Lecture 14

Today
- TCP state and connections
- TCP sending: stop and wait

Administrivia
- IP grading meetings: happening now

- TCP is out
    - Gearup I TONIGHT 5-7pm in CIT 165 => intro to project, thinking about sockets
    - Milestone I next week (sign up for meeting before spring break)
    - **Start early!!!!!**

- HW2: due next Tuesday (problem 3 is practice for TCP)

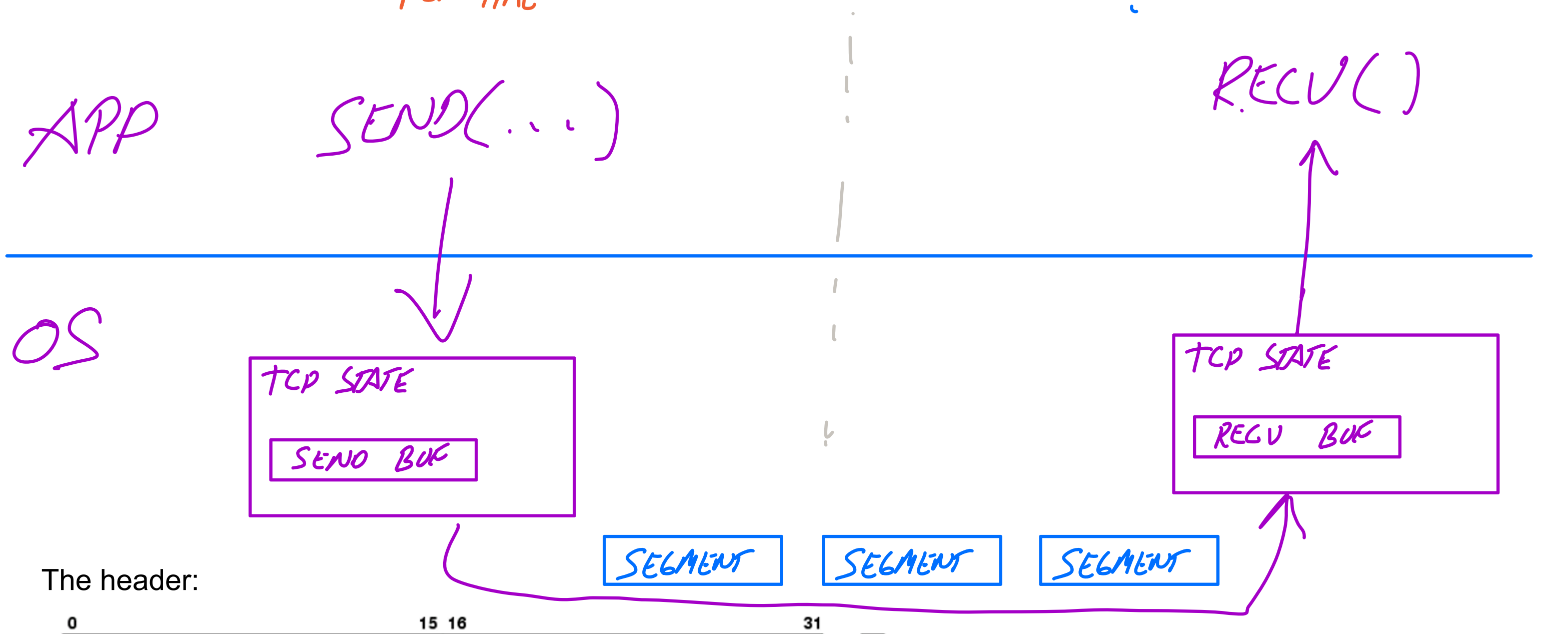
Lecture 14: TCP II

TCP: the story so far

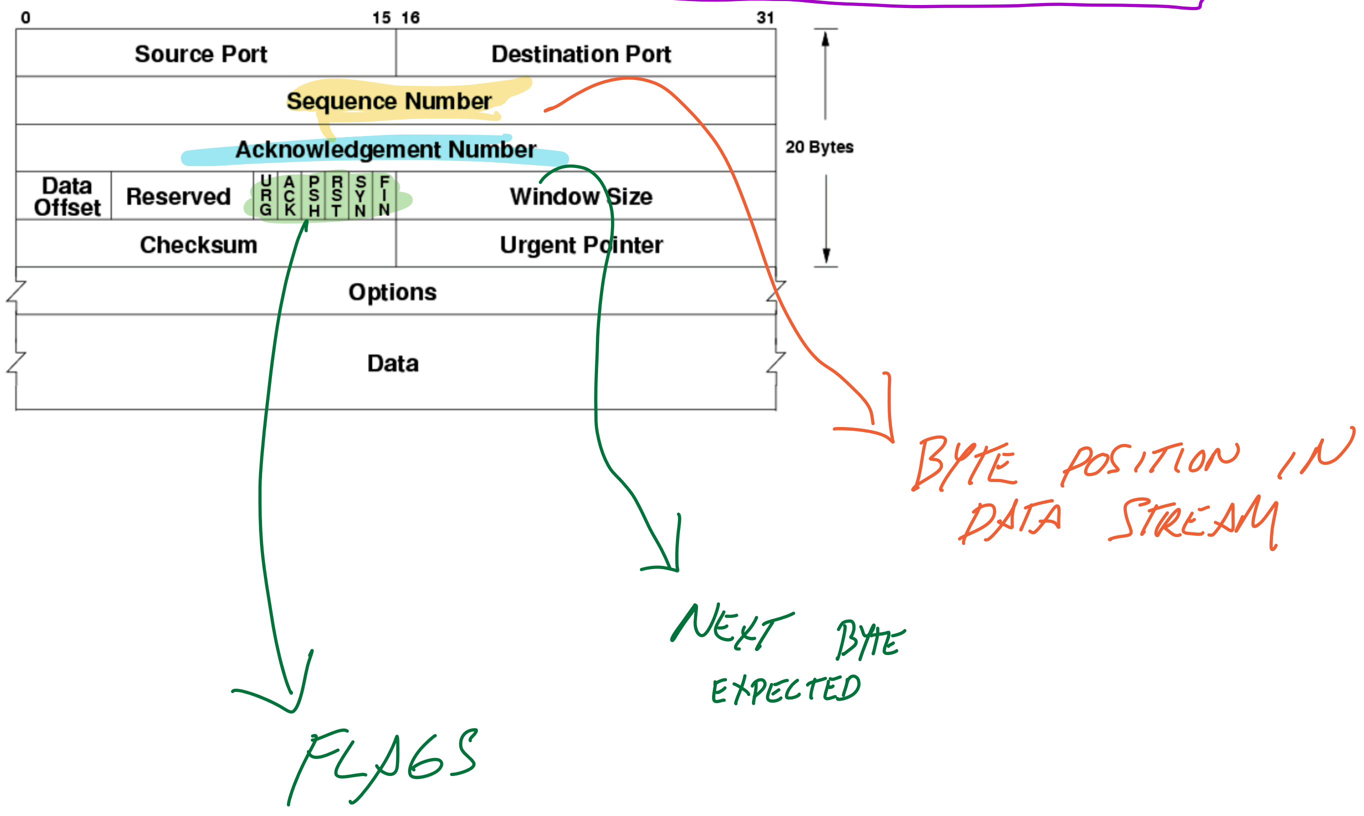
TODAY

"a reliable, connection-oriented, full duplex, ordered byte stream"

NEXT TIME



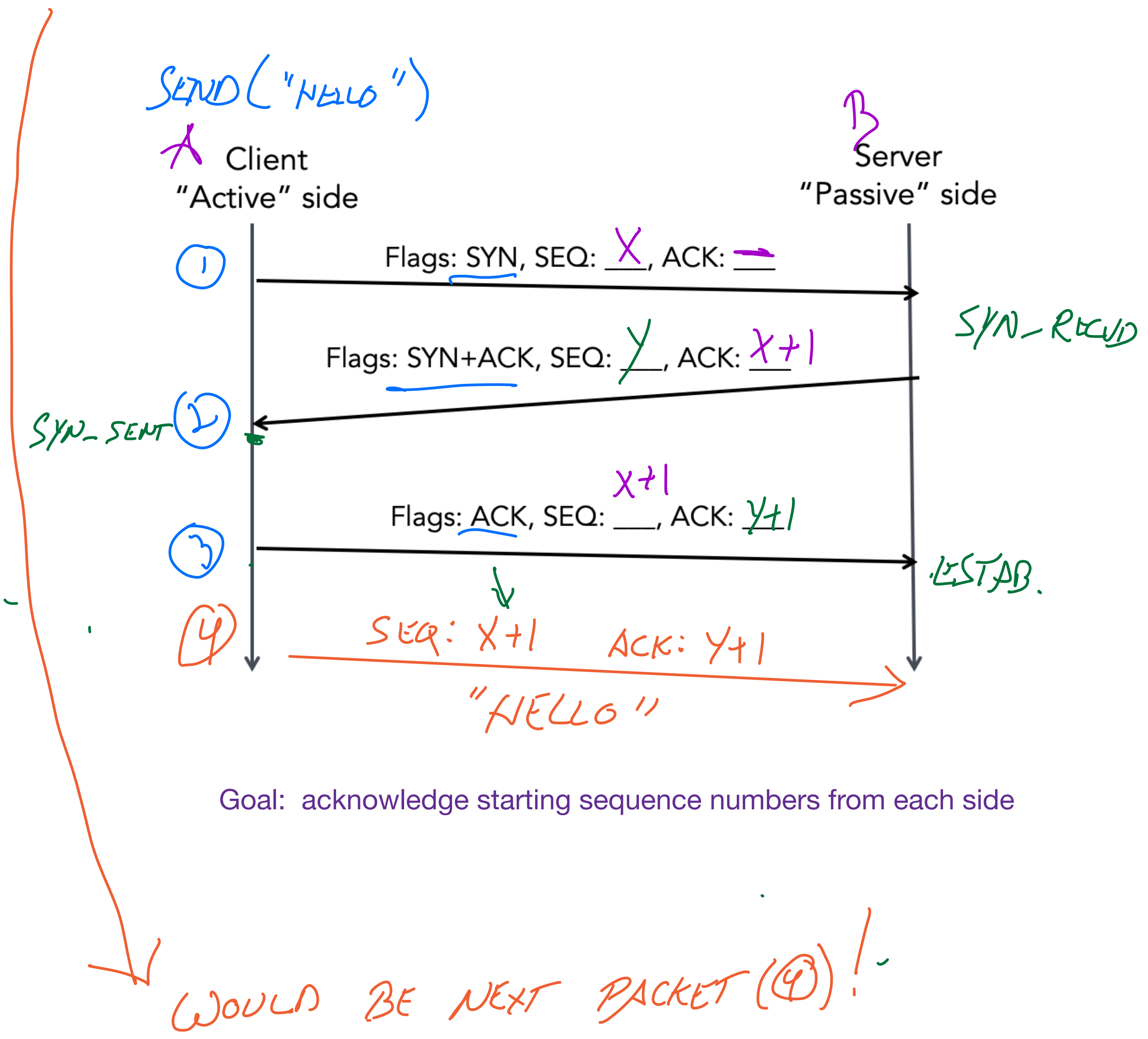
The header:



Warmup: TCP Handshake

1. What are the SEQ and ACK numbers? (From some starting values X and Y?)

2. When is the first data transmitted?

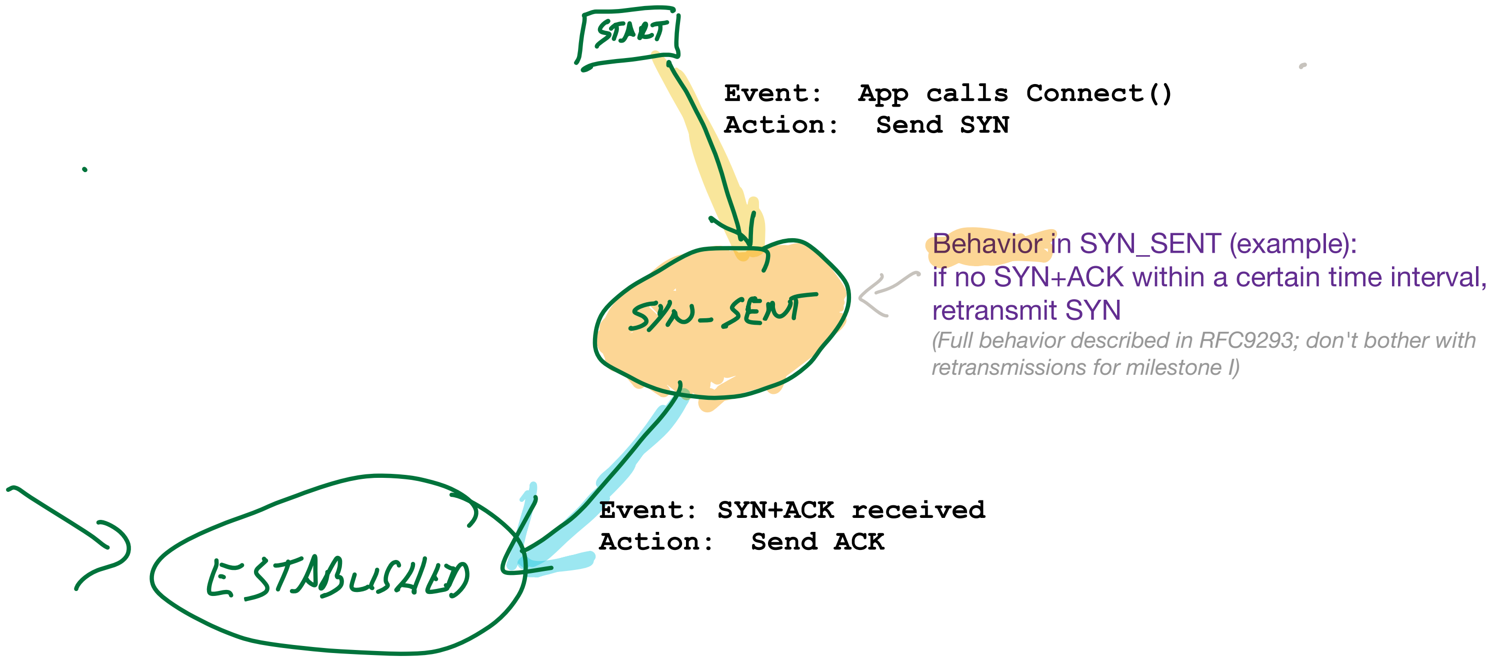


Goal: acknowledge starting sequence numbers from each side

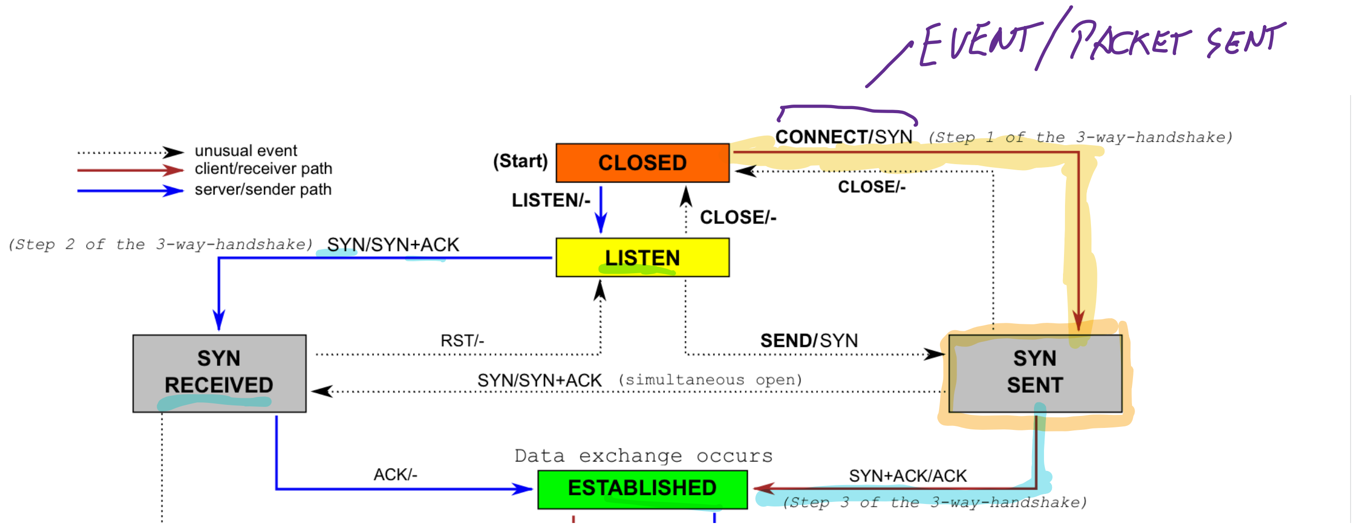
WOULD BE NEXT PACKET (4)!

TCP state machine: the story so far

Each connection has a "State" that defines how the TCP stack behaves
 => Events (receiving a packet, timeout, API call) transition between states


START
Event: App calls Connect()
Action: Send SYN
SYN-SENT
Behavior in SYN_SENT (example):
if no SYN+ACK within a certain time interval, retransmit SYN
(Full behavior described in RFC9293; don't bother with retransmissions for milestone I)
Event: SYN+ACK received
Action: Send ACK
ESTABLISHED

Here's a more precise diagram for the handshake steps (from Wikipedia's article on TCP):



Notation:

- *(Event that happens)/(packet sent in response)*
- ***Bold** indicates Socket API calls (e.g., syscalls)*

<u>Keeping track of connections</u>

For each connection, what data do we need to keep track of so far?

i.e., what per-connection "state" is required? (in the general sense, not just the state machine's state)

IP/port info (5-tuple)
Your initial sequence number
Sequence number from other side
What state you're in

Send buffer for data to send
Receive buffer for data that has arrived

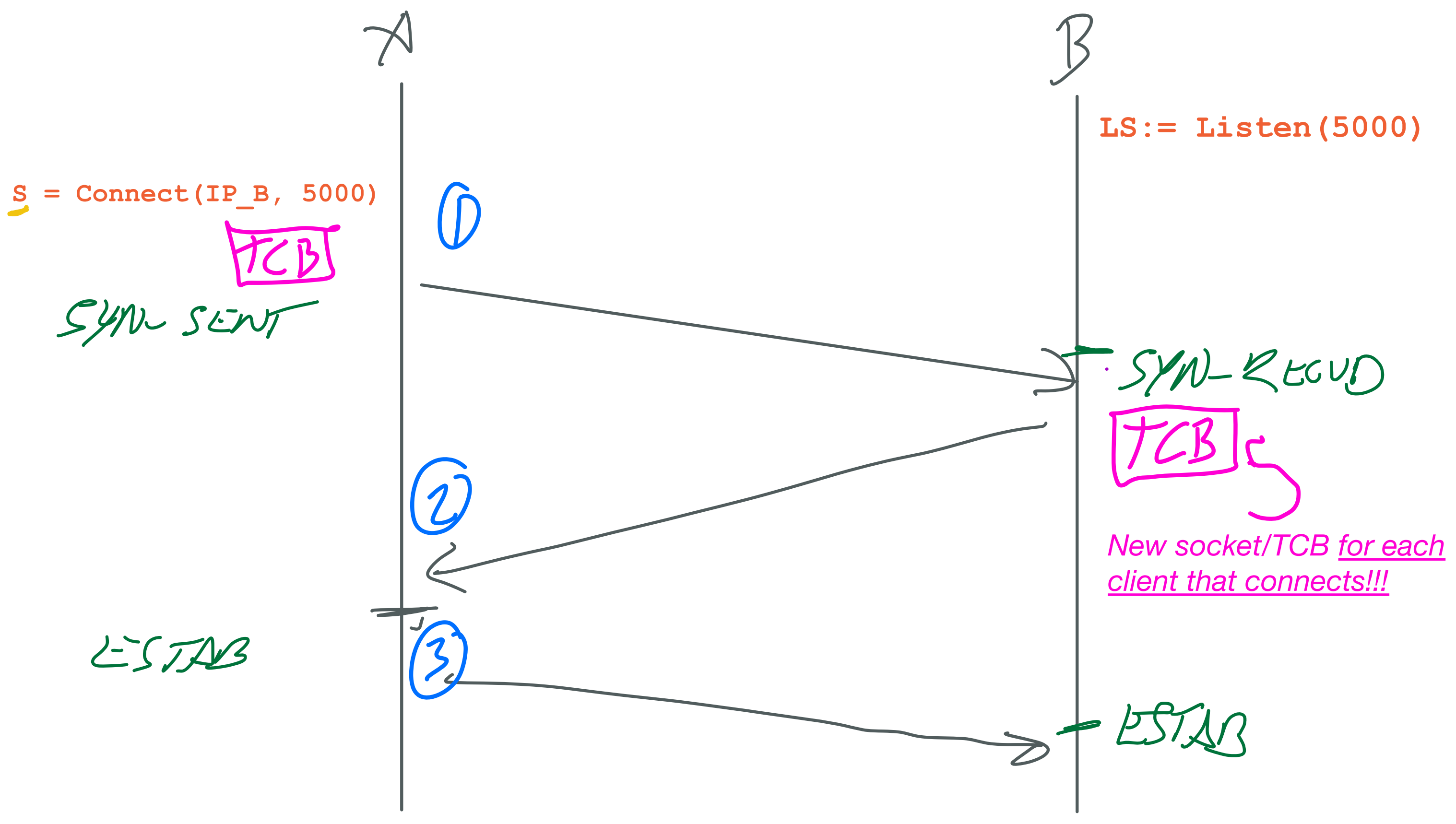
(Timers for retransmissions)

=> **<u>TCB: transmission control block</u>**: set of state information for a known connection

Think: parts of a struct

When to create the TCB? (or, more generally, when to create data representing each socket?)

- Server/passive side: listening on a connection (*though this is slightly different)
- Client/active side: initiating a connection (i.e., sending a SYN, `Connect()`)
- Server/passive side: accepting a new connection (receiving SYN)



```
clientConn := net.Dial("1.2.3.4:5000")

...
```

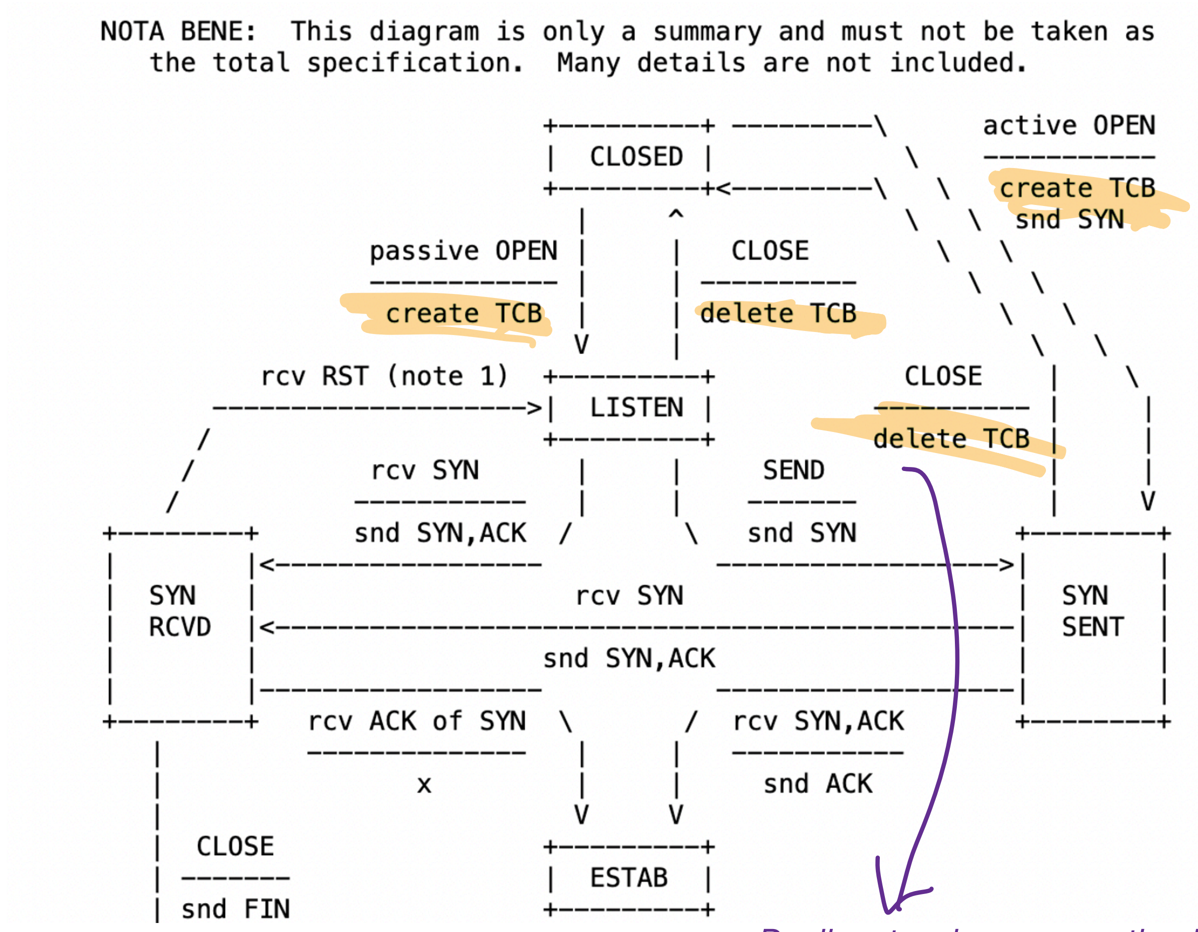
```
listenConn, err := net.Listen("tcp", ":5000")

for {
        clientConn, err := listenConn.Accept()
        //go handleClient(clientConn)
}
```

(We didn't see this in class, but here's another picture for thinking about when the TCB gets created, from the state machine diagram in the RFC.)

Transmission Control Block (TCB)

From RFC 9293, Sec 3.3.2:



```
    NOTA BENE:  This diagram is only a summary and must not be taken as
       the total specification.  Many details are not included.

                              +---------+ ---------\      active OPEN
                              |  CLOSED |            \    -----------
                              +---------+<---------\   \   create TCB
                                |     ^              \   \  snd SYN
                   passive OPEN |     |   CLOSE        \   \
                   ------------ |     | ----------       \   \
                    create TCB  |     | delete TCB         \   \
                                V     |                      \   \
            rcv RST (note 1)  +---------+            CLOSE    |    \
         -------------------->|  LISTEN |          ---------- |     |
        /                     +---------+          delete TCB |     |
       /           rcv SYN      |     |     SEND              |     |
      /           -----------   |     |    -------            |     V
  +--------+      snd SYN,ACK  /       \   snd SYN          +--------+
  |        |<-----------------           ------------------>|        |
  |  SYN   |                    rcv SYN                     |  SYN   |
  |  RCVD  |<-----------------------------------------------|  SENT  |
  |        |                  snd SYN,ACK                   |        |
  |        |------------------           -------------------|        |
  +--------+   rcv ACK of SYN  \       /  rcv SYN,ACK       +--------+
     |         --------------   |     |   -----------
     |                x         |     |     snd ACK
     |                          V     V
     |  CLOSE                 +---------+
     | -------                |  ESTAB  |
     | snd FIN                +---------+
```

Deallocate when connection is closed! (More on this later)

Important note: *this is a starting point, for a detailed description of how sockets get created, and how Listen/Accept work, see TCP gearup I.*

Keeping track of socket state

Where do we keep all of this info?
 - Application side: each socket is assigned a file descriptor number => maps to some kernel data structure
 - Network-side: whenever a TCP packet is received, OS needs to consult the **socket table** to map packet => some connection
 => This is where the 5-tuple comes in!

Proto	Local (yours)		Remote (theirs)		Socket
	IP	Port	IP	Port	
tcp	1.2.3.4	22	5.6.7.8	12453	TCB_1
tcp	1.2.3.4	22	100.3.15.7	66452	TCB_2
tcp	*	22	*	*	LISTEN

Tuple (local IP, local port, remote IP, remote port) => socket struct, including TCB

But what about the sockets with ""??*
=> Created when listening for new connections => acts like a "placeholder" to tell the TCP stack that we want to process new connections on a certain port

Therefore, we can think if of it like there are two "types" of sockets:

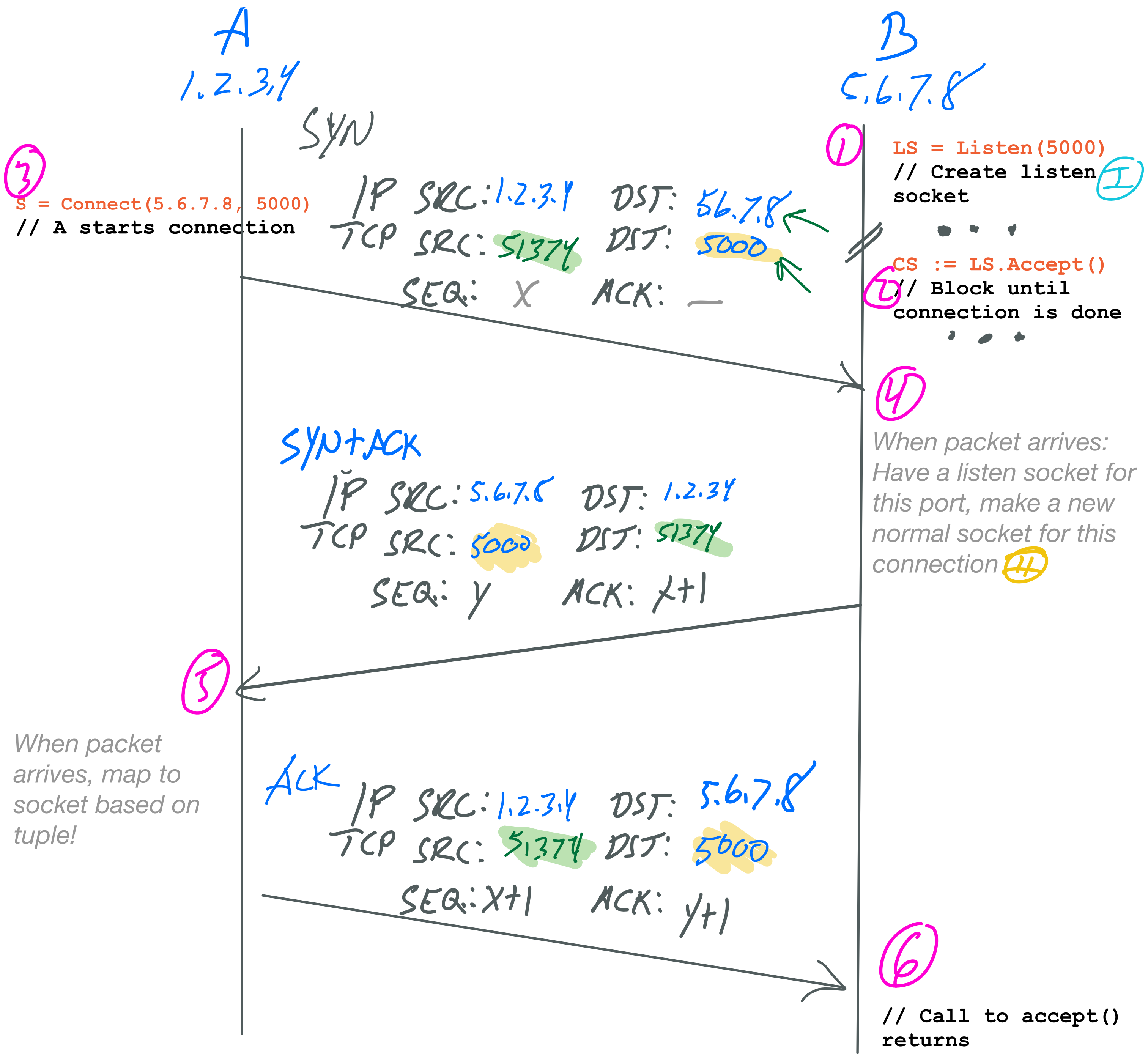
"Normal" sockets
 => connection two specific endpoints
 => has TCB, can send/recv data

Listen sockets
 - Created by receiver to accept new connections
 - Just a "placeholder" => can't send/receive, since not connected to anything!
 - When client connects, normal socket is created between server and that client
=> result of listen(), accept()
 => When a client connects, a new "normal" socket is created between the server and that client

Important note: *this is a starting point, for a detailed description of how sockets get created, and how Listen/Accept work, see TCP gearup I.*

***Important*: this diagram is mainly scratchwork--see the TCP gearup 1 notes for a fully-annotated example (with some more project-specific info)!**

Example: B listens on a port, A connects



```
LS = Listen(5000)
// Create listen socket
. . .
CS := LS.Accept()
// Block until connection is done
. . .
```

```
S = Connect(5.6.7.8, 5000)
// A starts connection
```

When packet arrives: Have a listen socket for this port, make a new normal socket for this connection

When packet arrives, map to socket based on tuple!

```
// Call to accept() returns
```

LOCAL		REMOTE		STATE
IP	PORT	IP	PORT	
1.2.3.4	51374	5.6.7.8	5000	~~SYN_SENT~~ ESTAB

	LOCAL		REMOTE		STATE
	IP	PORT	IP	PORT	
I	*	5000	*	*	*
II	5.6.7.8	5000	1.2.3.4	51374	~~SYN_RECVD~~ ESTAB