# 2.2.1 Primitivas básicas del interfaz Sockets

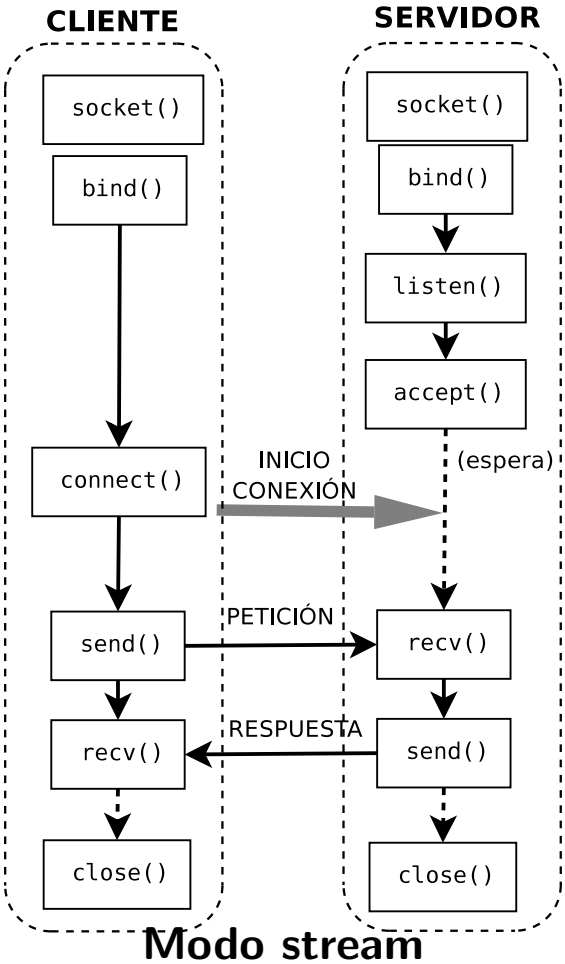
## (a) Modos de operación

- El interfaz de Sockets soporta dos modos de operación
  - **modo stream (modo TCP)**: establece una conexión entre los dos extremos que permite enviar flujos de bytes en ambas direcciones
    - Existen dos papeles definidos implicitamente:
      **sockets de servidor:** esperan recibir conexiones
      **sockets de tráfico:** { inician conexiones (en los clientes); envían/reciben datos (en clientes y servidor) }
    - Funciones: { `socket()`, `bind()`; `listen()`, `accept()`; `connect()`; `send()`, `recv()` }
  - **modo datagrama (modo UDP)**: no se establece conexión, cada mensaje ($datagrama$) es independiente
    - Funciones: { `socket()`, `bind()`; `sendto()`, `revfrom()` }
- En ambos casos se ofrece al programador un interfaz análogo al interfaz de acceso a ficheros del S.O.
  - lectura $\longrightarrow$ recepción
  - escritura $\longrightarrow$ envío
- **Importante:** la información intercambiada entre los extremos depende de los procesos
  - La interfaz socket no impone la sintaxis o semántica de los mensajes intercambiados
  - Los procesos deben acordar y gestionar el formato de los mensajes
  - Los procesos deben manejar explícitamente los tipos de datos y realizar los cambios de codificación necesarios
    - Cliente y servidor deben usar el mismo esquema para empaquetar y aplanar los datos ($marshaling$)

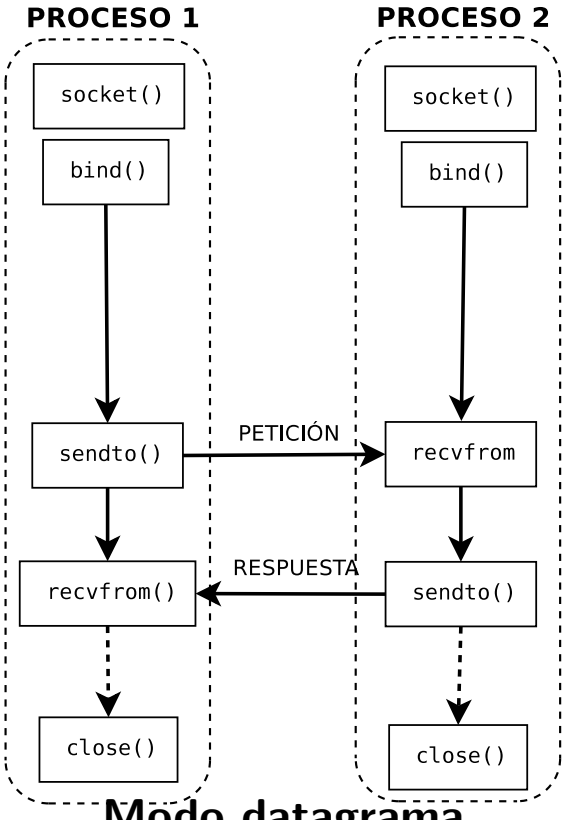
## (b) Primitivas básicas

- **socket()**: crea un socket (de tipo *stream* o *datagrama*)
- **bind()**: asocia la dirección local (IP+nº puerto) al socket
  - Sólo es obligatorio asignar dir. local (nº puerto) en servidores en modo stream
  - En clientes si no se asigna se hará automáticamente en su primer uso [connect() o sendto()]
- **listen()**: pone un socket de servidor en modo espera [no bloqueante]
- **connect()**: establece la conexión con una máquina remota
  - exige indicar dirección del proceso servidor destino (IP+nº puerto)
- **accept()**: indica que se ha recibido y aceptado una conexión de un cliente (tomada de la cola de espera) y genera un nuevo socket para que el servidor intercambie información con el cliente [bloqueante]
- **send(), sendto()**: envío de datos (stream, datagrama) [no bloqueante]
- **recv(), revfrom()**: recepción de datos (stream, datagrama) [bloqueante]

## (c) Esquema de funcionamiento




Modo stream




Modo datagrama

## Ventajas uso de Sockets

- Rendimiento y mínima sobrecarga (control total sobre los datos enviados)
- Máxima flexibilidad a la hora de desarrollar aplicaciones (control total sobre los datos enviados)
- Disponibilidad: interfaz socket disponible en múltiples S.O. y lenguajes de programación

## Inconvenientes uso de Sockets

- Necesidad de realizar un manejo explícito de los datos (formateo, codificación, etc)
  - Requiere acordar/negociar los formatos de transmisión
  - Requiere aplanar/desaplanar los datos estructurados antes de ser enviados/recibidos (*marshaling* o serialización)
- Escasa transparencia de localización
  - Necesario manejar explicitamente dir. IP, puertos, protocolo, etc
- Complejidad y dificultad de programación