



Università degli Studi dell'Insubria
Dipartimento di Scienze Teoriche e Applicate

Programmazione Concorrente e Distribuita Socket

Luigi Lavazza

Dipartimento di Scienze Teoriche e Applicate

luigi.lavazza@uninsubria.it



Applicazioni distribuite

- Applicazione: un insieme di programmi coordinati per svolgere una data funzione.
- Un'applicazione è distribuita se prevede più programmi eseguiti (o processi) su differenti calcolatori connessi tramite una rete.
 - ▶ Es: Web Browser (Firefox, IE, Chrome, Safari, Opera ...) e Web Server (server http)



Protocollo applicativo

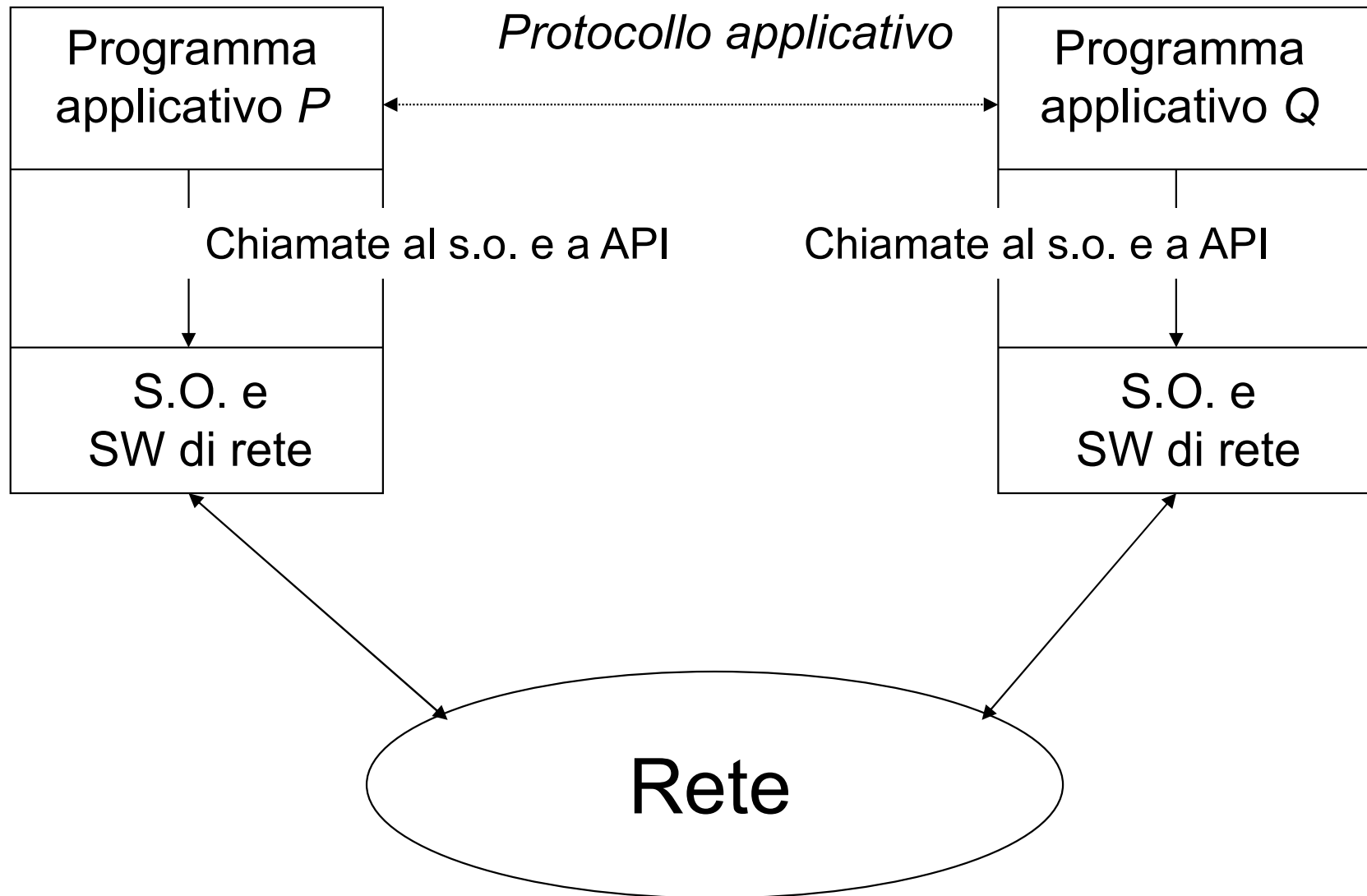
- Le regole per la comunicazione in una applicazione distribuita sono dette protocollo applicativo.
 - ▶ Es. il protocollo applicativo della navigazione Web è detto HyperText Transfer Protocol - HTTP.
- Il protocollo applicativo deve essere definito opportunamente e comune a tutti i programmi dell'applicazione.
 - ▶ Es. ogni messaggio scambiato è terminato dalla stringa “\0 \0 \0”.



Interfacce e protocolli

- I programmi applicativi utilizzano opportune interfacce (API - Application Programming Interface), fornite dal sistema operativo e dal software di rete, per accedere ai servizi di comunicazione
 - ▶ Nascondono i dettagli dei livelli inferiori.
- Il protocollo applicativo rappresenta le regole di comunicazione, e considera il contenuto della comunicazione.
 - ▶ Realizzabile usando, attraverso API, i servizi disponibili

Interfacce e protocolli



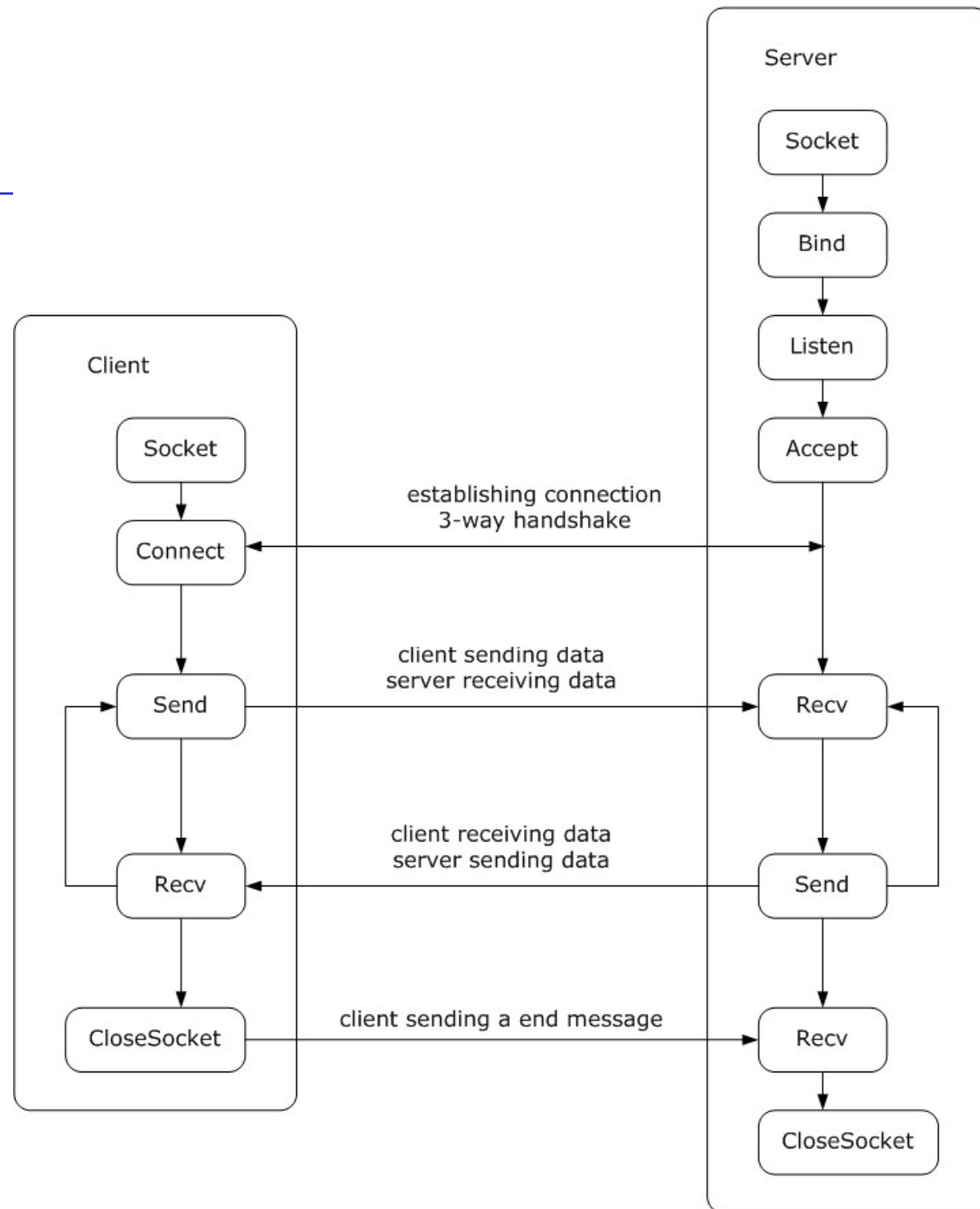


Programmare la comunicazione via API

- Fortunatamente esiste uno standard de facto per usare TCP/IP (e UDP/IP) : l'interfaccia **socket**.
 - ▶ Un'API (Application Programming Interface)
 - ▶ Disponibile per i linguaggi C, Java e molti altri

Socket

- I Socket di Berkeley hanno origine dal sistema operativo Unix BSD 4.2 (rilasciato nel 1983) come una API.
- Tutti i moderni sistemi operativi ne hanno ora almeno un'implementazione: l'interfaccia socket di Berkeley è di fatto diventata l'interfaccia standard per la connessione a Internet.
- TCP sockets flow diagram ⇒



API socket

- L'interfaccia (API) socket è disponibile su Unix, Windows e altre piattaforme
- La disponibilità universale dell'interfaccia socket rende possibile la **portabilità** dei programmi di rete
- Due applicazioni basate su uno stesso protocollo di Trasporto possono interagire
 - ▶ ad esempio un'applicazione Java che utilizza TCP tramite l'API socket-Java può interagire senza problemi con una applicazione C che utilizza TCP tramite l'API socket C.
- Due applicazioni basate su diversi protocolli di Trasporto non possono interagire: ad esempio un'applicazione che utilizza UDP non può interagire con una applicazione che utilizza TCP.
- In effetti è insito nella nozione di protocollo che un insieme di programmi usino lo stesso protocollo.



- 172.16.254.1



172 . 16 . 254 . 1

10101100.00010000.11111110.00000001

Un byte = otto bit

Trenta-due bit ($4 * 8$), o 4 byte

- Attribuito da DNS (Domain Name System)



Come identificare l'indirizzo IP dal nome di dominio in Java

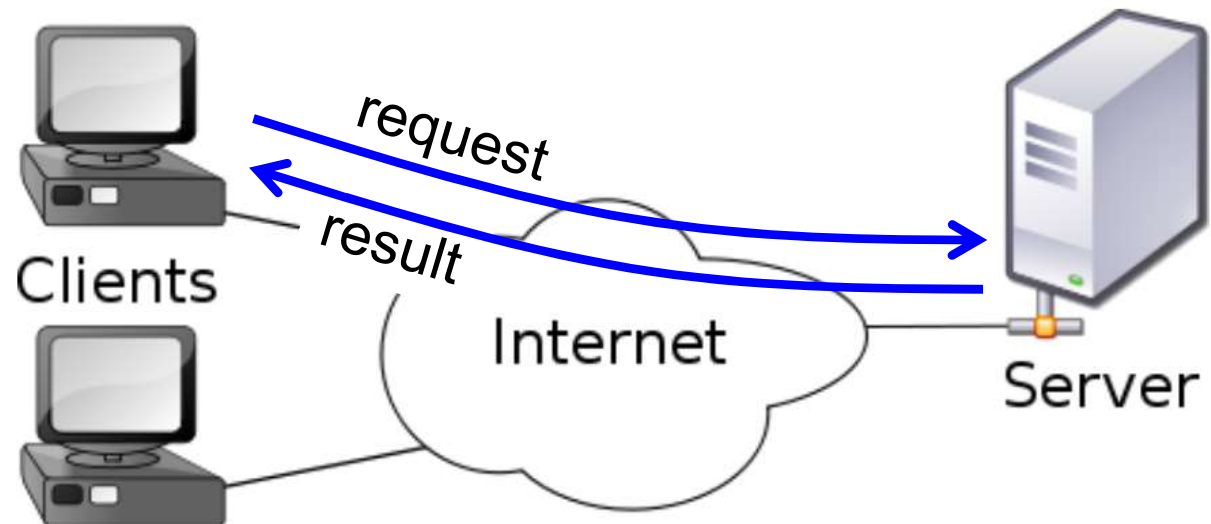
- L'indirizzo IP si può ottenere con `InetAddress.getByName()`
 - ▶ utilizzabile poi per costruire un socket

```
import java.net.*;
public class IPAddressSolver{
    public static void main(String[] args) throws Exception {
        if (args.length!=1) {
            System.err.println("Usage: IPAddressSolver MachineName");
            System.exit(1);
        }
        InetAddress a=InetAddress.getByName(args[0]);
        System.out.println(a);
    }
}
```

```
Scrivendo: java IPAddressSolver google.com
ottengo: google.com/216.58.208.174
scrivendo: java IPAddressSolver www.cefriel.com
ottengo: www.cefriel.com/151.101.1.195
```

Client e Server in Java

- Il server deve rimanere in ascolto di richieste di connessione, e questo viene fatto dall'oggetto speciale **ServerSocket**.
- Il client cerca di stabilire una connessione con un server usando un oggetto di tipo **Socket**.
- Effettuata la connessione, questa viene trasformata in un **I/O stream**, e da allora in poi si può trattare la connessione come se si stesse leggendo da e scrivendo su un file.
- Le regole con cui scrivere/leggere sono dettate dai protocolli di livello più alto.





127.0.0.1

- 127.0.0.1 is the loopback Internet protocol (IP) address also referred to as the **localhost**.
- The address is used to establish an IP connection to the same machine or computer being used by the end-user.
- Utile per fare prove, quando non si hanno diverse macchine in rete.
- Infatti i processi che comunicano possono stare sulla stessa macchina
- In effetti i socket sono comunemente usati anche per fare inter-process communication in locale.



Client e Server: test senza rete

- Per molte ragioni, si potrebbe non avere una macchina client, una macchina server e una rete a disposizione per testare i programmi
- In questo caso si può usare l'indirizzo speciale localhost che corrisponde all'indirizzo IP "127.0.0.1" pensato per test senza la rete.
- Queste tre istruzioni producono tutte lo stesso risultato.

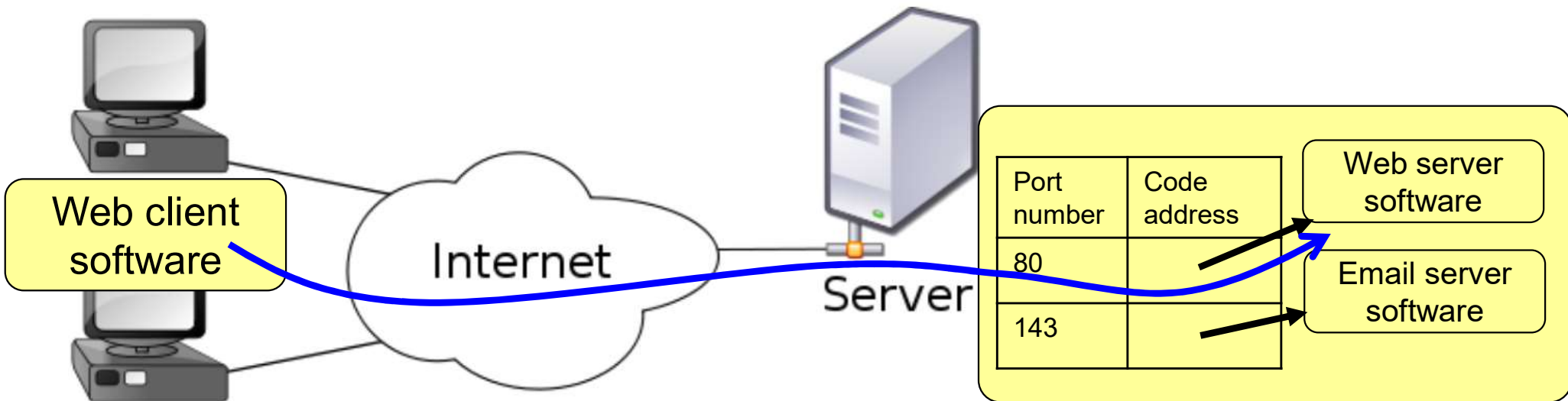
```
InetAddress addr=InetAddress.getByName(null) ;
```

```
InetAddress addr=InetAddress.getByName("localhost") ;
```

```
InetAddress addr=InetAddress.getByName("127.0.0.1") ;
```

Client e Server: l'indirizzo IP non basta!

- Un **indirizzo IP** **identifica una macchina**, **ma** non è sufficiente per identificare un processo server in modo univoco, visto che su una sola macchina possono essere attivi molti processi server.
- Ogni macchina IP contiene anche le **porte**, e quando si sta impostando un client o un server è necessario scegliere una porta attraverso la quale il client e il server si connettono.
- Attenzione: le porte da 1 a 1024 solitamente sono riservate dal sistema





Indirizzamento in Java

- java.net fornisce le seguenti classi di indirizzamento correlate:
- **InetAddress**: indirizzo IP
 - ▶ Inet4Address: indirizzo a 32-bit IPv4 (circa 4.3×10^9)
 - ▶ Inet6Address: indirizzo a 128-bit IPv6 (circa 3.4×10^{38})
- **SocketAddress**: indirizzo Socket
 - ▶ InetSocketAddress: IP Socket Address (indirizzo IP + porta)



Socket (presa)

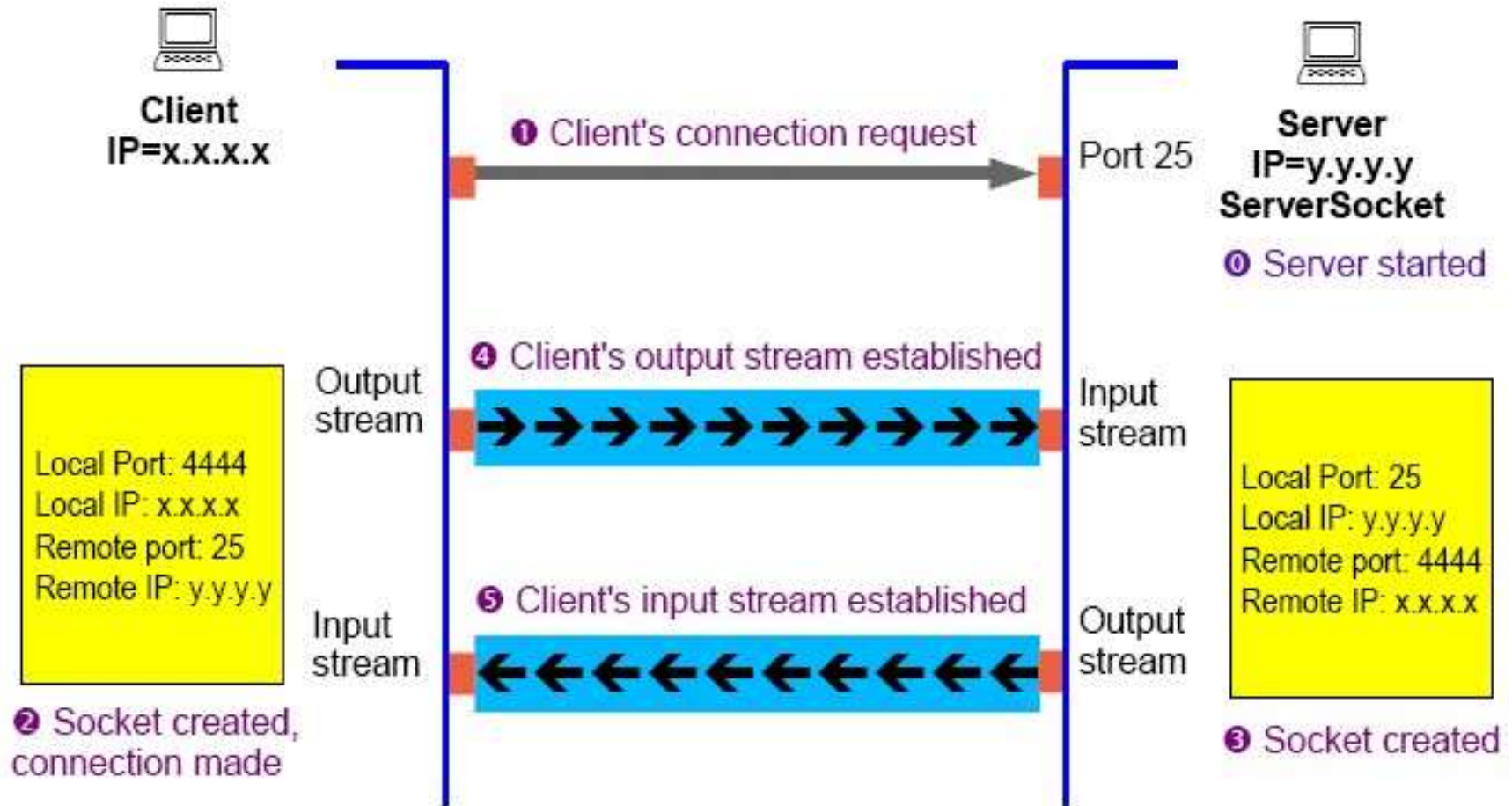
- Un socket rappresenta il “terminale” di un collegamento tra due macchine.
- Per una determinata connessione, c'è un socket su ogni macchina
- Si può immaginare un ipotetico “cavo” (InputStream/OutputStream) tra le due macchine con ciascuna estremità collegata ad un “socket” che persiste finché non viene esplicitamente disconnesso.



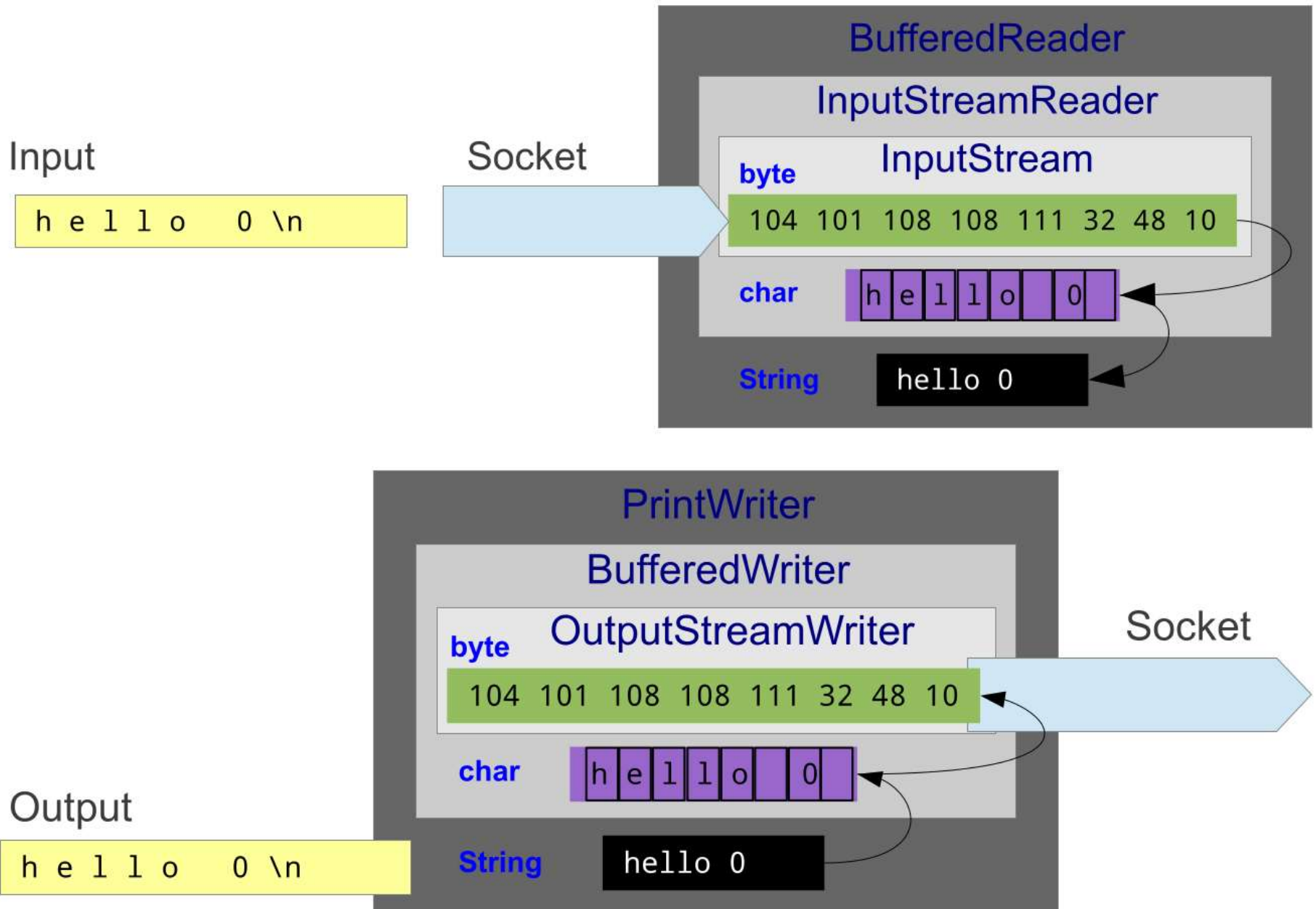
Socket

- In Java esistono due classi di socket stream-based:
 - ▶ **ServerSocket**, che un **server** utilizza per "ascoltare" le connessioni in ingresso
 - ▶ **Socket**, che un **client** utilizza, al fine di avviare una connessione.
- **ServerSocket** usa il metodo **accept()** per ritornare un **Socket** nel momento in cui il client si connette
- Dopo di che si ha una connessione Socket-to-Socket
- A questo punto, è possibile utilizzare i metodi **getInputStream()** e **getOutputStream()** su ogni **Socket**.
- Quando si crea un **ServerSocket**, si dà solo il numero di porta su cui si accetteranno le connessioni.
- Quando si crea un **Socket** è necessario dare sia l'indirizzo IP che il numero di porta in cui si sta tentando di connettersi.

Stream Socket overview



InputStream e OutputStream

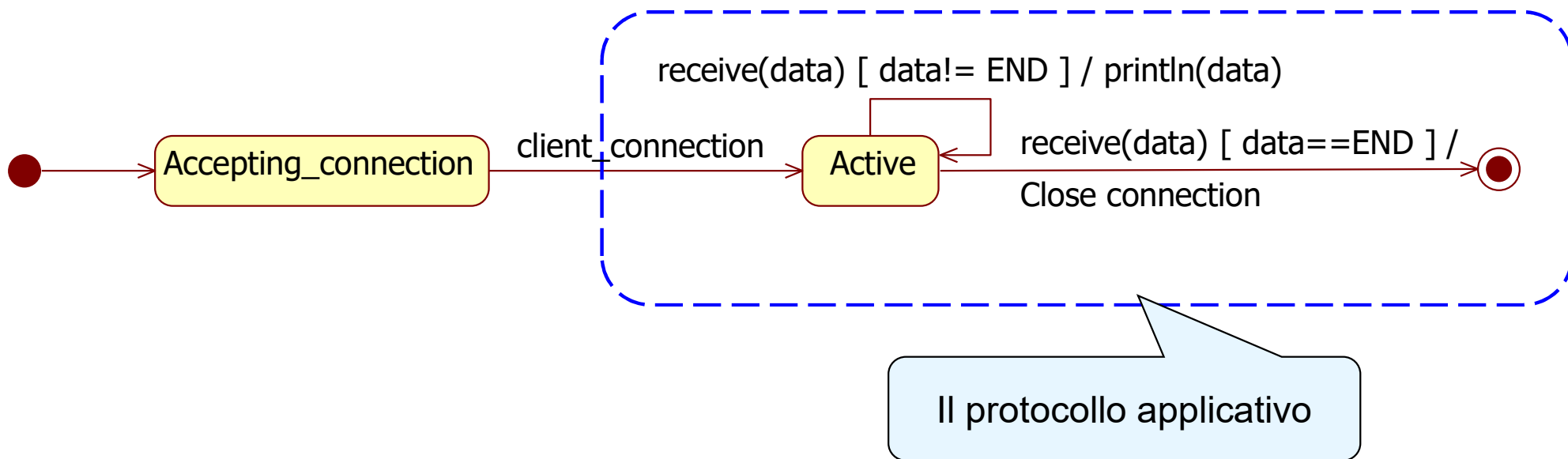




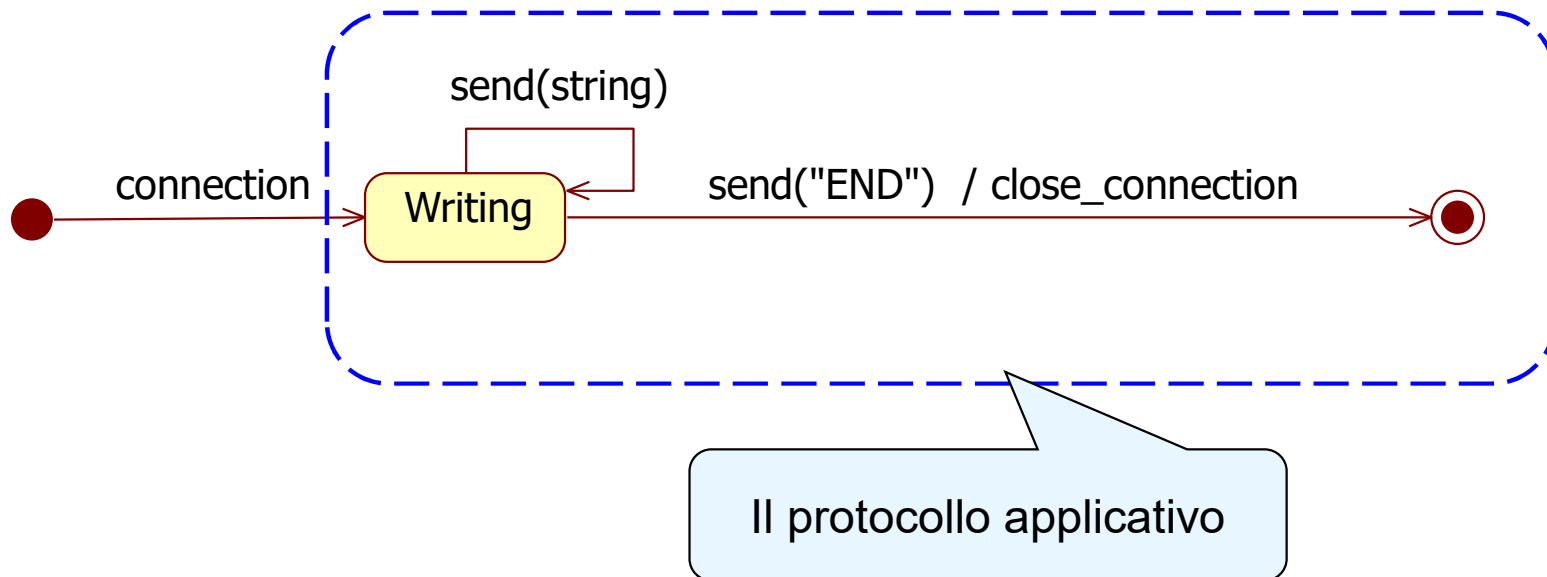
Esempio

- Programmiamo una semplice applicazione distribuita
- Il client
 1. Si connette al server
 2. Manda al server una sequenza di stringhe, terminate dalla stringa “END”
 3. Si sconnette
- Il server
 1. Accetta connessioni
 2. Iterativamente:
 3. Riceve una stringa
 - Se la stringa ricevuta non è “END” la visualizza
 - Se la stringa ricevuta è “END” esce dal ciclo
 4. Chiude la connessione

Server: stati e protocollo applicativo



Client: stati e protocollo applicativo





Server

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

public static void main(String[] args) throws IOException {
    new ExServer().exec();
}
```



Server

```
public class ExServer {
    public static final int PORT=8080;
    void exec() throws IOException {
        ServerSocket s = new ServerSocket(PORT);
        System.out.println("Server started: "+s);
        try {
            Socket mySocket=s.accept();
            try {
                System.out.println("Server: conn. accepted: "+mySocket);
                BufferedReader in=
                    new BufferedReader(new InputStreamReader(
                        mySocket.getInputStream()));
                while(true) {
                    String str=in.readLine();
                    if(str.equals("END")) break;
                    System.out.println("Server received:"+str);
                }
            }
            finally { mySocket.close(); }
        } finally { s.close(); }
    }
}
```



Client

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

public static void main(String[] args) throws IOException {
    new ExClient().exec();
}
```



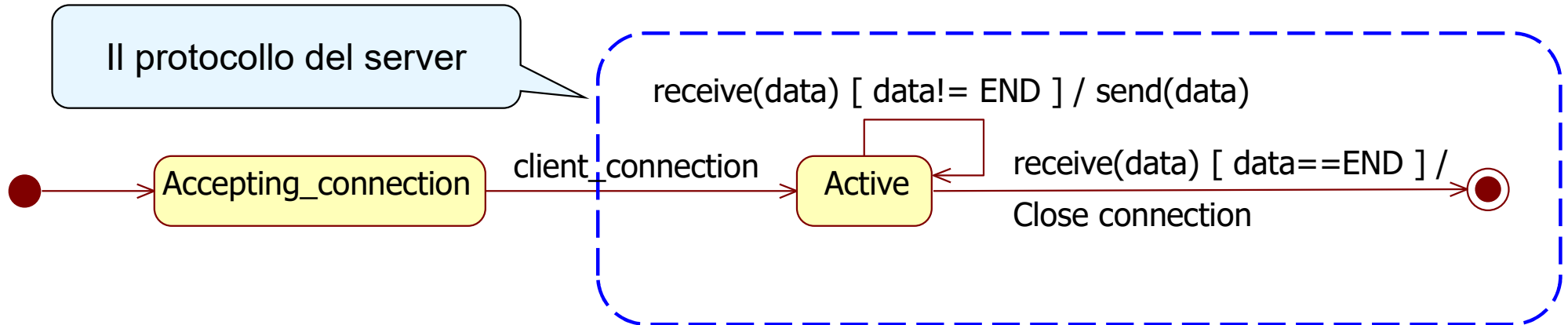
Client

La connessione potrebbe non andare a buon fine

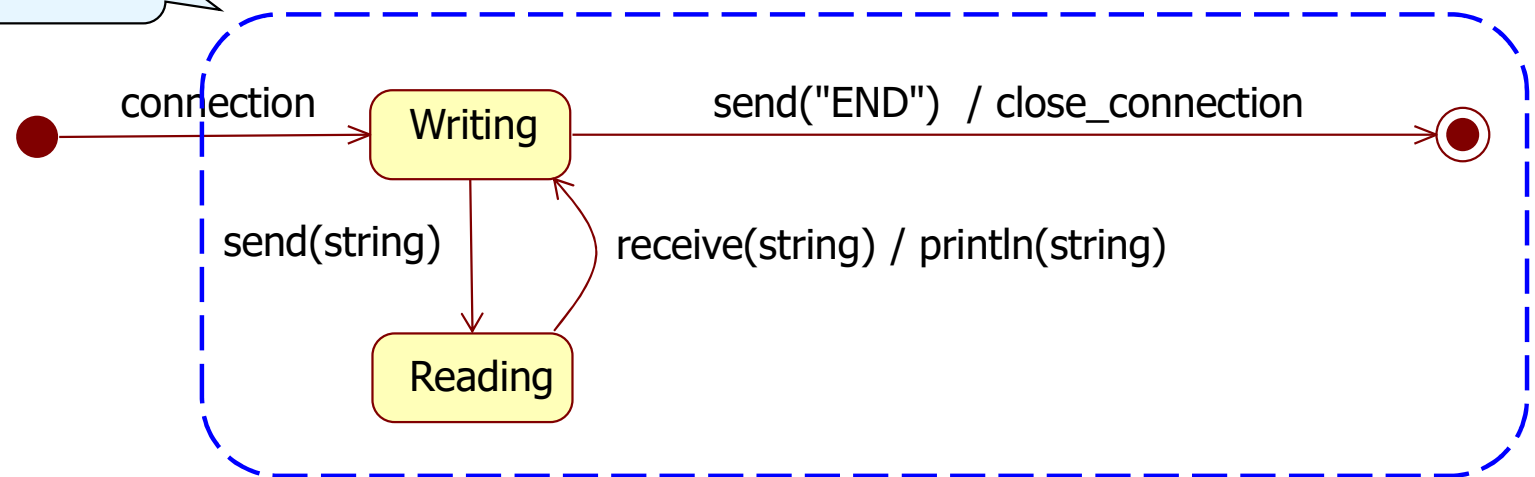
```
public class ExClient {
    void exec() throws IOException {
        InetAddress addr = InetAddress.getByName(null);
        System.out.println("addr = " + addr);
        Socket socket = new Socket(addr, 8080);
        try {
            System.out.println("Client connected: " + socket);
            PrintWriter out = new PrintWriter(new BufferedWriter(
                new OutputStreamWriter(socket.getOutputStream())), true);
            for (int i = 0; i < 10; i++) {
                System.out.println("Client sends hello_" + i);
                out.println("hello_" + i);
            }
            out.println("END");
        } finally {
            System.out.println("Client: closing...");
            socket.close();
        }
    }
}
```

Un semplice Server che ritorna qualunque cosa arrivi dal client (servizio Echo)

Il protocollo del server



Il protocollo del client





Echo server

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

public class EchoServer {
    public static final int PORT=8080;
    public static void main(String[] args) throws IOException {
        new EchoServer().exec();
    }
    private void exec() throws IOException {
        // next slide
    }
}
```



Echo server

```
private void exec() throws IOException {  
    ServerSocket s = new ServerSocket(PORT);  
    System.out.println("Server started: "+s);  
    try {  
        Socket mySocket=s.accept();  
        try {  
            System.out.println("Server: conn. accepted: "+mySocket);  
            BufferedReader in=  
                new BufferedReader(new InputStreamReader(  
                    mySocket.getInputStream()));  
            PrintWriter out=new PrintWriter(new BufferedWriter(  
                new OutputStreamWriter(mySocket.getOutputStream()), true);  
            while(true) {  
                String str=in.readLine();  
                if(str.equals("END")) break;  
                System.out.println("Server echoing:"+str);  
                out.println(str);  
            }  
        }  
        finally { mySocket.close(); }  
    } finally { s.close(); }  
}
```



Echo client

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

public class EchoClient {
    public static void main(String[] args) throws IOException {
        new EchoClient().exec();
    }
    private void exec() throws IOException {
        // next slide
    }
}
```



Echo client

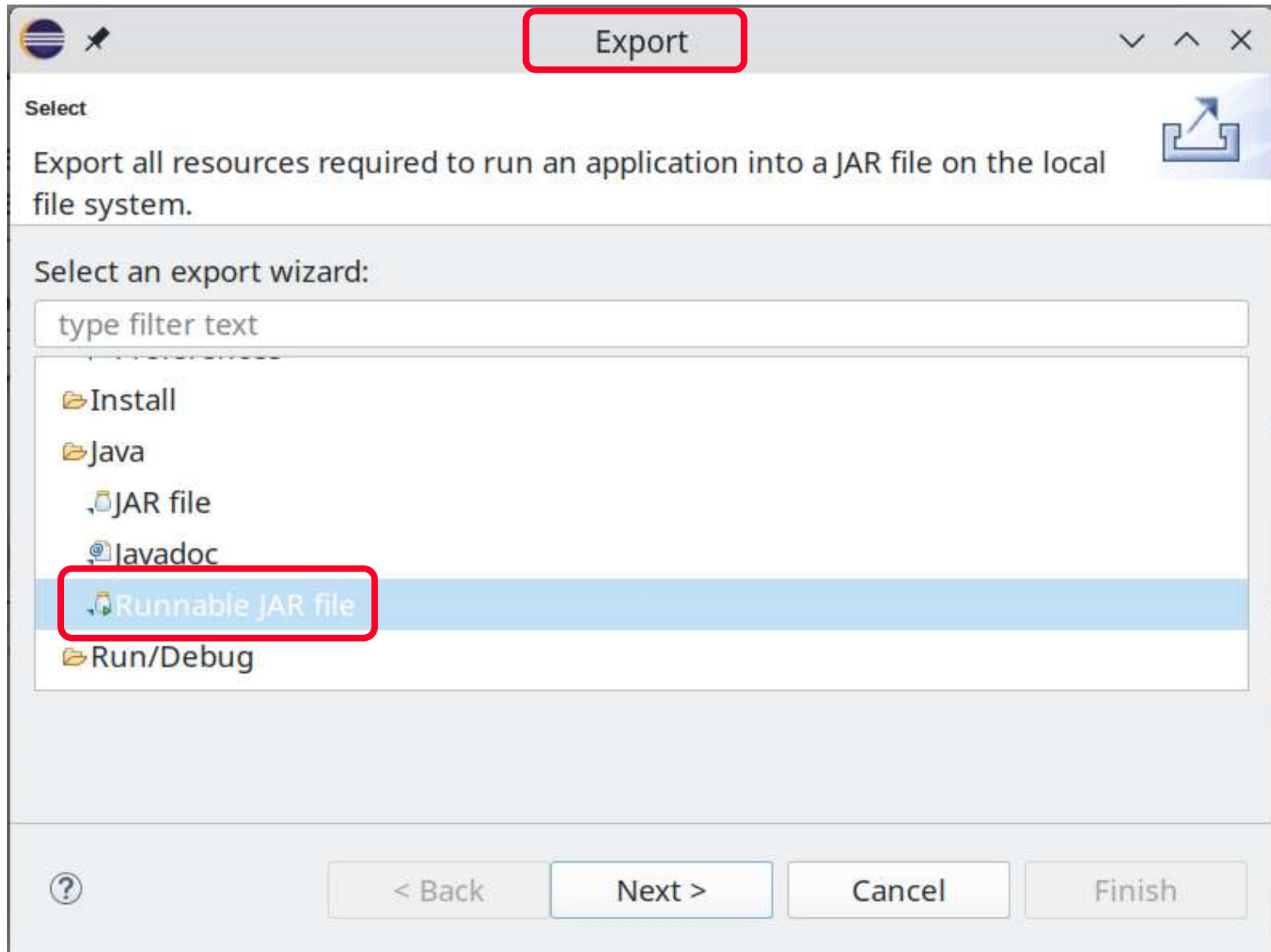
```
private void exec() throws IOException {
    String str;
    InetAddress addr = InetAddress.getByName(null);
    System.out.println("Client found addr = " + addr);
    Socket socket = new Socket(addr, 8080);
    try {
        System.out.println("Client connected via " + socket);
        BufferedReader in = new BufferedReader(
            new InputStreamReader(socket.getInputStream()));
        PrintWriter out = new PrintWriter(new BufferedWriter(
            new OutputStreamWriter(socket.getOutputStream())), true);
        for (int i = 0; i < 10; i++) {
            str="Hello_" + i;
            out.println(str);
            System.out.println("Client sent to server:"+str);
            str = in.readLine();
            System.out.println("Client received from server:"+str);
        }
        out.println("END");
    } finally { socket.close(); }
}
```



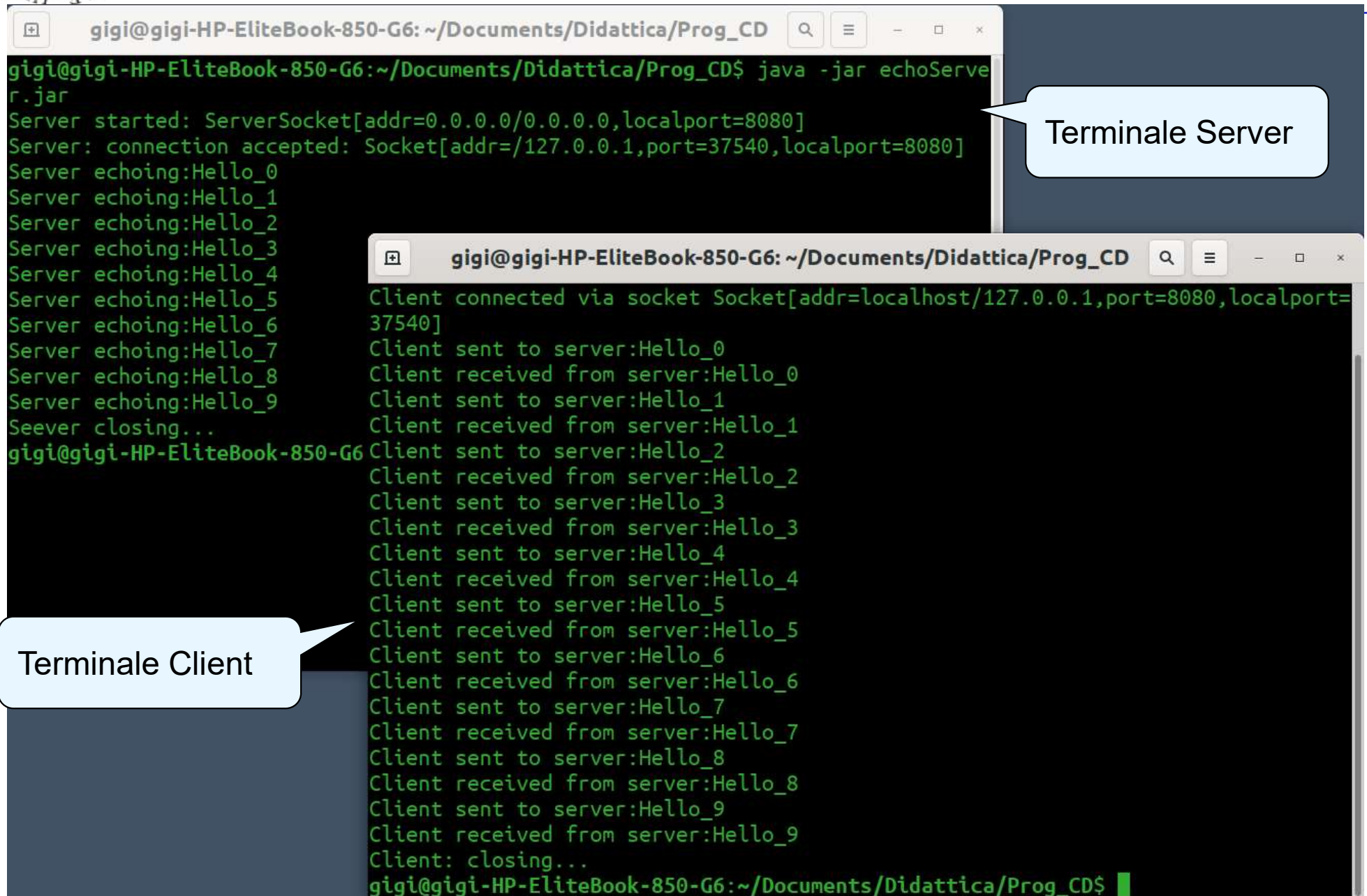
Esecuzione

- Client e server si possono eseguire dentro a Eclipse, oppure
- Si possono creare degli eseguibili da lanciare da terminali virtuali.
 - ▶ Vediamo come

Esportiamo dei jar eseguibili



Esecuzione in terminale



The image shows two terminal windows. The top window, titled 'gigi@gigi-HP-EliteBook-850-G6: ~/Documents/Didattica/Prog_CD', shows the execution of a Java server program. It starts with the command 'java -jar echoServer.jar'. The output shows the server starting on port 8080, accepting a connection from 127.0.0.1 on port 37540, and then echoing back messages 'Hello_0' through 'Hello_9' before closing. A callout bubble labeled 'Terminale Server' points to this window. The bottom window, also titled 'gigi@gigi-HP-EliteBook-850-G6: ~/Documents/Didattica/Prog_CD', shows the execution of a Java client program. It starts with the output 'Client connected via socket Socket[addr=localhost/127.0.0.1,port=8080,localport=37540]'. Then, it sends messages 'Hello_0' through 'Hello_9' to the server and receives the same messages back. Finally, it outputs 'Client: closing...'. A callout bubble labeled 'Terminale Client' points to this window.

```
gigi@gigi-HP-EliteBook-850-G6: ~/Documents/Didattica/Prog_CD$ java -jar echoServer.jar
Server started: ServerSocket[addr=0.0.0.0/0.0.0.0,localport=8080]
Server: connection accepted: Socket[addr=/127.0.0.1,port=37540,localport=8080]
Server echoing:Hello_0
Server echoing:Hello_1
Server echoing:Hello_2
Server echoing:Hello_3
Server echoing:Hello_4
Server echoing:Hello_5
Server echoing:Hello_6
Server echoing:Hello_7
Server echoing:Hello_8
Server echoing:Hello_9
Server closing...
gigi@gigi-HP-EliteBook-850-G6: ~/Documents/Didattica/Prog_CD$

gigi@gigi-HP-EliteBook-850-G6: ~/Documents/Didattica/Prog_CD$
Client connected via socket Socket[addr=localhost/127.0.0.1,port=8080,localport=37540]
Client sent to server:Hello_0
Client received from server:Hello_0
Client sent to server:Hello_1
Client received from server:Hello_1
Client sent to server:Hello_2
Client received from server:Hello_2
Client sent to server:Hello_3
Client received from server:Hello_3
Client sent to server:Hello_4
Client received from server:Hello_4
Client sent to server:Hello_5
Client received from server:Hello_5
Client sent to server:Hello_6
Client received from server:Hello_6
Client sent to server:Hello_7
Client received from server:Hello_7
Client sent to server:Hello_8
Client received from server:Hello_8
Client sent to server:Hello_9
Client received from server:Hello_9
Client: closing...
gigi@gigi-HP-EliteBook-850-G6: ~/Documents/Didattica/Prog_CD$
```



Daytime Server

- Un esempio di Server Socket che in un loop infinito accetta delle connessioni Socket per inviare la data corrente al Client.



Daytime Server

```
import java.io.*;
import java.net.*;
import java.util.*;
public class DaytimeServer {
    public final static int DayTime_PORT= 1333;
    ServerSocket server;
    Socket connection = null;
    DaytimeServer() throws IOException{
        server = new ServerSocket(DayTime_PORT);
    }
    public static void main(String[] args) {
        DaytimeServer srv;
        try {
            srv = new DaytimeServer();
            srv.exec();
        } catch (IOException e) {
            System.out.println("Server start failed.");
        }
    }
}
```



Daytime Server

```
private void exec() {  
    while (true) {  
        try {  
            connection = server.accept();  
            Writer out =  
                new OutputStreamWriter(connection.getOutputStream());  
            Date now = new Date();  
            out.write(now.toString() + "\r\n");  
            out.flush();  
            connection.close();  
        }  
        catch (IOException ex) {}  
        finally {  
            try {  
                if (connection != null) connection.close();  
            } catch (IOException ex) {}  
        }  
    }  
}
```

Fallisce se il client ha già chiuso la connessione



Daytime client

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

public class DaytimeClient {
    InetAddress addr;
    Socket connection;
    BufferedReader in;
    DaytimeClient(String servName) throws IOException{
        addr = InetAddress.getByName(servName);
        System.out.println("addr = " + addr);
        connection = new Socket(addr, DaytimeServer.DayTime_PORT);
        System.out.println("socket = " + connection);
        in = new BufferedReader(
            new InputStreamReader(connection.getInputStream()));
    }
    // main ed exec nella prossima slide
}
```



Daytime client

```
private void exec() {
    String str;
    try {
        str = in.readLine();
        System.out.println(str);
    } catch (IOException e) {
        System.out.println("Daytime reading failed");
    }
    finally {
        try { connection.close(); } catch (IOException e) {}
    }
}

public static void main(String[] args) {
    DaytimeClient cli;
    try {
        cli = new DaytimeClient("localhost");
        cli.exec();
    } catch (IOException e) {
        System.out.println("Client: connection failed");
    }
}
```



DayTime: client output

addr = localhost/127.0.0.1

socket = Socket[addr=localhost/127.0.0.1,port=1333,localport=52696]

Wed Apr 14 12:27:59 CEST 2021

closing...



Transmission Control Protocol (TCP)

- Gli esempi visti fino ad ora utilizzano TCP, progettato per la massima affidabilità e garantisce che i dati arrivino a destinazione.
- Le caratteristiche di TCP sono
 - ▶ consentire la ritrasmissione dei dati persi
 - ▶ fornire percorsi multipli attraverso diversi router nel caso in cui uno di questi diventi indisponibile
 - ▶ consegna dei byte nell'ordine in cui sono stati inviati.
- Per garantire tutto questo controllo e affidabilità TCP ha un grande **overhead** come costo da pagare.
- Si può usare un secondo protocollo, chiamato User Datagram Protocol (UDP), che non garantisce che i pacchetti saranno consegnati e nemmeno che arriveranno nell'ordine in cui sono stati inviati.



TCP

vs.

UDP

-
- Clients and servers that communicate via a TCP socket, have a **dedicated** point-to-point **channel**.
 - To communicate, the parties
 - ▶ establish a **connection**,
 - ▶ transmit the data, and then
 - ▶ close the connection.
 - All data sent over the channel is received in the same **order** in which it was sent. This is guaranteed by the channel.
- In contrast, applications that communicate via datagrams send and receive completely **independent packets** of information.
 - These clients and servers do not have and do not need a dedicated point-to-point channel.
 - The **delivery** of datagrams to their destinations is **not guaranteed**.
 - Nor is the order of their arrival.



User Datagram Protocol (UDP)

- **Datagram definition**
A datagram is an independent, self-contained message sent over the network whose arrival, arrival time, and content are not guaranteed.
- The `java.net` package contains three classes to help you write Java programs that use datagrams to send and receive packets over the network:
 - **`DatagramSocket`, `DatagramPacket`**: An application can send and receive `DatagramPackets` through a `DatagramSocket`.
 - **`MulticastSocket`**: `DatagramPackets` can be broadcast to multiple recipients all listening to a `MulticastSocket`.



Esempio UDP

- Il server
 - ▶ riceve un pacchetto di dati dal client,
 - ▶ ne estrae una stringa e la mostra a video
 - ▶ la rimanda al client, dopo averla convertita in lettere maiuscole
- Il client:
 - ▶ legge una stringa da terminale
 - ▶ confeziona un pacchetto contenente la stringa e lo spedisce al server
 - ▶ riceve un pacchetto dal server, estrae la stringa contenuta, mostra e termina



Esempio UDP: server

```
import java.io.IOException;
import java.net.*;
public class UDPserver {
    DatagramSocket serverSocket;
    byte[] receiveData;
    byte[] sendData;
    DatagramPacket receivePacket;
    UDPserver() throws SocketException{
        serverSocket = new DatagramSocket(9876);
        receiveData = new byte[1024];
        sendData = new byte[1024];
        receivePacket = new DatagramPacket(receiveData,
                                           receiveData.length);
    }
    // main ed exec nella prossima slide
}
```

Prepara il pacchetto da usare per le ricezioni



Esempio UDP: server

```
private void exec() throws IOException {
    while (true) {
        serverSocket.receive(receivePacket);
        String sentence = new String(receivePacket.getData());
        System.out.println("RECEIVED: " + sentence);
        InetAddress IPAddress = receivePacket.getAddress();
        int port = receivePacket.getPort();
        String capitalizedSentence = sentence.toUpperCase();
        sendData = capitalizedSentence.getBytes();
        serverSocket.send(new DatagramPacket(
            sendData, sendData.length, IPAddress, port));
    }
}

public static void main(String args[]) {
    try {
        UDPserver srv = new UDPserver();
        srv.exec();
    } catch (IOException e) {
        System.out.println("Server failed");
    }
}
```

Riceve il pacchetto

Crea il pacchetto e lo
spedisce



Esempio UDP: client

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

public class UDPclient {
    BufferedReader inFromUser;
    DatagramSocket clientSocket;
    InetAddress IPAddress;
    byte[] sendData;
    byte[] receiveData;
    UDPclient() throws SocketException, UnknownHostException{
        inFromUser = new BufferedReader(
            new InputStreamReader(System.in));
        clientSocket = new DatagramSocket();
        IPAddress = InetAddress.getByName("localhost");
        sendData = new byte[1024];
        receiveData = new byte[1024];
    }
    // main ed exec nella prossima slide
}
```



Esempio UDP: client

```
private void exec() throws IOException {
    System.out.println("type a string: ");
    String sentence = inFromUser.readLine();
    sendData = sentence.getBytes();
    DatagramPacket sendPacket = new DatagramPacket(sendData,
        sendData.length, IPAddress, 9876);
    clientSocket.send(sendPacket);
    DatagramPacket receivePacket =
        new DatagramPacket(receiveData, receiveData.length);
    clientSocket.receive(receivePacket);
    String modifiedSentence=new String(receivePacket.getData());
    System.out.println("FROM SERVER:" + modifiedSentence);
    clientSocket.close();
}

public static void main(String args[]) {
    try {
        UDPclient cli = new UDPclient();
        cli.exec();
    } catch (IOException e) {
        System.out.println("Client failed");
    }
}
```

Prepara il pacchetto
da spedire

Spedisce

Prepara il pacchetto per ricezione

Riceve



Esecuzione (lato client)

type a string:

bubu

FROM SERVER:BUBU

UDP Image Server

- Un esempio di Server Socket che usa il protocollo UDP per inviare un'immagine al Client. Ogni immagine viene scomposta in tanti DatagramPacket da 1024 byte.
- Questo esempio si può estendere all'invio dei frame di un video. . .





UDP Image Server

```
class UDPImageServer {
    public static void main(String args[]) throws Exception {
        DatagramSocket serverSocket = new DatagramSocket(9876);
        while (true) {
            // RECEIVE
            byte[] receiveData = new byte[1024];
            DatagramPacket receivePacket =
                new DatagramPacket(receiveData, receiveData.length);
            System.out.println("waiting ");
            serverSocket.receive(receivePacket);
            String sentence = new String(receivePacket.getData());
            System.out.println("RECEIVED: " + sentence);
            // SEND...
        }
    }
}
```



UDP Image Server

```
class UDPImageServer {
    public static void main(String args[]) throws Exception {
        DatagramSocket serverSocket = new DatagramSocket(9876);
        while (true) {
            // RECEIVE ...
            // SEND
            InetAddress IPAddress = receivePacket.getAddress();
            int port = receivePacket.getPort();
            File file = new File("ange.png");
            System.out.println("Starting to send file: " + file);
            FileInputStream fis = new FileInputStream(file);
            int size = 0;
            while (true) {
                ...
            }
        }
    }
}
```



UDP Image Server

```
while (true) {  
    byte[] sendData = new byte[1024];  
    size = fis.read(sendData);  
    if(size == -1) {  
        byte[] sendEOFData = "END_FILE".getBytes();  
        System.out.println("Send: " + new String(sendEOFData));  
        DatagramPacket sendPacket = new DatagramPacket(  
            sendEOFData, sendEOFData.length, IPAddress, port);  
        serverSocket.send(sendPacket);  
        break;  
    } else {  
        DatagramPacket sendPacket = new DatagramPacket(  
            sendData, size, IPAddress, port);  
        serverSocket.send(sendPacket);  
    }  
    Thread.sleep(1);  
}
```



UDP Image Client

```
class UDPImageClient {
    public static void main(String args[]) throws Exception {
        DatagramSocket clientSocket = new DatagramSocket();
        InetAddress IPAddress = InetAddress.getByName("localhost");
        byte[] sendData;
        sendData = "START".getBytes();
        DatagramPacket sendPacket = new DatagramPacket(
            sendData, sendData.length, IPAddress, 9876);
        clientSocket.send(sendPacket);
        File file = new File("./myCopy.png");
        FileOutputStream fos = new FileOutputStream(file);
    }
}
```



UDP Image Client

```
while (true) {
    byte[] receiveData = new byte[1024];
    DatagramPacket receivePacket =
        new DatagramPacket(receiveData, receiveData.length);
    clientSocket.receive(receivePacket);
    String modifiedSentence =
        new String(receivePacket.getData());
    if (modifiedSentence.startsWith("END_FILE")) {
        System.out.println("RECEIVED FROM SERVER: " +
            modifiedSentence);

        fos.close();
        break;
    }
    fos.write(receivePacket.getData(), 0,
        receivePacket.getLength());
}
clientSocket.close();
System.out.println("CLIENT: finished");
}}
```

SERVER CHE GESTISCONO PIÙ DI UN CLIENT ALLA VOLTA

Server che gestiscono più di un client alla volta

- Gli esempi di Server visti finora sono in grado di gestire un solo client alla volta.
- Un tipico server deve poter gestire diversi client in parallelo.
- Possiamo ottenere questo risultato attraverso il multithreading
 1. si crea un unico **ServerSocket** nel server
 2. si attende una nuova connessione attraverso **accept()**
 3. quando **accept()** ritorna un **Socket** connesso, si crea un nuovo thread il cui compito è “servire” unicamente quel particolare client.
 - A tale scopo si passa al nuovo thread il socket connesso al client.
 - Quando il thread ha finito di servire il suo client termina
 4. Appena creato il thread ci si mette di nuovo in attesa di un nuovo client con **accept()** (tornando al punto 2)



Server che accetta più connessioni contemporanee

```
import java.io.*;
import java.net.*;
public class MultiServer {
    static final int PORT = 8080;
    ServerSocket theServerSocket;
    MultiServer() throws IOException{
        theServerSocket = new ServerSocket(PORT);
        System.out.println("Server Started");
    }
    // exec nella prossima slide
    public static void main(String[] args) {
        try {
            MultiServer multiSrv = new MultiServer();
            multiSrv.exec();
        } catch (IOException e) {
            System.out.println("Server failed");
        }
    }
}
```



Server che accetta più connessioni contemporanee

```
private void exec() throws IOException {
    try {
        while (true) {
            Socket socket = theServerSocket.accept();
            try {
                new ServerSlave(socket);
            } catch (IOException e) {
                // If it fails, close the socket,
                // otherwise the thread will close it:
                socket.close();
            }
        }
    } finally {
        theServerSocket.close();
    }
}
```



Server slave

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

class ServerSlave extends Thread {
    private Socket socket;
    private BufferedReader in;
    private PrintWriter out;
    public ServerSlave(Socket s) throws IOException {
        socket = s;
        in = new BufferedReader(new
            InputStreamReader(socket.getInputStream()));
        out = new PrintWriter(new BufferedWriter(new
            OutputStreamWriter(socket.getOutputStream())), true);
        start();
    }
    // run nella prossima slide
}
```

Il socket ricevuto come
argomento e' già connesso!

auto-flush



Server slave

```
public void run() {
    boolean finito=false;
    try {
        while (!finito) {
            String str = in.readLine();
            if (str.equals("END")) {
                finito=true;
            } else {
                System.out.println("Echoing: " + str);
                out.println(str);
            }
        }
        System.out.println("closing...");
    } catch (IOException e) {
        System.err.println("IO Exception");
    } finally {
        try { socket.close(); }
        catch (IOException e) {
            System.err.println("Socket not closed");
        }
    }
}
```



Client

```
import java.io.*;
import java.net.*;
public class Client {
    InetAddress addr;
    Socket socket;
    Client() {
        try {
            addr = InetAddress.getByName(null);
        } catch (UnknownHostException e) {
            System.err.println("Client: could not get IP address");
            System.exit(0);
        }
        System.out.println("addr = " + addr);
    }
    public static void main(String[] args) {
        new Client().exec(args.length==0?"anonymous":args[0]);
    }
    // exec alla prossima slide
}
```

NB: il client ignora se il server è dedicato o no.



Client

```
void exec(String myName) {
    try {
        socket = new Socket(addr, 8080);
        System.out.println("socket = " + socket);
        BufferedReader in = new BufferedReader(new
            InputStreamReader(socket.getInputStream()));
        PrintWriter out = new PrintWriter(new BufferedWriter(new
            OutputStreamWriter(socket.getOutputStream())), true);
        for (int i = 0; i < 10; i++) {
            Thread.sleep(1500);
            out.println("hello " + i + " from "+myName);
            String str = in.readLine();
            System.out.println(str);
        }
        out.println("END");
    } catch (IOException | InterruptedException e) {
        System.err.println("JabberClient: IO problems");
    }
    System.out.println("closing...");
    try { socket.close(); } catch (IOException e) {}
}
```



Testing di un server multithread

- Per testare il server multi-thread occorre stabilire più connessioni contemporanee.
- Per fare questo basta creare diversi processi che eseguono tutti il client.
 - ▶ Si aprono tanti terminali
 - ▶ In ciascuno si esegue `java Client.class`



Test con più clienti

```
gigi@gigi-HP-EliteBook-850-G6: ~/Documents/Didattica/Prog_CD/Code/2021/Lez08/MultiServer
a MultiServer
Server Started
Echoing: hello 0 from pippo
Echoing: hello 0 from pluto
Echoing: hello 1 from pippo
Echoing: hello 1 from pluto
Echoing: hello 2 from pippo
Echoing: hello 2 from pluto
Echoing: hello 3 from pippo
Echoing: hello 3 from pluto
Echoing: hello 4 from pippo
Echoing: hello 4 from pluto
Echoing: hello 5 from pippo
Echoing: hello 5 from pluto
Echoing: hello 6 from pippo
Echoing: hello 6 from pluto
Echoing: hello 7 from pippo
Echoing: hello 7 from pluto
Echoing: hello 8 from pippo
Echoing: hello 8 from pluto
Echoing: hello 9 from pippo
closing...
Echoing: hello 9 from pluto
closing...
```

```
gigi@gigi-HP-EliteBook-850-G6: ~/Documents/Didattica/Prog_CD/Code/2021/Lez08/MultiServer/Client
gigi@gigi-HP-EliteBook-850-G6: ~/Documents/Didattica/Prog_CD/Code/2021/Lez08/MultiServer/Client$
gigi@gigi-HP-EliteBook-850-G6: ~/Documents/Didattica/Prog_CD/Code/2021/Lez08/MultiServer/Client$ java Client pluto
addr = localhost/127.0.0.1
socket = Socket[addr=localhost/127.0.0.1,port=8080]
hello 0 from pluto
hello 1 from pluto
hello 2 from pluto
hello 3 from pluto
hello 4 from pluto
hello 5 from pluto
hello 6 from pluto
hello 7 from pluto
hello 8 from pluto
hello 9 from pluto
closing...
gigi@gigi-HP-EliteBook-850-G6: ~/Documents/Didattica/Prog_CD/Code/2021/Lez08/MultiServer/Client$
```

```
gigi@gigi-HP-EliteBook-850-G6: ~/Documents/Didattica/Prog_CD/Code/2021/Lez08/MultiServer/Client
gigi@gigi-HP-EliteBook-850-G6: ~/Documents/Didattica/Prog_CD/Code/2021/Lez08/MultiServer/Client$ java Client pippo
addr = localhost/127.0.0.1
socket = Socket[addr=localhost/127.0.0.1,port=8080]
hello 0 from pippo
hello 1 from pippo
hello 2 from pippo
hello 3 from pippo
hello 4 from pippo
hello 5 from pippo
hello 6 from pippo
hello 7 from pippo
hello 8 from pippo
hello 9 from pippo
closing...
gigi@gigi-HP-EliteBook-850-G6: ~/Documents/Didattica/Prog_CD/Code/2021/Lez08/MultiServer/Client$
```

Client multipli per testare il server

- La creazione manuale di diversi processi client può essere scomoda.
- Si può pensare di automatizzare il test creando un programma multi-thread in cui ogni thread si comporta come un client che si collega al server da testare.
 - ▶ Questa modalità è meno realistica della precedente, perché i diversi processi client potrebbero trovarsi su macchine diverse, mentre i diversi thread usati per il testing eseguono necessariamente sulla medesima macchina.





Client multi-thread per testare il server

- Ogni client è un **Thread** che crea un **Socket** per connettersi con lo stesso server usando **InetAddress**
- Ogni thread ha una durata limitata e alla fine chiude il **Socket**.
- Una costante **MAX_THREADS** nel main verrà usata per limitare il numero massimo di Thread contemporaneamente in esecuzione.
 - ▶ Cambiando il valore di questa costante si può vedere dove il vostro sistema comincia ad avere problemi per troppe connessioni.



Multi-Client per più connessioni

```
import java.net.*;
import java.util.concurrent.*;

public class MultiClient {
    static final int MAX_THREADS = 4;
    public static void main(String[] args) {
        try {
            new MultiClient().exec();
        } catch (UnknownHostException e) {
            System.out.println("multi client failed");
        }
    }
    // exec alla prossima slide
}
```



Multi-Client per più connessioni

```
void exec() throws UnknownHostException {
    int sleepTime=0;
    int threadTotalCount=0;
    InetAddress addr = InetAddress.getByName(null);
    while (threadTotalCount<20) {
        if (Client.threadCount() < MAX_THREADS) {
            new Client(addr, threadTotalCount++);
            sleepTime=100;
        } else {
            sleepTime=20;
        }
        try {
            Thread.sleep(ThreadLocalRandom.current().
                                nextInt(sleepTime));
        } catch (InterruptedException e) { }
    }
}
```



Multi-Client per più connessioni

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

public class Client extends Thread {
    Socket socket;
    BufferedReader in;
    PrintWriter out;
    private int id;
    private static int threadcount = 0; // client correnti
    public static int threadCount() {
        return threadcount;
    }
    static synchronized void addToThreadCount(int d) {
        threadcount+=d;
    }
}
```



Multi-Client per più connessioni

```
Client(InetAddress addr, int n) {
    id = n;
    System.out.println("Making client " + id);
    addToThreadCount(1);
    try {
        socket = new Socket(addr, MultiServer.PORT);
    } catch (IOException e) {System.err.println("Socket failed");}
    try {
        in = new BufferedReader(
            new InputStreamReader(socket.getInputStream()));
        out = new PrintWriter(new BufferedWriter(
            new OutputStreamWriter(socket.getOutputStream())), true);
        start();
    } catch (IOException e) {
        System.err.println("Client thread "+id+" failure");
        try {
            socket.close();
        } catch (IOException e2) {
            System.err.println("Socket not closed");
        }
    }
}
```




Multi-Client per più connessioni

```
public void run() {
    try {
        for (int i = 0; i < 10; i++) {
            out.println("Client " + id + ": " + i);
            try { Thread.sleep(
                ThreadLocalRandom.current().nextInt(200, 800));
            } catch (InterruptedException e) { }
            String str = in.readLine();
            System.out.println(str);
        }
        out.println("END");
    } catch (IOException e) {
        System.err.println("IO Exception");
    } finally { // Always close it:
        try { socket.close();
        } catch (IOException e) {
            System.err.println("Socket not closed");
        }
        addToThreadCount(-1);
    }
}
```




Output del Client e del Server

● Client:

```
Making client 0
Making client 1
Making client 2
Making client 3
Client 3: 0
Client 0: 0
Client 1: 0
Client 2: 0
Client 1: 1
Client 2: 1
Client 0: 1
Client 3: 1
Client 2: 2
Client 3: 2
Client 1: 2
Client 0: 2
Client 1: 3
Client 2: 3
Client 3: 3
Client 2: 4
Client 0: 3
Client 1: 4
Client 3: 4
Client 0: 4
```

● Server:

```
Server Started
Echoing: Client 0: 0
Echoing: Client 1: 0
Echoing: Client 2: 0
Echoing: Client 3: 0
Echoing: Client 3: 1
Echoing: Client 0: 1
Echoing: Client 1: 1
Echoing: Client 2: 1
Echoing: Client 1: 2
Echoing: Client 2: 2
Echoing: Client 0: 2
Echoing: Client 3: 2
Echoing: Client 2: 3
Echoing: Client 3: 3
Echoing: Client 1: 3
Echoing: Client 0: 3
Echoing: Client 1: 4
Echoing: Client 2: 4
Echoing: Client 3: 4
Echoing: Client 2: 5
Echoing: Client 0: 4
```

Caratteristiche del programma appena visto

- Nel **main** della classe **MultiClient** la creazione di un nuovo Thread può essere rallentata per più o meno millisecondi.
- Ad esempio
 - ▶ **Thread.currentThread().sleep(1000)**: pochissimi Thread client saranno in esecuzione contemporaneamente
 - ▶ **Thread.currentThread().sleep(1)**: molti Thread client potrebbero essere in esecuzione contemporaneamente
 - ▶ **//Thread.currentThread().sleep(1)**: sicuramente viene raggiunto il numero massimo di Thread in esecuzione
- **ATTENZIONE**: quando si scrive(legge) su un socket, se non c'è abbastanza spazio(dati) disponibile nel buffer gestito dal S.O., la chiamata di scrittura **out.println()** o di lettura **in.readLine()** si blocca.
NB: Se un thread si blocca in lettura o scrittura, allora non è più in grado di fare altro



Esempio: Client che risponde alle domande del Server

- Questo esempio mostra un Client/Server (TCP) che si scambiano domande e risposte.
- Creiamo un file QnA.txt con domande seguite da risposte, una per ogni riga. Ad esempio

```
What caused the craters on the moon?  
meteorites  
How far away is the moon (in miles)?  
239000  
How far away is the sun (in millions of miles)?  
93  
Is the Earth a perfect sphere?  
no  
What is the internal temperature of the Earth (in degrees F)?  
9000
```



Client che risponde alle domande del Server

```
public class QAClient {
    Socket socket = null;
    BufferedReader in = null;
    PrintWriter out = null;
    QAClient(String address){
        try {
            socket = new Socket(address, QAServer.PORTNUM);
            in = new BufferedReader(new InputStreamReader(
                socket.getInputStream()));
            out = new PrintWriter(new BufferedWriter(new OutputStreamWriter(
                socket.getOutputStream())), true);
        } catch (IOException e) {
            System.err.println("Couldn't create stream socket");
            System.exit(1);
        }
    }
    public static void main(String[] args) {
        if (args.length != 1) {
            new QAClient("localhost").exec();
        } else
            new QAClient(args[0]).exec();
    }
}
```



Client che risponde alle domande del Server

```
void exec() {
    String serverQuestion;
    String userAnswer;
    BufferedReader userIn = new BufferedReader(new InputStreamReader(
                                                System.in));

    // domande e risposte con il server
    try {
        while ((serverQuestion = in.readLine()) != null) {
            System.out.println("Server: " + serverQuestion);
            if (serverQuestion.equals("END"))
                break;
            userAnswer=userIn.readLine();
            System.out.println("Client: " + userAnswer);
            out.println(userAnswer);
        }
        out.close(); in.close(); socket.close();
    } catch (IOException e) {
        System.err.println("I/O error trying to talk to server");
    }
}
```



Server che fa le domande al Client

```
public class QAServer {
    public static final int PORTNUM = 1234;
    enum QAserverState {WAITFORCLIENT, WAITFORANSWER, WAITFORCONFIRM};
    private ArrayList<String> questions = new ArrayList<String>();
    private ArrayList<String> answers = new ArrayList<String>();
    private ServerSocket serverSocket;
    private int num = 0; // indice domanda da fare
    private QAserverState state = QAserverState.WAITFORCLIENT;
    private File qaFile;
    private Random rand = new Random(System.currentTimeMillis());

    public static void main(String[] args) {
        if (args.length != 1) {
            System.out.println("You must supply the QA filename");
            return;
        } else {
            new QAServer(args[0]).exec();
        }
    }
}
```



Server che fa le domande al Client

```
public QAServer(String fileName) {
    try {
        serverSocket = new ServerSocket(PORTNUM);
        System.out.println("Server up and running...");
    } catch (IOException e) {
        System.err.println("Exception: couldn't create socket");
        System.exit(1);
    }
    qaFile = new File(fileName);
    if(!qaFile.exists()) {
        System.err.println("Error: "+ fileName+" doesn't exist!");
        System.exit(1);
    }
    if (!initQnA()) {
        System.err.println("Error: couldn't initialize Q&A");
        System.exit(1);
    }
    num = Math.abs(rand.nextInt()) % questions.size();
}
```



Server che fa le domande al Client

```
private boolean initQnA() {
    BufferedReader br=null;;
    try {
        br = new BufferedReader(new InputStreamReader(
            new DataInputStream(new FileInputStream(qaFile))));
        String strLine;
        while ((strLine = br.readLine()) != null) {
            questions.add(strLine);
            if ((strLine = br.readLine()) != null)
                answers.add(strLine);
        }
    } catch (IOException e) {
        System.err.println("I/O error trying to read questions");
        return false;
    } finally {
        try { br.close(); } catch (Exception e) { }
    }
    return true;
}
```




Server che fa le domande al Client

```
public void exec() {
    Socket clientSocket = null;
    String outLine, inLine;
    while (true) {
        if (serverSocket == null) return;
        try {
            clientSocket = serverSocket.accept();
        } catch (IOException e) { System.exit(1); }
        try {
            BufferedReader is = new BufferedReader(new InputStreamReader(
                clientSocket.getInputStream()));
            PrintWriter os = new PrintWriter(new BufferedWriter(
                new OutputStreamWriter(clientSocket.getOutputStream())), true);
            os.println(processInput(null));
            while ((inLine = is.readLine()) != null) {
                outLine = processInput(inLine);
                os.println(outLine);
                if (outLine.equals("END"))
                    break;
            }
            os.close(); is.close(); clientSocket.close();
        } catch (Exception e) { System.err.println("Exception "+e); }
    }
}
```



Server che fa le domande al Client

```
String processInput(String inStr) {
    String outStr = "";
    switch (state) {
        case WAITFORCLIENT: // Ask a question
            outStr = questions.get(num);
            state = QAserverState.WAITFORANSWER;
            break;
        case WAITFORANSWER: // Check the answer
            if (inStr.equalsIgnoreCase(answers.get(num)))
                outStr = "That's correct! Want another? (y/n)";
            else
                outStr = "Wrong, the correct answer is " + answers.get(num) +
                    ". Want another? (y/n)";
            state = QAserverState.WAITFORCONFIRM;
            break;
    }
}
```



Server che fa le domande al Client

```
case WAITFORCONFIRM: // See if they want another question
    if (inStr.equalsIgnoreCase("y")) {
        num = Math.abs(rand.nextInt()) % questions.size();
        outStr = questions.get(num);
        state = QAserverState.WAITFORANSWER;
    } else {
        outStr = "END";
        state = QAserverState.WAITFORCLIENT;
    }
    break;
}
return outStr;
}
```



Funzionamento

```
gigi@hp-850g2-lavazza: ~/Documents/Didattica/Prog_CD/Esempi/Lez7_QA
File Edit View Search Terminal Help
drwxrwxr-x 2 gigi gigi 4096 mar 29 12:23 Lez7_echo
drwxrwxr-x 2 gigi gigi 4096 apr 2 21:10 Lez7_MultiJabber
drwxrwxr-x 2 gigi gigi 4096 apr 2 21:54 Lez7_QA
drwxrwxr-x 2 gigi gigi 4096 apr 2 19:48 Lez7_UDP_imageServer
drwxrwxr-x 2 gigi gigi 4096 apr 2 18:19 Lez7_UDPserver
drwxrwxr-x 2 gigi gigi 4096 apr 2 20:38 Lez7_UDPserver
-rw-rw-r-- 1 gigi gigi 244 apr 2 21:54 Lez7_UDPserver
-rw-rw-r-- 1 gigi gigi 73154 apr 2 21:54 Lez7_UDPserver
gigi@hp-850g2-lavazza:~/Documents/Didattica/Prog_CD/Esempi/Lez7_QA$ ls
total 24
-rw-rw-r-- 1 gigi gigi 9149 apr 2 21:54 Lez7_UDPserver
-rw-rw-r-- 1 gigi gigi 10497 apr 2 21:54 Lez7_UDPserver
gigi@hp-850g2-lavazza:~/Documents/Didattica/Prog_CD/Esempi/Lez7_QA$ ./Lez7_UDPserver
Server up and running...
What caused the craters on the moon?
How far away is the moon (in miles)?
How far away is the sun (in millions)?
Is the Earth a perfect sphere?
What is the internal temperature of the Earth?
Client: meteorites
Server: That's correct! Want another? (y/n)
y
Client: y
Server: How far away is the moon (in miles)?
200000
Client: 200000
Server: Wrong, the correct answer is 239000. Want another? (y/n)
y
Client: y
Server: Is the Earth a perfect sphere?
no
Client: no
Server: That's correct! Want another? (y/n)
n
Client: n
Server: END
gigi@hp-850g2-lavazza:~/Documents/Didattica/Prog_CD/Esempi/Lez7_QA$
```



QA: Server multi client

- Modifichiamo il programma server in modo che accetti molti client contemporaneamente.
- Il programma client non ha bisogno di essere modificato.



Server master

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

public class QAServer {
    public static final int PORTNUM = 1234;
    private ArrayList<String> questions = new ArrayList<String>();
    private ArrayList<String> answers = new ArrayList<String>();
    private ServerSocket serverSocket;
    private File qaFile;
```



Server master

```
public QAServer(String fileName) {
    try {
        serverSocket = new ServerSocket(PORTNUM);
        System.out.println("Server up and running...");
    } catch (IOException e) {
        System.err.println("Exception: couldn't create socket");
        System.exit(1);
    }
    qaFile = new File(fileName);
    if(!qaFile.exists()) {
        System.err.println("Error: "+ fileName+" doesn't exist!");
        System.exit(1);
    }
    if (!initQnA()) {
        System.err.println("Error: couldn't initialize Q&A");
        System.exit(1);
    }
}
```



Server master

```
private boolean initQnA() {
    BufferedReader br=null;;
    try {
        br = new BufferedReader(new InputStreamReader(
            new DataInputStream(new FileInputStream(qaFile))));
        String strLine;
        while ((strLine = br.readLine()) != null) {
            questions.add(strLine);
            if ((strLine = br.readLine()) != null)
                answers.add(strLine);
        }
    } catch (IOException e) {
        System.err.println("I/O error trying to read questions");
        return false;
    } finally {
        try { br.close(); } catch (Exception e) { }
    }
    return true;
}
```




Server master

```
public void exec() {
    Socket clientSocket = null;
    while (true) {
        if (serverSocket == null)
            return;
        try {
            clientSocket = serverSocket.accept();
        } catch (IOException e) { System.exit(1); }
        new QASlave(clientSocket, questions, answers).start();
    }
}

public static void main(String[] args) {
    if (args.length != 1) {
        System.out.println("Usage: java Class-Name <address>");
        return;
    } else {
        new QAServer(args[0]).exec();
    }
}
```



Server slave

```
import java.io.*;
import java.net.*;
import java.util.*;
public class QASlave extends Thread{
    enum QAServerState {WAITFORCLIENT, WAITFORANSWER,
                        WAITFORCONFIRM};

    private ArrayList<String> questions = new ArrayList<String>();
    private ArrayList<String> answers = new ArrayList<String>();
    private int num = 0;
    Socket clientSocket;
    private QAServerState state = QAServerState.WAITFORCLIENT;
    private Random rand = new Random();
    BufferedReader is;
    PrintWriter os;
    public QASlave(Socket s, ArrayList<String> q,
                  ArrayList<String> a) {
        questions=q;
        answers=a;
        clientSocket=s;
    }
```



Server slave

```
public void run() {
    String inLine;
    try {
        is = new BufferedReader(new InputStreamReader(
                                clientSocket.getInputStream()));
        os = new PrintWriter(new BufferedWriter(new
            OutputStreamWriter(clientSocket.getOutputStream()), true);
        os.println(processInput(null));
        while ((inLine = is.readLine()) != null) {
            String outLine = processInput(inLine);
            os.println(outLine);
            if (outLine.equals("END"))
                break;
        }
        os.close(); is.close(); clientSocket.close();
    } catch (Exception e) { System.err.println("Exception "+e); }
}
```



Server slave

```
String processInput(String inStr) {  
    String outStr = "";  
    switch (state) {  
        case WAITFORCLIENT: // Ask a question  
            outStr = questions.get(num);  
            state = QAserverState.WAITFORANSWER;  
            break;  
        case WAITFORANSWER: // Check the answer  
            if (inStr.equalsIgnoreCase(answers.get(num)))  
                outStr = "That's correct! Want another? (y/n)";  
            else  
                outStr = "Wrong, the correct answer is " +  
                    answers.get(num) + ". Want another? (y/n)";  
            state = QAserverState.WAITFORCONFIRM;  
            break;  
    }
```

Come prima
(la logica non cambia)



Slave

```
case WAITFORCONFIRM: // See if they want another question
    if (inStr.equalsIgnoreCase("y")) {
        num = Math.abs(rand.nextInt()) % questions.size();
        outStr = questions.get(num);
        state = QAserverState.WAITFORANSWER;
    } else {
        outStr = "END";
        state = QAserverState.WAITFORCLIENT;
    }
    break;
}
return outStr;
}
}
```



Test con due client

```
gigi@gigi-HP-EliteBook-850-G6: ~/Documents/Didattica/Prog_CD/Code/2021/Lez08/QA_multis...
gigi@gigi-HP-EliteBook-850-G6: ~/Documents/Didattica/Prog_CD/Code/2021/Lez08/QA_multis...
./QnA.txt
Server up and running...

gigi@gigi-HP-EliteBook-850-G6: ~/Documents/Didattica/Prog_CD/Code/2021/Lez08/QA_multis...
AClient
Server: What caused the craters on the moon?
meteorites
Client: meteorites
Server: That's correct! Want another? (y/n)
y
Client: y
Server: How far away is the moon (in miles)?
239000
Client: 239000
Server: That's correct! Want another? (y/n)
n
Client: n
Server: END
gigi@gigi-HP-EliteBook-850-G6: ~/Documents/Didattica/Prog_CD/Code/2021/Lez08/QA_multis...

gigi@gigi-HP-EliteBook-850-G6: ~/Documents/Didattica/Prog_CD/Code/2021/Lez08/QA_multis...
gigi@gigi-HP-EliteBook-850-G6: ~/Documents/Didattica/Prog_CD/Code/2021/Lez08/QA_multis...$ java Q
AClient
Server: What caused the craters on the moon?
moles
Client: moles
Server: Wrong, the correct answer is meteorites. Want another? (y/n)
y
Client: y
Server: Is the Earth a perfect sphere?
no
Client: no
Server: That's correct! Want another? (y/n)
y
Client: y
Server: What is the internal temperature of the Earth (in degrees F)?
5678
Client: 5678
Server: Wrong, the correct answer is 9000. Want another? (y/n)
n
Client: n
Server: END
```



ATTENZIONE

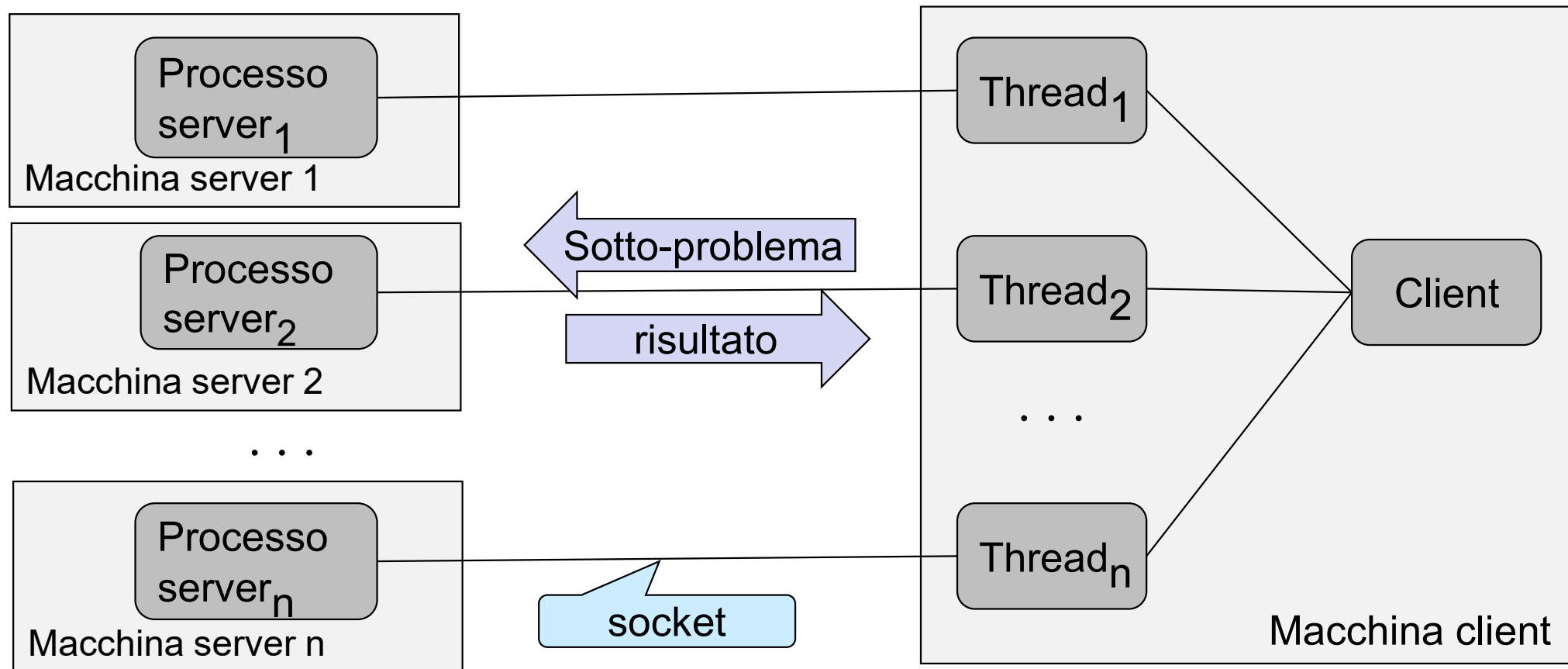


```
new QASlave(clientSocket, questions, answers).start();
```

- Ogni slave riceve un riferimento alla lista delle domande e alla lista delle risposte.
- Queste liste sono condivise
- Quindi in linea di principio c'è pericolo di corse critiche
- Nel caso visto le liste vengono solo lette dai thread **QASlave**, quindi non c'è pericolo
- In generale, se gli slave modificano i dati condivisi, bisogna proteggerli con metodi **synchronized**.

Molteplici server e parallelismo

- In molti casi è possibile suddividere un problema in tanti sotto-problemi indipendenti analizzabili singolarmente in parallelo.
- In particolare quando si hanno tanti processi server che forniscono tutti lo stesso servizio e girano ciascuno su un processore diverso



Esempio: Word Count

- Dato un insieme di documenti di testo, si vuole contare il numero di parole presenti complessivamente nei documenti.
- Dividiamo il problema complessivo in un insieme di piccoli task, sullo stile di Map-Reduce.
- Ci sono N server disponibili.
 - ▶ Se girano su macchine diverse, i server lavorano effettivamente parallelo
- Il client genera N thread, ciascuno dei quali comunica con un diverso server, chiedendogli di risolvere un sotto-problema.
- Il client raccoglie i risultati dai suoi vari thread e li assembla.



Word Count - Client

```
public class WordCountClient {  
    int numServers;  
    String theArgs[];  
    WordCountClient(String args[]){  
        numServers=args.length/2;  
        theArgs=args;  
    }  
  
    public static void main(String args[]) {  
        WordCountClient wcc=new WordCountClient(args);  
        try {  
            wcc.exec();  
        } catch (Exception e) {  
            System.err.println("qualcosa storto ando`");  
        }  
    }  
}
```

Tante coppie ⟨indirizzo IP, porta⟩ quanti sono i server da usare.



Word Count - Client

```
void exec() throws UnknownHostException, IOException{
    int port;
    String addr;
    ArrayList<WordCounter> counters=new ArrayList<WordCounter>();
    long t0=System.currentTimeMillis();
    BarrierReachedAction reducer=new BarrierReachedAction(t0);
    CyclicBarrier cyclicBarrier=new CyclicBarrier(numServers,
                                                    reducer);

    int incr=400/numServers;
    for(int i=0; i<numServers; i++) {
        addr=args[i*2];
        port=Integer.parseInt(args[i*2+1]);
        counters.add(new WordCounter(addr, port, cyclicBarrier,
                                      (i)*incr, (i+1)*incr));
    }
    reducer.setCounters(counters);
    for(int i=0; i<numServers; i++) {
        counters.get(i).start();
    }
}
```



Word Count – Client Thread

```
public class WordCounter extends Thread {
    CyclicBarrier cyclicBarrier;
    public int wordCount=0;    // risultato, poi letto da
                              // BarrierReachedAction

    int startFile, endFile;
    int thePORT;
    InetAddress addrIP;
    WordCounter(String addr, int port, CyclicBarrier c, int start,
                int end) throws UnknownHostException, IOException {
        cyclicBarrier=c;
        startFile=start;
        endFile=end;
        thePORT=port;
        addrIP = InetAddress.getByName(addr) ;
        System.out.println("word counter slave ready (addr="
                           +addrIP+" and port="+thePORT+" ) " );
    }
}
```



Word Count – Client Thread

```
public void run() {
    String str;
    try {
        Socket socket = new Socket(addrIP, thePORT);
        BufferedReader in = new BufferedReader(
            new InputStreamReader(socket.getInputStream()));
        PrintWriter out = new PrintWriter(new BufferedWriter(
            new OutputStreamWriter(socket.getOutputStream())), true);
        out.println(startFile);
        out.println(endFile);
        str = in.readLine();
        wordCount=Integer.parseInt(str);
        System.out.println("word counter slave counted "+
                           wordCount+" words; going to wait");
        cyclicBarrier.await();
    } catch (Exception e1) {
        System.out.println("word counter slave aborting "+e1);
    }
}
```



Word Count – Barrier implementa reduction

```
public class BarrierReachedAction implements Runnable {
    private ArrayList<WordCounter> wordCounters;
    long t0=0;
    public BarrierReachedAction(long start) {
        t0=start;
    }
    public void setCounters(ArrayList<WordCounter> wordCounters) {
        this.wordCounters=wordCounters;
    }
    public void run() {
        System.out.println("BarrierReached!");
        int totalWords=0;
        for(WordCounter wc:wordCounters) {
            totalWords+=wc.wordCount;
        }
        long totalTimeElapsed=System.currentTimeMillis()-t0;
        System.out.println("Elapsed time = "+totalTimeElapsed);
        System.out.println("Word count is = " + totalWords);
    }
}
```



Word Count – Server

```
public class CounterServer {
    int wordCount=0;
    String filePath;
    int startFile, endFile;
    int myPORT;
    BufferedReader in;
    PrintWriter out;
    CounterServer(String path, String port){
        myPORT=Integer.parseInt(port);
        filePath=path;
    }
    public static void main(String args[]) throws IOException {
        if(args.length!=2) {
            System.out.println("usage: java CounterServer"+
                               " <file path> <PORT>");
        } else {
            CounterServer CS=new CounterServer(args[0], args[1]);
            CS.serve();
        }
    }
}
```



Word Count – Server

```
void serve() throws IOException {
    String str;
    ServerSocket server = new ServerSocket(myPORT) ;
    Socket connection = null;
    while (true) {
        try {
            connection = server.accept() ;
            in=new BufferedReader(new InputStreamReader(
                                    connection.getInputStream())) ;
            out=new PrintWriter(new BufferedWriter(new
                OutputStreamWriter(connection.getOutputStream()), true) ;
            str=in.readLine() ;
            this.startFile=Integer.parseInt(str) ;
            str=in.readLine() ;
            this.endFile=Integer.parseInt(str) ;
            count(); // qui c'e` il servizio di conteggio
            out.println(wordCount) ;
            System.out.println("server sent:"+wordCount) ;
            connection.close() ;
        }
    }
}
```




Word Count – Server

```
// conclusione serve()
    catch (IOException ex) {
        System.err.println("server IO error");
    }
    finally {
        try {
            if (connection != null) connection.close();
        } catch (IOException ex) {}
    }
}
```



Word Count – Server

```
public void count() {  
    BufferedReader br = null;  
    this.startFile=startFile;  
    this.endFile=endFile;  
    try {  
        for(int i=startFile; i<endFile; i++) {  
            br=null;  
            String fileName=filePath+"/file_" + i + ".txt";  
            br=new BufferedReader(new FileReader(fileName));  
            String line=br.readLine();  
            while(line!=null) {  
                String[] wordArray=line.split("\\s+");  
                wordCount+=wordArray.length;  
                line=br.readLine();  
            }  
        }  
        br.close();  
    } catch(Exception exc) { System.out.println(exc); }  
}
```



Test con un solo server

```
gigi@gigi-HP-EliteBook-850-G6: ~/Documents/Didattica...  
[1] 31leggo ../../../../Borse/file_379.txt  
gigi@gigi-HP-EliteBook-850-G6: ~/Documents/Didattica...  
a Countleggo ../../../../Borse/file_380.txt  
Connectleggo ../../../../Borse/file_381.txt  
Connectleggo ../../../../Borse/file_382.txt  
serverleggo ../../../../Borse/file_383.txt  
serverleggo ../../../../Borse/file_384.txt  
serverleggo ../../../../  
finitoleggo ../../../../gigi@gigi-HP-EliteBook-850-G6:~/Documents/Didattica/Prog_CD/Code/2021/Lez08/Word  
finitoleggo ../../../../CounterMapReduce/Client$ java WordCountClient localhost 9999  
serverleggo ../../../../word counter slave starting  
serverleggo ../../../../word counter slave operating on 0,400  
serverleggo ../../../../word counter slave counted 3179 words; going to wait  
leggo ../../../../BarrierReached!  
leggo ../../../../Elapsed time = 172  
leggo ../../../../Word count is = 3179  
leggo ../../../../gigi@gigi-HP-EliteBook-850-G6:~/Documents/Didattica/Prog_CD/Code/2021/Lez08/Word  
leggo ../../../../CounterMapReduce/Client$  
leggo ../../../../  
leggo ../../../../  
leggo ../../../../  
finito  
server sent:3179
```



Test con due server

```
gigi@gigi-HP-EliteBook-850-G6: ~/Documents/Didattica/Prog_CD/...
gigi@gigi-HP-EliteBook-850-G6:~/Documents/Didattica/Prog_CD/Code/2021/Lez08/WordCounterMapReduce/Server$ java CounterServer ../../../../../../Borse 9999 &
[1] 21408
gigi@gigi-HP-EliteBook-850-G6:~/Documents/Didattica/Prog_CD/Code/2021/Lez08/WordCounterMapReduce/Server$ java CounterServer ../../../../../../Borse 9998
Connection accepted: Socket[addr=/127.0.0.1,port=55290,localport=9999]
Connection accepted: Socket[addr=/127.0.0.1,port=47574,localport=9998]
server received:0
server received:200
server received:200
server received:400
finito
finito
server sent:1660
server sent:1519

gigi@gigi-HP-EliteBook-850-G6:~/Documents/Didattica/Prog_CD/Code/2021/Lez08/WordCounterMapReduce/Client$ java WordCountClient localhost 9999 localhost 9998
word counter slave starting
word counter slave starting
word counter slave operating on 0,200
word counter slave operating on 200,400
word counter slave counted 1519 words; going to wait
word counter slave counted 1660 words; going to wait
BarrierReached!
Elapsed time = 124
Word count is = 3179
gigi@gigi-HP-EliteBook-850-G6:~/Documents/Didattica/Prog_CD/Code/2021/Lez08/WordCounterMapReduce/Client$
```




Test con quattro server

```
gigi@gigi-HP-EliteBook-850-G6: ~/Documents/Didattica/Prog_CD/Code/2021/Lez08/WordCounterMa...
gigi@gigi-HP-EliteBook-850-G6:~/Documents/Didattica/Prog_CD/Code/2021/Lez08/WordCounterMapReduce/Server$ java C
ounterServer ../../../../../../Borse 9900 &
[1] 21545
gigi@gigi-HP-EliteBook-850-G6:~/Documents/Didattica/Prog_CD/Code/2021/Lez08/WordCounterMapReduce/Server$ java C
ounterServer ../../../../../../Borse 9901 &
[2] 21563
gigi@gigi-HP-EliteBook-850-G6:~/Documents/Didattica/Prog_CD/Code/2021/Lez08/WordCounterMapReduce/Server$ java C
ounterServer ../../../../../../Borse 9902 &
[3] 21581
gigi@gigi-HP-EliteBook-850-G6:~/Documents/Didattica/Prog_CD/Code/2021/Lez08/WordCounterMapReduce/Server$ java C
ounterServer ../../../../../../Borse 9903
Connection accepted: Socket[addr=/127.0.0.1,port=60264,localport=9900]
Connection accepted: Socket[addr=/127.0.0.1,port=47766,localport=9903]
Connection accepted: Socket[addr=/127.0.0.1,port=44290,localport=9901]
Connection accepted: Socket[addr=/127.0.0.1,port=54872,localport=9902]
server received:100
server received:200server received:300
server received:200server received:300
server received:400
server received:0
server received:100
finito
finitoserver sent:827
server sent:807
finito
server sent:712
finito
server sent:833

gigi@gigi-HP-EliteBook-850-G6:~/Documents/Didattica/Prog_CD/Code/2021/Lez08/Word
CounterMapReduce/Client$ java WordCountClient localhost 9900 localhost 9901 loca
lhost 9902 localhost 9903
word counter slave starting
word counter slave starting
word counter slave starting
word counter slave starting
word counter slave operating on 200,300
word counter slave operating on 300,400
word counter slave operating on 100,200
word counter slave operating on 0,100
word counter slave counted 827 words; going to wait
word counter slave counted 807 words; going to wait
word counter slave counted 712 words; going to wait
word counter slave counted 833 words; going to wait
BarrierReached!
Elapsed time = 159
Word count is = 3179
gigi@gigi-HP-EliteBook-850-G6:~/Documents/Didattica/Prog_CD/Code/2021/Lez08/Word
CounterMapReduce/Client$
```



Bibliografia

- Molte delle informazioni presentate sono state estratte dal capitolo 1 del libro:
 - ▶ Thinking in Enterprise Java
 - ▶ by Bruce Eckel et. Al.