CS3.301 Operating Systems and Networks

Transport Layer and how it works!

Karthik Vaidhyanathan

https://karthikvaidhyanathan.com

INTERNATIONAL INSTITUTE OF INFORMATION TECHNOLOGY
HYDERABAD

SERC
Software Engineering Research Centre

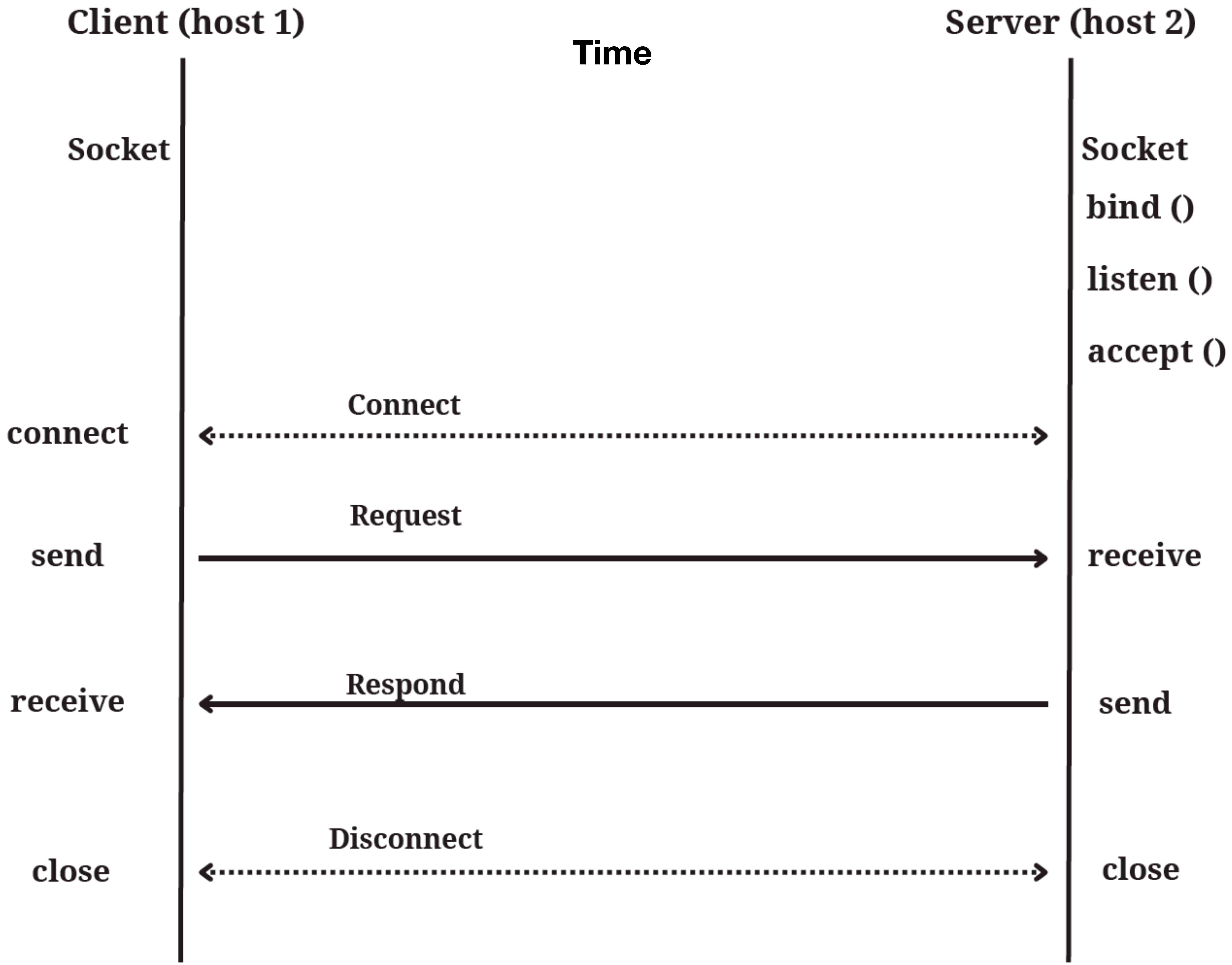
Acknowledgement

The materials used in this presentation have been gathered/adapted/generate from various sources as well as based on my own experiences and knowledge -- Karthik Vaidhyanathan

Sources:

- Computer Networks, 6e by Tanebaum, Teamster and Wetherall
- Computer Networks: A Top Down Approach by Kurose and Ross
- Computer Networking essentials, Youtube Channel
- Other online sources which are duly cited

Using Sockets



Client (host 1)

Time

Server (host 2)

Socket

Socket

bind ()

listen ()

accept ()

Connect

connect

Request

send

receive

Respond

receive

send

Disconnect

close

close

More about Ports

- Application process is identified by tuple (IP address, Protocol, Port)
    - Port are 16-bit integers representing “mailboxes” that process leases
- Servers are often bind to “well-known-ports”
- Clients are assigned ephemeral ports
    - Chosen by the OS temporarily

Some well Known Ports

Port	Protocol	Use
20, 21	FTP	File Transfer
22	SSH	Remote login
25	SMTP	Email
80	HTTP	World wide web
443	HTTPS	Secured web
543	RTSP	Media Player Control

An Opportunity for a Context Switch?

- The calls of establishing socket are blocking calls
  - connect(), accept(), receive()
  - Once the call is made, OS halts the program to wait to receive some response
  - They are essentially **System calls**
  - Trap instruction is called and there is an opportunity for a context switch

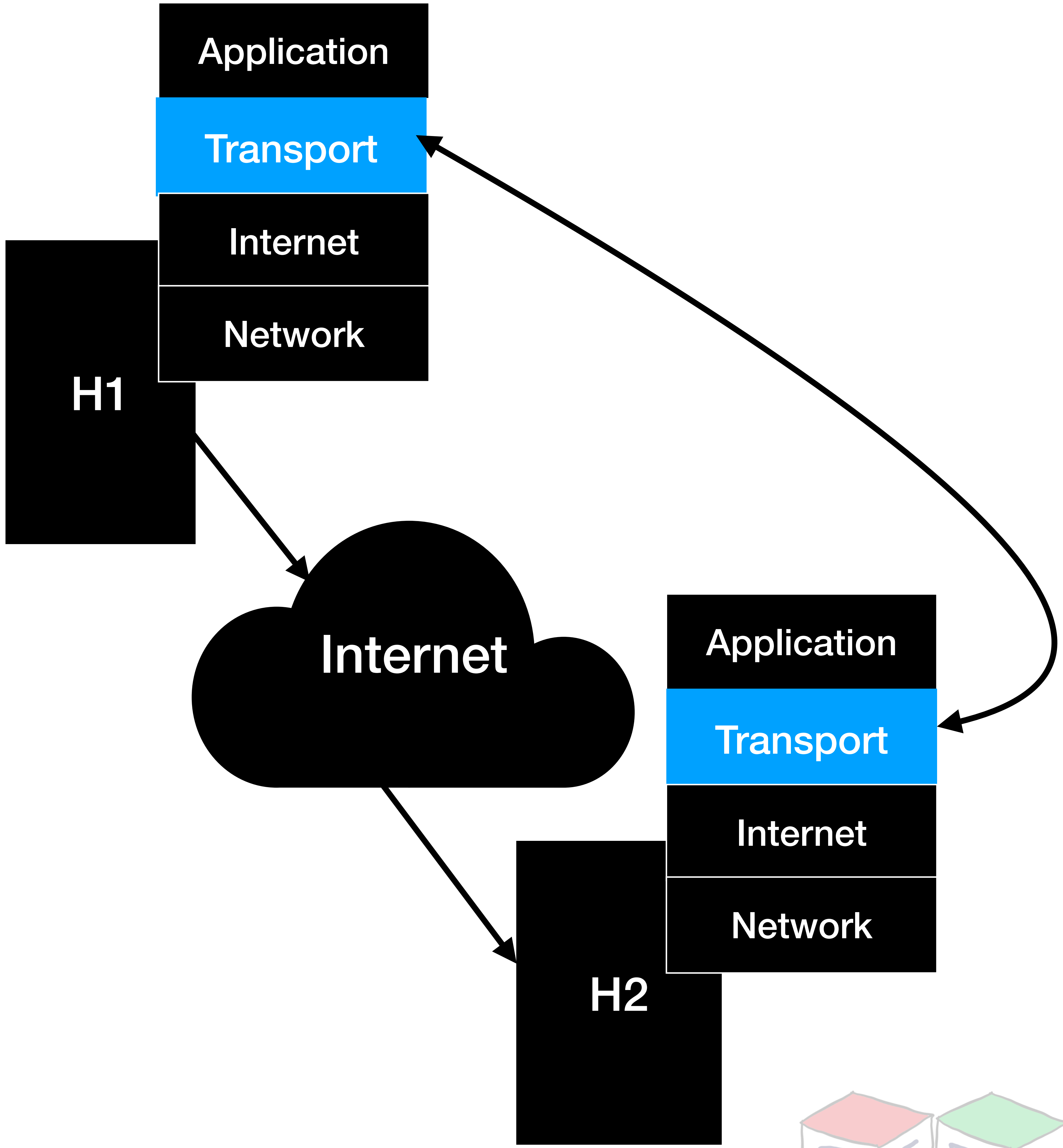
Let us take a step back

Types of Links

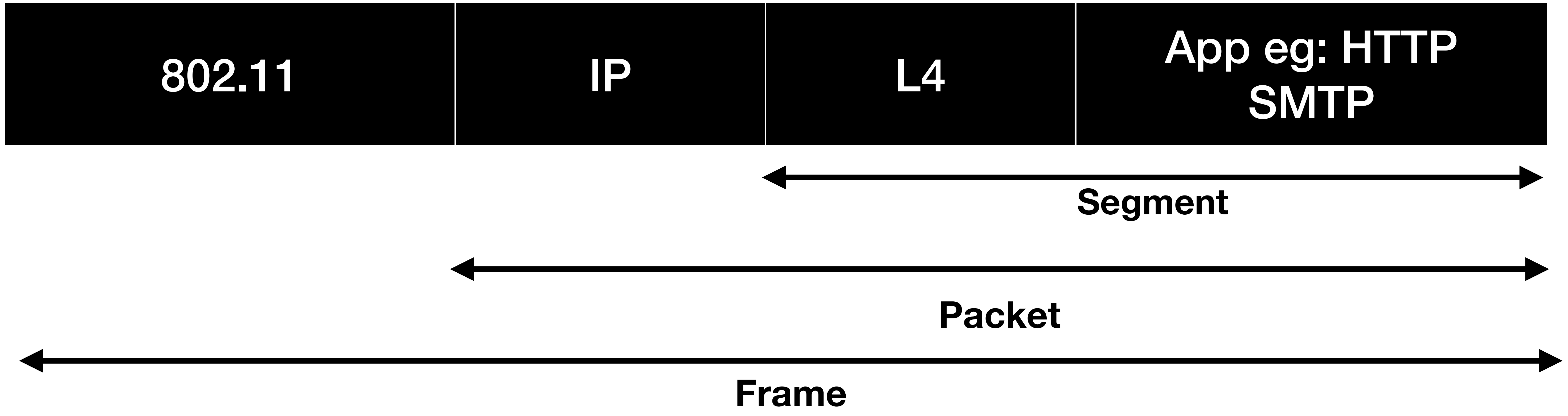
- **Full Duplex**
  - Bidirectional
  - Both sides at the same time
- **Half-duplex**
  - Bidirectional
  - Both the sides but only one direction at a time (eg: walkie talkies)
- **Simplex**
  - Unidirectional

Transport Services and Protocols

- Provides logical communication between application processes running on different hosts
- Transport protocols actions in the end systems:
  - Sender: breaks application messages into segments, passes to network layer
  - Receiver: reassembles messages into messages, passes to application layer
- Protocols: TCP, UDP



Quick Recap

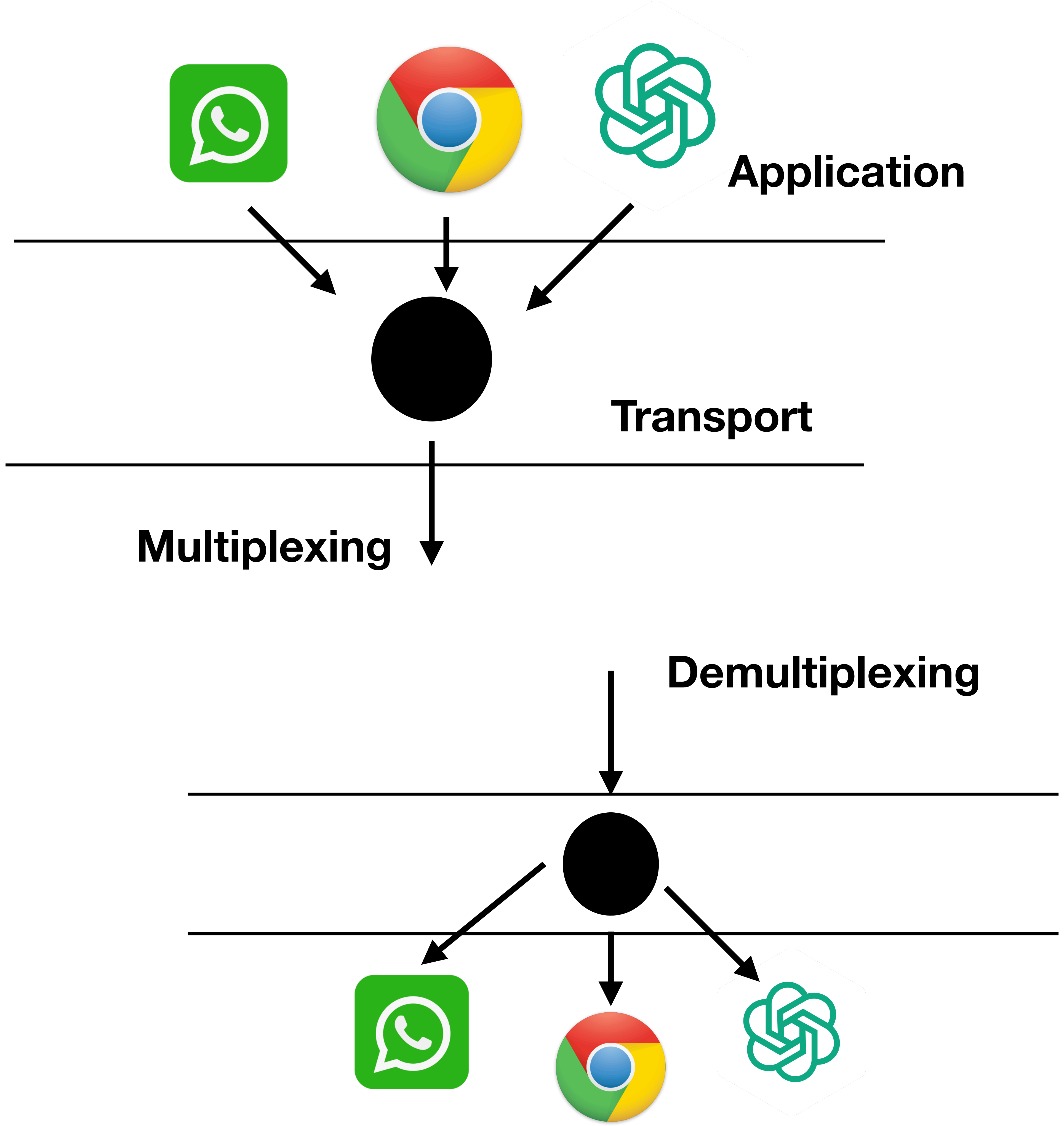


- Segments carry data across the network
- **Segments** are carried within the packets, within frames
- Each layer adds a **header** (Above L4 will be replaced by its header)

Multiplexing and Demultiplexing

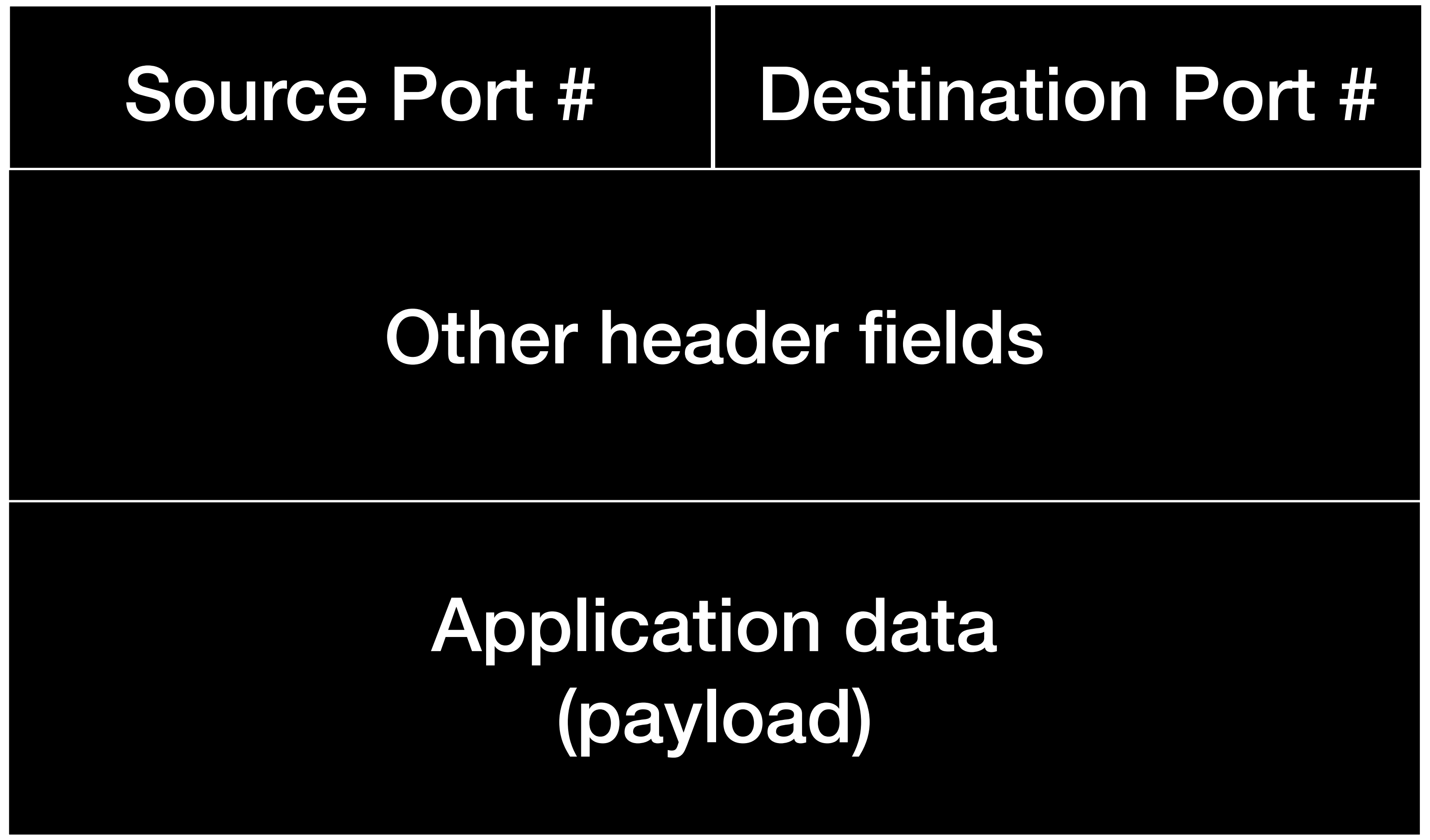
- **Multiplexing as sender**: Handle data from multiple sockets, add transport header

- **Demultiplexing as receiver**: Use header info to deliver received segments to correct socket



Working of Demultiplexing

- Host receives IP datagrams
  - Each datagram has source IP address, destination IP address
  - Each datagram carries one transport layer segment
  - Each segment has source and destination port number
- IP addresses and ports are used to direct segment to appropriate socket



TCP/UDP Segment format

Connection Oriented vs Connectionless

Demultiplexing Scenarios

- **Connection oriented (TCP)**
  - TCP socket identified by 4 tuple
    - Source IP, destination IP, source port and destination port
  - Receiver uses all 4 to direct segment to appropriate socket
  - Server may support many TCP sockets
    - **Each socket has it own client**

- **Connectionless (UDP)**
  - UDP socket identified by 2 tuple
    - Destination IP and port
  - Receiver uses the port to redirect to the corresponding socket
  - UDP segments with same destination port but different IP or source port
    - **Redirected to same socket**

TCP vs UDP

TCP	UDP
Connection Oriented	Not Connection Oriented
Reliability (order is maintained and retransmission)	Unreliable
Higher overhead - reliability, error checking, etc	Low overhead
Flow control (based on network)	No implicit flow control
Error detection - retransmit erroneous packets	Has some error checking - Erroneous packets are discarded without notification
Congestion Control	No Congestion Control
Use cases: HTTP/HTTPS, File transfer, Mail	Use cases: Streaming data, VoIP, DNS queries, ..

Connection Oriented and Reliability

- **Connection Oriented**
    - In TCP, the connection is first established before the data is transmitted
    - In UDP there is no notion of connection starting and ending (use timeout)
- **Reliability**
    - Confirmation of data delivery (Acknowledgement is there) in TCP
        - Order is preserved or maintained
        - Error can be handled (Awareness). TCP can handle it.
    - In UDP there is no confirmation, the client trusts that there is someone to receive the data (Fire and Forget)
        - No error awareness (at L4). Protocol does not handle it

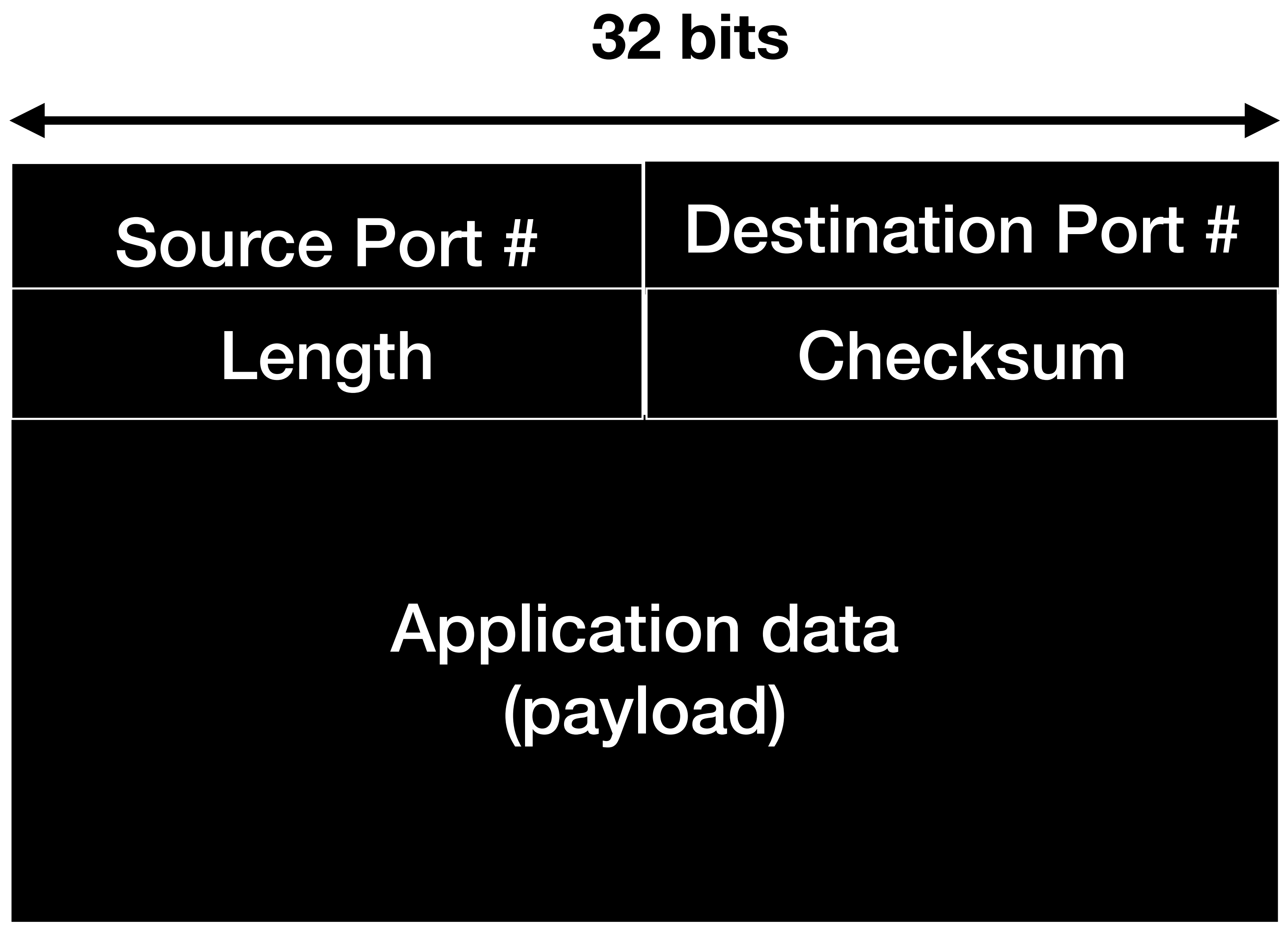
Flow Control and Overhead

- Flow Control
  - TCP can adjust the transmission rate to use maximum available bandwidth
  - Check how much the receiver can receive and adjust accordingly

- Overhead
  - TCP Adds a larger header to the data ~ 20 bytes or even more
  - TCP has more features that does not exist in UDP
  - In UDP the header length is ~ 8 bytes

UDP Segment Header

- Length: In bytes of the UDP segment including the header
- Checksum: For error detection (16 bit value which represents the sum of UDP header, payload and Pseudo header from IP layer)
    - Supports Error detection
    - Makes use of 1’s compliment arithmetic to find the sum



UDP Segment Format

Checksum Process

- **Sender**
  - All contents of the header including IP addresses are treated as sequence of 16 bit integers
  - Checksum: addition (one’s complement) of segment content
- **Receiver**
  - Compute checksum of received content
  - Check if received and header checksum are equal - No error
  - Else, Error detected

Example

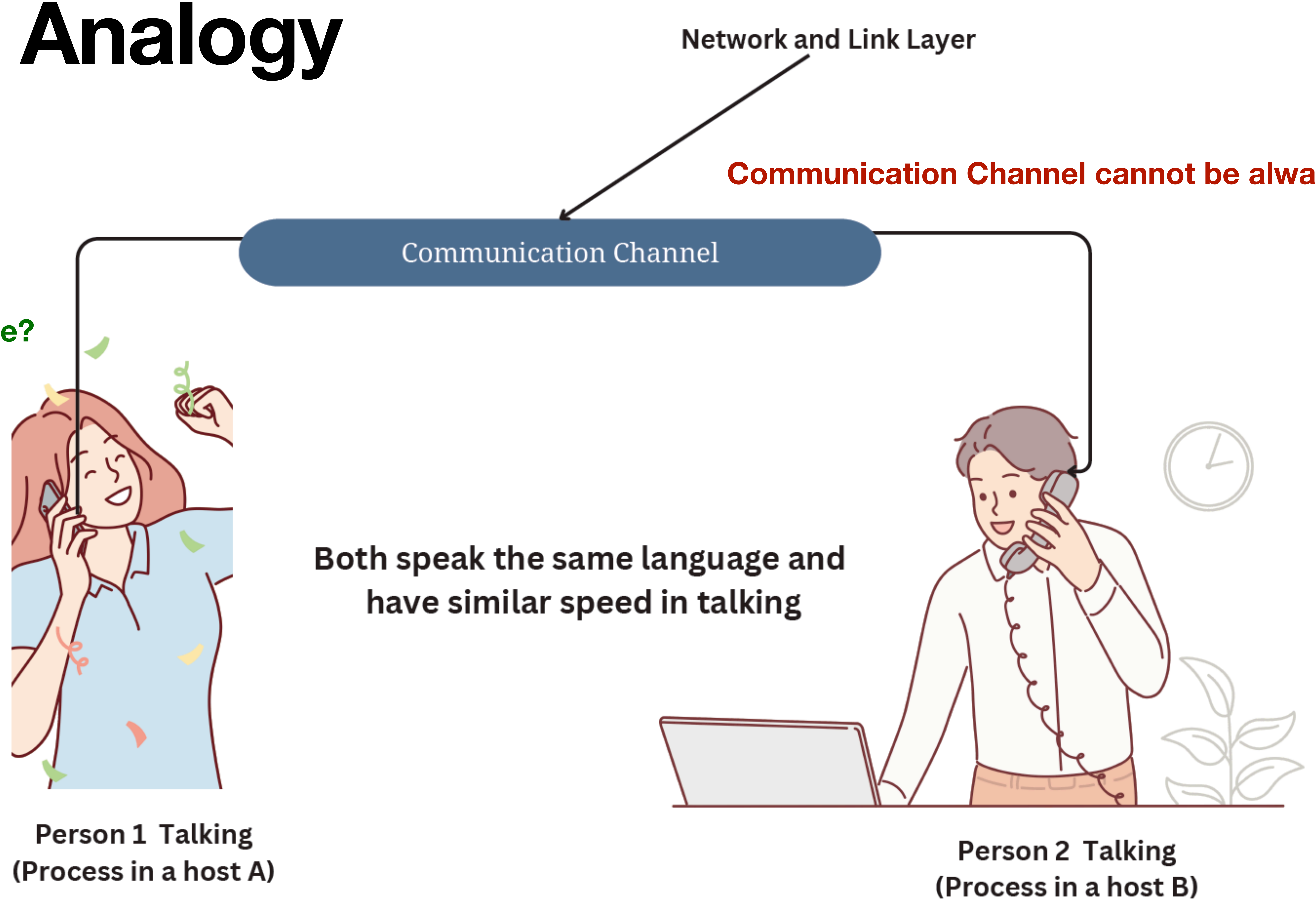
	1 1 1 0 0 1 1 0 0 1 1 0 0 1 1 0
	1 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1
Add it back	1 1 0 1 1 1 0 1 1 1 0 1 1 1 0 1 1
Sum	1 0 1 1 1 0 1 1 1 0 1 1 1 1 0 0
Checksum	0 1 0 0 0 1 0 0 0 1 0 0 0 0 1 1

TCP is the most used protocol on the internet. How does TCP work?

What all you need to provide some features that TCP provides?

A Small Analogy

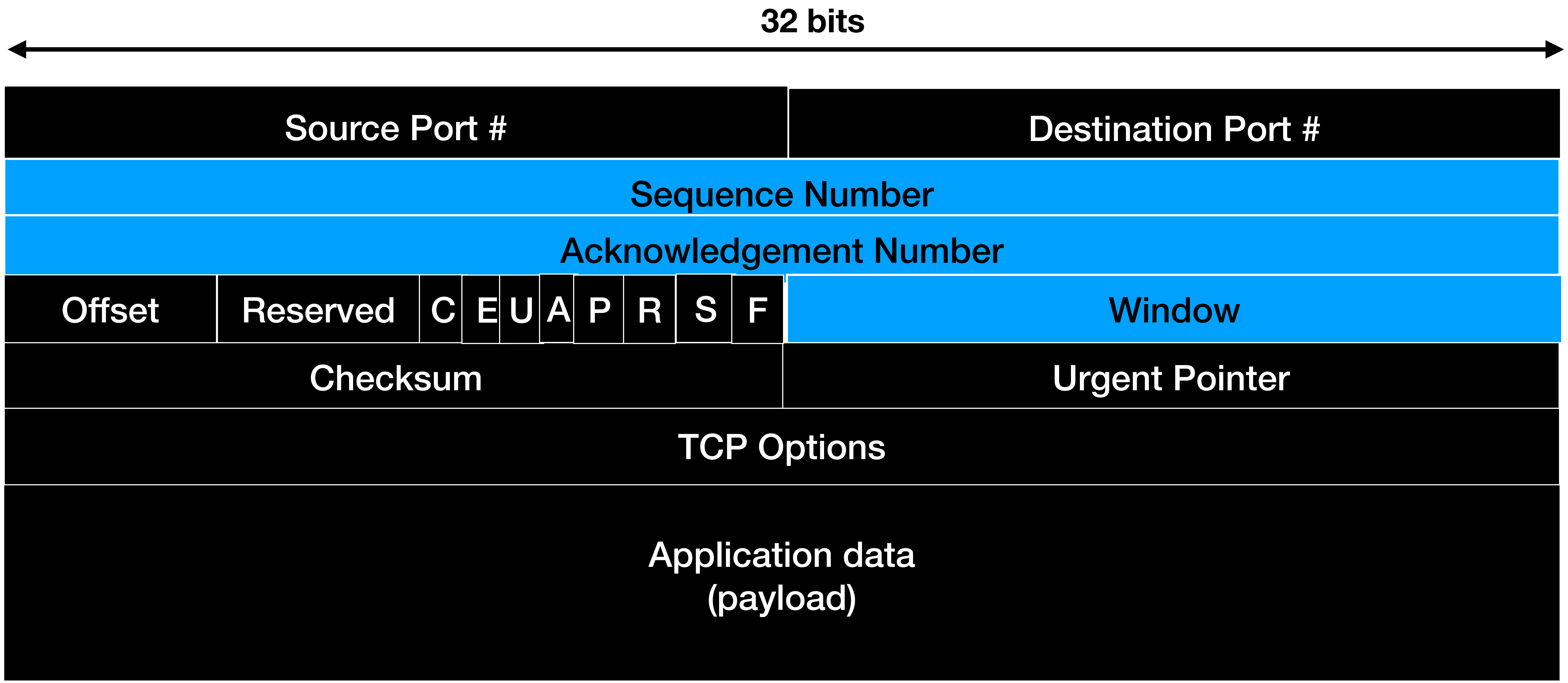
What can we do from the protocol perspective?


Network and Link Layer
Communication Channel
Both speak the same language and have similar speed in talking
Person 1 Talking (Process in a host A)
Person 2 Talking (Process in a host B)

Communication Channel cannot be always reliable!!

Do we foresee some challenges?

Lets go into TCP - Header



32 bits

Source Port # | Destination Port #

Sequence Number

Acknowledgement Number

Offset | Reserved | C | E | U | A | P | R | S | F | Window

Checksum | Urgent Pointer

TCP Options

Application data
(payload)

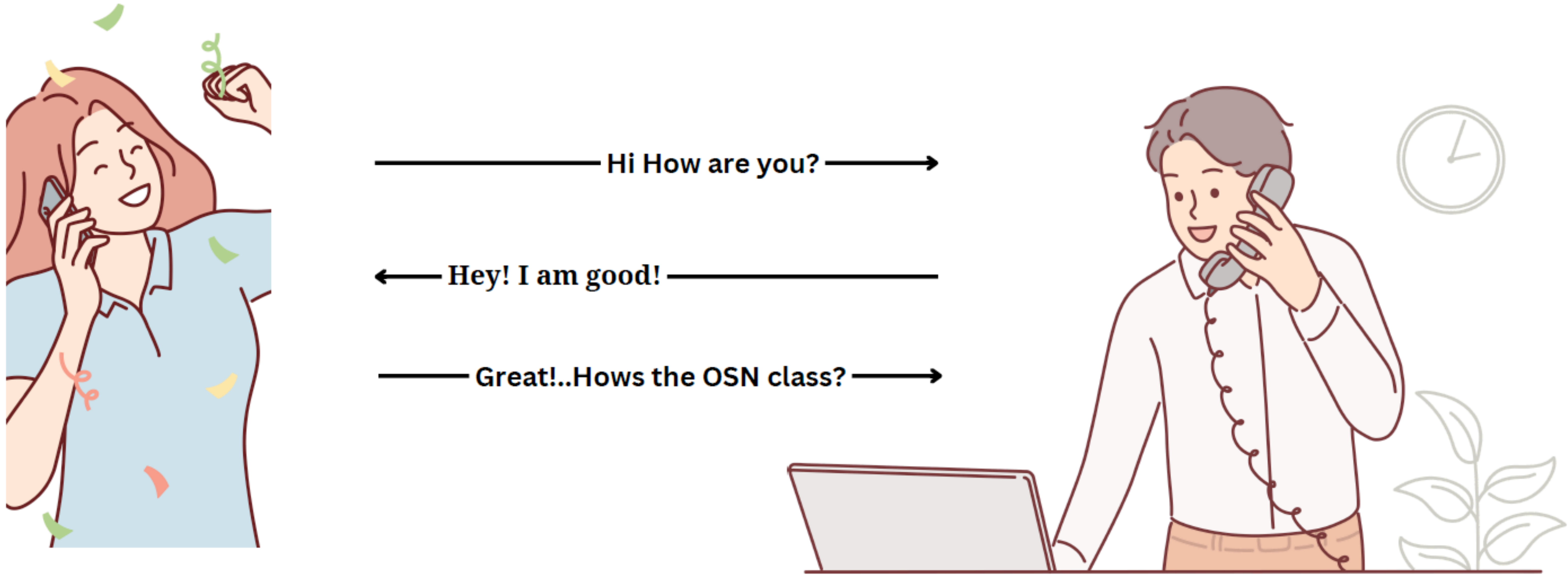
TCP Segment Header

Header Elements

- **Sequence number:** Tracks bytes that are sent (# of bytes that are sent)
- **Acknowledgement number:** Tracks bytes that are received (Sequence number of the next expected byte)
- **Window/Receive Window:** Number of bytes the receiver can accept (Flow control)
- **A:** Acknowledgement bit
- **R, S, F:** Connection management
- **C, E:** Congestion notification
- **Offset:** Length of the TCP header

What do ACK and Sequence Number do?

Reliability!!



Hi How are you? →

← Hey! I am good!

Great!..Hows the OSN class? →

Person 1 Talking
(Process in a host A)

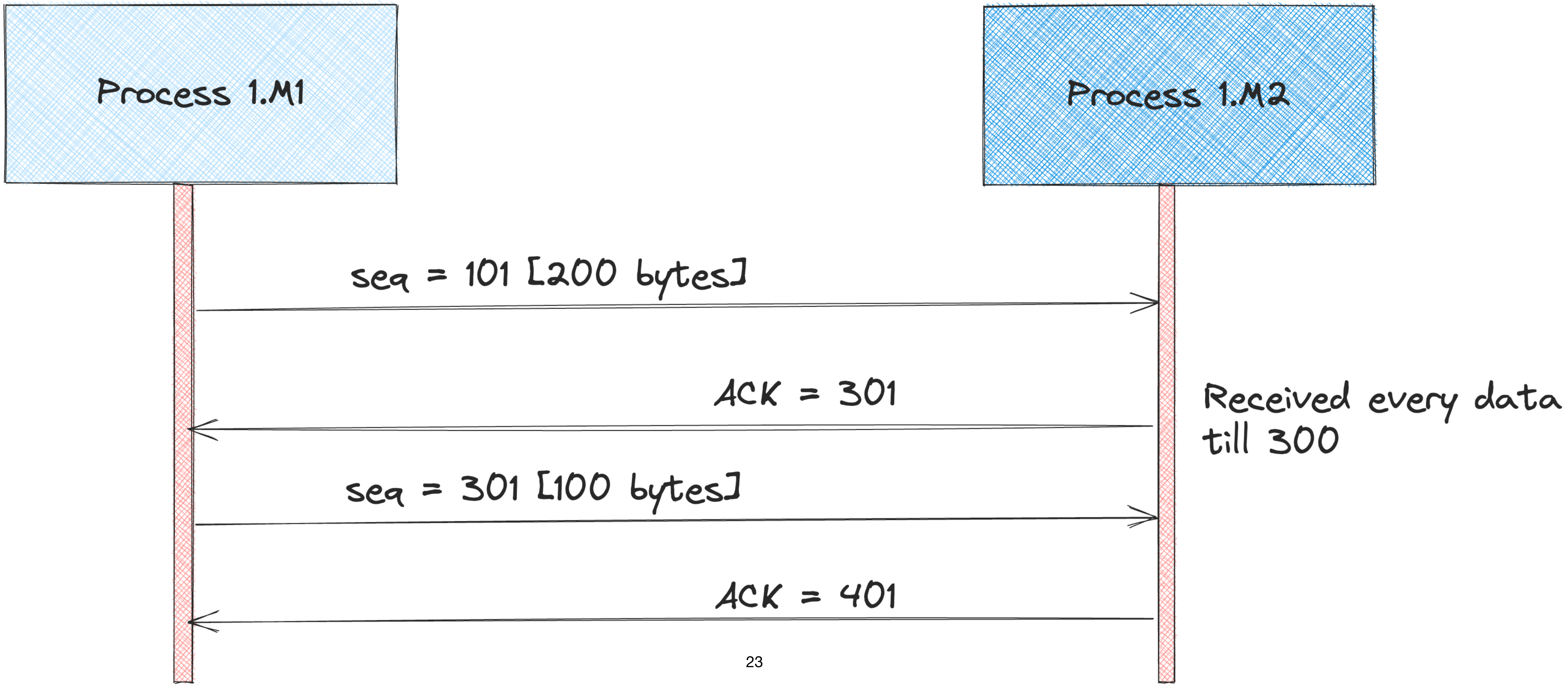
Person 2 Talking
(Process in a host B)

Each word the Person 1 says
reaches person 2 in the same order

Whatever Person 1 Says, Person 2
acknowledges before adding new
points to the conversation

What do ACK and Sequence Number do?

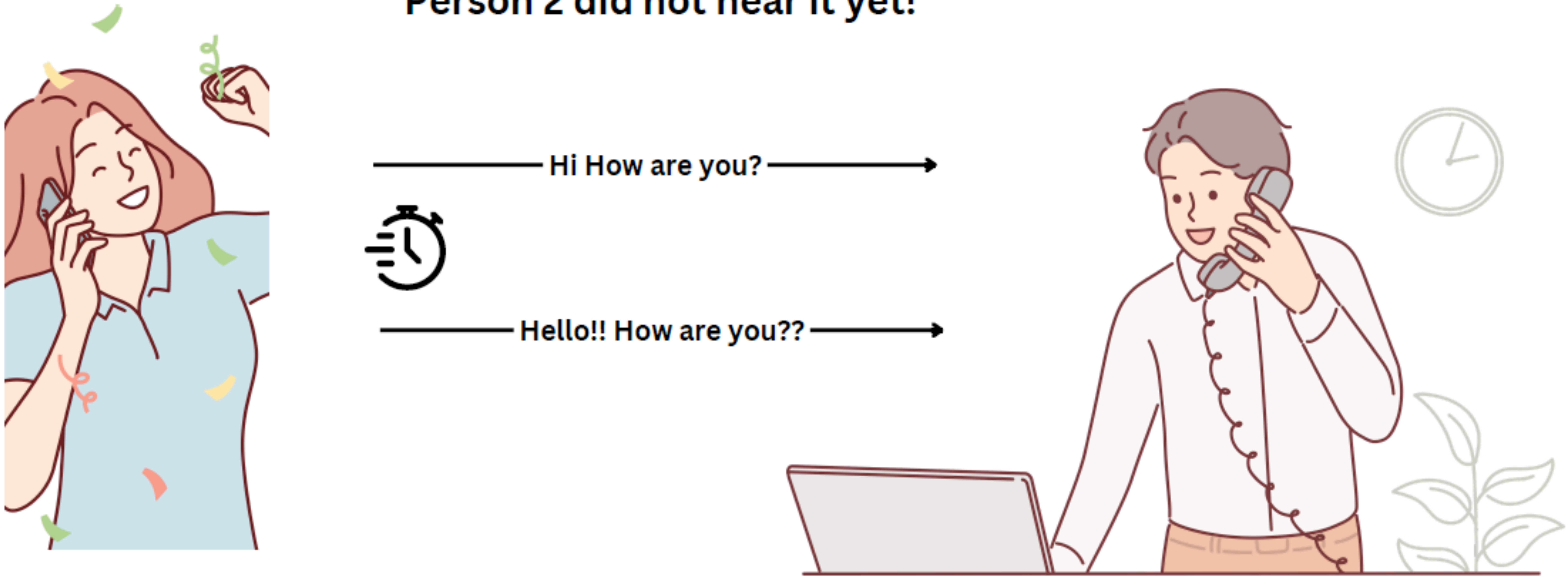
Reliability!!



How to handle if data is lost?

Can we retransmit?

Person 1 is trying to Speak
Person 2 did not hear it yet!



Hi How are you?

Hello!! How are you??

Person 1 Talking
(Process in a host A)

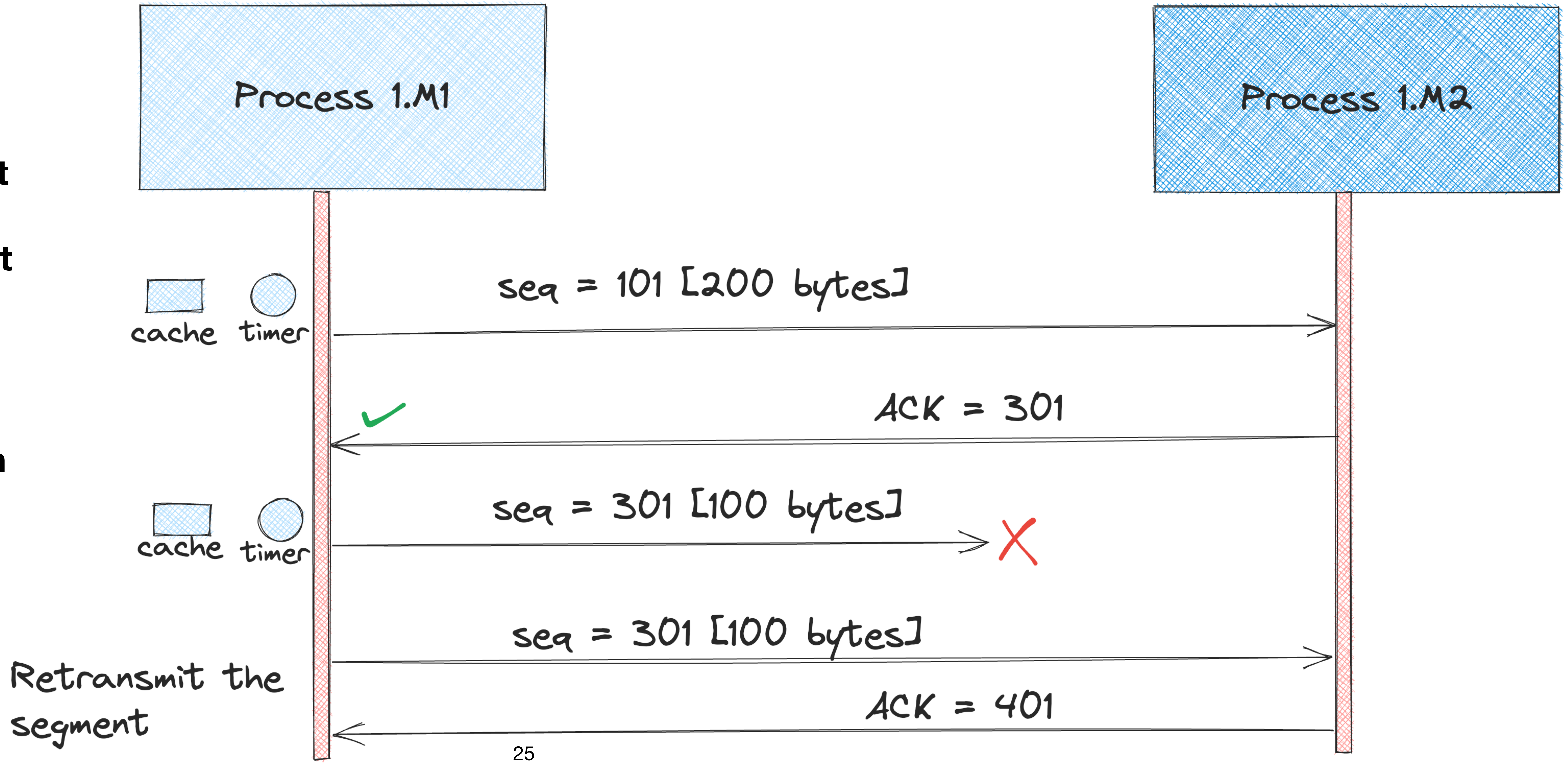
Person 2 Talking
(Process in a host B)

How to handle if data is lost?

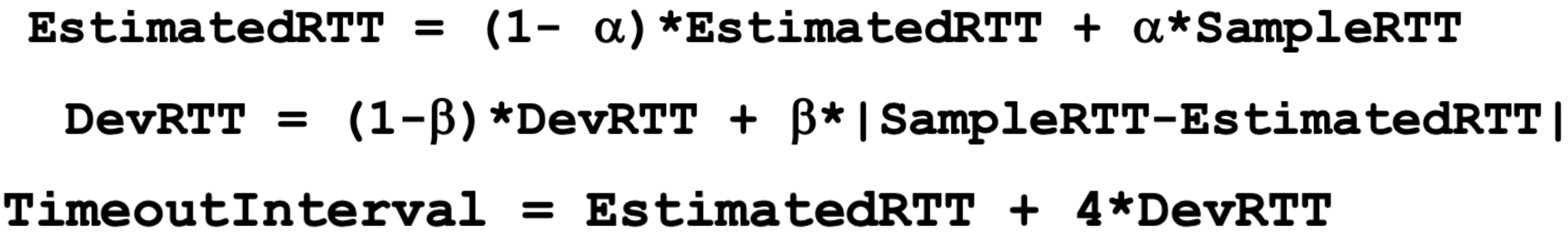
Retransmission timeout also known as Round Trip Timeout (RTT)

TCP caches every data sent
in a buffer (OS supports)
Until retransmission timeout

What if ACK does not reach
Back Process 1.M1?



How to calculate RTT?



$$\text{EstimatedRTT} = (1-\ \alpha)*\text{EstimatedRTT} + \alpha*\text{SampleRTT}$$

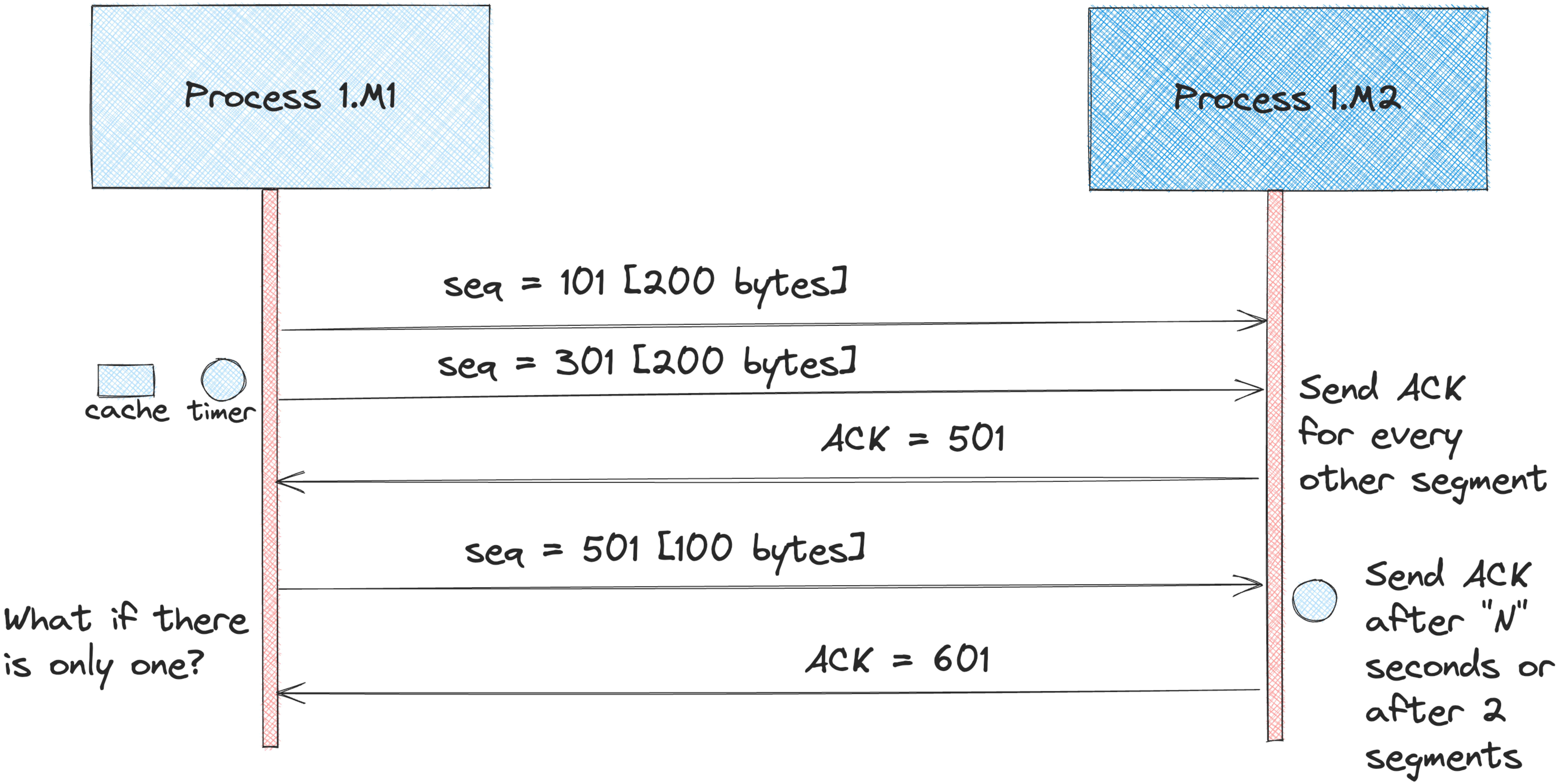
$$\text{DevRTT} = (1-\beta)*\text{DevRTT} + \beta*|\text{SampleRTT}-\text{EstimatedRTT}|$$

$$\text{TimeoutInterval} = \text{EstimatedRTT} + 4*\text{DevRTT}$$

- **SampleRTT:** Time measured from segment transmission until ACK receipt
- **EstimatedRTT:** Estimated weighted moving average (EWMA) $\alpha = 0.25$
- **DevRTT:** EWMA of sampleRTT deviation from EstimatedRTT $\beta = 0.75$
- **TimeoutInterval:** Estimated Time plus some kind of safety margin

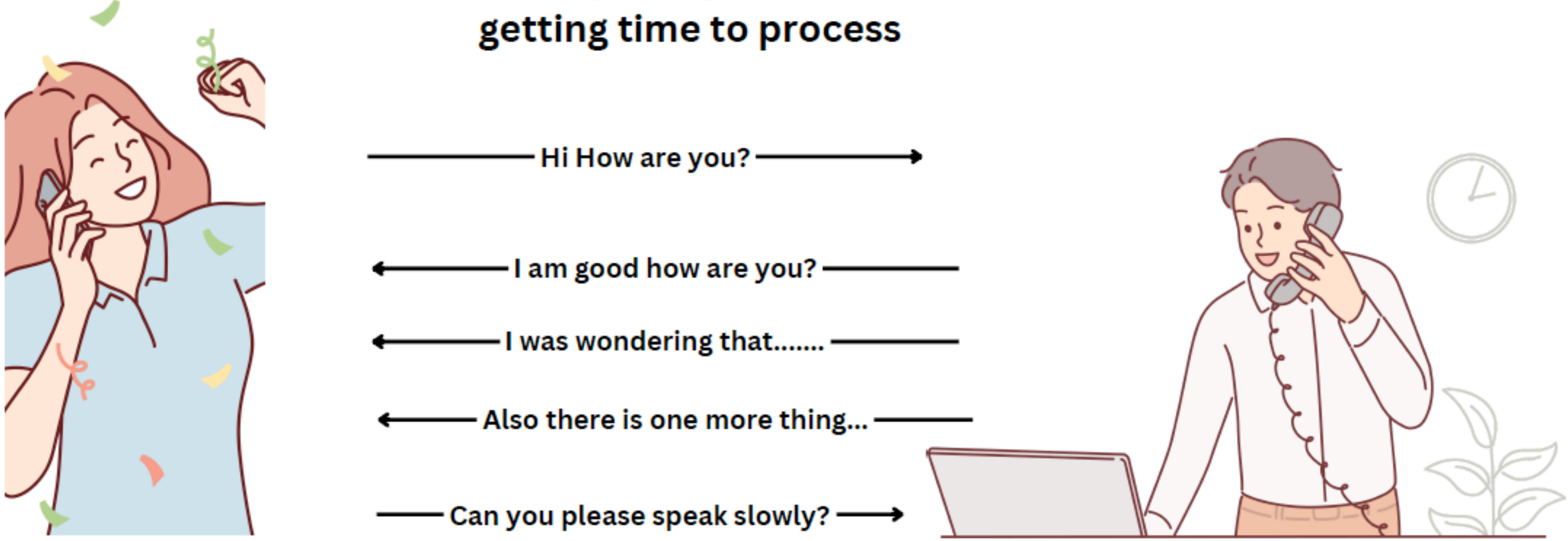
Do We need to Send ACK for each segment?

Use delayed acknowledgements



What if the speed is high?

Person 2 is speaking fast and many things..Person 1 is not getting time to process



Hi How are you? →

← I am good how are you?

← I was wondering that.......

← Also there is one more thing...

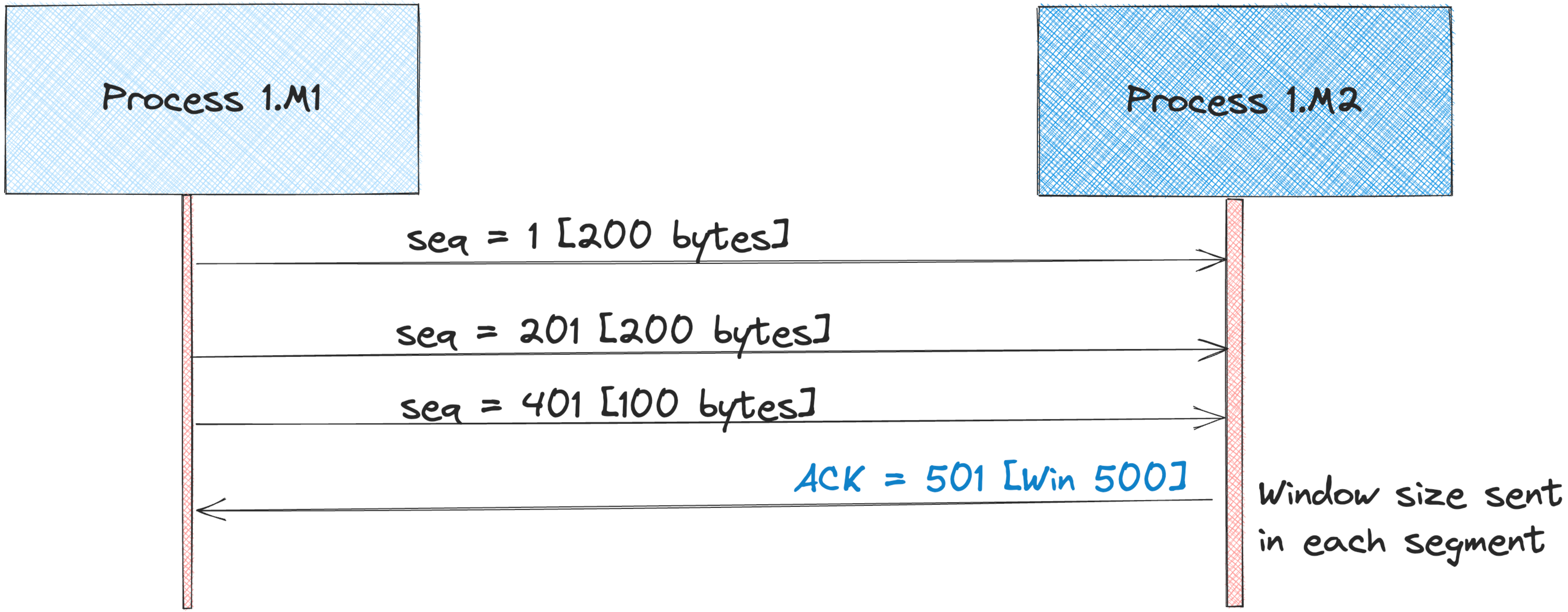
Can you please speak slowly? →

Person 1 Talking
(Process in a host A)

Person 2 Talking
(Process in a host B)

Sending too much data is also problem

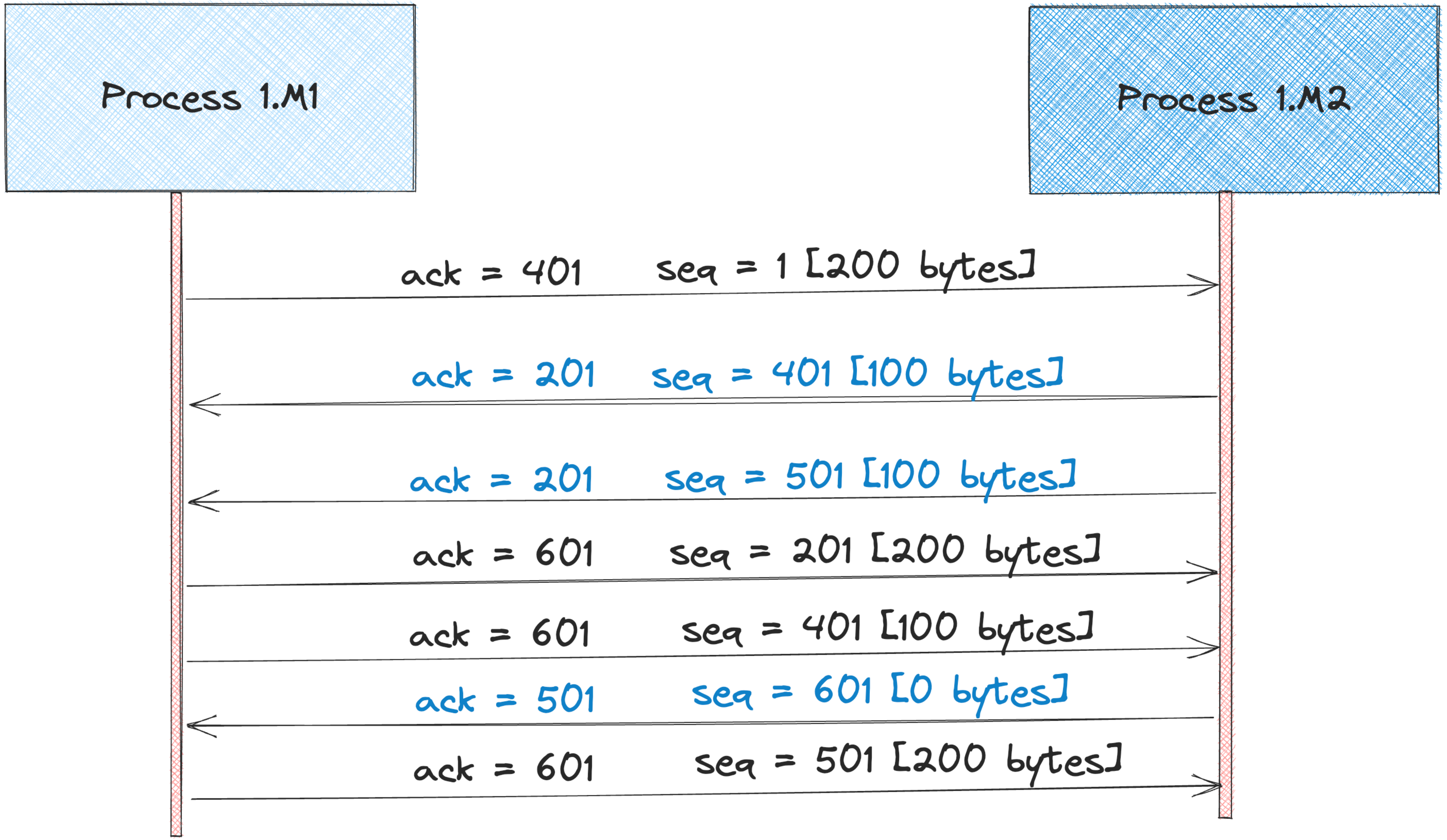
Window Size - Flow Control



- Dynamic update of Window size will enable flow control
- What if Process 1.M2 sends a windows size of 0?

TCP is bidirectional

Both Senders can send data



Process 1.M1
Process 1.M2
ack = 401 seq = 1 [200 bytes]
ack = 201 seq = 401 [100 bytes]
ack = 201 seq = 501 [100 bytes]
ack = 601 seq = 201 [200 bytes]
ack = 601 seq = 401 [100 bytes]
ack = 501 seq = 601 [0 bytes]
ack = 601 seq = 501 [200 bytes]

How to choose sequential numbers?

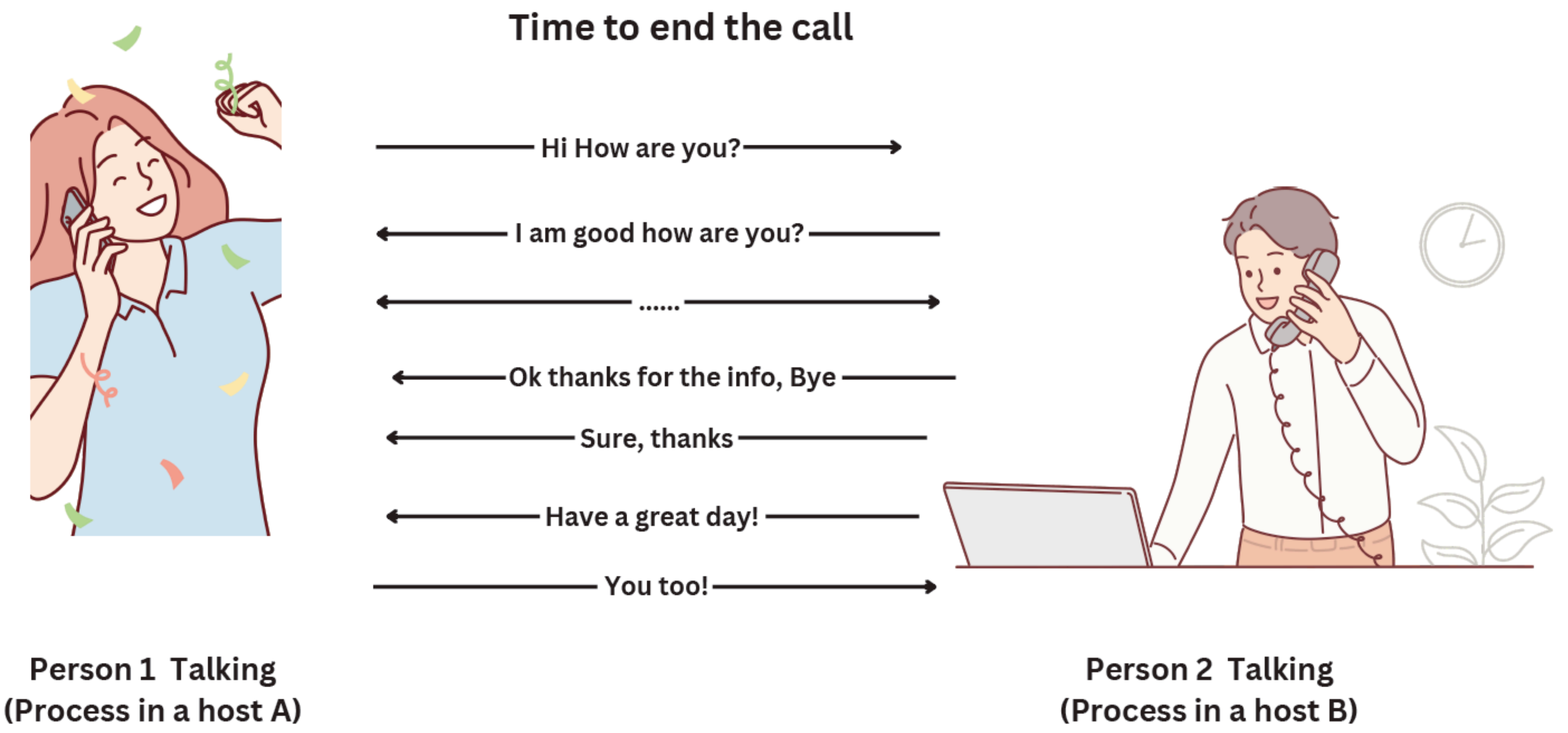
- Initial sequence numbers are randomly chosen by the senders
- Each can select a sequence number during the connection establishment
- Connection establishment in TCP happens through 3-way handshake
- The 3-way handshake consist of 4 events:
  - Process 1.M1 sends a connection request with SYN bit set and sequence number of X
  - Process 1.M2 acknowledges the connection request and sends back an ACK with X+1
  - Process 1.M2 also sends a request with the SYN bit set and sequence number [Y]
  - Process 1. M1 acknowledges the receipt by sending ACK [Y+1]

Three Way Handshake

Establishing Connection

Process 1.M1
Process 1.M2
set syn bit = 1
ack bit =0
seq = 100 [0 bytes] aCK = 0 [WIN = 100]
aCK = 101 seq = 300 [0 bytes] [WIN = 400]
set syn bit = 1
ack bit =1
set syn bit = 1
ack bit =0
ack = 301 seq = 101 [0 bytes] [WIN = 500]

Closing Connection



Time to end the call

Hi How are you? →

← I am good how are you?

← ...... →

← Ok thanks for the info, Bye

← Sure, thanks

← Have a great day!

You too! →

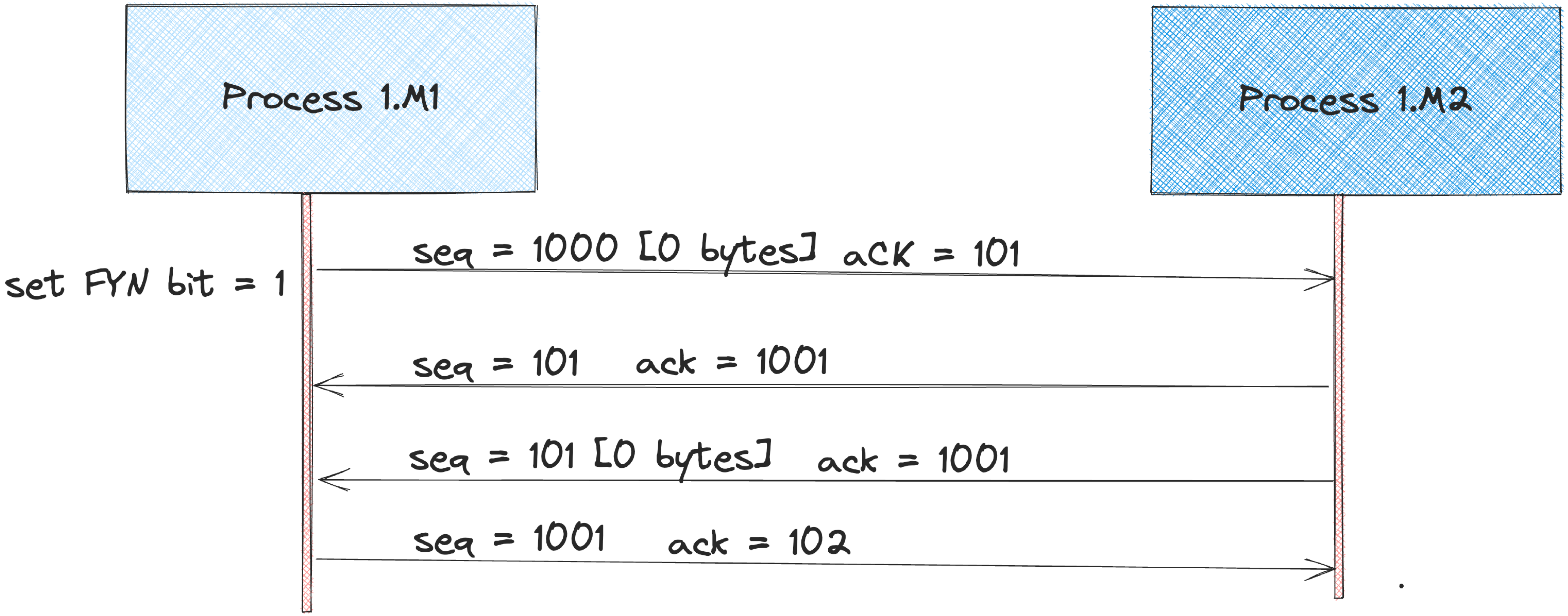
Person 1 Talking
(Process in a host A)

Person 2 Talking
(Process in a host B)

- TCP has two ways to close connection: **FIN** and **RST** flags

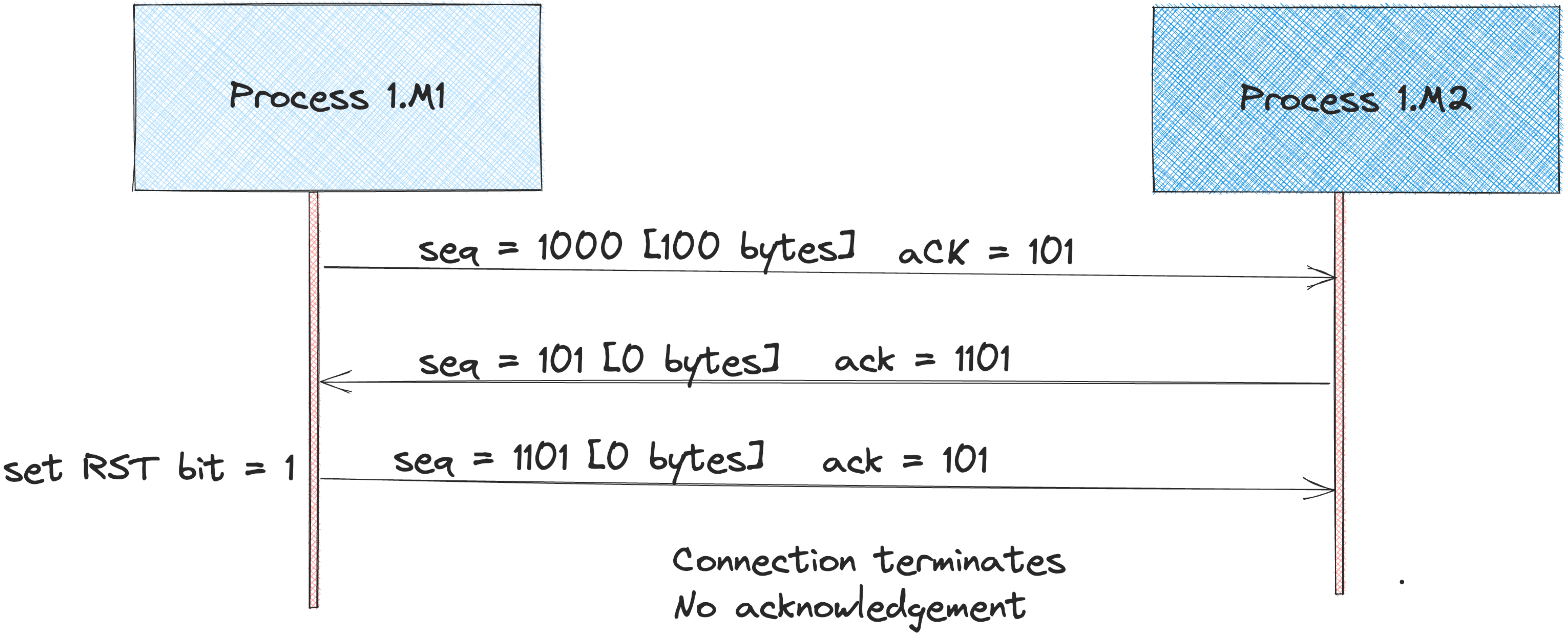
Using FYN bit

Graceful termination



Using RST Flags

Ungraceful closing


Process 1.M1
Process 1.M2
seq = 1000 [100 bytes] aCK = 101
seq = 101 [0 bytes] ack = 1101
set RST bit = 1
seq = 1101 [0 bytes] ack = 101
Connection terminates
No acknowledgement

But we need Memory!

How does OS handle the memory requirements of all these?

Where is the process stored? What about network buffer?

SCAN ME

Thank you

Course site: karthikv1392.github.io/cs3301_osn
Email: karthik.vaidhyanathan@iiit.ac.in
Twitter: @karthi_ishere