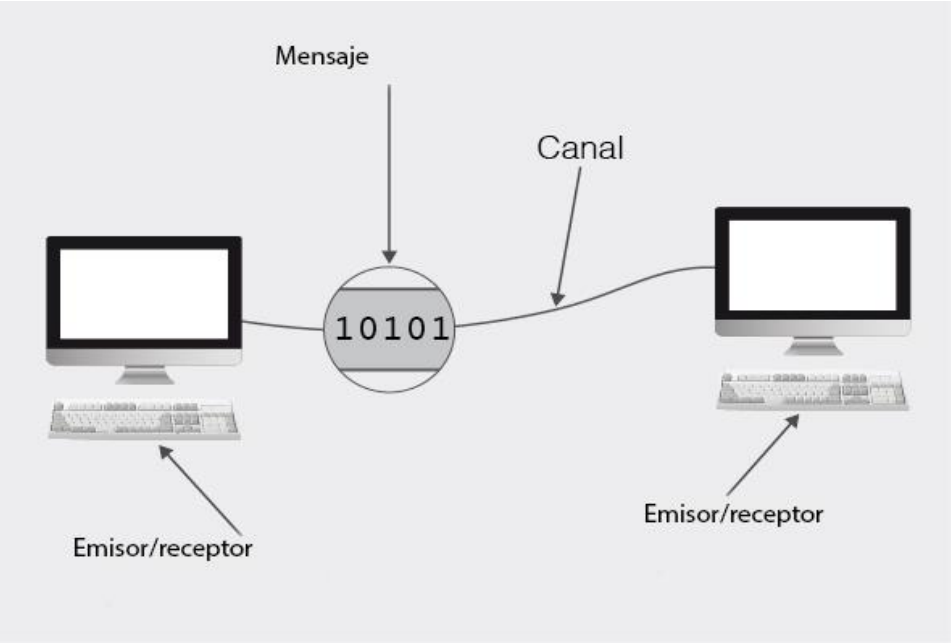
TEMA 3 APLICACIONES CLIENTE-SERVIDOR

PROGRAMACION DE SERVICIOS Y PROCESOS.

Sistema de comunicación

- Un sistema de comunicación es un sistema que **transmite** información desde un lugar (**emisor** o transmisor) a otro lugar (**receptor**). A la información que se pretende llegue al receptor se la denomina **mensaje**.



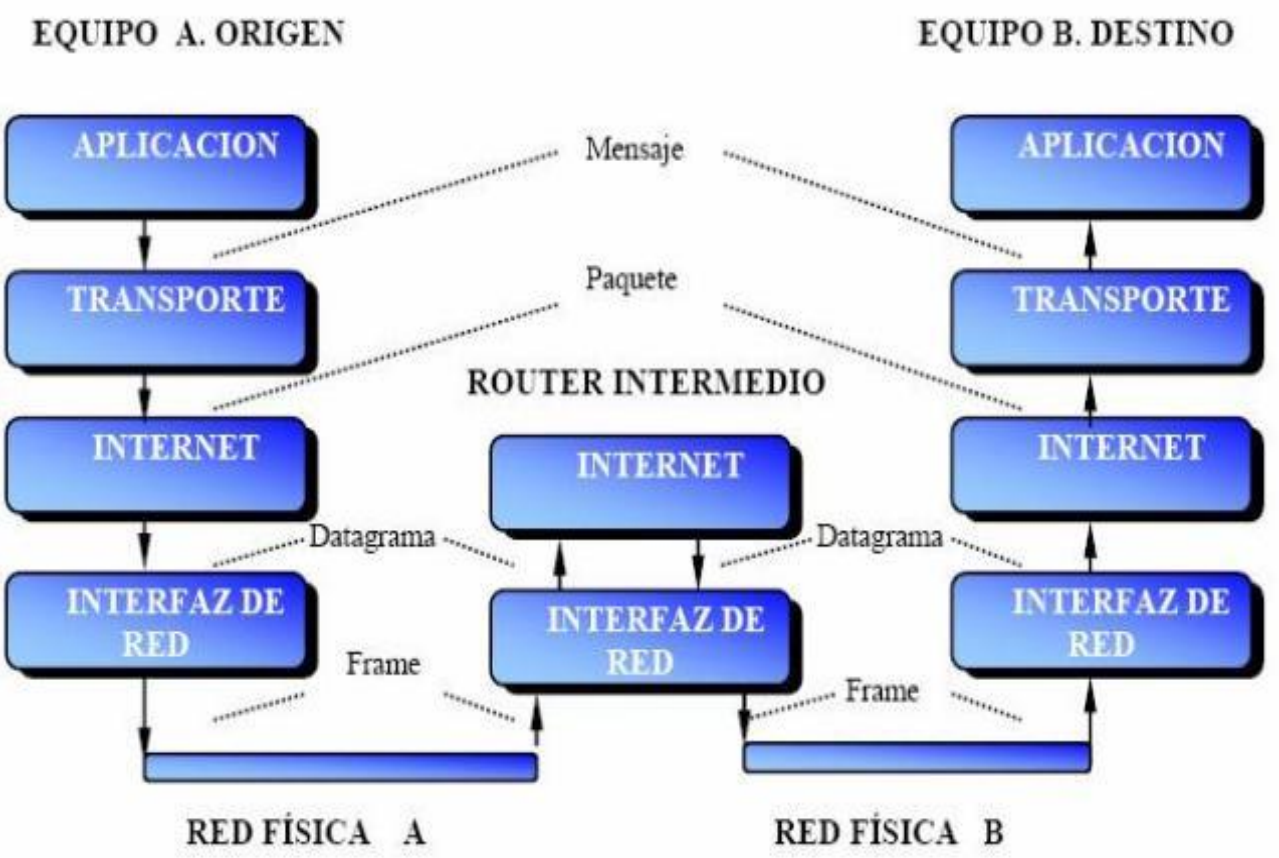
Arquitecturas de comunicaciones y protocolos

- Los ordenadores son dispositivos que funcionan muy bien cuando tienen un conjunto de reglas muy claras y son capaces de hacer ese conjunto de reglas de manera repetitiva tantas veces como sea necesario. A este conjunto de reglas lo denominamos **protocolos de comunicaciones**.
- **Para reducir la complejidad del diseño, las redes se organizan como una serie de capas o niveles**. La idea es dividir la funcionalidad que lleva consigo el proceso de para que su implementación sea más sencilla.
- **Se denomina arquitectura de una red al conjunto organizado de capas y protocolos de la misma.** Los protocolos gestionan niveles de comunicación distintos. Las reglas de alto nivel definen como se comunican las aplicaciones, mientras las de bajo nivel definen como se transmiten las señales por el cable.

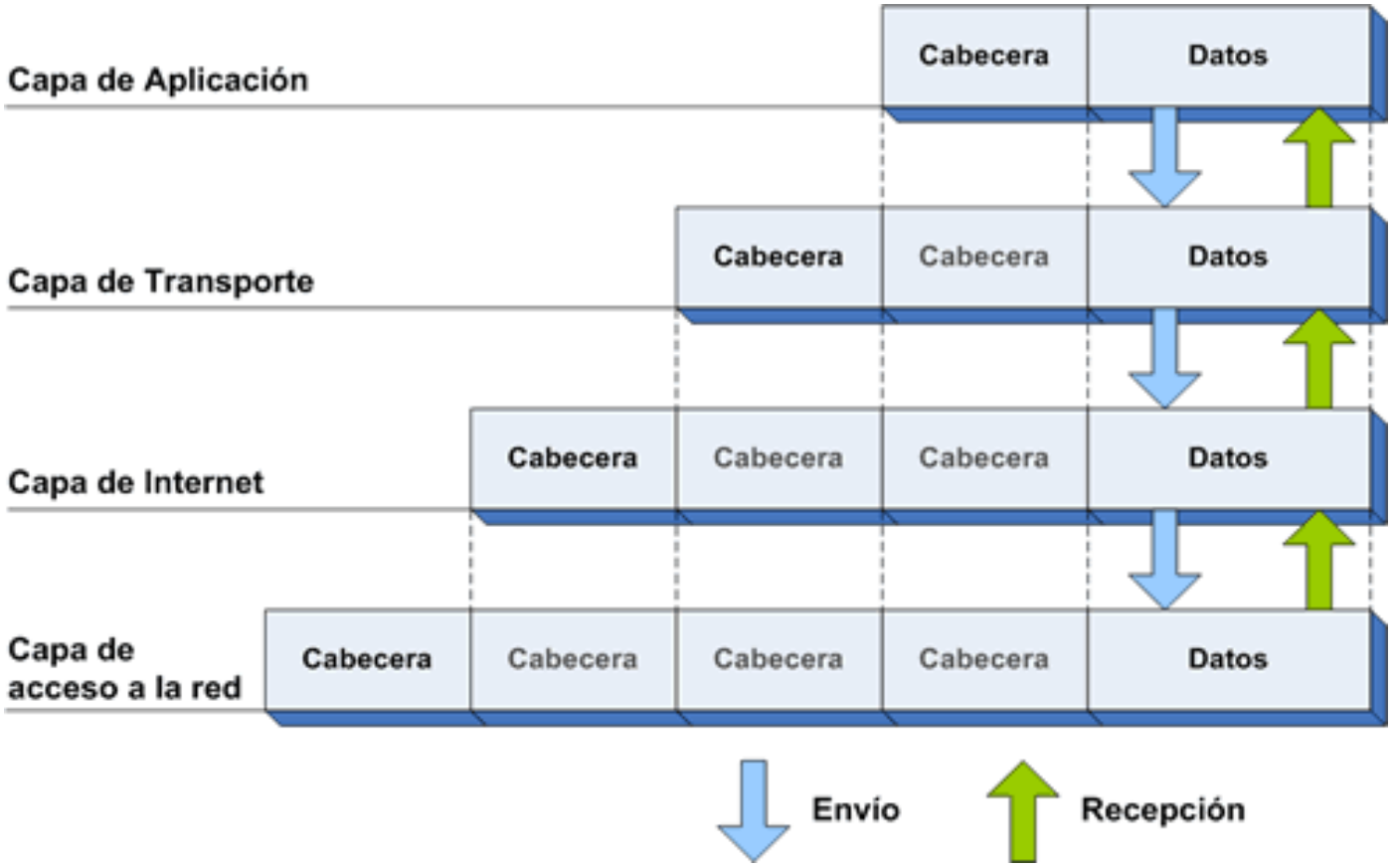
Recordando TCP/IP

- El modelo TCP/IP tiene cuatro capas:
- **La capa host-red. Esta capa permite comunicar el ordenador con el medio que conecta el equipo a la red.** En esta capa se realiza un direccionamiento físico utilizando las direcciones MAC.
- **La capa de red. Permite que los equipos envíen** paquetes en cualquier red y viajen de forma independiente a su destino. En esta capa se realiza direccionamiento por IP, ya que ésta es la capa encargada de enviar un determinado mensaje a su dirección IP de destino.

Recordando TCP/IP



Recordando TCP/IP



Recordando TCP/IP

- **La capa de transporte. Permite que los equipos lleven a cabo una conversación.** Hay dos protocolos de transporte:
  - TCP (Transmission Control Protocol) y UDP (User Datagram Protocol). El protocolo TCP es un protocolo orientado a conexión y fiable.
  - UDP es un protocolo no orientado a conexión y no fiable.
- En esta capa además se realiza el direccionamiento por puertos.
- En la capa de red los paquetes van del origen al destino.
- La capa de transporte se encarga de que la información se envíe a la aplicación adecuada (mediante un determinado puerto).

Recordando TCP/IP

- **La capa de aplicación. Esta capa engloba las funcionalidades de las capas de sesión, presentación y aplicación del modelo OSI**. Incluye todos los protocolos de alto nivel relacionados con las aplicaciones que se utilizan en Internet (por ejemplo HTTP, FTP, TELNET).

Conexiones TCP/UDP.

- Existen dos tipos de conexiones:
- **TCP (Transmission Control Protocol).**
  - **Es un protocolo orientado a la conexión que permite que un flujo** de bytes originado en una máquina se entregue sin errores en cualquier máquina destino.
  - Fragmenta el flujo entrante de bytes en mensajes y pasa cada uno a la capa de red.
  - Proceso TCP receptor reensambla los mensajes recibidos para formar el flujo de salida.
  - TCP también se encarga del control de flujo para asegurar que un emisor rápido no pueda saturar a un receptor lento con más mensajes de los que pueda gestionar.

Conexiones TCP/UDP.

- **UDP (User Datagram Protocol).**
  - **Es un protocolo sin conexión, para aplicaciones que no necesitan la** asignación de secuencia ni el control de flujo TCP y que desean utilizar los suyos propios.
  - También se utilizan para las consultas de petición y respuesta del tipo cliente-servidor, y en aplicaciones en las que la velocidad es más importante que la entrega precisa, como las transmisiones de voz o de vídeo.
- De esta forma, a la hora de programar nuestra aplicación deberemos elegir el protocolo que queremos utilizar según nuestras necesidades: TCP o UDP.

Puertos de comunicación

- Con la capa de red: Trabaja con dirección IP.
- Pero para que una aplicación pueda comunicarse con otra aplicación es necesario establecer a qué aplicación se conectará.
- El método que se emplea es el de definir direcciones de transporte en las que los procesos pueden estar a la escucha de solicitudes de conexión.
- Estos puntos terminales se llaman puertos.

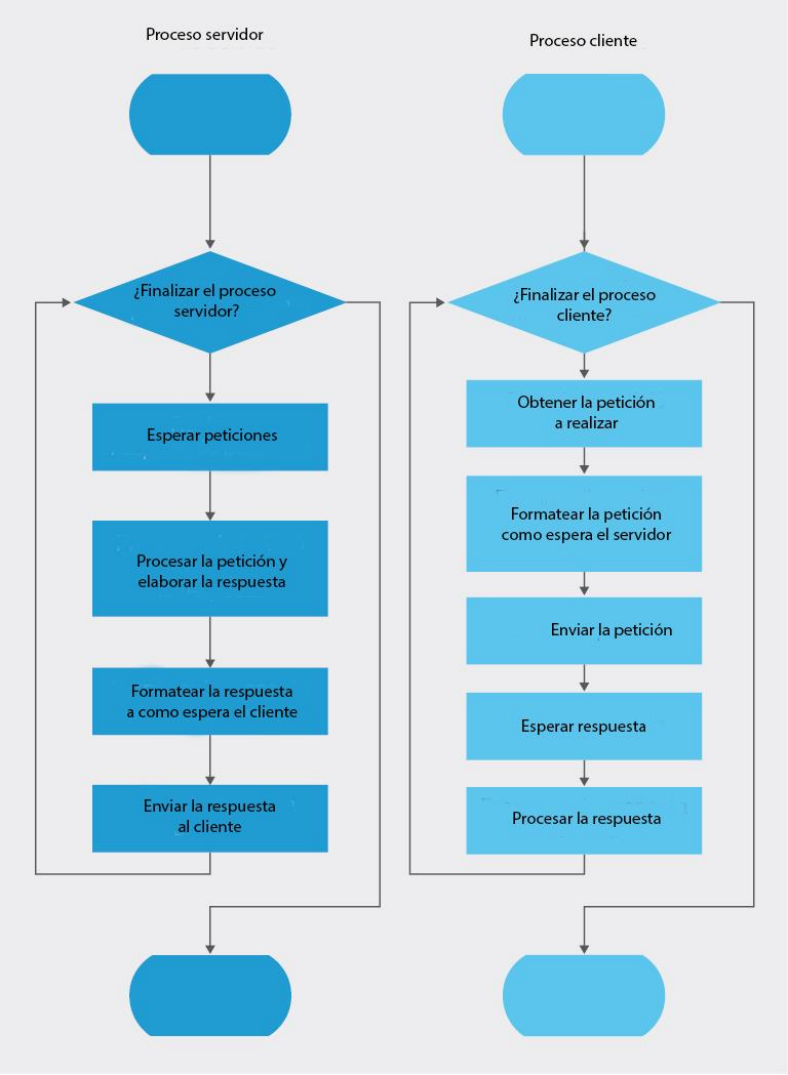
Puertos de comunicación

- Existen tres rangos de puertos establecidos:
- **Puertos conocidos [0, 1023]. Son puertos reservados** a aplicaciones de uso estándar como: 21 – FTP (File Transfer Protocol), 22 – SSH (Secure SHell), 53 – DNS (Servicio de nombres de dominio), 80 – HTTP (Hypertext Transfer Protocol), etc.
- **Puertos registrados [1024, 49151].** Estos puertos son asignados por IANA para un servicio específico o aplicaciones. Estos puertos pueden ser utilizados por los usuarios libremente.
- **Puertos dinámicos [49152, 65535].** Este rango de puertos no puede ser registrado y su uso se establece para conexiones temporales entre aplicaciones.

Modelos de comunicaciones.

Modelo Sistema de Información Distribuido

PARADIGMA CLIENTE-SERVIDOR



Sockets

- Los sockets permiten la comunicación entre procesos de diferentes equipos de una red. Un socket, es un punto de información por el cual un proceso puede recibir o enviar información.
- Para crear un socket hay que tener claro el tipo de socket que se quiere crear (TCP o UDP).

Sockets TCP

- Al utilizar sockets TCP, el servidor utiliza un puerto por el que recibe las diferentes peticiones de los clientes.
- Normalmente, el puerto del servidor es un puerto bajo [1-1023].

Socket TCP - Conexión

Sockets TCP

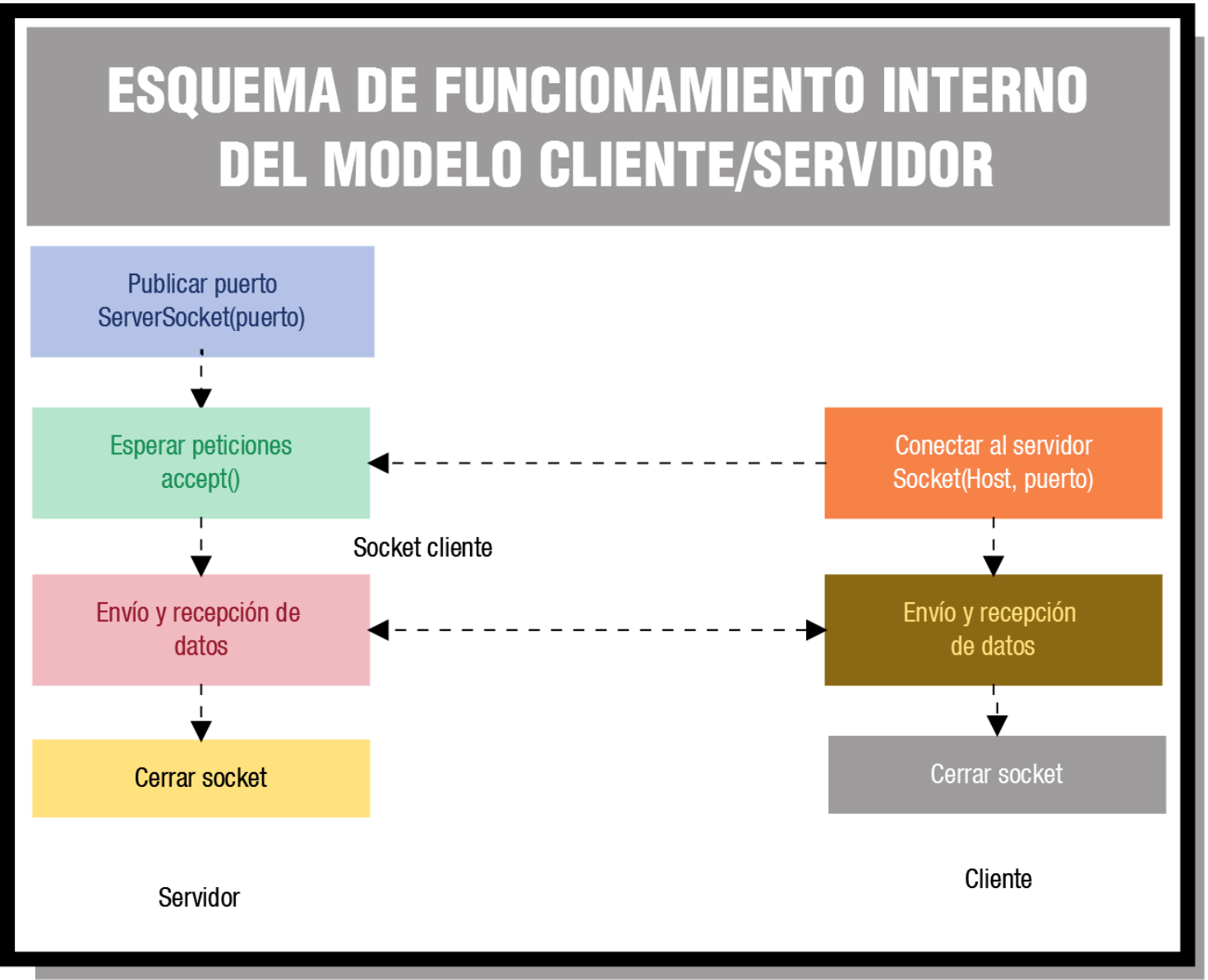
- Cuando el cliente se conecta al servidor se crea un nuevo socket que será en el encargado de permitir el envío y recepción de datos.
- Se crea de forma dinámica y se encuentra en el rango 49152 y 655335.
- Así, el puerto por donde se reciben las conexiones de los clientes queda libre.

TCP: transferencia de datos

Sockets TCP

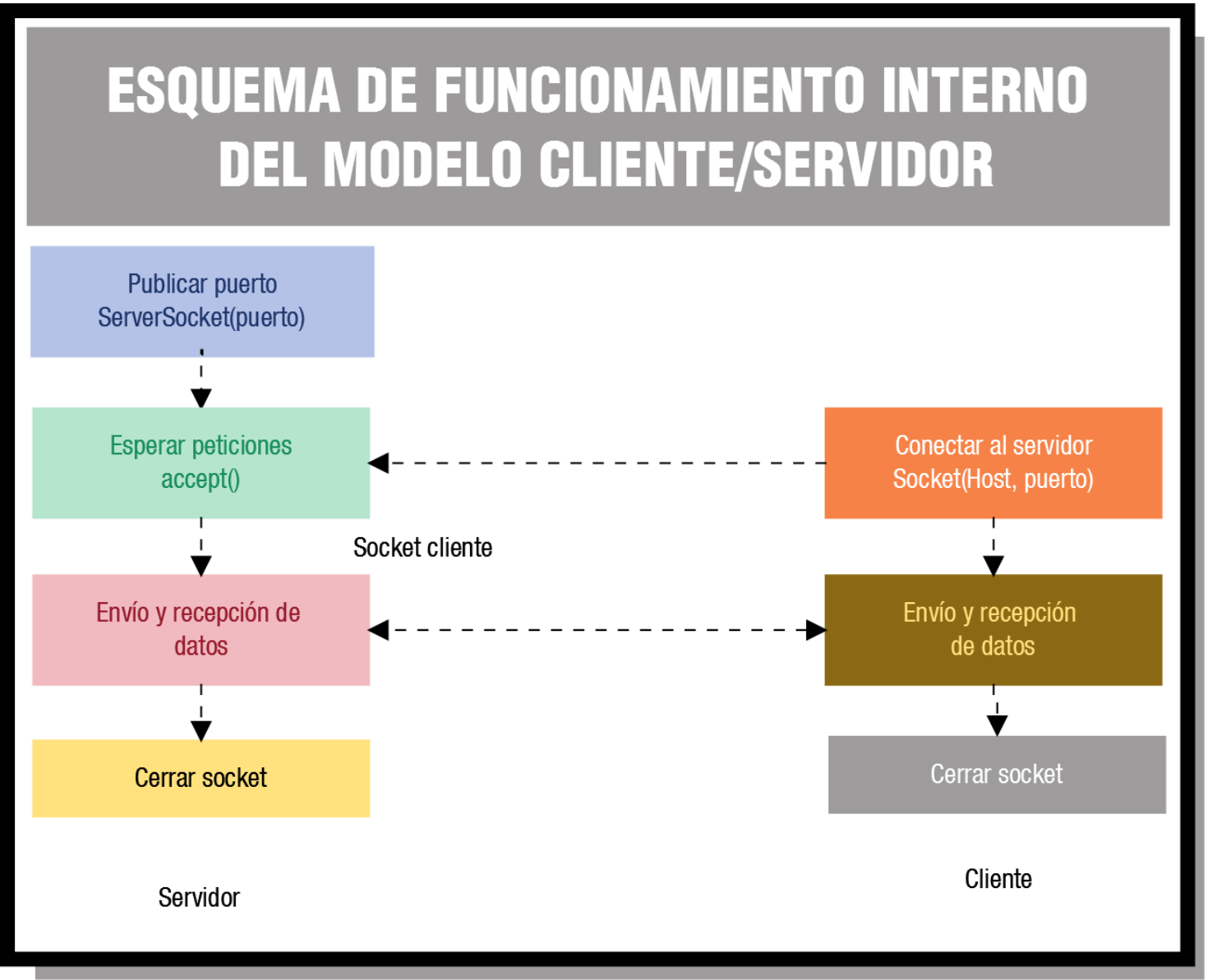
- El paquete java.net de Java proporciona la clase Socket que permite la comunicación por red.
- También incluye la **clase ServerSocket**, que permite a un servidor escuchar y recibir las peticiones de los clientes por la red.
- Y la **clase Socket** permite a un cliente conectarse a un servidor para enviar y recibir información.

Sockets TCP en el servidor.



Vemos el ejemplo 1.Servidor

Sockets TCP en el cliente.



Vemos el ejemplo 2.Cliente

Flujo de Entrada y de Salida

- Una vez establecida la conexión entre el cliente y el servidor se inicializa la variable del tipo Socket que en el ejemplo se llama sCliente.
- Para poder enviar o recibir datos a través del socket es necesario establecer un stream (flujo) de entrada o de salida según corresponda.
- Para ello utilizamos las clases DataInputStream y DataOutputStream. (Ver API)

Ejemplo

- Para continuar con el ejemplo anterior y poder utilizar los sockets para enviar información vamos a realizar un ejemplo muy sencillo en el que el servidor va a aceptar tres clientes (de forma secuencial no concurrente) y le va a indicar el número de cliente que es.
- 3.ServidorModificado
- Se ejecuta tres veces el proyecto 3.1.ClienteModificado

Sockets UDP

- En el caso de utilizar sockets UDP no se crea una conexión y básicamente permite enviar y recibir mensajes a través de una dirección IP y un puerto.
- Estos mensajes se gestionan de forma individual y no se garantiza la recepción o envío del mensaje como si ocurre en TCP.

Sockets UDP

- Para utilizar sockets UDP en java tenemos la clase DatagramSocket y para recibir o enviar los mensajes se utiliza clase DatagramPacket.
- Cuando se recibe o envía un paquete se hace con la siguiente información: mensaje, longitud del mensaje, equipo y puerto.

Ejemplo

- A continuación, para aprender a programar Sockets UDP se va a realizar un ejemplo sencillo donde intervienen dos procesos:
- **ReceptorUDP.** Inicia el puerto 1500 y muestra en pantalla todos los mensajes que llegan a él.
- **EmisorUDP.** Permite enviar por líneas de comandos mensajes al receptor por el puerto 1500.

Como manejamos el ejemplo.

- Para realizar la prueba:
- Lanzamos el ReceptorUDP. Este se queda esperando. 4.ReceptorUDP
- Lanzamos EmisorUDP tantas veces como queramos. (Lleva dos argumentos de entrada). 5.EmisorUDP
- El receptor va recibiendo los mensajes que se le envían del emisor.

EJEMPLO

- Vemos un programa ejemplo de emisor y receptor con sockets UDP. 6.EjemploUDP
- Ejercicio: Realiza el ejercicio anterior utilizando sockets TCP. El cliente se conecta al servidor y le dice cual es el número entero del que quiere calcular su cuadrado. El servidor calcula el cuadrado, lo imprime por pantalla y se lo envía al cliente, el cual también lo imprime 7.EjemploTCPCuadradoNumero

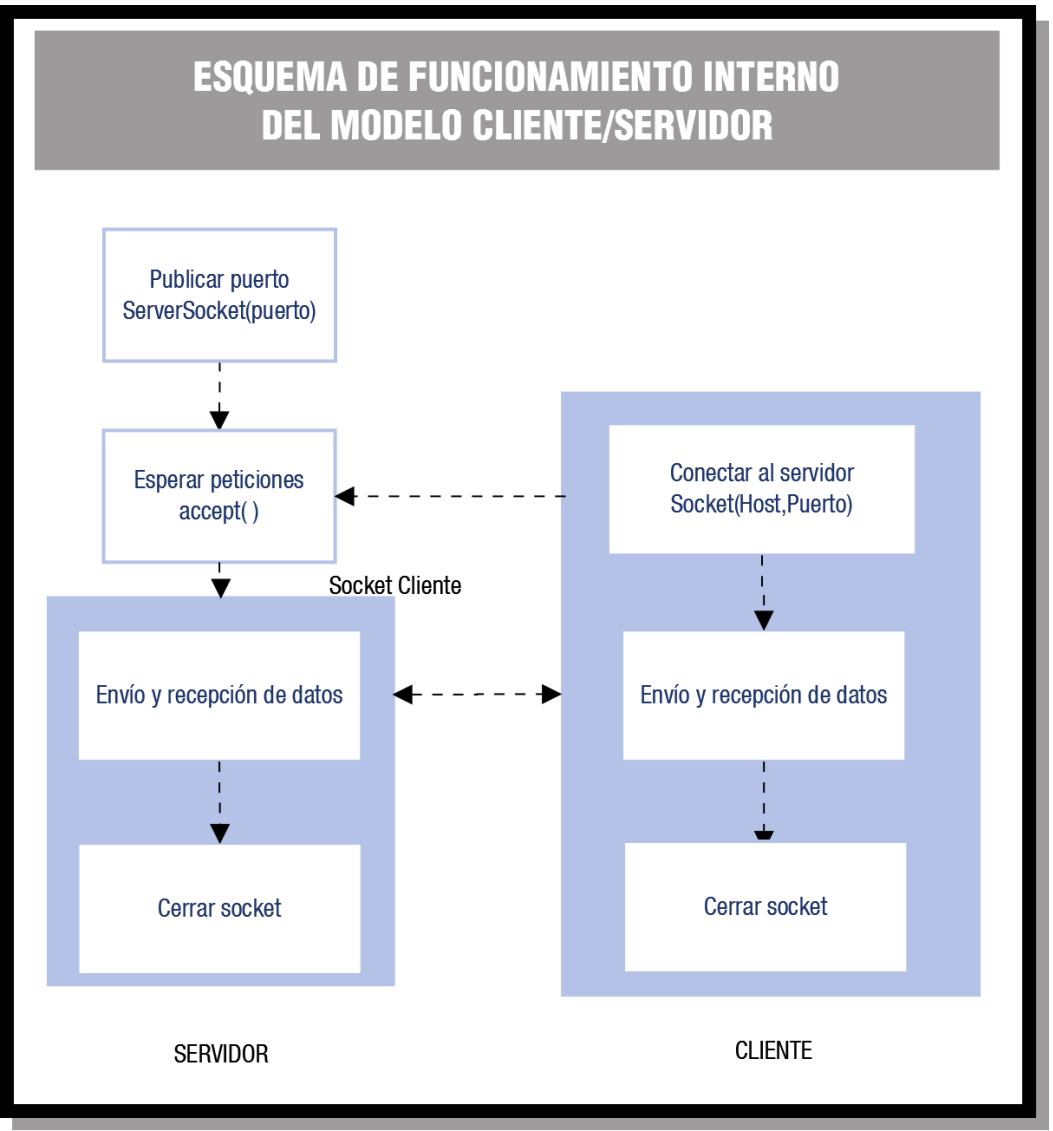
Optimización de sockets

- Con sockets es importante optimizar su funcionamiento.
- También hay que garantizar la seguridad del sistema.
- La aplicación debe contar con ciertas características.

Optimización de sockets

- **Atender múltiples peticiones simultáneamente.**
- **Seguridad. Para asegurar el sistema, como mínimo, el servidor debe ser** capaz de evitar la pérdida de información, filtrar las peticiones de los clientes para asegurar que éstas están bien formadas y llevar un control sobre las diferentes transacciones de los clientes.
- Por último, es necesario dotar a nuestro sistema de mecanismos para **monitorizar los tiempos de respuesta de los clientes para ver el comportamiento del sistema.**

Atender múltiples peticiones simultáneas.



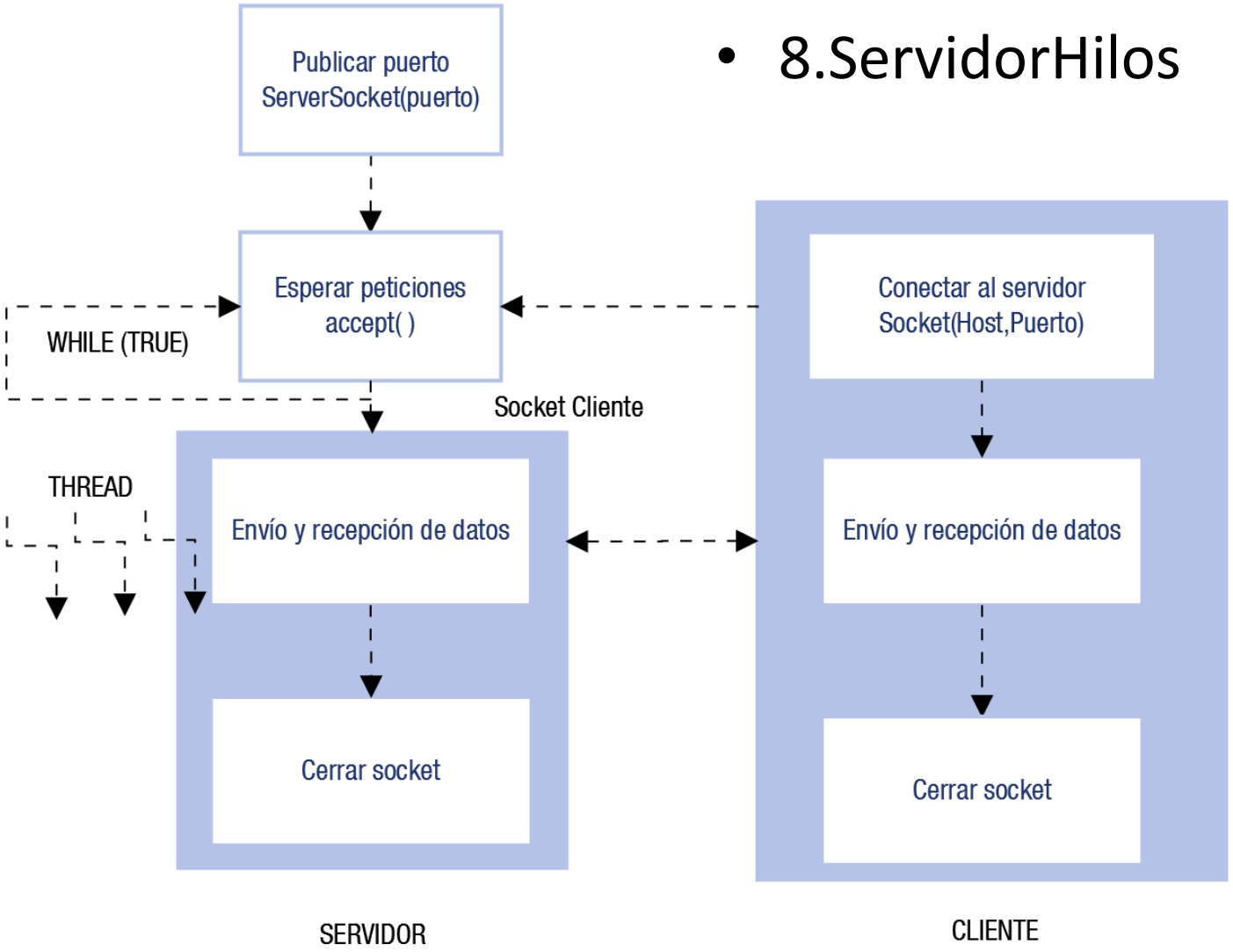
Objetivo: múltiples clientes usen el servidor de forma simultánea.

La parte que atiende al cliente (zona coloreada de azul) se atienda de forma independiente para cada uno de los clientes.

Atender múltiples peticiones simultáneas.

- Modificaciones en el código del servidor:
  - Se usa un bucle while para cada vez que se realice una conexión con un cliente, crear un hilo de ejecución, que será quien atienda al cliente.
  - Tendremos tantos hilos como clientes conectados al servidor.
- Vemos un esquema representativo:

ESQUEMA DE FUNCIONAMIENTO INTERNO DEL MODELO CLIENTE/SERVIDOR CONCURRENTE



- 8.ServidorHilos

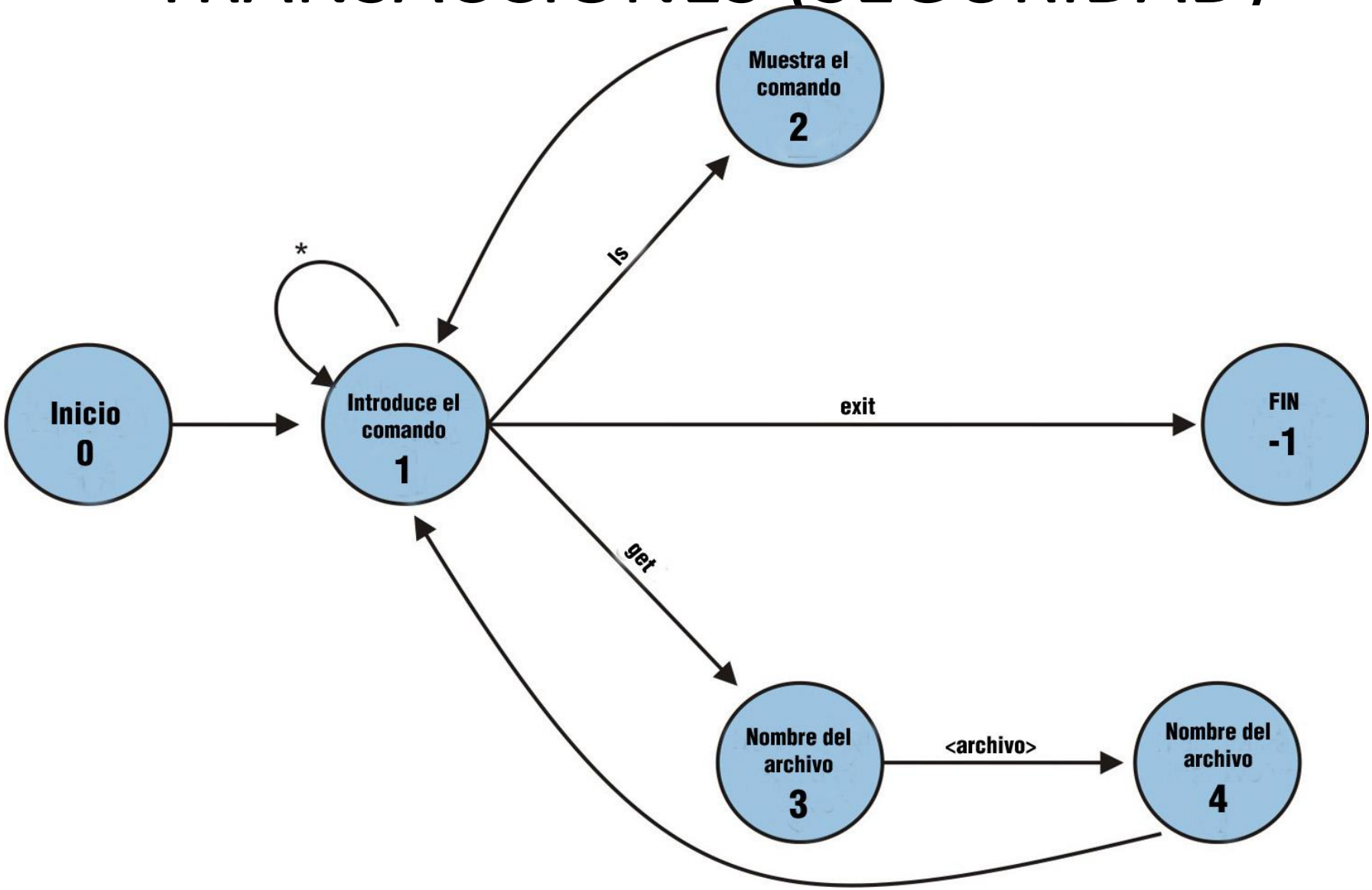
TRANSACCIONES (SEGURIDAD)

- Uno de los principales fallos de seguridad que se producen en la programas clientes/servidor es que el cliente pueda realizar:
  - Operaciones no autorizadas: El servidor no puede procesar una orden a la que el cliente no tiene acceso.
  - Mensajes mal formados: El cliente envía al servidor mensajes mal formados que producen un error de procesamiento del sistema llegando incluso a dejar "colgado" el servidor.

TRANSACCIONES (SEGURIDAD)

- Para evitar cualquier problema de seguridad es muy importante modelar el flujo de información y el comportamiento del servidor con un diagrama de estados o autómata.
- Lo vemos con un ejemplo. En la siguiente diapositiva vemos un esquema del comportamiento del servidor.

TRANSACCIONES (SEGURIDAD)



TRANSACCIONES (SEGURIDAD)

- El servidor se inicia en el estado 0. Envía al cliente el mensaje *Introduce el comando. El cliente puede enviar los comandos:*
  - *ls que va al estado 2 mostrando el contenido del directorio y vuelve automáticamente al estado 1.*
  - *get que le lleva al estado 3 donde le solicita al cliente el nombre del archivo a mostrar. Al introducir el* nombre del archivo se desplaza al estado 4 donde muestra el contenido del archivo y vuelve automáticamente al estado 1.
  - *exit que le lleva directamente al estado donde finaliza la conexión del cliente (estado -1).*
  - Cualquier otro comando hace que vuelva al estado 1 solicitándole al cliente que introduzca un comando válido.

TRANSACCIONES (SEGURIDAD)

- Para poder seguir el comportamiento del autómata el servidor tiene que definir dos variables *estado y comando.*
- La variable *estado almacena la posición en la que se encuentra y la variable comando es el comando que recibe* el servidor y el que permite la transición de un estado a otro.
- Se modela el comportamiento del autómata utilizando estructuras case e if.

MONITORIZAR TIEMPOS DE RESPUESTA

- Un aspecto muy importante son los tiempos de respuesta del servidor. Desde que el cliente realiza una petición hasta que recibe su resultado intervienen dos tiempos:
  - **Tiempo de procesamiento.** Es el tiempo que el servidor necesita para procesar la petición del cliente y enviar los datos.
  - **Tiempo de transmisión.** Es el tiempo que transcurre para que los mensajes viajen a través de los diferentes dispositivos de la red hasta llegar a su destino.

MONITORIZAR TIEMPOS DE RESPUESTA

- A continuación vamos a ver un ejemplo en el que se calcula el tiempo de trasmisión de datos entre una aplicación Cliente y Servidor. Para ello, el servidor le va a enviar al cliente un mensaje con el tiempo del sistema en milisegundos y el cliente cuando reciba el mensaje calculará la diferencia entre el tiempo de su sistema y el del mensaje.