

# BLOC II. Comunicarea între procesele unui sistem distribuit

Sisteme și servicii distribuite

Inginerie Telematică de gradul III

# Cuprins

## 1. Introducere

## 2.Prize

1. Caracteristicile comunicării între procese
2. Arhitectura TCP/IP
  
3. Definirea prizelor
  
- 4.Prize Java

## 3. Comunicarea de grup

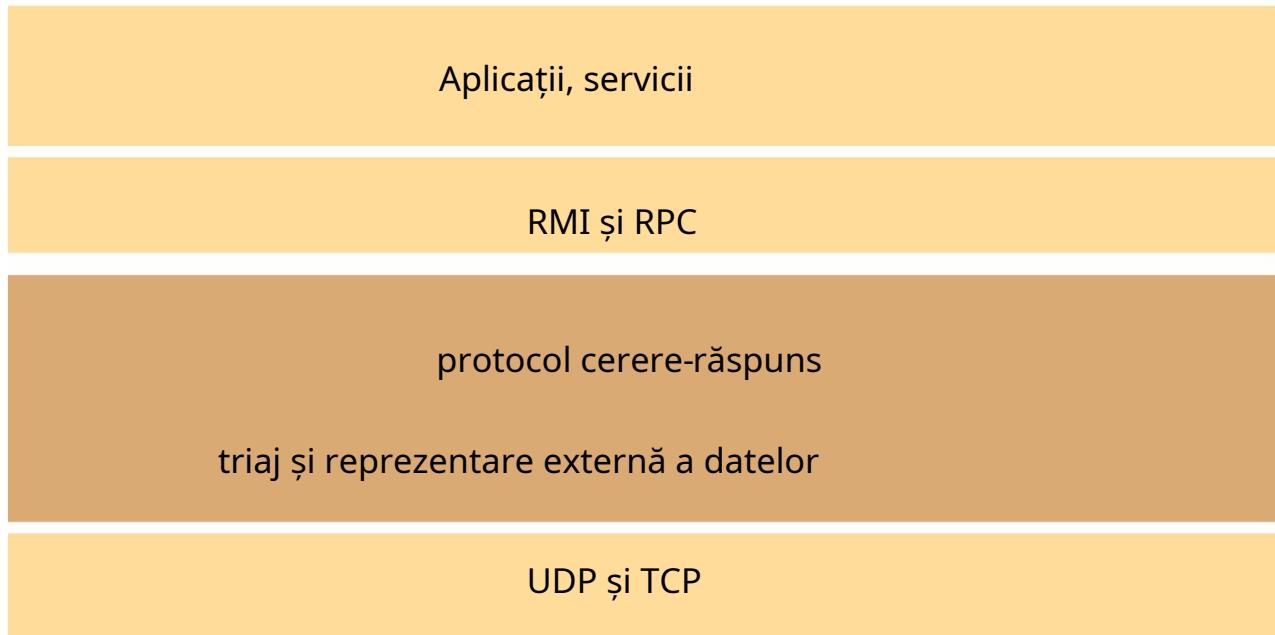
1. Tipuri de grupuri
2. Caracteristici ale comunicării de grup

## 4. Invocarea metodei de la distanță (RMI)

1. Protocole cerere-răspuns
2. RMI Design
  
3. Implementarea RMI
  
- 4.JavaRMI

# Introducere

acest  
capitol



# Cuprins

## 1. Introducere

## 2. Priză

1. Caracteristicile comunicării între procese 2. Arhitectura TCP/IP

3. Definirea prizelor

4. Priză Java

## 3. Comunicarea de grup

1. Tipuri de grupuri 2.

Caracteristici ale comunicării de grup

## 4. Invocarea metodei de la distanță (RMI)

1. Protocole cerere-răspuns

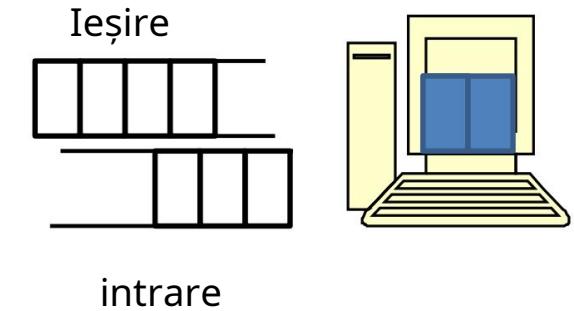
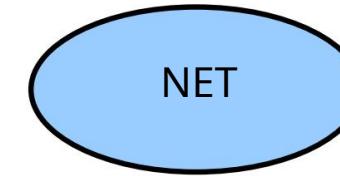
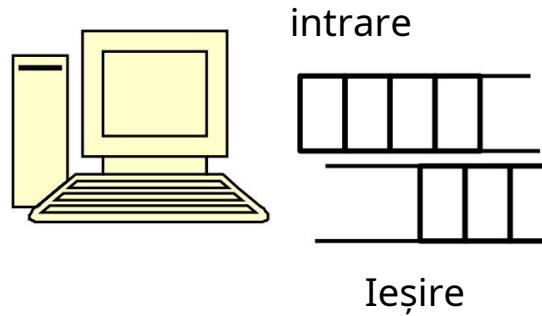
2. RMI Design

3. Implementarea RMI

4. JavaRMI

# Characteristicile comunicării între proceselor

- Trimiterea de mesaje între oricare două procese:
  - Trimite()
  - A primi()



# Caracteristicile comunicării între proceselor

## Comunicarea între procese

### SINCRONĂ

- Procesele expeditorului și receptorului sunt sincronizate în fiecare mesaj
- Blocare Trimitere() și primire().

### ASINCRON

- Send() NU blochează: expeditor poate continua să efectueze sarcini după copierea mesajului în memoria tampon local
- Receive() poate fi blocant sau neblocant (este necesare întreruperi)

# Cuprins

## 1. Introducere

### 2.Prize

1. Caracteristicile comunicării între procese 2. Arhitectura TCP/IP

3. Definirea prizelor

4.Prize Java

## 3. Comunicarea de grup

1. Tipuri de grupuri 2.

Caracteristici ale comunicării de grup

## 4. Invocarea metodei de la distanță (RMI)

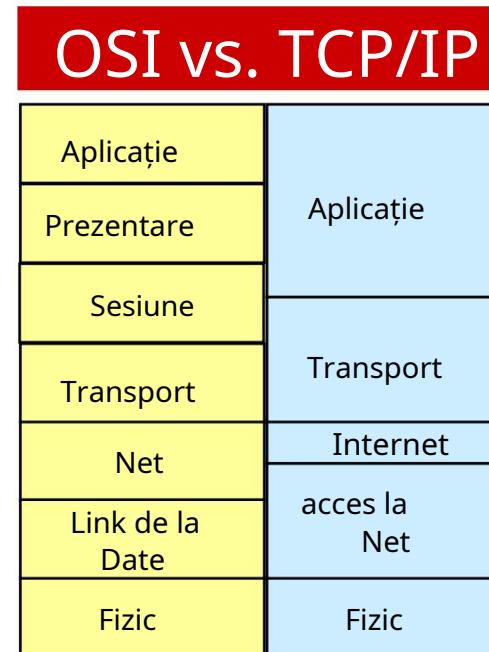
1. Protocole cerere-răspuns

2. RMI Design

3. Implementarea RMI

4.JavaRMI

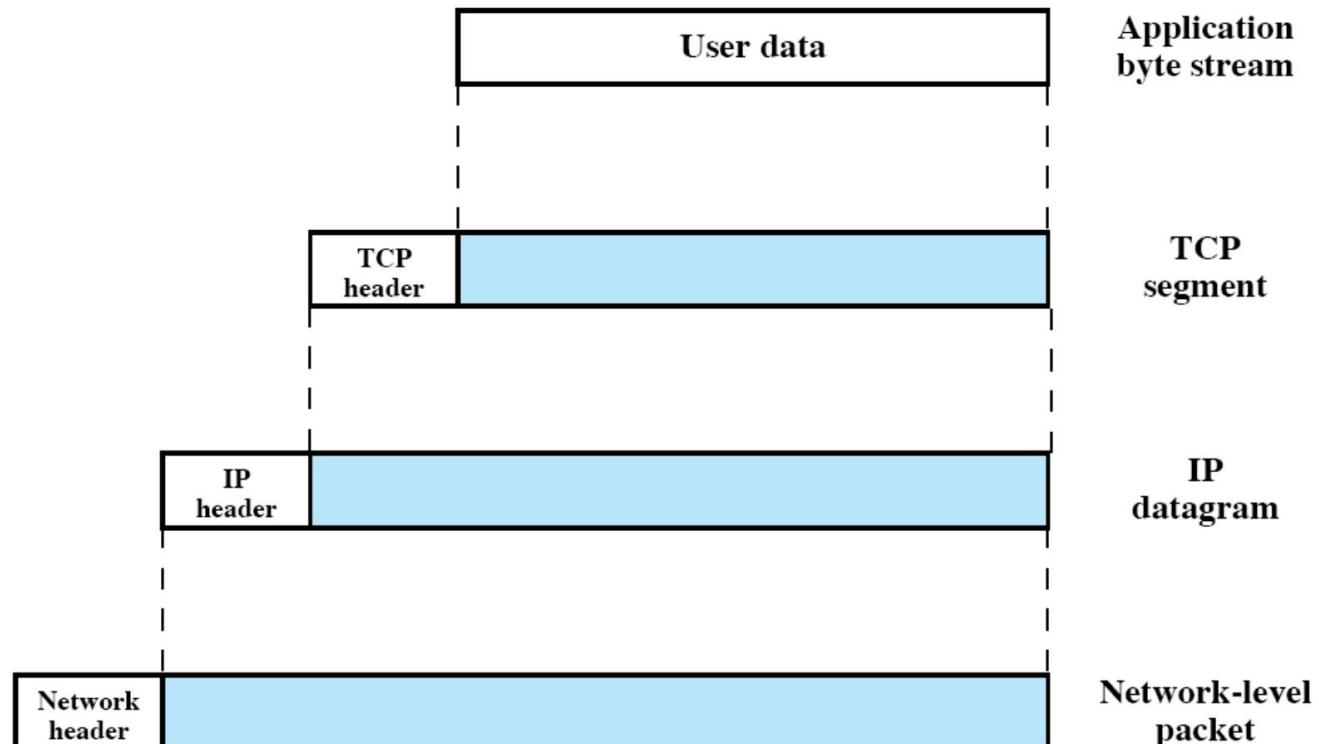
# Arhitectura TCP/IP



# Arhitectura TCP/IP

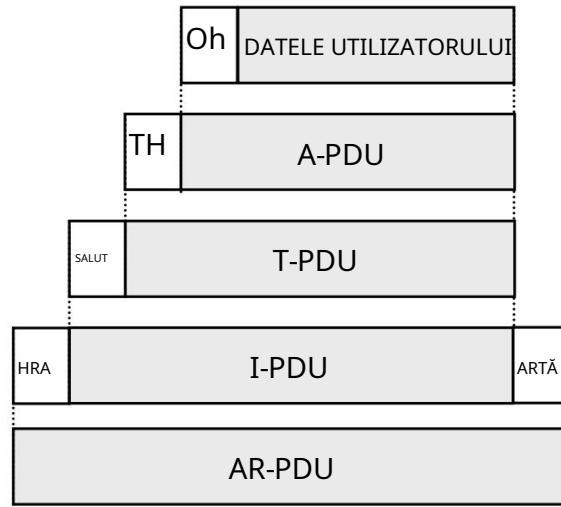
- Protocol Data Units (PDU) în arhitectura TCP/IP

- Application Data
- Segmente
- Pachete
- parcele
- biți

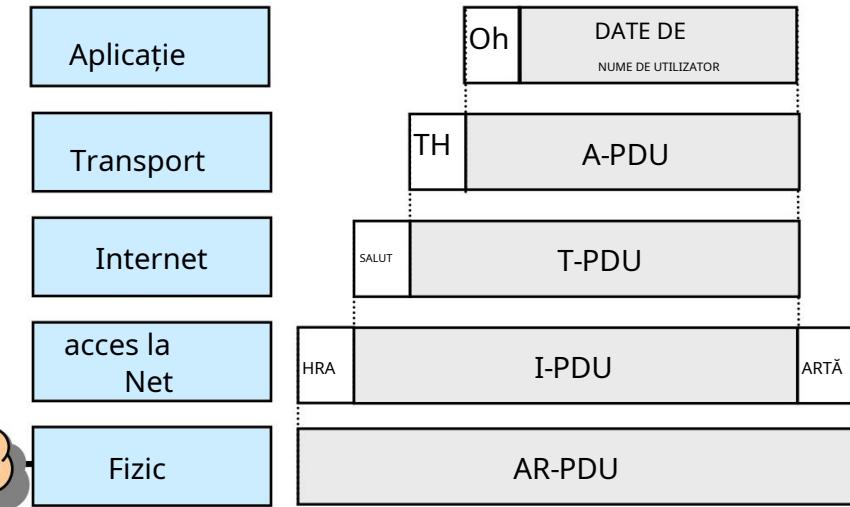


# Arhitectura TCP/IP

## ENCAPSULARE



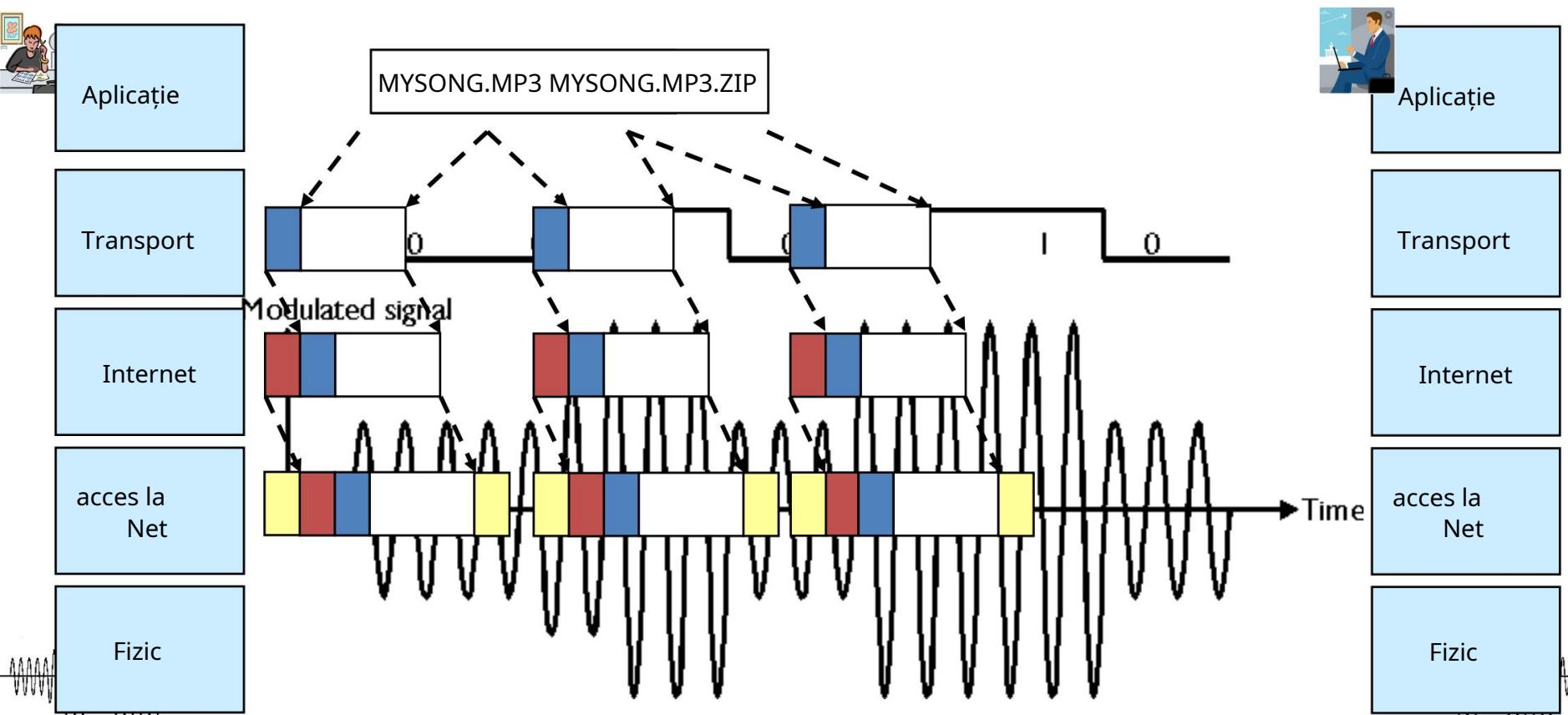
## DECAPSULARE



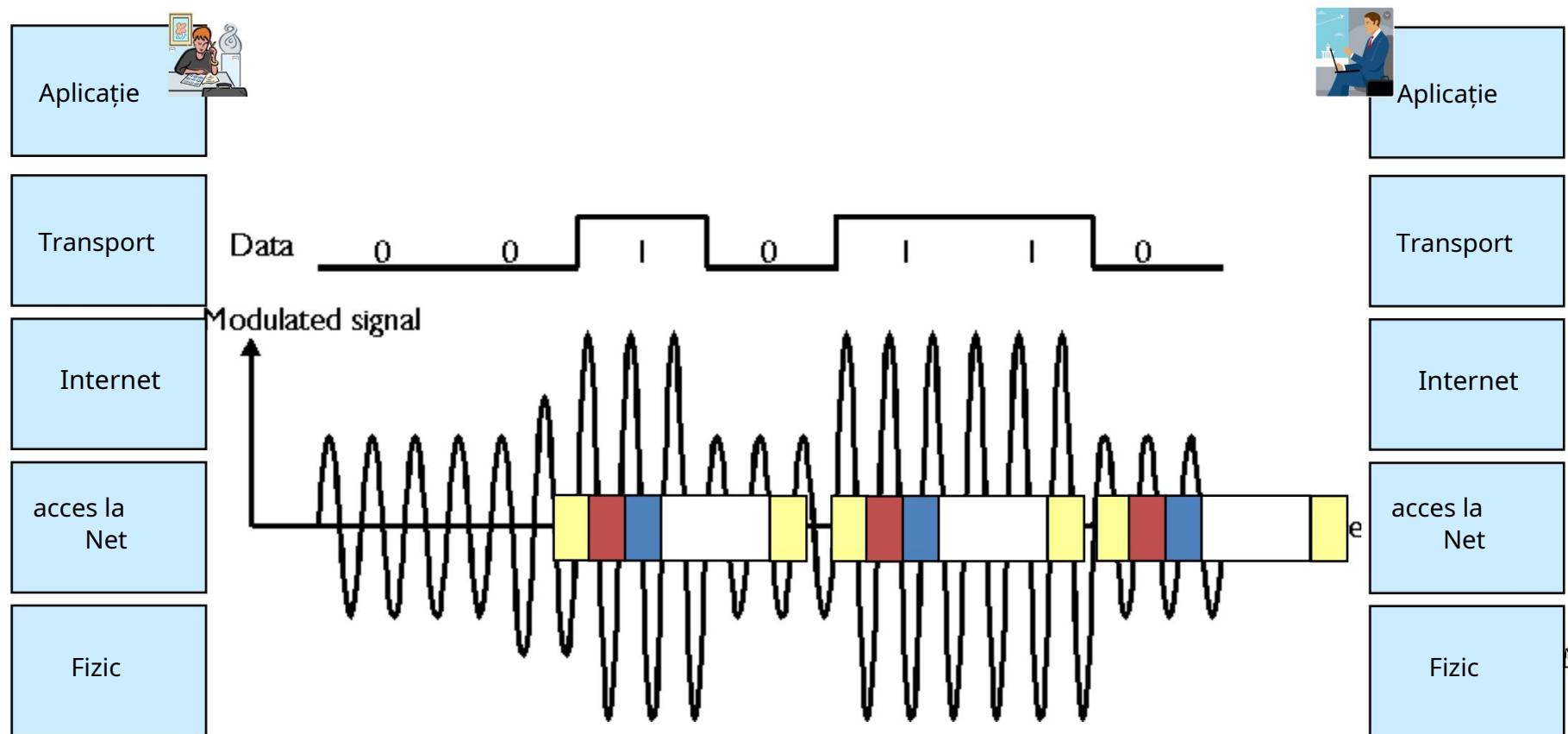
H⇒ antet

T⇒ cola

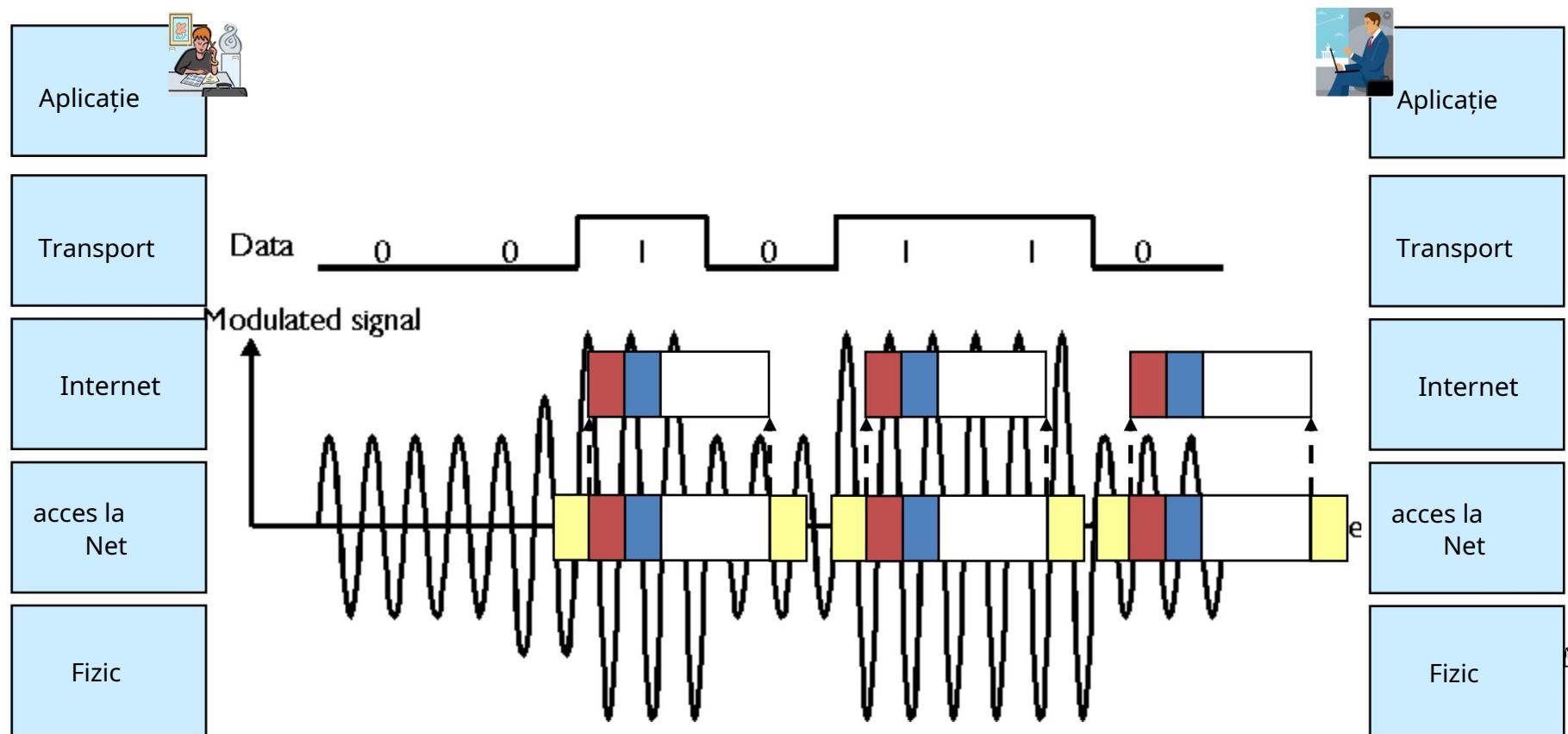
# Arhitectura TCP/IP



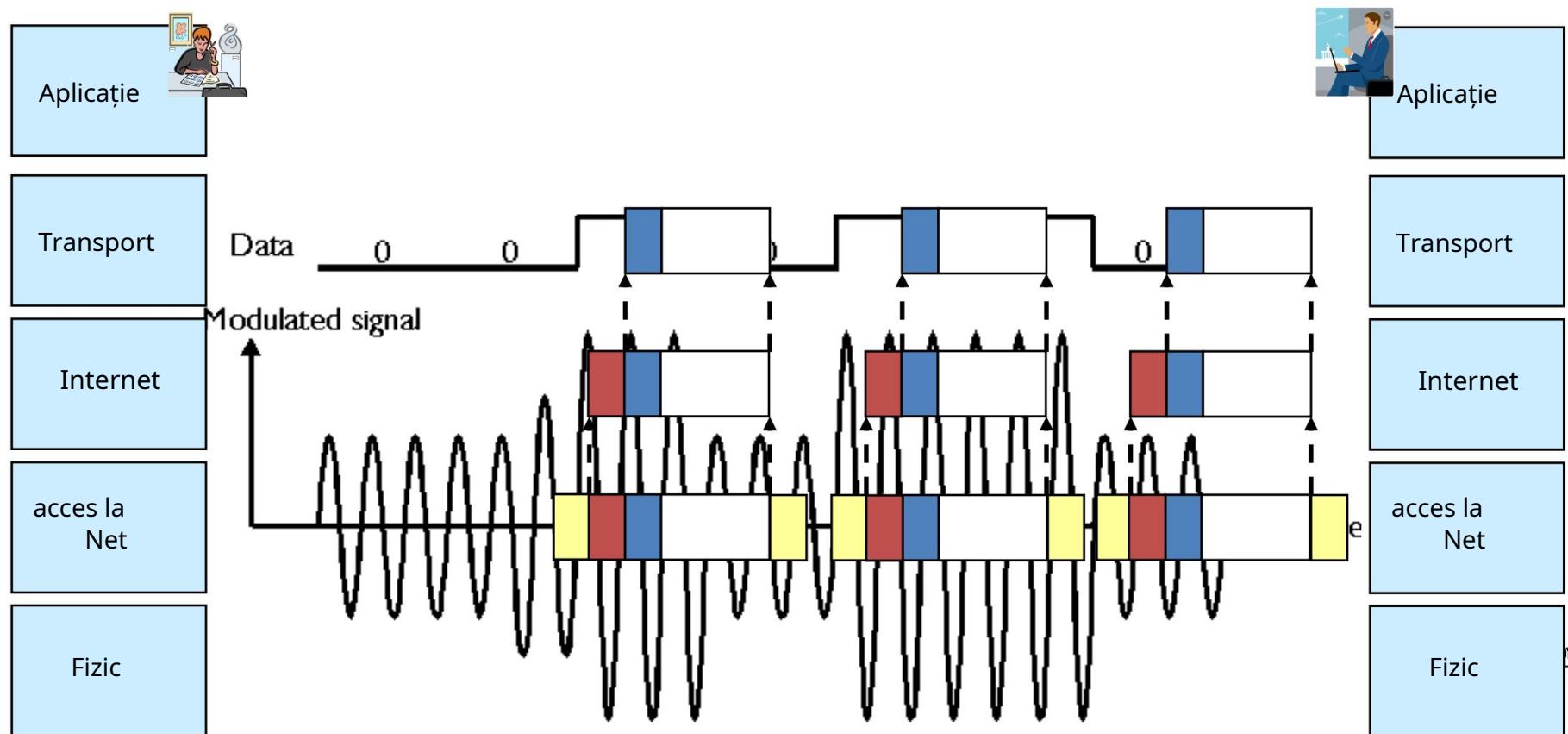
# Arhitectura TCP/IP



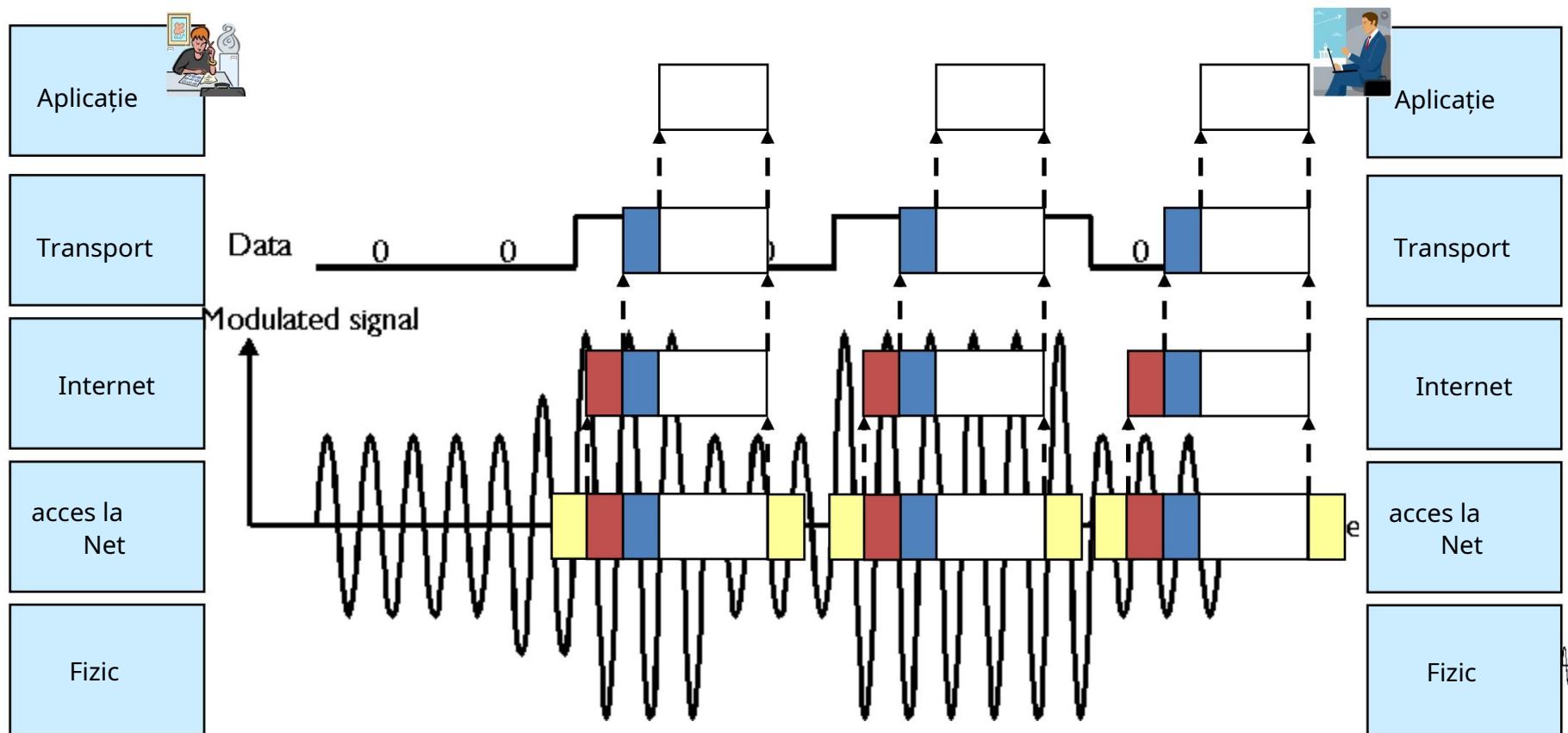
# Arhitectura TCP/IP



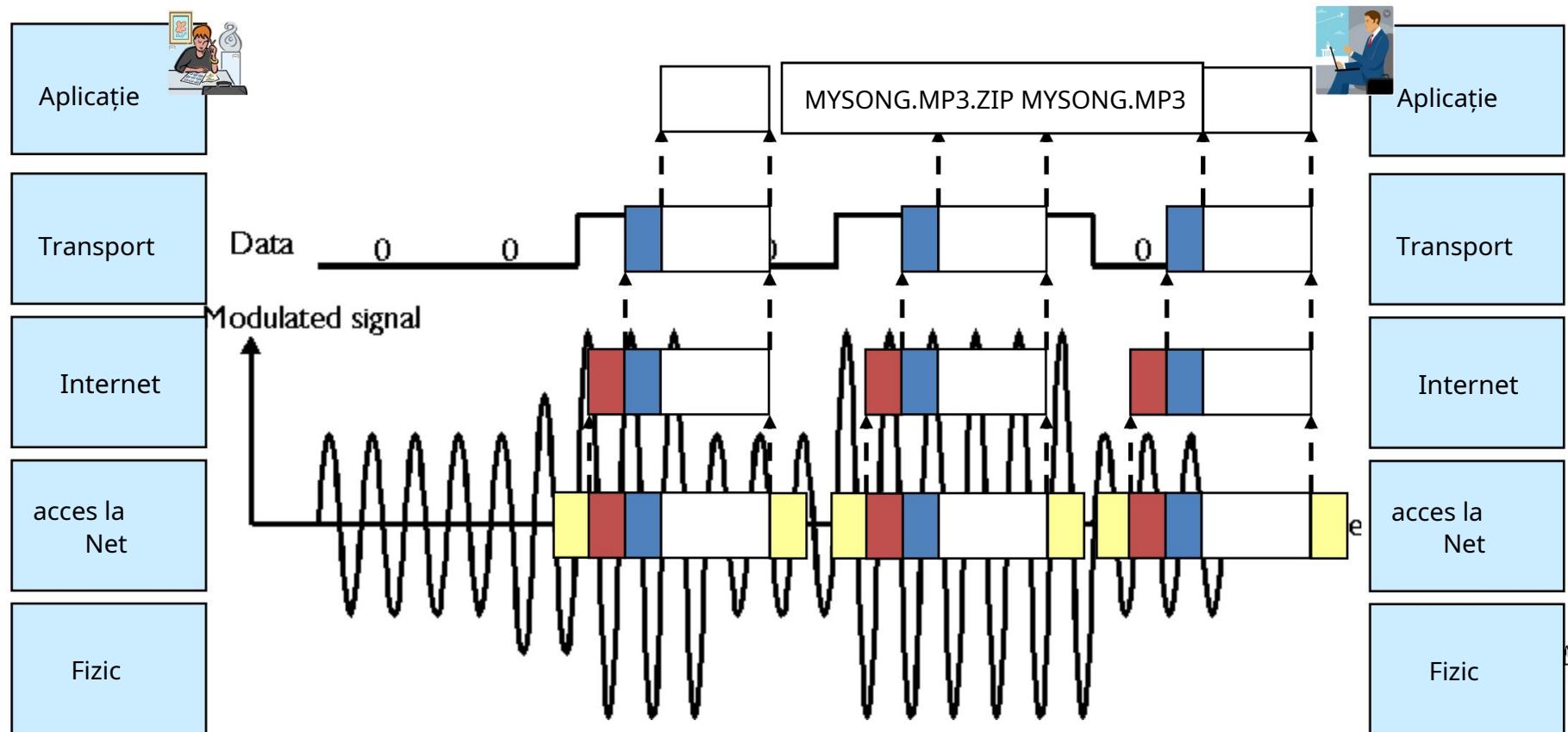
# Arhitectura TCP/IP



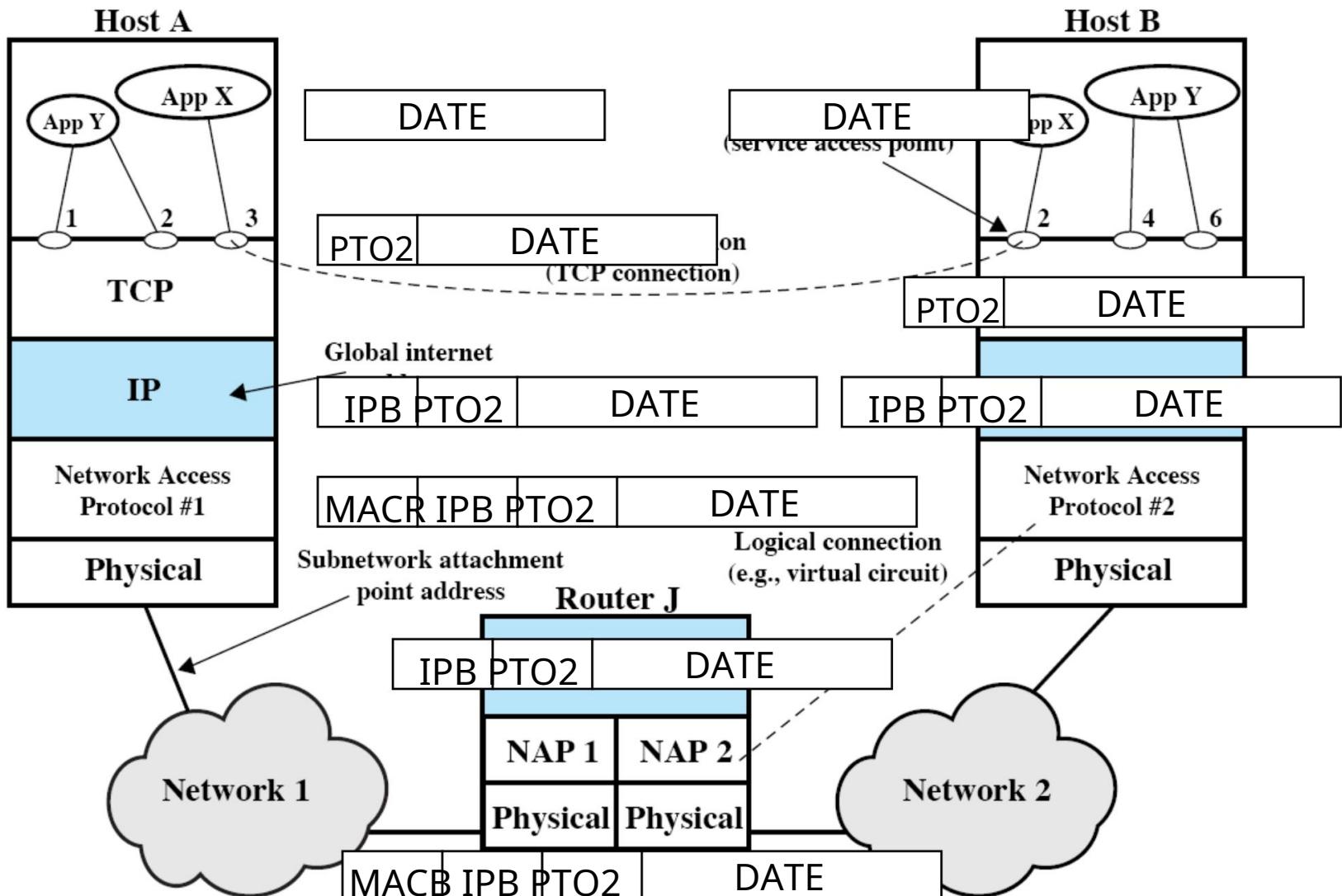
# Arhitectura TCP/IP



# Arhitectura TCP/IP



# Arhitectura TCP/IP



# Cuprins

## 1. Introducere

## 2. Priză

- 1. Caracteristicile comunicării între procese
- 2. Arhitectura TCP/IP

3. Definirea prizelor

4. Priză Java

## 3. Comunicarea de grup

- 1. Tipuri de grupuri
- 2.

Caracteristici ale comunicării de grup

## 4. Invocarea metodei de la distanță (RMI)

1. Protocole cerere-răspuns

2. RMI Design

3. Implementarea RMI

4. JavaRMI

# Definiția prizei

- În protocolele Internet, mesajele sunt trimise în perechi <adresă IP, port local>

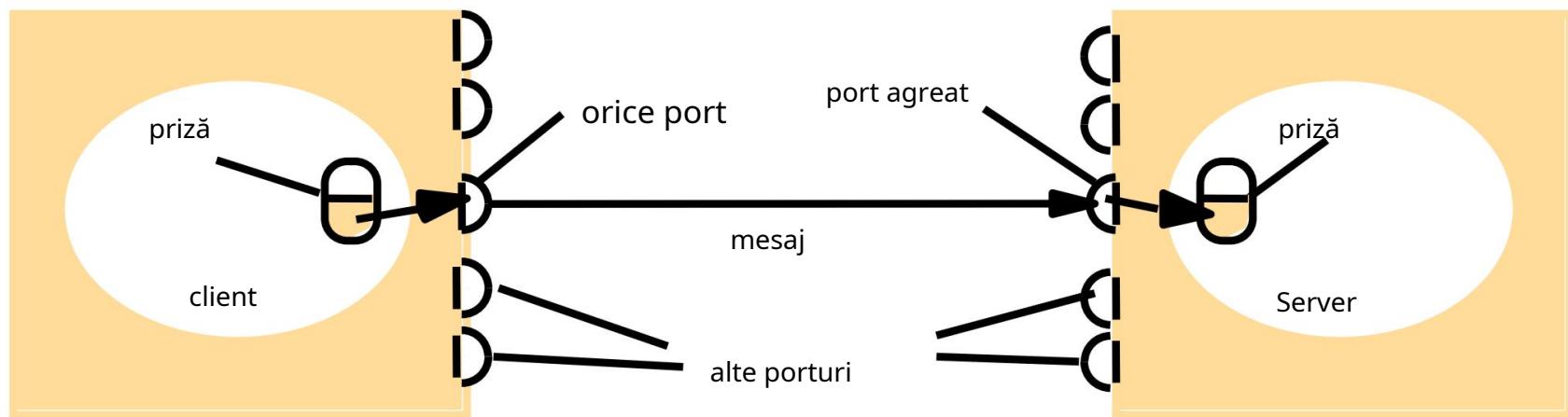
Port local: destinația unui mesaj într-un computer specificat de un număr întreg

- Procesele pot folosi mai multe porturi de la care să primească postări
- În mod normal, serverele își fac publice numerele de port, astfel încât clienții să le poată utiliza (exemplu: 20-21 FTP, 22 SSH, 80 HTTP, 1863 MSN MESSENGER, ...)

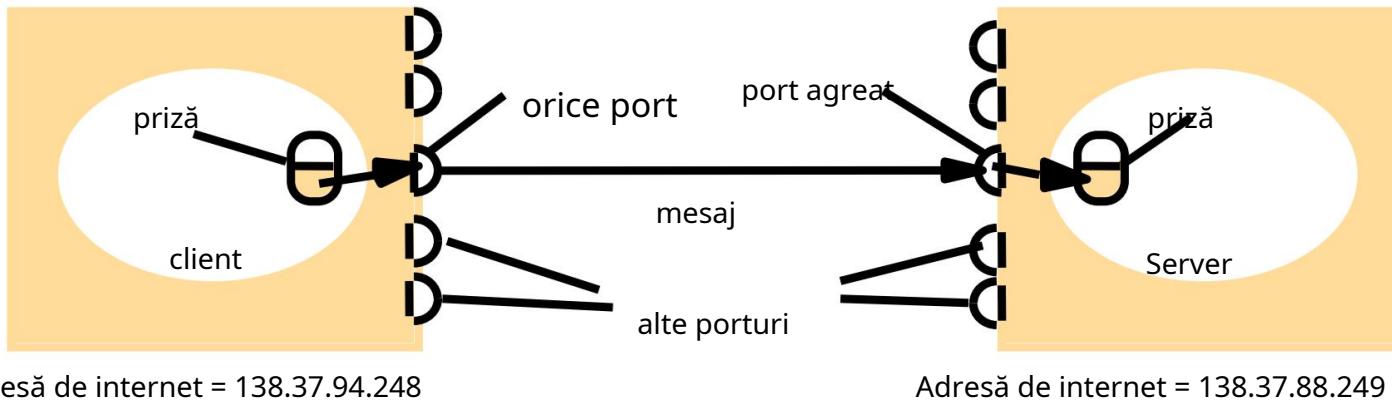
0-1023	PORTURI CUNOSCUTE	
1024-49151	PORTURI ÎNREGISTERATE	
49152-65535	PORTURI DINAMICE/PRIVATE	IANA (Internet Numere atribuite Autoritate)

# Definiția prizei

- Ele oferă un punct final în comunicarea între proceselor
- Comunicarea constă în transmiterea unui mesaj de la un socket dintr-un proces către un socket dintr-un alt proces

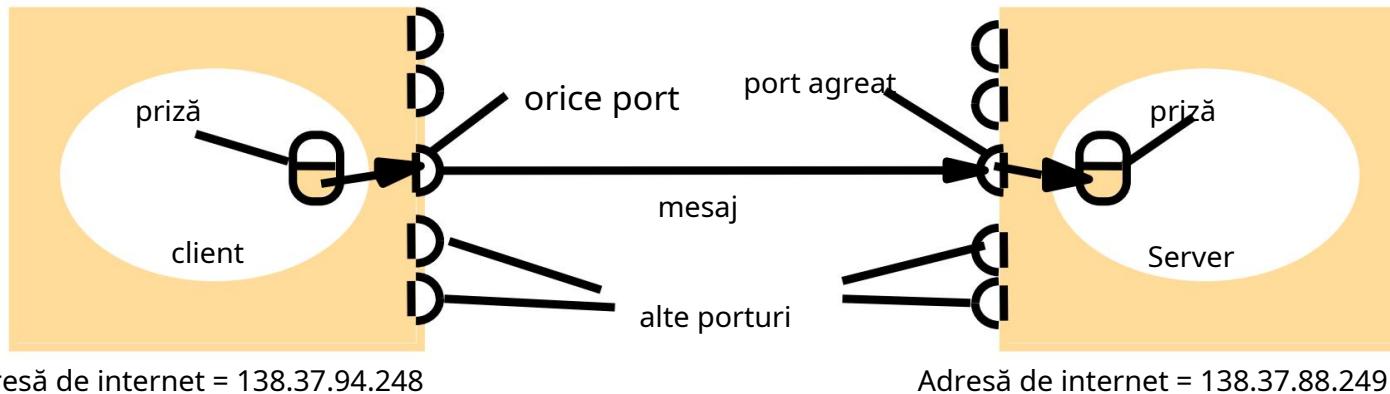


# Definiția prizei



- Pentru ca procesele să primească mesaje, soclul lor trebuie să fie legat la un port local și la o adresă IP.
- Mesajele trimise către un <port, adresă IP> pot fi primite numai de procesul al cărui socket are acel IP și acel port asociat.
- Procesele pot folosi același socket pentru a trimite și a primi mesaje
- Un proces poate folosi mai multe porturi pentru a primi mesaje DAR porturile nu pot fi partajate între mai multe procese de pe aceeași gazdă (excepție multicast)
- Socket-urile sunt asociate cu un protocol de transport specific (TCP, UDP, etc.)

# Definiția prizei



- Perechea de socket <@IP\_destination, @IP\_origin, pt\_destination, pt\_origin> definește fără echivoc o conexiune TCP sau UDP .
  - Socket server: dacă aplicația cunoscută va fi asociată IP\_server și known\_pt [0-1024], dacă aplicația utilizator/programator va fi asociată cu IP\_server și registered\_pt [1025-49151]
  - Client socket: dacă aplicația cunoscută va fi asociată cu IP\_client și pt\_dynamic [49151-65535], dacă aplicația utilizator/programator va fi asociată cu IP\_client și pt\_any [1025-65535]

# Cuprins

## 1. Introducere

## 2. Priză

- 1. Caracteristicile comunicării între procese
- 2. Arhitectura TCP/IP

- 3. Definirea prizelor

- 4. Priză Java

## 3. Comunicarea de grup

- 1. Tipuri de grupuri
- 2.

Caracteristici ale comunicării de grup

## 4. Invocarea metodei de la distanță (RMI)

- 1. Protocole cerere-răspuns

- 2. RMI Design

- 3. Implementarea RMI

- 4. JavaRMI

# Prize Java

---

- Clasa Java InetAddress:
  - Clasa reprezentând adrese IP
  - Instanțele de InetAddress care conțin adrese IP sunt poate fi creat prin apelarea metodei statice InetAddress folosind a Numele DNS ca argument

```
InetAddress aComputer = InetAddress.getByName("www.upct.es");
```

# Prize Java

## InetAddressExample.java

```
0 import java.util.Enumeration;
1 import java.net.*;
2
3 public class InetAddressExample {
4
5     public static void main(String[] args) {
6
7         // Get the network interfaces and associated addresses for this host
8         try {
9             Enumeration<NetworkInterface> interfaceList = NetworkInterface.getNetworkInterfaces();
10            if (interfaceList == null) {
11                System.out.println("--No interfaces found--");
12            } else {
13                while (interfaceList.hasMoreElements()) {
14                    NetworkInterface iface = interfaceList.nextElement();
15                    System.out.println("Interface " + iface.getName() + ":");
16                    Enumeration<InetAddress> addrList = iface.getInetAddresses();
17                    if (!addrList.hasMoreElements()) {
18                        System.out.println("\t(No addresses for this interface)");
19                    }
20                    while (addrList.hasMoreElements()) {
21                        InetAddress address = addrList.nextElement();
22                        System.out.print("\tAddress "
23                            + ((address instanceof Inet4Address ? "(v4)"
24                                : (address instanceof Inet6Address ? "(v6)" : "(?))))));
25                        System.out.println(": " + address.getHostAddress());
26                    }
27                }
28            }
29        }
30    }
31}
```

Lista cu asta  
ale gazdei  
interfețe

Obțineți și imprimați  
adresele fiecărei  
interfețe din listă

# Prize Java

## InetAddressExample.Java

```
29 } catch (SocketException se) {
30     System.out.println("Error getting network interfaces:" + se.getMessage());
31 }
32 // Get name(s)/address(es) of hosts given on command line
33 for (String host : args) {
34     try {
35         System.out.println(host + ":");
36         InetAddress[] addressList = InetAddress.getAllByName(host);
37         for (InetAddress address : addressList) {
38             System.out.println("\t" + address.getHostName() + "/" + address.getHostAddress());
39         }
40     } catch (UnknownHostException e) {
41         System.out.println("\tUnable to find address for " + host);
42     }
43 }
44 }
45 }
46 }
```

Obțineți o listă de adrese pentru numele/adresa dată

Iterează prin lista de imprimare fiecare

# Prize Java

---

- InetAdress: proprietăți de testare

- boolean isAnyLocalAddress() •
- boolean isLinkLocalAddress() •
- boolean isLoopbackAddress() •
- boolean isMulticastAddress() •
- boolean isMCGlobal() • boolean  
isMCLinkLocal() • boolean  
isMCNodeLocal() • boolean  
isMCNodeLocal() • boolean  
isMOrgLocal () • boolean  
isMCOrgLocal () timeout) • boolean  
isReachable (NetworkInterface netif, int ttl, int timeout)

- NetworkInterface: Crearea, obținerea de informații

- static Enumeration<NetworkInterface> getNetworkInterfaces() •
- static NetworkInterface getByInetAddress(adresă InetAddress) •
- static NetworkInterface getByName(nume sir) •
- Enumeration<InetAddress> getInetAddresses() • String getName()
- String getDisplayName()

# Prize: Java: TCP

## Flux TCP: socketuri de flux

- Conexiune TCP: canal bidirectional ale cărui capete sunt identificate printr-o adresă IP și un număr de port
- Orientat spre conexiune: stabilire (clientul solicită conectarea la server), transfer de date, închidere
- Clasele Socket și ServerSocket în Java
  - Socket -> unul dintre capetele conexiunii TCP
  - ServerSocket -> ascultă cererile de conexiune TCP și creează una nouă instanță socket pentru a gestiona fiecare conexiune de intrare
- Când serverul acceptă conexiunea, este creat un nou socket pentru a comunica C/S, păstrându-l pe cel vechi pentru a asculta cereri noi
- Două fluxuri: C->S și S->C
- Puteti selecta câte date sunt scrise/citite din flux și puteti forța trimiterea datelor (controlul fluxului)
- Confirmări, time out, redirecționări, mesaje duplicate, comandă
- Integritate și validitate
- Exemple: HTTP, FTP, telnet,...

Ce clase gestionează serverele? Si Clienti?

# Prize Java: TCP

## Flux TCP: socketuri de flux

### - Client TCP

1. Build Socket instance: Constructorul stabilește o conexiune TCP la <gazdă la distanță, port> specificată
2. Comunicare folosind fluxuri de intrare/ieșire socket:  
o instanță conectată a Socket conține un InputStream și un OutputStream care sunt utilizate ca și alte fluxuri de intrare/ieșire ale Java
3. Închiderea conexiunii folosind metoda Socket \* close().

\* Când un proces își închide socketul, nu va mai putea scrie date în fluxul de ieșire.

Orice date din bufferul de ieșire sunt trimise și plasate în bufferul de intrare al socket-ului destinație cu o indicație că fluxul s-a încheiat. Socket-ul de destinație poate citi datele din buffer, dar orice citire ulterioară cu tamponul de intrare gol primește un răspuns la sfârșitul fluxului.

# Prize Java: TCP

## Flux TCP: socketuri de flux

### - Client TCP

```
0 import java.net.Socket;
1 import java.net.SocketException;
2 import java.io.IOException;
3 import java.io.InputStream;
4 import java.io.OutputStream;
5
6 public class TCPEchoClient {
7
8     public static void main(String[] args) throws IOException {
9
10         if ((args.length < 2) || (args.length > 3)) // Test for correct # of args
11             throw new IllegalArgumentException("Parameter(s): <Server> <Word> [<Port>]");
12
13         String server = args[0];           // Server name or IP address
14         // Convert argument String to bytes using the default character encoding
```

# Prize Java: TCP

## Flux TCP: socketuri de flux

### - Client TCP

```

15 byte[] data = args[1].getBytes();
16 int servPort = (args.length == 3) ? Integer.parseInt(args[2]) : 7;
17 // Create socket that is connected to server on specified port
18 Socket socket = new Socket(server, servPort);
19 System.out.println("Connected to server... sending echo string");
20
21
22 InputStream in = socket.getInputStream();
23 OutputStream out = socket.getOutputStream();
24
25 out.write(data); // Send the encoded string to the server
26
27
28 // Receive the same string back from the server
29 int totalBytesRcvd = 0; // total bytes received so far
30 int bytesRcvd; // Bytes received in last read
31 while (totalBytesRcvd < data.length) {
32     if ((bytesRcvd = in.read(data, totalBytesRcvd,
33         data.length - totalBytesRcvd)) == -1)
34         throw new SocketException("Connection closed prematurely");
35     totalBytesRcvd += bytesRcvd;
36 } // data array is full
37
38 System.out.println("Received: " + new String(data));
39
40 socket.close(); // Close the socket and its streams
41 }
42 }
```

**De ce nu doar o citire()?**

**Inutil un port pentru a primi datele de la server?**

Socketurile TCP trimit și primesc secvențe de octeți. Această metodă returnează o matrice de octeți care reprezintă sirul.

Determină portul pe care ascultă serverul

Creați socket și conectați-l la server și la portul specificat

Cu fiecare instanță Socket avem un flux de intrare și un flux de ieșire. Trimitem date prin Socket dacă scriem octeți în fluxul de ieșire. Primim octeți citind din fluxul de intrare.

read() blochează până când unele date sunt disponibile, citește până la numărul maxim de octeți specificat și returnează numărul de octeți care au fost stocați în matrice. Bucla face ca matricea de date să fie umplută până când sunt primiți atât de mulți octeți cât au fost trimiși.

Convertiți matricea de octeți receptionați într-un sir și scoateți-l pe ecran

Închideți priza

# Prize Java: TCP

## Flux TCP: socketuri de flux

### - Client TCP

- Socket: Creare

- `Socket (InetAddress remoteAddr, int remotePort)`
- `Socket (String remoteHost, int remotePort)`
- `Socket (InetAddress remoteAddr, int remotePort, InetAddress localAddress, int localPort)`
- `Socket (String remoteHost, int remotePort, InetAddress localAddress, int localPort)`
- `priză ()` → Priză neconectată, trebuie să folosească metoda `connect()`.

→ Sunt alese adresa locală implicită și  
unele porturi disponibile

# Prize Java: TCP

## Flux TCP: socketuri de flux

- Client TCP
  - Priză: Operații
    - conexiune nulă (destinație SocketAddress)
    - void connect (destinație SocketAddress, int timeout)
    - InputStream getInputStream()
    - OutputStream getOutputStream()
    - voidClose()
    - void shutdownInput()
    - void shutdownOutput()

# Prize Java: TCP

## Flux TCP: socketuri de flux

### -TCP Server \_

- Stabiliti punctul final al unei comunicari si asteptați conexiunile a clientilor
1. Construiți instanța ServerSocket specificând portul local: acest socket ascultă dacă există cereri de conectare cu acel port
  2. În mod repetat:
    1. Apelați metoda accept() a clasei ServerSocket pentru a obține următoarea conexiune client (cerere primită). După stabilirea conexiunii la noul client, este creată o instanță Socket pentru a lucra cu această nouă conexiune.
    2. Comunicați cu clientul utilizând InputStream și OutputStream al noului Socket
    3. Când ati terminat, inchideți conexiunea folosind metoda close () a socketului

# Prize Java: TCP

## Flux TCP: socketuri de flux

### -TCPServer -

```

0 import java.net.*; // for Socket, ServerSocket, and InetAddress
1 import java.io.*; // for IOException and Input/OutputStream
2
3 public class TCPEchoServer {
4
5     private static final int BUFSIZE = 32; // Size of receive buffer
6
7     public static void main(String[] args) throws IOException {
8
9         if (args.length != 1) // Test for correct # of args
10             throw new IllegalArgumentException("Parameter(s): <Port>");
11
12         int servPort = Integer.parseInt(args[0]);
13
14         // Create a server socket to accept client connection requests
15         ServerSocket servSock = new ServerSocket(servPort);
16
17         int recvMsgSize; // Size of received message
18         byte[] receiveBuf = new byte[BUFSIZE]; // Receive buffer
19
20         while (true) { // Run forever, accepting and servicing connections
21             Socket clntSock = servSock.accept(); // Get client connection
22
23             SocketAddress clientAddress = clntSock.getRemoteSocketAddress();
24             System.out.println("Handling client at " + clientAddress);
25

```

Creați ServerSocket și ascultați cererile clientului pe portul specificat

Buclă infinită pentru a gestiona cererile primite de la clienți

Blocați până să se întâmple cererea. Acceptați cererea clientului. Returnați o instanță Socket, conectată la socket-ul clientului și gata să citească și să scrie date

# Prize Java: TCP

## Flux TCP: socketuri de flux

### -TCPserver \_

```
20     while (true) { // Run forever, accepting and servicing connections
21         Socket clntSock = servSock.accept();      // Get client connection
22
23         SocketAddress clientAddress = clntSock.getRemoteSocketAddress();
24         System.out.println("Handling client at " + clientAddress);
25
26         InputStream in = clntSock.getInputStream();
27         OutputStream out = clntSock.getOutputStream();
28
29         // Receive until client closes connection, indicated by -1 return
30         while ((recvMsgSize = in.read(recvBuf)) != -1) {
31             out.write(recvBuf, 0, recvMsgSize);
32         }
33         clntSock.close(); // Close the socket. We are done with this client!
34     }
35     /* NOT REACHED */
36 }
37 }
```

Returnează adresa și portul socket-ului clientului

Obține InputStream și OutputStream ale Socket -ului pe care le-am creat pe server. Ceea ce scriem în acest Output Stream ieșe în InputStream

a clientului. Ce scrie clientul în OutputStream va ieși în acest InputStream.

Citiți octeți din InputStream și scrieți-i în OutputStream până când clientul închide conexiunea

închideți priza

# Prize Java: TCP

## Flux TCP: socketuri de flux

### -TCP Server -

- ServerSocket: Creare

Asociat la un port local și gata să accepte conexiuni de intrare

- ServerSocket (int localPort)
- ServerSocket (int localPort, int queueLimit)
- ServerSocket (int localPort, int queueLimit, InetAddress localAddress)
- ServerSocket() → Nu este asociat cu niciun port local, trebuie să fie legat de un port folosind bind()

- ServerSocket: Operare

- void bind (port int)
- void bind (int port, int queueLimit)
- Socketaccept()
- void close()

# Prize Java: UDP

## Datagrama UDP: sociuri de datagramă

- Trimis de la expeditor la destinatar fără confirmări sau retransmisii/repetiții -> protocol fără conexiune
- Este necesar să specificați dimensiunea matricei în octeți unde să primească mesajul (maximum  $2^{16}$  octeți)
- Trimitere fără blocare și primire blocată
- Posibile defecte de omisiune (conform modelelor SID)
- Exemple: DNS
- Detectarea erorilor de transmisie (coruptia datelor), eliminarea mesajelor corupte
- Prizele UDP nu trebuie conectate înainte de a fi utilizate

# Prize Java: UDP

## Datagrama UDP: socket -uri de datagramă

- Fiecare datagramă poartă adresa destinației fiind independentă de alte datagrame
- De îndată ce este creat, soclul UDP poate fi folosit pentru a trimite/primi mesaje către/de la orice adresă
- Prizele UDP pot întâmpina pierderi și reordonări
- Clasele DatagramPacket și DatagramSocket în Java
  - DatagramPacket -> unitate de trimitere a mesajelor în UDP, utilizată cu metodele send() și receive() ale unui DatagramSocket, inclusiv trimiterea <adresă, port> sau expeditor <adresă, port>
  - DatagramPacket: Creare
    - DatagramPacket(date octet[], lungime int)
    - DatagramPacket (byte[] date, int offset, int lungime)
    - DatagramPacket (byte[] data, InetAddress remoteAddr, int remotePort)
    - DatagramPacket(byte[] data, int offset, InetAddress remoteAddr, int remoteport)
    - DatagramPacket(byte[] data, int length.SocketAddress sockAddr)
    - DatagramPacket(byte[] data, int offset, int length.SocketAddress sockAddr)

# Prize Java: UDP

## Datagrama UDP: socket -uri de datagramă

### - Client UDP

1. Construiți instanța DatagramSocket, specificați opțional adresa și portul local
2. Comunicați prin trimiterea și primirea instanțelor de DatagramPacket folosind metodele send() și receive() de DatagramSocket
3. Când ați terminat, eliminați resursele socketului folosind metoda close() a lui DatagramSocket

# Prize Java: UDP

## Datagrama UDP: sociuri de datagramă

### - Client UDP

```

0 import java.net.DatagramSocket;
1 import java.net.DatagramPacket;
2 import java.net.InetAddress;
3 import java.io.IOException;
4 import java.io.InterruptedIOException;
5
6 public class UDPEchoClientTimeout {
7
8     private static final int TIMEOUT = 3000;    // Resend timeout (milliseconds)
9     private static final int MAXTRIES = 5;        // Maximum retransmissions
10
11    public static void main(String[] args) throws IOException {
12
13        if ((args.length < 2) || (args.length > 3)) { // Test for correct # of args
14            throw new IllegalArgumentException("Parameter(s): <Server> <Word> [<Port>]");
15        }
16        InetAddress serverAddress = InetAddress.getByName(args[0]); // Server address
17        // Convert the argument String to bytes using the default encoding
18        byte[] bytesToSend = args[1].getBytes();
19
20        int servPort = (args.length == 3) ? Integer.parseInt(args[2]) : 7;
21
22        DatagramSocket socket = new DatagramSocket();
23
24        socket.setSoTimeout(TIMEOUT); // Maximum receive blocking time (milliseconds)
25

```

Creați un DatagramSocket care poate trimite date către orice socket UDP. Va fi selectată orice adresă de interfață disponibilă, idem port (ați fi putut folosi setLocalPort() și setLocalAddress() sau alt constructor).

Controlează timpul maxim în care operațiunea receive() va bloca

# Prize Java: UDP

## Datagrama UDP: sociuri de datagramă

### - Client UDP

```

26 DatagramPacket sendPacket = new DatagramPacket(bytesToSend, // Sending packet
27     bytesToSend.length, serverAddress, servPort);
28
29 DatagramPacket receivePacket =                               // Receiving packet
30     new DatagramPacket(new byte[bytesToSend.length], bytesToSend.length);
31
32 int tries = 0;      // Packets may be lost, so we have to keep trying
33 boolean receivedResponse = false;
34 do {
35     socket.send(sendPacket);           // Send the echo string
36     try {
37         socket.receive(receivePacket); // Attempt echo reply reception
38
39         if (!receivePacket.getAddress().equals(serverAddress)) { // Check source
40             throw new IOException("Received packet from an unknown source");
41         }
42         receivedResponse = true;
43     } catch (InterruptedException e) { // We did not get anything
44         tries += 1;
45         System.out.println("Timed out, " + (MAXTRIES - tries) + " more tries...");
46     }
47 } while ((!receivedResponse) && (tries < MAXTRIES));

```

Creați o instanță de DatagramPacket pentru a trimite informații. Sunt indicate datele de trimis, dimensiunea datelor, adresa de destinație și portul de destinație.

Creați o instanță de DatagramPacket pentru a primi date. Indică unde sunt stocate datele primite și lungimea matricei.

Se trimit datagrama

Recepția datagramelor.  
Acestă metodă se blochează până când expiră expirarea sau se primește datagrama.

# Prize Java: UDP

## Datagrama UDP: sociuri de datagramă

### - Client UDP

```
48     if (receivedResponse) {  
49         System.out.println("Received: " + new String(receivePacket.getData()));  
50     } else {  
51         System.out.println("No response -- giving up.");  
52     }  
53 }  
54     socket.close();  
55 }  
56 }
```



→ Închideți priza

# Prize Java: UDP

## Datagrama UDP: sociuri de datagramă

### - Client UDP

- DatagramSocket: Creare
  - DatagramSocket()
  - DatagramSocket (int localPort)
  - DatagramSocket (int localPort, InetAddress localAddr)
- DatagramSocket: Conexiune și Închidere
  - conexiune nulă (InetAddress remoteAddr, int remotePort)
  - conexiune nulă (SocketAddress remoteSockAddr)
  - void deconectare() → Eliminați asocierea DatagramSocket cu portul și adresa de la distanță
  - void close()

Setează adresa și portul de la distanță. Priza poate comunica doar cu acea destinație. Socket primește date numai de la această destinație

# Prize Java: UDP

## Datagrama UDP: sociuri de datagramă

-UDPServer \_

1. Construiți o instanță DatagramSocket, specificați portul local și optional adresa locală -> server gata să primească datagrame de la orice client
2. Primiți o instanță a DatagramPacket folosind metoda receive() de la DatagramSocket. Când această metodă este executată datagrama conține adresa clientului -> putem ști unde să trimitem răspunsul (dacă este necesar)
3. Comunicați prin trimiterea și primirea DatagramPackets folosind metodele send() și receive() ale DatagramSocket()

# Prize Java: UDP

## Datagrama UDP: sociuri de datagramă

### -UDPServer \_

```

0 import java.io.IOException;
1 import java.net.DatagramPacket;
2 import java.net.DatagramSocket;
3
4 public class UDPEchoServer {
5
6     private static final int ECHOMAX = 255; // Maximum size of echo datagram
7
8     public static void main(String[] args) throws IOException {
9
10         if (args.length != 1) { // Test for correct argument list
11             throw new IllegalArgumentException("Parameter(s): <Port>");
12         }
13
14         int servPort = Integer.parseInt(args[0]);
15
16         DatagramSocket socket = new DatagramSocket(servPort);
17         DatagramPacket packet = new DatagramPacket(new byte[ECHOMAX], ECHOMAX);
18
19         while (true) { // Run forever, receiving and echoing datagrams
20             socket.receive(packet); // Receive packet from client
21             System.out.println("Handling client at " + packet.getAddress().getHostAddress()
22                             + " on port " + packet.getPort());
23             socket.send(packet); // Send the same packet back to client
24             packet.setLength(ECHOMAX); // Reset length to avoid shrinking buffer
25         }
26         /* NOT REACHED */
27     }
28 }
```

(1) De ce trebuie să indice portul?

(2) Ce efect are ECHOMAX?

(3) Există blocaj?

(4) De unde sunt acceptate pachetele la recepție?

(5) Cum știți adresa de destinație și portul?

Creați o instanță DatagramSocket (1)

Creați o instanță de DatagramPacket (2)

Primește date în pachet (3) (4)

Trimite aceleași date în pachet (5)

# Cuprins

## 1. Introducere

## 2. Priză

1. Caracteristicile comunicării între procese 2. Arhitectura TCP/IP

3. Definirea prizelor

4. Priză Java

## 3. Comunicarea de grup

1. Tipuri de grupuri 2.

Caracteristici ale comunicării de grup

## 4. Invocarea metodei de la distanță (RMI)

1. Protocole cerere-răspuns

2. RMI Design

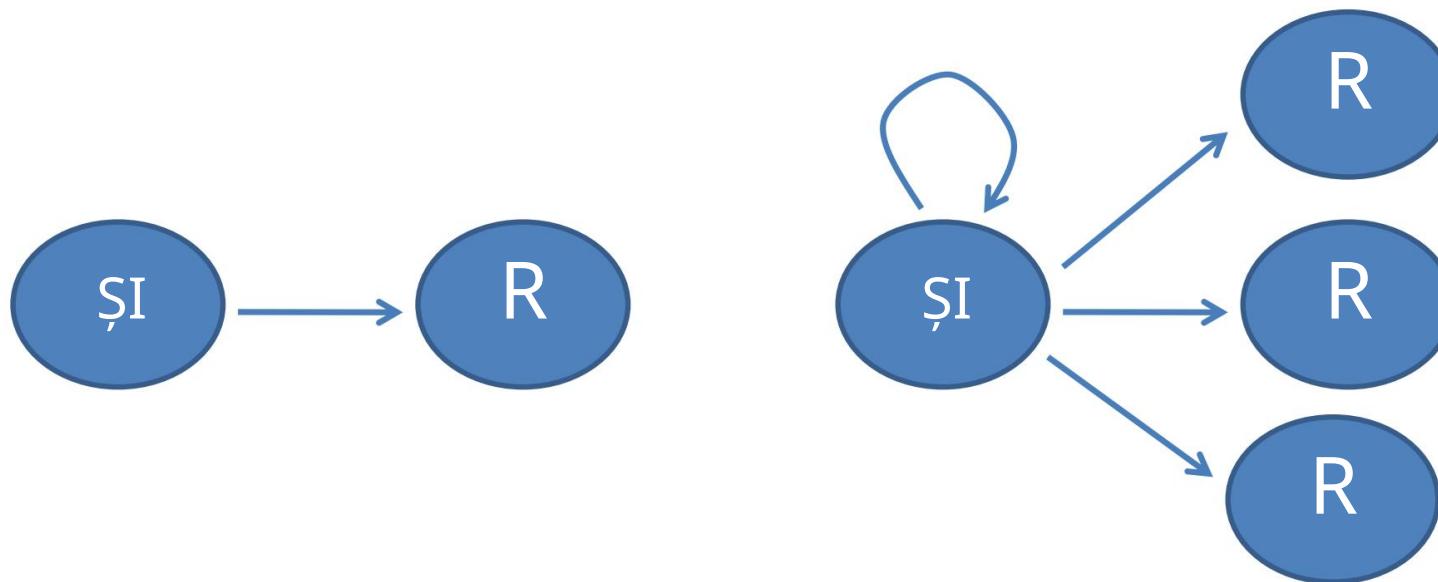
3. Implementarea RMI

4. JavaRMI

# comunicare de grup

Grup: colecție de procese care acționează împreună într-un anumit sistem

- Proprietate fundamentală: atunci când un mesaj este trimis grupului însuși, toți membrii grupului îl primesc



# comunicare de grup

Grup: colecție de procese care acționează împreună într-un anumit sistem

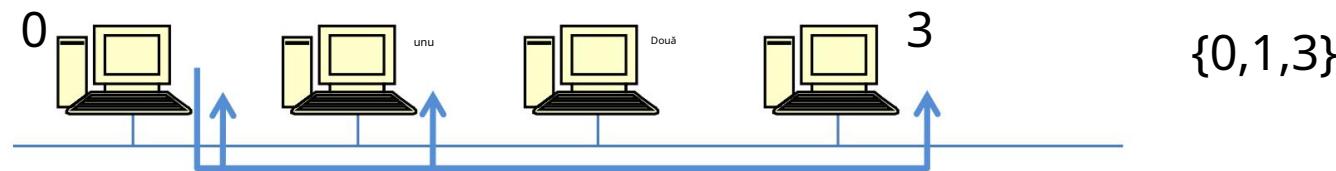
- Tipuri de grupuri:

- Închis
- Deschis
- Static
- Dinamic
- Suprapus
- De la colegi
- Ierarhic

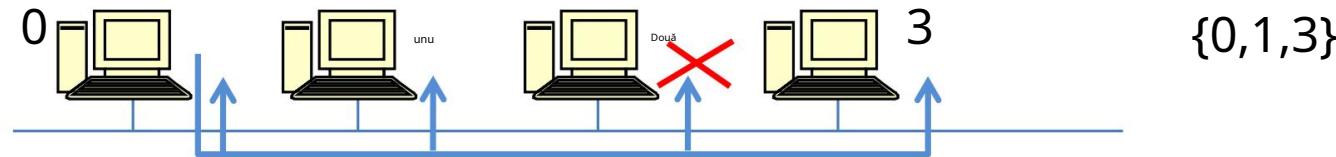
# comunicare de grup

- Pentru a implementa comunicarea de grup

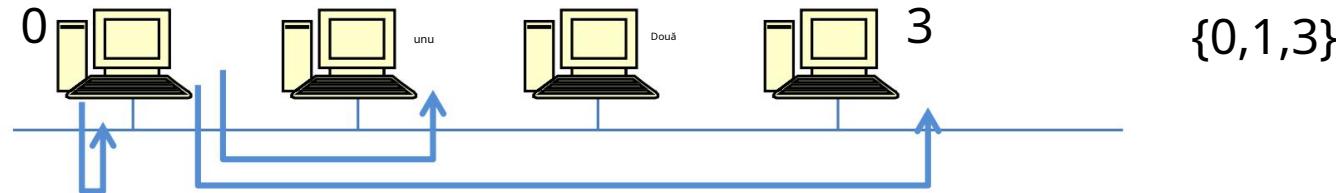
- Multistream



- Transmitere simplă



- Unitransmisie



# comunicare de grup

---

- Calitatea de membru: este necesară comunicarea în grup metoda de creare și ștergere a grupurilor (includeți membri noi, eliminați membri etc.)
  - Server de grup:
    - menține baza de date și membrii •
    - Direct, eficient și ușor de implementat
    - Punct de eșec
  - Management distribuit

# comunicare de grup

---

- Adresare: Pentru a trimite un mesaj către grup, o procedură trebuie să aibă o modalitate de a specifica acel grup.
  - Oferirea fiecărui grup a unei adrese (adresă IP multicast)
  - Alte opțiuni: unitransmisie, difuzare, adresare predicat, ...

# comunicare de grup

---

- Atomicitate: atunci când un mesaj este trimis grupului, acesta mesajul trebuie să ajungă la toți membrii grupului sau niciunul
- Comandă: un sistem distribuit care utilizează comunicarea de grup trebuie să aibă o semantică bine definită în ceea ce privește ordinea de livrare a mesajelor
  - Comandă totală: toate mesajele trimise unui grup sunt primite de toate procesele în aceeași ordine
  - Ordonare cauzală: și `sendmx` ® `sendmy` þ `receivemx` ® primesc în toate procesele

# Cuprins

## 1. Introducere

## 2. Priză

1. Caracteristicile comunicării între procese 2. Arhitectura TCP/IP

3. Definirea prizelor

4. Priză Java

## 3. Comunicarea de grup

1. Tipuri de grupuri 2.

Caracteristici ale comunicării de grup

## 4. Invocarea metodei de la distanță (RMI)

1. Protocole cerere-răspuns

2. RMI Design

3. Implementarea RMI

4. JavaRMI

# Invocarea metodei de la distanță (RMI)

---

- Programarea orientată pe obiect este extinsă pentru a permite obiectelor din diferite procese să comunice între ele prin RMI.
- RMI este o extensie a invocării metodei local, permitând unui obiect dintr-un proces să invoke metode ale unui obiect dintr-un alt proces

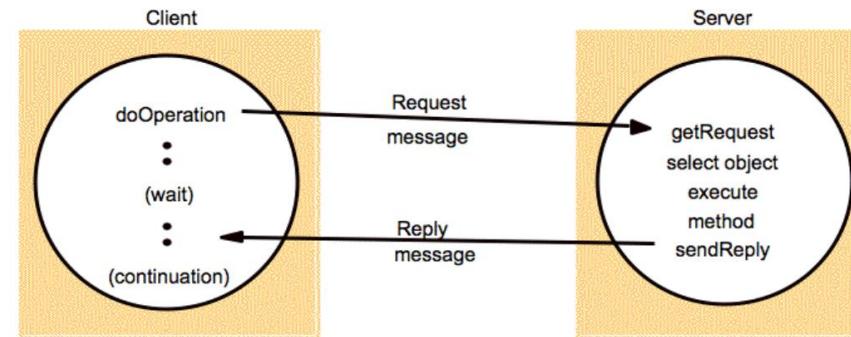
# Protocole cerere-răspuns

---

- Această formă de comunicare este concepută pentru roluri de suport și schimb de mesaje în interacțiuni tipice client-server
- În mod normal, comunicarea cerere-răspuns este:
  - sincron: clientul se blochează până când răspunsul sosește de la server (deși este posibil să îl implementezi ca asincron)
  - Fiabil: răspunsul de la server servește drept confirmare

# Protocole cerere-răspuns

- Protocol bazat pe trei primitive de comunicare:  
doOperation, getRequest, sendReply



`public byte[] doOperation (RemoteObjectRef o, int methodId, byte[] arguments)`

sends a request message to the remote object and returns the reply.

The arguments specify the remote object, the method to be invoked and the arguments of that method.

`public byte[] getRequest ()`

acquires a client request via the server port.

`public void sendReply (byte[] reply, InetAddress clientHost, int clientPort);`

sends the reply message `reply` to the client at its Internet address and port.

Instanță a clasei RemoteRef care reprezintă referințe la servere la distanță

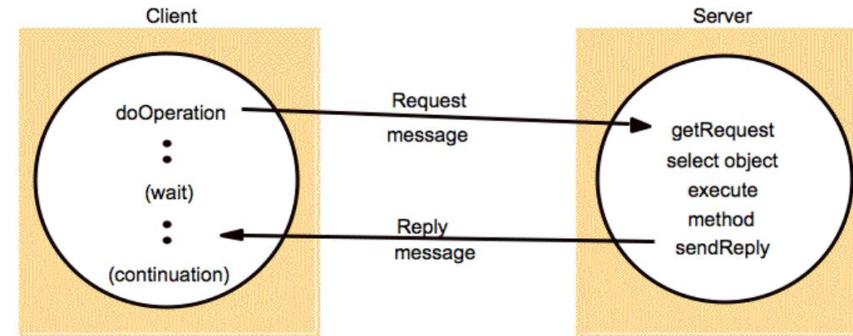
Această clasă oferă metode pentru a obține adresa IP și portul asociat cu serverul

Trimiteți un mesaj de solicitare către server și apelați `receive()` pentru a obține un răspuns

Prin urmare, oricine sună `doOperation` se blochează până când serverul face operația și transmite mesajul de răspuns către client

# Protocole cerere-răspuns

- Protocol bazat pe trei primitive de comunicare:  
doOperation, getRequest, sendReply



`public byte[] doOperation (RemoteObjectRef o, int methodId, byte[] arguments)`

sends a request message to the remote object and returns the reply.

The arguments specify the remote object, the method to be invoked and the arguments of that method.

`public byte[] getRequest ()`,

acquires a client request via the server port.

`public void sendReply (byte[] reply, InetAddress clientHost, int clientPort);`

sends the reply message `reply` to the client at its Internet address and port.

Este folosit de un proces de server pentru a obține cereri de servicii. Odată ce operația de efectuat este invocată, va folosi `sendReply` pentru a trimite mesajul de răspuns

Odată ce operația de efectuat este invocată, va folosi `sendReply` pentru a trimite mesajul de răspuns

# Protocole cerere-răspuns

- Informațiile care sunt transmise într-un mesaj de solicitare sau răspuns:

messageType	<i>int (0=Request, 1=Reply)</i>
requestId	<i>int</i>
objectReference	<i>RemoteObjectRef</i>
methodId	<i>int or Method</i>
arguments	<i>// array of bytes</i>

- Identifieri de mesaje: necesari pentru gestionarea mesajelor pentru a oferi o livrare fiabilă sau o comunicare cerere-răspuns.
  - Două câmpuri: requestID (int)
  - Identifierul procesului de trimitere (ex. IP și port)

# Protocole cerere-răspuns

- Model de eșec: Dacă cele trei primitive de comunicare sunt implementați cu UDP atunci
  - Eșecuri de comunicare
    - Eșecuri de omisiune
    - Livrarea mesajelor în aceeași ordine în care sunt primite nu este garantată. au fost trimisi
  - Eșecuri de proces
- Un timeout poate fi utilizat în doOperation pentru acele cazuri în care un server se blochează sau se pierde un mesaj de cerere-răspuns.
- Timeouts: diverse opțiuni despre ce trebuie făcut dacă apare un timeout în doOperation
  - Semnificați defecțiunea clientului
  - Repetați trimitera mesajului de solicitare până când primiți răspuns sau până când este sigur că serverul nu răspunde (dar nu din cauza pierderii mesajului sau a întârzierii rețelei)

# Protocole cerere-răspuns

- Eliminați mesajele duplicate: Dacă mesajul duplicat cererea este retransmisă ☐ serverul poate primi de mai multe ori serverul execută aceeași operațiune de mai multe ori?
  - Protocol cerere-răspuns conceput pentru a recunoaște mesajele succesive de la același client cu același identificator de solicitare și pentru a filtra duplicatele
- Mesaje de răspuns pierdut: dacă serverul a trimis deja răspunsul când aceeași cerere sosește din nou ☐ execută operația din nou, dacă nu a stocat rezultatul
  - Operație idempotent: operație care poate fi executată de mai multe ori obținându-se același rezultat
    - Un server cu toate operațiunile idempotente nu ar trebui să faceți griji să executați aceeași operațiune de mai multe ori

# Protocole cerere-răspuns

---

- Istorici: Serverele care trebuie să retransmitem un răspuns fără a executa din nou operația pot folosi istoricul lor.
  - Structură care conține o înregistrare a mesajelor care au fost transmis
  - Fiecare intrare conține: identificatorul cererii, mesajul, clientul identificator
  - Costul memoriei? Soluție? Clienții pot face o singură solicitare la un moment dat... încă o problemă? Dacă e ultima cerere?

# Protocole cerere-răspuns

- Tipuri de protocole cerere-răspuns: se vor comporta diferit în cazul unor erori de comunicare.
  - Protocol de solicitare(R).
    - Nu există nicio valoare de răspuns în operarea de la distanță și clientul nu nevoie de confirmare de finalizare
  - Protocol cerere-răspuns (RR).
    - Util pentru majoritatea schimburilor C/S, nu este necesară confirmarea (Răspunsul propriu al lui S funcționează, la fel cu următoarea solicitare de la C)
  - protocol cerere-răspuns-confirmare răspuns (RRA).
    - Ack-ul conține requestID-ul mesajului de răspuns care se confirmă, astfel încât S poate elimina mesajele din istoricul său (cel care este confirmat și cele anterioare), nu blochează C dar consumă mijloace
- Exemplu de protocol cerere-răspuns: HTTP
  - [http://en.wikipedia.org/wiki/Hypertext\\_Transfer\\_Protocol](http://en.wikipedia.org/wiki/Hypertext_Transfer_Protocol)
  - <http://www.tutorialspoint.com/http/index.htm>

# Design RMI

## MODEL OBIECTUL

- Programul orientat pe obiecte constă dintr-o colecție de obiecte (date + metode)
- Comunicarea între obiecte Referințe la obiecte. În java, o variabilă stochează un obiect, de fapt stochează o referință la acel obiect.

Obiect a cărui metodă se numește=țintă=receptor Interfețe. Ei definesc metode cu argumentele, rezultatele și excepțiile lor.

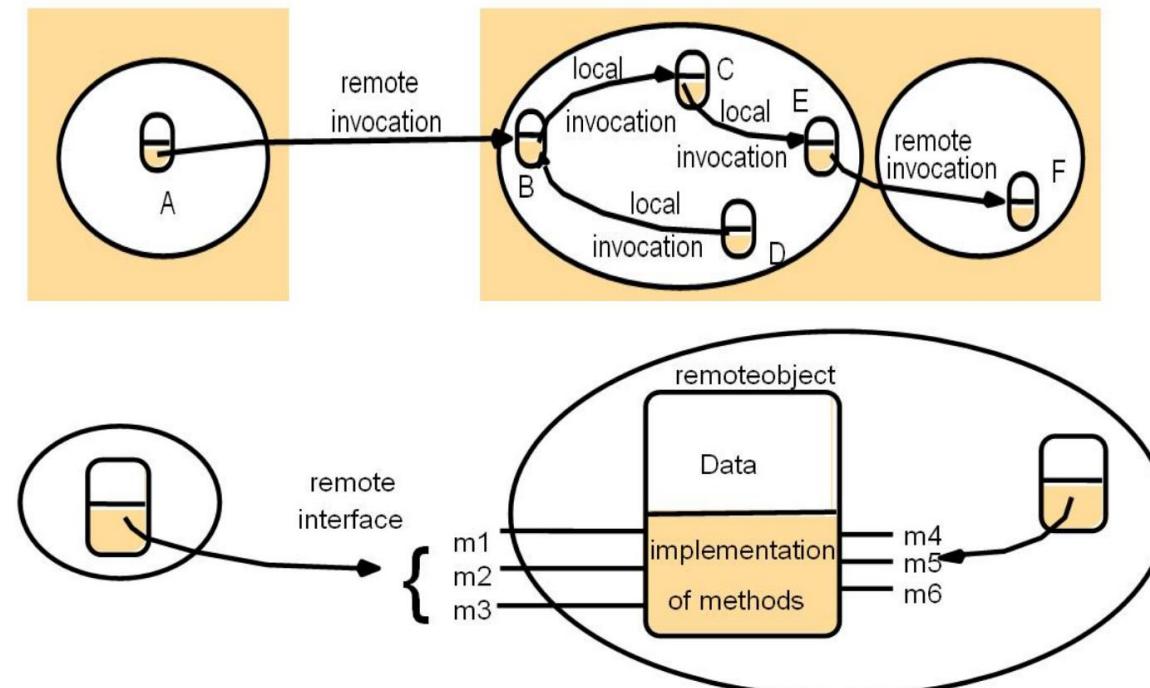
Acțiuni. Ele sunt pornite atunci când un obiect apelează o metodă pe alt obiect.

Excepții. Mod curat de a gestiona erorile de rulare.

Colectarea gunoiului. Spațiu liber ocupat de obiecte care nu sunt folosite.

## OBIECTE DISTRIBUITE

- Starea unui obiect este formată din valorile variabilelor sale.
- În program Orientat obiect, starea unui program împărțită în diferite stări ale obiectelor sale → face distribuția logică fizică → UȘOAR
- Sisteme de obiecte distribuite în arhitectura C/S



# Design RMI

## Semantica invocarii

doOperation(): mesaj de cerere de reîncercare, filtrare pentru duplicate, retransmite rezultate

- varietate de combinații

-Ar putea fi: apelantul nu poate ști dacă o metodă a fost executată o dată sau nu (nu există toleranță la erori)

- Cel puțin o dată: apelantul primește un rezultat sau o excepție care raportează că nu a fost primit niciun rezultat (prin retransmiterea mesajelor de solicitare)

-Cel mult o dată: invocatorul fie primește un rezultat, fie o excepție care raportează niciun rezultat primit (toate măsurile de toleranță la erori)

## Transparentă

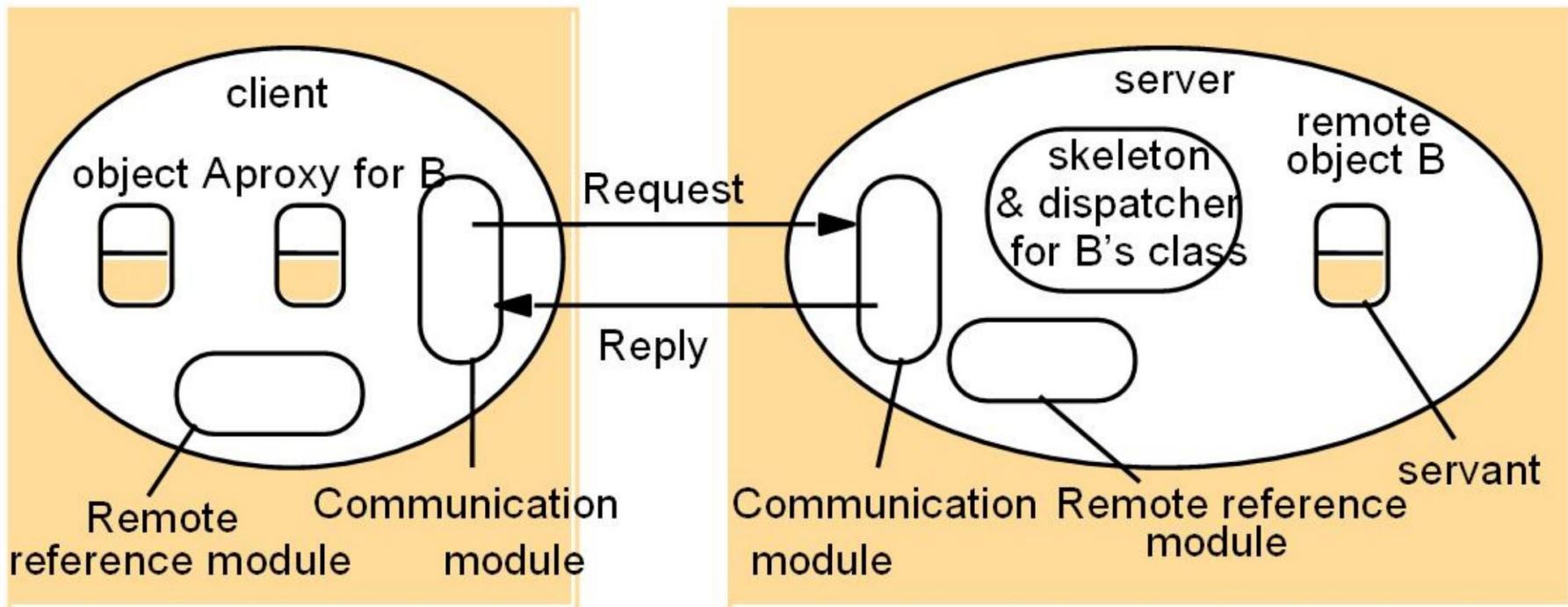
Ambalare  
trecerea mesajului  
Locație și contact  
cu obiect la distanță

}  
- Mai vulnerabil la eșec decât apelurile locale

SINTAXA DE INVOCARE LA DISTANȚĂ  
CA LOCAL, dar interfețele trebuie să indice dacă un obiect este local sau la distanță

Măsuri de toleranță la erori			Semantica invocarii
Retransmiterea mesajului de solicitare	Duplicare de filtrare	Reexecutarea procedurii sau retransmiterea răspunsului	
Nu	Nu se aplică	Nu se aplică	Ar putea fi
da	Nu	Reluați procedura	Cel puțin o dată
da	da	Răspunsul releu	Cel mult o dată

# Implementarea RMI



# Implementarea RMI

---

- Modul de comunicare
  - Ei execută protocolul cerere-răspuns între C/S
  - Furnizați semantică de invocare
- Modul de referință la distanță
  - Traduceți referințe între obiecte locale și la distanță
  - Creați referințe la obiecte aflate la distanță
  - În fiecare proces, acest modul are un tabel de obiecte la distanță ăcorespondență dintre referințele la obiecte locale din acel proces și referințele la obiecte la distanță
    - Când obiectul de la distanță este transmis pentru prima oară (argument/rezultat) acestui modul î se cere să creeze o referință la obiectul de la distanță (adăugându-l la tabel)
    - Când sosește o referință la un obiect la distanță (mesaj de cerere/răspuns), acestui modul î se cere să traducă în referință locală corespunzătoare (proxy sau obiect la distanță)

# Implementarea RMI

---

- Software RMI
  - Stratul software = middleware
  - Proxy: invocarea metodei de la distanță apare locală
    - Ascundeți detaliile snap-ului de la distanță
    - Ambalarea/despachetarea argumentelor/rezultatelor
    - Trimiterea/primirea mesajelor de la/către client
    - Un proxy pentru fiecare obiect la distanță la care clientul are o referință la obiect la distanță
  - Distribuitor: primește mesajul de solicitare de la modul comunicare, folosește idMethod pentru a selecta metoda corespunzătoare din schelet, transmîndu-i mesajul de solicitare
  - Skeleton: implementează metodele interfeței de la distanță
    - Despachetează argumentele mesajului de cerere și apelează metoda corespunzătoare a obiectului de la distanță, așteaptă rezultatul, pachetele rezultă într-un mesaj de răspuns

# Implementarea RMI

---

- Generarea de clase pentru fiecare proxy, distribuitor și schelet
  - Automat prin compilatoare de interfață
- Programe client și server
  - Server: conține clasele pentru distribuitori și schelete și implementări ale claselor tuturor obiectelor la distanță pe care le suportă
    - Inițializare (creați și porniți obiecte la distanță)
    - Înregistrare în legătură
  - Client: va conține clasele fiecărui proxy pentru toate obiectele la distanță pe care le invocă (puteți folosi bind pentru a căuta referințe la distanță)
- Linker (legare)
  - Serviciu care suportă un tabel care conține relații cu nume textuale și referințe la obiecte la distanță (registre de server, căutări de clienți)