Lecture 11

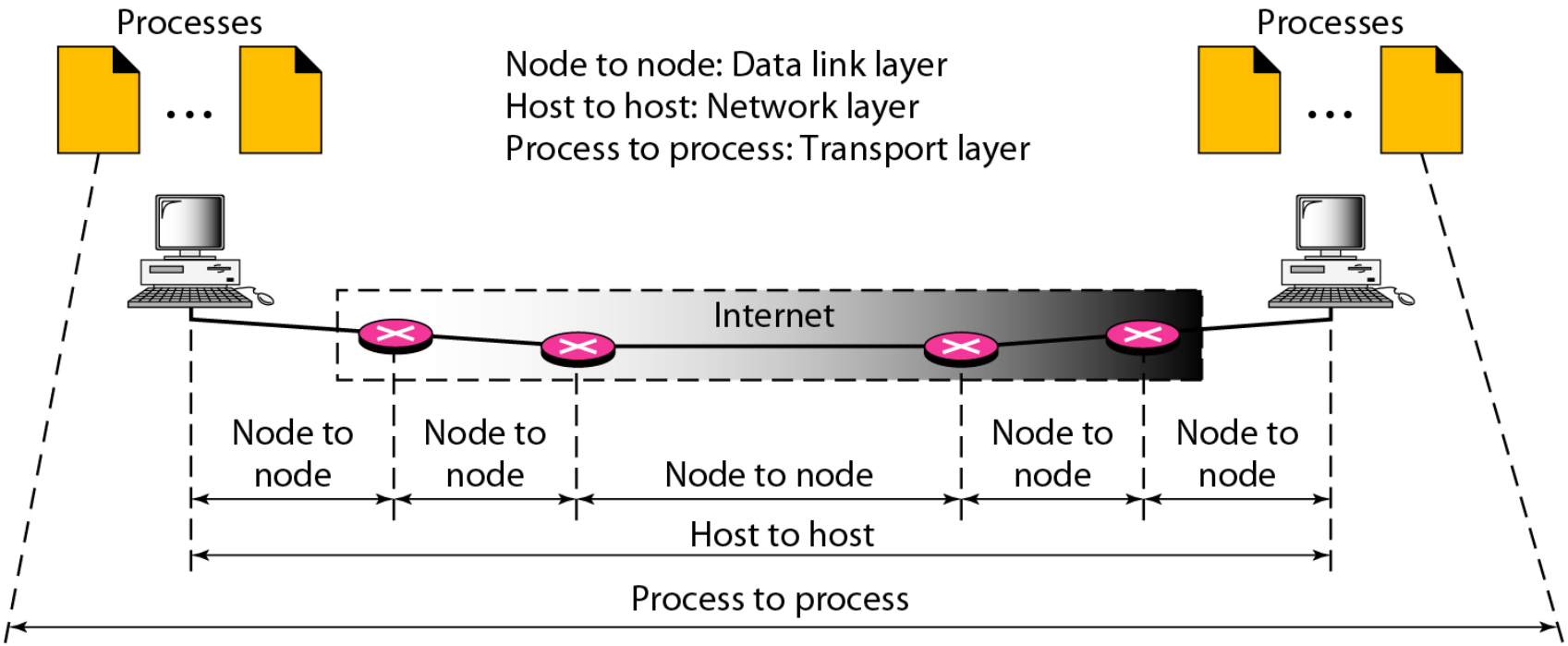
Transport Layer

23-1 PROCESS-TO-PROCESS DELIVERY

Real communication takes place between two processes (application programs). At any moment, several processes may be running on the source host and several on the destination host. We need a mechanism to deliver data from one process running on the source host to the corresponding process running on the destination host.

The transport layer is responsible for process-to-process delivery—the delivery of a packet, part of a message, from one process to another. Two processes communicate in a client/server relationship.

Figure 23.1 *Types of data deliveries*



Port numbers

In the transport layer, we need a transport layer address, called a port number, to choose among multiple processes running on a host. The destination port number is for delivery, while the source port number is needed for reply.

In the Internet model, the port numbers are 16-bit integers between 0 and 65535. The client program defines itself with a port number, chosen randomly by the transport layer software, referred to as ephemeral port number.

The Internet has decided to use universal port numbers for servers, called well-known port numbers, such that a client knows its corresponding port number at the server.

Figure 23.2 *Port numbers*

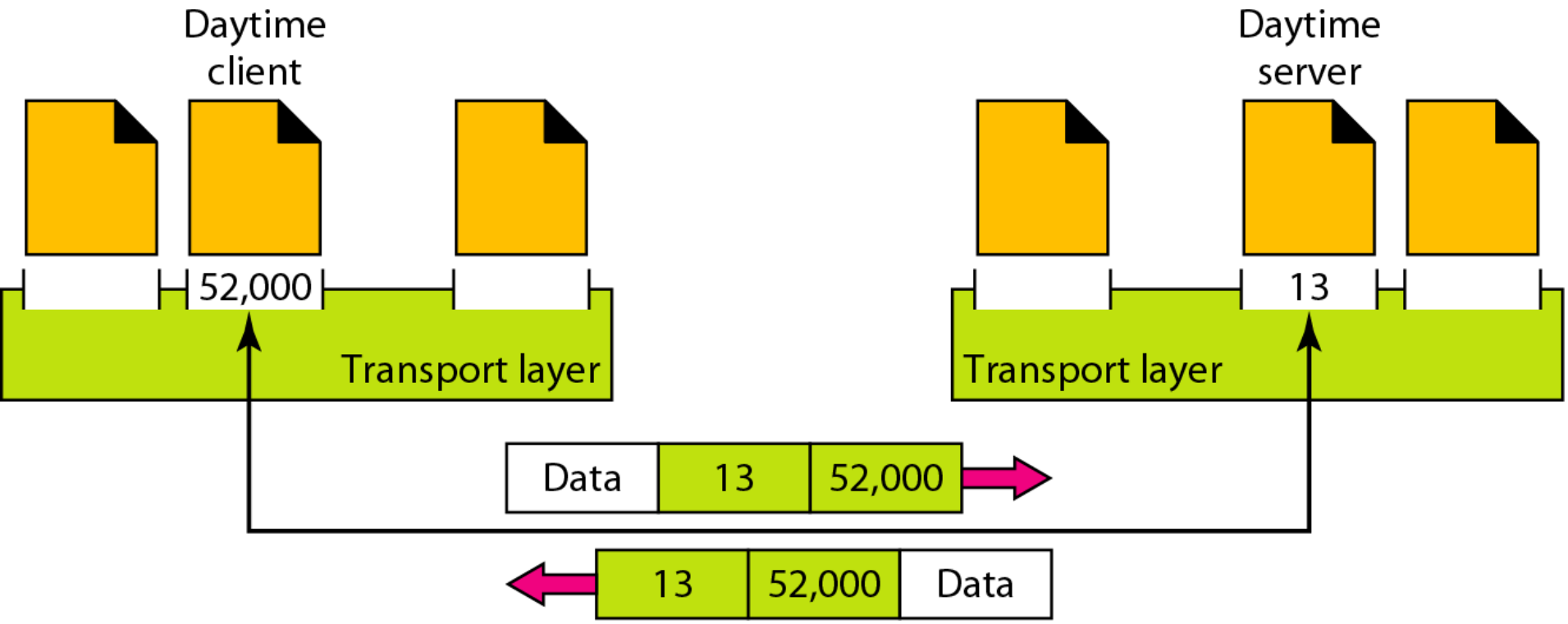


Figure 23.3 *IP addresses versus port numbers*

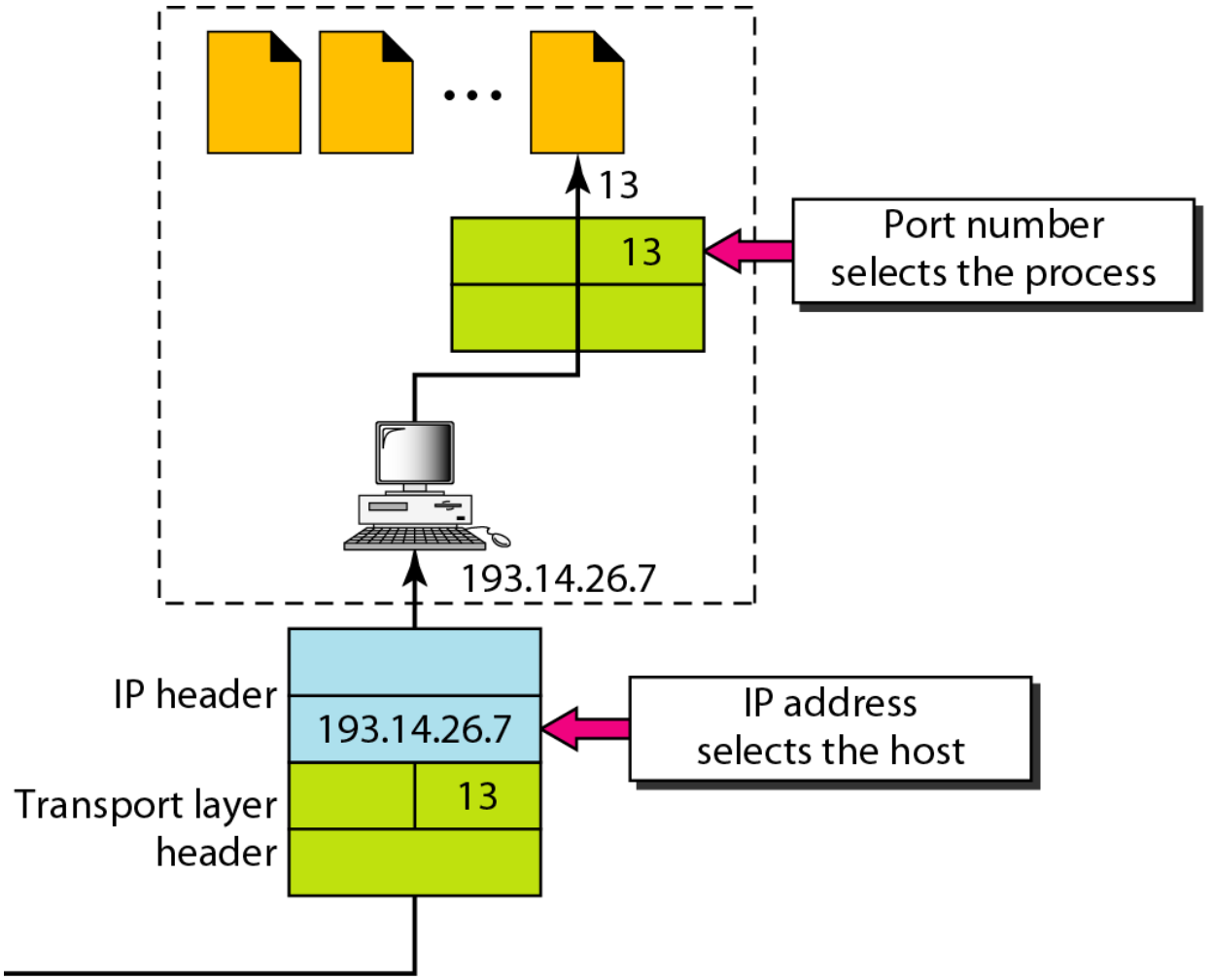


Figure 23.5 *Socket address*

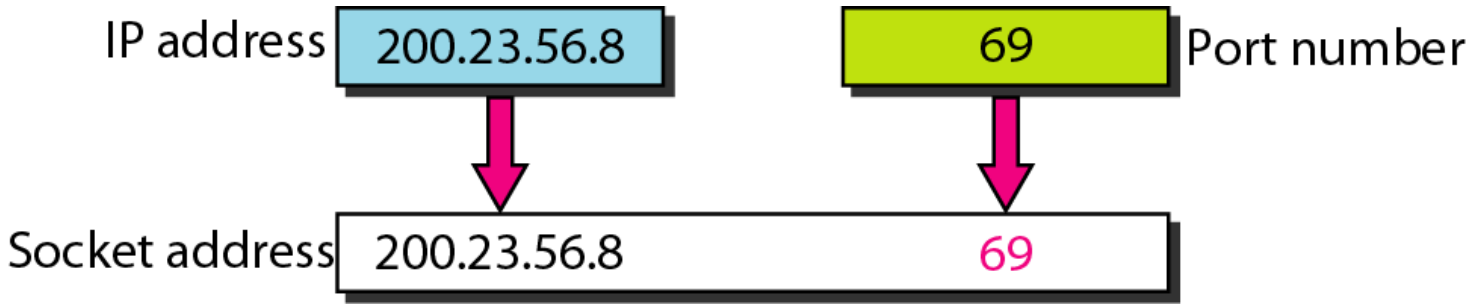
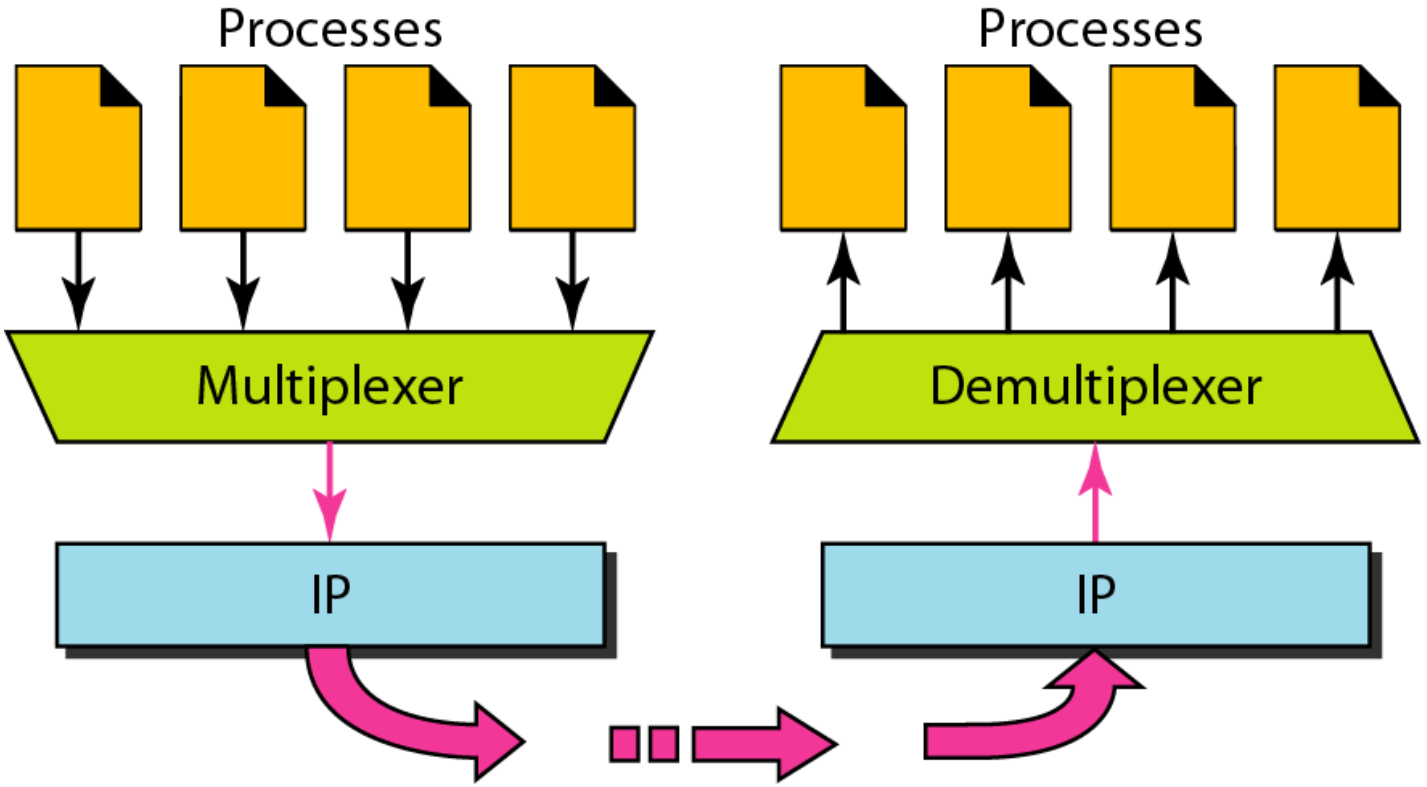


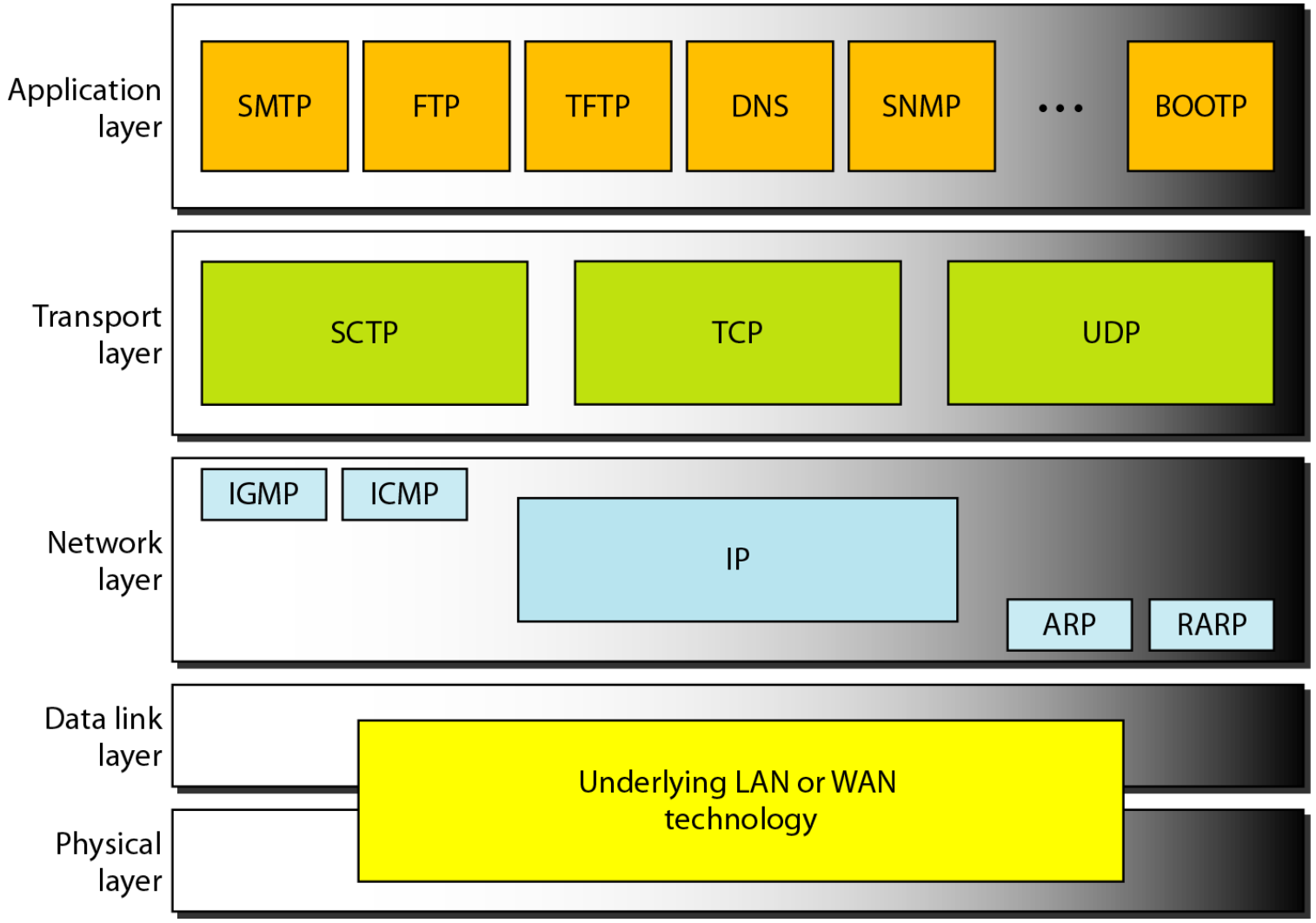
Figure 23.6 *Multiplexing and demultiplexing*



At each host, there are several processes that need to send/receive packets. However, there is only one transport layer protocol.

Multiplexing: the transport layer protocol accepts messages from different processes, differentiated by their assigned port numbers. After adding header, the transport layer passes the packets to L3.

Figure 23.8 *Position of UDP, TCP, and SCTP in TCP/IP suite*



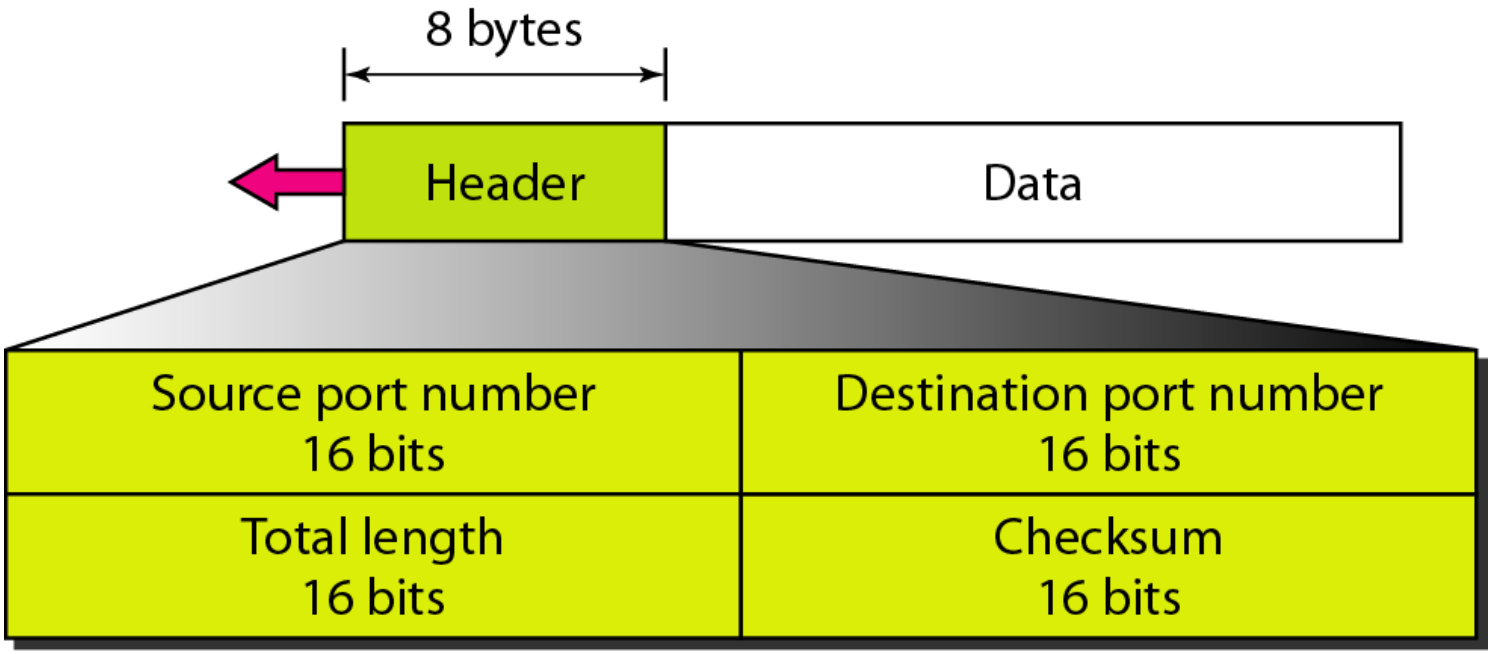
23-2 USER DATAGRAM PROTOCOL (UDP)

The User Datagram Protocol (UDP) is called a connectionless, unreliable transport protocol (no flow or error control). It does not add anything to the services of IP except to provide process-to-process communication instead of host-to-host communication. UDP data units are called user datagrams.

Table 23.1 *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	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)

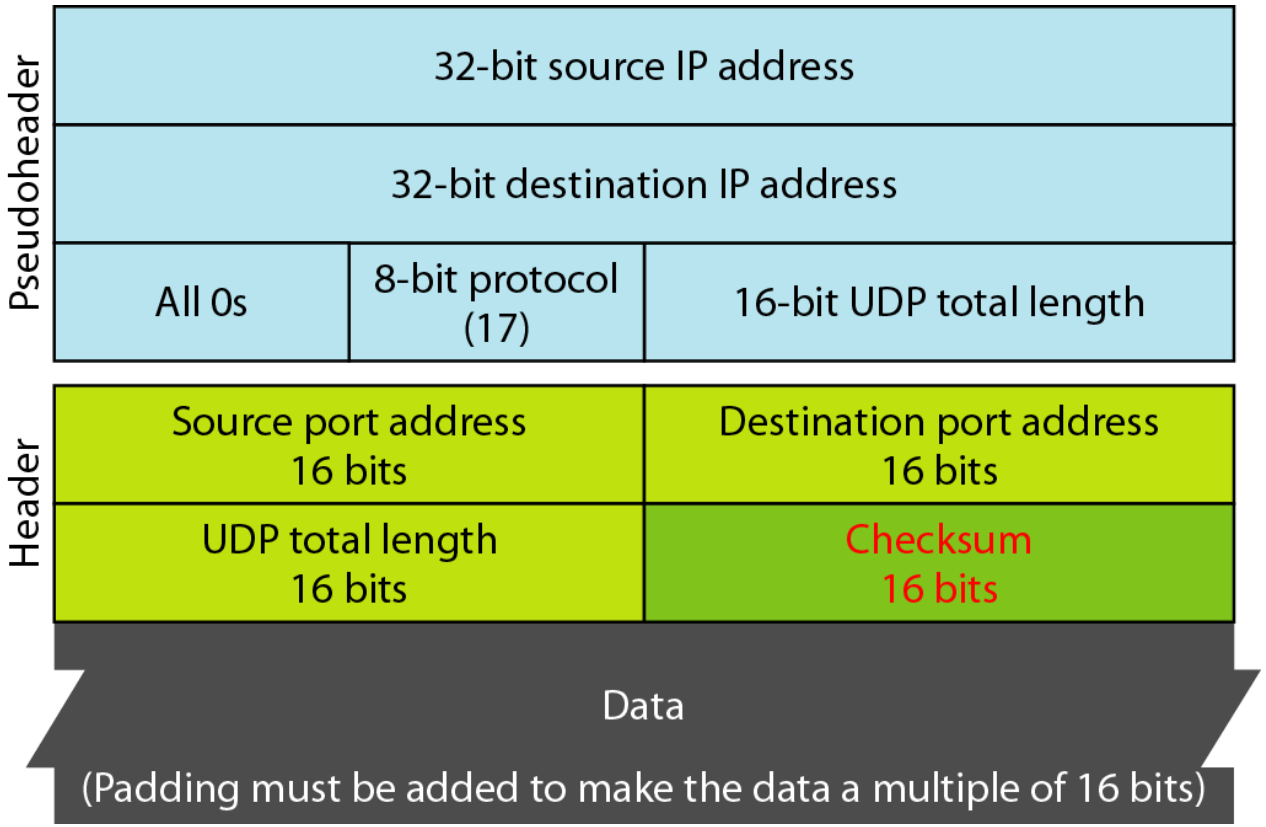
Figure 23.9 *User datagram format*



Note

UDP length
= IP length – IP header's length

Figure 23.10 *Pseudoheader for checksum calculation*



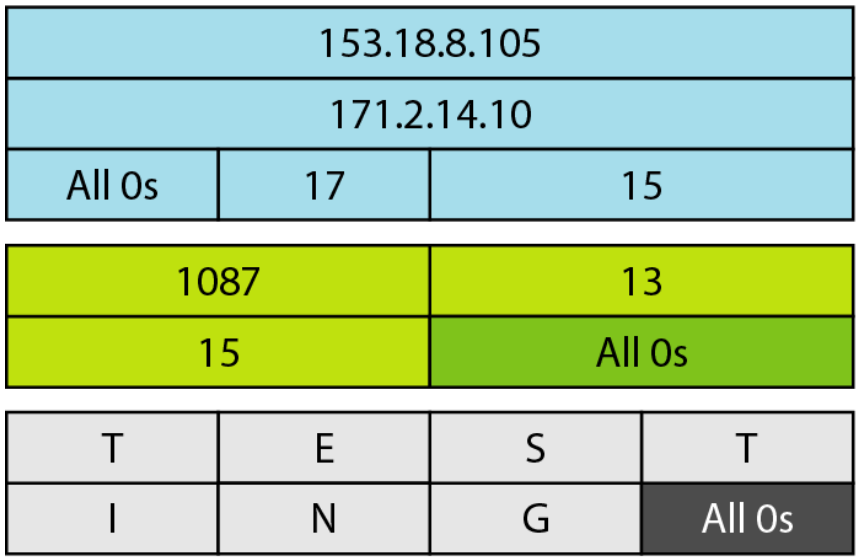
Pseudoheader and padding are only for checksum calculation. Will NOT be sent.

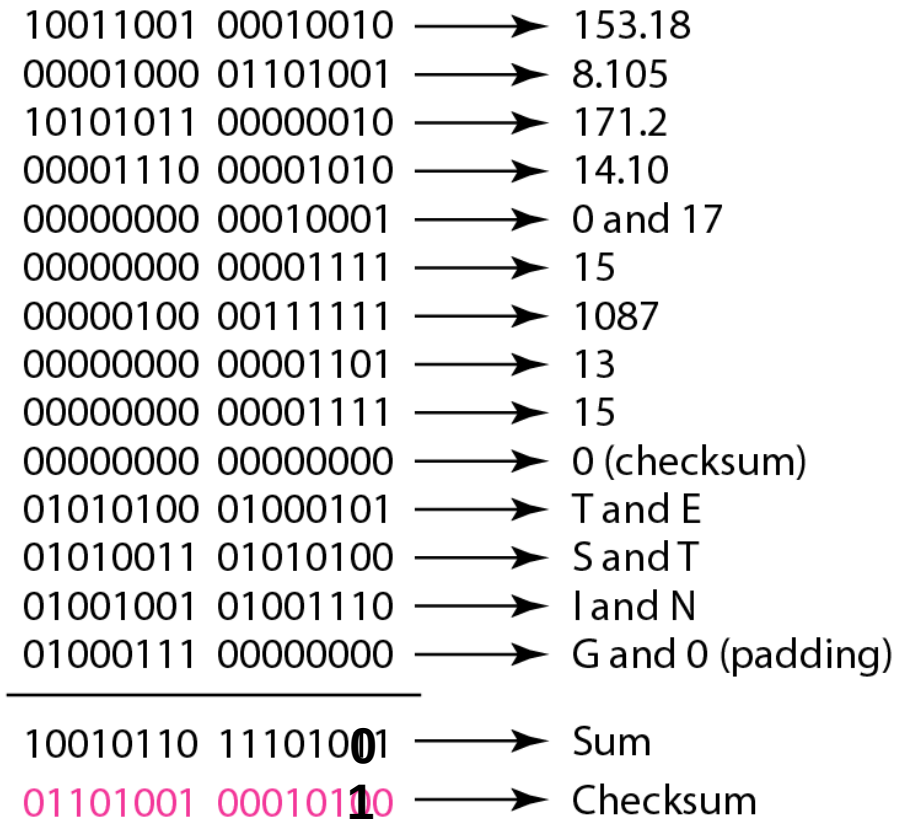
Purpose of pseudoheader: provide protection against misrouted datagrams.

Example 23.2

Figure 23.11 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.

Figure 23.11 *Checksum calculation of a simple UDP user datagram*





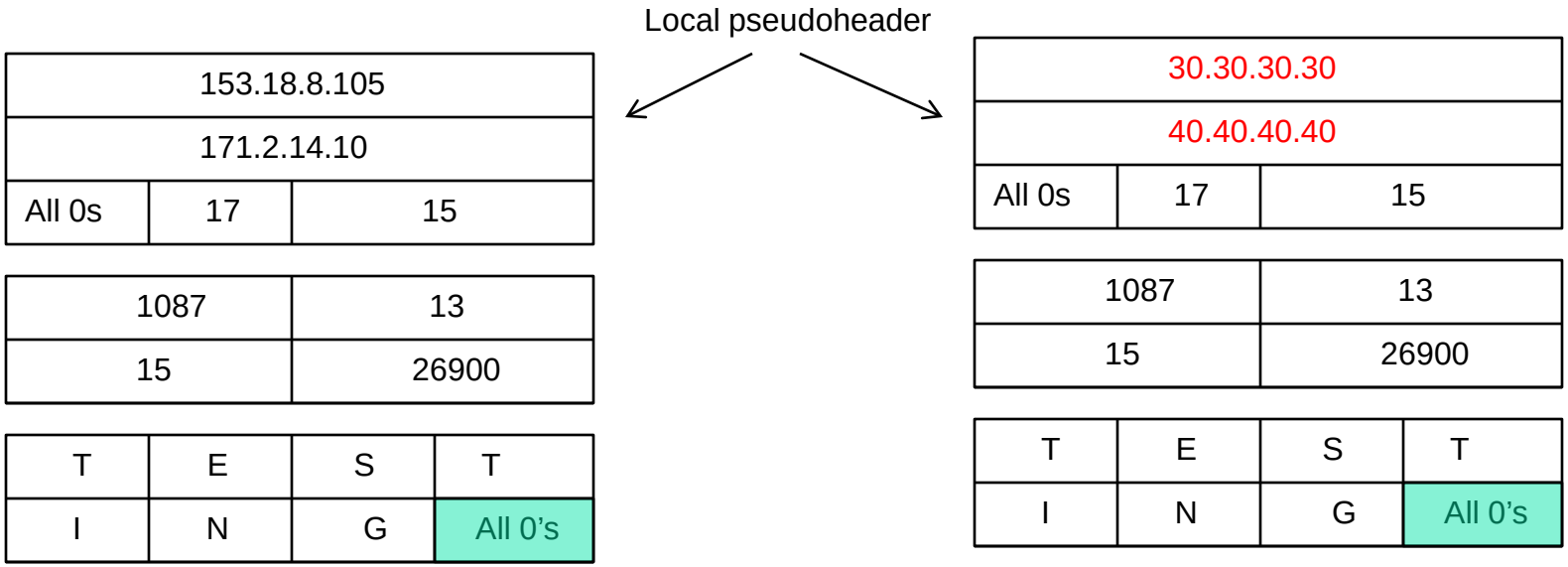
Carry is dropped at the most significant bit.

Inclusion of the checksum in a UDP datagram is optional.

If the checksum is not calculated, fill the field by all 1s.

Checksum checking at the receiver

The receiver adds padding, and pseudoheader based on its local information.



If received by correct receiver: addition of all fields gets all 1s (binary), and thus, the user datagram is accepted

If received by a wrong receiver: addition of all fields does not get all 1s (binary), and thus, the user datagram is dropped

23-3 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.

Like UDP, TCP provides process-to-process communication using port numbers

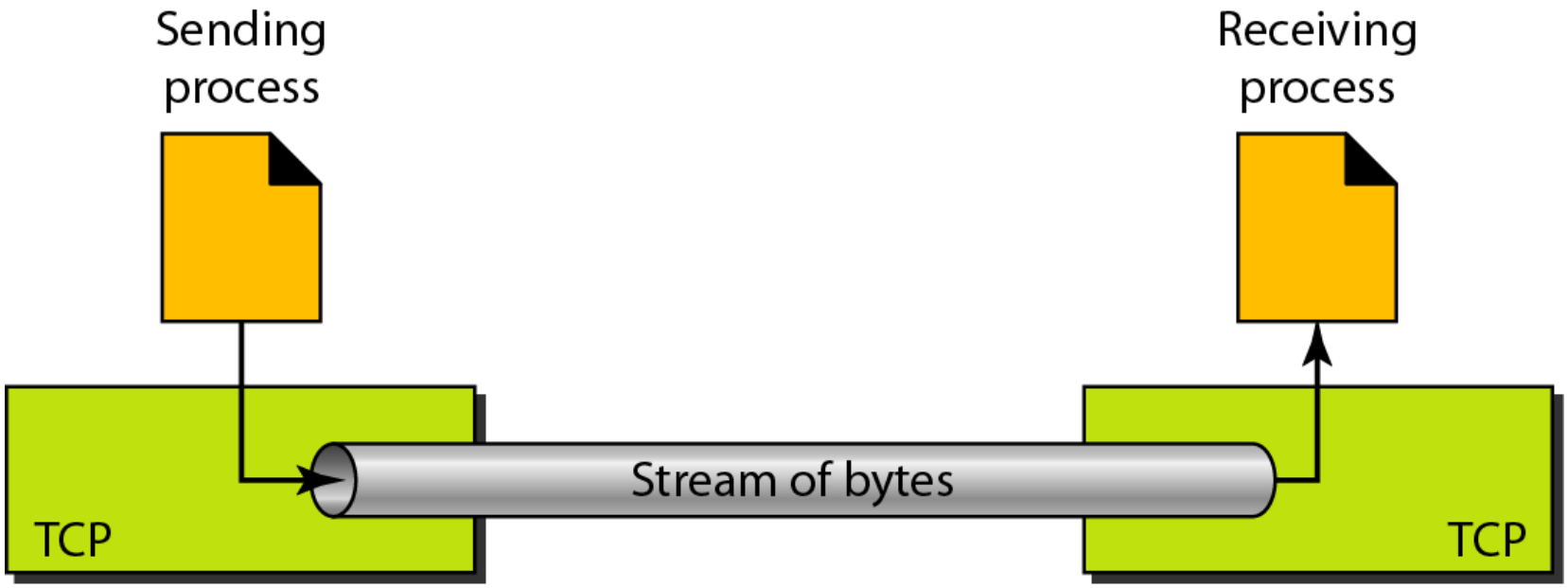
Table 23.2 *Well-known ports used by TCP*

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

Stream delivery

TCP creates an environment in which the sending and receiving processes seem to be connected by an imaginary "tube" that carries their data across the Internet.

In the transport layer, TCP groups a number of bytes together into a packet called segment. TCP adds a header to each segment and delivers the segment to the IP layer for transmission.



Byte number and sequence number

Each byte the TCP receives has a byte number. For the first byte, TCP generates a random number between 0 and 2^{32}-1. For example, it the number happens to be 1057 and the total data to be sent are 6000 bytes, then the bytes are numbered from 1057 to 7056. The byte numbering is used for flow and error control.

The sequence number of each segment is the number of the first byte carried in that segment.

Example 23.3

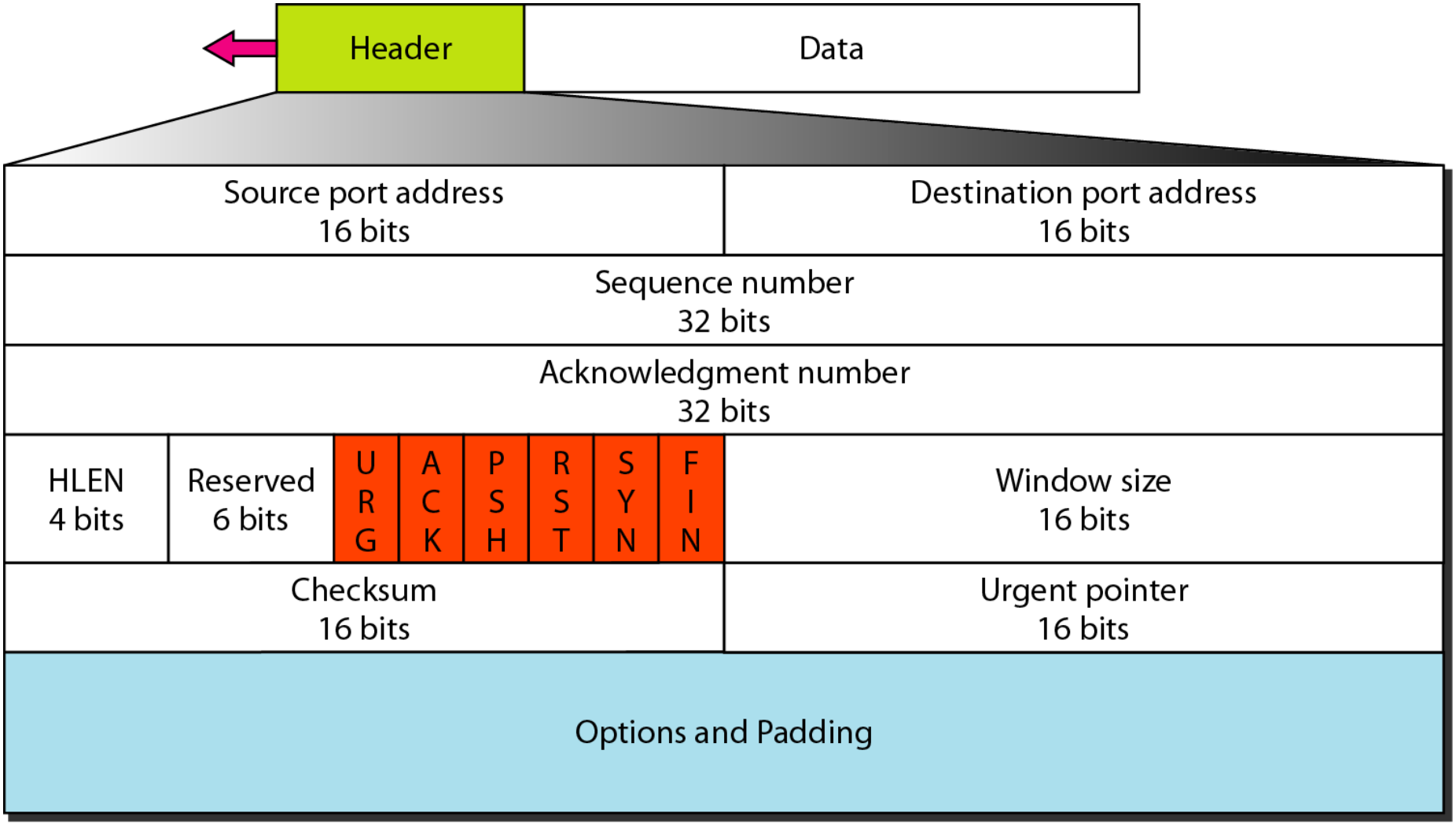
A TCP connection is transferring a file of 5000 bytes. The following shows the sequence number for each segment:

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

Figure 23.16 *TCP segment format*



HLEN: the number of 4-byte words in the TCP header. 5~15

Window size: define the size of the window, in bytes, that the other party must maintain. The length of the field is 16 bits, which means the maximum size of the window is 65536 bytes. This value is normally referred to as the receiving window (rwnd) and is determined by the receiver. The sender must obey the dictation of the receiver in this case.

Checksum: follows the same procedure as that for UDP. Inclusion of the checksum for TCP is mandatory. The same pseudoheader, serving the same purpose, is added to the segment. For the TCP pesudoheader, the value for the protocol field is 6.

Options: up to 40 bytes.

Figure 23.17 *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

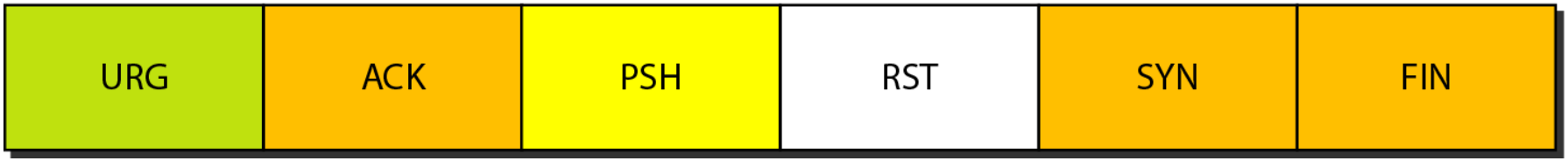
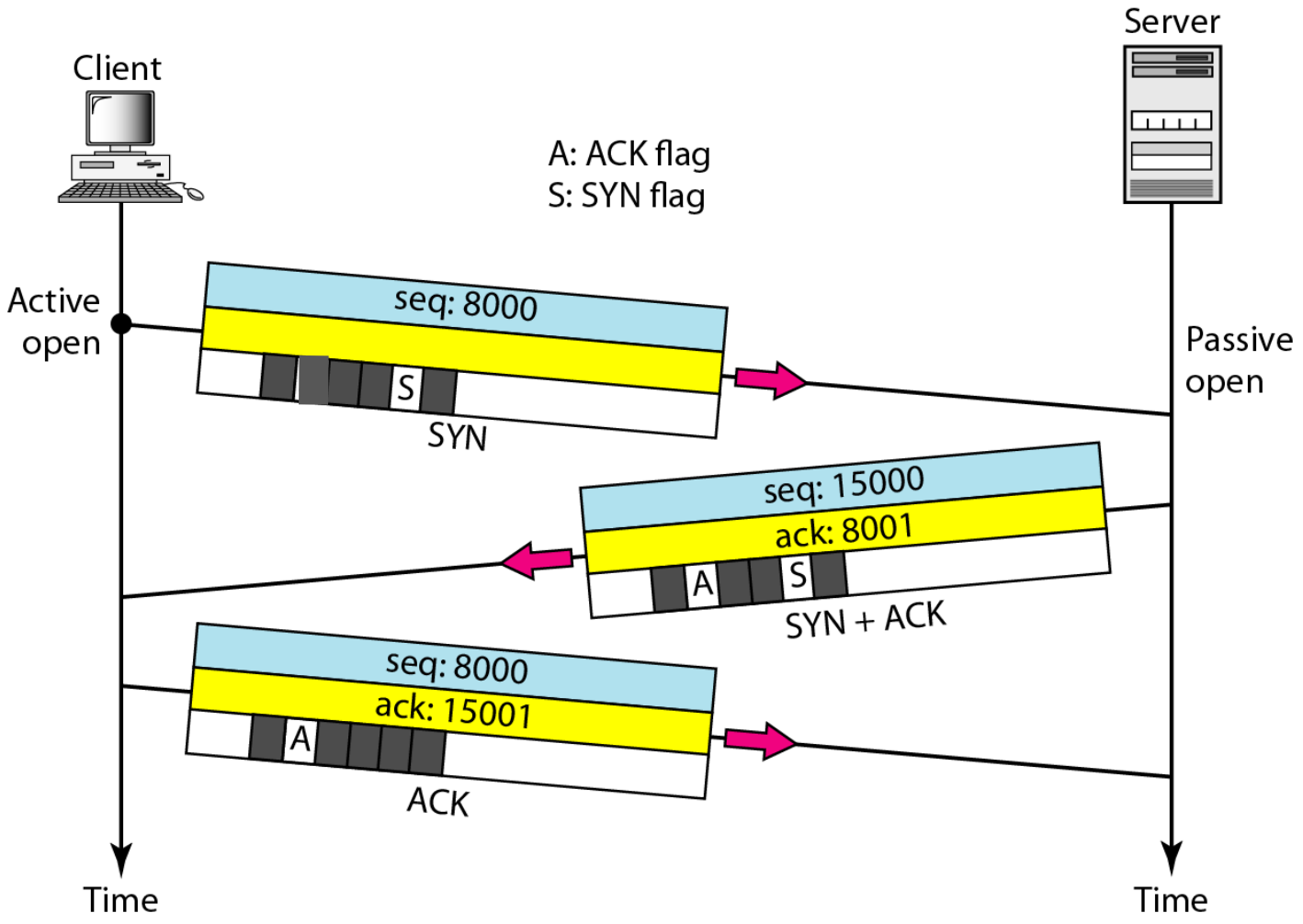


Table 23.3 ***Description of flags in the control field***

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

Figure 23.18 *Connection establishment using three-way handshaking*



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 23.19 *Data transfer*

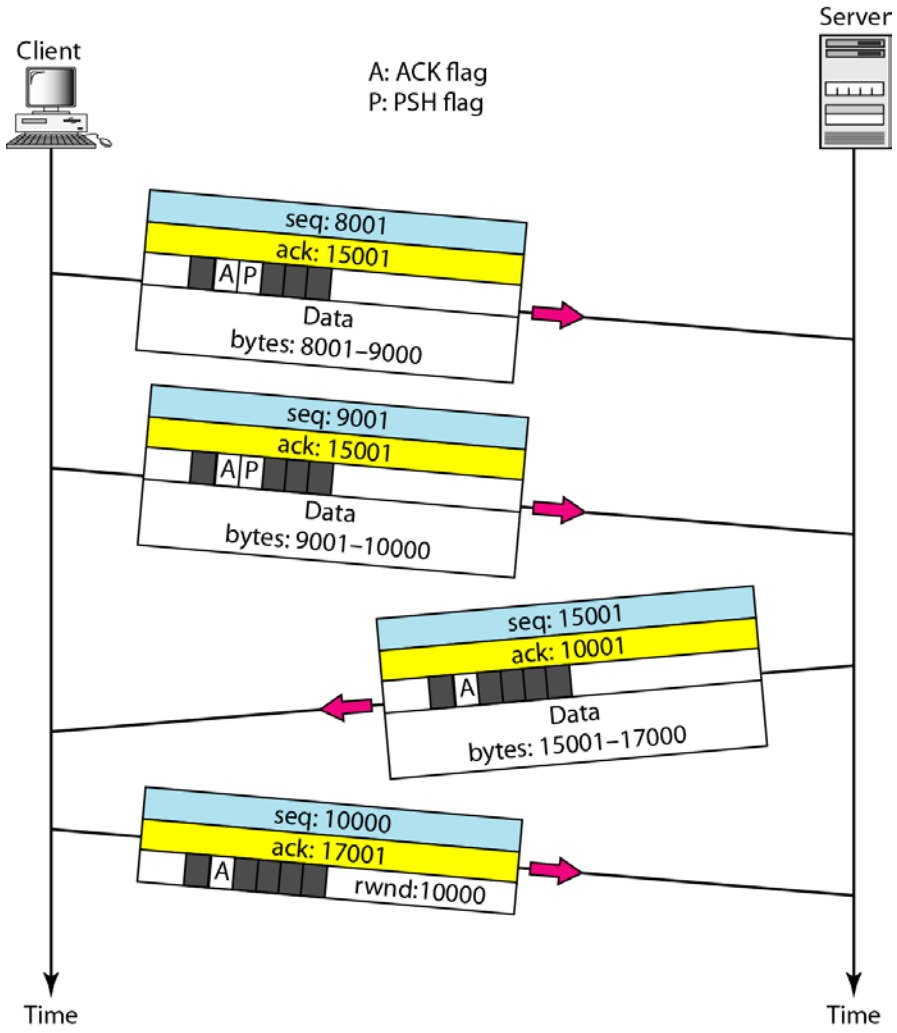
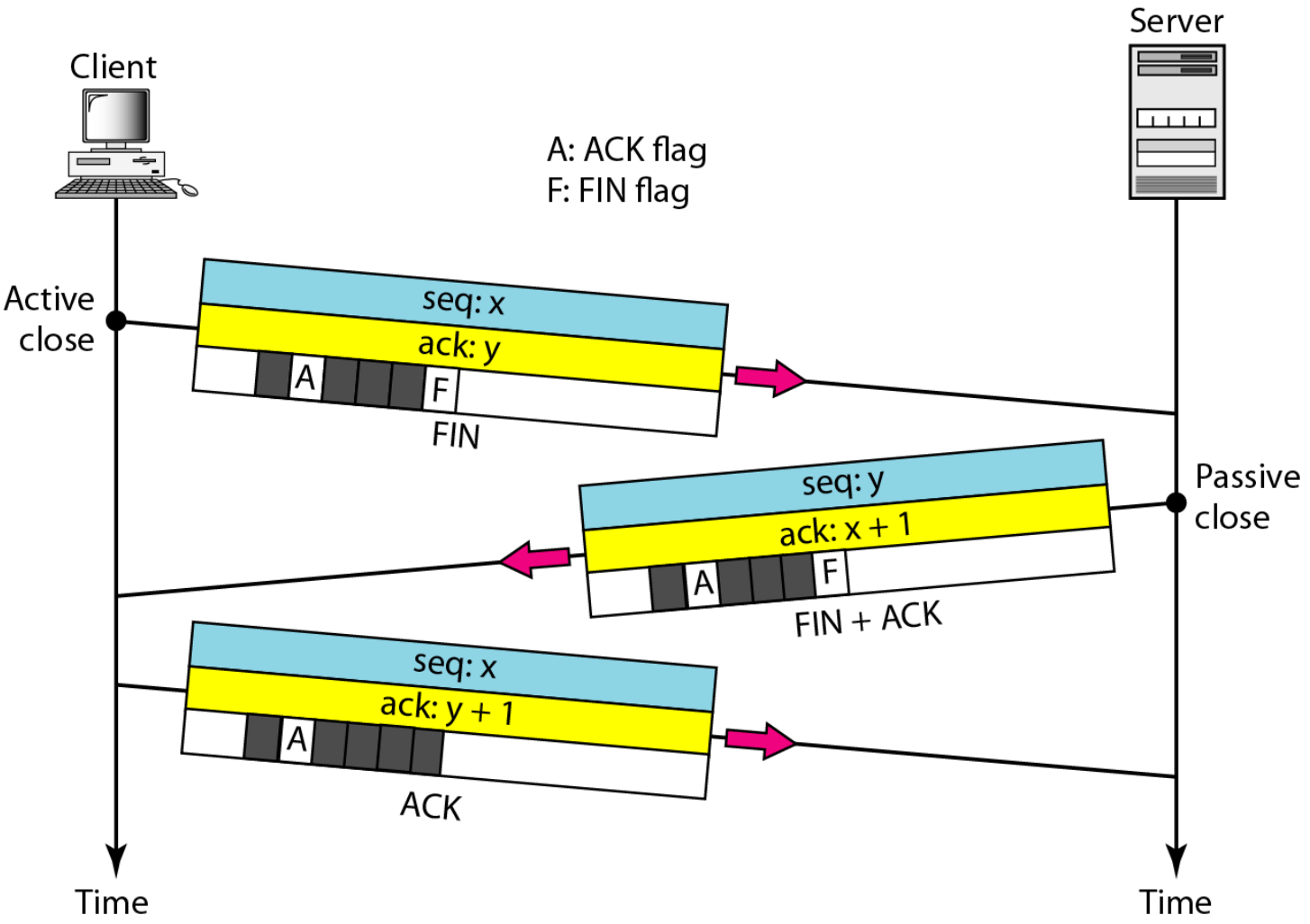


Figure 23.20 *Connection termination using three-way handshaking*



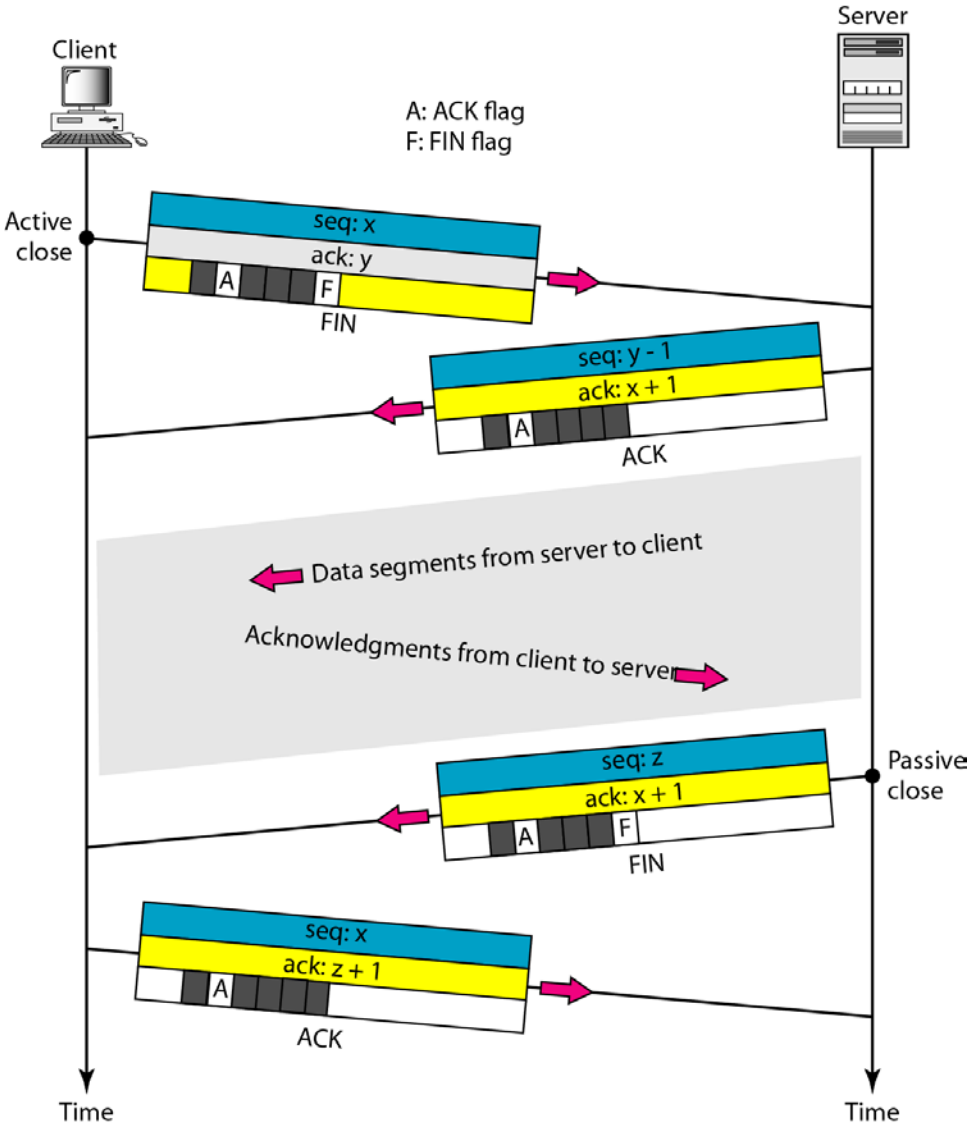
Note

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

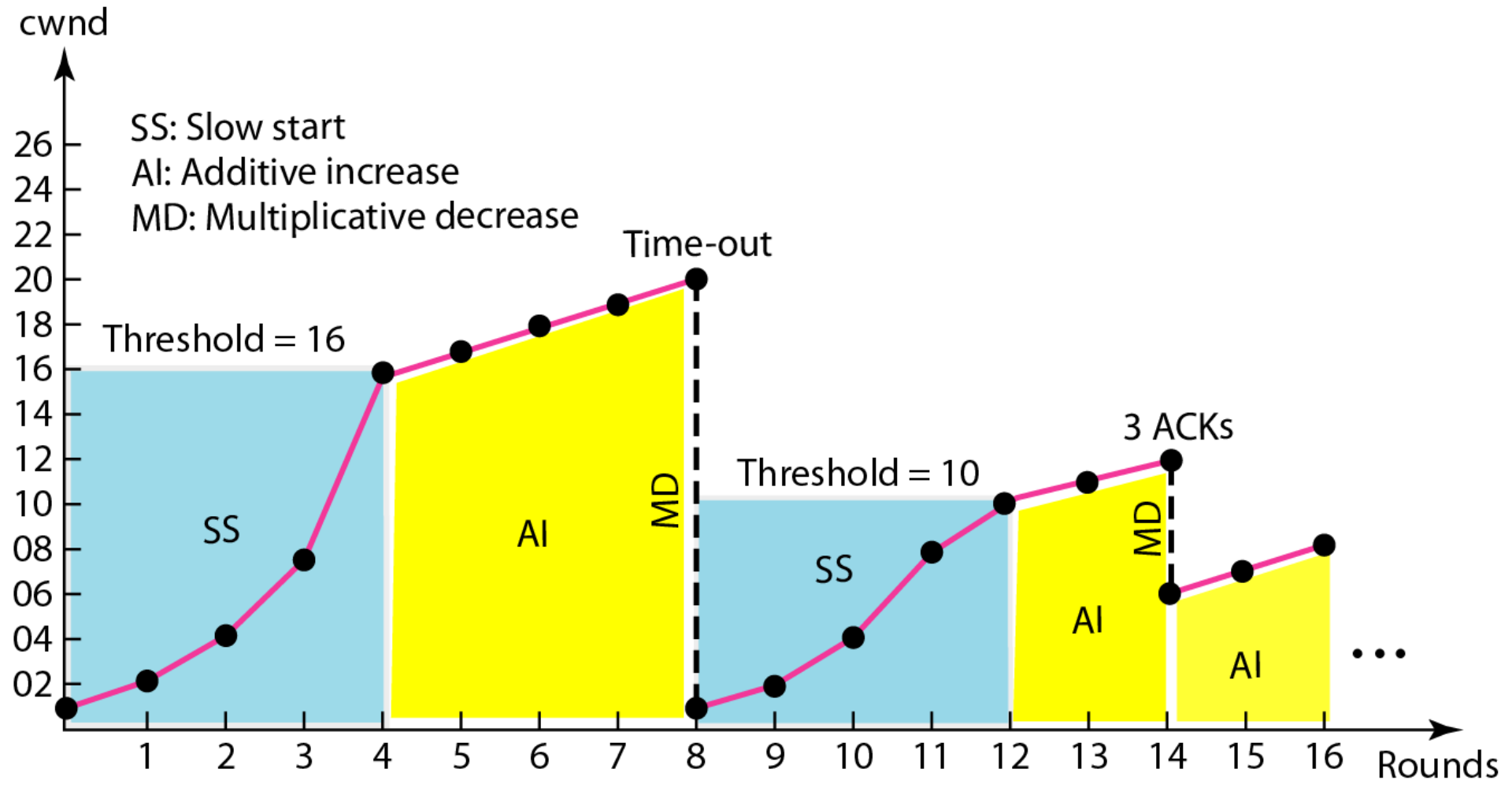
Note

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

Figure 23.21 *Half-close*



TCP *Congestion example*



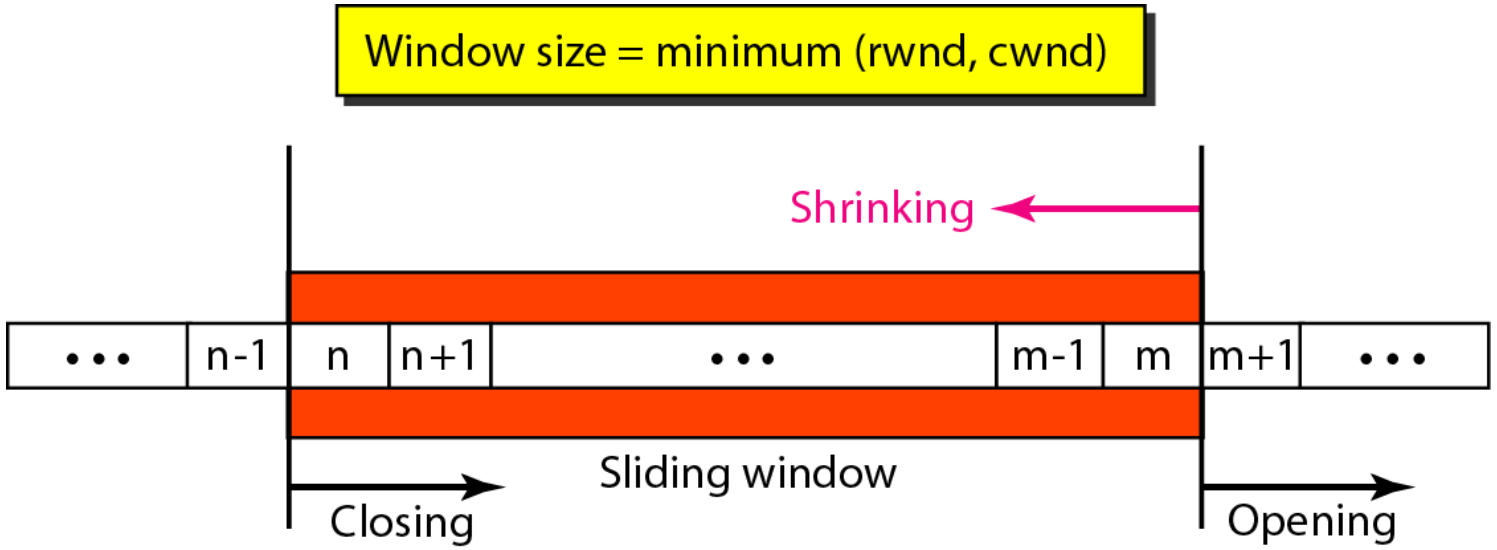
Note

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

Note

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

Figure 23.22 *Sliding window for flow control*



Receiver window (rwnd): the value advertised by the opposite end in a segment containing acknowledgement. It is the number of bytes the other end can accept before its buffer overflows and data are discarded.

Congestion window (cwnd): a value determined by the network to avoid congestion.

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.

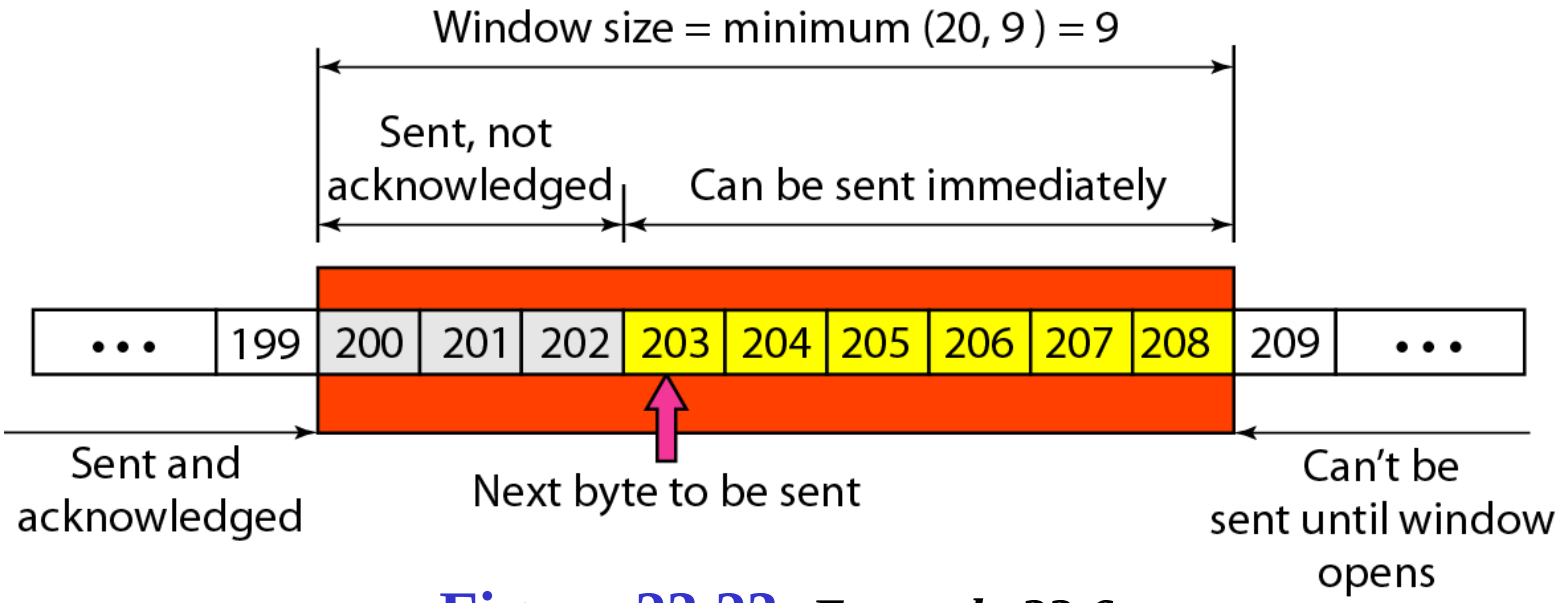


Figure 23.23 ***Example 23.6***

Note

In modern implementations, a retransmission occurs if the retransmission timer expires or three duplicate ACK segments have arrived.

Note

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

Figure 23.24 *Normal operation*

The client does not have further data to be sent to the server.

If it has data, the client does not wait. It will send data and ACK in a single segment.

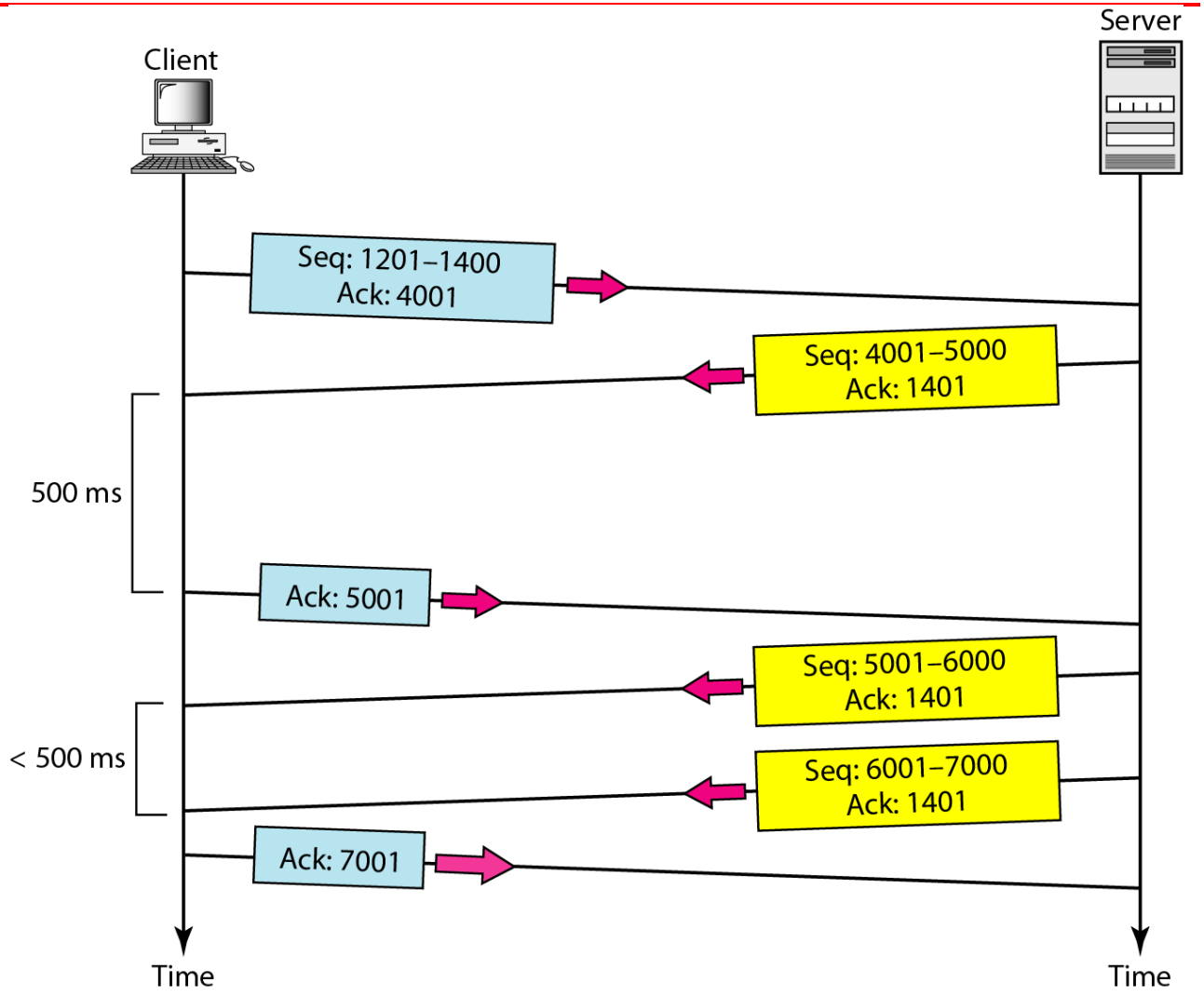
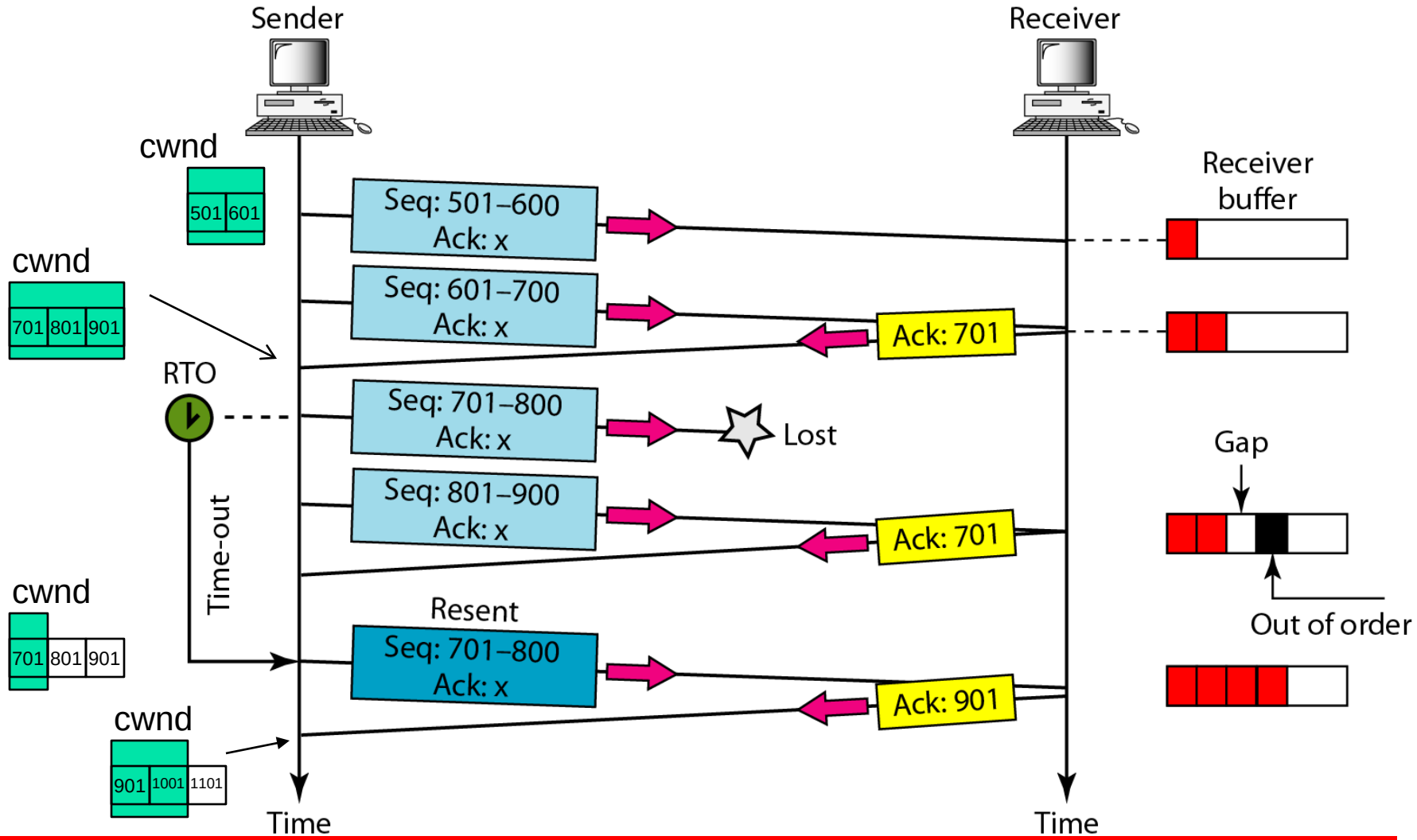


Figure 23.25 *Lost segment*

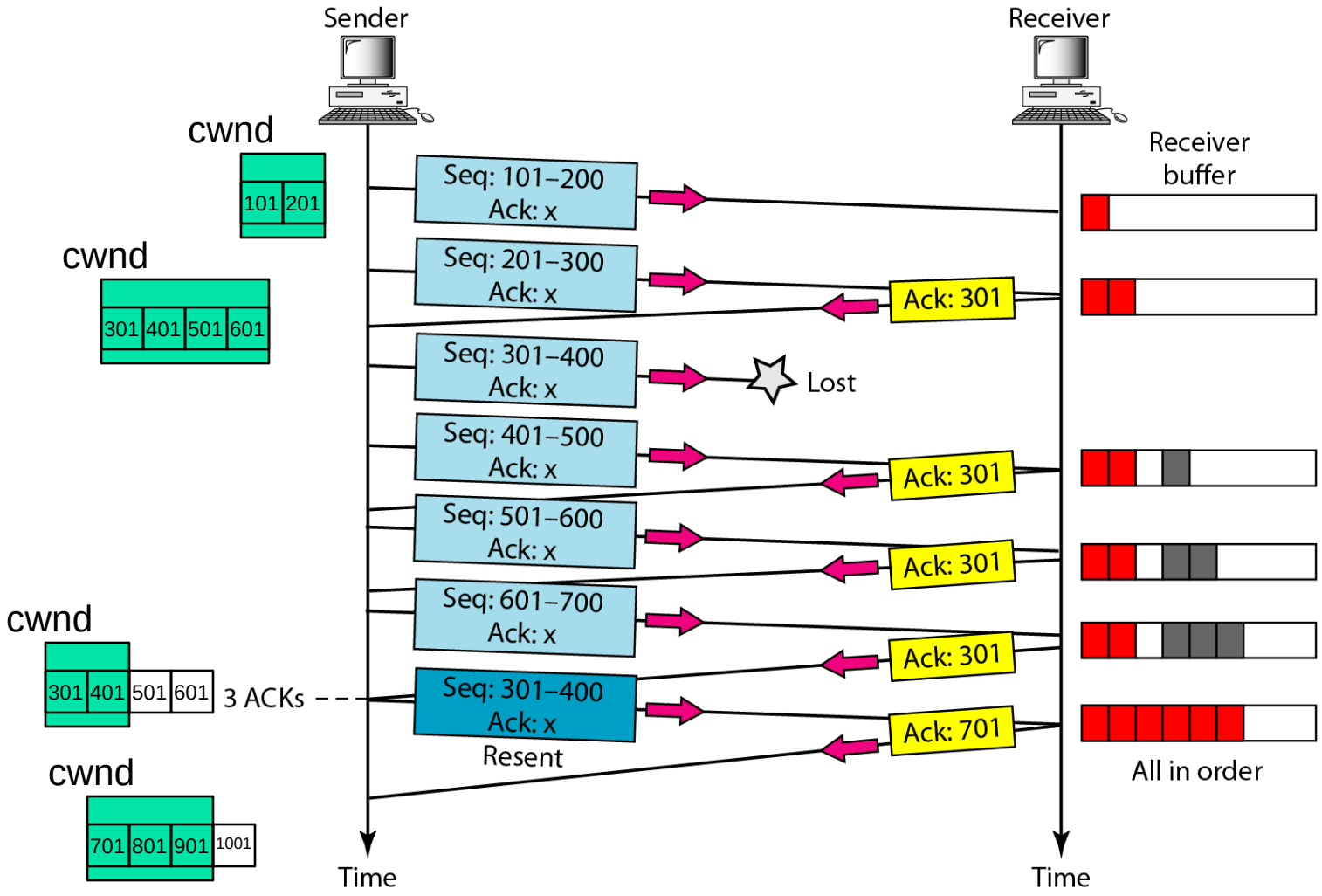


Assume the TCP is in AI at the beginning. When ACK 701 is received, the sender only has two segments to send.

Note

The receiver TCP delivers only ordered data to the process.

Figure 23.26 *Fast retransmission*



Assume the TCP is in slow start at the beginning.