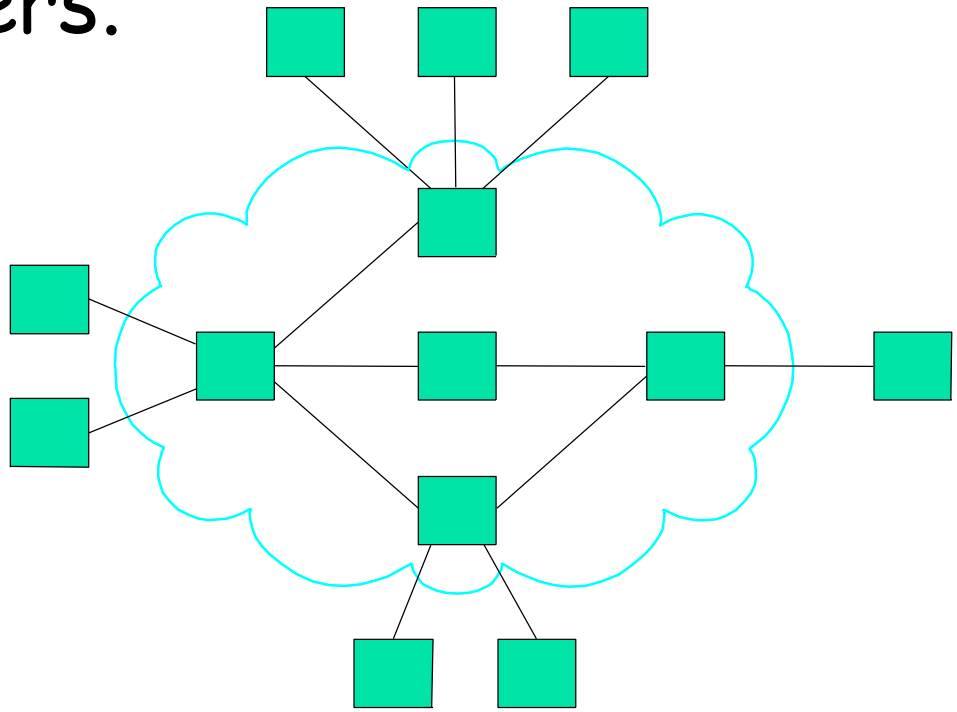
UNIX Sockets

Computer Network

- *A **computer network*** is an interconnected collection of autonomous computers.



What a Network Includes

- A network includes:
  - Special purpose ***hardware*** devices that:
    - Interconnect transmission media
    - Control transmission of data
    - Run protocol software
  - Protocol ***software*** that:
    - Encodes and formats data
    - Detects and corrects problems encountered during transmission

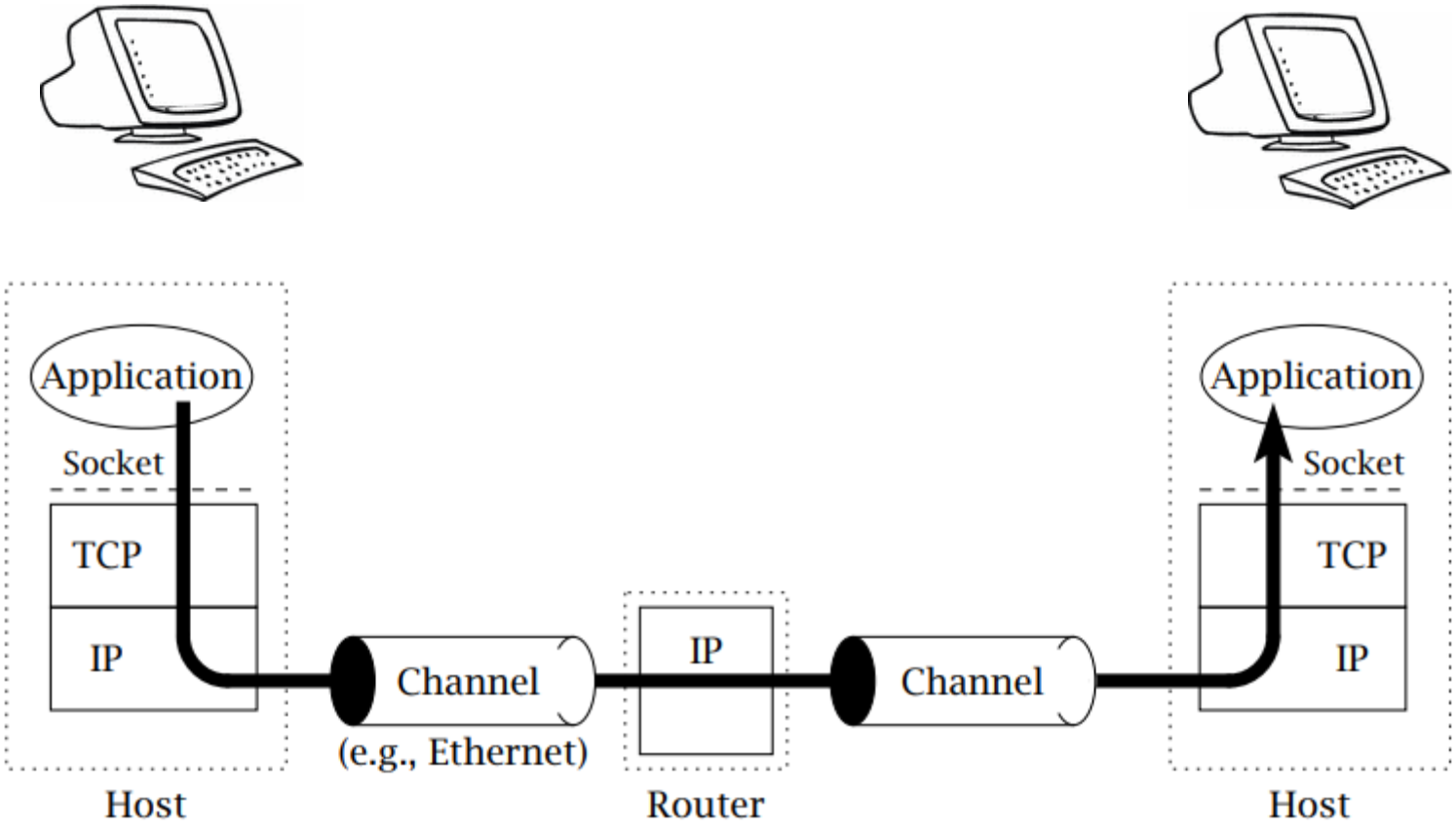
Addressing and Routing

- ***Address***: byte-string that identifies a node
  - usually unique
- ***Routing***: process of forwarding messages to the destination node based on its address
- Types of addresses
  - unicast: message sent to one station on the network. It is node-specific
  - broadcast: messages sent to all nodes/stations on the network
  - multicast: messages are sent to a group of stations. Some subset of nodes on the network

Basic Paradigm for Communication

- Most network applications can be divided into two pieces: a **client** and a **server**.
- A Web browser (a client) communicate with a Web server.
- A Telnet client that we use to log in to a remote host.
- A user who needs access to data located at remote server.

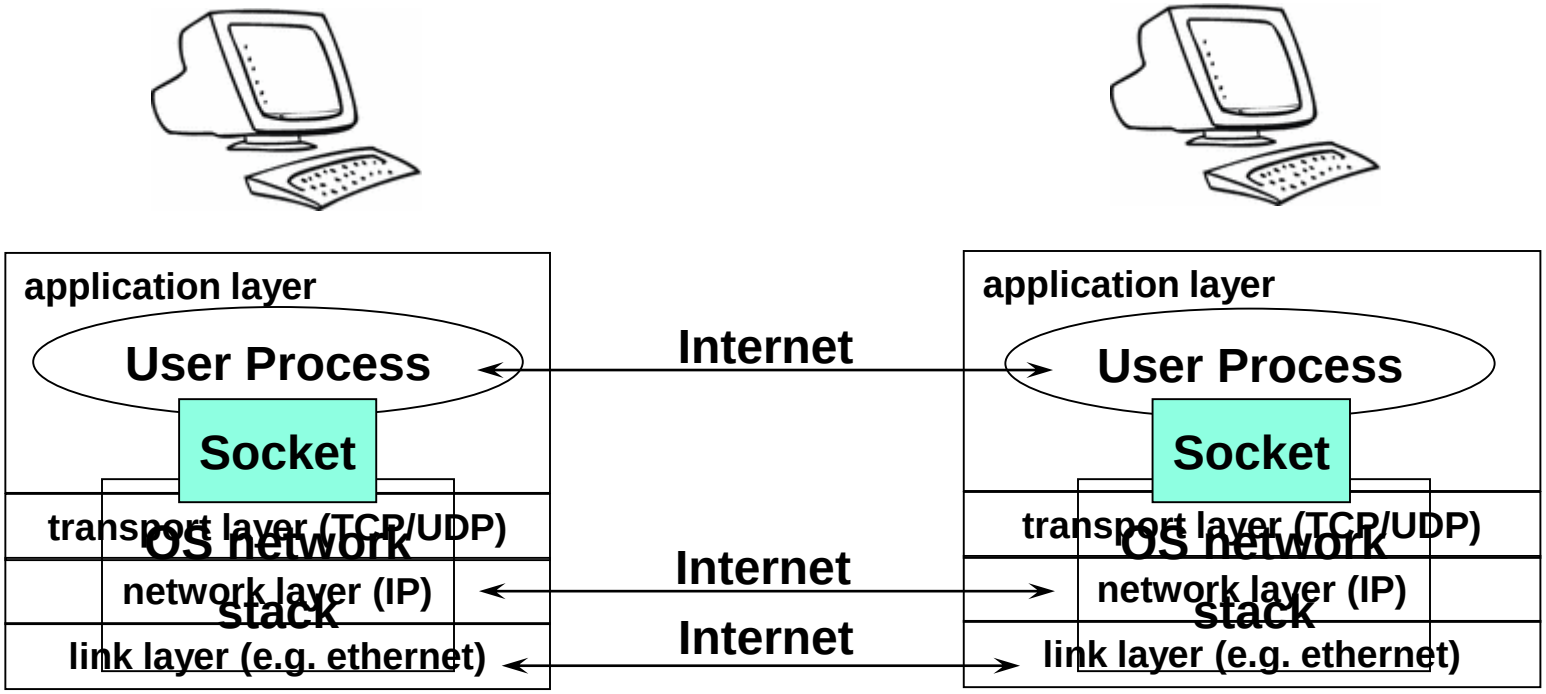
A TCP/IP Network



Sockets

- A socket is an abstraction through which an application may send and receive data, in much the same way as an open-file handle allows an application to read and write data to a stable storage.
- Applications plugged in to the same network can communicate with each other.
- Information written to the socket by an application on one machine can be read by an application on a different machine and vice versa.

Socket and Process Communication



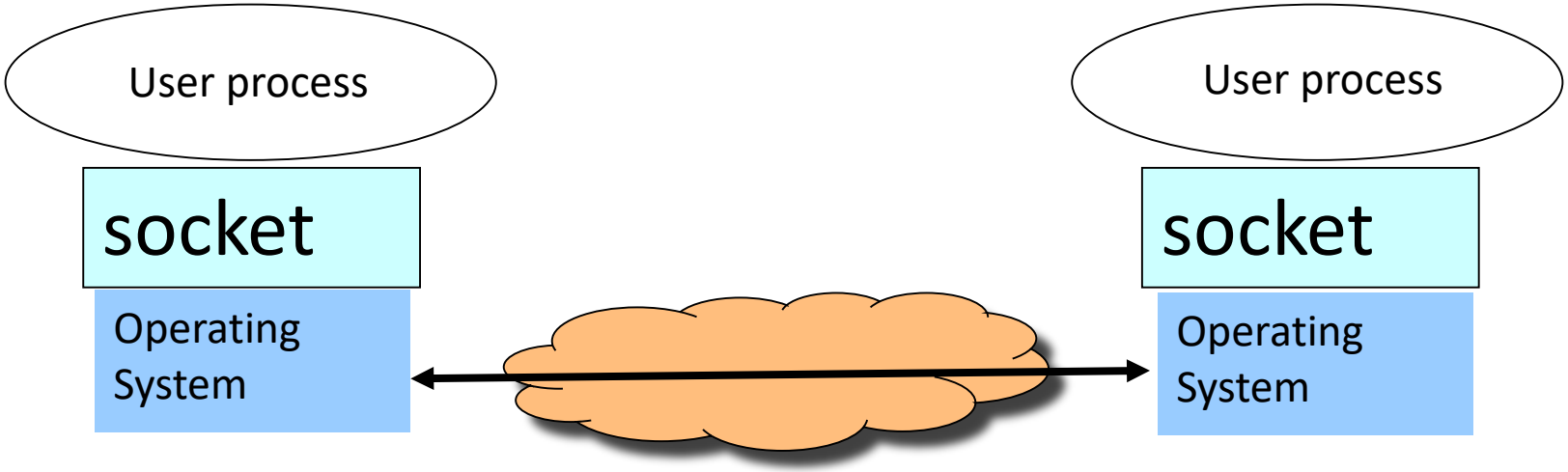
The interface that the OS provides to its networking subsystem

Delivering the Data: Division of Labor

- Network
  - Deliver data packet to the destination host
  - Based on the destination IP address
- Operating system
  - Deliver data to the destination socket
  - Based on the destination port number (e.g., 80)
- Application
  - Read data from and write data to the socket
  - Interpret the data (e.g., render a Web page)

Socket: End Point of Communication

- Sending messages from one process to another
  - Messages must traverse the underlying network
- Process sends and receives through a "socket"
  - In essence, the doorway leading in/out of the house
- Socket as an Application Programming Interface
  - Supports the creation of network applications



Two Types of Application Processes Communication

- Datagram Socket (UDP)
  - Collection of messages
  - Best effort
  - Connectionless
- Stream Socket (TCP)
  - Stream of bytes
  - Reliable
  - Connection-oriented

User Datagram Protocol (UDP): Datagram Socket

UDP

- Single socket to receive messages
- No guarantee of delivery
- Not necessarily in-order delivery
- Datagram – independent packets
- Must address each packet

Postal Mail

- Single mailbox to receive letters
- Unreliable
- Not necessarily in-order delivery
- Letters sent independently
- Must address each mail

Example UDP applications
Multimedia, voice over IP (Skype)

Transmission Control Protocol (TCP): Stream Socket

TCP

- Reliable – guarantee delivery
- Byte stream – in-order delivery
- Connection-oriented – single socket per connection
- Setup connection followed by data transfer

Telephone Call

- Guaranteed delivery
- In-order delivery
- Connection-oriented
- Setup connection followed by conversation

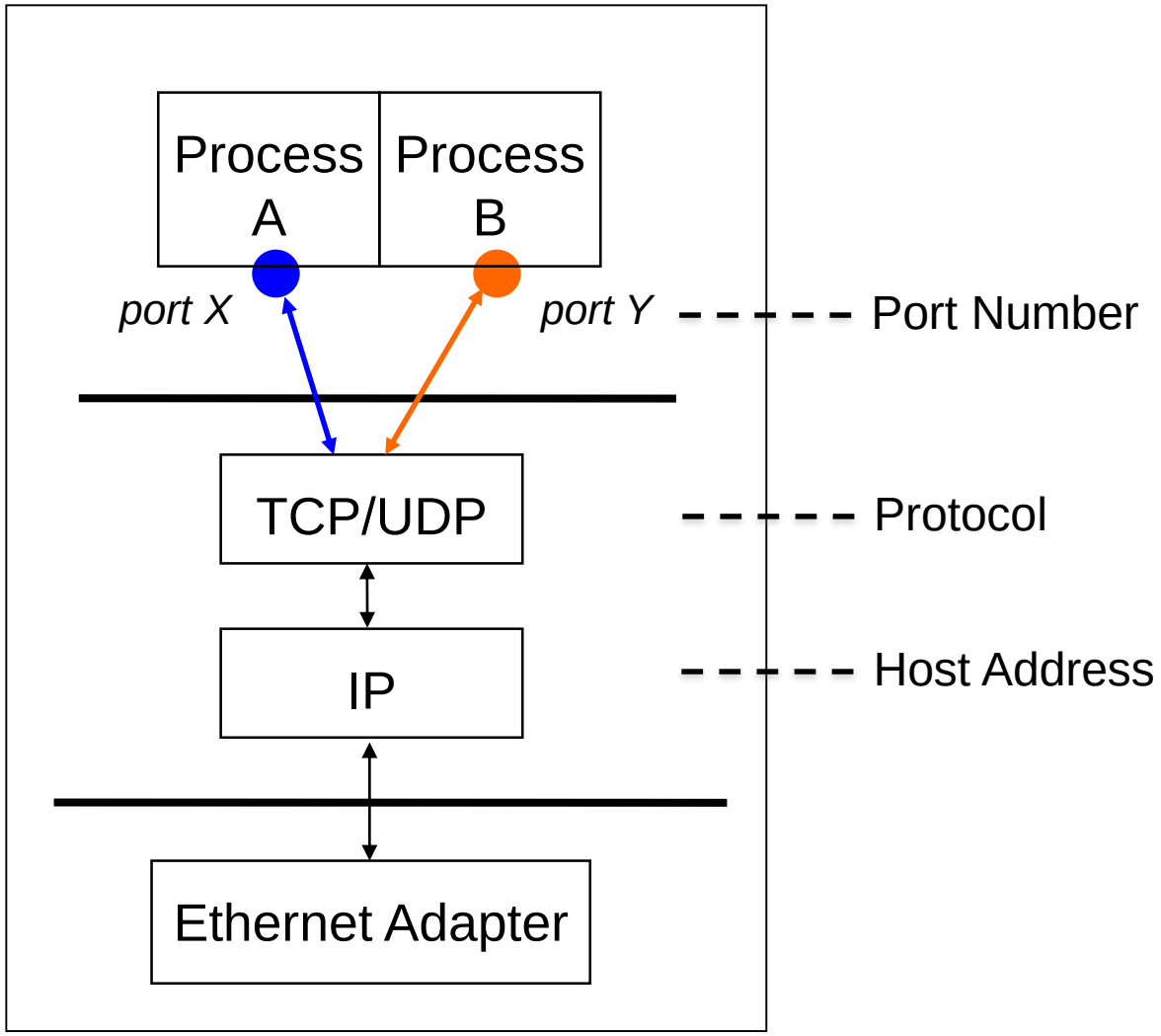
Example TCP applications

Web, Email, Telnet

Socket Identification

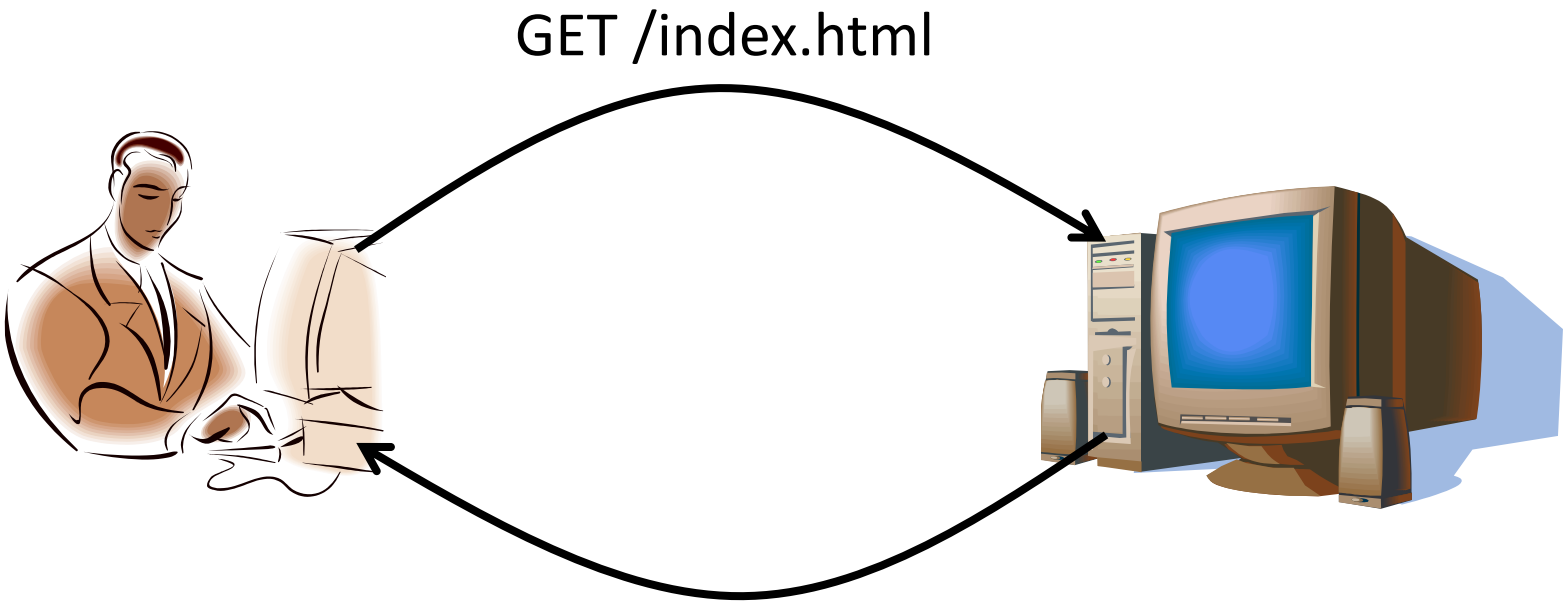
- Communication Protocol
  - TCP (Stream Socket): streaming, reliable
  - UDP (Datagram Socket): packets, best effort
- Receiving host
  - Destination **address** that uniquely identifies the host
  - An **IP** (IPv4) **address** is a 32-bit quantity. IPv6 is 128 bits.
- Receiving socket
  - Host may be running many different processes
  - Destination **port** that uniquely identifies the socket
  - A **port number** is a 16-bit quantity

Socket Identification (Cont.)



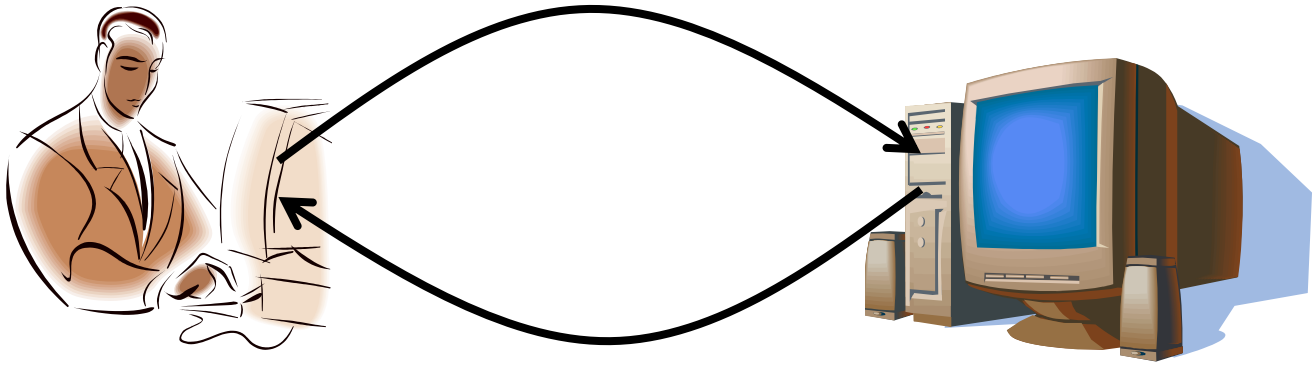
Clients and Servers

- Client program
  - Running on end host
  - Requests service
  - E.g., Web browser
- Server program
  - Running on end host
  - Provides service
  - E.g., Web server



Client-Server Communication

- Client "sometimes on"
  - Initiates a request to the server when interested
  - E.g., Web browser on your laptop or cell phone
  - Doesn't communicate directly with other clients
  - Needs to know server's address
- Server is "always on"
  - Handles services requests from many client hosts
  - E.g., Web server for the www.cnn.com Web site
  - Doesn't initiate contact with the clients
  - Needs fixed, known address



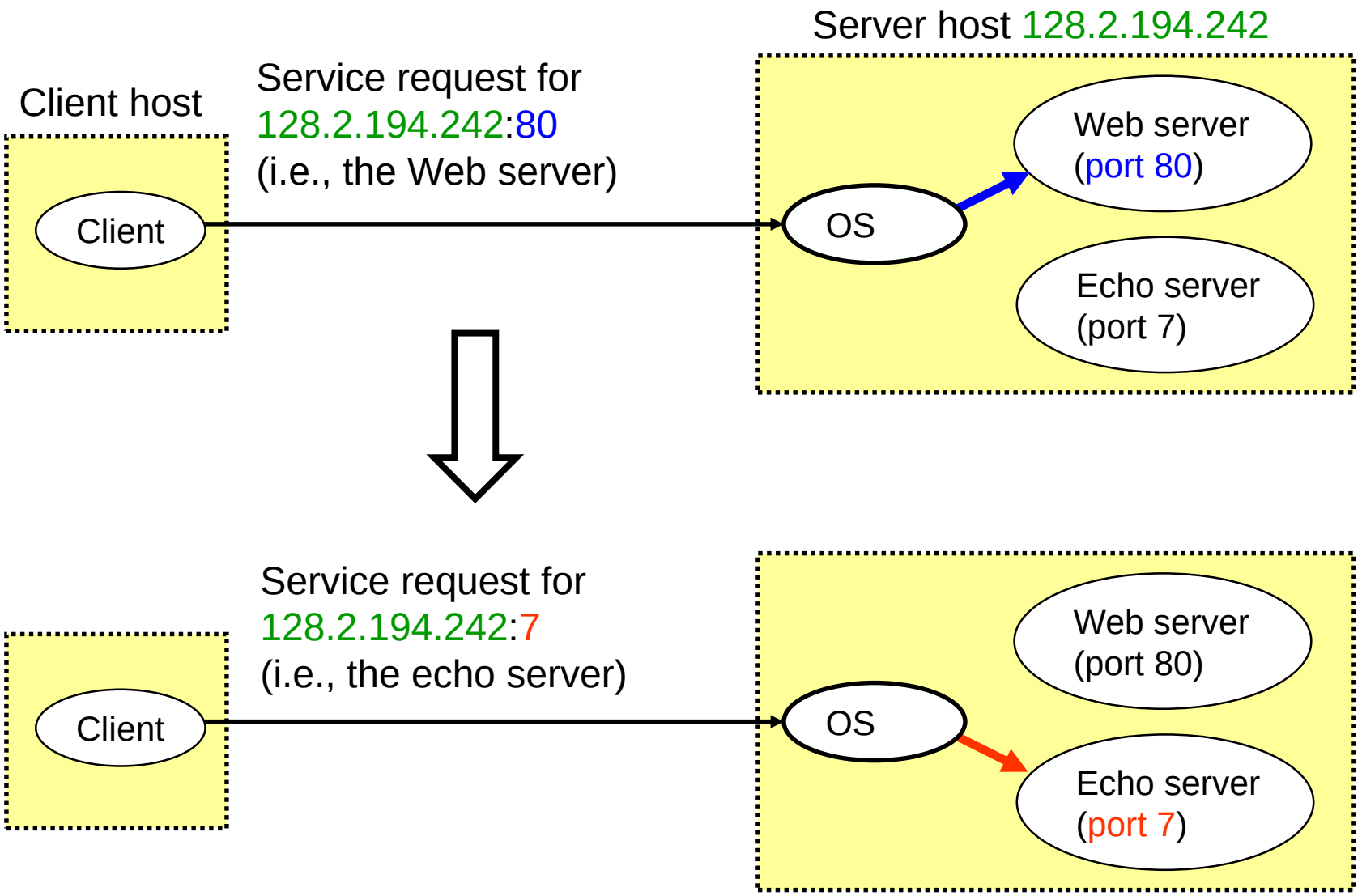
Client and Server Processes

- Client process
  - process that initiates communication
- Server Process
  - process that waits to be contacted

Knowing What Port Number To Use

- Popular applications have well-known ports
  - E.g., port 80 for Web and port 25 for e-mail
  - See http://www.iana.org/assignments/port-numbers
- Well-known vs. ephemeral ports
  - Server has a well-known port (e.g., port 80)
    - Between 0 and 1023 (requires root to use)
  - Client picks an unused ephemeral (i.e., temporary) port
    - Between 1024 and 65535
- Uniquely identifying traffic between the hosts
  - Two IP addresses and two port numbers
  - Underlying transport protocol (e.g., TCP or UDP)

Using Ports to Identify Services



Basic Paradigm for Communication

- Establish contact (connection).
- Exchange information (bi-directional).
- Terminate contact.

Client-Server Paradigm

- Server waits for client to request a connection.
- Client contacts server to establish a connection.
- Client sends request.
- Server sends reply.
- Client and/or server terminate connection.

UNIX TCP Communication

- The server uses the **accept** system call to accept connection requested by a client.
- After a connection has been established, both processes may then use the **write** (**send**) and **read** (**recv**) operations to send and receive messages.

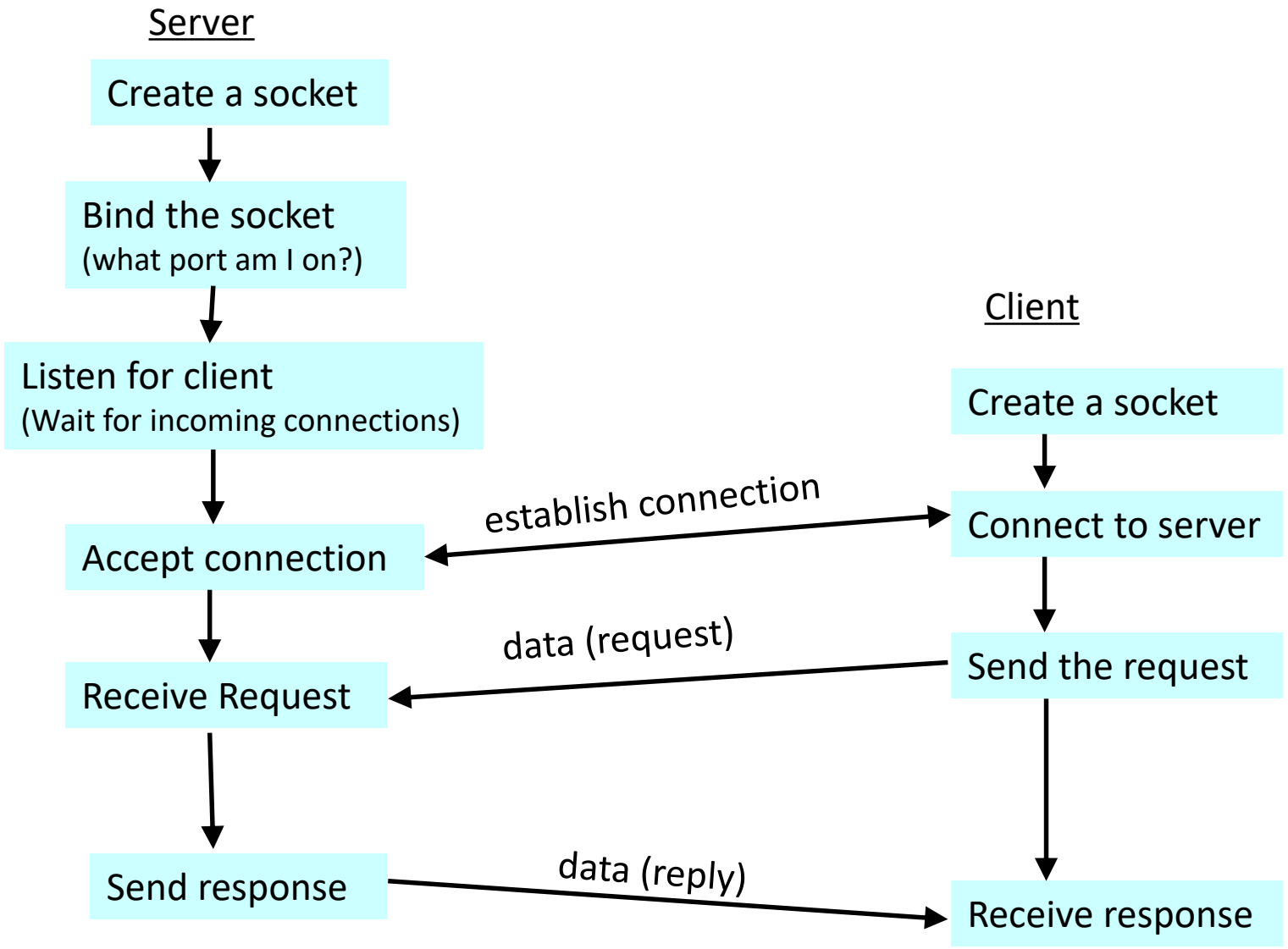
Example - Programming Client

- Initialization:
  - **gethostbyname** - look up server
  - **socket** - create socket
  - **connect** - connect to server port
- Transmission:
  - **send** – send message to server
  - **recv** - receive message from server
- Termination:
  - **close** - close socket

Example - Programming Server

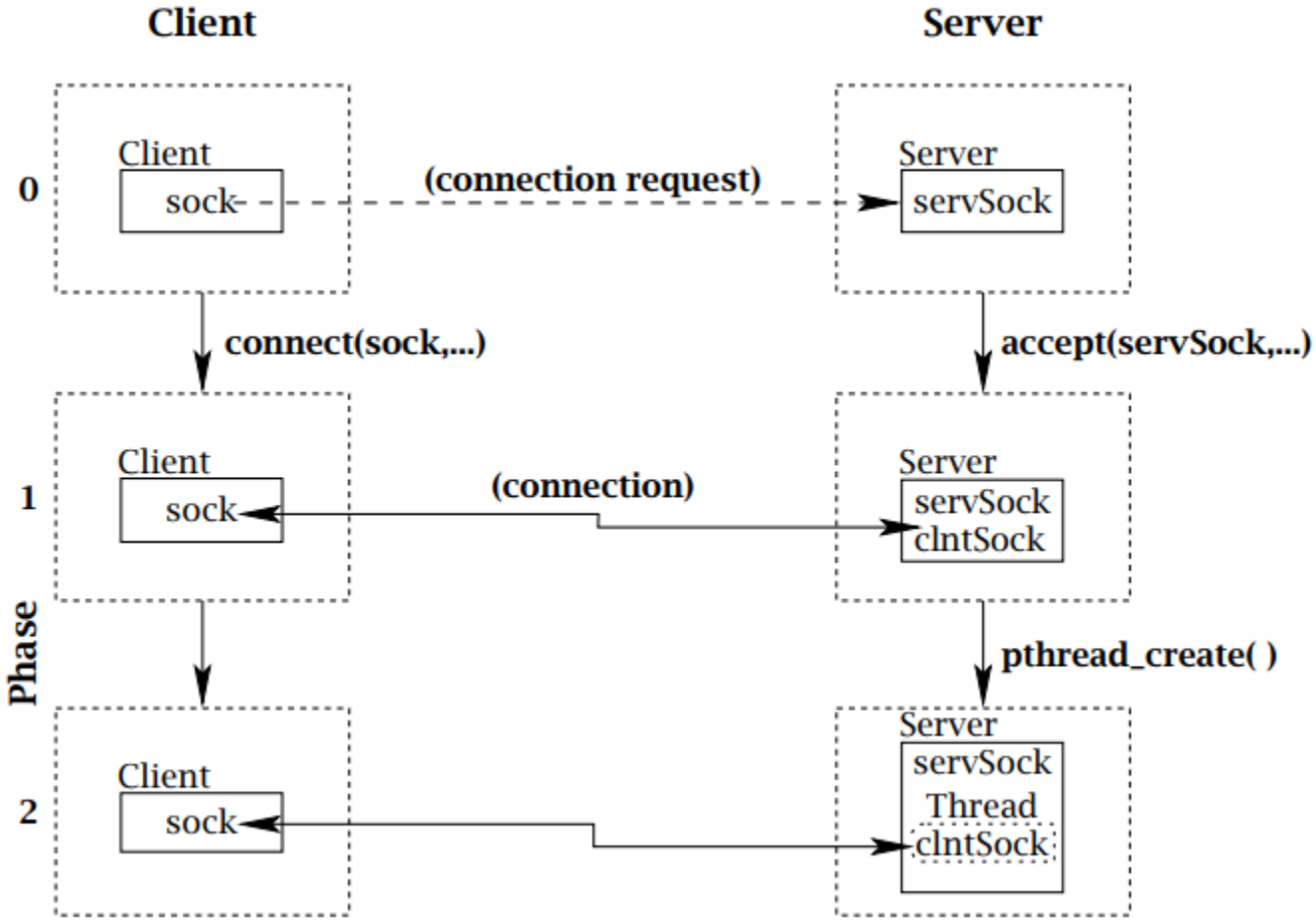
- Initialization:
  - **socket** - create socket
  - **bind** – bind socket to the local address
  - **listen** - associate socket with incoming requests
- Loop:
  - **accept** - accept incoming connection
  - **recv** - receive message from client
  - **send** - send message to client
- Termination:
  - **close** - close connection socket

Client-Server Communication Stream Sockets (TCP): Connection-oriented



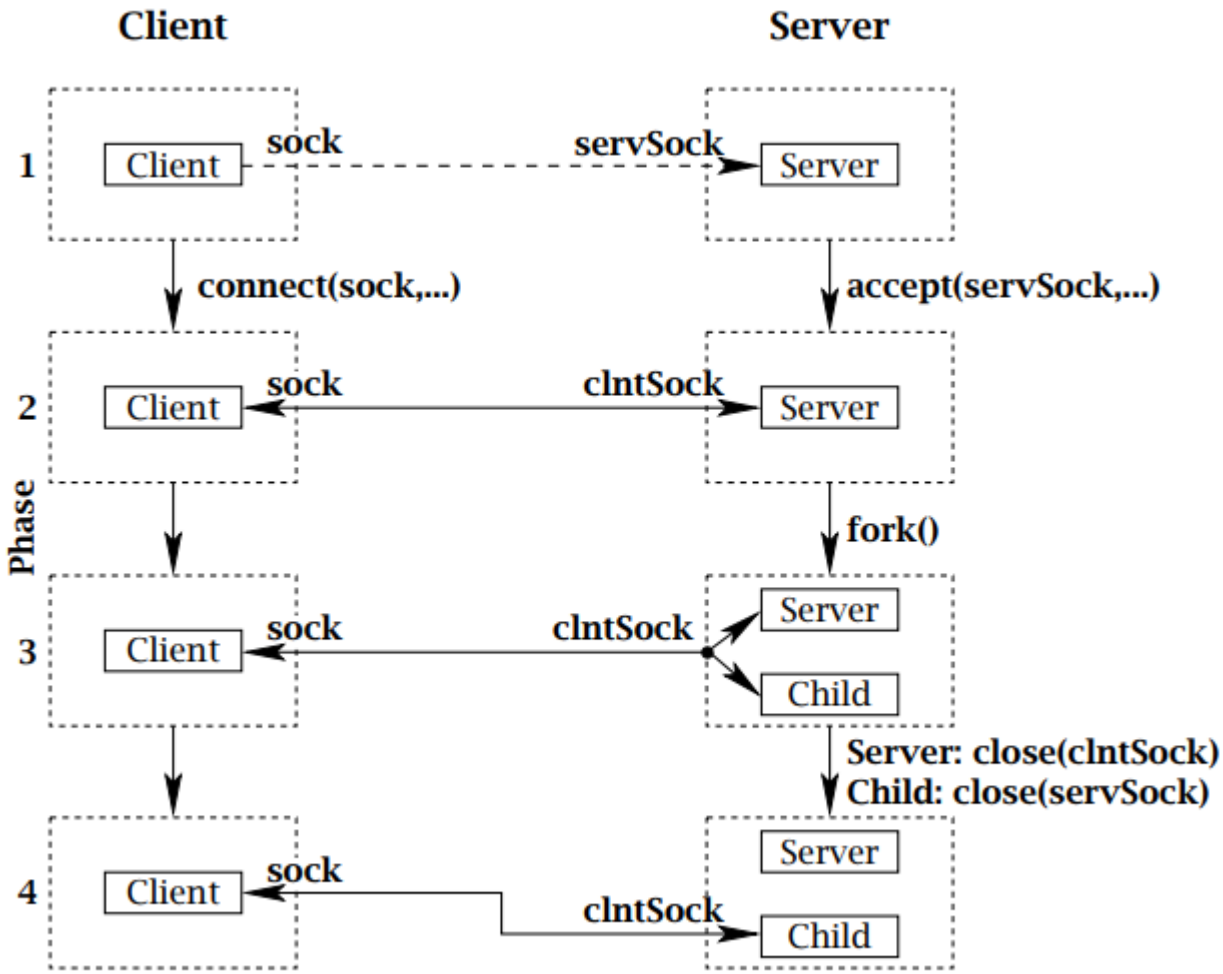
Threaded TCP echo server

int clntSock; // Socket descriptor for client
int servSock; // Socket descriptor for server

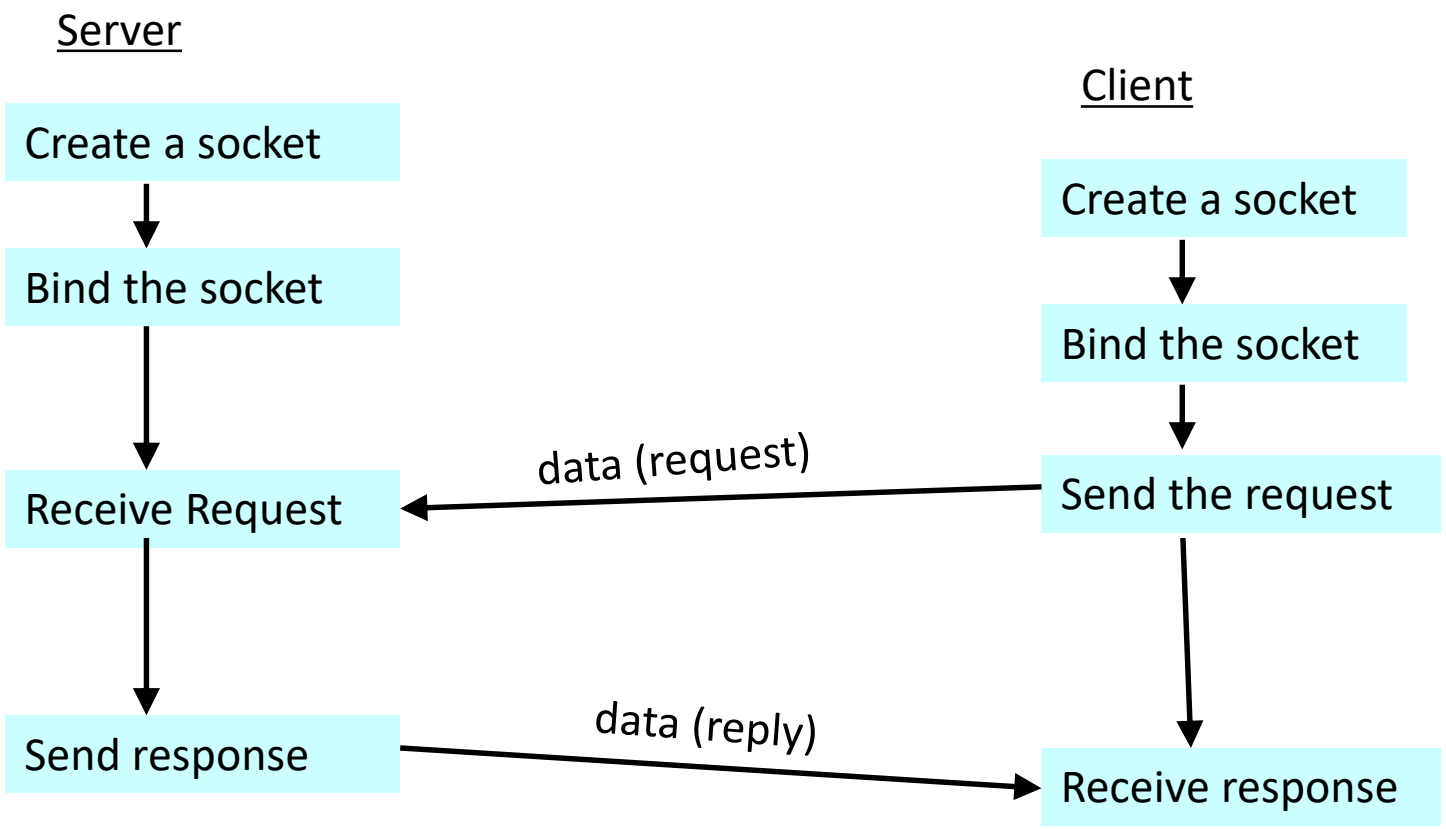


Forking a TCP echo server

int clntSock; // Socket descriptor for client
int servSock; // Socket descriptor for server



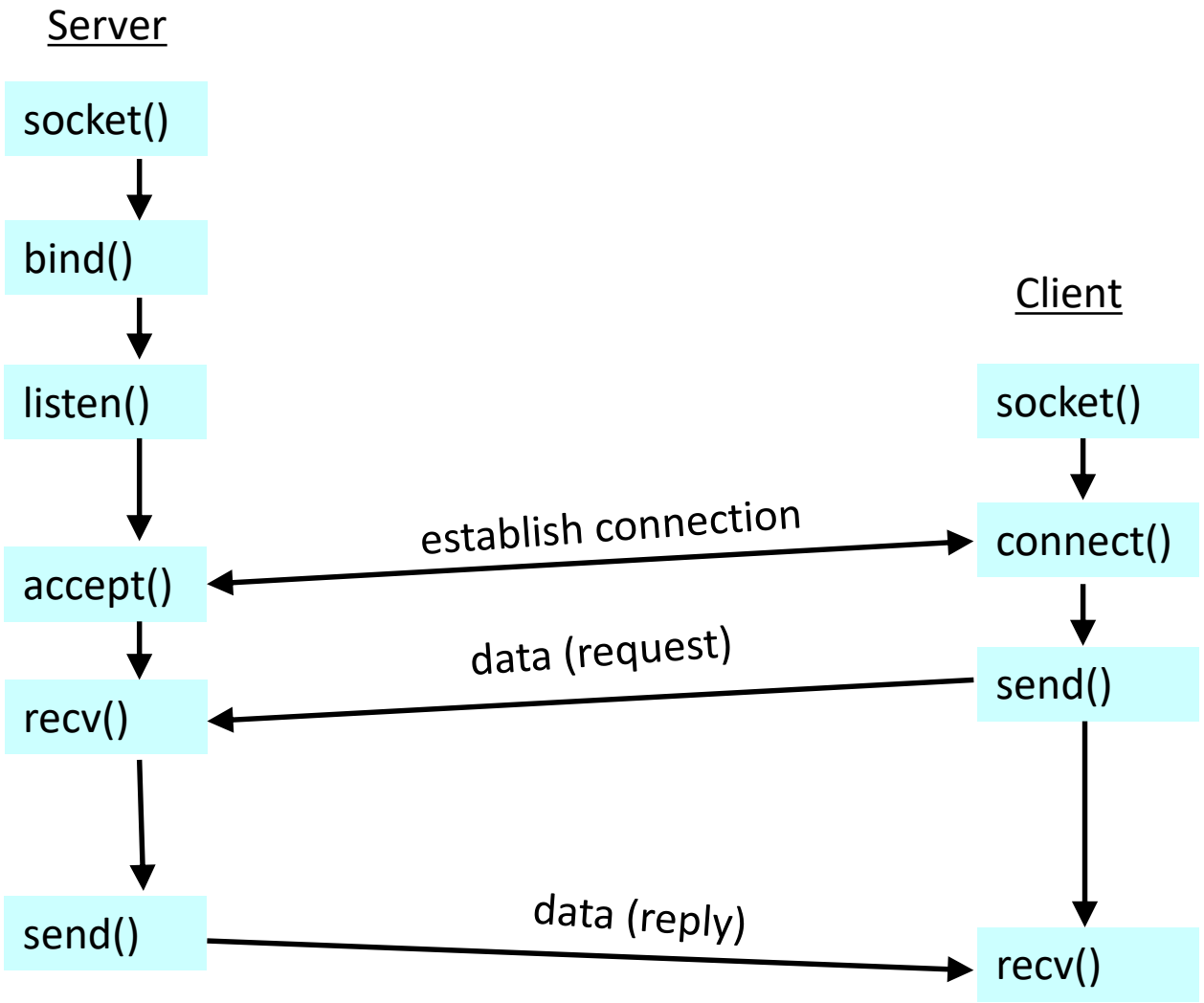
Client-Server Communication Datagram Sockets (UDP): Connectionless



UNIX Socket API

- Socket interface
  - Originally provided in Berkeley UNIX
  - Later adopted by all popular operating systems
  - Simplifies porting applications to different OSes
- In UNIX, everything is like a file
  - All input is like reading a file
  - All output is like writing a file
  - File is represented by an integer file descriptor
- API implemented as system calls
  - E.g., connect, send, recv, close, …

Connection-oriented Example (Stream Sockets -TCP)



Connectionless Example (Datagram Sockets - UDP)