Visit : www.csepracticals.com
Owned by : CSEPracticals

1. Understand Services offered by TCP
2. TCP State Transition Diagram and 3-Way handshake
3. What is meant by Connection Oriented
4. Flow and Congestion Control
5. TCP hdr segment format
6. TCP various Timers
7. Advanced Features : SACK, dupACK, Fast Recovery, Nagle Algorithm, Fast retransmit etc
8. TCP Windowing Mechanism
9. Understand Why TCP is designed the way it is today !!

Visit : csepracticals.com
Owned by : CSEPracticals

Getting started

Overview of OSI Model and TCP/IP Stack
Transport Layer Overview
Transport Layer Standardized Protocols – UDP/TCP
UDP Vs TCP

Visit : www.csepracticals.com
Owned by : CSEPracticals

OSI Model Basics

- TCP is a transport layer protocol, designed for reliable communication between processes
- Let us start with the basics and understand the transport layer and understand the picture at the broader level
- In this section of the course, we will understand how TCP as a protocol fits in the Networking TCP/IP stack (Implementation of OSI Model)
- Let us build the background first . . .

Visit : www.csepracticals.com
Owned by : CSEPracticals

- **OSI Model and TCP IP stack**

Theoretical OSI model

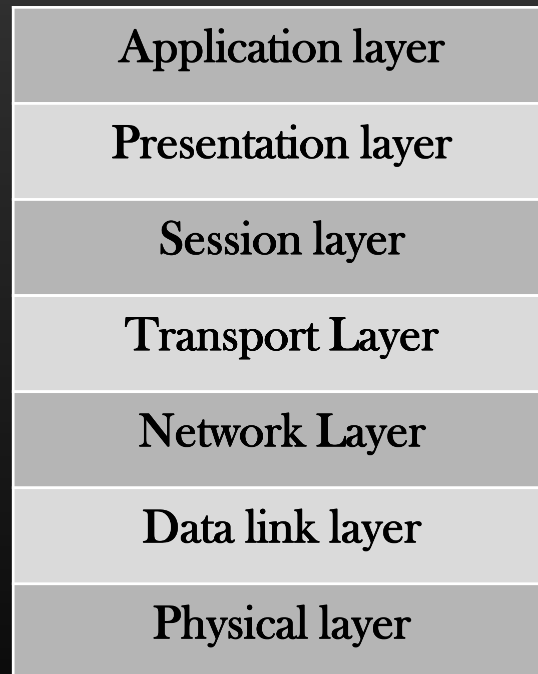
Application layer
Presentation layer
Session layer
Transport Layer
Network Layer
Data link layer
Physical layer

The Open Systems Interconnection model is a conceptual model that characterizes and standardizes the communication functions of a telecommunication or computing system without regard to its underlying internal structure and technology

Reference/Standard/Guideline . . .

- **OSI Model and TCP IP stack**

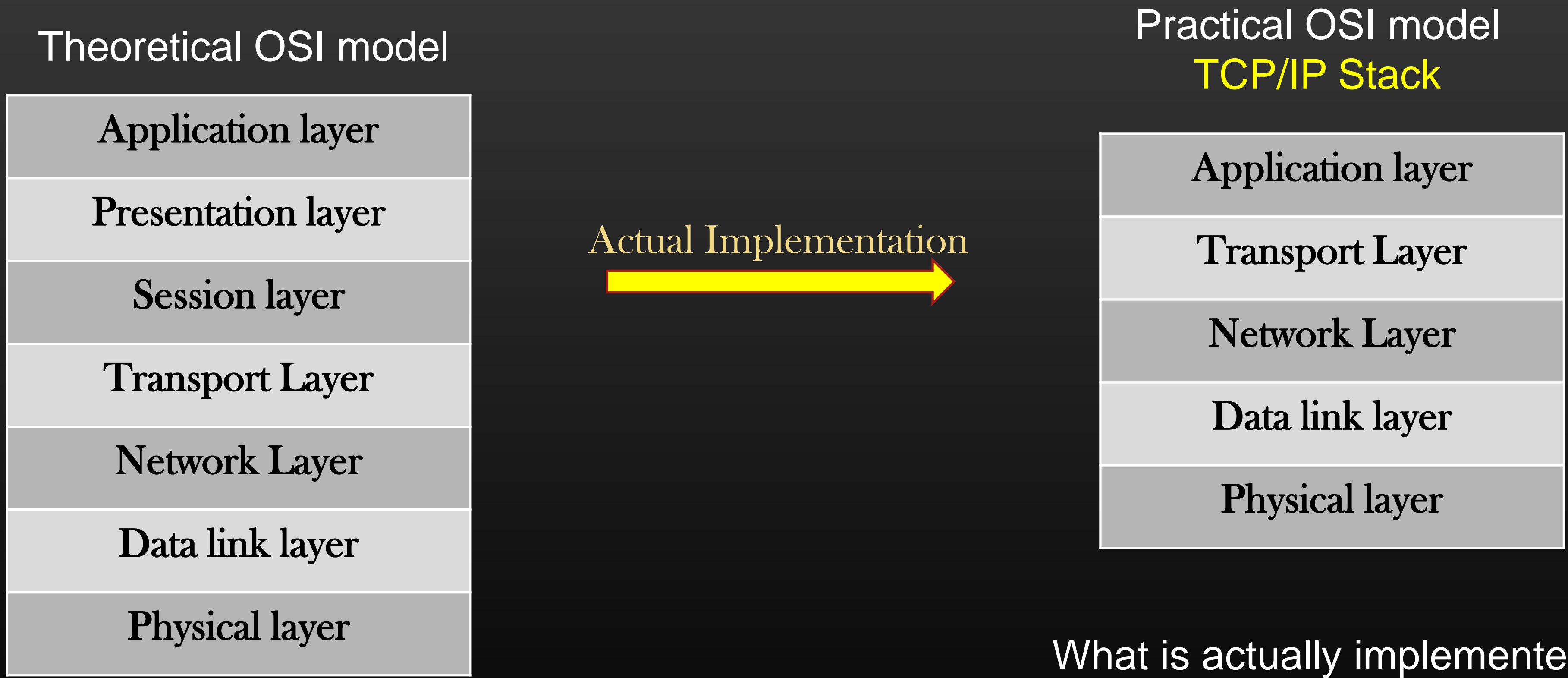
Theoretical OSI model



Reference/Standard/Guideline . . .

- Description of the Networking subsystem (network stack)
- It is a guideline . . .
- Layer – logically complete functionality of a networking component is referred to as a layer
- Each layer has a specific function
- Functions of layers do not overlap
- Data/packet moves across the layers bi-directionally
- All layers stack together to built a complete networking subsystem
- One most common example of OSI model implementation is TCP/IP network stack which runs inside your OS

- OSI Model and TCP IP stack



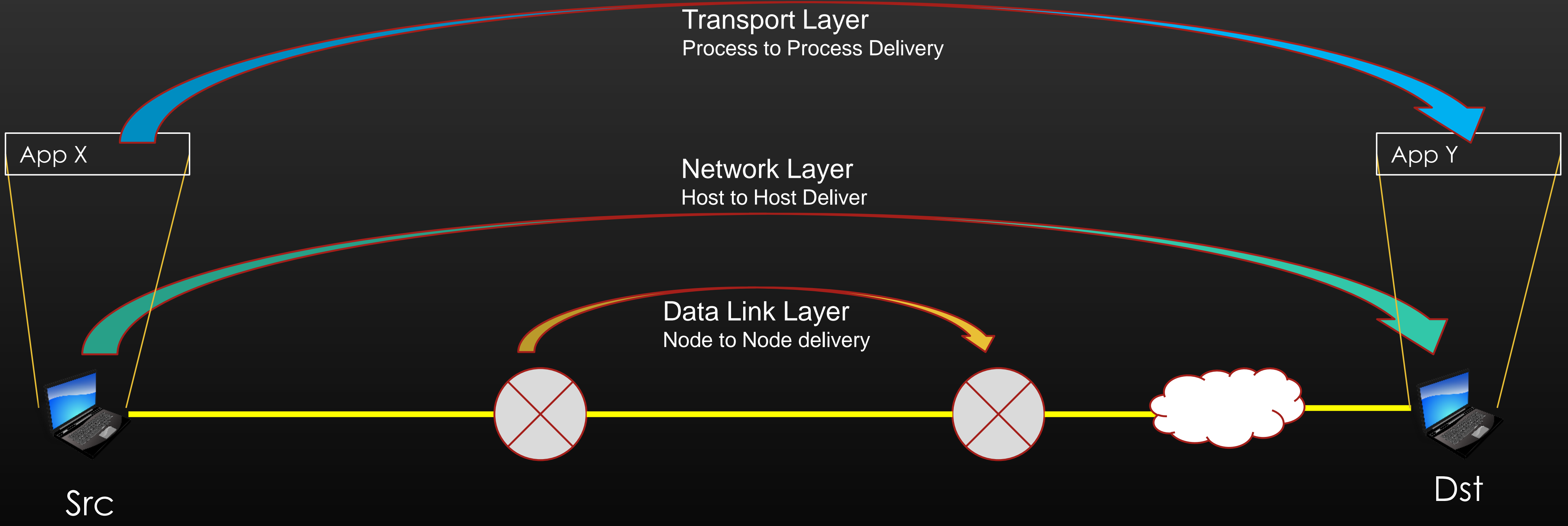
Reference/Standard/Guideline . . .

Pres. layer and session layer are partially Implemented in layers above it and below

OSI Model Basics

Application layer	Generic, implements networking application
Transport Layer	Process to Process Delivery
Network Layer	Source node to Destination node
Data link layer	From node to its adjacent node
Physical layer	Transmit data as electrical signals

- Isolated, Non-Overlapping Responsibilities

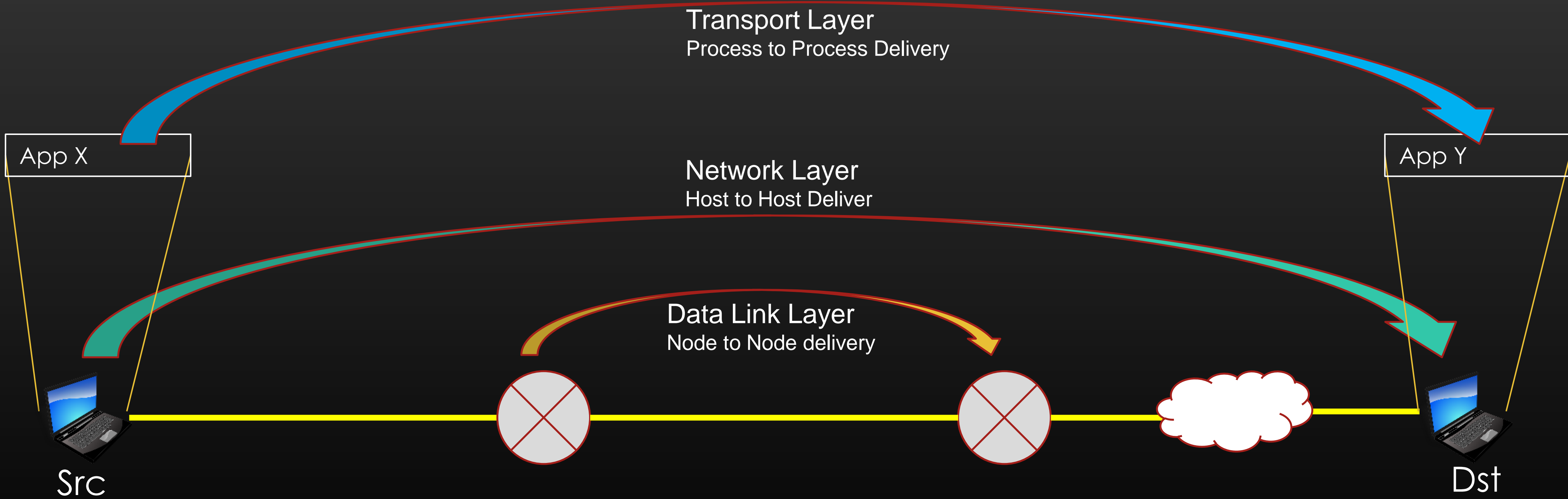


Visit : www.csepracticals.com
Owned by : CSEPracticals

OSI Model Basics

Application layer	Ping (ICMP), HTTP, WhatsApp, All mobile APPs etc
Transport Layer	UDP, TCP
Network Layer	IP, IPv6
Data link layer	Ethernet
Physical layer	

- Isolated, Non-Overlapping Responsibilities



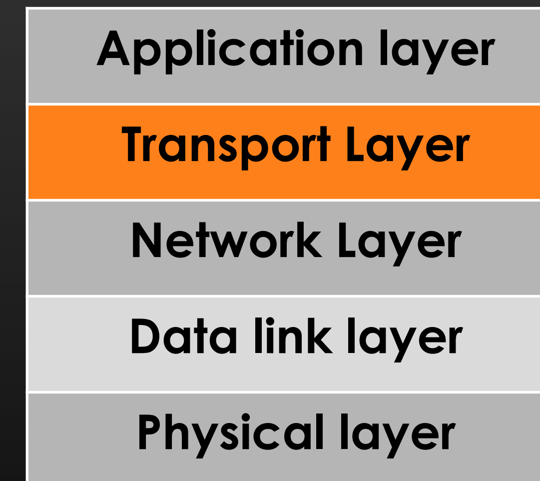
Visit : www.csepracticals.com
Owned by : CSEPracticals

Transport Layer

- **Table of Contents**

Transport Layer (also called Socket Layer)

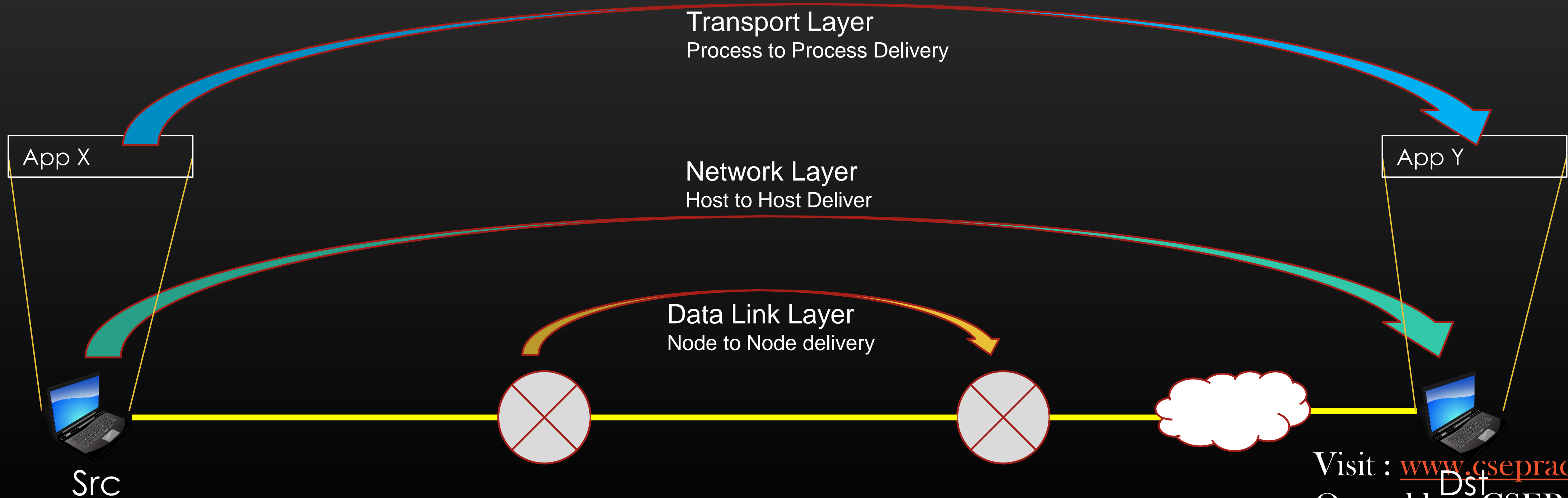
- Transport Layer Goals
- Transport Layer Protocols
  - UDP
  - TCP



Transport Layer Goals

- Goals :
  - Facilitate communication (data exchange) between applications running on different machines deployed in the Network
  - Transport layer provides two world-wide standardized famous protocols to achieve its goal :
    - User Datagram Protocol (UDP) protocol
    - Transmission Control Protocol (TCP)

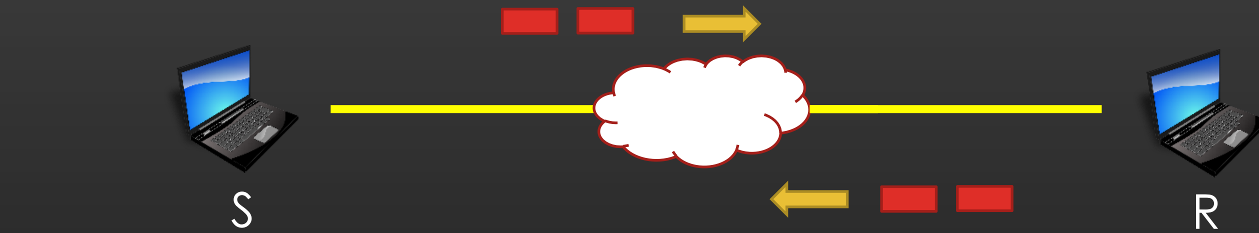
  - TCP and UDP most have the same end goal : Facilitate data exchange between processes, but they do it in a different way

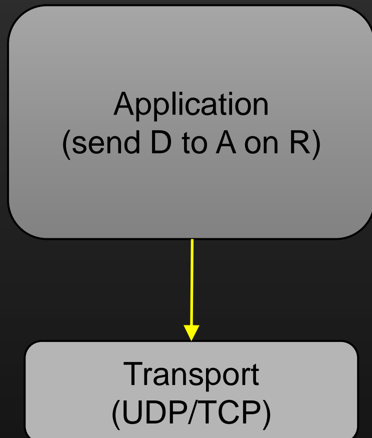


Visit : www.csepracticals.com
Owned by : CSEPracticals

User Datagram Protocol

- Very Simple and Straight-forward protocol for data exchange between process
- Work on *send and forget* Model
  - UDP protocol do not maintain any state of the peer it is communicating with





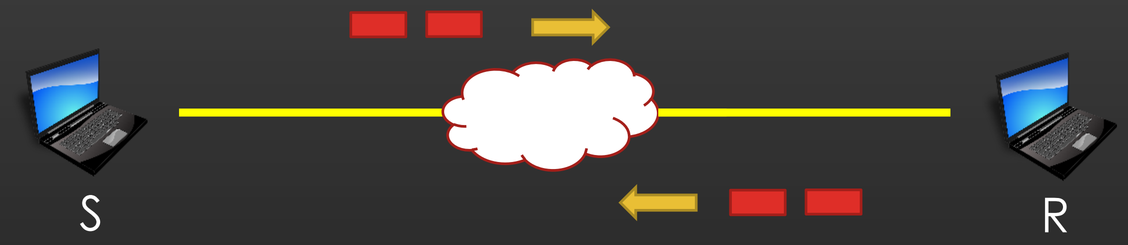
Connection-less

- Forget the recipient after sending data
- Not bother to say hello before sending Data

- UDP protocol do not remember who it was communicating with after sending data (Connection-less), and it also forgets that it has actually send any data (Stateless Protocol)
- UDP protocol sends data in chunks or discrete individual units called *datagrams*
- TCP protocol on the other hand is completely opposite to UDP – *Connection Oriented , Stateful and Byte Oriented*

Visit : www.csepracticals.com
Owned by : CSEPracticals

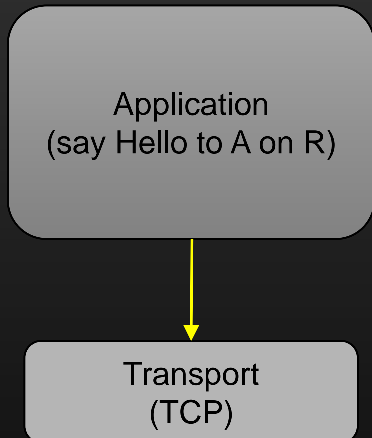
User Datagram Protocol



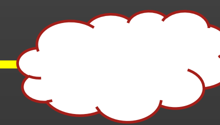
- Work on *send and forget* Model
  - *Unreliable delivery* : UDP don't care of the packet (datagram) has actually reached the destination or not
  - *Out of Order delivery :* UDP don't care if datagrams reaches the destination out of order !

Transmission Control Protocol

- Complex and result of research of over 20 yrs
- TCP is a *connection oriented protocol*
  - Mutually agree and know each other first



A = 100,
10.10.1.1



A = 10,
10.1.1.1

Connection request

Sender detail :
Port no = 100
Ip address = 10.10.1.1
Sender can process 5000B/sec

Connection Acknowledgement

Recv detail :
Port no = 80
Ip address = 10.1.1.1
Recvr can process 1000B/sec

Connection
Establishment

Visit : www.csepracticals.com
Owned by : CSEPracticals

Transmission Control Protocol

- TCP is a *stateful protocol*
  - Keep Track Of Data sent and recvd

A = 100,
10.10.1.1

S

A = 10,
10.1.1.1

R

Recv detail :
Port no = 80
Ip address = 10.1.1.1
Recvr can process 1000B/sec

Connection
Established

Sender detail :
Port no = 100
Ip address = 10.10.1.1
Sender can process 5000B/sec

Application
(say Hello to A on R)

Transport
(TCP)

100B sent, 500B bytes next
200B recvd
50th B resend

Data Housekeeping

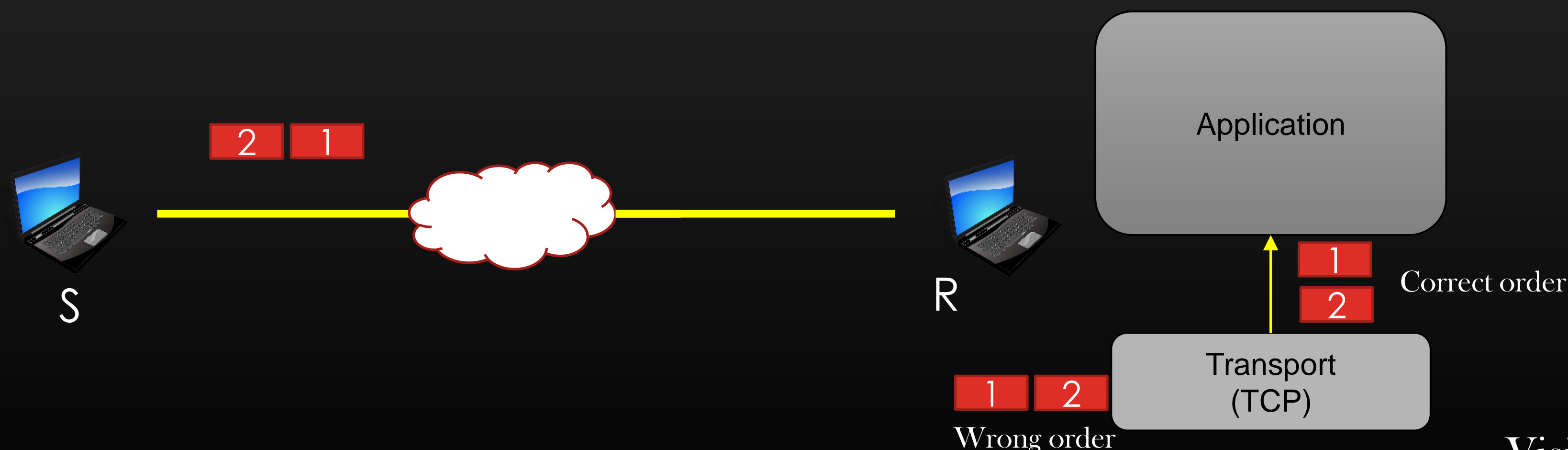
100B sent, 500B bytes next
200B recvd
50th B not recvd

Visit : www.csepracticals.com
Owned by : CSEPracticals

Transmission Control Protocol

- *Byte Oriented Protocol*
  - TCP Send & RECV data as continuous flow of bytes
  - Like flow of water thorough a pipe
  - Ensures every drop of water (= byte) is recvd by the recvr successfully
  - Every byte of data is tracked by TCP protocol

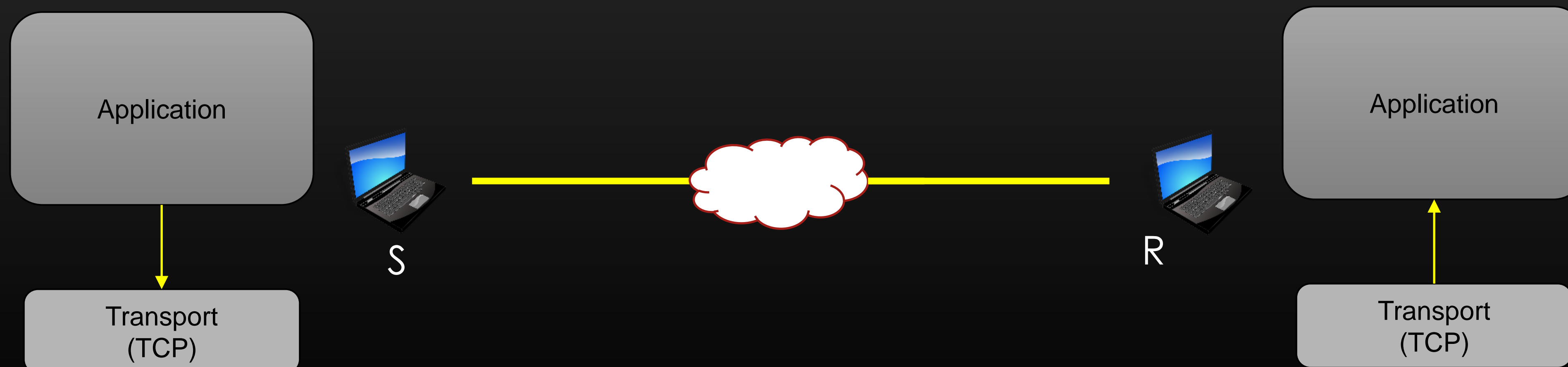
- *Out of order delivery of the packet*
  - TCP (the receiving) handles this gracefully
  - Ensures that data is consumed by the receiving application in the correct order



Visit : www.csepracticals.com
Owned by : CSEPracticals

Transmission Control Protocol

- *Reliable Delivery*
  - Ensure all Application data bytes are delivered to recipient, none should be missed
  - TCP Sender and receiver jointly implements Reliable delivery procedures
  - TCP implements ARQ (Automatic Repeat Request) for data recovery
  - Very detailed and complex mechanism – Hence a separate course



UDP Vs TCP

TCP	UDP
Slower and complex service	Simpler and fast service
Connection-oriented	Connectionless
Stateful	Stateless
Reliable : Can recover lost packets, detect malformed corrupted packets, react to congestion in the network. Order preservance etc	Unreliable : No such mechanism, a packet lost or corrupted is gone forever, Order cant be guaranteed
Byte Stream Oriented Protocol	Datagram oriented protocol
if the application needs reliability	Appln do not needs reliability
Eg : Downloading a software pkg	Eg : Audio/Video streaming

Summary

- We had a quick overview on TCP/IP stack and OSI Model
- We discussed two famous transport layer protocols : UDP and TCP
- In the remaining sections of the course, We shall going to have a deep-dive into TCP protocol internals !

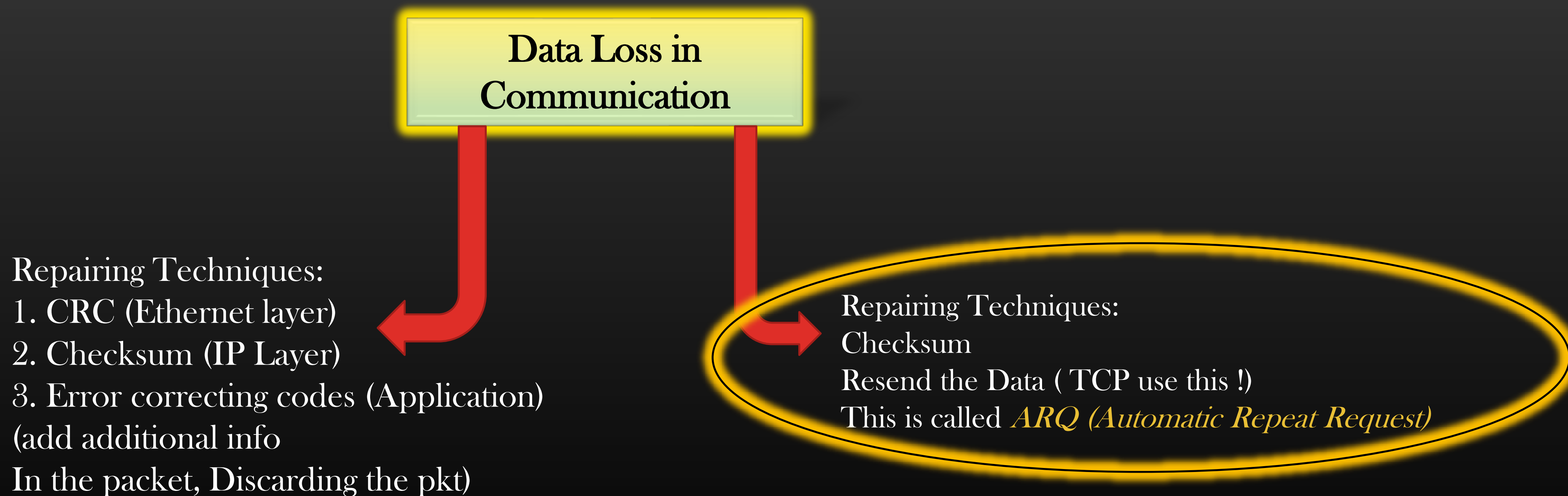
TCP

Overview

Visit : www.csepracticals.com
Owned by : CSEPracticals

- In this Section of the course , we will touch on all aspects of TCP but at a higher level, no drilling down as of now
- This will help us to build the base and understand what we shall going to explore in subsequent sections of the course
- This will also help us understand the higher level functionality of the TCP, and alert are mind in advance to ask right "how" and "why" questions
- Then in subsequent section of the course, we shall dive deep into technicalities of the TCP protocol in greater detail
- Understanding TCP in a clear and concise manner is a little tough and require some organized effort
- We have to be patient and go in an organized way step by step manner to conquer TCP

- TCP Goals :
  - TCP has loads of research behind it , and it is a result of research spanning over 20 years to make TCP stand where it is today
  - TCP has been designed for *Reliable Data Delivery in a lossy network*



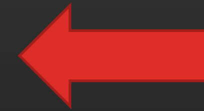
ARQ Challenges

- TCP sending and receiving process may reside anywhere on the network – separated by tens of intermediate routers in the network
- Intermediate routers can themselves impose problems – packet loss, slow routers etc
- Network itself (like bandwidth) impose problems on rate of communication
- In a nut-shell, there can be 'n' number of factors which causes disruption in data flow between tcp-sender and tcp-receiver
- Network is like open ocean, anything can happen any time !
- So, several Question arises to implement ARQ strategy to deal with packet loss/corruption or other anomalies imposed by disturbing agents of network

Visit : www.csepracticals.com
Owned by : CSEPracticals

ARQ Challenges

1. How Receiver detects that packet is malformed ?
2. How sender can determine whether the receiver has received the packet ?
3. How long the sender should wait for ACK from Receiver ?
4. What if ACK itself is lost ?
5. How receiver will manage when it receives packets out of sequence ?
6. What if receiver is slow than Sender Or Receiver receives duplicate copes of the packet ?
7. What if network itself is slower or recover over a period of time ?
8. With how much rate should the sender send the packets to receiver ?



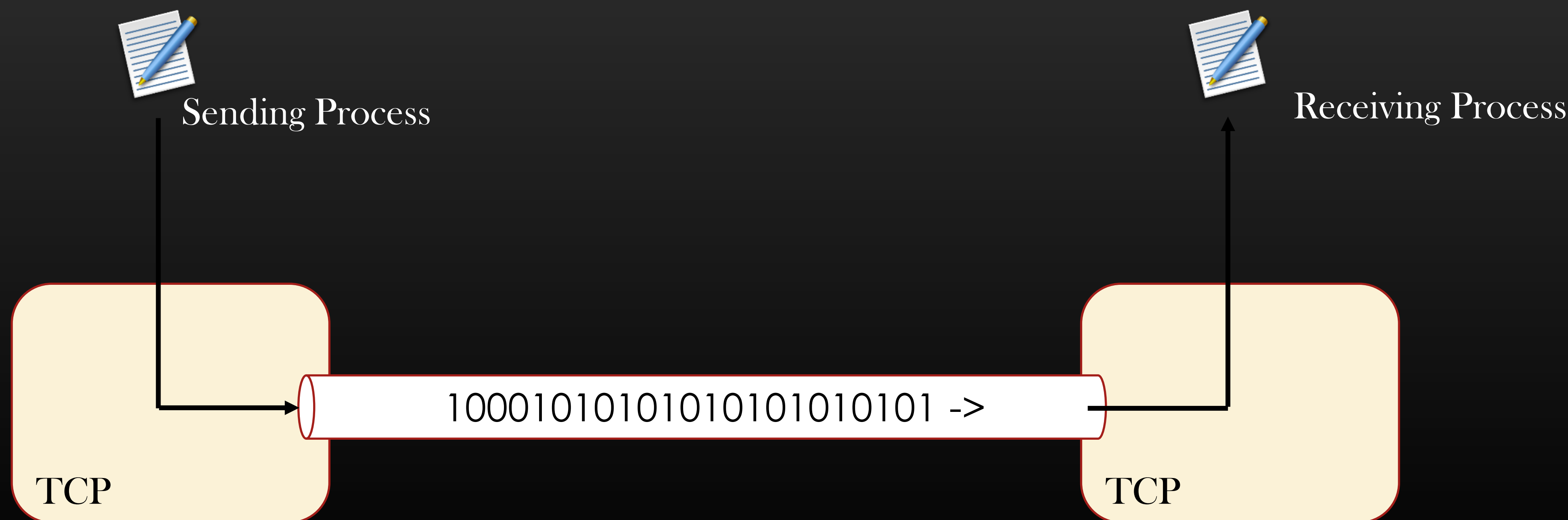
We shall try to find the answers to
All these questions in this course !

TCP ARQ mechanism takes above stated points into consideration to implement its reliable data delivery functionality
Over lossy network

Not implemented over night, it is an outcome of research spanning around 20 years with 100s of research papers !

TCP – Byte Oriented Protocol

- TCP Sender and Receiver Exchange data as a stream of bytes
- Analogy :
  - TCP sender is sending water flow towards receiver in a pipe, where each drop of water is a Byte
  - TCP Sender and Receiver keeps track of how much water is sent and received by keeping explicit track of each drop of water separately (byte of data)



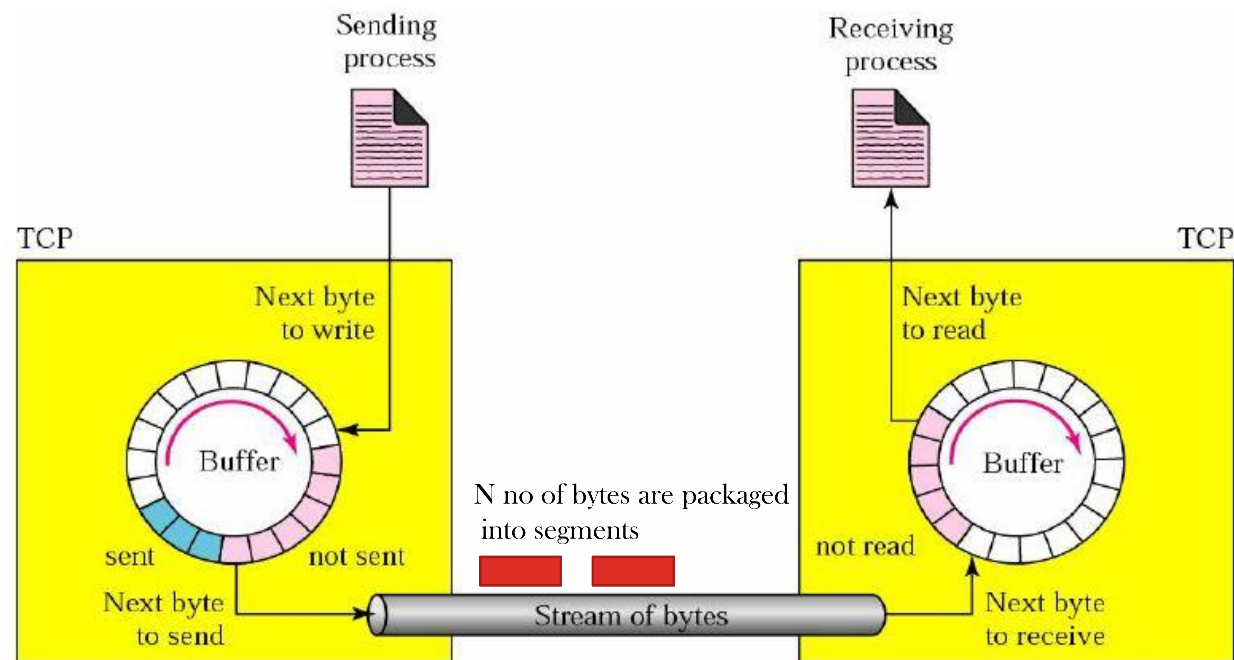
TCP – Byte Oriented Protocol

- TCP Keeps track of appln data sent and recvd at the Byte Level.
- Therefore TCP is called as Byte or Stream oriented protocol

- Analogy :
  - TCP sender is sending water flow towards receiver in a pipe, where each drop of water is a Byte
  - TCP Sender and Receiver keeps track of how much water is sent and received by keeping explicit track of each drop of water separately (byte of data)
  - Each byte of data is ***tracked*** by a unique id called ***Sequence no.*** at either ends

- However, Sending and Receiving speed may not be same

- Therefore, TCP sender and Receiver both needs buffers
  - Sending and Receiving Buffers
  - Implemented as Circular Queues



TCP – Connection Oriented Protocol

- TCP is COP, meaning, Sender and Receiver must mutually agree with each other that they want to establish TCP communication before actually exchange of TCP data
  Analogy : *Dialing a phone number, waiting for the other end to answer the call*

- By connection Setup means, Sender and receiver saves in its internal data structure the state of connection which includes :
  - > With whom are they communicating (IP address and port number)?
  - > How many bytes of data sent and received ?
  - > What is the next byte to expect or send over a connection ?
  - > What is peer's capacity to process the data ?

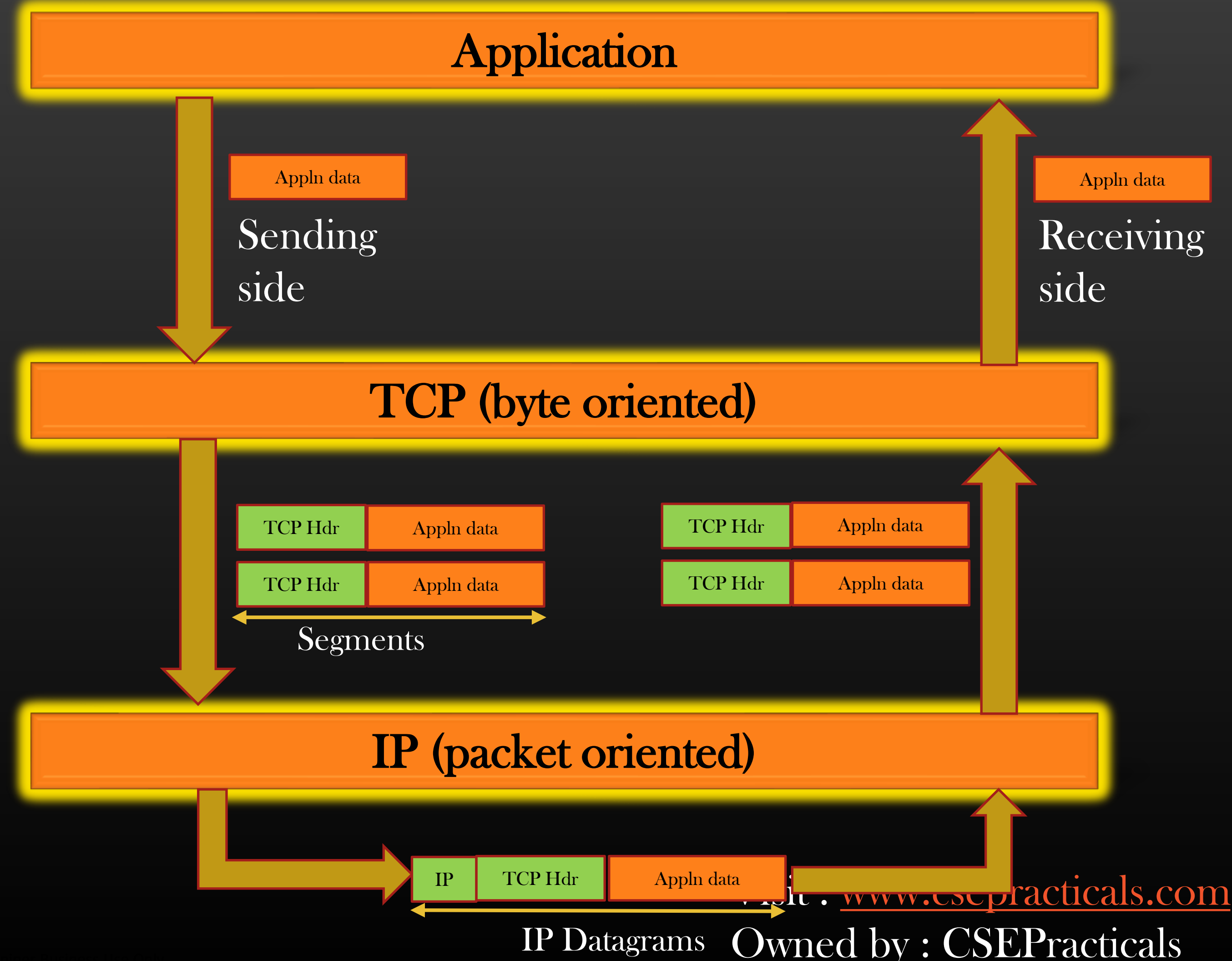
Connection is Virtual , Not physical !

- Both sender and Receiver save the state of connection
  - It is for this reason, that large file transfer peer the network when disrupted, can resume because Sender and receiver knows where they left last time

Connection is Duplex

Segments and Sequence Numbers

- TCP is sandwiched between Application and IP layer
- TCP packs the application data into discrete packages called Segments
- Size of segments is decided dynamically and keeps on changing depending on network or recipient state
- Segments size is chosen to avoid unnecessary fragmentation at IP layer
- Segments contain 'N' bytes of data, where N is segment size
- TCP stamp every byte it is sending in segments with a unique number called sequence number
- The SEQ no of first byte is also treated as Segment number



Visit : www.csepracticals.com
Owned by : CSEPracticals

Segments and Sequence Numbers

- Sequence number : Sequence number is the unique id of a Byte of data which TCP sender sends to TCP receiver
- Every byte of data provided by application to underlying TCP is assigned incremental sequence numbers by TCP
- SN of first byte of application data present in a segment is also referred to as segment number
- Example :

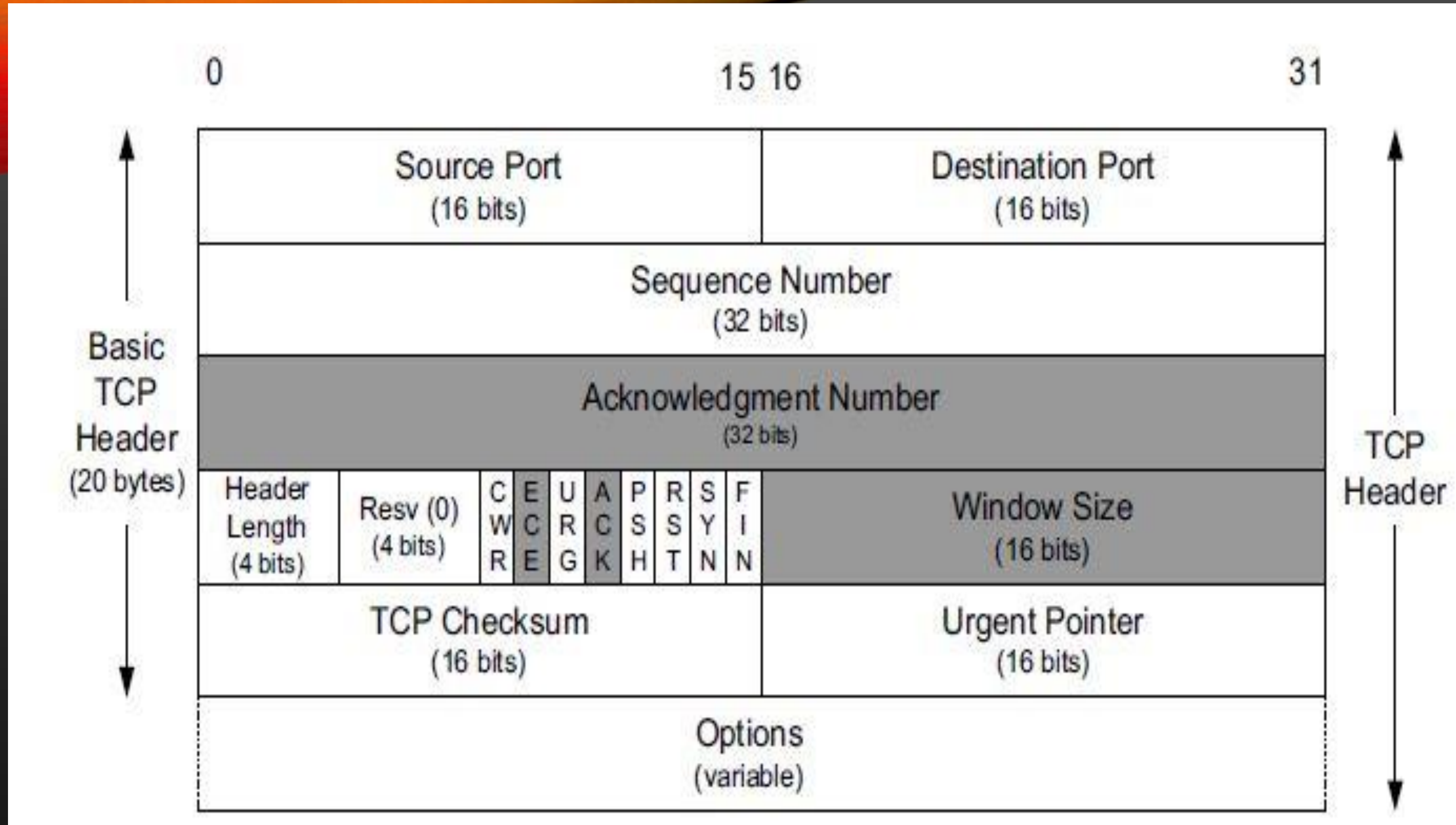
- Every Byte of Data is identified by a SEQ no so that TCP can keep track of each byte whether reliably delivered to receiver or not
- TCP do not examine “what” data application is sending to it and how it is structured. From TCP perspective all data is just 0’s and 1’s (junk !!)

Segments and Sequence Numbers

Numerical :

Slide no 34 and 35

Visit : www.csepracticals.com
Owned by : CSEPracticals

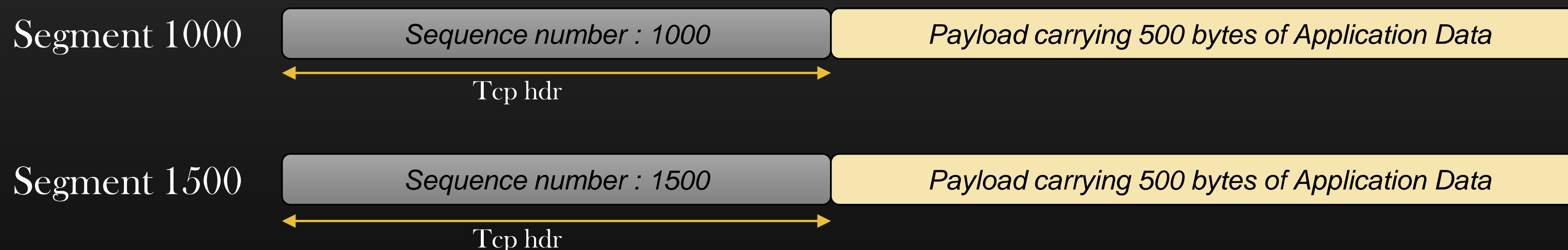


- Sequence Number field is mandatory and is always present in a TCP segment (irrespective of segment type)
- Acknowledgement Number is Valid only when ACK bit is set

- The flow of Segments between communication TCP processes in either direction is controlled and regulated by
  - Sequence Number (32 bit)
  - Acknowledgement Number (32 bit)

Sequence Number

- Sequence Number is like a unique identifier of the segment. In TCP, every byte has sequence number, not every segment
- Sequence number of the first byte in payload of segment is termed as sequence number of segment



- Sequence number is incremented by TCP sender by the amount of bytes the sender has sent in previous segment

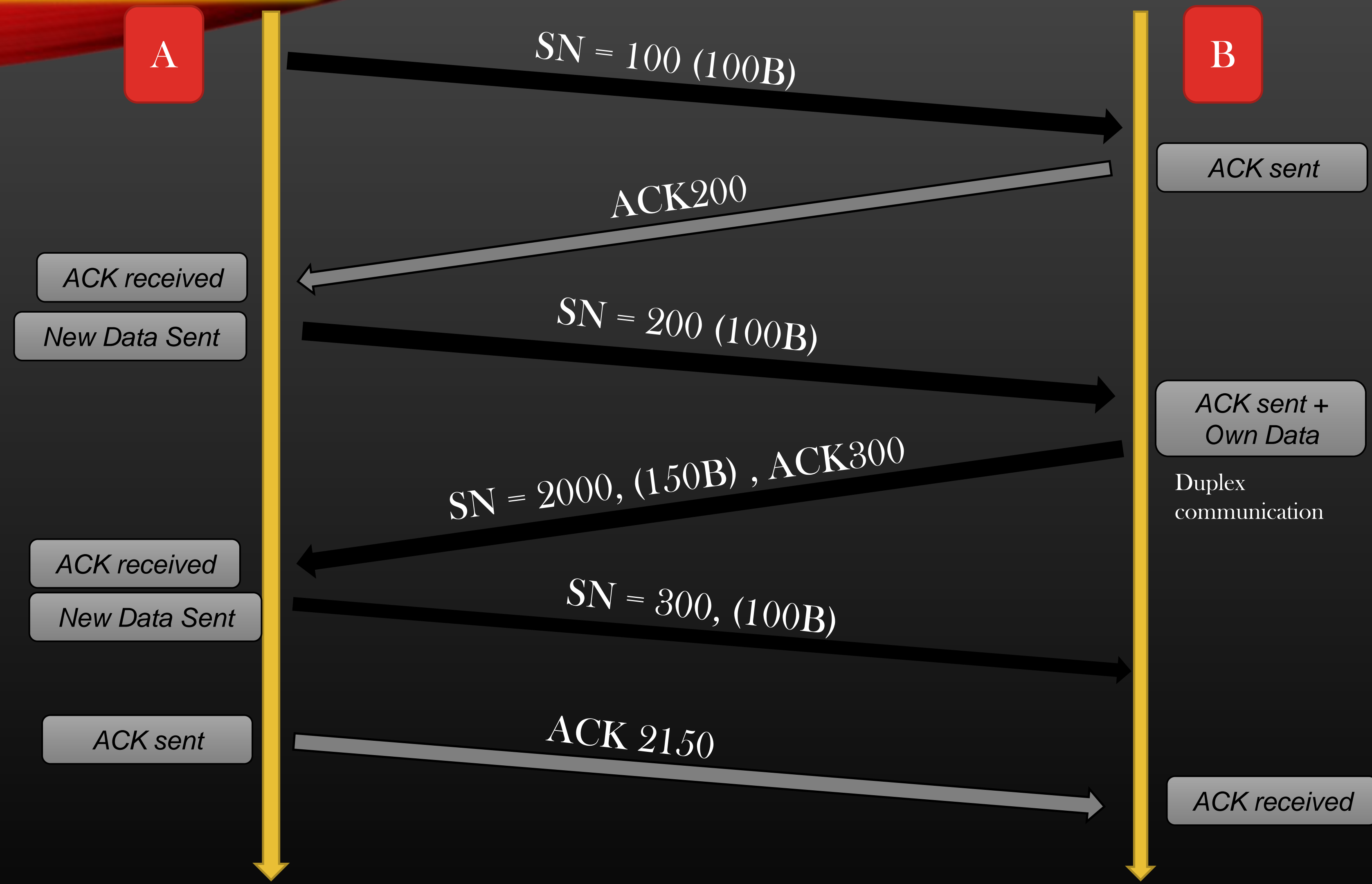
- The flow of Segments between communication TCP processes in either direction is controlled and regulated by
  - Sequence Number (32 bit)
  - Acknowledgement Number (32 bit)

Acknowledgement Number

- Acknowledgement number is the sequence number of the segment which the TCP receiver expects from TCP sender in the next segment
- In other words, if TCP Receiver specifies ACK # as 2000 with ACK bit set in segment, it means, TCP receiver is telling TCP sender – "*I have successfully received 1999 bytes of data, I am expecting 2000th byte and onwards now in your next segment*"
- ACK bit combined with ACK number is a feedback to the TCP sender from TCP receiver about the confirmation of the successful reception of TCP payload data
- TCP piggybacks – in the same segment, TCP sender can ship next payload bytes, specifying new sequence number and at the same time ACKnowledge the previous TCP data it has received from peer using ACK no and ACK bit

Visit : www.csepracticals.com
Owned by : CSEPracticals

TCP Segments Type



TCP Segments Type

- Data Segments
- Pure ACK Segments
- Data + ACK Segments

ACK tells the sender the next expected byte !!

TCP Reliable Data Delivery

- The main reason why TCP has been designed and one of the most widely standard protocol in use today is because it guarantees – Reliable Data Delivery
- Other Transport Protocol such as UDP/IP works on *"send and forget"* principle. There is no feedback mechanism from Recipient which tells sender to retransmit lost data

UDP's Send and Forget Scheme

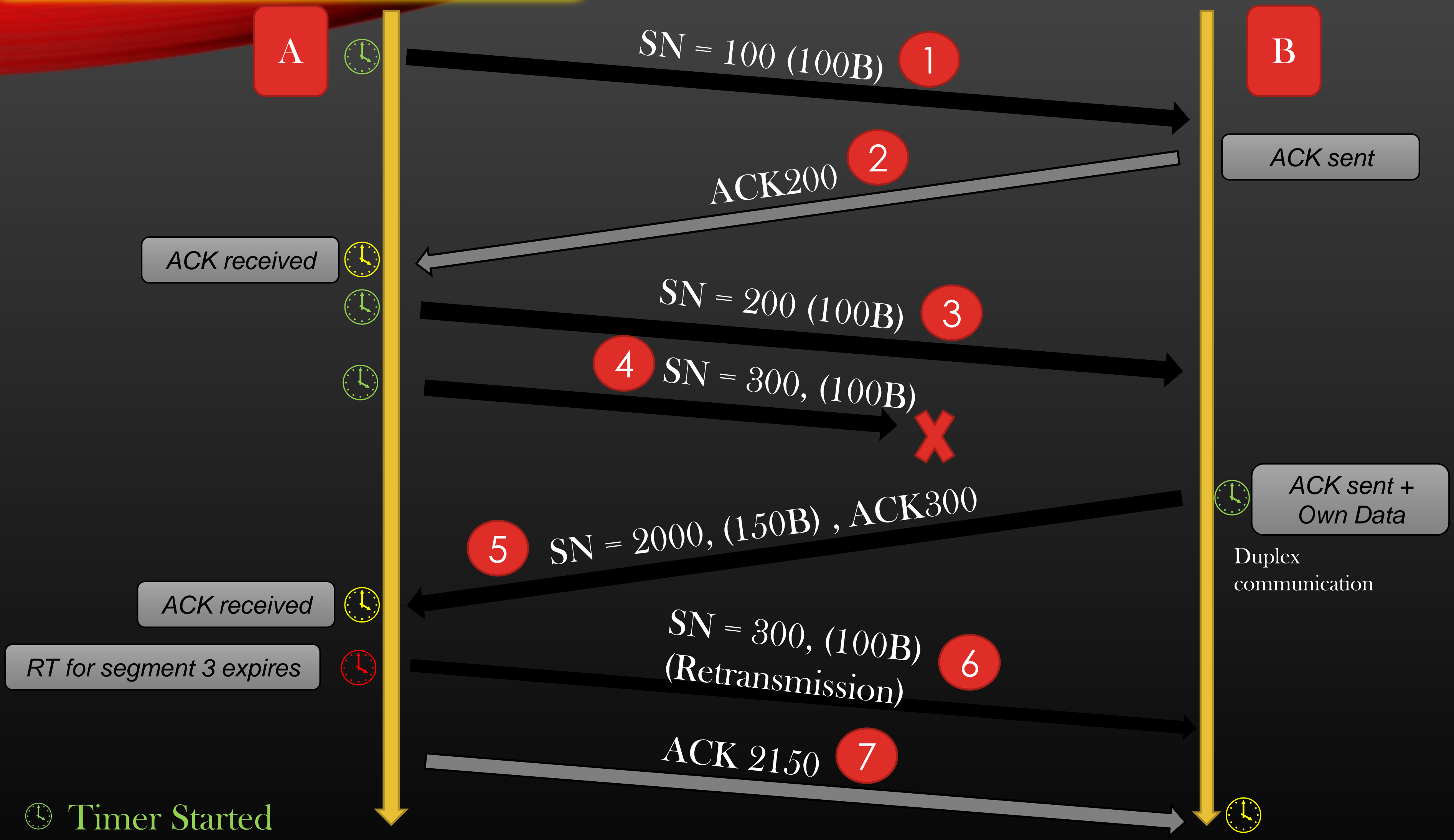
There is no mutual agreement either Between sender and receiver if they Really want to participate in communication

Sender has no way to determine if UDP DG3 has been delivered or not, if lost it is lost forever

TCP Reliable Data Delivery

- Contrary to "send and forget" scheme, TCP works on Feedback mechanism to implement Reliable delivery of data
- Remember Network Layer (L3 routing) also works on "send and forget" scheme
- It is the TCP recipient responsibility to send feedback msg to TCP Sender. These Feedbacks are called ACKs in TCP terms
- TCP sender starts the timer when it sends a segment. Before expiration of this Timer if it receives ACK from TCP recipient, then Sender assumes data has been delivered. This Timer is called Retransmission Timer and is set for each segment it has sent
- If TCP sender do not receives ACK from recipient, and RT expires, TCP sender assumes data has been lost and it retransmits the segment to recipient. In addition to retransmission, TCP sender takes certain action to avoid further congestion because it assumes that data is lost because of congestion in the network or Receiver is probably overwhelmed

TCP Reliable Data Delivery



Feedback scheme of TCP

Each time Device A sends a message, it starts a timer.

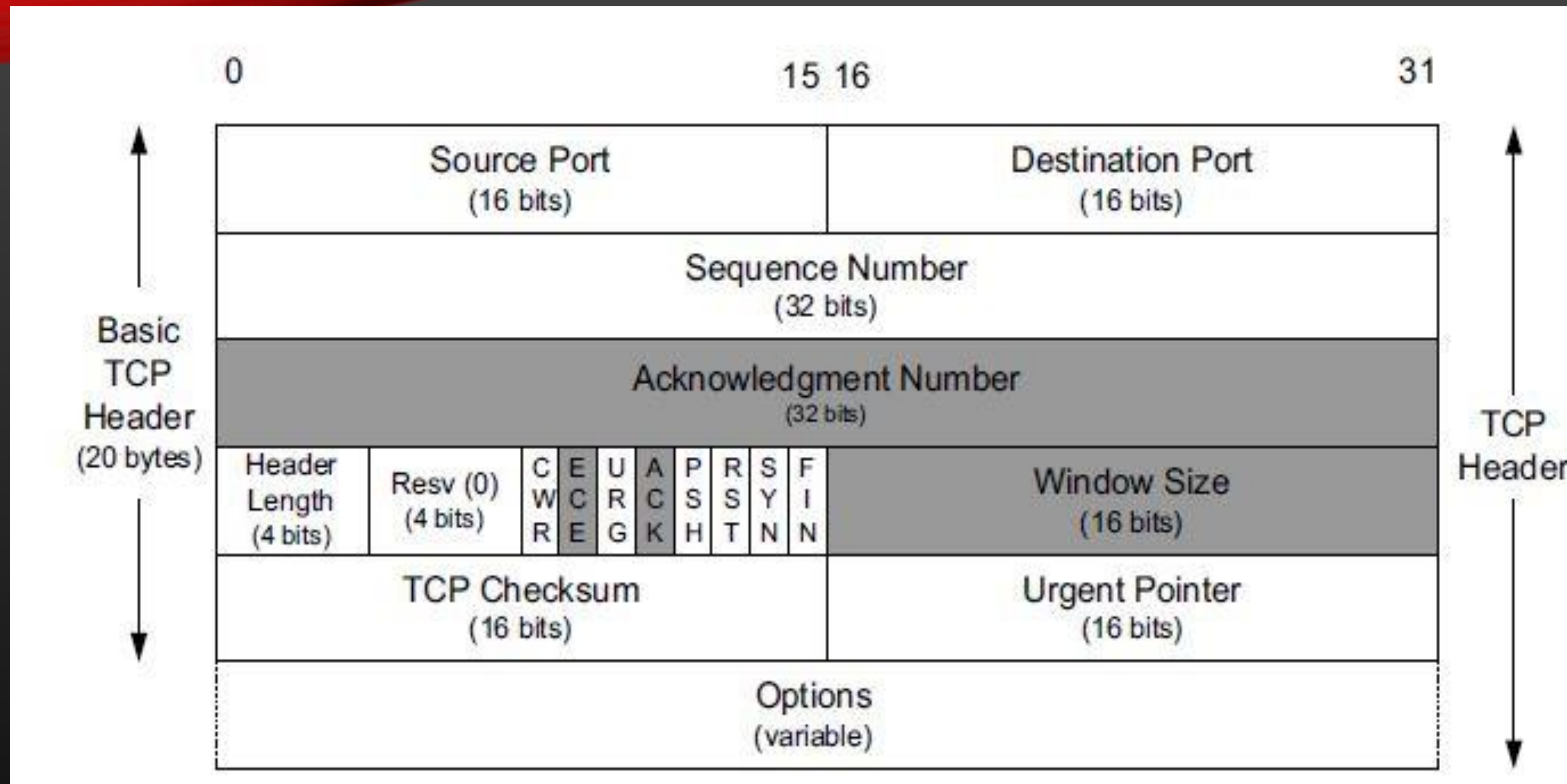
Device B sends an acknowledgment back to Device A when it receives a message, so that Device A knows that it successfully transmitted the message. If a message is lost, the timer goes off, and Device A retransmits the data

SNs runs independently In two directions

Visit : www.csepracticals.com
Owned by : CSEPracticals

TCP + IP

TCP Header Format



- Fields which are in grey are set by TCP recipient while sending the ACK segment to Sender
- Src and Dst port number identified the TCP sender and TCP recipient processes
- The 4-tuple [TCP sender IP address, TCP sender port number, TCP receiver IP address and TCP receiver port number] Identifies the TCP connection uniquely

Summary

- In this Section, We had a quick short tour over TCP protocol and tried understood its goals, objectives and functionality at a high level
- We understood what is meant by :
  - Connection Oriented Protocol
  - Byte Stream Oriented Protocol
  - Feedback Mechanism and Retransmission
  - TCP hdr format
- In the Subsequent Sections of the course, We shall dive deep into various features of TCP one by one and understand each of those in detail since, it is TCP MASTERCLASS course
- Along the way, you will have many numerical and assignments to grasp the idea better
- It is not a cake walk to understand TCP internal design in the first attempt

End Goal of this Course

- Encapsulating the end goals through just one diagram, our end-goal is to understand the below TCP Graph

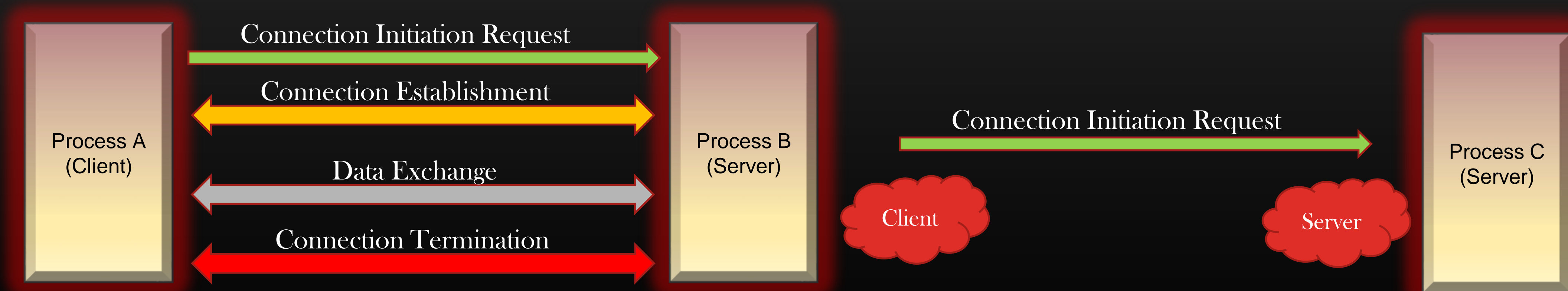
TCP

Connection Management

- TCP is a connection-oriented protocol.
  Before either end can send data to the other, a connection must be established between them
- In this section of the course, We shall discuss TCP connections management from start to finish
- We shall discuss the finite state machine for TCP connection management
- 3-way handshake mechanism
- Synchronization of ISNs (Initial Sequence numbers)
- Connection Termination

Mastering TCP -> Connection Management -> Who is Client and Who is Server ?

- First we should under the meaning of Server and Client
- Server
  - A Server is any process which is waiting for Connection Initiation request from Client Process
  - Server process, by itself, never starts the initiation of communication
  - It only responds to request from other process (clients)
  - Example : web server
- Client
  - A Client is any process which initiates the connection with the Server
  - Eg : Browser



- TCP Connection is uniquely defined by 4 tuples :

  [TCP client IP address, TCP client port number, TCP server IP address and TCP server port number]

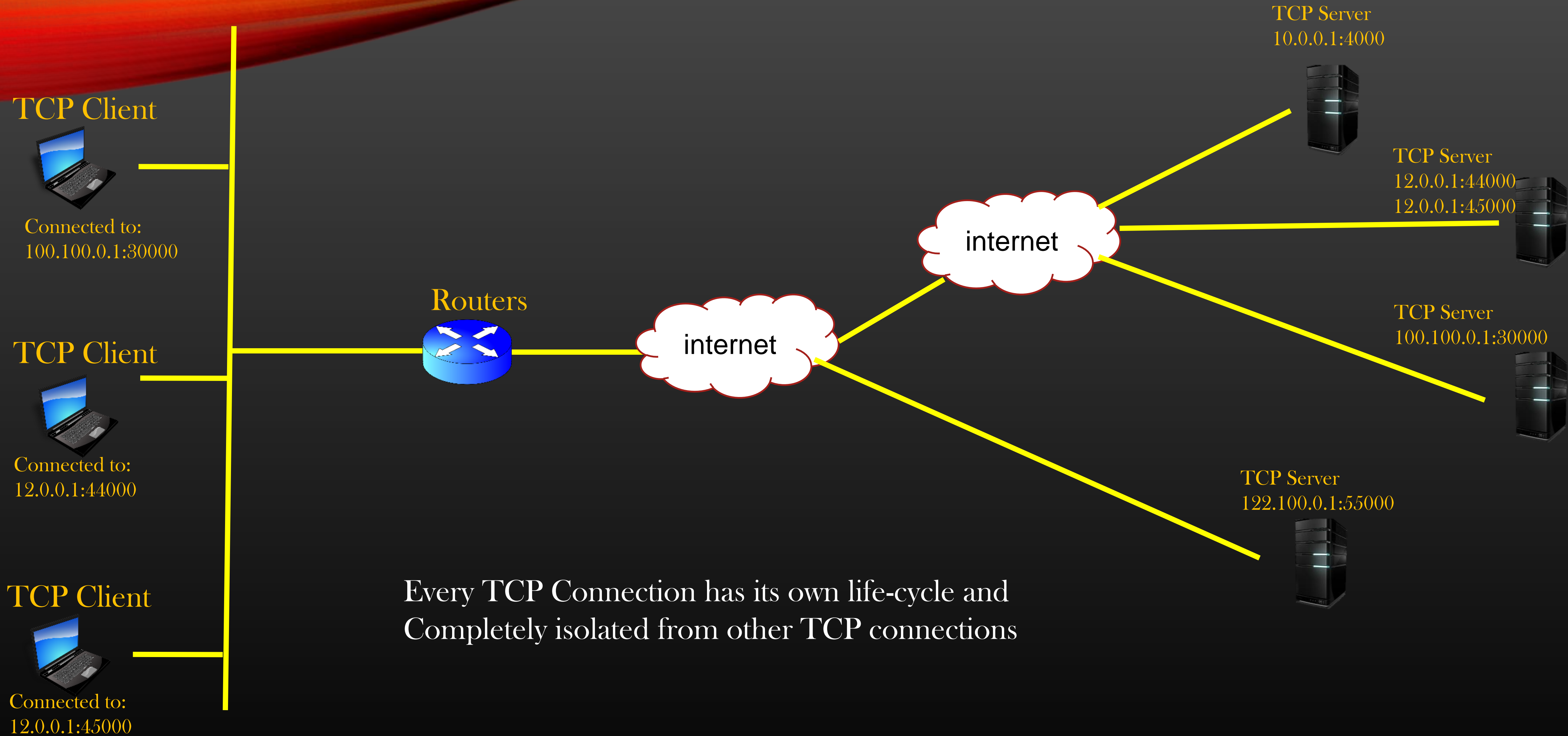
- TCP Server Process could be running anywhere on the internet/Network. Same is True for TCP Client

- For the TCP Client process to connect to TCP Server process , TCP client needs to know :
  - > TCP Server's machine IP address and
    (Helps Identifying the machine X running TCP process in the network)
  - > TCP Server process Port number
    (Machine X may have many TCP Server Process running, which among these ?)

- Similarly, TCP Server's need to know TCP client's IP address and port number sending the reply

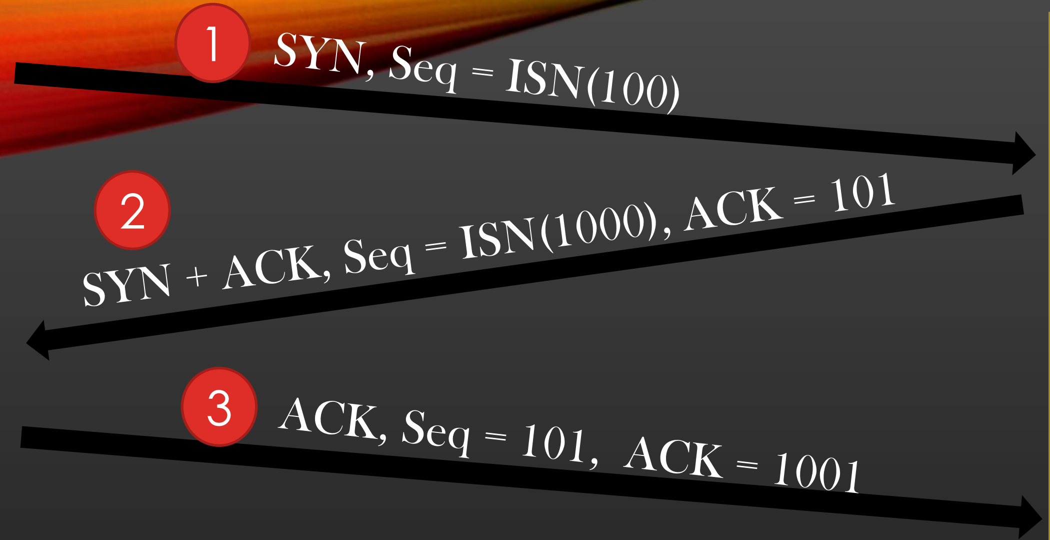
- When TCP client's initiates the TCP connection with TCP Server, TCP client sends its own IP address and port number to TCP Server in IP hdr and TCP hdr respectively

Visit : www.csepracticals.com
Owned by : CSEPracticals



Every TCP Connection has its own life-cycle and Completely isolated from other TCP connections

Visit : www.csepracticals.com
Owned by : CSEPracticals



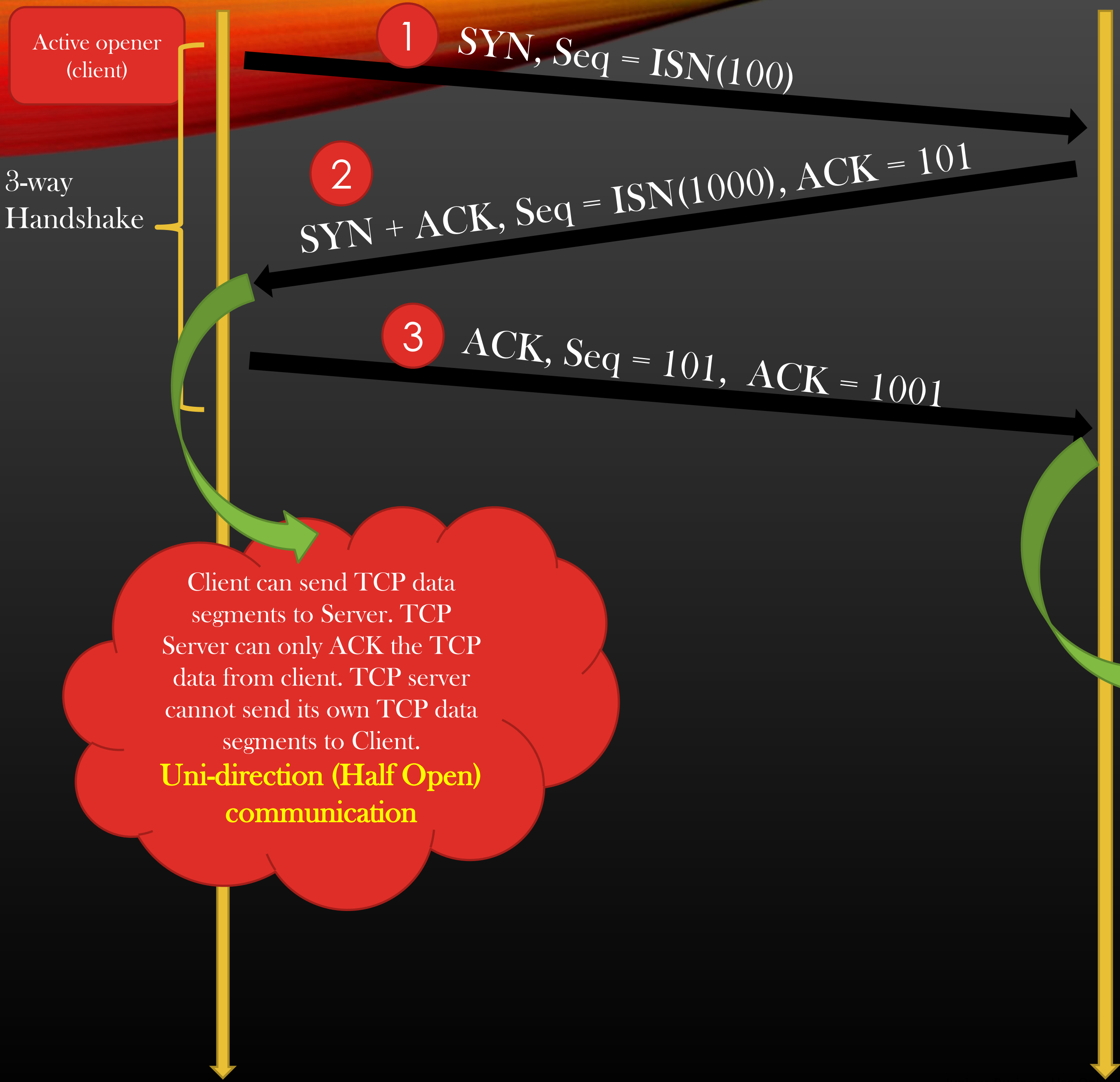
Both parties should show agreement to Communicate with each other !

1. *SYN – want to initiate a TCP connection*
   All my future segments will have seq no 100+
   Do not contain any application data, consume 1 sequence number

2. *SYN – want to initiate a TCP connection*
   ACK – client's request for connection initiation Specified in segment with seq no 101 -1 is accepted
   All my future segments will have seq no 1000+

3. *ACK – request specified in Server's segment with sequence no 1001 – 1 is Accepted*

Visit : www.csepracticals.com
Owned by : CSEPracticals

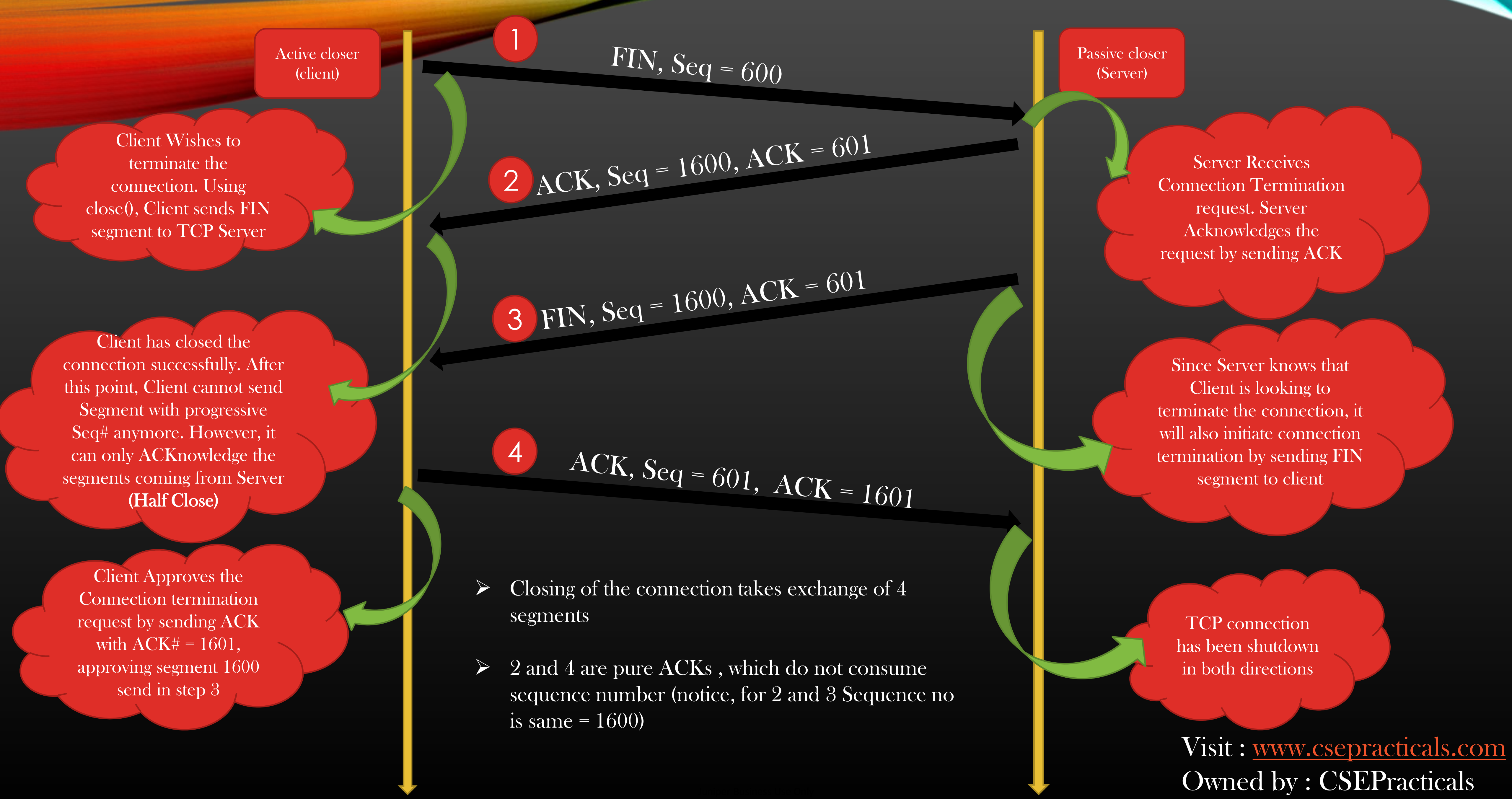


Passive opener (Server)

In the 1 and 2 , each party is telling the other Party the ISN it wishes to use
Step 1 and 2 combined is called **Sequence Number Synchronization**

Server Has got the permission from the client, and now TCP Server can also send data to TCP client.
Bi-Directional Communication

Visit : www.csepracticals.com
Owned by : CSEPracticals



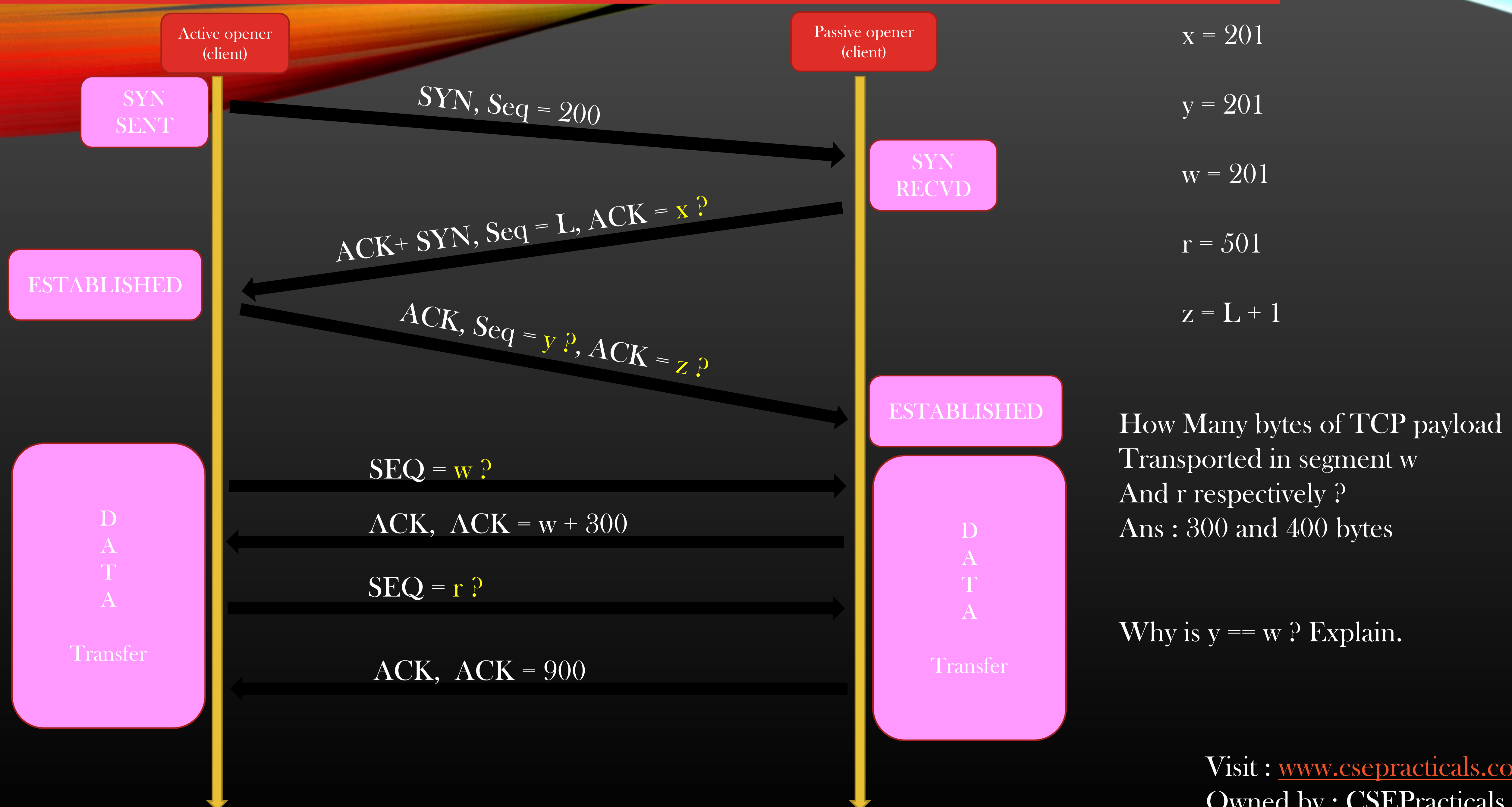
- Closing of the connection takes exchange of 4 segments
- 2 and 4 are pure ACKs , which do not consume sequence number (notice, for 2 and 3 Sequence no is same = 1600)

Visit : www.csepracticals.com
Owned by : CSEPracticals

- SYN segments do not contain any application data, yet they consume 1 sequence number because they need to be acknowledged
- FIN segments MAY not contain any application data, yet they consume atleast 1 sequence number because they need to be acknowledged
- Pure ACKs do not contain any application data, they do not consume any sequence number either because ACKs are not acknowledged
- Data Segments Consume as many sequence numbers as the no of application bytes they are carrying as payload

Any segments that needs an acknowledgement consumes a sequence number

Mastering TCP -> Connection Management -> Assignment



How Many bytes of TCP payload Transported in segment w And r respectively ?

Ans : 300 and 400 bytes

Why is y == w ? Explain.

Visit : www.csepracticals.com

Owned by : CSEPracticals

Active closer (client)

Passive closer (Server)

1 FIN, Seq = 600

2 ACK, Seq = 1600, ACK = 601

3 FIN, Seq = 1600, ACK = 601

4 ACK, Seq = 601, ACK = 1601

This is Pure ACK segment, in which only ACK bit is set. Such Segment do not contain any application payload, therefore they do not consume any sequence numbers.

Pure ACK Segment

ACKs are not acknowledged,
If they are lost, they are lost !

ACKs are no ACKed !!

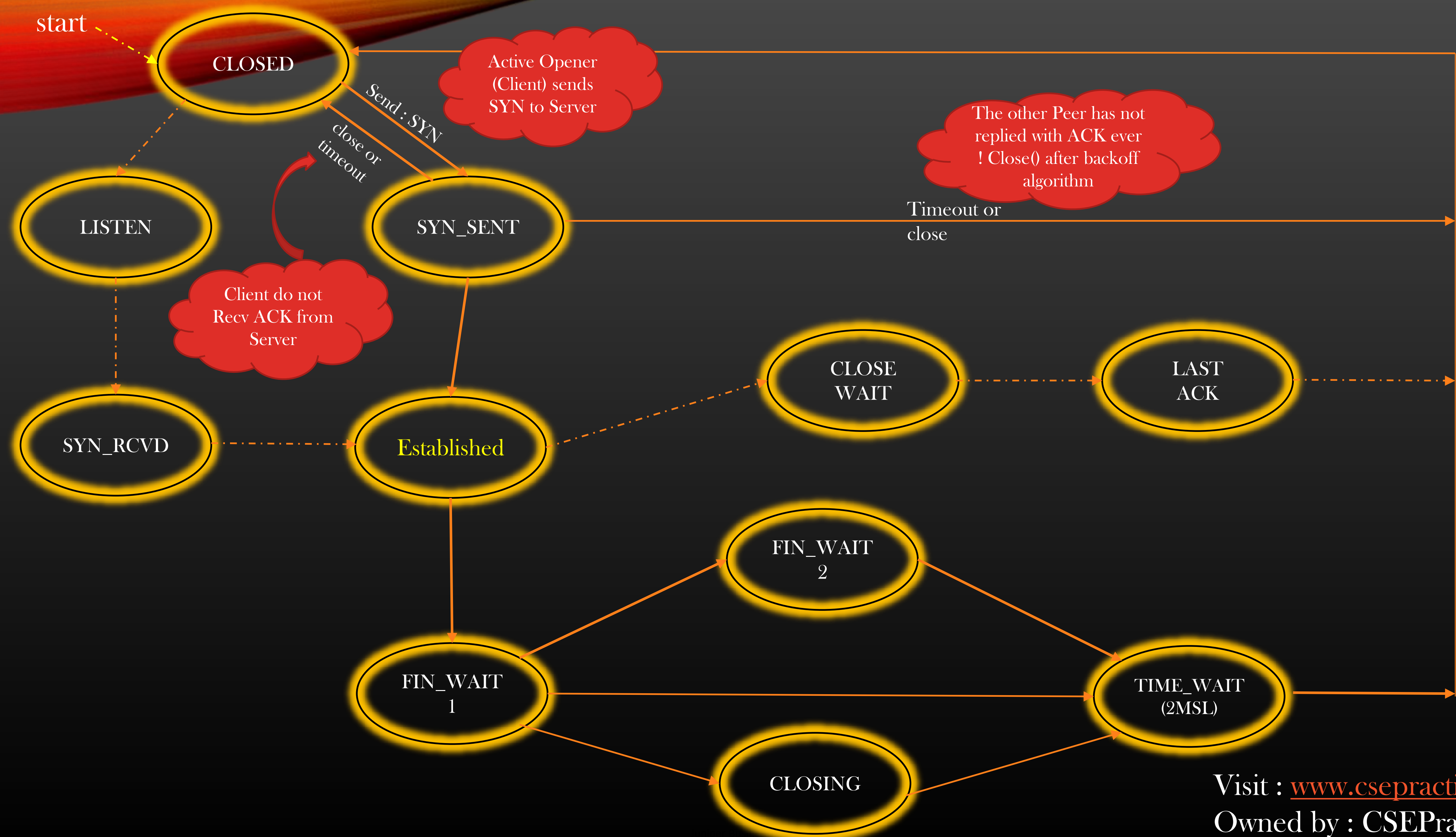
Visit : www.csepracticals.com
Owned by : CSEPracticals

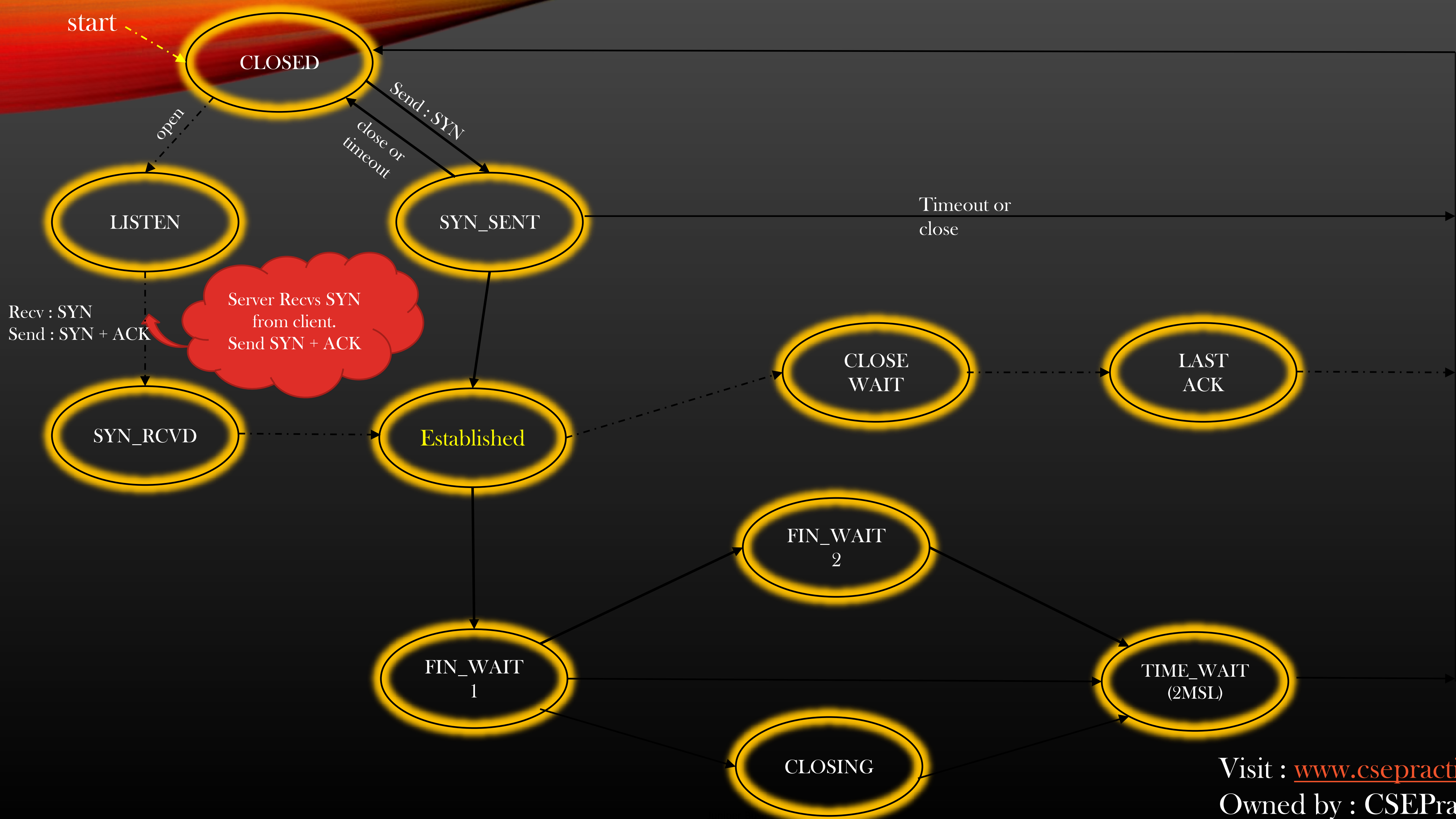
- Active opener i.e. Client sends Connection initiation request (SYN segment) to server which is already down. What will happen ?
- Obviously, the Server will not respond with any ACK. Clients waits for time t, and again probe server with another CIR. This continues . . . For how long ?

Exponential back-off

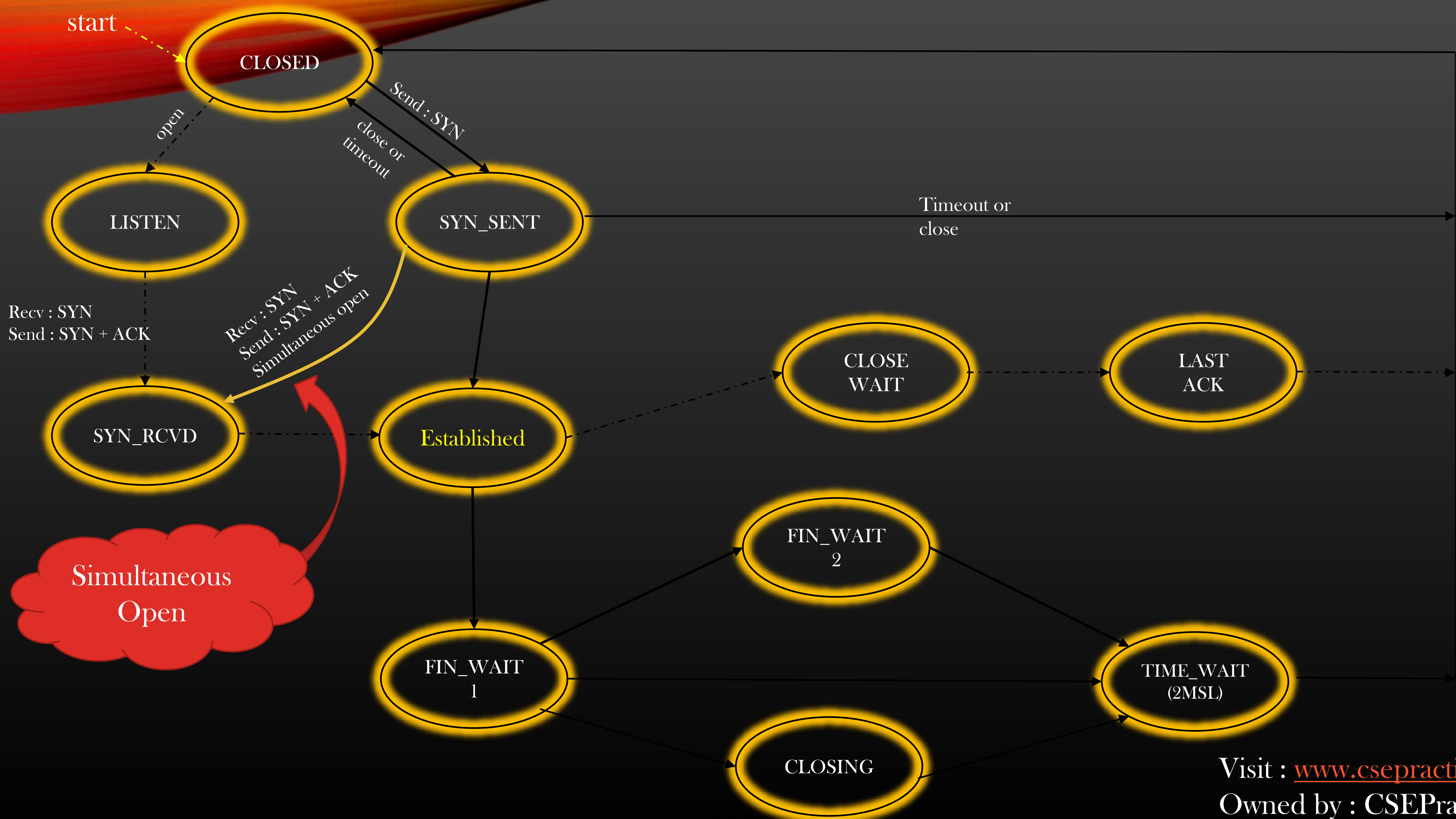
Default no of maximum retries is 5

- The rules that determine what TCP does are determined by what state TCP is in.
- The current state is changed based on various stimuli, such as segments that are transmitted or received, timers that expire, application reads or writes, or information from other layers.
- These rules can be summarized in TCP's state transition diagram
- To understand TCP state transition diagram, be ready to move back and forth between 3-way handshake, Connection termination Steps we already discussed
- Keep the TCP state transition diagram in mind to answer questions in examination . . .

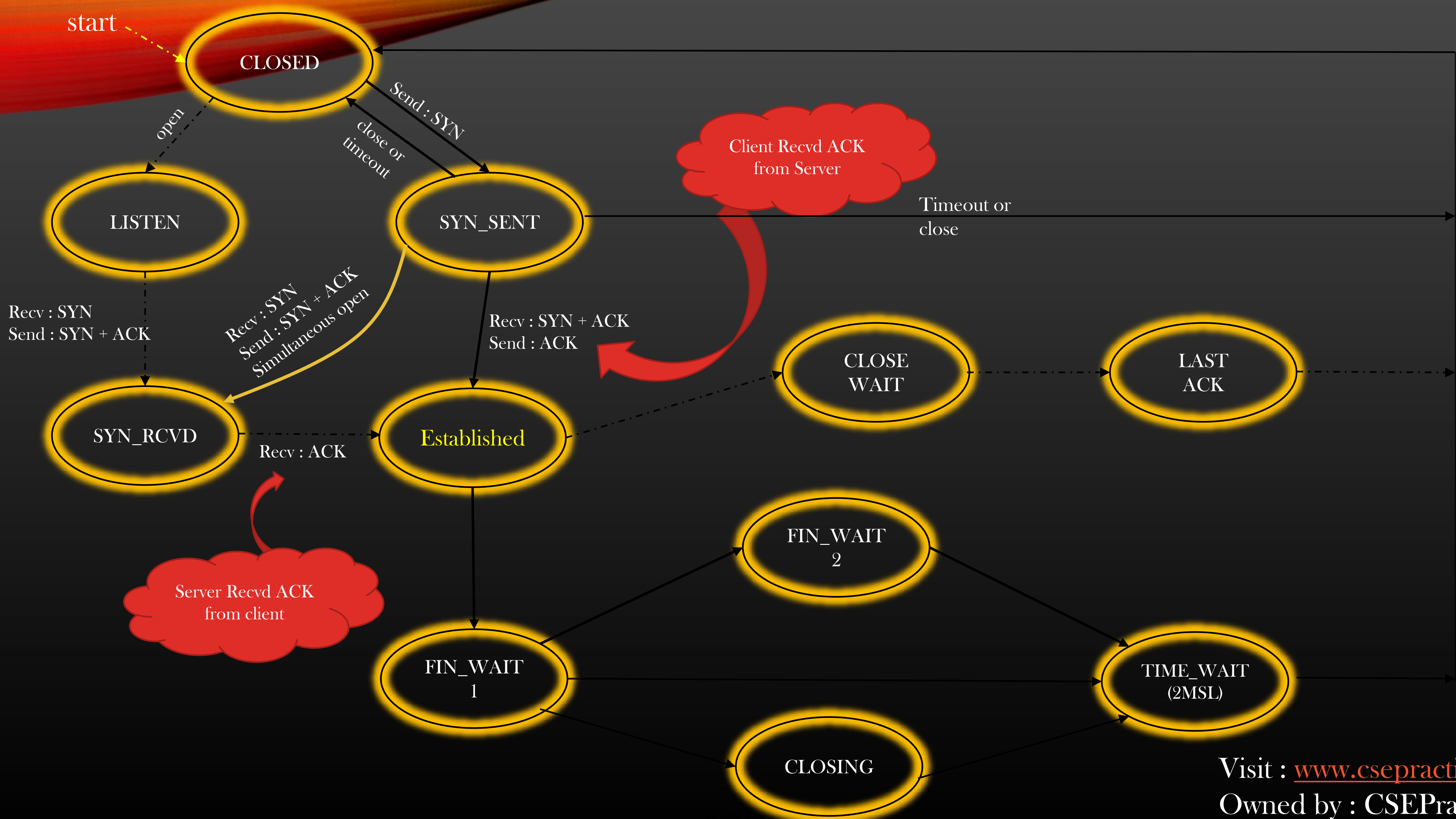

start
CLOSED
Active Opener
(Client) sends
SYN to Server
Send : SYN
close or
timeout
The other Peer has not
replied with ACK ever
! Close() after backoff
algorithm
LISTEN
SYN_SENT
Timeout or
close
Client do not
Recv ACK from
Server
CLOSE
WAIT
LAST
ACK
SYN_RCVD
Established
FIN_WAIT
2
FIN_WAIT
1
TIME_WAIT
(2MSL)
CLOSING



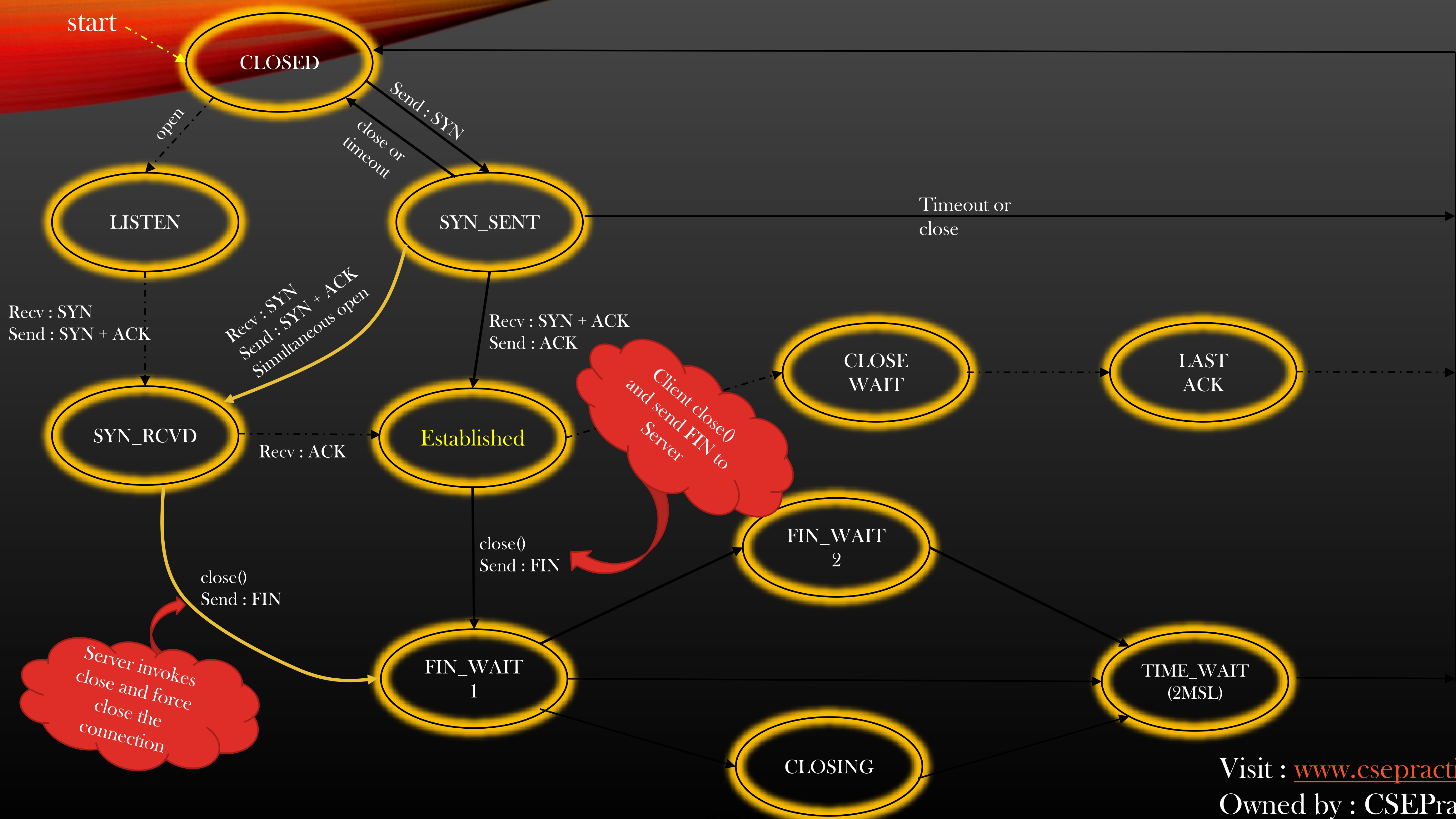
Visit : www.csepracticals.com
Owned by : CSEPracticals



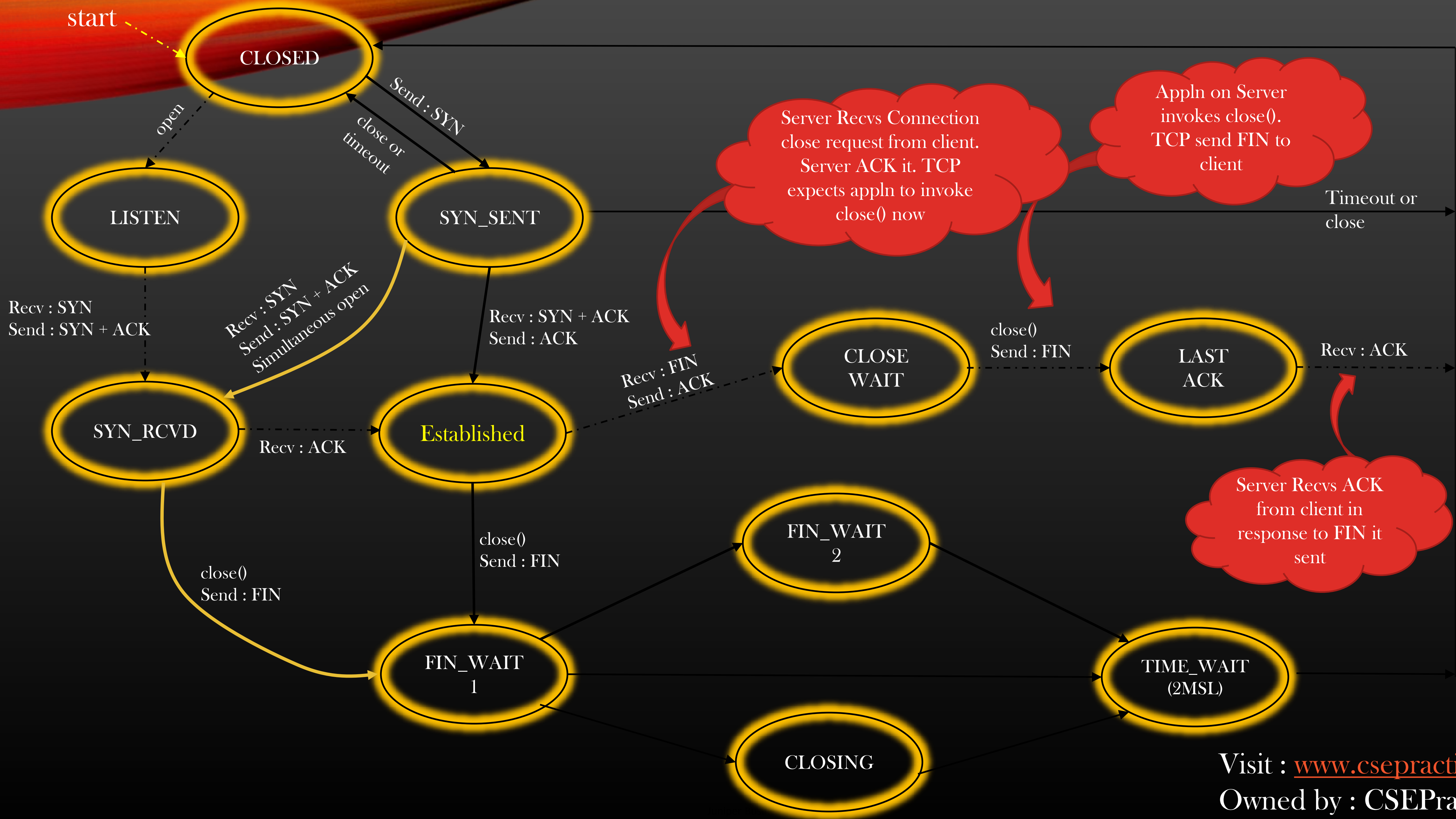
Visit : www.csepracticals.com
Owned by : CSEPracticals



Visit : www.csepracticals.com
Owned by : CSEPracticals

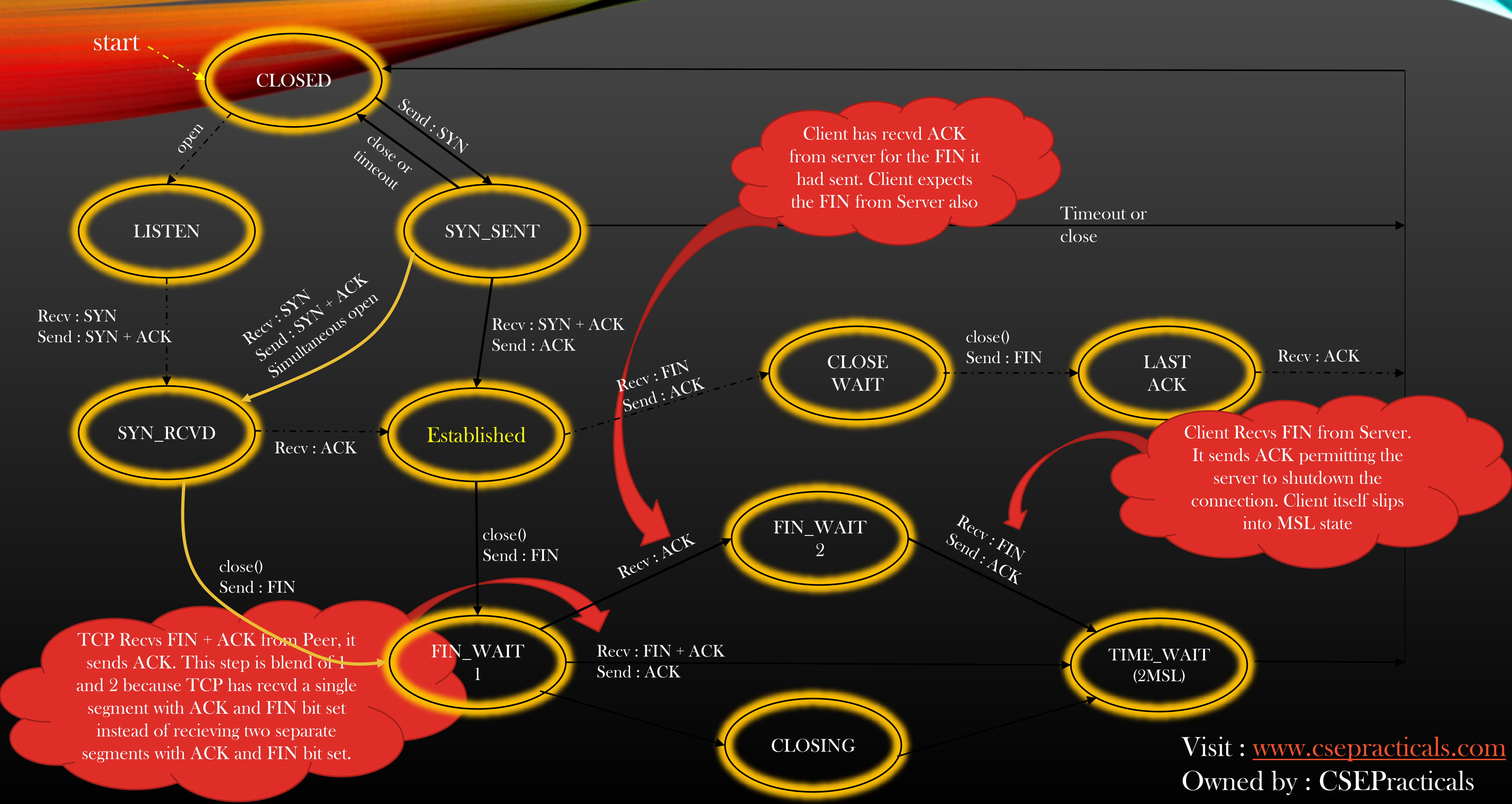


Visit : www.csepracticals.com
Owned by : CSEPracticals

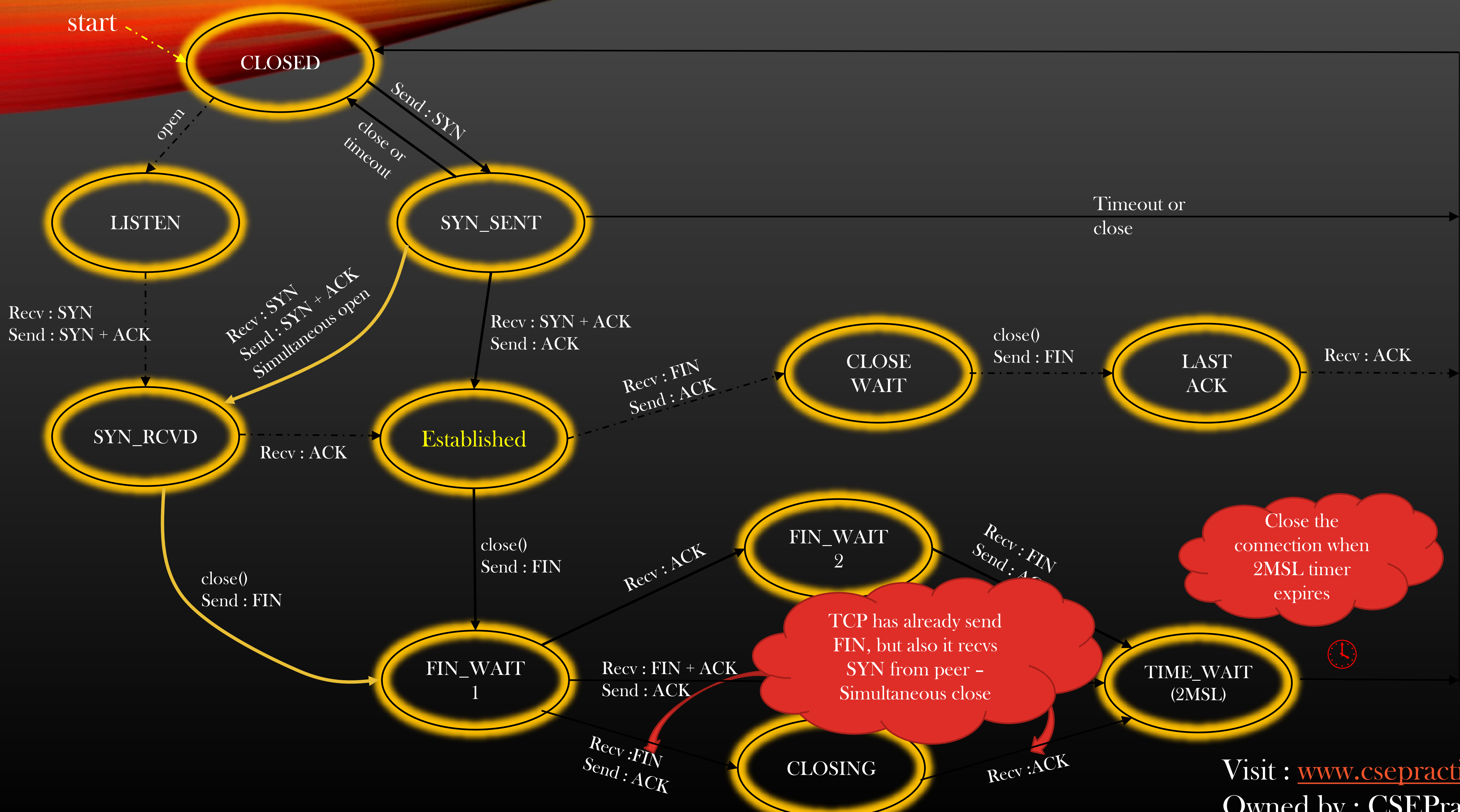


Visit : www.csepracticals.com
Owned by : CSEPracticals

Mastering TCP -> Connection Management -> Finite State Machine

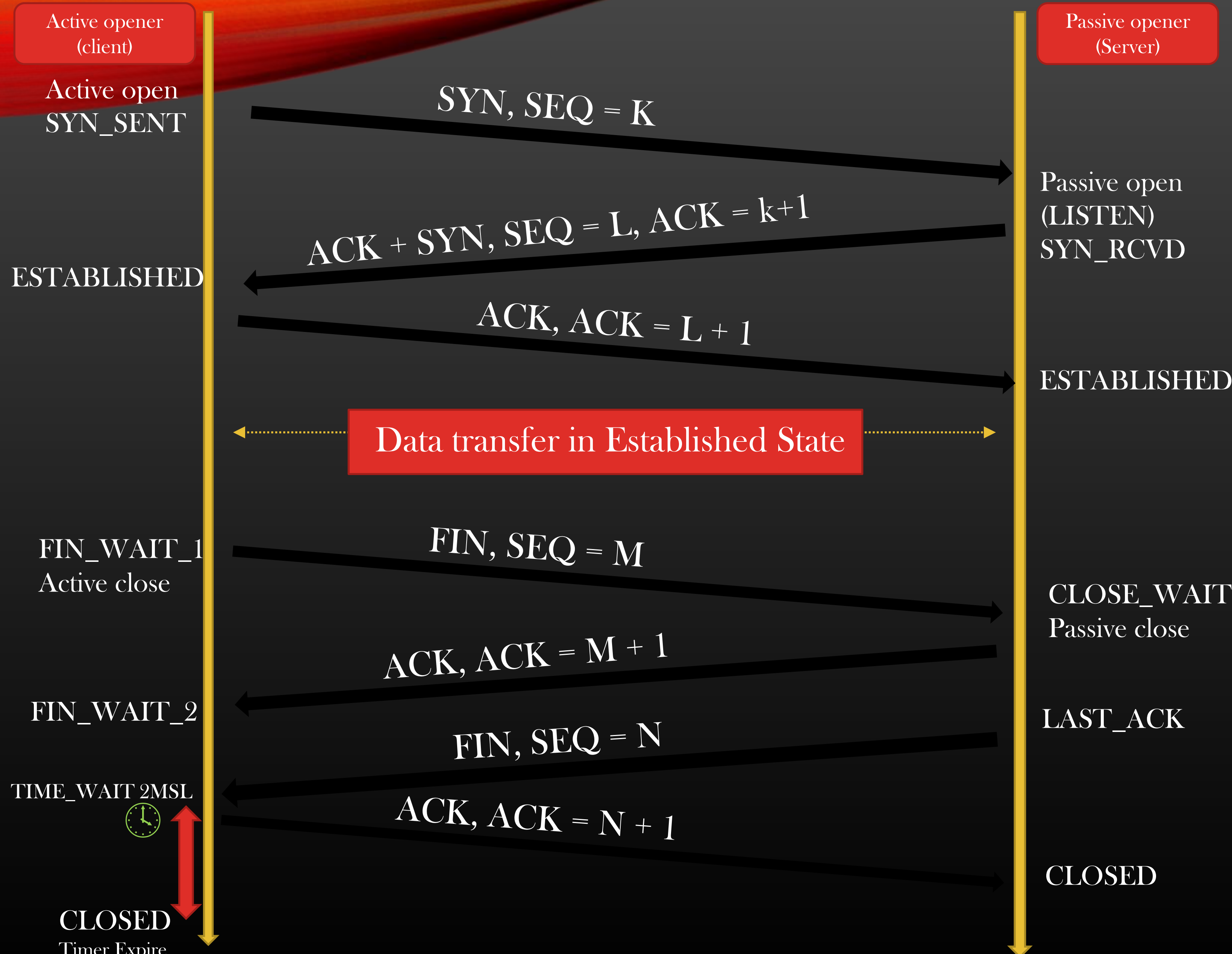


Mastering TCP -> Connection Management -> Finite State Machine



Visit : www.csepracticals.com
Owned by : CSEPracticals

Mastering TCP -> Connection Management -> Finite State Machine Simplified



Definitions :

SYN_SENT – Active opener sends SYN
SYN_RCVD – Passive opener recvs SYN, and send ACK for it
ESTABLISHED – When ACK is recvd for SYN

FIN_WAIT_1 – Active closer Sent the FIN, waiting For ACK

CLOSE_WAIT – passive closer recvd FIN and sent ACK for it. Waiting to send its own FIN now.

FIN_WAIT_2 – Active closer recvd ACK for its FIN, Waiting for FIN from other end now

LAST_ACK – passive closer sends its FIN in response To FIN it recvd from other end, waiting for ACK of this FIN

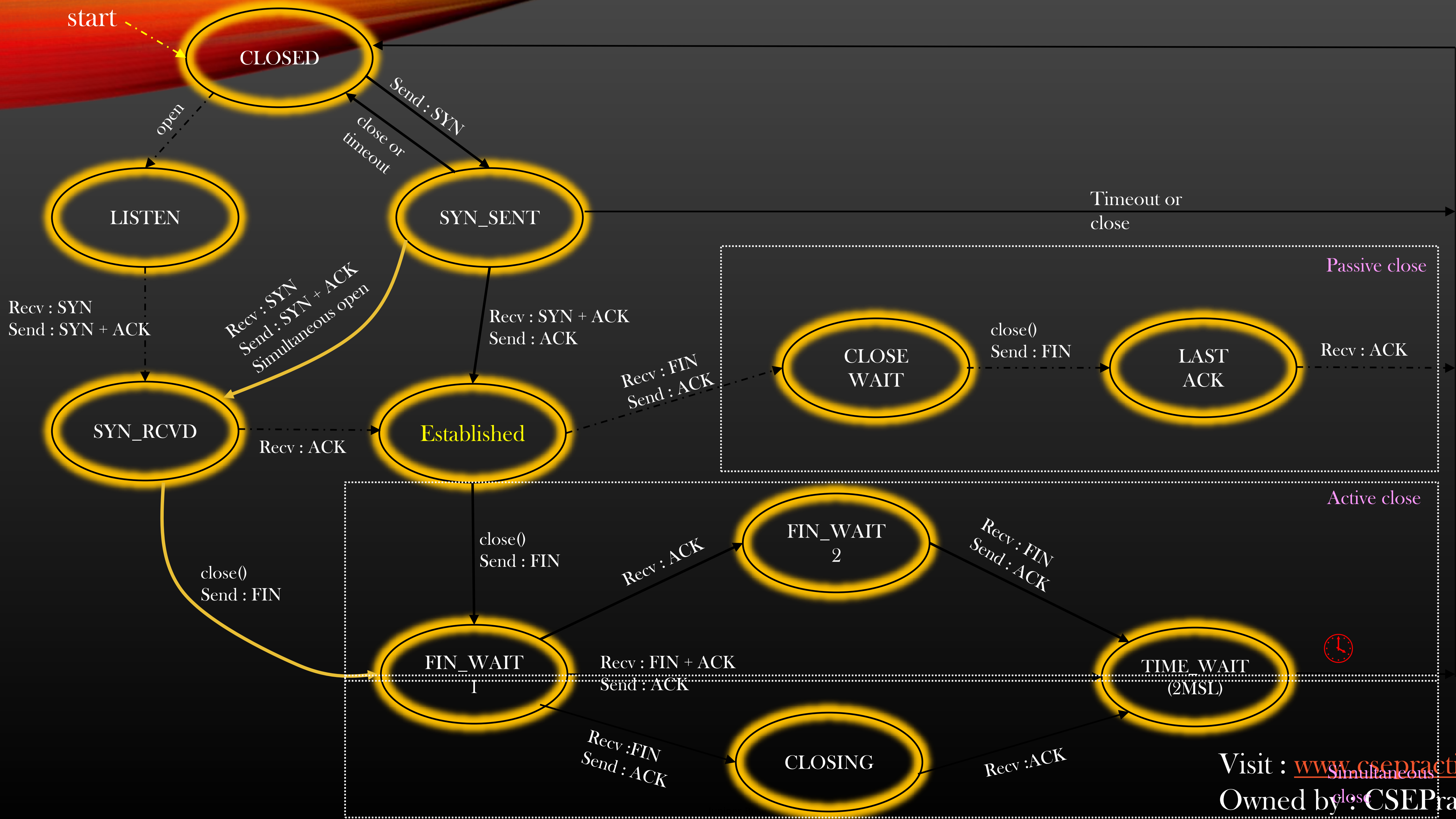
TIME_WAIT – active closer in FIN_WAIT_2 state recvd FIN from peer, sent ACK for it

CLOSED – Active closer's 2MSL timer expired
- passive closer in LAST_ACK state recvs ACK for its FIN

Note : Attached are FSM docs in resource

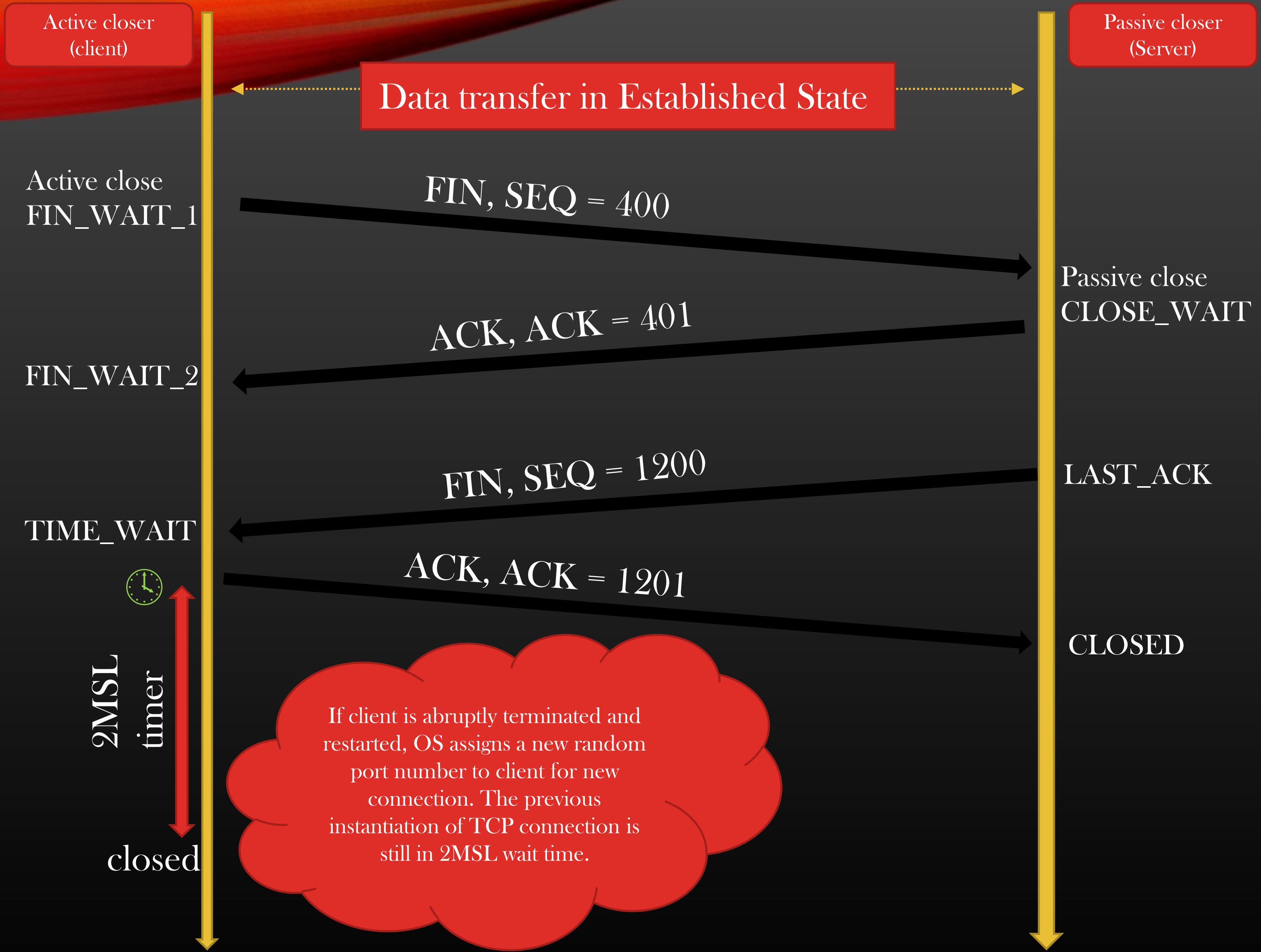
Visit : www.csepracticals.com
Owned by : CSEPracticals

Mastering TCP -> Connection Management -> Finite State Machine



Visit : www.csepracticals.com

Owned by : CSEPracticals



- TCP in FIN_WAIT_2 state when Recvs FIN from peer, enters into TIME_WAIT state where it starts the 2 MSL timer
- IF LAST ACK is lost , passive opener
  RTO timer times out, it resends
  FIN, SEQ = 1200 again
- Reception of FIN on active opener which is in 2MSL wait triggers retransmission of ACK = 1201 and 2MSL timer is reset
- This cycle repeats , this is done to ensure the connection is shutdown from both ends
- It is the active closer which undergo TIME_WAIT state, Passive closer do not
- Servers listen on well known port numbers, eg HTTP Servers on port # 80. If it is the server which did active close, then servers would go in TIME_WAIT state.
- If server which is in TIME_WAIT state, abruptly terminated and restarted, OS assigns it the same port number which it was using before. Since connection is still in 2MSL wait, error flashed : Address already in use
- To recover, you should wait for 2MSL time to restart the Server again successfully.

Visit : www.csepracticals.com
Owned by : CSEPracticals

More about 2 MSL wait time

- The TIME_WAIT state is also called the 2MSL wait state.
- MSL – Maximum Segment Lifetime
- MSL is the maximum amount of time any segment can exist in the network before being discarded
- It s value is commonly set to 30s, 1 min or 2 min.
- Given the MSL value for an implementation, the rule is: *When TCP performs an active close and sends the final ACK, that connection must stay in the TIME_WAIT state for twice the MSL. This lets TCP resend the final ACK in case it is lost*
- The final ACK is resent not because the TCP retransmits ACKs (they do not consume sequence numbers and are not retransmitted by TCP), but because the other side will retransmit its FIN (which does consume a sequence number).
- Indeed, TCP will always retransmit FINs until it receives a final ACK

TCP

Timeout and Retransmission

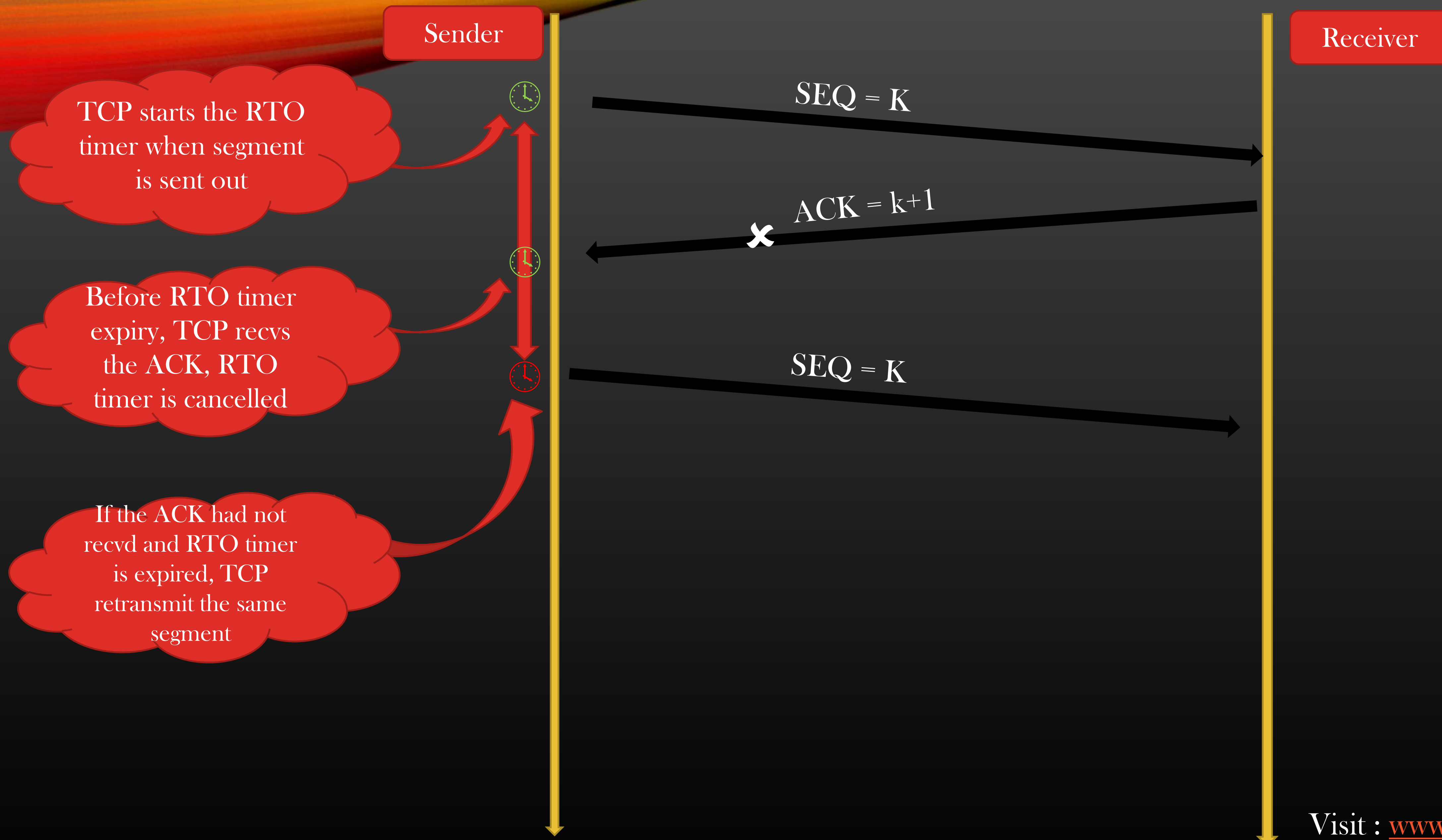
- The TCP protocol provides a reliable data delivery service between two applications using an underlying network layer (IP) that may lose, duplicate, or reorder packets

- In order to provide reliable delivery, TCP resends data it believes has been lost. But how TCP would know the data segments it had sent has been lost ?

- Simple ! TCP sets the timer when it sends data segments and expects an ACK from receiver for this data segment before the timer expires.
  > If ACK arrives before timer goes off, TCP believes the segment has been successfully delivered
  > If Timer goes off and ACK has not arrived yet, TCP assumes segment has been lost and it retransmits the same segment

- The Time interval of the timer is called *Retransmission timeout (RTO)*

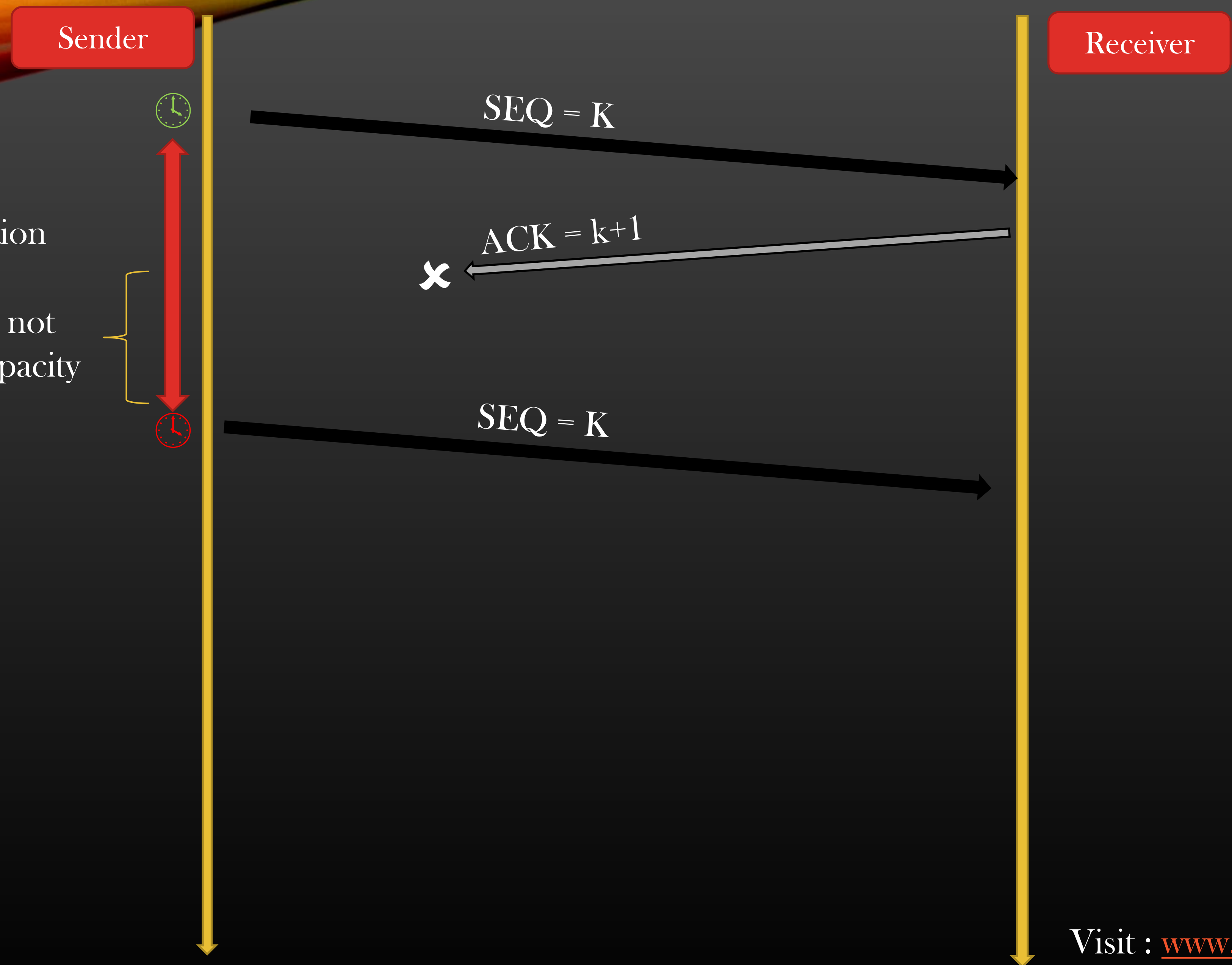
- *Illustration . . .*


Sender
Receiver
TCP starts the RTO timer when segment is sent out
SEQ = K
ACK = k+1
Before RTO timer expiry, TCP recvs the ACK, RTO timer is cancelled
SEQ = K
If the ACK had not recvd and RTO timer is expired, TCP retransmit the same segment

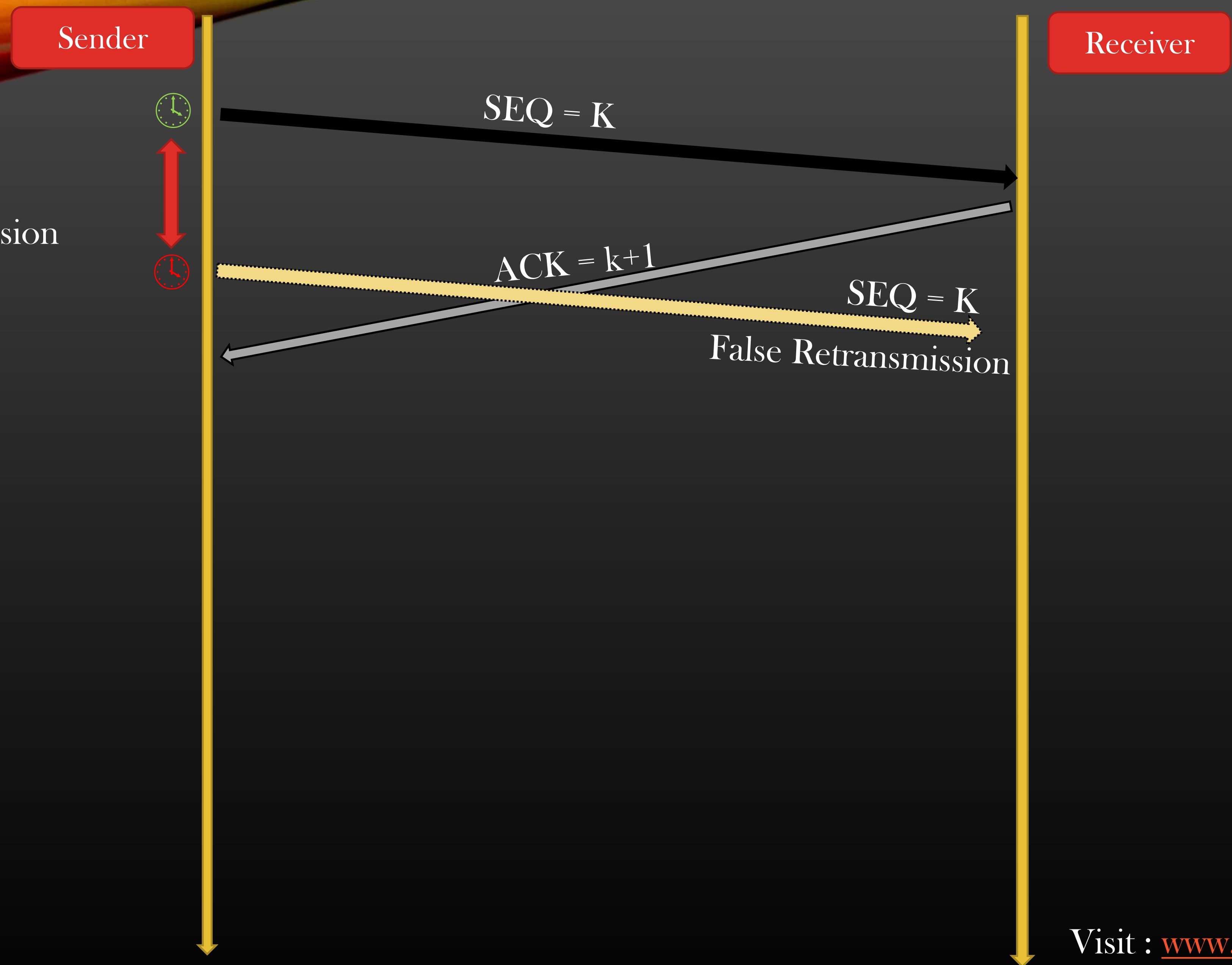
- *Question is :*
  - What should be the appropriate value of RTO ?
    - RTO cannot be fixed because networks are every very dynamic, keep changing over time
    - Intermediate routers routing TCP segments may be slow or fast or congested for some reason
    - Thus RTO value needs to be computed by TCP sender dynamically during the course of its operation, keep updating it constantly as per the network latency and depending on various factors
    - Too large RTO, TCP performance is compromised
    - Too less RTO, false retransmission

Visit : www.csepracticals.com
Owned by : CSEPracticals

- Too long RTO !
- Network Under-Utilization



- Too short RTO !
- Unnecessary retransmission
- Network Congestion



- In this Section of the course we will discuss the retransmission mechanism of TCP which is of two types :
  - *Timer based Retransmission*
  - *Fast retransmission*

On Segment Lost

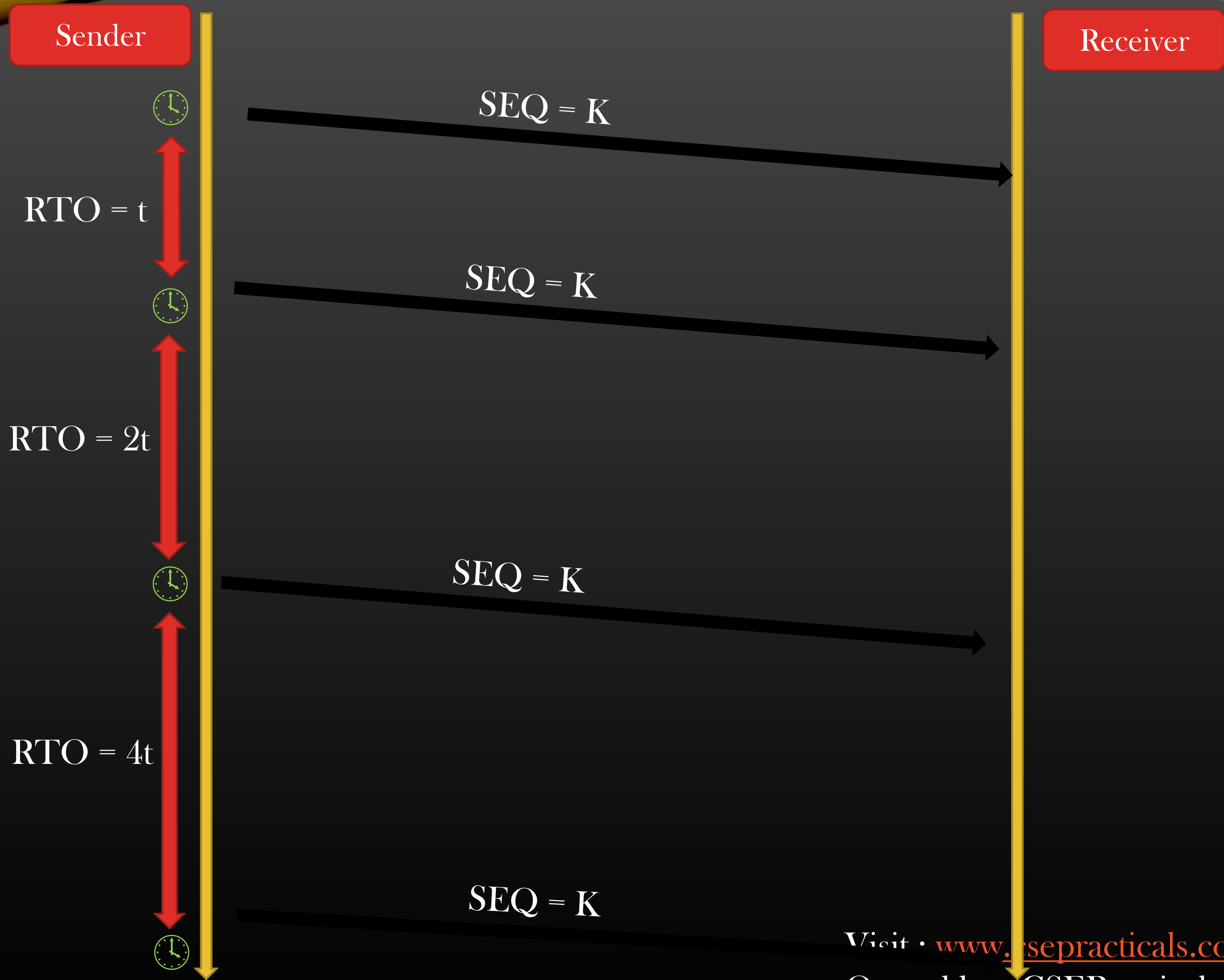
- When TCP sender detects that some segments it had sent has been lost, a choice need to be made by TCP sender whether
  - It has to send more new fresh segments , Or
  - It needs to retransmit lost segments

- Other Questions arises when TCP sender detects segments lost are :
  - Does it has to change segment size for retransmitted segments
  - How many segments to retransmit
  - How RTO is updated to adopt to new network state

Not only just retransmission, In-fact TCP sender has to perform various task to adopt to the Congested Network – Remember loss of segments is the indication of Congested Network to TCP

Exponential back-off

- Doubling the RTO for every consecutive retransmission is called exponential back-off
- Default number of retries TCP sender does to send a segment is 3
- Exponential back-off prevents retransmissions from being sent too quickly and further adding to network congestion



Visit : www.csepracticals.com
Owned by : CSEPracticals

Setting the correct RTO value

- As stated earlier, TCP sender keeps updating the RTO value for a connection dynamically depending on the network state
- RTT – Round-trip time
  - It is the time interval measured when segment is sent, and its ACK is received by the TCP sender
- RTO is measured from sample set of RTTs measured for some previous TCP segments sent by the TCP sender
  - For Example RTO can be calculated from RTTs of previous 10 segments delivered to the TCP receiver
- The RTO is estimated for each TCP connection separately
- RTO tends to be high for congested network, tends to be lesser for fast networks

- RTTs can bounce up and down, so we want to aim for an average RTT value for the connection.
- This average should respond to consistent movement up or down in the RTT, without overreacting to a few very slow or fast acknowledgments.
- To allow this to happen, the RTT calculation uses a smoothing formula:

*New RTT = (x * Old RTT) + ((1-x) * Newest RTT Measurement)*

New RTT: Computed RTT of most Recent Segment
Old RTT: Average RTT of previous N segments
Newest RTT Measurement: Measured RTT of them most recent segment

Where x – smoothing factor between 0 and 1

Higher value of x closer to 1 :

provide better smoothing and avoiding sudden changes as a result of one very fast or very slow RTT measurement. Conversely, this also slows down how quickly TCP reacts to more aggressive changes in RTT

Lower value of x closer to 0 :

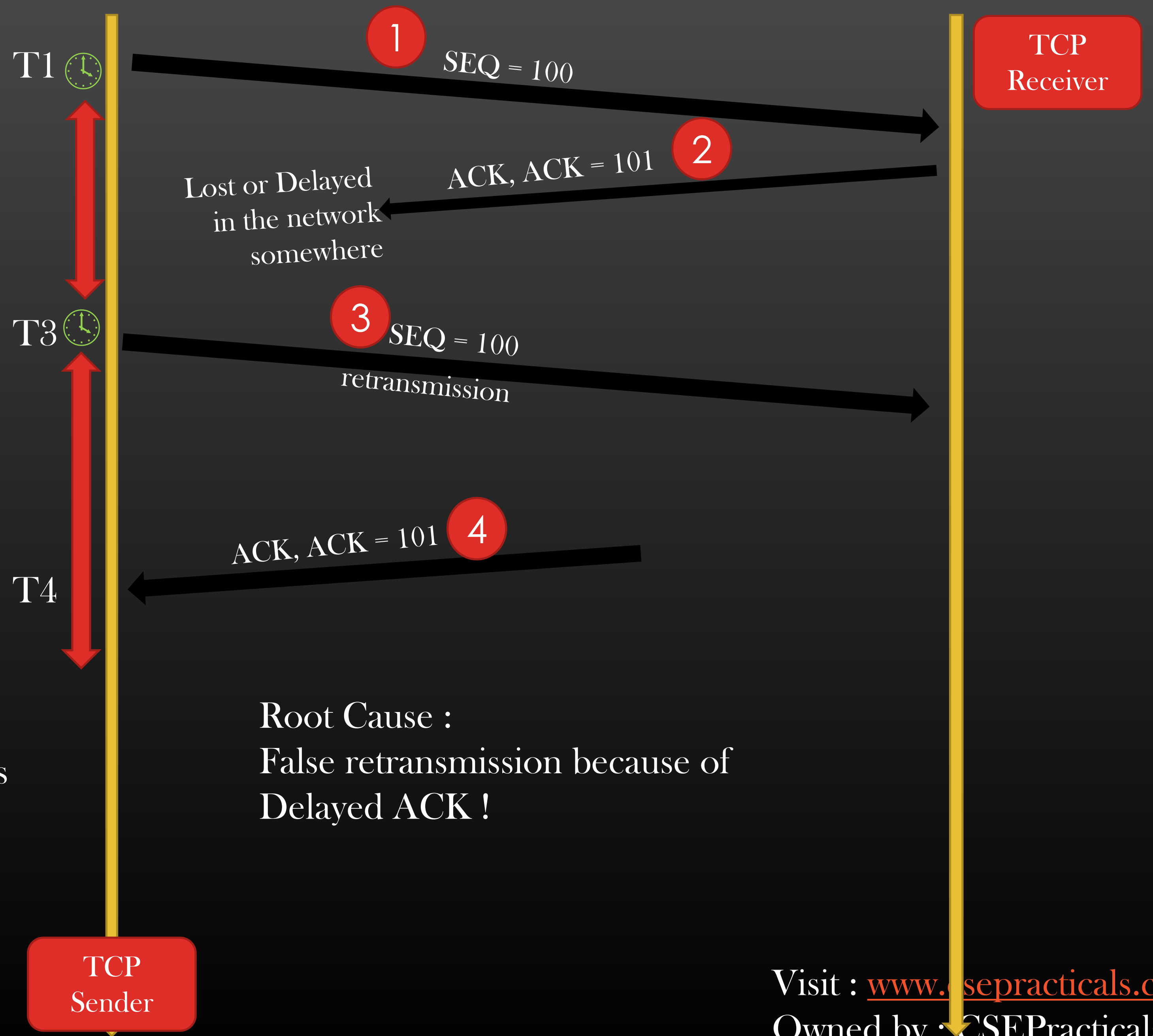
make the TCP to respond more aggressively to changes in measured RTT, but can cause overreaction when RTTs fluctuate wildly

- RTO = Average of RTTs of last 'n' segments sent by TCP sender
- RTO is set to 1s when TCP connection just starts and it do not have any historical RTTs sample to compute RTO

Visit : www.csepracticals.com
Owned by : CSEPracticals

Retransmission Ambiguity Problem

- 2 is the ACK triggered by 1
- TCP sender has no way to determine whether the ACK 4 is :
  - same as 2
    - RTT1 = T4 – T1
  - New ACK triggered by packet 3
    - RTT2 = T4 – T3
- The computed RTT would impact the RTO value of the TCP connection
- There is no way TCP sender can make a decision whether to choose RTT1 or RTT2 as RTT for segment 1
- This is Ambiguity !
- Solution : Karn's Algorithm



Visit : www.csepracticals.com
Owned by : CSEPracticals

Karns Algorithm

- TCP's solution to Retransmission Ambiguity Problem is based on the use of a technique called *Karn's algorithm*, after its inventor, Phil Karn

- Karn's Algorithms has two parts :
  - *Ignore measured RTT for retransmitted segments for RTO evaluation*
    - Because measured RTT for retransmitted segments would skew the RTO incorrectly, throw away the unreliable data
    - This solves the problem of retransmission ambiguity
    - But at the same time, this will prevent sending TCP to take corrective measures to segment losses which is potentially due to network congestion breaking the main strength of TCP – Adoptive transmission

  - *Use back-off RTO for retransmitted segments and do not consider their measured RTT for RTO evaluation*
    - subsequent retransmission timers are double the previous
    - The back-off factor is not reset until there is a successful data transmit that does not require a retransmission

- Best to understand with the help of example ! ☺
  - Advice : Read the above statements again after going through the example !

Karns Algorithm example

- Refer to Separate Doc

Karns Algorithm Analysis

In our Example, Karns Algorithms performed these three major tasks :

- Every-time the Segment was retransmitted, RTO was doubled of the previous
  - Segment with SEQ = 150 was retransmitted 2 times with RTO of 4s and 8s respectively
  - This exponentially slows down TCP from further congesting the already congested network

- RTT measurement of retransmitted segments was not used for RTO evaluation
  - When TCP Sender recvd ACK 200, it did not consider the RTT of segment with SEQ = 150 for RTO evaluation

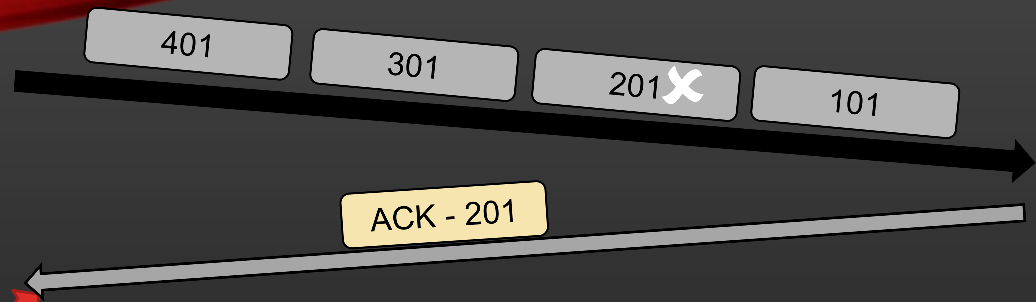
- When TCP sender is able to send TCP segments without having to retransmit it, inflated RTO value was restored to original. RTT of this segment was considered for RTO evaluation
  - RTO was updated from 8s back to 2s straightaway
  - Successful transmission of Segment with SEQ = 200 in first attempt is an indication of network recovery, so it helps TCP Sender to restore its rate of sending data to recvr, Network Must not left under-utilized

Fast Retransmission

- We learnt how TCP depends on Timer to detect that segment has been lost and re-trigger the lost segment
- But, Timer based re-transmission often leads to under utilization of network capacity
  - Sender has to sit idle waiting until RTO timer expires, segment many have lost long before
- Therefore, Now we shall discuss another strategy in which TCP sender do not have to depend on Timer for segment loss detection and retransmission called – *Fast Retransmit*
- It is called Fast Retransmit because TCP sender almost immediately detect the segment loss and retransmit it instantly. This is much more efficient than *Timer based retransmission* scheme

Fast Retransmission

- End Goal is same : Retransmit the lost segment, only difference is in the methodology of how to detect that segment has been lost
- In Fast Retransmit, TCP sender triggers segment retransmission based on feedback from receiver rather than relying on Retransmission timer expiry, hence segment loss repair is even quicker
- A typical TCP implementation implements both FAST retransmit and timer based retransmission strategy
- Let us start with the discussion with what does TCP receiver do when it receives segments out of order



1-100									

1-100	101-200	H	301-400	401-500						

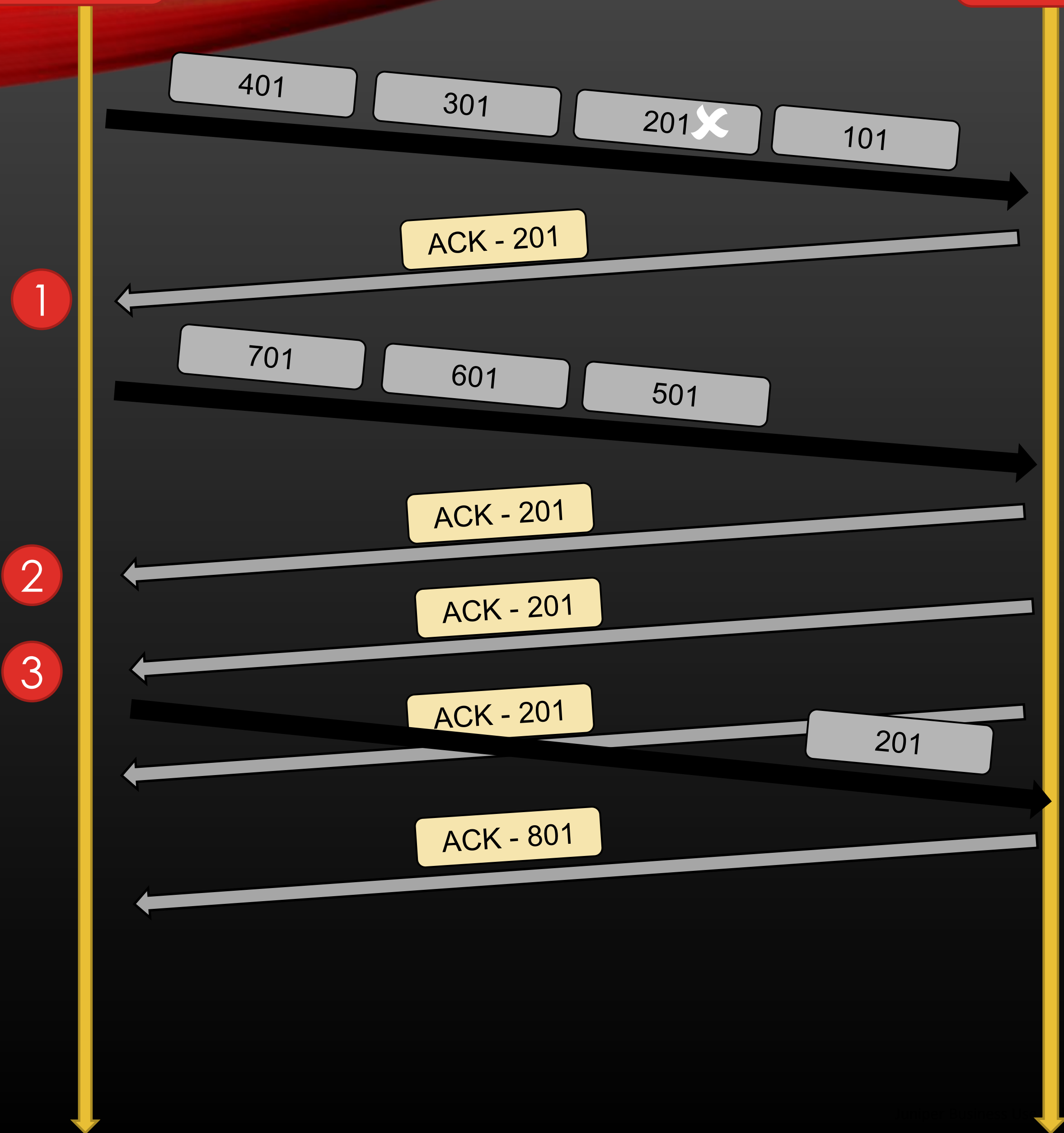
What Should TCP sender do when it recvs this ACK ? Will it send DS 201 ?

TCP Sender retransmist the segment 201 only when it recvs 3 ACK 201 !!

- TCP Recvr Do not like Holes in its recv buffer
- TCP recvr accepts the out of order bytes but,
- TCP recvr sends an ACK in order to fill the Holes first before demanding new fresh segments

Visit : www.csepracticals.com
Owned by : CSEPracticals

Mastering TCP -> Timeout and Retransmission -> Fast Re-Transmit ->out of order reception of segments



TCP Receiver

1-100									

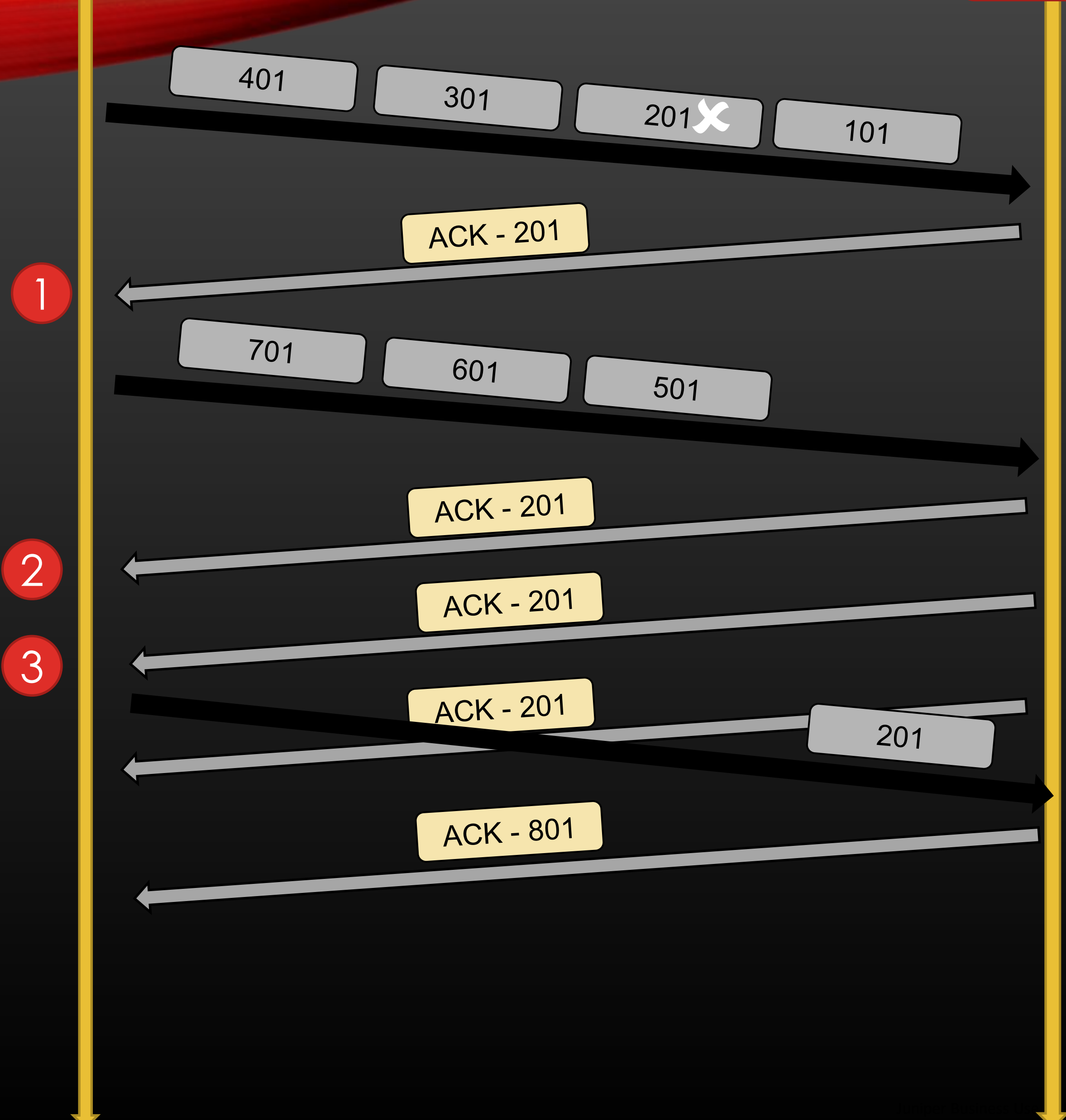
1-100	101-200	H	301-400	401-500						

1-100	101-200	H	301-400	401-500	501-600	601-700	701-800			

1-100	101-200	201-300	301-400	401-500	501-600	601-700	701-800			

Visit : www.csepracticals.com
Owned by : CSEPracticals

Mastering TCP -> Timeout and Retransmission -> Fast Re-Transmit ->out of order reception of segments



- Every time the TCP Receiver recvs out of order segments, it triggers the same ACK to fill the first hole immediately
- TCP sender sees the same ACK again and again, therefore they are called duplicate ACK (ACK = 201)
- This make TCP sender conclude that segments are being recvd by TCP receiver out of order or probably some are even lost
- Duplicate ACK tells the Sender the Ist hole in Receiver buffer
- When TCP sender Receives the 3 consecutive duplicate ACK with same ACK#, Sender retransmit the segment
  - Duplicate threshold (dupthresh) = 3

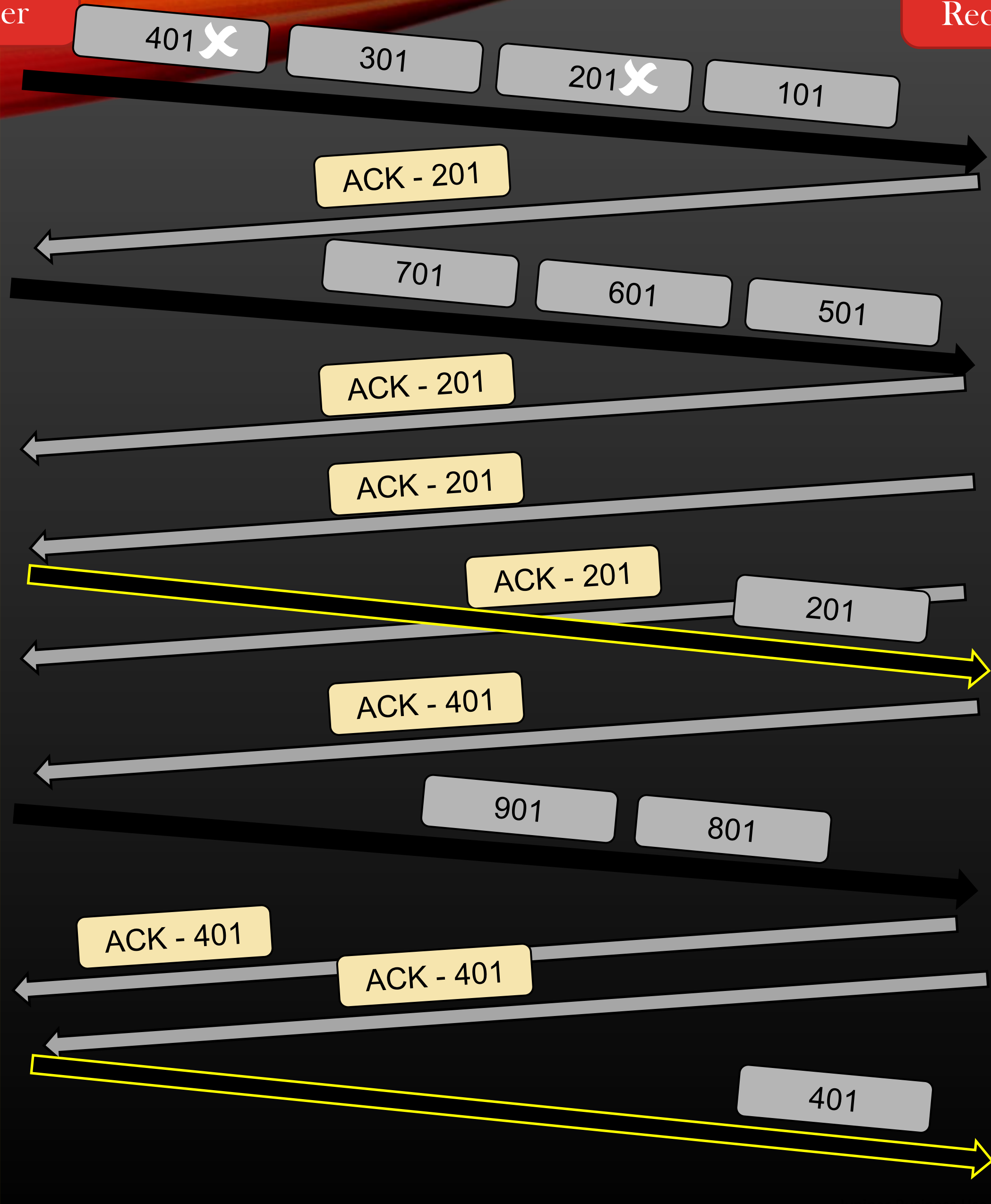
1-100	101-200	201-300	301-400	401-500	501-600	601-700	701-800			

- One Problem here – Using dupACKs TCP sender can repair only one hole in TCP recvr's buffer at a time
- One more Example !

Visit : www.csepracticals.com
Owned by : CSEPracticals

Mastering TCP -> Timeout and Retransmission -> Fast Re-Transmit ->Multiple Holes Repair

Assumption
Segment size = 100B



1-100									

1-100	101-200	H1	301-400	H2					

1-100	101-200	H1	301-400	H2	501-600	601-700	701-800		

Hole Repaired

- Holes are repaired one by one
- TCP recvr has to send 3 dupACK to repair each hole
- Is there anyway to repair all holes in one go ?
- Yes – Selective ACKs !!

1-100	101-200	201-300	301-400	H2	501-600	601-700	701-800		

1-100	101-200	201-300	301-400	H2	501-600	601-700	701-800	801-900	901-1000

Hole Repaired

1-100	101-200	201-300	301-400	401-500	501-600	601-700	701-800	801-900	901-1000

- We observed , dupACK enable TCP to fast retransmit the lost segments !
- This is Great Optimization
- But this Optimization comes at the cost !!
- I have intentionally not shown the penalty paid by TCP to implement fast retransmit through dupACK in previous example – to simplify the example
- Let us discuss the disadvantage of *fast-retransmit-through-dupACK* –

Redundant Retransmissions

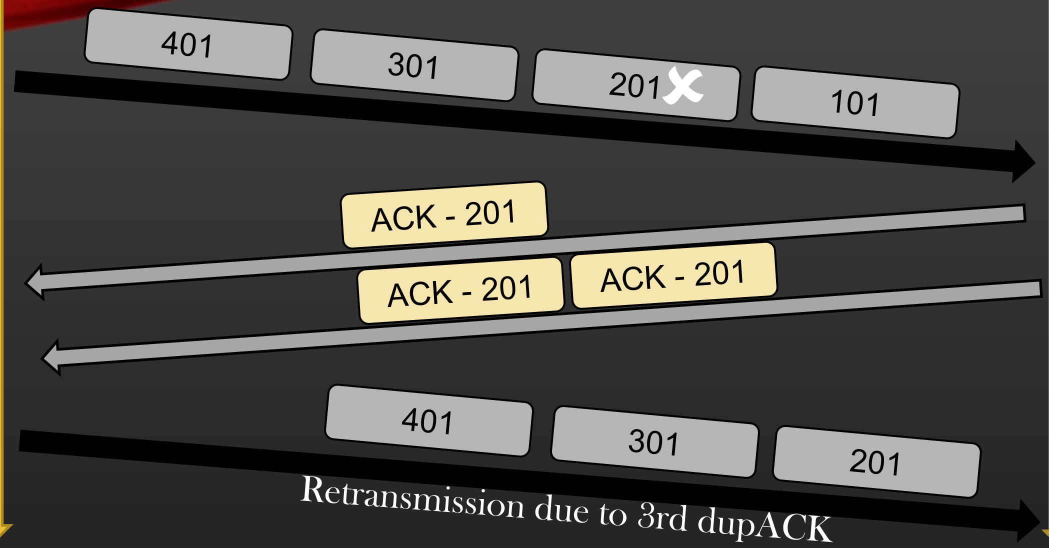
Mastering TCP -> Timeout and Retransmission -> Fast Re-Transmit ->Redundant retransmissions

TCP Sender

Assumption
Segment size = 100B

TCP Receiver

Case 1

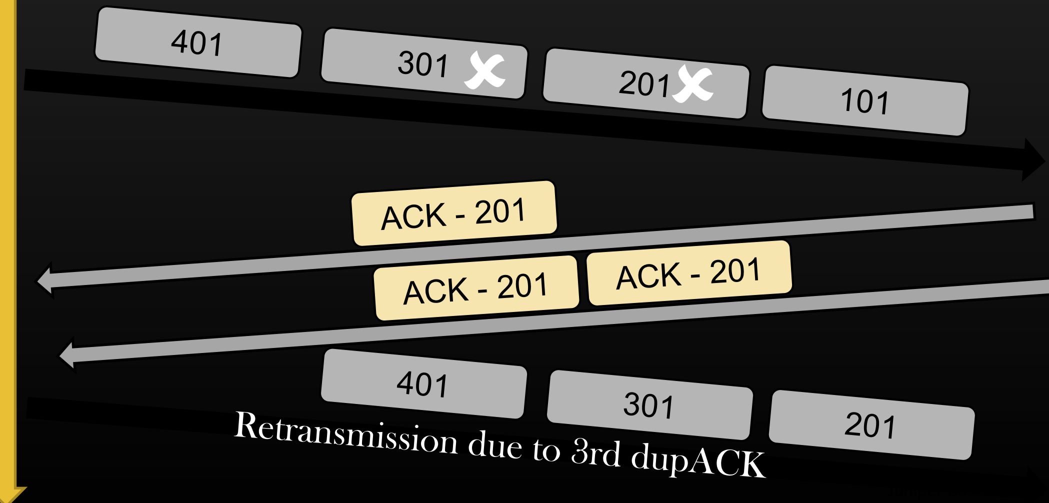


1-100									

1-100	101-200	H1	301-400	401-500						

- Both the cases are identical to TCP Sender
- ACK 201 tells TCP senders that recvr has recvd bytes upto seq no 200 only, beyond that TCP sender do not know any thing about rest of the bytes [202--500], byte 201th is definitely not recvd by Receiver
- In case 1, Segment no 301 and 401 are unnecessary transmitted
- In case 2, Segment no 401 is unnecessarily retransmitted

Case 2



1-100	101-200	H1	H2	401-500						

- In general, dupACK triggers retransmission of lost segment instantly (Advantage), but also triggers retransmission of already transmitted unlost segment (Disadvantage) – Solution – Again SACKs!!

Visit : www.csepracticals.com
Owned by : CSEPracticals

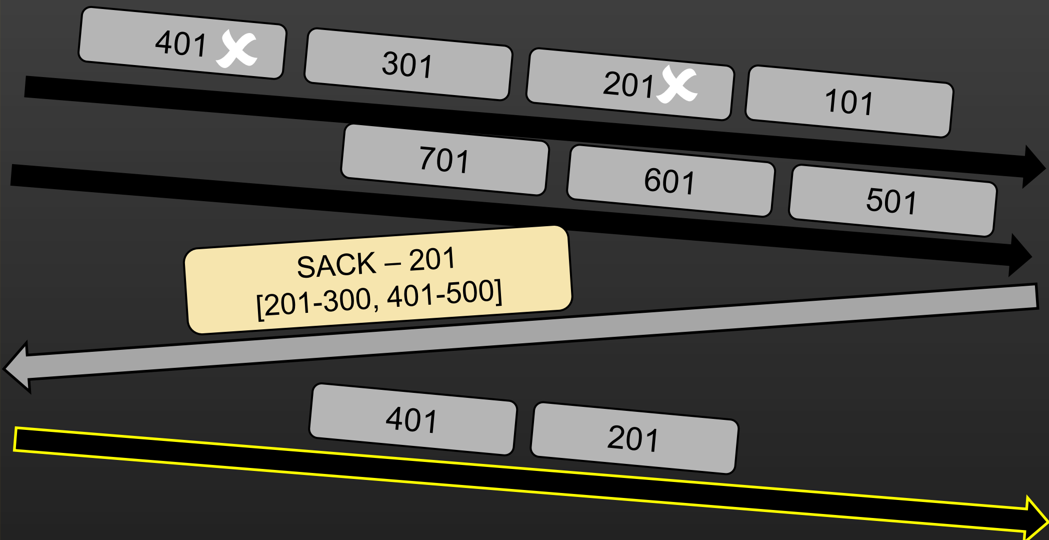
Timer based Retransmission	Fast Retransmission
Segments is retransmitted when Retransmission timer expires	Segments is retransmitted instantly When Sender receives 3 dupACK
Result in Sender to sit idle for some time	Instant retransmission of lost/OOO segments
Network Under Utilization	Network optimal utilization
No redundant retransmission of Segments	dupACK leads to redundant retransmission of segments, Network bandwidth wastage, contribution to congestion etc.

Mastering TCP -> Timeout and Retransmission -> Fast Re-Transmit -> Select Acknowledgements

TCP Sender

TCP Receiver

Assumption
Segment size = 100B



1-100									

1-100	101-200	H1	301-400	H2	501-600	601-700	701-800		

Holes Repaired

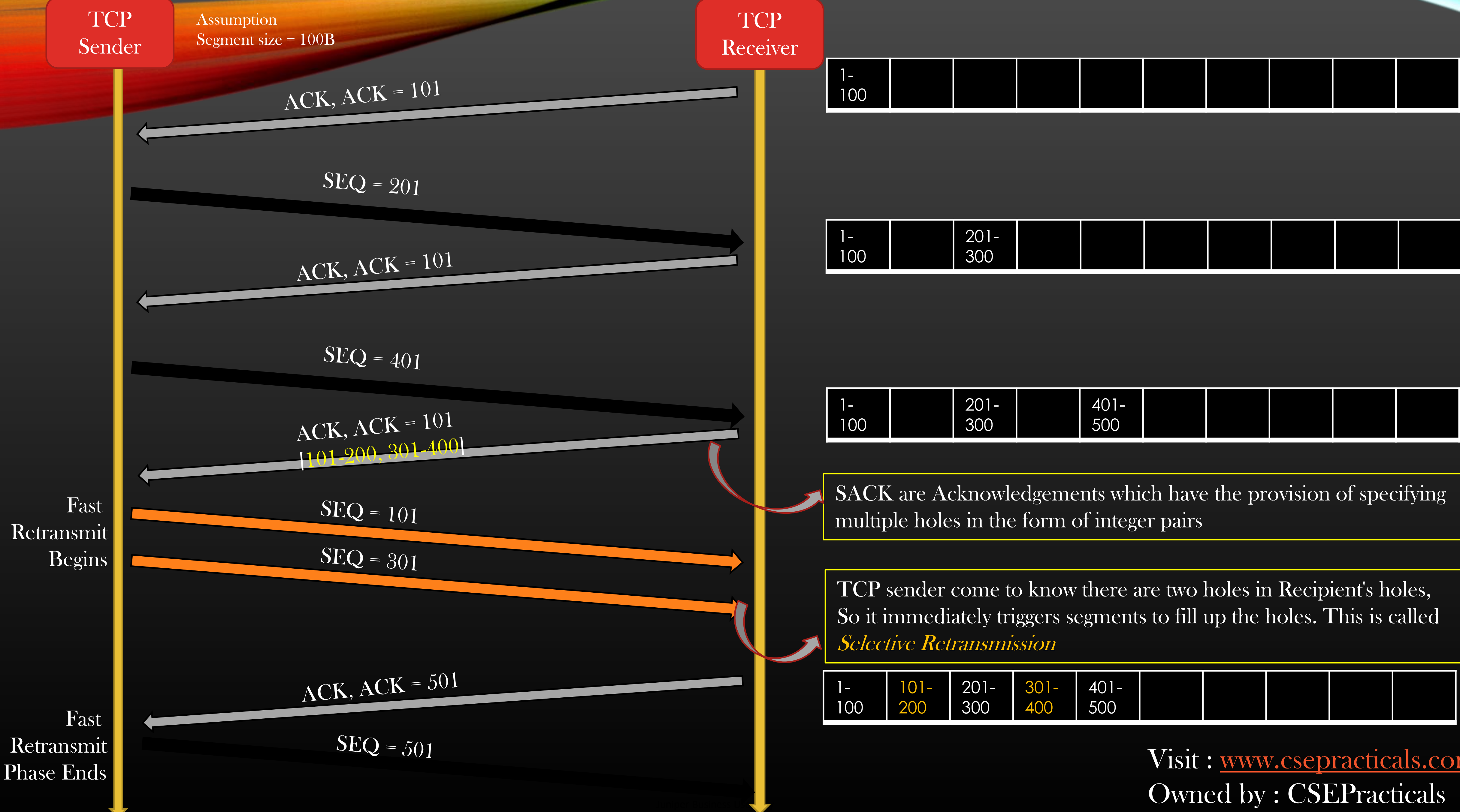
Hole Repaired

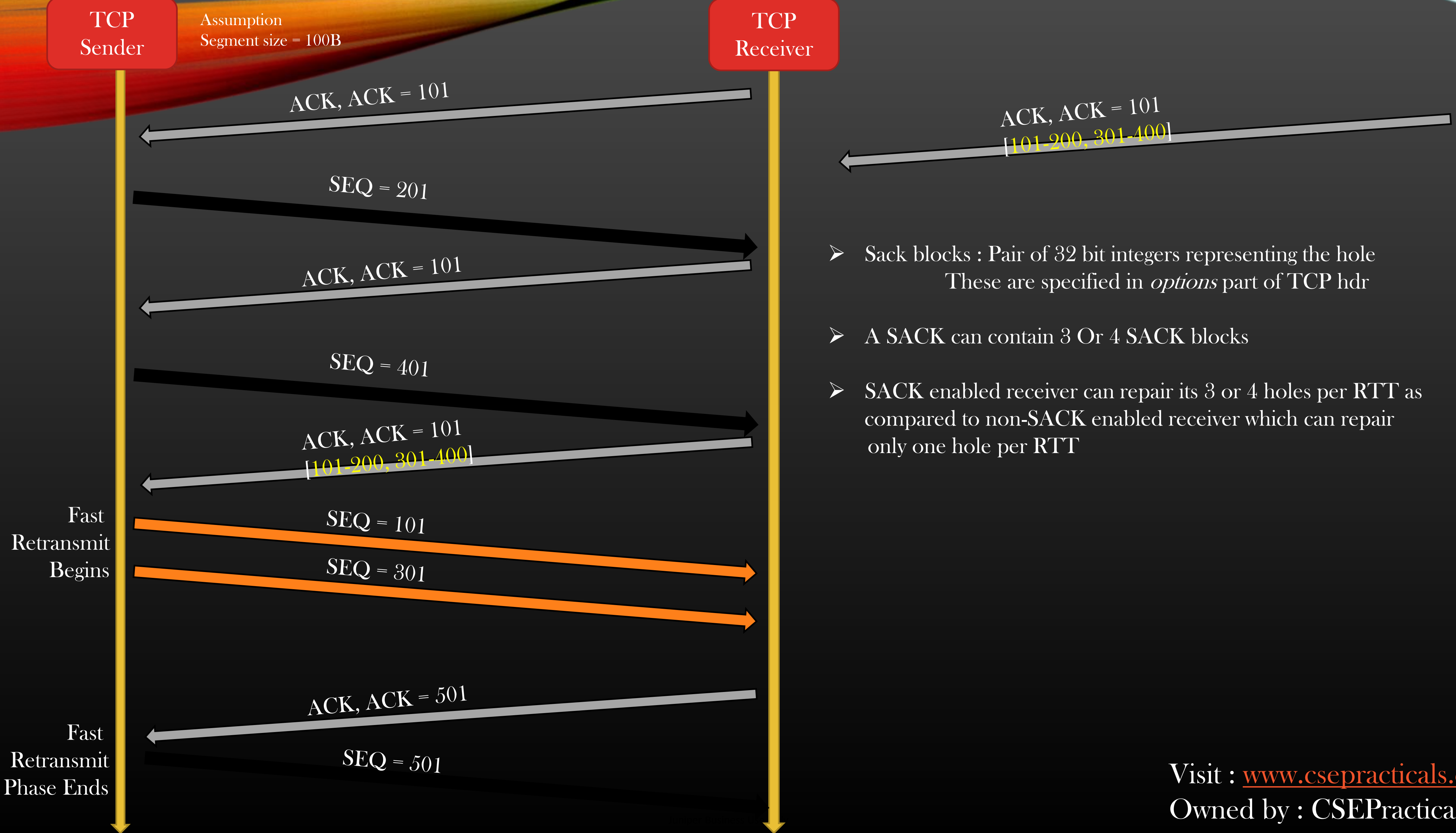
1-100	101-200	201-300	301-400	401-500	501-600	601-700	701-800		

- Recvr sends SACK when it have multiple holes in its recv buffer
- SACK specify the multiple holes in the form of L,U
- TCP Sender process all L,U pairs in SACK, and retransmit the segments accordingly
- No need of duplicate ACKs !

Visit : www.csepracticals.com
Owned by : CSEPracticals

Mastering TCP -> Timeout and Retransmission -> Fast Re-Transmit ->Select Acknowledgements




Mastering TCP -> Timeout and Retransmission -> Fast Re-Transmit ->Select Acknowledgements
TCP Sender
Assumption
Segment size = 100B
TCP Receiver
ACK, ACK = 101
SEQ = 201
ACK, ACK = 101
SEQ = 401
ACK, ACK = 101
[101-200, 301-400]
Fast Retransmit Begins
SEQ = 101
SEQ = 301
ACK, ACK = 501
Fast Retransmit Phase Ends
SEQ = 501
ACK, ACK = 101
[101-200, 301-400]
Sack blocks : Pair of 32 bit integers representing the hole
These are specified in options part of TCP hdr
A SACK can contain 3 Or 4 SACK blocks
SACK enabled receiver can repair its 3 or 4 holes per RTT as compared to non-SACK enabled receiver which can repair only one hole per RTT
Visit : www.csepracticals.com
Owned by : CSEPracticals

TCP

Data Flow

And

Window Management

Questions Answered in this section :

- What should be the Segment size ?
- How many segments a TCP sender is allowed to send to Receiver in one go ?
- How many Segments can receiver receive in its receiving buffer ?
- How TCP sender knows the capacity of TCP receiver to receive and process segments ?

- In this section, we shall explore the Sliding Window Mechanism used by TCP which is used to achieve:
  - Reliable data delivery
  - Congestion and flow control
    - Managing the rate at which data is sent so that it does not overwhelm the device that is receiving it Or network

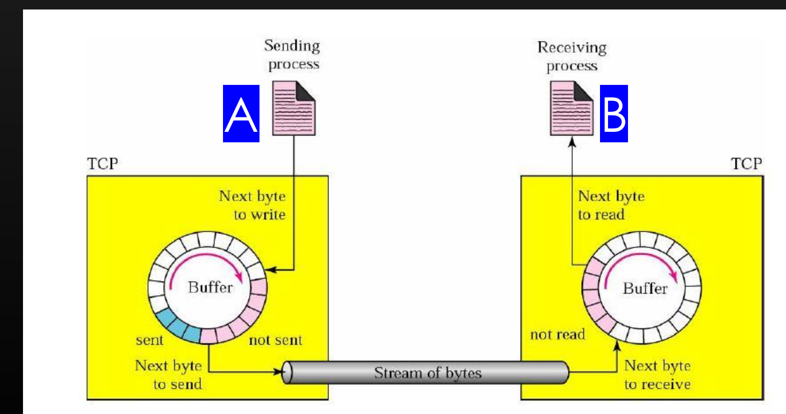
- Remember TCP connection is a duplex communication, therefore both the parties are both senders and receivers

- In Diagram, the Sending Process A has a circular buffer which is called a send window. Similarly, Receiving process B also has a circular buffer which is called a recv window

- Since Sending process A is also a receiving process B for byte stream flowing from B to A, Process A also maintains a recv window and process B also maintains a send window (Not shown in the diagram)

- Thus both processes A and B has both
  - Send Window
  - Recv Window

- Send Window of A is paired up with recv window of B
- Recv Window of A is paired up with Send Window of B
- The Send window of one device is the recv window of other and vice versa



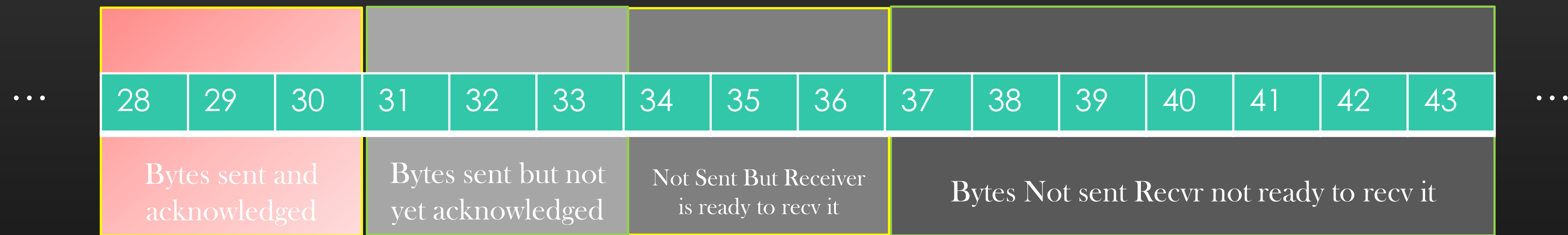
...als.com
Owned by : CSEPracticals

- TCP is a sliding Window Protocol, meaning it manages its flow control, congestion control, reliable data delivery by managing its send/recv windows

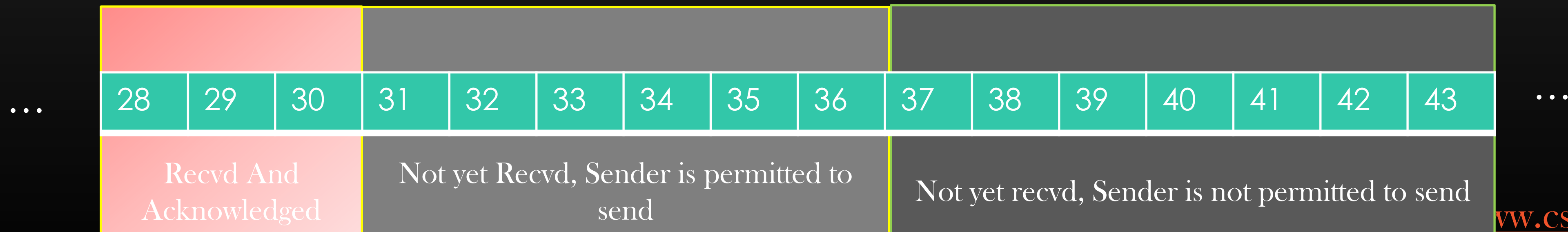
- TCP Send Window (4 Categories)

  - We can take the snapshot of the TCP send Window at any point of time, and classify the bytes in 4 categories as per the diagram below. Remember TCP is byte oriented protocol, it keeps track of data flow at byte level not segment level

  - At any given point of time, We can classify the bytes of data in Send Window of TCP sender into four categories



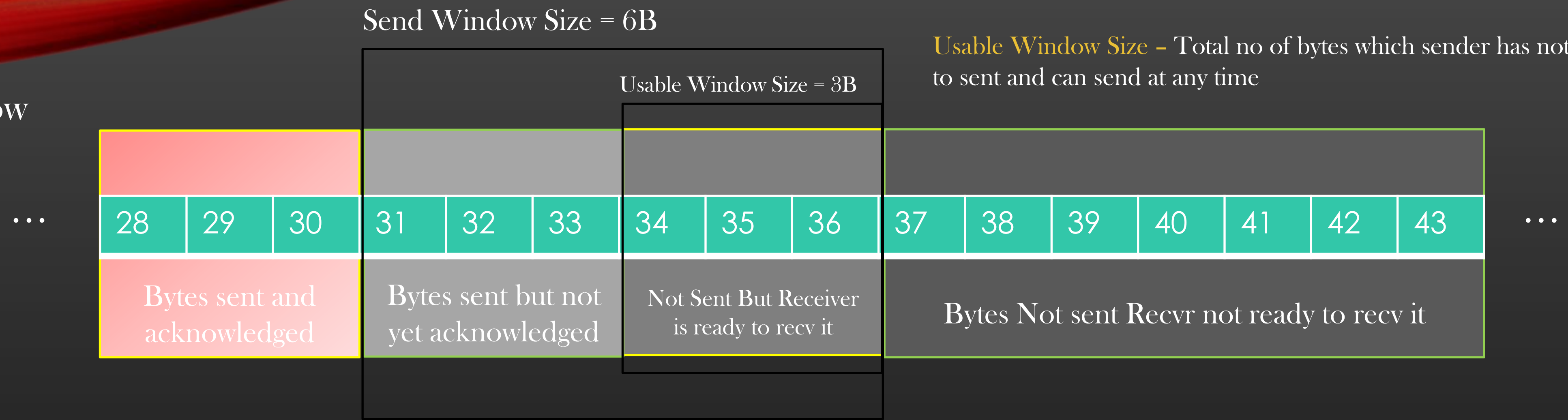
- TCP Recv Window (3 Categories)



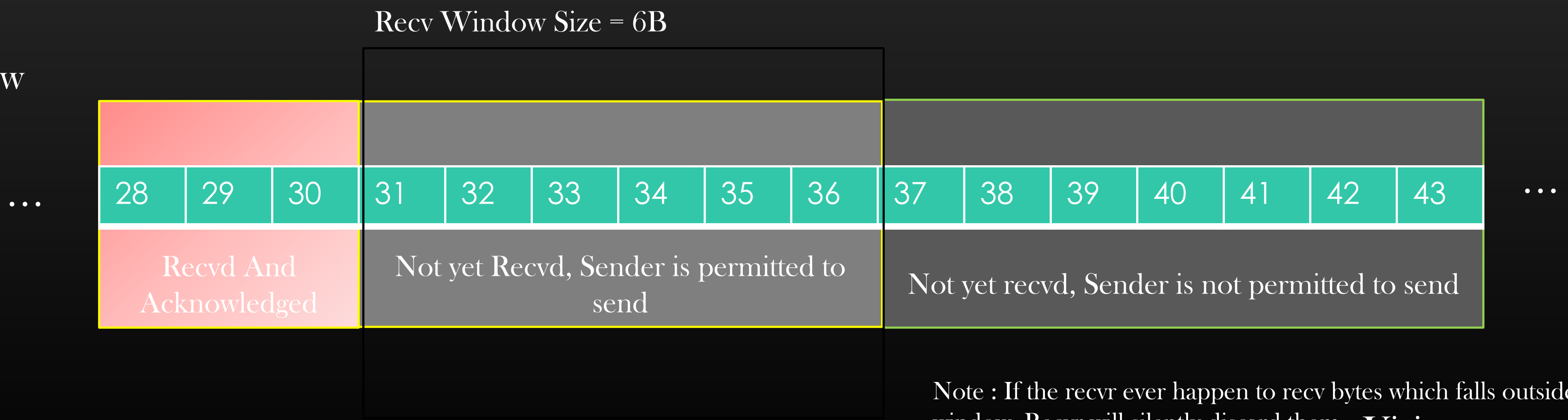
Send Window Size – Total Number of Bytes Which sender can send (sent + not sent but ready to sent)

Usable Window Size – Total no of bytes which sender has not sent but ready to sent and can send at any time

Send Window



Recv Window



Note : If the recvr ever happen to recv bytes which falls outside the recv window, Recvr will silently discard them

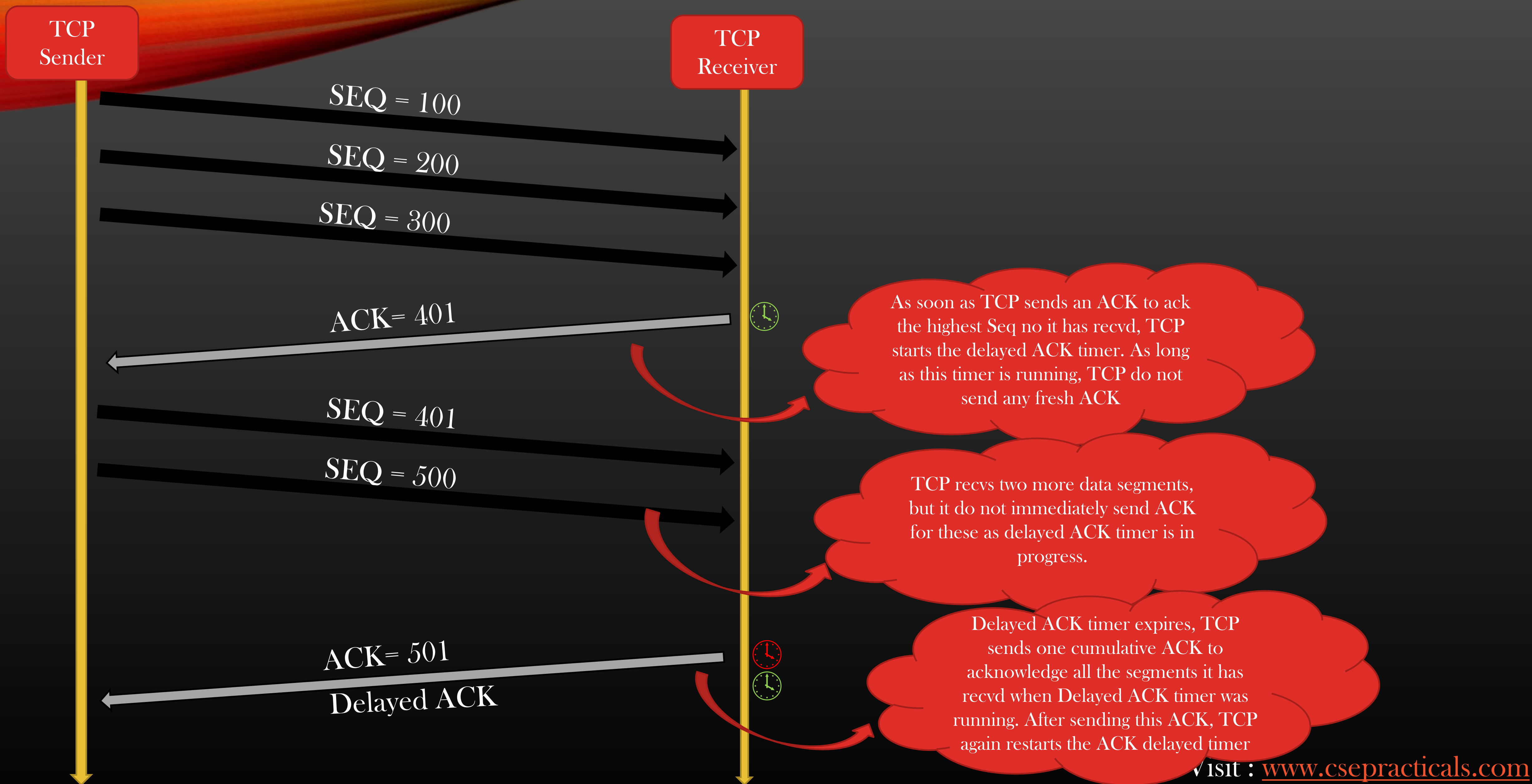
Visit : www.csepracticals.com
Owned by : CSEPracticals

- TCP Acknowledgement Number is the mechanism which TCP recvr uses to tell the TCP sender *how many bytes it has received, and what it expect next*
- TCP receiver DO NOT send ACK for every segment or byte of data it receives – Highly inefficient
- Acknowledging every byte by Receiver will trigger too many ACK segments, if this happens then TCP hdr overhead (useless data) consumes more network bandwidth and resources than TCP payload (useful data)
- When TCP receiver recvs too many segments in quick succession , it acknowledges all of them by single ACK

- Of-course, TCP cannot delay the cumulative ACK segments indefinitely otherwise it will trigger unnecessary retransmission
- Cumulative ACKs(also called Delayed ACKs) Causes less traffic

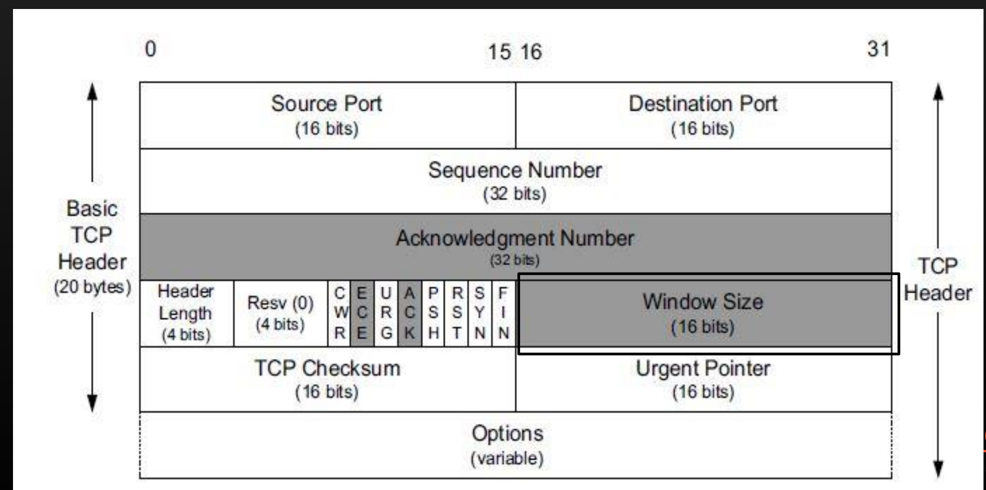
Visit : www.csepracticals.com
Owned by : CSEPracticals

Mastering TCP -> Cumulative Acknowledgement

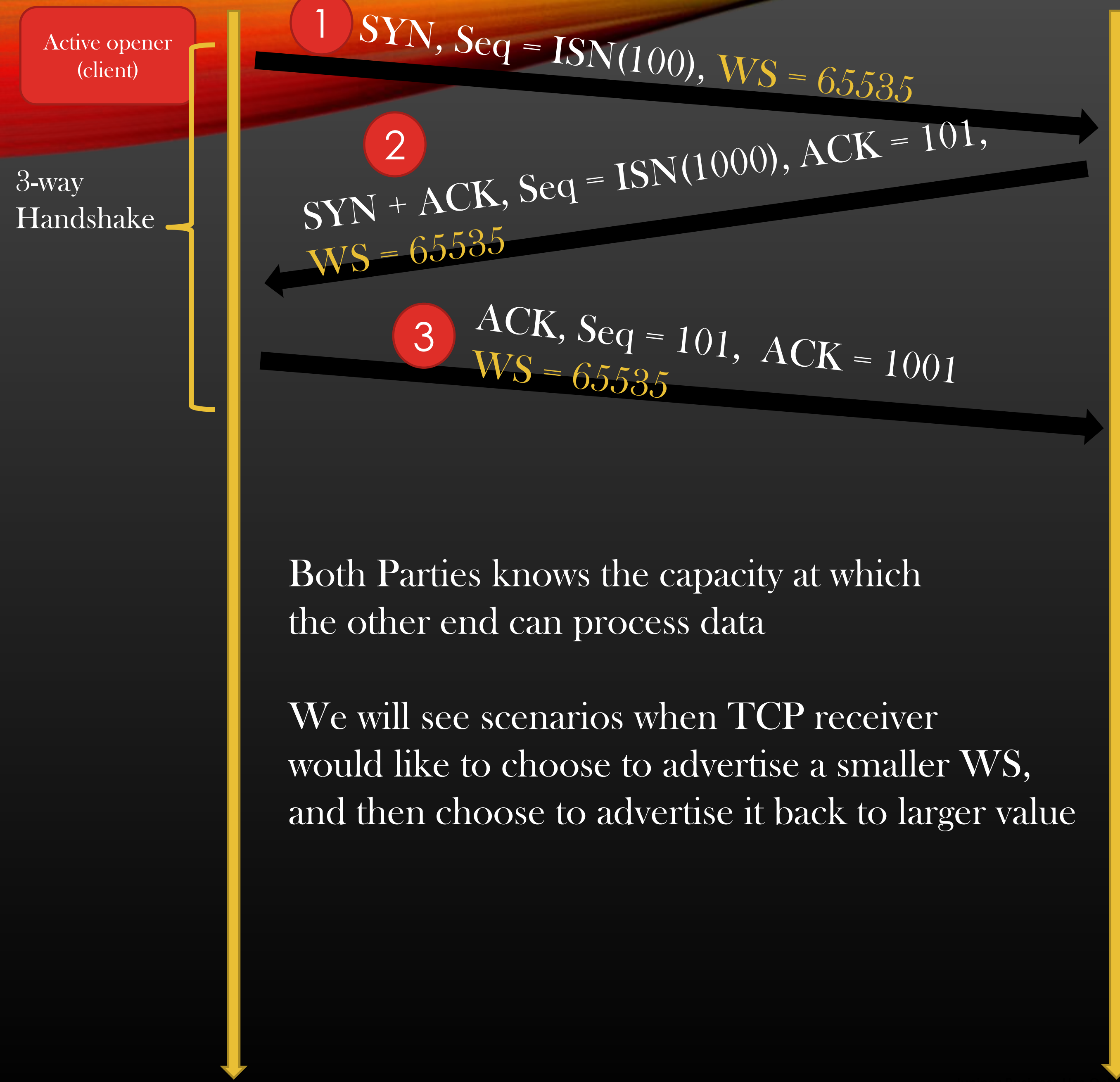


Visit : www.csepracticals.com
Owned by : CSEPracticals

- The TCP receiver advertise the size of its recv window in every ACK that it sends to the TCP sender
- TCP sender having recvd this advertisement sets the size of its send window to the value advertised by recvr
- By definition, Send Window determines the no of bytes the TCP sender can send in one go
- Thus , TCP receiver controls the size of TCP sender's Send window, this controls the rate at which the TCP sender can send the data to Receiver – This is called *Window based flow control*
- Overwhelming/congested TCP Receiver tends to reduce its recv window size and advertise reduced size of its recv window in ACK to TCP Sender, thus, mitigating the congestion
- Both Peers Advertise the size of their respective TCP Recv Window to other during TCP connection establishment phase - three way handshake
- TCP hdr format ->
- Window Size = 16 bit



com
Owned by : CSEPracticals



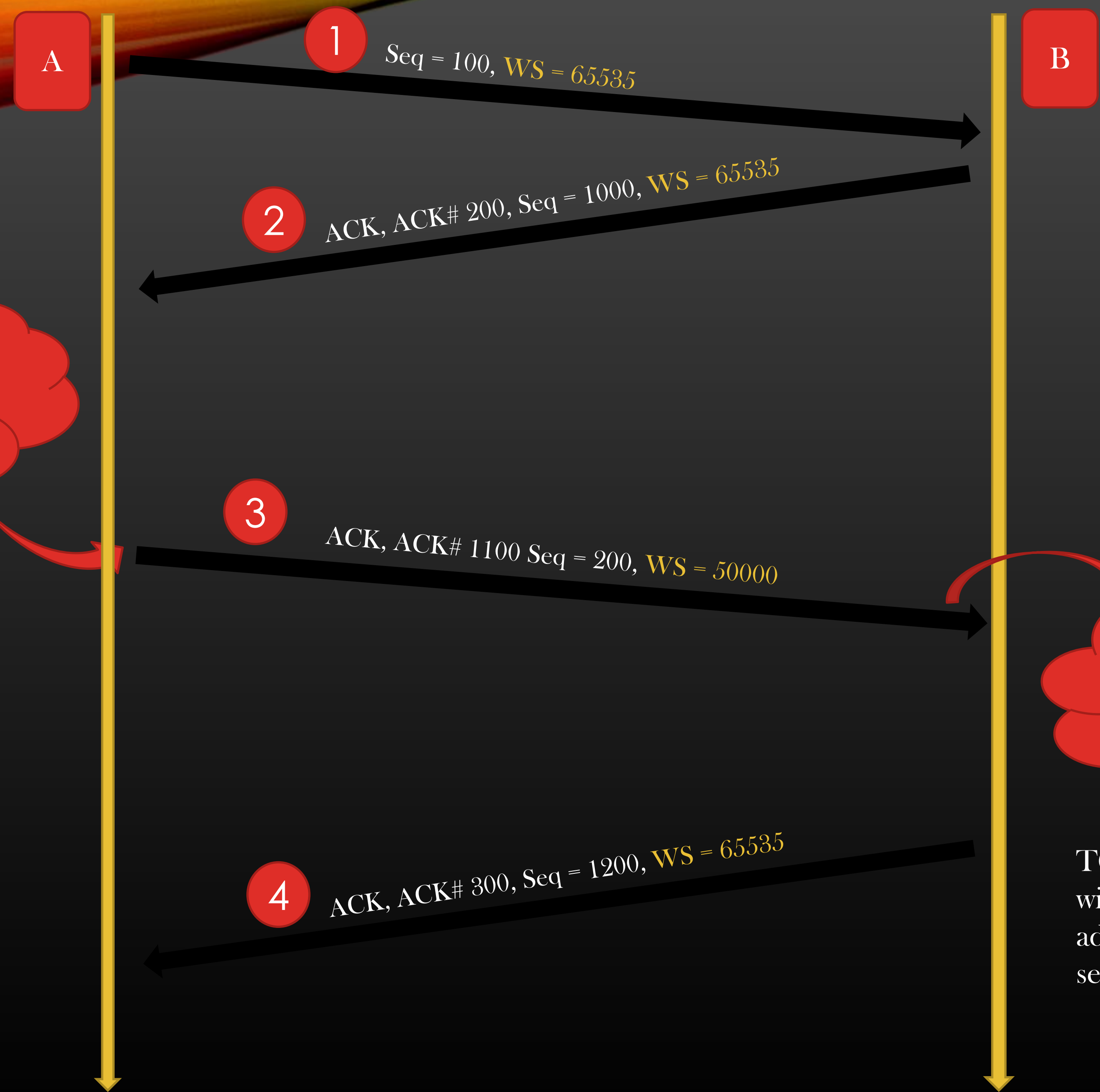
Passive opener (Server)

1. SYN – want to initiate a TCP connection
   Advertise the recv Window Size = 65535,
   Since network Or peer's state/capacity is not known

2. SYN + ACK
   Also Advertise the recv window Size = 65535

3. ACK – handshake complete
   WS is advertised in all TCP segments

Visit : www.csepracticals.com
Owned by : CSEPracticals



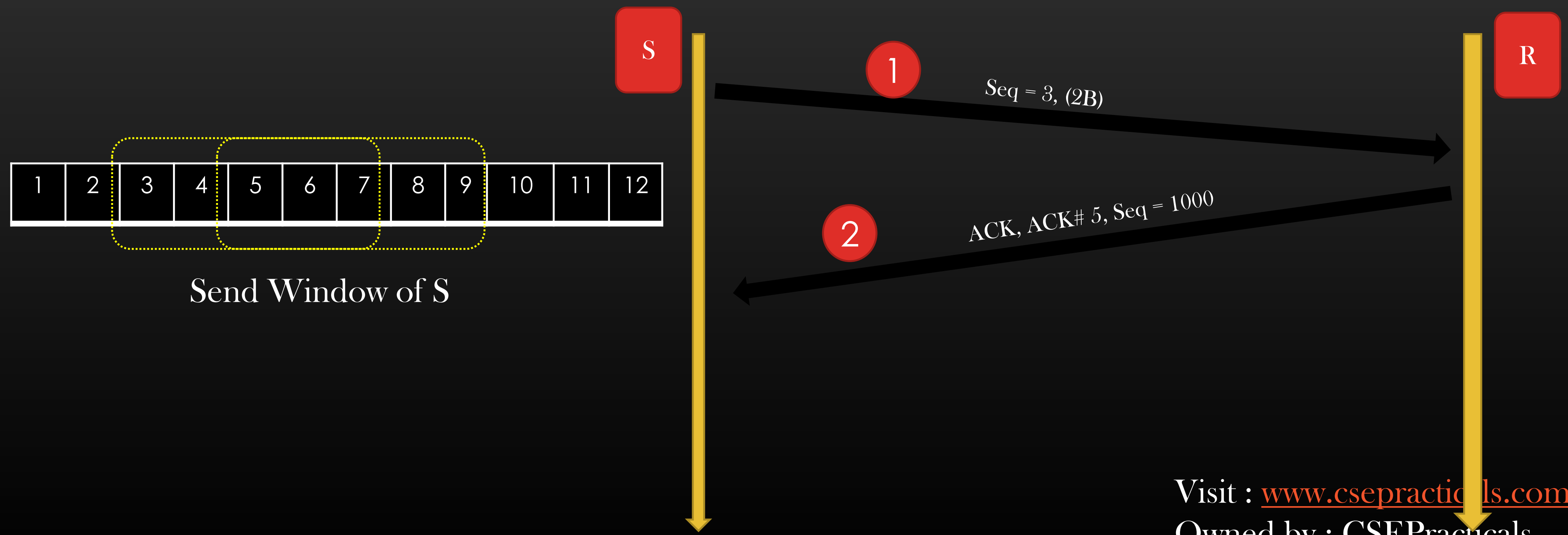
TCP Sender shrinks Or expands its send window size depending On window size advertisement in the most recent recvd TCP segment

Visit : www.csepracticals.com
Owned by : CSEPracticals

Sliding Window Rules

- Now, we shall do one example which illustrates the role of send window and recv window in keeping track of bytes sent and recvd between TCP peers

- Sliding Window Rules :
  - Whenever the pure ACK is received, send window of recipient of ACK slides

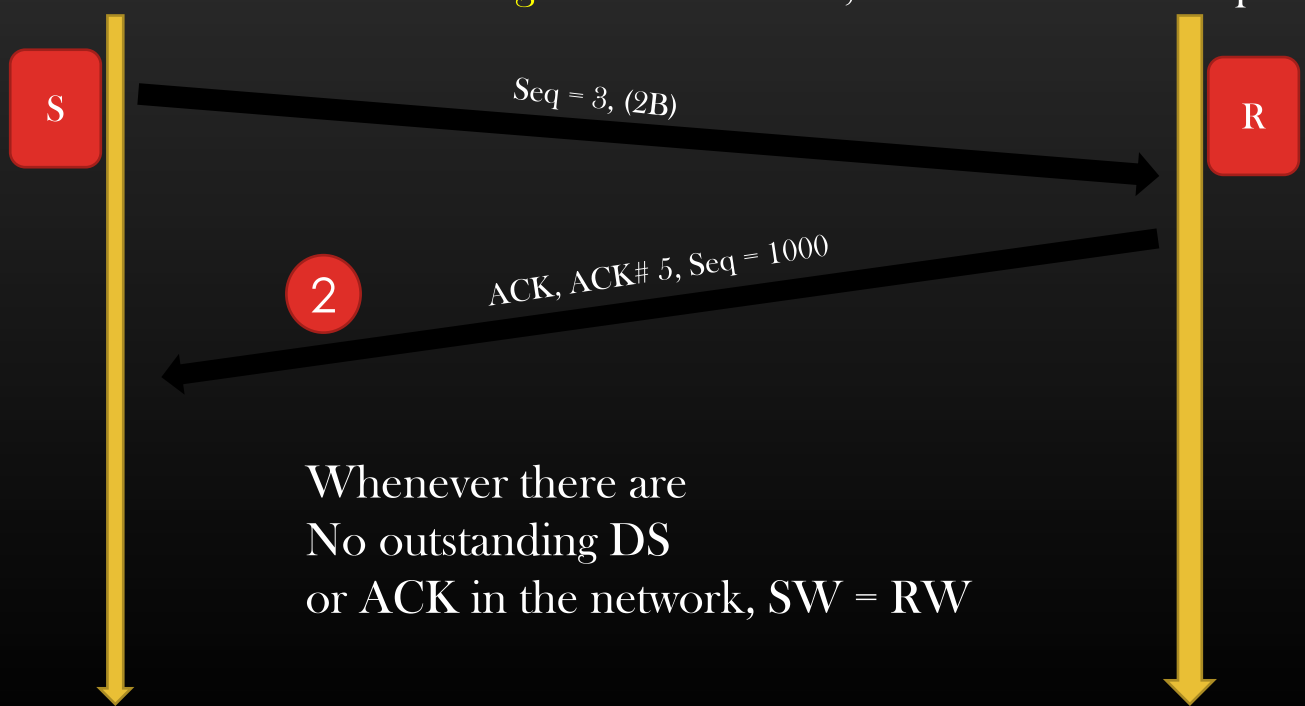


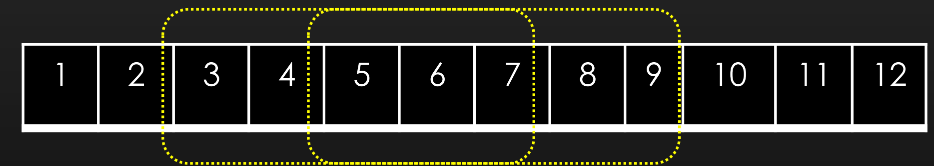
Visit : www.csepracticals.com
Owned by : CSEPracticals

Sliding Window Rules

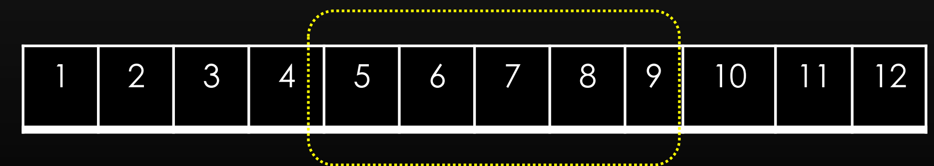
- Now, we shall do one example which illustrates the role of send window and recv window in keeping track of bytes sent and recvd between TCP peers

- Sliding Window Rules :
  - Whenever the pure ACK is received, send window of recipient of ACK slides
  - Whenever the Data segment is received, recv window of recipient of data segment slides





Recv Window of R



Send Window of S

Visit : www.csepracticals.com
Owned by : CSEPracticals

Sliding Window Rules

- Now, we shall do one example which illustrates the role of send window and recv window in keeping track of bytes sent and recvd between TCP peers

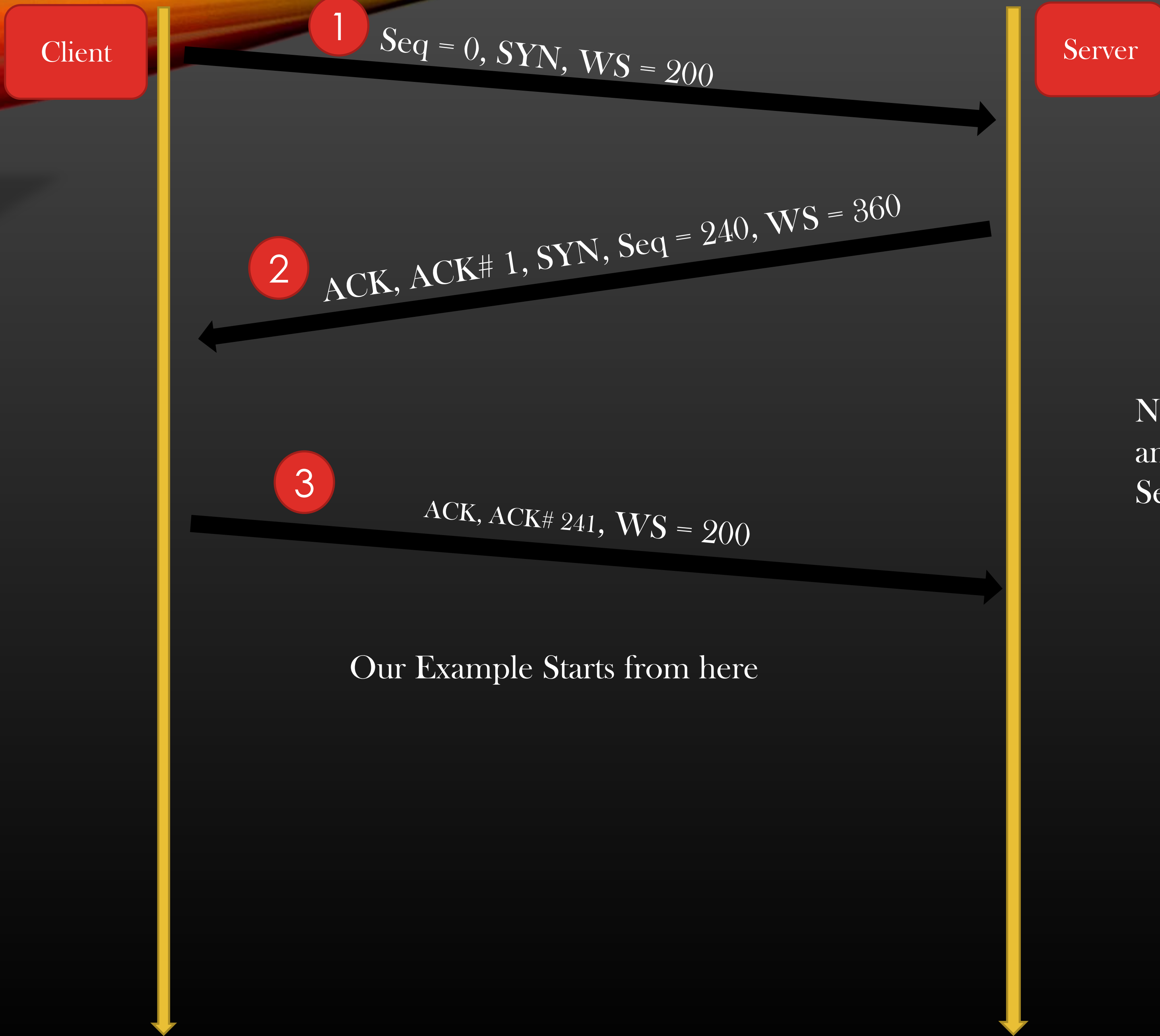
- Sliding Window Rules :
  - Whenever the pure ACK is received, send window of recipient of ACK slides
  - Whenever the Data segment is received, recv window of recipient of data segment slides
  - Whenever the Data segment combined with ACK is recvd, recv and send window of recipient slides

  - Note : Windows Slides whenever there is reception of Data Segment or ACK, TCP Sender Windows (send or reecv) do not slides when TCP *SENDS* any type of segment be it data segment or ACK or Both

- Warning : Its time for you to take a pen and notebook and practice the example in parallel with me, else you will get lost !! The example is a bit complicated, yet easy once if you get it right

Window Management Example

- Now our Client and Server has established the connection and are ready to exchange TCP data
- Problem Statement
  - Let us suppose client and server wants to carry out following data exchange
    - Client send 140B data request to Server
    - Server replies in two installments – 80B reply and 280B reply
  - We shall see how send and recv windows on either ends are adjusted/slides as the data exchange happen between client and server as per the above scheme
  - Let us Start . . .



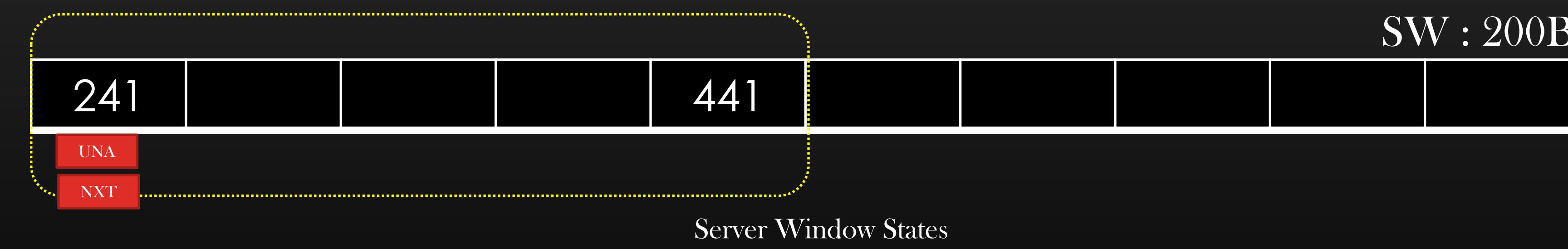
Next Slide shows the state Of Send and Recv Window Of Client and Server after successful handshake

Visit : www.csepracticals.com
Owned by : CSEPracticals

Initial Setup

Notice The correspondence between Send and Recv Windows between Client And Server !

Client Window States

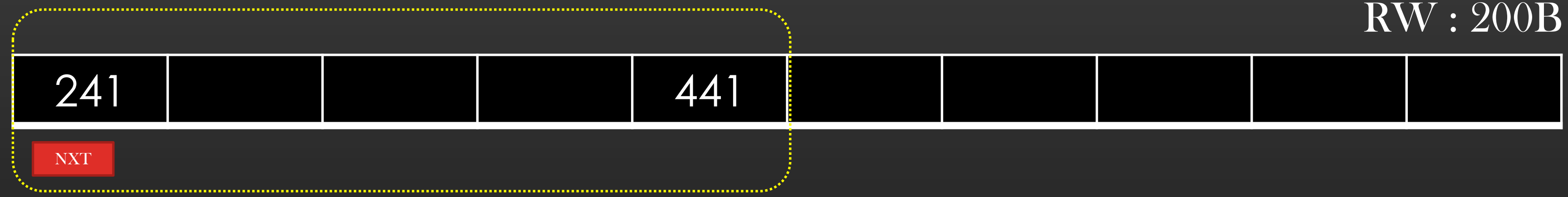


Server Window States

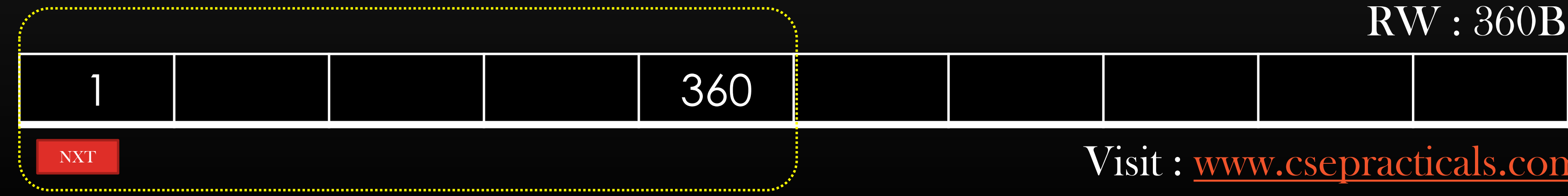
Visit : www.csepracticals.com
Owned by : CSEPracticals

Phase 1

Client Window States



Server Window States



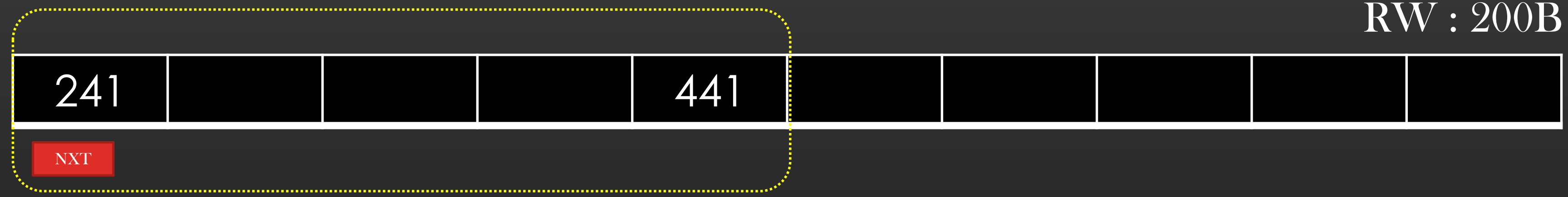
Visit : www.csepracticals.com
Owned by : CSEPracticals

1 Seq# 1, 140B, WS = 200

SW : 360B

Client Window States

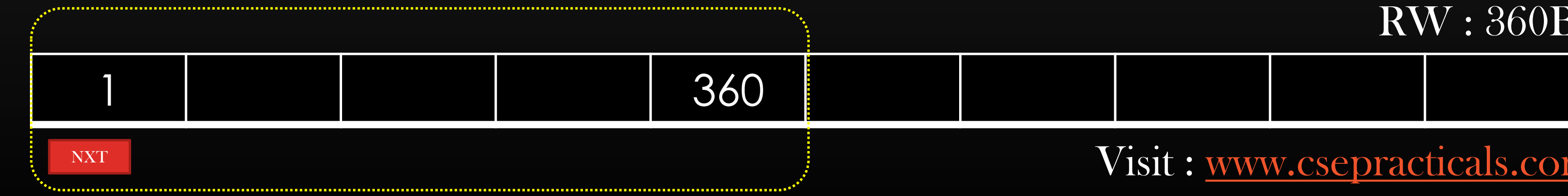
RW : 200B



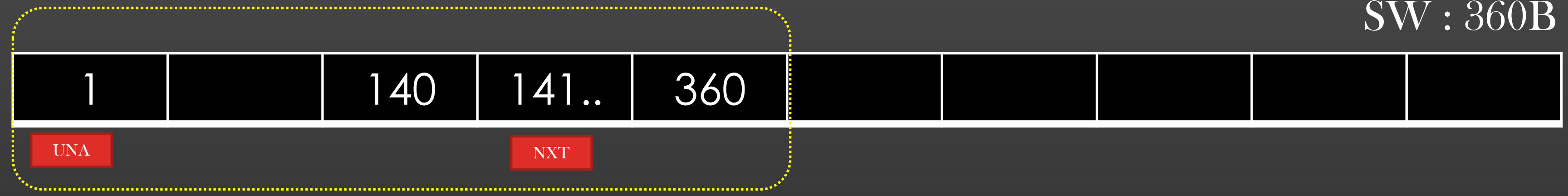
SW : 200B

Server Window States

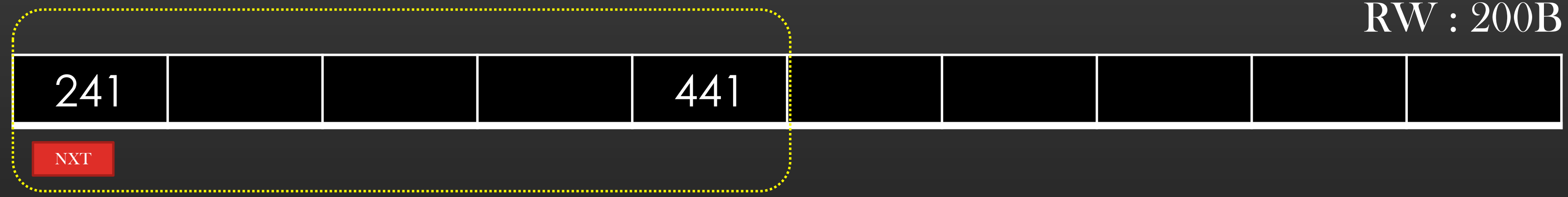
RW : 360B

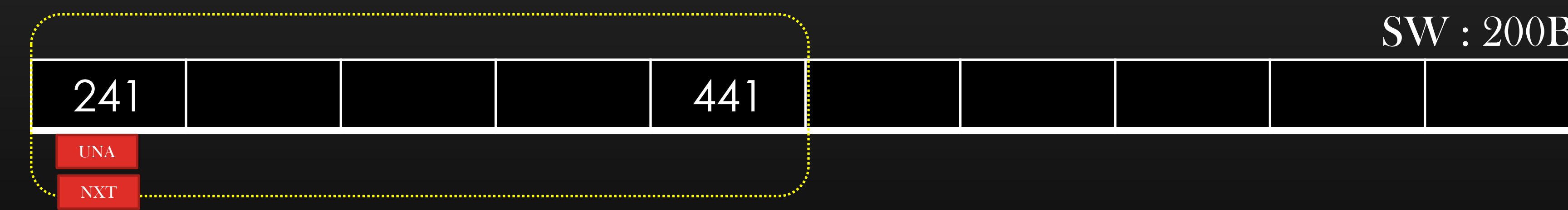


Visit : www.csepracticals.com
Owned by : CSEPracticals



Client Window States

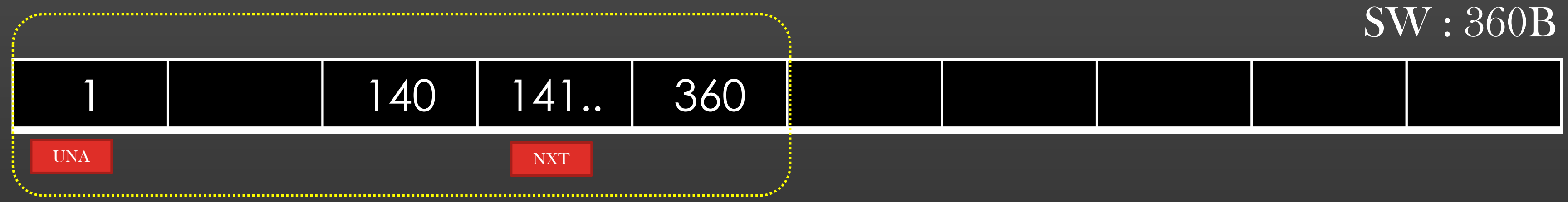




Server Window States

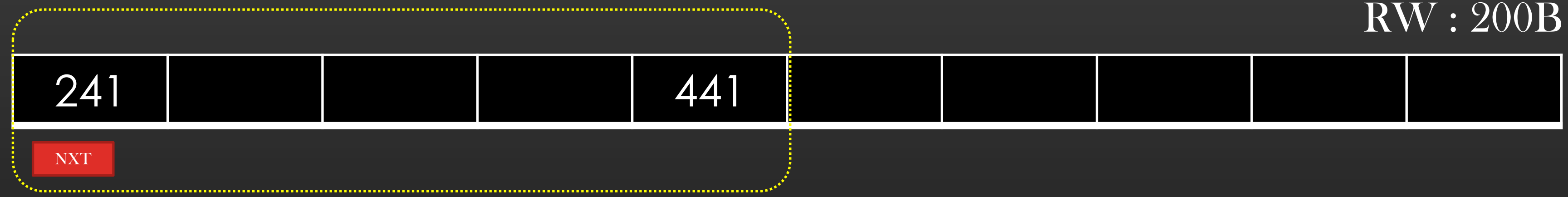
Visit : www.csepracticals.com
Owned by : CSEPracticals

SW : 360B

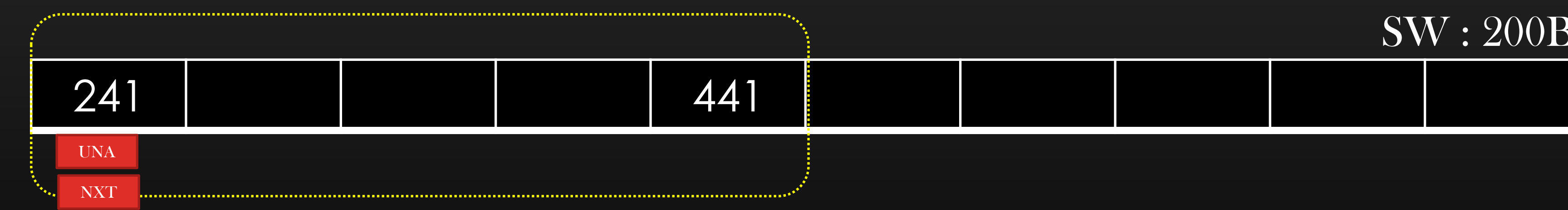


Client Window States

RW : 200B



SW : 200B



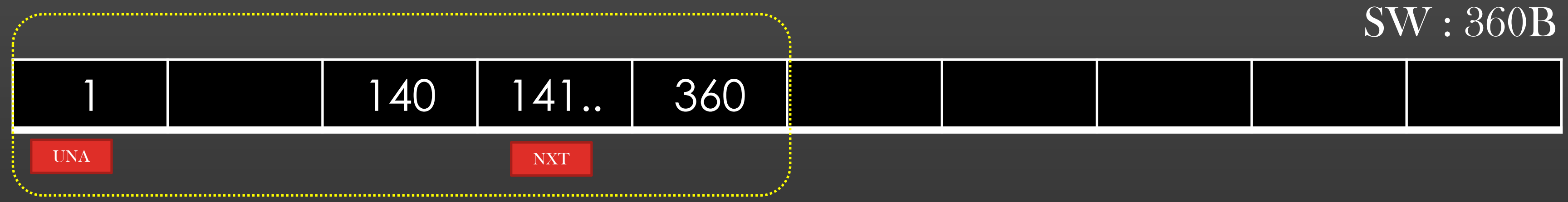
Server Window States

RW : 360B

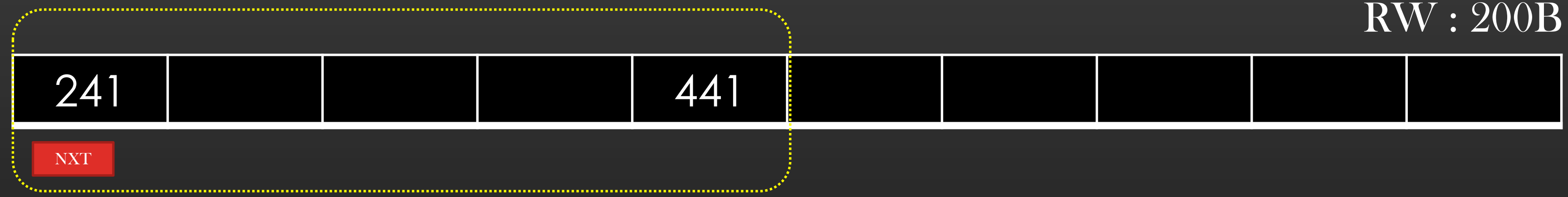
Visit : www.csepracticals.com
Owned by : CSEPracticals

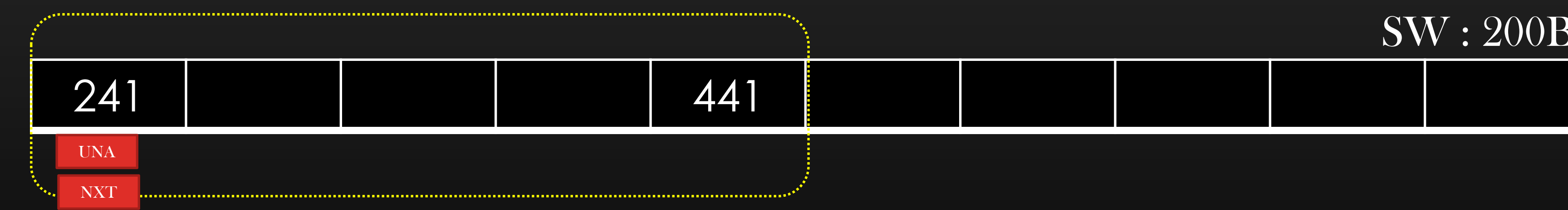
1 Seq# 1, 140B, WS = 200

2 Seq# 241, ACK# 141, 80B, WS = 360



Client Window States





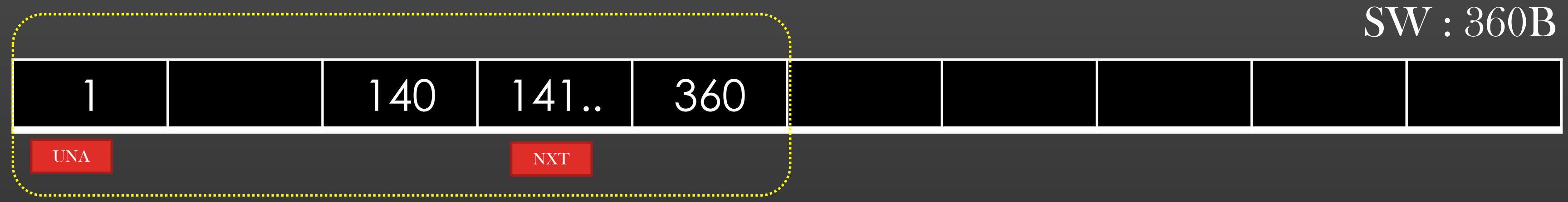
Server Window States

Visit : www.csepracticals.com
Owned by : CSEPracticals

1 Seq# 1, 140B, WS = 200

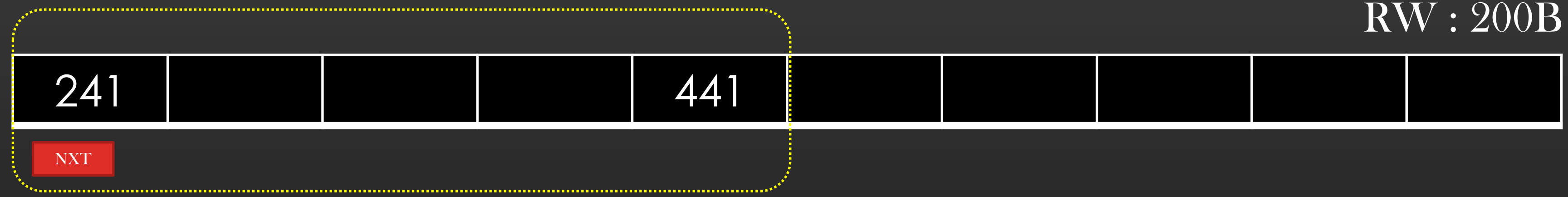
2 Seq# 241, ACK# 141, 80B, WS = 360

SW : 360B



Client Window States

RW : 200B



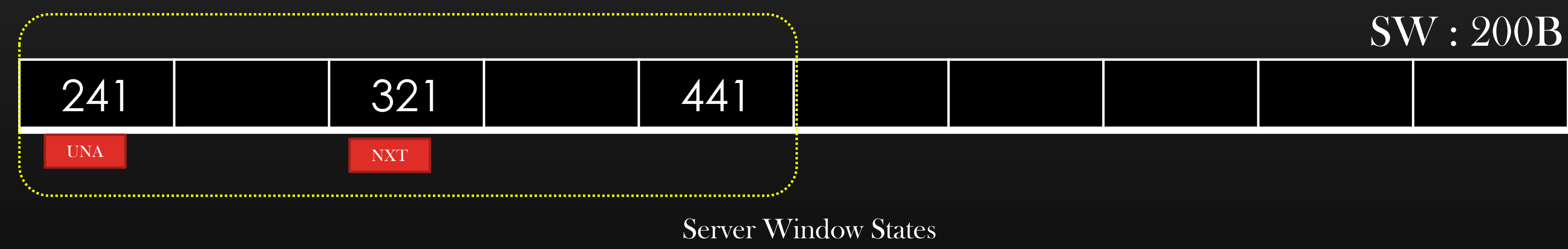
SW : 200B

Server Window States

RW : 360B

1 Seq# 1, 140B, WS = 200

2 Seq# 241, ACK# 141, 80B, WS = 360

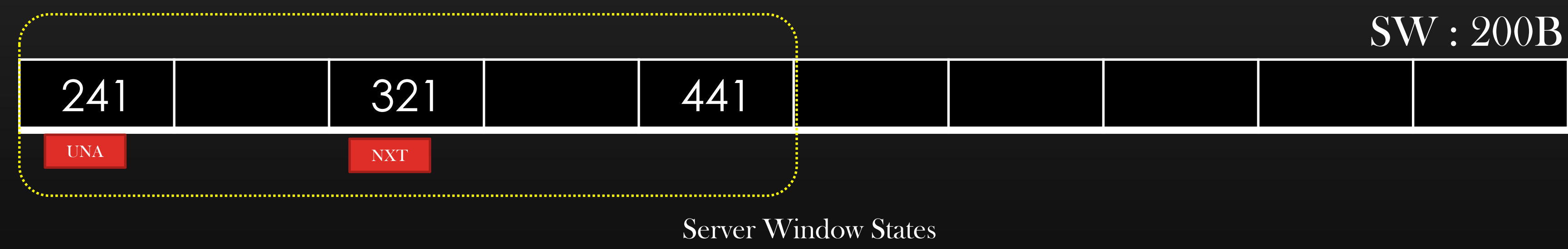


Visit : www.csepracticals.com
Owned by : CSEPracticals

1 Seq# 1, 140B, WS = 200

2 Seq# 241, ACK# 141, 80B, WS = 360

Client Window States



Server Window States

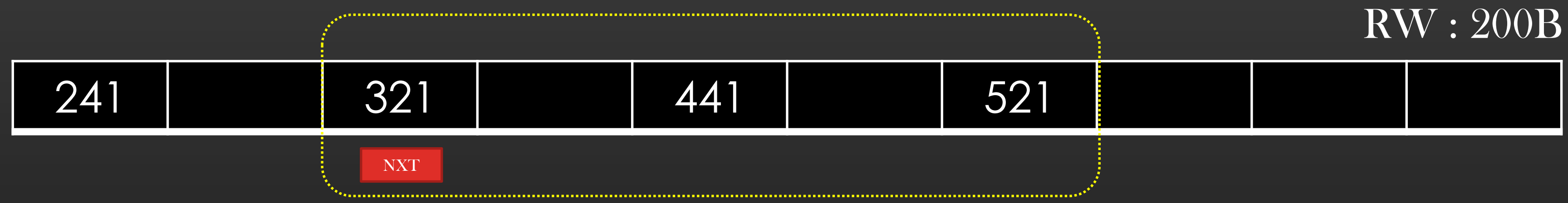
Visit : www.csepracticals.com
Owned by : CSEPracticals

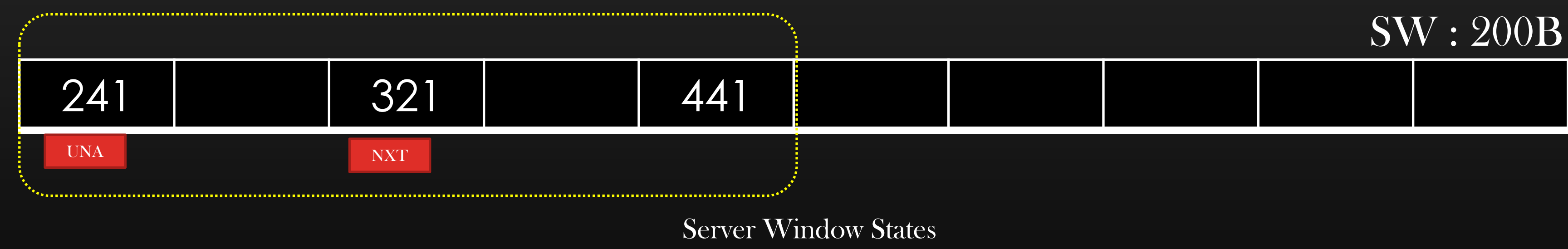
1 Seq# 1, 140B, WS = 200

2 Seq# 241, ACK# 141, 80B, WS = 360

3 Seq# 141, ACK# 321, WS = 200

Client Window States





Server Window States

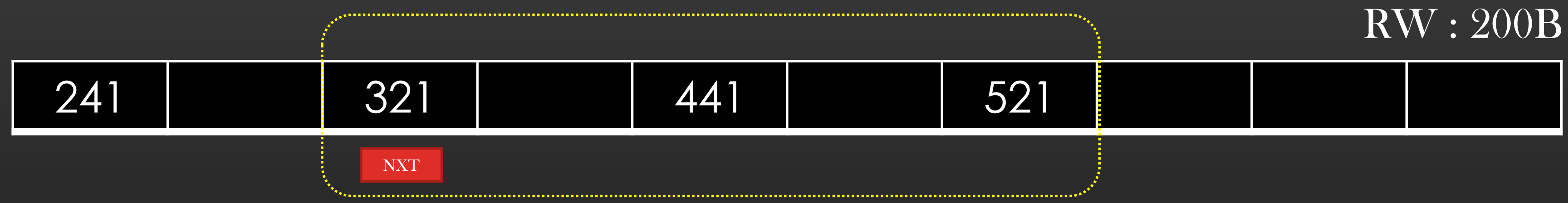
Visit : www.csepracticals.com

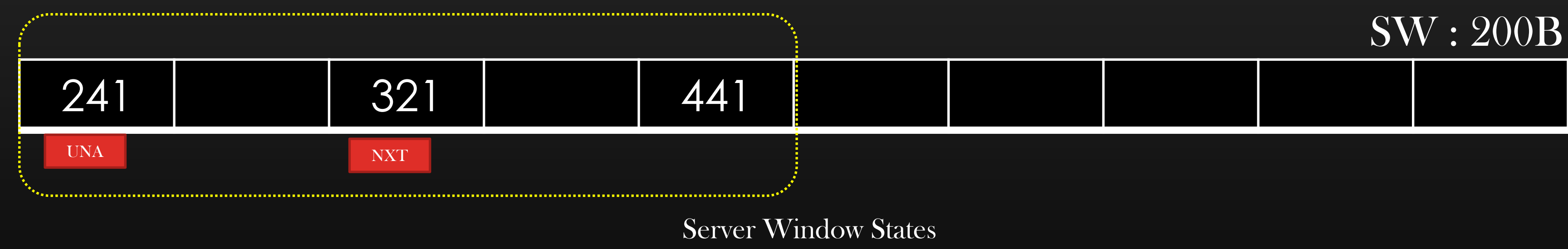
Owned by : CSEPracticals

Phase 2

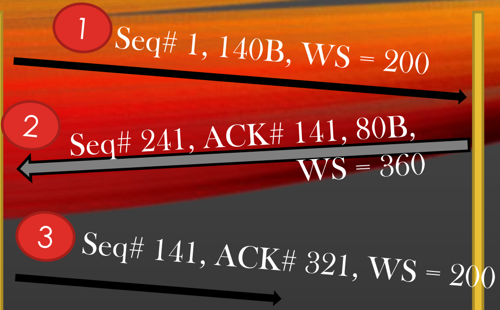
Visit : www.csepracticals.com
Owned by : CSEPracticals

Client Window States





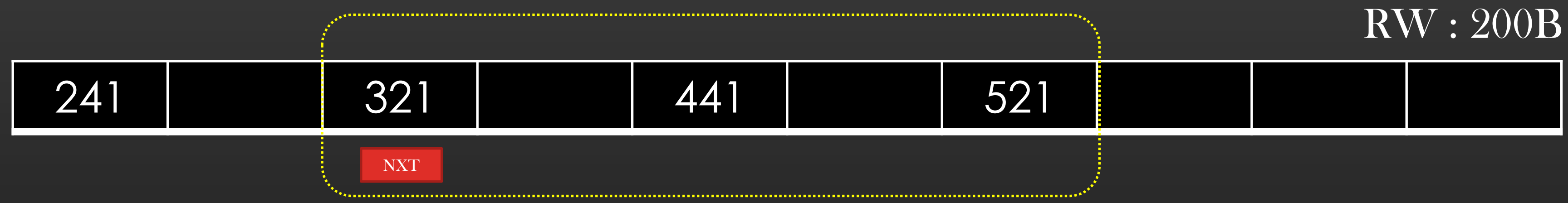
Server Window States



SW : 360B

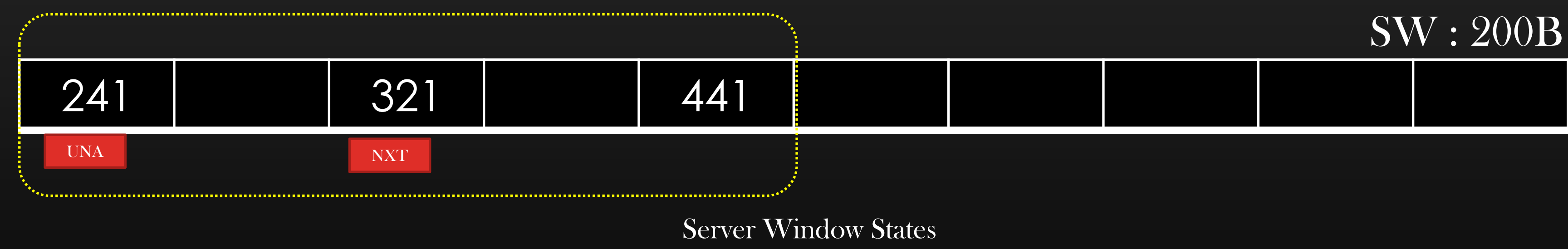
Client Window States

RW : 200B



Suppose : Appln on TCP Server generates 280 B of data to be sent to client at this moment

SW : 200B

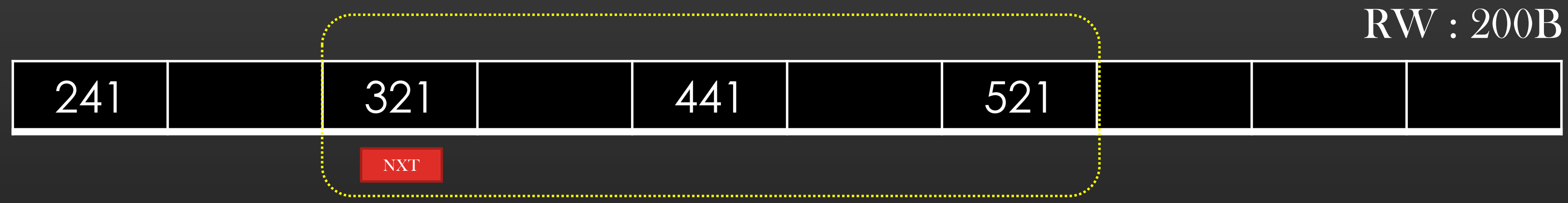


Server Window States

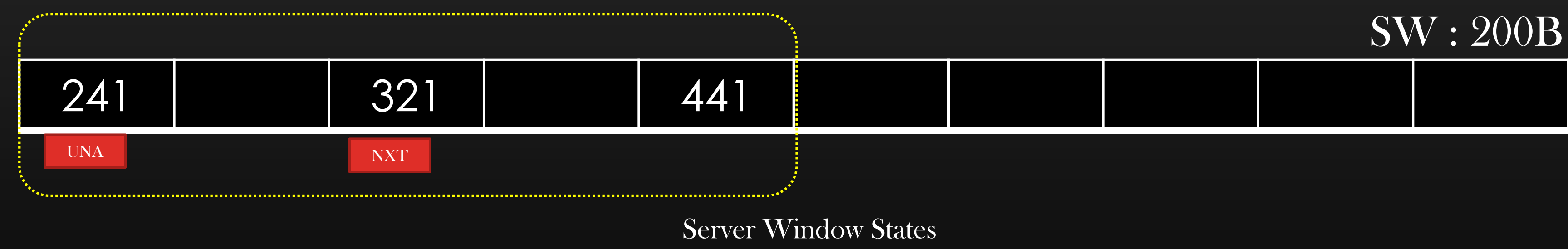
RW : 360B

Visit : www.csepracticals.com
Owned by : CSEPracticals

Client Window States



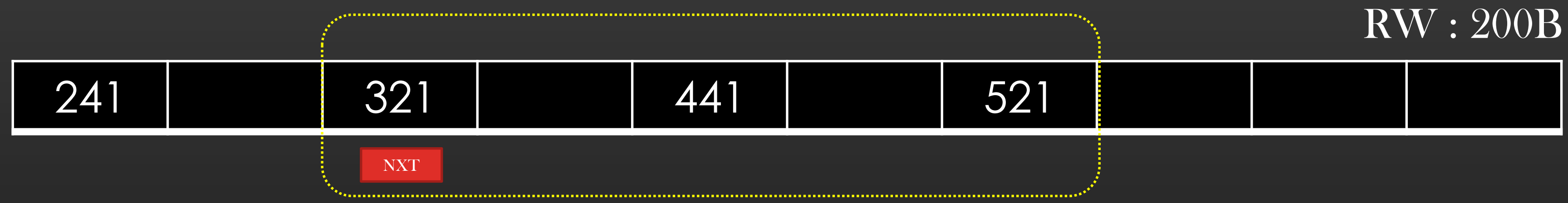
Appln on TCP Server sends only 120B of data, pending data = 280 – 120 = 160B



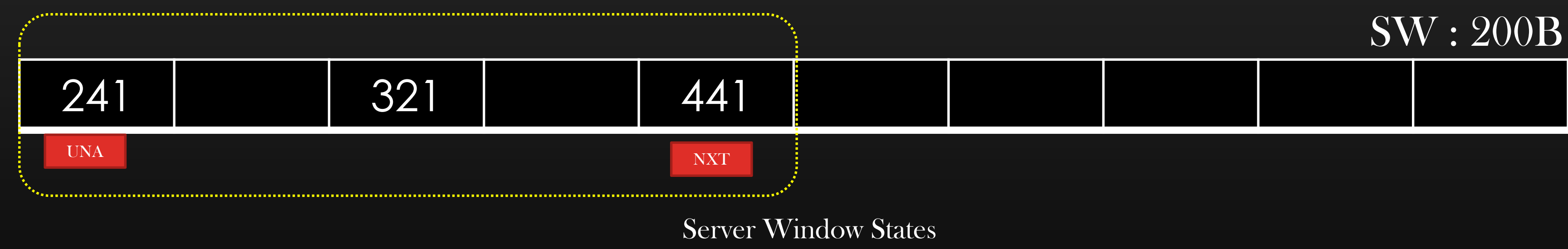
Server Window States

Visit : www.csepracticals.com
Owned by : CSEPracticals

Client Window States



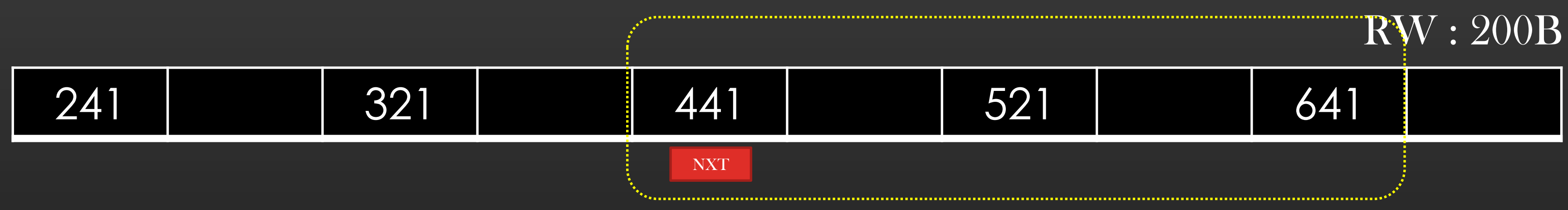
Appln on TCP Server sends only 120B of data, pending data = 280 – 120 = 160B



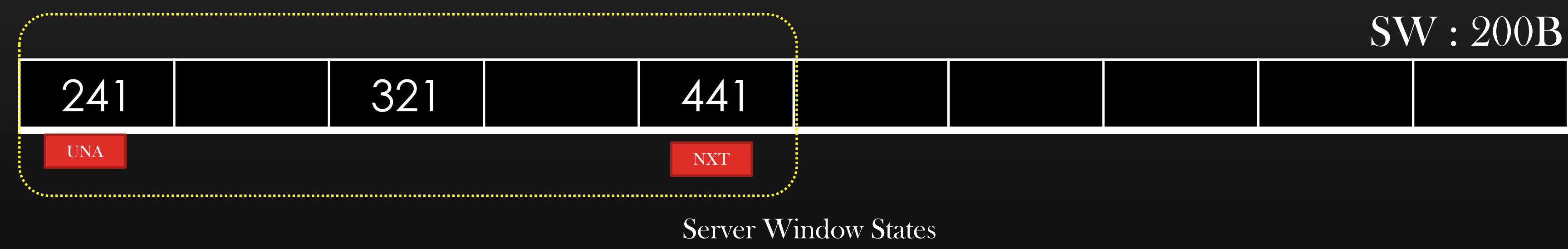
Server Window States

Visit : www.csepracticals.com
Owned by : CSEPracticals

Client Window States



Appln on TCP Server sends only 120B of data, pending data = 280 – 120 = 160B

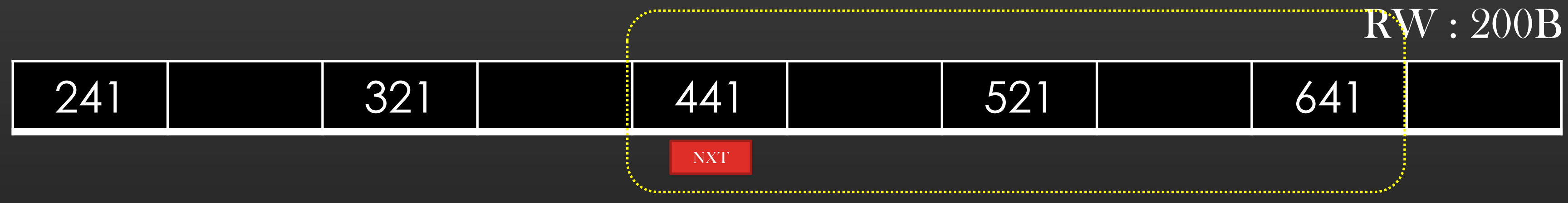


Server Window States

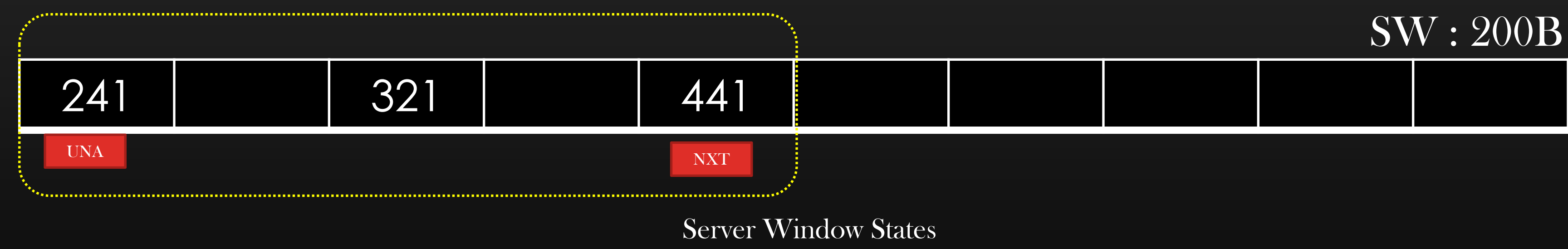
Visit : www.csepracticals.com

Owned by : CSEPracticals

Client Window States



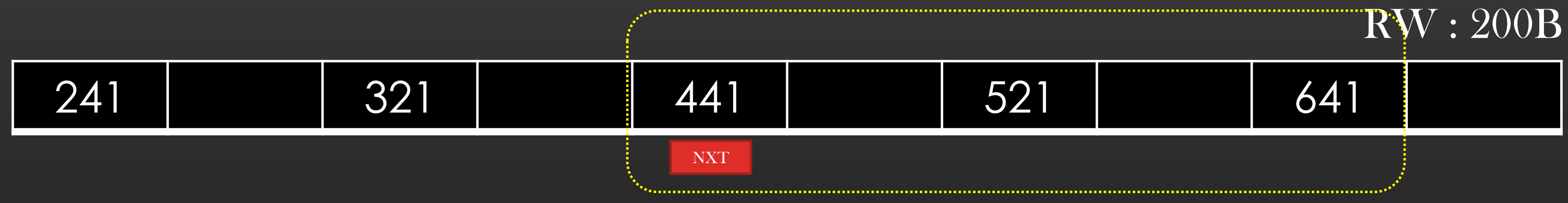
Appln on TCP Server sends only 120B of data, pending data = 280 – 120 = 160B



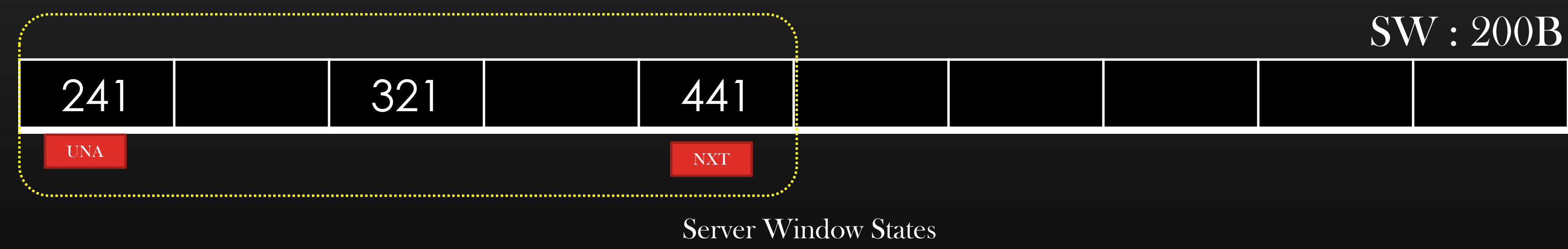
Server Window States

Visit : www.csepracticals.com
Owned by : CSEPracticals

Client Window States



Note : At this point the TCP Server's send Window is completely exhausted. TCP Server Cannot sent any data to client unless its send window make some room !



Server Window States

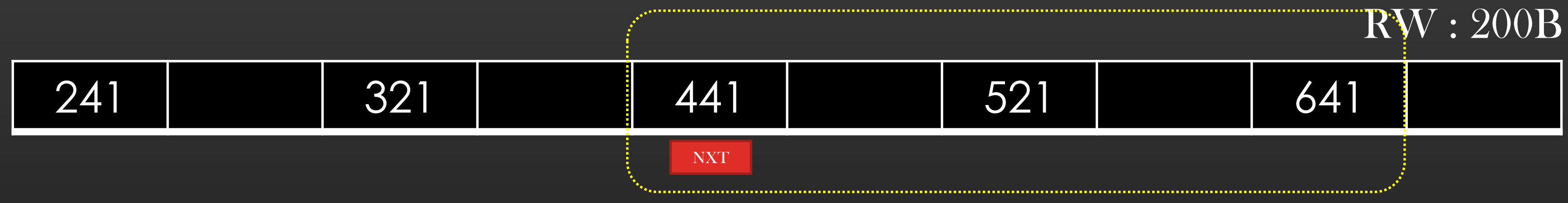
Visit : www.csepracticals.com
Owned by : CSEPracticals

Phase 3

SW : 360B

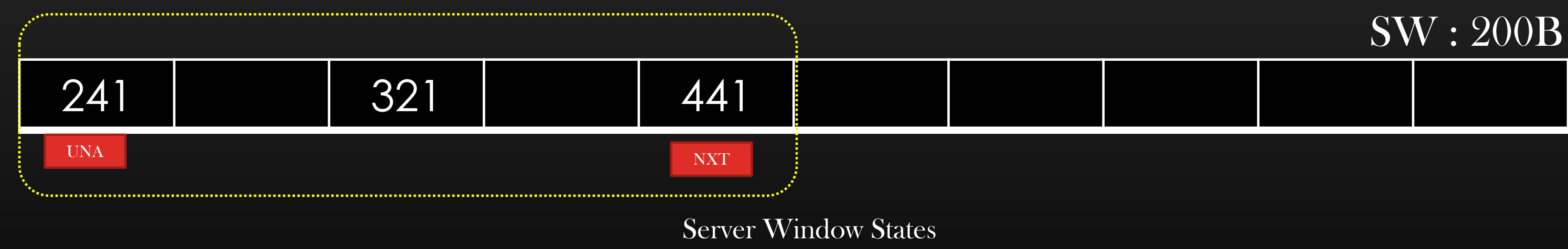
Client Window States

RW : 200B



Reminder : Appln on TCP Server sends only 120B of data, pending data = 280 – 120 = 160B

SW : 200B

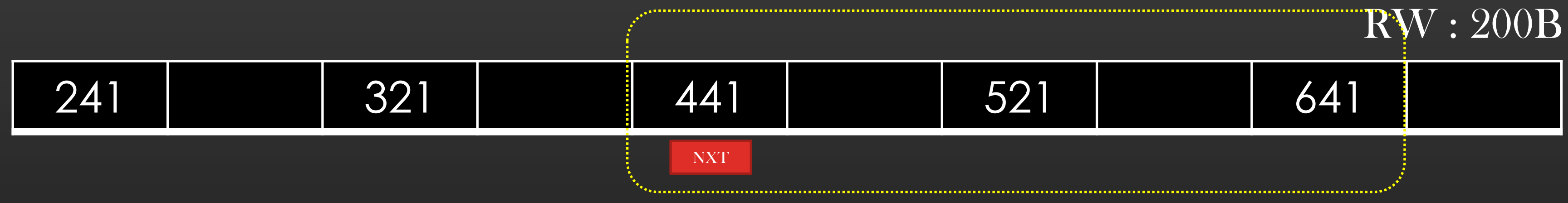


Server Window States

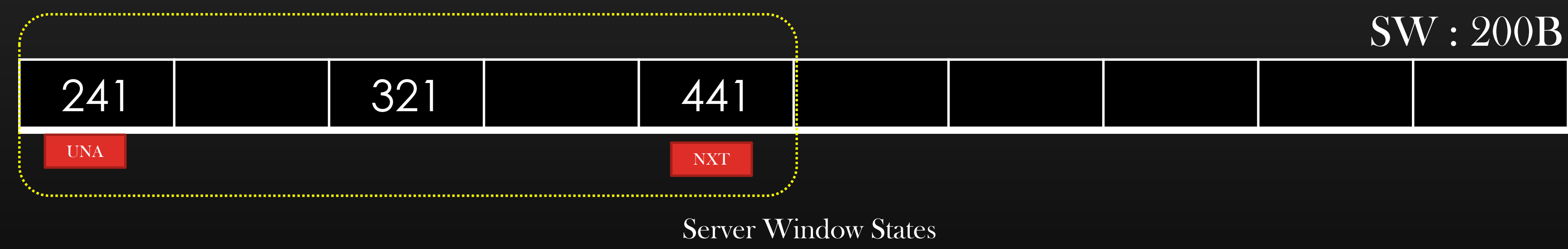
RW : 360B

Visit : www.csepracticals.com
Owned by : CSEPracticals

Client Window States

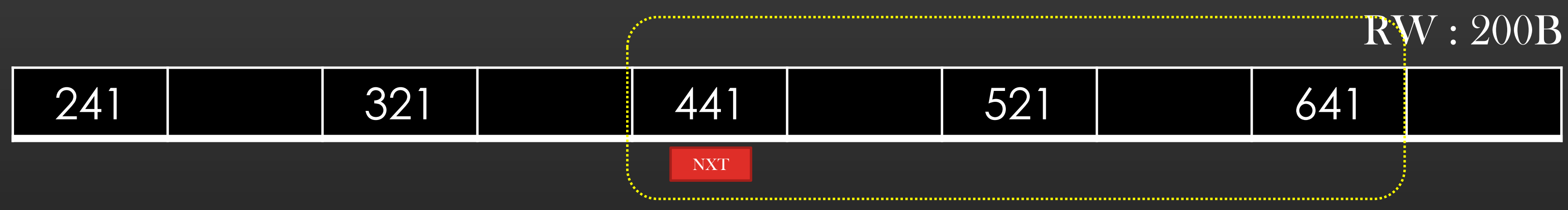


Reminder : Appln on TCP Server sends only 120B of data, pending data = 280 – 120 = 160B

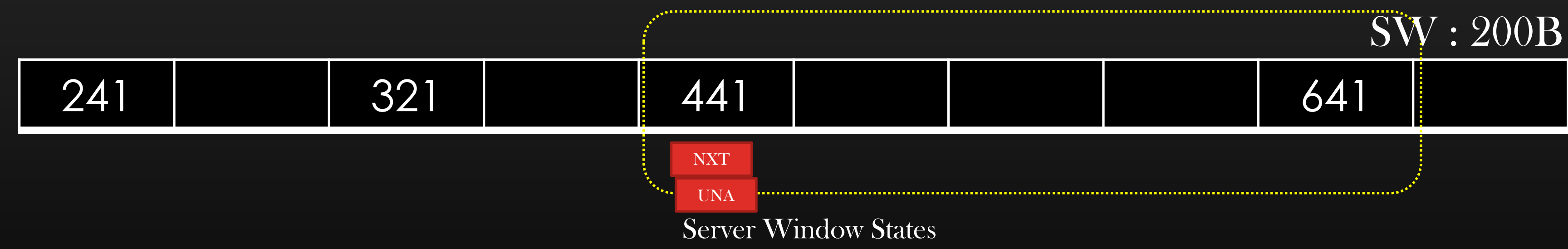


Server Window States

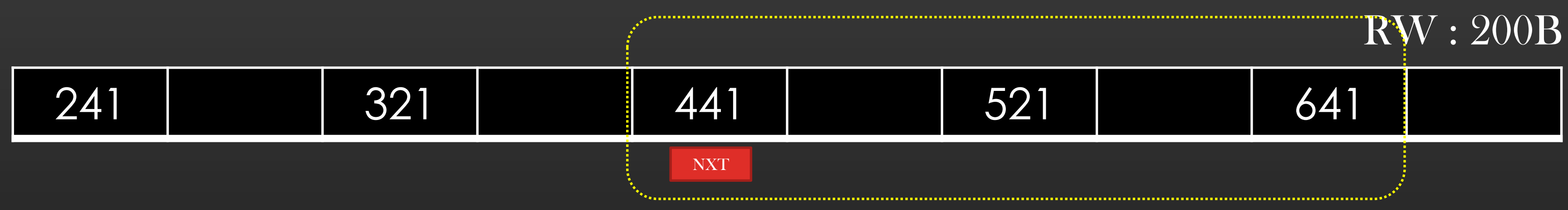
Visit : www.csepracticals.com
Owned by : CSEPracticals



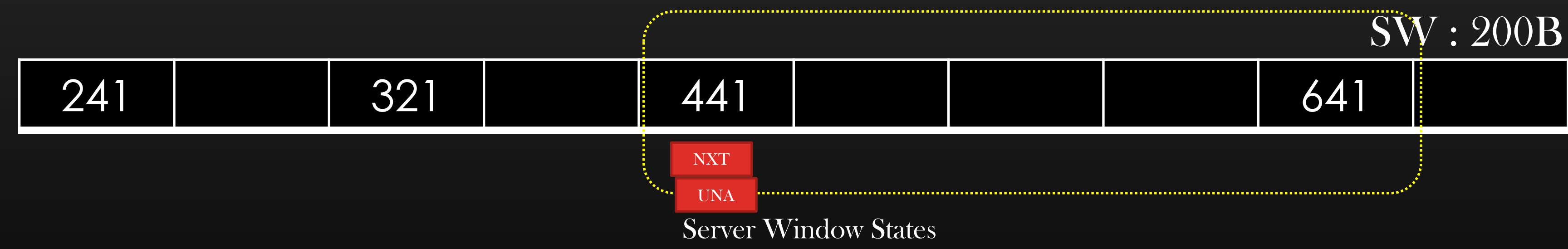
Reminder : Appln on TCP Server sends only 120B of data, pending data = 280 – 120 = 160B



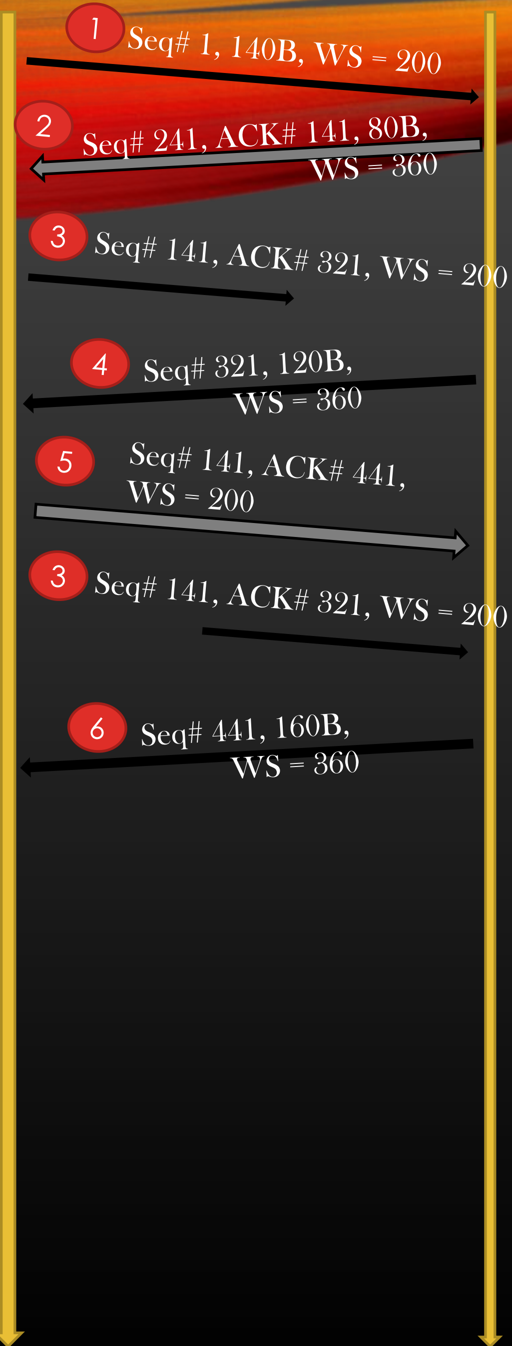
Visit : www.csepracticals.com
Owned by : CSEPracticals



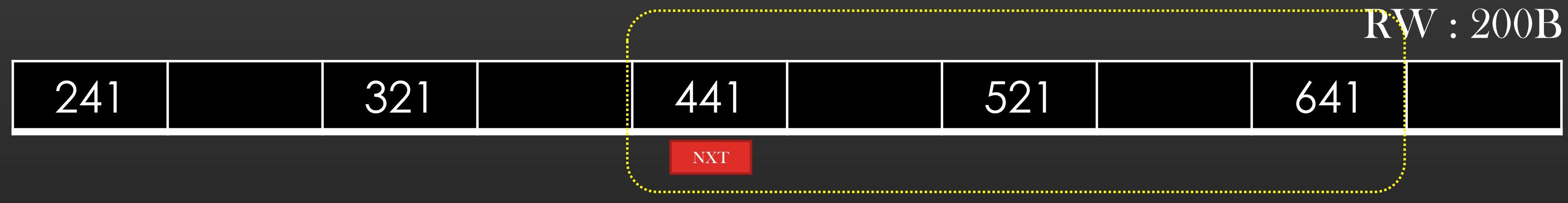
Reminder : Appln on TCP Server sends only 120B of data, pending data = 280 – 120 = 160B



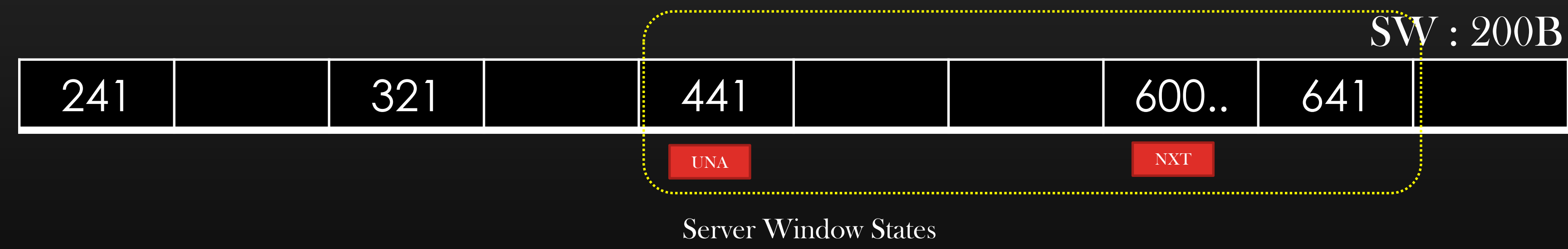
Visit : www.csepracticals.com
Owned by : CSEPracticals



Client Window States



Reminder : Appln on TCP Server sends only 120B of data, pending data = 280 – 120 = 160B

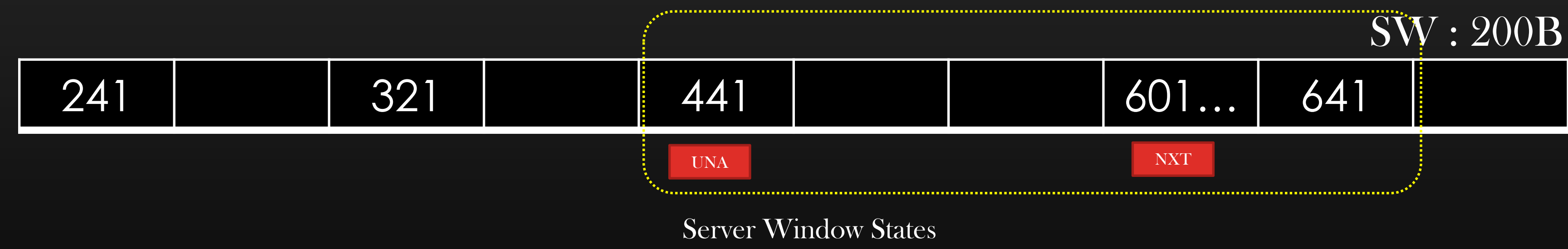


Server Window States

Visit : www.csepracticals.com
Owned by : CSEPracticals

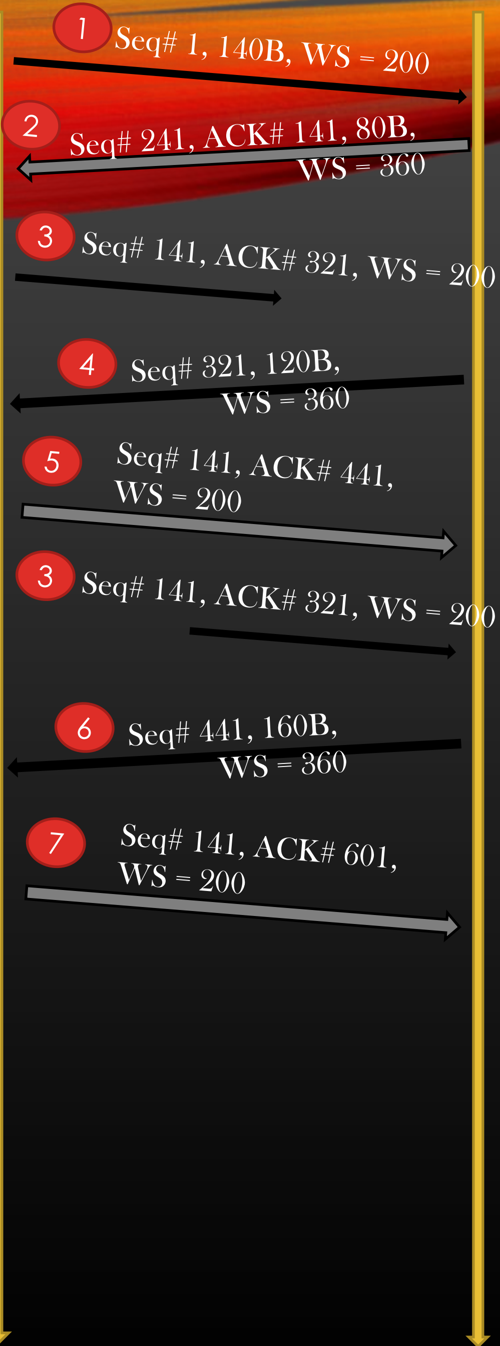
Client Window States

Reminder : Appln on TCP Server sends only 120B of data, pending data = 280 – 120 = 160B

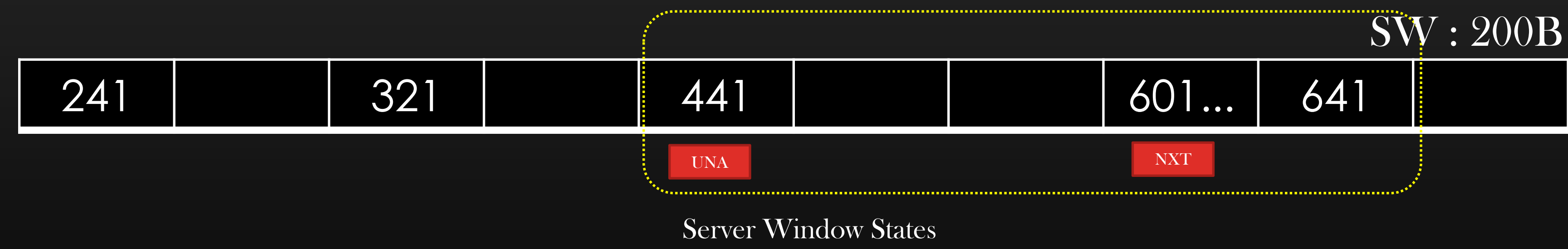


Server Window States

Visit : www.csepracticals.com
Owned by : CSEPracticals

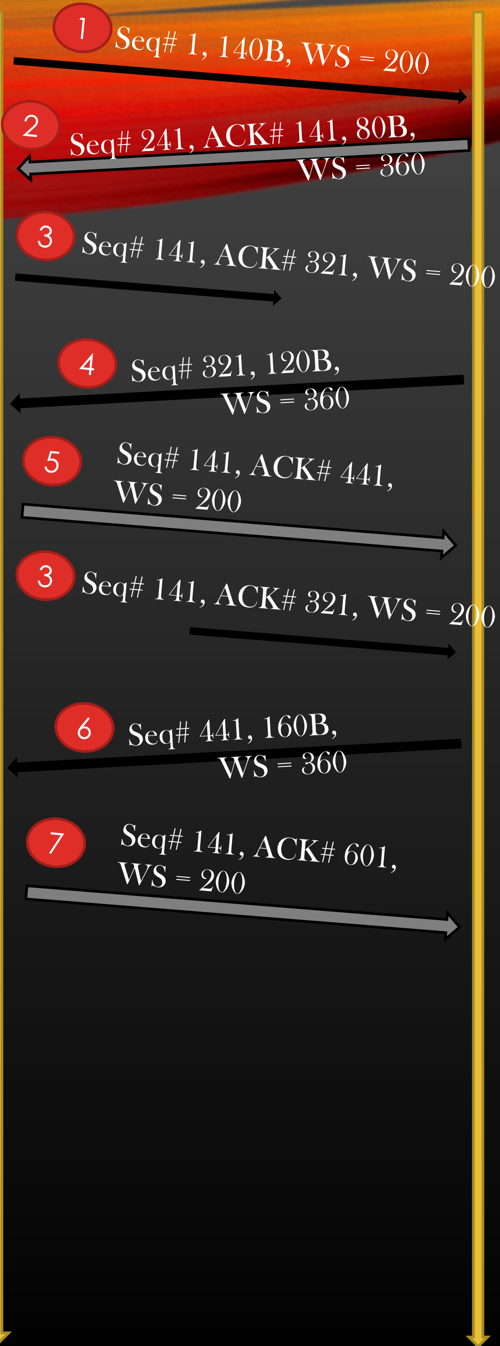


Client Window States



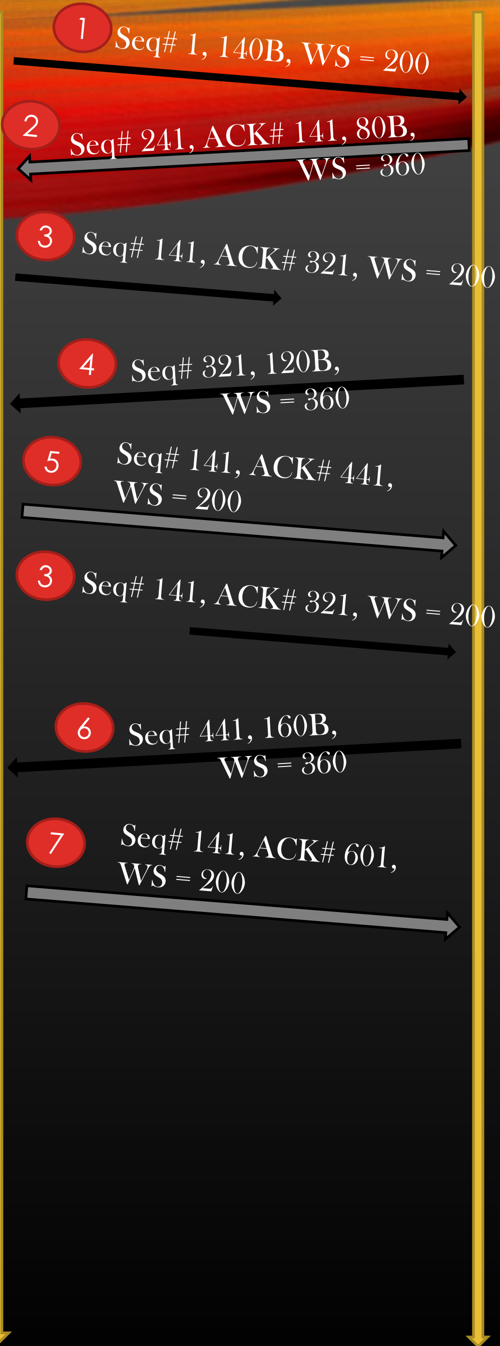
Server Window States

Visit : www.csepracticals.com
Owned by : CSEPracticals



Client Window States

Server Window States



Client Window States

Note : No Segments are in transit, no pending data or pending ACK, All windows are synchronized

Server Window States

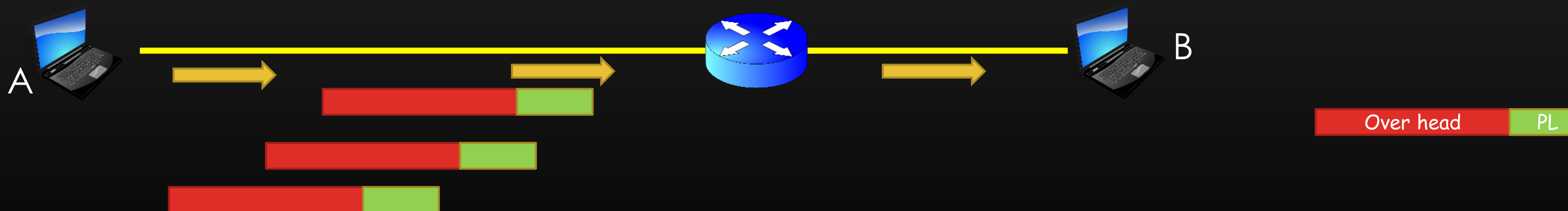
Visit : www.csepracticals.com
Owned by : CSEPracticals

Observations

1. Reception of Data Segment Causes Recv Window to Slide
2. Reception of ACK causes Send window to slide
3. Reception of Data Segment + ACK causes Recv and Send Window to Slide
4. Sending of Data Segment updates Next pointer of send window
5. Sending of ACK updates nothing on Sender's Send Or Recv Window
6. When there is no data segment or ACK in transit, no pending Data segment Or ACK, Send and Recv windows are clones on two sides
   - Send Window Of Sender = Recv Window Of receiver
   - Recv Window Of Sender = Send Window of receiver

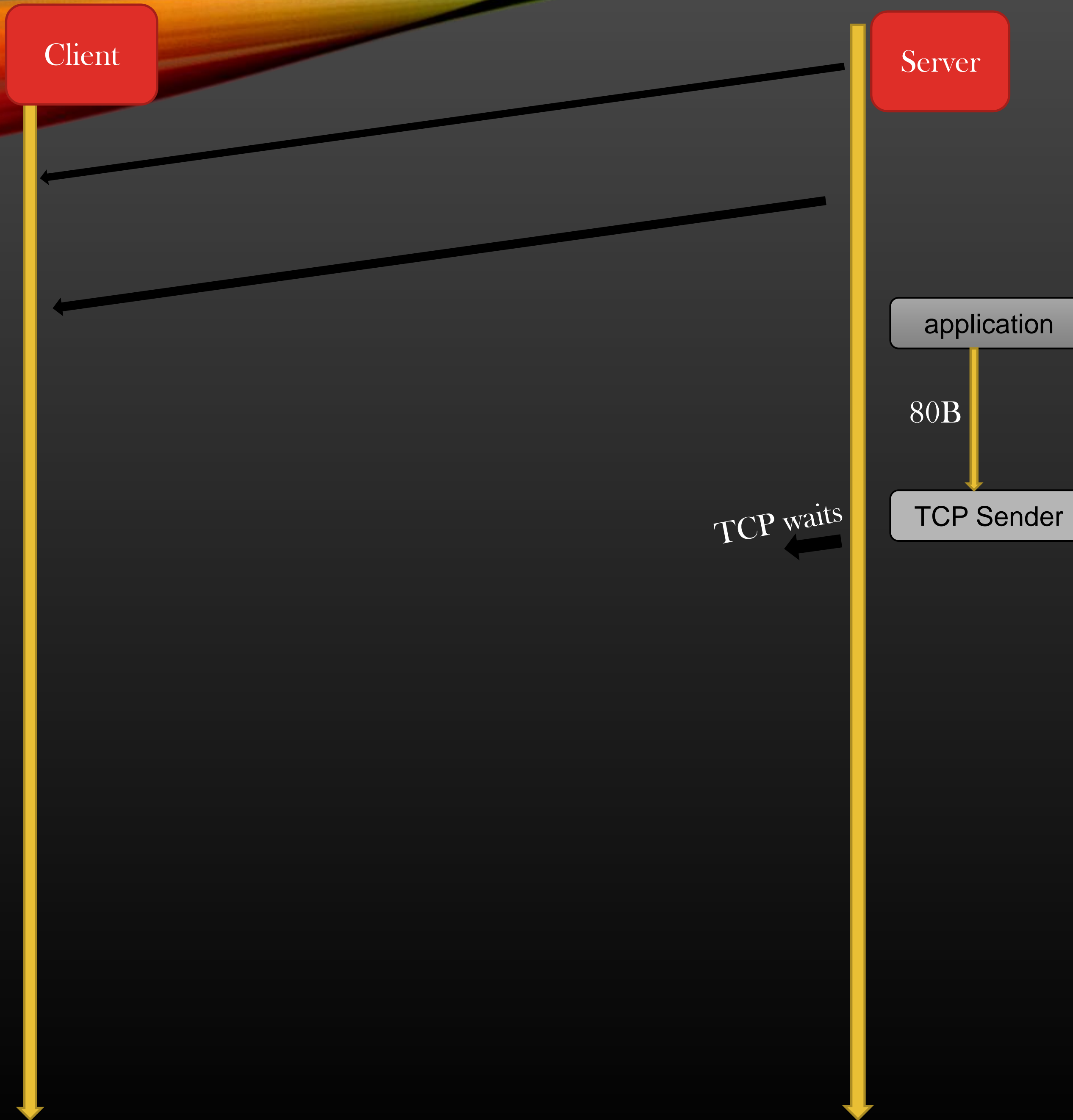
TCP Tinygrams

- TCP Tinygrams are TCP data segments carrying application payload of considerable small sizes as compared to the TCP overhead (TCP hdr size)
- TCP Default header size is 20B (without option field)
- If TCP payload is mere 2-5 bytes being carried by TCP packets, then such packets are terms as TCP tinygrams
- If TCP pushes too many tinygrams into the network, then much of the network bandwidth and recourses are wasted by useless TCP overhead data rather than by TCP useful application data (payload)
- Application on TCP sender may generate very small chunks of application data in quick succession, forcing underlying TCP to send too many tinygrams into the network



Soln : Nagle Algorithm : Avoid TCP Sender to send tinygrams into the network, unless there is no choice !!

Nagle's Algorithm

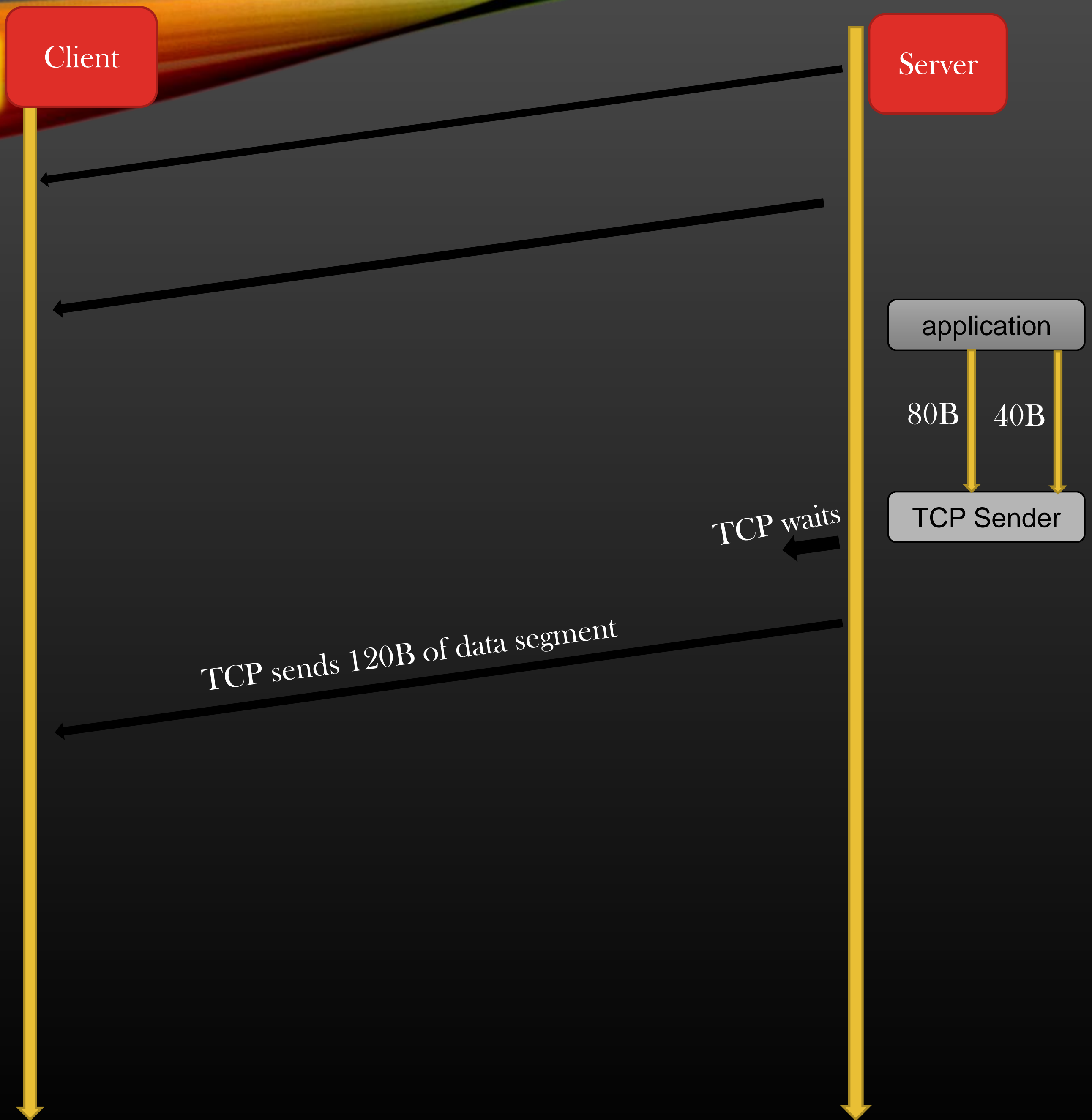


TCP has been configured to avoid send data Segments of size less than t = 100B

TCP do not send 80B immediately, and wait for :

1. Either application sends more data
2. Or all outstanding data segments have been Acknowledged (Nagle Algorithm)

Visit : www.csepracticals.com
Owned by : CSEPracticals



application

80B 40B

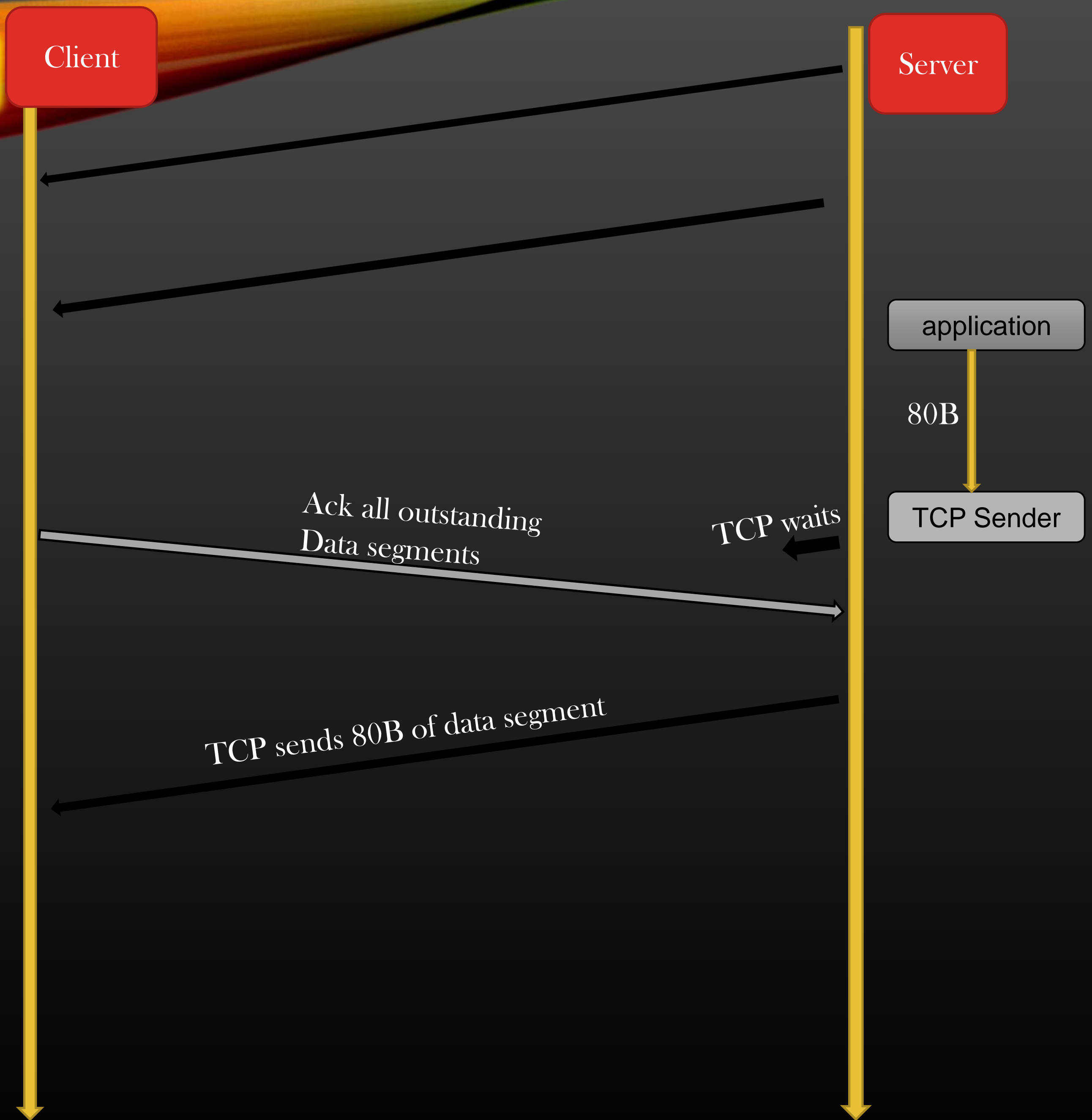
TCP Sender

TCP has been configured to not to send data Segments of size less than t = 100B
TCP do not send 80B immediately, and wait for :

1. Either application sends more data
2. Or all outstanding data segments have been Acknowledged (Nagle Algorithm)

Visit : www.csepracticals.com
Owned by : CSEPracticals

Nagle's Algorithm



TCP has been configured to not to send data Segments of size less than t = 100B

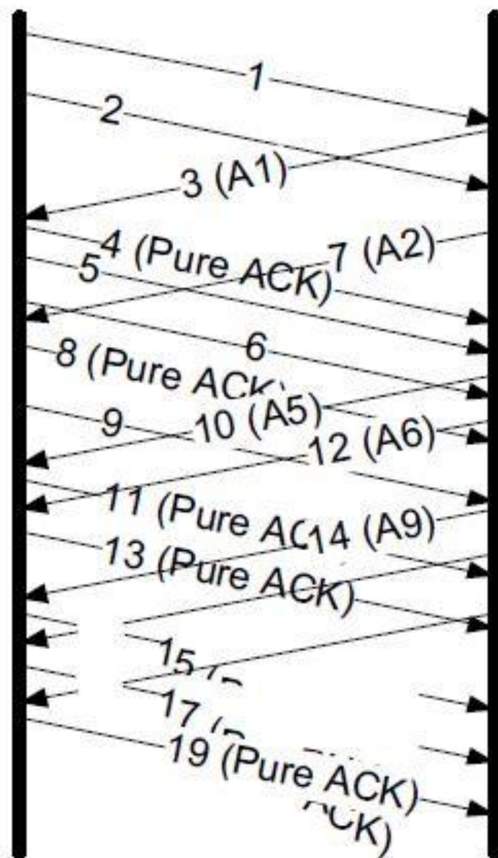
TCP do not send 80B immediately, and wait for :

1. Either application sends more data
2. Or all outstanding data segments have been Acknowledged (Nagle Algorithm)

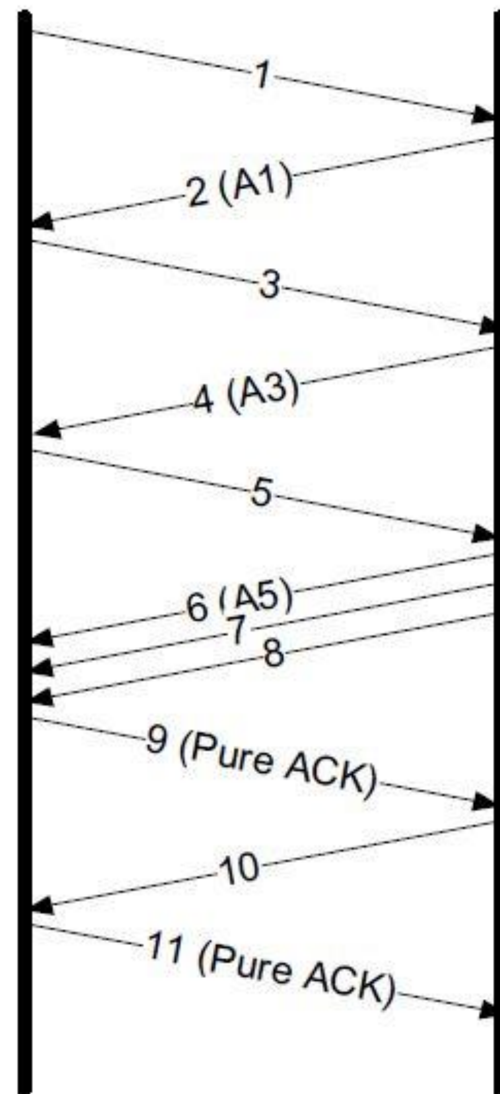
Nagle's Algorithm

- Benefits :
  - Avoid injecting too many tinygrams in the network for efficiency
  - Tinygrams are sent only when all outstanding segments have been removed from network ensuring tinygrams to not contribute to Network congestion/under-utilization
  - The beauty of the algorithm is self-clocking : the faster the ACK comes back, the faster the data is sent
  - This is the trade-off the Nagle algorithm makes: fewer and larger packets are used, but the delay is higher
  - For a given amount of bidirectional data exchange between client and Server :
    - With Nagle Algorithm : No of segments (data + ACK) exchanged = n
    - Without Nagle Algorithm : No of segments (data + ACK) exchanged = m
    - $n < m$

Visit : www.csepracticals.com
Owned by : CSEPracticals



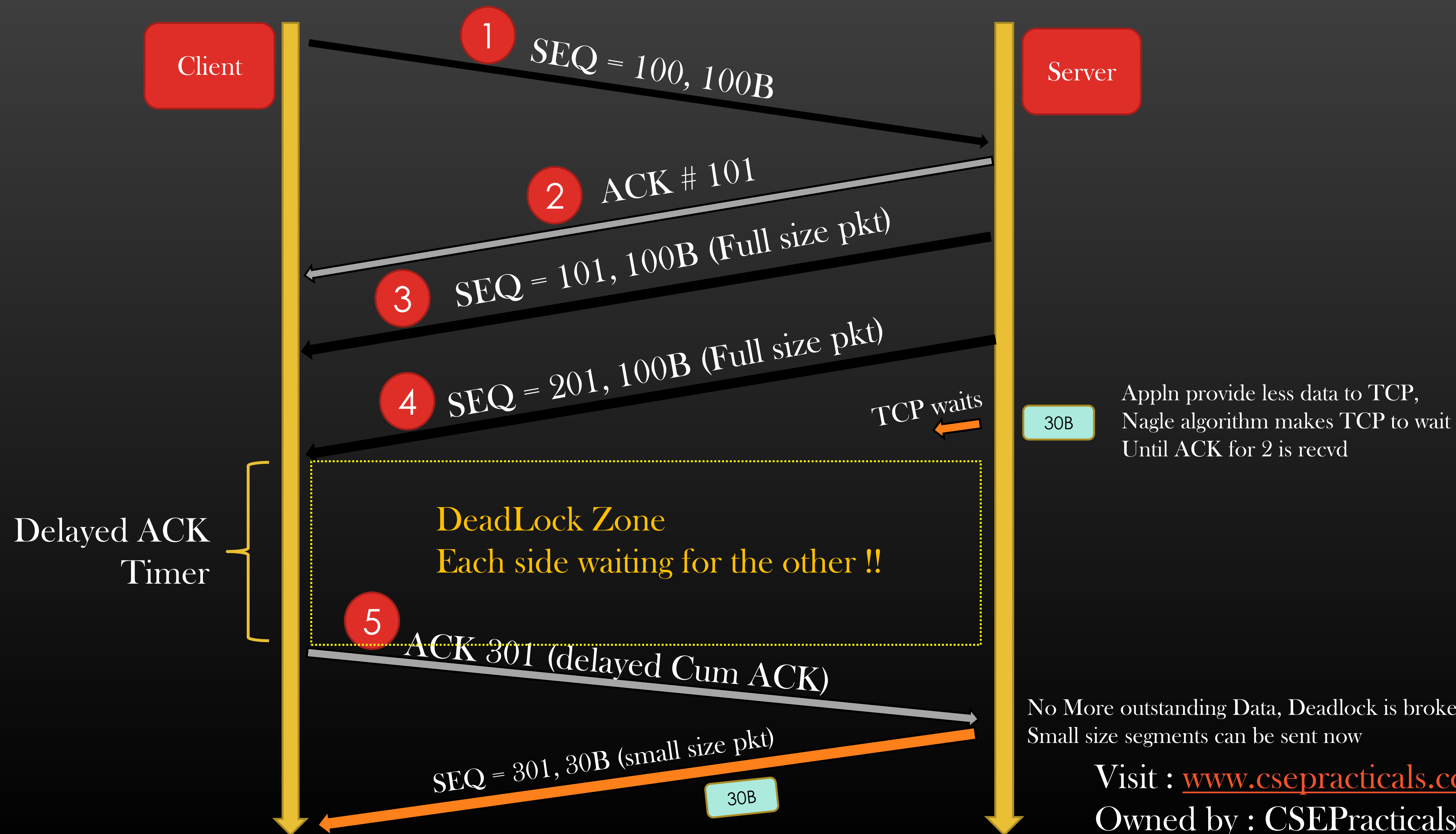
Nagle's Algorithm Disabled



Nagle's Algorithm Enabled

Mastering TCP -> Nagle's Algorithm ->Transient Deadlock

- A transient Deadlock is formed when we combine the Delayed ACK and Nagle's Algorithm together
- A transient Deadlock – Each side waiting for the other



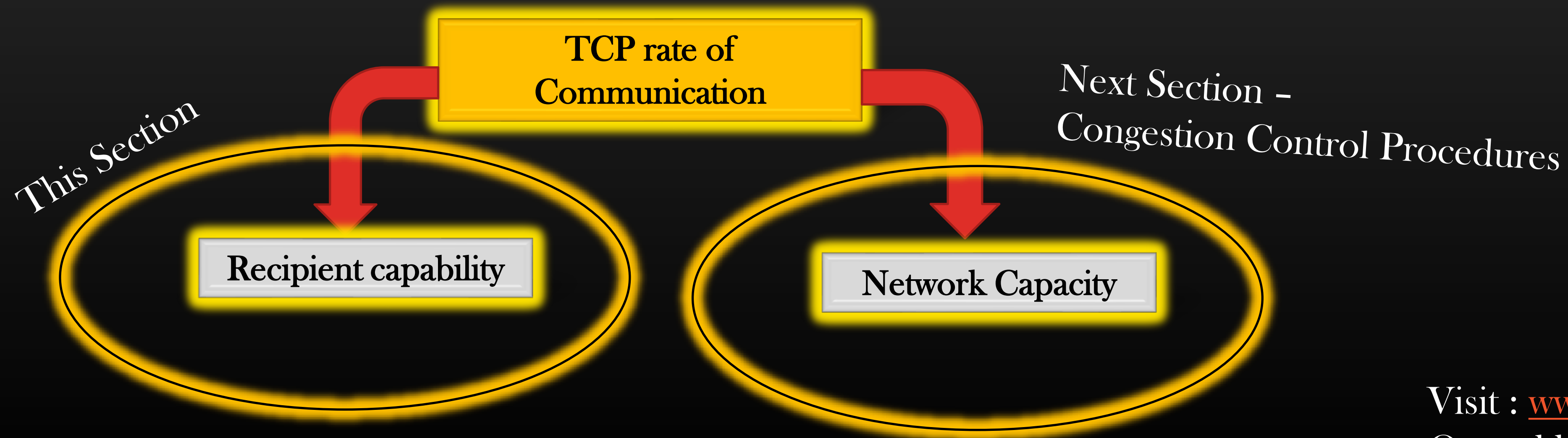
Visit : www.csepracticals.com
Owned by : CSEPracticals

TCP Window Resizing

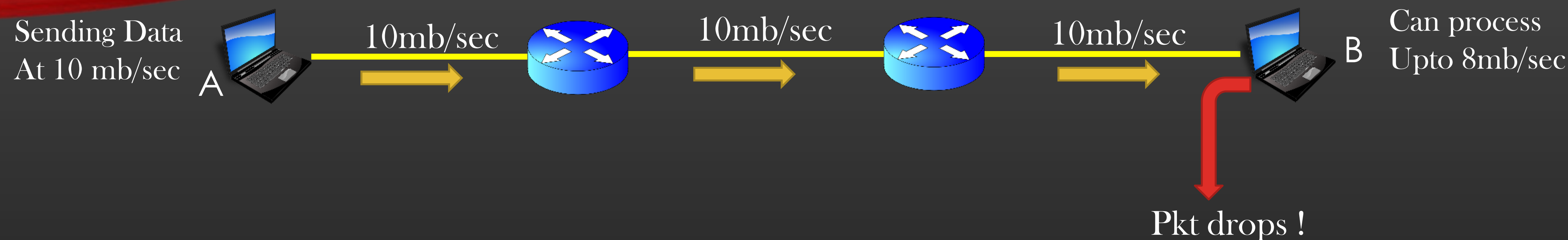
- TCP is adoptive protocol, meaning it responds to network or recipient state dynamically
  - TCP peers slow down the rate of data exchange if Network is busy or peers are slow consumers

- TCP controls the rate of data exchange byb resizing the send and recv windows
  - Smaller size send and recv windows – Lesser the rate of data exchange
  - Bigger size send and recv windows – Faster the rate of data exchange

- As stated earlier, the two entities which influences the rate of communication between TCP peers are :

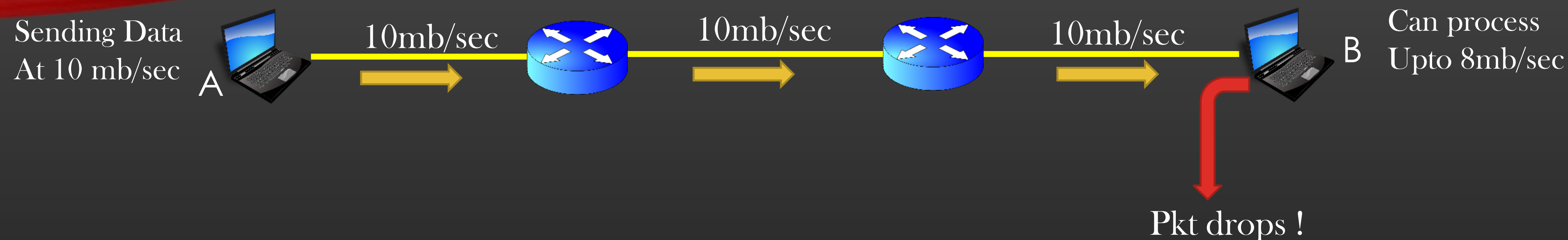


Problem Statement – Slow Receivers



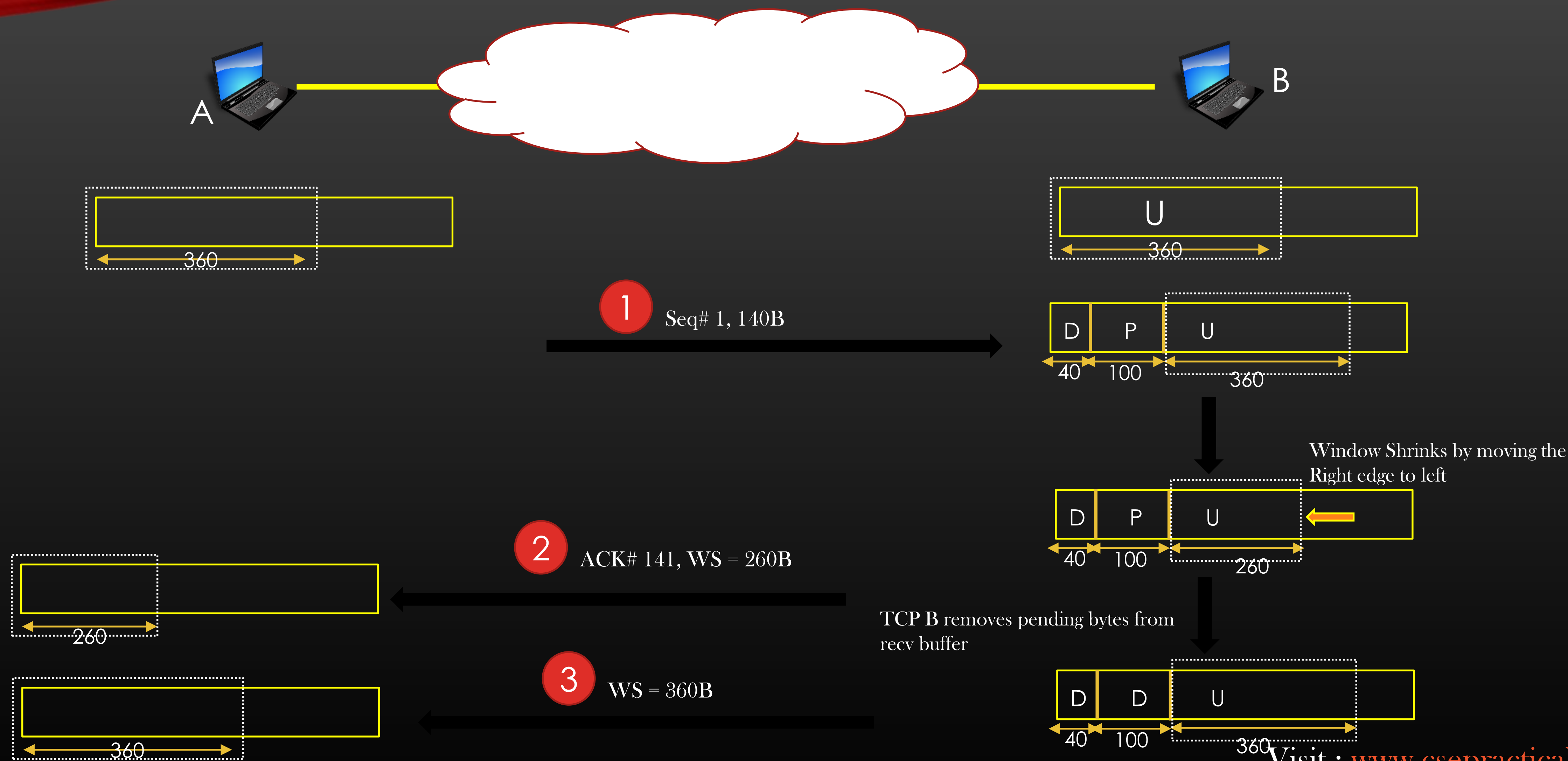
- B will drop the extra segments, causing A to retrigger retransmission, endless cycle . . .
- Slowness of B would ultimately lead to congestion in the network
- Solution : B should have a mechanism to tell A to slow down and send at slower rate
- Real World Scenario :
  Remember Machine B could be TCP server entertaining 10s of TCP clients at the same time, B may not be able to process the segments from each client instantly and may drop if clients overwhelms Server B

Problem Statement – Slow Receivers

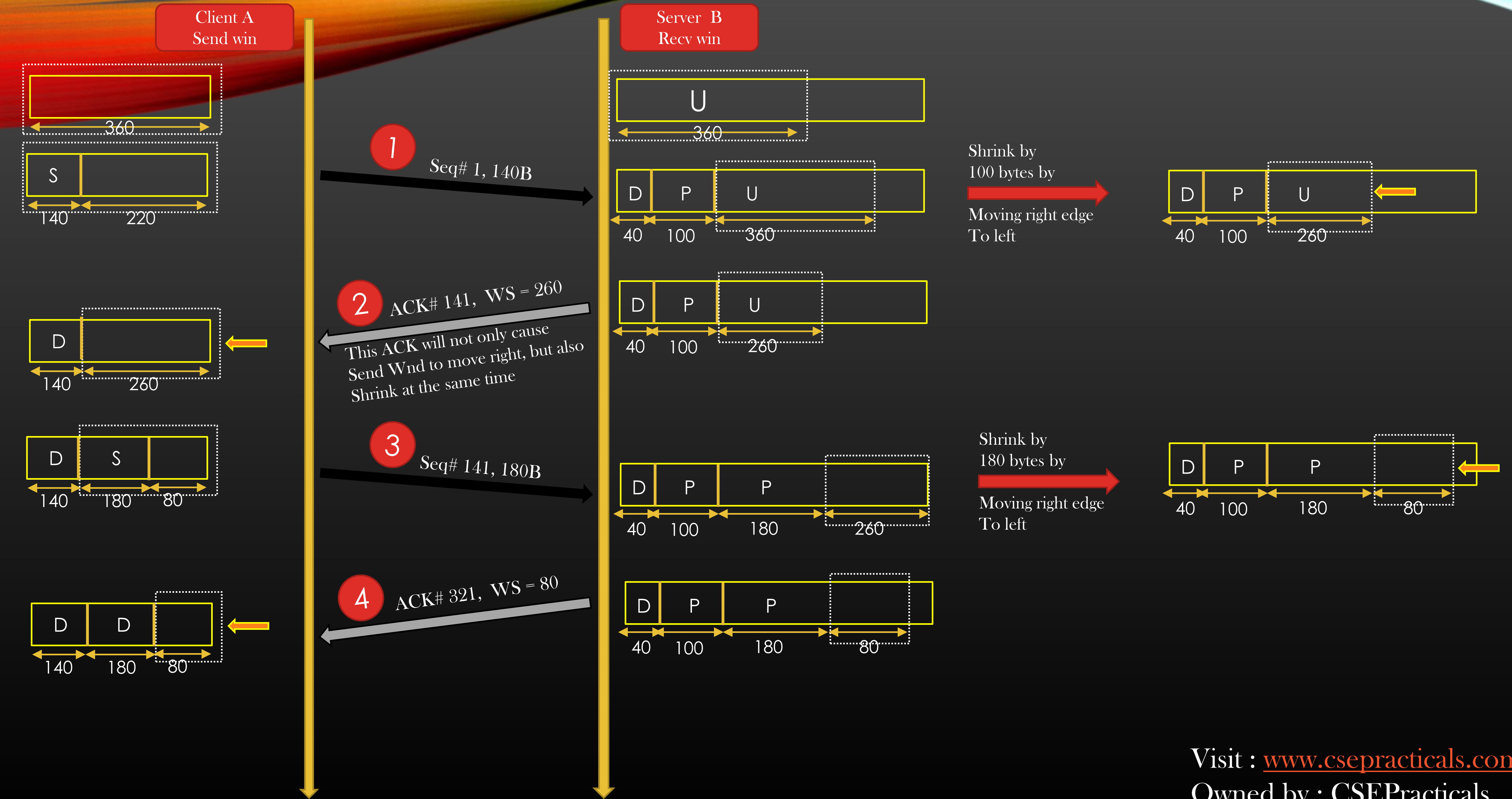


- Let us understand How Congestion *because of slow TCP receiver* can be avoided using window size reduction with the help of an example
- Let us assume
  - TCP Receiver (the server) is busy and it momentarily it cannot process the data being received from TCP sender

TCP Window ReSizing

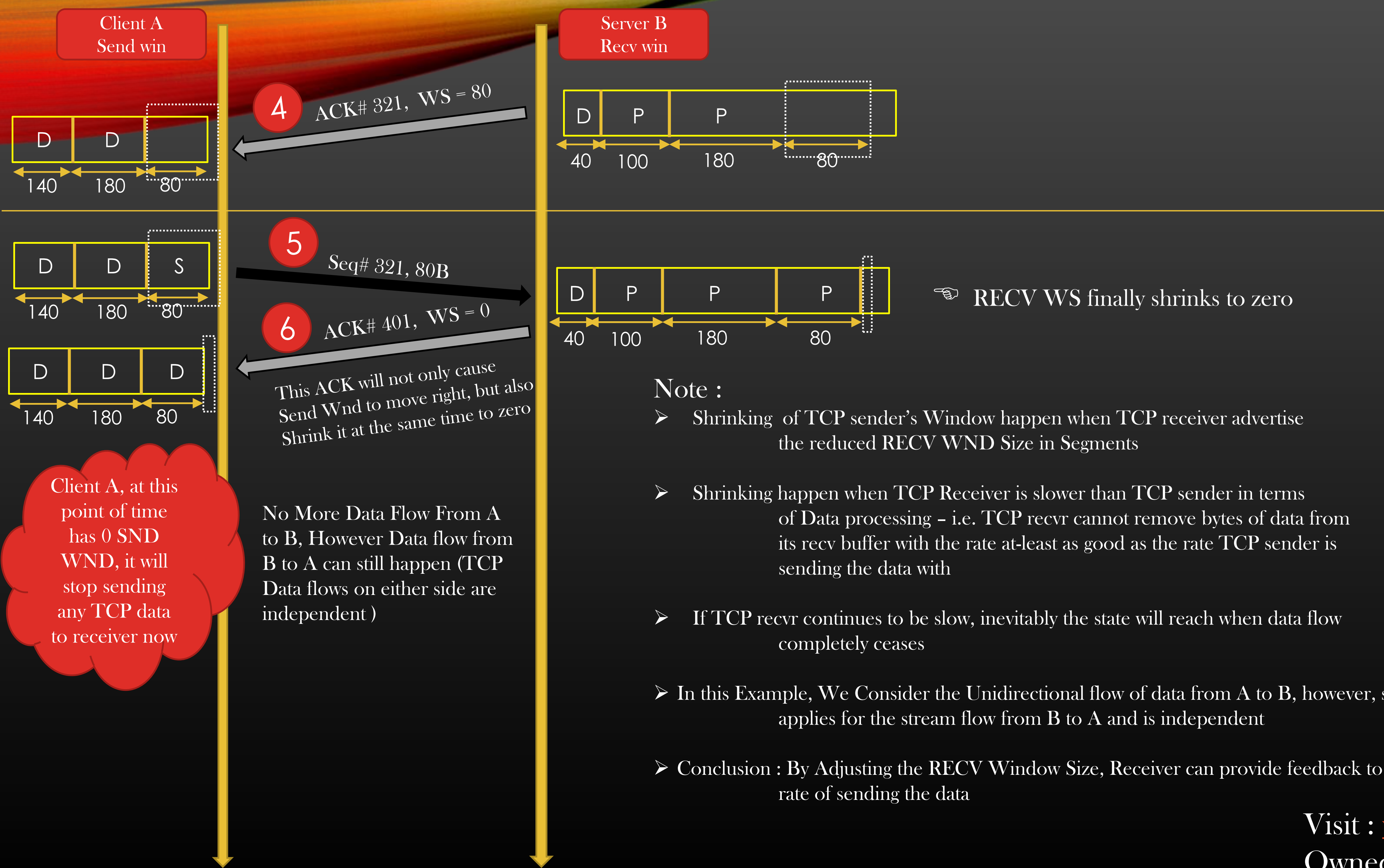


Visit : www.csepracticals.com
Owned by : CSEPracticals

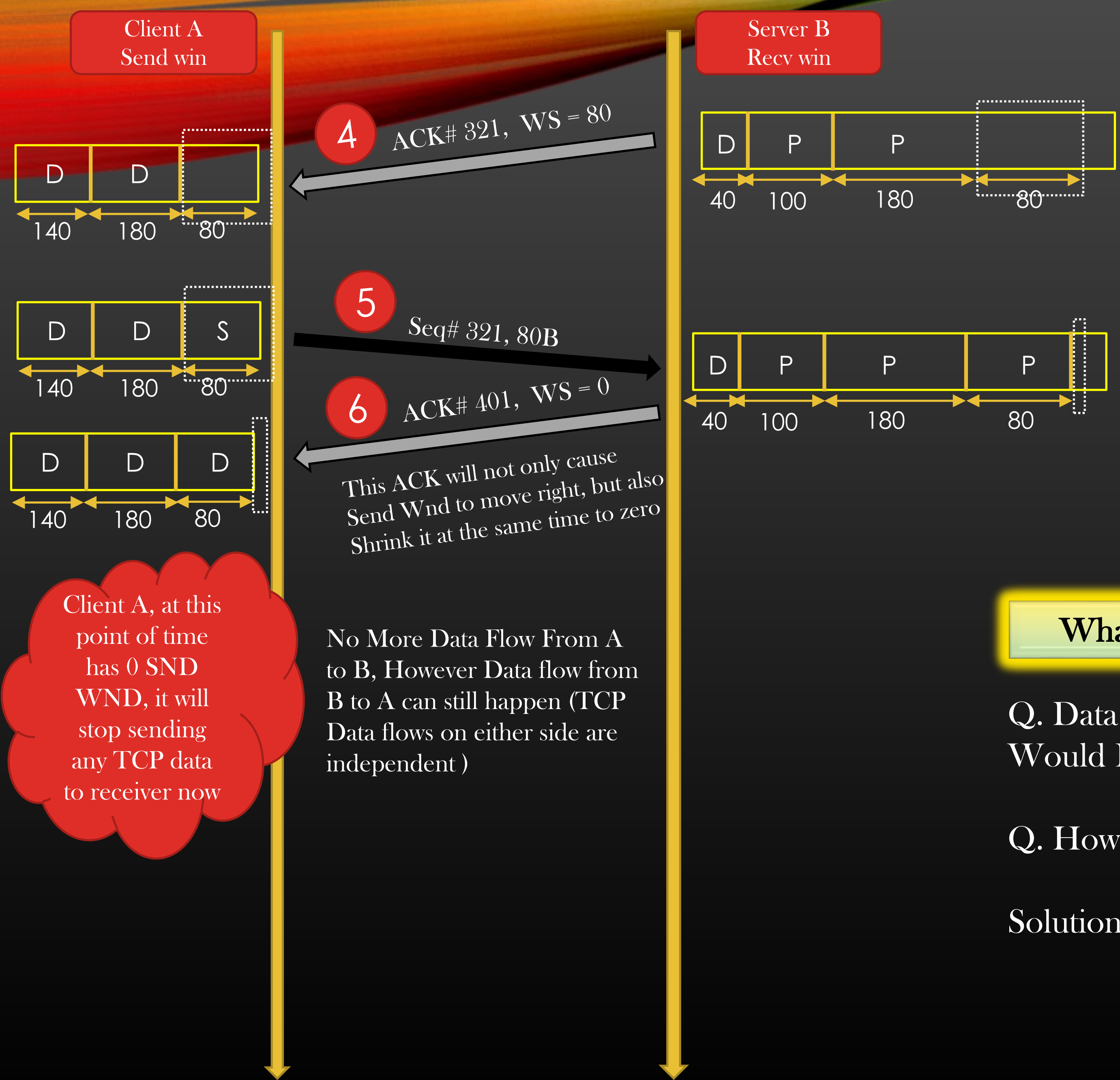


Visit : www.csepracticals.com
Owned by : CSEPracticals

Mastering TCP -> TCP Data Flow and Window Management -> Flow Control -> Window Size Reduction



Visit : www.csepracticals.com
Owned by : CSEPracticals



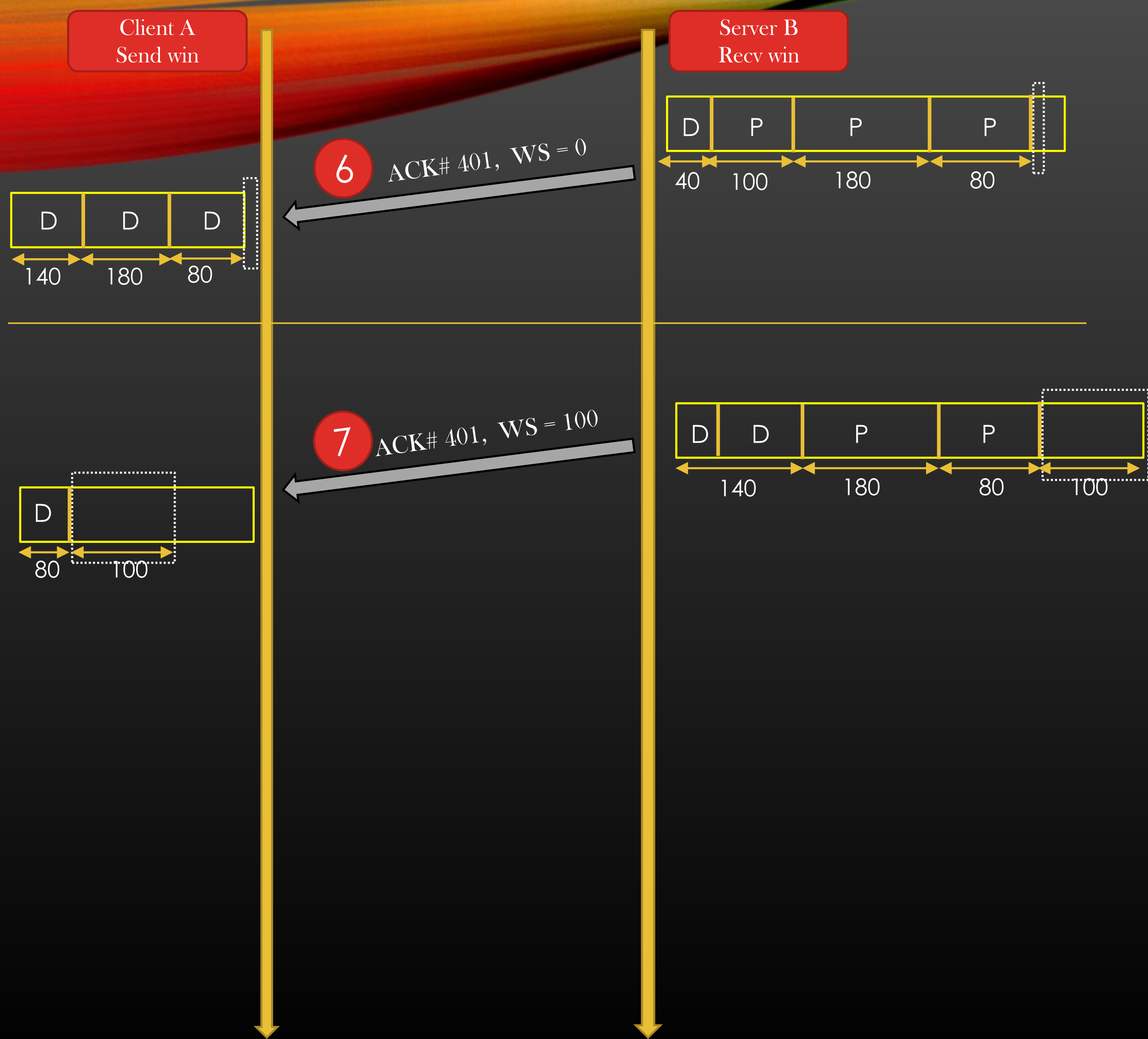
☜ RECV WS finally shrinks to zero

What Next !!

Q. Data flow cannot resume until A's SND WS grows again. Would Data flow from A to B ever restores ?

Q. How it will be restored ?

Solution : *Window Opening ACK Segments*



Window Reopening

- This Situation continues until the TCP Server removes some Pending bytes P from recv buffer and deliver to application
- Once some new room is created in recv buffer, Server Can enlarge or expand its recv window
- Server than generates a new pure ACK advertising a new size of its recv window

ACK, ACK # 401, WS = <new recv window size>

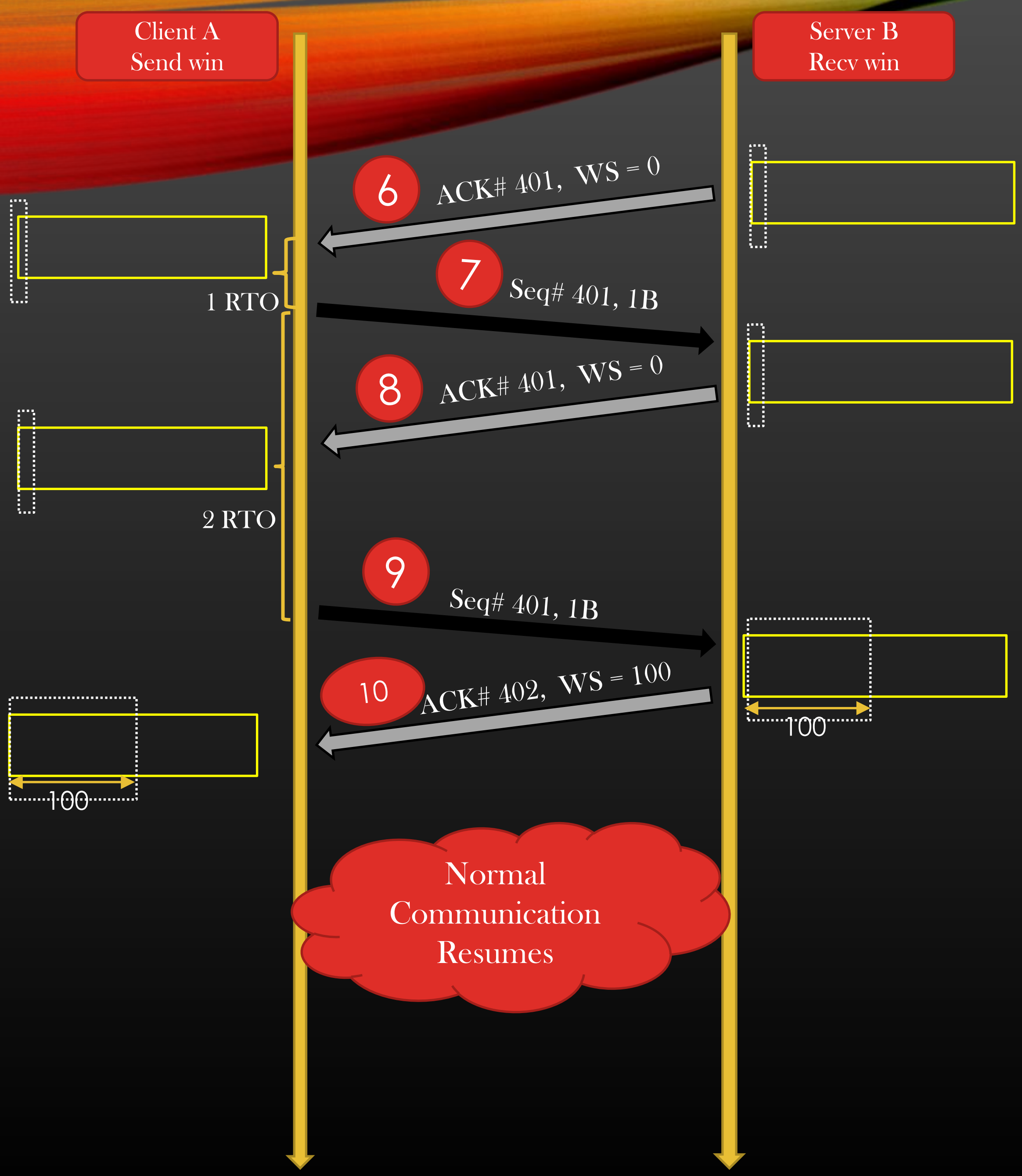
This new ACK is informally called *Window Opening ACK*

- Having Received this WOACK, TCP client reopens its send window by sliding the right edge to the right by the same amount and resume transmitting segments honoring the new send window size
- Communication is Resume from TCP Client to TCP Server once again

Visit : www.csepracticals.com
Owned by : CSEPracticals

Window Reopening

- But What if the WOACK (7) is lost !
- Remember TCP do not Acknowledge pure ACK segments, if they are lost they are lost forever.
- TCP is not reliable wrt to ACKs
- Deadlock !
- TCP B have no idea that WOACK has been lost, it believes TCP Sender has no data to sent
- TCP A would continue to have send window size = 0
- *Solution : Probe Segments*

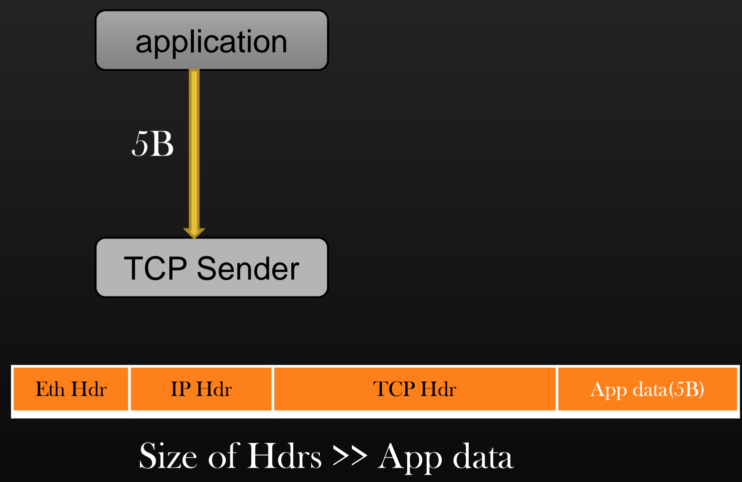


- The timer to send probe segments is called persist timer whose initial value is set to 1 RTO. Subsequent probe segments are sent as per exponential back off. TCP never gives up sending probe segments.
- Once Client Window Size reduced to Zero, it start sending Probe Segments periodically to the TCP server. Probe Segments are also called TCP ZeroWindowProbe Segments
- The purpose of the probe segments is to ask the status of of recv window
- Probe Segments contains 1 byte of APP data, meaning they are indistinguishable from regular data segments and hence TCP applies its retransmission policies to ZeroWindowProbe Segments i.e. retransmit them if they are lost (RTO time out Or dupack)
- TCP server replies probe segments with the pure ACK specifying current recv window size. These ACK are called TCP ZeroWindowProbeACK
- 7 and 9 and probe segments, and 10 is WOACK
- But one problem !!
  - If Server reopens its recv wnd by a very small size (say 5B), it will lead to transmission of data segments (C->S) of very small sizes which is inefficient (Network underutilization)
  - This problem is called Silly Window Syndrome which we discuss next, and discuss the solution

Visit : www.csepracticals.com
Owned by : CSEPracticals

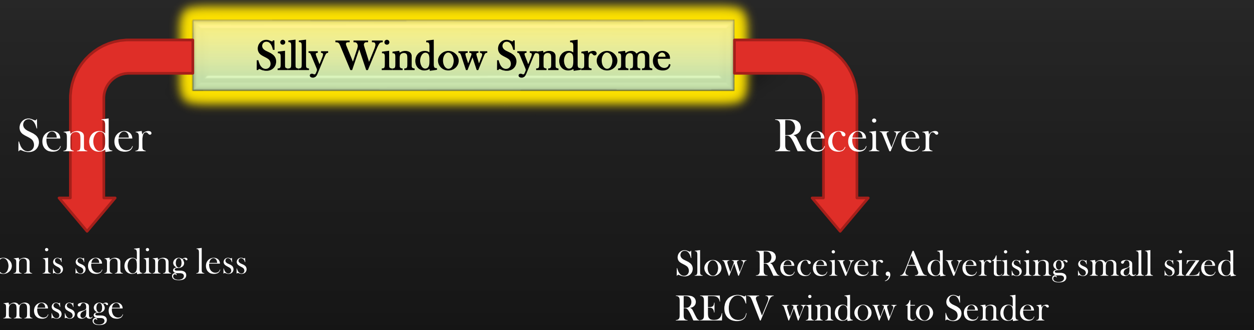
Silly Window Syndrome

- Silly Window Syndrome (SWS) is a situation When there is an exchange of small sized TCP data segments (Tinygrams) on a TCP connection
- This leads to Network under-utilization because useful data shipped per RTT is very less as compared to header overhead
  Analogy : Parceling a Bday gift worth $100, whereas cost of parceling is $1000 !!
- SWS can occur by defective TCP Sender Or TCP receiver



Size of Hdrs >> App data

SWS by TCP Sender



Sender:

Application is sending less bytes per message

TCP Sender do not Wait to accumulate enough application data and send immediately

Soln : Nagle Algorithm

Receiver:

Slow Receiver, Advertising small sized RECV window to Sender

Soln : SWS Avoidance Rules

Let us Discuss Rules to avoid SWS problem By Sender or Receiver

Visit : www.csepracticals.com
Owned by : CSEPracticals

Silly Window Syndrome Avoidance Rules

- SWS Avoidance rules encourage TCP to stop data flow completely rather than exchanging data in TCP Tinygrams

TCP Sender Rules (Send Segment when at-least one of the below condition are true)	TCP Receiver Rules (Do not Advertise the increased size of RECV Window until)
A full-size (MSS bytes) segment can be sent.	Usable Recv Window Size > = MSS Or Usable Recv Window Size > = ½ size of Receiver's buffer space Whichever is smaller
TCP can send at least one-half of the maximum-size window that the other end has ever advertised on this connection	
Send a segment immediately if there is no outstanding ACK i.e. all prev segments sent has been ACKd (Nagle Algorithm)	
Last Choice : send whatever TCP sender have if Nagle algorithm is disabled for this connection	

Now, Let us Practice the SWS avoidance with the help of example,

But before that we Need to understand Window Shrinkage Avoidance (WSA)

Visit : www.csepracticals.com
Owned by : CSEPracticals

Silly Window Syndrome -> Window Shrinkage Avoidance

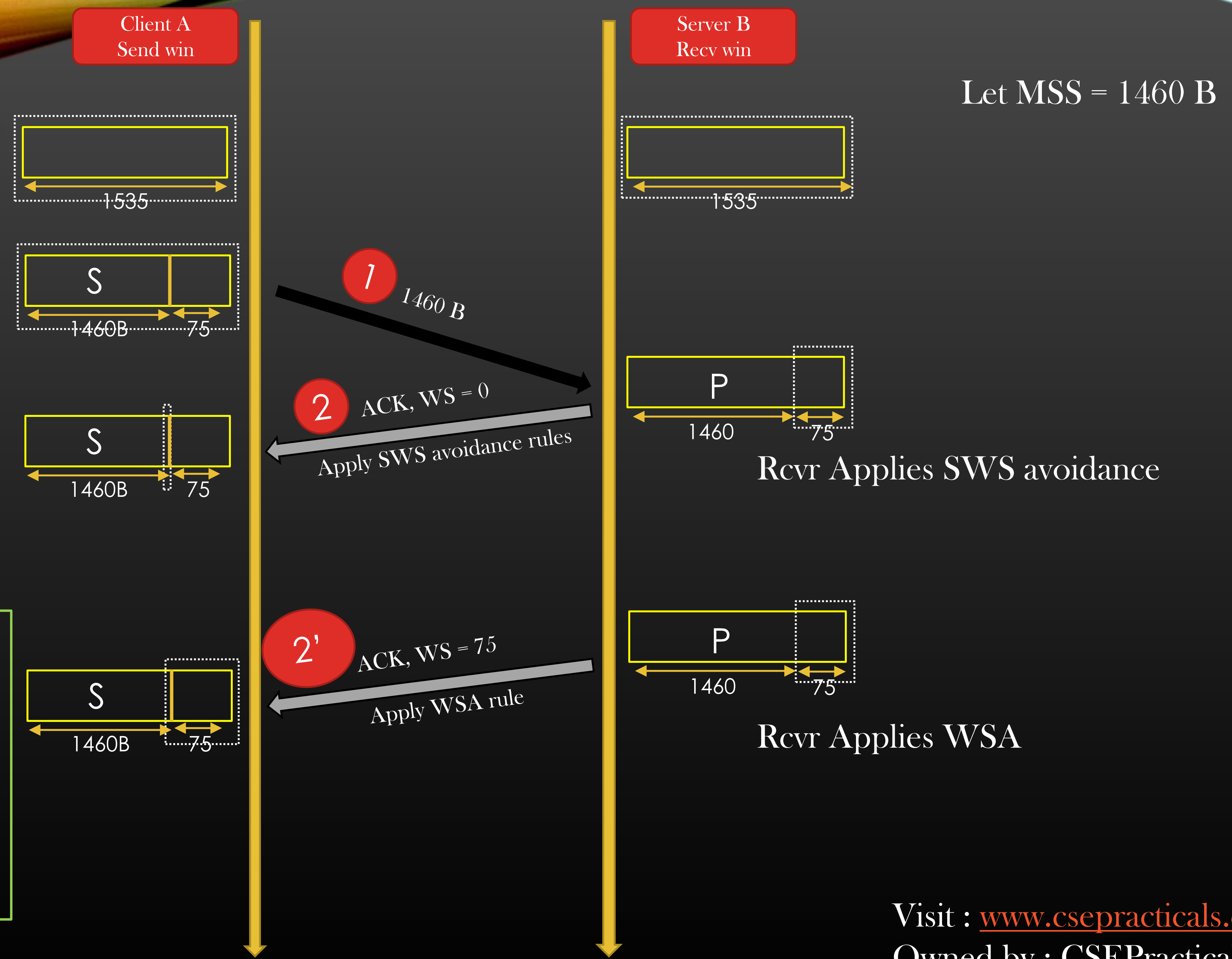
- We shall go through real example where we shall witness sender and receiver exercising SWS rules
- But before that we need to cover the last topic of SWS – Window Shrinkage Avoidance
- WSA is done by TCP receiver only on its RECV Window (since TCP sender's send window = TCP Receiver's recv window size =, effect of WSA also impact TCP sender's send window)
- WSA is enforced when TCP recvr's recv window usable size (empty space) reduced to less than MSS or ½ of original recv buffer size
- Here reduced means decreased from higher value to lower value, WSA is not enforced when recv window usable size is incremented
- Purpose of WSA is to prevent the right edge of TCP receiver's recv window to move to left
- I understand, Example is required !!

Silly Window Syndrome -> Window Shrinkage Avoidance -> Example

> Right Edge of the Window is moved left because recvr advertised Zero window size. This is called Window Shrinking. WSA aims to avoid this.

> Left edge of the window moves towards right as usual because 1460 data has been acknowledged

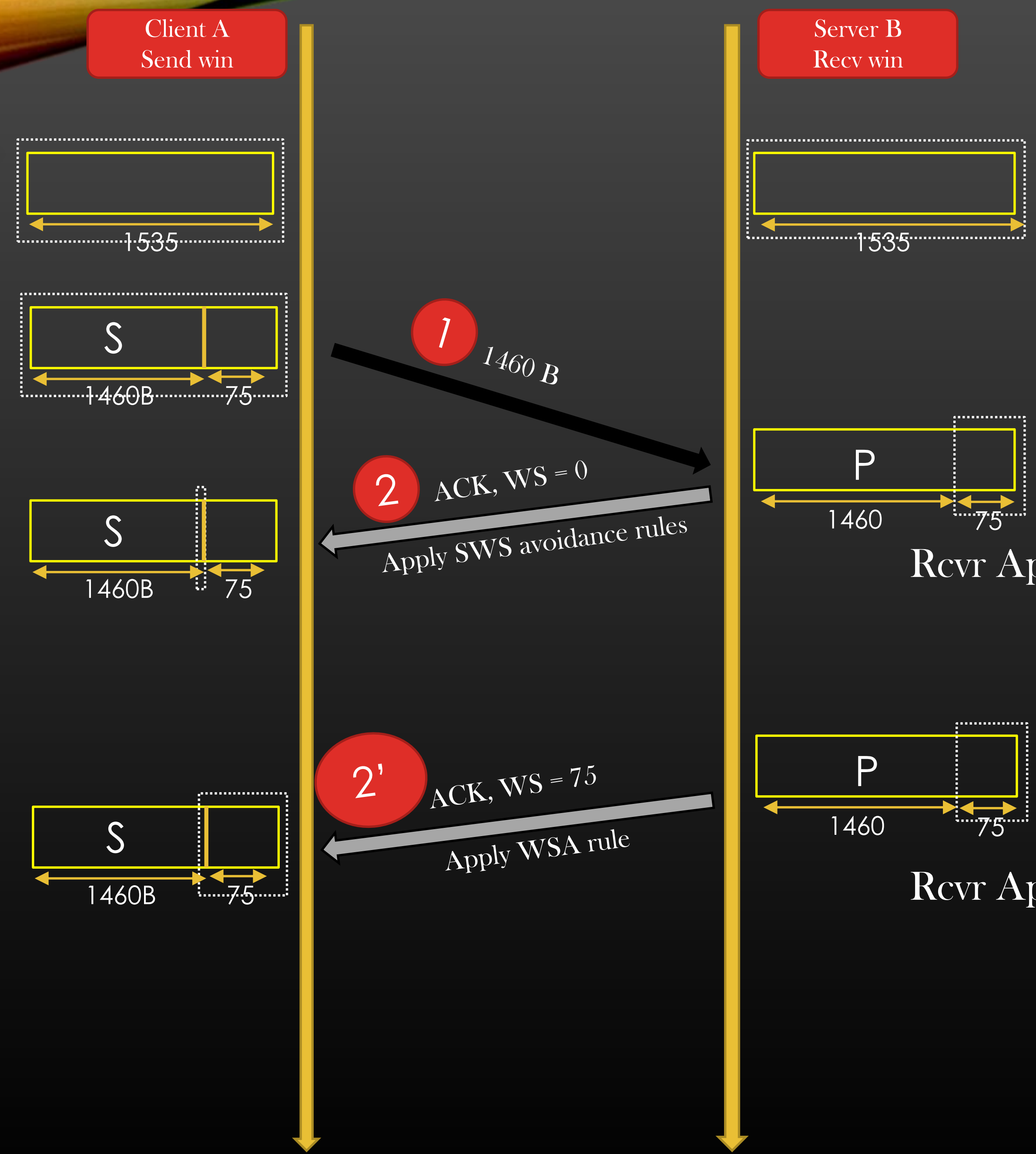
- Right Edge of the Window do not moved left Window is not Shrinked.
- Left edge of the window moves towards right as usual because 1460 data has been Acknowledged
- Sender now applies Sender's SWS avoidance algorithm to send next data segment



Visit : www.csepracticals.com
Owned by : CSEPracticals

Silly Window Syndrome -> Window Shrinkage Avoidance -> Example

Let MSS = 1460 B



Conclusion :

- Whenever the Recvr zero window advertisement causes Sender's send window to shrink, TCP recvr applies WSA over SWS

Visit : www.csepracticals.com
Owned by : CSEPracticals

Silly Window Syndrome Avoidance in Action

- Now we shall go through real world example where we have collected segments for unidirectional communication between client (Sender) and Server (Receiver)
- Needless to mention, we take unidirectional communication to understand the concept, whereas all concept applies to the data flows in either direction
- Instead of Arrow based Diagram, we will use a table this time
- This example will illustrate :
  - Handshake
  - SWS avoidance
  - WSA
  - Zero window Advertisement
  - ZeroWindowProbe Segments
  - ZeroWindowProbeACK Segments
  - Window Opening ACK Segments

Silly Window Syndrome Avoidance in Action -> Example

- Refer to Excel Sheet Doc

TCP

Congestion Control

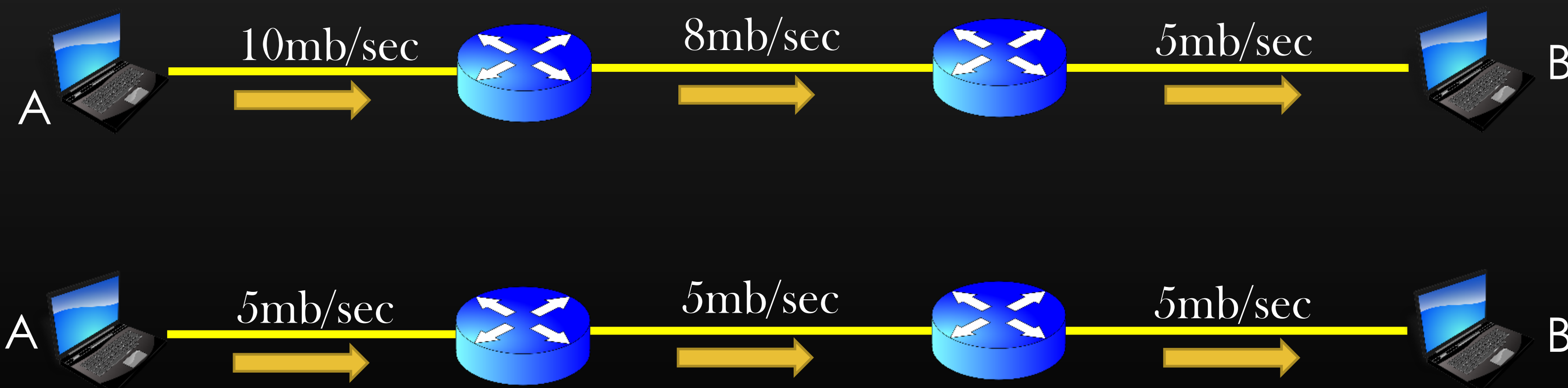
Time for TCP Sender To take responsibility

Visit : www.csepracticals.com
Owned by : CSEPracticals

- We learnt : Window Based Flow Control is triggered by overwhelming TCP Receiver (by reducing the window size) which is finding it difficult to process the bytes at the rate at which the sender is sending it

- But, What, if TCP receiver is not slow, but it is the network in the middle between Sender and Receiver which is slow. In this case, TCP receiver would not reduce its recv window size because it cannot find :
    - Whether TCP sender itself is sending bytes at low rate
    - TCP sender is not slow but Network is congested and dropping the segments making Sender appear slow

- To cope-up with the slow network (slow routers, slow links, less memory etc), TCP uses its *Congestion Control Procedures* which we shall discuss in this section. CCP is triggered by sender without any assistance/feedback from TCP receiver like in case of flow control



B do not know if A is sending data at 10mb/sec or 5 mb/sec

- Goal : TCP Sender must slow down when it has reason to believe the network is about to be congested
  TCP Sender must speed up when it has reason to believe the network is recovered

- Challenge :
  - The challenge is to determine exactly when and how TCP should slow down, and when it can speed up again

Flow control Deals with Slow Receivers, and is driven by Receivers
Congestion Control Deals with slow Networks, and is driven by TCP Sender

What is Congestion ?

The situation when a router or other network entities is forced to discard data because it cannot handle the arriving traffic rate, is called Congestion.

Congestion can cause the performance of a network to be reduced so badly that it becomes unusable

TCP implements Congestion Control Procedures to deal with slow/congested networks

Without CCP, slow network would drop packet only to trigger TCP Sender to retransmit lost segments – making the situation even worse. CCP enable TCP Sender to adopt itself to ever changing dynamic Network state

- There is no explicit signaling mechanism to detect the existence of congestion in the network (Recall But in flow control there was signaling mechanism)
  - Slow-down routers would not send any feedback to TCP sender to report the existence of congestion
  - Instead, TCP sender has to be self-sufficient to detect the situation of congestion. It has no help from middle-men network entities

- CCP can be roughly divided into three parts
  - 1. TCP sender somehow detect that congestion is about to happen
  - 2. TCP Sender slow down the rate of sending segments, and determine how slow
  - 3. TCP sender somehow should be able to detect that network congestion state is improved, and it can increase the rate of sending data, and also determine how fast

Congestion Window

TCP Sender Must inject packet in the network at the rate at which network can handle, or Receiver can handle, whichever is less

> Receiver's RECV window restrict the sender from injecting the packets at the rate recvr cannot handle
> But how to restrict the sender from injecting the packets at the rate Network can handle – We need an additional restriction on TCP sender's send window, and that restriction is additional window – *Congestion Window*

$$W = \min(cwnd, awnd)$$

cwnd – size of congestion window
awnd – size of recvr's advertised window

> *Congestion Window is the measure of Network capacity*
> using the above relation, TCP sender is allowed to send W more bytes into the network

Note : We have already seen awnd is variable and keep on changing during the course of communication
likewise, cwnd is also a variable and keep on changing depending on traffic-carrying capacity of the network

Thus, values of *W, cwnd, awnd* have to be dynamically updated by the TCP sender during the course of TCP connection
We shall see the *W, cwnd, awnd* collaboratively work with the help of example shortly

Congestion Control Algorithms

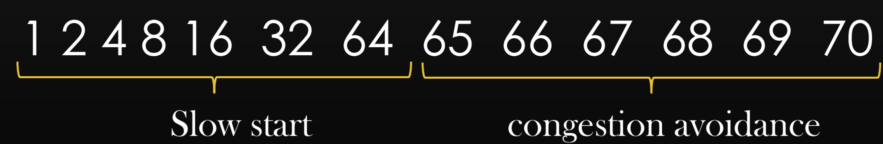
- TCP Congestion Control Procedures involves two algorithms :

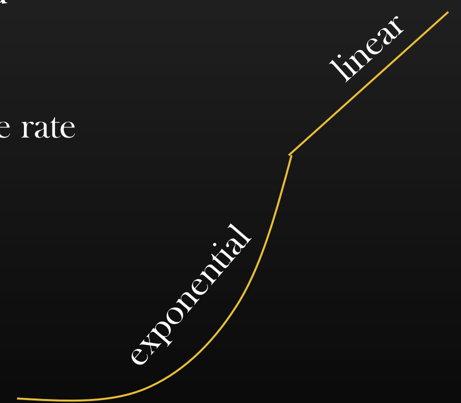
Slow Start

- Executed when Connection is established afresh
- For new fresh Connection, TCP sender do not know the appropriate value of *cwnd, therefore cwnd = 1MSS*
- Remember, cwnd is the estimate of network capacity
- Goal : Determine the accurate value cwnd, which allow sender to send data at the throttle rate
- Mechanics : TCP Sender starts injecting packet in the network, starting at a lower rate, and increasing the rate exponentially and keep on increasing Until Certain conditions C are met

Congestion Avoidance

- Executed Immediately after slow start has finished
- By this time, appropriate value of cwnd has been determined
- TCP continue to inject more packet increasing the rate linearly until packet loss is detected again





Visit : www.csepracticals.com
Owned by : CSEPracticals

Slow Start Algorithm

Goal : To determine the maximum rate at which the TCP sender can inject the segments into the network without experiencing packet loss.

Slow Start Algorithm is triggered on TCP sender side When :

1. New Connection has just established
2. Retransmission timeout (RTO) for a data segment happen (pkt loss)
3. When TCP sender do not send any data and stay idle for some time

To begin with, Initial value of cwnd is set to 1MSS in above three cases. Therefore no of Bytes Sender can send in the first data-segment is W :

$$W = \min (cwnd = 1, awnd)$$

Let us See Slow start Algorithm in Action . . .

Slow Start Algorithm in Action

Let, all units are in MSS for simplicity
Initial Cwnd = 1
MSS = 1460B
awnd = 10

SND W = min(cwnd, awnd)

Thus, per RTT, cwnd is doubled. This is called *Multiplicative increase*

SND W = 10 and become Stable here

W = 10

Sender A

Rcvr B

3-way handshake done

Cwnd =1
SND = 1

1

2

Cwnd =2
SND = 2

2

3

4

Cwnd =4
SND = 4

4,5,6,7

8

Cwnd =8
SND = 8

8, 9 . . .15

16

Cwnd =16
SND = 10

16,17 ...25

26

Cwnd =16
SND = 10

Visit : www.csepracticals.com
Owned by : CSEPracticals

Slow Start Algorithm

Points to Remember :

1. *cwnd* is doubled per *good ACK* only. Good ACK is the ACK whose ack# is the largest ever recvd by TCP sender

2. if *awnd* is very large (2 ^ 16), then cwnd keeps on doubling per good ACK received. A stage is reached when cwnd shall be so large that Sender would experience a packet loss.

3. Now Some Questions :

Q. For How long the Slow Start Algorithm Executed by TCP Sender ?
Q. When Would TCP sender switch from Slow Start to Congestion Avoidance Phase (Or Vice Versa) ?
Q. What TCP sender Would do if it experience a segment loss in slow start phase Or Congestion Avoidance Phase ?

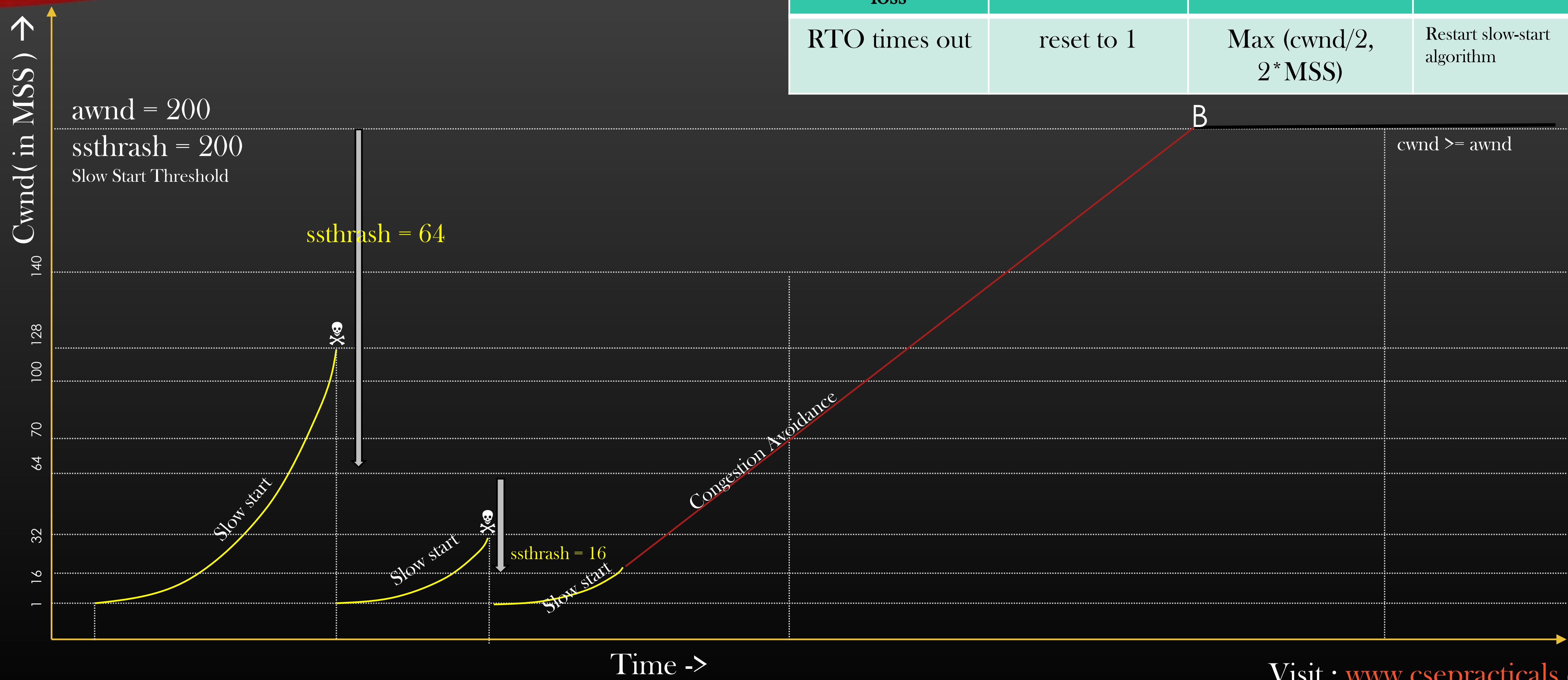
Let us find answer to these Questions step by step …

But before that, Let us take the graphical example of slow start algorithm in Action !

Graph of Slow Start Algorithm

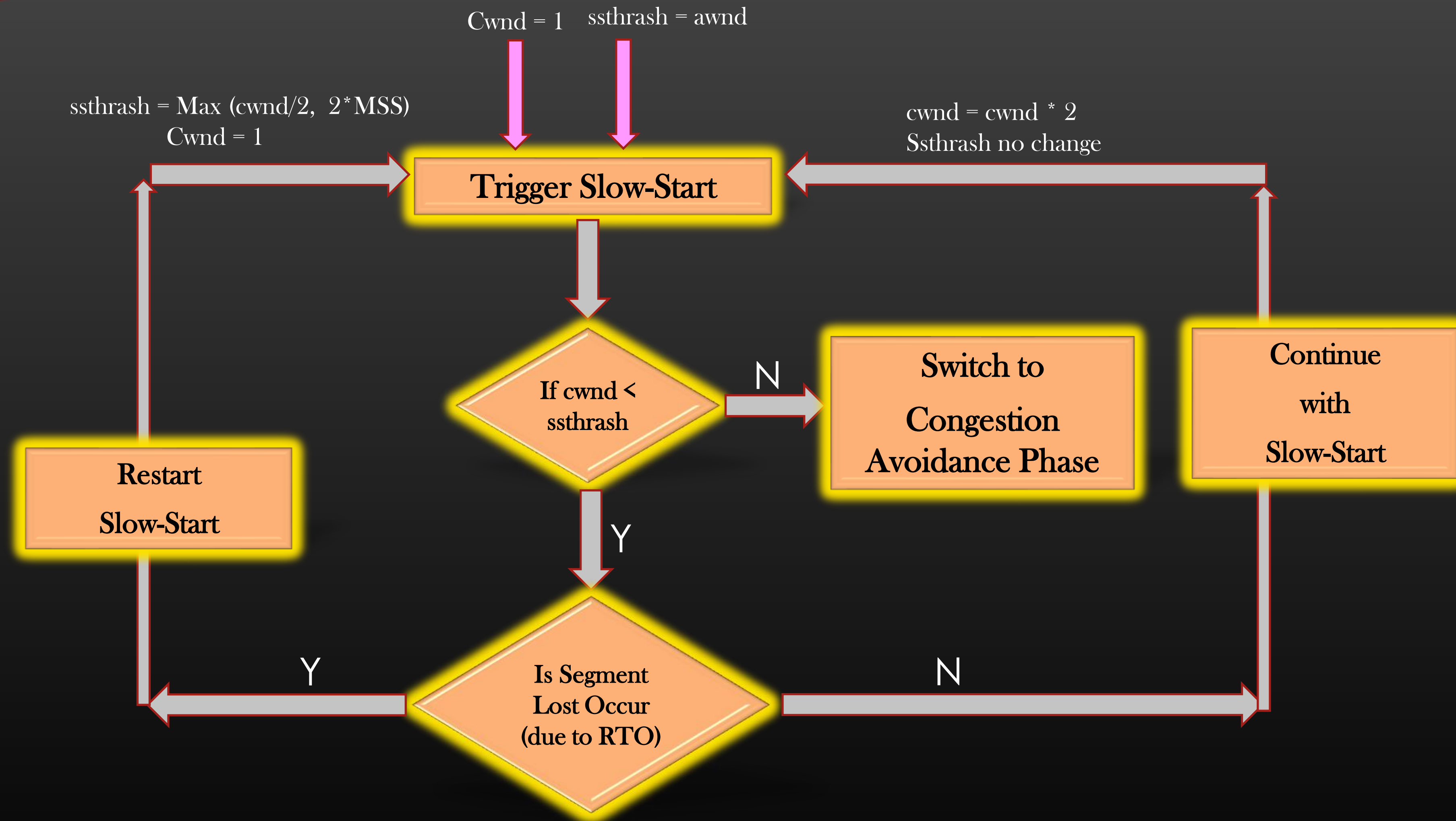
☠ -- packet loss due to RTO

Reason for pkt loss	cwnd	ssthrash	explanation
RTO times out	reset to 1	Max (cwnd/2, 2*MSS)	Restart slow-start algorithm



Visit : www.csepracticals.com
Owned by : CSEPracticals

Slow Start Flowchart



Visit : www.csepracticals.com
Owned by : CSEPracticals

Congestion Avoidance

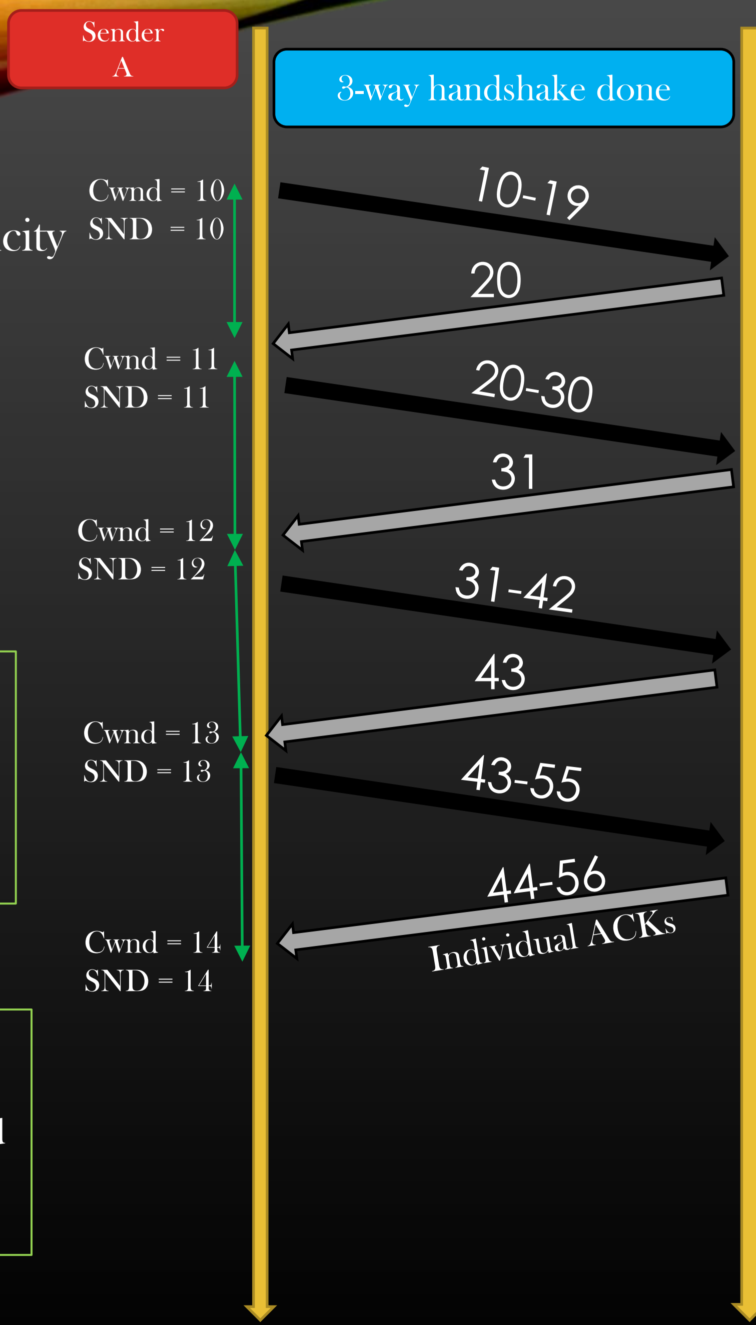
- TCP is always in a constant try to send as maximum as possible the data bytes into the network while respecting :
  - The network traffic carrying capacity and
  - receiver's capability

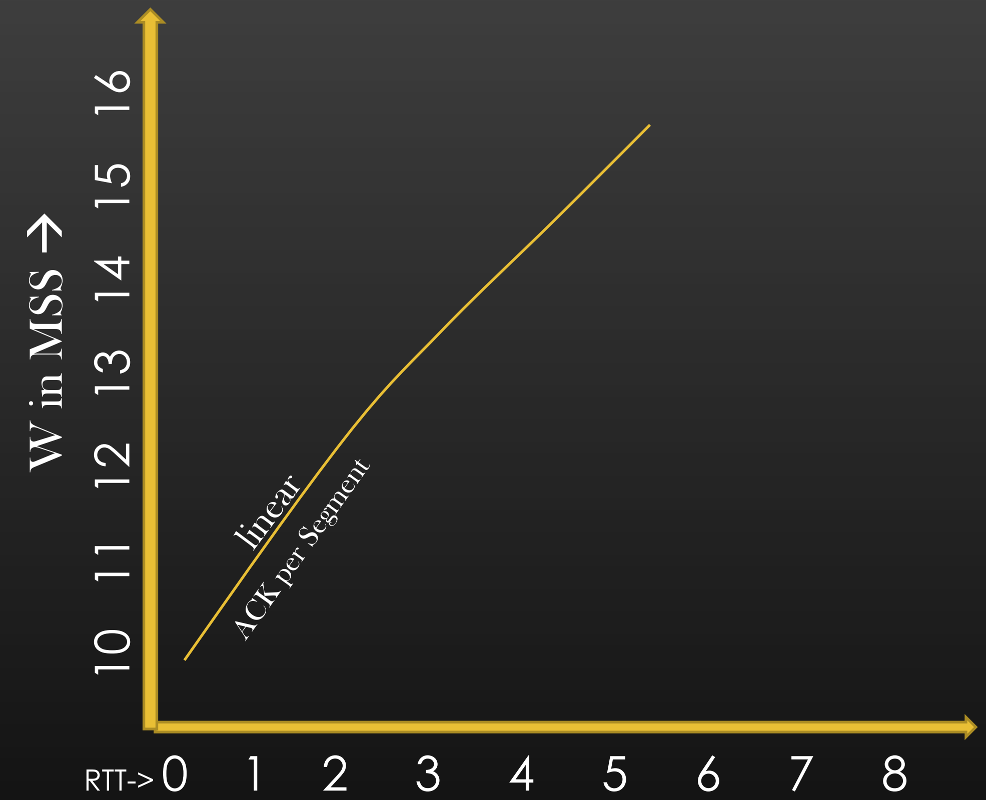
- In CA phase, TCP Sender keep probing the network for any additional bandwidth/capacity if it has to offer to the connection, but, like slow-start, TCP do not probe network as aggressively in CA phase

Slow Start:	Congestion Avoidance:
Cwnd (and hence SND Window) was increased exponentially for each successfully recvd good ACK This was called *Multiplicative increase*	Cwnd (and hence SND window) is increased linearly by 1MSS for each successfully recvd Good ACK. This is called *Additive increase*

Congestion Avoidance In Action

Let, all units are in MSS for simplicity
Cwnd = ssthrash = 10
MSS = 1460B
awnd = Infinite





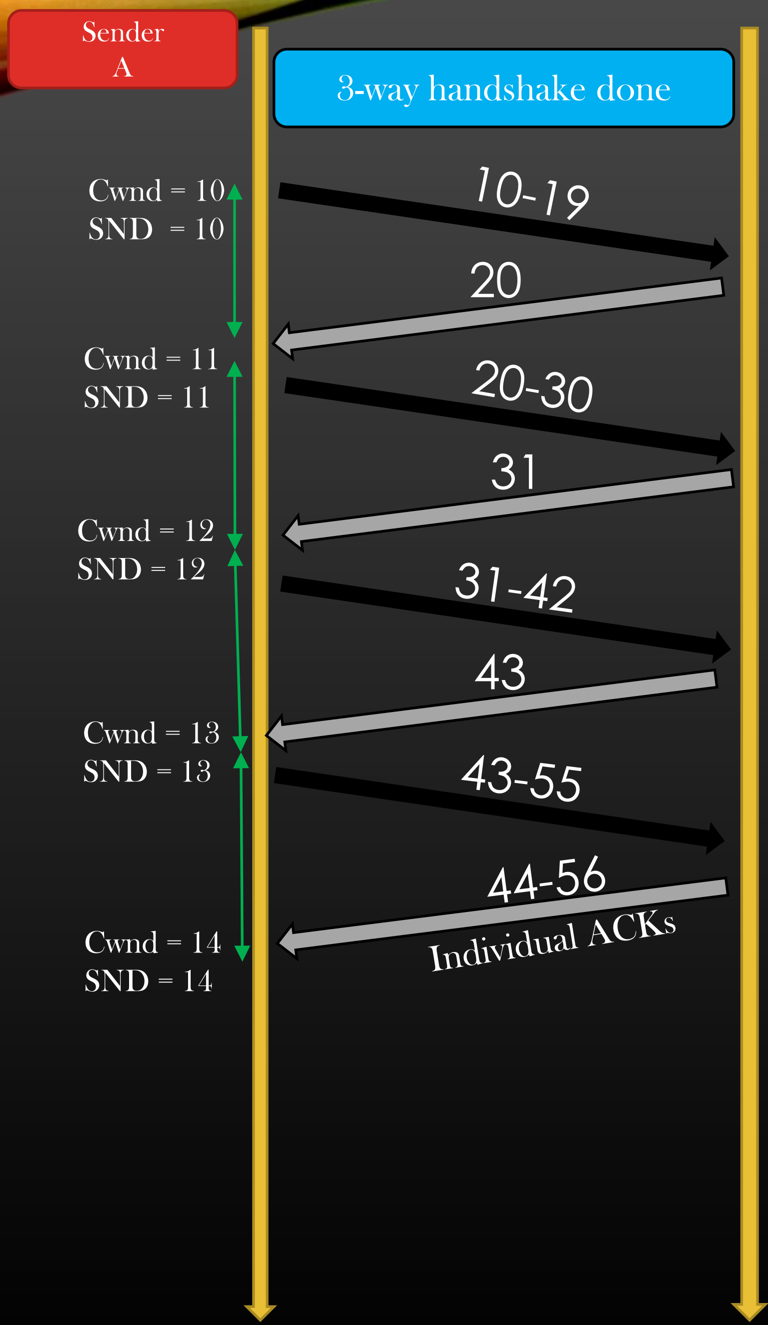
Cwnd is incremented linearly by 1MSS for each successfully recvd Good ACK
This is called *Additive increase*

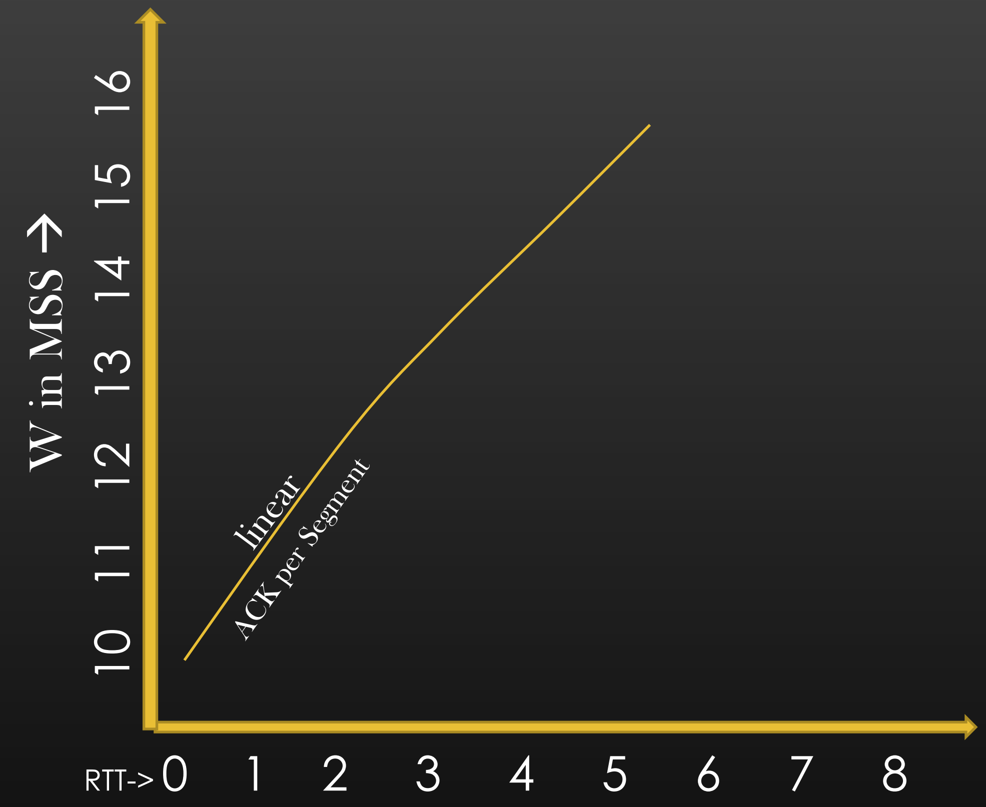
When TCP Sender Receives Individual ACKs in quick succession, it treats them As one single good ACK, and increase cwnd By 1 unit only.
(same applies to slow start algorithm)

Visit : www.csepracticals.com
Owned by : CSEPracticals

Congestion Avoidance In Action

- Since, Sender is receiving ACK without any packet loss, it Keeps increasing its cwnd by 1MSS, and Hence linearly increasing the rate of Sending segments into network Until network gives up (pkt is lost)
- When TCP Sender detects the packet loss in congestion avoidance phase, it triggers *congestion control selection procedure, coming up Next ...*





Visit : www.csepracticals.com
Owned by : CSEPracticals

Congestion Control Algorithm Selection

- Now that we have learnt two CCP : Slow-Start and Congestion Avoidance, Let us try to put them both together

- The two Algorithms are mutually exclusive , i.e. exactly one of them is in execution at a any given point of time, the two never runs simultaneously

- ***How TCP Decides which algorithm it should execute : Slow start or Congestion Avoidance, and when ?***

- Remember, we talked about *ssthrash* – the final value of *cwnd/2* when slow start exits due to RTO timeout (segment loss). The value of *ssthrash* determines which algorithm to execute next : *slow start or congestion avoidance.*

- Initial Value of ssthrash when Connection starts is set to awnd

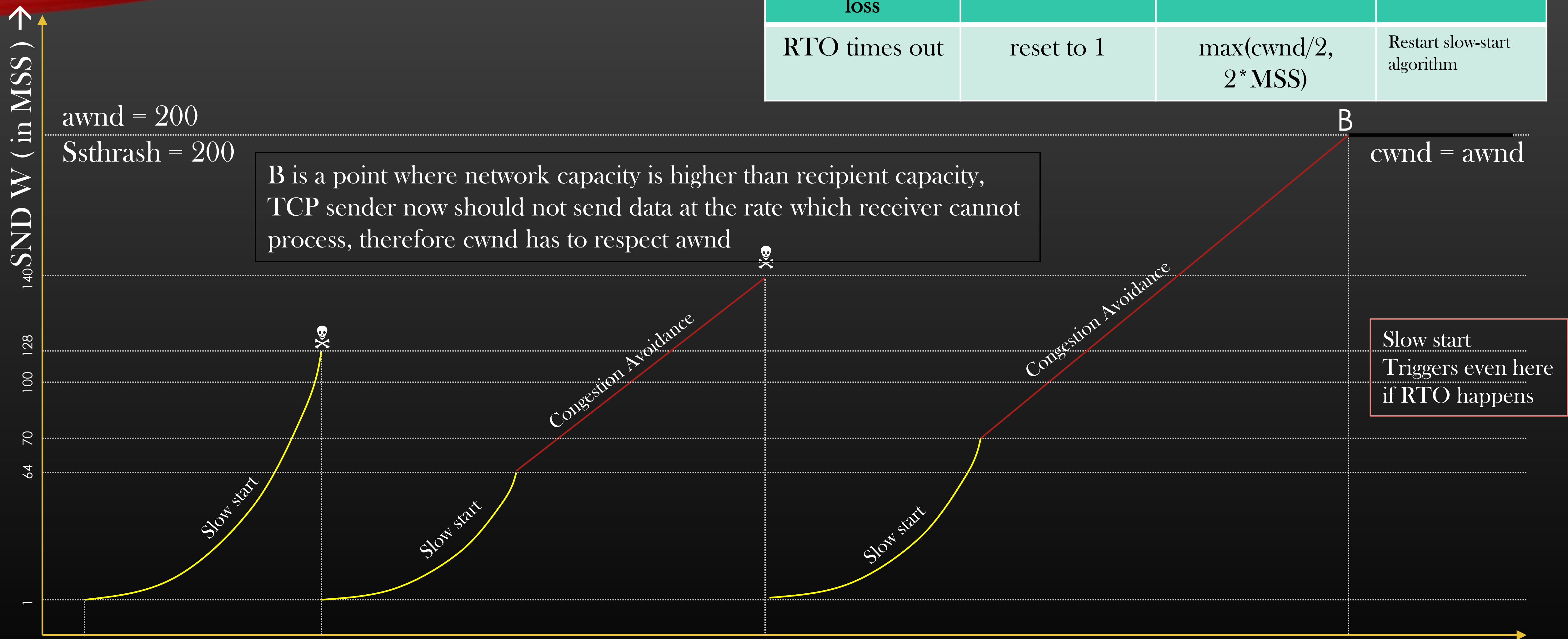
- Let us try to visualize the slow-start and congestion Avoidance algorithm put together once again . . .

Visit : www.csepracticals.com
Owned by : CSEPracticals

Slow Start and Congestion Avoidance – Put together

☠ -- packet loss due to RTO

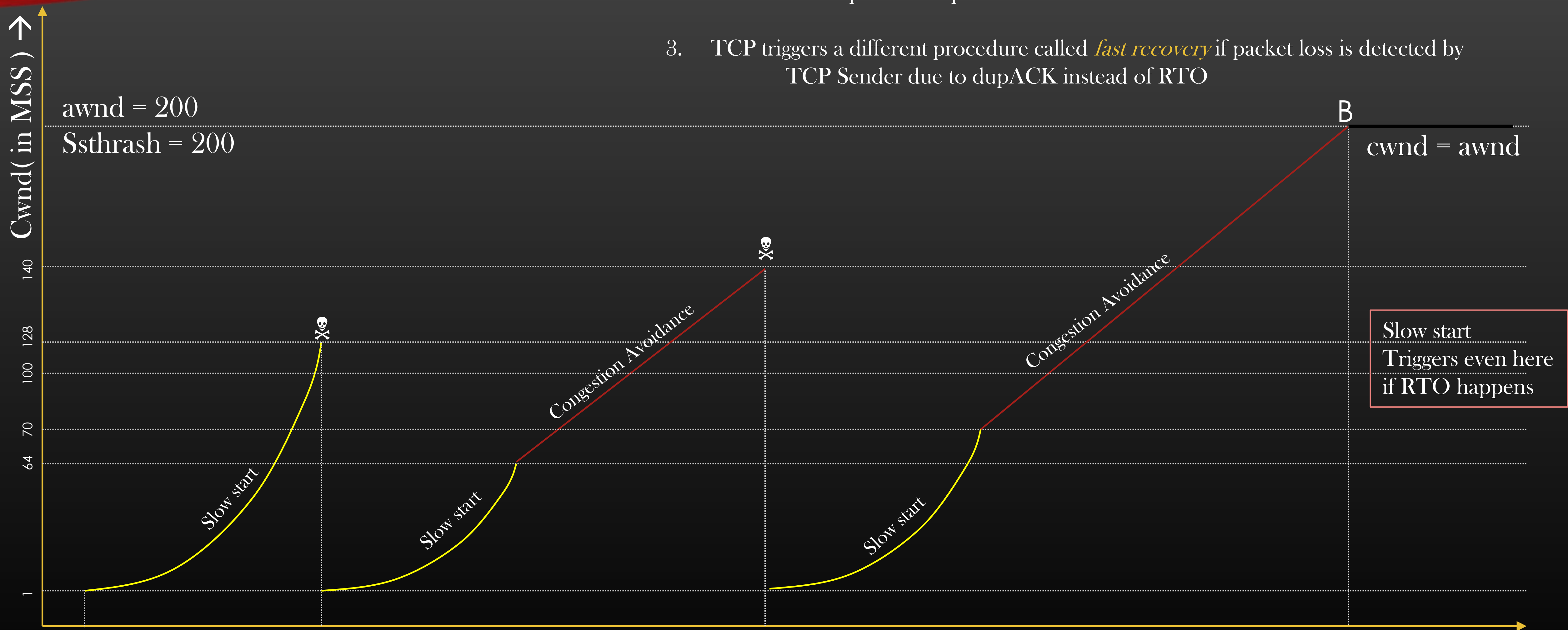
Reason for pkt loss	cwnd	ssthrash	explanation
RTO times out	reset to 1	max(cwnd/2, 2*MSS)	Restart slow-start algorithm



Visit : www.csepracticals.com
Owned by : CSEPracticals

Need for Fast Recovery

1. Every time RTO happens, TCP has to start from beginning -> Network Underutilization
2. We have assumed, pkt loss is detected by TCP sender due to RTO and not due to reception of dupACKs.
3. TCP triggers a different procedure called *fast recovery* if packet loss is detected by TCP Sender due to dupACK instead of RTO

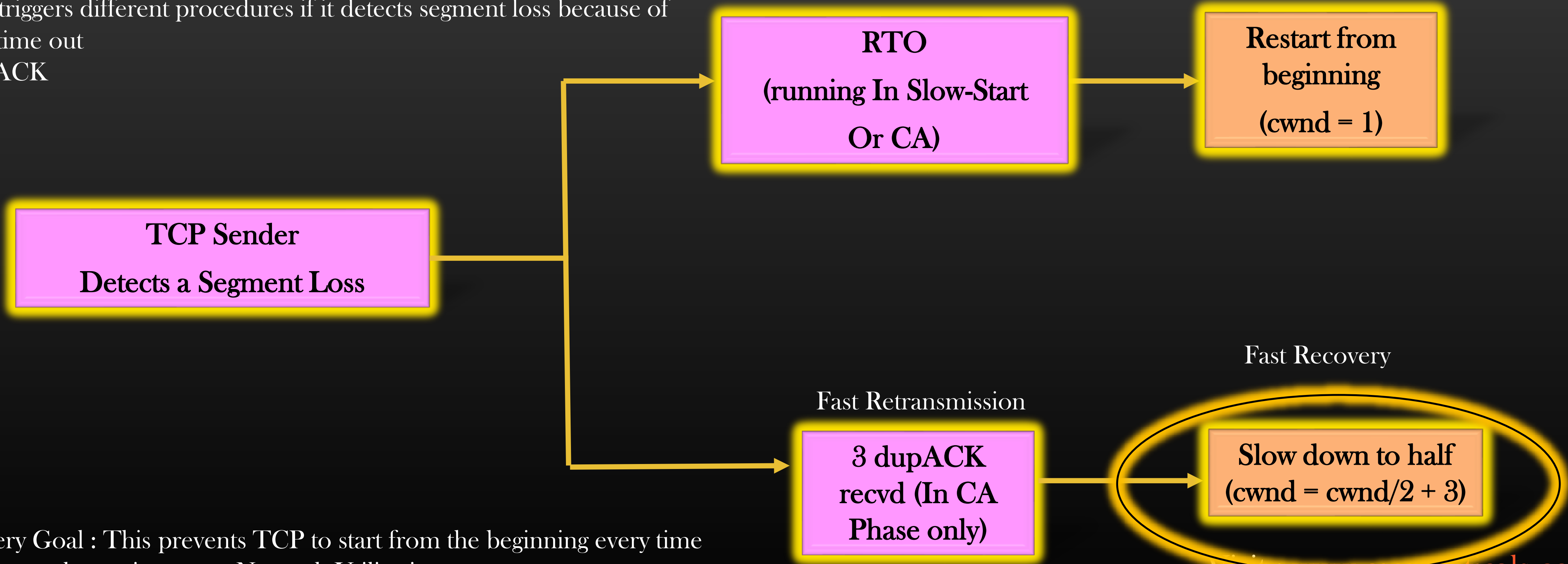


Fast Recovery

- Fast Recovery is the process by virtue of which TCP sender avoids restarting from the very beginning (cwnd = 1), instead it choose to slow down the rate of data to almost half when pkt loss Is detected.

- Whenever packet loss is detected, cwnd and ssthrash variables both are updated by TCP sender. How these values are updated depends on how the TCP sender detects the packet loss – Due to RTO Or reception of 3 dupACKs

- TCP sender triggers different procedures if it detects segment loss because of
  - RTO time out
  - 3 dupACK



☞ Fast Recovery Goal : This prevents TCP to start from the beginning every time segment loss occurs , hence improves Network Utilization

Visit : www.csepracticals.com
Owned by : CSEPracticals

Fast Recovery – Window Variables Updates

☠ -- packet loss due to RTO

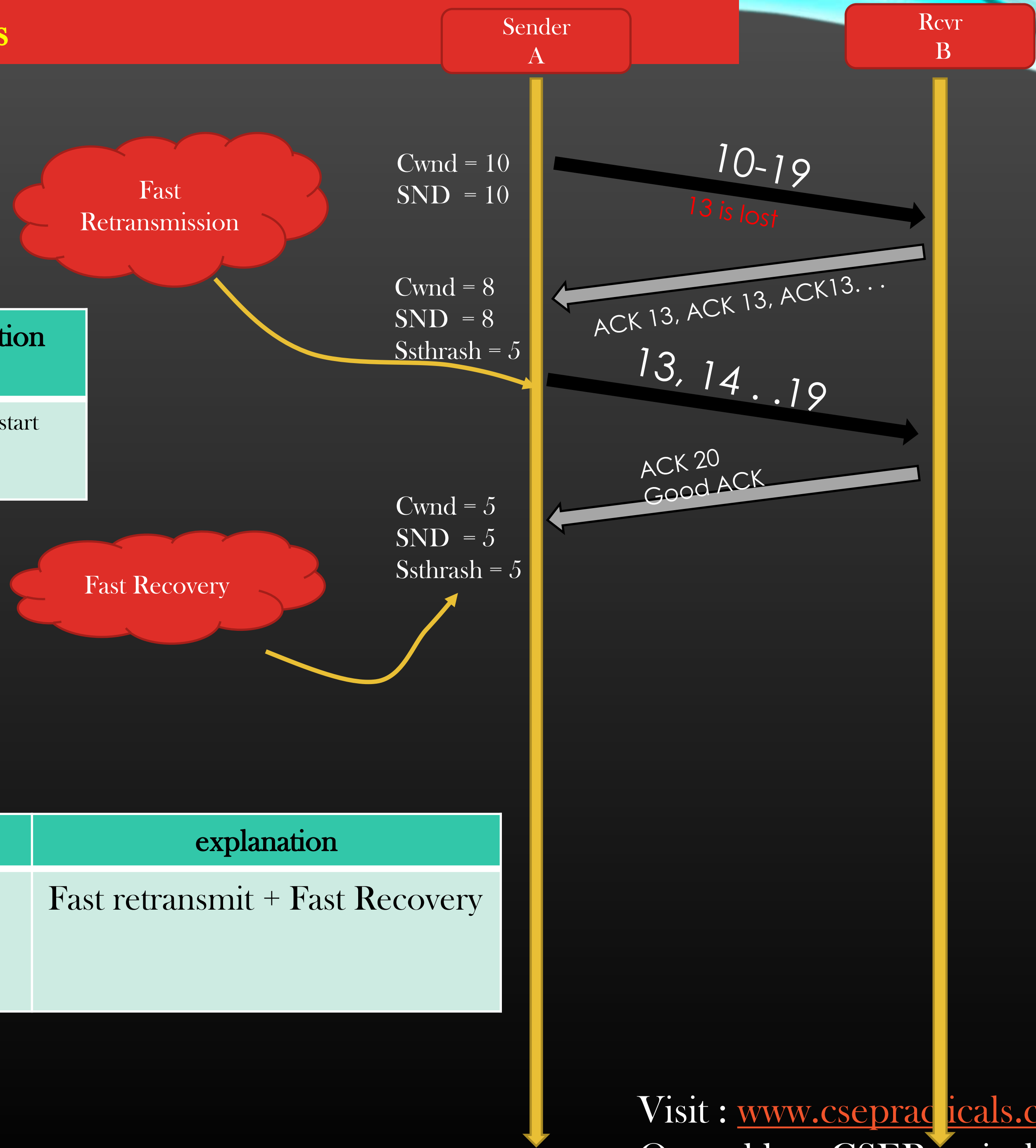
Reason for pkt loss	cwnd	ssthrash	explanation
RTO times out	reset to 1	max(cwnd/2, 2*MSS)	Restart slow-start algorithm

Table 1

☠ -- packet loss due to 3 dupACK

Reason for pkt loss	cwnd	ssthrash	explanation
3 dupACK recvd	= cwnd/2 + 3 When good ACK is recvd then cwnd = ssthrash	= cwnd / 2	Fast retransmit + Fast Recovery

Table 2

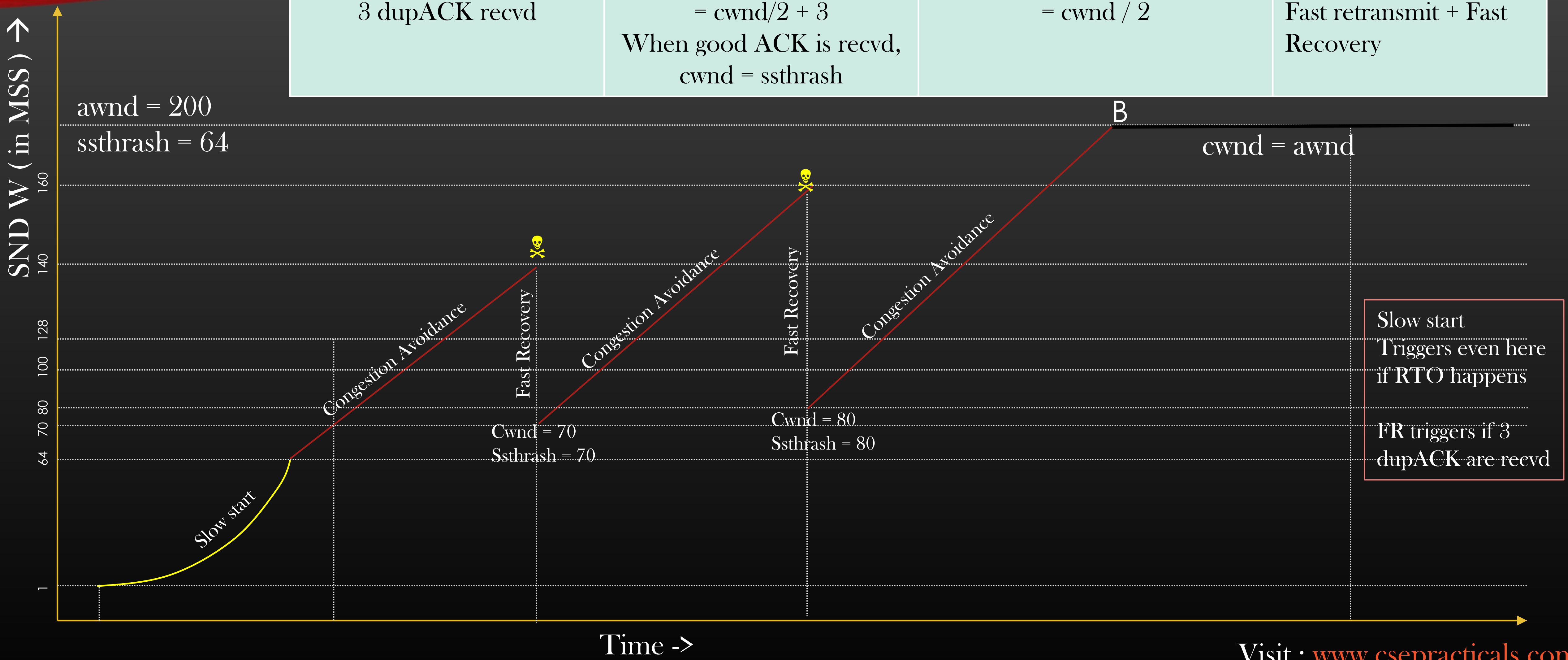


Visit : www.csepracticals.com
Owned by : CSEPracticals

Fast Recovery Graph

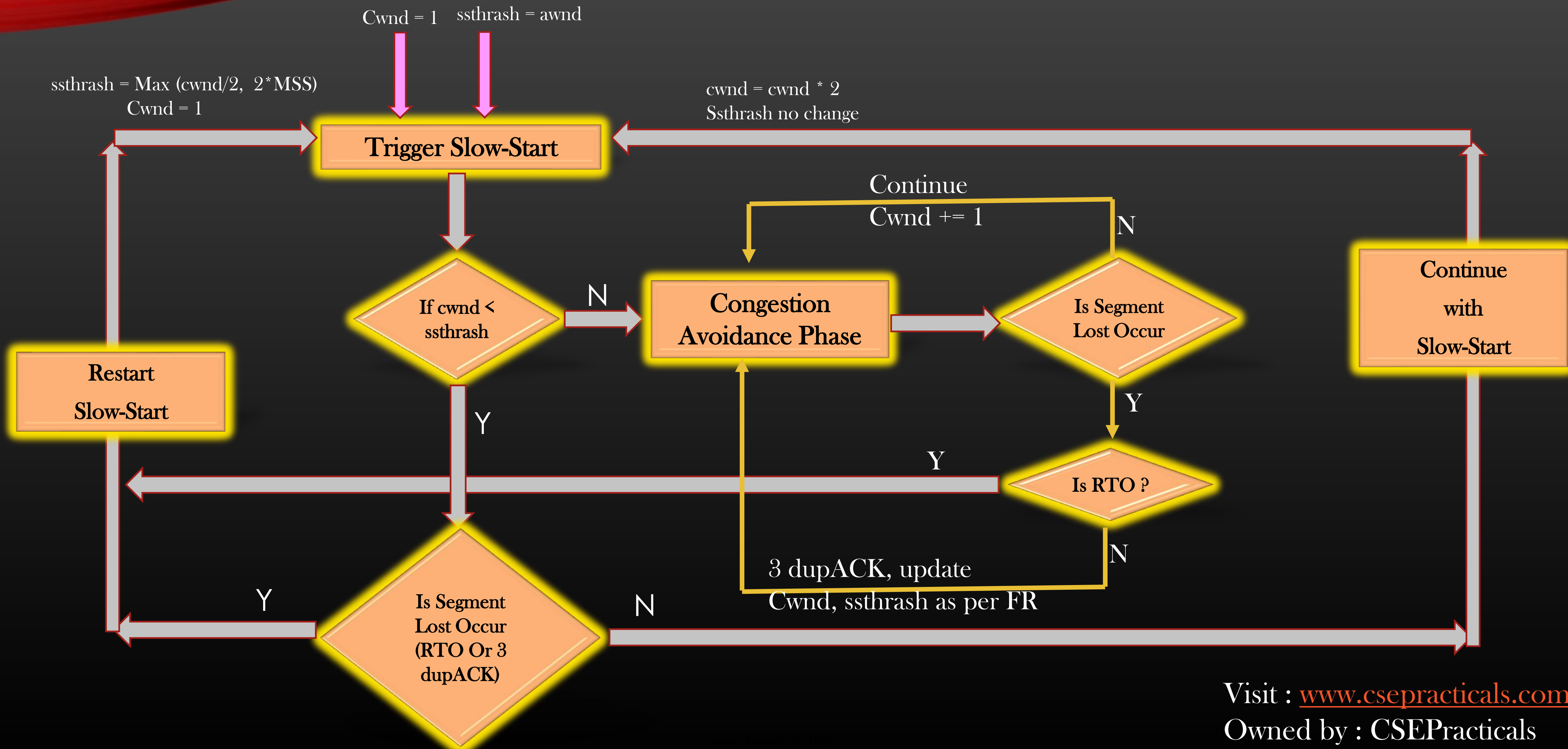
☠ -- packet loss due to 3 dupACK

Reason for pkt loss	cwnd	ssthrash	explanation
3 dupACK recvd	= cwnd/2 + 3 When good ACK is recvd, cwnd = ssthrash	= cwnd / 2	Fast retransmit + Fast Recovery



Visit : www.csepracticals.com
Owned by : CSEPracticals

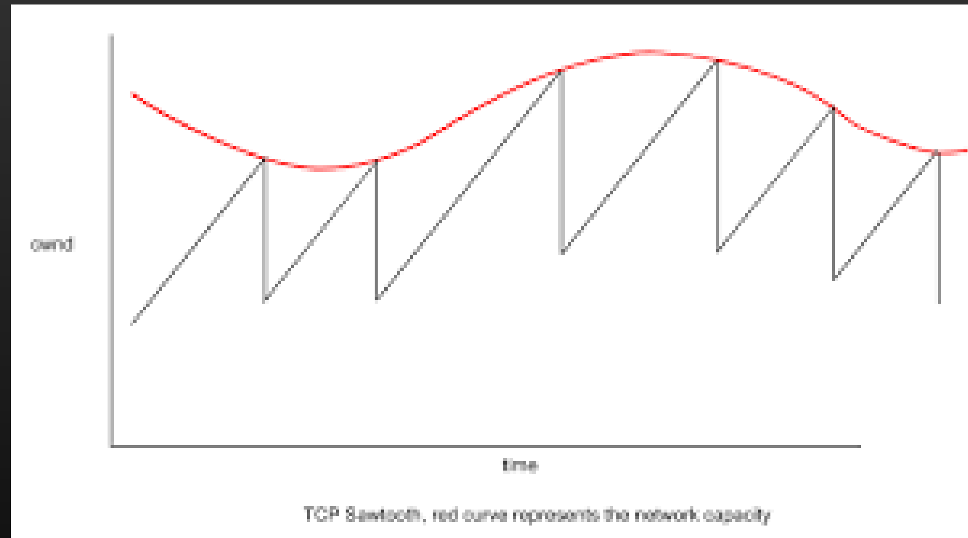
Congestion Control Complete Flowchart



Visit : www.csepracticals.com
Owned by : CSEPracticals

TCP Graph in General

- So, how does a TCP graph showing rate of sending data Vs Time looks like in General in a typical network
- It would look somewhat like a zig-zag graph



TCP Sawtooth, red curve represents the network capacity