Data Link Flow Control - Recap (Sliding Windows)

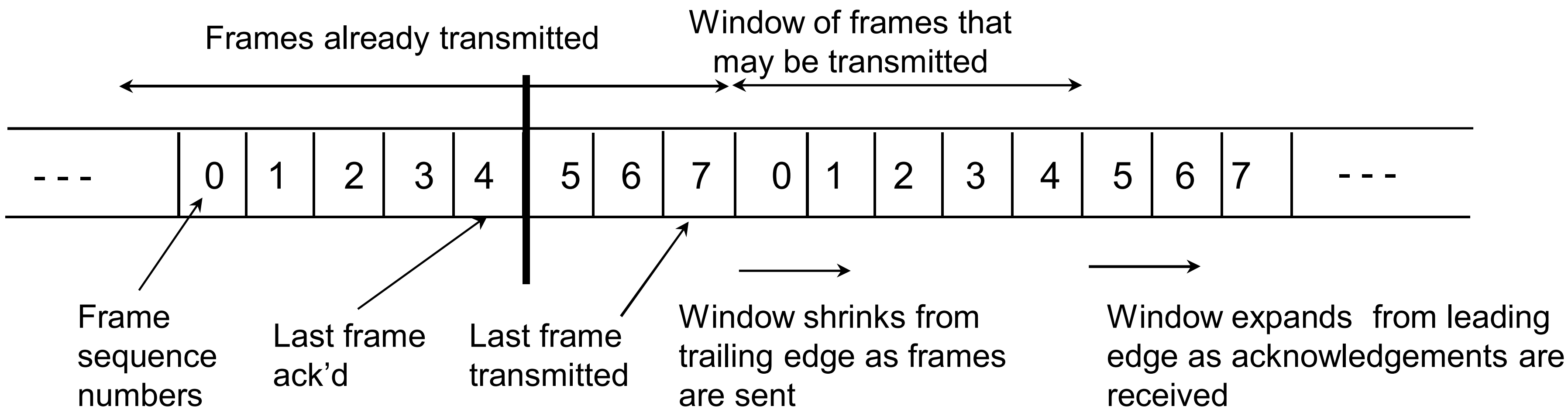
- This technique allows *multiple* frames to be in transit simultaneously
- Both stations use an extended buffer size to hold multiple frames
- The Sending/Receiving stations maintain a list of frames already sent/received
- This technique allows for much <u>more</u> *efficient link utilization*
- The transmission link is effectively treated as a *pipeline* that can be *filled* with many frames in transit *simultaneously*

Sliding Windows Flow Control

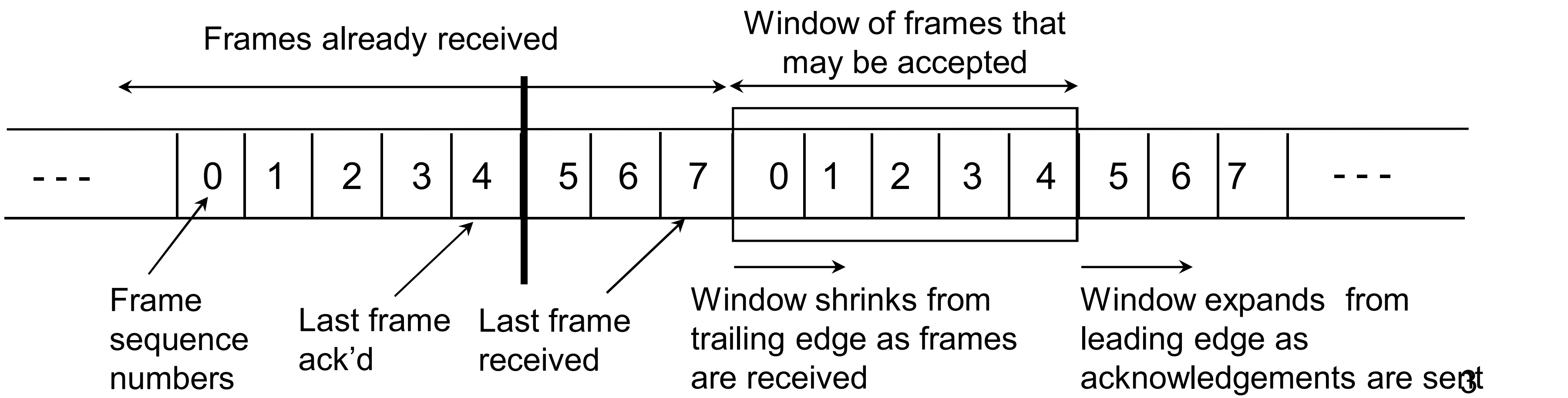
- Stations A and B each allocate buffer space for *W* frames
  - i.e. Station B can *accept W* frames and Station A can *send W* frames without any acknowledgement being sent or received
- Each frame contains a *sequence number*
- Station B sends *acknowledgements* that include the sequence number of the *next* frame expected
  - i.e. Station B is prepared to receive the next *W* frames starting at the *sequence number* indicated e.g. RR5

Sliding Windows Flow Control

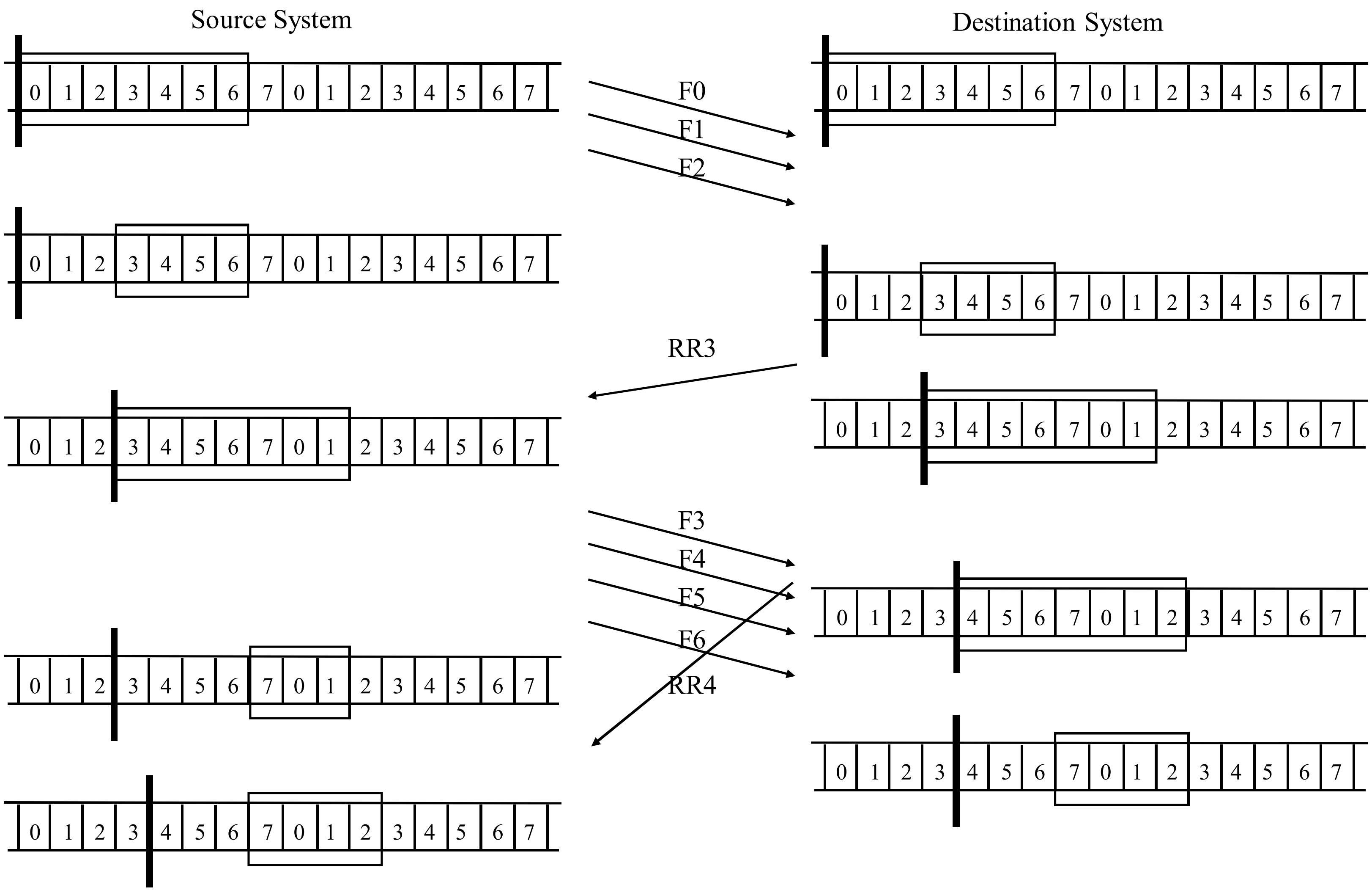
Transmitters Perspective



Receiver's Perspective



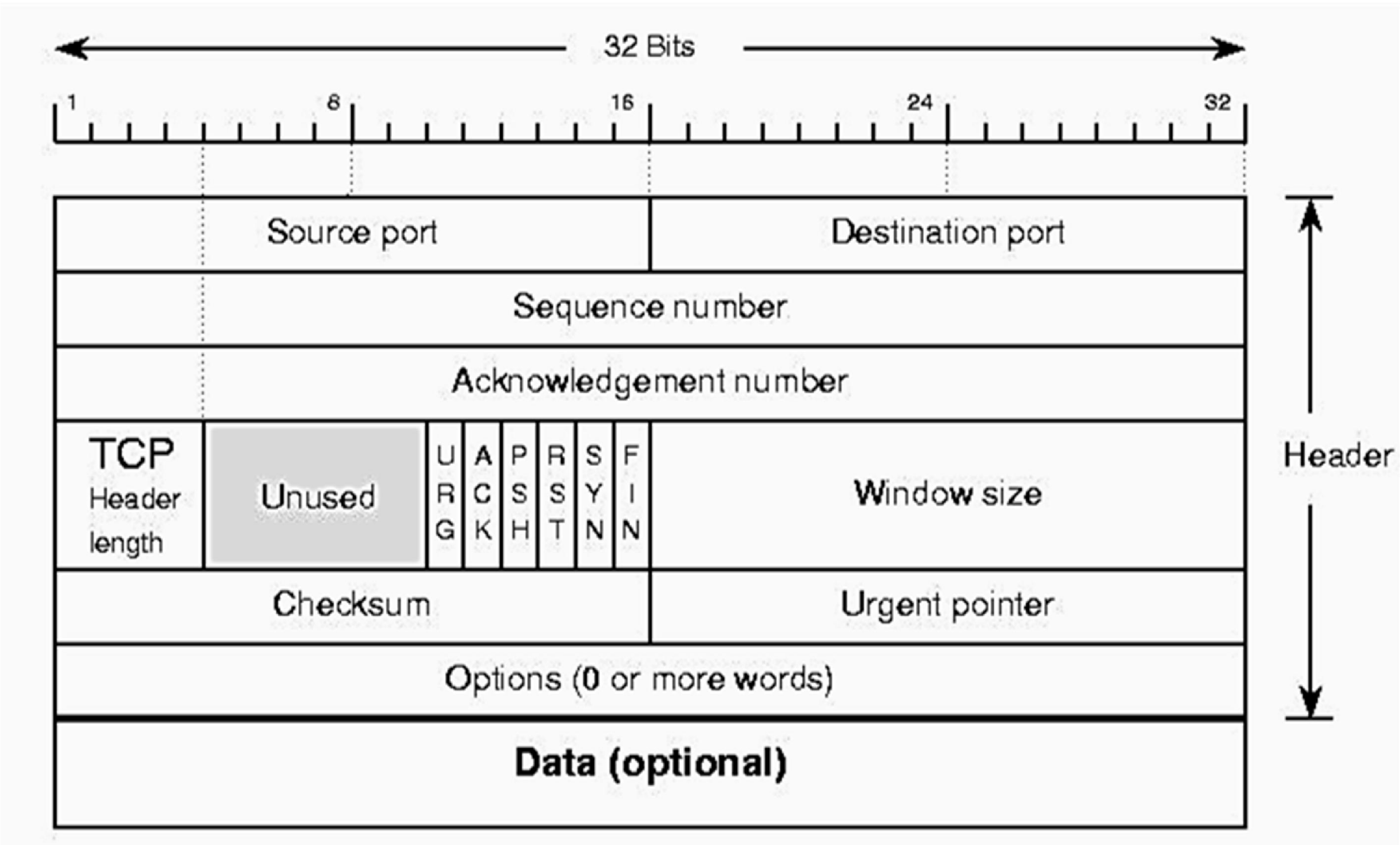
Example Sliding Windows



Sliding Windows Flow Control

- *Multiple* frames can be *acknowledged* using a single control message (*implicit acknowledgement*)
  - e.g. Receipt of ACK for frame **2** (RR3) followed later by ACK for frame **5** (RR6) *implies* acknowledgement of frames **3** and **4**
- Station A maintains a list of frame numbers it is allowed to *send*
- Station B maintains a list of frame numbers it is prepared to receive
- These lists can be considered as *windows*

TCP Segment Format



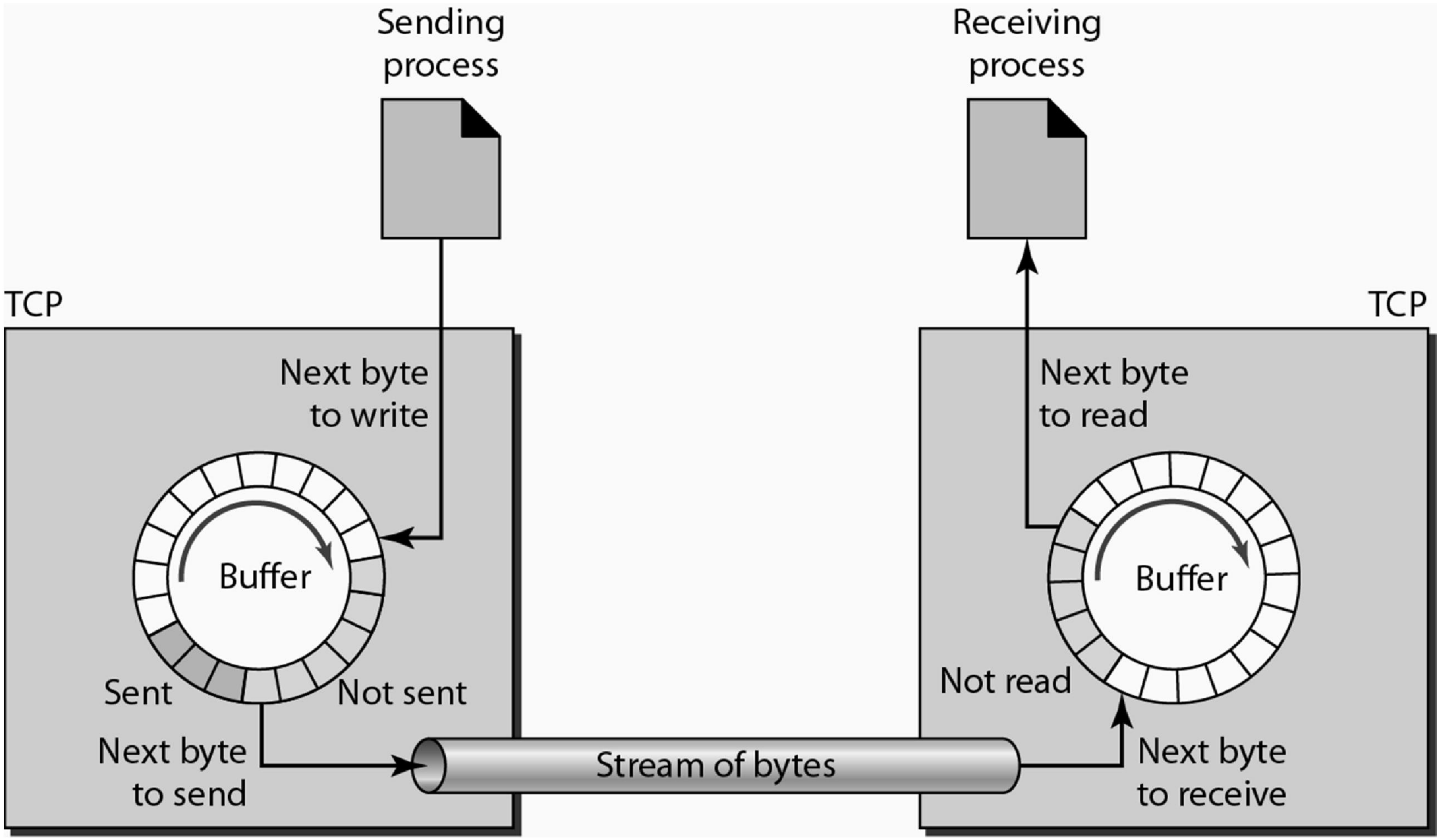
TCP Flow Control – *Buffers* and *Windows*

- Recall that TCP creates two buffers per socket:
  - One for incoming data (RECVQ)
  - One for outgoing data (SENDQ)
- The incoming buffers can easily overflow
- To prevent this the receiving TCP entity uses a ***window mechanism***

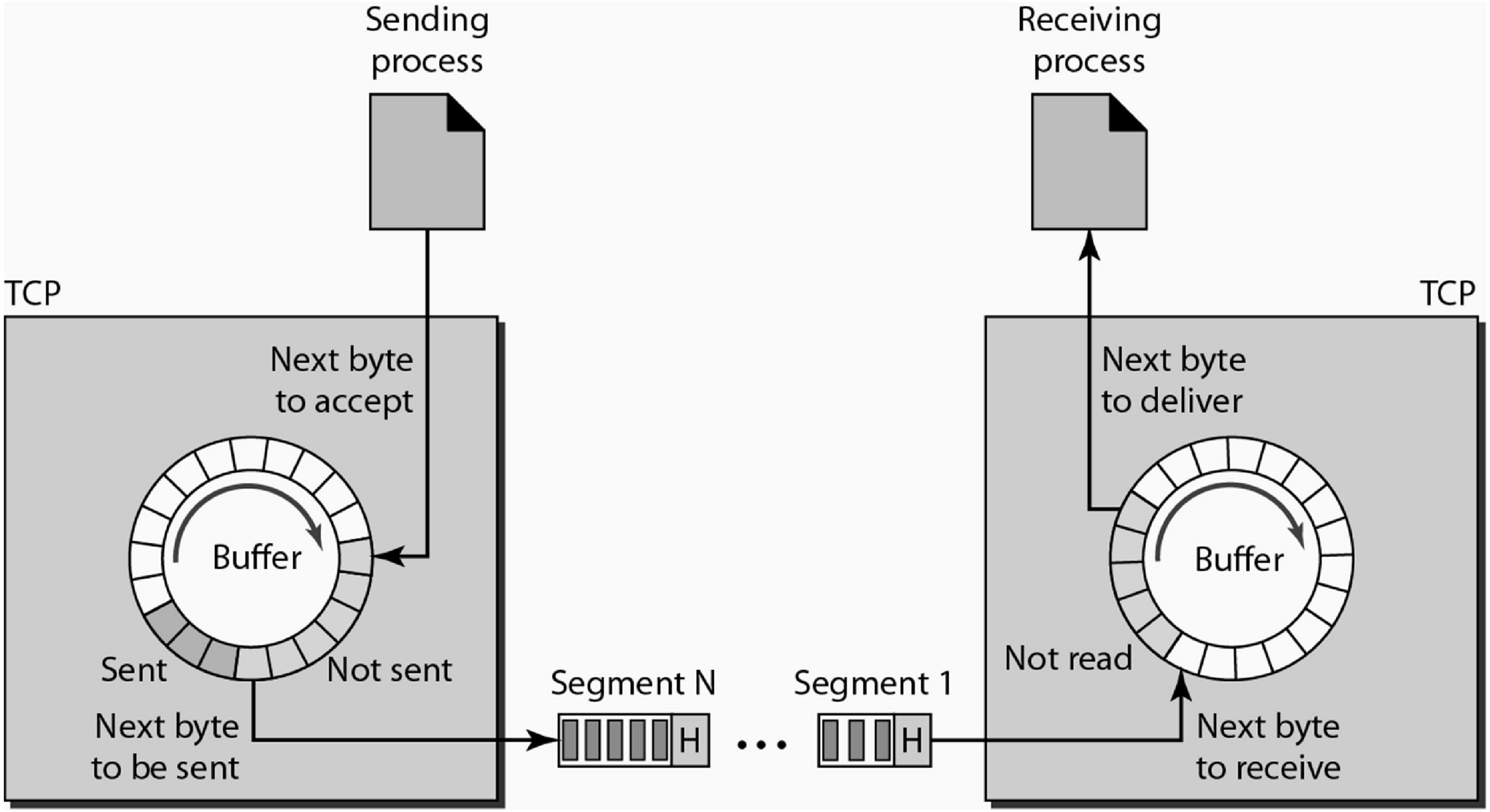
TCP Flow Control – *Buffers* and *Windows*

- Each end of the connection allocates a ***window*** to hold incoming data:
  - The size of the <u>initial</u> window is set during *Phase 1, Connection Initialization*, when both sides send their SYN messages (using the *Window Size* field)
  - Thereafter, throughout *Phase 2, Data Exchange*, <u>all</u> ***Ack*** messages include a ***window advertisement***
- The *window advertisement* can be <u>positive</u> or <u>zero</u> depending on space availability in RECVQ.

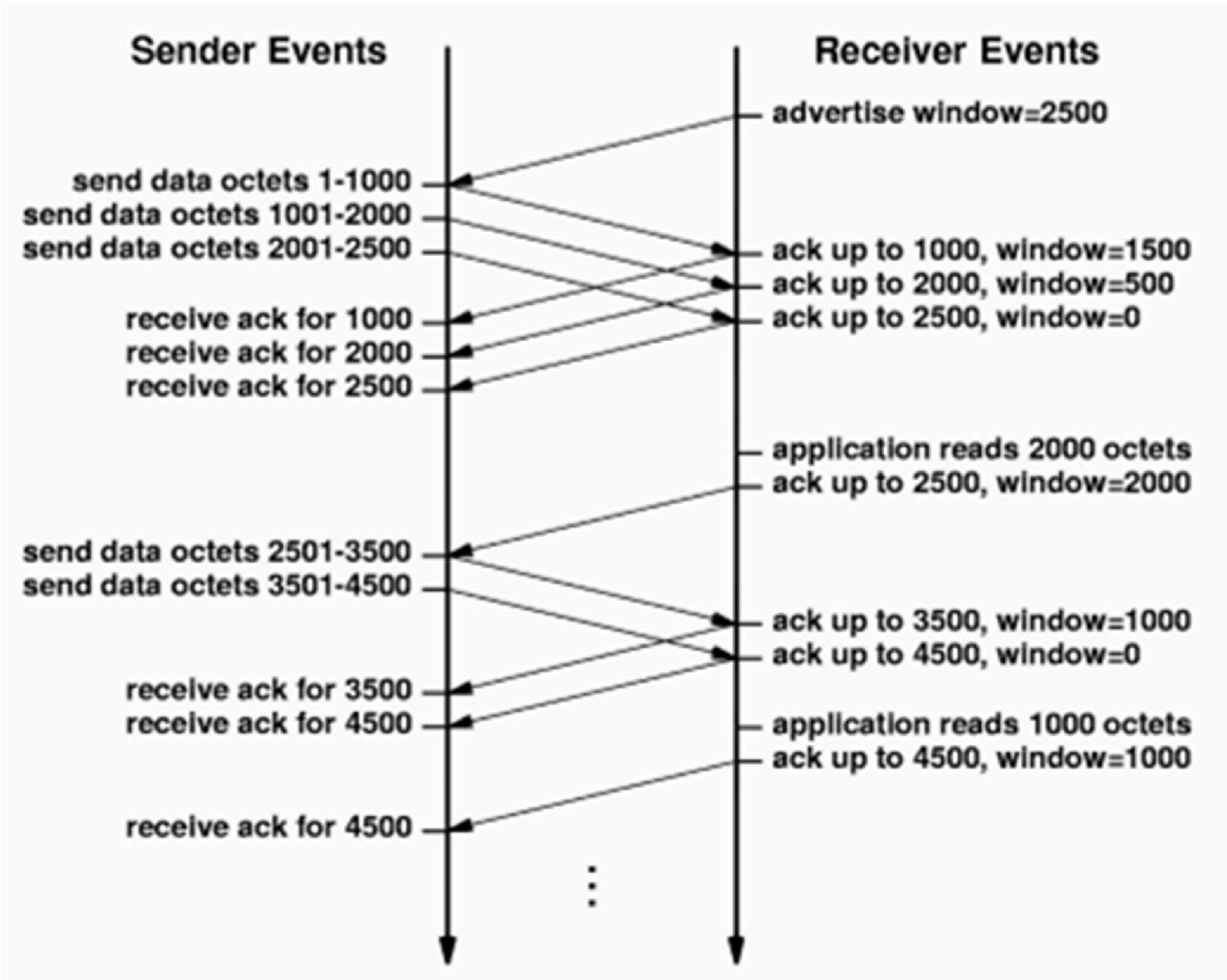
TCP Internal Data Buffers



TCP Internal Data Buffers



Operation of *window advertisements*



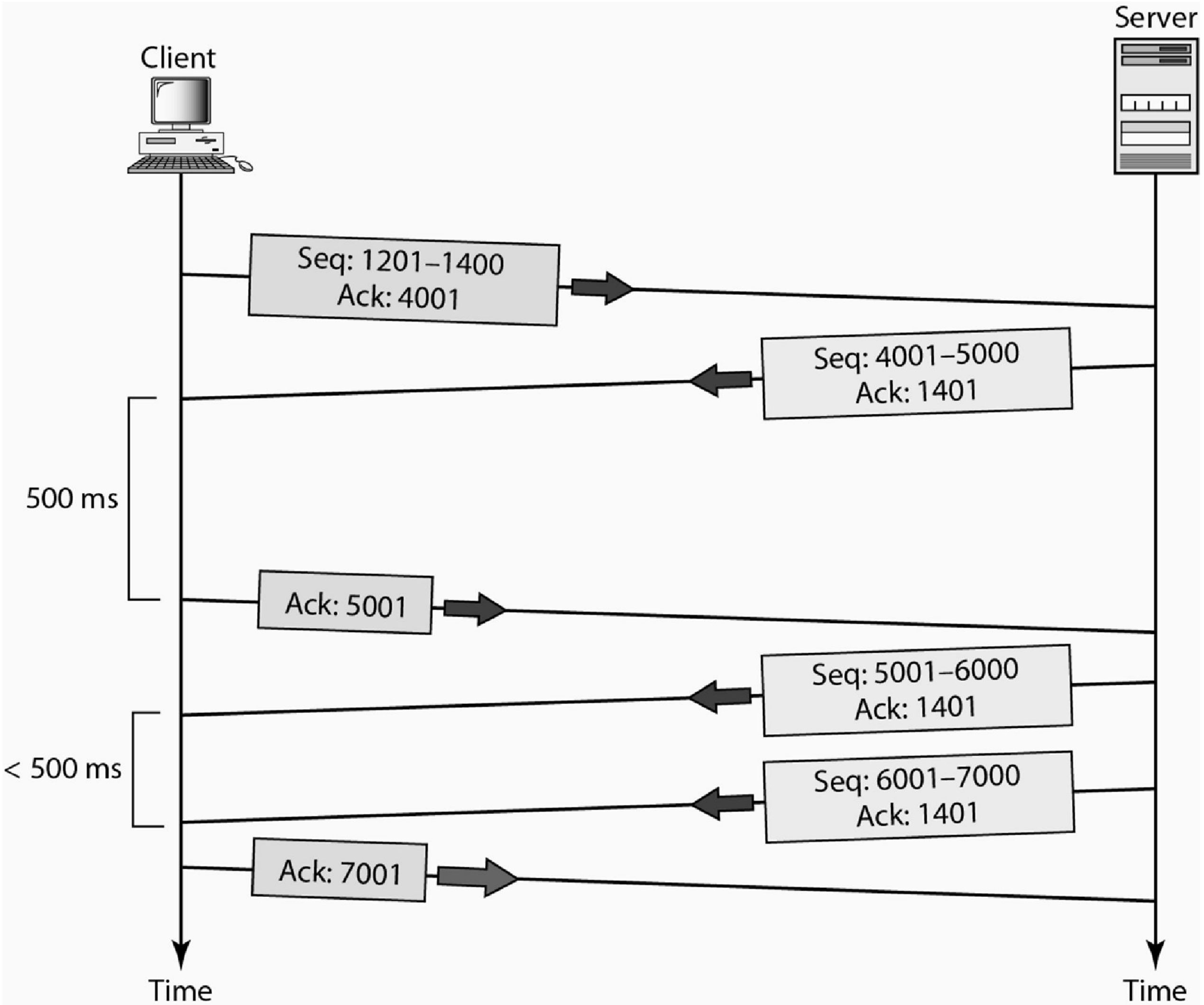
Error Control - Achieving Reliability

- TCP must address the following reliability problems:
  - Unreliable delivery by the underlying communication system
    - Segments can be lost, duplicated, delayed, or delivered out of order by the underlying communication system
  - Computer reboot
    - Segments from previous connections can arrive. The receiving host having just re-booted will not know what to do with them
- To overcome these problems TCP must:
  - Ensure that all messages are unambiguous
  - Be able to deal with *lost packets*

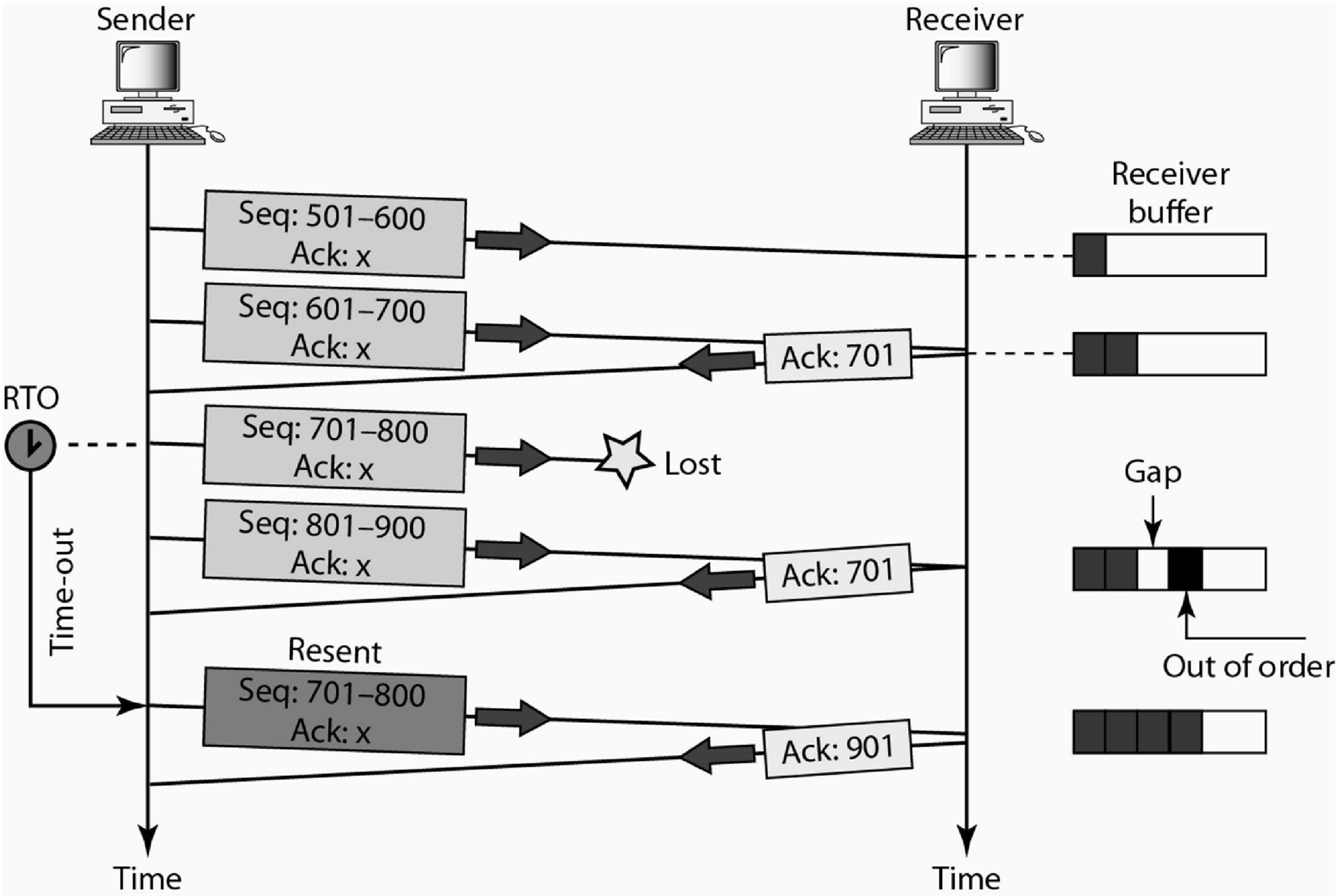
Handling Packet Loss - Retransmission

- TCP implements a *retransmission scheme*
- This involves sending *acknowledgements* and the use of *timers*
- This scheme is the key to the success of TCP
- How long should TCP wait before retransmitting?
  - It depends upon:
    - The underlying network
    - Traffic levels
- TCP uses an *adaptive retransmission* scheme

TCP Data Flow – Normal Operation



TCP Data Flow – Lost Segment Scenario



TCP's *Adaptive Retransmission* Scheme

- TCP estimates the *round-trip delay* for <u>each</u> active connection
- Using a *weighted average* together with a *variance* factor TCP calculates a value for the *retransmission time*
- This helps TCP to react <u>quickly</u> to changes in *traffic levels* and to <u>maximize</u> throughput on each connection