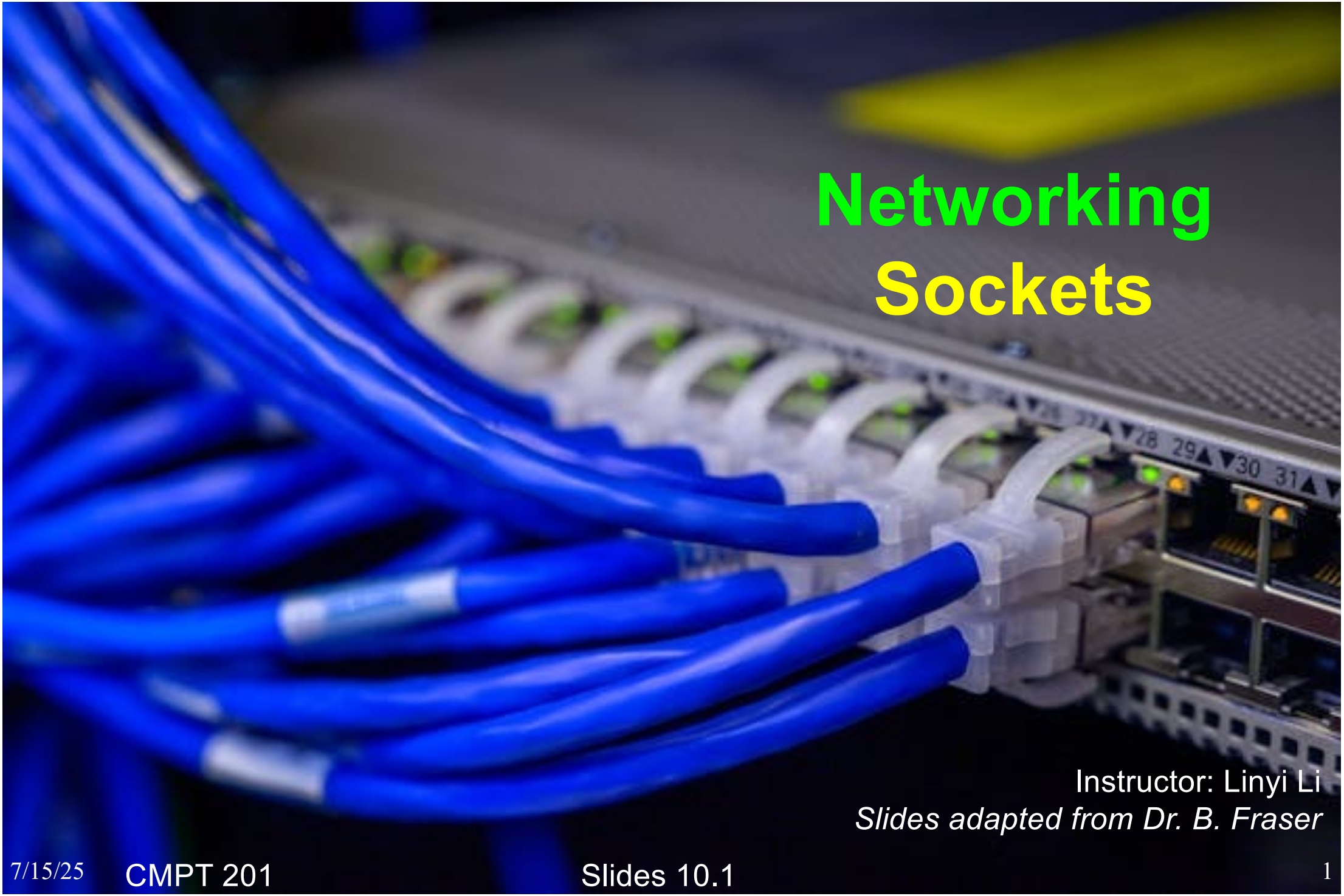


Networking
Sockets

Instructor: Linyi Li
Slides adapted from Dr. B. Fraser

Topics

- How does software do something complicated like networking? Layers!
- What are the two types of sockets?
- What syscalls can we use to work with sockets?

Networking

- Programs can communicate with each other via a network.
  - Can be across a network (wifi, wired, ...)
  - Can be on the same computer!
- More Resources
  - Beej's Guide to Network Programming
    https://beej.us/guide/bgnet/ is popular.
  - The Linux Programming Interface (our recommended text)
    is also great.

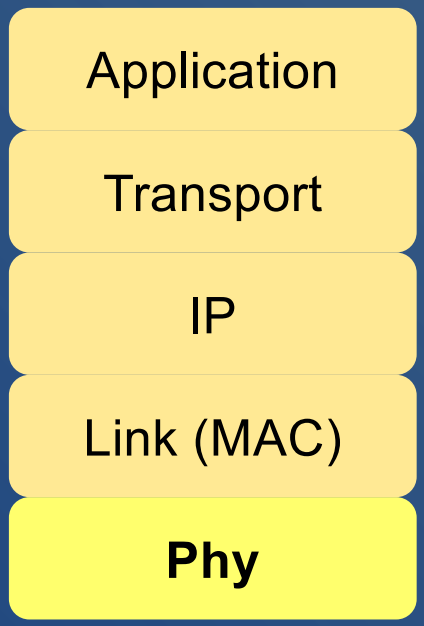
Basics of the Networking Stack

Networking Stack

- Stack
  - .. Software uses a “network stack” to organize responsibilities into layers.
  - Each layer provides a service to the layer above it.

Physical Layer

- Phy (Physical) Layer:
  .. Does hardware Control:
  generates and receives signals.
  – Need to know how to
  physically send and receive data.
  – Focuses on voltage and signalling.



- Analogy:
  Amazon package delivery:
  .. need a car and a driver.

Link Layer

- Link (MAC) Layer:
  ..Does local network addressing and routing
  - This is only for a (small area) local network.
  - E.g., wired or wireless local network.
  - LAN = Local Area Network

- Link layer has MAC addresses for addressing.
  - MAC =.. Medium Access Control

  MAC address look like: 05:35:5a:30:f9:05

- Analogy: Amazon package delivery:
  - need an address and routes

  (how to get there).

IP (Network) Layer

- IP ("Network") Layer:

  .. Does inter-network addressing and routing

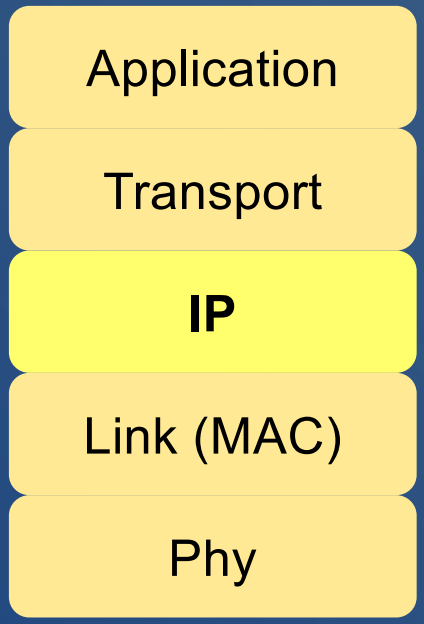
  – IP =.. Internet Protocol

- What if you want to connect a wired local network with a wireless local network?

  – Still need addressing and routing but it needs to be something common for both wired and wireless.

- ..This layer uses IP addresses for addressing.

  – IP addresses look like: 192.168.7.53



Transport Layer

- Transport Layer:
  - Can do packet tracking / retransmission
- Imagine sending/receiving lots of packages: 3 problems can occur:
  - Packages can be lost

Think car crash; or human errors like losing a package in a warehouse.

  - Packages can arrive out of order

They may be delivered by different trucks via different routes.

  - Packages can be duplicated

If the sender mistakenly thinks the package is lost and re-sends.

Transport Layer (cont)

- Some applications require an in-order reliable byte stream.
- Need a way to control these things:
  - TCP (Transmission Control Protocol) provides protection against these things:
    - No loss, no out of order, or duplication.
  - UDP (User Datagram Protocol) does not provide any protection!

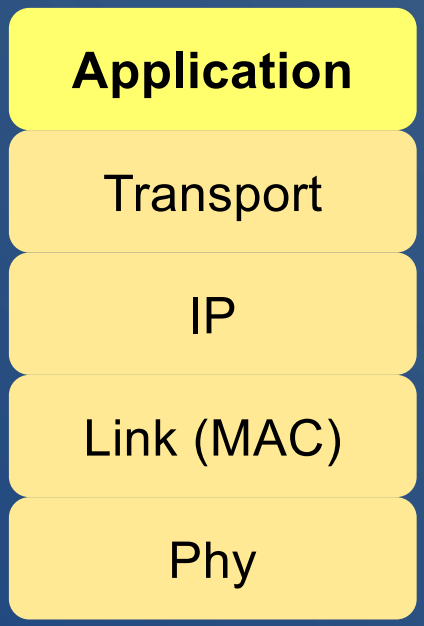
Raw datagrams = packets.

- Port Number
  - Use a socket port number to identify where on a computer (~which application) to send a packet
  - E.g., port 80, or 443

Application Layer

- Application Layer:
  .. what the application is doing.

  –Often features a well-known protocol such as: HTTP and FTP.



ABCD Spot the Address

- Which of the following is _____?

1) an IP Address
2) a MAC Address
3) Port Number

(a) 8001

(b) 19:02:16:08:07:01

(c) 153.10.23.103

(d) 0xF532 5E85 0005 235F

Socket Interface

Socket Syscalls

- An application can use a socket to communicate with another process (local or remote)

- There are five key syscalls
  - socket()
  - bind()
  - listen()
  - accept()
  - connect()

socket()

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

- .. Returns a file descriptor.
- Functions to send/receive
  - socket-specific calls: send(), recv(), sendto(), recvfrom()
  - file I/O calls: read(), write()

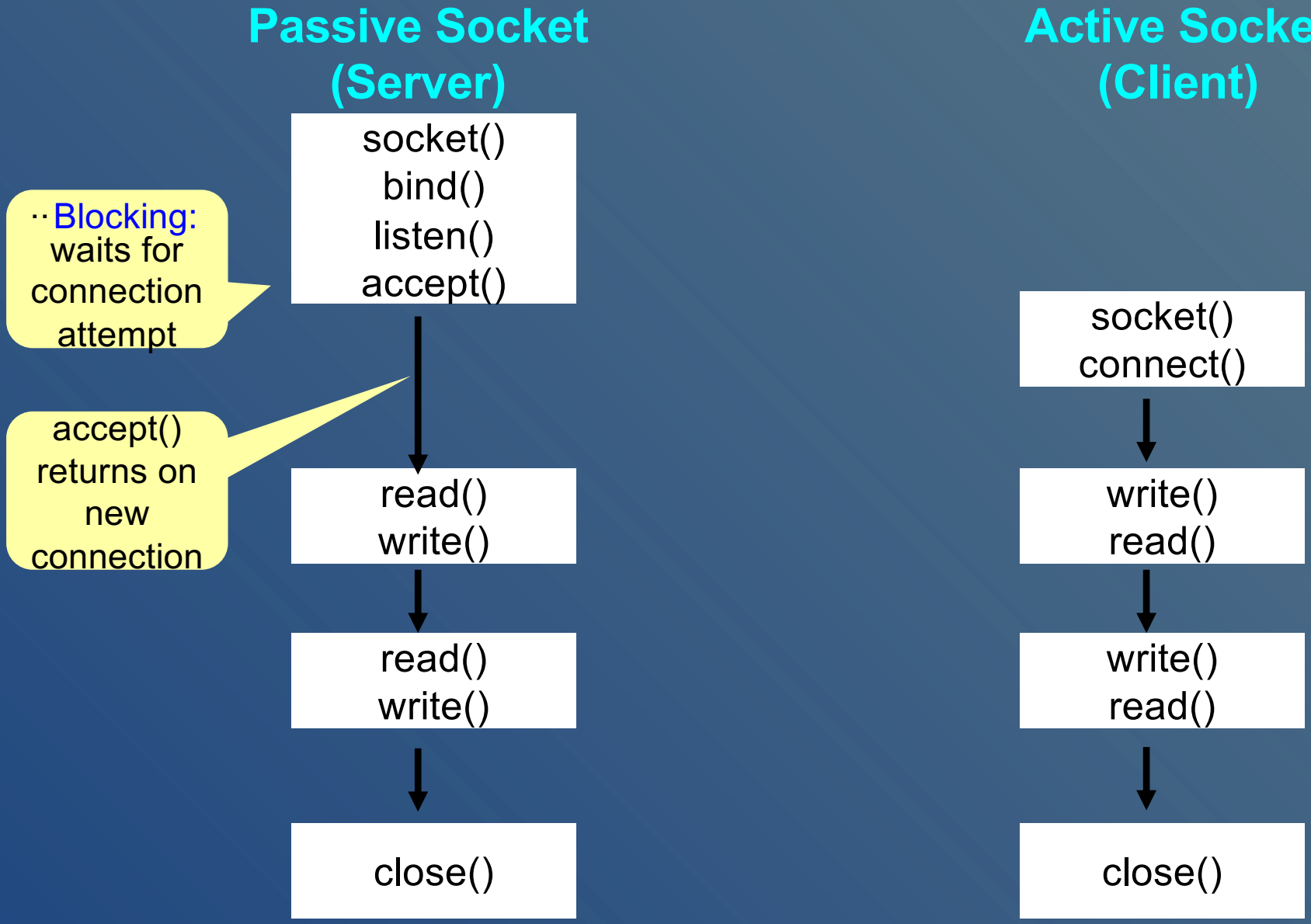
- int domain
  - Specifies what protocol is used. What is a protocol?
    - .. It defines a set of rules that an entity needs to follow to communicate with another entity using the same rules.
  - Domain examples
    - AF_UNIX: Local communication (this computer)
    - AF_INET: IPv4 Internet protocols
    - AF_INET6: IPv6 Internet protocols

socket() cont

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

- int type
  - SOCK_STREAM: TCP
    - sequenced, reliable, two-way, connection-based byte stream.
      - Connection-based / connection-oriented: will explain later
  - SOCK_DGRAM: UDP
    - datagrams (connectionless, unreliable packets of a max length).
      - Connectionless: will explain later.
- int protocol
  - Always 0 for us; not used for AF_UNIX, AF_INET, and AF_INET6.
  - Some domains allow different protocols.

Stream Socket Sequence (TCP)



TCP Explanation: bind()

- socket() creates a socket.
- bind()..binds the socket to an address.
  - Uses a generic address struct.

```
struct sockaddr {
    sa_family_t sa_family;
    char        sa_data[14];
    // size varies.
    // bind() given struct size.
```

  - Different protocols use different structs (with different-yet-similar names, and different fields).

TCP Explanation: listen(), accept()

- listen().. marks the socket as passive
  - i.e., it's used to wait for a connection to come (a server).
  - By default, a socket is active.
- accept()..accepts a new connection.
  - Returns a **new socket** to use for the new connection.
  - The original socket is only used to accept new connections.
- connect().. Client connects to a passive socket.
  - "connection-oriented" means we establish a connection first.

ABCD TCP Call Sequence

- Which of the following is the most likely sequence of calls for a TCP server?

a)

```
socket()
bind()
listen()
read()
accept()
write()
close()
```

b)

```
socket()
bind()
listen()
accept()
read()
write()
close()
```

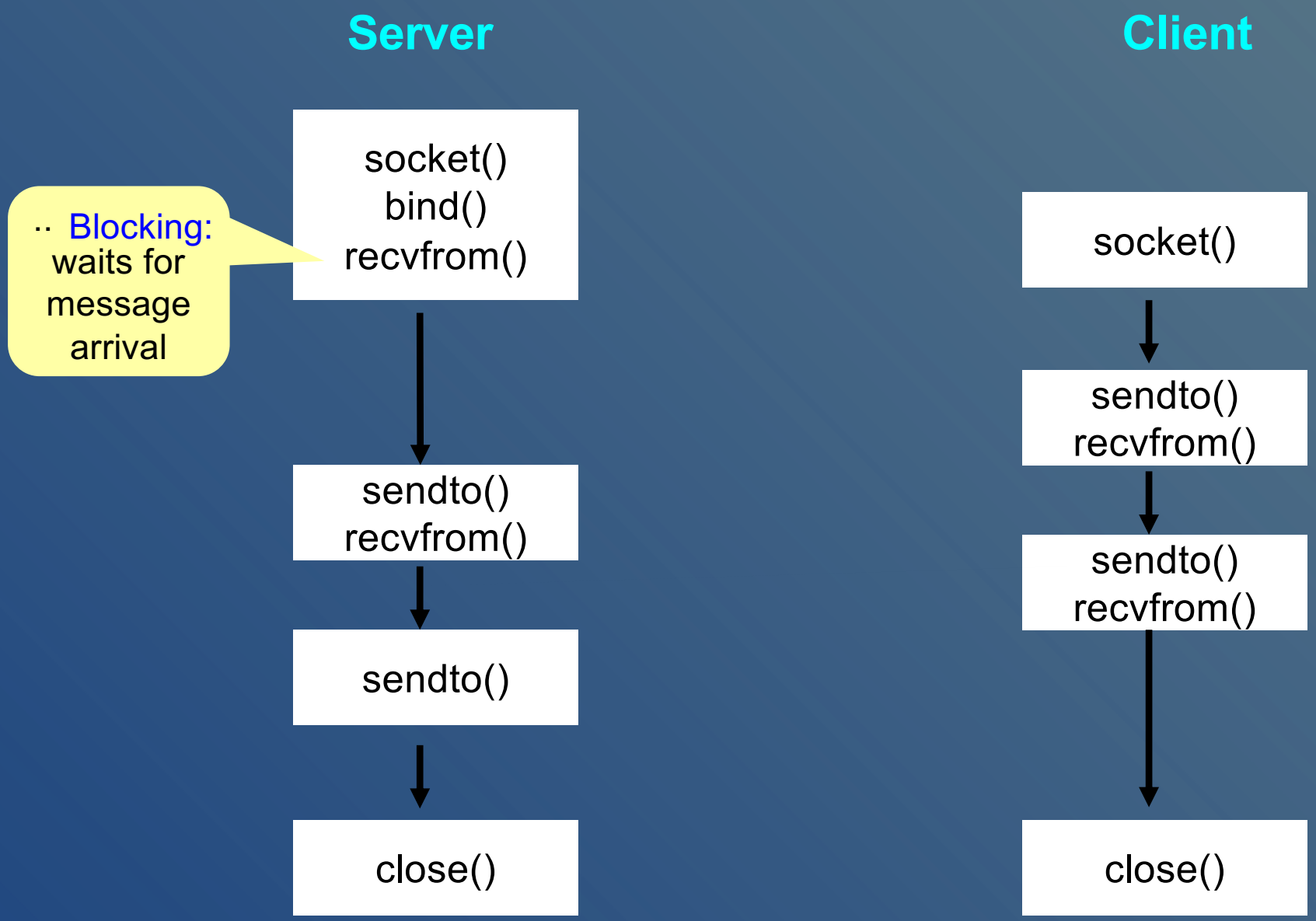
c)

```
socket()
bind()
listen()
accept()
write()
read()
close()
```

d)

```
socket()
bind()
write()
listen()
accept()
read()
close()
```

Datagram Socket Sequence (UDP)



UDP Explanation

- "connectionless" means
  .. we do not establish a connection first.
  - It is like an SMS message that is received one-off
  - Each time we receive a message we are told who sent it.
- UDP has no active or passive sockets
  - sendto() needs to specify the receiver's address every time.
  - recvfrom() tells you who sent it.

ABCD UDP Call Sequence

- Which of the following is the most likely sequence of calls for a UDP server?

a)

```
socket()
bind()
sendto()
recvfrom()
close()
```

b)

```
socket()
bind()
listen()
sendto()
close()
```

c)

```
socket()
bind()
read()
write()
close()
```

d)

```
socket()
bind()
recvfrom()
sendto()
close()
```

ABCD: Who's call is it?

- Which of the options on the right is most likely to use all of the following calls (*not in order*):

```
connect()
close()
read()
socket()
write()
```

(a) UDP Client

(b) UDP Server

(c) TCP Client

(d) TCP Server

ABCD: Who's call is it?

- Which of the options on the right is most likely to use all of the following calls (*not in order*):

bind()
close()
recvfrom()
sendto()
socket()

(a) UDP Client
(b) UDP Server
(c) TCP Client
(d) TCP Server

ABCD: Who's call is it?

- Which of the options on the right is most likely to use all of the following calls (*not in order*):

accept()
bind()
close()
listen()
read()
socket()
write()

(a) UDP Client

(b) UDP Server

(c) TCP Client

(d) TCP Server

TCP Activity

- Create two TCP programs: server and client.
  - Implement the socket sequence using AF_UNIX. (Local machine)
  - The client should be able to send messages typed on the terminal to the server.
  - The server should be able to print out the messages.
  - man unix for detailed info for AF_UNIX.
  - An AF_UNIX address uses struct sockaddr_un:

```
struct sockaddr_un {
   sa_family_t sun_family;      /* AF_UNIX */
   char        sun_path[108];   /* Pathname = "tmp" */
};
```

UDP Activity

- Create two UDP programs: server and client.
  - Implement the socket sequence using AF_UNIX. (Local machine)
  - The client should be able to send messages typed on the terminal to the server.
  - The server should be able to print out the messages.
  - man unix for detailed info for AF_UNIX.
  - An AF_UNIX address uses struct sockaddr_un:

```
struct sockaddr_un {
   sa_family_t sun_family;      /* AF_UNIX */
   char        sun_path[108];   /* Pathname = "tmp" */
};
```

Summary

- Network Stack has layers (bottom-up)
  - phy, link, IP, transport, application
- Socket: Connect to communicate across network.
- TCP:
  - Connection-oriented; in-order delivery.
  - Server:
    socket(), bind(), listen(), accept(), read(), write()... close()
  - Client: socket(), connect(), write(), read(), ... close()
- UDP:
  - Connectionless
  - Server: socket(), bind(), recvfrom(), sendto(), ... close()
  - Client: socket(), sendto(), recvfrom(), close()