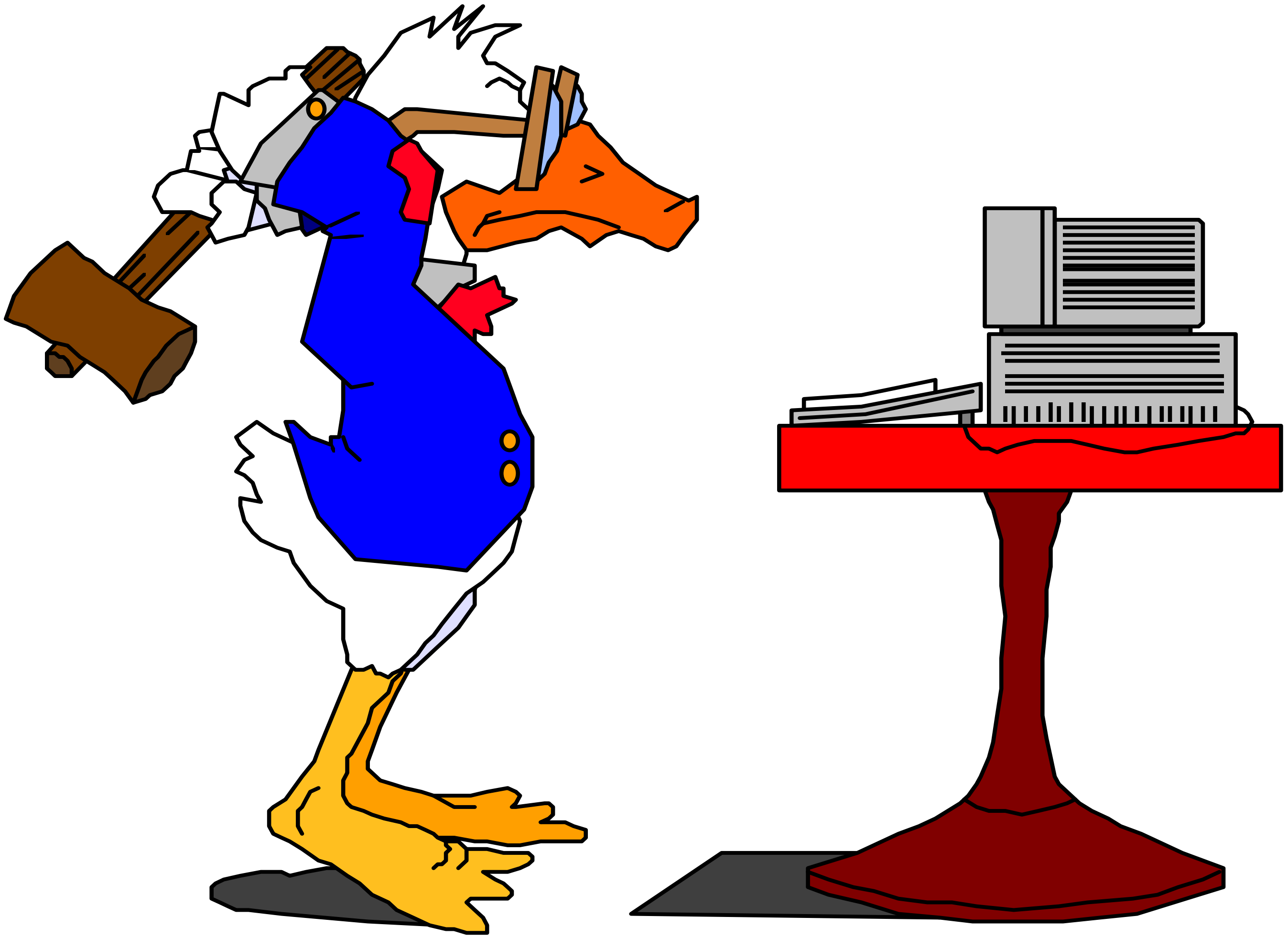
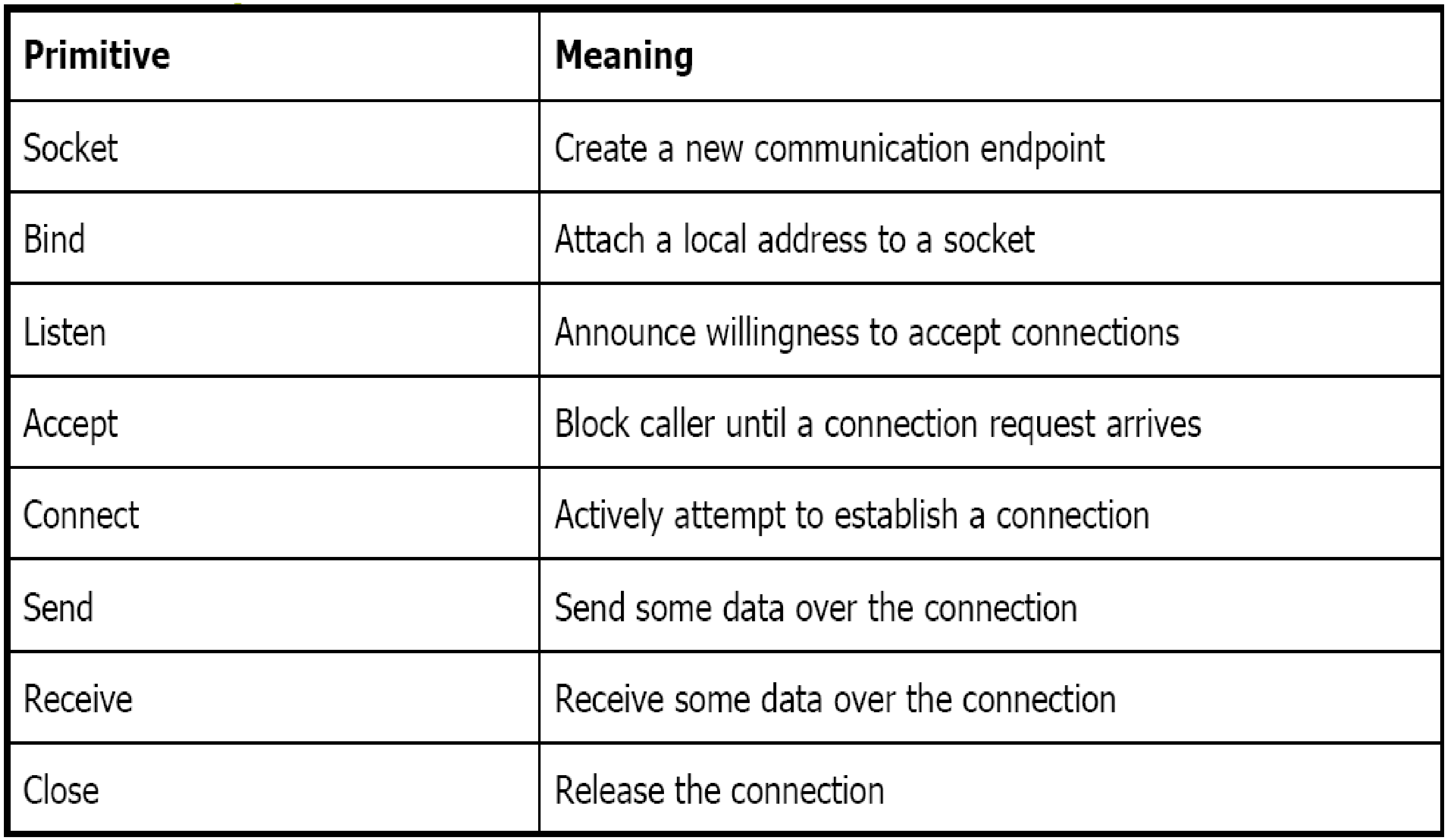
Socket Programming



Client-Server communication

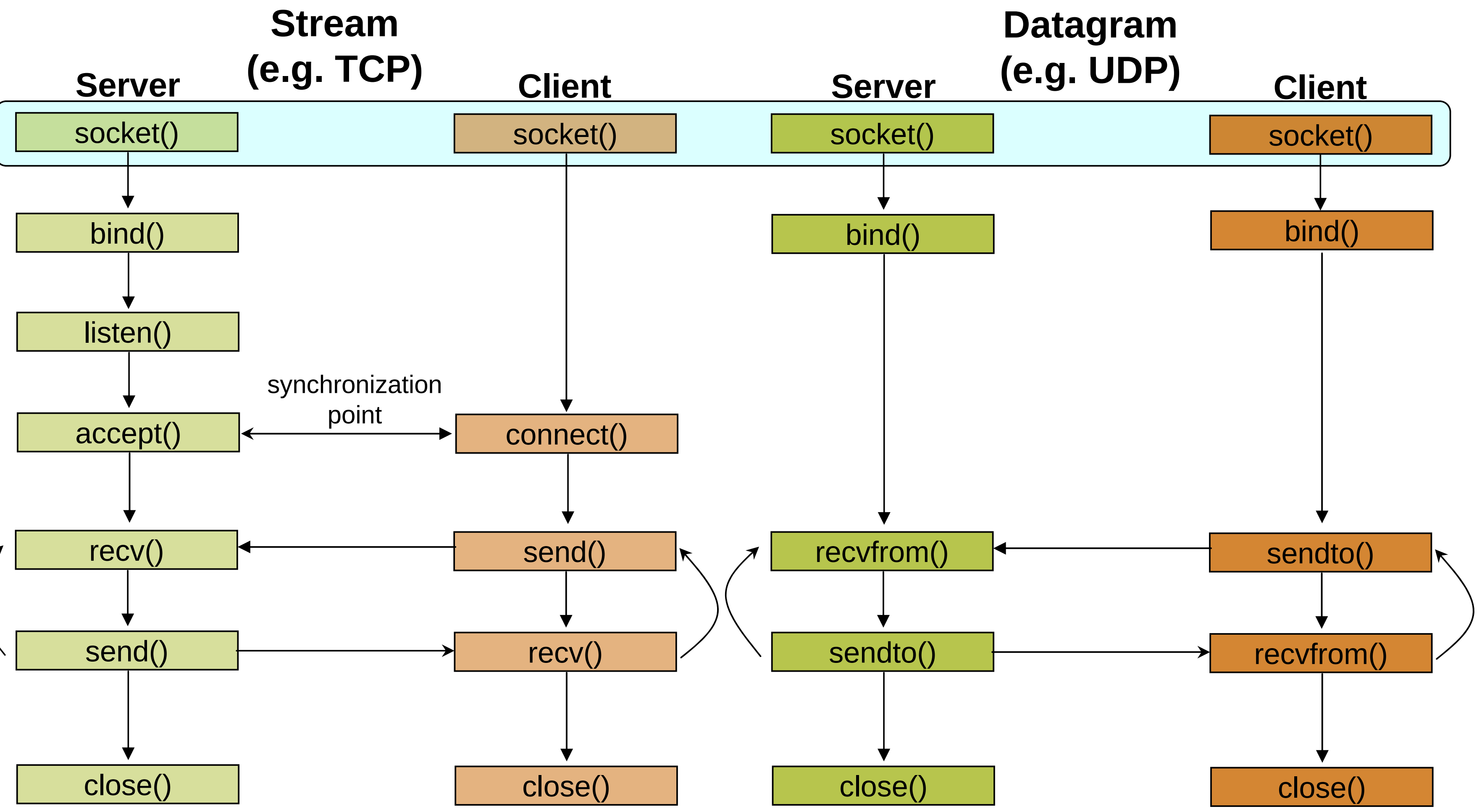
- Server
  - passively waits for and responds to clients
  - passive socket
- Client
  - initiates the communication
  - must know the address and the port of the server
  - active socket

Sockets - Procedures



Primitive	Meaning
Socket	Create a new communication endpoint
Bind	Attach a local address to a socket
Listen	Announce willingness to accept connections
Accept	Block caller until a connection request arrives
Connect	Actively attempt to establish a connection
Send	Send some data over the connection
Receive	Receive some data over the connection
Close	Release the connection

Client - Server Communication - Unix

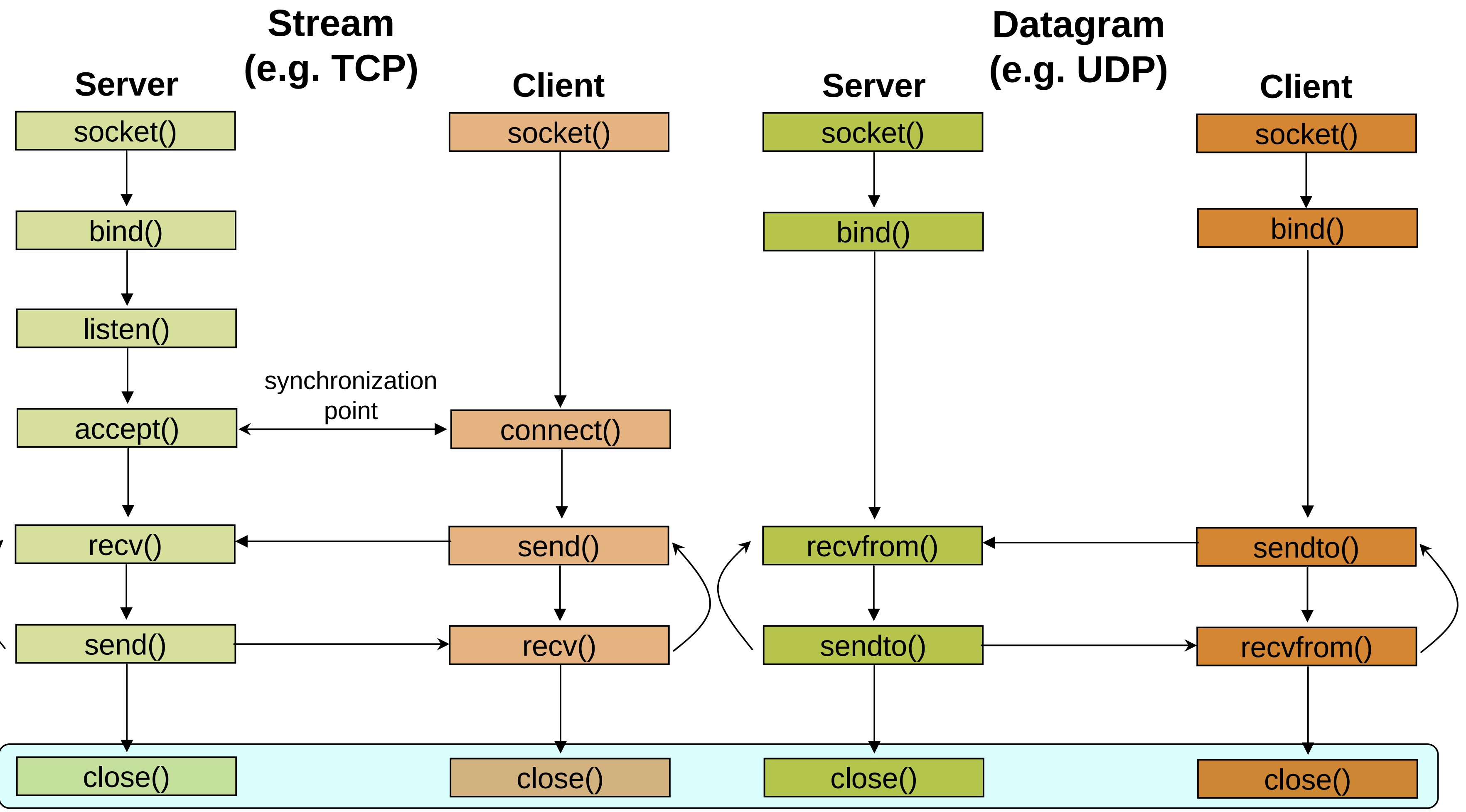


Socket creation in C: `socket()`

- `int sockid = socket(family, type, protocol);`
  - sockid: socket descriptor, an integer (like a file-handle)
  - family: integer, communication domain, e.g.,
    - PF_INET, IPv4 protocols, Internet addresses (typically used)
    - PF_UNIX, Local communication, File addresses
  - type: communication type
    - SOCK_STREAM - reliable, 2-way, connection-based service
    - SOCK_DGRAM - unreliable, connectionless, messages of maximum length
  - protocol: specifies protocol
    - IPPROTO_TCP IPPROTO_UDP
    - usually set to 0 (i.e., use default protocol)
  - upon failure returns -1

☞ NOTE: socket call does not specify where data will be coming from, nor where it will be going to – it just creates the interface!

Client - Server Communication - Unix



Socket close in C: `close()`

- When finished using a socket, the socket should be closed
- `status = close(sockid);`
  - sockid: the file descriptor (socket being closed)
  - status: 0 if successful, -1 if error
- Closing a socket
  - closes a connection (for stream socket)
  - frees up the port used by the socket

Specifying Addresses

- Socket API defines a generic data type for addresses:

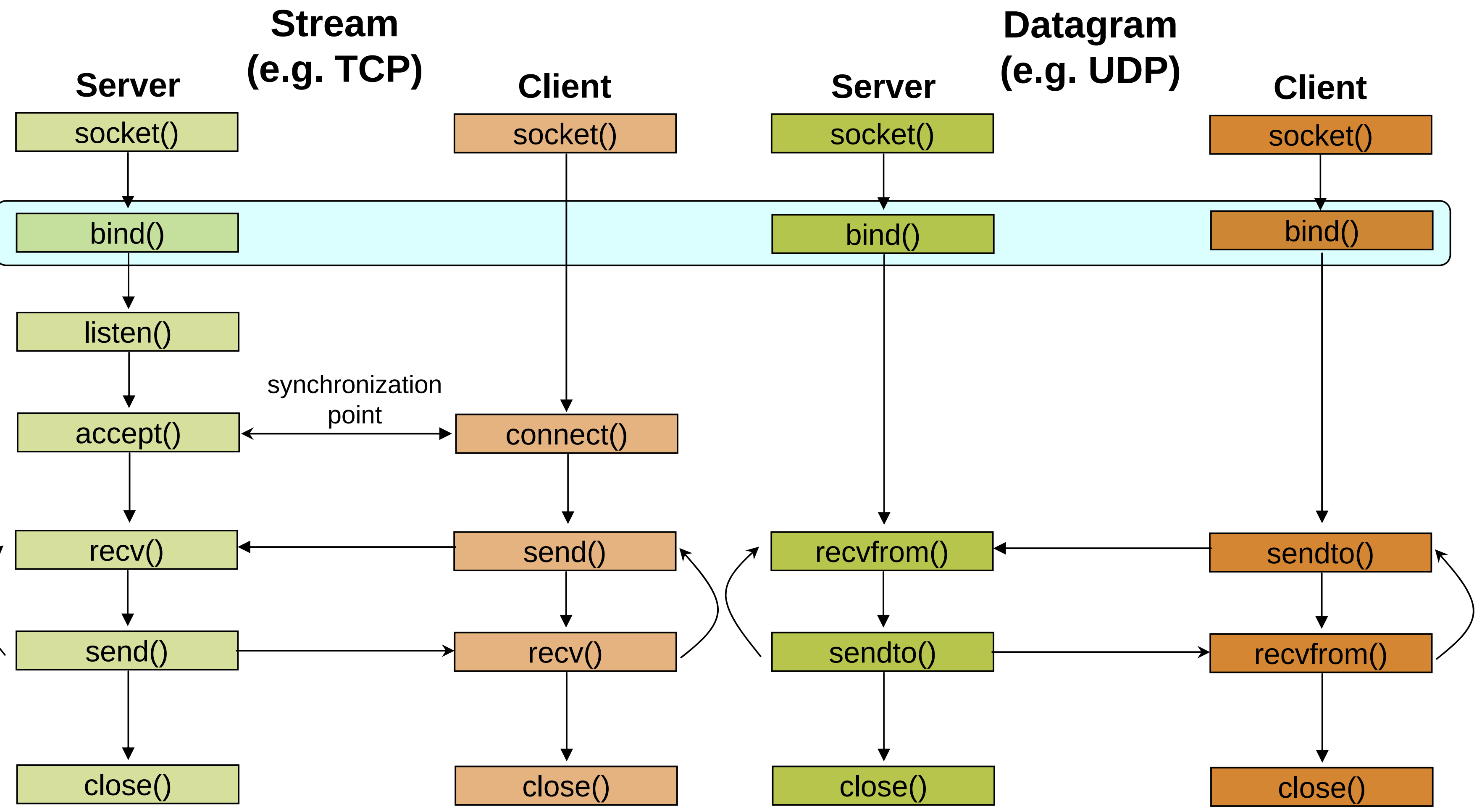
```
struct sockaddr {
    unsigned short sa_family;  /* Address family (e.g. AF_INET) */
    char sa_data[14];          /* Family-specific address information */
}
```

- Particular form of the sockaddr used for TCP/IP addresses:

```
struct in_addr {
    unsigned long s_addr;          /* Internet address (32 bits) */
}
struct sockaddr_in {
    unsigned short sin_family;     /* Internet protocol (AF_INET) */
    unsigned short sin_port;       /* Address port (16 bits) */
    struct in_addr sin_addr;     /* Internet address (32 bits) */
    char sin_zero[8];              /* Not used */
}
```

☞ **Important**: sockaddr_in can be casted to a sockaddr

Client - Server Communication - Unix



Assign address to socket: `bind()`

- associates and reserves a port for use by the socket

- `int status = bind(sockid, &addrport, size);`
  - sockid: integer, socket descriptor
  - addrport: struct sockaddr, the (IP) address and port of the machine
    - for TCP/IP server, internet address is usually set to INADDR_ANY, i.e., chooses any incoming interface
  - size: the size (in bytes) of the addrport structure
  - **status**: upon failure -1 is returned

bind()- Example with TCP

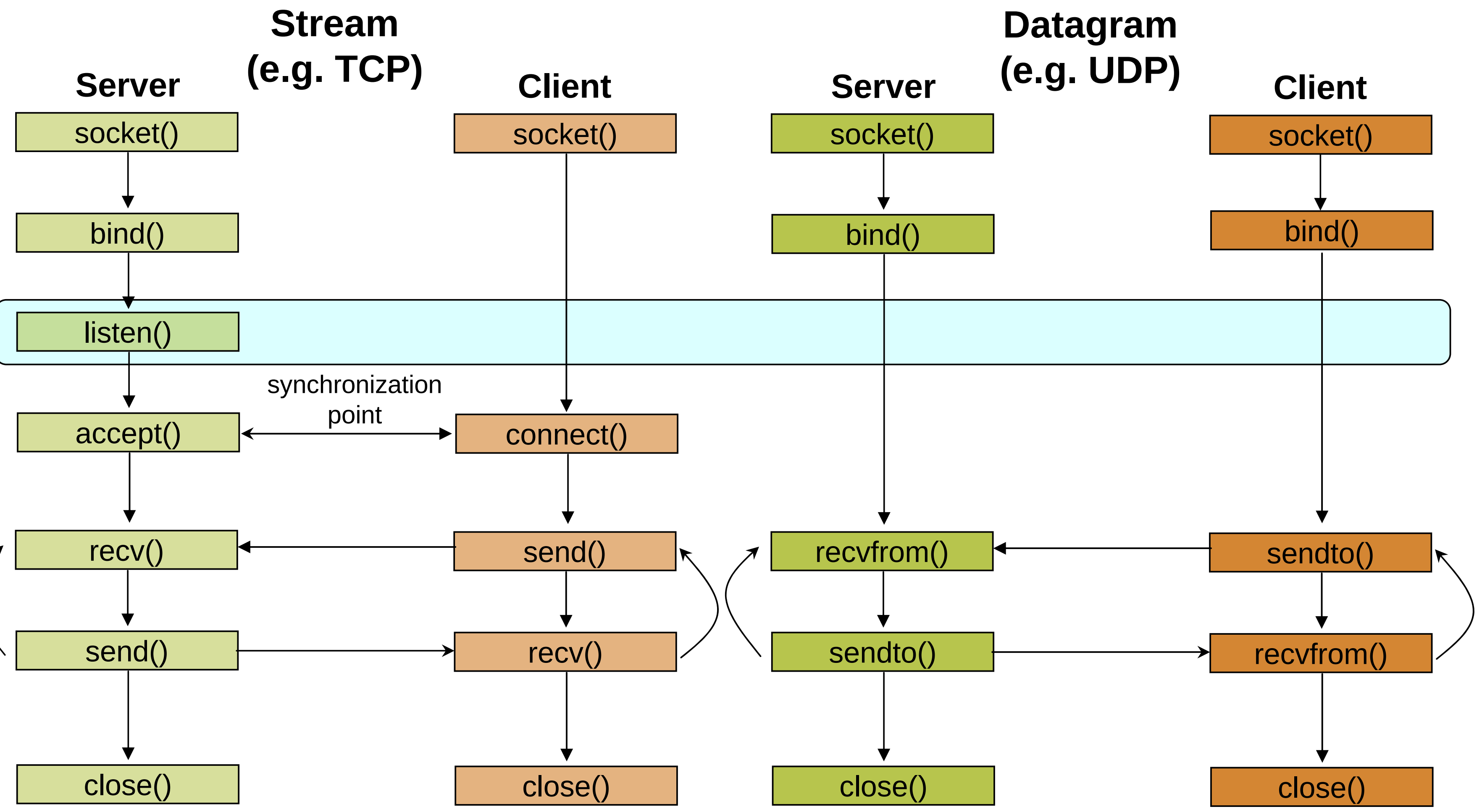
```
int sockid;
struct sockaddr_in addrport;
sockid = socket(PF_INET, SOCK_STREAM, 0);

addrport.sin_family = AF_INET;
addrport.sin_port = htons(5100);
addrport.sin_addr.s_addr = htonl(INADDR_ANY);
if(bind(sockid, (struct sockaddr *) &addrport, sizeof(addrport))!= -1) {
   …}
```

Skipping the `bind()`

- bind can be skipped for both types of sockets

- Datagram socket:
  - if only sending, no need to bind. The OS finds a port each time the socket sends a packet
  - if receiving, need to bind
- Stream socket:
  - destination determined during connection setup
  - don’t need to know port sending from (during connection setup, receiving end is informed of port)

Client - Server Communication - Unix



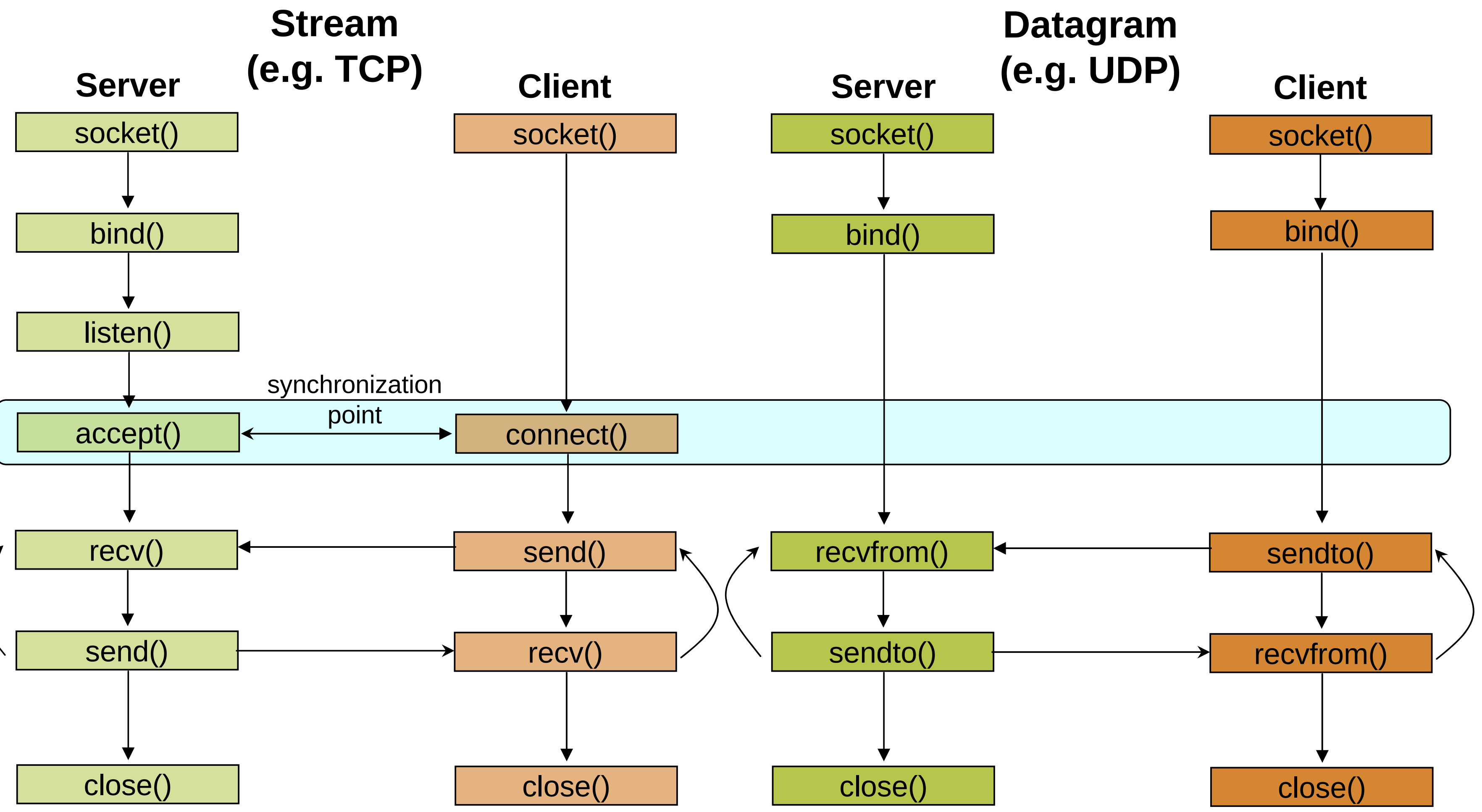
Assign address to socket: `bind()`

- Instructs TCP protocol implementation to listen for connections

- `int status = listen(sockid, queueLimit);`
  - sockid: integer, socket descriptor
  - queuelen: integer, # of active participants that can “wait” for a connection
  - **status**: 0 if listening, -1 if error
- `listen()` is **non-blocking**: returns immediately

- The listening socket (sockid)
  - is never used for sending and receiving
  - is used by the server only as a way to get new sockets

Client - Server Communication - Unix



Establish Connection: `connect()`

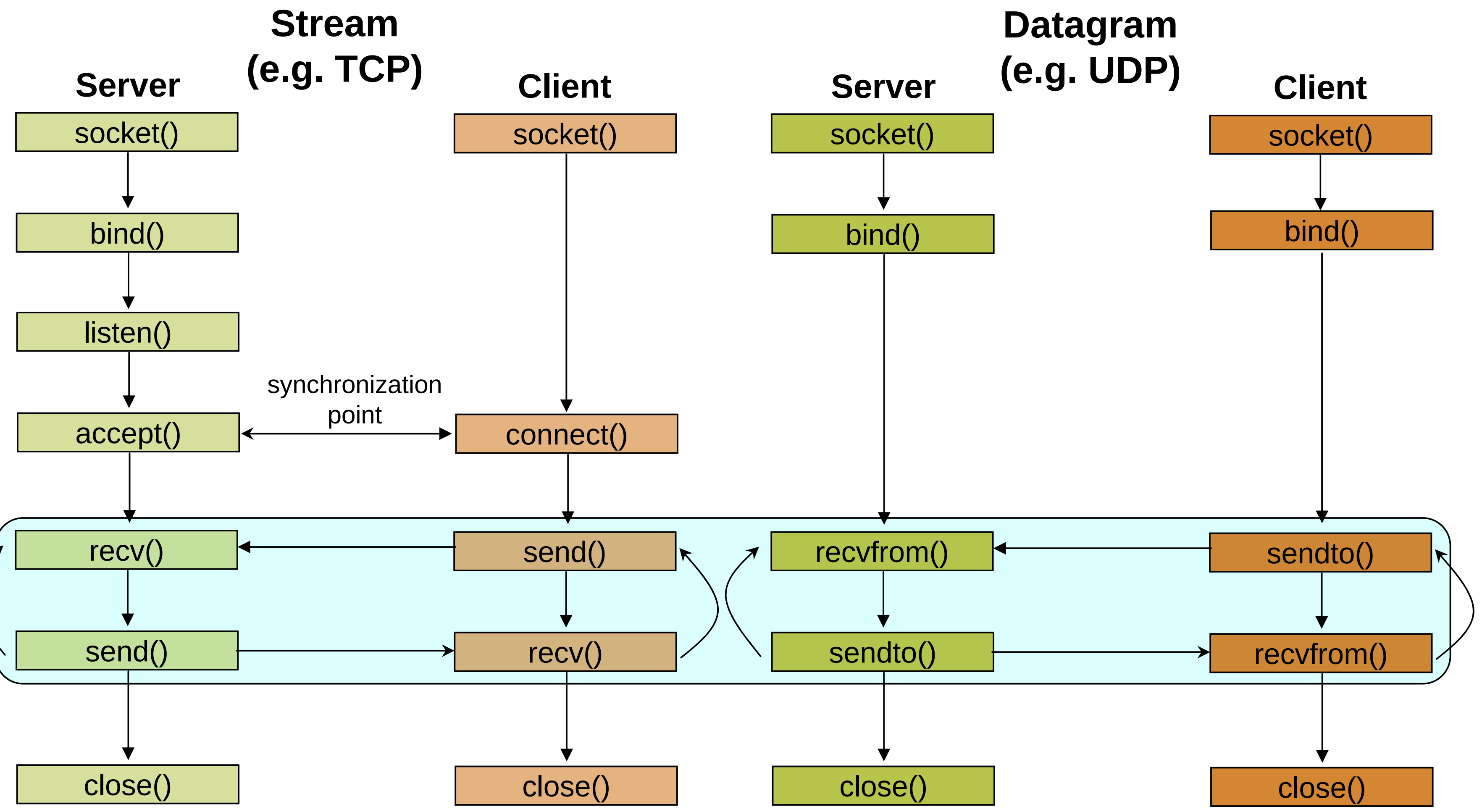
- The client establishes a connection with the server by calling `connect()`

- `int status = connect(sockid, &foreignAddr, addrlen);`
  - sockid: integer, socket to be used in connection
  - foreignAddr: struct sockaddr: address of the passive participant
  - addrlen: integer, sizeof(name)
  - status: 0 if successful connect, -1 otherwise
- `connect()` is **blocking**

Incoming Connection: `accept()`

- The server gets a socket for an incoming client connection by calling `accept()`
- `int s = accept(sockid, &clientAddr, &addrLen);`
  - s: integer, the new socket (used for data-transfer)
  - sockid: integer, the orig. socket (being listened on)
  - clientAddr: struct sockaddr, address of the active participant
    - filled in upon return
  - addrLen: sizeof(clientAddr): value/result parameter
    - must be set appropriately before call
    - adjusted upon return
- `accept()`
  - is **blocking**: waits for connection before returning
  - dequeues the next connection on the queue for socket (sockid)

Client - Server Communication - Unix



Exchanging data with stream socket

- `int count = send(sockid, msg, msgLen, flags);`
  - msg: const void[], message to be transmitted
  - msgLen: integer, length of message (in bytes) to transmit
  - flags: integer, special options, usually just 0
  - count: # bytes transmitted (-1 if error)
- `int count = recv(sockid, recvBuf, bufLen, flags);`
  - recvBuf: void[], stores received bytes
  - bufLen: # bytes received
  - flags: integer, special options, usually just 0
  - count: # bytes received (-1 if error)
- Calls are **blocking**
  - returns only after data is sent / received

Exchanging data with datagram socket

-
  ```
  int count = sendto(sockid, msg, msgLen, flags,
  &foreignAddr, addrlen);
  ```
  - msg, msgLen, flags, count: same with `send()`
  - foreignAddr: struct sockaddr, address of the destination
  - addrLen: sizeof(foreignAddr)

-
  ```
  int count = recvfrom(sockid, recvBuf, bufLen,
  flags, &clientAddr, addrlen);
  ```
  - recvBuf, bufLen, flags, count: same with `recv()`
  - clientAddr: struct sockaddr, address of the client
  - addrLen: sizeof(clientAddr)
- Calls are **blocking**
  - returns only after data is sent / received

Example - Echo

- A client communicates with an “echo” server
- The server simply echoes whatever it receives back to the client