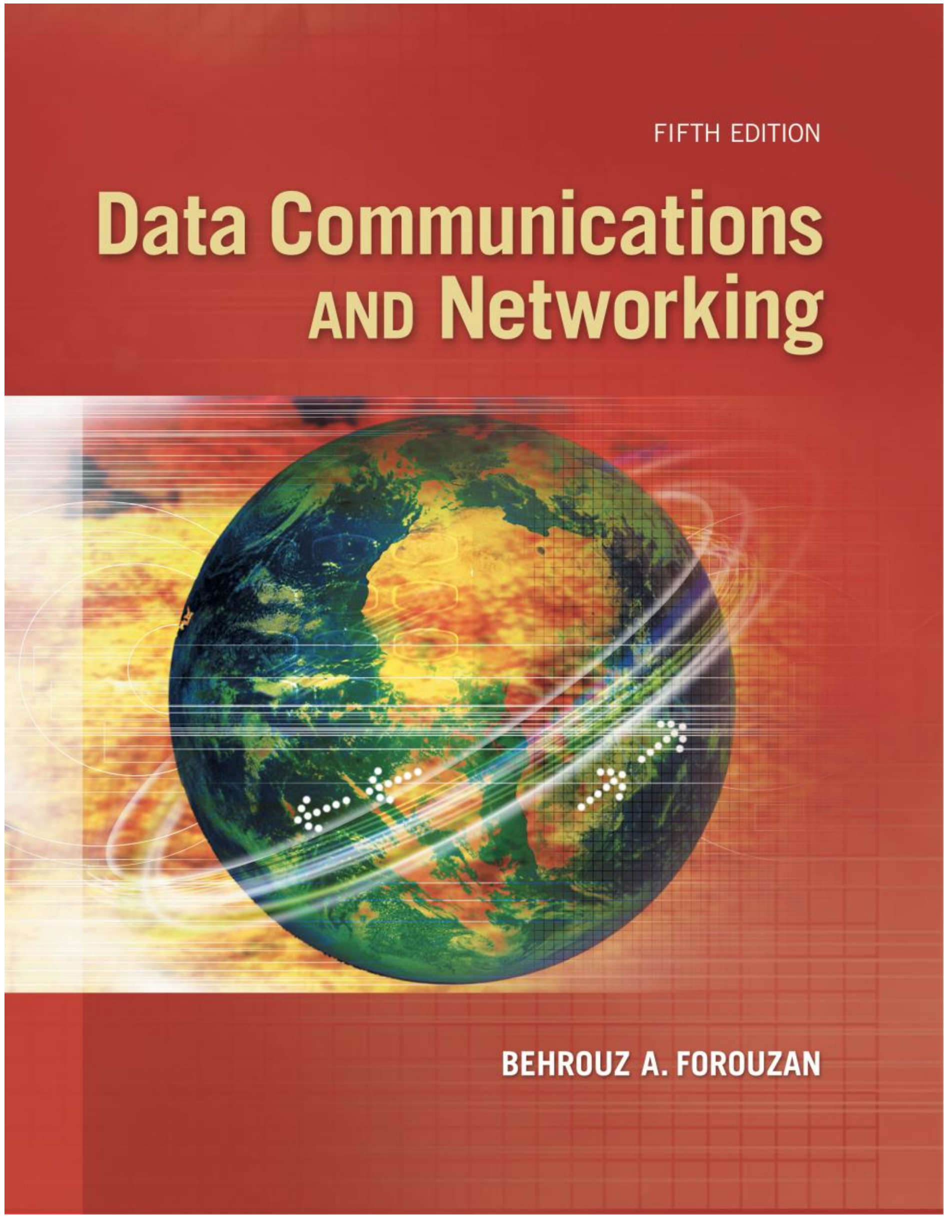
Chapter 23

Introduction To Transport Layer



Chapter 23: Outline

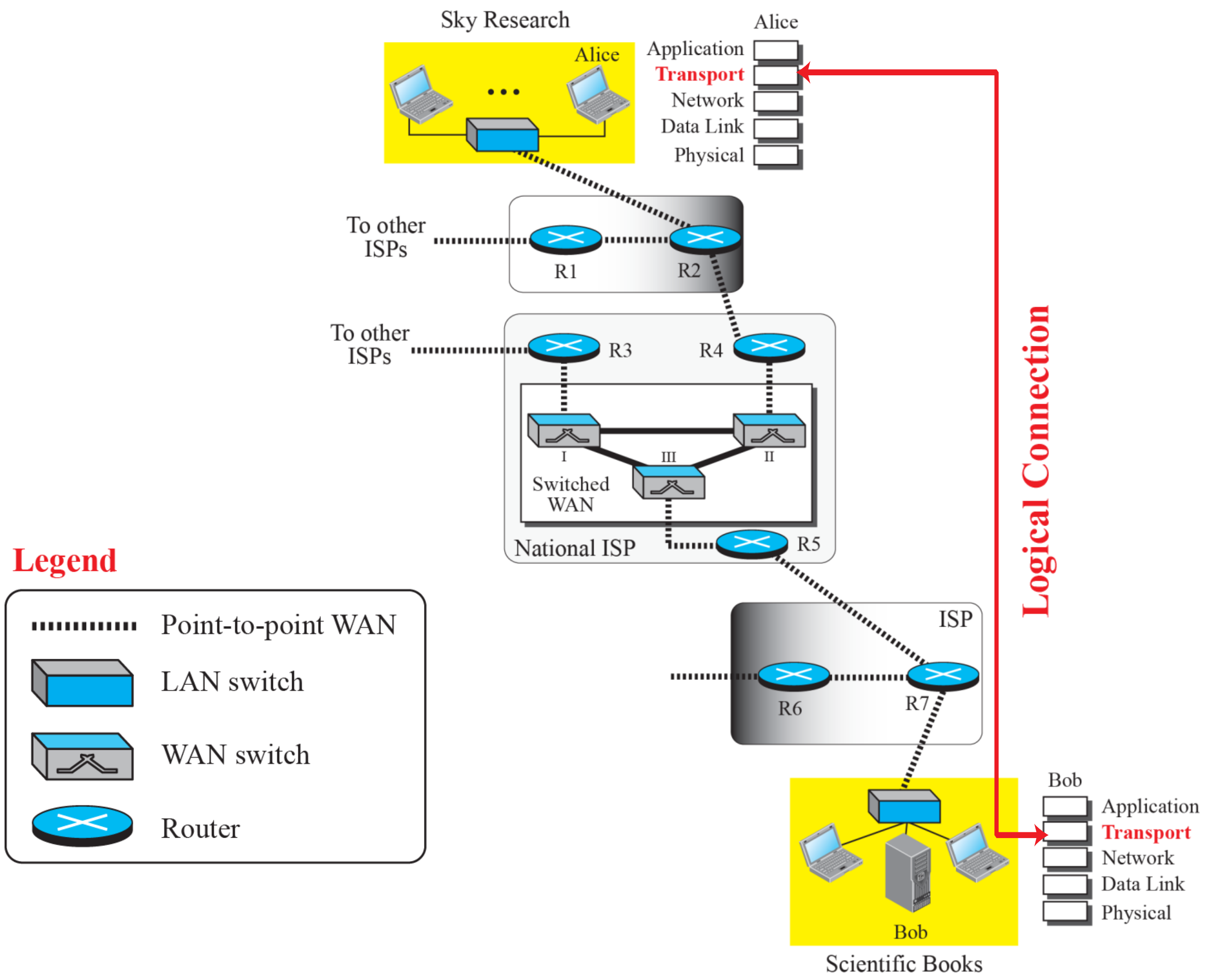
23.1 INTRODUCTION

23.2 TRANSPORT-LAYER PROTOCOLS

23-1 INTRODUCTION

The transport layer is located between the application layer and the network layer. It provides a process-to-process communication between two application layers, one at the local host and the other at the remote host. Communication is provided using a logical connection. Figure 23.1 shows the idea behind this logical connection.

Figure 23.1: Logical connection at the transport layer



23.23.1 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.

Figure 23.2: Network layer versus transport layer

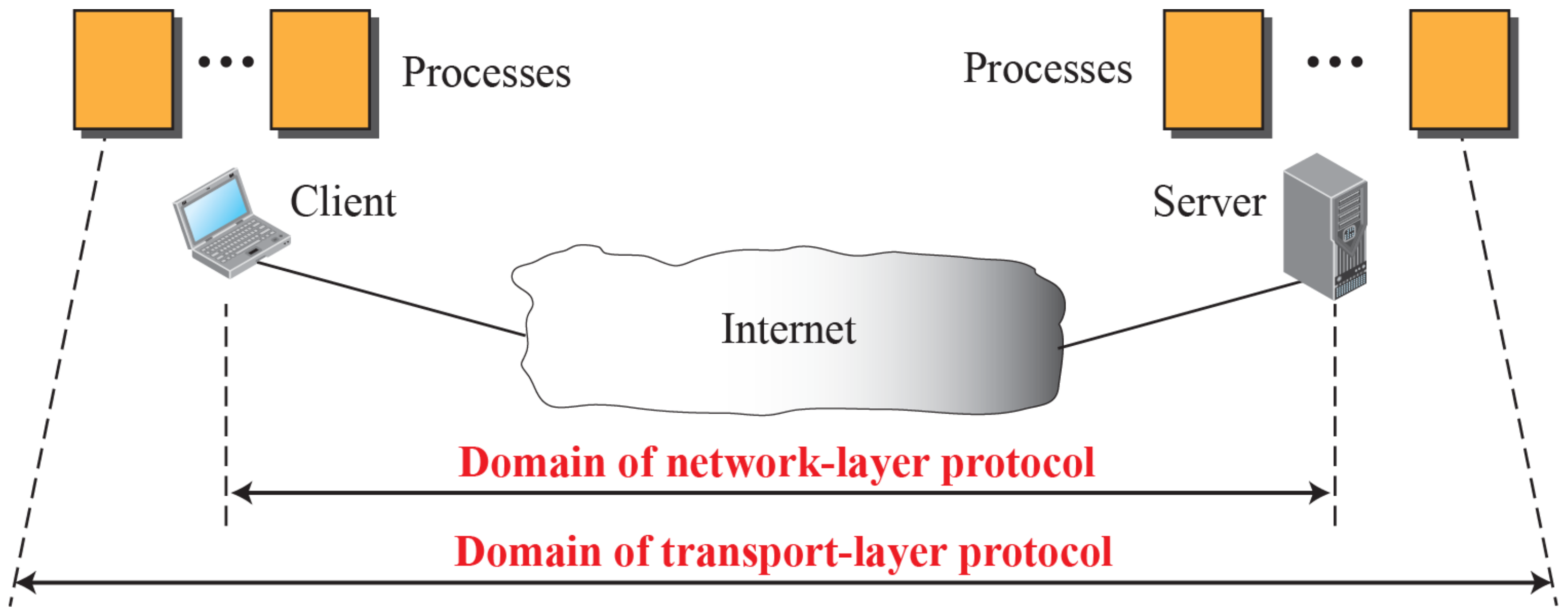

Processes
Processes
Client
Server
Internet
Domain of network-layer protocol
Domain of transport-layer protocol

Figure 23.3: Port numbers

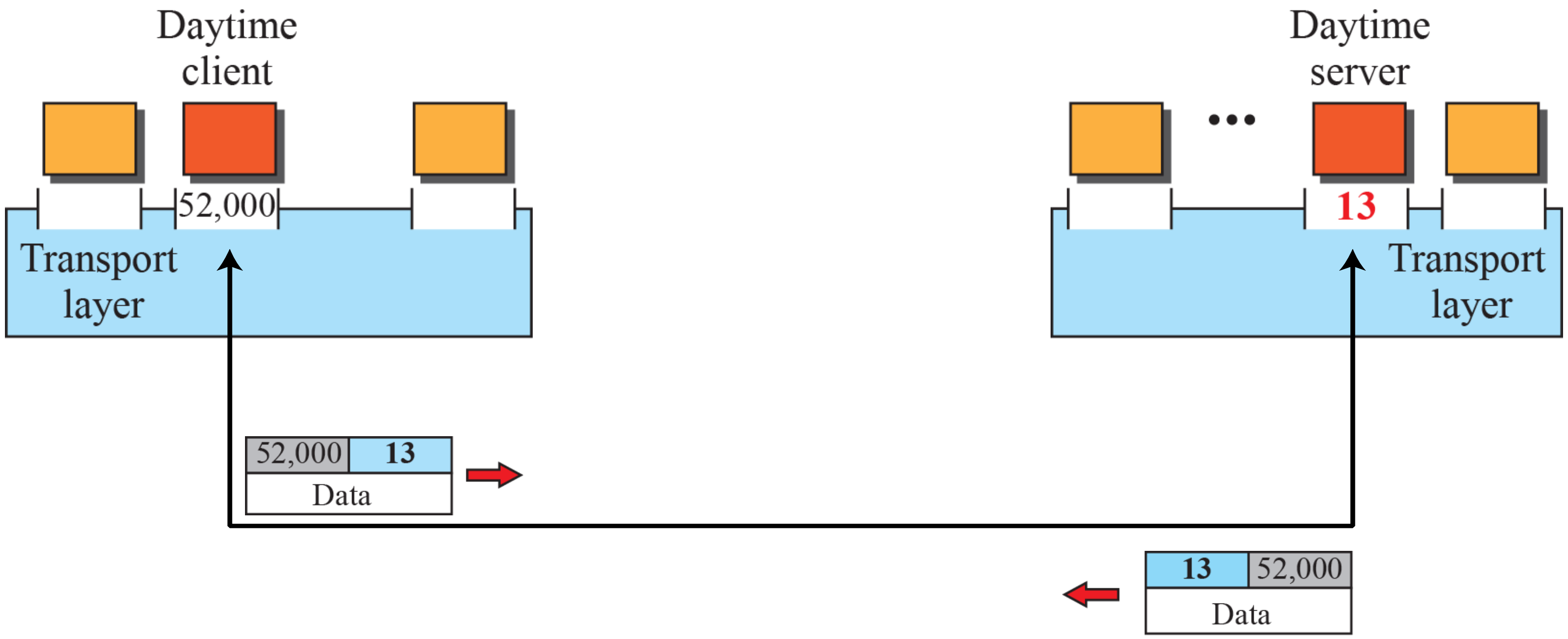


Figure 23.4: IP addresses versus port numbers

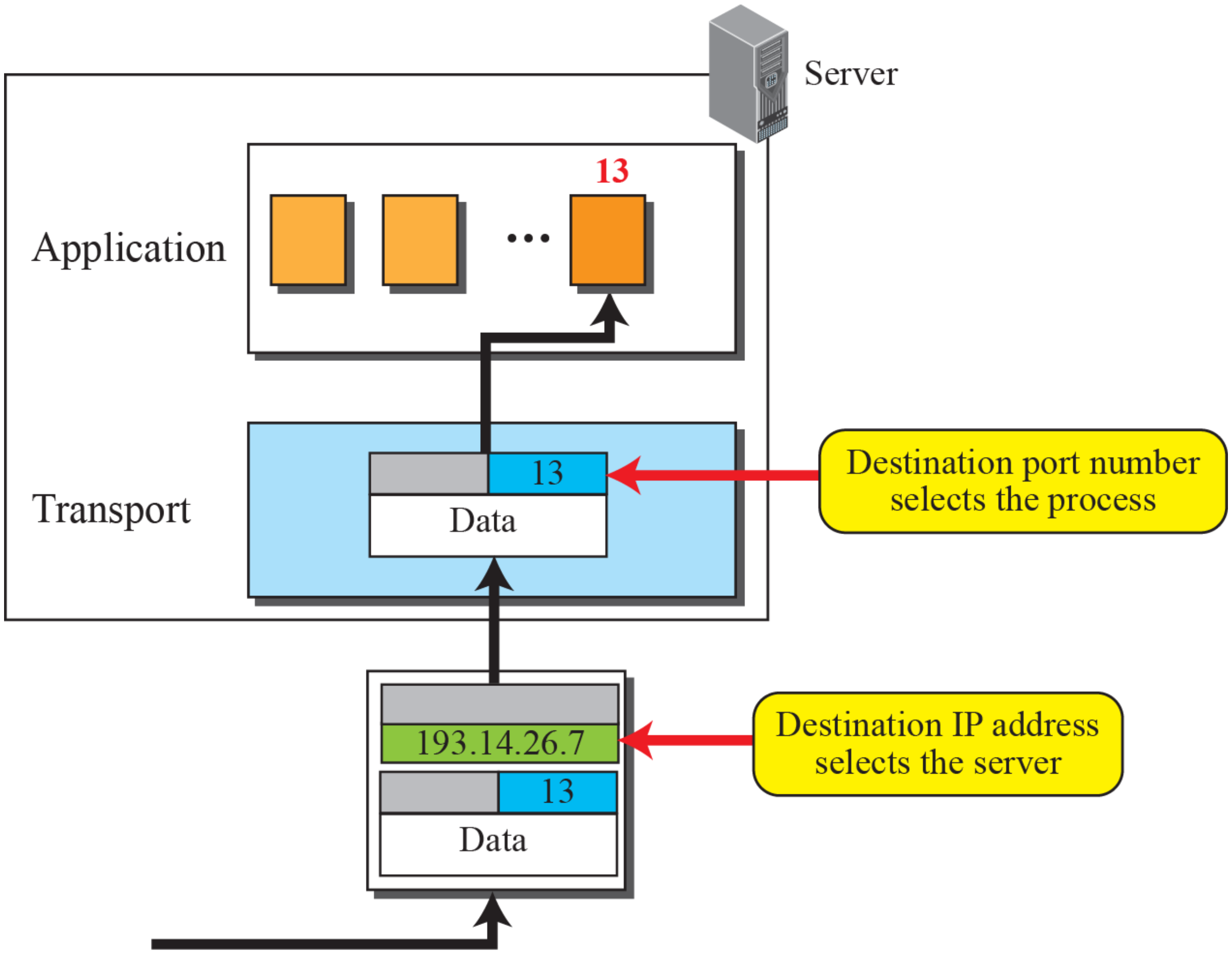

Server
Application
13
Transport
13
Data
Destination port number selects the process
193.14.26.7
13
Data
Destination IP address selects the server

Figure 23.5: ICANN ranges

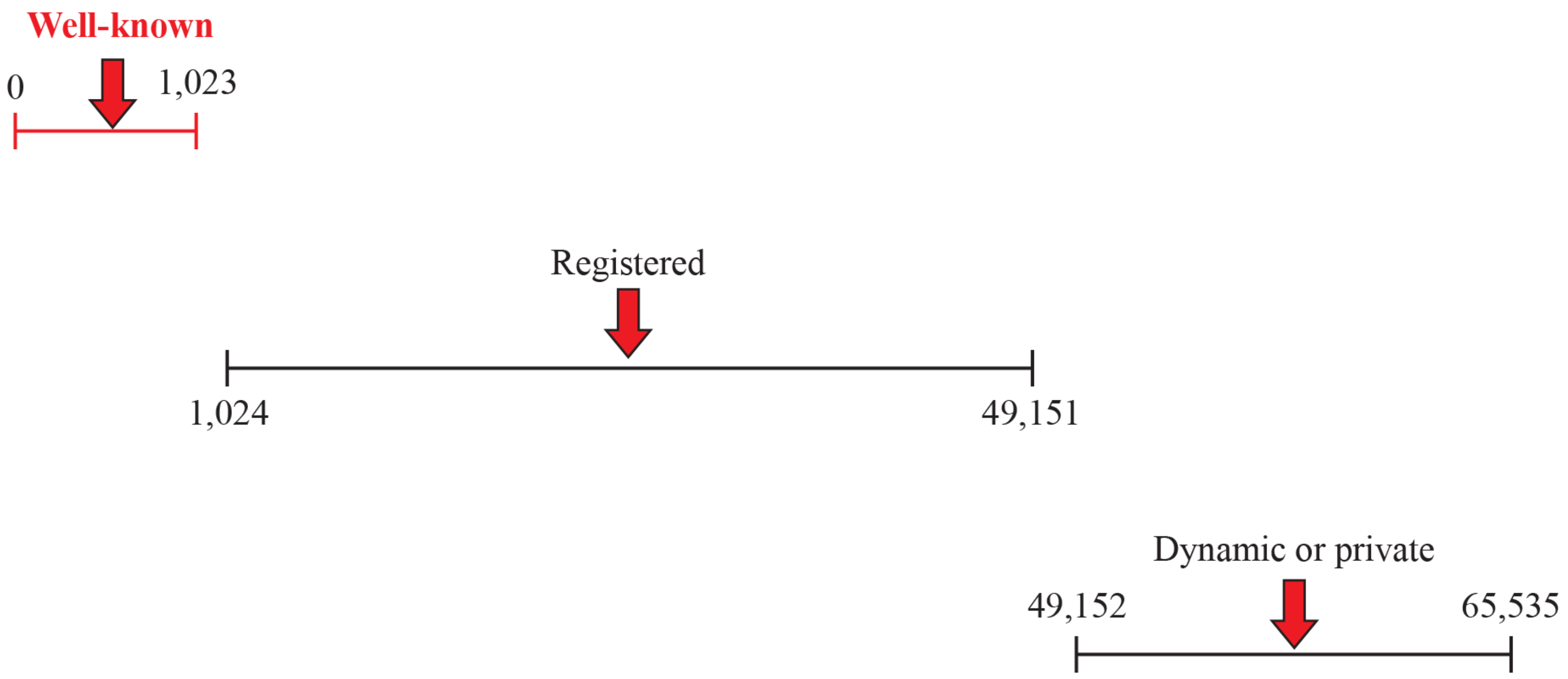


Figure 23.6: Socket address

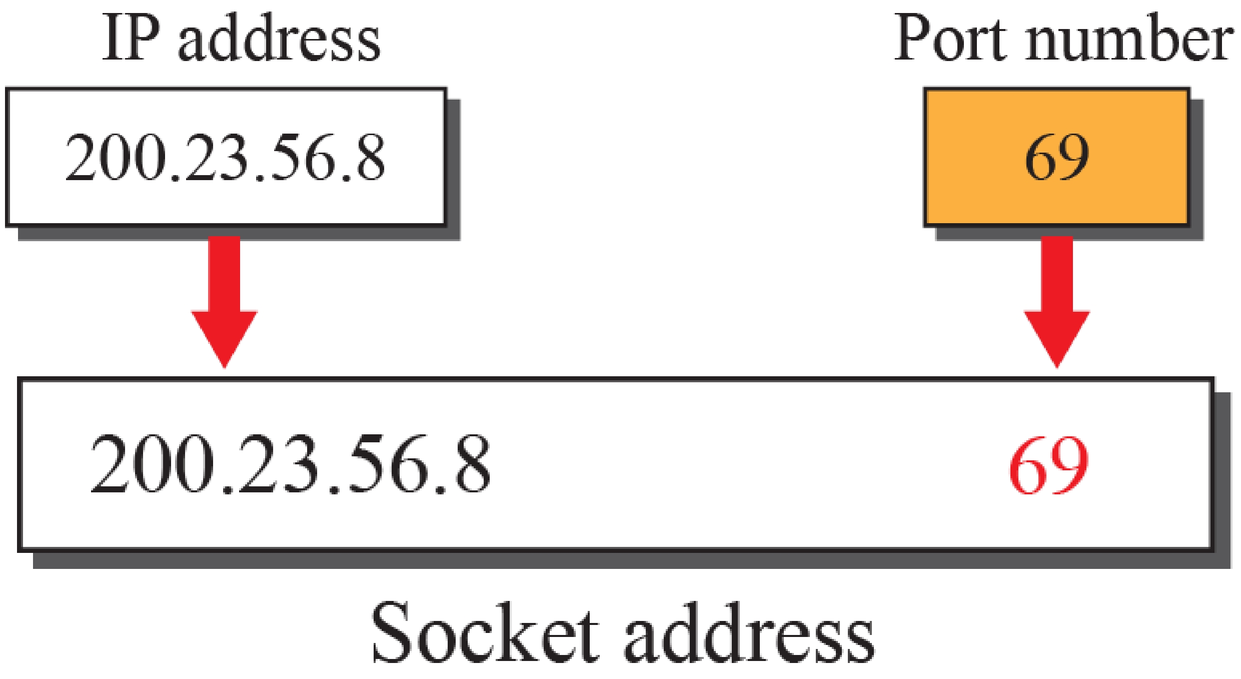


Figure 23.7: Encapsulation and decapsulation

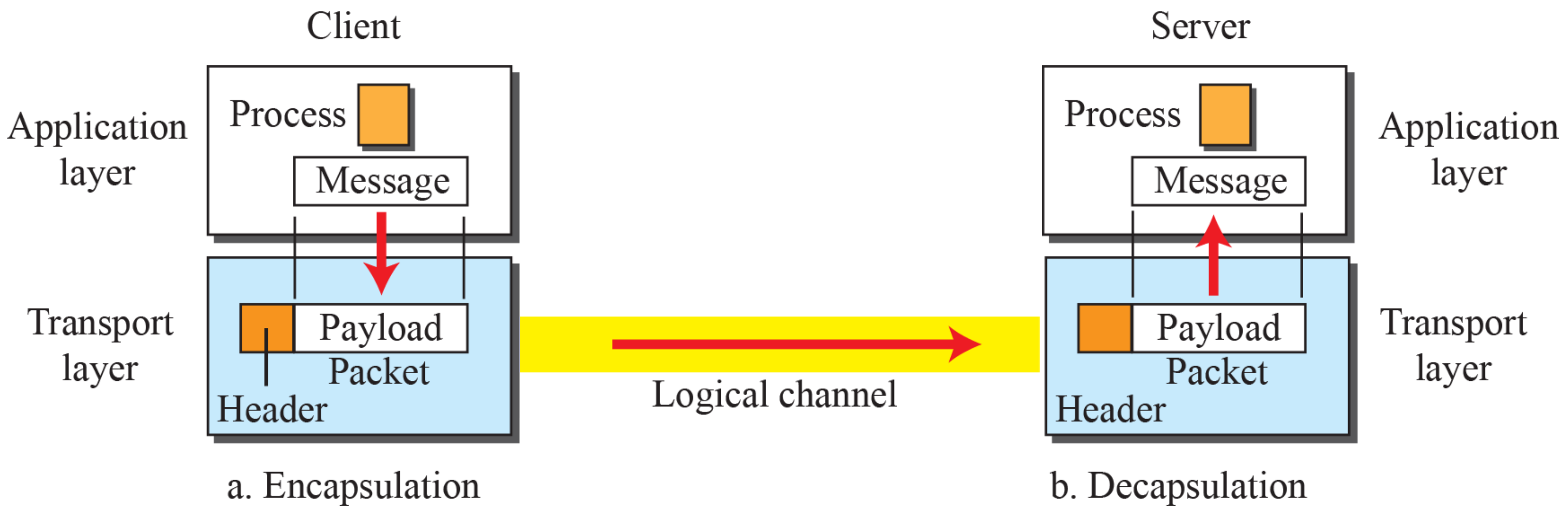


Figure 23.8: Multiplexing and demultiplexing

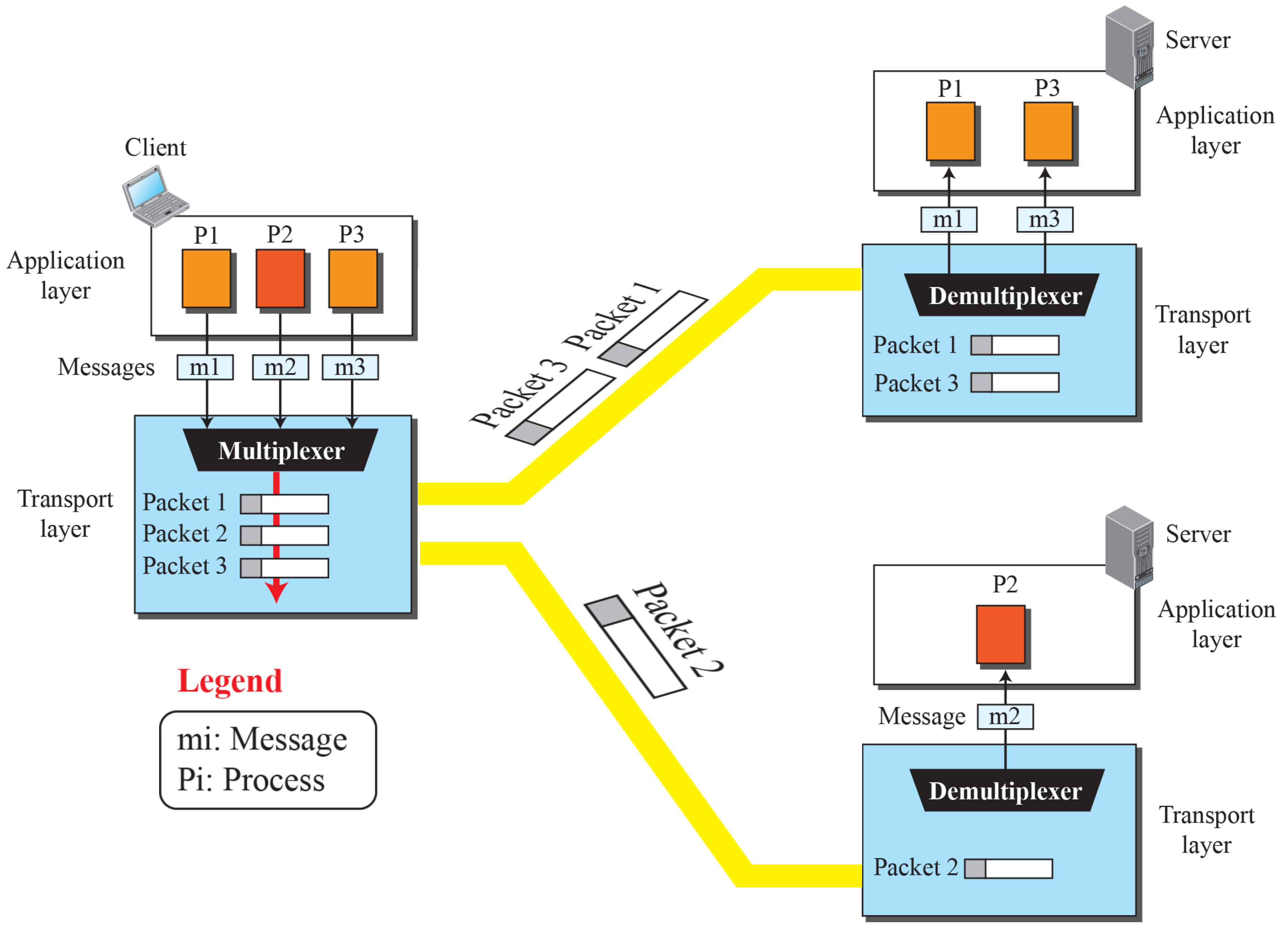


Figure 23.9: Pushing or pulling

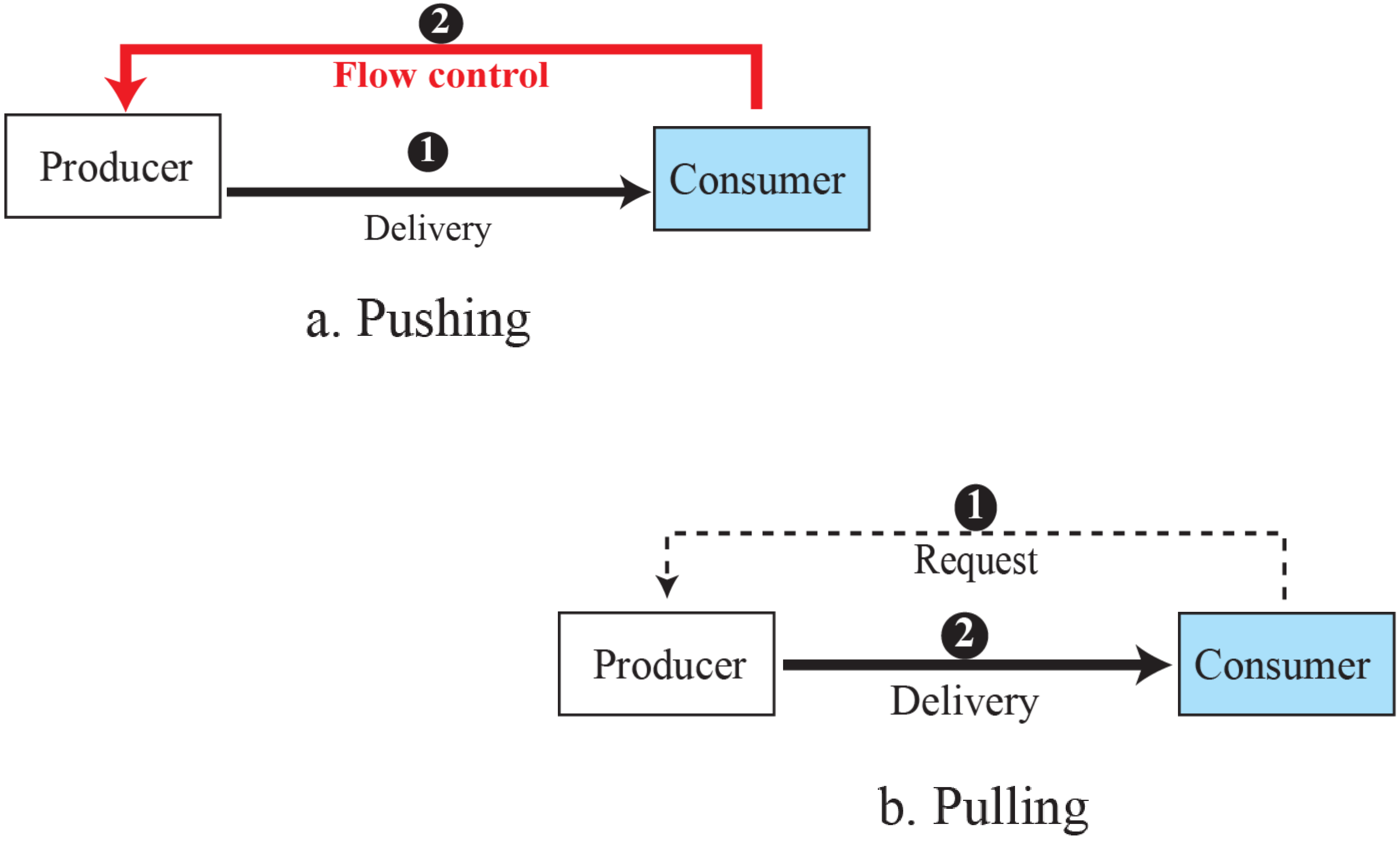


Figure 23.10: Flow control at the transport layer

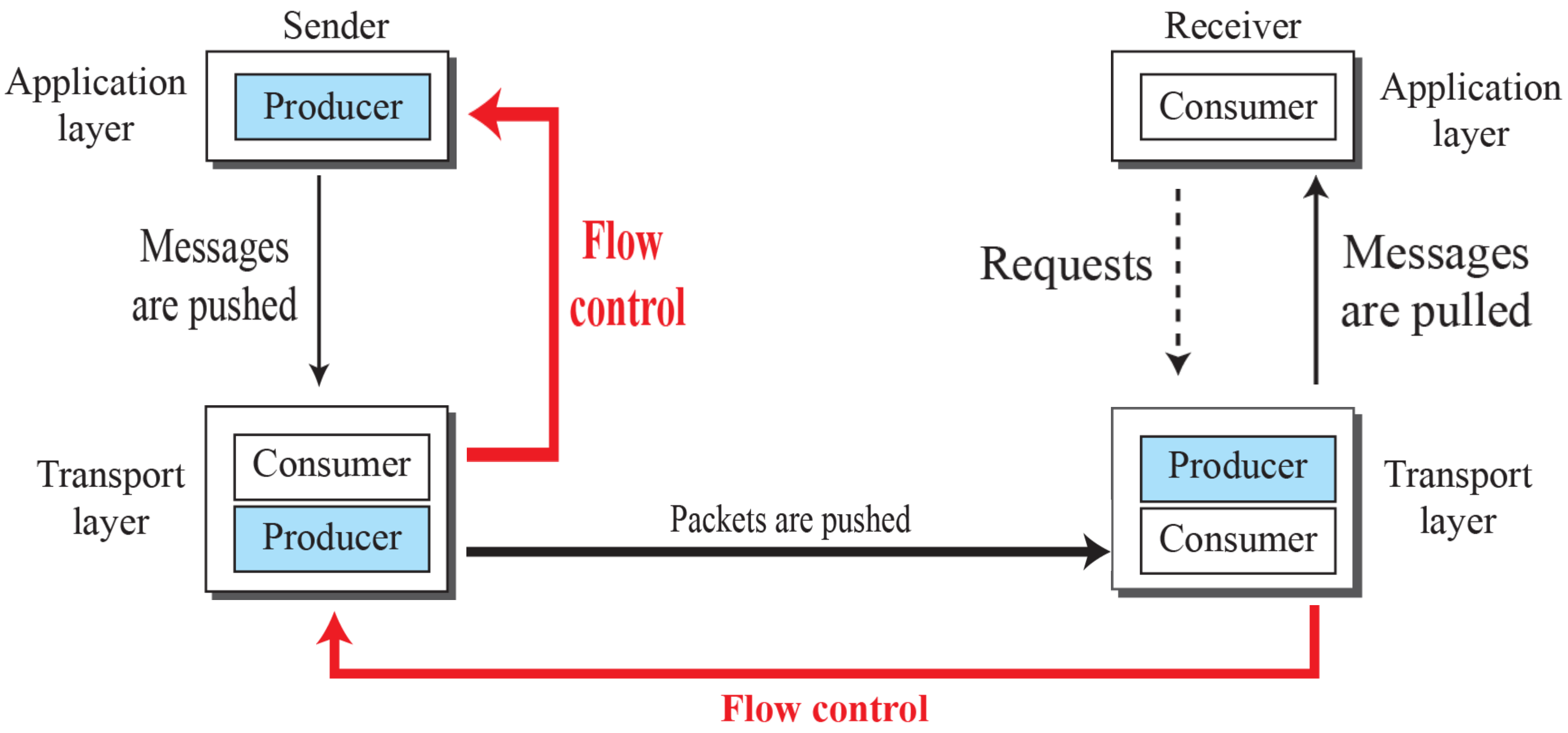


Figure 23.11: Error control at the transport layer

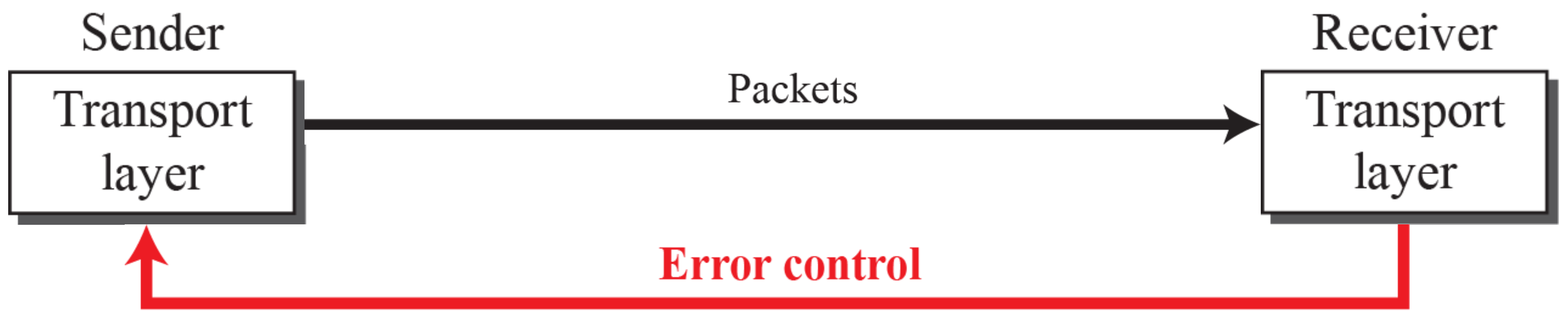
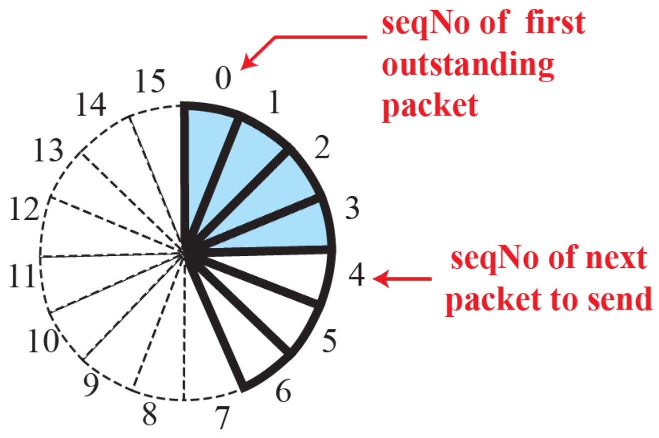
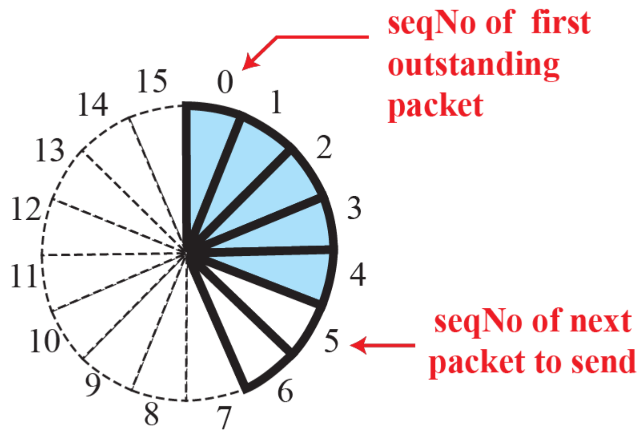


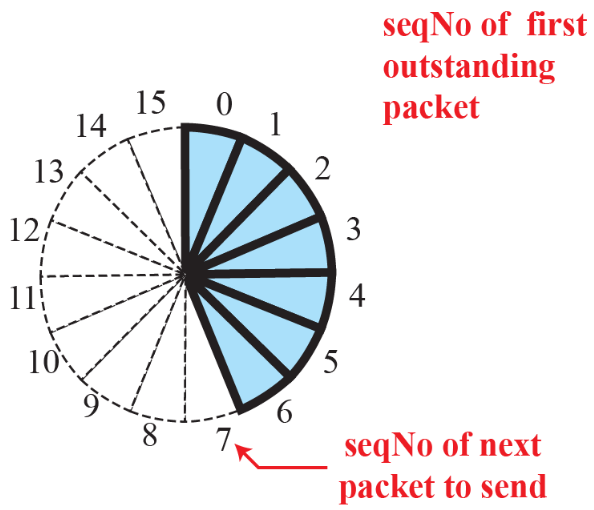
Figure 23.12: Sliding window in circular format


seqNo of first outstanding packet
seqNo of next packet to send
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15

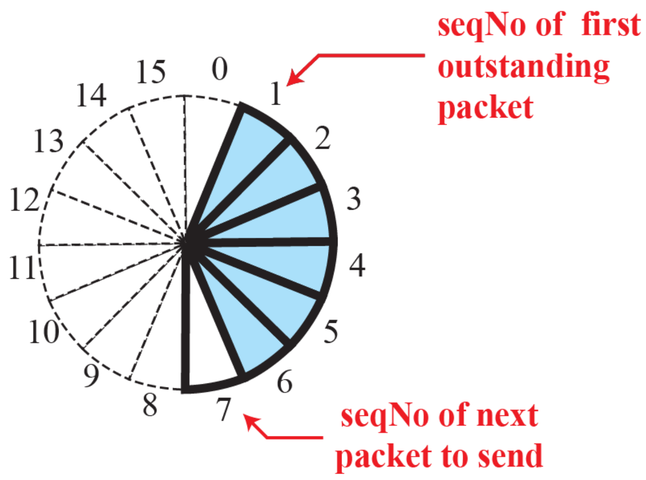
a. Four packets have been sent.


seqNo of first outstanding packet
seqNo of next packet to send
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15

b. Five packets have been sent.

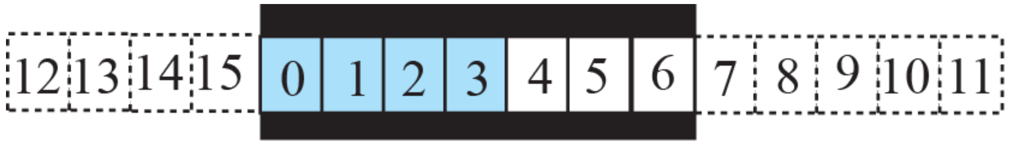

seqNo of first outstanding packet
seqNo of next packet to send
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15

c. Seven packets have been sent; window is full.

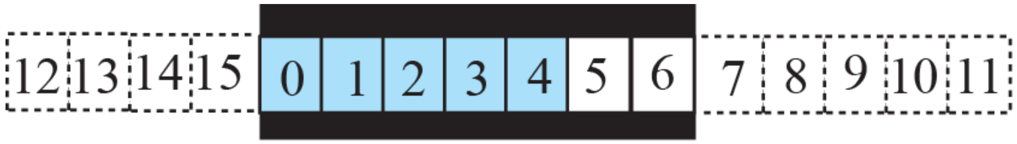

seqNo of first outstanding packet
seqNo of next packet to send
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15

d. Packet 0 has been acknowledged; window slides.

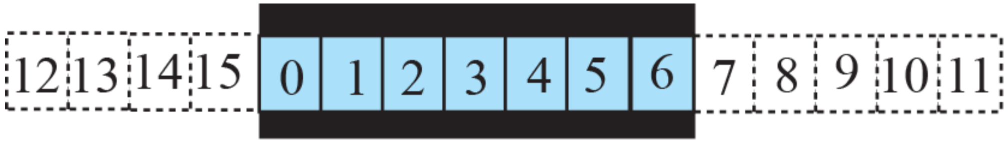
Figure 23.13: Sliding window in linear format



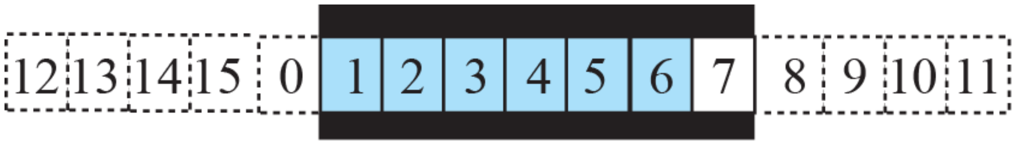
a. Four packets have been sent.



b. Five packets have been sent.



c. Seven packets have been sent;
window is full.



d. Packet 0 has been acknowledged;
window slides.

23.23.2 Connection

A transport-layer protocol, like a network-layer protocol, can provide two types of services: connectionless and connection-oriented. The nature of these services at the transport layer, however, is different from the ones at the network layer. At the network layer, a connectionless service may mean different paths for different datagrams belonging to the same message. Connectionless service at the transport layer means independency between packets; connection-oriented means dependency. Let us elaborate on these two services.

Figure 3.14: Connectionless service

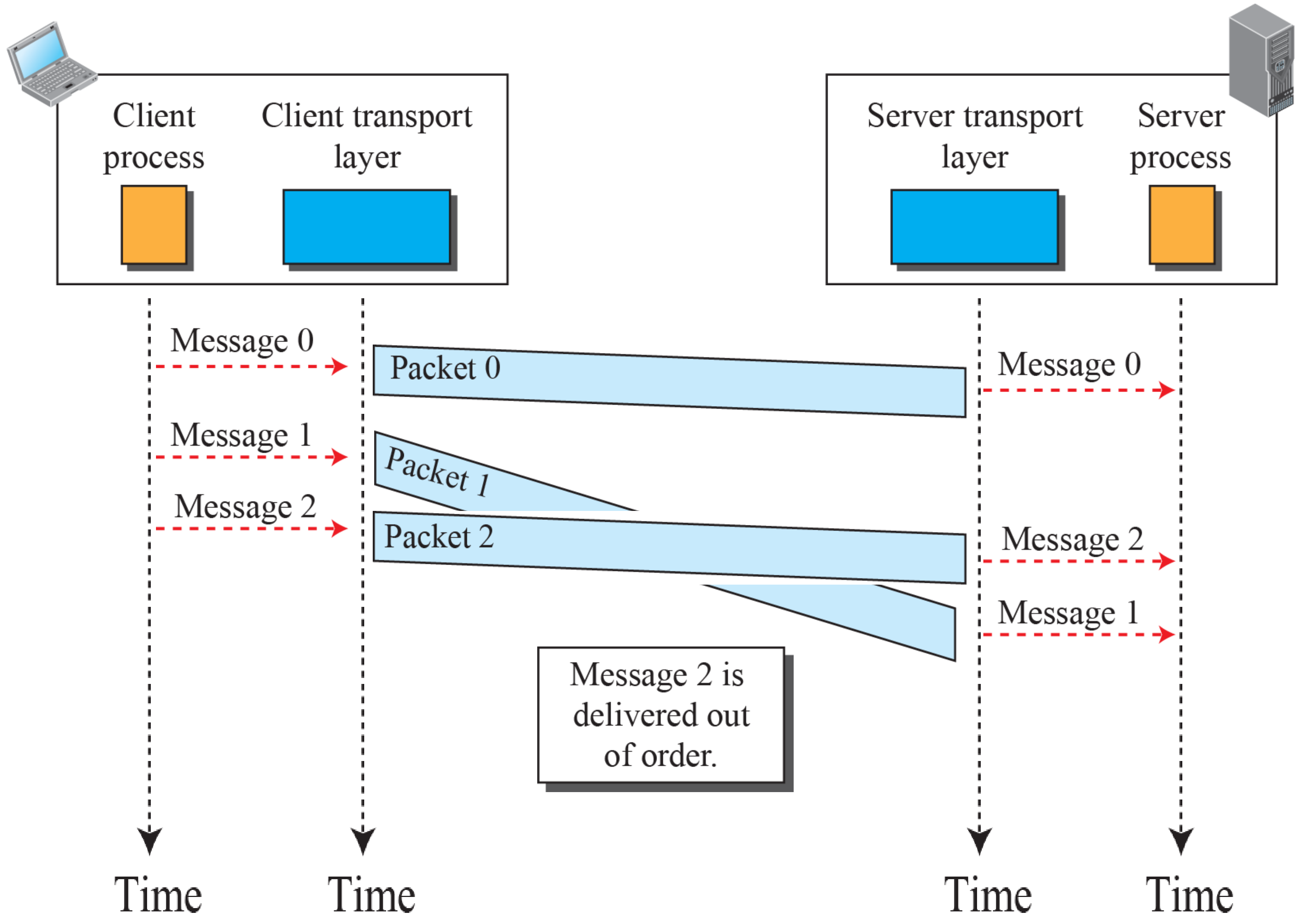


Figure 23.15: Connection-oriented service

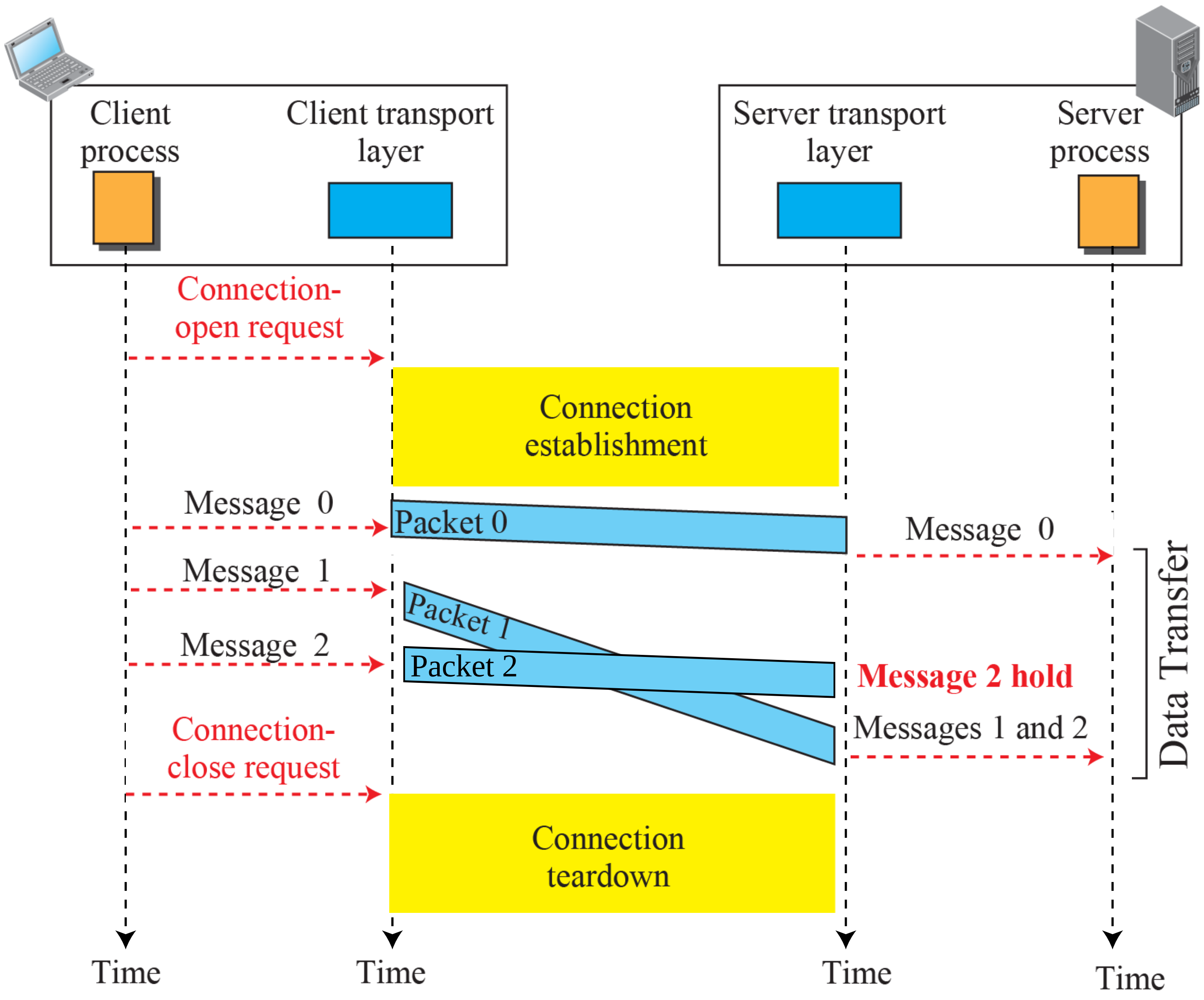
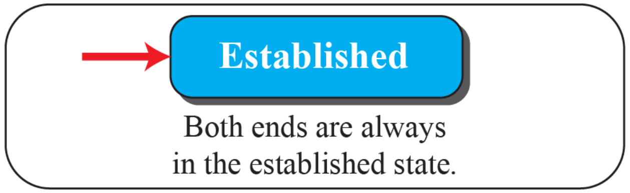


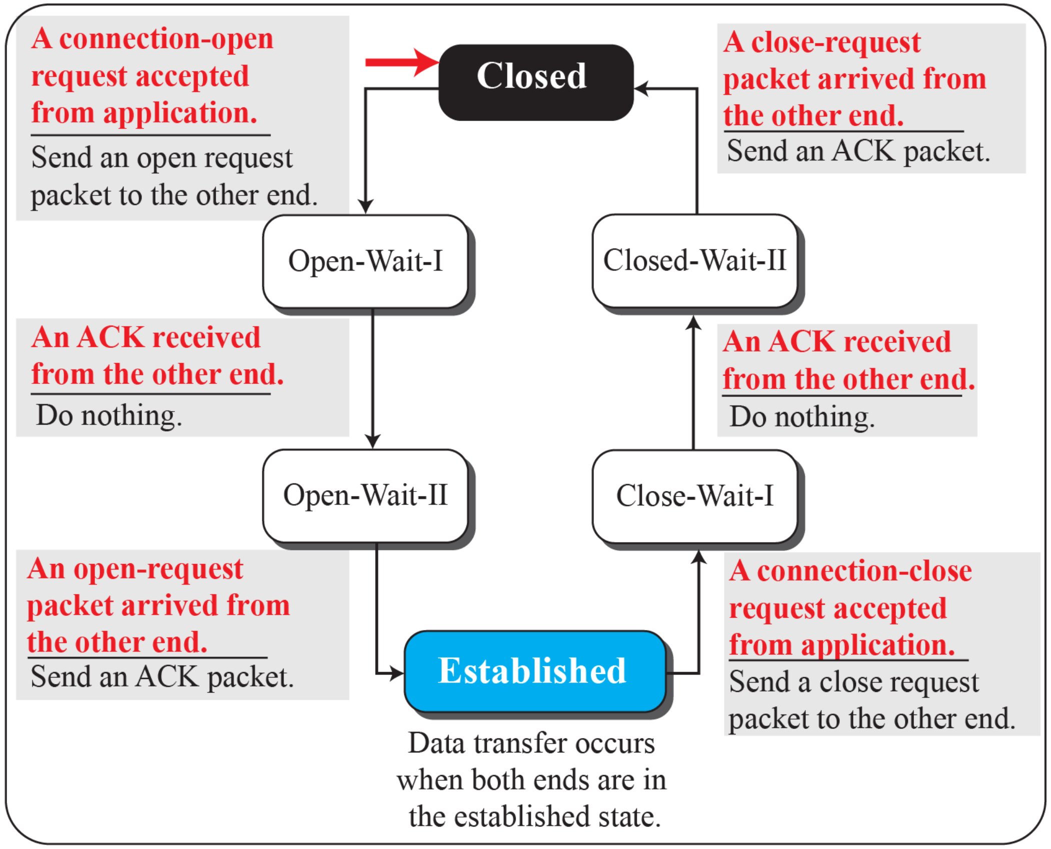
Figure 23.16: Connectionless and connection-oriented service represented as FSMs

FSM for connectionless transport layer



Note:
The colored arrow shows the starting state.

FSM for connection-oriented transport layer



A connection-open request accepted from application.
Send an open request packet to the other end.
Closed
A close-request packet arrived from the other end.
Send an ACK packet.
Open-Wait-I
Closed-Wait-II
An ACK received from the other end.
Do nothing.
An ACK received from the other end.
Do nothing.
Open-Wait-II
Close-Wait-I
An open-request packet arrived from the other end.
Send an ACK packet.
Established
A connection-close request accepted from application.
Send a close request packet to the other end.
Data transfer occurs when both ends are in the established state.

23-2 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.

23.2.1 Simple Protocol

Our first protocol is a simple connectionless protocol with neither flow nor error control. We assume that the receiver can immediately handle any packet it receives. In other words, the receiver can never be overwhelmed with incoming packets. Figure 23.17 shows the layout for this protocol.

Figure 23.17: Simple protocol

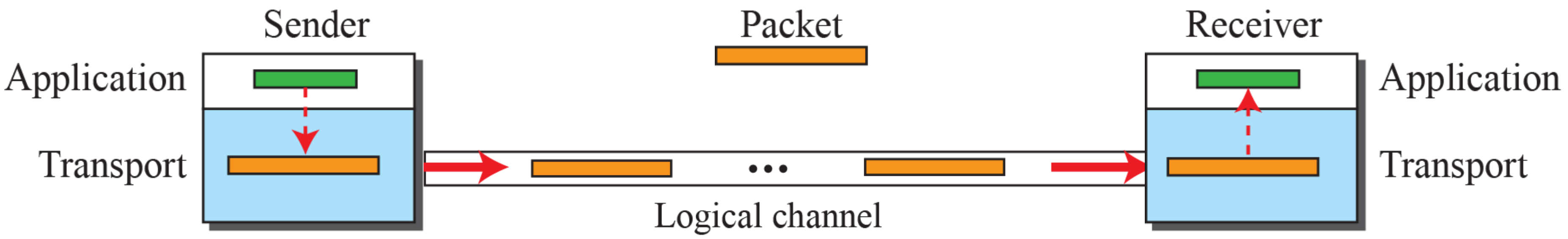
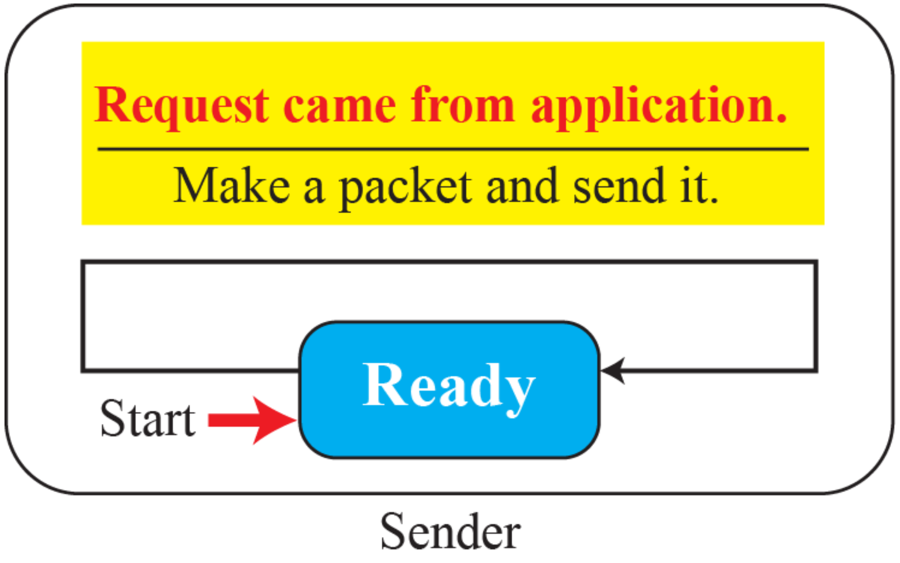
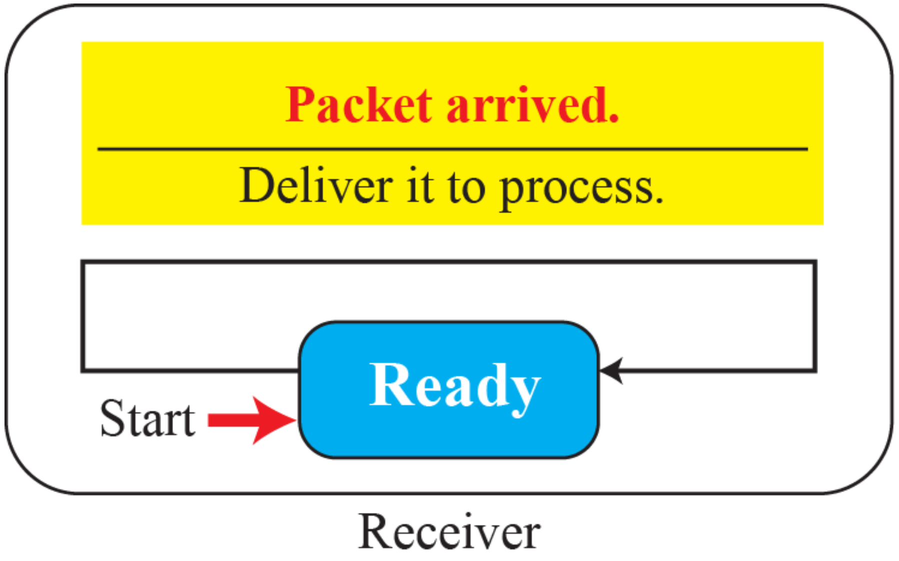


Figure 23.18: FSMs for the simple protocol

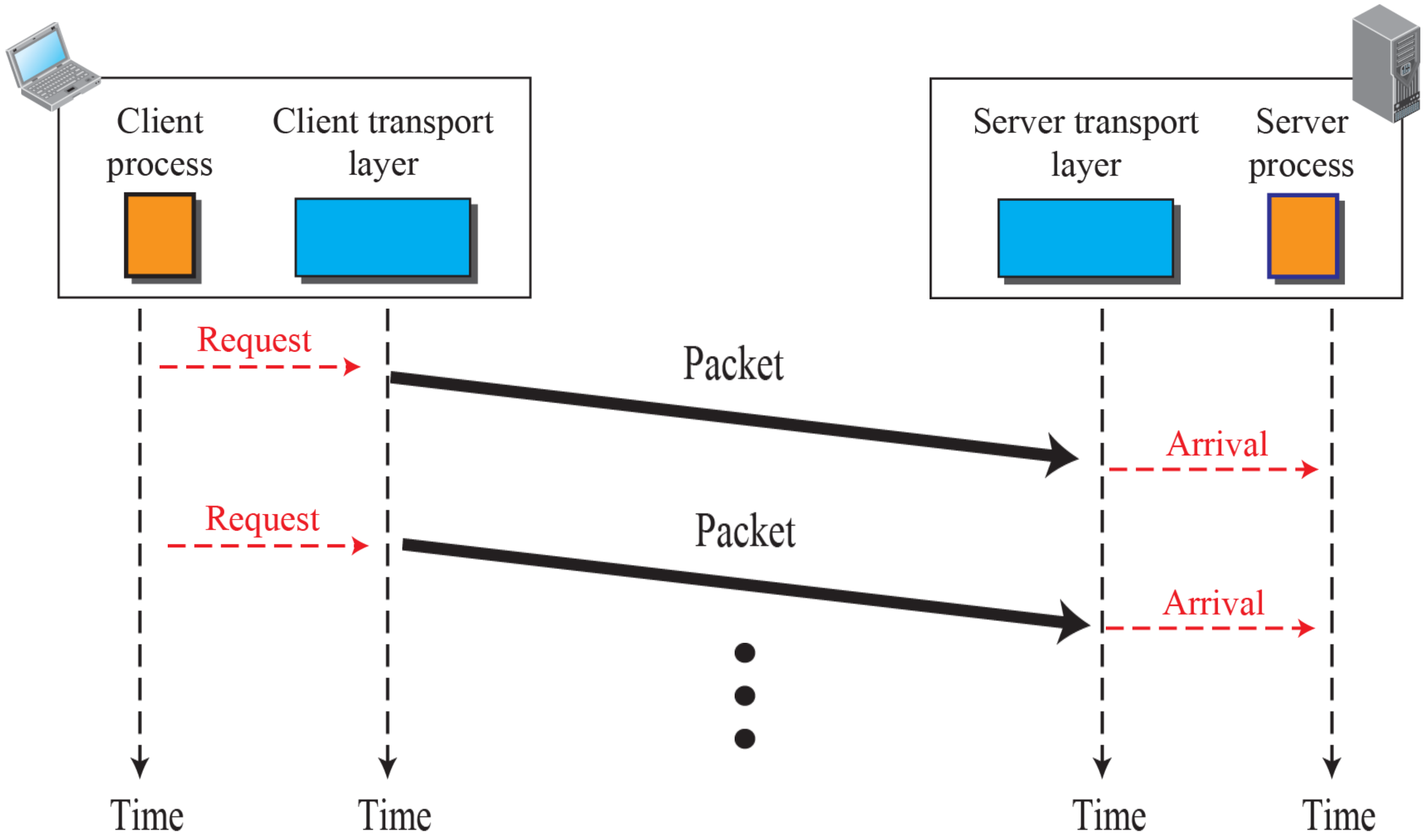




Example 23.3

Figure 23.19 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 23.19: Flow diagram for Example 3.3



23.2.2 Stop-and-Wait Protocol

Our second protocol is a connection-oriented protocol called the Stop-and-Wait protocol, which uses both flow and error control. Both the sender and the receiver use a sliding window of size 23. The sender sends one packet at a time and waits for an acknowledgment before sending the next one. To detect corrupted packets, we need to add a checksum to each data packet. When a packet arrives at the receiver site, it is checked. If its checksum is incorrect, the packet is corrupted and silently discarded.

Figure 23.20: Stop-and-Wait protocol

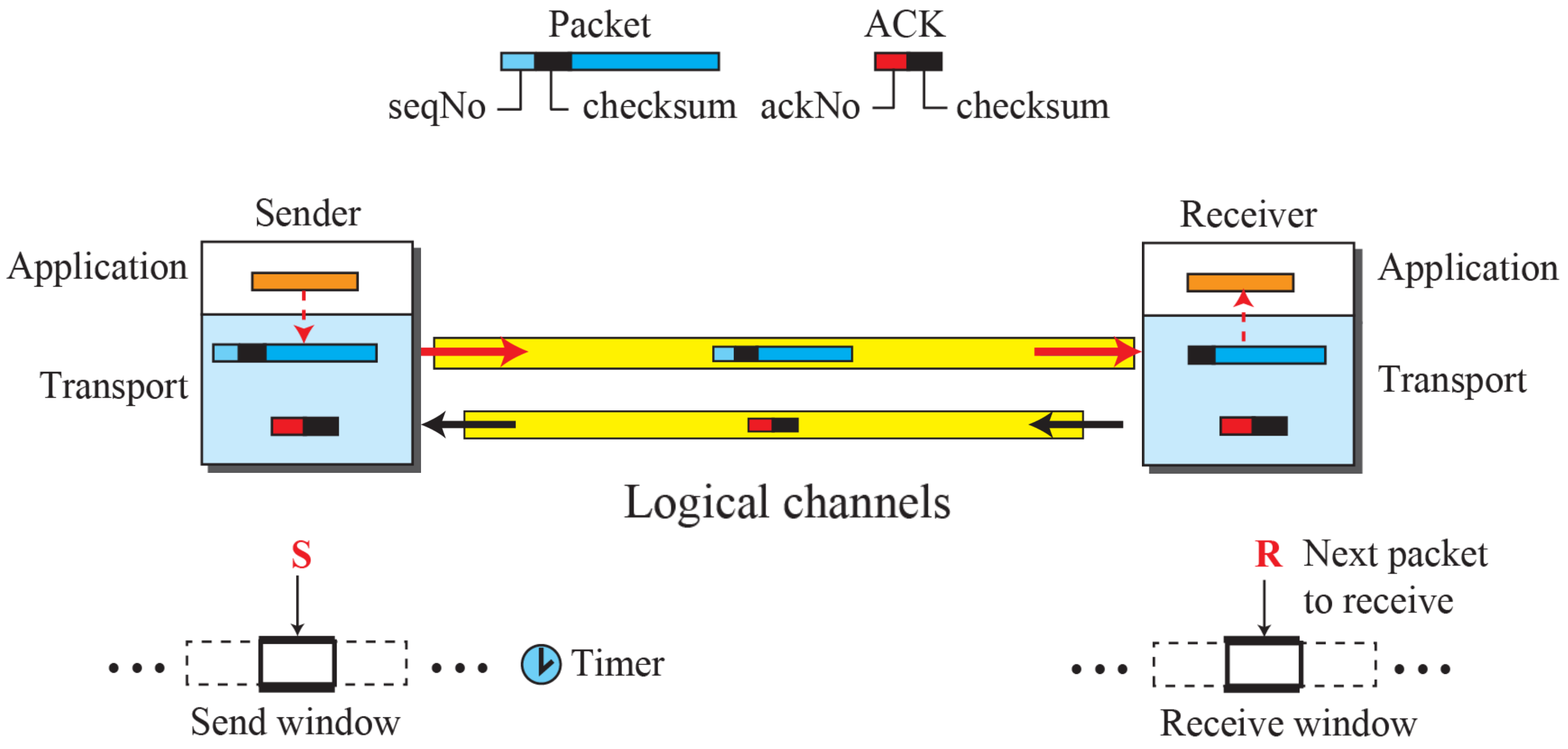
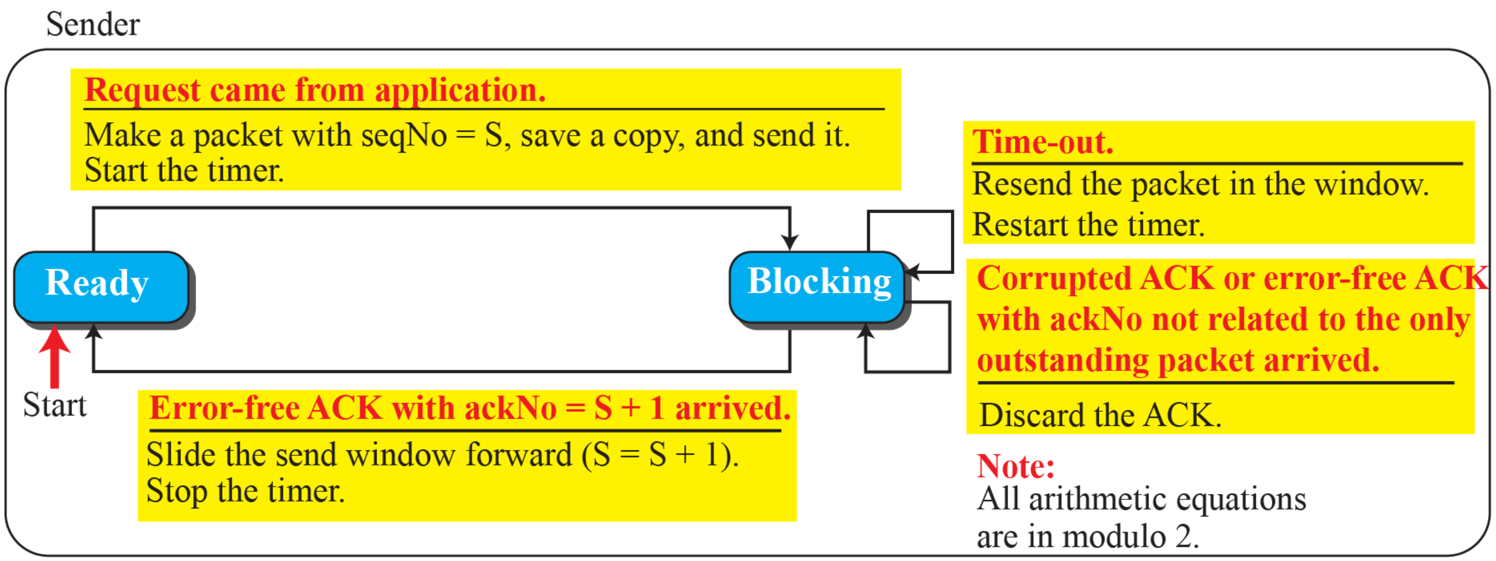
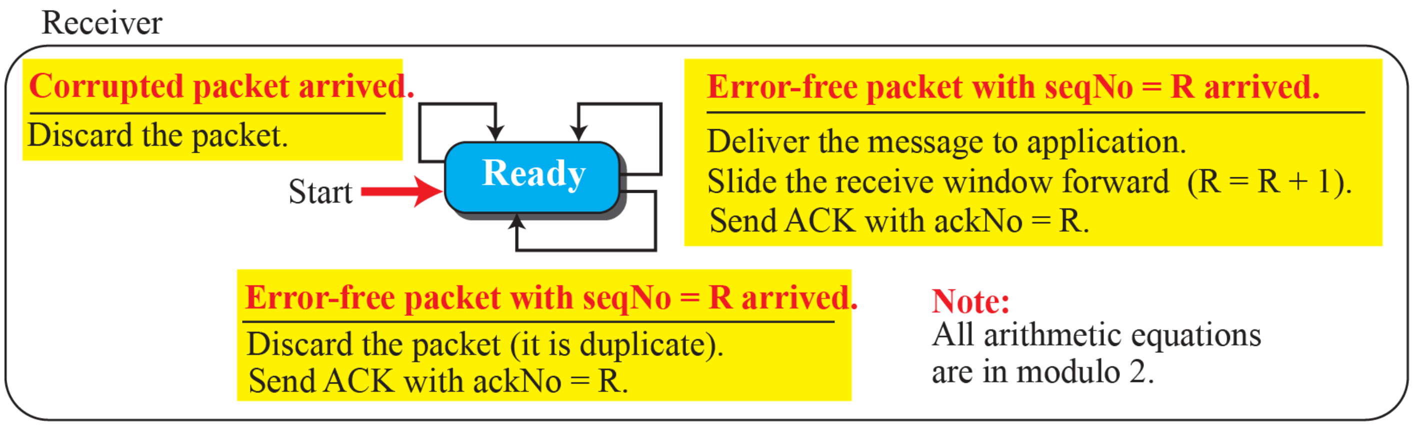


Figure 23.21: FSM for the Stop-and-Wait protocol



Sender
Request came from application.
Make a packet with seqNo = S, save a copy, and send it.
Start the timer.
Time-out.
Resend the packet in the window.
Restart the timer.
Ready
Blocking
Corrupted ACK or error-free ACK with ackNo not related to the only outstanding packet arrived.
Discard the ACK.
Start
Error-free ACK with ackNo = S + 1 arrived.
Slide the send window forward (S = S + 1).
Stop the timer.
Note:
All arithmetic equations are in modulo 2.

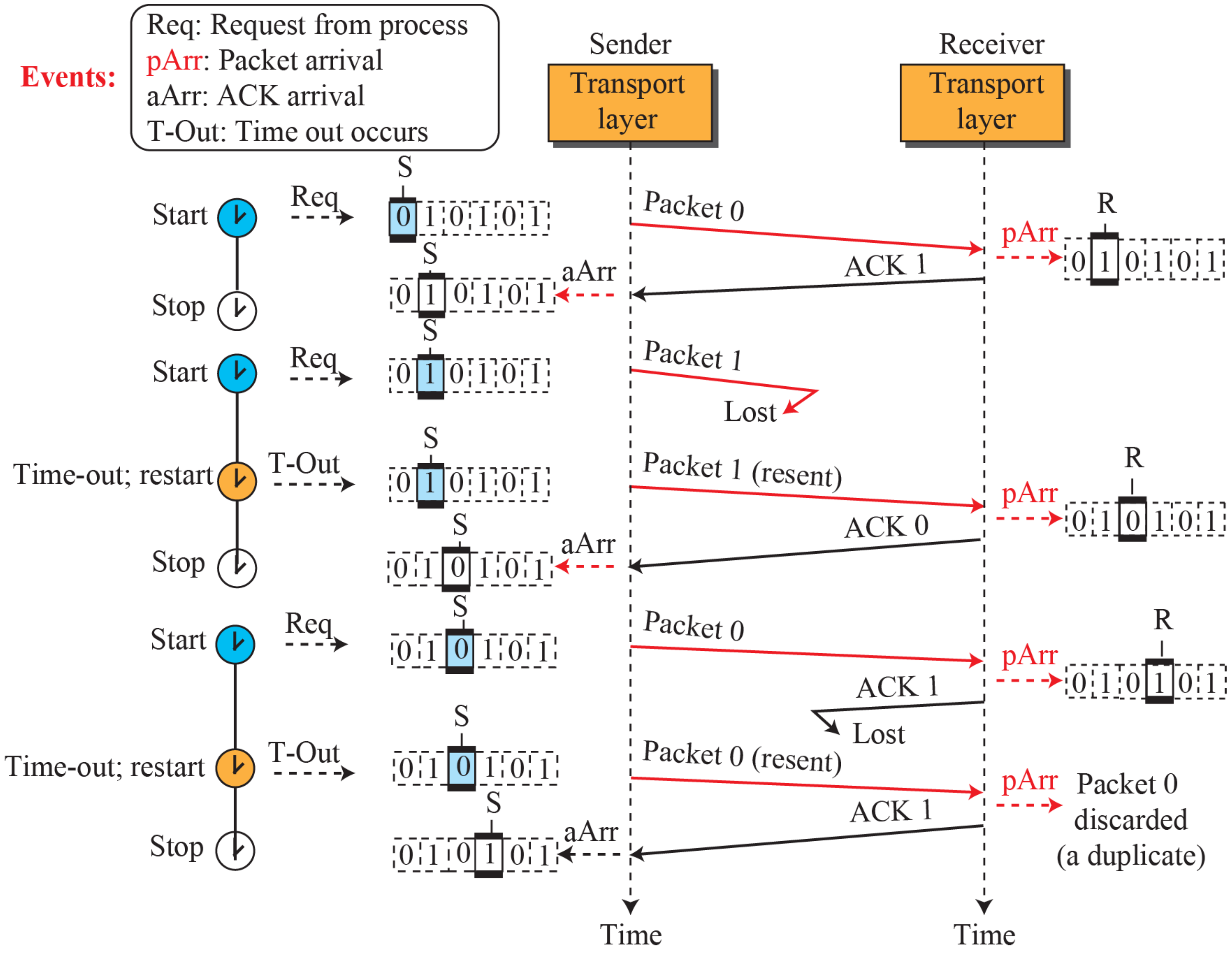


Receiver
Corrupted packet arrived.
Discard the packet.
Error-free packet with seqNo = R arrived.
Deliver the message to application.
Slide the receive window forward (R = R + 1).
Send ACK with ackNo = R.
Start
Ready
Error-free packet with seqNo ≠ R arrived.
Discard the packet (it is duplicate).
Send ACK with ackNo = R.
Note:
All arithmetic equations are in modulo 2.

Example 23.4

Figure 23.22 shows an example of the 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 23.22: Flow diagram for Example 3.4



Example 23.5

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. The link utilization is only 1,000/20,000, or 5 percent.

Example 23.6

What is the utilization percentage of the link in Example 23.5 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.

23.2.3 Go-Back-N Protocol (GBN)

To improve the efficiency of transmission (to fill the pipe), multiple packets must be in transition while the sender is waiting for acknowledgment. In other words, we need to let more than one packet be outstanding to keep the channel busy while the sender is waiting for acknowledgment. In this section, we discuss one protocol that can achieve this goal; in the next section, we discuss a second. The first is called Go-Back-N (GBN).

Figure 23.23: Go-Back-N protocol

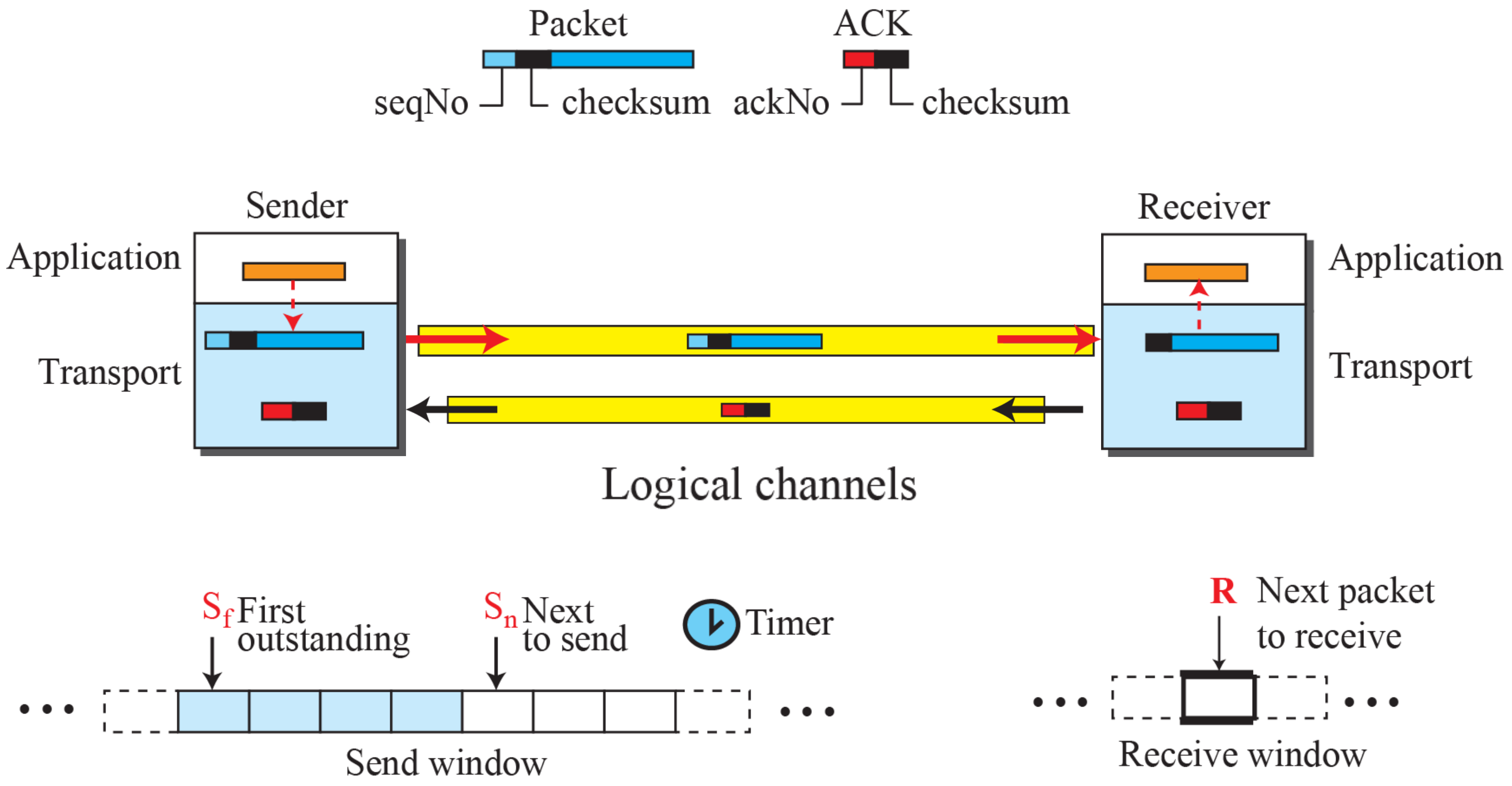


Figure 23.24: Send window for Go-Back-N

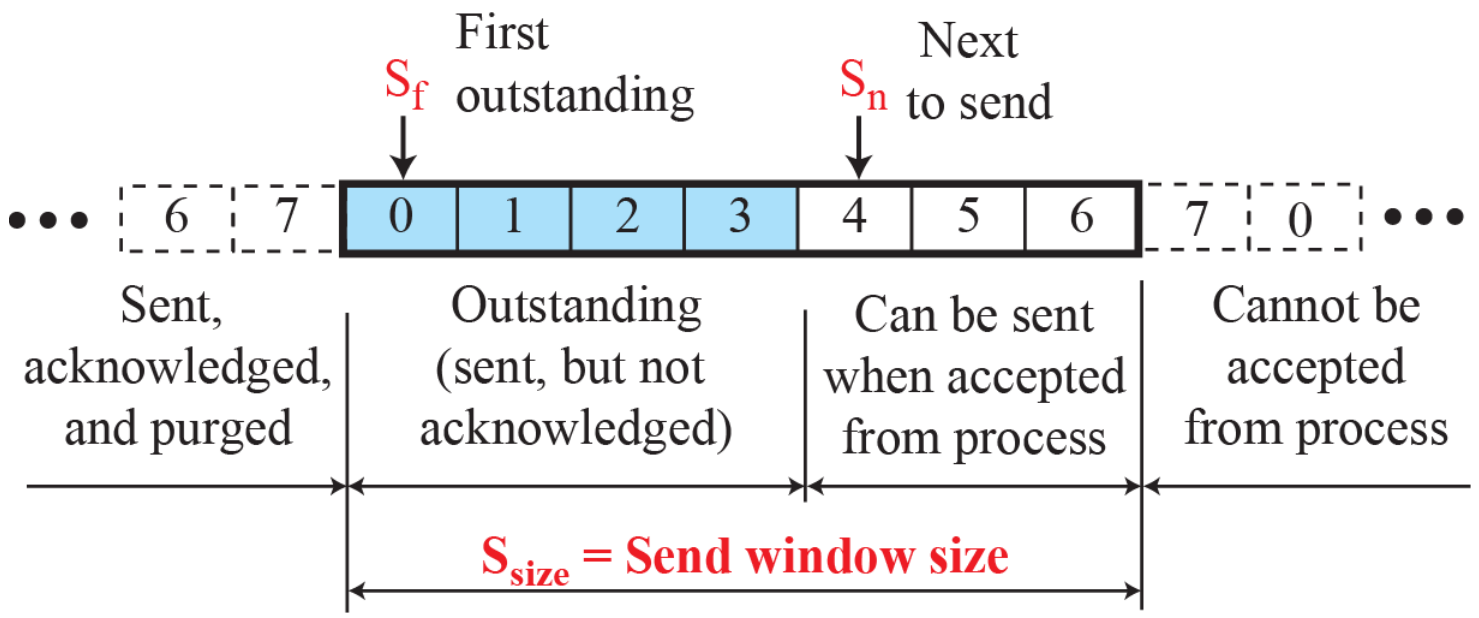
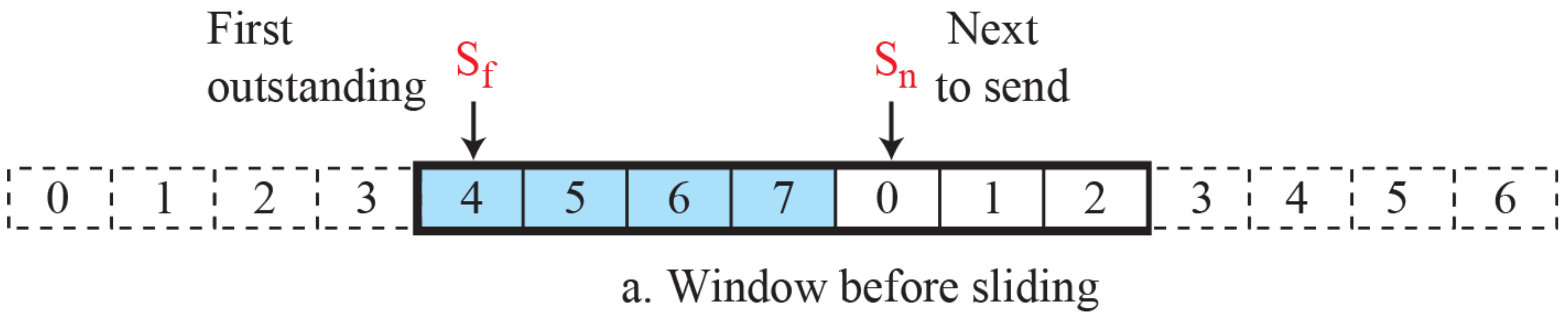
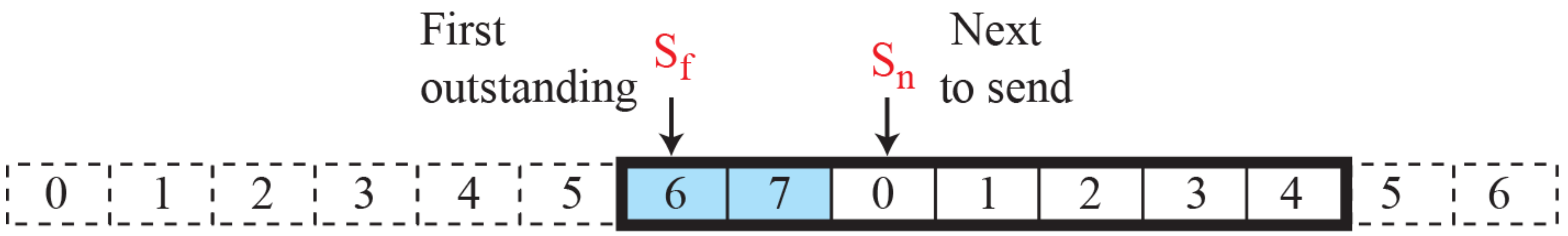

First outstanding
S_f
Next to send
S_n
6
7
0
1
2
3
4
5
6
7
0
Sent, acknowledged, and purged
Outstanding (sent, but not acknowledged)
Can be sent when accepted from process
Cannot be accepted from process
S_{size} = Send window size

Figure 23.25: Sliding the send window



a. Window before sliding

Sliding direction



b. Window after sliding (an ACK with ackNo = 6 has arrived)

Figure 23.26: Receive window for Go-Back-N

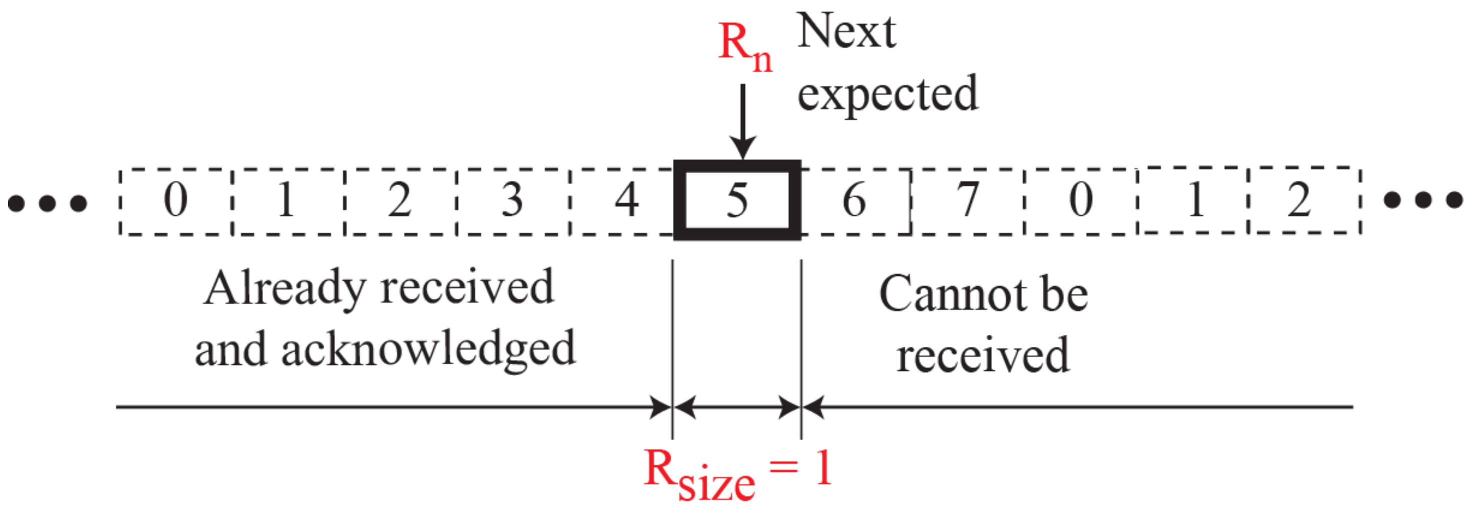
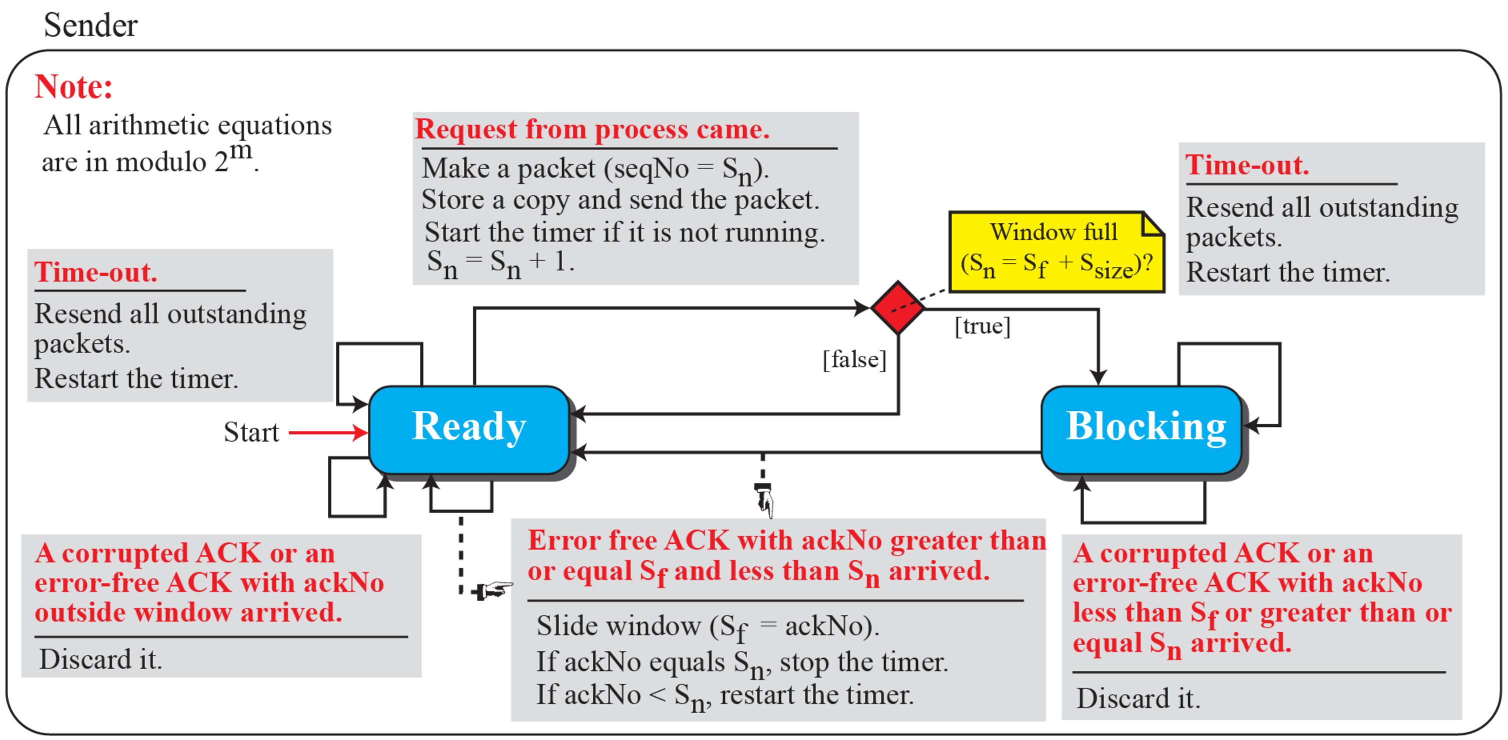
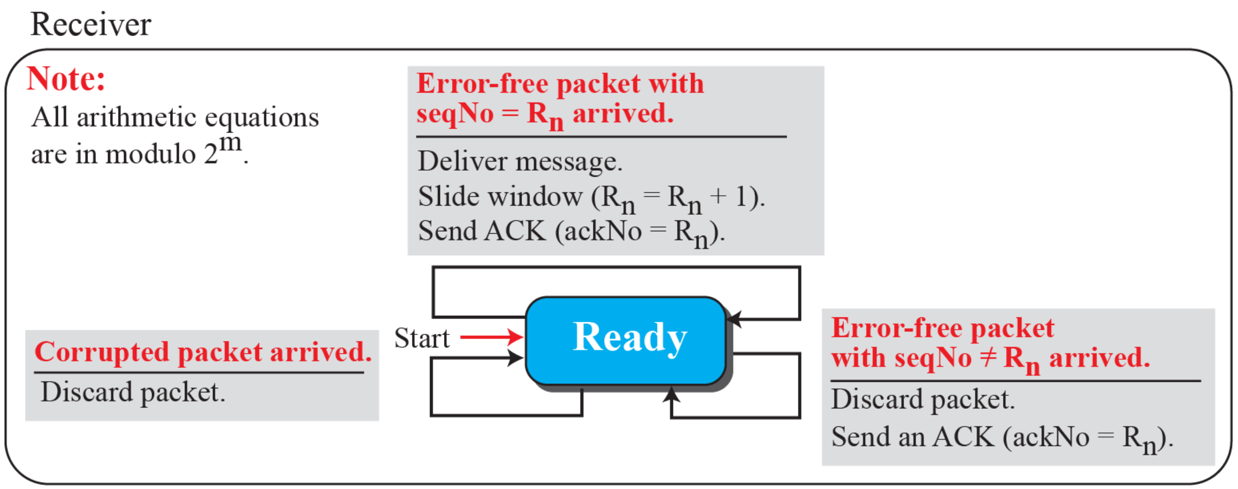


Figure 23.27: FSMs for the Go-Back-N protocol

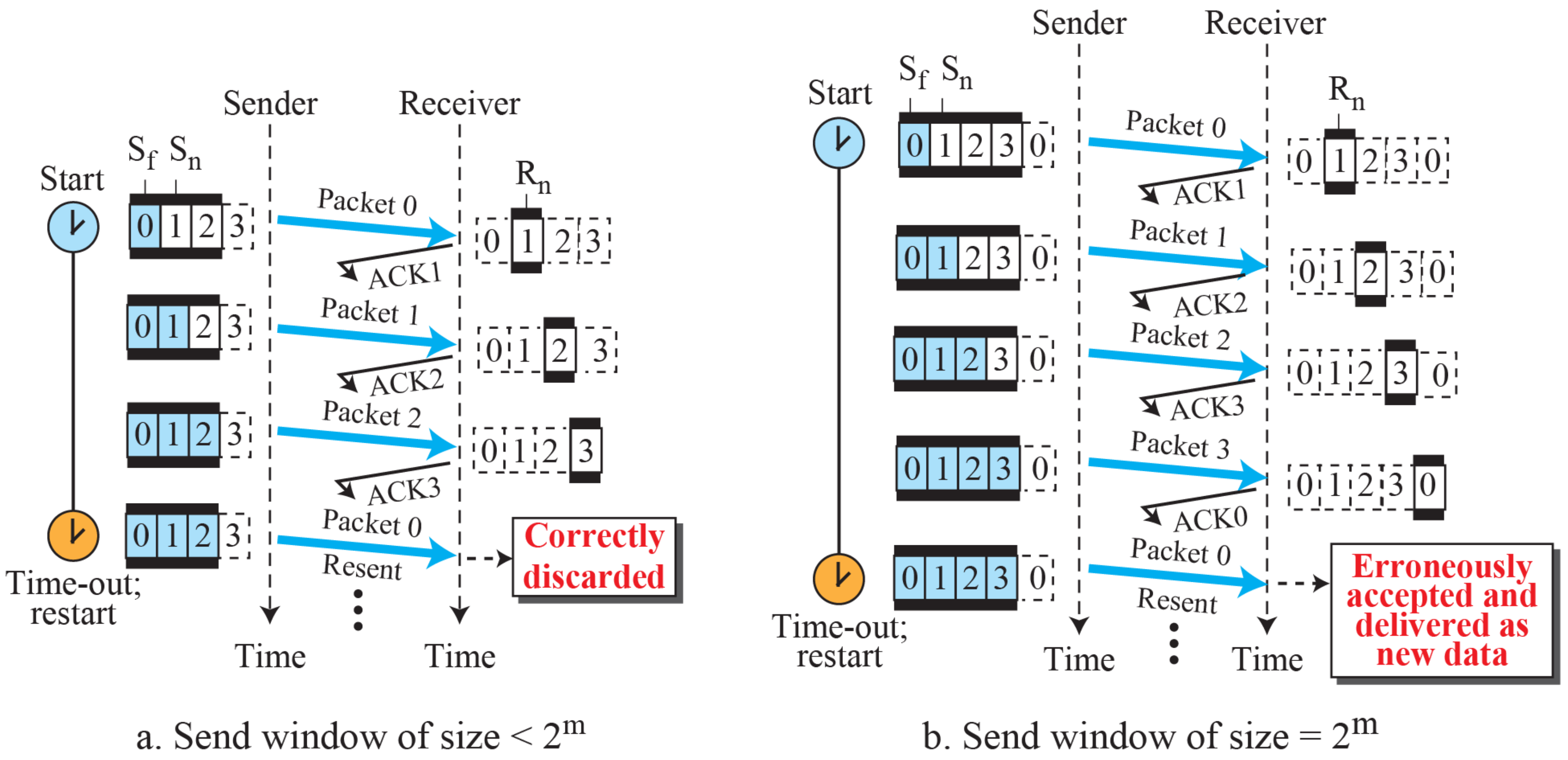


Sender
Note:
All arithmetic equations are in modulo 2^m.
Request from process came.
Make a packet (seqNo = S_n).
Store a copy and send the packet.
Start the timer if it is not running.
$S_n = S_n + 1$.
Window full ($S_n = S_f + S_{size}$)?
[true]
[false]
Time-out.
Resend all outstanding packets.
Restart the timer.
Time-out.
Resend all outstanding packets.
Restart the timer.
Start
Ready
Blocking
A corrupted ACK or an error-free ACK with ackNo outside window arrived.
Discard it.
Error free ACK with ackNo greater than or equal S_f and less than S_n arrived.
Slide window (S_f = ackNo).
If ackNo equals S_n, stop the timer.
If ackNo < S_n, restart the timer.
A corrupted ACK or an error-free ACK with ackNo less than S_f or greater than or equal S_n arrived.
Discard it.



Receiver
Note:
All arithmetic equations are in modulo 2^m.
Error-free packet with seqNo = R_n arrived.
Deliver message.
Slide window ($R_n = R_n + 1$).
Send ACK (ackNo = R_n).
Start
Ready
Corrupted packet arrived.
Discard packet.
Error-free packet with seqNo ≠ R_n arrived.
Discard packet.
Send an ACK (ackNo = R_n).

Figure 23.28: Send window size for Go-Back-N



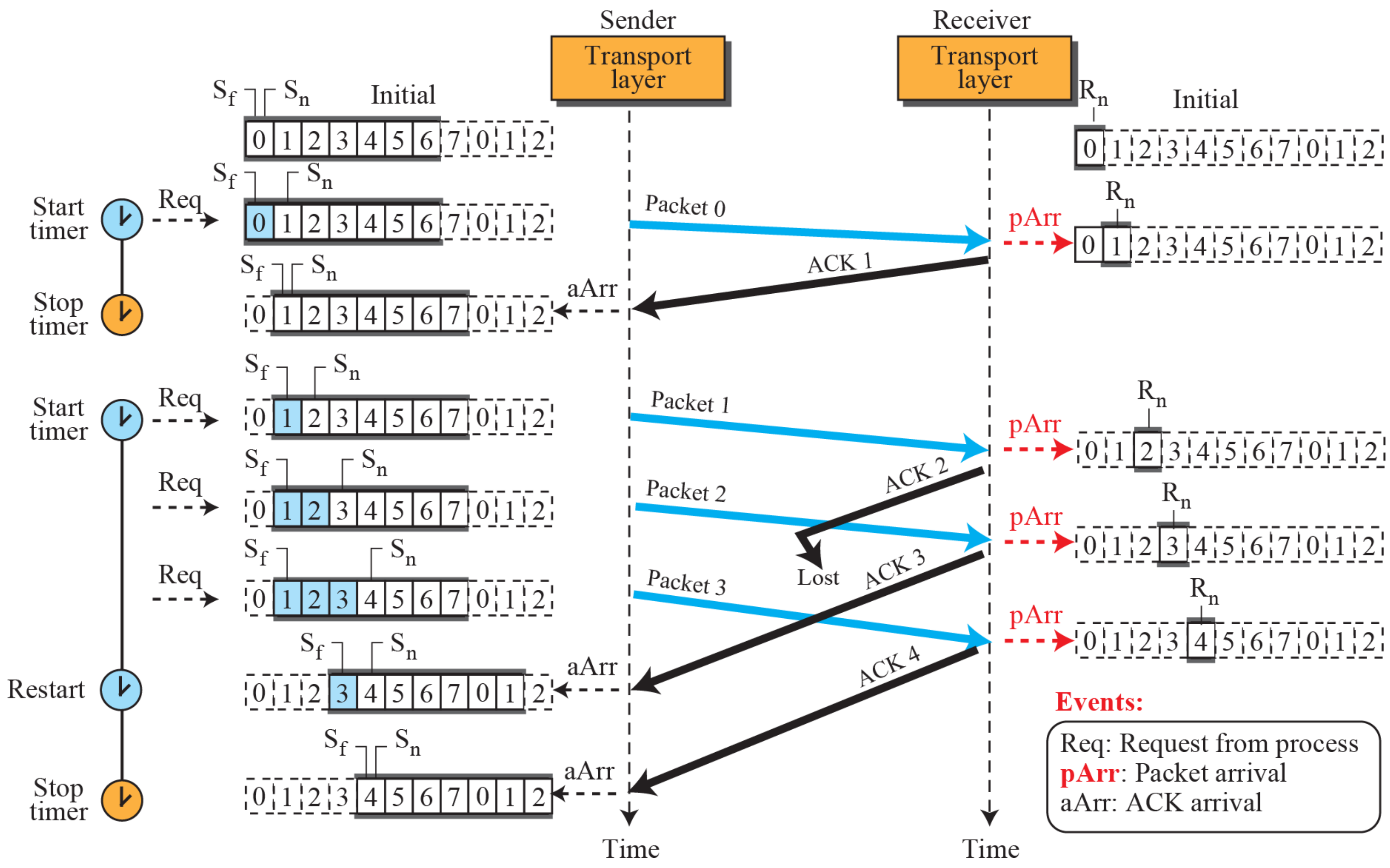
a. Send window of size < 2^m

b. Send window of size = 2^m

Example 23.7

Figure 23.29 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.

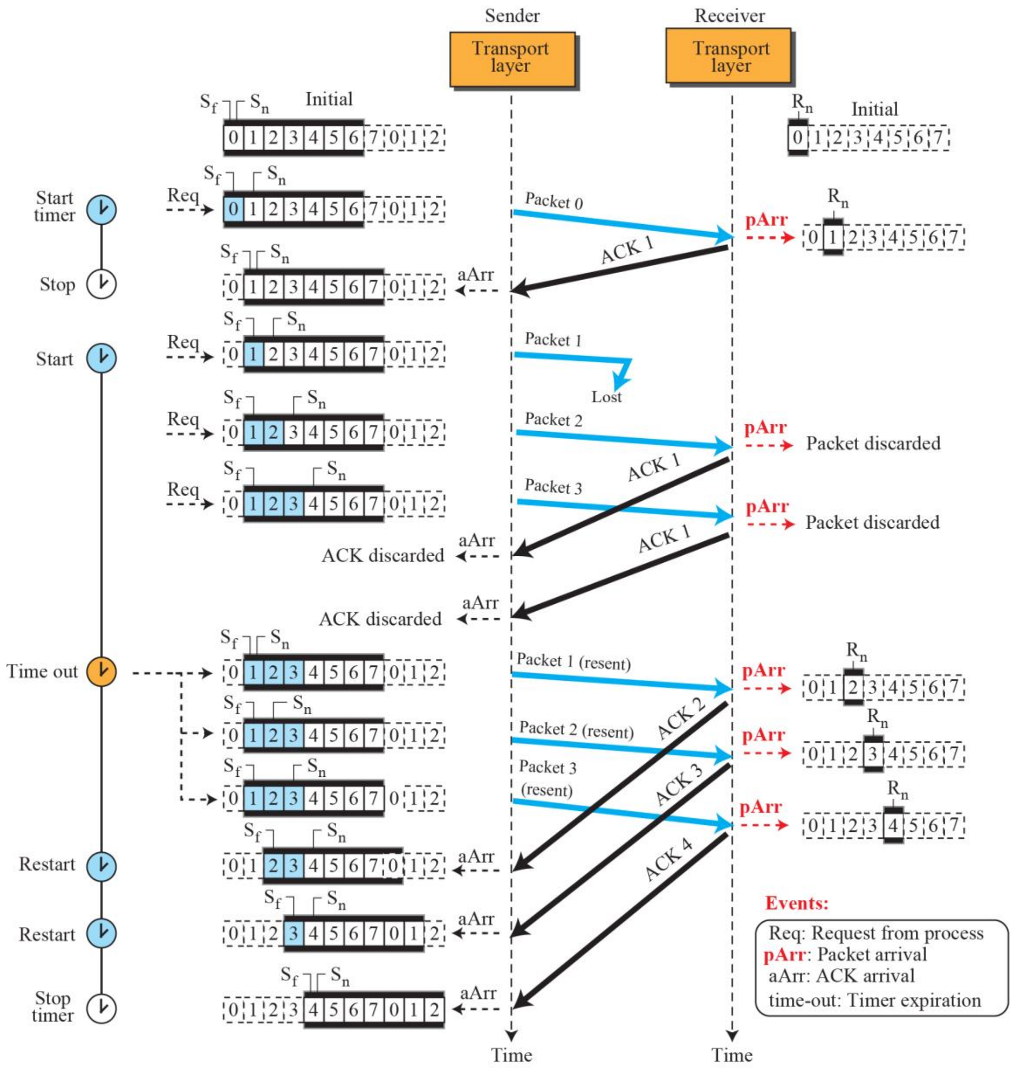
Figure 23.29: Flow diagram for Example 3.7



Example 23.8

Figure 23.30 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 23. However, these ACKs are not useful for the sender because the ackNo is equal to 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 23.30: Flow diagram for Example 3.8



Sender
Transport layer
Receiver
Transport layer
Initial
Start timer
Stop
Start
Time out
Restart
Restart
Stop timer
Req
aArr
Packet 0
ACK 1
pArr
Packet 1
Lost
Packet 2
Packet discarded
Packet 3
ACK discarded
Packet 1 (resent)
Packet 2 (resent)
Packet 3 (resent)
ACK 2
ACK 3
ACK 4
Time
Events:
Req: Request from process
pArr: Packet arrival
aArr: ACK arrival
time-out: Timer expiration