Programación Orientada a Servicios
Arquitectura de la Web

Programa de
Ingeniería en Computación
UAM – Azcapotzalco

A cargo de:
Dra. Maricela Claudia Bravo Contreras
mcbc@correo.azc.uam.mx

CBI DIVISIÓN CIENCIAS BÁSICAS E INGENIERÍA
UAM - Azcapotzalco

Casa abierta al tiempo
Universidad Autónoma Metropolitana
Azcapotzalco

Diferencia entre conceptos de la Web

- Página Web
- Sitio Web
- Servidor Web
- Aplicación Web
- Servicio Web

Página Web

Una **página web** es un documento electrónico escrito en un lenguaje llamado **HTML** (siglas del inglés Hypertext Markup Language, o Lenguaje de Marcación de Hipertexto).

Las páginas Web pueden contener texto, gráficos, vídeo, animaciones, sonido y **elementos interactivos.**

Cada página tiene una dirección única que se denomina una **URL** que identifica su ubicación en el servidor.

Las páginas Web contienen generalmente **hiperenlaces** a otras páginas web.

Los **hiperenlaces** son textos e imágenes que **hacen referencia a URL's de otras páginas web**.

Sitio Web

Un **sitio web** se compone de una o más páginas web referidas a un asunto común, como a una persona, un negocio, una organización o a un tema, tal como el deporte.

La primera página se llama la **página de inicio**, y hace las funciones de un índice,indicando el contenido del sitio.

En la página de inicio puede hacer click en los **hiperelaces** para acceder a otras páginas web.

SERVIDOR WEB

Servidores

- Un servidor es un programa que se ejecuta en computadoras normalmente más poderosas que las computadoras personales.

- Se ejecuta sobre sistemas operativos que soportan concurrencia, paralelismo y multiprogramación, por ejemplo Windows 200x server, o basados en Unix.

Tipos de Servidores

a. **Servidor de archivos**. Proporciona acceso a sistemas de archivos distribuidos. Los clientes pueden buscar en directorios, leer y escribir bloques de archivos, etc.

b. **Servidor de bases de datos**. Proporcionan acceso a uno o más DBMS. Las solicitudes de los clientes se realizan normalmente mediante lenguaje SQL.

c. **Servidor de aplicaciones**. Proporcionan acceso a procedimientos remotos, mediante la invocación de los clientes.

d. **Servidor de correo**. Ofrece servicio de envío y recepción de mensajes de correo, así como mensajería instantánea.

e. **Servidor Web**.

¿Qué es un Servidor Web?

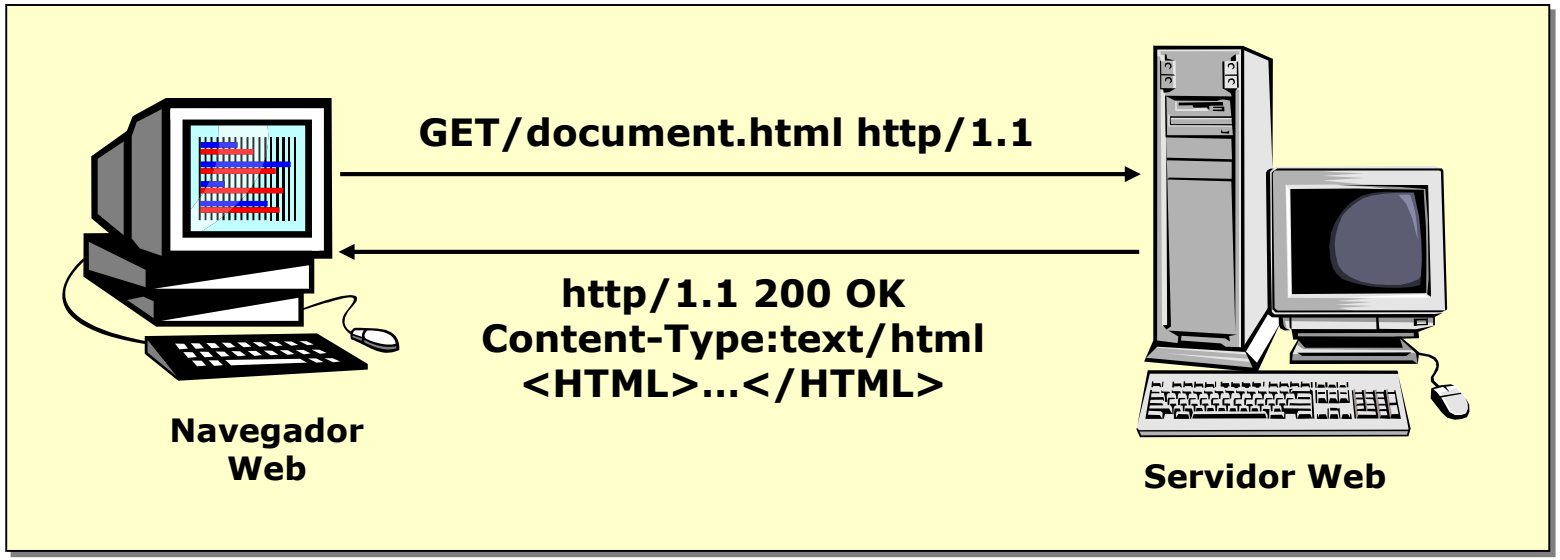
- Un servidor Web o demonio HTTP es un programa que controla el flujo de datos entrantes y salientes de una computadora conectada a Intranet e Internet.
- Un servidor Web es un programa de aplicación que atiende las solicitudes HTTP realizadas por los navegadores.
- Escucha peticiones en el número de puerto 80, normalmente.

Servidor Web

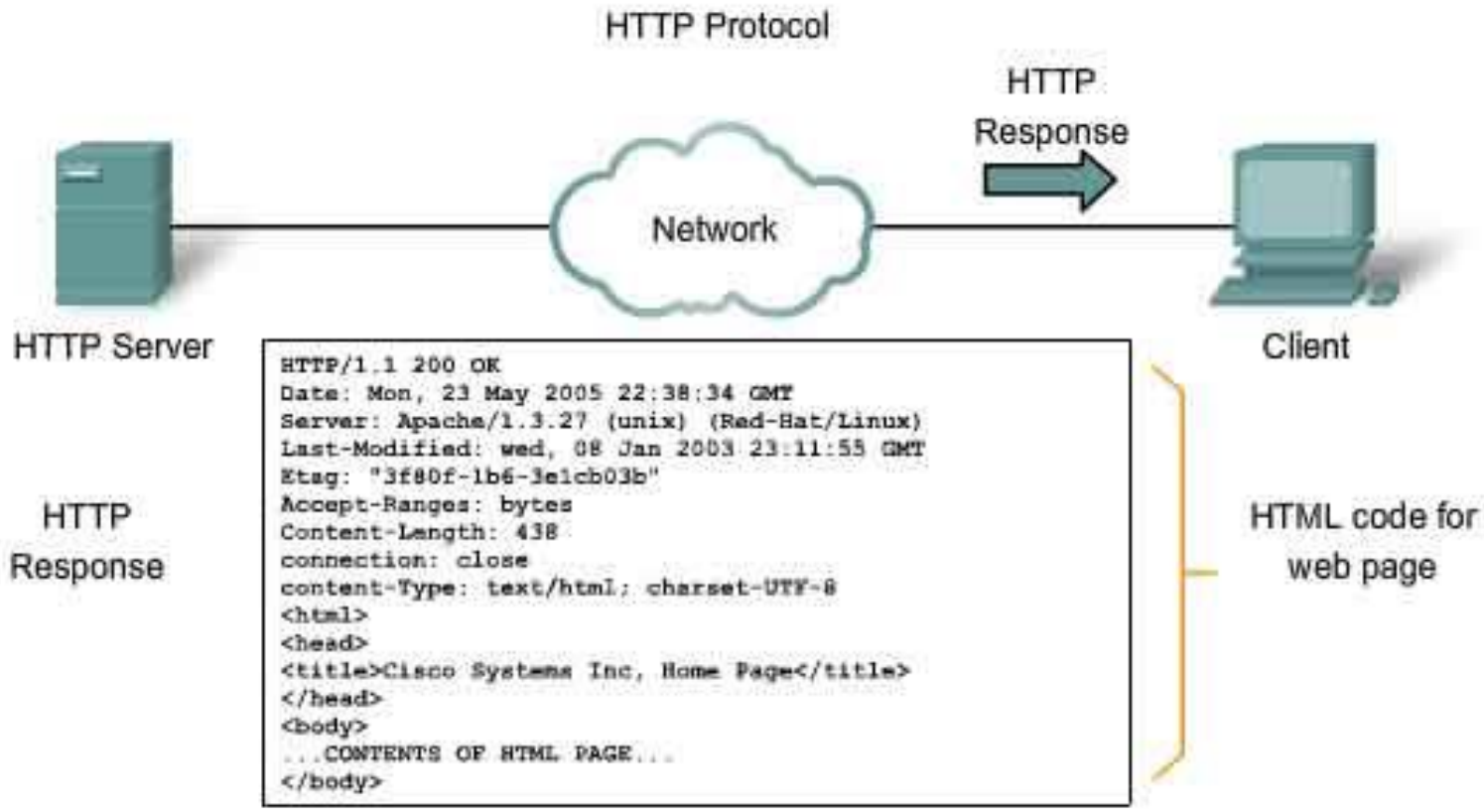
- **Algunos ejemplos:**
  1. CERN httpd
  2. Apache (Libre, servidor más usado del mundo, según Wikipedia)
  3. IIS
  4. Resin
  5. Tomcat (Libre, del proyecto Jakarta de Apache)
  6. Geronimo (Libre, orientado a J2EE, del proyecto Jakarta de Apache, actualmente se encuentra en desarrollo)
  7. JBoss
  8. JOnAS
  9. Cherokee

Protocolo HTTP

- Es un protocolo de petición/respuesta sin estado cuya operación básica es la siguiente :

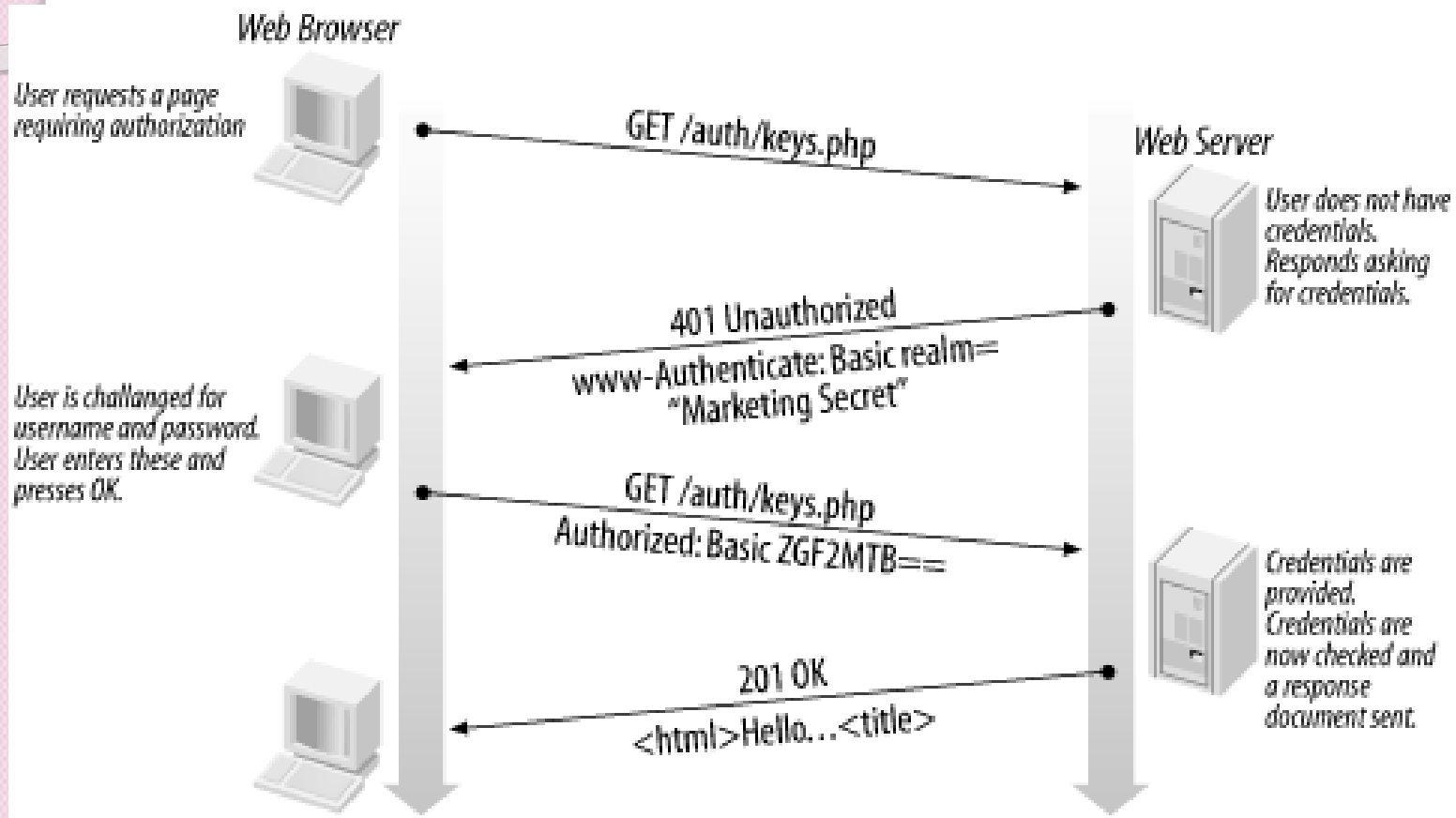


Protocolo HTTP

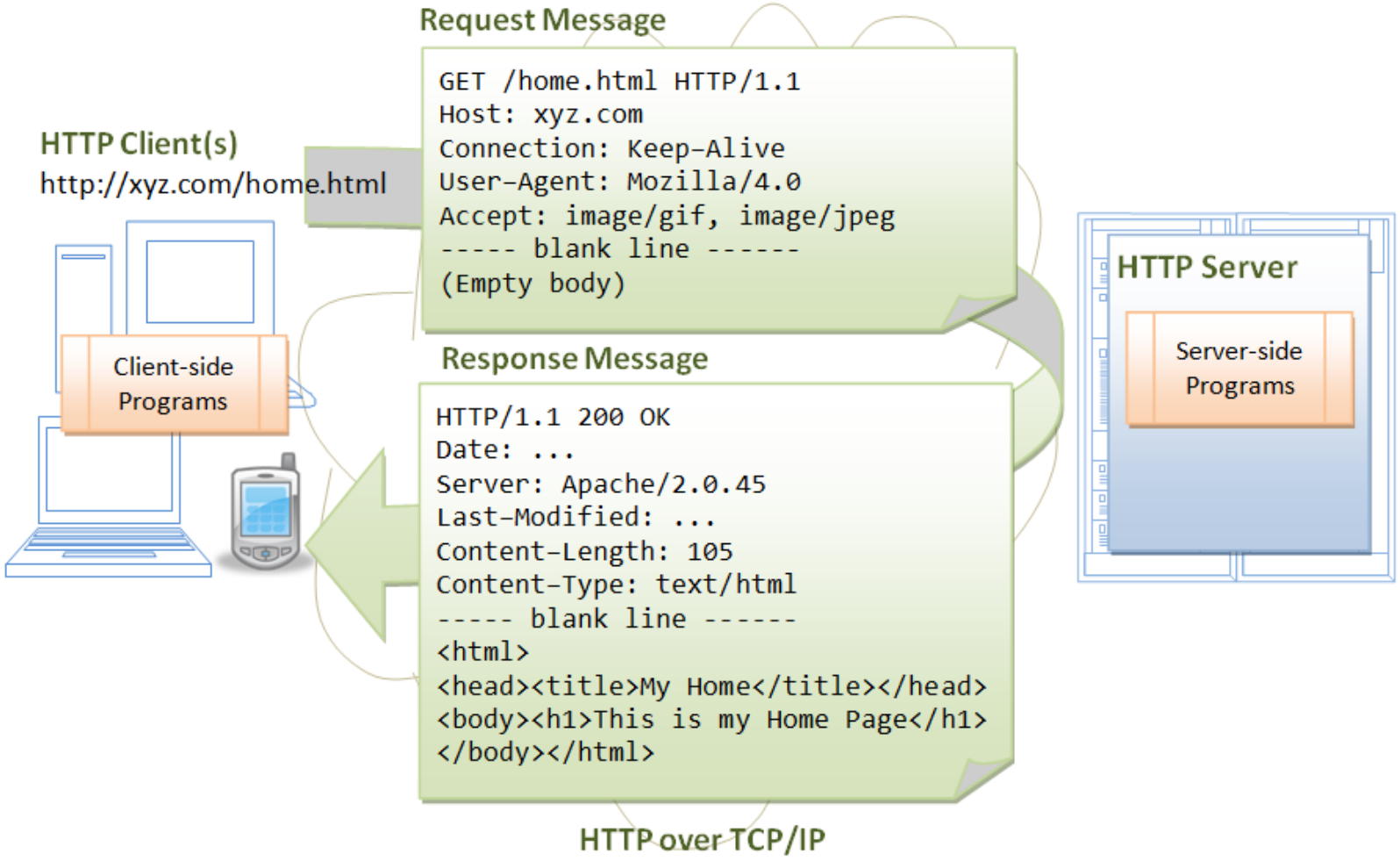


In response to the request, the HTTP server returns code for a web page.

Protocolo HTTP



HTTP Request y Response



IMPLEMENTACIÓN DE UN SERVIDOR DE WEB MEDIANTE SOCKETS

Pila de protocolos TCP/IP

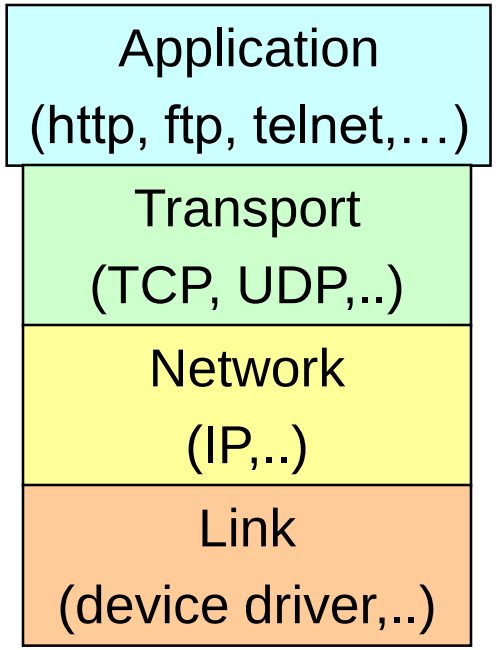
Capa de Aplicación
- Aplicaciones estándar
  - HTTP
  - FTP
  - Telnet
- Aplicaciones de usuario

Capa de Transporte
- TCP
- UDP
- Interfaces de programación:
  - Sockets

Capa de Red
- IP

Capa de Enlace
- Drivers de dispositivos

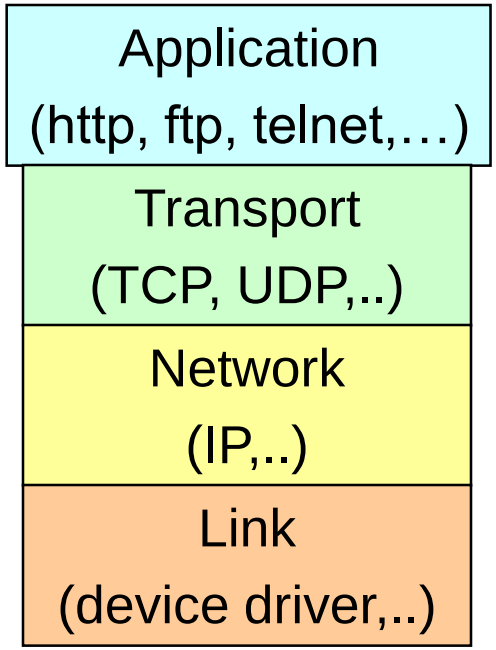
- TCP/IP Stack



Pila de protocolos TCP/IP

- TCP (Transport Control Protocol)
- Es un protocolo orientado a conexión que proporciona un flujo de datos confiable entre dos computadoras.
- Ejemplo de aplicaciones:
  - HTTP
  - FTP
  - Telnet

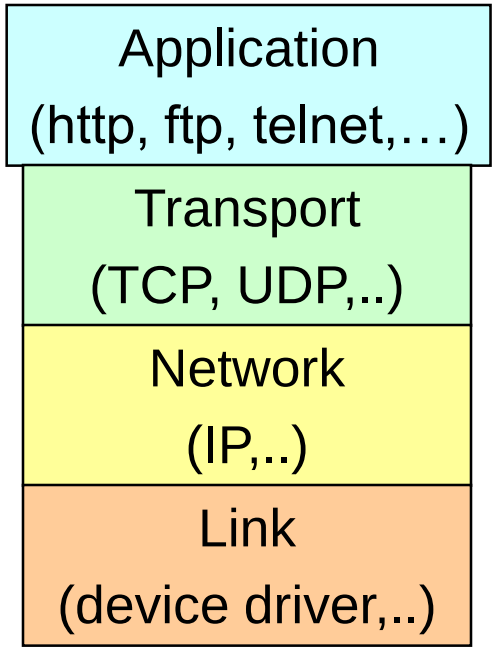
- TCP/IP Stack



Pila de protocolos TCP/IP

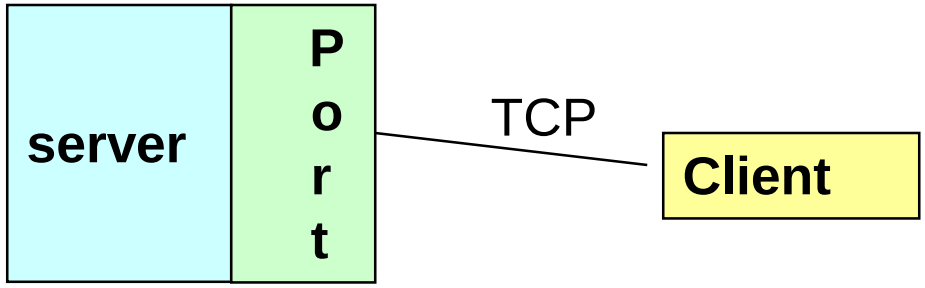
- UDP (User Datagram Protocol)
- Es un protocolo que envía paquetes de datos independientes, llamados datagramas, de una computadora a otra, sin garantizar su llegada.
- Ejemplo de aplicaciones:
  - Clock server
  - Ping

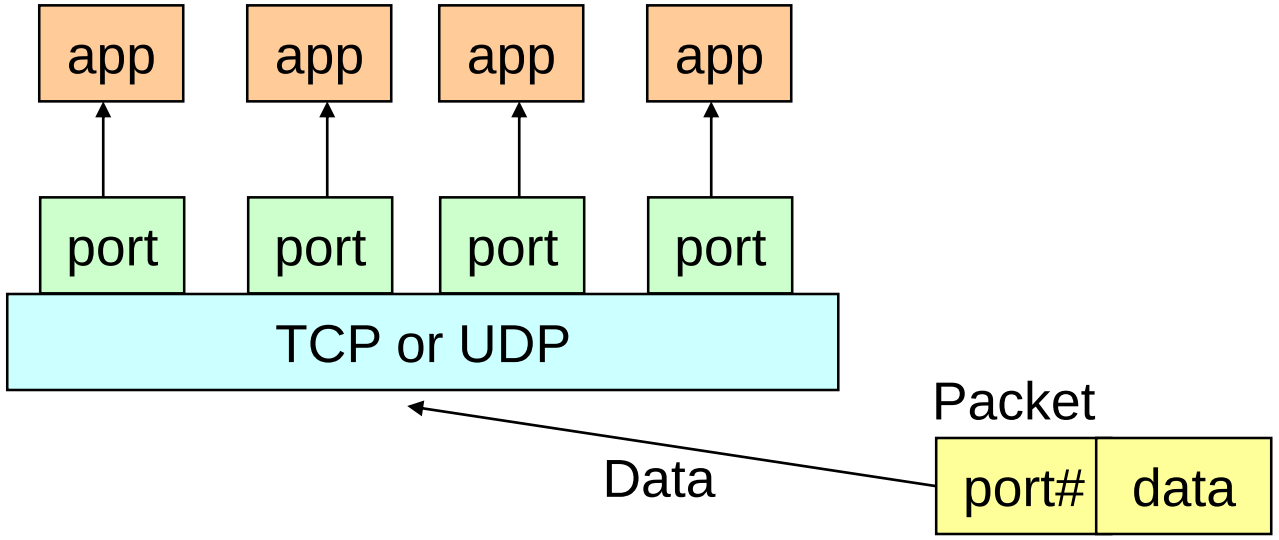
- TCP/IP Stack



¿Qué son los puertos?

- TCP y UDP utilizan puertos para enviar datos entrantes a un proceso particular que se esté ejecutando en la computadora.





¿Qué son los puertos?

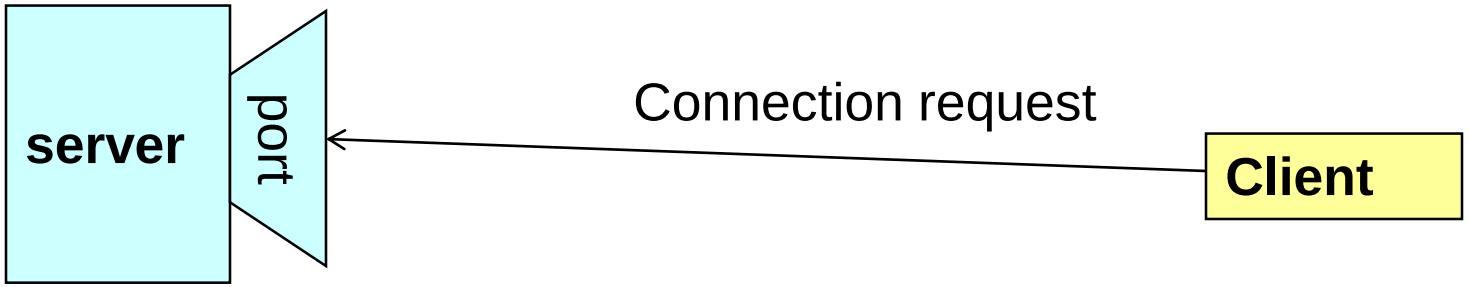
- Los puertos son representados por valores enteros positivos de 16 bits.
- Algunos puertos están reservados para soportar servicios preestablecidos:
  - FTP 21/TCP
  - Telnet 23/TCP
  - SMTP 25/TCP
  - HTTP 80/TCP
- Los procesos o servicios de usuarios generalmente usan números de puertos >= 1024.

Sockets

- Los sockets proporcionan una interfaz para la programación de redes en la capa de transporte.
- Las comunicaciones de redes utilizando Sockets es muy similar al manejo de I/O en archivos.
  - De hecho, el manejo de sockets es tratado como el manejo de archivos.
  - Los streams utilizados en operaciones de I/O de archivos también son aplicables a I/O basado en sockets.
- La comunicación basada en Sockets es independiente del lenguaje de programación.
  - Esto es, que un programa de socket escrito en Java también se puede comunicar con un programa escrito en Java o con un programa de socket no escrito en Java.

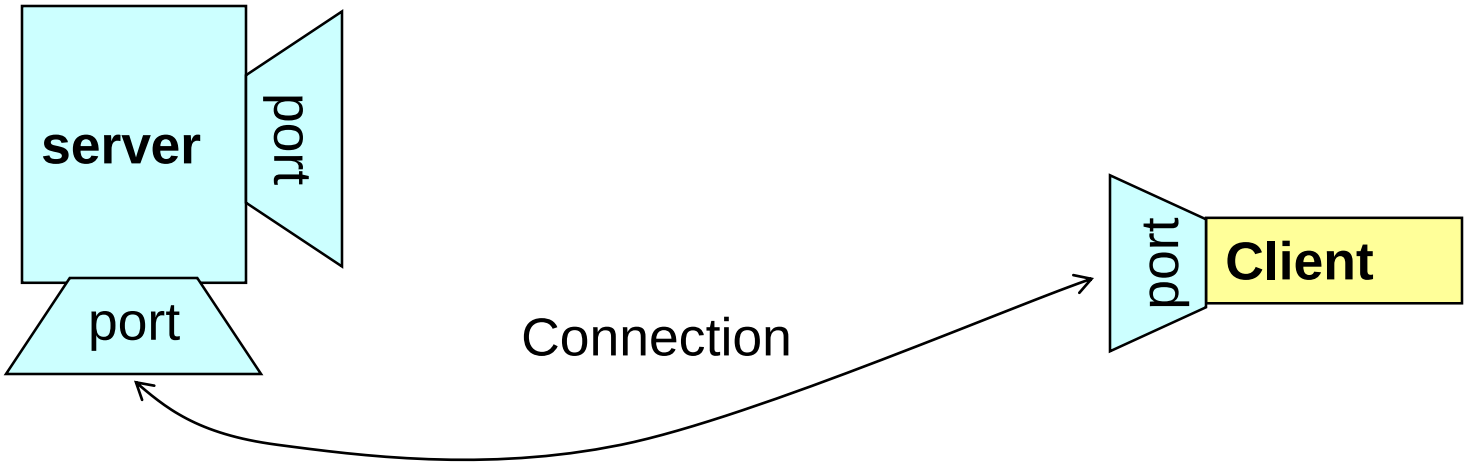
Comunicación entre Sockets

- Un servidor (programa) corre en una computadora específica y tiene un socket que se asocia con un puerto específico. El servidor se mantiene en espera escuchando al socket para cuando un cliente realiza una petición de conexión.



Comunicación entre Sockets

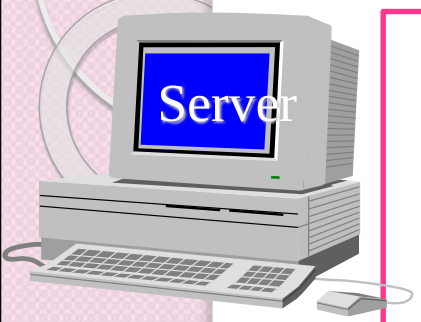
- Si todo sale bien, el servidor acepta la conexión. Después de la aceptación, el servidor obtiene un nuevo socket asociado a un puerto diferente. Necesita un nuevo socket (consecuentemente un número de puerto diferente), de tal forma que puede continuar escuchando al socket original para solicitudes de conexión mientras que atiende al cliente conectado.

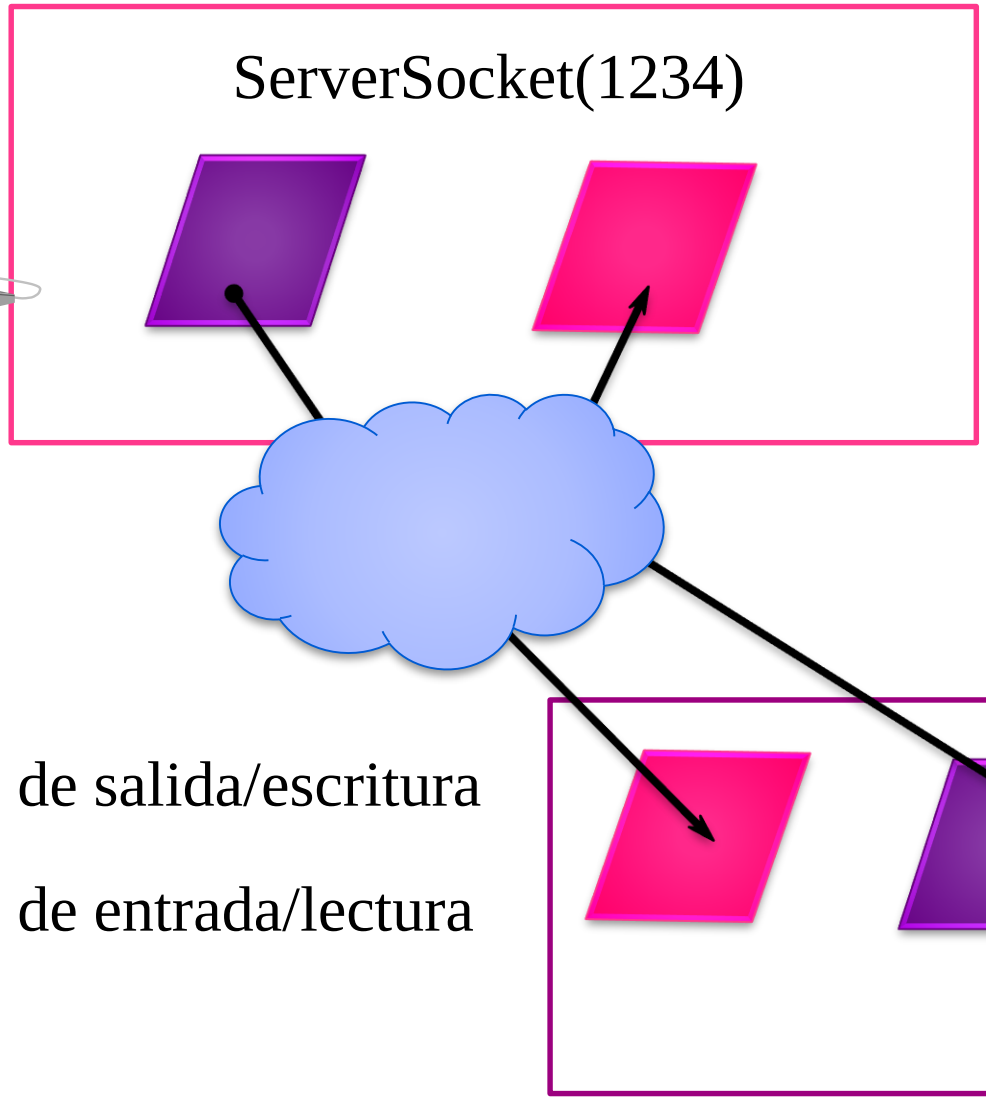


Clases de Java para crear Sockets

- Un socket es un endpoint de un enlace de comunicación bi-direccional entre dos programas ejecutándose en la red.
- Un socket se asocia a un número de puerto de tal forma que la capa de TCP puede identificar la aplicación a la cual están destinados los datos.
- El paquete de Java .net proporciona dos clases:
  - Socket – para implementar un cliente
  - ServerSocket – para implementar un servidor.

Java Sockets



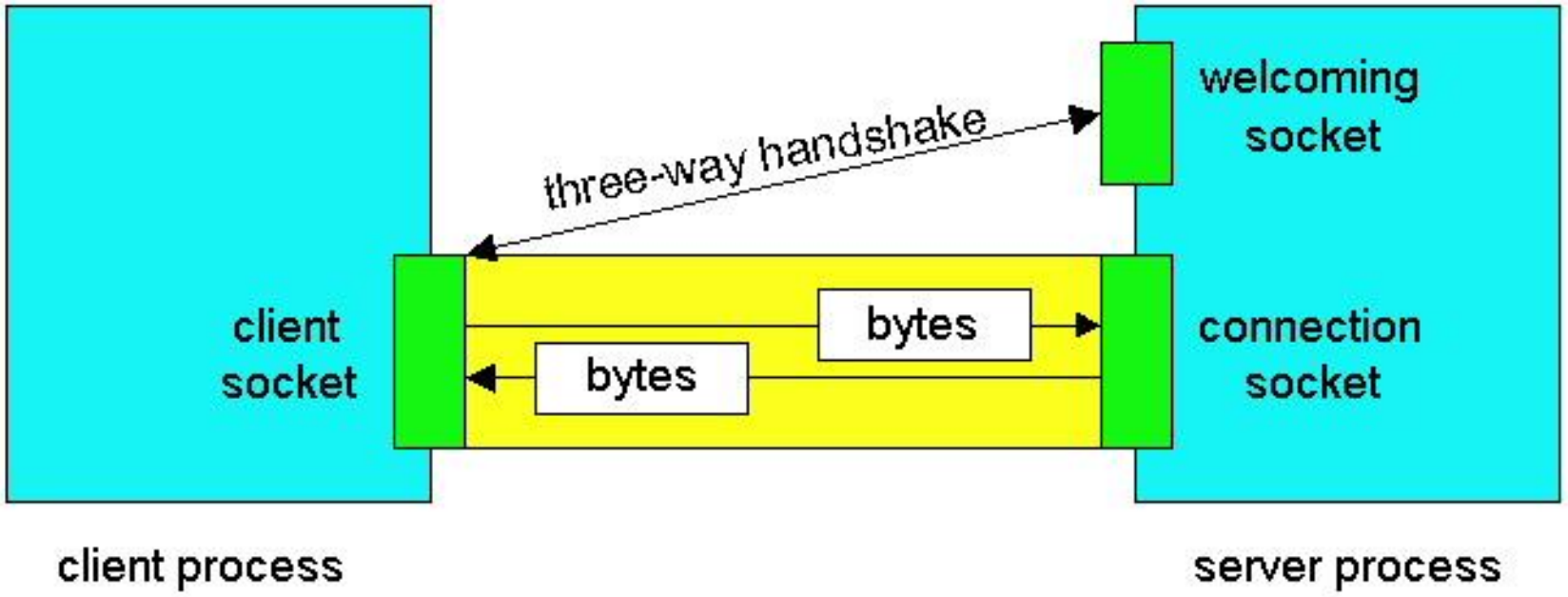


Flujo de salida/escritura

Flujo de entrada/lectura

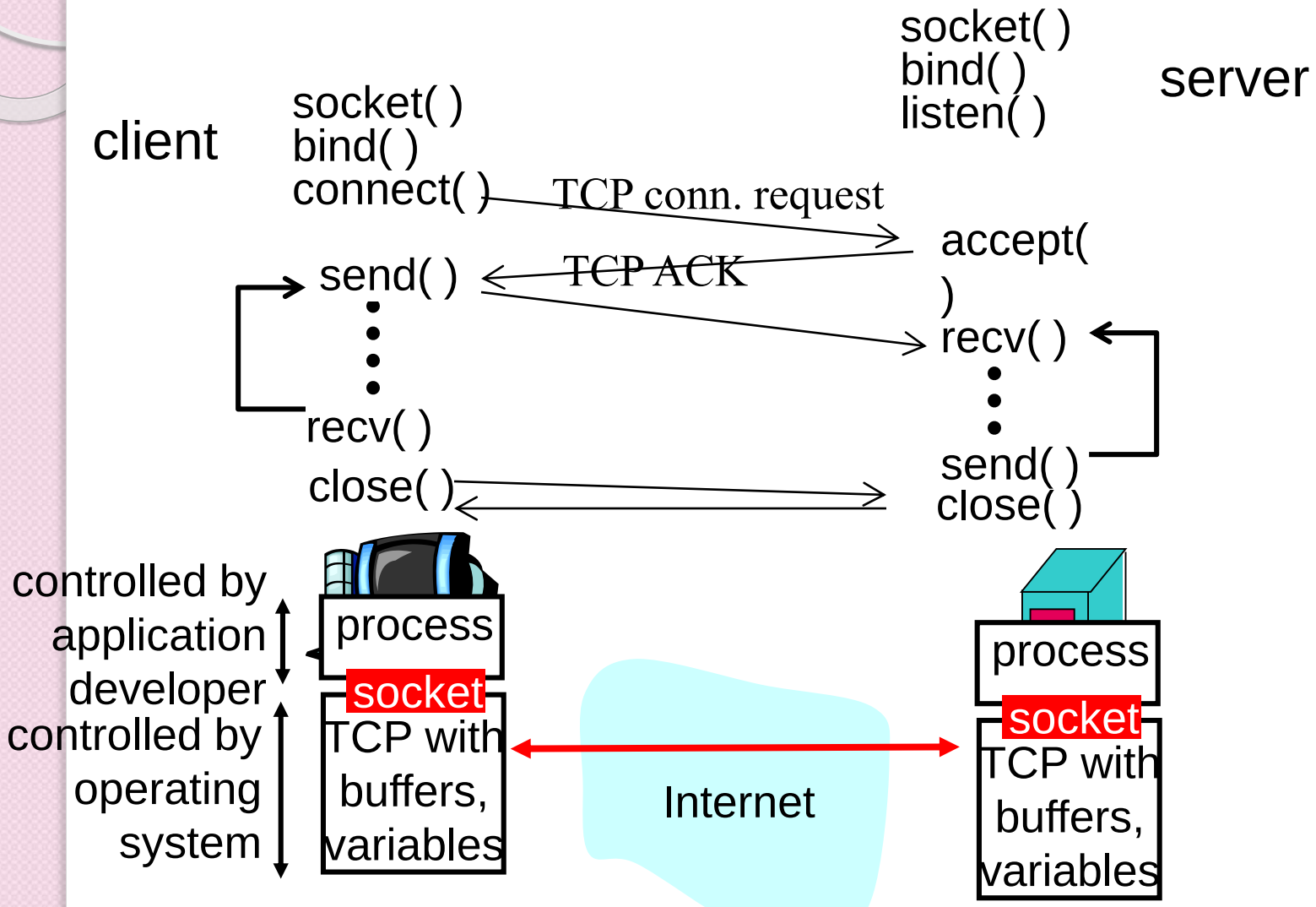
Puede ser un dominio como “mandroo.cs.mu.oz.au”

Sockets

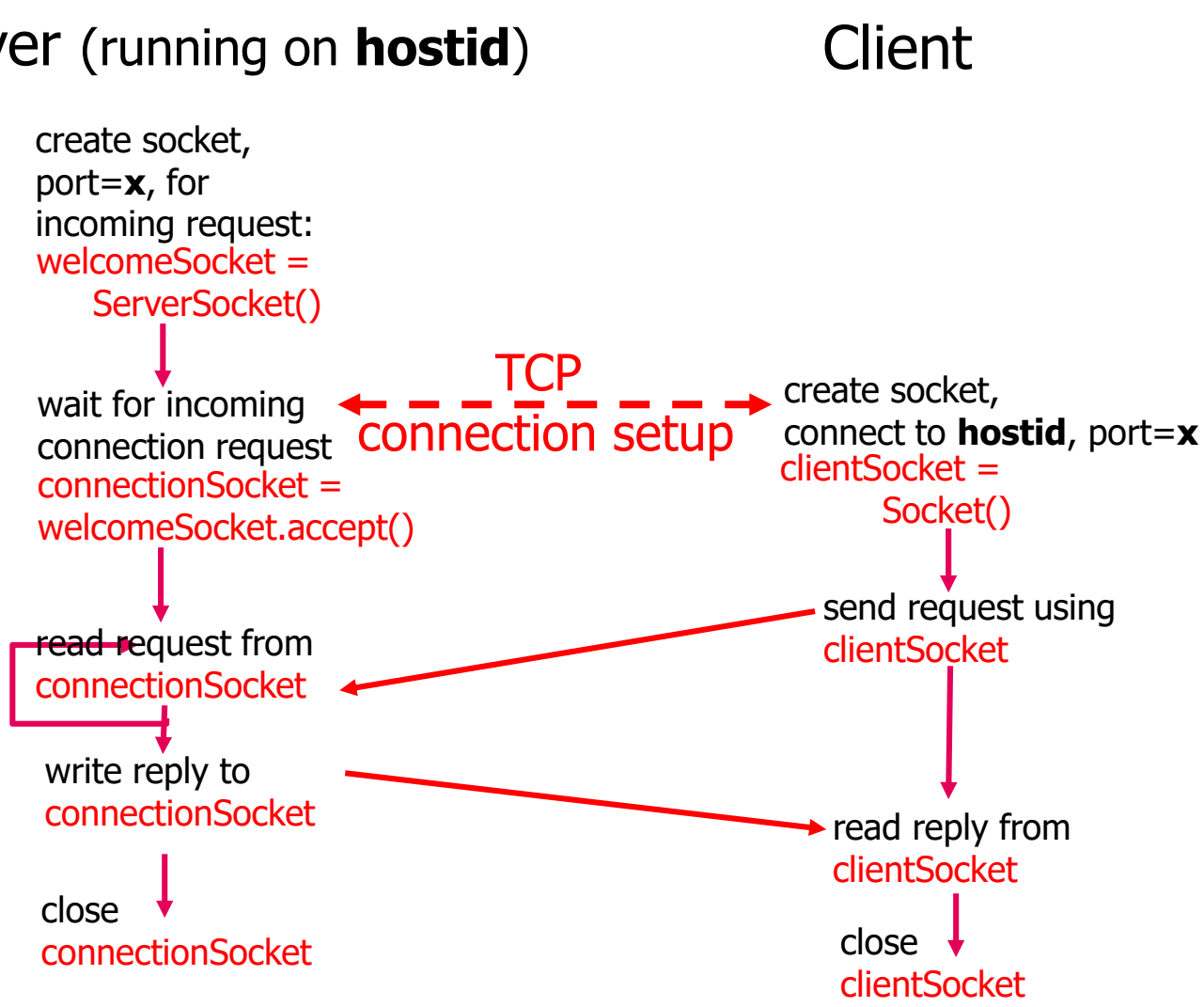

welcoming
socket
three-way handshake
client
socket
bytes
bytes
connection
socket
client process
server process

Programación de Sockets usando TCP

TCP service: transferencia confiable de flujos de bytes



Client/server socket interaction: TCP



Ejemplo de un Server

1. Crear el Server Socket:

```
ServerSocket server;
DataOutputStream os;
DataInputStream is;
server = new ServerSocket( PORT );
```

2. Espera solicitudes de clientes:

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

3. Crea flujos de I/O para comunicarse con el cliente

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

4. Realiza comunicación con un cliente

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

5. Cierra el socket: `client.close();`

Ejemplo de un Cliente

1. Crear un objeto de Socket:

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

2. Crea flujos de I/O para comunicarse con el servidor.

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

3. Realiza I/O o comunicación con el server:
   - Receive data from the server:

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

   - Send data to the server:

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

4. Cierra el socket cuando termina:

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

Un Server Simple

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

    public class SimpleServer
    {
      public static void main(String args[]) throws IOException
      {
        // Registrar el servicio en el puerto 1234
        ServerSocket s = new ServerSocket(1245);
        //Espera y acepta conexiones
        Socket s1 = s.accept();
        //Obtiene un flujo de comunicación asociado con el socket
        OutputStream s1out = s1.getOutputStream();
        DataOutputStream dos = new DataOutputStream (s1out);
        //Envia un mensaje
        dos.writeUTF("Hola que tal");
        //Cierra la conexión, pero no el socket del servidor
        dos.close();
        s1out.close();
        s1.close();
      }

}
```

Un Cliente Simple

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

public class SimpleClient
{
  public static void main(String args[]) throws IOException
  {
    //Abrir una conexión al server en el puerto 1234
    Socket s1 = new Socket("localhost",1245);
    //Obtener un manejador de flujo de entrada del socket y leer la entrada
    InputStream s1In = s1.getInputStream();
    DataInputStream dis = new DataInputStream(s1In);
    String st = new String (dis.readUTF());
    System.out.println(st);
    //Cerrar la conexion
    dis.close();
    s1In.close();
    s1.close();
  }
}
```

Ejecución

- Ejecutar Server en el localhost
  - java SimpleServer

- Ejecutar el Client en cualquier máquina:
  - java SimpleClient

    Hola que tal

- Si se ejecuta el cliente cuando el server no está escuchando:
  - java SimpleClient

```
Exception in thread "main" java.net.ConnectException: Connection refused
        at java.net.PlainSocketImpl.socketConnect(Native Method)
        at java.net.PlainSocketImpl.doConnect(PlainSocketImpl.java:320)
        at java.net.PlainSocketImpl.connectToAddress(PlainSocketImpl.java:133)
        at java.net.PlainSocketImpl.connect(PlainSocketImpl.java:120)
        at java.net.Socket.<init>(Socket.java:273)
        at java.net.Socket.<init>(Socket.java:100)
        at SimpleClient.main(SimpleClient.java:6)
```

Servidor Web

- Maneja solamente una petición HTTP
- Acepta y parsea la petición HTTP
- Obtiene el archivo requerido del sistema de archivos del servidor
- Crea un mensaje de respuesta HTTP, el cual consiste del archivo precedido por líneas de cabecera.
- Envía la respuesta directamente al cliente.

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

public class WebServer
{

  public static void main(String[] args)
  {
     String requestMessageLine;
     String fileName;
     try
      {
        ServerSocket listenSocket = new ServerSocket(8000);
        Socket connectionSocket = listenSocket.accept();

        BufferedReader inFromClient = new BufferedReader(
                new InputStreamReader(connectionSocket.getInputStream()));

        DataOutputStream outToClient = new
DataOutputStream(connectionSocket.getOutputStream());

        requestMessageLine = inFromClient.readLine();
        StringTokenizer tokenizedLine = new StringTokenizer(requestMessageLine);

                if (tokenizedLine.nextToken().equals("get"))
                {
                 fileName = tokenizedLine.nextToken();
                        if (fileName.startsWith("/") == true )
                          fileName = fileName.substring(1);
```

```
File file = new File(fileName);

int numOfBytes = (int) file.length();
FileInputStream inFile = new FileInputStream (fileName);
byte[] fileInBytes = new byte[numOfBytes];
inFile.read(fileInBytes);
outToClient.writeBytes("HTTP/1.1 200 Document Follows\r\n");

if (fileName.endsWith(".jpg"))
        outToClient.writeBytes("Content-Type: image/jpeg\r\n");

if (fileName.endsWith(".gif"))
        outToClient.writeBytes("Content-Type: image/gif\r\n");
```

```
 outToClient.writeBytes("Content-Length: " + numOfBytes
+ "\r\n");
 outToClient.writeBytes("\r\n");
 outToClient.write(fileInBytes, 0, numOfBytes);
 connectionSocket.close();
}
else System.out.println("Bad Request Message");
}
catch (IOException e)
{
        e.printStackTrace();
}
}

}
```

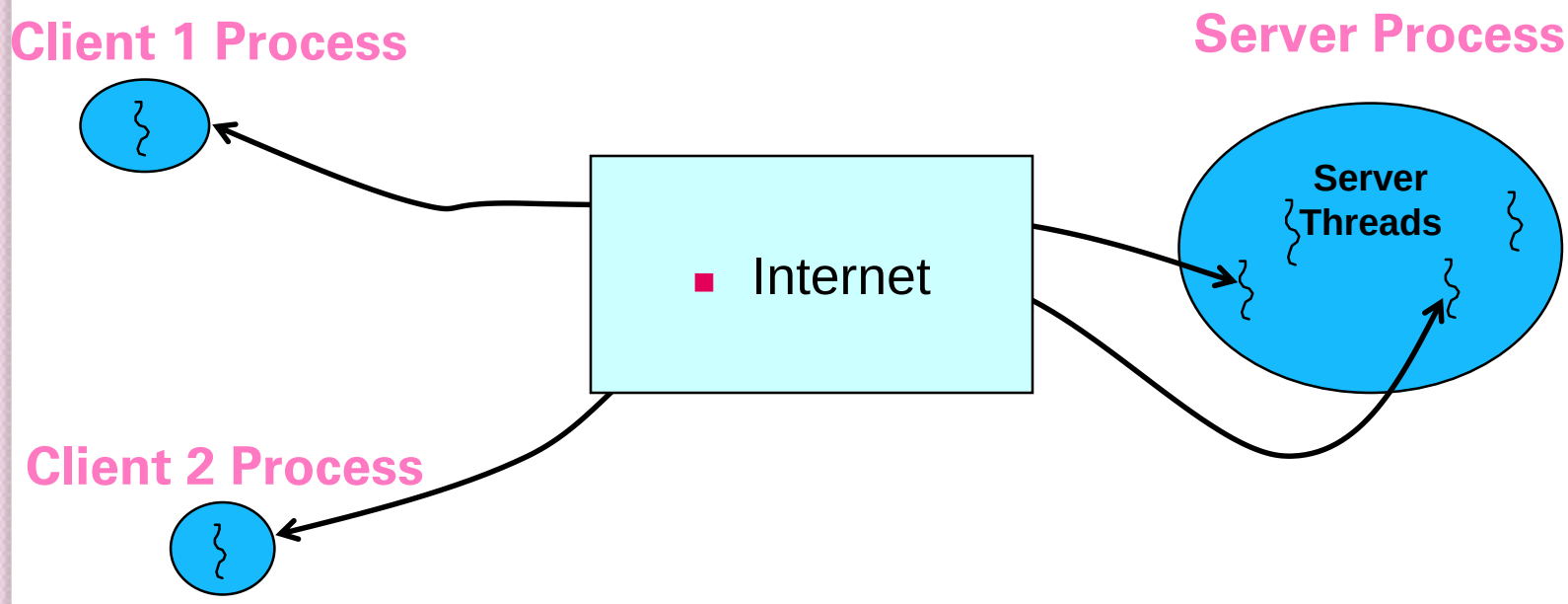
Socket Exceptions

```
try {
    Socket client = new Socket(host, port);
    handleConnection(client);
}
catch(UnknownHostException uhe) {
    System.out.println("Unknown host: " + host);
    uhe.printStackTrace();
}
catch(IOException ioe) {
System.out.println("IOException: " + ioe);
    ioe.printStackTrace();
}
```

Servidor en un Ciclo: Siempre en Ejecución

```
import java.net.*;
import java.io.*;
public class SimpleServerLoop
{
  public static void main(String args[]) throws IOException
  {
    ServerSocket s = new ServerSocket(1234);
    while(true)
    {
          Socket s1=s.accept();
        OutputStream s1out = s1.getOutputStream();
          DataOutputStream dos = new DataOutputStream (s1out);
          dos.writeUTF("Hola");
          dos.close();
          s1out.close();
          s1.close();
    }
  }
}
```

Multithreaded Server: Para servir a múltiples clientes concurrentemente



Servidor de Eco con Hilos

```
import java.io.*;
import java.net.*;
import java.util.*;
public class servidorEcoconHilos
{
        public static void main(String[] args) {
          try {
            int i = 1;
            ServerSocket s = new ServerSocket(8189);
            while(true)
             {
                  Socket entrante = s.accept();
                  System.out.println("generando hilo " + i);
                  Runnable r = new ManejadorHilos(entrante, i);
                  Thread t = new Thread(r);
                  t.start();
                  i++;
             }
          }
```

```
catch(IOException e) {
                        e.printStackTrace();
                }
            }
}
//Clase para el manejo de hilos
Class ManejadorHilos implements Runnable
{
   public ManejadorHilos(Socket i, int c)
   {
      entrante = i;
      contador = c;
    }

   public void run()
   {
      try {
            try {
                InputStream secuenciaEntrada = entrante.getInputStream();
                OutputStream secuenciaSalida = entrante.getOutputStream();
                 Scanner in = new Scanner(secuenciaEntrada);
                 PrintWriter out = new PrintWriter(secuenciaSalida, true);
                 out.println("Escriba ADIOS para salir");
```

```
//Reproducir la entrada del cliente
boolean terminado = false;
while(!terminado && in.hasNextLine()) {
   String linea = in.readLine();
   out.println("Eco:  " + linea );
   if (linea.trim().equals("ADIOS"))
                terminado = true;
}
}
finally
{
   entrante.close();
}
Catch(IOException e)
{
    e.printStackTrace();
}
}//End del run

private Socket entrante;
private int contador;
}
```

Ejercicio de sección

- Implementar un servidor de Web con sockets en Java que esté prendido en el puerto 8000, que este siempre prendido y que maneje múltiples conexiones a través de hilos en Java.
- Que el servidor entregue documentos y recursos con la extensión jpeg, gif, mpeg, vms, doc, pdf, y html.

¡GRACIAS!