person: the skeleton

```
    } catch (IOException ex) {
       if (ex.getMessage().equals("Connection reset")) {
         System.out.println("Client closed connection");
       } else {
         System.err.println("error on the network");
         ex.printStackTrace(System.err);   System.exit(2);
       }
     } catch (ClassNotFoundException ex) {
       System.err.println("error while reading object from the net");
       ex.printStackTrace(System.err);     System.exit(3);
     }
    }//fine del ciclo while(true)
  }  //fine del metodo run
}    //fine della classe
```

A “do it yourself” implementation

4. Person: the stub

```
package distributedobjectdemo;
import java.net.Socket;
import java.io.*;

public class Person_Stub implements Person {
  Socket socket;
  String machine="localhost";
  int port=9000;

  public Person_Stub() throws Throwable {
    socket=new Socket(machine,port);
  }
  protected void finalize(){
    System.err.println("closing");
    try { socket.close(); }
    catch (IOException ex) {ex.printStackTrace(System.err); }
  }
  // la classe continua...
```

A “do it yourself” implementation

4. Person: the stub

```
public int getAge() throws Throwable {
   ObjectOutputStream outstream=
         new ObjectOutputStream(socket.getOutputStream());
   outstream.writeObject("age");
   outstream.flush();
   ObjectInputStream instream=
         new ObjectInputStream(socket.getInputStream());
   return instream.readInt();
  }

  public String getName() throws Throwable {
   ObjectOutputStream outstream=new
ObjectOutputStream(socket.getOutputStream());
   outstream.writeObject("name");
   outstream.flush();
   ObjectInputStream instream=
         new ObjectInputStream(socket.getInputStream());
   return (String)instream.readObject();
  }
} // fine della classe
```

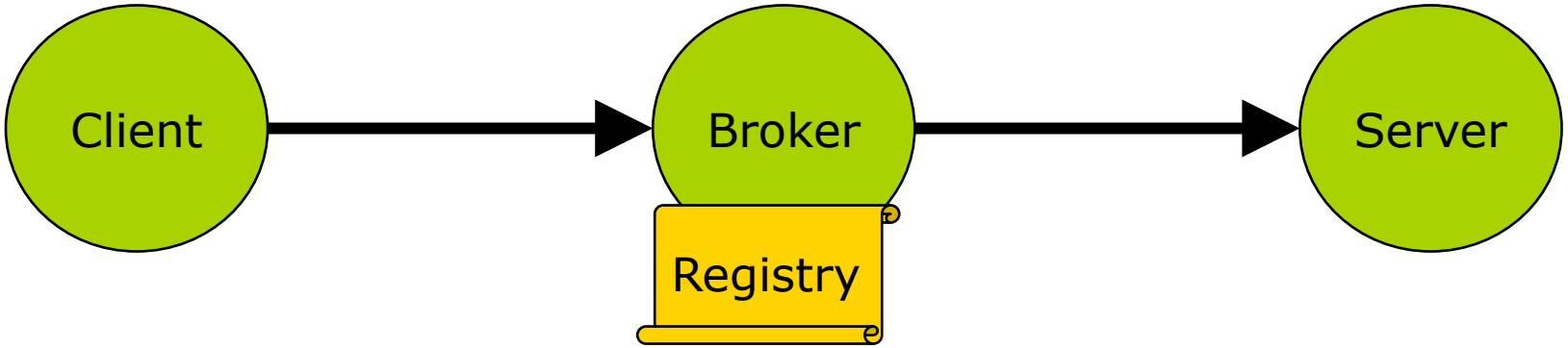
A “do it yourself” implementation

5. Person: the client

```
package distributedobjectdemo;

public class Client {

  public Client() {
    try {
      Person person=new Person_Stub();
      int age=person.getAge();
      String name=person.getName();
      System.out.println(name+" is "+age+" years old");
    }
    catch (Throwable ex) {
      ex.printStackTrace(System.err);
    }
  }
  public static void main(String[] args) {
    Client client1 = new Client();
  }
}
```

Open issues

- multiple instances
- Automatic stub and skeleton generation
- on demand server dentification
- on demand remote class activation



Distributed Objects

An RMI basic implementation

(example taken from
https://www.mkyong.com)

Remote Interface

1. Define the common interface

```
package it.unitn.rmiinterface;
import java.rmi.Remote;
import java.rmi.RemoteException;

public interface RMIInterface extends Remote {

    public String helloTo(String name) throws RemoteException;

}
```

The Server

2. Implement the service

```
package it.unitn.rmiserver;
import java.rmi.Naming;
import java.rmi.RemoteException;
import java.rmi.server.UnicastRemoteObject;
import it.unitn.RMIInterface;
```

```
public class ServerOperation extends UnicastRemoteObject
        implements RMIInterface{
    private static final long serialVersionUID = 1L;

    protected ServerOperation() throws RemoteException {
        super();
    }

    @Override
    public String helloTo(String name) throws RemoteException{
        System.err.println(name + " is trying to contact!");
        return "Server says hello to " + name;
    }
```

The Server

3. Create Registry

```
import java.rmi.registry.*;
LocateRegistry.createRegistry(1099);
```

The Server

```
    public static void main(String[] args){
        try {
            Naming.rebind("//localhost/MyServer", new
                    ServerOperation());
            System.err.println("Server ready");
        } catch (Exception e) {
            System.err.println("Server exception: " +
                    e.toString());
            e.printStackTrace();
        }
    }
}
```

4. Register yourself

The client

```
package it.unitn.rmiclient
import java.net.MalformedURLException
import java.rmi.Naming
import java.rmi.NotBoundException
import java.rmi.RemoteException
import javax.swing.JOptionPane
import it.unitn.RMIInterface
```

```
public class ClientOperation {
        private static RMIInterface remoteObj;
        public static void main(String[] args)
                throws MalformedURLException, RemoteException,
                        NotBoundException {
        remoteObj = (RMIInterface) Naming.lookup("//
                        localhost/MyServer");
        String txt = JOptionPane.showInputDialog("What is
                        your name?");
        String response = remoteObj.helloTo(txt);
        JOptionPane.showMessageDialog(null, response);
        }
}
```

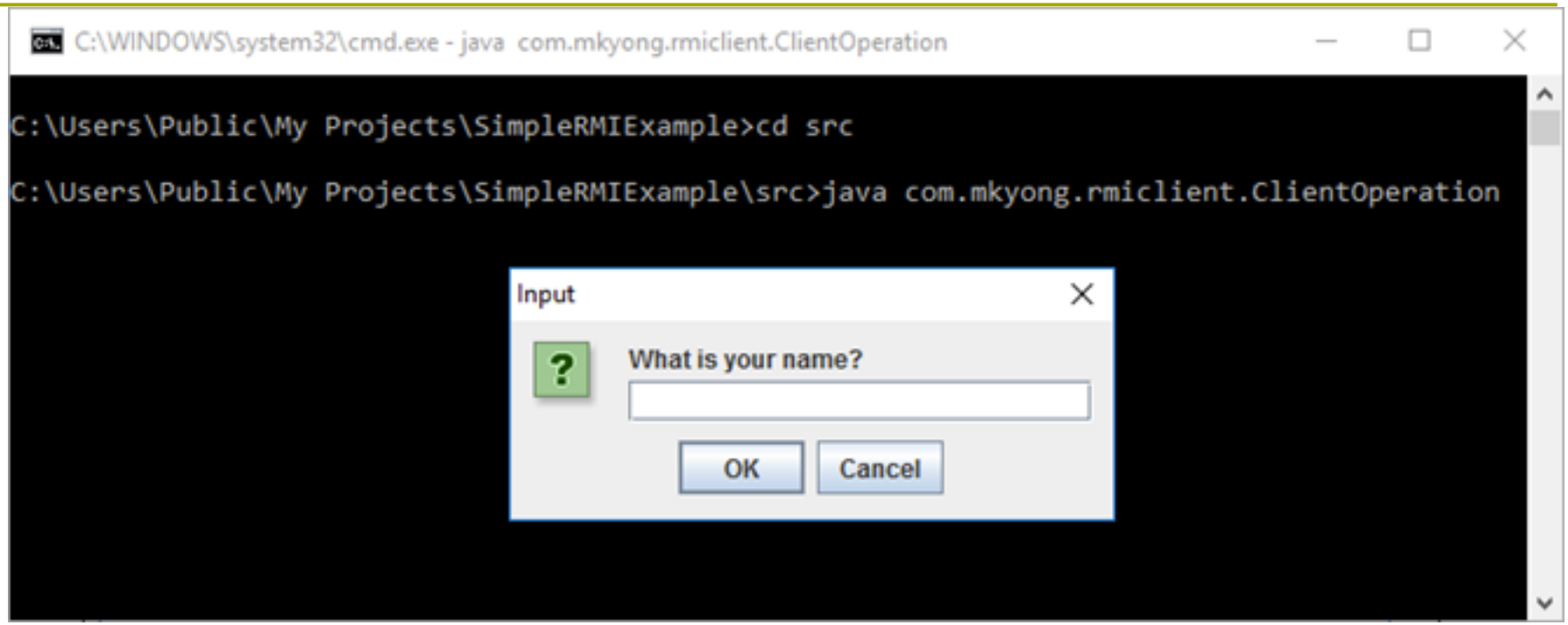
5. Lookup The Service

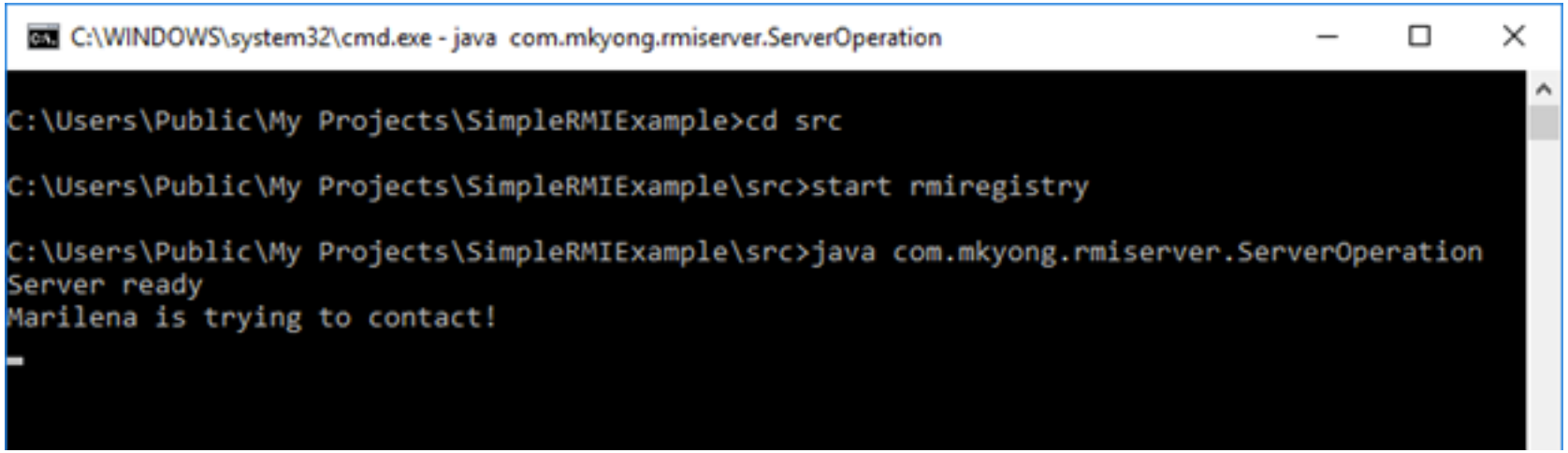
6. Use Service

Deploy

```
COMPILE:
javac src/it/unitn/rmiinterface/RMIInterface.java src/it/unitn/
rmiserver/ServerOperation.java src/it/unitn/rmiclient/
ClientOperation.java

START REGISTRY
cd src
start rmiregistry

START SERVER
cd src
java it.unitn.rmiserver.ServerOperation

START CLIENT
cd src
java it.unitn.rmiclient.ClientOperation
```

Running


C:\WINDOWS\system32\cmd.exe - java com.mkyong.rmiclient.ClientOperation
C:\Users\Public\My Projects\SimpleRMIExample>cd src
C:\Users\Public\My Projects\SimpleRMIExample\src>java com.mkyong.rmiclient.ClientOperation
Input
What is your name?
OK
Cancel


C:\WINDOWS\system32\cmd.exe - java com.mkyong.rmiserver.ServerOperation
C:\Users\Public\My Projects\SimpleRMIExample>cd src
C:\Users\Public\My Projects\SimpleRMIExample\src>start rmiregistry
C:\Users\Public\My Projects\SimpleRMIExample\src>java com.mkyong.rmiserver.ServerOperation
Server ready
Marilena is trying to contact!