CS 484 Lecture 7
The Transport Layer

Fall 2022

Joshua Reynolds

Layer 4 - Transport Layer

Layer 1 - Physical Signalling

Layer 2 - Data Links

Layer 3 - Networking

Layer 4 - Getting Service Guarantees:

- Delivery confirmation
- In-order delivery
- Congestion Control

TCP - Transmission Control Protocol

Guarantees in-order delivery

Includes confirmation of delivery

Has congestion control

Implemented in the operating system

Security connection: You can sometimes tell what operating system is running by how its TCP stack behaves

UDP - User Datagram Protocol

No guarantee of data arrival order

No congestion control

No delivery report

Why do we want this? Isn’t it basically just the same as the Internet Protocol?

UDP - User Datagram Protocol

No guarantee of data arrival order

No congestion control

No delivery report

Why do we want this? Isn’t it basically just the same as the Internet Protocol?

The idea of ports

“Ports”

A computer may have many network processes that all need to have network conversations at the same time.

Operating systems need a way to keep track of all network conversations

So, they give them each a number between $0\text{-}2^{16}$

Operating Systems have two separate lists - UDP ports and TCP ports

Standard TCP Ports for Applications

Service	Port
HTTP	80
SMTP	25
SSH	22

HTTPS	443
FTP	20
Telnet	23
WHOIS	53

Standard UDP Ports for Applications

Service	Port
DNS	53
NTP	123

Everything is a File

In Linux, we pretend everything is a file.

Bytes coming in order from the keyboard driver to a terminal are no different than bytes being read in order from a file on disk

We can link programs together by treating STDIN and STDOUT like files

What else has in-order bytes you can read and write to?

Everything is a File

In Linux, we pretend everything is a file.

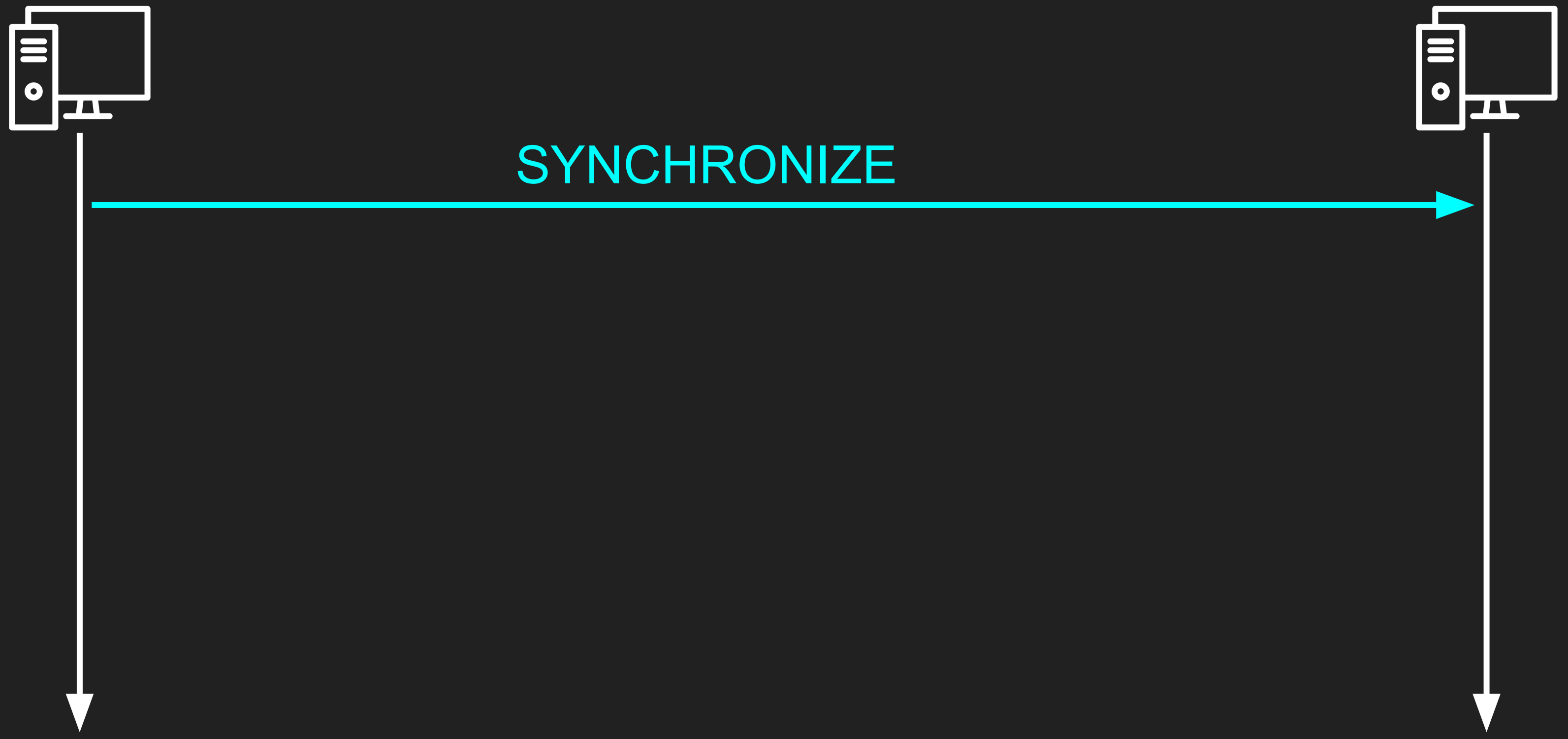
Bytes coming in order from the keyboard driver to a terminal are no different than bytes being read in order from a file on disk

We can link programs together by treating STDIN and STDOUT like files

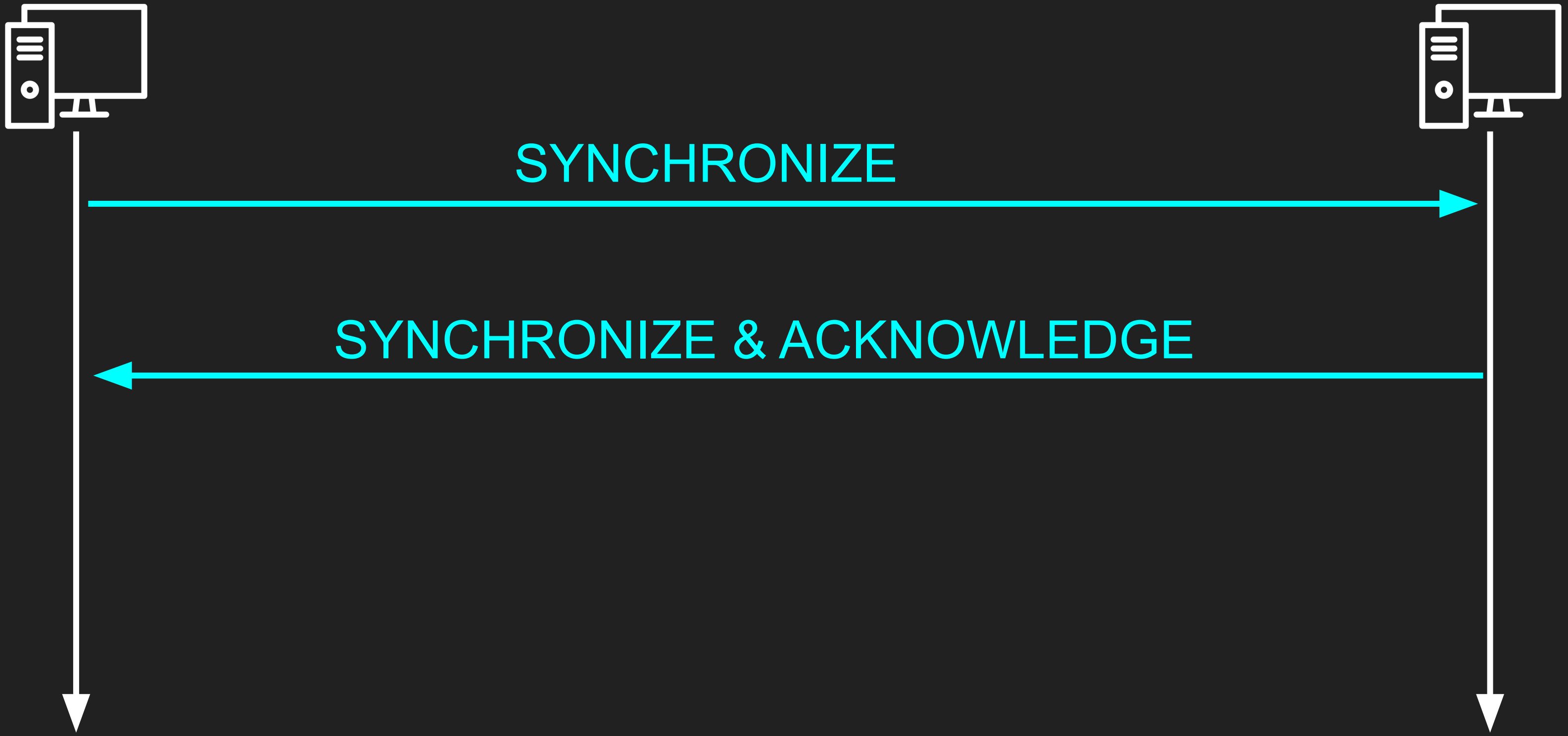
What else has in-order bytes you can read and write to?

TCP Connections!!!

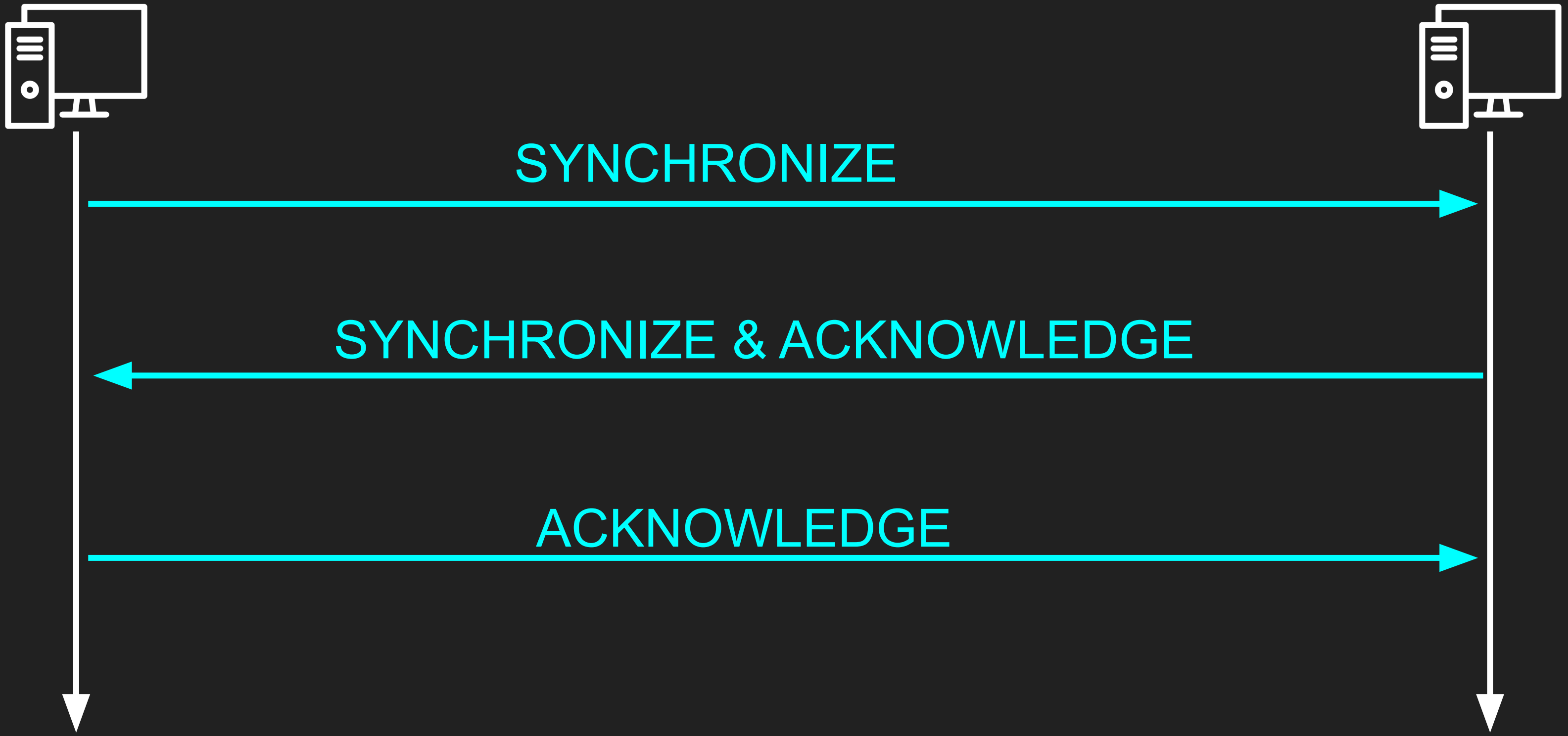
TCP Setup: 3-way handshake



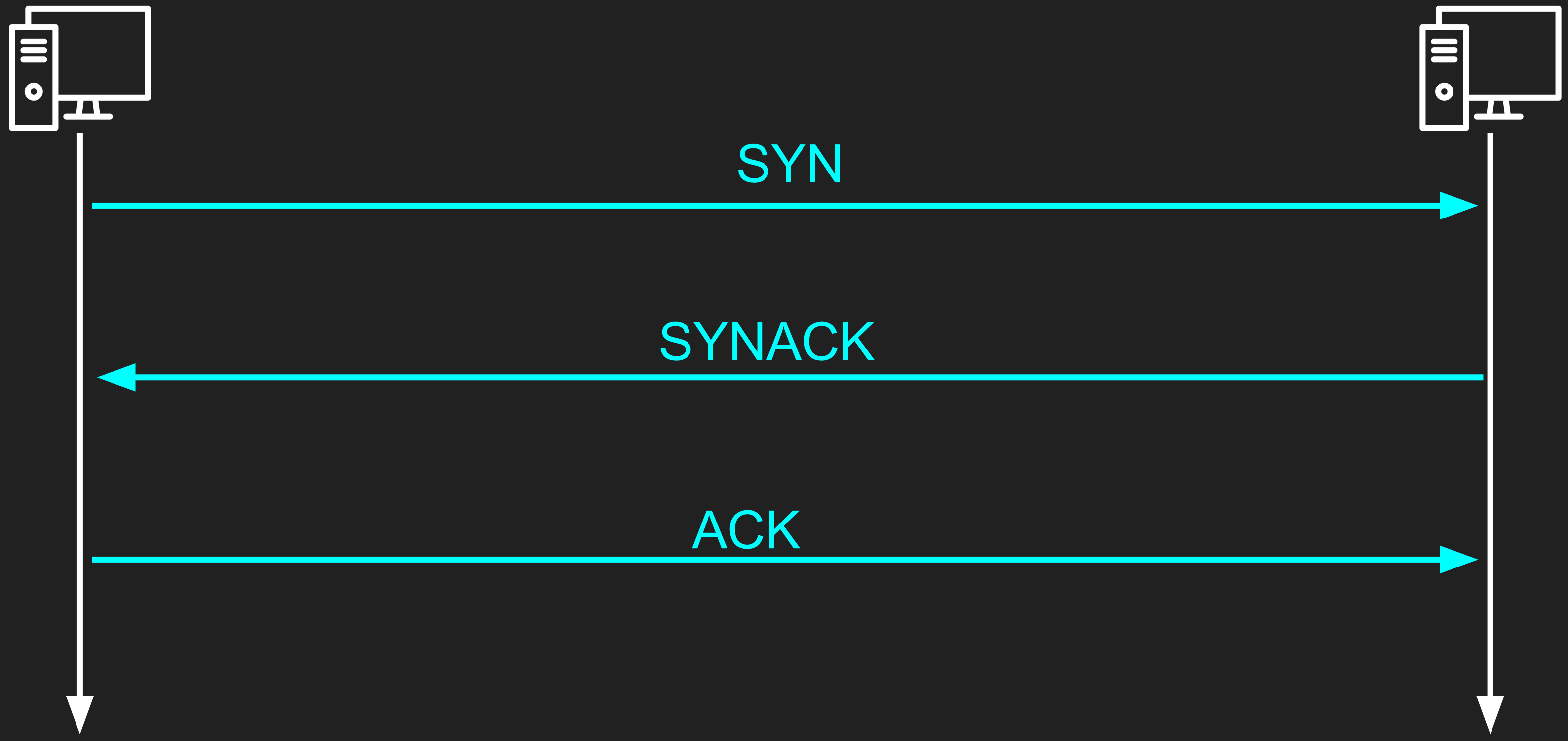
TCP Setup: 3-way handshake



TCP Setup: 3-way handshake



TCP Setup: 3-way handshake



TCP: Data Acknowledgement

TCP: Data Acknowledgement

TCP: Data Acknowledgement

TCP: Data Acknowledgement

TCP: Data Acknowledgement

Sequence Numbers and Data Acknowledgements

TCP keeps track of how much data each side has sent and received.

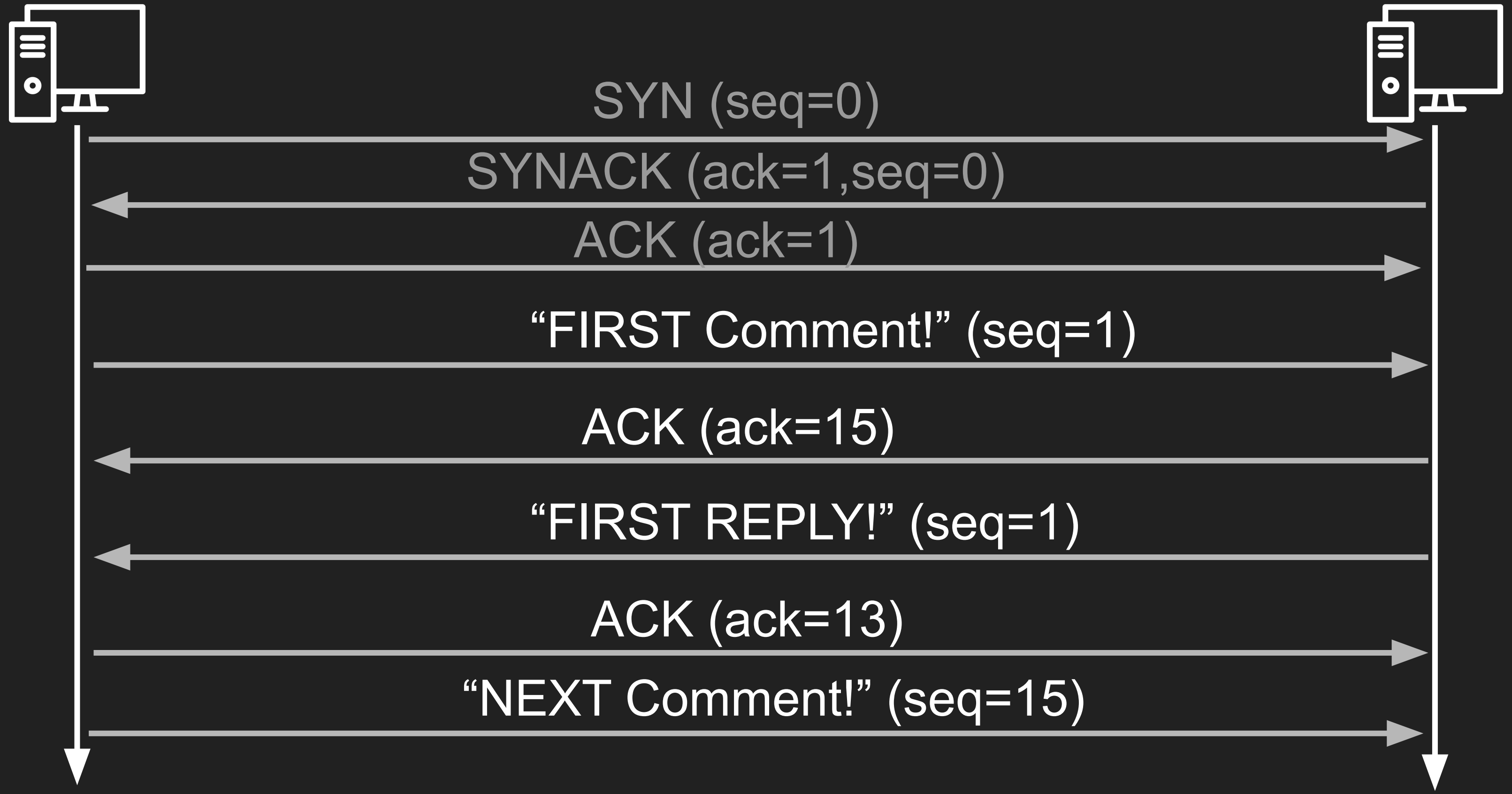
Sequence Number:

- The number of bytes I have sent so far
- An empty SYN packet counts as 1 byte sent

Acknowledgement Number:

- The number of bytes I have received so far
- An empty SYN packet counts as 1 byte received

TCP: (Sequence Number, Acknowledgement Number)



TCP Game

3 Servers

4 Routers

Everyone else is a client

How TCP Does Guaranteed Ordering

Sequence numbers keep track of the order data should be coming in.

If you sent data, but it was never acknowledged, you need to retransmit

The End-To-End Argument

If you are going to check errors, you might as well do it at higher layers.

Checking at lower layers may be more trouble than it is worth.

If you need to retransmit, usually it isn’t worth figuring out exactly why

It’s faster to just try again as long as probability of success is high

TCP: The Send and Receive Windows

These windows are a size limit measured in bytes.

Since both sides of a TCP connection can send data, each one has its own receive window and send window.

If you are sending, you can send up to your sending window number of bytes before you have to stop and wait for an ACK.

If you are receiving, you tell the other computer to never send more data than you can buffer at a time, your receive window.

Computer A's receive window is Computer B's sending window, and vice versa

TCP Window Variations

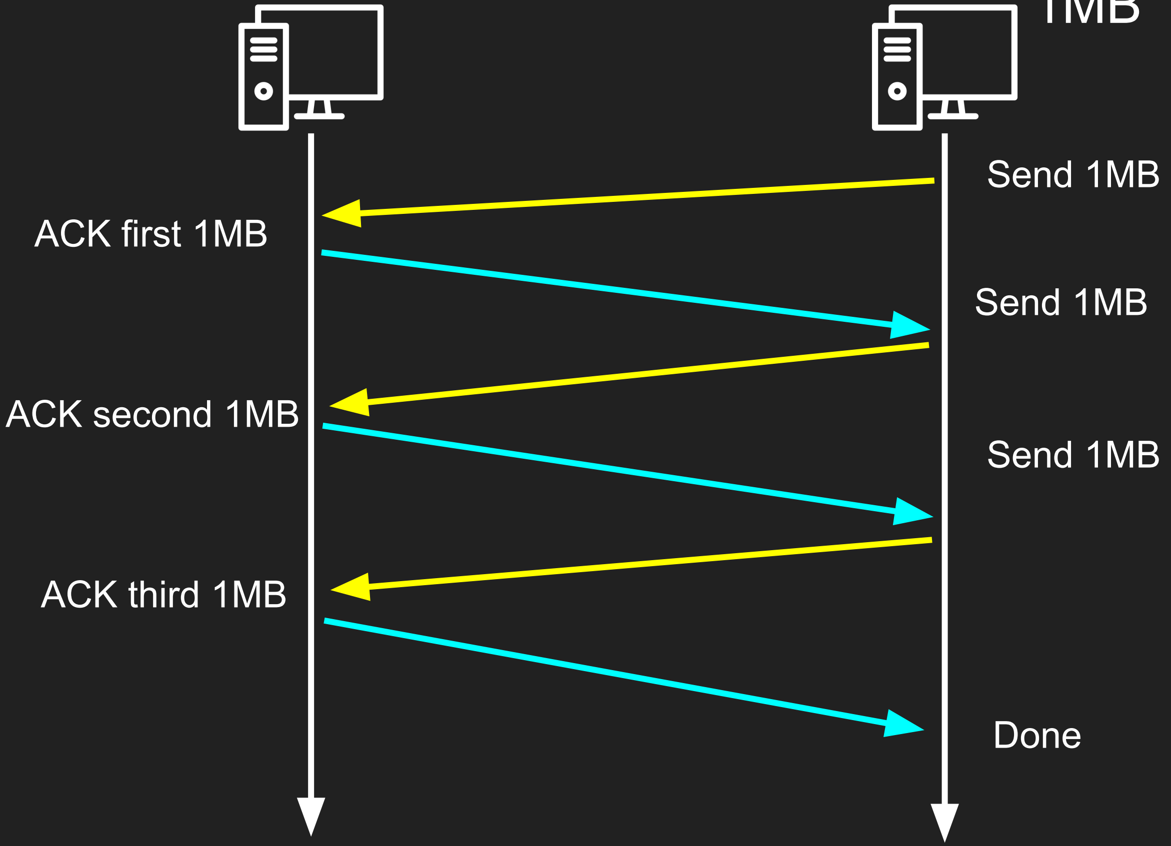
Stop-and-Wait: Break data up into chunks equal to the size of the sending window. Send data one chunk at a time, then wait for an ACK

Sliding Windows: Break the data up into chunks smaller than the size of the sending window. Send as many as you can. Every time you get an ACK of a smaller part, you can immediately send more

Multiple ACKs: TCP allows you to ACK multiple data packets with one ACK. The sender knows what is ACKed by the acknowledgement number

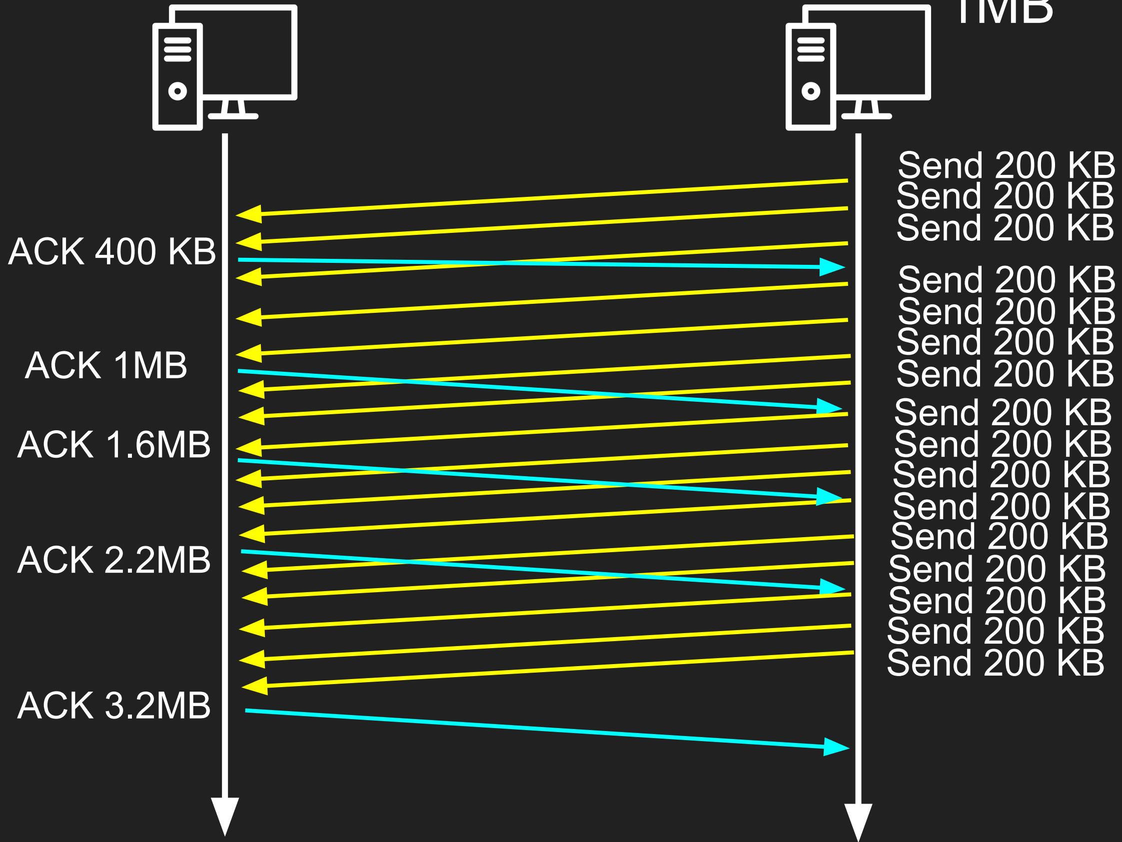
TCP Stop and Wait

Sending Window: 1MB



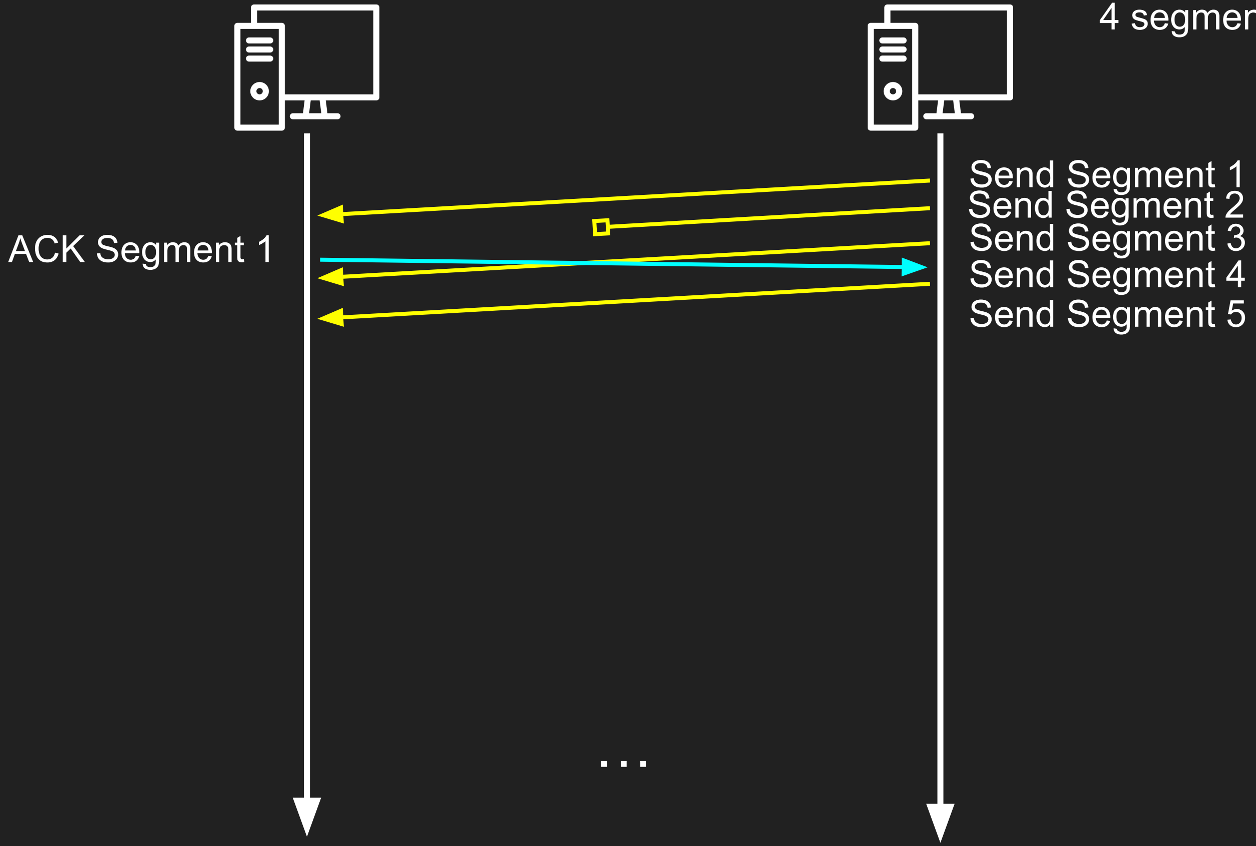
TCP Sliding Window

Sending Window: 1MB

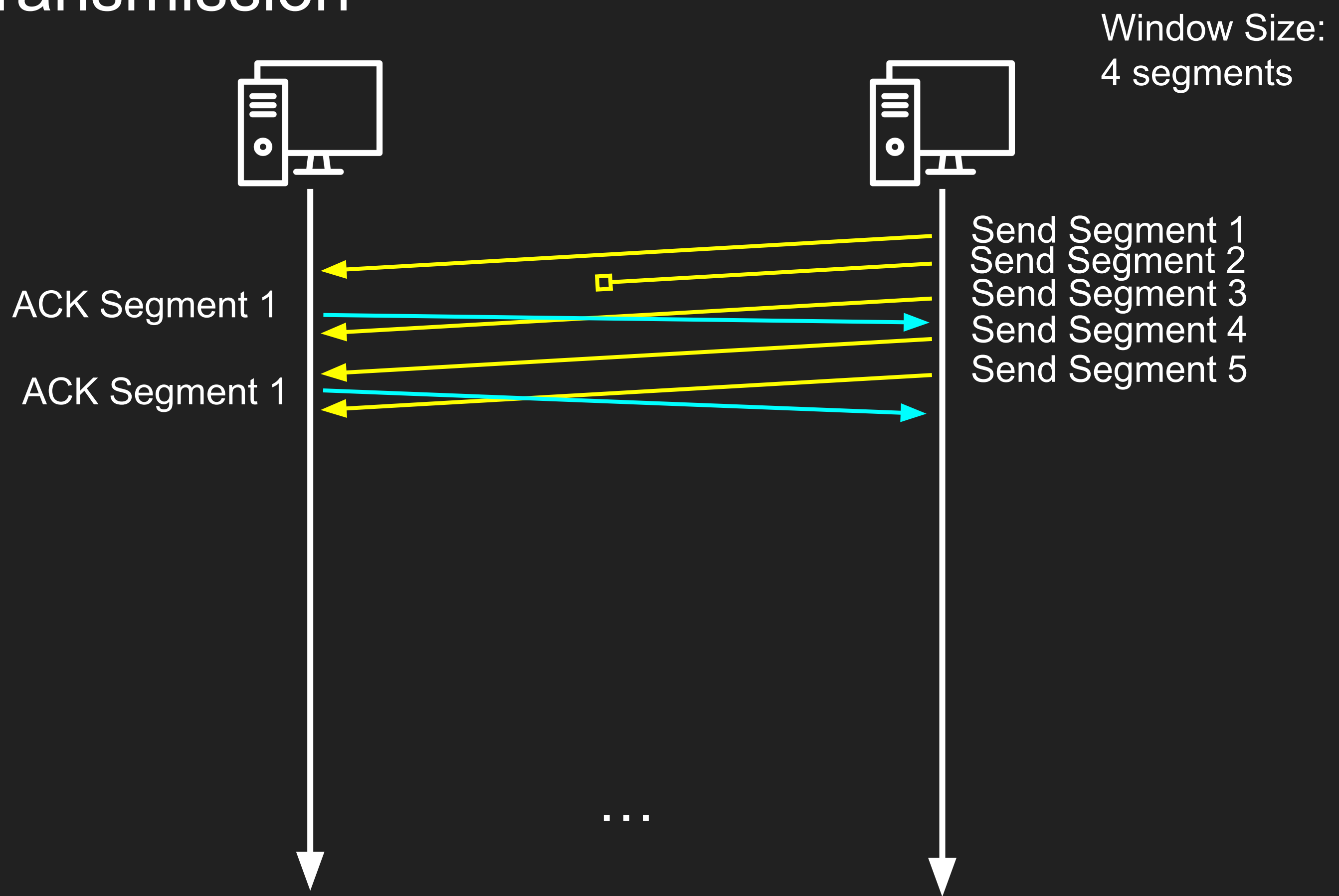


TCP Retransmission

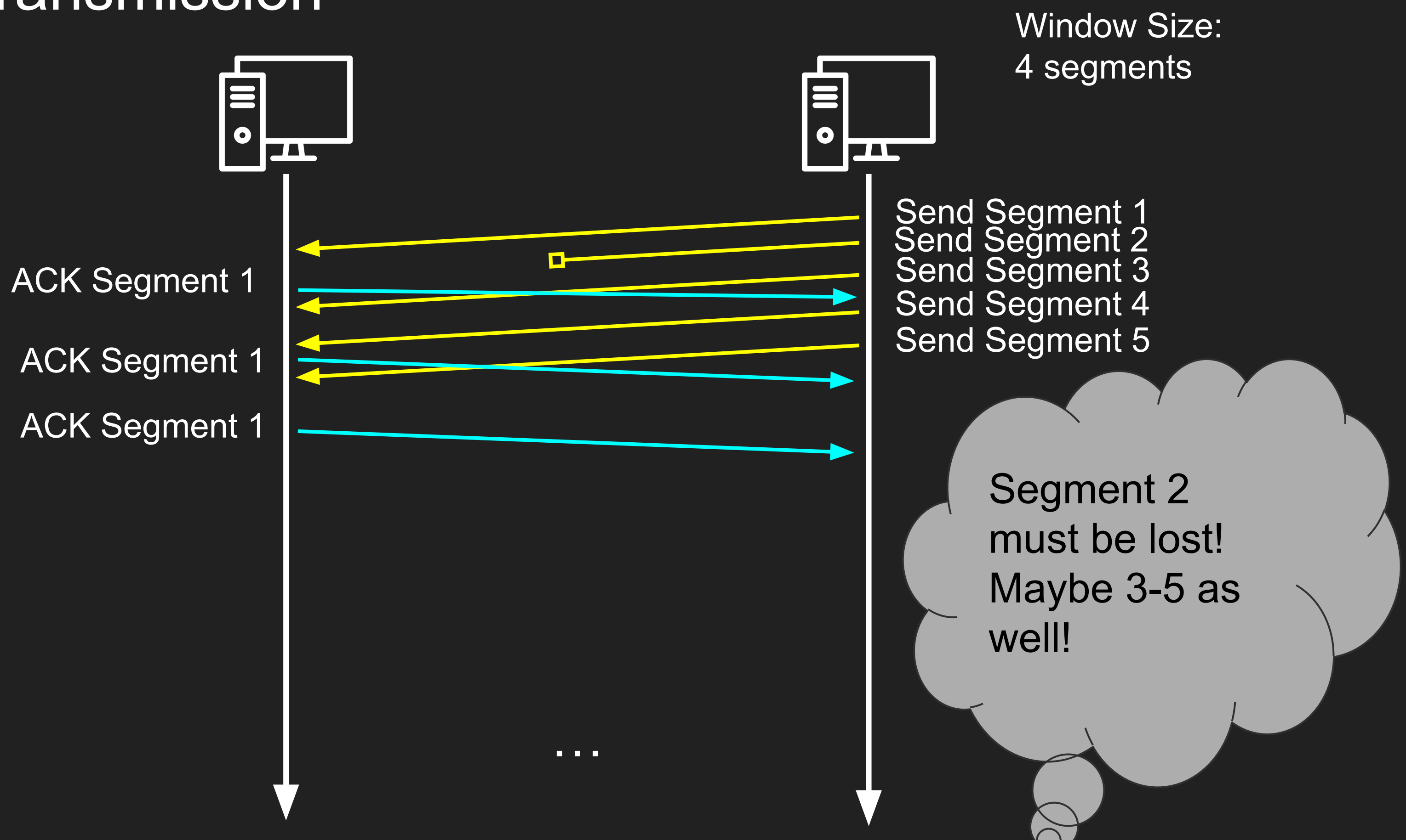
Window Size:
4 segments



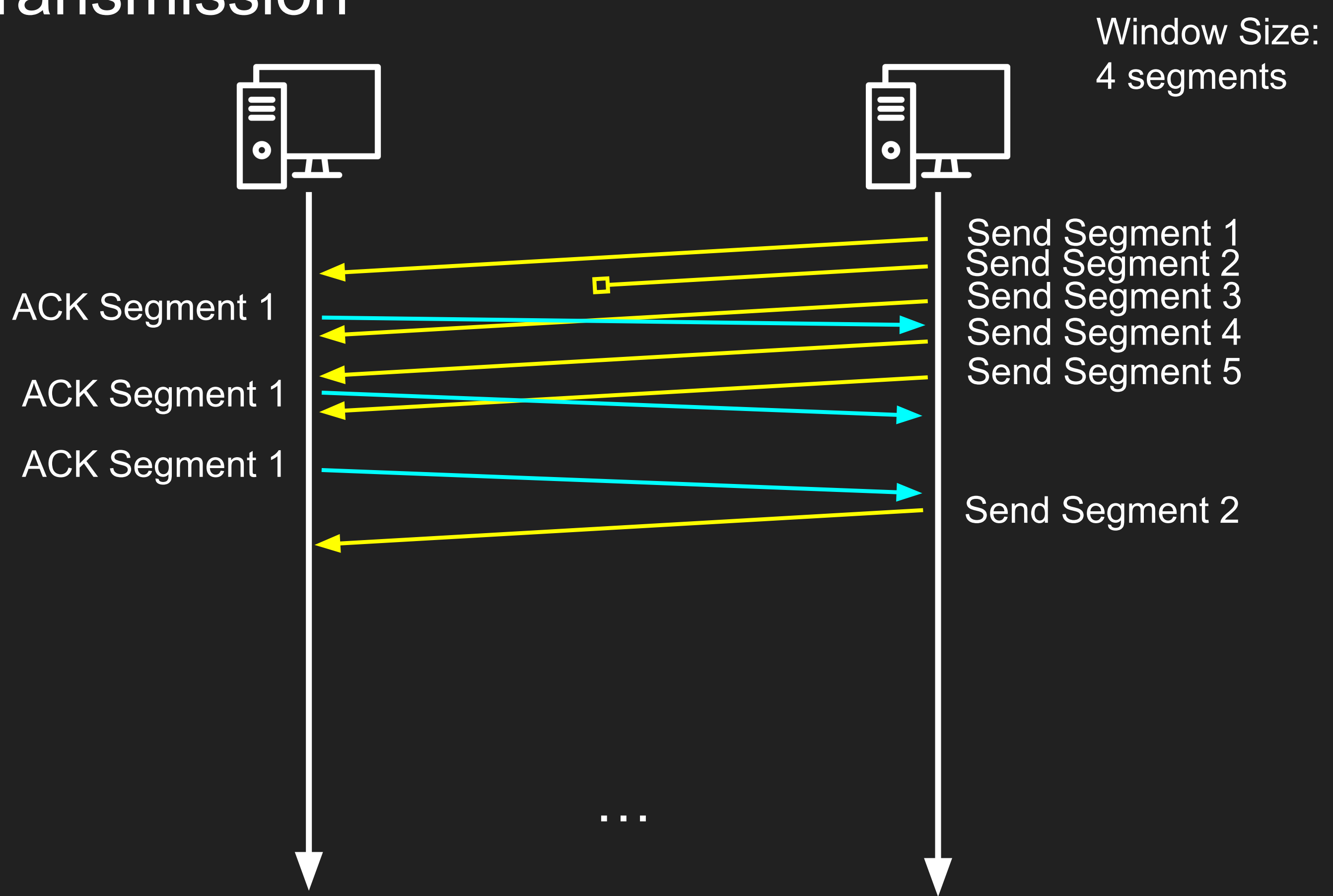
TCP Retransmission



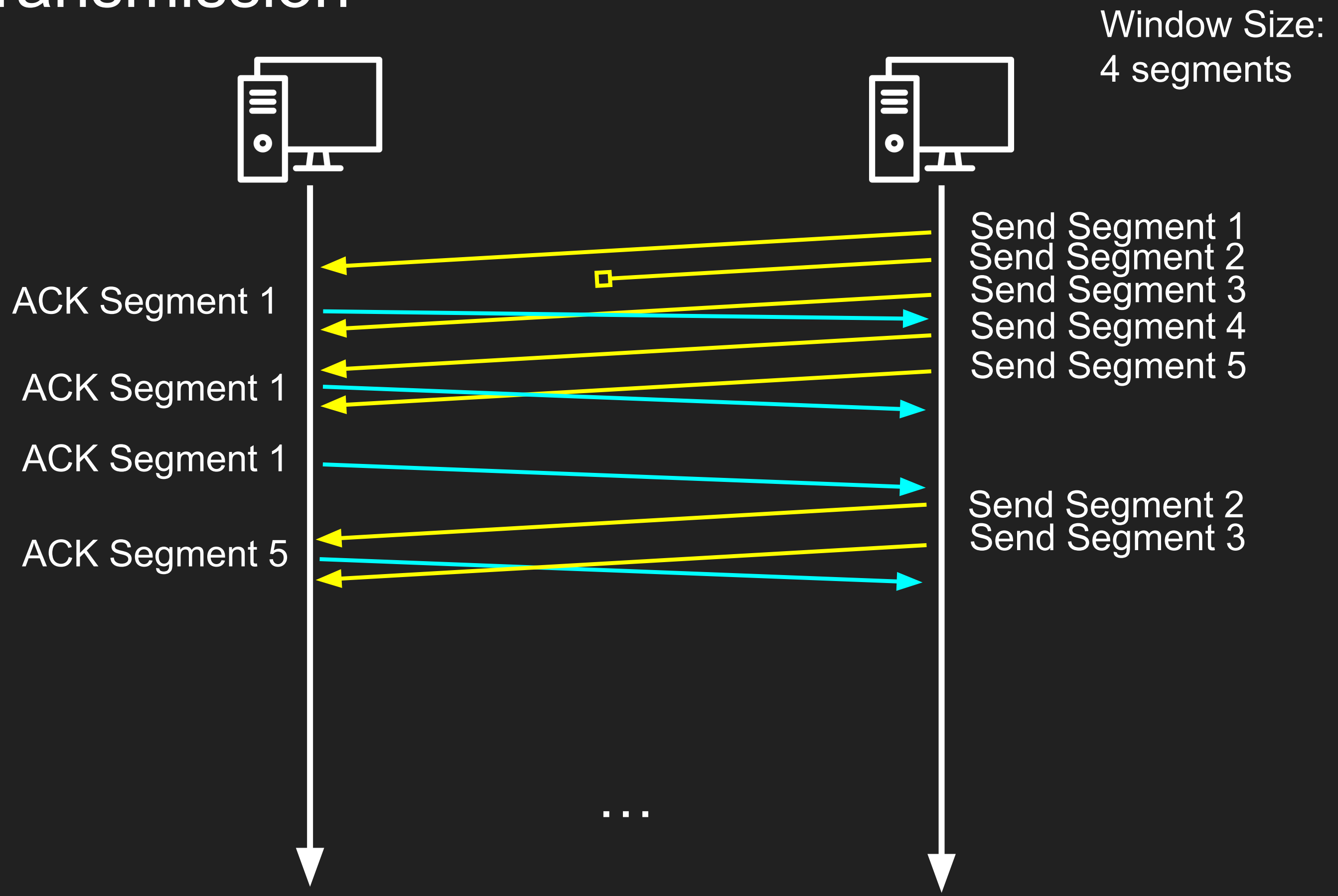
TCP Retransmission



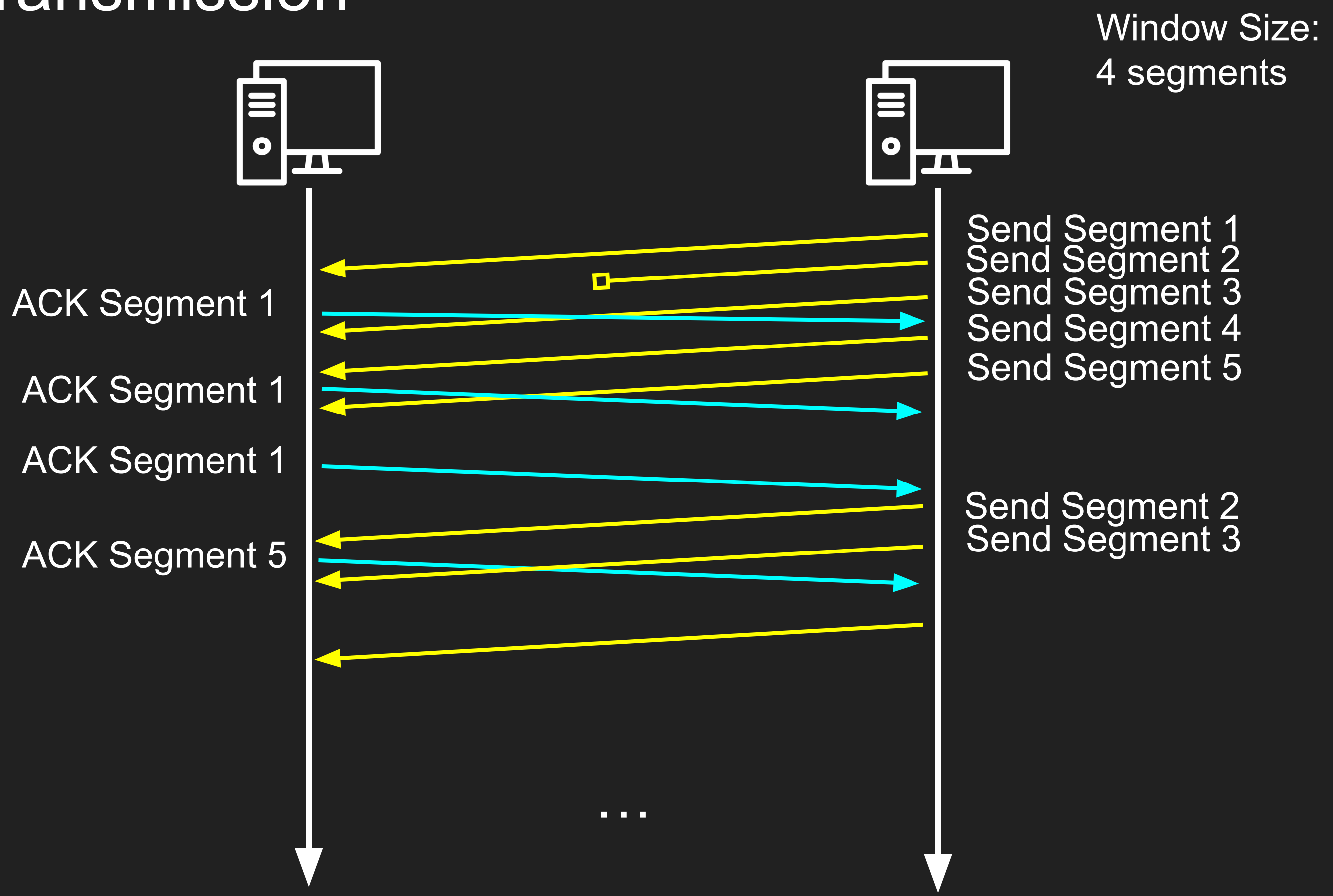
TCP Retransmission



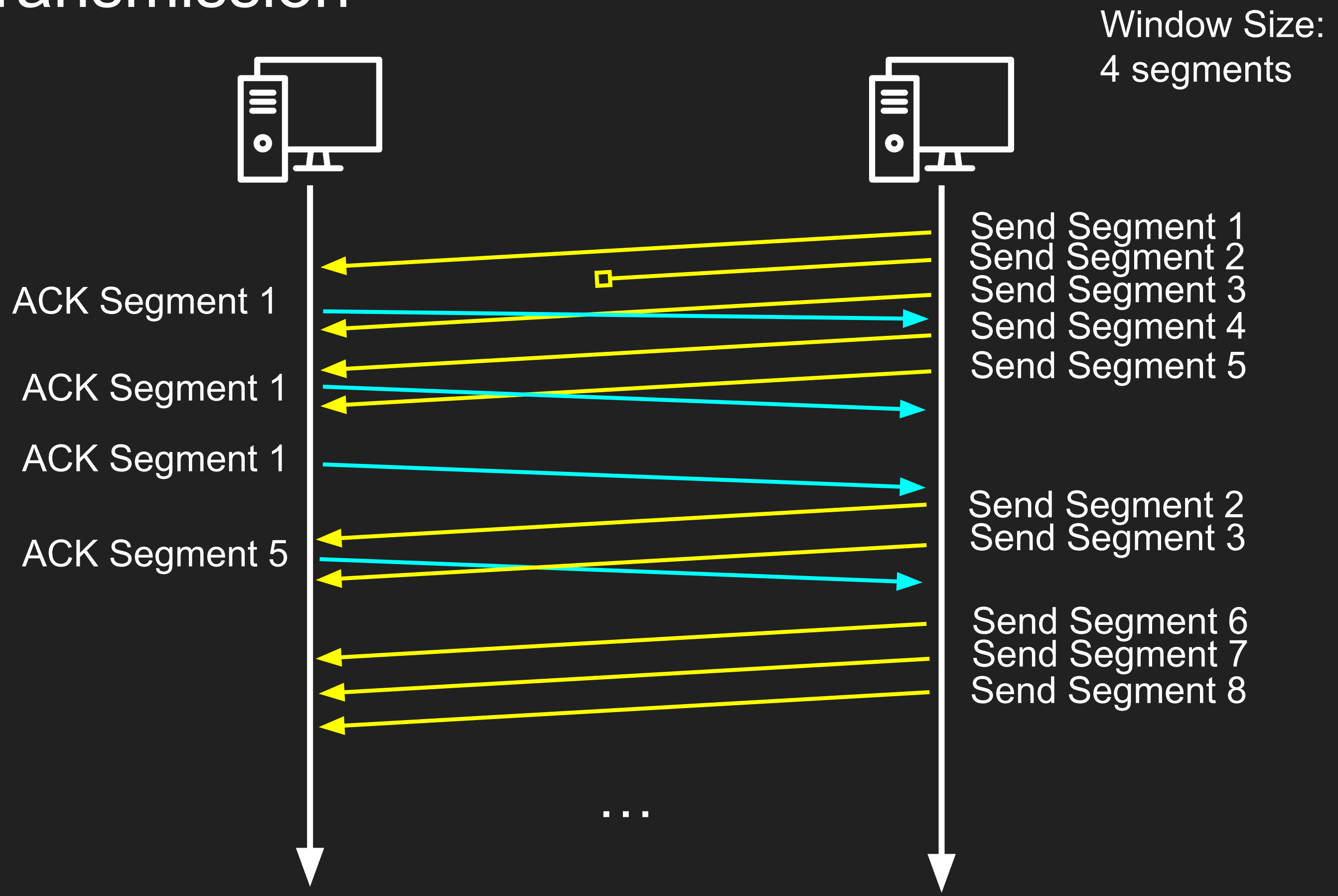
TCP Retransmission



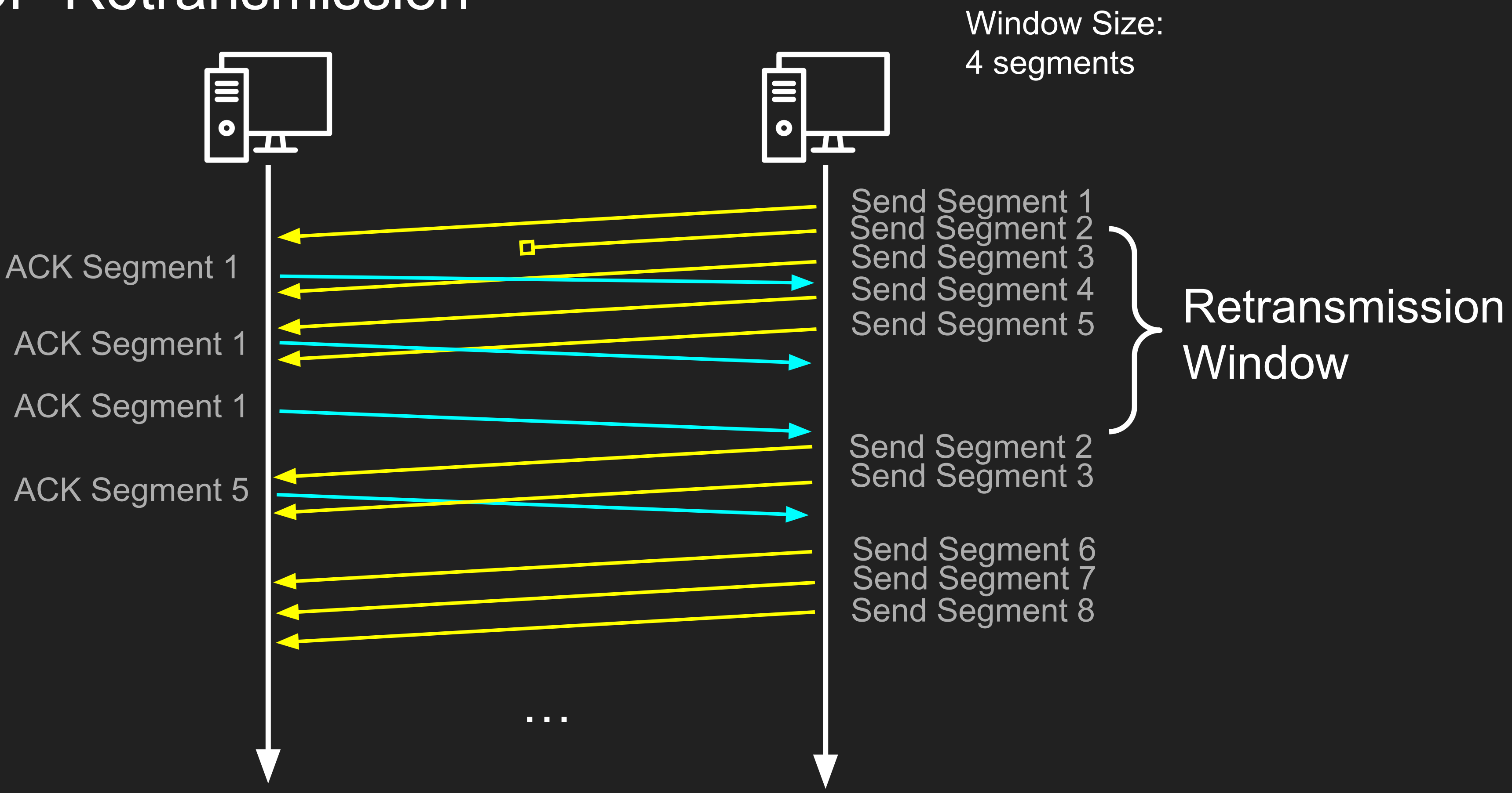
TCP Retransmission



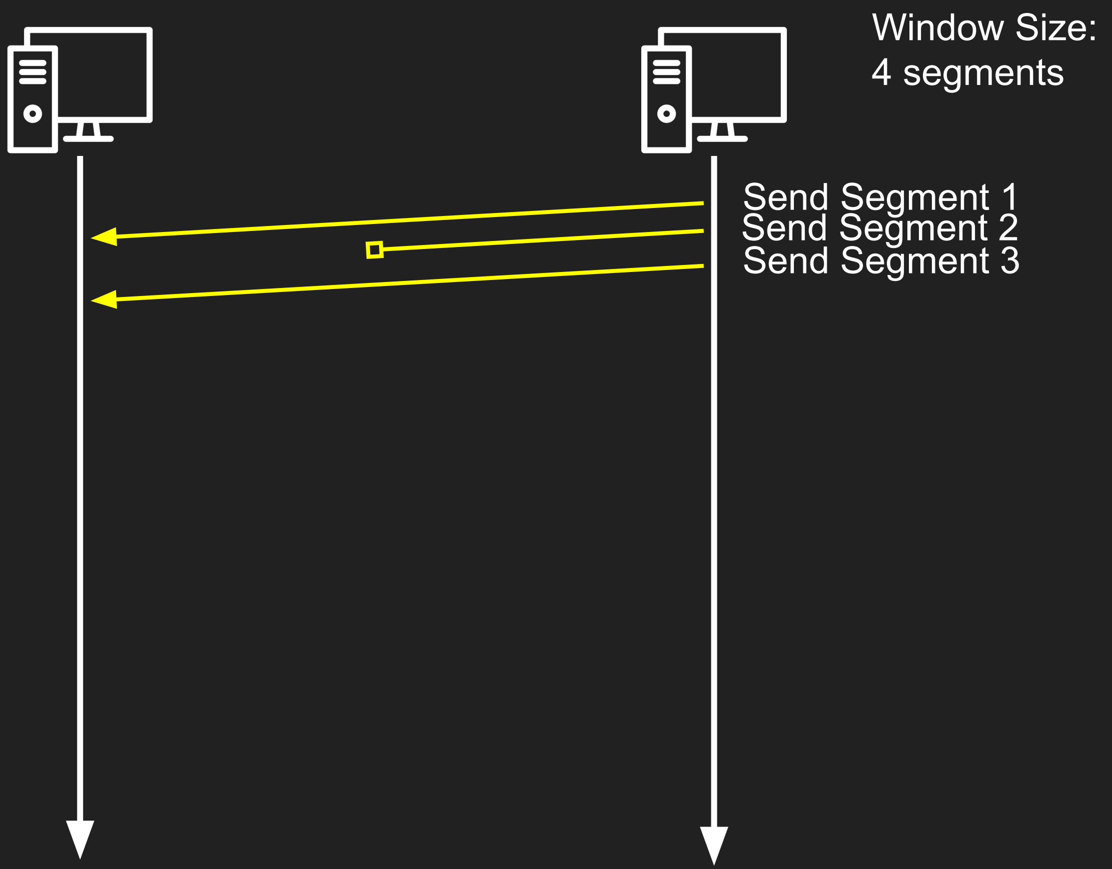
TCP Retransmission



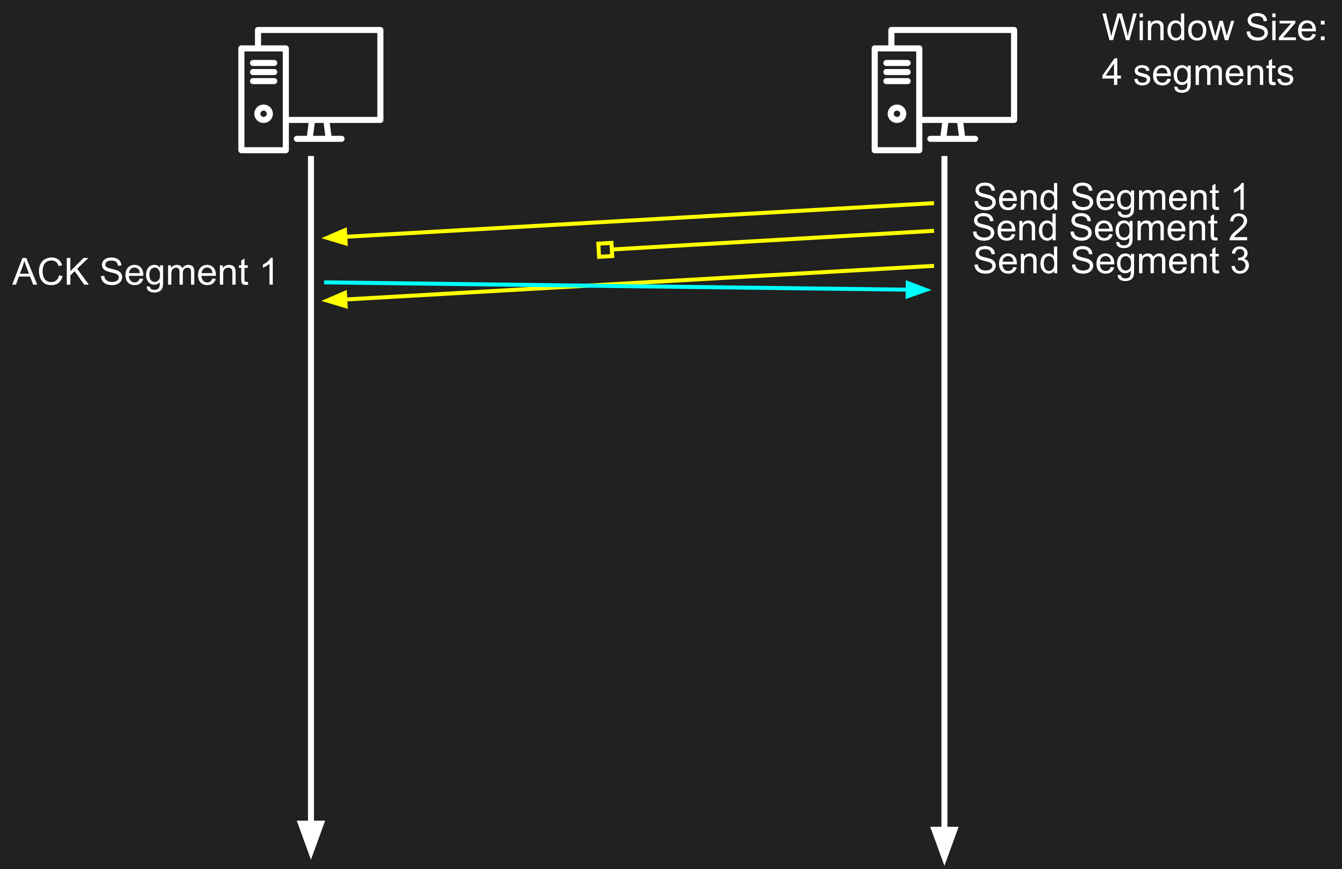
TCP Retransmission



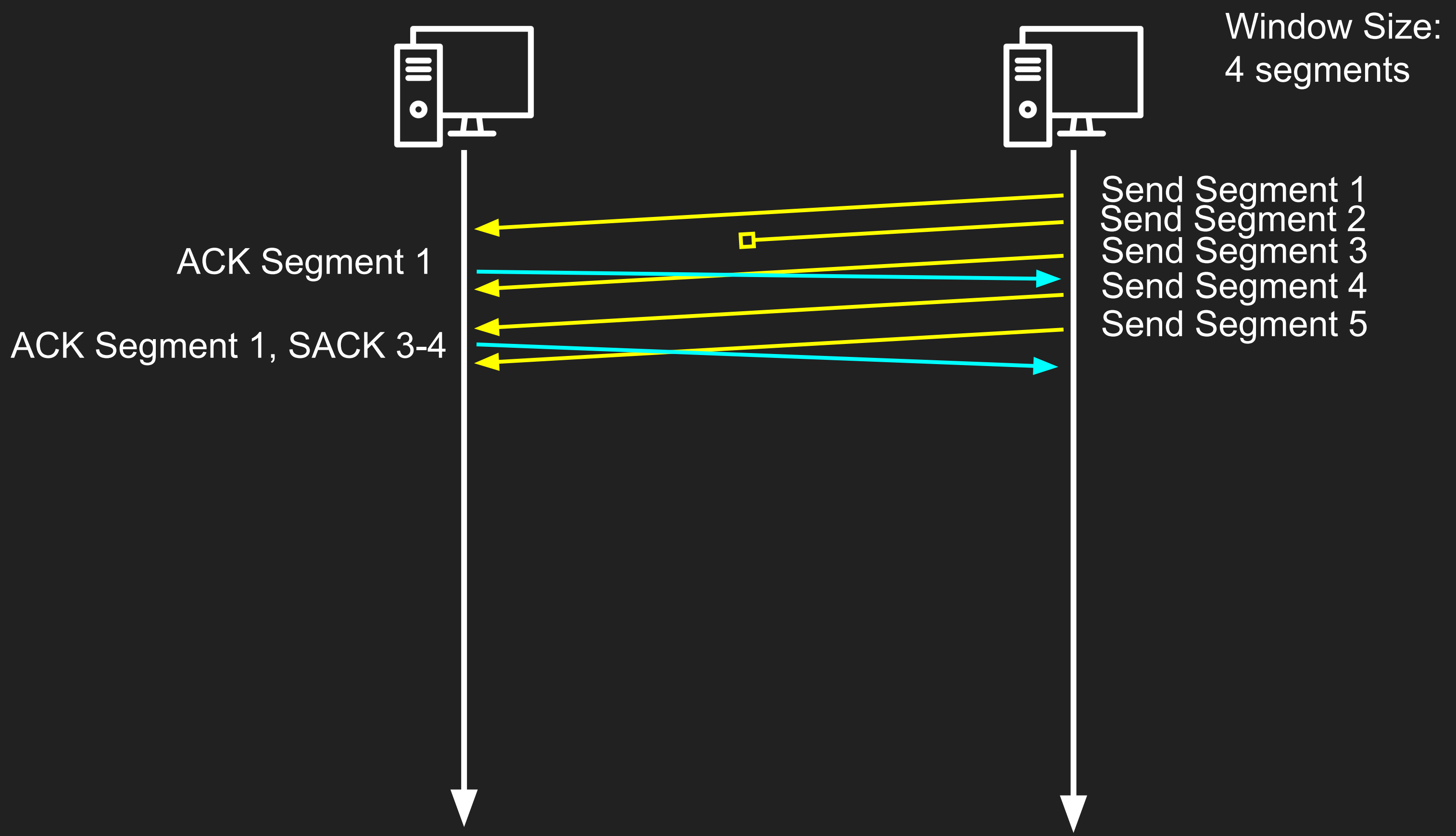
TCP Retransmission: Selective Acknowledgement (SACK)



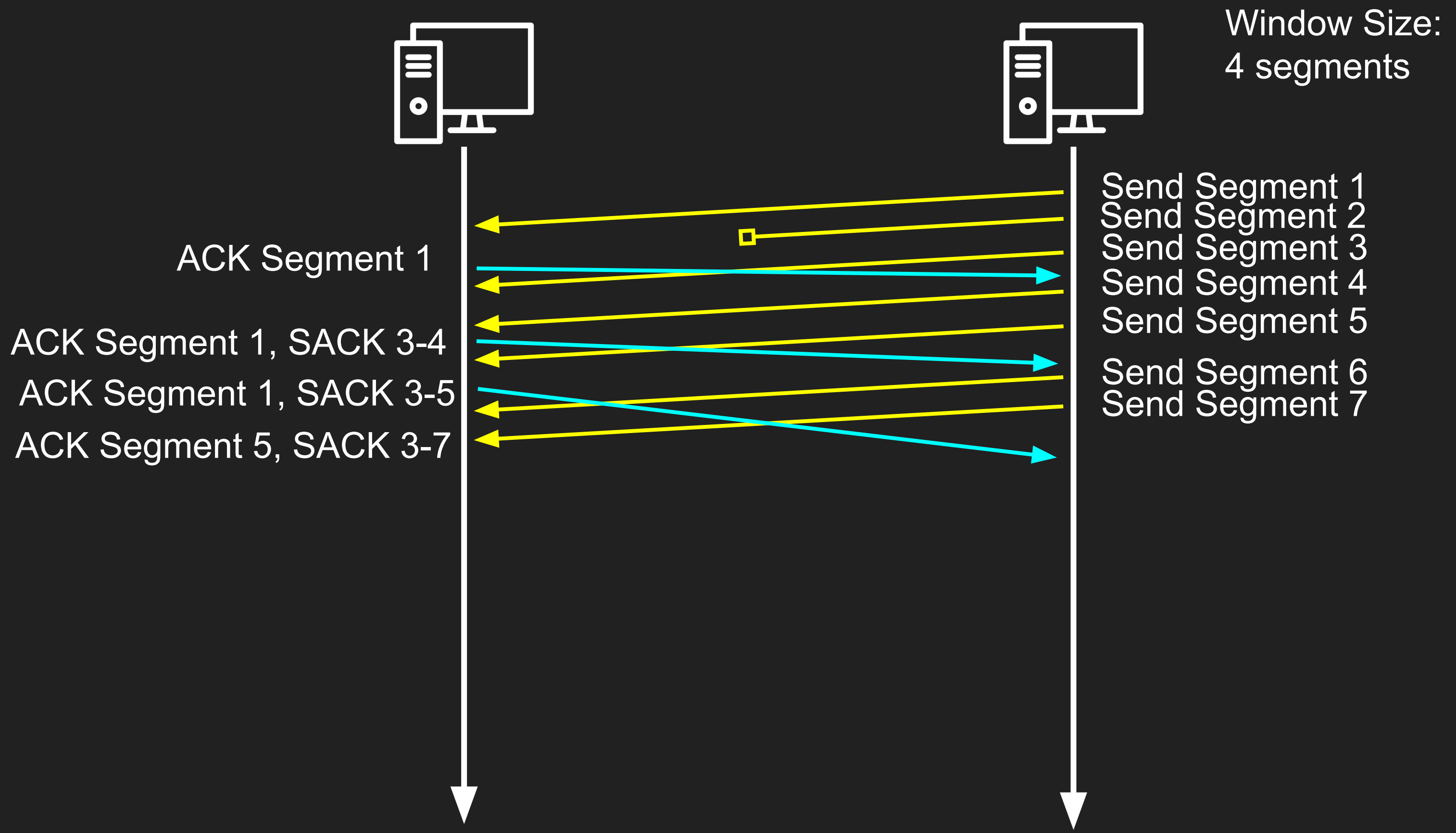
TCP Retransmission: Selective Acknowledgement (SACK)



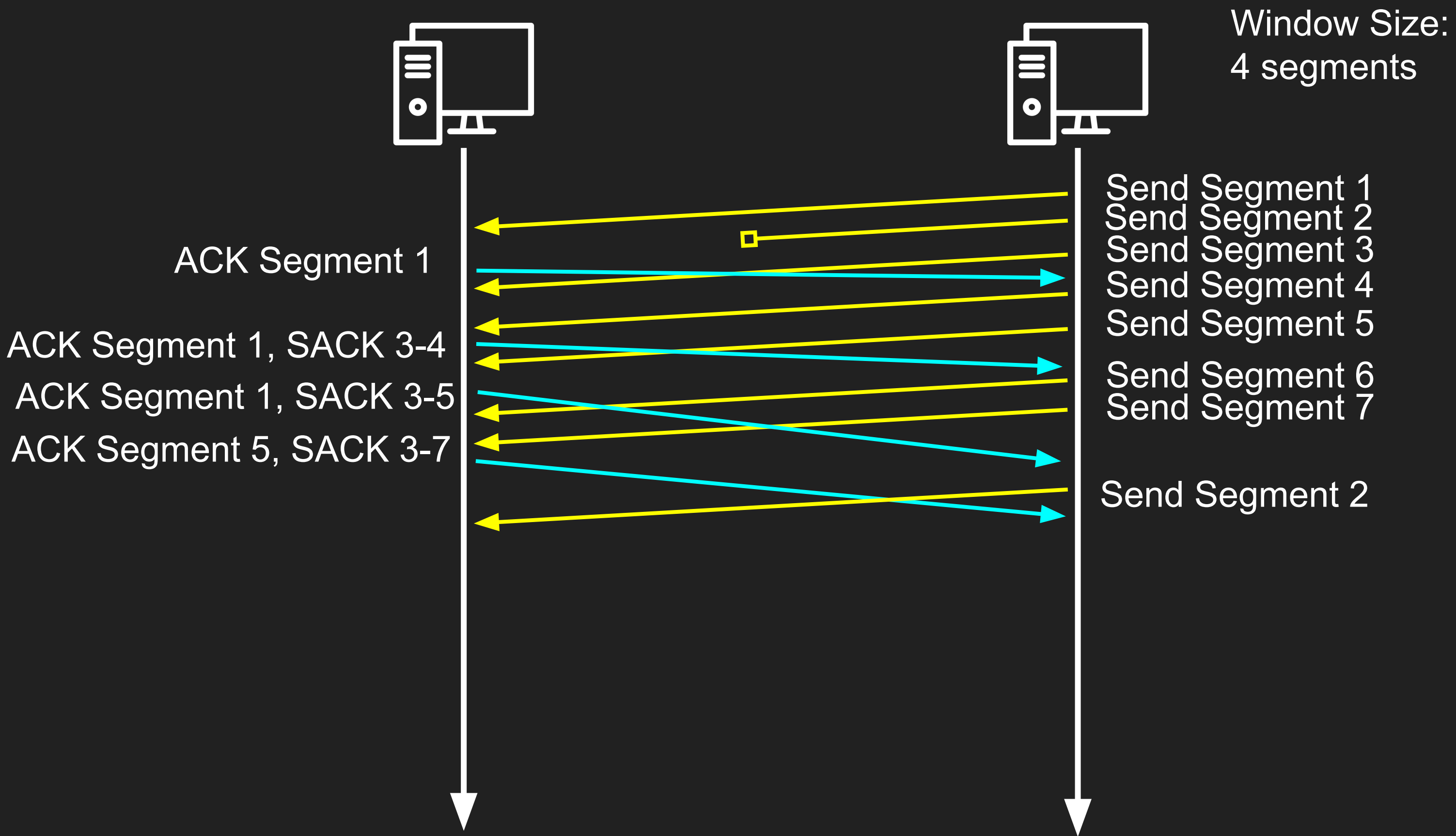
TCP Retransmission: Selective Acknowledgement (SACK)



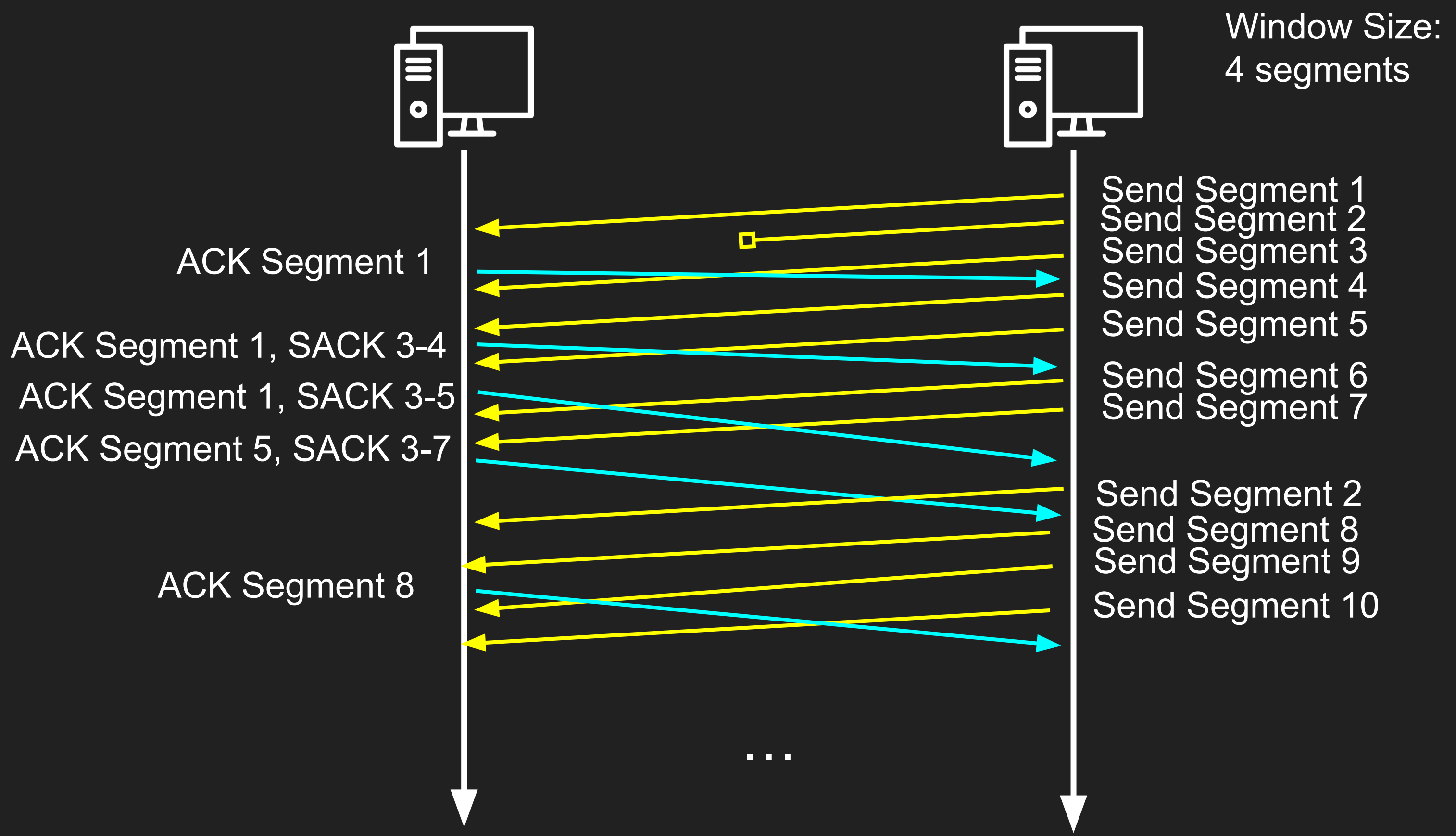
TCP Retransmission: Selective Acknowledgement (SACK)



TCP Retransmission: Selective Acknowledgement (SACK)



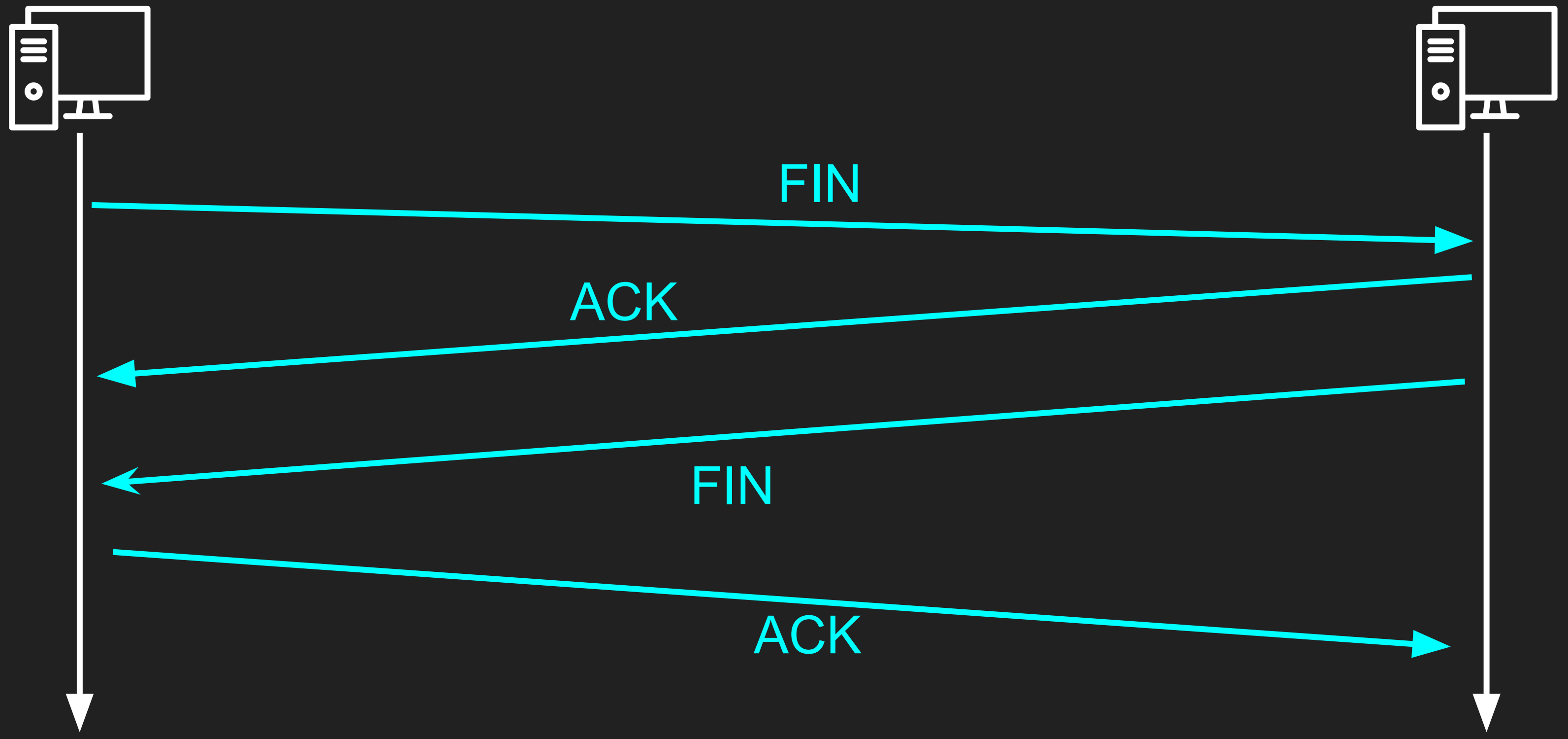
TCP Retransmission: Selective Acknowledgement (SACK)



What happens when an ACK is lost?

Draw scenarios on chalkboard

Polite TCP Connection Termination



TCP is Self-Clocking

Can you believe it? Why?

How TCP Detects Congestion Control and Avoidance

Congestion is detected because TCP is self-clocking

If the clock is running slow, we have congestion

If ACKs are coming in at the same rate as I send segments, I could send at a faster rate

If I am sending more packets than I get ACK’s back I should slow down

TCP Slow Start

An exponential increase in speed as you get started.

Goes until we first see congestion or we reach the SlowStart threshold.

Congestion Control (Normal State)

Normally we increase transmission rate always until we see congestion

In normal mode, we increase speed linearly til congestion

If we timeout without hearing anything, go to slowest possible and do Slow Start

If we get 3 ACKs with the same ACK number, cut speed in half

AIMD Additive Increase Multiplicative Decrease

How does this help share bandwidth fairly?

Game Theory

TCP Flavors

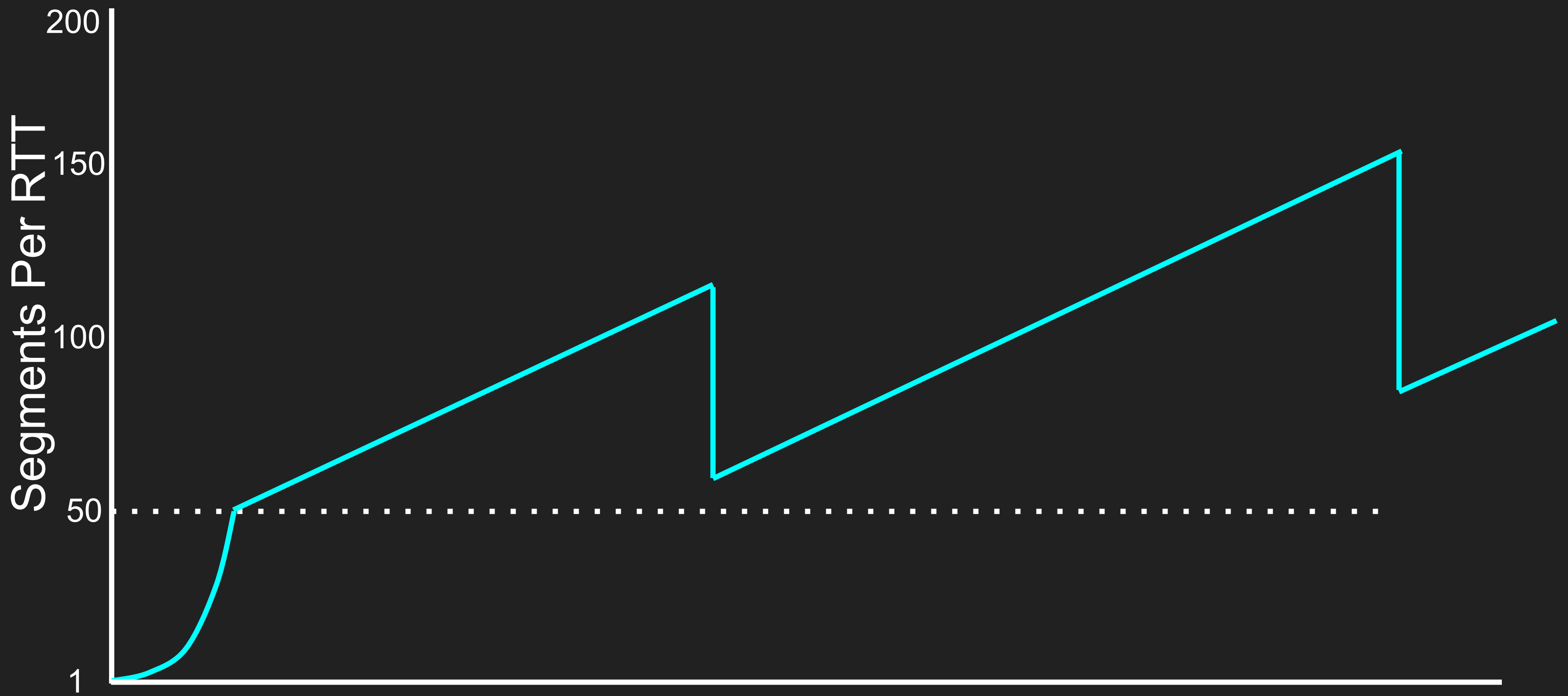
TCP Tahoe: Always goes back to 1 and does slow start

TCP Reno: AIMD

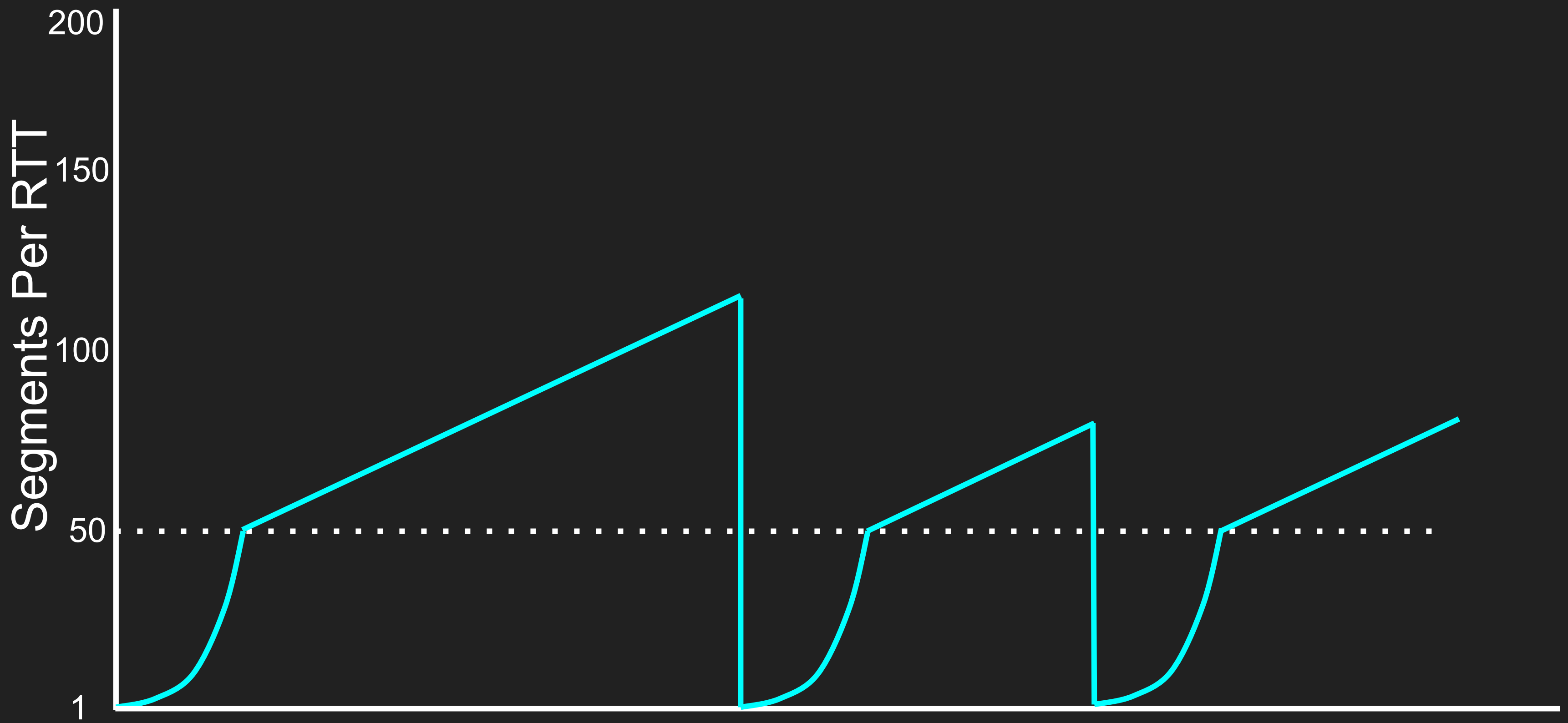
TCP Vegas: Measures RTT, time to get an ACK. Detects congestion faster. Gives up bandwidth to TCP flavors which detect collisions more slowly!

TCP CUBIC: Cubic Increase, Multiplicative Decrease

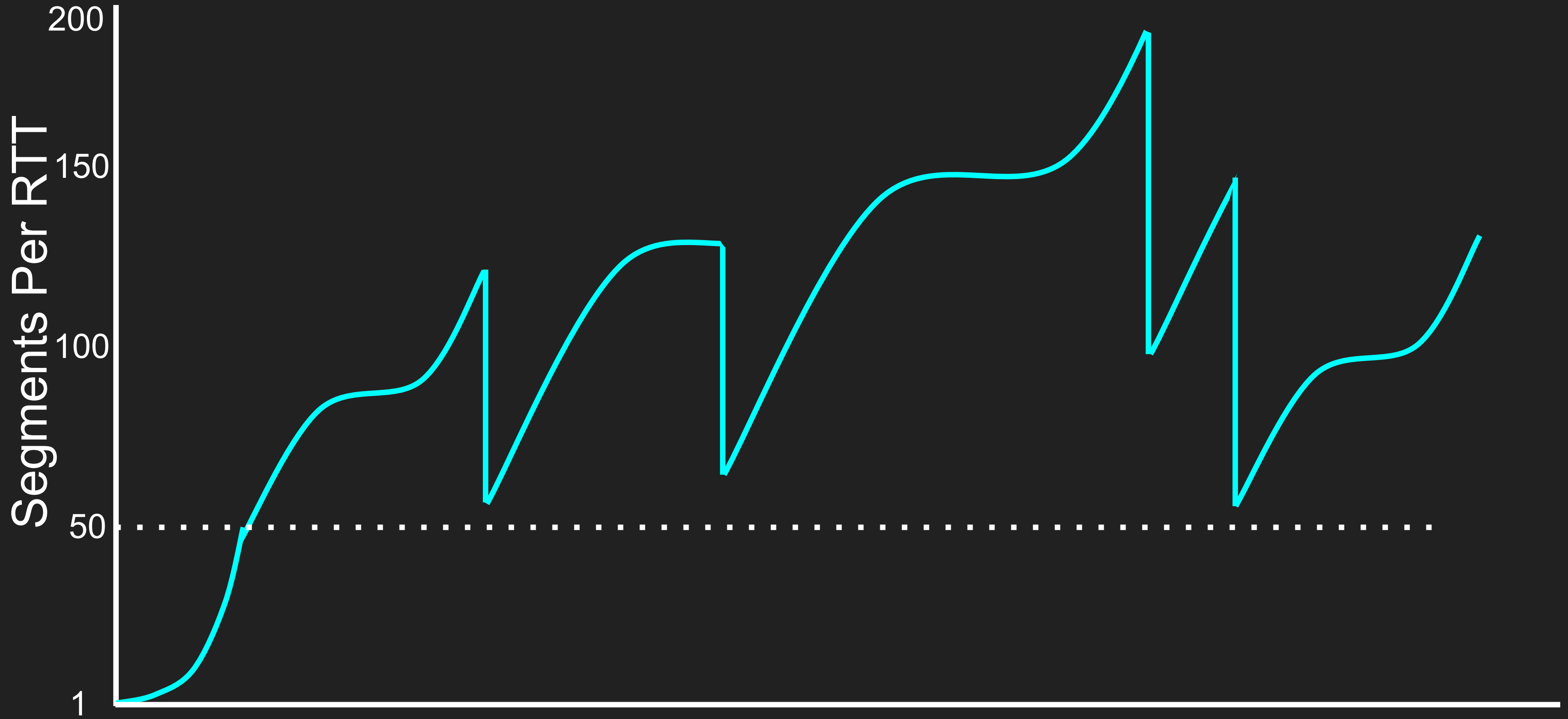
Name that TCP Flavor



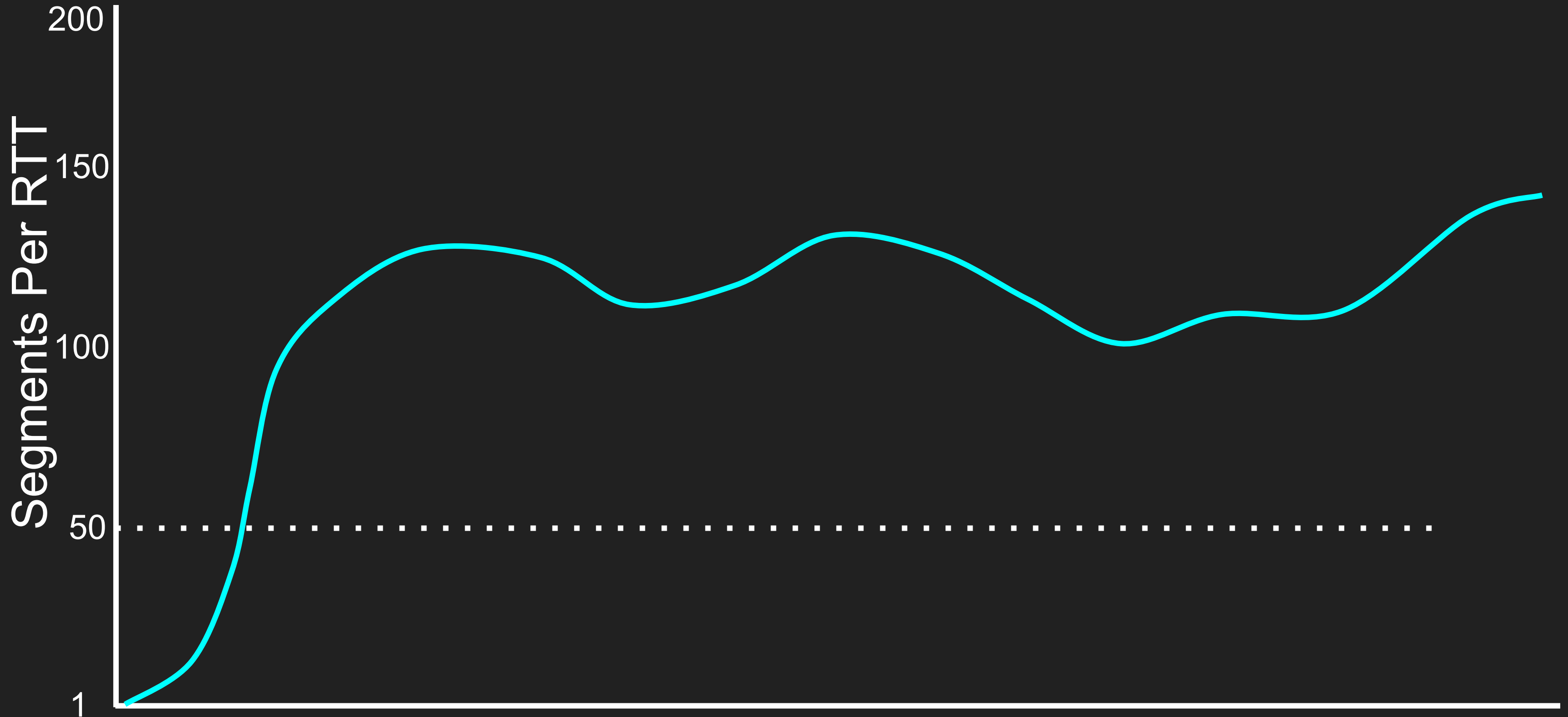
Name that TCP Flavor



Name that TCP Flavor



Name that TCP Flavor



What flavor are you running?

Windows: Reno

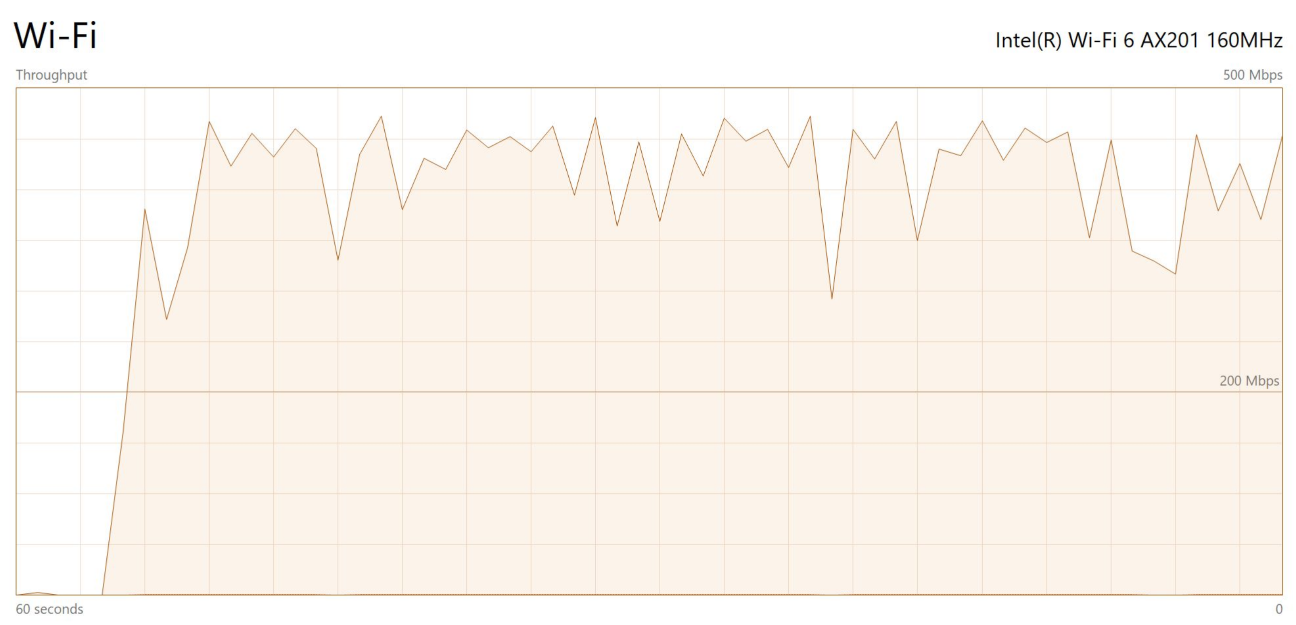
Linux & Macintosh: Cubic

Who will get more bandwidth when Windows and *nix machines share a network?

Why?

What would happen if you switched your machine to TCP Vegas?

Real TCP - Downloading a Large ISO



Sockets

A special type of “file” that connects to a network.

The OS takes care of most Layer 4 tasks so you can write applications on top.

When you “read” from a socket, you are reading bytes sent by the other computer from the OS buffer.

When you “write” to a socket, you are telling the OS to send bytes over TCP to the other computer.

Creating a TCP Socket in Linux

```
#include<sys/socket.h>
int make_socket()
{
    int socket_desc;
    socket_desc = socket(AF_INET , SOCK_STREAM , 0);
    return socket_desc ;
}
```

Creating a TCP Socket in Linux

```
#include<sys/socket.h>
int make_socket()
{
    int socket_desc;
    socket_desc = socket(AF_INET , SOCK_STREAM , 0);
    return socket_desc ;
}
```

IPv4 Address Family

Choose TCP which makes an ordered stream of data

System call to open a socket

Asking Linux to connect to an address over TCP

```
#include<sys/socket.h>

#include <string.h>

int connect_socket()

{

    char* addr = “1.1.1.1:443”;

    int socket = make_socket()

    connect(socket, addr, strlen(addr));

}
```

Asking Linux to accept TCP connections and listen

```
#include<sys/socket.h>

#include <string.h>

int make_listen_socket()

{

    char* addr = “0.0.0.0:80”;

    int socket = make_socket()

    bind(socket, addr, strlen(addr));

    listen(socket, SOMAXCONN);

    return socket;

}
```

Asking Linux to accept TCP connections and listen

```
#include<sys/socket.h>

#include <string.h>

int make_listen_socket()

{

    char* addr = "0.0.0.0:80";

    int socket = make_socket()

    bind(socket, addr, strlen(addr));

    listen(socket, SOMAXCONN);

    return socket;

}
```

Associate this socket with all local addresses, port 80

Do TCP handshakes and queue up connections for me to handle

Handling incoming connections

A thread for every connection - inefficient

A thread for every CPU core, and work from a shared queue (Apache)

A thread for every CPU core, and work from a per-thread event loop of responsibilities (NGINX)

Network Address Translation

How does a router keep track of multiple TCP connections from multiple machines?

Network Address Translation

How does a router keep track of multiple TCP connections from multiple machines?

- Each TCP conversation has a unique identifier:
    - (SRC IP, SRC Port, DST IP, DST Port)
- Each UDP conversation has a unique identifier:
    - (SRC IP, SRC Port, DST IP, DST Port)
- Also translates ICMP addresses (ping, traceroute):
    - (SRC IP, DST IP, Query ID)

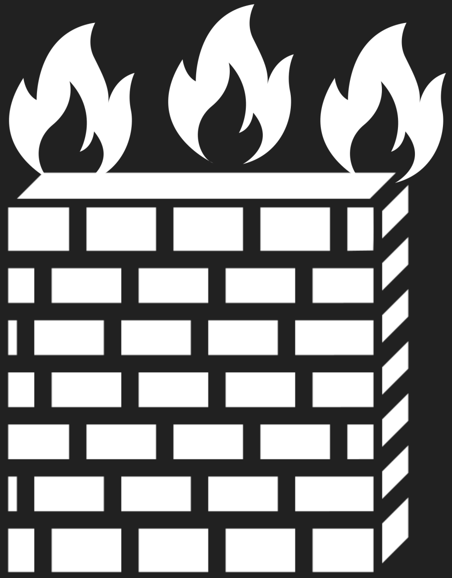
Firewall

Similar to a Routing Table

Rules for IP ranges and Port ranges across TCP & UDP

Implemented in Routers and individual OSes

Rarely is a machine “only” a firewall



Port Forwarding

An option for the NAT to pass traffic through so external TCP connections can be started by a machine outside the firewall.

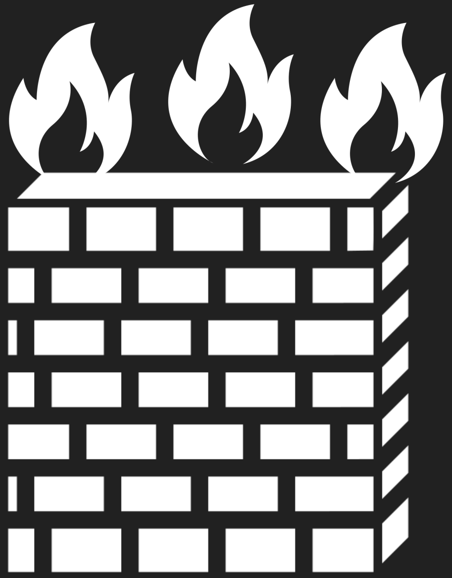
Ex:

SRC IP: ALL

DST Port: 3535

INTERNAL IP: 192.168.1.101

INTERNAL PORT: 22



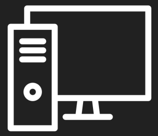
Firewall Rules (ufw) - what do these do?

```
sudo ufw default deny incoming

sudo ufw default allow outgoing

sudo ufw allow ssh

sudo ufw allow 6000:6007/udp

sudo ufw allow from 203.0.113.0/24

sudo ufw allow in on eth0 to any port 80

sudo ufw deny from 203.0.113.4
```

Midterm Exam Coming – No class 26th or 28th

Midterm will be in class, timed to match class (75min)

Image Credits

Computer by donkey_21 from Noun Project 

Fire by YANDI RS from Noun Project 

wall by arif fajar yulianto from Noun Project