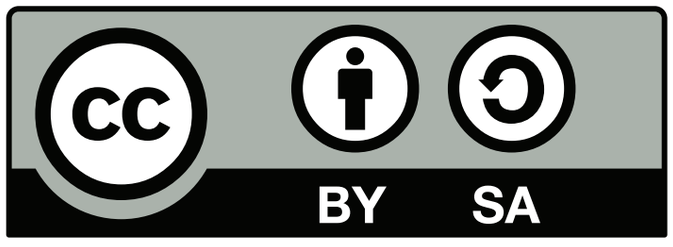
Tecnologia e Applicazioni Internet 2008/9

Lezione 1 - Socket e HTTP

Matteo Vaccari
http://matteo.vaccari.name/
matteo.vaccari@uninsubria.it

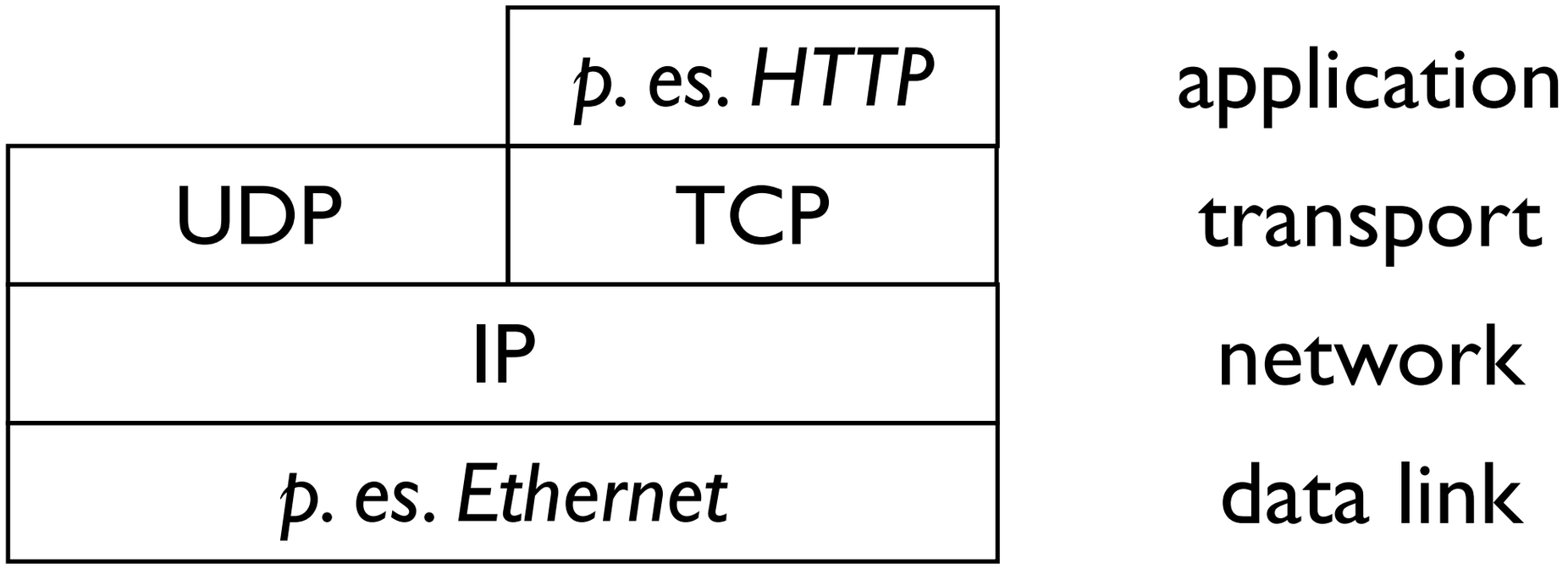


Socket e HTTP

- Richiami di reti: stack TCP/IP
- Uso di socket in Java
- Un server TCP generico
- Un semplice server HTTP
- Una semplice applicazione web

Richiami di reti

Lo stack TCP/IP

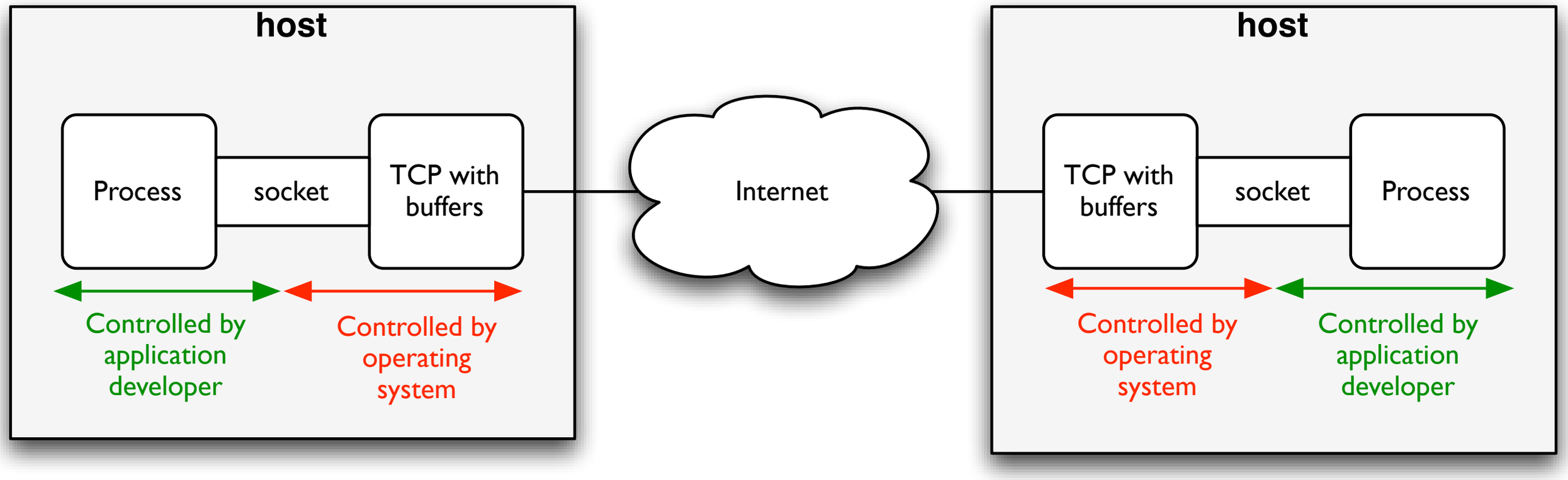


Connessione TCP

Una connessione fornisce due stream con correzione di errore automatica

A 10.000 metri d’altezza, abbiamo una socket “client” e una socket “server”, e sono connesse da una coppia di stream di byte, che comunicano nelle due direzioni

Sockets: a more precise model

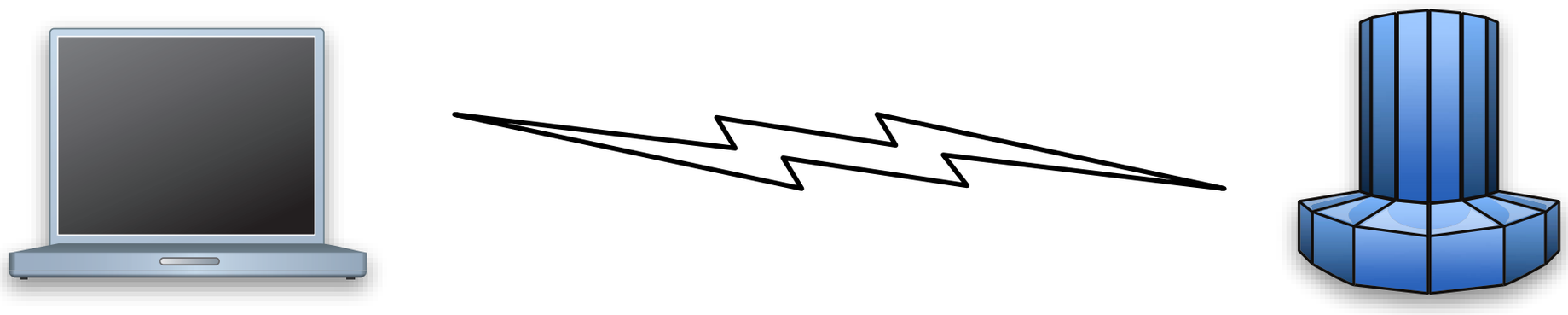


Gli stream sono “illusioni” fornite dal sistema operativo

Disegno da una presentazione di Jonbook Lee:
http://perleybrook.umfk.maine.edu/slides/spring%202004/cos420/Socket_Programming.ppt
Le socket sono un’interfaccia di comunicazione fra un processo e il sistema operativo. Il sistema operativo conserva dei buffer che mantengono i byte che non sono stati ancora consumati (letti) o spediti (di cui abbiamo ricevuto conferma di ricezione dall’altro lato.)

Una *connessione* TCP è identificata da 4 elementi

- Indirizzo locale (32 bit)
- Porta locale (16 bit)
- Indirizzo remoto (32 bit)
- Porta remota (16 bit)



192.168.111.99 porta 49982

www.google.com (74.125.43.147)
porta 80

Simple TCP client

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

public class TcpClient {
    public void main(String[] args) throws IOException {
        String host = "example.com";
        int port = 1234;
        Socket socket = new Socket(host, port);
        try {
            process(socket.getInputStream(), socket.getOutputStream());
        } finally {
            socket.close();
        }
    }

    private void process(InputStream inputStream, OutputStream outputStream) {
        // ...
    }
}
```

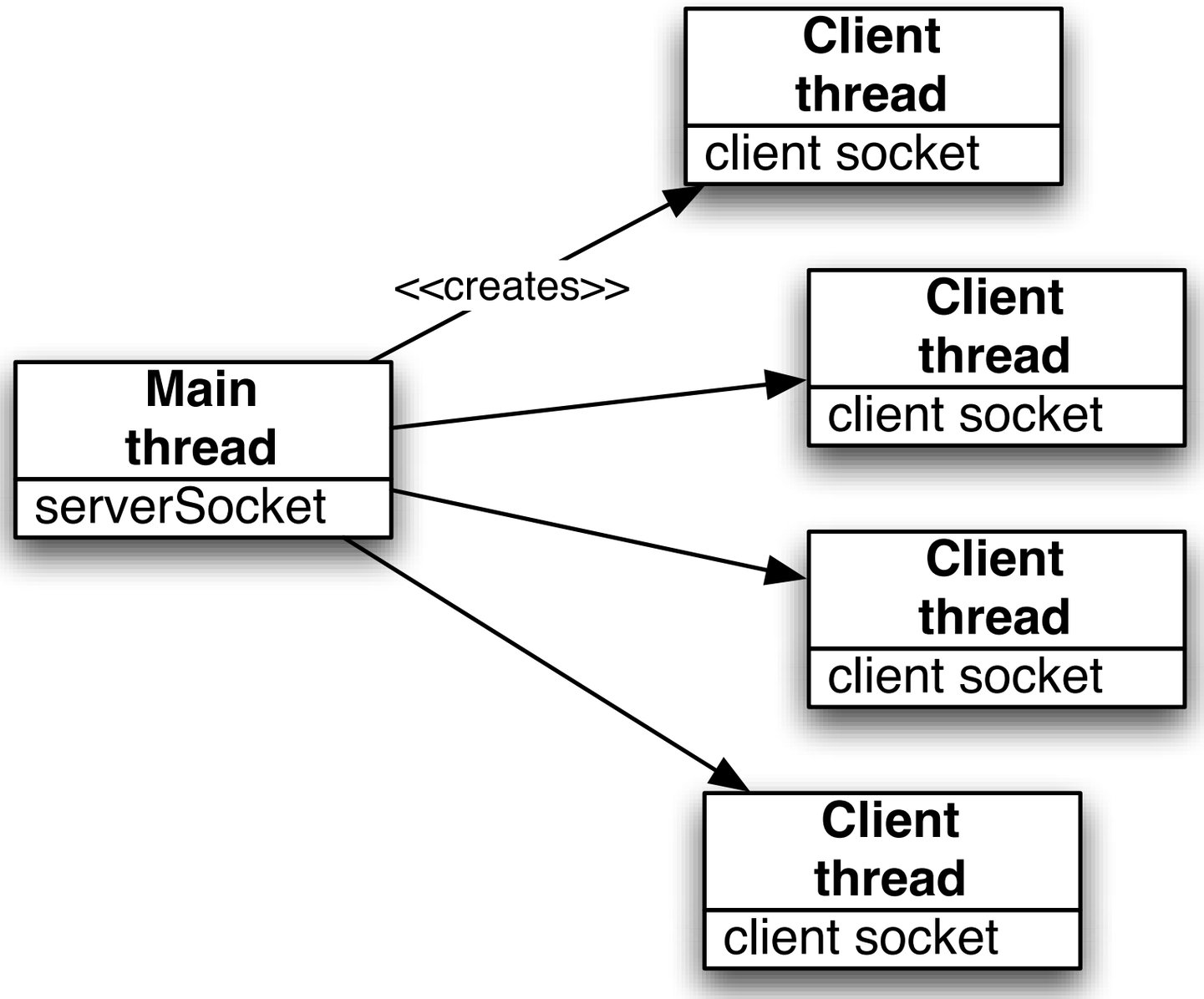
Simple iterative TCP server

```
public class TcpServer {
    public static void main(String[] args) throws IOException {
        ServerSocket serverSocket = new ServerSocket(1234);
        while (true) {
            Socket client = serverSocket.accept();
            try {
                process(client.getInputStream(), client.getOutputStream());
            } finally {
                client.close();
            }
        }
    }

    private static void process(InputStream inputStream, OutputStream
outputStream) {
        // ...
    }
}
```

La serverSocket chiama accept() per mettersi in attesa di un nuovo client. Quando la accept() esce, restituisce una socket che si può usare per comunicare con il client. Questo esempio di codice può gestire un solo client per volta. Si noti che il protocollo applicativo viene implementato in una funzione separata che può essere testata indipendentemente dalla rete.

Concurrent Server



Simple *concurrent* TCP server

```
class ConcurrentTcpServer {
    public static void main(String[] args) throws IOException {
        ServerSocket serverSocket = new ServerSocket(1234);
        while (true) {
            Socket client = serverSocket.accept();
            ClientHandler clientHandler = new ClientHandler(client);
            clientHandler.start();
        }
    }
}
```

```
class ClientHandler extends Thread {
    private final Socket client;
    public ClientHandler(Socket client) {
        this.client = client;
    }
    public void run() {
        // ...
        client.close();
    }
}
```

Ad ogni client viene assegnato un nuovo thread. In questo modo il server riesce a gestire più conversazioni contemporaneamente.

Leggere testo da una socket

```
InputStream inputStream = socket.getInputStream();
// inputStream.read();          // restituisce un *byte*

InputStreamReader streamReader = new InputStreamReader(inputStream);
// streamReader.read();         // restituisce un *char*

BufferedReader bufferedReader = new BufferedReader(streamReader);
// bufferedReader.readLine(); // restituisce una *riga* di testo
```

Che fatica usare java.io

Il design desiderato