Introduction to the Transport Layer

OBJECTIVES:

- To define process-to-process communication at the transport layer and compare it with host-to-host communication at the network layer.
- To discuss the addressing mechanism at the transport layer, to discuss port numbers, and to define the range of port numbers used for different purposes.
- To explain the packetizing issue at the transport layer: encapsulation and decapsulation of messages.
- To discuss multiplexing (many-to-one) and demultiplexing (one-to-many) services provided by the transport layer.
- To discuss flow control and how it can be achieved at the transport layer.

OBJECTIVES (*continued*):

- To discuss error control and how it can be achieved at the transport layer.
- To discuss congestion control and how it can be achieved at the transport layer.
- To discuss the connectionless and connection-oriented services at the transport layer and show their implementation using an FSM.
- To discuss the behavior of four generic transport-layer protocols and their applications: simple protocol, Stop-and-Wait protocol, Go-Back-N protocol, and Selective-Repeat protocol.
- To describe the idea of bidirectional communication at the transport layer using the piggybacking method.

Chapter Outline

13-1 TRANSPORT-LAYER SERVICES

As we discussed in Chapter 2, the transport layer is located between the network layer and the application layer. The transport layer is responsible for providing services to the application layer; it receives services from the network layer. In this section, we discuss the services that can be provided by a transport layer; in the next section, we discuss the principle beyond several transport layer protocols.

Topics Discussed in the Section

- ✓ **Process-to-Process Communication**
- ✓ **Addressing: Port Numbers**
- ✓ **Encapsulation and Decapsulation**
- ✓ **Multiplexing and Demultiplexing**
- ✓ **Flow Control**
- ✓ **Error Control**
- ✓ **Congestion Control**
- ✓ **Connectionless and Connection-Oriented Services**

Figure 13.1 *Network layer versus transport layer*

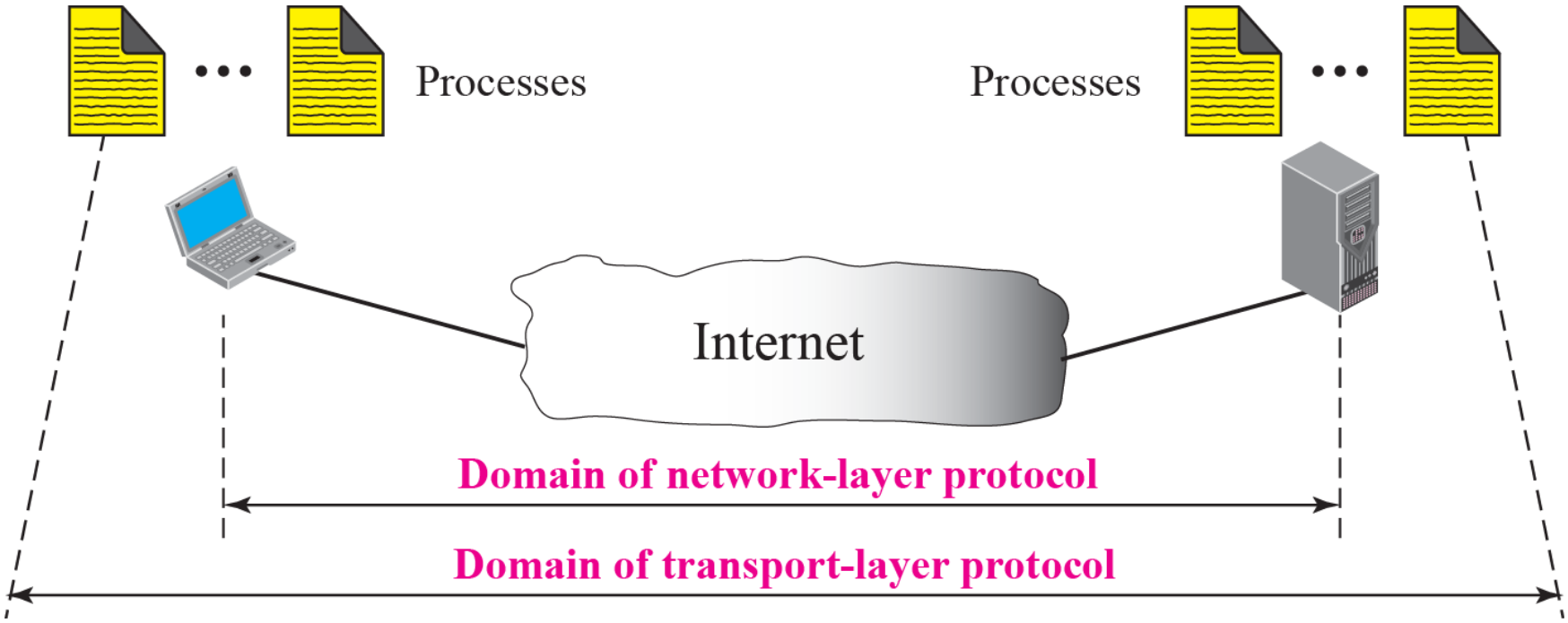


Figure 13.2 *Port numbers*

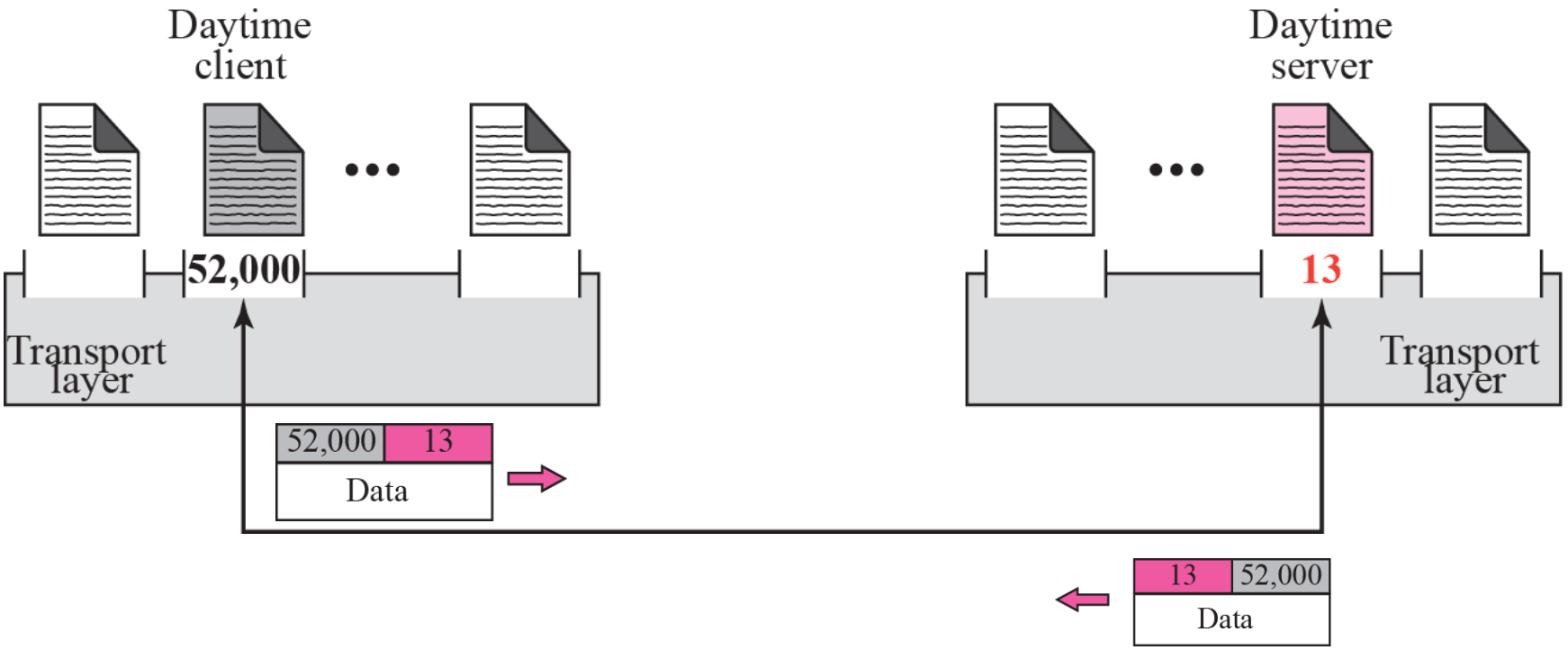


Figure 13.3 *IP addresses versus port numbers*

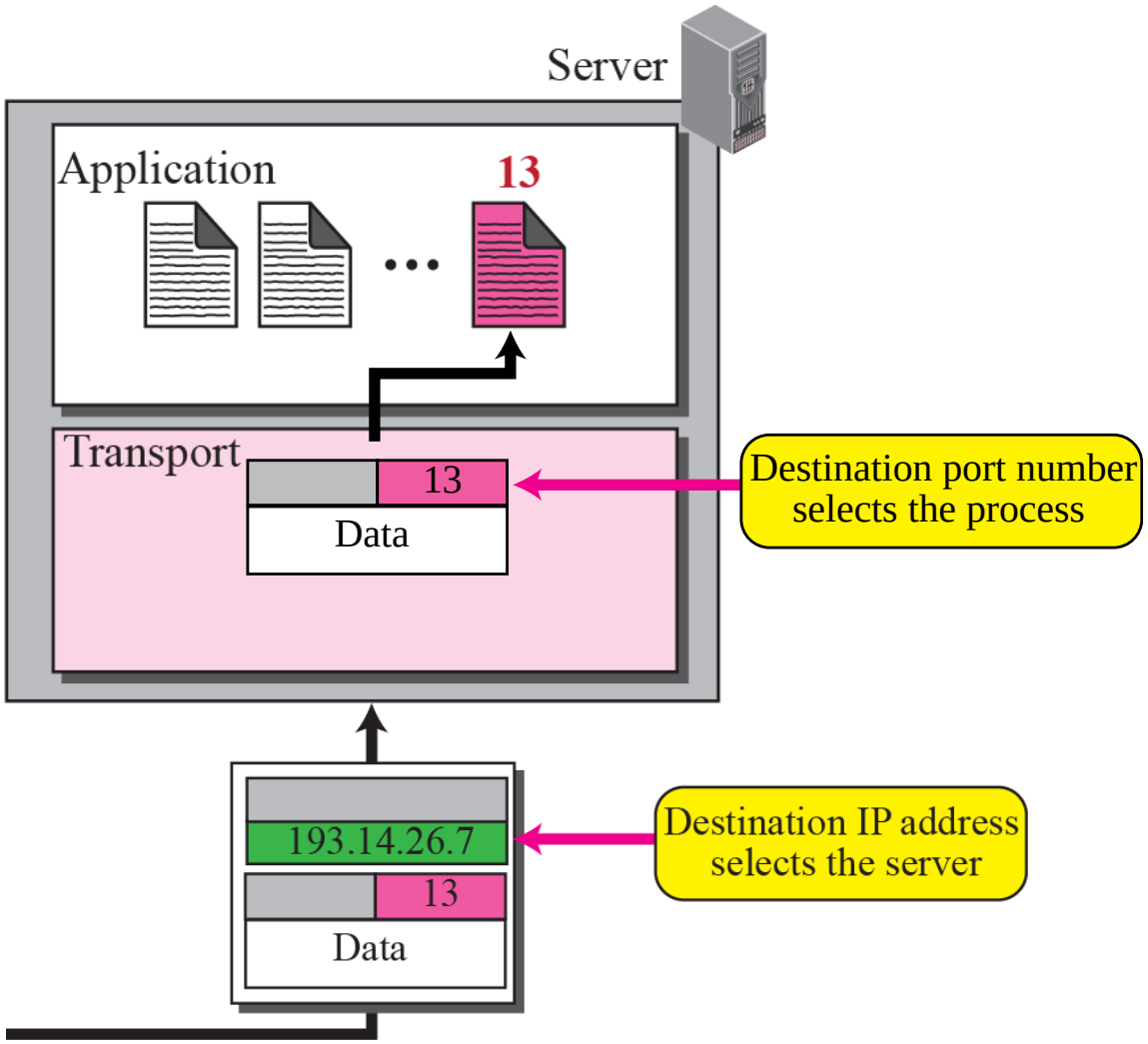
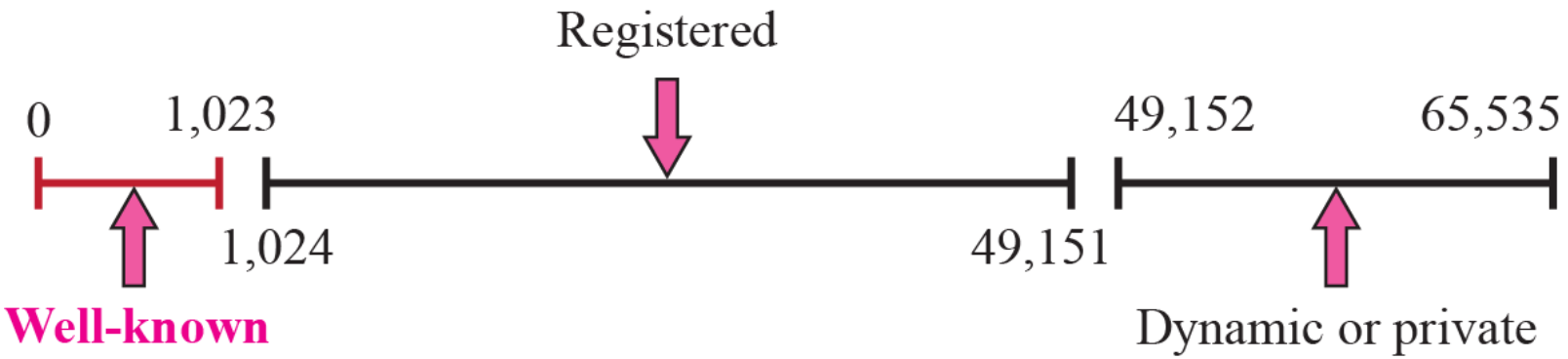


Figure 13.4 *ICANN ranges*



Note

The well-known port numbers are less than 1,024.

Figure 13.5 *Socket address*

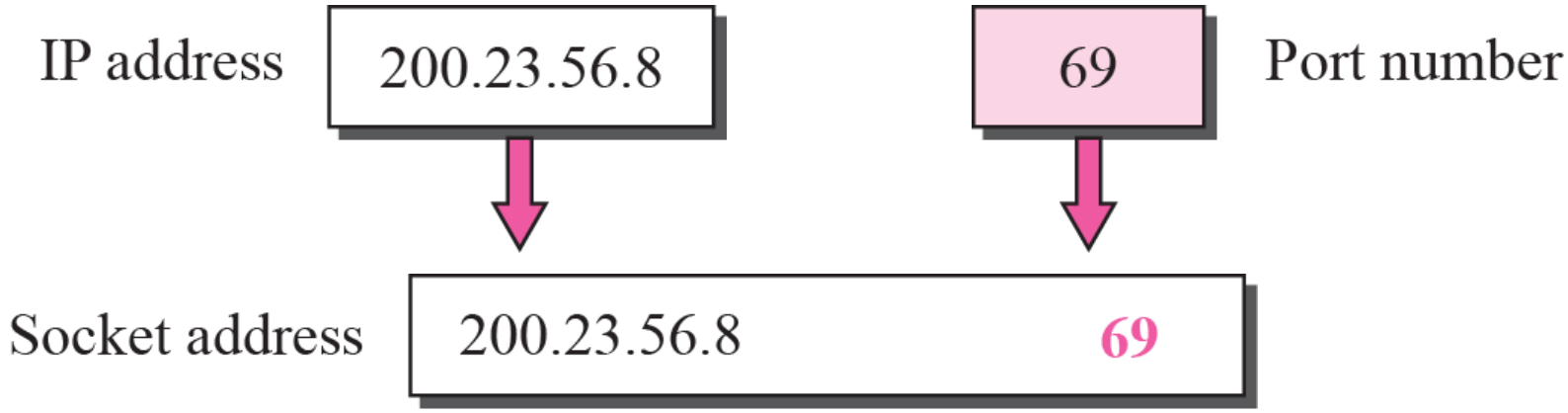


Figure 13.6 *Encapsulation and decapsulation*

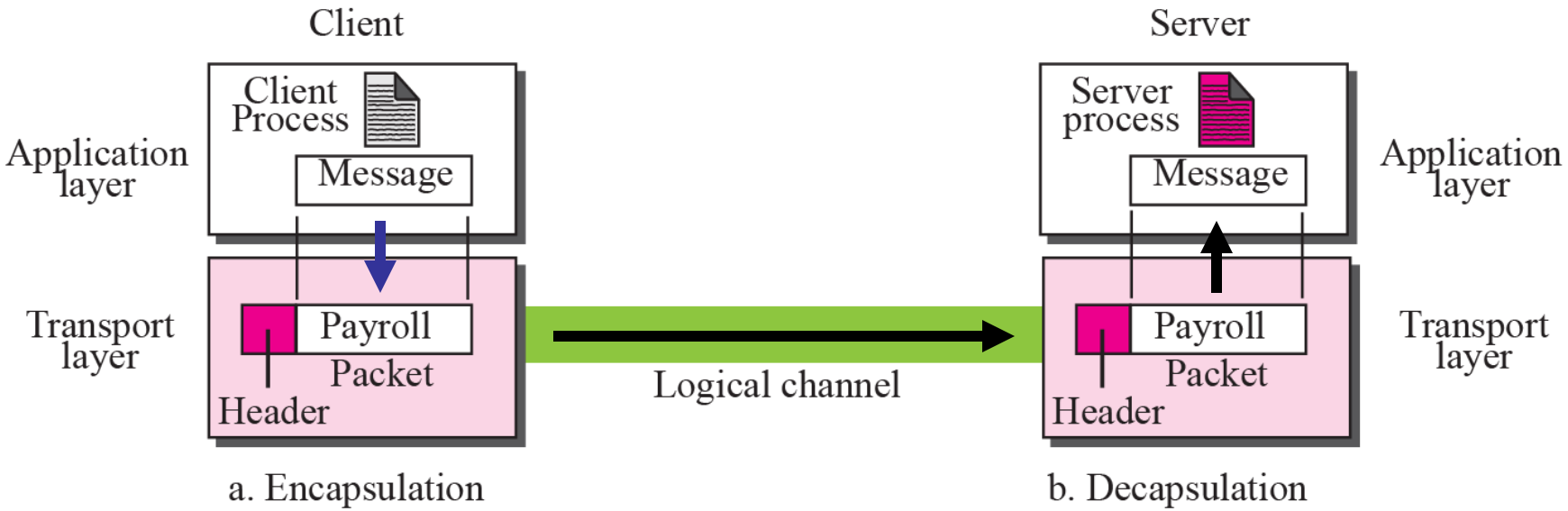


Figure 13.7 *Multiplexing and demultiplexing*

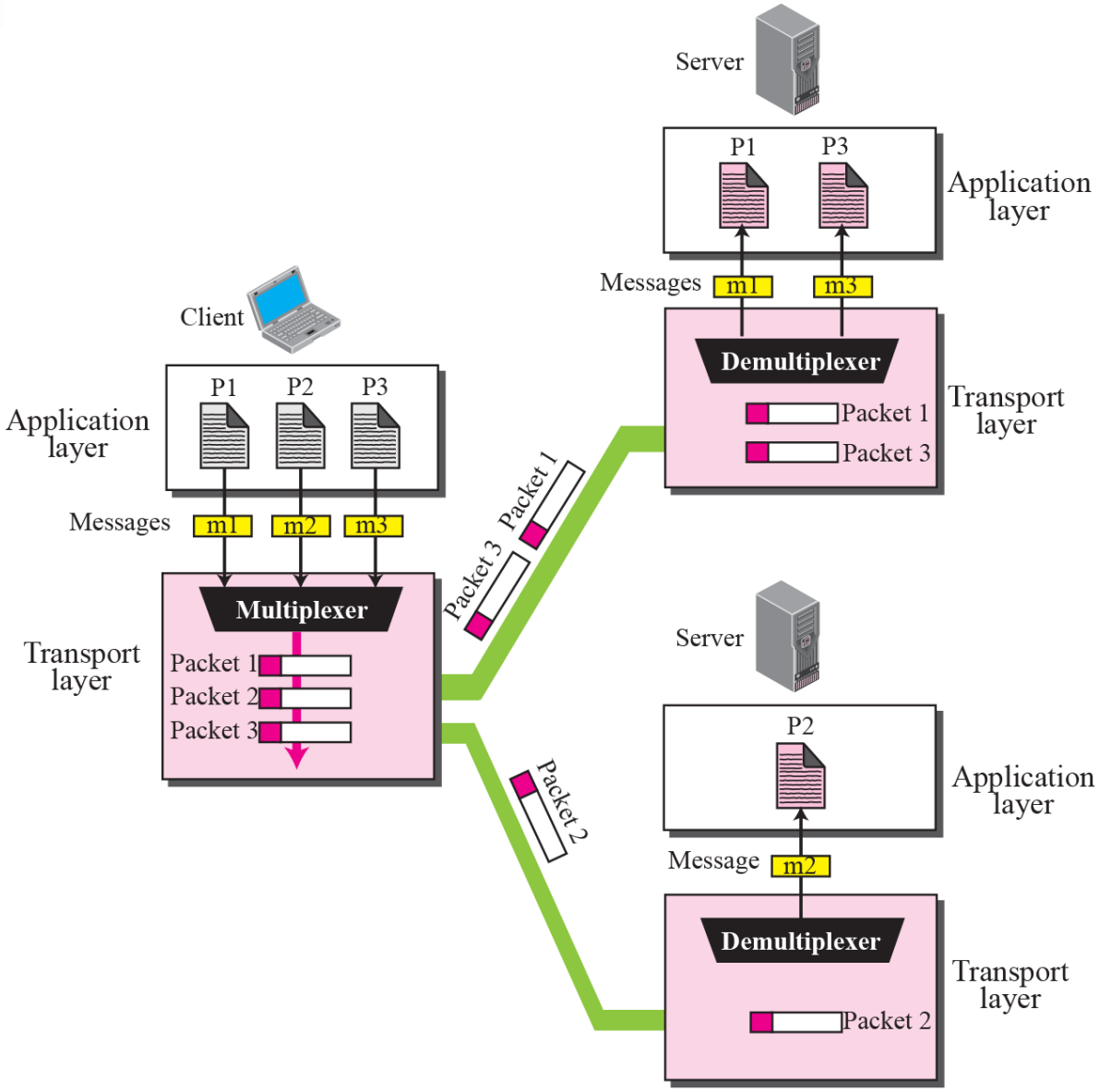
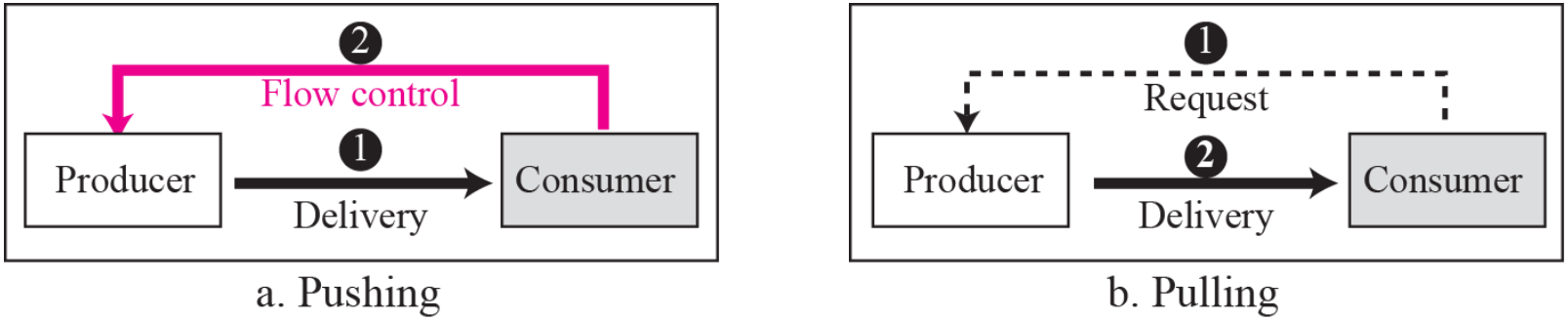


Figure 13.8 *Pushing or pulling*



a. Pushing

b. Pulling

Figure 13.9 *Flow control at the transport layer*

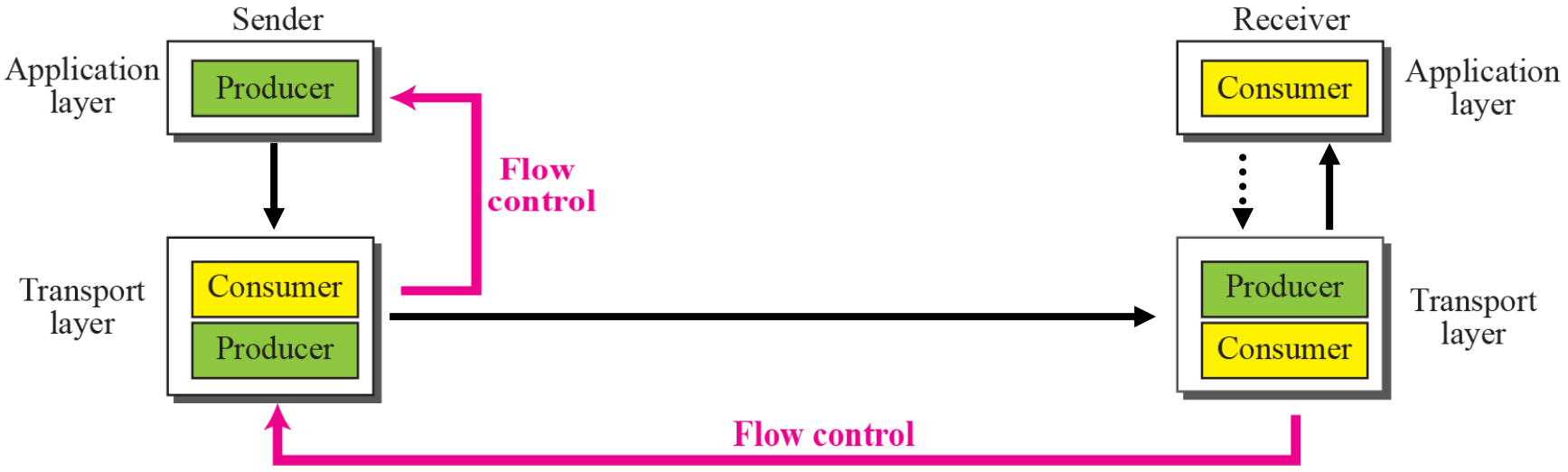
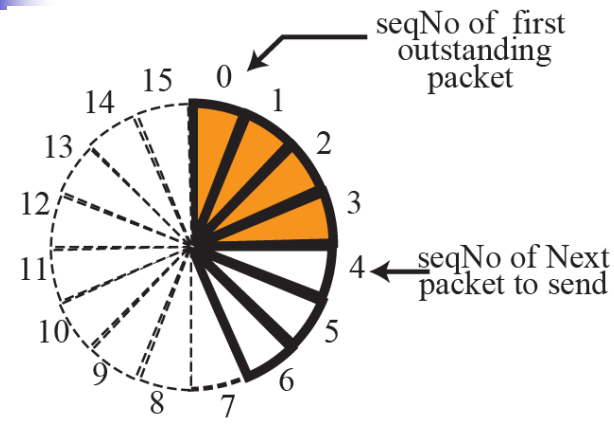
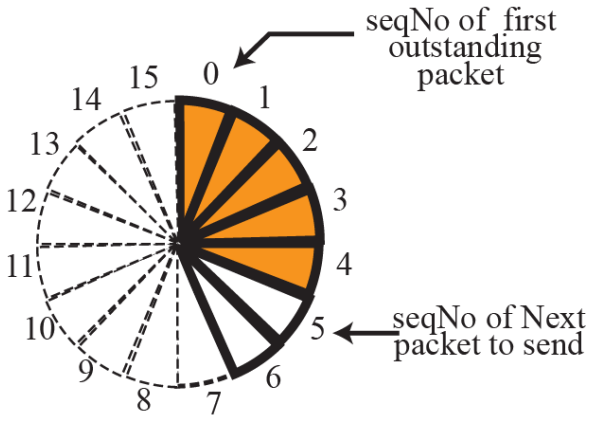


Figure 13.10 *Error control at the transport layer*

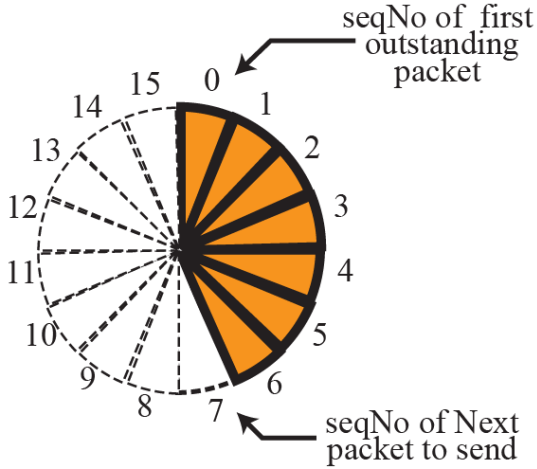
Figure 13.11 *Sliding window in circular format*



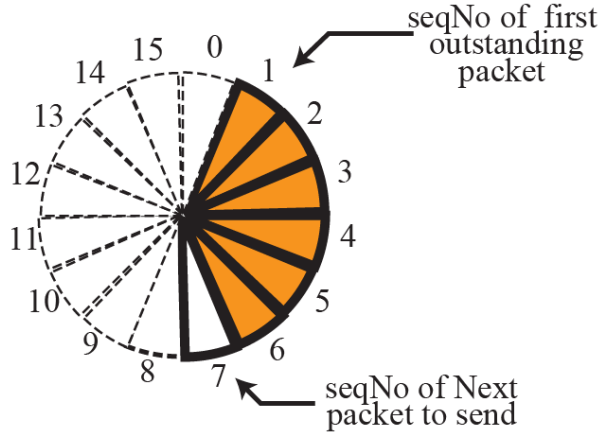
a. Four packets have been sent



b. Five packets have been sent



c. Seven packets have been sent
window is full



d. Packet 0 has been acknowledged,
window slides

Figure 13.12 *Sliding window in linear format*

a. Four packets have been sent

b. Five packets have been sent

c. Seven packets have been sent
window is full

d. Packet 0 have been acknowledged
and window slid

Figure 13.13 *Connectionless service*

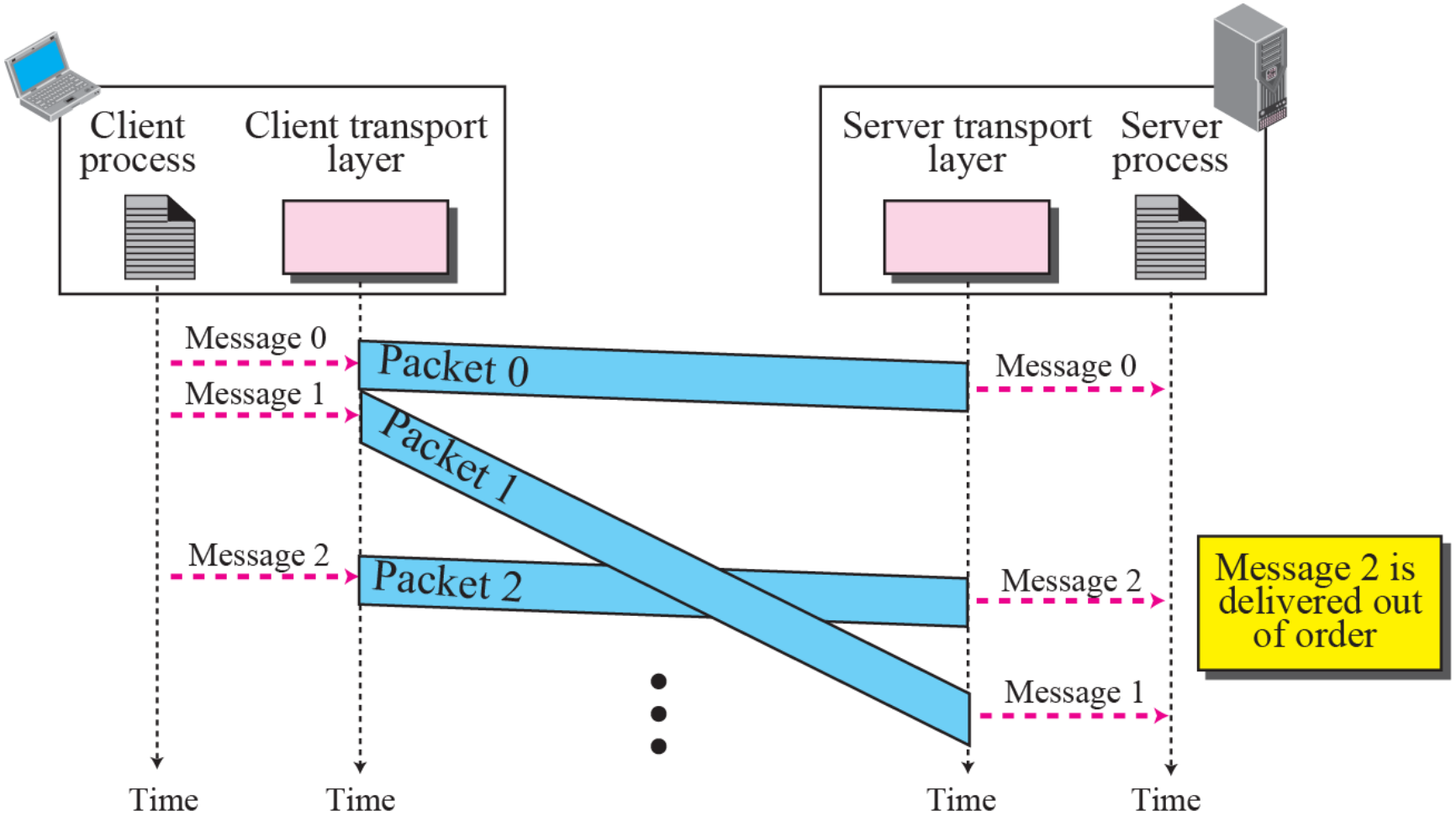


Figure 13.14 *Connection-oriented service*

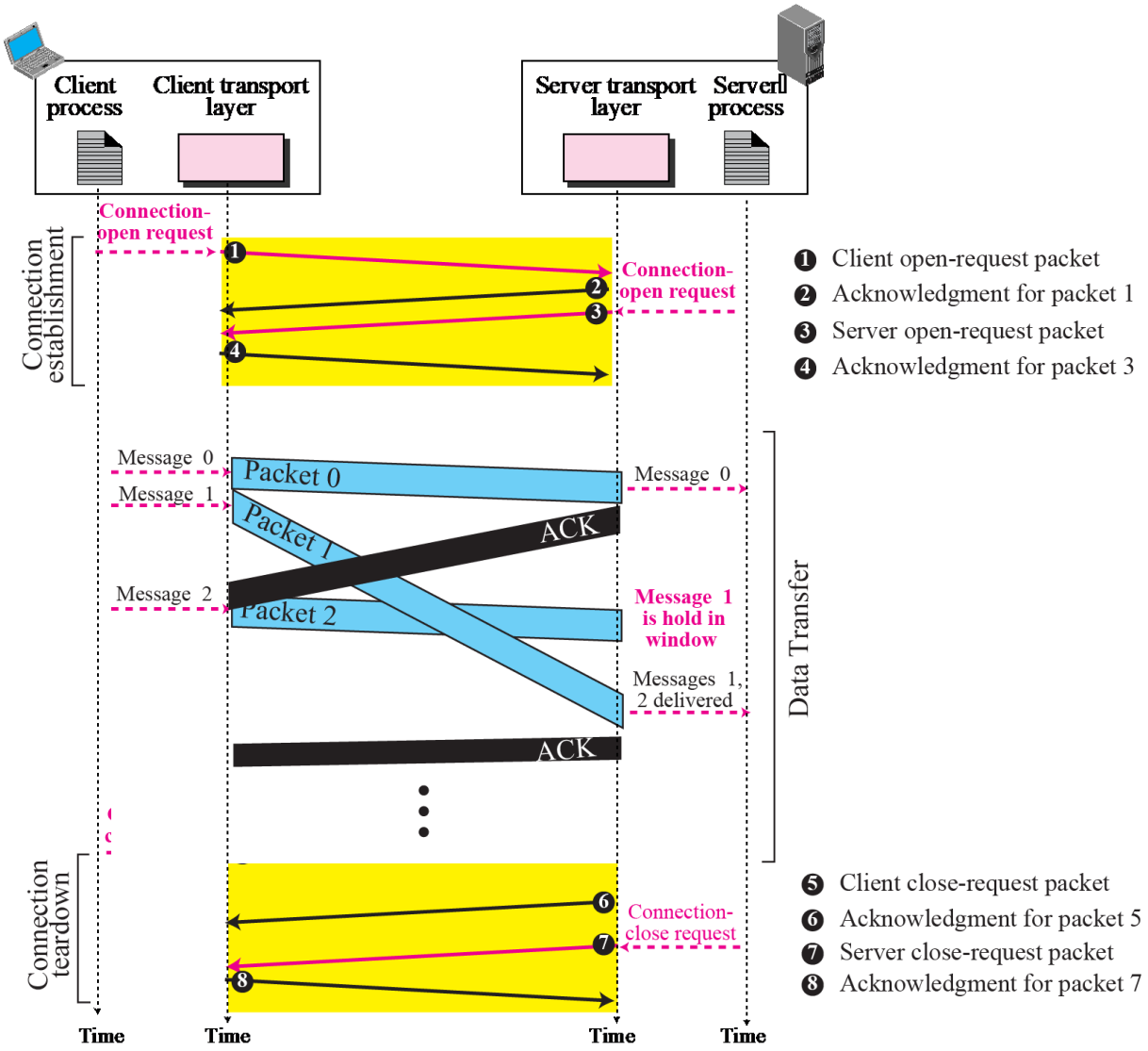
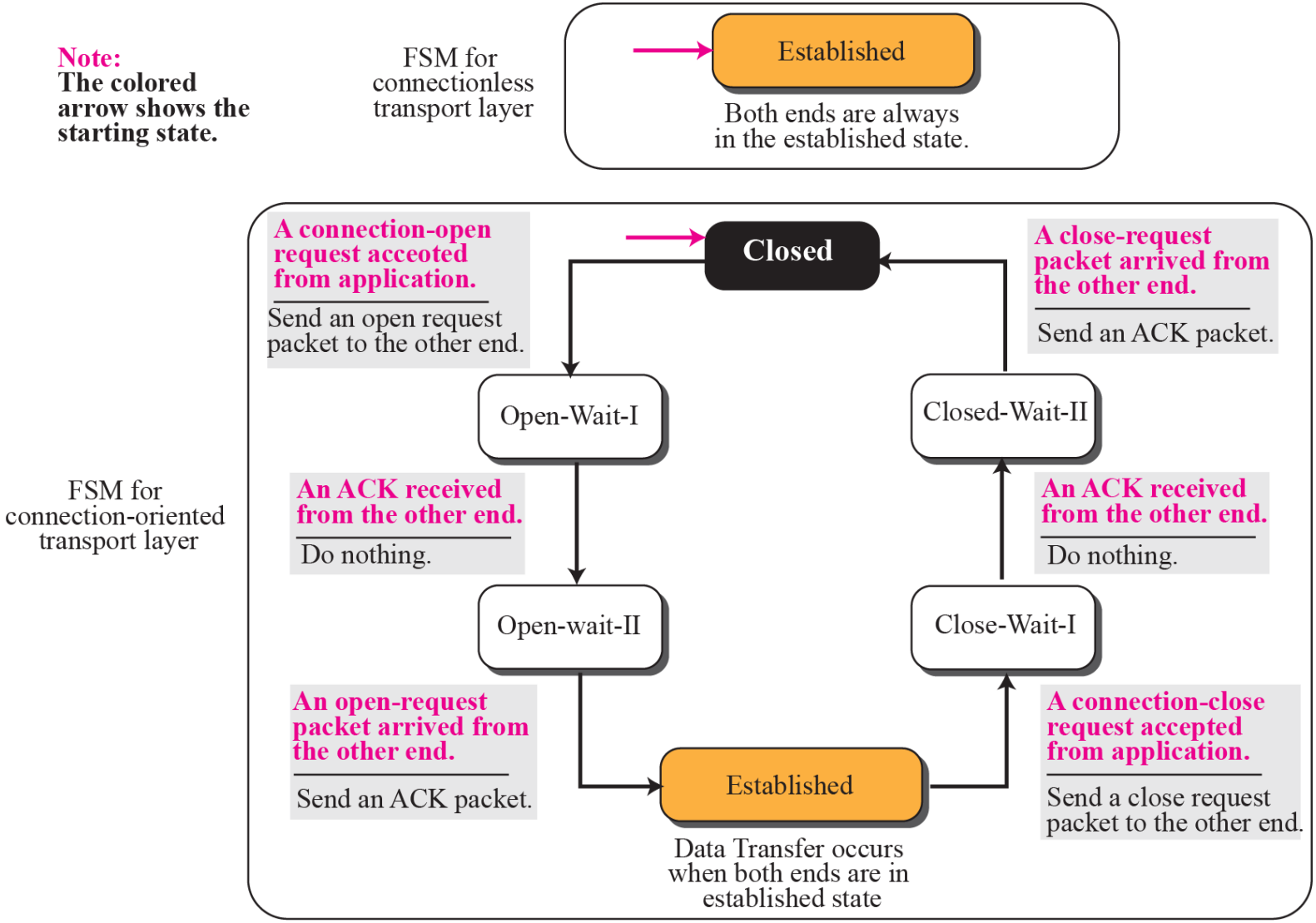


Figure 13.15 *Connectionless and connection-oriented services as FSMs*



13-2 TRANSPORT-LAYER PROTOCOLS

We can create a transport-layer protocol by combining a set of services described in the previous sections. To better understand the behavior of these protocols, we start with the simplest one and gradually add more complexity. The TCP/IP protocol uses a transport layer protocol that is either a modification or a combination of some of these protocols.

Topics Discussed in the Section

- ✓ **Simple Protocol**
- ✓ **Stop-and-Wait Protocol**
- ✓ **Go-Back-*N* Protocol**
- ✓ **Selective-Repeat Protocol**
- ✓ **Bidirectional Protocols: Piggybacking**

Figure 13.16 *Simple protocol*

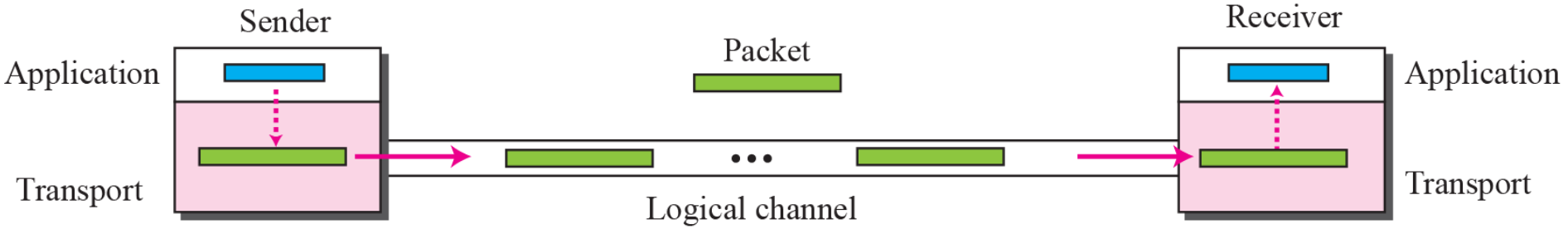
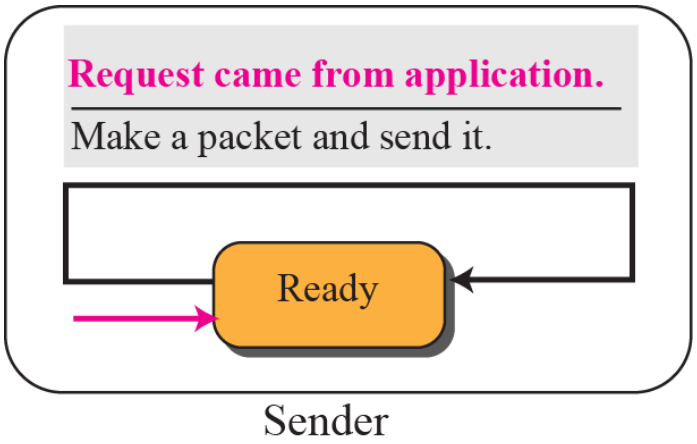
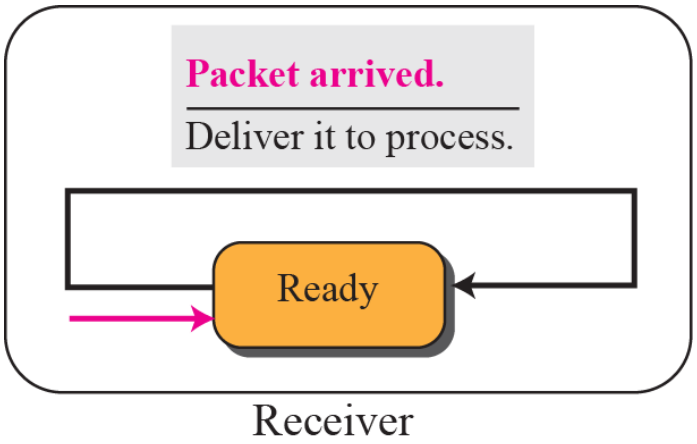


Figure 13.17 *FSMs for simple protocol*





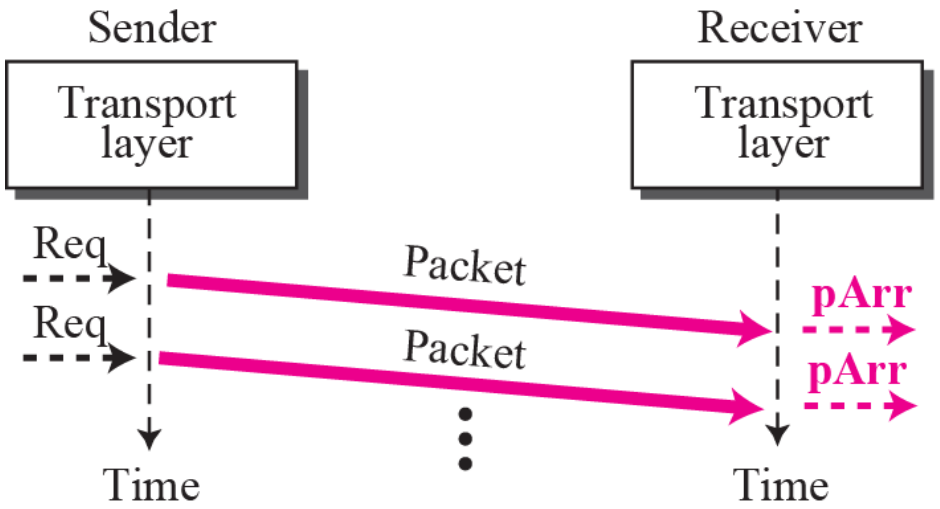
Note

The simple protocol is a connectionless protocol that provides neither flow nor error control.

Example 13.3

Figure 13.18 shows an example of communication using this protocol. It is very simple. The sender sends packets one after another without even thinking about the receiver.

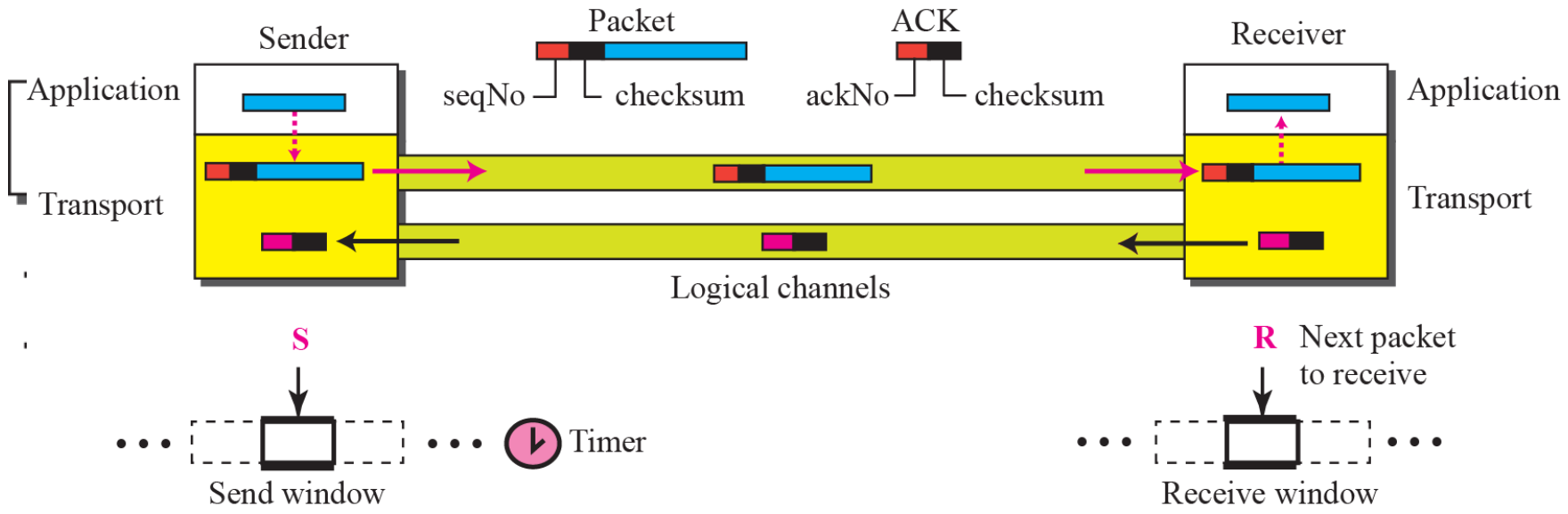
Figure 13.18 *Example 13.3*



Events:

Req: Request came from process
pArr: Packet arrived

Figure 13.19 *Stop-and-wait protocol*



Note

In Stop-and-Wait protocol, flow control is achieved by forcing the sender to wait for an acknowledgment, and error control is achieved by discarding corrupted packets and letting the sender resend unacknowledged packets when the timer expires.

Note

In the Stop-and-Wait protocol, we can use a 1-bit field to number the packets. The sequence numbers are based on modulo-2 arithmetic.

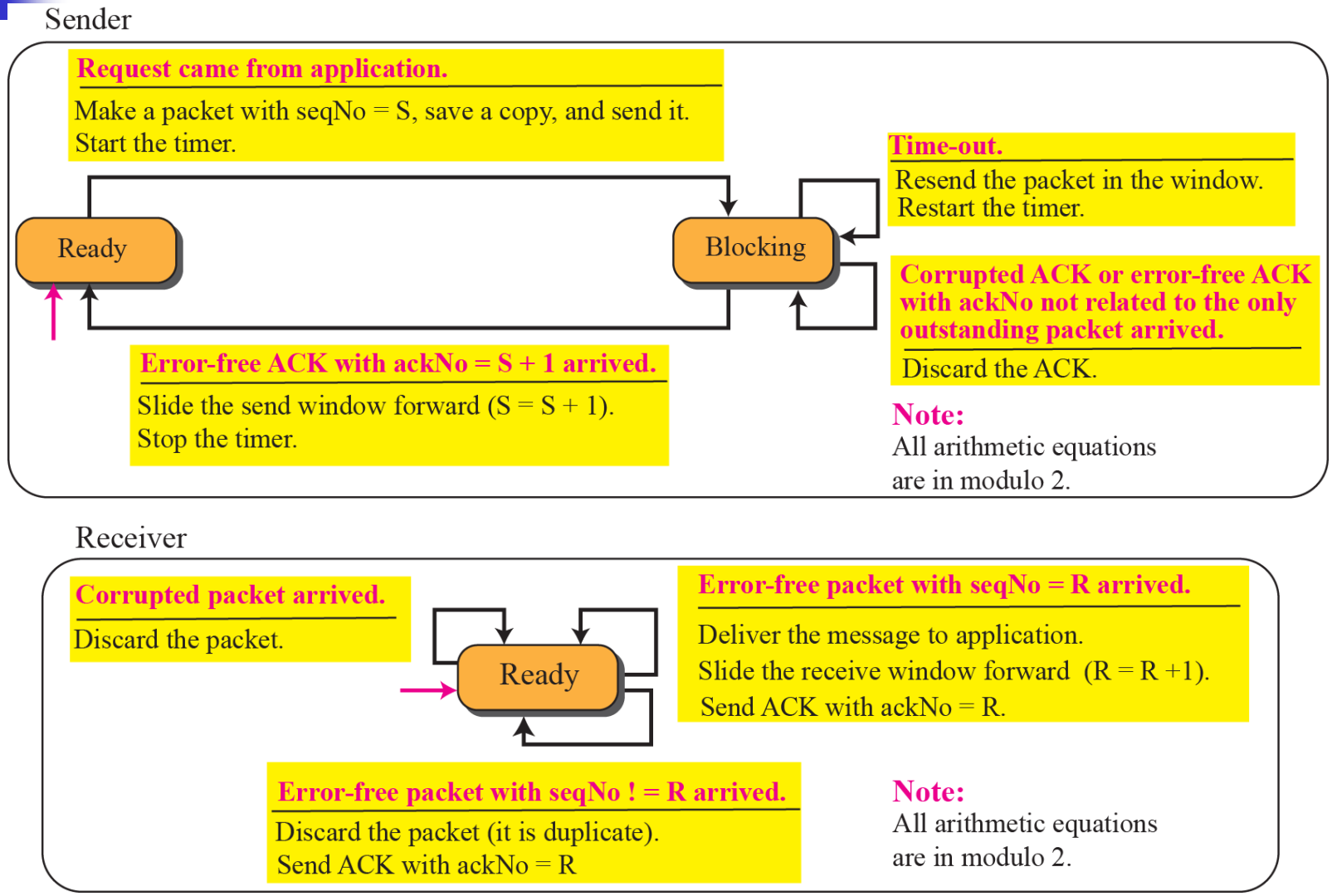
Note

In the Stop-and-Wait protocol, the acknowledgment number always announces in modulo-2 arithmetic the sequence number of the next packet expected.

Note

All calculation in the Stop-and-Wait protocol is in modulo 2.

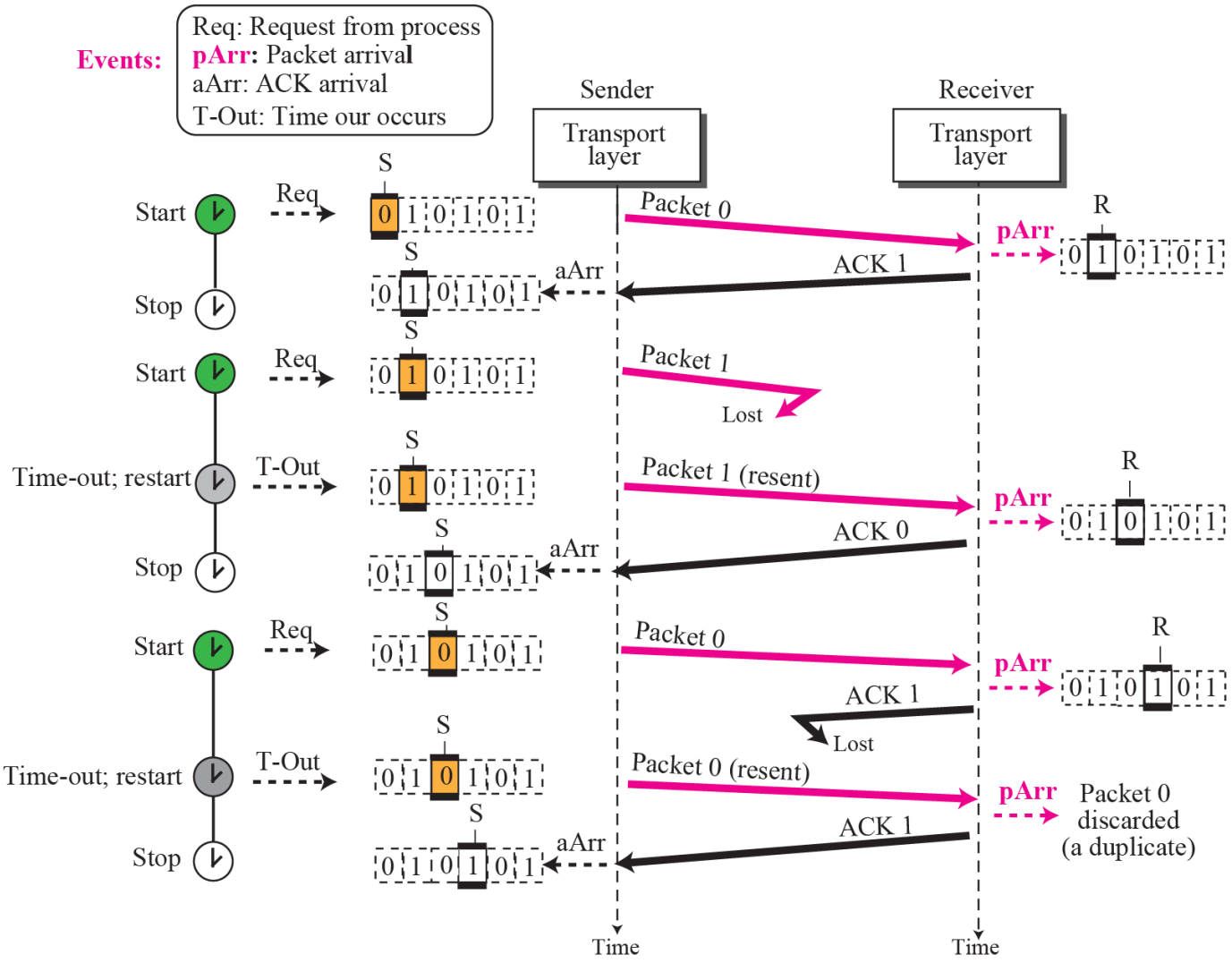
Figure 13.20 *FSMs for stop-and-wait protocol*



Example 13.4

Figure 13.21 shows an example of Stop-and-Wait protocol. Packet 0 is sent and acknowledged. Packet 1 is lost and resent after the time-out. The resent packet 1 is acknowledged and the timer stops. Packet 0 is sent and acknowledged, but the acknowledgment is lost. The sender has no idea if the packet or the acknowledgment is lost, so after the time-out, it resends packet 0, which is acknowledged.

Figure 13.21 *Example 13.4*



Example 13.5

Assume that, in a Stop-and-Wait system, the bandwidth of the line is 1 Mbps, and 1 bit takes 20 milliseconds to make a round trip. What is the bandwidth-delay product? If the system data packets are 1,000 bits in length, what is the utilization percentage of the link?

Solution

The bandwidth-delay product is $(1 \times 10^6) \times (20 \times 10^{-3}) = 20{,}000$ bits. The system can send 20,000 bits during the time it takes for the data to go from the sender to the receiver and the acknowledgment to come back. However, the system sends only 1,000 bits. We can say that the link utilization is only 1,000/20,000, or 5 percent. For this reason, for a link with a high bandwidth or long delay, the use of Stop-and-Wait wastes the capacity of the link.

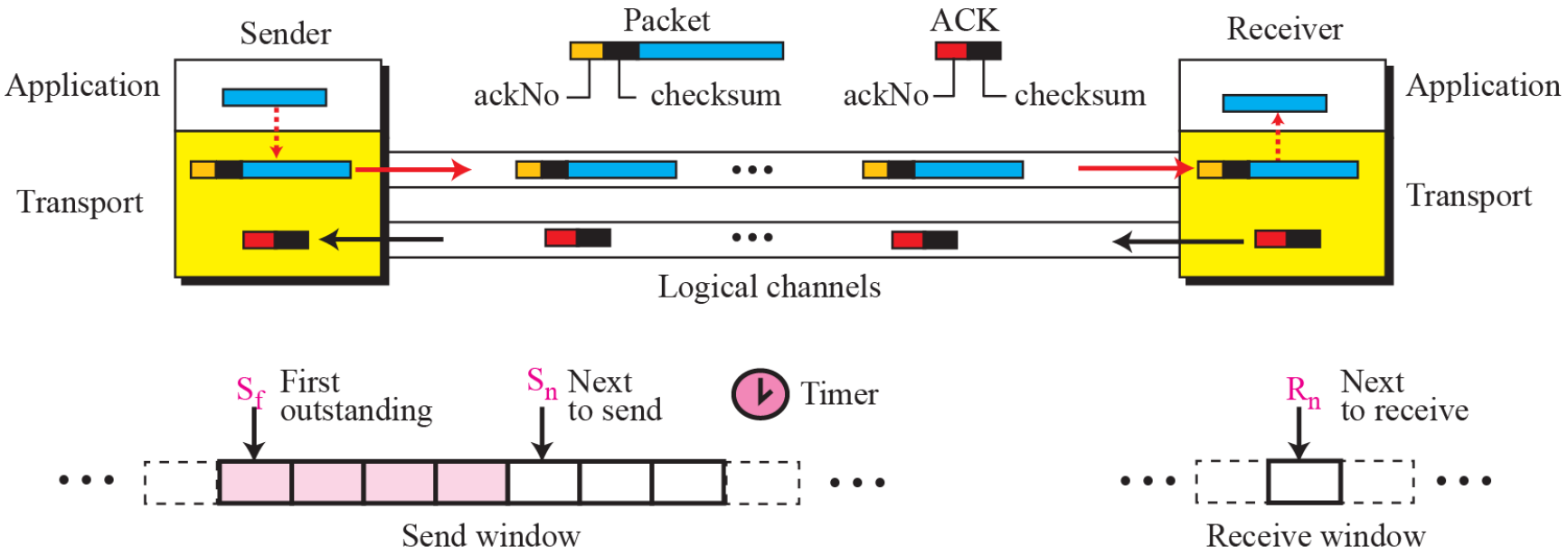
Example 13.6

What is the utilization percentage of the link in Example 13.5 if we have a protocol that can send up to 15 packets before stopping and worrying about the acknowledgments?

Solution

The bandwidth-delay product is still 20,000 bits. The system can send up to 15 packets or 15,000 bits during a round trip. This means the utilization is 15,000/20,000, or 75 percent. Of course, if there are damaged packets, the utilization percentage is much less because packets have to be resent.

Figure 13.22 *Go-Back-N protocol*



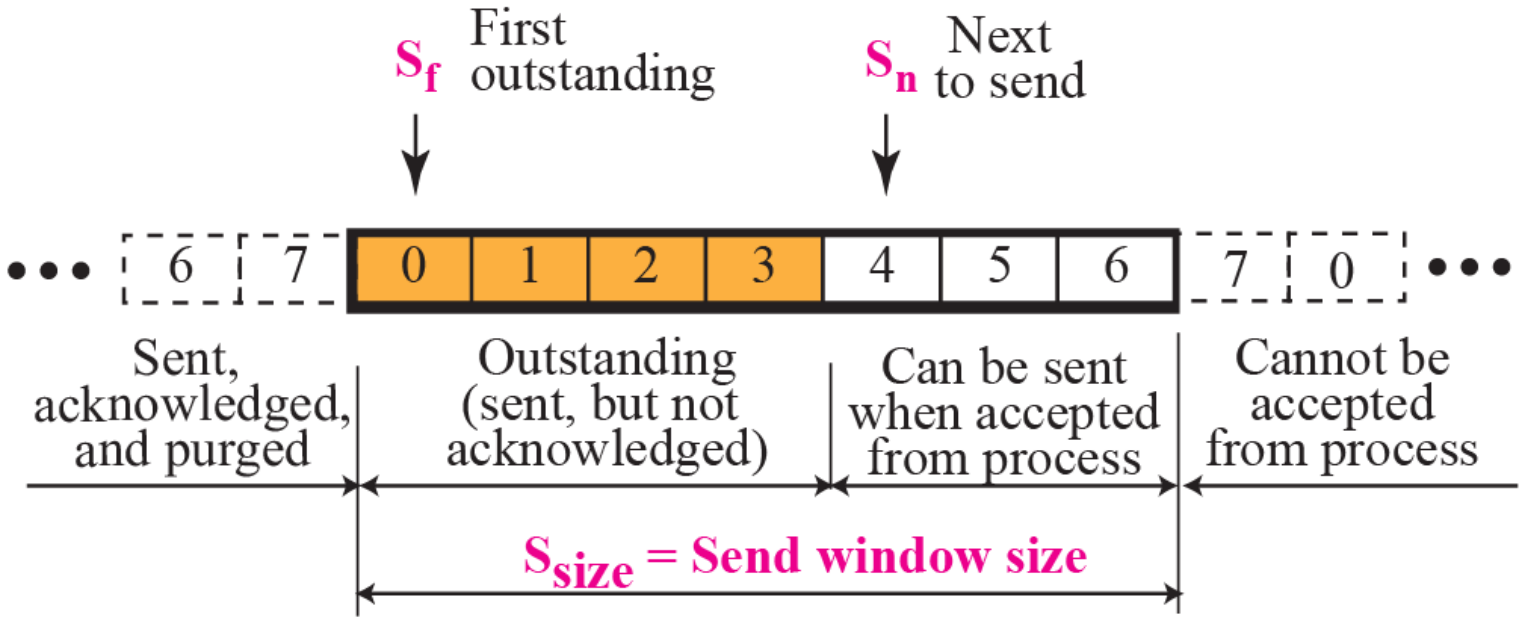
Note

In the Go-Back-N Protocol, the sequence numbers are modulo 2^m, where m is the size of the sequence number field in bits.

Note

In the Go-Back-N protocol, the acknowledgment number is cumulative and defines the sequence number of the next packet expected to arrive.

Figure 13.23 *Send window for Go-Back-N*



Note

The send window is an abstract concept defining an imaginary box of maximum size = $2^m - 1$ with three variables: S_f, S_n, and S_{size}.

Note

The send window can slide one or more slots when an error-free ACK with ackNo between S_f and S_n (in modular arithmetic) arrives.

Figure 13.24 *Sliding the send window*

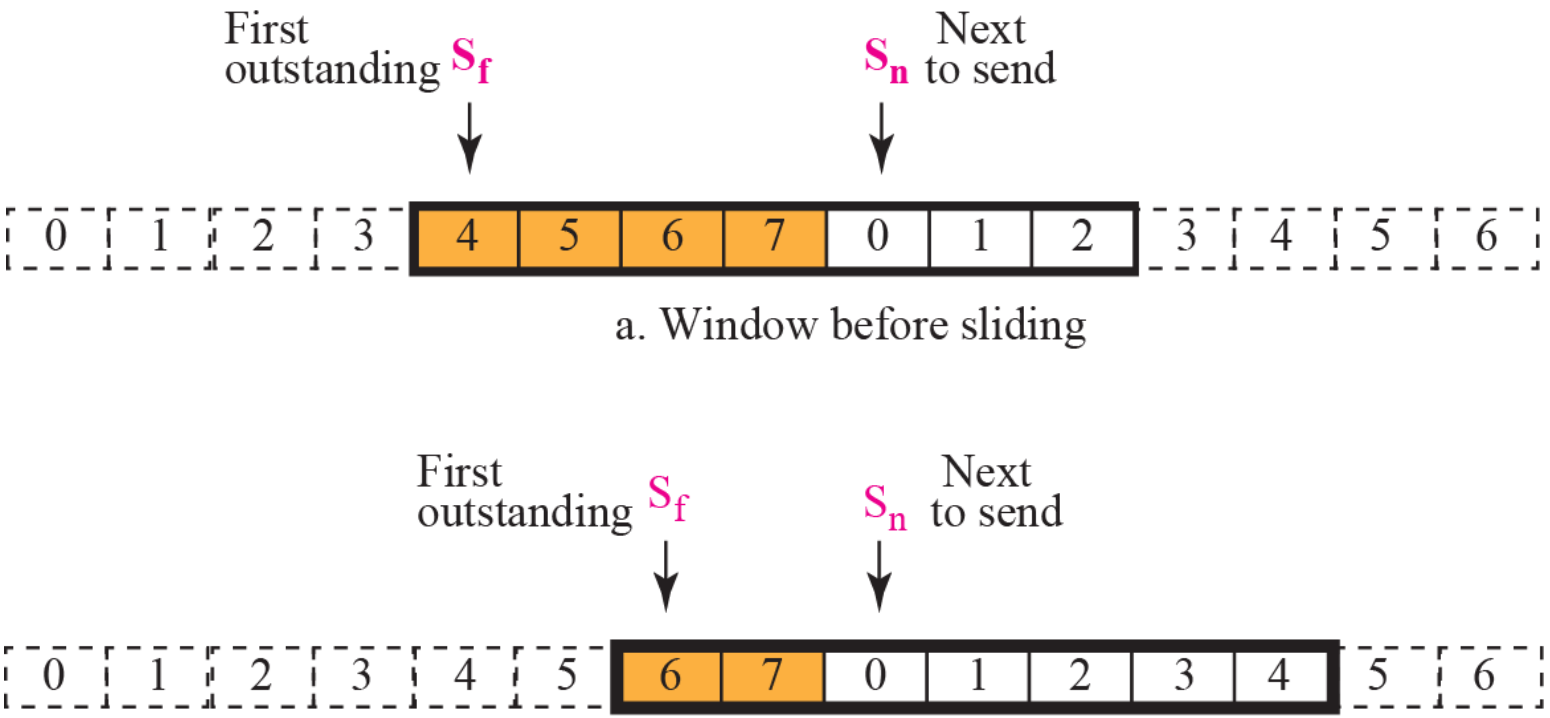
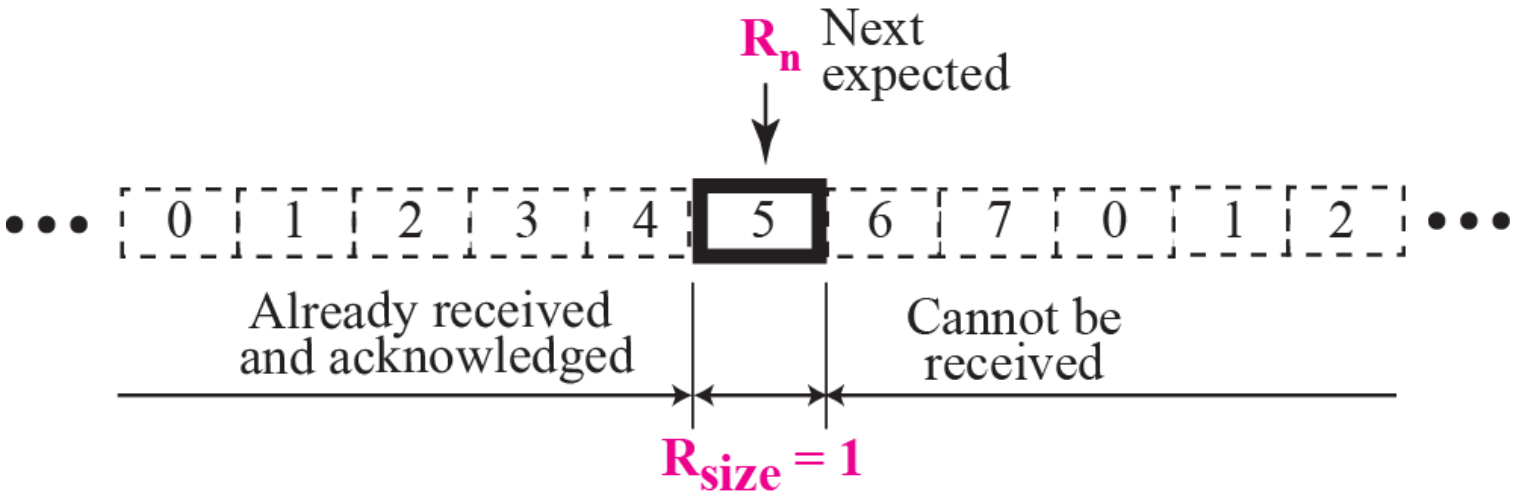


Figure 13.25 *Receive window for Go-Back-N*



Note

The receive window is an abstract concept defining an imaginary box of size 1 with one single variable R_n.

The window slides when a correct packet has arrived; sliding occurs one slot at a time.

Figure 13.26 *FSMs for Go-Back-N*

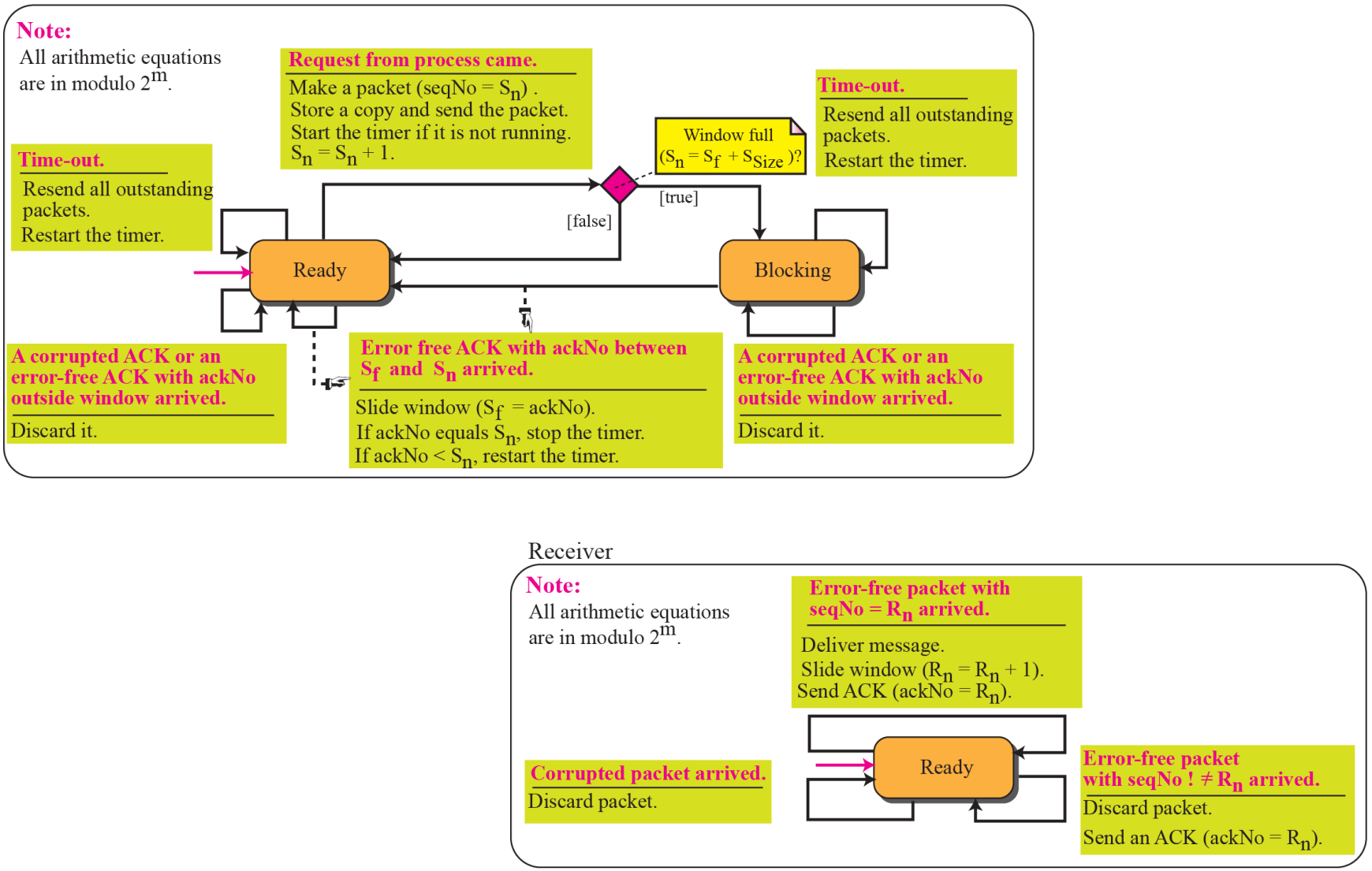
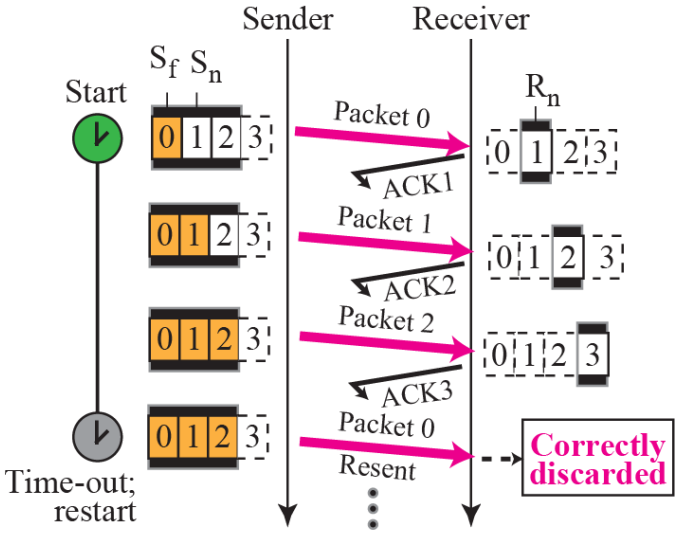
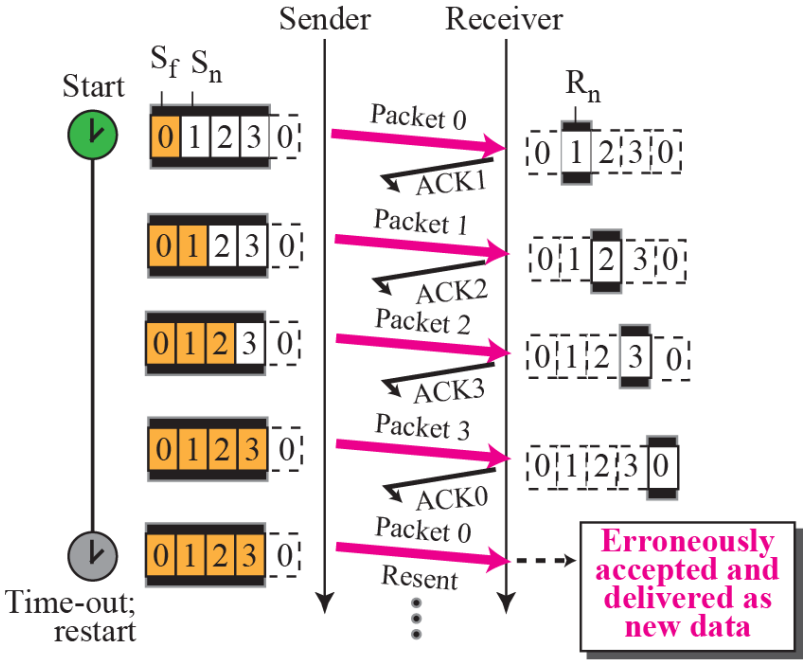


Figure 13.27 *Send window size for Go-Back-N*



a. Send window of size < 2^m



b. Send window of size = 2^m

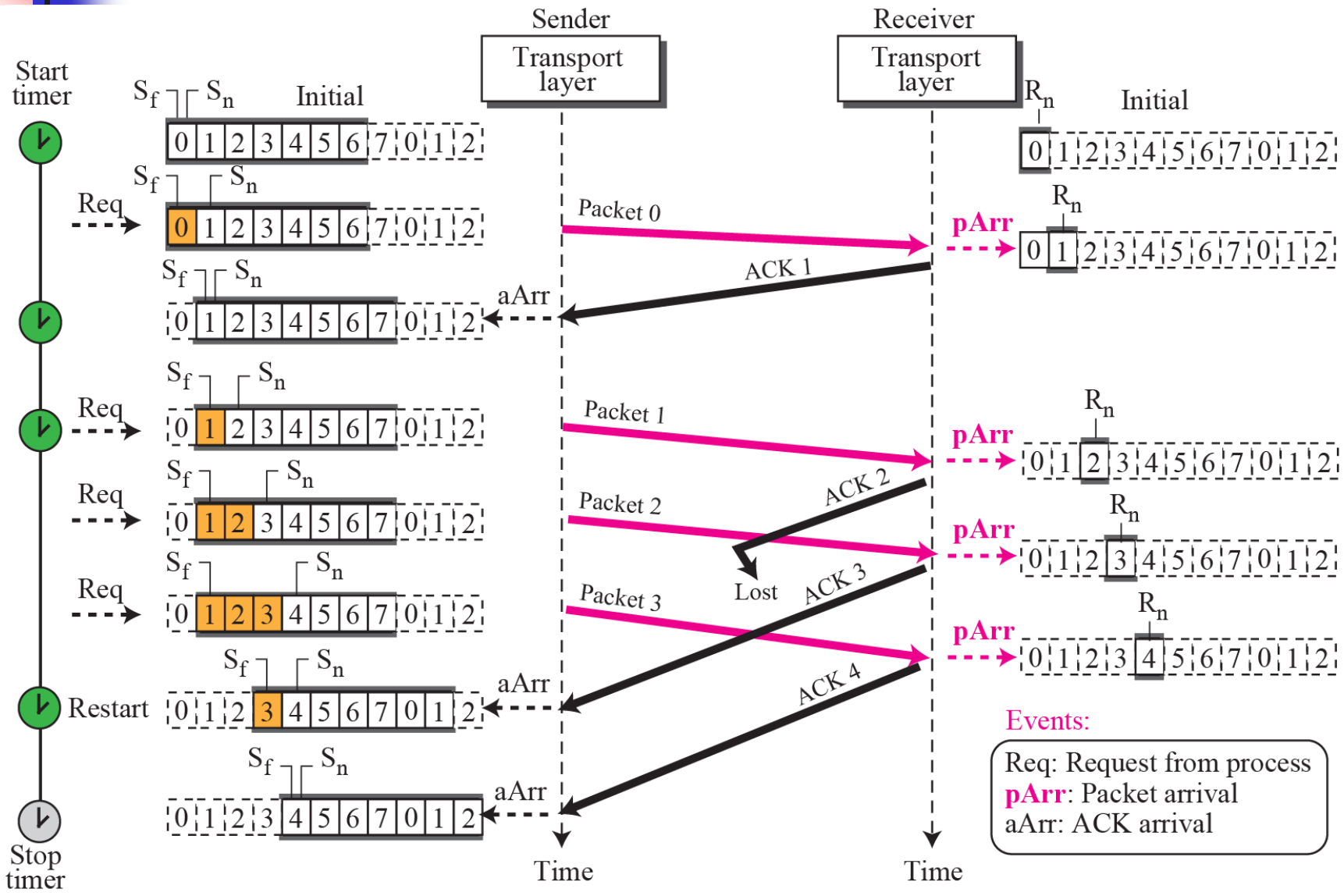
Note

In the Go-Back-N protocol, the size of the send window must be less than 2m; the size of the receive window is always 1.

Example 13.7

Figure 13.28 shows an example of Go-Back-N. This is an example of a case where the forward channel is reliable, but the reverse is not. No data packets are lost, but some ACKs are delayed and one is lost. The example also shows how cumulative acknowledgments can help if acknowledgments are delayed or lost.

Figure 13.28 *Example 13.7*



Example 13.8

Figure 13.29 shows what happens when a packet is lost. Packets 0, 1, 2, and 3 are sent. However, packet 1 is lost. The receiver receives packets 2 and 3, but they are discarded because they are received out of order (packet 1 is expected). When the receiver receives packets 2 and 3, it sends ACK1 to show that it expects to receive packet 1. However, these ACKs are not useful for the sender because the ackNo is equal S_f , not greater that S_f . So the sender discards them. When the time-out occurs, the sender resends packets 1, 2, and 3, which are acknowledged..

Figure 13.29 *Example 13.8*

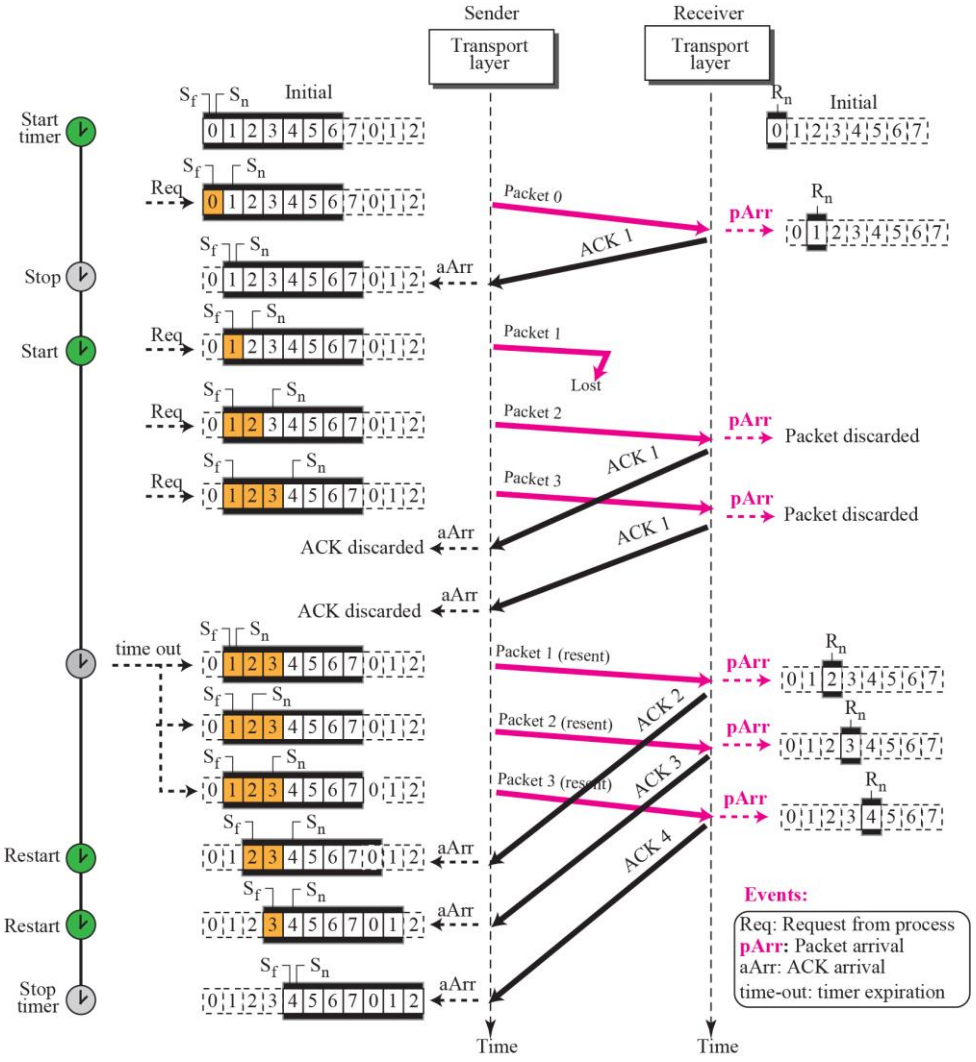


Figure 13.30 *Outline of Selective-Repeat*

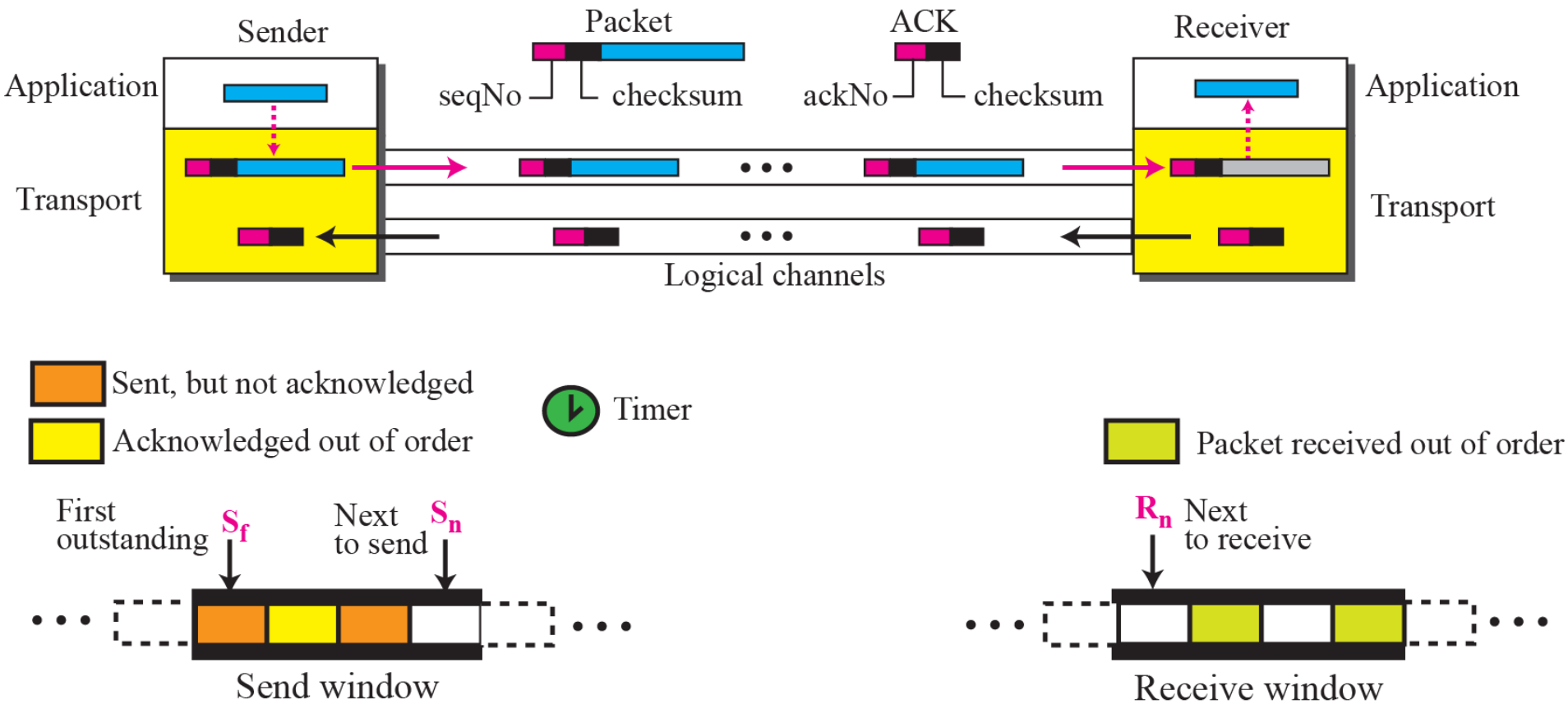


Figure 13.31 *Send window for Selective-Repeat protocol*

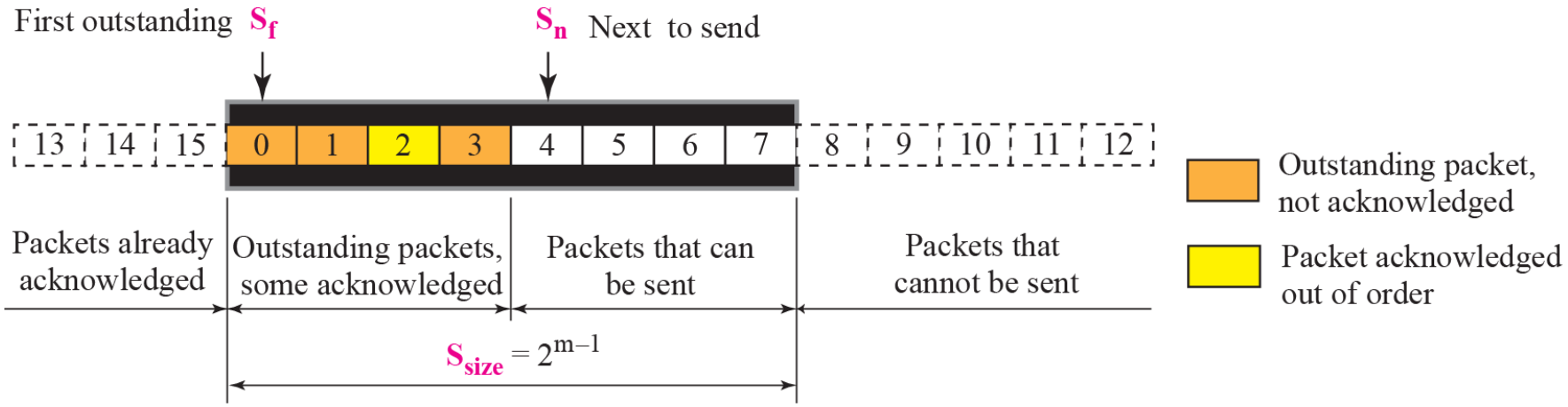
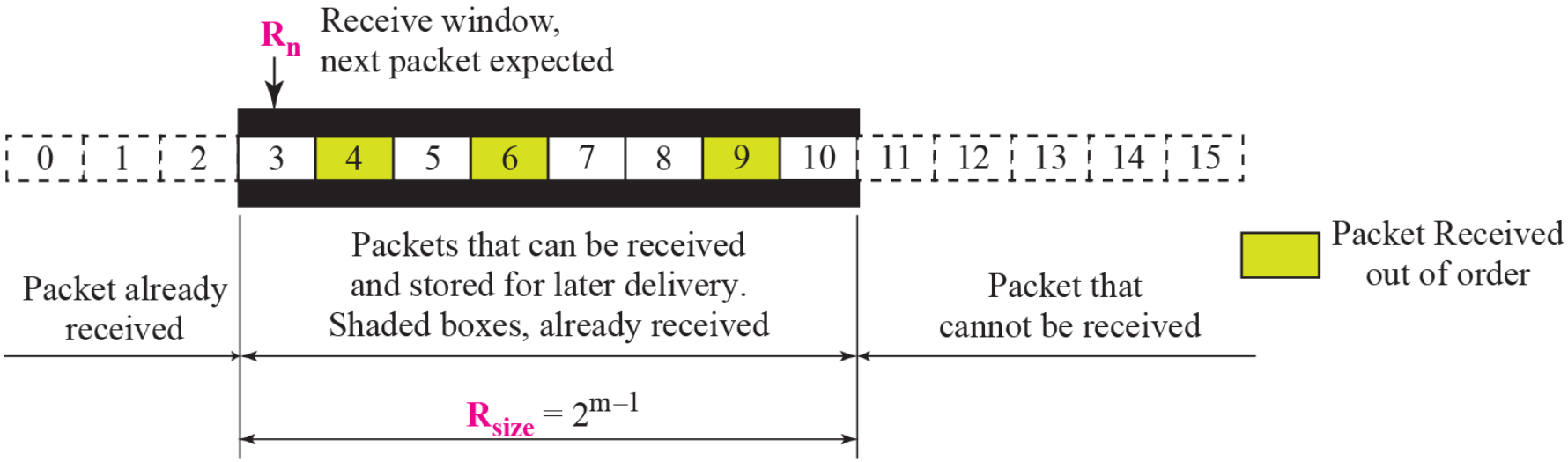


Figure 13.32 *Receive window for Selective-Repeat protocol*



Note

In the Selective-Repeat protocol, an acknowledgment number defines the sequence number of the error-free packet received.

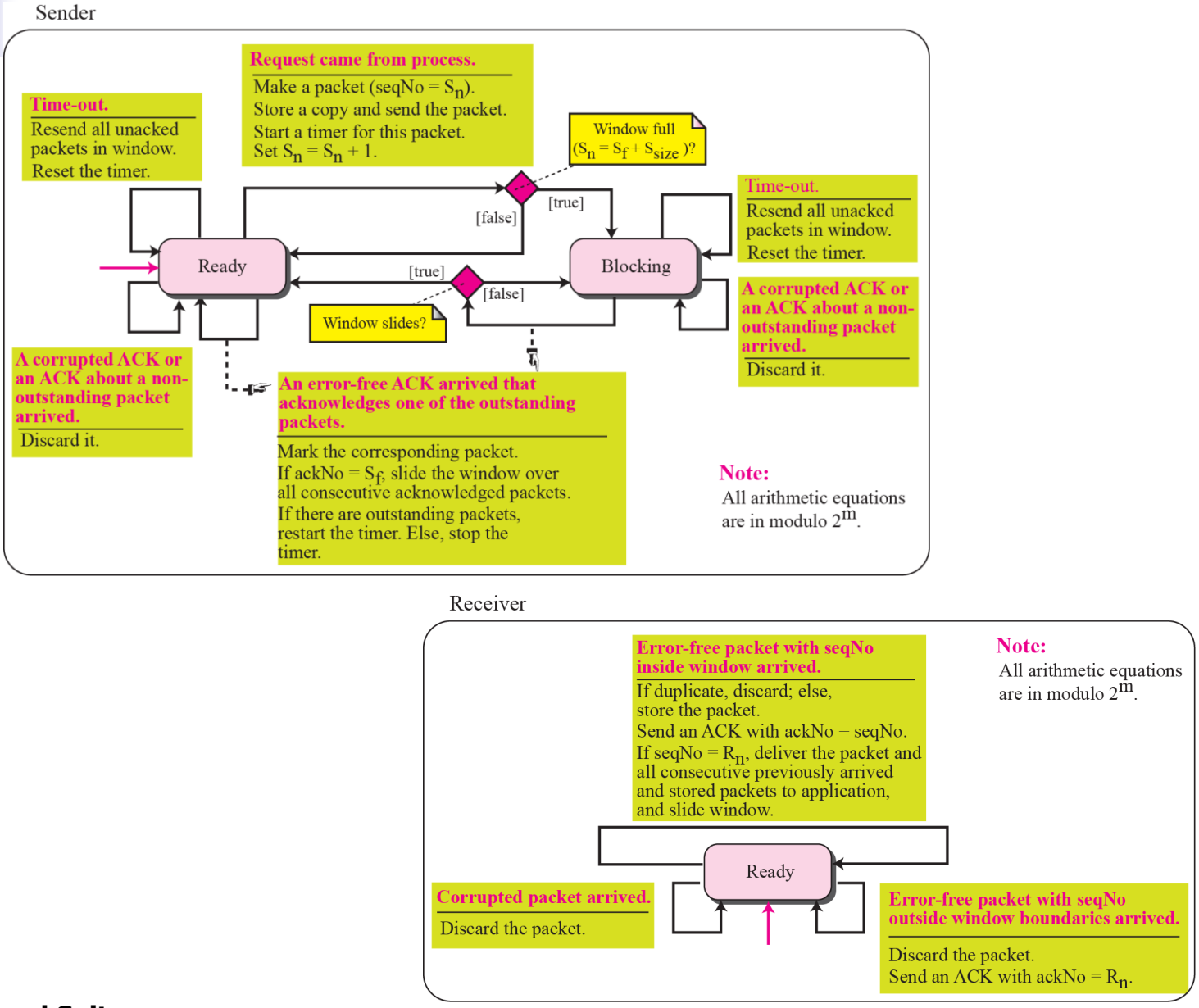
Example 13.9

Assume a sender sends 6 packets: packets 0, 1, 2, 3, 4, and 5. The sender receives an ACK with ackNo = 3. What is the interpretation if the system is using GBN or SR?

Solution

If the system is using GBN, it means that packets 0, 1, and 2 have been received uncorrupted and the receiver is expecting packet 3. If the system is using SR, it means that packet 3 has been received uncorrupted; the ACK does not say anything about other packets.

Figure 13.33 *FSMs for SR protocol*



Example 13.10

This example is similar to Example 3.8 (Figure 13.29) in which packet 1 is lost. We show how Selective-Repeat behaves in this case. Figure 13.34 shows the situation. At the sender, packet 0 is transmitted and acknowledged. Packet 1 is lost. Packets 2 and 3 arrive out of order and are acknowledged. When the timer times out, packet 1 (the only unacknowledged packet) is resent and is acknowledged. The send window then slides.

Figure 13.34 *Example 13.10*

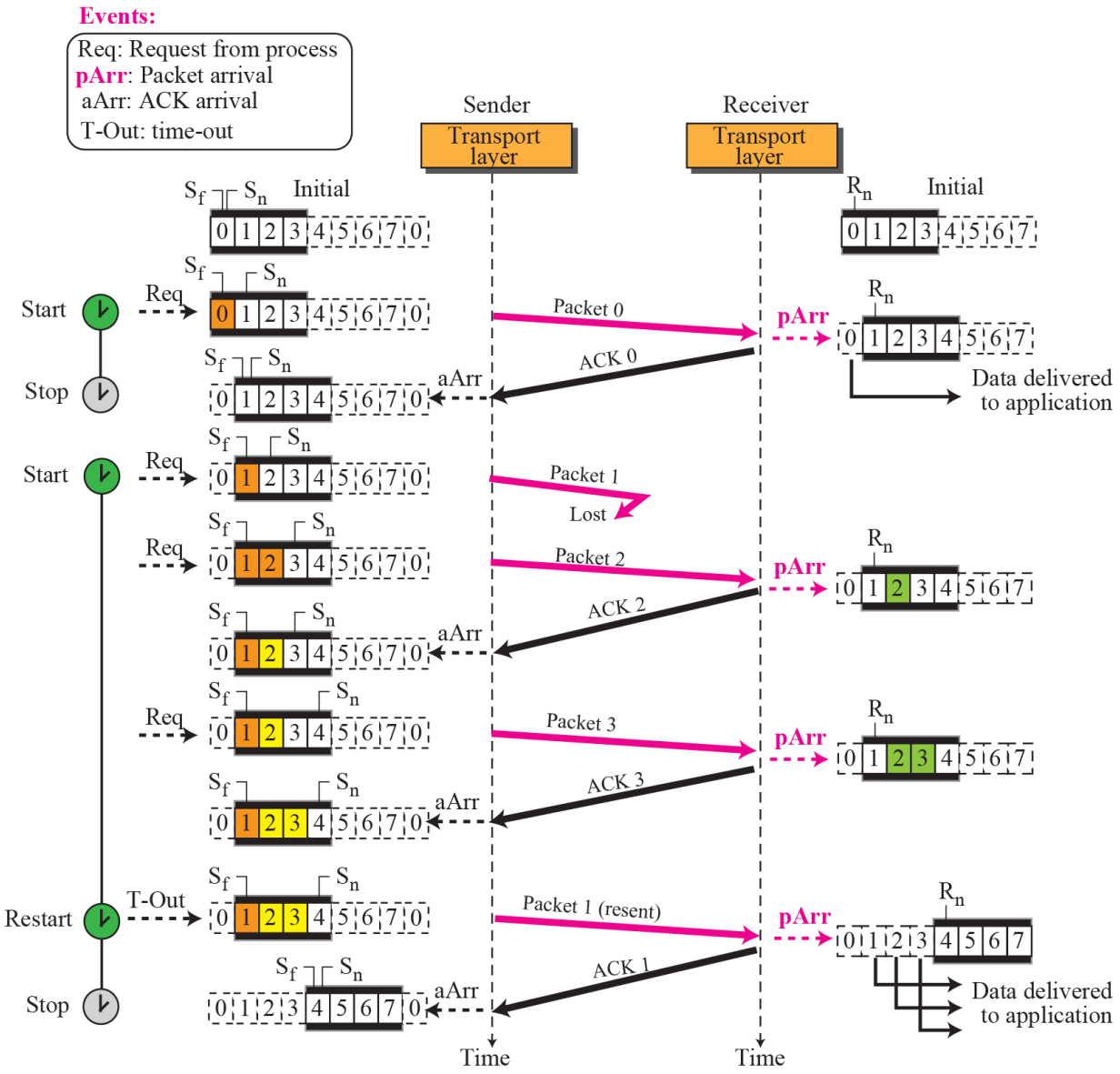
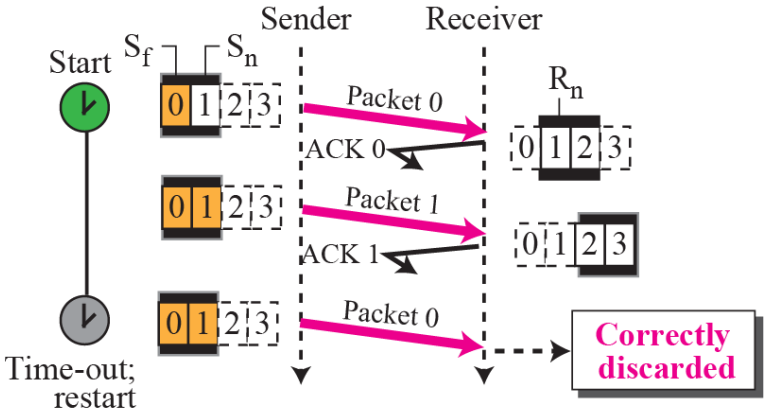
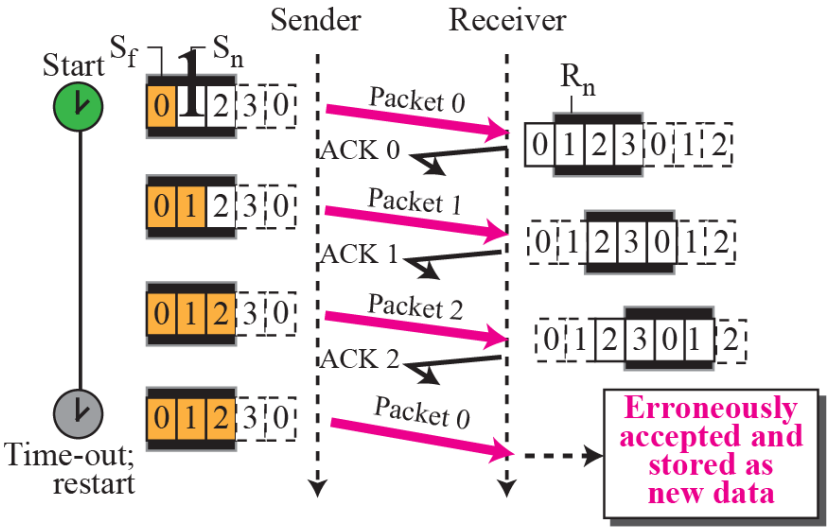


Figure 13.35 *Selective-Repeat window size*



a. Send and receive windows of size = 2^{m-1}

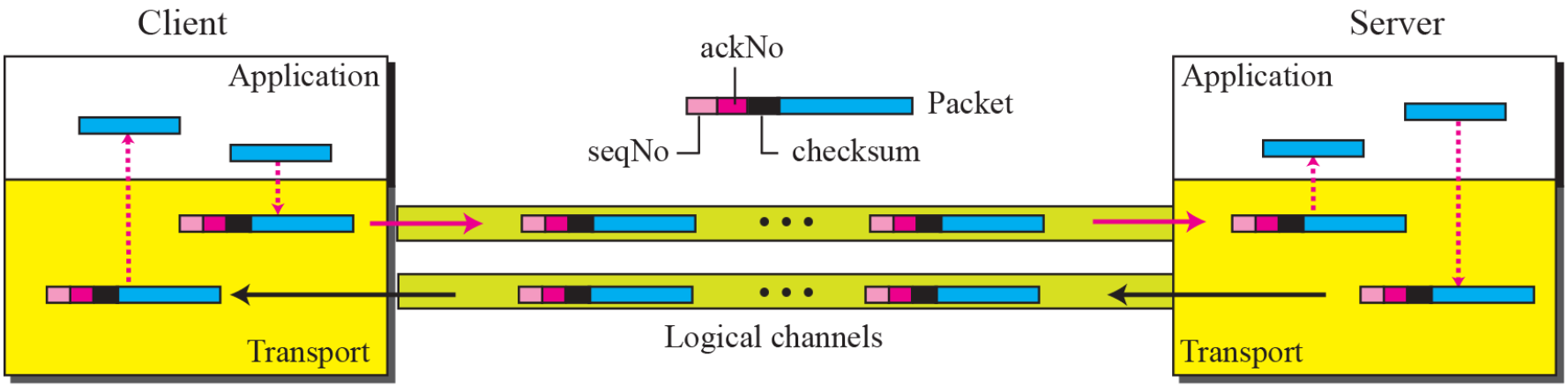


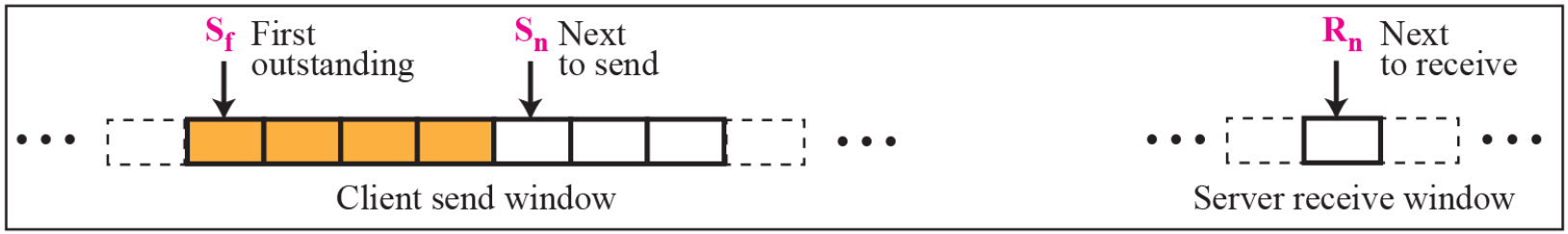
b. Send and receive windows of size > 2^{m-1}

Note

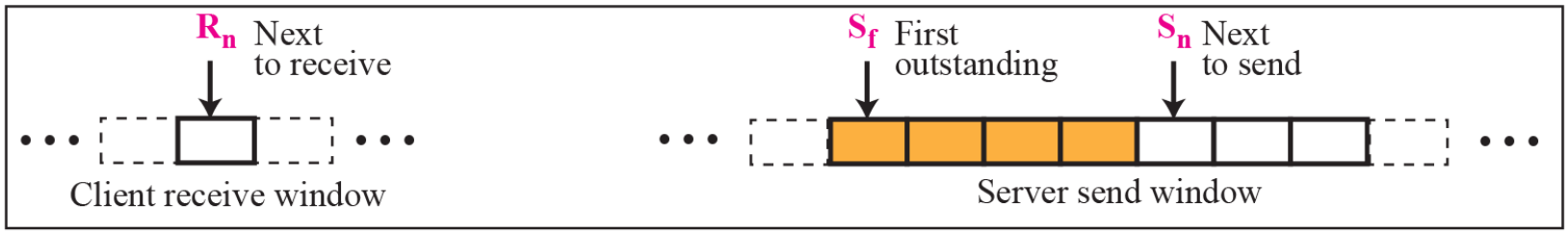
In Selective-Repeat, the size of the sender and receiver window can be at most one-half of 2m.

Figure 13.36 *Design of piggybacking for Go-Back-N*





Windows for communication from client to server



Windows for communication from server to client

Transmission Control Protocol (TCP)

OBJECTIVES:

- To introduce TCP as a protocol that provides reliable stream delivery service.
- To define TCP features and compare them with UDP features.
- To define the format of a TCP segment and its fields.
- To show how TCP provides a connection-oriented service, and show the segments exchanged during connection establishment and connection termination phases.
- To discuss the state transition diagram for TCP and discuss some scenarios.
- To introduce windows in TCP that are used for flow and error control.

OBJECTIVES (*continued*):

- To discuss how TCP implements flow control in which the receive window controls the size of the send window.
- To discuss error control and FSMs used by TCP during the data transmission phase.
- To discuss how TCP controls the congestion in the network using different strategies.
- To list and explain the purpose of each timer in TCP.
- To discuss options in TCP and show how TCP can provide selective acknowledgment using the SACK option.
- To give a layout and a simplified pseudocode for the TCP package.

Chapter Outline

15-1 TCP SERVICES

Figure 15.1 shows the relationship of TCP to the other protocols in the TCP/IP protocol suite. TCP lies between the application layer and the network layer, and serves as the intermediary between the application programs and the network operations.

Topics Discussed in the Section

- ✓ **Process-to-Process Communication**
- ✓ **Stream Delivery Service**
- ✓ **Full-Duplex Communication**
- ✓ **Multiplexing and Demultiplexing**
- ✓ **Connection-Oriented Service**
- ✓ **Reliable Service**

Figure 15.1 *TCP/IP protocol suite*

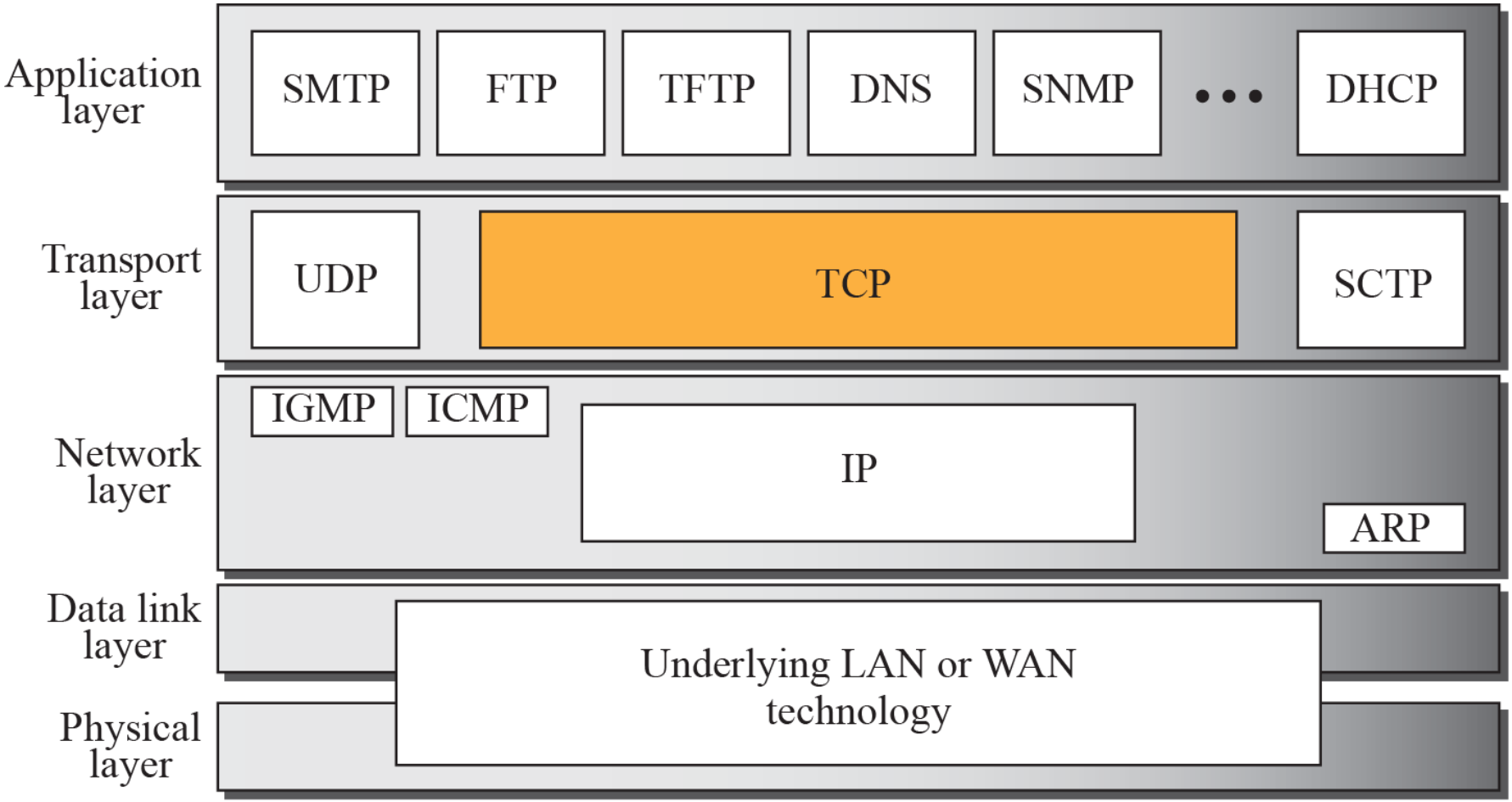


Table 15.1 *Well-known Ports used by TCP*

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 and 21	FTP	File Transfer Protocol (Data and Control)
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

Figure 15.2 *Stream delivery*

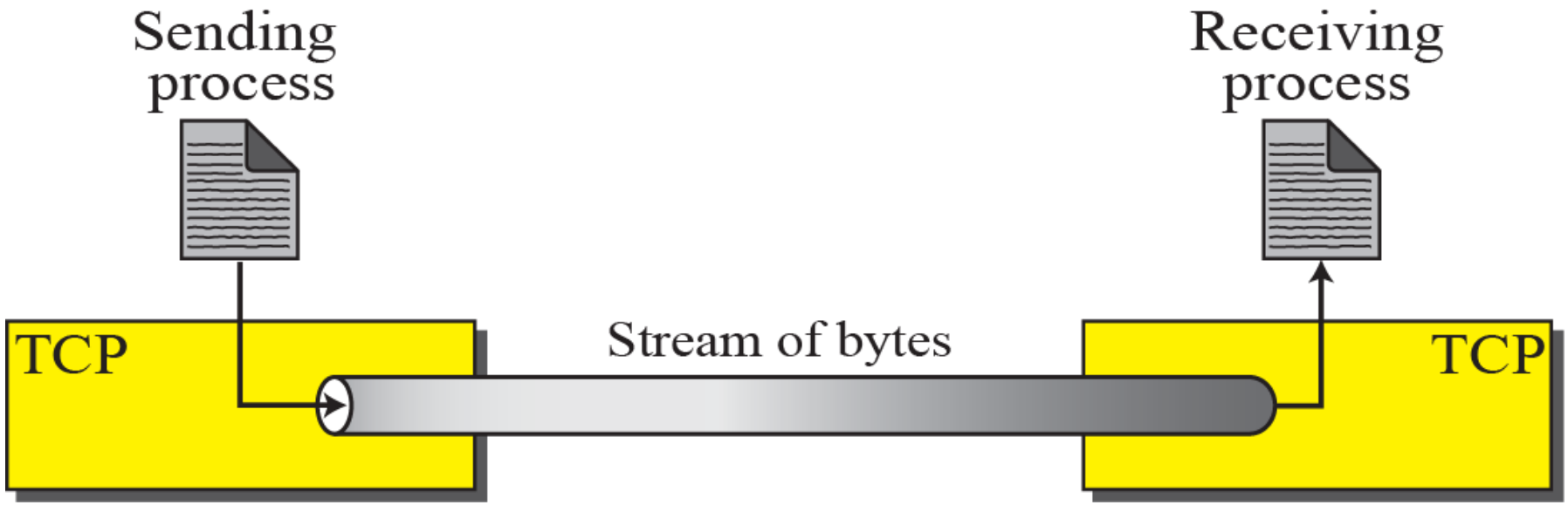


Figure 15.3 *Sending and receiving buffers*

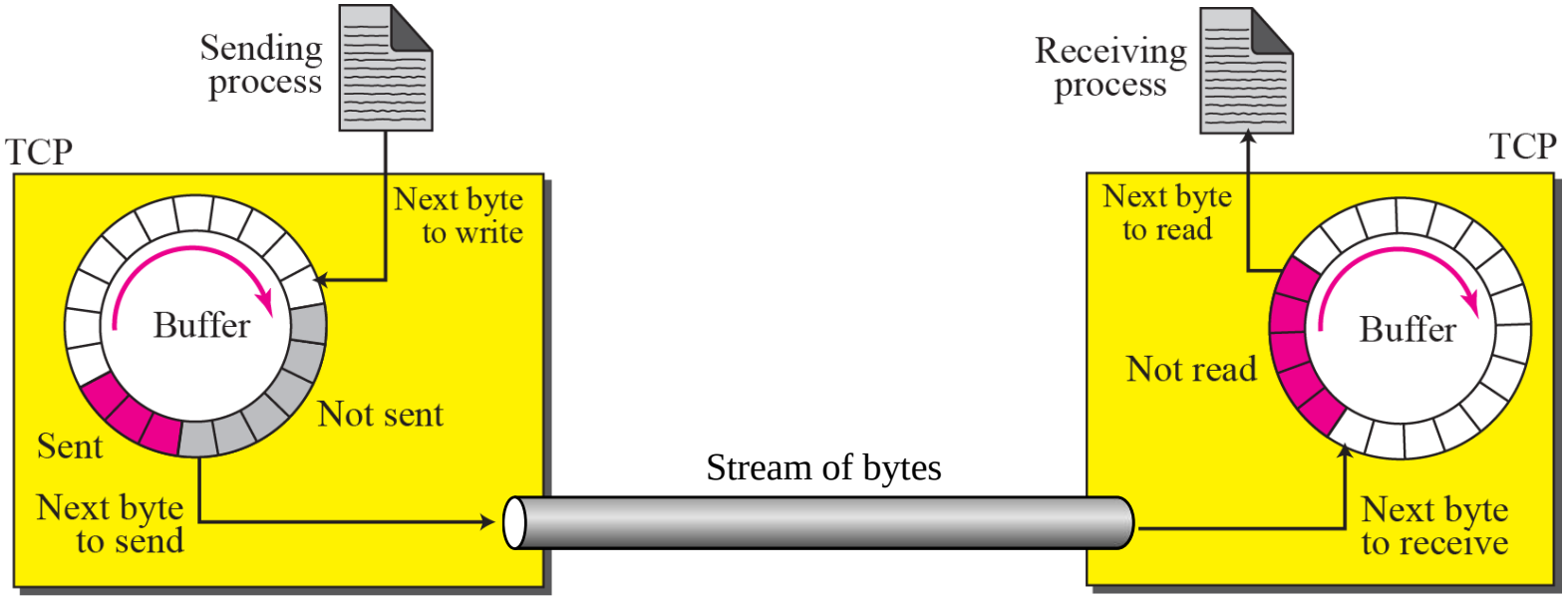
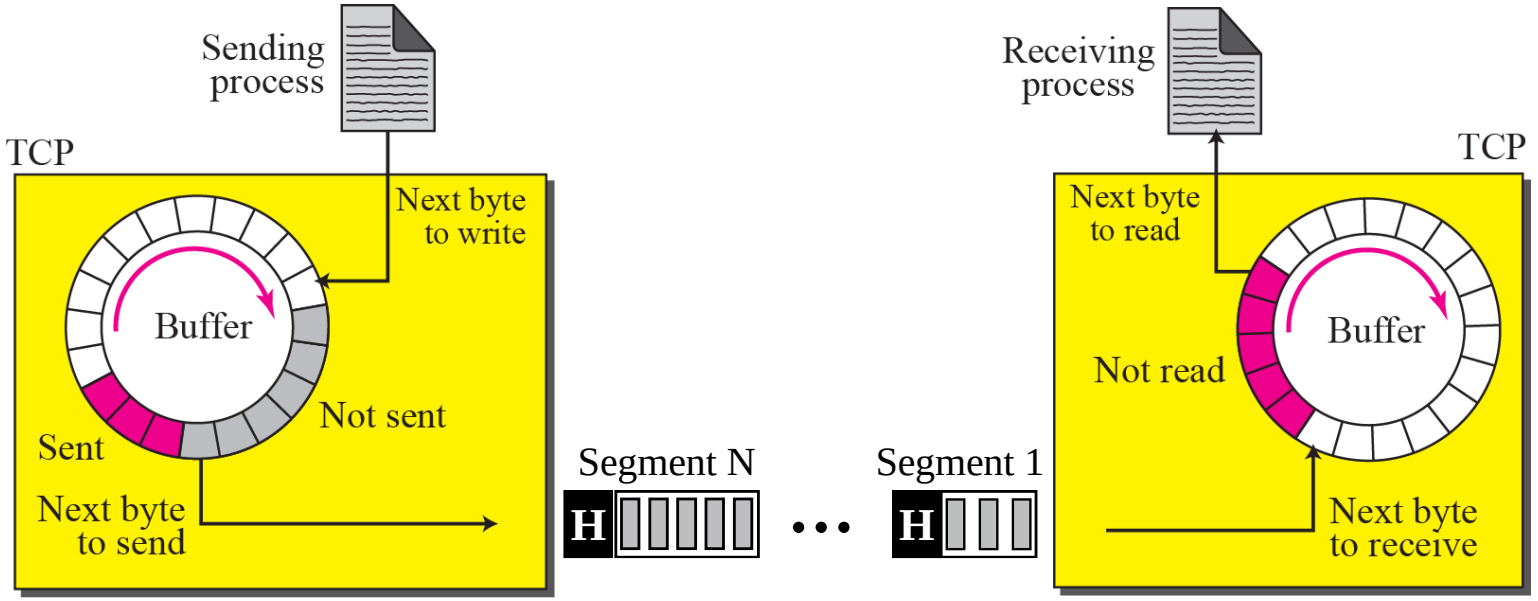


Figure 15.4 *TCP segments*



15-2 TCP FEATURES

To provide the services mentioned in the previous section, TCP has several features that are briefly summarized in this section and discussed later in detail.

Topics Discussed in the Section

- ✓ **Numbering System**
- ✓ **Flow Control**
- ✓ **Error Control**
- ✓ **Congestion Control**

Note

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

The numbering starts with an arbitrarily generated number.

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.

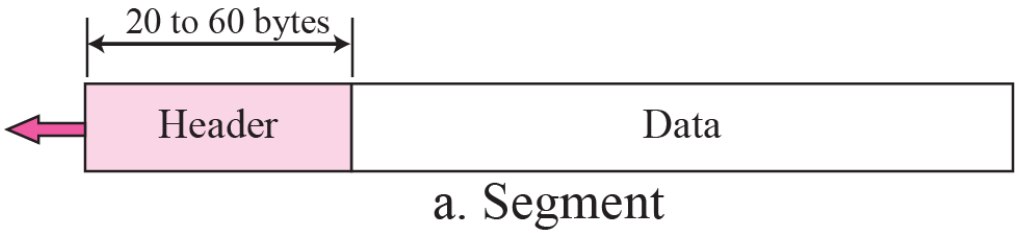
15-3 SEGMENT

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

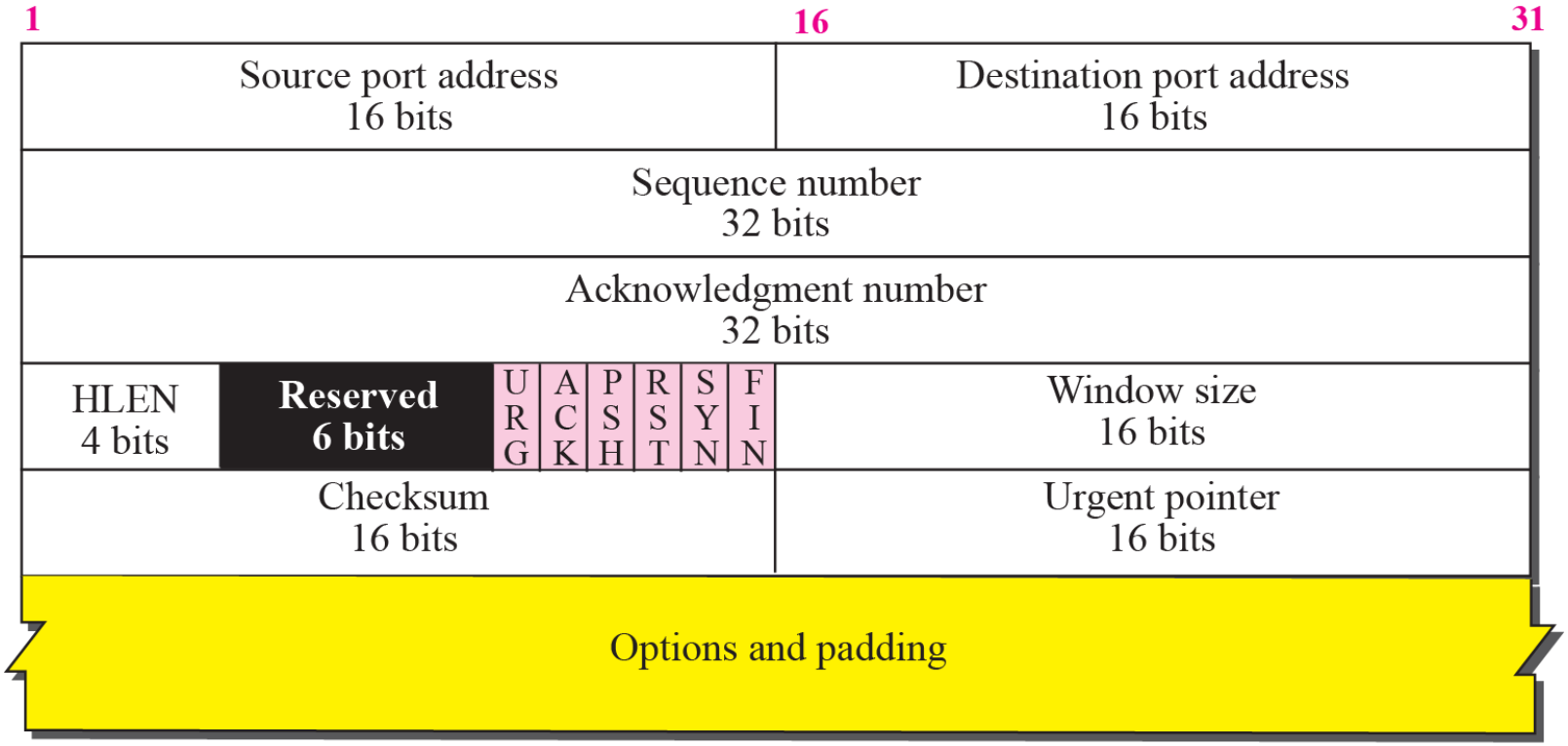
Topics Discussed in the Section

- ✓ **Format**
- ✓ **Encapsulation**

Figure 15.5 *TCP segment format*



a. Segment



b. Header

Figure 15.6 *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

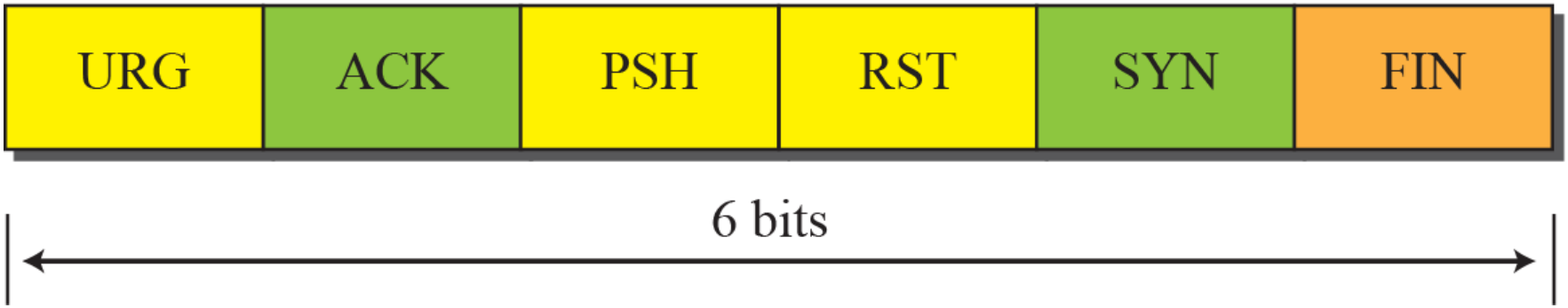
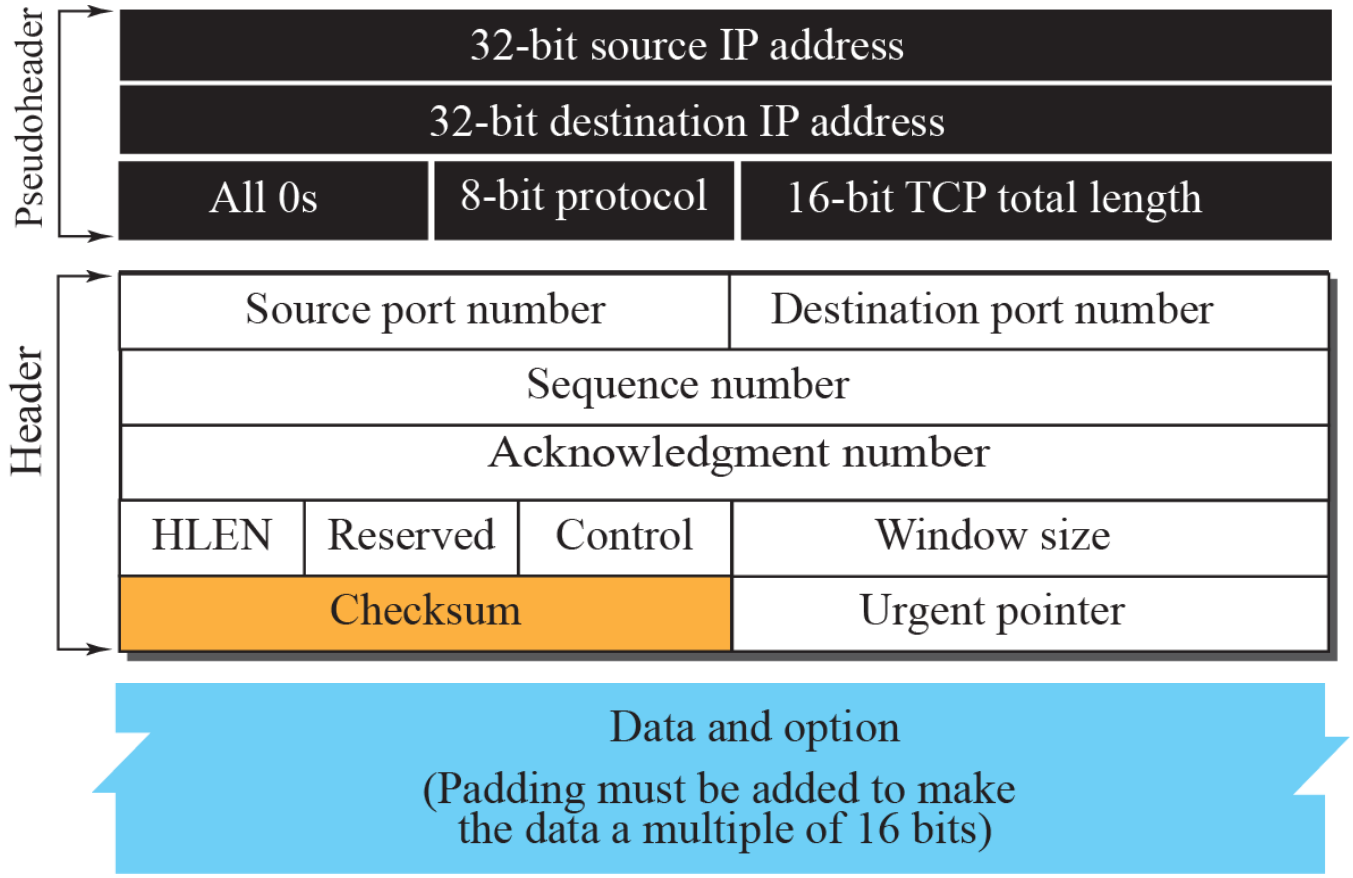


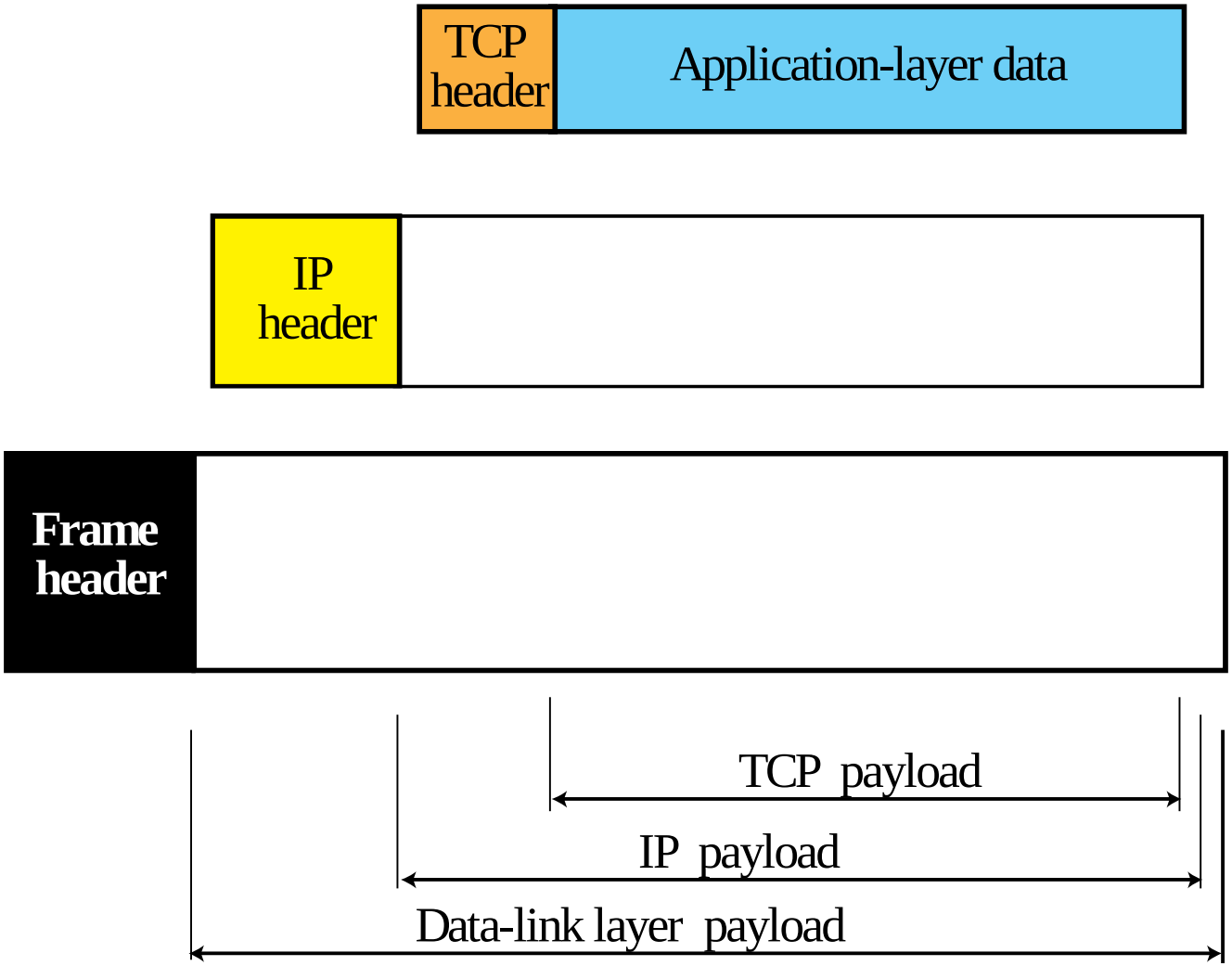
Figure 15.7 *Pseudoheader added to the TCP segment*



Note

The use of the checksum in TCP is mandatory.

Figure 15.8 *Encapsulation*



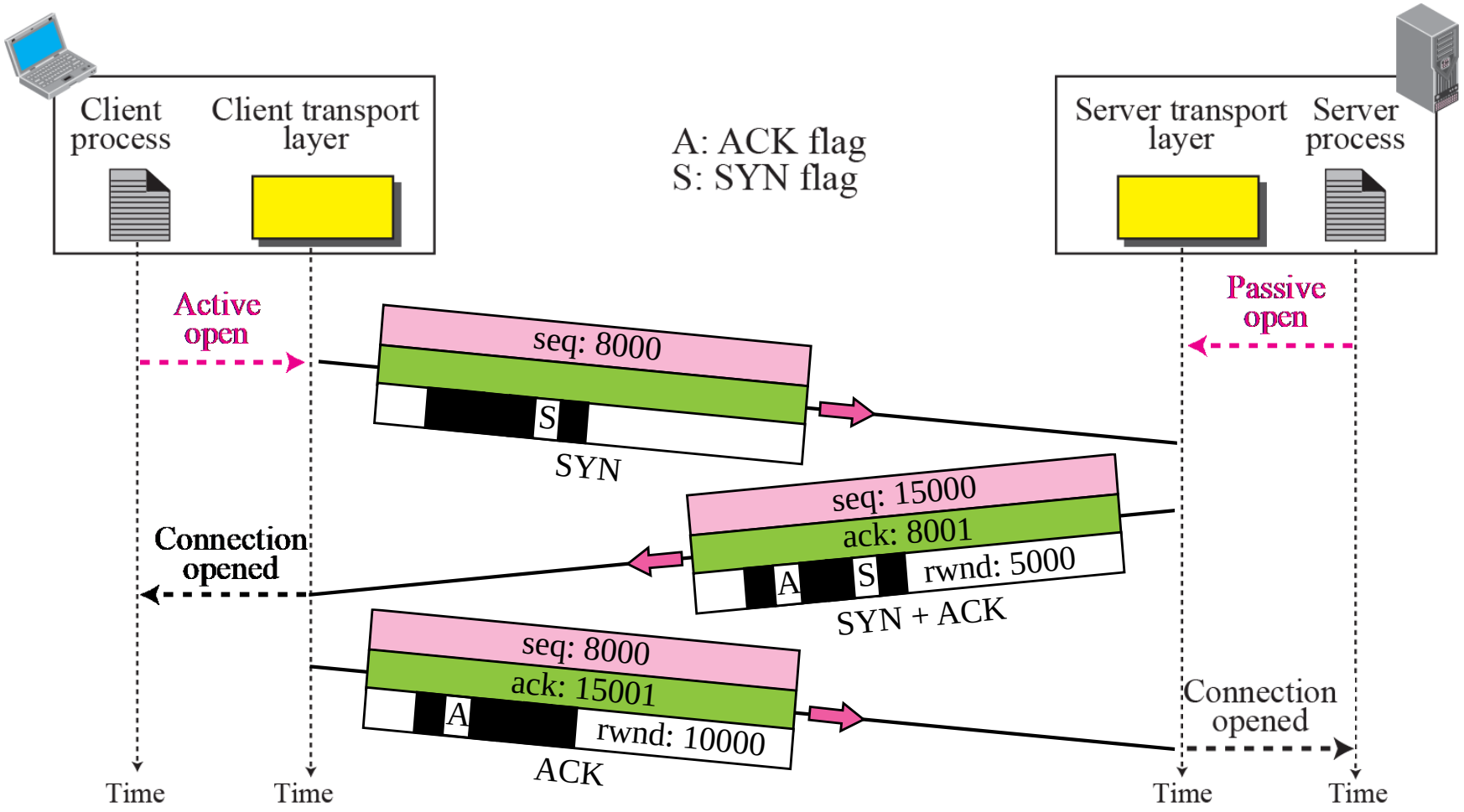
15-4 A TCP CONNECTION

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.

Topics Discussed in the Section

- ✓ **Connection Establishment**
- ✓ **Data Transfer**
- ✓ **Connection Termination**
- ✓ **Connection Reset**

Figure 15.9 *Connection establishment using three-way handshake*



Note

A SYN segment cannot carry data, but it consumes one sequence number.

Note

A SYN + ACK segment cannot carry data, but does consume one sequence number.

Note

An ACK segment, if carrying no data, consumes no sequence number.

Figure 15.10 *Data Transfer*

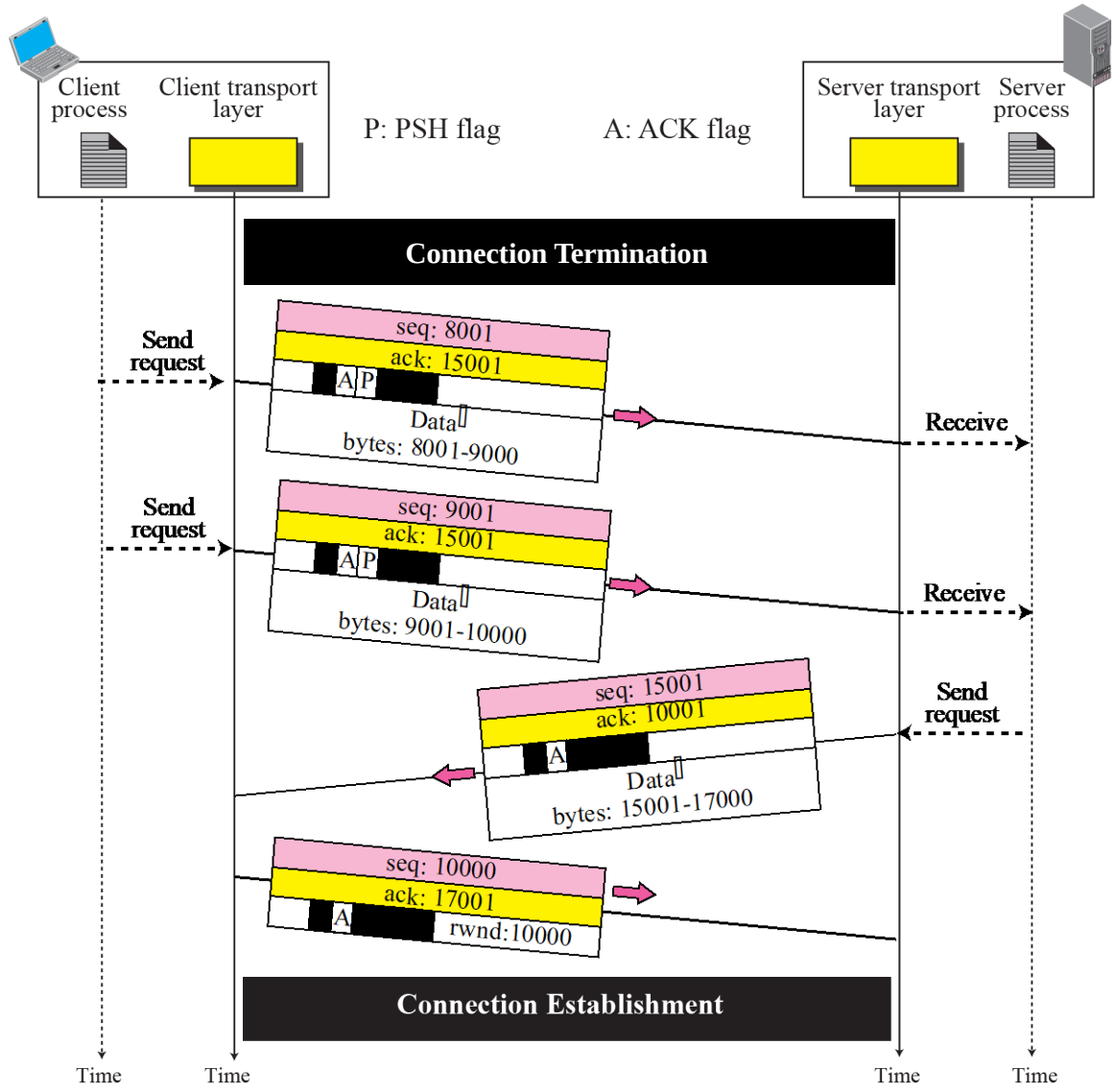
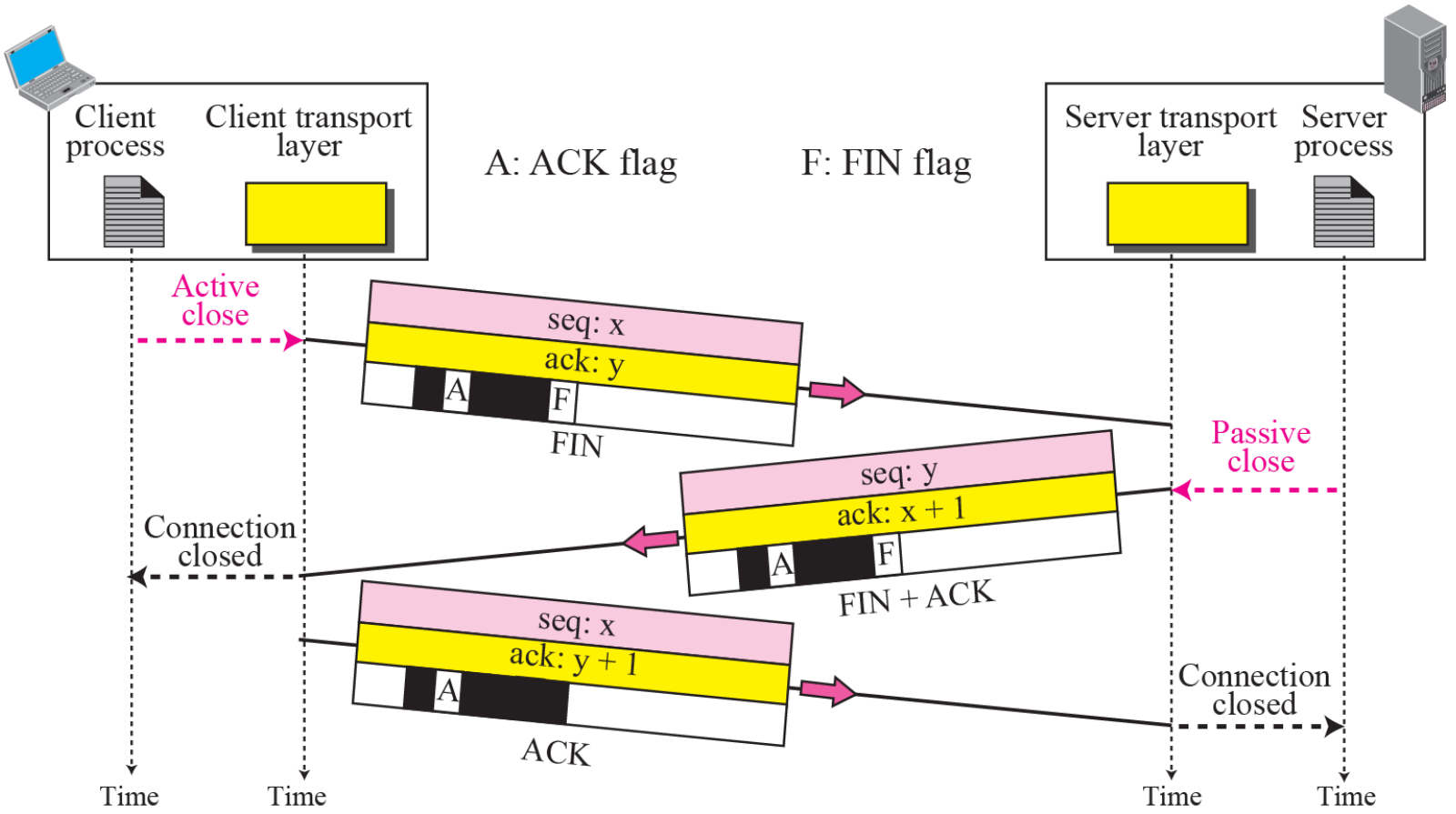


Figure 15.11 *Connection termination using three-way handshake*



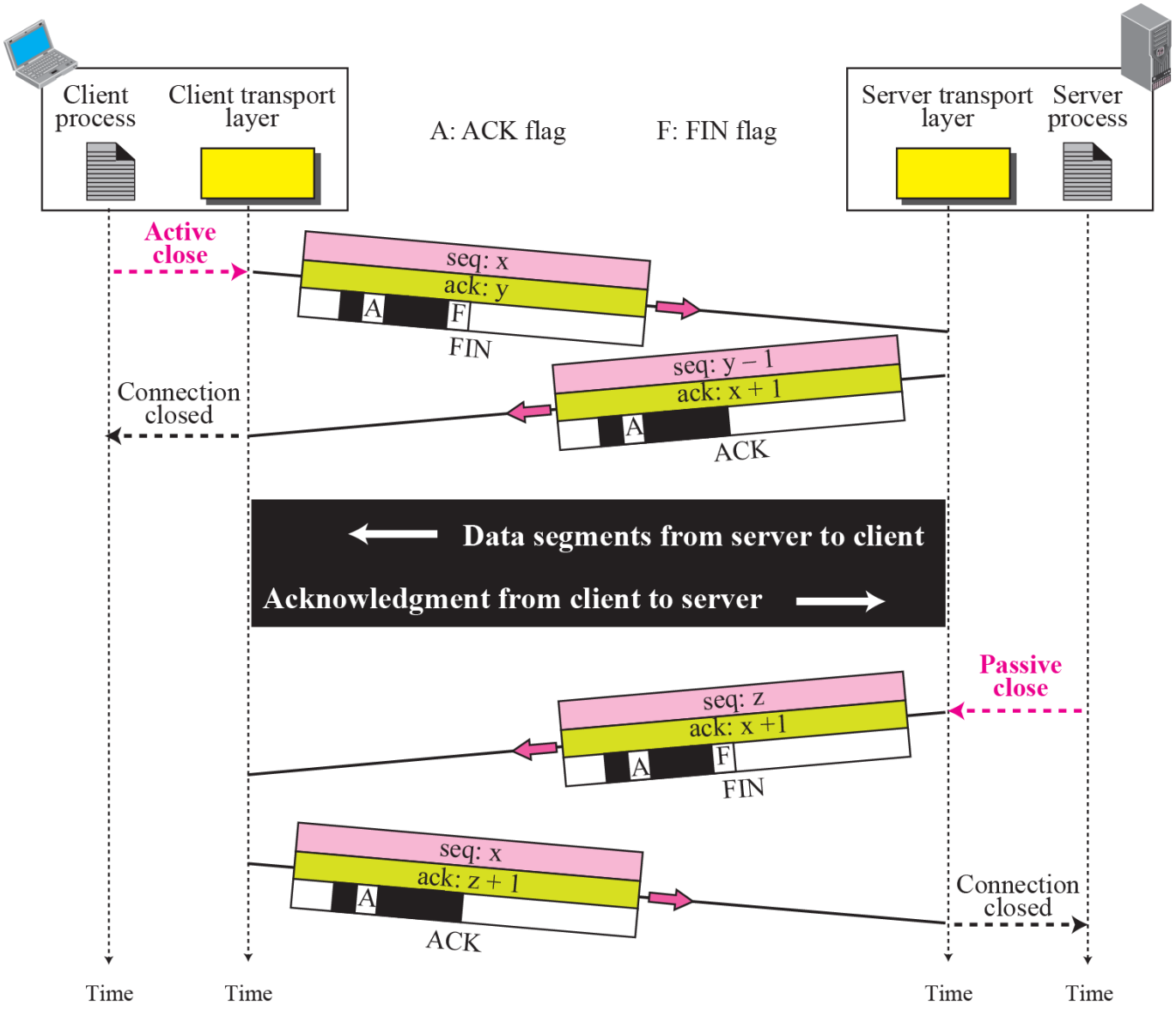
Note

The FIN segment consumes one sequence number if it does not carry data.

Note

The FIN + ACK segment consumes one sequence number if it does not carry data.

Figure 15.12 *Half-Close*



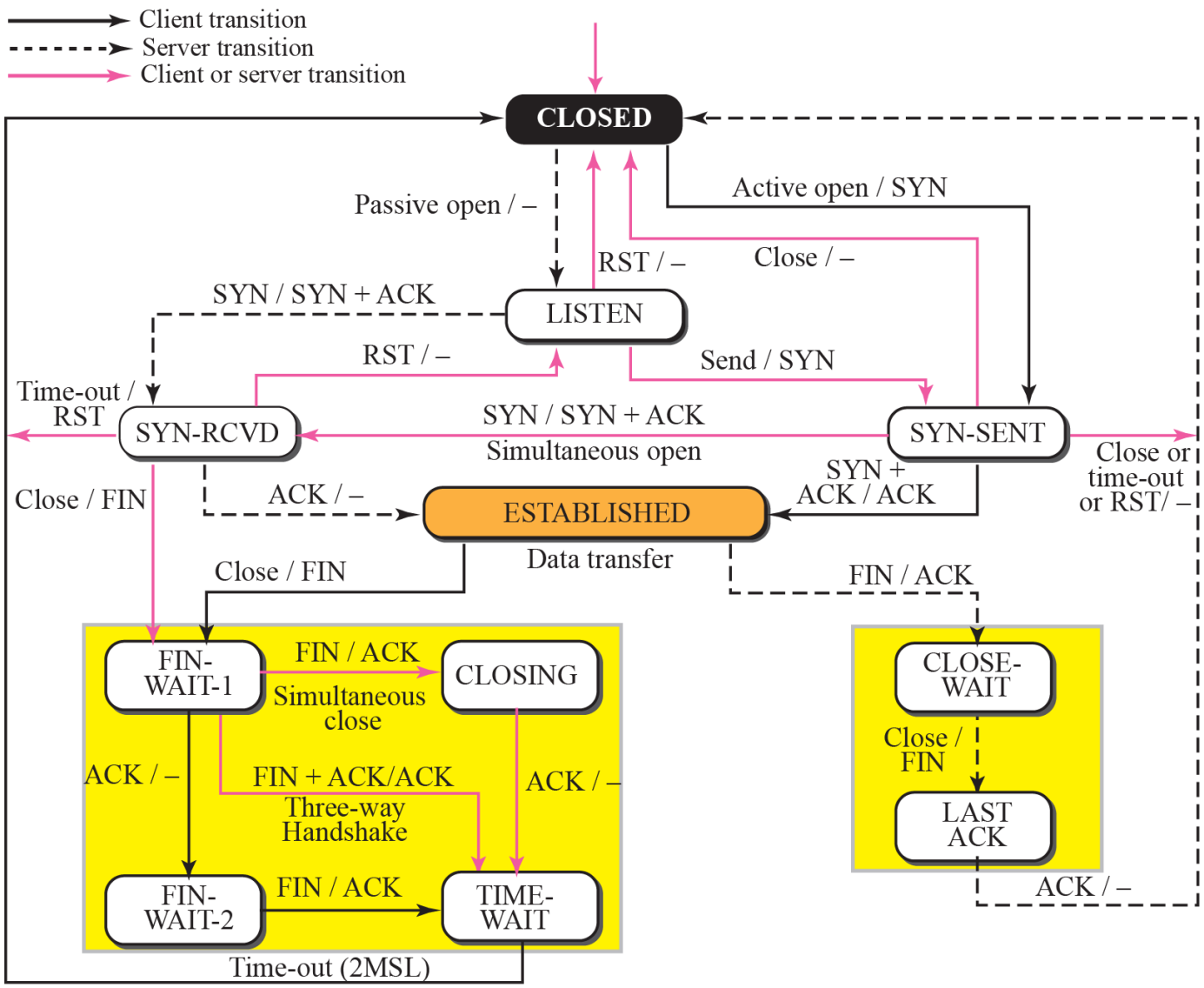
15-5 STATE TRANSITION DIAGRAM

To keep track of all the different events happening during connection establishment, connection termination, and data transfer, TCP is specified as the finite state machine shown in Figure 15.13.

Topics Discussed in the Section

- ✓ **Scenarios**

Figure 15.13 *State transition diagram*



Note

The state marked as ESTBLISHED in the FSM is in fact two different sets of states that the client and server undergo to transfer data.

Table 15.2 *States for TCP*

State	*Description*
CLOSED	No connection exists
LISTEN	Passive open received; waiting for SYN
SYN-SENT	SYN sent; waiting for ACK
SYN-RCVD	SYN+ACK sent; waiting for ACK
ESTABLISHED	Connection established; data transfer in progress
FIN-WAIT-1	First FIN sent; waiting for ACK
FIN-WAIT-2	ACK to first FIN received; waiting for second FIN
CLOSE-WAIT	First FIN received, ACK sent; waiting for application to close
TIME-WAIT	Second FIN received, ACK sent; waiting for 2MSL time-out
LAST-ACK	Second FIN sent; waiting for ACK
CLOSING	Both sides decided to close simultaneously

Figure 15.14 *Transition diagram for connection and half-close termination*

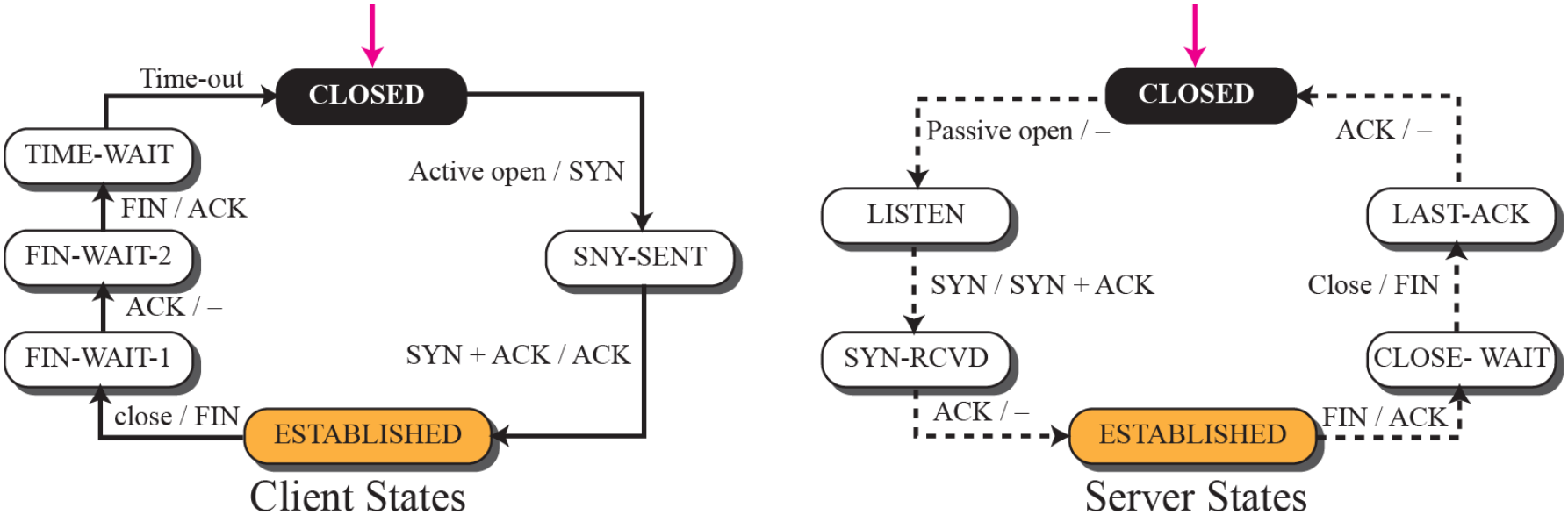


Figure 15.15 *Time-line diagram for Figure 15.14*

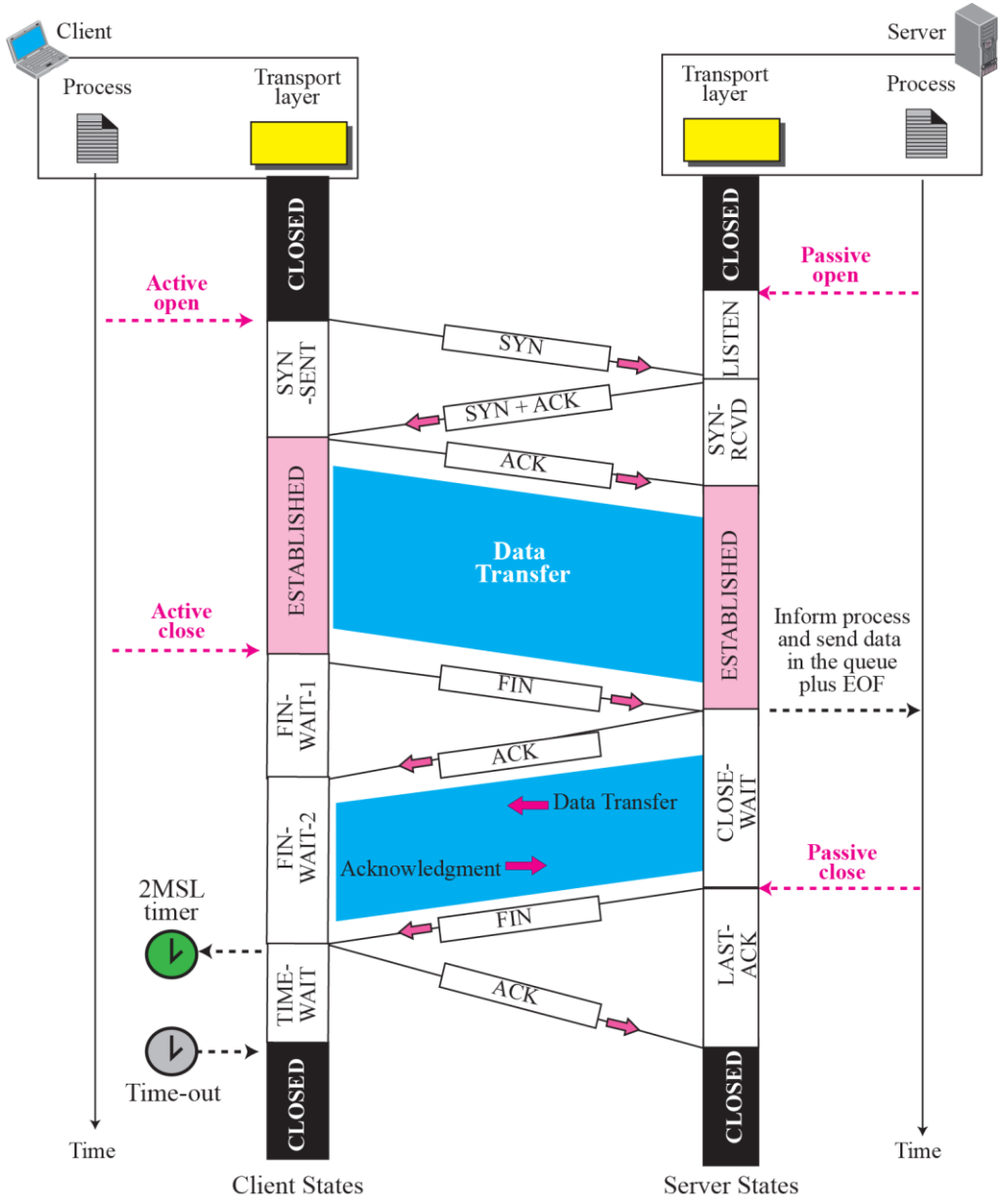


Figure 15.16 *Transition diagram for a common scenario*

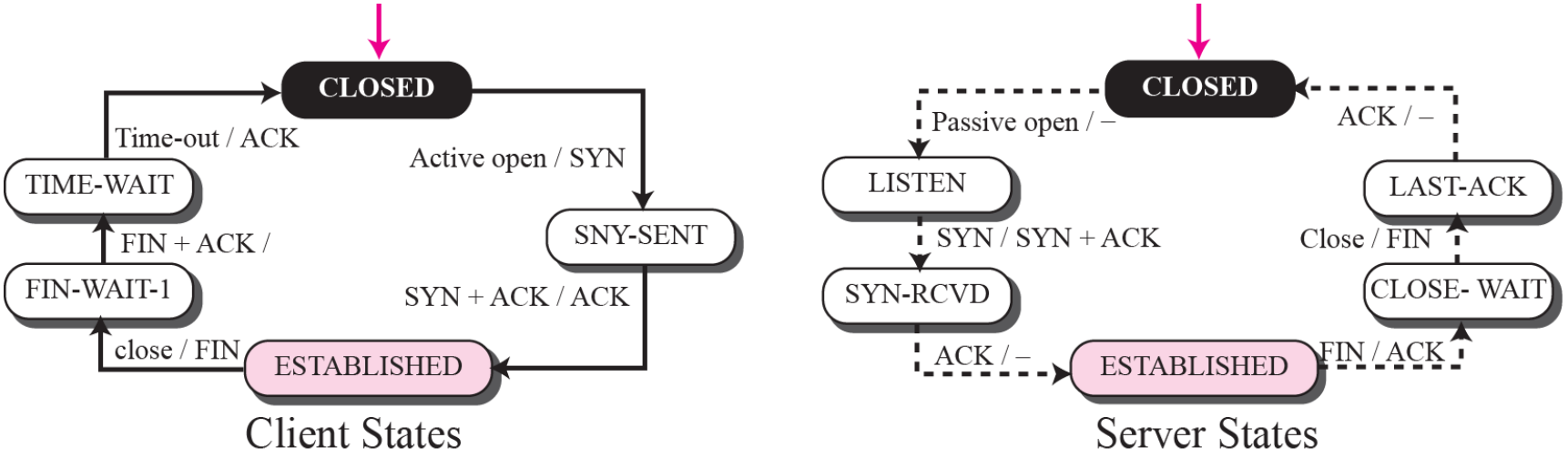


Figure 15.17 *Time line for a common scenario*

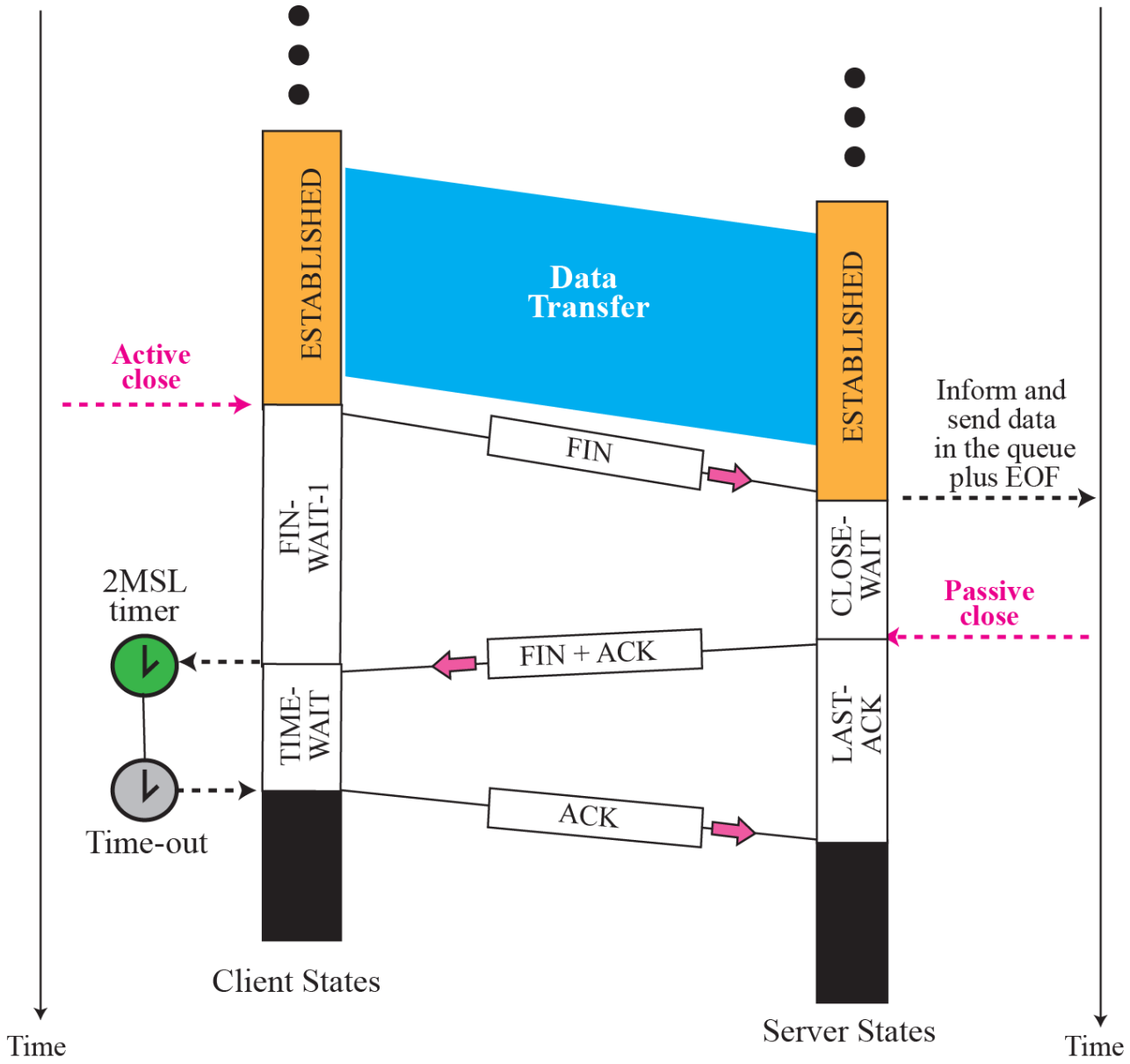


Figure 15.18 *Simultaneous open*

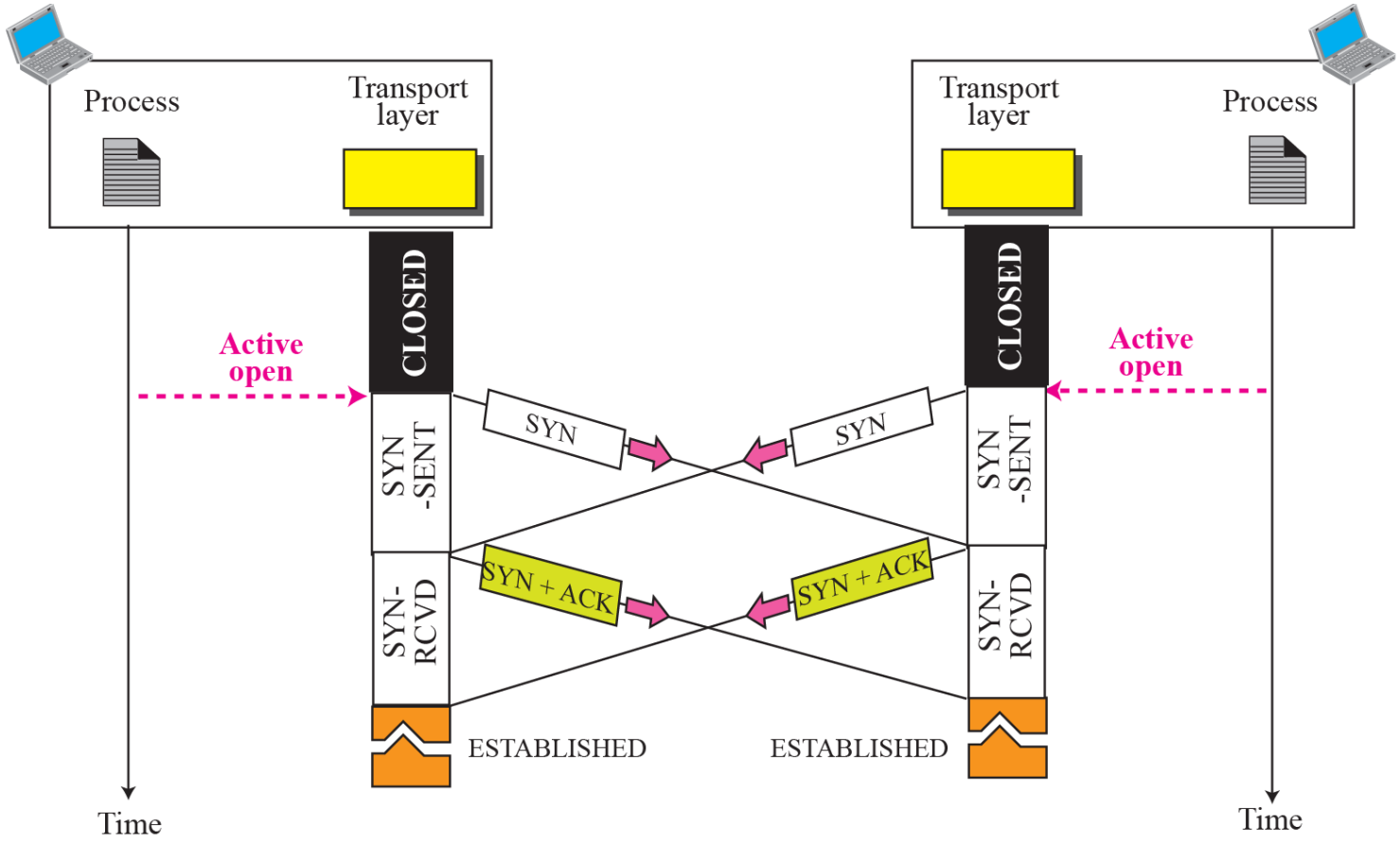


Figure 15.19 *Simultaneous close*

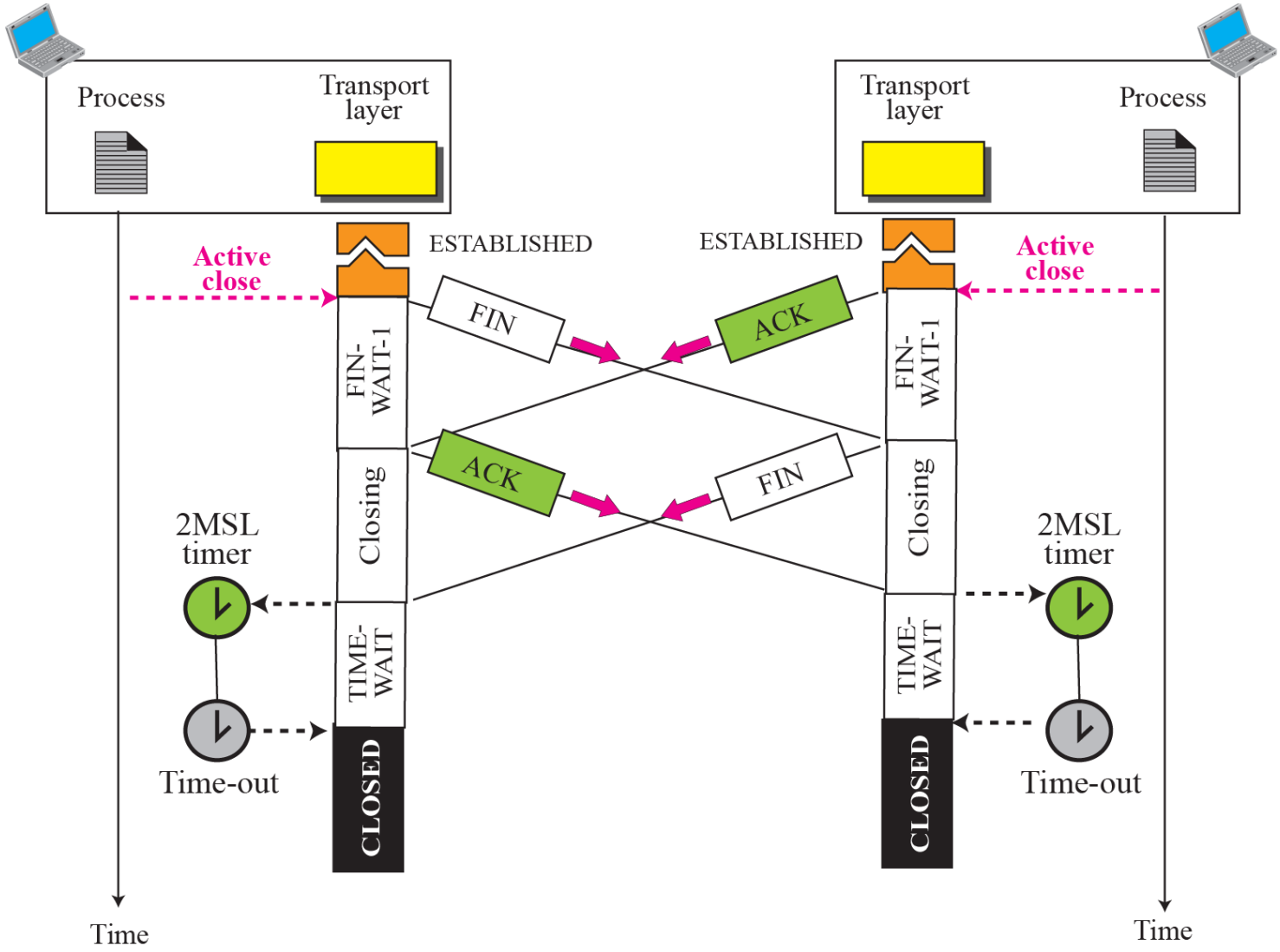


Figure 15.20 *Denying a connection*

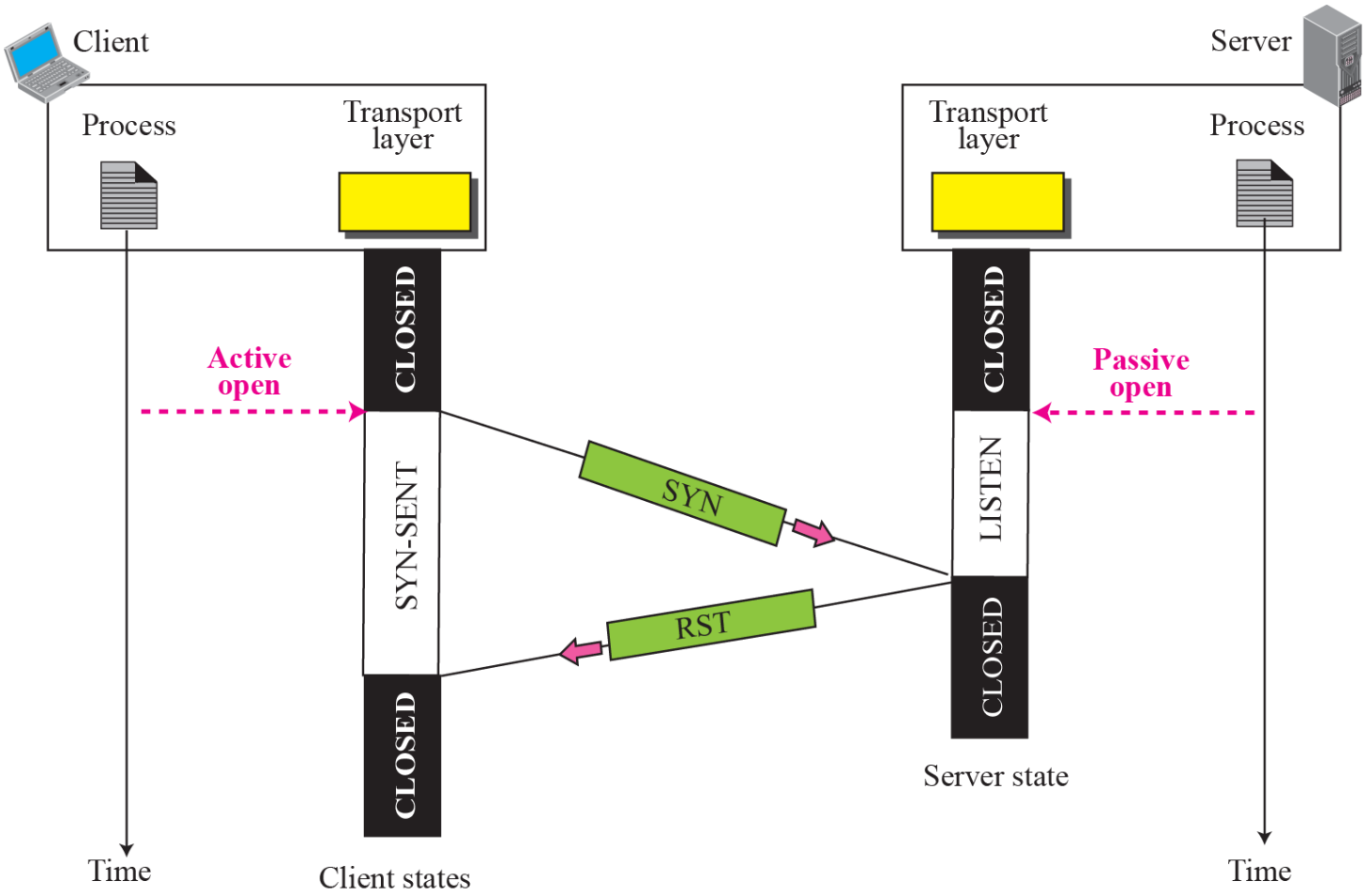
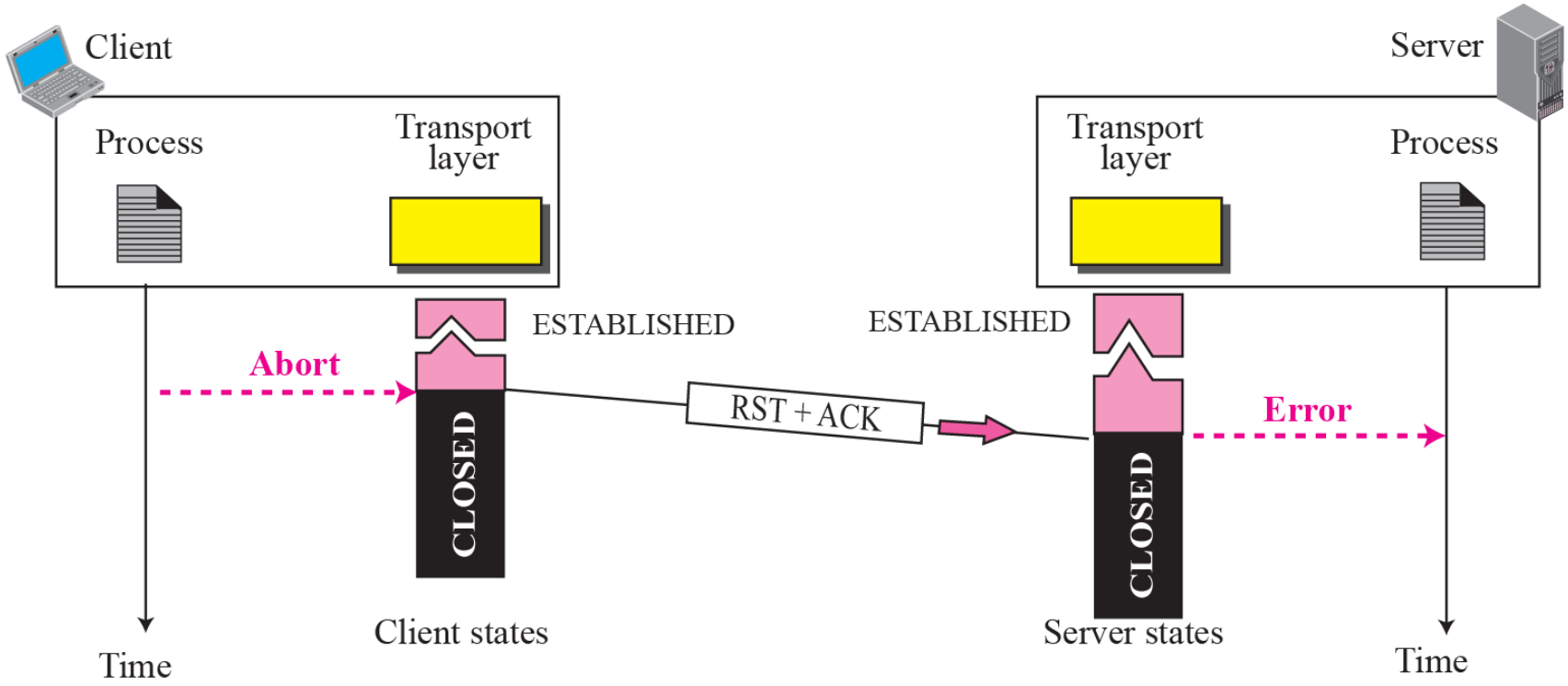


Figure 15.21 *Aborting a connection*



15-6 WINDOWS IN TCP

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.

Topics Discussed in the Section

- ✓ **Send Window**
- ✓ **Receive Window**

Figure 15.22 *Send window in TCP*

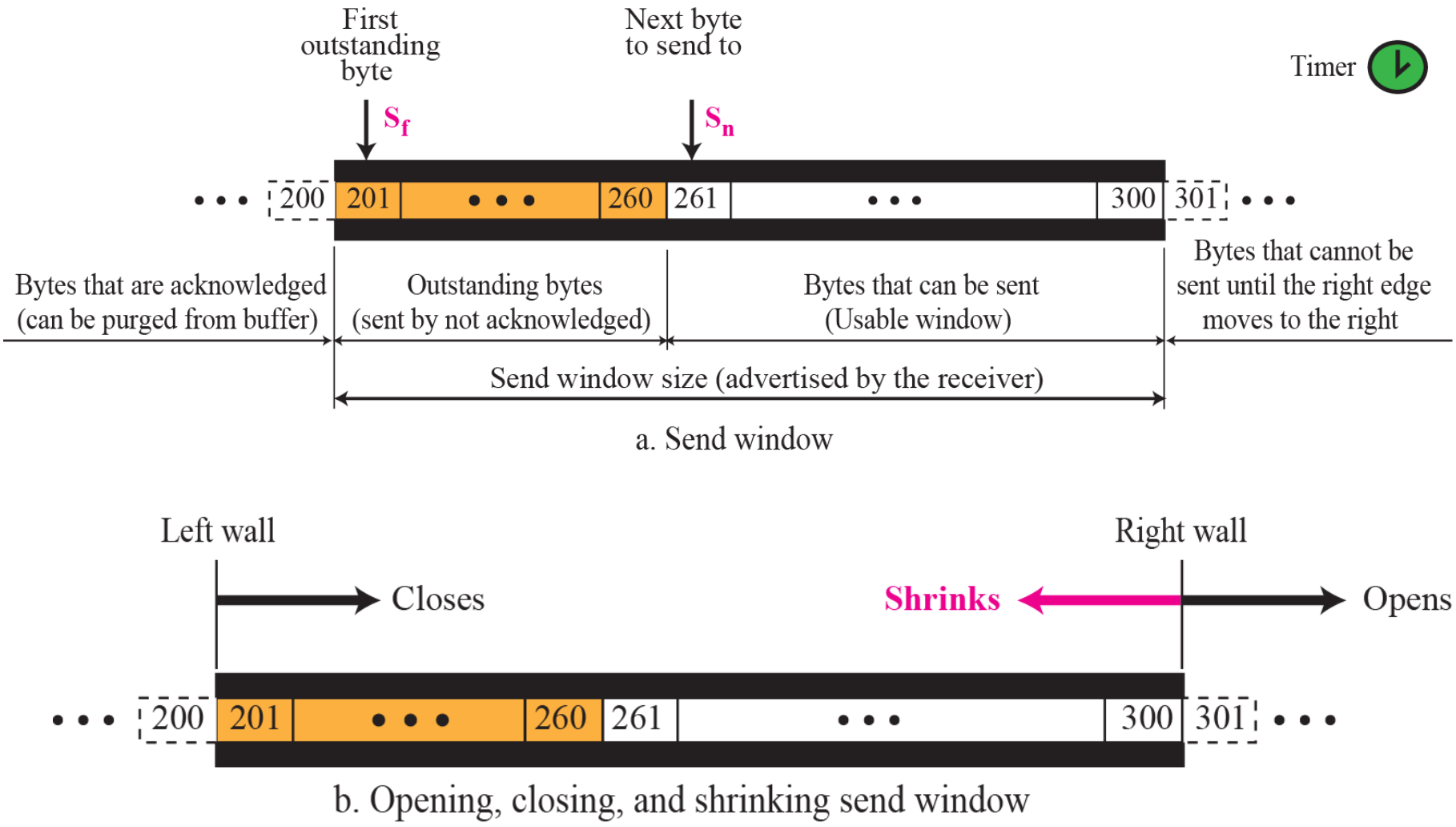
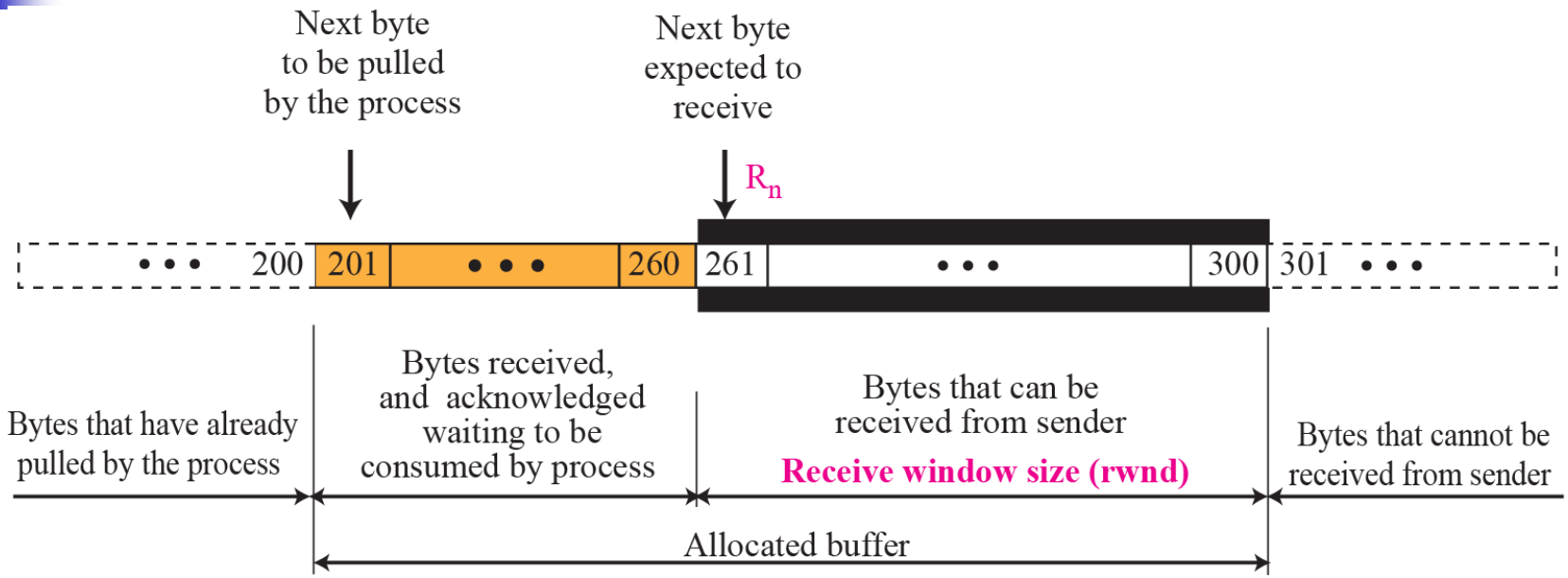
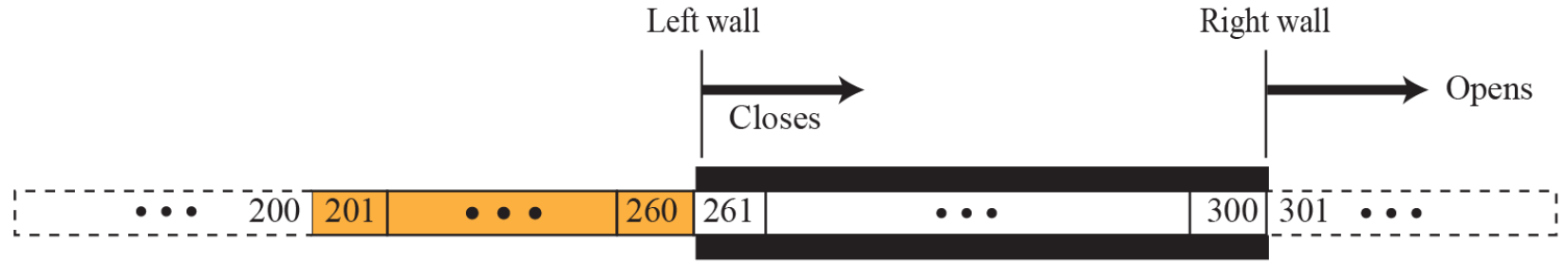


Figure 15.23 *Receive window in TCP*



a. Receive window and allocated buffer



b. Opening and closing of receive window

15-7 FLOW CONTROL

As discussed in Chapter 13, flow control balances the rate a producer creates data with the rate a consumer can use the data. TCP separates flow control from error control. In this section we discuss flow control, ignoring error control. We temporarily assume that the logical channel between the sending and receiving TCP is error-free. Figure 15.24 shows unidirectional data transfer between a sender and a receiver; bidirectional data transfer can be deduced from unidirectional one as discussed in Chapter 13.

Topics Discussed in the Section

- ✓ **Opening and Closing Windows**
- ✓ **Shrinking of Windows**
- ✓ **Silly Window Syndrome**

Figure 15.24 *TCP/IP protocol suite*

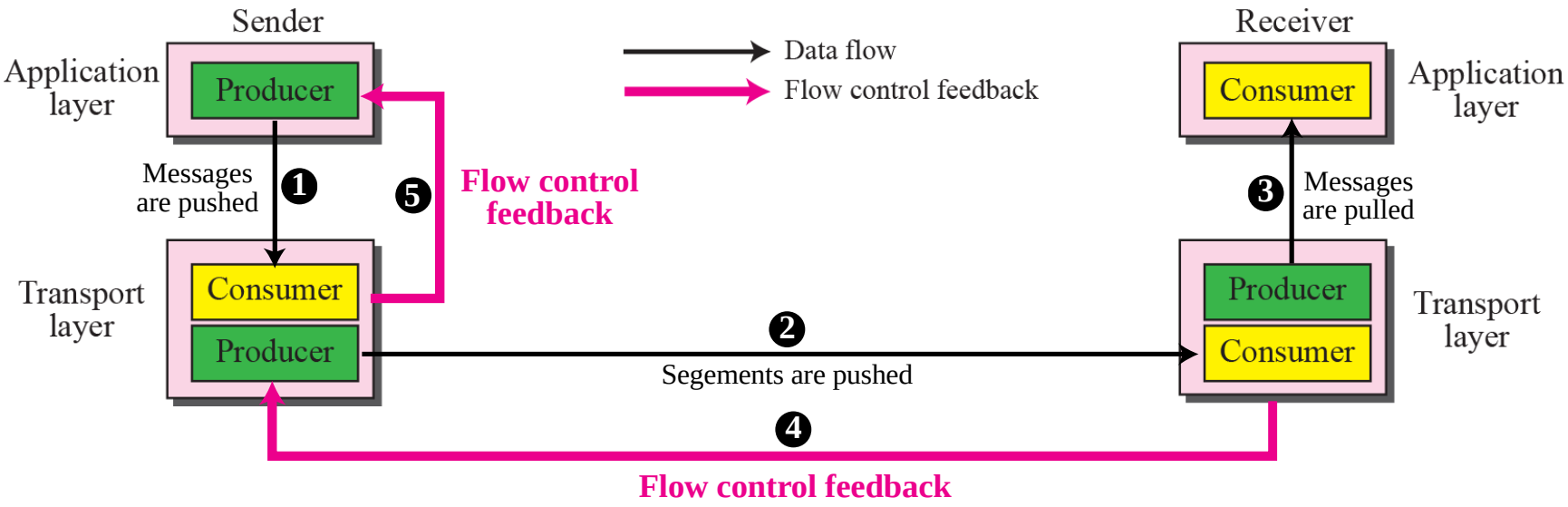
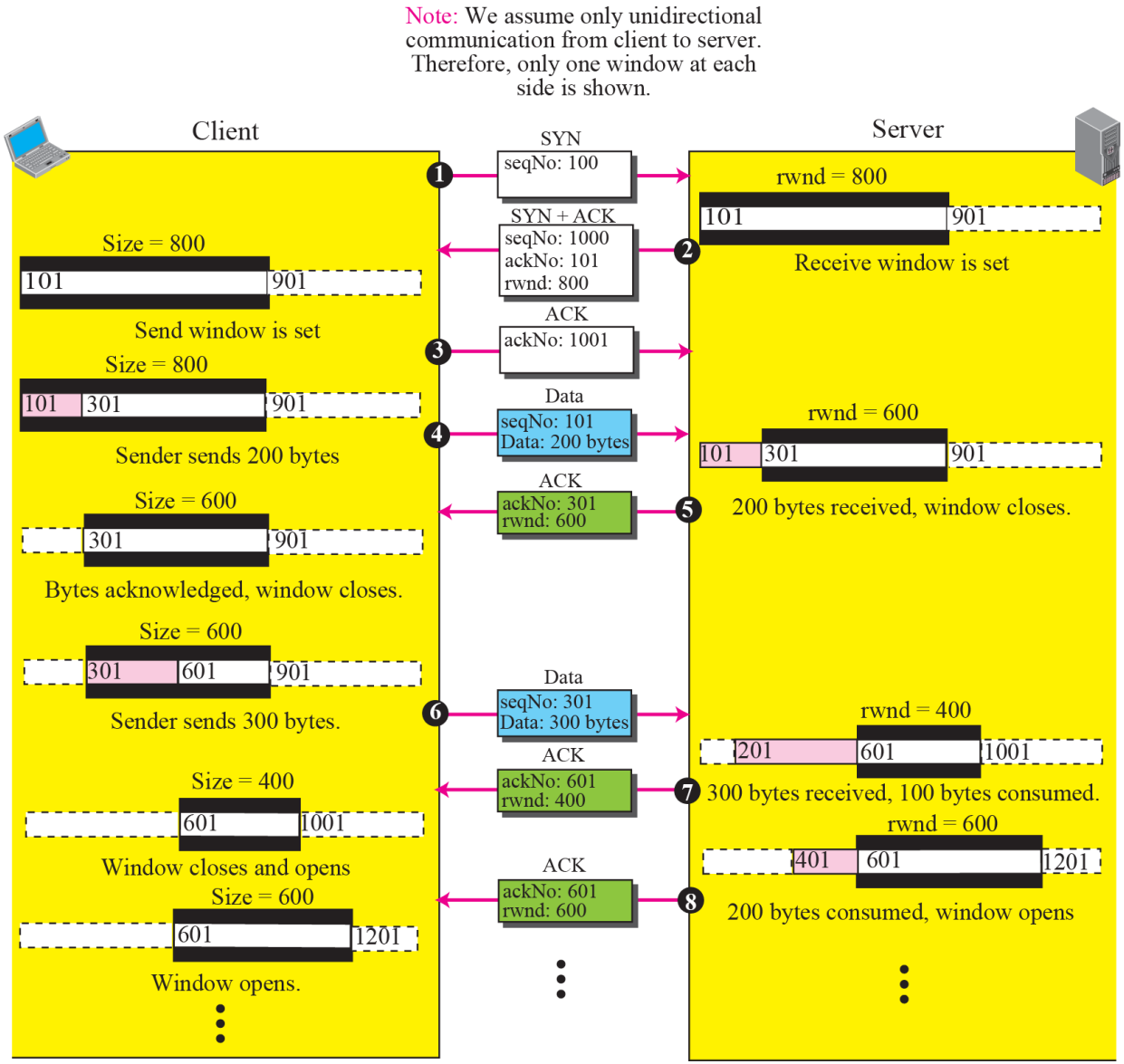


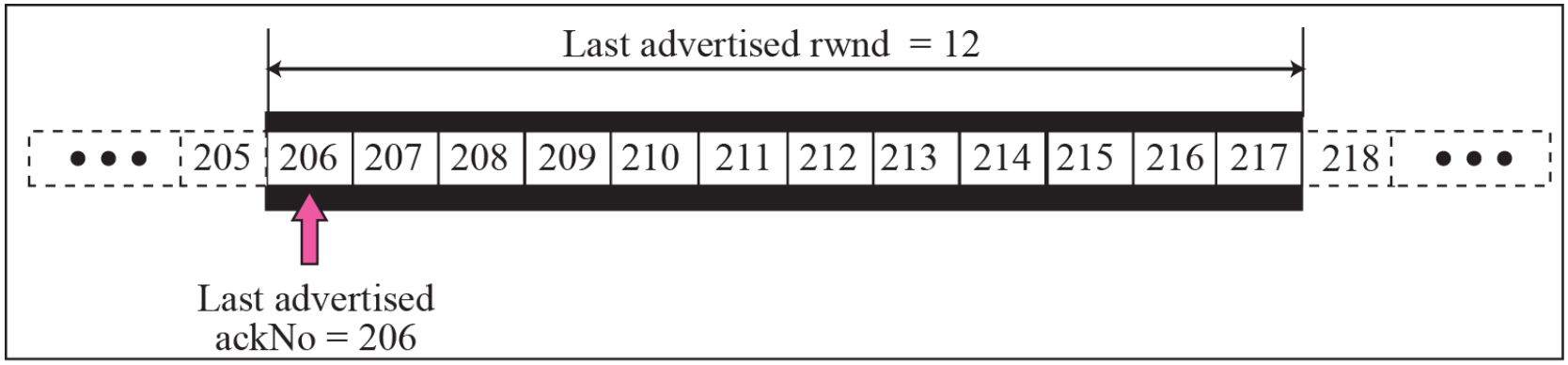
Figure 15.25 *An example of flow control*



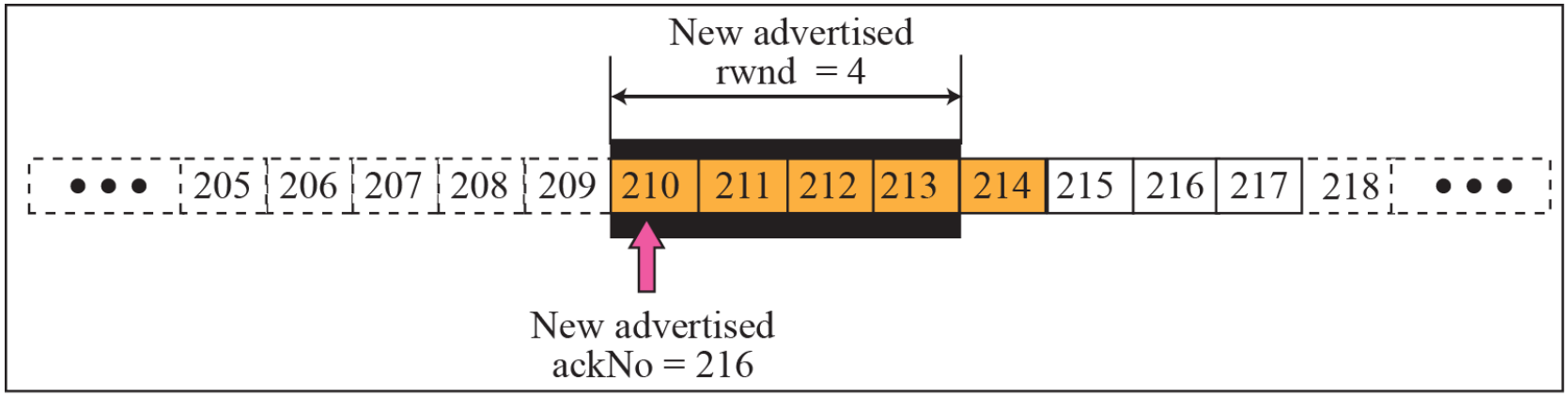
Example 15.2

Figure 15.26 shows the reason for the mandate in window shrinking. Part a of the figure shows values of last acknowledgment and *rwnd*. Part b shows the situation in which the sender has sent bytes 206 to 214. Bytes 206 to 209 are acknowledged and purged. The new advertisement, however, defines the new value of *rwnd* as 4, in which 210 + 4 < 206 + 12. When the send window shrinks, it creates a problem: byte 214 which has been already sent is outside the window. The relation discussed before forces the receiver to maintain the right-hand wall of the window to be as shown in part a because the receiver does not know which of the bytes 210 to 217 has already been sent. One way to prevent this situation is to let the receiver postpone its feedback until enough buffer locations are available in its window. In other words, the receiver should wait until more bytes are consumed by its process.

Figure 15.26 *Example 15.2*



a. The window after the last advertisement



b. The window after the new advertisement; window has shrunk

15-8 ERROR CONTROL

TCP is a reliable transport layer protocol. This means that an application program that delivers a stream of data to TCP relies on TCP to deliver the entire stream to the application program on the other end in order, without error, and without any part lost or duplicated.

Error control in TCP is achieved through the use of three tools: *checksum*, *acknowledgment*, and *time-out*.

Topics Discussed in the Section

- ✓ **Checksum**
- ✓ **Acknowledgment**
- ✓ **Retransmission**
- ✓ **Out-of-Order Segments**
- ✓ **FSMs for Data Transfer in TCP**
- ✓ **Some Scenarios**

Note

ACK segments do not consume sequence numbers and are not acknowledged.

Note

Data may arrive out of order and be temporarily stored by the receiving TCP, but TCP guarantees that no out-of-order data are delivered to the process.

Note

TCP can be best modeled as a Selective Repeat protocol.

Figure 15.27 *Simplified FSM for sender site*

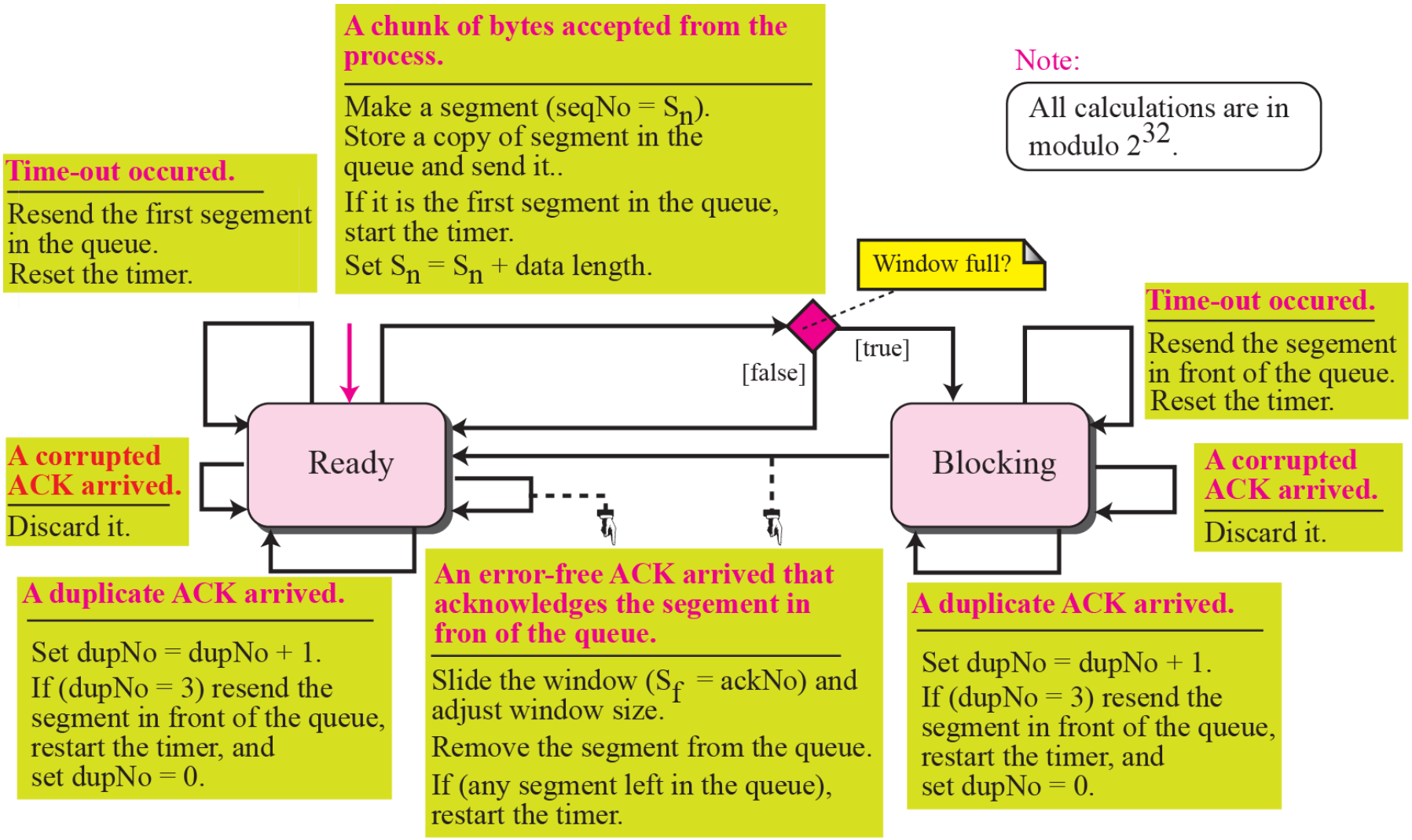


Figure 15.28 *Simplified FSM for the receiver site*

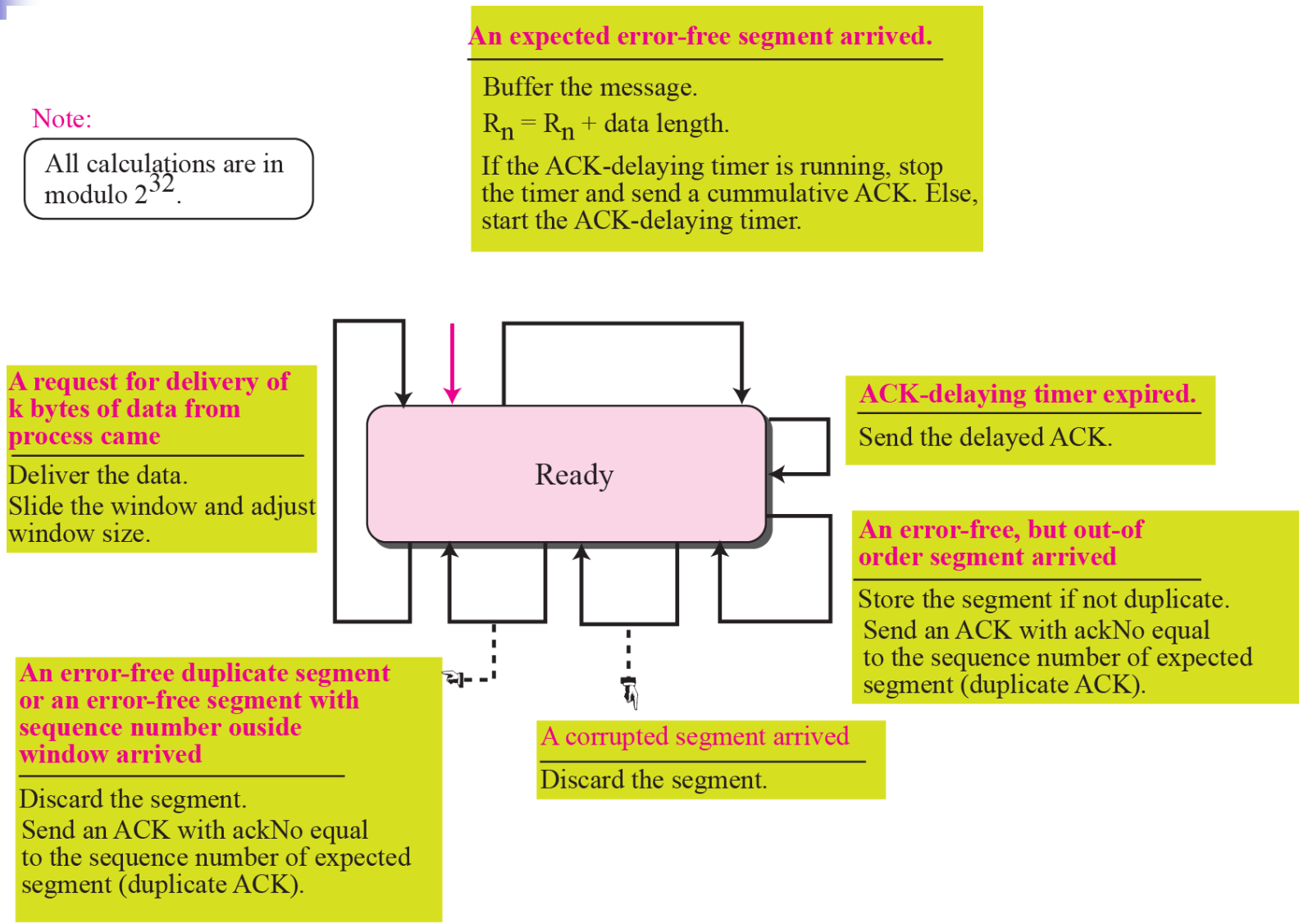


Figure 15.29 *Normal operation*

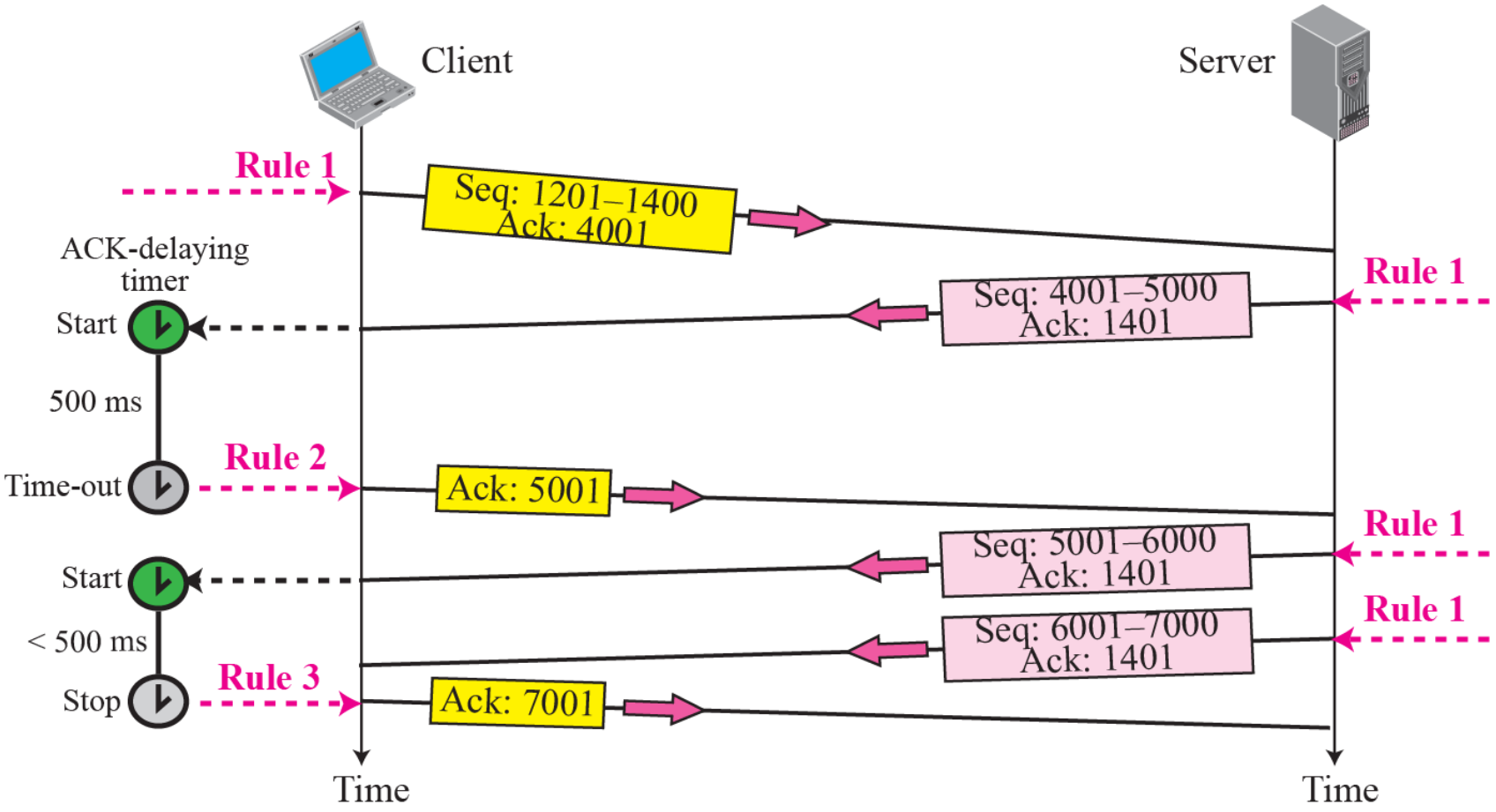
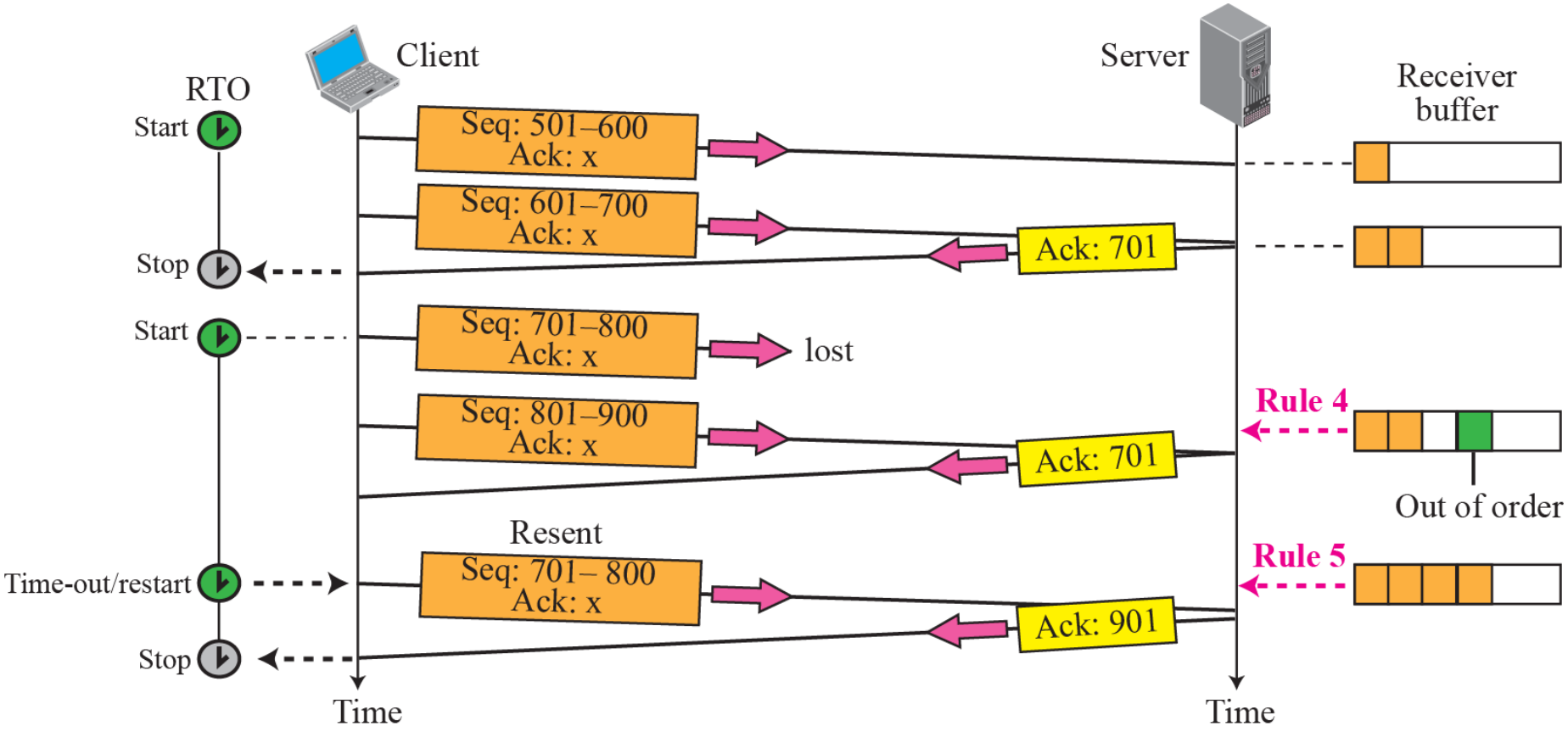


Figure 15.30 *Lost segment*



Note

The receiver TCP delivers only ordered data to the process.

Figure 15.31 *Fast retransmission*

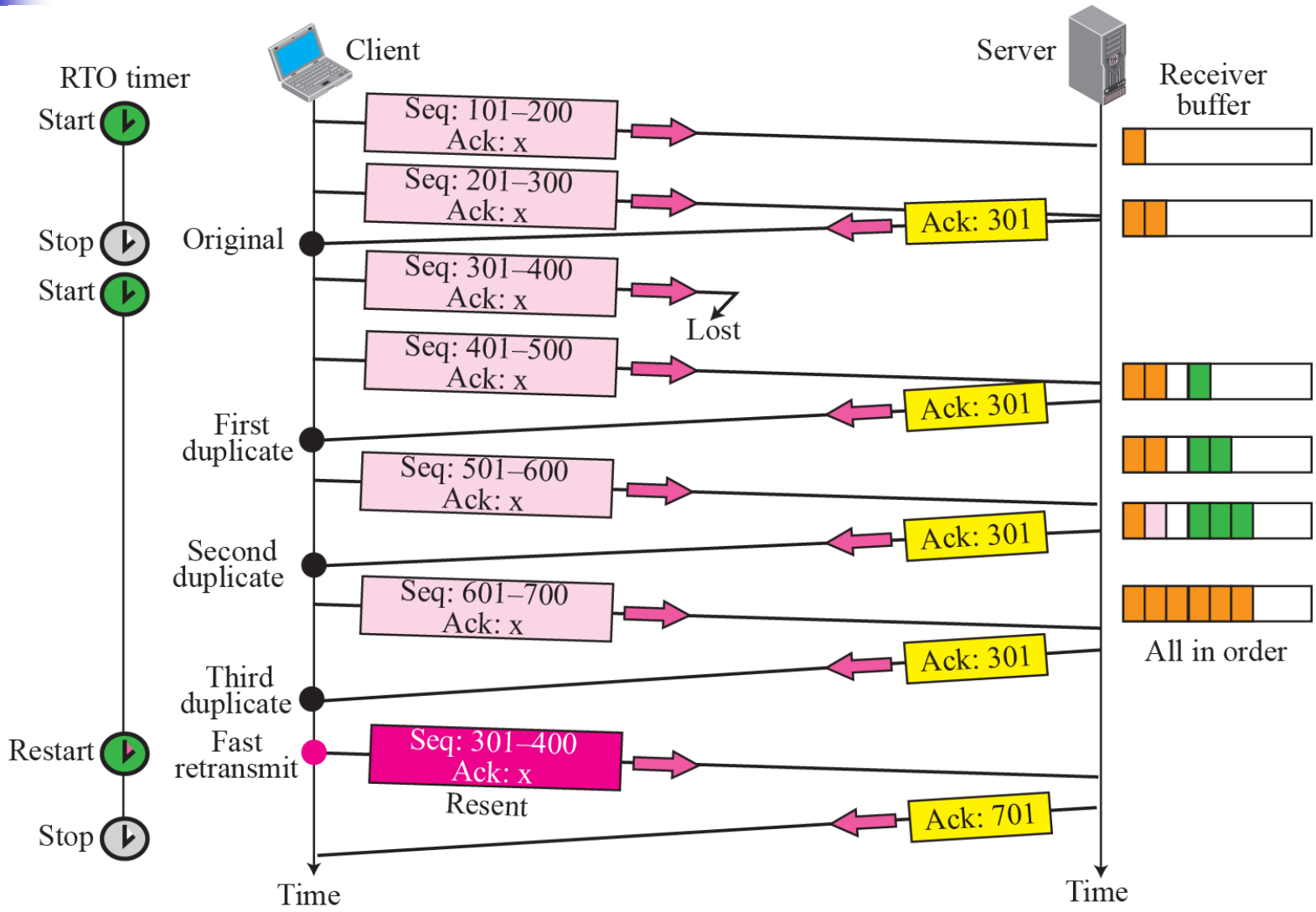


Figure 15.32 *Lost acknowledgment*

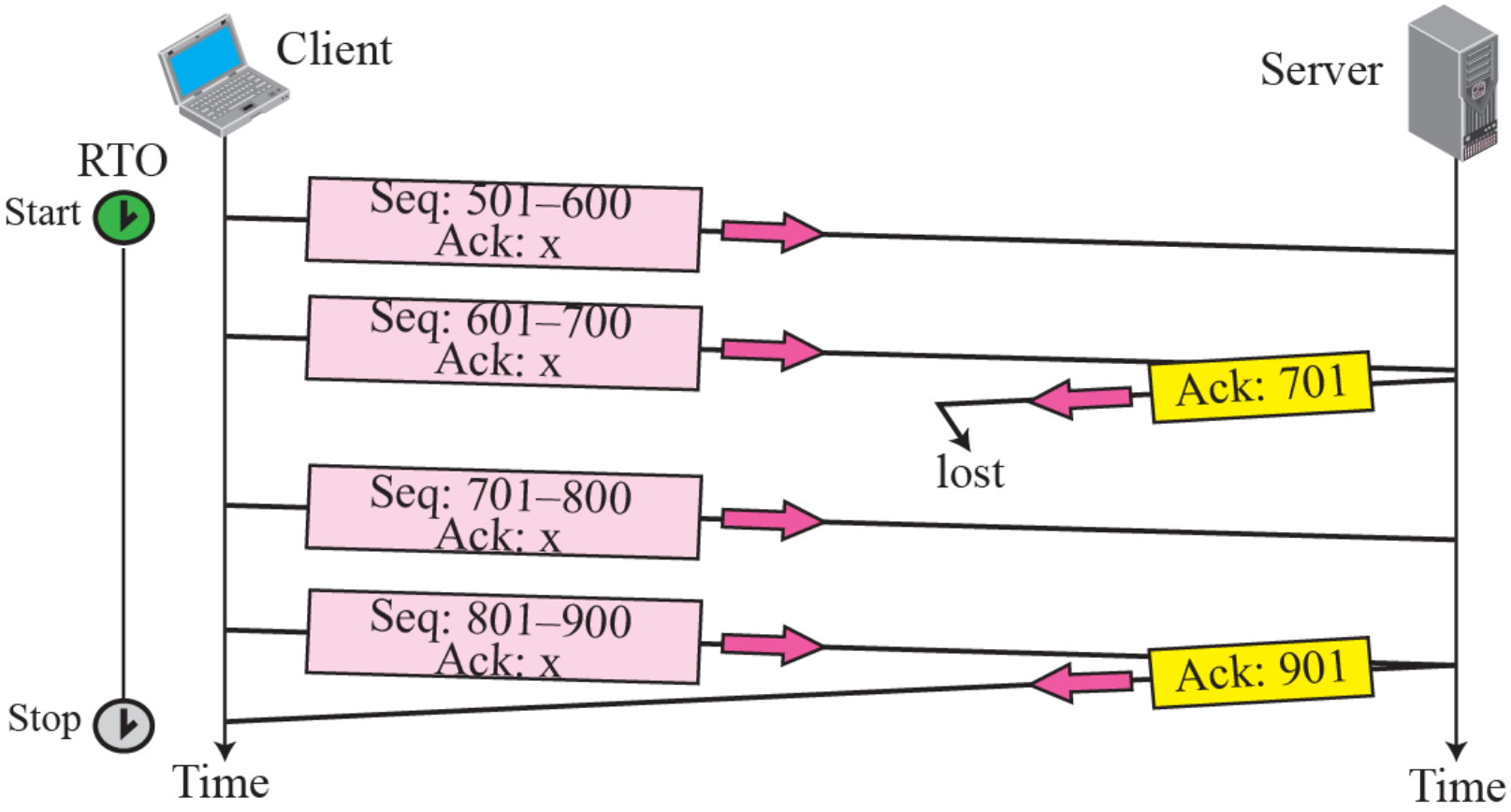
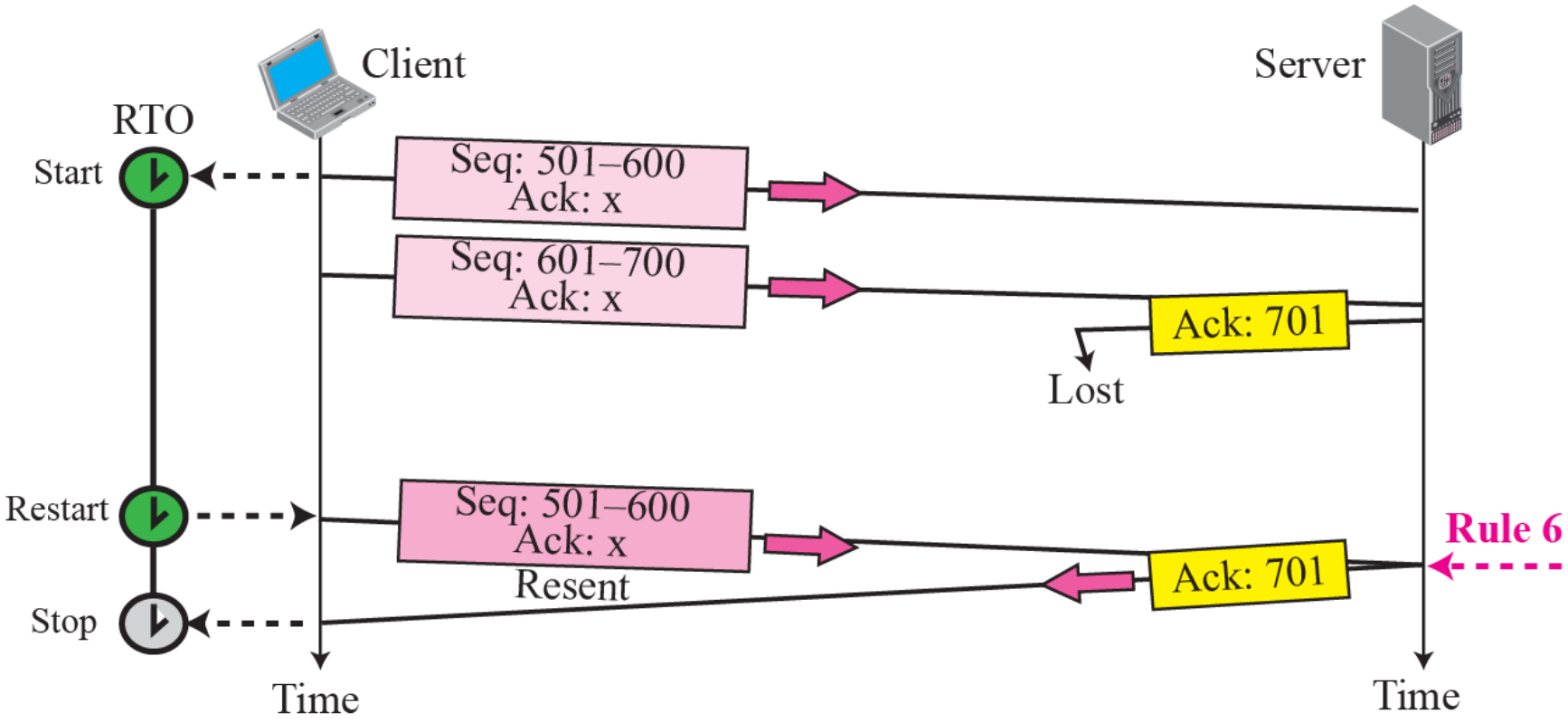


Figure 15.33 *Lost acknowledgment corrected by resending a segment*



Note

Lost acknowledgments may create deadlock if they are not properly handled.

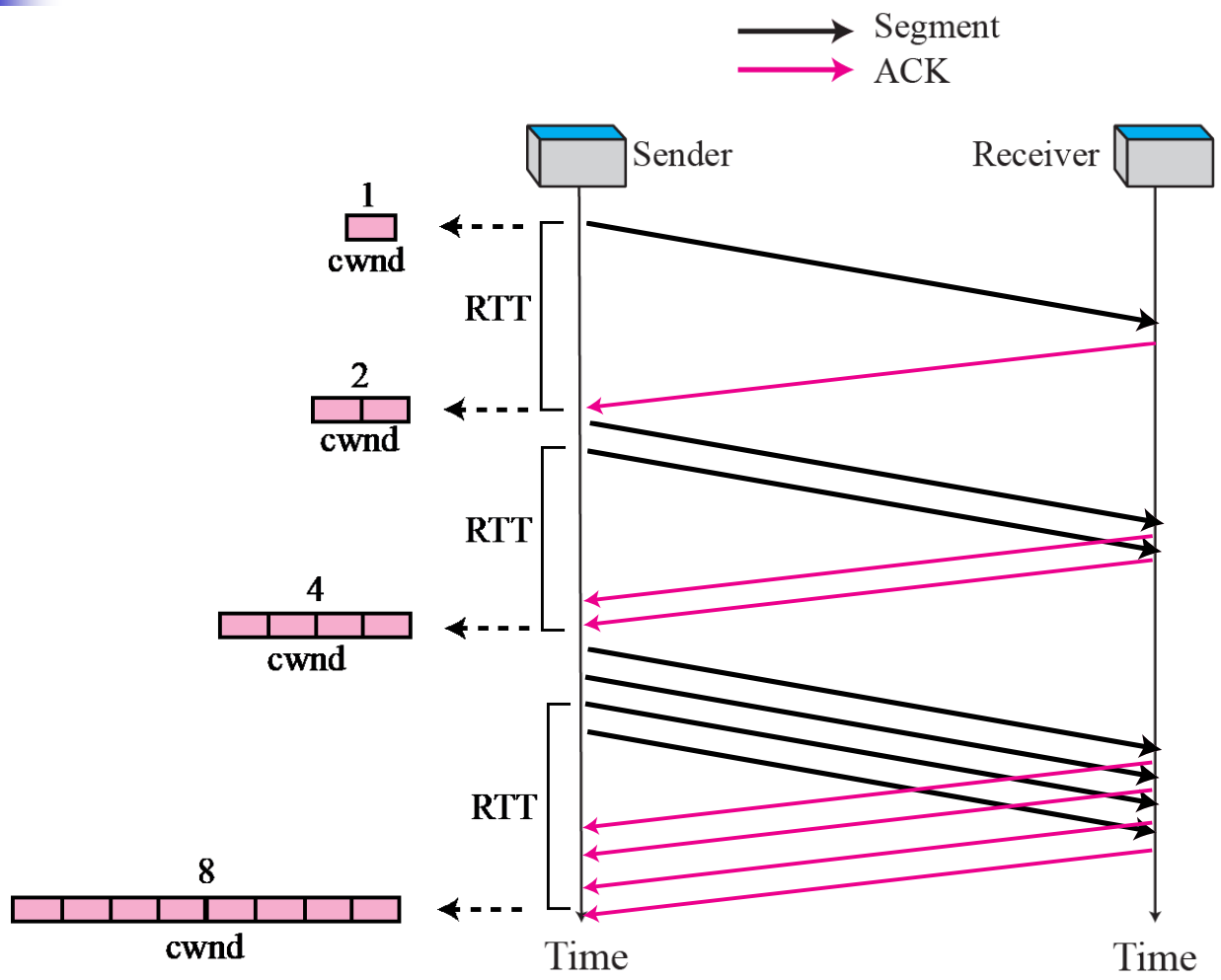
15-9 CONGESTION CONTROL

We discussed congestion control in Chapter 13. 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.

Topics Discussed in the Section

- ✓ **Congestion Window**
- ✓ **Congestion Policy**

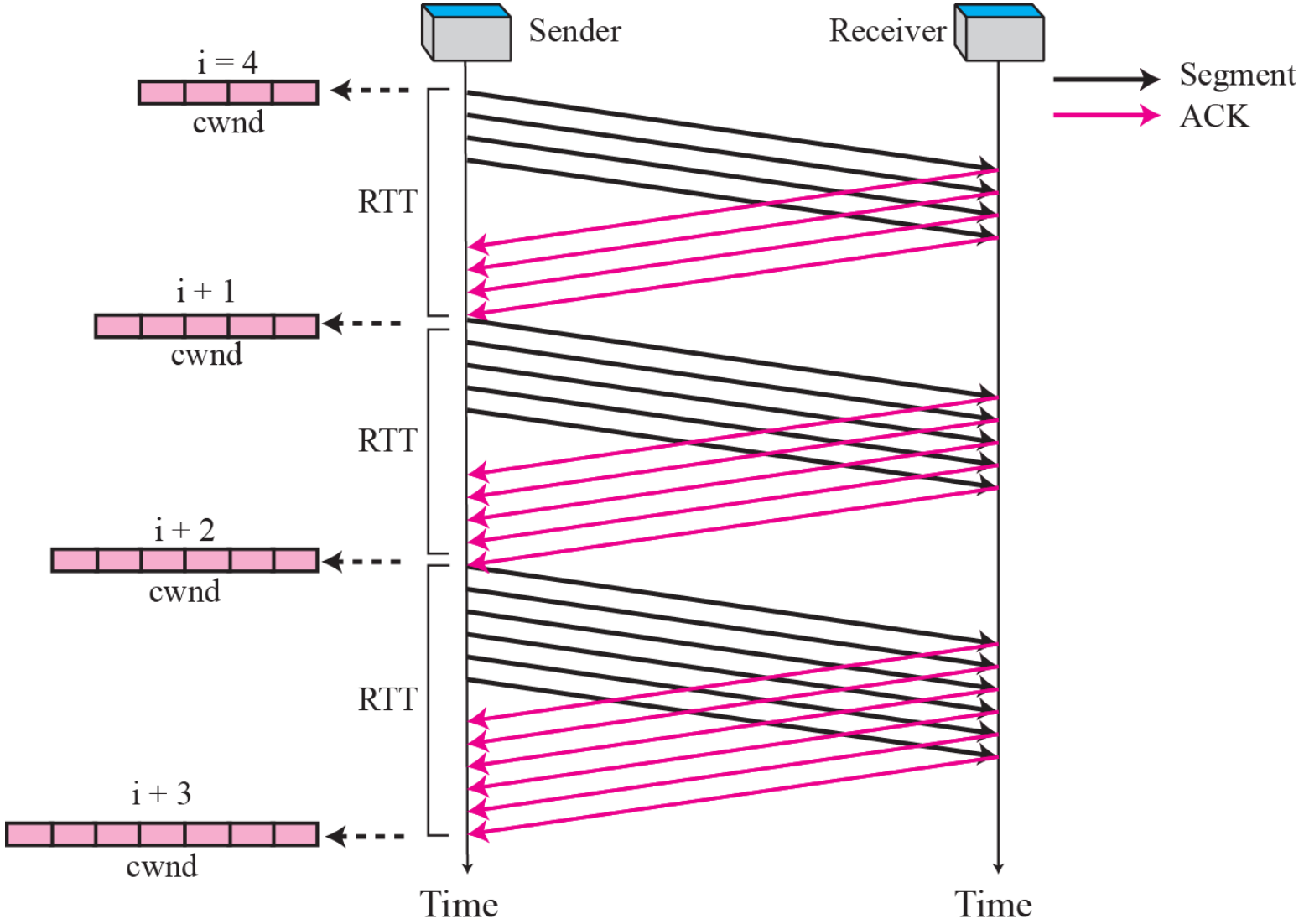
Figure 15.34 *Slow start, exponential increase*



Note

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

Figure 15.35 *Congestion avoidance, additive increase*



Note

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

Figure 15.36 *TCP Congestion policy summary*

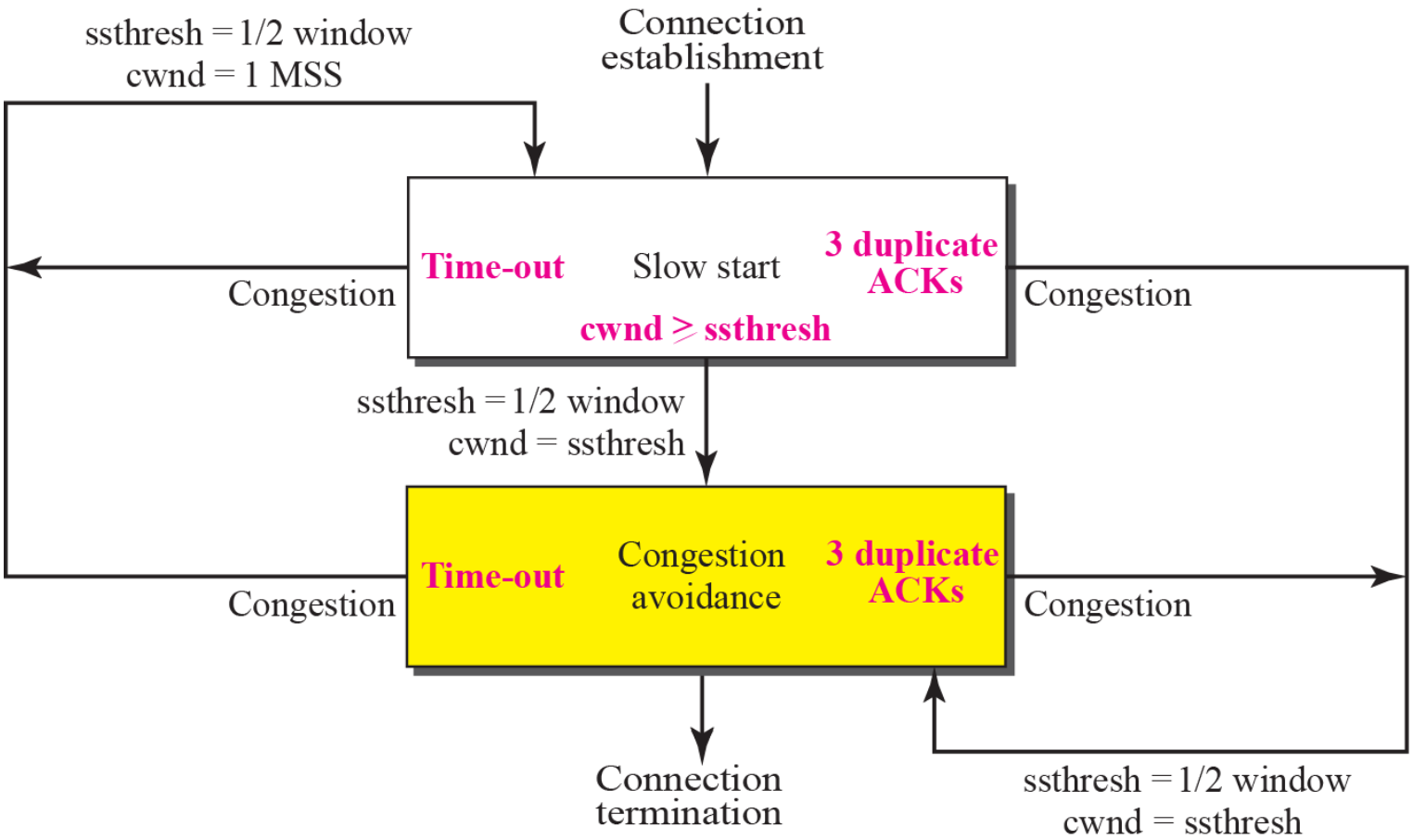
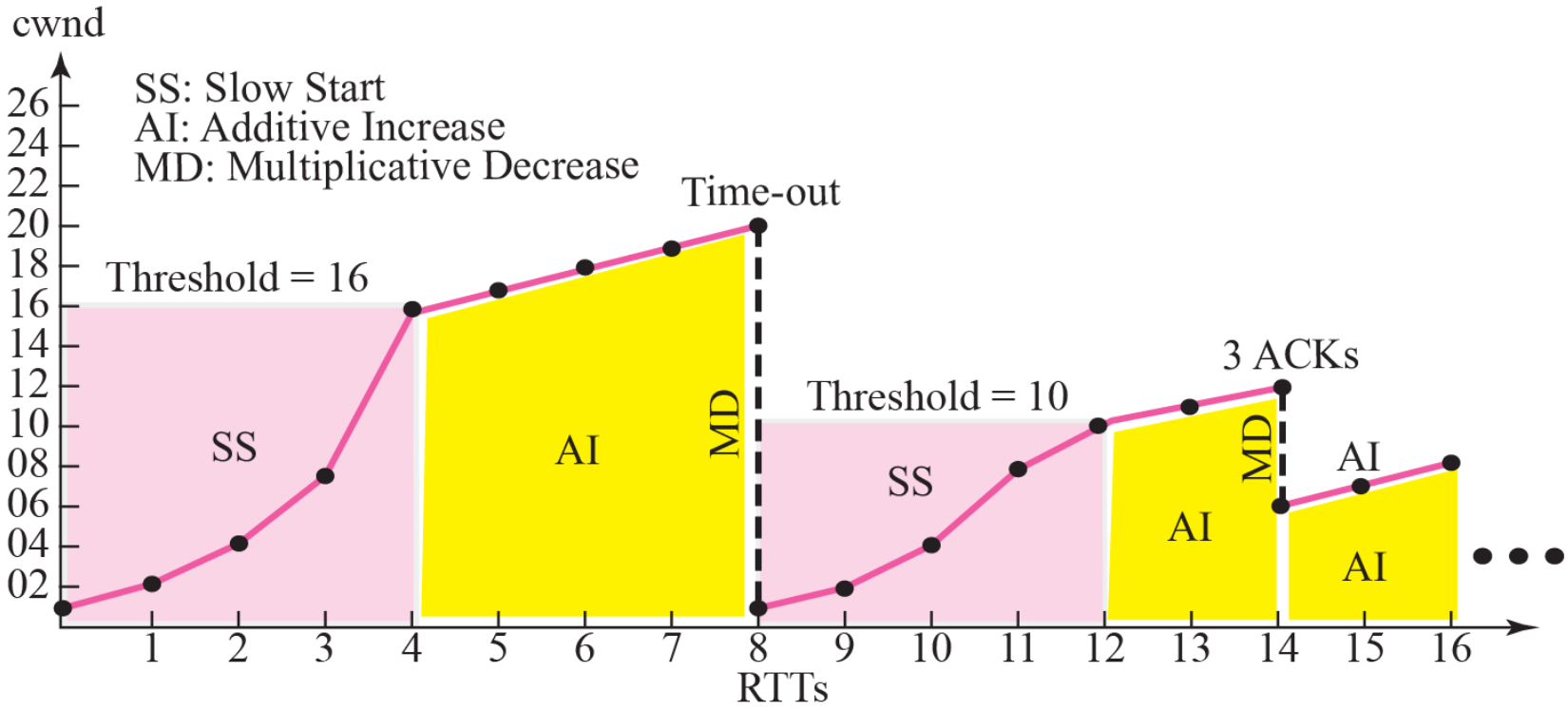


Figure 15.37 *Congestion example*



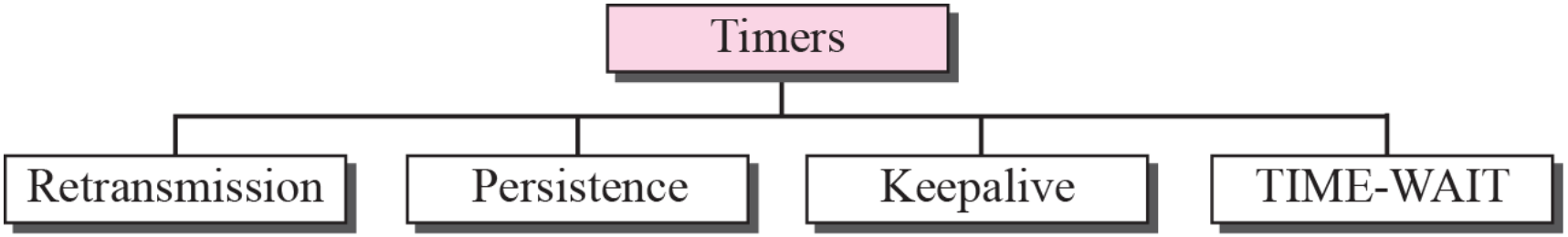
15-10 TCP TIMERS

To perform its operation smoothly, most TCP implementations use at least four timers as shown in Figure 15.38 (slide 83).

Topics Discussed in the Section

- ✓ **Retransmission Timer**
- ✓ **Persistence Timer**
- ✓ **Keepalive Timer**
- ✓ **TIME-WAIT Timer**

Figure 15.38 *TCP timers*



Note

In TCP, there can be only one RTT measurement in progress at any time.

Example 15.3

Let us give a hypothetical example. Figure 15.39 shows part of a connection. The figure shows the connection establishment and part of the data transfer phases.

1. When the SYN segment is sent, there is no value for RTTM, RTTS, or RTTD. The value of RTO is set to 6.00 seconds. The following shows the value of these variable at this moment:

$$RTO = 6$$

2. When the SYN+ACK segment arrives, RTTM is measured and is equal to 1.5 seconds.

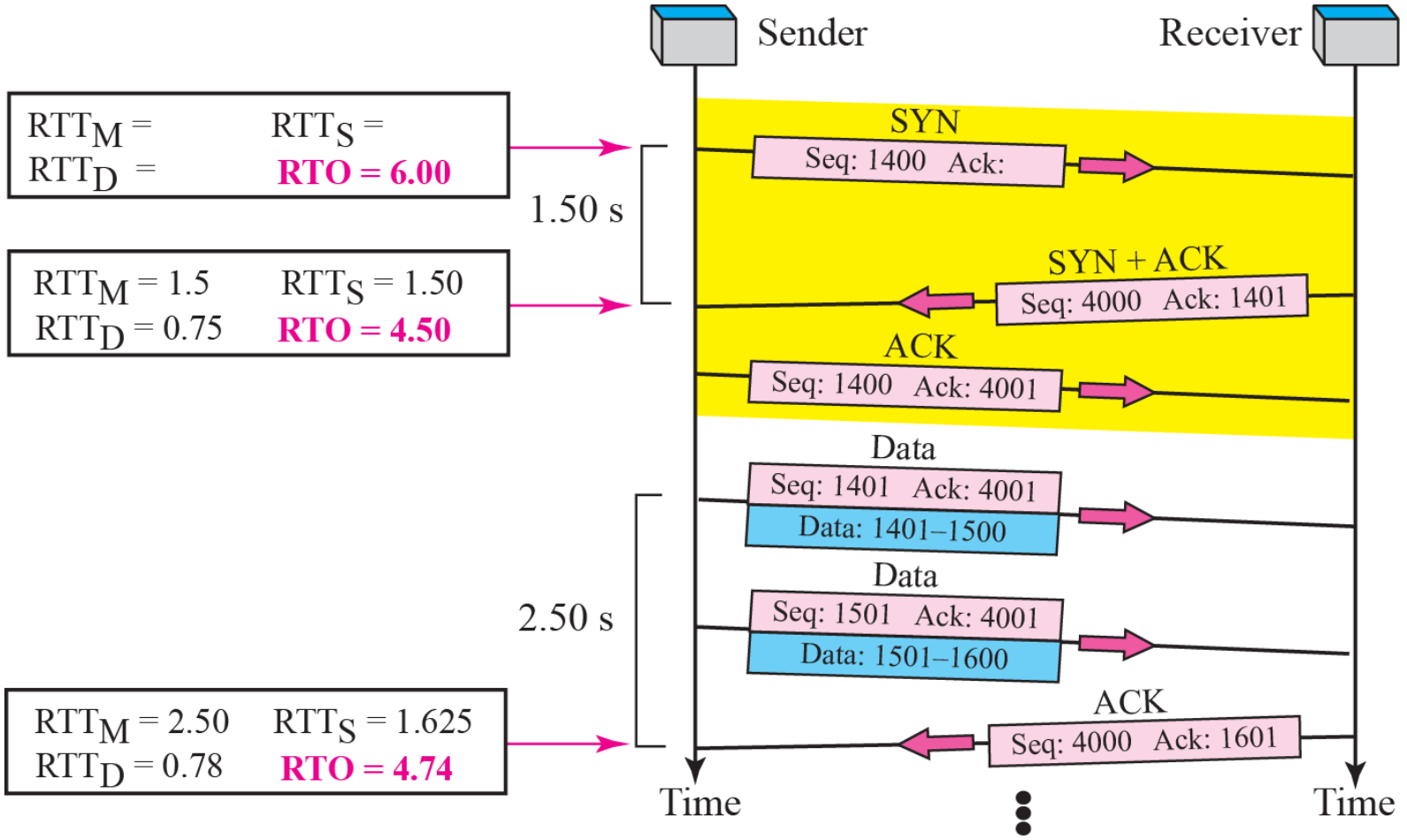
$$RTT_M = 1.5$$
$$RTT_S = 1.5$$
$$RTT_D = (1.5) / 2 = 0.75$$
$$RTO = 1.5 + 4 \times 0.75 = 4.5$$

Example 15.3 *Continued*

3. When the first data segment is sent, a new RTT measurement starts. No RTT measurement starts for the second data segment because a measurement is already in progress. The arrival of the last ACK segment is used to calculate the next value of RTTM. Although the last ACK segment acknowledges both data segments (cumulative), its arrival finalizes the value of RTTM for the first segment. The values of these variables are now as shown below.

$$\text{RTT}_M = 2.5$$
$$\text{RTT}_S = 7/8 \times 1.5 + (1/8) \times 2.5 = 1.625$$
$$\text{RTT}_D = 3/4\ (7.5) + (1/4) \times |1.625 - 2.5| = 0.78$$
$$\text{RTO} = 1.625 + 4 \times 0.78 = 4.74$$

Figure 15.39 *Example 15.3*



Note

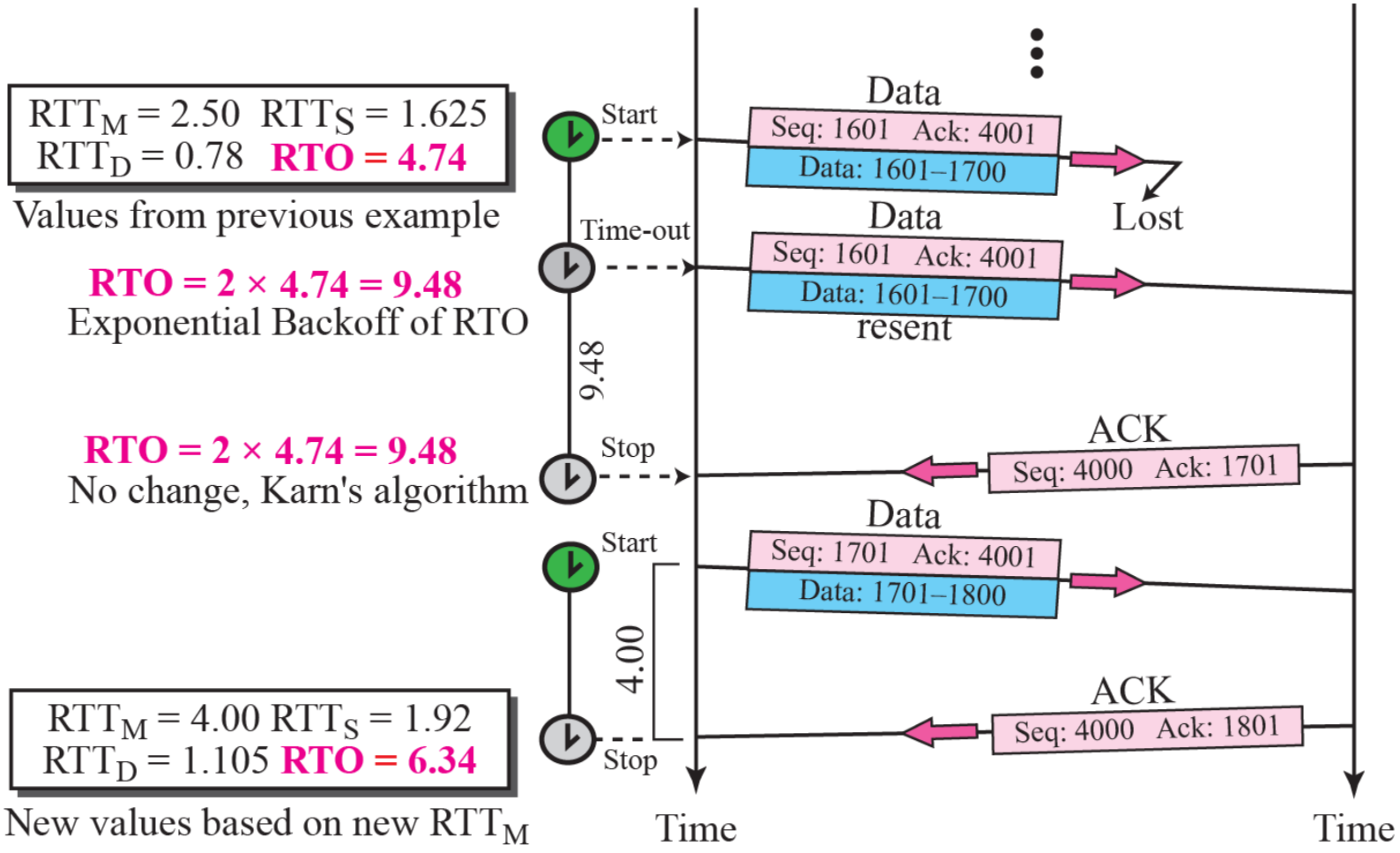
TCP does not consider the RTT of a retransmitted segment in its calculation of a new RTO.

Example 15.4

Figure 15.40 is a continuation of the previous example. There is retransmission and Karn's algorithm is applied.

The first segment in the figure is sent, but lost. The RTO timer expires after 4.74 seconds. The segment is retransmitted and the timer is set to 9.48, twice the previous value of RTO. This time an ACK is received before the time-out. We wait until we send a new segment and receive the ACK for it before recalculating the RTO (Karn's algorithm).

Figure 15.40 *Example 15.4*



User Datagram Program (UDP)

OBJECTIVES:

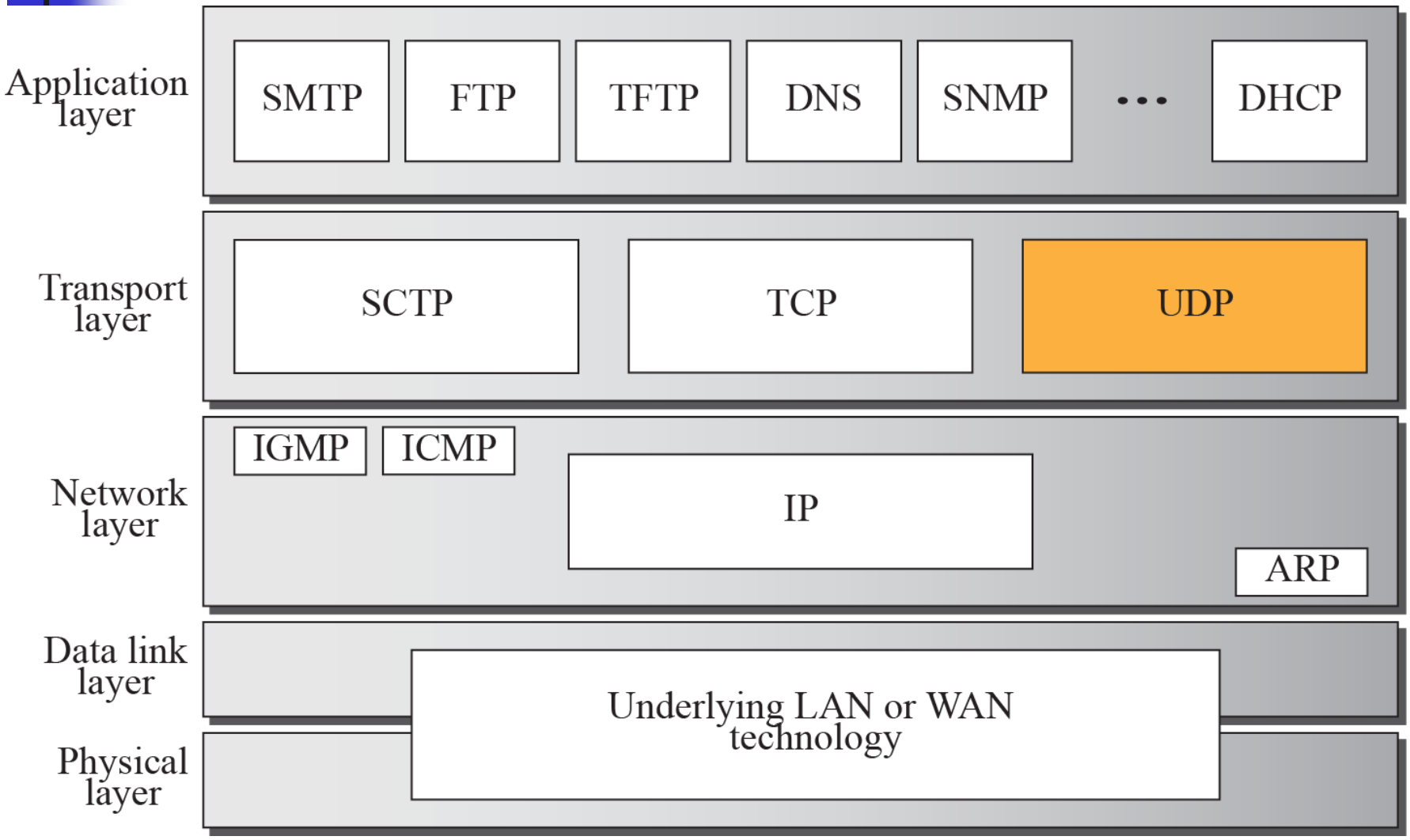
- To introduce UDP and show its relationship to other protocols in the TCP/IP protocol suite.
- To explain the format of a UDP packet and discuss the use of each field in the header.
- To discuss the services provided by the UDP such as process-to-process delivery, multiplexing/demultiplexing, and queuing.
- To show how to calculate the optional checksum and the sender the needs to add a pseudoheader to the packet when calculating the checksum.
- To discuss how some application programs can benefit from the simplicity of UDP.
- To briefly discuss the structure of the UDP package.

Chapter Outline

14-1 INTRODUCTION

Figure 14.1 shows the relationship of the User Datagram Protocol (UDP) to the other protocols and layers of the TCP/IP protocol suite: UDP is located between the application layer and the IP layer, and serves as the intermediary between the application programs and the network operations.

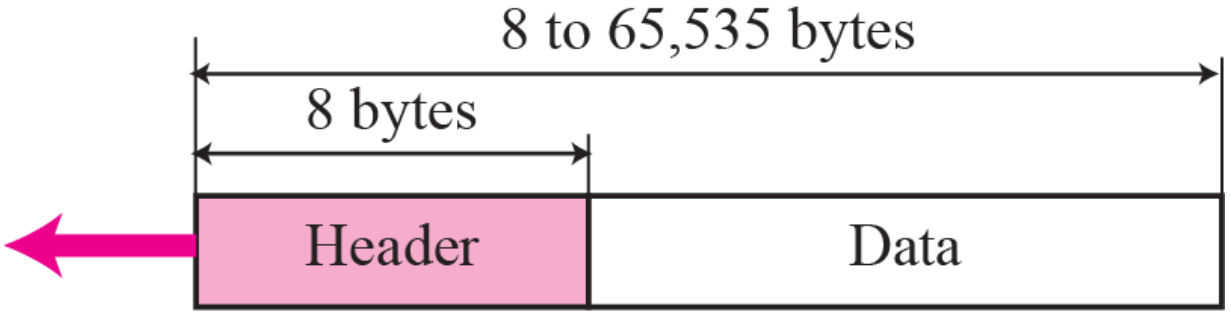
Figure 14.1 *Position of UDP in the TCP/IP protocol suite*



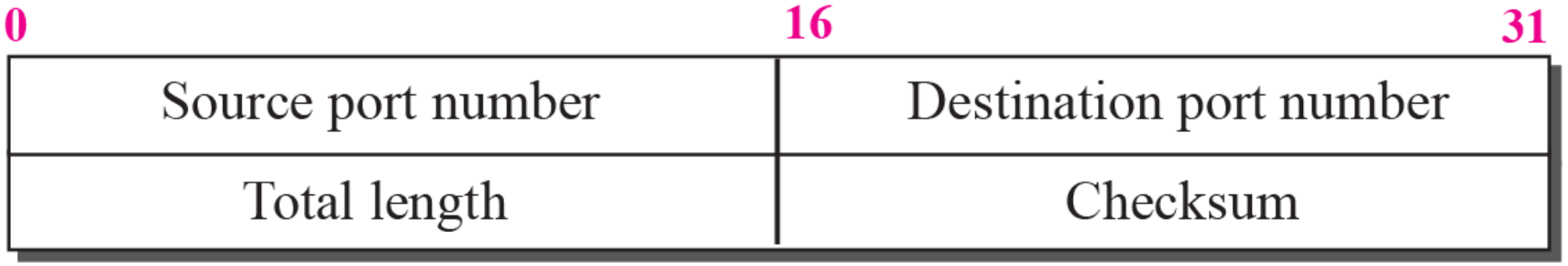
14-2 USER DATAGRAM

UDP packets, called user datagrams, have a fixed-size header of 8 bytes. Figure 14.2 shows the format of a user datagram.

Figure 14.2 *User datagram format*



a. UDP user datagram



b. Header format

Example 14.1

The following is a dump of a UDP header in hexadecimal format.

```
CB84000D001C001C
```

a. What is the source port number?
b. What is the destination port number?
c. What is the total length of the user datagram?
d. What is the length of the data?
e. Is the packet directed from a client to a server or vice versa?
f. What is the client process?

Example 14.1 *Continued*

Solution

a. The source port number is the first four hexadecimal digits $(CB84)_{16}$ or 52100.

b. The destination port number is the second four hexadecimal digits $(000D)_{16}$ or 13.

c. The third four hexadecimal digits $(001C)_{16}$ define the length of the whole UDP packet as 28 bytes.

d. The length of the data is the length of the whole packet minus the length of the header, or 28 – 8 = 20 bytes.

e. Since the destination port number is 13 (well-known port), the packet is from the client to the server.

f. The client process is the Daytime (see Table 14.1).

14-3 UDP Services

We discussed the general services provided by a transport layer protocol in Chapter 13. In this section, we discuss what portions of those general services are provided by UDP.

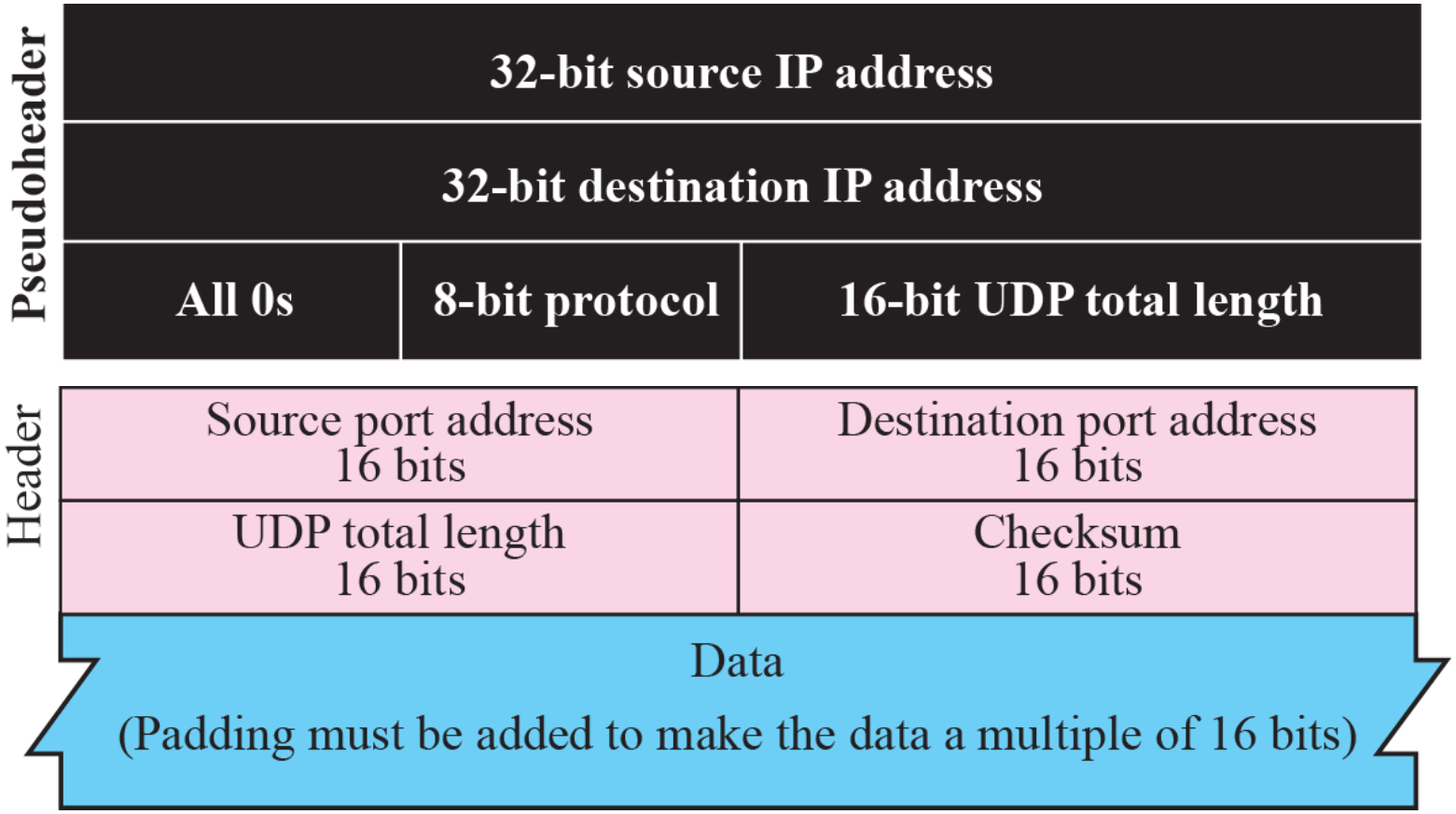
Topics Discussed in the Section

- ✓ Process-to-Process Communication
- ✓ Connectionless Service
- ✓ Flow Control
- ✓ Error Control
- ✓ Congestion Control
- ✓ Encapsulation and Decapsulation
- ✓ Queuing
- ✓ Multiplexing and Demultiplexing
- ✓ Comparison between UDP and Generic Simple Protocol

Table 14.1 *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	Domain	Domain Name Service (DNS)
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)

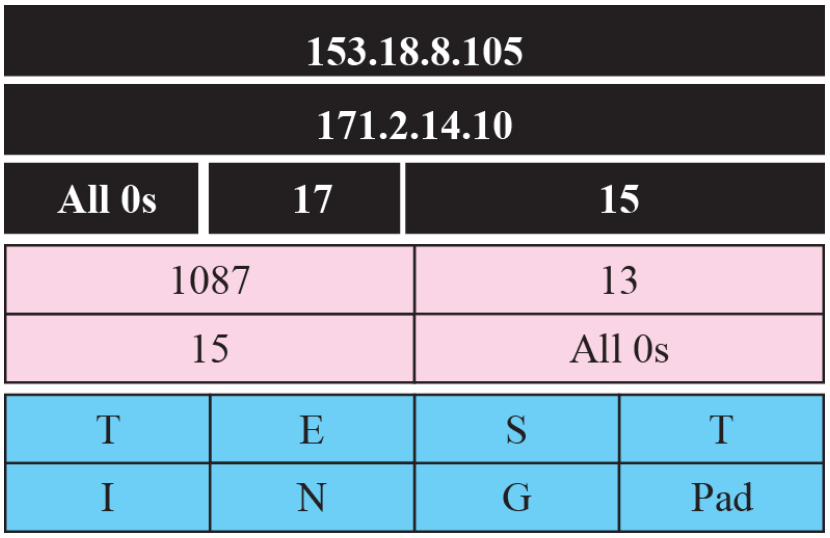
Figure 14.3 *Pseudoheader for checksum calculation*

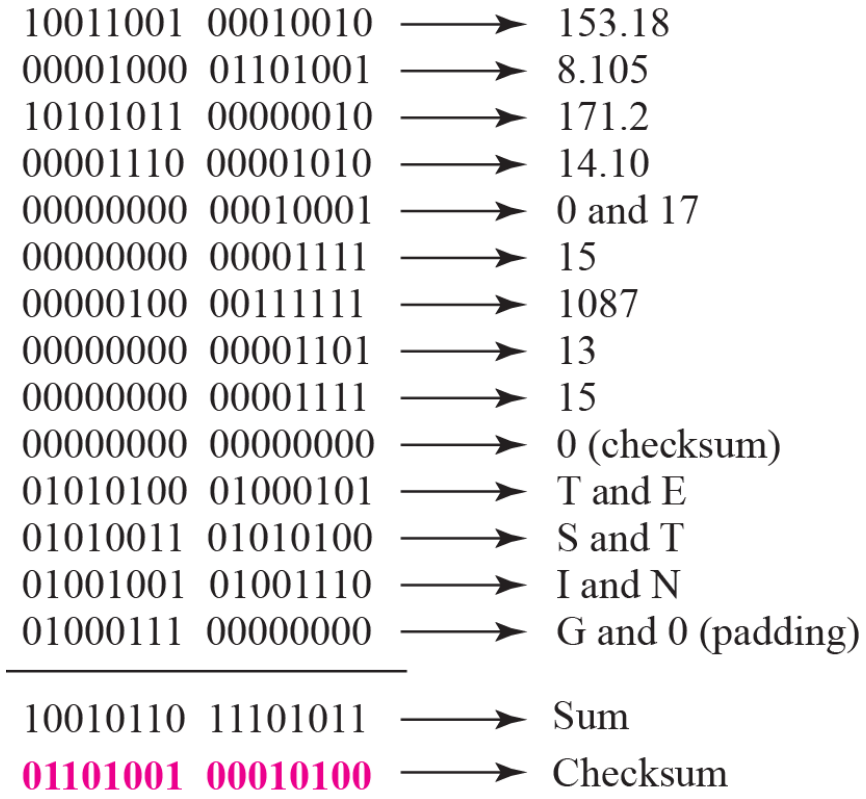


Example 14.2

Figure 14.4 shows the checksum calculation for a very small user datagram with only 7 bytes of data. Because the number of bytes of data is odd, padding is added for checksum calculation. The pseudoheader as well as the padding will be dropped when the user datagram is delivered to IP (see Appendix F).

Figure 14.4 *Checksum calculation for a simple UDP user datagram*





Example 14.3

What value is sent for the checksum in one of the following hypothetical situations?

a. The sender decides not to include the checksum.

b. The sender decides to include the checksum, but the value of the sum is all 1s.

c. The sender decides to include the checksum, but the value of the sum is all 0s.

Example 14.3 *Continued*

Solution

a. The value sent for the checksum field is all 0s to show that the checksum is not calculated.

b. When the sender complements the sum, the result is all 0s; the sender complements the result again before sending. The value sent for the checksum is all 1s. The second complement operation is needed to avoid confusion with the case in part a.

c. This situation never happens because it implies that the value of every term included in the calculation of the sum is all 0s, which is impossible; some fields in the pseudoheader have nonzero values (see Appendix D).

Figure 14.5 *Encapsulation and decapsulation*

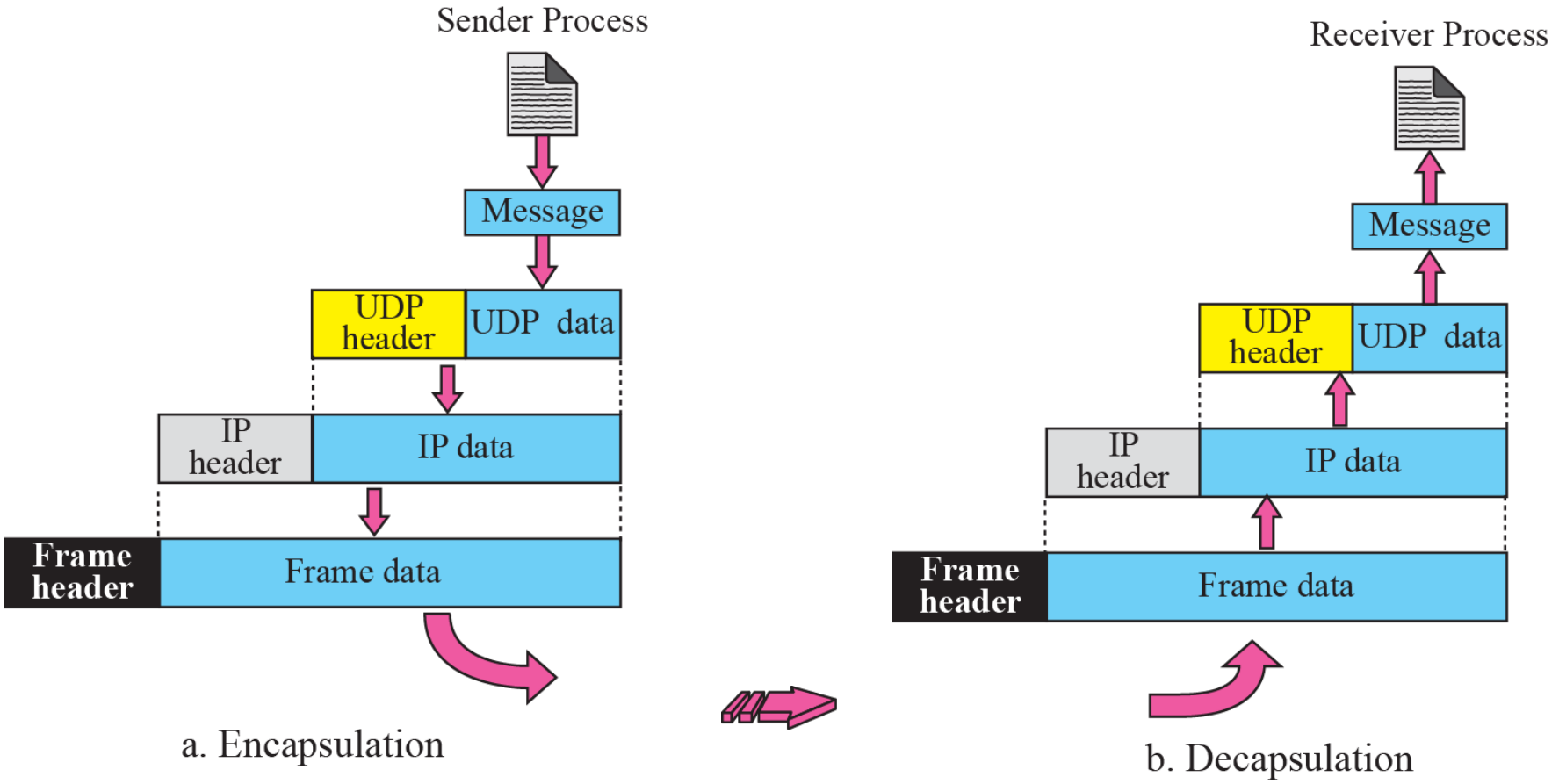


Figure 14.6 *Queues in UDP*

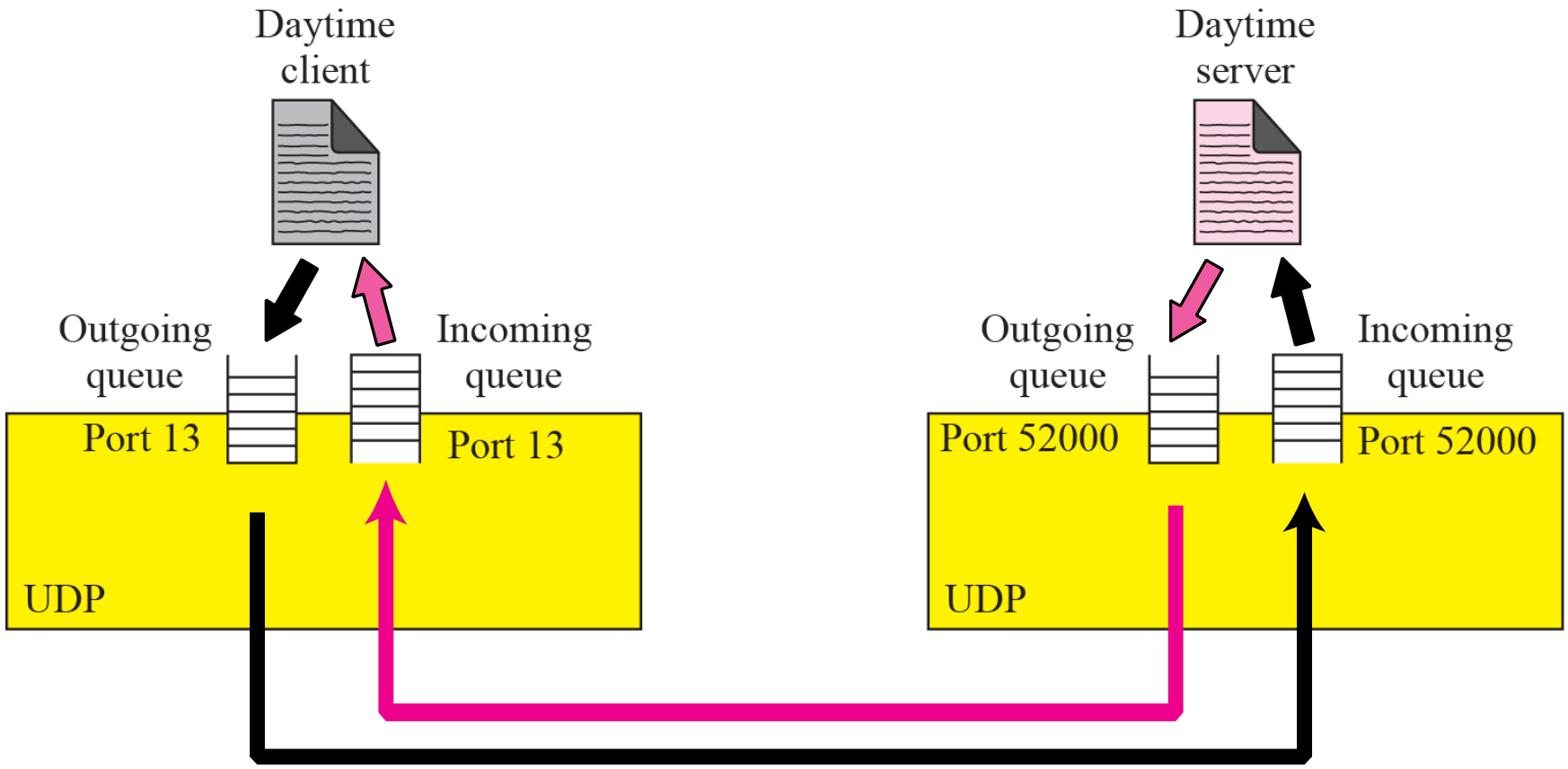
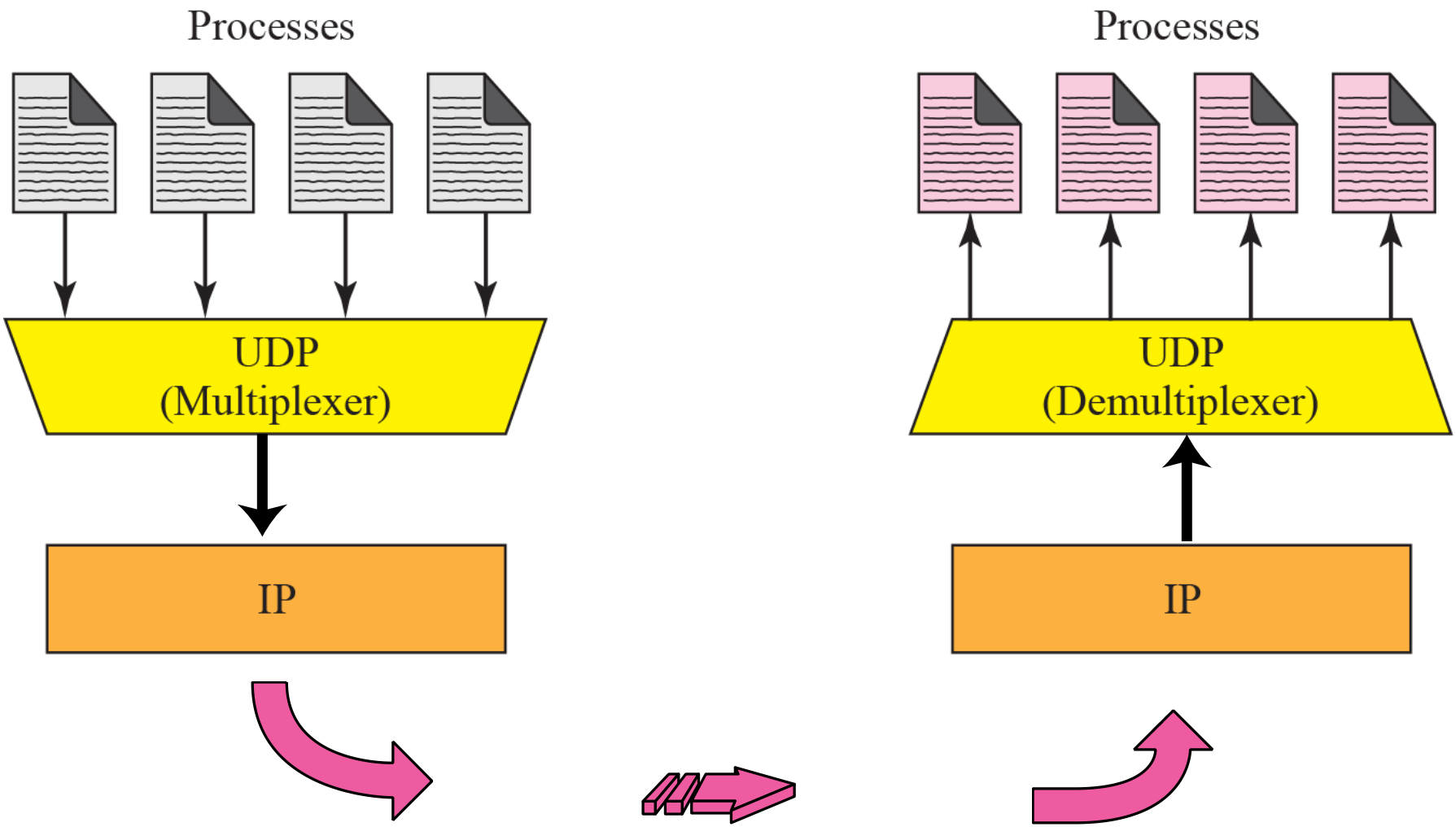


Figure 14.7 *Multiplexing and demultiplexing*



Note

UDP is an example of the connectionless simple protocol we discussed in Chapter 13 with the exception of an optional checksum added to packets for error detection.

14-4 UDP APPLICATION

Although UDP meets almost none of the criteria we mentioned in Chapter 13 for a reliable transport-layer protocol, UDP is preferable for some applications. The reason is that some services may have some side effects that are either unacceptable or not preferable. An application designer needs sometimes to compromise to get the optimum.

Topics Discussed in the Section

- ✓ **UDP Features**
- ✓ **Typical Applications**

Example 14.4

A client-server application such as DNS (see Chapter 19) uses the services of UDP because a client needs to send a short request to a server and to receive a quick response from it. The request and response can each fit in one user datagram. Since only one message is exchanged in each direction, the connectionless feature is not an issue; the client or server does not worry that messages are delivered out of order.

Example 14.5

A client-server application such as SMTP (see Chapter 23), which is used in electronic mail, cannot use the services of UDP because a user can send a long e-mail message, which may include multimedia (images, audio, or video). If the application uses UDP and the message does not fit in one single user datagram, the message must be split by the application into different user datagrams. Here the connectionless service may create problems. The user datagrams may arrive and be delivered to the receiver application out of order. The receiver application may not be able to reorder the pieces. This means the connectionless service has a disadvantage for an application program that sends long messages.

Example 14.6

Assume we are downloading a very large text file from the Internet. We definitely need to use a transport layer that provides reliable service. We don't want part of the file to be missing or corrupted when we open the file. The delay created between the delivery of the parts are not an overriding concern for us; we wait until the whole file is composed before looking at it. In this case, UDP is not a suitable transport layer.

Example 14.7

Assume we are watching a real-time stream video on our computer. Such a program is considered a long file; it is divided into many small parts and broadcast in real time. The parts of the message are sent one after another. If the transport layer is supposed to resend a corrupted or lost frame, the synchronizing of the whole transmission may be lost. The viewer suddenly sees a blank screen and needs to wait until the second transmission arrives. This is not tolerable. However, if each small part of the screen is sent using one single user datagram, the receiving UDP can easily ignore the corrupted or lost packet and deliver the rest to the application program. That part of the screen is blank for a very short period of the time, which most viewers do not even notice. However, video cannot be viewed out of order, so streaming audio, video, and voice applications that run over UDP must reorder or drop frames that are out of sequence.

14-5 UDP PACKAGE

To show how UDP handles the sending and receiving of UDP packets, we present a simple version of the UDP package.

We can say that the UDP package involves five components: a control-block table, input queues, a control-block module, an input module, and an output module.