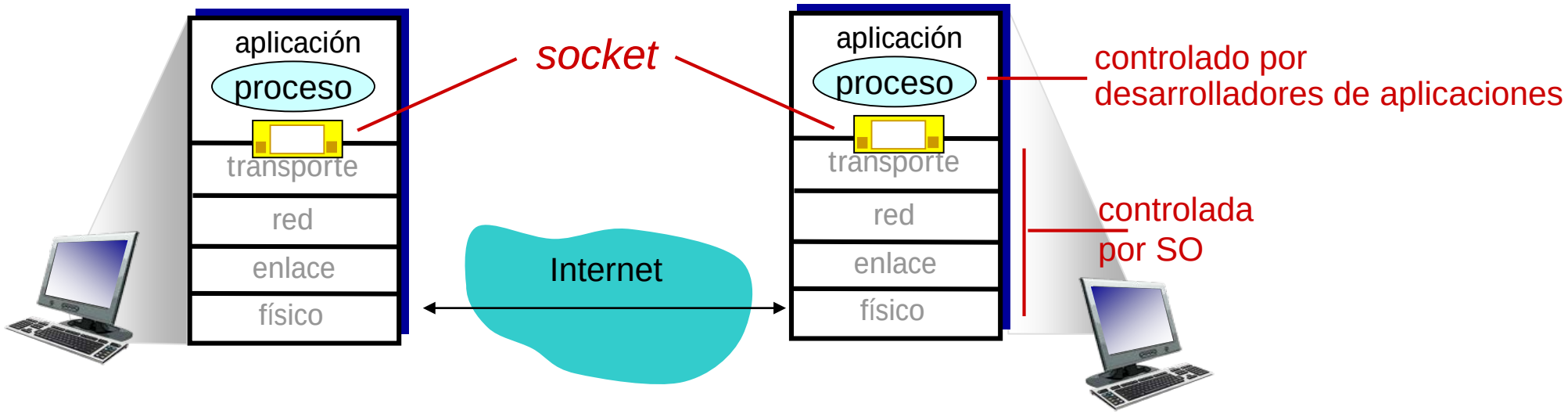
Programación de sockets

objetivo: aprender a construir aplicaciones cliente / servidor que se comunican usando sockets

socket: puerta entre proceso de aplicación y el protocolo de extremo final de transporte



Programación de sockets

Dos tipos de conectores para dos servicios de transporte:

- *UDP:* datagramas no fiable
- *TCP:* orientado a flujo fiable, byte

Ejemplo de aplicación:

1. El Cliente lee una línea de carácteres (datos) de su teclado y envía los datos al servidor.
2. El servidor recibe los datos y convierte los carácteres en mayúsculas.
3. El servidor envía los datos modificados al cliente.
4. El cliente recibe los datos modificados y muestra la línea en su pantalla.

High-level process

```
// Fire up connection
// to the server
socket()
connect()

// Exchange data
while (!done)
{
  send()
  recv()
}

// Shutdown
close()
```

Client program

```
// Initial socket setup
socket()
bind()
listen()
while (1)
{
  // Wait for new caller
  accept()

  // Exchange data
  while (!done)
  {
    recv()
    send()
  }
  // Disconnect
  close()
}
```

Server program

Programación de sockets *con UDP*

UDP: no "conexión" entre cliente y servidor

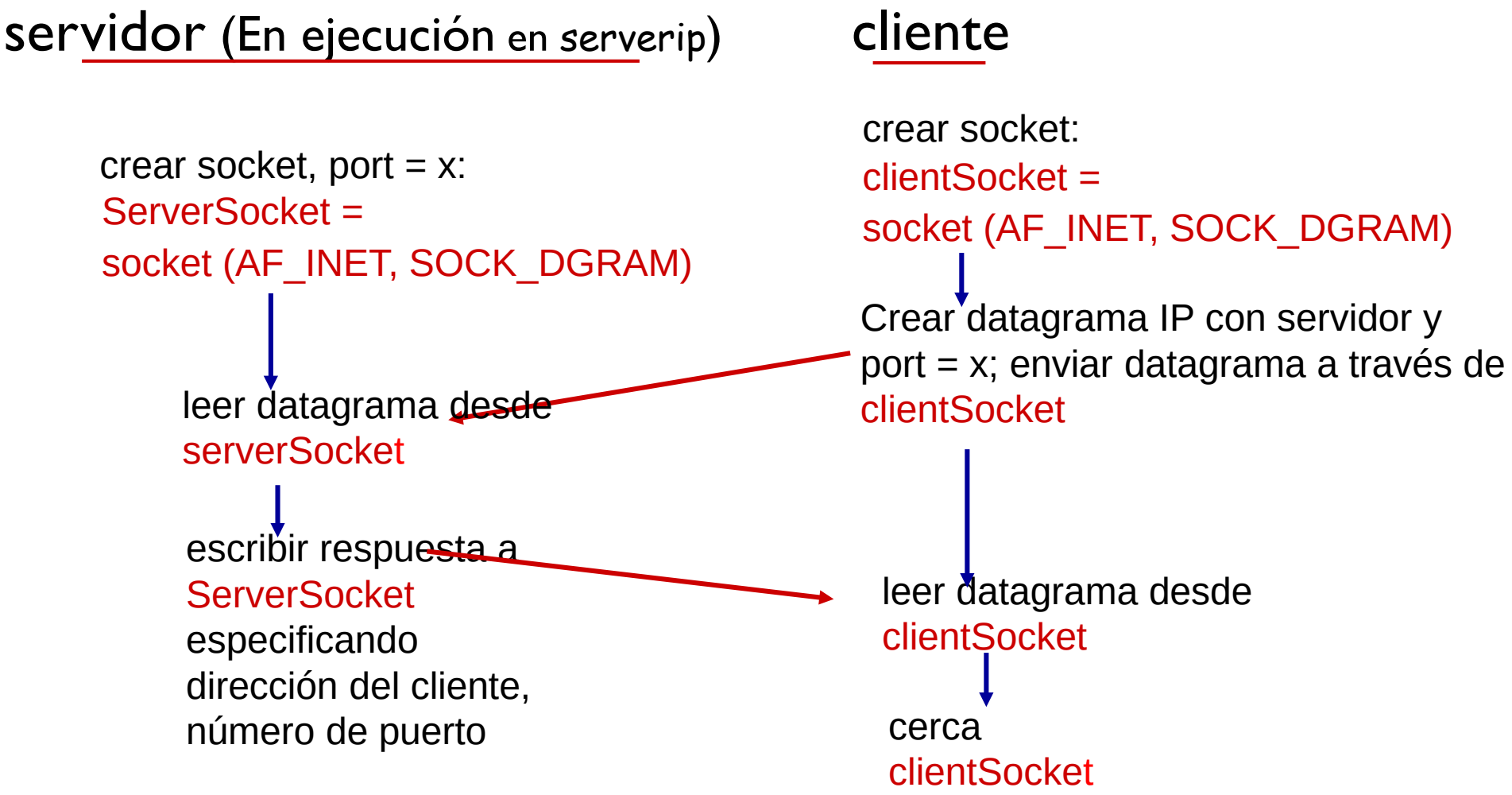
- sin estableciomiento de conexión antes de enviar datos
- El emisor establece explícitamente la dirección IP de destino y el número de puerto para cada paquete
- el receptor extrae la dirección IP del remitente y el número de puerto del paquete recibido

UDP: los datos transmitidos se pueden perder o pueden recibirse fuera de orden

Punto de vista de la aplicación:

- UDP proporciona transferencia *poco fiable* de grupos de bytes ("datagramas") entre el cliente y el servidor

Interacción socket Cliente/Servidor: UDP



Ejemplo: cliente UDP

Python UDPClient

```
from socket import *
serverName = 'hostname'
serverPort = 12000
clientSocket = socket (socket.AF_INET,
                       socket.SOCK_DGRAM)
frase = raw_input ('Entrada frase minúsculas:')
clientSocket.sendto(frase, (serverName, serverPort))
frasemodificada, serverAddress =
                       clientSocket.recvfrom (2048)
print frasemodificada
clientSocket.close ()
```

incluye la Socket library de Python

crea el socket UDP para el servidor

obtiene la entrada del teclado del usuario

asocia el nombre del servidor y el puerto al mensaje; enviar en el socket

lectura de los carácteres de respuesta del socket en el string

imprime la cadena recibida y cierra elsocket

IPv4

Transporte UDP

Envío de datos al Servidor + puerto

Espera respuesta

Ejemplo de aplicaciones: servidor UDP

Python UDPServer

```
from socket import*
serverPort = 12000
ServerSocket = socket (AF_INET, SOCK_ DGRAM)
serverSocket.bind (('', ServerPort))
print "El servidor está listo para recibir"
While 1:
    frase, clientAddress = serverSocket.recvfrom (2048)
    frasemodificada = frase.upper ()
    serverSocket.sendto (frasemodificada, clientAddress)
```

Creación del socket UDP IPV4

crea el socket UDP

Escucha en un puerto específico

asigna el socket al número de puerto local 12000

bucle infinito

Bloquea hasta que llegue el mensaje
Obtiene datos del emisor

leer de socket UDP en el mensaje, para conseguir la dirección del cliente (IP del cliente y puerto)

Envía los datos procesados

envia la cadena de mayúsculas de nuevo al cliente

Programación de sockets *con TCP*

cliente debe contactar con el servidor

- proceso de servidor debe primero estar en ejecución
- elservidor debe haber creado el socket (puerta) que acoge el contacto del cliente

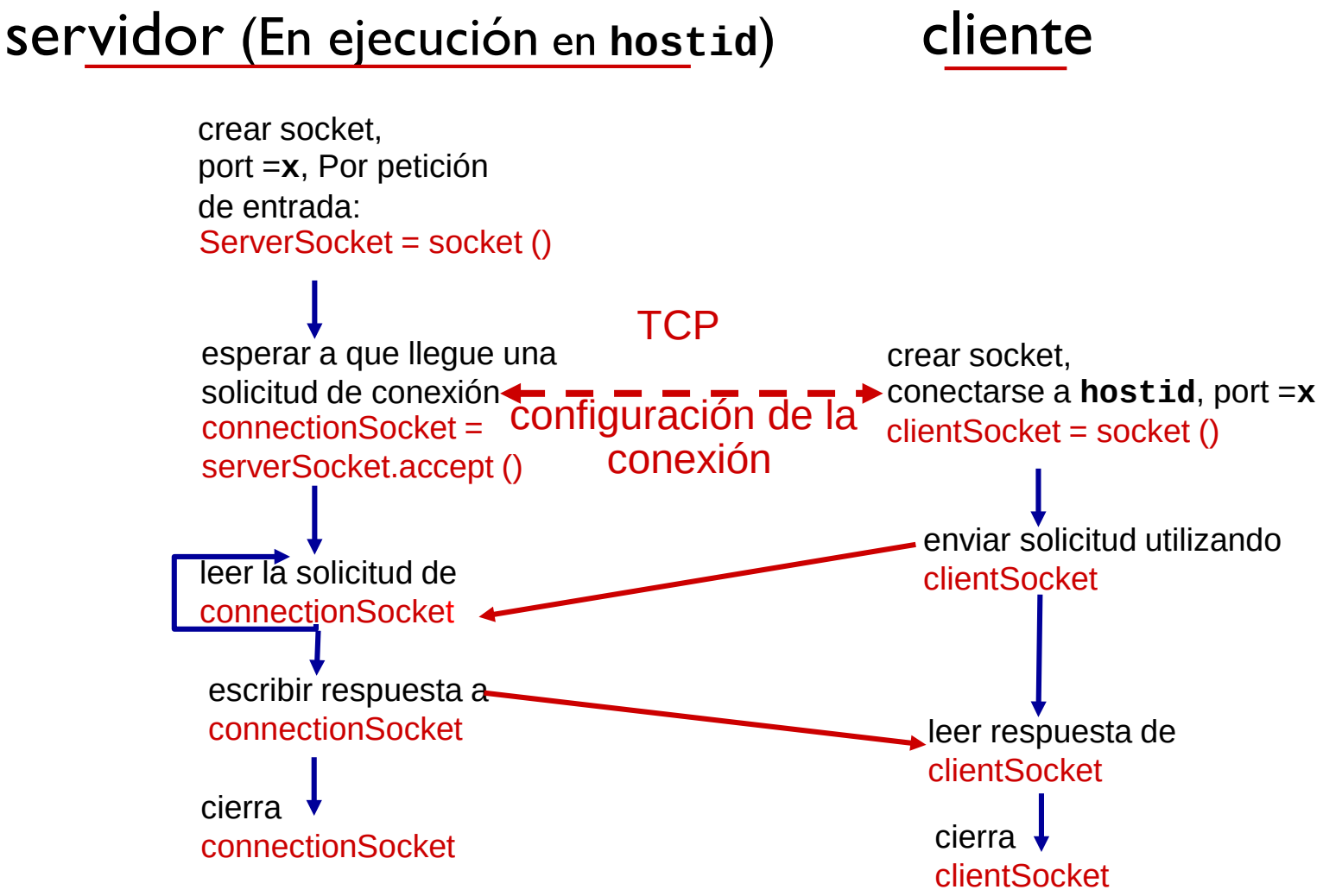
el cliente contacta al servidor a través de:

- La creación de un socket TCP, especificando la dirección IP, el número de puerto del proceso del servidor
- *cuando el cliente crea el socket:* el cliente TCP establece la conexión con el servidor TCP
- cuando fue contactado por el cliente, *el servidor TCP crea un nuevo socket* para el proceso de servidor que se comunica con ese cliente en particular
  - permite al servidor hablar con múltiples clientes
  - los números de puerto de origen se utilizan para distinguir los clientes (más en el capítulo 3)

Desde el punto de vista de la aplicación:

TCP proporciona transferencia fiable, de flujo de bytes ("pipe") entre el cliente y el servidor

Interacción socket Cliente/Servidor: TCP



Ejemplo: cliente TCP

Python TCPClient

```
from socket import*
serverName = 'nombre_servidor'
serverPort = 12000
clientSocket = socket (AF_INET, SOCK_STREAM)
clientSocket.connect ((serverName, serverPort))
frase = raw_input ('Entrada frase minúsculas:')
clientSocket.send (frase)

frasemodificada = clientSocket.recv (1024)
print 'Del  Servidor:', frasemodificada
clientSocket.close ()
```

Transporte TCP

crea el socket TCP para el servidor, puerto destino 12000

Crea la conexión

No hay necesidad de adjuntar el nombre del servidor y el puerto

Envía datos (no necesita especificar a quien)

Cierra la conexión TCP

Ejemplo de aplicaciones: servidor TCP

Python TCPServer

```
from socket import *
serverPort = 12000
serverSocket = socket (AF_INET, SOCK_STREAM)
serverSocket.bind (('', ServerPort))
serverSocket.listen (1)
print 'El servidor está listo para recibir'
while 1:
    connectionSocket, addr = serverSocket.accept ()

    frase = connectionSocket.recv (1024)
    frasemodificada = frase.upper ()
    connectionSocket.send (frasemodificada)

connectionSocket.close ()
```

Crea un socket TCP

Crea el socket TCP de acogida

Empieza a escuchar las conexiones TCP entrantes

El servidor comienza a escuchar las solicitudes TCP entrantes

Bloquea hasta que alguien hace una llamada, el socket se crea al retorno

bucle para siempre

El espera el servidor en accept () para las solicitudes entrantes, nuevo socket creado en el retorno

leer bytes del socket (pero no la dirección como en UDP)

Cierra la conexión para este cliente (pero no el socket de acogida)

Cierra la conexión del cliente TCP, no el socket de acogida