# Multi-agent systems

**Andrej Lúčny**

**KAI FMFI UK**

**lucny@fmph.uniba.sk**

**http://www.agentspace.org/mas**

# Implementation of MAS

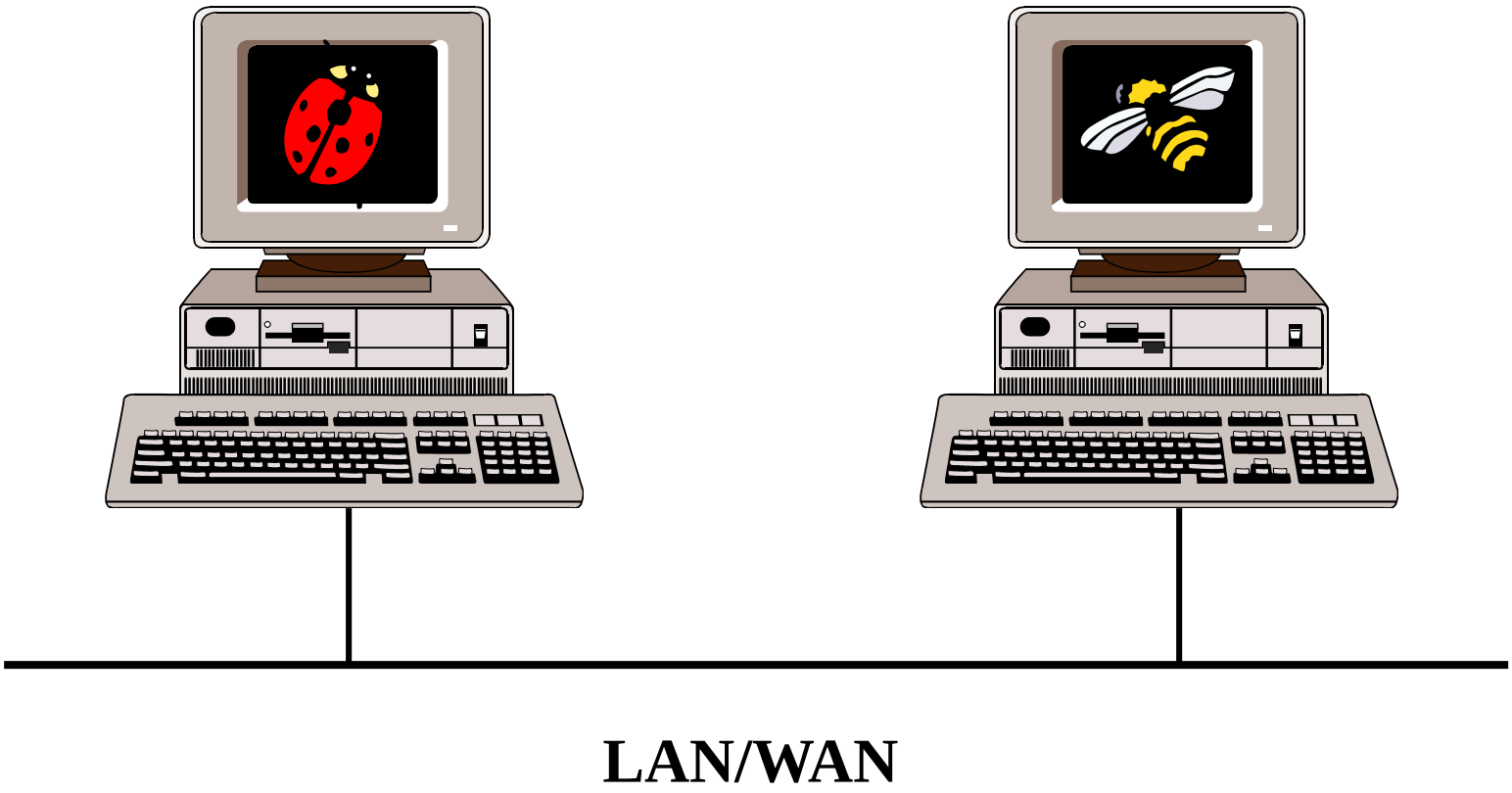
Usually we employ and existing sw. framework

- Middleware (distributed system)
- VM with threads (multi-thread system)
- IPC (multi-process system)

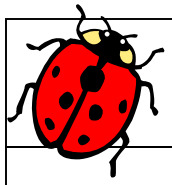
| | | |
|---|---|---|
| CORBA, RMI<br>SOAP<br>TCP/IP (TCP, HTTP) | JVM | SRR |
| LAN-WAN | VM | IPC |
| Network programming | Object oriented programming | concurent programming |

| | | |
|---|---|---|
| CORBA, RMI<br>SOAP<br>TCP/IP (TCP, HTTP) | JVM | SRR |
| LAN-WAN | VM | IPC |
| Network programming | Object oriented programming | concurent programming |

# MAS as middleware



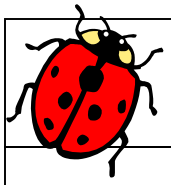
*layers*



# OSI model

**LAN/WAN**

*layers*



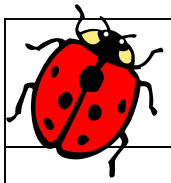
physical

transmission to communication media, frames

# OSI model

**LAN/WAN**

*layers*

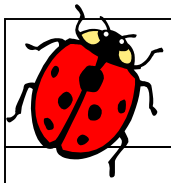


| | |
|---|---|
| | |
| | |
| | |
| | |
| datalink | Physical addresses MAC and conflict resolution |
| physical | transmission to communication media, frames |

# OSI model

**LAN/WAN**

*layers*

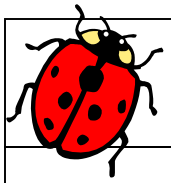


| | |
|---|---|
| | |
| | |
| | |
| network | Logic address, routing, packets |
| datalink | Physical addresses MAC and conflict resolution |
| physical | transmission to communication media, frames |

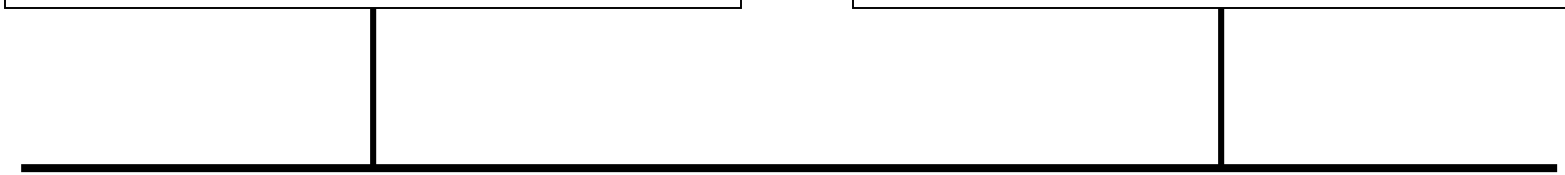
# OSI model

**LAN/WAN**

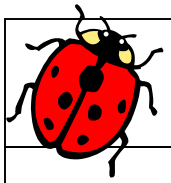
*layers*



| | |
|---|---|
| | |
| | |
| transport | Translation of data to packets and back (streams, datagrams) |
| network | Logic address, routing, packets |
| datalink | Physical addresses MAC and conflict resolution |
| physical | transmission to communication media, frames |



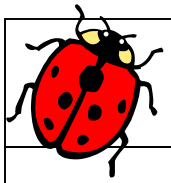
OSI model **LAN/WAN**

| *layers* | |
|---|---|
|  |  |
| | |
| session | Connection keeping, dialog control |
| transport | Translation of data to packets and back (streams, datagrams) |
| network | Logic address, routing, packets |
| datalink | Physical addresses MAC and conflict resolution |
| physical | transmission to communication media, frames |

# OSI model

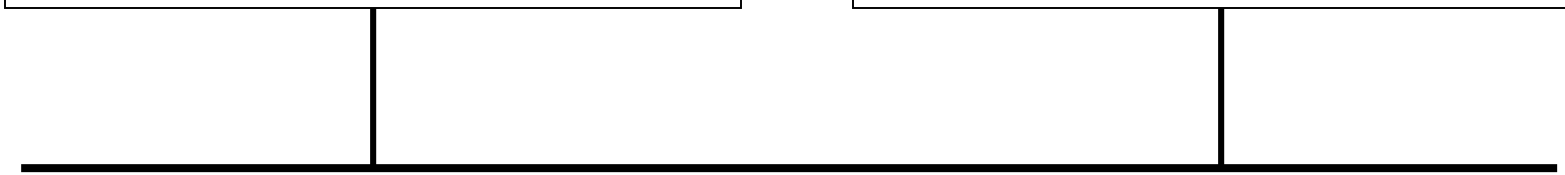
**LAN/WAN**

*layers*



| |
|---|
| presentation |
| session |
| transport |
| network |
| datalink |
| physical |

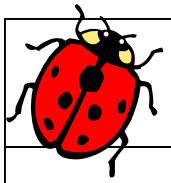
| |
|---|
| Normalization of communicated data |
| Connection keeping, dialog control |
| Translation of data to packets and back (streams, datagrams) |
| Logic address, routing, packets |
| Physical addresses MAC and conflict resolution |
| transmission to communication media, frames |



# OSI model

**LAN/WAN**

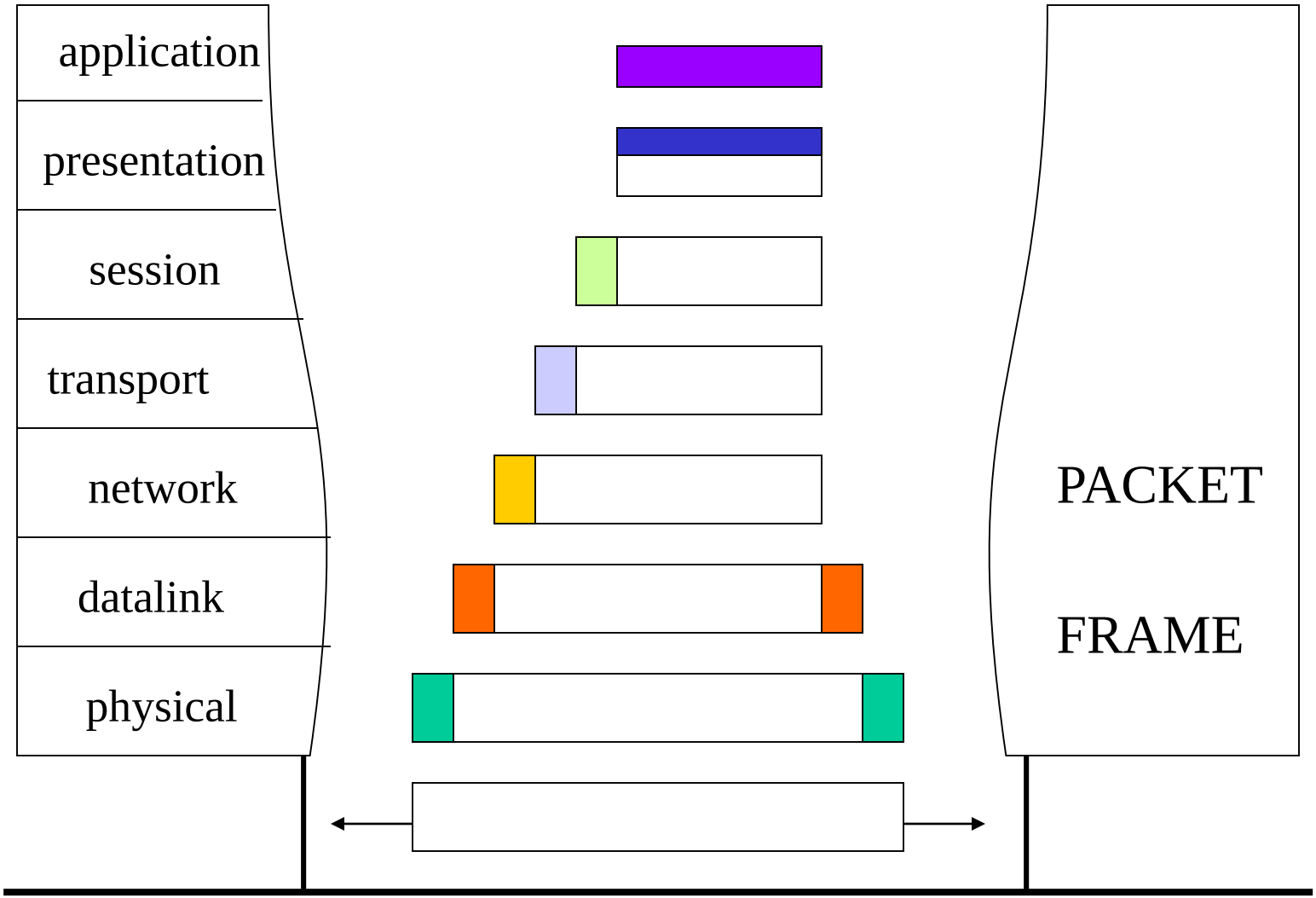
*layers*



| | |
|---|---|
| application |  Application logic |
| presentation | Normalization of communicated data |
| session | Connection keeping, dialog control |
| transport | Translation of data to packets and back (streams, datagrams) |
| network | Logic address, routing, packets |
| datalink | Physical addresses MAC and conflict resolution |
| physical | transmission to communication media, frames |

# OSI model

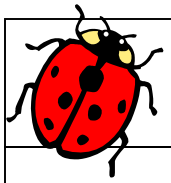
**LAN/WAN**




# OSI model

**LAN/WAN**

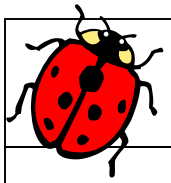
*layers*



| | |
|---|---|
| application | File Transfer FTP,<br>Simple Mail Transfer Protocol SMTP |
| presentation | |
| session | |
| transport | Transmission control protocol TCP<br>User datagram procotol UDP |
| network | Internet protocol IP |
| datalink | EtherNet |
| physical | |

**Internet** **LAN/WAN** **TCP/IP**

*layers*



application

presentation

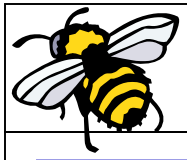
session

transport

network

datalink

physical



"hallo ladybug!"

14 h a l l o _ l a d y b u g !

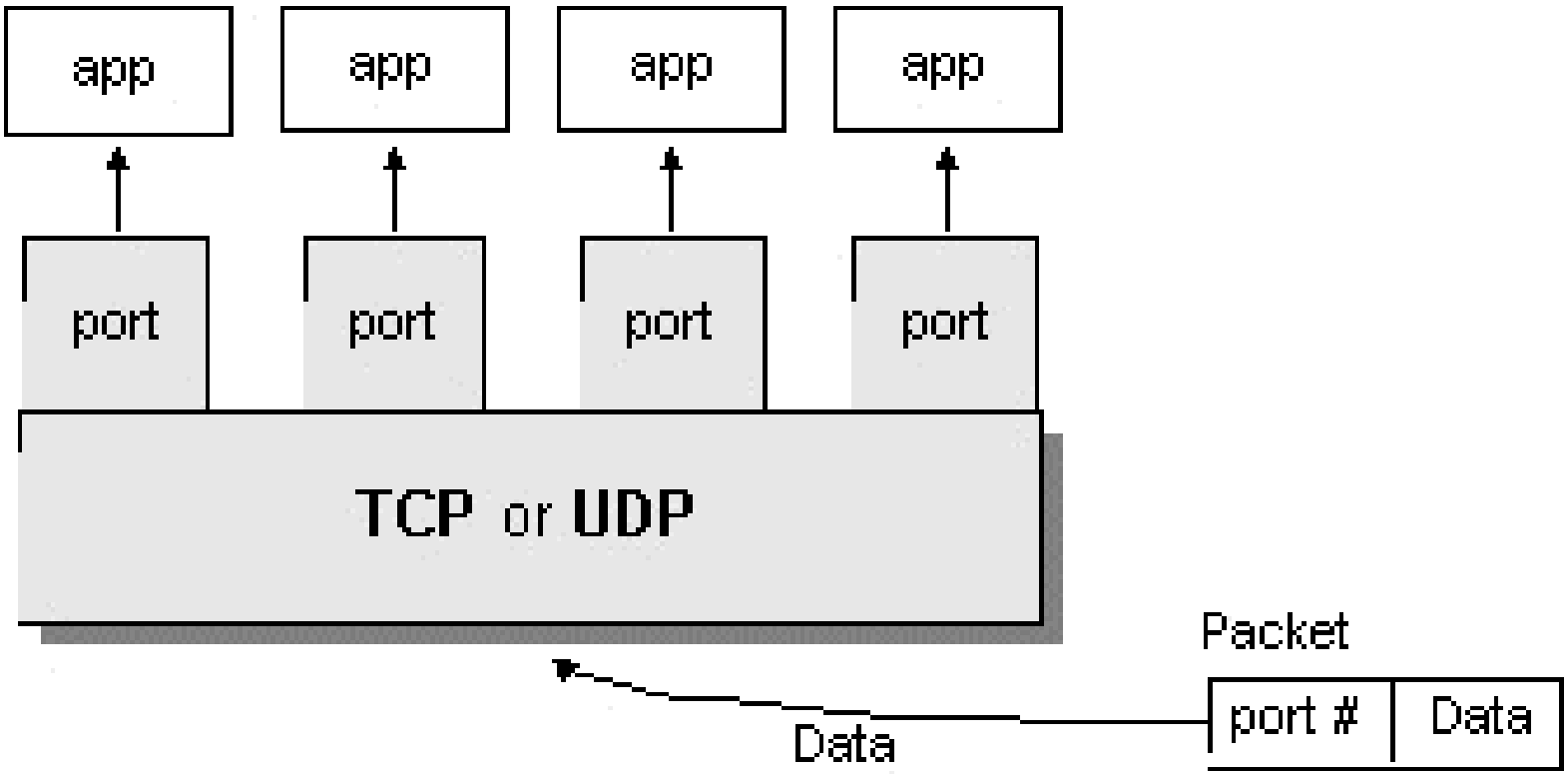
Connect "ladybug", …

1/2 14 h a l l o _ 2/2 l a d y b u g !

Route to ladybug.medow.sk

Route to 0095AC 0495AE

Internet **LAN/WAN** TCP/IP



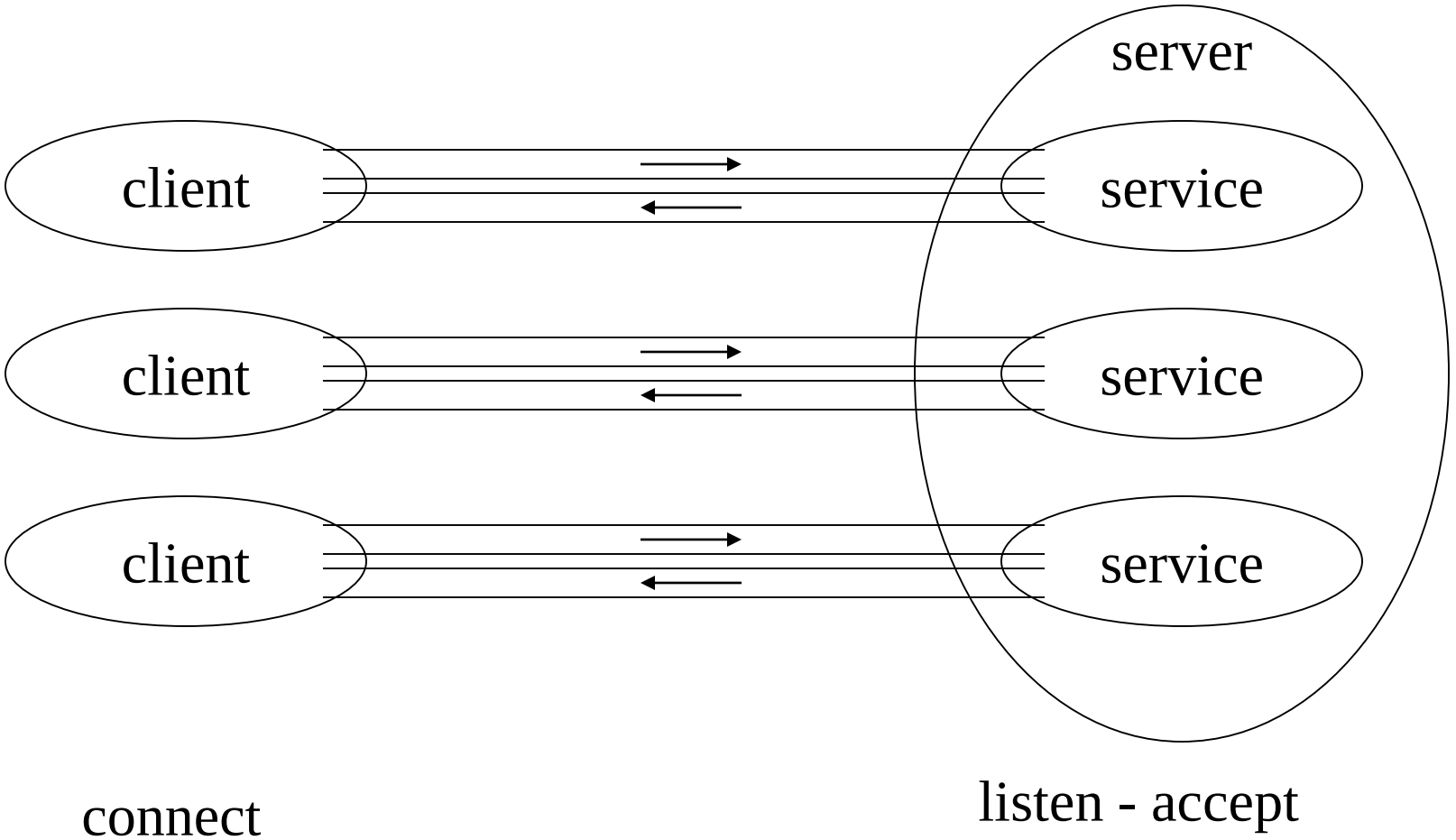

0 – 1023 - 65535

TCP/IP

# Application protocols of Internet

- daytime 13/tcp,udp
- ftp 21/tcp + further ports
- telnet 23/tcp
- ssh,scp 22/tcp
- DNS 53/tcp,udp
- pop3 110/tcp
- vnc 5900/tcp
- ntp 123/tcp
- tomcat cluster 4001/tcp
- snmp 161/udp
- gopher 70/tcp
- www-http 80/tcp
- irc 6667/tcp
- smtp 25/tcp
- www-https 443/tcp
- PostgreSQL 5432/tcp
- DRBD 7789/tcp
- Heartbeat 694/upd

# Socket


server
client
service
client
service
client
service
connect
listen - accept

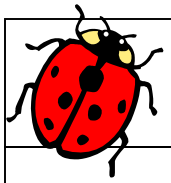
server

```
import java.io.*;
import java.net.*;

public class Server {
  public static final int PORT = 7171;

  public static void main(String[] args) throws IOException {
    ServerSocket s = new ServerSocket(PORT);
    System.out.println("Started: " + s);
    try {
      Socket socket = s.accept();
      try {
        System.out.println("Connection accepted: " + socket);
        BufferedReader in = new BufferedReader( new InputStreamReader(socket.getInputStream()));
        PrintWriter out = new PrintWriter(socket.getOutputStream(),true);
        while (true) {
          String str = in.readLine();
          if (str.equals("END")) break;
          System.out.println("Echoing: " + str);
          out.println(str);
        }
      } finally {
        System.out.println("closing...");
        socket.close();
      }
    } finally {
      s.close();
    }
  }
}
```

# client

```
import java.io.*;
import java.net.*;

public class Client {
  public static void main(String[] args) throws IOException {
    InetAddress addr = InetAddress.getByName("localhost");
    System.out.println("addr = " + addr);
    Socket socket = new Socket(addr, Server.PORT);
    try {
      System.out.println("socket = " + socket);
      BufferedReader in = new BufferedReader(new InputStreamReader(socket.getInputStream()));
      PrintWriter out = new PrintWriter(socket.getOutputStream(),true);
      for (int i = 0; i < 10; i++) {
        out.println("howdy " + i);
        String str = in.readLine();
        System.out.println(str);
      }
      out.println("END");
    } finally {
      System.out.println("closing...");
      socket.close();
    }
  }
}
```

*layers*



application

presentation

session

transport

network

datalink

physical

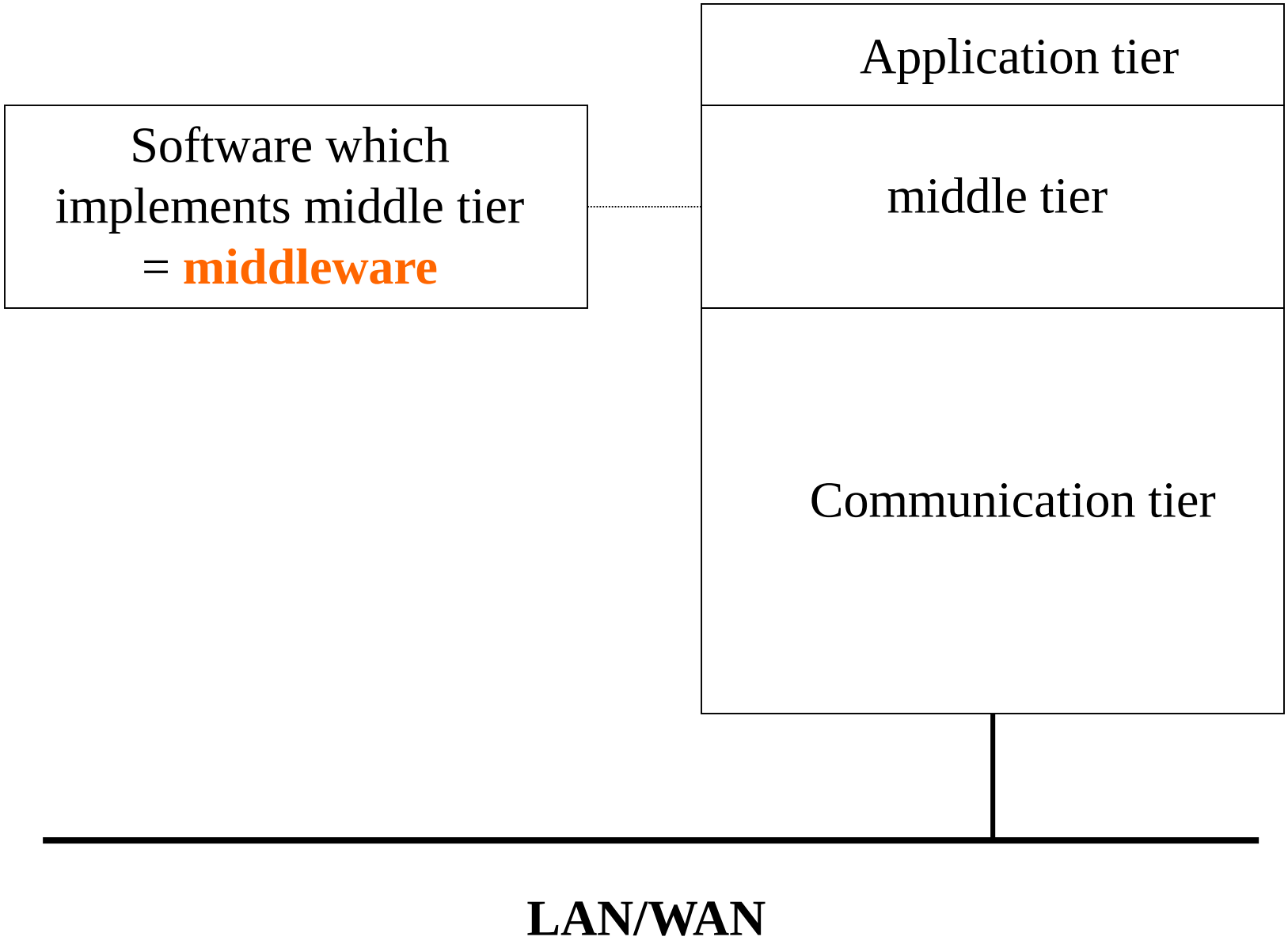
*tiers*

Application tier

?

Communication tier

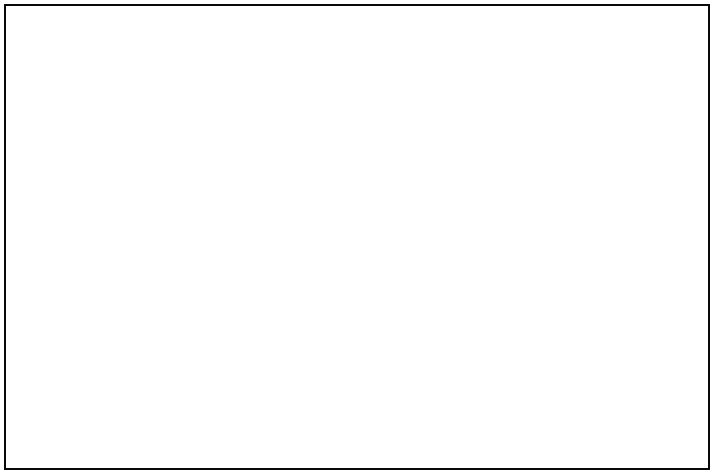
**LAN/WAN**


Application tier
Software which implements middle tier = middleware
middle tier
Communication tier
LAN/WAN

# Middleware

distributed objects

services



message passing

transactions

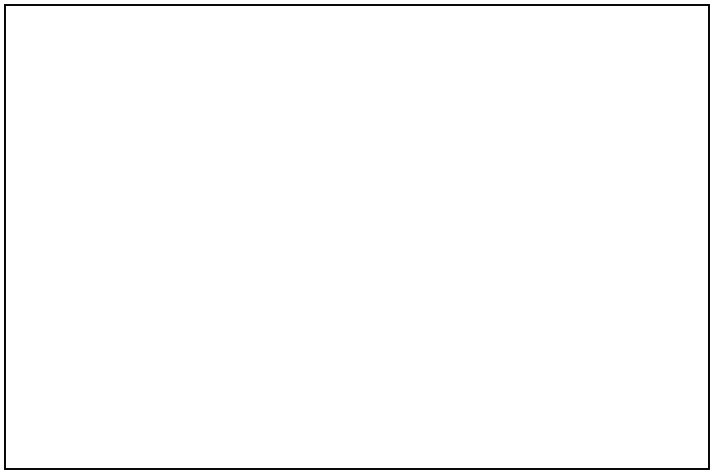
# Middleware

CORBA-POA,

RMI, COM+

RMI-IIOP

RPC, CORBA,
Web services

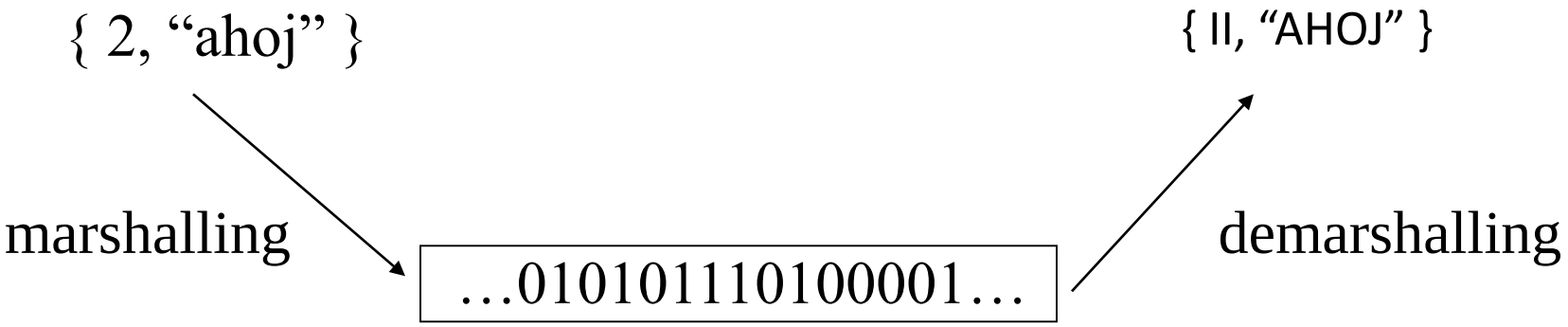


Middleware 2000

SQL

JADE, Aglobe,
Cougaar, ADE …

# Marshalling / Demarshalling




- binary formats
- text formats

# How to impelement marshalling?

- We need to turn an object into sequence of bytes or characters (the marshalled object) during the program course

# Reflection model

- Reflection model enables to find class of and any object instance dynamically (during course of program) and its attributes, methods and their argument types
- RM enable to write program which generates source code in programming language of interface for a given object instance
- JRM – implementation of RM in Java

# *Reflection model*

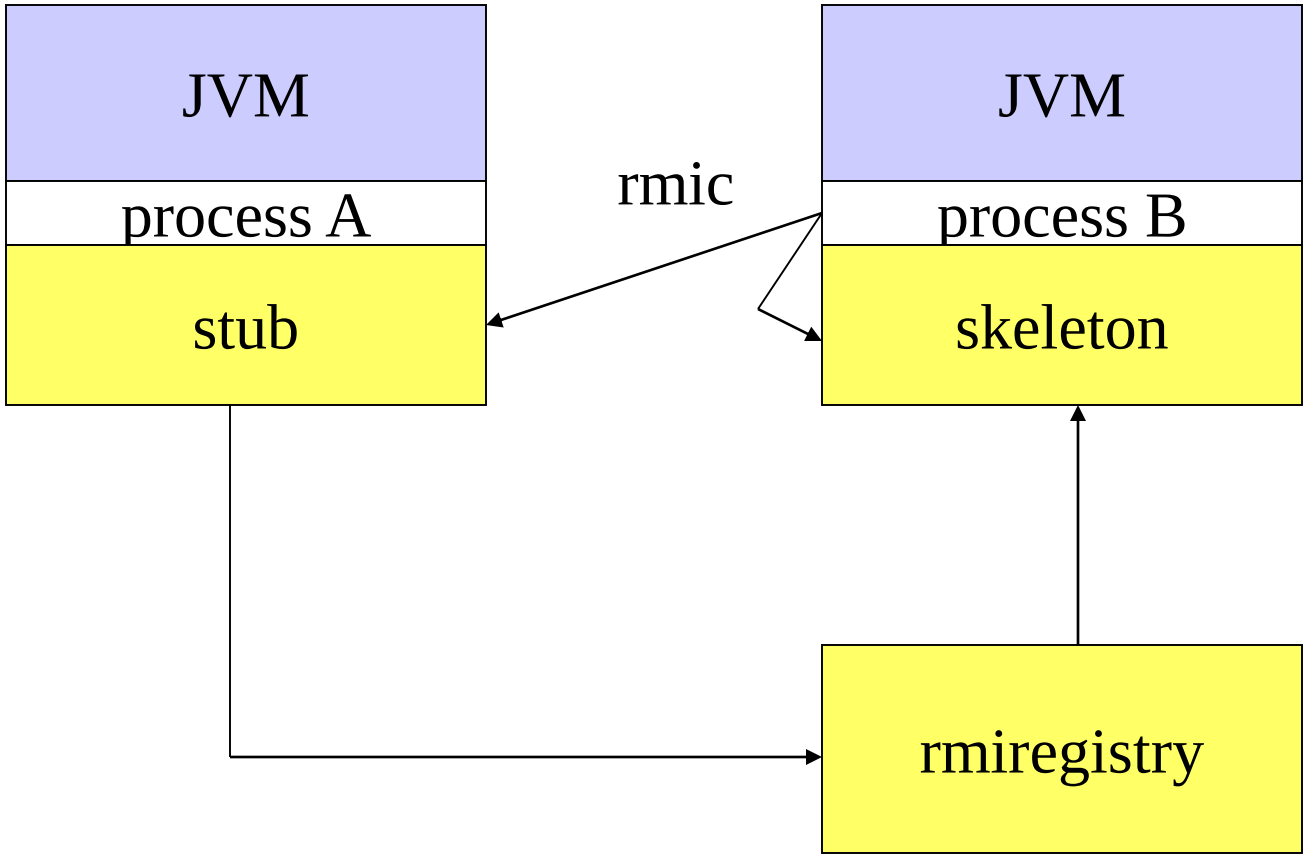
```
import java.lang.reflect.*;

public class RF {
  public static void main (String[] args) throws Exception {
    Gulka b = new Ball(1.0f);
    Class cl = g.getClass();
    Field[] field = cl.getDeclaredFields();
    for (int i=0; i<field.length; i++) {
      Object obj = field[i].get(b);
      System.out.println(field[i].getName()+" = "+obj);
    }
  }
}
```

```
serialVersionUID = 112132444
radius = 1.0
x = 0.0
y = 0.0
```

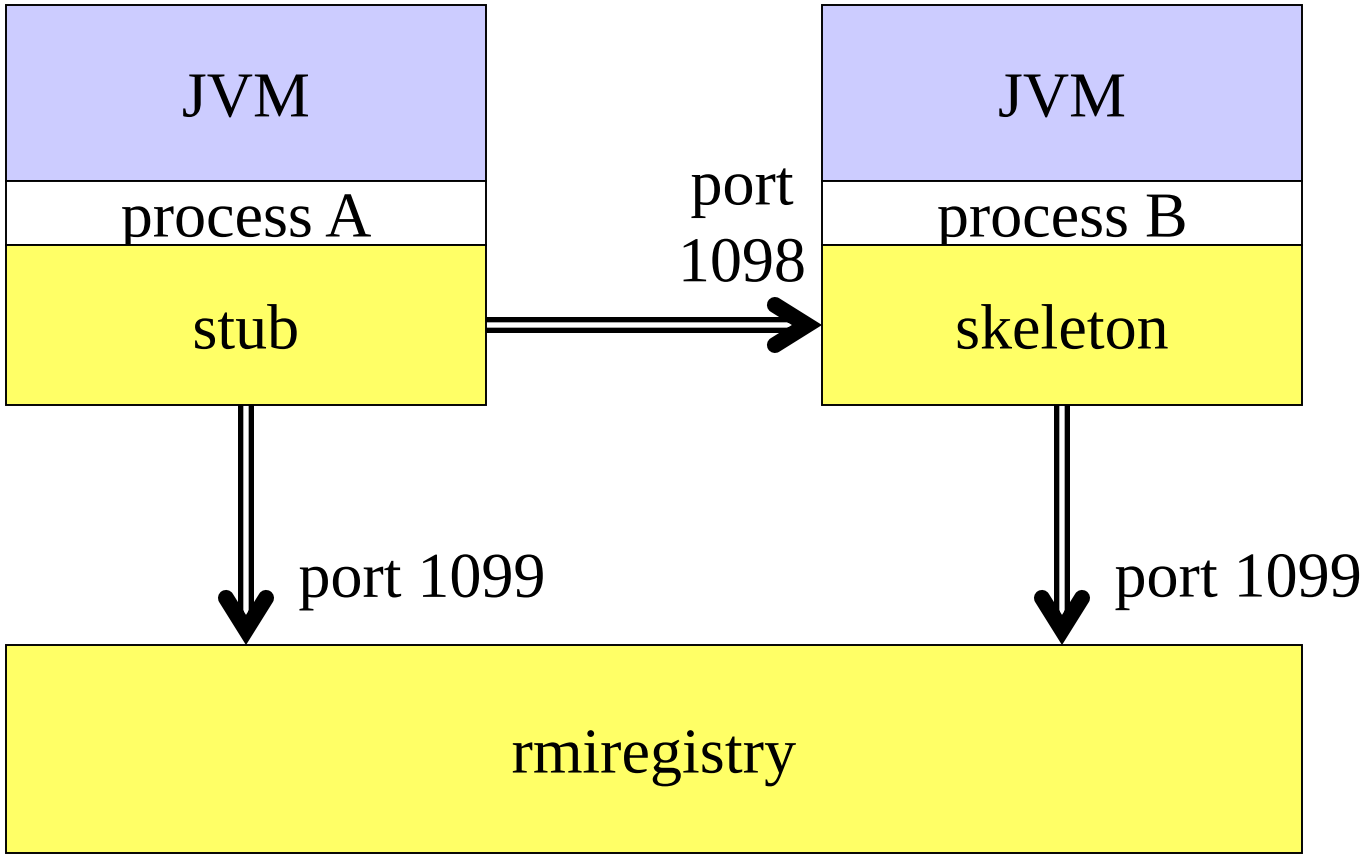
# Stub / Skeleton

- Stub implements marshalling a demarshalling at the calling side (client)
- Skeleton implements the same at the called side (server)
- Stub represents server at the client side
- Skeleton represents cliet at the server side
- If reflection model is not available, it is not possible to write an univerzal stub or skeleton = we have to generate them seperately for each object class (we ought to get an utility which provides that)

# RMI



# TCP sockets for RMI



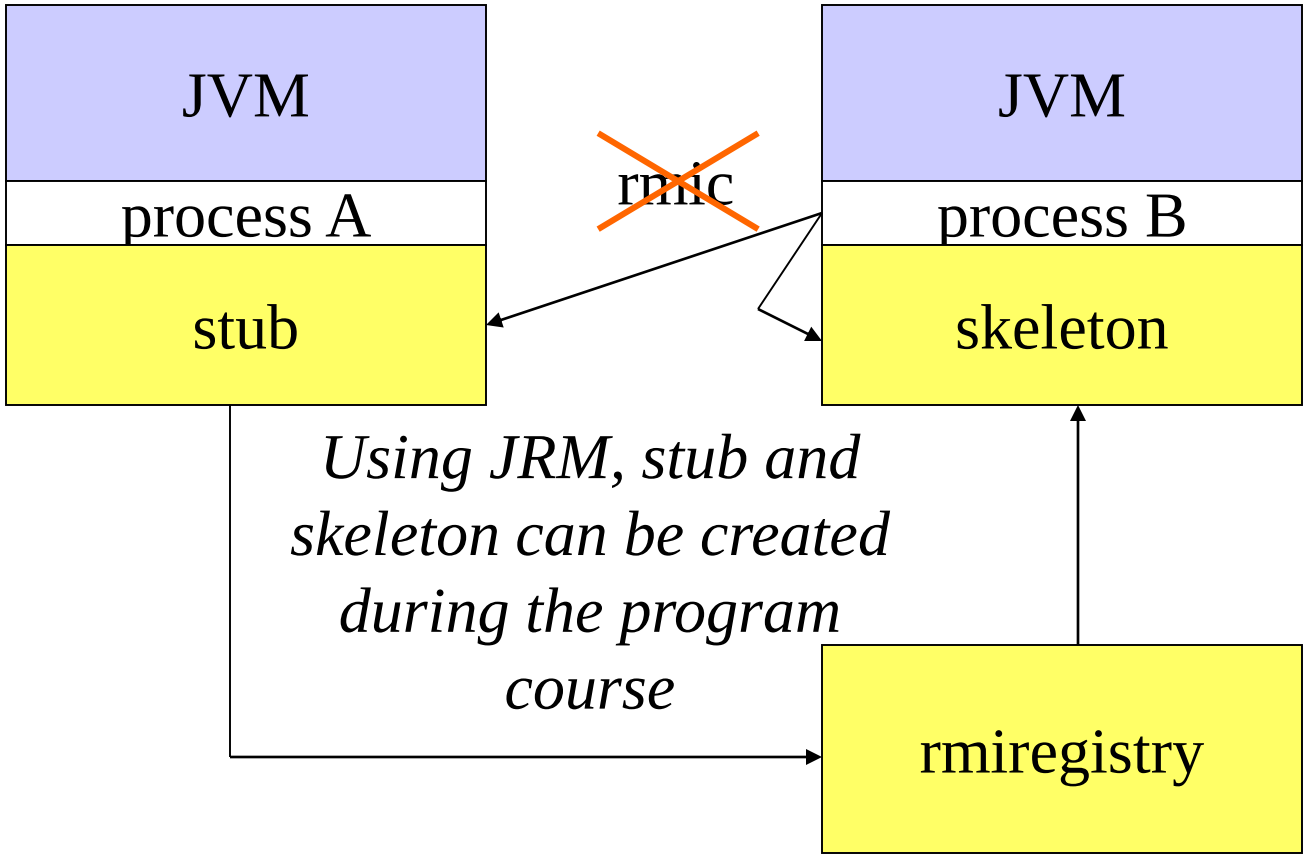
# RMI – how to use

- Implement interface which the distributed object is instance of. It is standard interface just each prototype throws RemoteException
- Implement part of server which register the object
- Implement application logic of the server
- Employ interface of the distributed object in client as if the object is available locally

# RMI

- Because of one language, RMI do not need IDL
- RMI employs native marshalling of Java (interface Serializable, serialVersionUID)
- RMI Registry serves for locating of the service
- RMI registry is TCP service, it is working on port 1099

# RMI using JRM



server

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

public interface Space extends Remote {
    void write (Serializable key, Serializable value) throws RemoteException;
    Serializable read (Serializable key) throws RemoteException;
    void delete (Serializable key) throws RemoteException;
}

public class RemoteSpace implements Space {
  public void write (Serializable key, Serializable value) throws RemoteException {...}
  public Serializable read (Serializable key) throws RemoteException {...}
  public void delete (Serializable key) throws RemoteException {...}

  public RemoteSpace() throws RemoteException {}

  public static void main(String args[]) {
    try {
      RemoteSpace obj = new RemoteSpace();
      Space stub = (Space) UnicastRemoteObject.exportObject(obj, 7171);
      Registry registry = LocateRegistry.getRegistry();
      registry.rebind("//158.195.28.151:7171/SPACE", stub);
      System.err.println("Server ready");
    } catch (Exception e) {
    }
  }
}
```

client

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

public class Client {

  public static void main(String[] args) {
    try {
      Registry registry = LocateRegistry.getRegistry("158.195.28.151"); // on port 1099
      Space space = (Space) registry.lookup("//158.195.28.151:7171/SPACE"); // on port 7171
      System.out.println(space.read("a"));
      space.write("a","aaaa");
      System.out.println(space.read("a"));
    } catch(Exception e) {
    }
  }

}
```