
# Introduction to Data Communications (COMP 3721)

Instructor: Maryam Tanha

Fall 2021

# Learning Outcomes of This Lecture

- **By the end of this lecture you will be able to**
  - Explain how TCP works and what are its characteristics.

# Review

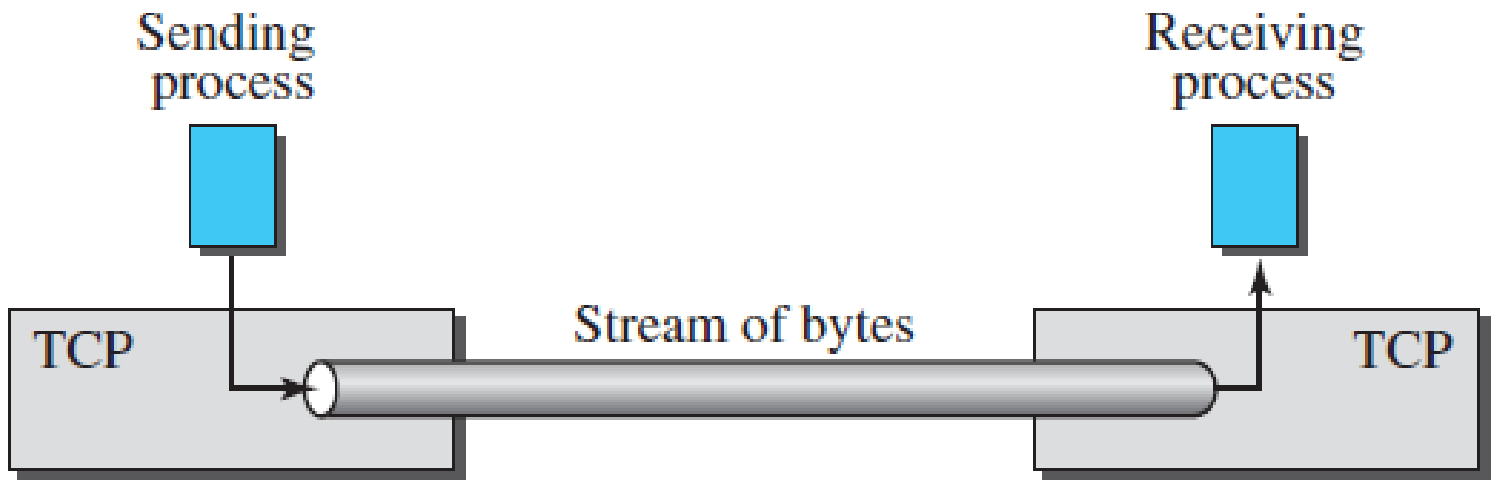
- **Transport layer**
  - process-to-process communications
  - addressing port numbers
    - ✓ temporary port numbers (>1023) for client processes
  - socket address -> IP address and port number

# TCP (Transmission Control Protocol)

- A reliable connection-oriented protocol.
- TCP packets are called segments
  - TCP groups a number of bytes together into a segment.
- TCP explicitly defines connection establishment, data transfer, and connection teardown phases (connection-oriented service)
  - a single logical pathway is used for the segments belonging to the same message.
- The connection is logical and data is exchanged in both directions (**full-duplex service**).
- TCP connection is point-to-point (i.e., between a single sender and a single receiver).

# TCP – A Stream-Oriented Protocol

- TCP, unlike UDP, is a stream-oriented (byte-oriented) protocol
  - the sending process delivers data as a stream of bytes and the receiving process obtains data as a stream of bytes



# TCP – Sending and Receiving Buffers

- **Why does TCP need buffers for storage?**

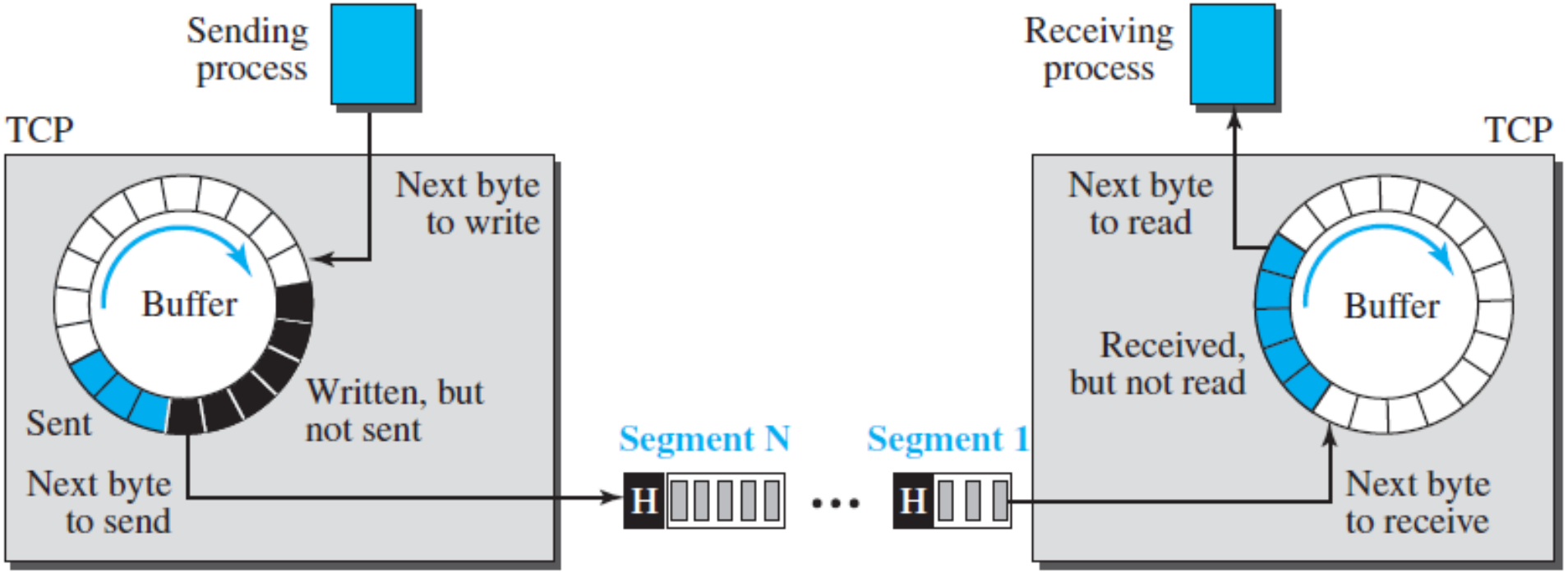
# TCP – Sending and Receiving Buffers

- **Why does TCP need buffers for storage?**
  - because the sending and the receiving processes may not necessarily write or read data at the same rate.

# TCP – Sending and Receiving Buffers

- **Why does TCP need buffers for storage?**
  - because the sending and the receiving processes may not necessarily write or read data at the same rate
- Two buffers, the sending buffer and the receiving buffer, one for each direction.
- Buffers are necessary for flow control and error control.
- Buffer is implemented as a circular array.

# TCP – Sending and Receiving Buffers – Continued




Note that segments are not necessarily all the same size

# TCP – Numbering System

- **TCP numbers all data bytes (octets) that are transmitted in a connection.**
- Numbering is independent in each direction.
- Byte numbering is used for flow and error control.
- **Two fields in the segment header**
  - *sequence number*
  - *acknowledgment number*
  - each of the above fields refer to a byte number and not a segment number.

# TCP – Sequence Number

- **The value in the sequence number field of a segment indicates the number assigned to the first data byte contained in that segment.**
- The sequence number, in each direction, is indicated as follows:
  1) the sequence number of the first segment is called ISN (initial sequence number), which is a random number (usually between 0 and $2^{32}$ -1).
  2) the sequence number of any other segment is the sequence number of the previous segment plus the number of bytes carried by the previous segment.
- Some segments, when carrying only control information, need a sequence number to allow an acknowledgment from the receiver.
  - segments for connection establishment, termination, or abortion.
  - each of these segments consume one sequence number as though it carries one byte, but there are no actual data.

# TCP – Sequence Number (Example)

Suppose a TCP connection is transferring a file of 5000 bytes. The first byte is numbered 10001. What are the sequence numbers for each segment if data are sent in five segments, each carrying 1000 bytes?

# TCP – Sequence Number (Example) – Solution

Suppose a TCP connection is transferring a file of 5000 bytes. The first byte is numbered 10001. What are the sequence numbers for each segment if data are sent in five segments, each carrying 1000 bytes?

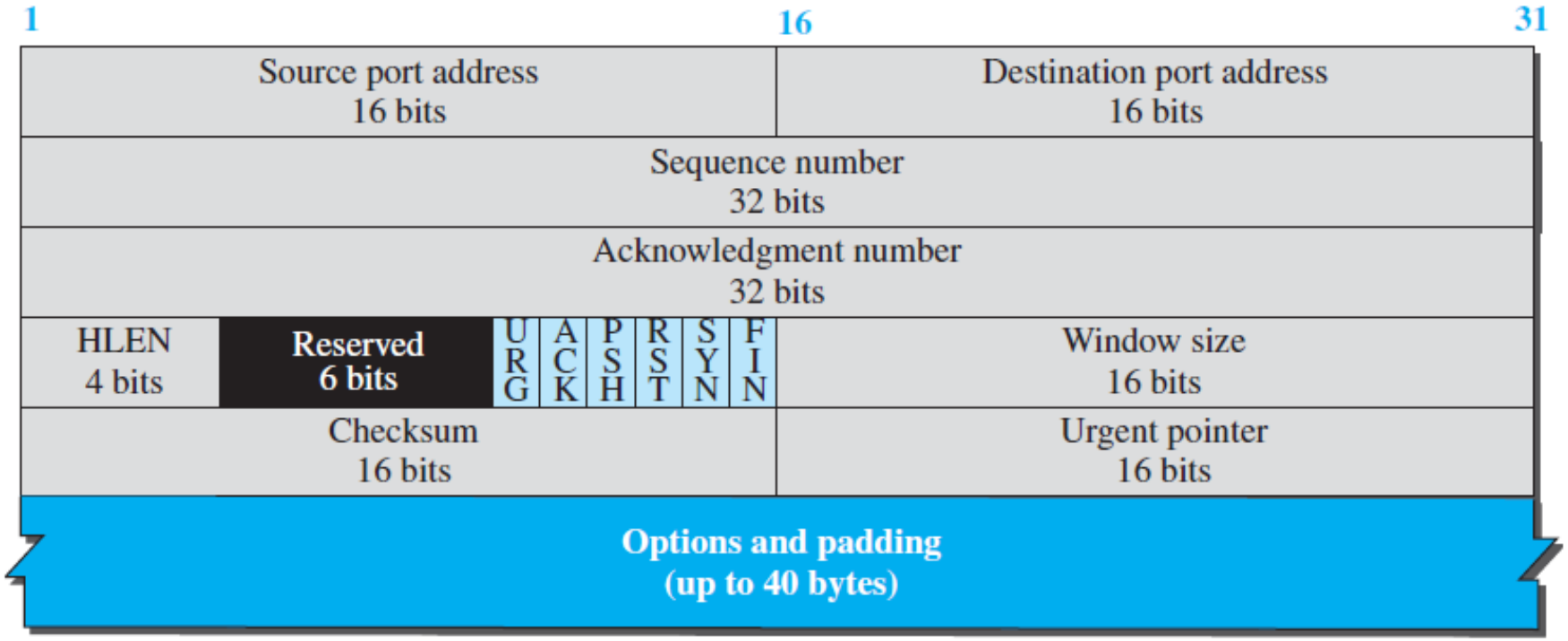
| | | | | | | | |
|---|---|---|---|---|---|---|---|
| Segment 1 | → | Sequence Number: | 10001 | **Range:** | 10001 | to | 11000 |
| Segment 2 | → | Sequence Number: | 11001 | **Range:** | 11001 | to | 12000 |
| Segment 3 | → | Sequence Number: | 12001 | **Range:** | 12001 | to | 13000 |
| Segment 4 | → | Sequence Number: | 13001 | **Range:** | 13001 | to | 14000 |
| Segment 5 | → | Sequence Number: | 14001 | **Range:** | 14001 | to | 15000 |

# TCP – Acknowledgement Number

- Each party uses an acknowledgment number to confirm the bytes it has received.
- 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**.
  - means that the party takes the number of the last byte that it has received, safe and sound, adds 1 to it, and announces this sum as the acknowledgment number.
  - in other words, **TCP only acknowledges bytes up to the first missing byte in the stream.**

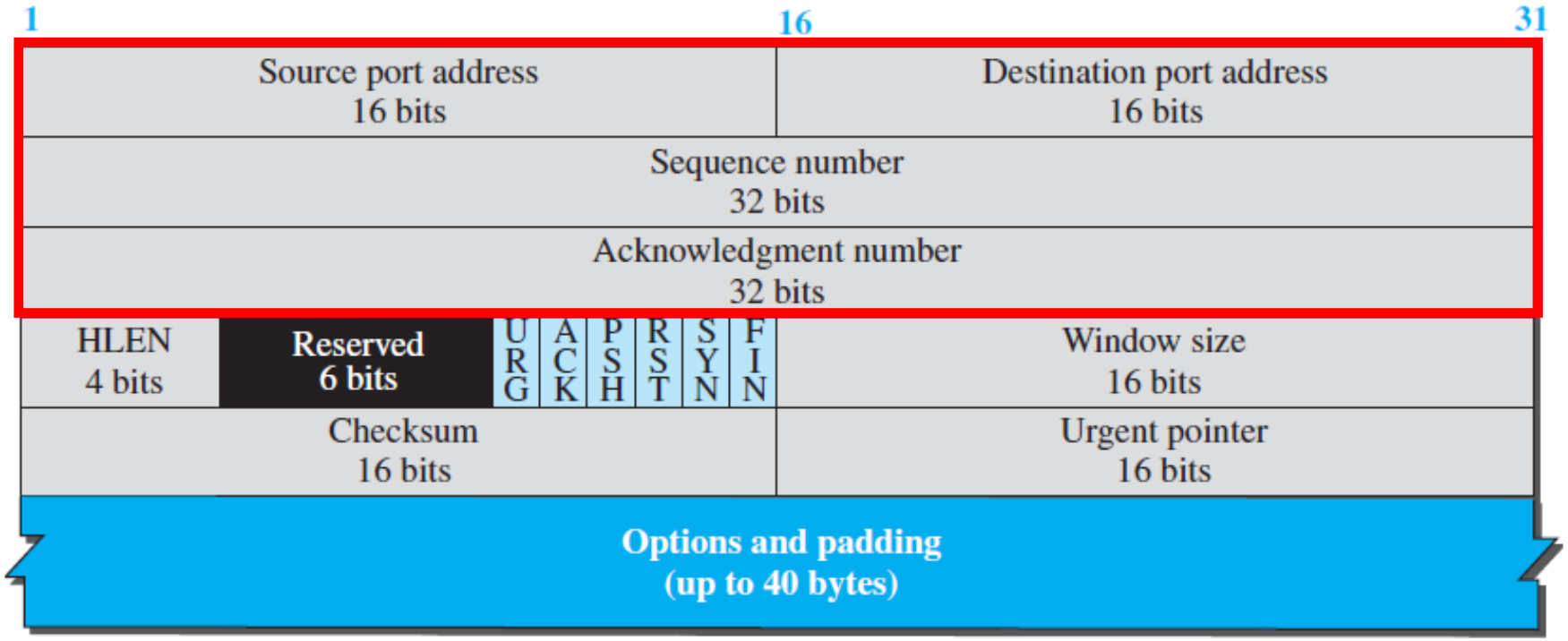
# TCP Segment – Format


a. Segment




b. Header

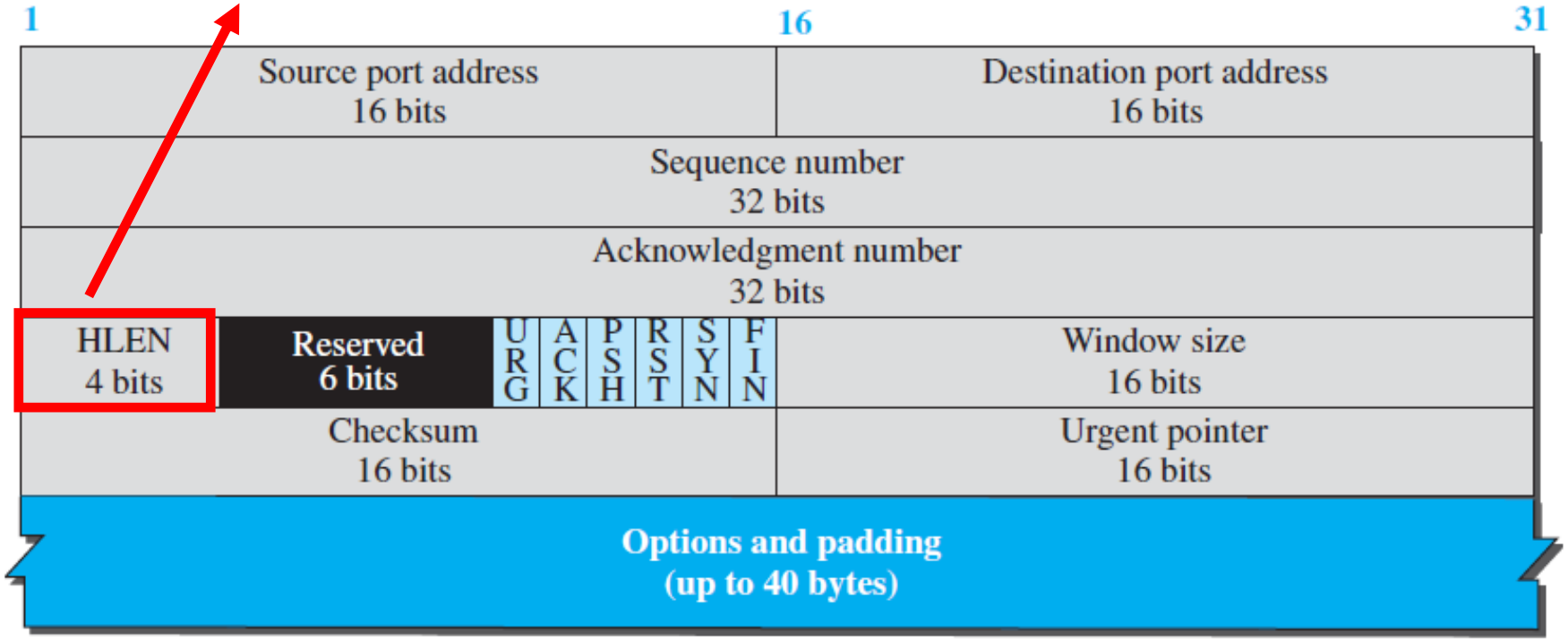
# TCP Segment – Format




b. Header

# TCP Segment – Format

**The length of the TCP header in words (4-bytes)**




b. Header

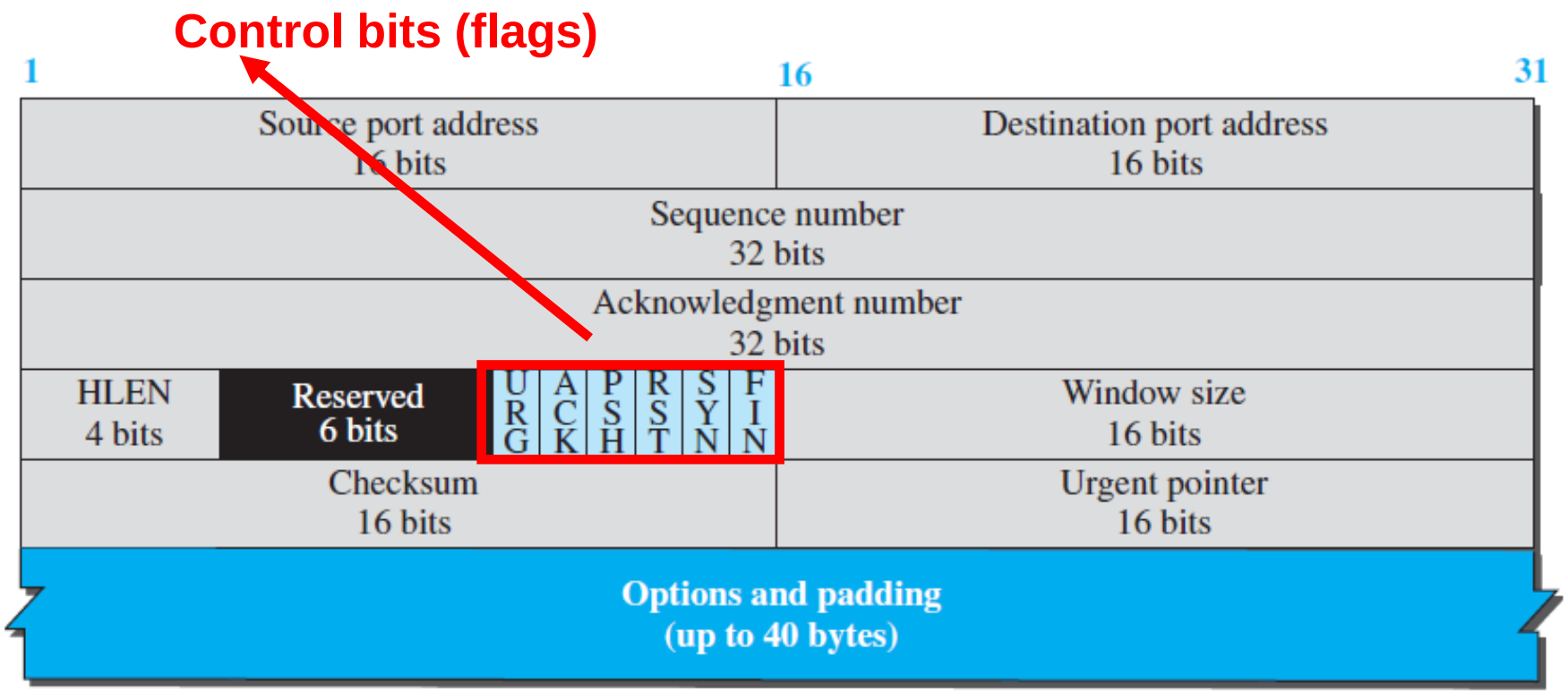
# TCP Segment – Format

**URG**: urgent data (generally not used)

**ACK**: ACK# is valid

**PSH**: push data now (generally not used)

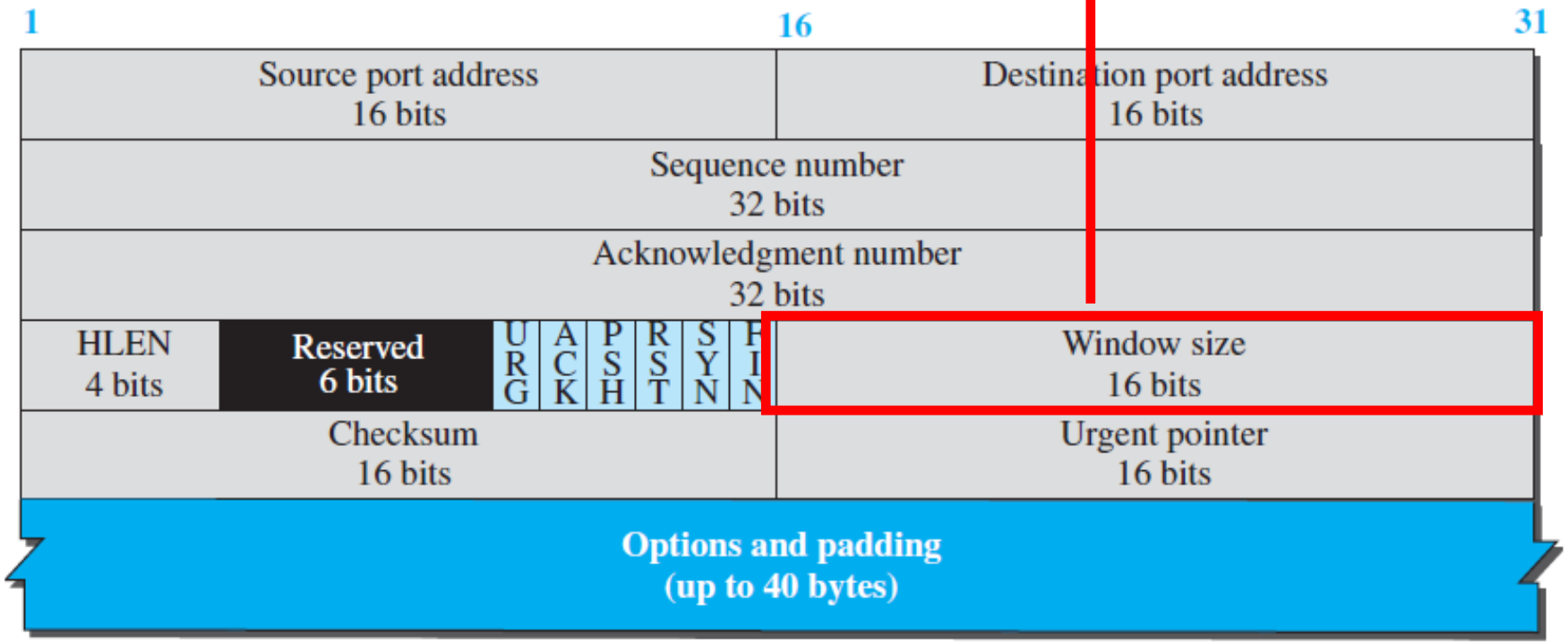
**RST, SYN, FIN**: connection establishment /setup and teardown




b. Header

# TCP Segment – Format

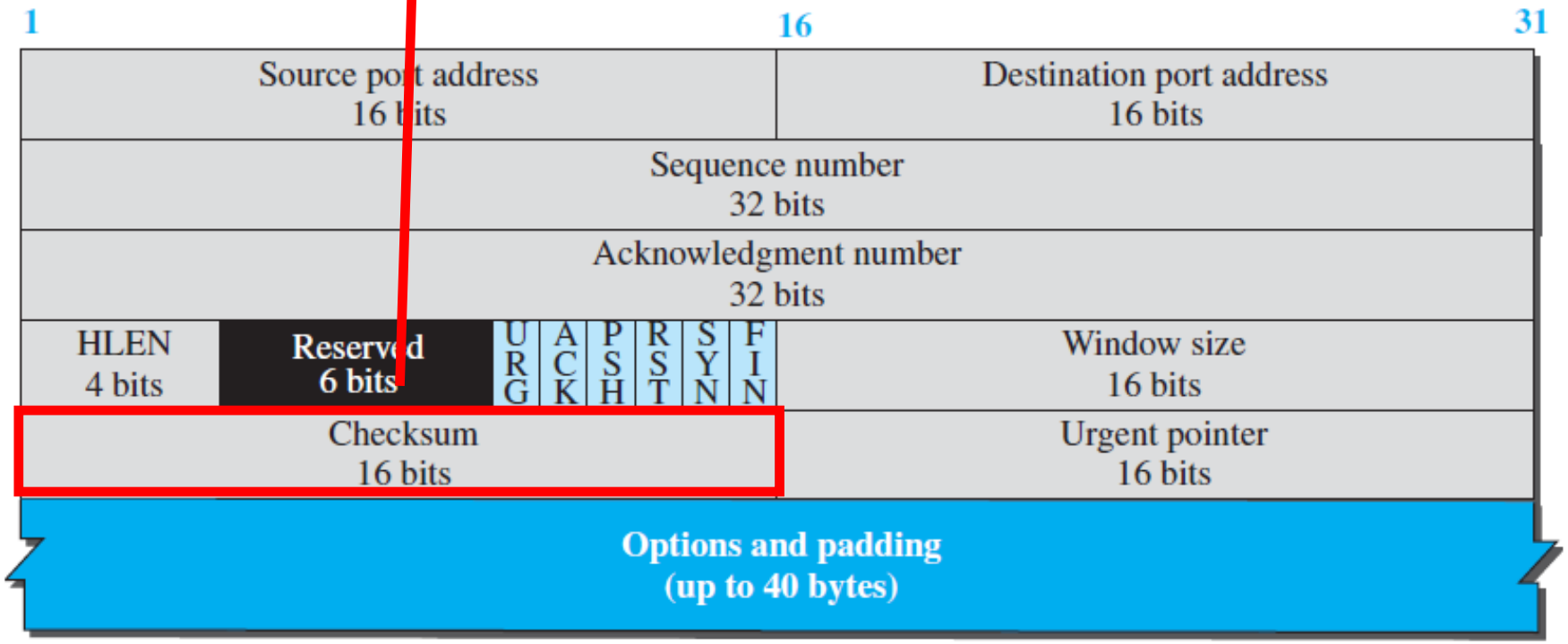
**Receiving window size (rwnd) and determined by the receiver. It is used for flow control.**




b. Header

# TCP Segment – Format

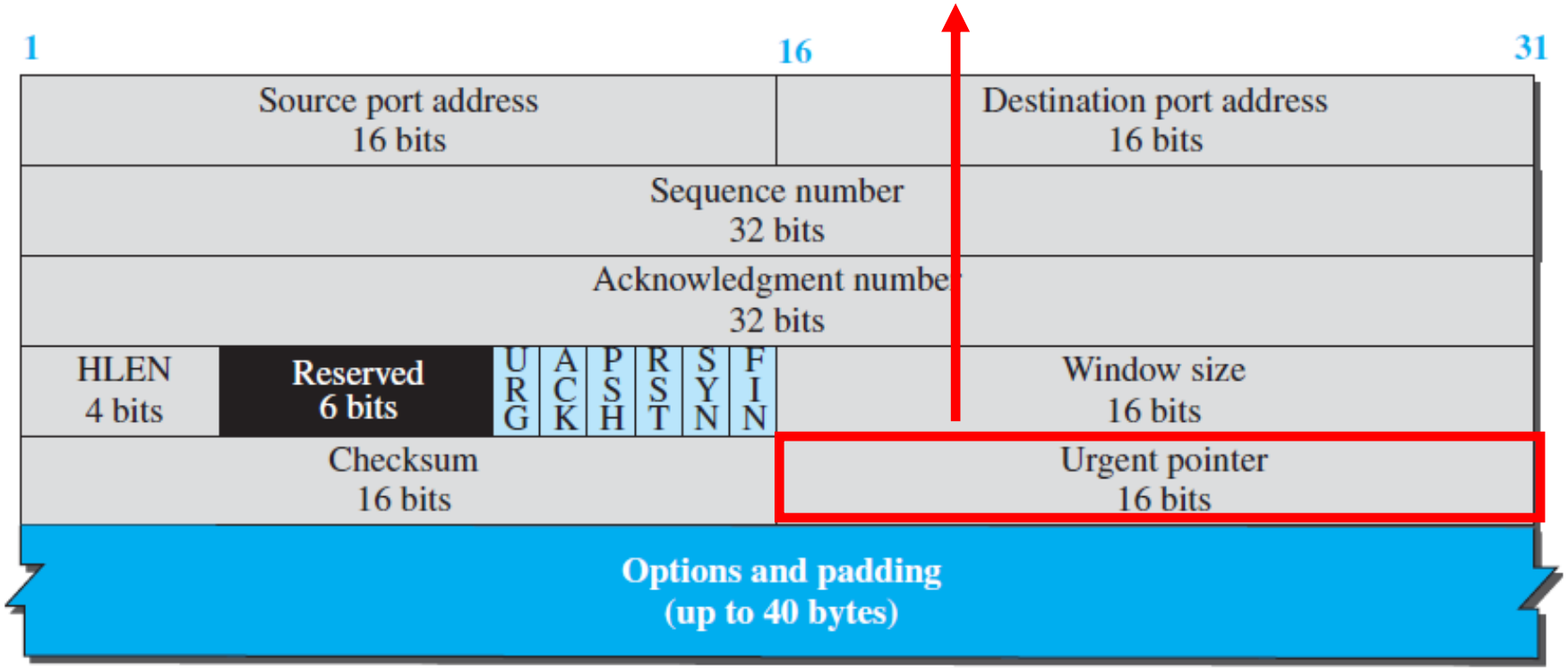
**Similar to UDP checksum calculation, i.e., it considers a pseudo header as well (the use of checksum is mandatory in TCP).**




b. Header

# TCP Segment – Format

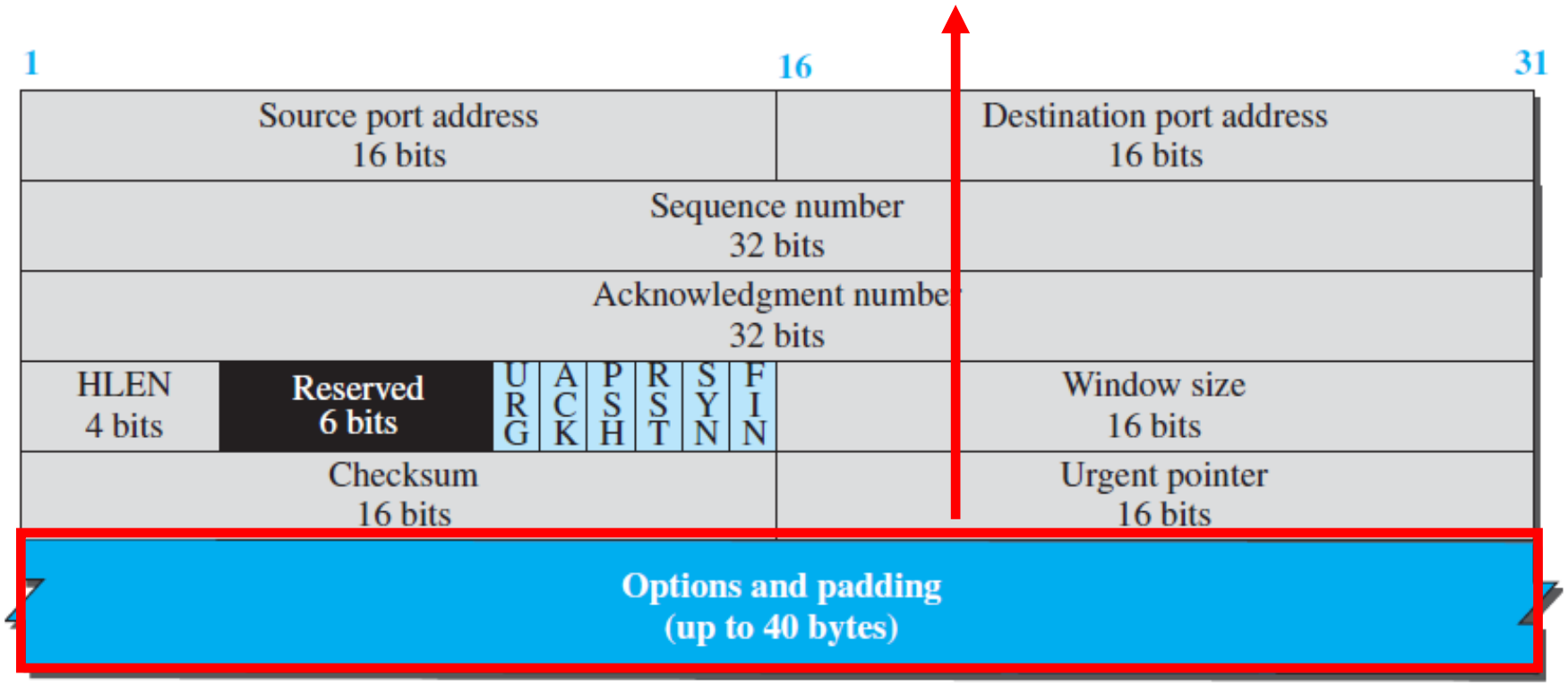
**Defines the end of the urgent data.**




b. Header

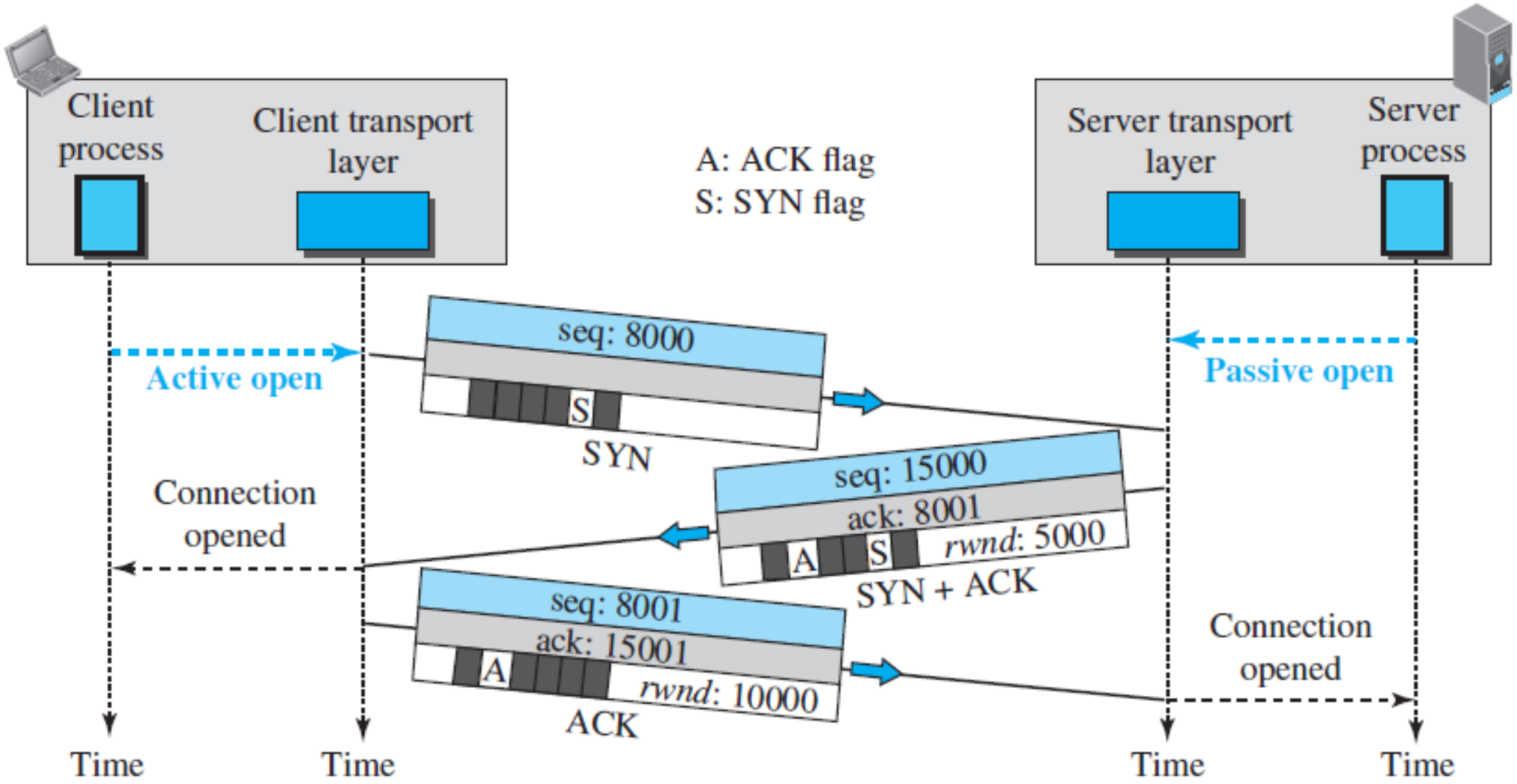
# TCP Segment – Format

**Optional field,** e.g., when a sender and receiver negotiate the maximum segment size (MSS)

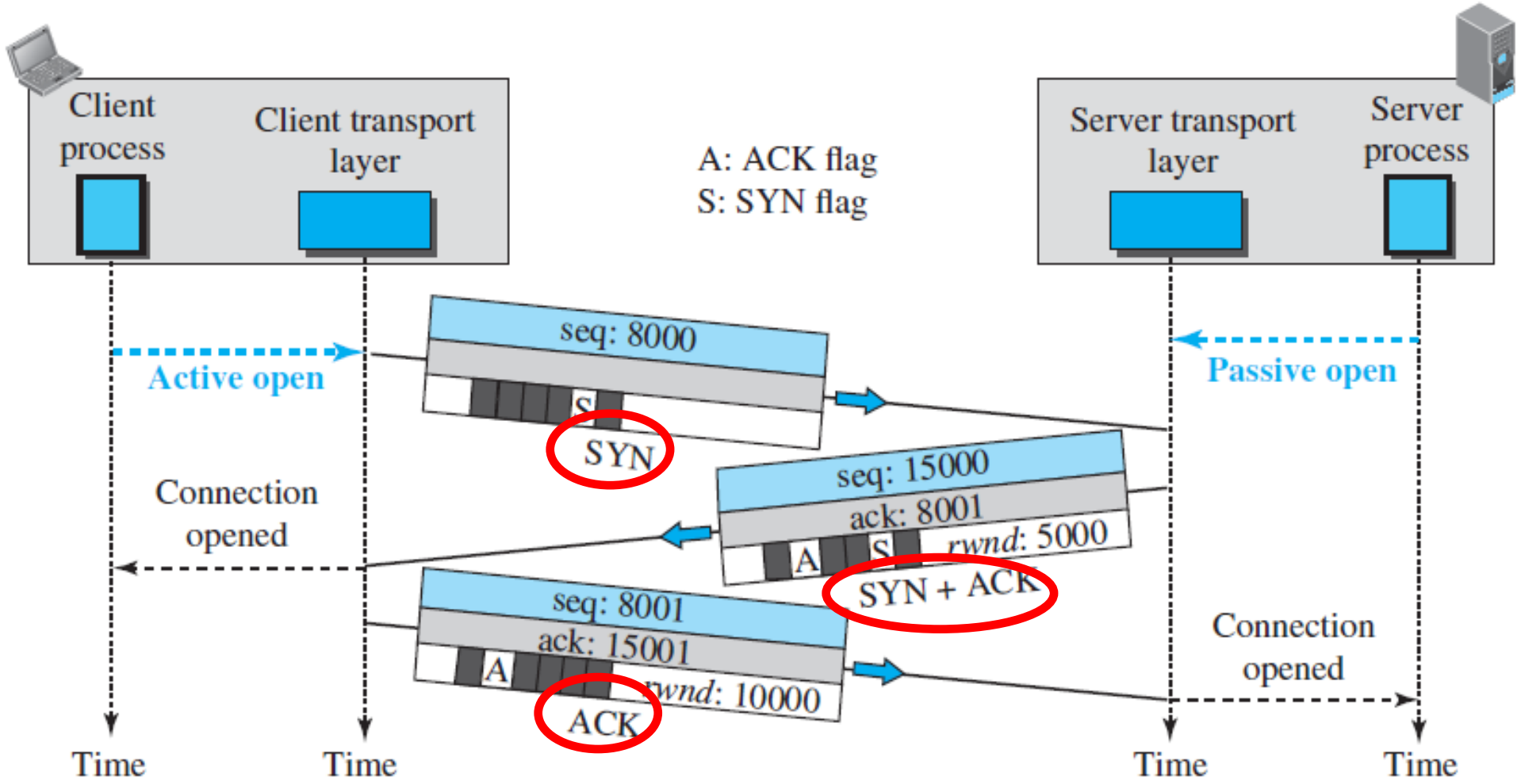



b. Header

# TCP – Connection Establishment



# TCP – Connection Establishment

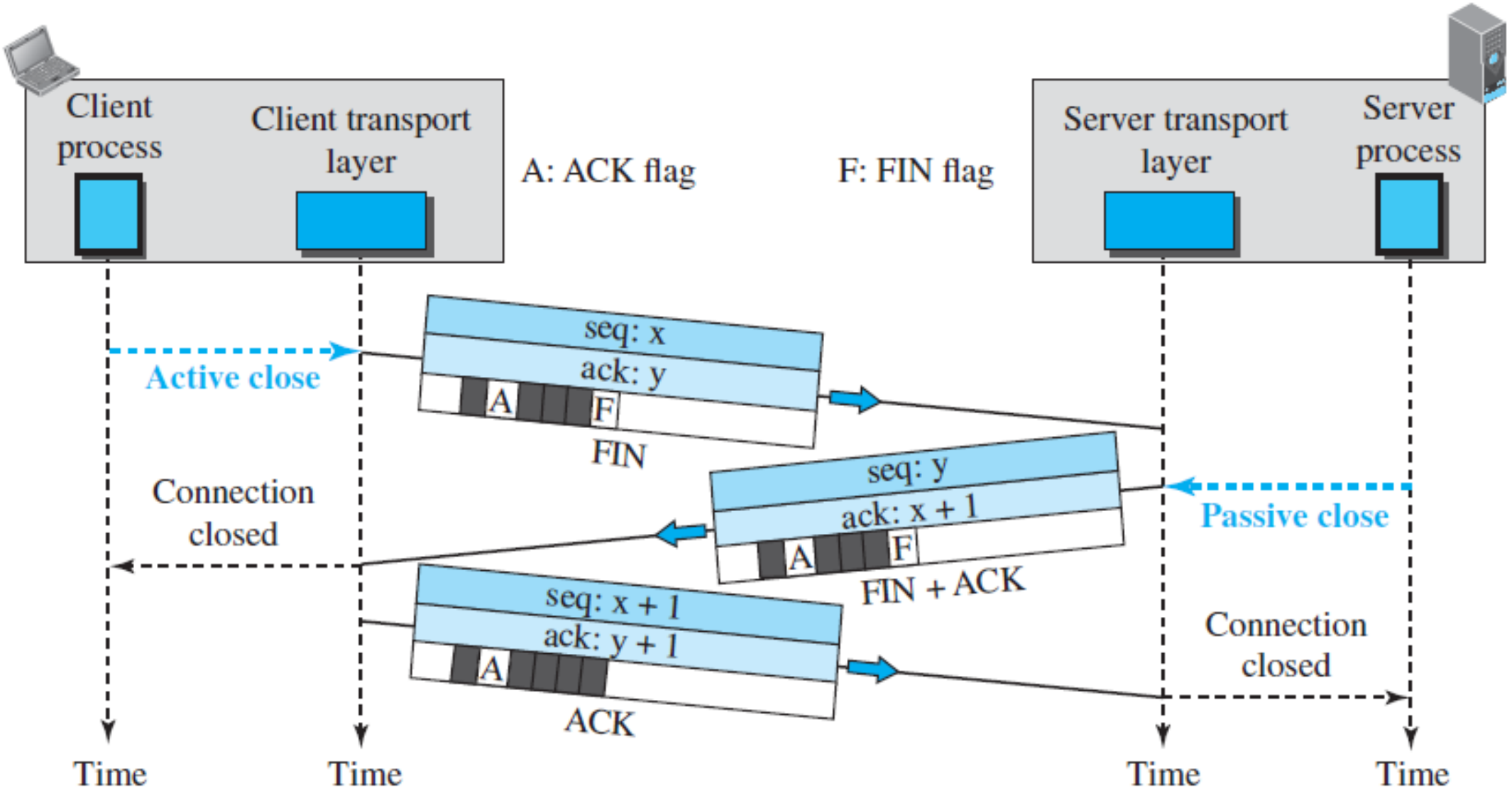



TCP connection-establishment procedure is often referred to as a **three-way handshake**.

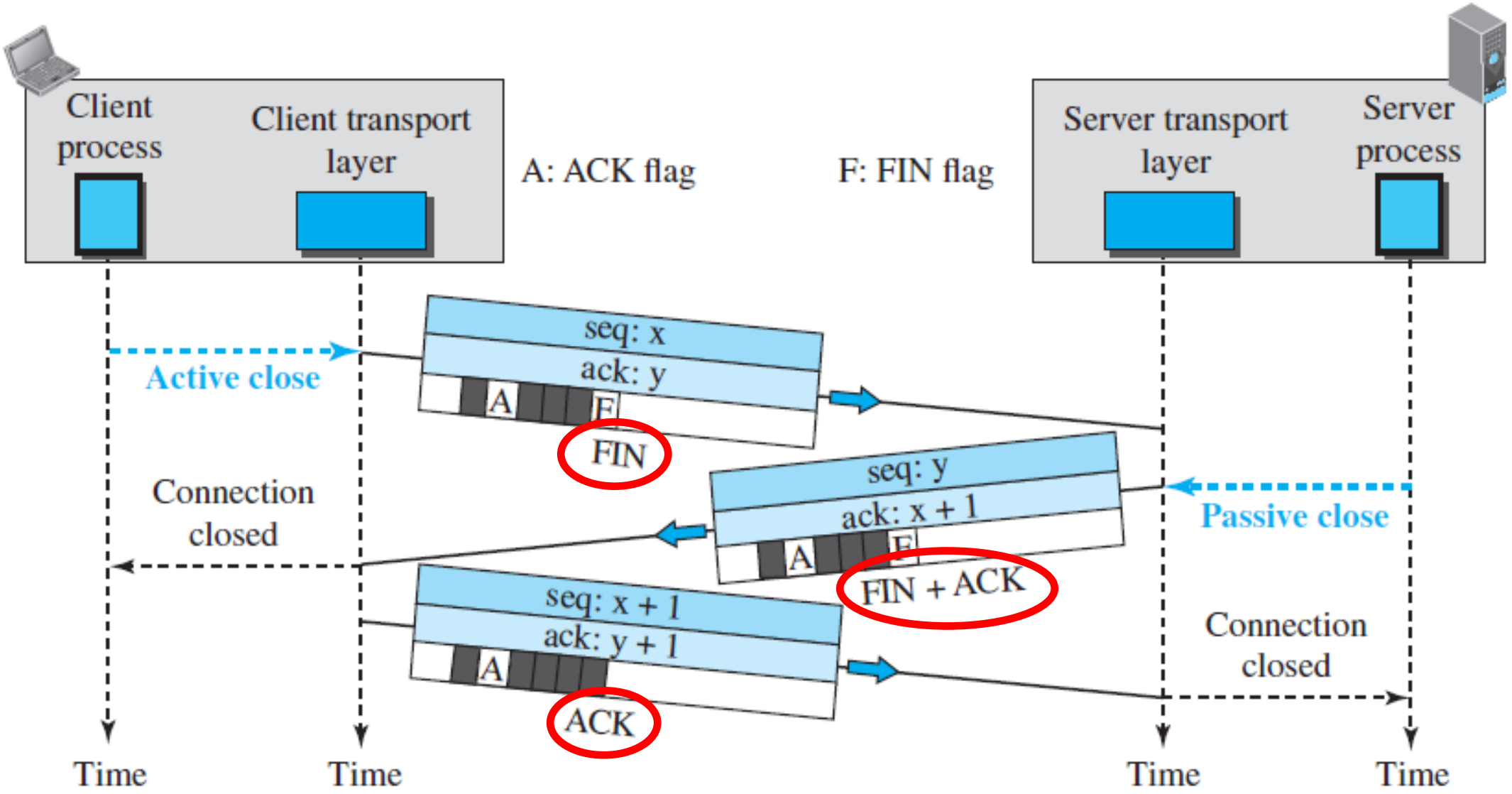
# TCP – Connection Termination

- Either of the two parties involved in exchanging data (client or server) can close the connection, although it is usually initiated by the client.
- Most implementations today allow **two options for connection termination**:
  1) three-way handshaking
  2) four-way handshaking with a half-close option

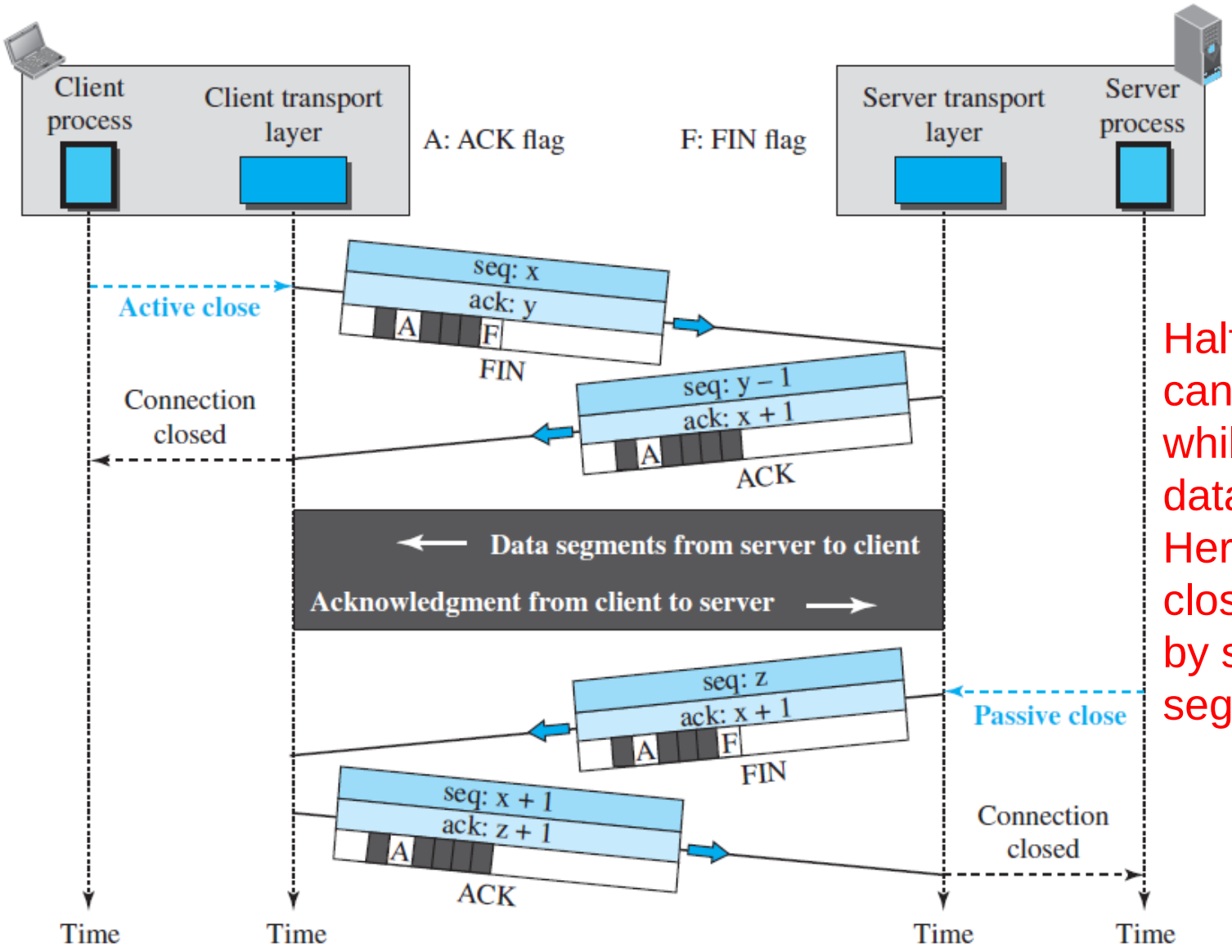
# TCP – Connection Termination (Three-way Handshaking)



# TCP – Connection Termination (Three-way Handshaking)



# TCP – Connection Termination (Half-Close)



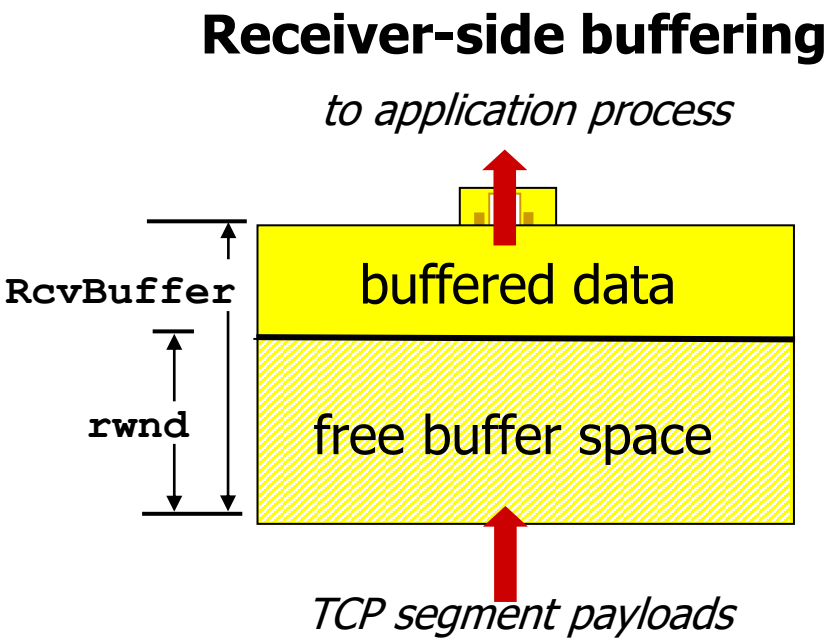

Half-close: one end can stop sending data while still receiving data.
Here, the client half-closes the connection by sending a FIN segment

# TCP – Flow Control

- Flow control is used to eliminate the possibility of the sender overflowing the receiver's buffer.
- TCP sender (at each side of the connection) maintains a variable called **receive window (rwnd).**
  - gives the sender an idea of how much free buffer space is available at the receiver.
  - initial value of of rwnd is equal to the receive buffer size (RcvBuffer)
  - LastByteSent – LastByteAcked <= rwnd
- At receiver: rwnd = RcvBuffer – [LastByteRcvd – LastByteRead]

# TCP flow control

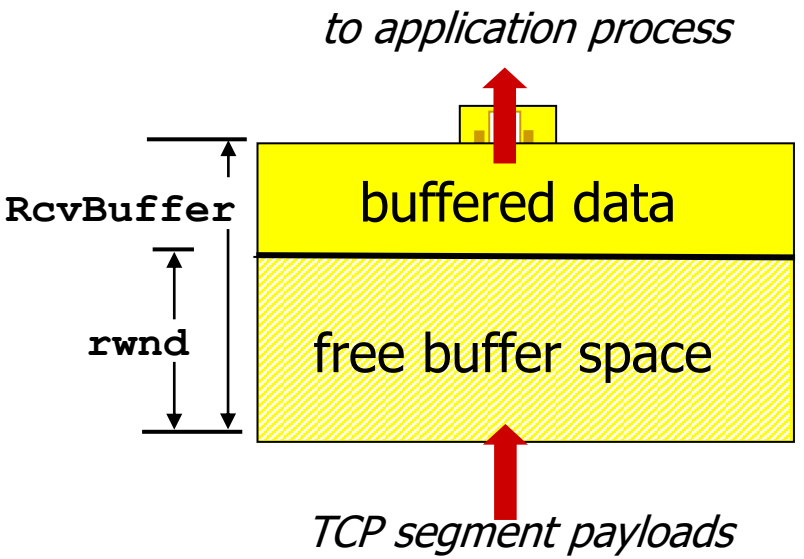
- Receiver “advertises” free buffer space by including **rwnd** value in TCP header of receiver-to-sender segments.
  - **RcvBuffer** size set via socket options (typical default is 4096 bytes)
  - many operating systems auto-adjust **RcvBuffer**



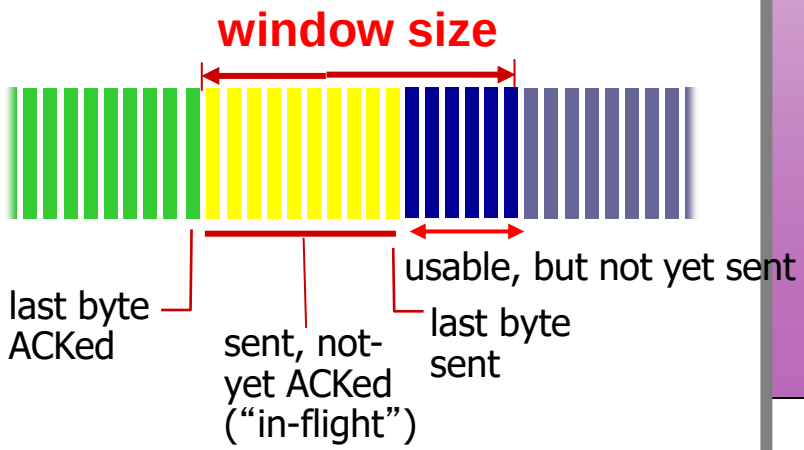
# TCP flow control

- Receiver "advertises" free buffer space by including **rwnd** value in TCP header of receiver-to-sender segments.
  - **RcvBuffer** size set via socket options (typical default is 4096 bytes)
  - many operating systems auto-adjust **RcvBuffer**
- Sender limits amount of unacked ("in-flight"or "outstanding") data to receiver's **rwnd** value (LastByteSent – LastByteAcked <= rwnd)
- Guarantees receive buffer will not overflow.

**Receiver-side buffering**




**Sender sequence number space**



# Error Control

- TCP is a reliable transport-layer protocol, i.e., 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.

# TCP Error Control Mechanisms

## 1) Retransmission

- the heart of TCP error control
- Retransmission after RTO (retransmission time-out)
  - ✓ the sending TCP maintains one RTO for each connection.
  - ✓ the value of RTO is dynamic and updated based on RTT (round-trip transmission time).
- Retransmission after Three Duplicate ACK Segments
  - ✓ this called fast retransmission.
  - ✓ if three duplicate acknowledgments (i.e., an original ACK plus three exactly identical copies) arrive for a segment, the next segment is retransmitted without waiting for the time-out.

# TCP Error Control Mechanisms – Continued

## 2) Checksum

- error detection, checking for a corrupted segment.

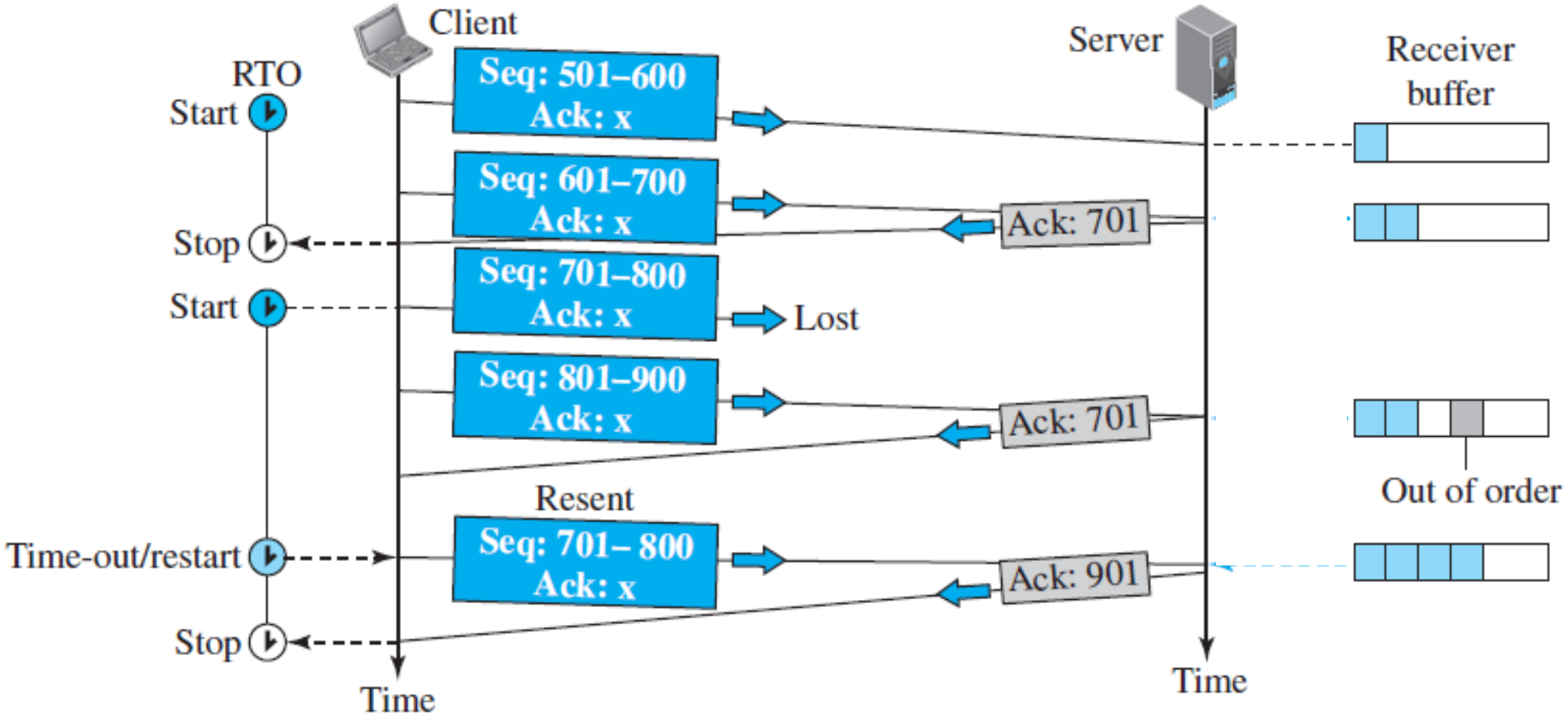
## 3) Acknowledgement

- confirms the receipt of data segments.

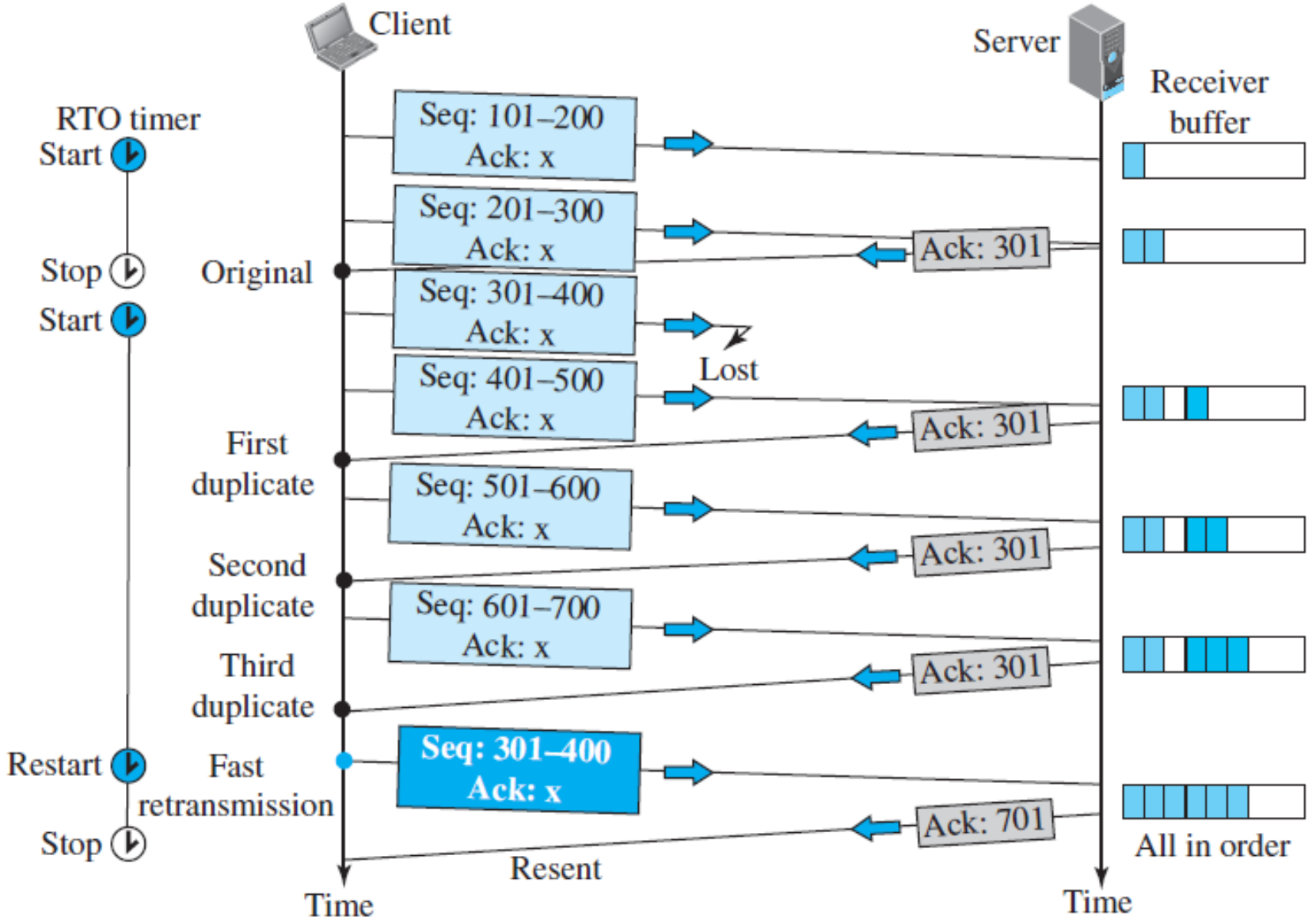
## 4) Sequence Number

- detection of a lost segment: gaps in the sequence numbers of the received segments.
- detection of a duplicate segment by receiver: received segments with duplicate sequence numbers.
- detection of out-of-order segments: the receiver flags the out-of-order segments and stores them temporarily until the missing segments arrive (TCP guarantees that data are delivered to the process in order).

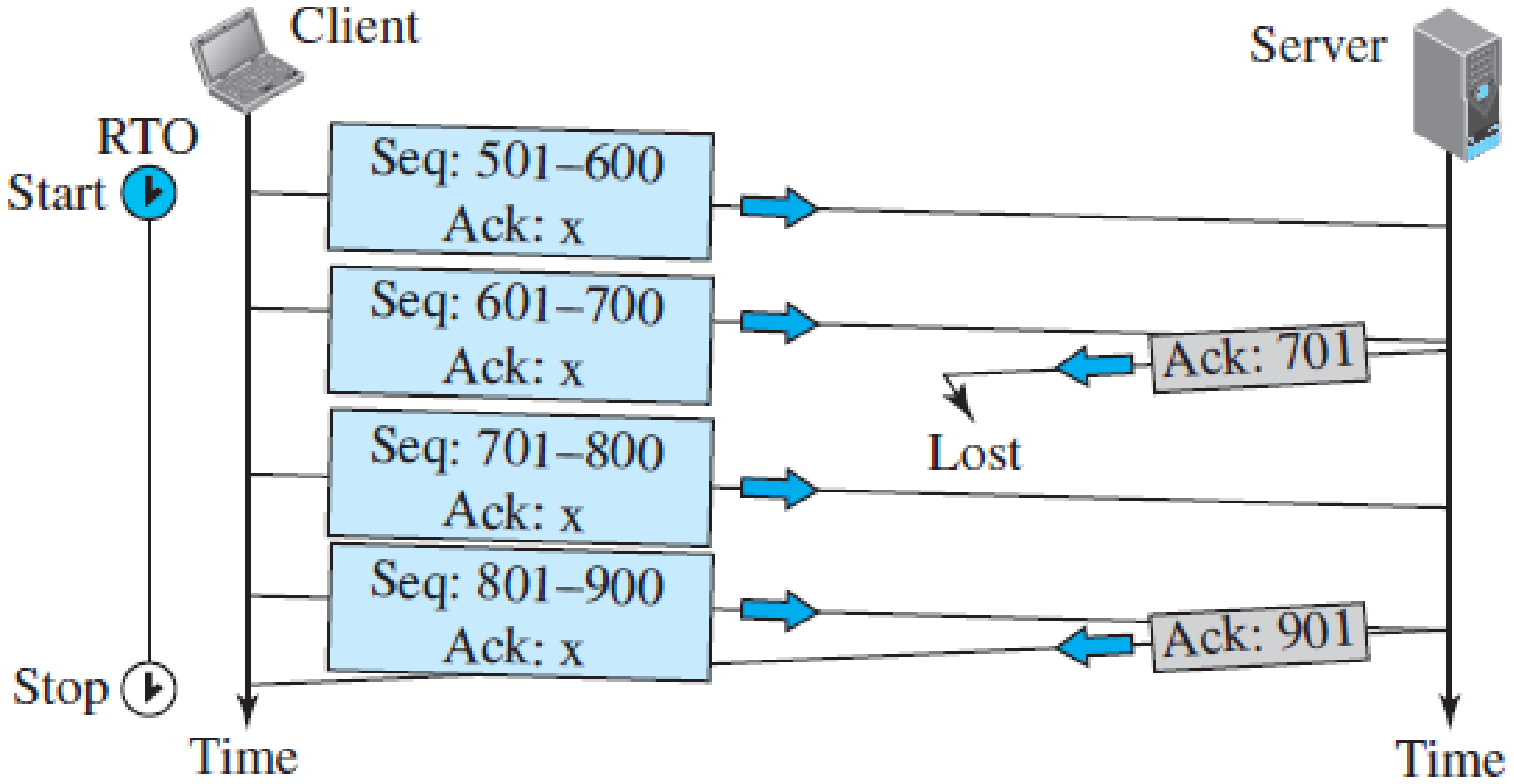
# TCP Error Control – Lost Segment



# TCP Error Control – Fast Retransmission

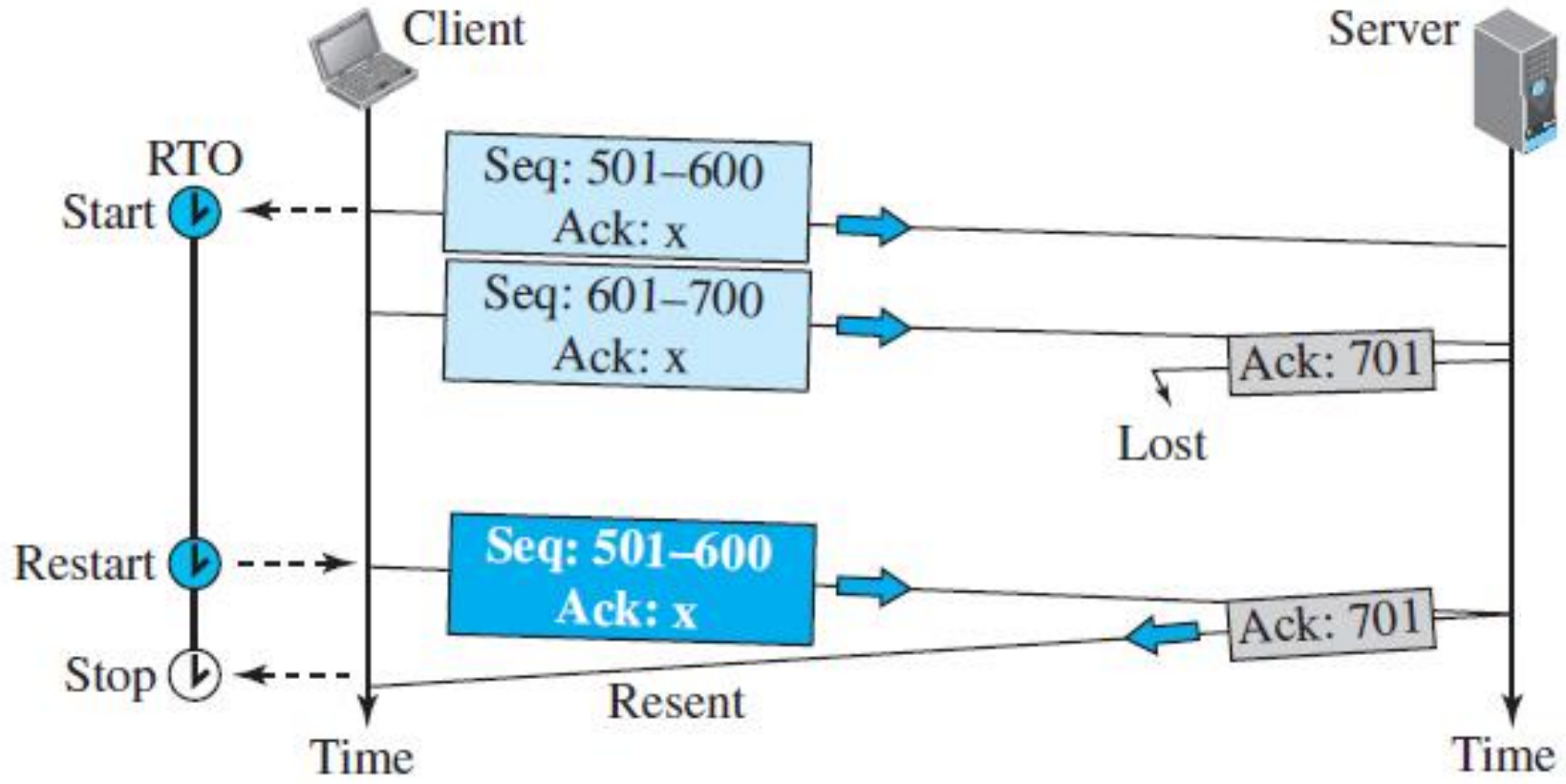


# TCP Error Control – Lost Acknowledgement




The next acknowledgment automatically corrects the loss of the previous acknowledgment (a key advantage of using cumulative acknowledgments).

# TCP Error Control – Lost Acknowledgement




Lost acknowledgment is corrected by resending a segment.

# TCP Congestion Control

- **TCP provides end-to-end congestion control.**
- **Congestion window (cwnd)**
  - imposes a constraint on the rate at which a TCP sender can send traffic into the network.
  - LastByteSent – LastByteAcked <= min{cwnd, rwnd}
  - the *cwnd* variable and the *rwnd* variable together define the size of the send window in TCP: **actual send window size = minimum (*rwnd*, *cwnd*)**

# TCP Congestion Detection

- **How a TCP sender can detect the possible existence of congestion in the network?**

# TCP Congestion Detection

- **How a TCP sender can detect the possible existence of congestion in the network?**
  - two events as signs of congestion in the network: time-out and receiving three duplicate ACKs

# TCP Congestion Detection

- **How a TCP sender can detect the possible existence of congestion in the network?**
  - two events as signs of congestion in the network: time-out and receiving three duplicate ACKs
- The TCP sender uses only one feedback from the other end to detect congestion: ACKs
  - the lack of regular, timely receipt of ACKs, which results in a time-out, is the sign of a strong congestion.
  - the receiving of three duplicate ACKs is the sign of a weak congestion in the network.

# TCP – MSS

- **What is Maximum Segment Size (MSS)?**
  - the maximum amount of application layer data in the segment (not the maximum size of TCP segment including headers)
  - the MSS is a value negotiated during the connection establishment, using an option of the same name (in the TCP header)

# TCP Congestion Control Algorithm

- **Three main components**
  1) Slow start
  2) Congestion Avoidance
  3) Fast Recovery

} mandatory

# TCP Congestion Control – Slow Start: Exponential Increase

- The size of the congestion window increases exponentially until it reaches a threshold (called SS: Slow Start threshold).

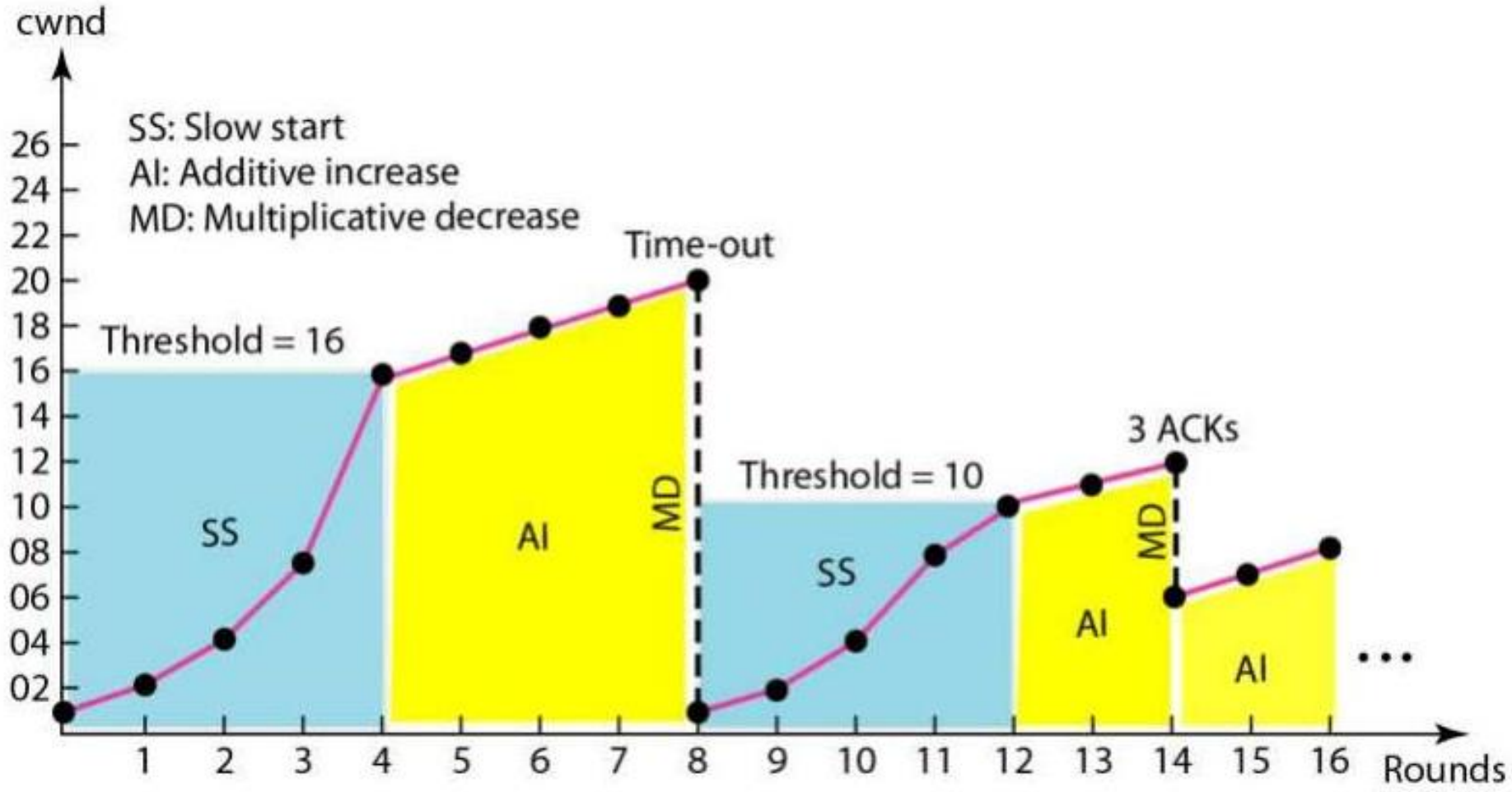
# TCP Congestion Control – Congestion Avoidance: Additive Increase

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

# TCP Congestion Control – Fast Recovery

- Fast recovery is optional.
- New versions of TCP use it.
- It starts when three duplicate ACKs arrive (light congestion).

# TCP Congestion Control – AIMD (Additive Increase Multiplicative Decrease)



# Summary

- **TCP**
  - TCP packets are called segments
  - reliable
  - connection-oriented
  - flow control (using sliding window)
  - error control (checksum, acknowledgement, sequence number, and time-out)
  - full-duplex
  - byte-oriented

# References

[1] Behrouz A.Forouzan, Data Communications and Networking, 5th Ed, 2013, McGraw-Hill companies.

[2] J. Kurose and K. Ross, Computer Networking: A Top-Down Approach, 7th Ed, 2017.