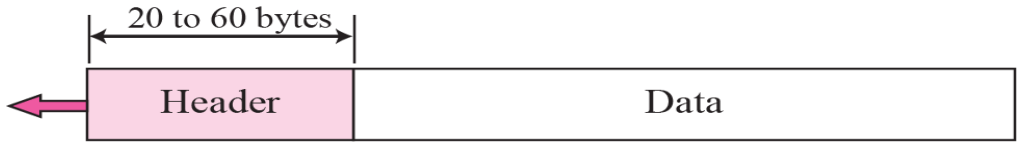
Transport Layer

Anand Baswade

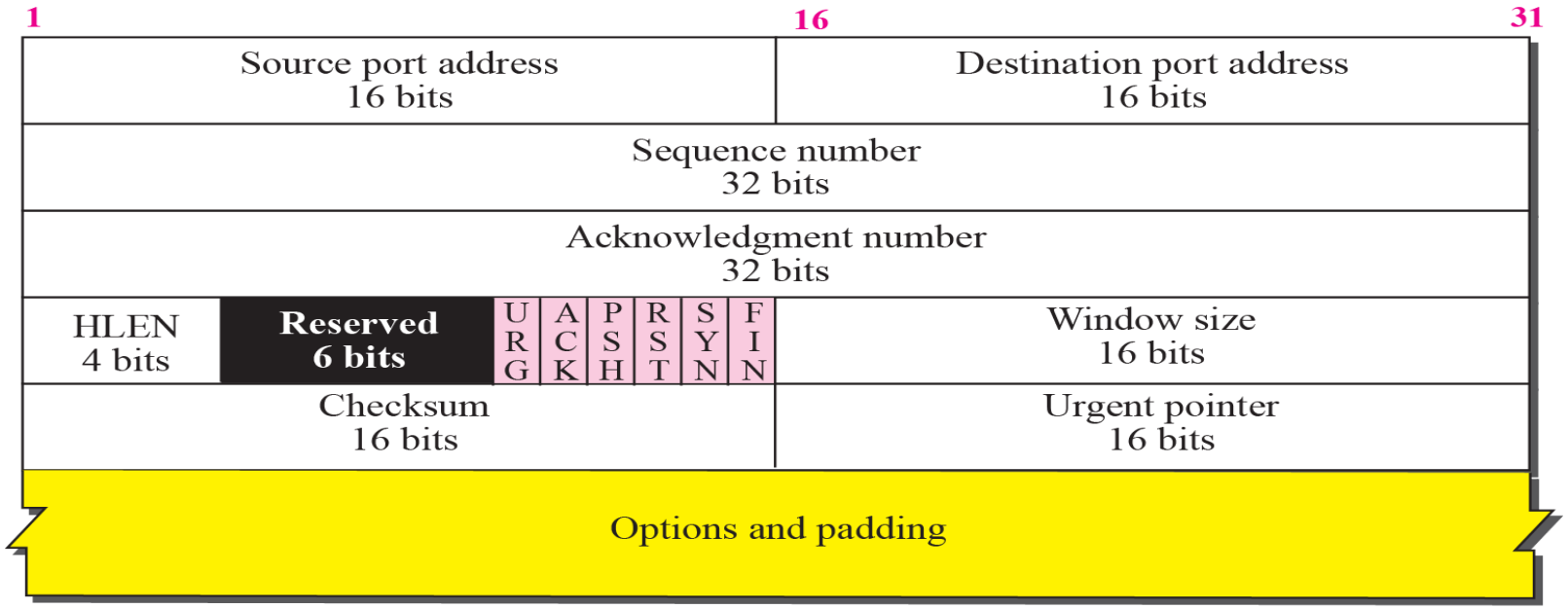
anand@iitbhilai.ac.in

TCP segment format

- Before discussing TCP in more detail, let us discuss the TCP packets themselves. A packet in TCP is called a segment.

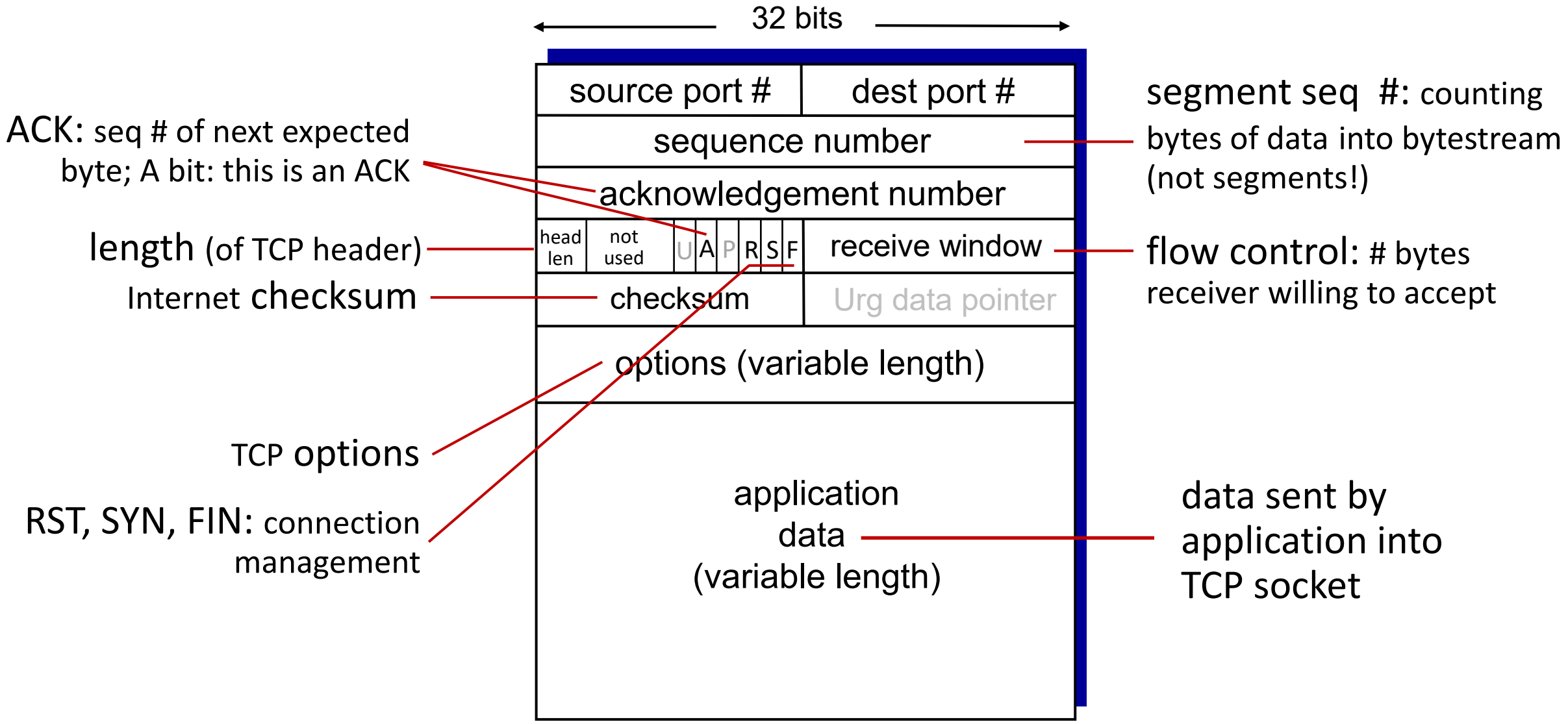


a. Segment



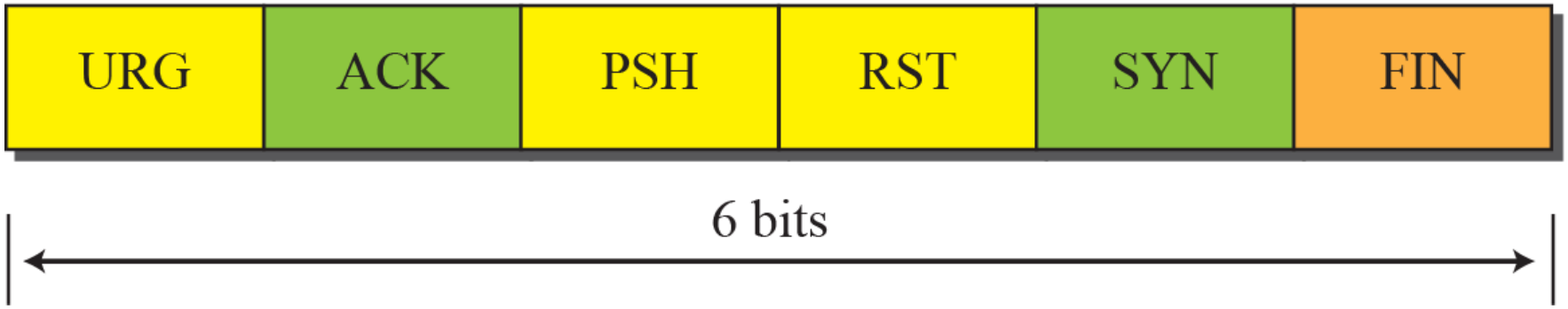
b. Header

TCP segment structure



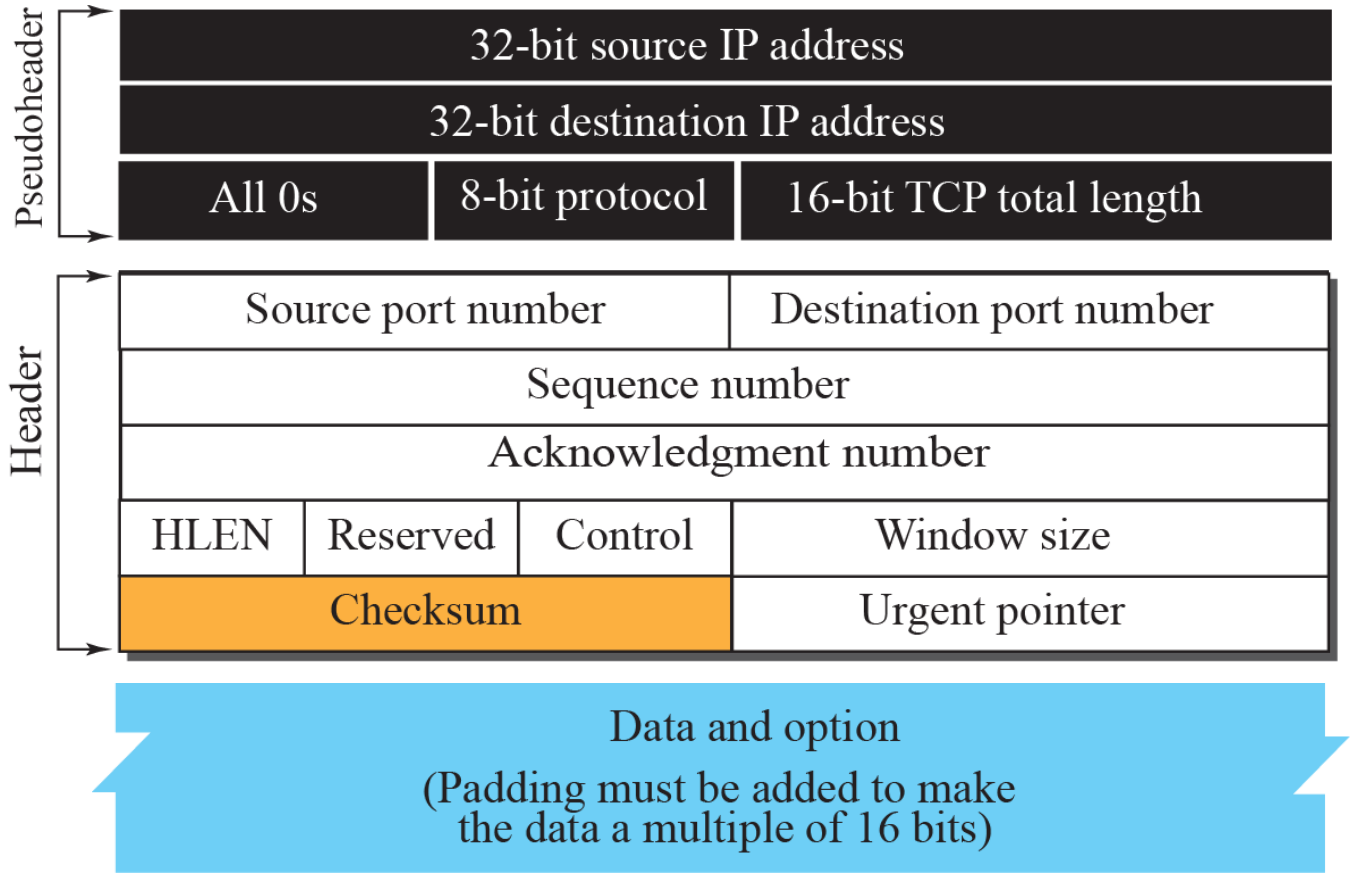
TCP Flag Bits

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



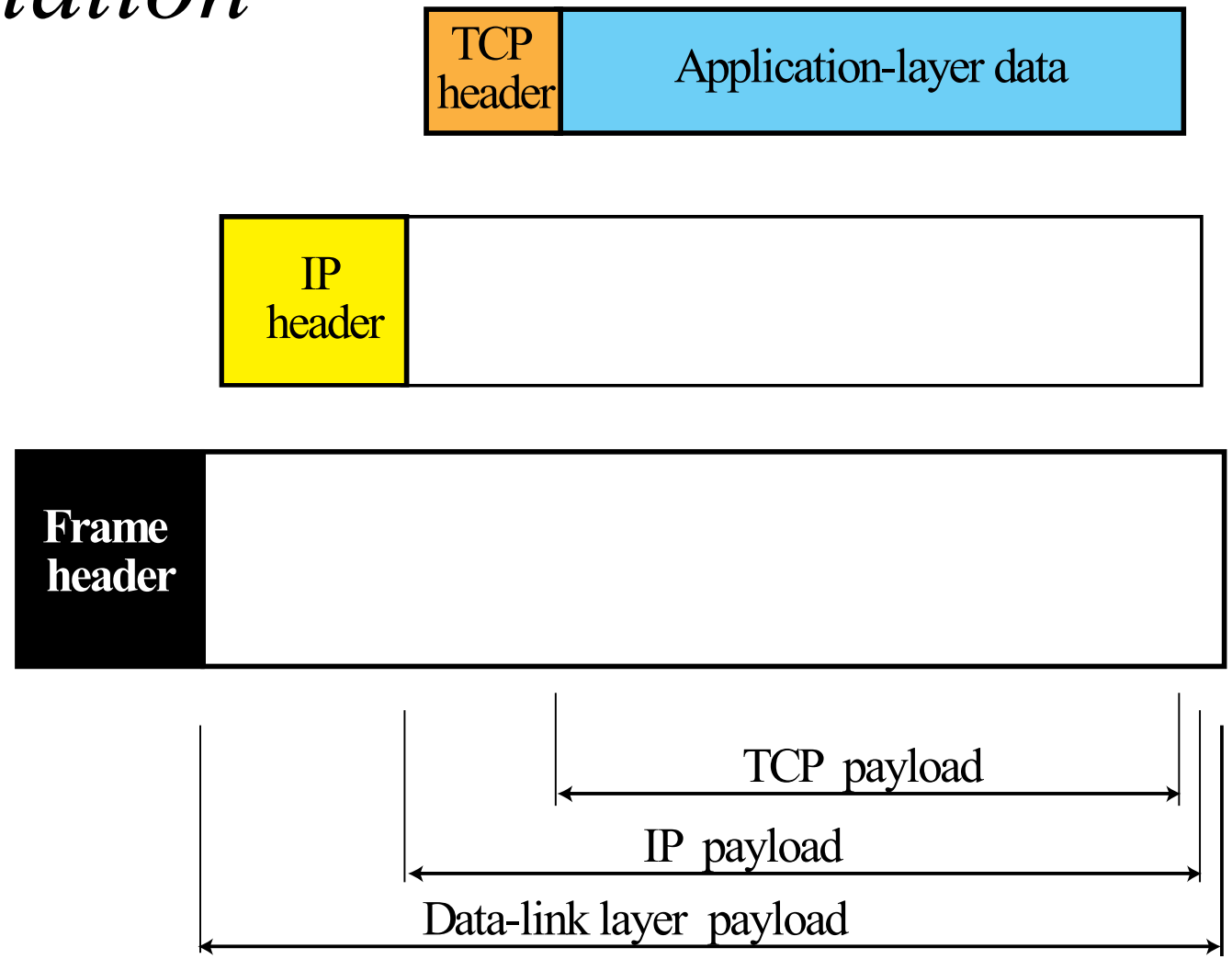
In practice URG and the urgent pointer are not used.

Pseudoheader added to the TCP segment



The use of the checksum in TCP is mandatory.

Encapsulation



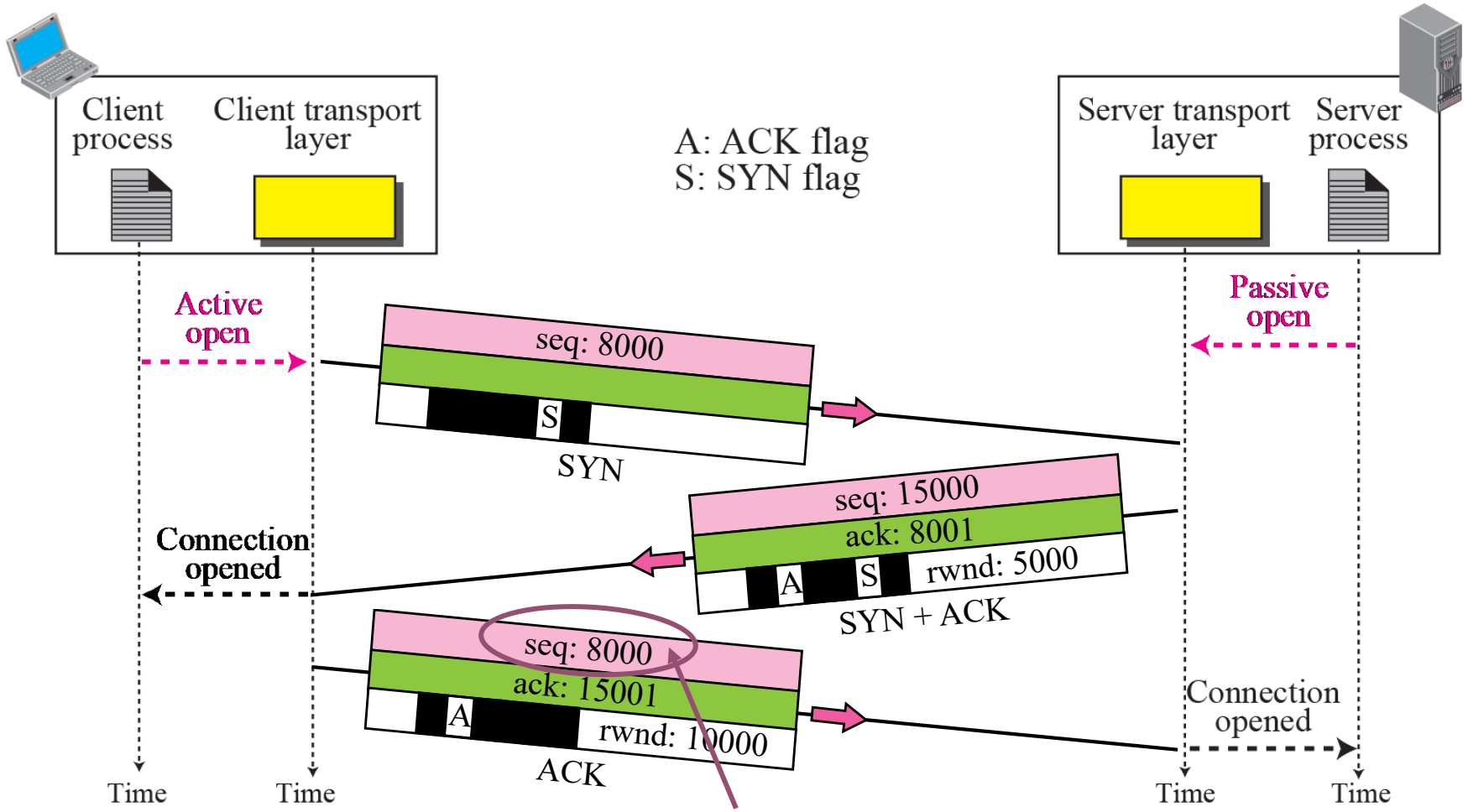
TCP Connection

- TCP is connection-oriented. It establishes a virtual path between the source and destination. All of the segments belonging to a message are then sent over this virtual path.

- You may wonder how TCP, which uses the services of IP, a connectionless protocol, can be connection-oriented. The point is that a TCP connection is virtual, not physical.

- TCP operates at a higher level. TCP uses the services of IP to deliver individual segments to the receiver, but it controls the connection itself. If a segment is lost or corrupted, it is retransmitted.

Connection establishment using three-way handshake



TCP 3-way handshake

Client state

```
clientSocket = socket(AF_INET, SOCK_STREAM)
```

LISTEN

```
clientSocket.connect((serverName,serverPort))
```

choose init seq num, x
send TCP SYN msg

SYNSENT

SYNbit=1, Seq=x

choose init seq num, y
send TCP SYNACK
msg, acking SYN

SYNbit=1, Seq=y
ACKbit=1; ACKnum=x+1

received SYNACK(x)
indicates server is live;
send ACK for SYNACK;
this segment may contain
client-to-server data

ESTAB

ACKbit=1, ACKnum=y+1

received ACK(y)
indicates client is live

Server state

```
serverSocket = socket(AF_INET,SOCK_STREAM)
serverSocket.bind(('',serverPort))
serverSocket.listen(1)
connectionSocket, addr = serverSocket.accept()
```

LISTEN

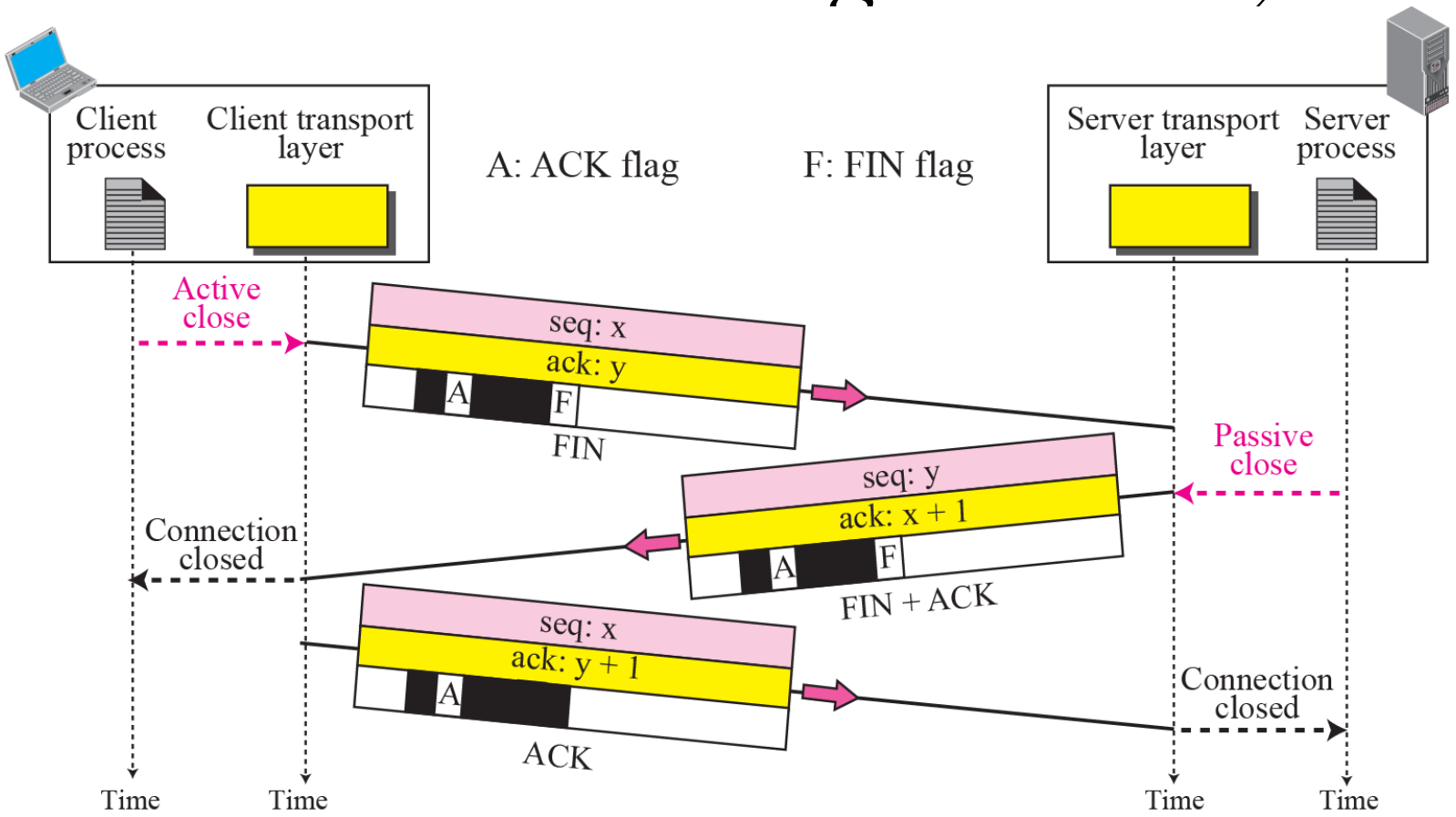
SYN RCVD

ESTAB

Cont..

- A SYN segment cannot carry data, but it consumes one sequence number.
- A SYN + ACK segment cannot carry data, but does consume one sequence number.
- An ACK segment, if carrying no data, consumes no sequence number.

Connection termination using three-way handshake

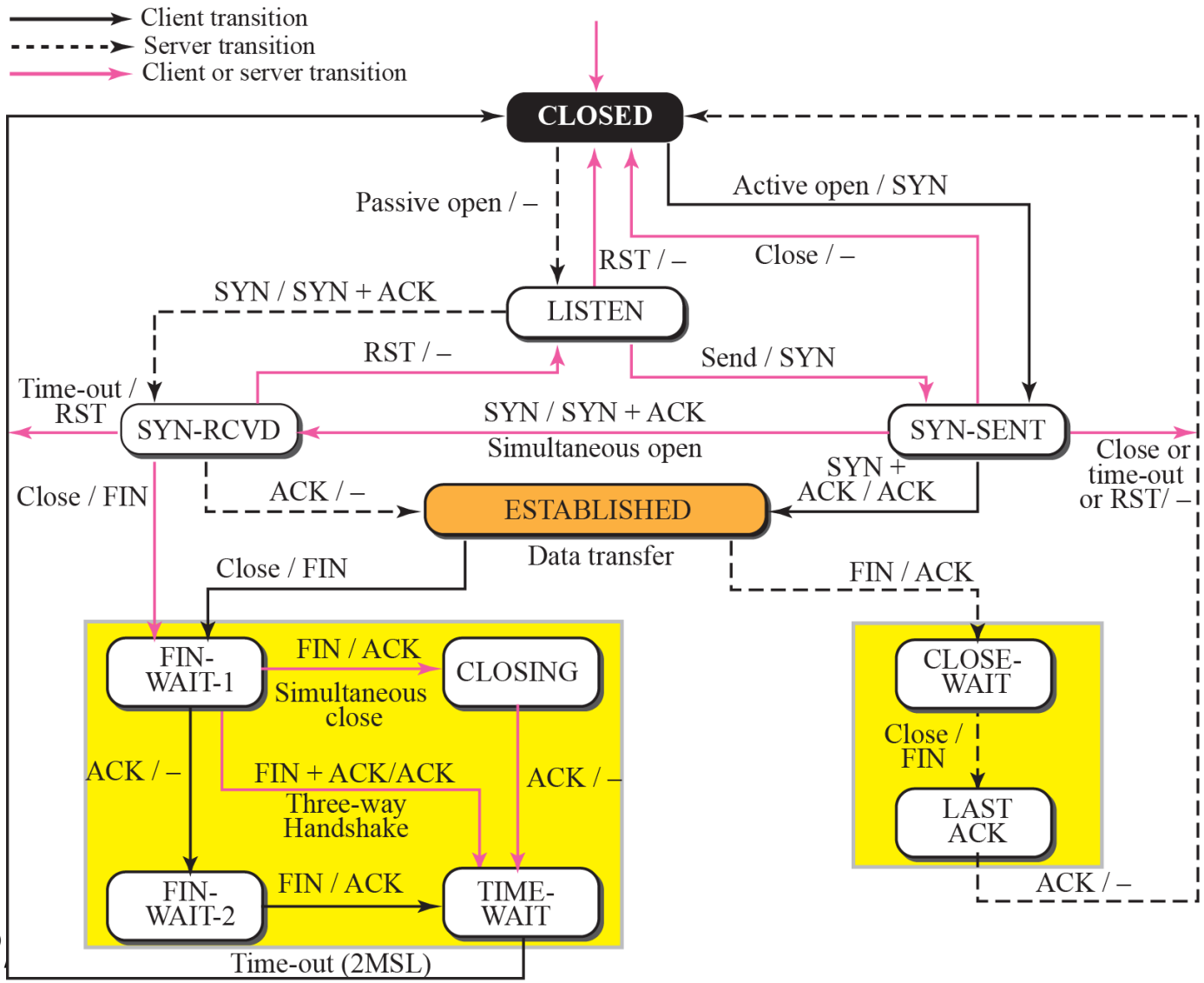


- The FIN segment consumes one sequence number if it does not carry data.
- The FIN + ACK segment consumes one sequence number if it does not carry data.

Closing a TCP connection

- client, server each close their side of connection
  - send TCP segment with FIN bit = 1
- respond to received FIN with ACK
  - on receiving FIN, ACK can be combined with own FIN
- simultaneous FIN exchanges can be handled

State transition diagram

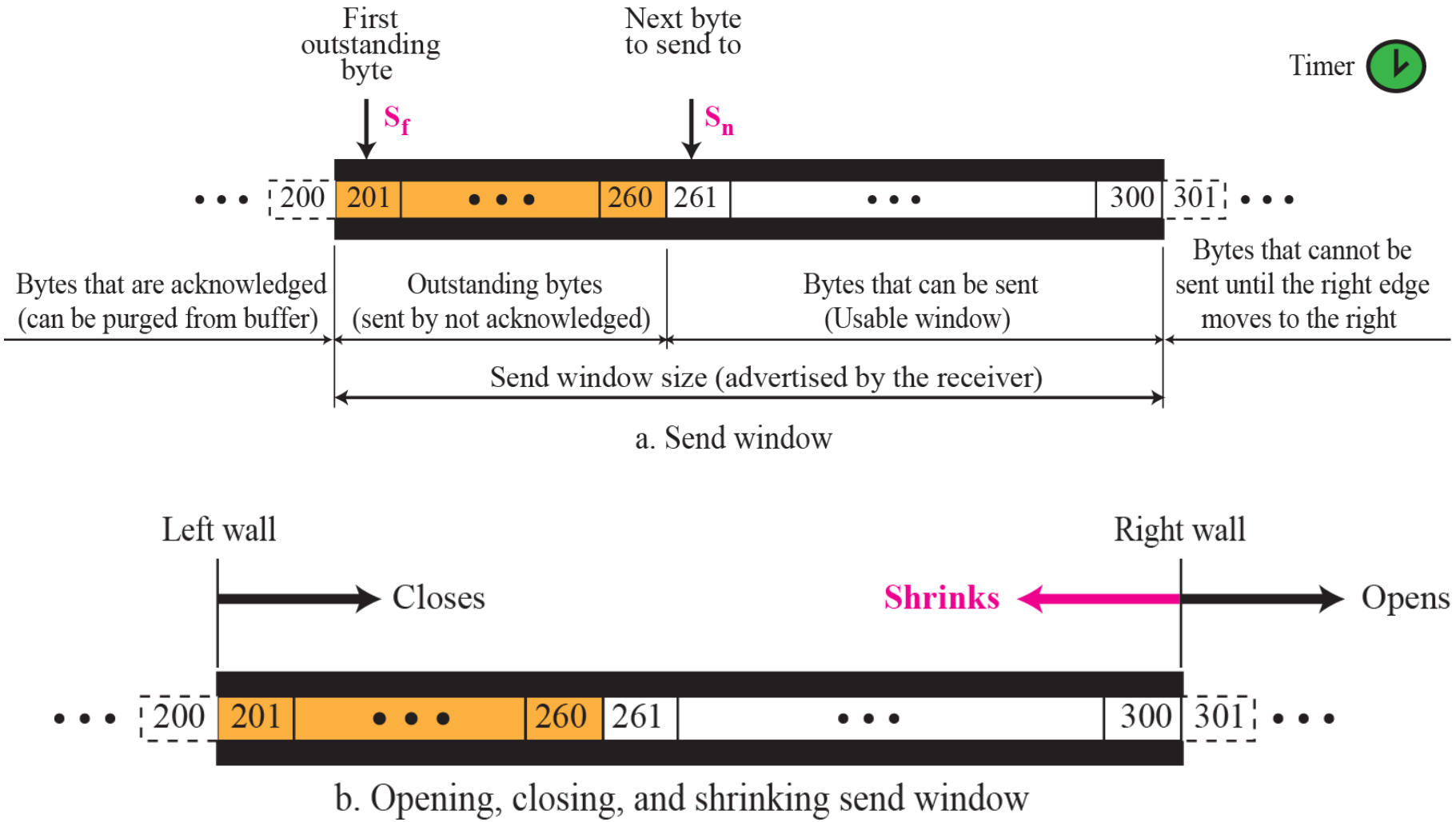


States of TCP

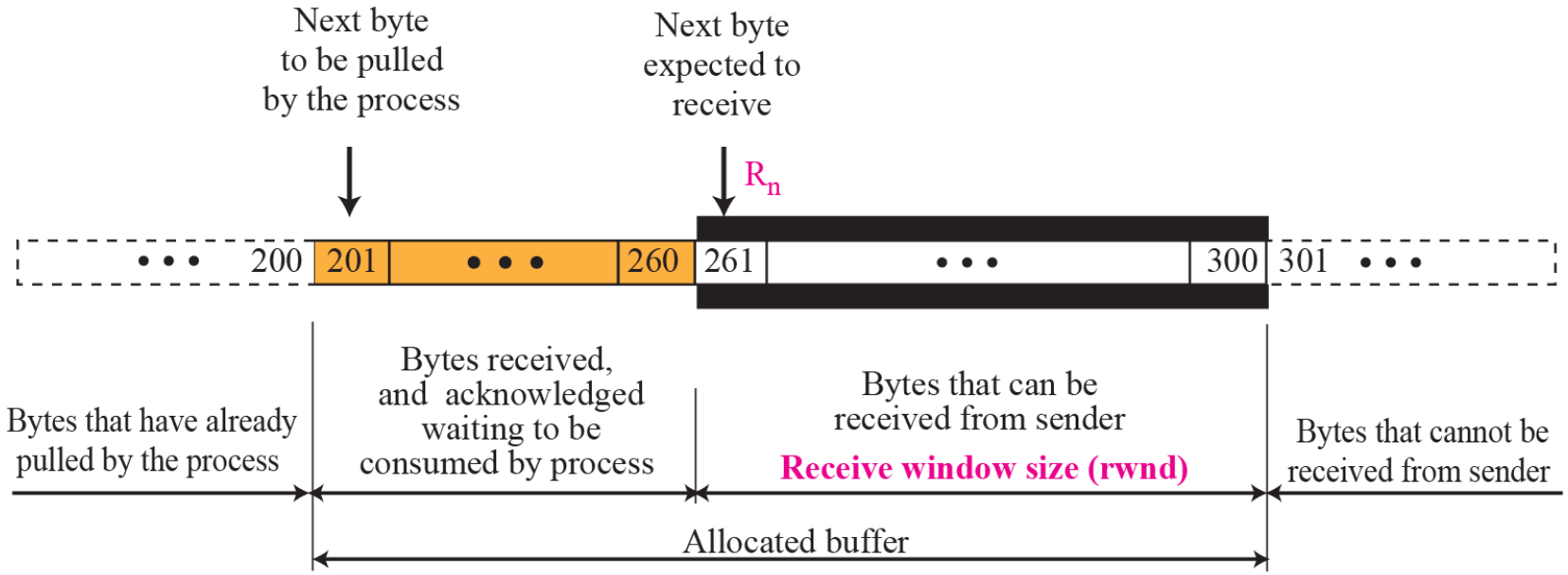
Table 15.2 *States for TCP*

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

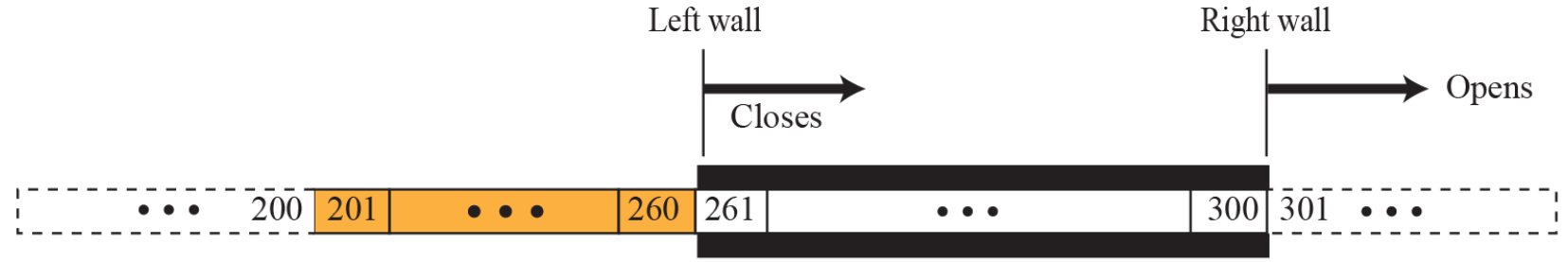
Windows in TCP: *Send window in TCP*



Receive window in TCP



a. Receive window and allocated buffer



b. Opening and closing of receive window

Silly Window Syndrome (1)

➢ Sending data in very small segments

1. Syndrome created by the Sender

- **Sending application program creates data slowly (e.g. 1 byte at a time)**
- Wait and collect data to send in a larger block
- How long should the sending TCP wait?
- Solution: Nagle's algorithm
  - When data come into the sender in small pieces, just send the first piece and buffer all the rest until the first piece is acknowledged.
  - Then send all buffered data in one TCP segment and start buffering again until the next segment is acknowledged.

Silly Window Syndrome (2)

2. Syndrome created by the Receiver
   - Receiving application program consumes data slowly (e.g. 1 byte at a time)
   - The receiving TCP announces a window size of 1 byte. The sending TCP sends only 1 byte...
   - Solution 1: Clark's solution
   - Sending an ACK but announcing a window size of zero until there is enough space to accommodate a segment of max. size or until half of the buffer is empty

Silly Window Syndrome (3)

- Solution 2: Delayed Acknowledgement
- The receiver waits until there is decent amount of space in its incoming buffer before acknowledging the arrived segments
- The delayed acknowledgement prevents the sending TCP from sliding its window. It also reduces traffic.
- Disadvantage: it may force the sender to retransmit the unacknowledged segments
- To balance: should not be delayed by more than 500ms

TCP Sender (simplified)

event: data received from application

- create segment with seq #
- seq # is byte-stream number of first data byte in segment
- start timer if not already running
  - think of timer as for oldest unACKed segment
  - expiration interval: **`TimeOutInterval`**

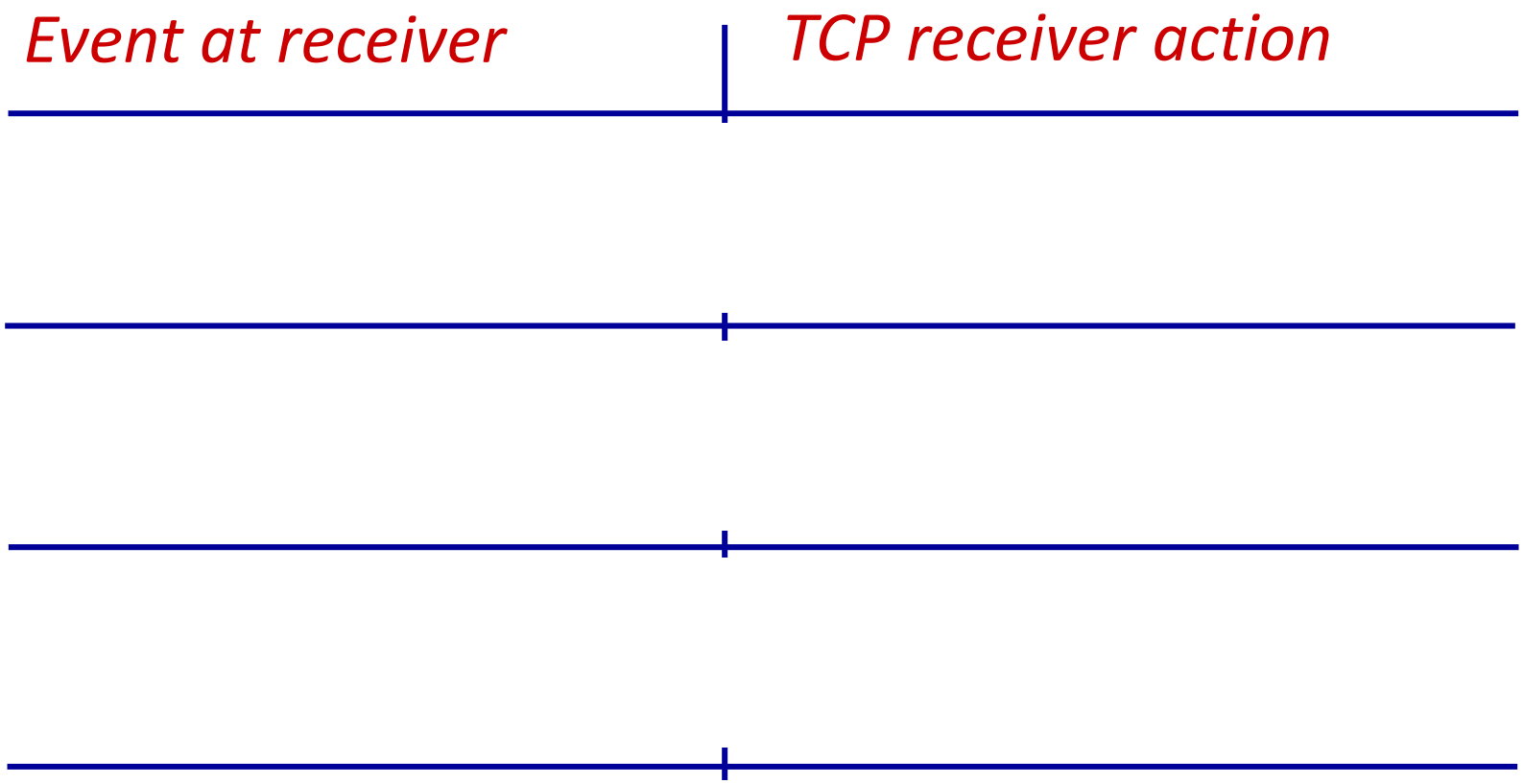
event: timeout

- retransmit segment that caused timeout
- restart timer

event: ACK received

- if ACK acknowledges previously unACKed segments
  - update what is known to be ACKed
  - start timer if there are still unACKed segments

TCP Receiver: ACK generation [RFC 5681]



Rules for Generating the ACKs

1. When one end sends a data segment to the other end, it must include an ACK. That gives the next sequence number it expects to receive. (Piggyback)

2. The receiver needs to delay sending (until another segment arrives or 500ms) an ACK segment if there is only one outstanding in-order segment. It prevents ACK segments from creating extra traffic.

3. There should not be more than 2 in-order unacknowledged segments at any time. It prevent the unnecessary retransmission

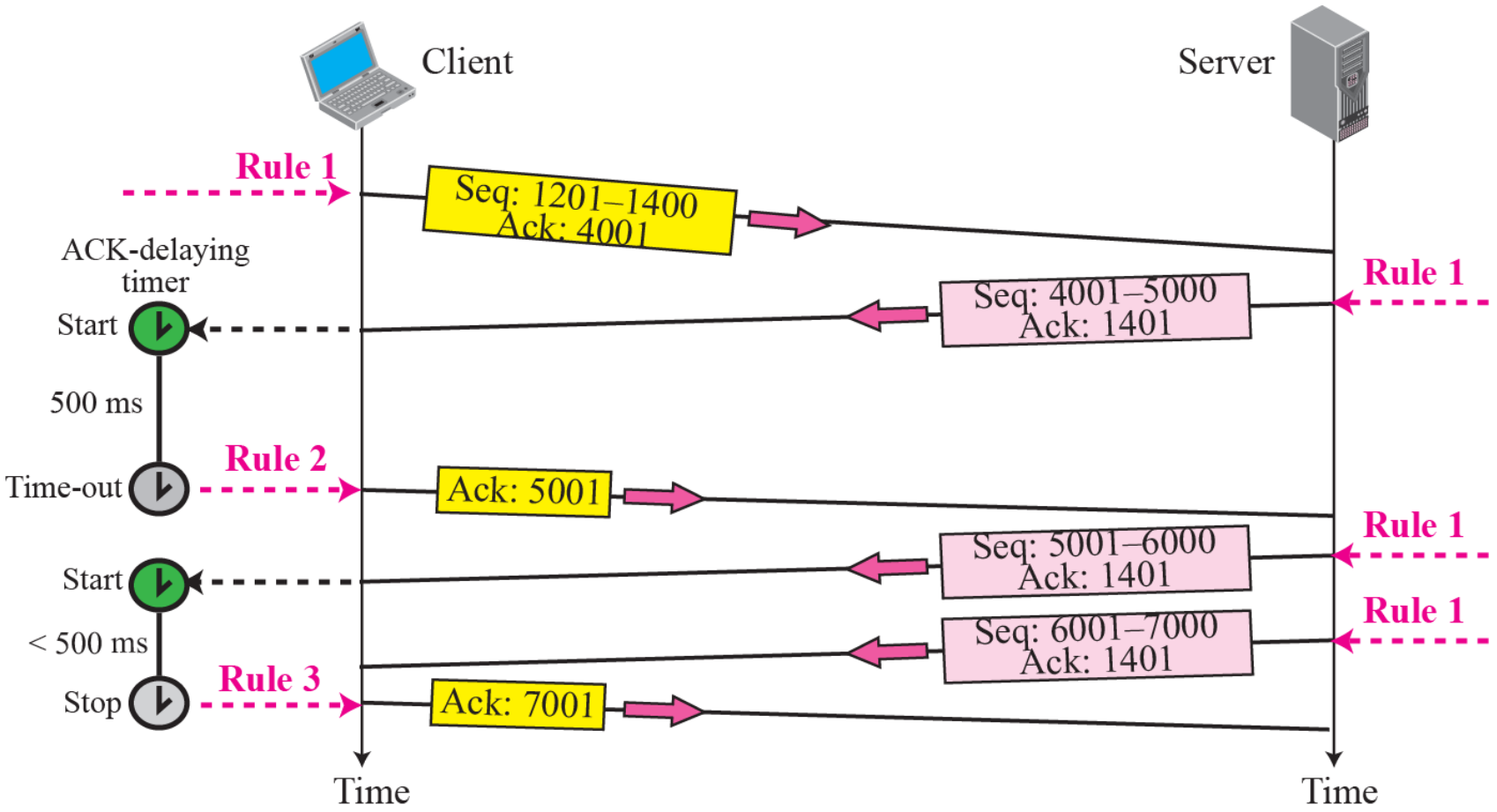
Rules for Generating the ACKs Cont..

4. When a segment arrives with an out-of-order sequence number that is higher than expected, the receiver immediately sends an ACK segment announcing the sequence number of the next expected segment. (for fast retransmission)

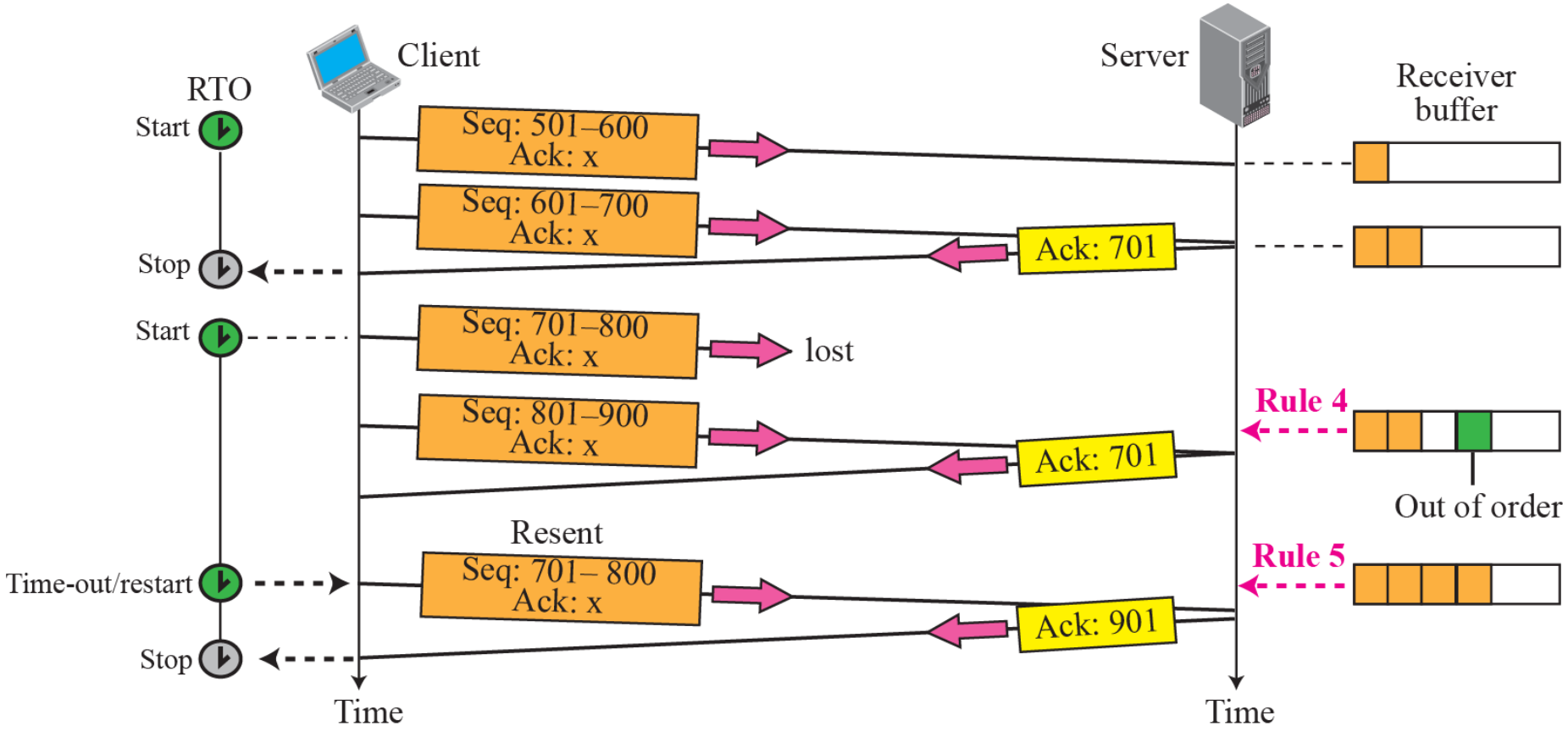
5. When a missing segment arrives, the receiver sends an ACK segment to announce the next sequence number expected.

6. If a duplicate segment arrives, the receiver immediately sends an ACK.

Some Scenarios: Normal operation



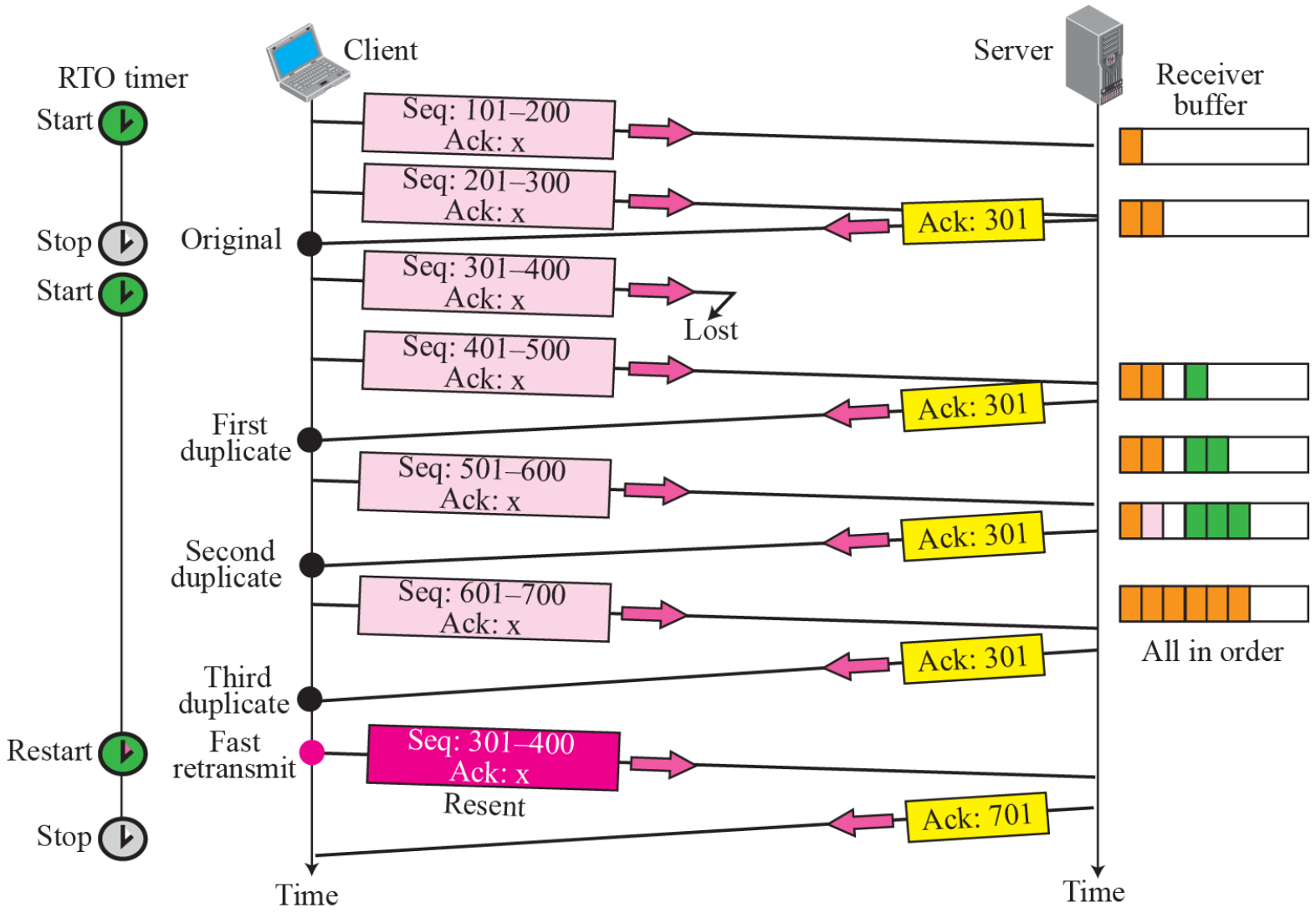
Lost segment



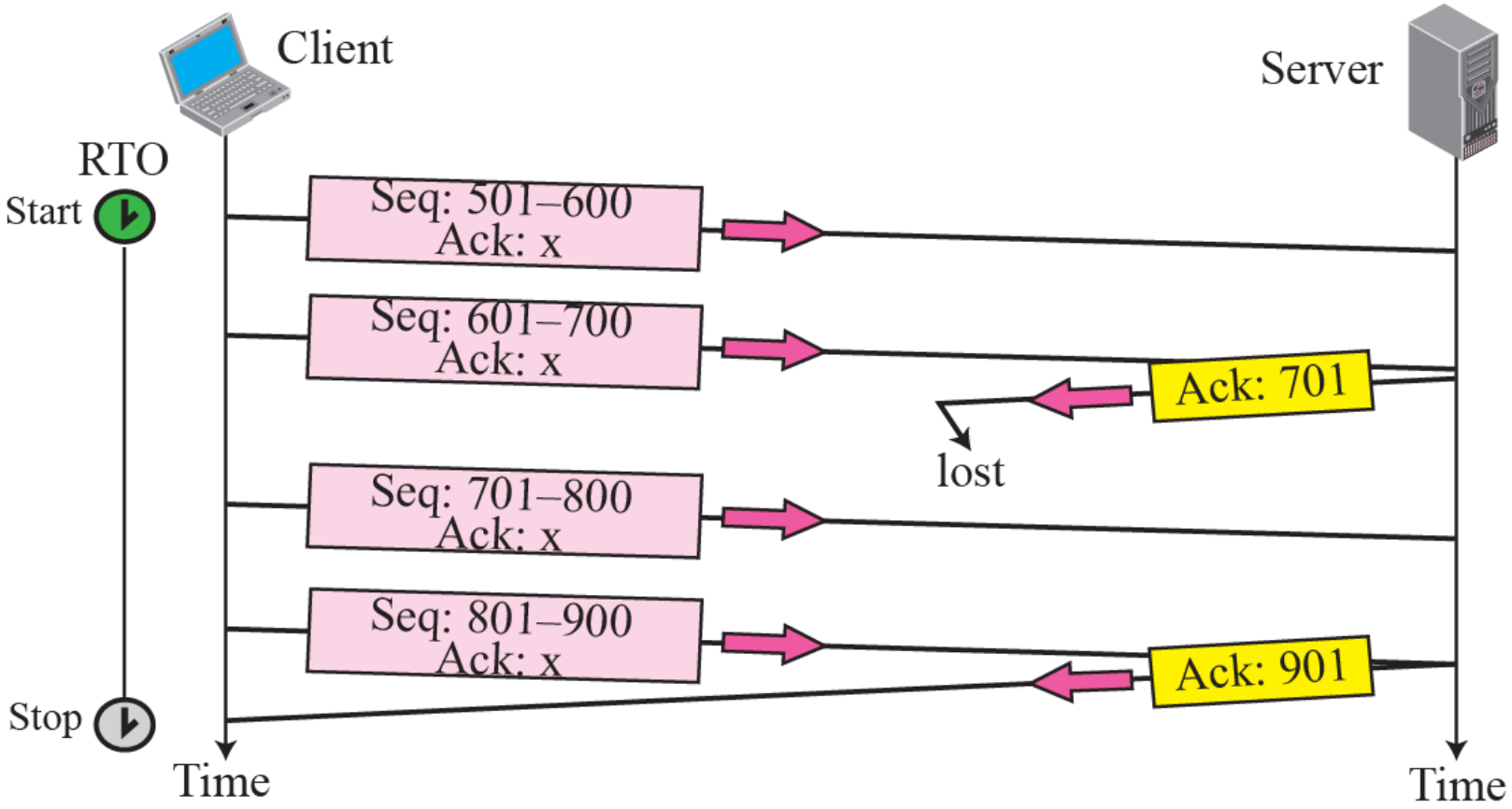
The receiver TCP delivers only ordered data to the process.

Fast retransmission

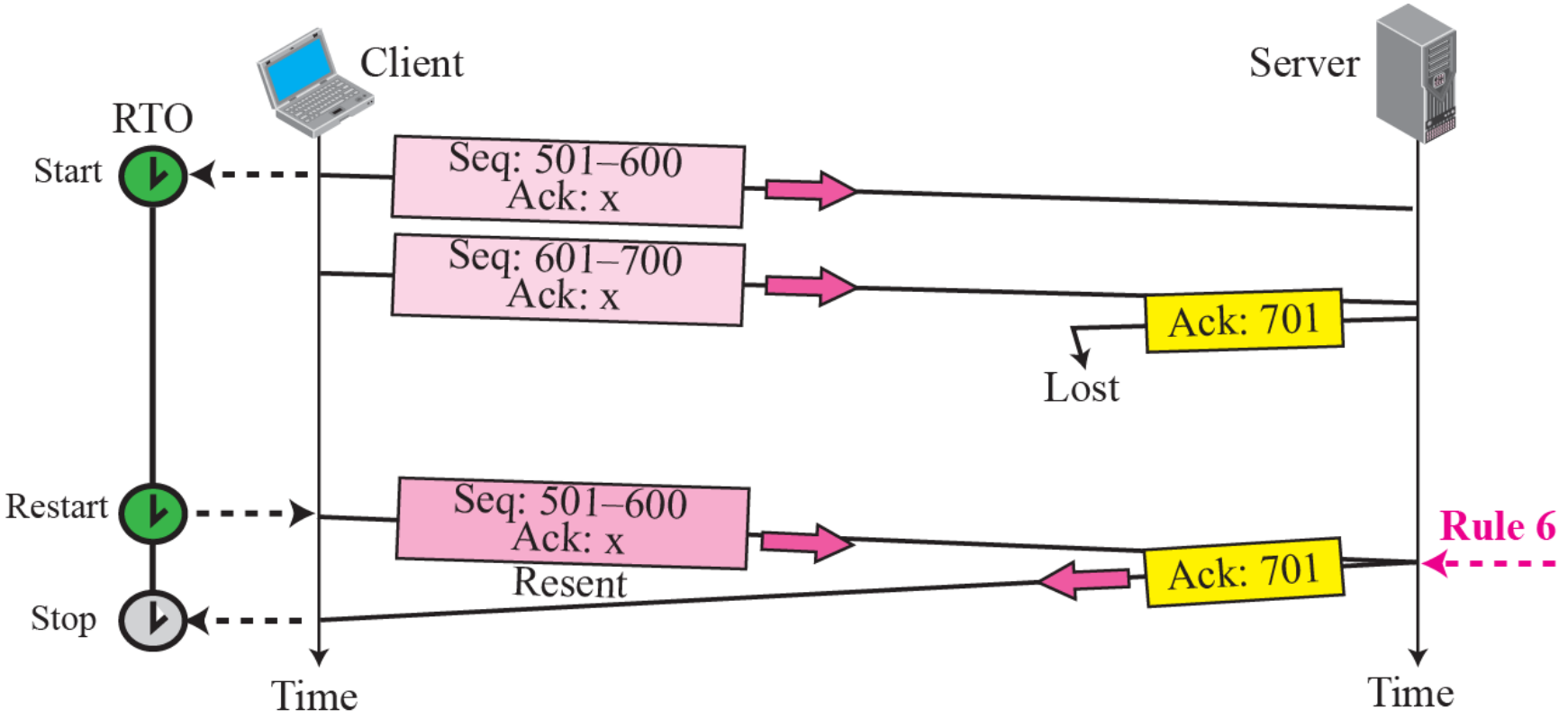
Receipt of three duplicate ACKs indicates 3 segments received after a missing segment – lost segment is likely. So retransmit!



Lost acknowledgment



Lost acknowledgment corrected by resending a segment



ACK and Out of Order Handling in TCP

Acknowledgement in TCP – Cumulative acknowledgement

Receiver has received bytes 0, 1, 2, _, 4, 5, 6, 7

- TCP sends a cumulative acknowledgement with ACK number 3, acknowledging everything up to byte 2
- Once 4 is received, a duplicate ACK with ACK number 3 (next expected byte) is forwarded
- After timeout, sender retransmits byte 3
- Once byte 3 is received, it can send another cumulative ACK with ACK number 8 (next expected byte)