Chapter 12

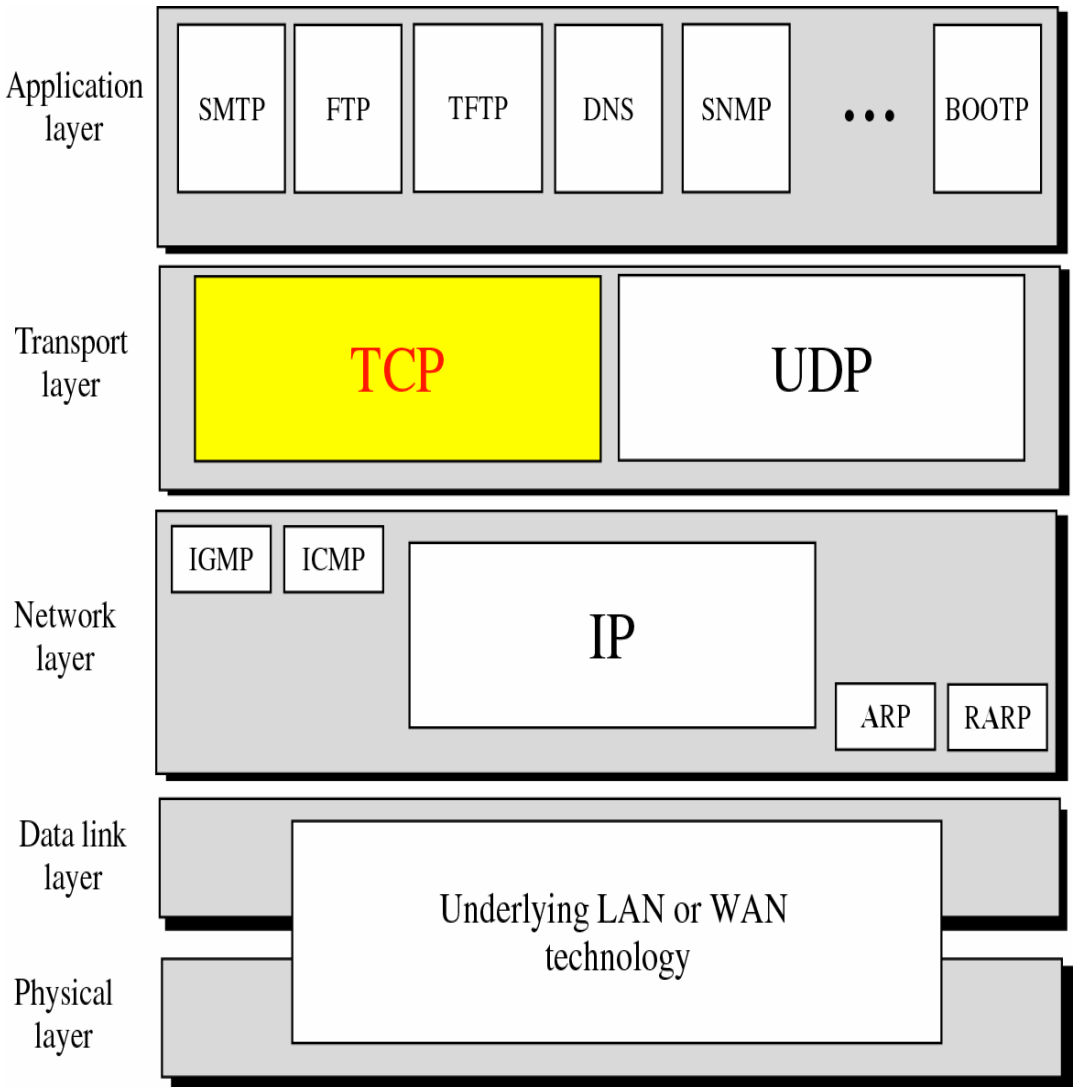
Transmission Control Protocol (TCP) – Part One

CONTENTS

Part Two

Part Two

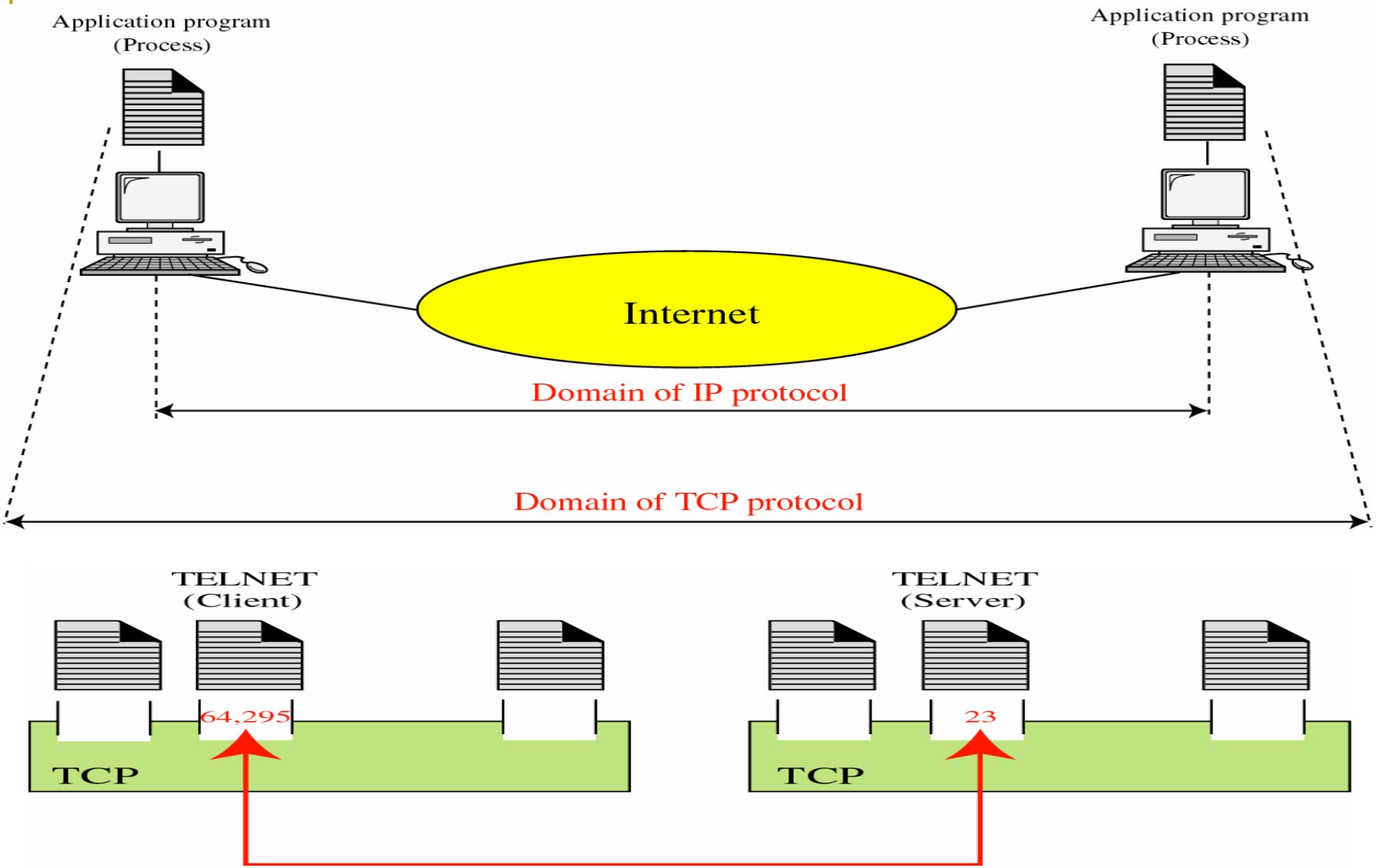
Position of TCP in TCP/IP protocol suite



TCP Provides:

- ✓ Process-to-Process Communication using *Ports*
- ✓ Flow Control using a *sliding window*
- ✓ Error control using *Acks, T-out, ReTrans*.
- ✓ Stream Transportation using *Connections*.

12.1 Process To Process Communication



12.2 TCP Services

1. Stream Delivery Service
   - Sending and Receiving Buffers
   - Segments

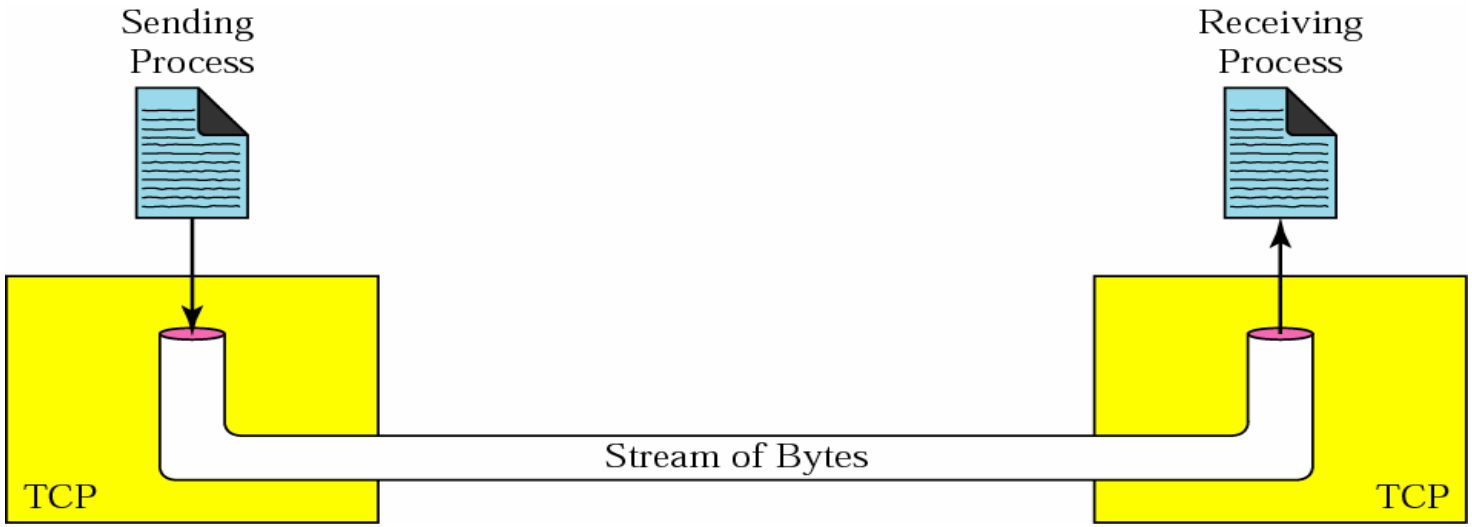
2. Full-Duplex Service:
   - Data can flow in both directions simultaneously

3. Connection-Oriented Service
   - Open Connection
   - Exchange Data
   - Close Connection

4. Reliable Service

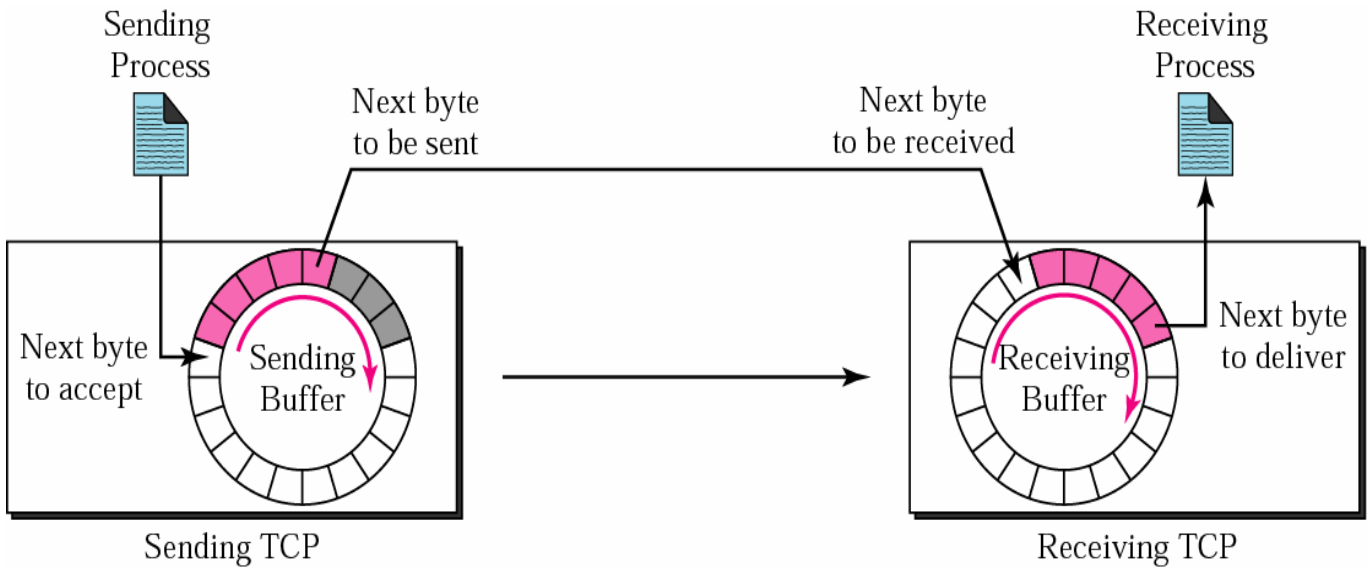
Stream delivery

Imaginary Tube

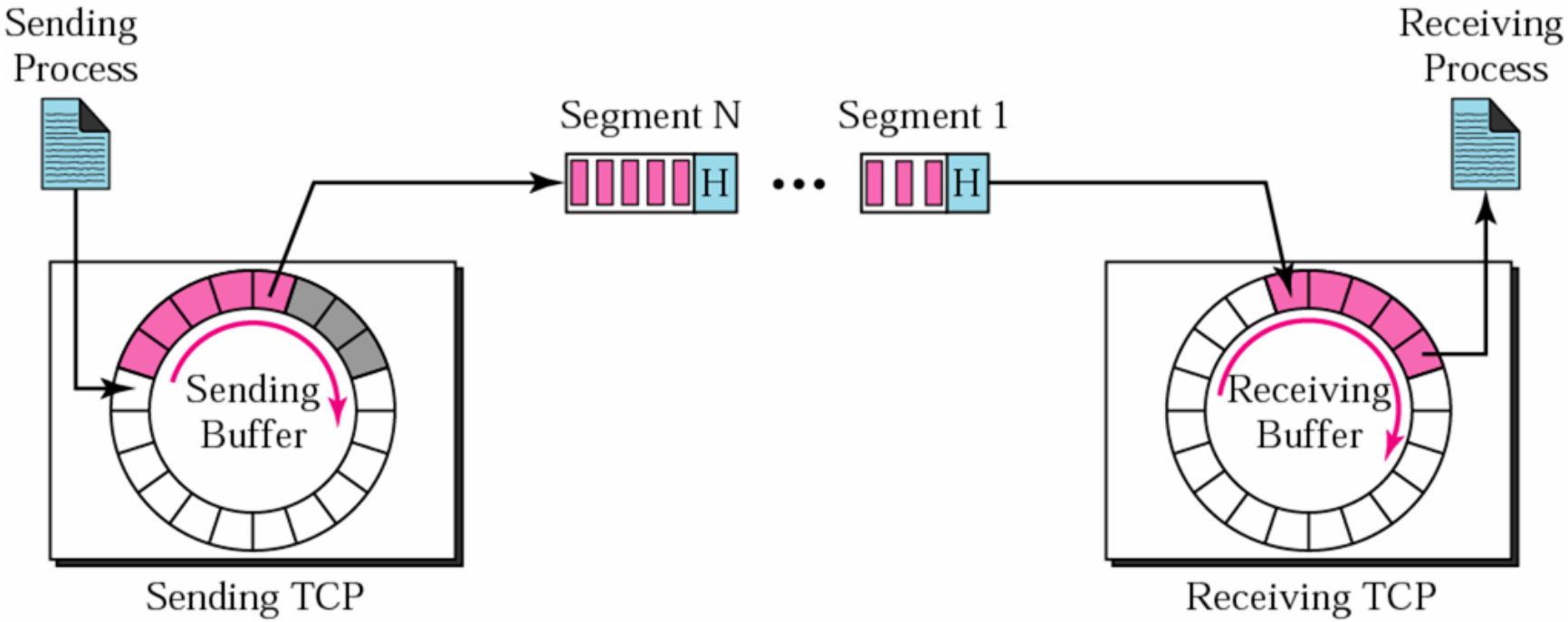


Use of Buffers
- **Speed disparity**
- **Flow Control**
- **Error Control**

One sending + one receiving buffer for each direction



TCP segments



- 1000's of bytes are grouped into a TCP *segment* with a header to be encapsulated into an IP packet.
- Segments
  - need not be of same size
  - may arrive out of order, be corrupted, or even get lost altogether.

12.3 Numbering Bytes

- The bytes (*not segments*) of data being transferred in each connection are numbered by TCP.
- Numbering is independent in each direction
- The numbering starts with a randomly generated 32-bit number (not necessarily 0).
- Each segment header has a *sequence number* field whose value defines the number of the first data byte contained in that segment.
- Each segment header has an *acknowledgment number* field whose value, if valid, defines the number of the next byte a party expects to receive. The acknowledgment number is cumulative.

Example 1

Imagine a TCP connection is transferring a file of 6000 bytes. The first byte is numbered 10010.
What are the sequence numbers for each segment if data is sent in five segments with the first four segments carrying 1,000 bytes and the last segment carrying 2,000 bytes?

Solution

The following shows the sequence number for each segment:

Segment 1 ➔ `10,010` `(10,010 to 11,009)`

Segment 2 ➔ `11,010` `(11,010 to 12,009)`

Segment 3 ➔ `12,010` `(12,010 to 13,009)`

Segment 4 ➔ `13,010` `(13,010 to 14,009)`

Segment 5 ➔ `14,010` `(14,010 to 16,009)`

12.4 Flow Control

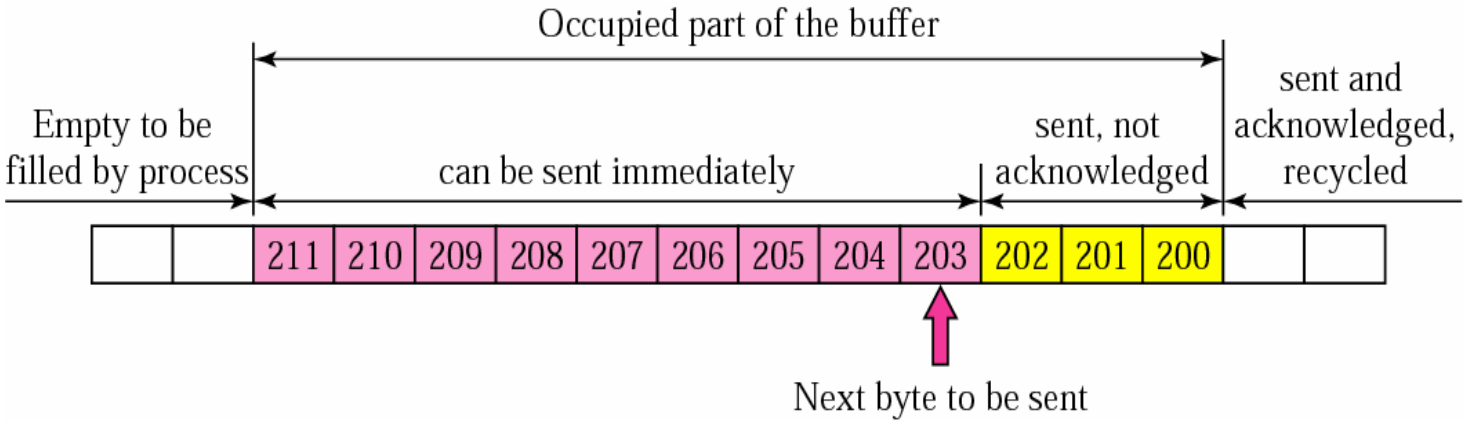
- How many bytes could be sent before waiting for an acknowledgement?

- A *sliding window* (for each connection) is imposed on the sending buffer to make transmission more efficient as well as to control the flow of data so that the destination does not become overwhelmed.

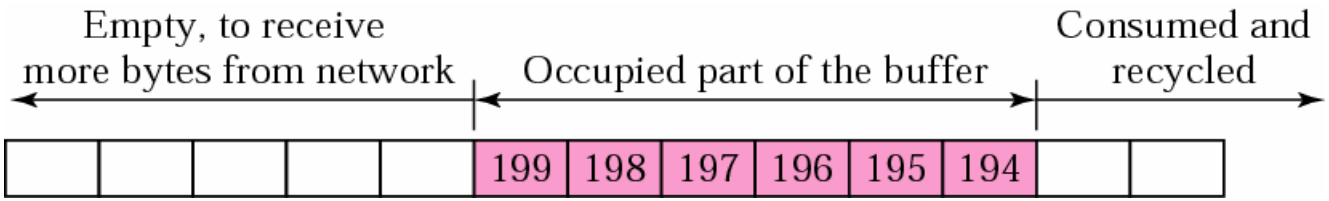
- A TCP's sliding window is byte oriented.

TCP Buffers

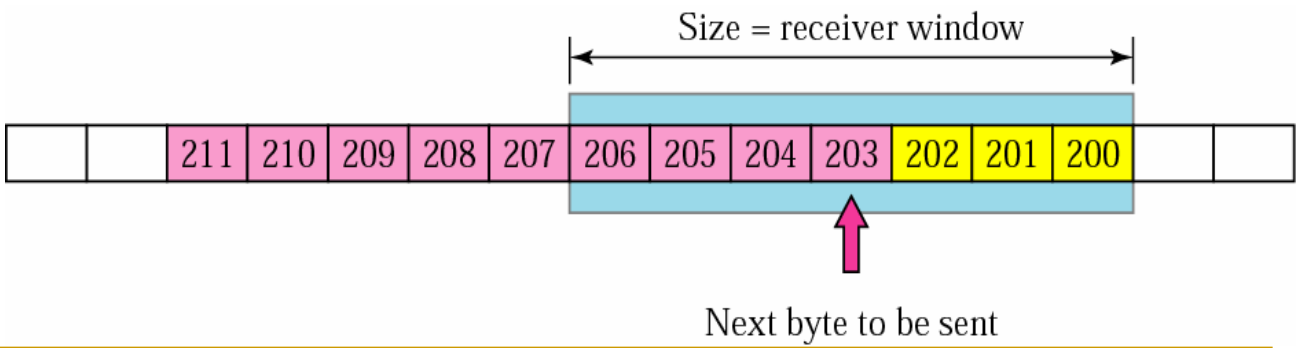
Sender buffer.
Could overflow
Receiver



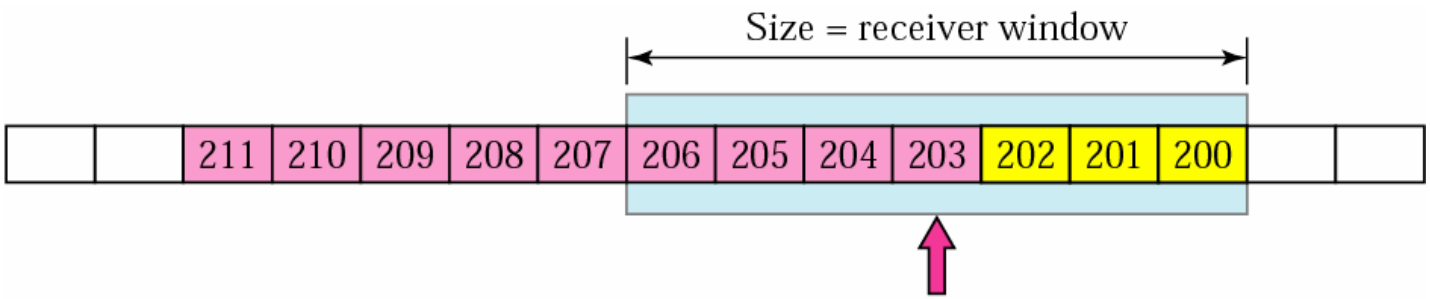
Receiver Window
= #of vacant buffers
= 7



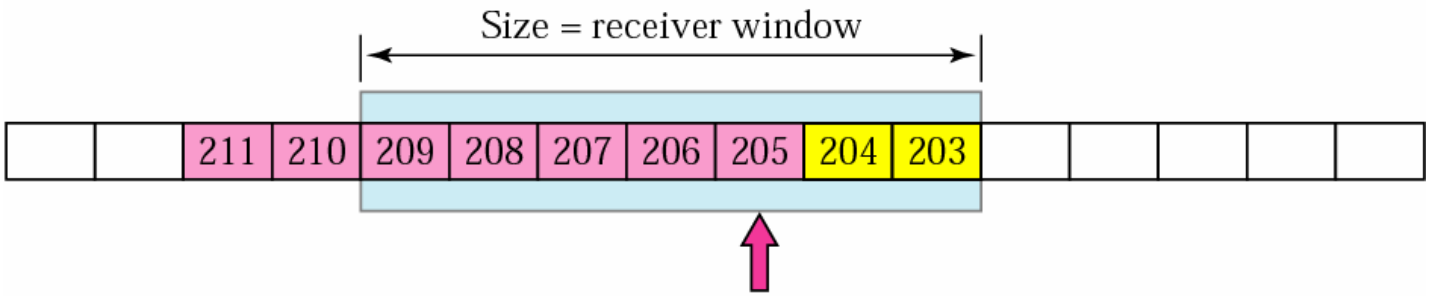
Sender window
≤ Receiver Window
= 7 bytes.
Yet 4 could be sent.



Sliding the sender window

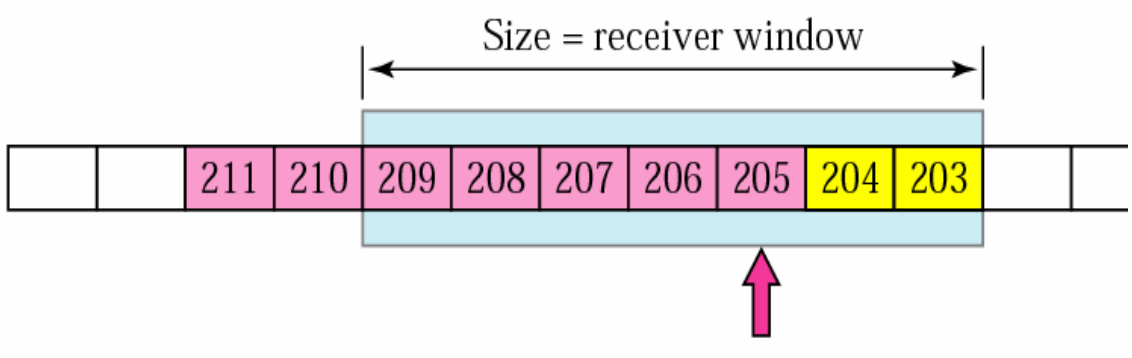


- Bytes 203 & 204 are sent
- Receivers Acks expecting byte 203 (*typo in textbook*) next, with its window size still 7



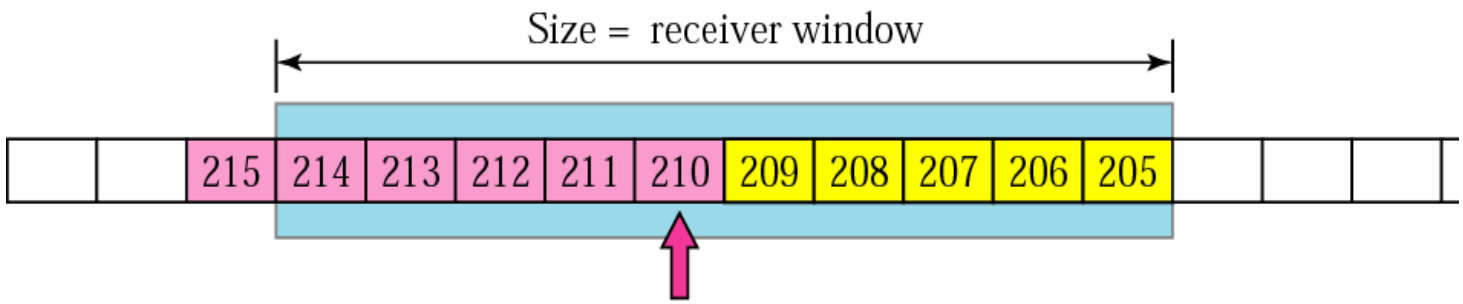
- Sender slides its window 3 position to the left

Expanding the sender window

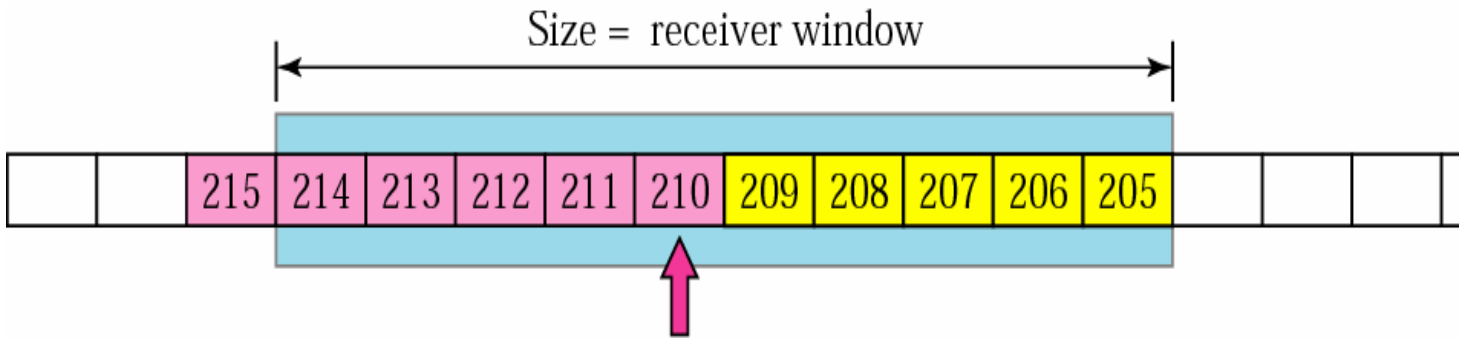


Expanding (Faster Consumption by Receiver)

- Receiver Acks two more bytes and increases window size to 10 (Why?)
- Sender expands its window and sends 5 more bytes, 4 more bytes arrive from Sending application

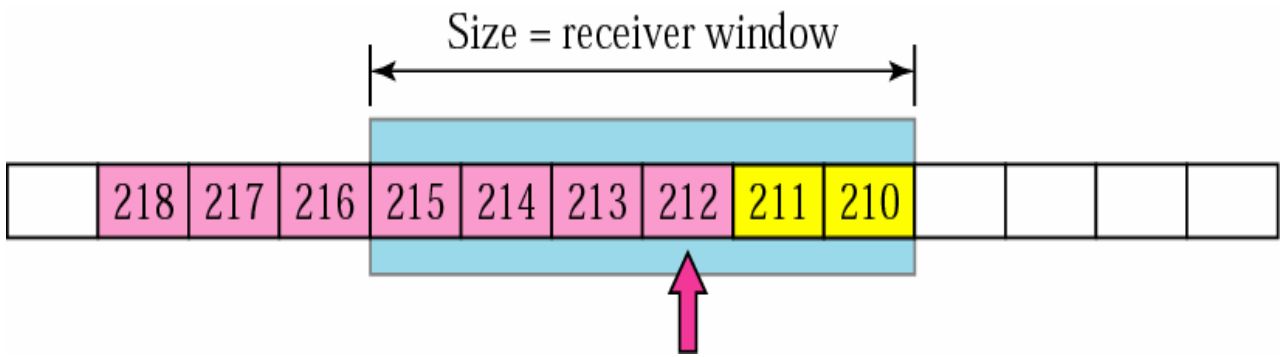


Shrinking the sender window



Shrinking (Slower Consumption by Receiver)

- Receiver Acks five more bytes, yet only one byte consumed, so it decreases window size to 6 (=10-5+1)
- Sender slides its window five positions to the left, shrinks its size to 6 and sends 2 more bytes. 3 bytes arrive from sending application



Remarks on the Sliding Window Protocol

- In TCP, the sender window size is totally controlled by the receiver window value. However, the actual window size can be smaller if there is congestion in the network.

- The Sender Window may be entirely closed (How? Why?) thus prohibiting any future transmission until further notice (How?).

- The source does not have to send a full window's worth of data.
- The destination can send an acknowledgment at any time.
- The Silly Window Syndrome could materialize.

12.5 Silly Window Syndrome

- What is it?
  - Only ONE byte of data is sent inside each TCP segment
  - A 41-byte IP packet (1 data + 20 TCP header + 20 IP header) carries one byte
- How does it occur?

1. By Sender: Application program is terribly slow; sends one byte at-a-time
   - Remedy? Nagle's algorithm:
   - Send first segment, even if only one byte of data
   - Subsequent segments are sent only when either
     - Previous segment is acknowledged (a fast network)
     - A Maximum-size (What is it?) segment worth of data has arrived from sending application (slow network)

Silly Window Syndrome "Continued"

- How does it occur?
  2. By Receiver: Application program is terribly slow; consumes one byte at-a-time. Eventually receiving buffer fills up.
     - Remedies?
       a) Clark's Solution: Ack ASAP with window size=0 until
          i) Half the buffer frees up
          ii) Enough space for a maximum-size (what is it?) segment frees up.
       b) Delayed Acknowledgement: Delay Ack until a decent amount of buffer frees up, but before, say, 500ms.
          - Reduce ACKs
          - Reduce retransmissions by sender

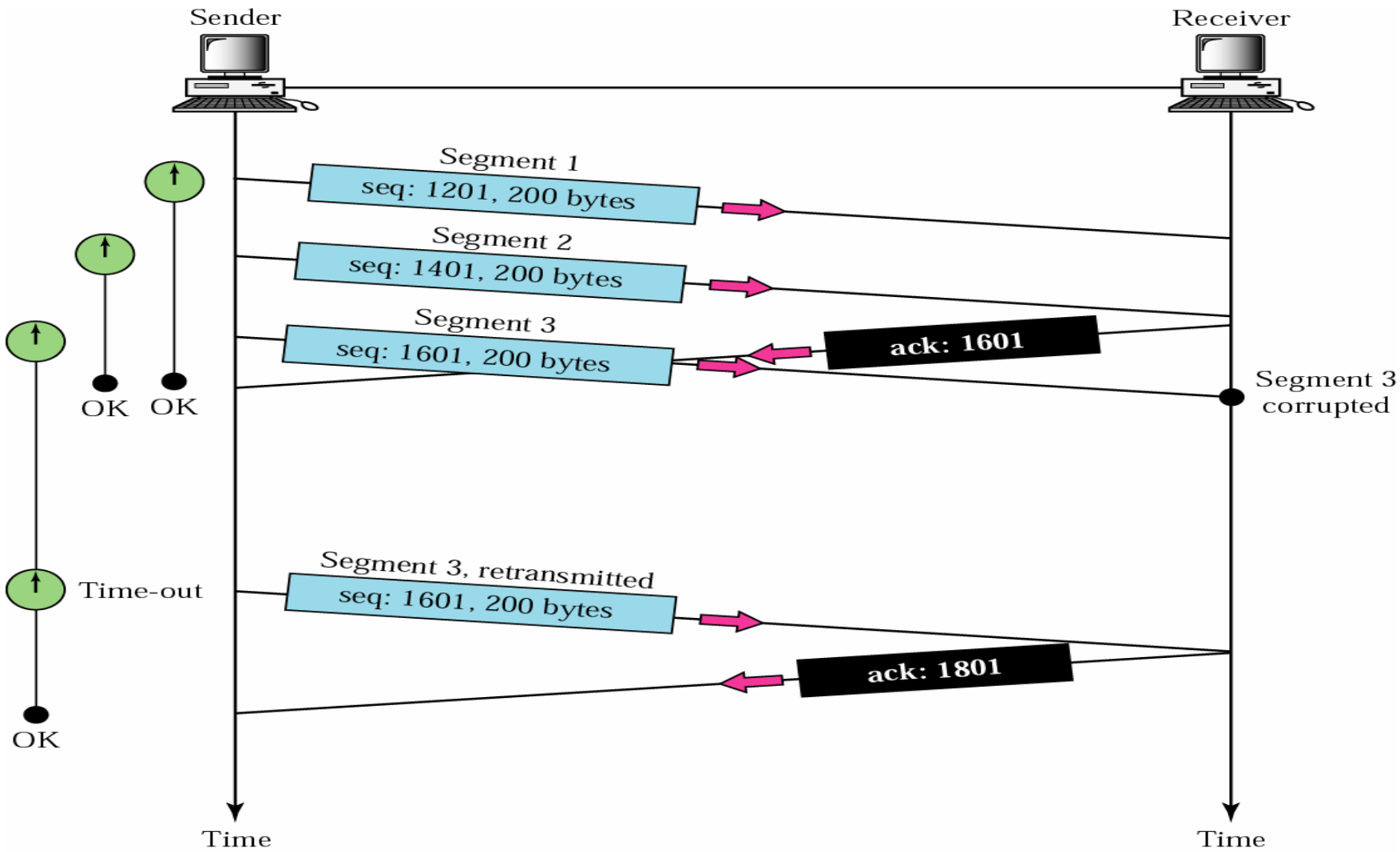
12.6 Error Control

- Deliver the data stream reliably:
  - All segments in order
  - No lost segments
  - No corrupted segments
  - No duplicated segments

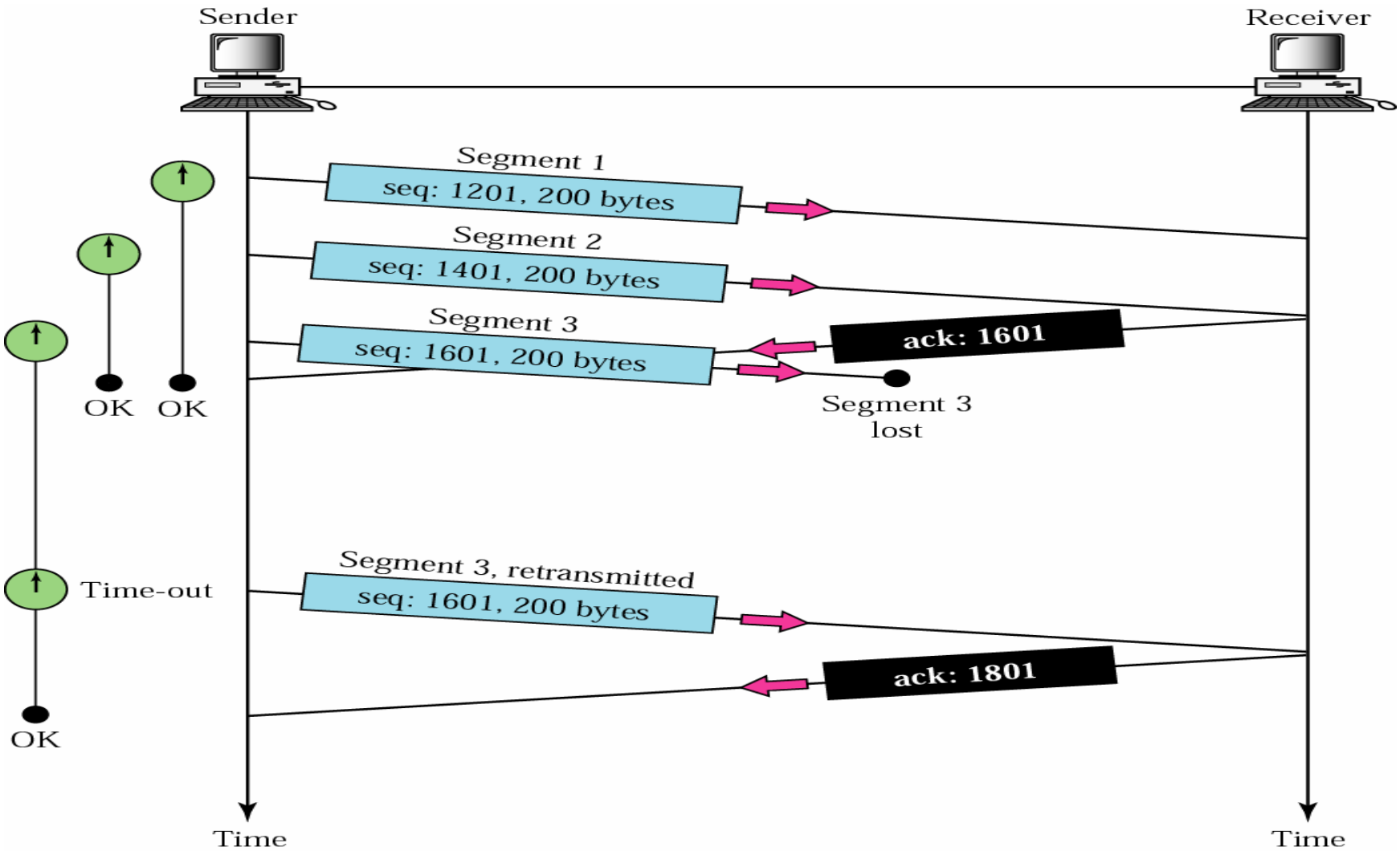
- Error control provides this level of reliability by detecting and correcting the above errors.

- Error Detection – using:
  - Checksums: detect corrupted segments
  - Acknowledgements: Confirm receipt of sound segments. No NACKs.
  - Timers: retransmit unacknowledged (due to loss or corruption) segments

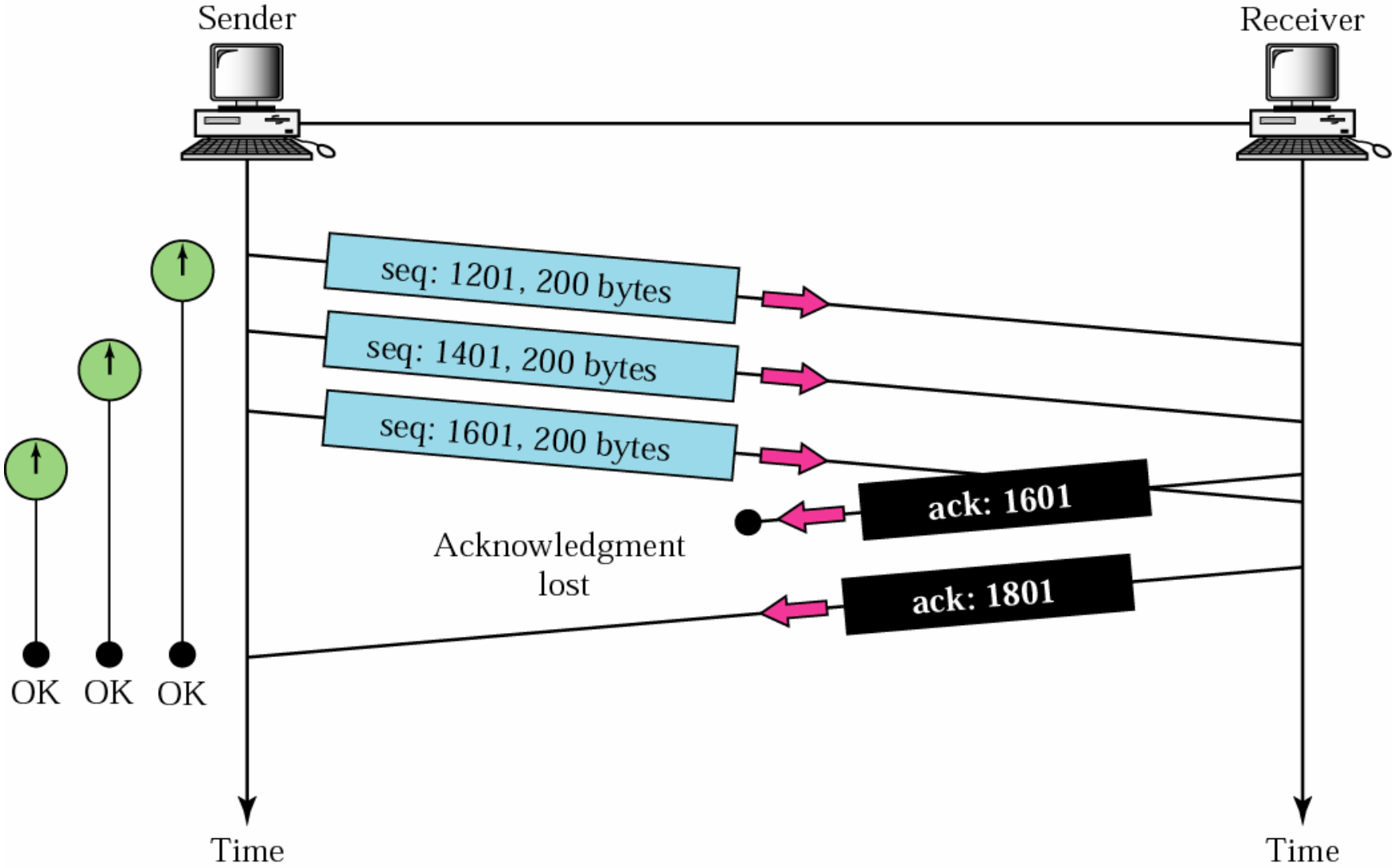
Corrupted segment



Lost segment



Lost acknowledgment

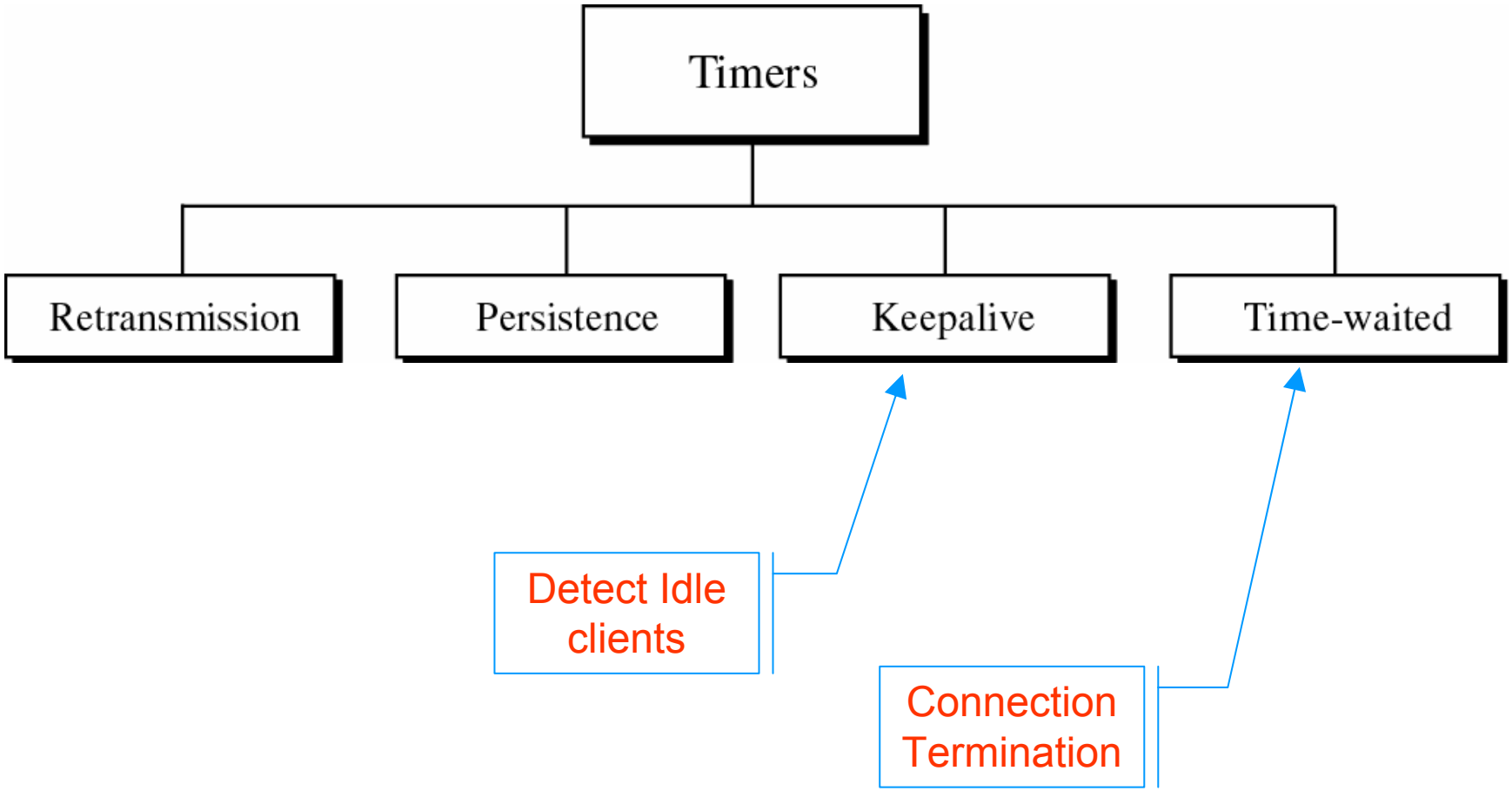


Duplication & Out-of-Order Arrival

- Duplication:
  - Detected by receiving a segment with a previously received sequence number. Action? Discard segment.

- Out-of-Order Segment:
  - Acknowledge a segment only if ALL preceding segments arrived safely.
  - If sender times-out and send duplicates, discard these duplicates.

12.7 TCP Timers



12.7.1 Retransmission Timer

- How long to wait for an ACK of a previously sent segment before retransmission.
- Depends on distance and network traffic density.
  - Retransmission time should be *dynamic*.
  - Retransmission time = 2 × RTT
- Dynamic Calculation of RTT:
  - Use a timestamp TCP option (discussed later), or
  a) Actually measure RTT of first two segments of a connection
  b) $RTT_{\text{future estimate}} = \alpha \times RTT_{\text{past}} + (1 - \alpha) \times RTT_{\text{current}}$
  c) Typically, $\alpha = 0.90$
  d) Do NOT consider retransmitted segments into the above calculation of RTT

12.7.2 Persistence Timer

- Sender receives an ACK with window size = 0
- Sender Stops transmission
- Receiver eventually has free buffer and sends an ACK with non-zero window size. This ACK segment is lost.
- Sender may wait forever. We have a deadlock

- Solution: Each time a sender gets a 0-window segment, it starts a *persistence* timer.
  - If timer goes off: probe the receiver to (re)update window size. (What is a probe and how is it interpreted by receiver? 5 bonus points for first two written responses with references)
  - If you get a non-zero window size from receiver cancel the persistence timer.