Transmission Control Protocol (TCP)

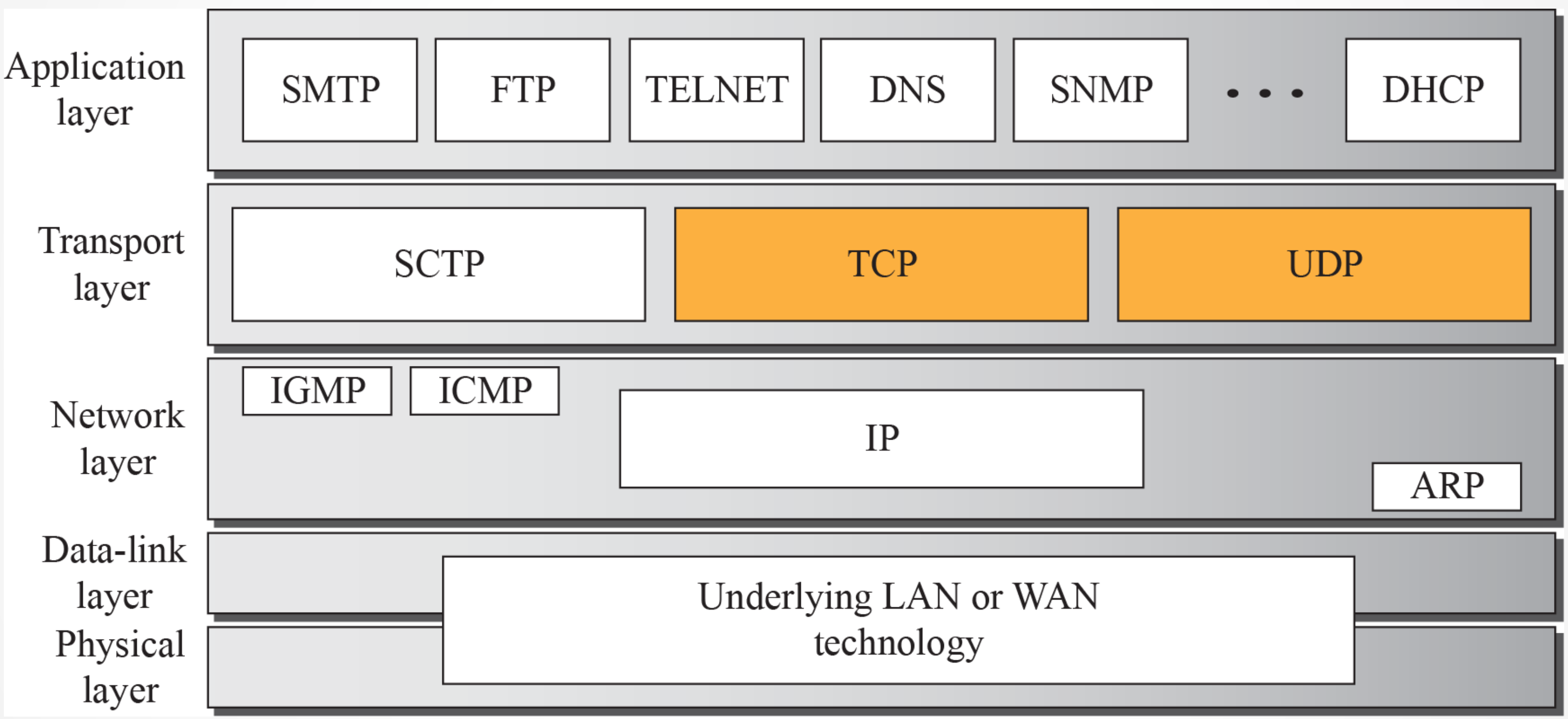
Internet Transport-Layer Protocols

- Transport layer provides logical communication between processes

- Internet supports a few transport layer protocols
  - UDP, TCP, SCTP

- TCP
  - connection-oriented : connection establishment, data transfer, and connection tear down.
  - reliable : uses a combination of GBN and SR protocols to provide reliability.

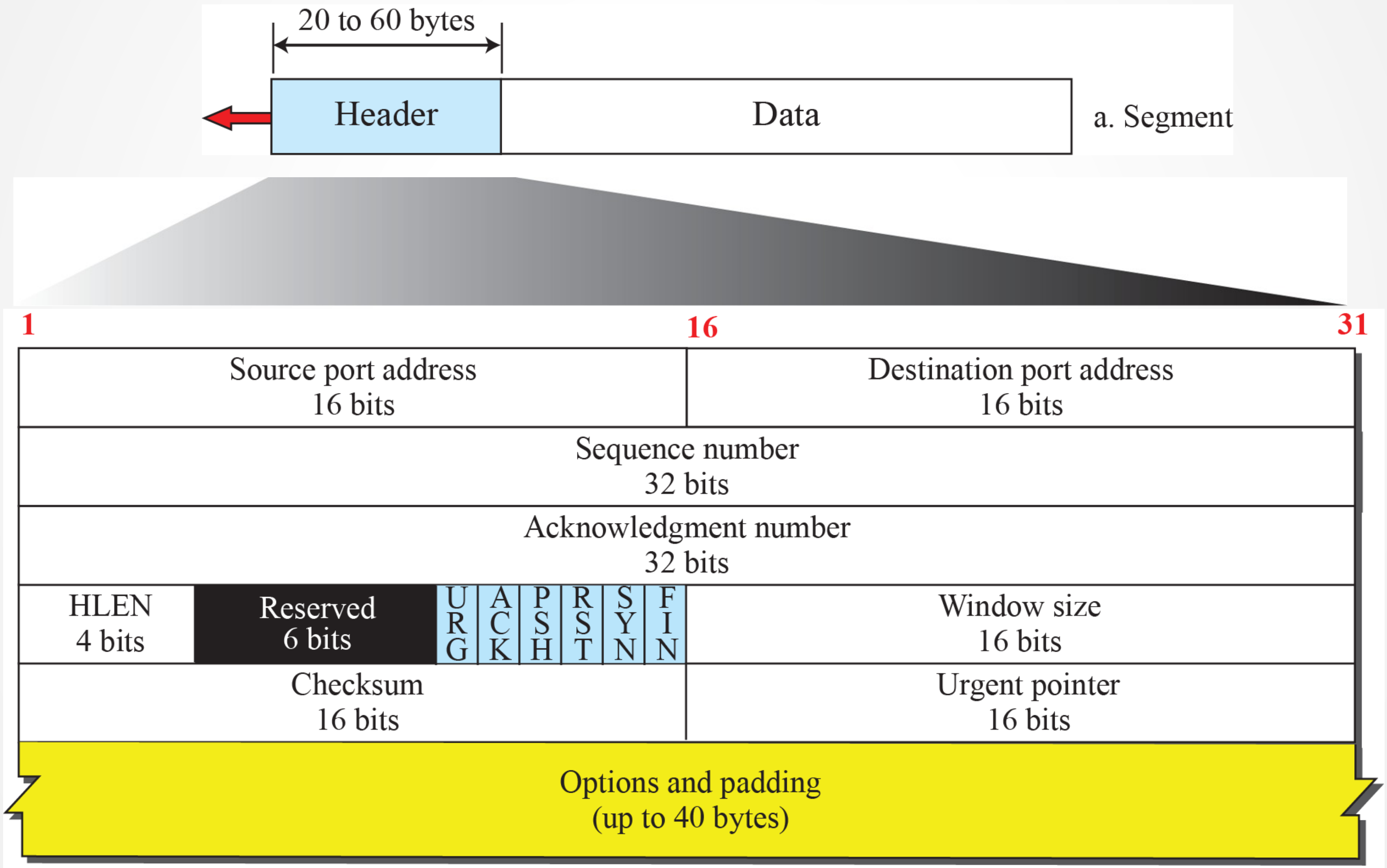
Transport-layer protocols in the TCP/IP Stack



TCP Services

- Process-to-Process Communication
- Stream Delivery Service
- Full-Duplex Communication
- Multiplexing and Demultiplexing
- Connection-Oriented Service
- Reliable Service

TCP Segment Format

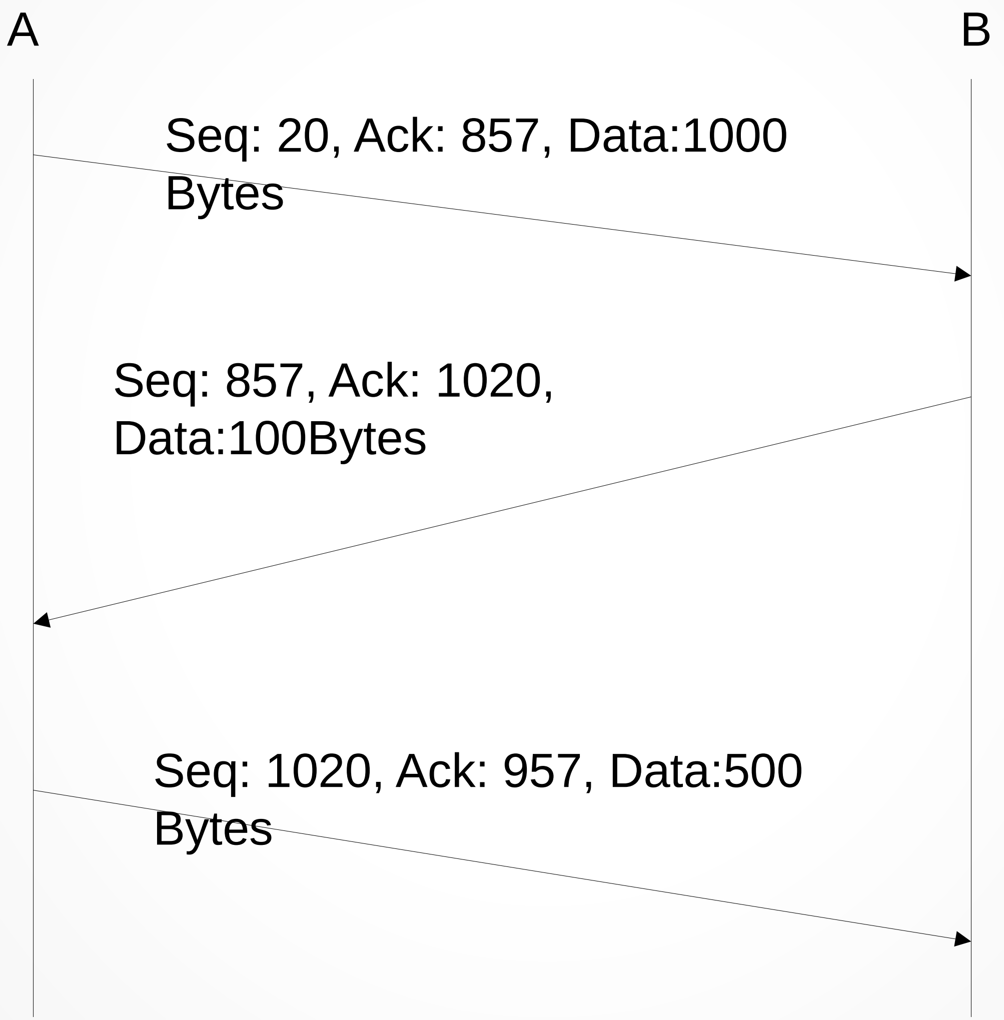


b. Header

TCP Segment Format

- Source/Destination Port: Identifies sending/receiving process
  - Client: Ephemeral port; Server: Well-known port
- Sequence Number:
  - Each byte has a sequence number
  - Sequence number field contains the sequence number of the first byte in the segment.
  - If SYN is present, this is the initial sequence number (ISN) and the first data byte is ISN+1.
- Acknowledgment Number:
  - Acknowledgment field carry information about flow in the other direction
  - Carries sequence number of next byte a host is expecting

Example


A
B
Seq: 20, Ack: 857, Data:1000 Bytes
Seq: 857, Ack: 1020, Data:100Bytes
Seq: 1020, Ack: 957, Data:500 Bytes

Initial Sequence Number (ISN)

- Why not start with Seqno zero?
- Segments from different connections can get mixed up
- Security risk when ISN's are predictable
- Original solution: Use a clock (e.g. increments every 4 microsec) to choose ISN
  - 32 bit sequence number wraps around in 4 hrs
- Current implementations use random ISN

TCP Segment Format

Flags (UAPRSF):

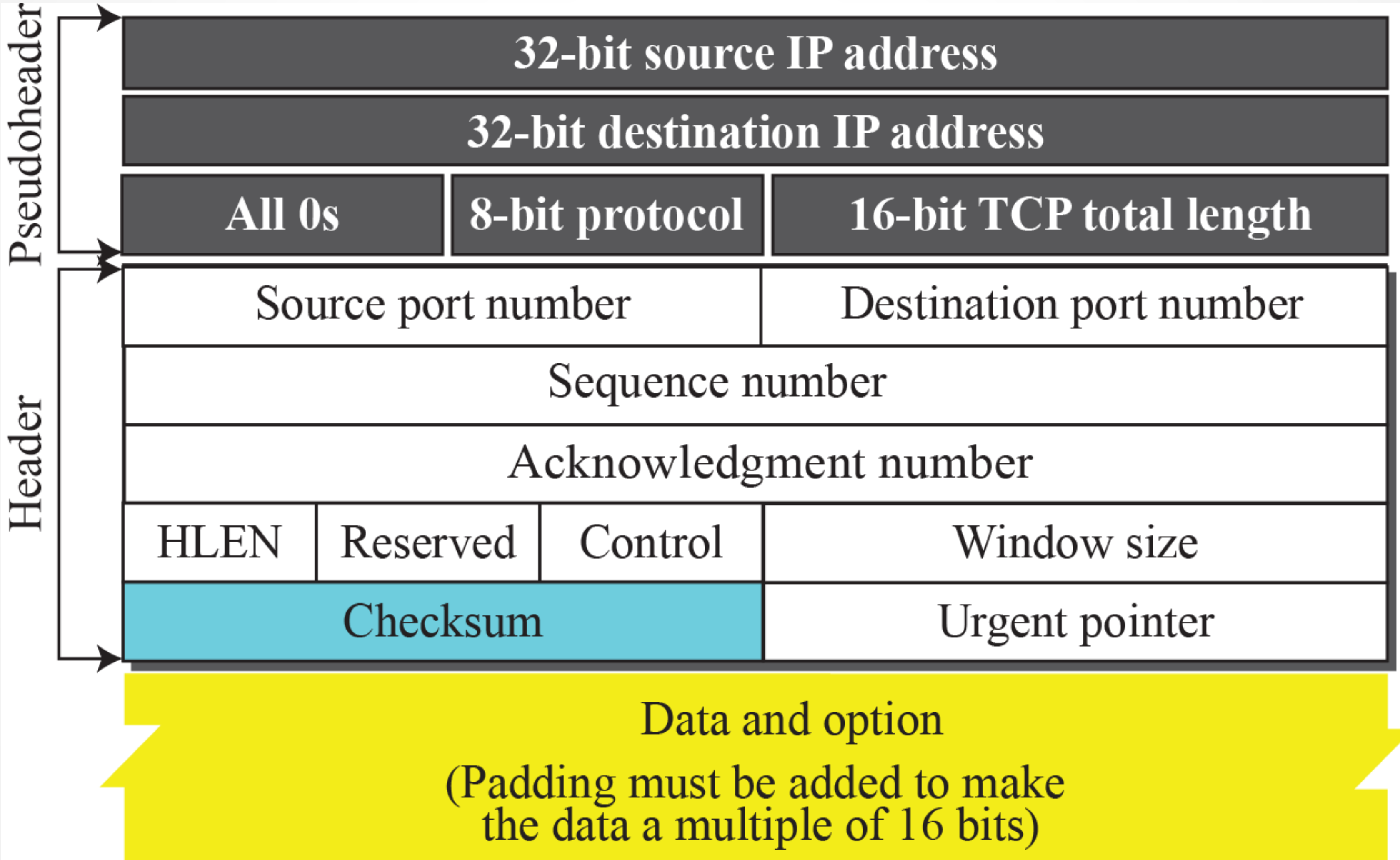
- U: Urgent flag indicates segment contains urgent data
  - UrgentPointer (bytes) indicates where in the segment non-urgent data begins
- A: Ack bit is set if the acknowledgment field is Valid
- P: Push flag indicates receiver should pass data to higher layers immediately
- R: Reset, used to abort connection
- S/F: Syn and Fin flags are used during connection establishment and termination

TCP Segment Format

Checksum:

- Similar to UDP
- Compulsory in IPv4 and IPv6
- Calculated over TCP header, data and pseudoheader
  – Pseudoheader: source, destination, protocol of IP header and TCP segment total length (calculated)

Pseudoheader for checksum calculation

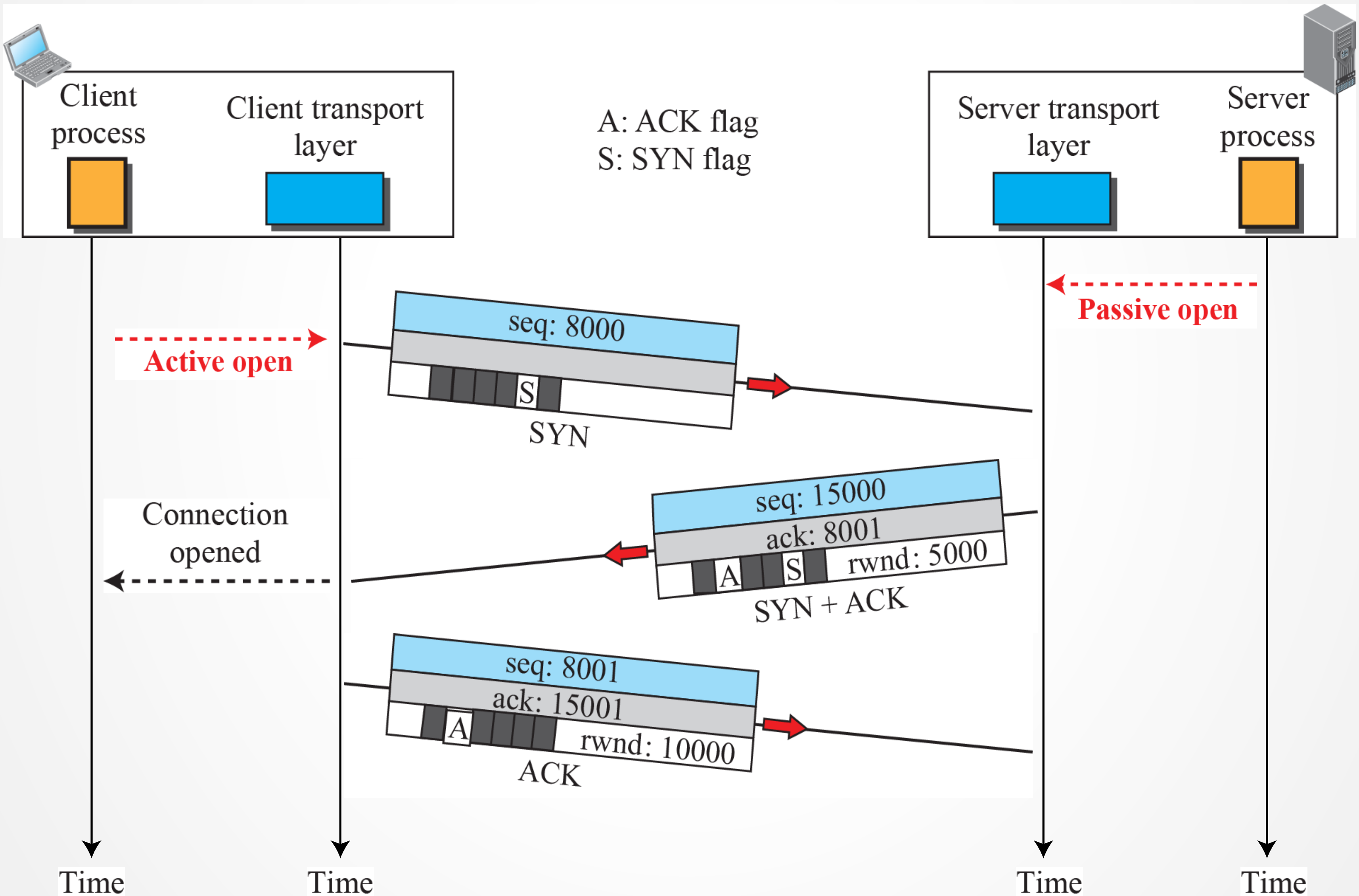


TCP Segment Format

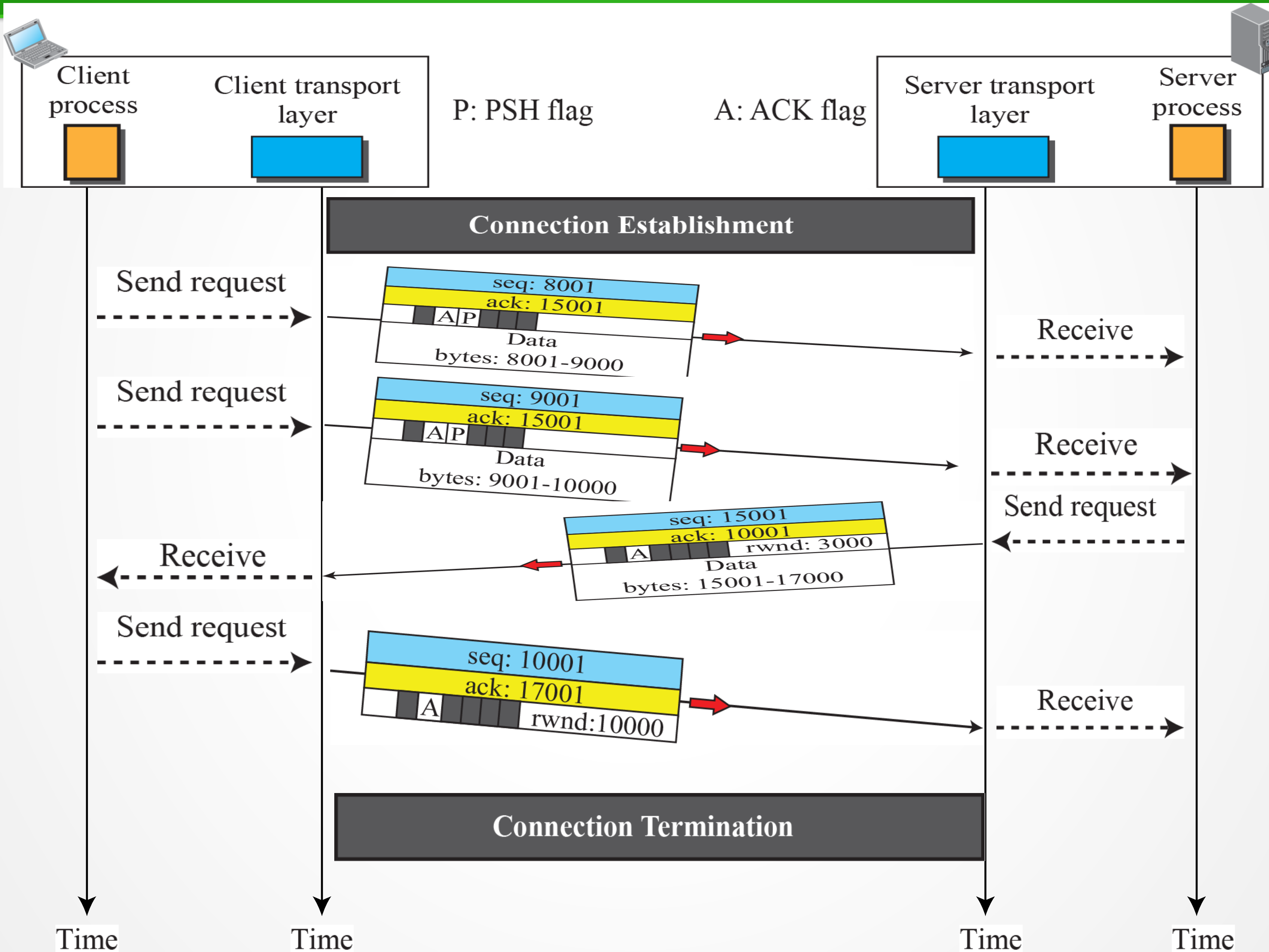
Options

- Can negotiate maximum segment size
- Can perform window scaling
- Permits use of selective-acks
  - Both to indicate the device supports selective acknowledgments and carry the actual ack information

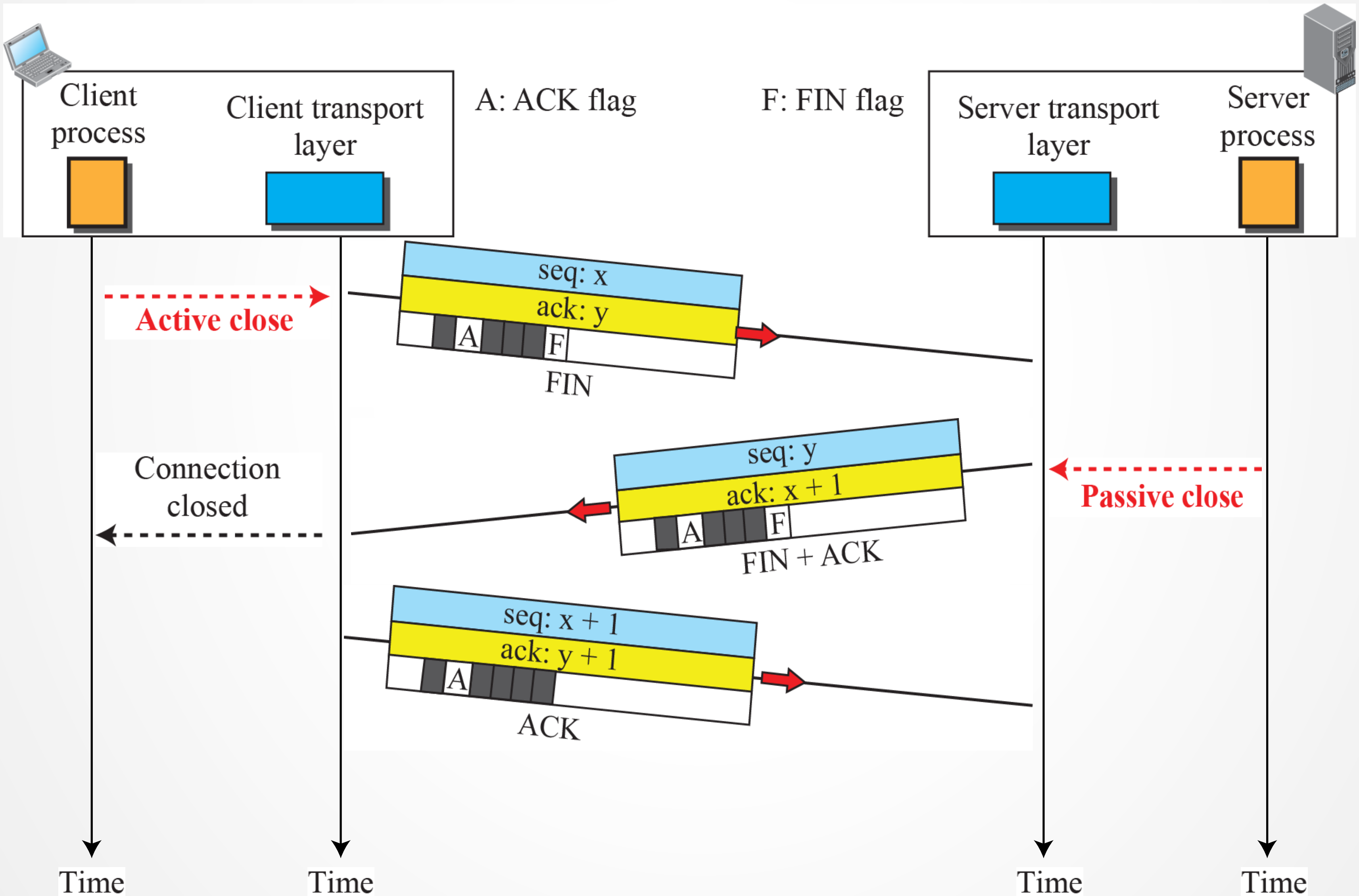
Connection establishment using three-way handshaking



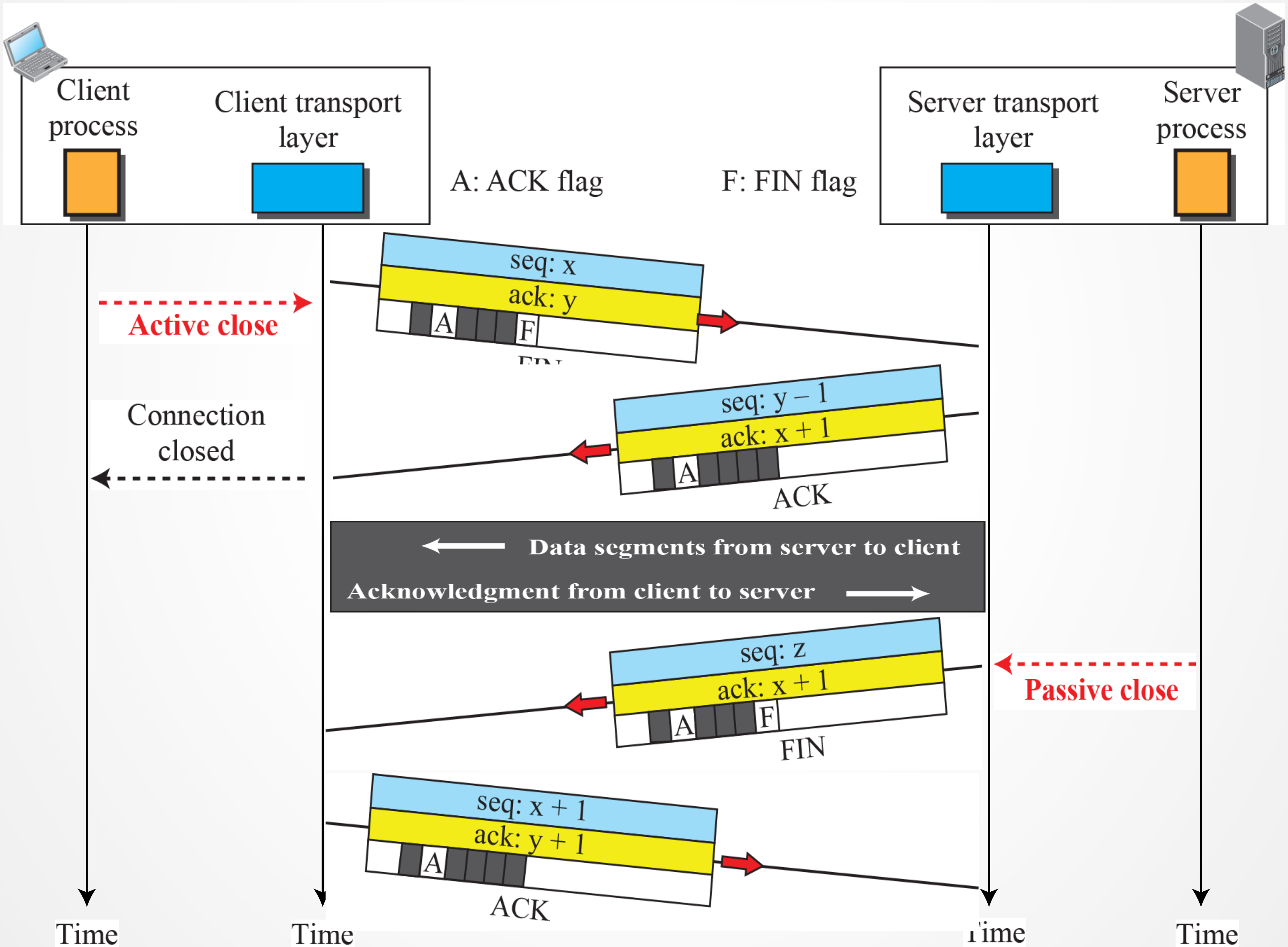
Data Transfer



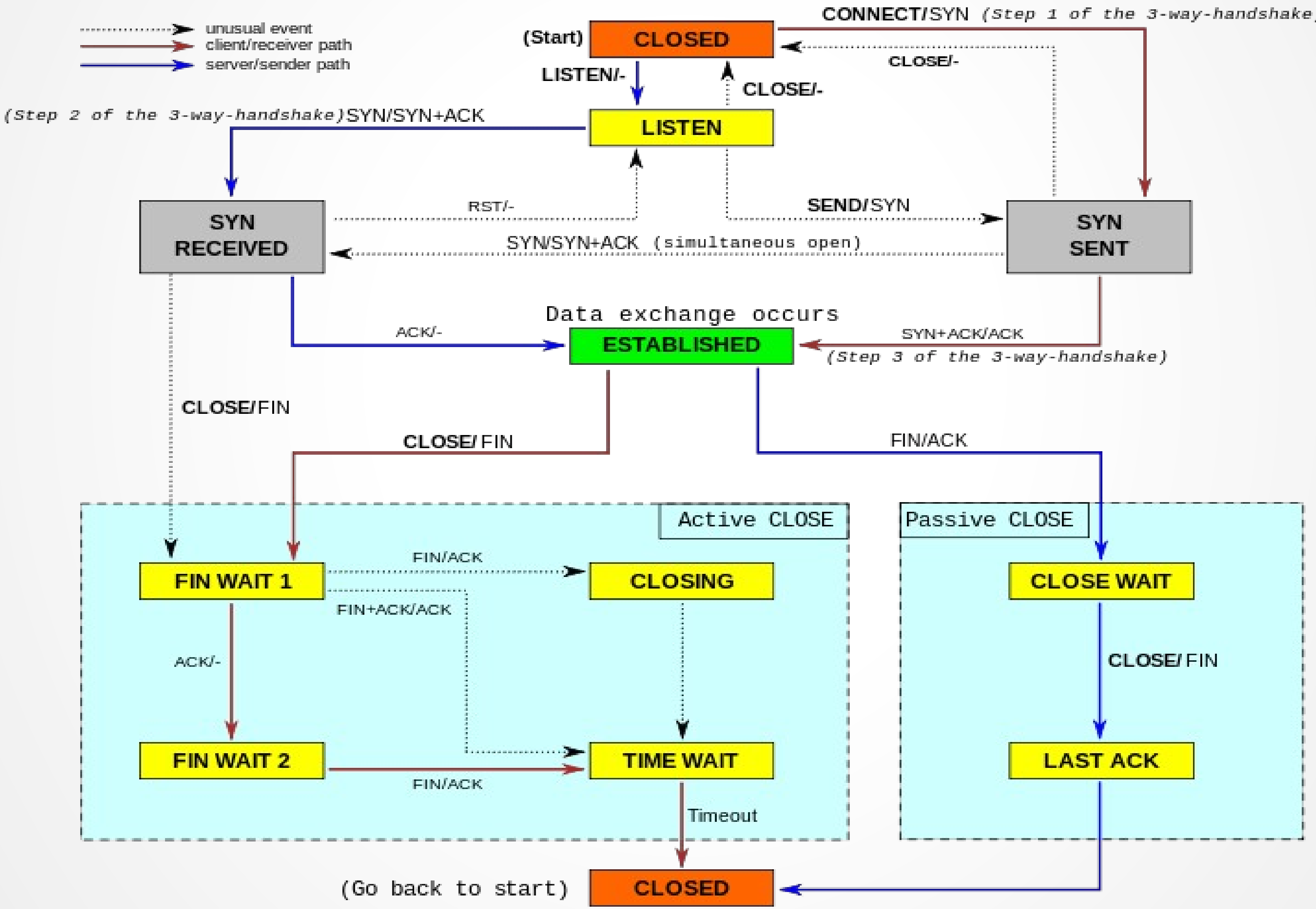
Connection termination using three-way handshaking



Connection termination using four-way handshaking (Half-close)



TCP State Diagram



Time-Wait State

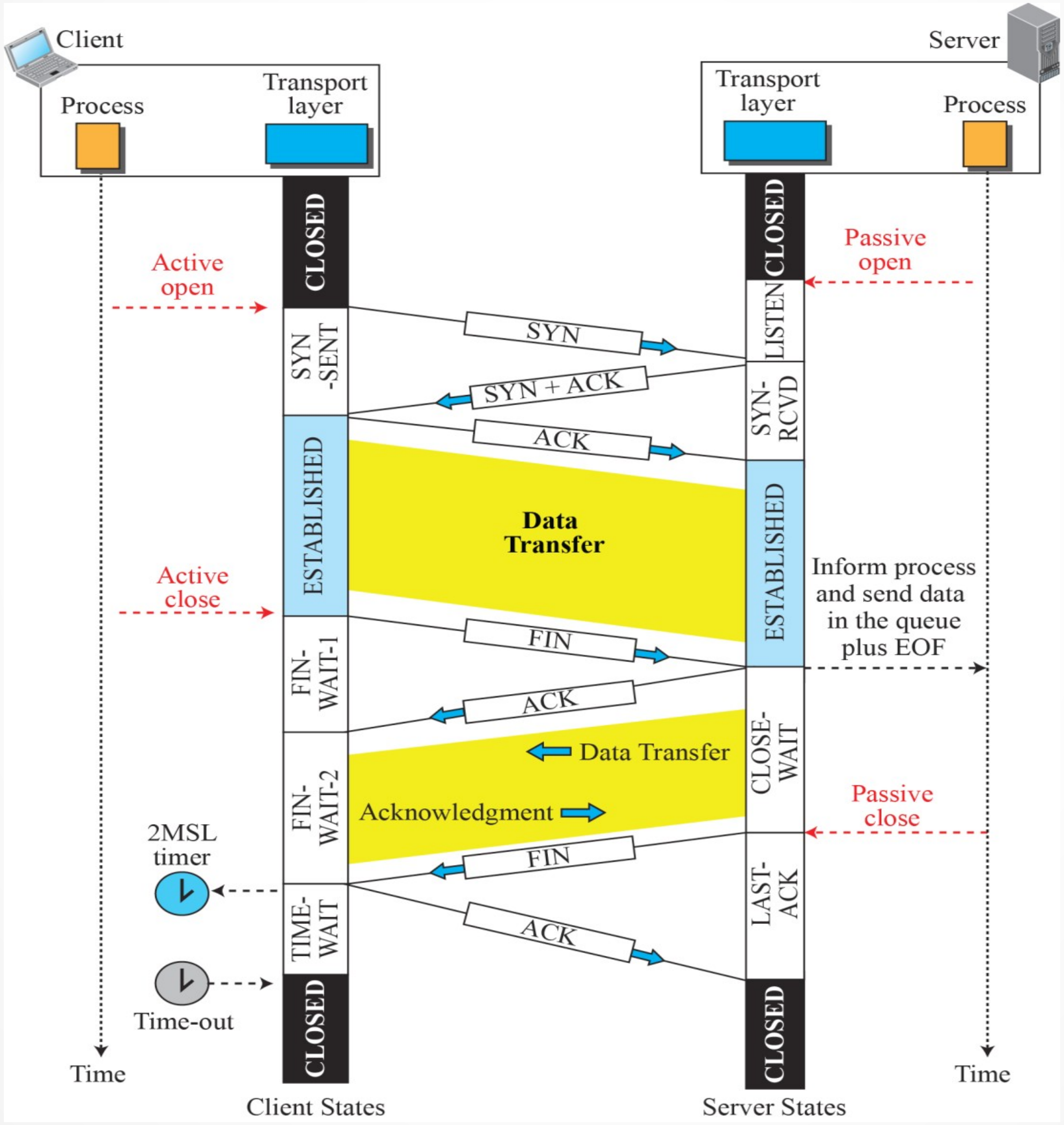
Wait in time-wait for 2*MSL (maximum segment lifetime)

- Helps clear out older packets in the network; prevents them from interfering with new connection
- Time spent in time-wait range from 30sec to 2 min

States for TCP

State	*Description*
CLOSED	No connection exists
LISTEN	Passive open received; waiting for SYN
SYN-SENT	SYN sent; waiting for ACK
SYN-RCVD	SYN+ACK sent; waiting for ACK
ESTABLISHED	Connection established; data transfer in progress
FIN-WAIT-1	First FIN sent; waiting for ACK
FIN-WAIT-2	ACK to first FIN received; waiting for second FIN
CLOSE-WAIT	First FIN received, ACK sent; waiting for application to close
TIME-WAIT	Second FIN received, ACK sent; waiting for 2MSL time-out
LAST-ACK	Second FIN sent; waiting for ACK
CLOSING	Both sides decided to close simultaneously

Time-line diagram for a common scenario

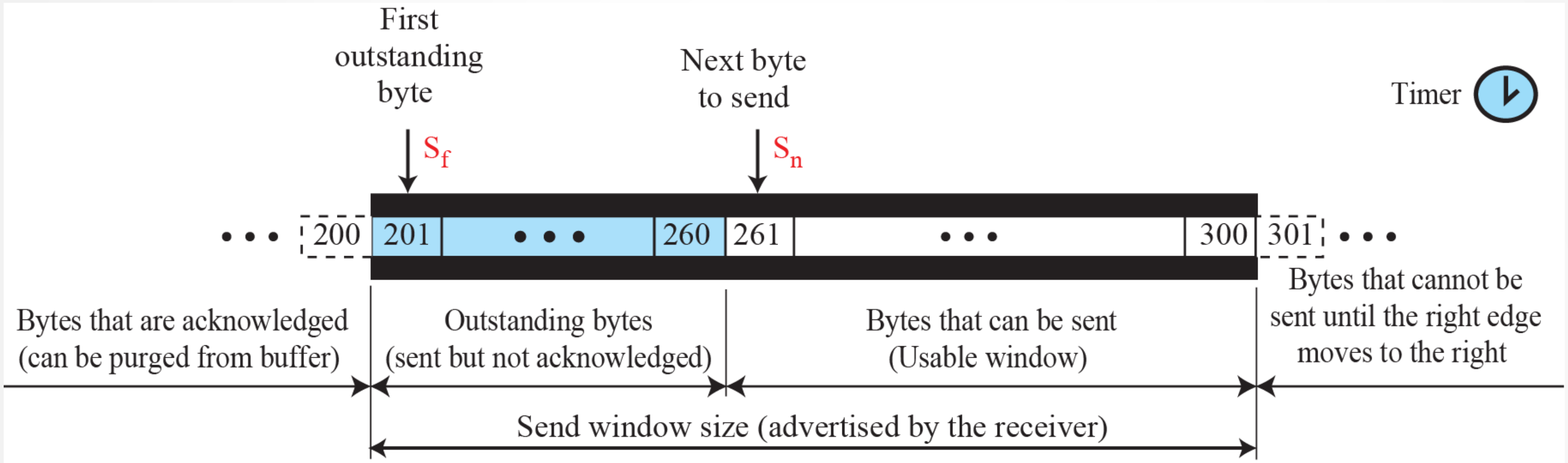


Windows in TCP

TCP uses two windows (send window and receive window) for each direction of data transfer.

- send window
- receive window

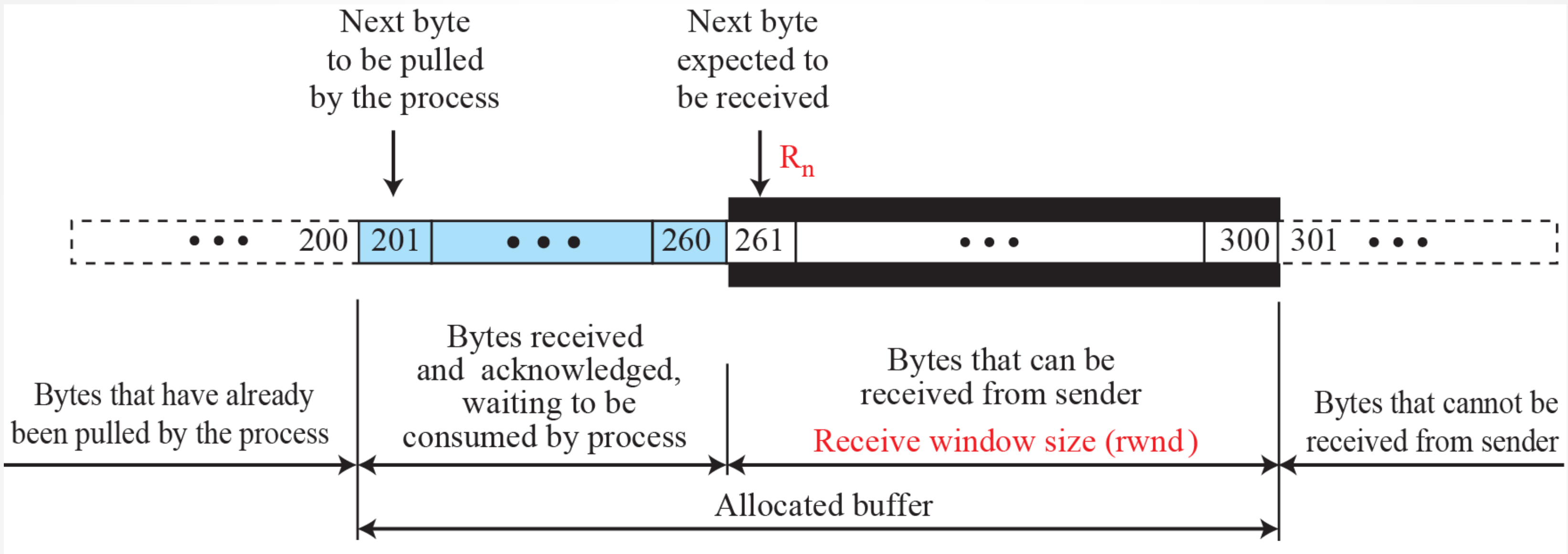
Send window in TCP



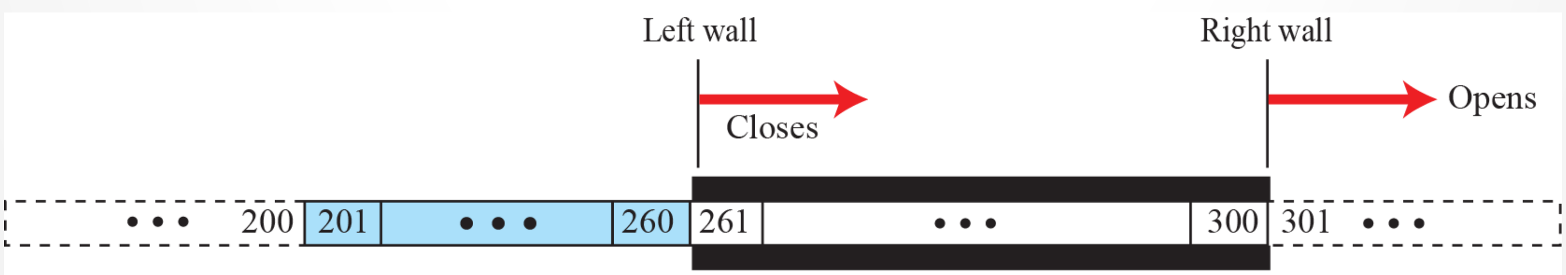
a. Send window

b. Opening, closing, and shrinking send window

Receive window in TCP



a. Receive window and allocated buffer



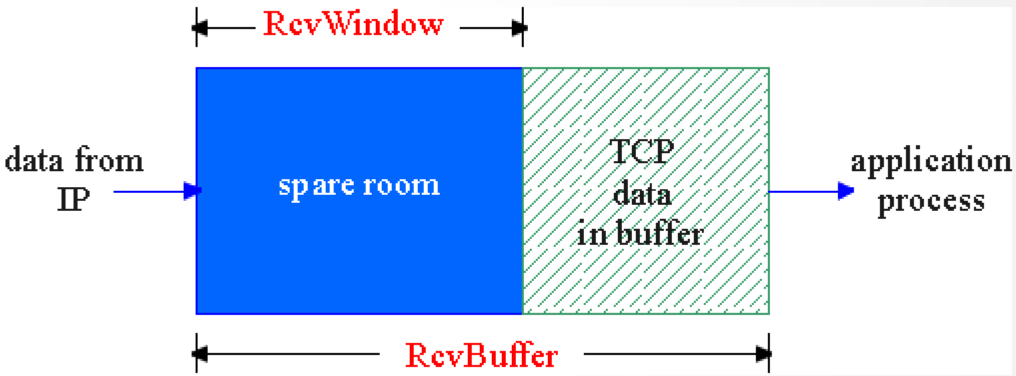
b. Opening and closing of receive window

Flow Control in TCP

Flow Control : sender won't overflow receiver's buffer by Transmitting too much, too fast.

How it works?

receive side of TCP connection has a receive buffer.



app process may be slow at reading from buffer

spare room in buffer

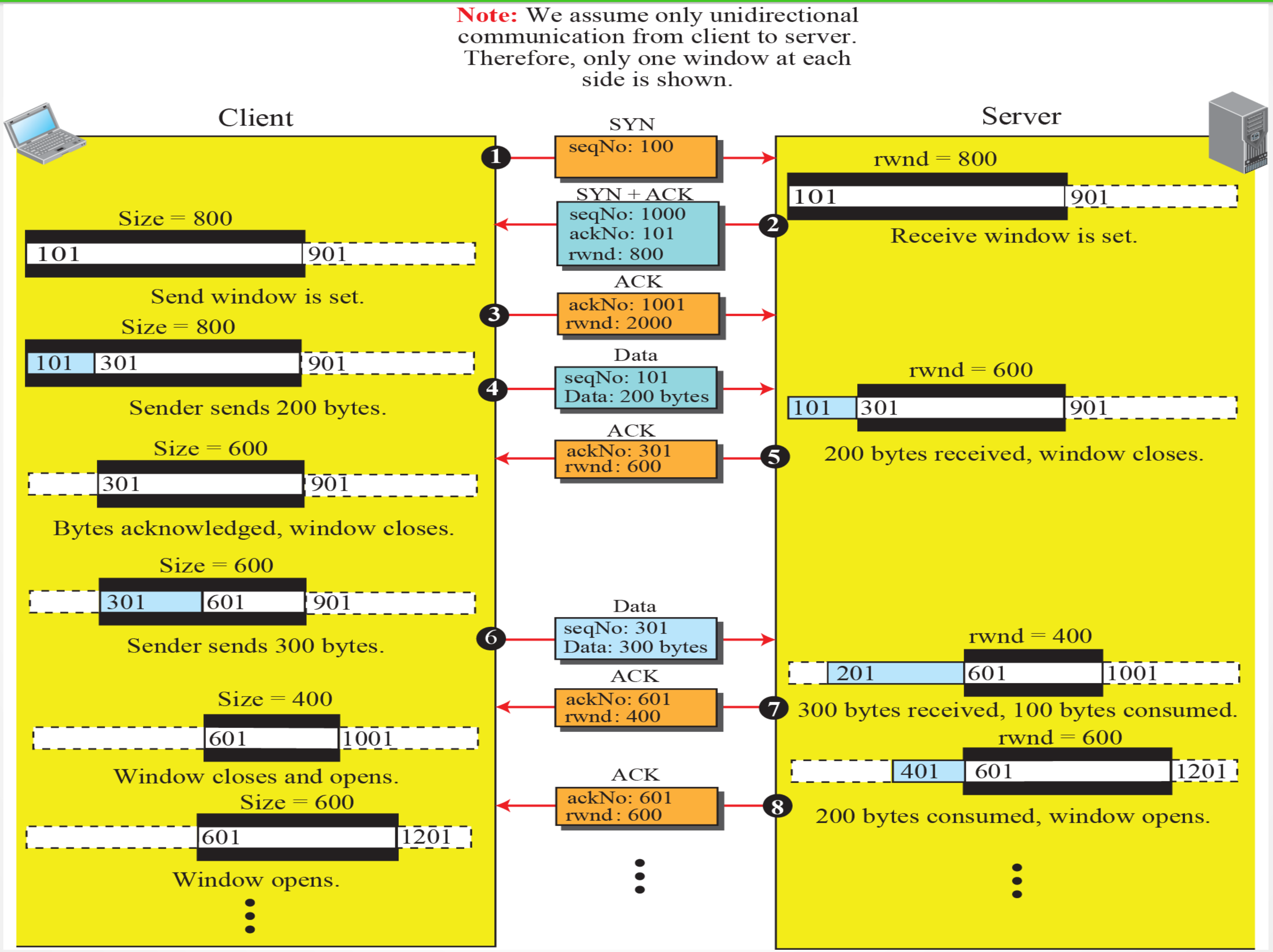
RcvWindow = RcvBuffer-[LastByteRcvd – LastByteRead]

rcvr advertises spare room by including value of RcvWindow in segments

sender limits unACKed data to RcvWindow

guarantees receive buffer doesn't overflow

Example: Flow Control in TCP

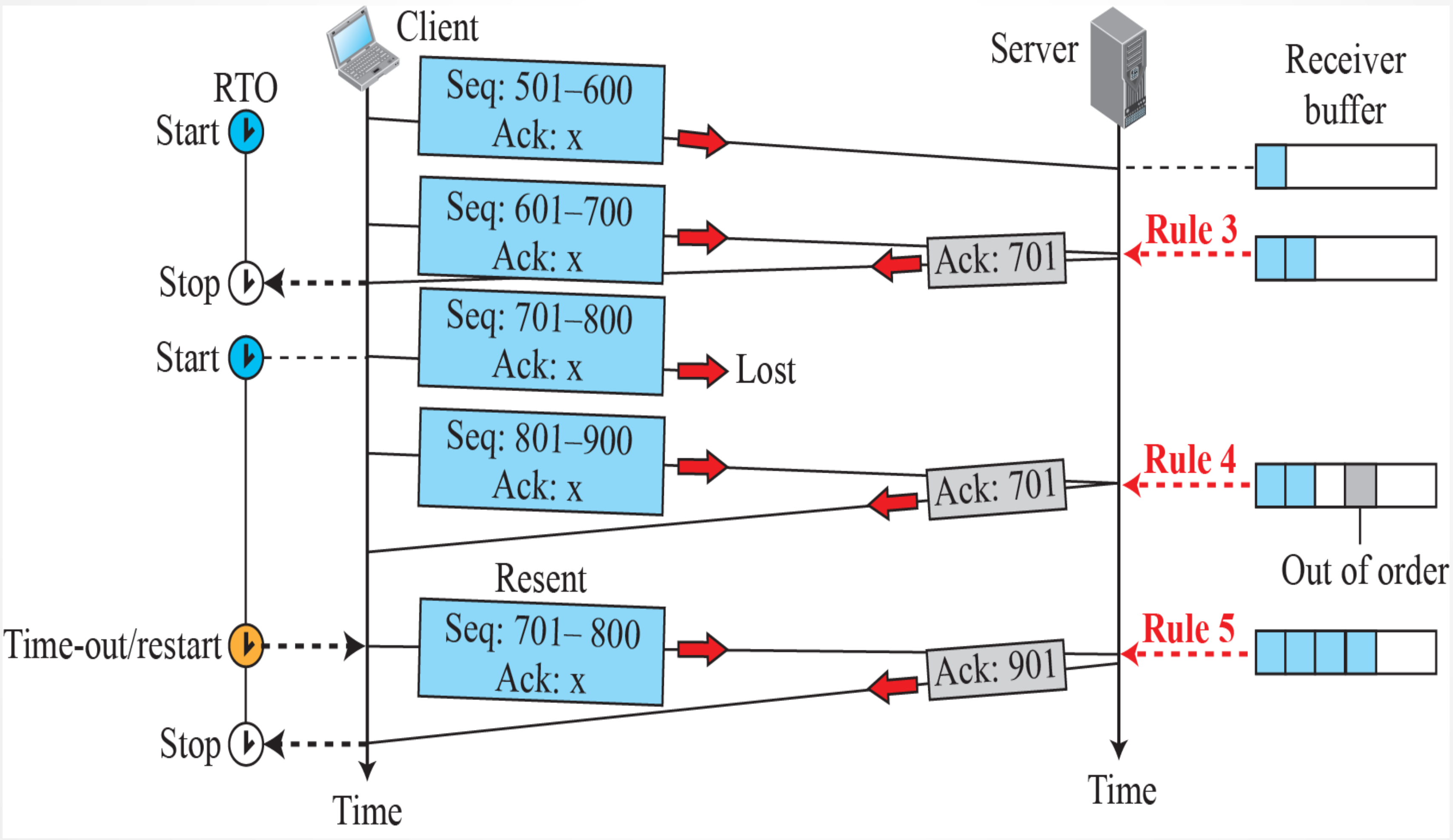


Error Control in TCP

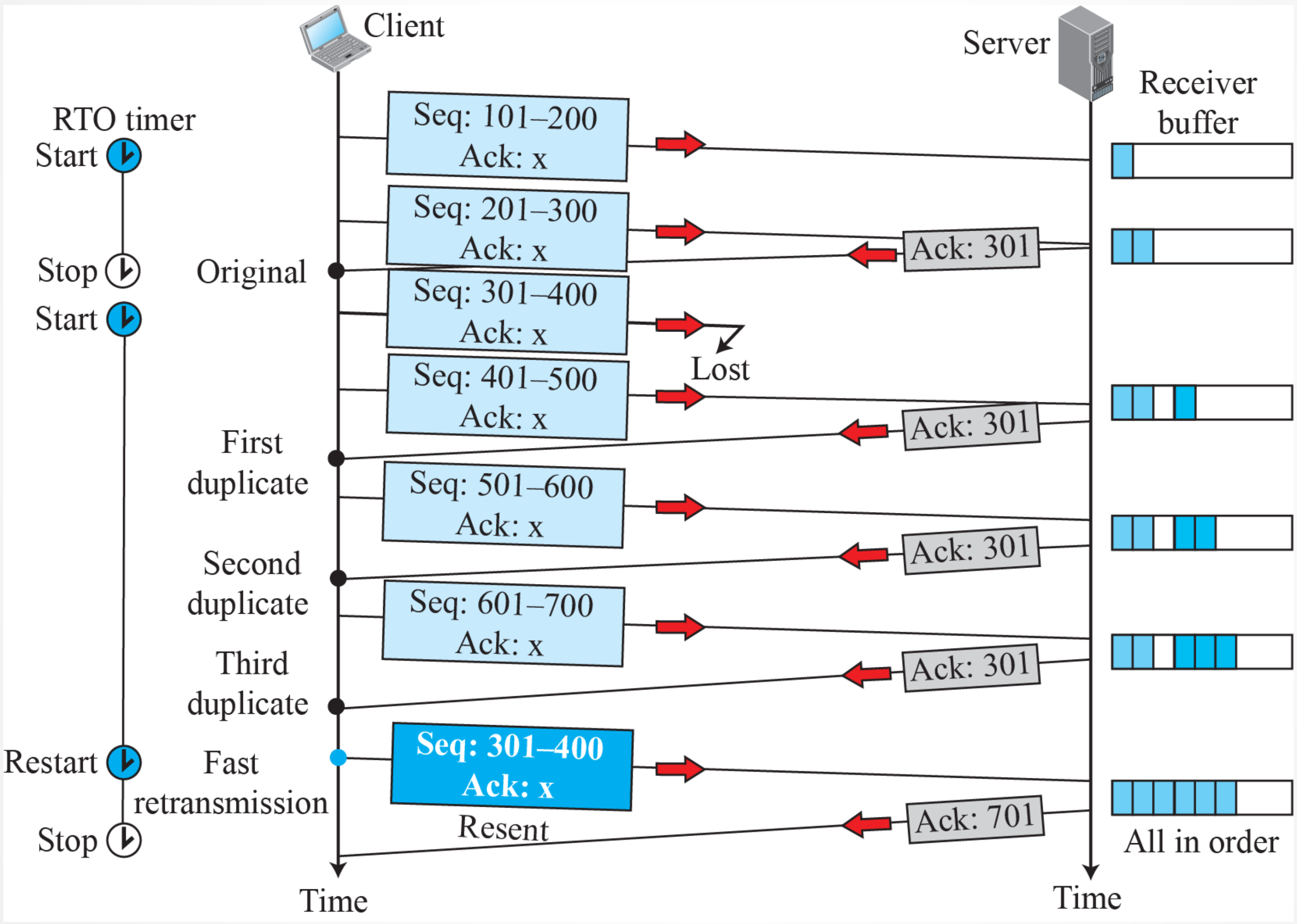
Error Control : to recover or conceal the effects from packet losses. Following techniques are used for the same.

1. **Checksum :** checks corrupted segment.
2. **Acknowledgment :** confirm the receipt of data segments.
   - **Cumulative Acknowledgment (ACK)**
     - Acknowledge receipt of segments cumulatively.
     - No feedback for discarded, lost, or duplicated segments
   - **Selective Acknowledgment (SACK)**
     - Does not replace ACK, but reports additional information to sender.
     - Reports a block of byte that is out of order or duplicated.
     - SACK is implemented as a option at the end of TCP header.
3. **Retransmission :** TCP maintains 1 RTO for each connection.
   - **Retransmission after Retransmission time-out (RTO) :** each time TCP retransmits, it sets the RTO to twice the previous value.
   - **Retransmission after 3 duplicate ACK segments(fast retransmission) :** when the RTO is very large this feature helps in retransmitting the segment without waiting for the timeout.

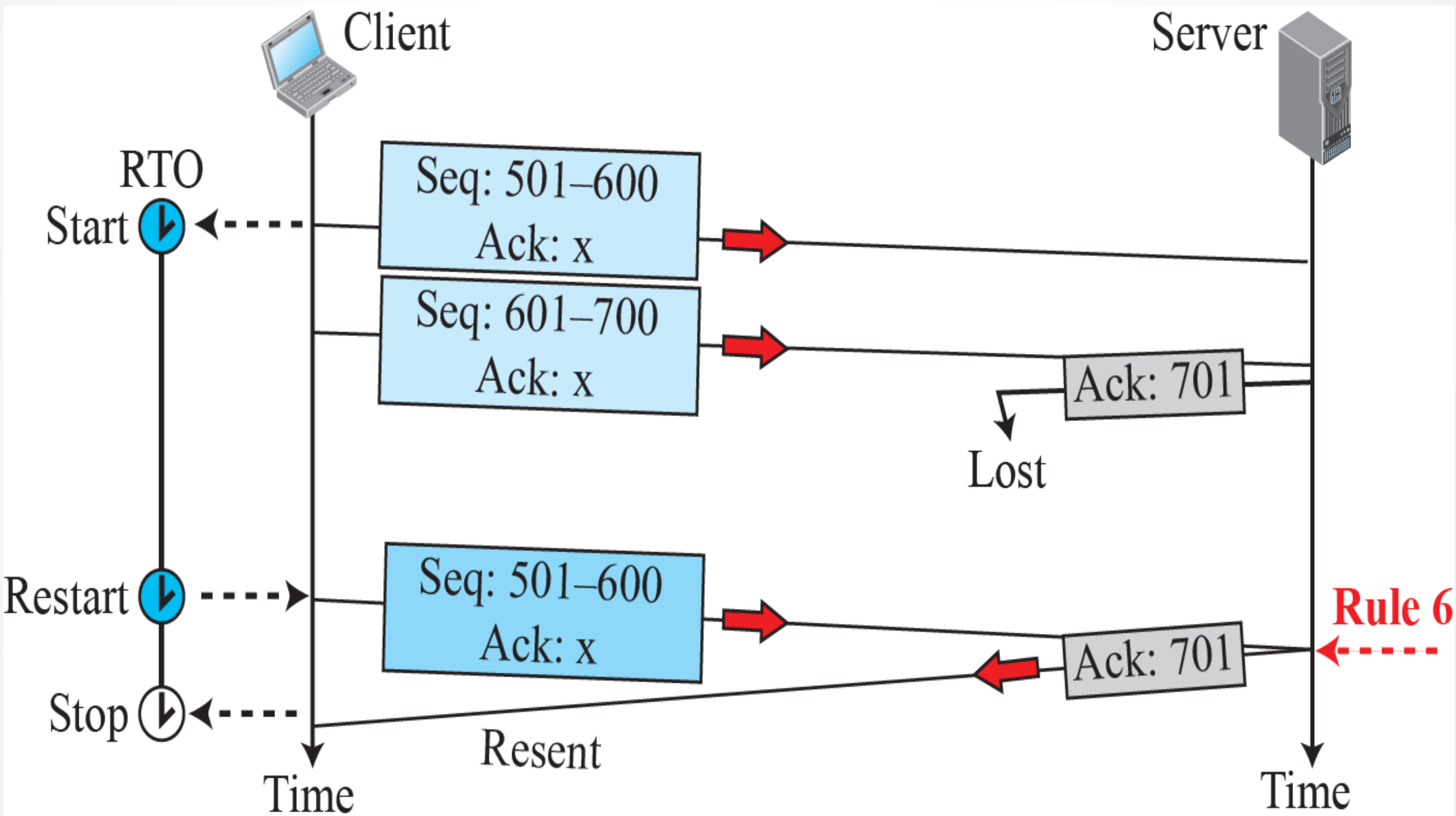
Lost Segment



Fast Retransmission



Lost ACK : Resending a Segment



Congestion Control: What and Why?

- Internet is used by many independent users
  - Resources (link capacity) are finite
  - If every user sends data at very high rate, it will cause "congestion"
    - − packets will be dropped, i.e. unreliable transmission
    - − seemingly high utilization of resource may be actually very low!
  - If every user sends data at very low rate, resource will not be well-utilized

→Users need to send data at the "correct" rate so that
  - Resources are well-utilized
  - Users get "reliable" data transfer
  - Resources should be shared "fairly"

→ This is the primary goal of congestion control

Congestion Control

Receiver flow control

1. Avoid overloading receiver
2. rwnd : receiver (advertised) window
3. Receiver sends rwnd to sender

Network congestion control

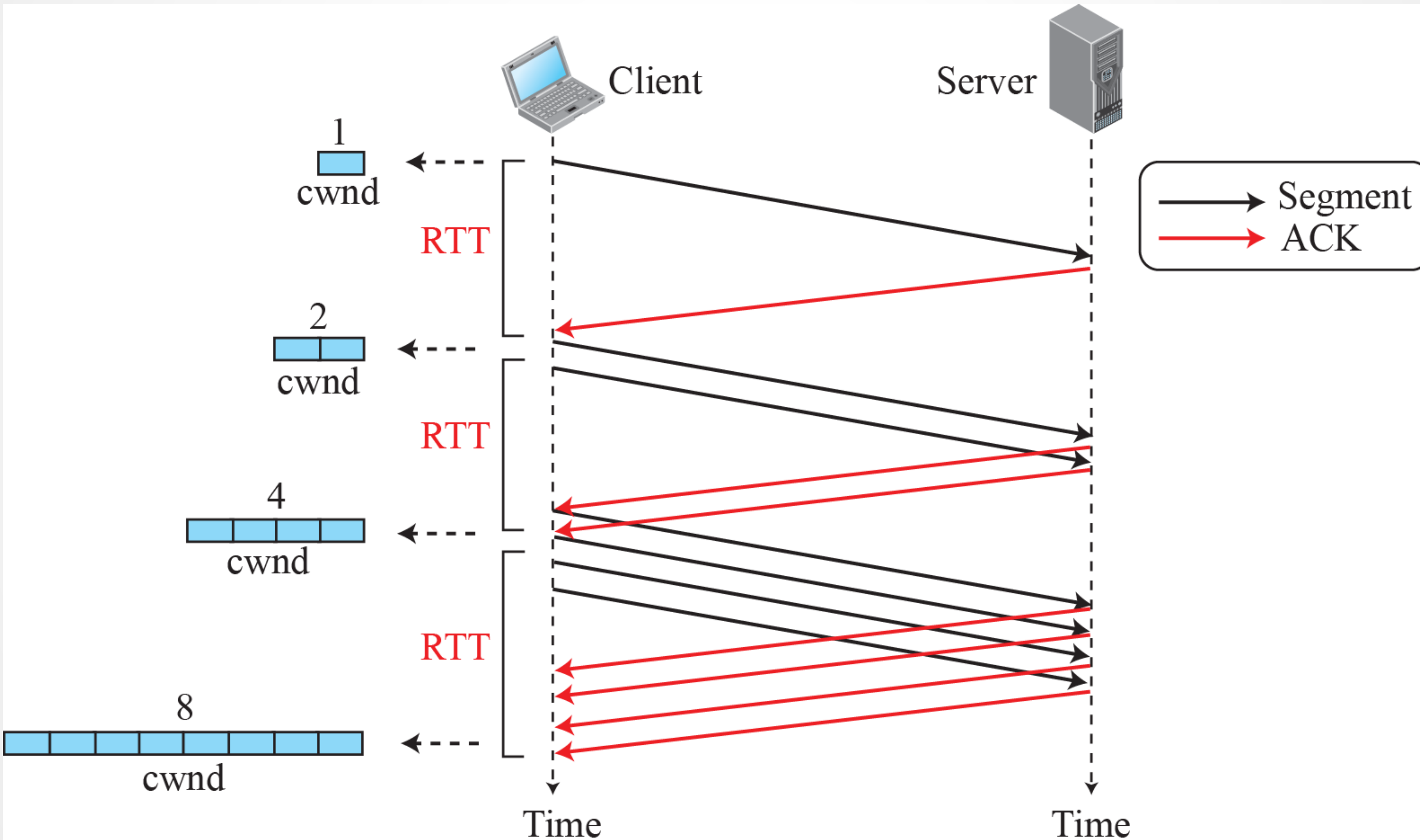
4. Sender tries to avoid overloading network
5. Packet loss indicates network congestion
6. cwnd: congestion window

Sender sets W = min (cwnd,rwnd)

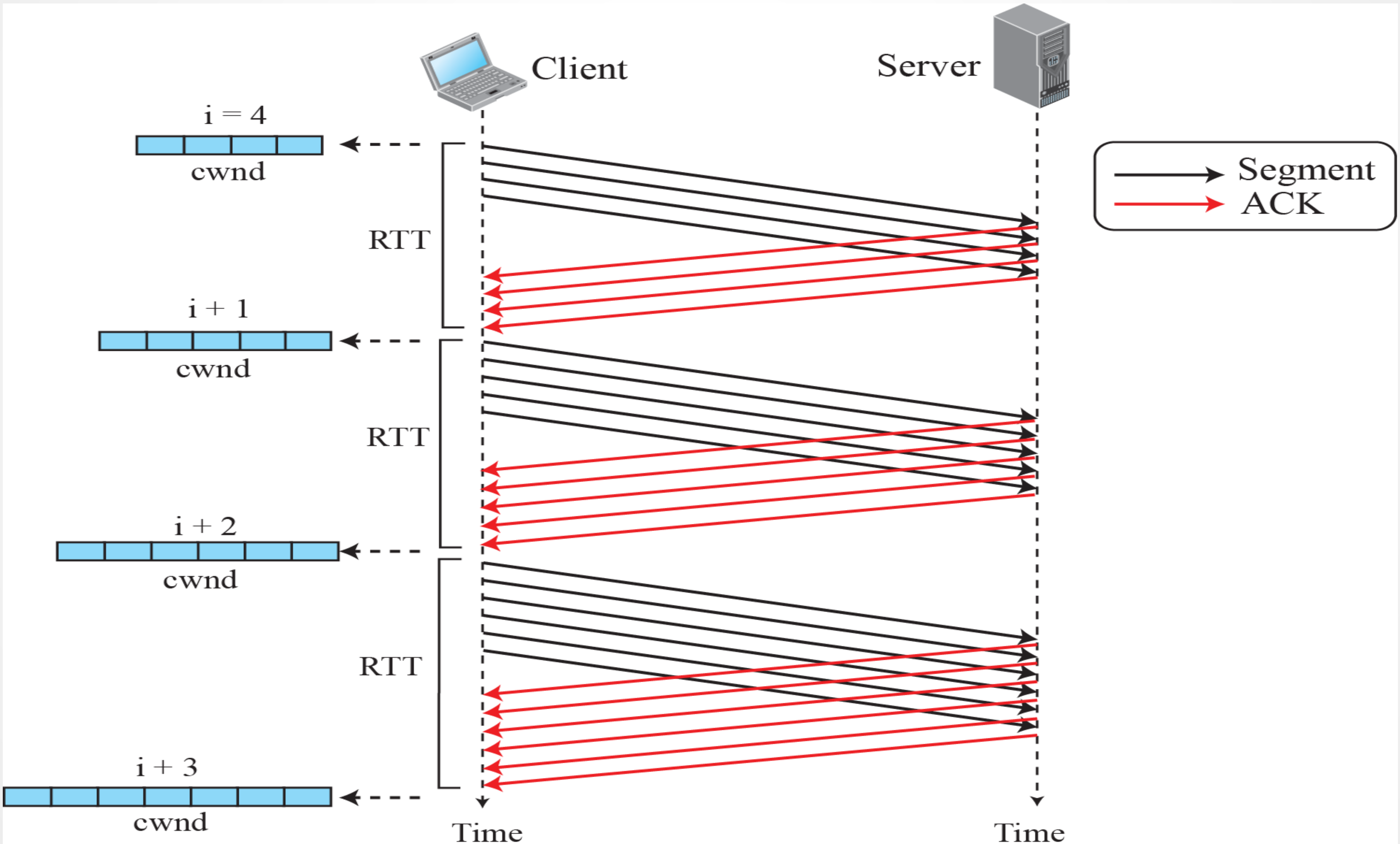
Idea: decrease cwnd when congestion is encountered;
Increase cwnd otherwise.

What value of cwnd to choose initially?

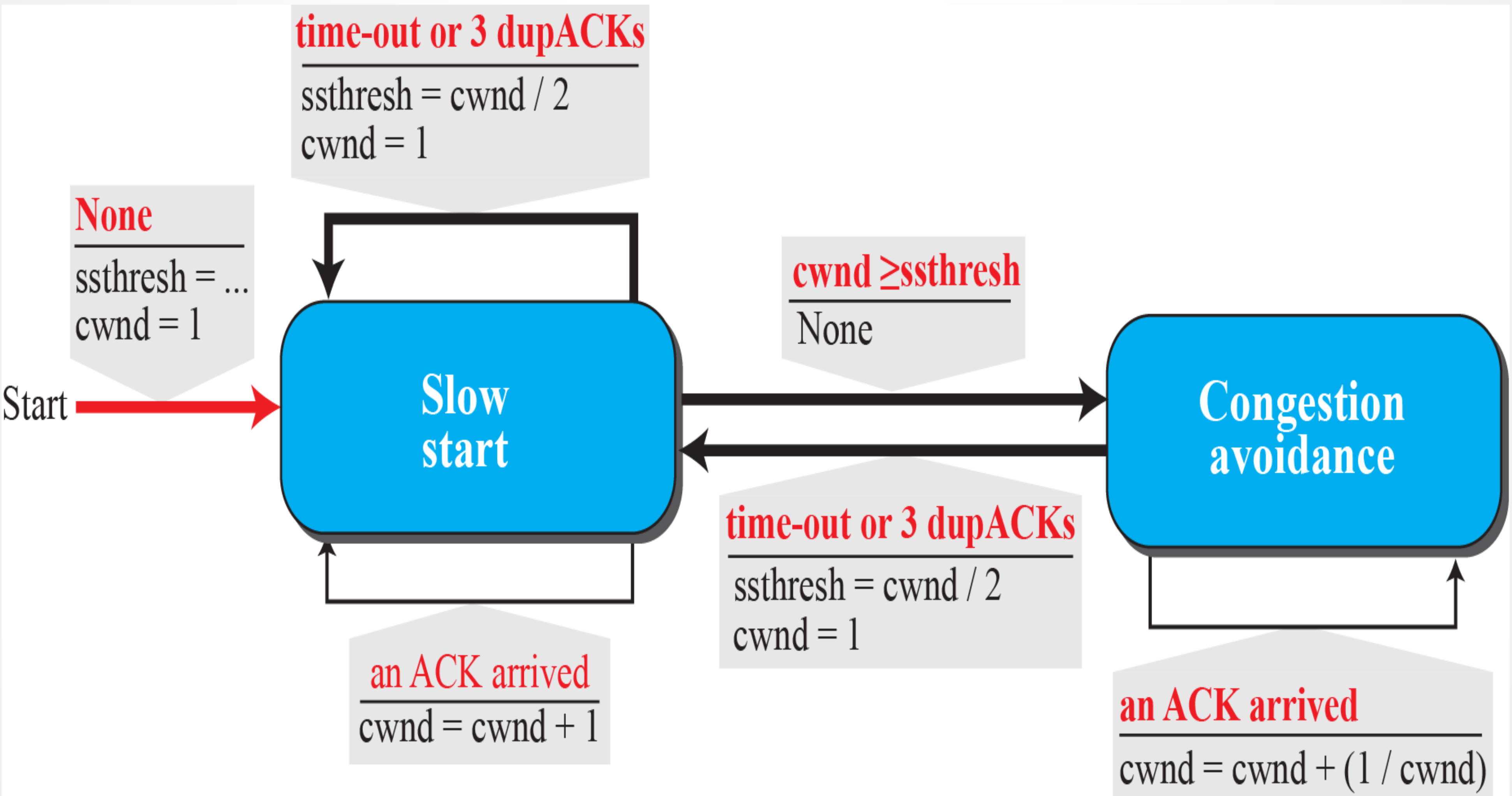
Slow Start (Exponential Increase)



Congestion avoidance (additive increase)



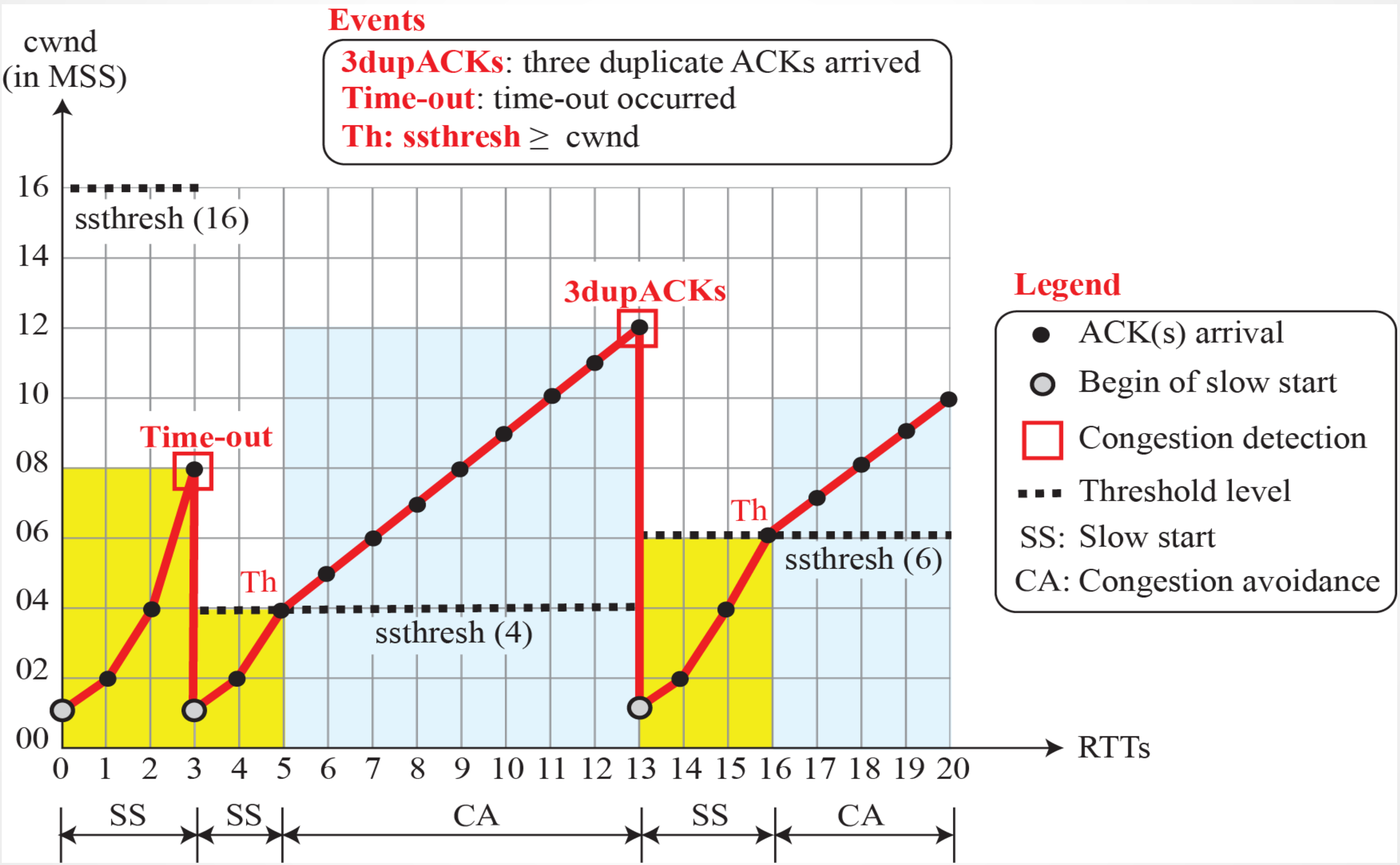
FSM for Taho TCP



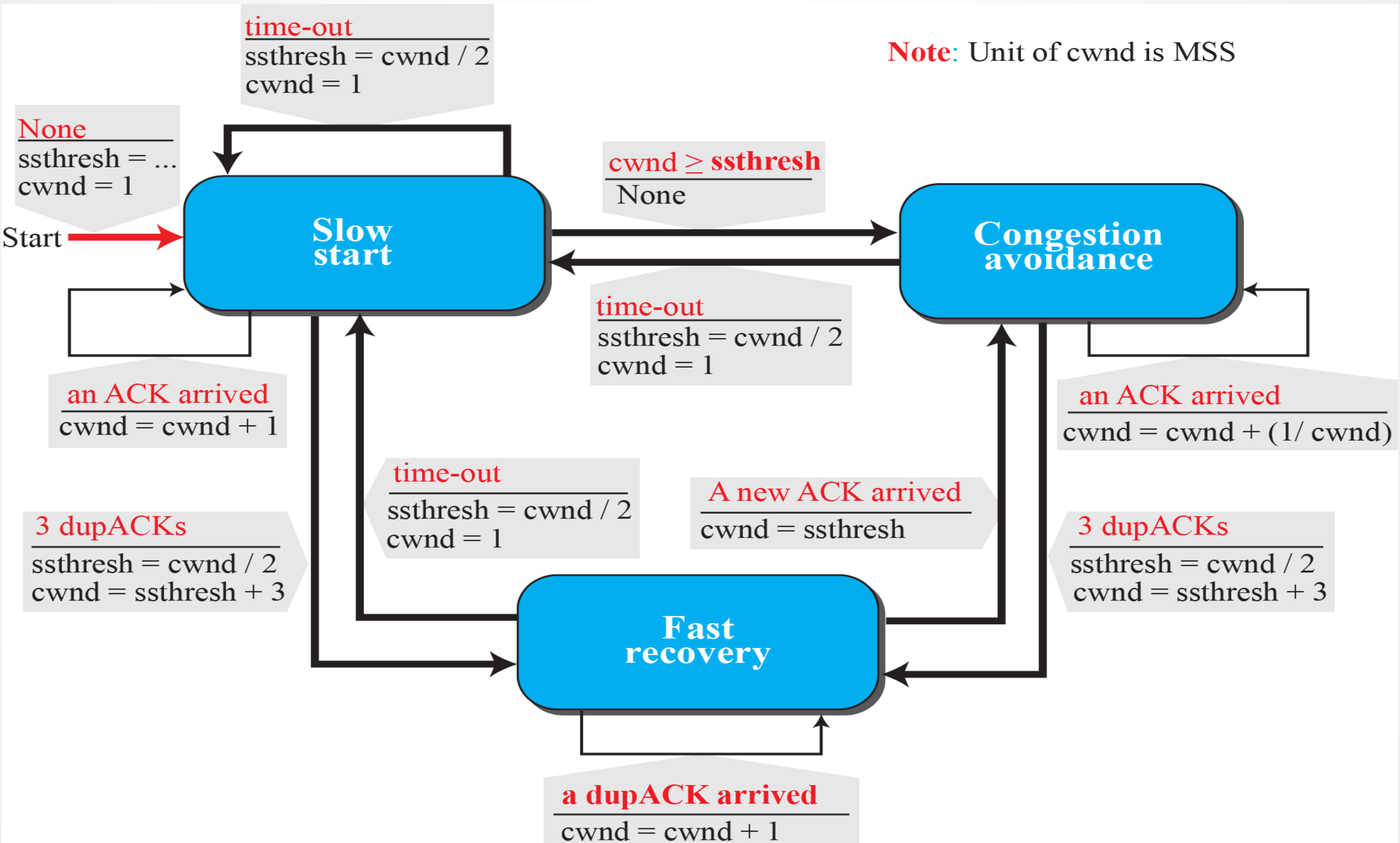
Example of Taho TCP

- TCP starts data transfer with the ssthresh variable value of 16 MSS.
- TCP begins at the slow-start (SS) state with the cwnd = 1 MSS.
- The congestion window grows exponentially, but a time-out occurs after the third RTT (before reaching the threshold).
- TCP assumes that there is congestion in the network.
- It immediately sets the new ssthresh = 4 MSS (half of the current cwnd, which is 8)
- Then begins a new slow start (SA) state with cwnd = 1 MSS.
- The congestion grows exponentially until it reaches the newly set threshold. TCP now moves to the congestion avoidance (CA) state and the congestion window grows additively until it reaches cwnd = 12 MSS.
- At this moment, three duplicate ACKs arrive, another indication of the congestion in the network.
- TCP again halves the value of ssthresh to 6 MSS and begins a new slow-start (SS) state.
- The exponential growth of the cwnd continues. After RTT 15, the size of cwnd is 4 MSS. After sending four segments and receiving only two ACKs, the size of the window reaches the ssthresh (6).
- Hence TCP moves to the congestion avoidance state. The data transfer now continues in the congestion avoidance (CA) state until the connection is terminated after RTT 20.

Example of Taho TCP



FSM for Reno TCP



Example of Reno TCP

- The same example illustrated through Reno TCP.
- The changes in the congestion window are the same until RTT 13 when three duplicate ACKs arrive.
- At this moment, Reno TCP drops the ssthresh to 6 MSS, but it sets the cwnd to a much higher value (ssthresh + 3 = 9 MSS) instead of 1 MSS.
- It now moves to the fast recovery state.
- We assume that two more duplicate ACKs arrive until RTT 15, where cwnd grows exponentially.
- In this moment, a new ACK (not duplicate) arrives that announces the receipt of the lost segment.
- It now moves to the congestion avoidance state, but first deflates the congestion window to 6 MSS as though ignoring the whole fast-recovery state and moving back to the previous track.

Example of Reno TCP