Unit No:3

Transport Layer

Marwadi University

Department of Computer Engineering

Computer Networks (3150710)

Dr. Sushil Kumar Singh
Associate Professor

Syllabus

- 3.1 Transport-layer services
- 3.2 Multiplexing and demultiplexing
- 3.3 Connectionless transport: UDP
- 3.4 Principles of reliable data transfer
- 3.5 Connection-oriented transport: TCP
  - segment structure
  - reliable data transfer
  - flow control
  - connection management
- 3.6 Principles of congestion control
- 3.7 TCP congestion control

Our goals:

- understand principles behind transport layer services:
  - multiplexing/demultiplexing
  - reliable data transfer
  - flow control
  - congestion control
- learn about transport layer protocols in the Internet:
  - UDP: connectionless transport
  - TCP: connection-oriented transport
  - TCP congestion control

Transport layer (TCP or UDP)

Transport Layer

- The Transport layer is responsible for the delivery of a message from one process to another.

Process to process means:

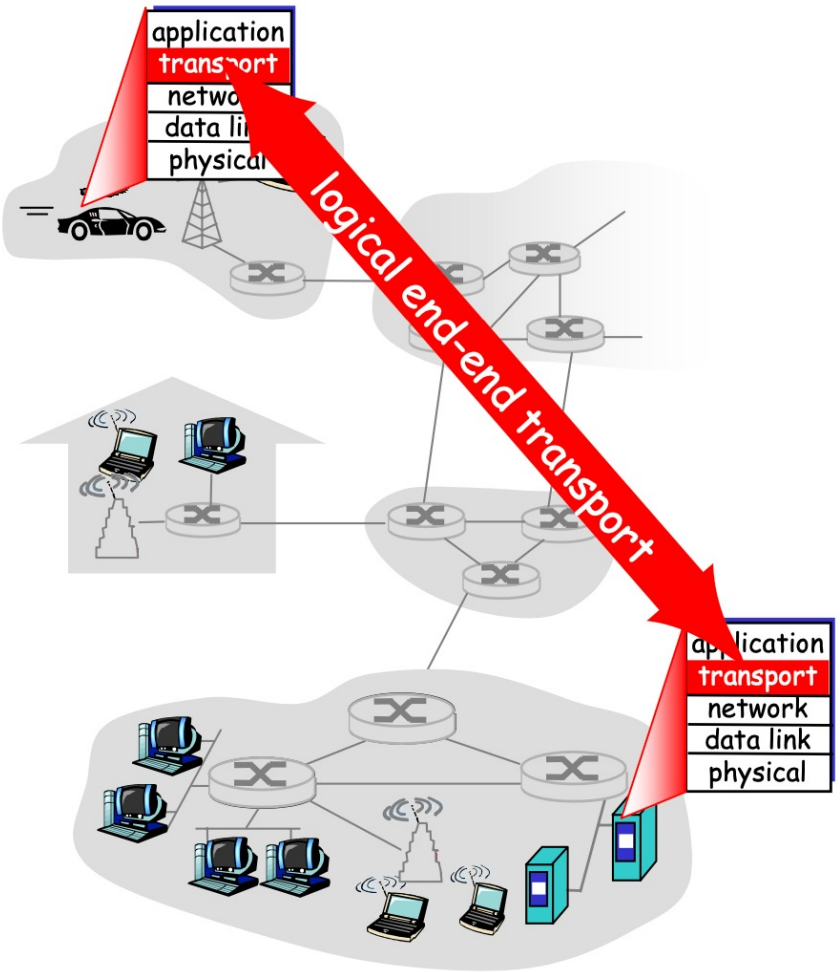
It is not just a source to destination delivery i.e from one computer to another.

It is a delivery from a specific process on one computer to a specific process on the other.

Transport layer (Services and Protocols)

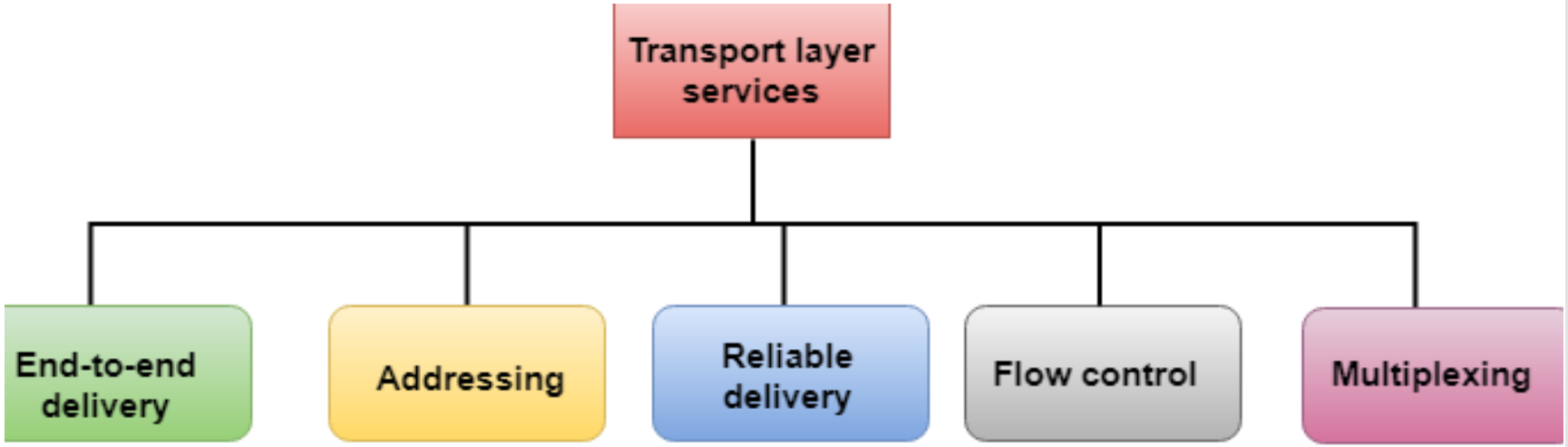
Transport services and protocols

- provide *logical communication* between app processes running on different hosts
- transport protocols run in end systems
  - send side: breaks app messages into segments, passes to network layer
  - rcv side: reassembles segments into messages, passes to app layer
- more than one transport protocol available to apps
  - Internet: TCP and UDP



Transport layer services

- Each of the applications in the application layer has the ability to send a message by using TCP or UDP.
- The application communicates by using either of these two protocols.
- Both TCP and UDP will then communicate with the internet protocol in the internet layer.
- The applications can read and write to the transport layer. Therefore, we can say that communication is a two-way process. (Process to Process Delivery)



Transport layer (Tr. and N/W Layer)

Transport vs. network layer

- *network layer:* logical communication between hosts
- *transport layer:* logical communication between processes
  - relies on, enhances, network layer services

Household analogy:

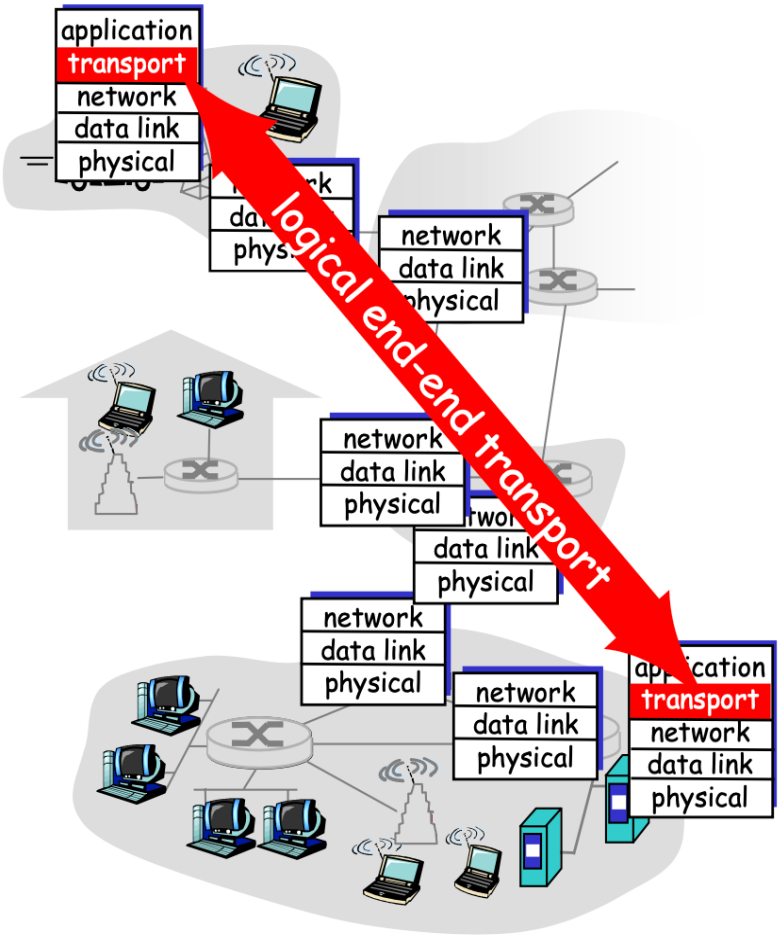
12 kids sending letters to 12 kids

- processes = kids
- app messages = letters in envelopes
- hosts = houses
- transport protocol = Ann and Bill
- network-layer protocol = postal service

Transport layer
(Protocols)

Internet transport-layer protocols

- reliable, in-order delivery (TCP)
  - congestion control
  - flow control
  - connection setup
- unreliable, unordered delivery: UDP
  - no-frills extension of "best-effort" IP
- services not available:
  - delay guarantees
  - bandwidth guarantees



Transport layer (Protocols)

Note:

The transport layer is responsible for process-to-process delivery.

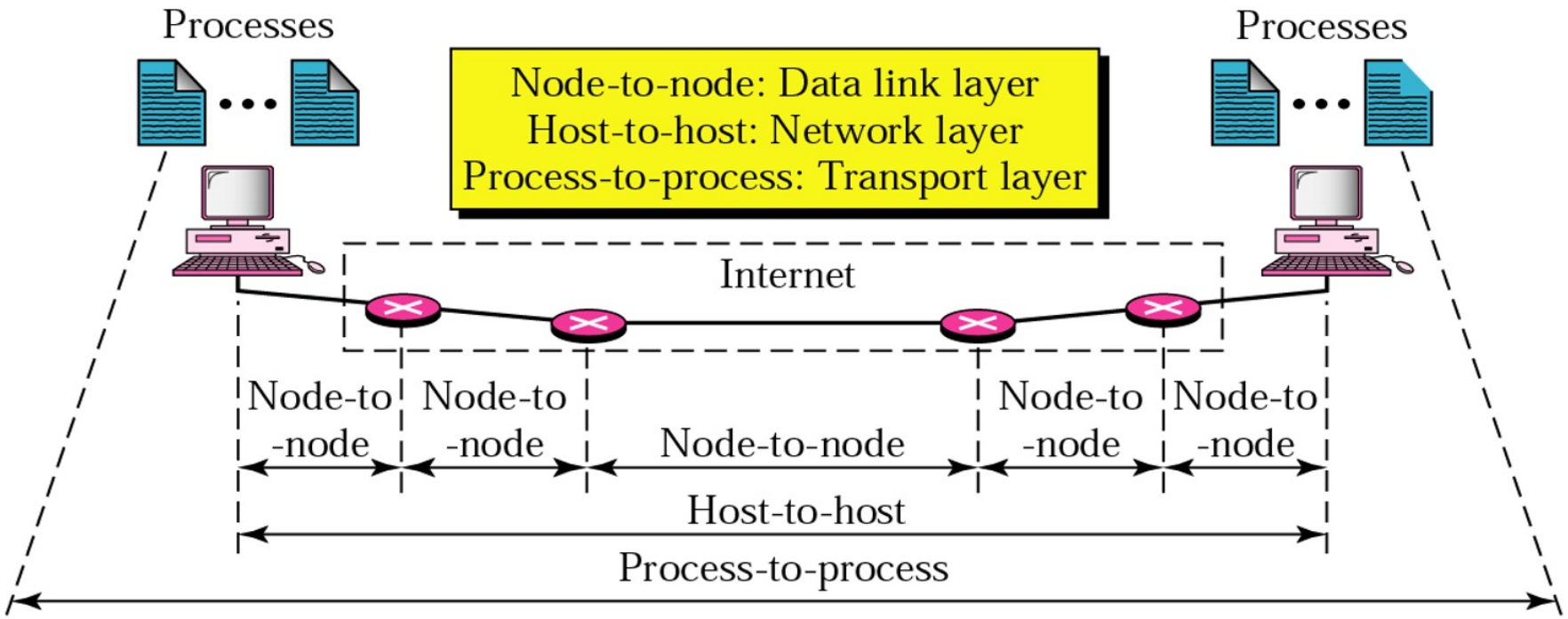
Transport layer protocol

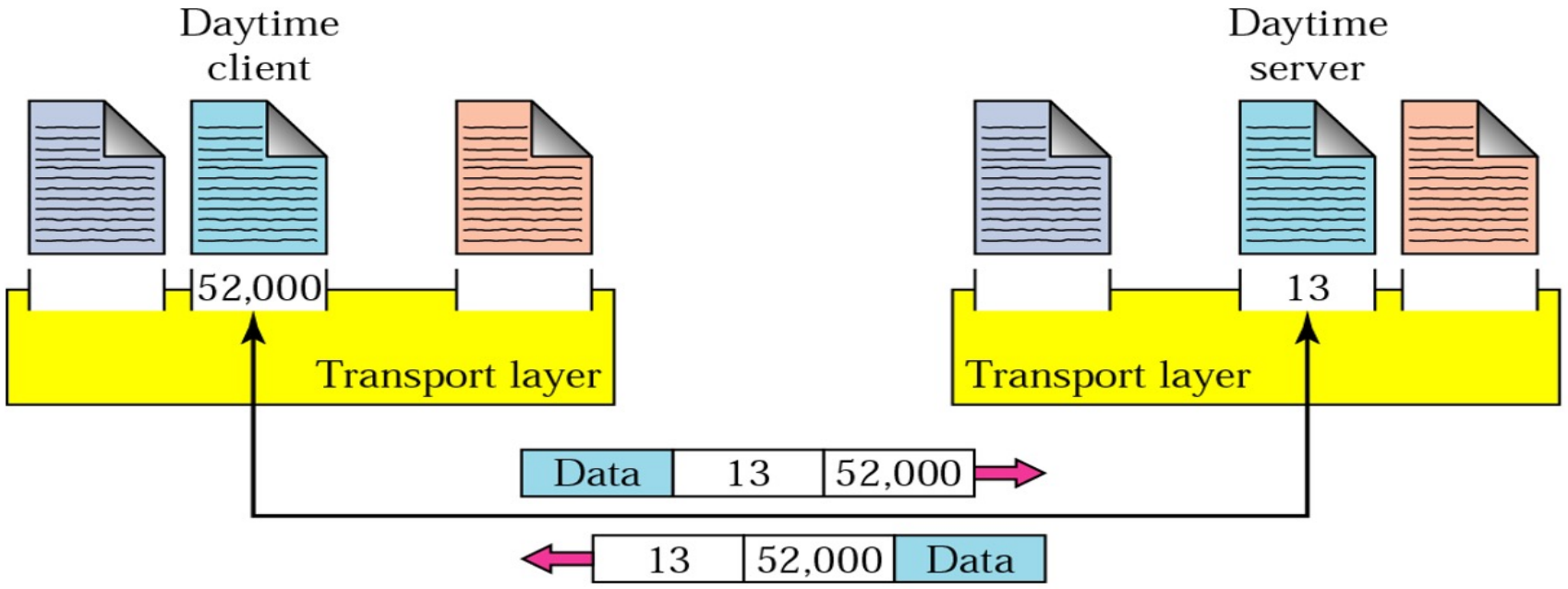
- A transport layer protocol can be either connectionless or connection-oriented.

- UDP-Connectionless protocol-Simple
- TCP-Connection oriented protocol-Complex
- SCTP-Connection oriented protocol-Designed for Multimedia application

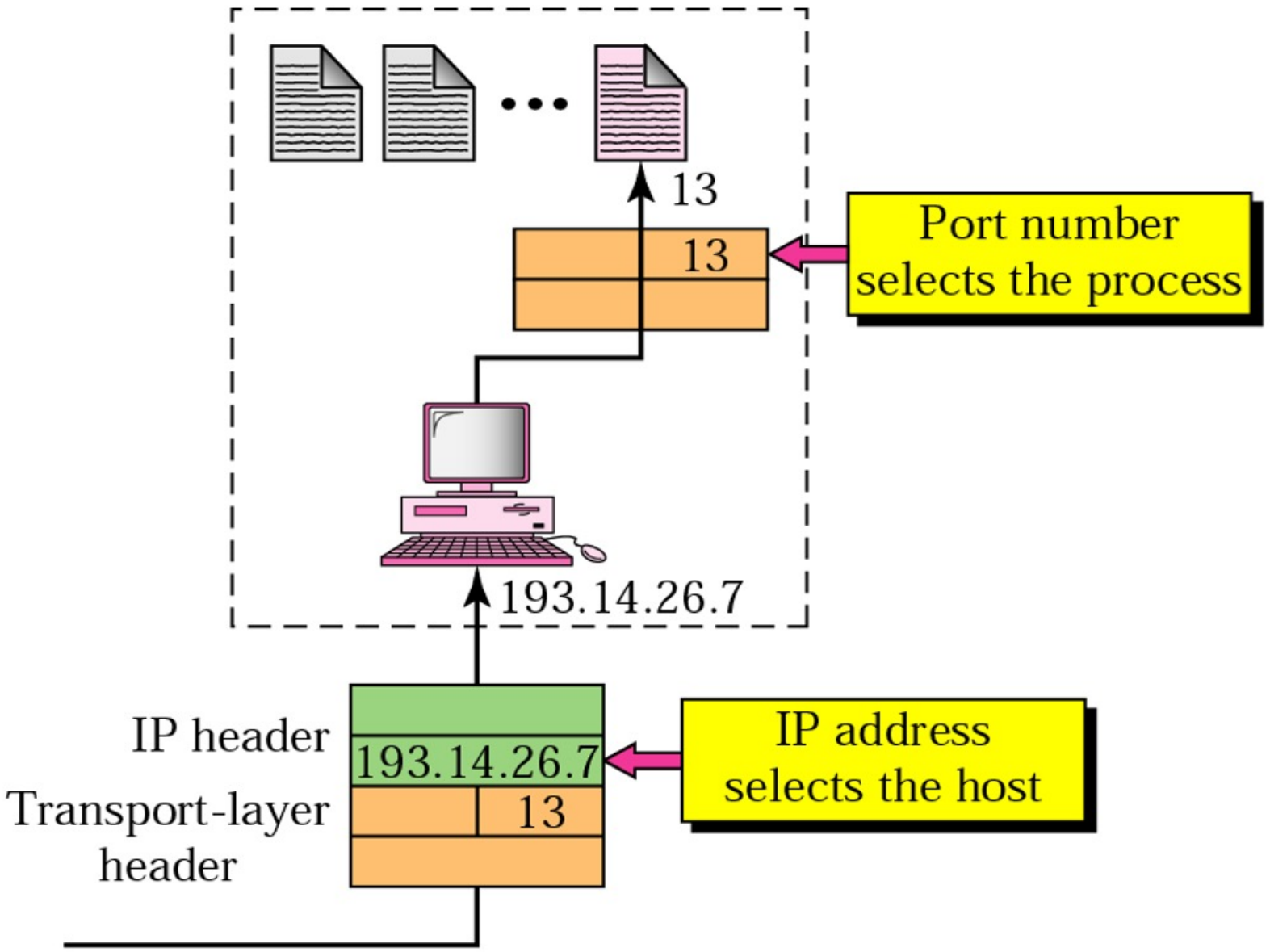
- (Stream control transport protocol)

Transport layer (Types of Delivery & Port Number)

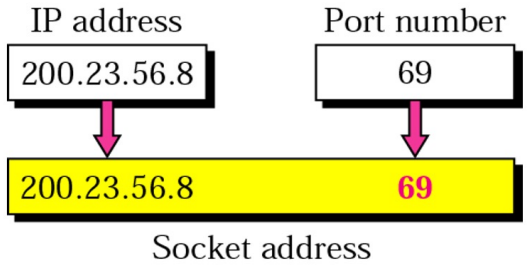


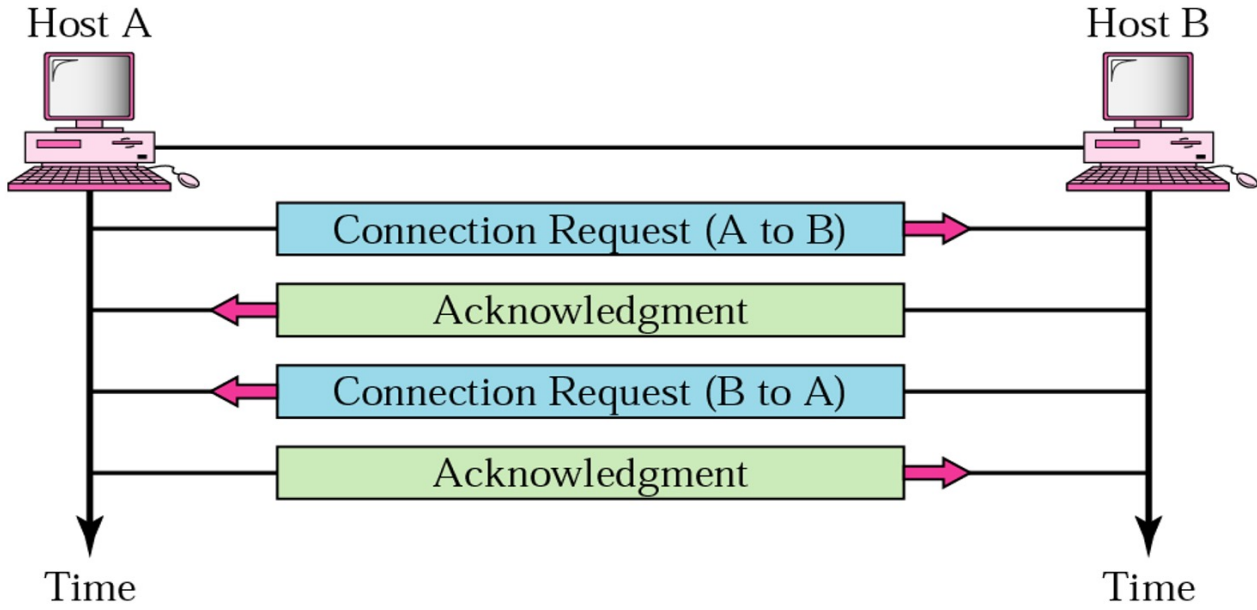


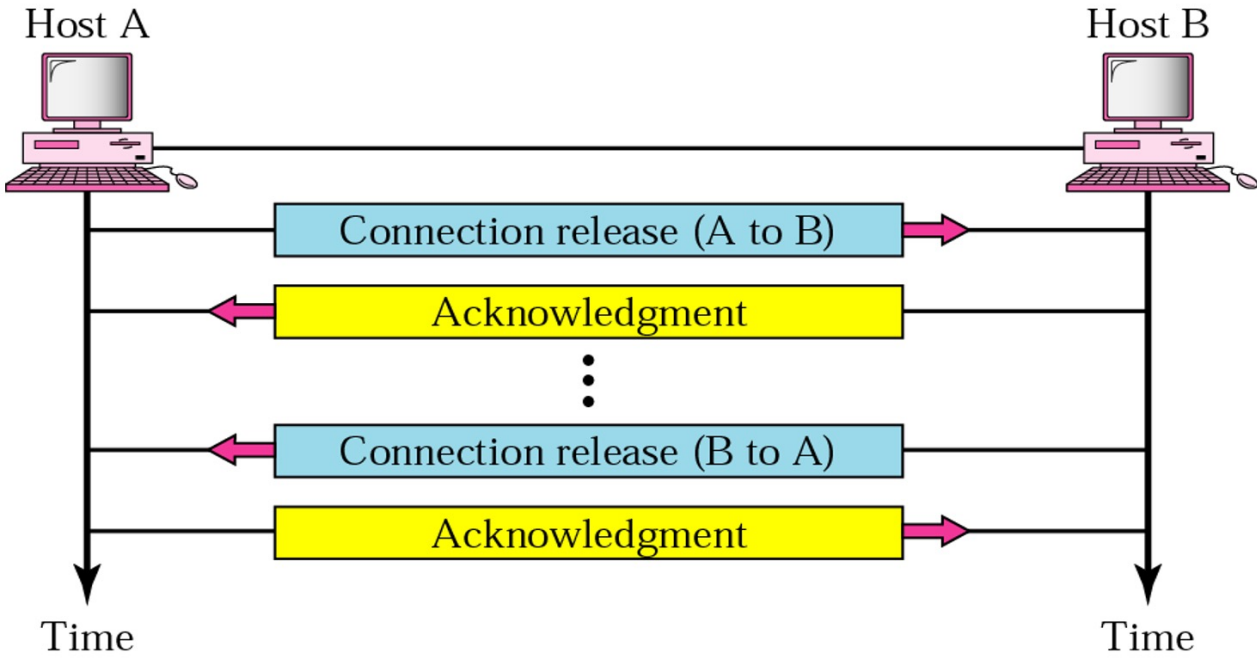
Transport layer (IP Addresses & Port Number)



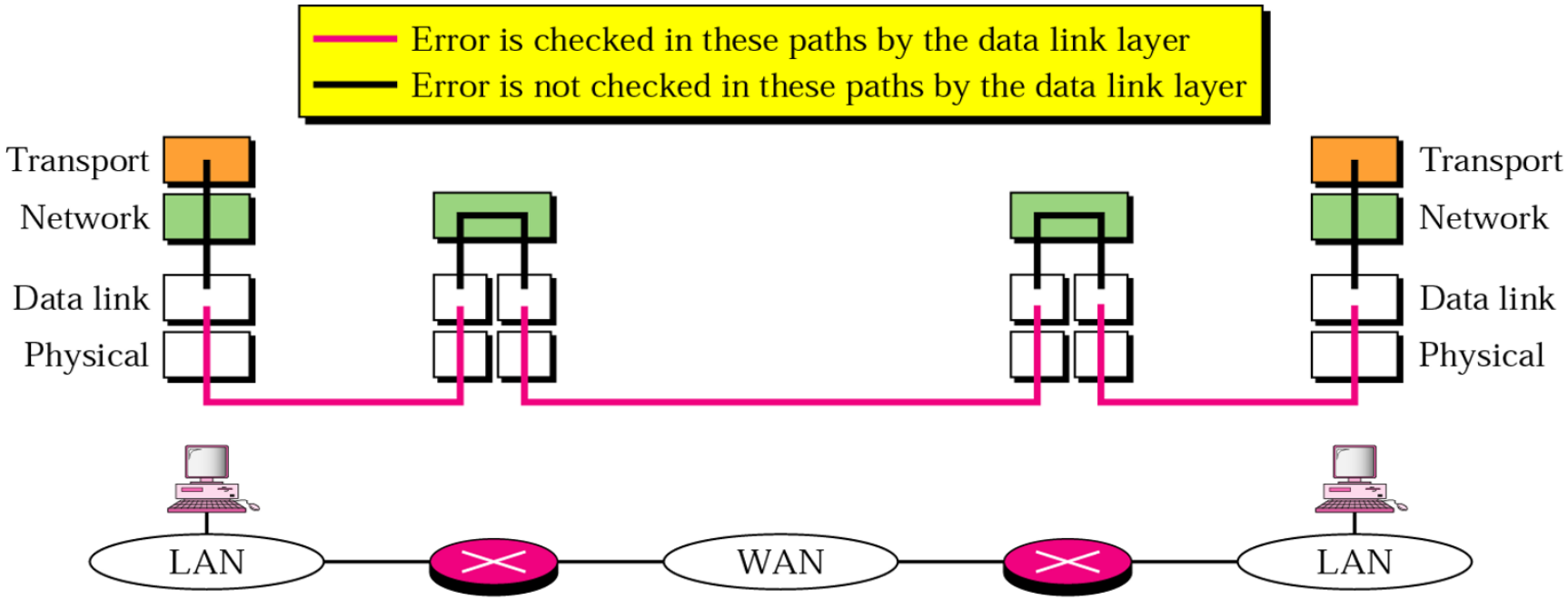
Transport layer (Socket Addresses & Connection Establishment and Termination)





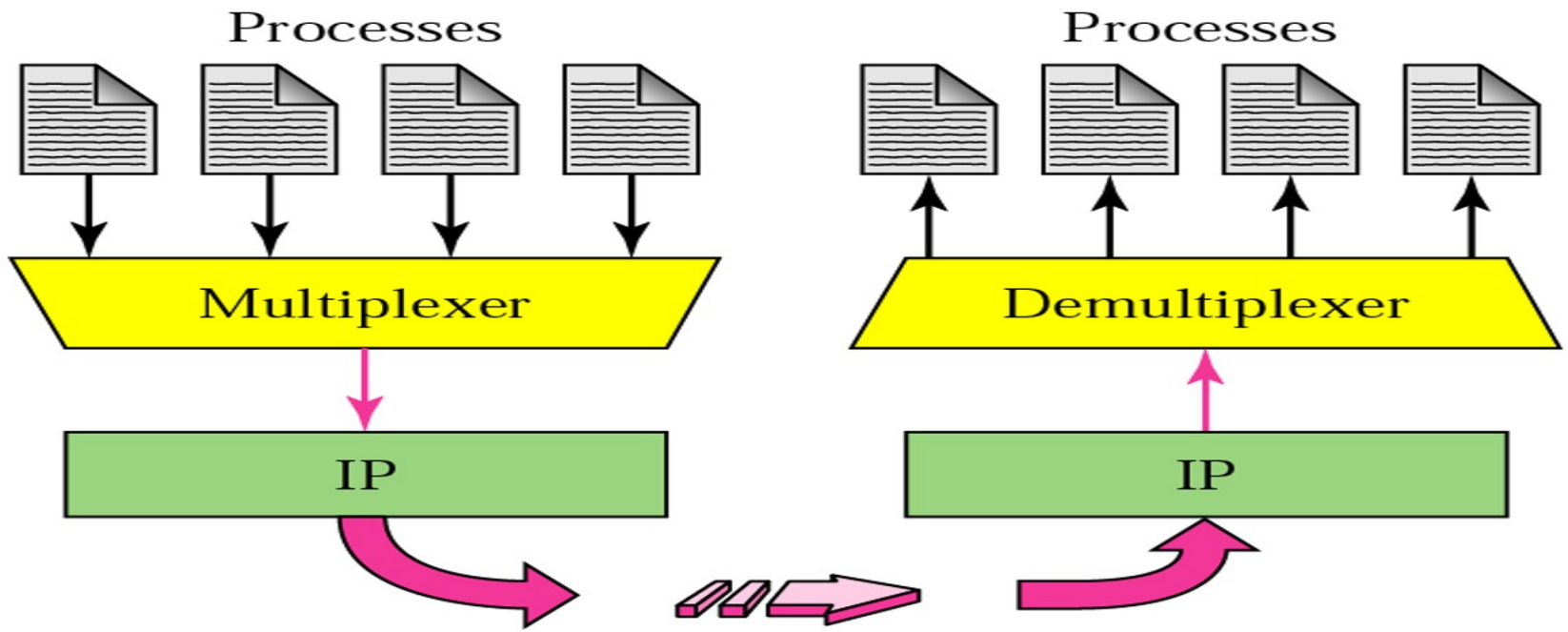


Transport layer (Error Control)



Transport layer (Syllabus)

- 3.1 Transport-layer services
- 3.2 Multiplexing and demultiplexing
- 3.3 Connectionless transport: UDP
- 3.4 Principles of reliable data transfer
- 3.5 Connection-oriented transport: TCP
  - segment structure
  - reliable data transfer
  - flow control
  - connection management
- 3.6 Principles of congestion control
- 3.7 TCP congestion control



Transport layer (Multiplexing and Demultiplexing)

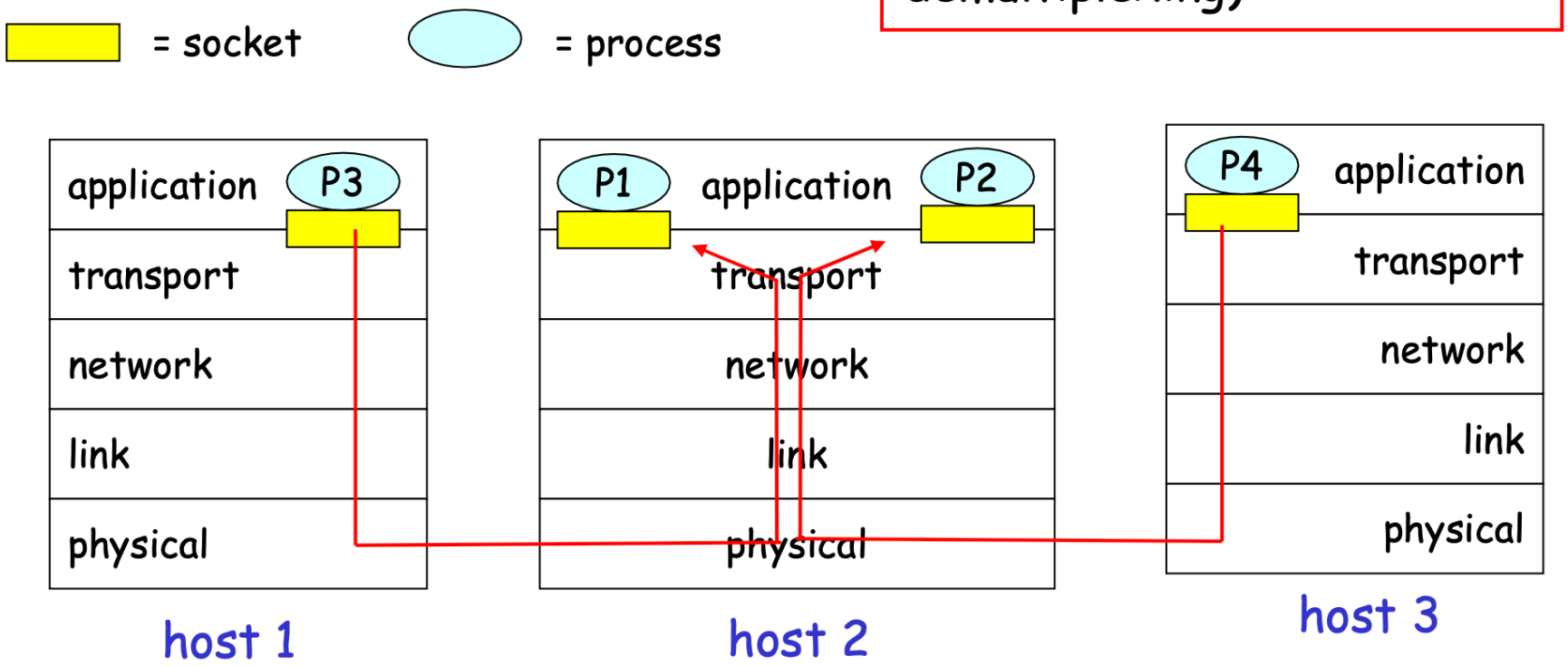
Multiplexing/demultiplexing

Demultiplexing at rcv host:

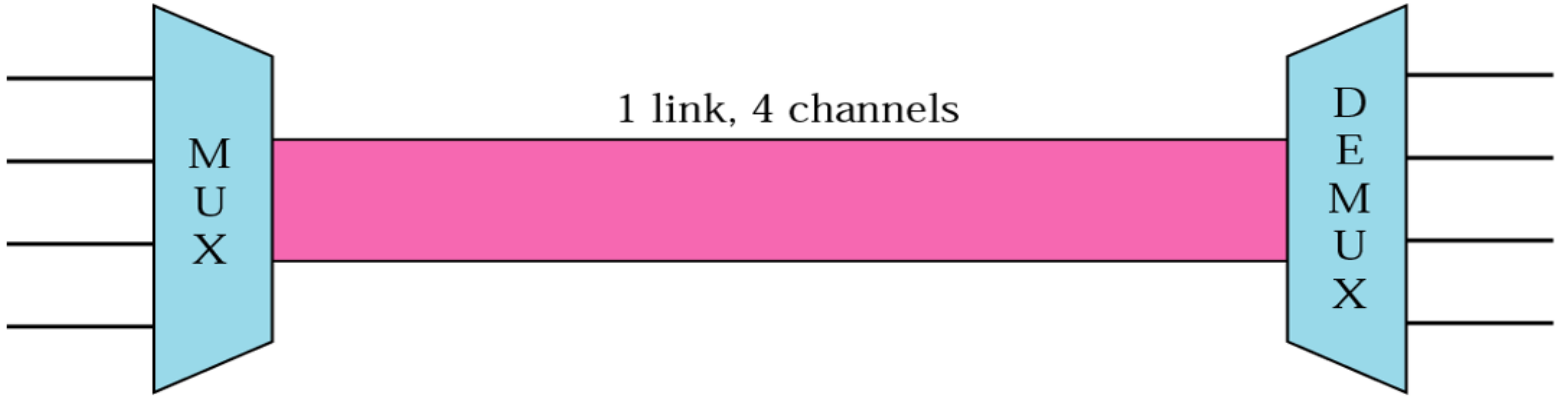
delivering received segments to correct socket

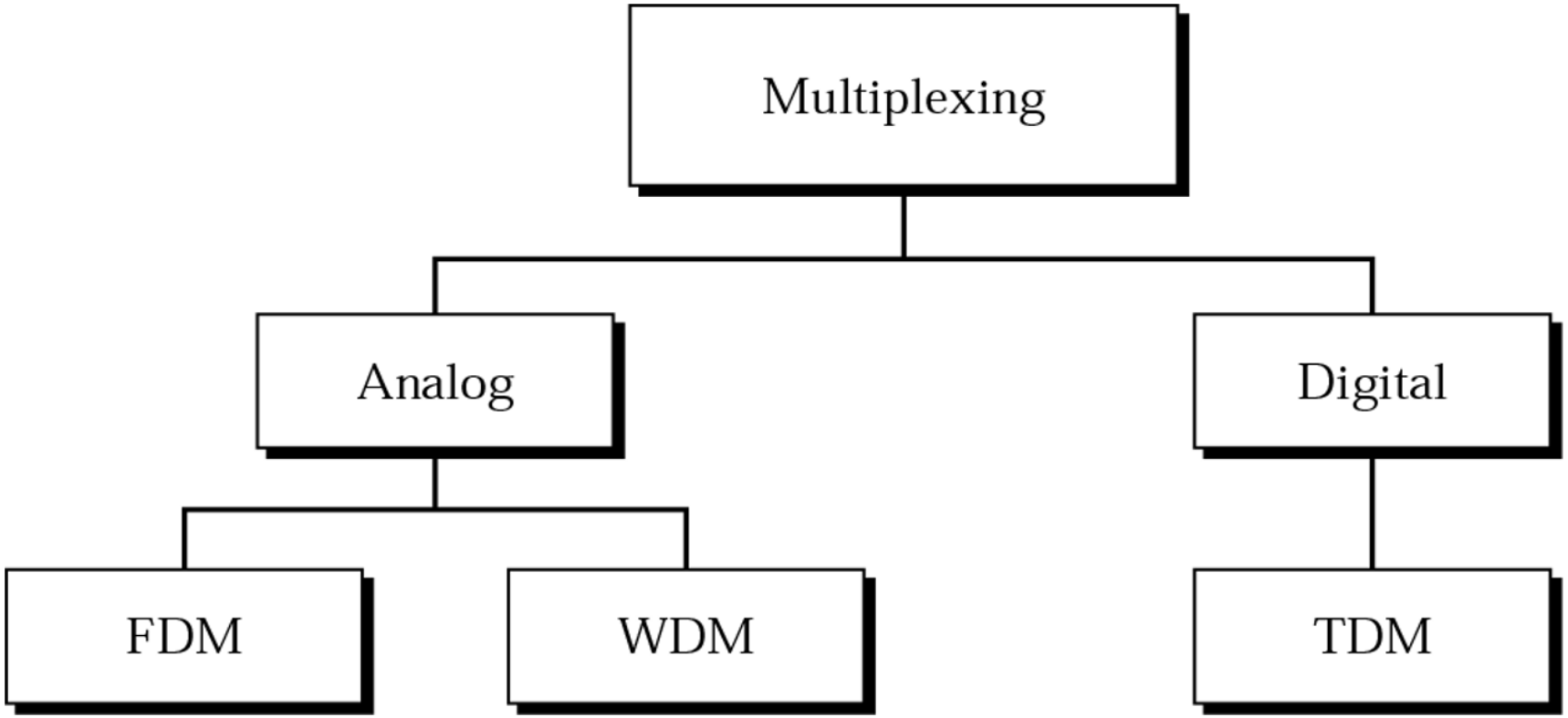
Multiplexing at send host:

gathering data from multiple sockets, enveloping data with header (later used for demultiplexing)



Transport layer (Multiplexing and Demultiplexing)



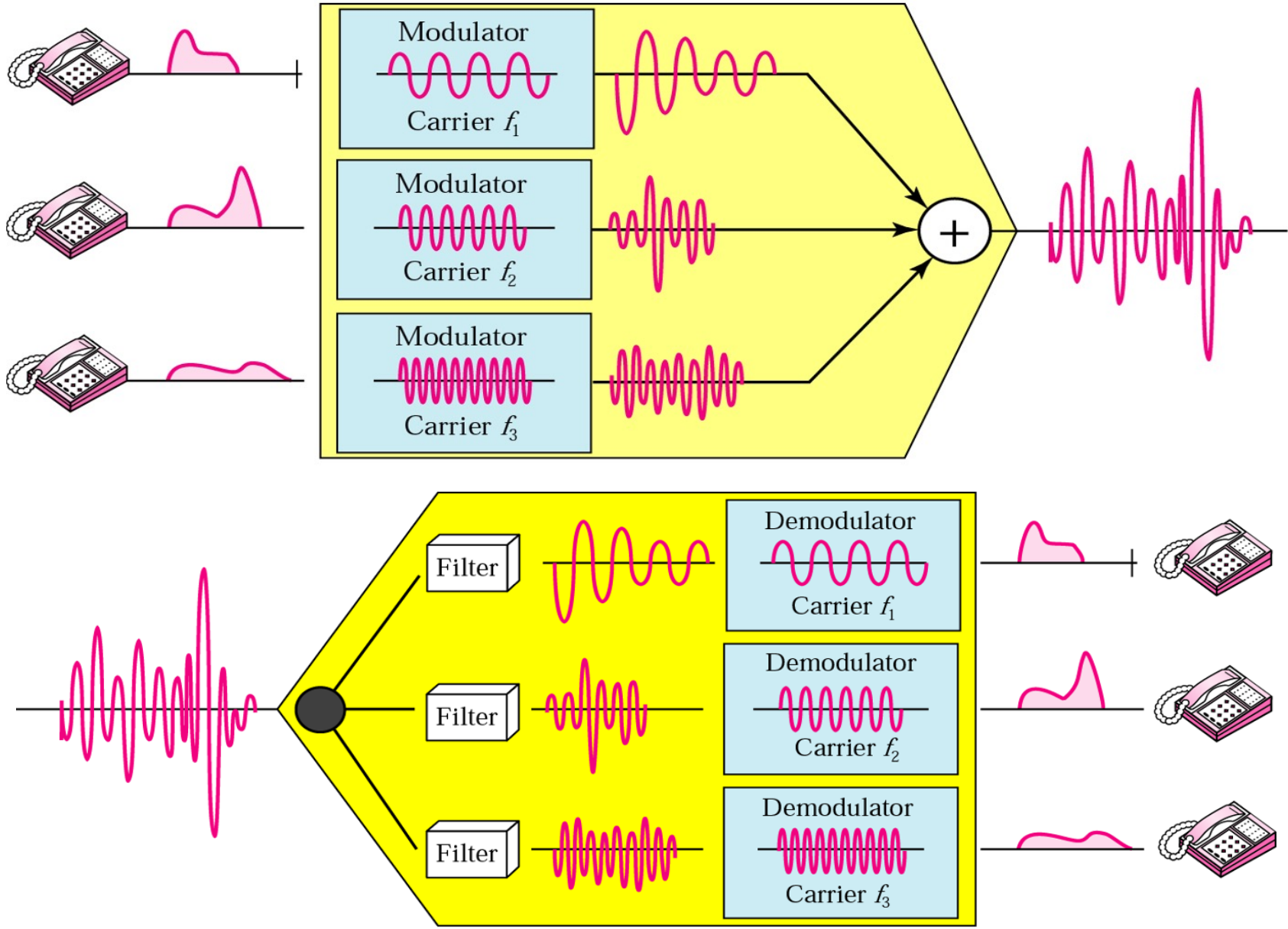


Transport layer (FDM- Frequency Division Multiplexing)

Note:

FDM is an analog multiplexing technique that combines signals.

Transport layer (FDM-Frequency Division Multiplexing)



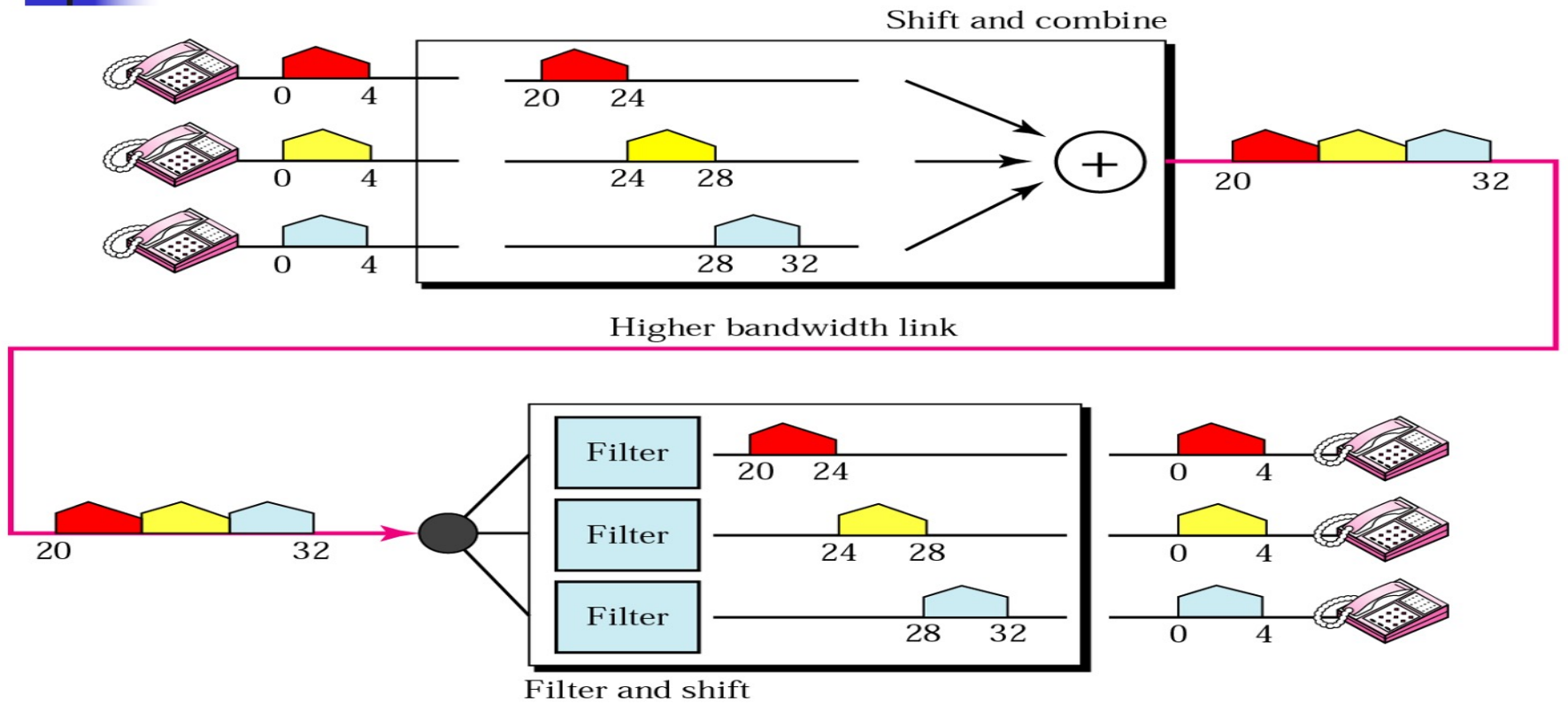
Transport layer (FDM- Frequency Division Multiplexing)

Example 1

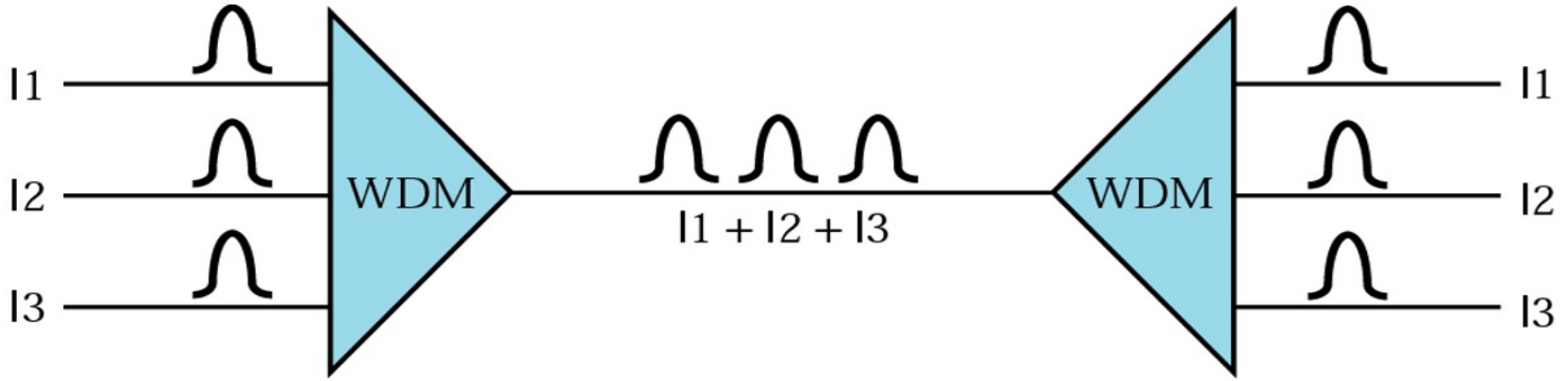
Assume that a voice channel occupies a bandwidth of 4 KHz. We need to combine three voice channels into a link with a bandwidth of 12 KHz, from 20 to 32 KHz. Show the configuration using the frequency domain without the use of guard bands.

Solution

Shift (modulate) each of the three voice channels to a different bandwidth, as shown in Figure 6.6.

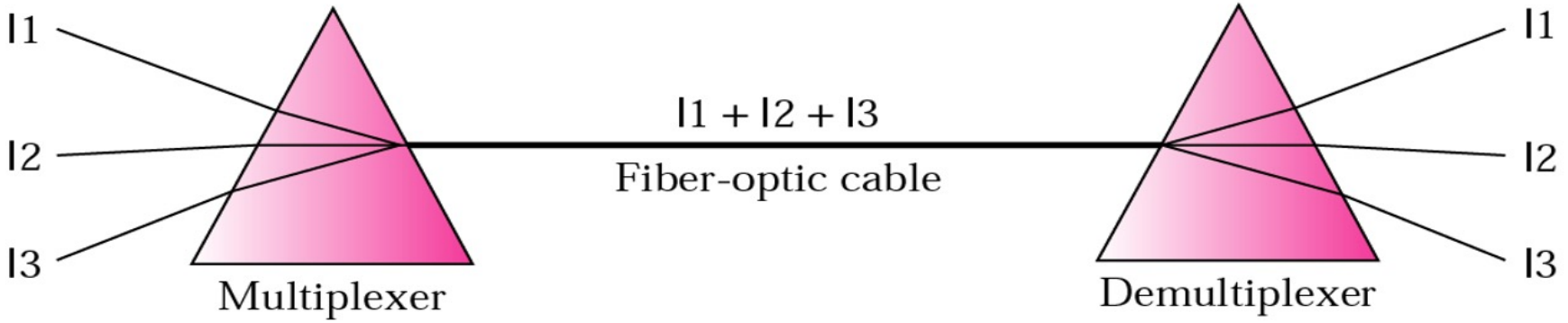


Transport layer (WDM- Wave Division Multiplexing)

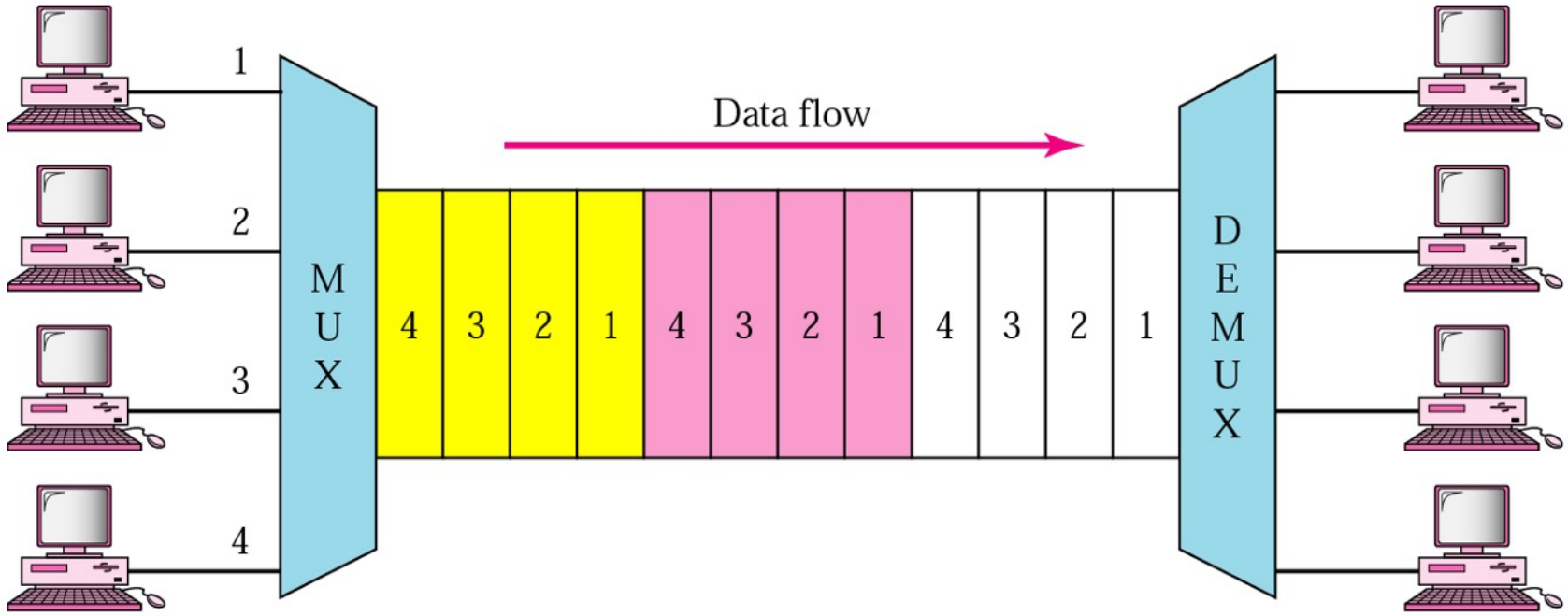


Note:

WDM is an analog multiplexing technique to combine optical signals.



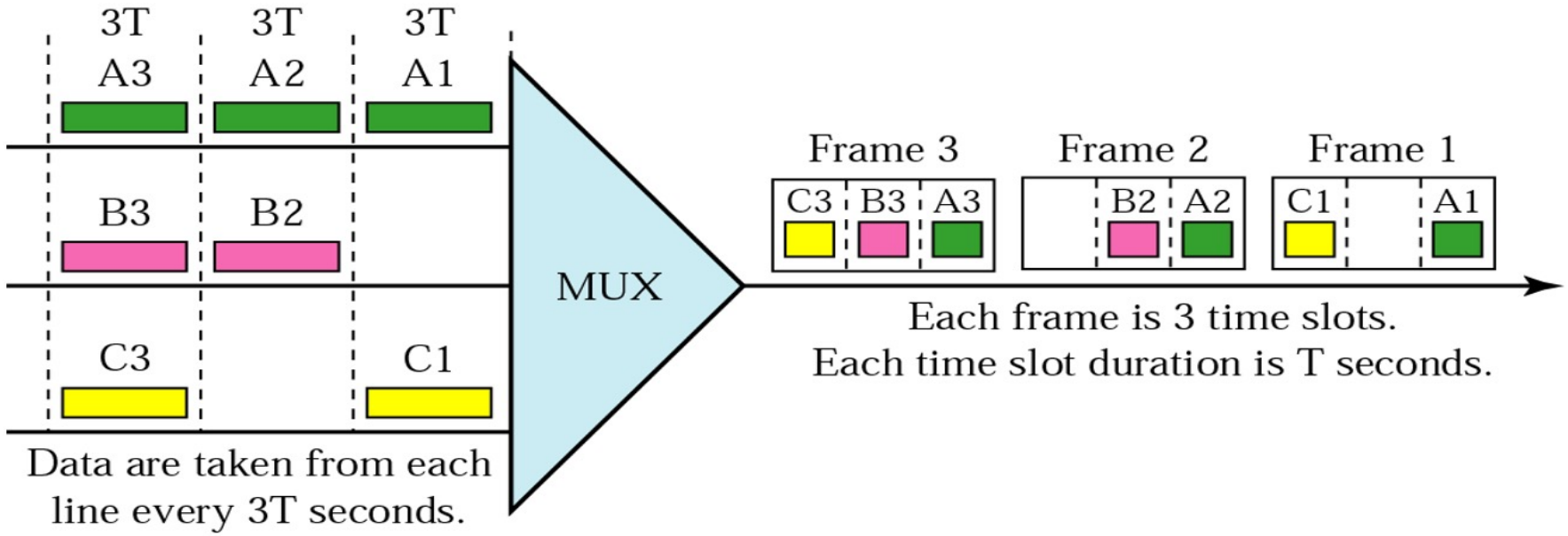
Transport layer (TDM- Time Division Multiplexing)



Note:

TDM is a digital multiplexing technique to combine data.

Transport layer (TDM- Time Division Multiplexing Frames)



Four 1-Kbps connections are multiplexed together. A unit is 1 bit. Find (1) the duration of 1 bit before multiplexing, (2) the transmission rate of the link, (3) the duration of a time slot, and (4) the duration of a frame?

Solution

We can answer the questions as follows:

1. ***The duration of 1 bit is 1/1 Kbps, or 0.001 s (1 ms).***
2. ***The rate of the link is 4 Kbps.***
3. ***The duration of each time slot 1/4 ms or 250 μs.***
4. ***The duration of a frame 1 ms.***

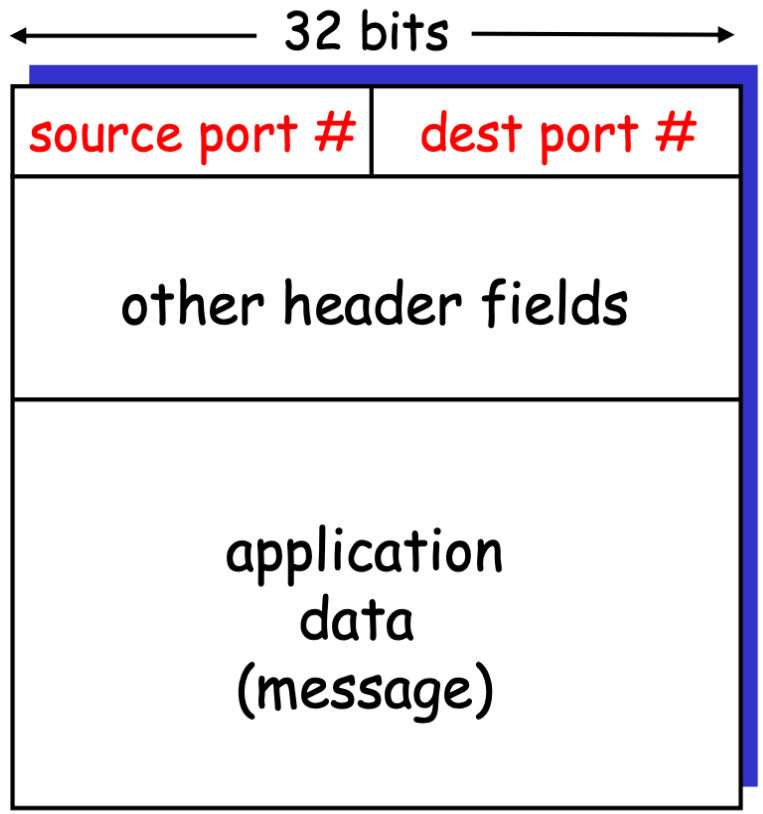
Transport layer (Difference between FDM, and TDM)

Sr no.	FDM	TDM
1.	The signals which are to be multiplexed are added in the time domain . But they occupy different slots in the frequency domain .	The signals which are to be multiplexed can occupy the entire bandwidth in the time domain .
2.	FDM is usually preferred for the analog signals .	TDM is preferred for the digital signals .
3.	Synchronization is not required .	Synchronization is required .
4.	The FDM requires a complex circuitry at Tx and Rx .	TDM circuitry is not very complex .
5.	FDM suffers from the problem of crosstalk due to imperfect BPF .	In TDM the problem of crosstalk is not severe .
6.	Due to bandwidth fading in the Tx medium , all the FDM channels are affected .	Due to fading only a few TDM channels will be affected .
7.	Due to slow narrowband fading taking place in the transmission channel may be affected in FDM .	Due to slow narrowband fading all the TDM channels may get wiped out .

Transport layer (Working of Demultiplexing)

How demultiplexing works

- host receives IP datagrams
  - each datagram has source IP address, destination IP address
  - each datagram carries 1 transport-layer segment
  - each segment has source, destination port number
- host uses IP addresses & port numbers to direct segment to appropriate socket



TCP/UDP segment format

Transport layer (Connectionless Demultiplexing)

Connectionless demultiplexing

- Create sockets with port numbers:

```
DatagramSocket mySocket1 = new
   DatagramSocket(12534);
DatagramSocket mySocket2 = new
   DatagramSocket(12535);
```

- UDP socket identified by two-tuple:

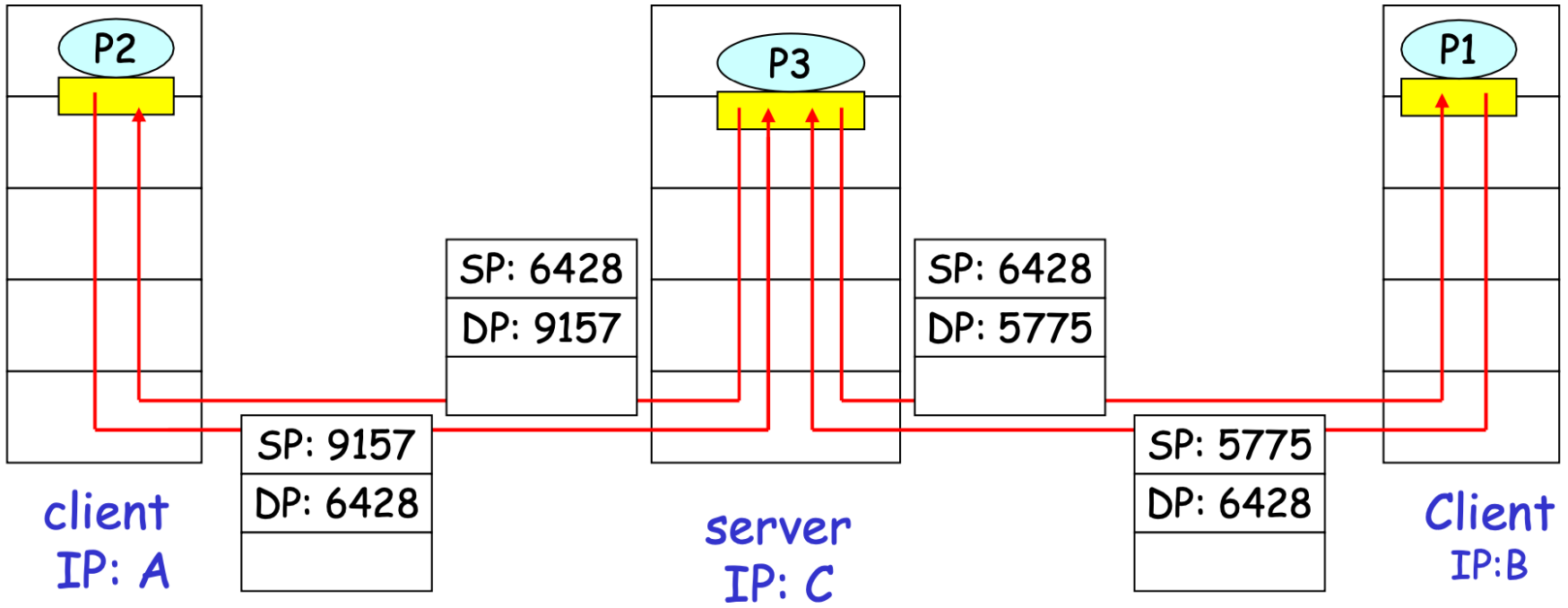
(dest IP address, dest port number)

- When host receives UDP segment:
  - checks destination port number in segment
  - directs UDP segment to socket with that port number
- IP datagrams with different source IP addresses and/or source port numbers directed to same socket

Transport layer (Connectionless Demultiplexing)

Connectionless demux (cont)

```
DatagramSocket serverSocket = new DatagramSocket(6428);
```



SP provides "return address"

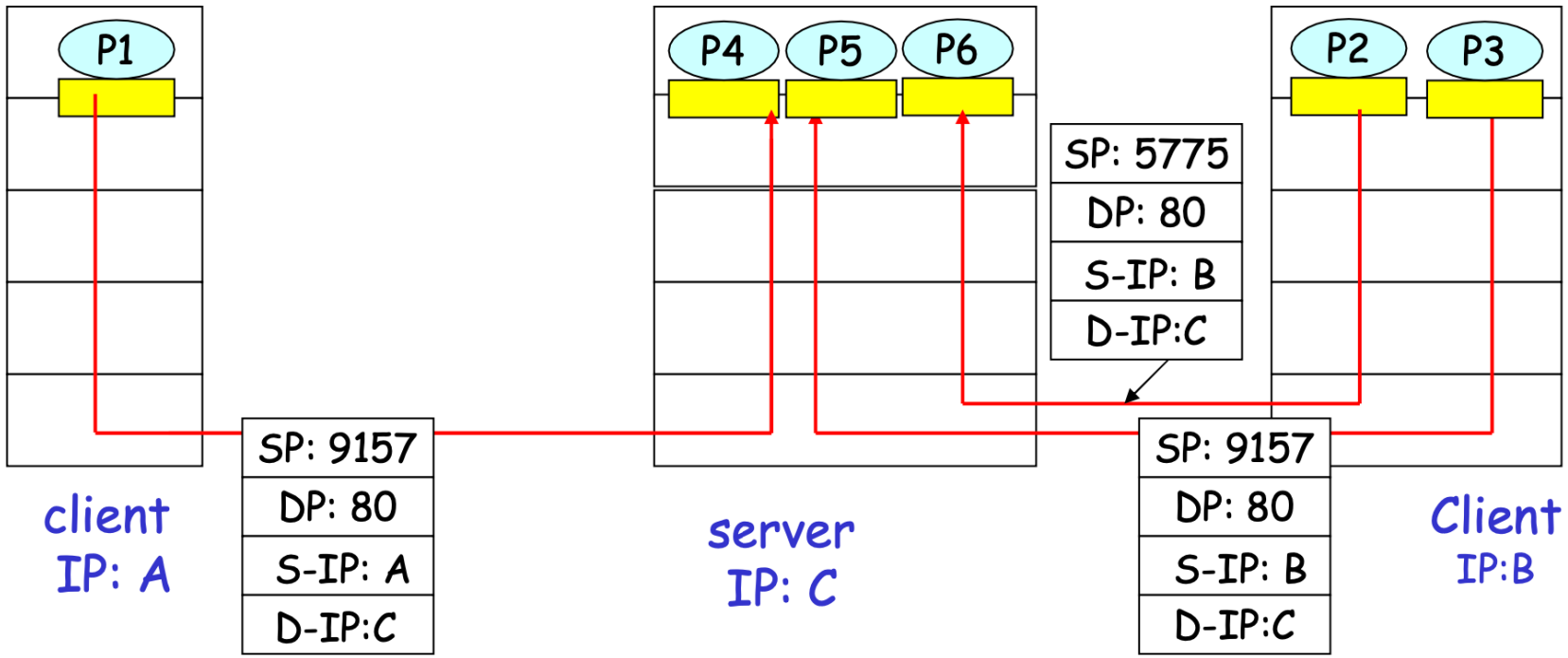
Transport layer (Connection-oriented Demultiplexing)

Connection-oriented demux

- TCP socket identified by 4-tuple:
  - source IP address
  - source port number
  - dest IP address
  - dest port number
- recv host uses all four values to direct segment to appropriate socket
- Server host may support many simultaneous TCP sockets:
  - each socket identified by its own 4-tuple
- Web servers have different sockets for each connecting client
  - non-persistent HTTP will have different socket for each request

Transport layer (Connection-oriented Demultiplexing)

Connection-oriented demux (cont)



Transport layer (Syllabus)

- 3.1 Transport-layer services
- 3.2 Multiplexing and demultiplexing
- 3.3 Connectionless transport: UDP
- 3.4 Principles of reliable data transfer
- 3.5 Connection-oriented transport: TCP
  - segment structure
  - reliable data transfer
  - flow control
  - connection management
- 3.6 Principles of congestion control
- 3.7 TCP congestion control

Note:

UDP is a connectionless, unreliable protocol that has no flow and error control. It uses port numbers to multiplex data from the application layer.

Transport layer (UDP-User Datagram Protocol)

UDP: The User Datagram Protocol

- **UDP is another transport protocol in the TCP/IP suite**
- **UDP provides an unreliable datagram service**
  - **Packets may be lost or delivered out of order**
  - **Users exchange datagrams (not streams)**
  - **Connection-less**
  - **Not buffered -- UDP accepts data and transmits immediately (no buffering before transmission)**
  - **Full duplex -- concurrent transfers can take place in both directions**

Transport layer (UDP-User Datagram Protocol)

UDP: User Datagram Protocol

- In TCP/IP protocol suite, using IP to transport datagram (similar to IP datagram).
- Allows a application to send datagram to other application on the remote machine.
- Delivery and duplicate detection are not guaranteed.
- Low overhead: faster than TCP

Transport layer (UDP-User Datagram Protocol Characteristics)

UDP Characteristics

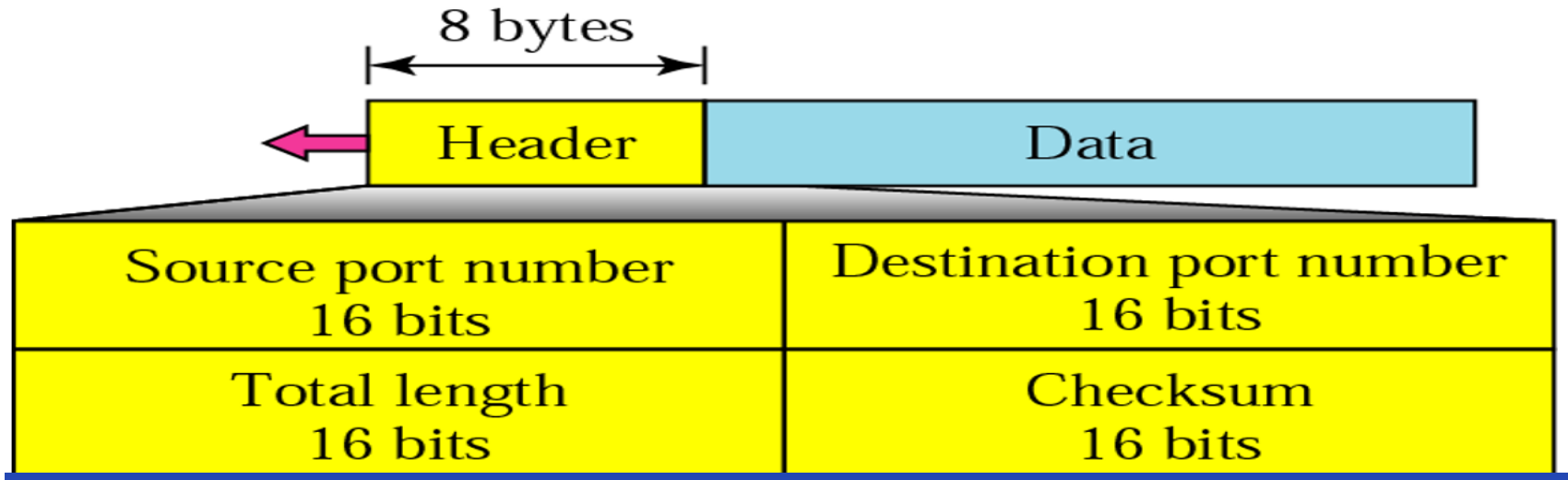
- **End-to-End**: an application sends/receives data to/from another application.
- **Connectionless**: Application does not need to preestablish communication before sending data; application does not need to terminate communication when finished.
- **Message-oriented**: application sends/receives individual messages (UDP datagram), not packets.
- **Best-effort**: same best-effort delivery semantics as IP. I.e. message can be lost, duplicated, and corrupted.
- **Arbitrary interaction**: application communicates with many or one other applications.
- **Operating system independent**: identifying application does not depend on O/S.

Transport layer (UDP-User Datagram Protocol)

Table 22.1 *Well-known ports used by UDP*

Port	Protocol	Description
7	Echo	Echoes a received datagram back to the sender
9	Discard	Discards any datagram that is received
11	Users	Active users
13	Daytime	Returns the date and the time
17	Quote	Returns a quote of the day
19	Chargen	Returns a string of characters
53	Nameserver	Domain Name Service
67	Bootps	Server port to download bootstrap information
68	Bootpc	Client port to download bootstrap information
69	TFTP	Trivial File Transfer Protocol
111	RPC	Remote Procedure Call
123	NTP	Network Time Protocol
161	SNMP	Simple Network Management Protocol
162	SNMP	Simple Network Management Protocol (trap)

Transport layer (UDP-User Datagram Protocol Header Format)



Note:

The calculation of checksum and its inclusion in the user datagram are optional.

UDP is a convenient transport-layer protocol for applications that provide flow and error control. It is also used by multimedia applications.

Transport layer (UDP-User Datagram Protocol Header Format)

UDP Header Fields

- ***UDP Destination Port***: identifies destination process
- ***UDP Source Port***: optional -- identifies source process for replies, or zero
- ***Message Length***: length of datagram in bytes, including header and data
- ***Checksum***: optional -- 16-bit checksum over header and data, or zero

Transport layer (UDP vs TCP)

UDP Versus TCP (1)

- **Choice of UDP versus TCP is based on:**
  - Functionality
  - Performance
- **Performance**
  - TCP's window-based flow control scheme leads to bursty bulk transfers (not rate based)
  - TCP's "slow start" algorithm can reduce throughput
  - TCP has extra overhead per segment
  - UDP can send small, inefficient datagrams

UDP Versus TCP (2)

- **Reliability**
  - TCP provides reliable, in-order transfers
  - UDP provides unreliable service -- application must accept or deal with
    - Packet loss due to overflows and errors
    - Out-of-order datagrams
- **Multicast and broadcast**
  - Supported only by UDP
  - TCP's error control scheme does not lend itself to reliable multicast
- **Data size**
  - UDP datagrams limited to IP MTU (64KB)

UDP Versus TCP (3)

- **Application complexity**
  - Application-level *framing* can be difficult using TCP because of the Nagle algorithm
  - Nagle algorithm controls when TCP segments are sent to use IP datagrams efficiently
  - But, data may be received and read by applications in different units than how it was sent

UDP versus TCP (4)

- **Which is used for ...**
  - HyperText Transfer Protocol (HTTP)?
  - File Transfer Protocol (FTP)?
  - Telnet?
  - Post Office Protocol (POP)?
  - Remote WHO (rwho)?
  - MBONE audio/video?
  - Real Player?
  - Network File System (NFS)?

Transport layer
PDU- Protocol Data Unit

PDU(SEGMENT)

- In the Transport layer a message is normally divided into transmittable segment.
- UDP treats each segment separately
- TCP creates a relationship between the segments using sequence numbers.

Transport layer UDP checksum

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

Sender:

- treat segment contents, including header fields, as sequence of 16-bit integers
- checksum: addition (one's complement sum) of segment contents
- sender puts checksum value into UDP checksum field

UDP Checksum

Receiver:

- compute checksum of received segment
- check if computed checksum equals checksum field value:
  - NO - error detected
  - YES - no error detected. *But maybe errors nonetheless?* More later

Transport layer
CHECK SUM

Internet checksum: example

example: add two 16-bit integers

	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
wraparound	(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

Note: when adding numbers, a carryout from the most significant bit needs to be added to the result

Transport layer
Solve Example

- 1) 1100001110000010
- 0111100110011101

- 2) 0001100110001101
- 1110001100101010

Transport layer
UDP
An Application

Identifying An Application

- UDP cannot extend IP address
  - No unused bits
- Cannot use OS-dependent quantity
  - Process ID, Task number, Job name
- Must work on all computer systems
- Technique
  - Each application assigned unique integer
  - Called ***protocol port number***

Transport layer
UDP
Protocol Port Number

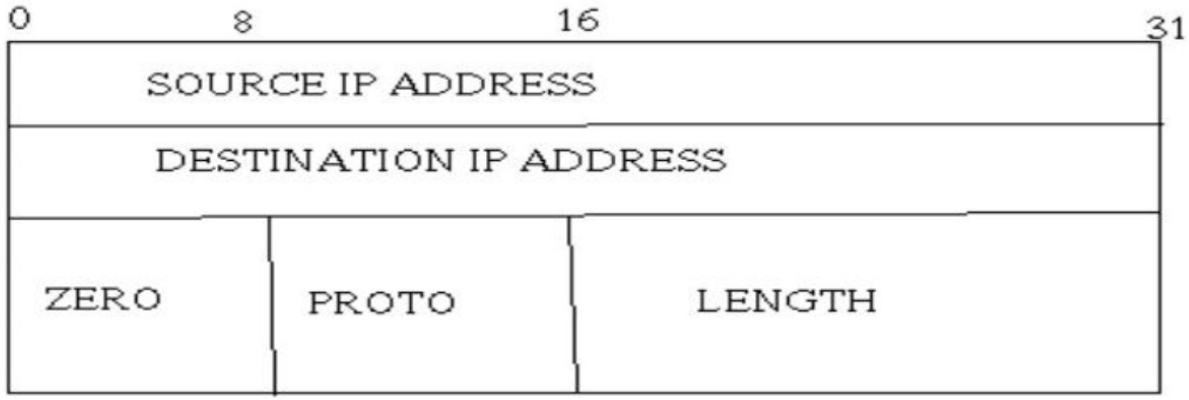
Protocol Port Number

- UDP uses ***Port Number*** to identify an application as an endpoint.
- UDP messages are delivered to the port specified in the message by the sending application
- In general, a port can be used for any datagram, as long as the sender and the receiver agrees
- In practice, a collection of well-known ports are used for special purposes such as telnet, ftp, and email. E.g. port 7 for Echo application.
- Local operating system provides an interface for processes to specify and access a port.

Transport layer
UDP
Checksum and
Pseudo
Header

Checksum and Pseudo Header

- UDP uses a pseudo-header to verify that the UDP message has arrived at both the correct machine and the correct port.



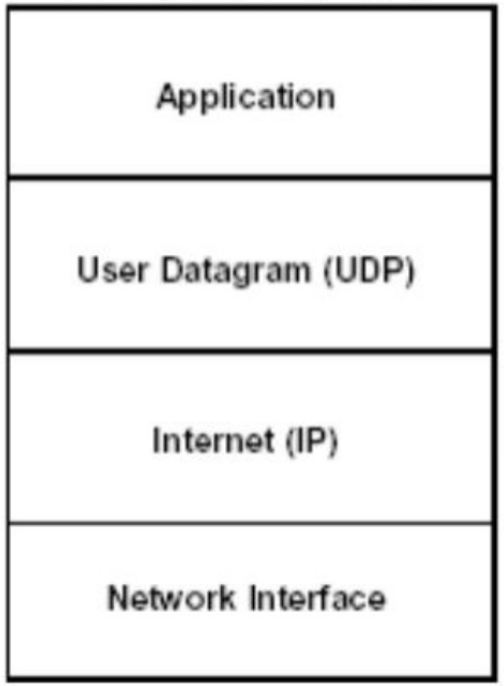
- Proto : IP protocol type code.
- Length : Length of the UDP datagram.

Transport layer
UDP
Encapsulation
and Layering

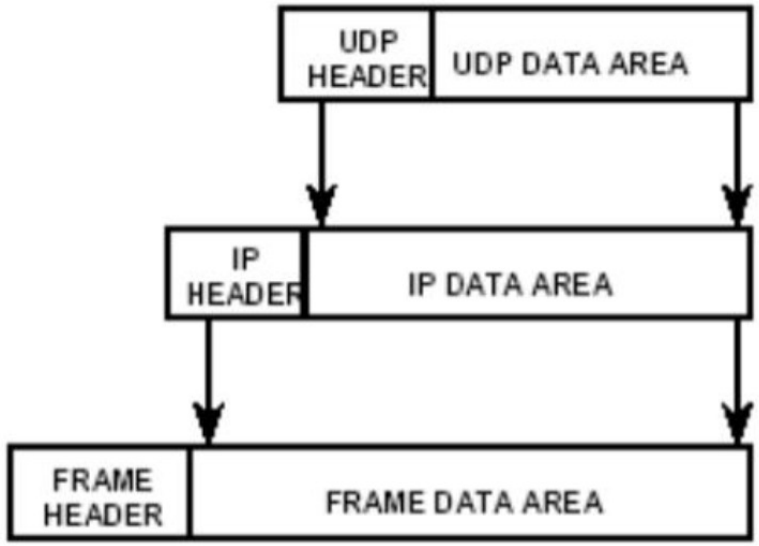
Encapsulation and Layering

- Protocol layering

Conceptual Layering



- UDP encapsulation



- UDP message is encapsulated into an IP datagram.
- IP datagram in turn is encapsulated into a physical frame for actually delivery.

Transport layer (Syllabus)

- 3.1 Transport-layer services
- 3.2 Multiplexing and demultiplexing
- 3.3 Connectionless transport: UDP
- 3.4 Principles of reliable data transfer
- 3.5 Connection-oriented transport: TCP
  - segment structure
  - reliable data transfer
  - flow control
  - connection management
- 3.6 Principles of congestion control
- 3.7 TCP congestion control

Transport layer (Flow and Error Control)

Flow and Error Control

Flow Control

Error Control

Transport layer (Flow and Error Control)

Note:

Flow control refers to a set of procedures used to restrict the amount of data that the sender can send before waiting for acknowledgment.

Note:

Error control in the data link layer is based on automatic repeat request, which is the retransmission of data.

Transport layer
Flow Control

- Receiver must inform the sender before the limits are reached and request that the transmitter to send fewer frames or stop temporarily.

- Since the rate of processing is often slower than the rate of transmission, receiver has a block of memory (buffer) for storing incoming data until they are processed.

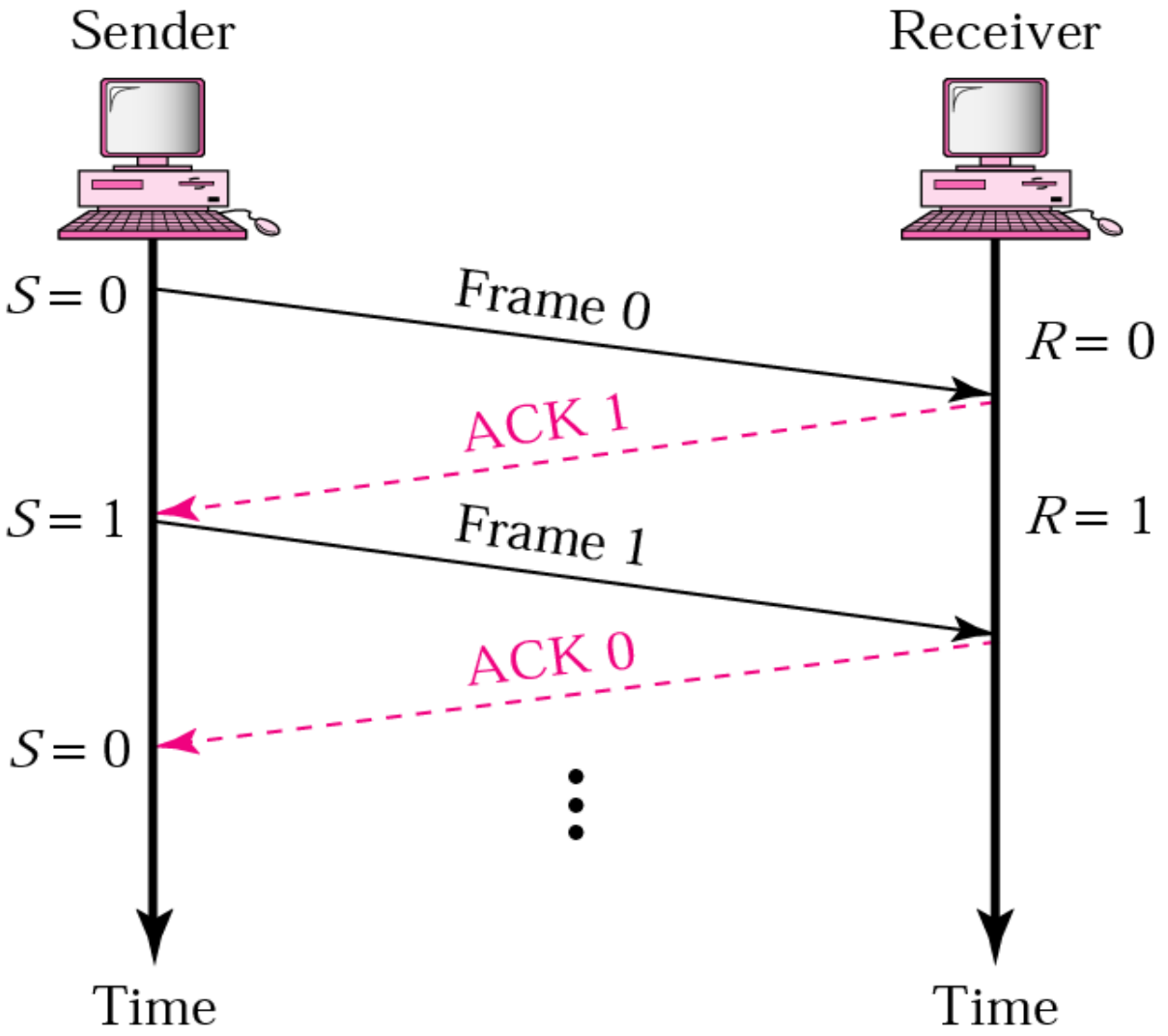
Transport layer
Error Control

- Error control includes both error detection – correction and notification about error.
- It allows the receiver to inform the sender if a frame is lost or damaged during transmission and coordinates the retransmission of those frames by the sender.
- Error control in the data link layer is based on automatic repeat request (ARQ). Whenever an error is detected, specified frames are retransmitted

Transport layer Error Control Mechanisms

- Stop-and-Wait
- Go-Back-N ARQ
- Selective-Repeat ARQ

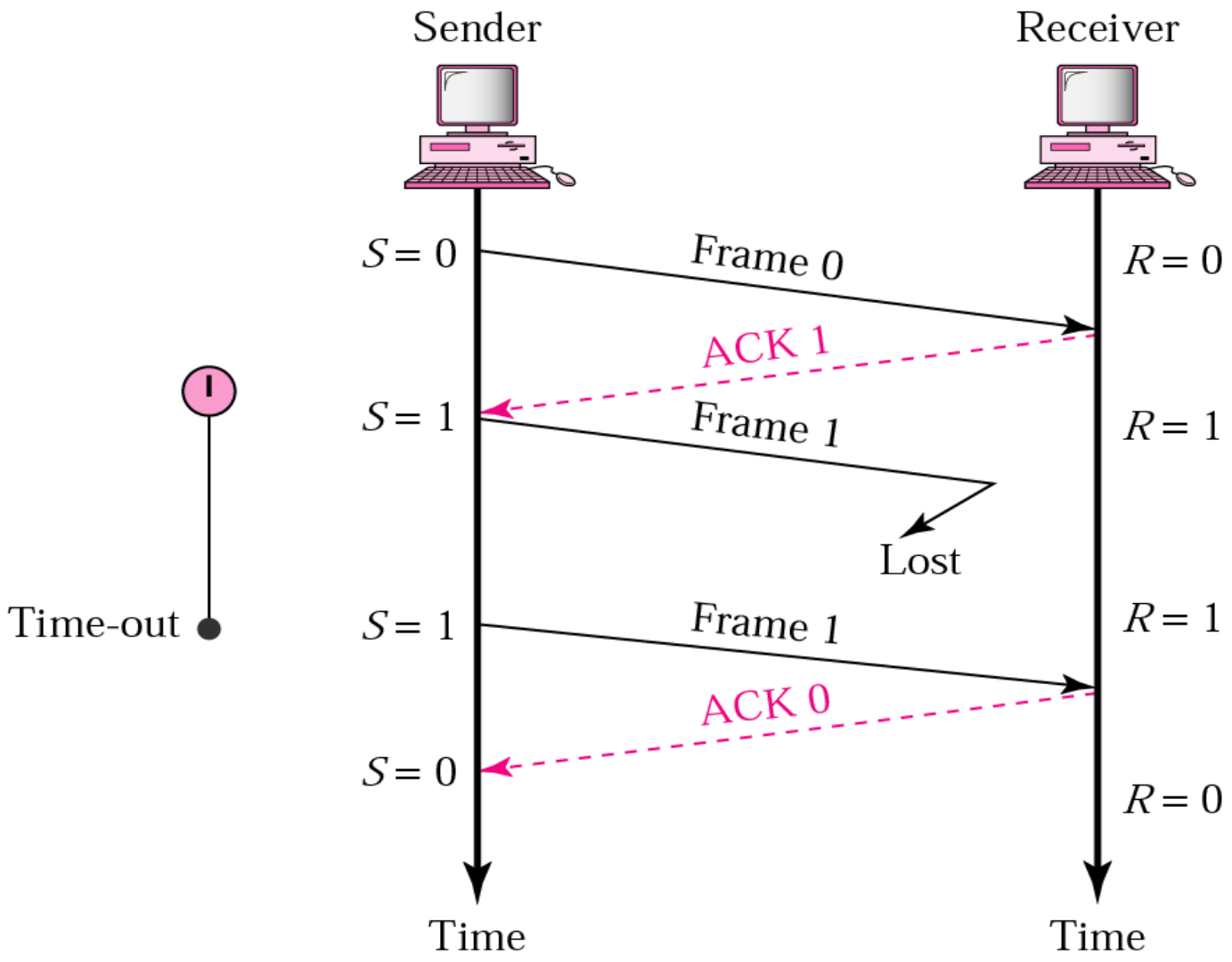
Transport layer
Error and Flow Control Mechanisms
Stop and Wait



Continue..

- Sender keeps a copy of the last frame until it receives an acknowledgement.
- For identification, both data frames and acknowledgements (ACK) frames are numbered alternatively 0 and 1.
- Sender has a control variable (S) that holds the number of the recently sent frame. (0 or 1)
- Receiver has a control variable (R) that holds the number of the next frame expected (0 or 1).
- Sender starts a timer when it sends a frame. If an ACK is not received within a allocated time period, the sender assumes that the frame was lost or damaged and resends it
- Receiver send only positive ACK if the frame is intact.
- ACK number always defines the number of the next expected frame

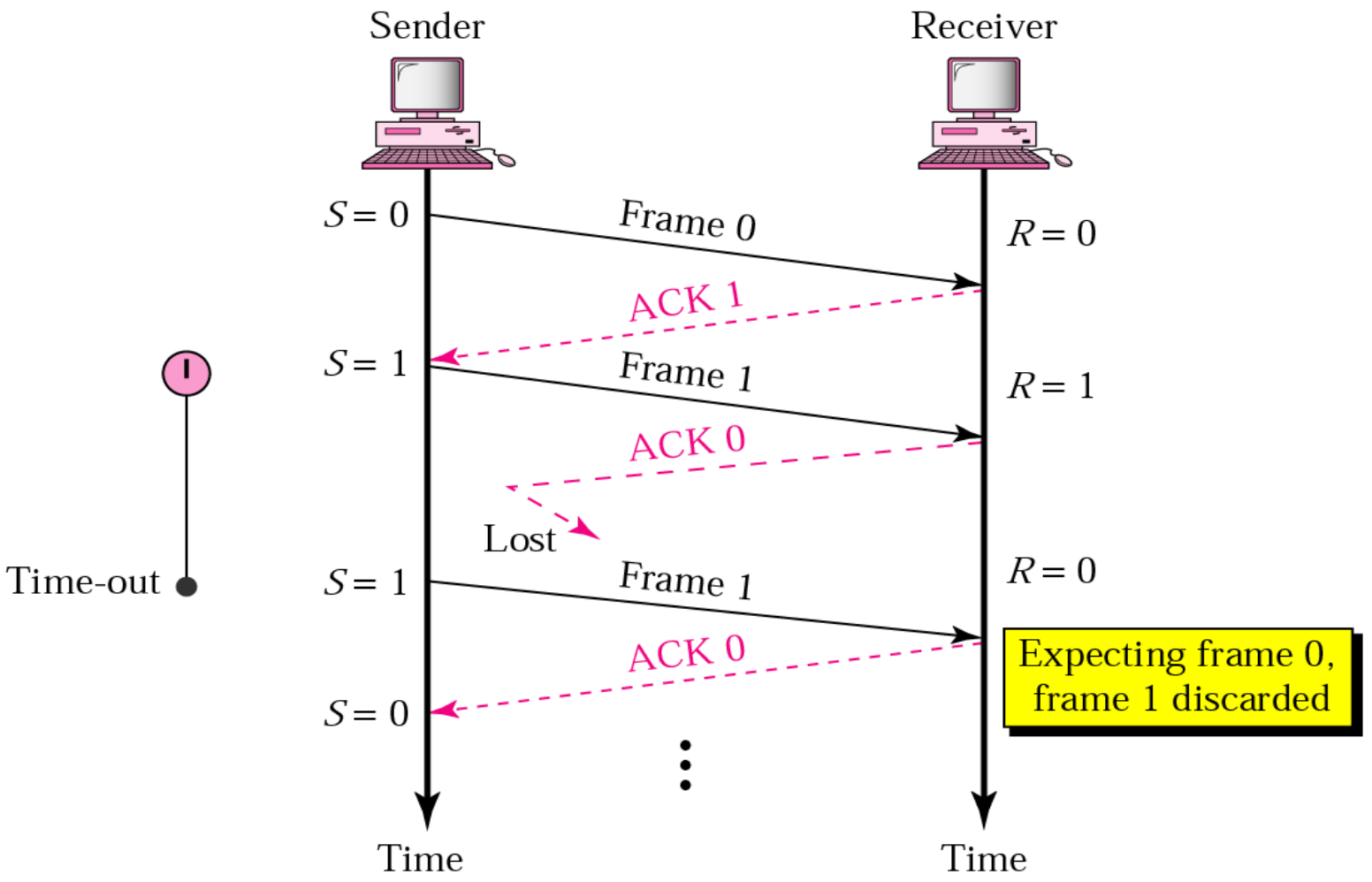
Stop-and-Wait ARQ (Automatic Repeat Request), Corrupt Frame



Continue..

- When a receiver receives a damaged frame, it discards it and keeps its value of R.
- After the timer at the sender expires, another copy of frame 1 is sent.

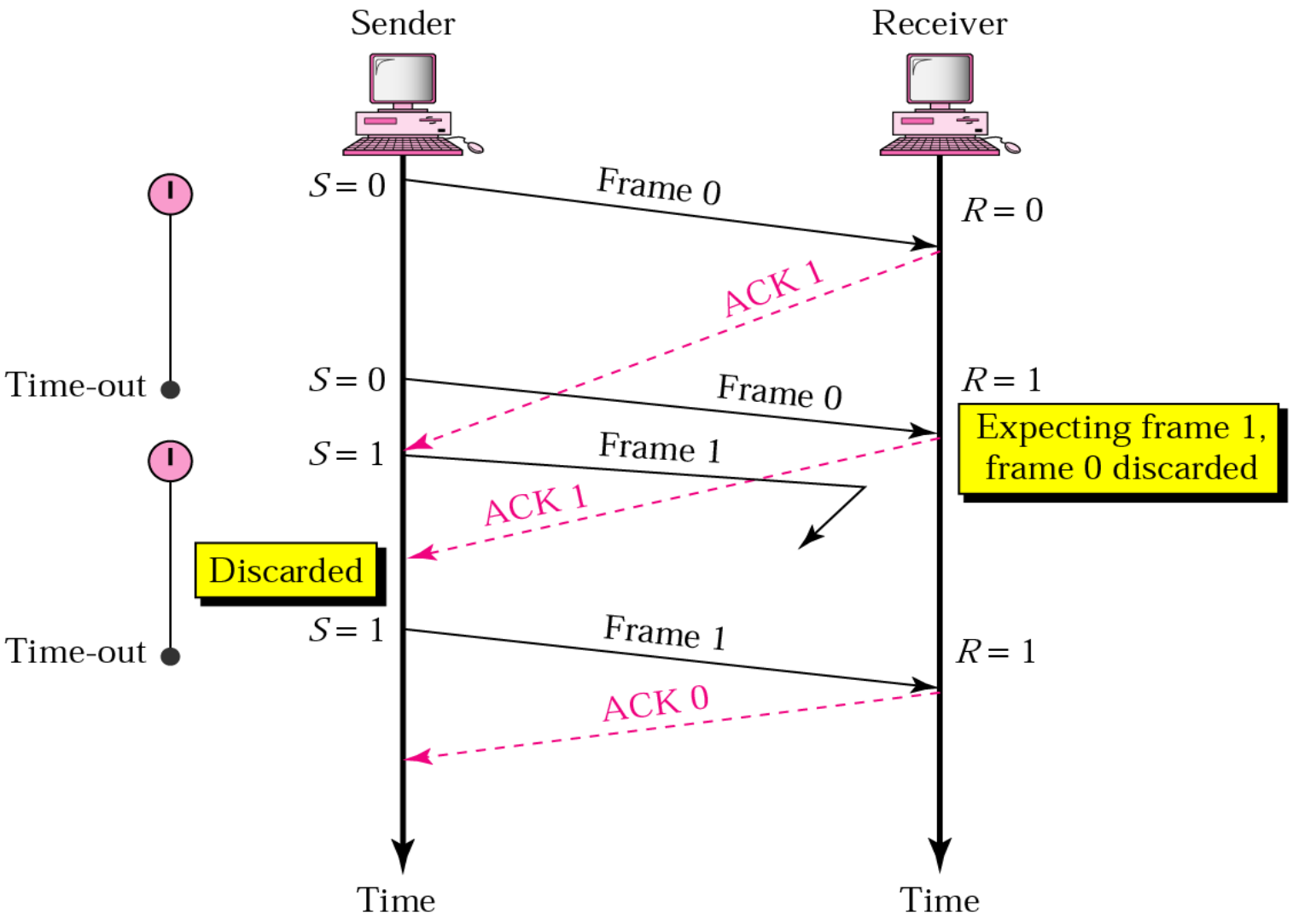
Stop-and-Wait, lost ACK frame



Continue..

- If the sender receives a damaged ACK, it discards it.
- When the timer of the sender expires, the sender retransmits frame 1.
- Receiver has already received frame 1 and expecting to receive frame 0 (R=0). Therefore, it discards the second copy of frame 1.

Stop-and-Wait, delayed ACK frame



Continue..

- The ACK can be delayed at the receiver or due to some problem
- It is received after the timer for frame 0 has expired.
- Sender retransmitted a copy of frame 0. However, R =1 means receiver expects to see frame 1. Receiver discards the duplicate frame 0.
- Sender receives 2 ACKs, it discards the second ACK

Continue..

Note:

In Stop-and-Wait ARQ, numbering frames prevents the retaining of duplicate frames.

Note:

Numbered acknowledgments are needed if an acknowledgment is delayed and the next frame is lost.

Disadvantage of Stop-and-Wait

- In stop-and-wait, at any point in time, there is only one frame that is sent and waiting to be acknowledged.
- This is not a good use of transmission medium.
- To improve efficiency, multiple frames should be in transition while waiting for ACK. (Sliding Window)
- Two protocol use the above concept,
  - **Go-Back-N ARQ**
  - **Selective Repeat ARQ**

Transport layer Error and Flow Control Mechanisms Sliding Window Protocols

- Go-Back-N ARQ
- Selective Repeat ARQ

Sliding Window

Sender

- Maintain sequence number of frames it is permitted to send -Sending window

Receiver

- Maintain sequence number of frames it is expected to accept - Receiver window

Sliding window protocol allow us to send multiple packets together

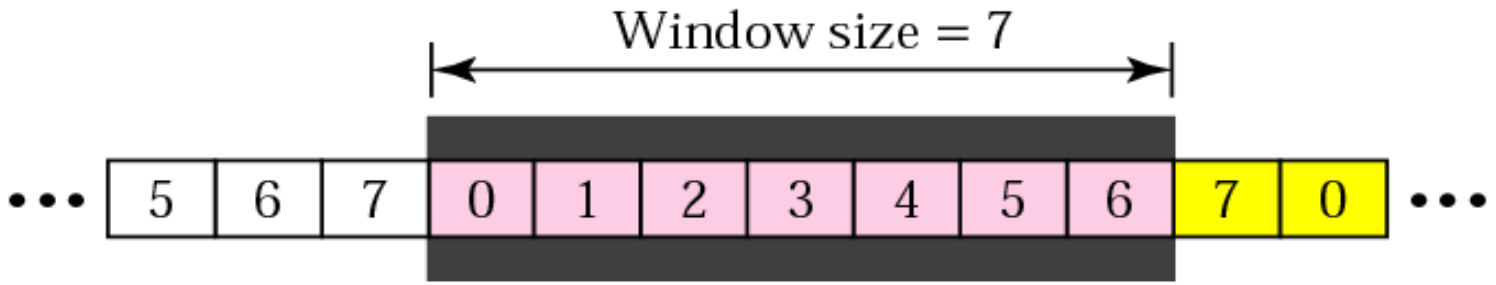
Go Back ARQ

- We can send up to W frames before worrying about ACKs.
- We keep a copy of these frames until the ACKs arrive.
- This procedure requires additional features to be added to Stop-and-Wait ARQ.

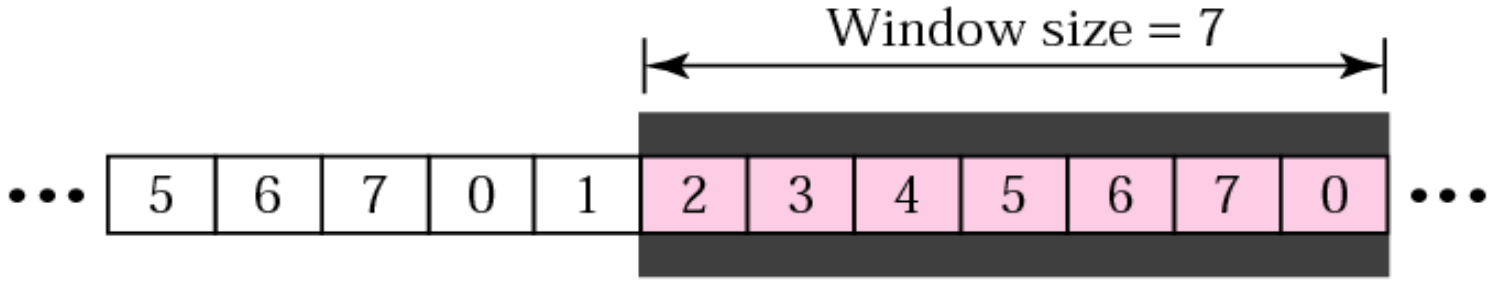
Continue..

- Frames from a sender are numbered sequentially.
- We need to set a limit since we need to include the sequence number of each frame in the header.
- If the header of the frame allows m bits for sequence numbers, the sequence numbers range from 0 to $2^m - 1$. For m = 3, sequence numbers are 0,1, 2, 3, 4, 5, 6, 7.

Sender Sliding Window – Go Back ARQ



a. Before sliding

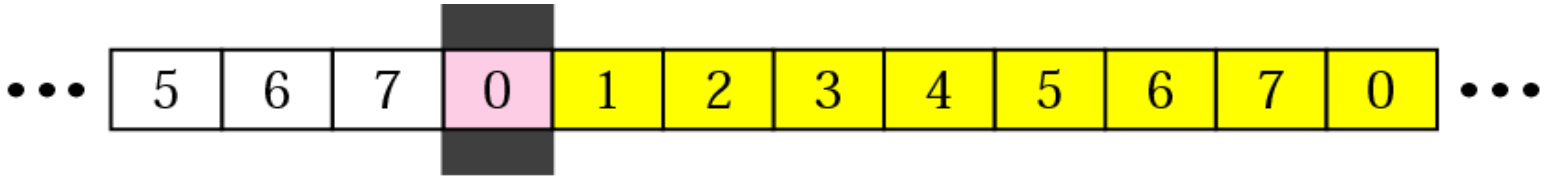


b. After sliding two frames

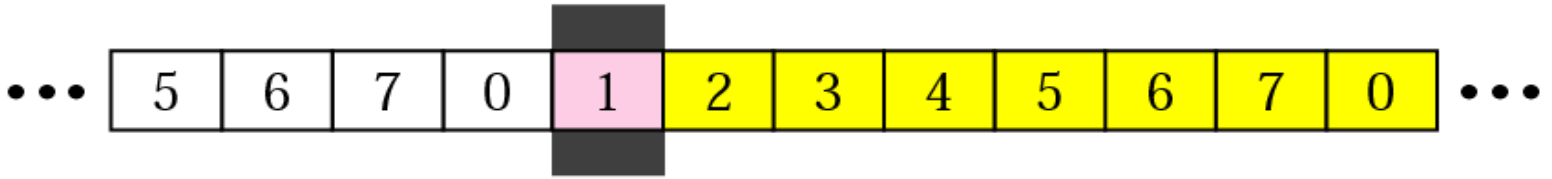
Continue..

- On the sending side, we use the concept of a window to hold the outstanding frames until they are acknowledged.
- The size of the window is at most 2^m -1 where m is the number of bits for the sequence number.
- The window slides to include new unsent frames when the correct ACKs are received.

Receiver Sliding Window – Go Back ARQ



a. Before sliding



b. After sliding

Continue...

- The size of the window at the receiving side is always 1 in this protocol.
- The receiver always looks for a specific frame to arrive in a specific order.
- Any frame arriving out of order is discarded and needs to be resent. The receiver is waiting for frame 0 in part a.

Acknowledgement

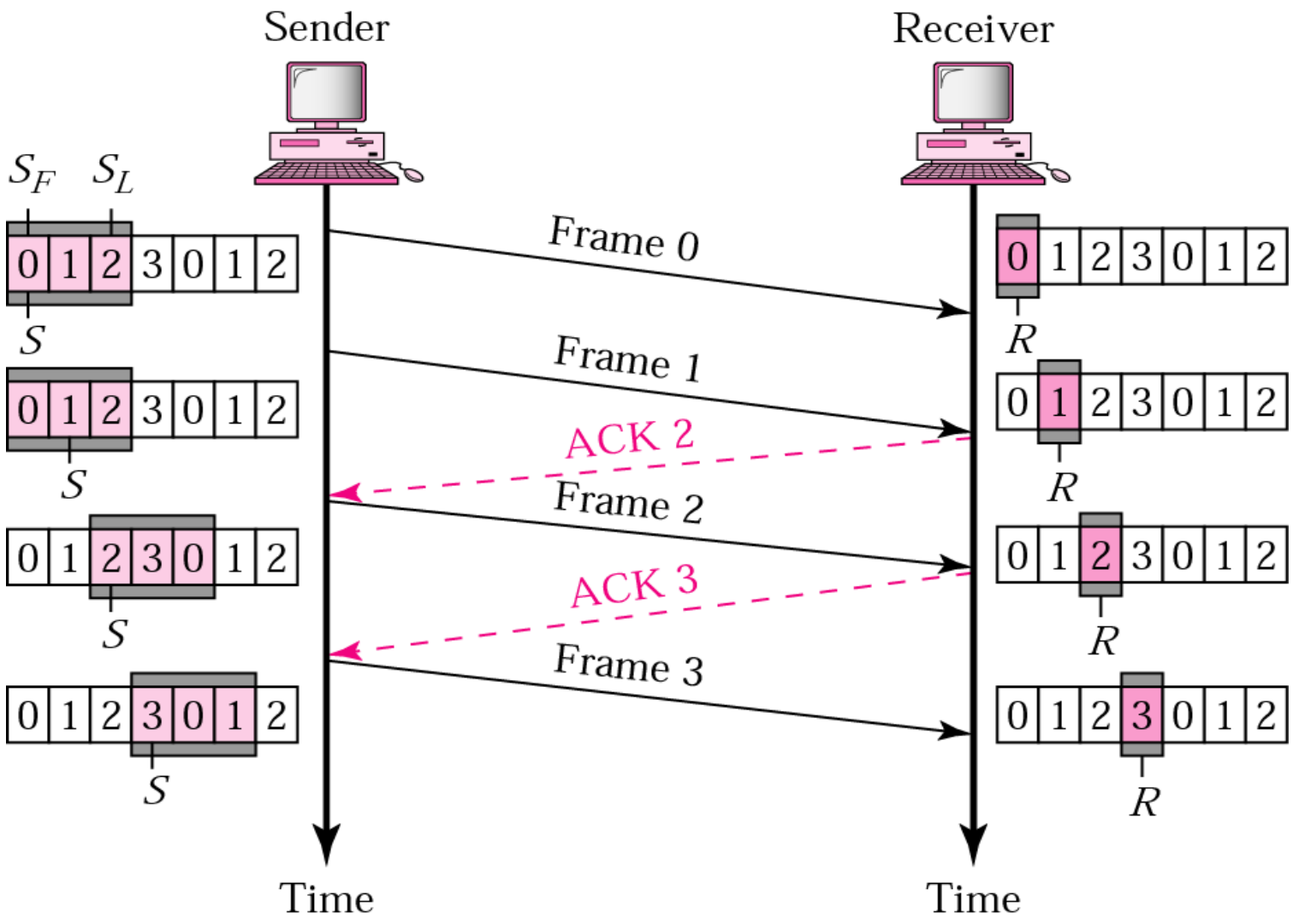
- Receiver sends positive ACK if a frame arrived safe and in order.
- If the frames are damaged/out of order, the receiver is silent and discards all subsequent frames until it receives the one it expects.
- The silence of the receiver causes the timer of the unacknowledged frame to expire.
- Then, the sender resends all frames, beginning with the one with the expired timer.

Continue..

- For example, suppose the sender has sent frame 6, but the timer for frame 3 expires (i.e., frame 3 has not been acknowledged), then the sender goes back and sends frames 3, 4, 5, and 6 again. Thus, it is called Go-Back-N-ARQ

- The receiver does not have to acknowledge each frame received; it can send one cumulative ACK for several frames.

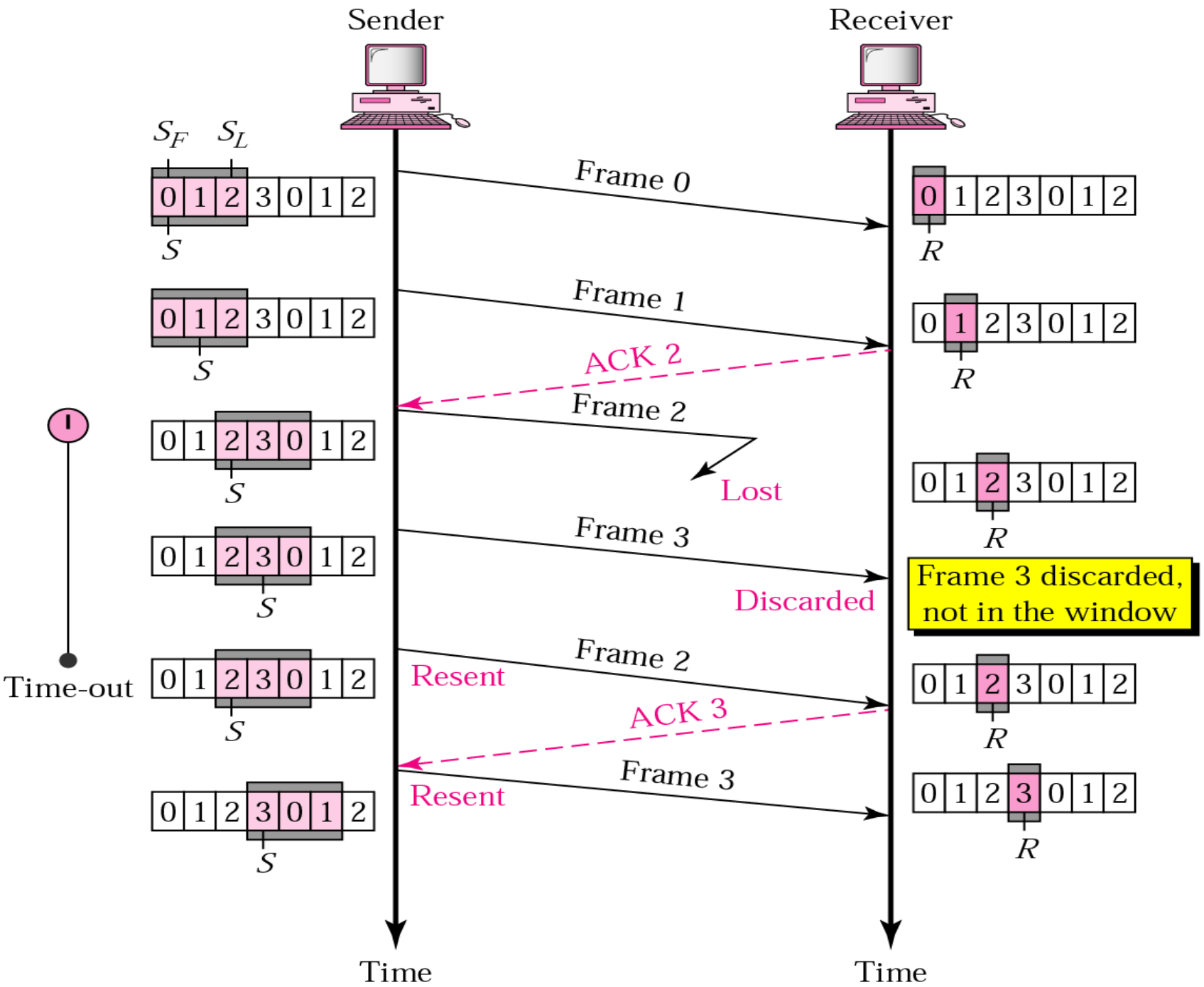
Go-Back-N ARQ, normal operation



Continue..

- The sender keeps track of the outstanding frames and updates the variables and windows as the ACKs arrive.

Go-Back-N ARQ, lost frame



Continue..

- Frame 2 is lost.
- When the receiver receives frame 3, it discards frame 3 as it expects frame 2 (according to the window).
- After the timer for frame 2 expires at the sender site, the sender sends frames 2 and 3. (go back to 2)

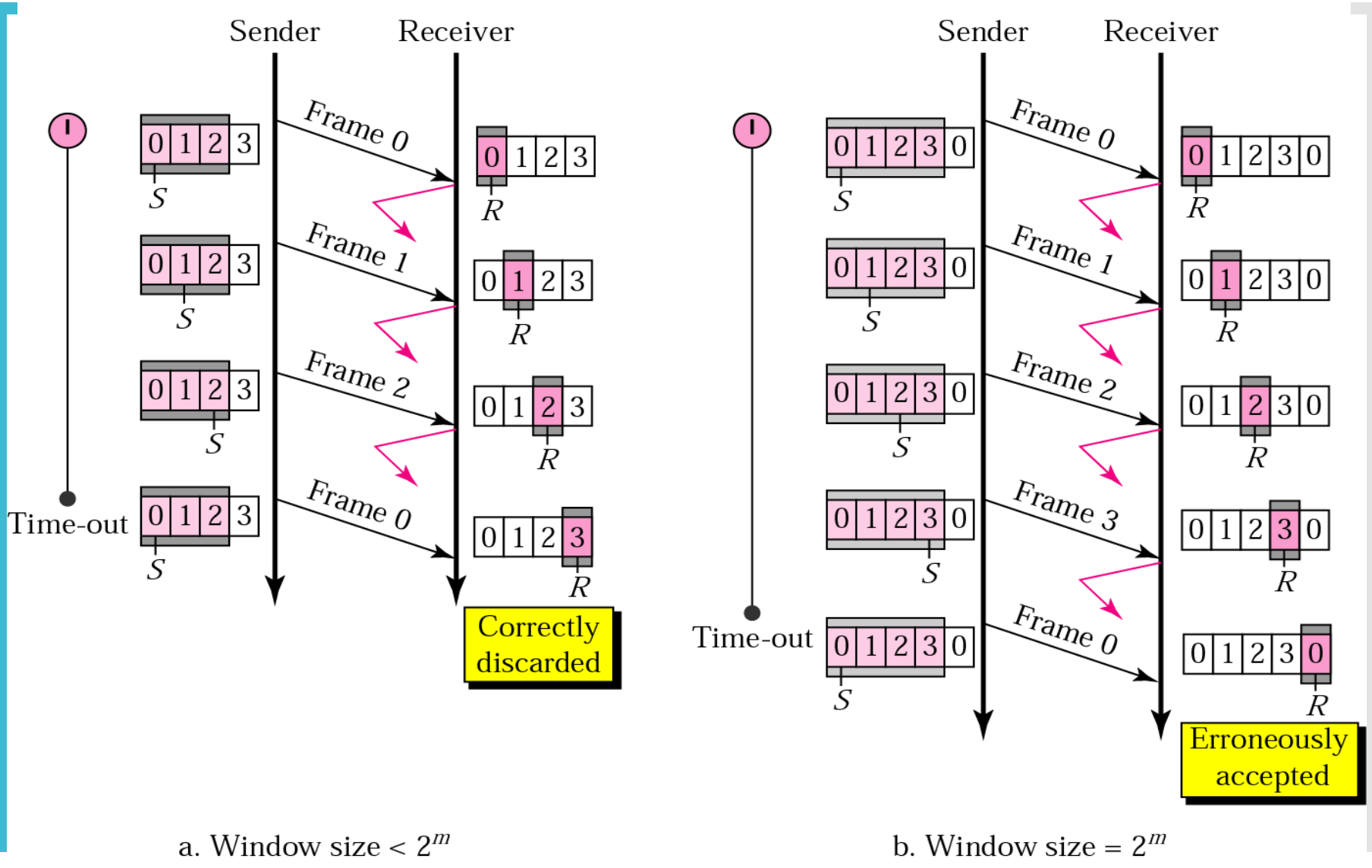
Go-Back-N ARQ, damaged/lost/ delayed ACK

- If an ACK is damaged/lost, we can have two situations:
- If the next ACK arrives before the expiration of any timer, there is no need for retransmission of frames because ACKs are cumulative in this protocol.
- If ACK1, ACK2, and ACk3 are lost, ACK4 covers them if it arrives before the timer expires.

Continue..

- If ACK_4 arrives after a time-out, the last frame and all the frames after that are resent.
- The receiver never resends an ACK (i.e., no timer is maintained for ACK) .
- A delayed ACK also triggers the resending of frames.

Go-Back-N ARQ, sender window size



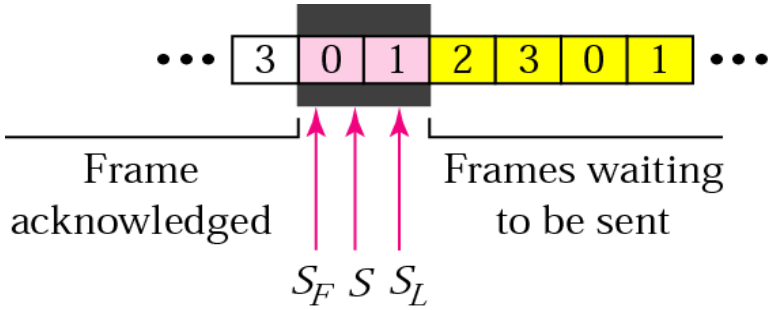
a. Window size < 2^m

b. Window size = 2^m

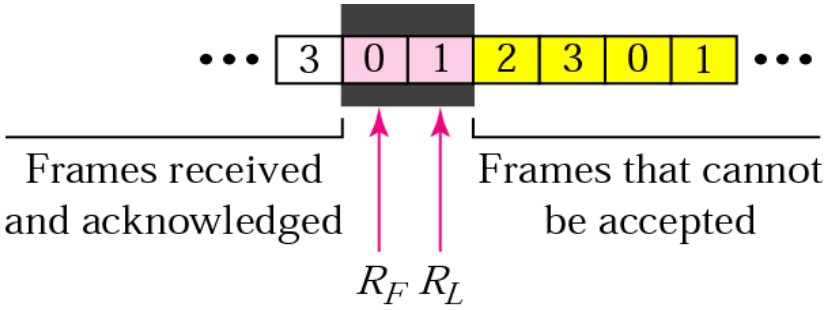
Continue..

- Size of the sender window must be less than 2^m. Size of the receiver is always 1. If m = 2, window size = $2^m - 1 = 3$.
- Fig compares a window size of 3 and 4.

Selective Repeat ARQ

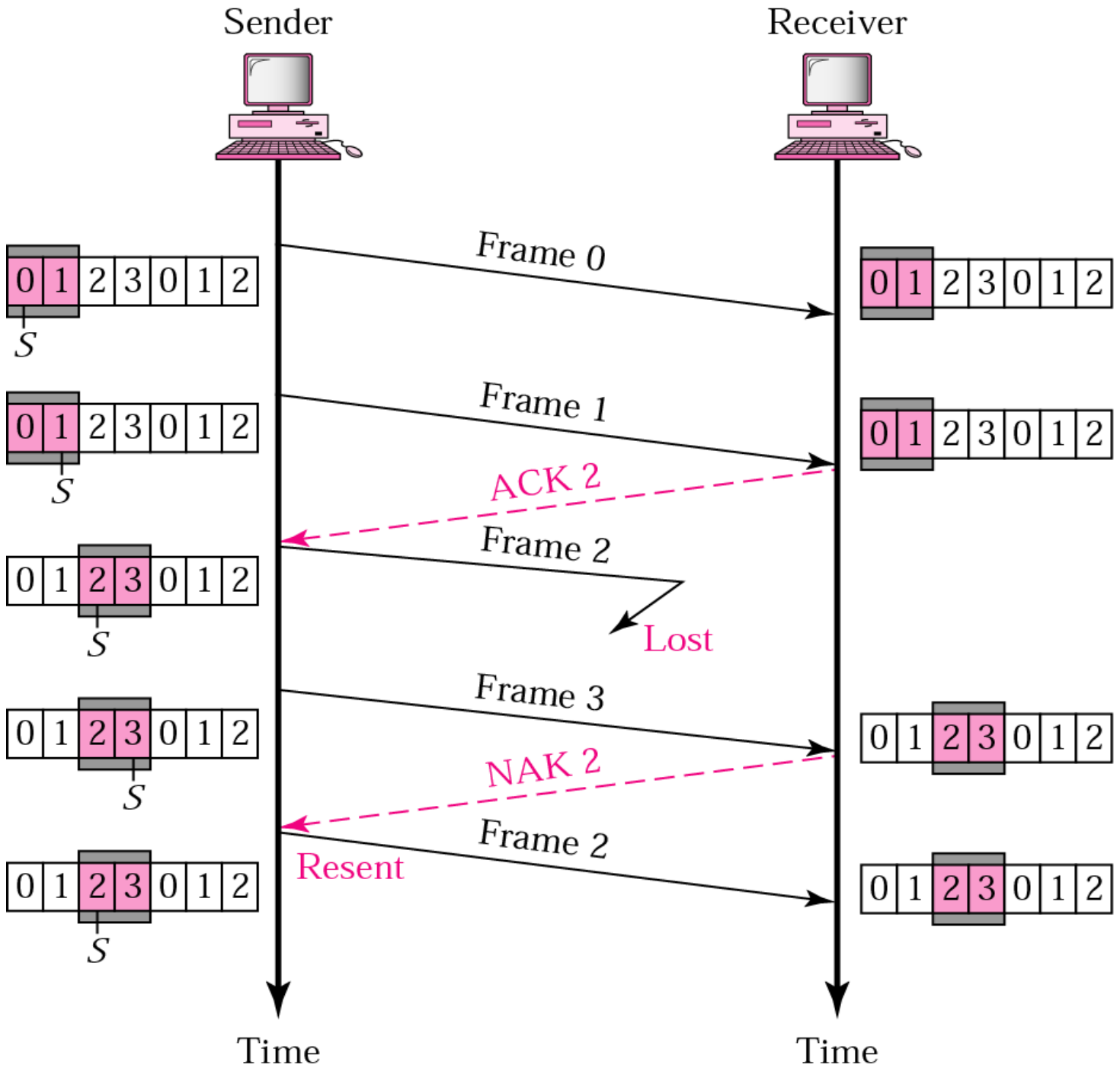


a. Sender window



b. Receiver window

Selective Repeat ARQ, lost frame



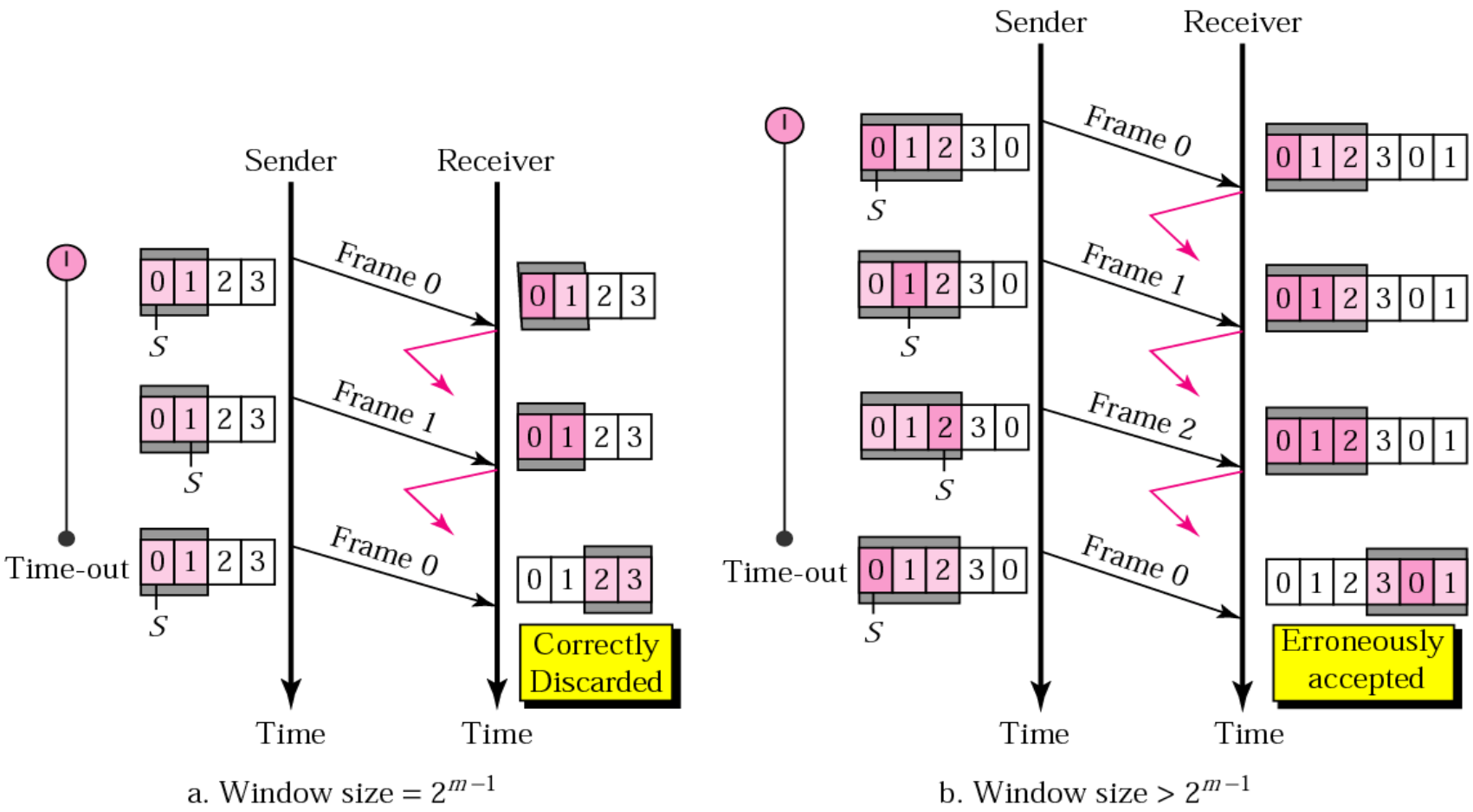
Continue..

- Frames 0 and 1 are accepted when received because they are in the range specified by the receiver window. Same for frame 3.
- The receiver sends a NAK2 to show that frame 2 has not been received, and then the sender resends only frame 2, which is accepted as it is in the range of the window.

Continue..

- ***In Selective Repeat ARQ, the sender and receiver window size must be at most one-half of 2^m.***

Selective Repeat ARQ, sender window size



Continue..

- Size of the sender and receiver windows must be at most one-half of 2^m. If m = 2, window size should be $2^m/2 = 2$.
- Fig compares a window size of 2 with a window size of 3. Window size is 3 and all ACKs are lost, sender sends duplicate of frame 0, window of the receiver expect to receive frame 0 (part of the window), so accepts frame 0, as the 1st frame of the next cycle – an **error**.

Continue..

Note:

A sliding window is used to make transmission more efficient as well as to control the flow of data so that the destination does not become overwhelmed with data. TCP's sliding windows are byte-oriented.

Transport layer (Syllabus)

- 3.1 Transport-layer services
- 3.2 Multiplexing and demultiplexing
- 3.3 Connectionless transport: UDP
- 3.4 Principles of reliable data transfer
- 3.5 Connection-oriented transport: TCP
  - segment structure
  - reliable data transfer
  - flow control
  - connection management
- 3.6 Principles of congestion control
- 3.7 TCP congestion control

Transmission Control Protocol (TCP)

TCP

Port Numbers

Services

Sequence Numbers

Segments

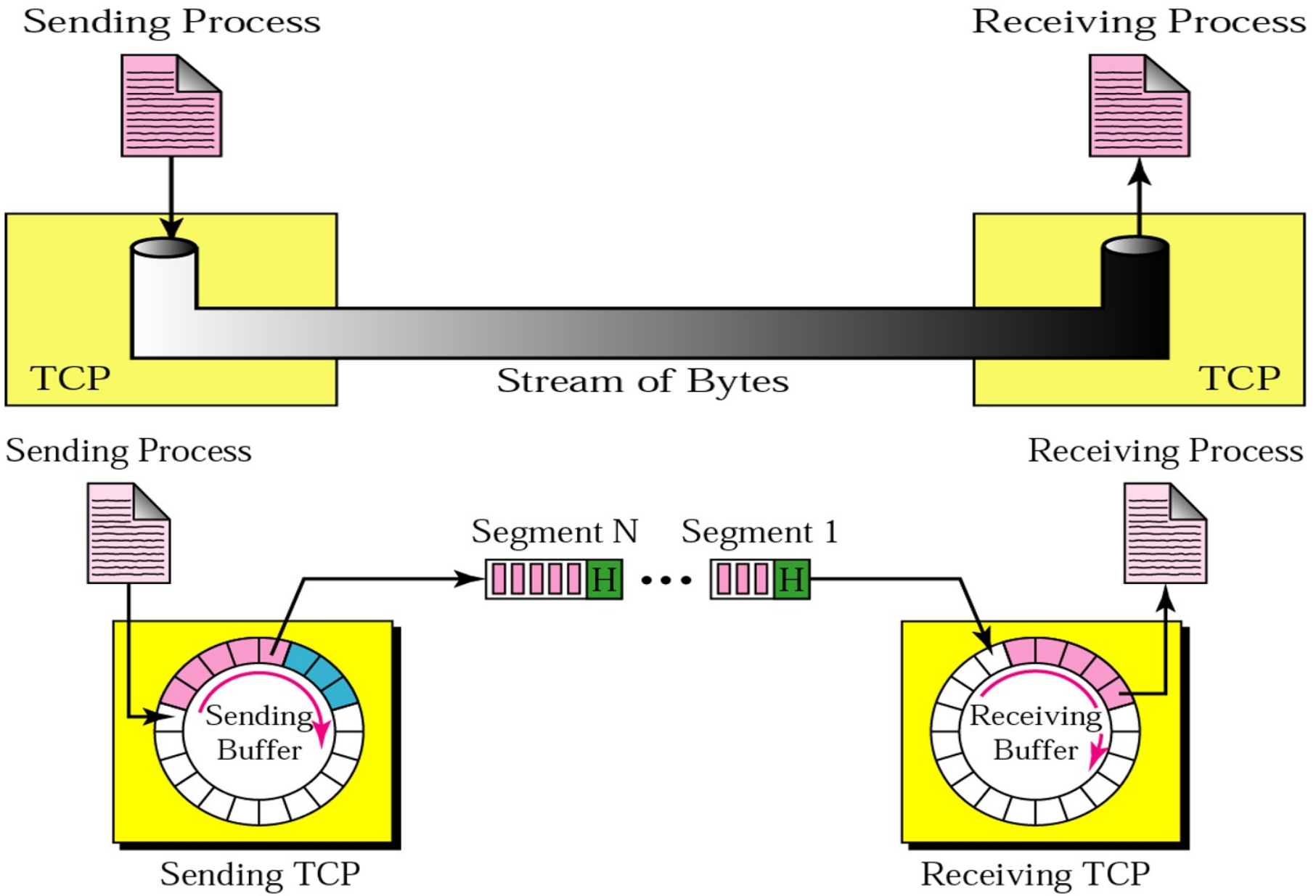
Connection

Transition Diagram

Flow and Error Control

Silly Window Syndrome

TCP (Stream Delivery, TCP Segment)



TCP
(Well Known Port)

Table 22.2 ***Well-known ports used by TCP***

Port	Protocol	Description
7	Echo	Echoes a received datagram back to the sender
9	Discard	Discards any datagram that is received
11	Users	Active users
13	Daytime	Returns the date and the time
17	Quote	Returns a quote of the day
19	Chargen	Returns a string of characters
20	FTP, Data	File Transfer Protocol (data connection)
21	FTP, Control	File Transfer Protocol (control connection)
23	TELNET	Terminal Network
25	SMTP	Simple Mail Transfer Protocol
53	DNS	Domain Name Server
67	BOOTP	Bootstrap Protocol
79	Finger	Finger
80	HTTP	Hypertext Transfer Protocol
111	RPC	Remote Procedure Call

TCP (Example)

Example 1

Imagine a TCP connection is transferring a file of 6000 bytes. The first byte is numbered 10010. What are the sequence numbers for each segment if data are sent in five segments with the first four segments carrying 1000 bytes and the last segment carrying 2000 bytes?

Solution

The following shows the sequence number for each segment:

Segment 1 ==> sequence number: 10,010 (range: 10,010 to 11,009)
Segment 2 ==> sequence number: 11,010 (range: 11,010 to 12,009)
Segment 3 ==> sequence number: 12,010 (range: 12,010 to 13,009)
Segment 4 ==> sequence number: 13,010 (range: 13,010 to 14,009)
Segment 5 ==> sequence number: 14,010 (range: 14,010 to 16,009)

TCP (Example)

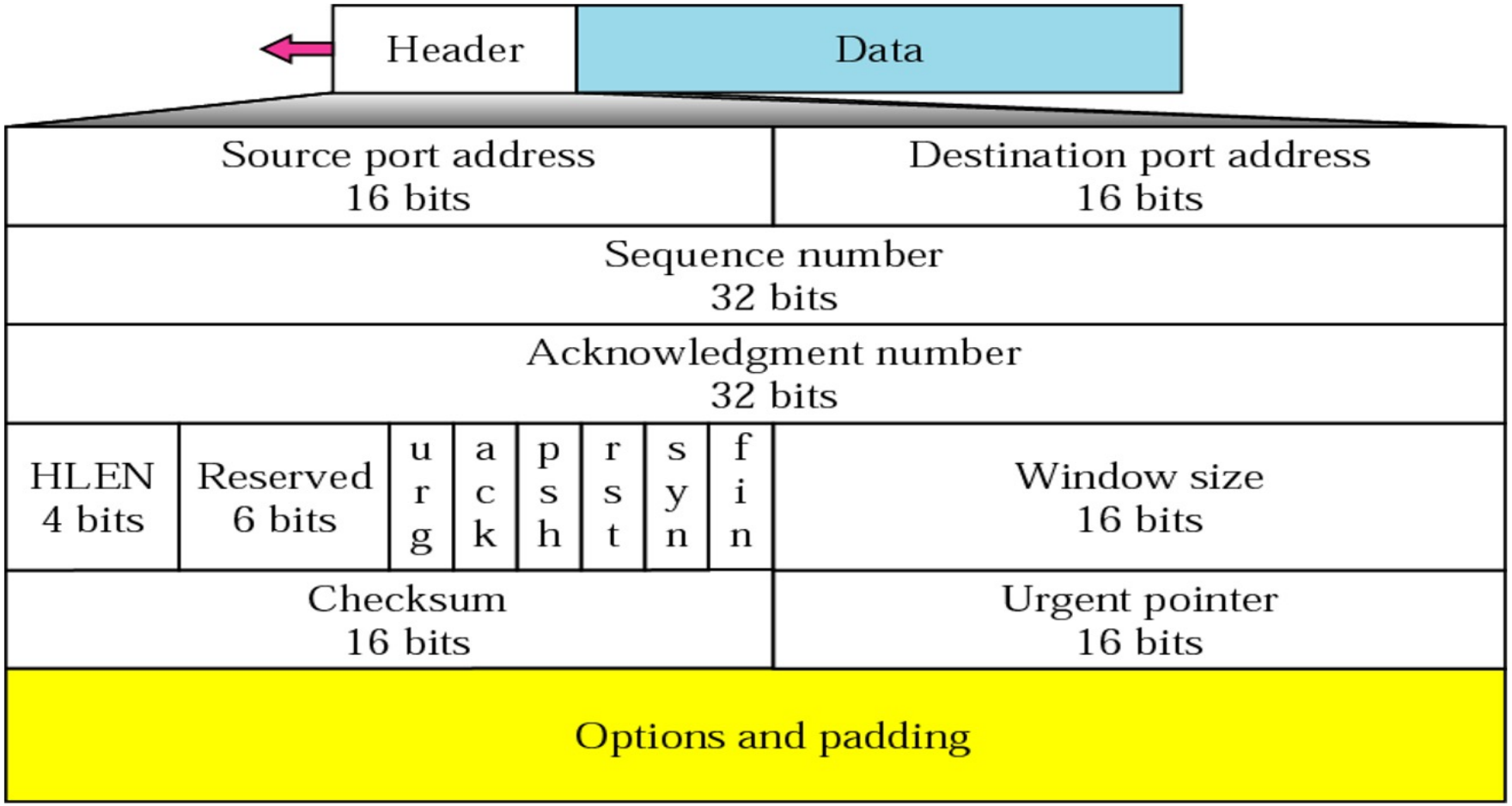
Note:

The bytes of data being transferred in each connection are numbered by TCP. The numbering starts with a randomly generated number.

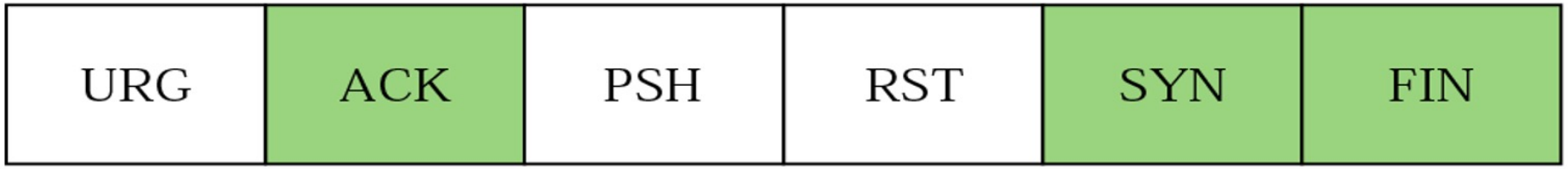
The value of the sequence number field in a segment defines the number of the first data byte contained in that segment.

The value of the acknowledgment field in a segment defines the number of the next byte a party expects to receive. The acknowledgment number is cumulative.

Transmission Control Protocol (TCP Segment Format)



URG: Urgent pointer is valid
ACK: Acknowledgment is valid
PSH: Request for push
RST: Reset the connection
SYN: Synchronize sequence numbers
FIN: Terminate the connection



TCP (Description of Flag Fields)

Table 22.3 ***Description of flags in the control field***

Flag	Description
URG	The value of the urgent pointer field is valid.
ACK	The value of the acknowledgment field is valid.
PSH	Push the data.
RST	The connection must be reset.
SYN	Synchronize sequence numbers during connection.
FIN	Terminate the connection.

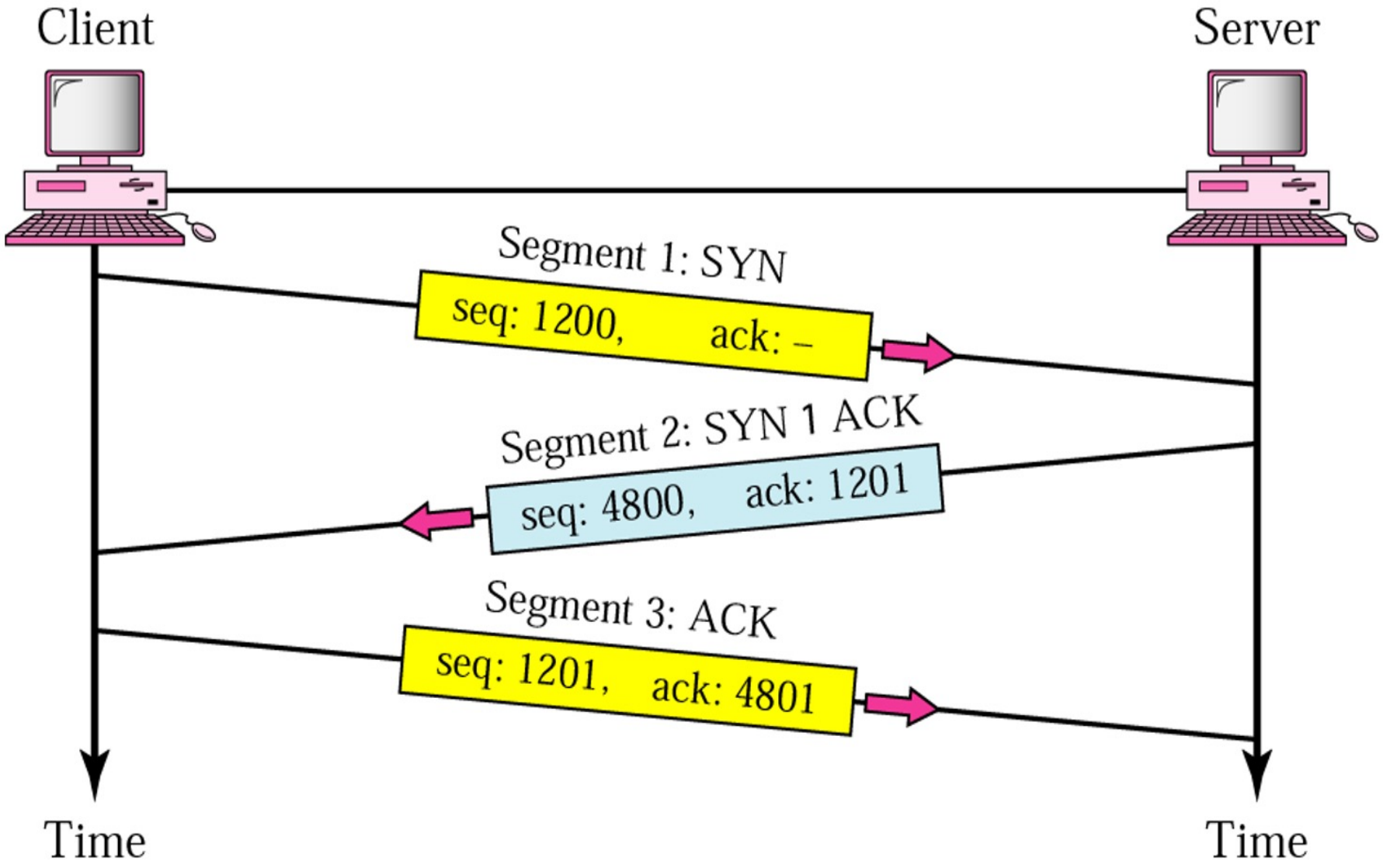
Difference between TCP and UDP Services

Description	*TCP*	*UDP*
Full Name	Transmission Control Protocol	User Datagram Protocol
Connection	TCP is a connection-oriented protocol.	UDP is a connectionless protocol.
Function	A point to point connection is established between client and server before sending message.	A point to point connection is not established before sending messages.
Usage	TCP is suited for applications that require high reliability, and transmission time is relatively less critical.	UDP is suitable for applications that need fast, efficient transmission, such as games.
Reliability	There is absolute guarantee that the data transferred remains intact and arrives in the same order in which it was sent.	There is no guarantee that the messages or packets sent would reach at all.
Use by other protocols	HTTP, HTTPs, FTP, SMTP, Telnet	DNS, DHCP, SNMP, RIP, VOIP
Ordering of data packets	TCP rearranges data packets in the order specified.	UDP has no inherent order as all packets are independent of each other.

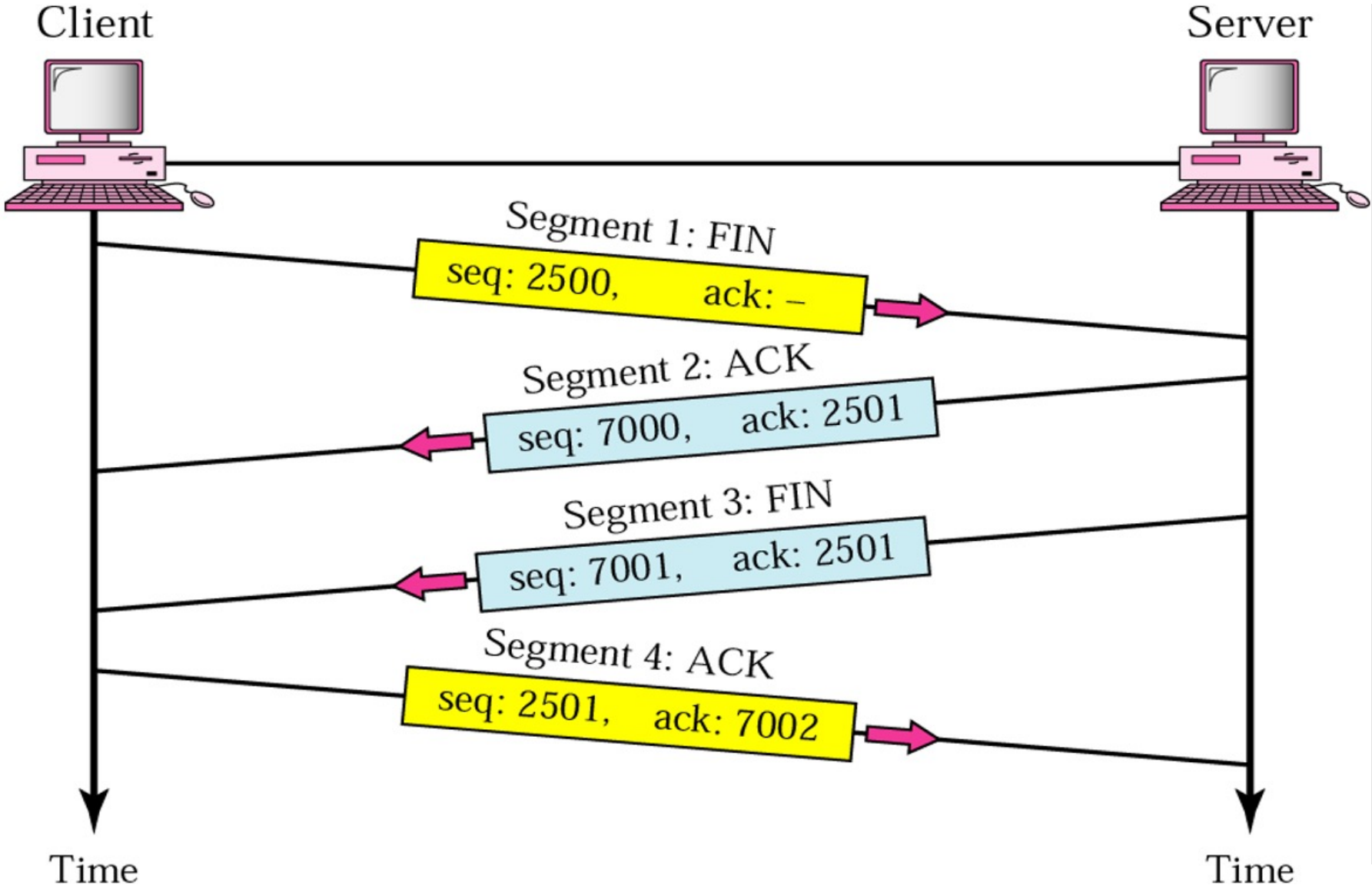
Continue..

Speed of transfer	The speed for TCP is slower than UDP.	UDP is faster because there is no error-checking for packets.
Header Size	TCP header size is 20 bytes	UDP Header size is 8 bytes.
Data Flow Control	TCP does Flow Control.	UDP does not have an option for flow control.
Error Checking	TCP does error checking	UDP does error checking, but no recovery options.
Acknowledgement	Acknowledgement segments	No Acknowledgment
Handshake	SYN, SYN-ACK, ACK	No handshake

TCP (Three Step Connection Establishment)



TCP (Four Step Connection Establishment)



TCP
(Four Step Connection Establishment)

Note:

In TCP, the sender window size is totally controlled by the receiver window value (the number of empty locations in the receiver buffer). However, the actual window size can be smaller if there is congestion in the network.

Principles of congestion control

***Congestion*:**

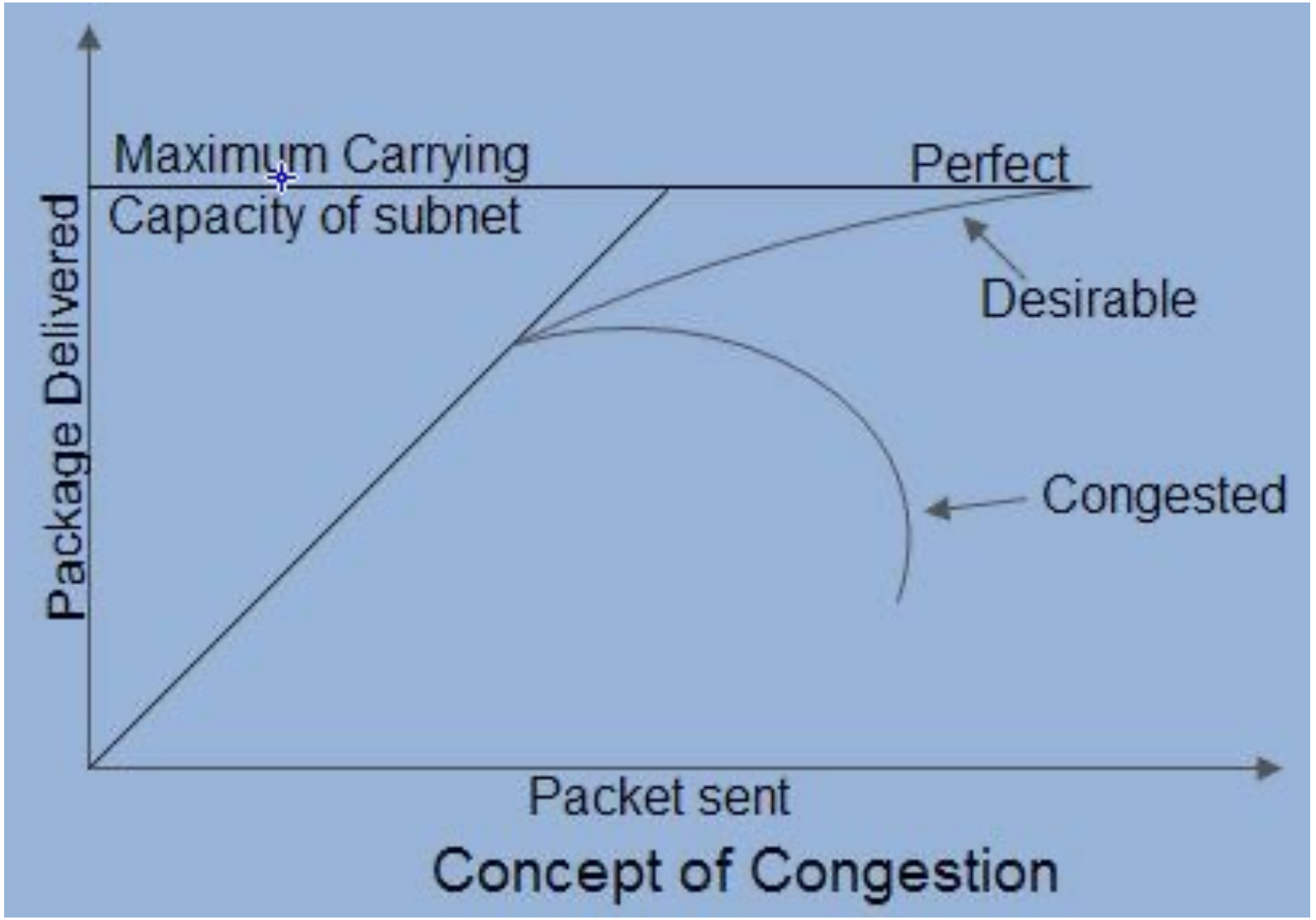
- Informally: "Too many sources are sending too much data too fast for the *network* to handle."
- Different from flow control!
- **Manifestations:**
  - lost packets (buffer overflow at routers)
  - long delays (queueing in router buffers)
- a top-10 problem!

Continue..

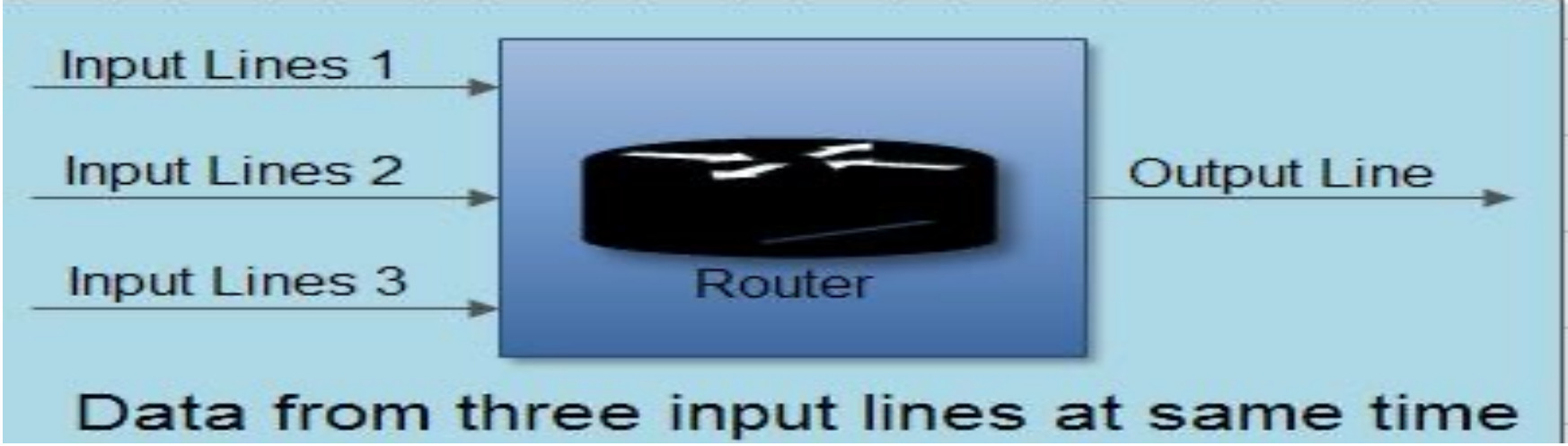
- Congestion is a situation in Communication Networks in which too many packets are present in a part of the subnet, and performance degrades.

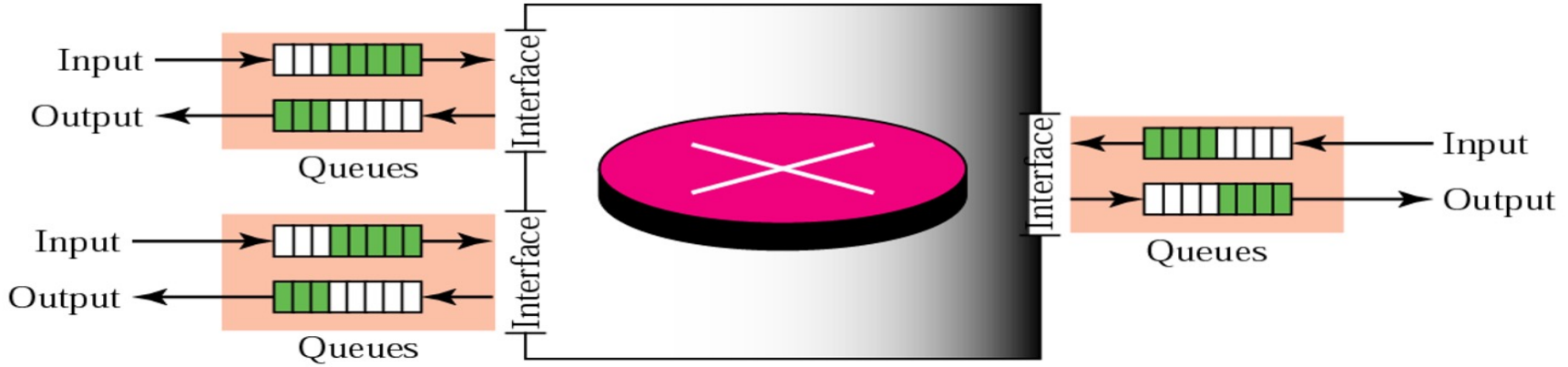
Congestion in a network may occur when the load on the network *(i.e.,* the number of packets sent to the network) is greater than the capacity of the network *(i.e.,* the number of packets a network can handle.).

Continue..


Maximum Carrying
Capacity of subnet
Perfect
Desirable
Congested
Package Delivered
Packet sent
Concept of Congestion

Causing of Congestion:





TCP assumes that the cause of a lost segment is due to congestion in the network.

Continue..

- The various causes of congestion in a subnet are:
- The input traffic rate exceeds the capacity of the output lines.
- If suddenly, a stream of packet start arriving on three or four input lines, and all need the same output line.
- In this case, a queue will be built up. The packet will be lost if there is insufficient memory to hold all the packets.
- Increasing the memory to unlimited size does not solve the problem.

Continue..

- This is because, by the time packets reach the front of the queue, they have already timed out (as they waited in the queue).
- When the timer goes off source transmits duplicate packet that are also added to the queue.
- Thus, the same packets are added again and again, increasing the load all the way to the destination.

Continue..

- Congestion in a subnet can occur if the processors are slow.
- Slow speed CPU at routers will perform the routine tasks such as queuing buffers, updating table etc slowly.
- As a result of this, queues are built up even though there is excess line capacity.

Continue…

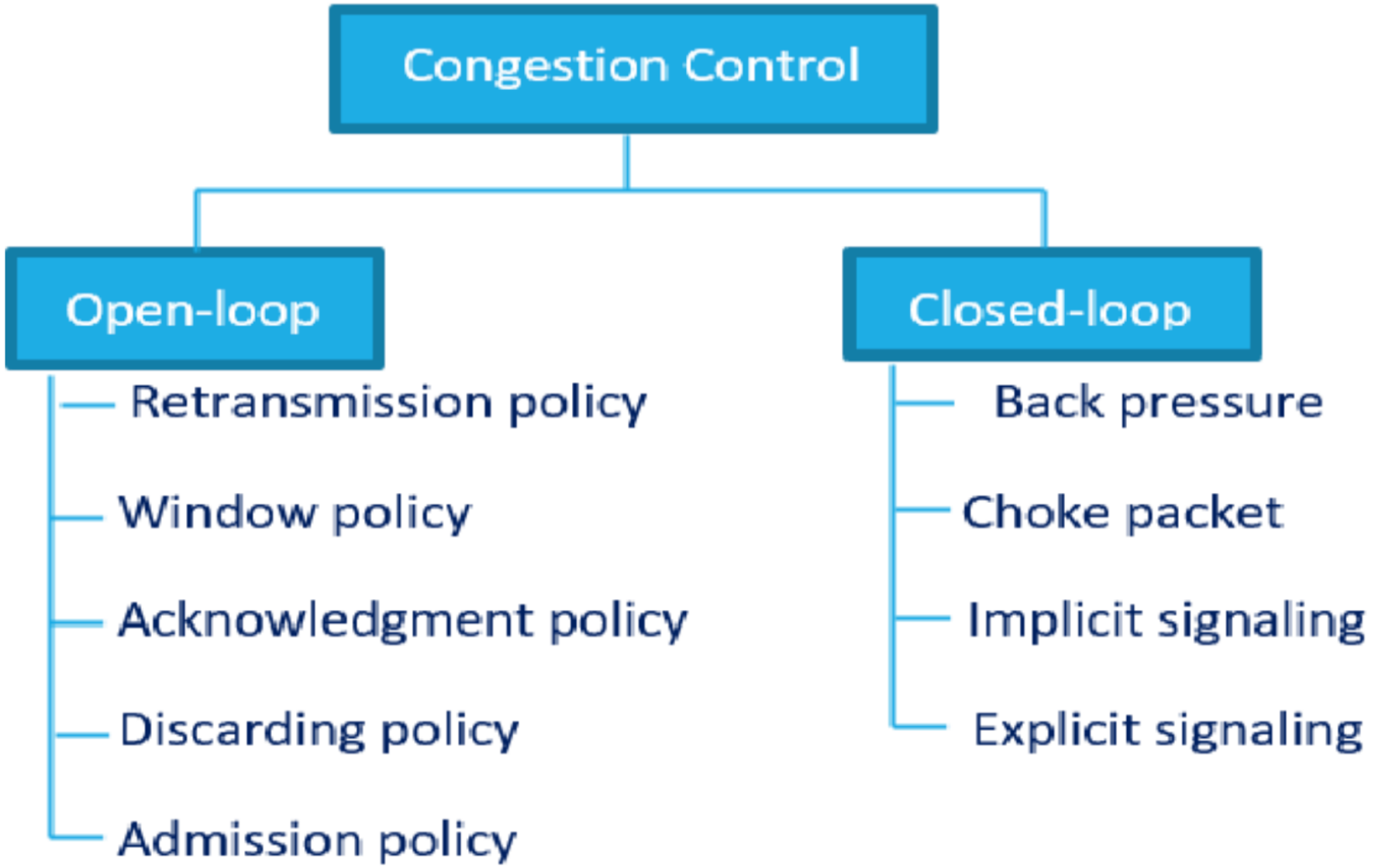
- Congestion is also caused by slow links. This problem will be solved when high speed links are used.
- But it is not always the case.
- Sometimes increase in link bandwidth can further deteriorate the congestion problem as higher speed links may make the network more unbalanced.

How to correct the Congestion Problem:

- Congestion Control refers to techniques and mechanisms that can either prevent congestion before it happens or remove congestion after it has happened.

- Congestion control mechanisms are divided into two categories: one prevents congestion from happening, and the other removes congestion after it has occurred.

Congestion Control Methods



Congestion Control Methods

- These two categories are:
- 1. Open loop
- 2. Closed loop

Congestion Control Methods

Open Loop Congestion Control

- In this method, policies are used to prevent the congestion before it happens.
- Congestion control is handled either by the source or by the destination.

The various methods used for open loop congestion control are:

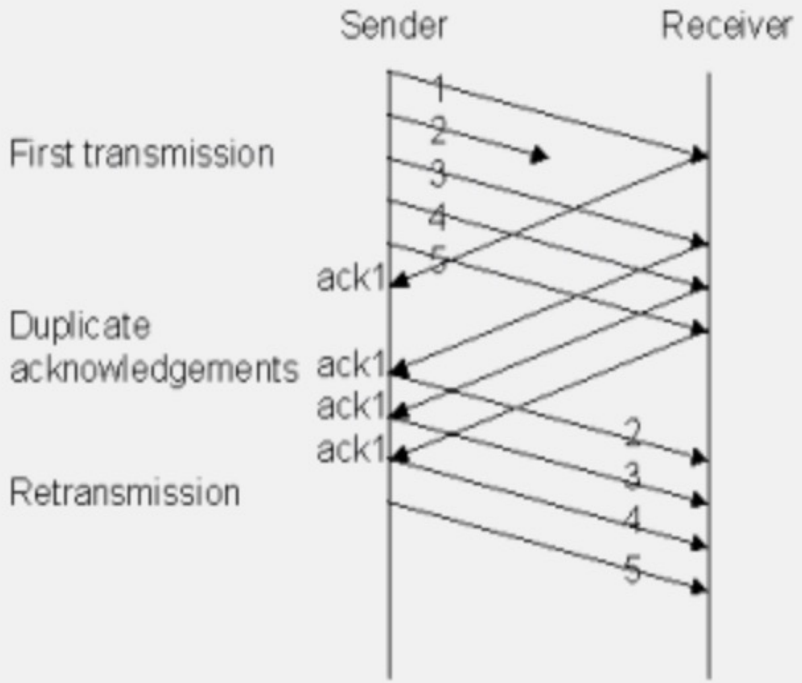
Open Loop Congestion Control

Retransmission Policy

- The sender retransmits a packet, if it feels that the packet it has sent is lost or corrupted.
- However, retransmission is a general way to increase the congestion in the network. But we need to implement a good retransmission policy to prevent congestion.
- The retransmission policy and the retransmission timers need to be designed to optimize efficiency and, at the same time, prevent congestion.

Open Loop Congestion Control

Retransmission Policy



- For every packet received, the recipient returns an Ack
- Recipient sends duplicate Acks if a packet is lost
- Sender re-transmits lost packets

Open Loop Congestion Control

Window Policy

- To implement a window policy, the selective repeat window method is used for congestion control.
- The selective Repeat method is preferred over the Go-back-N window as in the Go-back-N method, when the timer for a packet times out, several packets are resent, although some may have arrived safely at the receiver.
- Thus, this duplication may make congestion worse.
- The selective repeat method sends only the specific lost or damaged packets.

Open Loop Congestion Control

Acknowledgement Policy

- The acknowledgment policy imposed by the receiver may also affect congestion.
- If the receiver does not acknowledge every packet it receives, it may slow down the sender and help prevent congestion.
- Acknowledgments also add to the traffic load on the network. Thus, by sending fewer acknowledgments, we can reduce the load on the network.

Continue..

- To implement it, several approaches can be used:
- 1. A receiver may send an acknowledgment only if it has a packet to be sent.
- 2. A receiver may send an acknowledgment when a timer expires.
- 3. A receiver may also decide to acknowledge only *N* packets at a time.

Open Loop Congestion Control

Discarding Policy

- A router may discard less sensitive packets when congestion is likely to happen.
- Such a discarding policy may prevent congestion and, at the same time may not harm the integrity of the transmission.

Open Loop Congestion Control

Admission Policy

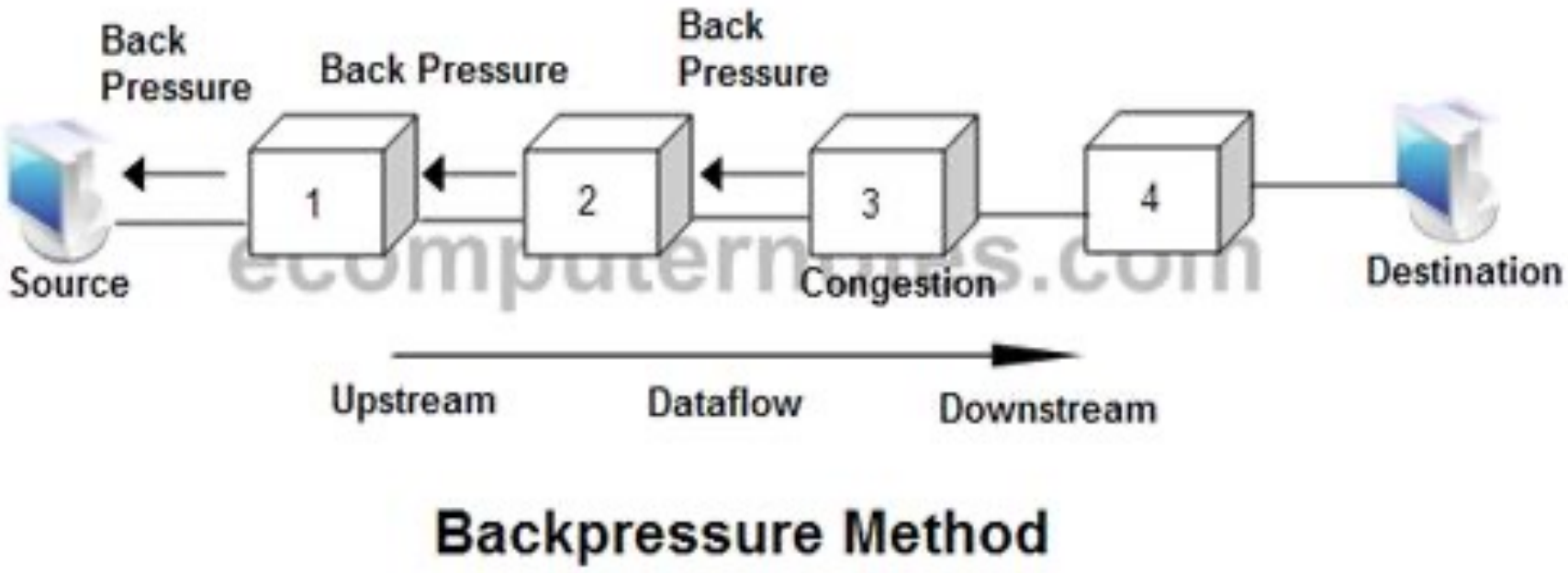
- An admission policy, which is a quality-of-service mechanism, can also prevent congestion in virtual circuit networks.
- Switches in a flow first check the resource requirement of a flow before admitting it to the network.

Closed Loop Congestion Control

- Closed loop congestion control mechanisms try to remove the congestion after it happens.
- The various methods used for closed-loop congestion control are:

Closed Loop Congestion Control

Backpressure



Backpressure Method

Continue..

- Backpressure is a node-to-node congestion control that starts with a node and propagates in the opposite direction of data flow.
- The backpressure technique can be applied only to virtual circuit networks. In such a virtual circuit, each node knows the upstream node from which a data flow is coming.

Continue..

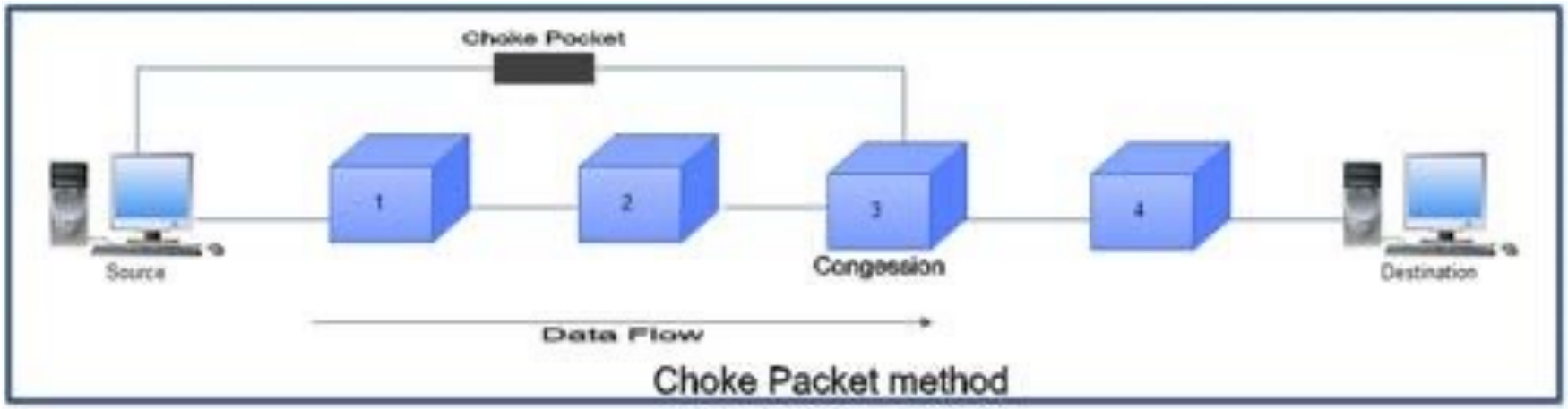
- In this method of congestion control, the congested node stops receiving data from the immediate upstream node or nodes.
- This may cause the upstream node on nodes to become congested, and they, in turn, reject data from their upstream node or nodes.

Continue..

- As shown in Fig node 3 is congested, and it stops receiving packets and informs its upstream node 2 to slow down. Node 2 in turn may be congested and informs node 1 to slow down.
- Now node 1 may create congestion and inform the source node to slow down. In this way, the congestion is alleviated.
- Thus, the pressure on node 3 is moved backward to the source to remove the congestion.

Closed Loop Congestion Control

Choke Packet



Choke Packet method

Continue..

- In this method of congestion control, a congested router or node sends a special type of packet called a choke packet to the source to inform it about the congestion.
- Here, congested node does not inform its upstream node about the congestion as in backpressure method.
- In choke packet method, congested node sends a warning directly to the source station *i.e.* the intermediate nodes through which the packet has travelled are not warned.

Closed Loop Congestion Control

Implicit Signalling

- In implicit signalling, there is no communication between the congested node or nodes and the source.
- The source guesses that there is congestion somewhere in the network when it does not receive any acknowledgment.

Continue..

- Therefore, the delay in receiving an acknowledgment is interpreted as congestion in the network.
- On sensing this congestion, the source slows down.
- This type of congestion control policy is used by TCP.

Closed Loop Congestion Control

Explicit Signalling

- In this method, the congested nodes explicitly send a signal to the source or destination to inform about the congestion.
- Explicit signalling is different from the choke packet method. In choke packed method, a separate packet is used for this purpose, whereas in the explicit signalling method, the signal is included in the packets that carry data.
- Explicit signalling can occur in either the forward direction or the backward direction.