F24

# TCP Gearup II

# Overview

- How to think about send/recv
- About buffers
- How to debug/test in wireshark
- Implementation notes
- Any questions you have

# Roadmap

## Milestone II

- Basic sending and receiving using your sliding window/send receive buffers
- Plan for the remaining features

# Key resources

- Lecture 14: Send/recv basics
- Lecture 15: How sliding window works, retransmissions, zero-window probing

- HW3: Do it sooner rather than later—it will help!

- Testing and tools stuff: "TCP getting started" and "Testing with Wireshark" in the docs

VWrite (“s” command in REPL)

- Input: some normal socket, data you want to send
  - => You need to define your send/recv buffer, what variables/state etc you need to represent them
- Load data into your send buffer
- Block if send buffer is full, otherwise return number of bytes send

VRead (“r” command)

- Input: normal socket, buffer for received data
- Read from recv buffer, write that data to whatever buffer was passed in
- If recv buffer is empty, block
- Return: number of bytes read***

Your goals:

- Defining data structures (buffers, etc), variables for how you keep track of things in the buffer
- Receive packets, load them into recv buffer
- Send packets from send buffer

# Sending and receiving: API

More info: "Socket API example" in docs

VWrite ("s" command)

- Input: normal socket, data to send
- Loads data into send buffer
- Block if send buffer is full

VWrite ("r" command)

- Input: normal socket, buffer for received data
- Read from recv buffer, write to app buffer
- Block if recv buffer empty
- Return: number of bytes read

Demo!

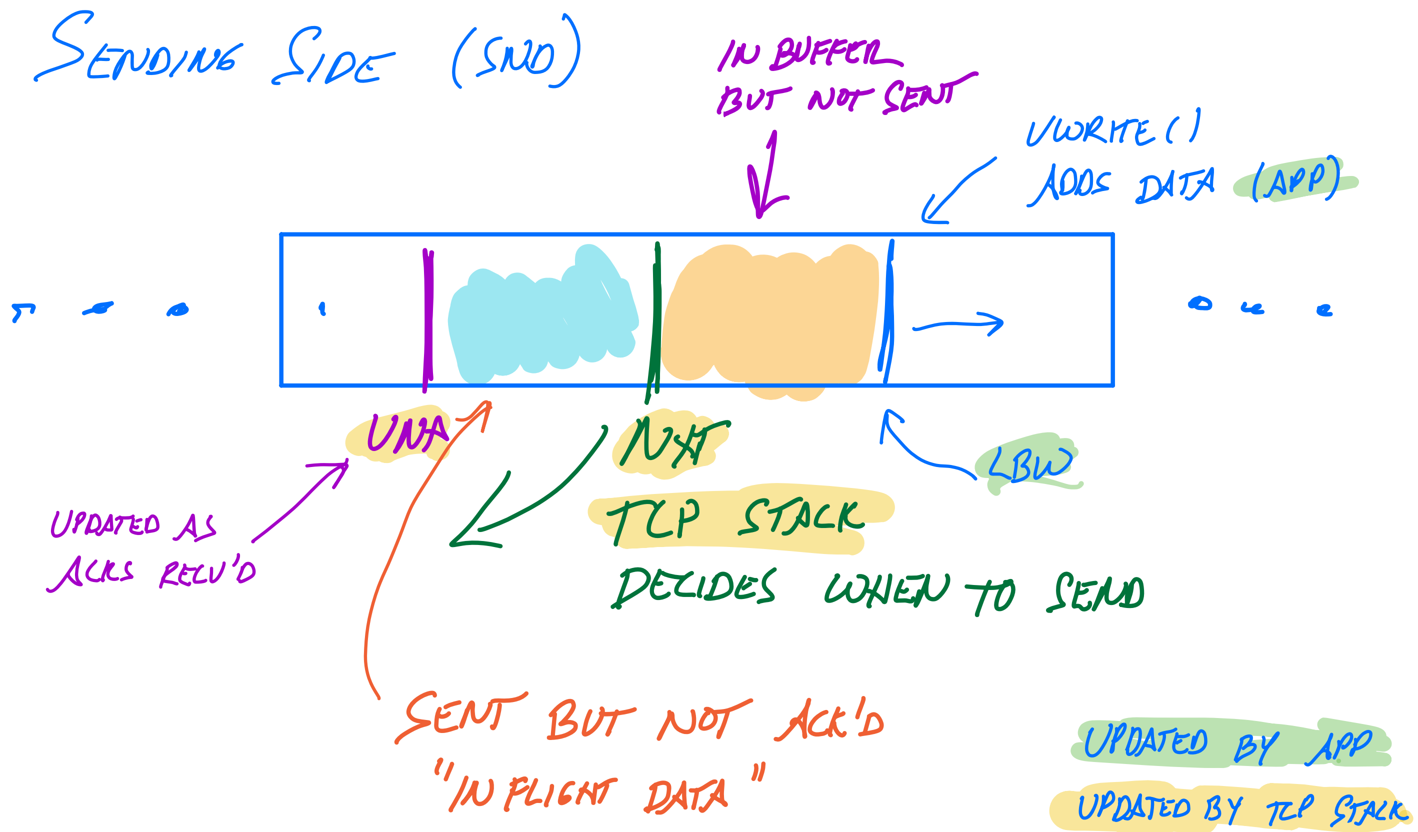
# Your buffers

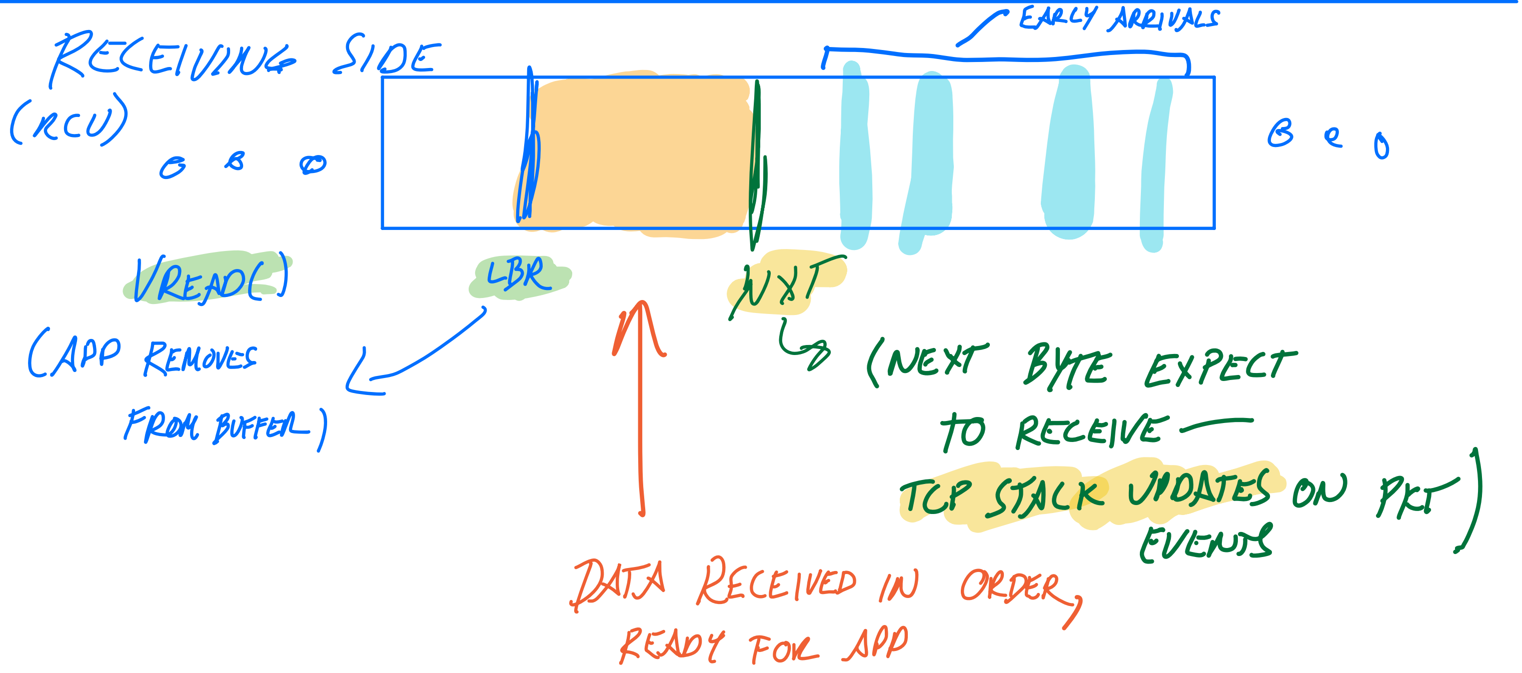
- Should use a circular buffer
- You get to decide on mechanics
  - How to keep track of read/write pointers
  - How to translate between sequence numbers => buffer indices
- You MAY use a library, but you should decide if this is what you want

For detailed info

=> RFC9293 Sec 3.3: what all the variables mean

⇒ Lecture 15: detailed breakdown of how to use buffers

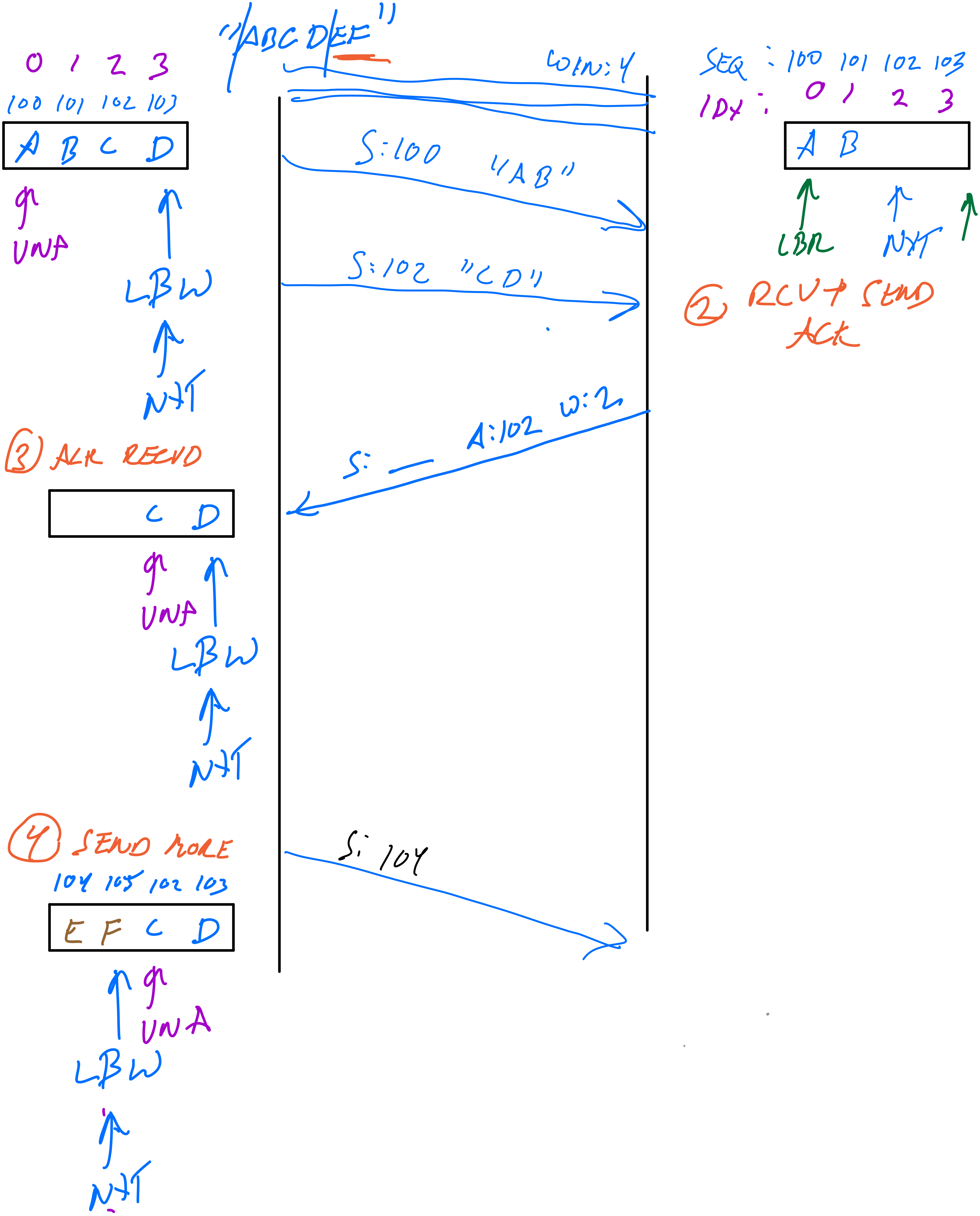



RECEIVING SIDE (RCV)
EARLY ARRIVALS
READ()
(APP REMOVES FROM BUFFER)
LBR
NXT
(NEXT BYTE EXPECT TO RECEIVE — TCP STACK UPDATES ON PKT EVENTS)
DATA RECEIVED IN ORDER, READY FOR APP

Want to see a better version of this? See the notes from lecture 15.

For more explanations, see RFC9293, Sec 3.3.

For more info on this part, we recommend doing HW3–it is designed to help here!



# What happens in the TCP stack?

Your TCP stack will have some threads—you decide what they do

When you get a new packet…

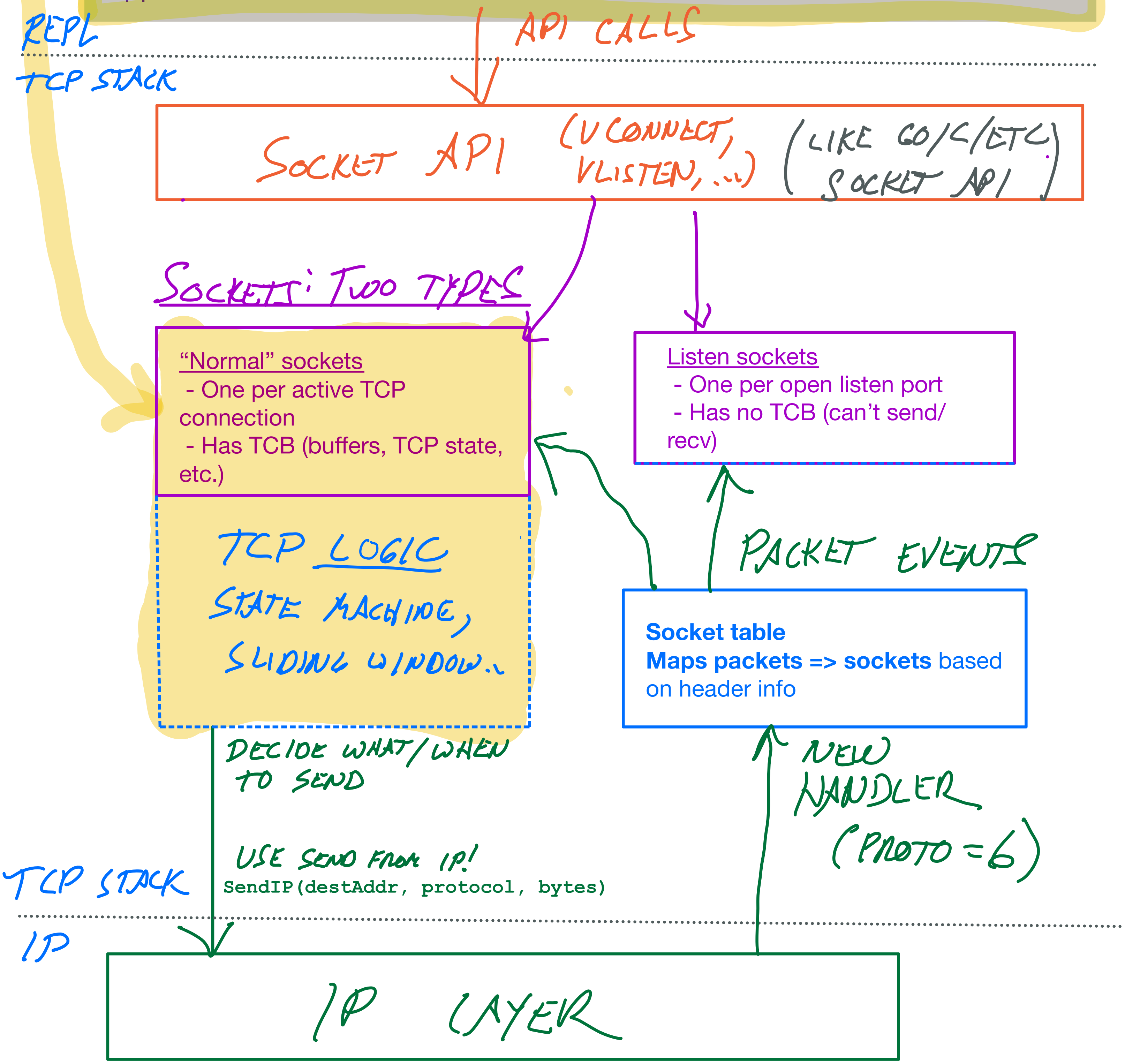
=> Look up 4-tuple in socket table => find socket struct

=> Socket struct => all your per-connection TCP state (buffers, sequence numbers, etc.…)

What to do with each segment? RFC9293 Sec 3.7.10 is your friend
=> + our modifications in "TCP notes" docs

## How does all of this fit into your work from before????

After Milestone I, most of your logic will be part of how you represent "normal-type" sockets
For any packet received/API call => map to a socket
 => Based on that socket's state (buffers, state machine, etc), what should happen?



"Normal" sockets
 - One per active TCP connection
 - Has TCB (buffers, TCP state, etc.)

Listen sockets
 - One per open listen port
 - Has no TCB (can't send/recv)

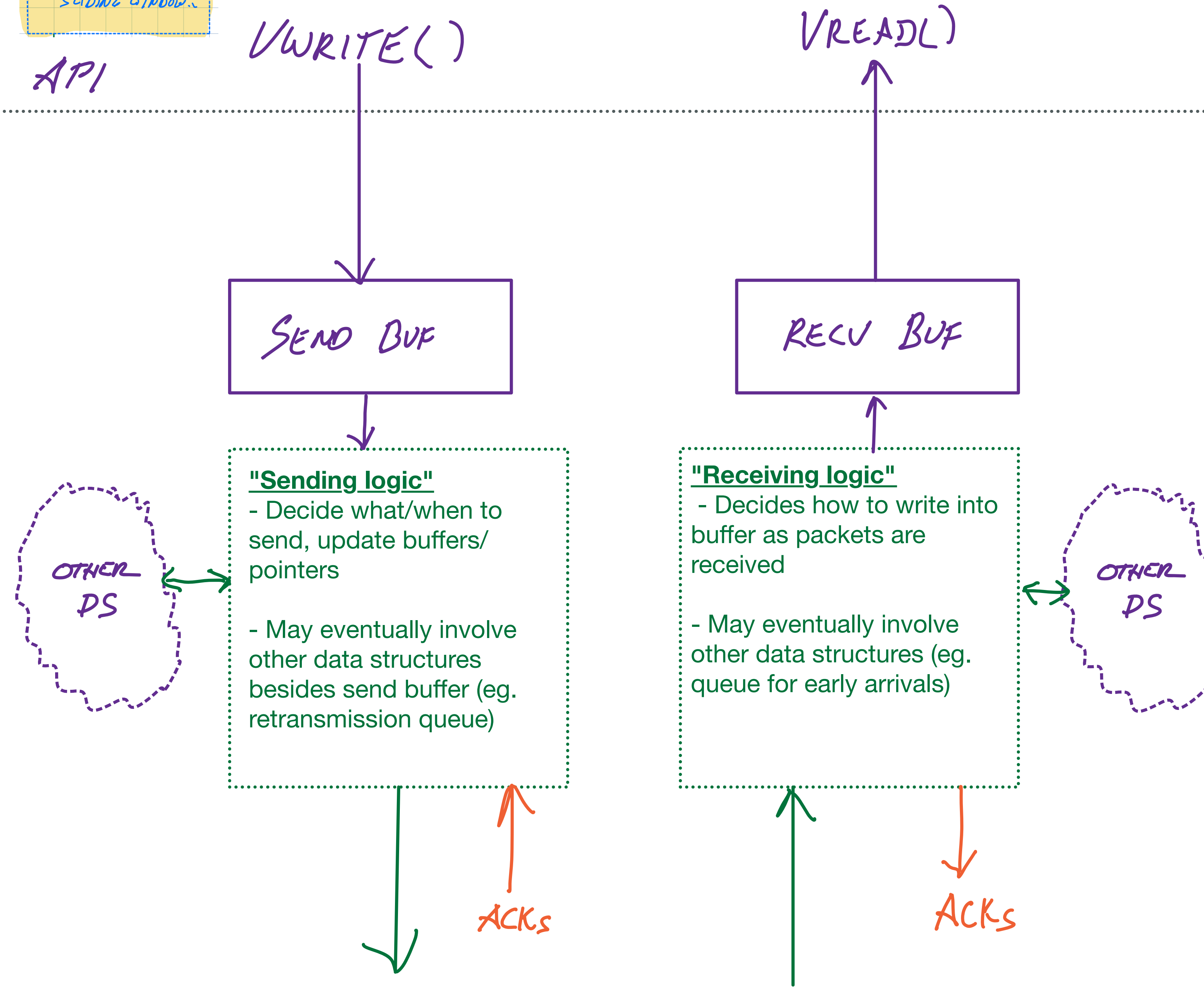
**Socket table**
**Maps packets => sockets** based on header info

```
SendIP(destAddr, protocol, bytes)
```

“Normal” sockets
- One per active TCP connection
- Has TCB (buffers, TCP state, etc.)

TCP LOGIC
STATE MACHINE,
SLIDING WINDOW...

# What happens inside a socket?



**"Sending logic"**
- Decide what/when to send, update buffers/ pointers

- May eventually involve other data structures besides send buffer (eg. retransmission queue)

**"Receiving logic"**
- Decides how to write into buffer as packets are received

- May eventually involve other data structures (eg. queue for early arrivals)

**Sends segments via your IP stack (eg. SendIP)**

**Packet arrivals**
*(when len(TCP payload) > 0)*

# Implementing VRead/VWrite

Performance requirement: send/recv process **MUST** be event driven

- No `time.Sleep`
- No busy-waiting

Where does this apply?

- REPL: s, r, sf, rf
- VRead/VWrite
- Deciding when to send, or check for new data

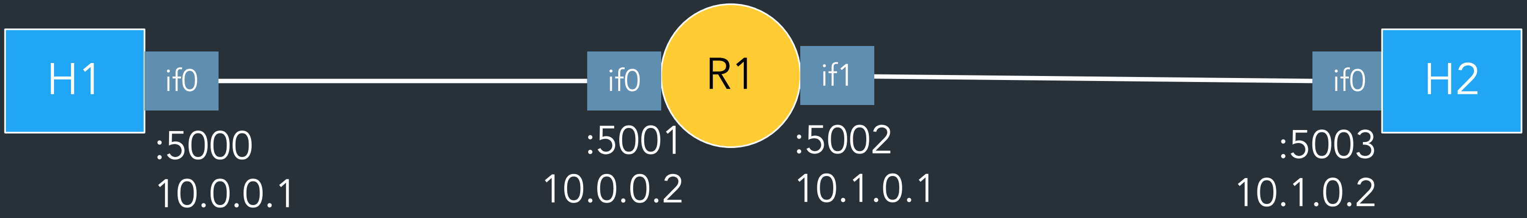
=> Channels, condition variables, etc. are your friends

# *Channels?*

**=> See code demo in video** (REALLY)

Also "channels demo" in docs and resources

# How to test TCP



Useful wireshark mechanics

- SEQ/ACK analysis
- Follow TCP stream
- Validating the checksum

Note: watching traffic in wireshark works differently in this project!
=> See "TCP getting started" guide for details

# Reference implementation

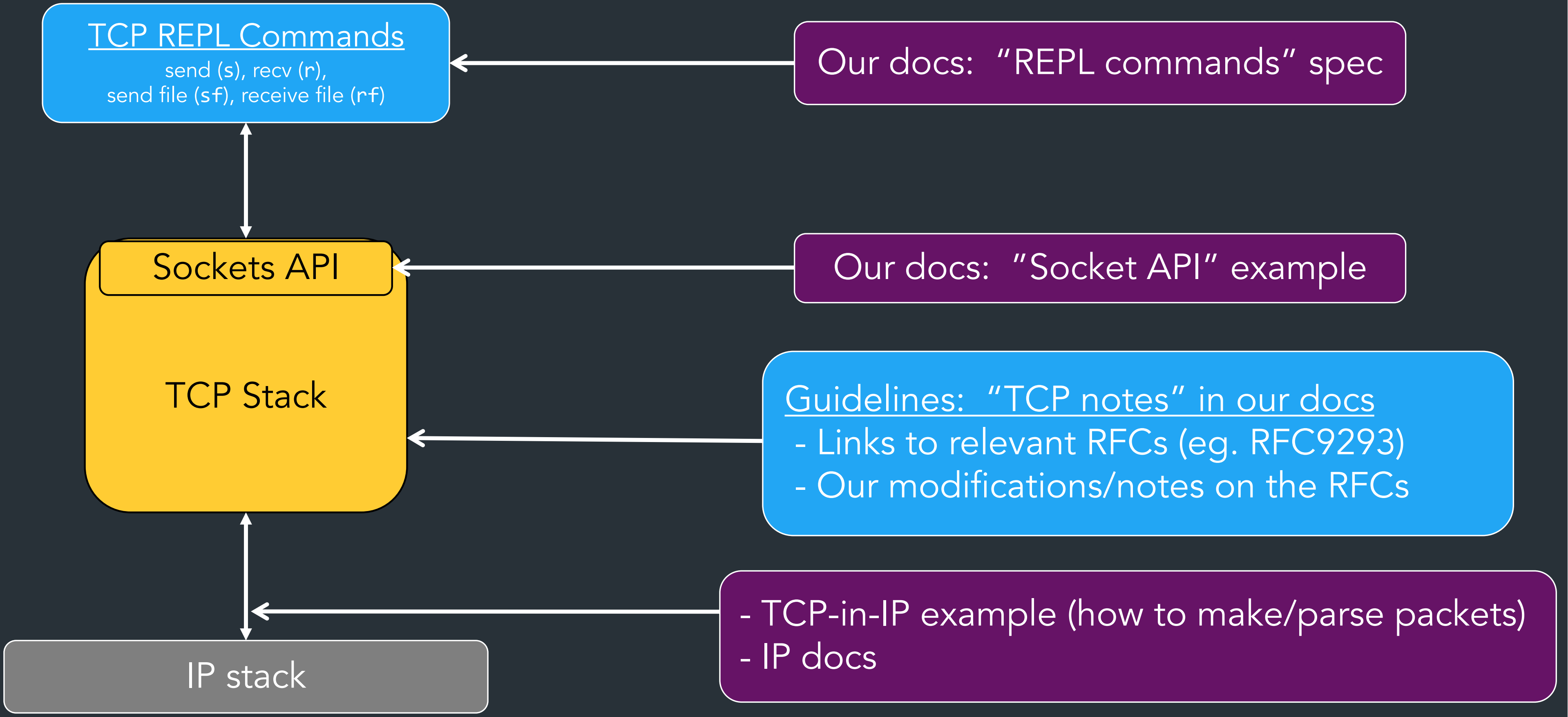
- Our implementation of TCP
- Try it and compare with your version!

Note: we're using a new reference this year (after 8+ years!)

- We've tested as best we can, but there may be bugs
- See Ed FAQ, docs FAQ for list of known bugs
- Let us know if you have issues!

⇒ If the spec disagrees with the reference implementation,
the spec wins-–don't propagate buggy behavior
(please help us find any discrepancies!)

# Where to get more info



TCP REPL Commands
send (s), recv (r),
send file (sf), receive file (rf)

Sockets API

TCP Stack

IP stack

Our docs: "REPL commands" spec

Our docs: "Socket API" example

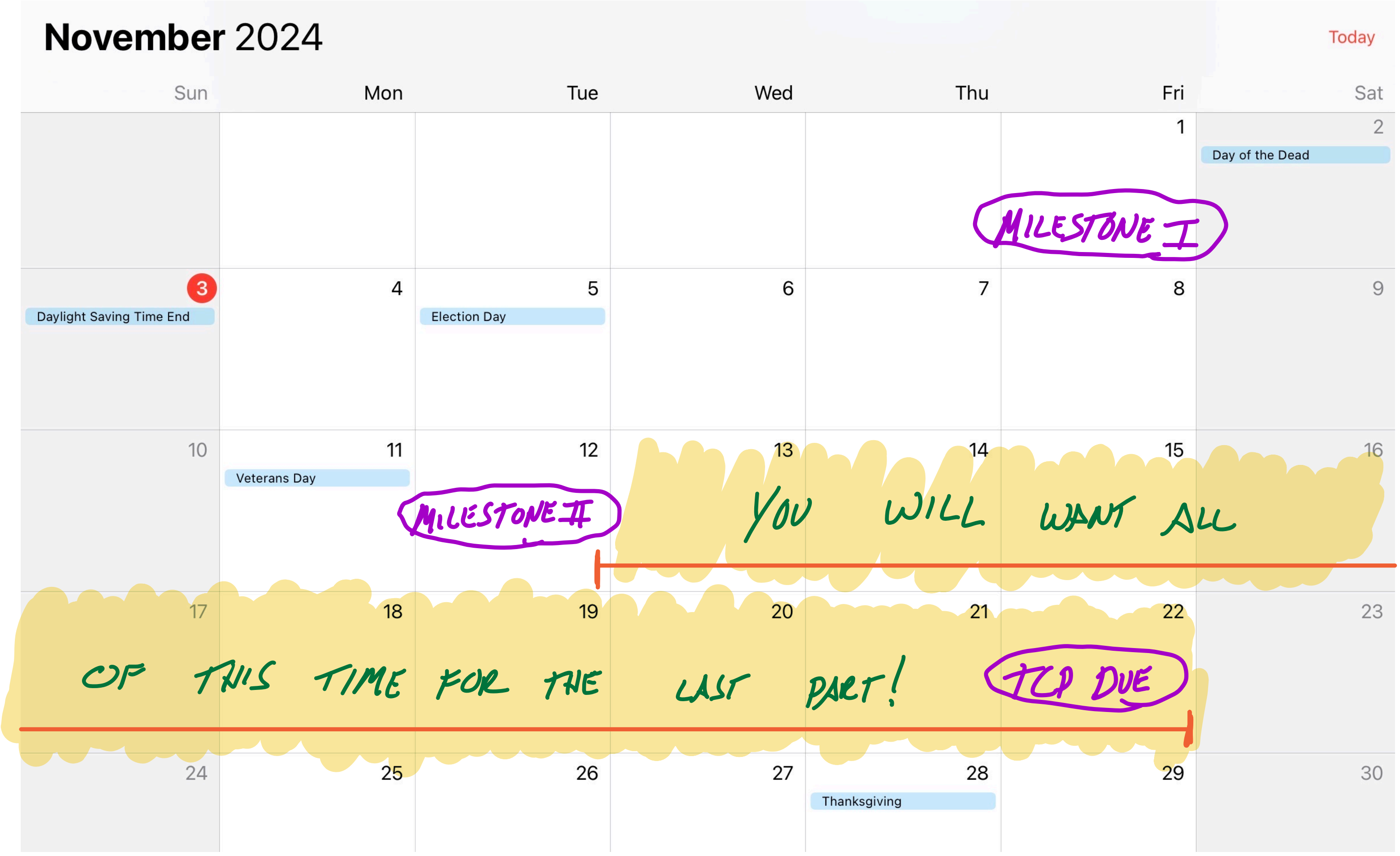
Guidelines: "TCP notes" in our docs
- Links to relevant RFCs (eg. RFC9293)
- Our modifications/notes on the RFCs

- TCP-in-IP example (how to make/parse packets)
- IP docs

# Roadmap

## Final deadline

- Retransmissions (+ computing RTO from RTT)
- Zero-window probing
- Connection teardown
- Sending and receiving files (sf, rf)



**November** 2024

Today

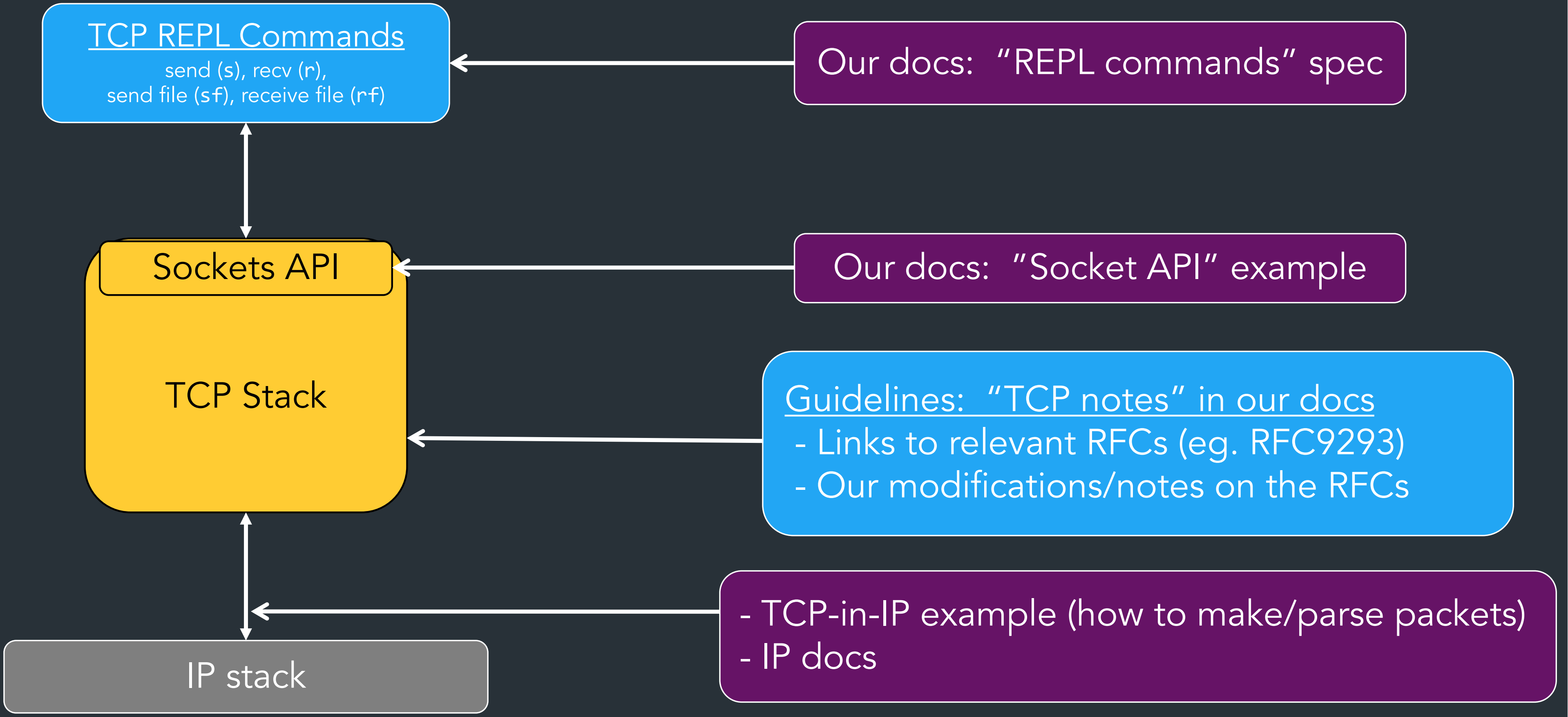
| Sun | Mon | Tue | Wed | Thu | Fri | Sat |
|---|---|---|---|---|---|---|
| | | | | | 1<br>MILESTONE I | 2<br>Day of the Dead |
| 3<br>Daylight Saving Time End | 4 | 5<br>Election Day | 6 | 7 | 8 | 9 |
| 10 | 11<br>Veterans Day | 12<br>MILESTONE II | 13 | 14 | 15 | 16 |
| 17 | 18 | 19 | 20 | 21 | 22<br>TCP DUE | 23 |
| 24 | 25 | 26 | 27 | 28<br>Thanksgiving | 29 | 30 |

YOU WILL WANT ALL OF THIS TIME FOR THE LAST PART!

**The features you need after Milestone II are not trivial—there is a lot of testing and debugging involved, so <u>do not underestimate this part</u>. All of your other course deadlines are set in order to ensure you have enough time between Milestone II and the TCP deadline.**

**<u>What this means now:</u> make sure you use your Milestone II time wisely so that you can spend the time afterward to focus on the other features!**

# Where to get more info



TCP REPL Commands

send (s), recv (r),
send file (sf), receive file (rf)

Sockets API

TCP Stack

IP stack

Our docs: "REPL commands" spec

Our docs: "Socket API" example

Guidelines: "TCP notes" in our docs
- Links to relevant RFCs (eg. RFC9293)
- Our modifications/notes on the RFCs

- TCP-in-IP example (how to make/parse packets)
- IP docs

# Closing thoughts

- Use your milestone time wisely!
- Wireshark is the best way to test—use it!
- Stuck? Don't know what's required? Just ask! (And see Ed FAQ)

We are here to help!