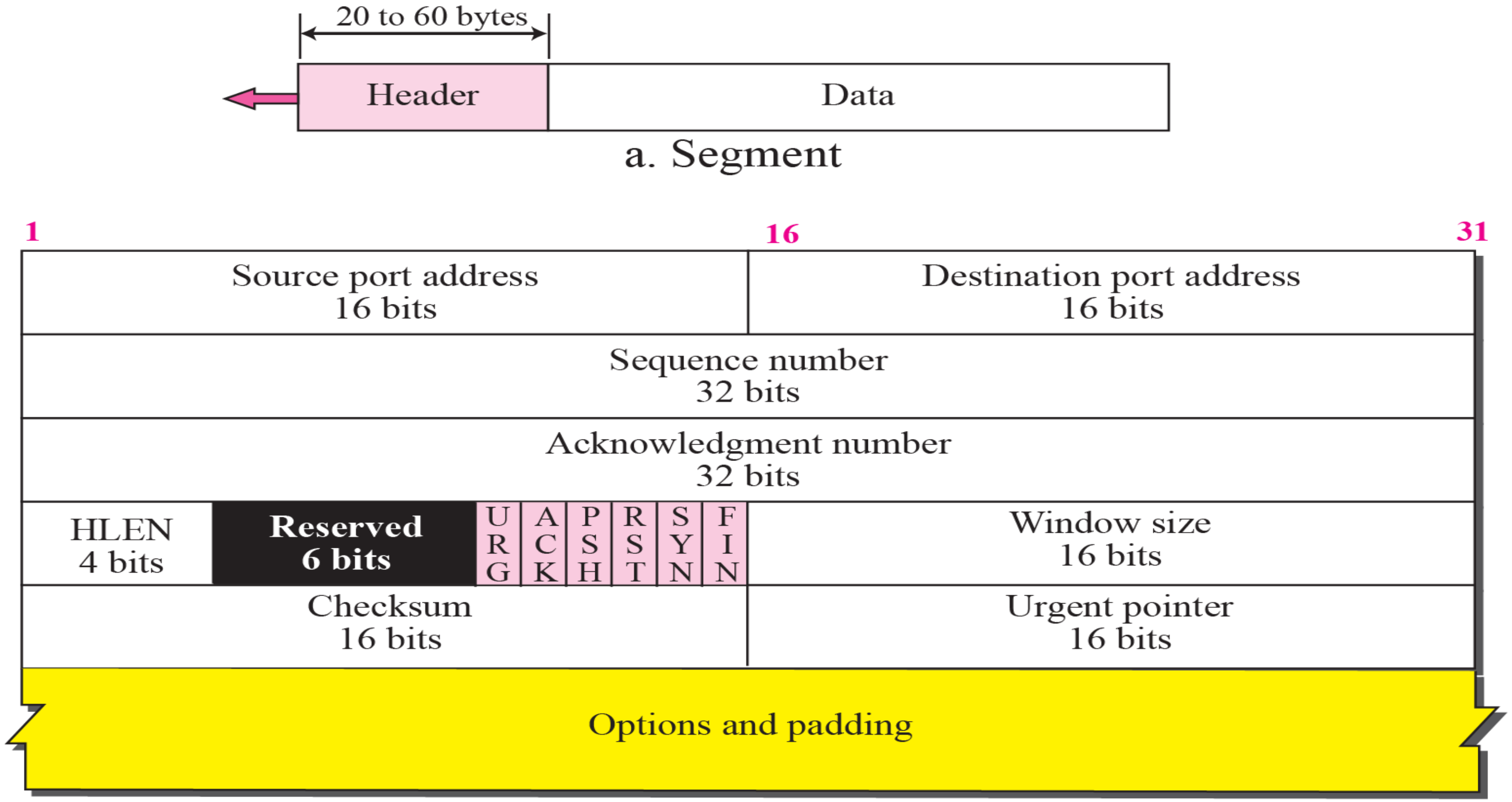
Transport Layer

Anand Baswade

anand@iitbhilai.ac.in

TCP segment format

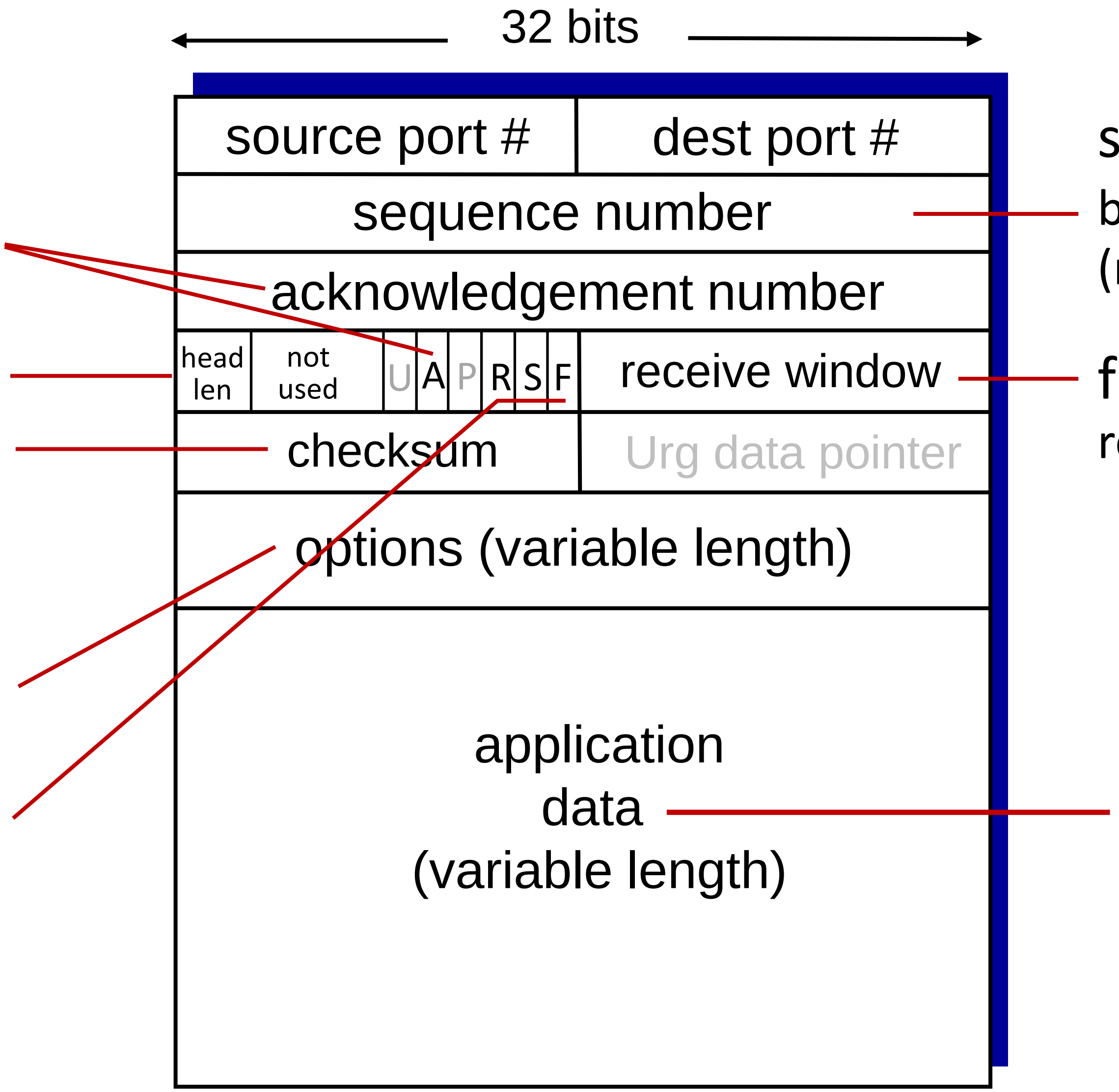
- Before discussing TCP in more detail, let us discuss the TCP packets themselves. A packet in TCP is called a segment.



a. Segment

b. Header

TCP segment structure



32 bits

source port # | dest port #

sequence number

acknowledgement number

head len | not used | U | A | P | R | S | F | receive window

checksum | Urg data pointer

options (variable length)

application data (variable length)

ACK: seq # of next expected byte; A bit: this is an ACK

length (of TCP header)

Internet **checksum**

TCP **options**

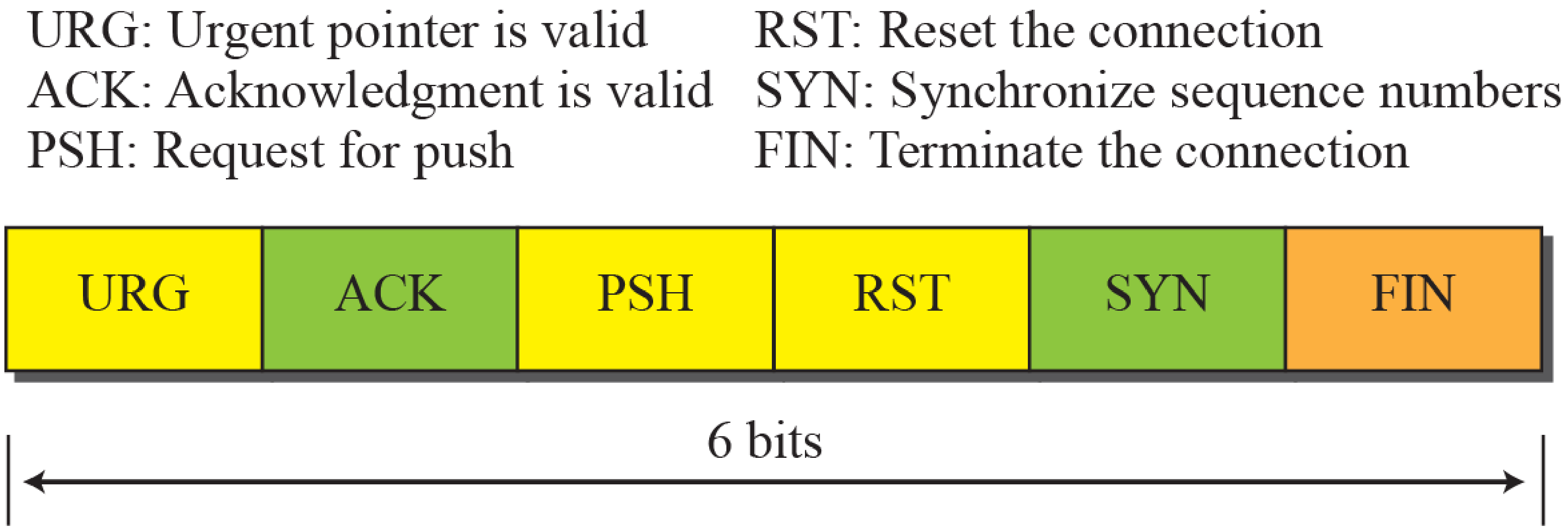
RST, SYN, FIN: connection management

segment seq #: counting bytes of data into bytestream (not segments!)

flow control: # bytes receiver willing to accept

data sent by application into TCP socket

TCP Flag Bits

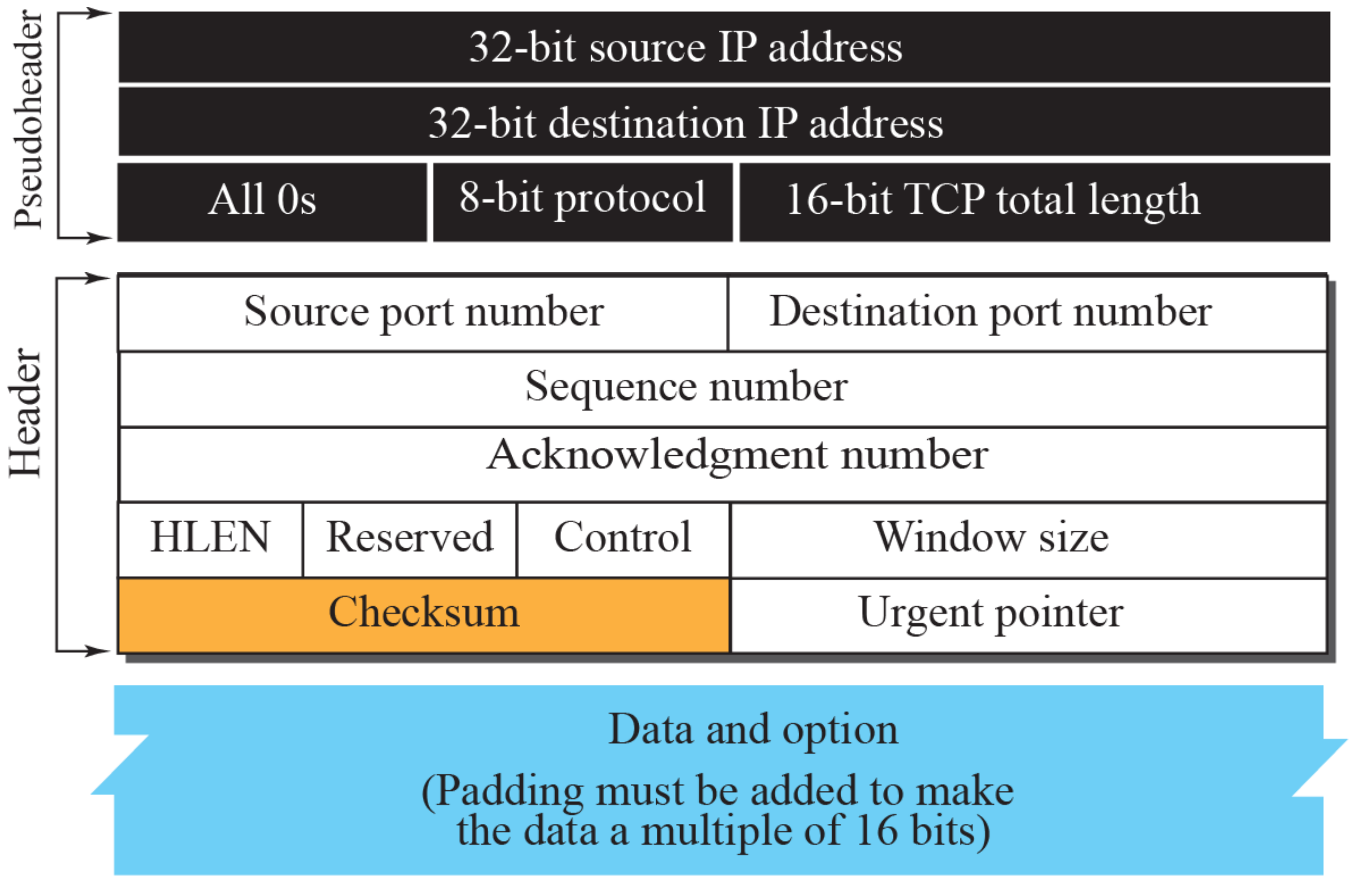


URG: Urgent pointer is valid
ACK: Acknowledgment is valid
PSH: Request for push

RST: Reset the connection
SYN: Synchronize sequence numbers
FIN: Terminate the connection

In practice, the PSH, URG, and the urgent pointer are not used.

Pseudoheader added to the TCP segment



Pseudoheader

32-bit source IP address

32-bit destination IP address

All 0s | 8-bit protocol | 16-bit TCP total length

Header

Source port number | Destination port number

Sequence number

Acknowledgment number

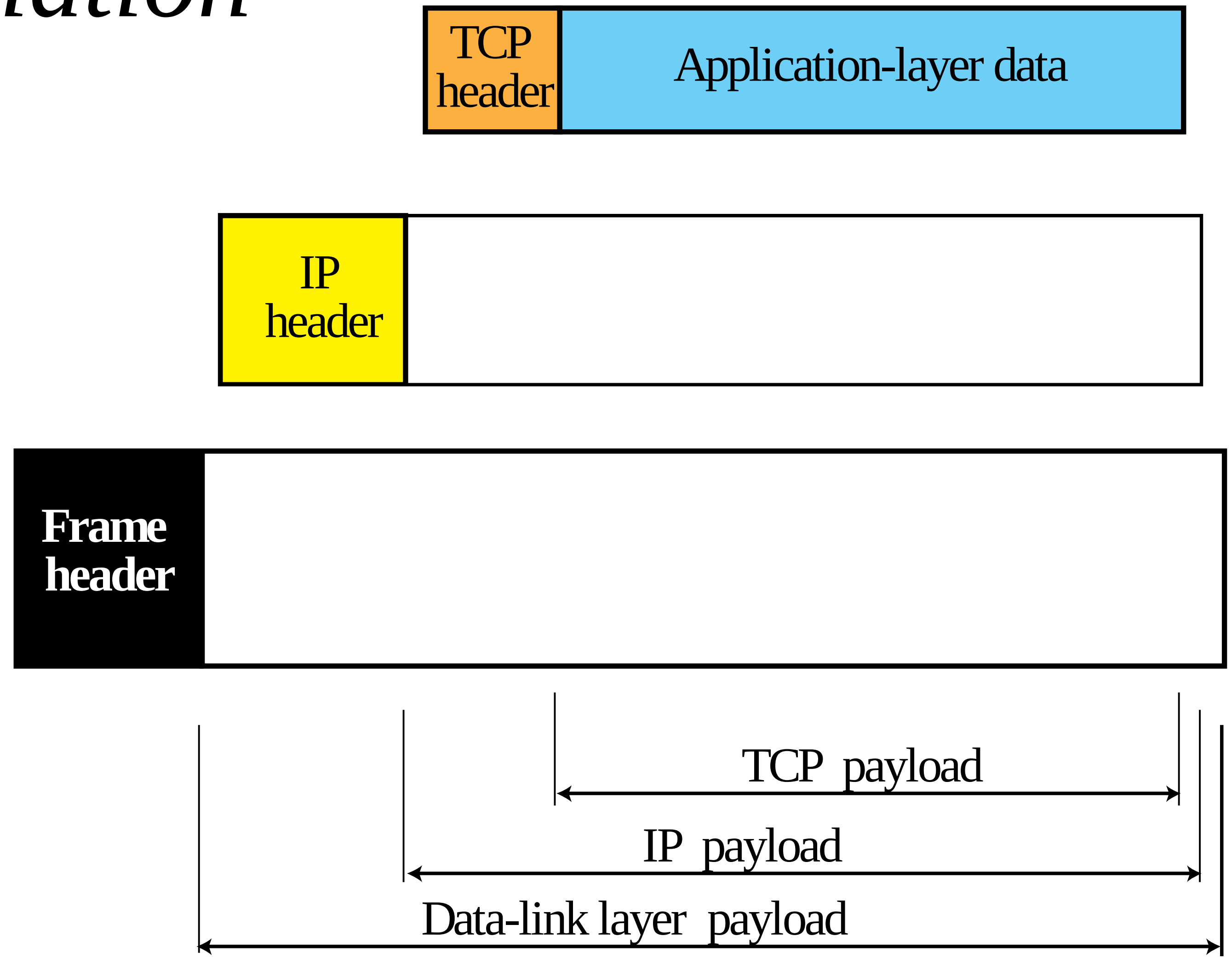
HLEN | Reserved | Control | Window size

Checksum | Urgent pointer

Data and option
(Padding must be added to make
the data a multiple of 16 bits)

The use of the checksum in TCP is mandatory.

Encapsulation



TCP/IP Protocol Suite

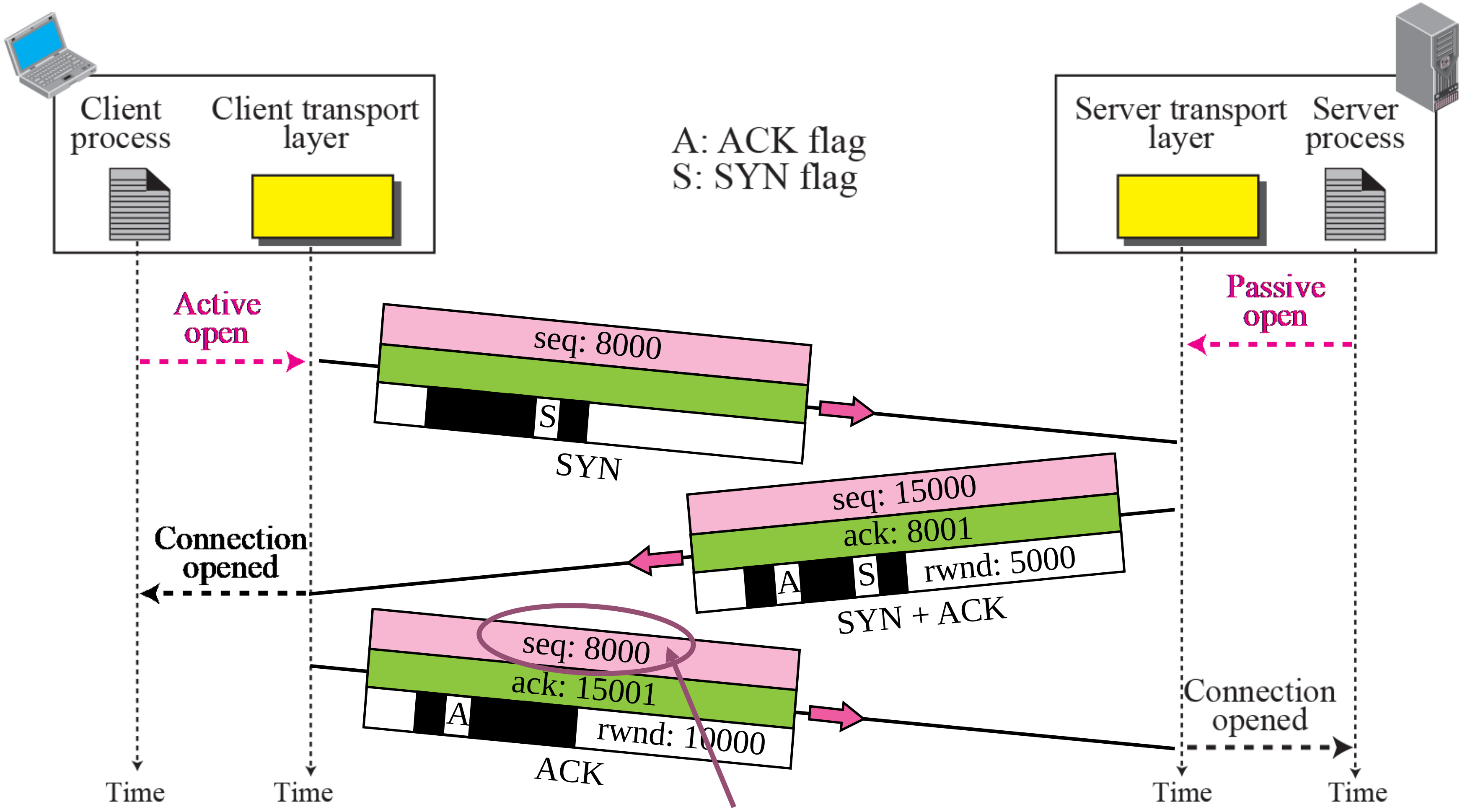
TCP Connection

- 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



Means "no data" !

seq: 8001 if piggybacking

TCP/IP Protocol Suite

TCP 3-way handshake

Client state

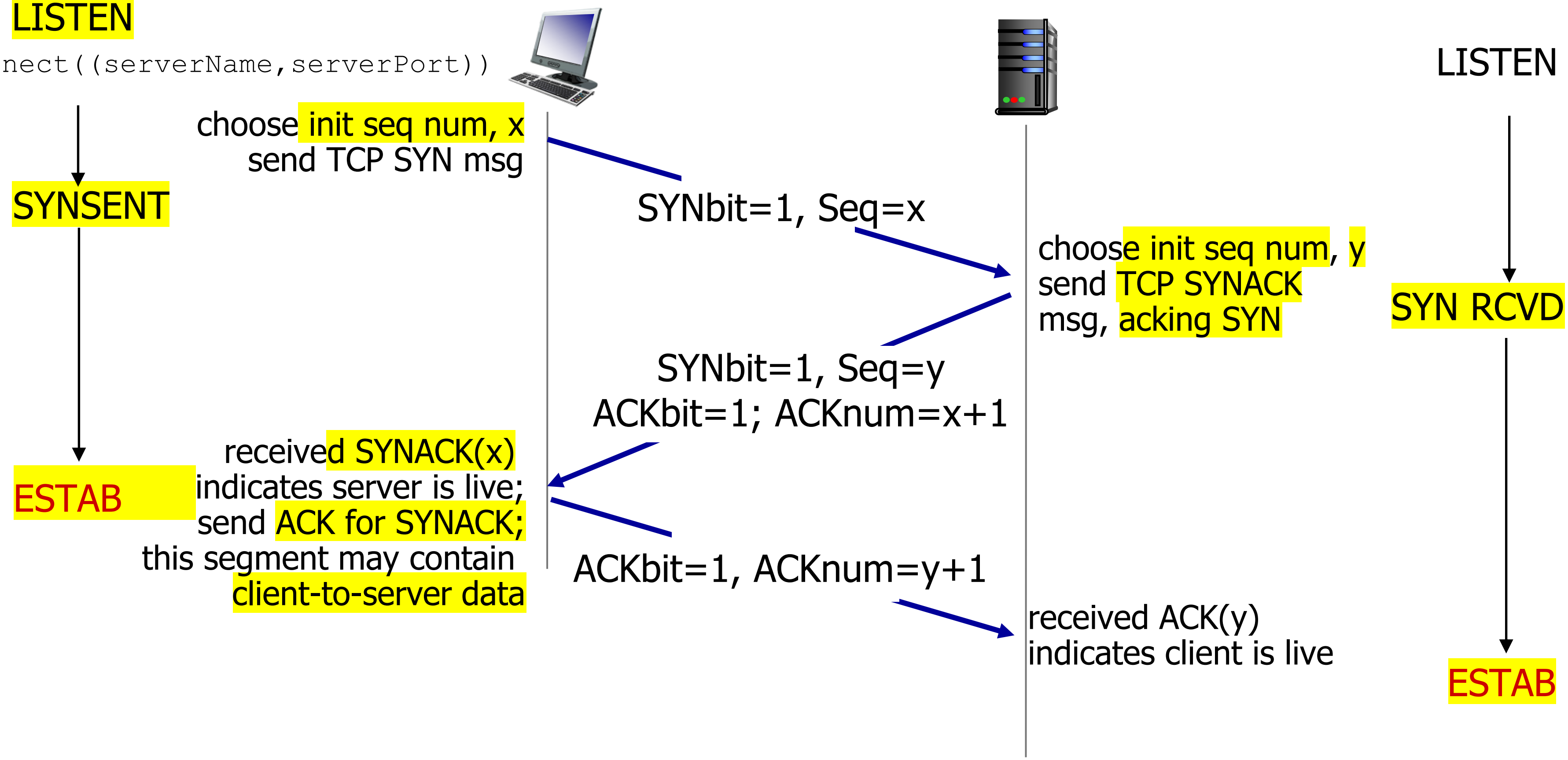
```
clientSocket = socket(AF_INET, SOCK_STREAM)
```

LISTEN

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

Server state

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



LISTEN

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

SYN RCVD

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

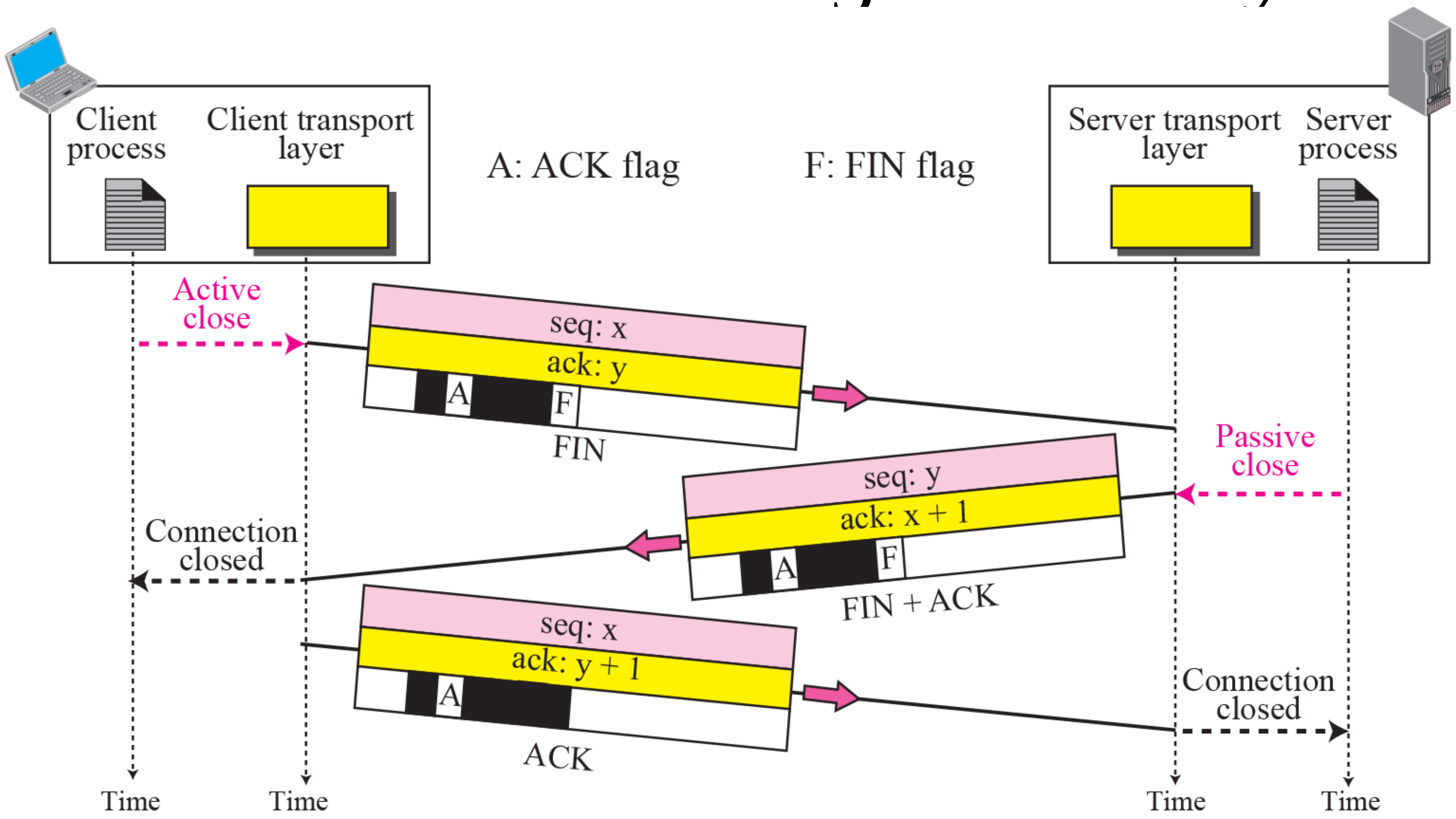
received ACK(y)
indicates client is live

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



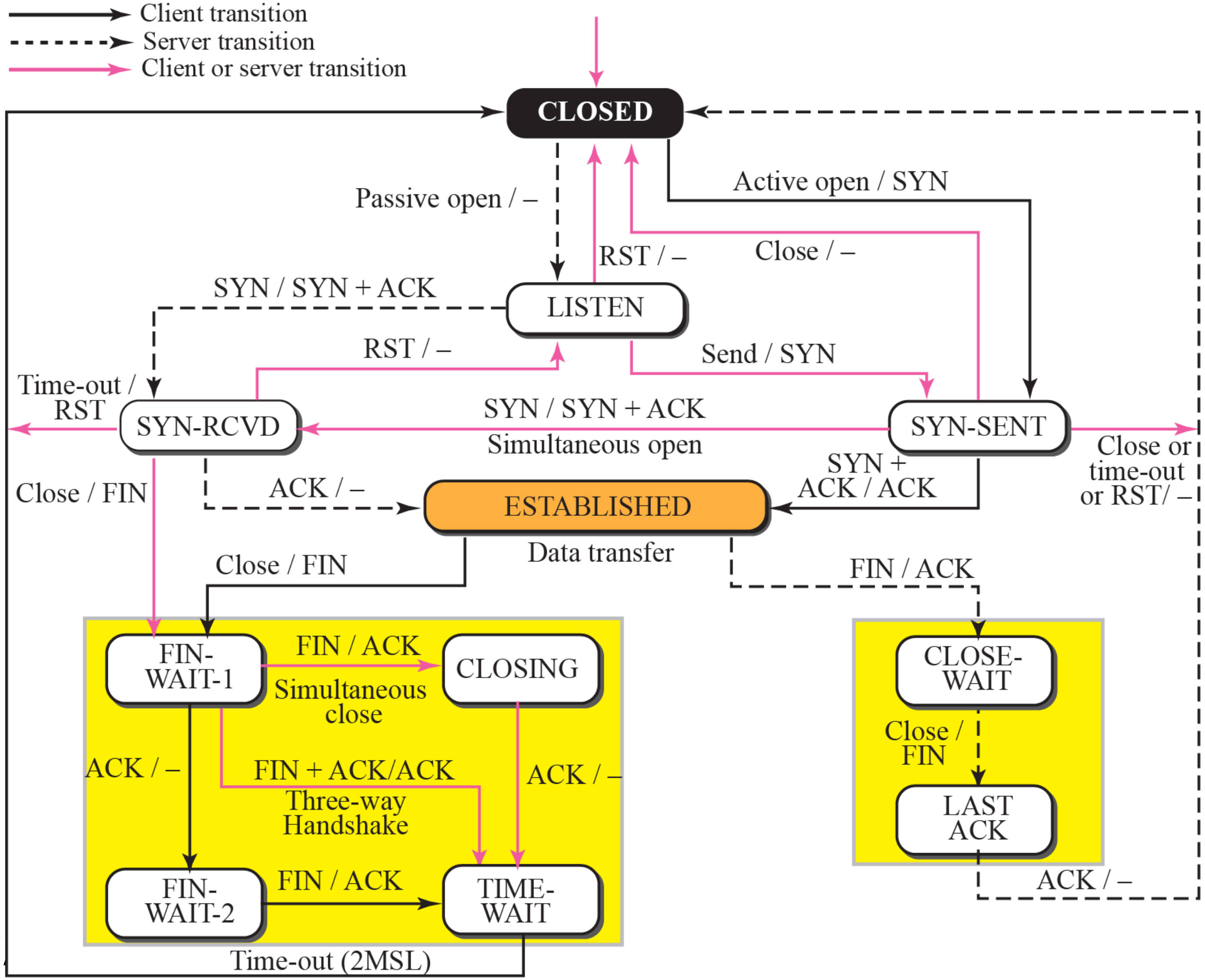
- The FIN segment consumes one sequence number if it does not carry data.
- The FIN + ACK segment consumes one sequence number if it does not carry data.

TCP/IP Protocol Suite

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



TCP

States of TCP

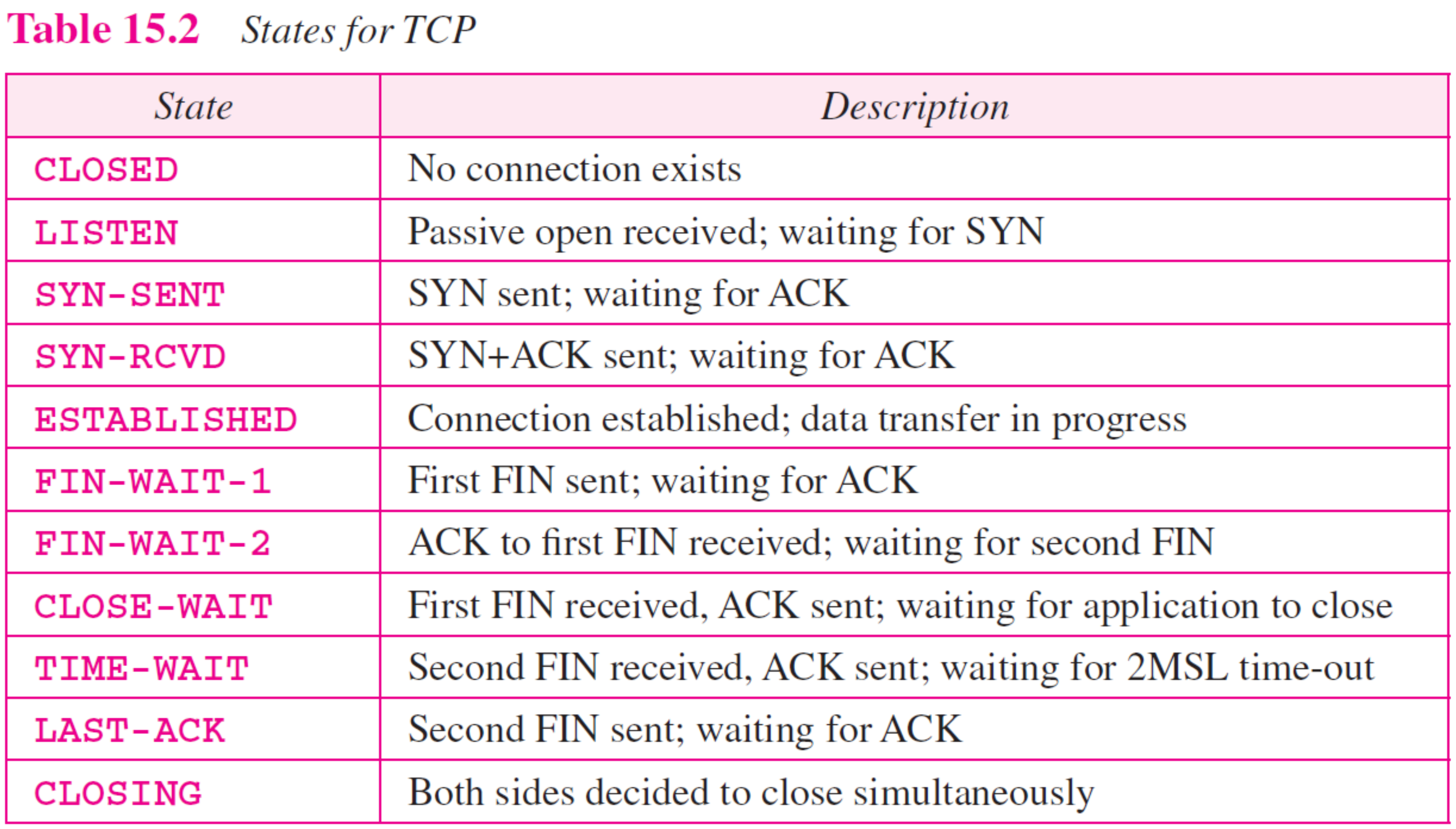
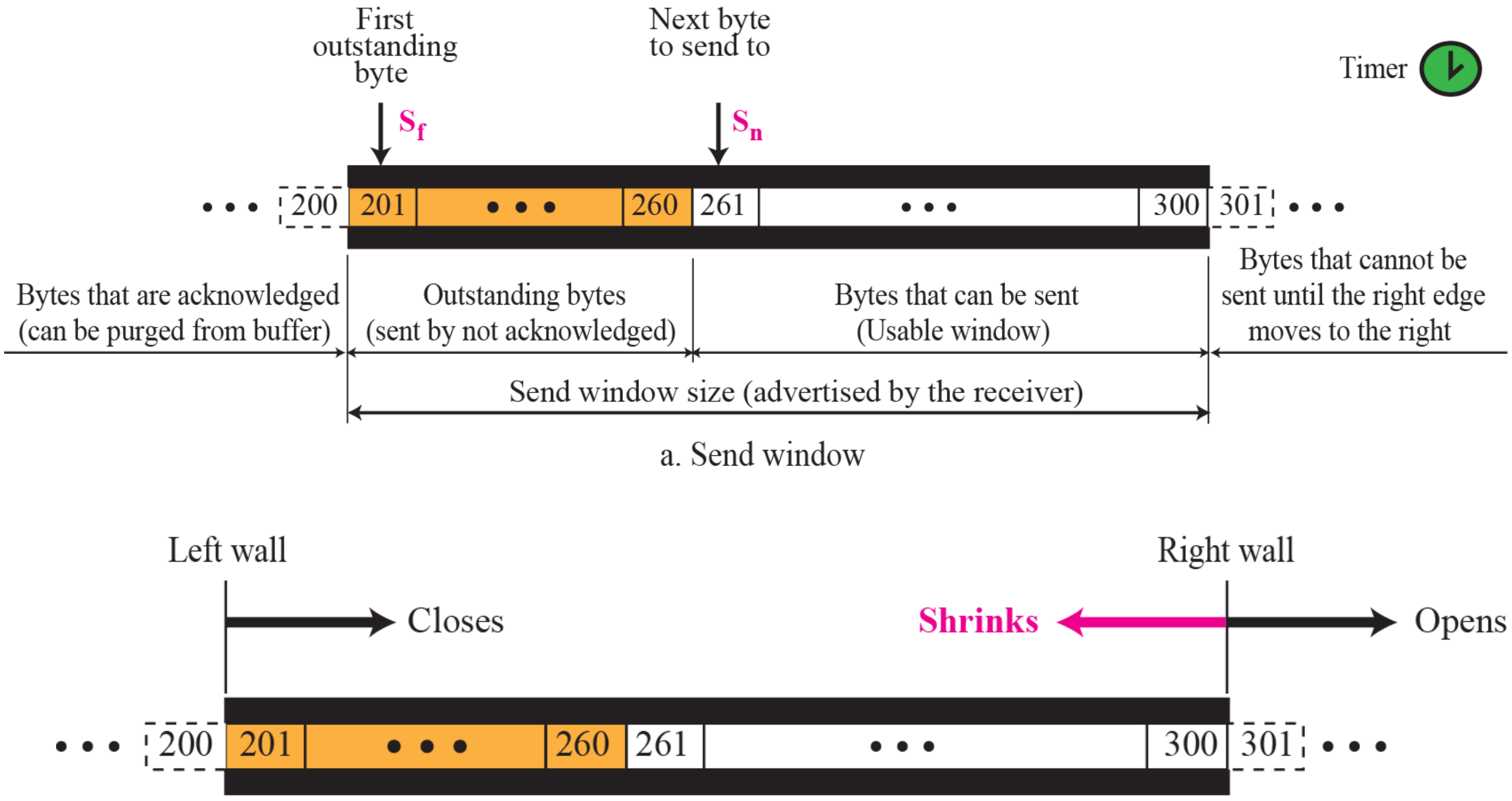


Table 15.2 *States for TCP*

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

TCP/IP Protocol Suite

Windows in TCP: *Send window in TCP*



First outstanding byte

S_f

Next byte to send to

S_n

Timer

• • • 200 201 • • • 260 261 • • • 300 301 • • •

Bytes that are acknowledged (can be purged from buffer)

Outstanding bytes (sent by not acknowledged)

Bytes that can be sent (Usable window)

Bytes that cannot be sent until the right edge moves to the right

Send window size (advertised by the receiver)

a. Send window

Left wall

Closes

Right wall

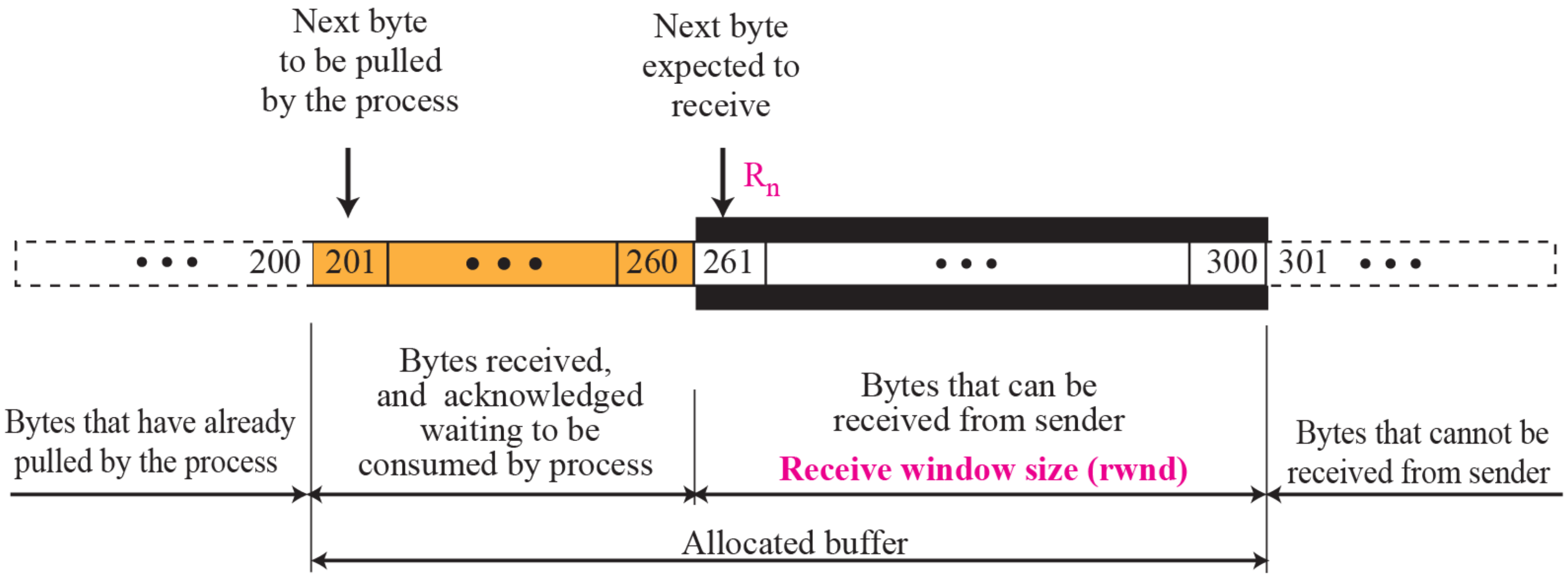
Shrinks

Opens

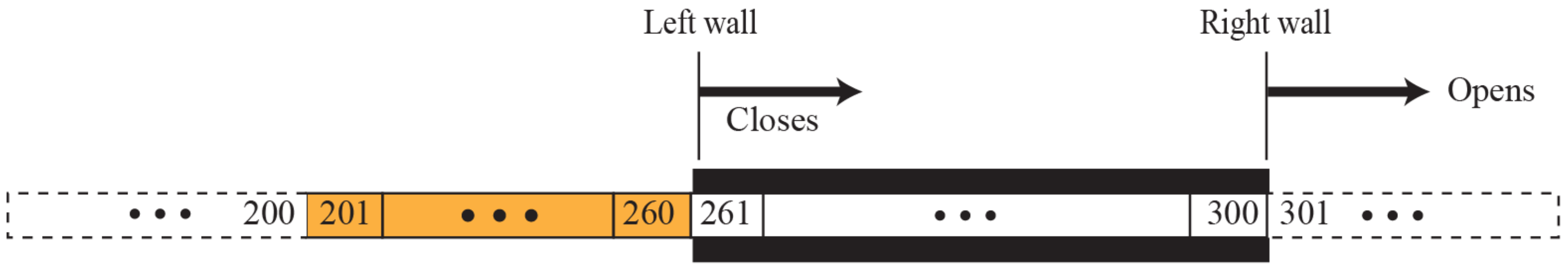
• • • 200 201 • • • 260 261 • • • 300 301 • • •

b. Opening, closing, and shrinking send window

Receive window in TCP


Next byte to be pulled by the process
Next byte expected to receive
R_n
• • • 200
201
• • •
260
261
• • •
300
301 • • •
Bytes that have already pulled by the process
Bytes received, and acknowledged waiting to be consumed by process
Bytes that can be received from sender
Receive window size (rwnd)
Bytes that cannot be received from sender
Allocated buffer

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.

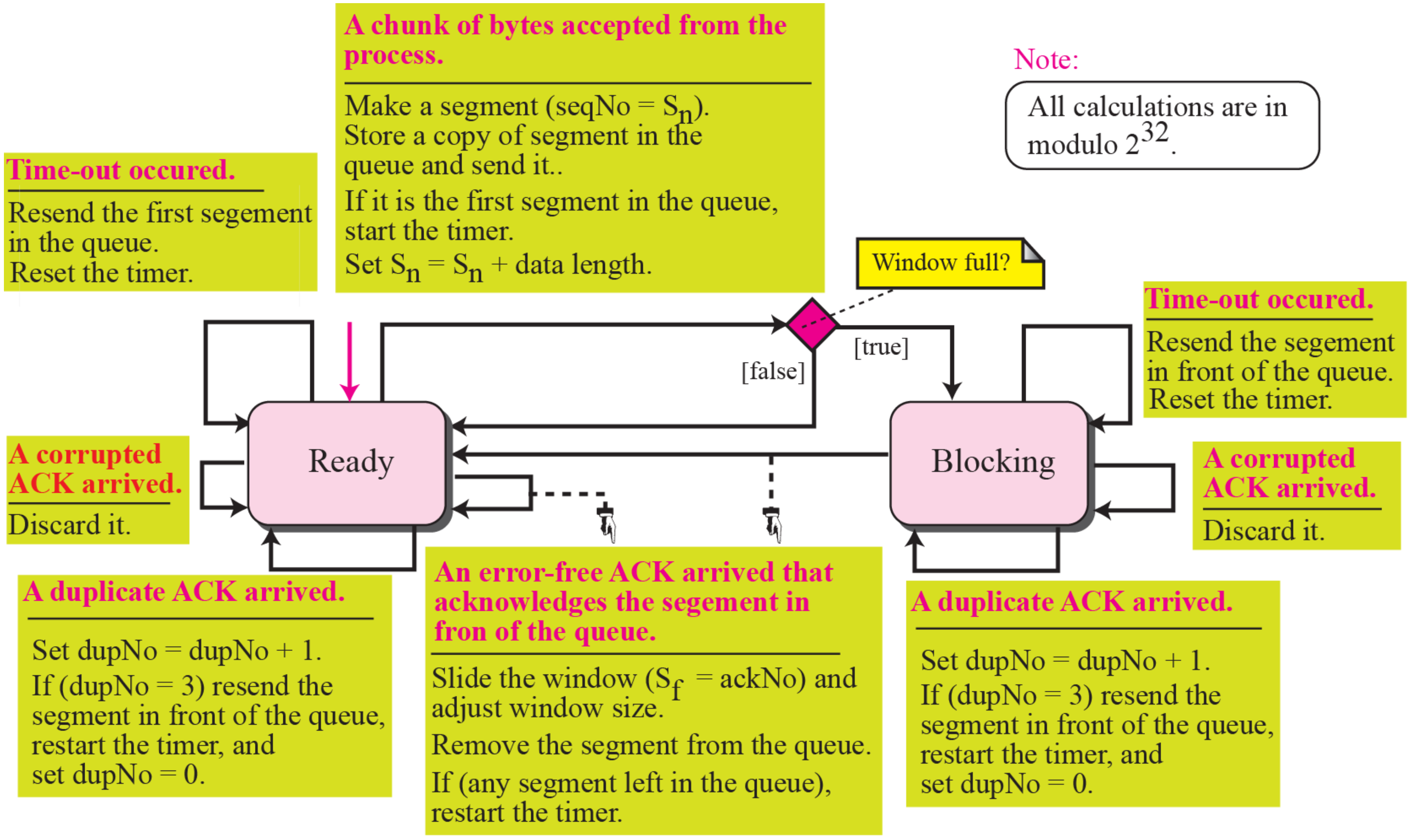
Silly Window Syndrome (2)

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

Simplified FSM for sender site



A chunk of bytes accepted from the process.

Make a segment (seqNo = S_n).
Store a copy of segment in the queue and send it..
If it is the first segment in the queue, start the timer.
Set $S_n = S_n$ + data length.

Note:

All calculations are in modulo 2^{32}.

Time-out occured.

Resend the first segement in the queue.
Reset the timer.

Time-out occured.

Resend the segement in front of the queue.
Reset the timer.

A corrupted ACK arrived.

Discard it.

A corrupted ACK arrived.

Discard it.

A duplicate ACK arrived.

Set dupNo = dupNo + 1.
If (dupNo = 3) resend the segment in front of the queue, restart the timer, and set dupNo = 0.

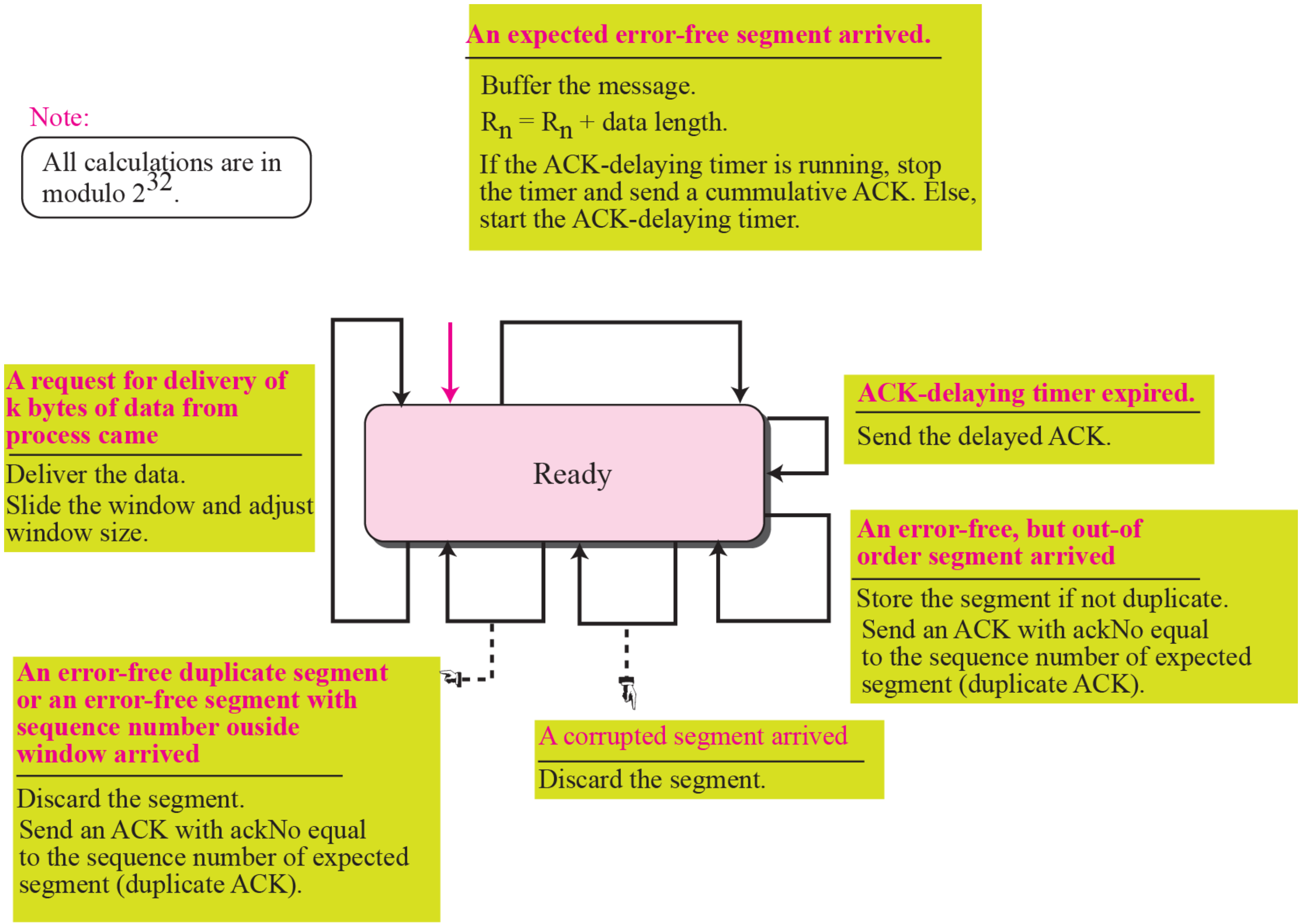
An error-free ACK arrived that acknowledges the segment in fron of the queue.

Slide the window (S_f = ackNo) and adjust window size.
Remove the segment from the queue.
If (any segment left in the queue), restart the timer.

A duplicate ACK arrived.

Set dupNo = dupNo + 1.
If (dupNo = 3) resend the segment in front of the queue, restart the timer, and set dupNo = 0.

Simplified FSM for the receiver site



Note:

All calculations are in modulo 2^{32}.

An expected error-free segment arrived.

Buffer the message.

$R_n = R_n$ + data length.

If the ACK-delaying timer is running, stop the timer and send a cummulative ACK. Else, start the ACK-delaying timer.

A request for delivery of k bytes of data from process came

Deliver the data.

Slide the window and adjust window size.

Ready

ACK-delaying timer expired.

Send the delayed ACK.

An error-free, but out-of order segment arrived

Store the segment if not duplicate.

Send an ACK with ackNo equal to the sequence number of expected segment (duplicate ACK).

An error-free duplicate segment or an error-free segment with sequence number ouside window arrived

Discard the segment.

Send an ACK with ackNo equal to the sequence number of expected segment (duplicate ACK).

A corrupted segment arrived

Discard the segment.