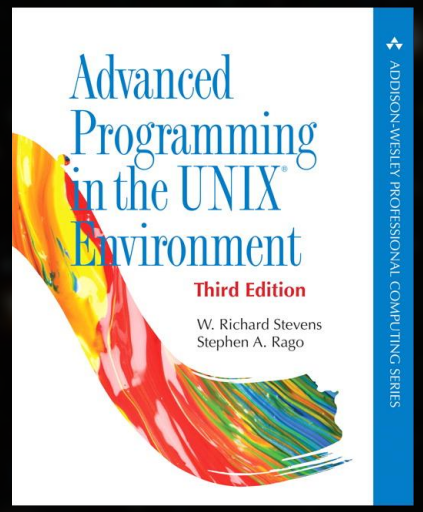
Knife in the Water (1962), Roman Polanski

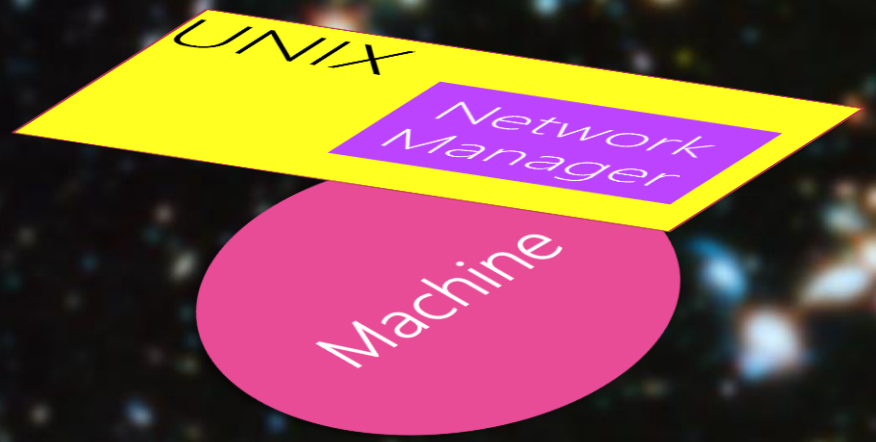
Lab08: Remarking based on new breakdown

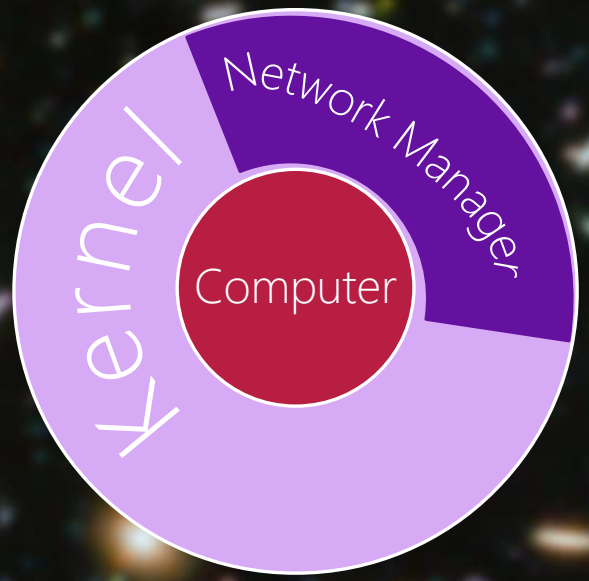
Lab09 & Lec09 Keys Released

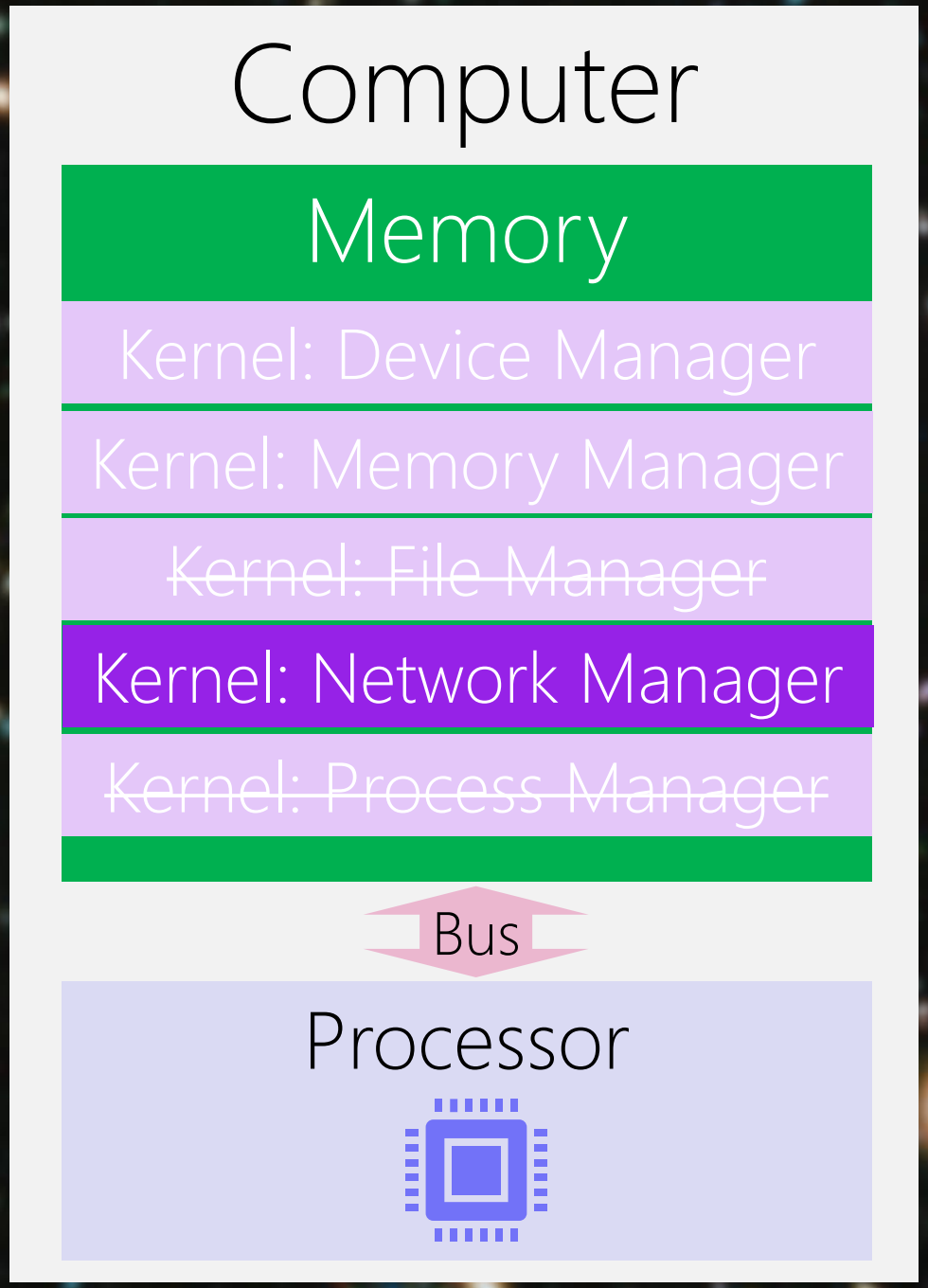
Marks will be Released Soon. Stay Tuned!



Chapter 16: Network IPC (Sockets)


UNIX
Network Manager
Machine

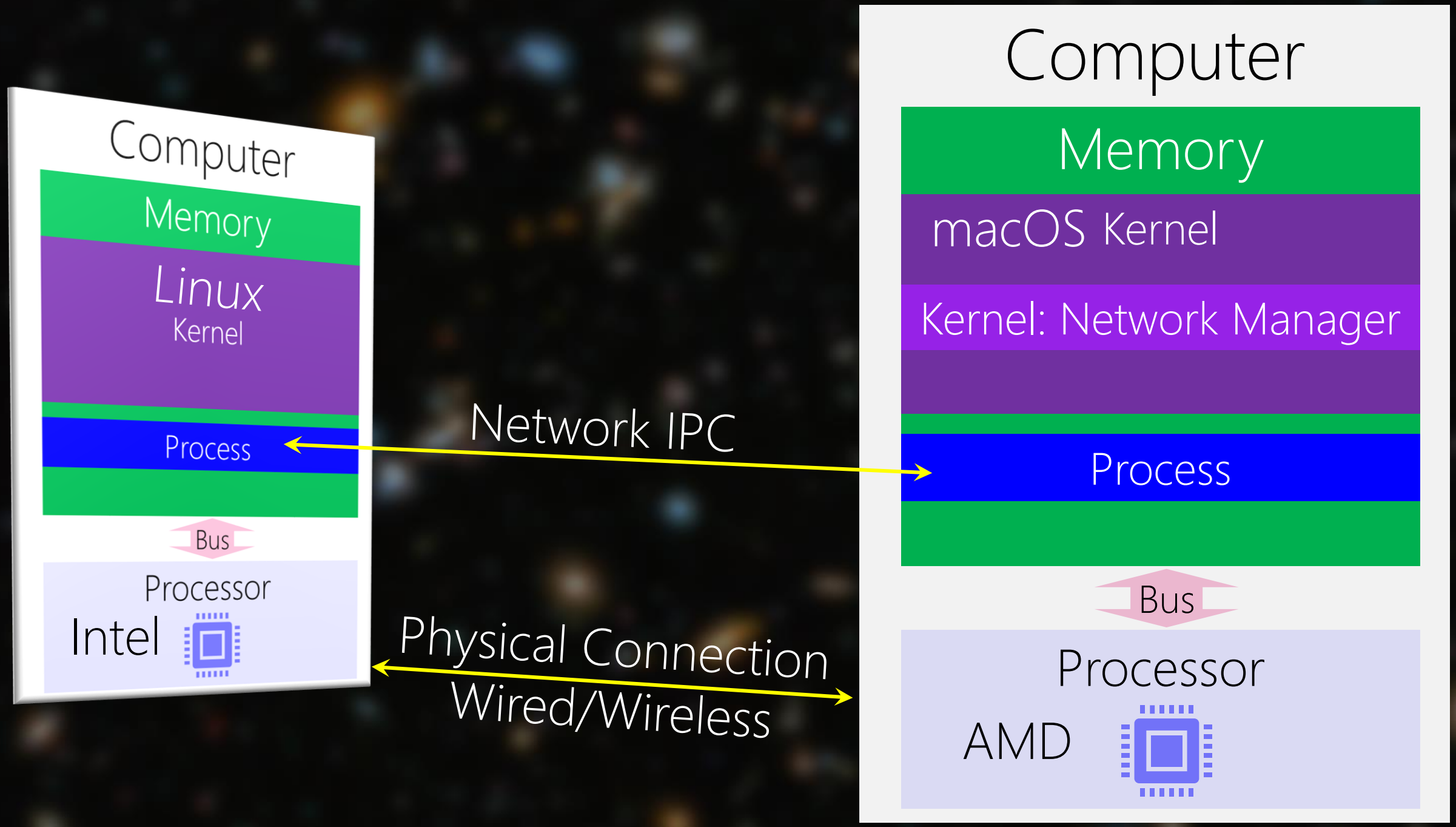

Kernel
Network Manager
Computer

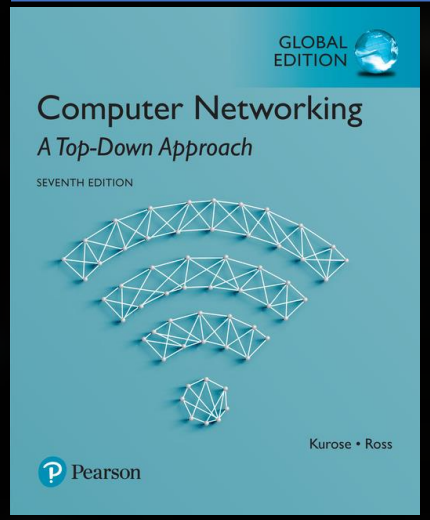

Computer
Memory
Kernel: Device Manager
Kernel: Memory Manager
~~Kernel: File Manager~~
Kernel: Network Manager
~~Kernel: Process Manager~~
Bus
Processor

Multi~~processing~~ Computers

aka Computer Network

Multiple Single Processor ~~Multiprocessor~~


Computer
Memory
Linux
Kernel
Process
Bus
Processor
Intel
Network IPC
Physical Connection
Wired/Wireless
Computer
Memory
macOS Kernel
Kernel: Network Manager
Process
Bus
Processor
AMD



COMP3670: Computer Networks

TCP/IP

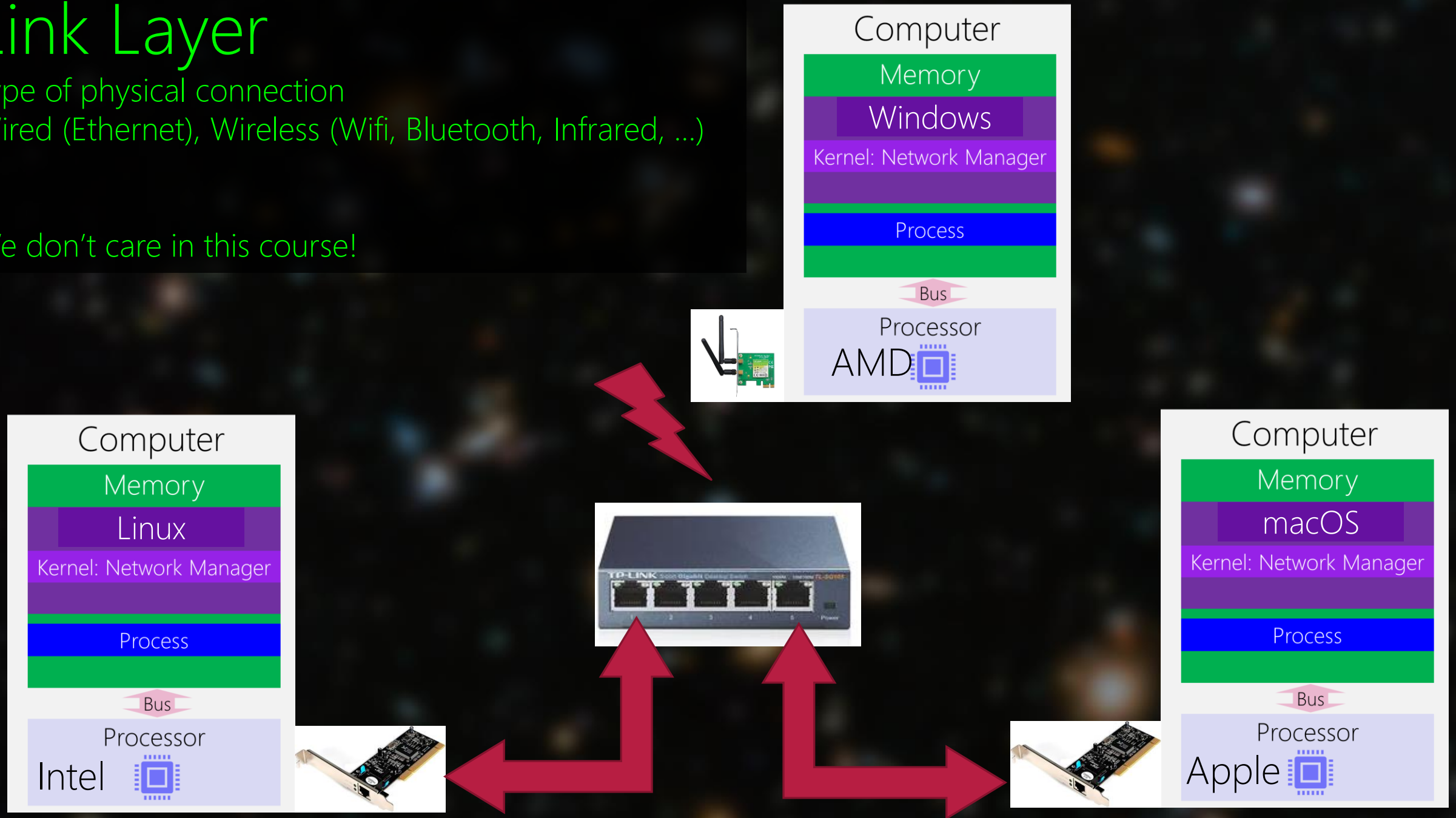
There are other network protocols!
TCP/IP is just a name. It does not represent all this protocol offers!

Link Layer

Type of physical connection

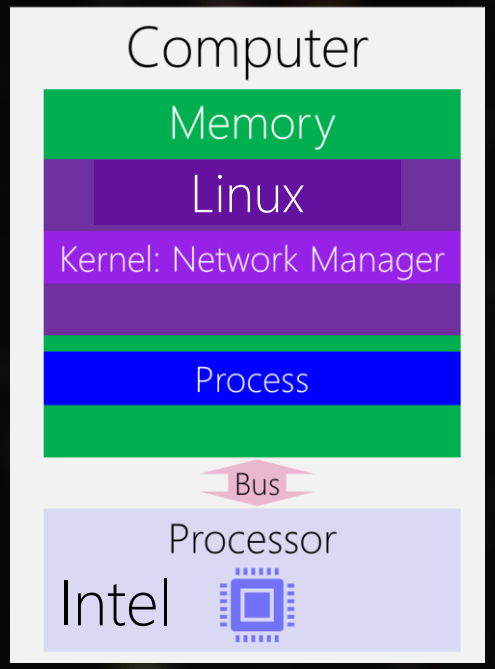
Wired (Ethernet), Wireless (Wifi, Bluetooth, Infrared, ...)

We don't care in this course!

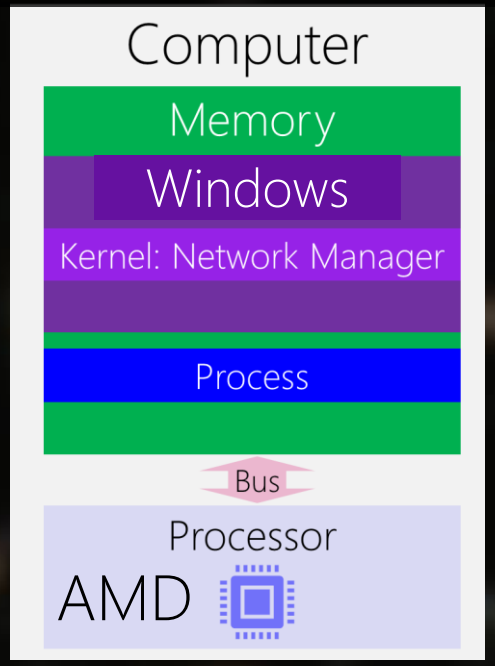


Inter-Network → Internet (Network) Layer → Internet Protocol (IP)

Computers' Address, Names, ...
We use the addresses in this course.
We don't care about the rest.
Why the format is like this?
Who assigns the addresses?
...

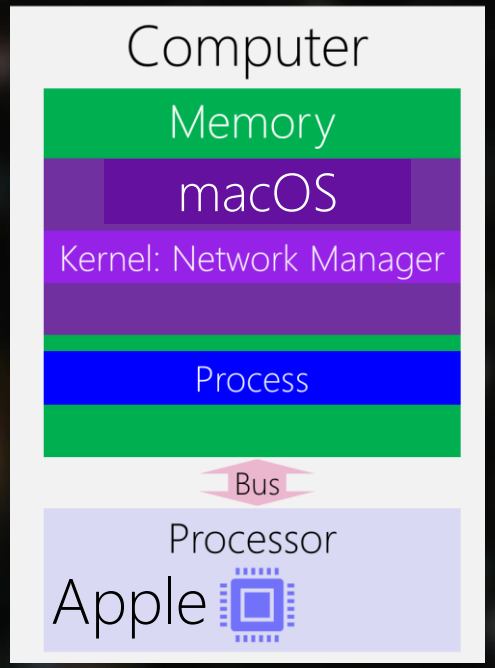


137.207.82.52



4.2.2.2

137.207.140.134



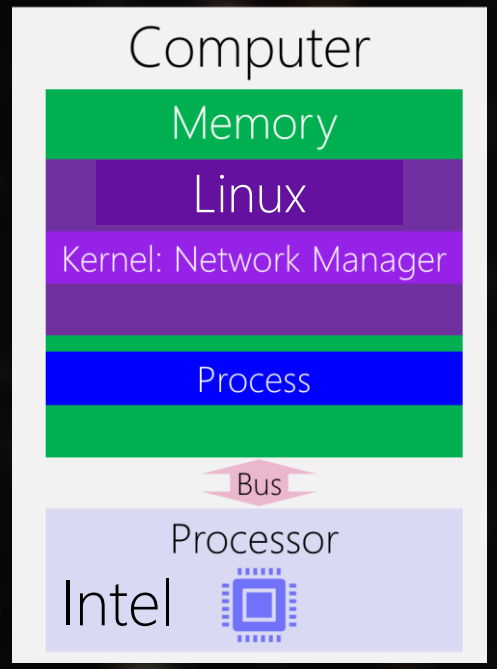
Transport Layer

Agreement on communication protocol

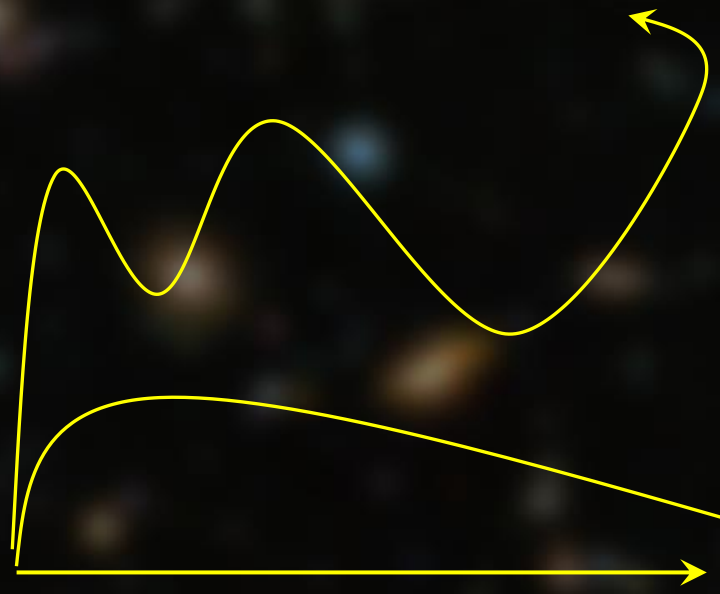
1) Connectionless == Sending a mail

User Datagram Protocol (UDP)

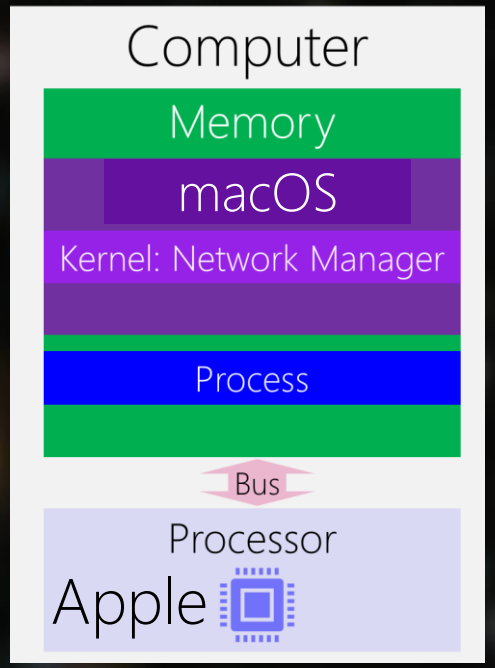
- No order (a mail may be sent sooner, but received later)
- No reliability (non-tracking mail.) Cannot see whether it is received or lost
- Each message is self-contained (Does not depends on previous or next mails)
- Simple and light (no overhead for sender, like PR card by government of Canada)



137.207.82.52



137.207.140.134



Transport Layer

Agreement on communication protocol

2) Connection-Oriented == Phone Call

Transmission Control Protocol (TCP)

- Foremost setup a connection to make sure there is a receiver ready
- Ordered (when you talk on the phone, the words are transferred in order)
- Reliability (there is an active listener)
- Each *packet* depends on previous or next packets
- Connection overhead for sender

Application Layer

Any process that wants to communicate via the network

TCP/IP

Just a name for [Link | Internet | Transport | Application] network protocol

TCP/IP

Just a name for [Link | Internet | Transport | Application] network protocol

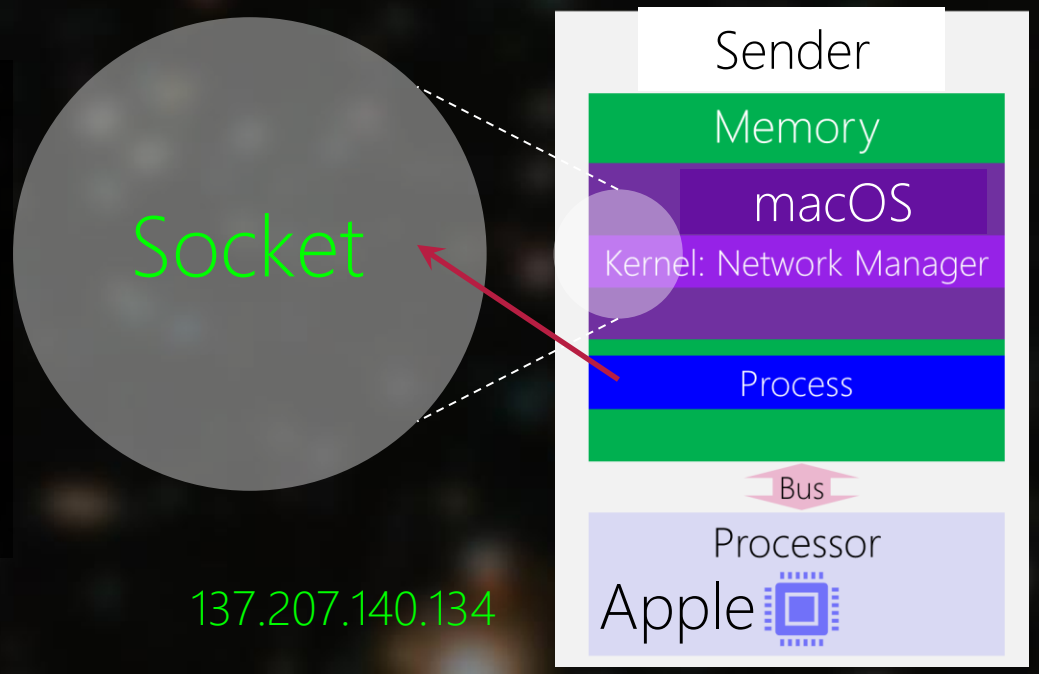
Socket Programming

TCP/IP: UDP

TCP/IP: UDP at Sender

Connectionless Communication
Sending a mail

1) Creating Socket
2) Binding to an Address (Optional)
3) Find the Receiver's Address
4) Send the Mail to the Receiver



```
hfani@bravo:~$ ./sender
socket has created for sender with sd:3
sender bound to the address:port = -2037592183 :53255
a mail has sent to the receiver at address:port = 877842313:53511
the content of the mail is <a 10 percent promotion for candian tire!>
```

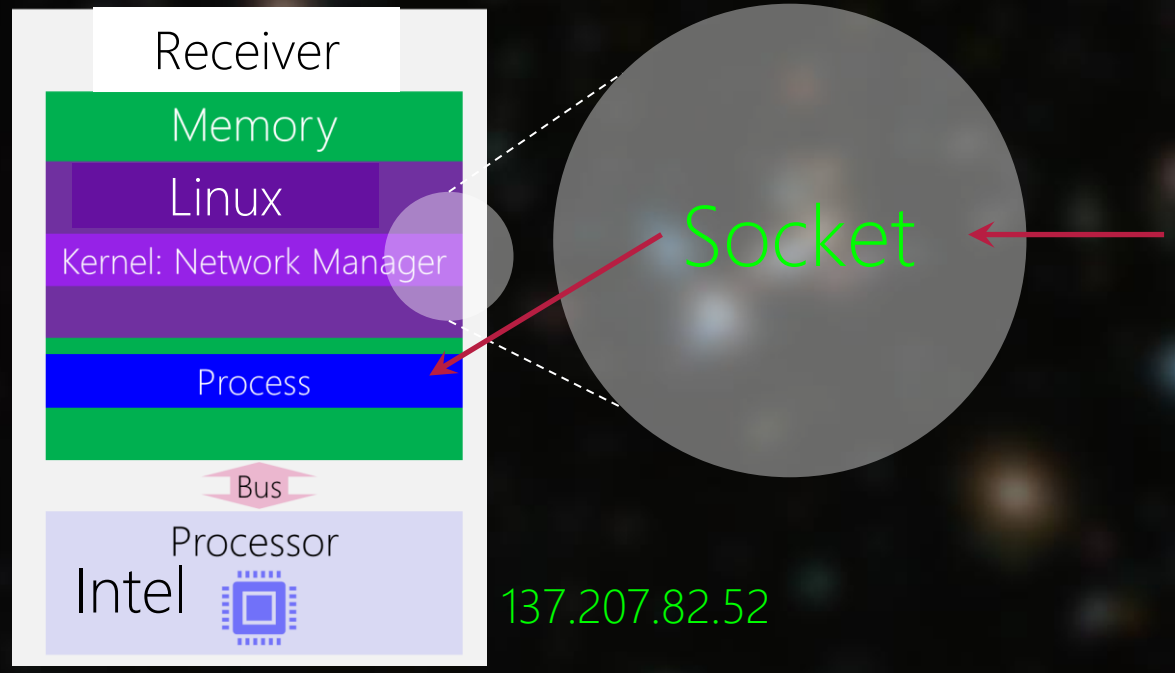
But there no receiver!
What happen to the mail?!

TCP/IP

Just a name for [Link | Internet | Transport | Application] network protocol

TCP/IP: UDP at Receiver

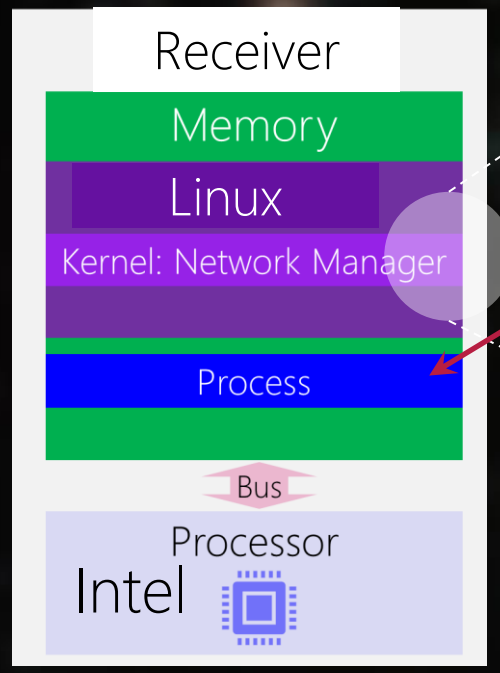
Connectionless Communication
Sending a mail

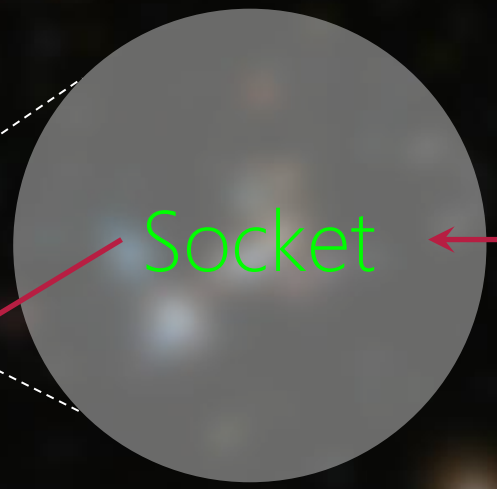


1) Creating Socket
2) Binding to an Address (MUST)
3) Wait to receive a mail
4) Find the Sender's Address (Optional)
5) Read the Mail from the Sender

TCP/IP: UDP at Receiver

Connectionless Communication
Sending a mail





137.207.82.52

1) Creating Socket

```
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
```

Kernel's Socket-related System Calls

TCP/IP Network Protocol

```c
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>

#include <stdio.h>
#include <string.h>
int main(void){
        int domain = AF_INET;//Network Protocol: TCP/IP
        int type = SOCK_DGRAM;//Connectionless
        int protocol = 0;//Default transport: UDP for Internet connectionless
```

Set up the type of network communication

```c
int receiver_sd;//socket descriptor ~= file descriptor
receiver_sd = socket(domain, type, protocol);
if (receiver_sd == -1){
        printf("error in creating socket!\n");
        exit(1);
}
else
        printf("socket has created for receiver with sd:%d\n", receiver_sd);
```

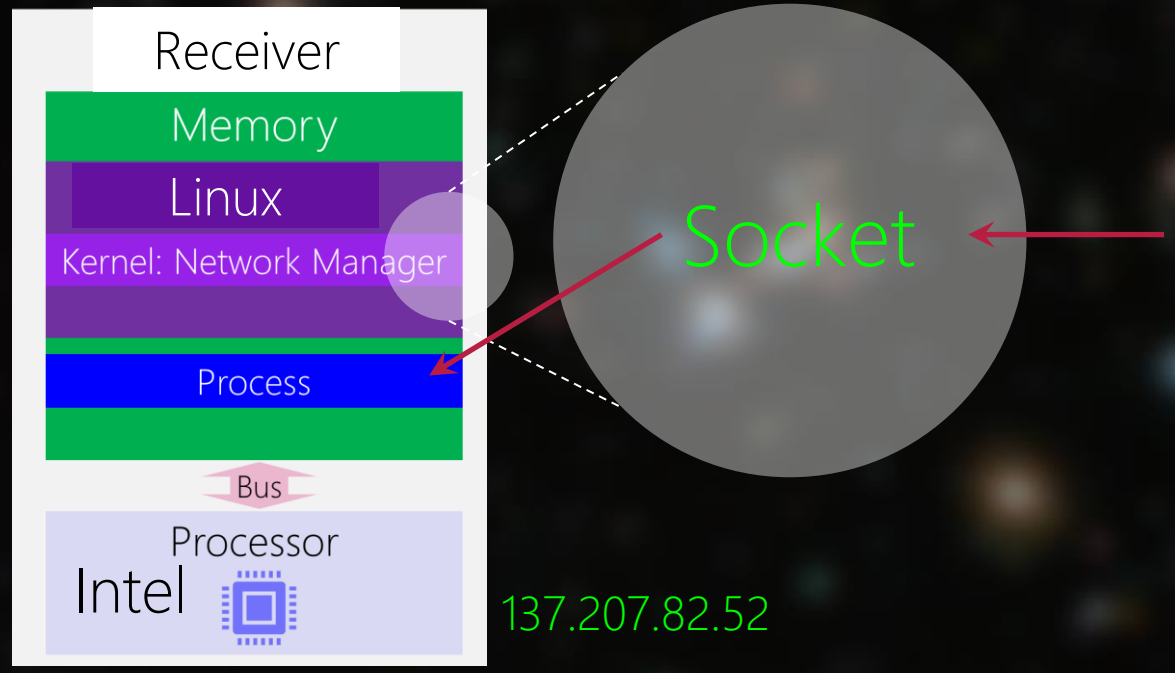
Open a socket and receive a socket descriptor

Very similar to `open()` a file and file descriptor

Indeed, behind the scene, there are implemented very similar!

TCP/IP: UDP at Receiver

Connectionless Communication
Sending a mail



1) Creating Socket
2) Binding to an Address (MUST)

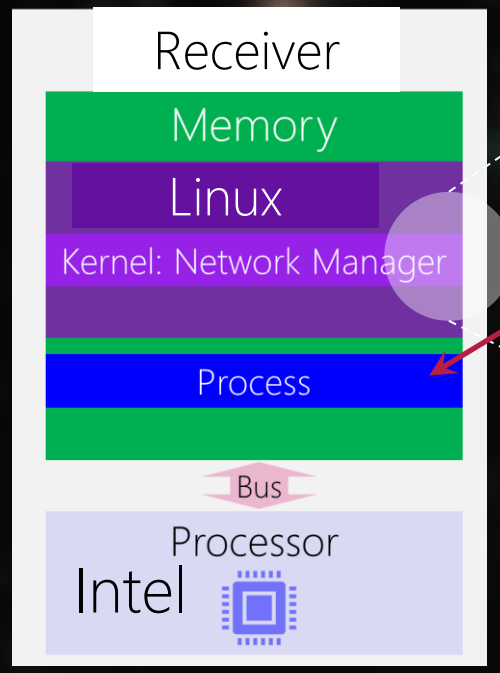
```
//Binding to an address is a must for receiver!
struct in_addr receiver_sin_address;
receiver_sin_address.s_addr = inet_addr("137.207.82.52");//nslookup `hostname`
int receiver_sin_port = htons(2001);//larger than 1024

struct sockaddr_in receiver_sin;
receiver_sin.sin_family = domain;
receiver_sin.sin_addr = receiver_sin_address;
receiver_sin.sin_port = receiver_sin_port;
int result = bind(receiver_sd, (struct sockaddr *) &receiver_sin, sizeof(receiver_sin));
if (result == -1){
        printf("error in binding receiver to the address:port = %d:%d\n", receiver_sin.sin_addr, rece
        exit(1);
}
else
        printf("receiver bound to the address:port = %d:%d\n", receiver_sin.sin_addr, receiver_sin.si
```

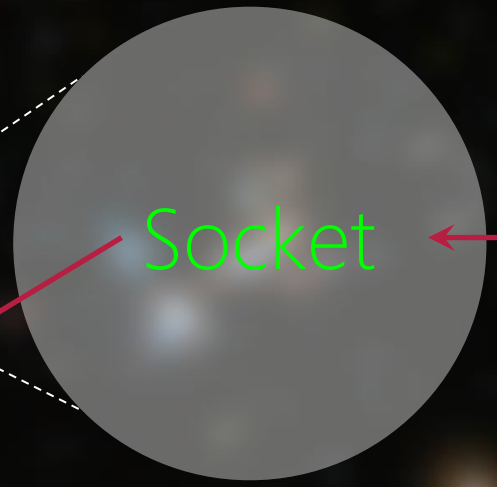
Very similar to the sender's binding of socket to IP:PORT

But for receiver it is a MUST. Why?

TCP/IP: UDP at Receiver

Connectionless Communication
Sending a mail





137.207.82.52

1) Creating Socket
2) Binding to an Address (MUST)
3) Wait to receive a mail

```
#include <sys/socket.h>
ssize_t recvfrom(int sockfd, void *restrict buf, size_t len, int flags, struct sockaddr *restrict addr, socklen_t *restrict addrlen);
```

Returns length of message in bytes, –1 on error

```
//wait for a message ...
struct sockaddr_in sender_sin;//I want to know who send the message
char mailbox[100];
int sender_sin_len;
setbuf(stdout, NULL);
while(1)
{
        result = recvfrom(receiver_sd, mailbox, sizeof(mailbox), 0, (struct sockaddr *) &sender_sin,
        if (result == -1){
                printf("error in opening mail from sender!\n");
                exit(1);
        }
        else
                printf("the content of mail is: %s", mailbox);
}

}
```

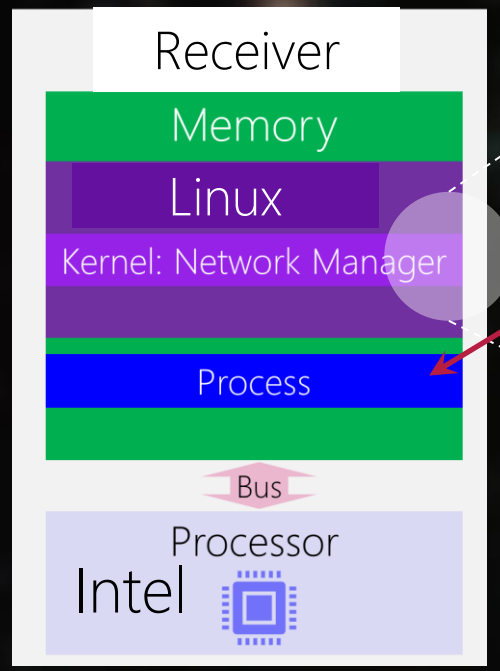
Wait to receive a mail at socket

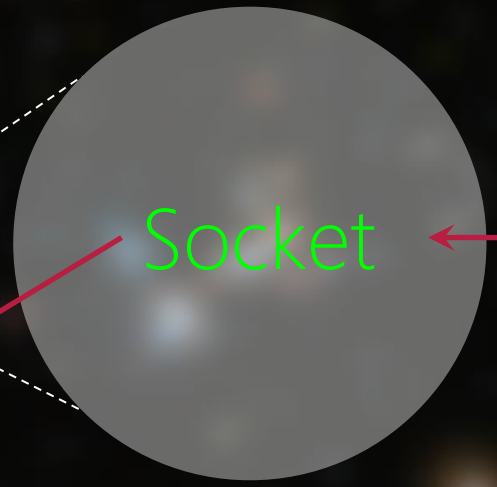
Very similar to `read()` from a file

It is a blocking call! It sleeps until a new mail

TCP/IP: UDP at Receiver

Connectionless Communication
Sending a mail





137.207.82.52

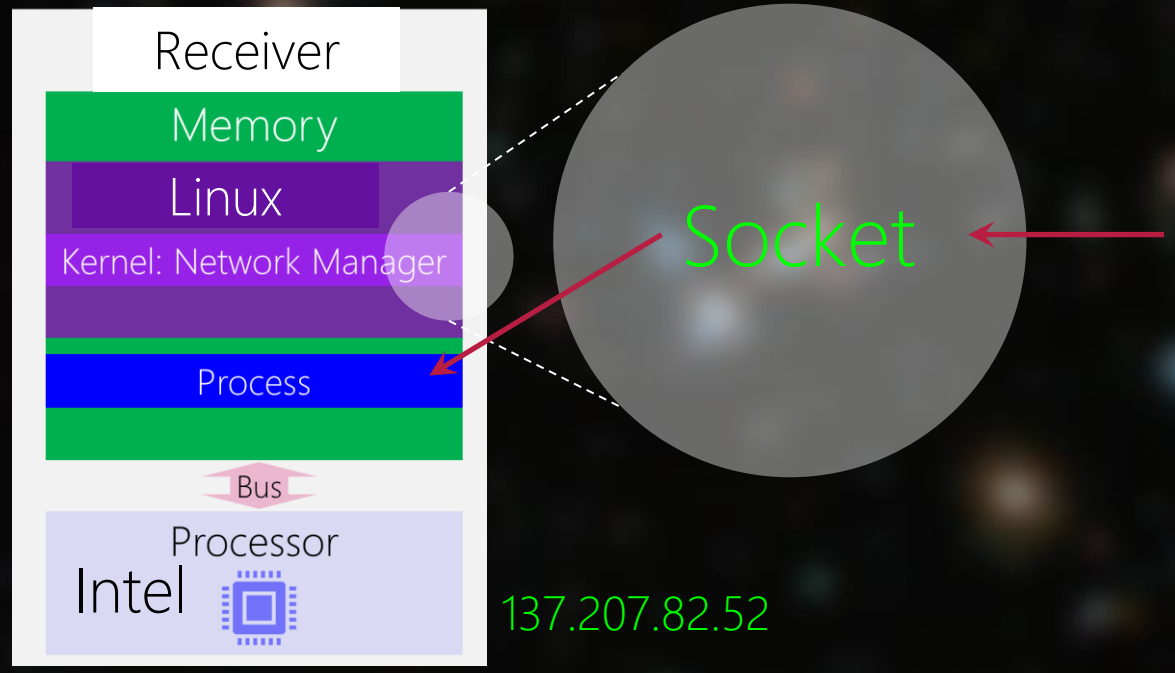
1) Creating Socket
2) Binding to an Address (MUST)
3) Wait to receive a mail
4) Find the Sender's Address (Optional)

```
//wait for a message ...
struct sockaddr_in sender_sin;//I want to know who send the message
char mailbox[100];
int sender_sin_len;
setbuf(stdout, NULL);
while(1)
{
        result = recvfrom(receiver_sd, mailbox, sizeof(mailbox), 0, (struct sockaddr *) &sender_sin,
        if (result == -1){
                printf("error in opening mail from sender!\n");
                exit(1);
        }
        else
                printf("the content of mail is: %s", mailbox);
}

}
```

You can ignore but you can know who is the sender and decide Sender's IP:PORT

TCP/IP: UDP at Receiver

Connectionless Communication
Sending a mail



1) Creating Socket
2) Binding to an Address (MUST)
3) Wait to receive a mail
4) Find the Sender's Address (Optional)
5) Read the Mail from the Sender

```
//wait for a message ...
struct sockaddr_in sender_sin;//I want to know who send the message
char mailbox[100];
int sender_sin_len;
setbuf(stdout, NULL);
while(1)
{
        result = recvfrom(receiver_sd, mailbox, sizeof(mailbox), 0, (struct sockaddr *) &sender_sin,
        if (result == -1){
                printf("error in opening mail from sender!\n");
                exit(1);
        }
        else
                printf("the content of mail is: %s", mailbox);
}

}
```

Like the `read()` buffer for file

```
//wait for a message ...
struct sockaddr_in sender_sin;//I want to know who send the message
char mailbox[100];
int sender_sin_len;
setbuf(stdout, NULL);
while(1)
{
        result = recvfrom(receiver_sd, mailbox, sizeof(mailbox), 0, (struct sockaddr *) &sender_sin,
        if (result == -1){
                printf("error in opening mail from sender!\n");
                exit(1);
        }
        else
                printf("the content of mail is: %s", mailbox);
}

}
```

Usually, receiver never dies. It is waiting to receive a new mail forever.

A better way to avoid receiver process waste time on waiting would be?

```
//wait for a message ...
struct sockaddr_in sender_sin;//I want to know who send the message
char mailbox[100];
int sender_sin_len;
setbuf(stdout, NULL);
while(1)
{
        result = recvfrom(receiver_sd, mailbox, sizeof(mailbox), 0, (struct sockaddr *) &sender_sin,
        if (result == -1){
                printf("error in opening mail from sender!\n");
                exit(1);
        }
        else
                printf("the content of mail is: %s", mailbox);
}

}
```

`fork()` and give the task of reading mails to the child!
The receiver then does other important tasks.

hfani@bravo: ~

```
hfani@bravo:~$ ./receiver
socket has created for receiver with sd:3
receiver bound to the address:port = 877842313:53511
```

/cygdrive/c

```
Administrator@hfani /cygdrive/c
$ ./sender
```

hfani@bravo: ~

```
hfani@bravo:~$ ./receiver
socket has created for receiver with sd:3
receiver bound to the address:port = 877842313:53511
the content of mail is: a 10 percent promotion for candian tire!
```

/cygdrive/c

```
Administrator@hfani /cygdrive/c
$ ./sender
socket has created for sender with sd:3
sender bound to the address:port = -2037592183:53255
a mail has sent to the receiver at address:port = 877842313:53511
the content of the mail is <a 10 percent promotion for candian tire!>

Administrator@hfani /cygdrive/c
$
```

hfani@bravo: ~

```
hfani@bravo:~$ ./receiver
socket has created for receiver with sd:3
receiver bound to the address:port = 877842313:53511
the content of mail is: a 10 percent promotion for candian tire!the content of m
ail is: a 10 percent promotion for candian tire!
```

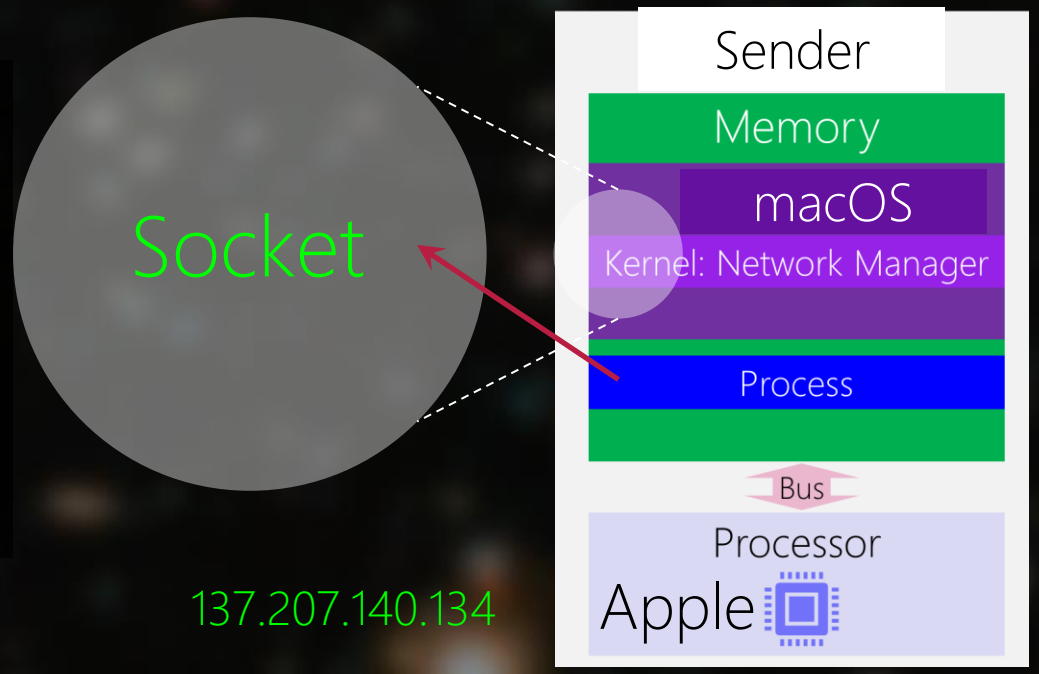
/cygdrive/c

```
Administrator@hfani /cygdrive/c
$ ./sender
socket has created for sender with sd:3
sender bound to the address:port = -2037592183:53255
a mail has sent to the receiver at address:port = 877842313:53511
the content of the mail is <a 10 percent promotion for candian tire!>

Administrator@hfani /cygdrive/c
$ ./sender
socket has created for sender with sd:3
sender bound to the address:port = -2037592183:53255
a mail has sent to the receiver at address:port = 877842313:53511
the content of the mail is <a 10 percent promotion for candian tire!>

Administrator@hfani /cygdrive/c
$
```

TCP/IP: UDP at Sender

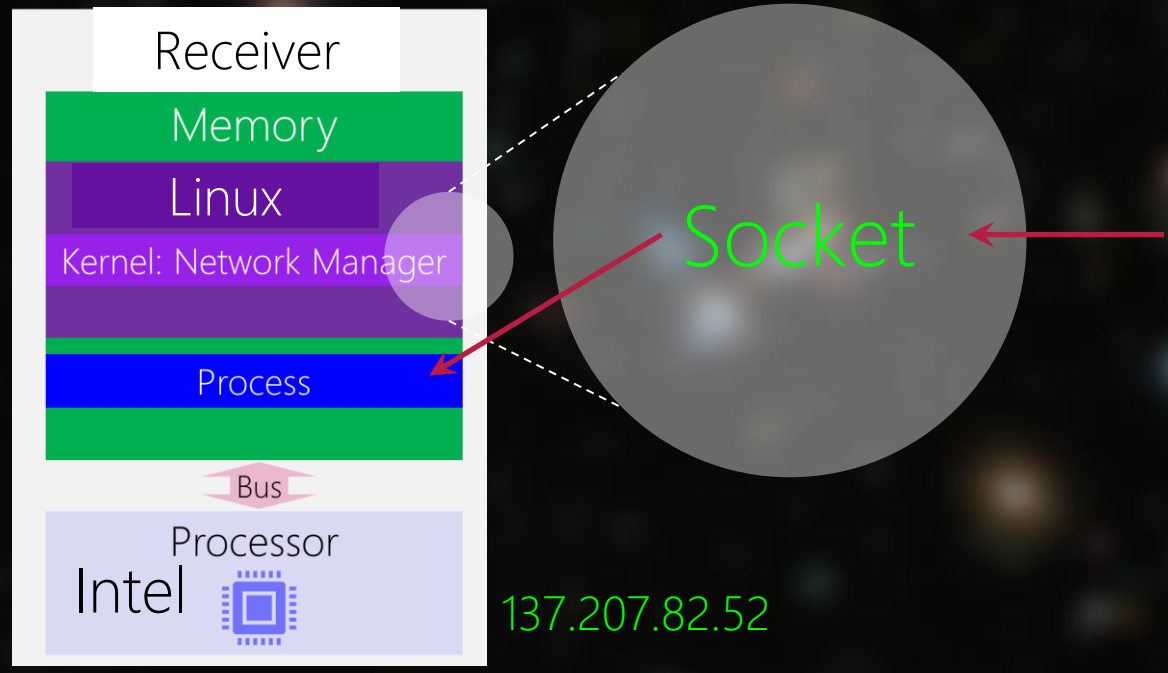
Connectionless Communication
Sending a mail

1) `socket()`
2) `bind()`
3) Receiver's Address
4) `sendto()`



TCP/IP: UDP at Receiver

Connectionless Communication
Sending a mail

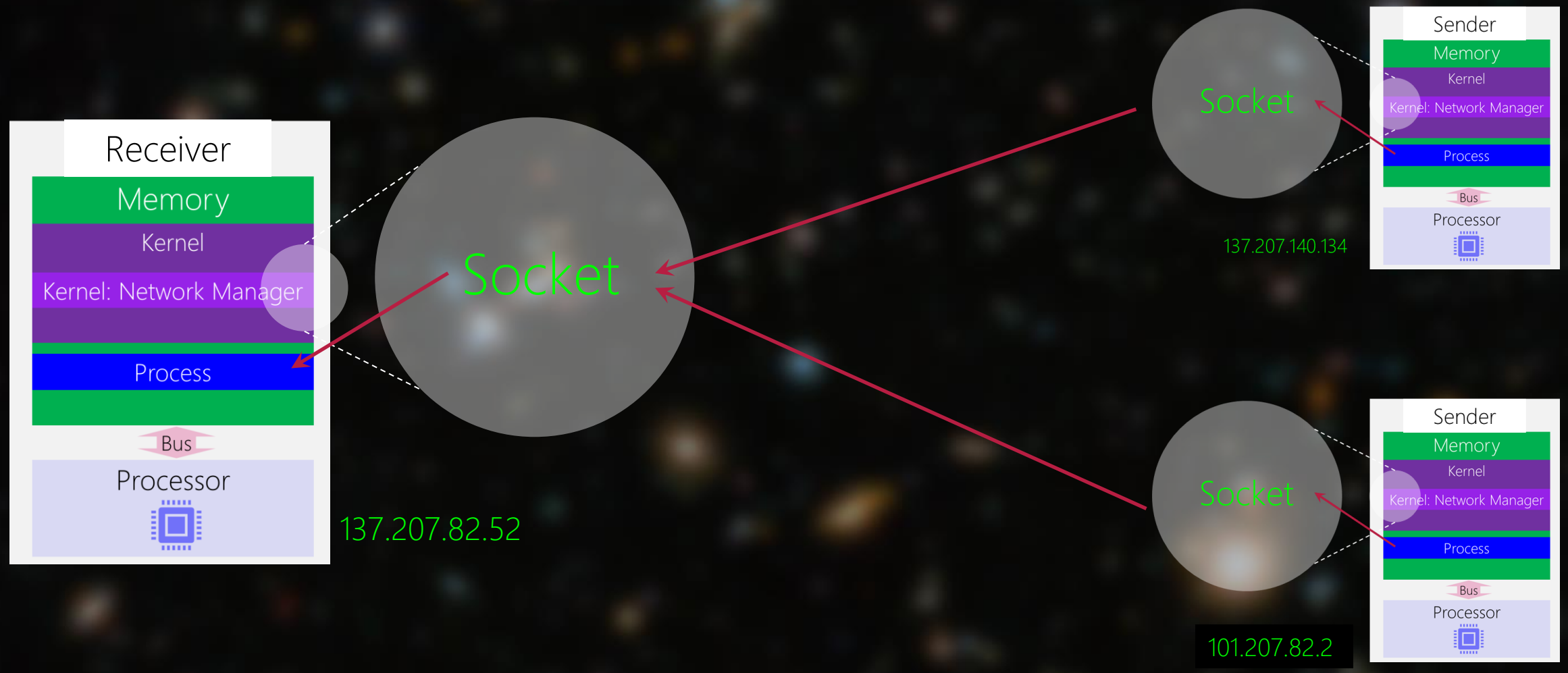


1) `socket()`
2) `bind()`
3) `recvfrom()`
4) Find the Sender's Address (Optional)
5) Read the Mail from the Sender

Why sender leave it to the kernel to handle its address?

TCP/IP : UDP

Many senders, single receiver



Is it a good practice to hardcode the IP:PORT in receiver?

Socket Programming

TCP/IP: TCP

Connection-Oriented, Reliable, Ordered

Lab 11

Philosophical Debate

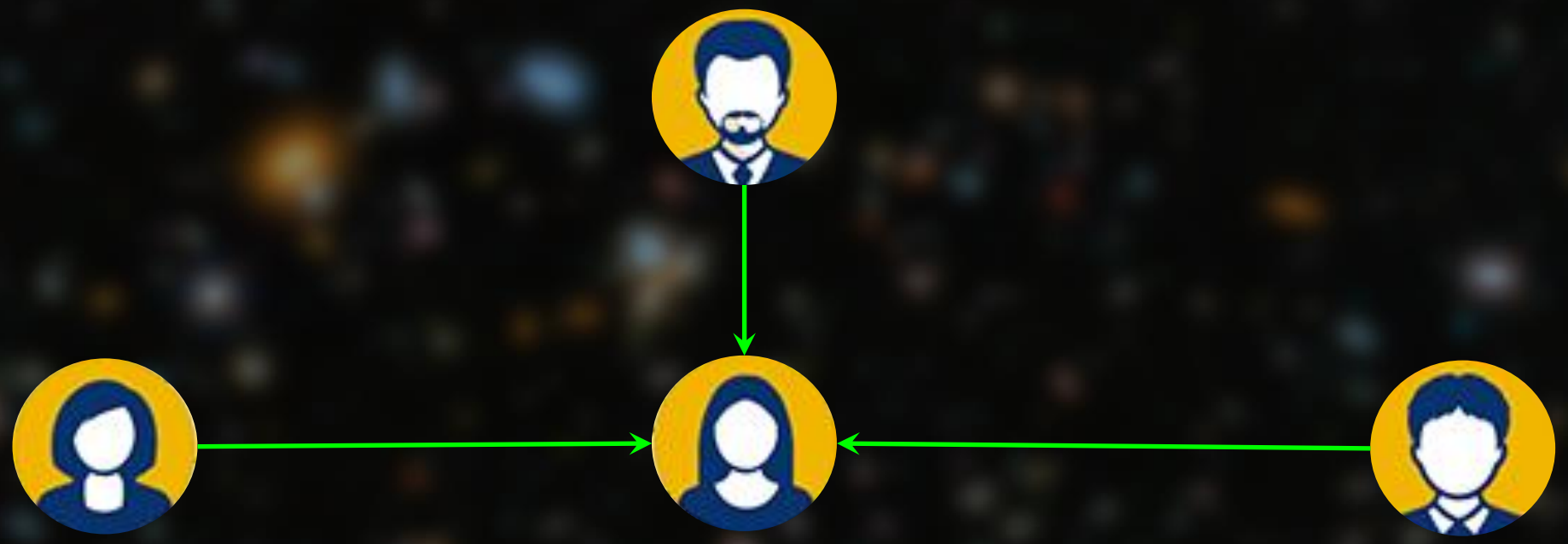
TCP/IP: TCP vs. UDP

Sender/Receiver vs. Client/Server

Any process can *initiate* a communication with other process

A contacts B

Some time later, B contacts A



A passive process! Never *initiate* a call.
Only replies if receives a call!
Never calls anybody!

The Server

Socket Programming

TCP/IP: TCP

Connection-Oriented, Reliable, Ordered

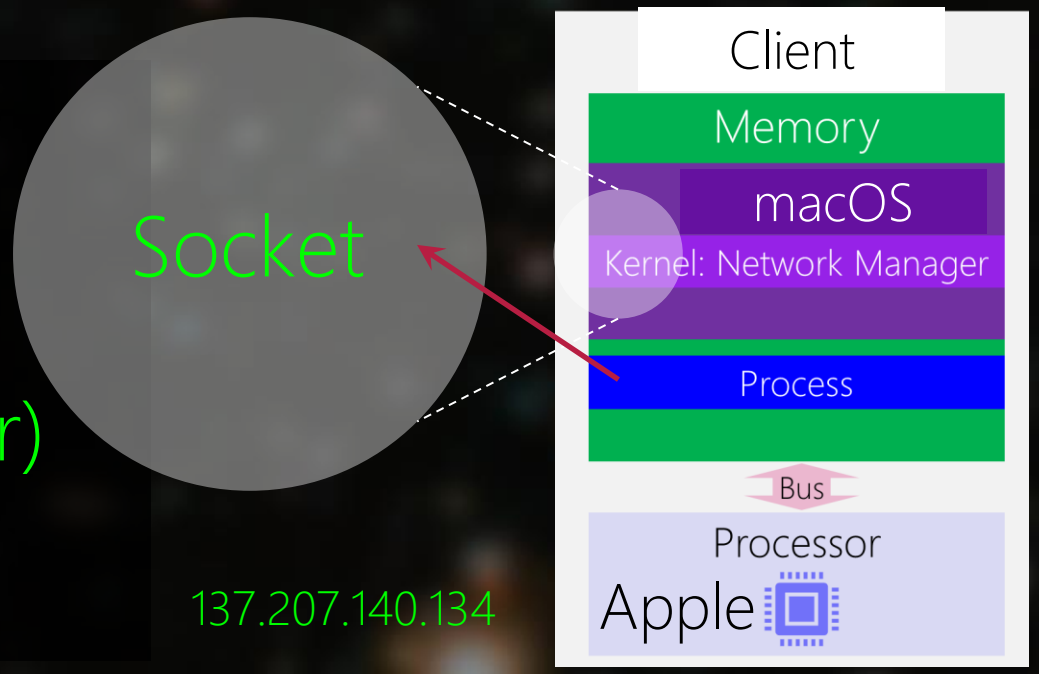
Clients call a Service Provider

TCP/IP: TCP at Clients

Connection-oriented Communication

Phone Call

1) Creating Socket
2) Binding to an Address (Optional)
3) Find The Server's Address
4) Make a Connection (Dial the Number)
5) If Connected, Communicate



137.207.140.134

TCP/IP: TCP at Client

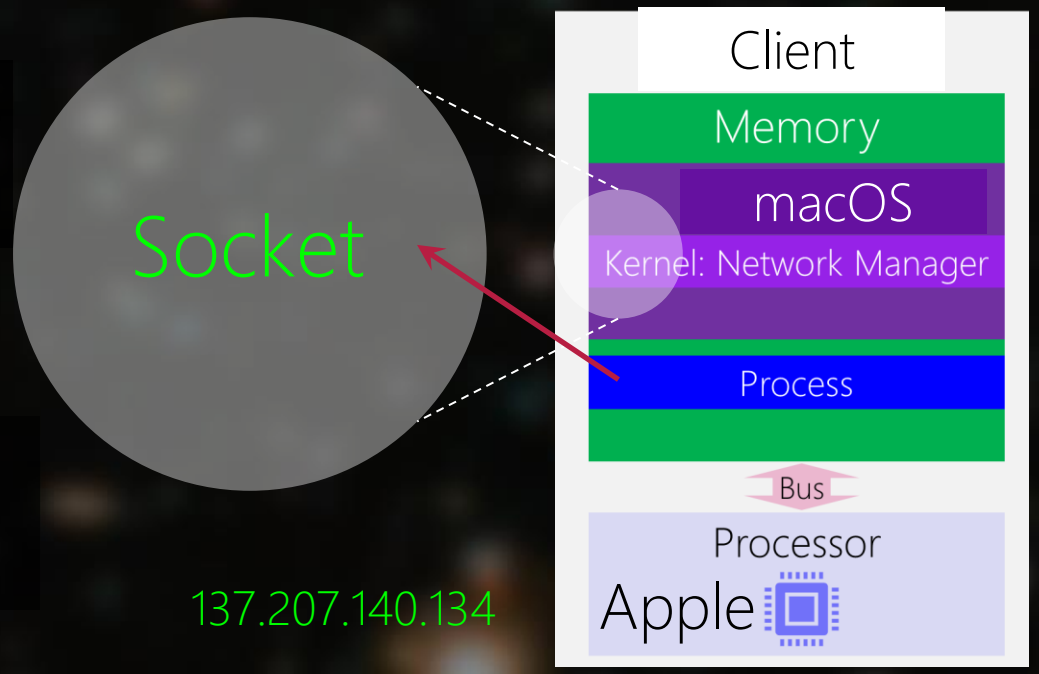
Connection-oriented Communication

Phone Call

1) Creating Socket

```
#include <sys/socket.h>
int socket(int domain, int type, int protocol);
```

Returns socket descriptor if OK, –1 on error



```
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
```

Kernel's Socket-related System Calls

TCP/IP Network Protocol

Set up the type of network communication

```c
#include <stdlib.h>

#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>

#include <stdio.h>
#include <string.h>
int main(void){
        int domain = AF_INET;//Network Protocol: TCP/IP
        int type = SOCK_STREAM;//Connection-Oriented
        int protocol = 0;//Default transport: TCP for Internet connection-oriented

        int client_sd;//socket descriptor ~= file descriptor
        client_sd = socket(domain, type, protocol);
        if (client_sd == -1){
                printf("error in creating socket for the client!\n");
                exit(1);
        }
        else
                printf("socket has created for the client with sd:%d\n", client_sd);
```

Domain	Description
AF_INET	IPv4 Internet domain
AF_INET6	IPv6 Internet domain (optional in POSIX.1)
AF_UNIX	UNIX domain
AF_UNSPEC	unspecified

Figure 16.1 Socket communication domains

Type	Description
SOCK_DGRAM	fixed-length, connectionless, unreliable messages
SOCK_RAW	datagram interface to IP (optional in POSIX.1)
SOCK_SEQPACKET	fixed-length, sequenced, reliable, connection-oriented messages
SOCK_STREAM	sequenced, reliable, bidirectional, connection-oriented byte streams

Figure 16.2 Socket types

Protocol	Description
IPPROTO_IP	IPv4 Internet Protocol
IPPROTO_IPV6	IPv6 Internet Protocol (optional in POSIX.1)
IPPROTO_ICMP	Internet Control Message Protocol
IPPROTO_RAW	Raw IP packets protocol (optional in POSIX.1)
IPPROTO_TCP	Transmission Control Protocol
IPPROTO_UDP	User Datagram Protocol

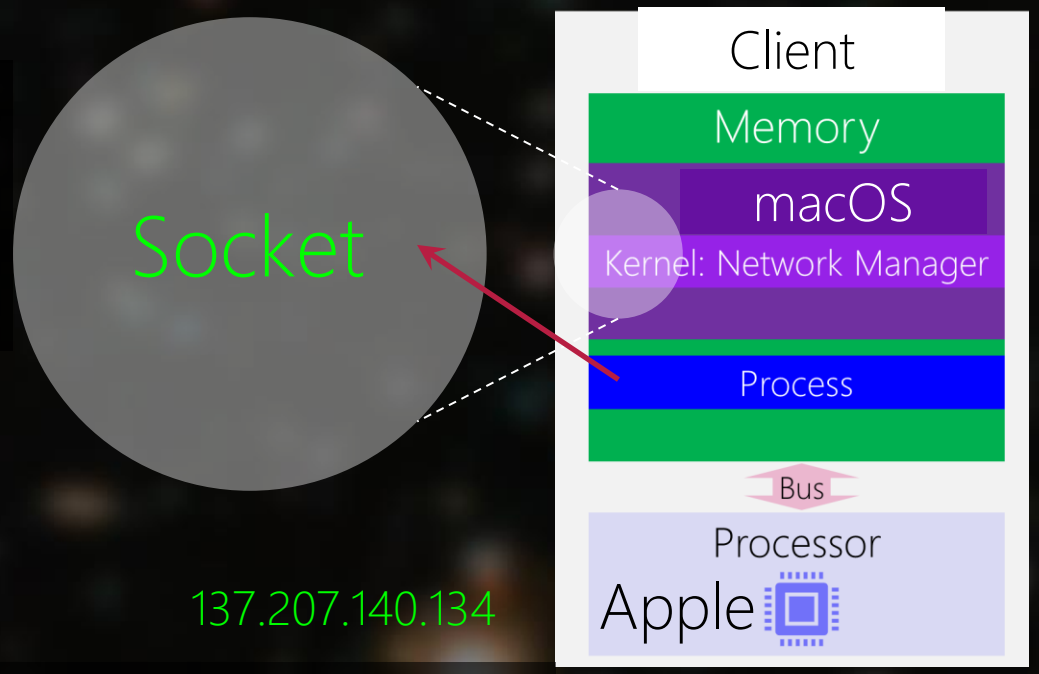
Figure 16.3 Protocols defined for Internet domain sockets

TCP/IP: TCP at Client

Connection-oriented Communication

Phone Call

1) Creating Socket
2) Binding to an Address (Optional)




```
#include <sys/socket.h>
int bind(int sockfd, const struct sockaddr *addr, socklen_t len);
```

Returns 0 if OK, −1 on error

Leave it to the *kernel* of client to handle IP:PORT

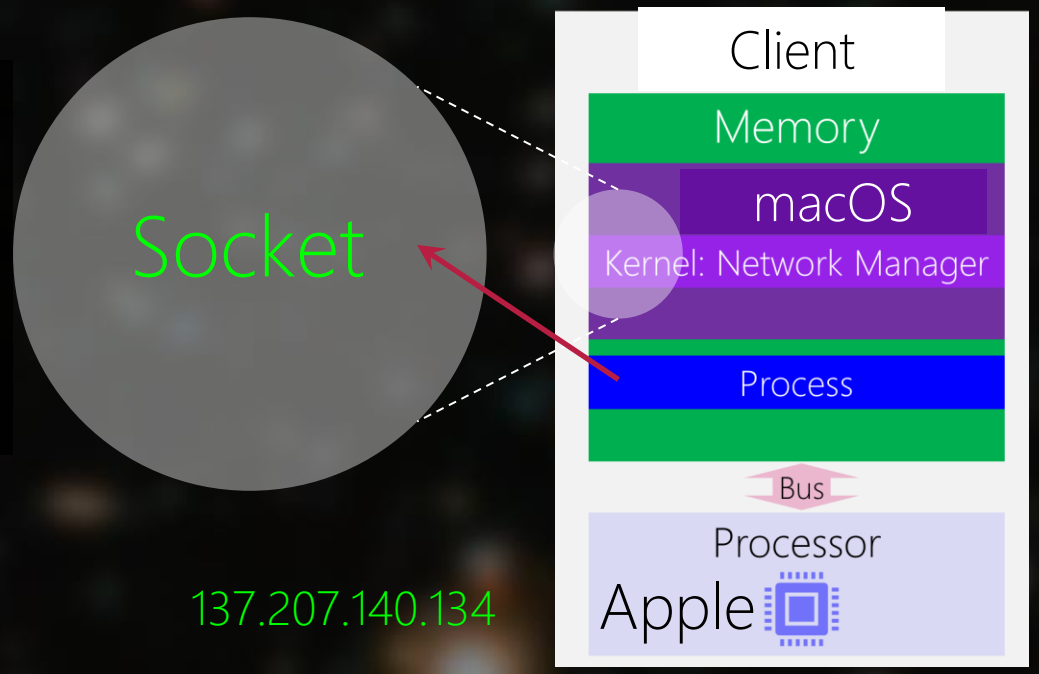
If interested to bind an IP:PORT explicitly, look at UDP slides at Sender!

TCP/IP: TCP at Client

Connection-oriented Communication

Phone Call

1) Creating Socket
2) Binding to an Address (Optional)
3) Find The Server's Address



```c
struct in_addr server_sin_address;
server_sin_address.s_addr = inet_addr("137.207.82.52");//ask!
int server_sin_port = htons(2021);//larger than 1024

struct sockaddr_in server_sin;
server_sin.sin_family = domain;
server_sin.sin_addr = server_sin_address;
server_sin.sin_port = server_sin_port;
```

IP:PORT of The Server

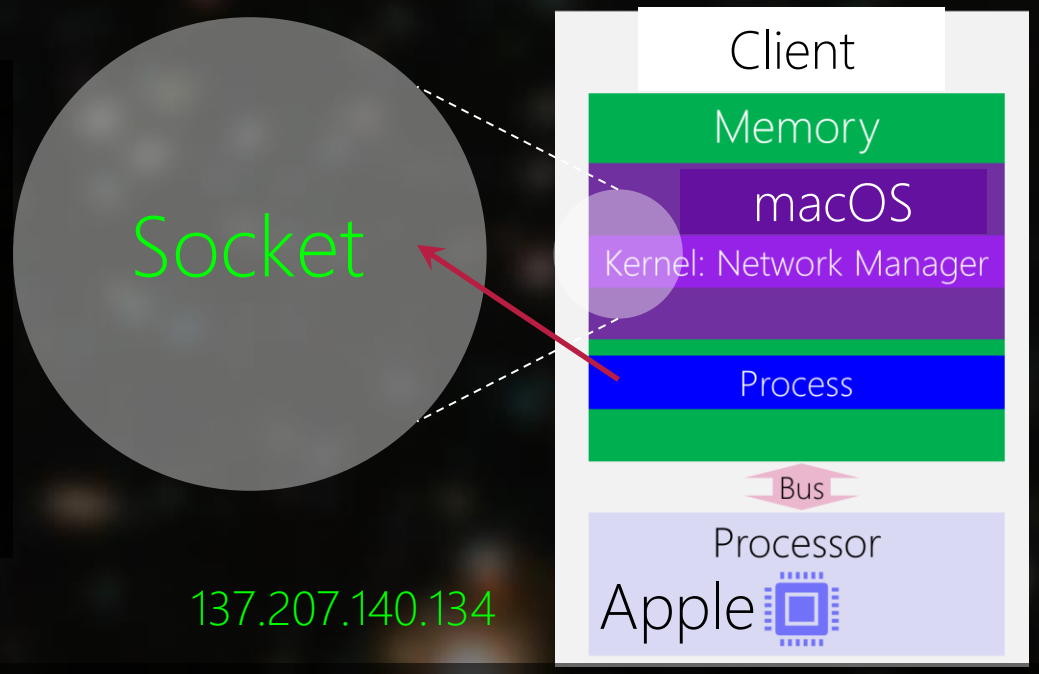
It is unique all around the world! Why?

TCP/IP: TCP at Client

Connection-oriented Communication

Phone Call

1) Creating Socket
2) Binding to an Address (Optional)
3) Find The Server's Address
4) Make a Connection to The Server




```
#include <sys/socket.h>
int connect(int sockfd, const struct sockaddr *addr, socklen_t len);
```

Returns 0 if OK, −1 on error

```
int result = connect(client_sd, (struct sockaddr *) &server_sin, sizeof(server_sin));
if (result == -1){
        printf("error in connecting to The Server at address:port = %d:%d\n", server_sin.sin_addr, server_sin.
        exit(1);
}
else
        printf("client is connected to The Server at address:port = %d:%d\n", server_sin.sin_addr, server_sin.
```

Make the phone call
(Dialing a number!)

The Server's IP:PORT

```
hfani@alpha:~$ cc client.c -o client
hfani@alpha:~$ ./client
socket has created for the client with sd:3
error in connecting to The Server at address:port = 877842313:58375
```

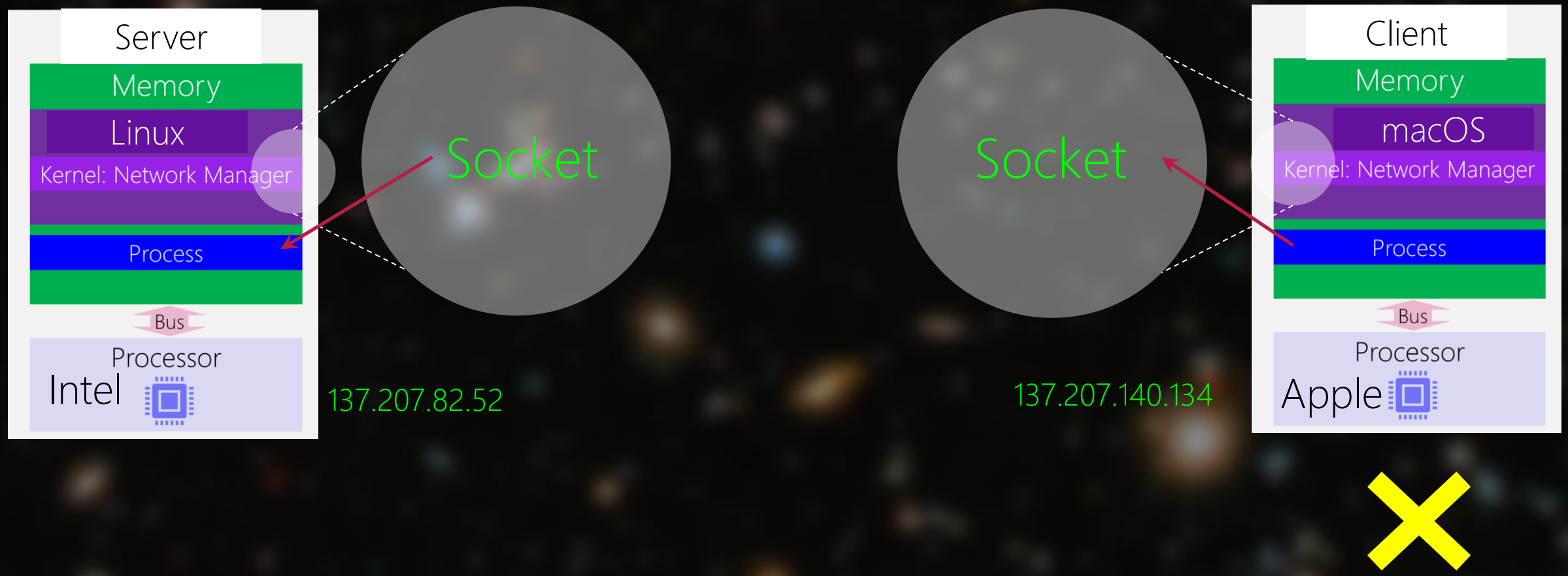
But there is no server!

If there is no connection, no communication!

We cannot move to step (5)

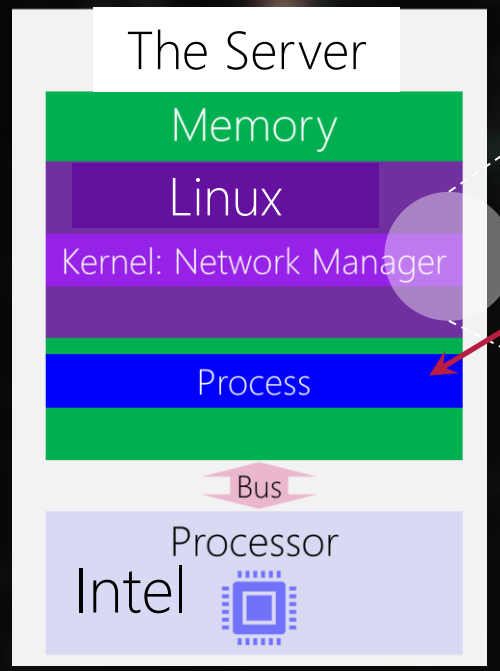
TCP/IP: TCP

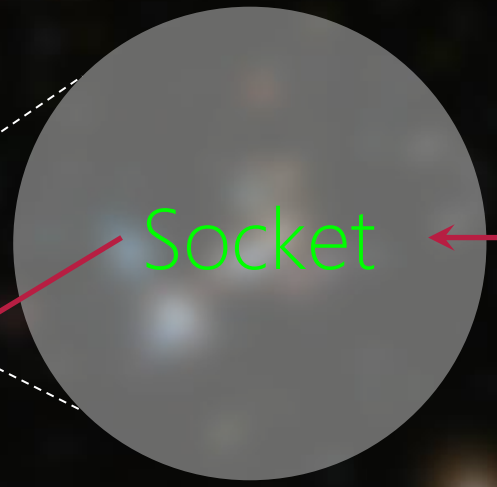
Just a name for [Link | Internet | Transport | Application] network protocol



TCP/IP: TCP at The Server

Connection-oriented Communication
Phone Call





137.207.82.52

1) Creating Socket
2) Binding to an Address (MUST)
3) Wait for Clients Phone Call
4) Accept Clients' Call
5) Communicate

Like a call center :)