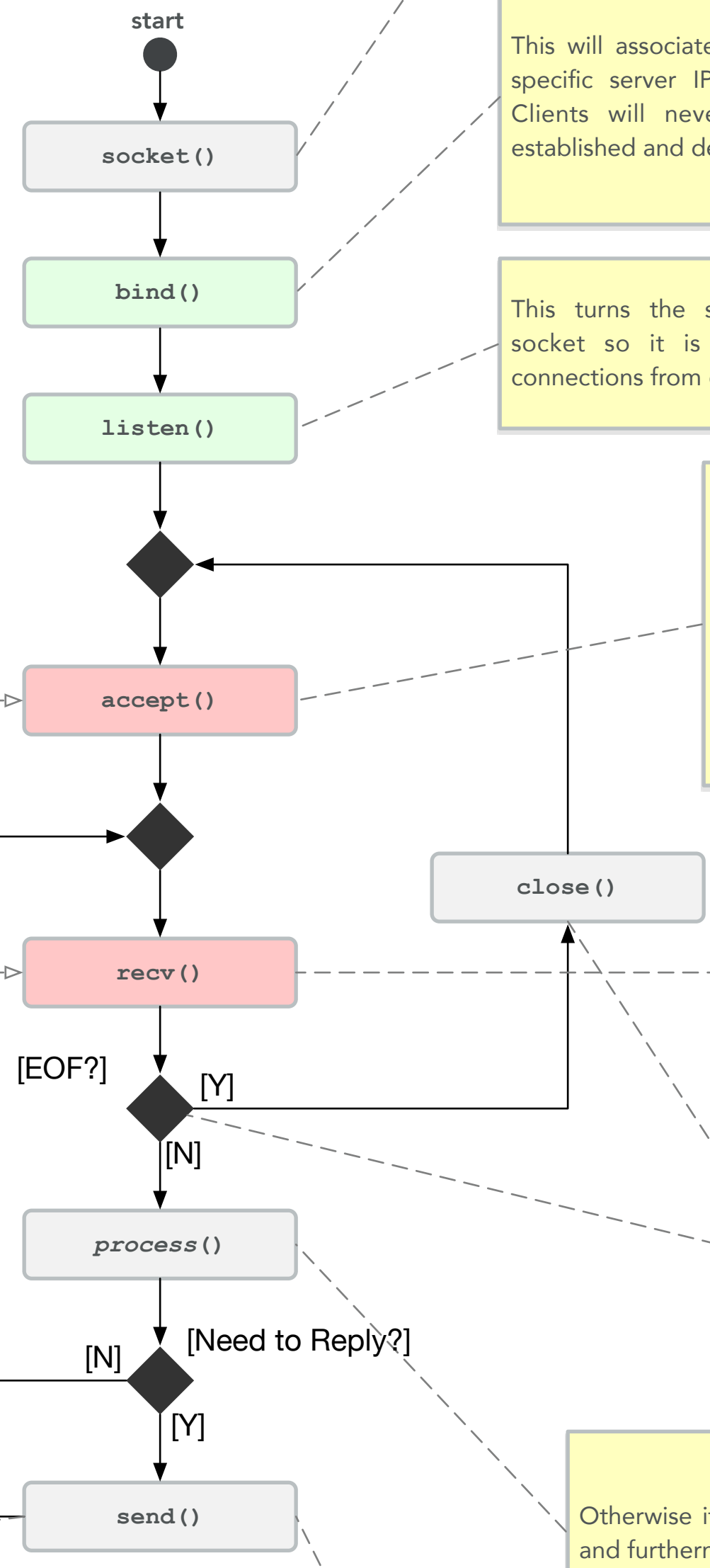


**TCP Server**



Create a new [listening] Socket. The server will 'listen' for connection requests via this socket.

This will associate your listening socket with a specific server IP address and Port Number. Clients will never 'find you' if this is not established and decided upon.

This turns the socket into a 'listening' socket so it is capable of 'accepting' connections from clients.

This causes the socket to now 'wait' for new connection requests. If there are NO requests pending it causes the server process to go to 'block' (and possibly go to sleep). Your code will 'hang here' until a client attempts to make a connection request with the 'connect()' function.

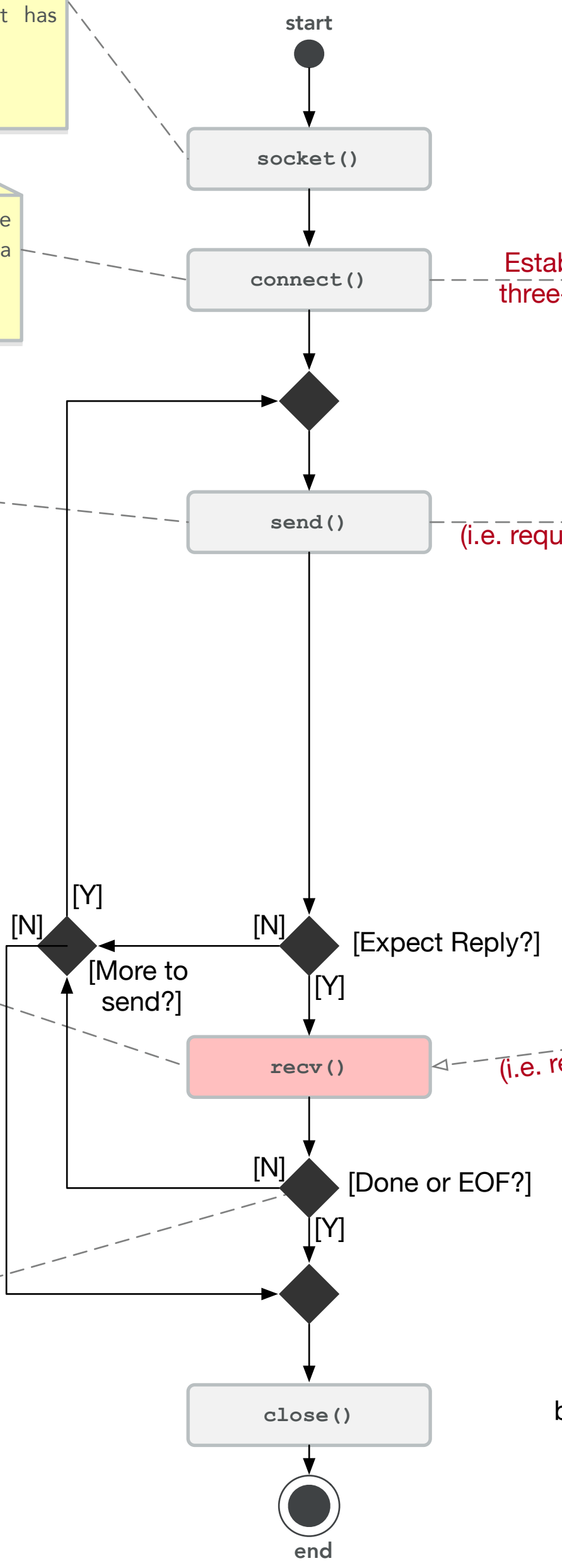
A connection means there's something to read, so we do so here, and store the received data in a buffer of some sort.

If we receive "end-of-file" notification we should close the connection. An EOF is implied when 'recv()' indicates it received zero bytes. This means the client terminated and closed its own connection.

Otherwise it is up to you how you process and furthermore interpret this data.

At this point the client *may* be WAITING for some kind of reply. What you send back is up to you, but *something* must be sent.

**TCP Client**



Creates a new socket, sets the domain (e.g. AF\_INET) designated as TCP or UDP (SOCK\_STREAM or SOCK\_DGRAM), but has NO associated IP or Port number yet.

Attempts to establish a TCP or UDP connection with a remote host by specifying its Port and IP address (contained in a special socket structure).

Once a connection has been established we can use this function to send ANY arbitrary data that is stored in some kind of preallocated buffer. It works pretty much like the 'write()' system call (but has more options).

In a client-server architecture, your server should REPLY back with some meaningful message, and therefore you should 'read' back what you 'expect'. This depends on your application. This could be something as simple as the server sending the string "ACK" back to the client.

Depending on the meaning of your messages you have to decide whether this means you're done or not. Once you're done with the connection, you must close it. The server will know the connection has been closed when the server attempts to read and instead gets 'zero bytes'.

Establish TCP Conn.  
three-way handshake

Data  
(i.e. request something)

Data  
(i.e. reply something)