Socket programming

CE 352, Computer Networks

Salem Al-Agtash

Lecture 7

Recap

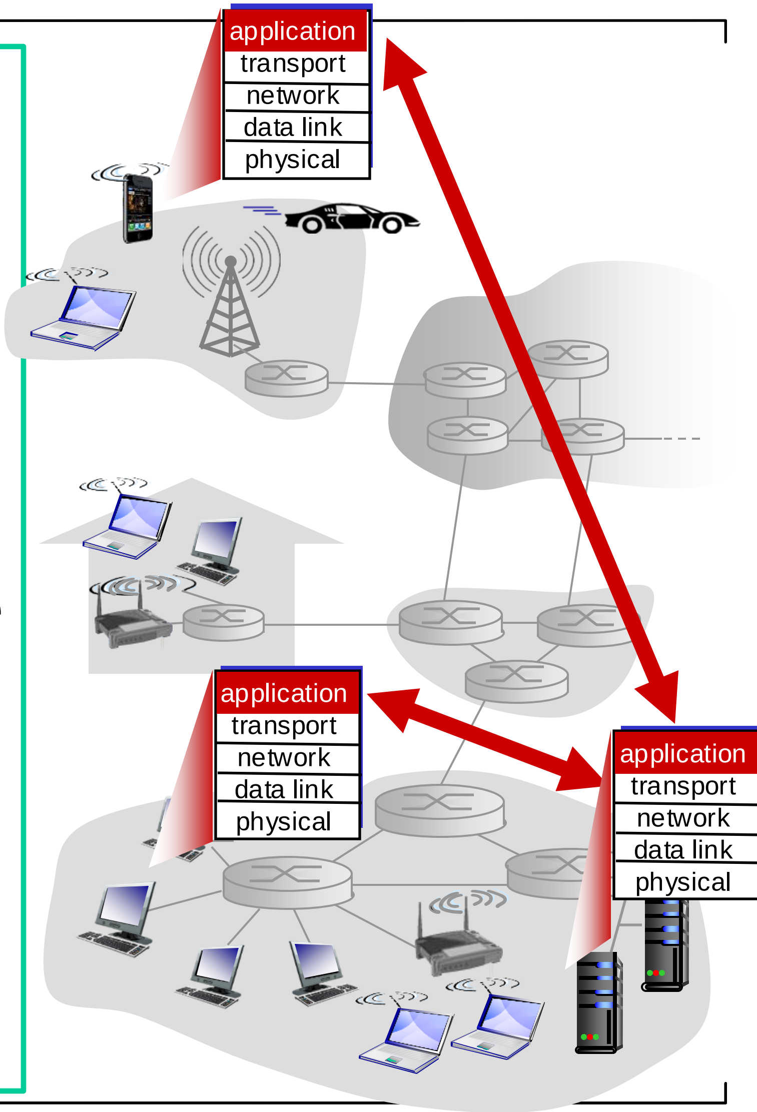
- Application layer
- Client – Server, Peer-to-Peer
- Communication (IPC, Sockets)
- Application protocols (http, FTP, ..)
- Transport protocol (TCP, UDP)
- The Web – WWW,

Today:

Socket Programming

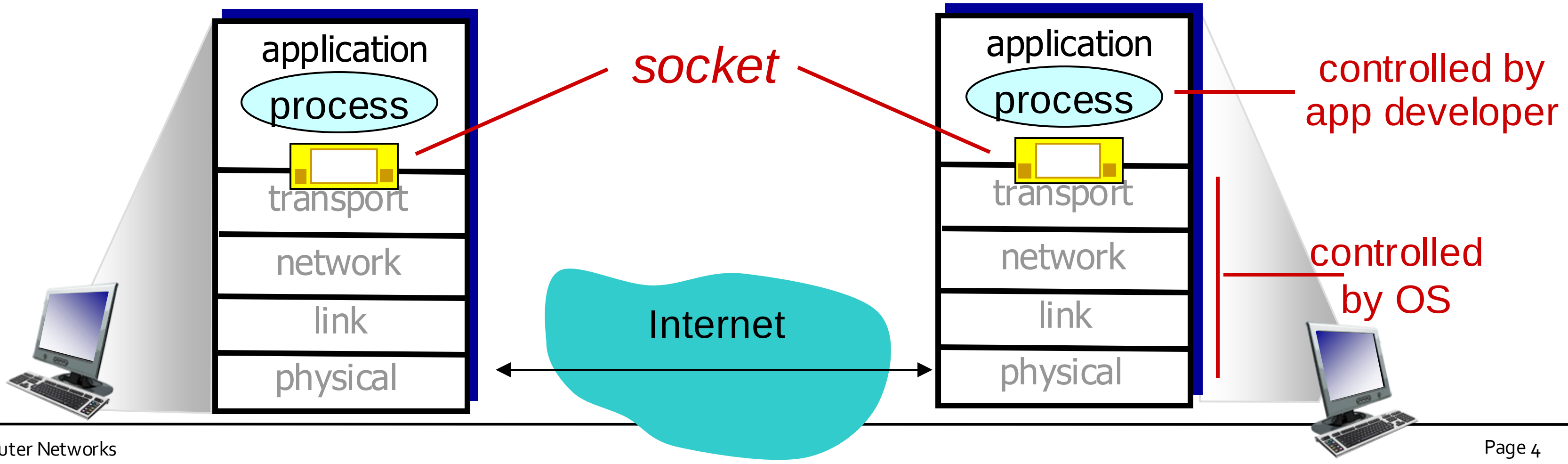
Recap (Applications on the Network)

- End-end system programs
- Architecture
  - Client – Server
  - Peer-to-peer (P2P)
- no need to write software for network-core devices
- Examples:
  - Web, e-mail, text messaging, remote login, file transfer
  - social networking, multi-user network games
  - VoIP, streaming stored video (YouTube, Hulu, Netflix)



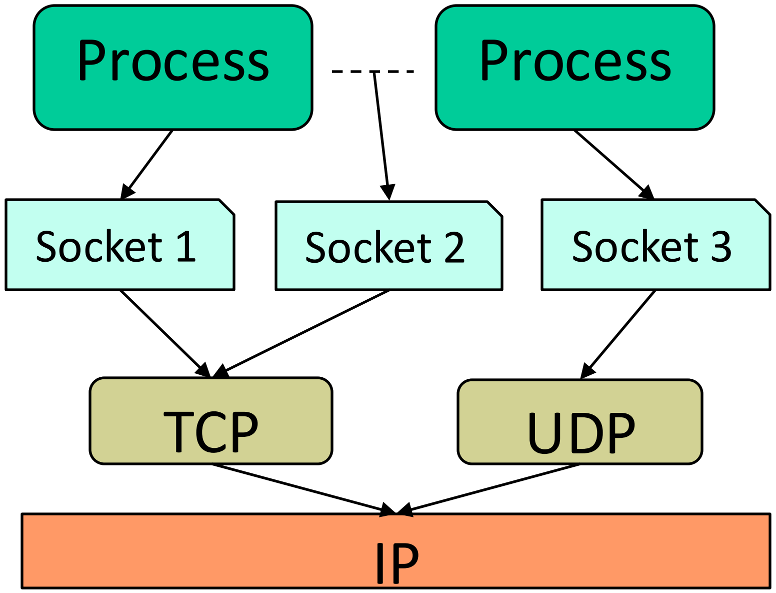
Recap (process communication)

- Process: Program in Execution
  - Same hosts: processes communicate using IPC defined by OS. e.g. Pipes, Shared Memory, Message Queues
  - Different hosts: processes communicate by exchanging messages. e.g. Client-Server, P2P
- Socket: Process sends/ receives messages via socket (IP + Port)
  - Sending process shoves message out door and relies on transport infrastructure to deliver message to socket at receiving process



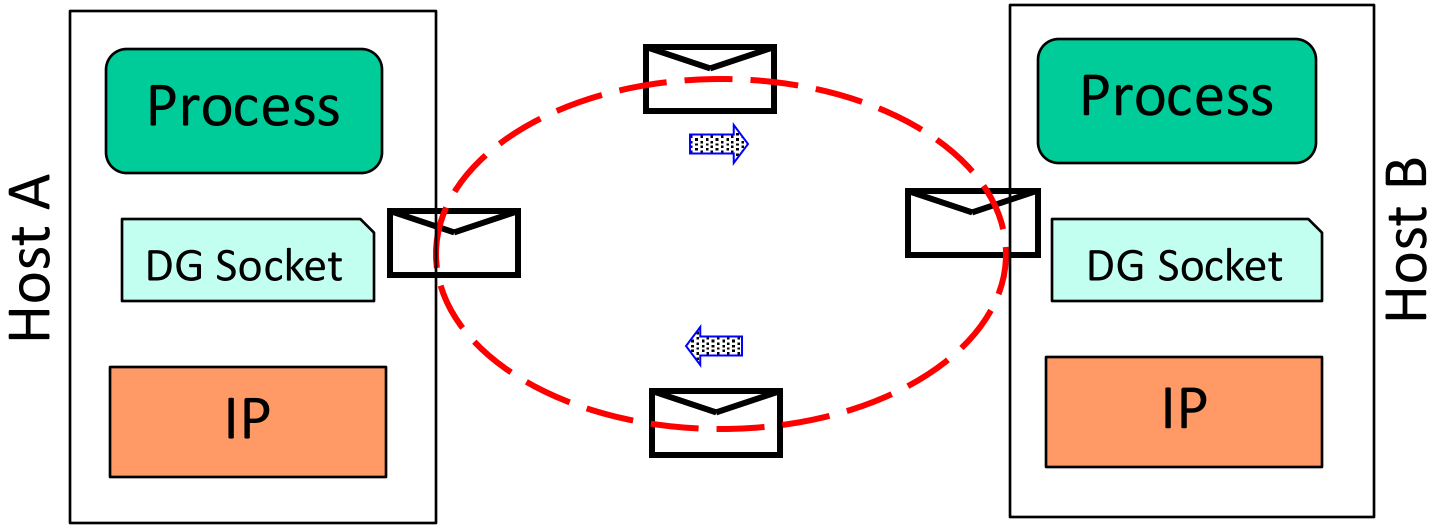
Berkeley Sockets

- Communication APIs developed for Unix systems in C programming language
- Socket types for transport services:
  - *UDP:* unreliable datagram, SOCK_DGRAM
  - *TCP:* reliable, byte stream-oriented, SOCK_STREAM
- Socket families:
  - *Unix internal protocols:* AF_UNIX
  - *Internet protocols:* AF_INET



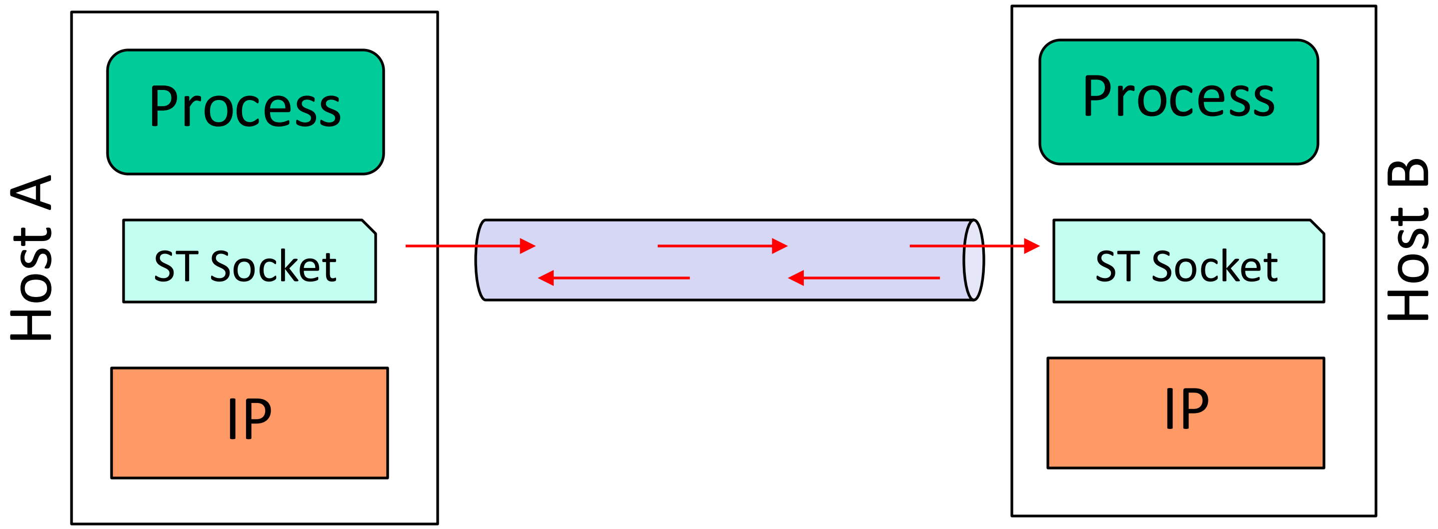
Datagram socket: UDP

- UDP: no "connection" between client & server
  - no handshaking before sending data
  - sender explicitly attaches IP destination address and port # to each packet
  - receiver extracts sender IP address and port# from received packet
- UDP: transmitted data may be lost or received out-of-order
- Application viewpoint: UDP provides *unreliable* transfer of groups of bytes ("datagrams") between client and server



Stream socket: TCP

- TCP: "connection" between client & server
  - Sever creates socket and begins to listen
  - Client contacts server by creating TCP socket, specifying IP/port of server
  - Server creates thread to communication with particular client
- Application viewpoint: TCP provides *reliable* in-order byte stream transfer ("pipe") between client and server



System calls

Fill in IP and Port

- `struct sockaddr_in servAddr, clientAddr;`

Create a socket

- `socket(AF_INET,SOCK_STREAM,0);`

Bind the socket

-
  ```
  bind(sockfd,(..)&servAddr,
                   sizeof(servAddr))
  ```

Server listens for connections

- `listen(sockfd,n);`

Client connects to a server

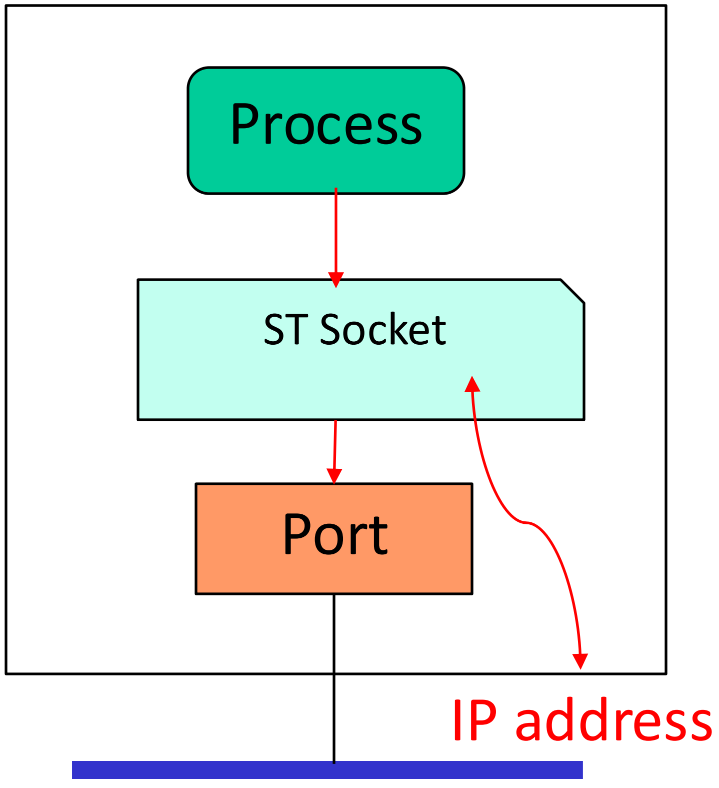
- `connect(sockfd,(..sockaddr*)&servAddr, sizeof(servAddr));`

Sever accepts connection

- `accept(sockfd,(struct sockaddr *)&clientAddr,sizeof(clientAddr));`

Read/write, send/receive

Binding address to socket



Socket data structures

```
struct  sockaddr_un {
        short  sun_family;   /* AF_UNIX*/
        char  sun_path[108] ;
   };
```

```
struct sockaddr_in {
   short  sin_family ; /*AF_INET*/
   u_short sin_port ; /* 16 bit port number */
   struct  in_addr  sin_addr; /*IP address*/
   char  sin_zero[8]; /*padding*/
};
struct  in_addr {
      u_long  s_addr ;
    } ;
```

socket ()

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

- #include <sys/socket.h>
- Domain: AF_UNIX or AF_INET (AF_INET6 for IPv6)
- Type: SOCK_STREAM or SOCK_DGRAM
- Protocol: typically 0 (system selects)

```
int sockfd =socket(AF_INET,SOCK_DGRAM,0);

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

bind ()

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

- Assigns address to the socket
- my_addr – of type struct sockaddr_in and needs to cast protocol independent struct (sockaddr *)

Example:

```
struct sockaddr_in servAddr;
servAddr.sin_family = AF_INET;
servAddr.sin_addr.s_addr = htonl(INADDR_ANY);
servAddr.sin_port = htons(5000);
bind(sockfd, (const struct sockaddr*)&servAddr, sizeof(servAddr))
```

Byte order, cast, INADDR_ANY

- Byte ordering (network: big-endian, host: little endian)

The **most significant** byte (the "big end") of the data is placed at the byte with the lowest address. The rest of the data is placed in order in the next three bytes in memory.

The **least significant** byte (the "little end") of the data is placed at the byte with the lowest address. The rest of the data is placed in order in the next three bytes in memory.

  - htonl(), htons(): host order to network order long, short
  - ntohl(), ntohs(): network order to host order long, short

- Cast (struct sockaddr_in*) to (struct sockaddr*)
  - bind() takes in protocol-independent (struct sockaddr*)

    -
      ```
      struct sockaddr {
              unsigned short sa_family; // address family
              char sa_data[14]; // protocol address
          };
      ```

- INADDR_ANY
  - bind a socket not to a specific IP, rather the socket accepts connections to all the IPs of the machine

listen( )

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

- Server establish listen queue when ready to receive data
- sockfd is the socket file descriptor
- n is the number of pending connections

Example:

```
listen(sockfd,3);
```

accept ( )

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

- Server accepts connection from client requesting to connect
- sockfd is the socket file descriptor
- *addr is a pointer to store client address,
- (struct sockaddr_in *) cast to (struct sockaddr *)
- addrlen – pointer to store size of addr (client address)

```
int clen = size(clientAddr);
int con_sockfd = accept(sockfd,(struct
                  sockaddr *) &clientAddr, &clen);
```

connect()

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

```
struct hostent {
    char * h_name;
    char ** h_aliases;
    int h_addrtype;
    int h_length;
    char ** h_addr_list;
    char *h_addr
};
```

```
struct sockaddr_in servAddr;
struct hostent *host;
                                   "localhost" or "127.0.0.1"
host=(struct hostent*)gethostbyname("www.coenclass.org");
servAddr.sin_family = AF_INET;
servAddr.sin_port = htons(5000);
servAddr.sin_addr = *((struct in_addr *)host->h_addr);
connect(sockfd, (struct sockaddr *)&servAddr, sizeof(struct
                                               sockaddr)))
```

send ( ) or sendto ( )

Used for TCP socket

```
ssize_t send(int con_sockfd, const void* buf,
                                         size_t len, int flags)
```

Used for UDP socket

```
ssize_t sendto(int socket, void *buffer, size_t
    length, int flags, const struct sockaddr*saddr,
                                                socklen_t addrlen );
```

```
send(con_sockfd, message,strlen(message), 0);
sendto(sockfd, message, strlen(message), 0, ……
 (struct sockaddr *)&servAddr,sizeof(servAddr));
```

recv ( ) or recvfrom ( )

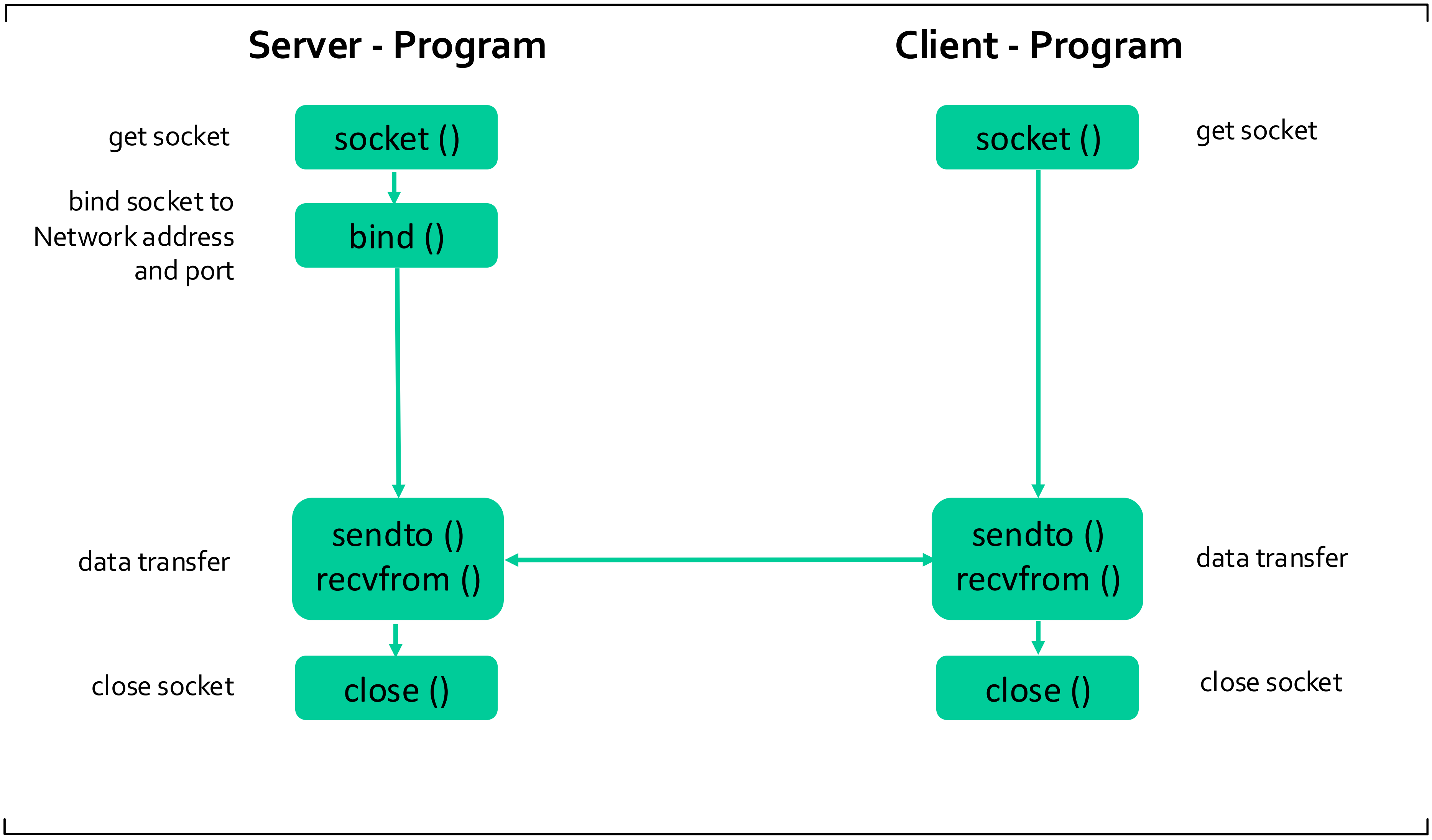
Used for TCP socket

```
ssize_t recv(int con_socket, void *buffer,
                        size_t length, int flags);
ssize_t recvfrom(int socket, void *buffer, size_t
   length, int flags, const struct sockaddr*caddr,
                                    socklen_t addrlen );
```

Used for UDP socket

```
recv(con_sockfd, message,strlen(message), 0);
n = recvfrom(sockfd, (char *)buffer, 1024, 0,…
  , (struct sockaddr*)&servAddr,sizeof(servAddr));
```

Client/server programs: UDP



Client/server interaction: UDP

server (running on serverIP)

create socket, port= x:

serverSocket =
socket(AF_INET,SOCK_DGRAM, 0)

```
bind(sockfd, (struct sockaddr *)&servAddr,
        sizeof(struct sockaddr))
```

read datagram from
serverSocket

write reply to
serverSocket
specifying
client address,
port number

client

create socket:

clientSocket =
socket(AF_INET,SOCK_DGRAM, 0)

Create datagram with server IP and
port=x; send datagram via
servAddr, clientSocket

read datagram from
clientSocket

close
clientSocket

UDP client

```
int sockfd;
char sbuf[1024];
struct sockaddr_in servAddr;
struct hostent *host;
host = (struct hostent *)gethostbyname("localhost");
if ((sockfd = socket(AF_INET, SOCK_DGRAM, 0)) < 0) {
perror("Failure to setup an endpoint socket");
exit(1);
}
servAddr.sin_family = AF_INET;
servAddr.sin_port = htons(5000);
servAddr.sin_addr = *((struct in_addr *)host->h_addr);

while(1){
printf("Client: Type a message to send to Server\n");
scanf("%s", sbuf);
sendto(sockfd, sbuf, strlen(sbuf), 0, (struct sockaddr *)&servAddr, sizeof(struc
sockaddr));
}
return 0;
}
```

create UDP socket for server →

Define server to send →

Prepare message to send to server →

Send message to socket for the identified server →

UDP server

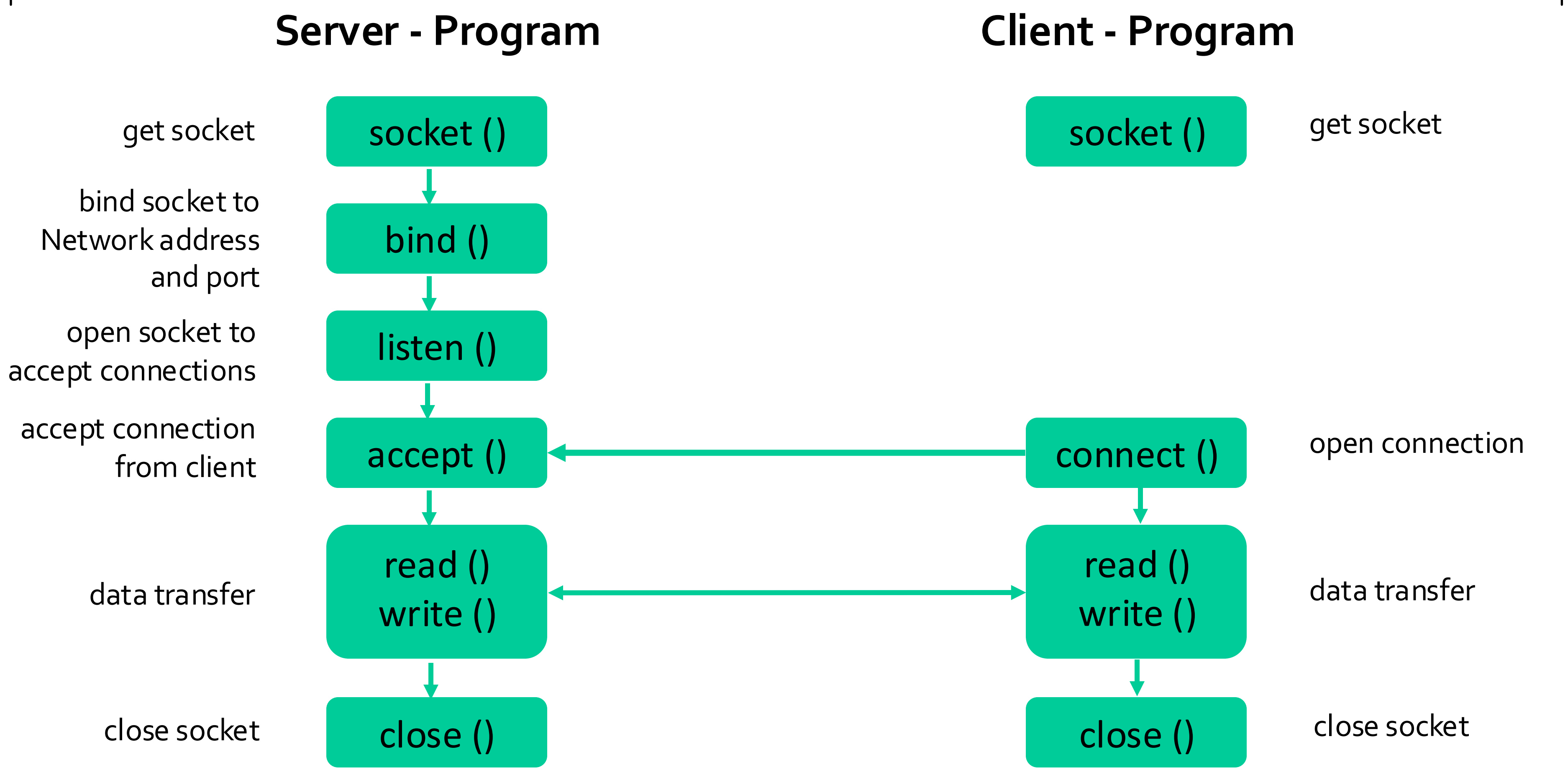
create UDP socket

bind socket to local IP and local port number 5000

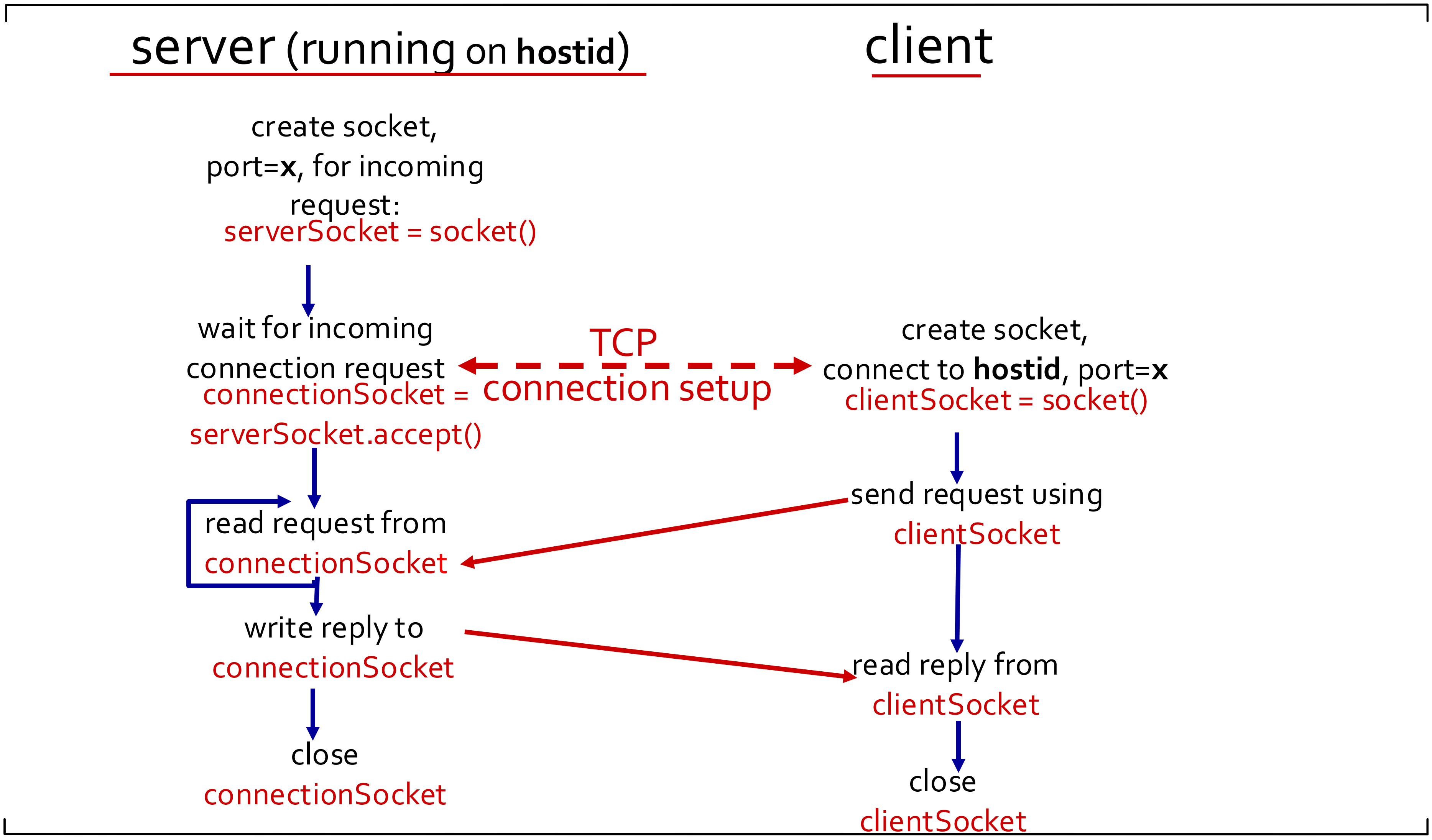
Read from UDP socket into message, getting client's address (client IP and port)

```
char rbuf[1024];
struct sockaddr_in servAddr, clienAddr;
socklen_t addrLen = sizeof(struct sockaddr);
if ((sockfd = socket(AF_INET, SOCK_DGRAM, 0)) < 0) {
perror("Failure to setup an endpoint socket");
exit(1);
}
servAddr.sin_family = AF_INET;
servAddr.sin_port = htons(5000); //Port 5000 is assigned
servAddr.sin_addr.s_addr = INADDR_ANY; //Local IP address if ((bind(sockfd, (struct
sockaddr *)&servAddr, sizeof(struct sockaddr))) < 0){
perror("Failure to bind server address to the endpoint socket");
exit(1);
}
while (1){
 printf("Server waiting for messages from client: \n");
 int nr = recvfrom(sockfd, rbuf, 1024, 0, (struct sockaddr *)&clienAddr, &addrLen);
rbuf[nr] = '\0';
 printf("Client with IP: %s and Port: %d sent message: %s\n",
inet_ntoa(clienAddr.sin_addr),ntohs(clienAddr.sin_port), rbuf);
}
return 0;
```

Client/server programs: TCP



Client/server interaction: TCP


server (running on **hostid**)
client
create socket, port=**x**, for incoming request:
serverSocket = socket()
wait for incoming connection request
connectionSocket = serverSocket.accept()
TCP connection setup
create socket, connect to **hostid**, port=**x**
clientSocket = socket()
send request using clientSocket
read request from connectionSocket
write reply to connectionSocket
read reply from clientSocket
close connectionSocket
close clientSocket

TCP client

Create TCP socket for server

Define server to connect

Write to socket descriptor

```
int sockfd, nr;
char sbuf[1024], rbuf[1024];
struct sockaddr_in servAddr;
struct hostent *host;
host = (struct hostent *)gethostbyname("localhost");
if ((sockfd = socket(AF_INET, SOCK_STREAM, 0)) < 0) {
perror("Failure to setup an endpoint socket");
exit(1);
}
servAddr.sin_family = AF_INET;
servAddr.sin_port = htons(5000);
servAddr.sin_addr = *((struct in_addr *)host->h_addr);
if (connect(sockfd, (struct sockaddr *)&servAddr, sizeof(struct sockaddr))){
perror("Failure to connect to the server");
exit(1);
}
while(1){
 printf("Client: Type a message to send to Server\n");
 scanf("%s", sbuf);
 write(sockfd, sbuf, strlen(sbuf));
 read(sockfd, rbuf, 1024);
 printf("Server: sent message: %s\n", rbuf);
}
close(sockfd);
return 0;
```

TCP server

create TCP socket

bind socket to IP and Port

Listen for connection requests on socket, then

accept connection

read message from client

write message to client

close connection

```
int sockfd, connfd, rb, sin_size;
char rbuf[1024], sbuf[1024];
struct sockaddr_in servAddr, clienAddr;
if ((sockfd = socket(AF_INET, SOCK_STREAM, 0)) < 0) {
perror("Failure to setup an endpoint socket");
exit(1);
}
servAddr.sin_family = AF_INET;
servAddr.sin_port = htons(5000);
servAddr.sin_addr.s_addr = INADDR_ANY;
if ((bind(sockfd, (struct sockaddr *)&servAddr, sizeof(struct sockaddr))) < 0){
perror("Failure to bind server address to the endpoint socket");
exit(1);
}
printf("Server waiting for client at port 5000\n");
listen(sockfd, 5);
sin_size = sizeof(struct sockaddr_in);
while (1){
if ((connfd = accept(sockfd, (struct sockaddr *)&clienAddr,  (socklen_t *)&sin_size)) < 0){
 perror("Failure to accept connection to the client");
 exit(1);
}
printf("Connection Established with client: IP %s and Port %d\n",    inet_ntoa(clienAddr.sin_addr),
ntohs(clienAddr.sin_port));
 while ((rb = read(connfd, rbuf, 1024))>0){
  rbuf[rb] = '\0';
  printf("Client sent: %s\n", rbuf);
  write(connfd, "Acknowledge", 20);
 }
 close(connfd);
}
close(sockfd);
return 0;
```

Summary

Today:

- Socket concept
- Berkeley socket – Unix/ Linux
- System calls:
  - socket, bind, connect, listen, accept, send/recv, sendto/recvfrom, read/write

Camino discussion:

- Reflection
- Exit ticket

Next time:

- Read read 2.2.5 and 2.4 of K&R
- follow on Canvas! material and announcements

Any questions?