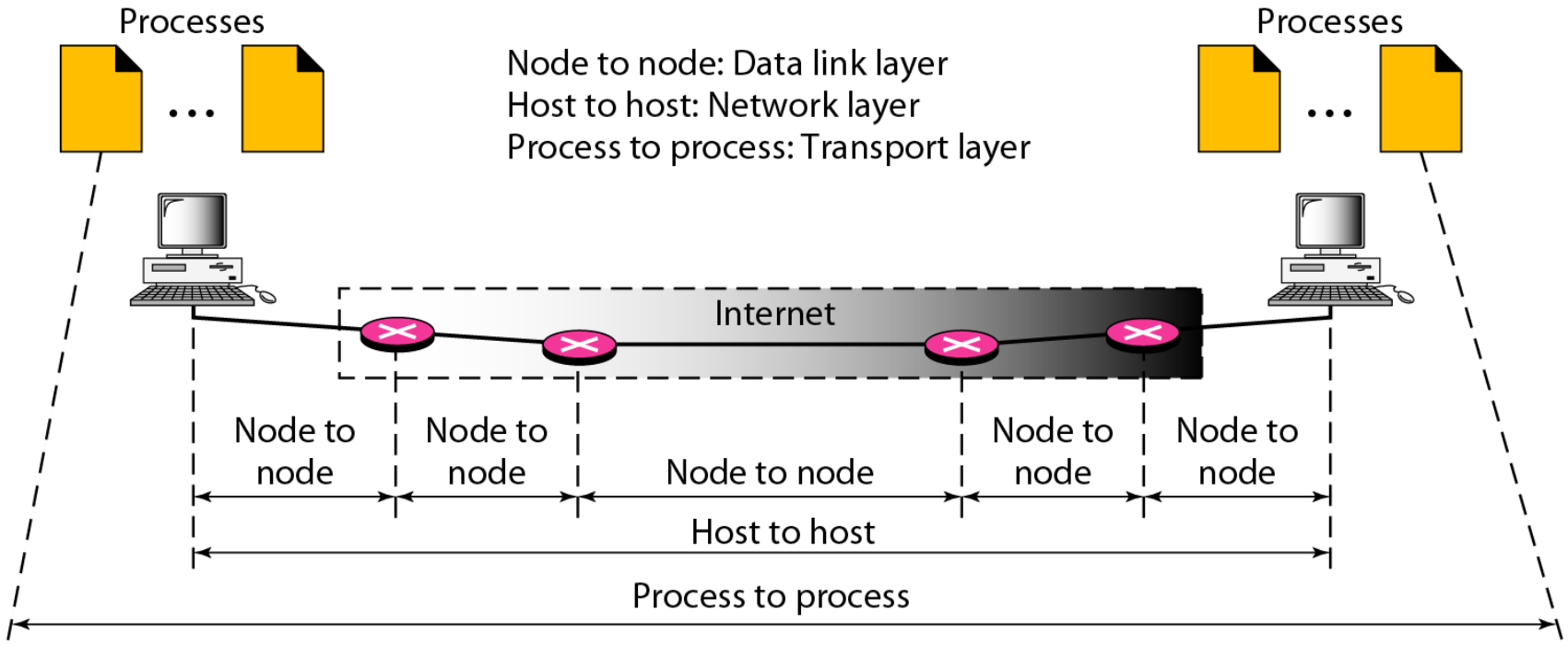
TRANSPORT LAYER

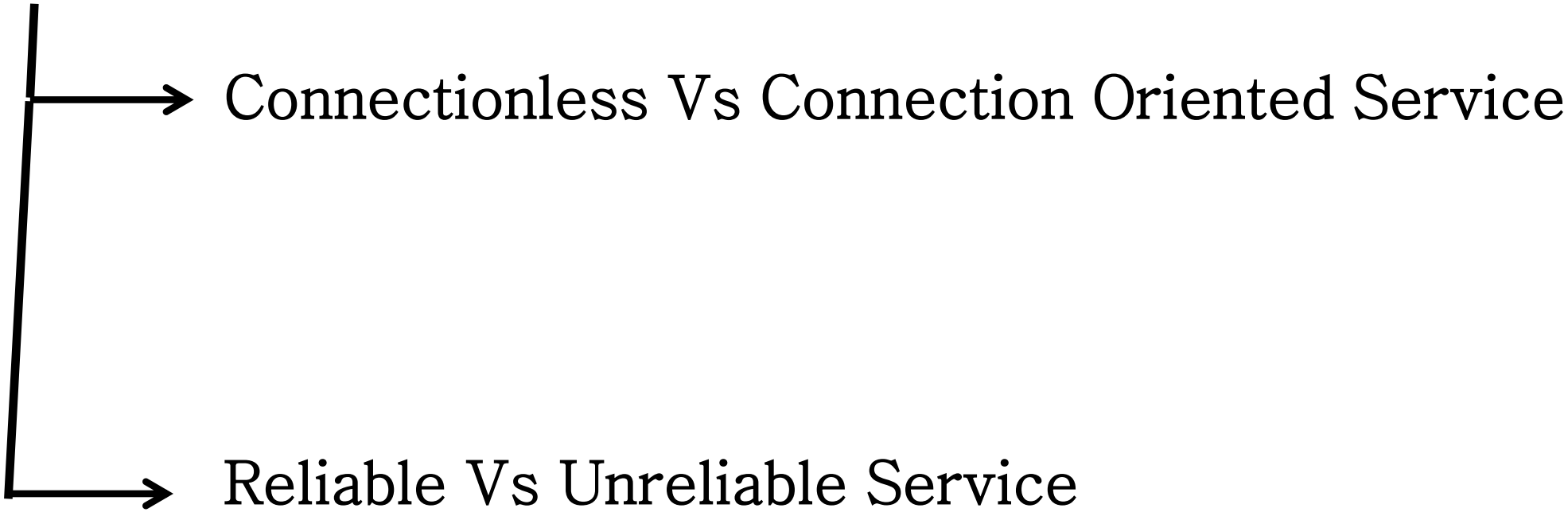
INTRODUCTION

- The transport layer is responsible for message delivery from process running in source computer to process running destination computer (PROCESS TO PROCESS DELIVERY))
- It provides *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
  - Recieve side: reassembles segments into messages, passes to app layer
- more than one transport protocol available to apps
  - Internet: TCP and UDP

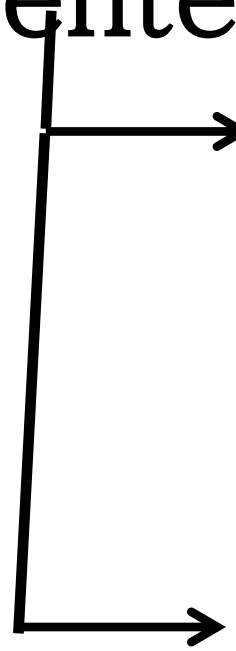
Types of data deliveries: Internet Stack



TRANSPORT LAYER SERVICES



Connectionless Vs Connection Oriented Service



Connectionless Service

- Without Connection Establishent or Connection Release
- Packets may be lost or delay
- No Acknowledgement

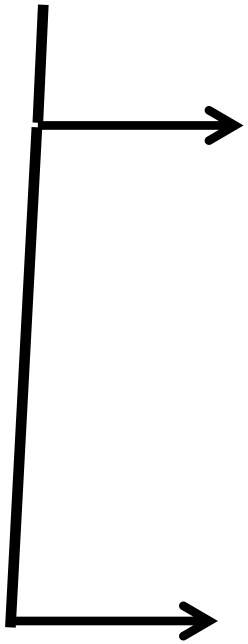
For Eg: UDP

Connection Oriented Service

- Connection Establishent or Connection Release
- Packets neither be lost
- Acknowledgement

For Eg: TCP

Reliable Vs Unreliable Service



Reliable

- Predefined Flow Control & Error Control
- Slow & More Complex

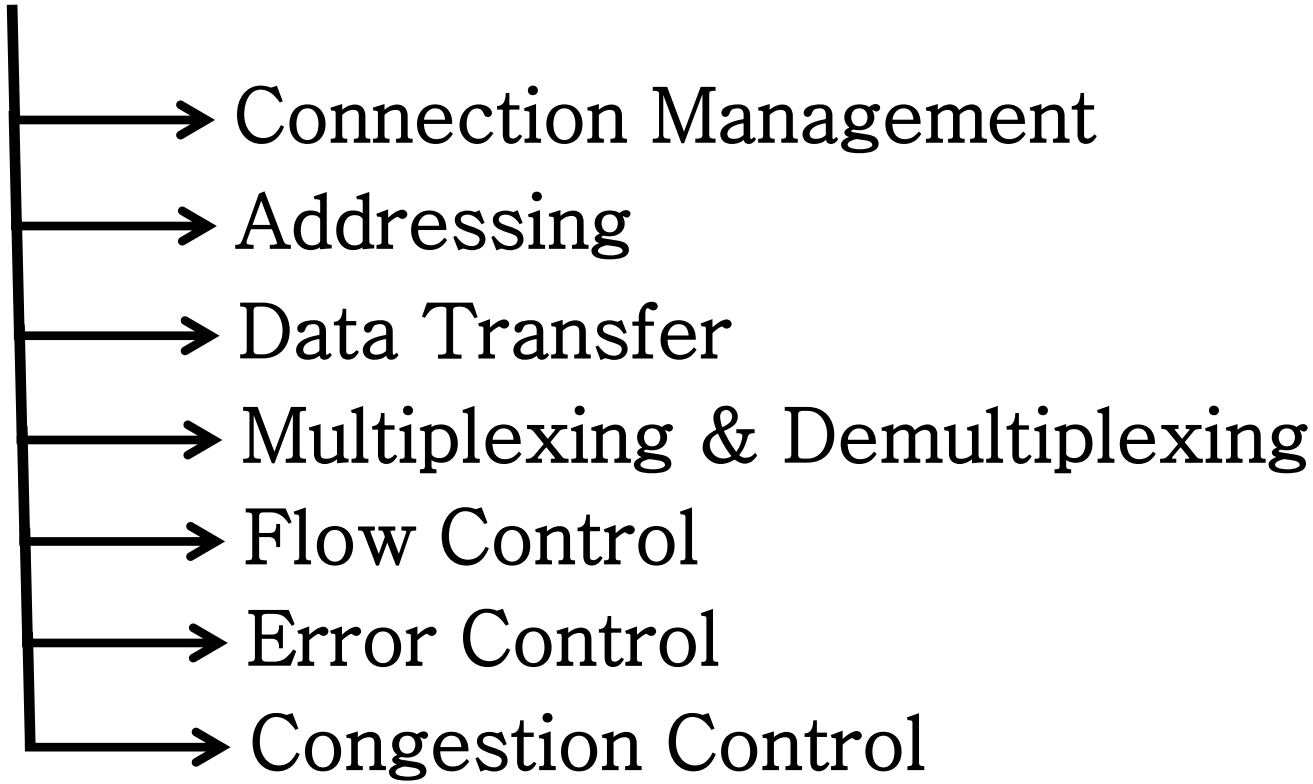
For Eg: TCP

Unreliable

- Own Flow & Error control mechanism as required
- Fast & Easy to implement

For Eg: UDP

TRANSPORT LAYER DESIGN ISSUE

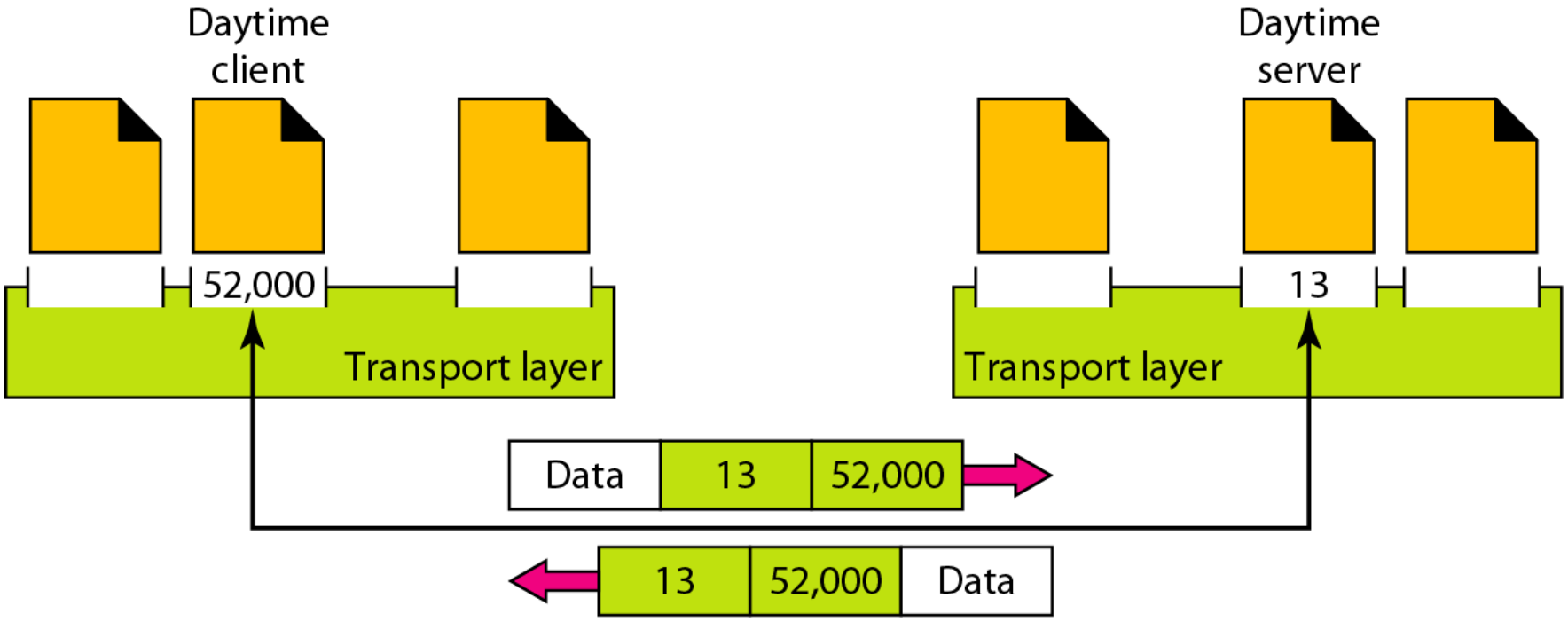


1. **Connection Management: Establishing, Maintaining & Releasing Connection:**
   The transport layer establishes, maintains & releases end-to-end transport connection on the request of upper layers. Establishing a connection involves allocation of buffers for storing user data, synchronizing the sequence numbers of packets etc. A connection is released at the request of upper layer.

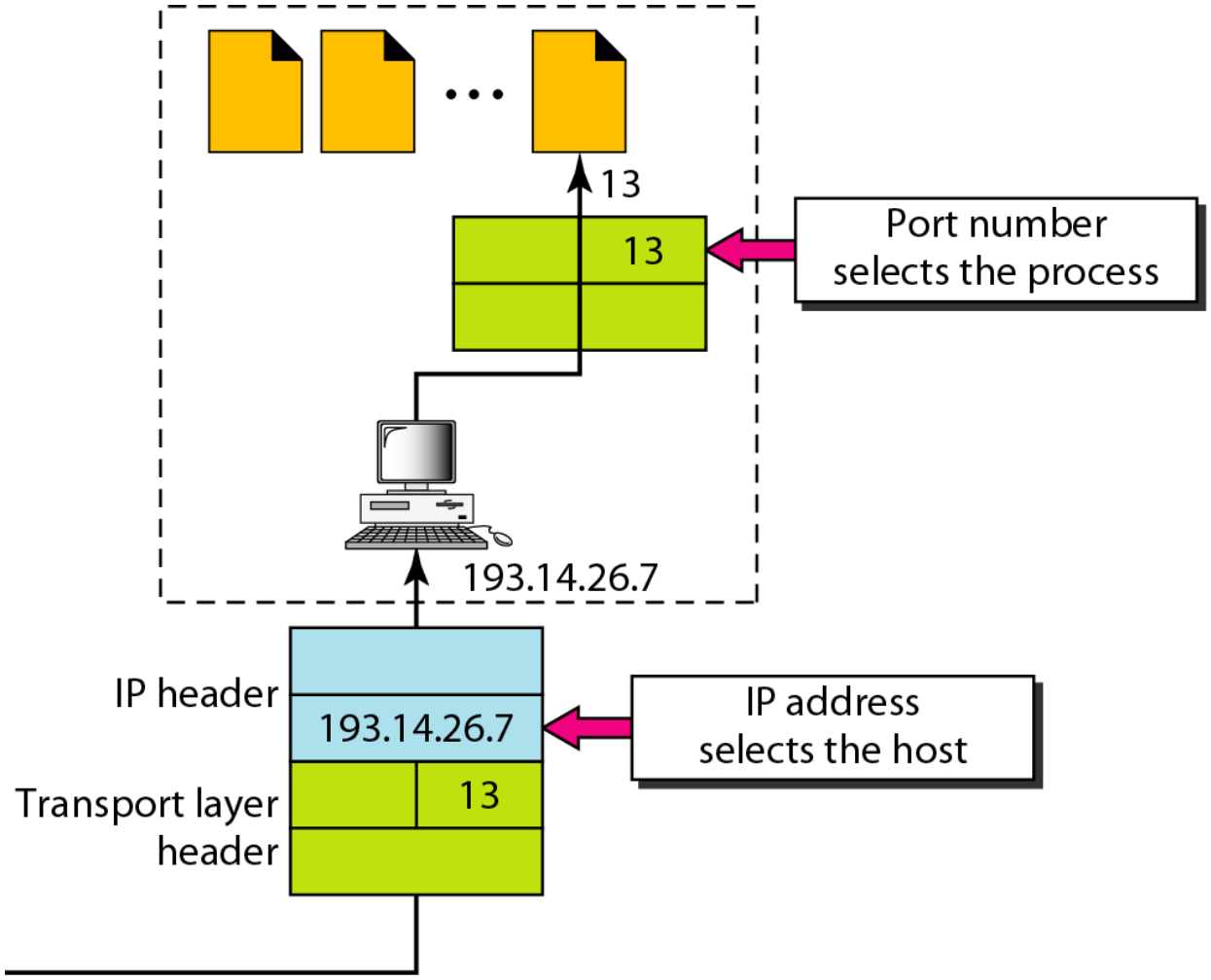
2. **Addressing:** In order to deliver the message from one process to another, an addressing scheme is required. Several process may be running on a system at a time. In order to identify the correct process out of the various running processes, transport layer uses an addressing scheme involving the concept of port numbers. Each process has a specific port number.
   In order to provide communication between two different processes on different networks, both IP address and port number, i.e. socket address is required. Socket address is a combination of IP address and port number.

Addressing

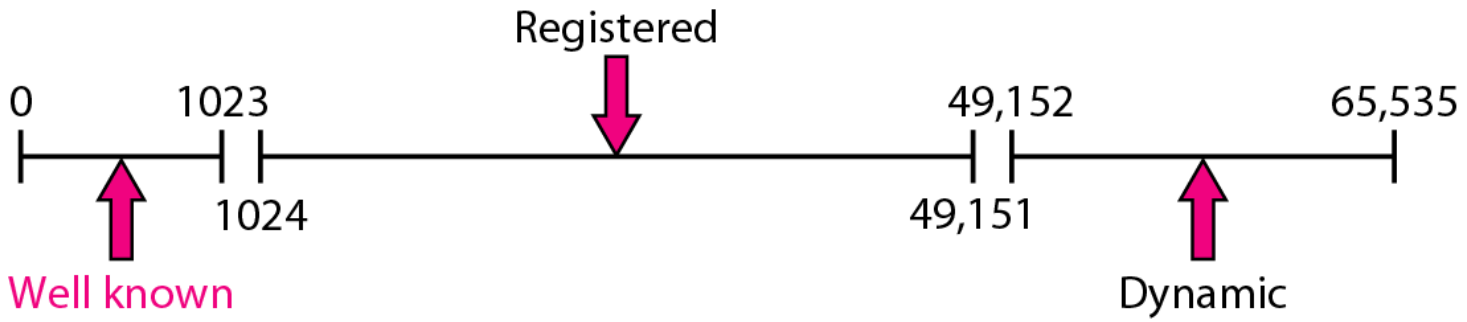
- Using Port address (16 bit)
- Ranges from 0 to 65,535



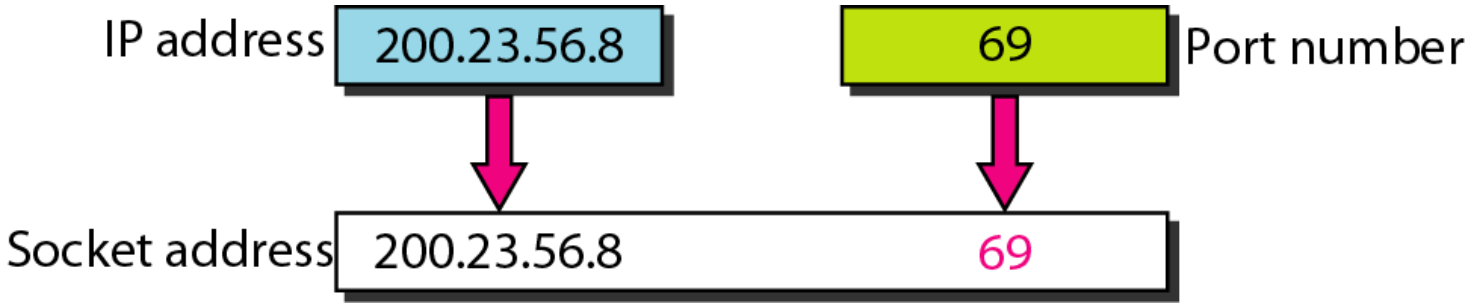
IP addresses versus port numbers



IANA ranges


Registered
0
1023
49,152
65,535
1024
49,151
Well known
Dynamic

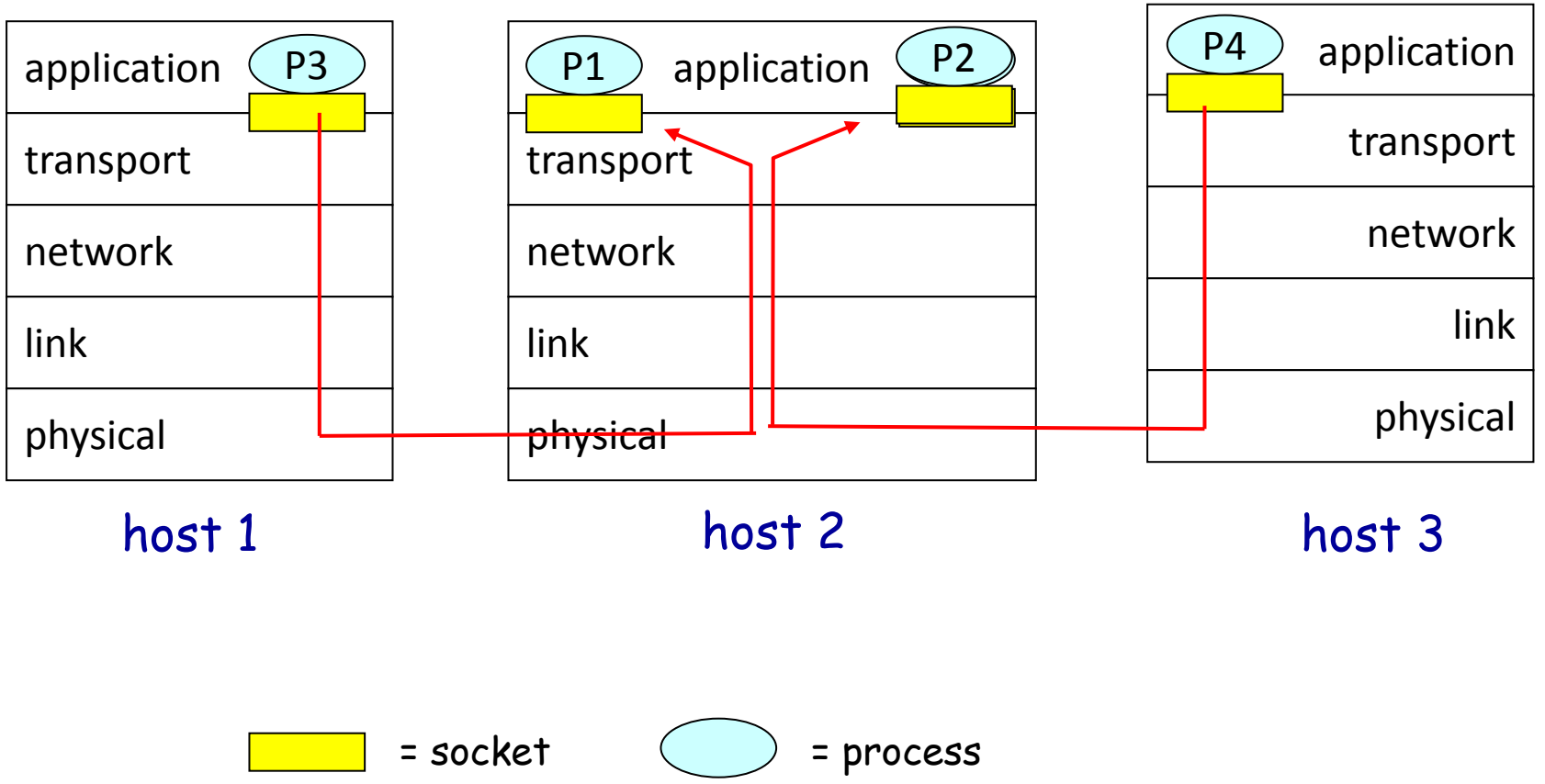
Socket address (IP address + Port Address)



What is the use of socket?

- The socket mechanism provides a means of inter-process communication (IPC).
- Socket is basically an API for enabling communication between two end points.
- A **socket** is one endpoint of a **two way** communication link between two programs running on the network.

Socket API


application
P3
transport
network
link
physical
P1
application
P2
transport
network
link
physical
P4
application
transport
network
link
physical
host 1
host 2
host 3
= socket
= process

3. **Data Transfer:** Transport layer breaks user data into smaller units and attaches a transport layer header to each unit forming a TPDU (Transport Layer Data Unit). The TPDU is handed over to the network layer for its delivery to destination. The TPDU header contains port number, sequence number, acknowledgement number, checksum and other fields.

4. **Multiplexing and Demultiplexing:** The addressing mechanism allows multiplexing and demultiplexing by the transport layer, as shown in Figure 2

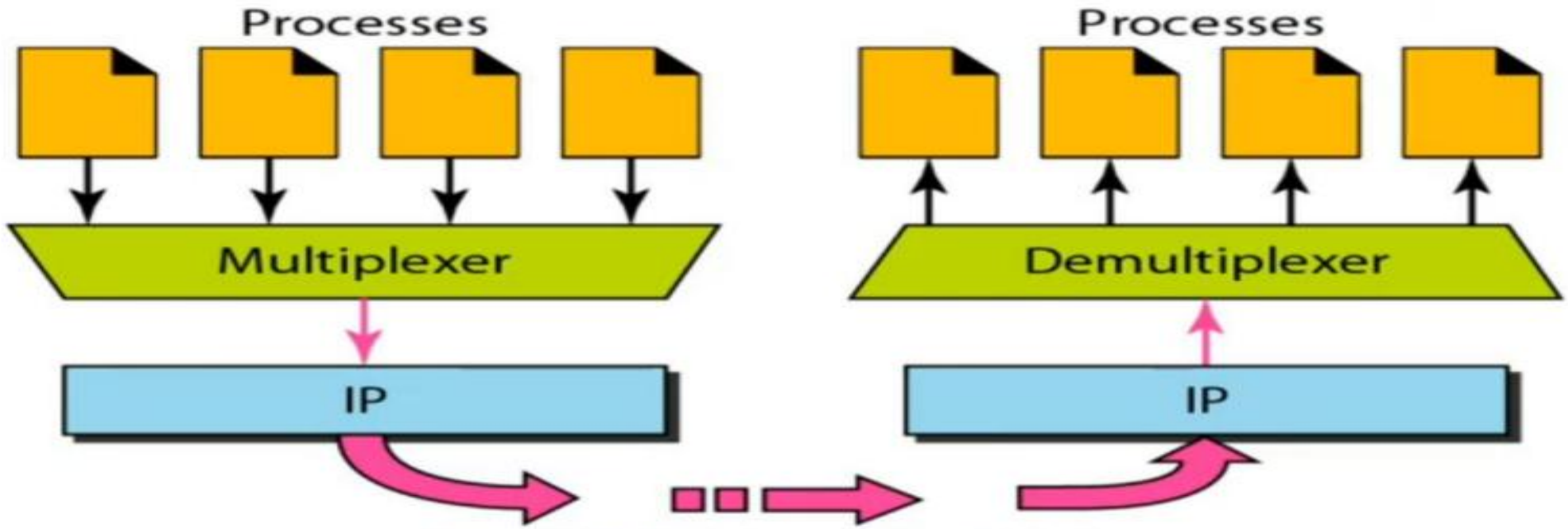


Figure 2: Multiplexing and demultiplexing

Multiplexing/demultiplexing

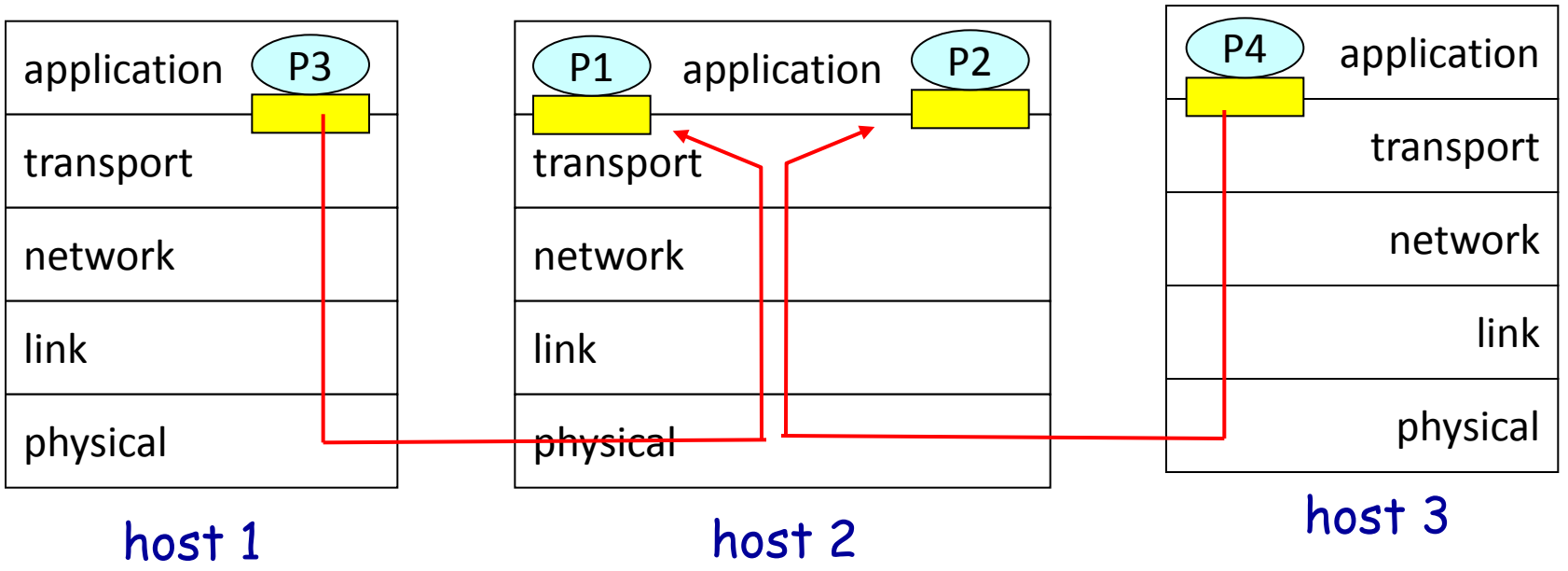
Demultiplexing at rcv host:

delivering received segments to correct socket

Multiplexing at send host:

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

= socket = process



5. **Flow Control:** Like data link layer, transport layer also performs flow control. However, flow control at transport layer is performed end-to-end rather than node-to-node. Transport Layer uses a sliding window protocol to perform flow control.

6. **Error Control:** Transport layer also provides end-to-end error control facility. Transport layer deals with several different types of errors:
    - Error due to damaged bits.
    - Error due to non delivery of TPDUs.
    - Error due to duplicate delivery of TPDUs.
    - Error due to delivery of TPDU to a wrong destination.

7. **Congestion Control:** Transport layer also handles congestion in the networks. Several different congestion control algorithms are used to avoid congestion.

Transport Layer Protocols (UDP and TCP)

- UDP provides
  - best effort delivery
  - Connectionless
  - Unreliable
  - Out of order delivery
- TCP
  - Reliable
  - In-order delivery
  - Congestion control
  - Flow control
  - Connection setup

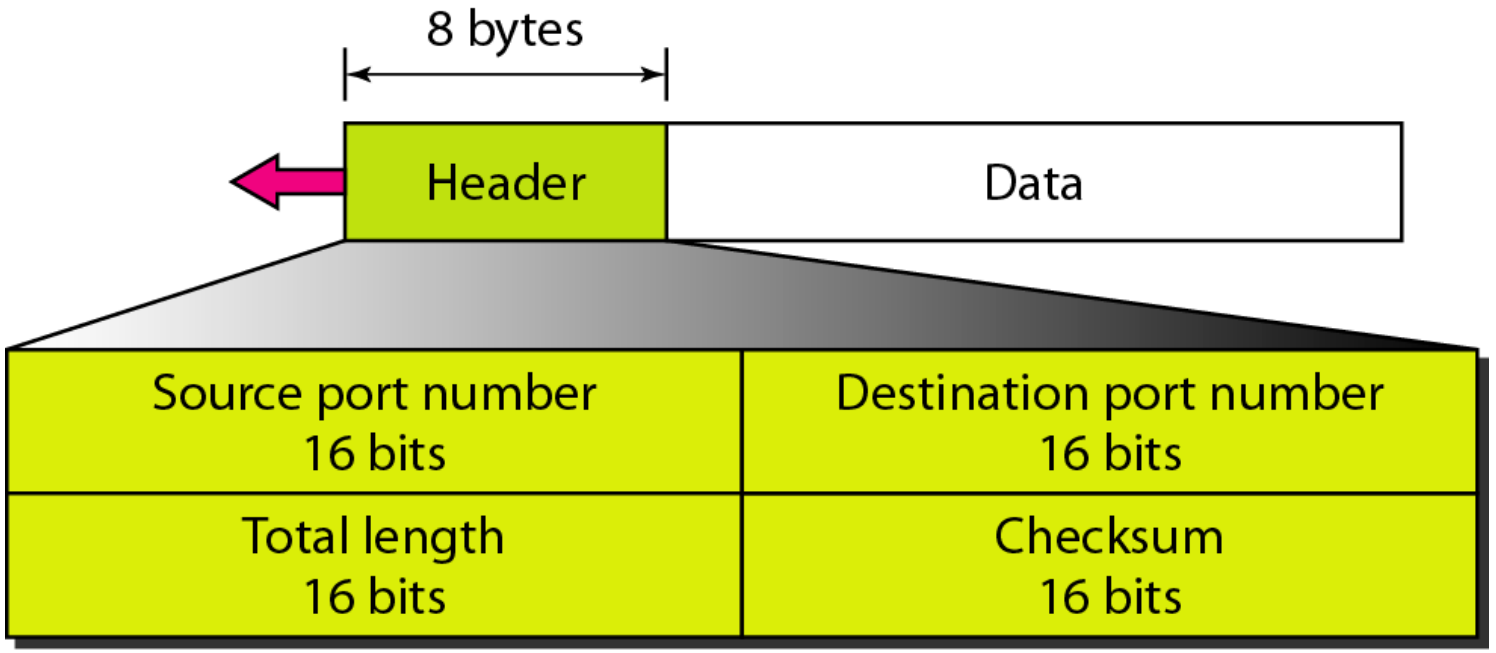
USER DATAGRAM PROTOCOL (UDP)

The User Datagram Protocol (UDP) is called a connectionless, unreliable transport protocol. It does not add anything to the services of IP except to provide process-to-process communication instead of host-to-host communication.

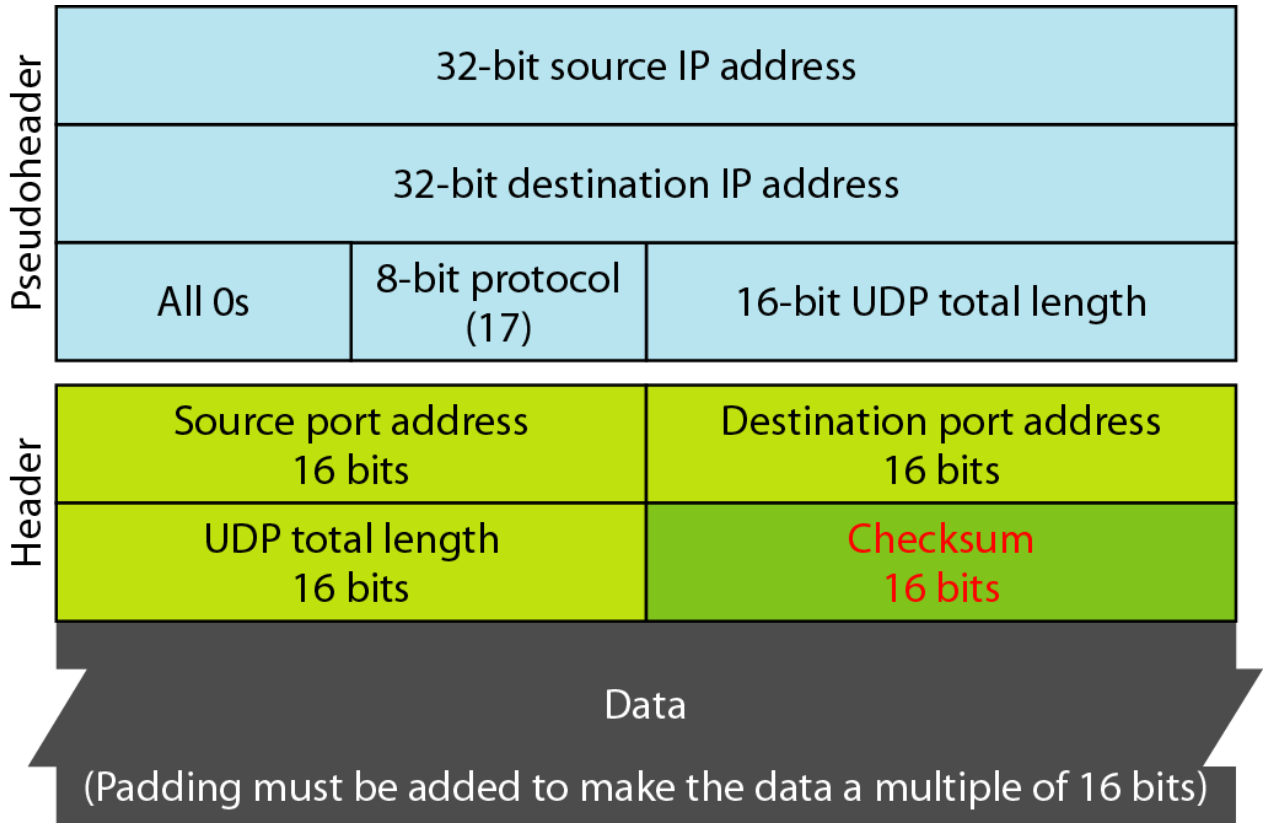
Well-known ports used with UDP

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)

User datagram format



Pseudoheader for checksum calculation



TCP

TCP is a connection-oriented protocol; it creates a virtual connection between two TCPs to send data. In addition, TCP uses flow and error control mechanisms at the transport level.

Stream delivery

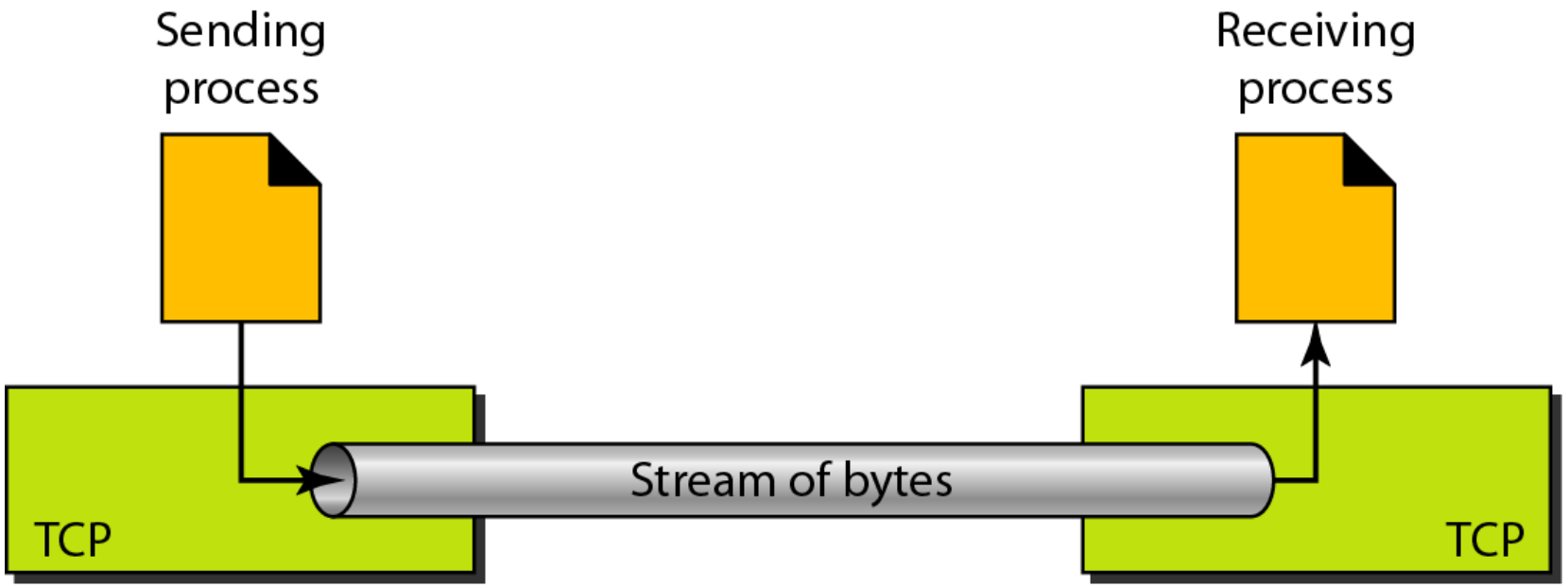
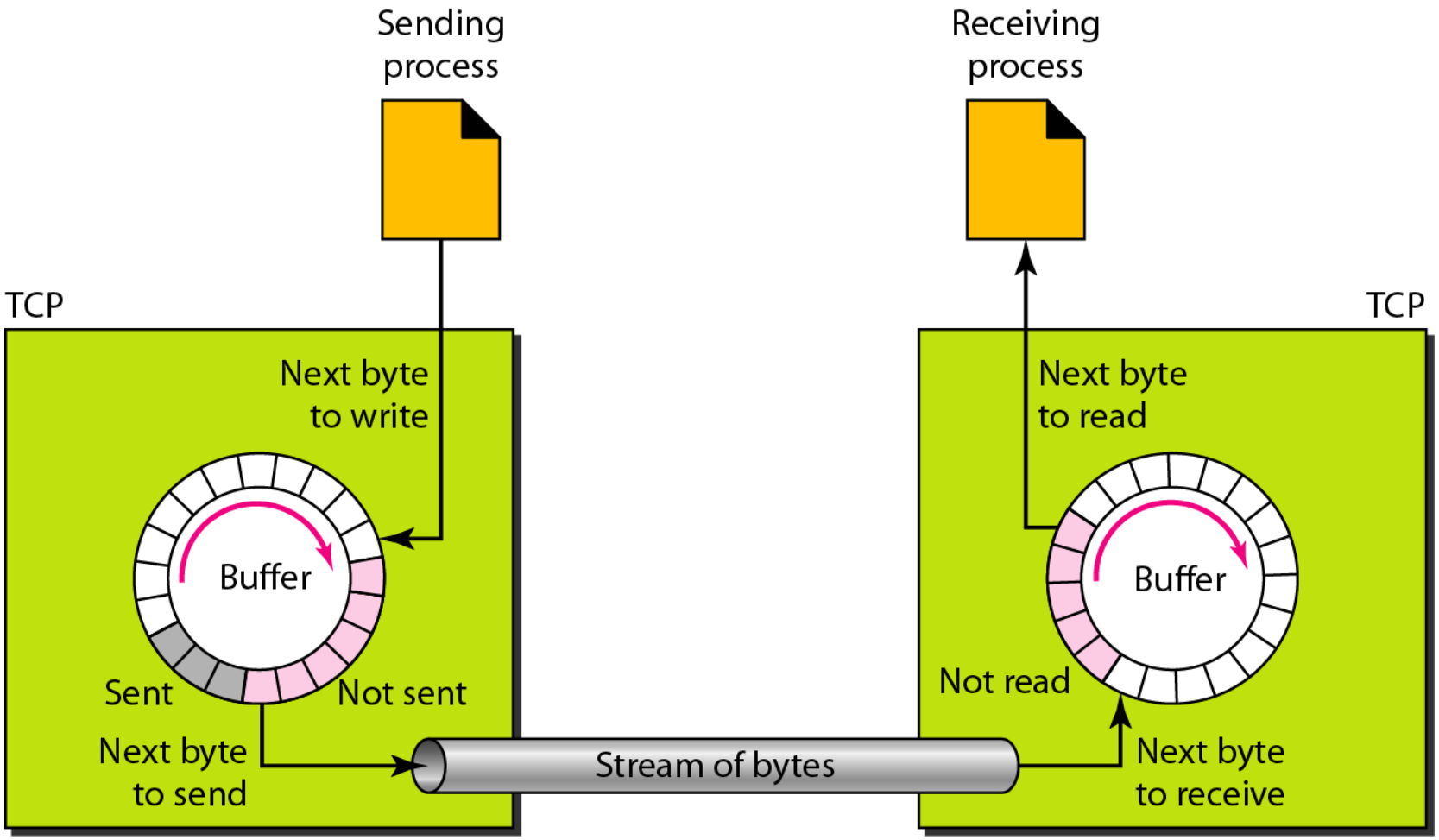
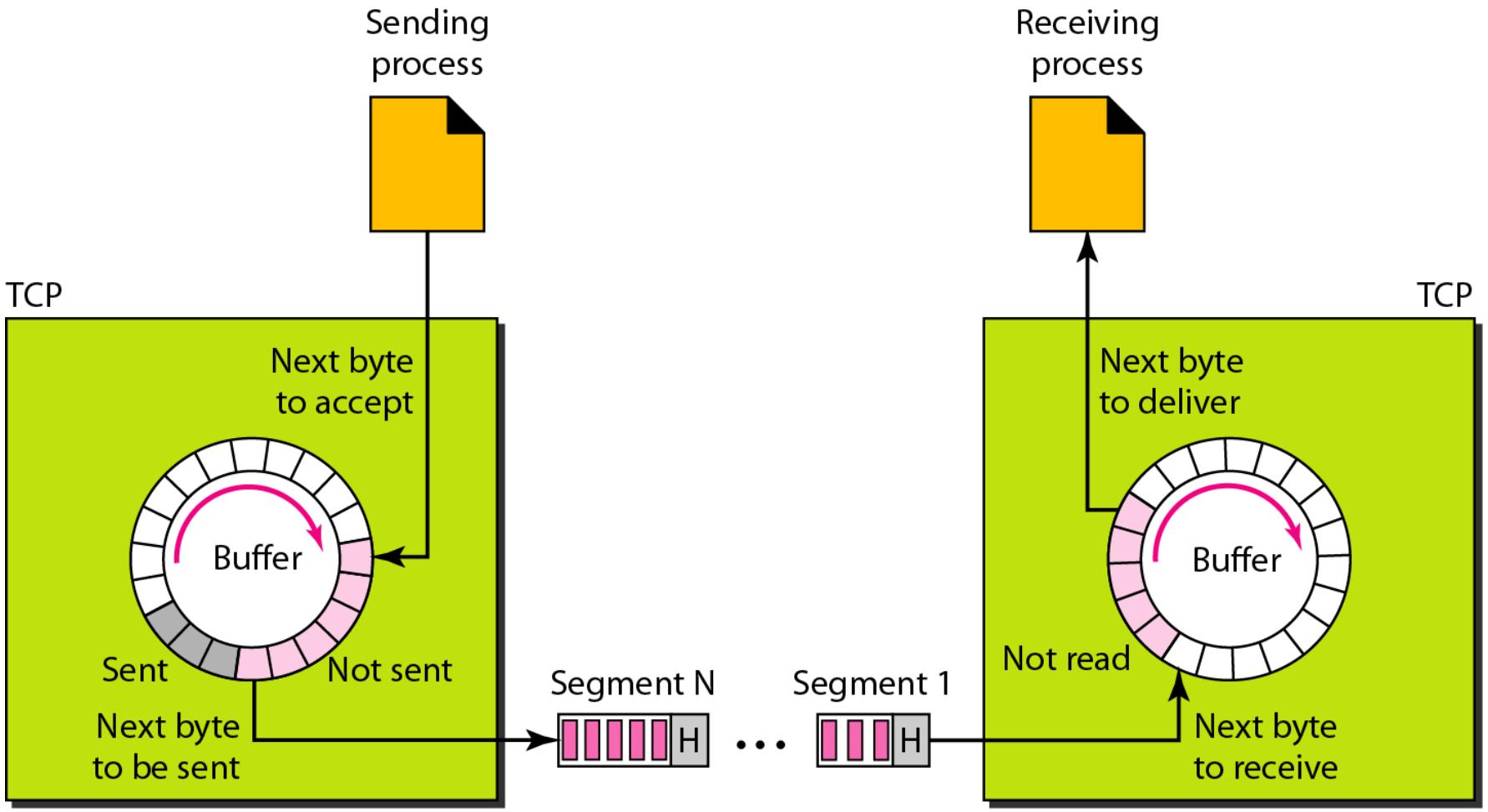


Figure *Sending and receiving buffers*



TCP segments



Note

The bytes of data being transferred in each connection are numbered by TCP.

The numbering starts with an arbitrarily generated number. Uses 32 bits.

Example 15.1

Suppose a TCP connection is transferring a file of 5,000 bytes. The first byte is numbered 10,001. What are the sequence numbers for each segment if data are sent in five segments, each carrying 1,000 bytes?

Solution

The following shows the sequence number for each segment:

Segment 1	→	Sequence Number:	10,001	Range:	10,001	to	11,000
Segment 2	→	Sequence Number:	11,001	Range:	11,001	to	12,000
Segment 3	→	Sequence Number:	12,001	Range:	12,001	to	13,000
Segment 4	→	Sequence Number:	13,001	Range:	13,001	to	14,000
Segment 5	→	Sequence Number:	14,001	Range:	14,001	to	15,000

Note

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

Note

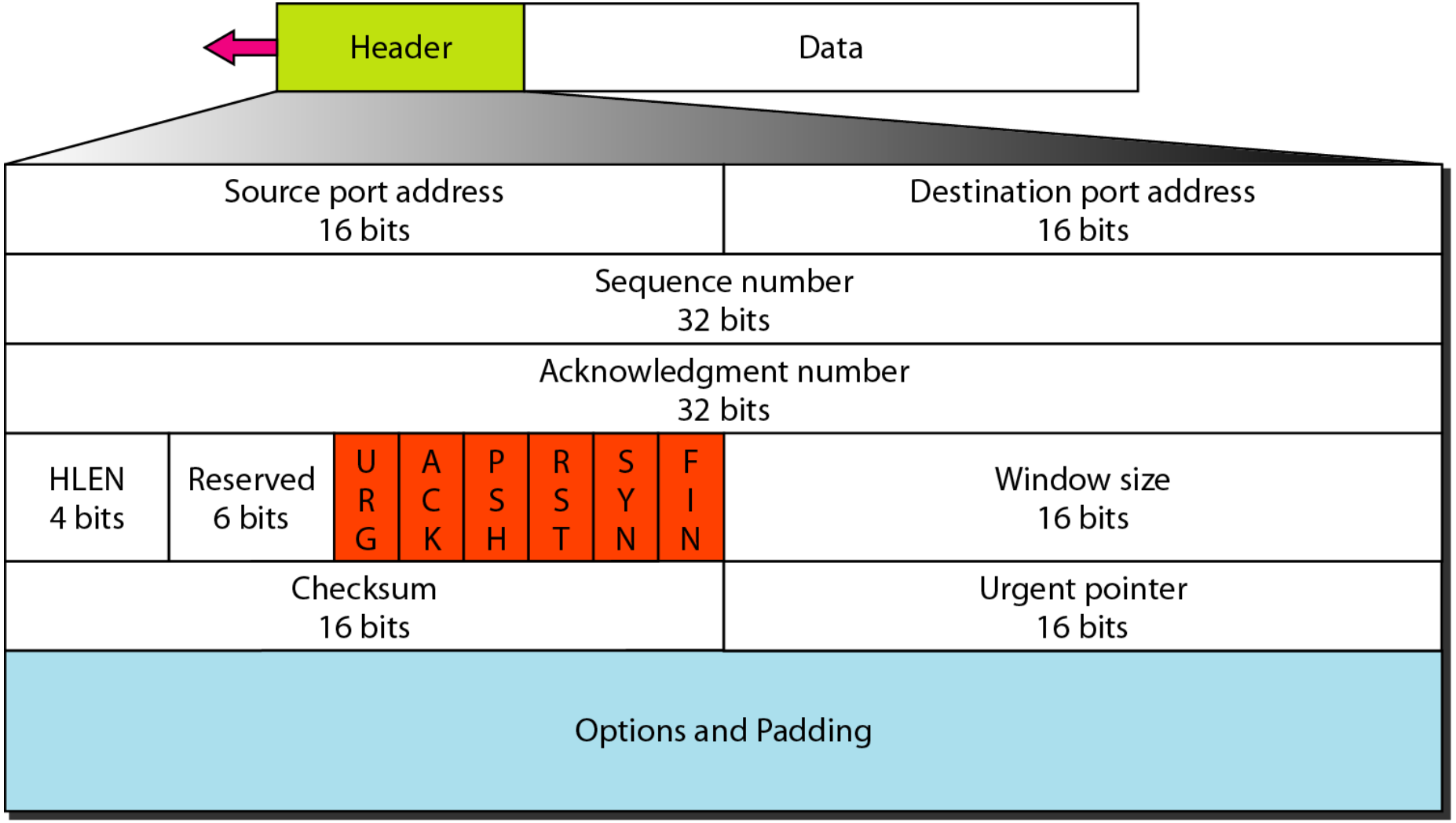
The value of the acknowledgment field in a segment defines the number of the next byte a party expects to receive.

The acknowledgment number is cumulative.

15-3 SEGMENT

Before discussing TCP in more detail, let us discuss the TCP packets themselves. A packet in TCP is called a segment.

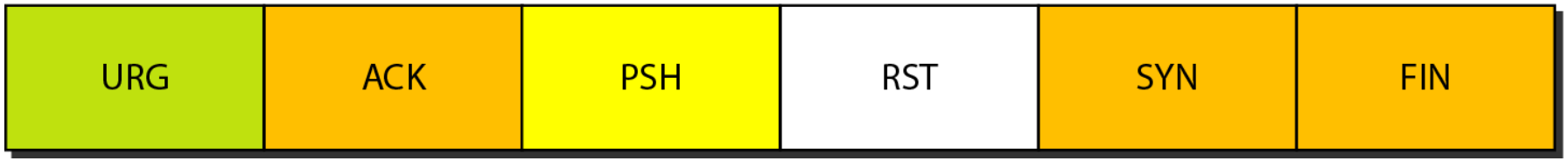
TCP segment format



Control field

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



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	Reset the connection.
SYN	Synchronize sequence numbers during connection.
FIN	Terminate the connection.

15-4 A TCP CONNECTION

TCP is connection-oriented. It establishes a virtual path between the source and destination. All of the segments belonging to a message are then sent over this virtual path. You may wonder how TCP, which uses the services of IP, a connectionless protocol, can be connection-oriented. The point is that a TCP connection is virtual, not physical. TCP operates at a higher level. TCP uses the services of IP to deliver individual segments to the receiver, but it controls the connection itself. If a segment is lost or corrupted, it is retransmitted.

Figure 15.9 *Connection establishment using three-way handshake*

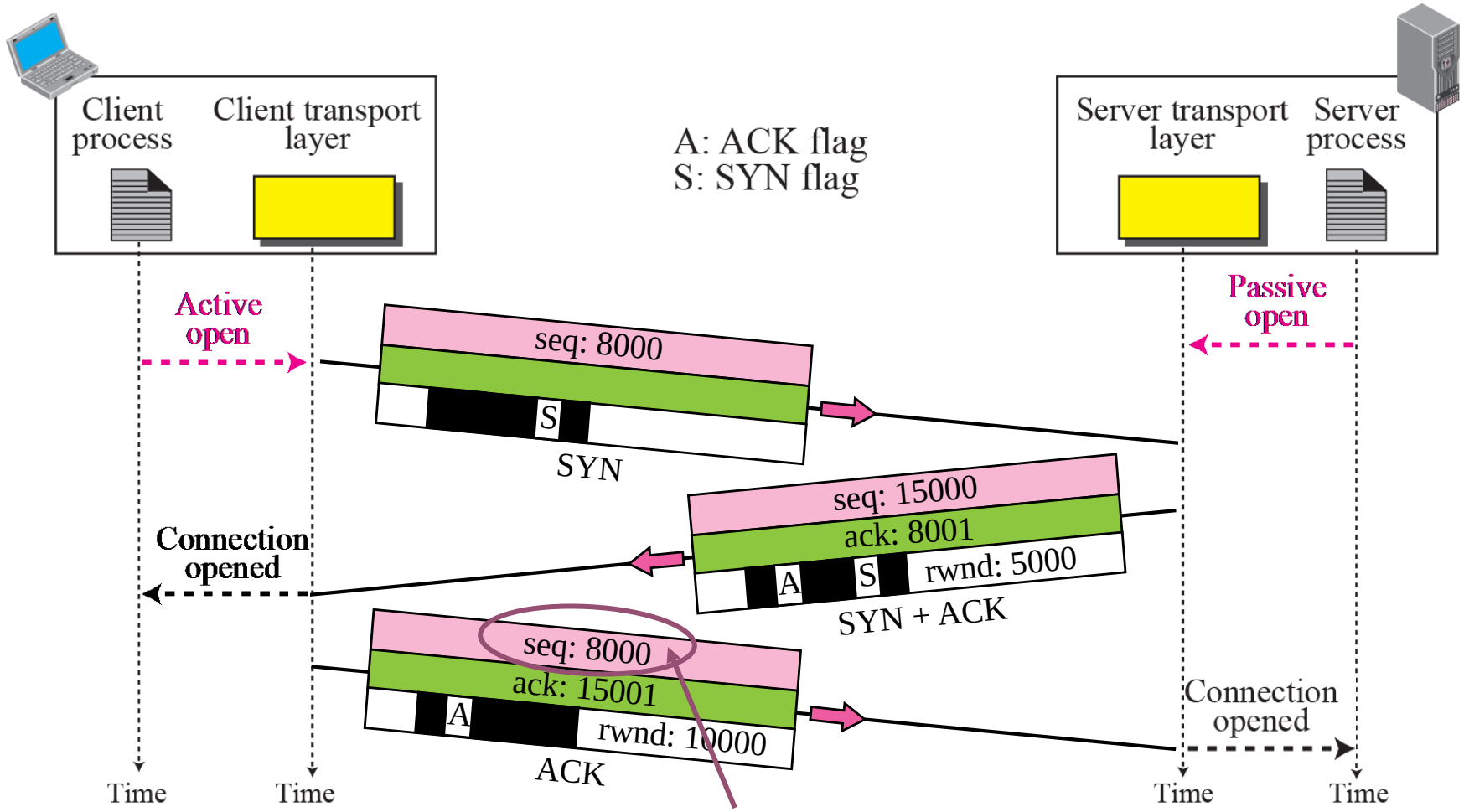
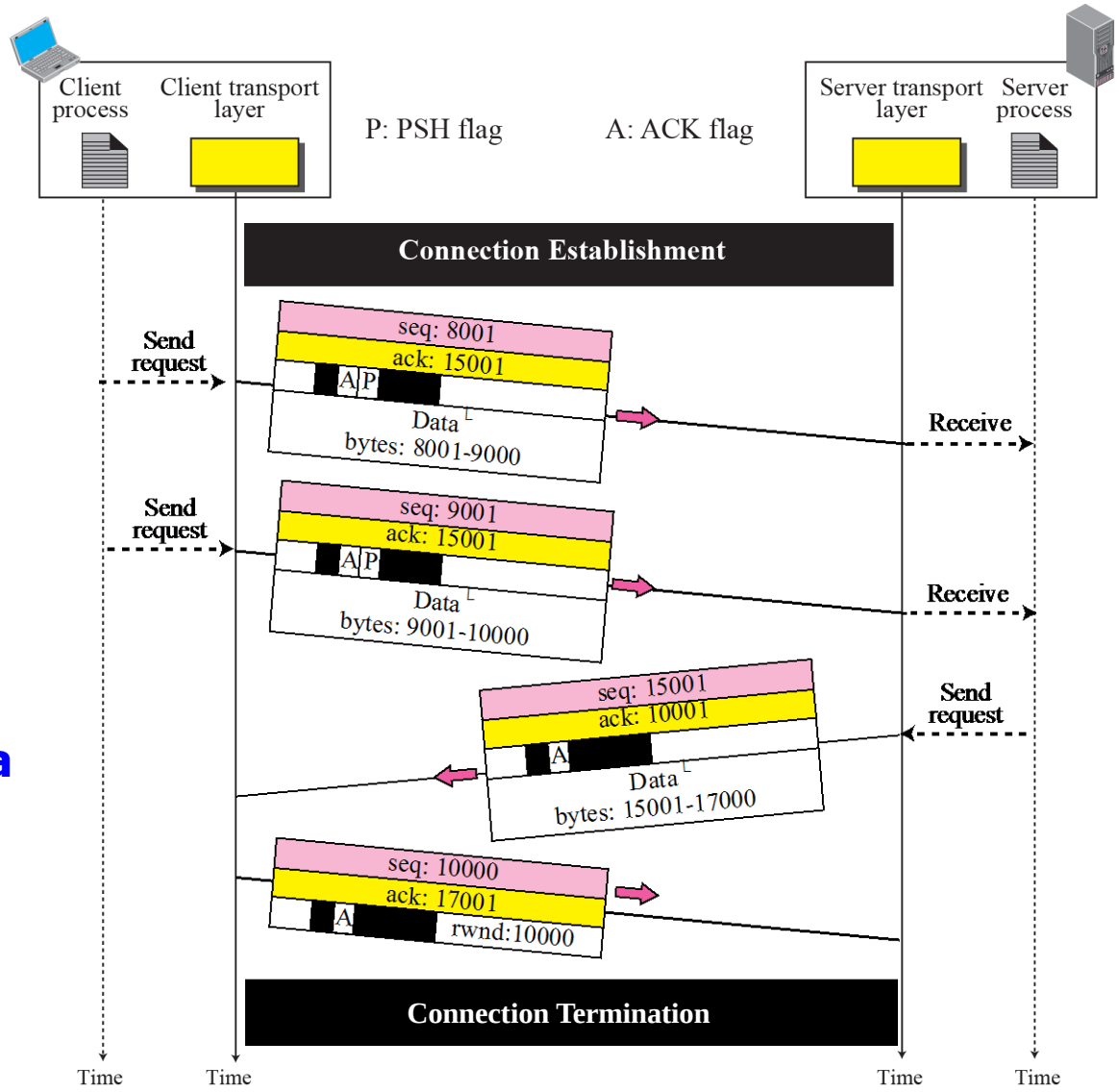


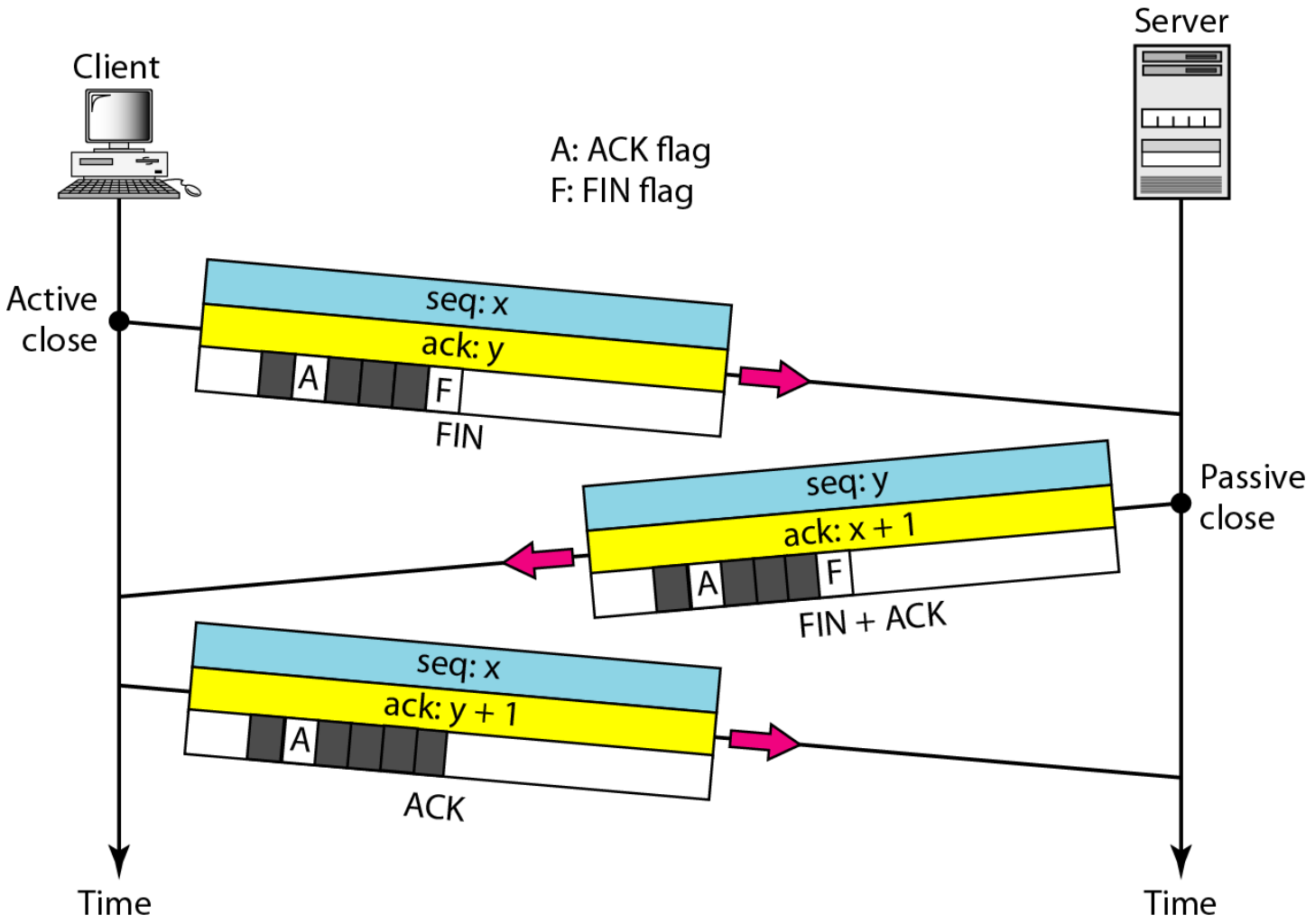
Figure 15.10 *Data Transfer*



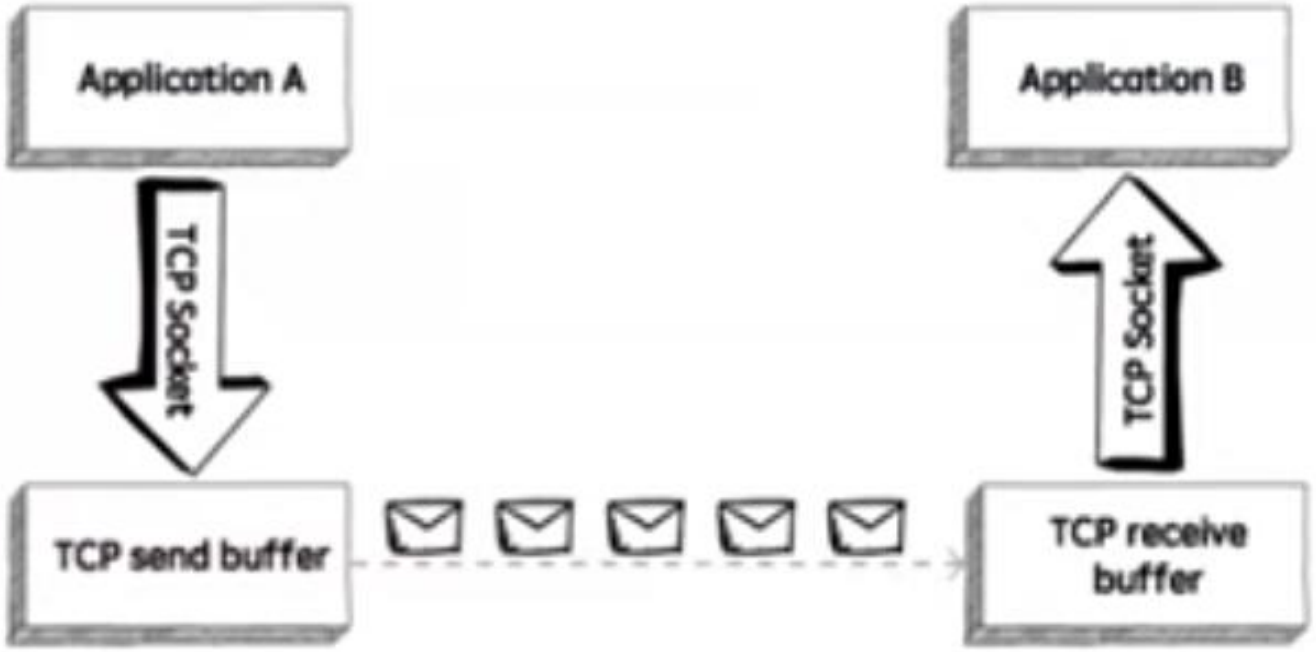
Pushing data

Urgent data

Connection termination using three-way handshaking



TCP Flow Control
&
Window Size



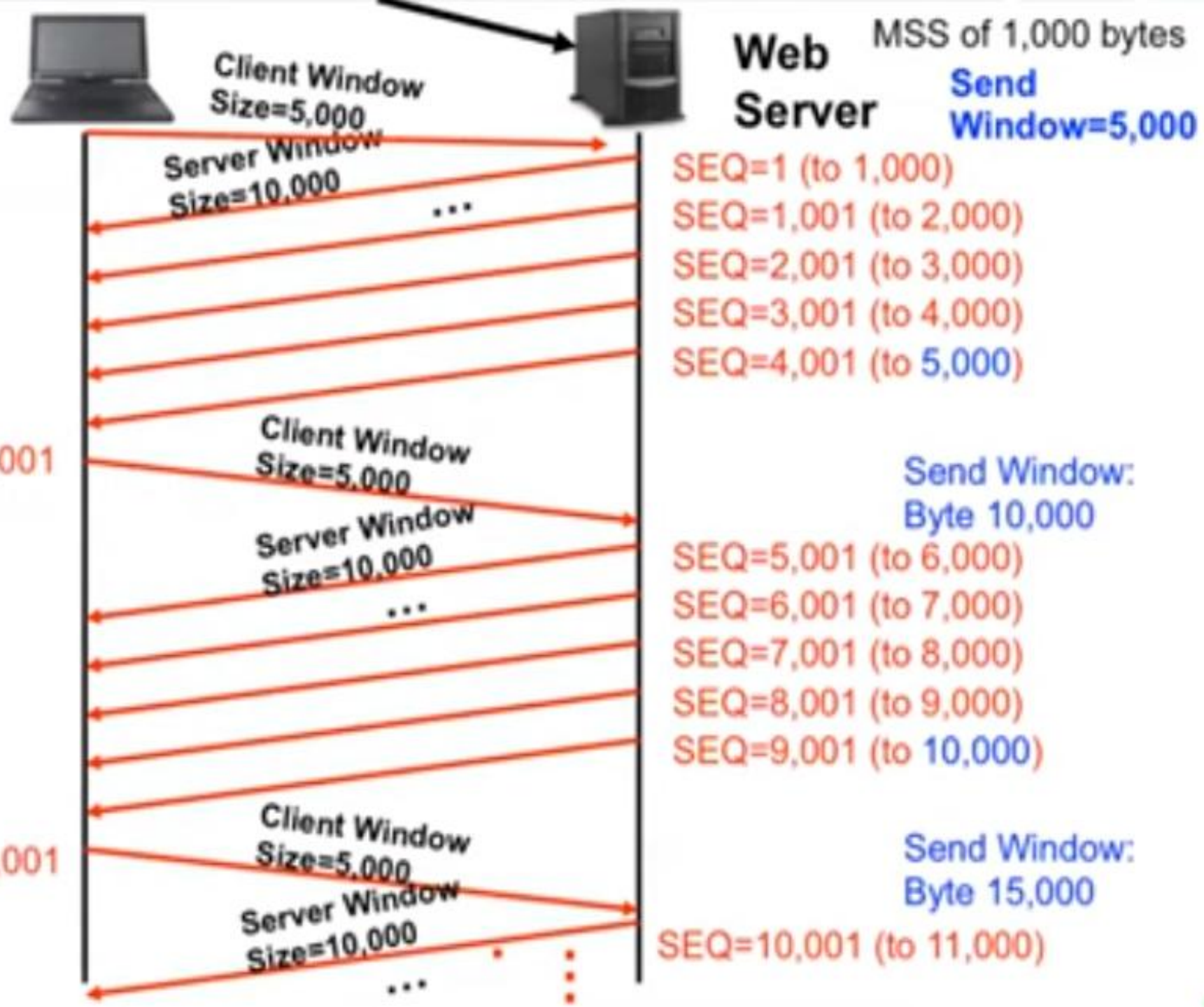
Lucy and the Chocolate Factory high res

- TCP Flow Control means that TCP will help ensure that the sender is not overwhelming the the receiver by sending more packets (segments) than it can handle.

Windowing and Flow Control

- TCP implements flow control by increasing / decreasing window sizes as required.
  - Window sizes are variable during the lifetime of a connection.

Client has a Window Size of 5,000 bytes



- Send Window Byte: This is the last byte that can be sent before receiving an ACK
- This is known as a **Stop-and-Wait** windowing protocol.
- Server must wait for acknowledgment before continuing to send data.
- A better method?

TCP Window Management

Before discussing data transfer in TCP and the issues such as flow, error, and congestion control, we describe the windows used in TCP.

- TCP uses two windows (send window and receive window) for each direction of data transfer, which means four windows for a bidirectional communication.
- To make the discussion simple, we make an assumption that communication is only unidirectional;
- The bidirectional communication can be inferred using two unidirectional communications with piggybacking.

TCP Window Management

Figure 15.22 *Send window in TCP*

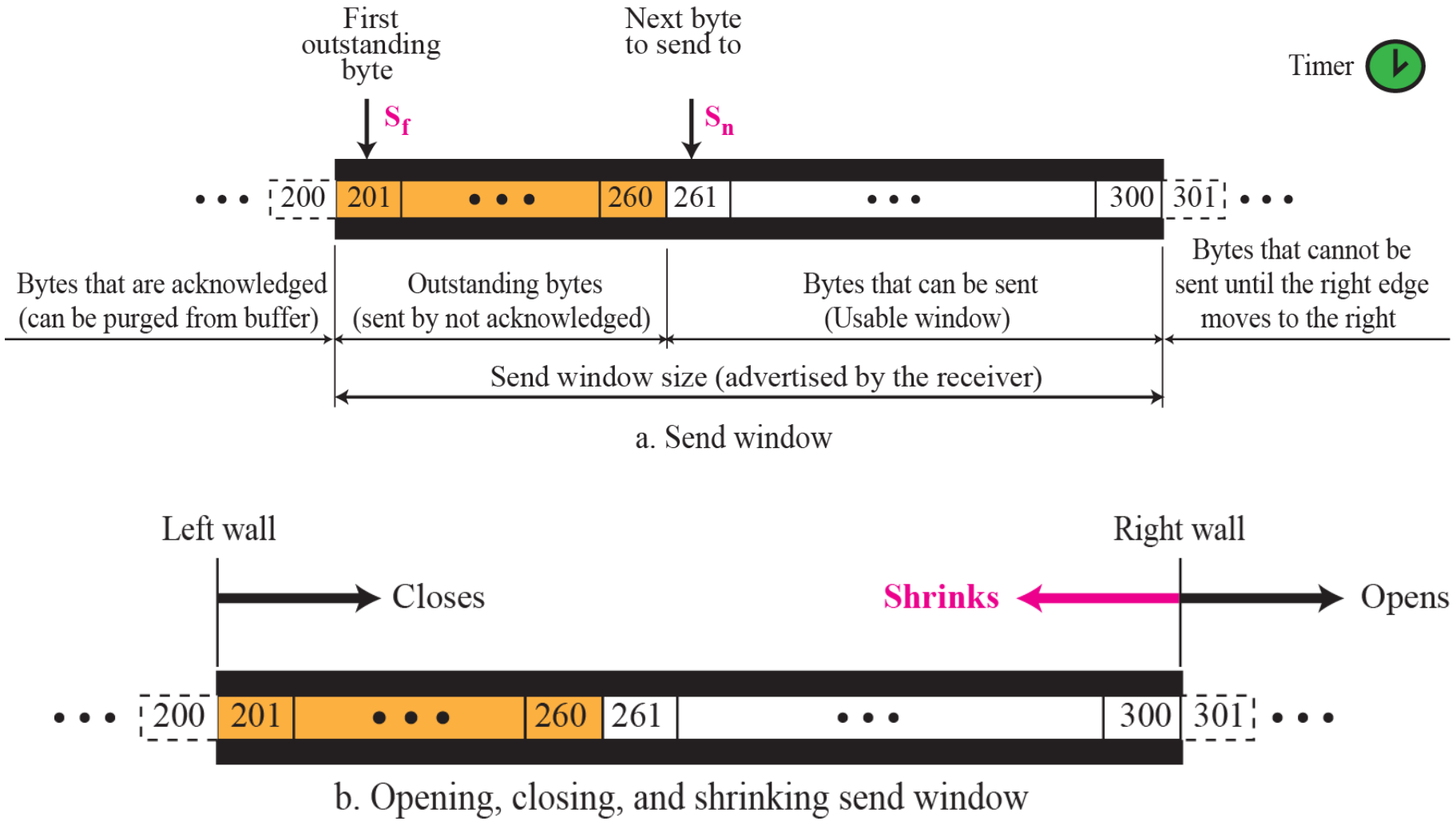


Figure 15.23 *Receive window in TCP*

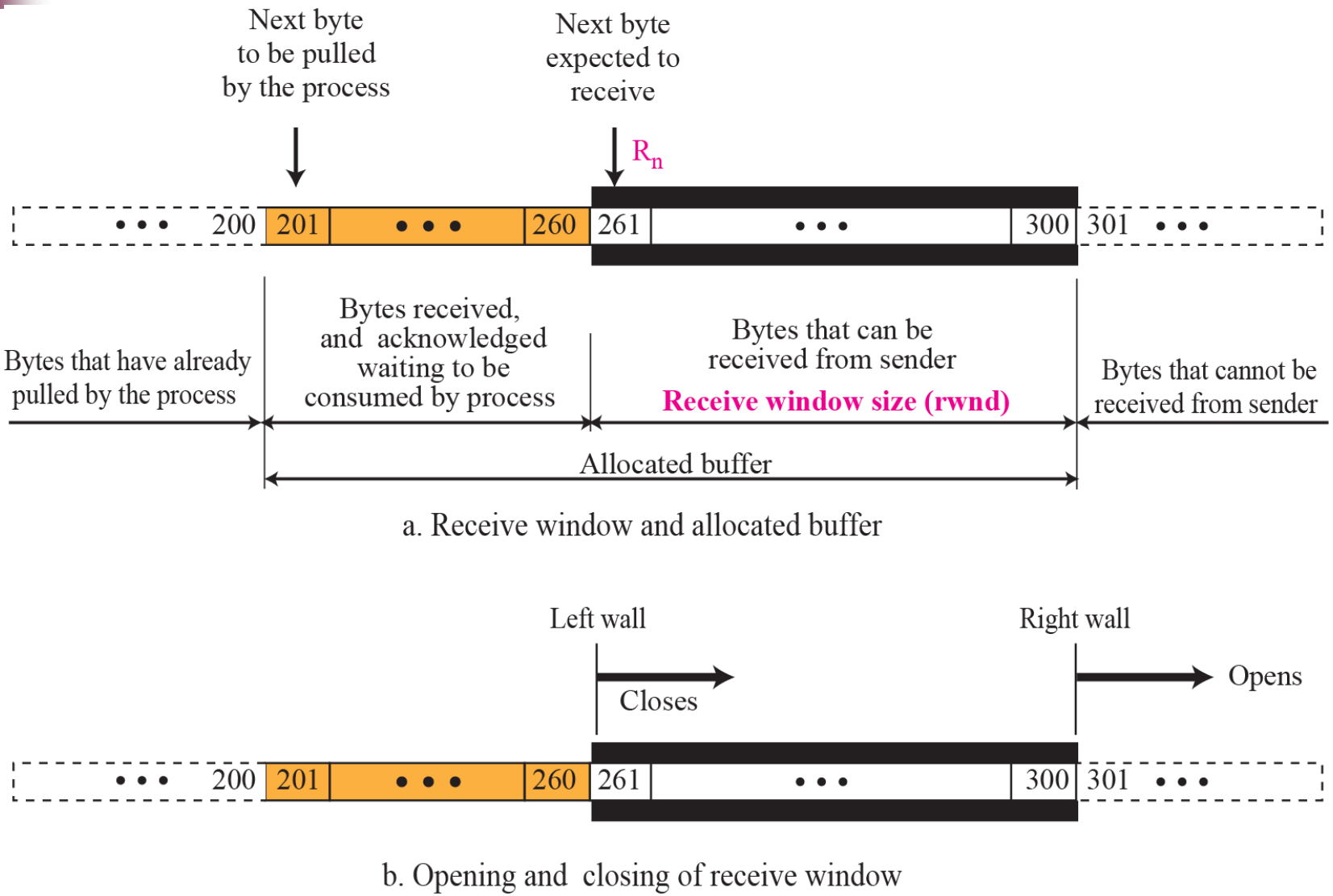
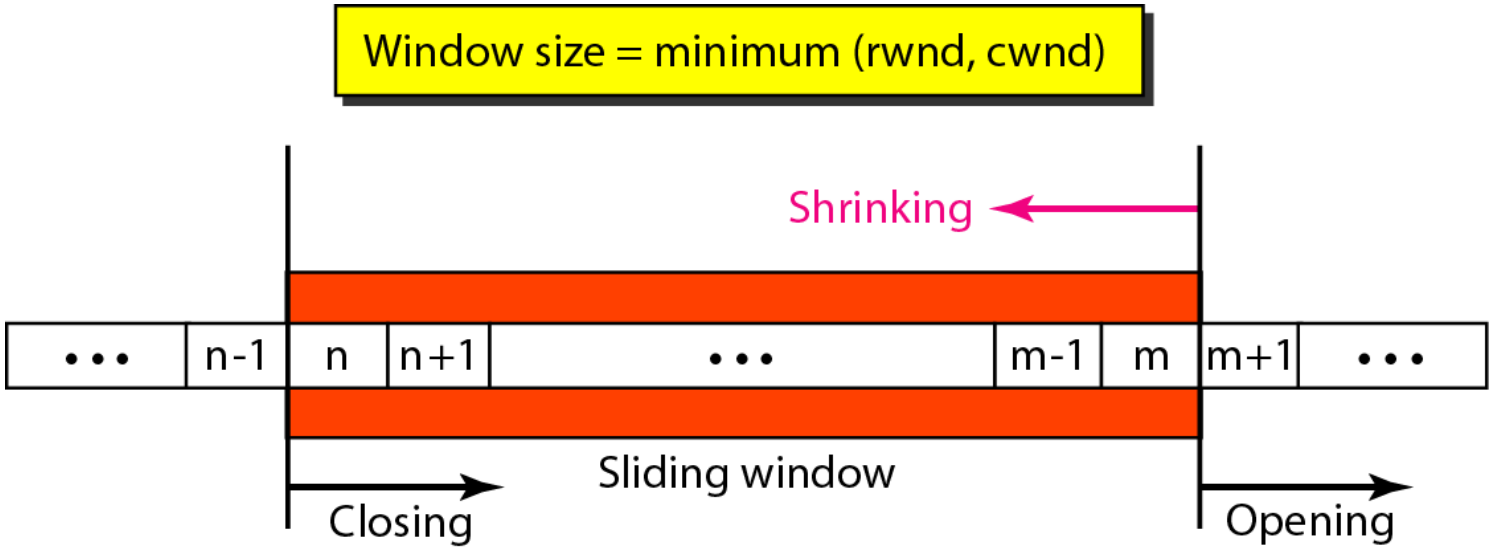


Figure 23.22 *Sliding window*



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 sliding windows are byte-oriented.

Example 23.4

What is the value of the receiver window (rwnd) for host A if the receiver, host B, has a buffer size of 5000 bytes and 1000 bytes of received and unprocessed data?

Solution

The value of rwnd = 5000 − 1000 = 4000. Host B can receive only 4000 bytes of data before overflowing its buffer. Host B advertises this value in its next segment to A.

Example 23.5

What is the size of the window for host A if the value of rwnd is 3000 bytes and the value of cwnd is 3500 bytes?

Solution

The size of the window is the smaller of rwnd and cwnd, which is 3000 bytes.

Example 23.6

Figure 23.23 shows an unrealistic example of a sliding window. The sender has sent bytes up to 202. We assume that cwnd is 20 (in reality this value is thousands of bytes). The receiver has sent an acknowledgment number of 200 with an rwnd of 9 bytes (in reality this value is thousands of bytes). The size of the sender window is the minimum of rwnd and cwnd, or 9 bytes. Bytes 200 to 202 are sent, but not acknowledged. Bytes 203 to 208 can be sent without worrying about acknowledgment. Bytes 209 and above cannot be sent.

Figure 23.23 *Example 23.6*

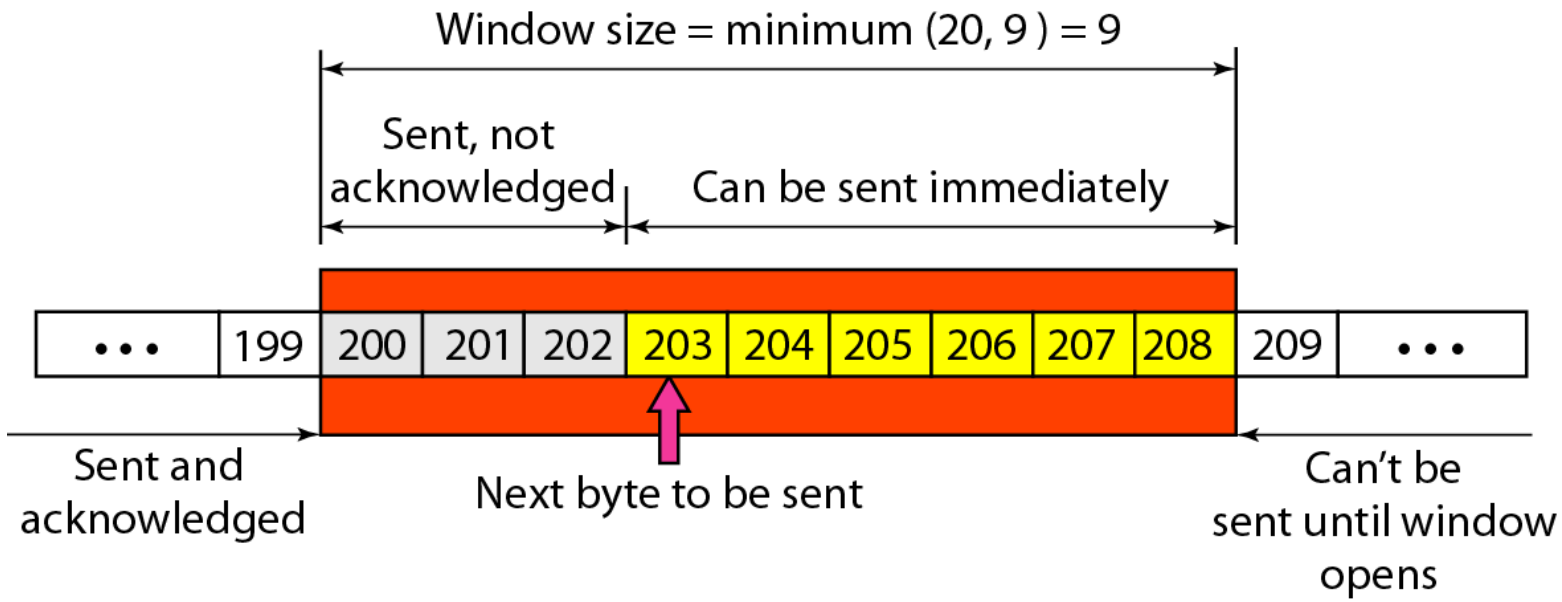


Figure 23.24 *Normal operation*

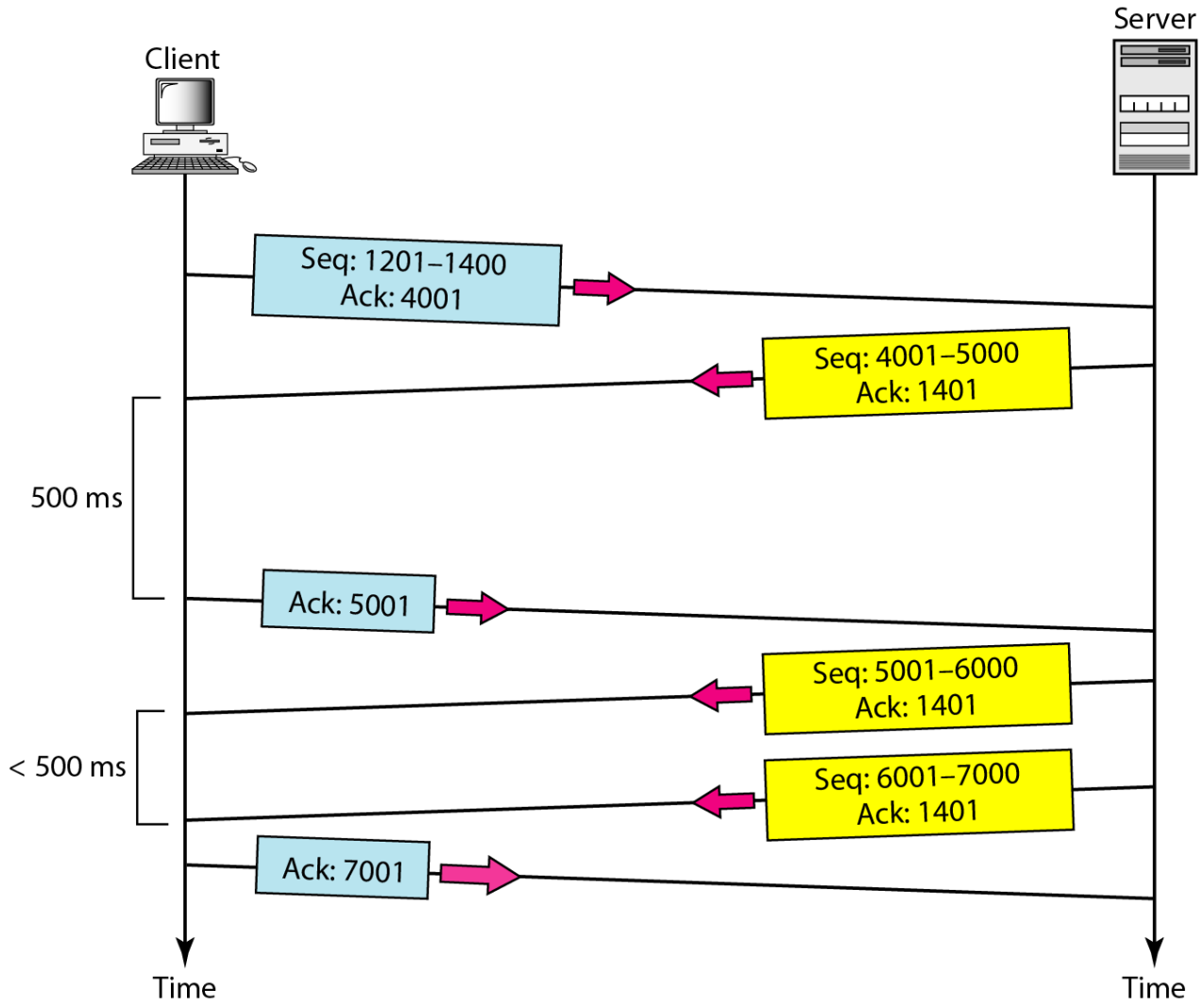
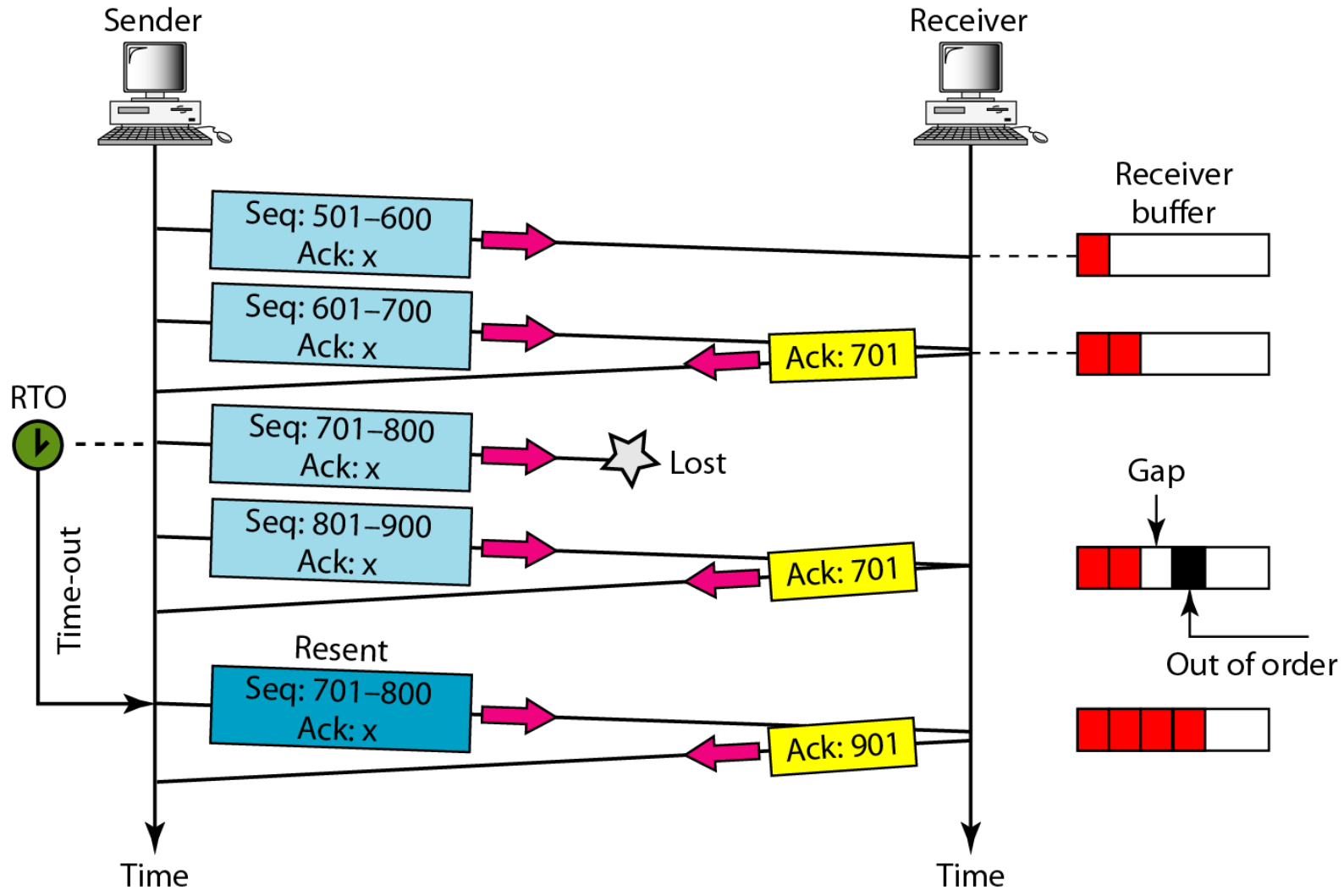


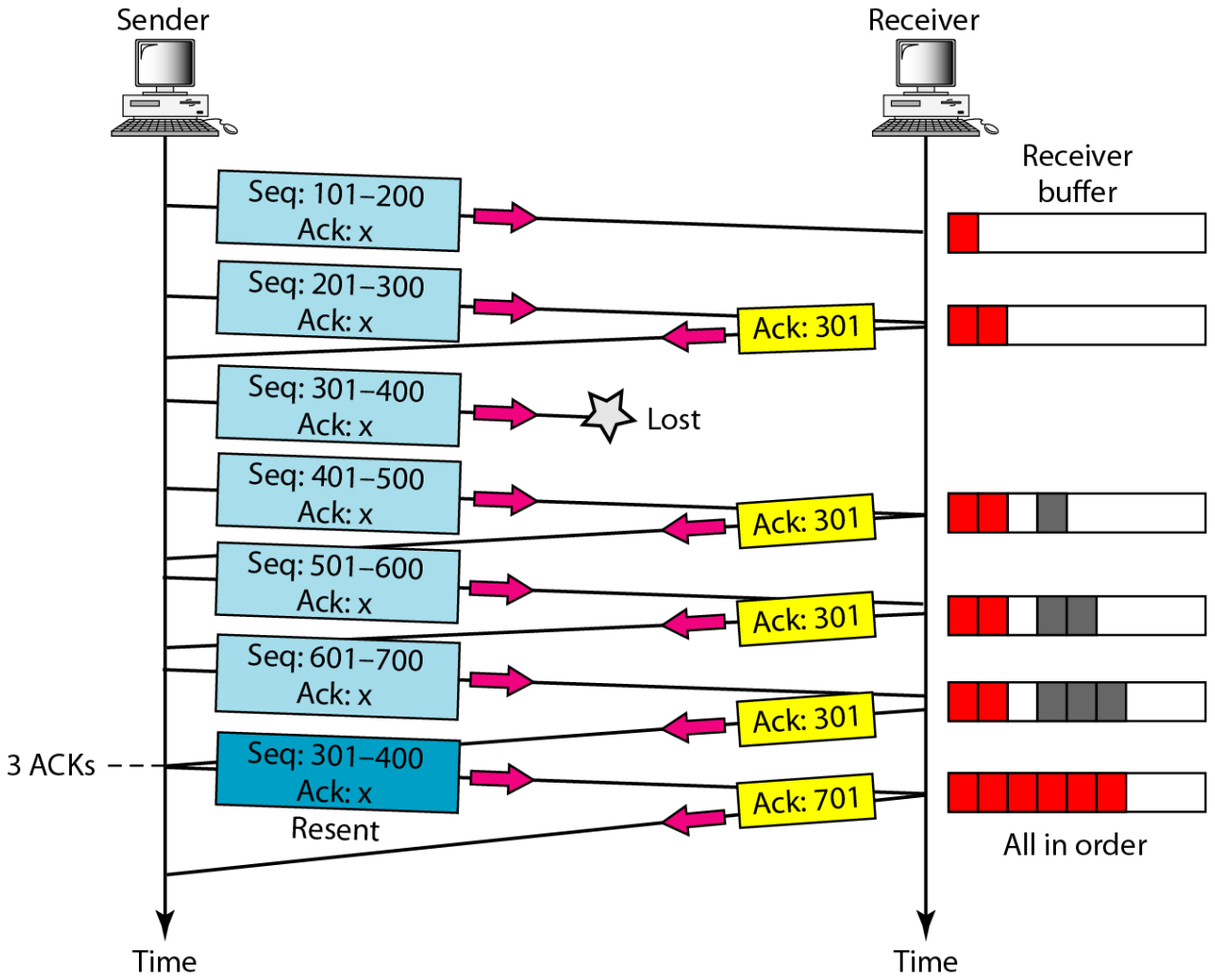
Figure 23.25 *Lost segment*



Note

The receiver TCP delivers only ordered data to the process.

Figure 23.26 *Fast retransmission*



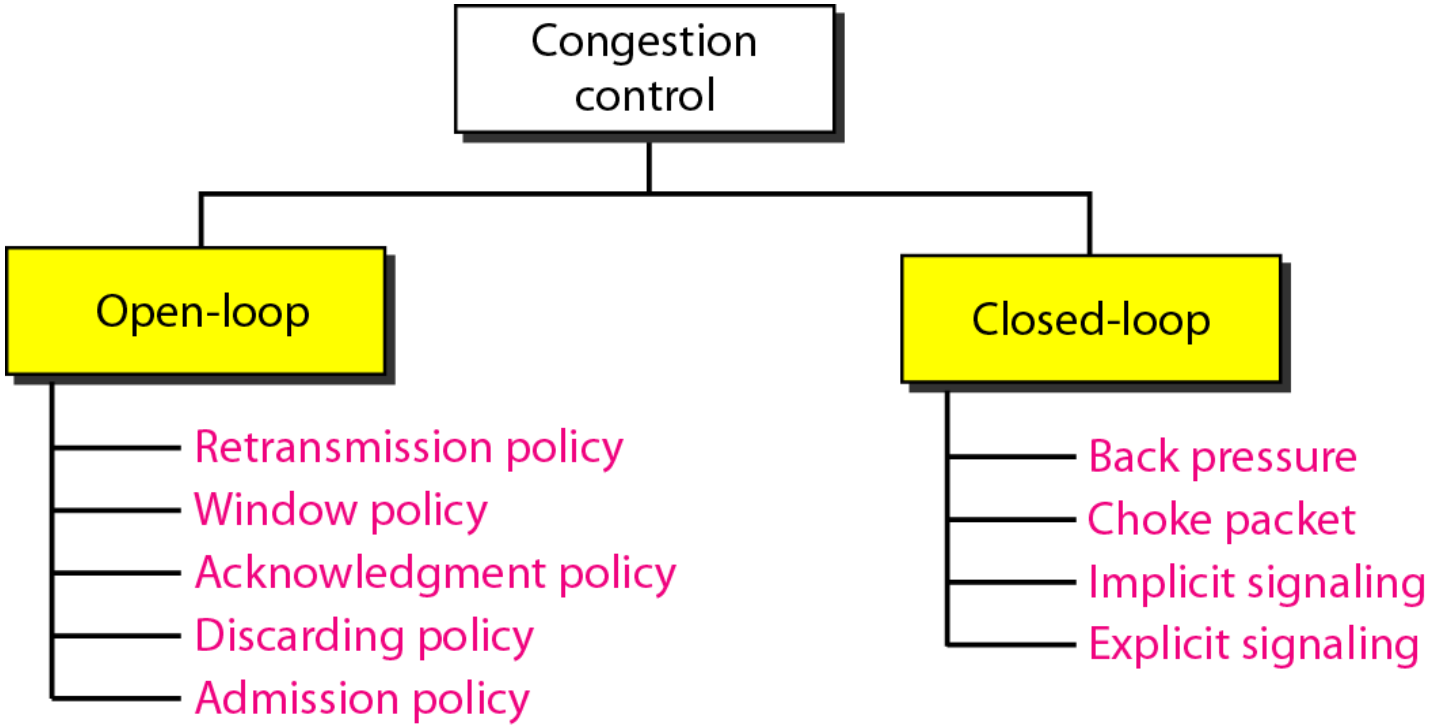
15-9 CONGESTION CONTROL

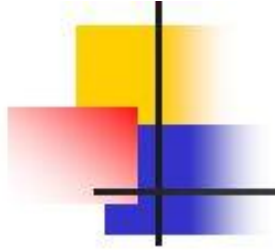
Congestion control in TCP is based on both open loop and closed-loop mechanisms. TCP uses a congestion window and a congestion policy that avoid congestion and detect and alleviate congestion after it has occurred.

Congestion Control Introduction:

- When too many packets are present in (a part of) the subnet, performance degrades. This situation is called **congestion**.
- As traffic increases too far, the routers are no longer able to cope and they begin losing packets.
- At very high traffic, performance collapses completely and almost no packets are delivered.
- Reasons of Congestion:
  - Slow Processors.
  - High stream of packets sent from one of the sender.
  - Insufficient memory.
  - High memory of Routers also add to congestion as becomes un manageable and un accessible. (Nagle, 1987).
  - Low bandwidth lines.

Figure 24.5 *Congestion control categories*





Congestion Control

- Open-loop congestion control [Prevention]
  - Windows policy: Type of window at sender may also affect congestion. Selective repeat is better than Go-Back-N.
  - Acknowledgement policy: Policy set by receiver may also affect congestion. If receiver does not acknowledge every packet it receives, it may slow down the sender and help prevent congestion.
  - Discard policy: Discard less sensitive packets [in audio transmission] at routers.
  - Admission policy: Switches in a flow first check the resource requirement of a flow before admitting it to the network.

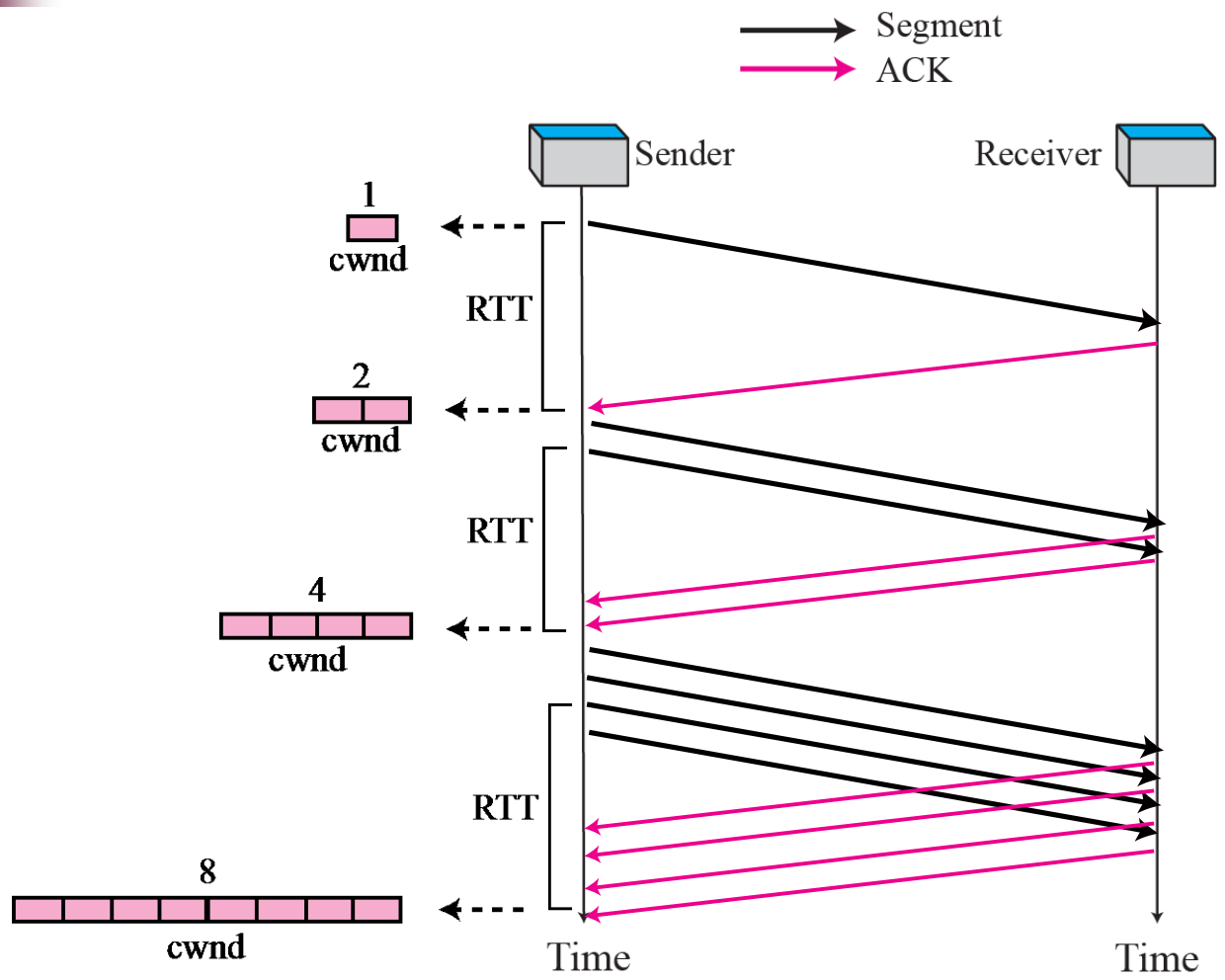
Closed-loop congestion control [Removal]

- Back Pressure: When a router is congested, it can inform the previous upstream router to reduce the rate of outgoing packets. The action can be recursive all the way to the router before the source.
- Choke Point: A packet sent by a router to the source to inform it of congestion. This type of control is similar to ICMP's source quench packet.
- Implicit Signaling: Source can detect an implicit signal concerning congestion and slow down its sending rate. For example, the mere delay in receiving an acknowledgement can be a signal that the network is congested.
- Explicit Signaling: Routers that experience congestion can send an explicit signal, the setting of a bit in a packet, for example, to inform the sender or the receiver of congestion.
  - Backward Signaling: Bit can be set in a packet moving in the direction opposite to the congestion; indicate the source.
  - Forward Signaling: Bit can be set in a packet moving in the direction of the congestion; indicate the destination.

Congestion Control in TCP

- Slow Start
- Additive Increase (Congestion Avoidance)
- Multiplicative decrease

Slow start, exponential increase



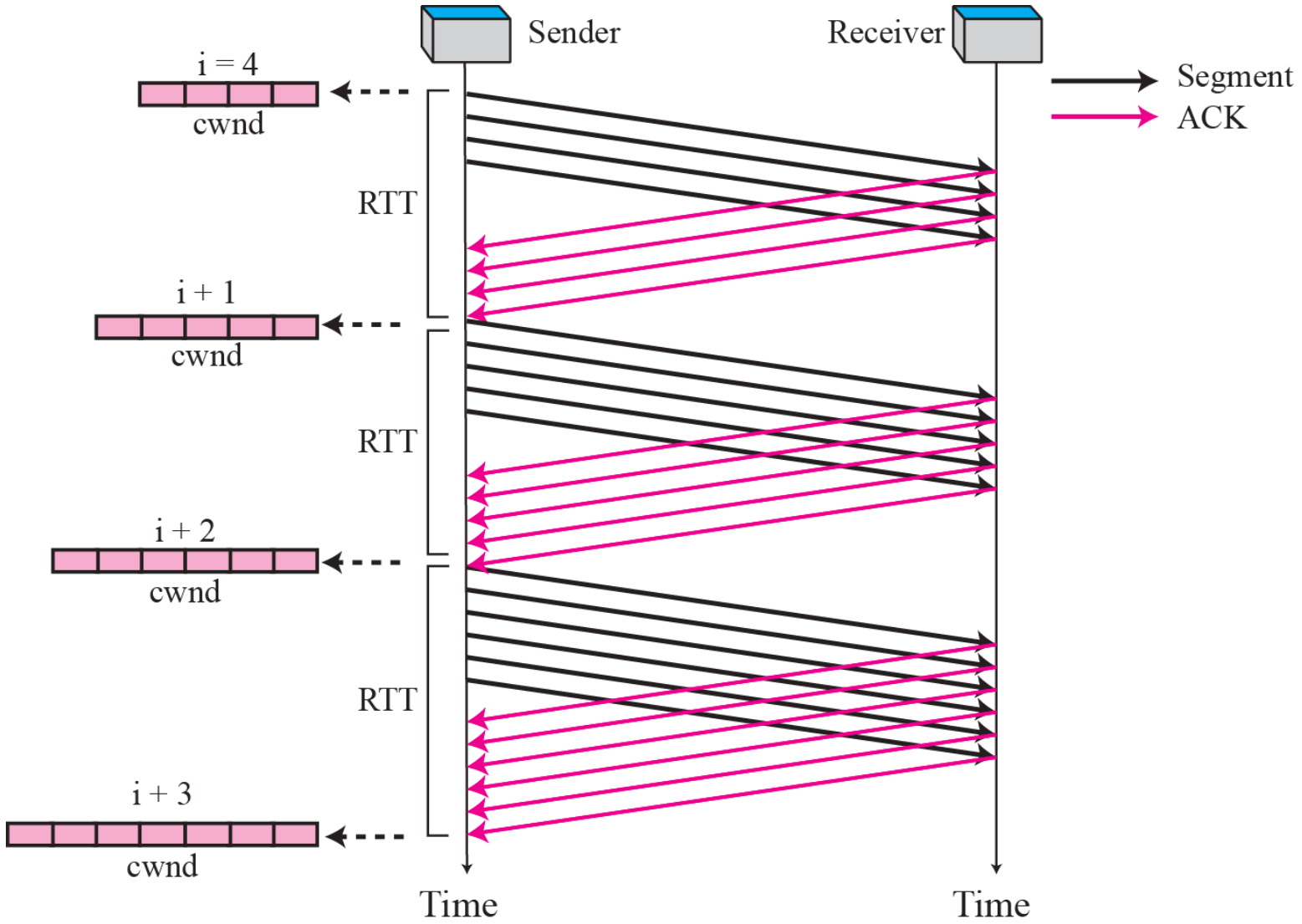
Note

In the slow start algorithm, the size of the congestion window increases exponentially until it reaches a threshold (ssthresh).

What will happen reaches to threshold

- ssthresh=window/2
- Cwnd=ssthresh

Figure *Congestion avoidance, additive increase multiplicative decreaase (AIMD)*



Note

In the congestion avoidance algorithm the size of the congestion window increases additively until congestion is detected.

What will happen reaches to threshold

- Congestion is detected by timeout
- ssthresh=window/2
- cwnd=1

Multiplicative Decrease

- Congestion is detected by 3 dup Acks
- ssthresh=window/2
- cwnd=ssthresh

- Packet =50 Packet size=1KB
- Lost Packets = 10,25,34,45

RTT	Packets
1	1
2	2 3
3	4 5 6 7
4	8 9 **10** 11 12 13 14 15

Window size=8

Congestion threshold = $\frac{\text{Window size}}{2} = \frac{8}{2} = 4$

5 10

6 11 12

7 13 14 15 16

Congestion window = Congestion threshold

Now Linear increment will happen

8 17 18 19 20 21

9 22 23 24 25 26 27

Window size=6

Congestion threshold = $\frac{\text{Window size}}{2} = \frac{6}{2} = 3$

10 25

11 26 27

12 28 29 30

threshold > Size of Window
Now linear increment will happen

13 31 32 33 34

Window size=4

Congestion threshold = $\frac{\text{Window size}}{2} = \frac{4}{2} = 2$

14	34				
15	35	36			

Congestion window = Congestion threshold

Now Linear increment will happen

16	37	38	39		
17	40	41	42	43	
18	44	**45**	46	47	48

Window size=5

Congestion threshold = $\frac{\text{Window size}}{2} = \frac{5}{2} = 2.5$

19 45

20 46 47

Congestion window = Congestion threshold

Now Linear increment will happen

21 48 49 50

Figure 15.37 *Congestion example*