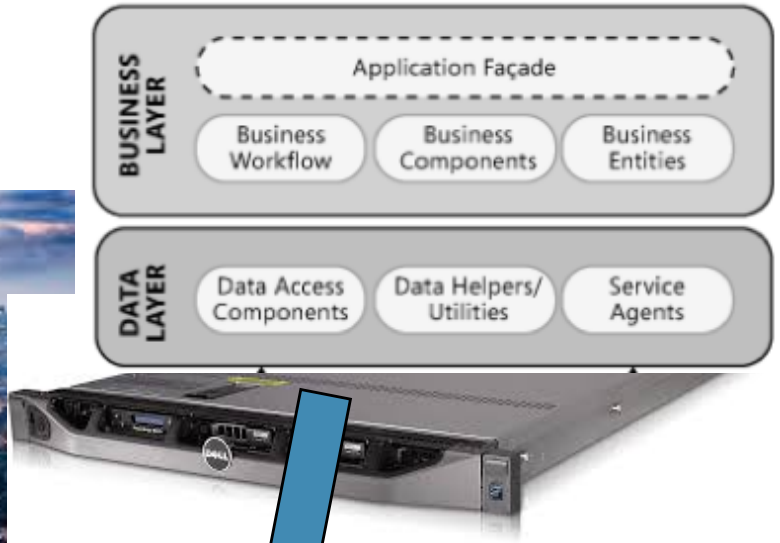
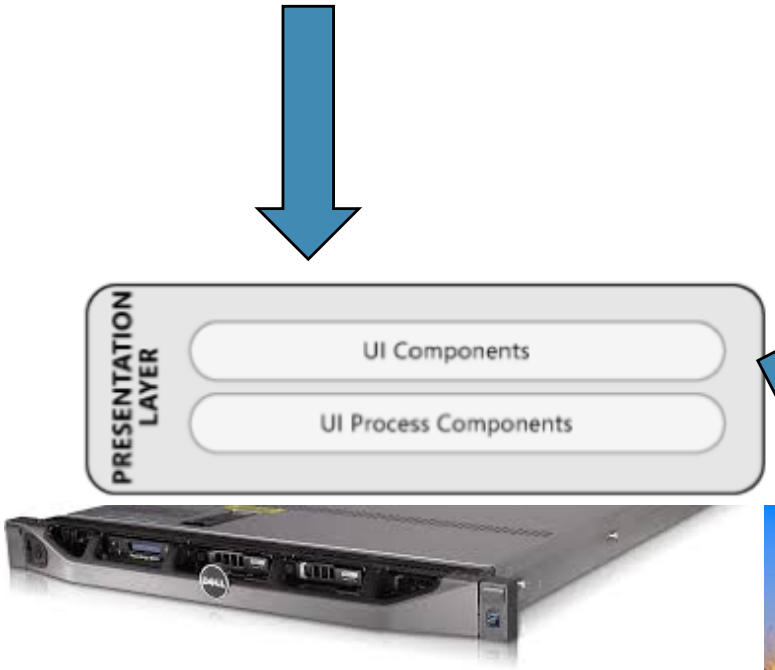
Distributed Objects

Remote Method Invokation

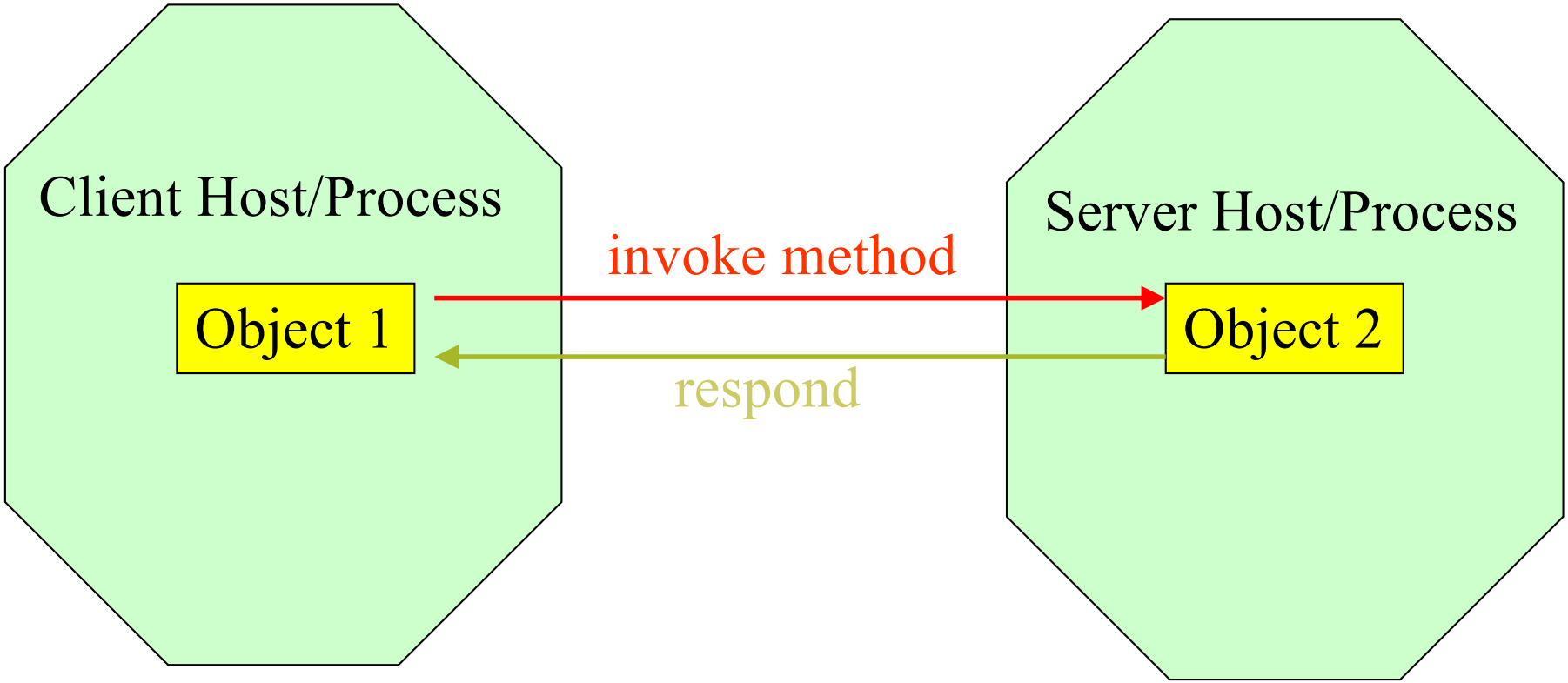
Distributed Systems





Object Oriented Paradigm

Distributed Object Oriented Paradigm



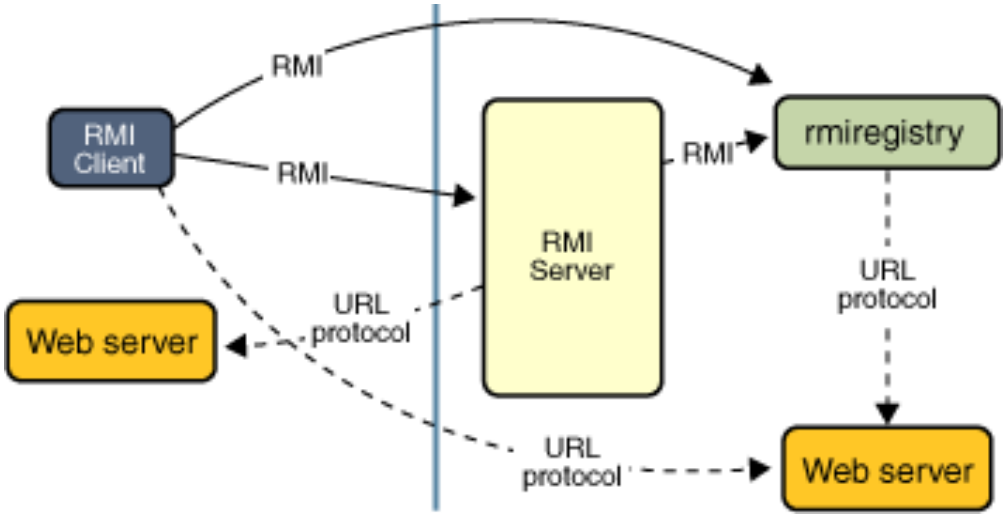
Distributed object applications need to:

- 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.

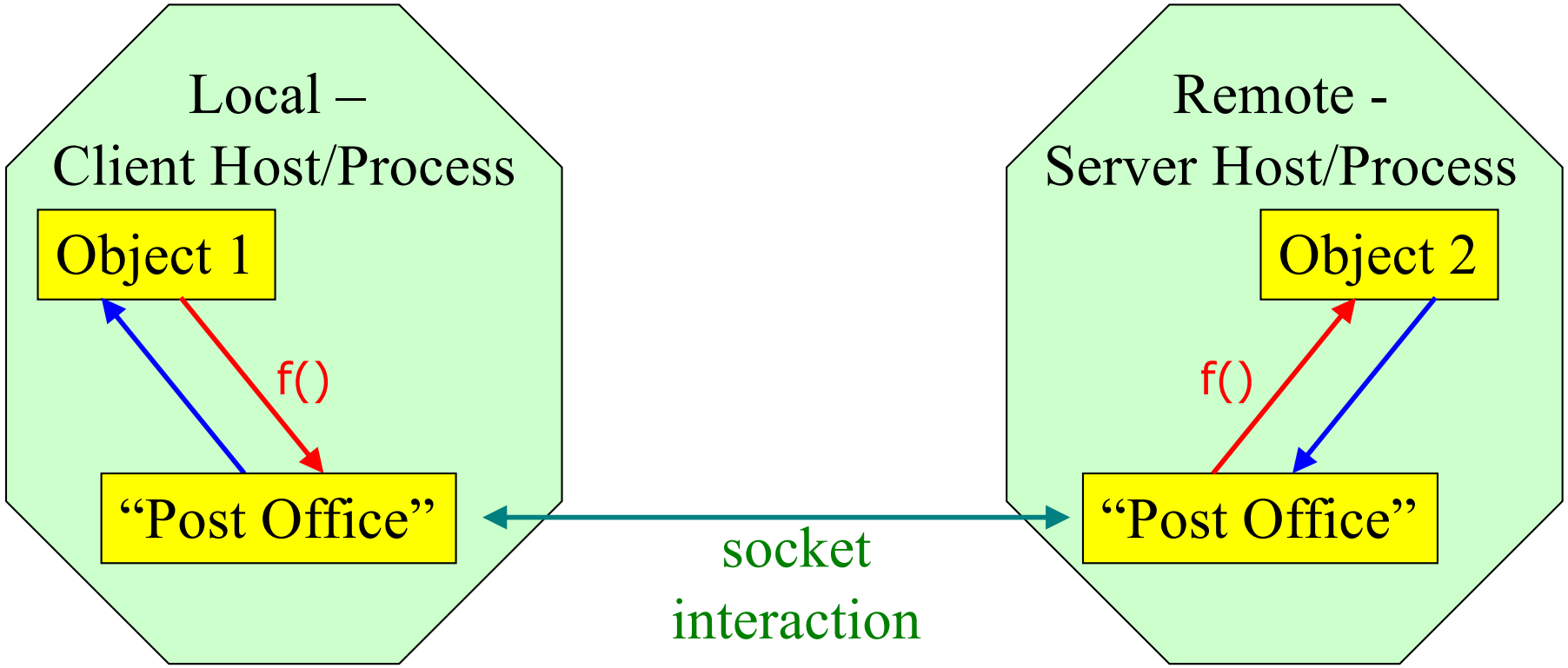
The RMI model



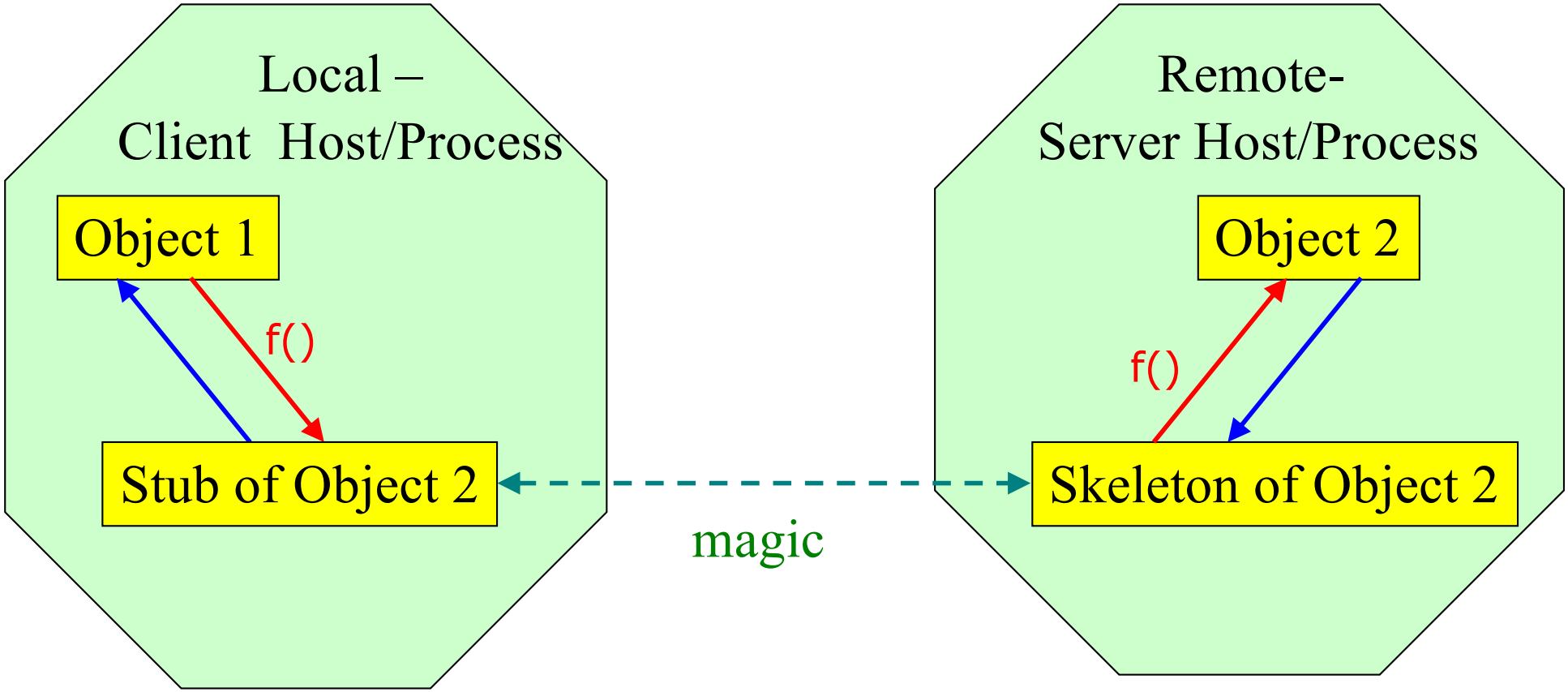
Q

How does RMI work?
What is its conceptual model?

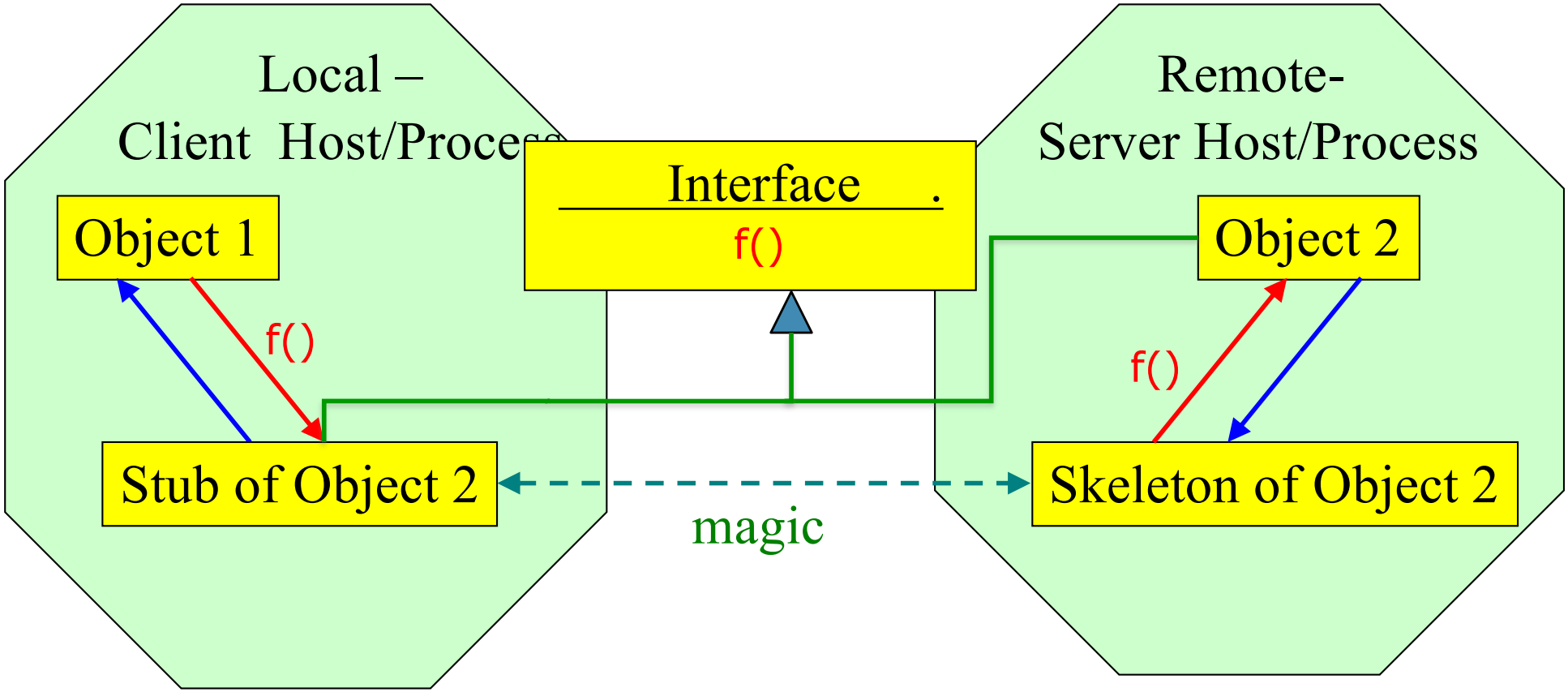
Distributed Object Oriented: implementation



Distributed Object Oriented: RMI paradigm



Distributed Object Oriented: RMI paradigm



Q

How could that be implemented?

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;
  }
// the class continues...
```

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);
     }
     // the method continues...
```

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();
    }
  }
//continue with the 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);
    }
   }//end of while(true)
  }  //end of run method
}   //end of class
```

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); }
  }
  // the class continues...
```

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();
 }
} // end of class
```

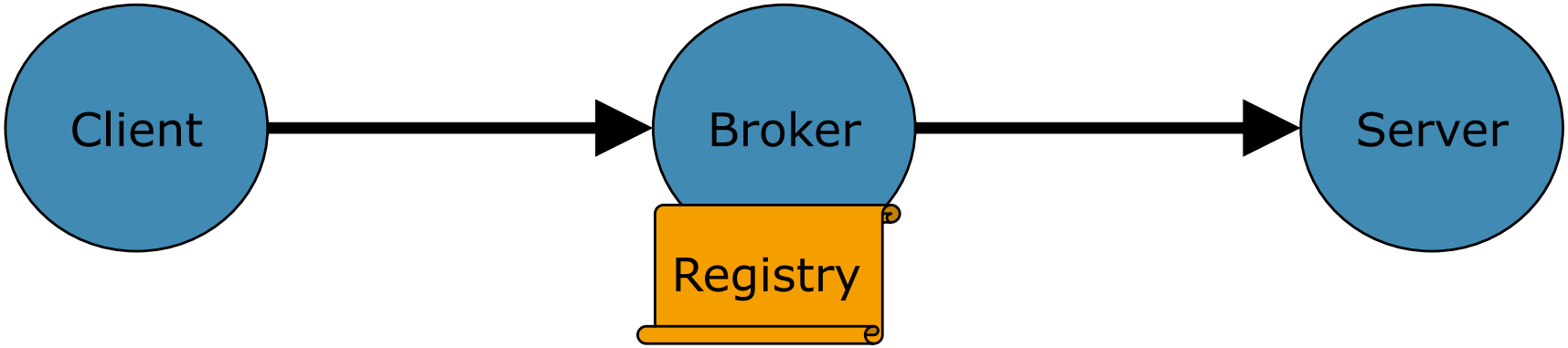
A “do it yourself” implementation

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



Q

How do I actually use RMI?

(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

4. Register yourself

```
    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();
        }
    }
}
```

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

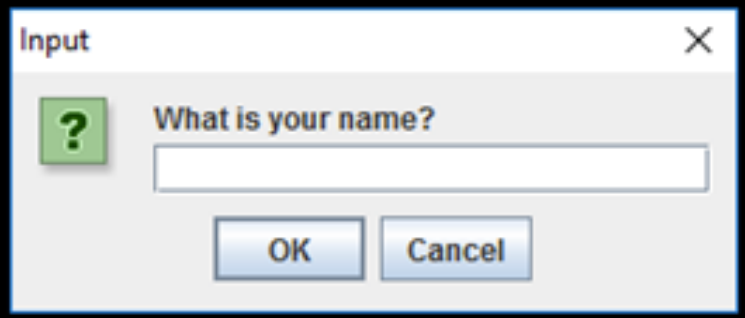
Deploy

```
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
```



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!
```

Q

What do you have to change
in the process and in the code
if you run client and server
on different machines?

Q

Can you access instance variables on the remote object?

Q

How do you pass parameters in RMI?

VERY IMPORTANT: Parameter passing

Java Standard:

```
void f(int x) :
Parameter x is passed by copy
void g(Object k) :
```

Parameter k and return value are passed by reference

Java RMI:

```
void h(Object k) :
```

Parameter k is passed by copy!

UNLESS k is a REMOTE OBJECT (in which case it is passed as a REMOTE REFERENCE, i.e. its stub is copied if needed)

IMPORTANT: Parameter passing

Passing By-Value

When invoking a method using RMI,all parameters to the remote method are passed *by-value* .This means that when a client calls a server,all parameters are copied from one machine to the other. To pass objects by value, they need to be serialized.

Passing by remote-reference

If you want to pass an object over the network by-reference,it must be a remote object, and it must implement java.rmi.Remote.A stub for the remote object is serialized and passed to the remote host. The remote host can then use that stub to invoke callbacks on your remote object. There is only one copy of the object at any time,which means that all hosts are calling the same object.

Q

What is serialization?

Serialization

- Any basic primitive type (int,char,and so on) is automatically serialized with the object and is available when deserialized.

- Java objects can be included with the serialized or not:

- Objects marked with the *transient* keyword are not serialized with the object and are not available when deserialized.

- Any object that is not marked with the transient keyword must implement *java.lang.Serializable* .These objects are converted to bit-blob format along with the original object. If your Java objects are neither transient nor implement *java.lang.Serializable,* a NotSerializable Exception is thrown when *writeObject()*is called.

Serialization

- All serializable classes must declare a

private static final field named serialVersionUID

to guarantee serialization compatibility between versions.

If no previous version of the class has been released, then the value of this field can be any long value, as long as the value is used consistently in future versions.

```
private static final long serialVersionUID = 227L;
```

When not to Serialize

- The object is large. Large objects may not be suitable for serialization because operations you do with the serialized blob may be very intensive. (one could save the blob to disk or transporting the blob across the network)

- The object represents a resource that cannot be reconstructed on the target machine.Some examples of such resources are database connections and sockets.

- The object represents sensitive information that you do not want to pass in a serialized stream..

Q

What is CORBA?
What is RMI-IIOP?

Common Object Request Broker Architecture

- The ORB is the basic mechanism by which objects transparently make requests to - and receive responses from - each other on the same machine or across a network.

A client need not be aware of the mechanisms used to communicate with or activate an object, how the object is implemented, or where the object is located.

see https://www.omg.org/spec/CORBA/3.4/Beta1

Interface Definition Language

- IDL is a descriptive language used to define data types and interfaces in a way that is independent of the programming language or operating system/processor platform. The IDL specifies only the syntax used to define the data types and interfaces.

It is normally used in connection with other specifications that further define how these types/interfaces are utilized in specific contexts and platforms.

see https://www.omg.org/spec/IDL/About-IDL/

RMI-IIOP

- RMI-IIOP is a special version of RMI that is compliant with CORBA.

RMI has some interesting features not available in RMI-IIOP,such as distributed garbage collection, object activation and downloadable class files.

EJB and J2EE mandate that you use RMI-IIOP, not RMI.

rmic –iiop generates IIOP stub and tie (instead of stub and skeleton)
rmic –idl generates OMG IDL

See docs.oracle.com/javase/7/docs/technotes/tools/#rmi

Preparing and executing

NOTES:

the skeleton does not exist any more as separate file(its functionality is absorbed by the class file).

the rmic functionality has been absorbed by javac, so the whole process becomes transparent (but even more misterious…)

See docs.oracle.com/javase/tutorial/rmi/ for an example of current usage of rmi

Preparing and executing - security

The JDK security model requires code to be granted specific permissions to be allowed to perform certain operations.

You need to specify a policy file when you run your server and client.

```
grant { permission java.net.SocketPermission "*:1024-65535",
"connect,accept";
permission java.io.FilePermission "c:\\…path…\\", "read"; };
```

```
java -Djava.security.policy=java.policy executableClass
```

Access to system properties

- **Nota: instead of specifìying a property at runtime (-D switch of java command), You can hardwire the property into the code:**

```
-Djava.security.policy=java.policy

System.getProperties().put(
            "java.security.policy",
            "java.policy");
```