CSCI 466: Networks

UDP and TCP

Reese Pearsall

Fall 2024

Announcements

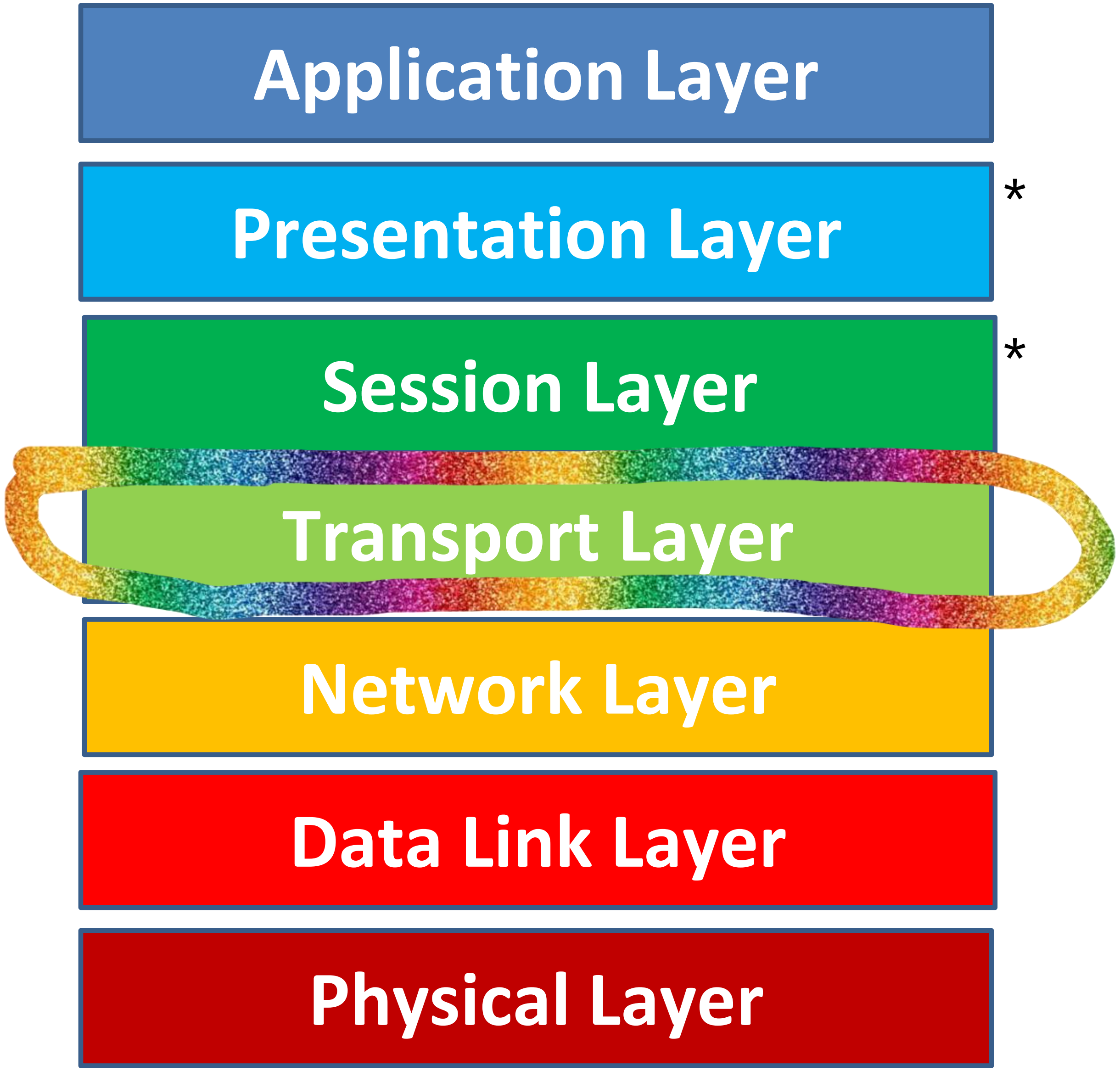
PA 2 Posted. Due Sunday October 13th

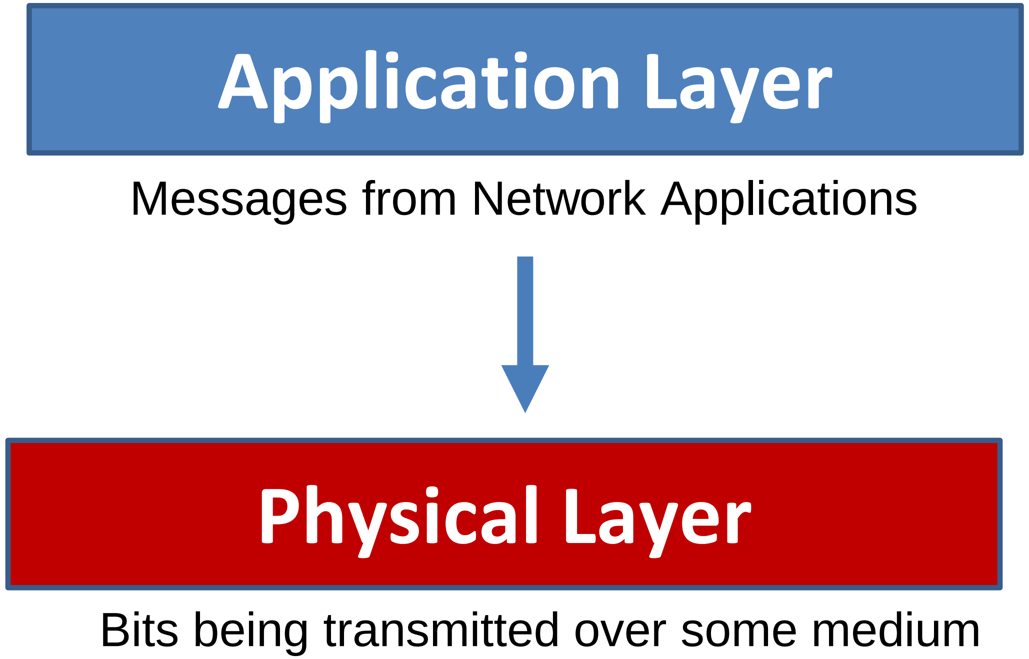
Wireshark Lab 2 Posted. Due Wednesday October 2nd

PA2 Demo

(time.sleep)

OSI Model




Application Layer
Messages from Network Applications
Physical Layer
Bits being transmitted over some medium

**In the textbook, they condense it to a 5-layer model, but 7 layers is what is most used*

Transport Layer

Transport Layer Protocols:

1. Transmission Control Protocol (TCP)
2. User Datagram Protocol (UDP)

UDP

- "no frills," "bare bones" Internet transport protocol
- "best effort" service, UDP segments may be:
  - lost
  - delivered out-of-order to app
- *connectionless:*
  - no handshaking between UDP sender, receiver
  - each UDP segment handled independently of others

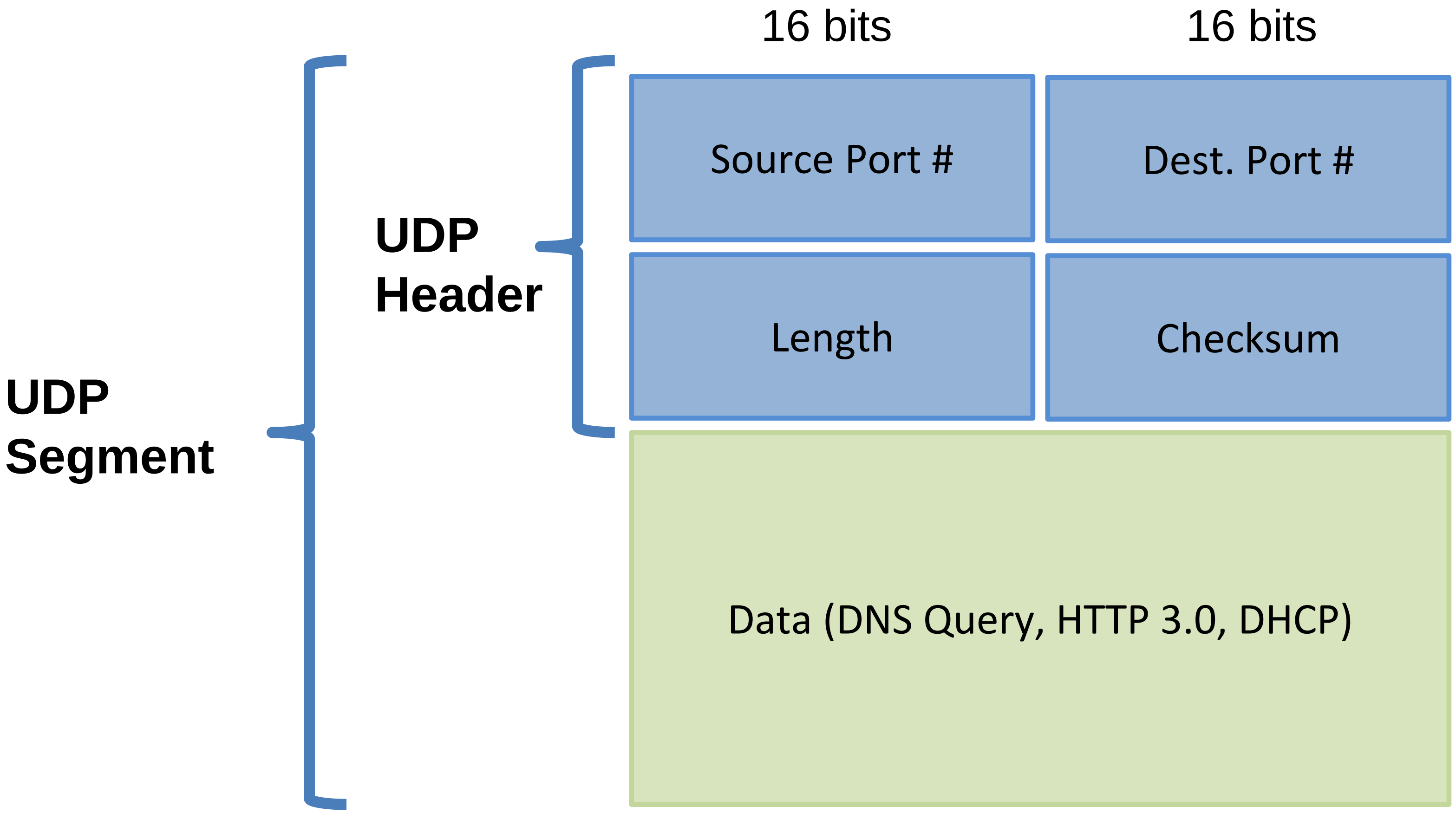
Why is there a UDP?

- no connection establishment (which can add RTT delay)
- simple: no connection state at sender, receiver
- small header size
- no congestion control
  - UDP can blast away as fast as desired!
  - can function in the face of congestion

Transport Layer

UDP

The UDP header is very small!! (8 bytes, 64 bits)



UDP Segment

UDP Header

16 bits

16 bits

Source Port #

Dest. Port #

Length

Checksum

Data (DNS Query, HTTP 3.0, DHCP)

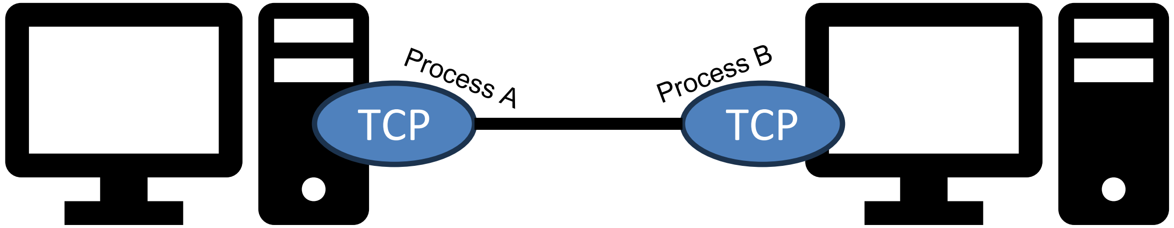
Transport Layer

TCP

- Connection oriented, point-to-point (1 to 1)

→ **TCP Handshake** must occur before data is being transmitted

A *logical* connection



- Reliable, in order, data transfer

- Cumulative ACKs

- Pipelining

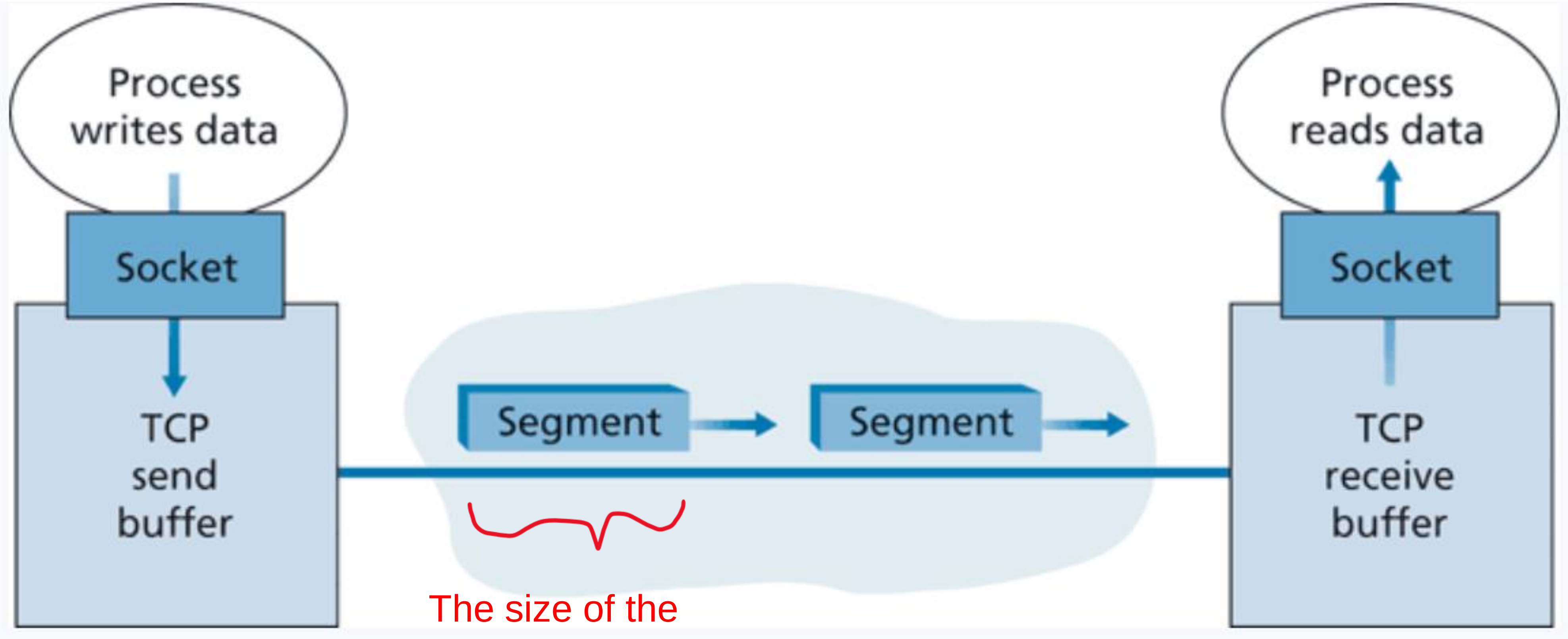
→ TCP Congestion and flow control set window size

- Flow controlled

→ Sender will not overwhelm receiver

- Full-duplex service

TCP



The size of the
segment is determined
by the maximum
segment size (MSS)

(Roughly 1500 bytes– size of a link layer frame)

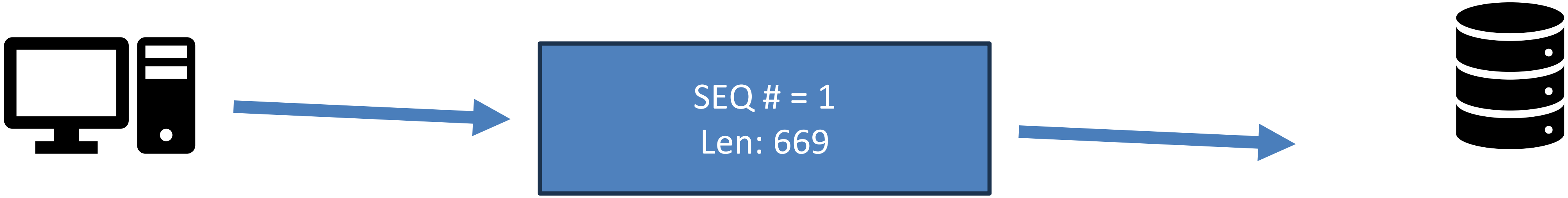
TCP Sequence Numbers

A TCP connection is transmitting a **byte stream**

Sequence numbers are based on *how much data has been sent*
Acknowledgement numbers are based on *how much data has been successfully received*

TCP Sequence Numbers

A TCP connection is transmitting a **byte stream**

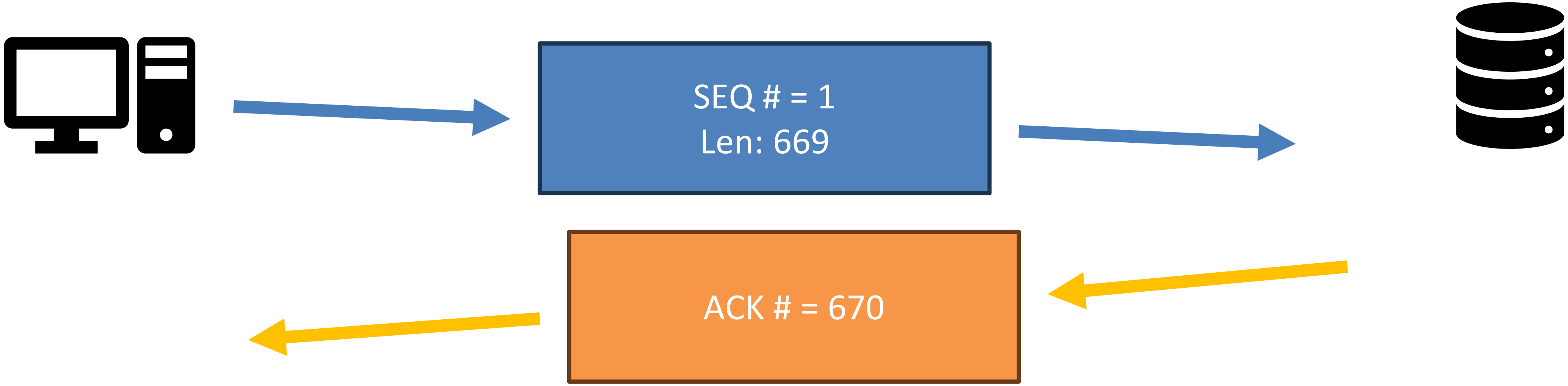


SEQ # = 1
Len: 669

Transport Layer

TCP Sequence Numbers

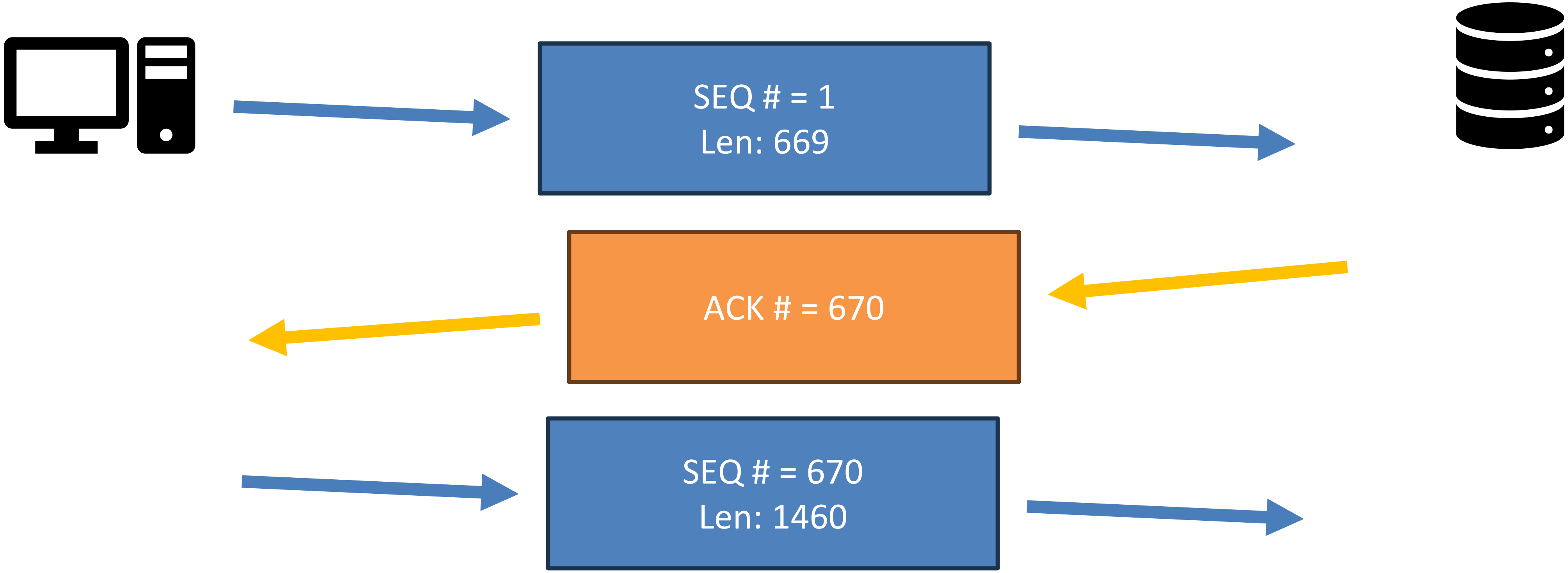
A TCP connection is transmitting a **byte stream**



Transport Layer

TCP Sequence Numbers

A TCP connection is transmitting a **byte stream**



SEQ # = 1
Len: 669

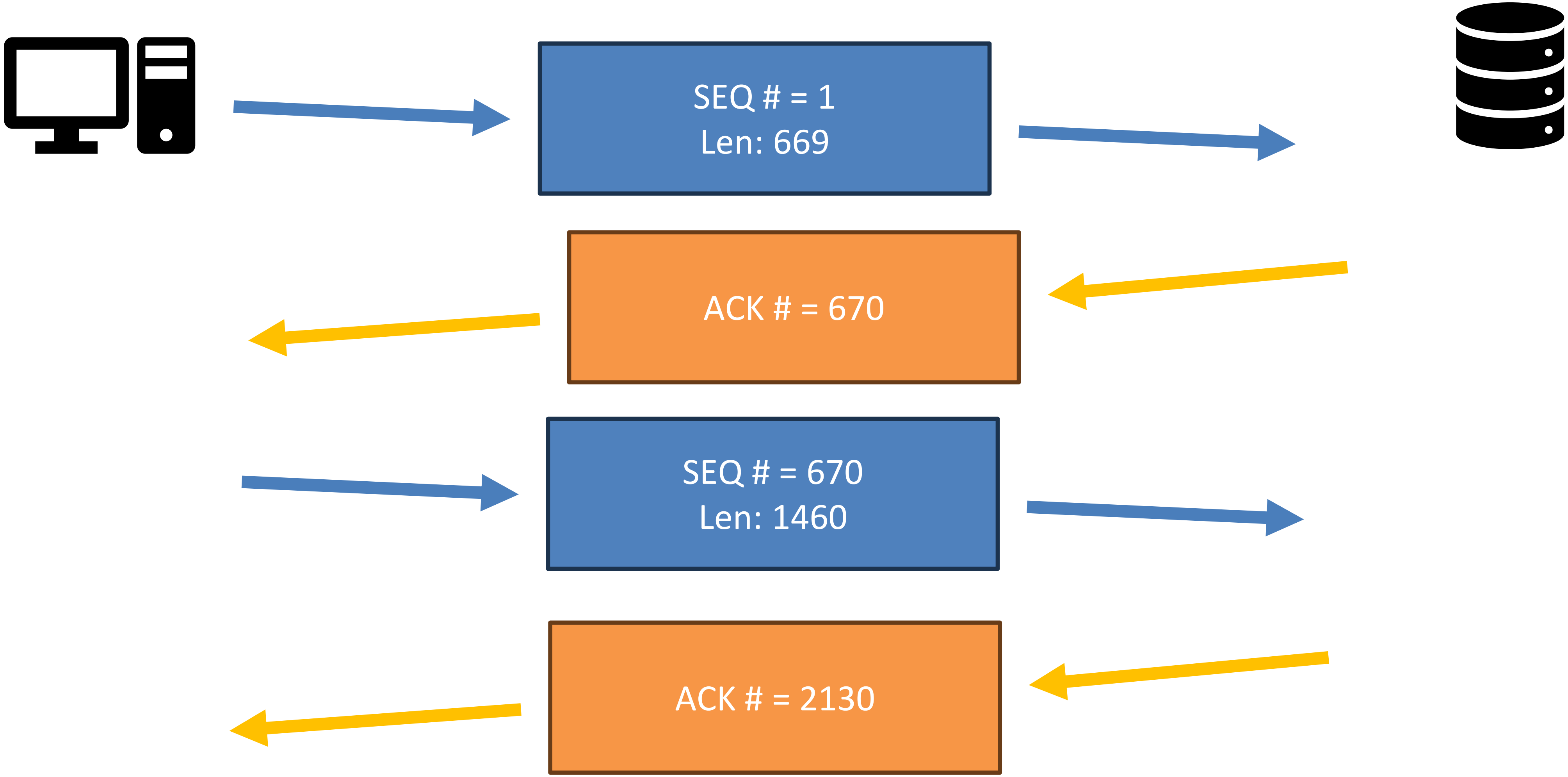
ACK # = 670

SEQ # = 670
Len: 1460

Transport Layer

TCP Sequence Numbers

A TCP connection is transmitting a **byte stream**



SEQ # = 1
Len: 669

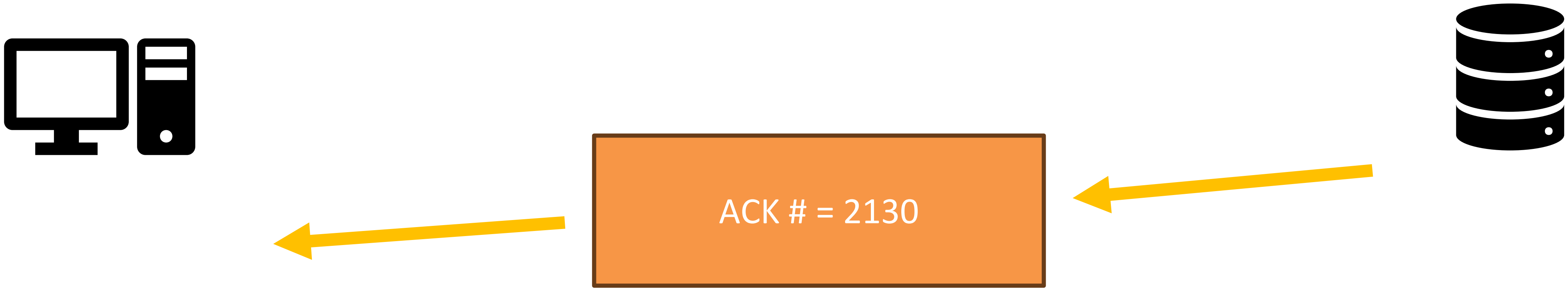
ACK # = 670

SEQ # = 670
Len: 1460

ACK # = 2130

TCP Sequence Numbers

A TCP connection is transmitting a **byte stream**

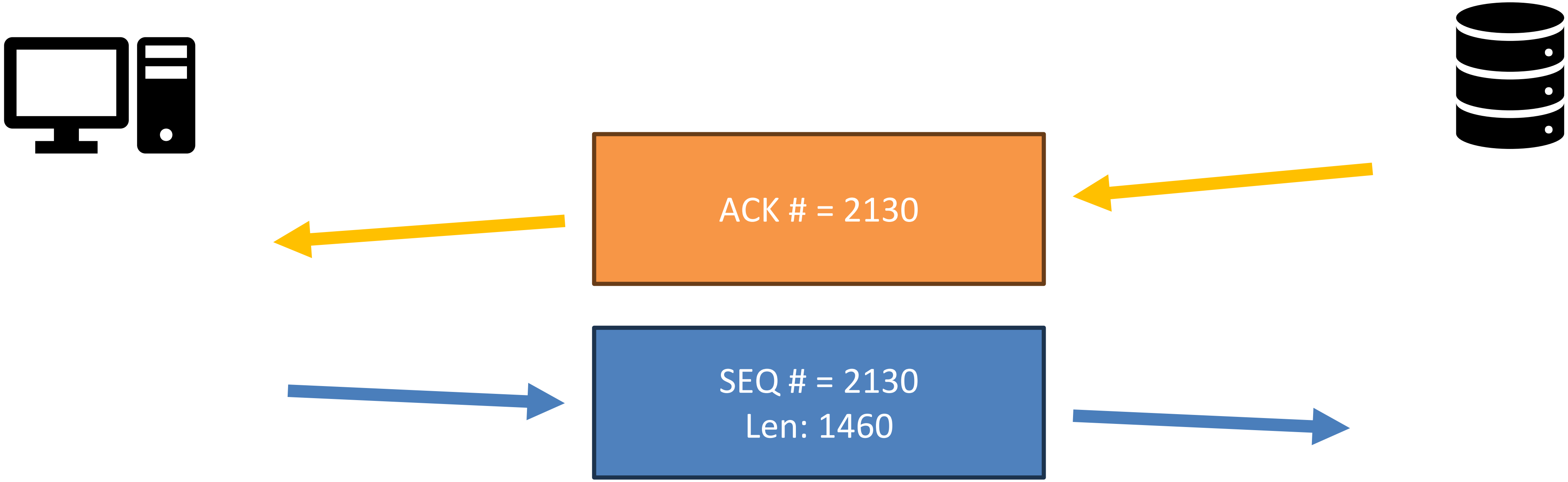


ACK # = 2130

Transport Layer

TCP Sequence Numbers

A TCP connection is transmitting a **byte stream**

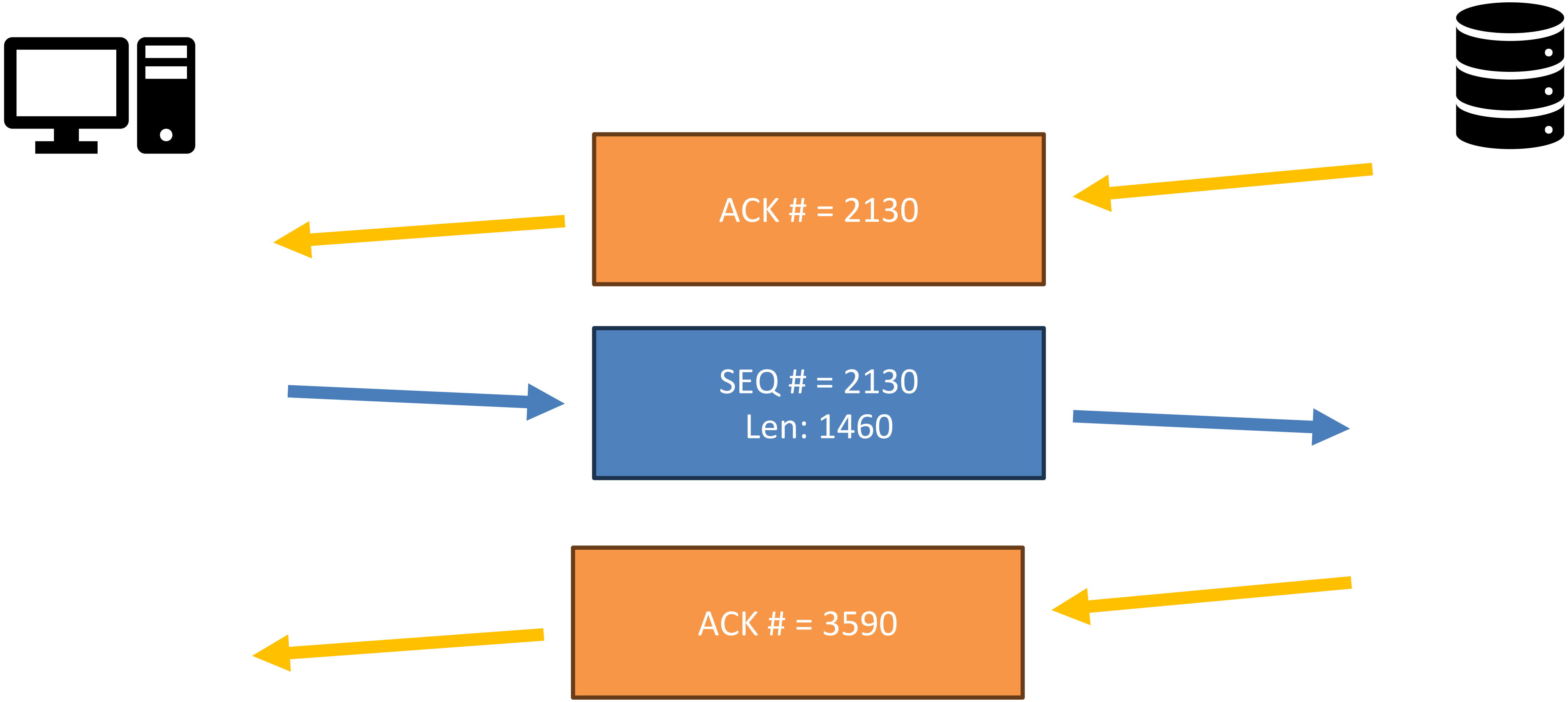


ACK # = 2130

SEQ # = 2130
Len: 1460

TCP Sequence Numbers

A TCP connection is transmitting a **byte stream**

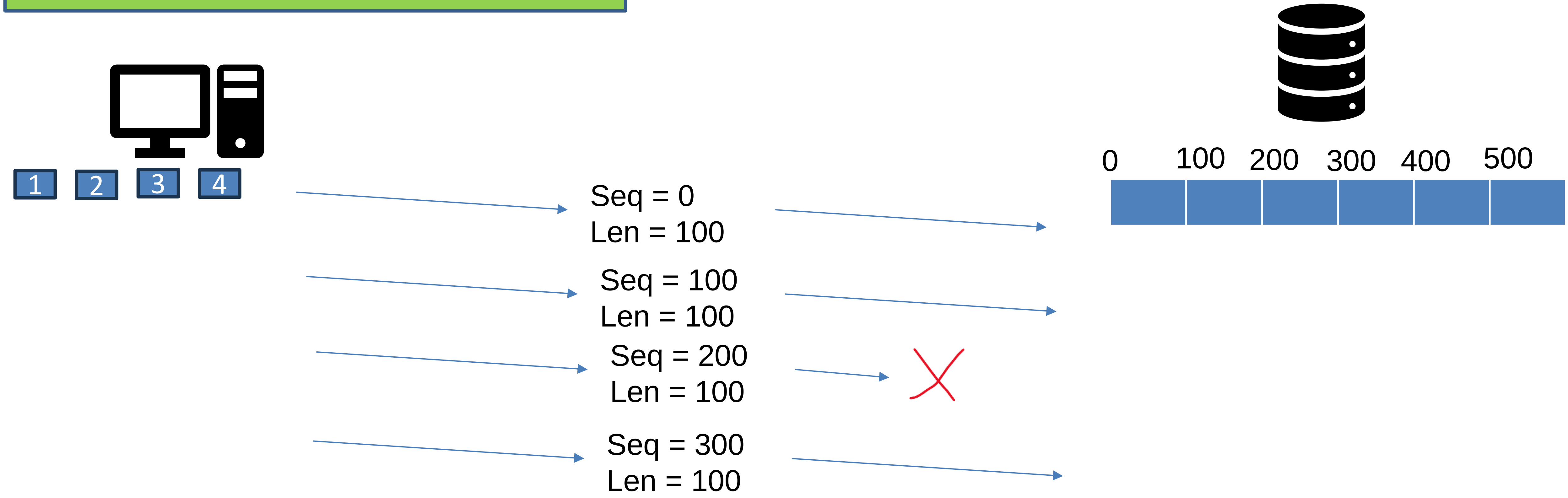


ACK # = 2130

SEQ # = 2130
Len: 1460

ACK # = 3590

TCP ACKs *(pipelining)*



1 2 3 4

0 100 200 300 400 500

Seq = 0
Len = 100

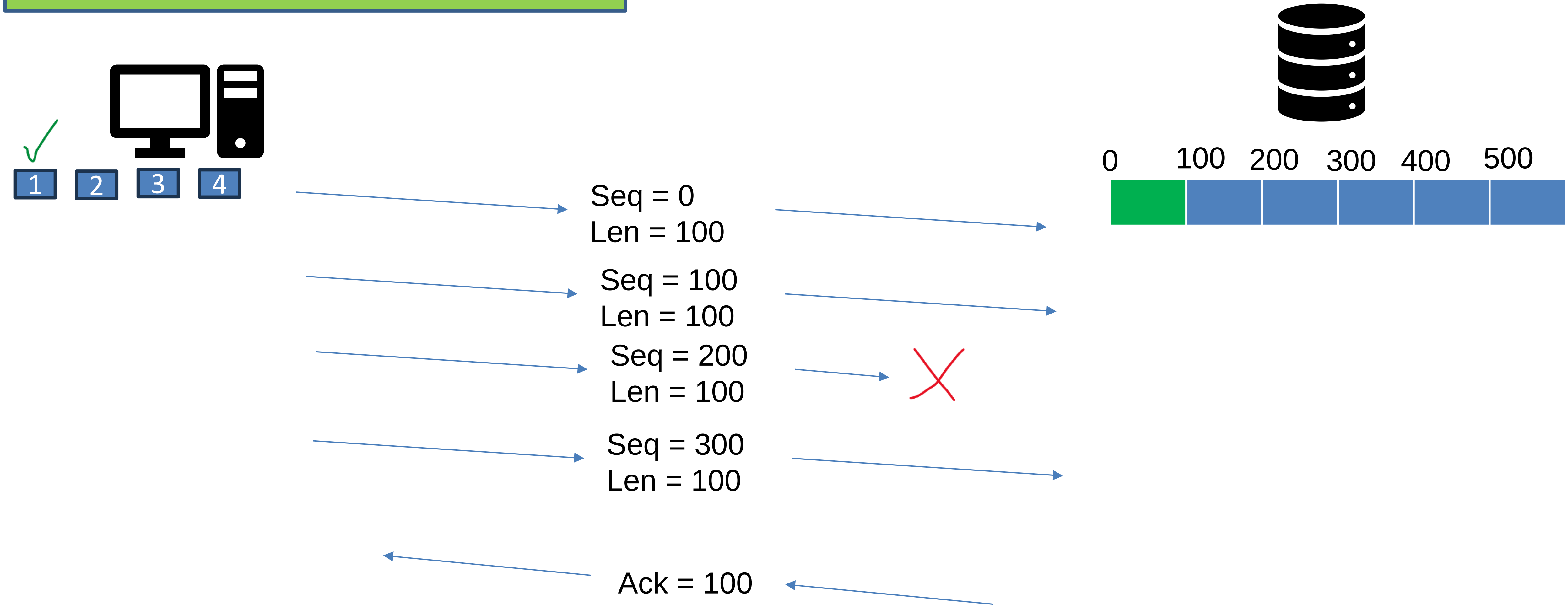
Seq = 100
Len = 100

Seq = 200
Len = 100

Seq = 300
Len = 100

Transport Layer

TCP ACKs *(pipelining)*



Seq = 0
Len = 100

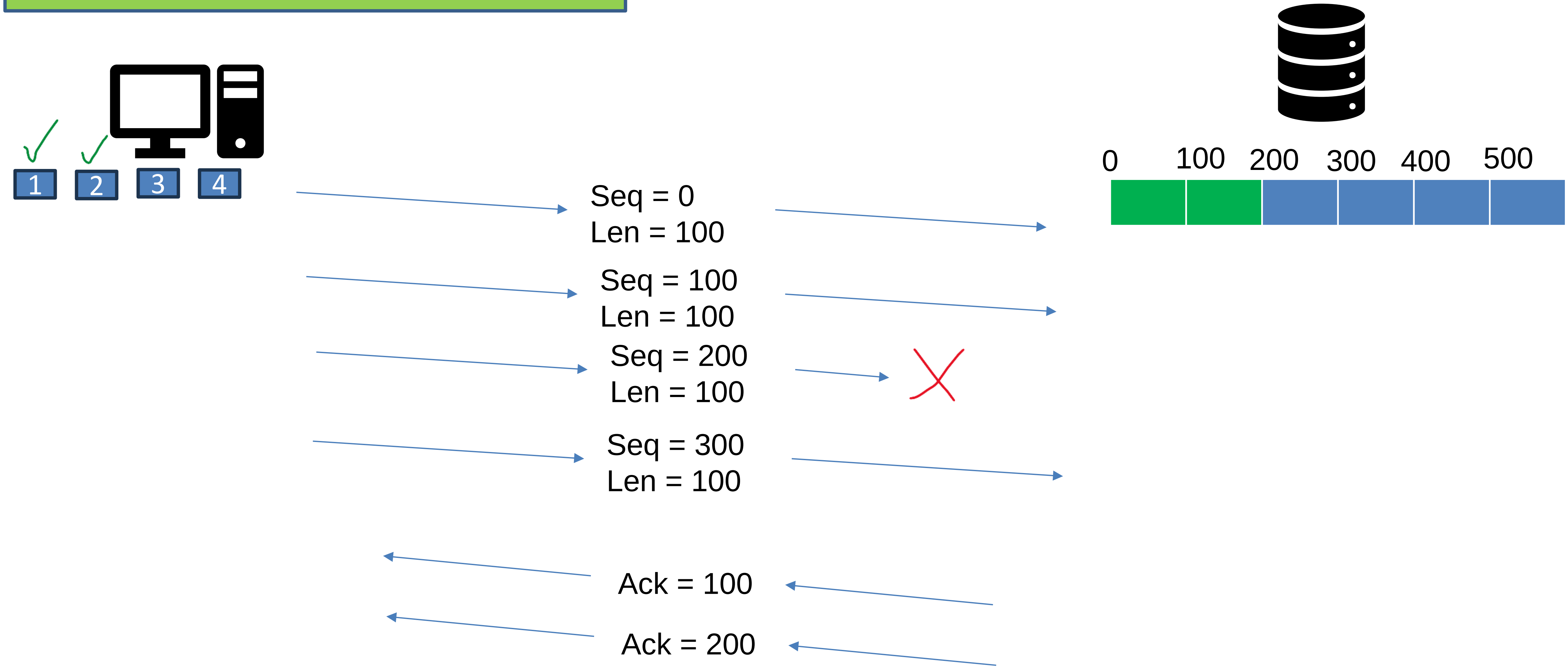
Seq = 100
Len = 100

Seq = 200
Len = 100

Seq = 300
Len = 100

Ack = 100

TCP ACKs *(pipelining)*



Seq = 0
Len = 100

Seq = 100
Len = 100

Seq = 200
Len = 100

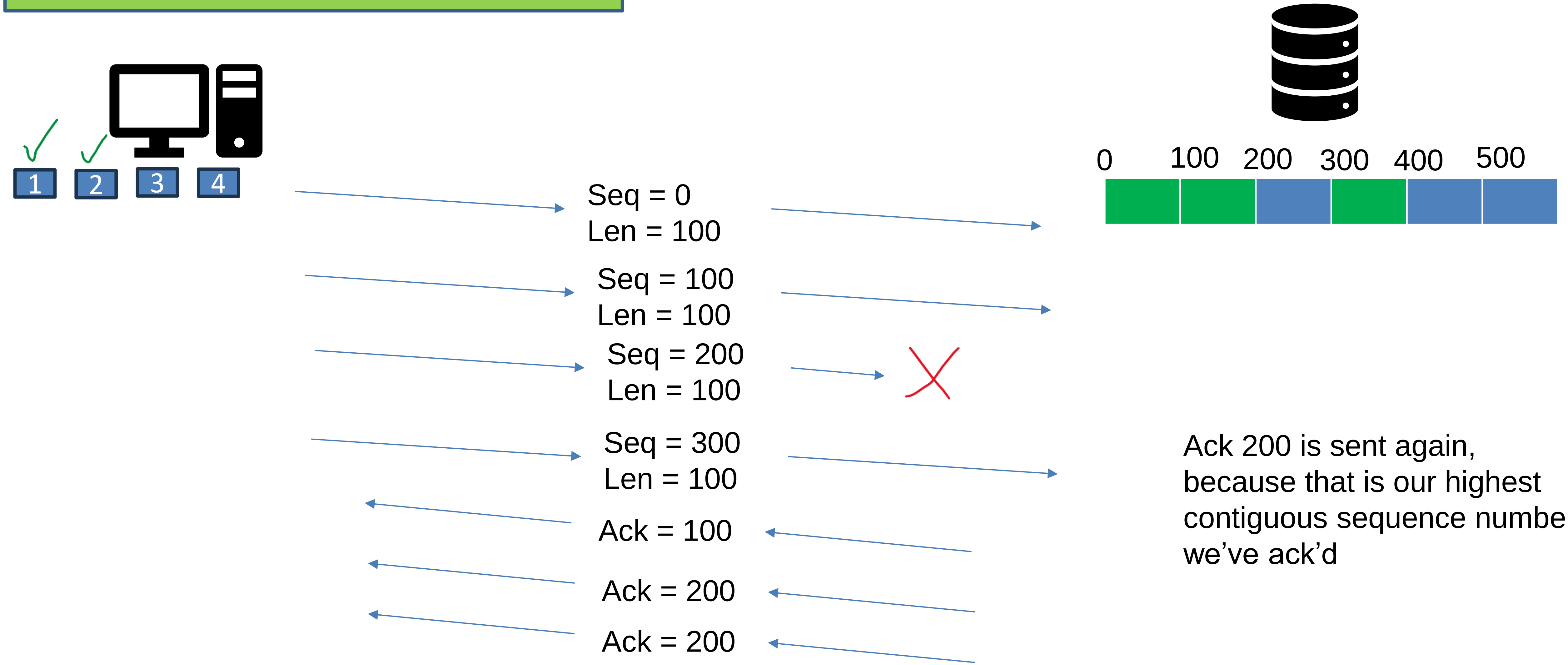
Seq = 300
Len = 100

Ack = 100

Ack = 200

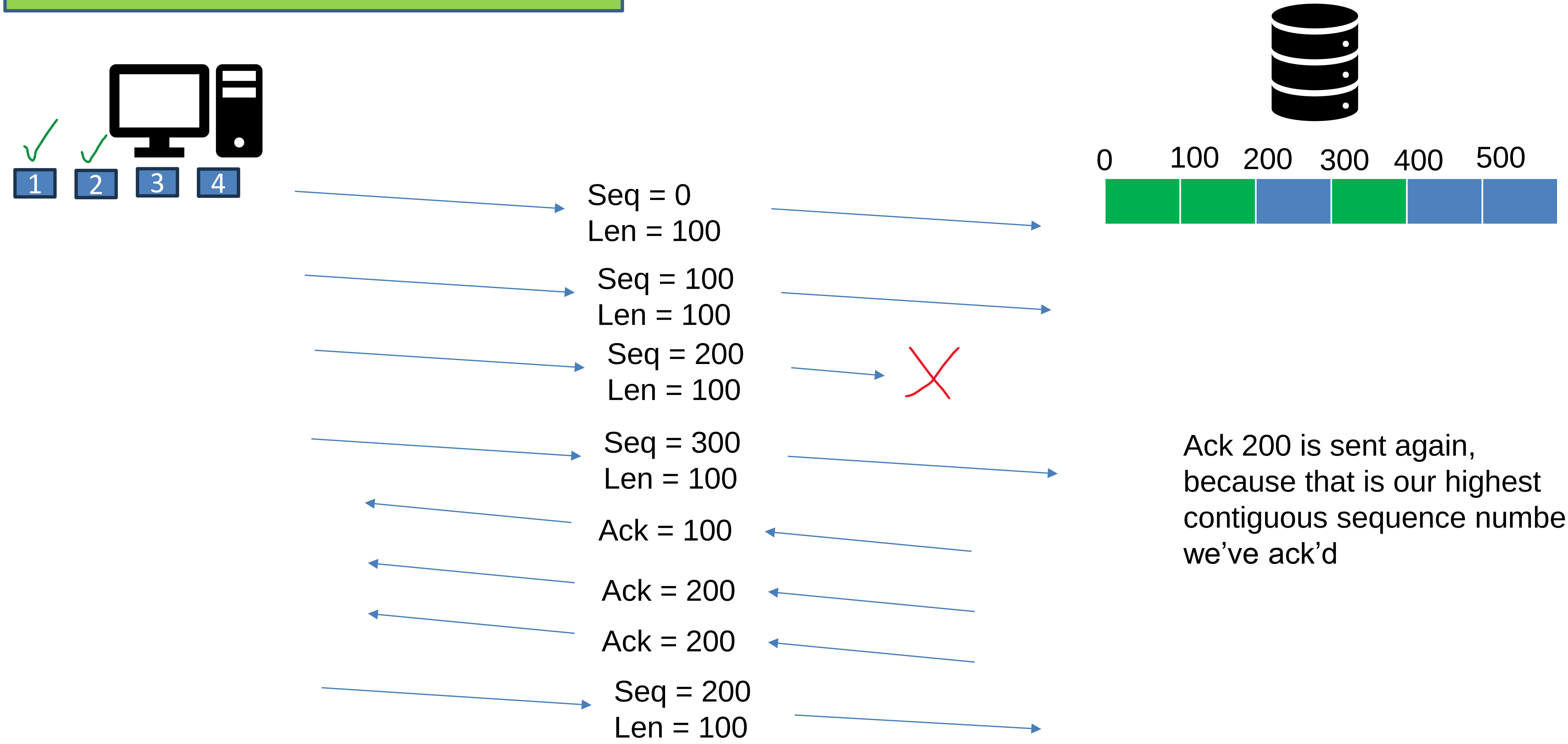
Transport Layer

TCP ACKs *(pipelining)*



Ack 200 is sent again, because that is our highest contiguous sequence number we've ack'd

TCP ACKs *(pipelining)*



Seq = 0
Len = 100

Seq = 100
Len = 100

Seq = 200
Len = 100

Seq = 300
Len = 100

Ack = 100

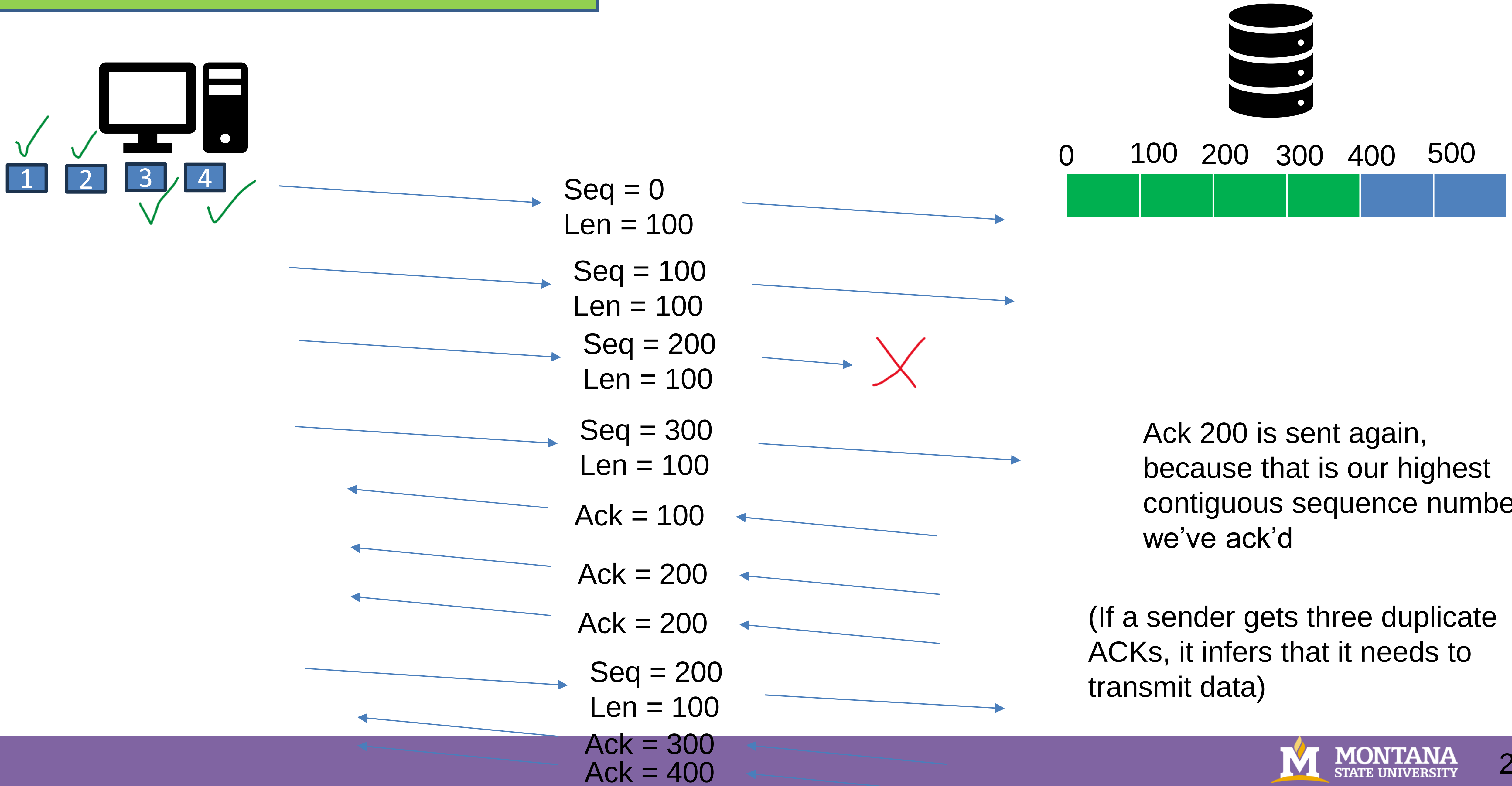
Ack = 200

Ack = 200

Seq = 200
Len = 100

Ack 200 is sent again, because that is our highest contiguous sequence number we've ack'd

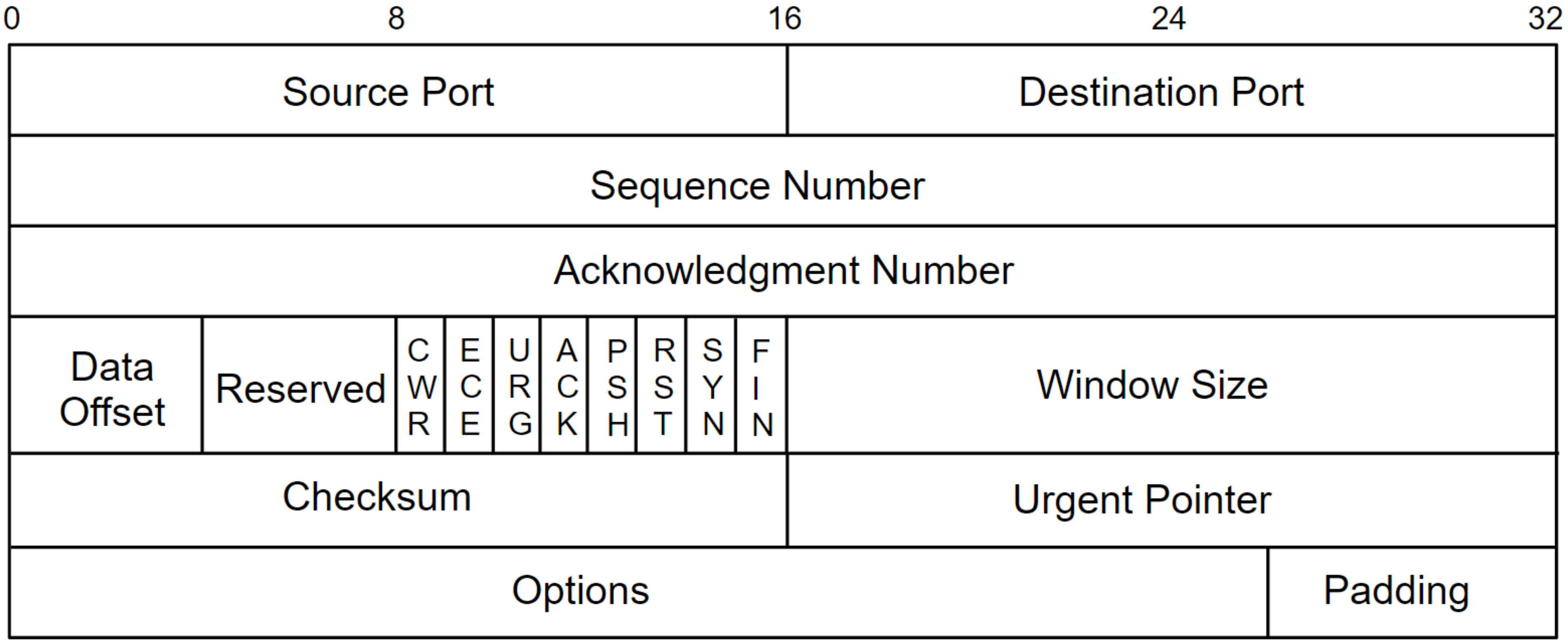
TCP ACKs *(pipelining)*



Ack 200 is sent again, because that is our highest contiguous sequence number we've ack'd

(If a sender gets three duplicate ACKs, it infers that it needs to transmit data)

TCP Header



0 8 16 24 32

Source Port | Destination Port

Sequence Number

Acknowledgment Number

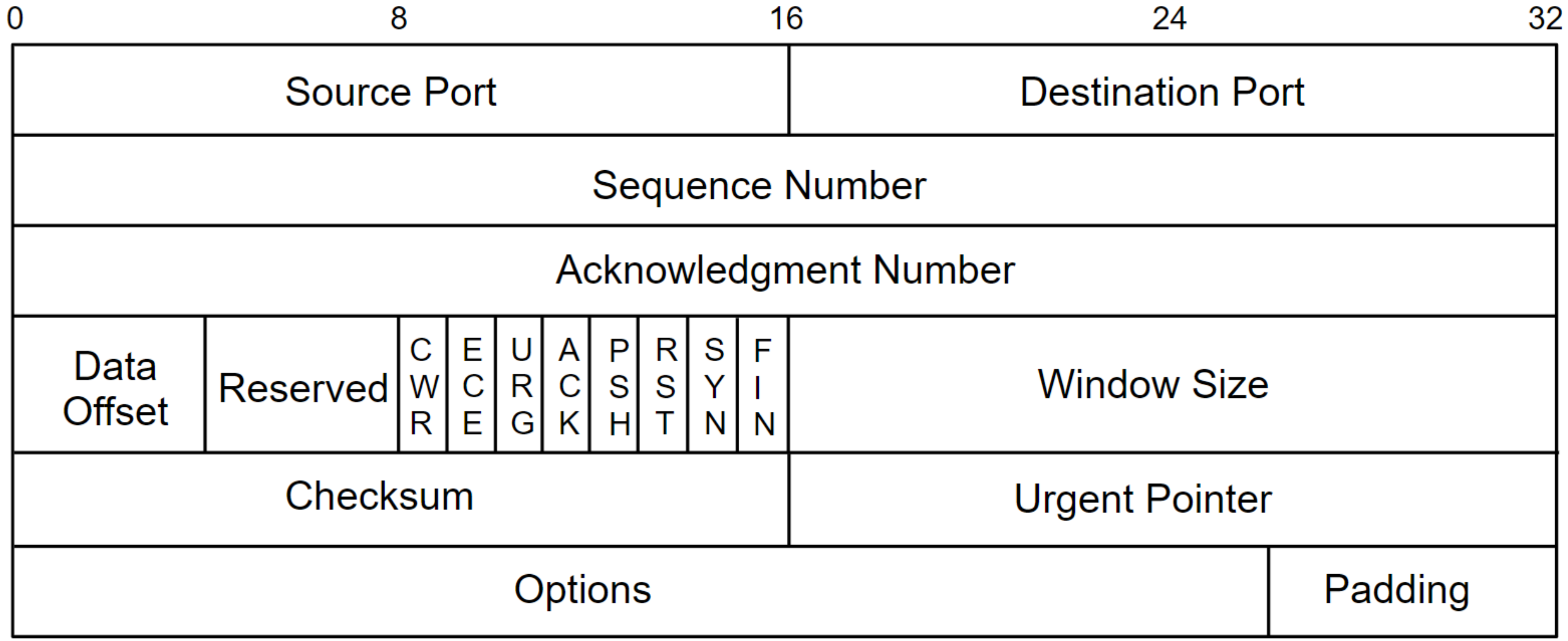
Data Offset | Reserved | CWR | ECE | URG | ACK | PSH | RST | SYN | FIN | Window Size

Checksum | Urgent Pointer

Options | Padding

TCP Header

(20-60 bytes of data)



0 8 16 24 32

Source Port | Destination Port

Sequence Number

Acknowledgment Number

Data Offset | Reserved | CWR | ECE | URG | ACK | PSH | RST | SYN | FIN | Window Size

Checksum | Urgent Pointer

Options | Padding

Transport Layer

TCP Header

(20-60 bytes of data)

0 8 16 24 32

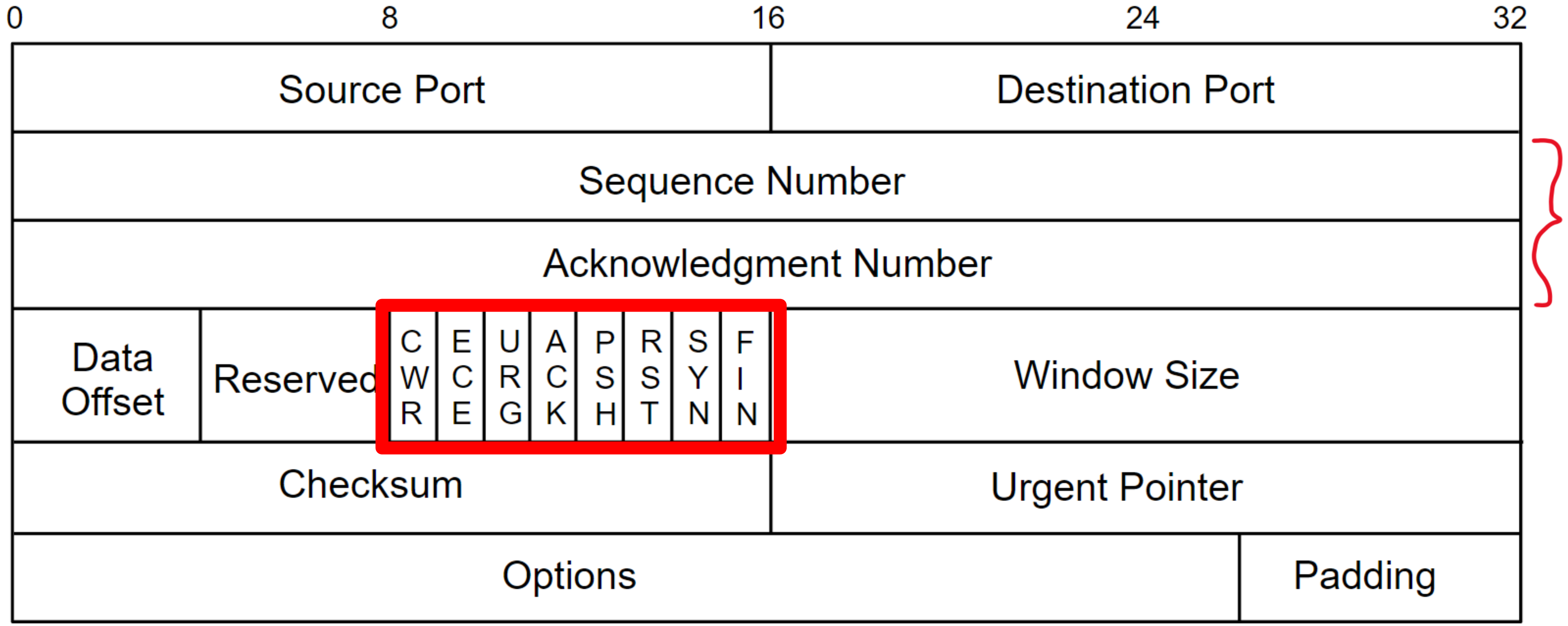
<table>
<tr><td colspan="16">Source Port</td><td colspan="16">Destination Port</td></tr>
<tr><td colspan="32">Sequence Number</td></tr>
<tr><td colspan="32">Acknowledgment Number</td></tr>
<tr><td colspan="4">Data Offset</td><td colspan="4">Reserved</td><td>CWR</td><td>ECE</td><td>URG</td><td>ACK</td><td>PSH</td><td>RST</td><td>SYN</td><td>FIN</td><td colspan="16">Window Size</td></tr>
<tr><td colspan="16">Checksum</td><td colspan="16">Urgent Pointer</td></tr>
<tr><td colspan="26">Options</td><td colspan="6">Padding</td></tr>
</table>

Count by bytes, not segment

Transport Layer

TCP Header

(20-60 bytes of data)



0 8 16 24 32

Source Port | Destination Port

Sequence Number

Acknowledgment Number

Data Offset | Reserved | CWR | ECE | URG | ACK | PSH | RST | SYN | FIN | Window Size

Checksum | Urgent Pointer

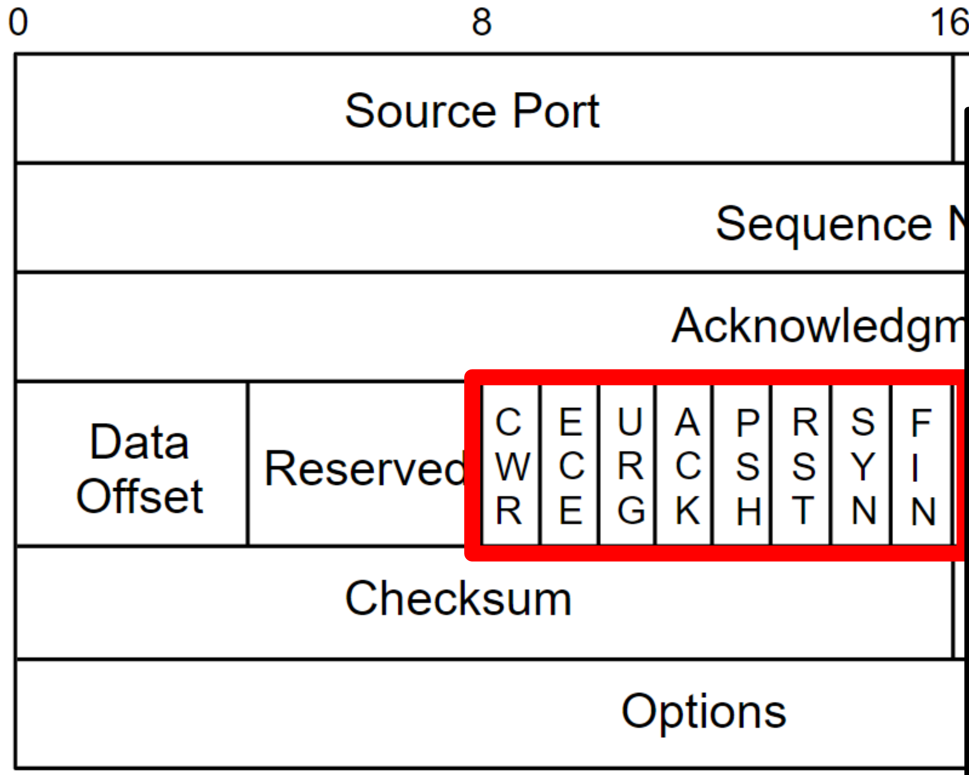
Options | Padding

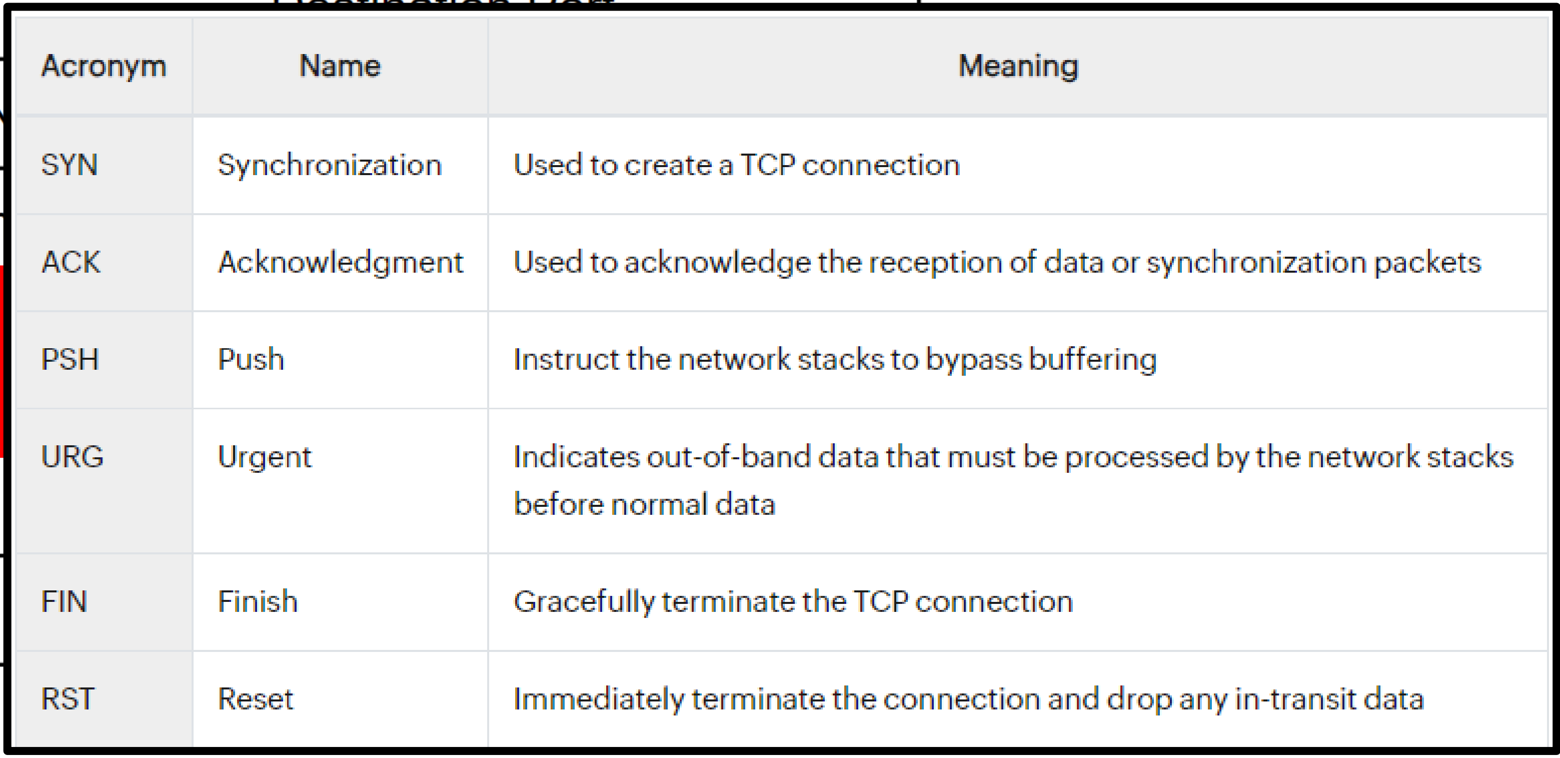
Count by bytes, not segment

Transport Layer

TCP Header

(20-60 bytes of data)





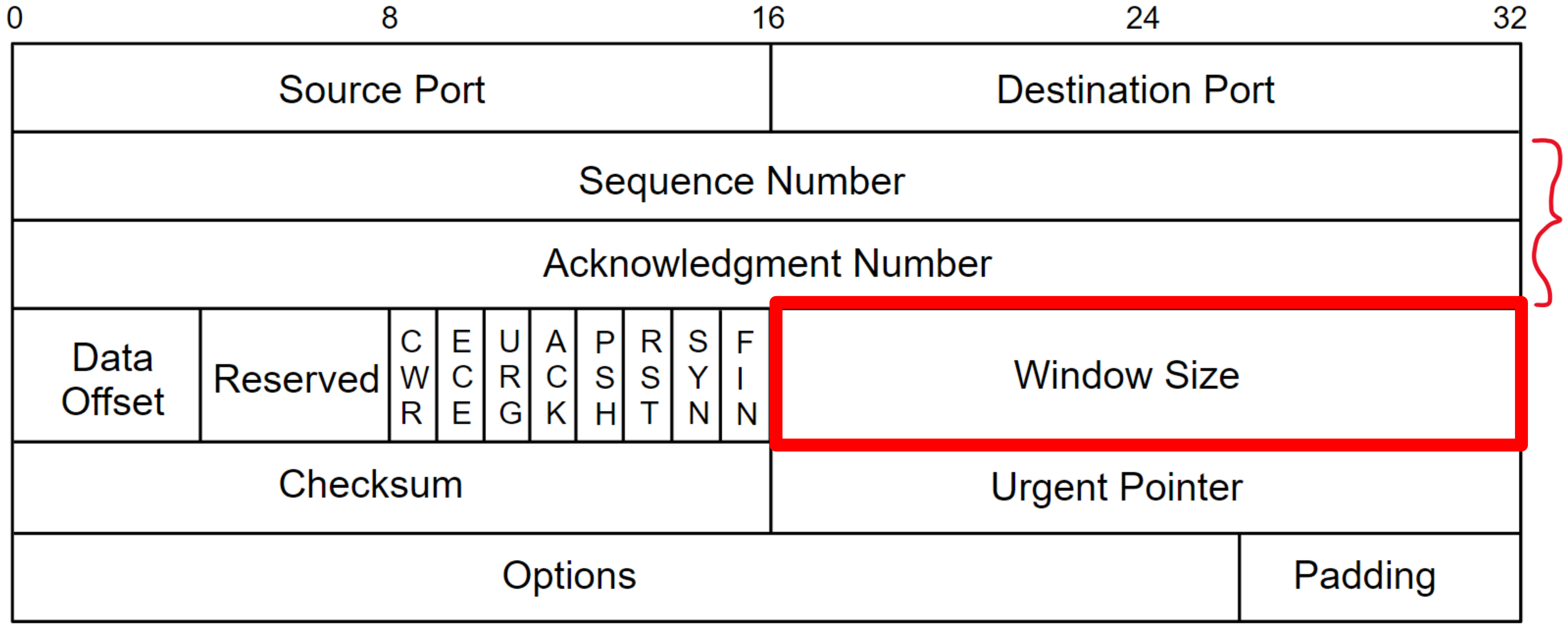
Acronym	Name	Meaning
SYN	Synchronization	Used to create a TCP connection
ACK	Acknowledgment	Used to acknowledge the reception of data or synchronization packets
PSH	Push	Instruct the network stacks to bypass buffering
URG	Urgent	Indicates out-of-band data that must be processed by the network stacks before normal data
FIN	Finish	Gracefully terminate the TCP connection
RST	Reset	Immediately terminate the connection and drop any in-transit data

CWR, ECE – Used for congestion control

Transport Layer

TCP Header

(20-60 bytes of data)

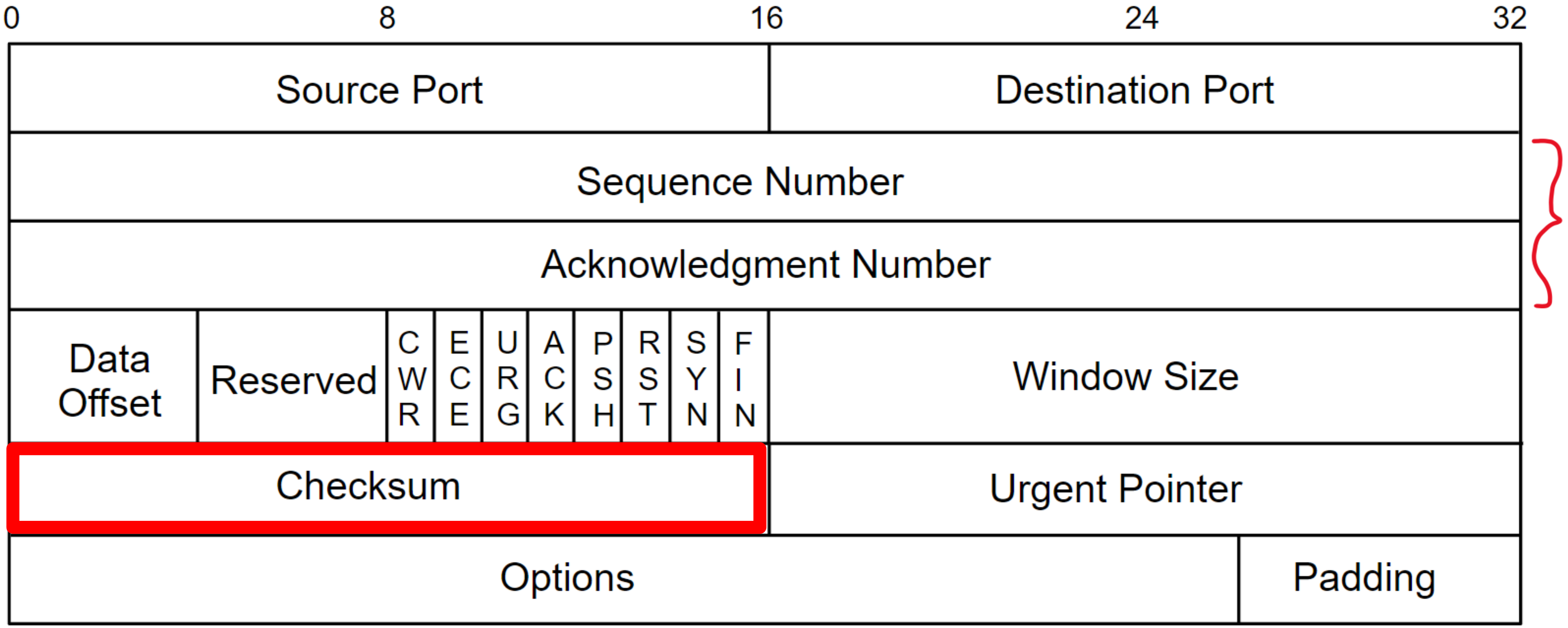


Count by bytes, not segment

How many bytes the receiver is willing to accept

TCP Header

(20-60 bytes of data)



0 8 16 24 32

Source Port

Destination Port

Sequence Number

Acknowledgment Number

Data Offset

Reserved

CWR ECE URG ACK PSH RST SYN FIN

Window Size

Checksum

Urgent Pointer

Options

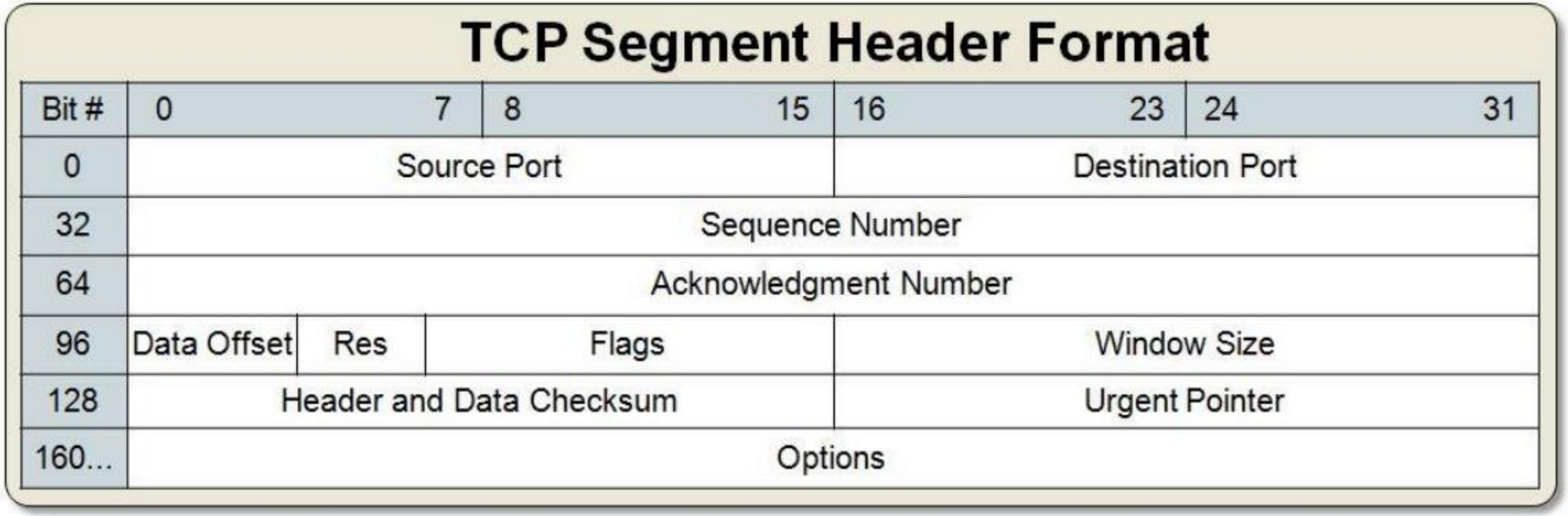
Padding

Count by bytes, not segment

Used to detect bit errors

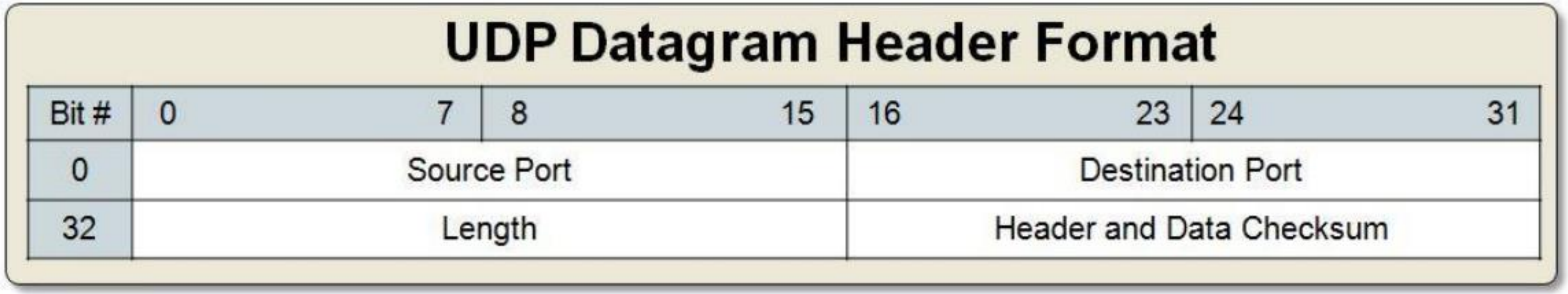
Transport Layer

TCP Header



TCP Segment Header Format

Bit #	0 7	8 15	16 23	24 31
0	Source Port		Destination Port	
32	Sequence Number			
64	Acknowledgment Number			
96	Data Offset Res	Flags	Window Size	
128	Header and Data Checksum		Urgent Pointer	
160...	Options			



UDP Datagram Header Format

Bit #	0 7	8 15	16 23	24 31
0	Source Port		Destination Port	
32	Length		Header and Data Checksum	

TCP Handshake

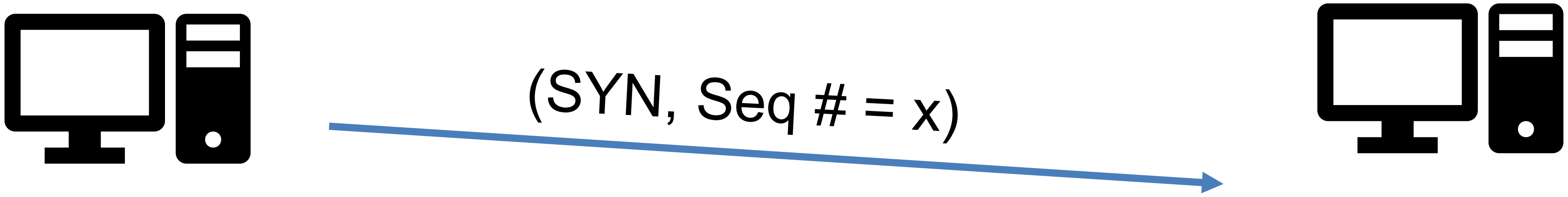
When a process wants to establish a TCP connection with another host, a **TCP handshake** must occur

TCP Handshake

When a process wants to establish a TCP connection with another host, a **TCP handshake** must occur

TCP Handshake

When a process wants to establish a TCP connection with another host, a **TCP handshake** must occur

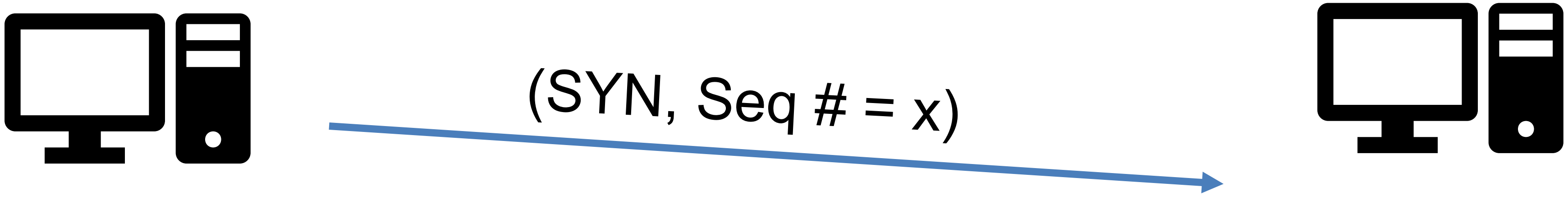


(SYN, Seq # = x)

Transport Layer

TCP Handshake

When a process wants to establish a TCP connection with another host, a **TCP handshake** must occur



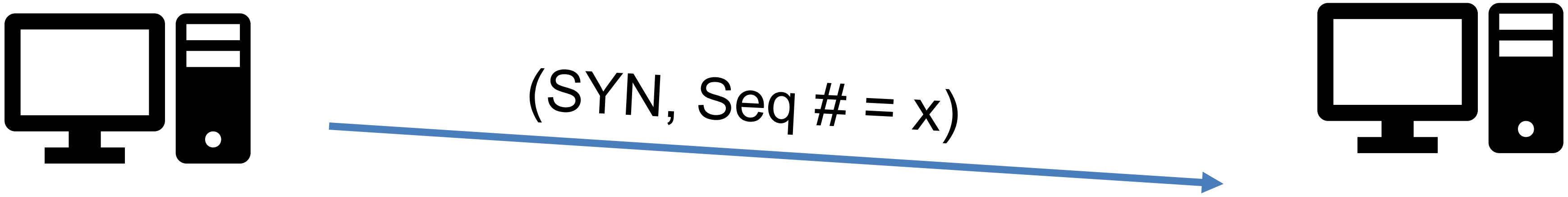
0 8 16 24 32

<table>
<tr><td colspan="10">Source Port</td><td colspan="2">Destination Port</td></tr>
<tr><td colspan="12">Sequence Number</td></tr>
<tr><td colspan="12">Acknowledgment Number</td></tr>
<tr><td>Data Offset</td><td>Reserved</td><td>CWR</td><td>ECE</td><td>URG</td><td>ACK</td><td>PSH</td><td>RST</td><td>SYN
1</td><td>FIN</td><td colspan="2">Window Size</td></tr>
<tr><td colspan="10">Checksum</td><td colspan="2">Urgent Pointer</td></tr>
<tr><td colspan="11">Options</td><td>Padding</td></tr>
</table>

Transport Layer

TCP Handshake

When a process wants to establish a TCP connection with another host, a **TCP handshake** must occur



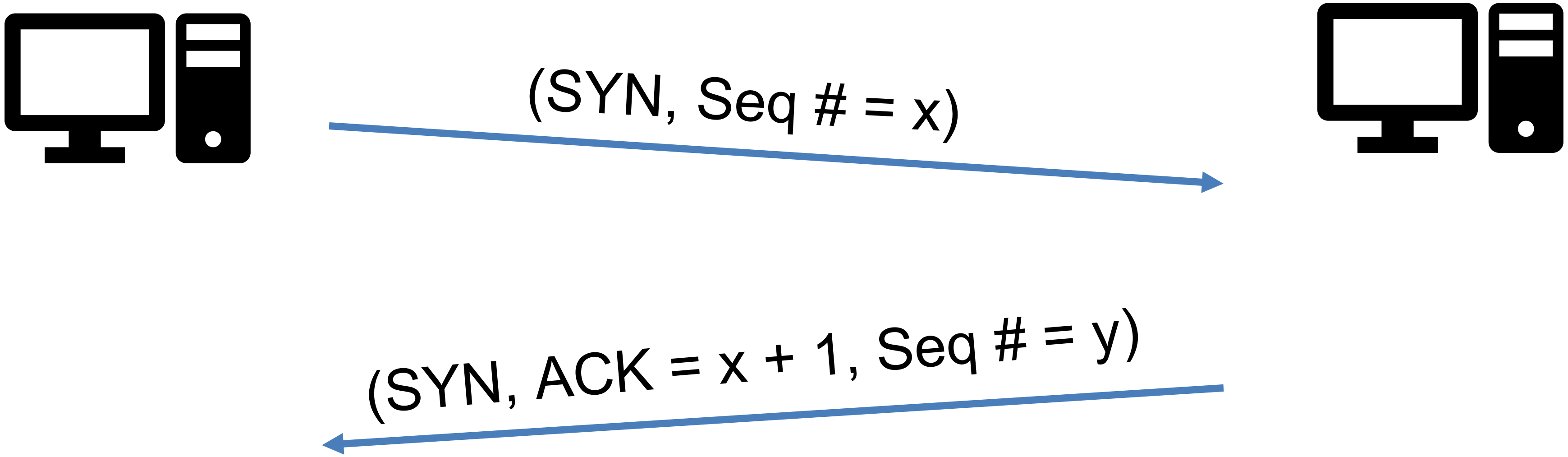
(SYN, Seq # = x)

When establishing the connection, enable the **SYN** flag (set to 1)

Set an initial sequence number

TCP Handshake

When a process wants to establish a TCP connection with another host, a **TCP handshake** must occur

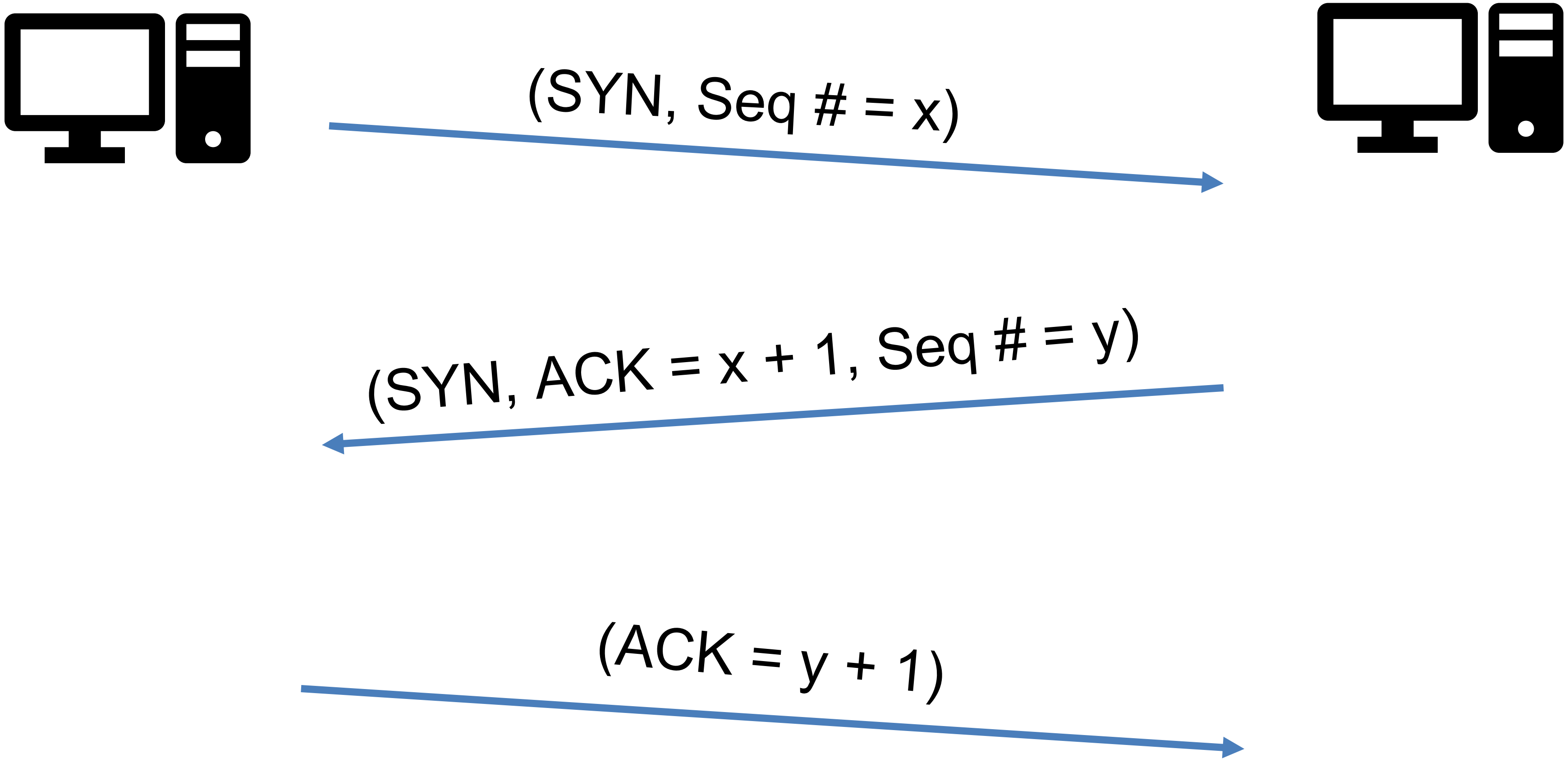


(SYN, Seq # = x)

(SYN, ACK = x + 1, Seq # = y)

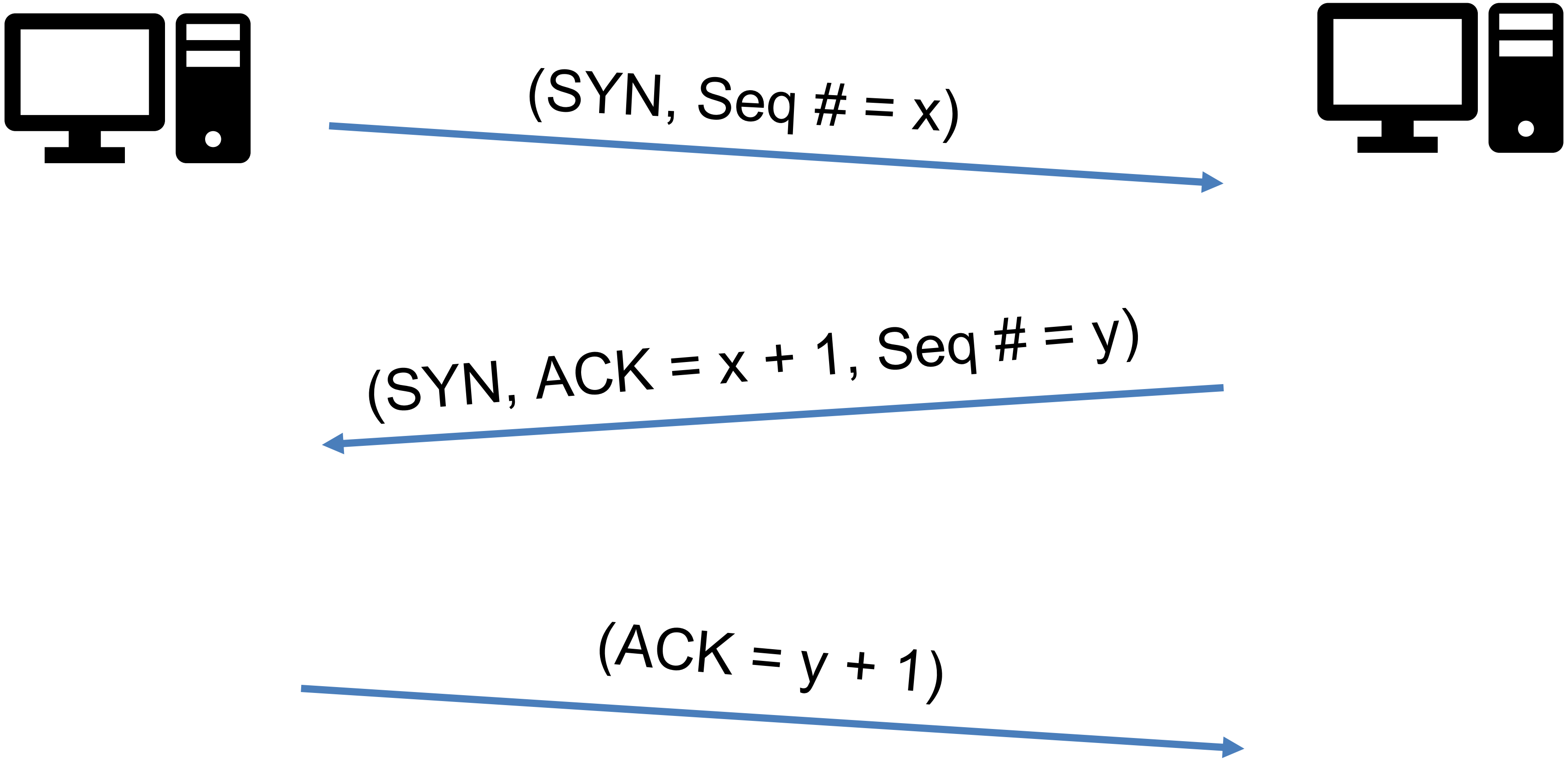
TCP Handshake

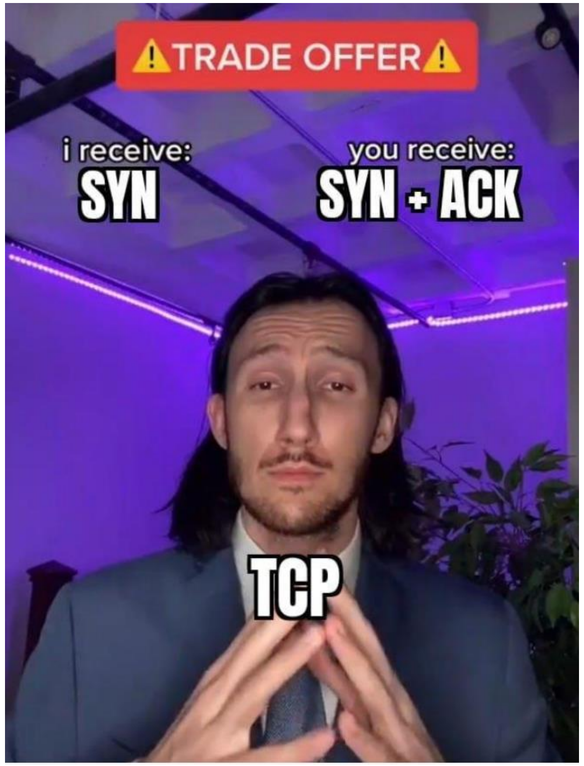
When a process wants to establish a TCP connection with another host, a **TCP handshake** must occur



TCP Handshake

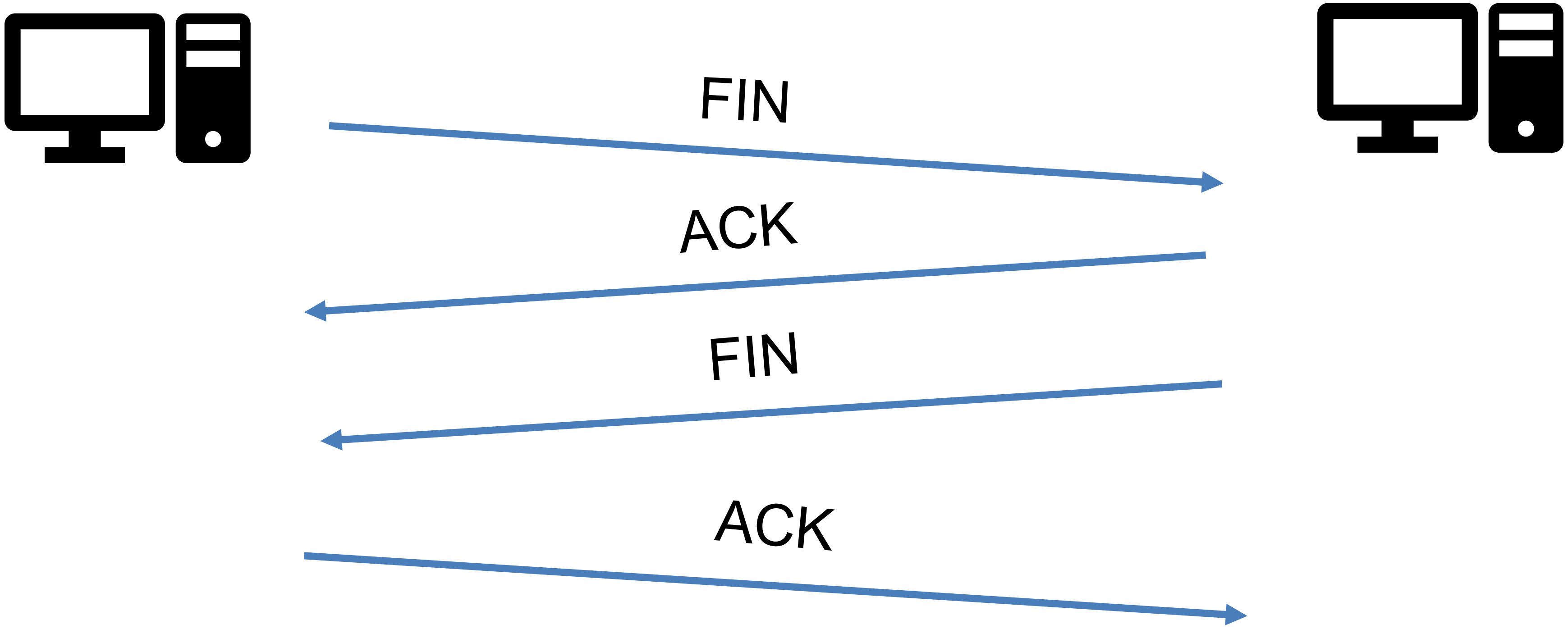
When a process wants to establish a TCP connection with another host, a **TCP handshake** must occur





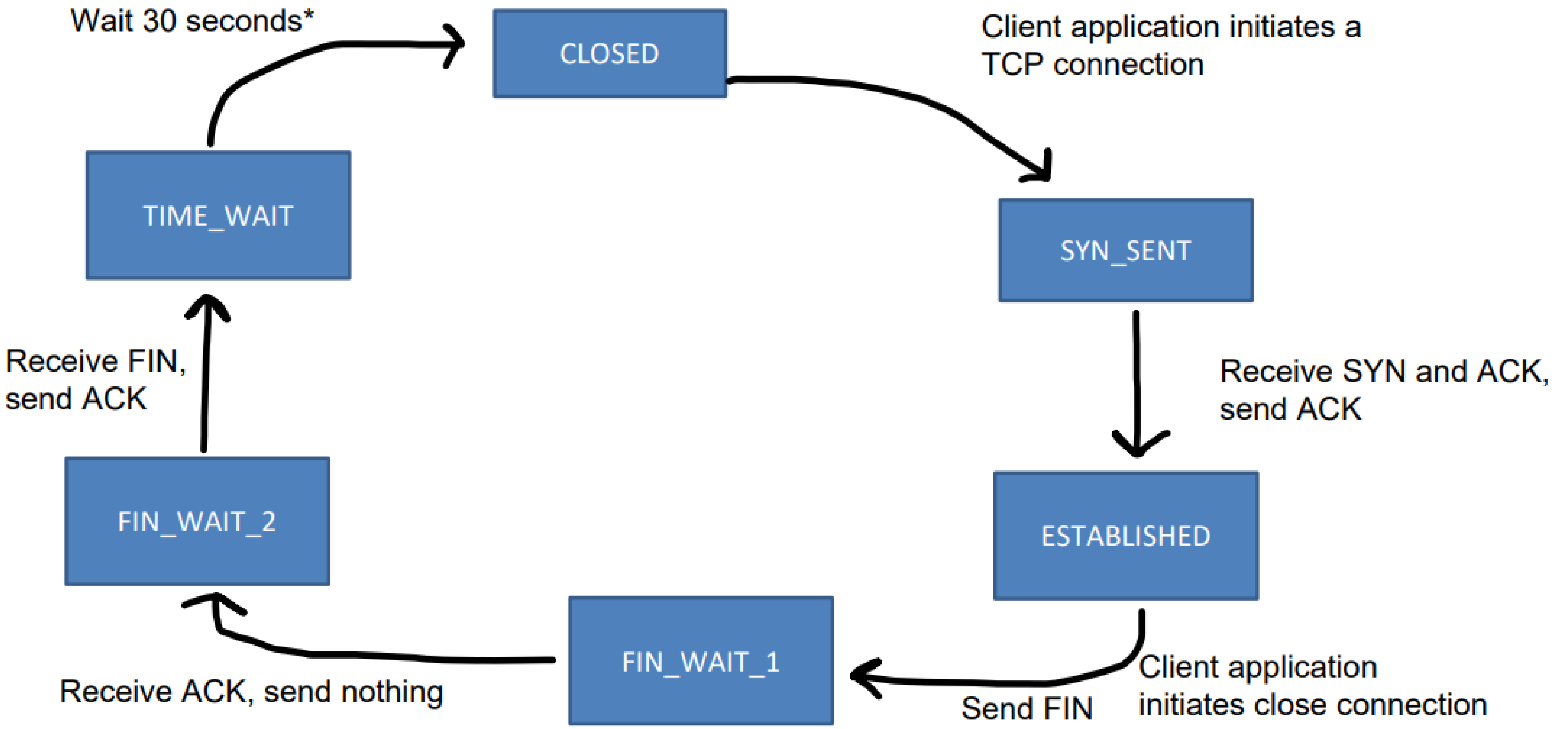
TCP Goodbye

When a process wants to terminate a TCP connection with another host, it sends a **FIN** packet



Transport Layer

TCP States



Wait 30 seconds*
CLOSED
Client application initiates a TCP connection
TIME_WAIT
SYN_SENT
Receive FIN, send ACK
Receive SYN and ACK, send ACK
FIN_WAIT_2
ESTABLISHED
FIN_WAIT_1
Receive ACK, send nothing
Send FIN
Client application initiates close connection

TCP States

What if we receive a packet that has an invalid port number?

TCP Packet → send a TCP segment back with the **RST** flag on

UDP Packet → Send an **ICMP** datagram (network layer thing)

TCP / UDP in Wireshark