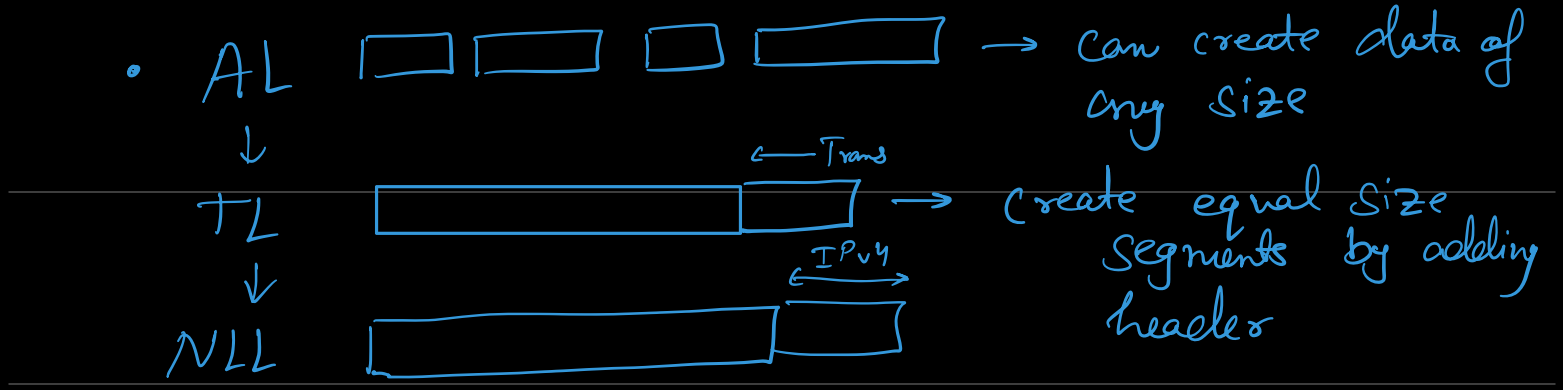


- TCP → connection oriented Application.

- Segmentation

- Transport Layer deals with port adresses.

TCP

Source port address 16 bits								Destination port addr. 16 bits
Sequence number 32 bits								
Acknowledgement number 32 bits.								
HLEN 4 bits	Reserved 6 bits	URG	ACK	PSH	RST	SYN	FIN	Window Size 16-bit
Checksum (16 bits)								Urgent Pointer (16 bits)
Options and padding.								

Synchronisation flag.

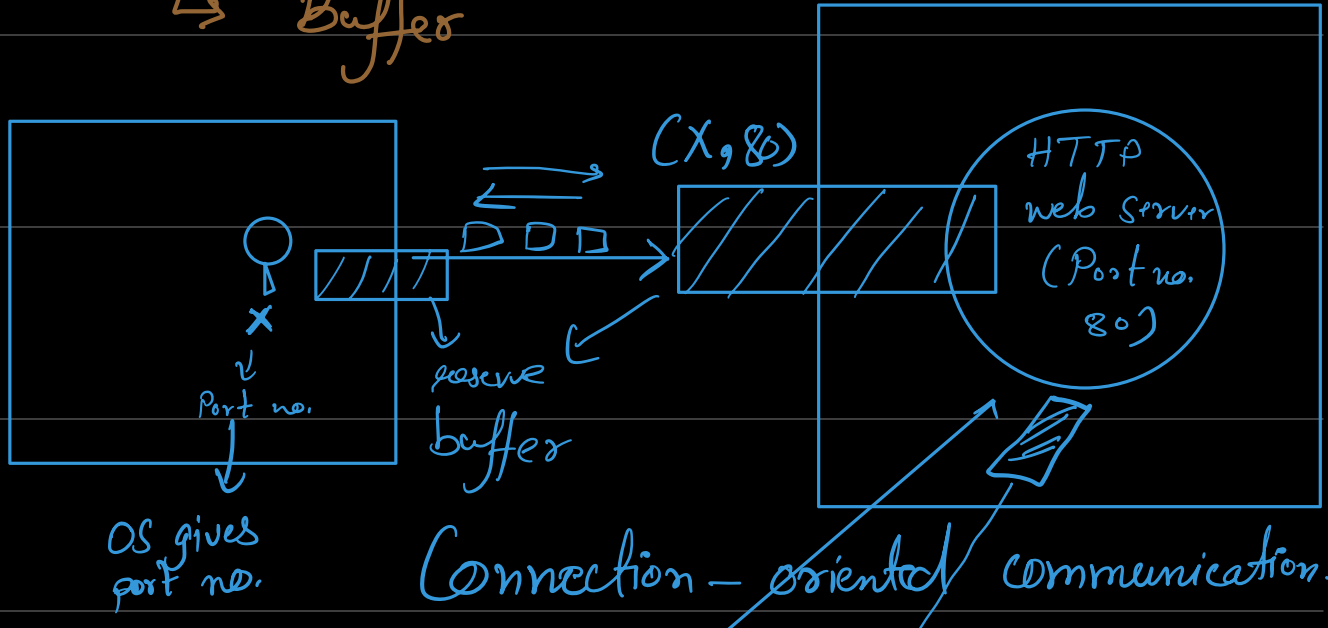
→ $0 - 2^{16} - 1$ port addresses available

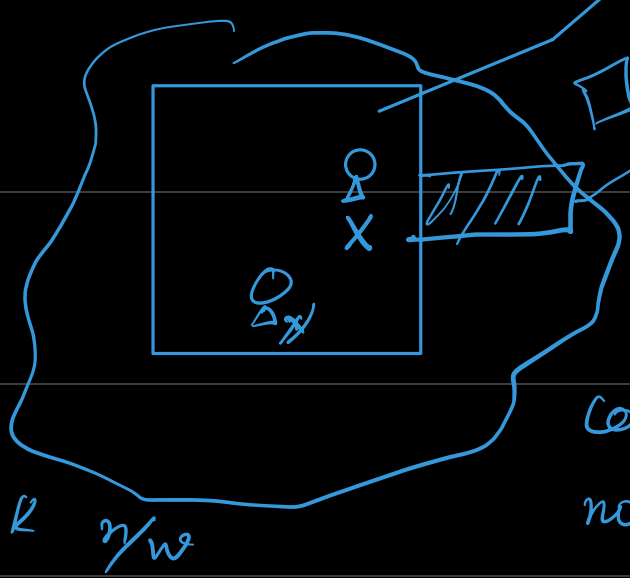
$0 - 65535$

↳ 0 - 1023 (well known port addresses)

- 1024 – 49151 (Registered Port Addresses)
- 49152 – 65535
  ↳ General purpose

- reserve resources
  ↳ Bandwidth
  ↳ CPU cycle
  ↳ Buffer





- Same port no can create confusion at the post no. 80.
- IP address can solve this issue.

→ When two processes from a single client can't be uniquely identified by IP only.

- Sockets are Combination of Port and IP Address.

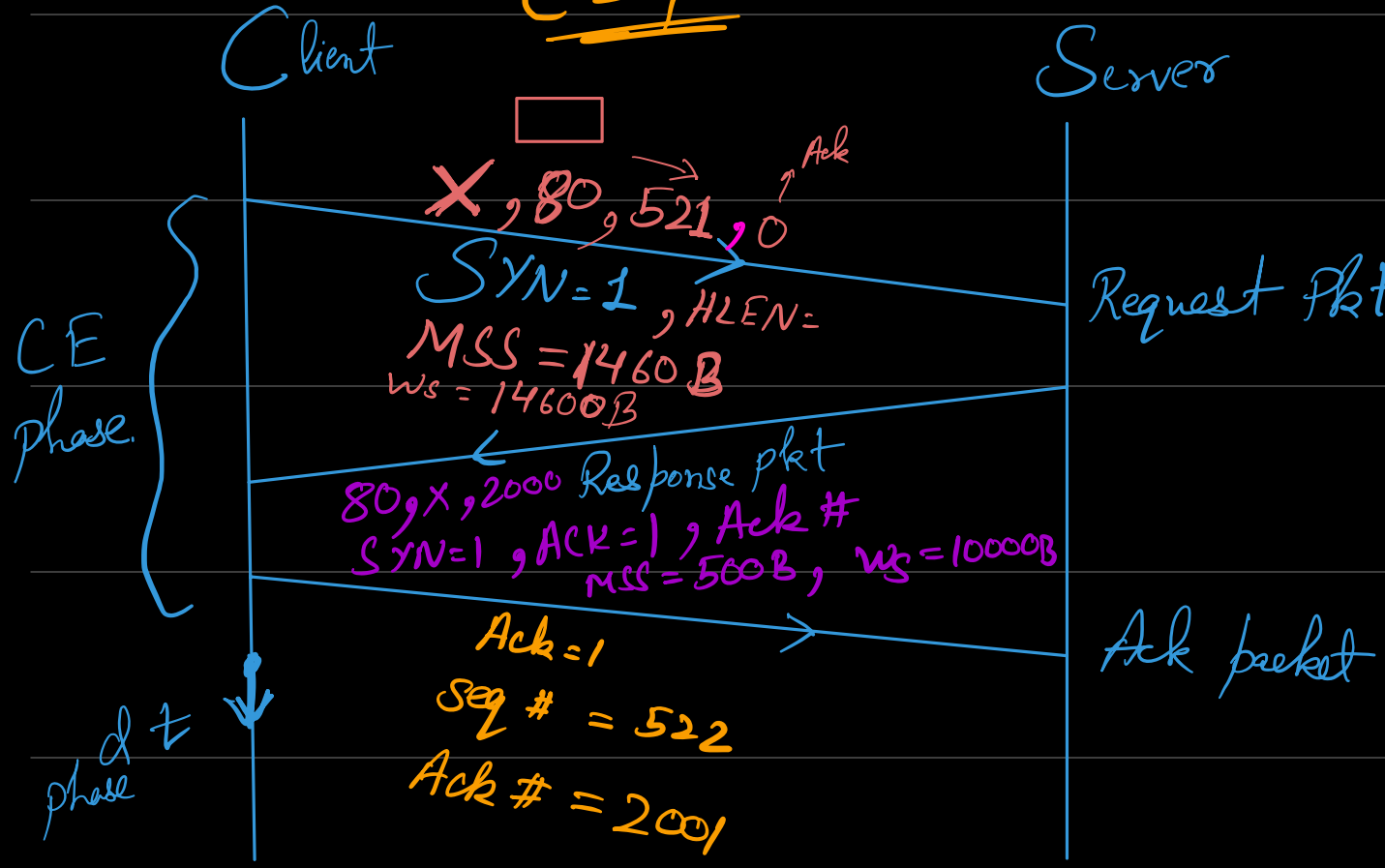
48 bit → 16 bit Port

32 bit → IPv4

- TCP Connection Establish →

① Connection establish phase
② Data transfer phase
③ Connection Termination phase.

CE phase



- Minm TCP header Size = 20 Bytes.

Max " " " = 60 Bytes
[40 Bytes options]

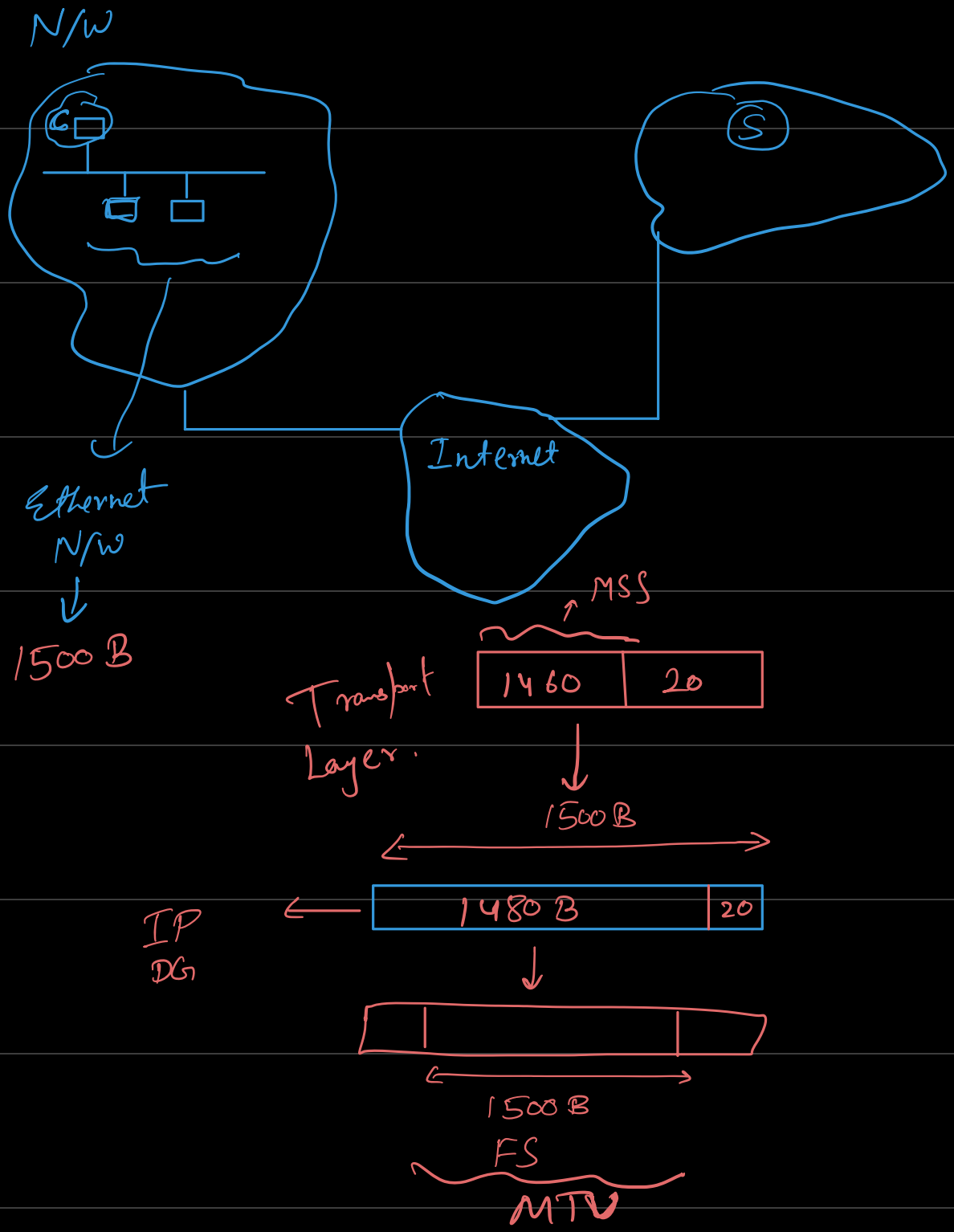
- Sequence Number → Start randomly from

[0 to $2^{32}-1$]

→ SYN is 1 for 1st packet.

→ MSS → maximum Segment Size
(↓ options)

→ Window Size



- **Window Size →**

Total receivable capacity = 14600B

Without Ack

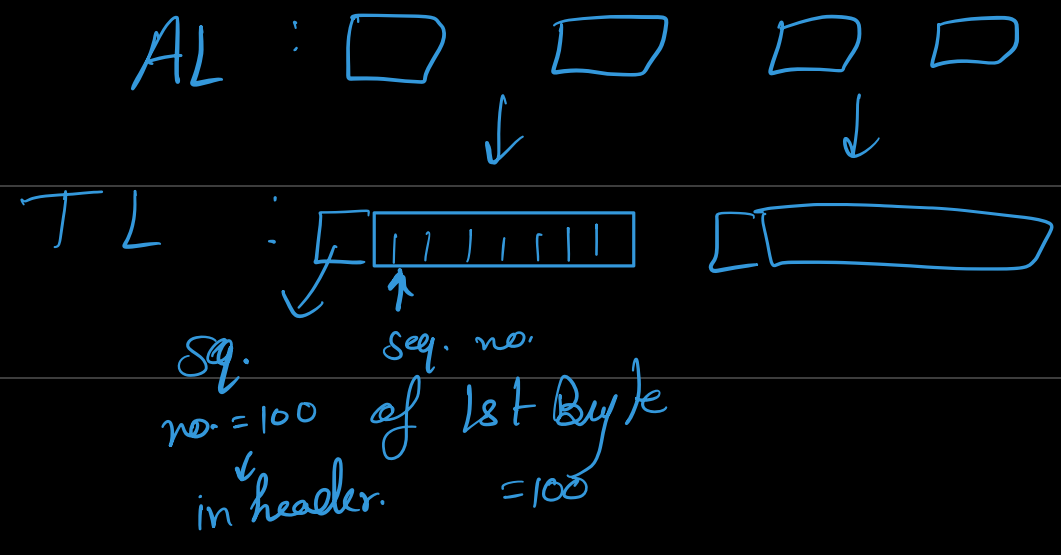
Server can send = 10 packets

- **Acknowledgement number →**

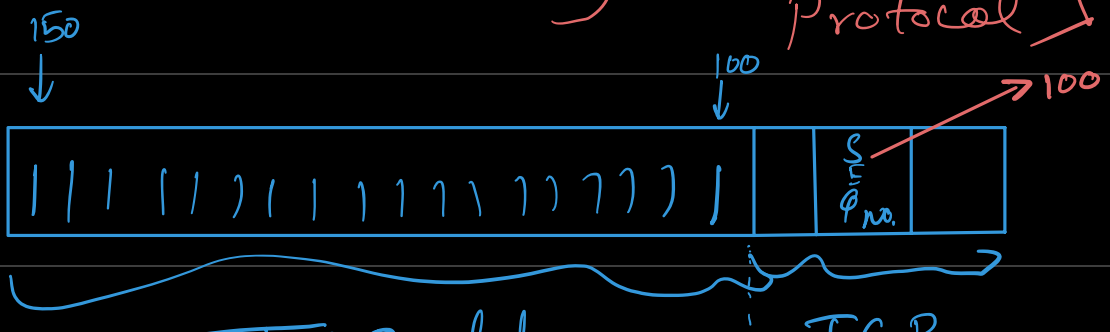
Ack # = last byte seq. # + 1

- **TCP is a Byte Ordering Protocol**

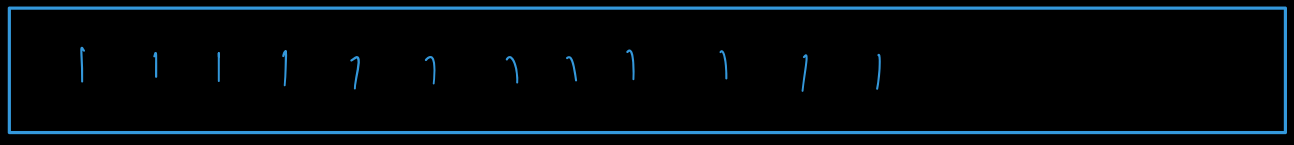
→ Every Byte is going to be counted.



[Byte Stream Protocol]

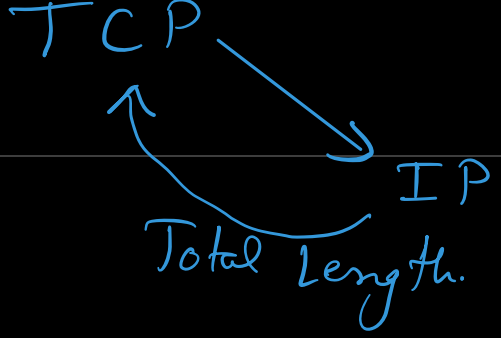


Reciever Side



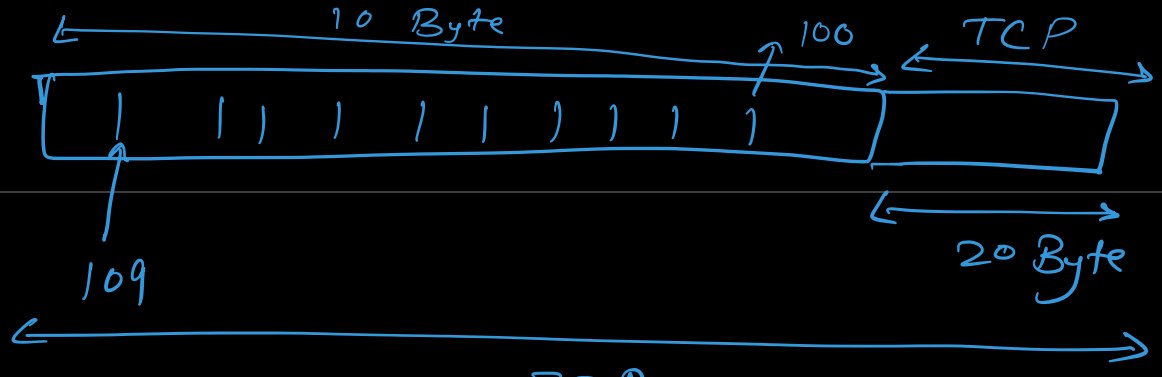
Ack # = last byte seq # + 1

For finding the Last byte seq. no. reciever will take help from Network Layer (particularly IP)



Ex°

Sender



receiver gets this segment

Ack# = 110

Reciever

$IP_{Total\ Length}$: 50 B

IP_{HL} : 20 B

$IP_{BL} = TCP_{Total\ Length} = 30\ B$

$TCP_{BL} = TCP_{TL} - TCP_{HL}$

$= 10\ B.$

Last Byte Seq # = (TCB_{BL} + First Byte Seq #) − 1

Q $IP_{TL} = 1000\ B$

$IP_{HL} = 5$

TCP segment is there in IP.

First Byte seq no. is 100.

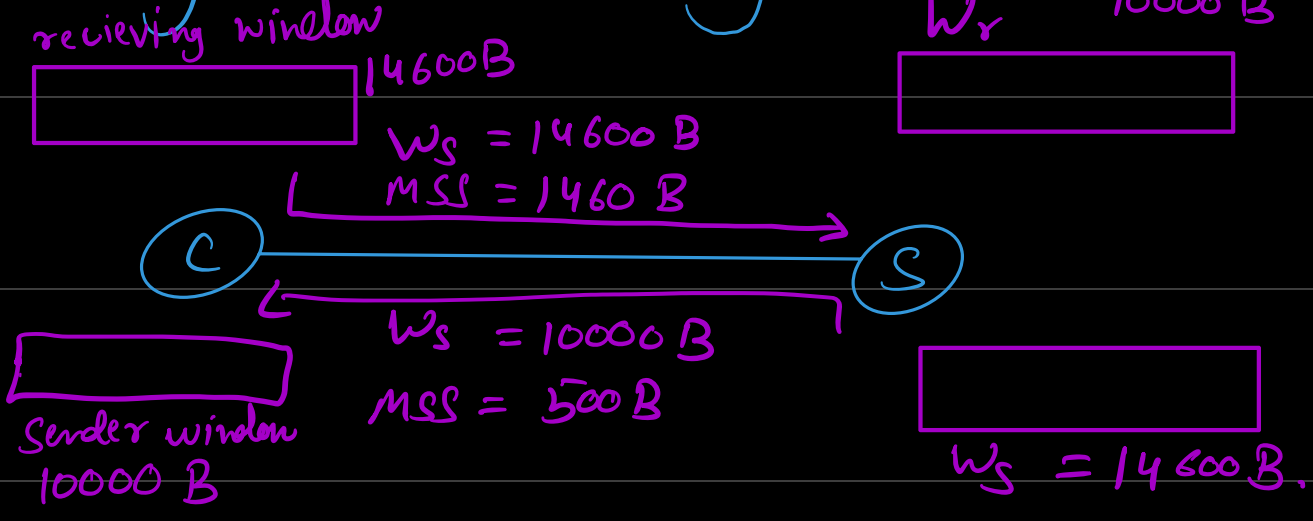
Find Ack no.

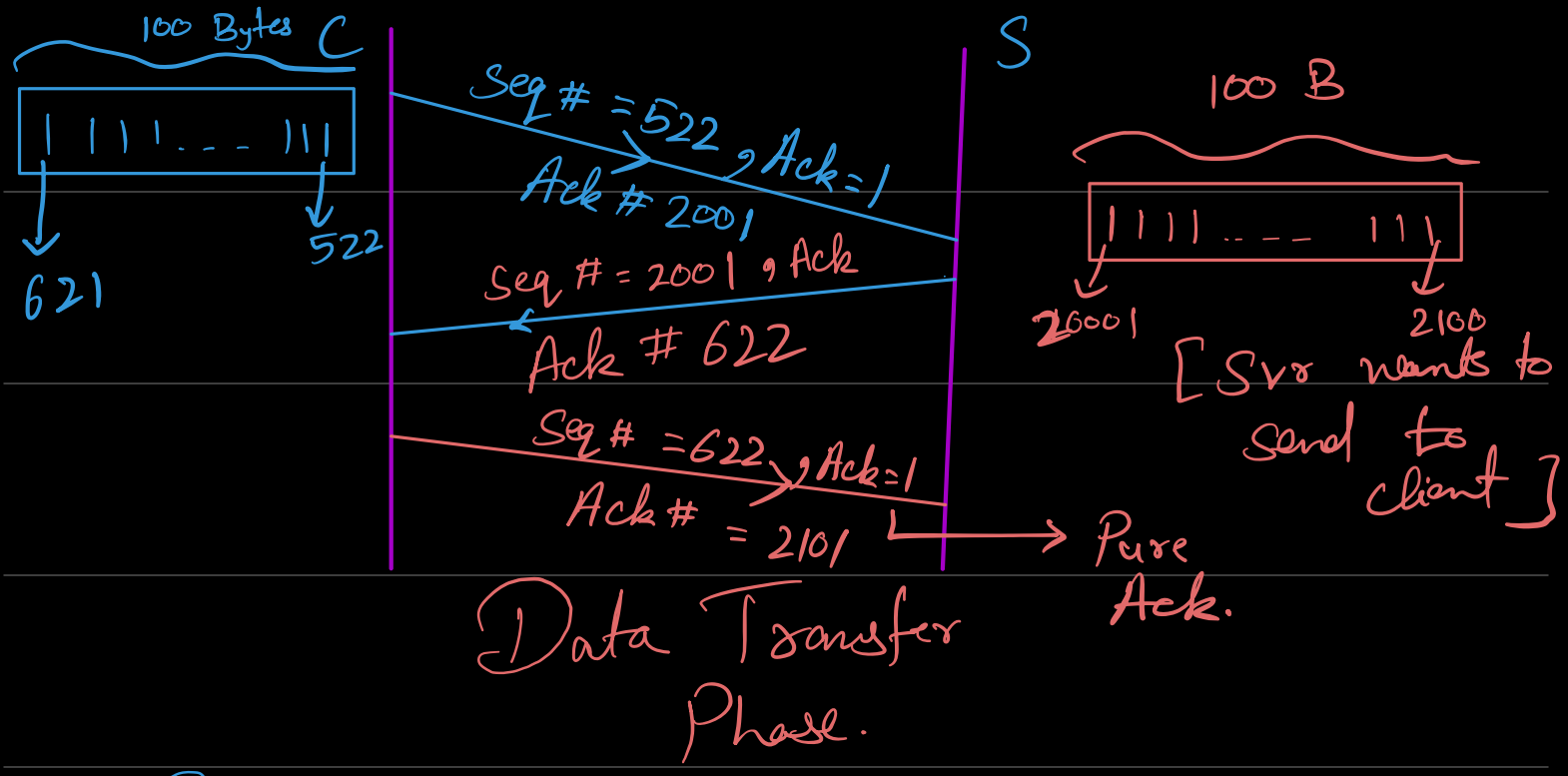
$TCP_{TL} = 980$

$PayLoad_{TCP} = 960$

Ack # = 960 + 100 = 1060

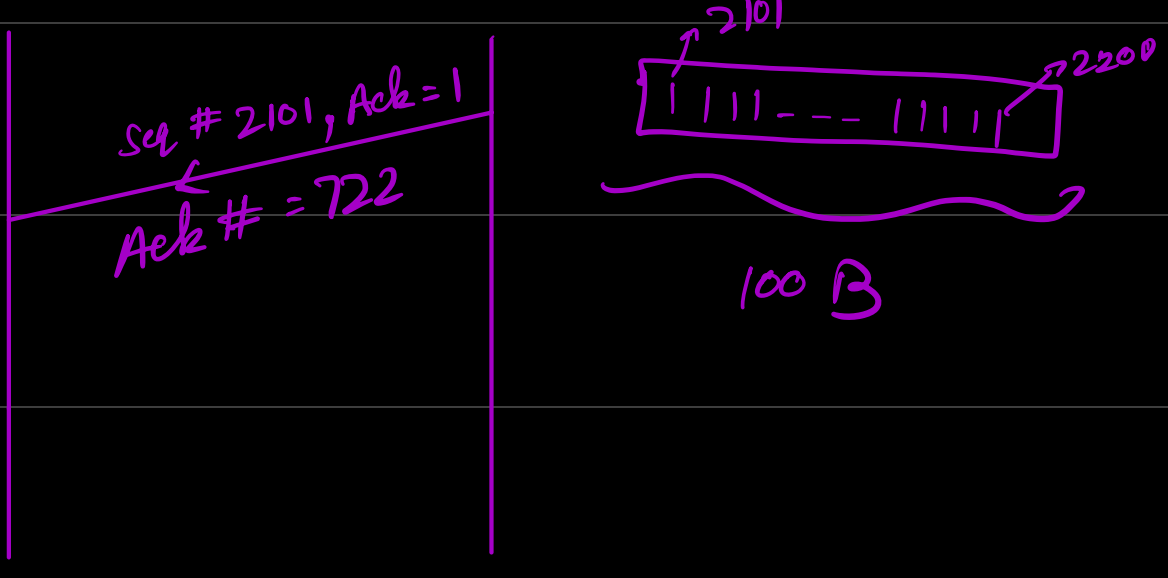
After establishing Connection





Data Transfer Phase.

Rule → Pure Ack. ka liye jo seq. no. use kiye h! we can reuse it.



Connection Termination

FIN = 1.

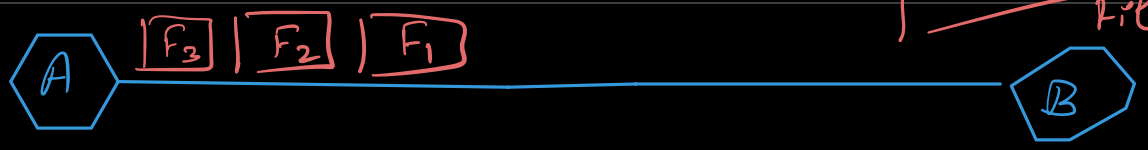
Refer to diagram on Classroom!

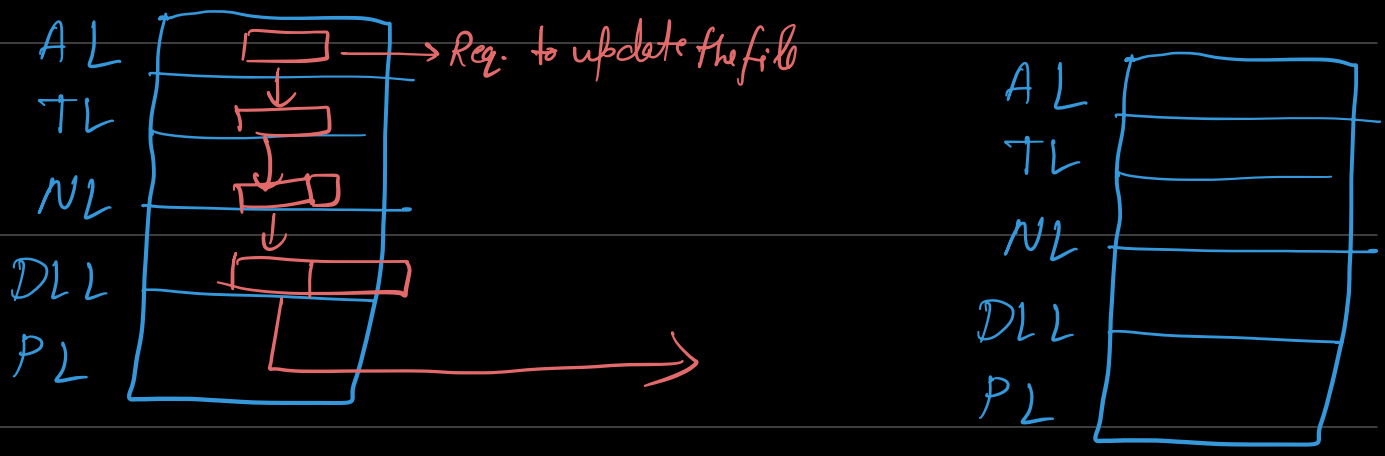
	SYN	Ack
First Pkt	1	1
Second pkt	1	1
Third pkt onwards	0	1
	0	1
	0	1

- **PSH Flag** → If PSH = 1 then AL is telling TCP to send the data without waiting for MSS.

- URG → [Urgent]

- Urgent Pointer





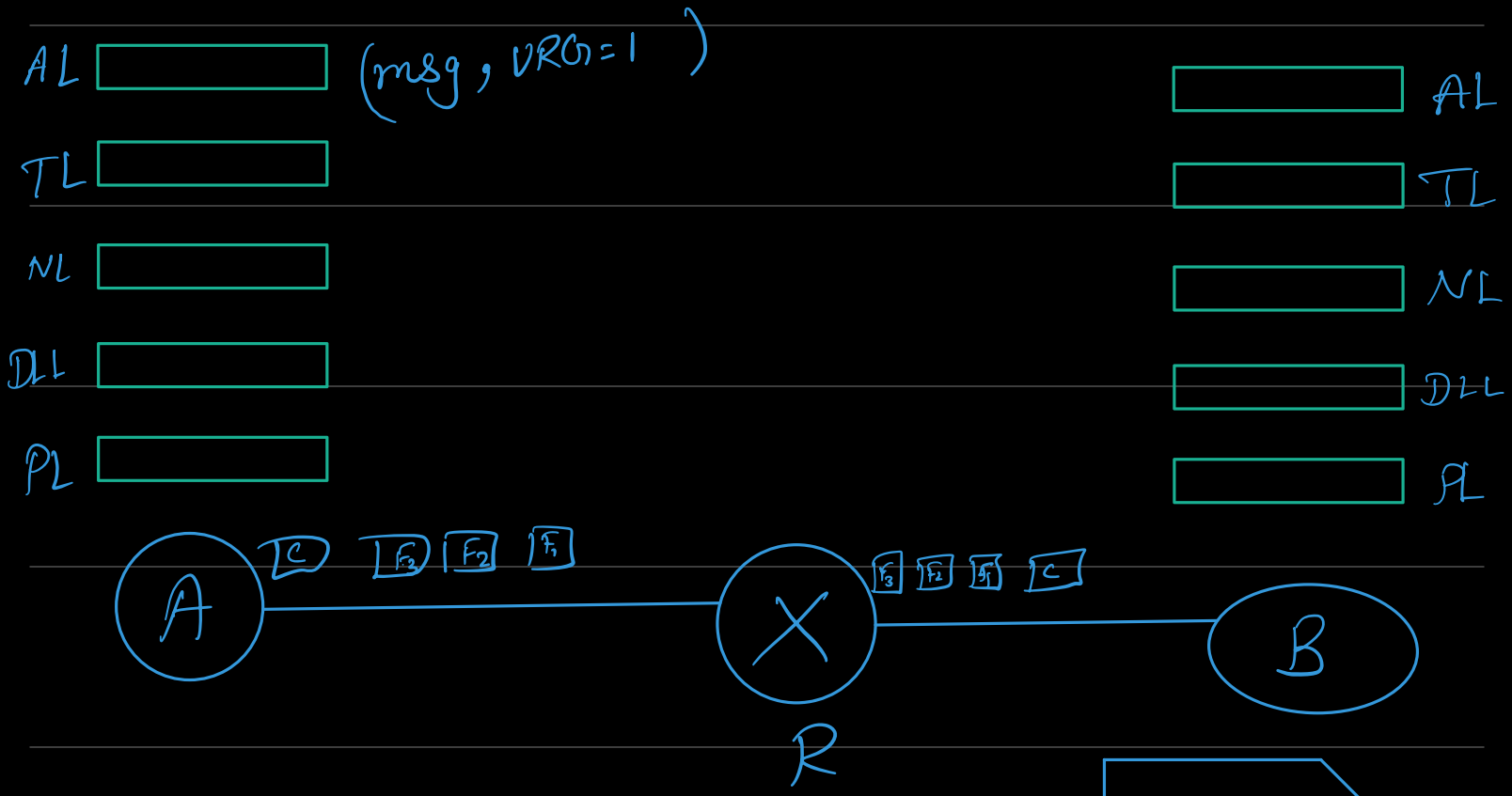
If we don't want the changes to happen due to some reason but the request is already sent.

Cntrl + C in TELNET breaks the connection.

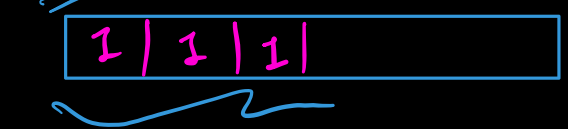
So it will drop the packet.

But it is in last. So urgent flag is used in this

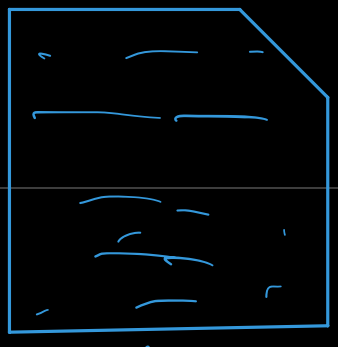
If URG is 1 then there will be Something in the Urgent pointer.



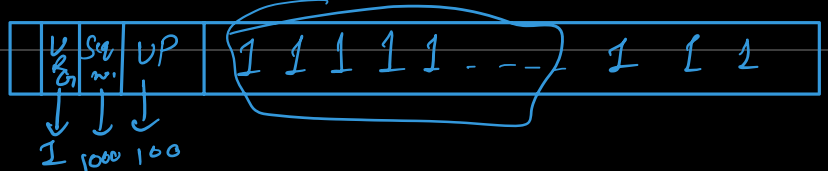
Cntrl + C



Priority in IP header.

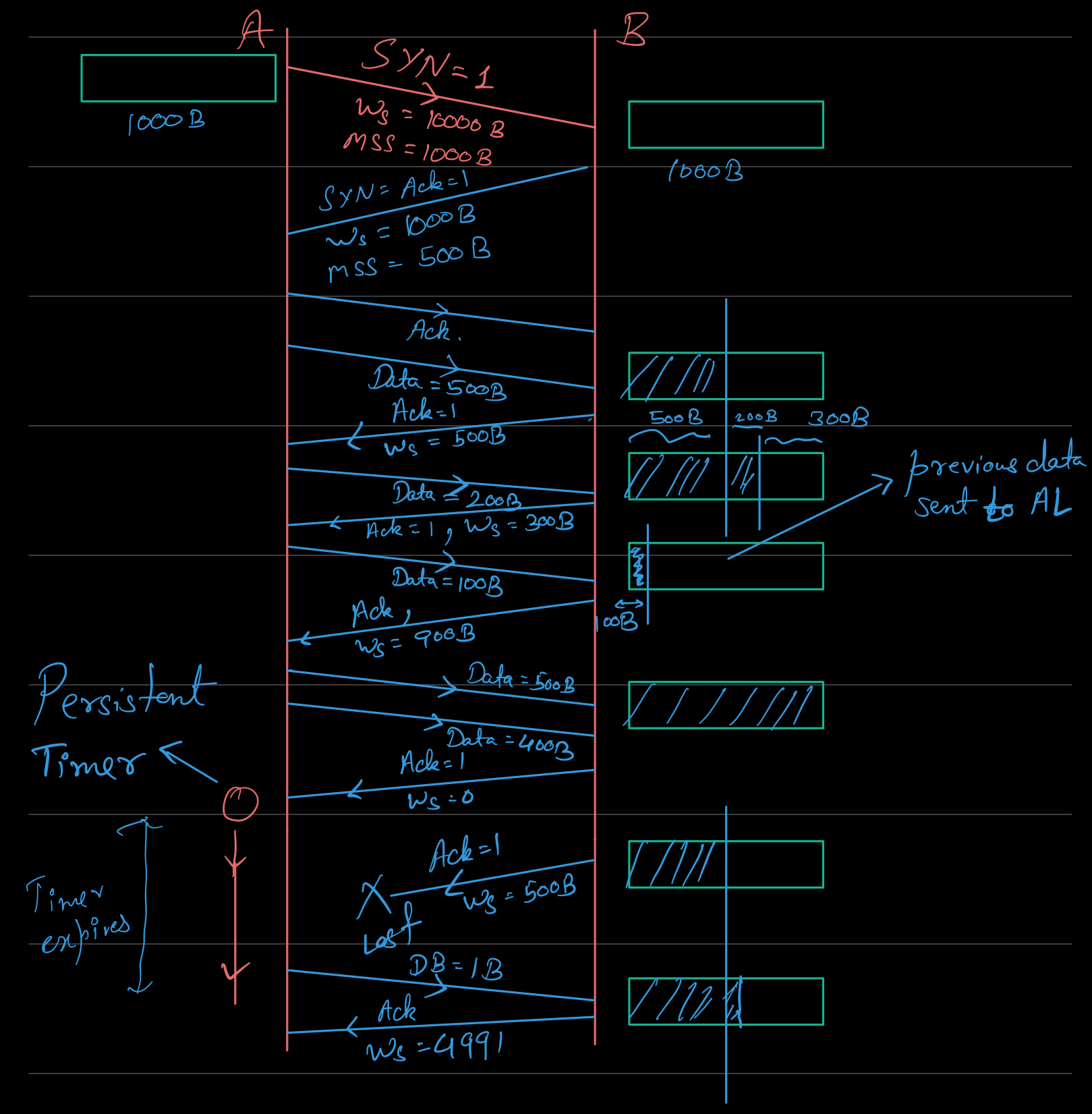


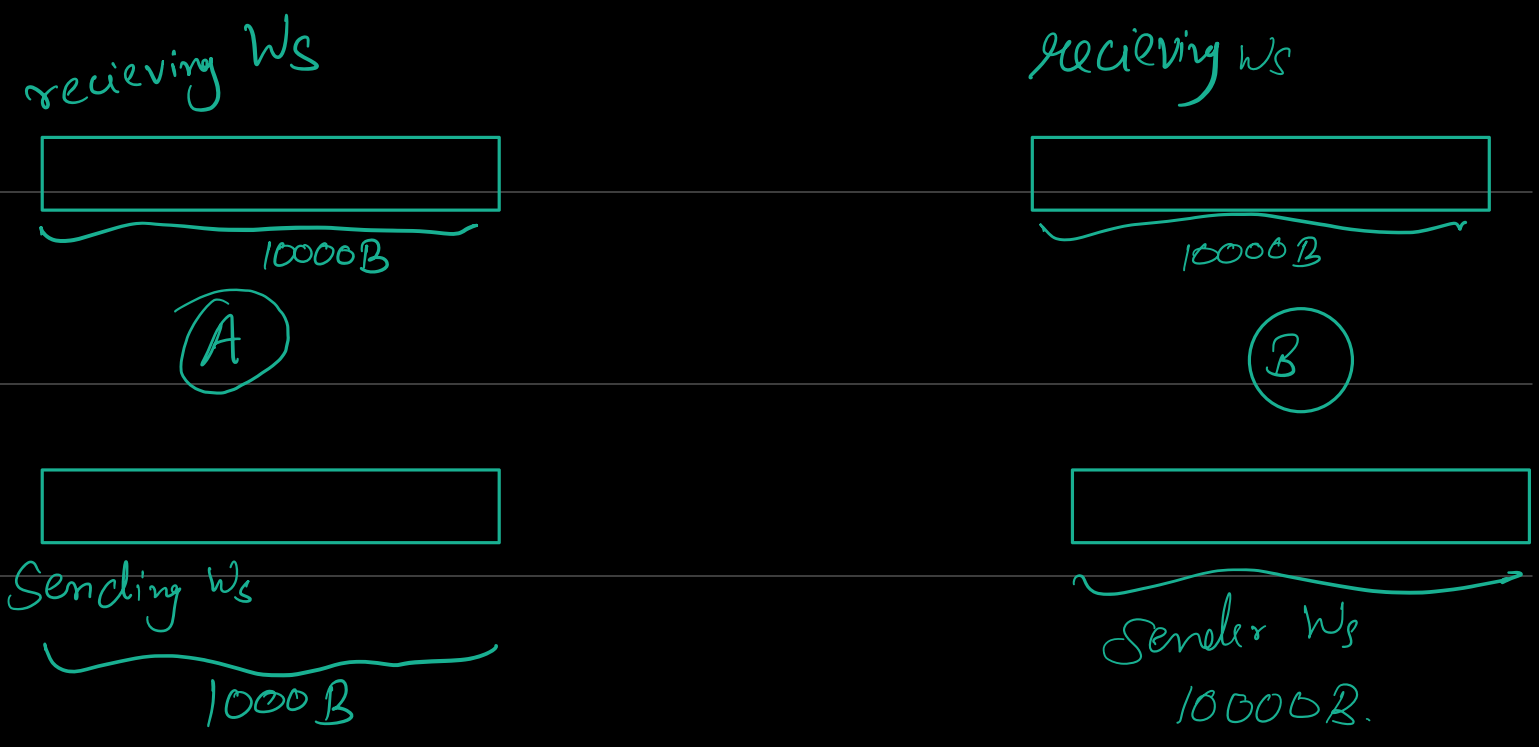
- Urgent pointer



1000 + 100 = 1100 Urgent bits

• Flow Control in TCP →





- You can take away 14 bit extra from options for defining the window size. [Window extension Option]

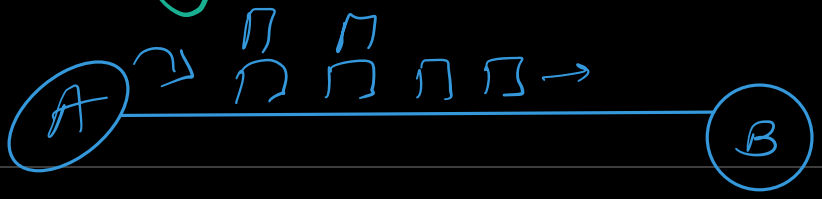
- Parameter negotiation Option → For defining MSS we are using this.

- Timestamp Option → We can inc. the bits of seq. no. using options.

- Padding → To make TCP header a multiple of 4 we take

extra bytes from options field.
[Scale effect]

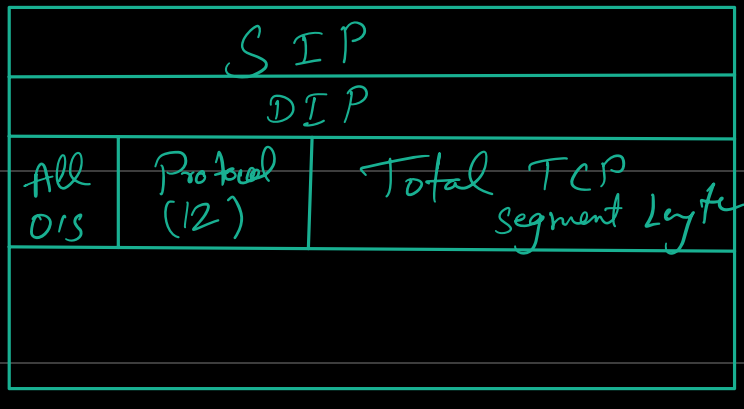
- **RST Flag** →



If reboot happens due to some problems.

after reboot
B will send pkt with RST = 1.

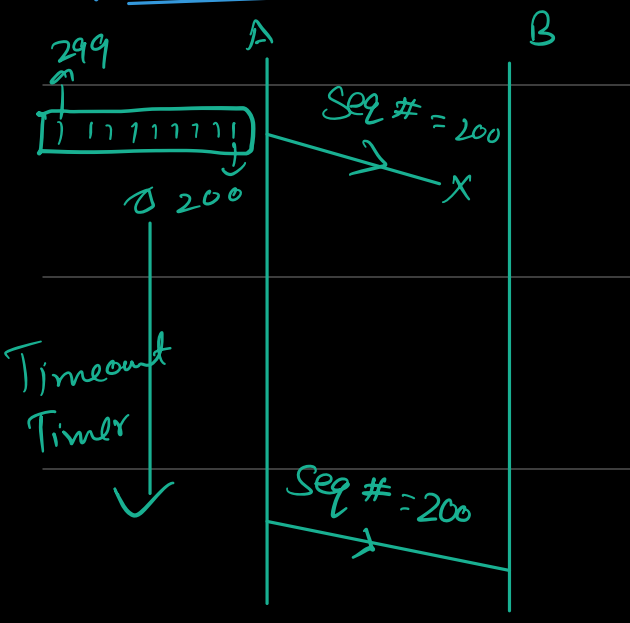
- **TCP Checksum** → Will be calculated on



IP Pseudo header

↳ TCP_{header}
↳ TCP_{Data}
↳ IP (Pseudo header)
↓
Incomplete header.

• Packet Process in TCP →



• Packet Loss [TCP]

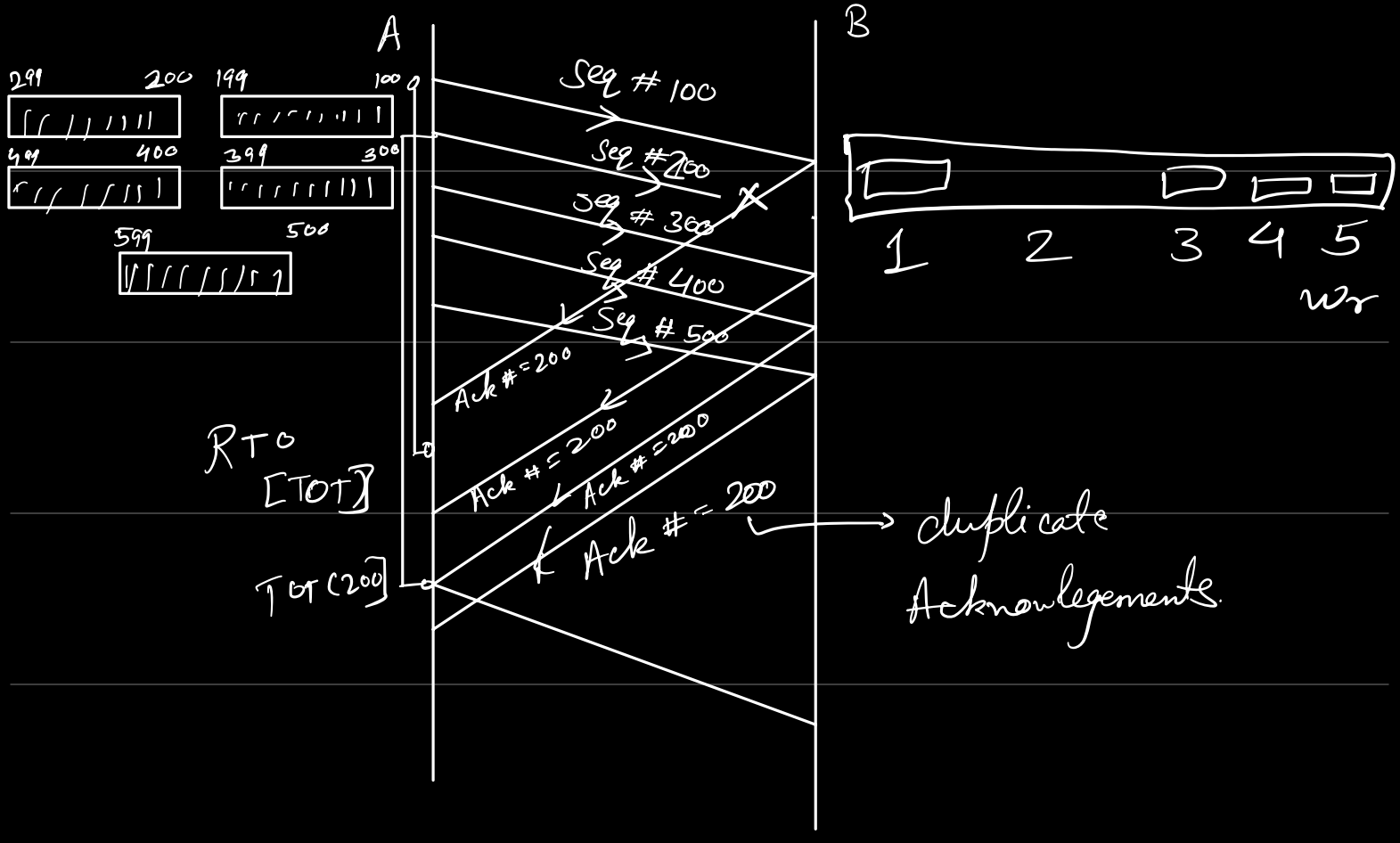
1) Segment # 1 : Seq. no. 100

2) Segment # 2 : Seq. no. 200

3) Segment # 3 : Seq no. 300

4) Segment # 4 : Seq. no. 400

5) Segment # 5 : Seq. no. 500



- Three Duplicate Acknowledgements →

Not to wait for Tot just retransmit

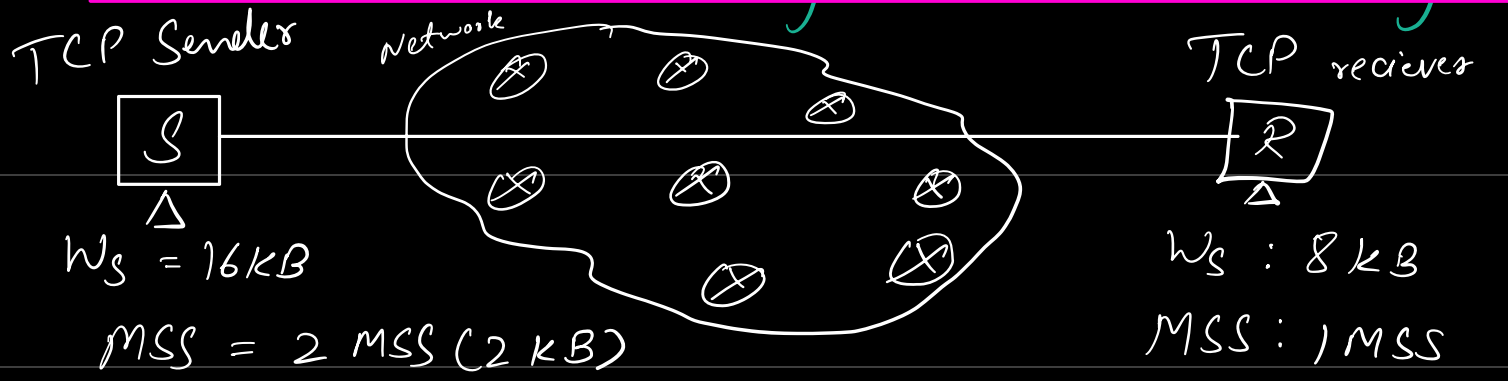
[Fast retransmission Algorithm]

- Network Congestion →

  → Backward Explicit Signaling

  Forward " "

- TCP CC [TCP congestion Control Algorithm]



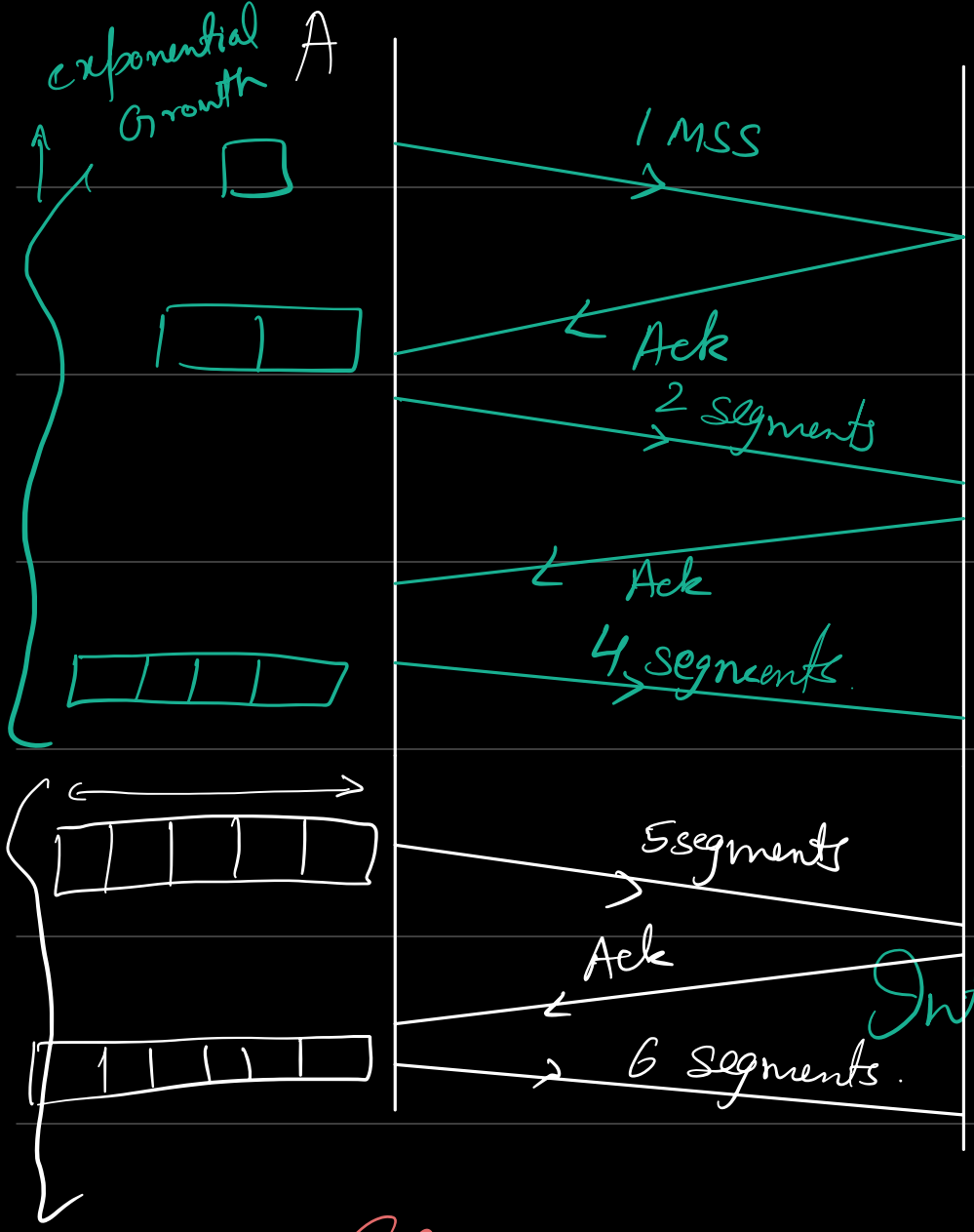
$$\min(16, 8) = 8$$

$$MSS\ (2, 1) = 1$$

You need to identify n_c → [network congestion]

No exact way to find n_c.

- We need to maintain congestion window to identify network capacity.



A → 16 KB

MSS → 2 KB

B → 8 KB

1 MSS (1 KB)

Actual window = min [cwd, Ws]

↓ congestion window

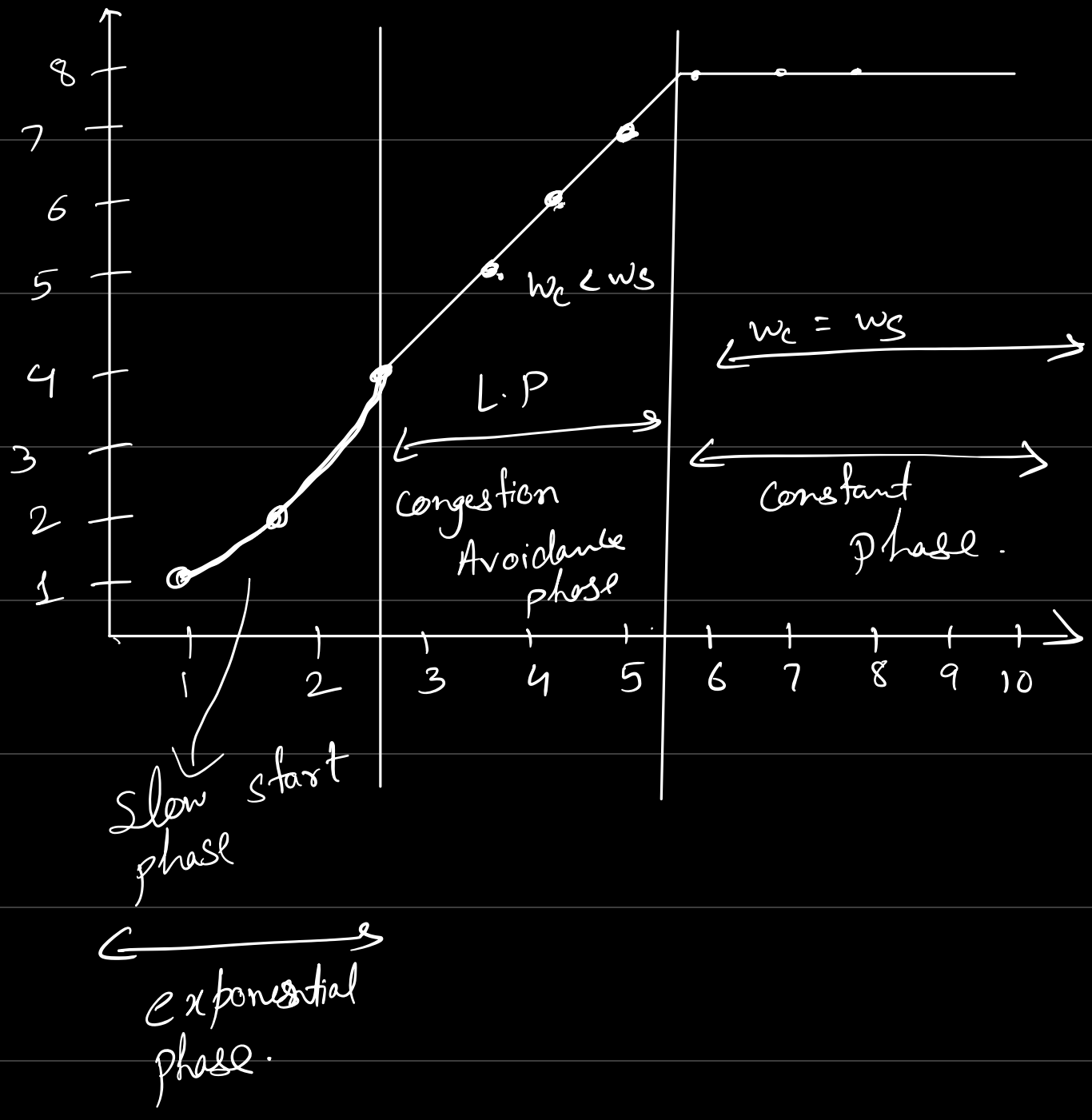
Initially cwd = 1.

↓ Linear phase.

$SS_{Thres} = \frac{Ws}{2} = \frac{8}{2} = 4$

↓ Slow start

Jab AW = 8 → Then constant phase.


8
7
6
5
4
3
2
1
1
2
3
4
5
6
7
8
9
10
$w_c < w_s$
$w_c = w_s$
L.P
congestion Avoidance phase
Constant Phase.
Slow start phase
exponential phase.

Example →

1) Agreed
WS: 64 KB

2) MSS : 1 KB

$SS_{Thresh} = 32$

$2^0, 2^1, 2^2, 2^3, 2^4, 2^5, 33, 34$

$2^0, 2^1, 2^2, 2^3, 2^4, 2^5$ — Exponential phase.

$33, 34$ — Congestion Avoidance phase

↑ Time out

$SS_{Thresh} = \frac{1}{2}$ curr WS

$= 1/2 \times 34 = 17$

Linear phase to E.P.

$2^0, 2^1, 2^2, 2^3, 2^4, 17, 18, 19, 20$

$2^0, 2^1, 2^2, 2^3, 2^4$ — EP/SSP

$17, 18, 19$ — CA or LP

20 — 3 dup Acks.

$SS_{Th} = \frac{1}{2}$ current w.s

$= 1/2 \times 20 = 10$

Still remain in congestion Avoidence Phase.

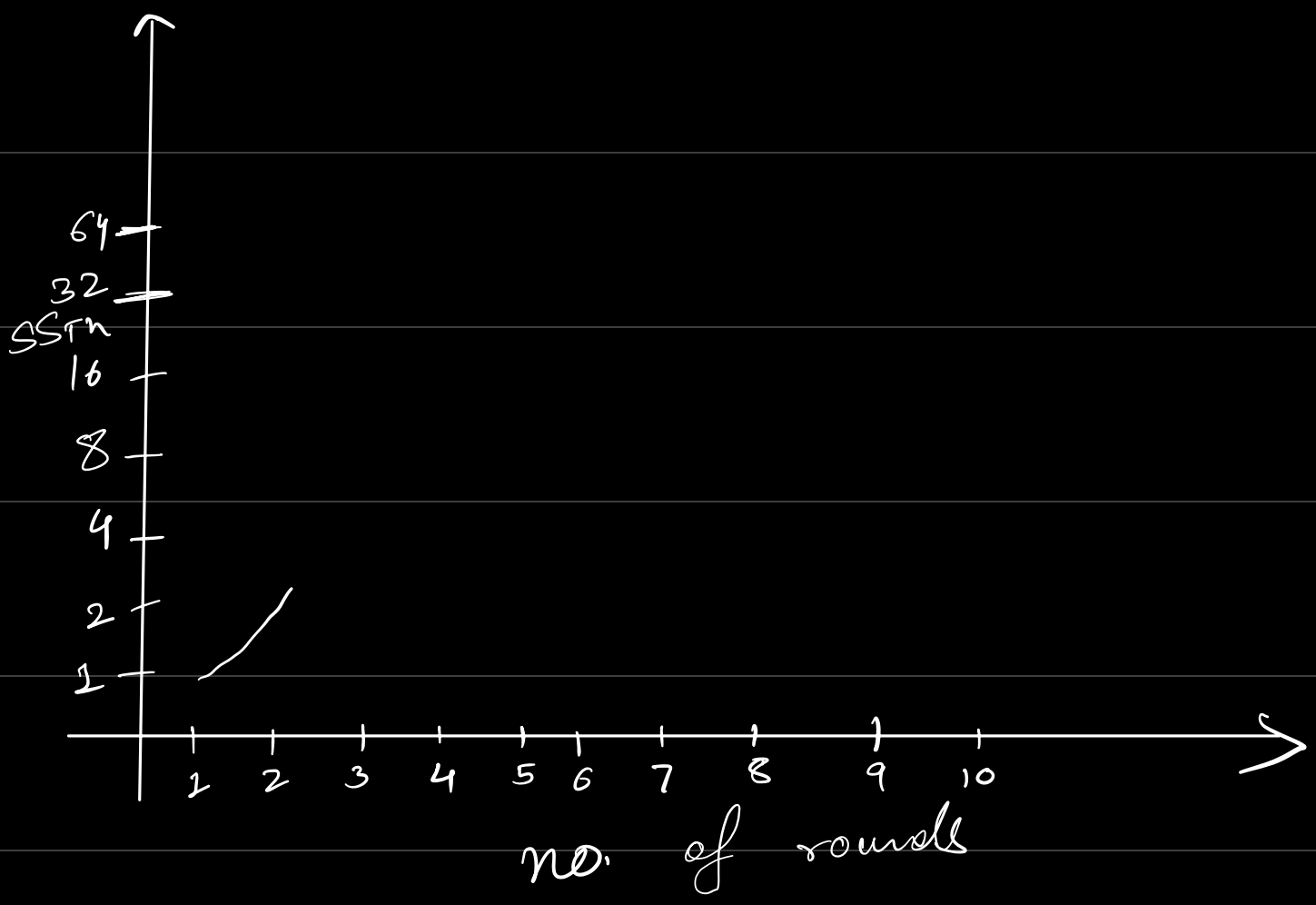
9 10, 11, 12, ↑ , 2^0, 2^1, 2^2, 5, 6, 7, 8, ↑

To

$SS_{Th} = 6$

3 dup Ack

4, 5, 6 ---- 64



Quality of Services

- Traffic Shaping
  - Token Bucket
  - Leaky Bucket
- Scheduling.