Cliente/Servidor en Java

Grupo ARCOS

Desarrollo de Aplicaciones Distribuidas

Ingeniería Informática

Universidad Carlos III de Madrid

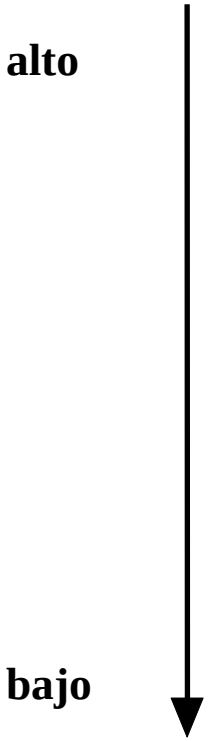
Contenidos

1. Introducción:
   1. Paradigma cliente/servidor
   2. Entorno de programación Java
2. Cliente/servidor en Java
   1. Introducción
   2. Ejemplo con sockets

Contenidos

1. **Introducción:**
   1. **Paradigma cliente/servidor**
   2. Entorno de programación Java
2. Cliente/servidor en Java
   1. Introducción
   2. Ejemplo con sockets

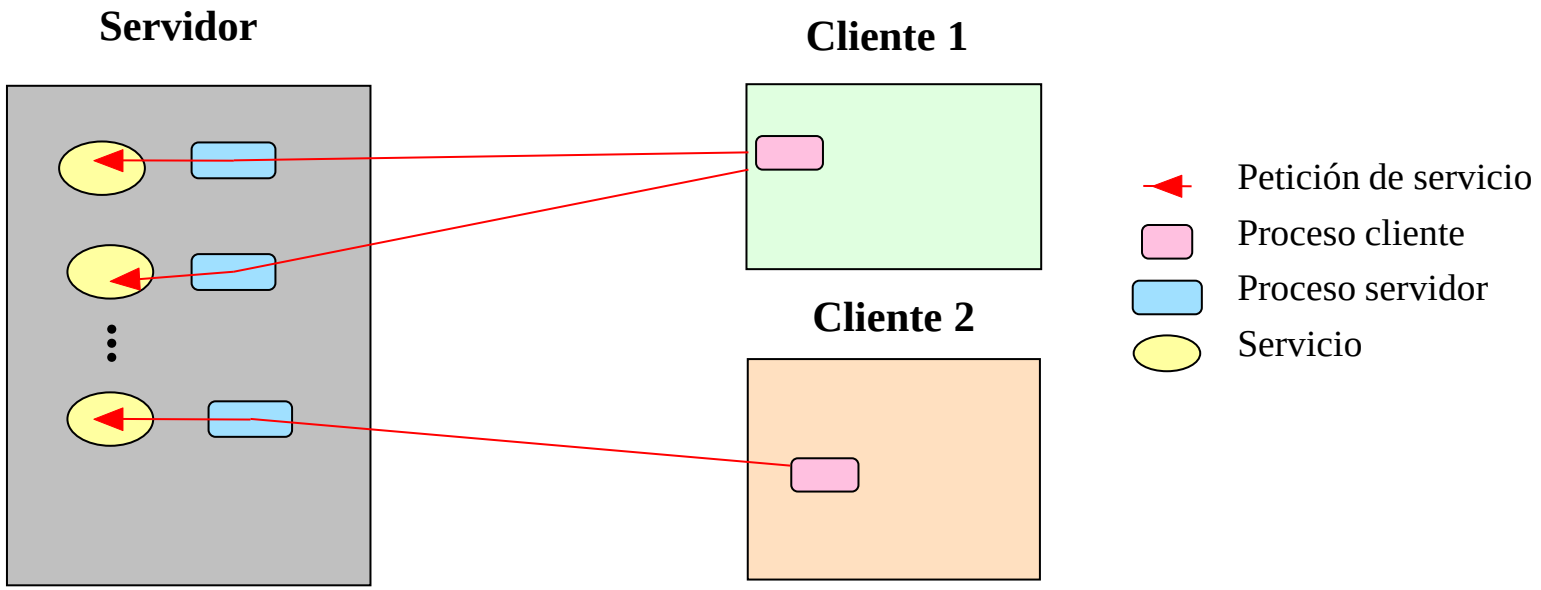
Paradigmas cliente/servidor y P2P



Espacio de objetos, aplicaciones colaborativas
Servicios de red, *object request broker*, agentes móviles
procedimientos remotos, métodos remotos
Cliente-servidor, *peer-to-peer*
Paso de mensajes

Paradigma cliente-servidor

- Asigna roles diferentes a dos procesos que colaboran:
  - Servidor: es el proveedor del servicio.
    Espera de forma pasiva la llegada de peticiones.
  - Cliente: invoca peticiones al servidor y aguarda su respuesta.

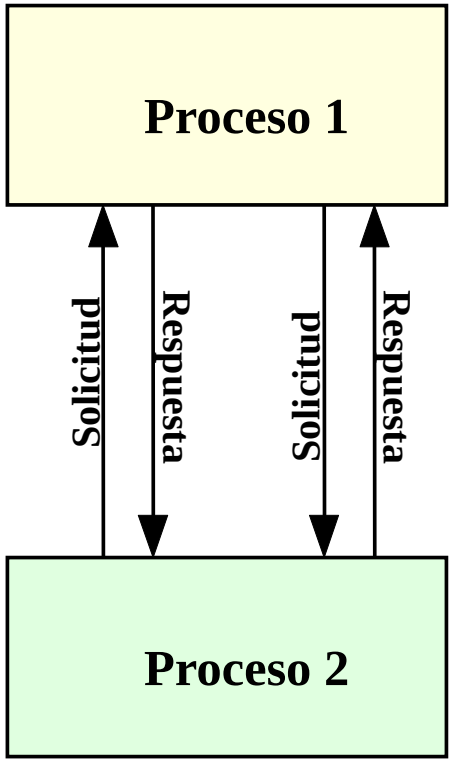


Paradigma cliente-servidor

- Proporciona una abstracción eficiente para facilitar los servicios de red.
- La asignación de roles asimétricos simplifica la sincronización.
- Paradigma adecuado para servicios centralizados.
  - Ejemplos: servicios de internet como HTTP, FTP, DNS, finger, etc.
- Implementación mediante *sockets*, llamada a procedimientos remotos (RPC) o invocación de métodos remotos (RMI).

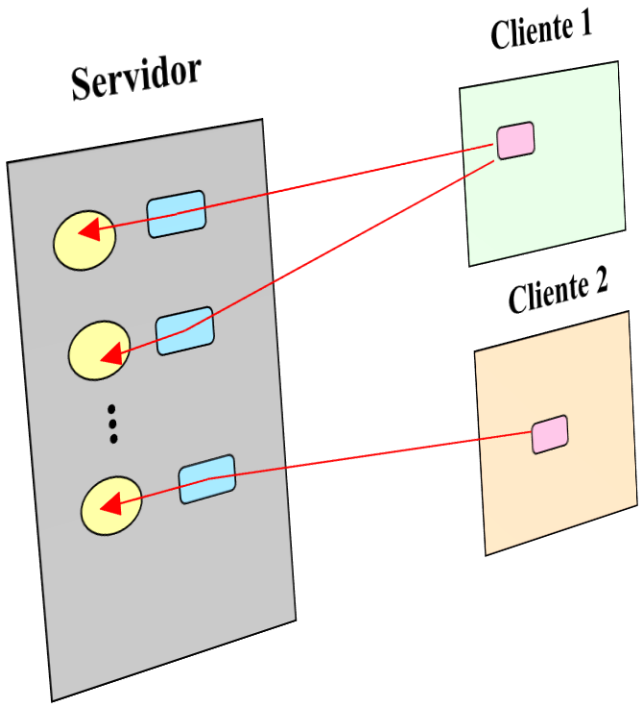
Paradigma *peer-to-peer*

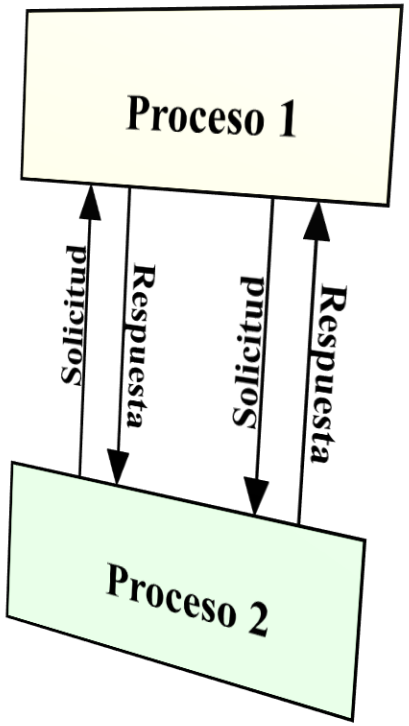
- Asignación de roles simétrica:
  - Los procesos participantes tienen el mismo papel
  - un mismo proceso puede actuar tanto como cliente como servidor
- Los recursos computacionales y los servicios son intercambiados entre los computadores.
  - Ejemplo: servicios de intercambio de ficheros como Gnutella



Paradigma *híbridos* (c/s + p2p)

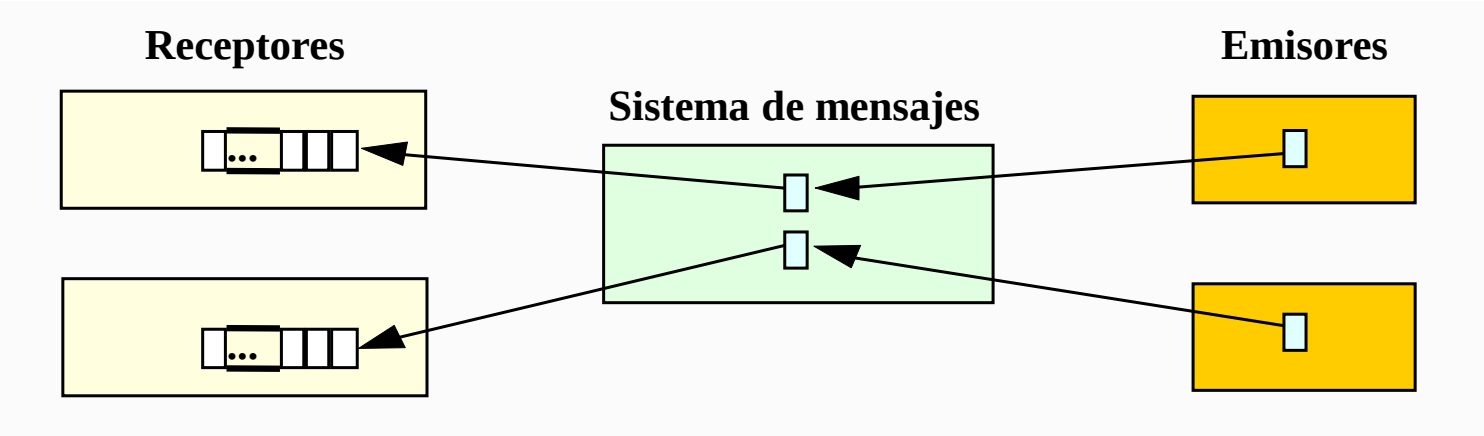
- Modelos híbridos cliente-servidor y *peer-to-peer*
  - Ejemplo: servicio de intercambio de ficheros Napster





Paradigma del sistema de mensajes

- También denominado *middleware* orientado a mensajes (MOM)
- El sistema de mensajes actúa de intermediario entre los procesos que se comunican
- Proceso:
  - Emisión al sistema de mensajes
  - Almacenamiento en la cola asociada al receptor
  - Envío al proceso receptor



Paradigma del sistema de mensajes

- Comunicación asíncrona y desacoplada.
- Una vez que el emisor envía el mensaje al sistema de mensajes, queda libre para realizar otra tarea.
- Existen dos subclases de sistema de mensajes: el punto a punto y el publicación/suscripción.
- Sistema de mensajes punto a punto:
  - El sistema de mensajes proporciona el *middleware* que gestiona cada cola de mensajes
  - Envío y recepción están desacopladas: uso del *threads* o procesos hijo

Paradigma del sistema de mensajes

- Sistema de mensajes publicación/suscripción:
  - Cada mensaje se asocia con un determinado evento.
  - Pasos:
    1. Cada participante se subscribe a los mensajes asociados a cada evento (operación *suscribir*).
    2. Cuando el evento ocurre el *middleware* distribuye el mensaje a todos los subscriptores (operación *publicar*).
  - Los eventos pueden ser iniciados por cualquier participante.
- Ejemplos de servicio:
  - *MQ*Series* de IBM
  - *Microsoft's Message Queue* (MSMQ)
  - *Java's Message Service* (JMS)

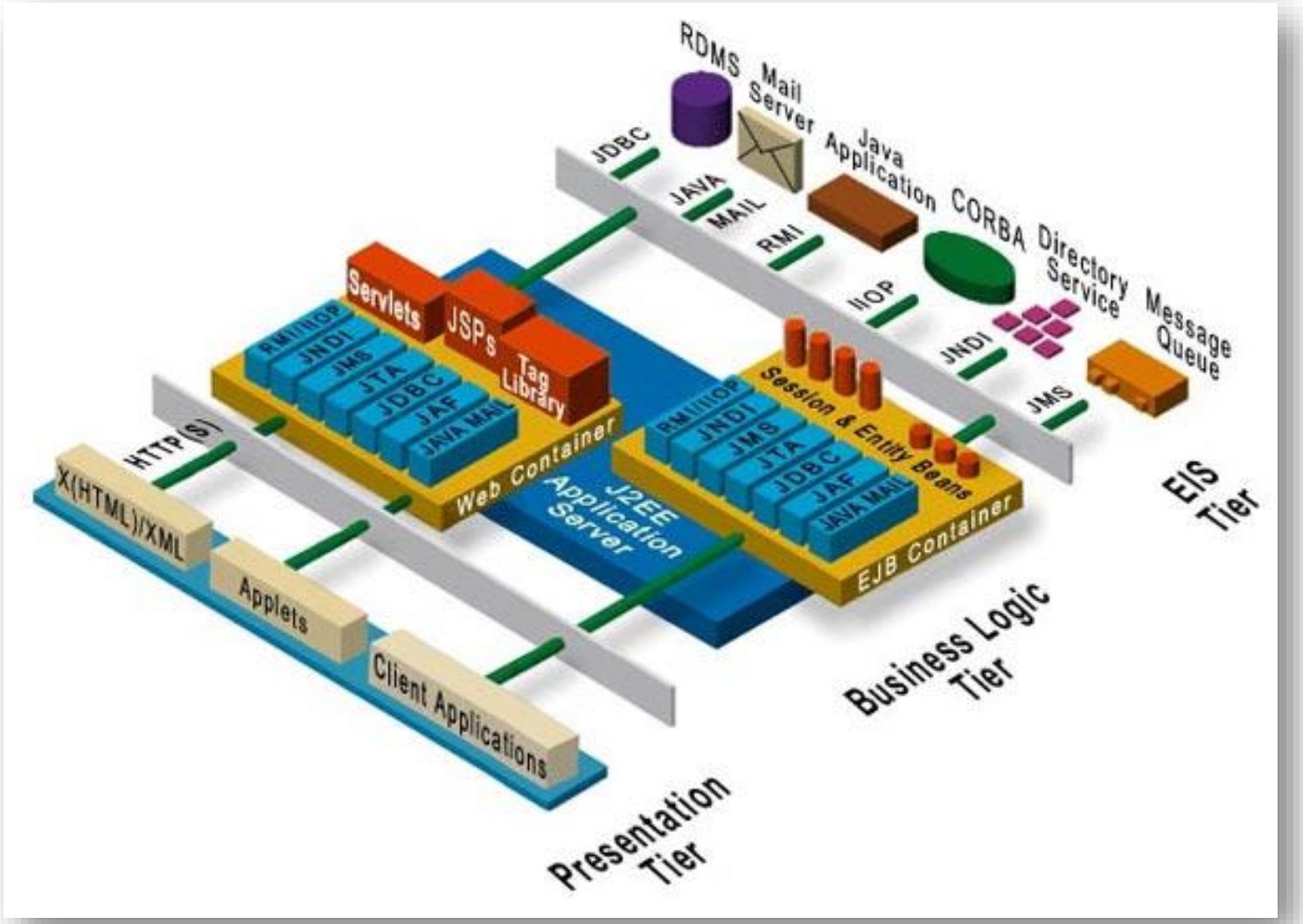
Contenidos

1. **Introducción:**
   1. Paradigma cliente/servidor
   2. **Entorno de programación Java**
2. Cliente/servidor en Java
   1. Introducción
   2. Ejemplo con sockets

Java: características

- Lógica basada en 3 capas:
  - Presentación
    - Interfaz con el usuario
  - Lógica de negocio
    - Programa que responde a las peticiones del usuario
  - Lógica de acceso a datos
    - Interfaz con el almacenamiento de datos (ej.: base de datos)

Java: características



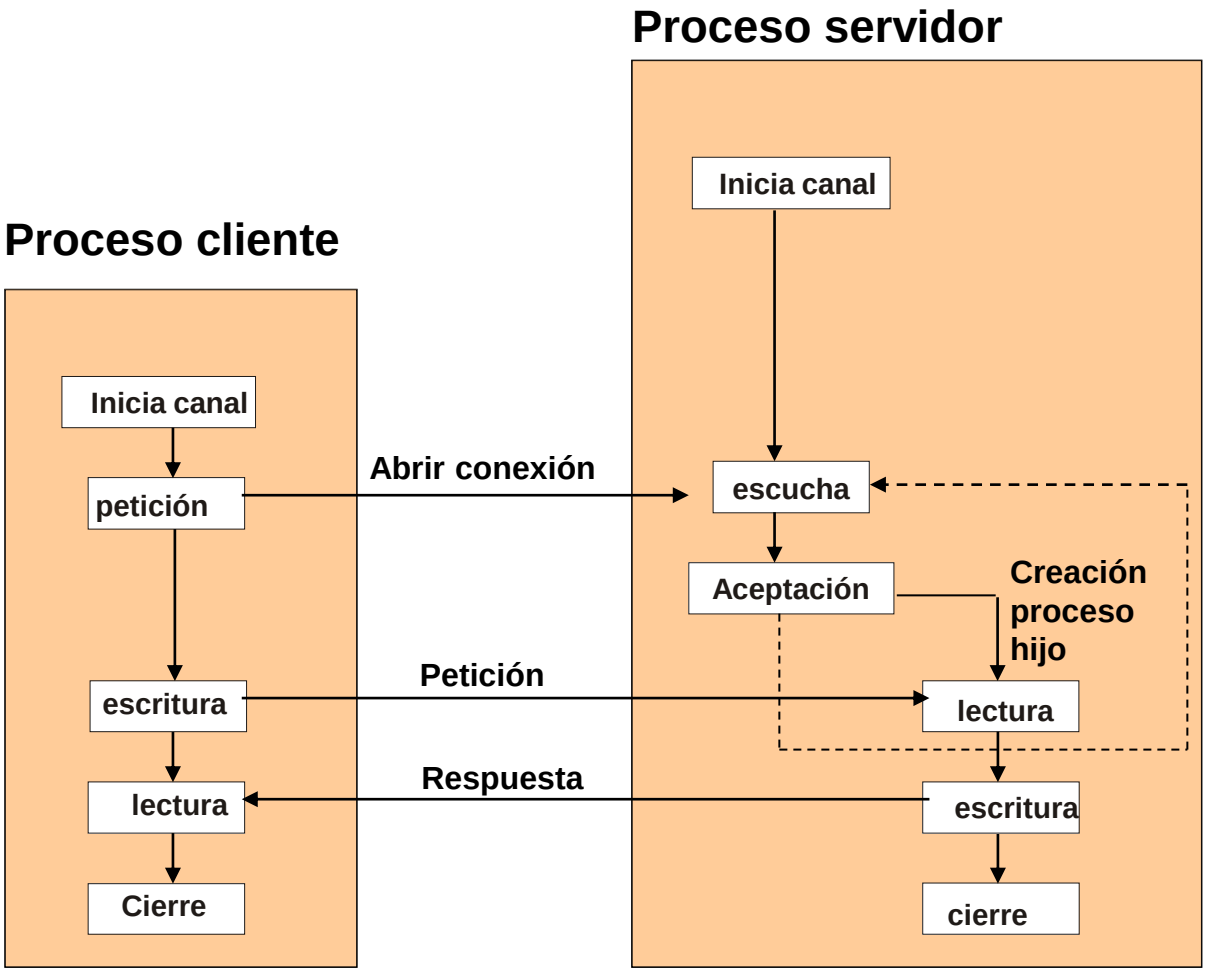
Contenidos

1. Introducción:
   1. Paradigma cliente/servidor
   2. Entorno de programación Java
2. **Cliente/servidor en Java**
   1. **Introducción**
   2. Ejemplo con sockets

Paradigma cliente-servidor

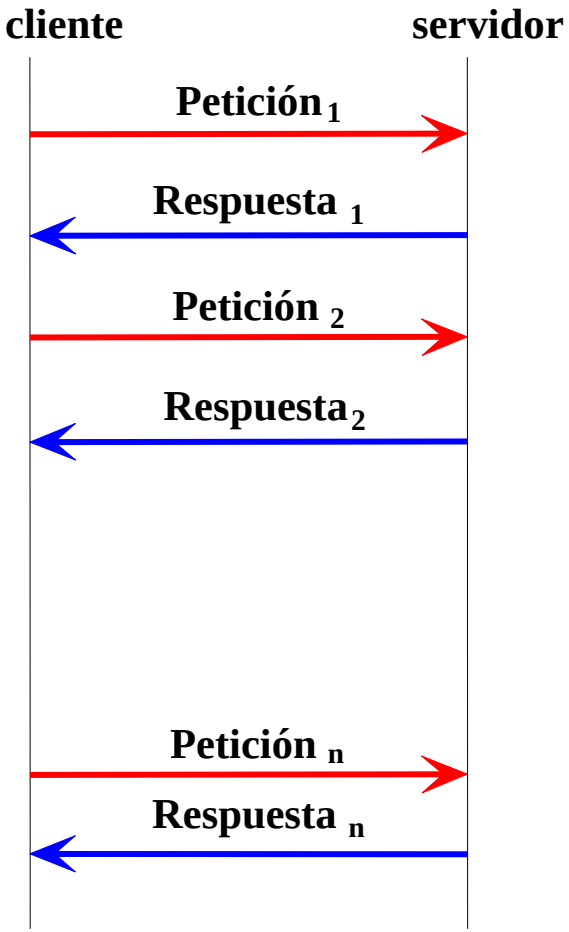
- Los procesos desempeñan dos roles asimétricos.
- Representa el paradigma de sistemas distribuidos con una mayor difusión.
- Acceso (por parte de los clientes) de servicios de red.
- Flujo de ejecución del servidor:
  1. Inicio servicio.
  2. Espera hasta aceptar petición de un cliente.
  3. Inicia sesión de servicio con el cliente.
  4. Vuelta al paso 2.

Paradigma cliente-servidor



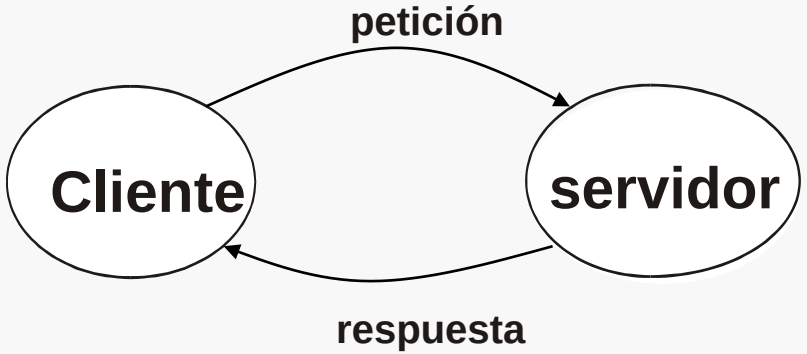
Paradigma cliente-servidor

- Protocolo de servicio:
  - Localización del servicio.
  - Comunicación entre procesos.
    - Sin conexión
    - Orientados a conexión
  - Sincronización de eventos.
  - Representación de datos.

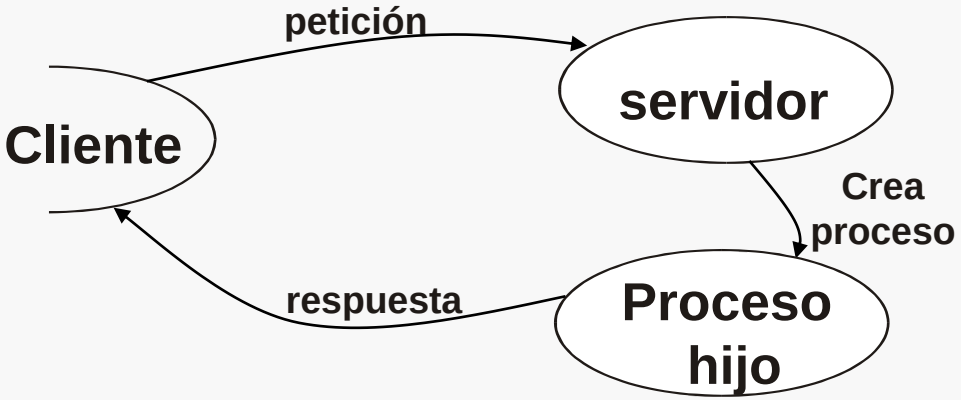


Paradigma cliente-servidor

- Gestión de la sesión por parte del servidor:
  - Servidor iterativo

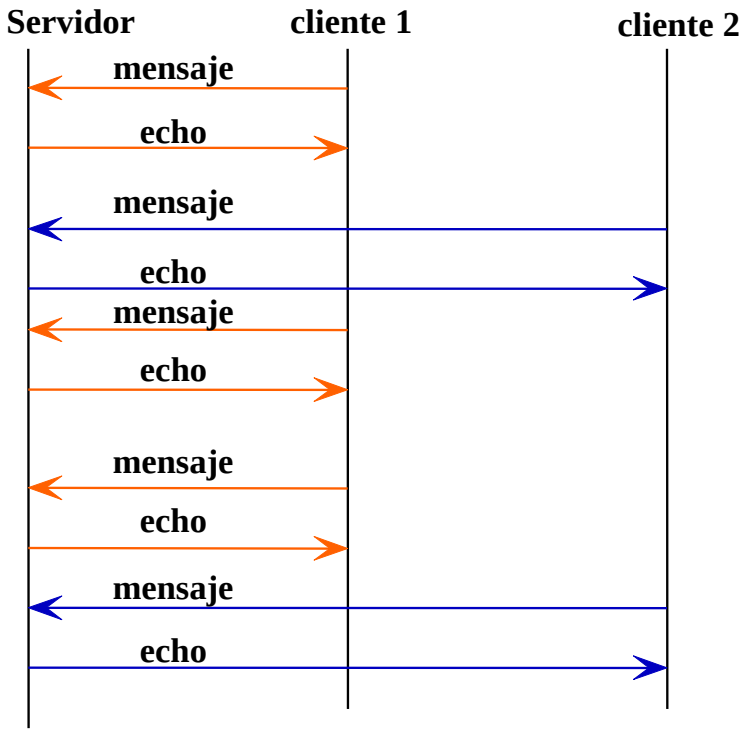


  - Servidor concurrente:
    - Procesos pesados
    - Procesos ligeros
    - **IPC** asíncronas

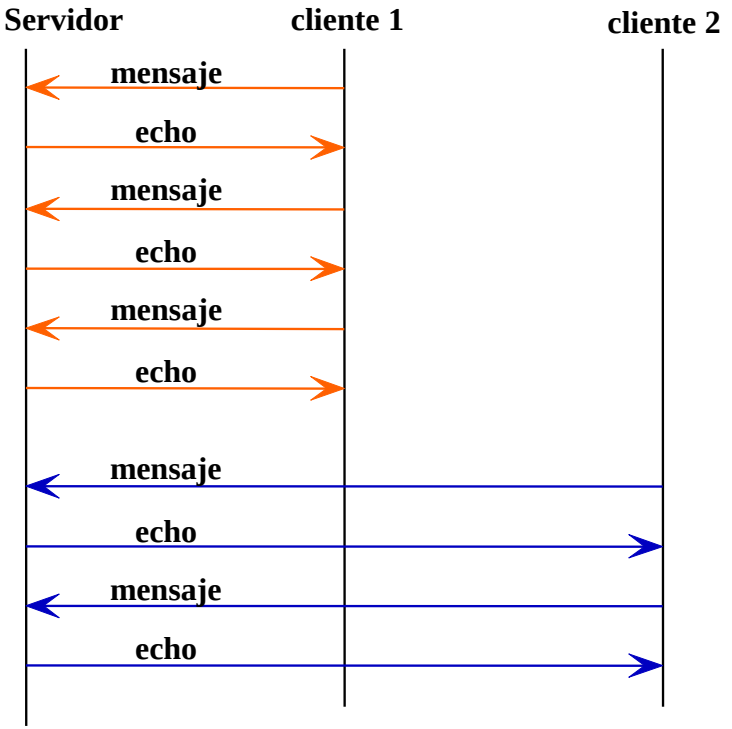


Paradigma cliente-servidor

- Servidor concurrente

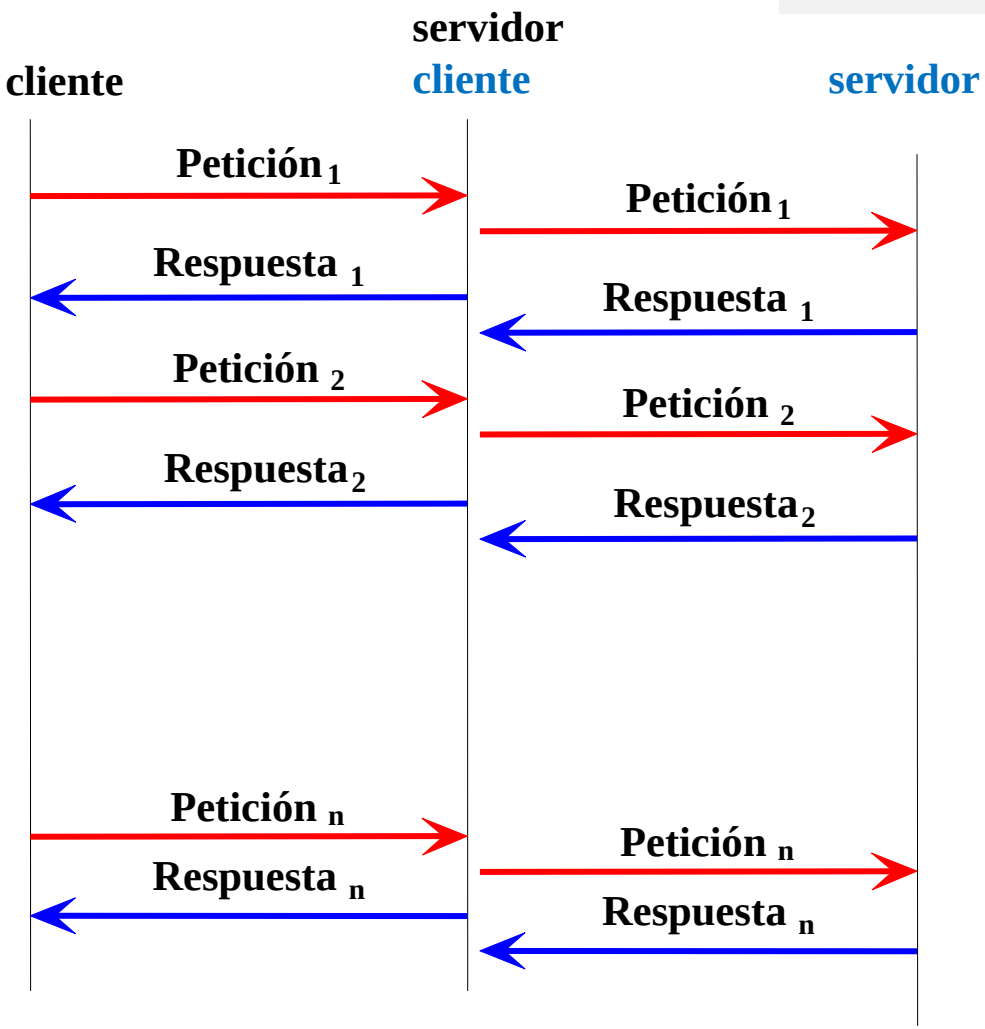


- Servidor secuencial



Paradigma cliente-servidor

- Arquitectura del *software*:
  - Presentación
  - Lógica de aplicación
  - Servicio (almacenamiento)



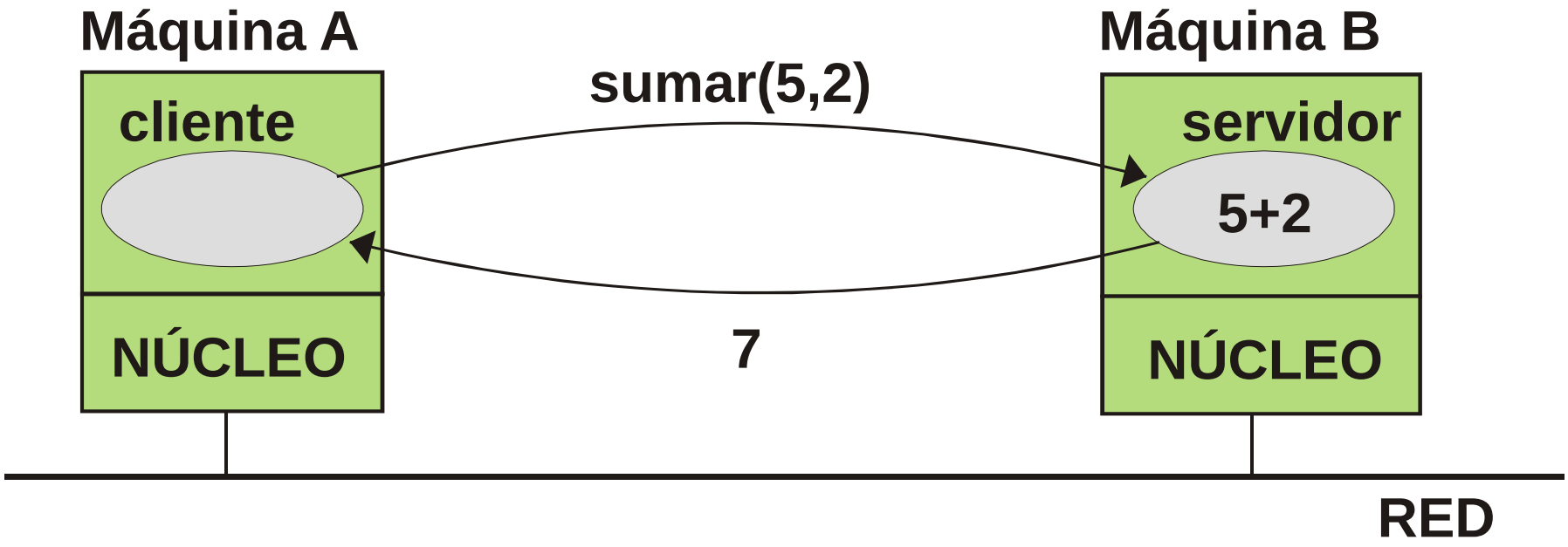
Tipos de servicios

- Sin estado
  - Ej.: daytime, echo, etc.
- Con estado
  - Con estado global
    - Ej.: *counter*
  - Con estado de sesión
    - Ej.: *ftp*
  - Servidor híbrido:
    información del estado se distribuye entre el servidor y el cliente.

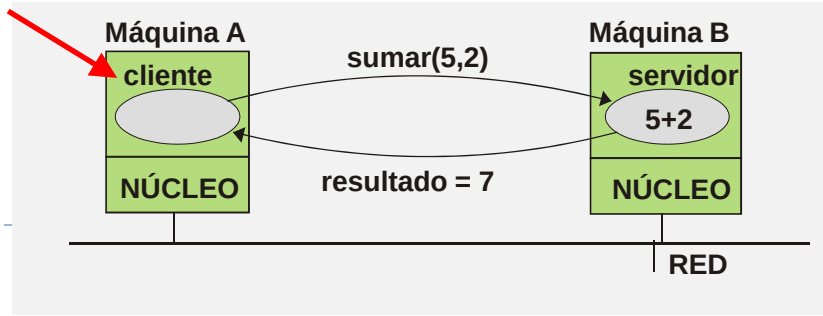
Contenidos

1. Introducción:
   1. Paradigma cliente/servidor
   2. Entorno de programación Java
2. **Cliente/servidor en Java**
   1. Introducción
   2. **Ejemplo con sockets**

Ejemplo (*streams*)

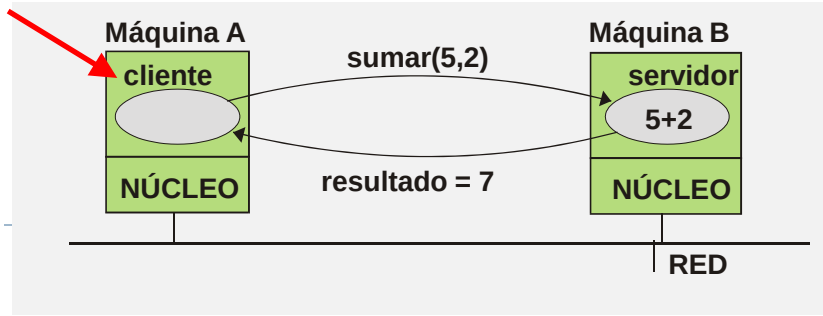


Client.java (1/2)



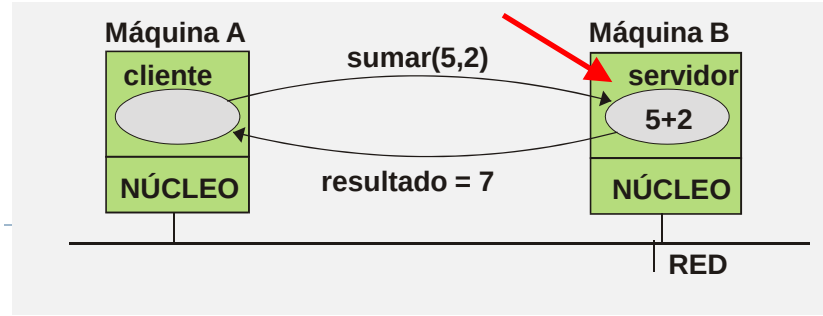

```
import java.io.* ;
import java.net.* ;

public class Client
{
   public static void main ( String [] args) {
      int  res;
      int  num[] = new int[2];

      if (args.length != 1) {
         System.out.println("Uso: cliente <host>");
         System.exit(0);
      }
      try {
         String host = args[0];
         Socket sc = new Socket(host, 2500); // socket servidor
         OutputStream ostream = sc.getOutputStream();
         ObjectOutput s = new ObjectOutputStream(ostream);
```

Client.java (2/2)



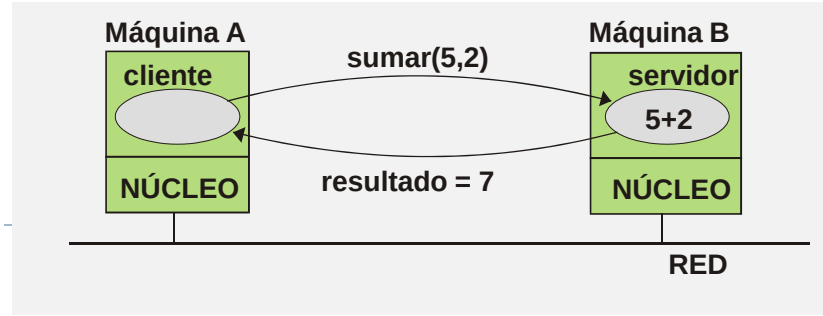

```
            num[0] = 5;   num[1] = 2;  //prepara la petición
            s.writeObject(num);
            s.flush();

        DataInputStream istream = new DataInputStream(sc.getInputStream());
            res = istream.readInt();
            sc.close();

            System.out.println("La suma es " + res);

        } catch (Exception e) {
            System.err.println("excepcion " + e.toString() );
            e.printStackTrace() ;
        }
    }
}
```

Server.java (1/2)




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

public class Server
{
   public static void main ( String [] args) {
      ServerSocket serverAddr = null;
      Socket sc = null;
      int num[] ;
      int res;
      try {
         serverAddr = new ServerSocket(2500);
      }
      catch (Exception e){
         System.err.println("Error creando socket");
      }
```

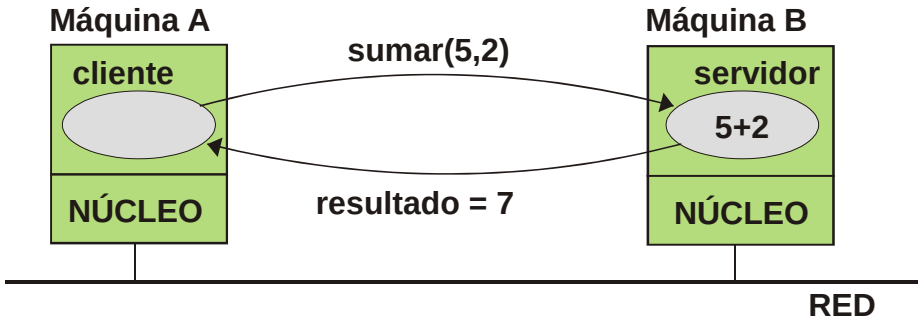
Server.java (2/2)




```
    while (true) {
      try {
          sc = serverAddr.accept(); // esperando conexión
          InputStream istream = sc.getInputStream();
          ObjectInput in = new ObjectInputStream(istream);
          num = (int[]) in.readObject();
          res = num[0] + num[1]; Thread.sleep(2000);
       DataOutputStream ostream = new DataOutputStream(sc.getOutputStream());
          ostream.writeInt(res);
          ostream.flush();
          sc.close();
      } catch(Exception e) {
          System.err.println( "excepcion " + e.toString() );
          e.printStackTrace() ;
      } // try

    } // while
  } // main
} // servidor
```

Compilación del ejemplo

guernika.lab.inf.uc3m.es



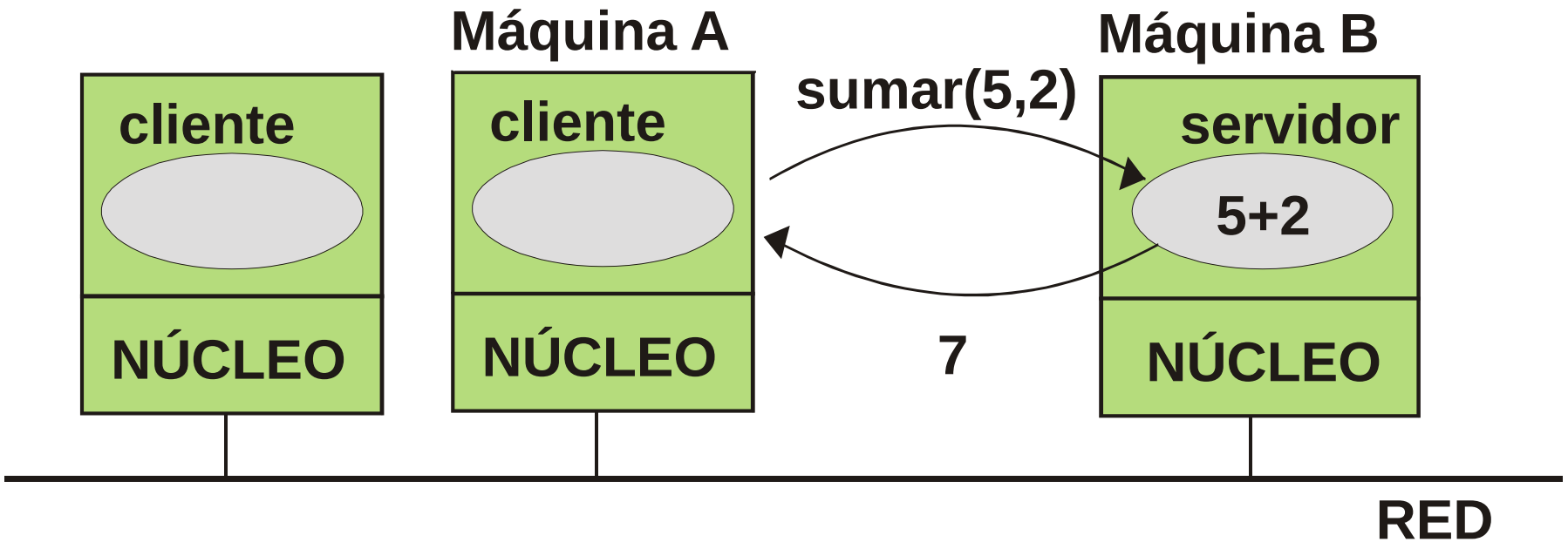

```
# javac  -cp /usr/lib/jvm/java-1.4.2-gcj-4.1-1.4.2.0/jre/lib/rt.jar  \
         -g  Client.java Server.java
```

Ejecución del ejemplo

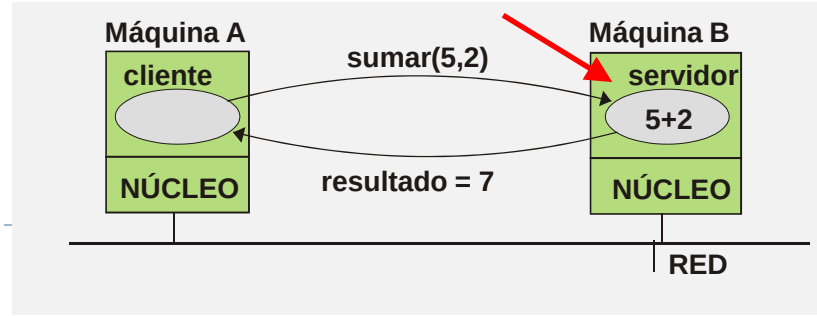
guernika.lab.inf.uc3m.es

```
# java Server &

# java Client
Uso: cliente <host>

# java Client  localhost
La suma es 7
```

Ejemplo 2 (*streams*)

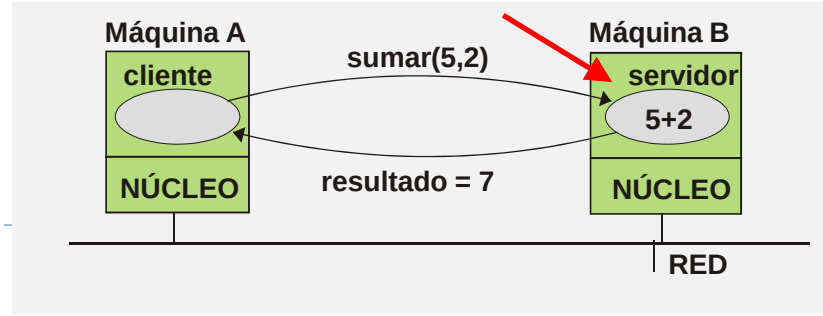


Server2.java (1/3)



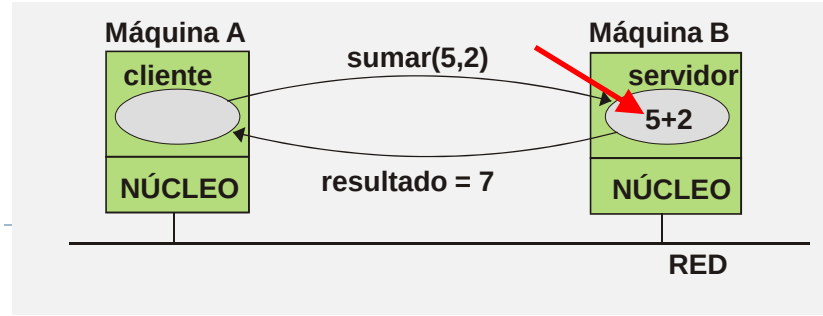

```
import java.io.* ;
import java.net.* ;
import java.lang.Thread ;

class clientHandler implements Runnable
{
   private Socket socket ;
   Thread t ;

   public clientHandler ( Socket socket ) {
      this.socket = socket ;
      this.t = new Thread(this) ;
      t.start() ;
   }
```

Server2.java (2/3)




```
   public void run () {
      int num[] ; int res;

      try {
            InputStream istream = socket.getInputStream();
            ObjectInput in = new ObjectInputStream(istream);
            num = (int[]) in.readObject();
            res = num[0] + num[1]; Thread.sleep(2000);
   DataOutputStream ostream = new DataOutputStream(socket.getOutputStream());
            ostream.writeInt(res);
            ostream.flush();
            socket.close();
      } catch (Exception e) {
         System.err.println("Error al operar con el cliente");
      }
   } // run
} // clientHandler
```

Server2.java (3/3)




```
public class Server2 {
   public static void main ( String [] args) {
      ServerSocket serverAddr = null;
      Socket sc = null;

      try {
         serverAddr = new ServerSocket(2500);
         while (true) {
           sc = serverAddr.accept(); // esperando conexión
           new clientHandler(sc) ;
         }
      } catch (Exception e) {
         System.err.println("excepcion " + e.toString() );
         e.printStackTrace() ;
      }
   } // main
} // servidor
```

clients.sh

```
#!/bin/sh
set -x

I=0
while [ $I -lt 10 ]; do
   java Client localhost &
   I=`expr $I + 1`
done
```

chmod a+x clients.sh

Ejecución del ejemplo

guernika.lab.inf.uc3m.es

```
# : servidor NO concurrente
# java Server &
# ./clients.sh
…
```

```
# : servidor SI concurrente
# java kill -9 %1
# java Server2 &
# ./clients.sh
…
```

Cliente/Servidor en Java

Grupo ARCOS

Desarrollo de Aplicaciones Distribuidas

Ingeniería Informática

Universidad Carlos III de Madrid