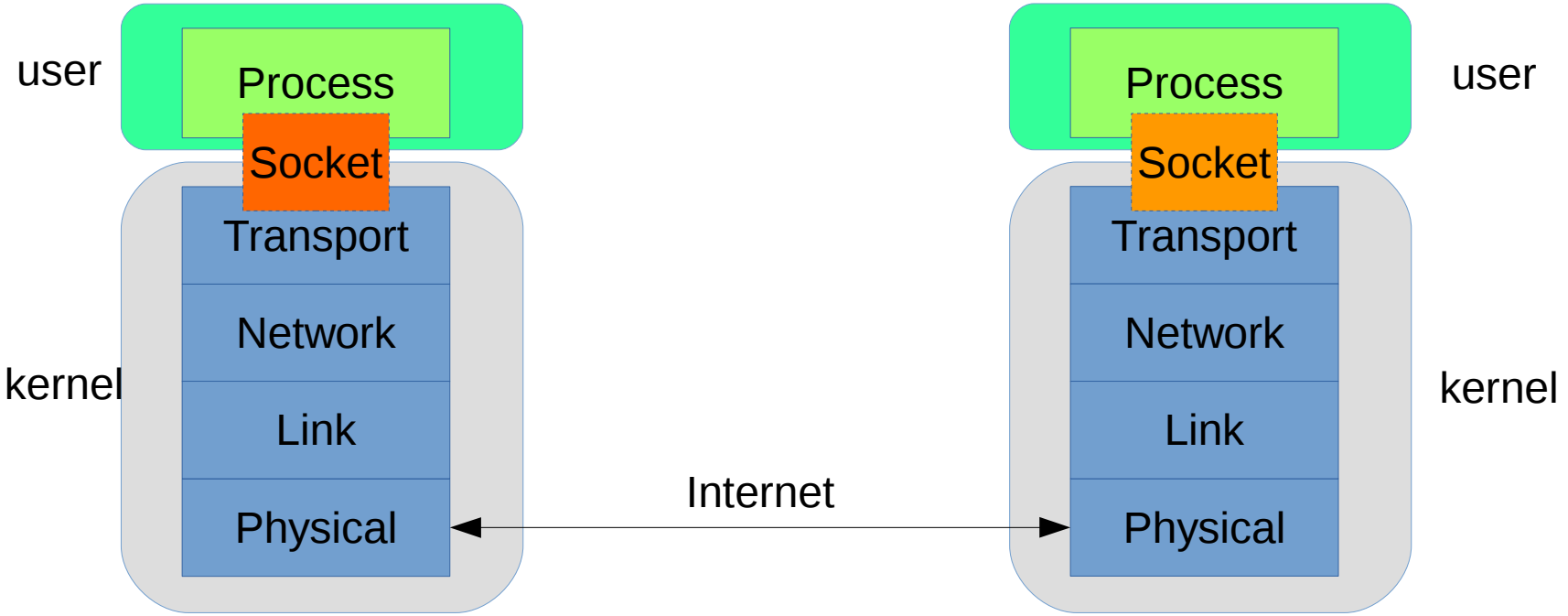
Linux Socket Programming

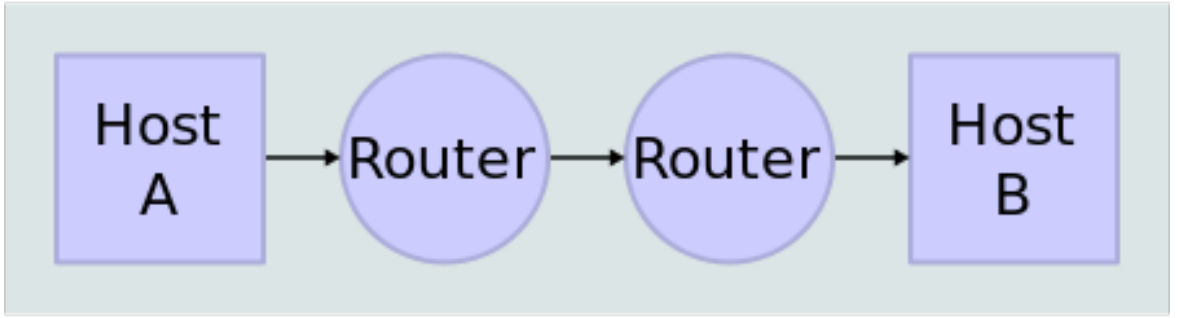
What is a Socket?



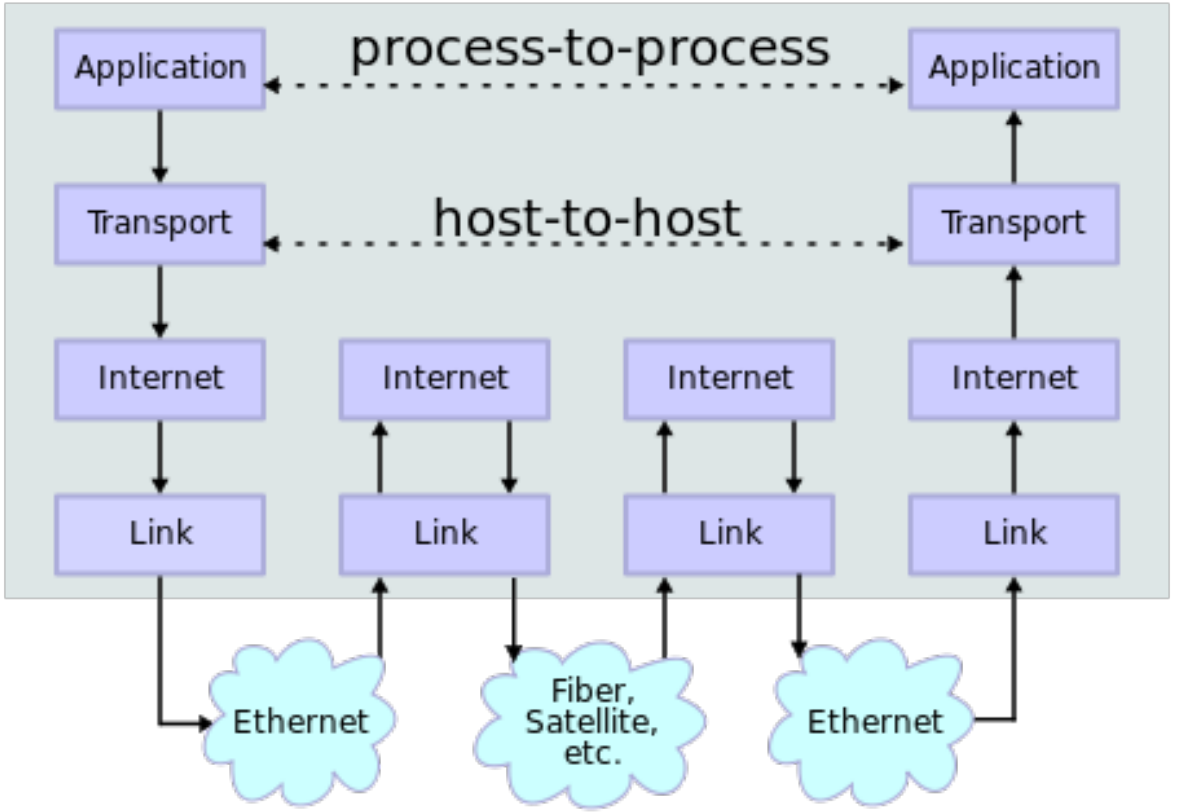
An interface between an application process and transport layer

Process needs peer process host IP Address and Port number

Network Topology



Data Flow



HTTP, SSH ...

UDP, TCP ...

IP

Socket types*

SOCK_STREAM	**SOCK_DGRAM**
Default protocol (TCP)	UDP
Connection oriented	Connection-less
Reliable	Unreliable
In order delivery	No guarantee
Feedback channel	No Feedback channel
eg. File transfer	eg. Voice, realtime services

Terminal Command*: man socket*

Preliminaries: Byte ordering

Little Endian	Big Endian (Network byte order)
Lower byte at lower memory address location	lower byte at Higher memory address location
Intel, Alpha etc.	Sun, Motorola, IBM etc.

Conversion functions

– *htons*() - Host to Network Short

– *htonl*() - Host to Network Long

– *ntohs*() - Network to Host Short

– *ntohl*() - Network to Host Long

Preliminaries: Socket structures

To access or attach IP address and Port Number to socket

```
struct sockaddr_in {
        short           sin_family;   // e.g. AF_INET
        unsigned short   sin_port;     // port number
        struct in_addr   sin_addr;     // see struct in_addr below
        char            sin_zero[8];  // zeros
    };  // size = 2+2+4+8 = 16
struct in_addr {
        unsigned long s_addr;  // IP address
    };

struct sockaddr {
    unsigned short    sa_family;
    char              sa_data[14];
};   //size 2+14=16
```

Typecasting : (struct sockaddr) struct sockaddr_in

Preliminaries: IP address conversations

inet_ntoa() - Converts a network address storeed in a **struct** in_addr to a dots-and-numbers format string xxx.xxx.xxx.xxx.

```
struct sockaddr_in ipa;
char *ipastring;
int inet_aton("127.0.0.1", &ipa.sin_addr);
```

inet_aton() - Converts a dots-and-numbers string into a **struct** in_addr

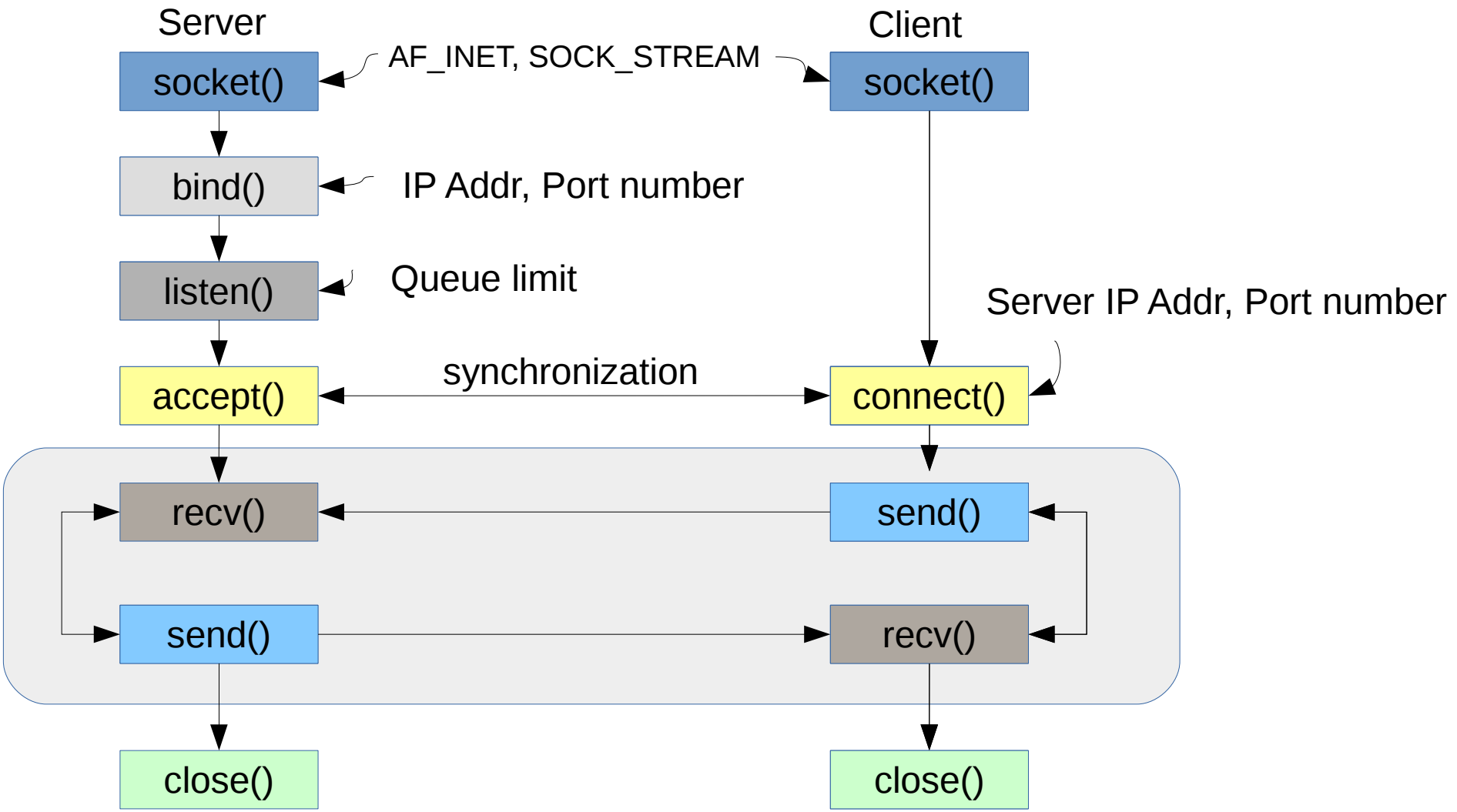
```
struct sockaddr_in ipa;
ipa.sin_addr = inet_addr("127.0.0.1");
```

inet_addr() - Similar to inet_aton (older version)

```
struct sockaddr_in ipa;
ipa.sin_addr = inet_addr("127.0.0.1");
```

Similar functions: inet_ntop(), inet_pton();

TCP-Client-Server Programming



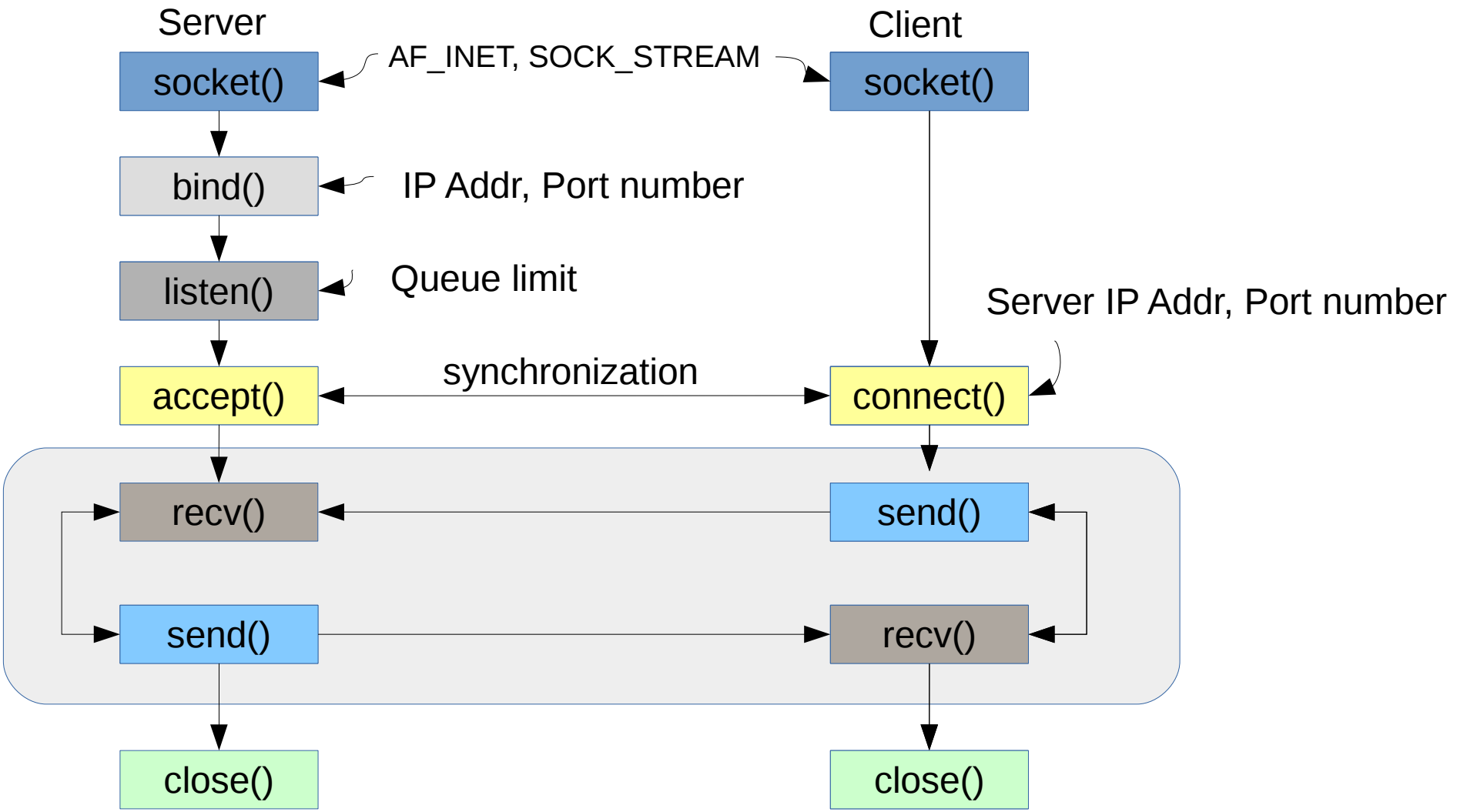
Creat a socket: *socket()*

int sock_id = *socket* (address_family, socket_type, flag);

address_family	:	AF_INET internet IPv4 AF_INET6 internet Ipv6
socket_type	:	SOCK_STREAM SOCK_DGRAM
flag	:	0, specify if to use other than default protocol
Return	:	Socket desriptor (similar to file descriptor in file handling)

Terminal Command*: man socket*

TCP-Client-Server Programming



Assign address to socket: *bind()*

Associates and reserves a port for use by the socket

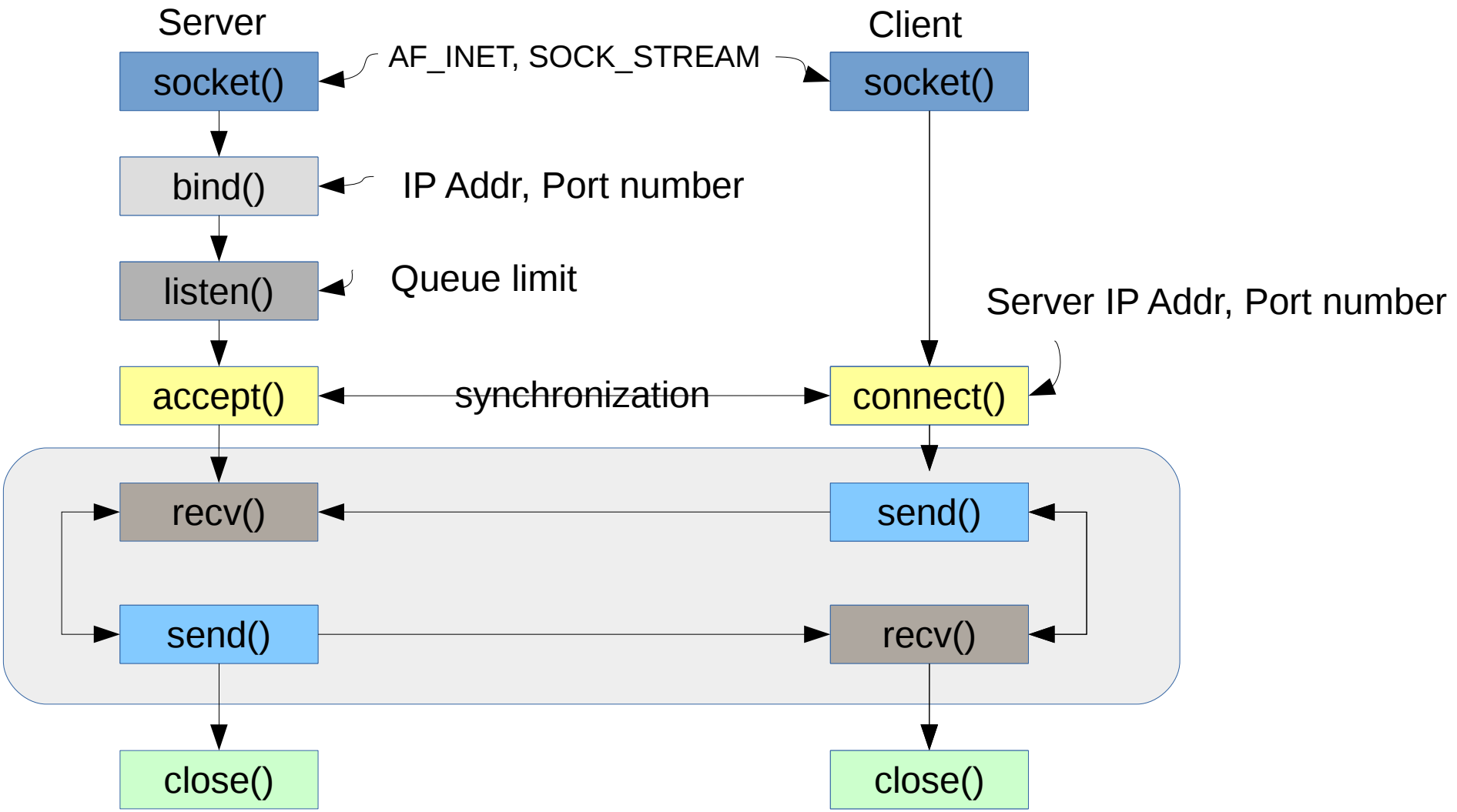
int status = *bind* (socket_id, &addport, sizeofaddrport);

socket_id	:	socket descriptor i.e. return int from *socket()*

addrport	:	**struct** sockaddr_in addrport;
		addrport.sin_family = AF_INET;// IPv4
		addrport.sin_port = *htons*(5000);
		addrport.sin_addr.s_addr = *htonl*(INADDR_ANY);
		OR
		addrport.sin_addr = *inet_addr*(*"xxx.xxx.xxx.xxx"*);
		OR
		int valid = *inet_aton("xxx.xxx.xxx.xxx",* &addrport.sin_addr);

sizeofaddrport	:	size of struct sockaddr_in i.e. *sizeof*(addport)
status	:	-1 upon failure

TCP-Client-Server Programming



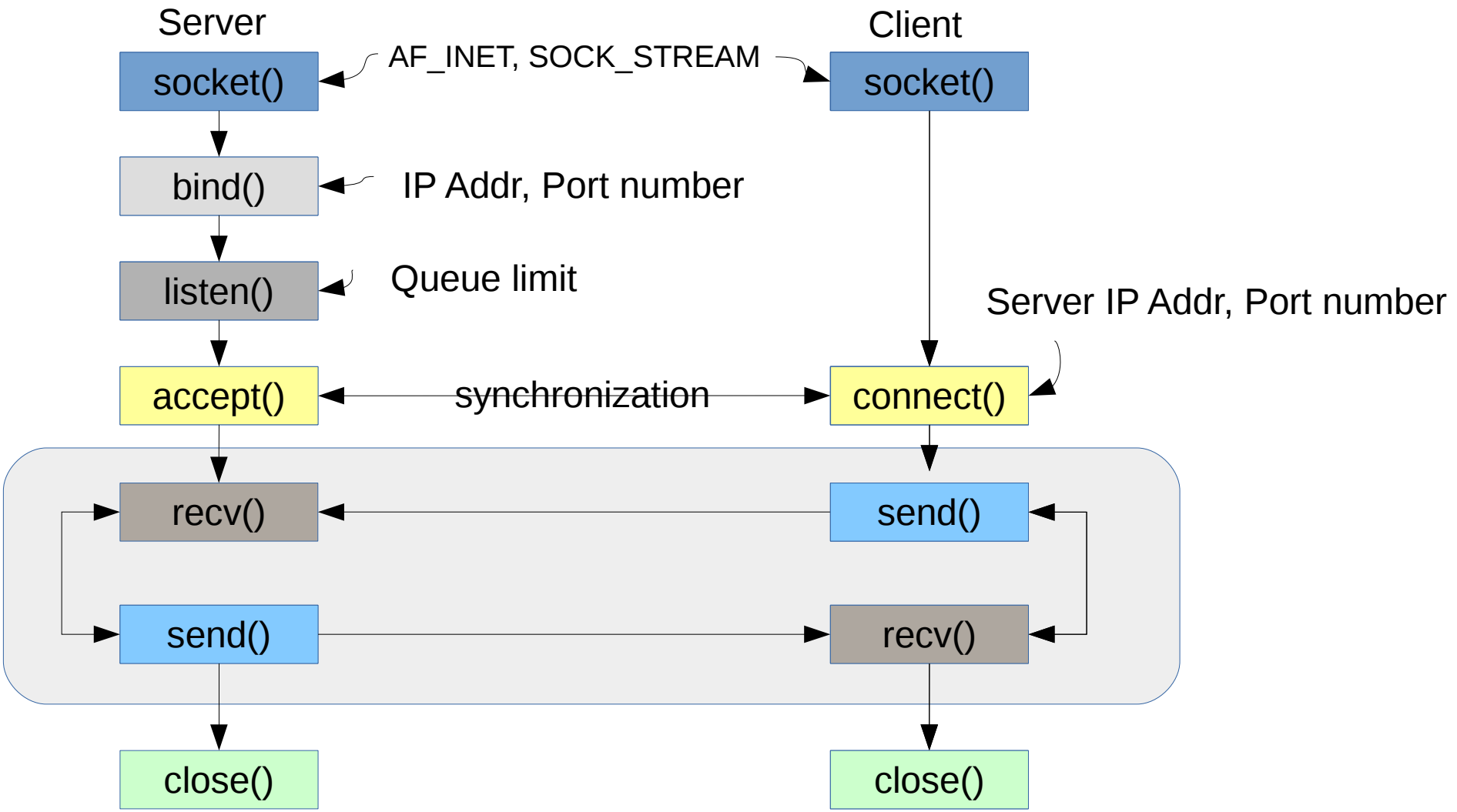
Instruct socket to listen for connection: *listen()*

Enqueue several clients and service each sequencially or concurrently
listen() is a non-blocking system call

```
int status = listen(socket_id, QueueLimit);
```

socket_id	:	socket descriptor i.e. return int from *socket()*
QueueLimit	:	integer value, defines number of active clients it can service
status	:	-1 upon failure, 0 on success

TCP-Client-Server Programming



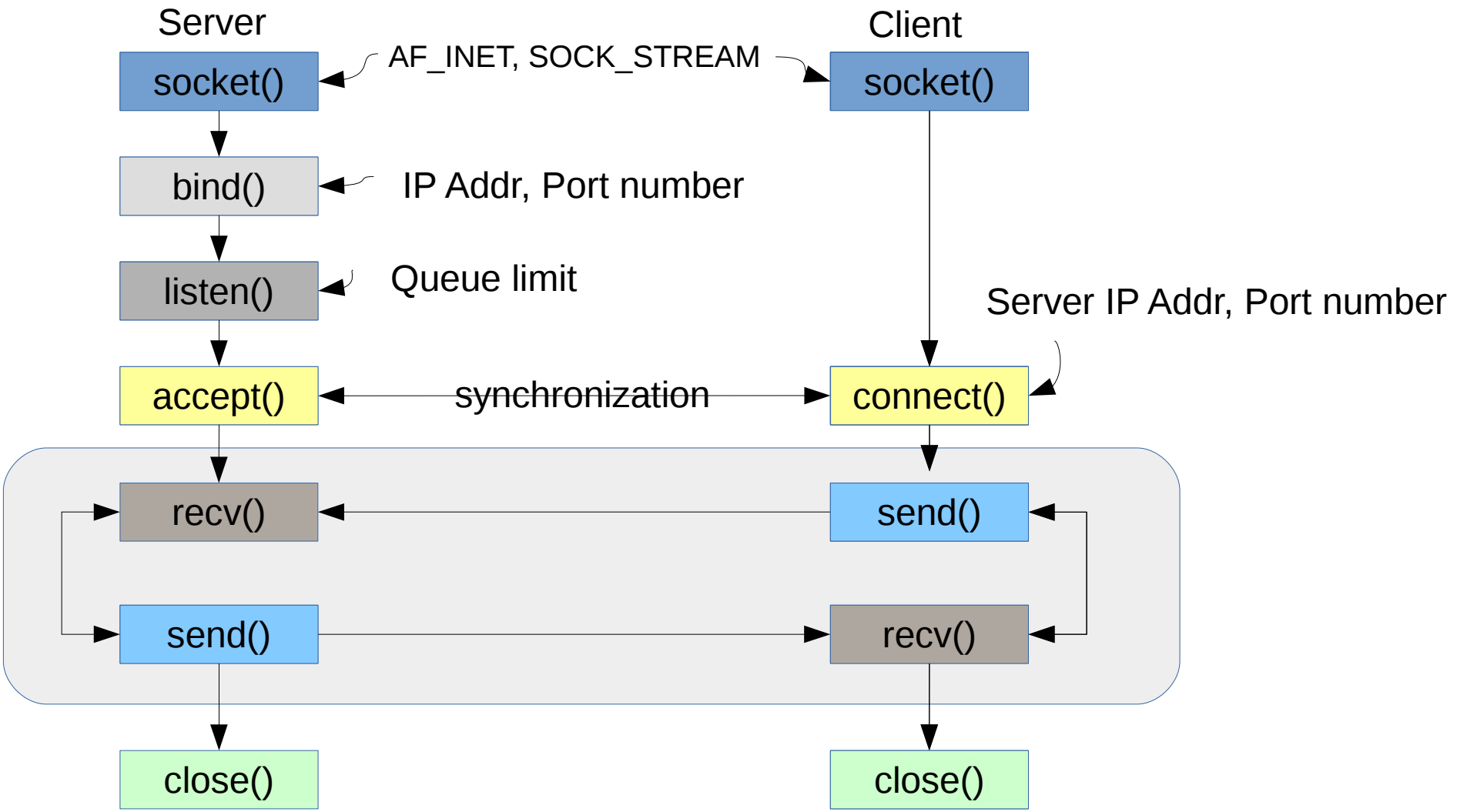
Service incoming Connection: *accept()*

Service clients by taking client from top of the queue, and dequeue next
accept() is a blocking type system call

int serv_socket_id = *accept*(socket_id, &client_addr, &addrLen);

socket_id	:	socket descriptor i.e. return int from *socket()*
client_addr	:	**struct** sockaddr_in client_addr; Filled in upon return Contains IP address and port number of incoming client
addrLen	:	*sizeof*(**struct** sockaddr_in)
serv_socket_id	:	function returns new socket descriptor through through which actual communication takes place

TCP-Client-Server Programming



Establish Connection: *connect*()

The client establishes a connection with the server by calling *connect*()
connect() is a blocking type system call

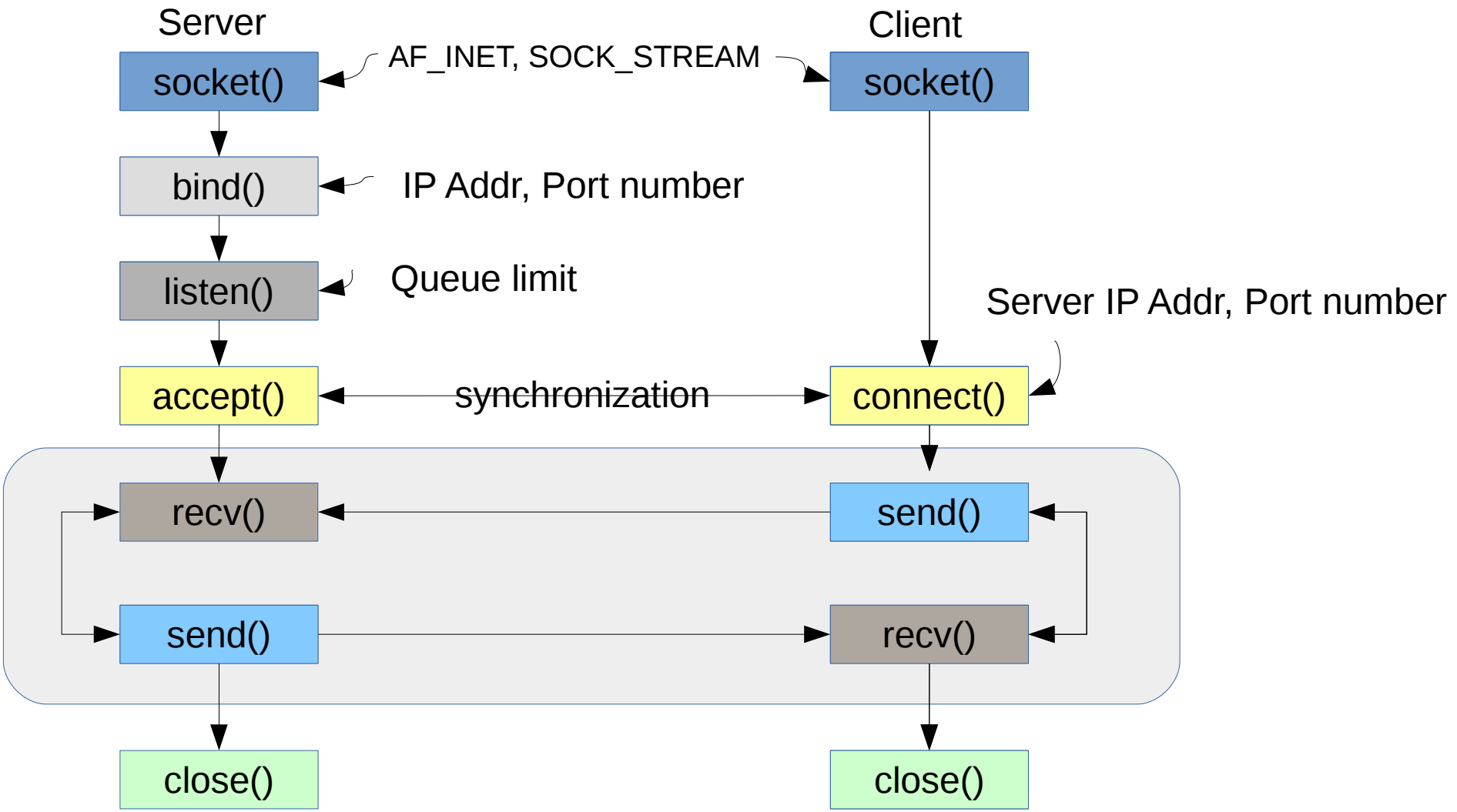
int status = *connect*(socket_id, &server_addr, addrLen);

socket_id	:	socket descriptor i.e. return int from *socket()*

server_addr	:	Filled in before calling *connect()*
		Contains IP address and port number of server
		server_addr.sin_family = AF_INET;// IPv4
		server_addr.sin_port = *htons*(5000);
		server_addr.sin_addr =
		inet_addr(*"xxx.xxx.xxx.xxx"*);
		OR
		int valid = *inet_aton("xxx.xxx.xxx.xxx",*
		&server_addr.sin_addr);

addrLen	:	*sizeof*(**struct** sockaddr_in)
status	:	-1 upon failure, 0 on success

TCP-Client-Server Programming



Actual communication

Sending data

int count = *send*(sock_id, msg, msgLen, flags);

OR

int count = *write*(sock_id, msg, msgLen)

Receiving data

int count = *recv*(sock_id, msg, msgLen, flags);

OR

int count = *read*(sock_id, msg, msgLen);

socket_id	:	socket descriptor i.e. return int from *socket() or accept()*
msg	:	char array containing message/data
msgLen	:	maximum size of msg array
flags	:	0 (default)
count	:	number of bytes processed on a given call

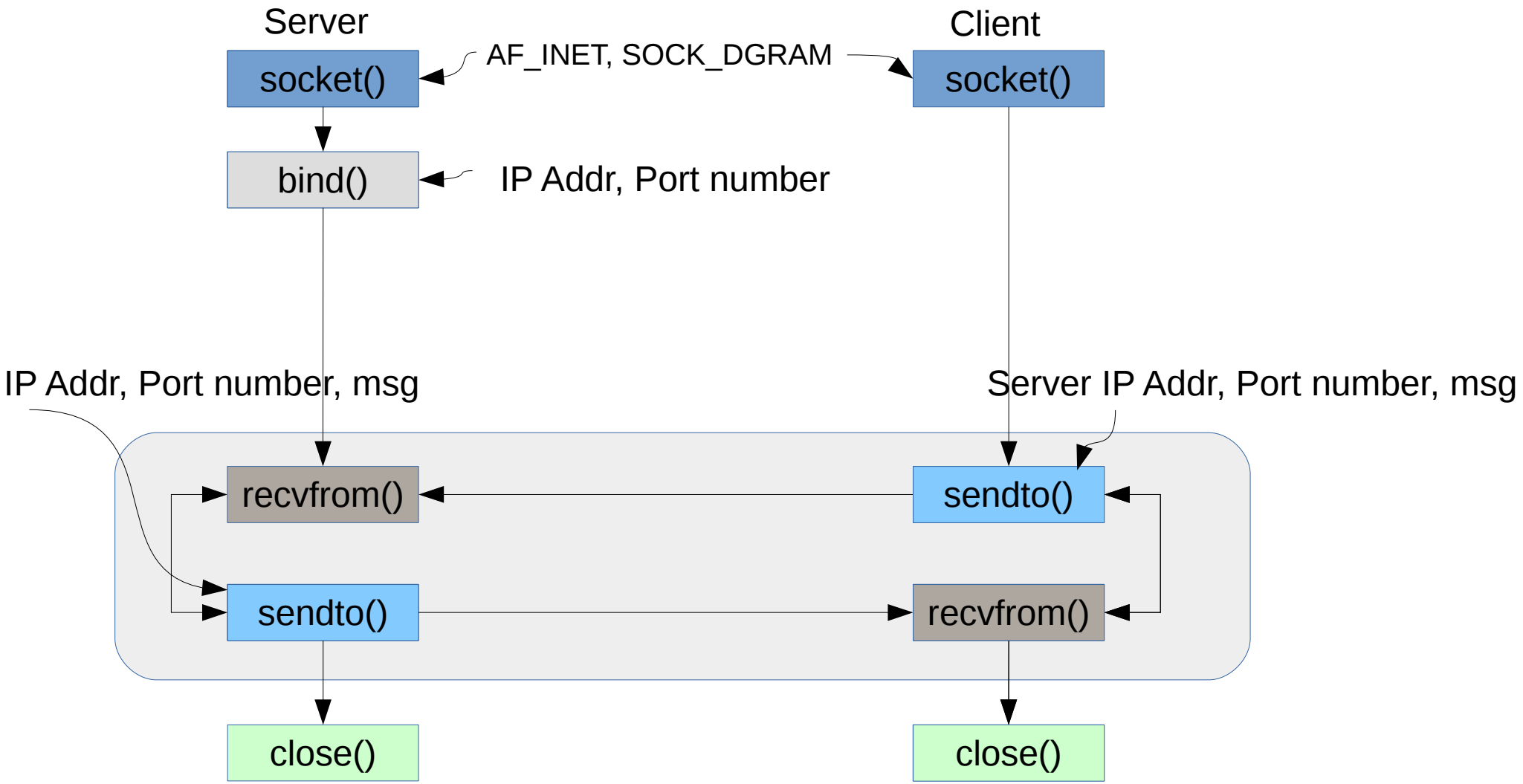
Close socket: *close()*

- To close the socket and inform peers

```
int status = close(socket_id);
```

socket_id	:	socket descriptor i.e. return int from *socket() or accept()*
status	:	returns 0 upon success

UDP Client-Server programming

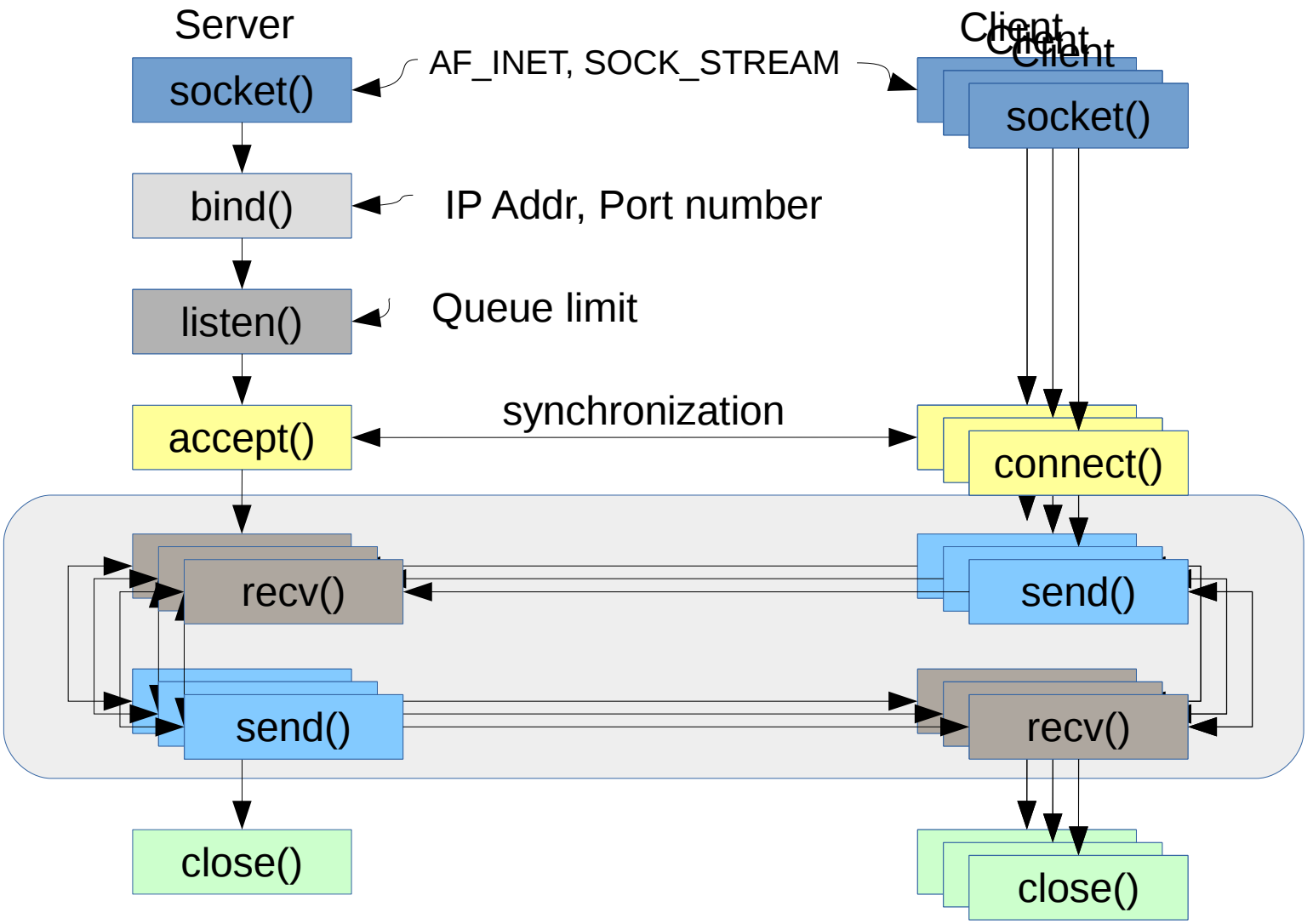


Connectionless communication

int count = *sendto*(socket_id, msg, msgLen, flags, &peer_addr, addrlen);
int count = *recvfrom*(socket_id, msg, msgLen, flags, &peer_addr, &addrlen);

socket_id	:	socket descriptor i.e. return int from *socket()*
msg	:	char array containing message/data
msgLen	:	maximum size of msg array
flags	:	0 (default)
peer_addr	:	**struct** sockaddr_in If *sendto()* is called before *recvfrom()*, peer_addr has to be filled up before calling *sendto()*
addrLen	:	size of **struct** sockaddr_in
count	:	number of bytes processed on a given call

Concurrent TCP-Client-Server



Concurrent server using *fork()*

```
...
...
listen(socket_id, Queuelimit);
while(1){
 serv_socket_id = accept(socket_id, &client_addr, &addrLen);

      If(fork()==0){ //child process
      ...
      handle client using read(), write()
      ...
      close(serv_socket_id);  // child copy of socket
      exit(0); // child exits
                        }
close(serv_socket_id);        // parent copy of socket
}
```

Sample codes

- udpc.c- UDP client program
- udps.c- UDP server program
- tcpc.c- TCP client program
- tcps.c- TCP server program
- tcpfc.c- TCP client program to download any file
- tcpfs.c- TCP server program to upload any file
- ctcps.c-Concurrent TCP server program
- ctcps.c-Concurrent TCP server program to upload file

➔ Open terminal

➔ *git clone https://github.com/cdotblr/socket*