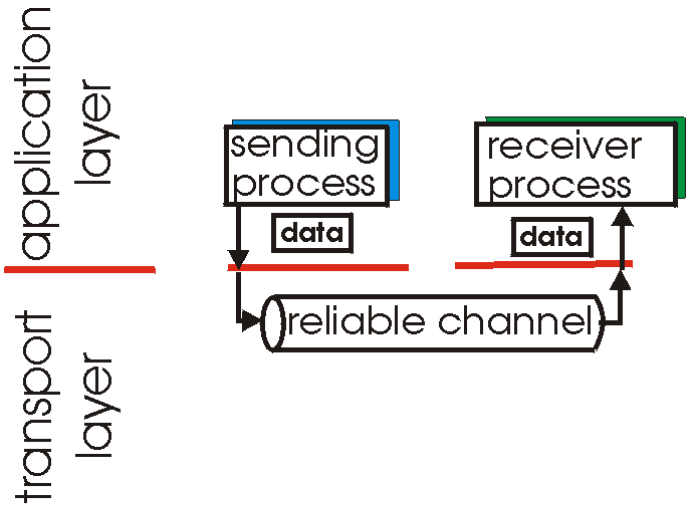
Reliable Transport Protocol

Principles of Reliable data transfer

- important in app., transport, link layers
- top-10 list of important networking topics!

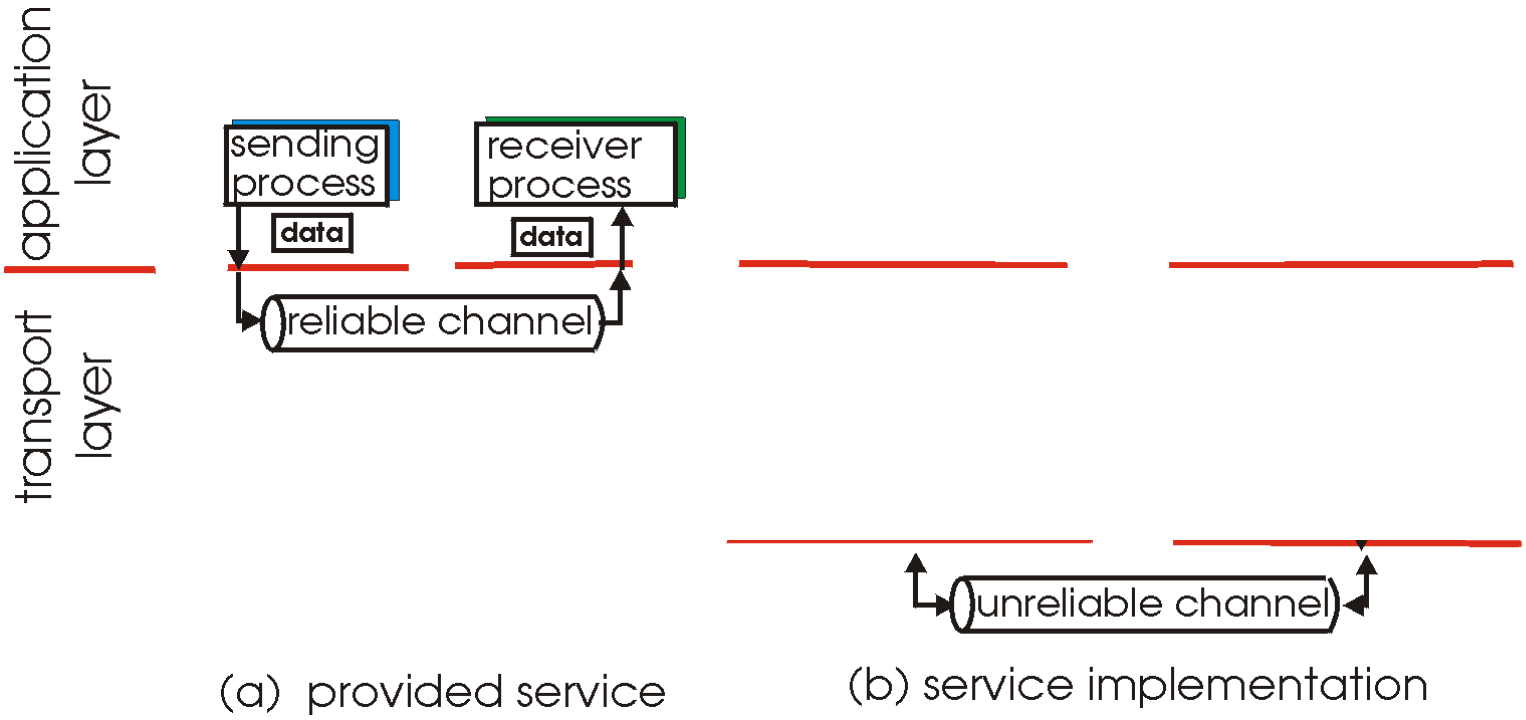


(a) provided service

- characteristics of unreliable channel will determine complexity of reliable data transfer protocol (rdt)

Principles of Reliable data transfer

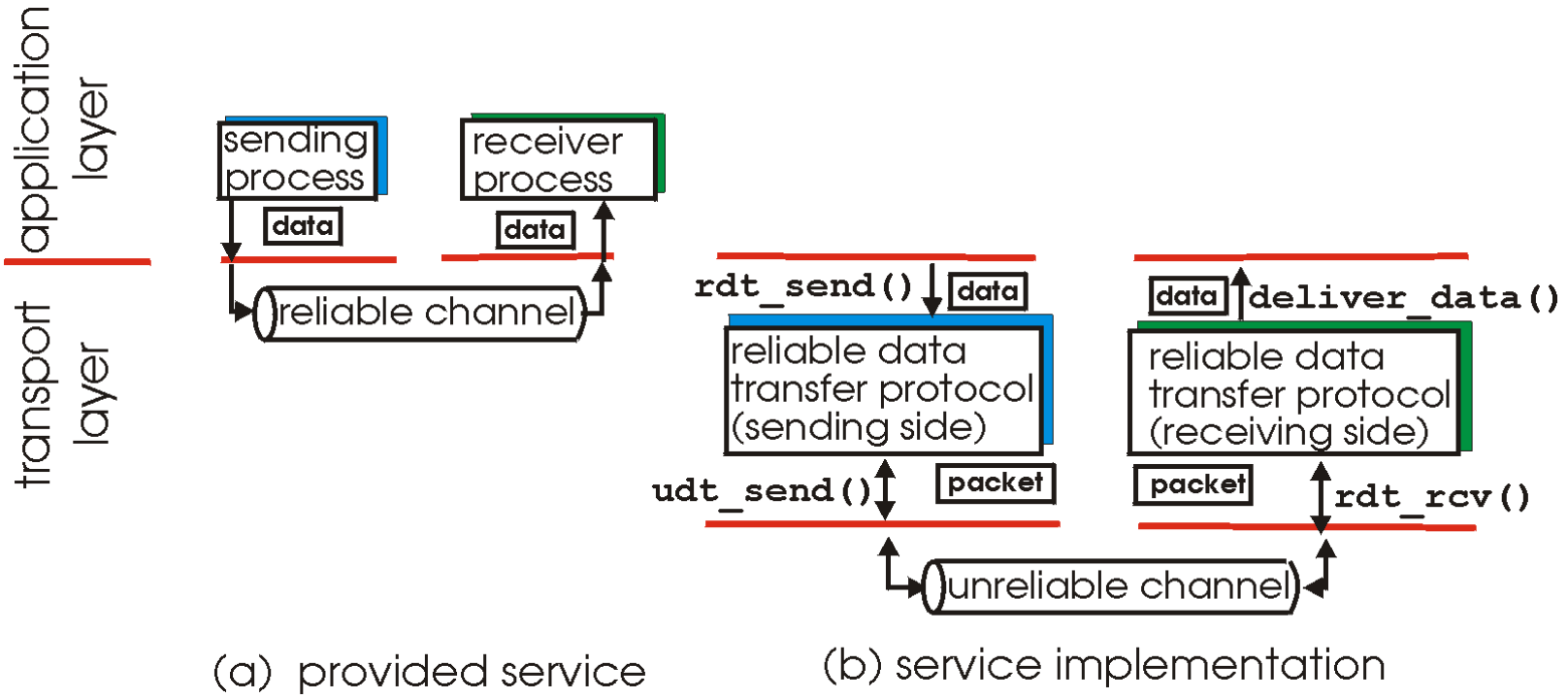
- important in app., transport, link layers
- top-10 list of important networking topics!



- characteristics of unreliable channel will determine complexity of reliable data transfer protocol (rdt)

Principles of Reliable data transfer

- important in app., transport, link layers
- top-10 list of important networking topics!

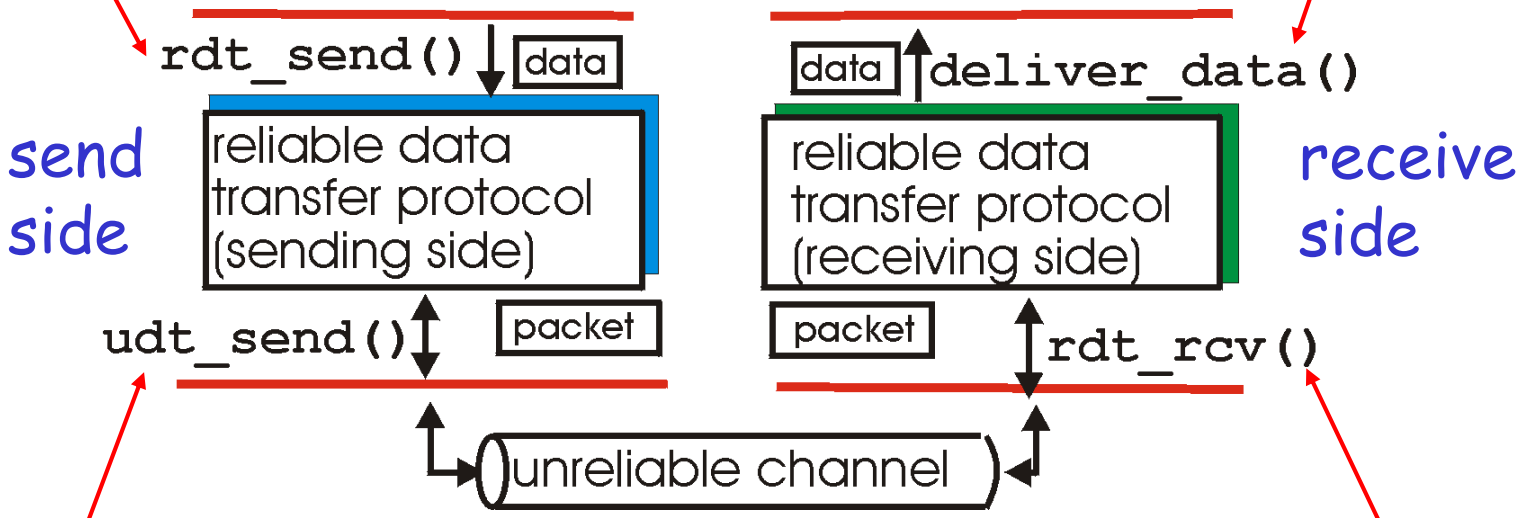


- characteristics of unreliable channel will determine complexity of reliable data transfer protocol (rdt)

Reliable data transfer: getting started

rdt_send(): called from above, (e.g., by app.). Passed data to deliver to receiver upper layer

deliver_data(): called by rdt to deliver data to upper



udt_send(): called by rdt, to transfer packet over unreliable channel to receiver

rdt_rcv(): called when packet arrives on rcv-side of channel

Transport layer
Transmission Control Protocol (TCP)

TRANSMISSION CONTROL PROTOCOL (TCP)

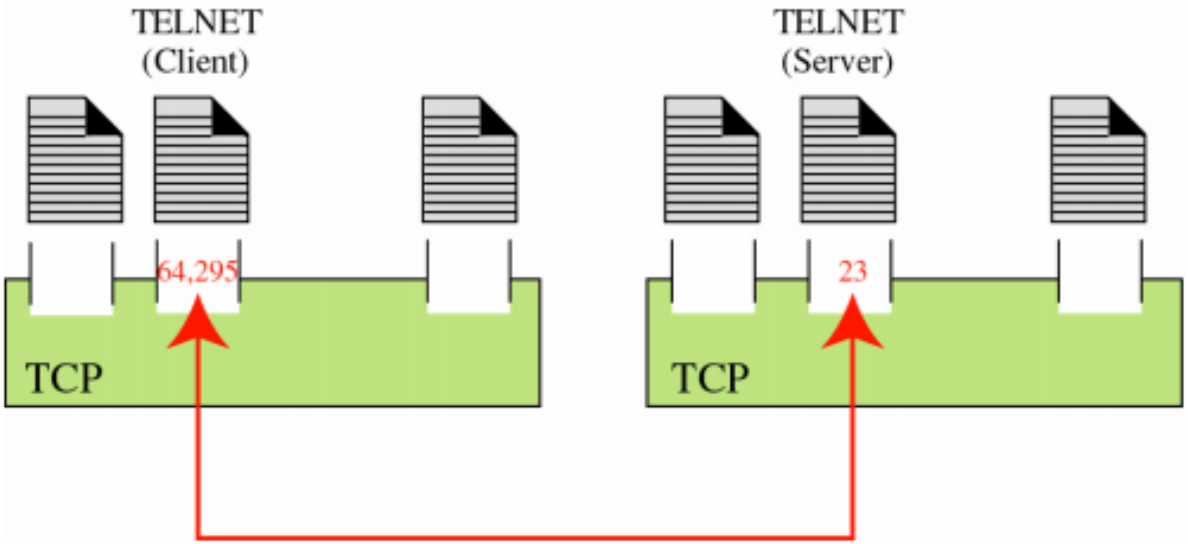
Sujoy Saha
Assistant Professor
Department of Computer Application
NIT durgapur

TCP SERVICES

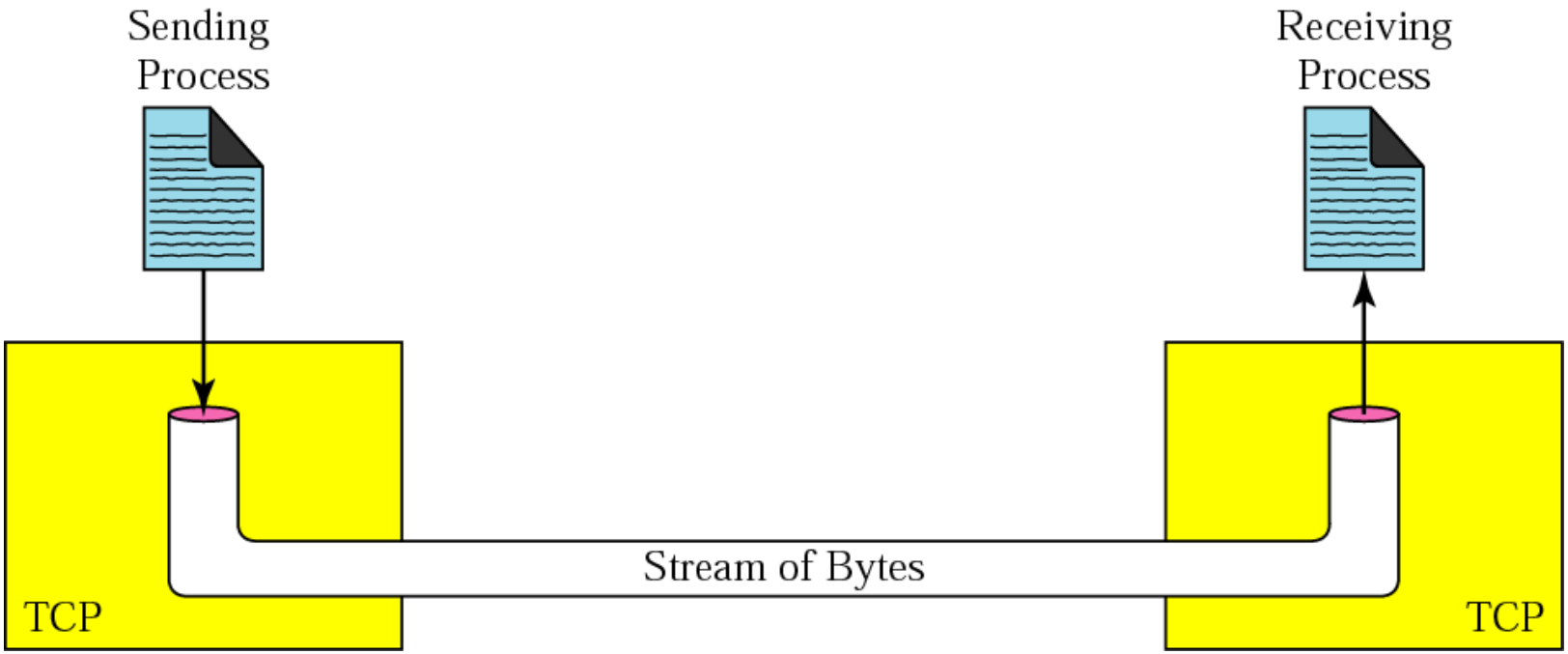
TCP Services

- TCP provides services to the processes at the application layer
  - Process-to-Process Communication
  - Stream Delivery Service
  - Full-Duplex Service
  - Connection-Oriented Service
  - Reliable Service

Port Numbers



Stream delivery

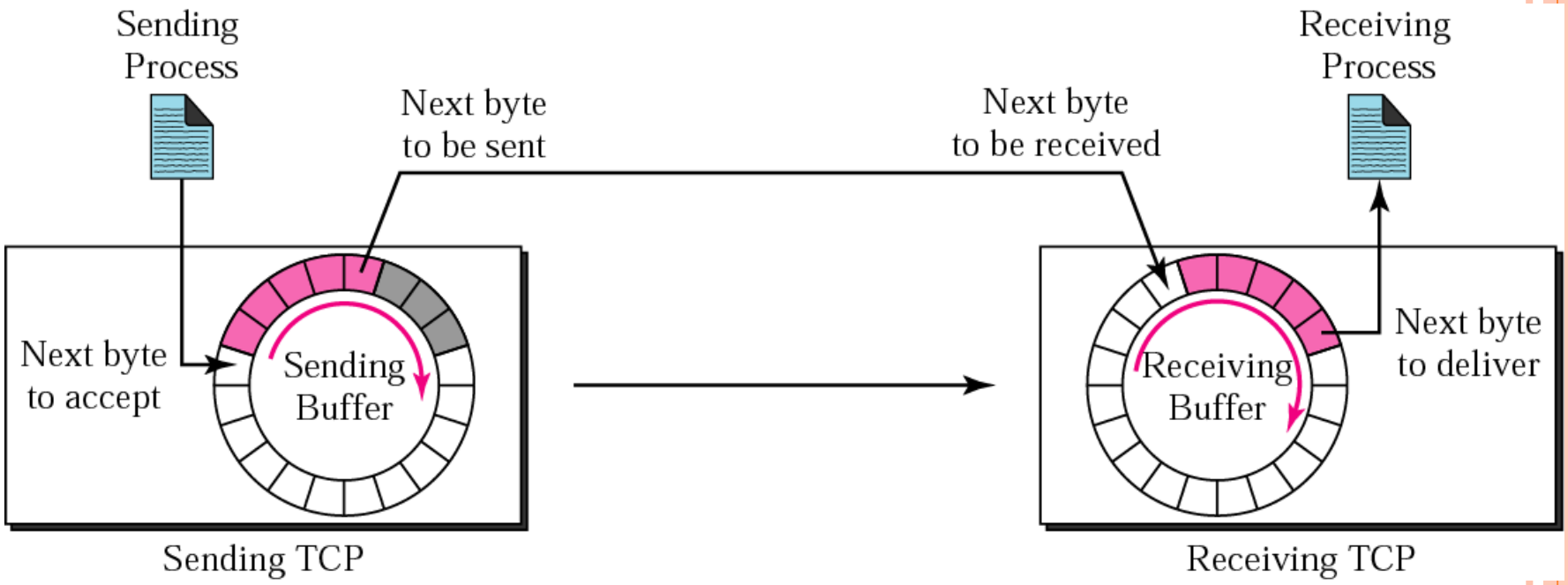


Stream Delivery Service (Cont.)

- However, the sending and receiving speed may not be the same
  - TCP needs buffers for storage

- Two buffers in TCP
  - Sending buffer and receiving buffer, one for each connection
  - Also used in flow- and error-control mechanisms

Sending and receiving buffers



Sender

Gray Chamber:- Bytes that have been send but not yet acknowledged.

Colored Area:-Bytes to be send.

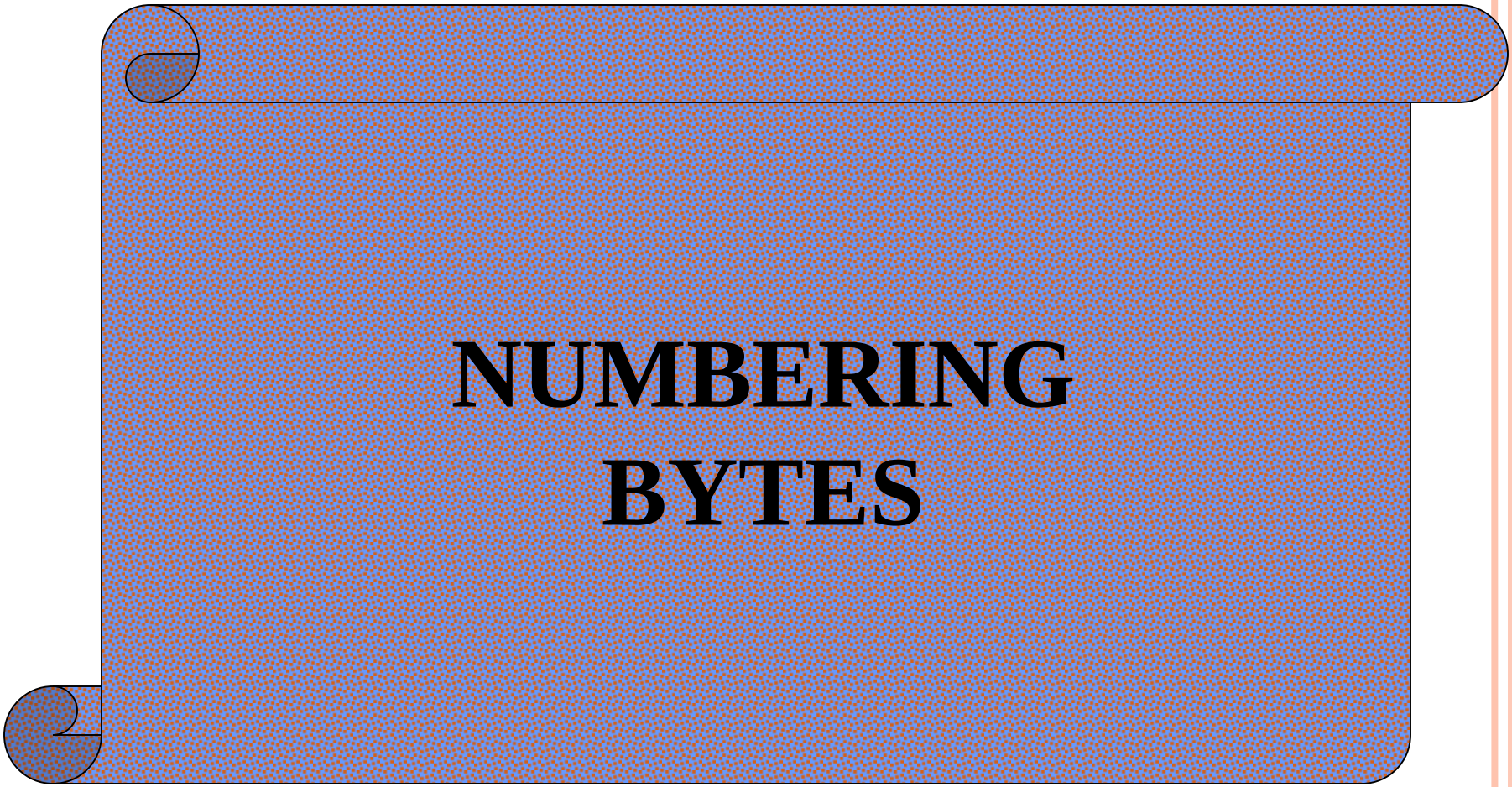
White Area:- Empty chamber, can be filled by sending process.

Receiver

Colored Area:-already received bytes by TCP & can be read by process

White Area:- Empty chamber, can be filled by bytes received from network.

NUMBERING BYTES



Numbering Bytes

- Although TCP use segments for transmission and reception
  - There is no field for a segment number in the segment header, i.e., TCP header

- TCP uses *sequence number* and *acknowledgement number* to keep track of the segment being transmitted or received
  - Notably, these two fields refer to the *byte number*, not the *segment number*

Note

The bytes of data being transferred in each connection are numbered by TCP. The numbering starts with a randomly generated number.

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)

Note

The value of the sequence number field in a segment defines the number of the first data byte contained in that segment.

Note

The value of the acknowledgment field in a segment defines the number of the next byte a party expects to receives. The acknowledgment number is cumulative.

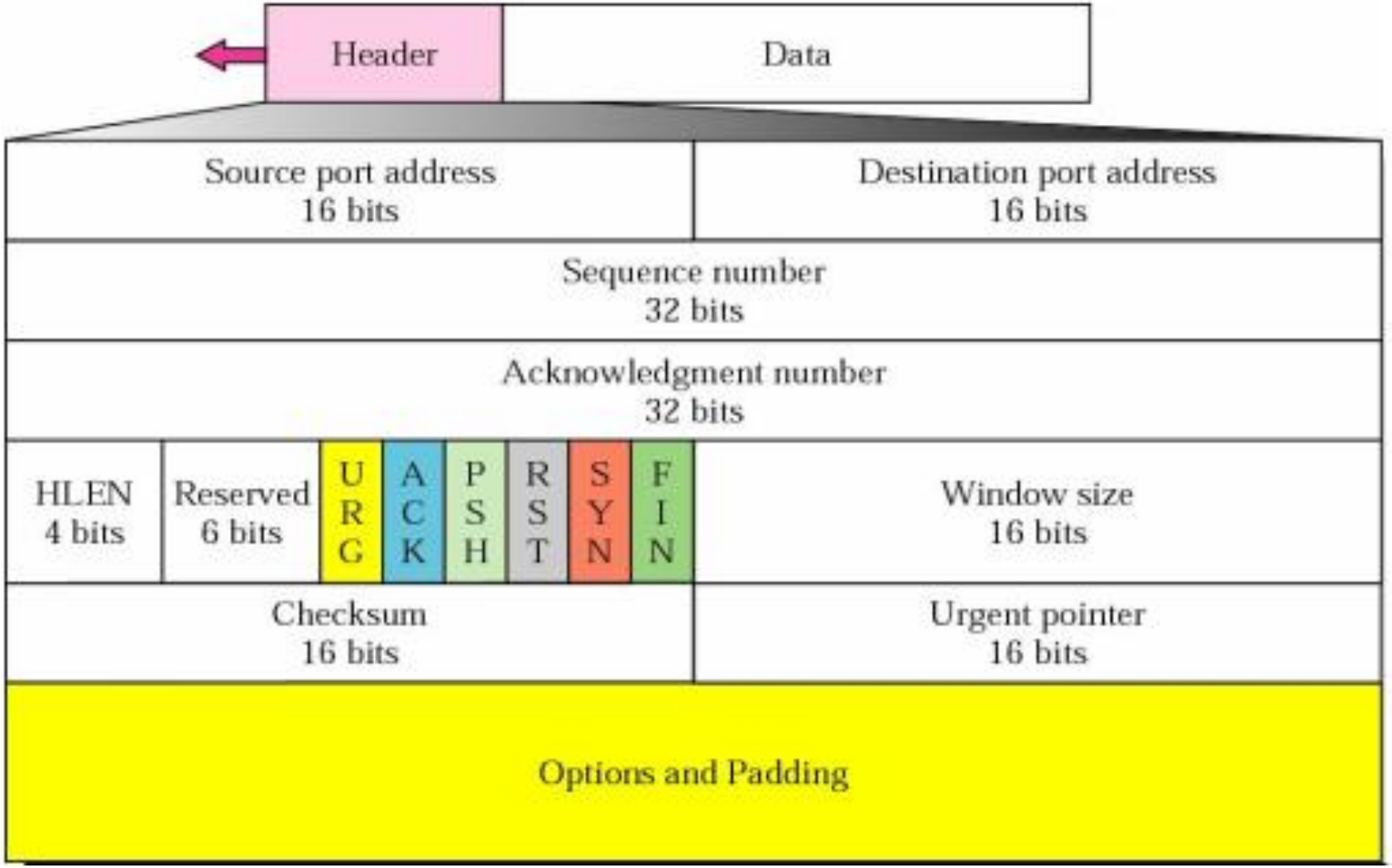
Acknowledgment Number

- Communication in TCP is full duplex
  - Both parties can send and receive data at the same time in a connection
- Each party numbers the bytes, usually with a different starting byte number
  - *Sequence number*: the number of the first byte carried by the segment
  - *Acknowledgment number*: the number of the next byte that *the party expects to receive*

Acknowledgment Number (Cont.)

- Acknowledgment number is *cumulative*
- For example, if a party uses 5,643 as an acknowledgment number
  - It has received all bytes from the beginning up to 5,642
  - Note that, *this does not mean that the party has received 5642 bytes*
    - The first byte number does not have to start from 0

TCP Segment Format



Control field

TCP Segment Format (Cont.)

- Control
  - URG: urgent pointer is valid
  - ACK: acknowledgment field is valid
  - PSH: push the data
  - RST: the connection must be reset
  - SYN: Synchronization sequence numbers during connection
  - FIN: terminate the connection

It is used to terminate the connection if sender feels something is wrong with the TCP connection or that the conversation should not exist.

PUSH Flag

MSS = 600 Byte

- Transport layer by default waits for some time for application layer to send enough data equal to maximum segment size so that the number of packets transmitted on network minimizes.
- which is not desirable by some application like interactive applications(chatting).
- This problem is solved by using PSH. Transport layer sets PSH = 1 and immediately sends the segment to network layer as soon as it receives signal from application layer.

TCP Segment Format (Cont.)

- Window size: 16-bit
  - Define the size of the receiving window, in bytes
    - Determined by the receiver
  - The maximum window size is 2^16 = 65535
- Checksum: 16-bit
  - Follow the same procedure as UDP
  - Checksum for TCP is mandatory (UDP is optional)
- Urgent pointer: 16-bit
  - Valid only if the urgent bit is set
  - Used when the segment contains urgent data
- Options: 0~40 bytes

12.4 A TCP CONNECTION

TCP is connection-oriented. A connection-oriented transport protocol establishes a virtual path between the source and destination. All of the segments belonging to a message are then sent over this virtual path. A connection-oriented transmission requires three phases: connection establishment, data transfer, and connection termination.

The topics discussed in this section include:

Connection Establishment
Data Transfer
Connection Termination
Connection Reset

Introduction

- TCP's connection-oriented transmission requires three phases
  - ***Connection establishment***
    - Three-way handshaking
  - ***Data transfer***
  - ***Connection termination***
    - Four-way handshaking

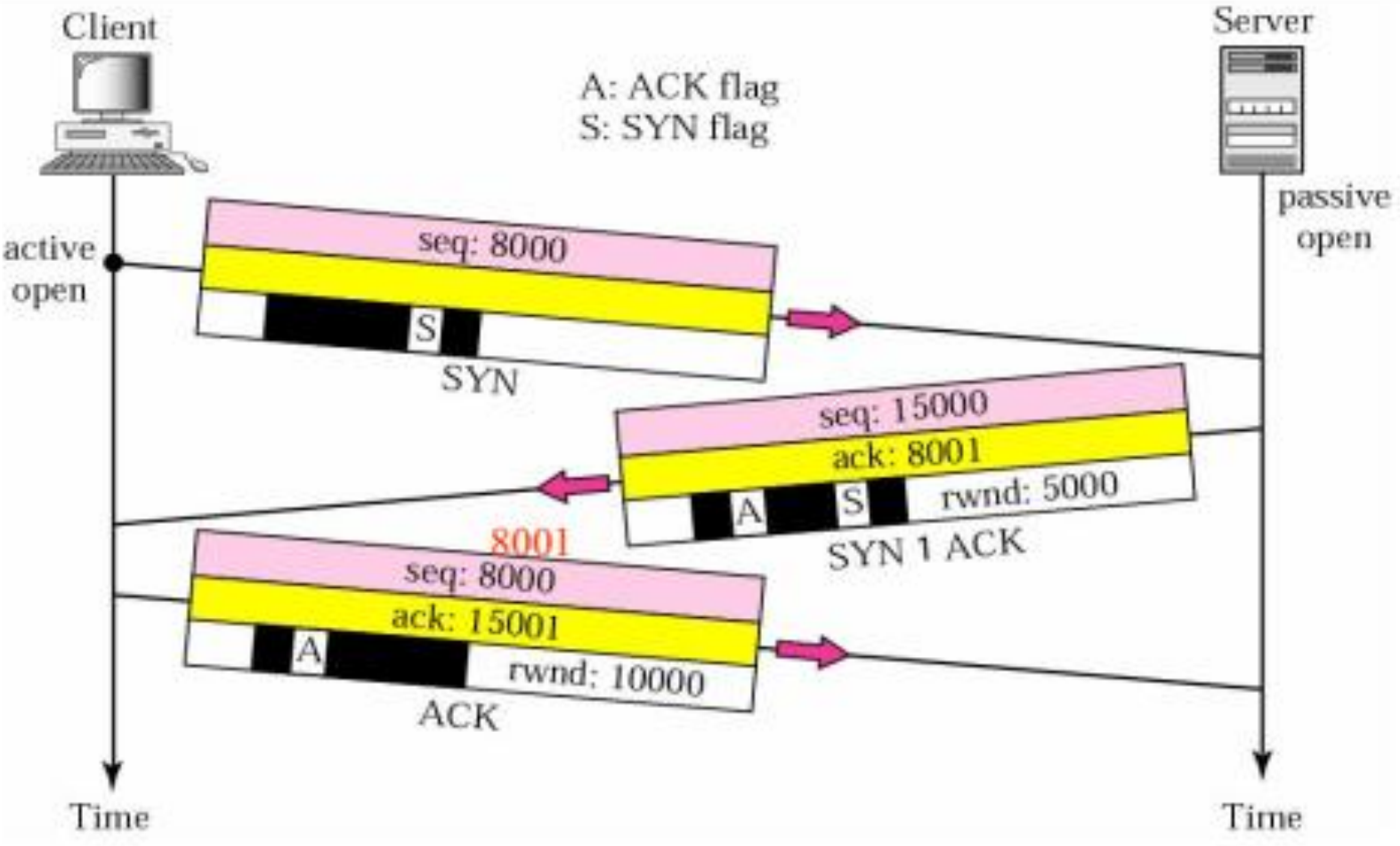
Connection Establishment

- Four actions are taken between host A and B
  - Host A sends a segment to announce its wish for connection and includes its initialization information
  - Host B sends a segment to acknowledge the request of A
  - Host B sends a segment that includes its initialization information
  - Host A sends a segment to acknowledge the request of B
- However
  - Step 2 and 3 can be combined into one step

Connection Establishment

- Example, a client wants to make a connection to a server
  - Server performs the *passive open*
    - Tell TCP that it is ready to accept a connection
  - Client performs the *active open*
    - Tell TCP that it needs to be connected to the server

Three-way Handshaking



3 Way handshaking

The client sends the first segment, a *SYN* segment

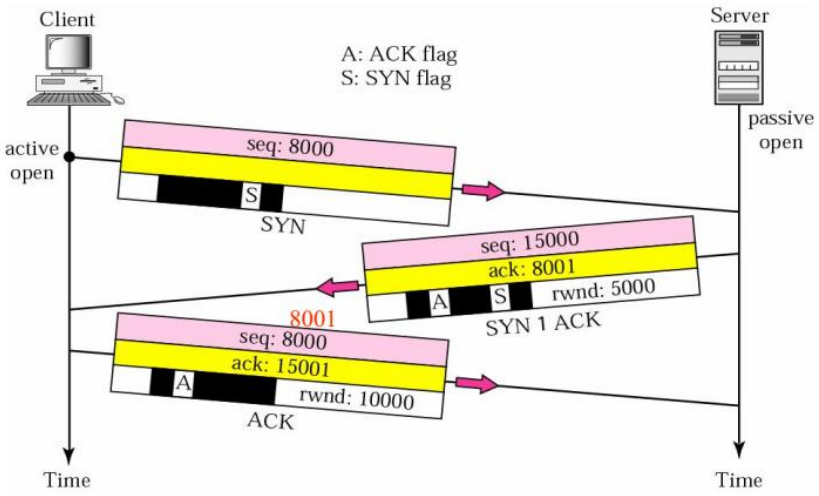
- Set the *SYN* flag
- The segment is used for synchronization of sequence number
  - o *Initialization sequence number (ISN)*
- If client wants to define MSS, add MSS option
- Does not contain any acknowledgment number
- Does not define the window size either
- Although a control segment and does not carry data
  - But consumes *one sequence number*

Receiver Window Size (rwnd)

- The **TCP window size**, or as some call it, the **TCP** receiver **window size**, is simply an advertisement of how much data (in bytes) the receiving device is willing to receive at any point in time.
- The receiving device can use this value to control the flow of data, or as a flow control mechanism.
- The **maximum segment size** (**MSS**) is a parameter of the *options* field of the TCP header that specifies the largest amount of data, specified in bytes, that a computer or communications device can receive in a single TCP segment. It does not count the TCP header or the IP header

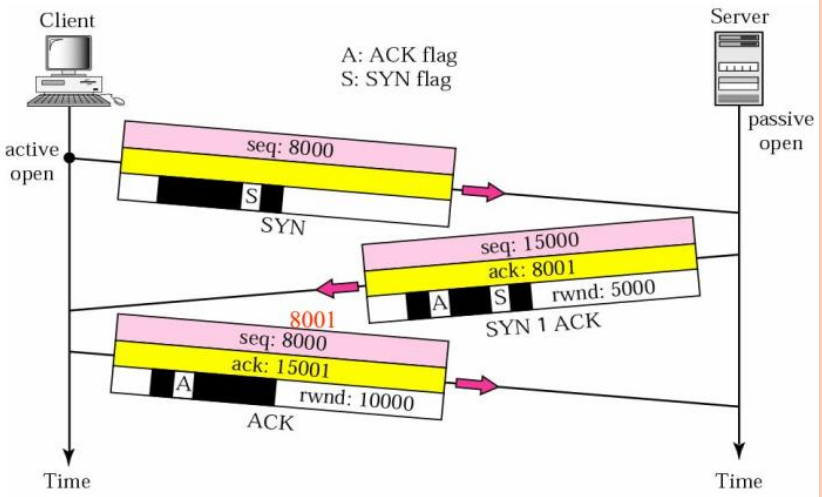
Three-way handshaking (Cont.)

2. The server sends a second segment, a ***SYN + ACK*** segment
   - Set the *SYN* and *ACK* flag
   - *Acknowledge* the receipt of the first segment using the ACK flag and acknowledgment number field
     - Acknowledgment number = client initialization sequence number + 1
     - Must also define the receiver window size for flow control
   - *SYN* information for the server
     - Initialization sequence number from server to client
     - Window scale factor if used
     - MSS is defined



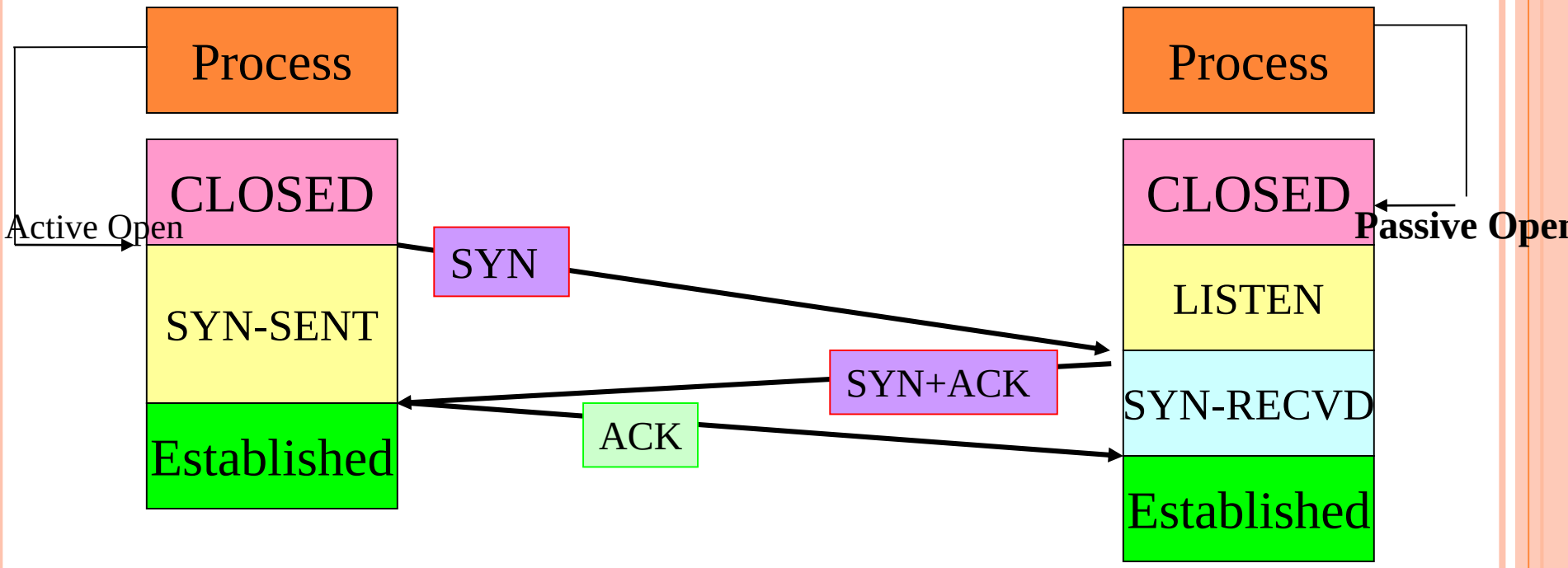
Three-way handshaking (Cont.)

3. The client sends the third segment, ***ACK*** segment
   - *Acknowledge* the receipt of second segment
     - ACK flag is set
       - Acknowledgement number = server initialization sequence number + 1
     - Must also define the server window size
       - Set the window size field
     - The sequence number is the same as the one in the SYN segment
       - ACK segment does not consume any sequence number
   - However, in some implementation, data can be sent with the third packet
     - Must have a new sequence number showing the byte number of the first byte in the data



A SYN segment cannot carry data, but it consumes one sequence number.

THREE-WAY HANDSHAKING – Sate Transition Diagram



SYN FLOODING ATTACK

- A malicious attacker sends a larger number of SYN segment to a server
  - Each with a fading source IP address
- The server will runs out of resource and may crash
  - ***Denial of service attack***

SYN FLOODING ATTACK

- This SYN Flooding attack belongs to a group of security attacks known as a **Denial of Service Attack.**

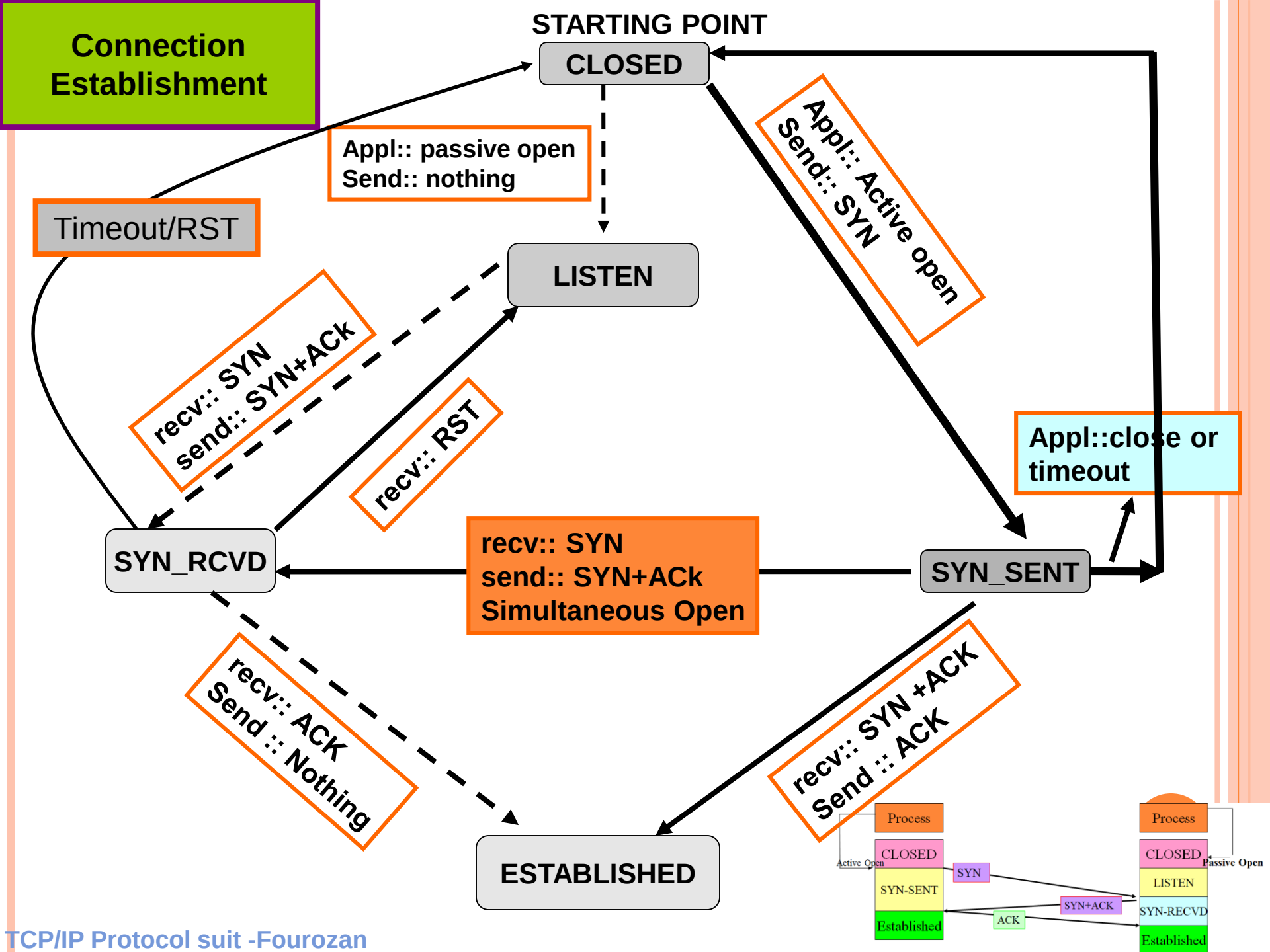
- An attacker monopolizes a system with so many service requests that the system collapses and denies service to every request.

SYN Flooding Attack (Cont.)

- Possible solutions
  - Impose a limit of connections requested during a period of time
  - Filter out datagrams coming from unwanted source addresses
  - Postpone resource allocation until the entire connection is set up
    - SCTP uses strategy, called ***cookie***

STATE TRANSITION DIAGRAM

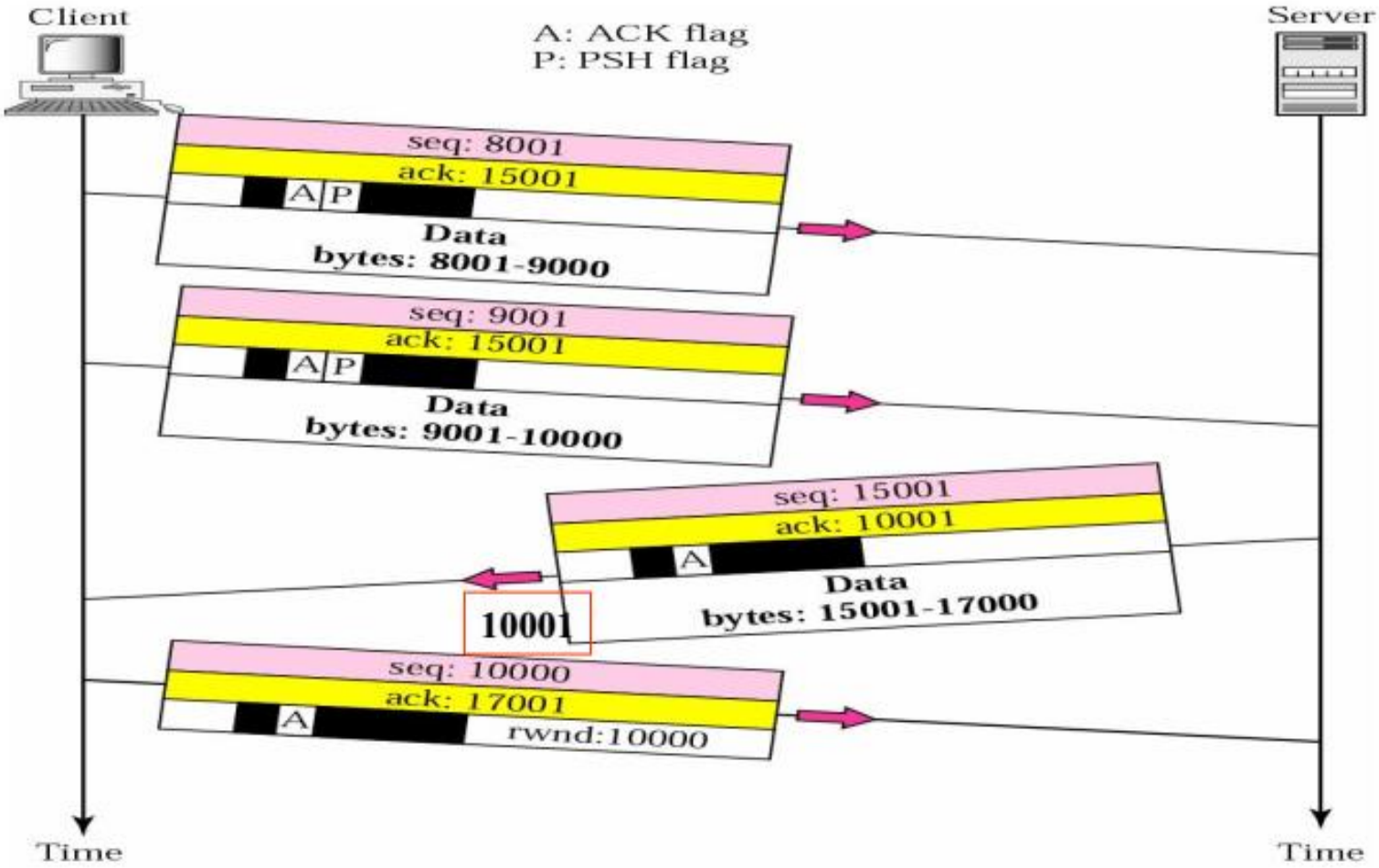
Connection Establishment

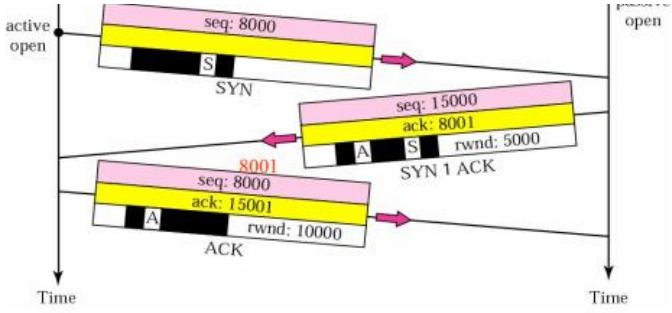


Data Transfer

- ***Bidirectional*** data transfer takes place after connection is established
  - Both parties can send data and acknowledgments in both direction
  - The acknowledgment can be piggybacked with the data

Example: a Data Transfer





Pushing Data

- In TCP, both sender and receiver have buffers to hold data
  - In sender, application data to be sent is temporary hold in the buffer
  - In receiver, receiving data is temporary hold in the buffer
  - Thus, for applications, they may encounter delayed transmission and reception

Pushing Data (Cont.)

- In some cases, *delayed transmission and reception* may not be acceptable
- TCP thus support ***PUSH*** operation
  - Sending TCP must create a segment and send the data immediately
    - Must not wait for the window to be filled
  - Receiving TCP must deliver data to the application immediately
    - Does not wait for more data to come

DATA TRANSFER

- Push Data

- PSH flag is used to facilitate real-time communication via TCP.
- Large buffers do more harm than good when dealing with real-time applications which require that data be transmitted as quickly as possible.

Example – Telnet Protocol

- The sending application informs TCP that data should be sent immediately.
- The PSH flag in the TCP header informs the receiving host that the data should be pushed up to the receiving application immediately.

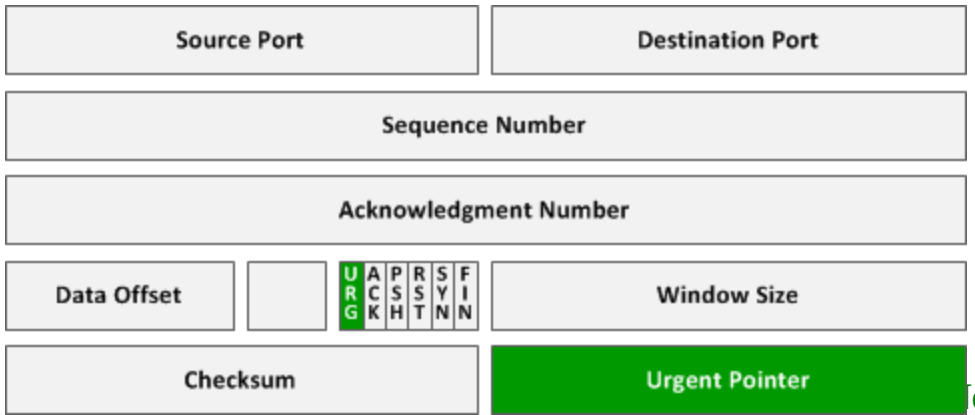
- **http://packetlife.net/blog/2011/mar/2/tcp-flags-psh-and-urg/**

- Urgent Data

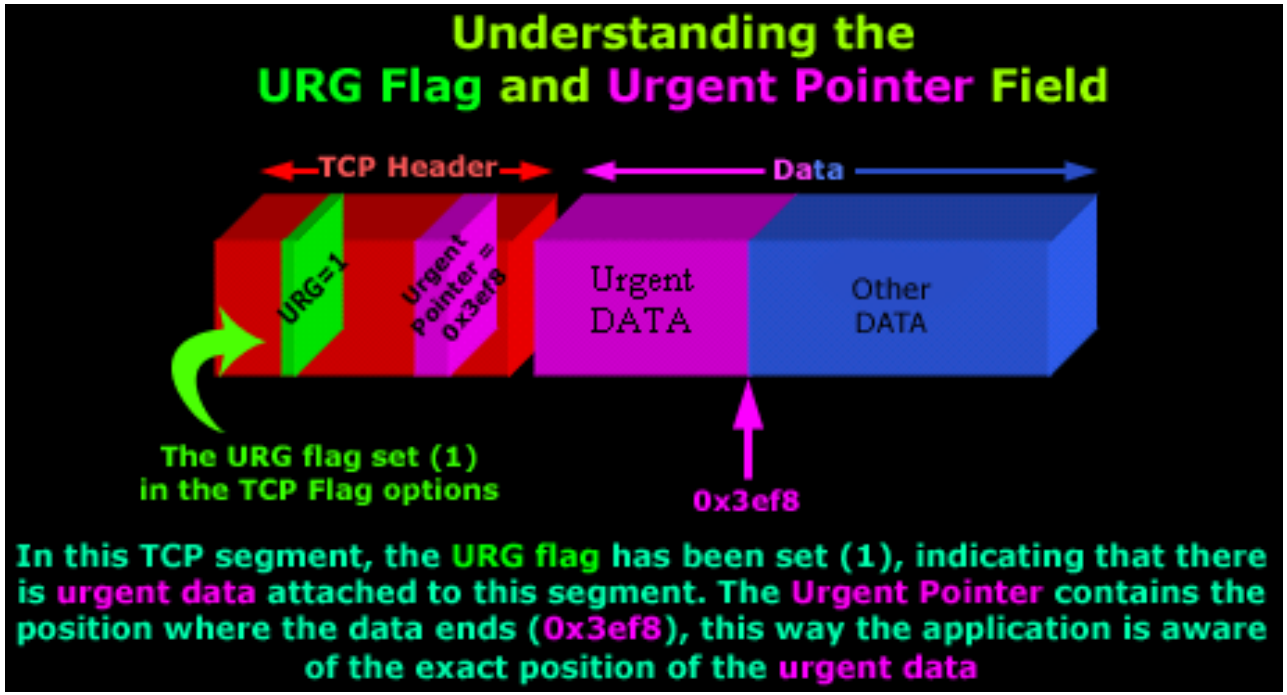
Urgent data is data that should be acted upon as soon as possible, even before "normal" data that may be waiting should be processed.

The URG flag is used to inform a receiving station that certain data within a segment is urgent and should be prioritized.

If the URG flag is set, the receiving station evaluates the urgent pointer, a 16-bit field in the TCP header. This pointer indicates how much of the data in the segment, counting from the first byte, is urgent.



URGENT

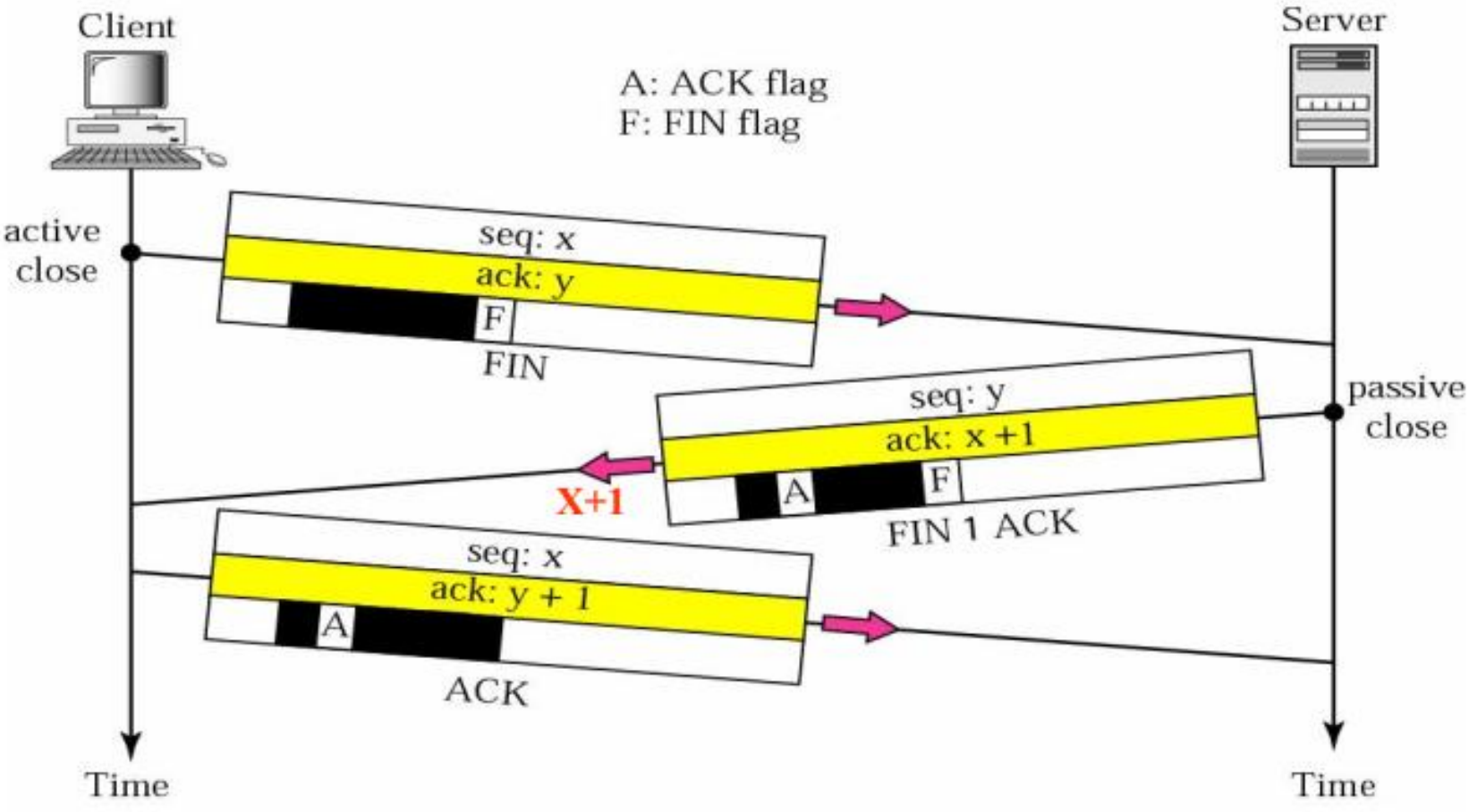


The urgent pointer flag in the TCP Flag allows us to mark a segment of data as 'urgent', while this urgent pointer field specifies where exactly the urgent data ends.

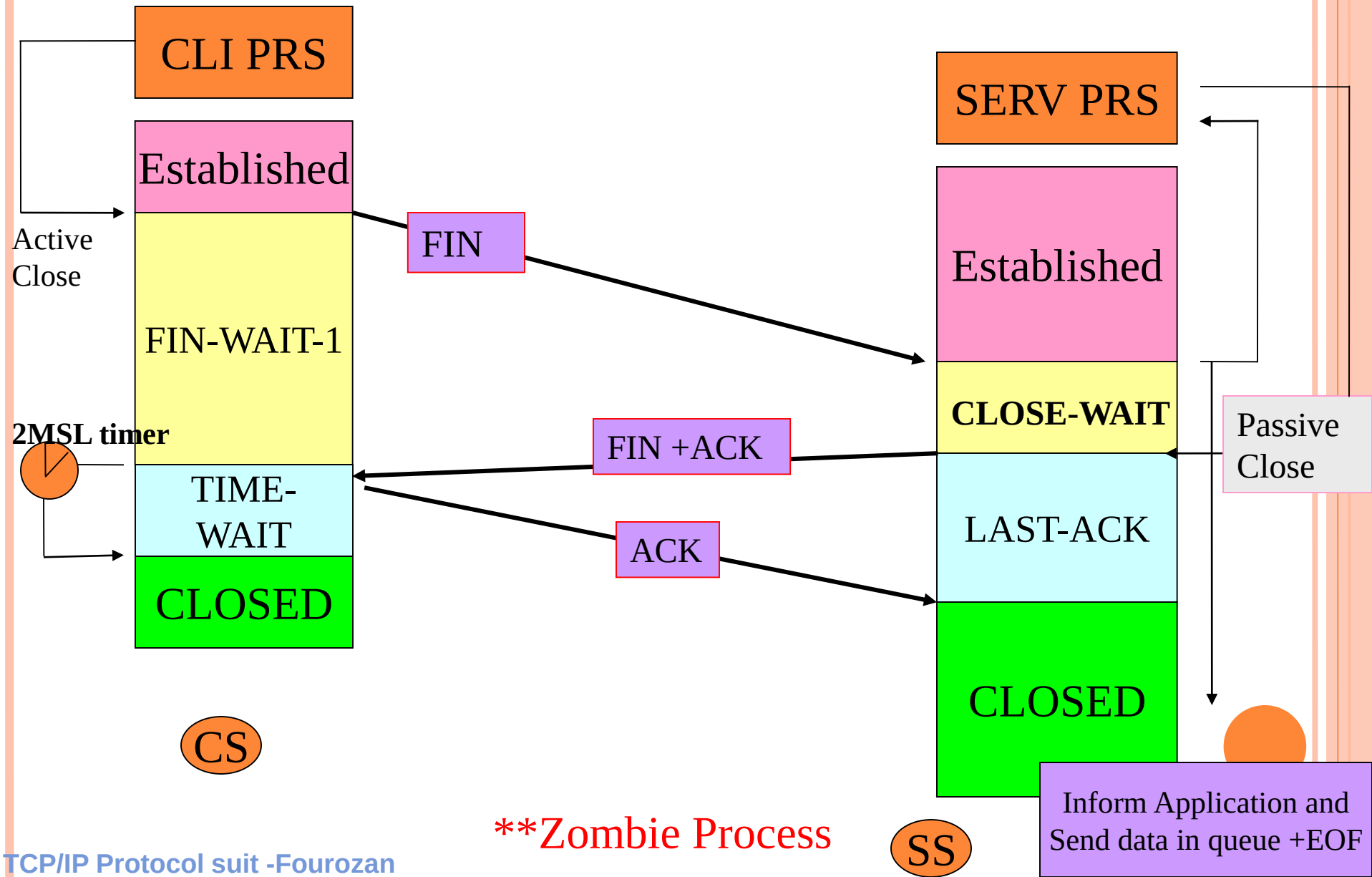
CONNECTION TERMINATION

- Three-way Handshaking
- Simultaneous Close
- Four Way Handshaking

Three-Way Handshaking



THREE-WAY HANDSHAKING (CT)



**Zombie Process

2MSL TIMER

MSL is the maximum time a segment can exist in the internet before it is dropped.

The common value for MSL is between 30 seconds and 1 minutes.

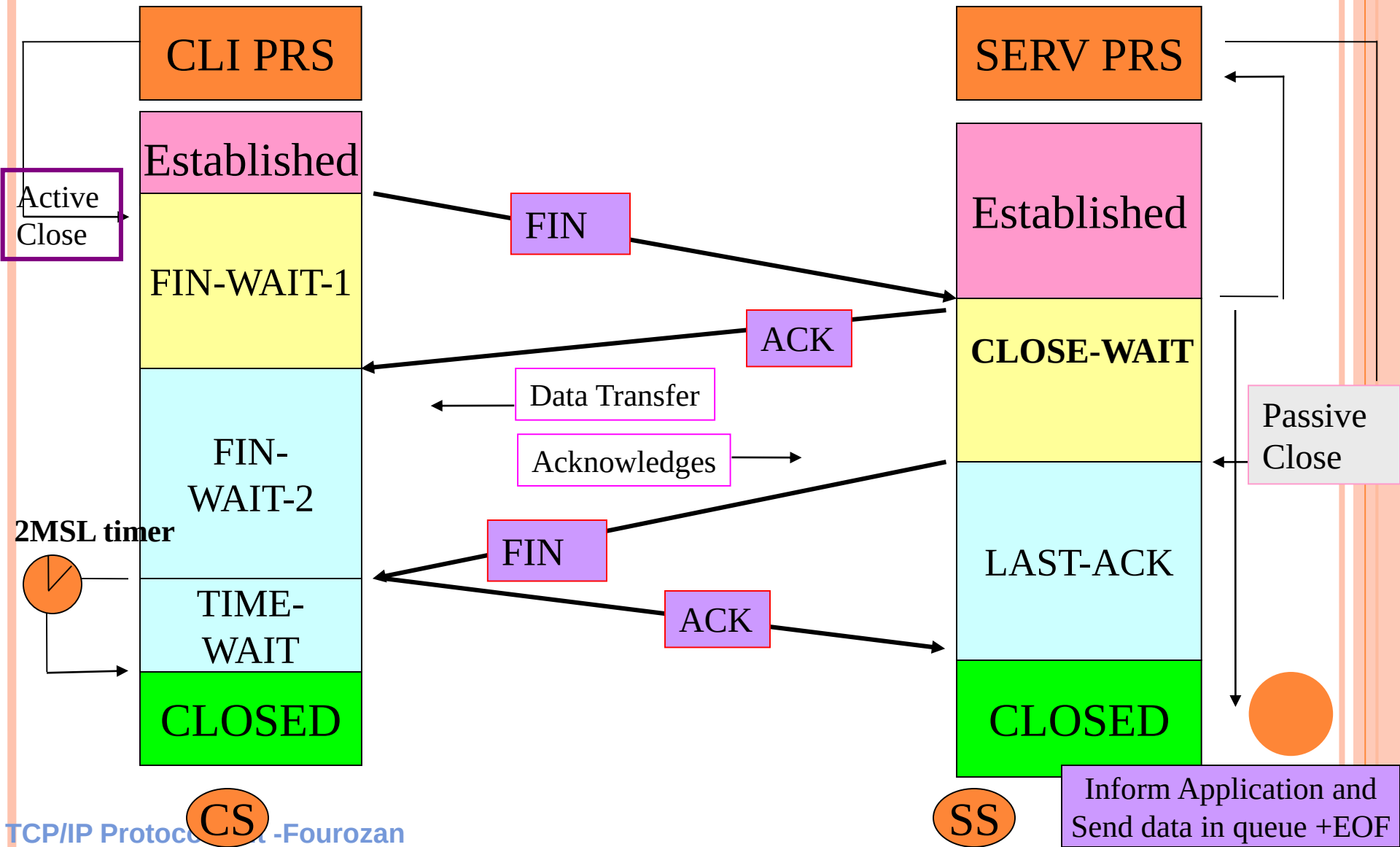
Main Reasons for the existence of the TIME-WAIT and the 2MSL Timer:-

If The Last ACK segment is lost → server assumes that last FIN (FIN+ACK) segment is lost and resends it.

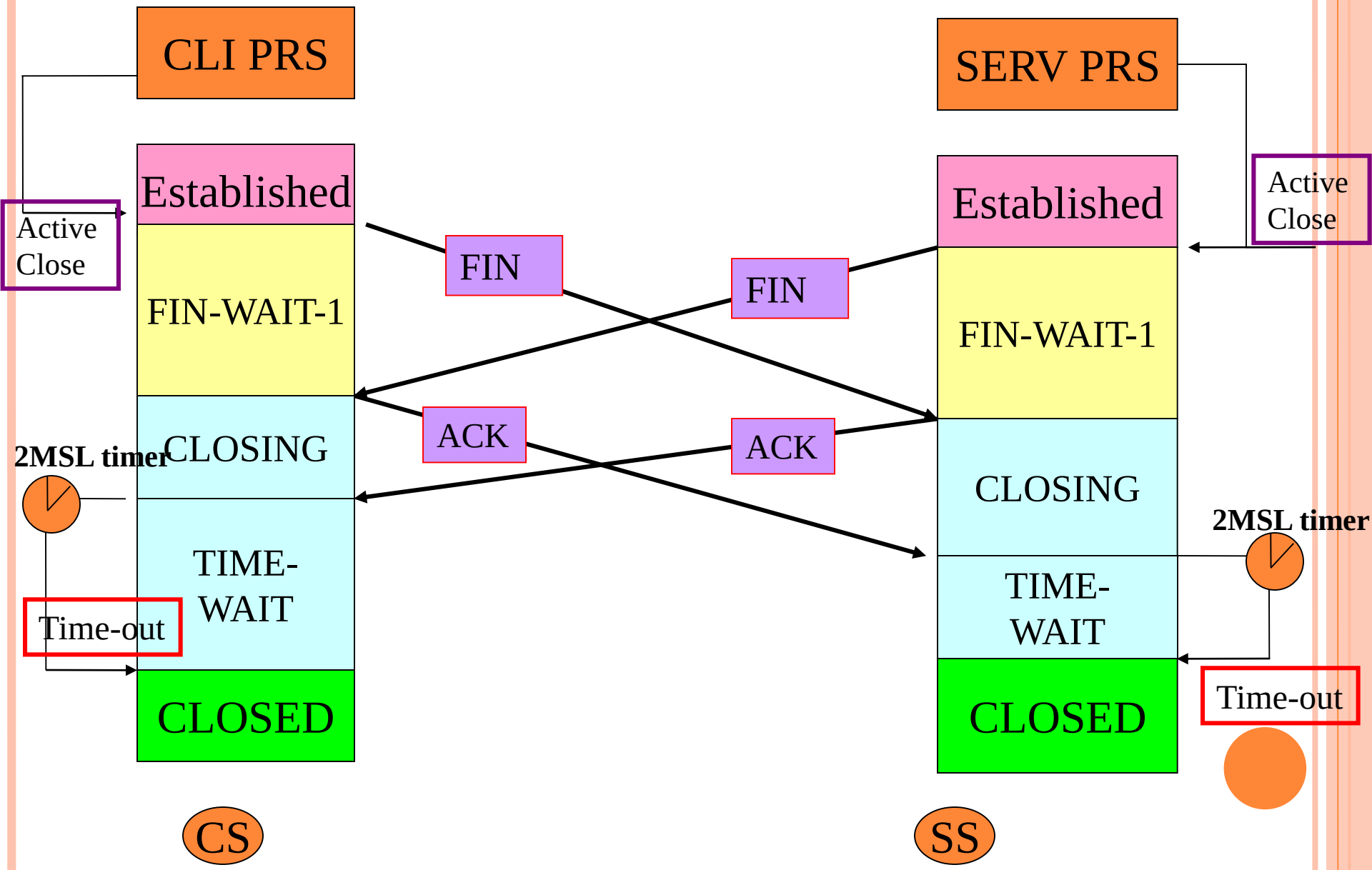
if the client goes to the CLOSED state and close the connection Before the 2MSL timer then server never close the connection

During the TIME-WAIT state if a new FIN arrives the client sent New ACK & restarts the 2MSL timer.

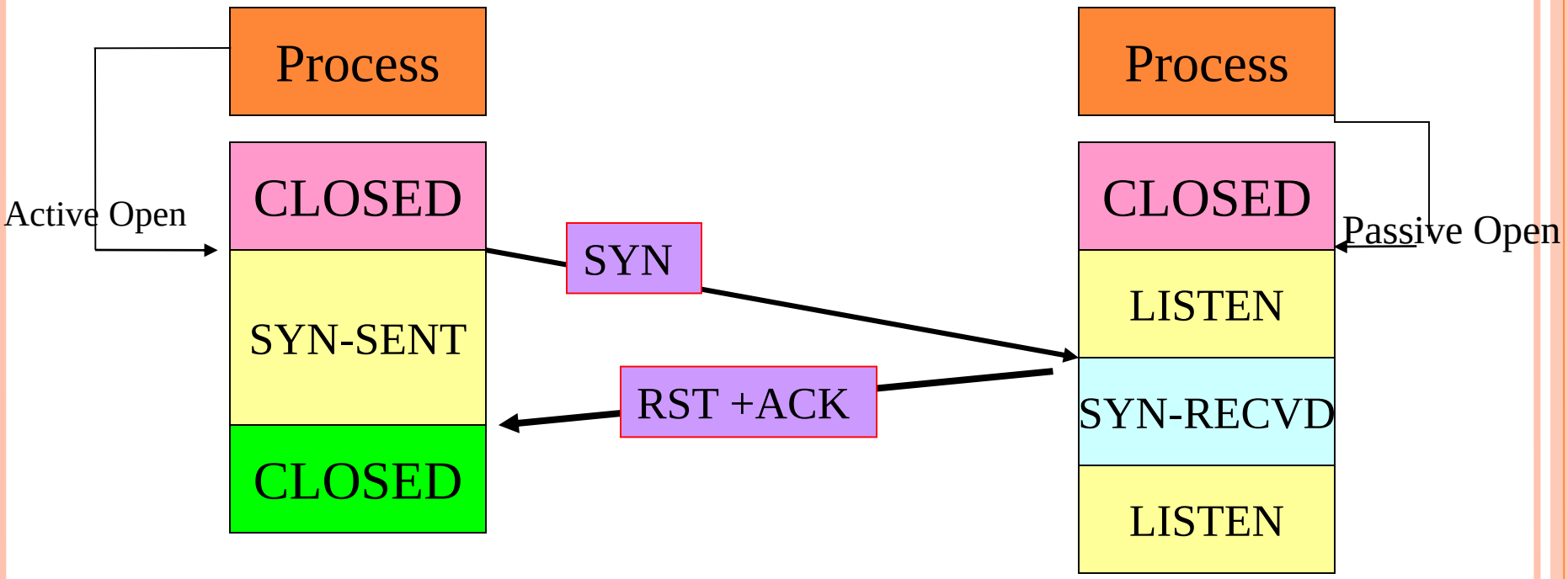
FOUR-WAY HANDSHAKING (CT)



SIMULTANEOUS CLOSE

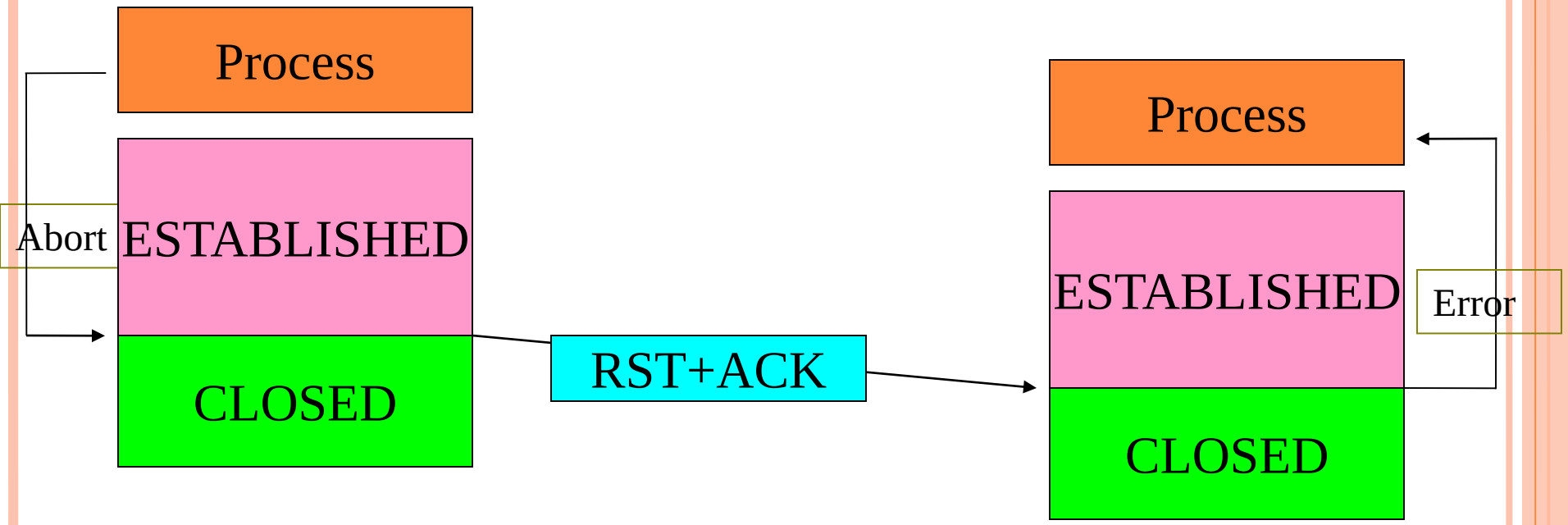


DENYING A CONNECTION



Server Denies the connection because the destination port number In the SYN segment defines a server that is not in the LISTEN state At the moment.

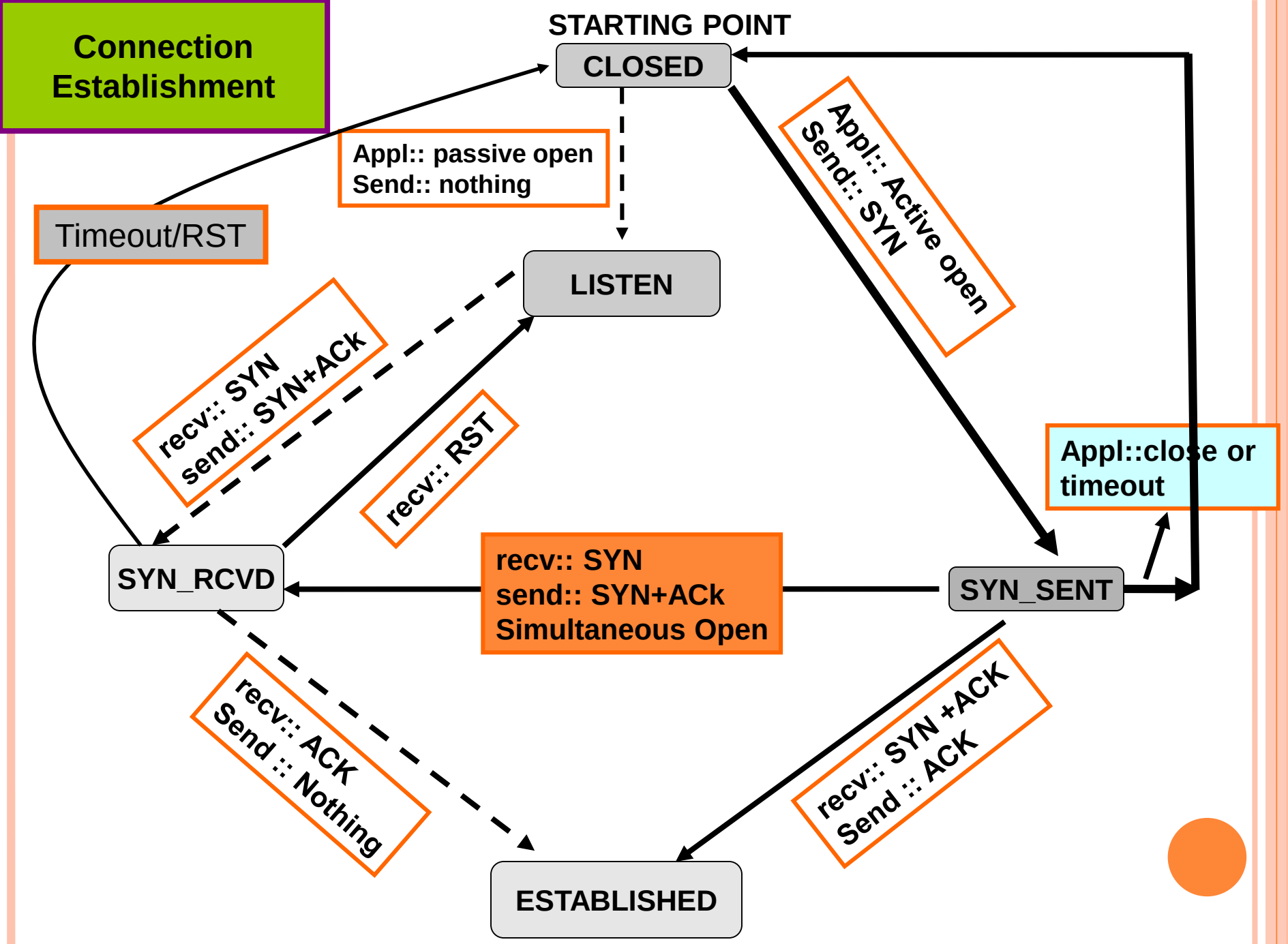
ABORTING A CONNECTION

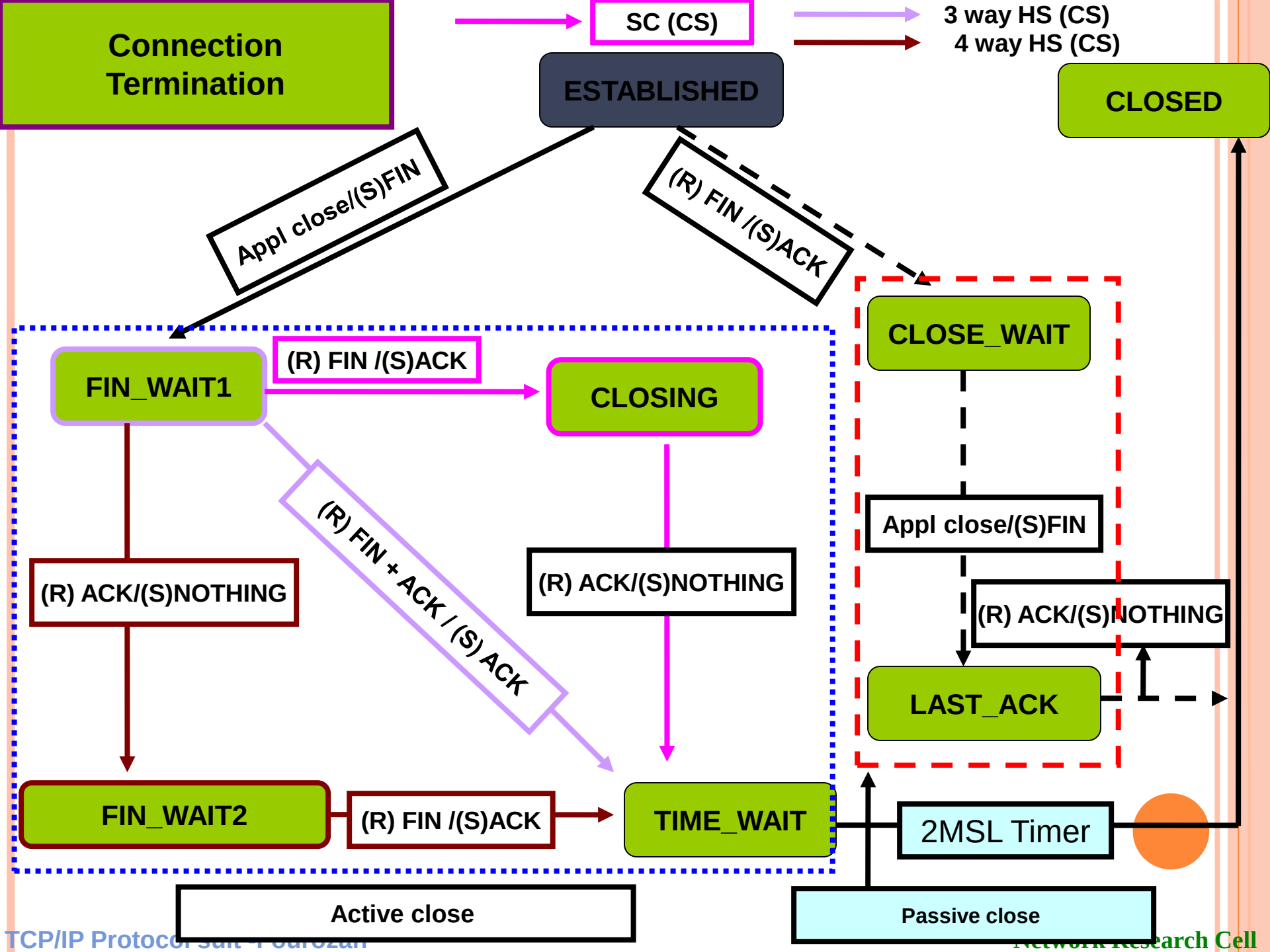


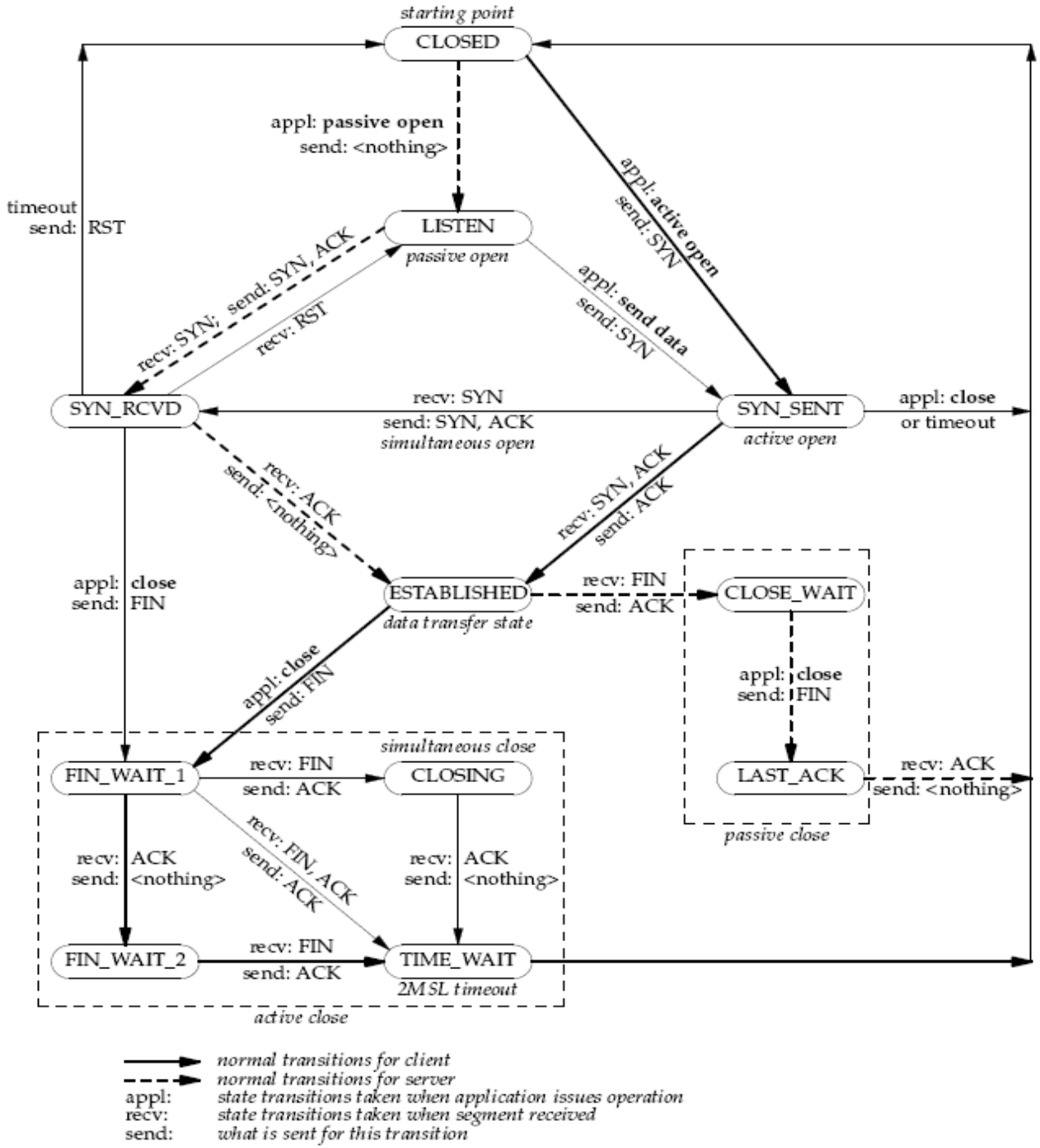
This can happen if the process has failed (infinite loop) or does not Want the data in queue to be sent. (Example ctrl +c)

STATE TRANSITION DIAGRAM

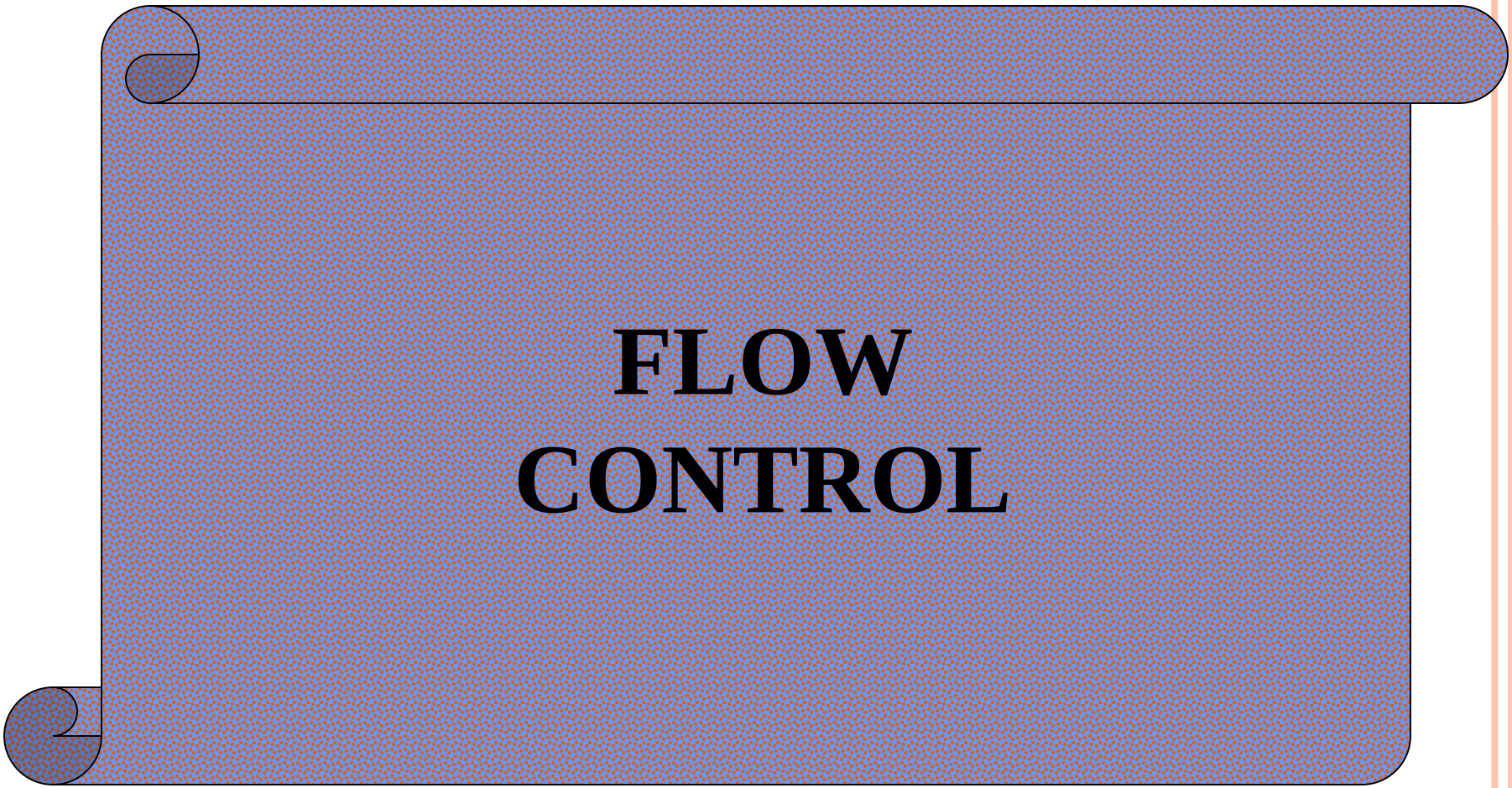
Connection Establishment




Connection Termination
SC (CS)
3 way HS (CS)
4 way HS (CS)
ESTABLISHED
CLOSED
Appl close/(S)FIN
(R) FIN /(S)ACK
CLOSE_WAIT
FIN_WAIT1
(R) FIN /(S)ACK
CLOSING
Appl close/(S)FIN
(R) ACK/(S)NOTHING
(R) FIN + ACK / (S) ACK
(R) ACK/(S)NOTHING
(R) ACK/(S)NOTHING
LAST_ACK
FIN_WAIT2
(R) FIN /(S)ACK
TIME_WAIT
2MSL Timer
Active close
Passive close


starting point
CLOSED
appl: passive open
send: <nothing>
appl: active open
send: SYN
timeout
send: RST
LISTEN
passive open
recv: SYN; send: SYN, ACK
recv: RST
appl: send data
send: SYN
SYN_RCVD
recv: SYN
send: SYN, ACK
simultaneous open
SYN_SENT
active open
appl: close
or timeout
recv: ACK
send: <nothing>
recv: SYN, ACK
send: ACK
appl: close
send: FIN
ESTABLISHED
data transfer state
recv: FIN
send: ACK
CLOSE_WAIT
appl: close
send: FIN
appl: close
send: FIN
simultaneous close
FIN_WAIT_1
recv: FIN
send: ACK
CLOSING
LAST_ACK
recv: ACK
send: <nothing>
passive close
recv: ACK
send: <nothing>
recv: FIN, ACK
send: ACK
recv: ACK
send: <nothing>
FIN_WAIT_2
recv: FIN
send: ACK
TIME_WAIT
2MSL timeout
active close
normal transitions for client
normal transitions for server
appl: state transitions taken when application issues operation
recv: state transitions taken when segment received
send: what is sent for this transition

FLOW CONTROL



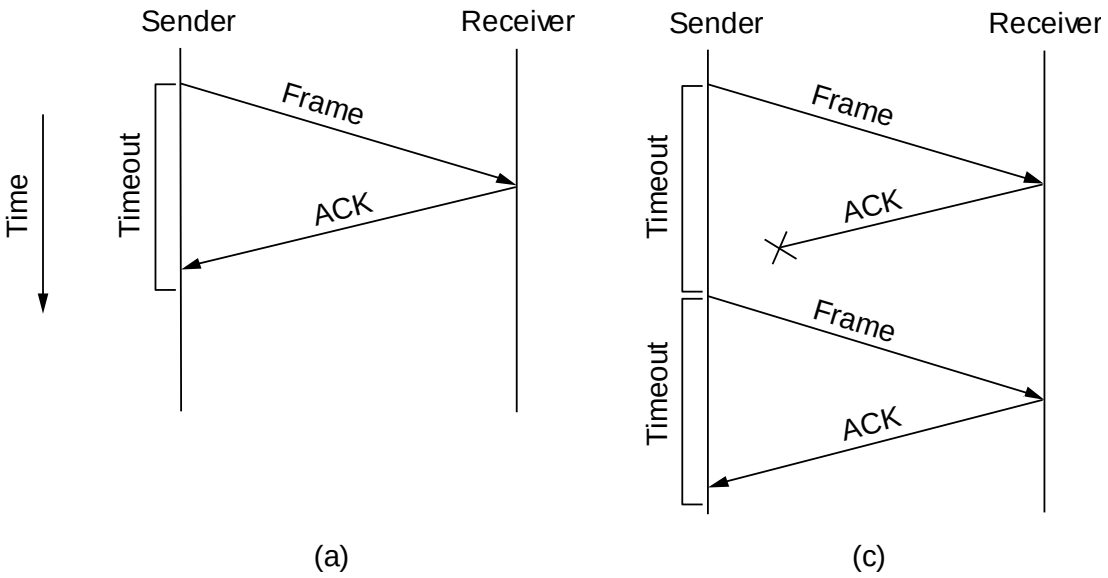
Methods of Reliability

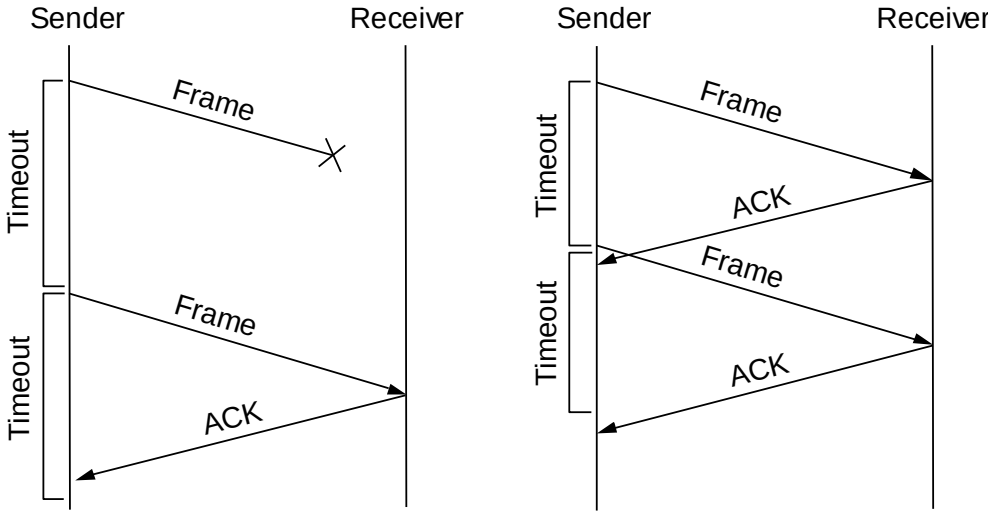
- **Packets can be lost and/or corrupted during transmission**
  - Bit level errors due to noise
  - Loss due to congestion
- **Use checksums to detect bit level errors**
  - Internet Checksum is optionally used to detect errors in UDP
    - Uses 16 bits to encode one's complement sum of data + headers
  - When bit level errors are detected, packets are dropped
- **Build reliability into the transmission protocol**
  - Using *acknowledgements* and *timeouts* to signal lost or corrupt frame

Acknowledgements & Timeouts

- **An *acknowledgement* (ACK) is a packet sent by one host in response to a packet it has received**
  - Making a packet an ACK is simply a matter of changing a field in the transport header
  - Data can be *piggybacked* in ACKs
- **A *timeout* is a signal that an ACK to a packet that was sent has not yet been received within a specified timeframe**
  - A timeout triggers a *retransmission* of the original packet from the sender
  - How are timers set?

Acknowledgements & Timeouts





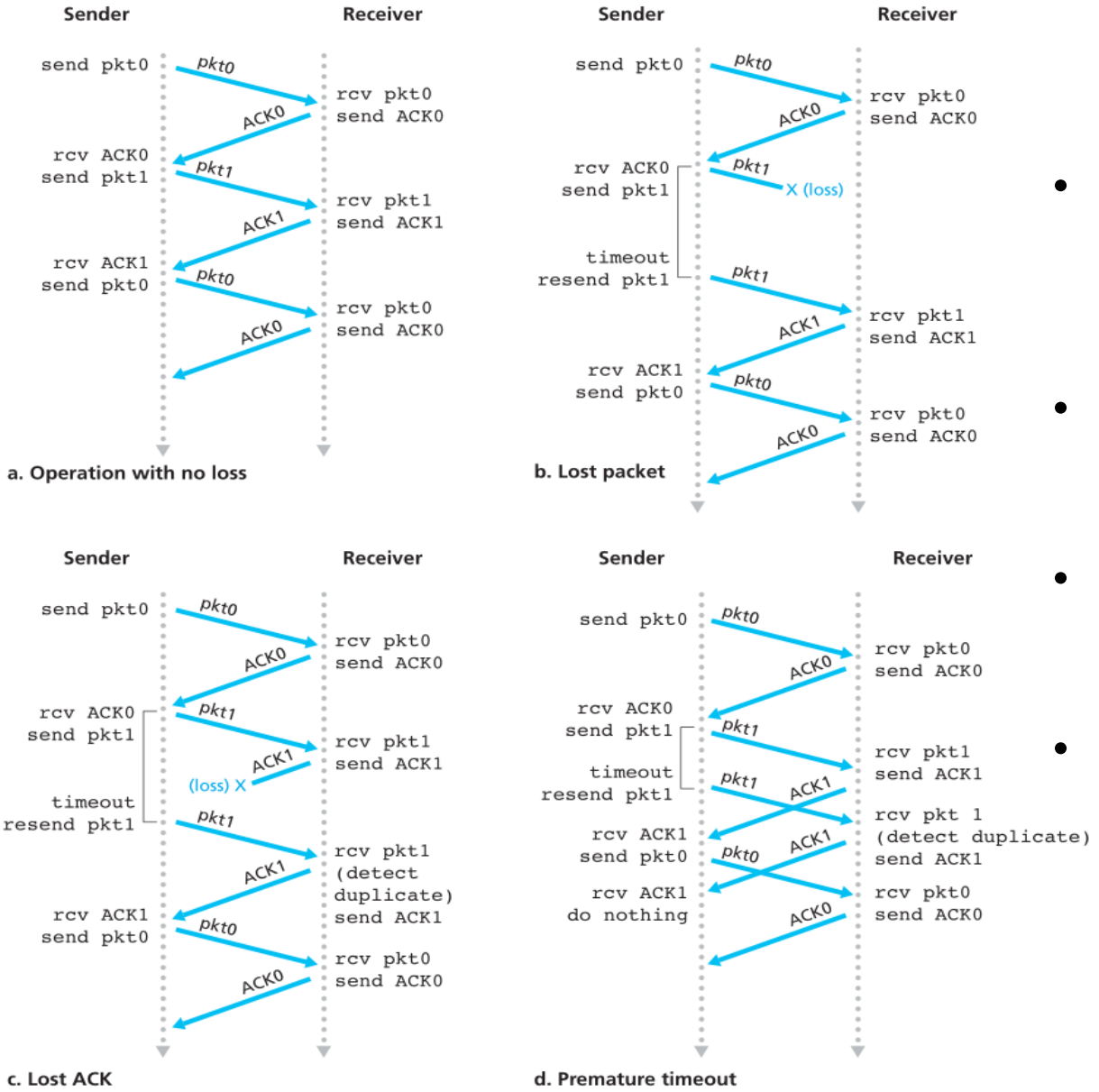
Propagation Delay

- Propagation delay is defined as the delay between transmission and receipt of packets between hosts
- Propagation delay can be used to estimate timeout period
- How can propagation delay be measured?
- What else must be considered in the measurement?

Exponentially weighted moving average RTT estimation

- EWMA was original algorithm for TCP
- Measure **SampleRTT** for each packet/ACK pair
- Compute weighted average of RTT
  - **EstRTT** = α x **EstimatedRTT** + β x **SampleRTT**
  - where $\alpha + \beta = 1$
  - α between 0.8 and 0.9
  - β between 0.1 and 0.2
- Set timeout based on **EstRTT**
  - **TimeOut** = 2 x **EstRTT**

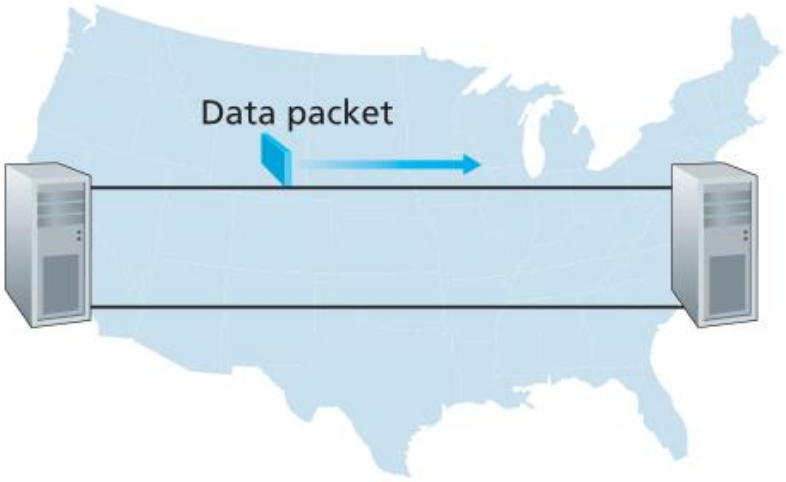
Stop & Wait Protocol



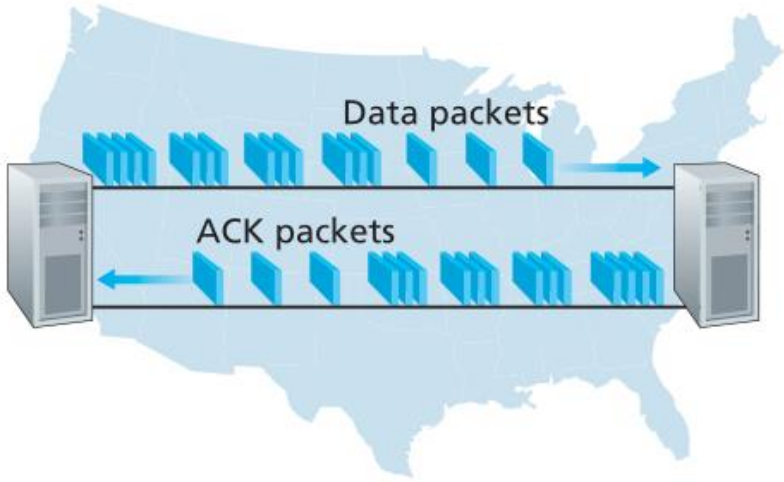
- Sender doesn't send next packet until he's sure receiver has last packet
- The packet/Ack sequence enables reliability
- Sequence numbers help avoid problem of duplicate packets
- Problem: keeping the pipe full

Solution: Pipelining via Sliding Window

- Allow multiple outstanding (un-ACKed) frames
- Upper bound on un-ACKed frames, called *window*



a. A stop-and-wait protocol in operation

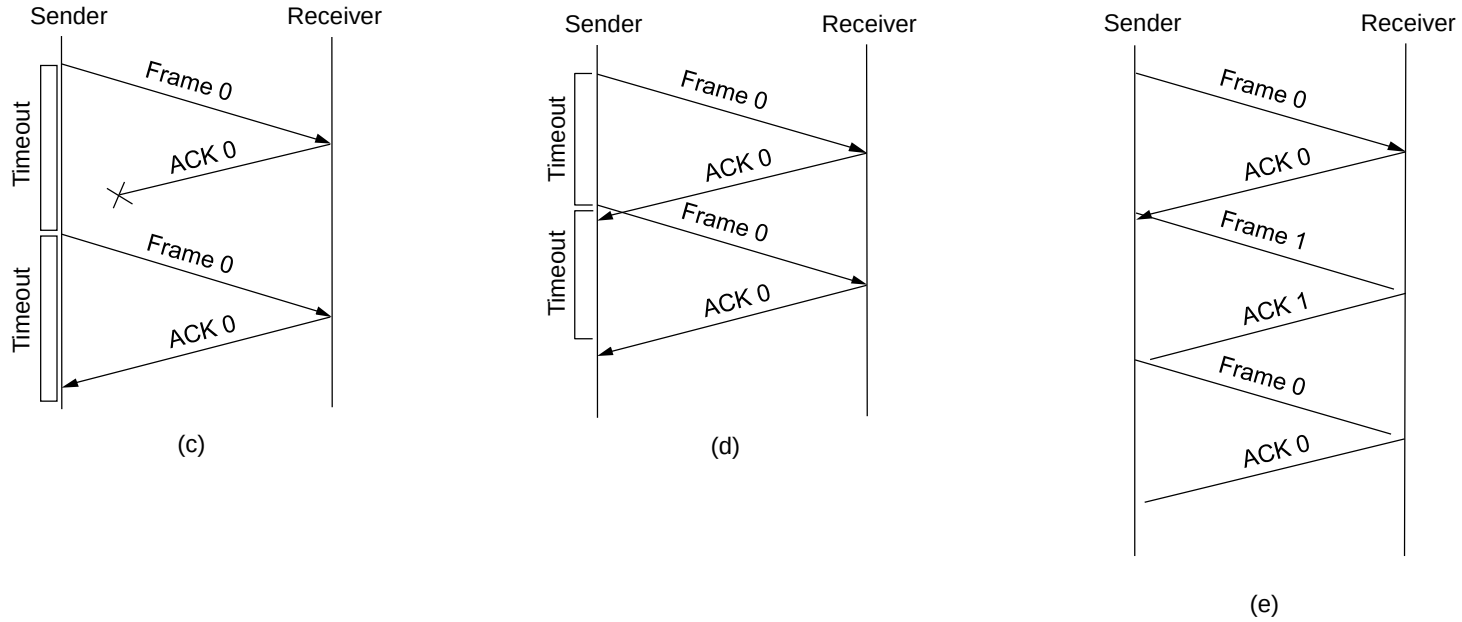


b. A pipelined protocol in operation

Buffering on Sender and Receiver

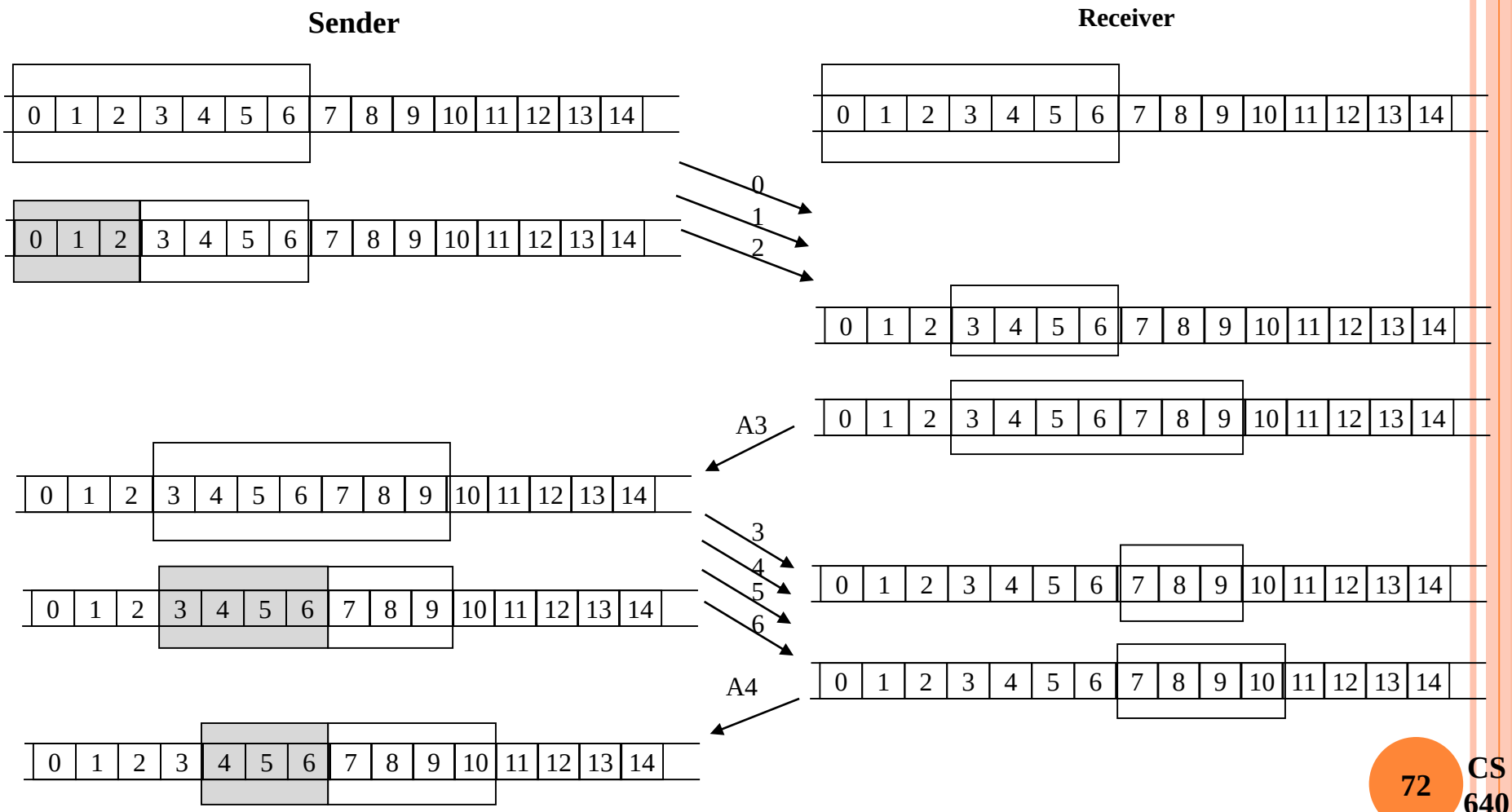
- Sender needs to buffer data so that if data is lost, it can be resent
- Receiver needs to buffer data so that if data is received out of order, it can be held until all packets are received
  - Flow control
- How can we prevent sender overflowing receiver's buffer?
  - Receiver tells sender its buffer size during connection setup
- How can we insure reliability in pipelined transmissions?
  - Go-Back-N
    - Send all N unACKed packets when a loss is signaled
    - Inefficient
  - Selective repeat
    - Only send specifically unACKed packets
    - A bit trickier to implement

Stop & wait sequence numbers



- Simple sequence numbers enable *the client* to discard duplicate copies of the same frame
- Stop & wait allows one outstanding frame, requires two distinct sequence numbers

Sliding Window Example



Note

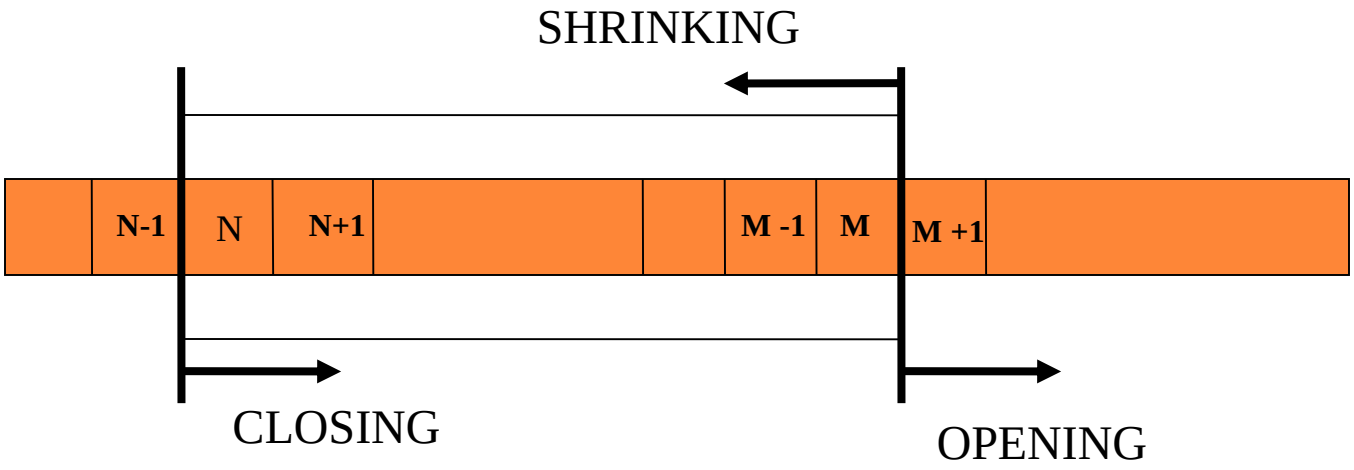
A sliding window is used to make transmission more efficient as well as to control the flow of data so that the destination does not become overwhelmed with data. TCP's sliding windows are byte oriented.

SLIDING WINDOW PROTOCOL

- To Accomplished flow control ,TCP uses a Sliding Window Protocol.
- Client can be sent data without worrying about acknowledgement.
- The window is **opened, closed and shrunk**.
- These Three activities are in the **control** of the **receiver** not the **sender**

SLIDING WINDOW PROTOCOL

Window Size=minimum(rwnd , cwnd)

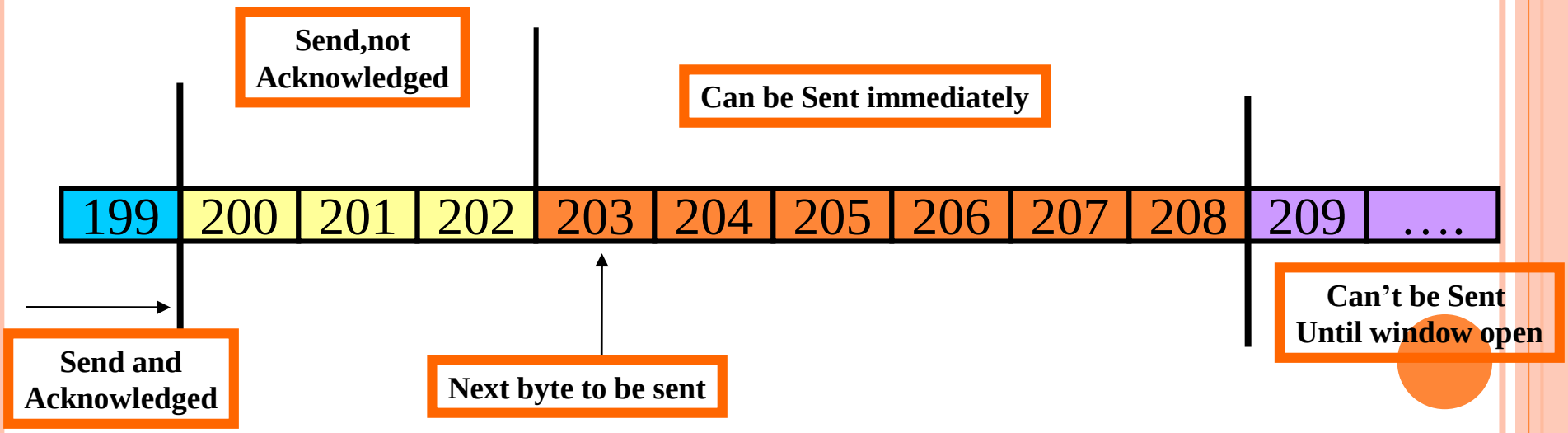


rwnd=receiver window size
cwnd=congestion window size

SLIDING WINDOW PROTOCOL

cwnd=20 bytes
The Receiver has sent an acknowledgement number of 200 with rwnd of 9 bytes .
Bytes 200 to 202 sent ,but not yet acknowledged.

Window size=min(20,9)=9



SLIDING WINDOW PROTOCOL

Problem:-The host receive a packet with an acknowledgement Value of 206 and an rwnd of 12.The value of cwnd is still 20.The host has sent bytes 206 to 209.Show the new window.

Now receiver receive a packet with an acknowledgement Value of 210 and an rwnd of 5.The value of cwnd is still 20. and previous rwnd is 12.Show the new window.

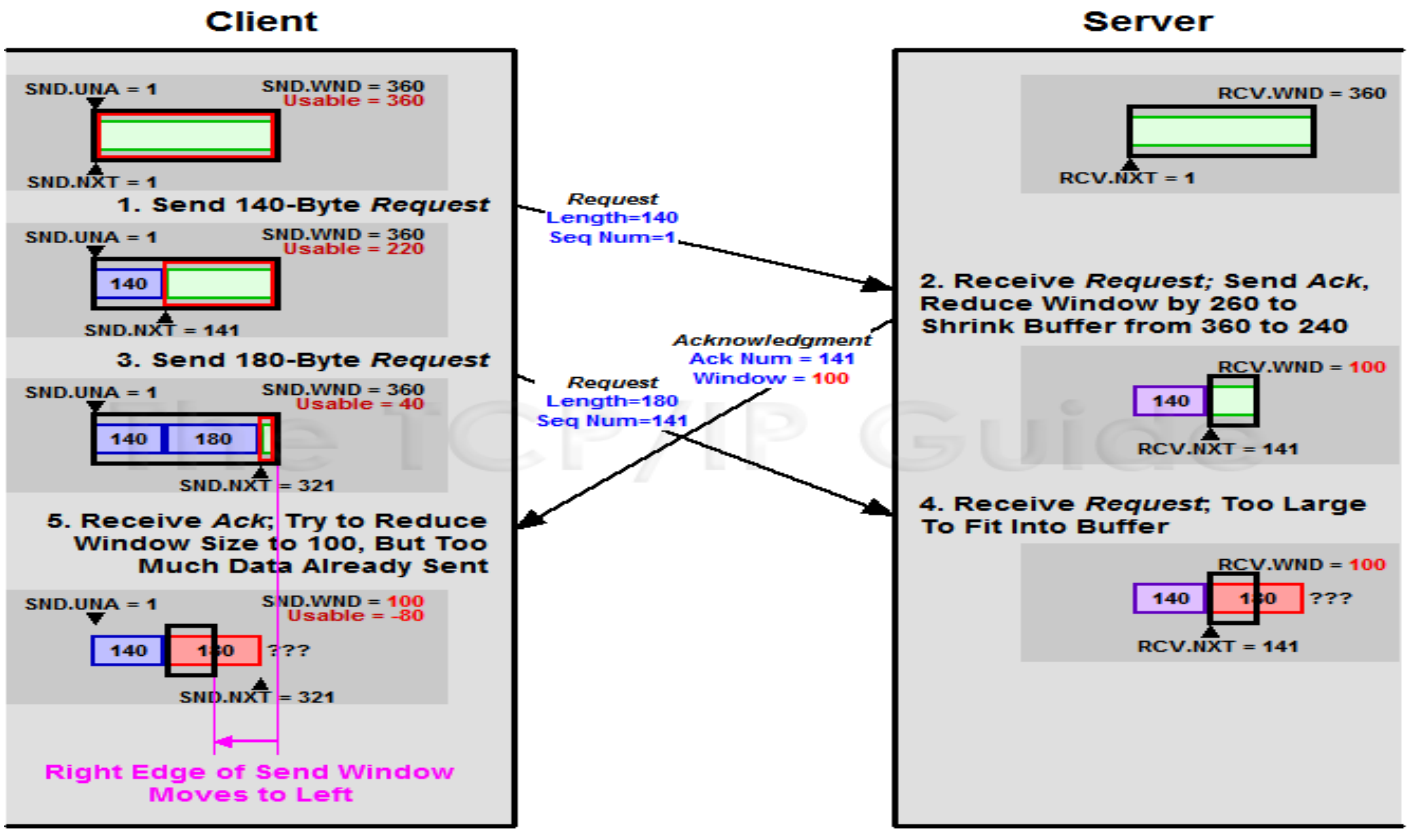
1.Receiver always avoid shrinking.

2.To avoid shrinking the sender window ,the receiver must wait More space is available in the buffer.

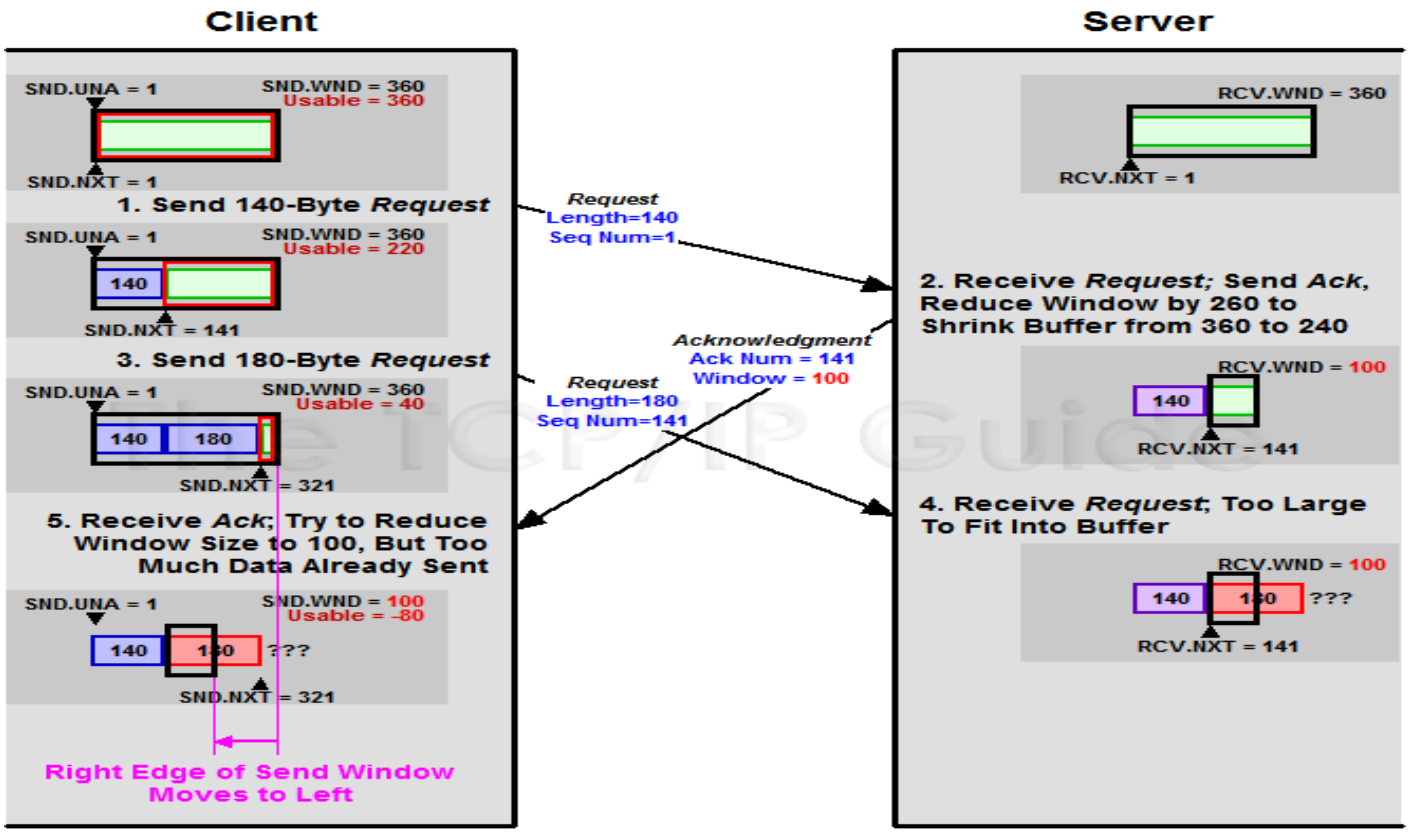
3.To prevent the right wall from the moving to the left , we must Always have the following relationship,

New ack + New rwnd >= Last ack + Last rwnd

Why Shrinking is always avoided?



- In this modification of the example of Figure , the client begins with a **usable window size of 360**.
- It sends a **140-byte segment** and then a short time thereafter sends one of **180 bytes**.
- The server is busy, however, and when it receives the first transmission decides to reduce its buffer to **240 bytes (from 360 byte)**.



- It holds the **140 bytes** just received and reduces its **receive window all the way down to 100**.
- When the client's **180-byte segment arrives**, there is only room for ***100 of the 180 bytes in the server's buffer***.
- When the client gets the new window size advertisement of 100, it will have a problem because it already has 180 bytes sent but not acknowledged.

Note

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.

Note

Some Points about TCP's Sliding Windows:

1. *The source does not have to send a full window's worth of data.*
2. *The size of the window can be increased or decreased by the destination.*
3. *The destination can send an acknowledgment at any time.*

Silly Window Syndrome

- Problem in the sliding window operation
  - The sending process creates data slowly
  - Or the receiving process consume data slowly
  - Or both
  - Result in the *sending* of data in very small segment
    - Reduce the network efficiency
- For example, send a one byte segment result in overhead of 41/1
  - Assume TCP header is 20 bytes + IP header is 20 bytes

We are using the capacity of the network very inefficient way

Syndrome Created by the Sender

- Sender application create data too slowly
  - For example, only 1 byte at a time
  - Sending TCP would create segments containing 1 byte of data
- Solution: prevent the sending TCP from sending the data *byte by byte*
  - Sending TCP must be forced to wait as it collects data to send in a larger block
  - But, how long should the sending TCP wait?

Nagle's Algorithm

- Nagle found the solution to above syndrome
  - The sending TCP sends the first piece of data it receives from the sending application
    - Even if it is only 1 byte
  - After sending the first segment, the sending TCP accumulates data in the buffer and wait until
    - Either the receiving TCP sends an acknowledge
    - Or until enough data has accumulated to fill a *maximum-sized segment*
  - Step 2 is repeated. Segment 3 is sent if an acknowledgment is received for segment 2 or enough data is accumulated to fill a maximum-size segment

Nagle's Algorithm (Cont.)

- Elegance
  - Very simple
  - Take into account *the speed of the application that creates the data* and *the speed of the network that transports the data*
    - If application is faster than the network
      - The segments are larger
    - If the application is slower than the network
      - The segment are smaller

Syndrome Created by the Receiver

- The receiving TCP may also create a silly window syndrome
  - If it is serving an application that consumes data slowly
- For example
  - Sender application create data in 1K byte blocks
  - But the receiving application consumes data 1 byte at a time
  - Assume the receiver buffer is 4K bytes
    - Buffer will be full soon
    - Sender then only can send 1 byte data to the receiver

Syndrome Created by the Receiver (Cont.)

- Solution
  - Clark's solution
  - Delayed acknowledgment

Clark's Solution

- Send an acknowledgment as soon as the data arrives but to announce a window size of zero until
  - Either there is enough space to accommodate a segment of maximum size
  - Or half of the buffer is empty

Delayed Acknowledgment

- Delay sending the acknowledgment
  - When a segment arrives, it is not acknowledged immediately
  - Receiver waits until there is a decent amount of space in its incoming buffer
- Delayed acknowledgment *prevents the sending TCP from sliding it window*

In **this case Sending window "Closing State" will stop**

Delayed Acknowledgment (Cont.)

- Another advantage
  - Reduce traffic
  - The receiver does not have to acknowledge each segment
- Disadvantage
  - The sender may retransmit the unacknowledged segment
- Solution
  - Defines the acknowledgment should not be delayed by more than 500 ms

ERROR CONTROL

ERROR CONTROL

- TCP is a reliable Transport Layer Protocol.
- TCP provides reliability using Error Control.
- Error Control includes mechanism for
  - ***detecting corrupted segments,***
  - ***lost segments,***
  - ***out of order segments*** and
  - ***duplicate*** segments.

ERROR CONTROL

- ACK segments do not consume sequence number and are not acknowledged.

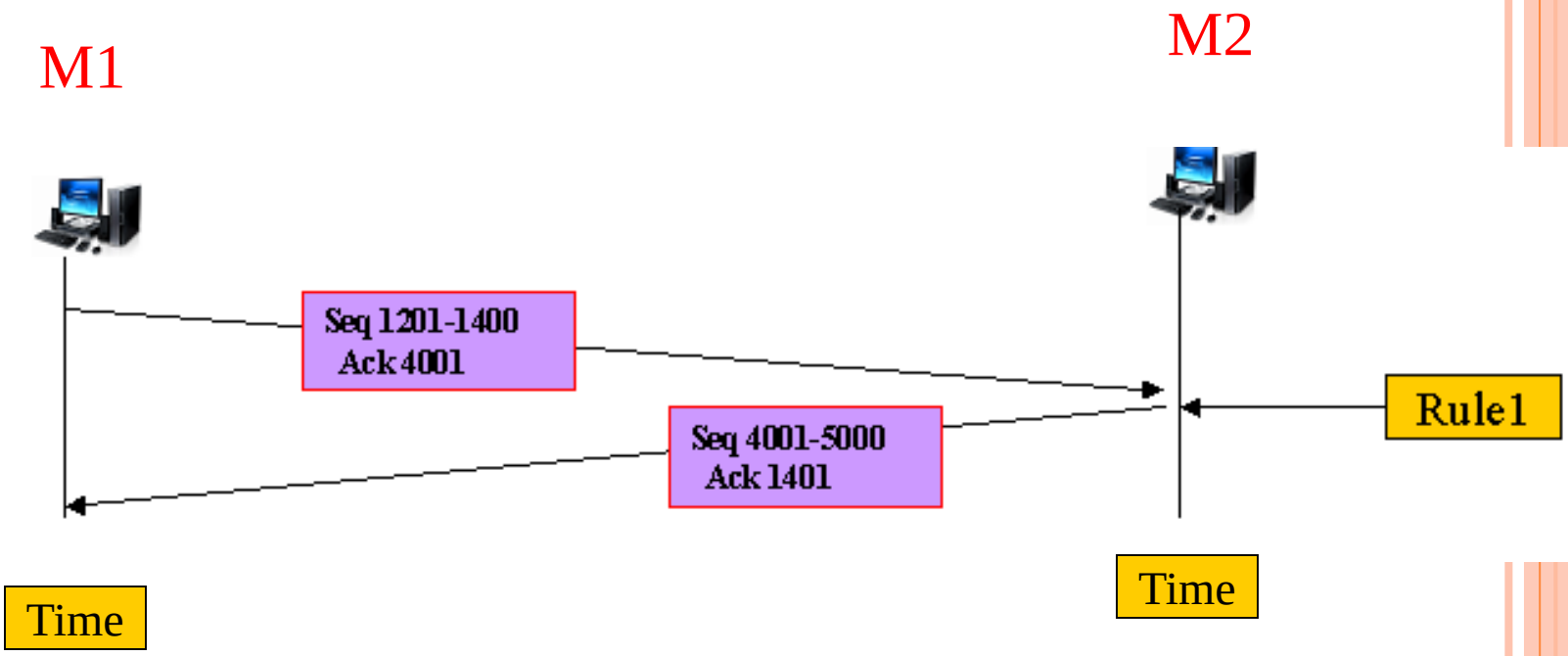
Retransmission

- In modern implementations, a retransmission occurs if the retransmission timer(RTO) expires or Three duplicate ACK segments have arrived.

ACKNOWLEDGEMENT RULE1

Normal Operation

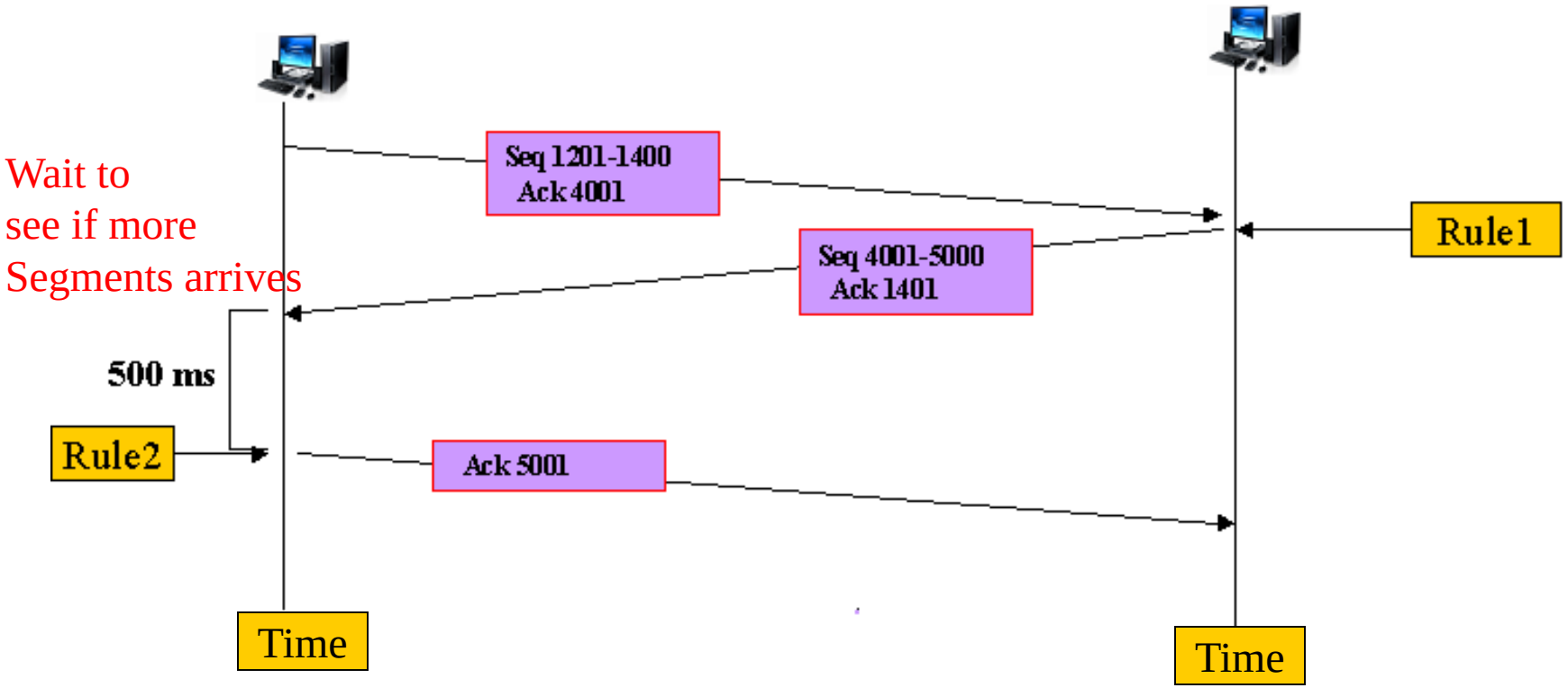
Rule1:- When one ends send a data segment to the other end It include an acknowledgement number (piggyback).



→ Reduce Traffic

ACKNOWLEDGEMENT RULE2

Normal Operation

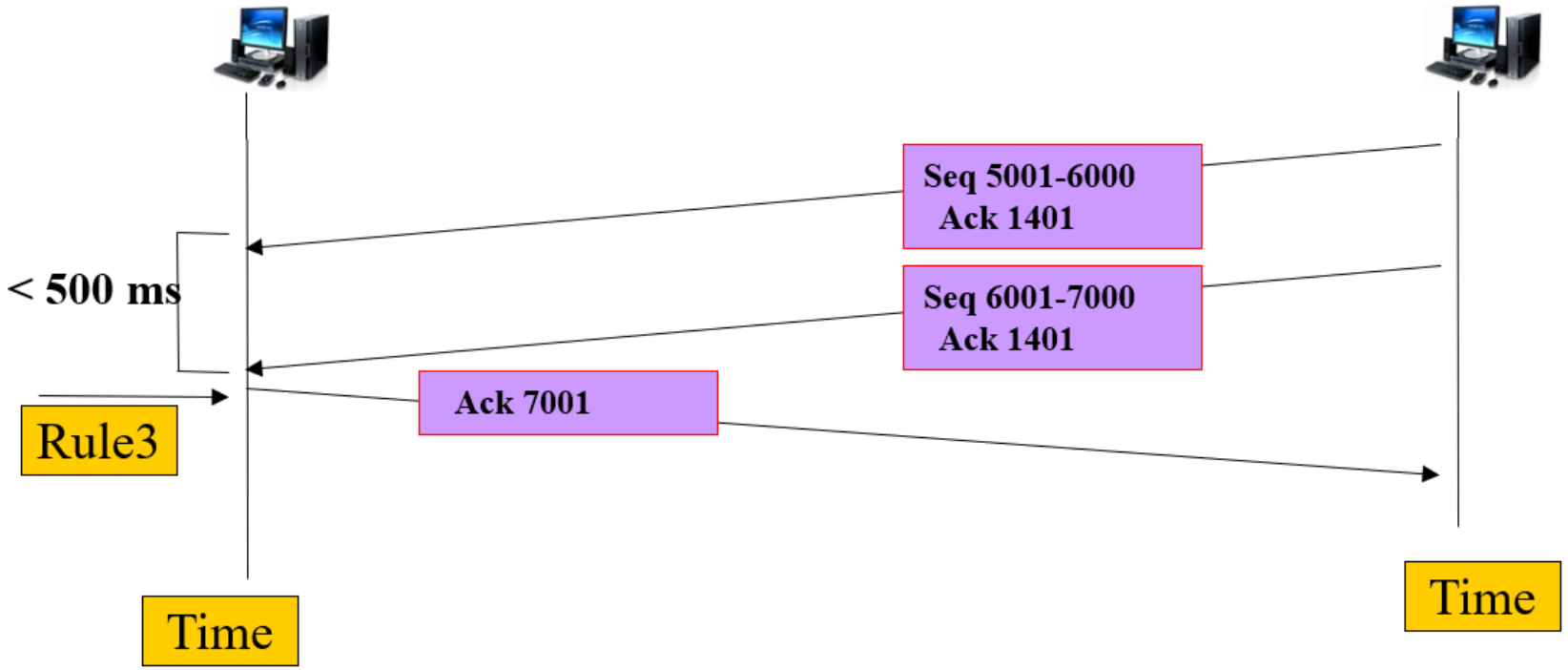


Rule2:- (a) When the receiver has no more data to be send
(b) It receives in-order segment (data:4001-5000)
(c) previous in-order segment already been acknowledgement (ACK1401)

The receiver ***delays sending an acknowledgement*** until ***another segment will come*** or ***until period of time*** has passed.

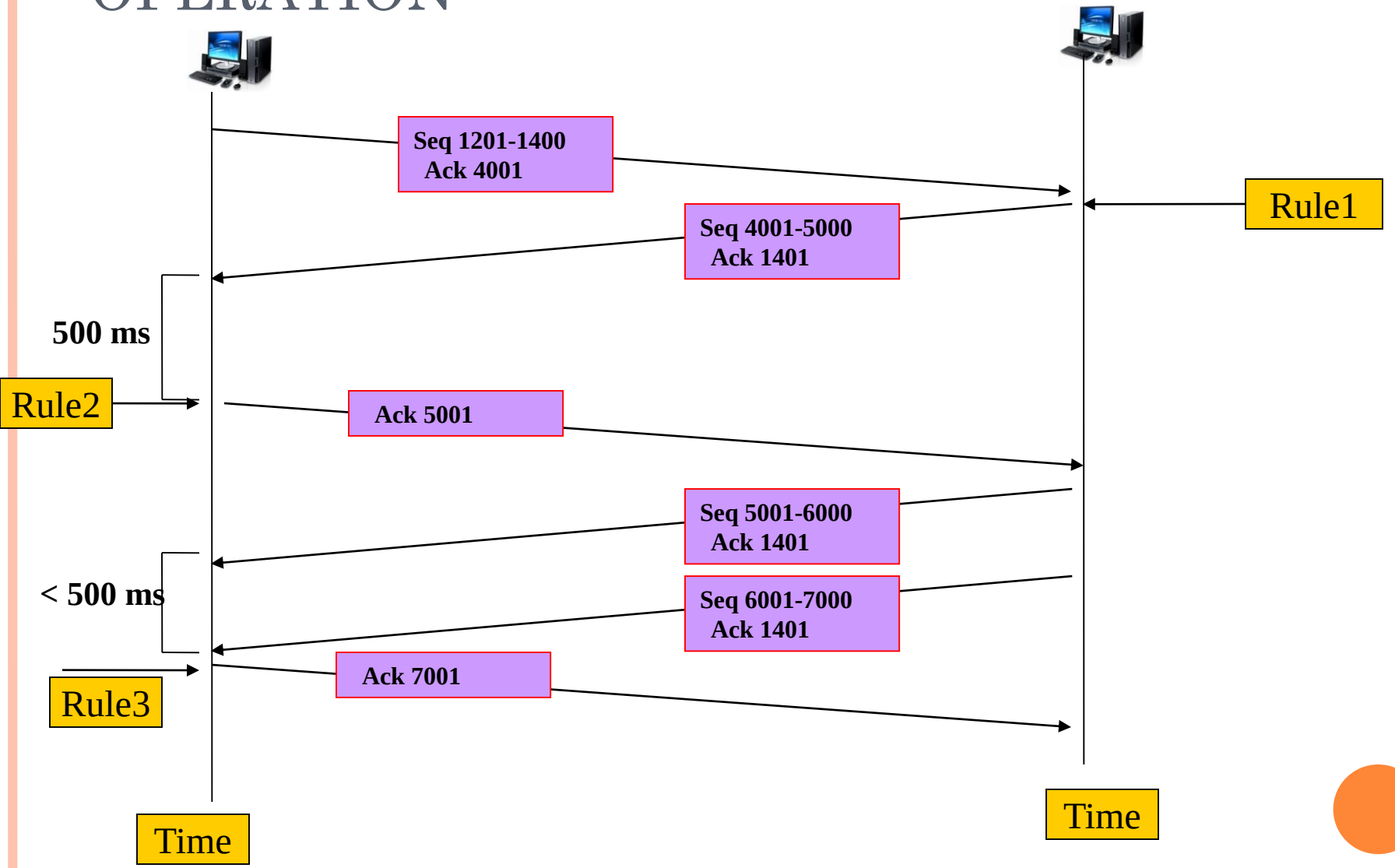
ACKNOWLEDGEMENT RULE3

Normal Operation

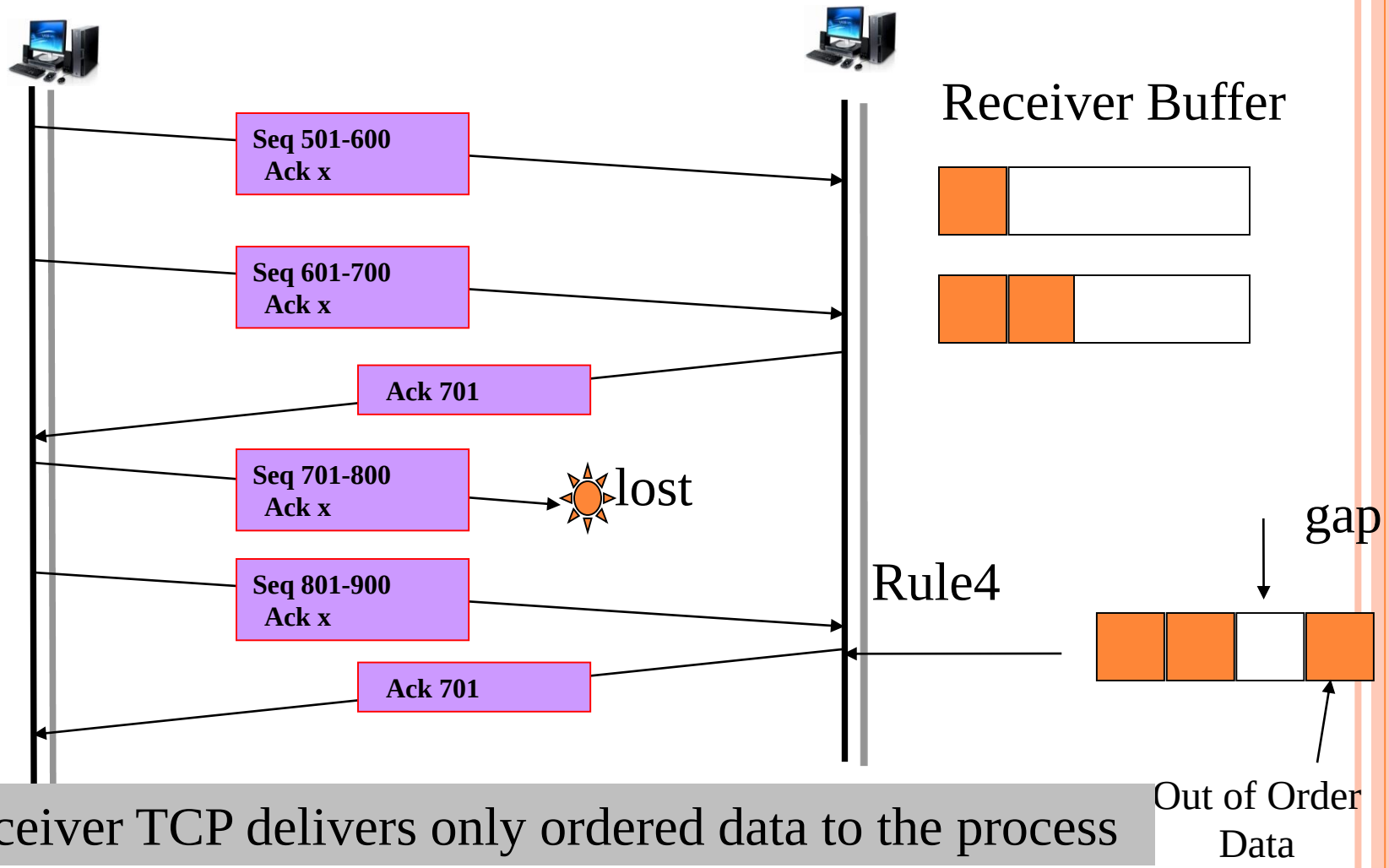


Rule3:-There should not be more than two in-order unacknowledgement Segment at any time.

ACKNOWLEDGEMENT NORMAL OPERATION



LOST SEGMENT → out of order data → RULE4

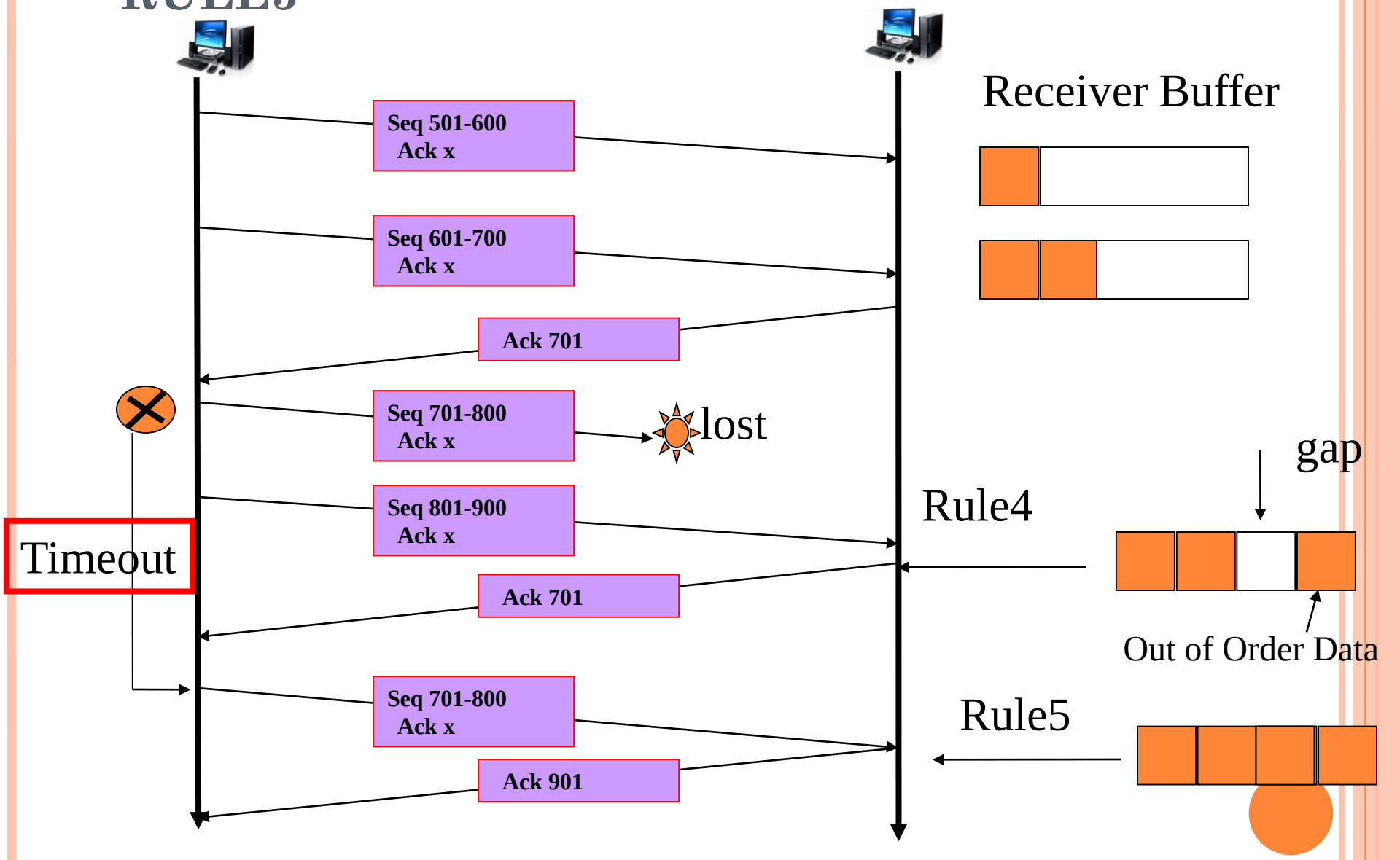


The Receiver TCP delivers only ordered data to the process

When segments arrives with an out-of-order data

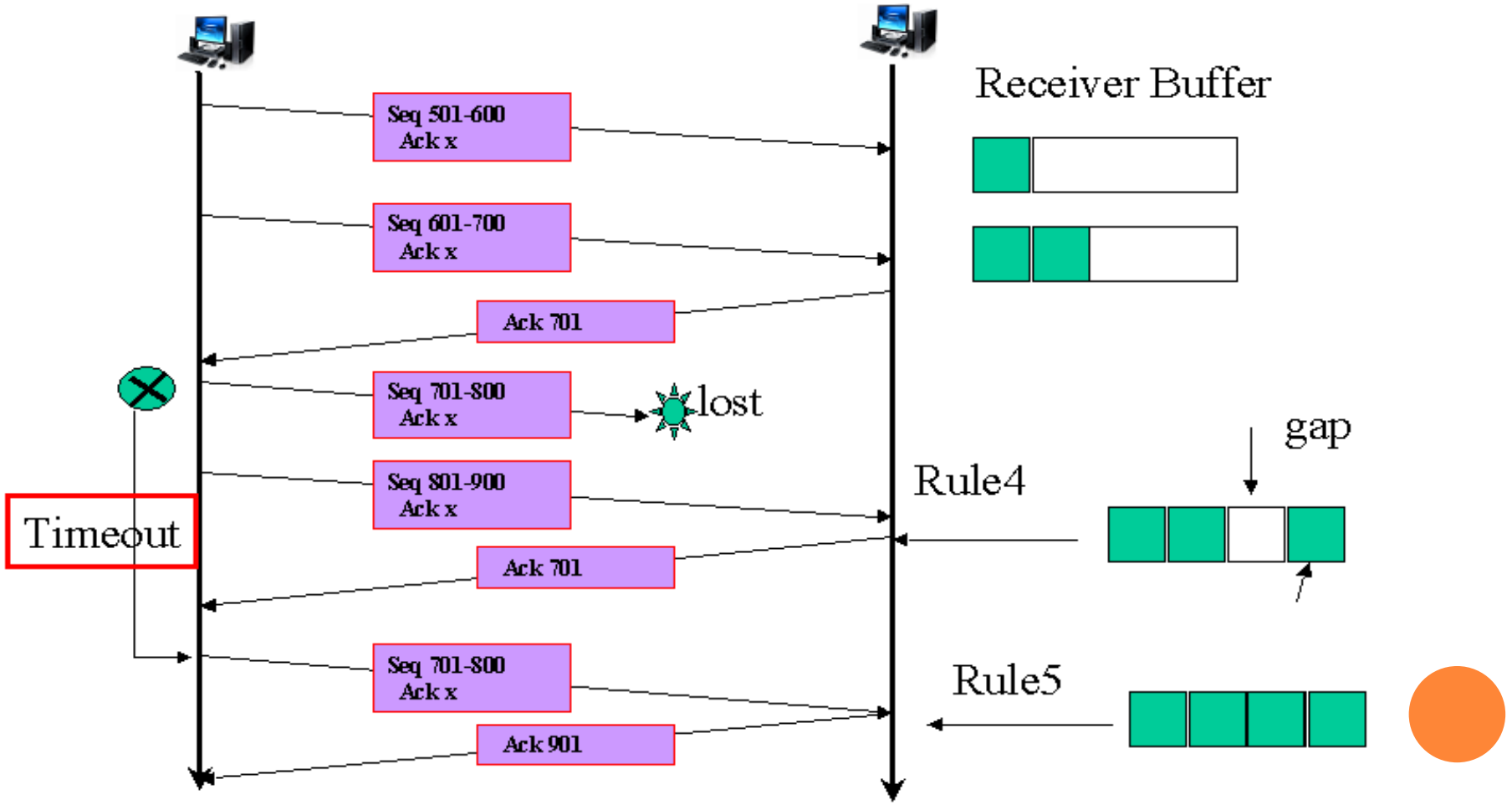
- The receiver Immediately sends an ACK segment.

LOST SEGMENT RULE5



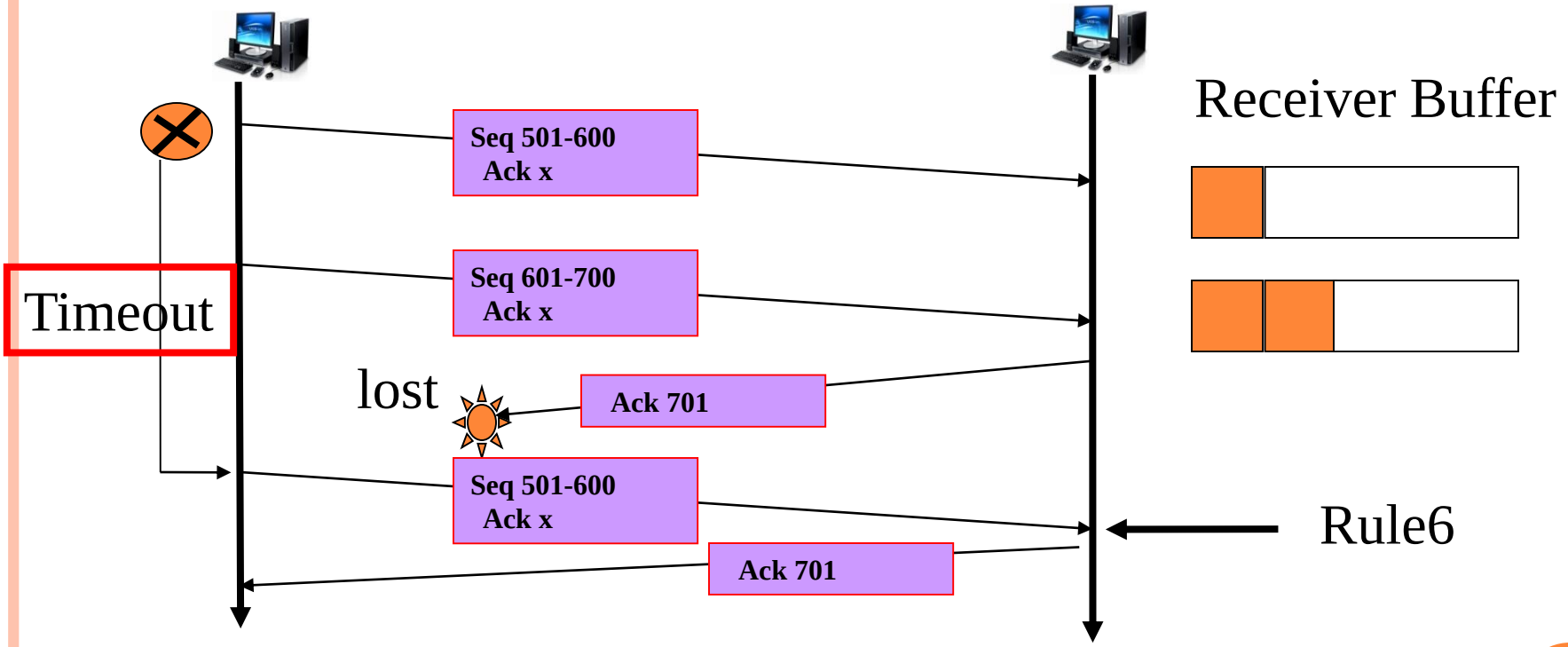
RULE 5 → Received out of order data

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

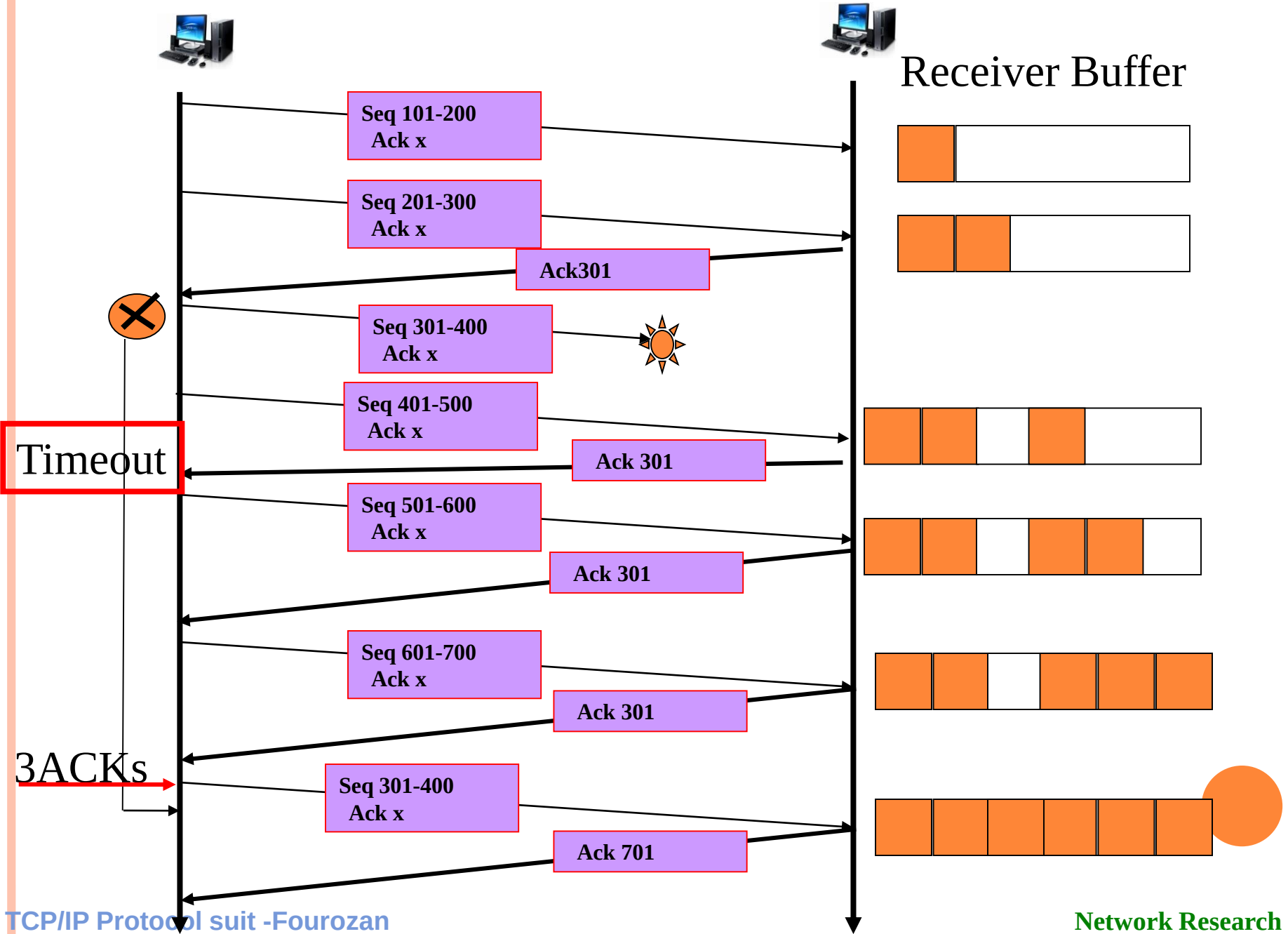


LOST ACKNOWLEDGEMENT RULE6

If a Duplicate segment arrives, the receiver immediately sends An acknowledgement.



FAST RETRANSMISSION



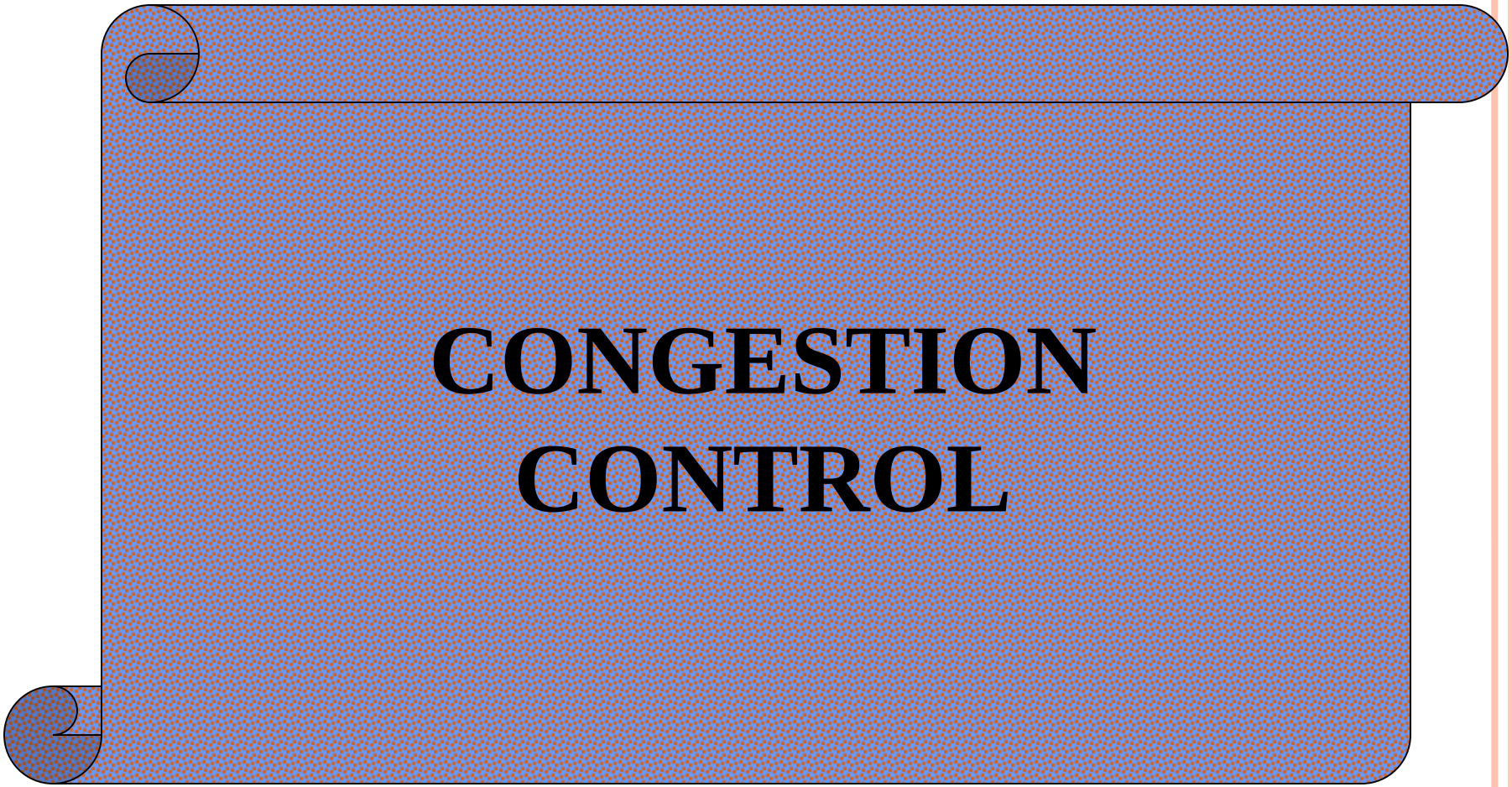
FAST RETRANSMISSION

The sender receives four Acknowledgements with the same value (Three duplicates).

Although the timer for segment 3 has not matured yet.

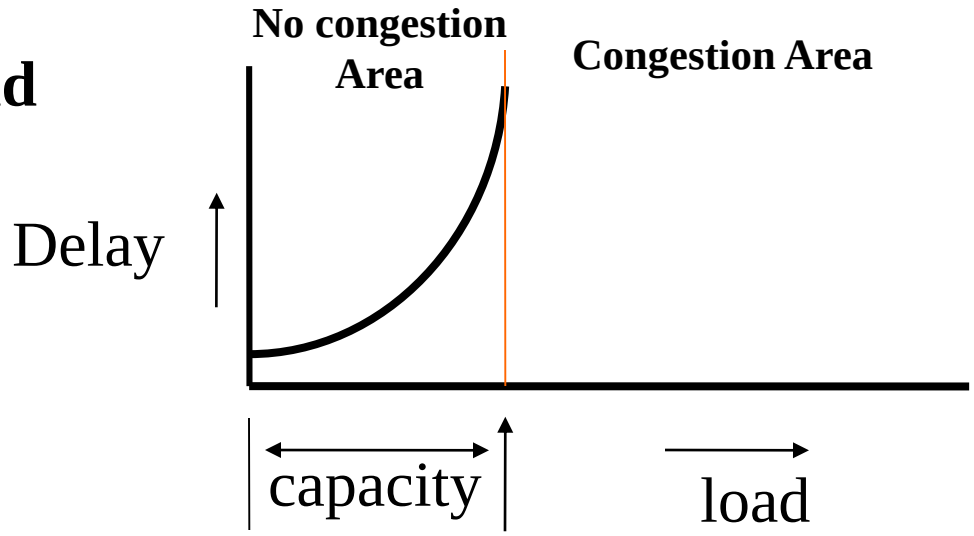
The rule for Fast Transmission requires the segment 3

CONGESTION CONTROL



NETWORK PERFORMANCE

Delay versus Load



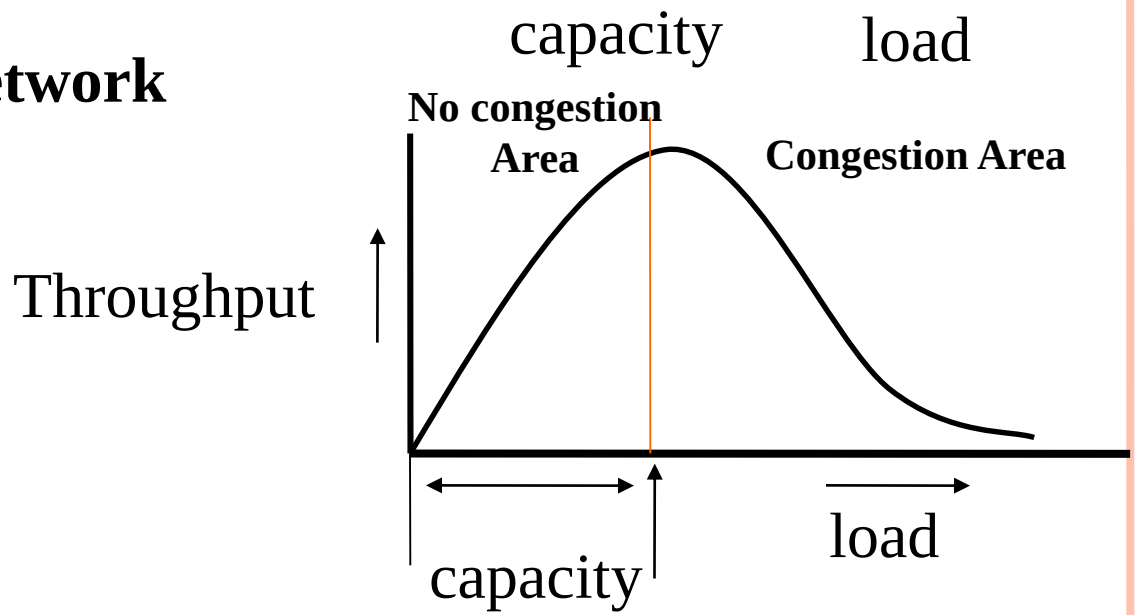
1.Load < < capacity of the Network , Delay is less because size of the Queue & processing delay are negligible.

2. Load = Capacity of the Network, Delay increase sharply because Waiting time is increase in queue.

3. Load >> capacity of the Network ,queues become longer and longer When almost no packet reach the destination.

NETWORK PERFORMANCE

Throughput versus Network Load



CONGESTION CONTROL MECHANISMS

Open Loop Congestion Control Mechanism

- Prevent congestion before it happens

Closed Loop Congestion Control Mechanism

- Remove congestion after it has happened.

OPEN LOOP CONGESTION CONTROL

(Prevent congestion before it happens)

Retransmission Policy

Acknowledgement Policy

Discard Policy

CLOSED LOOP CONGESTION CONTROL

(Remove congestion after it has happened)

Back Pressure- When Router is congested, it inform the **previous upstream router** to reduce the rate of outgoing packets. (Special Control Packet)

Chock Point- A chock point is a packet sent by a router to **the source** to inform it of congestion. (Special Control Packet)

Implicit Signaling-Source can detect an implicit signal (**Delay in receiving an acknowledgement packet**) warning of congestion and slow down its ending rate. (Not sending any Packet or not set any bit)

Explicit Signaling-Experience Router can send an **explicit signal by setting a special bit** in packet's header. (Bit Set in Packet)

CONGESTION CONTROL IN TCP

Congestion Window:-

Actual window size =minimum (rwnd,cwnd);

Sender's actual window size is determined not only by the receiver but also by congestion in the network.

Congestion Policy:-

TCP's general policy for handling congestion is based on Three phases:

1. Slow Start
2. Congestion Avoidance
3. Congestion Detection

Slow Start: Exponential Increase

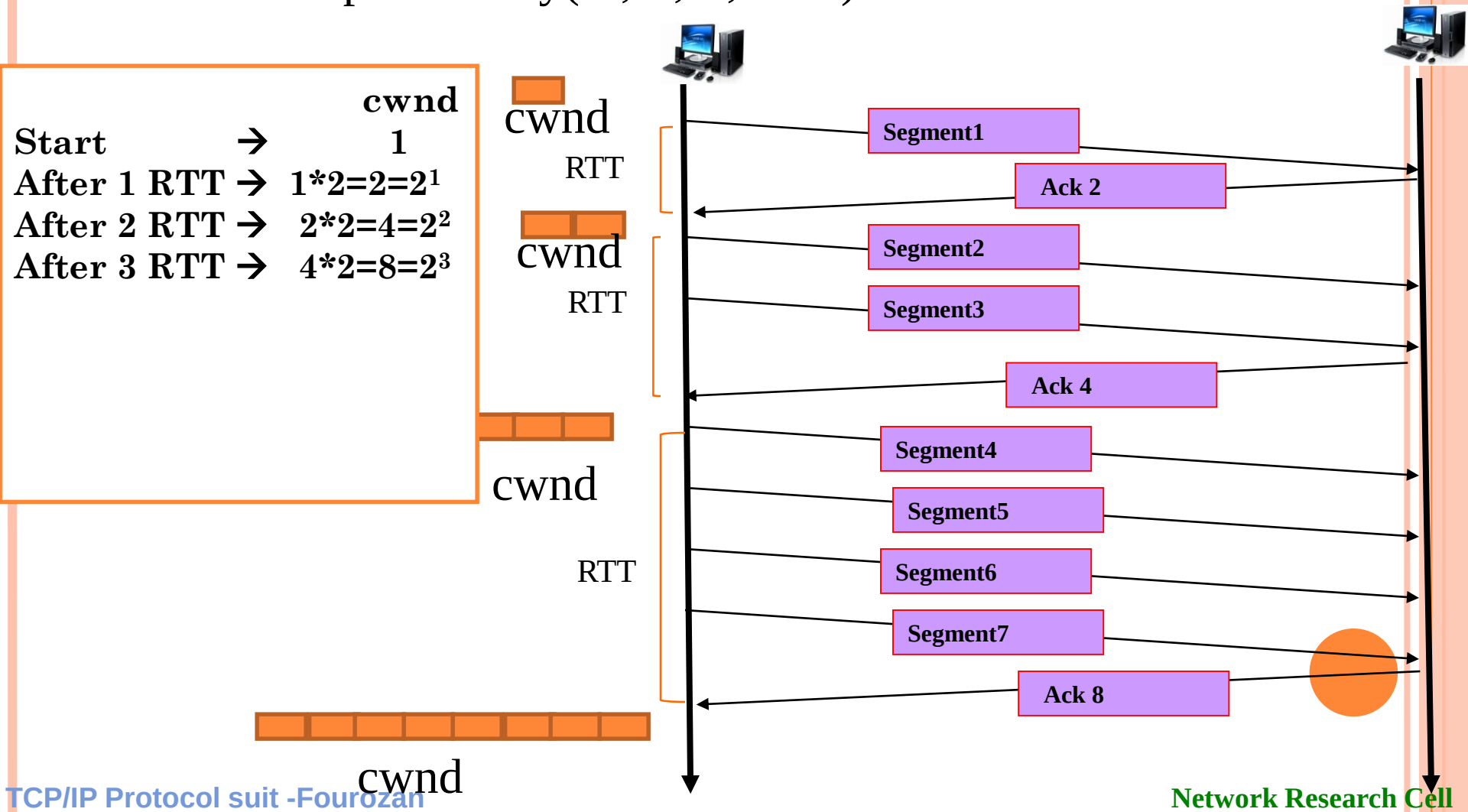
- At the beginning,
  - congestion window size = maximum segment size (MSS)
- MSS is determined during connection establishment using an option (mentioned later)
- For each segment that is acknowledged
  - Increase the congestion window size by one maximum segment unit
  - Until it reaches a *threshold, called ssthresh (show start threshold)*
    - Usually, *ssthresh* = 65535 bytes

Slow Start: Exponential Increase (Cont.)

- However, it is not actually "slow start"
  - The congestion window size increases *exponentially*
  - Start => cwnd = 1 = 2^0
  - After 1 RTT => cwnd = 2 = 2^1
  - After 2 RTT => cwnd = 4 = 2^2
  - After 3 RTT => cwnd = 8 = 2^3

SLOW START: EXPONENTIAL INCREASE

In the slow start algorithm the size of the congestion window Increases exponentially(2^0,2^1,2^2,2^3….) until it reaches a threshold.



Congestion Avoidance: Additive Increase

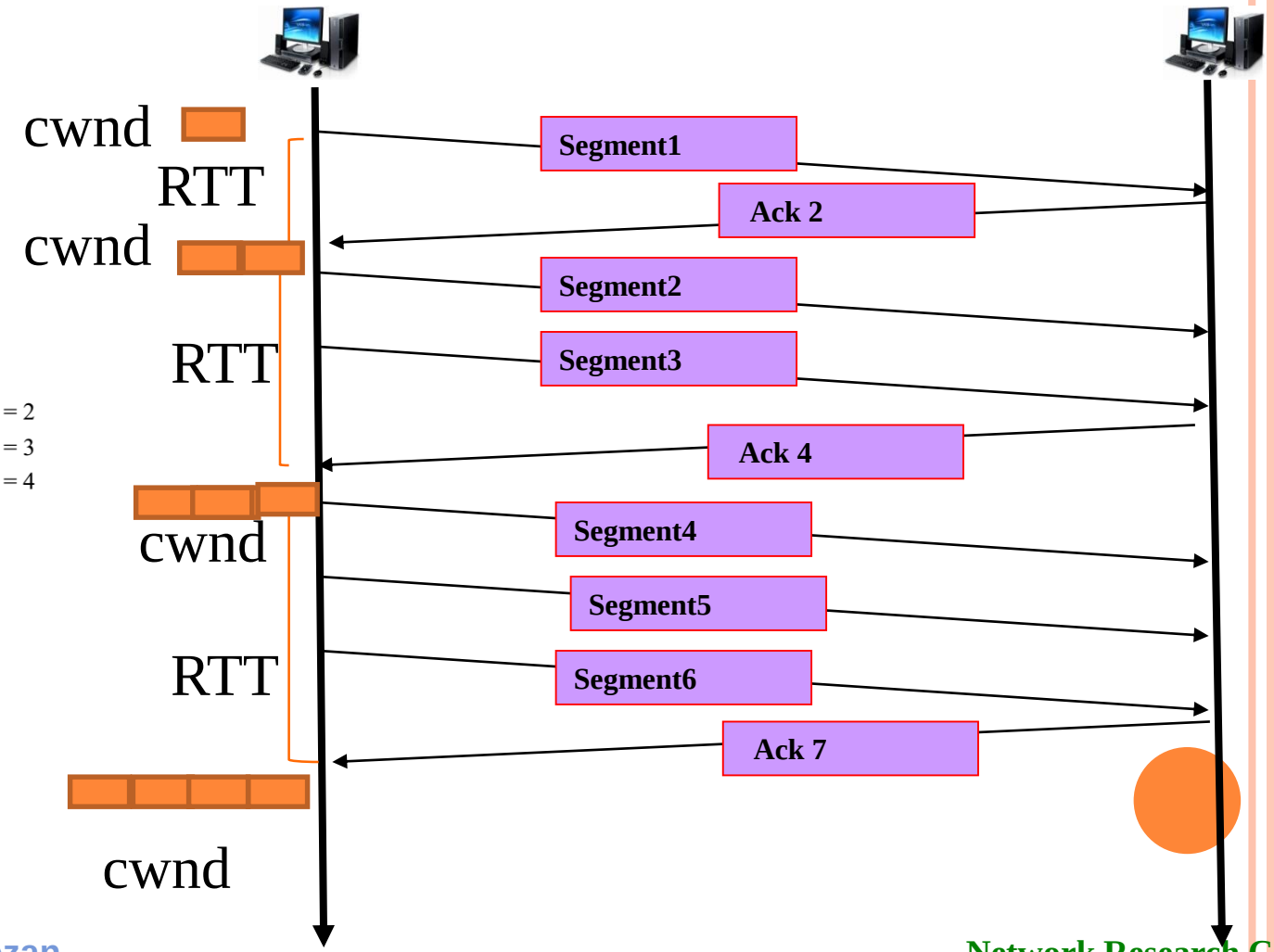
- Started after the congestion window size reaches the *ssthresh threshold*
- When the *whole window of segments* is acknowledged
  - The size of congestion window is increased *one*

Initially it will start after window size reaches ssthresh value

CONGESTION AVOIDANCE: ADDITIVE INCREASE

In the congestion avoidance algorithm the size of the congestion Window increases additively until congestion is detected.

- In the above figure
  - Start => cwnd = 1
  - After 1 RTT => cwnd = 1 + 1 = 2
  - After 2 RTT => cwnd = 2 + 1 = 3
  - After 3 RTT => cwnd = 3 + 1 = 4



Congestion Detection: Multiplicative Decrease

- If congestion occurs, the congestion window size must be decreased
- How to detect a congestion?
  - *The need to retransmit a segment*
- When to retransmit a segment
  - *When an RTO timer out*
  - *When three duplicate ACKs are received*

Congestion Detection: Multiplicative Decrease (Cont.)

- In both cases, the size of the threshold is ***half of the current congestion window size***
  - ***multiplicative decrease***
- However, different actions are taken
  1. If a time-out occurs: a strongly possibility of congestion
     - *The threshold should be set to half of the current congestion window size*
       - *Multiplicative decrease*
     - *The congestion window size should start from one again, i.e., cwnd = 1*
     - *The sender return to the slow start phase*

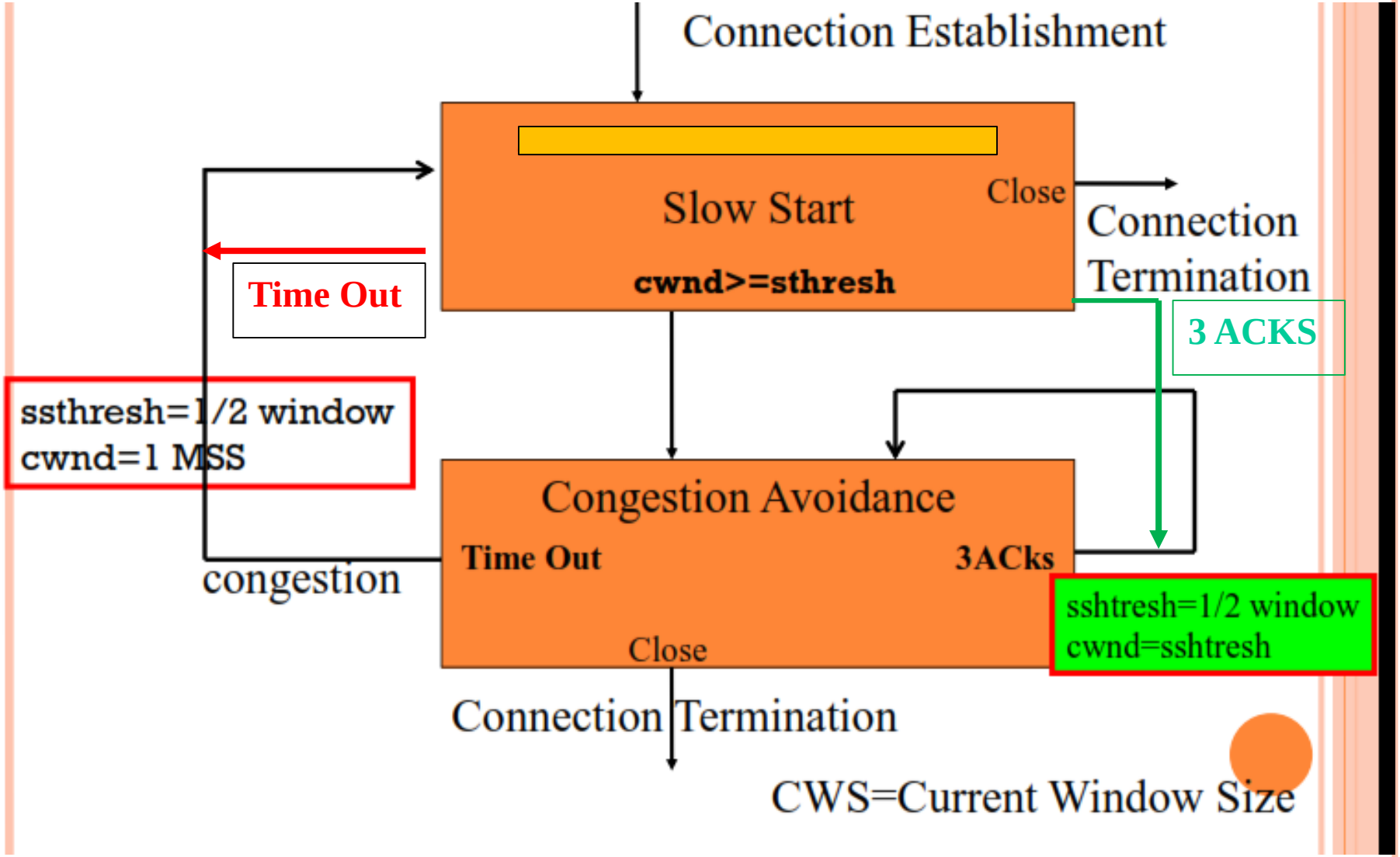
Congestion Detection: Multiplicative Decrease (Cont.)

- If three duplicated ACKs are received: a weaker possibility of congestion
- Invoke ***fast retransmission and fast recovery***
  - *The threshold should be set to half of the current congestion window size*
    - *Multiplicative decrease*
  - *The congestion window size = threshold again, i.e., cwnd = ssthresh*
  - *The sender starts the congestion avoidance phase*

½ of the current window size

ADDITIVE INCREASE

TCP CONGESTION POLICY SUMMARY



Note

TCP assumes that the cause of a lost segment is due to congestion in the network.

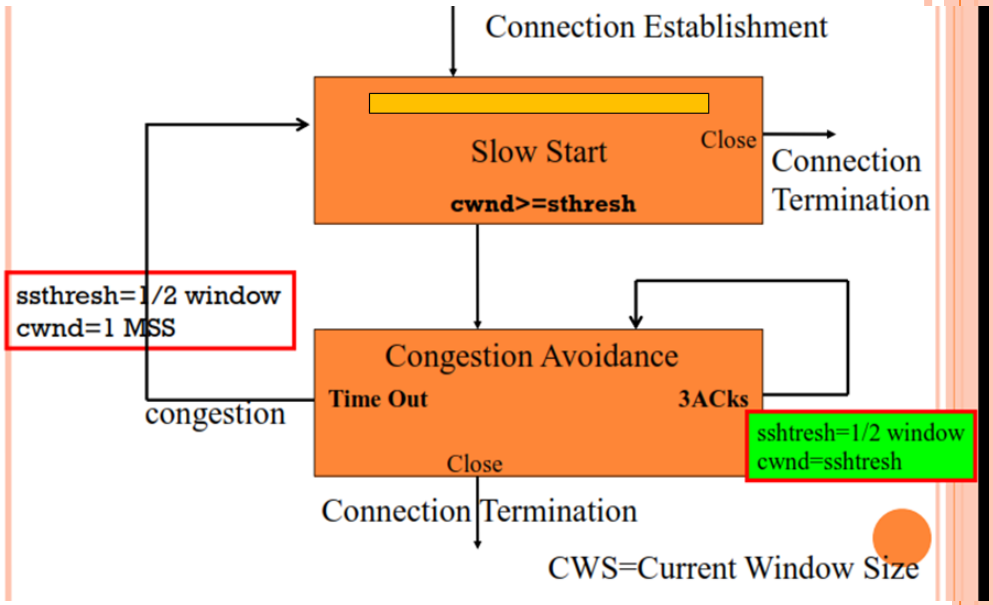
Note

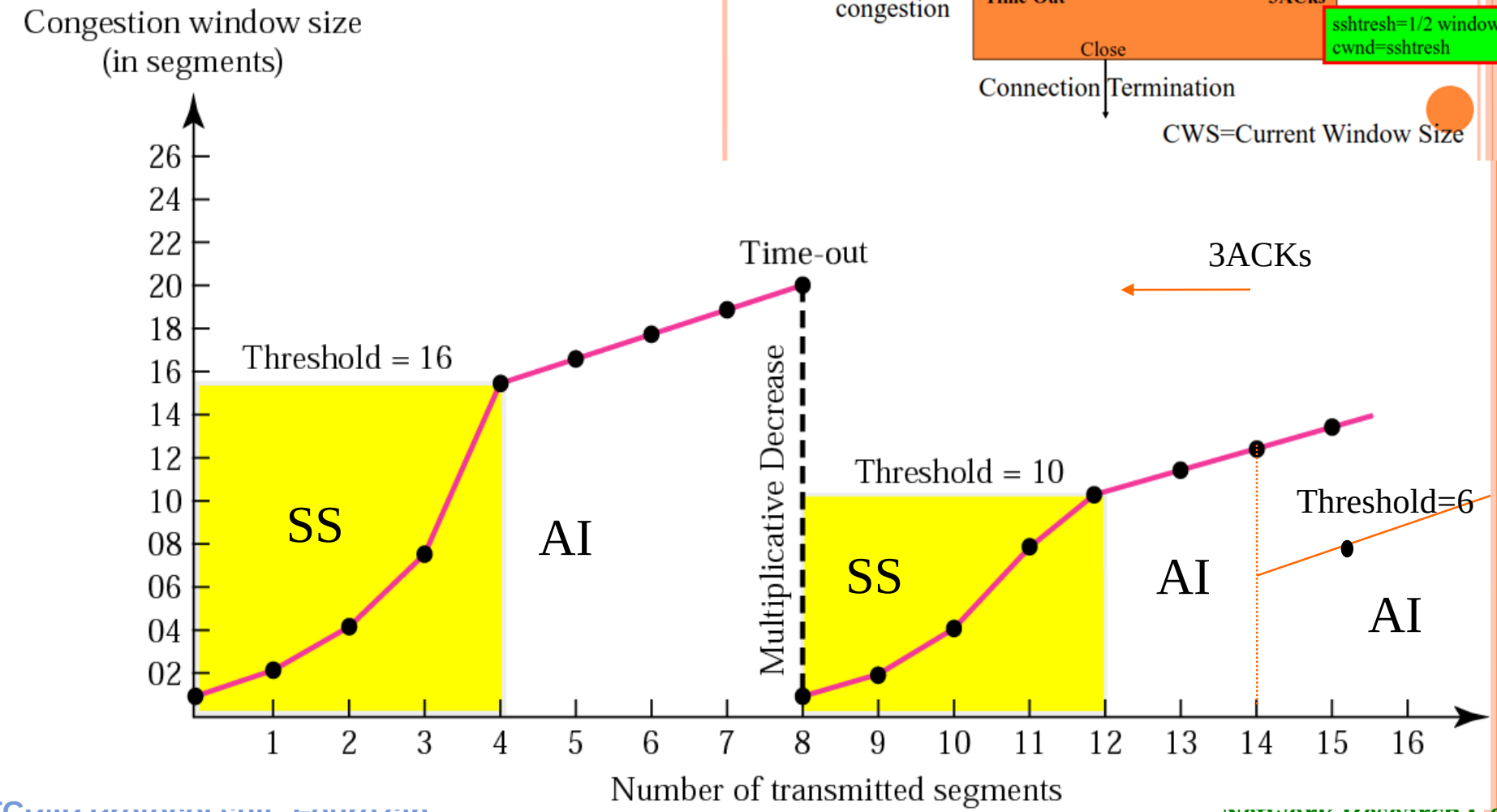
If the cause of the lost segment is congestion, retransmission of the segment not only does not remove the cause, it aggravates it.

Case Study

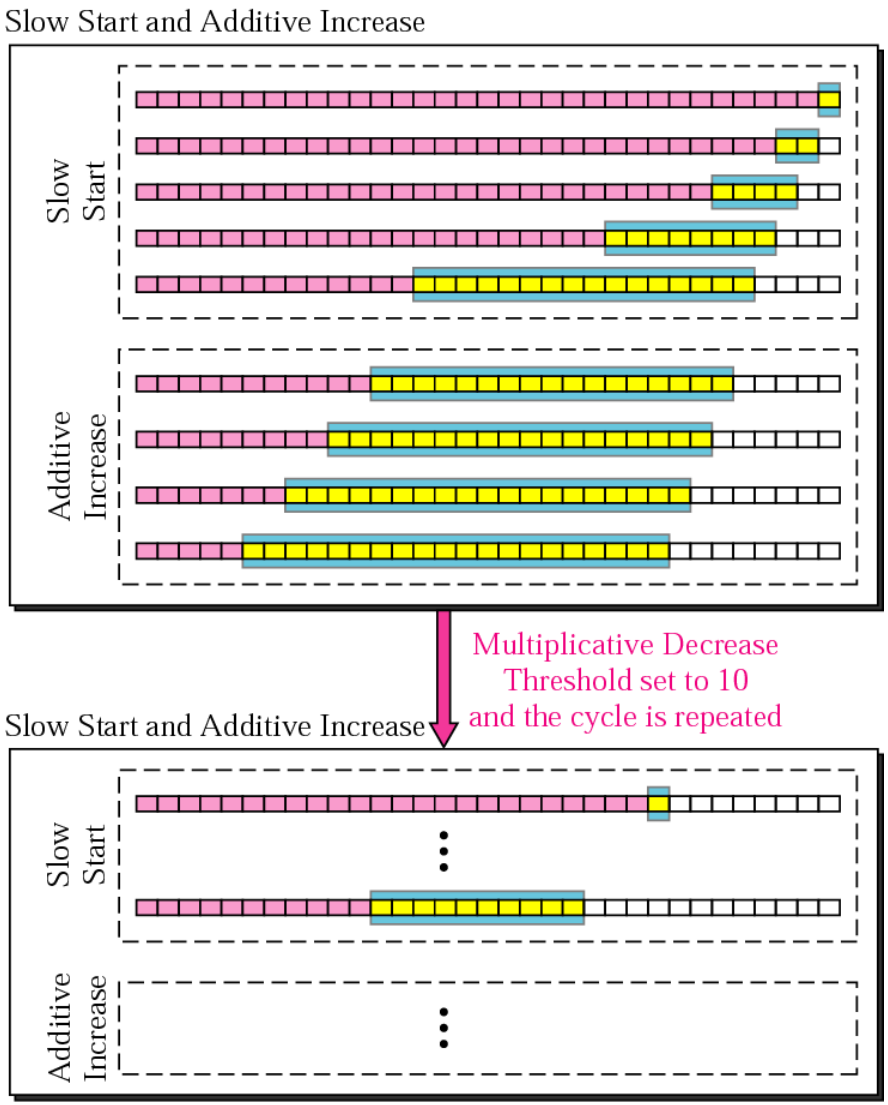
Maximum window size = 32 segments
Time out occurs when window size is 20.
When window size =12 , three Acks event occurred

Multiplicative decrease

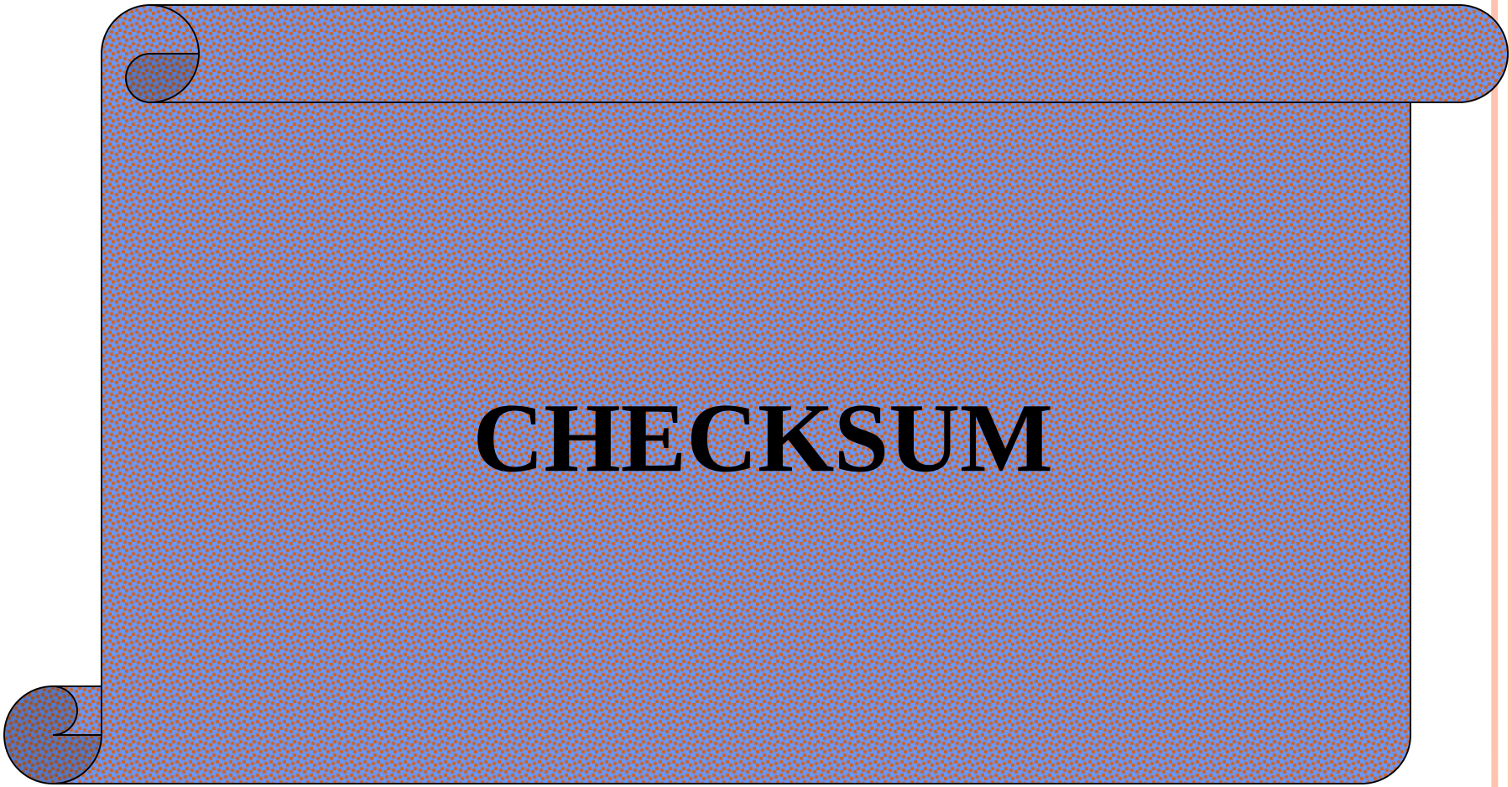




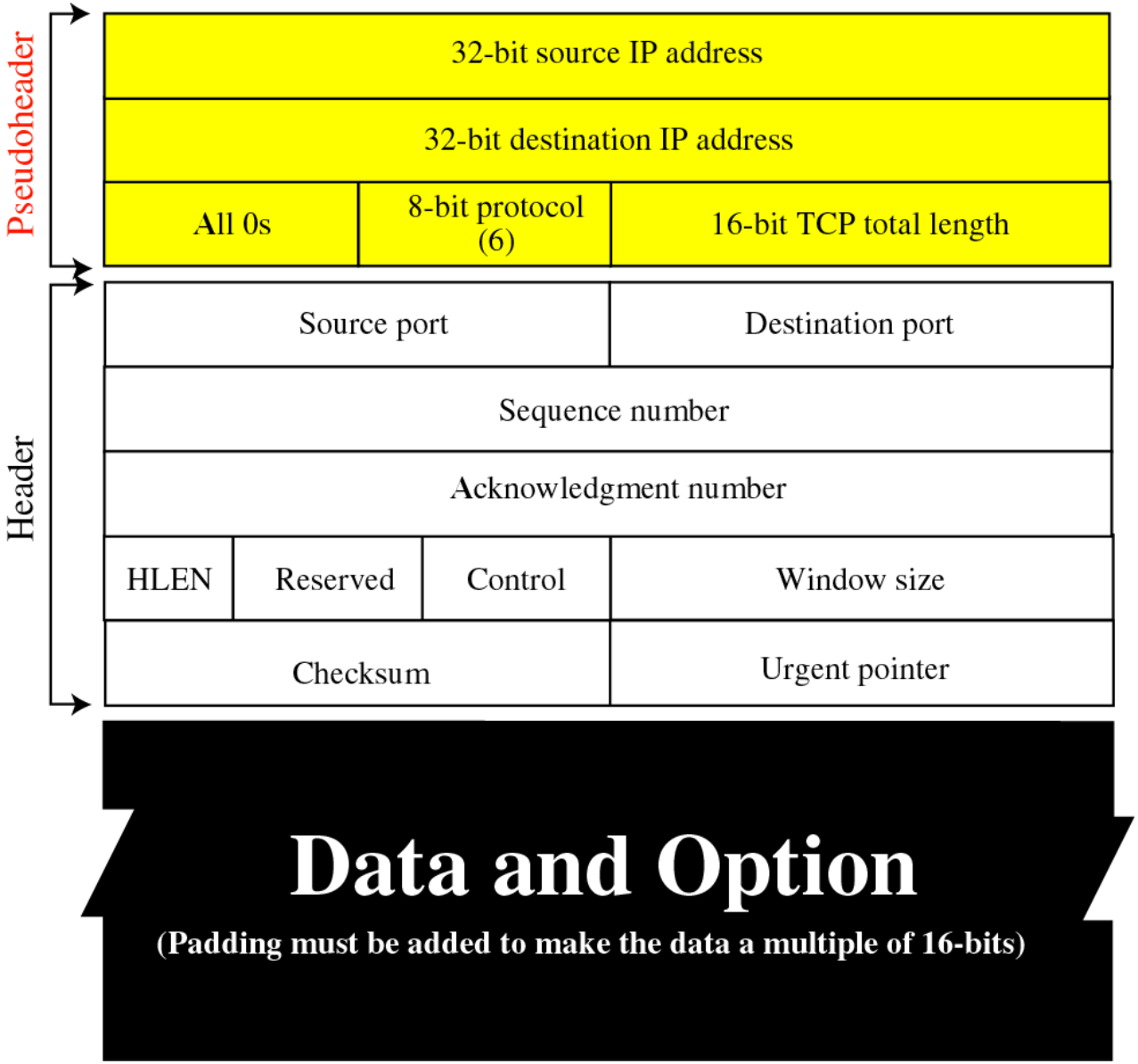
Congestion avoidance strategies



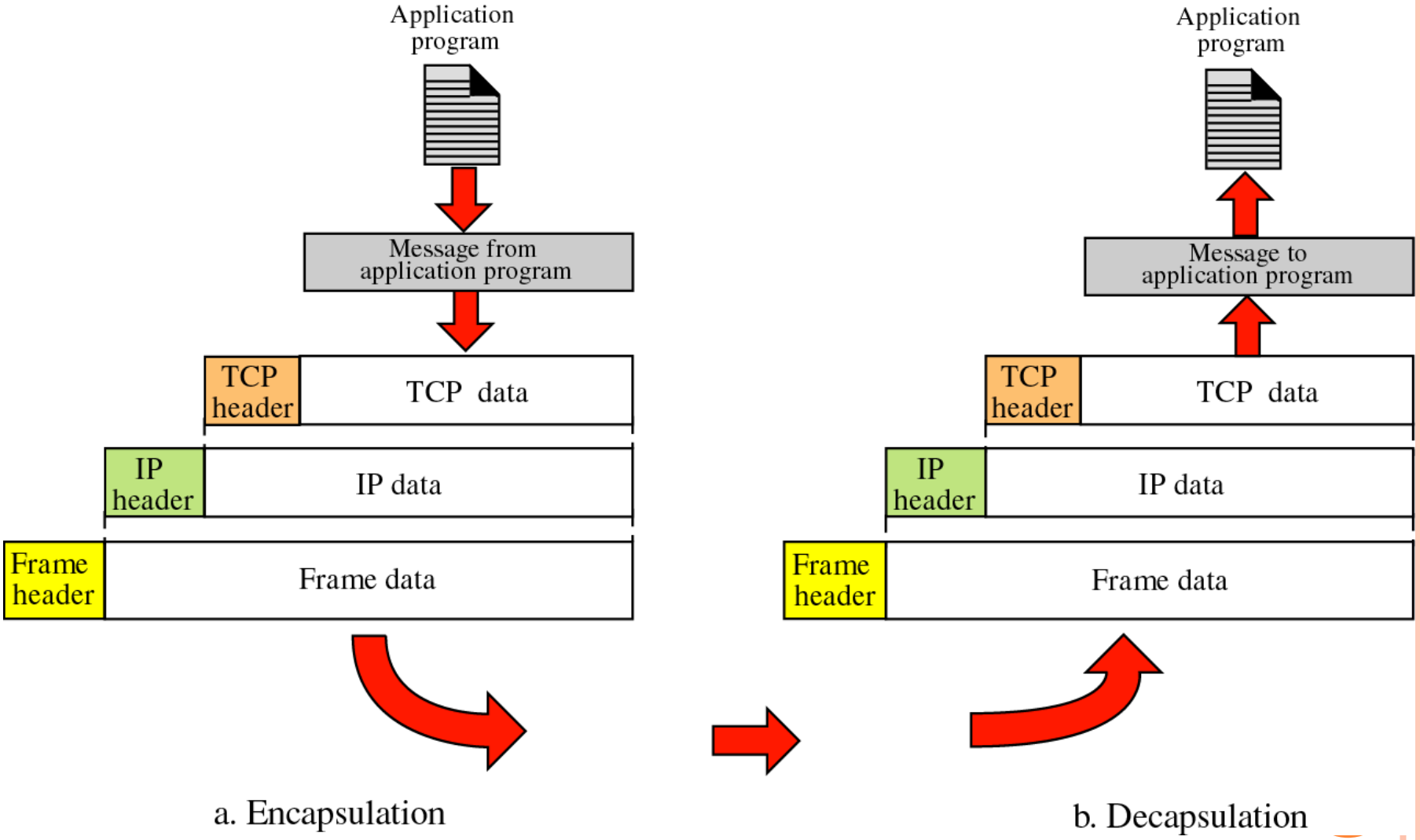
CHECKSUM



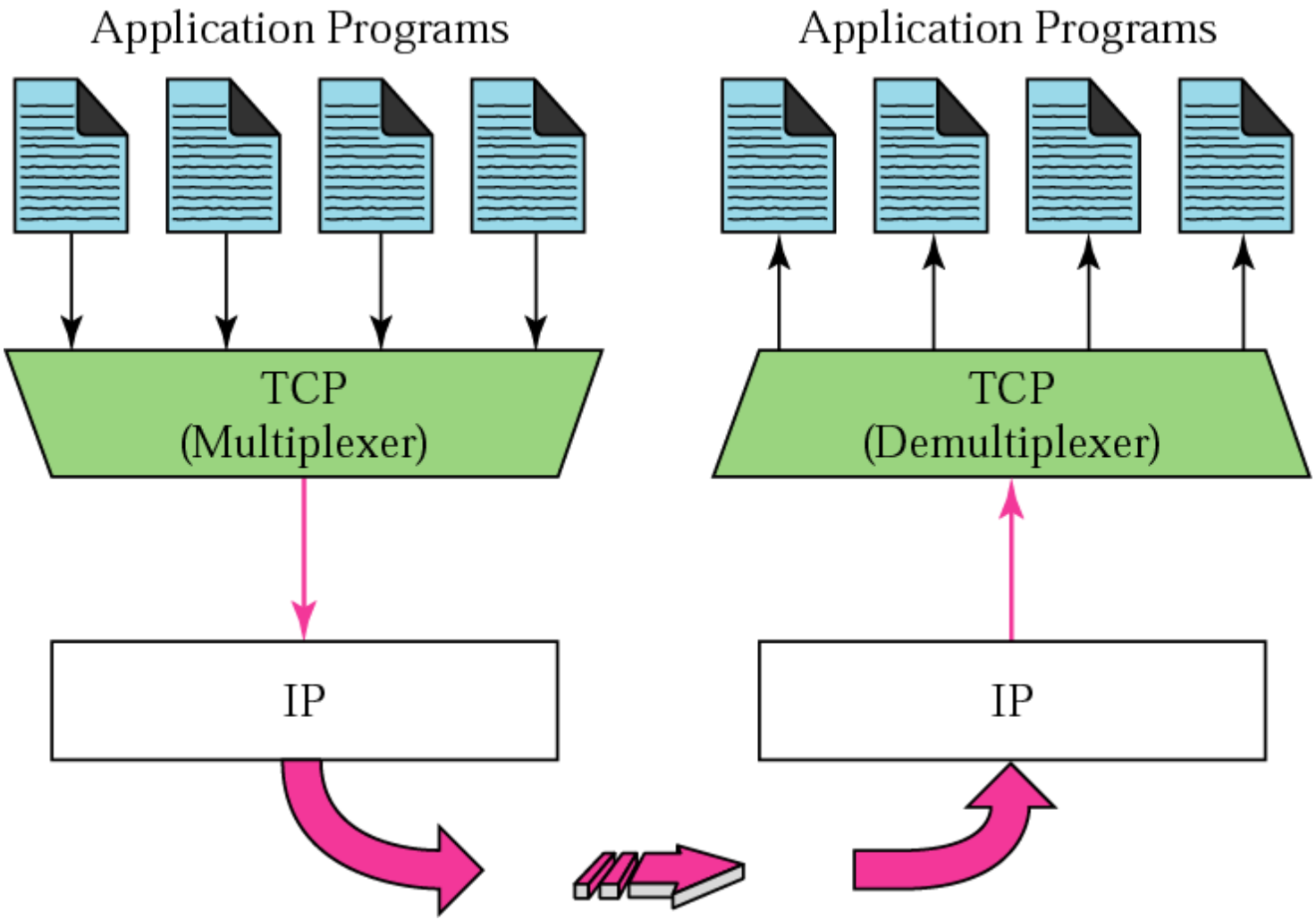
Pseudoheader added to the TCP datagram



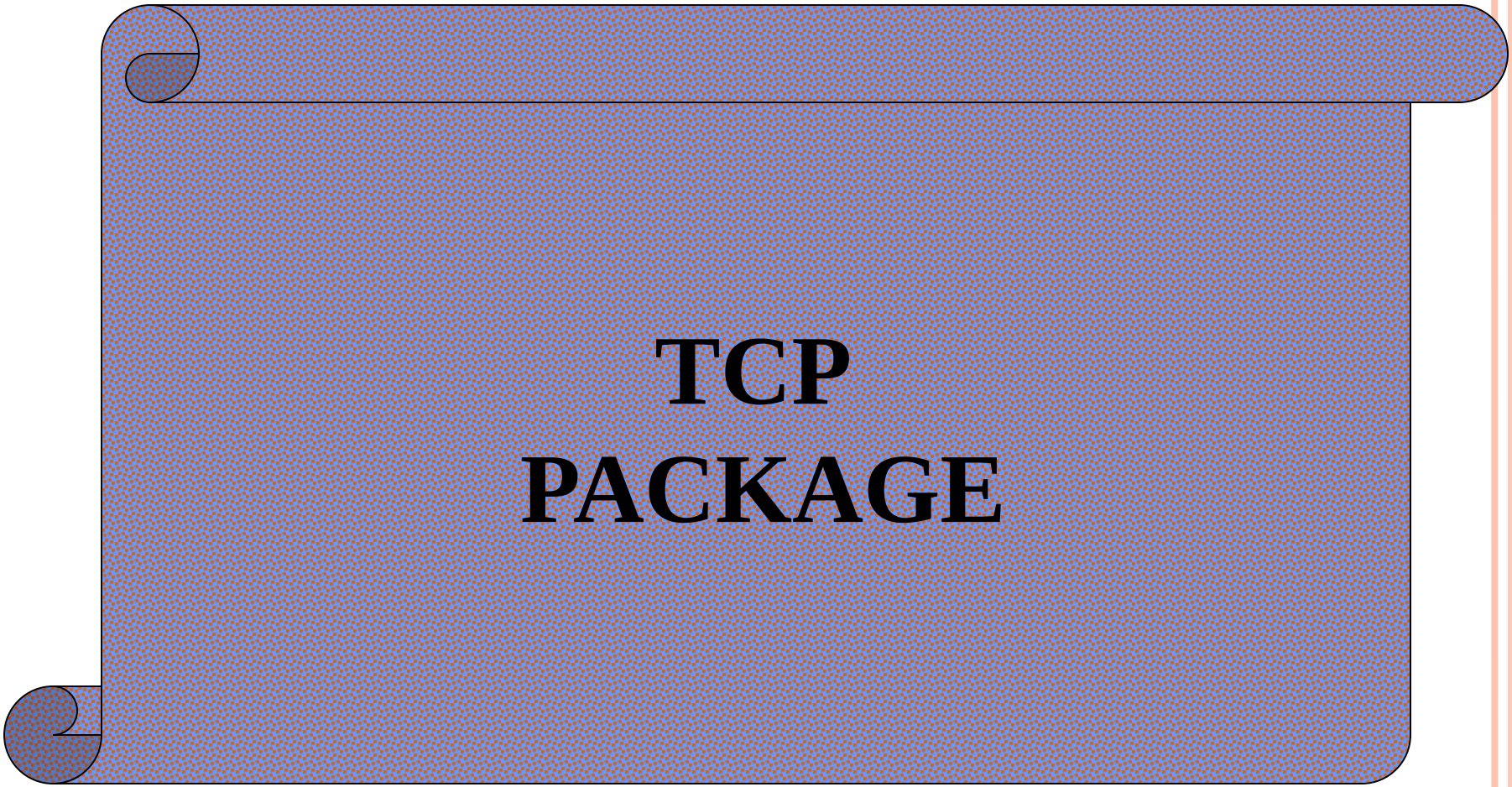
Encapsulation and decapsulation



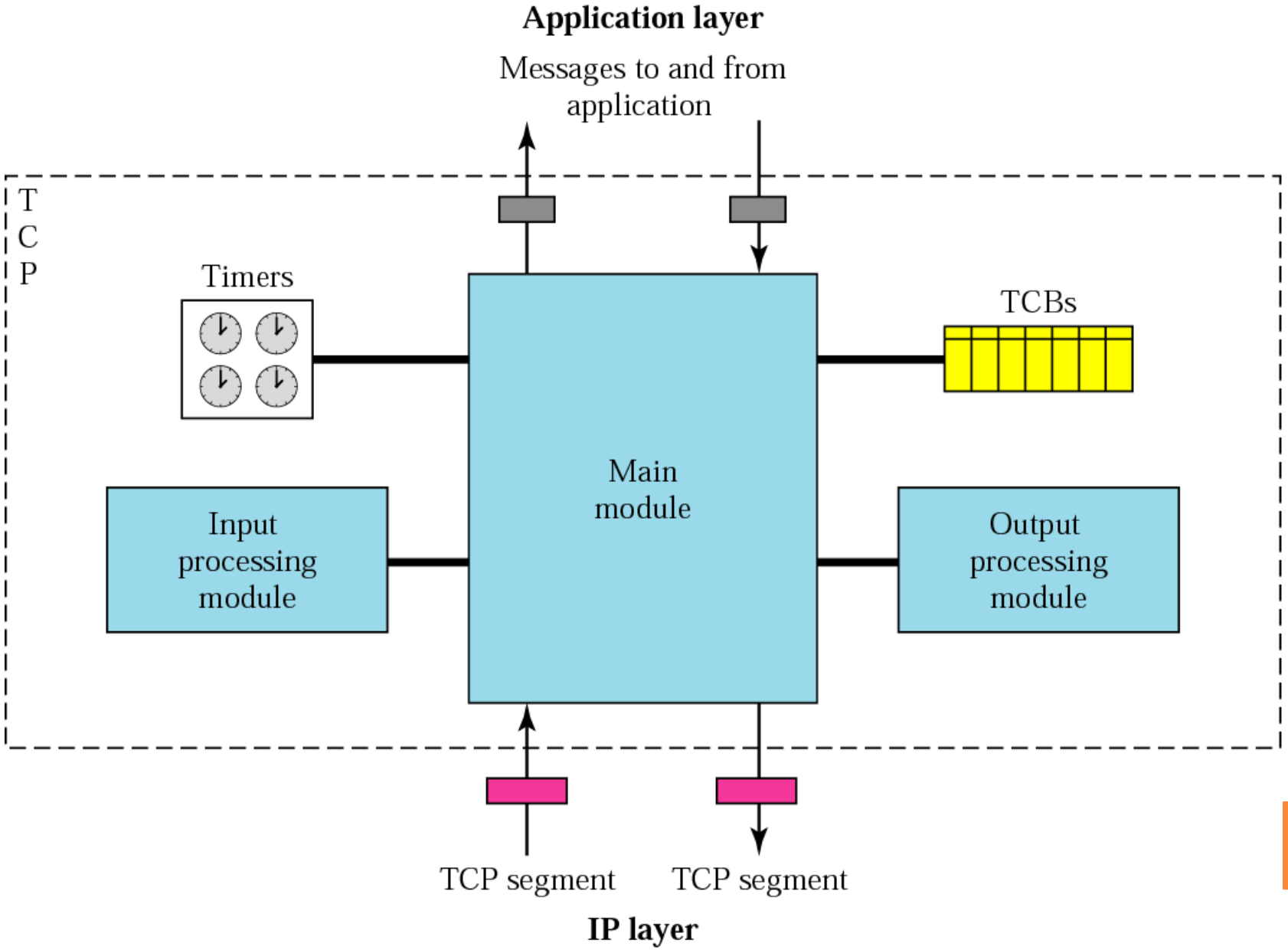
Multiplexing and demultiplexing



TCP PACKAGE



TCP package


Application layer
Messages to and from application
T
C
P
Timers
TCBs
Main module
Input processing module
Output processing module
TCP segment
TCP segment
IP layer

TCBs

Transmission control blocks