



Diplomski studij

**Informacijska i
komunikacijska tehnologija:**

Telekomunikacije i informatika

Računarstvo:

Programsko inženjerstvo i
informacijski sustavi

Računarska znanost

Ak.god. 2008./2009.

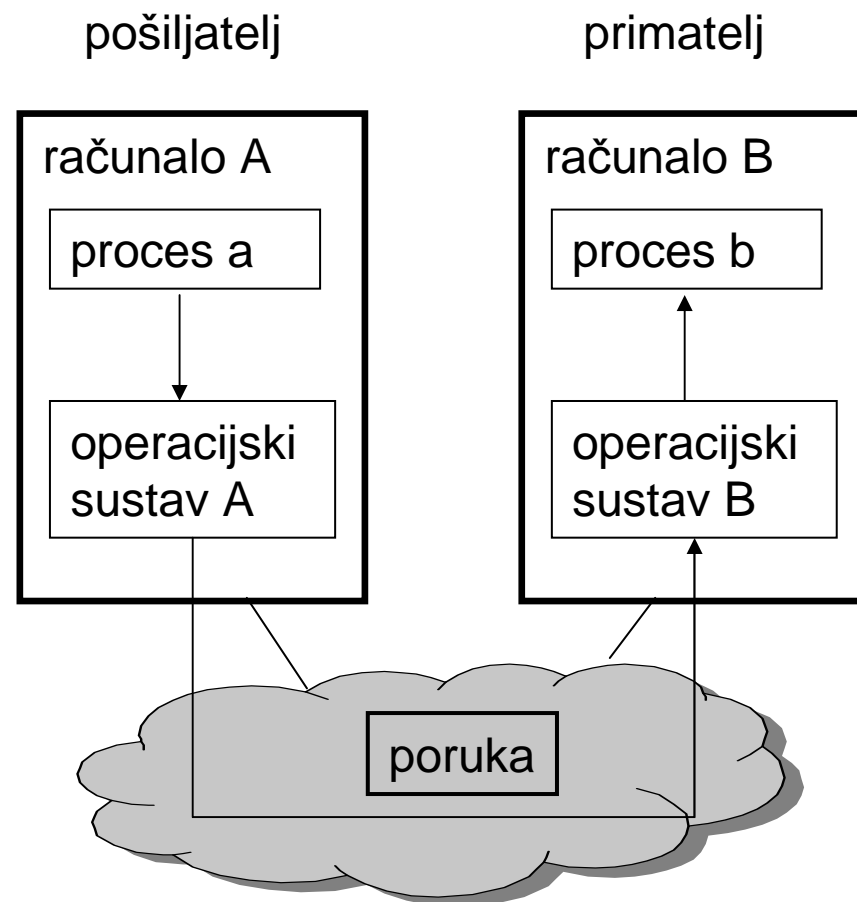
Raspodijeljeni sustavi

2.

Modeli raspodijeljene obrade:
Socket API, RPC i RMI

12.9.2008.

- ◆ Uvod
 - Interakcijski model
 - Komunikacija
 - Procesi
 - Međuoprema
- ◆ Međuoprema za komunikaciju raspodijeljenih procesa
 - Komunikacija korištenjem priključnica (Socket API)
 - Primjeri TCP/UDP klijenta i poslužitelja
 - Oblikovanje višedretvenog poslužitelja
 - Poziv udaljene procedure (*Remote Procedure Call* - RPC)
 - Poziv udaljene metode (*Remote Method Invocation* - RMI)



◆ Procesi

- izvode se na različitim računalima, autonomni su

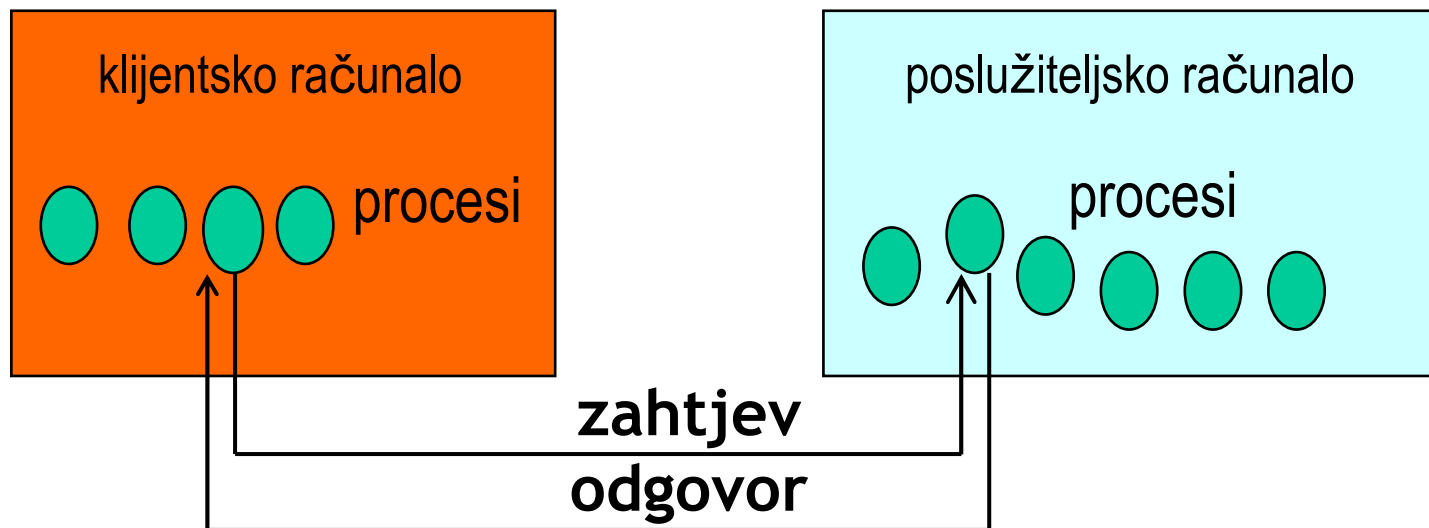
◆ Komunikacija

- prosljeđivanje poruka (engl. *message passing*), tj. razmjena poruka na mrežnom sloju

◆ Međuprocesna komunikacija

- engl. *interprocess communication* (IPC)
- potrebno je osigurati vremensku usklađenost (sinkronost) odvijanja i komunikacije procesa

Model klijent-poslužitelj



◆ KLIJENT

- zahtjeva uslugu
- šalje zahtjev poslužitelju i čeka odgovor

◆ POSLUŽITELJ

- nudi usluge
- prima i obrađuje dolazne zahtjeve te šalje odgovor klijentima

- ◆ Uvod
 - Interakcijski model
 - **Komunikacija**
 - Procesi
 - Međuoprema
- ◆ Međuoprema za komunikaciju raspodijeljenih procesa
 - Komunikacija korištenjem priključnica (Socket API)
 - Primjeri TCP/UDP klijenta i poslužitelja
 - Oblikovanje višedretvenog poslužitelja
 - Poziv udaljene procedure (*Remote Procedure Call* - RPC)
 - Poziv udaljene metode (*Remote Method Invocation* - RMI)

- ◆ **konekcijska**

- procesi eksplicitno kreiraju konekciju prije razmjene podataka, postoje kontrolne poruke za uspostavu konekcije

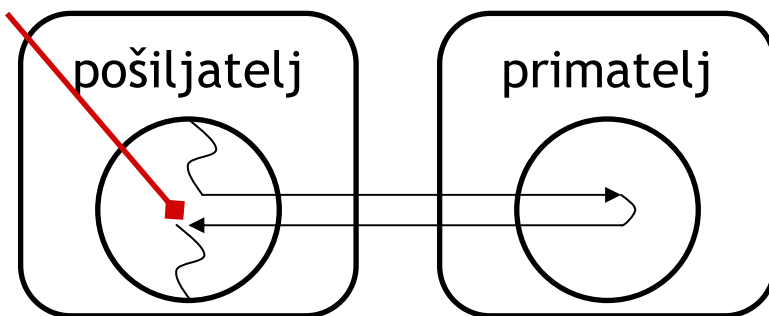
- ◆ **bezkonekcijska**

- sve poruke prenose podatke, nema kontrolnih poruka za uspostavu konekcije među procesima

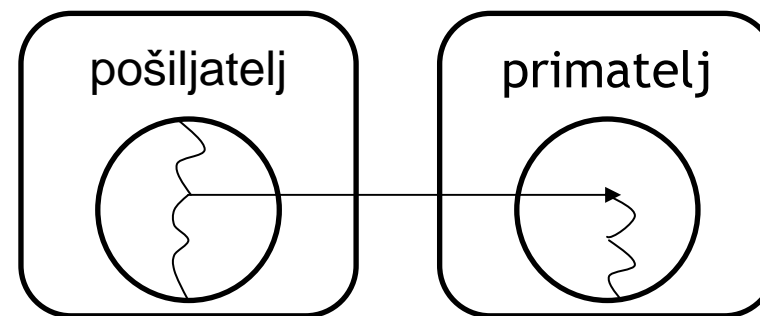
- ◆ **perzistentna komunikacija**
 - garantira isporuku poruke, poruka se pohranjuje u sustavu i isporučuje primatelju kada je to moguće
- ◆ **tranzijentna komunikacija**
 - nepouzdana, garantira isporuku poruke samo ako su pošiljalac i primatelj poruke istovremeno dostupni

- ♦ **sinkrona komunikacija**
 - blokira pošiljatelja do primitka potvrde o isporuci poruke primatelju
- ♦ **asinkrona komunikacija**
 - omogućuje pošiljatelju nastavak procesiranja odmah nakon slanja poruke

blokiranje

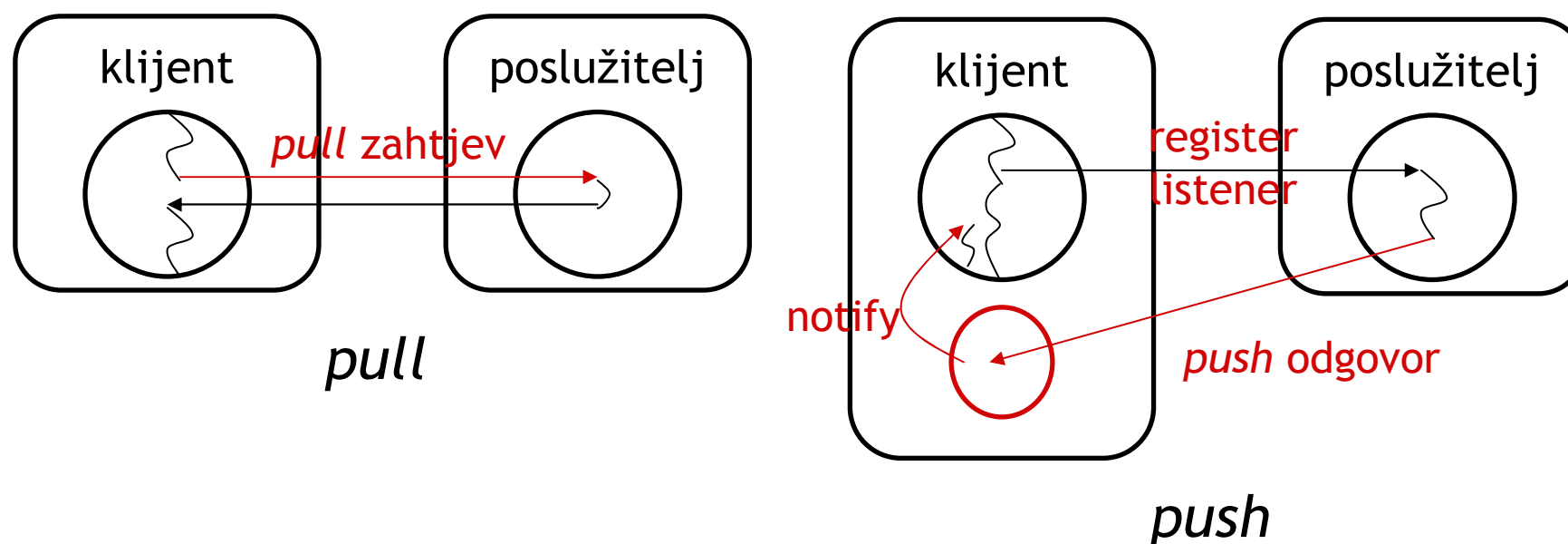


sinkrona
komunikacija



asinkrona
komunikacija

- ◆ **Komunikacija na načelu *pull* ili *push***
 - *pull* – “klasični” model zahtjev-odgovor
 - *push* – klijent registrira zahtjev i “sluša” odgovor, poslužitelj šalje odgovor nakon što završi obradu zahtjeva



- ◆ Uvod
 - Interakcijski model
 - Komunikacija
 - **Procesi**
 - Međuoprema
- ◆ Međuoprema za komunikaciju raspodijeljenih procesa
 - Komunikacija korištenjem priključnica (Socket API)
 - Primjeri TCP/UDP klijenta i poslužitelja
 - Oblikovanje višedretvenog poslužitelja
 - Poziv udaljene procedure (*Remote Procedure Call* - RPC)
 - Poziv udaljene metode (*Remote Method Invocation* - RMI)

- ◆ Definira se kao program u izvođenju (prisjetimo se operacijskih sustava)
- ◆ Višedretvenost je važna za efikasnu implementaciju raspodijeljenih procesa
 - omogućuje održavanje više logičkih konekcija s jednim procesom
 - višedretveni poslužitelj može paralelno obrađivati korisničke zahtjeve
 - višedretveni klijent može nastaviti s procesiranjem dok čeka odgovor poslužitelja (primjer: Web preglednik)

♦ vremenska (ne)ovisnost

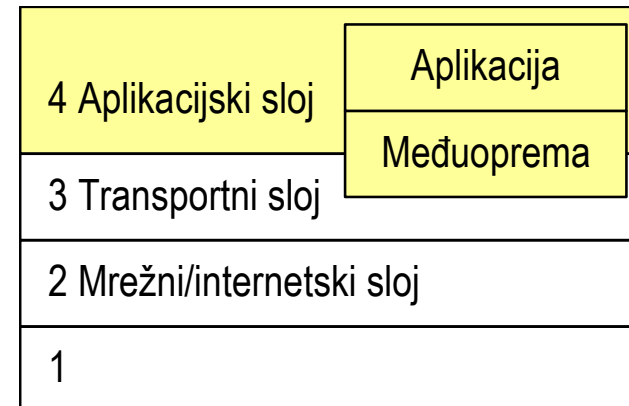
- vremenski ovisni procesi moraju biti istovremeno aktivni za realizaciju komunikacije
- vremenski neovisni procesi mogu komunicirati i ako nisu istovremeno aktivni

♦ ovisnost o referenci “sugovornika”

- proces je ovisan o referenci “sugovornika” ako mora znati jedinstveni identifikator (adresu) udaljenog procesa s kojim želi komunicirati
- proces može biti i neovisan o referenci, tj. ne mora znati jedinstveni identifikator udaljenog procesa

- ◆ Uvod
 - Interakcijski model
 - Komunikacija
 - Procesi
 - **Međuoprema**
- ◆ Međuoprema za komunikaciju raspodijeljenih procesa
 - Komunikacija korištenjem priključnica (Socket API)
 - Primjeri TCP/UDP klijenta i poslužitelja
 - Oblikovanje višedretvenog poslužitelja
 - Poziv udaljene procedure (*Remote Procedure Call* - RPC)
 - Poziv udaljene metode (*Remote Method Invocation* - RMI)

- ◆ Distribuirani sustavi koriste **međuopremu** (*middleware*), programsku infrastrukturu koja pruža **generičke usluge za jednostavniji razvoj distribuiranih aplikacija**
- ◆ U internetskom modelu međuoprema je smještena na aplikacijskom sloju između transportnog sloja i aplikacije



Međuoprema za komunikaciju raspodijeljenih procesa

- ◆ vrsta međuopreme za realizaciju komunikacije među udaljenim procesima
- ◆ protokoli za komunikaciju distribuiranih procesa na višem nivou apstrakcije od transportnog sloja
- ◆ omogućuje jednostavniji razvoj distribuiranih aplikacija, sakriva kompleksnost i heterogenost nižih slojeva

- ◆ Postojeća rješenja za komunikaciju raspodijeljenih procesa
 - komunikacija korištenjem priključnica (*socket API*)
 - poziv udaljene procedure (*remote procedure call*, RPC)
 - distribuirani objekti - poziv udaljene metode (*remote method invocation*, RMI)
 - komunikacija razmjenom poruka (*message-oriented interaction*)
 - model objavi-pretplati (*publish/subscribe*)

- ◆ Uvod
 - Interakcijski model
 - Komunikacija
 - Procesi
 - Međuoprema
- ◆ Međuoprema za komunikaciju raspodijeljenih procesa
 - Komunikacija korištenjem priključnica (Socket API)
 - Primjeri TCP/UDP klijenta i poslužitelja
 - Oblikovanje višedretvenog poslužitelja
 - Poziv udaljene procedure (*Remote Procedure Call* - RPC)
 - Poziv udaljene metode (*Remote Method Invocation* - RMI)

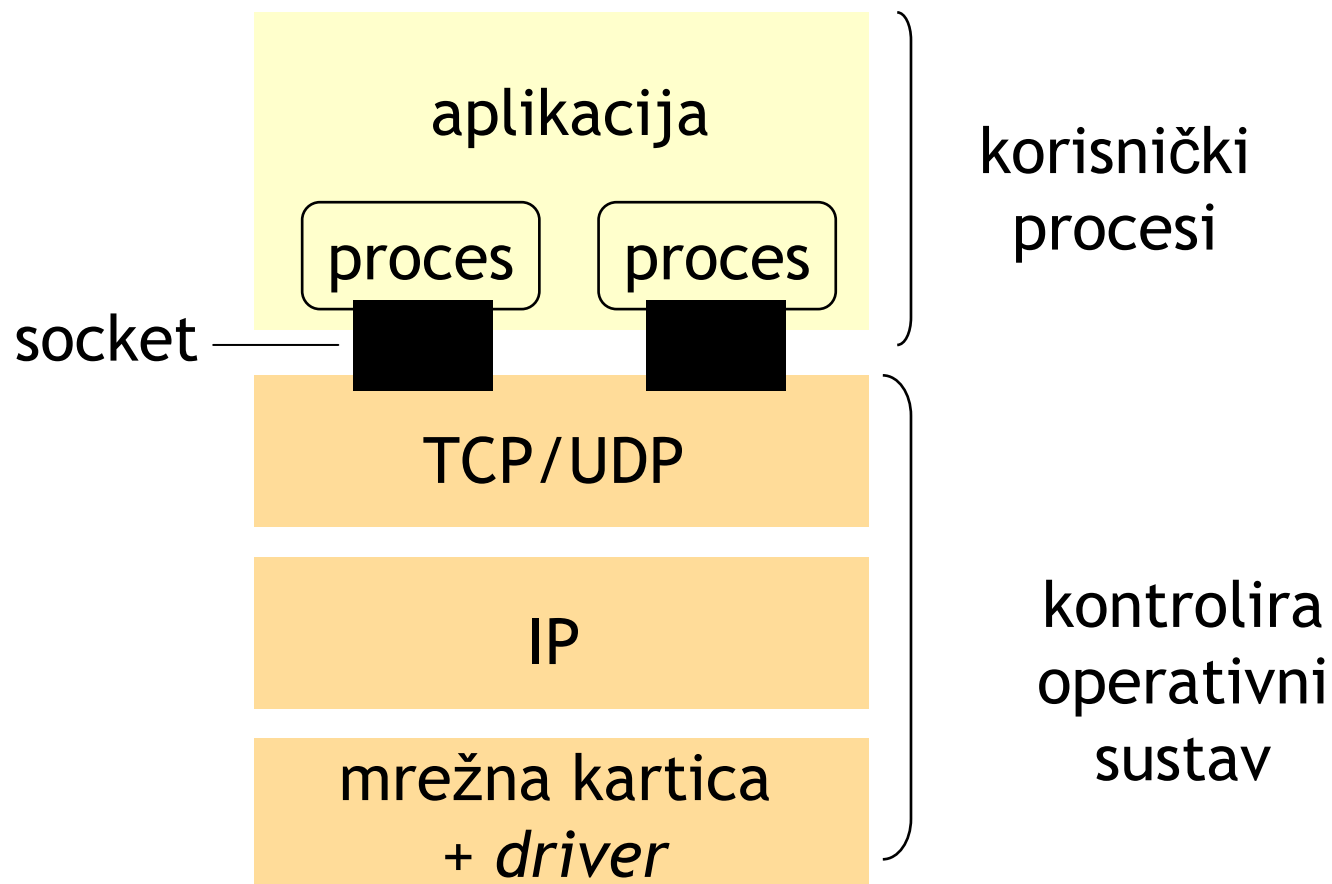
Socket API

- ◆ koristi funkcionalnost transportnog sloja
 - TCP – konekcijski protokol, pouzdan prijenos podataka
 - UDP – prijenos nezavisnih paketa (*datagrami*), nepouzdan prijenos
- ◆ priključnica (engl. *socket*)
 - komunikacijska točka preko koje aplikacija šalje podatke u mrežu i iz koje čita primljene podatke
 - viši nivo apstrakcije nad komunikacijskom točkom koju operativni sustav koristi za pristup transportnom sloju
 - veže se uz broj vrata (engl. *port*) koja jednoznačno određuju aplikaciju kojoj su poruke namijenjene

Komunikacija pomoću *socket*a

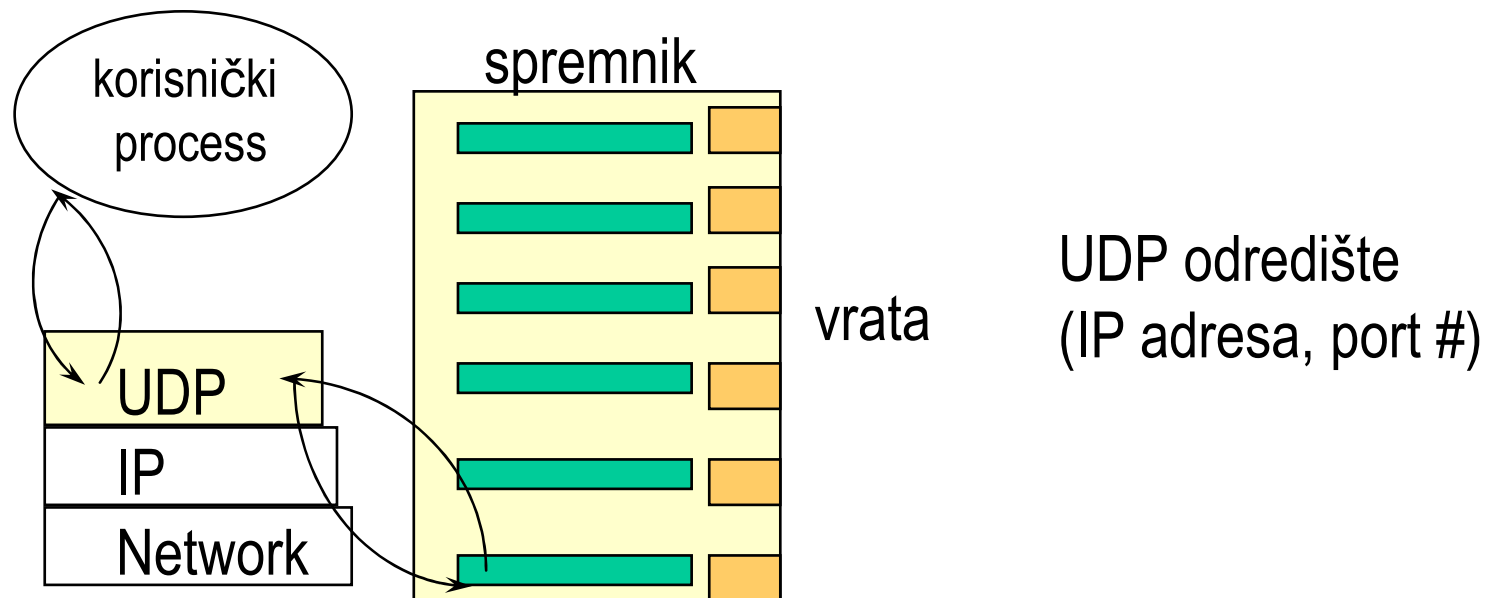


Zavod za telekomunikacije



User Datagram Protocol (UDP)

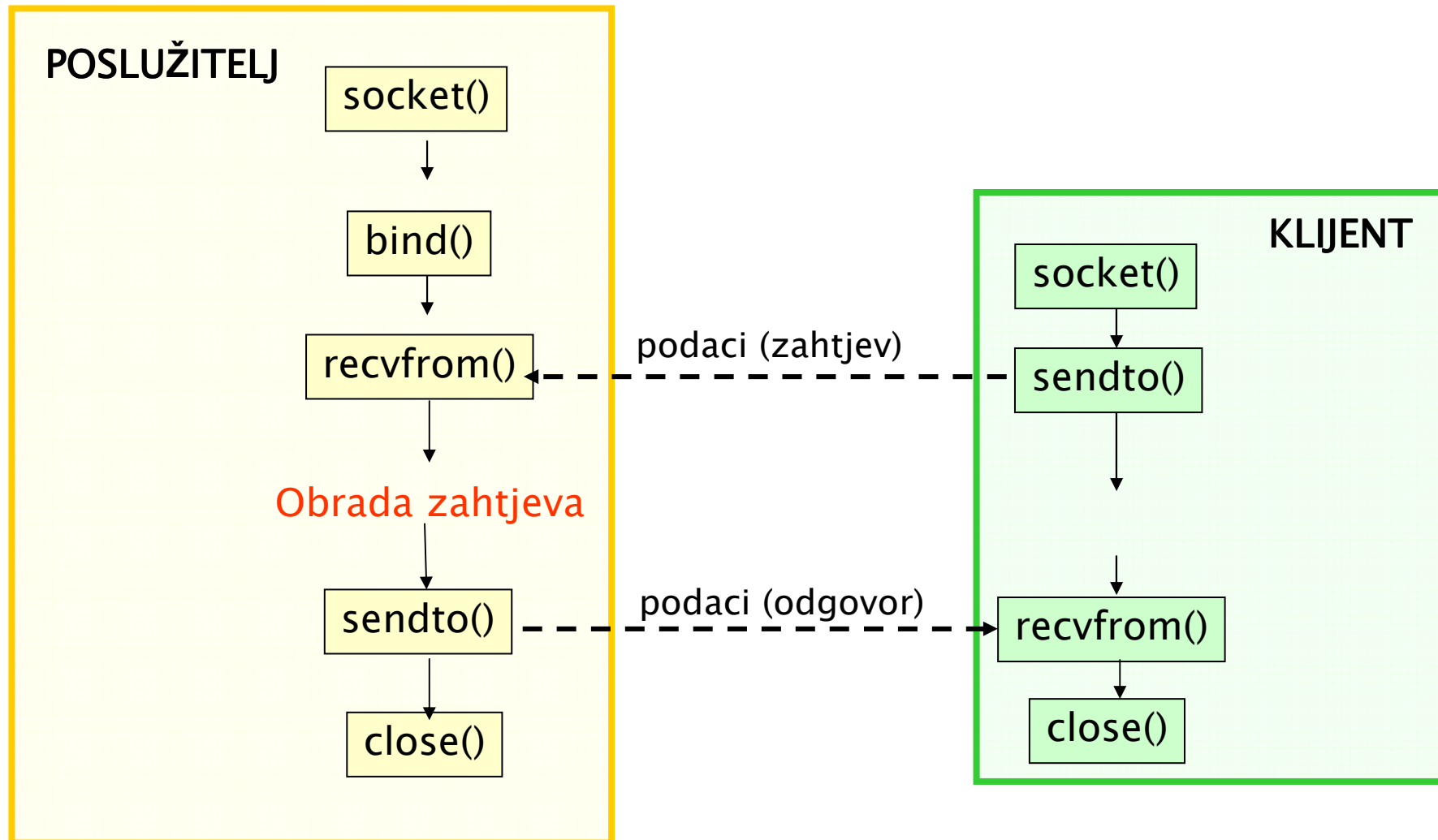
- komunikacija se odvija preko vrata (engl. *portova*) koje dodjeljuje operacijski sustav



Komunikacija pomoću socketa *UDP*



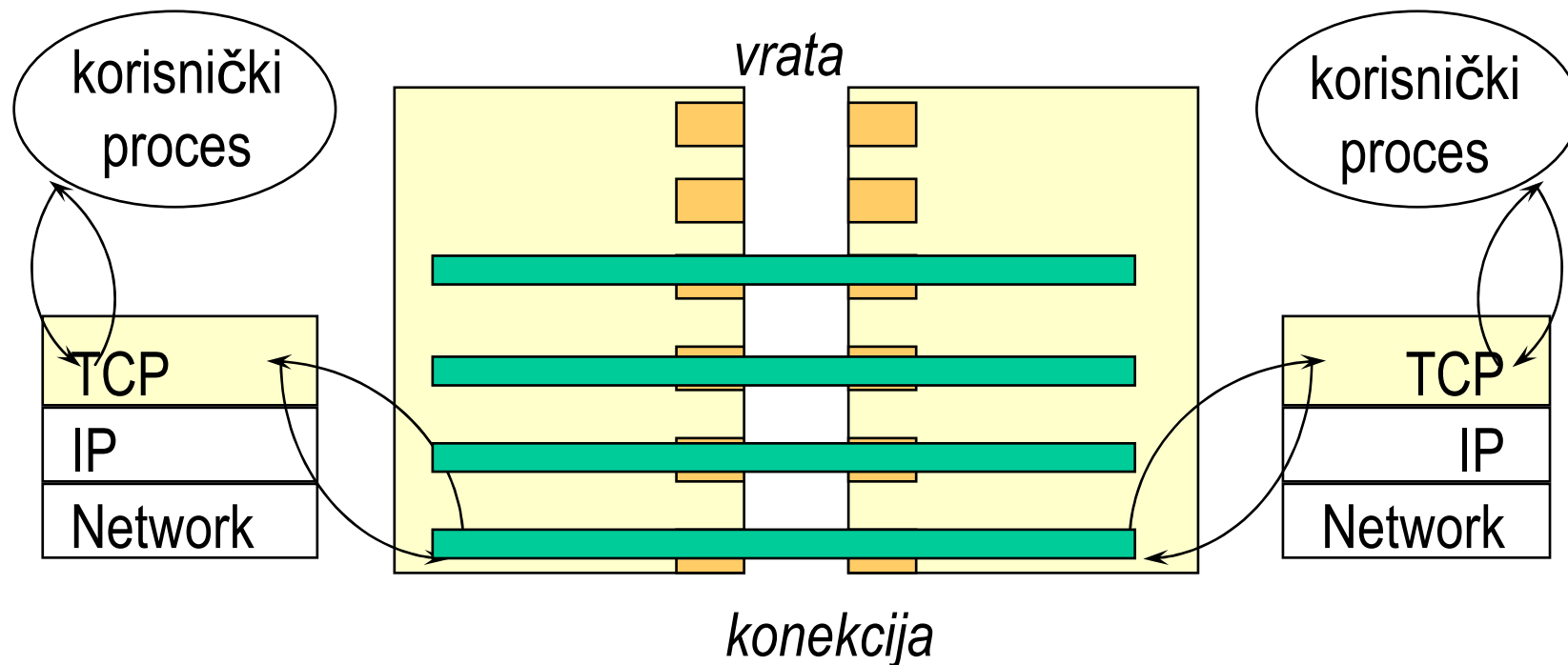
Zavod za telekomunikacije



- ◆ model klijent-poslužitelj
- ◆ vremenska ovisnost procesa
 - poslužitelj mora biti aktivan za primanje datagrama
- ◆ klijent mora znati identifikator poslužitelja
- ◆ tranzijentna komunikacija
- ◆ asinkrona komunikacija
 - klijent šalje datagram i nastavlja procesiranje
- ◆ može se koristiti za implementaciju komunikacije na načelu *pull* i *push*

Transmission Control Protocol (TCP)

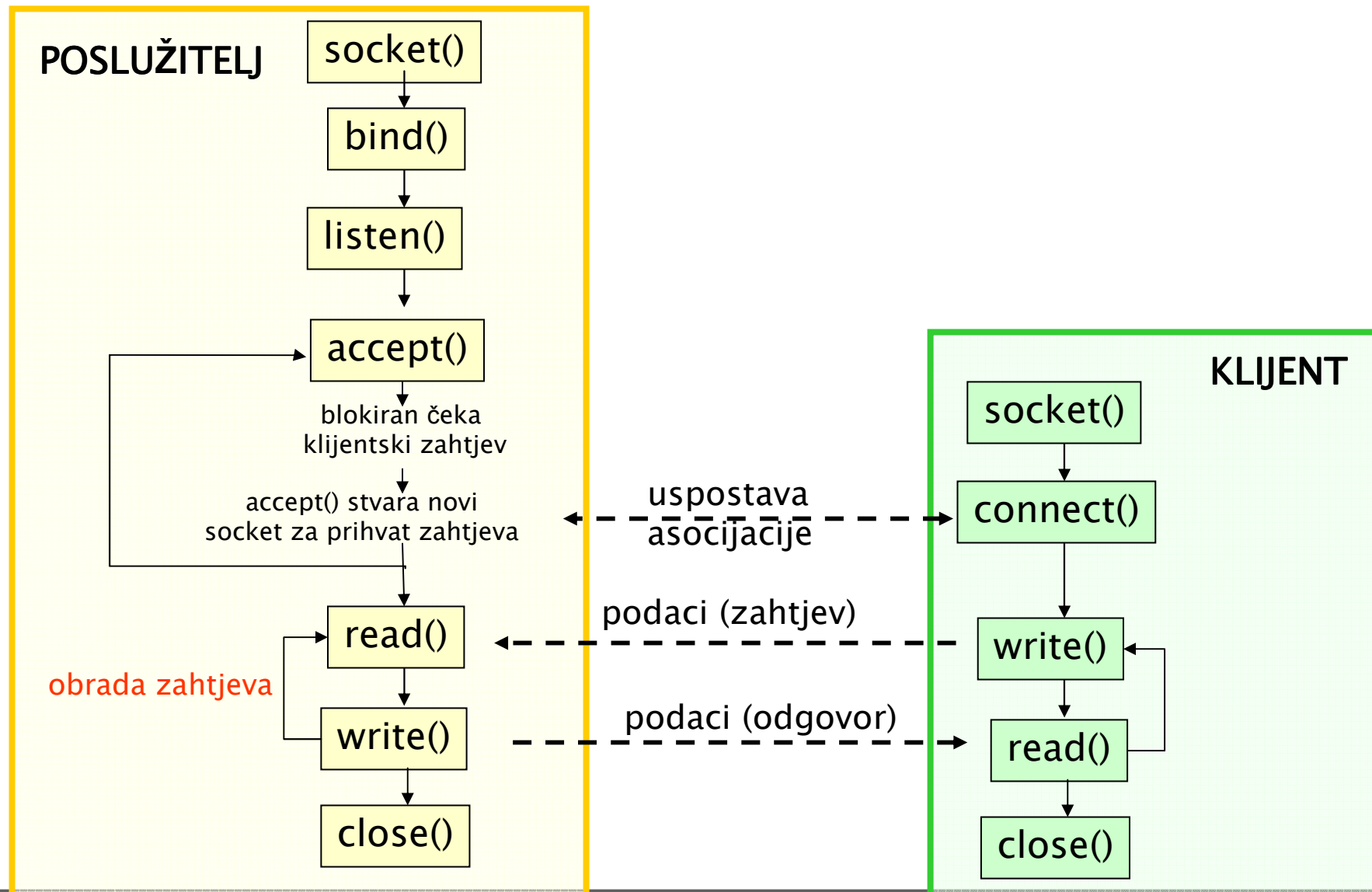
- konekcija između dvije krajnje točke koje se moraju dogovoriti o uspostavi konekcije

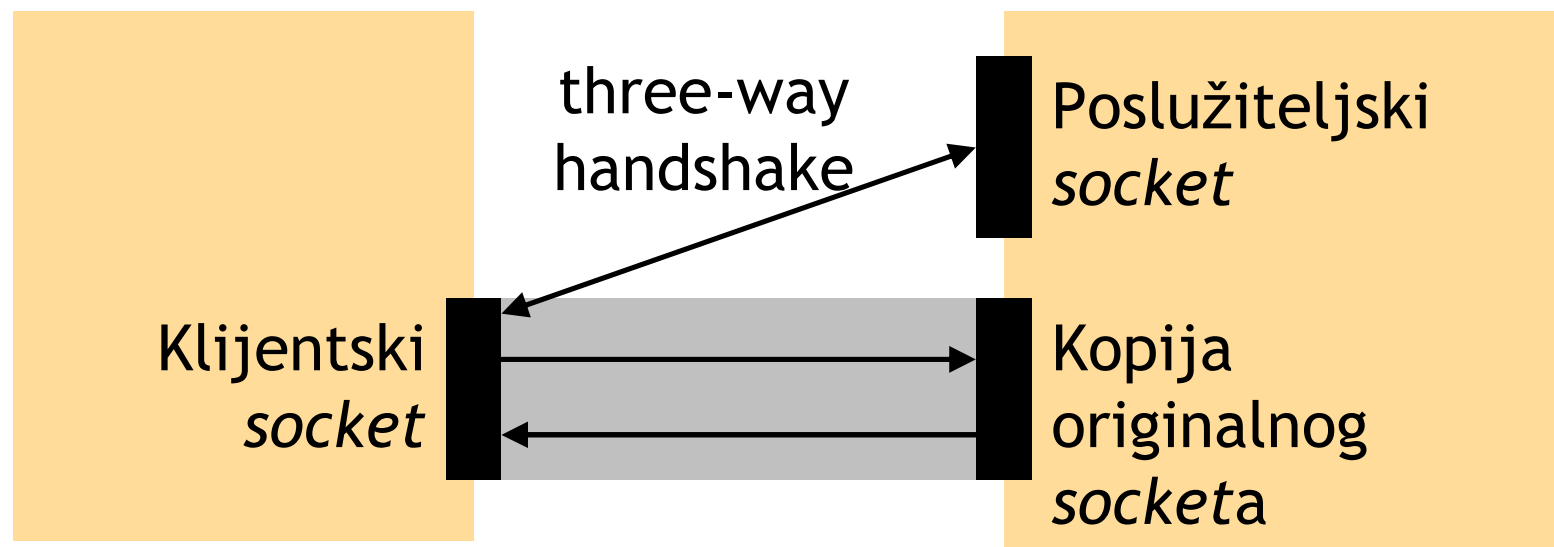


Konekcijska komunikacija pomoću socketa TCP



Zavod za telekomunikacije





- ◆ za svaki novi korisnički zahtjev kreira se novi socket (s novim brojem vrata) koji je kopija originalnog
- ◆ originalni poslužiteljski socket mora konstantno biti u stanju “osluškivanja”

- ◆ model klijent-poslužitelj
- ◆ vremenska ovisnost
 - klijent i poslužitelj moraju biti istovremeno dostupni
- ◆ klijent mora znati identifikator poslužitelja
- ◆ tranzijentna komunikacija
- ◆ sinkrona komunikacija
 - klijent šalje zahtjev za kreiranje konekcije i blokiran je do uspostave konekcije
- ◆ pokretanje komunikacije na načelu *pull*

- ◆ Uvod
 - Interakcijski model
 - Komunikacija
 - Procesi
 - Međuoprema
- ◆ Međuoprema za komunikaciju raspodijeljenih procesa
 - Komunikacija korištenjem priključnica (Socket API)
 - Primjeri TCP/UDP klijenta i poslužitelja
 - Oblikovanje višedretvenog poslužitelja
 - Poziv udaljene procedure (*Remote Procedure Call* - RPC)
 - Poziv udaljene metode (*Remote Method Invocation* - RMI)

<http://java.sun.com/javase/6/docs/api/java/net/package-summary.html>

- ◆ Osnovne klase

- Socket, ServerSocket, URL, URLConnection, (koriste TCP)
- DatagramPacket, DatagramSocket, MulticastSocket (koriste UDP)

- ◆ Java Networking Tutorial

<http://java.sun.com/docs/books/tutorial/networking/>

- ◆ http://www.unix.com.ua/oreilly/java-ent/jnut/ch16_01.htm

Primjer UDP poslužitelja (1)



Zavod za telekomunikacije

```
import java.io.IOException;
import java.net.DatagramSocket;
import java.net.DatagramPacket;

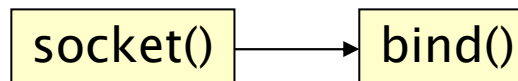
public class UDPServer {

    public static void main(String args[]) throws IOException
    {

        int port = 10001;                //server port
        byte[] rcvBuf = new byte[256]; //received bytes
        byte[] sendBuf = new byte[256]; //sent bytes
        String receivedString;

        //create a UDP socket and bind it to the specified port
        //on the local host
        DatagramSocket socket = new DatagramSocket( port );
```

...



Primjer UDP poslužitelja (2)



Zavod za komunikacije

...

```
while( true ){
    //create a DatagramPacket for receiving packets
    DatagramPacket packet =
        new DatagramPacket(rcvBuf,rcvBuf.length);
    //receive packet
    socket.receive( packet );
    receivedString = new String(packet.getData(
                                packet.getOffset(),
                                packet.getLength() ));

    sendBuf = receivedString.toUpperCase().getBytes();
    //create a DatagramPacket for sending packets
    DatagramPacket sendPacket =
        new DatagramPacket( sendBuf, sendBuf.length,
                            packet.getAddress(),
                            packet.getPort() );
    //send packet
    socket.send( sendPacket );
}
```

recvfrom()

obrada
zahtjeva

sendto()

Primjer UDP klijenta (1)



Zavod za telekomunikacije

```
import java.io.IOException;
import java.net.InetAddress;
import java.net.DatagramSocket;
import java.net.DatagramPacket;

public class UDPClient {

    public static void main(String args[]) throws IOException
    {

        String sendString = new String("Any string...");

        byte[] rcvBuf = new byte[256];
        byte[] sendBuf = new byte[256];
        sendBuf = sendString.getBytes();

        //Determines the IP address of a host, given the host's
        name
        InetAddress address = InetAddress.getByName("localhost");

        ...
    }
}
```

Primjer UDP klijenta (2)



Zavod za telekomunikacije

...

```
//Create a datagram socket and bind it to any available  
//port on the local host
```

```
DatagramSocket socket = new DatagramSocket(); socket()
```

```
//Create a datagram packet for sending data
```

```
DatagramPacket packet =  
    new DatagramPacket(sendBuf, sendBuf.length,  
address,10001);
```

```
socket.send( packet ); sendto()
```

```
//Create a datagram packet for receiving data
```

```
DatagramPacket rcvPacket =  
    new DatagramPacket(rcvBuf, rcvBuf.length);
```

```
socket.receive(rcvPacket); recvfrom()
```

```
String rcvString = new String( rcvPacket.getData(),  
                                rcvPacket.getOffset(),  
                                rcvPacket.getLength() );
```

```
socket.close(); close()
```

```
}}
```


TCP: implementacija poslužitelja



Zavod za telekomunikacije

1. Kreirati socket poslužitelja:

```
ServerSocket server;  
server = new ServerSocket( PORT );
```

2. Čekati korisnički zahtjev (blokira proces do klijentskog zahtjeva!!!):

```
Socket copySocket = server.accept();
```

3. Kreirati I/O stream za komunikaciju s klijentom

```
DataInputStream is = new DataInputStream(  
    client.getInputStream() );  
DataOutputStream os = new DataOutputStream(  
    client.getOutputStream() );
```

4. Komunikacija s klijentom

```
Receive from client: String line = is.readLine();  
Send to client: os.writeBytes("Hello\n");
```

5. Zatvoriti socket:

```
client.close();
```

1. Kreirati socket:

```
client = new Socket( server, port_id );
```

2. Kreirati I/O stream za komunikaciju s poslužiteljem:

```
is = new DataInputStream(client.getInputStream() );  
os = new DataOutputStream( client.getOutputStream() );
```

3. Komunikacija s poslužiteljem:

- //Receive data from the server:

```
String line = is.readLine();
```

- //Send data to the server:

```
os.writeBytes("Hello\n");
```

4. Zatvoriti socket:

```
client.close();
```

Primjer TCP poslužitelja (1)



Zavod za telekomunikacije

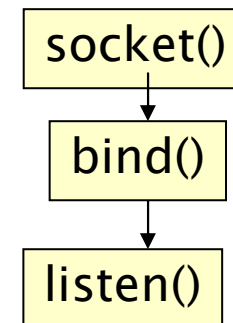
```
import java.io.PrintWriter;
import java.io.BufferedReader;
import java.io.InputStreamReader;
import java.io.IOException;
import java.net.Socket;
import java.net.ServerSocket;

public class TCPServer {
    public static void main(String args[]) throws IOException,
        UnknownHostException {

        int port = 10002;
        ServerSocket serverSocket = new ServerSocket(port);

        String rcvStr = null;
        String sendStr = null;

        ...
    }
}
```



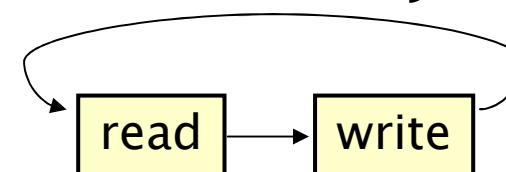
Primjer TCP poslužitelja (2)



Zavod za telekomunikacije

...

```
while( true ){  
    Socket socket = serverSocket.accept();  
    PrintWriter outToClient =  
        new PrintWriter(socket.getOutputStream(), true);  
    BufferedReader inFromClient = new BufferedReader(new  
        InputStreamReader(socket.getInputStream()));  
    while( (rcvStr = inFromClient.readLine()) != null ){  
        System.out.println( "Server received " + rcvStr );  
        if( rcvStr.equals( "\n" ) )  
            break;  
        outToClient.println(rcvStr.toUpperCase());  
        System.out.println( "Server sends:\t" + sendStr );  
    }  
    outToClient.close();  
    inFromClient.close();  
    socket.close();  
}
```



Primjer TCP klijenta (1)



Zavod za telekomunikacije

```
import java.io.IOException;
import java.io.PrintWriter;
import java.io.BufferedReader;
import java.io.InputStreamReader;
import java.net.Socket;

public class TCPClient {
    public static void main(String args[]) throws
        IOException, UnknownHostException {

        String serverName = new String("localhost");
        int port = 10002;

        //create the socket connection
        Socket clientSocket = new Socket( serverName,
port );
        String sendString = new String("Any string...");
        ...
    }
}
```

socket()



connect()

Primjer TCP klijenta (2)



Zavod za telekomunikacije

...

```
//get the socket's output stream and open a PrintWriter on it
```

```
PrintWriter outToServer =  
    new PrintWriter( clientSocket.getOutputStream(),  
true);
```

```
//get the socket's input stream and open a BufferedReader on it
```

```
BufferedReader inFromServer = new BufferedReader(  
    new InputStreamReader  
(clientSocket.getInputStream()));
```

```
outToServer.println(sendString);
```

write()

```
String rcvString = inFromServer.readLine();
```

read()

```
System.out.println("FROM SERVER:" + rcvString);
```

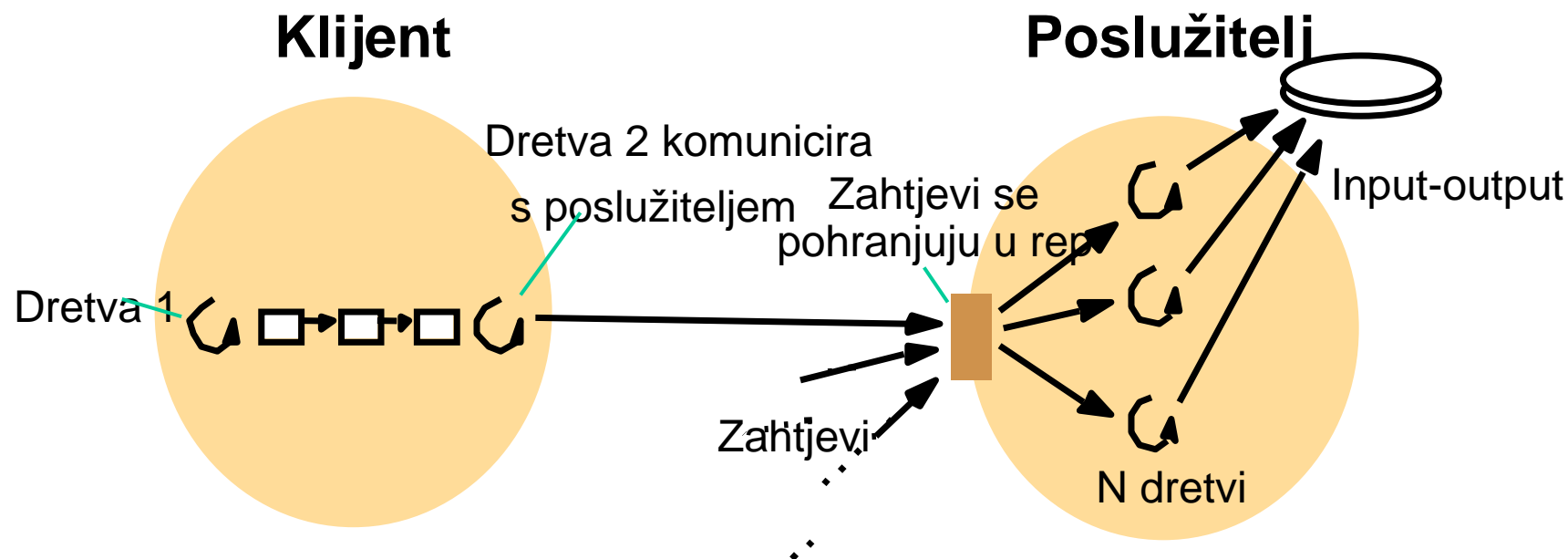
```
outToServer.println("\n");
```

```
clientSocket.close();
```

close()

```
}}
```

- ◆ Uvod
 - Interakcijski model
 - Komunikacija
 - Procesi
 - Međuoprema
- ◆ Međuoprema za komunikaciju raspodijeljenih procesa
 - Komunikacija korištenjem priključnica (Socket API)
 - Primjeri TCP/UDP klijenta i poslužitelja
 - Oblikovanje višedretvenog poslužitelja
 - Poziv udaljene procedure (*Remote Procedure Call* - RPC)
 - Poziv udaljene metode (*Remote Method Invocation* - RMI)



Uobičajene zadaće na strani klijenta:

- korisničko sučelje,
- složeno procesiranje,
- otvaranje mrežne konekcije i primanje podataka

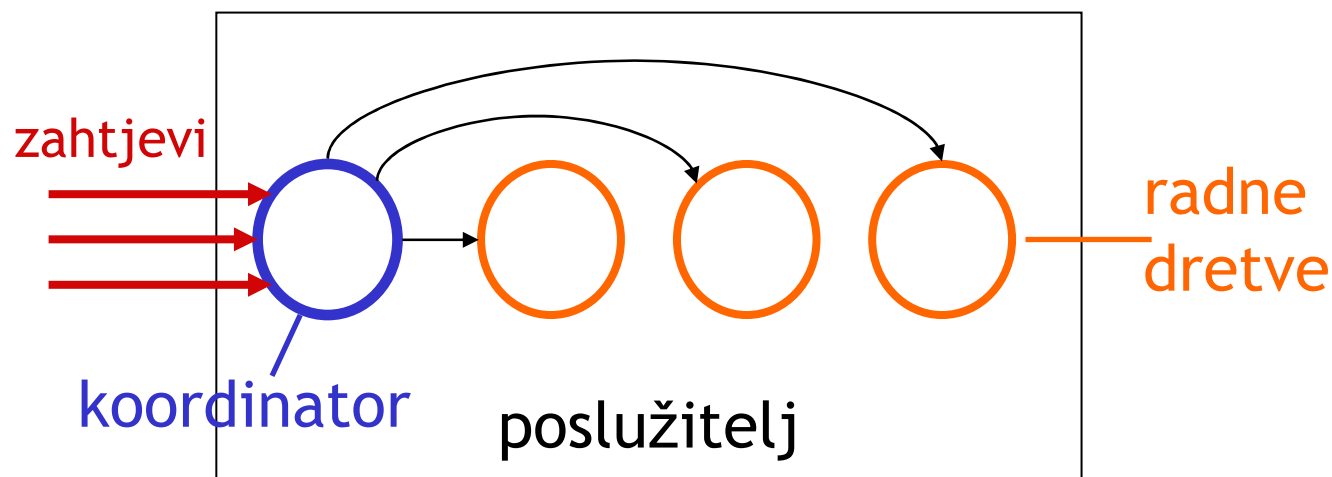
Uobičajene zadaće na strani poslužitelja:

- konkurentni klijentski pozivi
- rad s diskom

Višedretveni poslužitelj (1)



Zavod za telekomunikacije

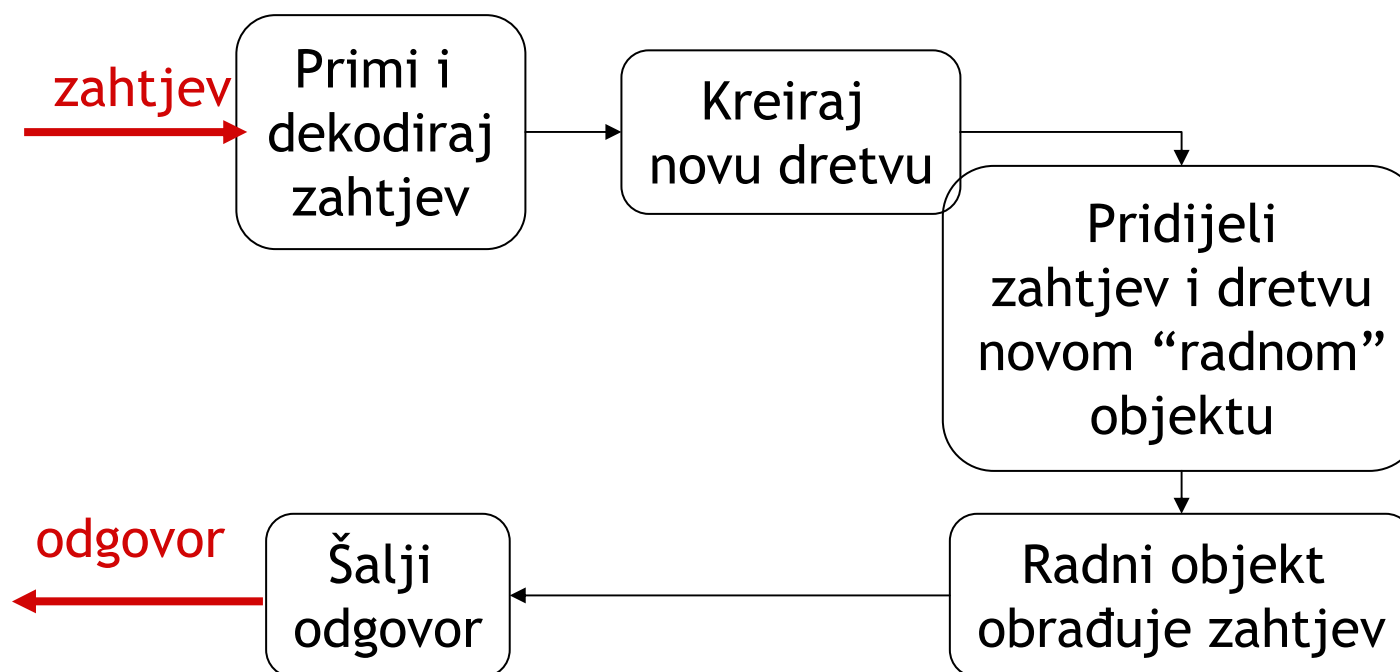


Model koordinator/radna dretva
(*dispatcher/worker model*)

Višedretveni poslužitelj (2)



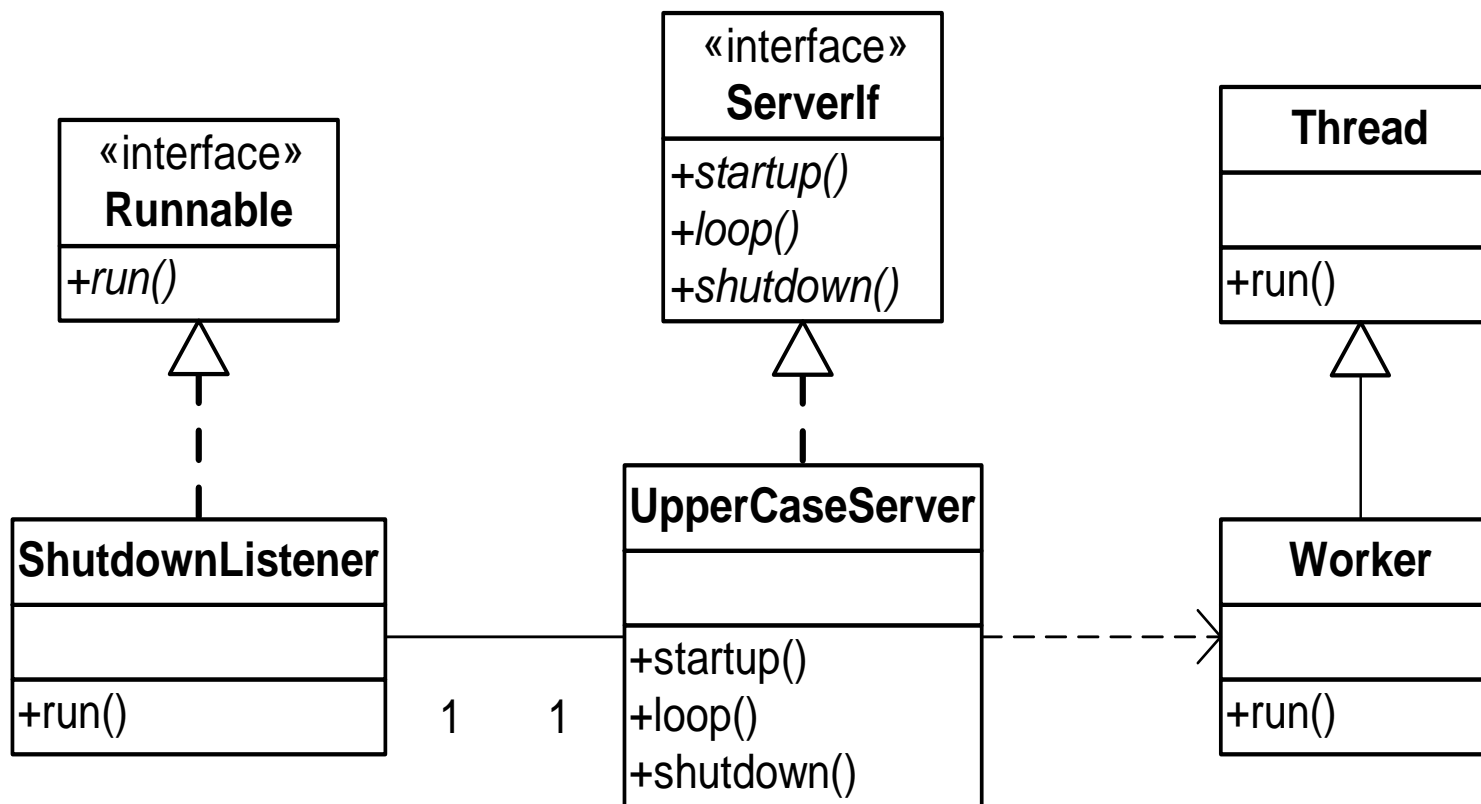
Zavod za telekomunikacije



Primjer višedretvenog poslužitelja



Zavod za telekomunikacije



```
public interface ServerIf {  
    // Server startup. Starts all services offered by the  
    server.  
    public void startup();  
  
    // Server loops when in running mode. The server must be  
    active  
    // to accept client requests.  
    public void loop();  
  
    // Server shutdown. Shuts down all services started during  
    //startup.  
    public void shutdown();  
  
    // Gets the running flag that indicates server running  
    status.  
    // @return running flag  
    public boolean getRunningFlag();  
  
    // Sets the running flag that indicates server running  
    status.  
    // @param flag running flag  
    public void setRunningFlag(boolean flag);
```

```
public class UpperCaseServer implements ServerIf {  
  
    private String      hostname = null;  
  
    private int         port;  
  
    private ServerSocket serverSocket;  
  
    private boolean runningFlag = false;  
  
    //Reference to a shutdown listener process, it accepts  
    shutdown  
    //requests.  
    private ShutdownListener shutdownListener = null;  
  
    //The number of active TCP connections.  
    private int activeConnections = 0;  
  
    private final static int MAX_CLIENTS = 10;  
  
    ...  
}
```

```
...
public UpperCaseServer( int portNo ) {

    try {
        //determine the local hostname
        hostName = InetAddress.getLocalHost().getHostName();
    } catch (UnknownHostException e) {
        e.printStackTrace();
        System.exit(-1);
    }

    port = portNo;
    if( port < 1024 ) {
        System.err.println( "UpperCaseServer: illegal port
number [" + port + "]" );
        System.exit(-1);
    }
}
...

```

...

```
//Starts all required server services.
public void startup() {
    try {
        //create server socket, bind it to the specified port
        //and set the max queue length for client requests
        serverSocket = new ServerSocket(port, MAX_CLIENTS);

        //set the running flag to true
        runningFlag = true;

        //create shutdown listener in a new thread
        shutdownListener = new ShutdownListener(this, 4456);
        new Thread( shutdownListener ).start();

    } catch( IOException e) {
        e.printStackTrace();
        System.exit(-1);
    }
}...
```

```
//The main loop for accepting client requests.
public void loop() {
    try {    //set timeout to avoid blocking
        serverSocket.setSoTimeout( 500 );
    } catch( SocketException e ){
        e.printStackTrace();
        System.exit(-1);
    }
    while( runningFlag ) {
        try {
            //accept requests and create a new Worker object
            //in a new thread, or throw a SocketTimeoutException
            Thread w = (Thread)new Worker(serverSocket.accept());
            w.start();
            activeConnections++;
        } catch( SocketTimeoutException ste ) {
            //do nothing, check the runningFlag
        } catch( IOException e ) {
            e.printStackTrace();
            System.exit(-1);
        }
    }
}
```


...

```
public void shutdown() {
    while( activeConnections > 0 ) {
        System.out.println( "WARNING: There are still active
            connections" );
        try {
            Thread.sleep( 5000 );
        } catch( java.lang.InterruptedException e ){}
    }
    if( activeConnections == 0 ) {
        System.out.println( "Server shutdown." );
        if( serverSocket != null )
            try {
                serverSocket.close();
            } catch( IOException e ) {
                e.printStackTrace();
                System.exit(-1);
            }
    }
}...
```

...

```
//The main method creates a new server object, initiates  
server
```

```
//startup, calls the loop method,
```

```
//and when the loop terminates invokes the shutdown  
method.
```

```
public static void main(String argv[]) {
```

```
    UpperCaseServer server = new UpperCaseServer();
```

```
    server.startup();
```

```
    server.loop();
```

```
    server.shutdown();
```

```
    System.exit(0);
```

```
}
```

Worker (1)



```
//inner class within UpperCaseServer
private class Worker extends Thread {
    Socket socket = null;
    Worker( Socket sckt ){
        super( "UpperCaseServer.Worker" );
        socket = sckt;
    }
    //Creates reader and writer for the socket, reads a line
    from
    //the reader, transforms it to upper case, and sends the
    //transformed string to client.
    public void run() {
        String clientInput = null;
        String capitalizedInput = null;
        try {
            PrintWriter outToClient =
                new PrintWriter(socket.getOutputStream(), true);
            BufferedReader inFromClient =
                new BufferedReader(new InputStreamReader
                    (socket.getInputStream()));
```

Worker (2)



Zavod za telekomunikacije

```
while( (clientInput = inFromClient.readLine()) !=
null ){
    System.out.println( "Server received from " +
        socket.getRemoteSocketAddress().toString() +
        ":\t"
        + clientInput );
    if( clientInput.equals( "\n" ))
        break;
    capitalizedInput = clientInput.toUpperCase();
    outToClient.println(capitalizedInput);
    System.out.println( "Server sends:\t" +
        capitalizedInput );
}
outToClient.close();
inFromClient.close();
if( socket != null )
    socket.close();
activeConnections--;
} catch( IOException e ) {
    e.printStackTrace();
}
}
```

ShutdownListener (1)



Zavod za telekomunikacije

```
//Creates a datagram socket that listens on port 4456 for
// shutdown requests.
public class ShutdownListener implements Runnable {
    //Reference to the server that will be shutdown.
    private ServerIf      server = null;
    private DatagramSocket socket = null;
    int                    port;
    //Running flag: when true the datagram socket is running and
    //accepting packets when false the listener process terminates.
    boolean                runningFlag = false;

    public ShutdownListener( ServerIf srv, int portNo ) {
        server = srv;
        port = portNo;
        if ( port > 1024 )
            try {
                socket = new DatagramSocket( port );
                runningFlag = true;
            } catch( IOException e ) {
                e.printStackTrace();
                System.exit(-1);
            }
    }
}
```

ShutdownListener (2)



Zavod za telekomunikacije

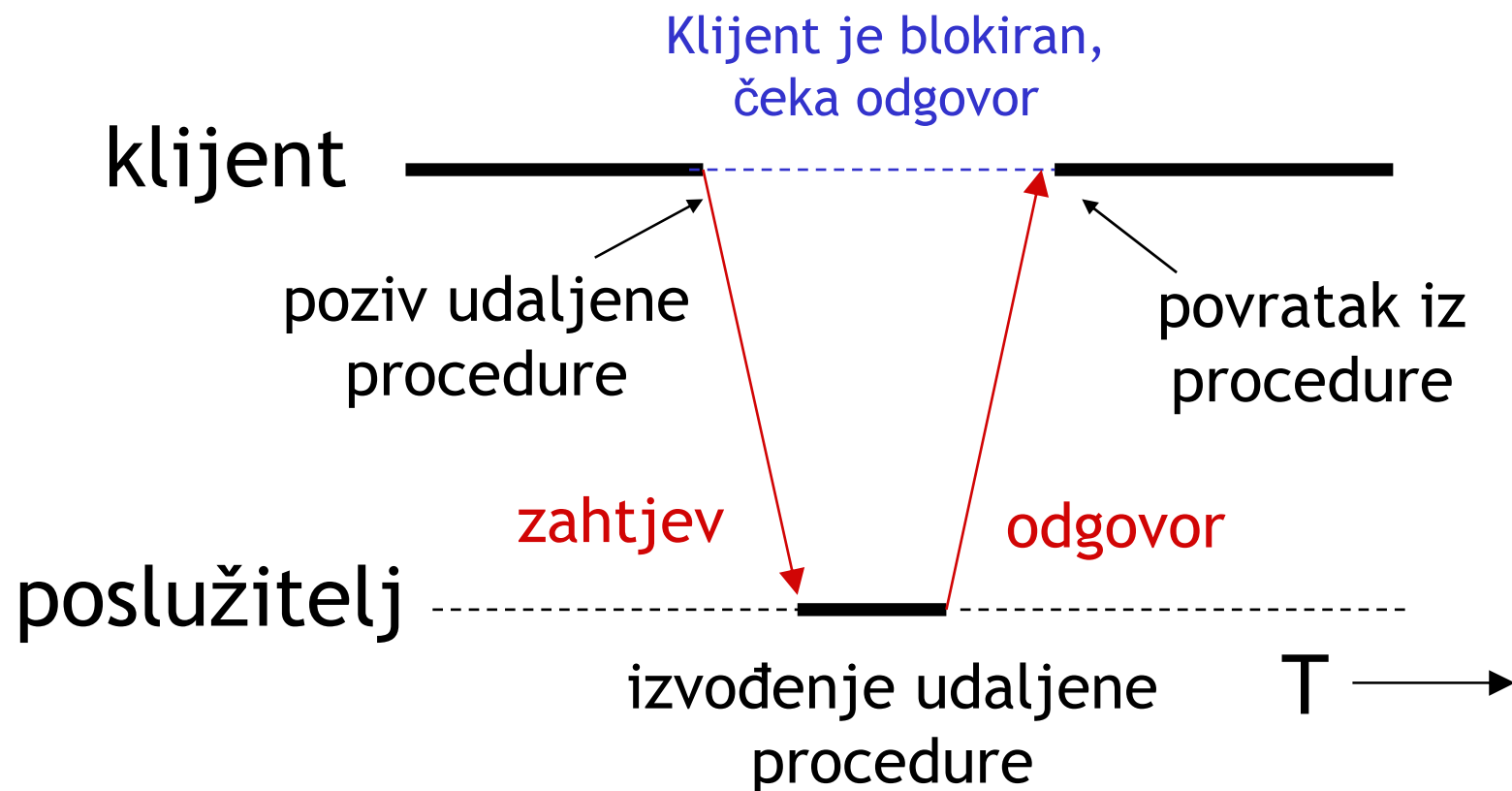
```
//Listens for datagram packets. When the packet contains the
string
//"Shutdown now" sets the runningFlag of the server to false.
public void run() {
    byte[] buf = new byte[256];
    String received;
    while( runningFlag ){
        DatagramPacket packet = new DatagramPacket(buf,buf.length);
        try {
            socket.receive( packet );
            received = new String( packet.getData(),
                                   packet.getOffset(), packet.getLength());
            if( received.equals( "Shutdown now" ) ) {
                server.setRunningFlag( false );
                //set my running flag to false
                runningFlag = false; }
        } catch( IOException e ) {
            e.printStackTrace();
        }
    }
    socket.close();
}
```

- ◆ Uvod
 - Interakcijski model
 - Komunikacija
 - Procesi
 - Međuoprema
- ◆ Međuoprema za komunikaciju raspodijeljenih procesa
 - Komunikacija korištenjem priključnica (Socket API)
 - Primjeri TCP/UDP klijenta i poslužitelja
 - Oblikovanje višedretvenog poslužitelja
 - Poziv udaljene procedure (*Remote Procedure Call - RPC*)
 - Poziv udaljene metode (*Remote Method Invocation - RMI*)

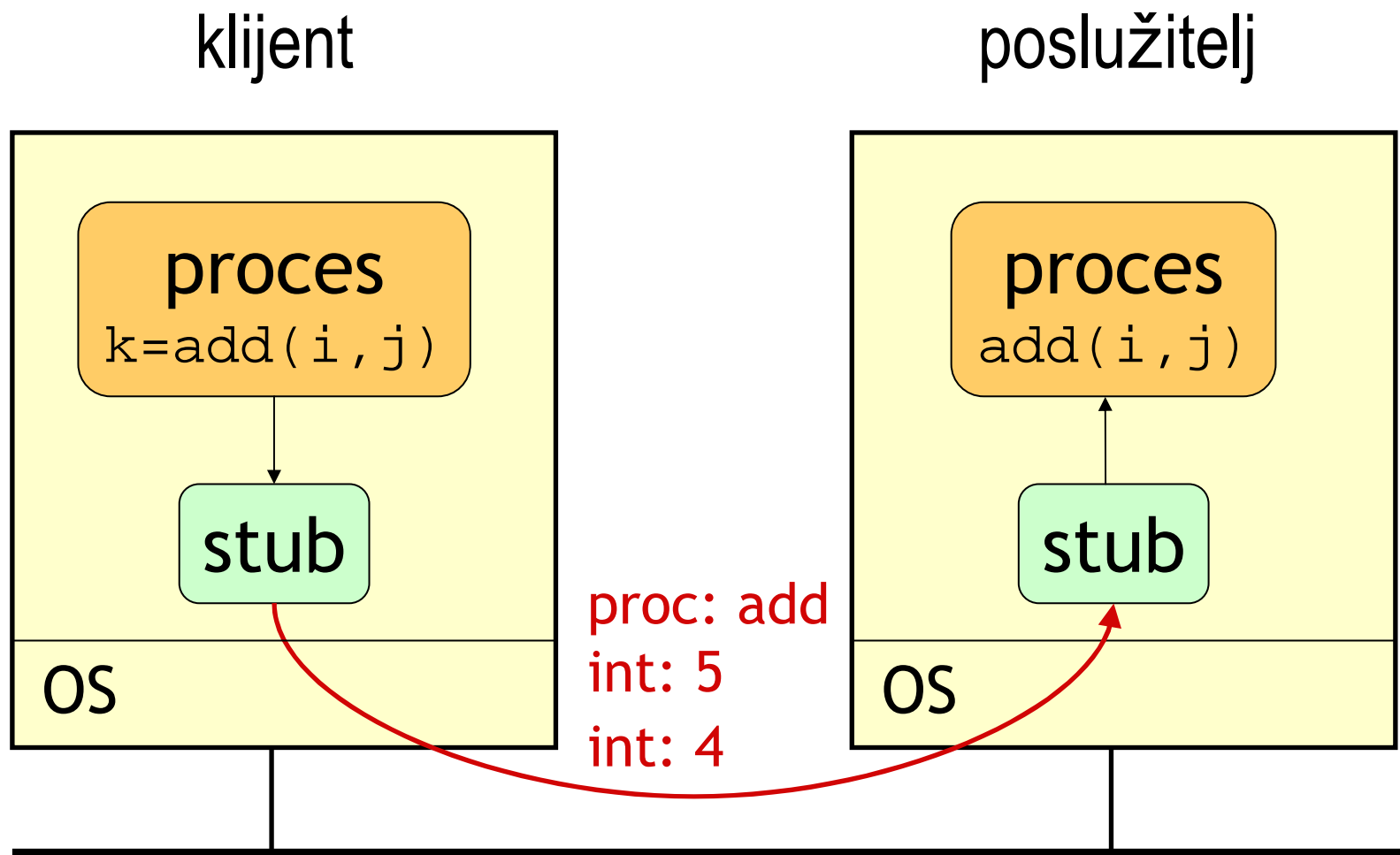
Poziv udaljene procedure (RPC)



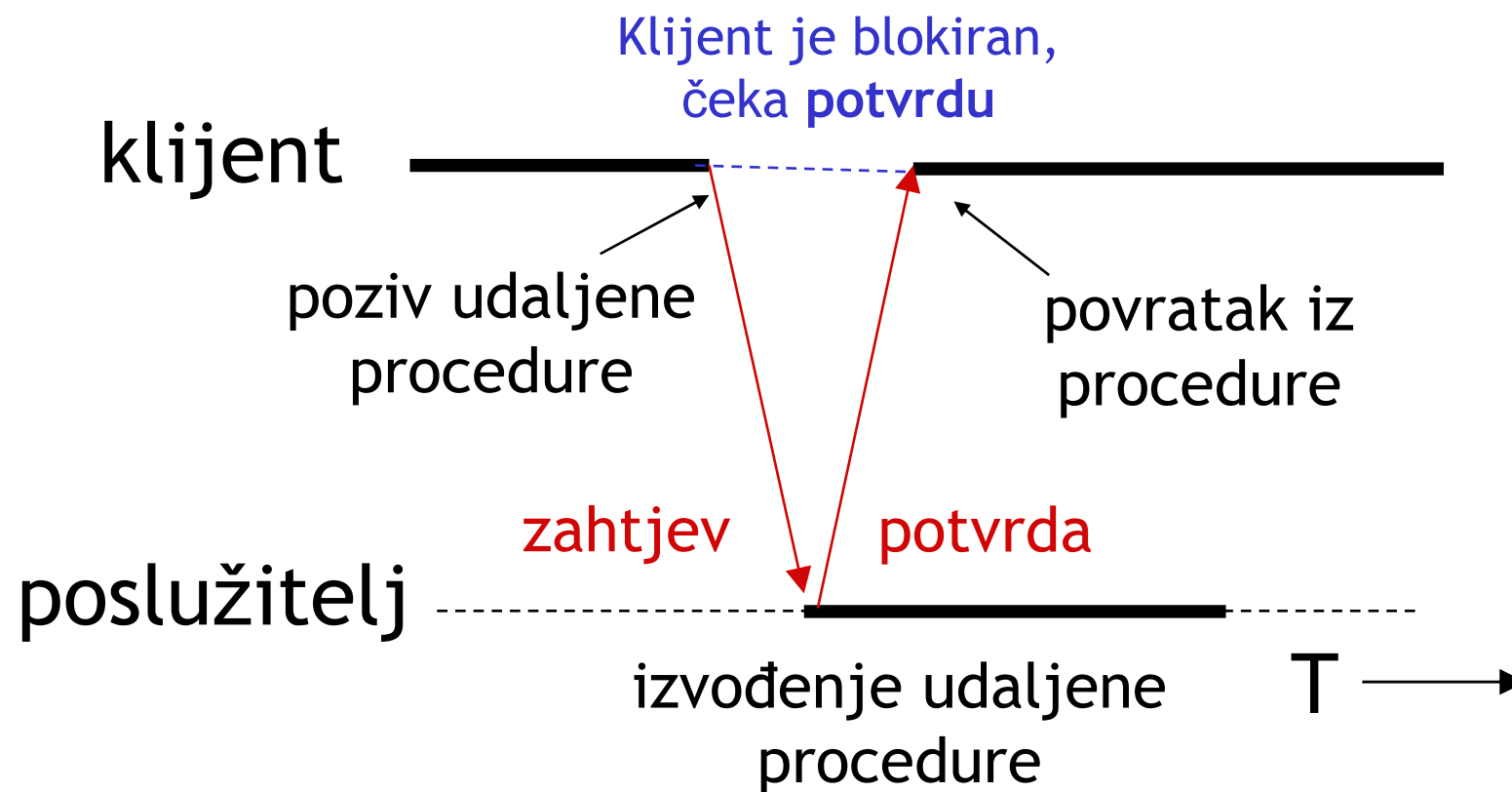
Zavod za telekomunikacije



Omogućiti procesima pozivanje i izvođenje procedura na udaljenom računalu.



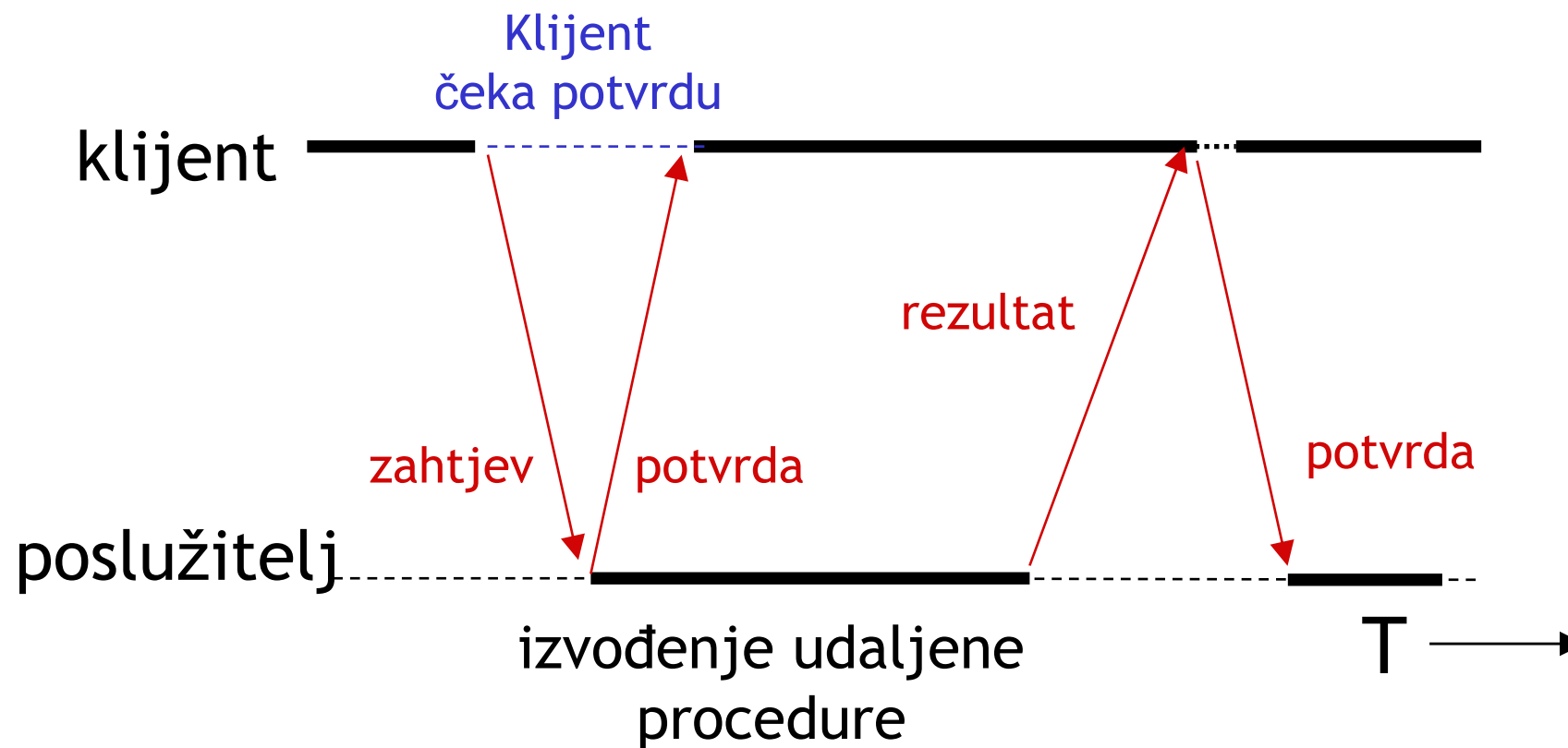
- ◆ *Marshaling* – “pakiranje” parametara ili rezultata u poruku
- ◆ *Unmarshaling* – čitanje parametara ili rezultata iz poruke
- ◆ Prenošenje vrijednosti parametra
 - Navodi se tip (npr. int, char, long) i vrijednost
 - Različita računala često koriste različite prikaze znakova
- ◆ Prenošenje parametara koristeći reference
 - Referenca ima smisla samo u adresnom prostoru procesa koji je koristi!
 - Kako prenijeti string na udaljeno računalo?
 - nije moguće koristiti referencu na string!
 - kopiranje cijelog stringa i “pakiranje” u poruku



Odgođeni sinkroni RPC



Zavod za telekomunikacije

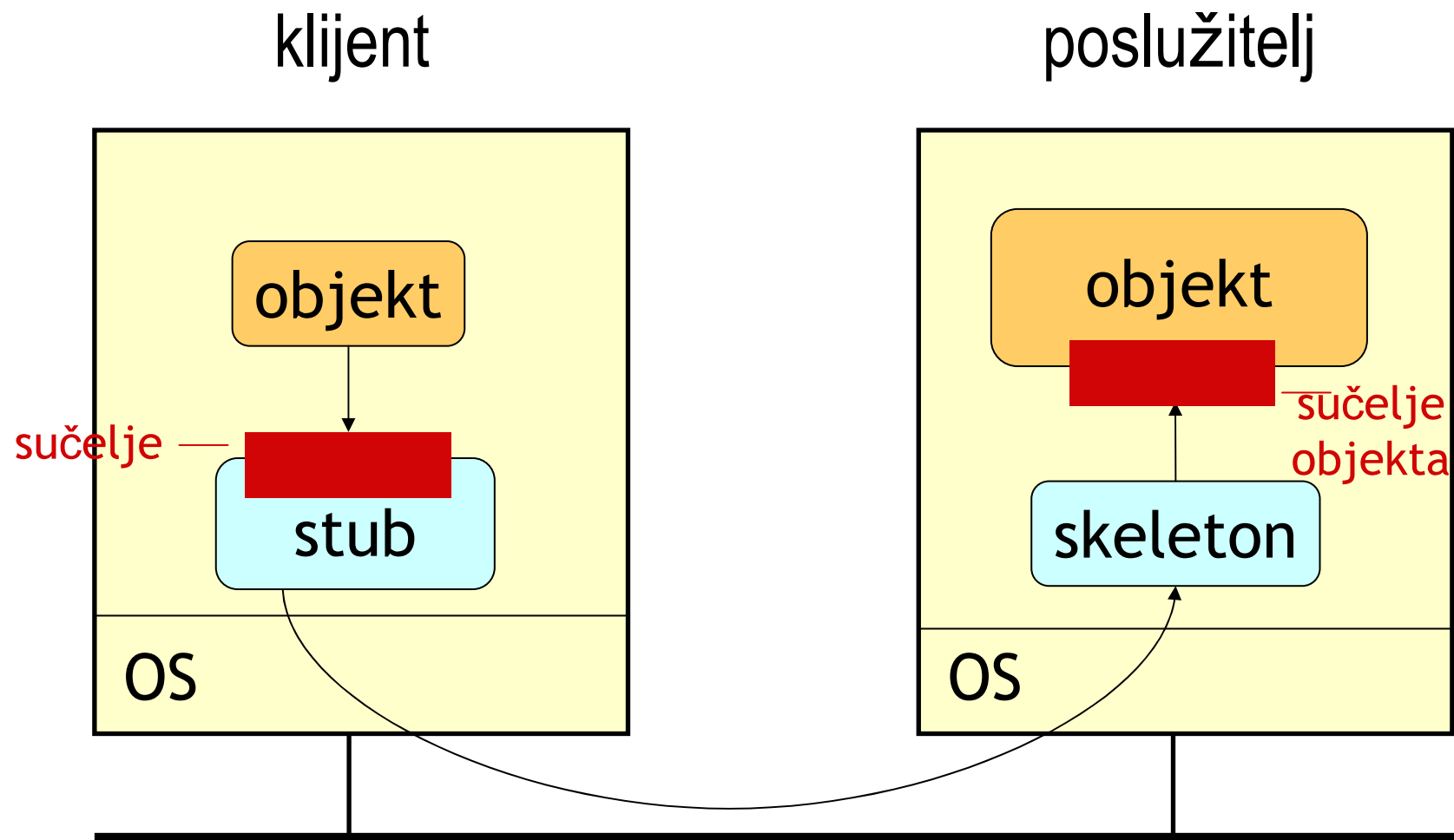


- ◆ Uvod
 - Interakcijski model
 - Komunikacija
 - Procesi
 - Međuprema
- ◆ Međuprema za komunikaciju raspodijeljenih procesa
 - Komunikacija korištenjem priključnica (Socket API)
 - Primjeri TCP/UDP klijenta i poslužitelja
 - Oblikovanje višedretvenog poslužitelja
 - Poziv udaljene procedure (*Remote Procedure Call* - RPC)
 - Poziv udaljene metode (*Remote Method Invocation* - RMI)

Remote Method Invocation (RMI)

- ◆ “nasljednik” poziva udaljene procedure, poziva se metoda udaljenog objekta
- ◆ distribuirani objekt
 - proširenje osnovnog objektnog modela na distribuirane objekte
 - odvajanje sučelja i implementacije objekta
- ◆ objekt (klijent) poziva metodu udaljenog objekta (poslužitelja) na transparentan način
 - identično pozivu metode lokalnog objekta

- ◆ Postoje reference na lokalne i udaljene objekte
- ◆ Svaki udaljeni objekt ima globalno jedinstven identifikator
 - npr. [ref: [endpoint:[161.53.19.24:1251](local),objID:[0]]]
- ◆ Potrebna je usluga za registriranje i pronalaženje udaljenih objekata (*directory service*)



- ◆ model klijent-poslužitelj
- ◆ vremenska ovisnost klijenta i poslužitelja
- ◆ klijent mora znati identifikator poslužitelja
- ◆ tranzijentna komunikacija
- ◆ sinkrona komunikacija
 - klijent je blokiran dok ne primi odgovor od strane poslužitelja
- ◆ pokretanje komunikacije na načelu *pull*

- ◆ A. S. Tanenbaum, M. Van Steen: Distributed Systems: Principles and Paradigms, Second Edition, Prentice Hall, 2007
poglavljja 2 i 3
- ◆ G. Coulouris, J. Dollimore, T. Kindberg: Distributed Systems: Concepts and Design, 4th edition, Addison-Wesley, 2005
poglavljja 4 i 5