Announcements

If you missed this lecture and are reading these slides online, you need to understand that these are not meant as a stand-alone resource but as an accompaniment to attending the class. If you need to learn the material because you missed the class, do the readings in the Kerrisk text.

How do we communicate between processes?

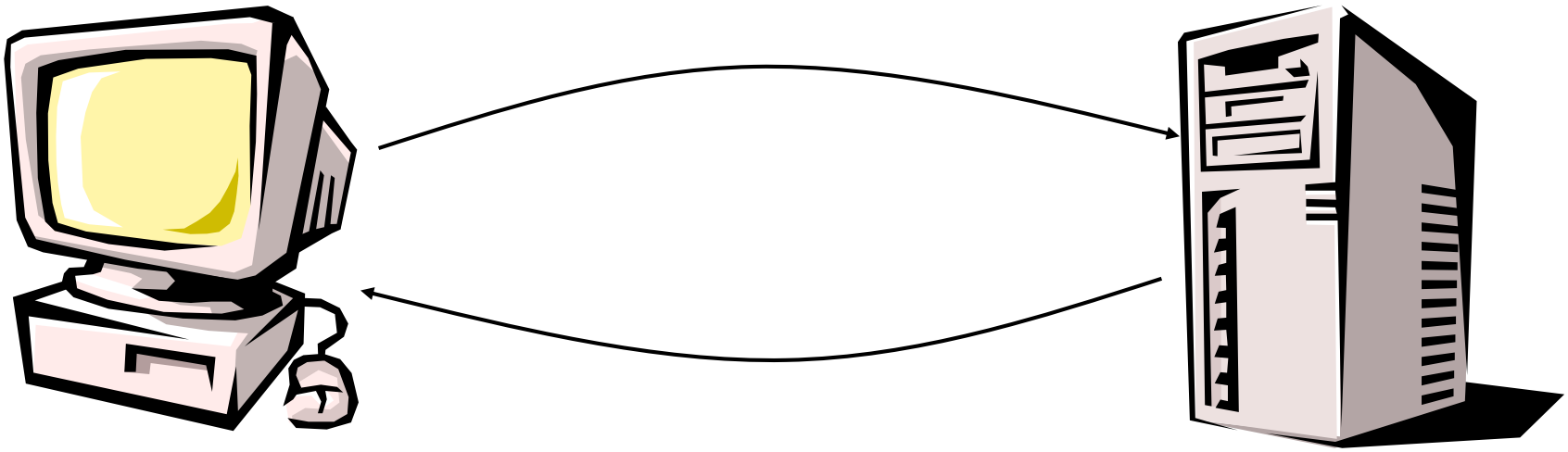
In order to use a pipe for interprocess communication what has to be true?

1. The process writing to the pipe must be the parent of the process reading from the pipe.
2. The process writing to the pipe must be the child of the process reading from the pipe.
3. The processes must be a parent/child combination but the direction of the pipe could go either way.
4. The processes must be on the same machine.

In order to use a pipe for interprocess communication what has to be true?

1. The process writing to the pipe must be the parent of the process reading from the pipe.
2. The process writing to the pipe must be the child of the process reading from the pipe.
3. The processes must be a parent/child combination but the direction of the pipe could go either way.
4. **The processes must be on the same machine.**

Sockets



- Once you set up the socket, the interface behaves like a file (or pipe)
  - entry in the file descriptor table
  - read/write/close
- Except — sockets are full-duplex (2 way)

Really? Across the Net?

- network protocols
- layers
- clients & servers
- addresses, ports, TCP/IP
- packets
- routing

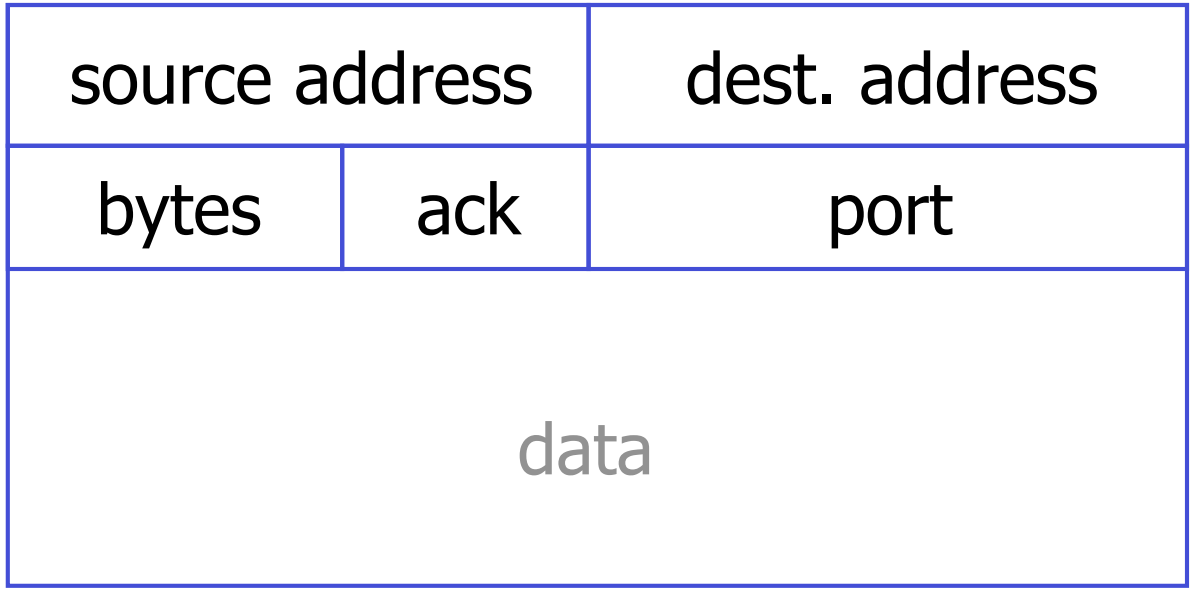
Relax! Because of the layers of protocols you don't need to know all the details of everything.

Protocols

- Computers use several layers of general protocols to communicate.
- To understand why these layers are important, think about how a company sends you an invoice for a purchase.

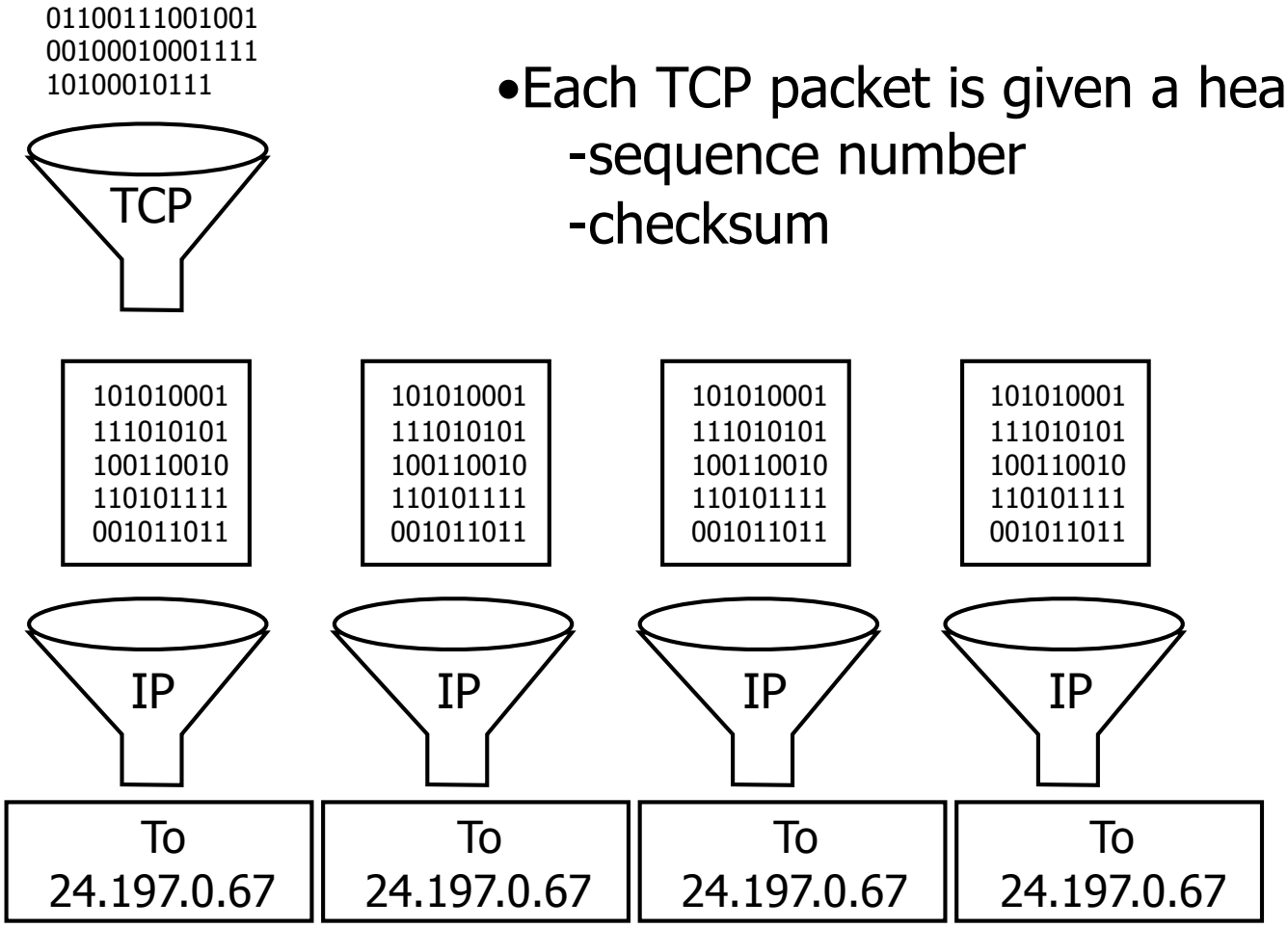
TCP/IP

- Transmission Control Protocol.
- Tells us how to package up the data.



Details

- make packets



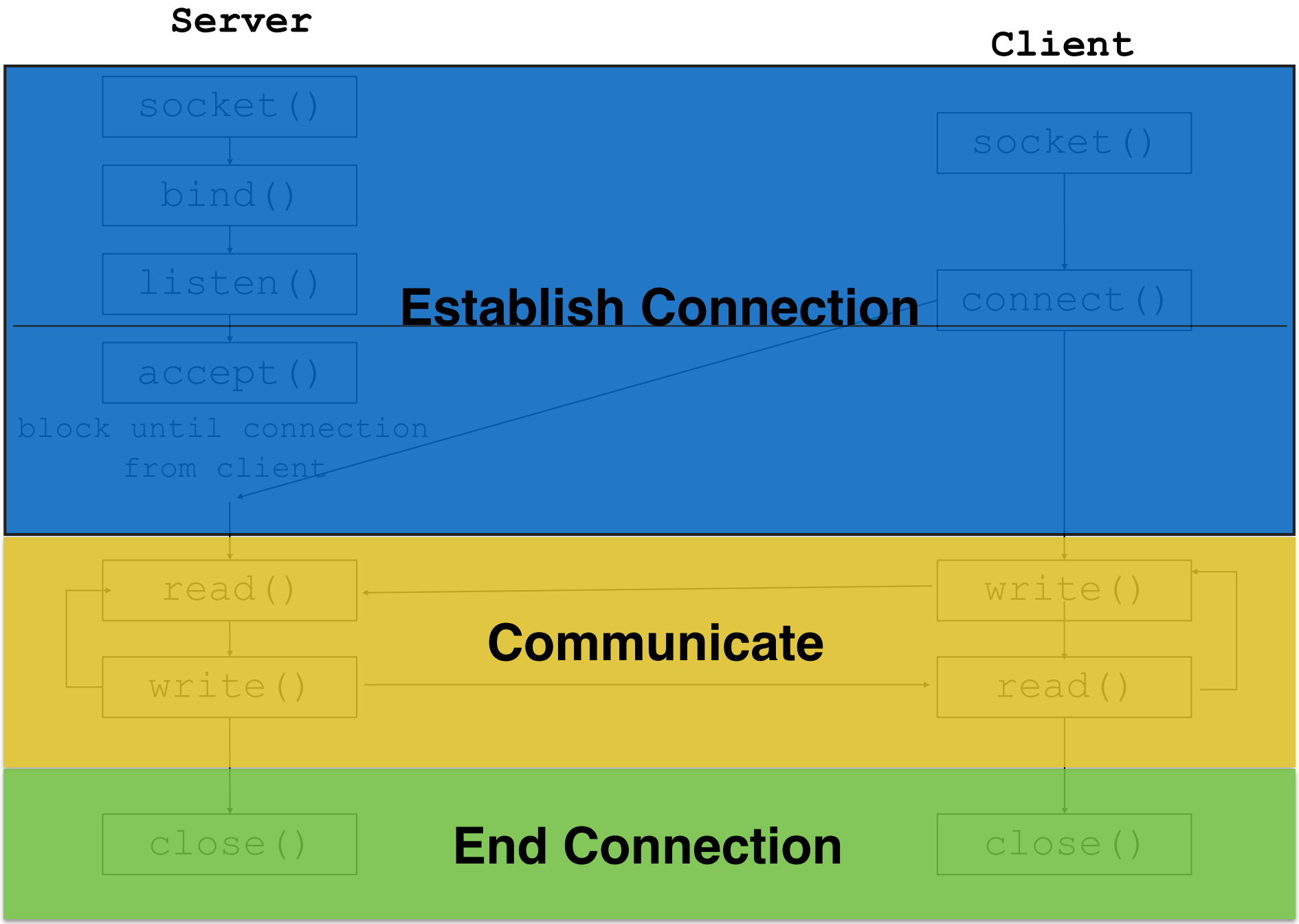
- Each TCP packet is given a header
  - -sequence number
  - -checksum

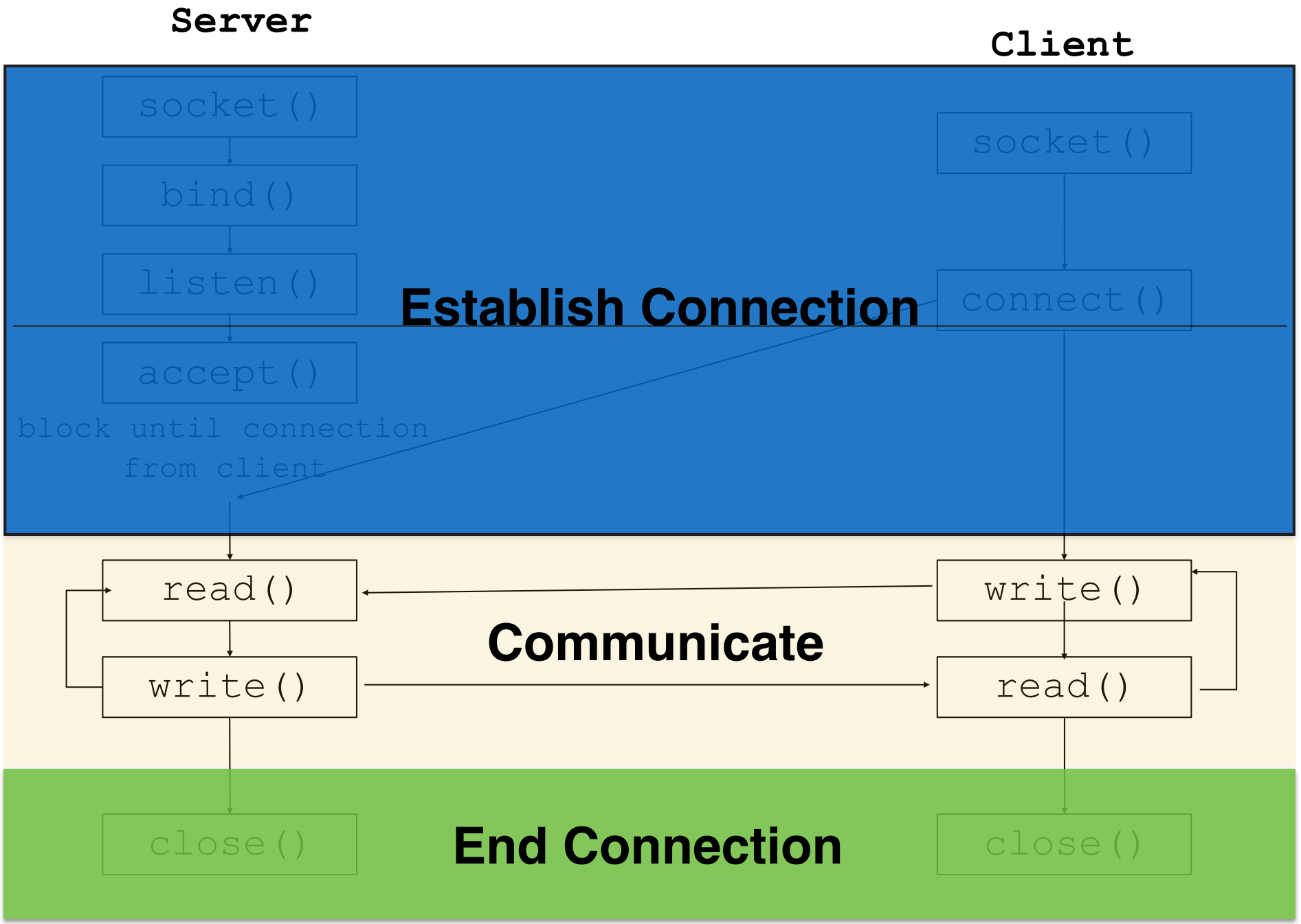
- put in an IP envelope with another header

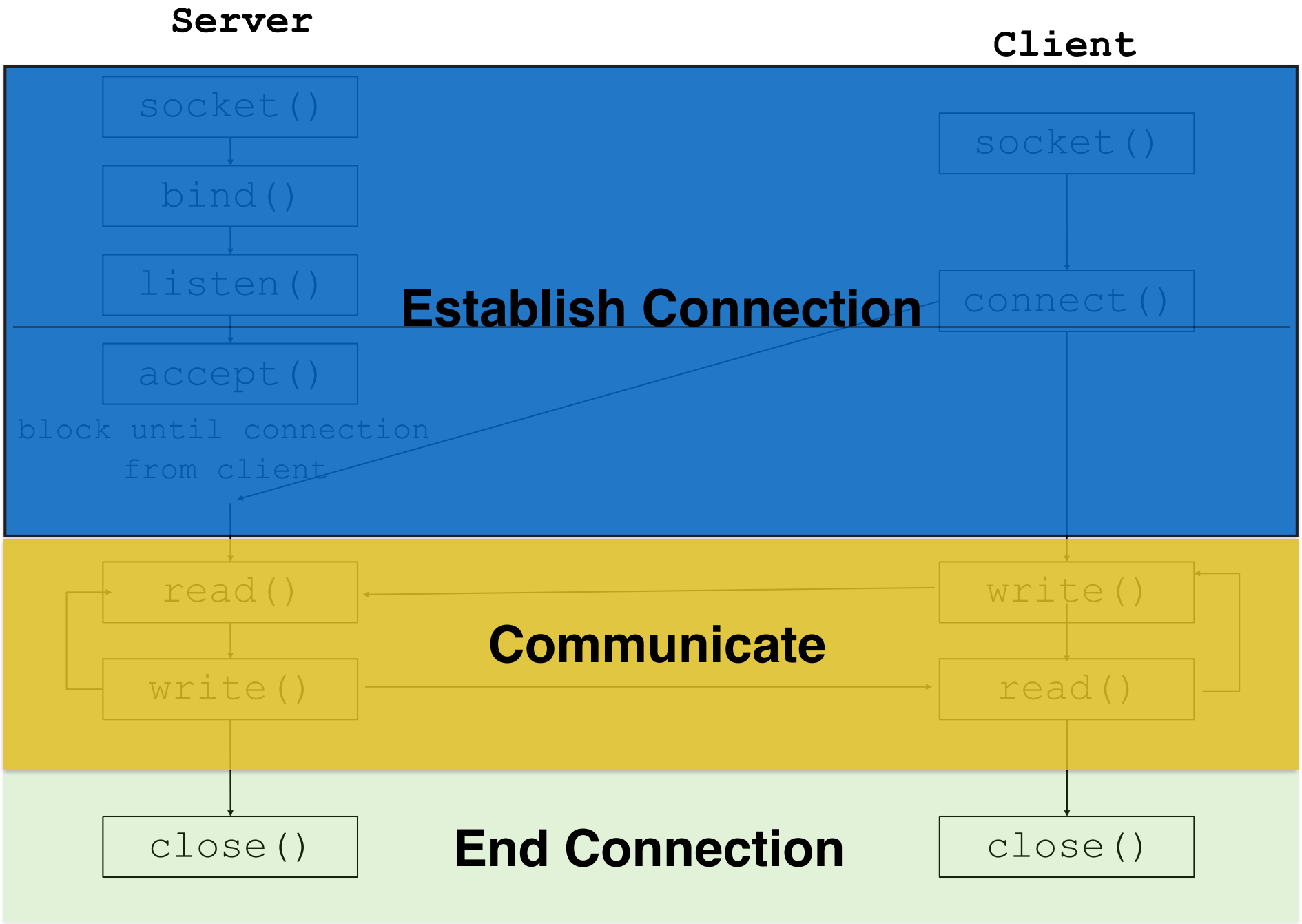
Client Server Model

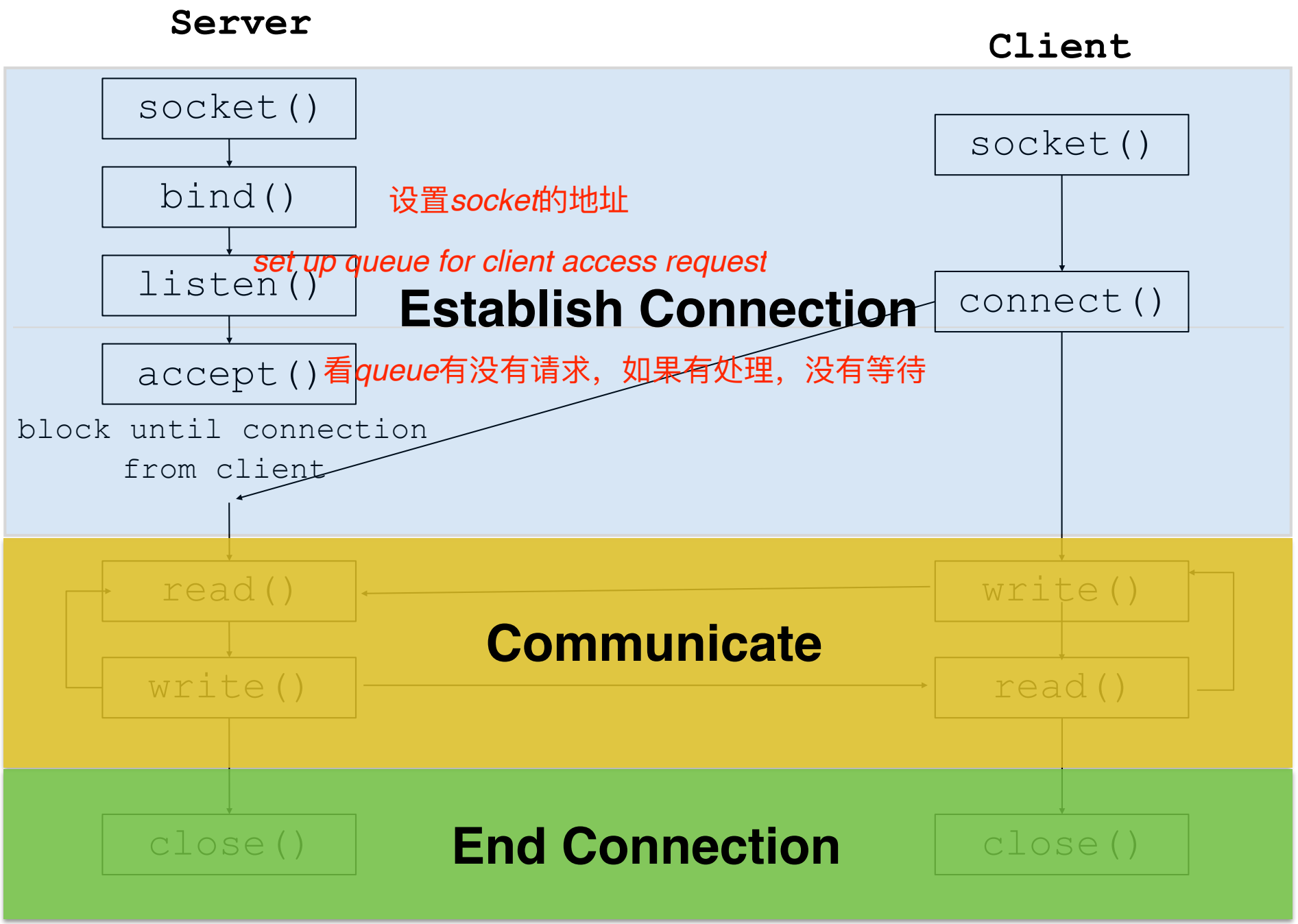
Server: Always available at a known location

Client: Shows up and initiates the connection


Server
Client
socket()
bind()
listen()
accept()
block until connection from client
socket()
connect()
Establish Connection
read()
write()
write()
read()
Communicate
close()
close()
End Connection


Server
Client
socket()
bind()
listen()
accept()
block until connection from client
socket()
connect()
Establish Connection
read()
write()
write()
read()
Communicate
close()
close()
End Connection


Server
Client
socket()
bind()
listen()
accept()
block until connection from client
socket()
connect()
Establish Connection
read()
write()
write()
read()
Communicate
close()
close()
End Connection


Server
Client
socket()
bind()
listen()
accept()
socket()
connect()
设置socket的地址
set up queue for client access request
Establish Connection
看queue有没有请求，如果有处理，没有等待
block until connection
from client
read()
write()
write()
read()
Communicate
close()
close()
End Connection

Learning More at Home

- We are only talking about connection-oriented sockets in CSC 209
- If you want to know about other sockets, read chapters 56-61 in Kerrisk
- If you missed this class and are trying to simply read these slides, read the chapters in Kerrisk listed on the course website

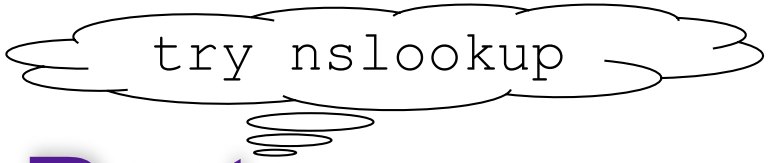
Connection-Oriented

Server

- Create a socket: `socket()`
- Assign an address to a socket: `bind()`
- Establish a queue for connections: `listen()`
- Get a connection from the queue: `accept()`

Client

- Create a socket: `socket()`
- Initiate a connection: `connect()`

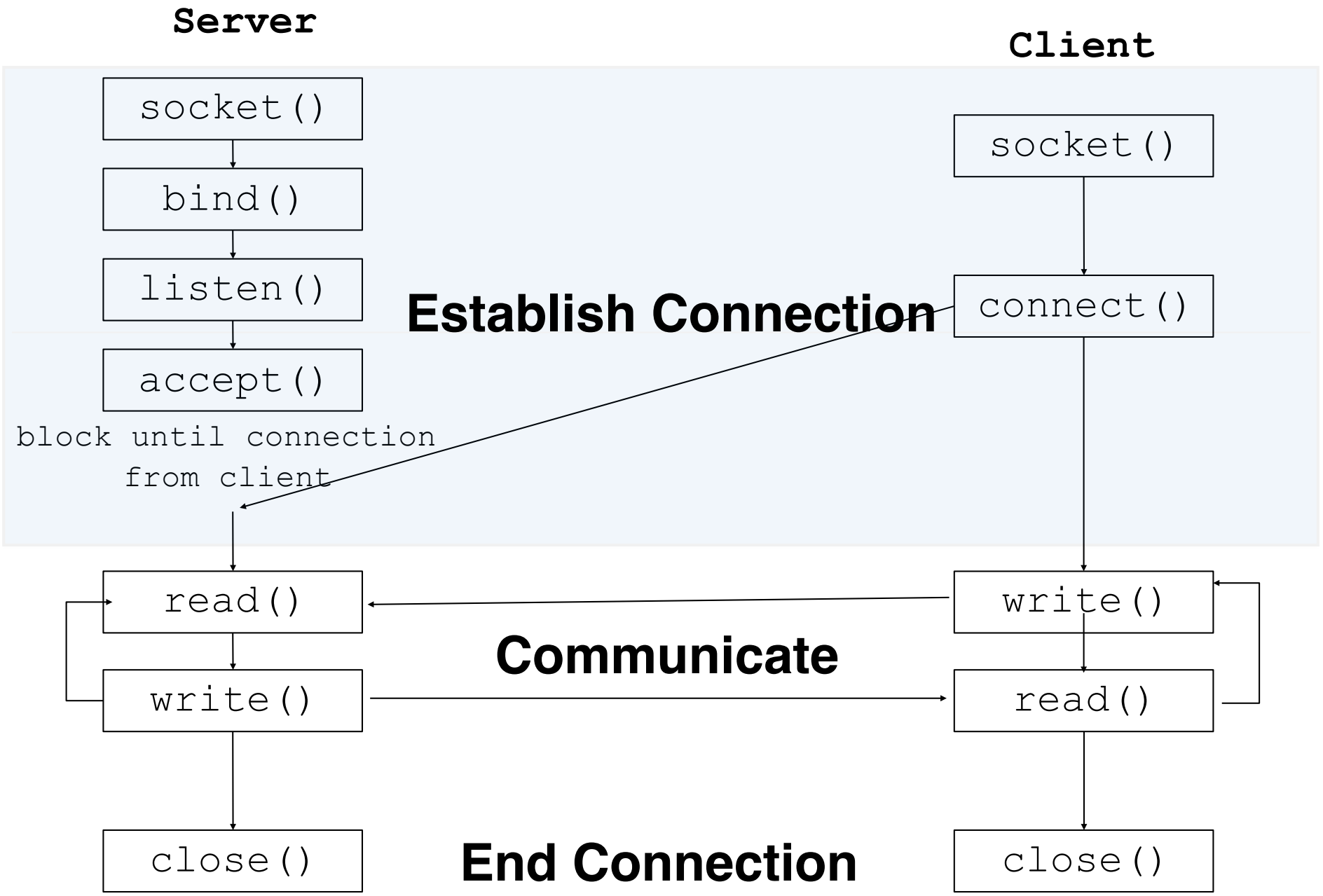


Addresses and Ports

- A socket pair is the two endpoints of the connection.
- An endpoint is identified by an IP address and a port.
- IPv4 addresses are 4 8-bit numbers:
  - 128.100.31.200 = wolf.teach.cs.toronto.edu
- Ports
  - because multiple processes can communicate with a single machine we need another identifier.

More on Ports

- Well-known ports: 0-1023
  - 80 = http
  - 22 = ssh
  - 23 = telnet
  - 21 = ftp
  - 25 = smtp (mail)
  - 194 = irc
- Registered ports: 1024-49151
  - 2709 = supermon
  - 26000 = quake
  - 3724 = world of warcraft
- Dynamic (private) ports: 49152-65535
  - You should pick ports in this range to avoid overlap


Server
Client
socket()
bind()
listen()
accept()
block until connection
from client
socket()
connect()
Establish Connection
read()
write()
write()
read()
Communicate
close()
close()
End Connection

Server side

```
int socket(int family, int type,
           int protocol);
```

- family specifies protocol family:
  - `AF_INET` – IPv4
  - `AF_LOCAL` – Unix domain
- type
  - `SOCK_STREAM`, `SOCK_DGRAM`, `SOCK_RAW`
- protocol
  - set to 0 except for RAW sockets
- returns a socket descriptor

bind to a name

```
int bind(int sockfd,
      const struct sockaddr *servaddr,
       socklen_t addrlen);
```

- sockfd – returned by socket
-
```
struct sockaddr_in {
  short          sin_family; /*AF_INET */
  u_short        sin_port;
  struct in_addr sin_addr;
  char           sin_zero[8]; /*filling*/
};
```
- `sin_addr` can be set to `INADDR_ANY` to communicate on any network interface.

Set up queue in kernel

```
int listen(int sockfd, int backlog)
```

- after calling `listen`, a socket is ready to accept connections
- prepares a queue in the kernel where partially completed connections wait to be accepted.
- `backlog` is the maximum number of partially completed connections that the kernel should queue.

Complete the connection

```
int accept(int sockfd,
           struct sockaddr *cliaddr,
           socklen_t *addrlen);
```

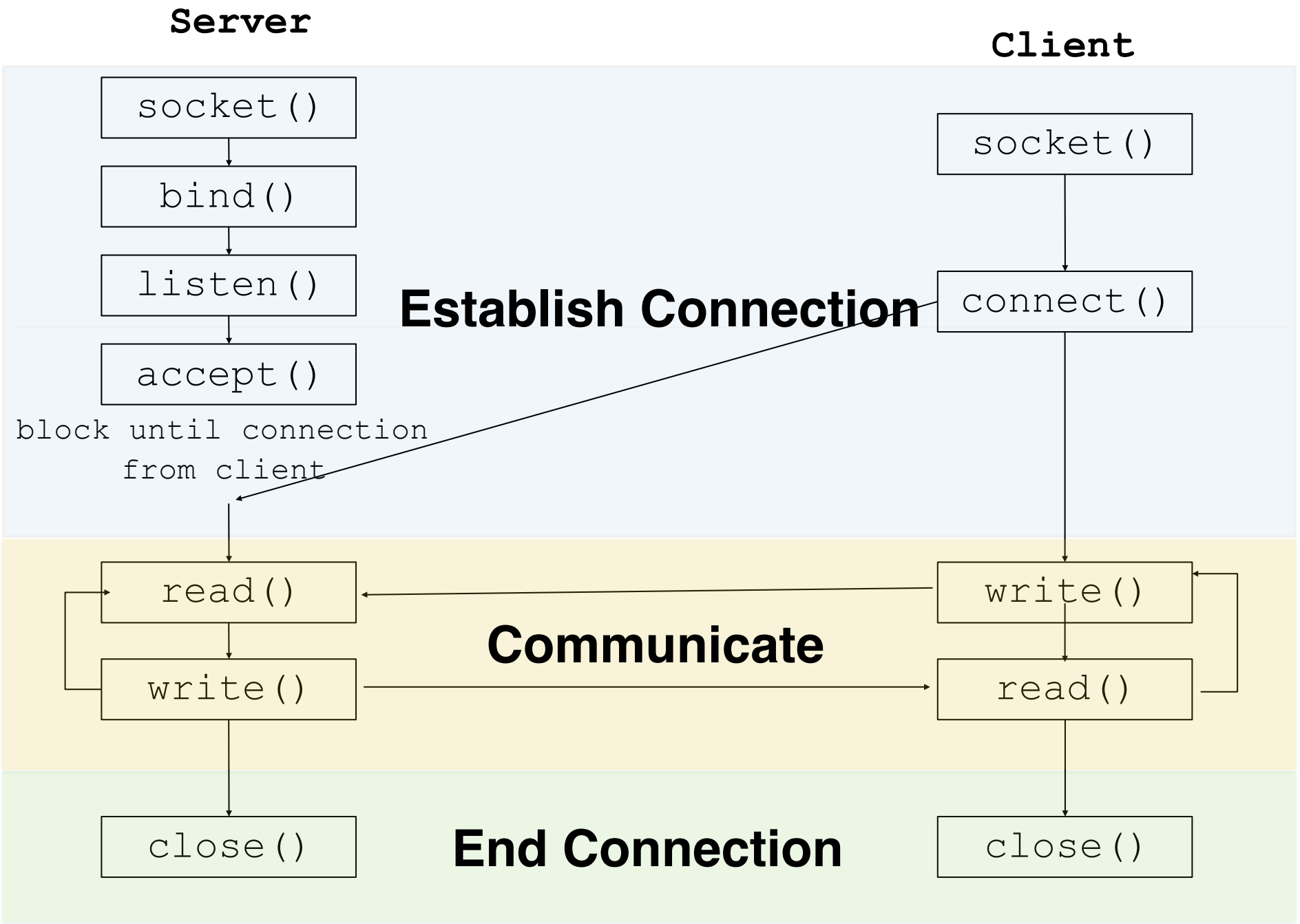
- blocks waiting for a connection (from the queue)
- returns a new descriptor which refers to the TCP connection with the client
- `sockfd` is the listening socket
- `cliaddr` is the address of the client
- reads and writes on the connection will use the socket returned by accept

Client side

- `socket()` – same as server, to say “how” we are going to talk

```
int connect(int sockfd,
      const struct sockaddr *servaddr,
      socklen_t addrlen);
```

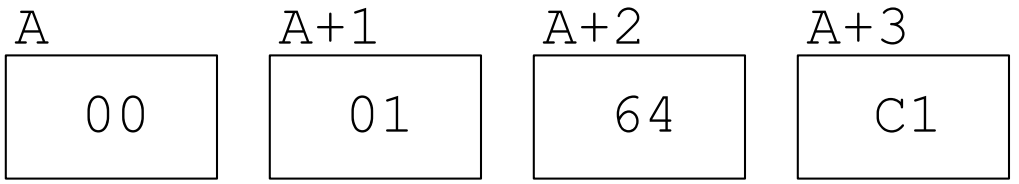
- the kernel will choose a dynamic port and source IP address.
- returns 0 on success and -1 on failure setting errno.


Server
Client
socket()
bind()
listen()
accept()
block until connection
from client
socket()
connect()
Establish Connection
read()
write()
write()
read()
Communicate
close()
close()
End Connection

Byte order

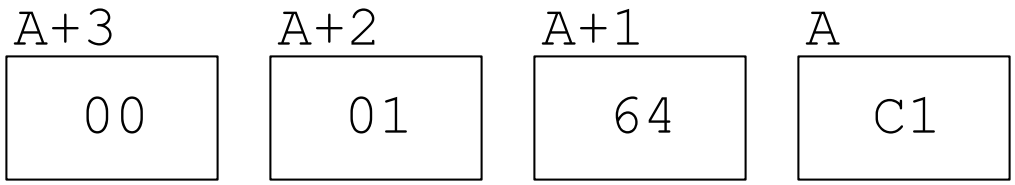
- Big-endian

91,329 =



- Little-endian

91,329 =



- Intel is little-endian, and Sparc is big-endian

Network byte order

- To communicate between machines with unknown or different “endian-ness” we convert numbers to network byte order (big-endian) before we send them.
- There are functions provided to do this:
  - `unsigned long htonl(unsigned long)`
  - `unsigned short htons(unsigned short)`
  - `unsigned long ntohl(unsigned long)`
  - `unsigned short ntohs(unsigned short)`

Network Newline

- `\r\n` rather than just `\n`

Helpful Tips

- Think carefully about the exact bytes you are sending and how the receiver will interpret them
- Pay attention to ends of strings and extra characters in char arrays
- Byte order & network newlines
- Don’t assume full lines will arrive in a single read (practice in this week’s lab)