SRM UNIVERSITY *AP*

Andhra Pradesh

Computer Networks

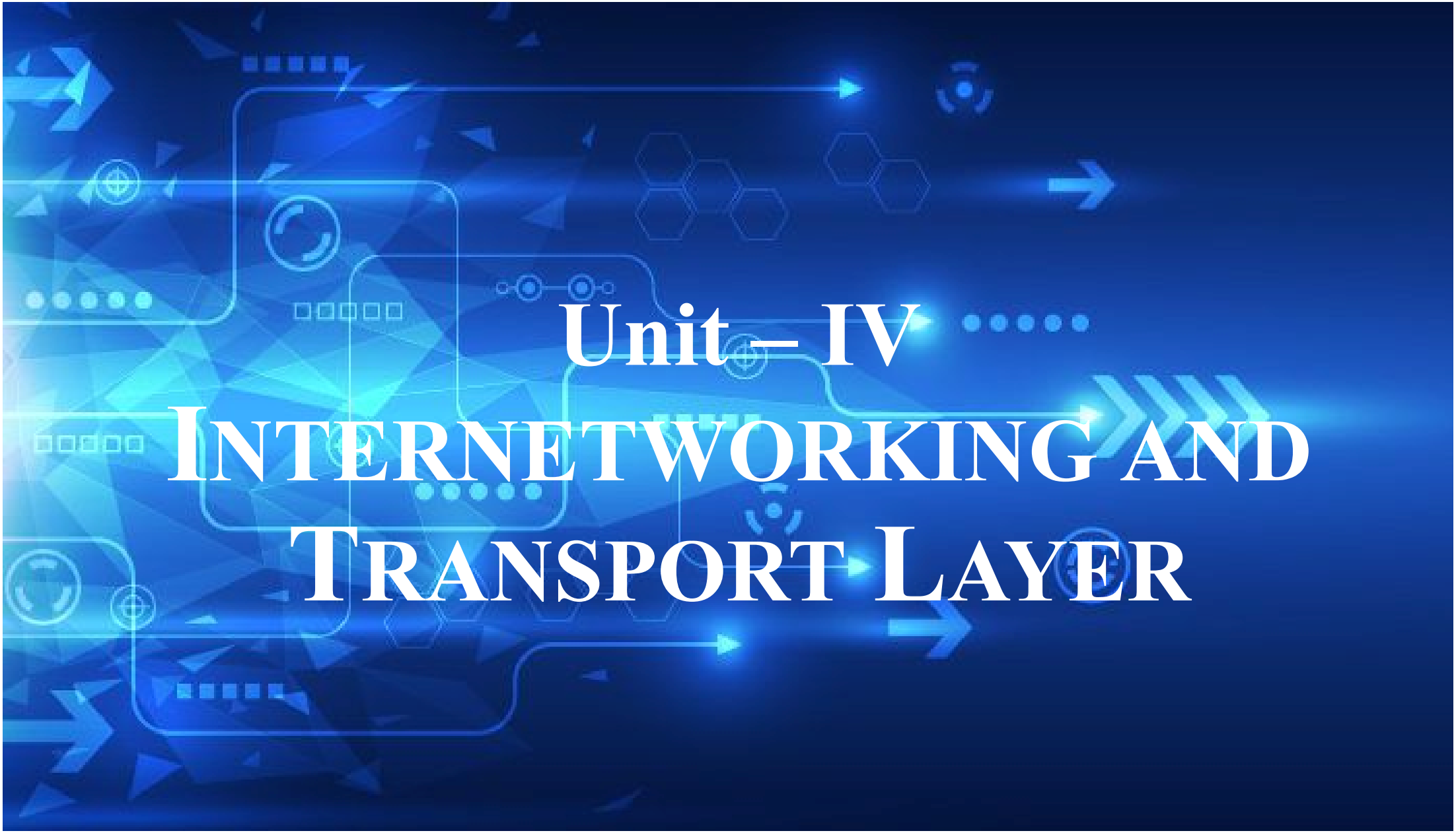
CSE 303

Course Instructor

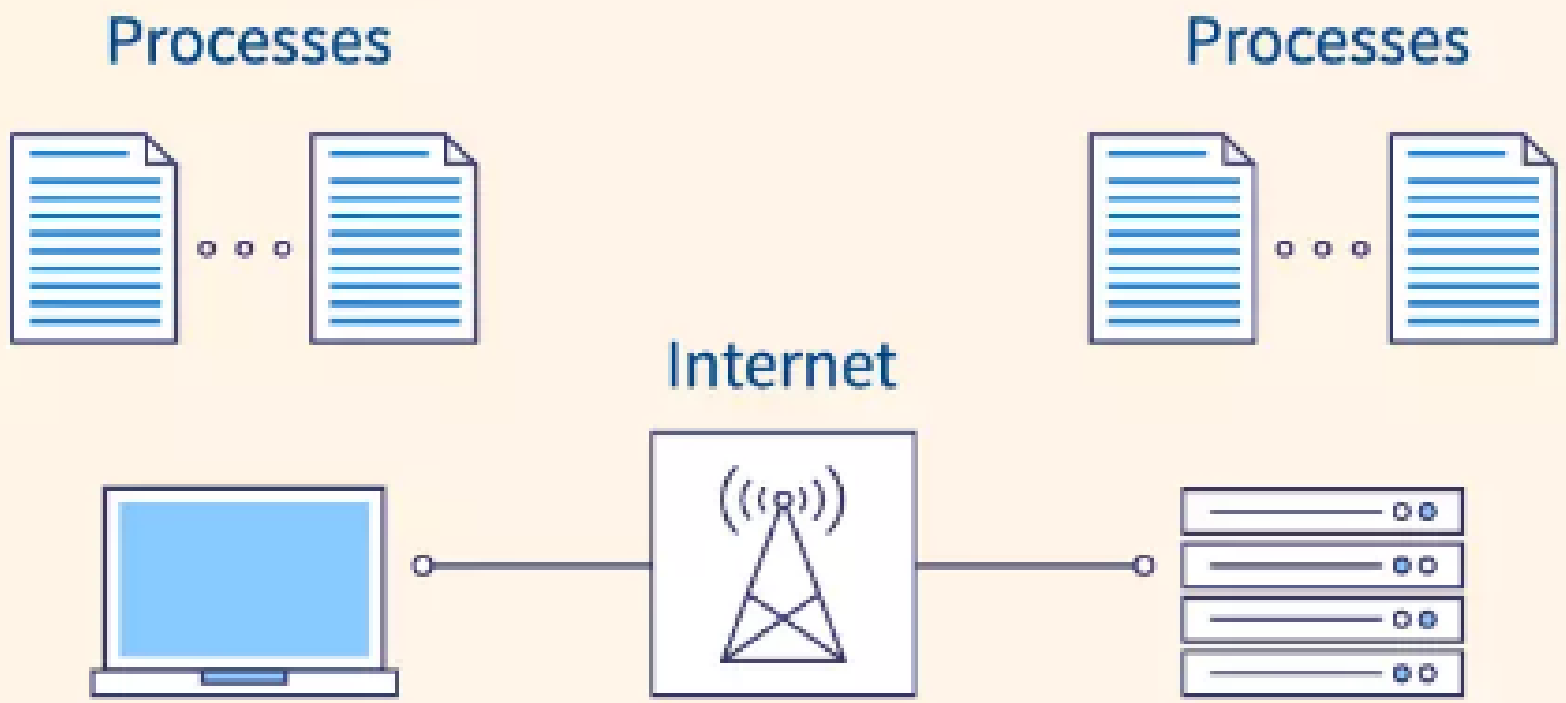
DR. HEMA KUMAR YARNAGULA

Assistant Professor

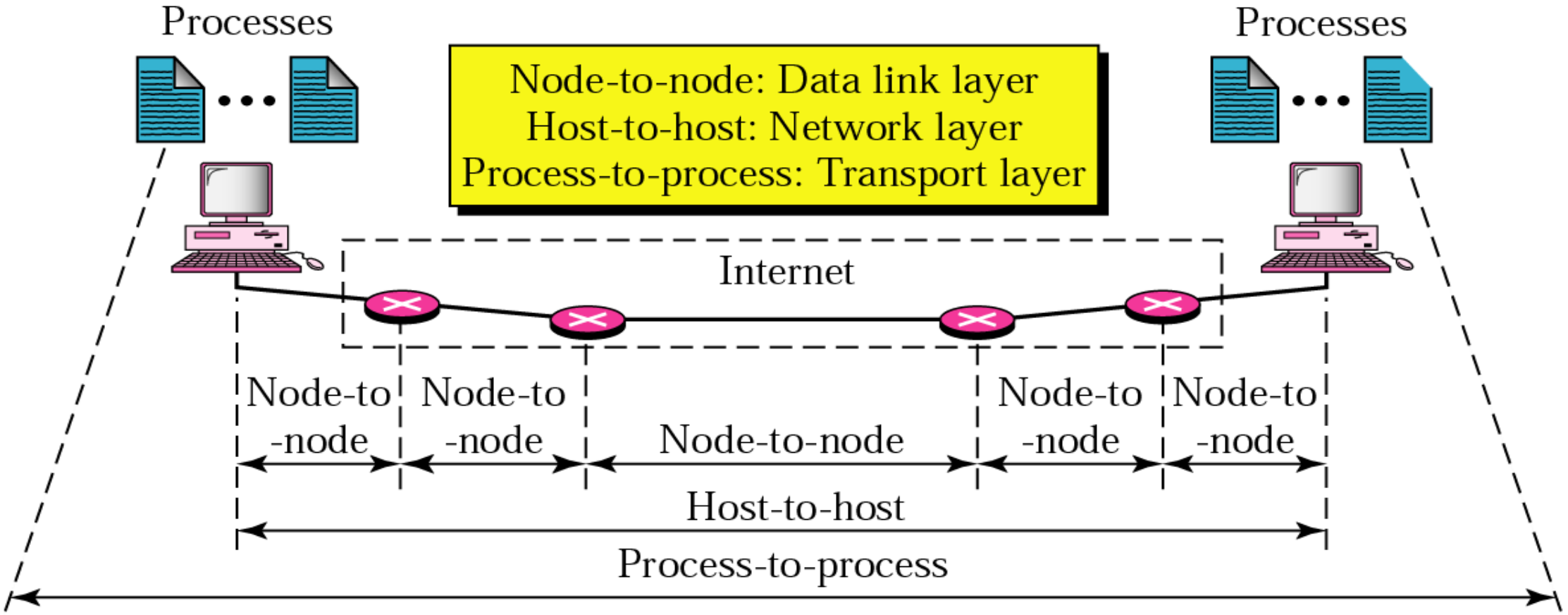
Dept. of Computer Sc and Engg., SRM University - AP


Unit– IV
INTERNETWORKING AND
TRANSPORT LAYER

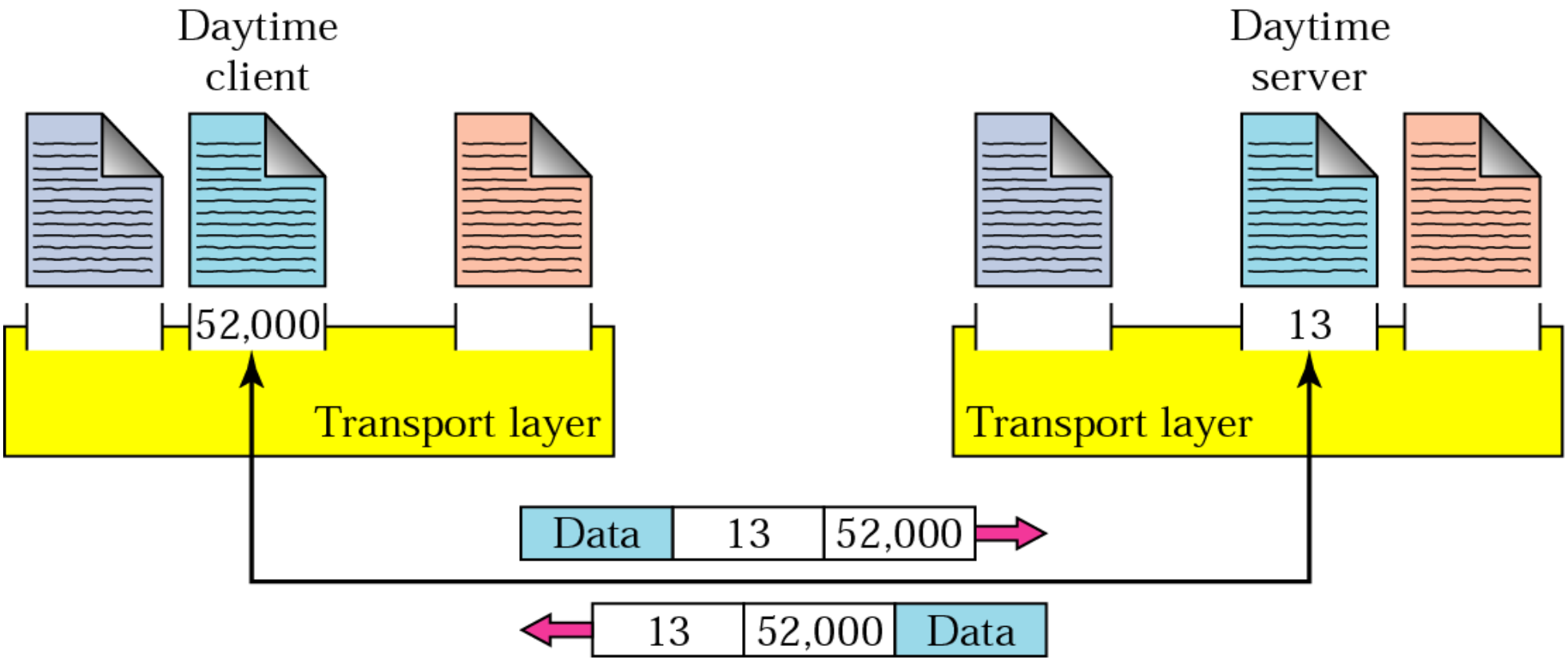
TRANSPORT LAYER



Types of Data Deliveries in CN

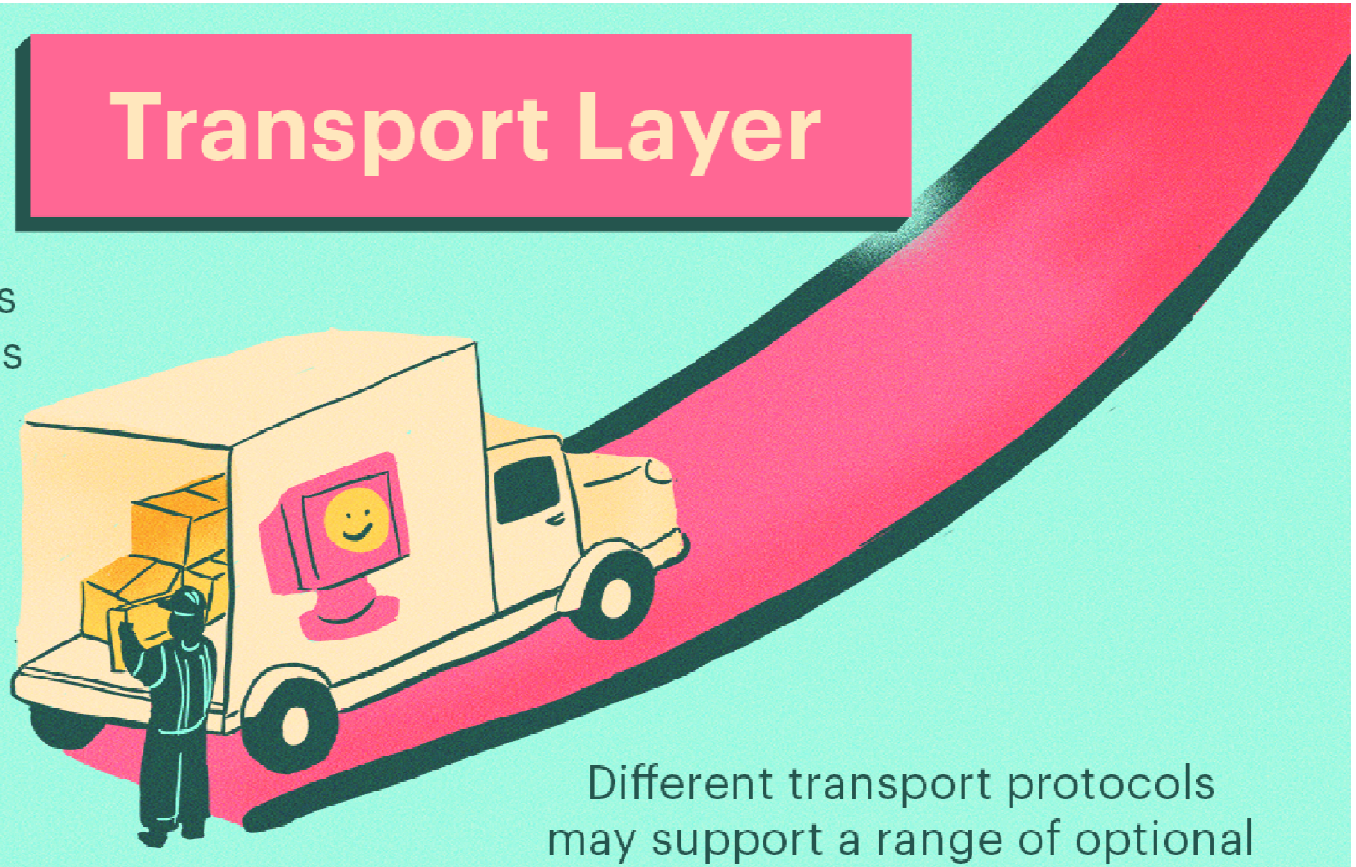


Types of Data Deliveries in Transport Layer



Transport Layer

Delivers data across network connections like TCP



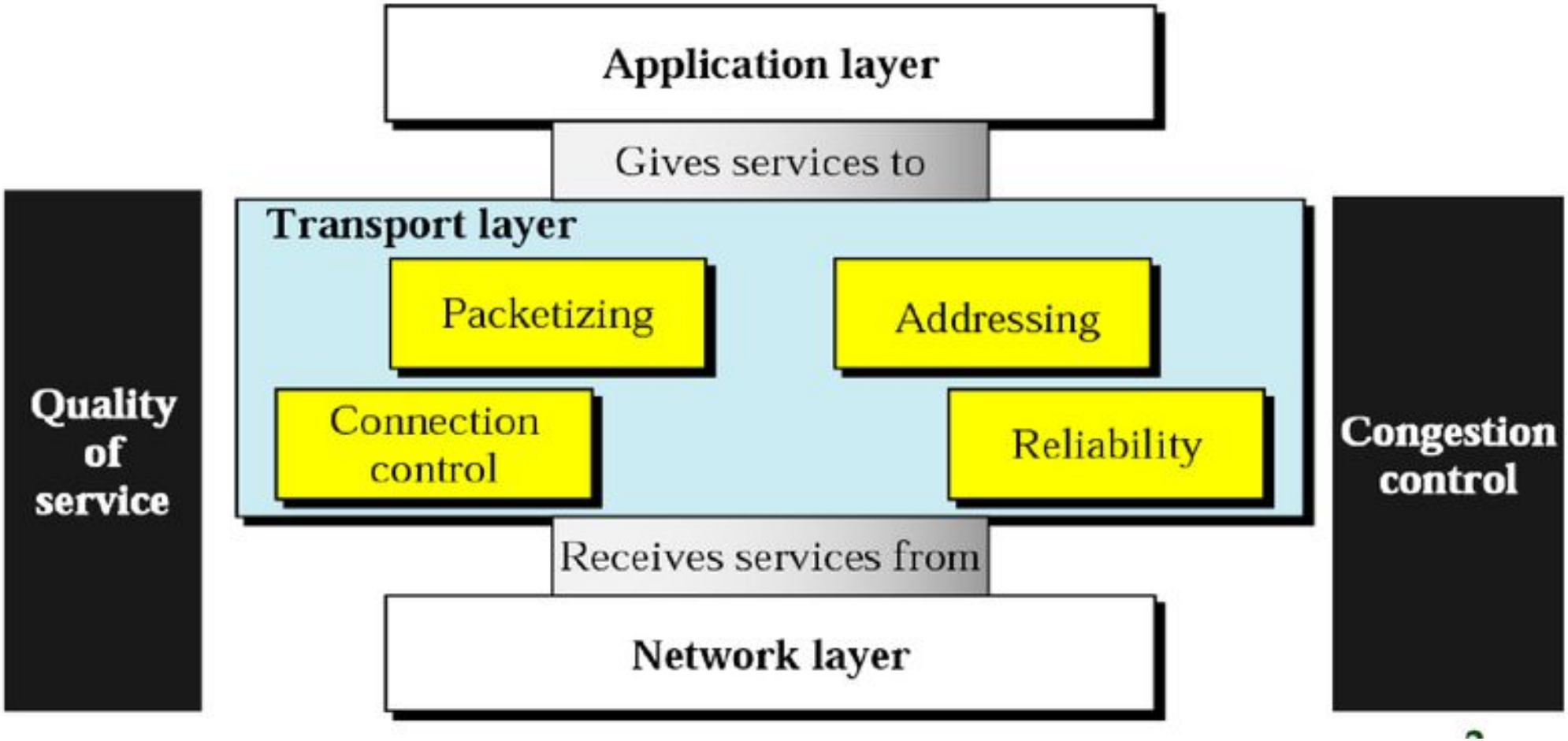
Different transport protocols may support a range of optional capabilities including:

Error recovery

Flow Control

Support for re-transmission

Role and Position of Transport Layer



Transport Services and Protocols

- Provide **logical communication** between app processes running on different hosts

- Transport protocols run in end systems (hosts)

Sender side:

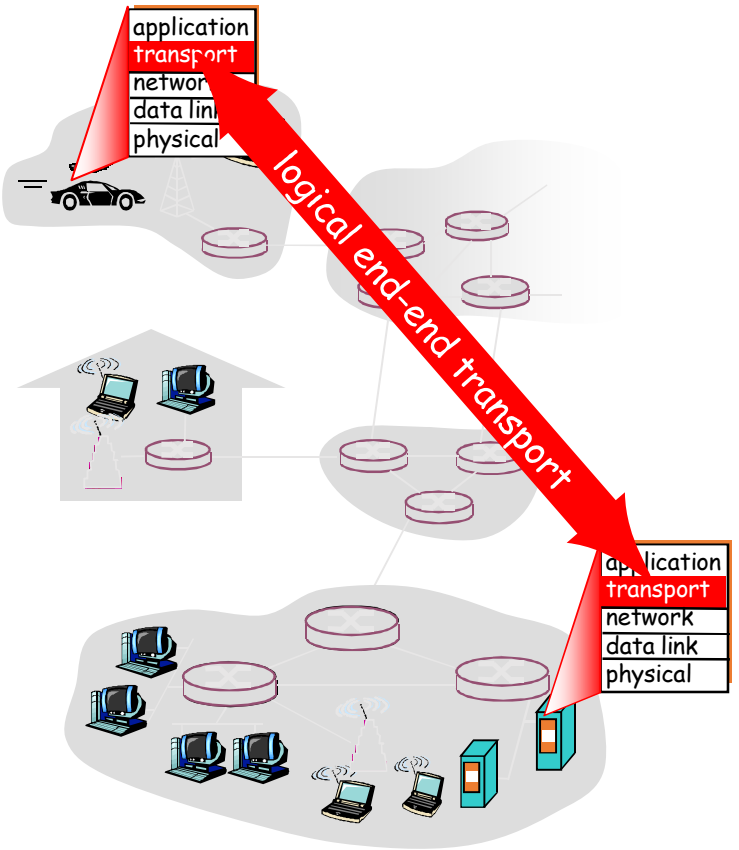
- breaks app messages into segments
- passes segments to network layer

Receiver side:

- reassembles segments into messages
- passes message to application layer

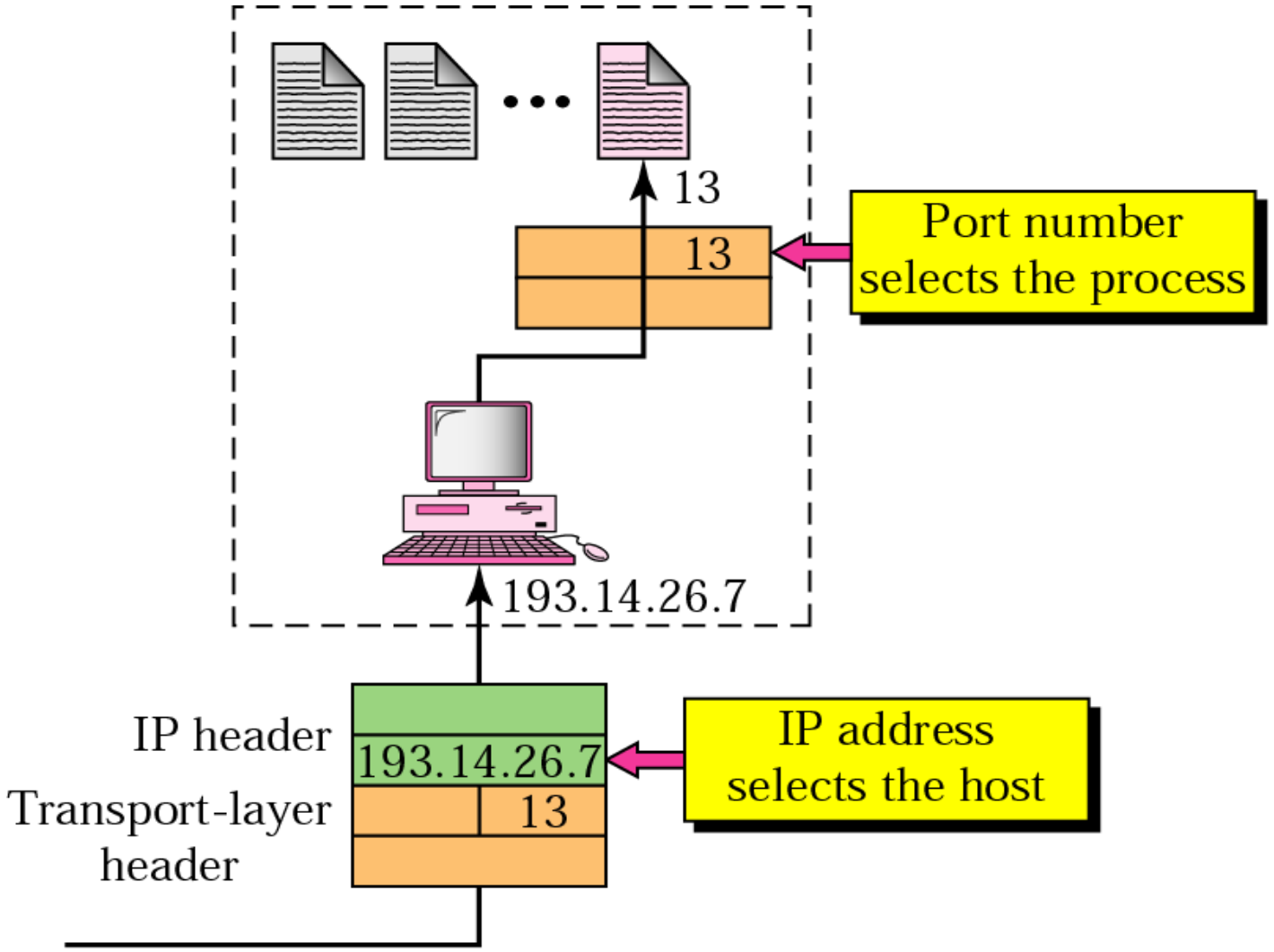
Transport Services and Protocols

- More than one transport protocol available to application
- Internet uses: TCP and UDP, SCTP



Note:
Network layer: logical communication between hosts
Transport layer: logical communication between processes

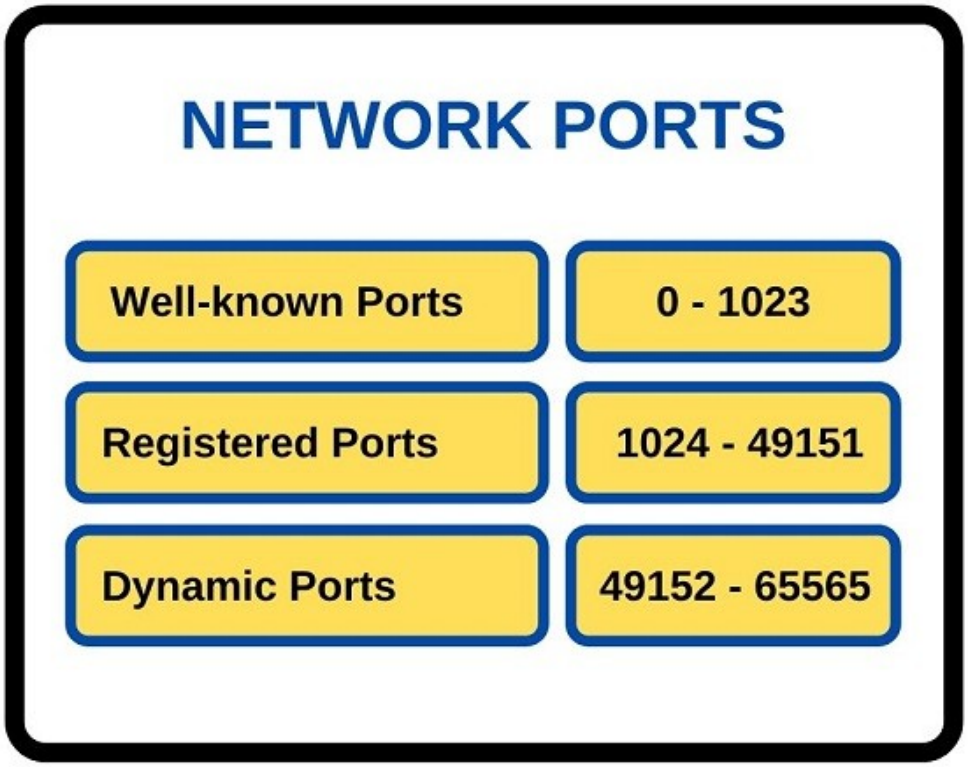
IP Addresses versus Port Numbers



WELL KNOWN

NETWORK PORT NUMBERS

Port Numbers



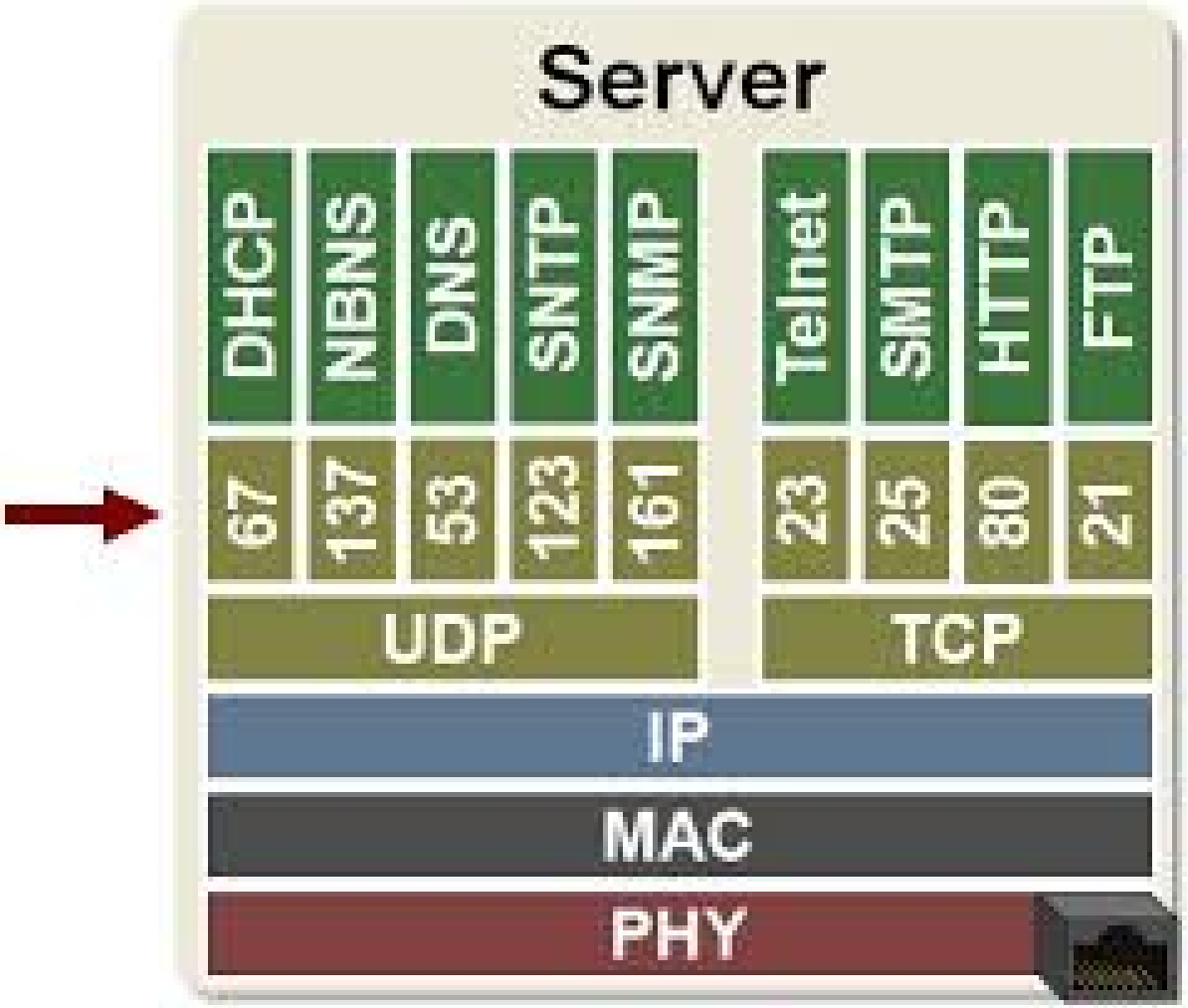
Port Number	Description
0	Well Known Ports
1	
2	
↓	
1023	
1024	Registered Ports
1025	
1026	
↓	
49151	
49152	Dynamic Ports (Private Ports)
49153	
49154	
↓	
65535	

Well Known Port Numbers

Port Number	Protocol
20, 21	File Transfer Protocol (FTP)
22	Secure Shell (SSH)
23	Telnet Protocol
25	Simple Mail Transfer Protocol (SMTP)
53	Domain Name System (DNS)
67, 68	Dynamic Host Configuration Protocol (DHCP)
80	HyperText Transfer Protocol (HTTP)
110	Post Office Protocol (POP3)
137	NetBIOS Name Service
143	Internet Message Access Protocol (IMAP4)
443	Secure HTTP (HTTPS)
445	Microsoft-DS (Active Directory)

Well Known Port Numbers versus Protocols

COMMON PORTS CHEAT SHEET

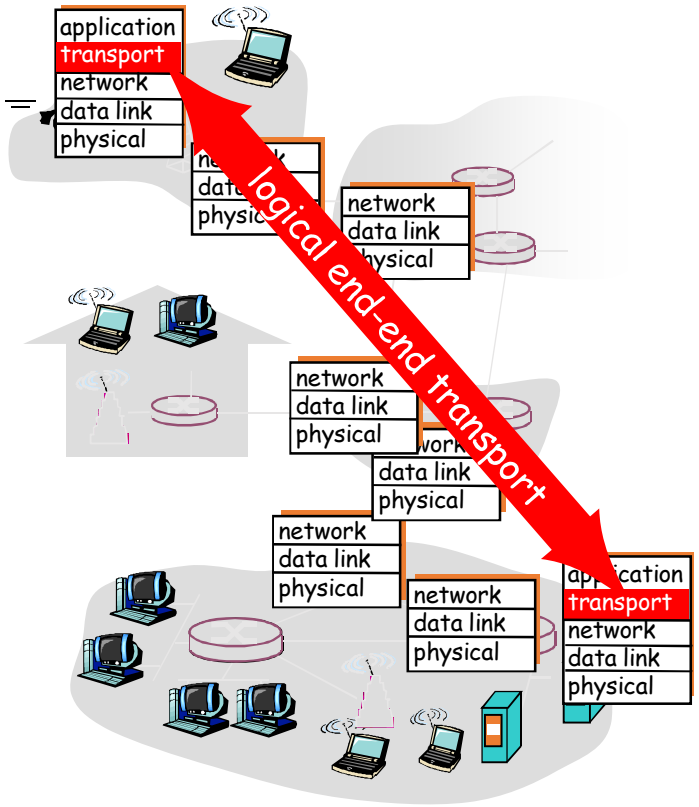


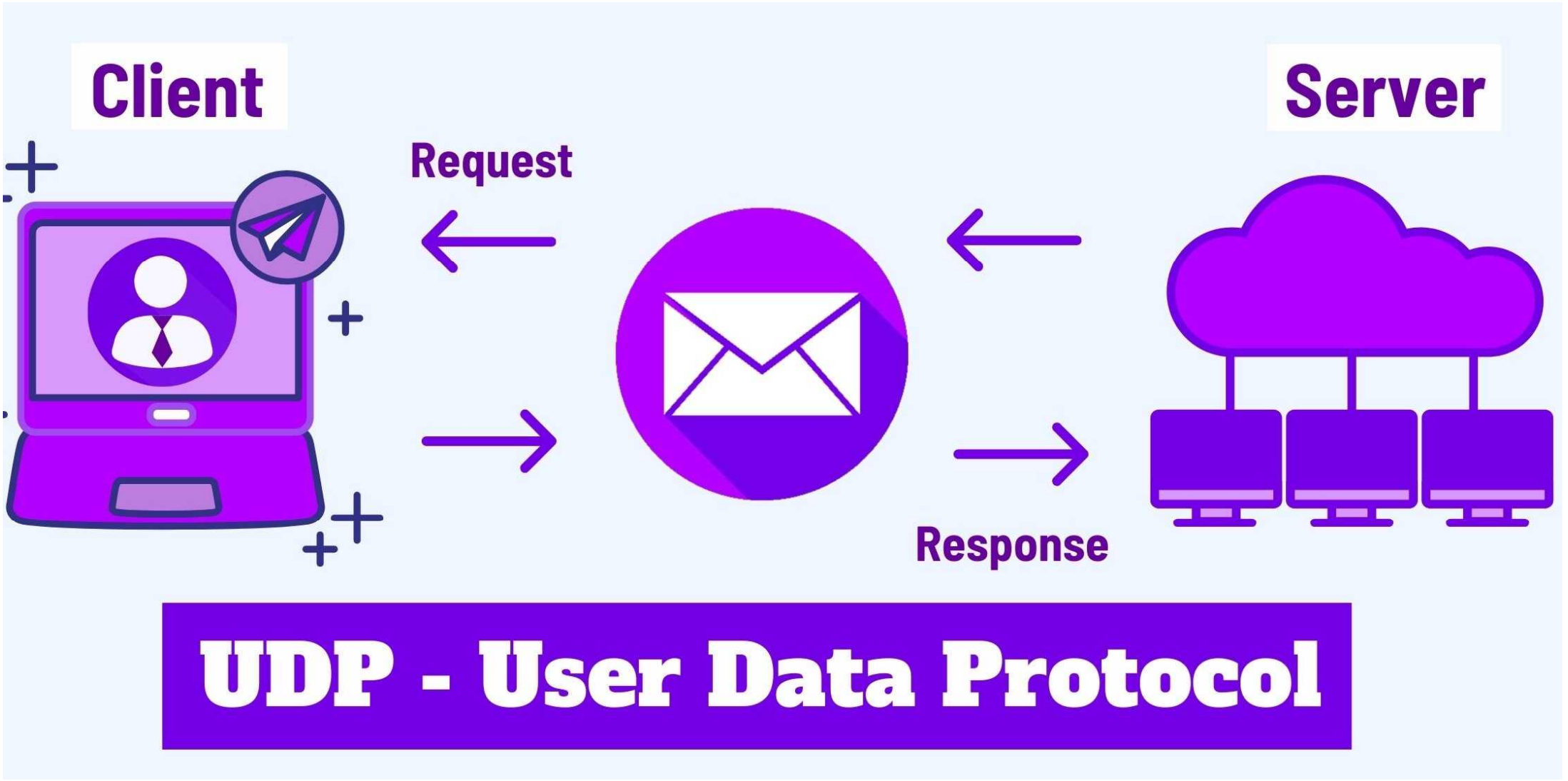
Internet Transport Layer Protocols

- The transport layer is represented by two protocols: TCP and UDP.

- **Reliable, in-order delivery: TCP**
  - Congestion control
  - Flow control
  - Connection setup

- **Unreliable, unordered delivery: UDP**
  - No congestion control
  - No flow control
  - No need for connection setup
  - Support for broadcast and multicast




Client
Server
Request
Response
UDP - User Data Protocol

UDP: User Datagram Protocol

- UDP stands for User Datagram Protocol.
- UDP is drafted in **RFC 768** in August 1980.
- UDP **provides a non-sequential transmission** of data.
- It is a **connectionless** transport protocol.

UDP: User Datagram Protocol

- UDP protocol is used in applications where the **speed and size** of data transmitted is considered as **more important** than the **security and reliability**.

UDP: User Datagram Protocol

- Provides "**best effort**" delivery
- UDP segments may be:
  - lost
  - delivered out of order to app
- *Connectionless Service:*
  - no handshaking between UDP sender, receiver
  - each UDP segment handled independently of others

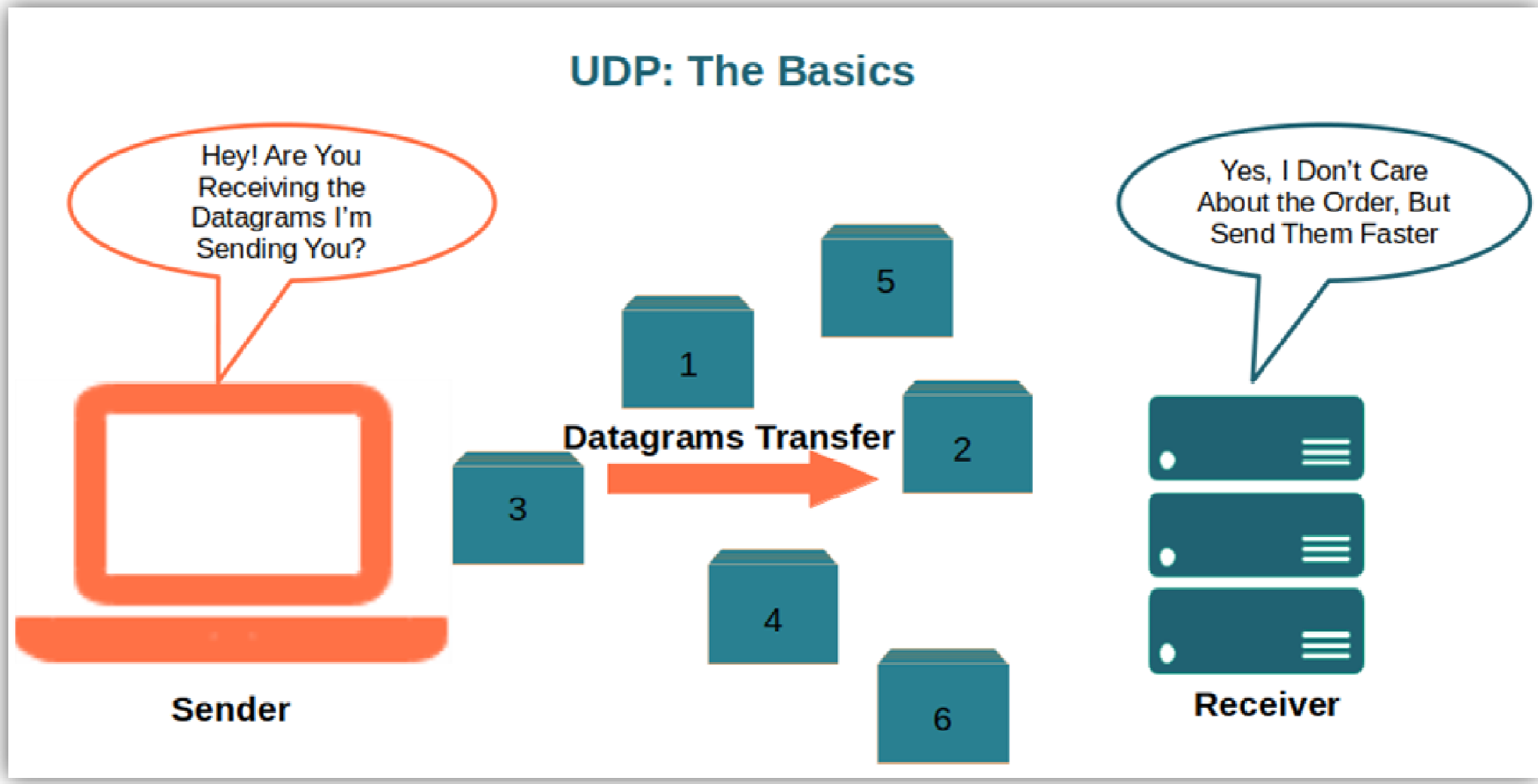
Why still UDP is there?

- No connection establishment (which can add delay)
- Simple: no connection state at sender, receiver

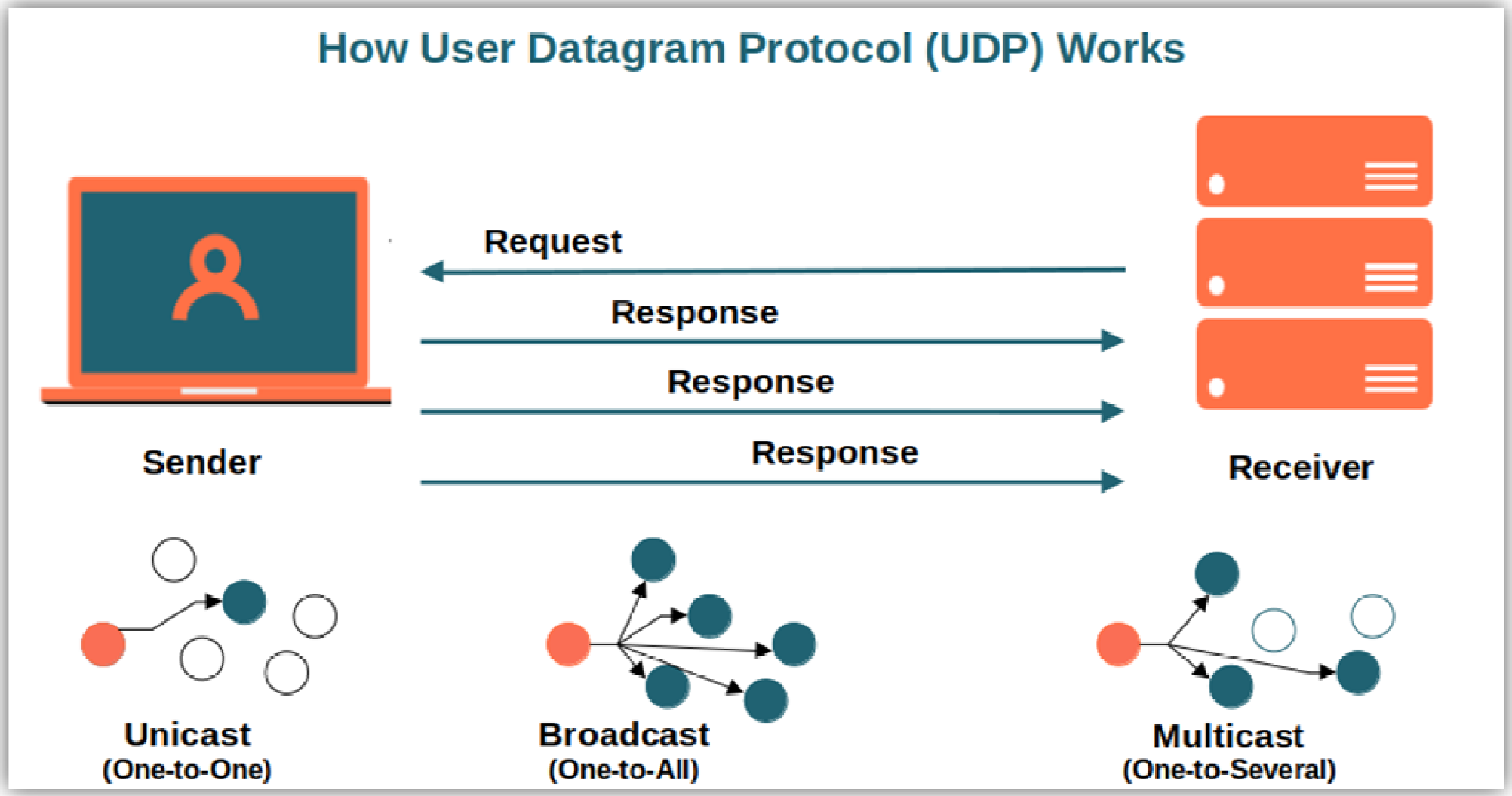
TCP – 20 Bytes, UDP – 8 Bytes

- Small segment header
- No congestion control: UDP can blast away as fast as desired

UDP: User Datagram Protocol



UDP: User Datagram Protocol



UDP: User Datagram

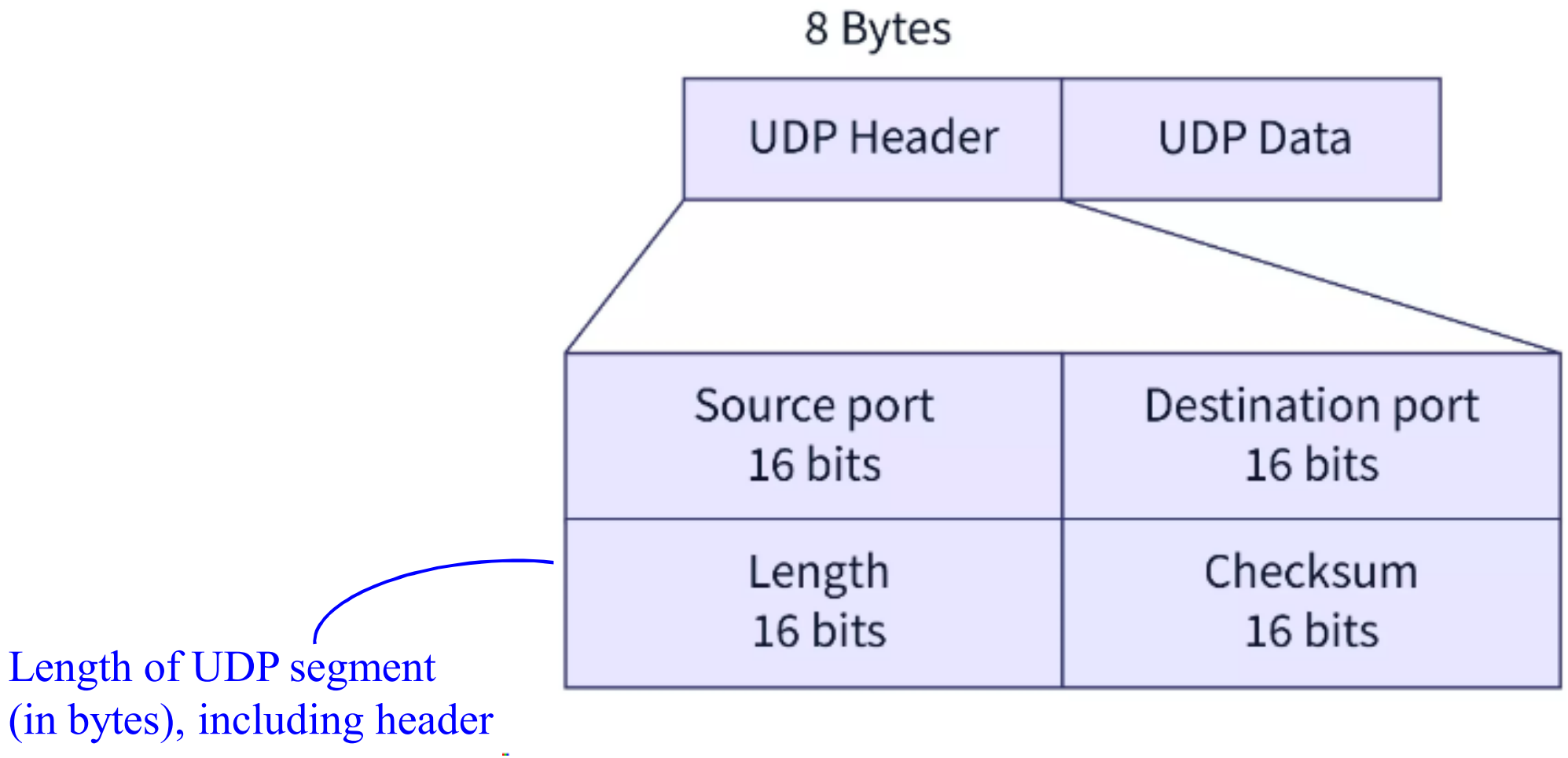
- **User Datagram** is defined as a packet produced by UDP Protocol.

- Each User Datagram has **two components**:
  - A header information
  - Data received from the above layer (i.e., Segment)

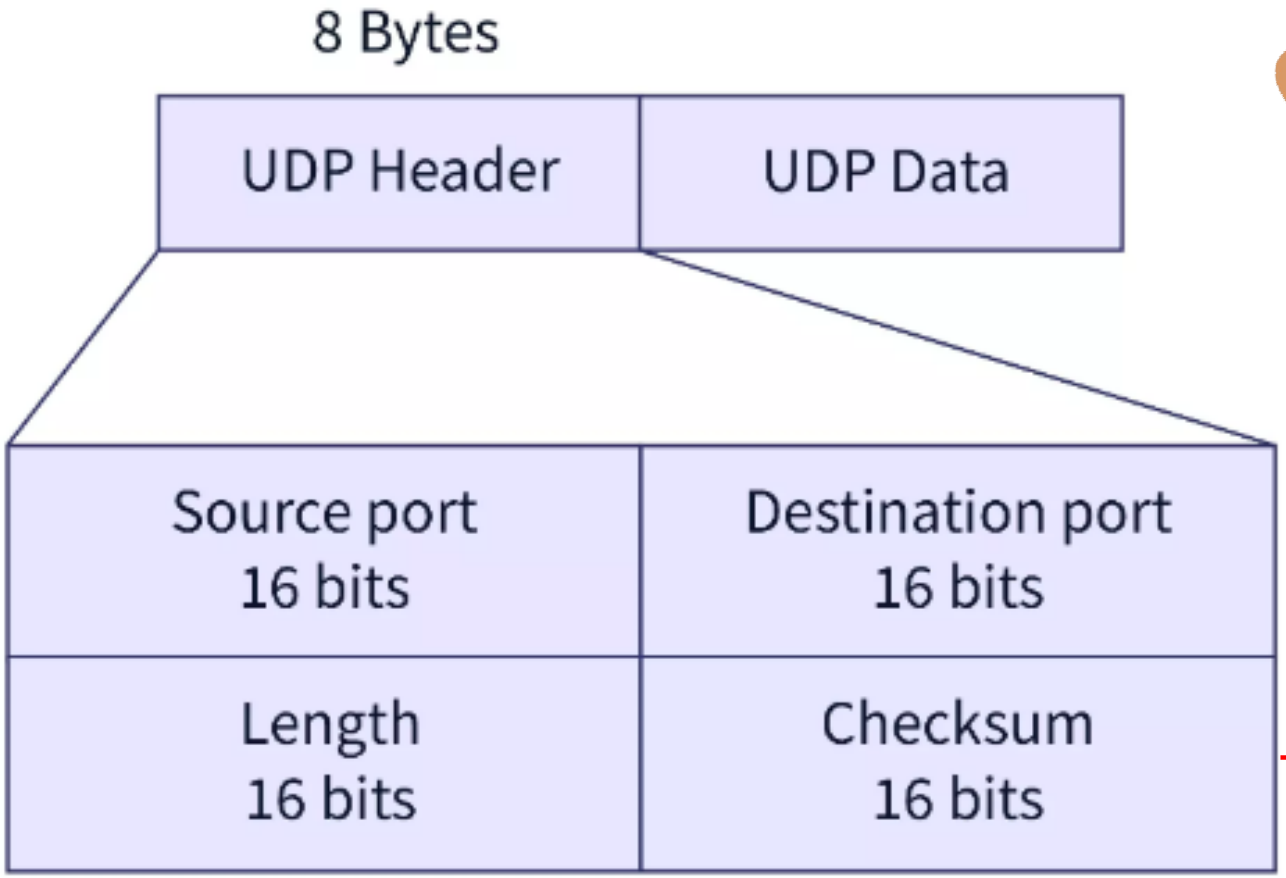
UDP: User Datagram

- UDP protocol adds the following into the Header:
  - Transport level addresses
    - Sender Port Number
    - Destination Port Number
  - Checksum for error control
  - Information of length to the data received from the layer above it

UDP: User Datagram Format



UDP: User Datagram Format



Checksum

once again??

UDP: Checksum

Goal: To detect "**errors**" (e.g., flipped bits) in transmitted Segment.

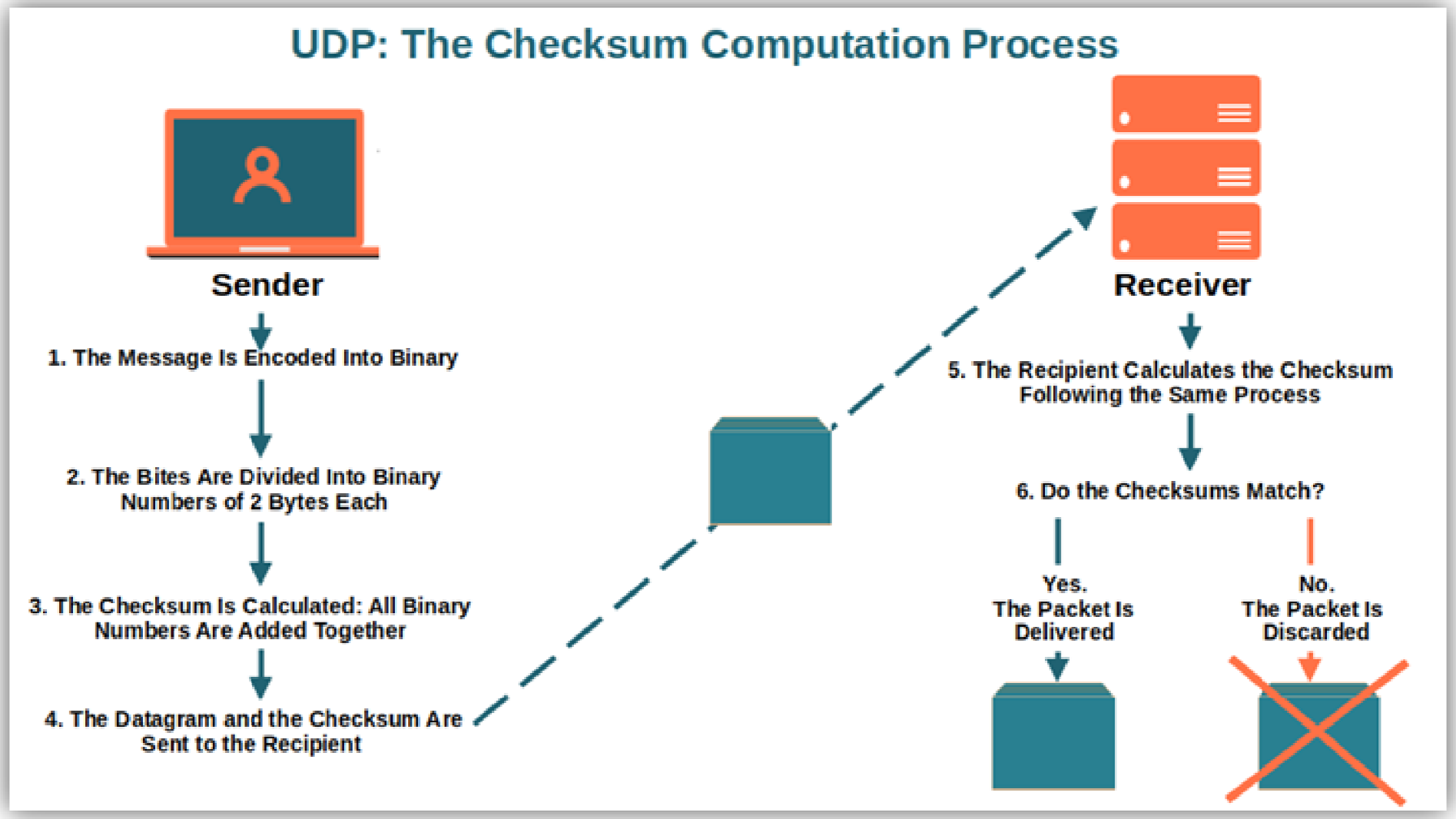
Sender:

- Treat segment contents as sequence of **16-bit integers**
- Checksum: addition of segment contents (1's complement sum)
- Sender puts checksum value into UDP checksum field

Receiver:

- Compute checksum of received segment
- Check if computed checksum equals checksum field value:
  - **NO** - error detected
  - **YES** - no error detected

UDP: Checksum



UDP: Checksum Example

- Three parts of 16 bits each:
  - 0110011001100110
  - 0101010101010101
  - 0000111100001111
- Adding the three, calling it '**r**':
  - 1100101011001010
- Send the four parts:
  - the original three and
  - 1's complement of 'r' to destination

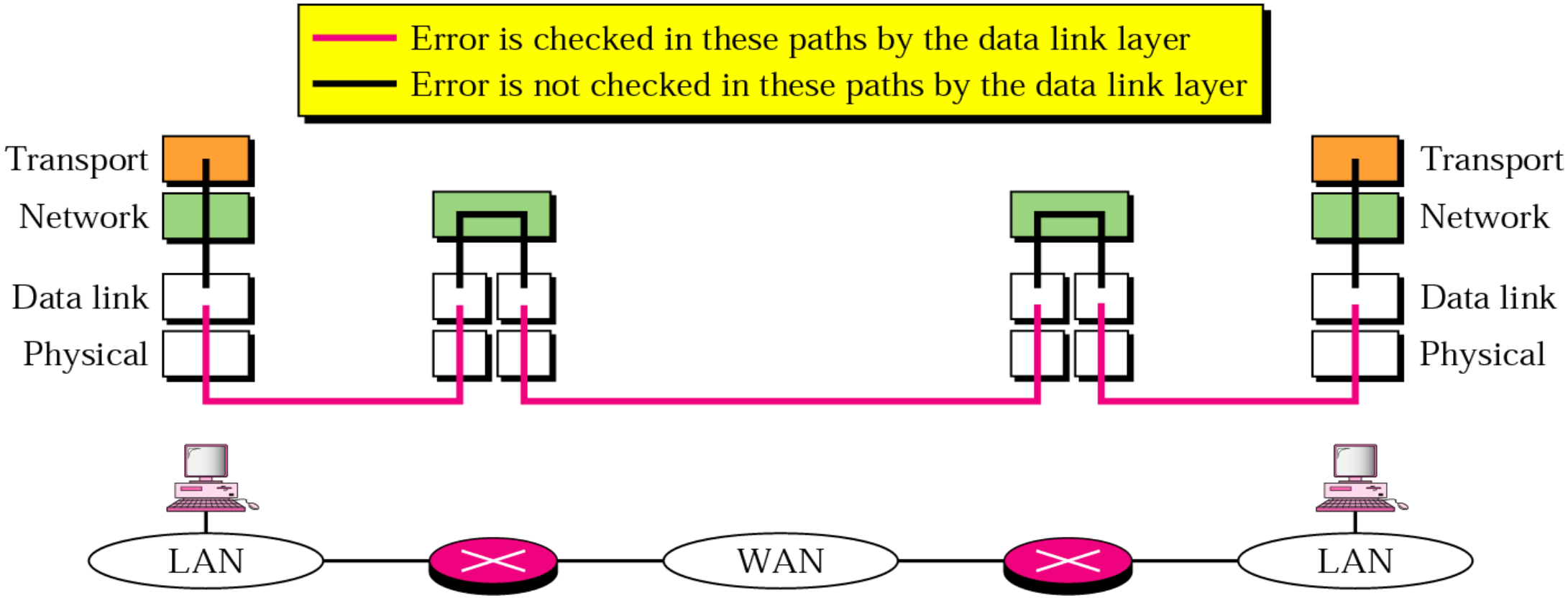
- The 1's complement of 'r' is:
  - 0011010100110101
- At destination, the sum of four parts should be:
  - **1111111111111111**
- **If the packet is damaged:**
  - **1111101111111111** (**zeros!!**)

UDP: Checksum

Why do we provide error checking, again?

- Because, **No guarantee** that it is provided in all of the links between source and destination

UDP: Checksum



Applications of UDP

- UDP is considered a suitable protocol for **multicasting**.
- UDP protocol can be utilized for **smaller size data**
- UDP is also used for **routing updates** such as in Routing Information Protocol (RIP).
- UDP protocol is generally used for **real-time applications**

Applications of UDP

Some of the **networking implementations that use UDP** as its transport layer protocol are:

- BOOTP, DHCP
- DNS (Domain Name Service)
- NNP (Network News Protocol)
- NTP (Network Time Protocol)
- Quote of the day protocol
- TFTP, RTSP, RIP

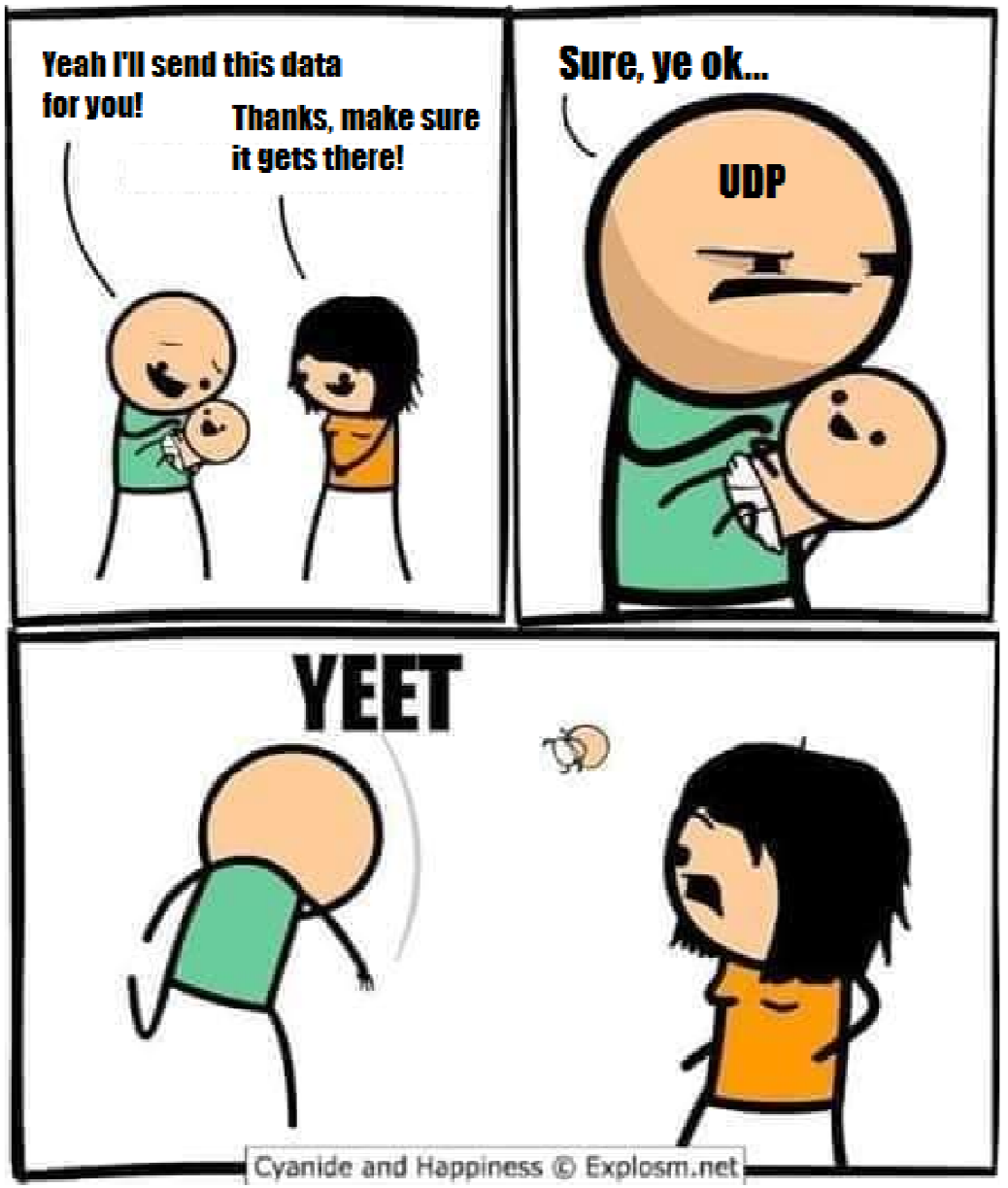
Advantages of UDP

- UDP also provides multicast and broadcast transmission of data.
- UDP protocol is preferred more for small transactions such as DNS lookup.
- It is a connectionless protocol, therefore there is no compulsion to have a connection-oriented network.
- UDP provides fast delivery of messages.

Disadvantages of UDP

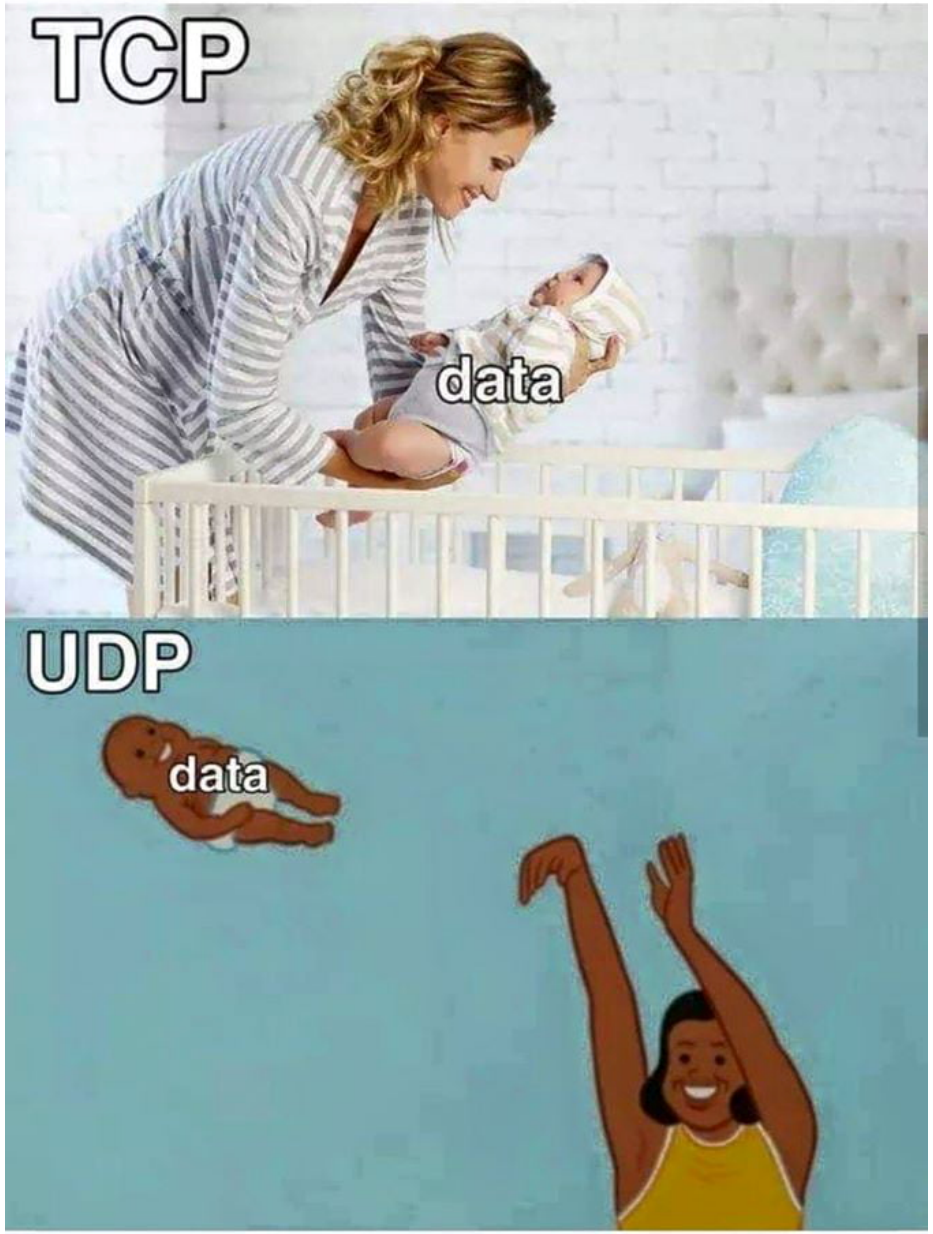
- In UDP protocol there is no guarantee that the packet is delivered.
- UDP protocol suffers from worse packet loss.
- UDP protocol has no congestion control mechanism.
- UDP protocol does not provide the sequential transmission of data.

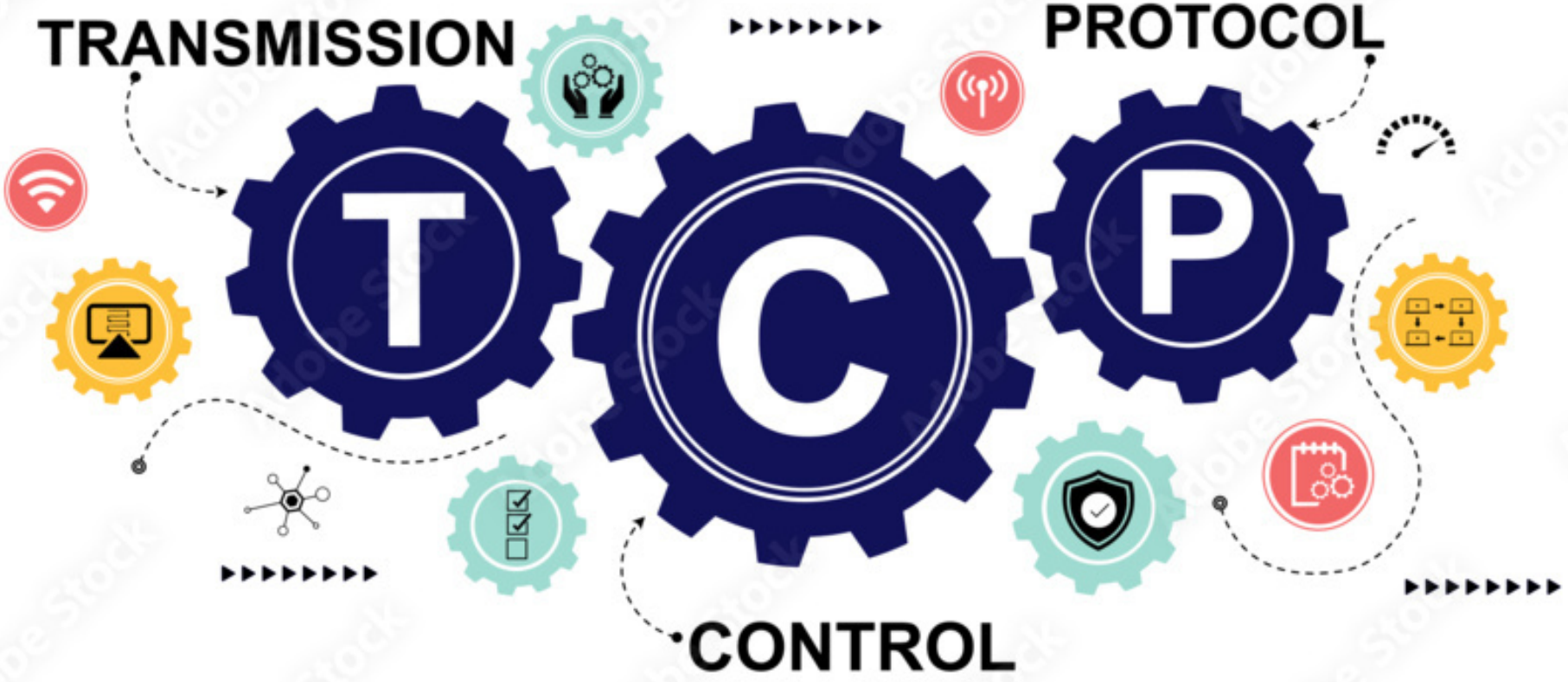
How UDP Works ==>



What if we need Guaranteed Delivery?

==>




TRANSMISSION
PROTOCOL
T
C
P
CONTROL

Transmission Control Protocol (TCP): Overview

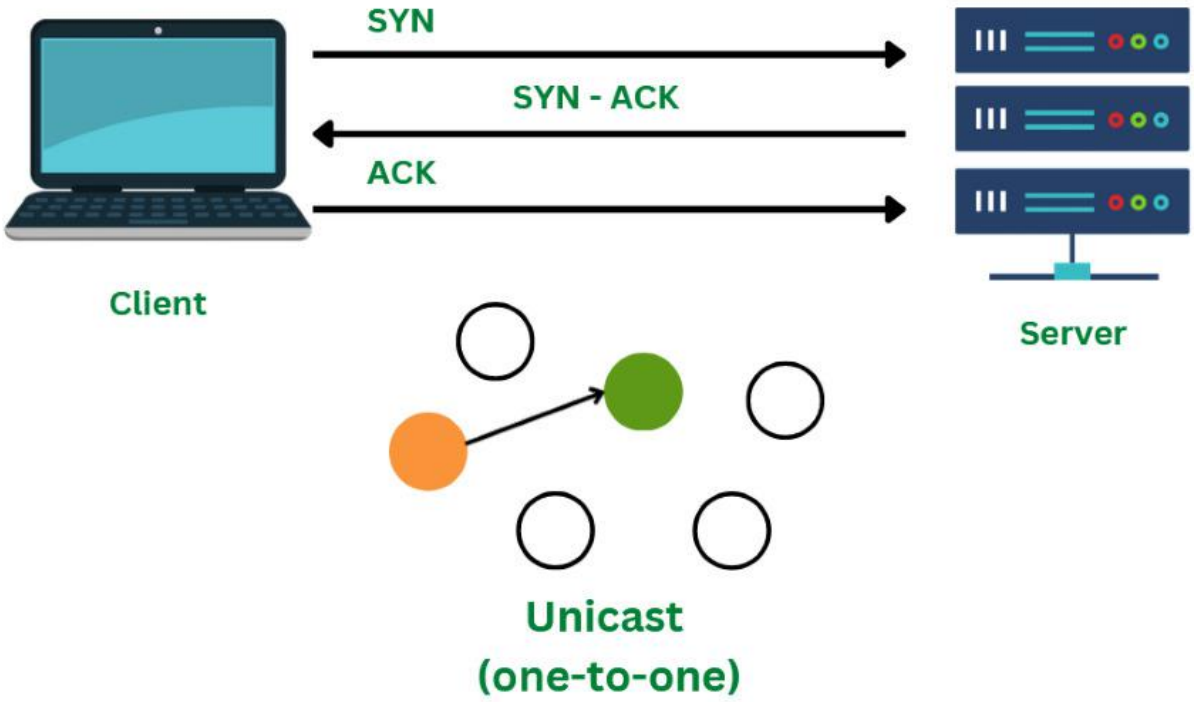
- TCP is one of the main protocols of the Internet protocol suite.
- TCP is drafted in RFCs: 793, 1122, 1323, 2018, 2581
- It is a **connection-oriented protocol** for communications
- The Internet Protocol (IP), which establishes the technique for sending data packets between computers, works with TCP.

TCP: Overview

Features of TCP

Point-to-Point (Unicast):

- All communication has one sender, one receiver



Features of TCP

Connection Oriented:

- A **secured connection is being established** between the sender and the receiver.
- **How:** using handshaking (exchange of control msgs)
- It means sender and receiver are connected to each other till the completion of the data transfer process.
- The order of the data is maintained i.e. order remains same before and after transmission.

Features of TCP

Full Duplex Connection:

- **Bi-directional data flow** in same connection
- Data can be transmitted from receiver to the sender or vice - versa at the same time.
- MSS: maximum segment size
- It **increases efficiency** of data flow between sender and receiver.

Features of TCP

Flow Control:

- Flow control limits the rate at which a sender transfers data.
- **Aim:** Sender will not overwhelm receiver
- This is done to ensure reliable delivery.
- The receiver continually hints to the sender on how much data can be received (using a sliding window)

Features of TCP

Congestion Control:

- TCP takes into account the level of congestion in the network
- **Aim:** sender will not overwhelm network
- Congestion level is determined by the amount of data sent by a sender

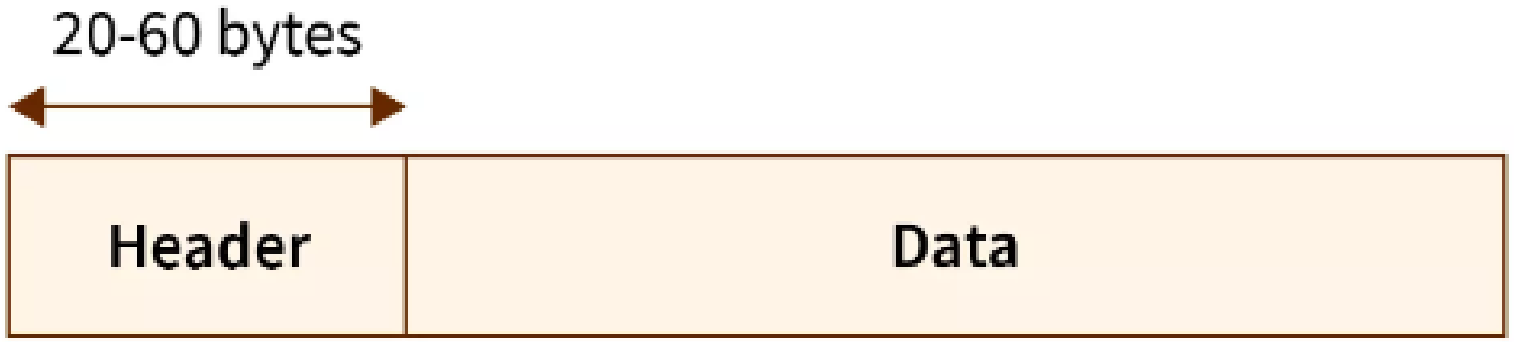
Features of TCP

Error Control:

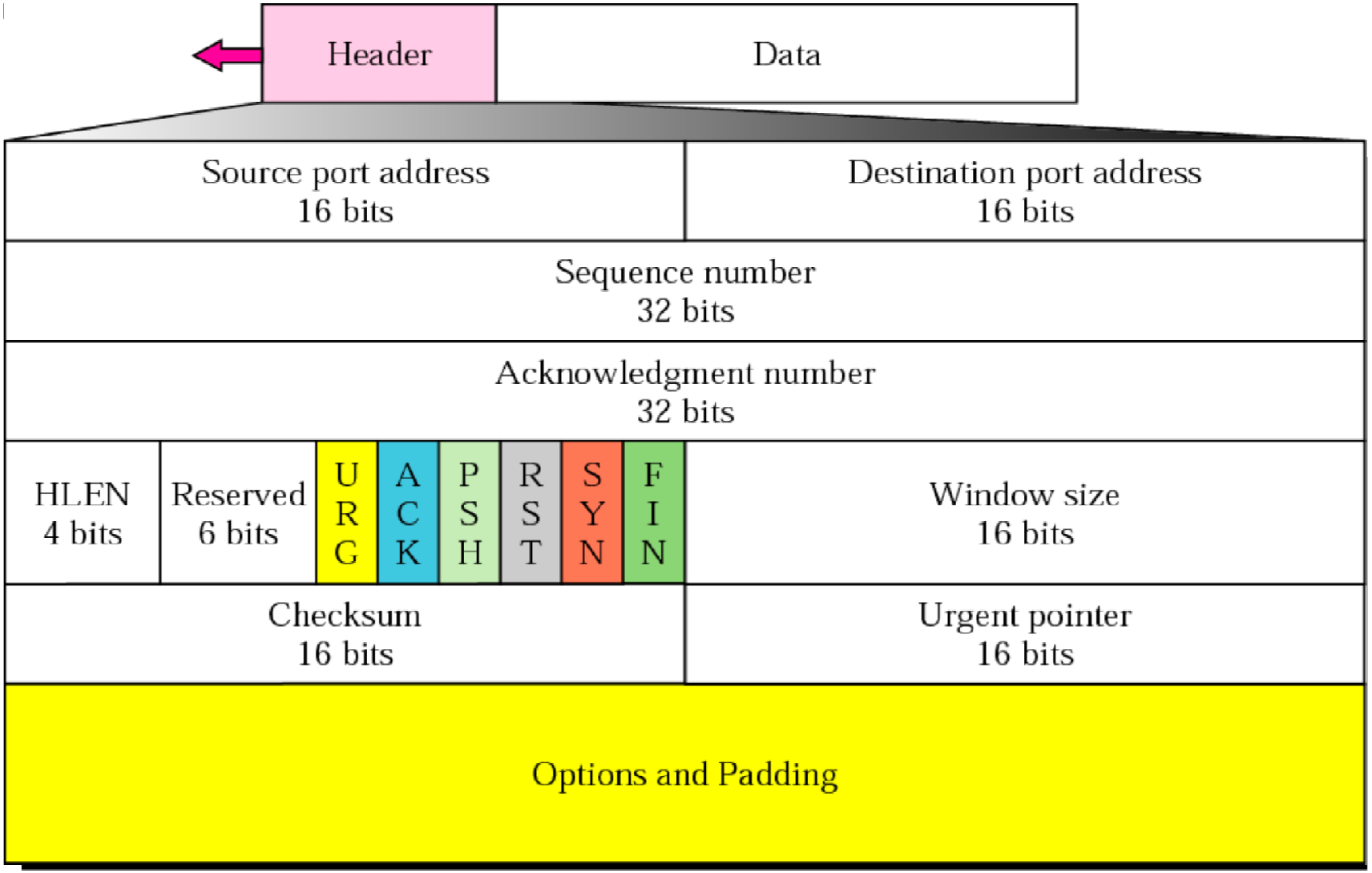
- TCP implements an error control mechanism for reliable data transfer
- Error control is byte-oriented
- Segments are checked for error detection

TCP Segment Structure

- TCP segment contains two things:
  - Header Field
  - Data Field

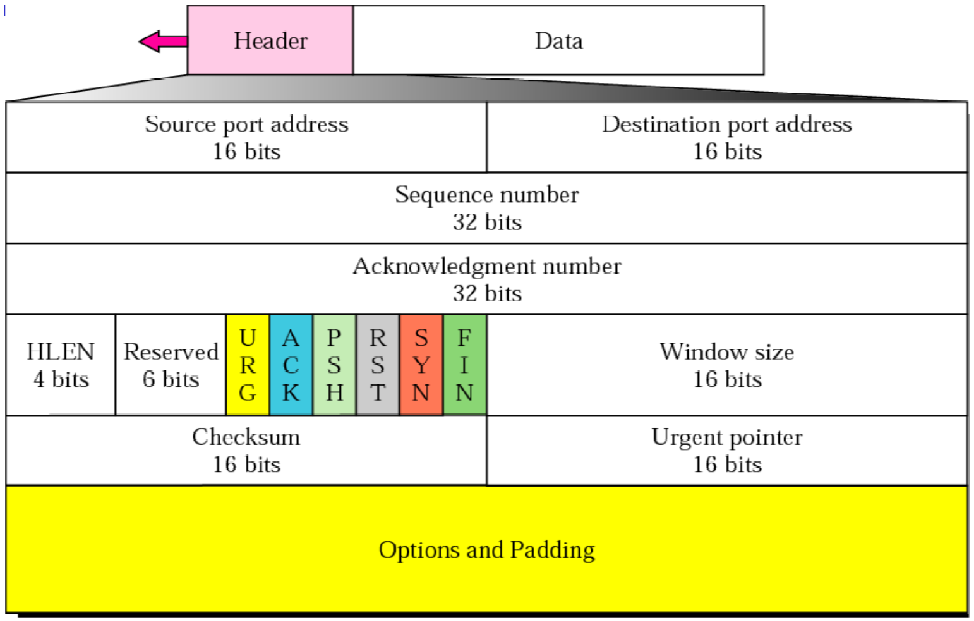


TCP: Header Format



TCP: Header Format

- The header of the TCP is of:
  - **minimum 20 bytes and**
  - **maximum of 60 bytes.**



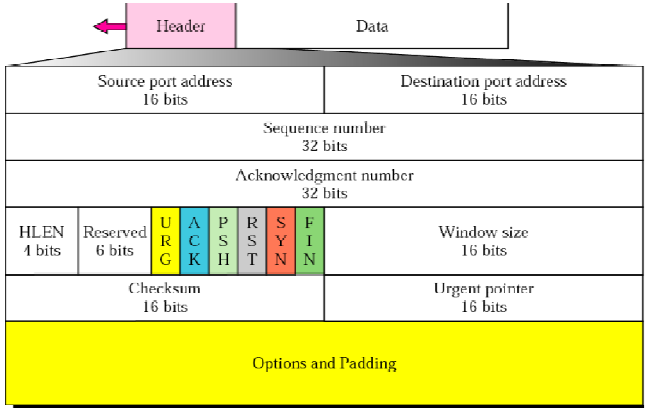
Let us briefly discuss the various fields:

- **Source Port:** Defines the port address of the source.
- **Destination Port:** Defines the port address of the destination.
- **Sequence Number:** It contains the sequence number of a segment or data bytes in a session.

TCP: Header Format

Acknowledgment Number:

- ACK flags contain next sequence number of the segment of data
- It works as an acknowledgment for the previously received data.

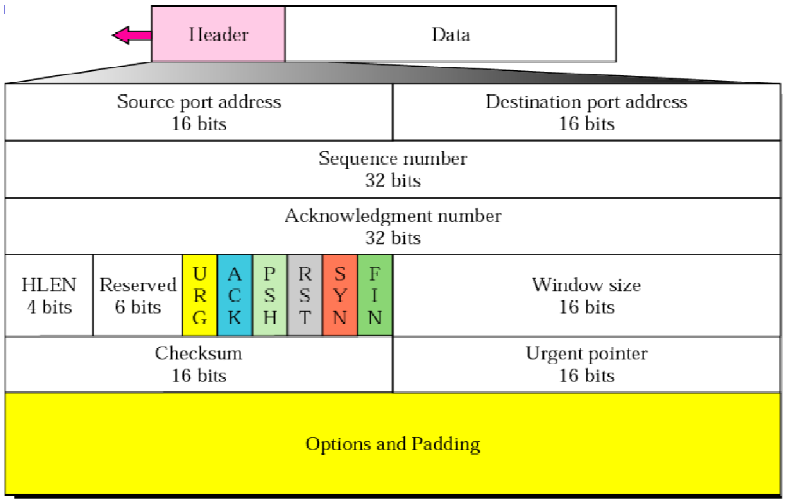


HLEN:

- HLEN stands for Header Length.
- It has **4-bit value**.
- Using 4 bits, it specifies the length of the header.
- The **minimum value for HLEN is 5 and maximum is 15**.

- **Reserved:** It is reserved for any future use and it is of 6-bits.

TCP: Header Format



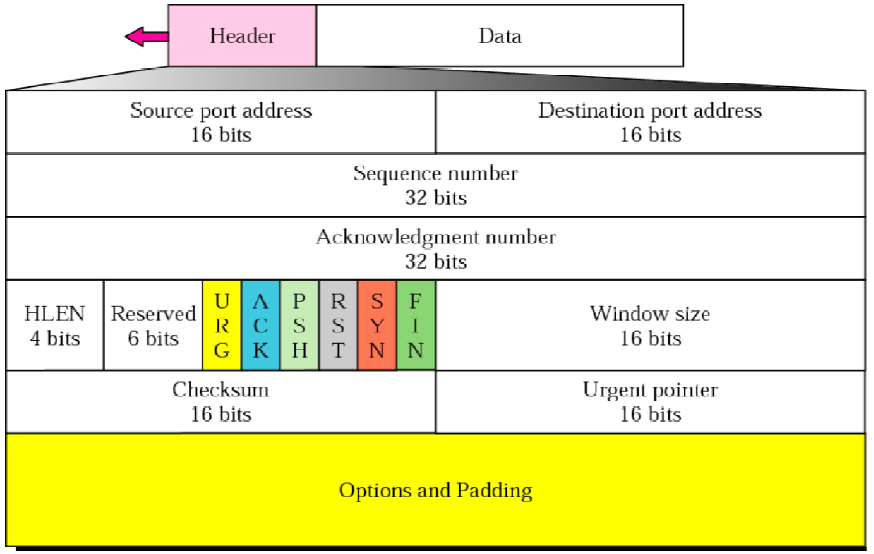
Flags:

- The flag is a 1-bit value.
- Flags are of mainly 6 types and are also known as control bits.

1. **URG**: URG represents an urgent pointer, so, if the URG value is set to 1 then the data is processed urgently.

2. **ACK**: ACK denotes acknowledgment, so, if its bit value is set to 0 then the data packet does not contain an acknowledgment.

TCP: Header Format

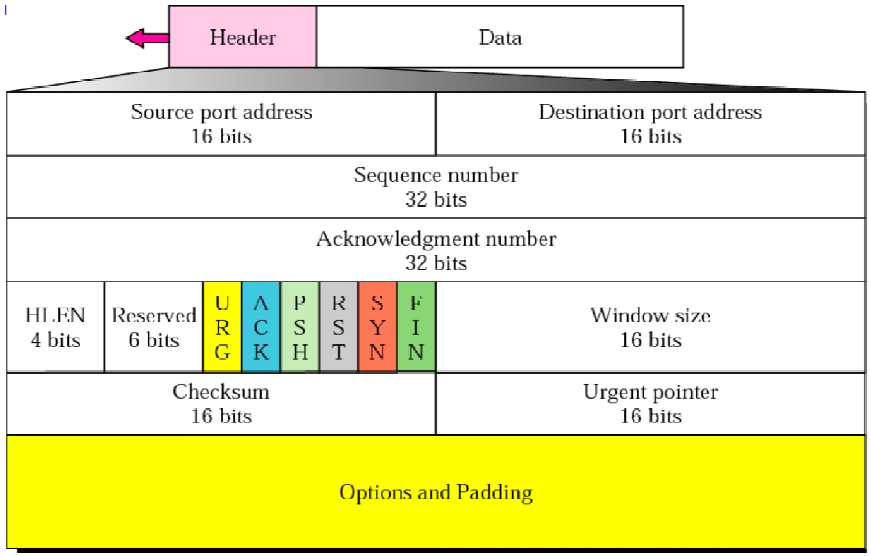


Flags:

3. **PSH:** PSH represents PUSH, so, if the PSH bit is set to 1 then the receiver is requested to push the data without buffering it.

4. **RST:** RST stands for restart, so if its bit value is set to 1 then the connection needs to be restarted.

TCP: Header Format

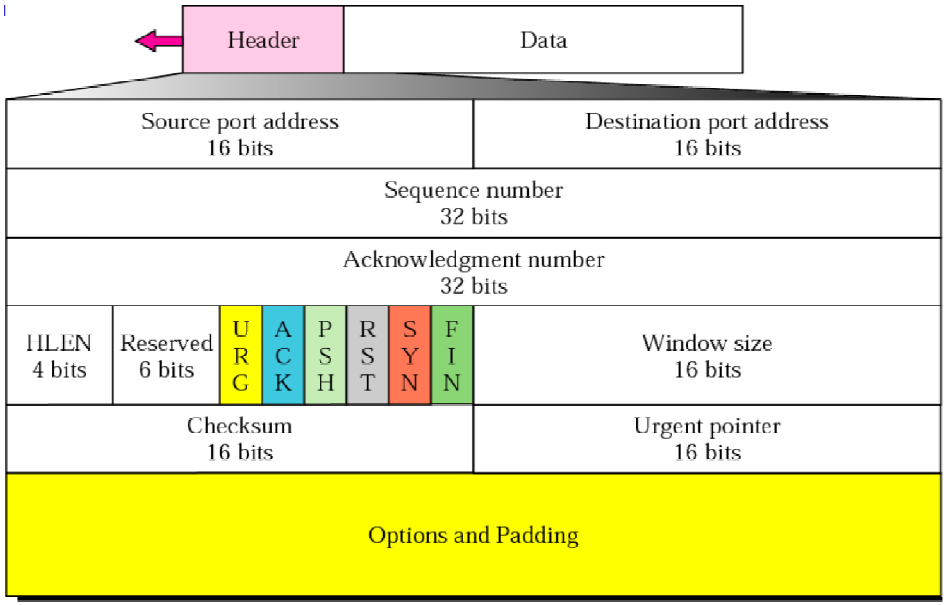


Flags:

5. **SYN:** SYN bit as discussed earlier is used to establish a connection.

6. **FIN:** FIN represents finish, so, if its value is set to 1 then the connection is to be closed.

TCP: Header Format

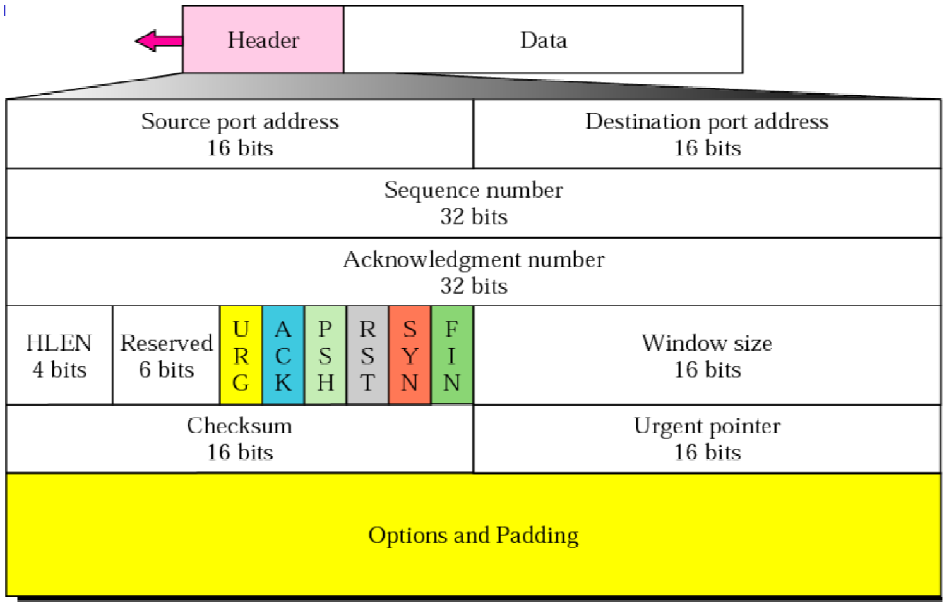


Window Size:

- The window size is a 16-bit field
- It contains the **data size** that the **receiver can accept**.

- It helps in the **flow control mechanism**.
- The window size field is determined by the receiver only.

TCP: Header Format

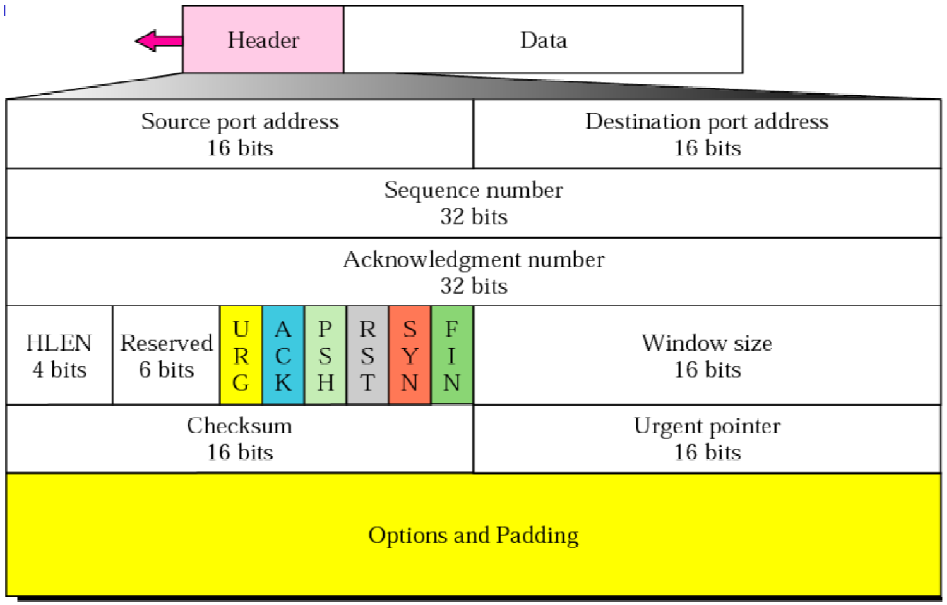


Checksum:

- The checksum is also a **16-bit field**

- It contains the checksum of the header and the data.

TCP: Header Format

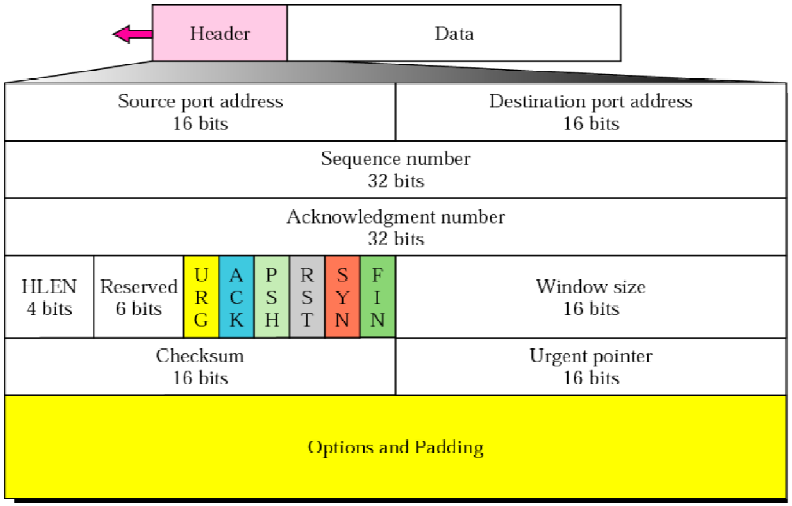


Urgent Pointer:

- If the value of URG flag is set to 1

- Then the urgent Pointer field **points to the urgent data byte**.

TCP: Header Format



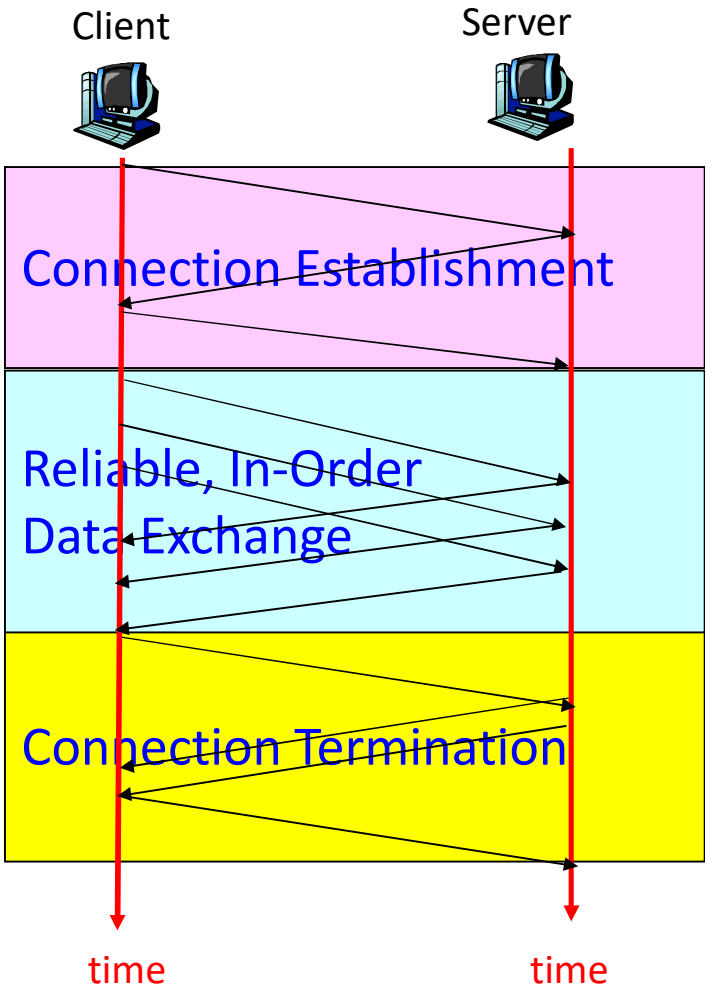
Options and Padding:

- As the name suggests, the option field contains options that are not in the regular header.
- The options field is described as a 40-byte word.
- **Padding is used** in the case when the data in the **option field is less than 40-byte**
- So, padding (adding extra 0 bits) helps to make the option bit reach the 40-byte word boundary.

Typical TCP Transaction

A TCP Transaction consists of **3 Phases**:

- **Connection Establishment**
  - Handshaking between client and server
- **Reliable, In-Order Data Exchange**
  - Recover any lost data
  - through retransmissions and ACKs
- **Connection Termination**
  - Closing the connection



TCP Connection Establishment

- TCP sender, receiver **establish "connection"** before exchanging data segments

- It initialize TCP variables:
  - Seq. Nos, buffers, flow control info (e.g. RcvWindow)

- Client: connection initiator
  - *Socket clientSocket = new Socket("hostname", port#);*

- Server: contacted by client
  - *Socket connectionSocket = welcomeSocket.accept();*

Connection Establishment (contd.)

Three Way Handshake:

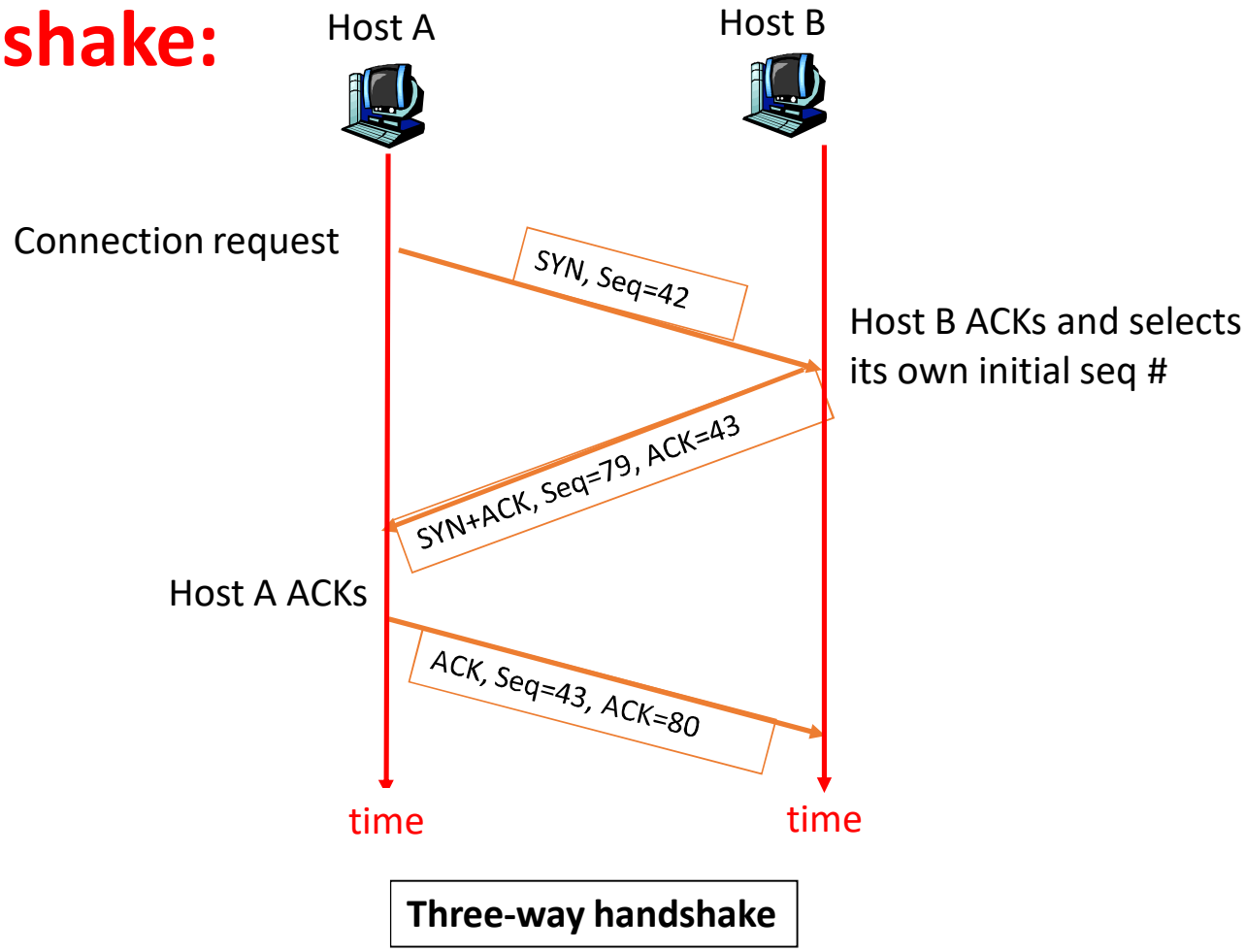
- **Step 1:** client host sends TCP SYN segment to server
  - specifies a random initial seq #
  - no data is sent.

- **Step 2:** server host receives SYN, replies with SYNACK segment
  - server allocates buffers
  - specifies server initial seq. #

- **Step 3:** client receives SYNACK, replies with ACK segment, which may contain data

Connection Establishment (contd.)

Three Way Handshake:



Three-way handshake

Connection Establishment (contd.)

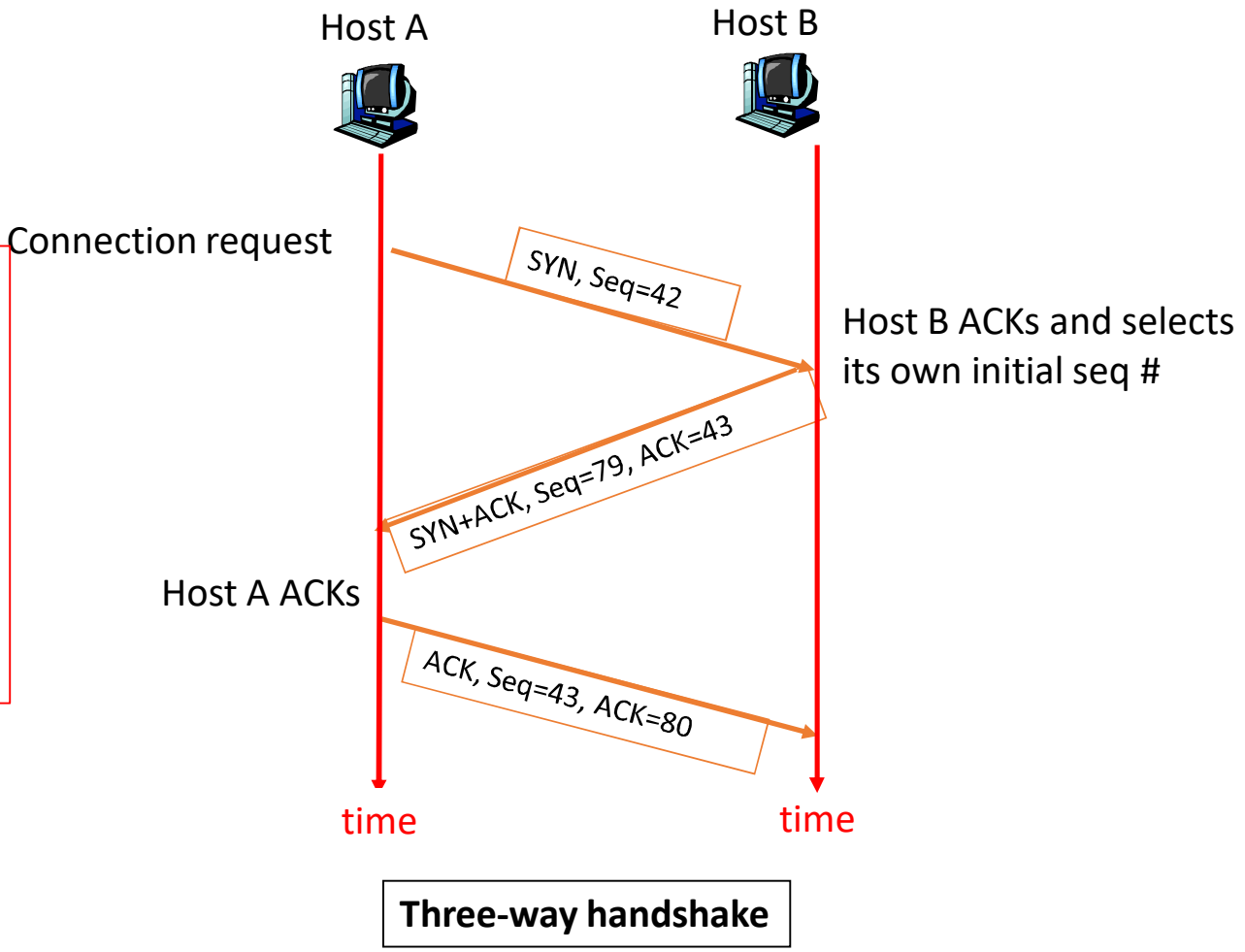
Three Way Handshake:

Seq. #'s:

- byte stream "number" of first byte in segment's data

ACKs:

- seq # of next byte expected from other side
- cumulative ACK



Three-way handshake

TCP Connection Termination

Closing a connection:

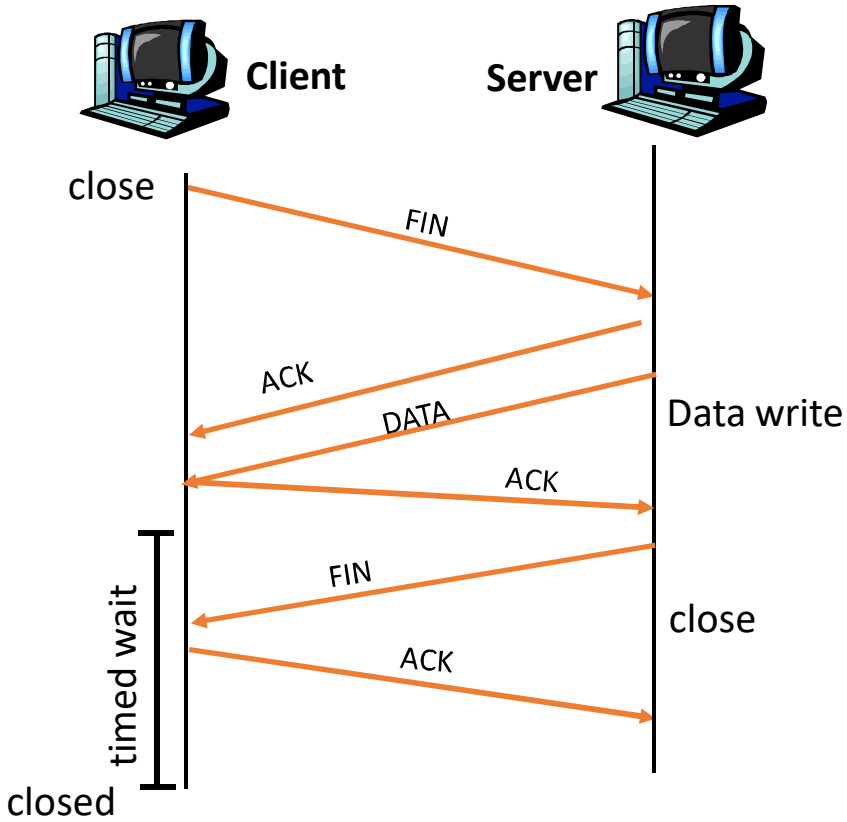
- Client closes socket:
  - *clientSocket.close();*

- **Step 1:** client end system sends **TCP FIN control segment** to server

- **Step 2:** server receives FIN, replies with ACK.
  - Server might send some buffered but not sent data before closing the connection.
  - Server then sends FIN and moves to Closing state.

TCP Connection Termination

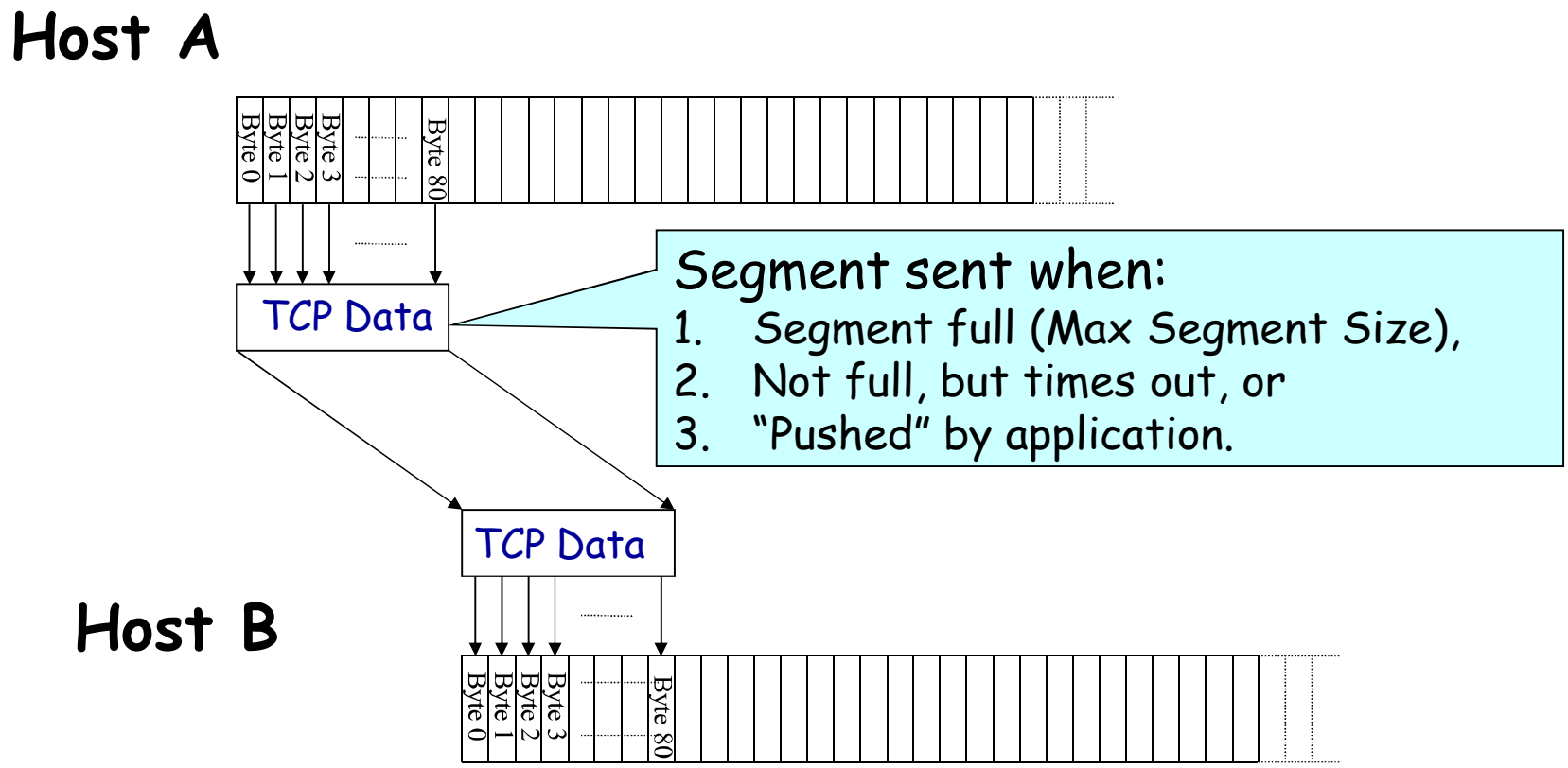
Closing a connection:



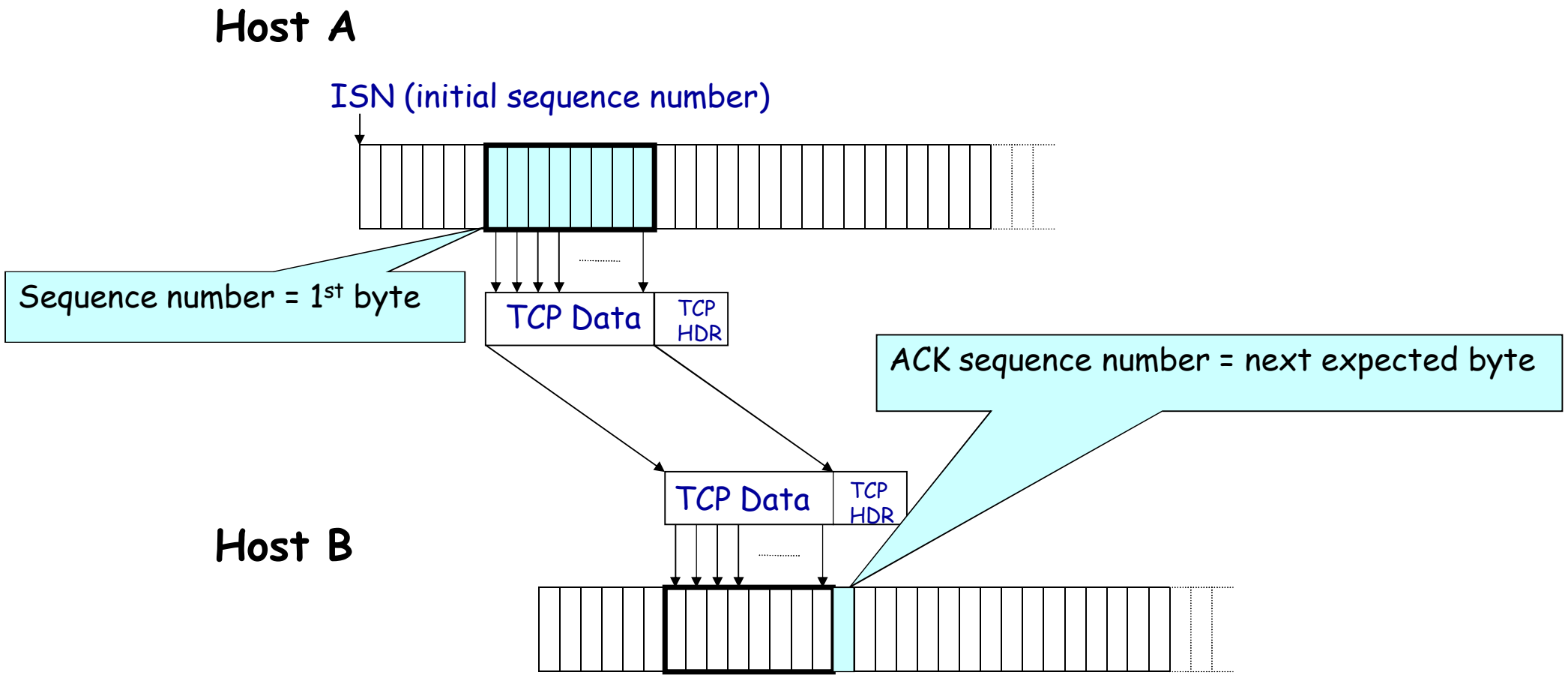
Byte stream to TCP Segment

- Accepts byte stream from processes – breaks them into 64KB or less size chunks
- Each chunks mapped to separate datagram (which is further mapped to IP payload)
- On reception of IP packet, the payload is given to TCP entity.
- Creates the byte stream back.

Emulated Using TCP "Segments"



Sequence Numbers in TCP



TCP Segment Format

IP Packet:

- No bigger than Maximum Transmission Unit (MTU)
- E.g., up to 1500 bytes on an Ethernet

TCP Packet:

- IP packet with a TCP header and data inside
- TCP header is typically 20 bytes long

TCP Segment:

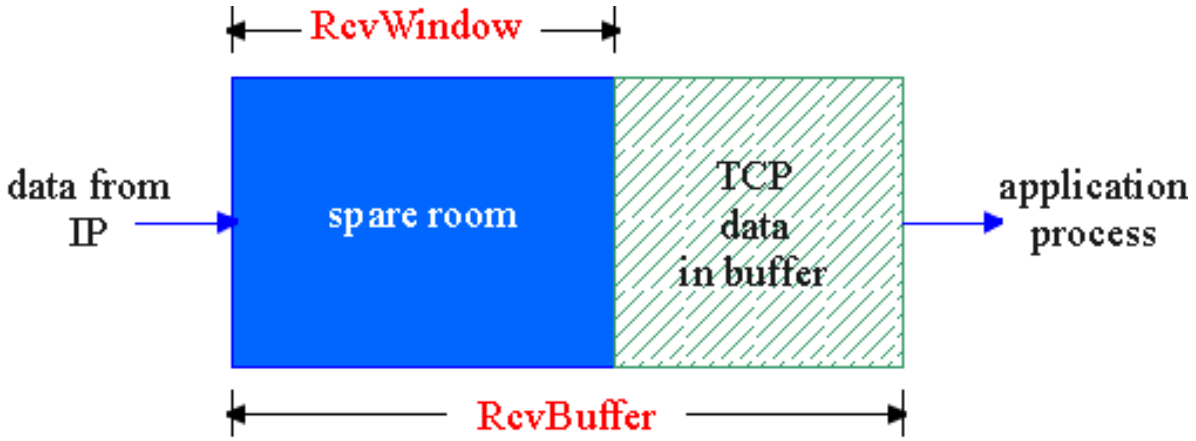
- No more than Maximum Segment Size (MSS) bytes
- E.g., up to 1460 consecutive bytes from the stream

TCP Flow Control

Flow Control

Sender won't overflow receiver's buffer by transmitting too much and too fast

- **Speed-matching service:** matching the send rate to the receiving app's drain rate

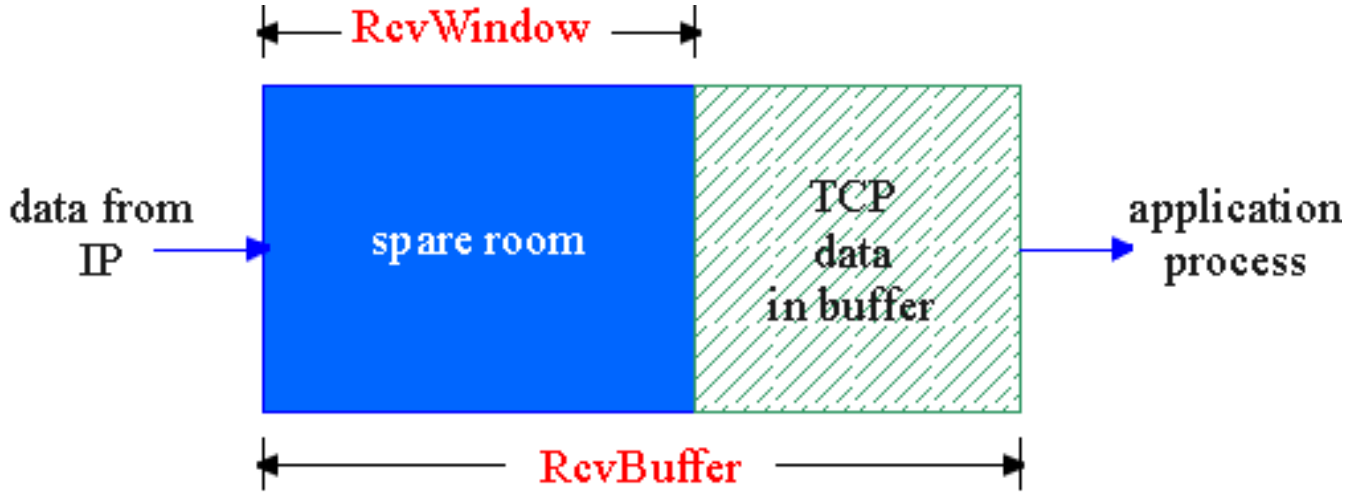


TCP Flow Control: How It Works

(Suppose TCP receiver discards out-of-order segments)

- **Spare room in buffer**

  RcvWindow = RcvBuffer - [LastByteRcvd - LastByteRead]

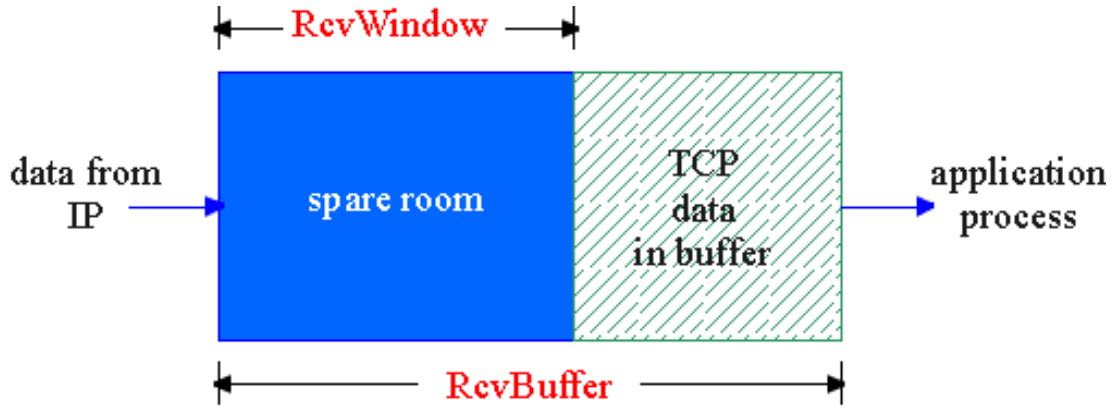


TCP Flow Control: How It Works

- Then Receiver advertises spare room by including value of *RcvWindow* in segments

- Sender limits **unACKed data to RcvWindow**

- This guarantees receive buffer doesn't overflow



TCP Flow Control: How It Works

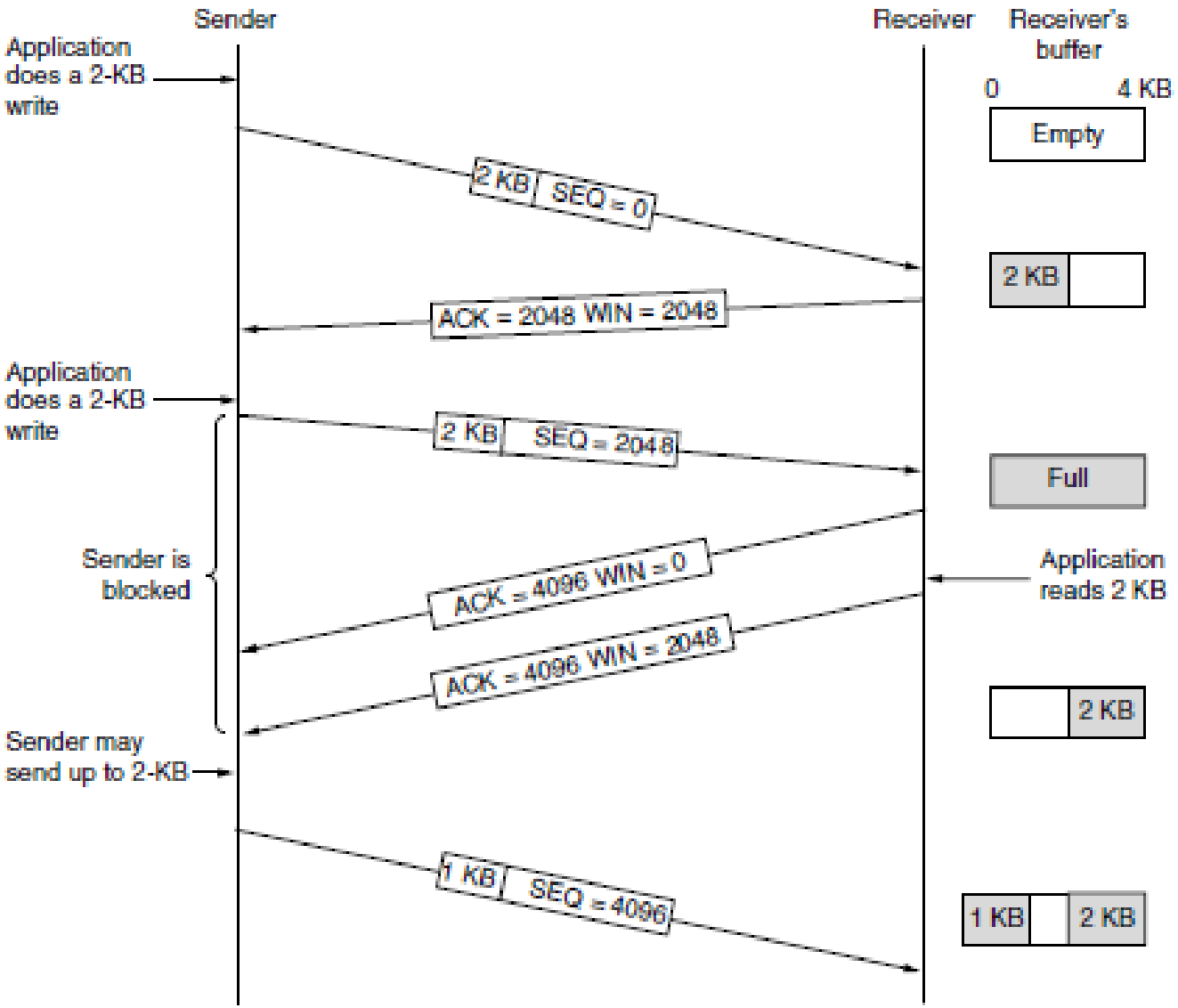
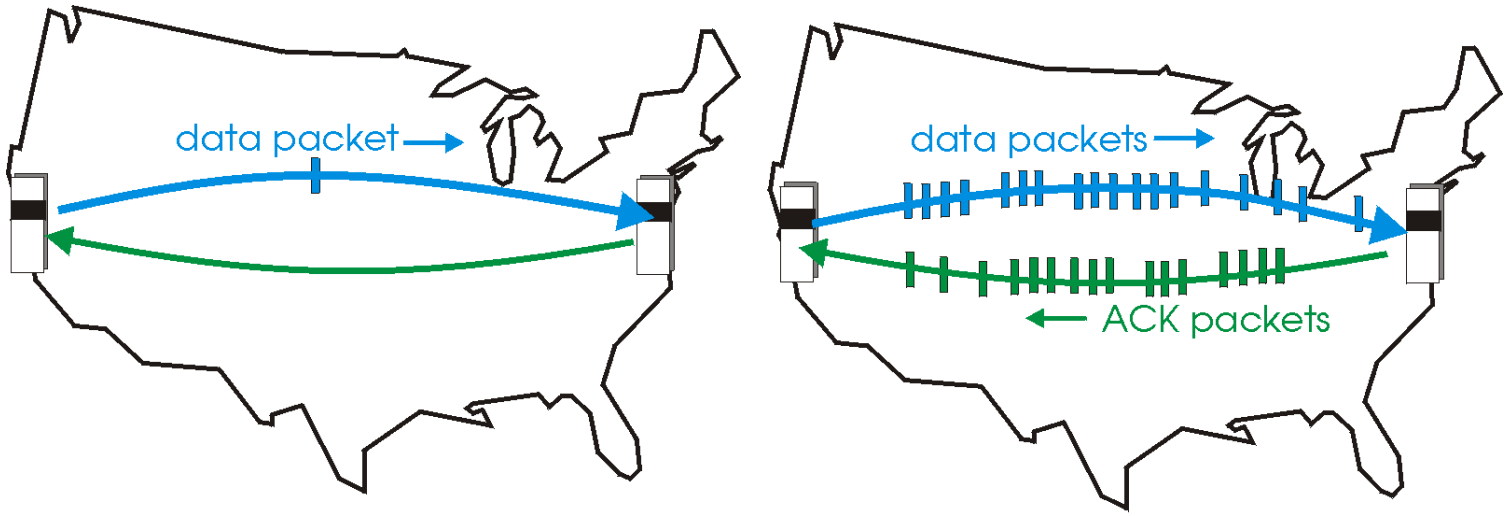


Figure 6-40. Window management in TCP.

Pipelined Protocols

- **Pipelining:** sender allows multiple, “in-flight”, yet-to-be-acknowledged packets
  - range of sequence numbers must be increased
  - buffering at sender and/or receiver

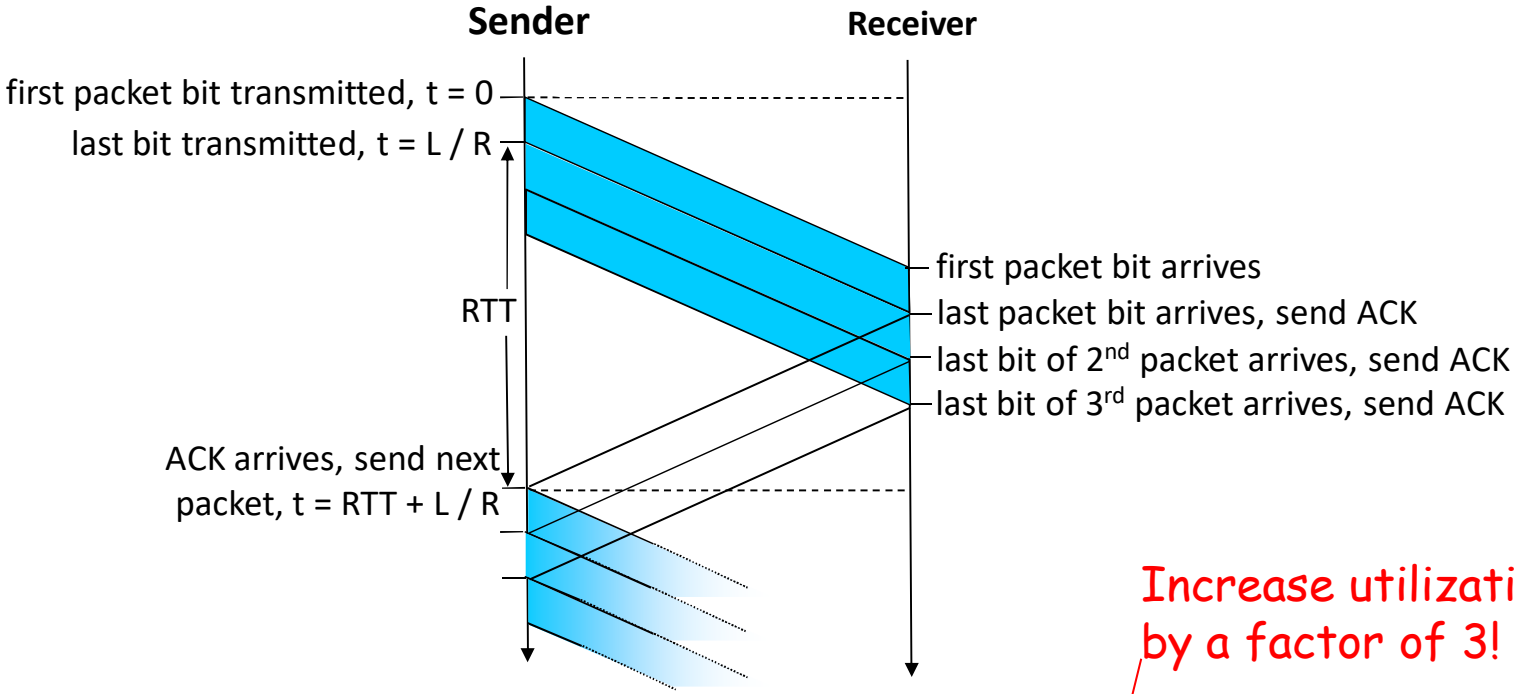


(a) a stop-and-wait protocol in operation

(b) a pipelined protocol in operation

- Two generic forms of pipelined protocols: Go-Back-N, Selective Repeat

Pipelining -> Increased Utilization



Increase utilization by a factor of 3!

$$U_{sender} = \frac{3 * L / R}{RTT + L / R} = \frac{.024}{30.008} = 0.0008$$

We assume a 1 Gbps link, 15 ms prop. delay, 8000 bit packet:

Pipelined Protocols

Go-back-N: big picture:

- Sender can have up to N unacked packets in pipeline
- Rcvr only sends cumulative ACKs
  - Doesn't ack packet if there's a gap
- Sender has timer for oldest unacked packet
  - If timer expires, retransmit all unacked packets

Selective Repeat: big piccture

- Sender can have up to N unacked packets in pipeline
- Rcvr acks individual packets
- Sender maintains timer for each unacked packet
  - When timer expires, retransmit only unack packet

Congestion Control

Congestion:

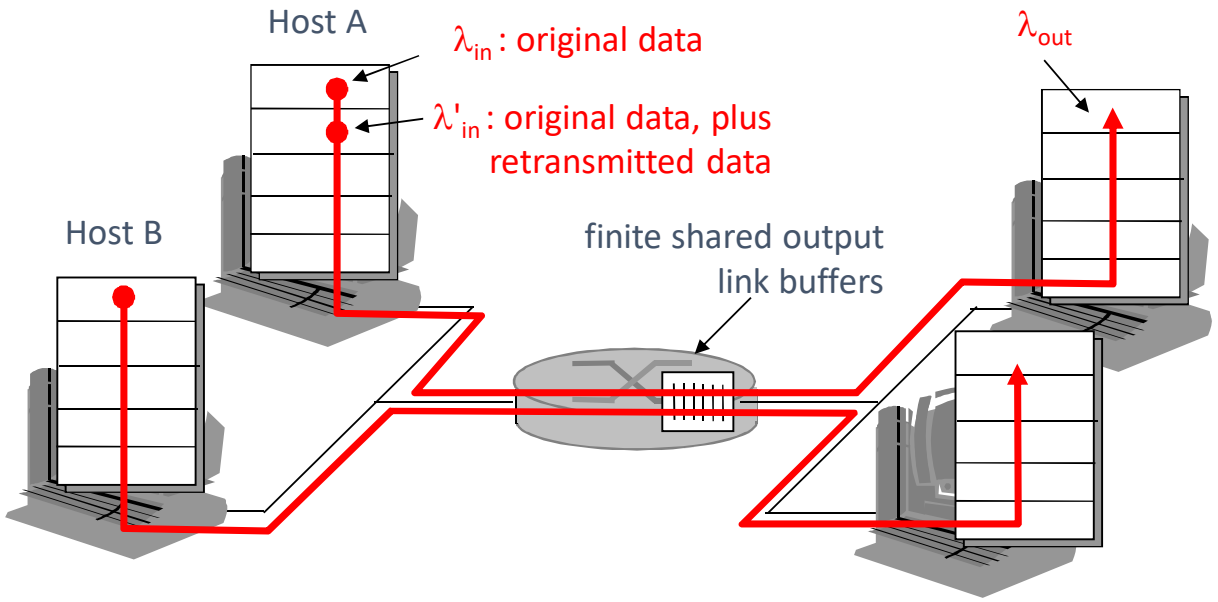
- informally: “too many sources sending too much data too fast for network to handle”.

- It is different from flow control!!

- Its Manifestations:
  - lost packets (buffer overflow at routers)
  - long delays (queueing in router buffers)

Causes/costs of congestion

- Every router has ***finite*** buffers
- The sender ***retransmission*** of lost packet

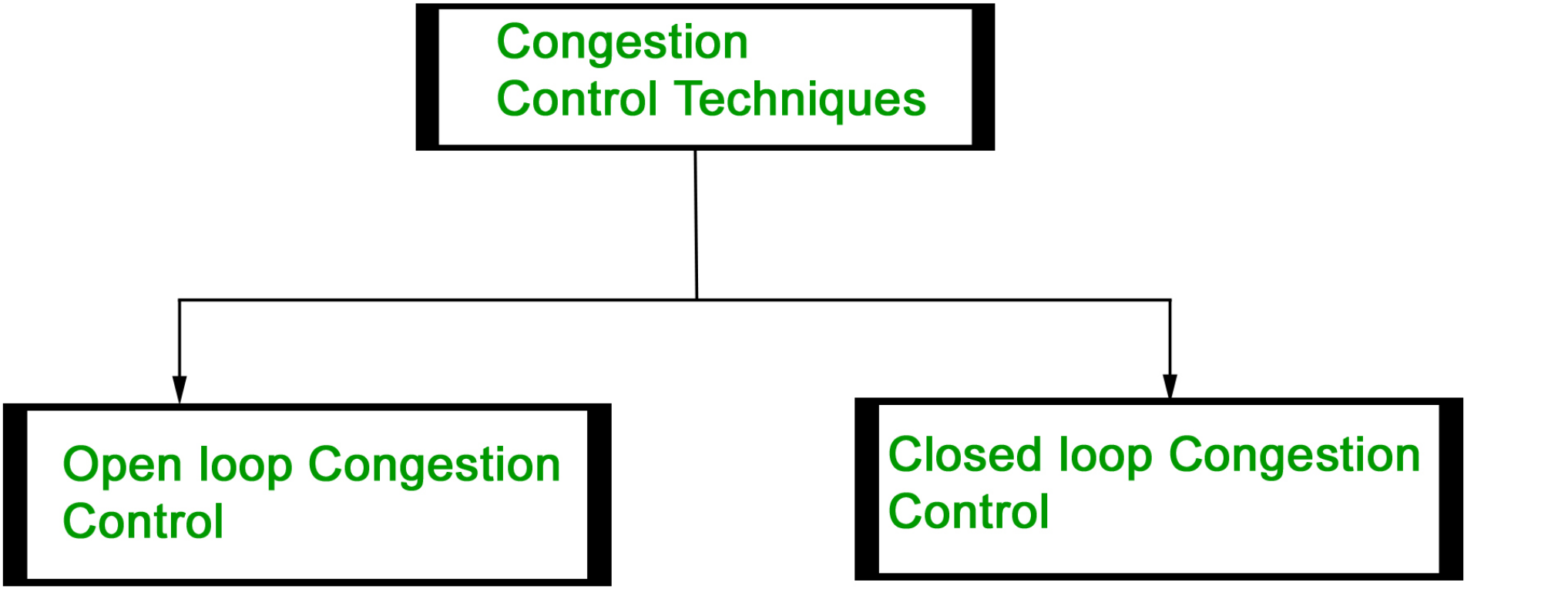


Causes and costs of congestion

Other factors could be:

- Excessive consumption of bandwidth
- Improper subnet management
- Multicasting
- Outdated hardware
- Aggressive routing protocol

Approaches towards Congestion Control



Open Loop Congestion Control

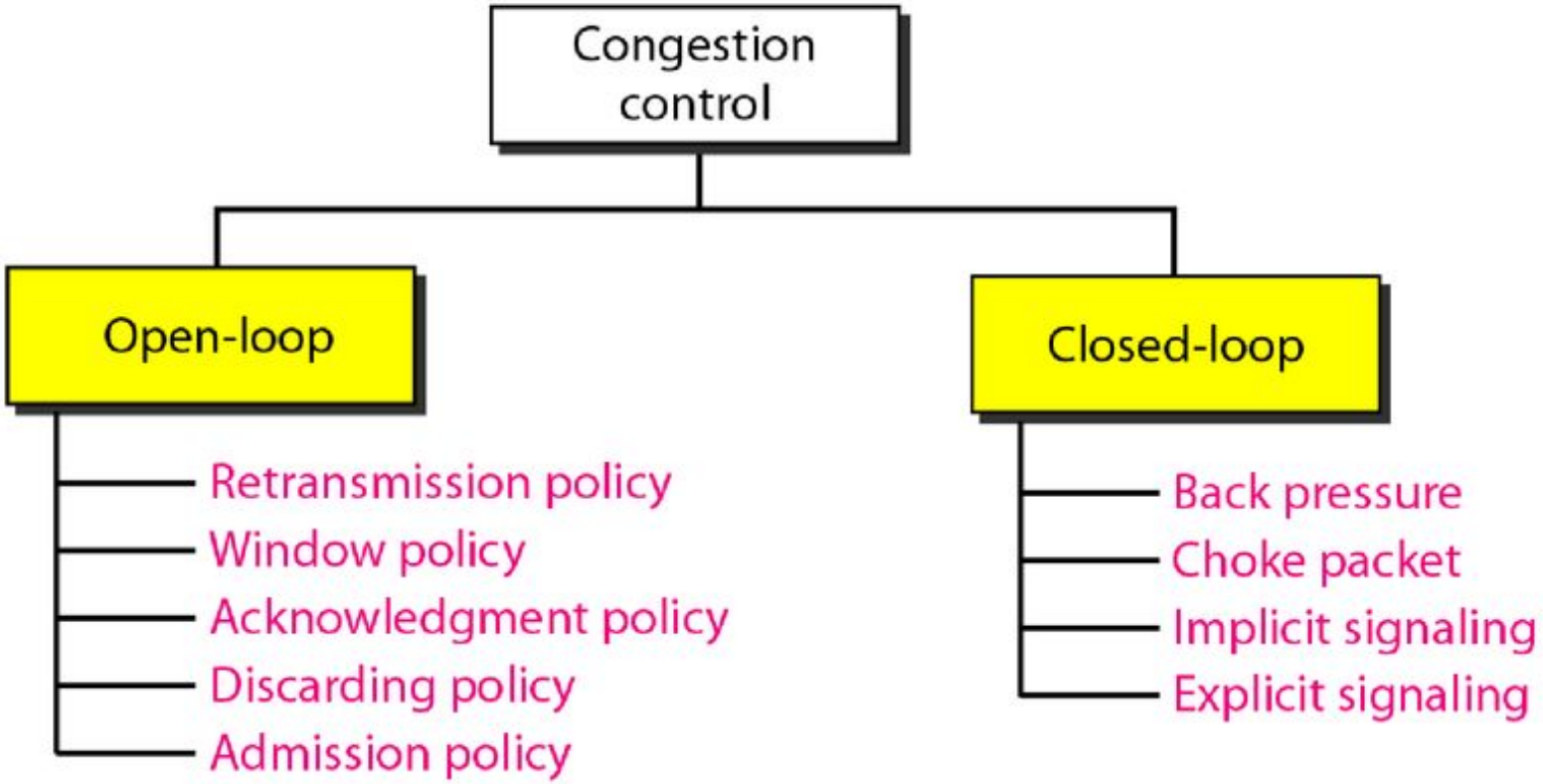
- Open loop congestion control policies are applied to **prevent congestion before it happens**.
- Also called as **Congestion-Avoidance**
- The congestion control is handled either by the source or the destination.
- This is implemented in the **TCP Layer**

Closed Loop Congestion Control

- Closed loop congestion control techniques are **used to treat or alleviate congestion after it happens**.

- This is implemented in the **Network Layer**

Congestion Control



End-end vs Network-assisted

End-end congestion control:

- no explicit feedback from network
- congestion inferred from end-system observed packet loss, packet delay
- approach taken by TCP

Network-assisted congestion control:

- routers provide feedback to end systems (i.e., to senders)
  - single bit indicating congestion (ECN)
  - explicit rate sender should send at

TCP Congestion Control

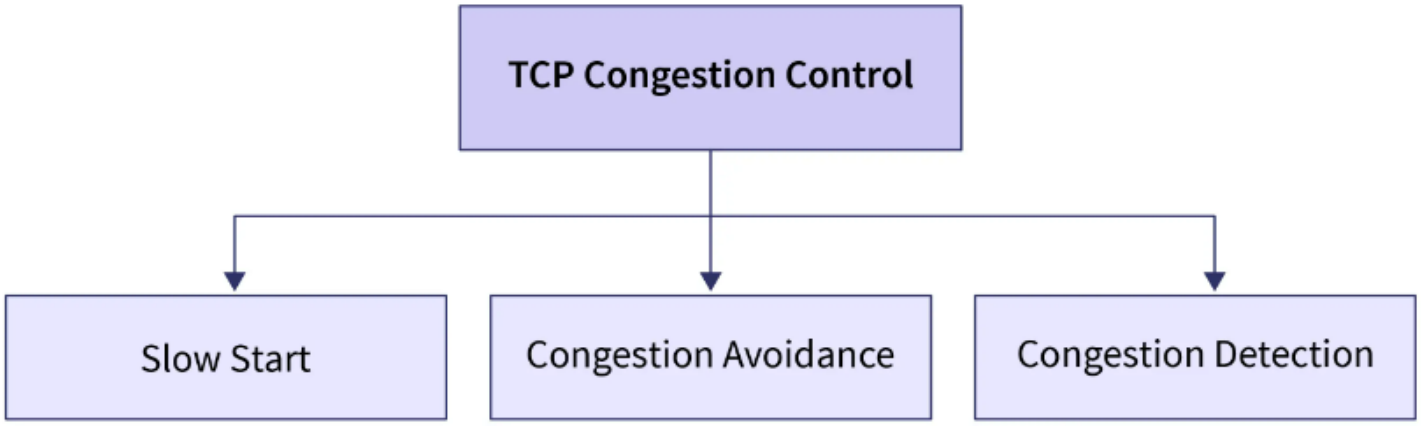
- TCP congestion control is a method used by the TCP protocol
- **Why:** To manage data flow over a network and prevent/avoid congestion.
- **How:** TCP uses a congestion window and congestion policy that avoids congestion.

TCP Congestion Control

- Previously, we assumed that only the receiver could dictate the sender's window size.
- We ignored another entity here, the network.
- What if the network cannot deliver the data as fast as it is created by the sender?
- In other words, the **network is a second entity** that also determines the size of the sender's window.

Congestion Policy in TCP

- Congestion in TCP is handled by using these three phases:
  - **Slow Start Phase:**
  - **Congestion Avoidance Phase:**
  - **Congestion Detection Phase:**



TCP Congestion Control

Different Variables used in TCP Congestion Control

Congestion Window (Cwnd)

- It limits the amount of data to be sent by the sender into the network even before receiving the acknowledgment.

Slow start Threshold (ssThresh)

- It is a value, which is used in the slow start phase

Maximum Receiver Capacity / Receiver Window Size

- The sender should not send data greater than that of the size of receiver window.

Slow Start Phase

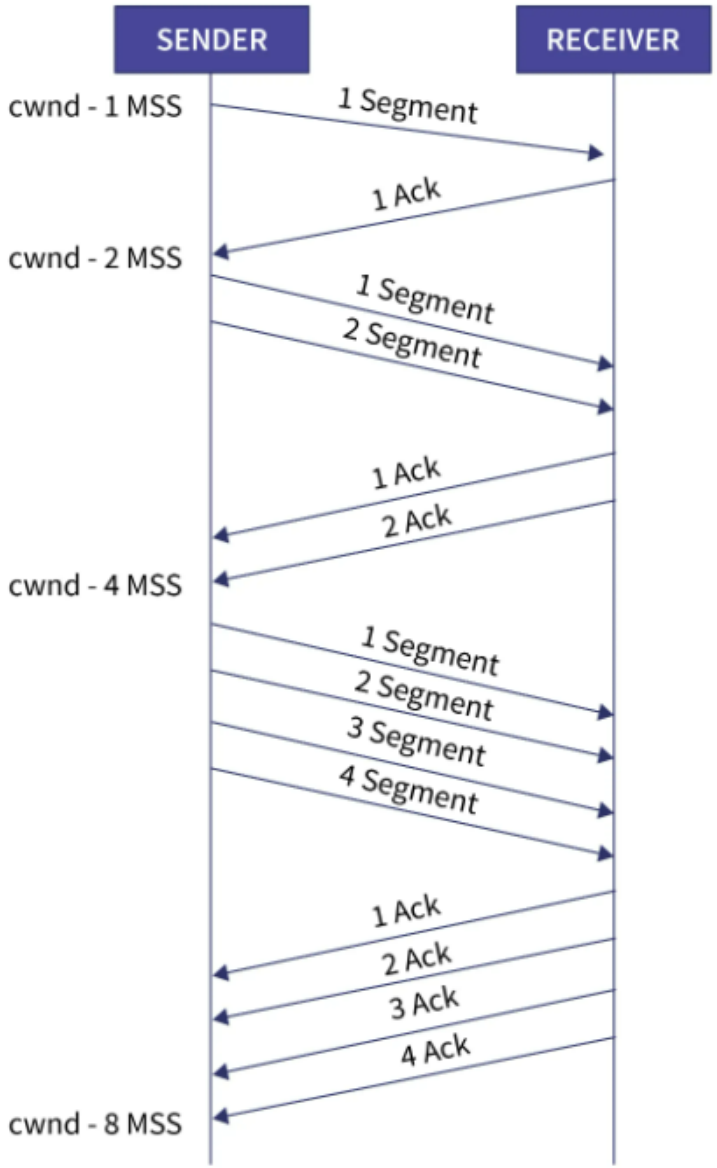
- Initially, sender sets congestion window size = maximum segment size (1 MSS)

CWnd = 1 MSS

- The sender increases the size of the congestion window (Cwnd) after receiving the ACK (acknowledgment).

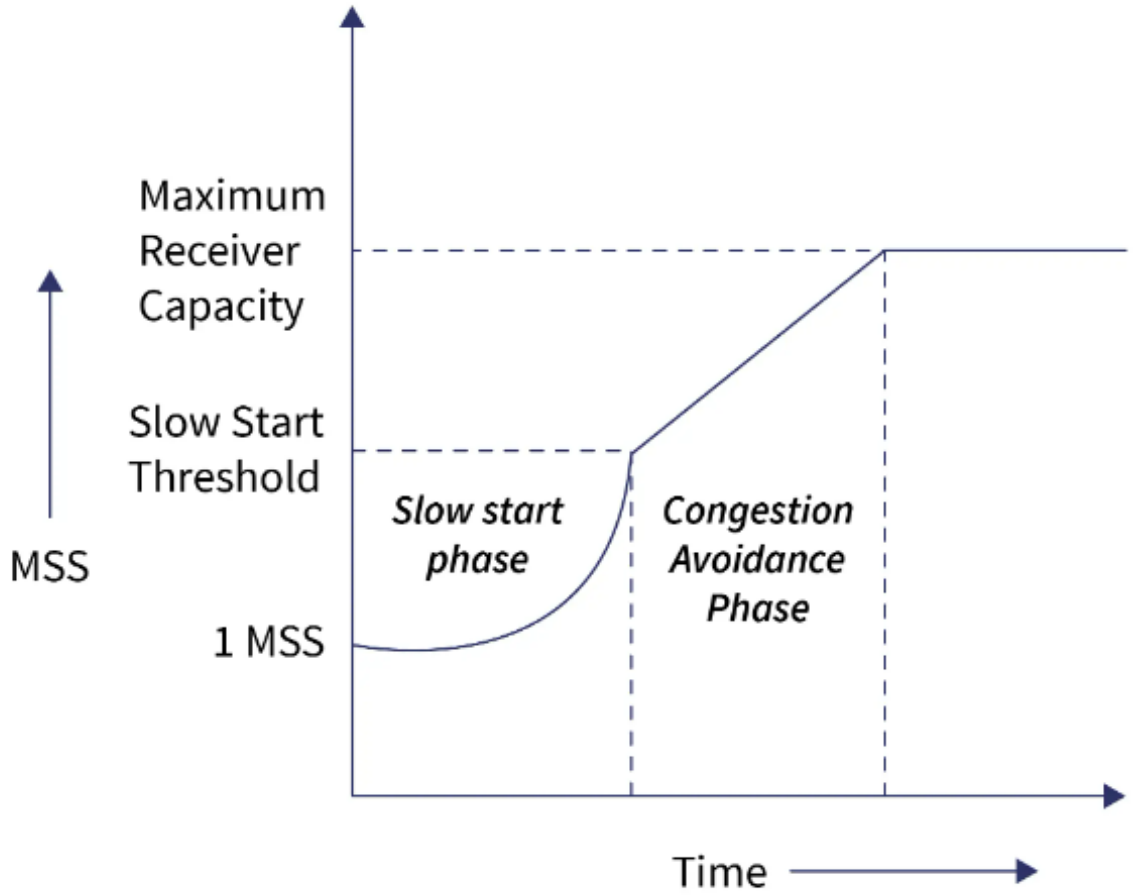
- The size of the **congestion window increases exponentially** in this phase.

Slow Start Phase



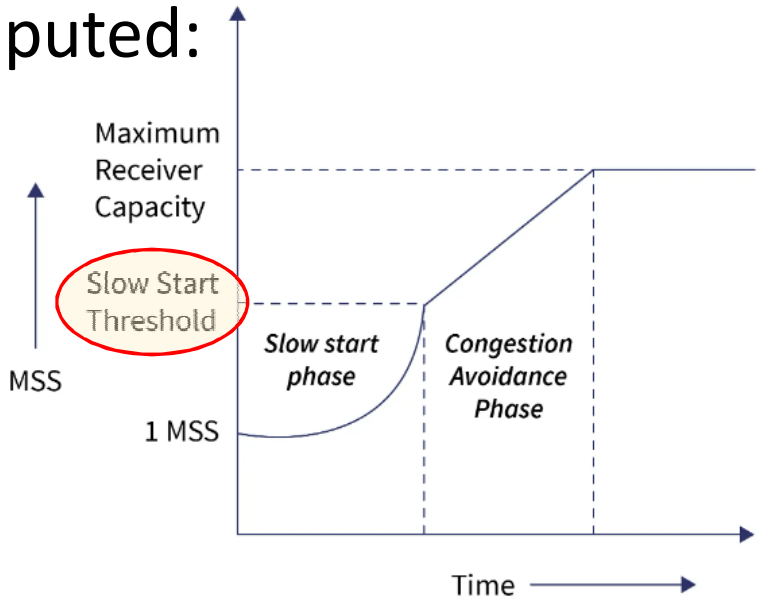
Slow Start Phase

- This phase continues until **Cwnd reaches its slow start threshold.**



Slow Start Phase

- How this **slow start threshold** value is computed:



The **formula** for determining the threshold is given:

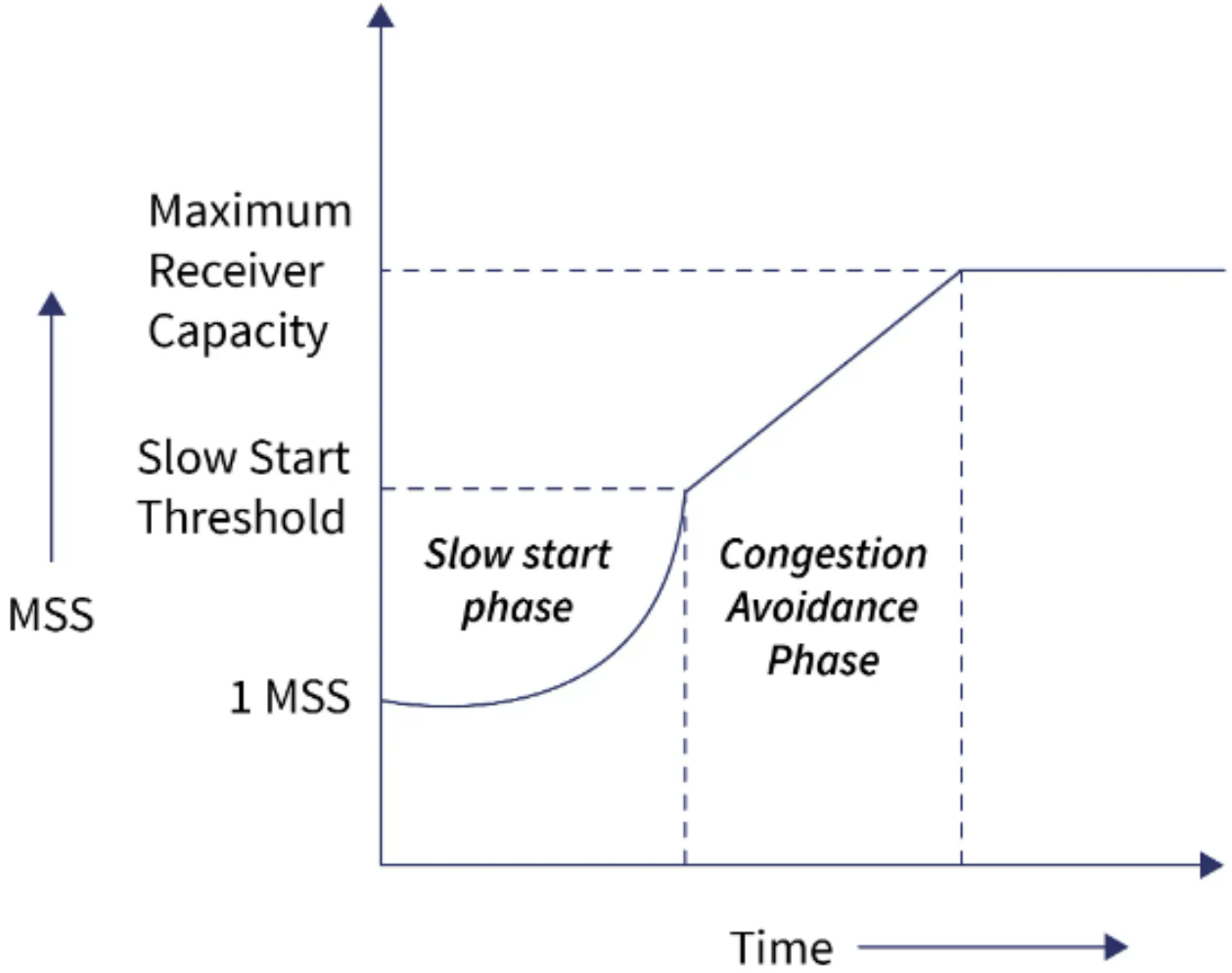
```
Threshold = Maximum number of TCP segments that the receiver window can accommodate / 2
= (Receiver window size / Maximum Segment Size) / 2
```

Congestion Avoidance Phase

After the SS_threshold is reached, it enters Avoidance phase:

- The size of the congestion window is increased by the sender linearly
- **Why:** in order to avoid congestion.
- Each time an acknowledgment is received, the sender increments the size of the congestion window by 1.

Congestion Avoidance Phase



Congestion Avoidance Phase

- This phase continues until the size of the CWnd window becomes equal to that of the receiver window size.

- Then, it is **set to constant** to avoid overwhelming receiver.

Congestion Detection Phase

- However, during this the sender can identify the **segment loss**

- The sender can act depending on the **type of loss detected**.
  - Case-01: Detection On Time Out
  - Case-02: Detection Of Receiving 3 Duplicate Acknowledgements

Congestion Detection Phase

Case-01: Detection On Time Out

- In this, the **time-out timer expires** even before receiving acknowledgment for a segment.

- It suggests a **stronger possibility of congestion** in a network
  - there are chances that a segment has been dropped in the network

Congestion Detection Phase

Reaction in response to Detection on time out:

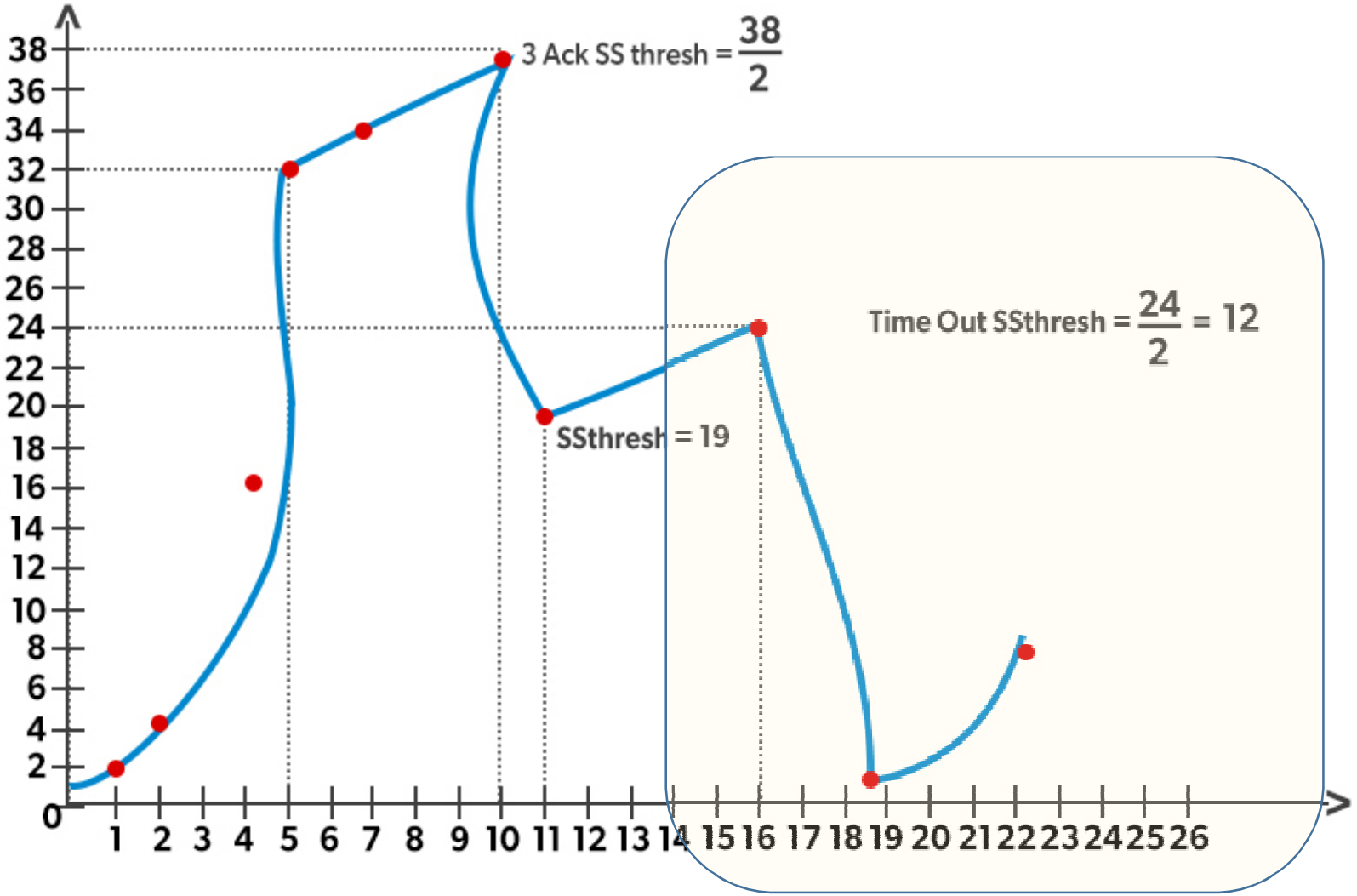
- Setting the threshold to start at half of the current size of the window

  i.e., **SS_threshold = CWnd/2**

- Decreasing the size of the congestion window to 1 MSS

- Slow start phase is resumed

Congestion Policy in TCP



Congestion Detection Phase

Case-02: Detection Of Receiving 3 Duplicate Acknowledgements

- This case suggests the **weaker possibility of congestion** in the network.

- The chances are that fewer segments have dropped while the one sent later might have reached.

Congestion Detection Phase

Reaction on receiving 3 duplicate acknowledgments:

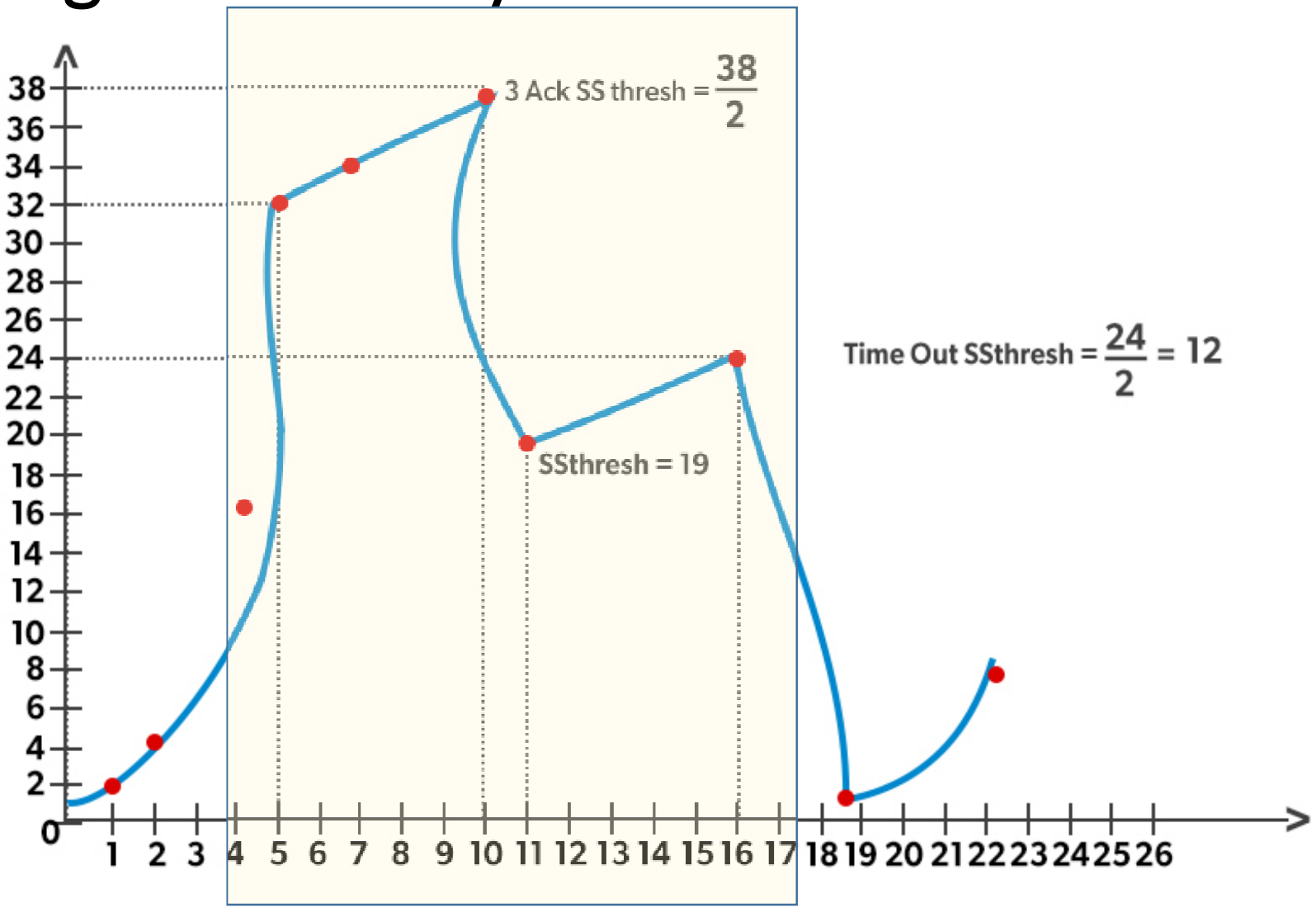
- Setting the threshold to start at half of the current size of the window

  i.e., **SS_threshold = CWnd/2**

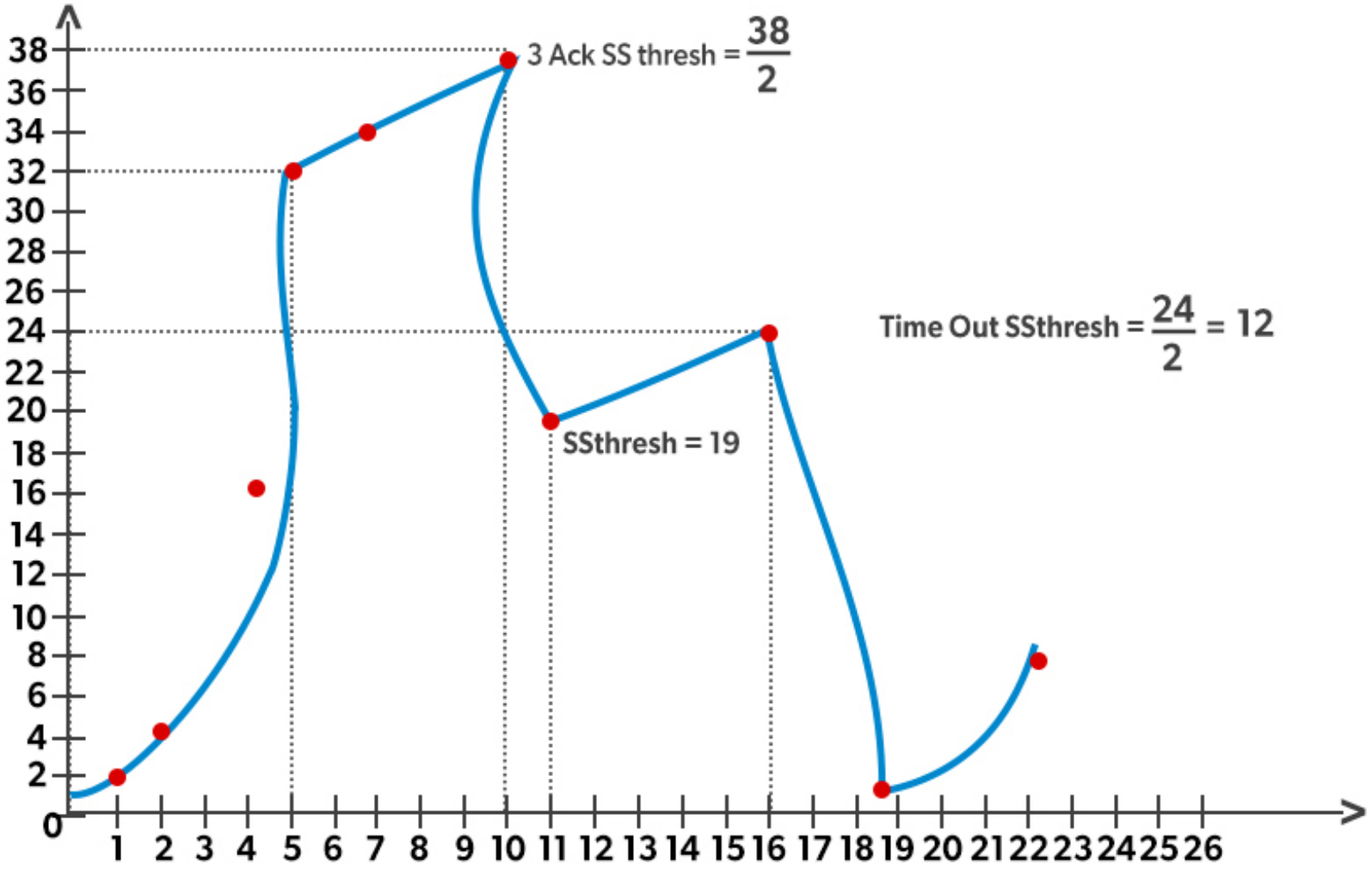
- Decreasing the size of the congestion window to that of the slow start threshold

- The congestion avoidance phase is resumed

Congestion Policy in TCP



Congestion Policy in TCP



Congestion Policy in TCP

- Incrementing the Cwnd by 1 MSS (for each ACK received) and reducing SS_Threshold to half of the current size of the window (on segment loss) is called as:

TCP's Additive Increase Multiplicative Decrease (AIMD)

thank you