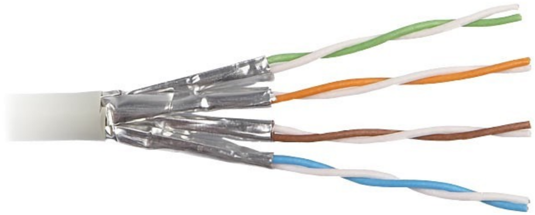
Lecture 25: Networking

CS 105 Fall 2025

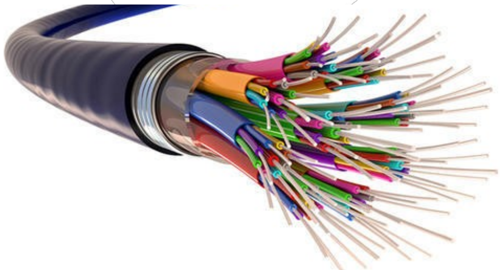
Physical Layer

- Twisted Pair



- Coaxial

- Fiber

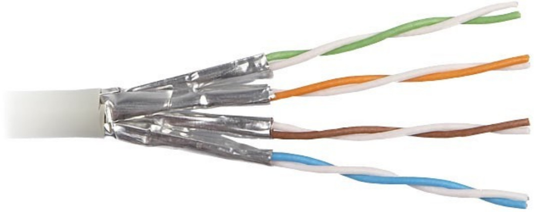


- Radio

Data Link Layer

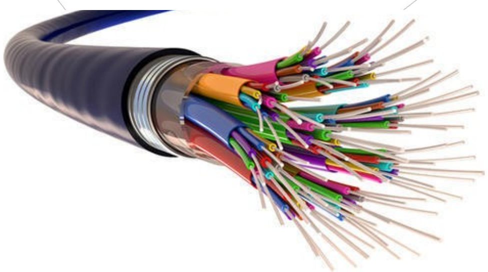
- DSL
- Ethernet
- WiFi (802.11)
- 3G
- LTE
- 5G

Twisted Pair

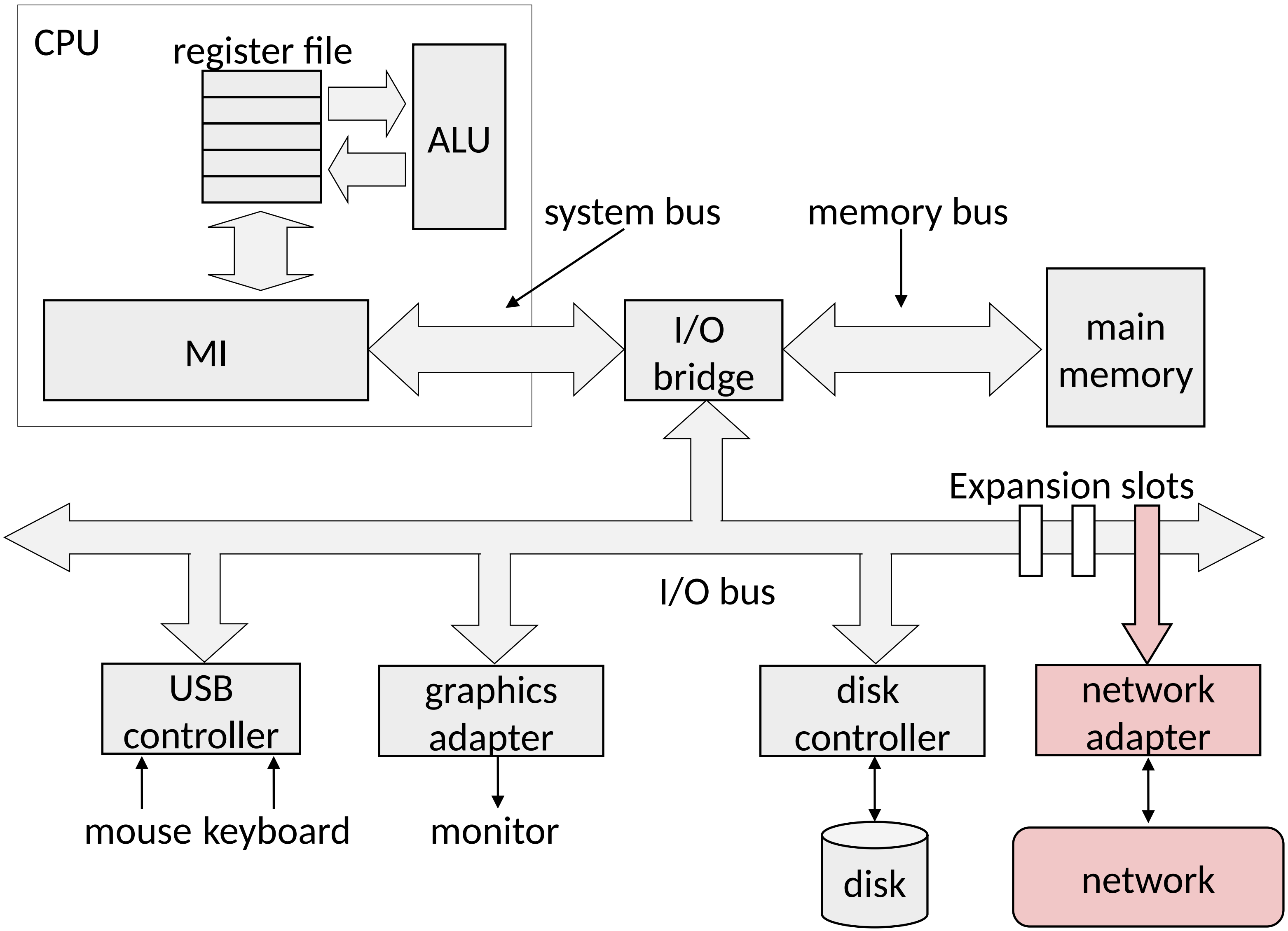


Coaxial

Fiber



Radio


CPU
register file
ALU
MI
system bus
memory bus
I/O bridge
main memory
Expansion slots
I/O bus
USB controller
graphics adapter
disk controller
network adapter
mouse keyboard
monitor
disk
network

Data Link Layer

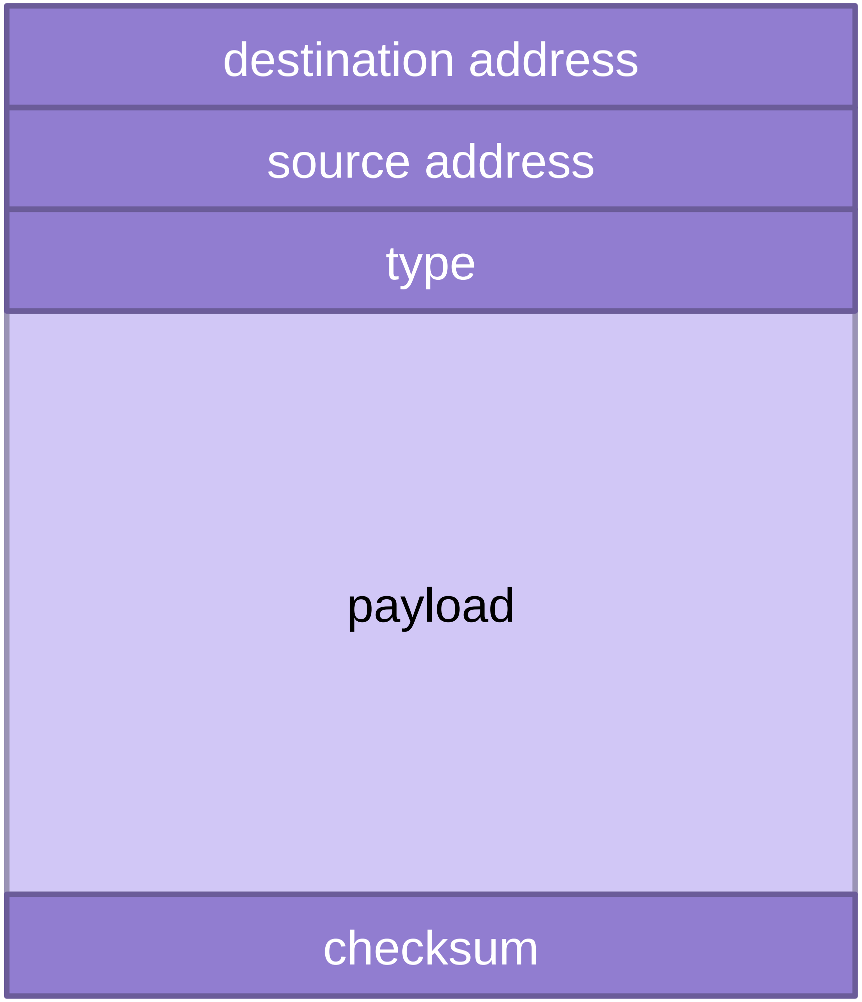
- Each host has one or more network adapter (aka NIC)
  - handles particular physical layer and protocol
- Each network adapter has a media access control (MAC) address
  - unique to that network instance
- Messages are organized as packets

Example: Ethernet

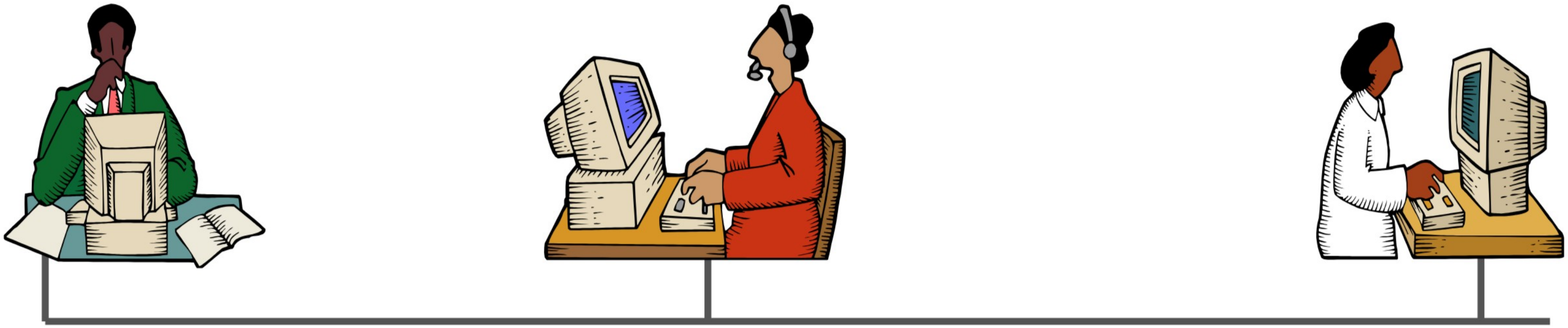
- Developed 1976 at Xerox
- Simple, scales pretty well
- Very successful, still in widespread use

- Example address: b8:e3:56:15:6a:72

- **Carrier sense:** listen before you speak
- **Multiple access:** multiple hosts on network
- **Collision detection:** detect and respond to cases where two messages collide



Example: Ethernet



- Carrier sense: broadcast if wire is available
- In case of collision: stop, sleep, retry
  - sleep time is determined by collision number
  - abort after 16 attempts

Example: Ethernet

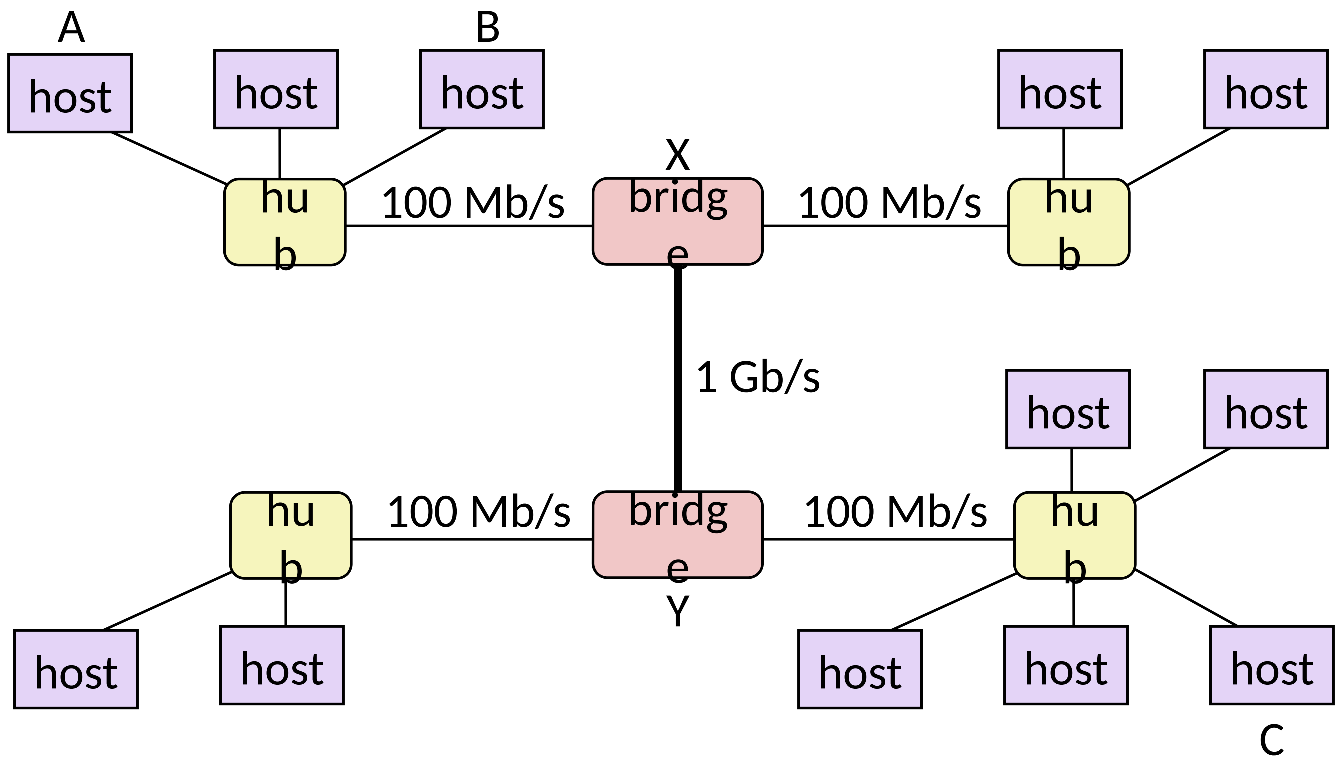
Advantages

- completely decentralized
- inexpensive
  - no state in the network
  - no arbiter
  - cheap physical links

Disadvantages

- data is available for all to see
  - can place ethernet card in promiscuous mode and listen to all messages
- endpoints must be trusted
- In large/high-traffic networks, many collisions

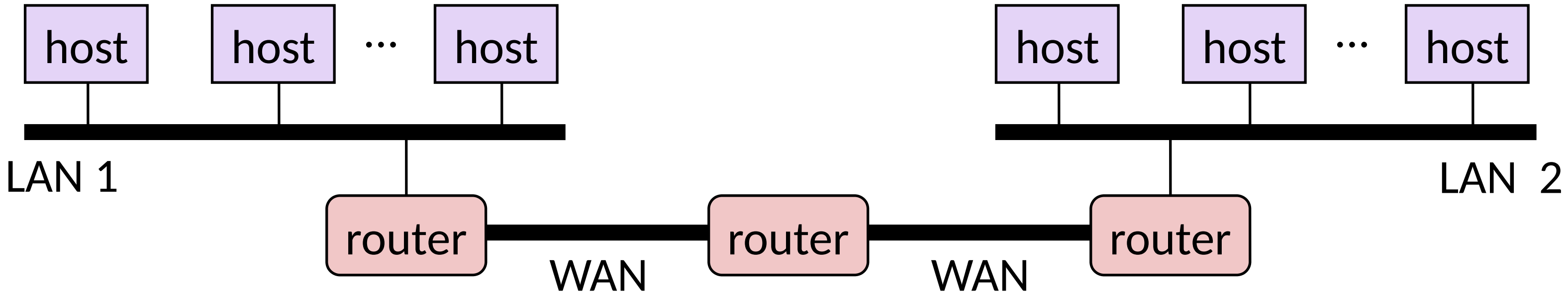
Bridged Ethernet



- Spans building or campus
- Bridges cleverly learn which hosts are reachable from which ports and then selectively copy frames from port to port

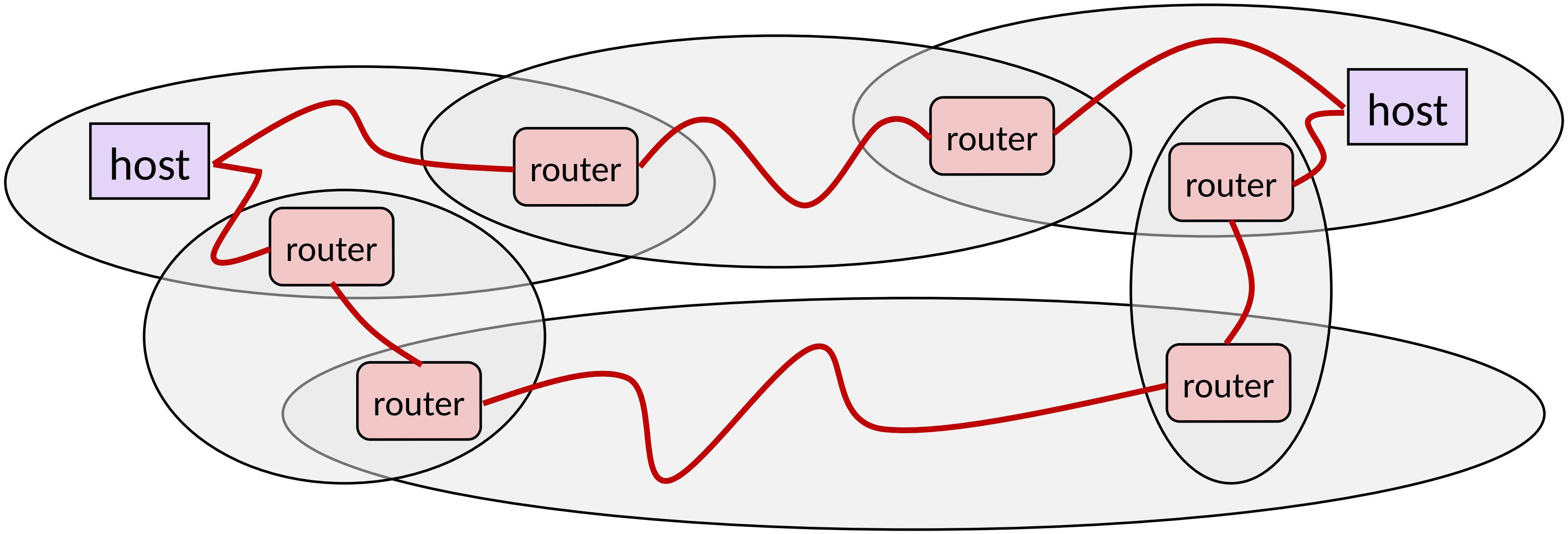
Network Layer

- There are lots of lots of local area networks (LANs)
  - each determines its own protocols, address format, packet format
- What if we wanted to connect them together?
  - physically connected by specialized computers called routers
  - routers with multiple network adapters can translate
  - standardize address and packet formats



- This is a internetwork
  - aka wide-area network (WAN)
  - aka internet

Logical Structure of an internet



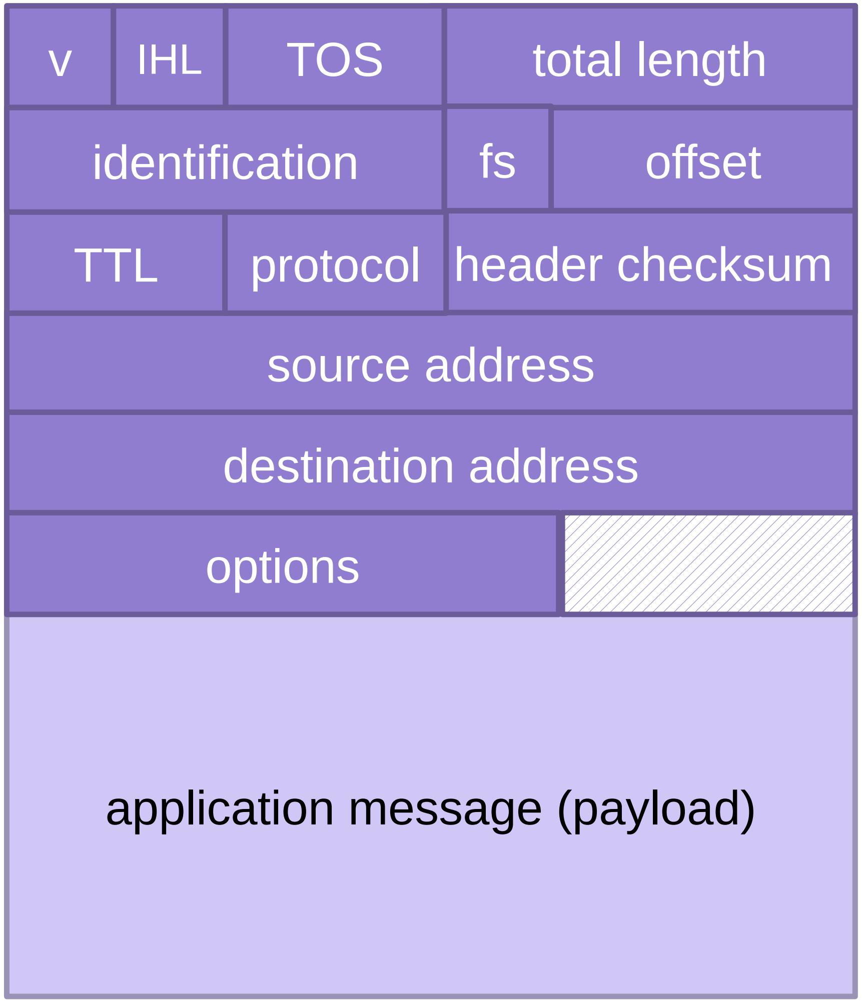
- Ad hoc interconnection of networks
  - No particular topology
  - Vastly different router & link capacities
- Send packets from source to destination by hopping through networks
  - Router forms bridge from one network to another
  - Different packets may take different routes

Internet Protocol (IP)

- Initiated by the DoD in 60s-70s
- Currently transitioning (very slowly) from IPv4 to IPv6

- Example address: 128.84.12.43

- interoperable
- network dynamically routes packets from source to destination



Aside: IPv4 and IPv6

- The original Internet Protocol, with its 32-bit addresses, is known as *Internet Protocol Version 4* (IPv4)

- 1996: Internet Engineering Task Force (IETF) introduced *Internet Protocol Version 6* (IPv6) with 128-bit addresses
  - Intended as the successor to IPv4

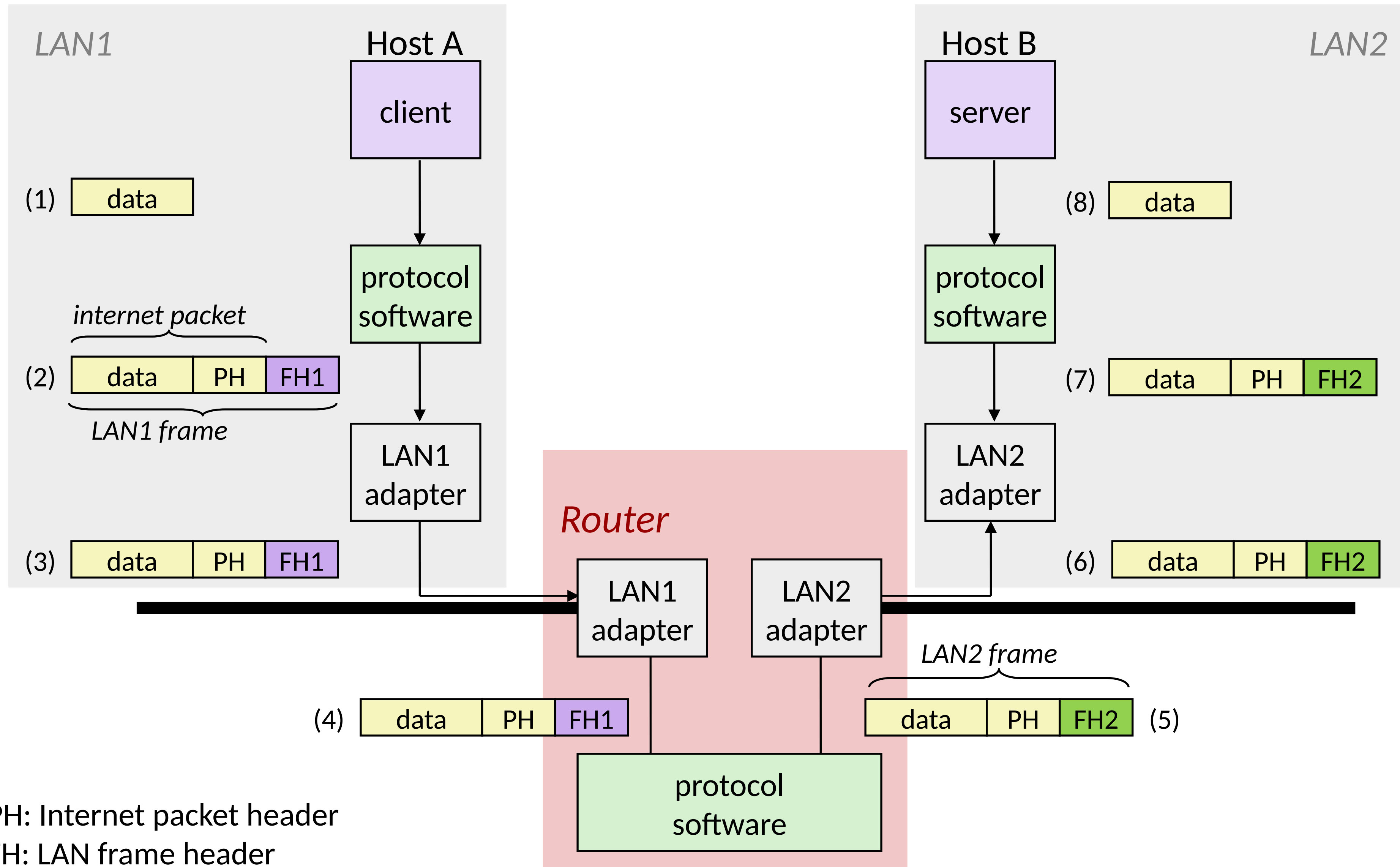
- As of April 2023, majority of Internet traffic still carried by IPv4
  - 38-44% of users access Google services using IPv6.

- We will focus on IPv4, but will show you how to write networking code that is protocol-independent.

Exercise: IP addresses

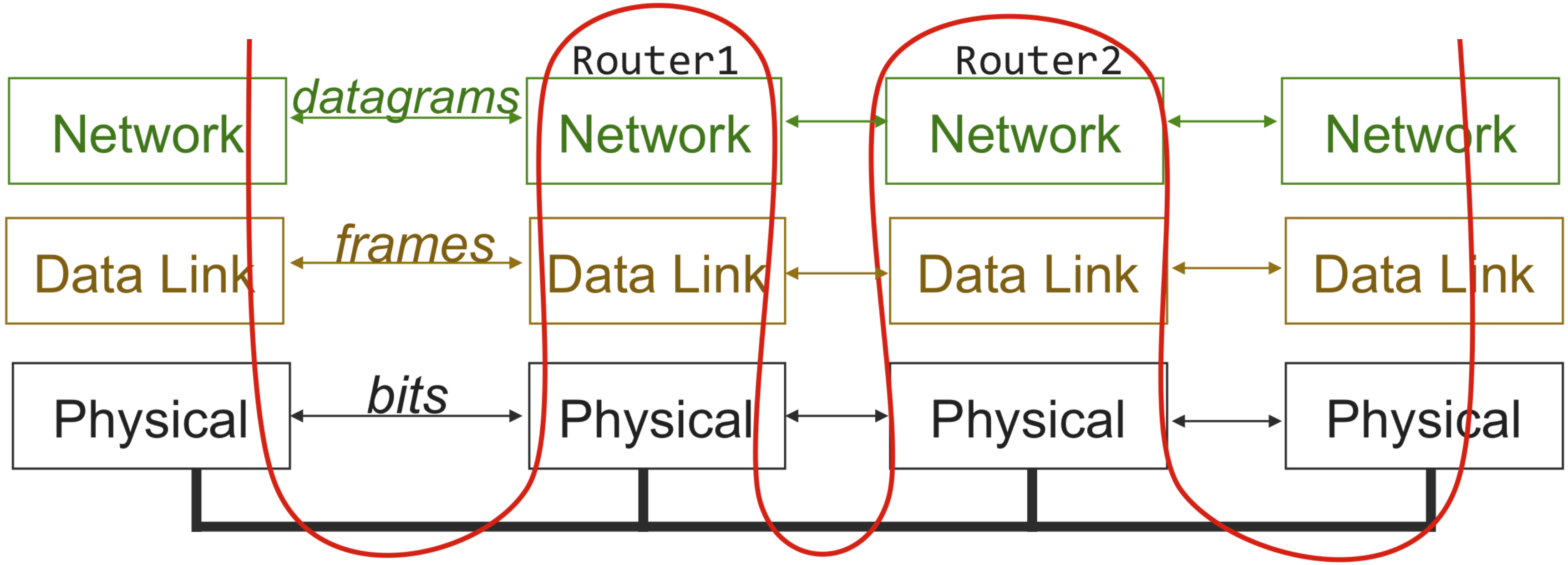
What is the current IP address assigned to your computer or phone?

Transferring internet Data Via Encapsulation

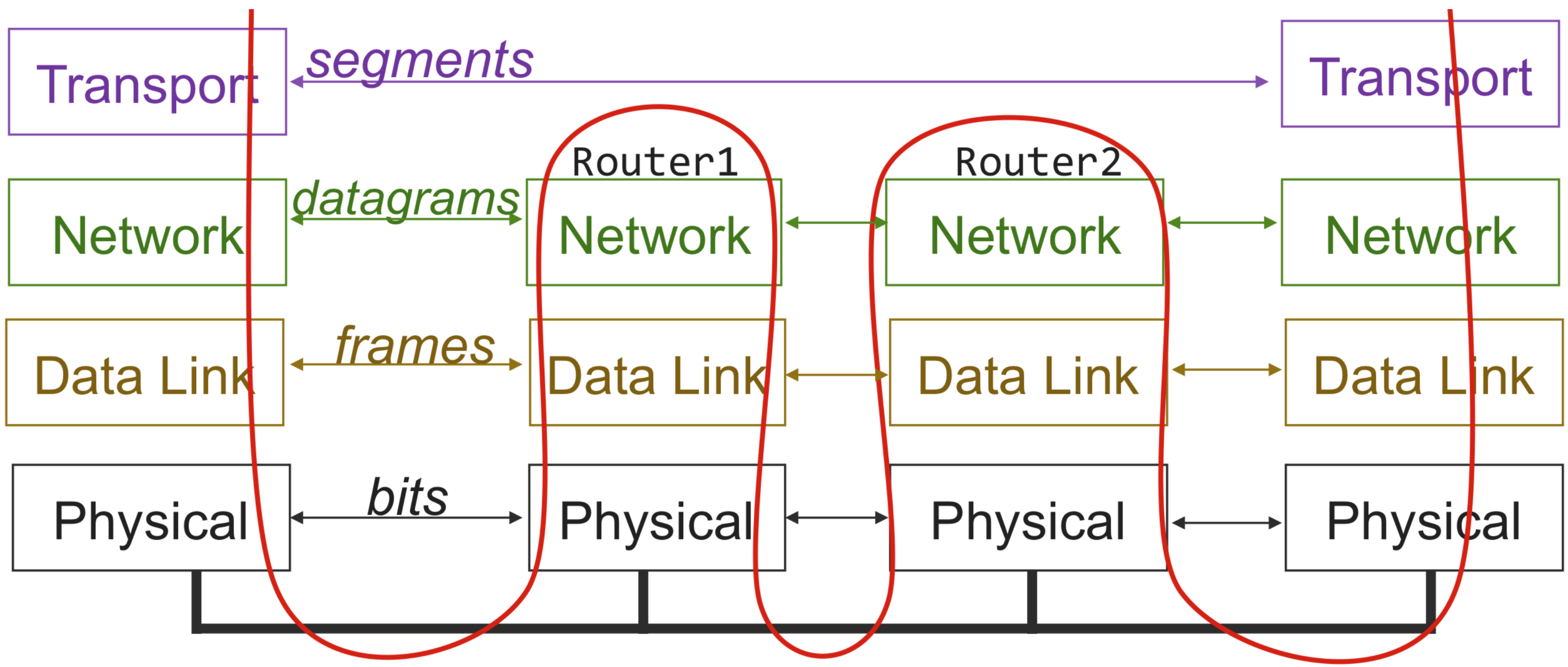


PH: Internet packet header
FH: LAN frame header

Routing


Router1
Router2
Network
datagrams
Network
Network
Network
Data Link
frames
Data Link
Data Link
Data Link
Physical
bits
Physical
Physical
Physical

Transport Layer



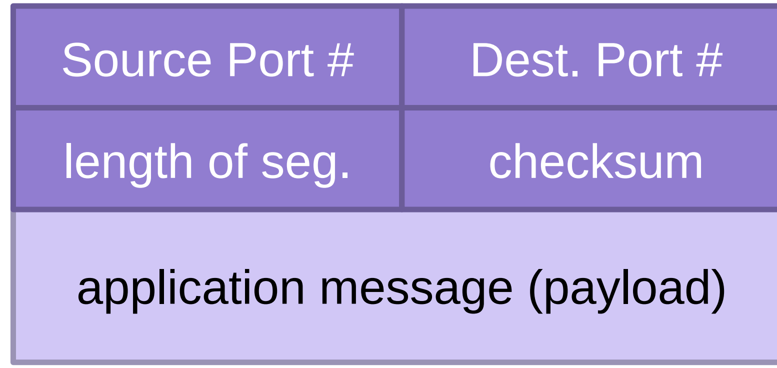
Transport Layer

- Clients and servers communicate by sending streams of bytes over a **connection**.

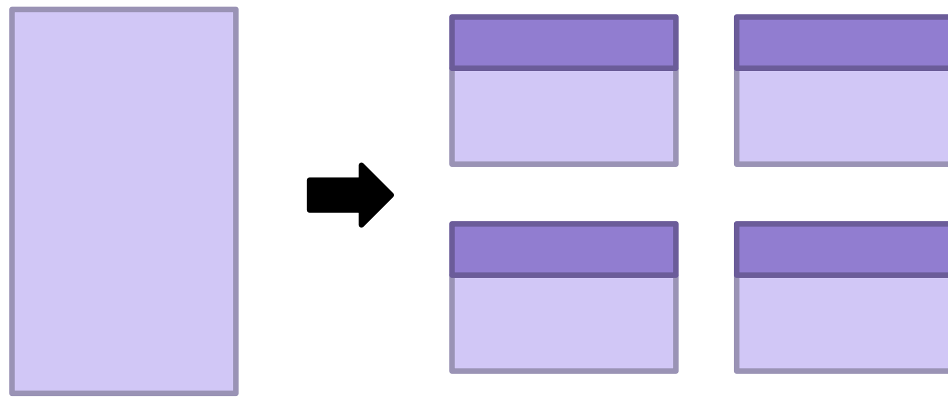
- A transport layer endpoint is identified by an **IP address** and a **port**, a 16-bit integer that identifies a process
  - Ephemeral port: Assigned automatically by client kernel when client makes a connection request.
  - Well-known port: Associated with some service provided by a server (e.g., port 80 is associated with Web servers)

Transport Layer Segments

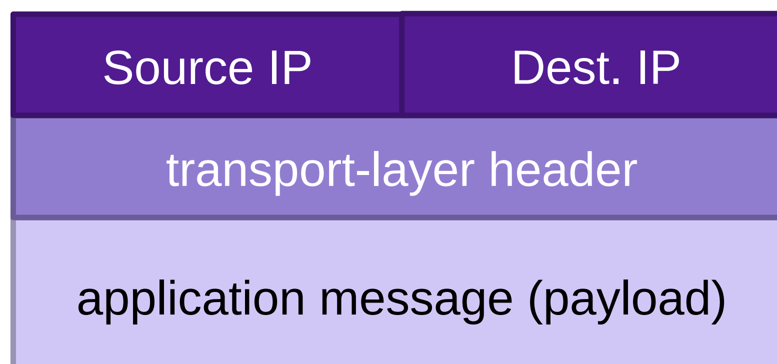
- Sending application:
  - specifies IP address and port
  - uses socket bound to source port



- Transport Layer:
  - breaks application message into smaller chunks
  - adds transport-layer header to each message to form a segment

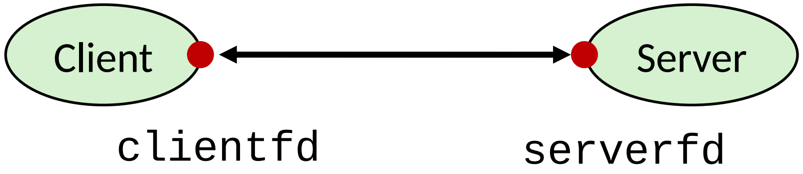


- Network Layer (IP):
  - adds network-layer header to each datagram



Sockets

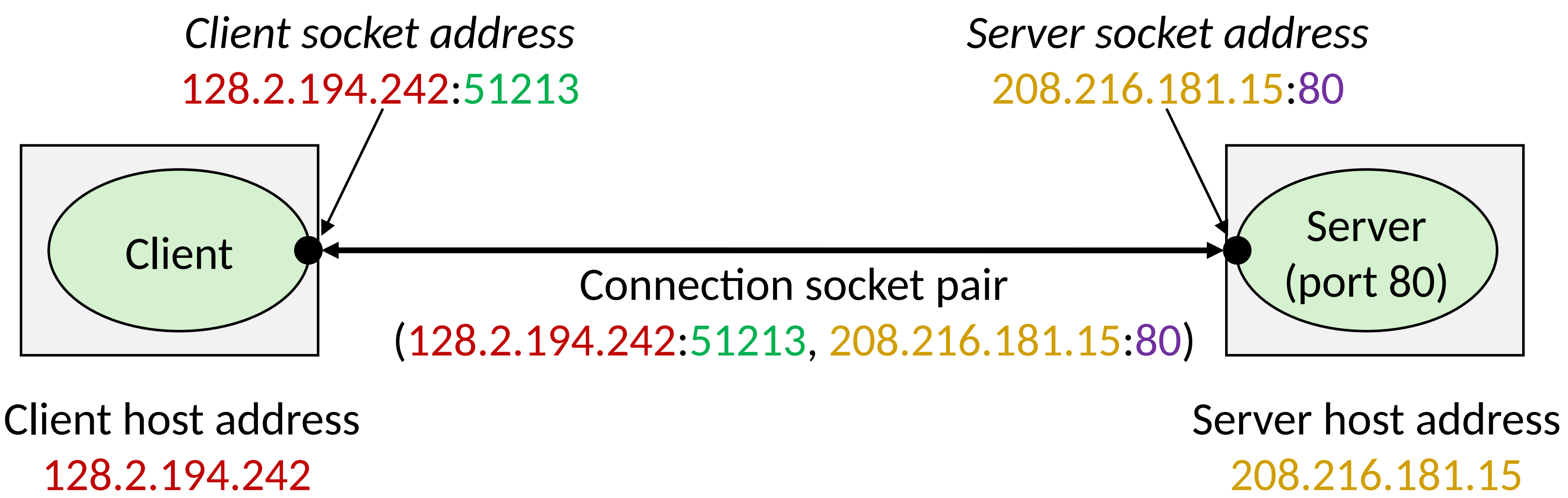
- What is a socket?
  - IP address + port
  - To the kernel, a socket is an endpoint of communication
  - To an application, a socket is a file descriptor that lets the application read/write from/to the network
    - ***Note:*** All Unix I/O devices, including networks, are modeled as files
- Clients and servers communicate with each other by reading from and writing to socket descriptors



- The main distinction between regular file I/O and socket I/O is how the application "opens" the socket descriptors

Anatomy of a Connection

- A connection is uniquely identified by the socket addresses of its endpoints (*socket pair*)
  - **`(cliaddr:cliport, servaddr:servport)`**



51213 is an ephemeral port allocated by the kernel

80 is a well-known port associated with Web servers

Well-known Ports and Service Names

- Popular services have permanently assigned **well-known ports** *and* corresponding ***well-known service names***:
  - echo server: 7/echo
  - ssh servers: 22/ssh
  - email server: 25/smtp
  - Web servers: 80/http

- Mappings between well-known ports and service names is contained in the file `/etc/services` on each Linux machine.

Should the transport layer guarantee packet delivery?

Transport Layer Protocols

User Datagram Protocol (UDP)

- **unreliable, unordered delivery**
- connectionless
- best-effort, segments might be lost, delivered out-of-order, duplicated
- reliability (if required) is the responsibility of the app

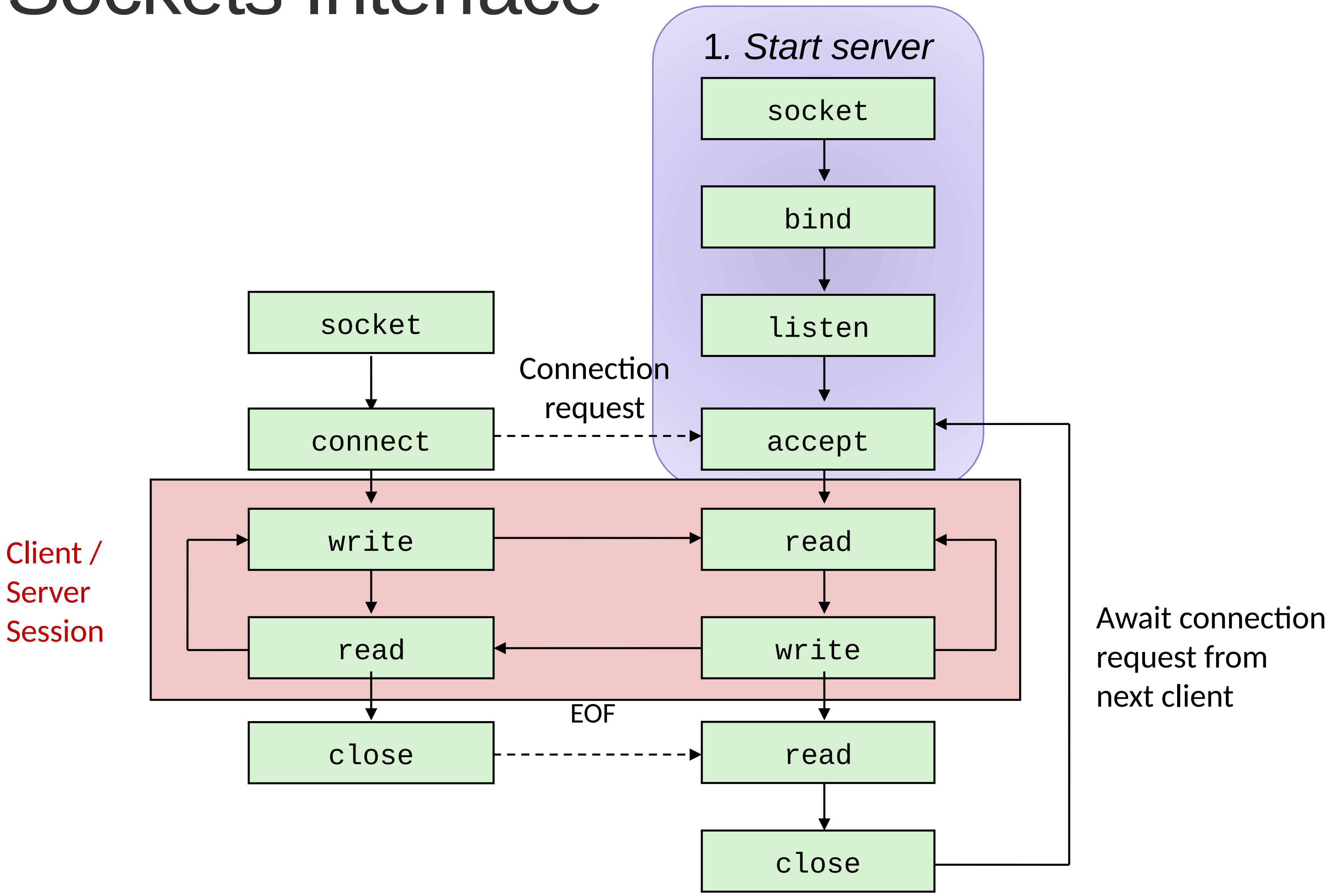
Transmission Control Protocol (TCP)

- **reliable, in-order delivery**
- connection setup
- flow control
- congestion control

Transport Protocols by Application

Application	Application-Level Protocol	Transport Protocol
Name Translation	DNS	Typically UDP
Routing Protocol	RIP	Typically UDP
Network Management	SNMP	Typically UDP
Remote File Server	NFS	Typically UDP
Streaming multimedia	(proprietary)	UDP or TCP
Internet telephony	(proprietary)	UDP or TCP
Remote terminal access	Telnet	TCP
File Transfer	(S)FTP	TCP
Email	SMTP	TCP
Web	HTTP(S)	TCP

Sockets Interface



Sockets Interface: `socket`

- Clients and servers use the `socket` function to create a *socket descriptor*:

```
int socket(int domain, int type, int protocol)
```

- Example:

```
int clientfd = socket(AF_INET, SOCK_STREAM, 0);
```

Indicates that we are using 32-bit IPV4 addresses

Indicates that the socket will be the end point of a TCP connection

Protocol specific! Best practice is to use `getaddrinfo` to generate the parameters automatically, so that code is protocol independent.

Sockets Interface: `bind`

- A server uses `bind` to ask the kernel to associate the server's socket address with a socket descriptor:

```
int bind(int sockfd, SA* addr, socklen_t addrlen);
```

- The process can read bytes that arrive on the connection whose endpoint is `addr` by reading from descriptor `sockfd`.
- Similarly, writes to `sockfd` are transferred along connection whose endpoint is `addr`.

Best practice is to use `getaddrinfo` to supply the arguments `addr` and `addrlen`.

Sockets Interface: `listen`

- By default, kernel assumes that descriptor from socket function is an active socket that will be on the client end of a connection.
- A server calls the listen function to tell the kernel that a descriptor will be used by a server rather than a client:

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

- Converts `sockfd` from an active socket to a **listening socket** that can accept connection requests from clients.

- `backlog` is a hint about the number of outstanding connection requests that the kernel should queue up before starting to refuse requests.

Sockets Interface: `accept`

- Servers wait for connection requests from clients by calling `accept`:

```
int accept(int listenfd, SA *addr, int *addrlen);
```

- Waits for connection request to arrive on the connection bound to `listenfd`, then fills in client's socket address in `addr` and size of the socket address in `addrlen`.
- Returns a **connected descriptor** that can be used to communicate with the client via Unix I/O routines.

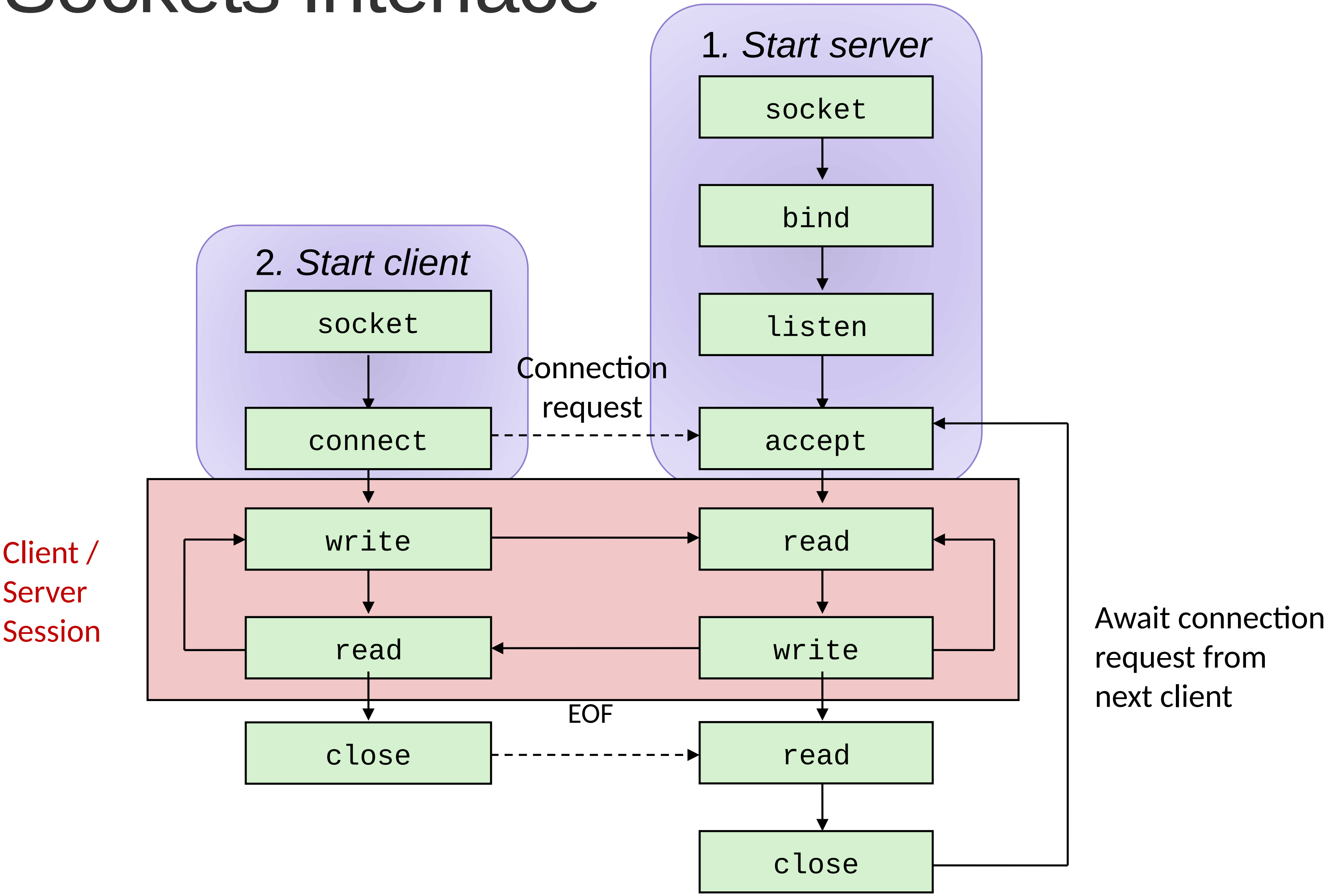
Connected vs. Listening Descriptors

- Listening descriptor
  - End point for client connection requests
  - Created once and exists for lifetime of the server

- Connected descriptor
  - End point of the connection between client and server
  - A new descriptor is created each time the server accepts a connection request from a client
  - Exists only as long as it takes to service client

- Why the distinction?
  - Allows for concurrent servers that can communicate over many client connections simultaneously
    - E.g., Each time we receive a new request, we fork a child to handle the request

Sockets Interface



Sockets Interface: connect

- A client establishes a connection with a server by calling connect:

```
int connect(int clientfd, SA* addr, socklen_t addrlen);
```

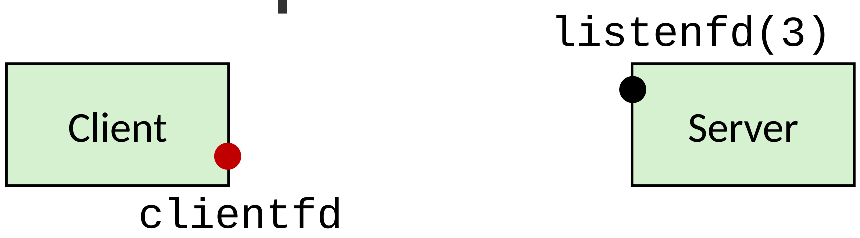
- Attempts to establish a connection with server at socket address `addr`
  - If successful, then `clientfd` is now ready for reading and writing.
  - Resulting connection is characterized by socket pair

    `(x:y, addr.sin_addr:addr.sin_port)`

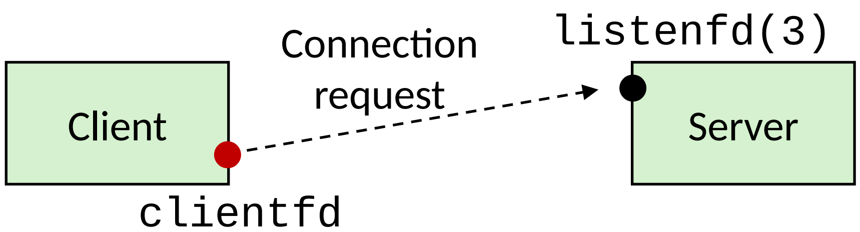
    - x is client address
    - y is ephemeral port that uniquely identifies client process on client host

Best practice is to use `getaddrinfo` to supply the arguments `addr` and `addrlen`.

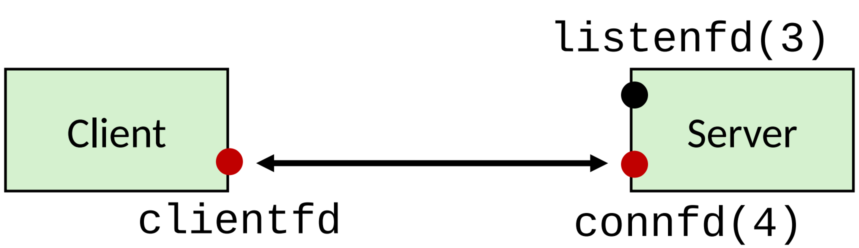
`accept` Illustrated



1. Server blocks in `accept`, waiting for connection request on listening descriptor `listenfd`

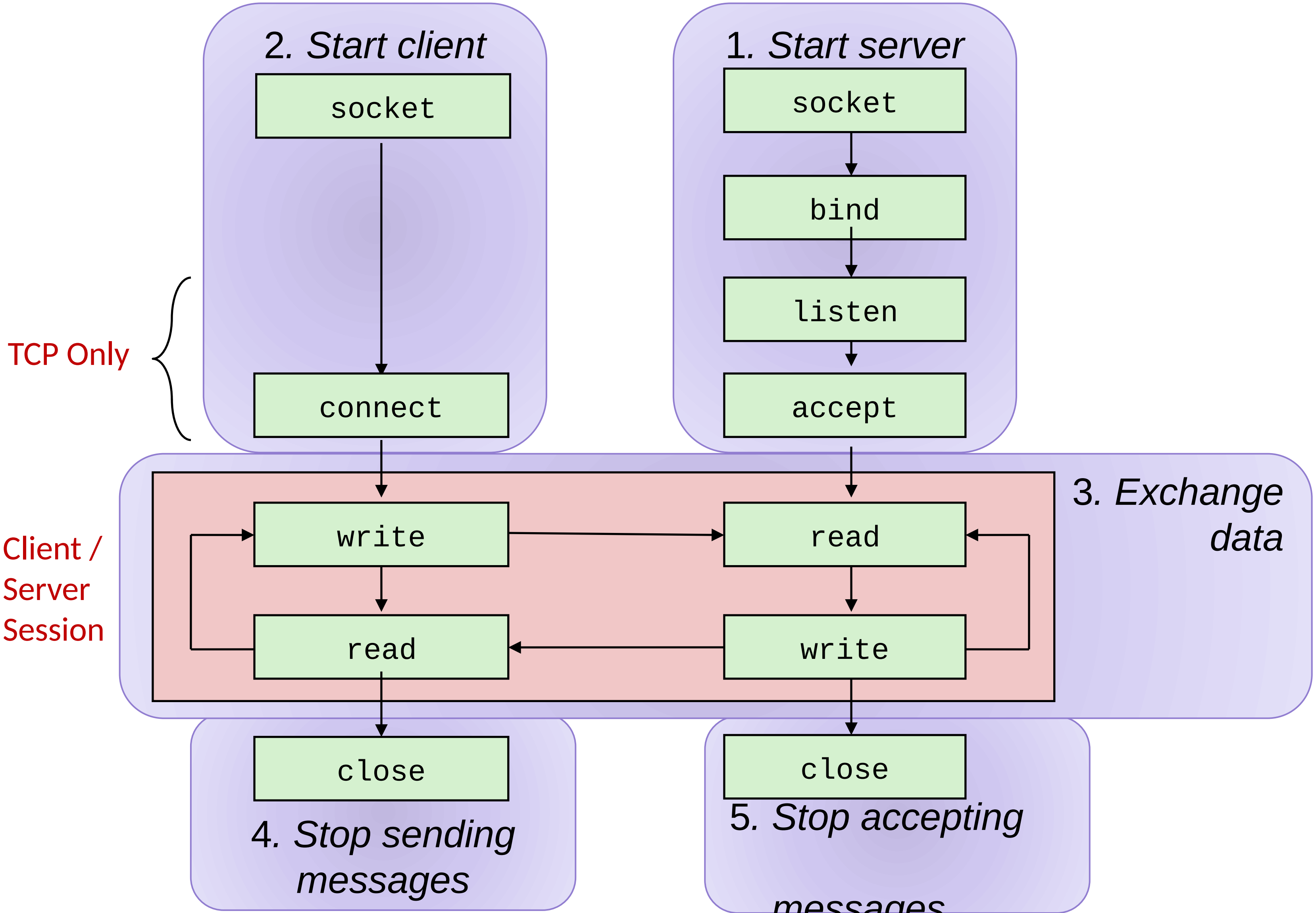


2. Client makes connection request by calling and blocking in `connect`

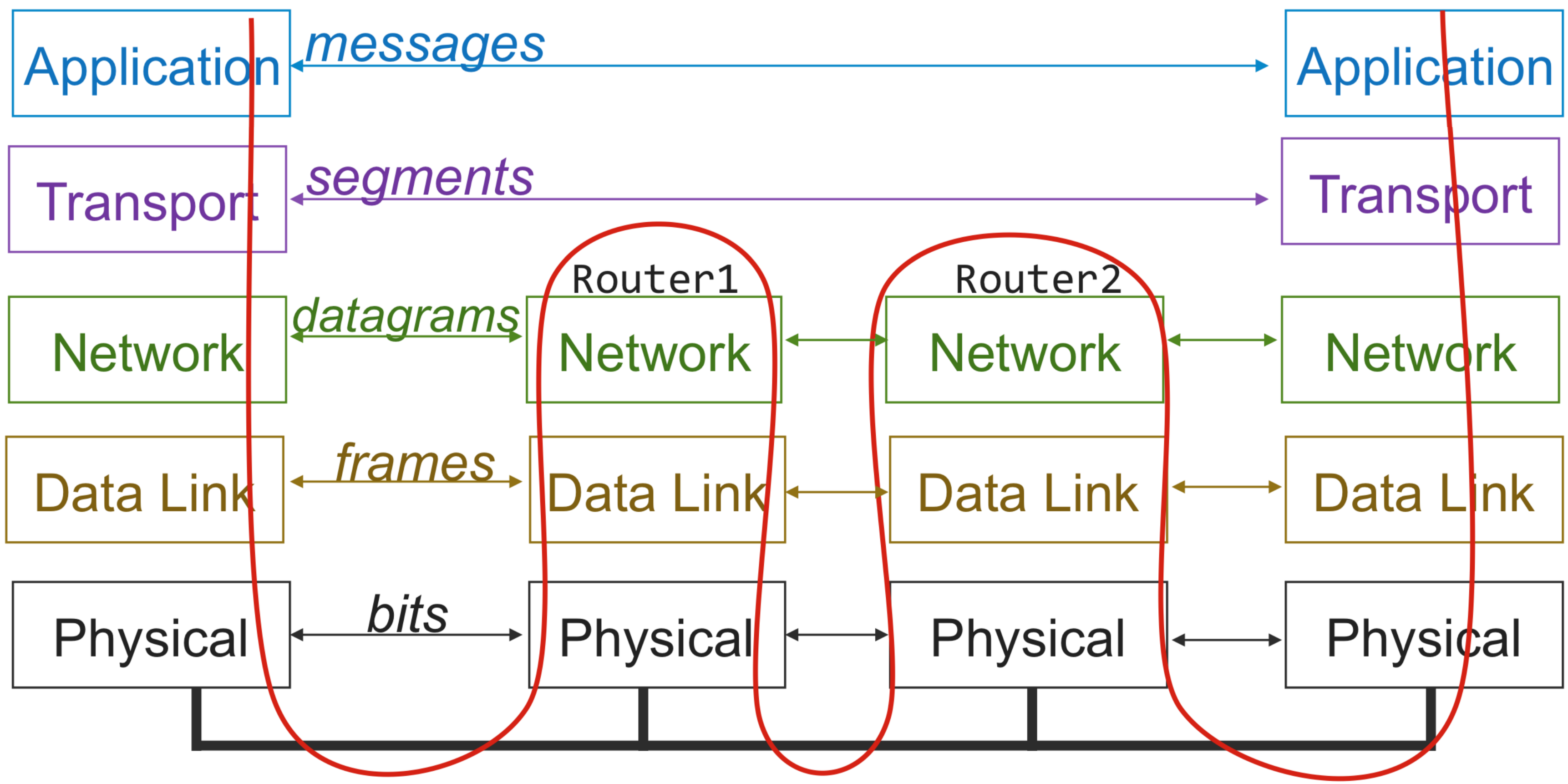


3. Server returns `connfd` from `accept`. Client returns from `connect`. Connection is now established between `clientfd` and `connfd`

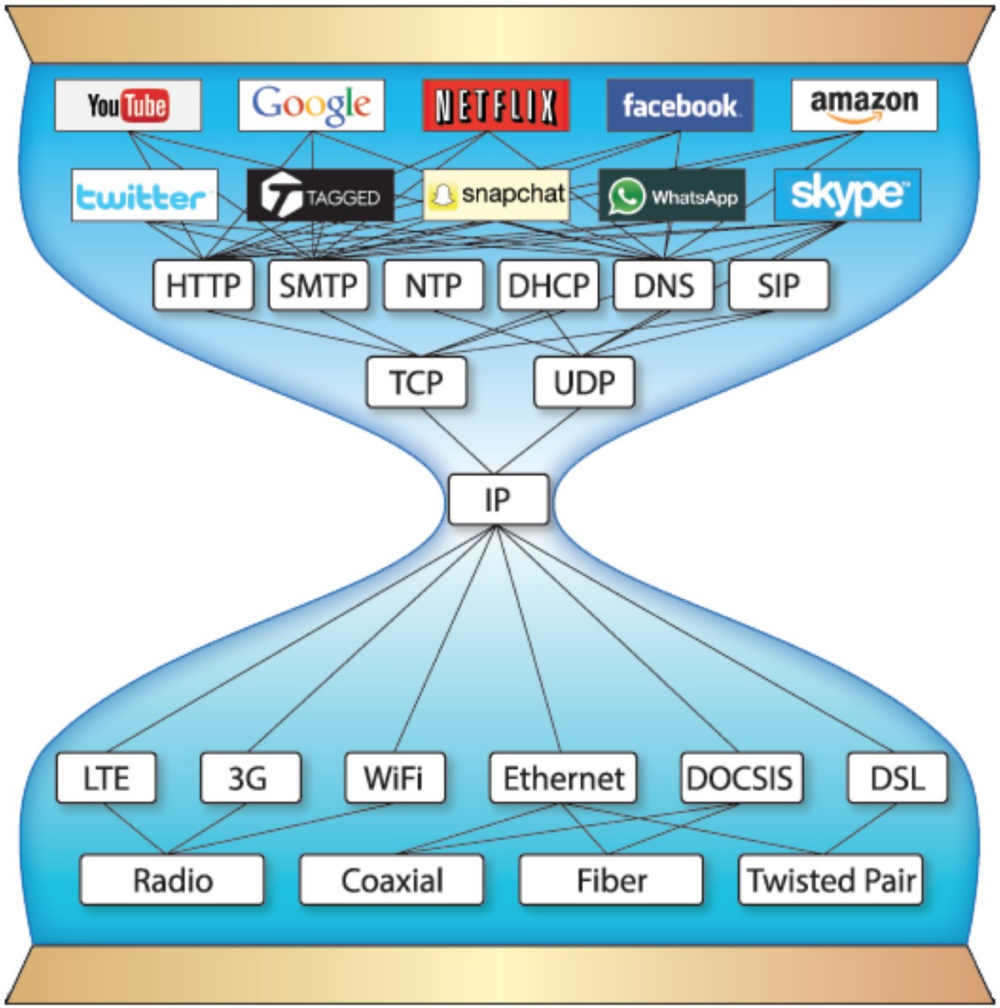
Sockets Interface



The Big Picture



The Big Picture



Application

Transport

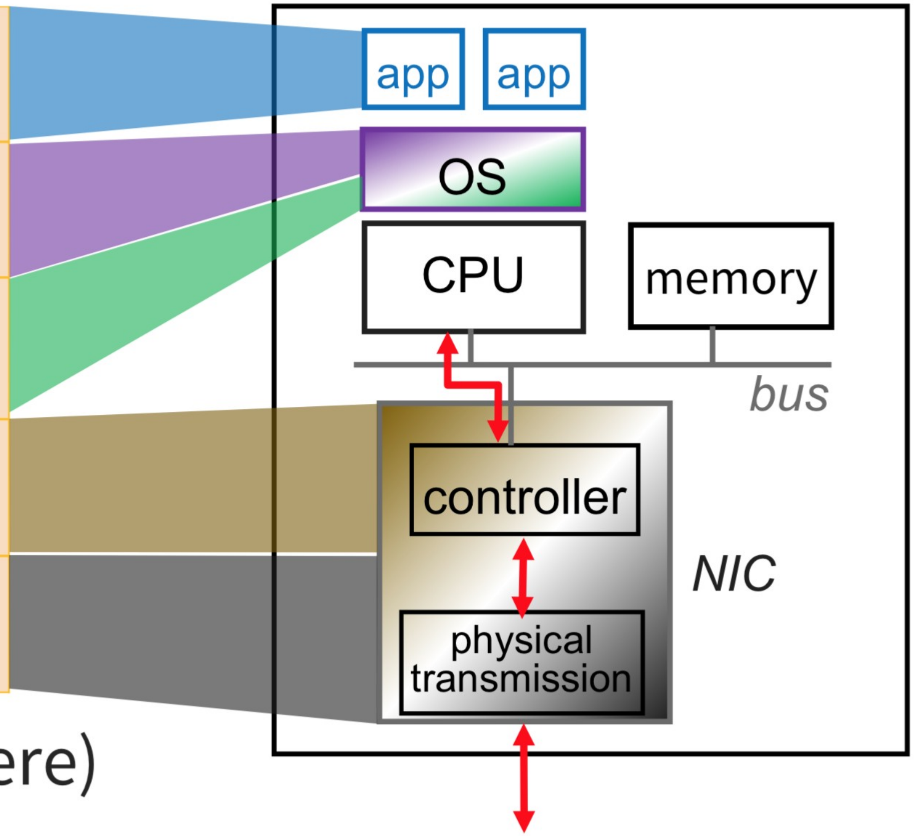
Network

Data Link

Physical

Hardware and Software Interfaces

Application	HTTP, FTP, DNS (*these*^ are usually in libraries)
Transport	TCP, UDP
Network	IP, ICMP (ping)
Link	Ethernet, WiFi
Physical	wires, signal encoding



(Hard to draw firm lines here)