Computer Networks
Lecture 8

Transmission Control Protocol (TCP)

Introduction

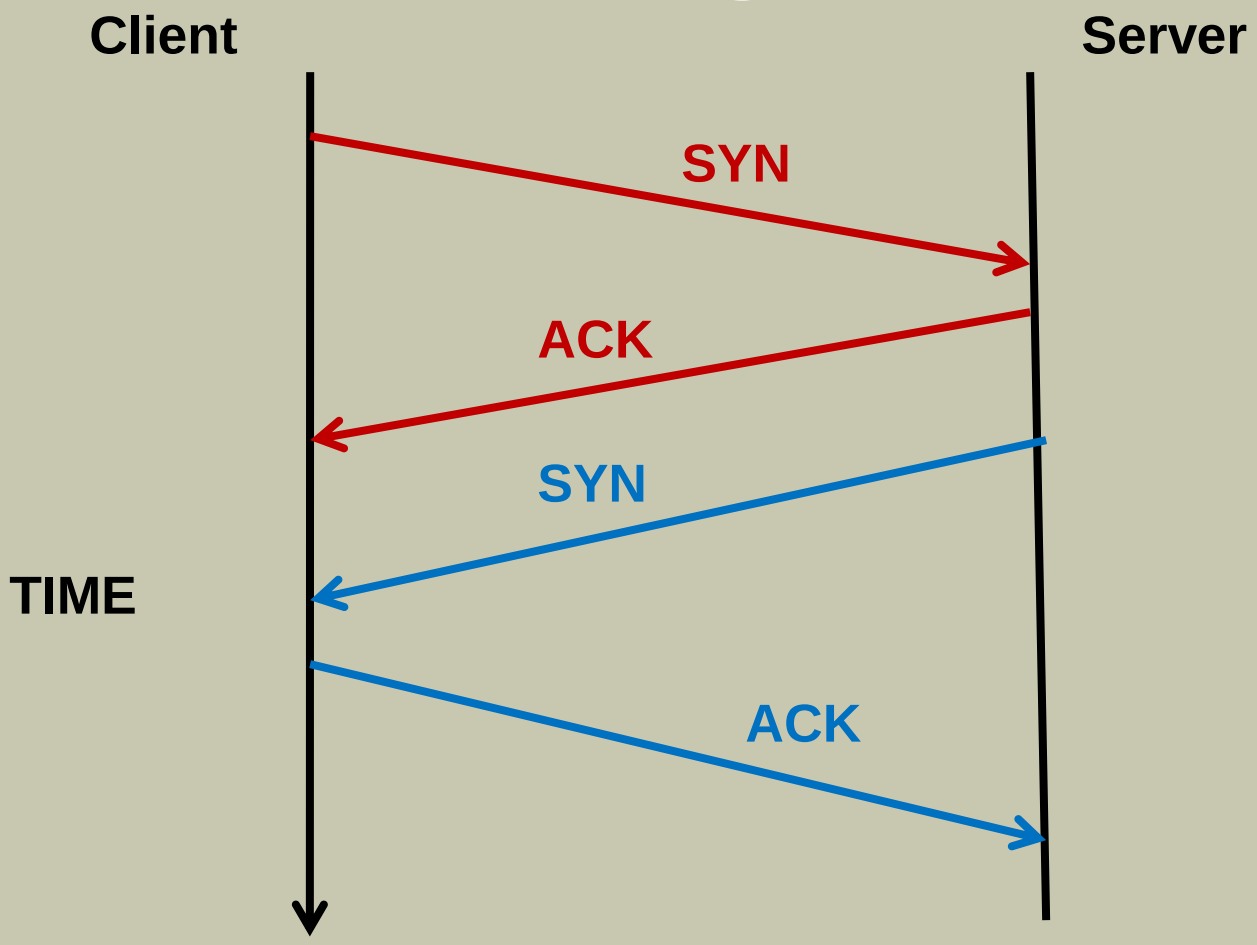
- TCP is one of the transport layer protocols of TCP/IP
- Error control, flow control and congestion control exist
- TCP can be categorized as a reliable protocol
- Combination TCP/IP is reliable

TCP Connection Process

- TCP-Client and TCP-Server connection process has three phases

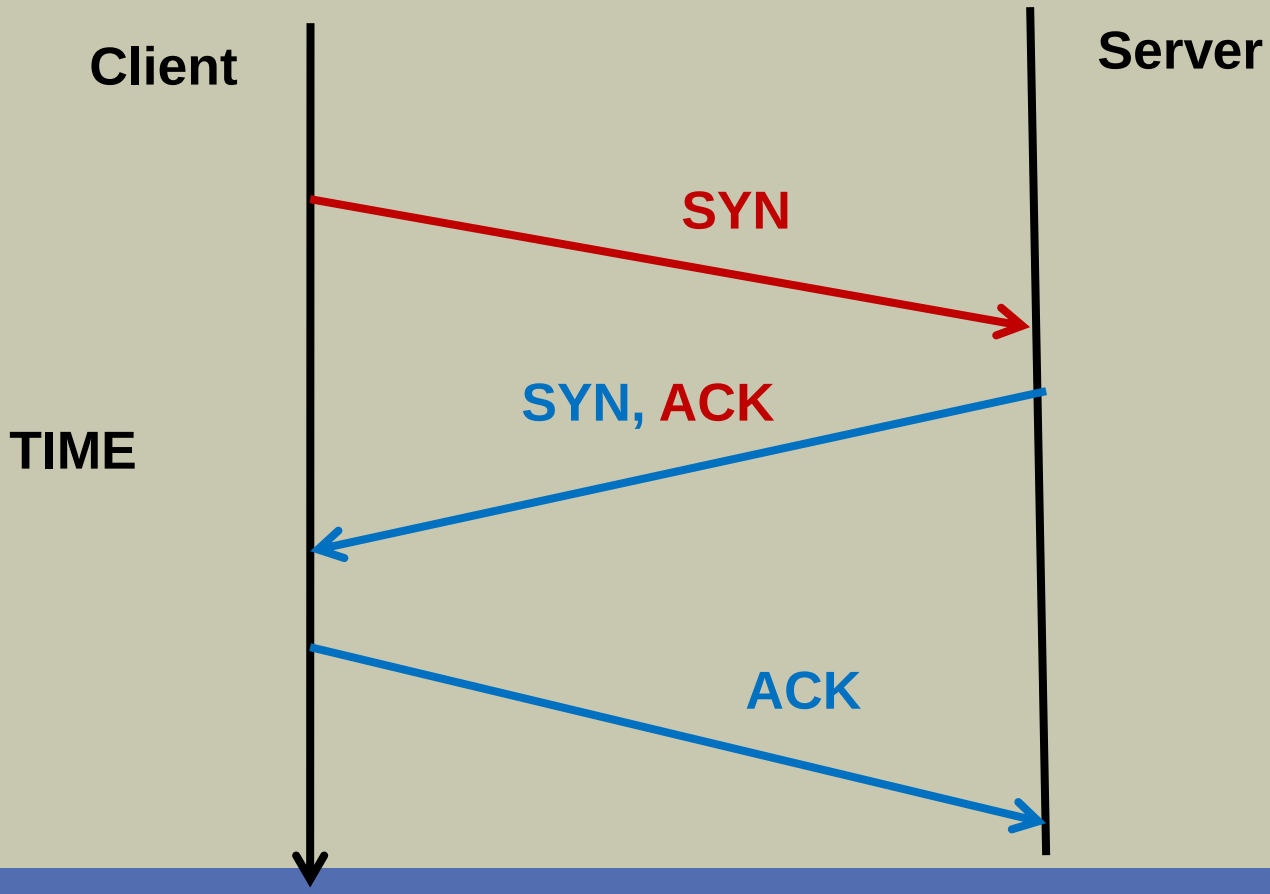
  - Connection Establishment
  - Data Transferring
  - Connection Termination

Connection Establishment



Three way handshake

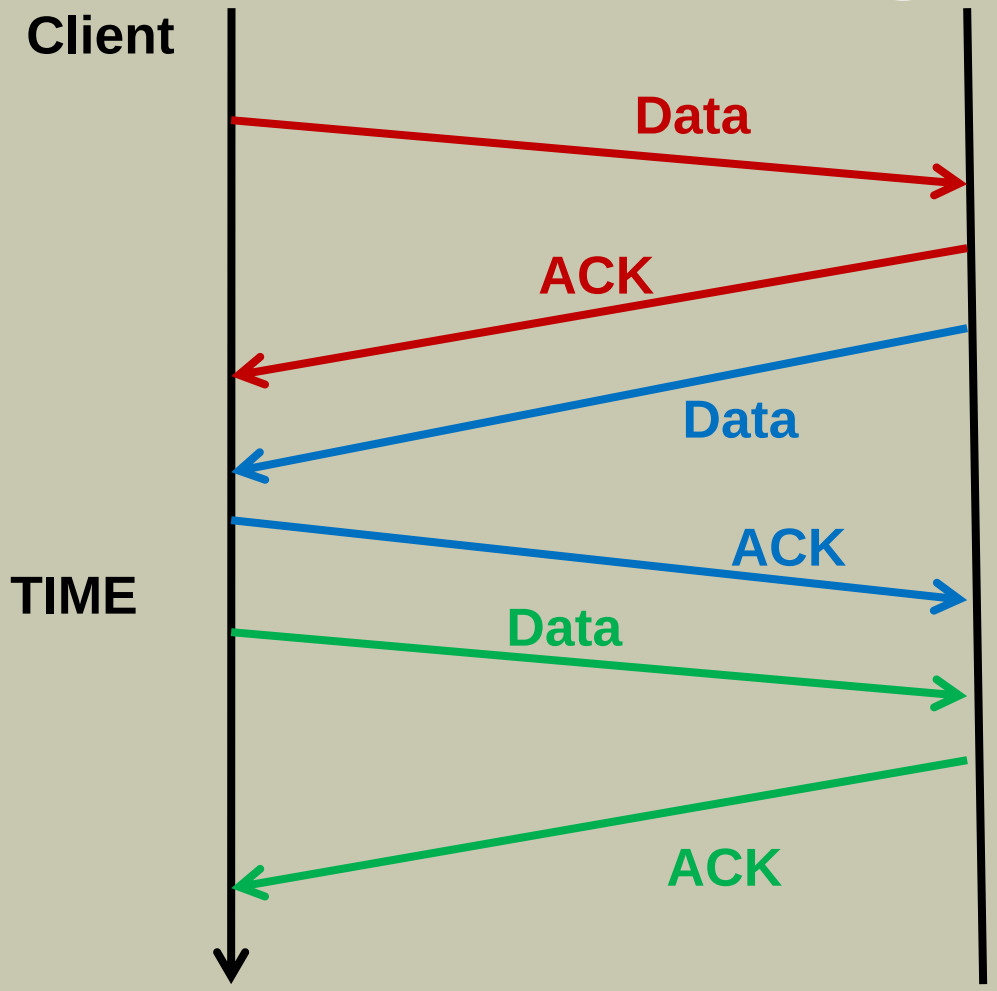
- Four steps can be reduced to three steps



Three way handshake cont.

- The SYN, SYN/ACK and ACK are 1-byte messages
- After the connection is established, data can be transmitted in full duplex mode between a client and a server

Data Transfer

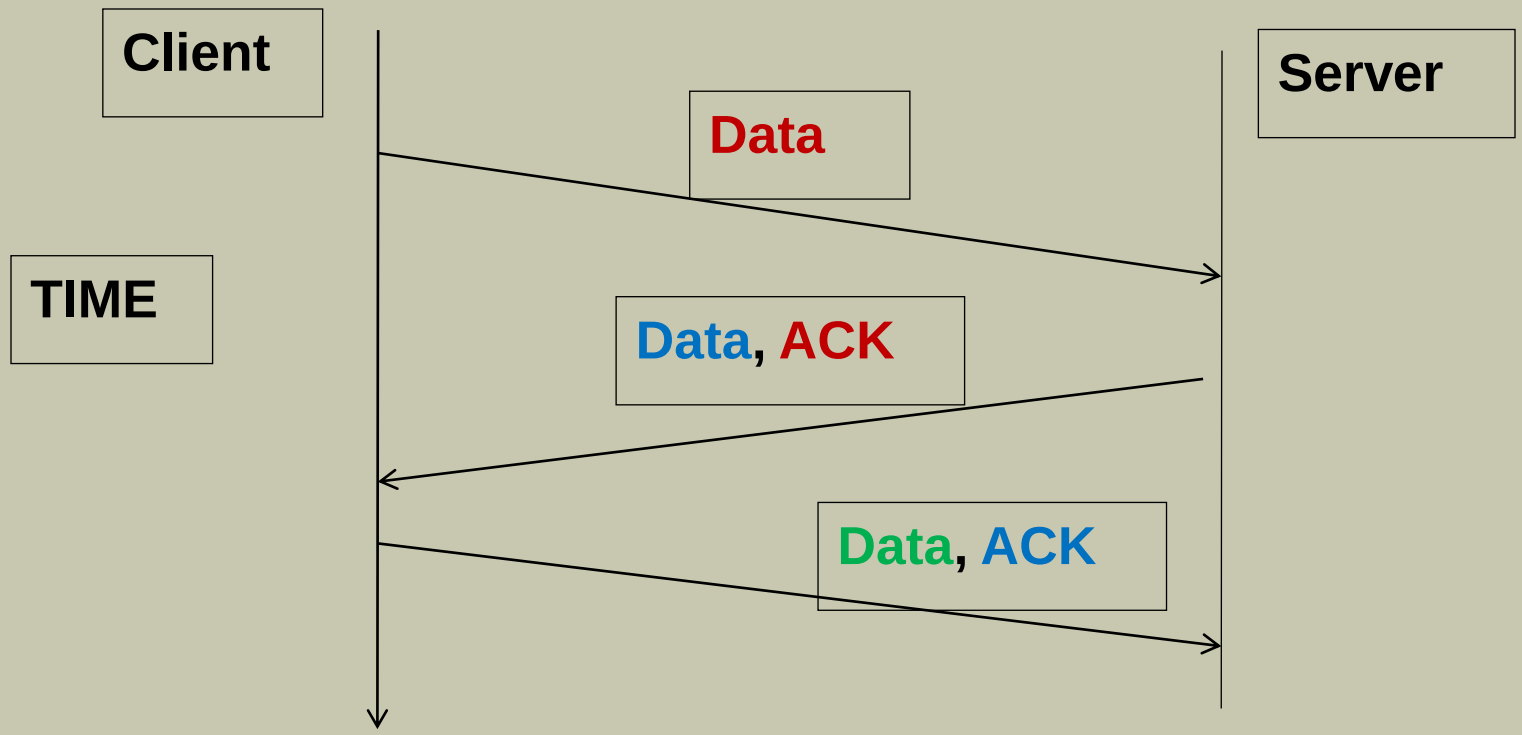


- TCP is a reliable protocol.
- TCP sends an Acknowledgement (ACK) for each segment of received data.

Data Transfer cont.

- **Piggybacking**

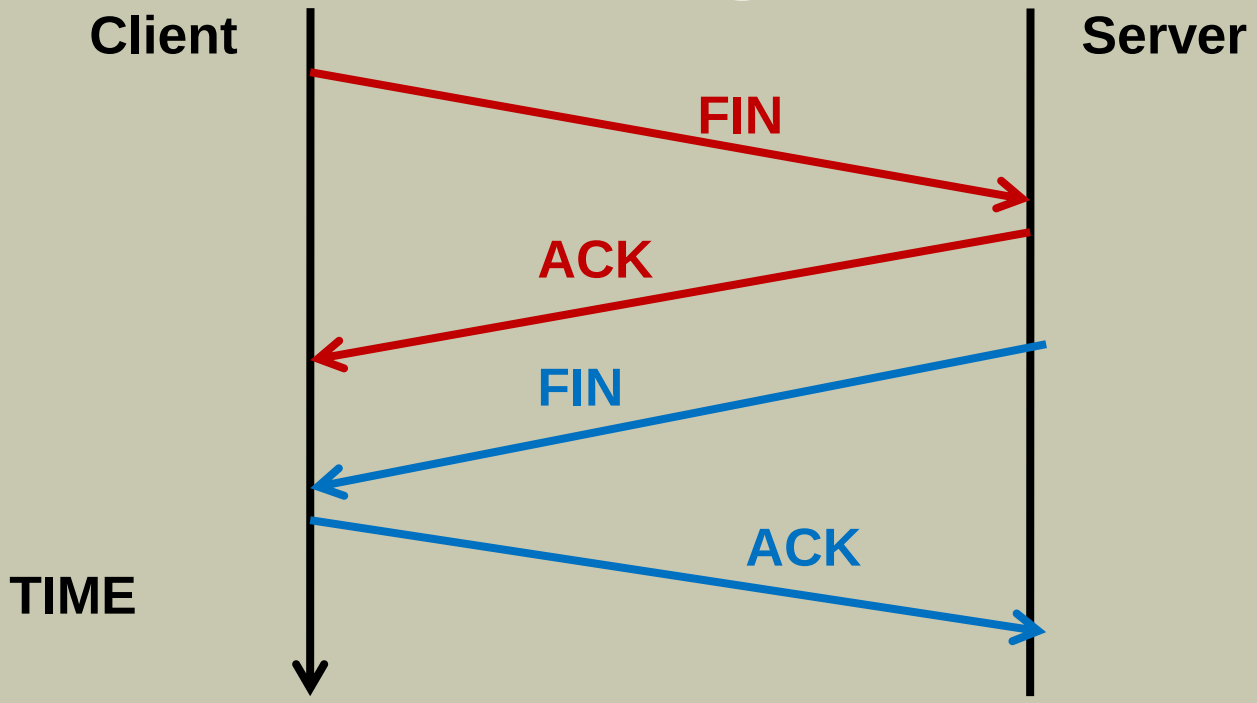
  Sending Data and ACK together.



Data Transfer cont.

- Data is transferred as **Segments**
- Each segment is given an identification called a **Sequence Number**.
- Acknowledgement Number =
  Sequence Number (Received) + Number of bytes in the segment

Terminating the connection



- Normally the connection termination request is initiated by the client
- FIN and ACK are considered one-byte messages

Problems related to data transfer

- The following three problems must be addressed in data transferring process for a better data transmission

  - Error control
  - Flow control
  - Congestion control

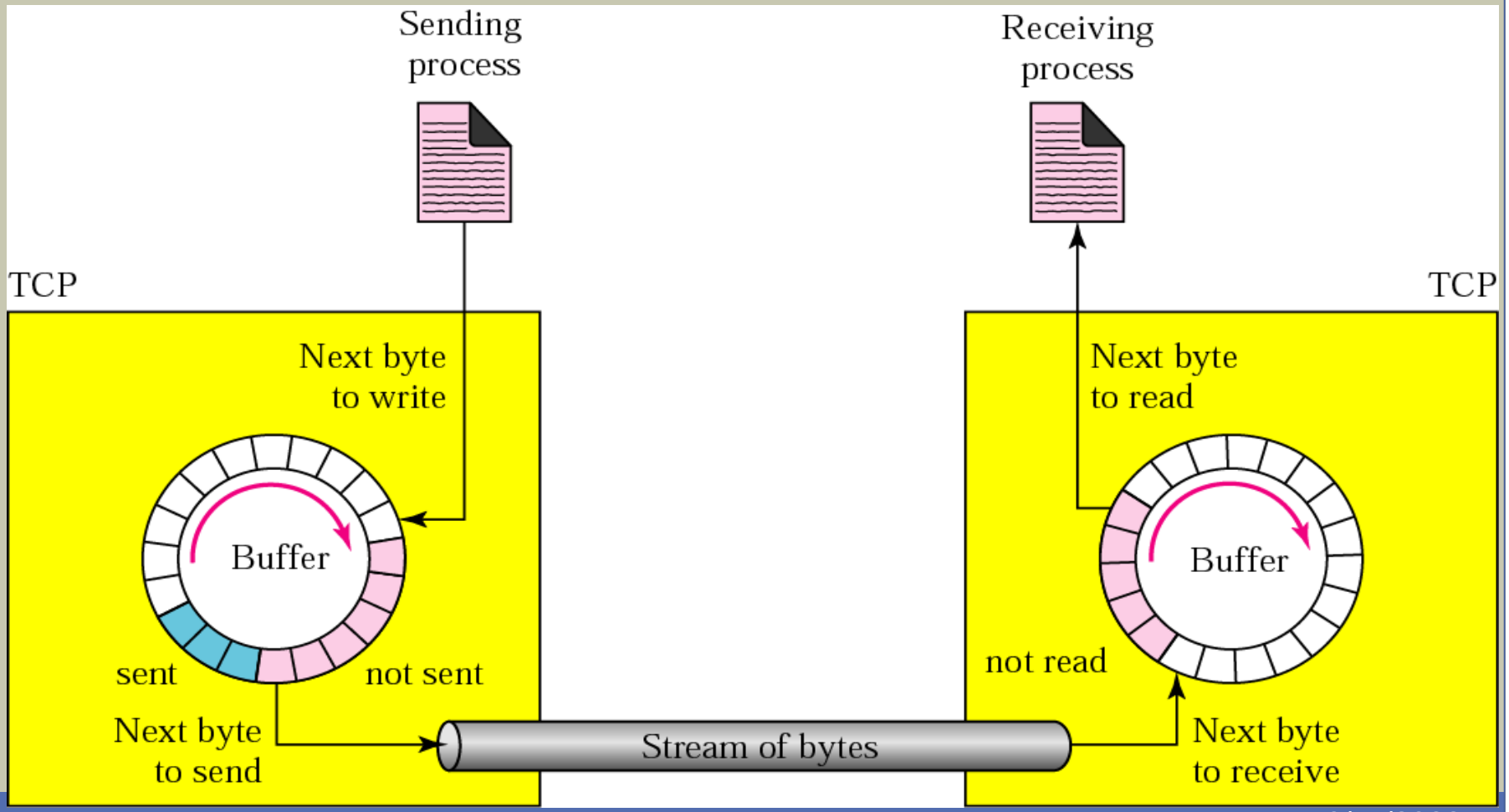
Error Control

- TCP receiver uses checksum bits for error detection
- If there are no errors it sends an acknowledgement to the sender
- If errors are found, the receiver does not send any negative acknowledgement to the sender

Error Control cont.

- In this process, the sender and receiver should have the following facilities
  - Sender buffer
  - Receiver buffer
  - Timer
- Sender buffer needs to keep the segments, until it receives an acknowledgement from the receiver
- Receiver buffer needs to keep the segment, until the error checking is over
- The sender needs a timer to check whether the Retransmission Time is expired after sending the segment

Flow control



Slow Applications

- If the application is very slow and it sends 1 byte at a time
- Application layer send data to the Tx buffer of the TCP layer
- The TCP protocol adds a 20byte TCP header and sends to IP layer
- The IP layer adds 20 bytes of IP header and sends data link layer
- Data link layer also adds a header and a trailer (say 26 bytes)
- Altogether 20+20+26 = 66 byte overhead is added to application data

Slow Applications cont.

- 1 byte of data is combined with 66 byte overhead bits
- This is a very inefficient data transfer
- TCP has the facility to improve such a situation by defining the minimum data required for a segment
- TCP waits until such an amount of data is collected
- After that the TCP segment is sent to the IP layer

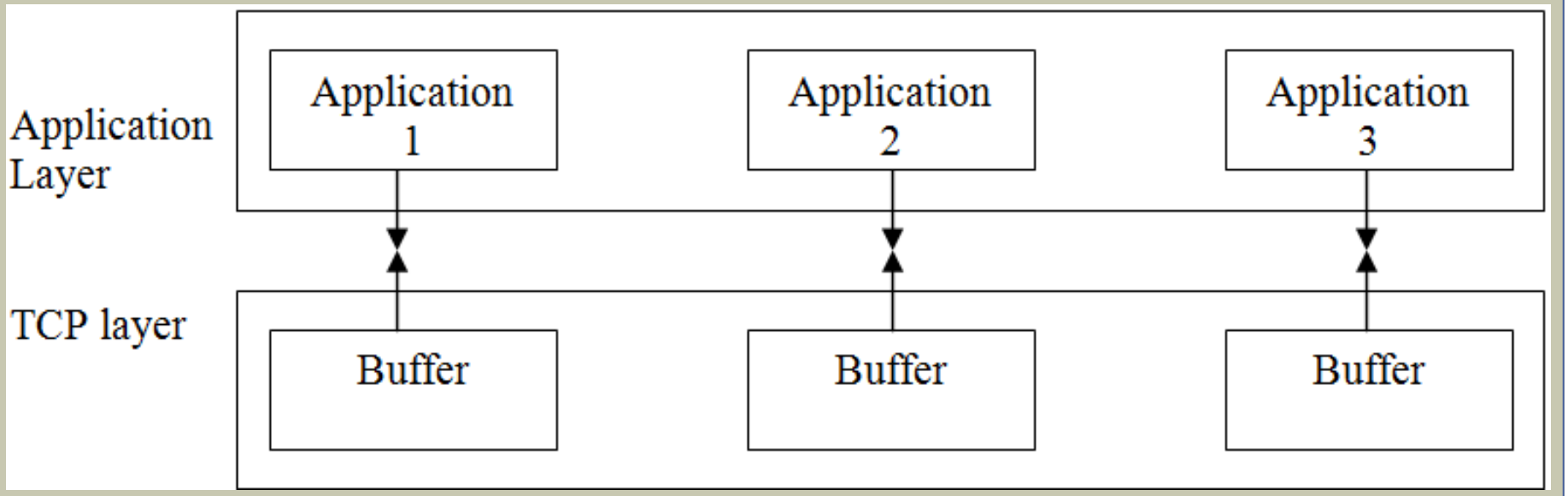
Fast application/ Slow network/ Slow receiver

- Application layer buffers and TCP layer buffer sizes are decided on the following factors
  - Speed of application
  - Speed of computer
  - Speed of the network
  - Congestion of network
- If the sender application is faster than the network, it will overflow the TCP buffer and gives a **" runtime error" – PCs Stuck**

Needs to Restart

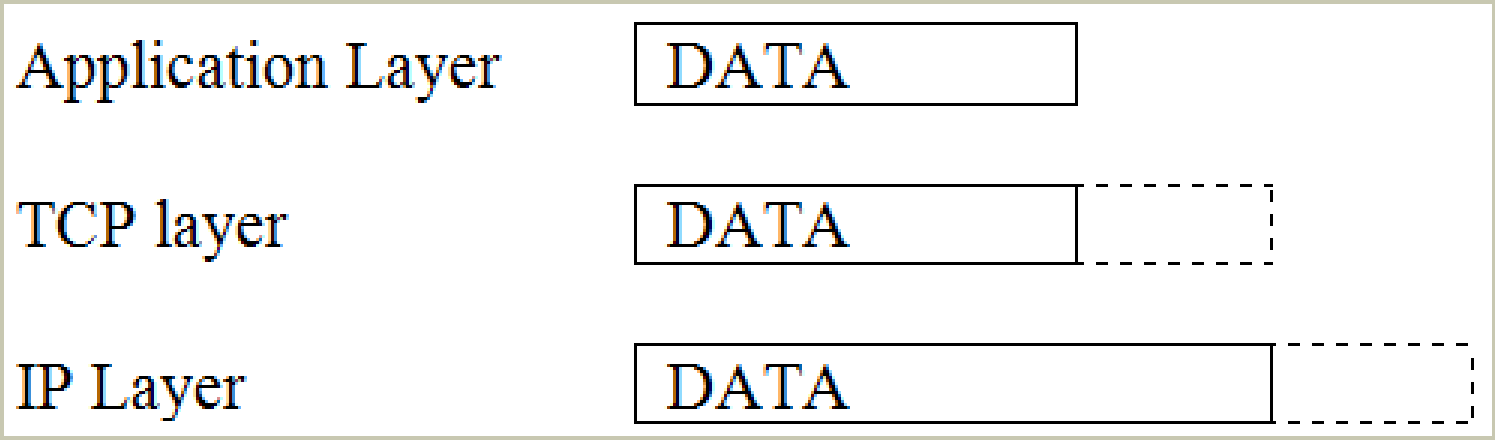
Multiplexing

- TCP layer can handle several application processes at the same time
  - - E-mail, Web browsing , Transfer files
- Port Number identifies the application



TCP with IP layer

- IP layer receives data from the transport layer
- Therefore the type of data in IP packet can be TCP, UDP.
- Each type of data is given a unique protocol number



Port number

- The port number is a 16 bit binary number in the TCP
- It is in the range of 0-65535
- The port numbers are divided into three ranges.
  - Well known ports
  - Registered ports
  - Dynamic ports/ Ephemeral ports

Well-known ports

- Port numbers ranging from 0-1023 is called well-known ports

Assigned for Standard Server Processes

Port	*Protocol*	*Description*
7	Echo	Echoes a received datagram back to the sender
9	Discard	Discards any datagram that is received
11	Users	Active users
13	Daytime	Returns the date and the time
17	Quote	Returns a quote of the day
19	Chargen	Returns a string of characters
20	FTP, Data	File Transfer Protocol (data connection)
21	FTP, Control	File Transfer Protocol (control connection)
23	TELNET	Terminal Network
25	SMTP	Simple Mail Transfer Protocol
53	DNS	Domain Name Server
67	BOOTP	Bootstrap Protocol
79	Finger	Finger
80	HTTP	Hypertext Transfer Protocol
111	RPC	Remote Procedure Call

Registered ports

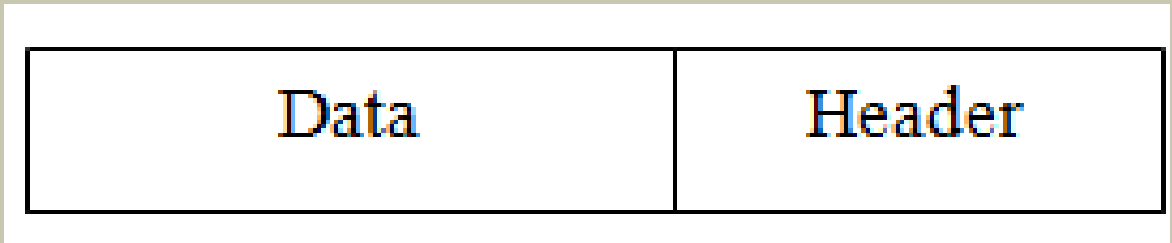
- The ports ranging from 1024 - 49,151 are to be registered with IANA to prevent duplicating
- Used for proprietary server processors or any client process
- Normally not used

Dynamic ports

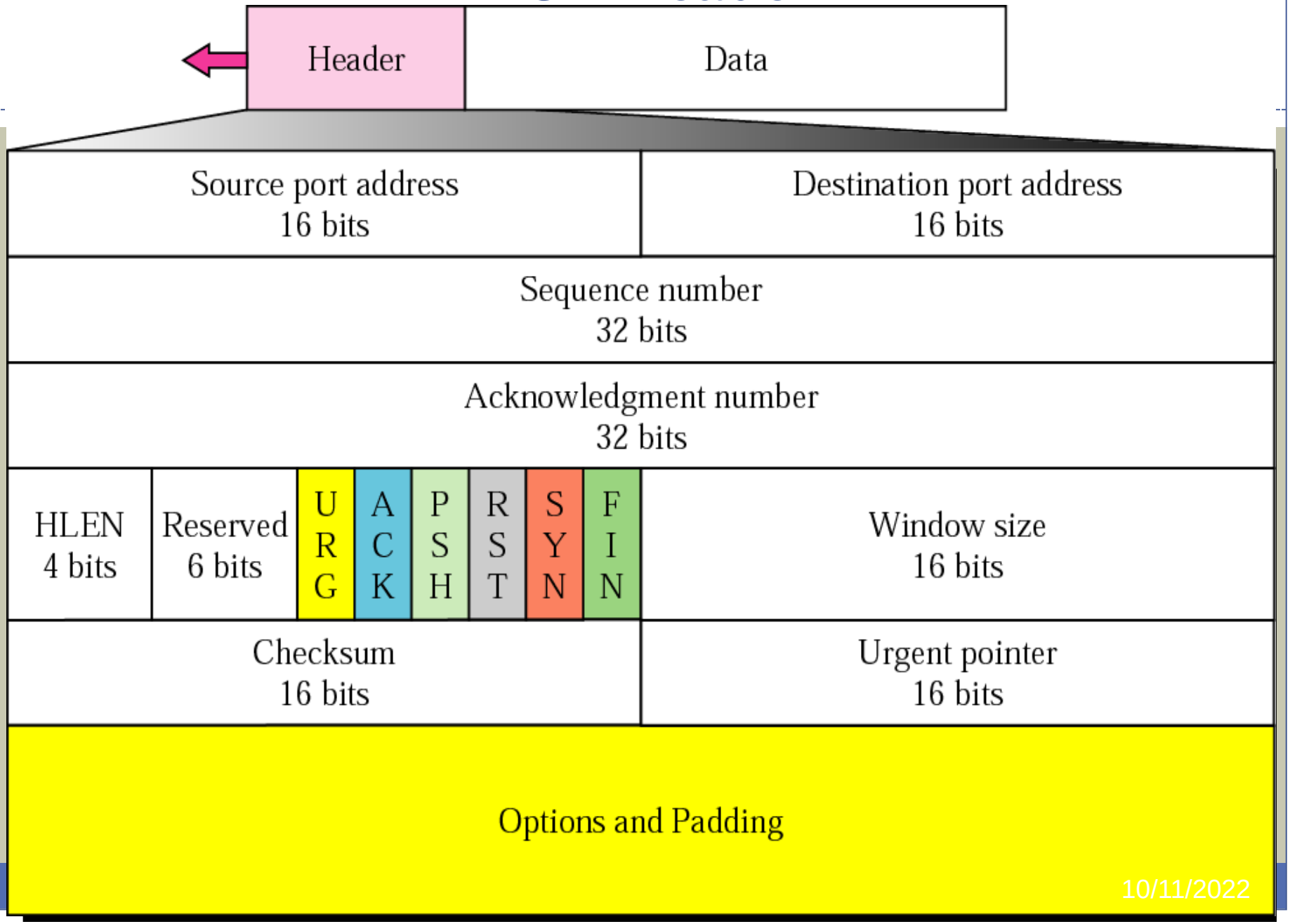
- The ports numbers from 49,152 to 65,535 are dynamic or ephemeral ports
- They are used by client processes temporarily

TCP segment

- TCP segment consists of data and TCP header
- The standard TCP header length is 20 bytes
- With the option fields it can be expanded up to 60 bytes



TCP Header


Header
Data
Source port address
16 bits
Destination port address
16 bits
Sequence number
32 bits
Acknowledgment number
32 bits
HLEN
4 bits
Reserved
6 bits
URG
ACK
PSH
RST
SYN
FIN
Window size
16 bits
Checksum
16 bits
Urgent pointer
16 bits
Options and Padding

TCP header cont.

Source port address

- A 16-bit field
- Port number range = 0 – 65535
- **For client TCP header this is a dynamic port number**
- **For server TCP header this is a well-known port number**

Destination port address

- Is the destination process port number
- A 16-bit field

Sequence number

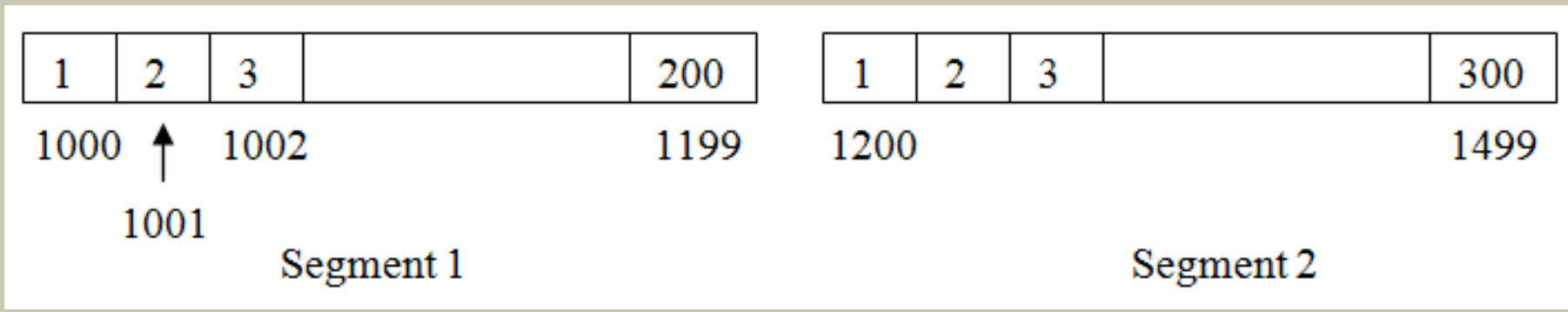
- A 32-bit field
- At the time of establishing TCP connection the first sequence number should be decided

Initial Sequence Number (ISN)

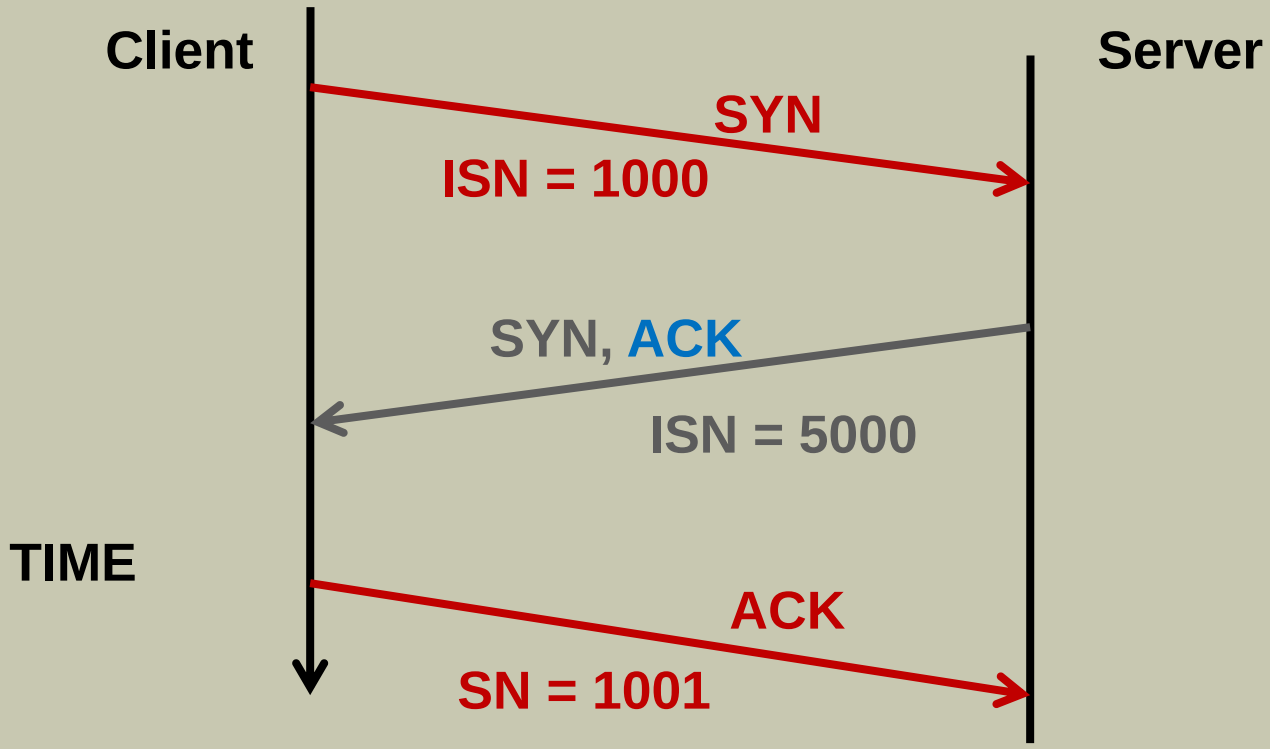
- It can be any arbitrarily number between 0 and 2^{32}-1.

Sequence number cont.

- The allocation of sequence number uses the following process
  - Suppose the first data segment has 200 bytes
  - Second data segment has 300 bytes
  - Sequence number of first segment (ISN) is 1000

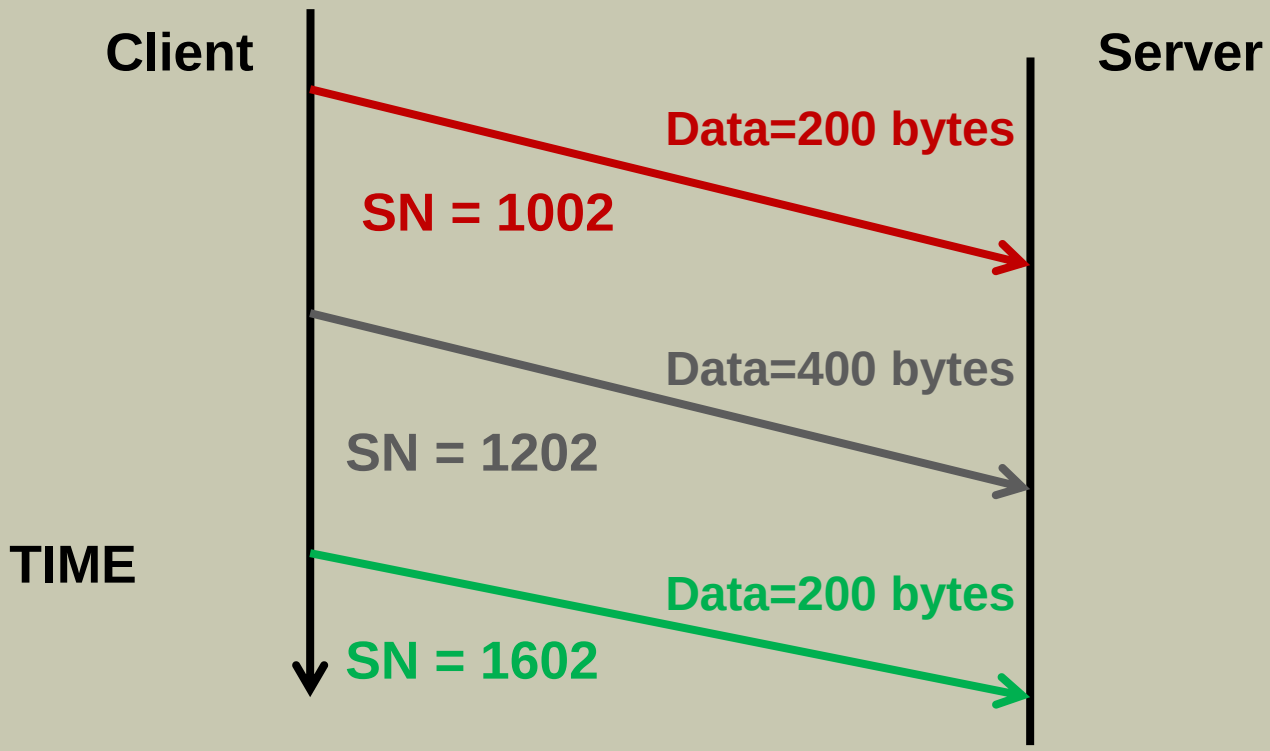


Sequence numbers of three-way handshake

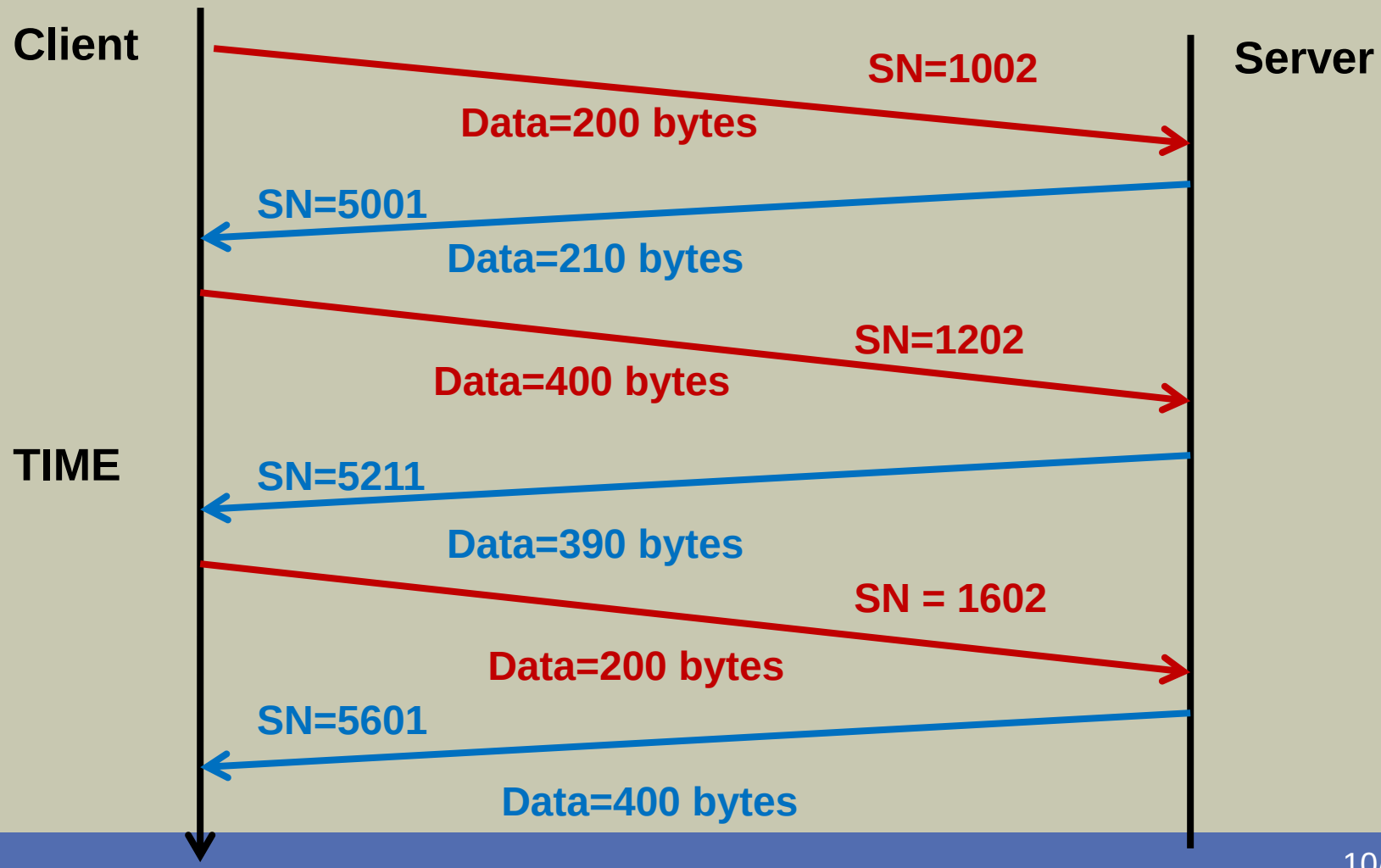


- SYN and ACK : 1 byte each

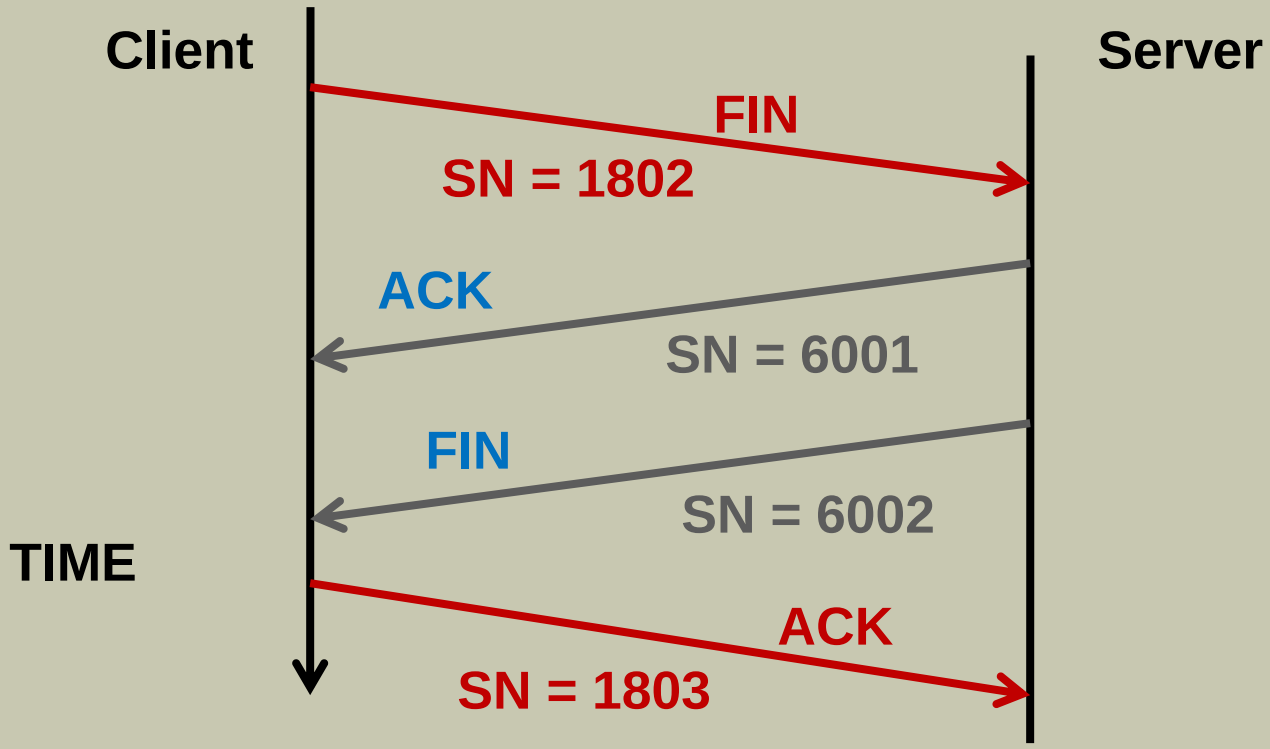
Sequence numbers of data segments



Sequence numbers of data segments cont.



Sequence numbers of connection termination



- FIN and ACK : 1 byte each

Acknowledgement Number

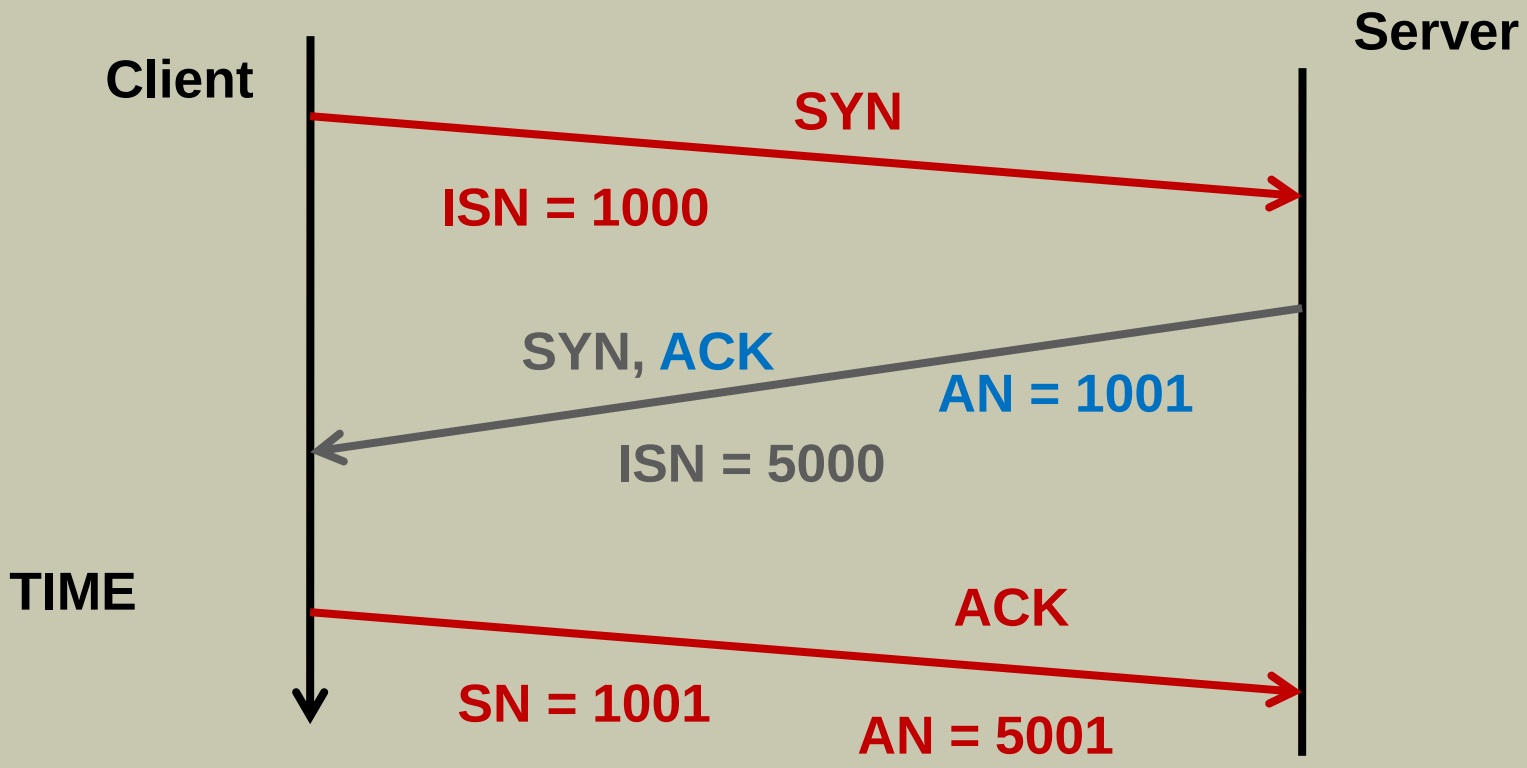
- A 32-bit field
- An acknowledgement number is sent by the receiver for each data segment it receives
- It is the next sequence number expected by the receiver

Acknowledgement Number cont.

- Relationship among sequence number, number of bytes in a segment and the acknowledgement number sent by the receiver

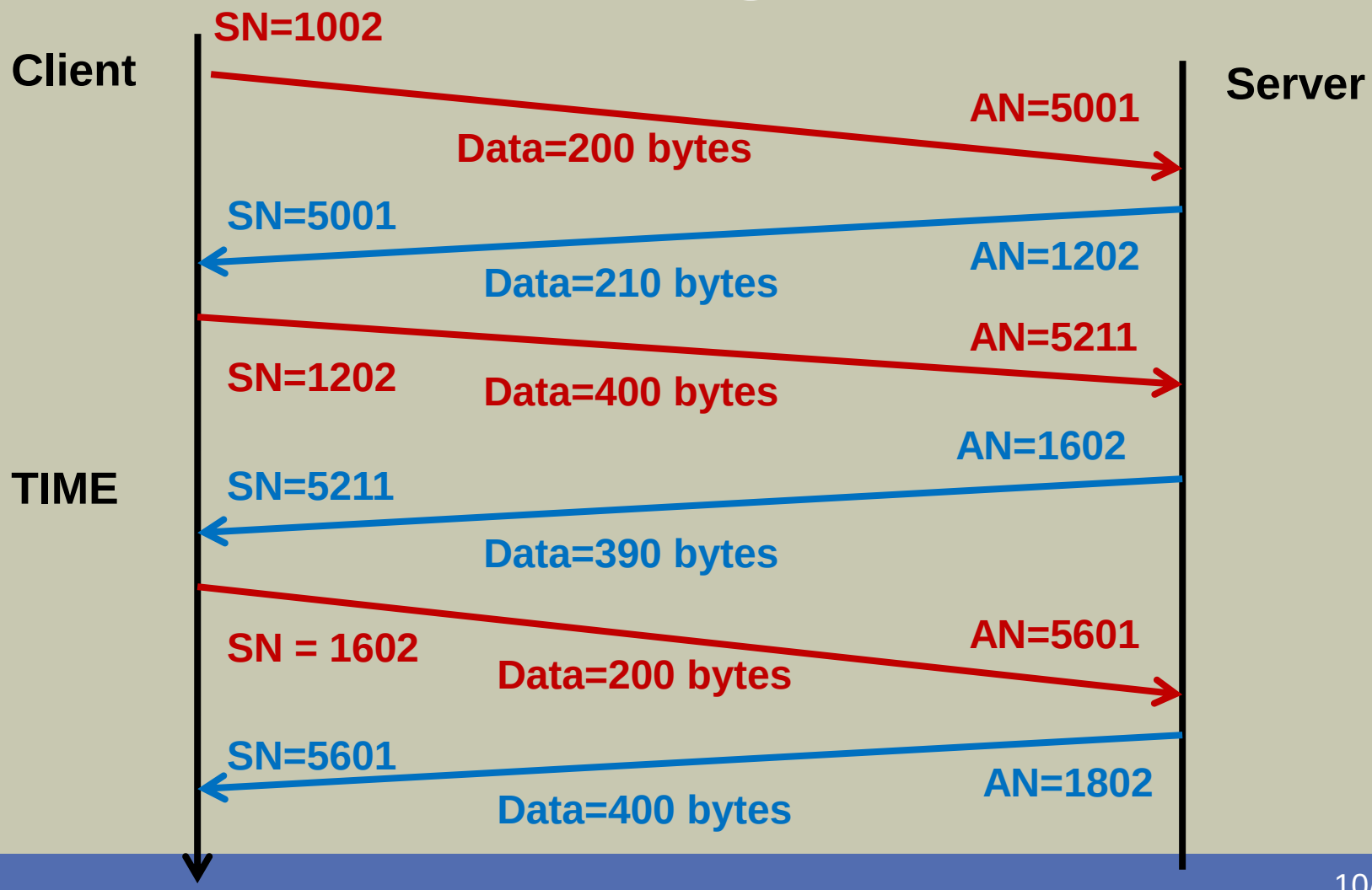
Sequence Number of segment	No. of bytes in the segment	Acknowledgement number send by receiver
1000 (SYN)	1	1001
1001 (ACK)	1	1002
1002	200	1202
1202	400	1602
1602	200	1802
1802 (FIN)	1	1803

Acknowledgement numbers of three-way handshake

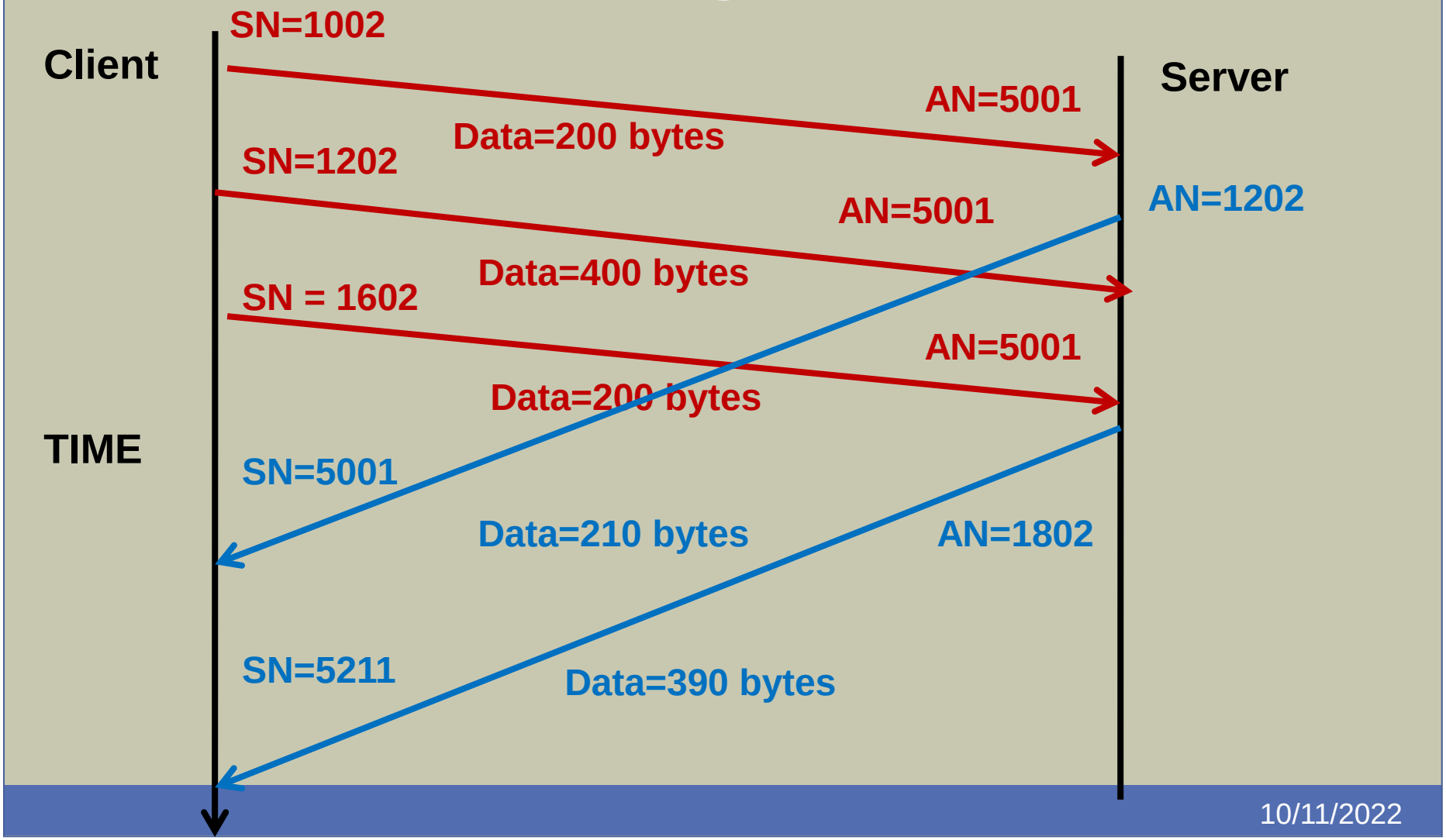


- SYN and ACK : 1 byte each

Acknowledgement numbers of data segments cont.



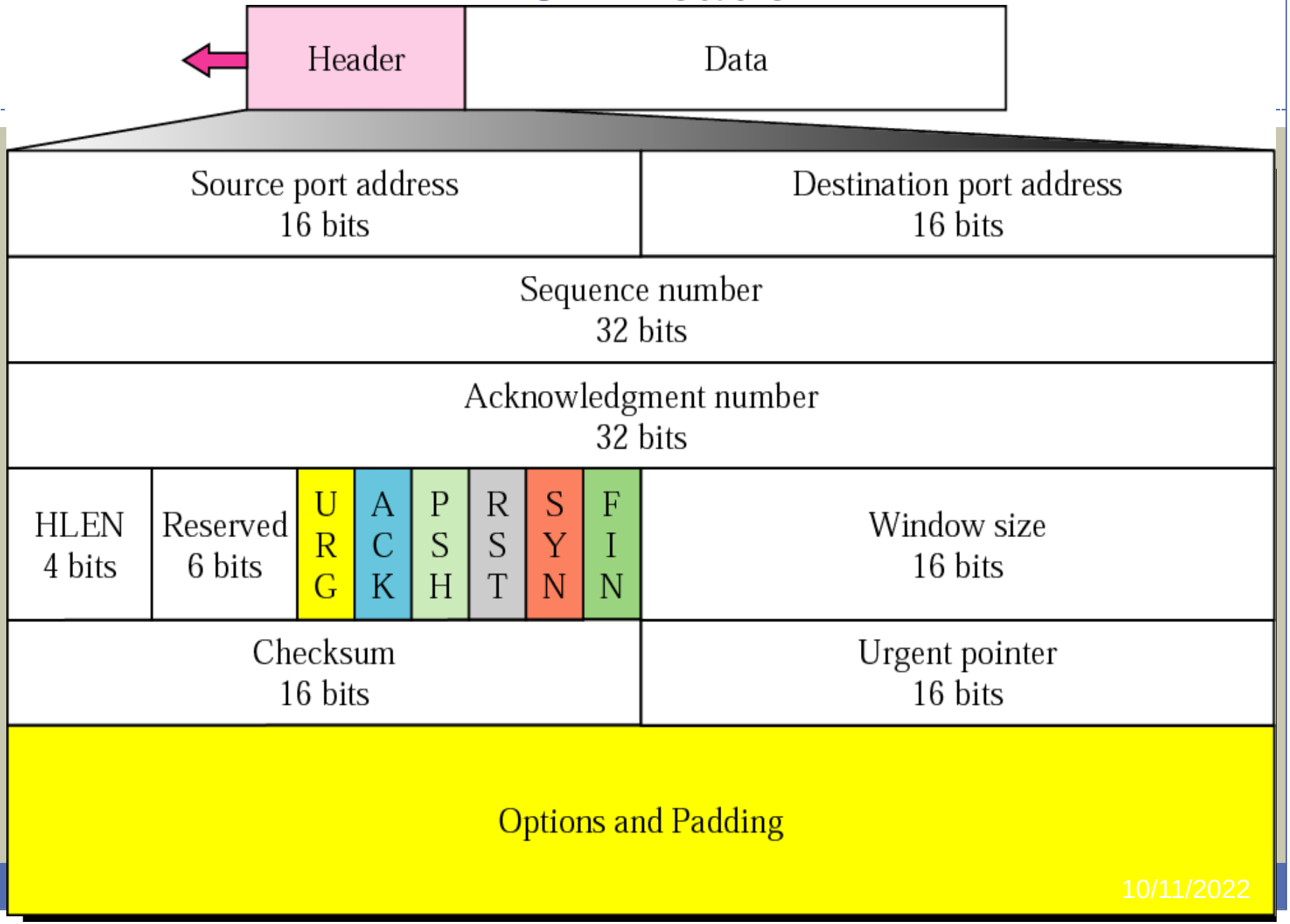
Acknowledgement numbers of data segments cont.



SN and AN in Segment transfer

Client				Server			
Segment	No. of byte(s)	SN	AN	Segment	No. of byte(s)	SN	AN
SYN	1	5000	-				
				SYN, ACK	1	2000	5001
ACK	1	5001	2001				
Data	100	5002	2001				
				Data	1500	2001	5102
Data	1000	5102	3501				
Data	2000	6102	3501				
				Data	3000	3501	8102
Data	500	8102	6501				
				Data	1000	6501	8602
FIN	1	8602	7501				
				ACK	1	7501	8603
				FIN	1	7502	8603
ACK	1	8603	7503				

TCP Header

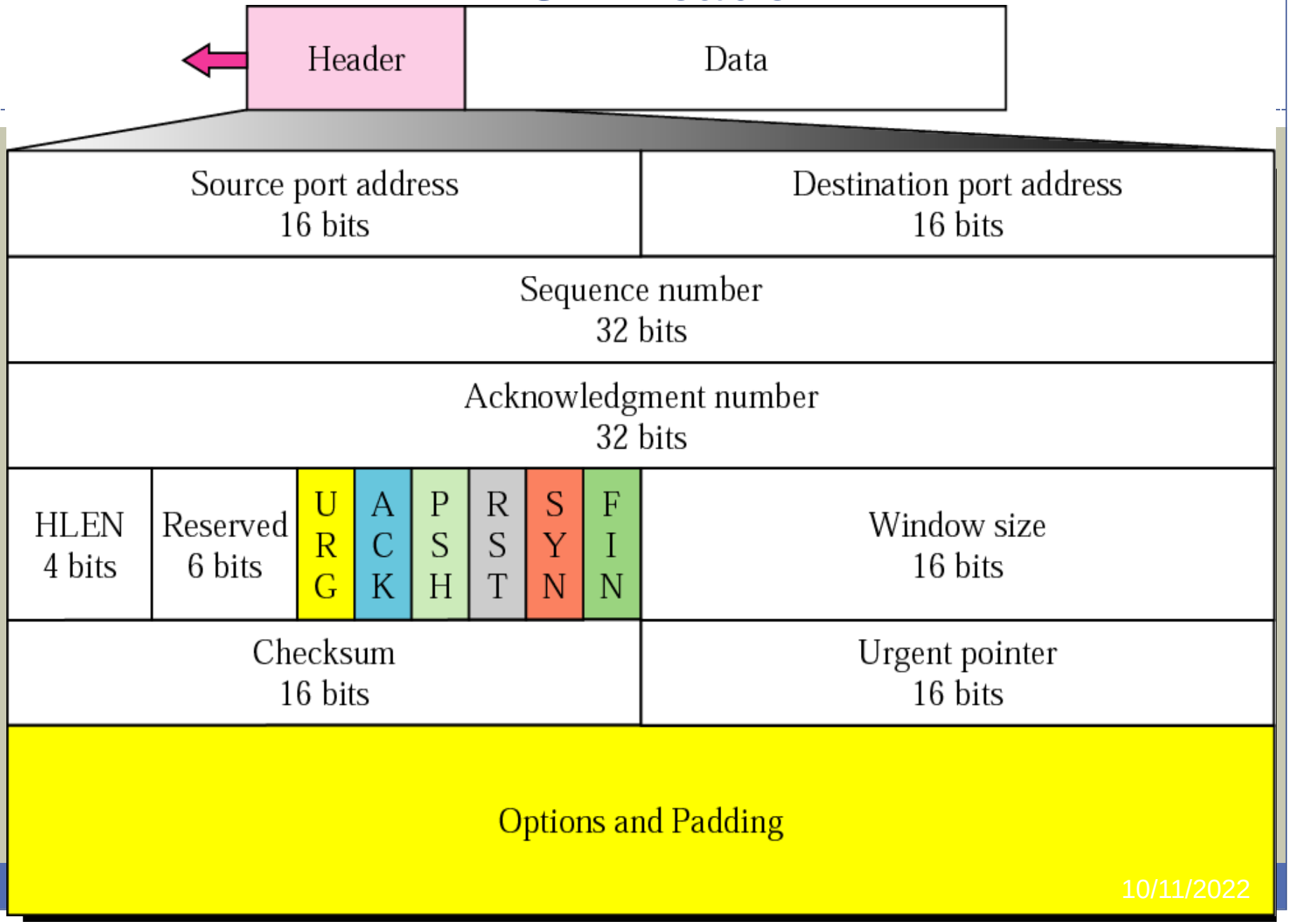

Header
Data
Source port address
16 bits
Destination port address
16 bits
Sequence number
32 bits
Acknowledgment number
32 bits
HLEN
4 bits
Reserved
6 bits
URG
ACK
PSH
RST
SYN
FIN
Window size
16 bits
Checksum
16 bits
Urgent pointer
16 bits
Options and Padding

Header Length (HLEN)

- This is a 4-bit field
- Header length in bytes = HLEN x 4
- The standard size of header is 20 bytes
- The maximum size of header is 60 bytes

(20 standard bytes and 40 optional bytes)

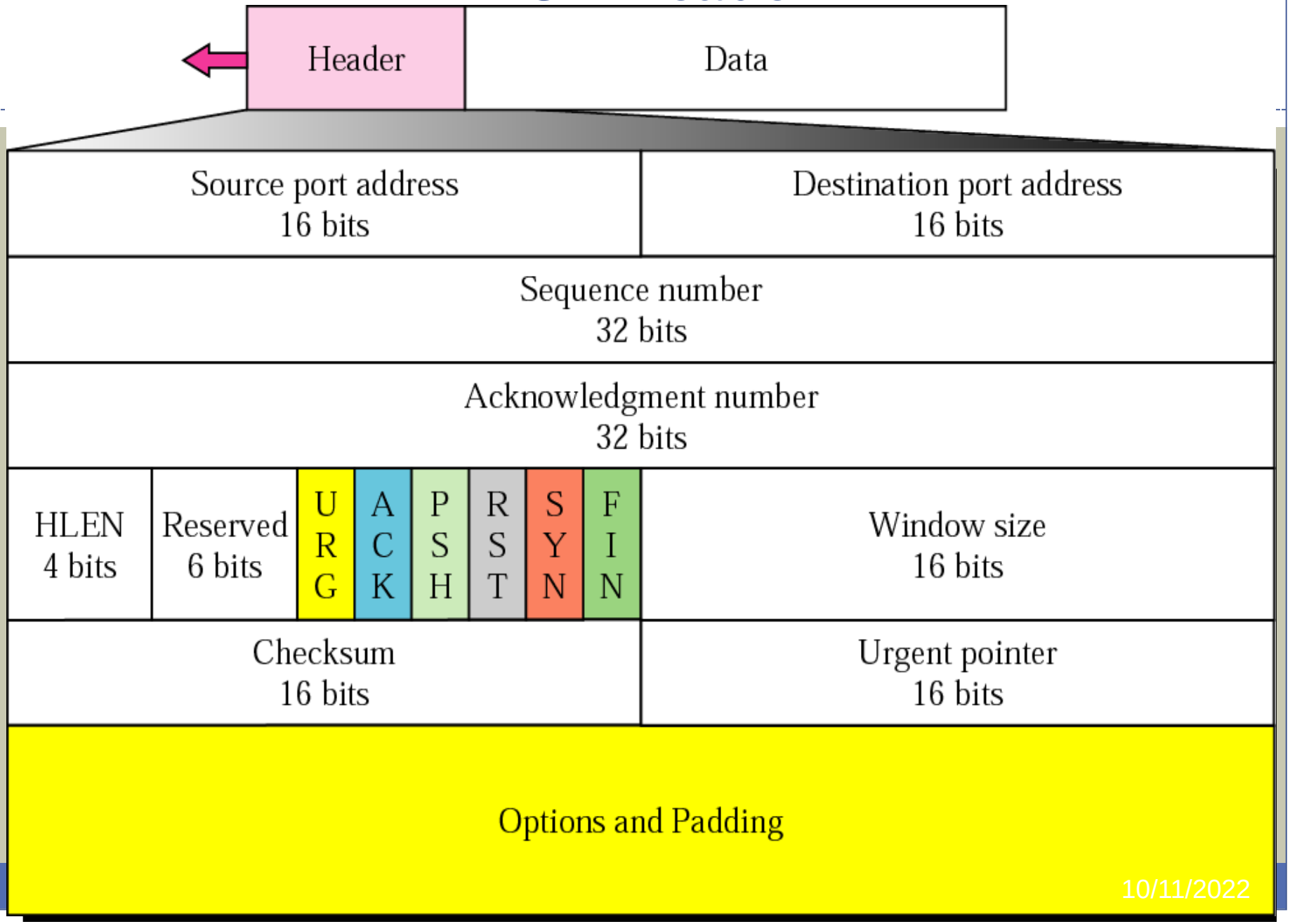
TCP Header


Header
Data
Source port address
16 bits
Destination port address
16 bits
Sequence number
32 bits
Acknowledgment number
32 bits
HLEN
4 bits
Reserved
6 bits
URG
ACK
PSH
RST
SYN
FIN
Window size
16 bits
Checksum
16 bits
Urgent pointer
16 bits
Options and Padding

Reserved

- This 6 bit field reserved for future use

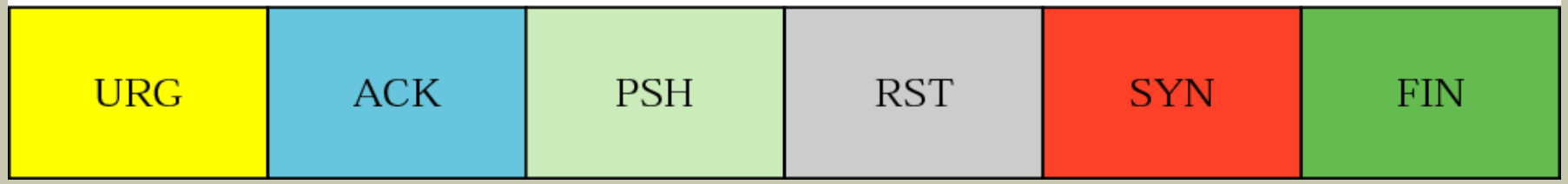
TCP Header


Header
Data
Source port address
16 bits
Destination port address
16 bits
Sequence number
32 bits
Acknowledgment number
32 bits
HLEN
4 bits
Reserved
6 bits
URG
ACK
PSH
RST
SYN
FIN
Window size
16 bits
Checksum
16 bits
Urgent pointer
16 bits
Options and Padding

Control

- This is also a six bit field

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



URG - Urgent

- Normally receiver reads segments according to ascending order
- If this flag is set, that segment should be read immediately

ACK - Acknowledgement

- If data is not available in the segment,

  then this is an ACK segment
- If data is available in the segment,

  and if the acknowledgement number is ‘n’’

  it indicates that data bytes up to byte n-1 is OK and waiting for sequence number n

PSH - Push

- If push flag is set, it does not care about the **window size** recommended by other party and **start the data transmission** to the other party

RST - Reset

- The connection must be reset (Destroy / Terminate)
- This can happen due to three reasons
  - **Request for an unidentified port**
  - **Client or server has a problem**
  - **The other side TCP is idle for a long time**

RST - Reset cont.

Reason 1 :

- The client requests a connection for server to an unidentified port
- In such situation server send RST to client to destroy the connection

RST - Reset cont.

Reason 2 :

- The connection has been established
- After some time the client or server has a problem and request the other party to destroy the connection
- Then it (client or server) sends RST to other party.

RST - Reset cont.

Reason 3 :

- The other side TCP is idle for a long time
- Then RST is sent to the other party to destroy the connection

SYN - Synchronize

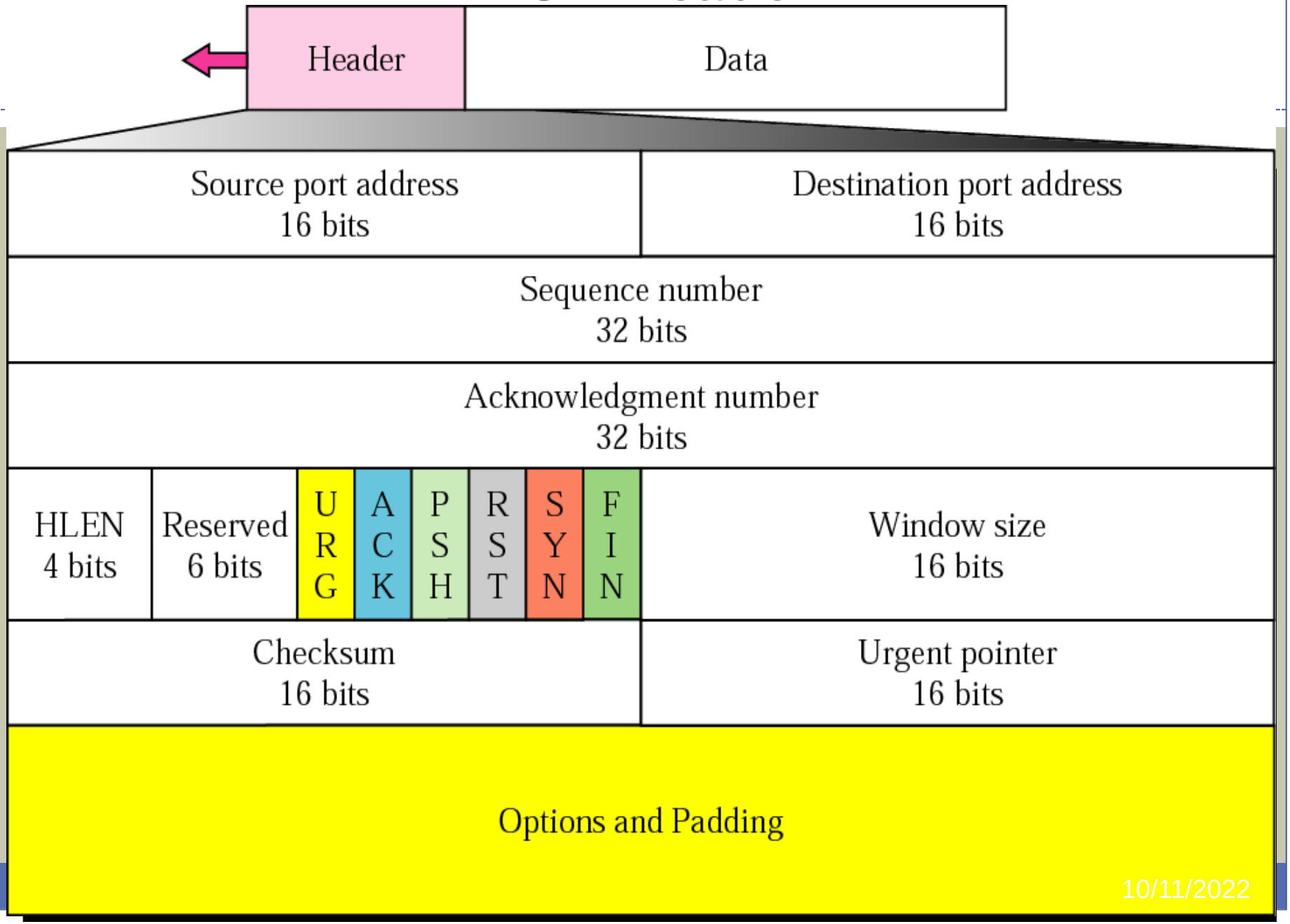
- Synchronize to sequence numbers (Initial Sequence Number) suggested at the time of connection establishment

FIN - Finish

- Request to terminate the connection to that direction
- If client sent FIN (that is set FIN flag) to server,

  it means that client request from server to disconnect the connection of client server direction

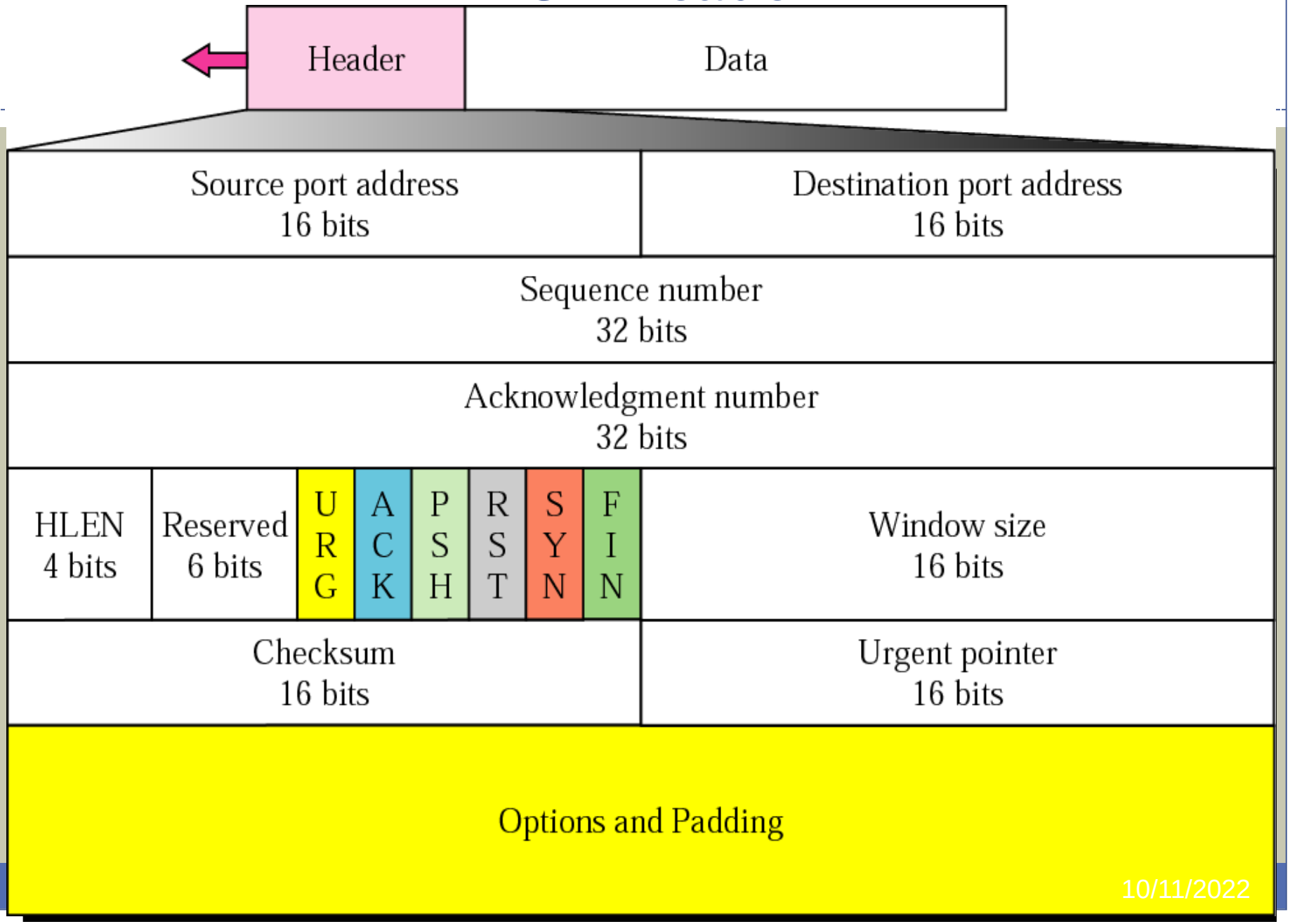
TCP Header


Header
Data
Source port address
16 bits
Destination port address
16 bits
Sequence number
32 bits
Acknowledgment number
32 bits
HLEN
4 bits
Reserved
6 bits
URG
ACK
PSH
RST
SYN
FIN
Window size
16 bits
Checksum
16 bits
Urgent pointer
16 bits
Options and Padding

Window size

- This is a 16-bit field
- Indicates how many bytes (segment data) can be maintained in the other party

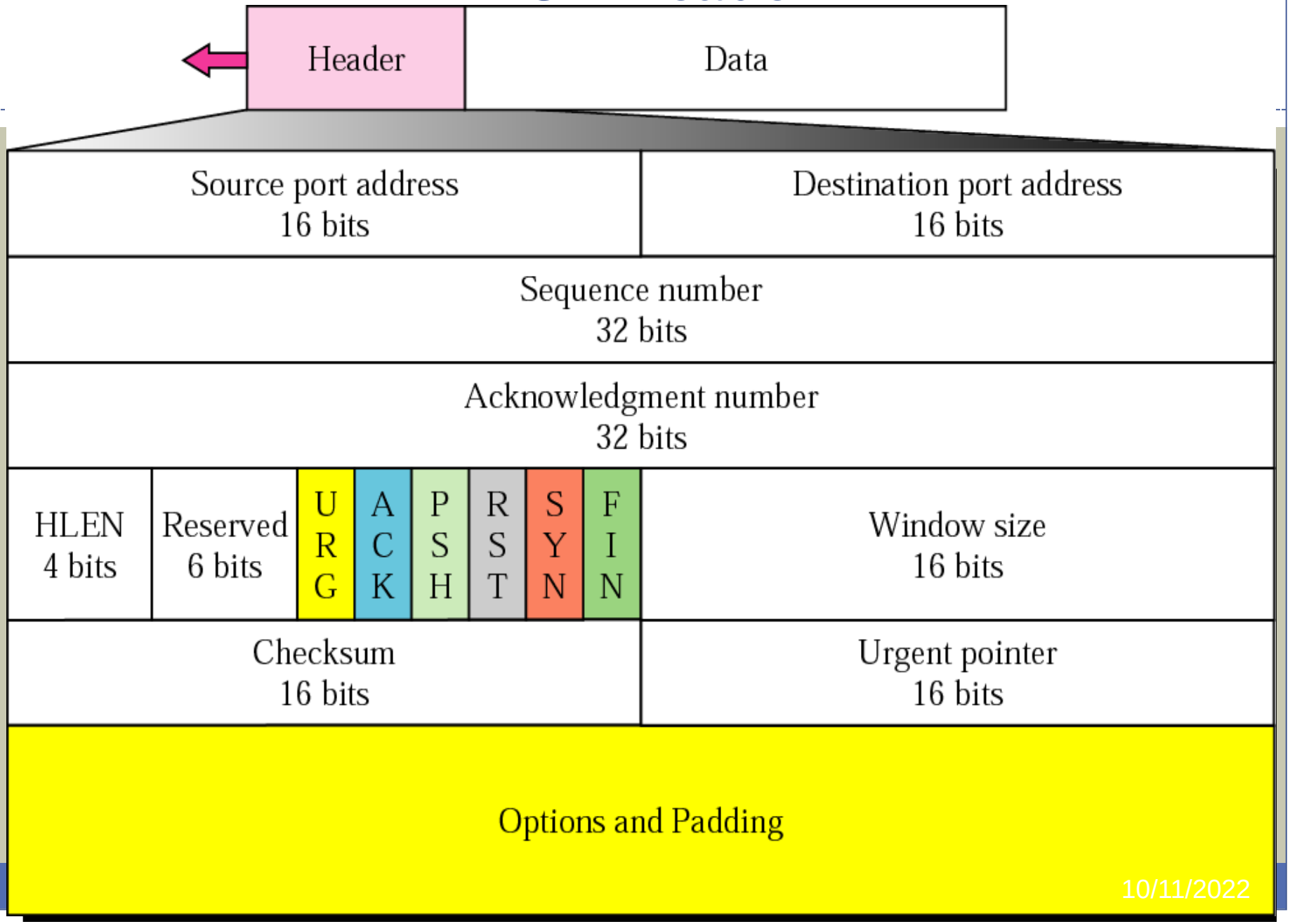
TCP Header


Header
Data
Source port address
16 bits
Destination port address
16 bits
Sequence number
32 bits
Acknowledgment number
32 bits
HLEN
4 bits
Reserved
6 bits
URG
ACK
PSH
RST
SYN
FIN
Window size
16 bits
Checksum
16 bits
Urgent pointer
16 bits
Options and Padding

Checksum

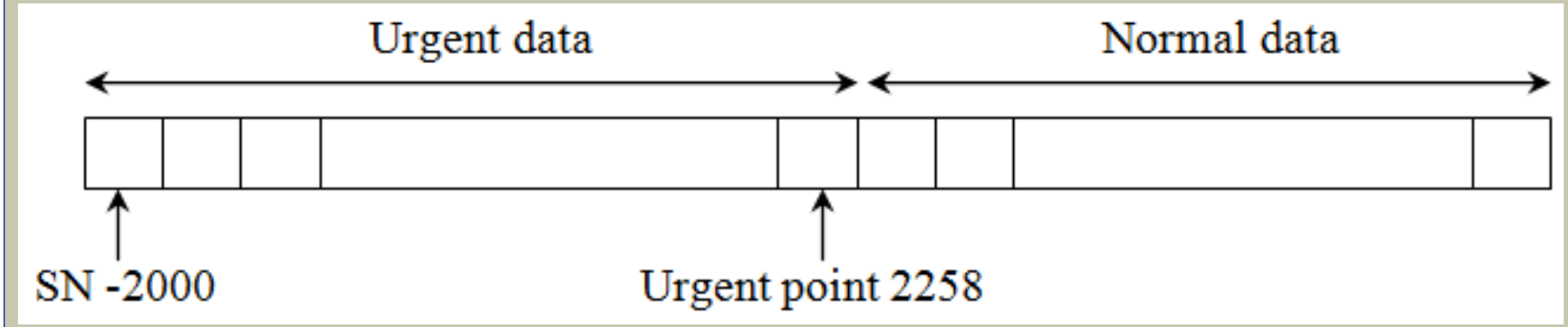
- This is a 16 bit field
- Used to check whether there are any errors

TCP Header


Header
Data
Source port address
16 bits
Destination port address
16 bits
Sequence number
32 bits
Acknowledgment number
32 bits
HLEN
4 bits
Reserved
6 bits
URG
ACK
PSH
RST
SYN
FIN
Window size
16 bits
Checksum
16 bits
Urgent pointer
16 bits
Options and Padding

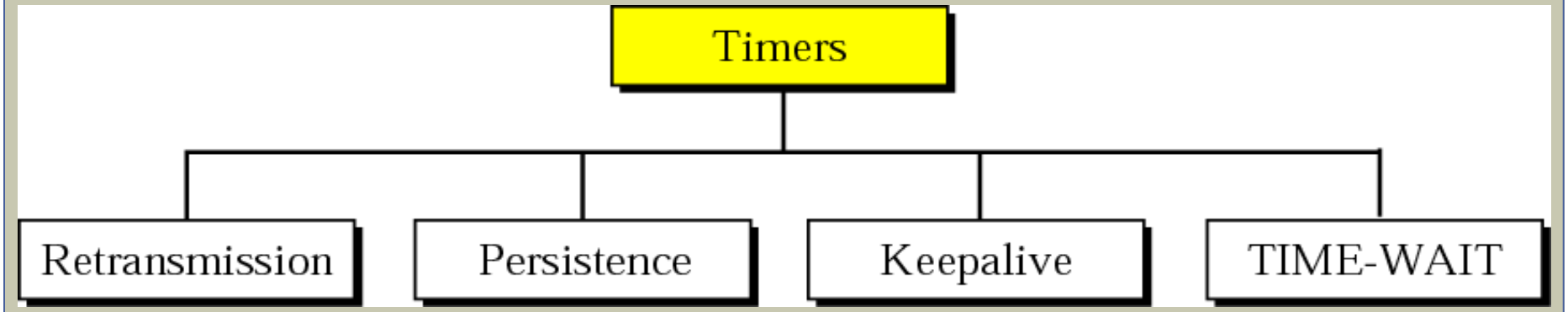
Urgent Pointer

- This is a 16-bit field valid only if the urgent flag is set
- This value gives the byte value **at the end of urgent data**



- The bytes 2000 to 2258 are urgent data

TCP Timers



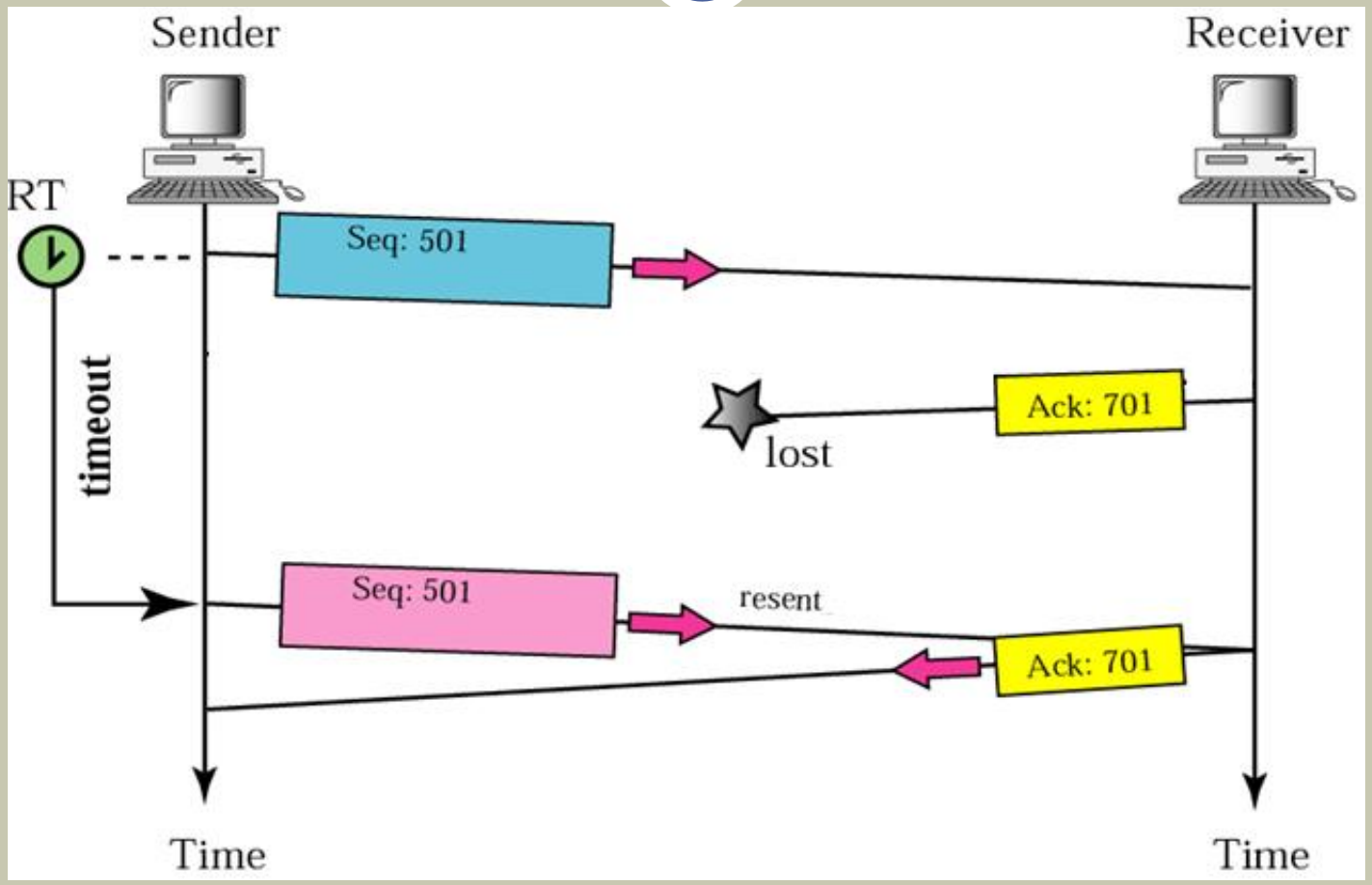
TCP Timers cont.

Timer	Purpose
Retransmission	For error control
Persistence	To avoid problems of zero window size advertisement
Keep alive	To check whether client is alive if it is idle for a long time
Time- waited	To avoid problems with delayed FIN segments

Retransmission Timer

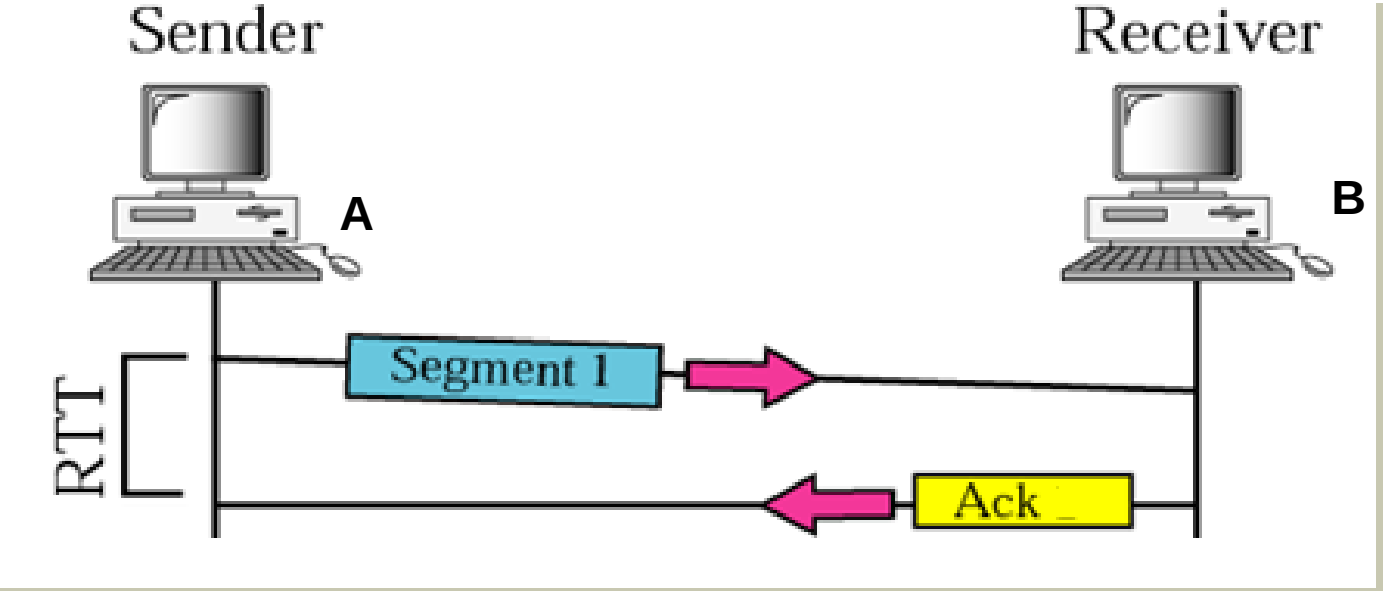
- This timer starts after sending a segment
- If the acknowledgement is not received before this timer expires, the same segment is re-transmitted and the timer is reset
- If the acknowledgement is received before retransmission timer expires that timer will be destroyed (Timer will be stopped)

Retransmission Timer cont.



- Retransmission time =2 x Round Trip Time

Round Trip Time



The time involved here are

- Propagation delay from A to B
- Processing time at B
- Propagation delay from B to A
- The total of above times are called Round Trip Time

Measured Round Trip Time

- Can measure Round Trip Time Using the “time stamp” obtained from the option field of TCP header
- TCP sends a segment, starts a timer, and waits for acknowledgement
- It measures the time between sending of the segment and receiving of the acknowledgement

Setting the Round Trip Time

Setting RTT= **α x Previous RoundTT+(1-α) Current RoundTT**

α = 90%

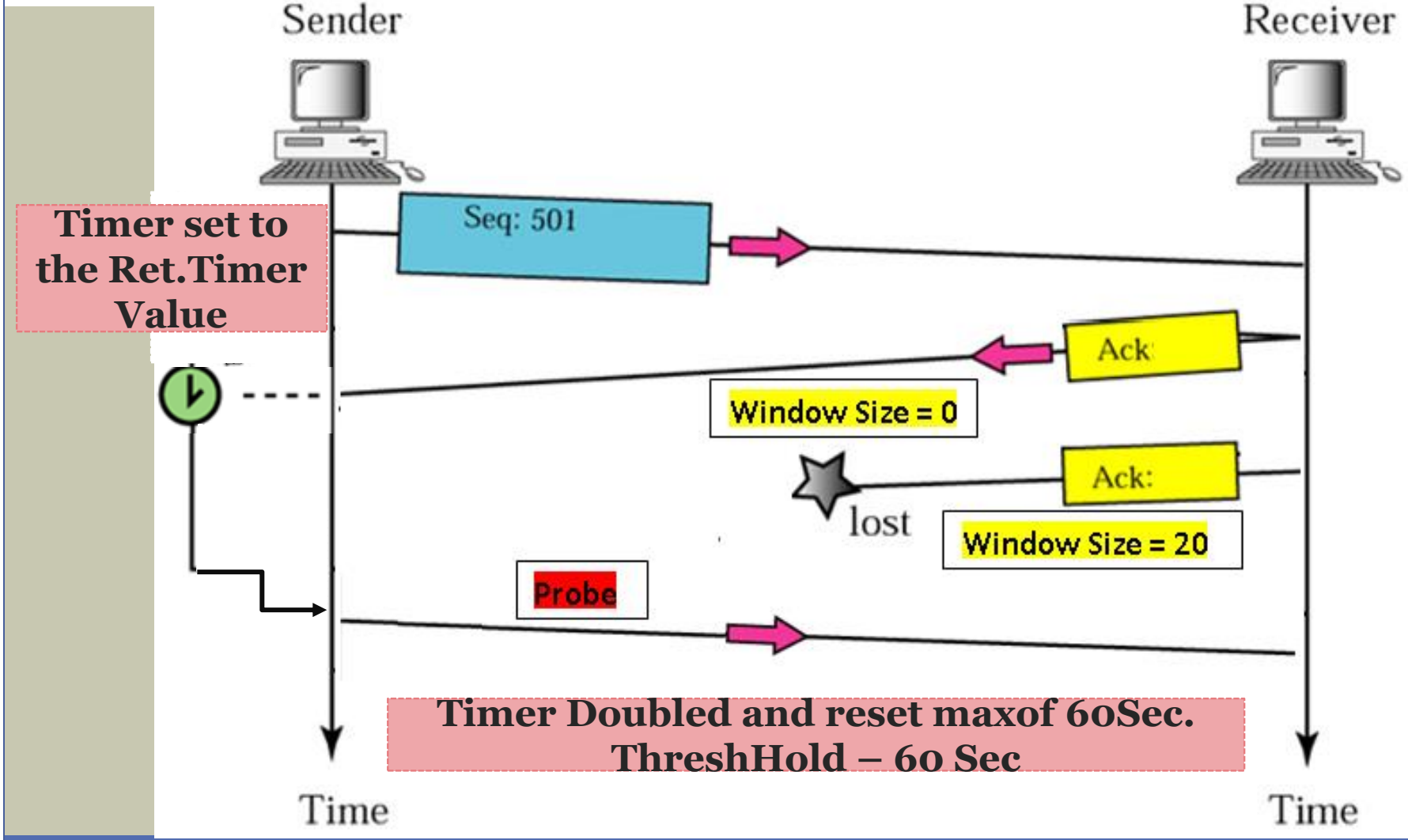
Setting RTT= 0.9 x Previous RoundTT+ 0.1 Current RoundTT

- There is a difference between measured Round Trip Time and setting Retransmission Time
  - Retransmission timer =2 x **Setting** RTT

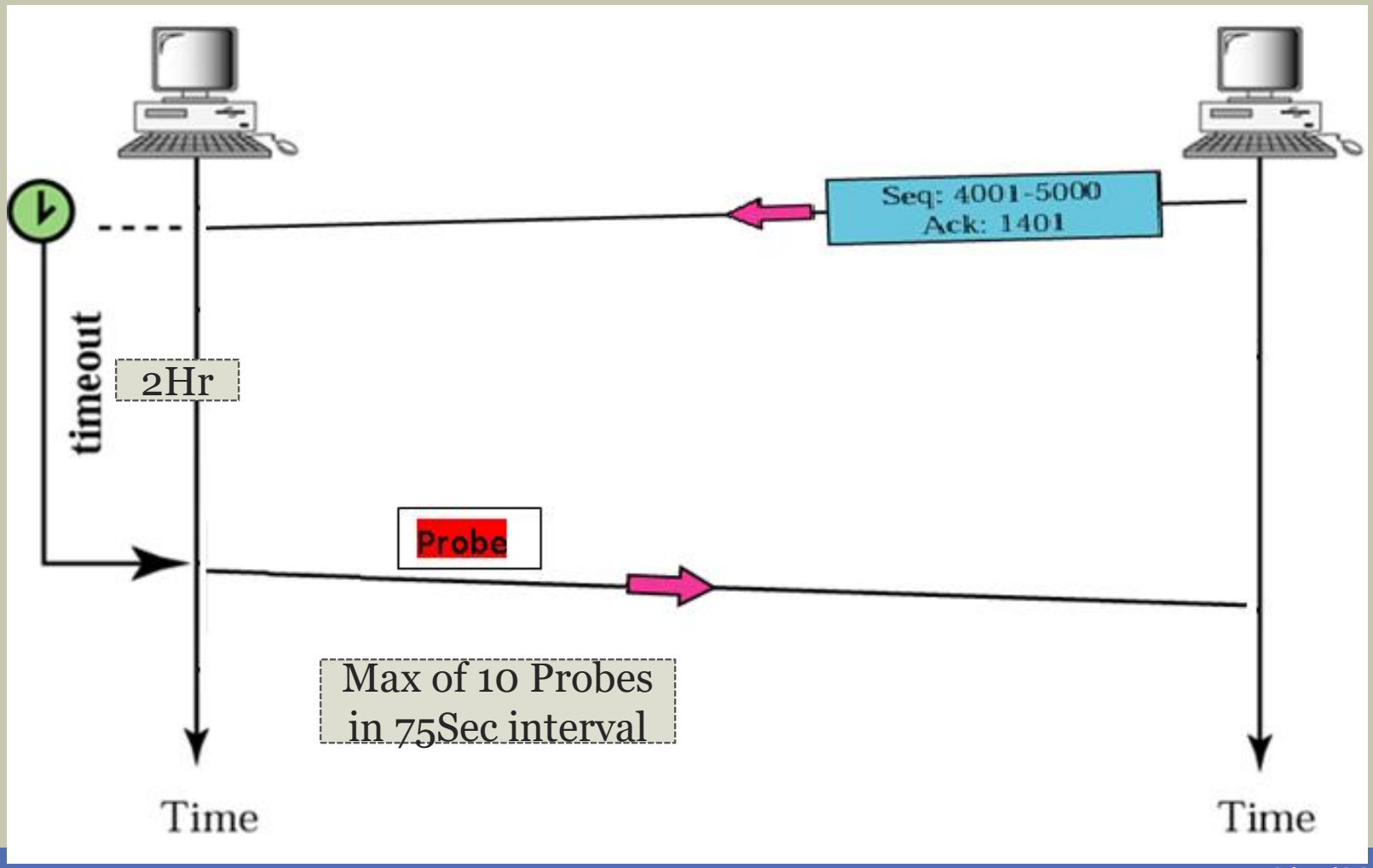
RTT cont.

Segment	Measured RTT (ms)	Set RTT (ms)	Ret Timer
n	250	---	
n+1	200	---	500
n+2	180	90%250+10%200 = 245	490
n+3	220	90%200+10%180 = 198	396

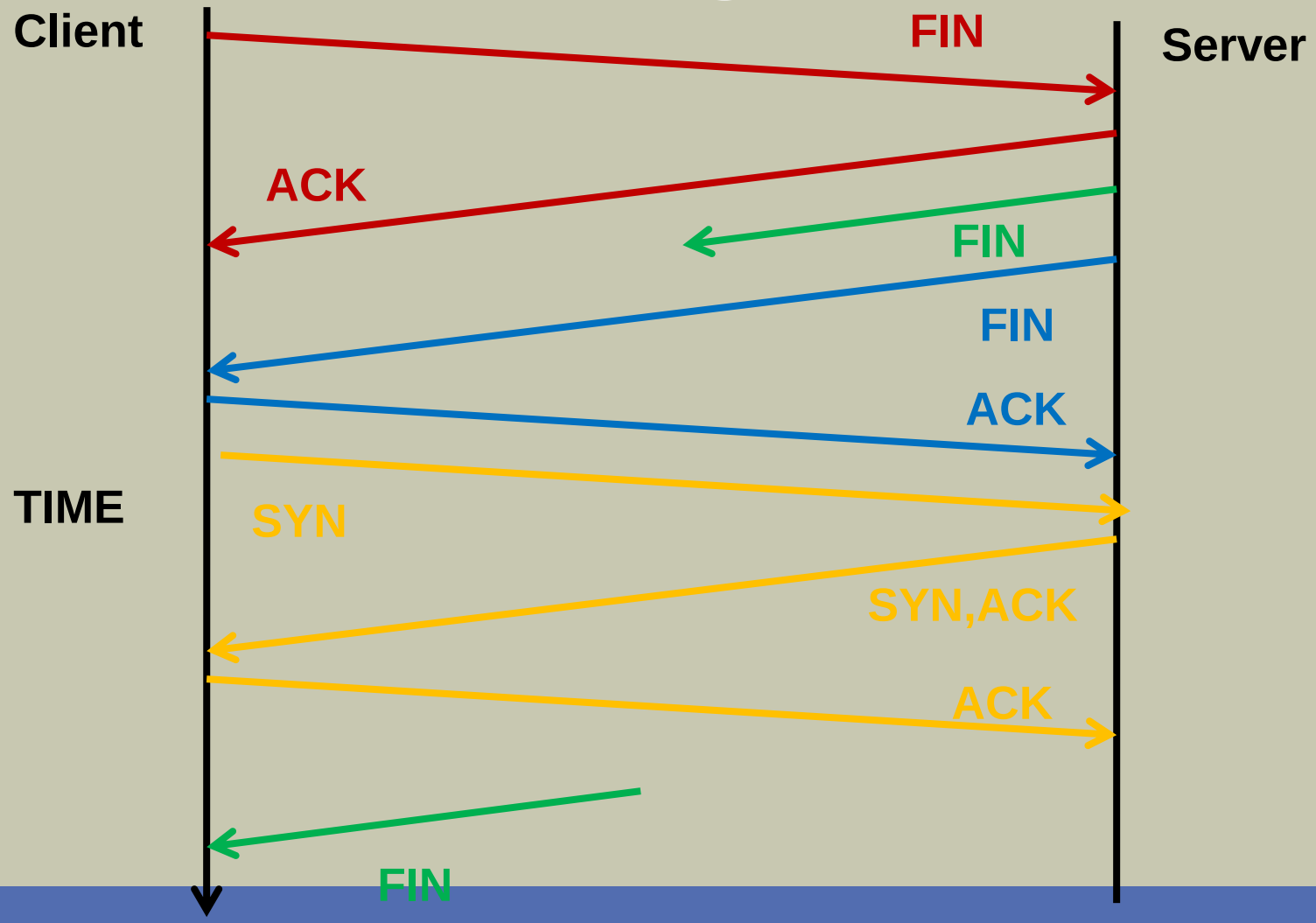
Persistence Timer



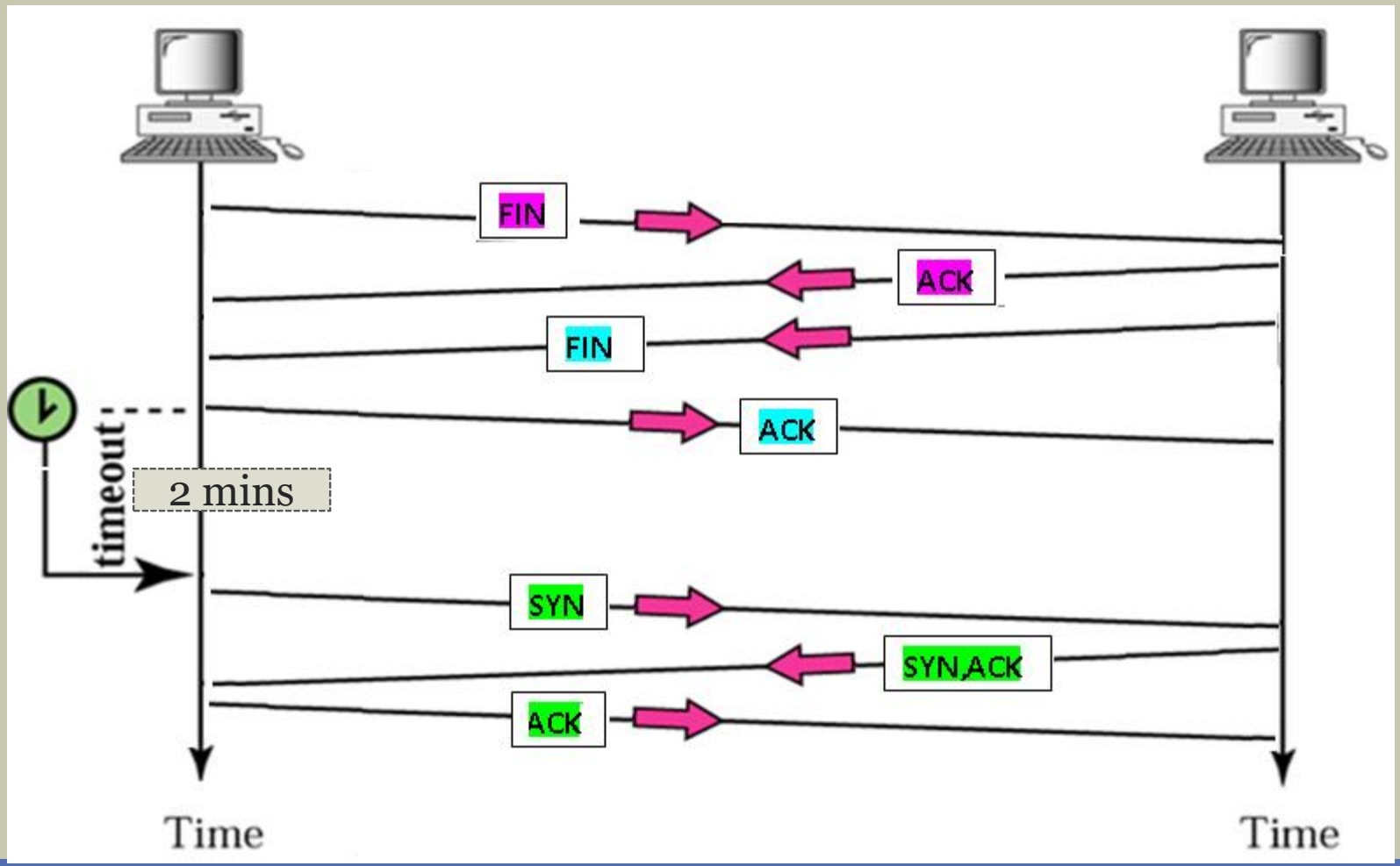
Keep alive Timer



Time Waited Timer



Time Waited Timer cont.



Error Control

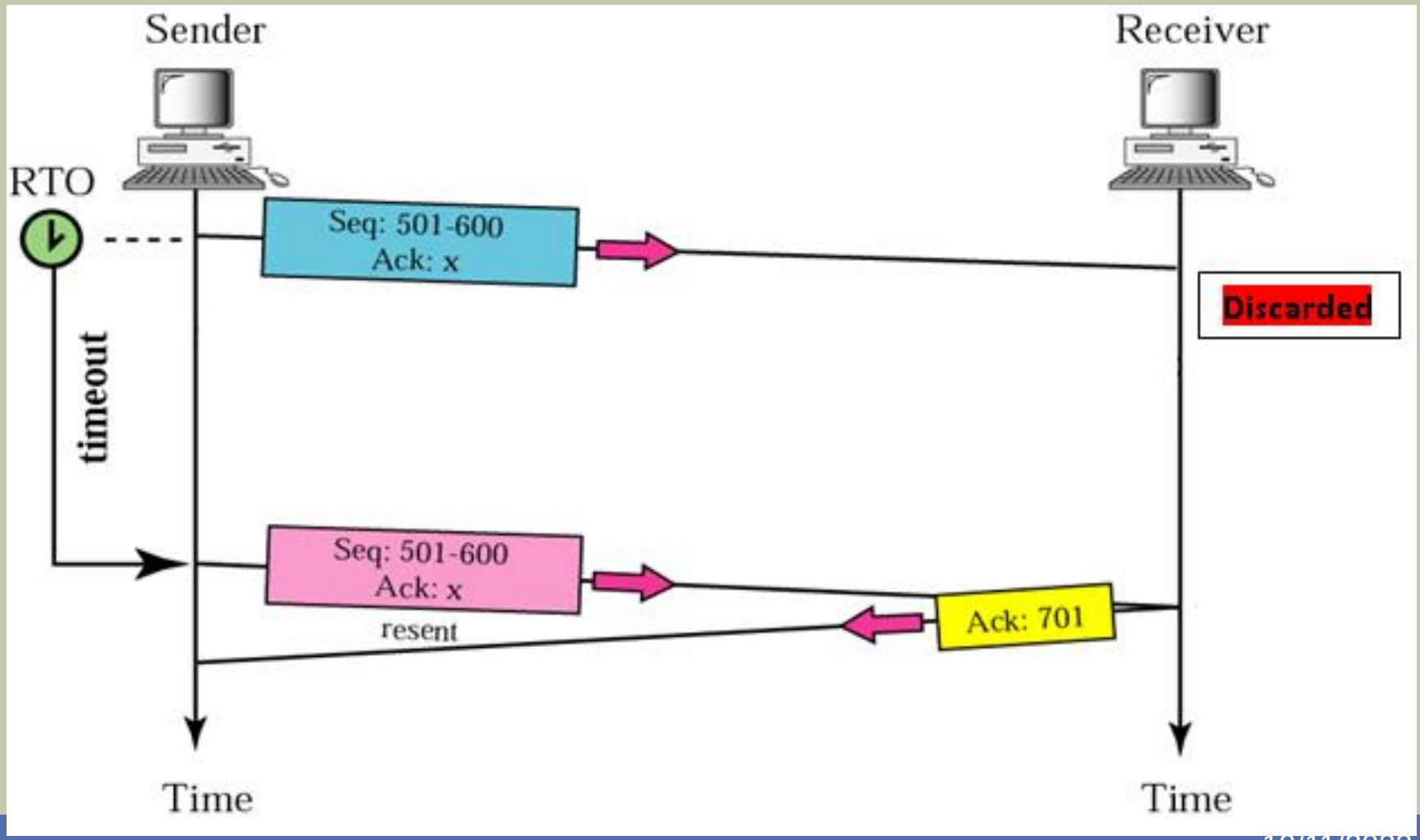
- TCP uses the backward error control
- For error detection, the checksum bits in the TCP header is used by the receiver
- If the receiver detects errors, it will discard that segment
- The receiver will not send any negative acknowledgement to sender

Error Control cont.

Error control process considers,

- Errors in the received segment (corrupted segments)
- Segment is lost on the way before reaching the receiver (last segment)
- Duplicate received segments
- Out of order segments (the segment numbers are not received in order)
- Lost an acknowledgement on the way

Corrupted Segment



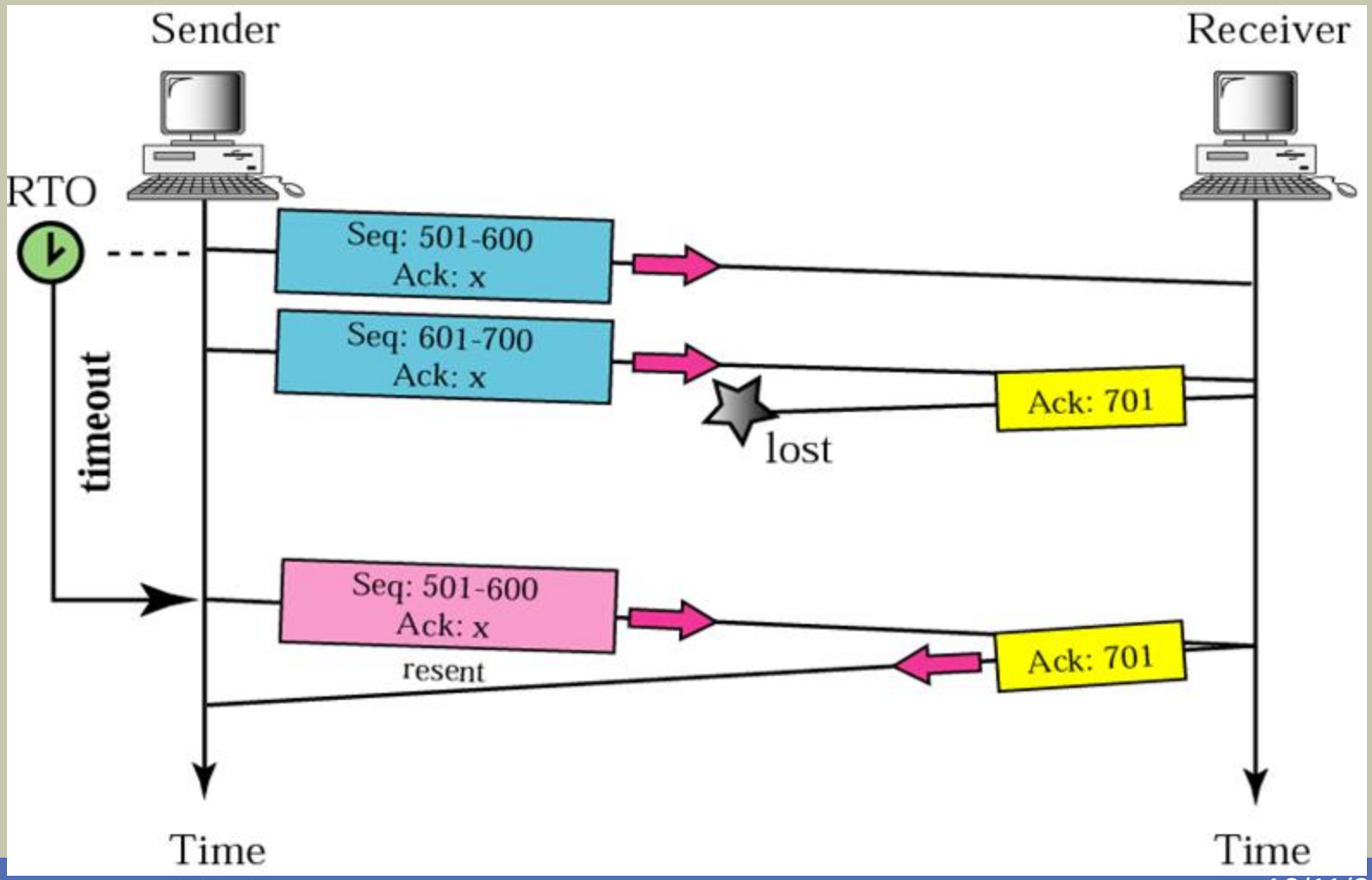
Out of order segment

- The IP packets can travel through different routers
- Therefore the segments can reach the receiver TCP layer out of order
- The TCP layer waits until previous segment(s) are received and then acknowledges

Duplicate segment

- If IP packet travels through a long way the retransmission timer expire and sender retransmit the same segment
- The receiver receives both segments
- If the first segment received has no errors, it is accepted and acknowledged
- The second segment is ignored by the TCP layer

Lost acknowledgement

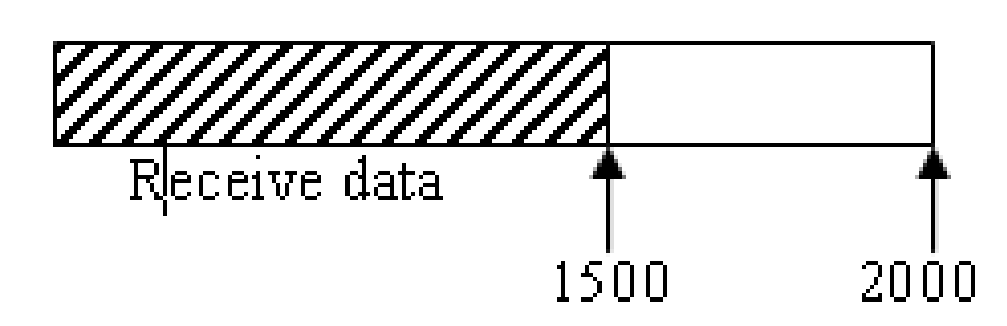


Window Size

- Receiver TCP buffer can be overflowed due to two reasons
  - The receiver TCP buffer receives data very fast
  - The receiver application consumes data very slowly
- Receiver TCP should inform the sender TCP how much bytes of data it can accommodate
- It is called the window size

Windows Size cont.

- Scenario 1 : Suppose the TCP buffer size is 2000 bytes



- If it receives 1500 bytes of data, 500 bytes buffer space is free
- Then it informs the sender the window size is 500

Windows Size cont.

- Scenario 2 : Application may have consumed another 800 bytes,
- The remaining data bytes in the buffer is

  1500 – 800 = 700 bytes
- New window size is

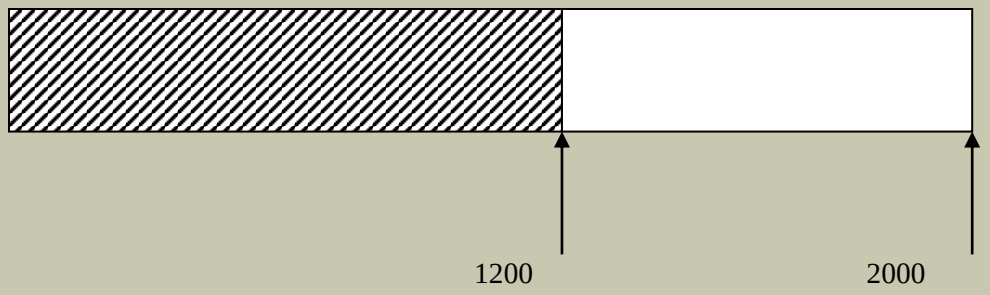
  2000-700 = 1300

Windows Size cont.

- Scenario 3 : Sender may send 500 bytes
- The remaining data bytes in the buffer is

  700 + 500 = 1200 bytes
- New window size is

  2000-1200 = 800



Windows Size cont.

- Scenario 4 : Sender may send 800 bytes. But the receiver application was busy and it did not consume any data.
- The remaining data bytes in the buffer is

  1200 + 800 = 2000 bytes
- New window size is

  2000-2000 = 0

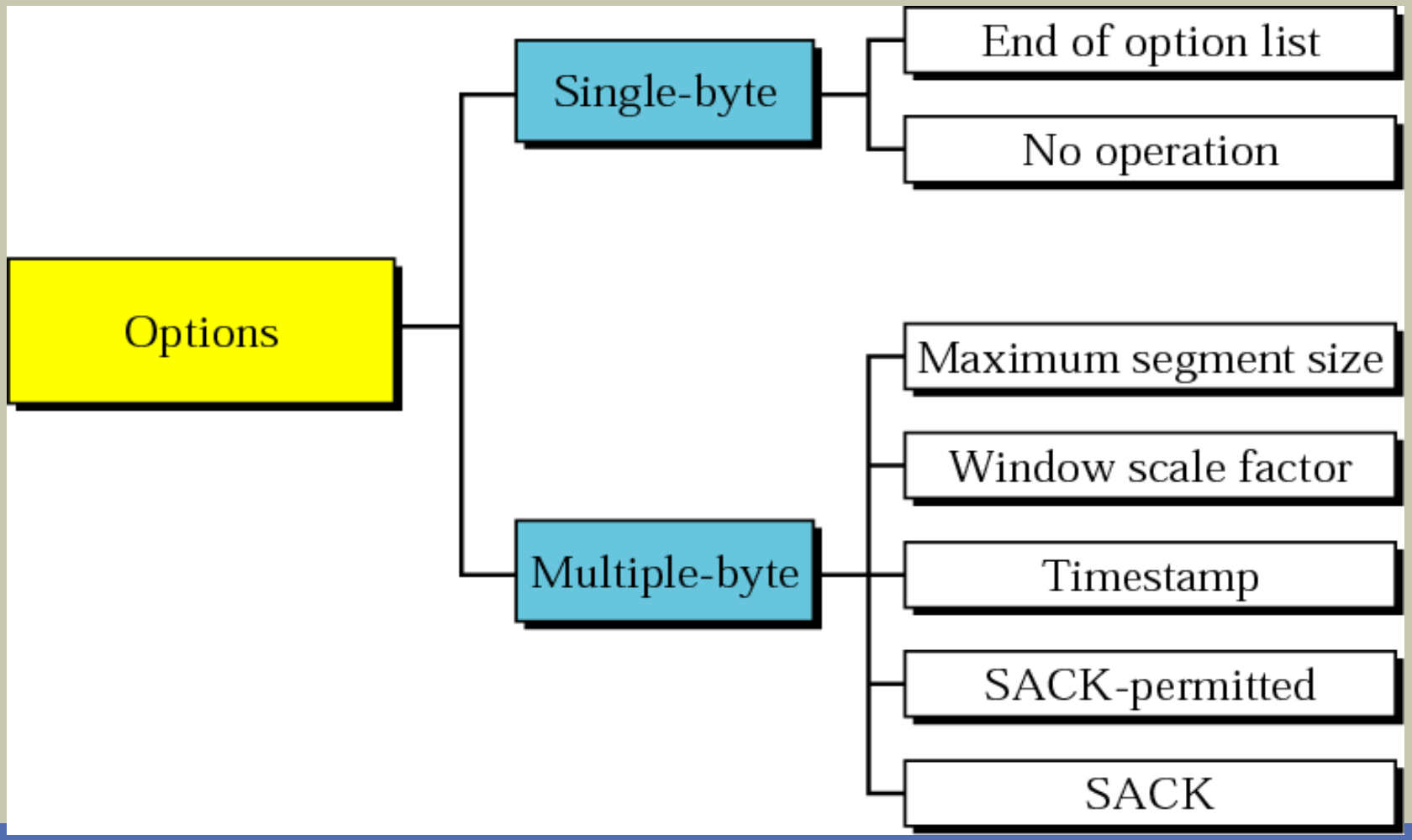
Windows Size cont.

- Scenario 5 : Receiver application consumed 200 bytes of data
- The remaining data bytes in the buffer is

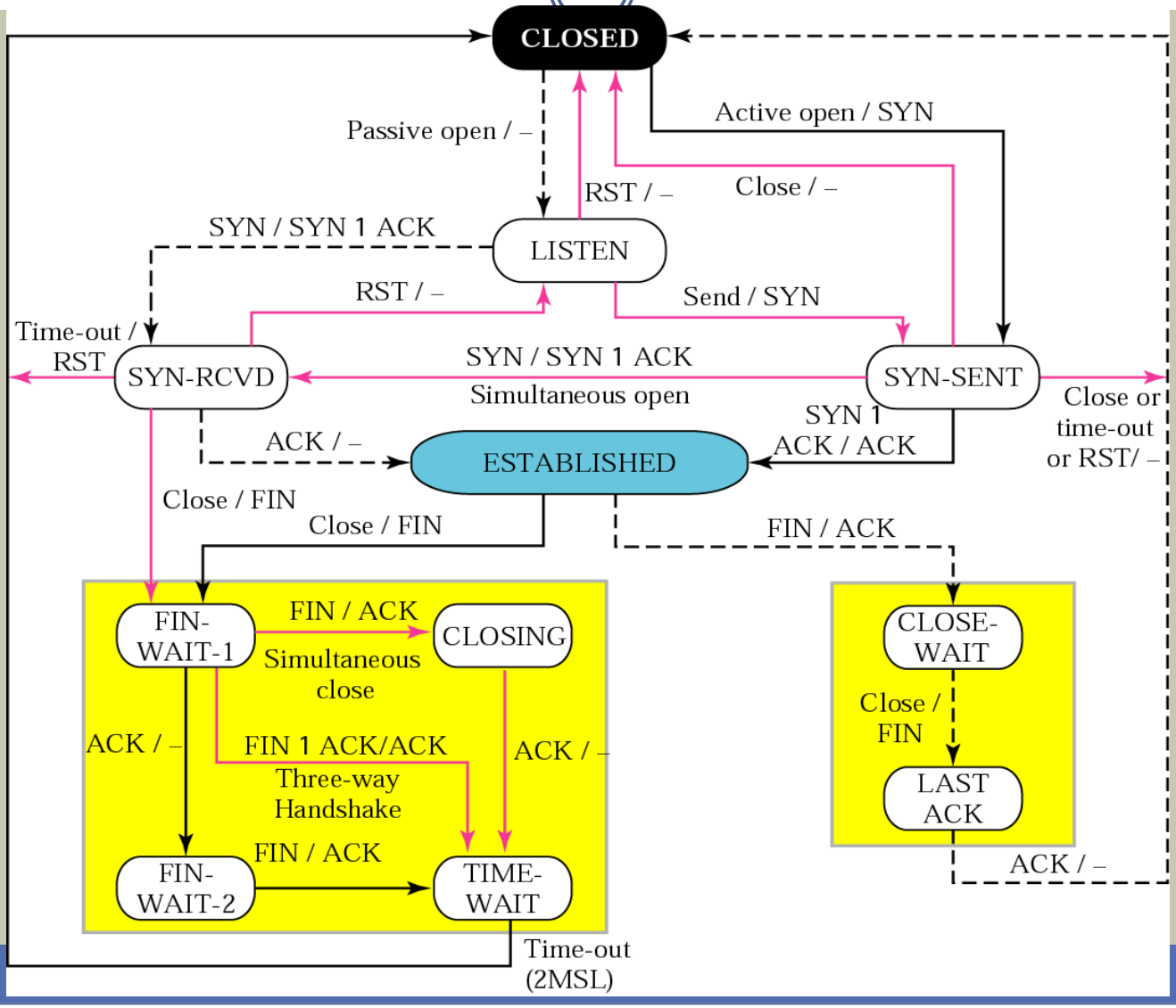
  2000 - 200 = 1800 bytes
- New window size is

  2000-1800 = 200

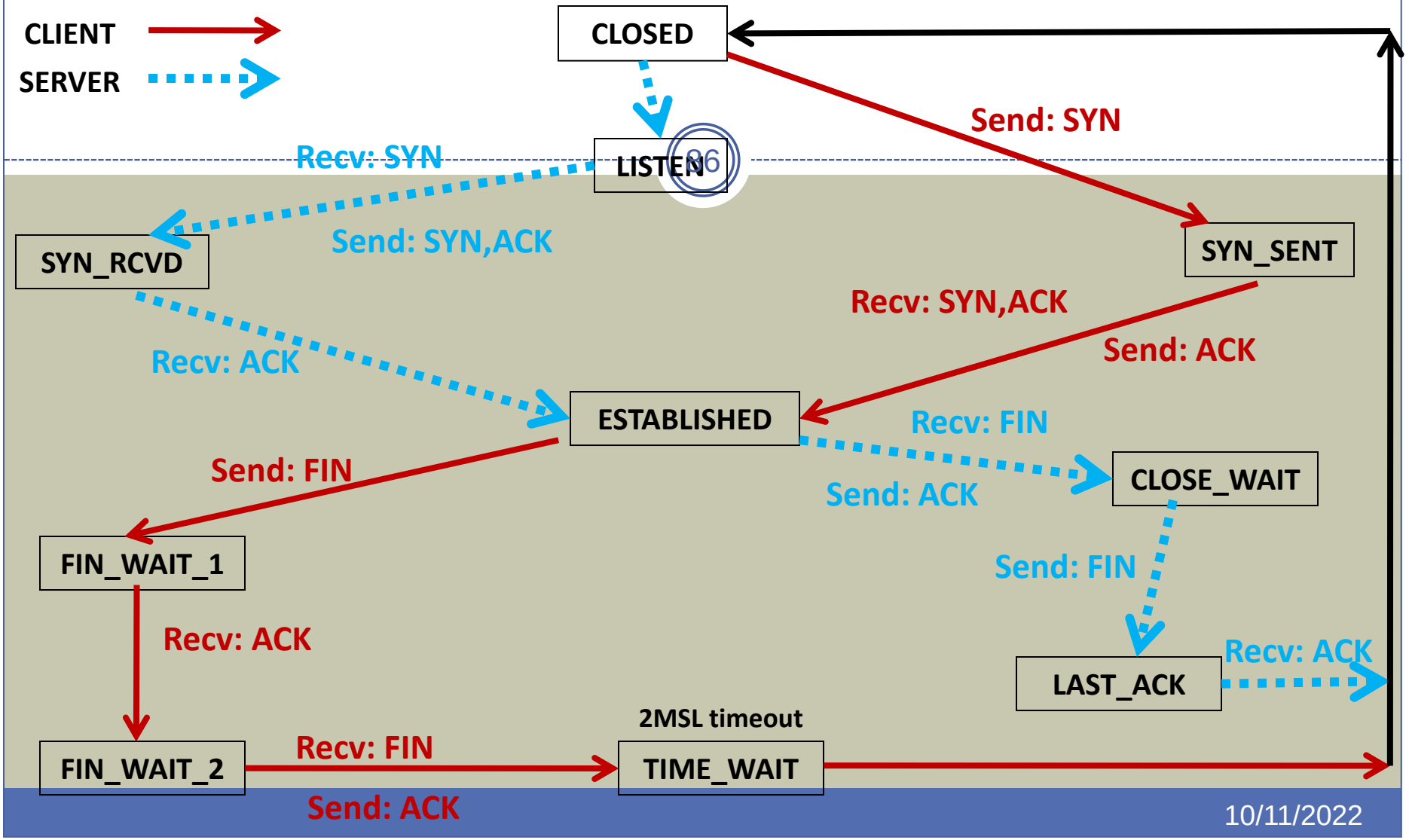
TCP Option

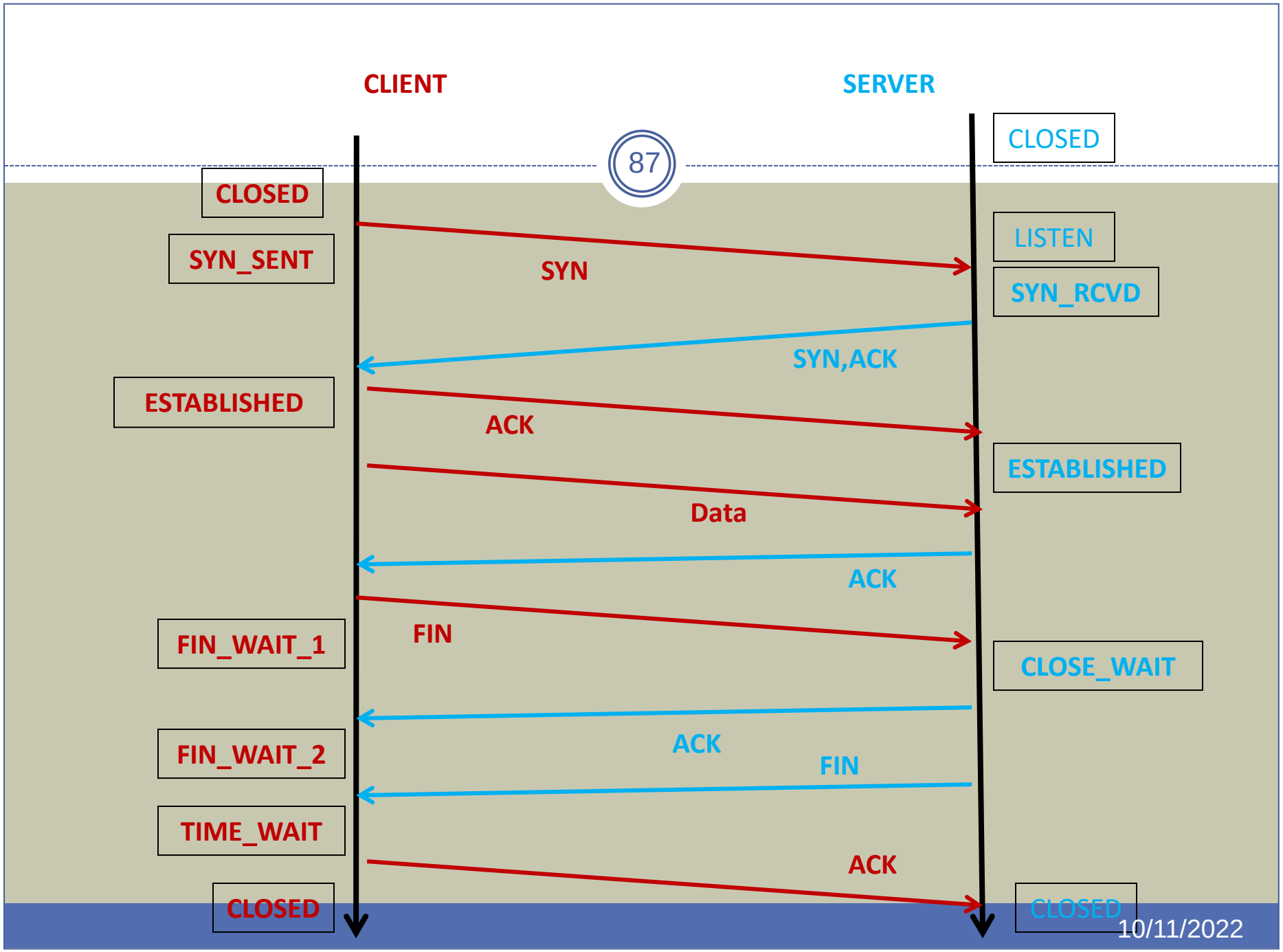

Options
Single-byte
End of option list
No operation
Multiple-byte
Maximum segment size
Window scale factor
Timestamp
SACK-permitted
SACK

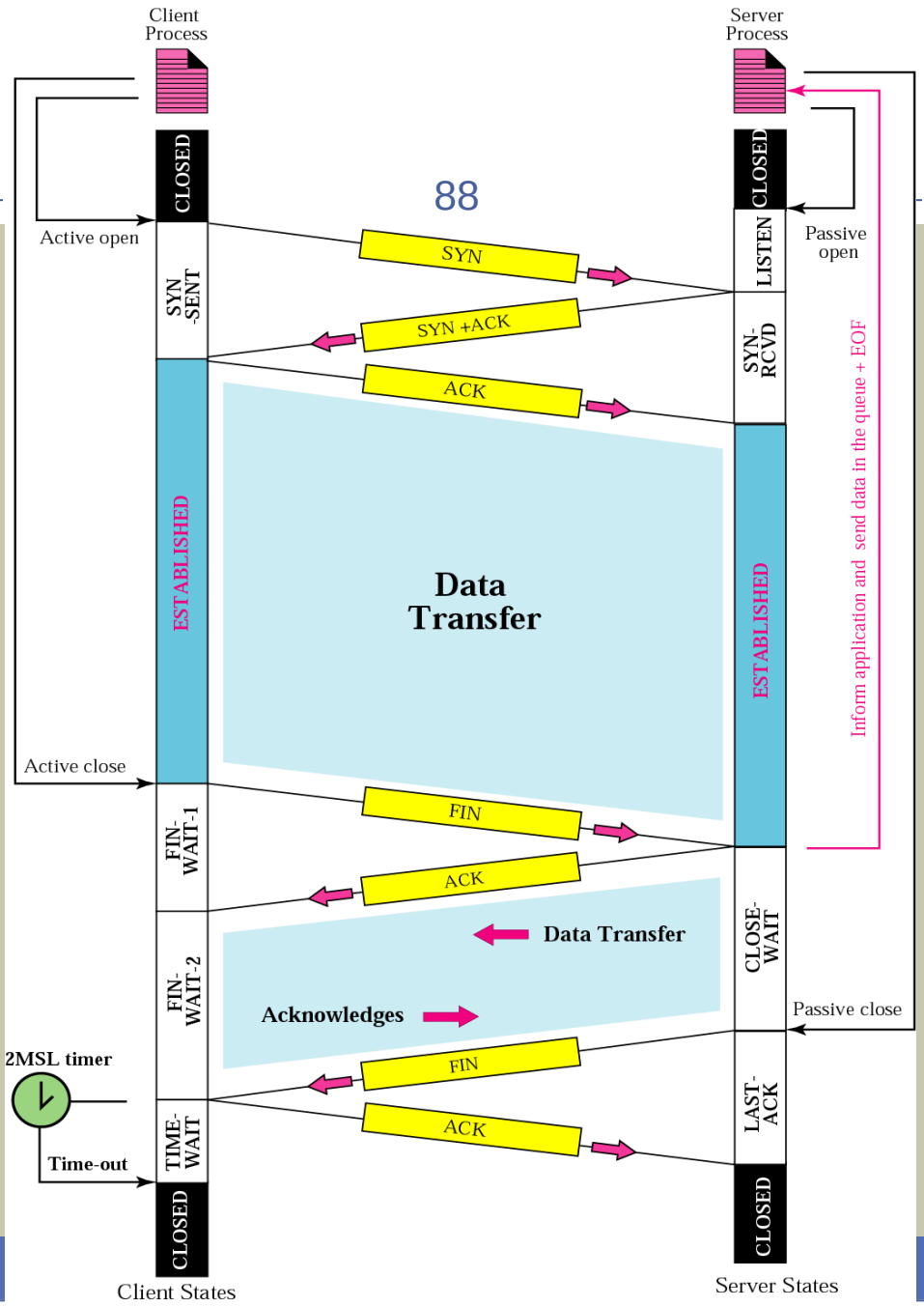
TCP state Transition Diagram

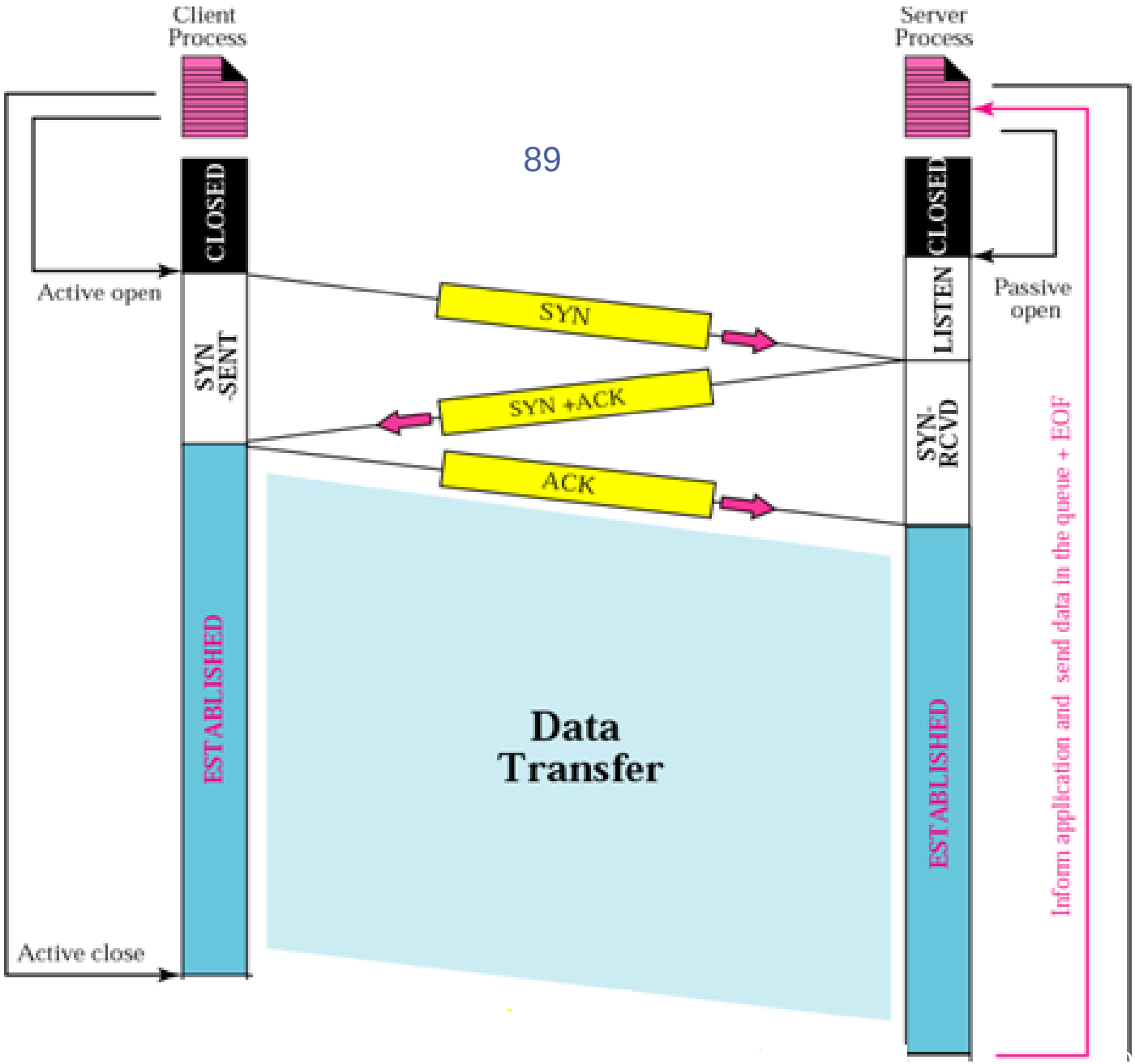


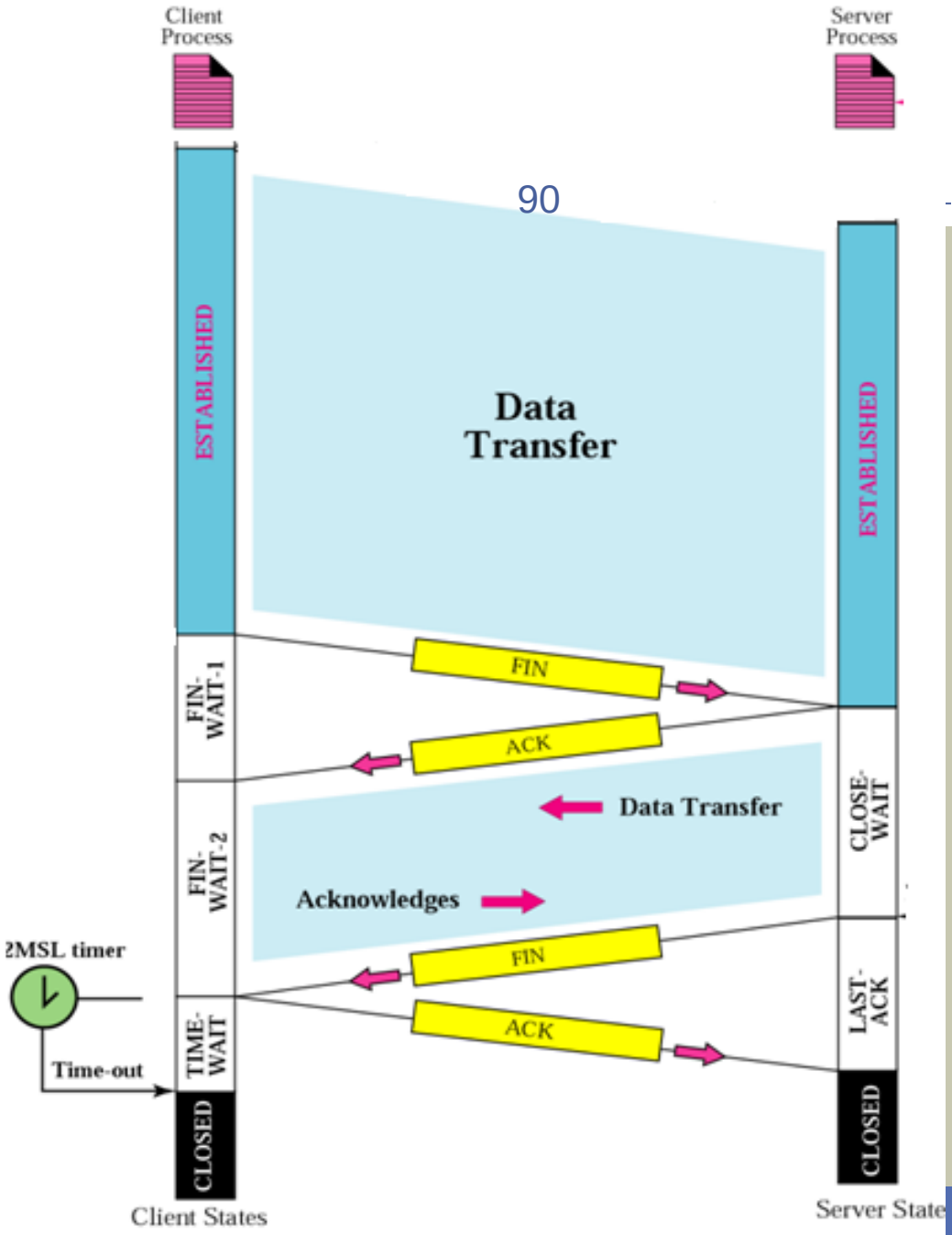
TCP state transition




CLIENT
SERVER
87
CLOSED
CLOSED
LISTEN
SYN_SENT
SYN
SYN_RCVD
SYN,ACK
ESTABLISHED
ACK
ESTABLISHED
Data
ACK
FIN_WAIT_1
FIN
CLOSE_WAIT
ACK
FIN_WAIT_2
FIN
TIME_WAIT
ACK
CLOSED
CLOSED


Client Process
Server Process
CLOSED
CLOSED
Active open
Passive open
SYN
LISTEN
SYN-SENT
SYN +ACK
SYN-RCVD
ACK
ESTABLISHED
Data Transfer
ESTABLISHED
Inform application and send data in the queue + EOF
Active close
FIN
ACK
FIN-WAIT-1
Data Transfer
Acknowledges
CLOSE-WAIT
FIN-WAIT-2
Passive close
2MSL timer
FIN
ACK
LAST-ACK
Time-out
TIME-WAIT
CLOSED
CLOSED
Client States
Server States


Client Process
Server Process
CLOSED
CLOSED
Active open
SYN
LISTEN
Passive open
SYN SENT
SYN +ACK
SYN-RCVD
ACK
ESTABLISHED
Data Transfer
ESTABLISHED
Inform application and send data in the queue + EOF
Active close


Client Process
Server Process
ESTABLISHED
ESTABLISHED
Data Transfer
FIN
ACK
FIN-WAIT-1
Data Transfer
Acknowledges
FIN-WAIT-2
CLOSE-WAIT
2MSL timer
FIN
ACK
Time-out
TIME-WAIT
LAST-ACK
CLOSED
CLOSED
Client States
Server State

State	*Description*
CLOSED	There is no connection
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 have decided to close simultaneously

User Datagram Protocol (UDP)

The UDP operation is same as TCP with the following differences

- UDP does not have a connection establishment process
- UDP does not have a connection termination process
- UDP does not have error control, flow control and congestion control mechanisms
- UDP header has only 8 bytes

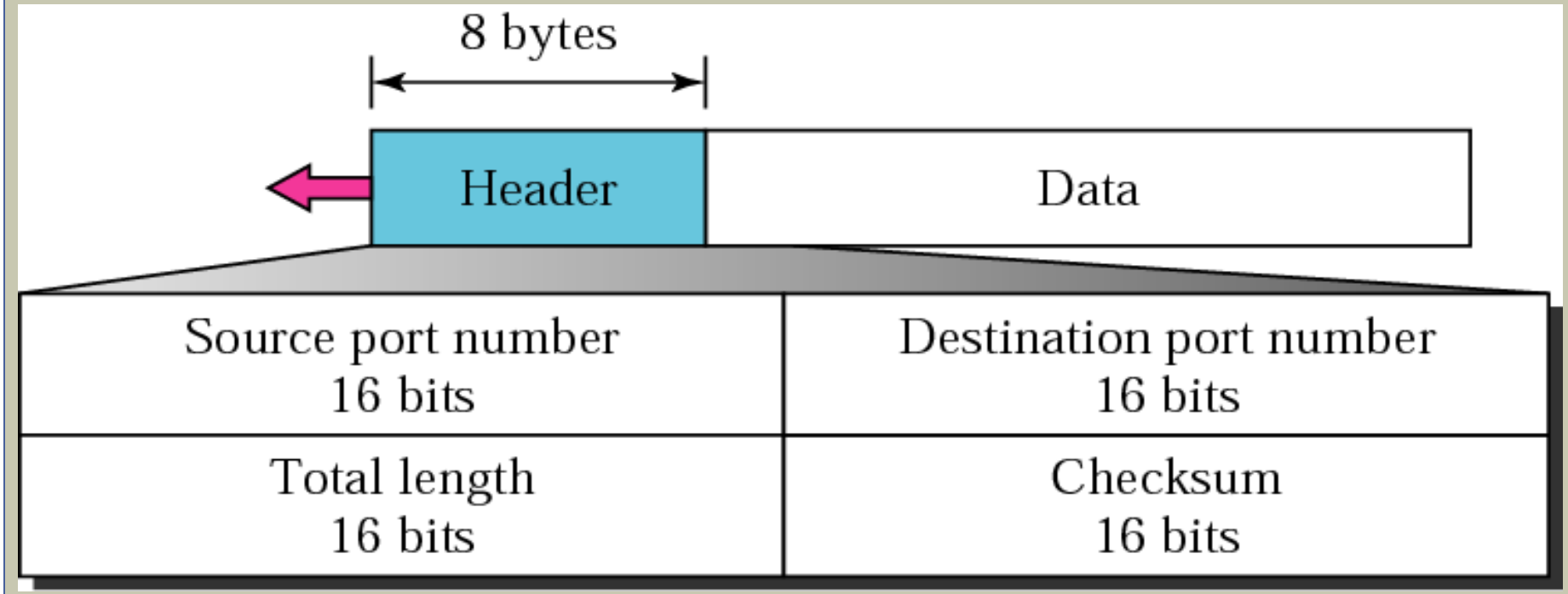
UDP cont.

- Since UDP does not get any feedback from the receiver, there is no guarantee of delivering data to the receiver by UDP
- Therefore UDP is an unreliable simple protocol
- Because of its simplicity it is used for specific applications especially the broadcast type applications

UDP Port numbers

Port Number	Application
69	TFTP
53	DNS
161	SNMP
520	RIP

UDP Header


8 bytes
Header
Data
Source port number
16 bits
Destination port number
16 bits
Total length
16 bits
Checksum
16 bits