Multiagentové systémy

Andrej Lúčny

KAI FMFI UK

lucny@fmph.uniba.sk

http://www.agentspace.org/mas

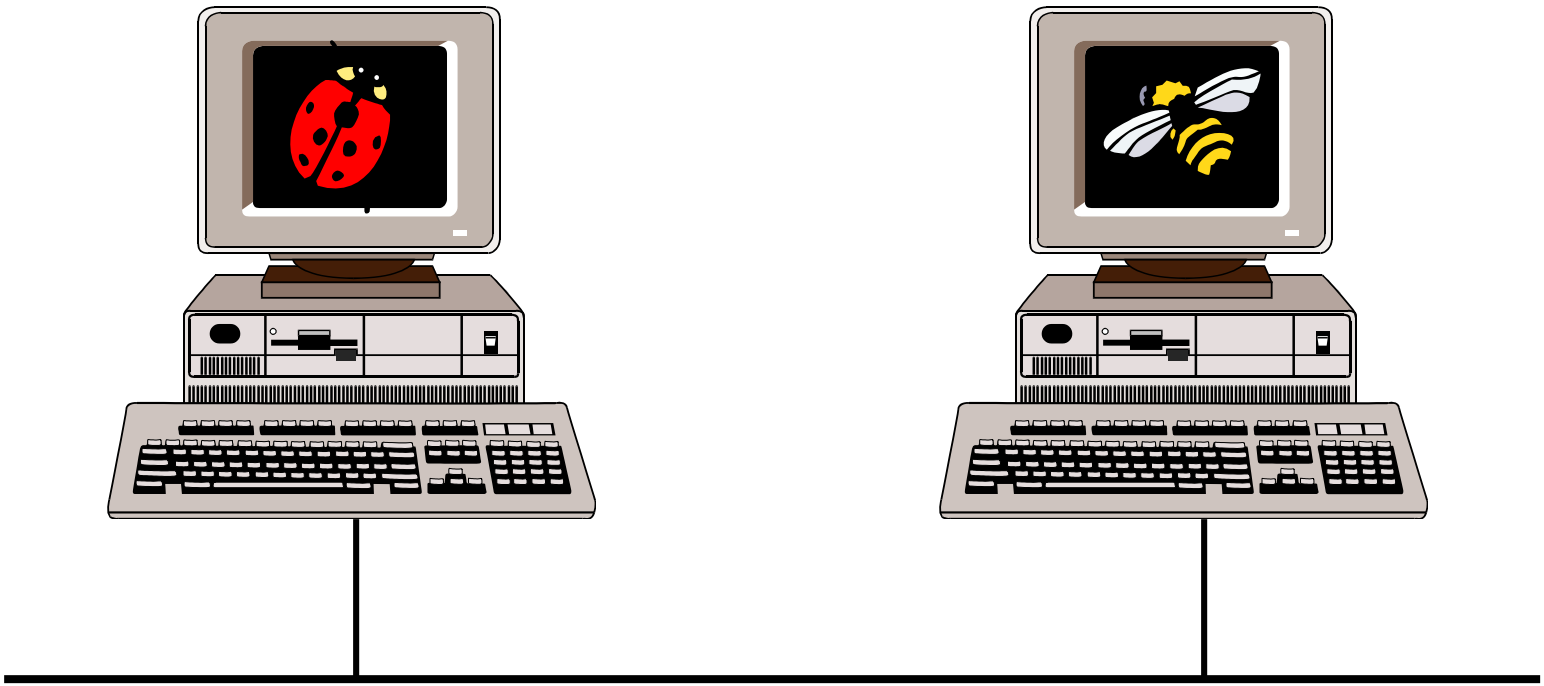
Implementácia MAS

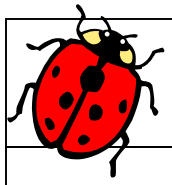
Spravidla sa opierame o nejaký existujúci sw.

- Middleware (distribuované prostredie)
- IPC (multiprocesové prostredie)
- VM s vláknami (multivláknové prostredie)

CORBA, RMI	SRR	JVM
	shared memory	
TCP/IP (TCP, HTTP)		
LAN-WAN	IPC	VM
Sieťové programovanie	„concurent“ programovanie	Objektovo orientované programovanie

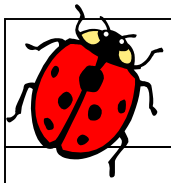
MAS ako middleware





OSI model

LAN/WAN

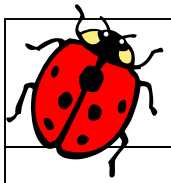


fyzická

Rieši fyzické záležitosti, popisuje médium po ktorom sa komunikuje

OSI model

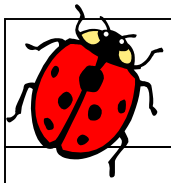
LAN/WAN



spojová	Fyzické adresy MAC a LCC napr konflikt pri vysielaní framu
fyzická	Rieši fyzické záležitosti, popisuje médium po ktorom sa komunikuje

OSI model

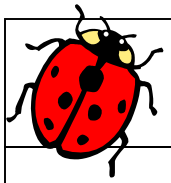
LAN/WAN



sieťová	logické adresy, routovanie, packety
spojová	Fyzické adresy MAC a LCC napr konflikt pri vysielaní framu
fyzická	Rieši fyzické záležitosti, popisuje médium po ktorom sa komunikuje

OSI model

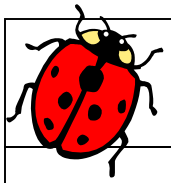
LAN/WAN



transportná	Prevod komunikovaných dát do podoby packeto a späť (streamy, datagramy)
sieťová	logické adresy, routovanie, packety
spojová	Fyzické adresy MAC a LCC napr konflikt pri vysielaní framu
fyzická	Rieši fyzické záležitosti, popisuje médium po ktorom sa komunikuje

OSI model

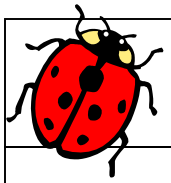
LAN/WAN



relačná	Nadväzovanie spojenia, riadenie dialógu
transportná	Prevod komunikovaných dát do podoby packeto a späť (streamy, datagramy)
sieťová	logické adresy, routovanie, packety
spojová	Fyzické adresy MAC a LCC napr konflikt pri vysielaní framu
fyzická	Rieši fyzické záležitosti, popisuje médium po ktorom sa komunikuje

OSI model

LAN/WAN

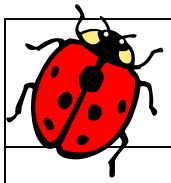


prezentačná
relačná
transportná
sieťová
spojová
fyzická

Dohoda na tvare komunikovaných dát
Nadväzovanie spojenia, riadenie dialógu
Prevod komunikovaných dát do podoby packeto a späť (streamy, datagramy)
logické adresy, routovanie, packety
Fyzické adresy MAC a LCC napr konflikt pri vysielaní framu
Rieši fyzické záležitosti, popisuje médium po ktorom sa komunikuje

OSI model

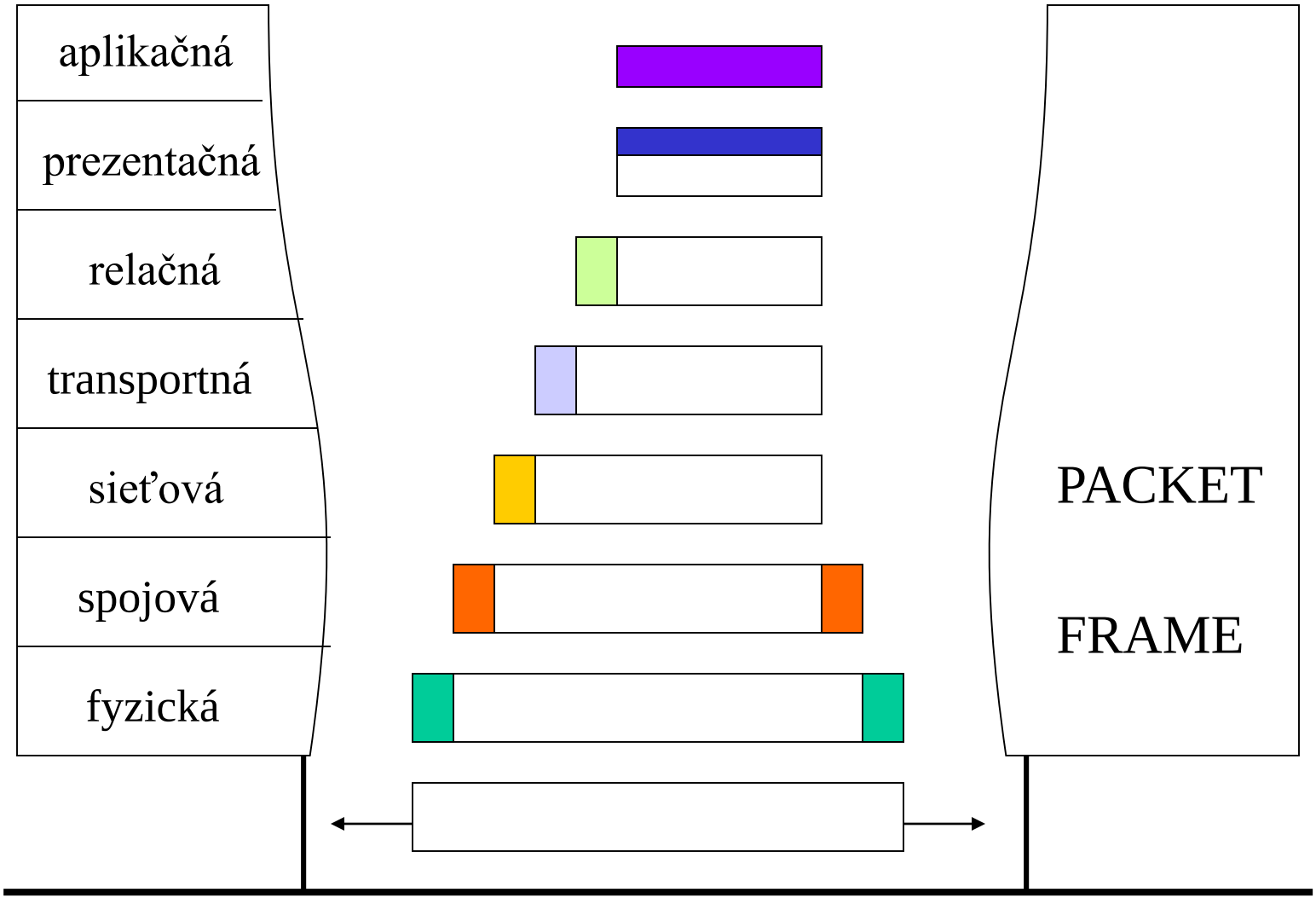
LAN/WAN



aplikačná	 Aplikačná logika, kód komunikujúceho programu
prezentačná	Dohoda na tvare komunikovaných dát
relačná	Nadväzovanie spojenia, riadenie dialógu
transportná	Prevod komunikovaných dát do podoby packeto a späť (streamy, datagramy)
sieťová	logické adresy, routovanie, packety
spojová	Fyzické adresy MAC a LCC napr konflikt pri vysielaní framu
fyzická	Rieši fyzické záležitosti, popisuje médium po ktorom sa komunikuje

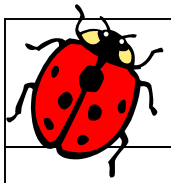
OSI model

LAN/WAN



OSI model

LAN/WAN



aplikačná

prezentačná

relačná

transportná

sieťová

spojová

fyzická

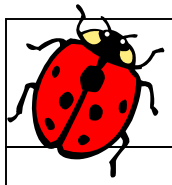
File Transfer FTP,
Simple Mail Transfer Protocol SMTP

Transmission control protocol TCP
User datagram procotol UDP

Internet protocol IP

EtherNet

Internet **LAN/WAN** TCP/IP



aplikačná

prezentačná

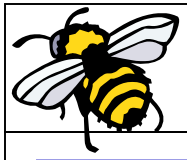
relačná

transportná

sieťová

spojová

fyzická



“ahoj lienka!”

12 a h o j _ l i e n k a !

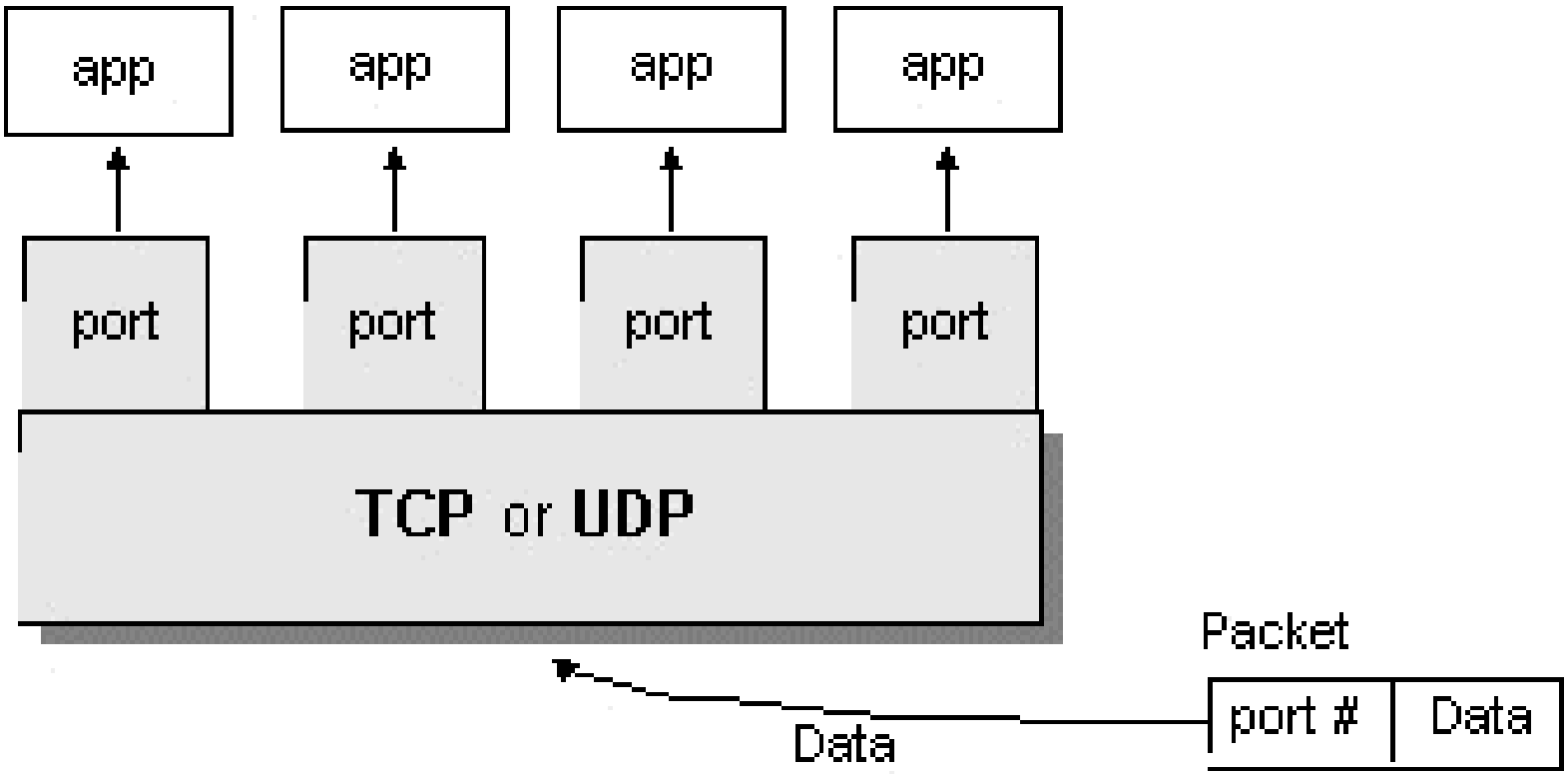
Connect “lienka”, …

1/2 12 a h o j _ 2/2 l i e n k a !

Route to lienka.luka.sk

Route to 0095AC 0495AE

Internet **LAN/WAN** TCP/IP



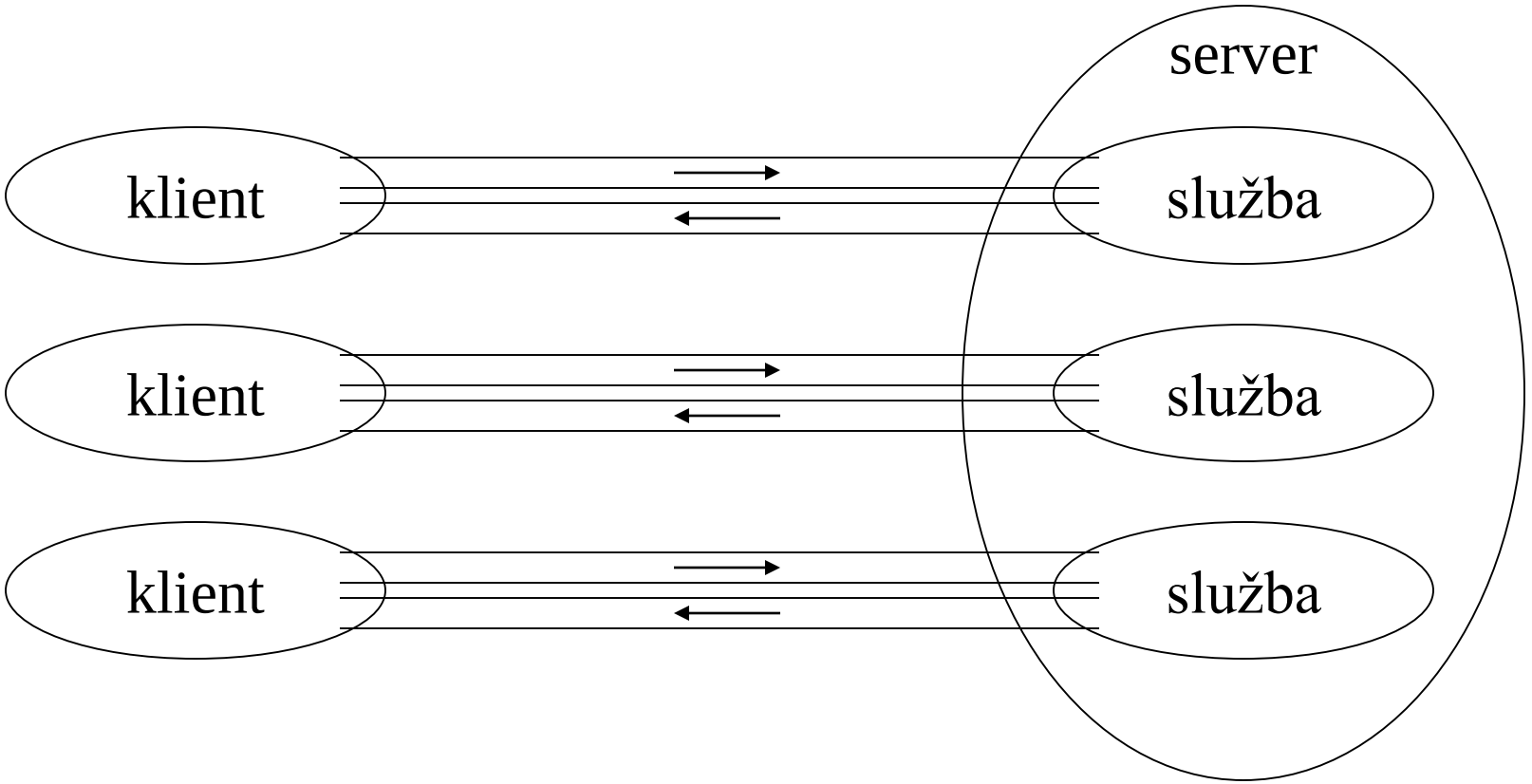
0 – 1023 - 65535

TCP/IP

Applikačné protokoly Internetu

- daytime 13/tcp,udp
- ftp 21/tcp + ďalšie porty
- 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

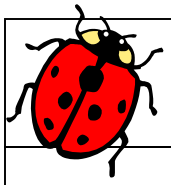

server
klient
služba
klient
služba
klient
služba
connect
listen - accept

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();
    }
  }
}
```

klient

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



aplikačná

prezentačná

relačná

transportná

sieťová

spojová

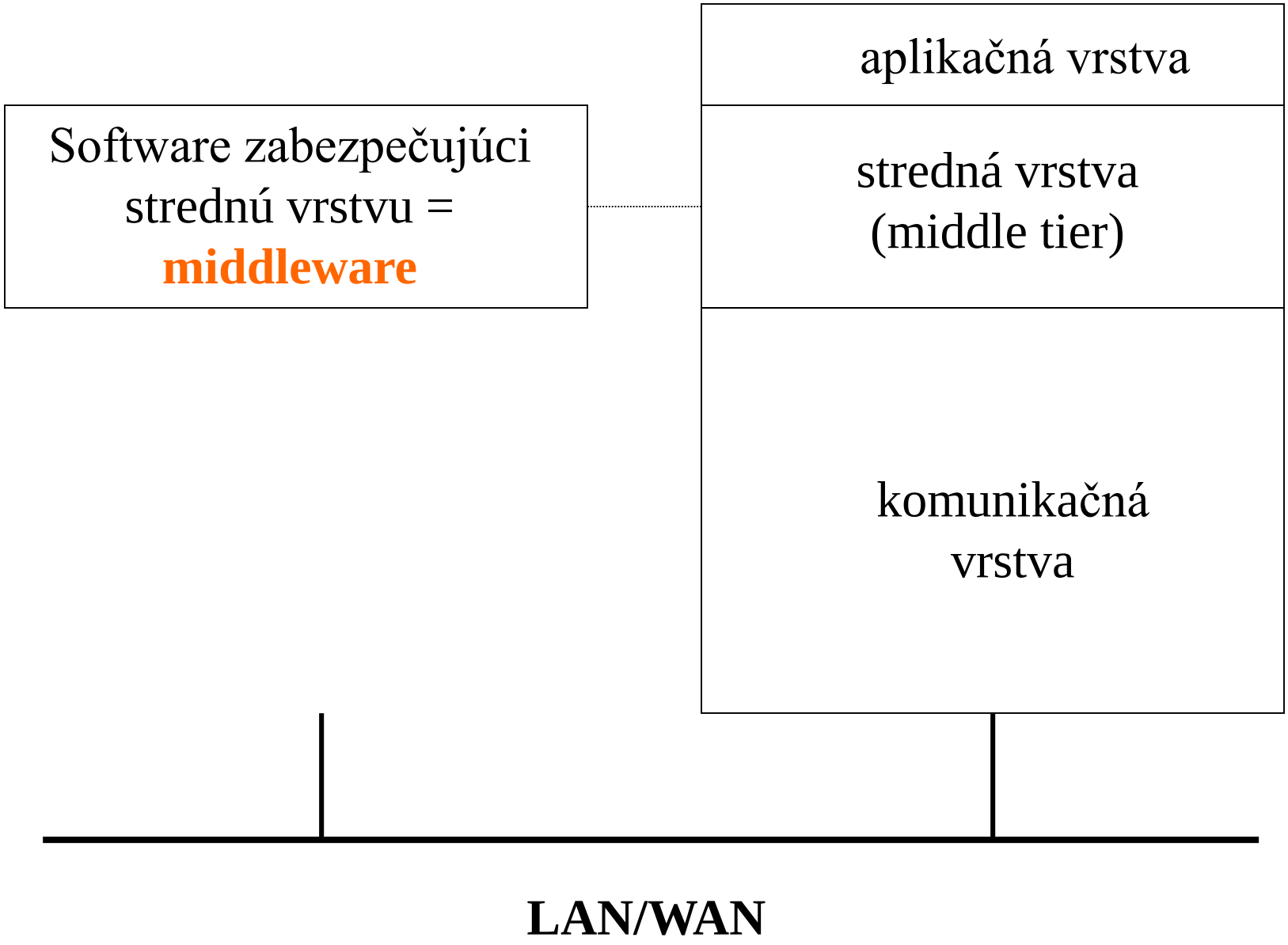
fyzická

aplikačná vrstva

?

komunikačná vrstva

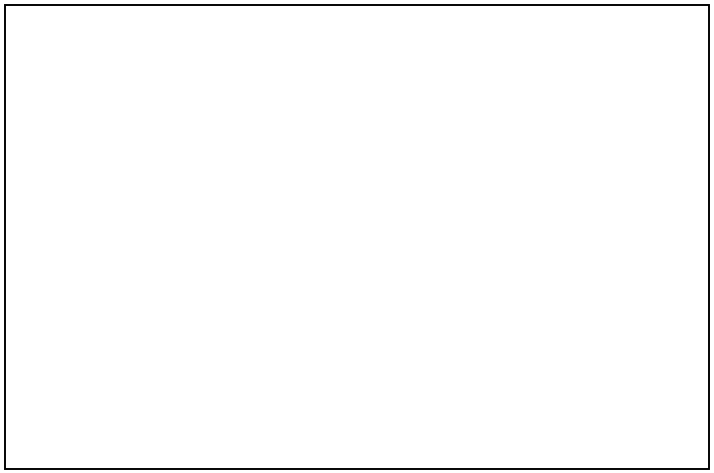
LAN/WAN


aplikačná vrstva
Software zabezpečujúci strednú vrstvu = middleware
stredná vrstva (middle tier)
komunikačná vrstva
LAN/WAN

Middleware

distribuované objekty

služby



správy

transakcie

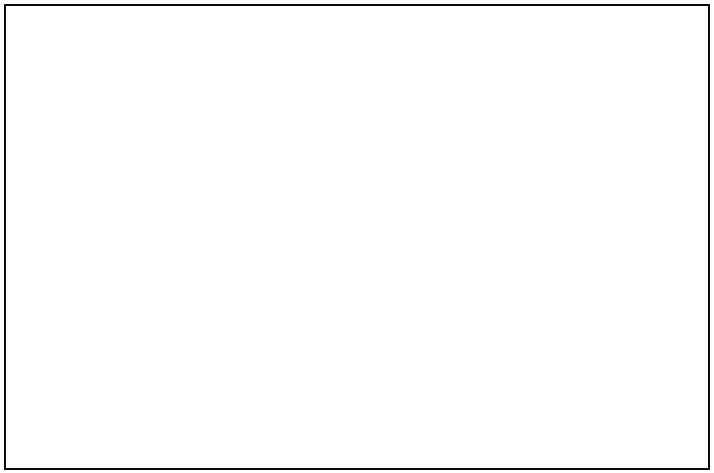
Middleware

CORBA-POA,

RMI, COM+

RMI-IIOP

RPC, CORBA,
Web services

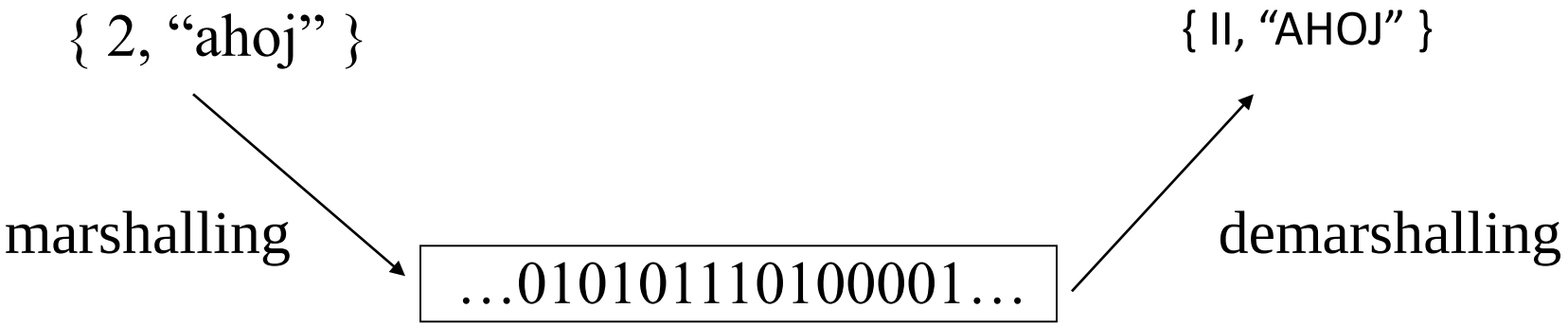


Middleware 2000

SQL

JADE, Aglobe,
Cougaar, ADE …

Marshalling / Demarshalling



- binárne formy
- textové (znakové) formy

Ako realizovať marshalling?

- Potrebujeme vedieť počas behu programu k danej inštancii (k danému objektu) vygenerovať jej podobu v postupnosti bytov či znakov (marshallovaný objekt)

IDL

- Interface definition language
- Opíšeme ako vyzerá rozhranie objektu v definičnom jazyku
- (mohli by sme definíciu v jednom z podporovaných jazykov povýšiť na IDL, ale definuje sa skôr špecifický jazyk)
- IDL používa napr. CORBA

IDL

```
module HelloApp
{
  interface Hello
  {
  string sayHello();
  oneway void shutdown();
  };
};
```

Reflekčný model

- Reflection model – umožňuje dynamicky zistiť triedu, premenné a metódy ľubovoľného objektu (introspekcia)
- RM umožňuje napr. napísať program, ktorý k ľubovoľnému objektu vygeneruje kód v programovacom jazyku, po skompilovaní ktorého dostaneme triedu objektu
- JRM – implementácia RM v Jave

Reflekčný model

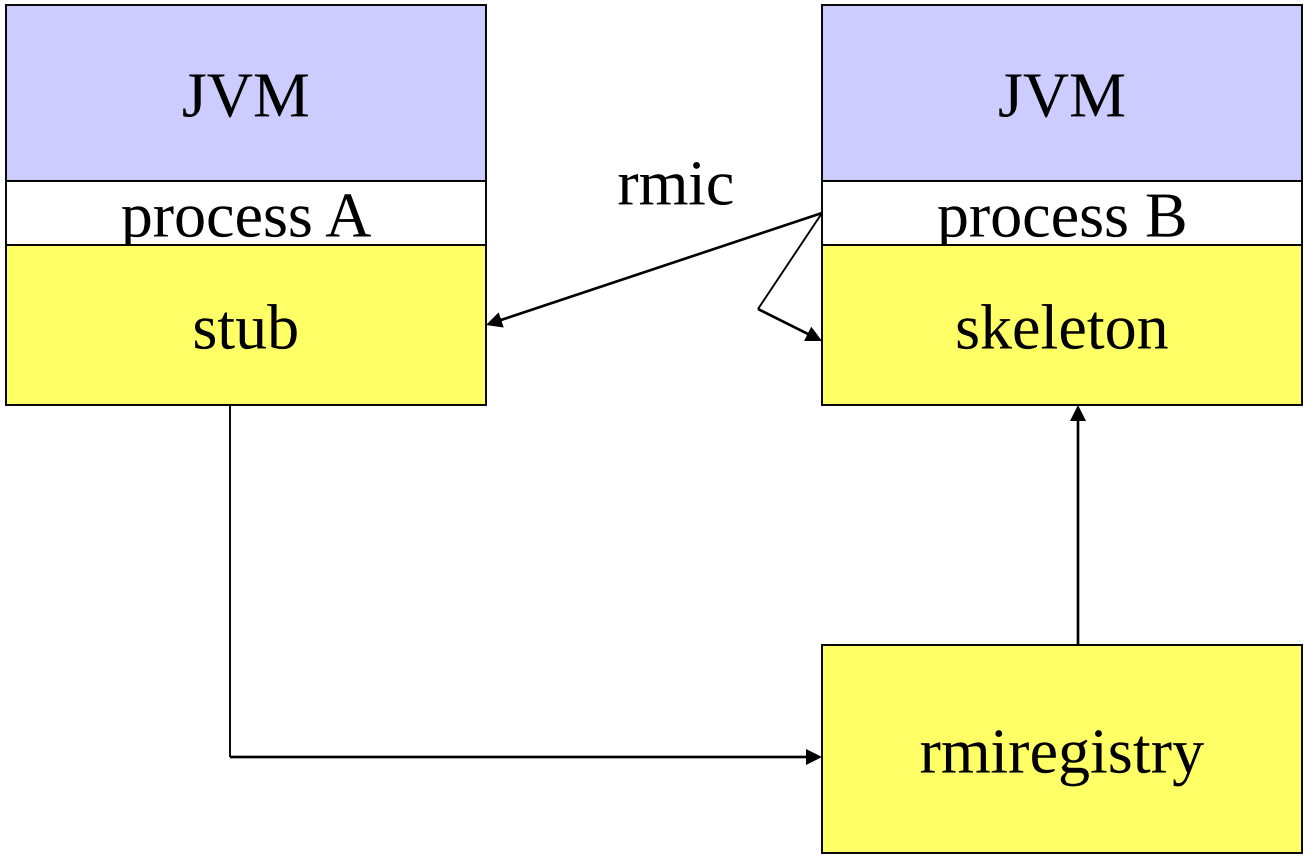
```
import java.lang.reflect.*;

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

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

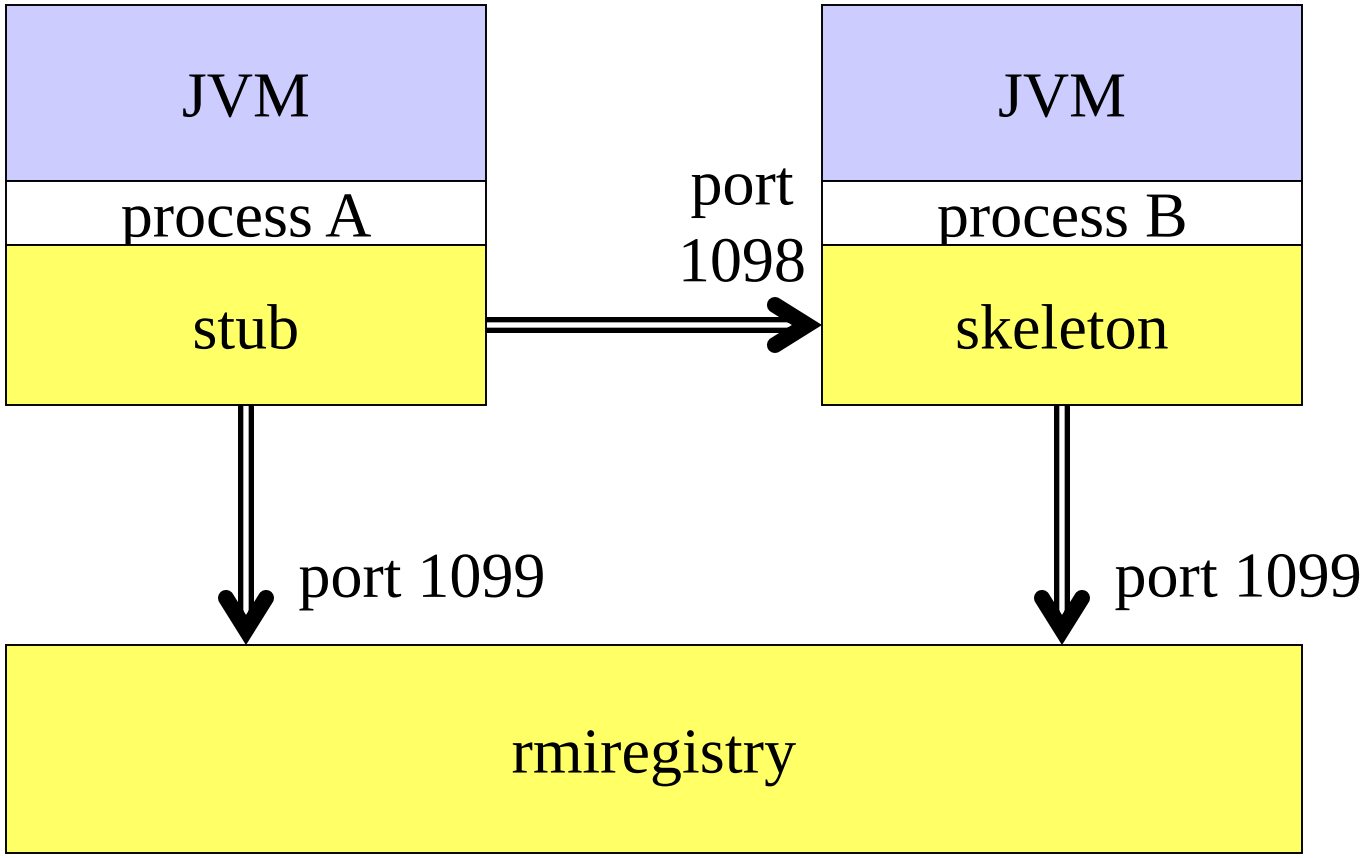
Stub / Skeleton

- Stub realizuje marshalling a demarshalling na strane volajúceho (klienta)
- Skeleton realizuje to isté u volaného (servera)
- Stub reprezentuje server u klienta
- Skeleton reprezentuje klient u servera
- Pokiaľ nemáme reflekčný model, nie je možné napísať univerzálny stub alebo skeleton = musíme ich vygenerovať pre každé použitie (máme na to nejakú utilitu)

RMI



TCP sockety pre RMI



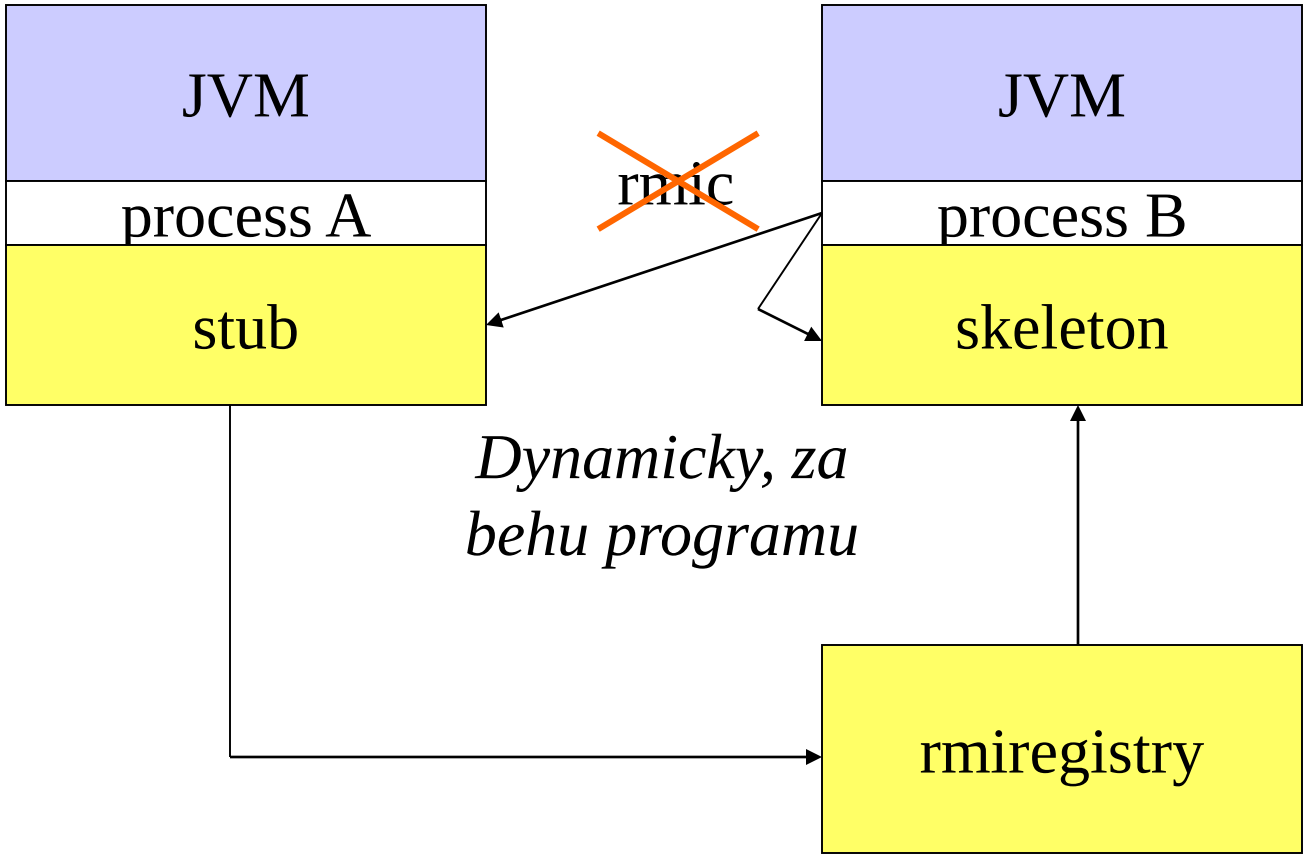
RMI - použitie

- Napíšeme rozhranie objektu, ktoré sa minimálne líši od bežných objektov
- Implementujeme časti vyplývajúce z potreby nadviazať spojenie
- Implementujeme logiku servera
- Používame rozhranie v klientovi ako keby to bol prítomný objekt

RMI

- Vďaka obmedzeniu sa na Java RMI nepotrebuje IDL (čo by sme nepotrebovali ak by sme lepšie využili JRM?)
- Využíva štandardný marshalling Javy (rozhranie Serializable, serialVersionUID)
- Na nájdenie servera sa používa RMI Registry (služba na báze TCP/IP)
- RMI registry pracuje na porte 1099

RMI s JRM


JVM
process A
stub
rmic
JVM
process B
skeleton
Dynamicky, za
behu programu
rmiregistry

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) {
    }
  }
}
```

klient

```
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) {
    }
  }

}
```