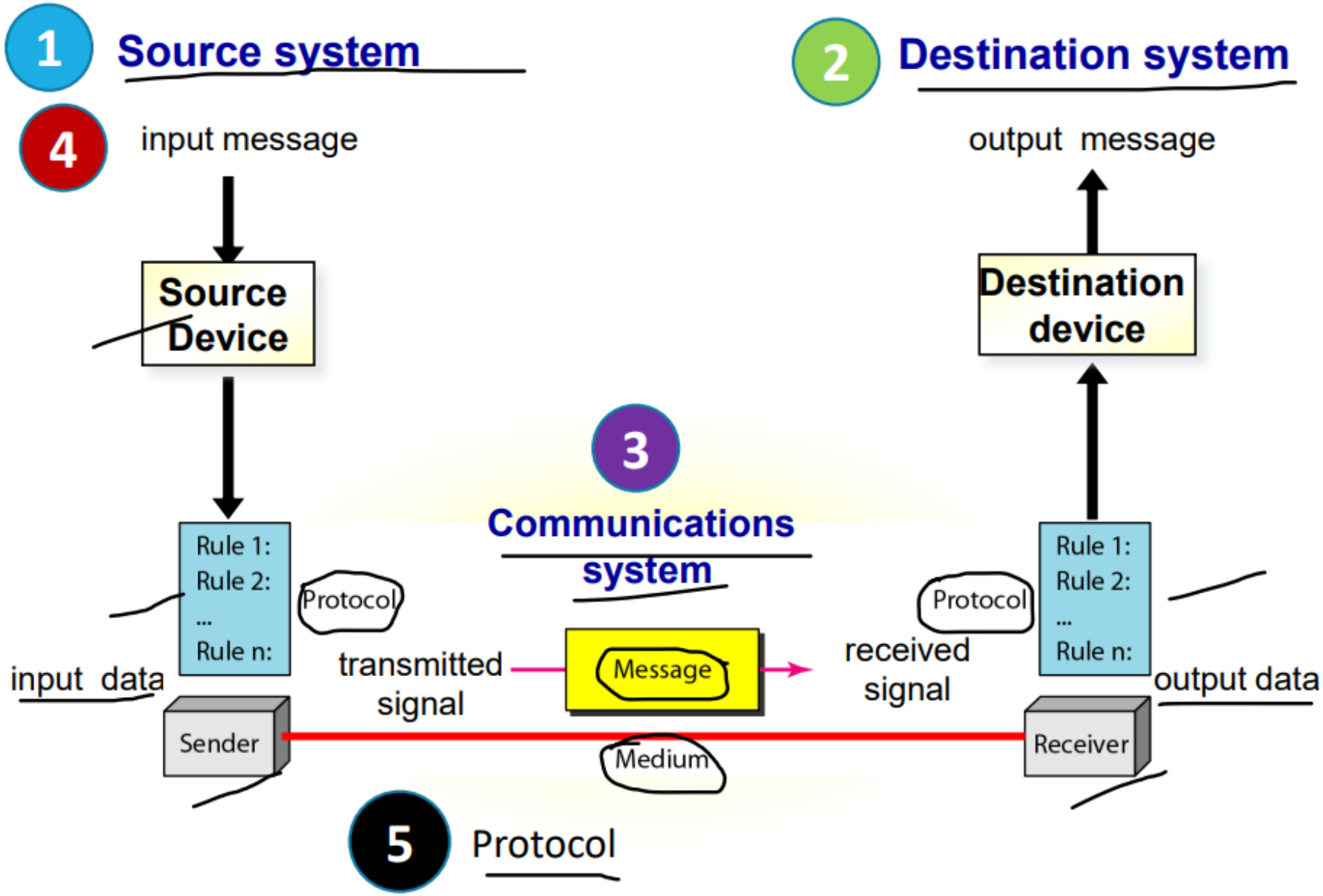
DCN Diagrams

Unit 1

Communication Model



Data Flow

- Direction of data flow

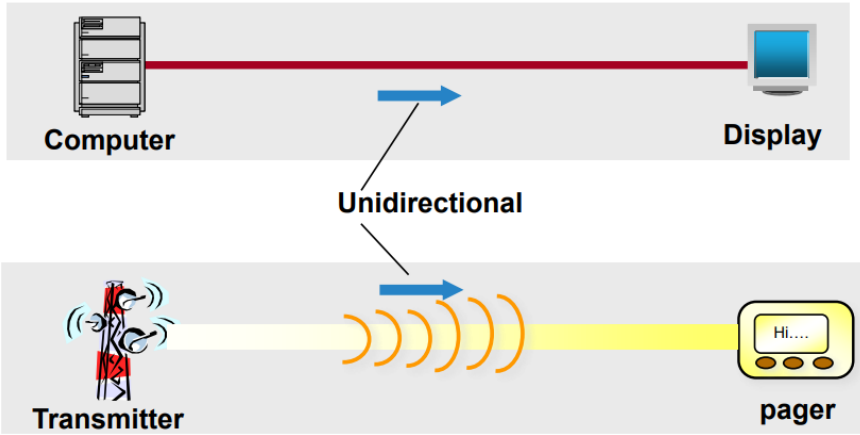
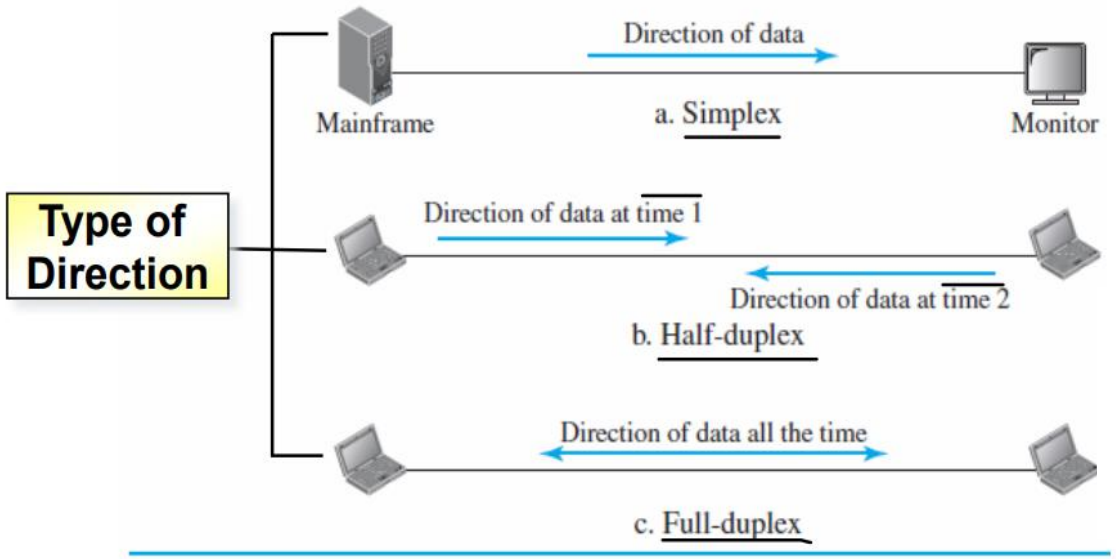
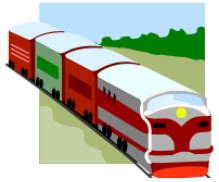
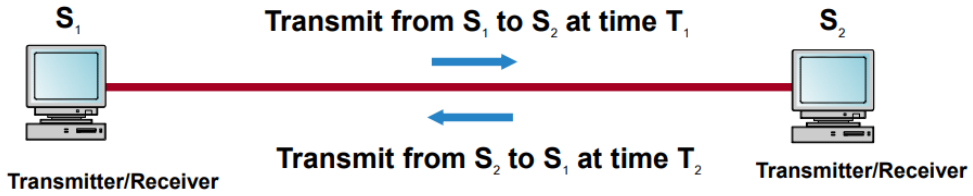
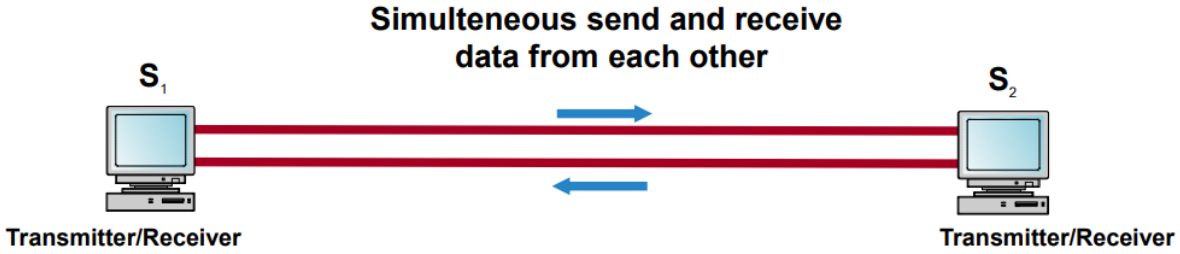


Figure 1.2 *Data flow (simplex, half-duplex, and full-duplex)*



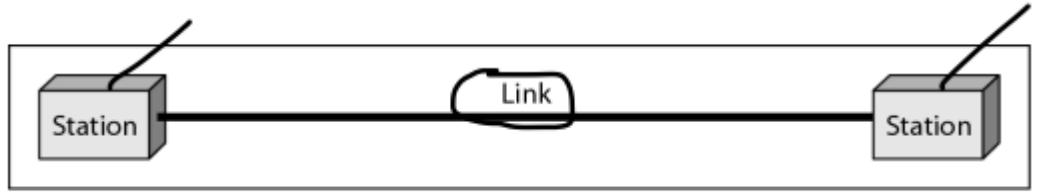




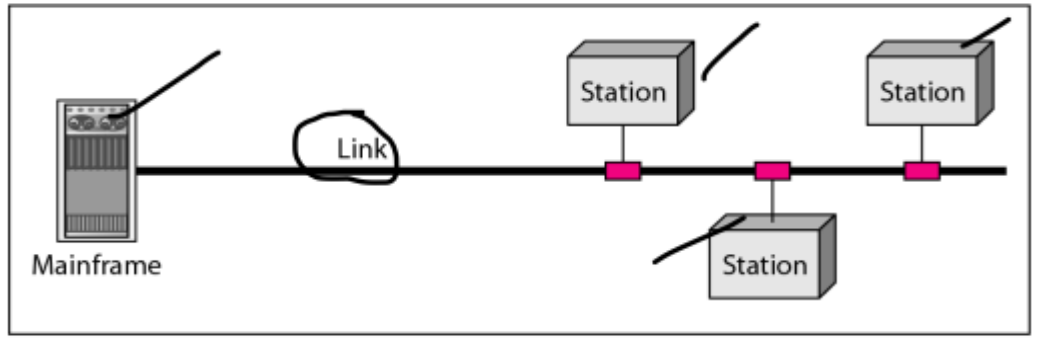


Line configurations

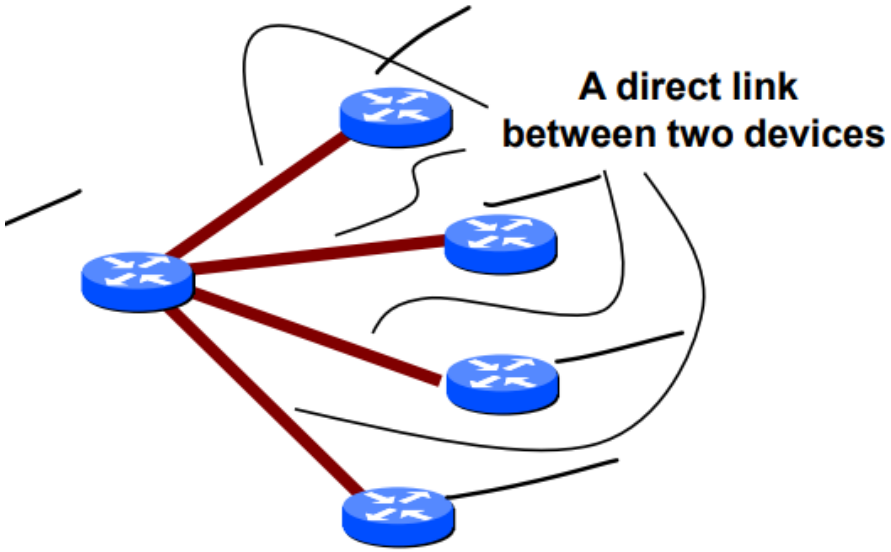
- Point-to-Point
- Multipoint

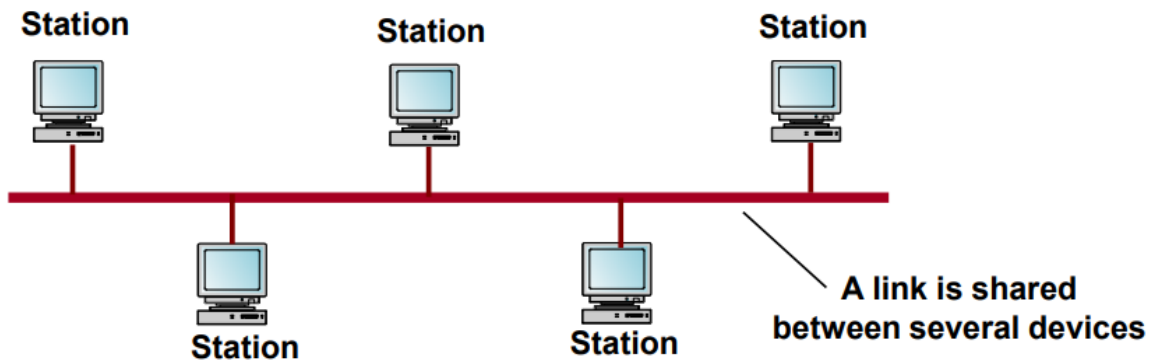


a. Point-to-point



b. Multipoint





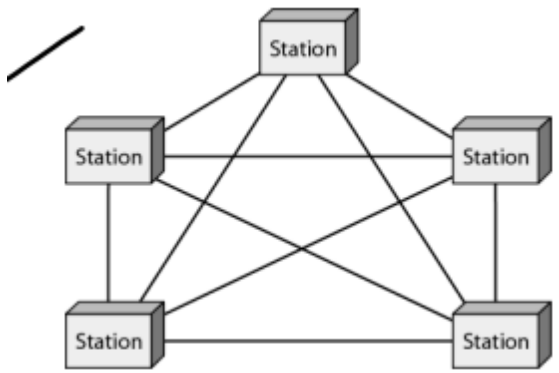
Common Network Topologies

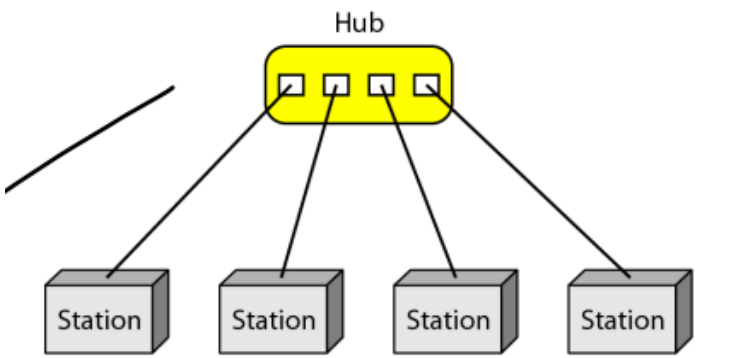
Mesh

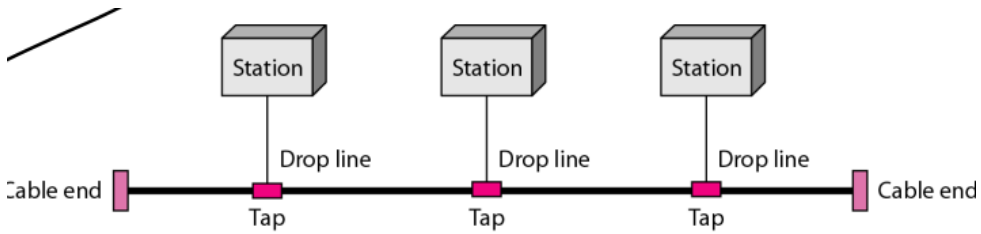
Star

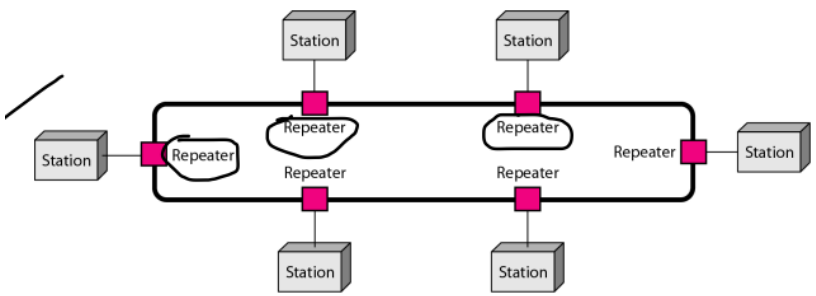
Bus

Ring

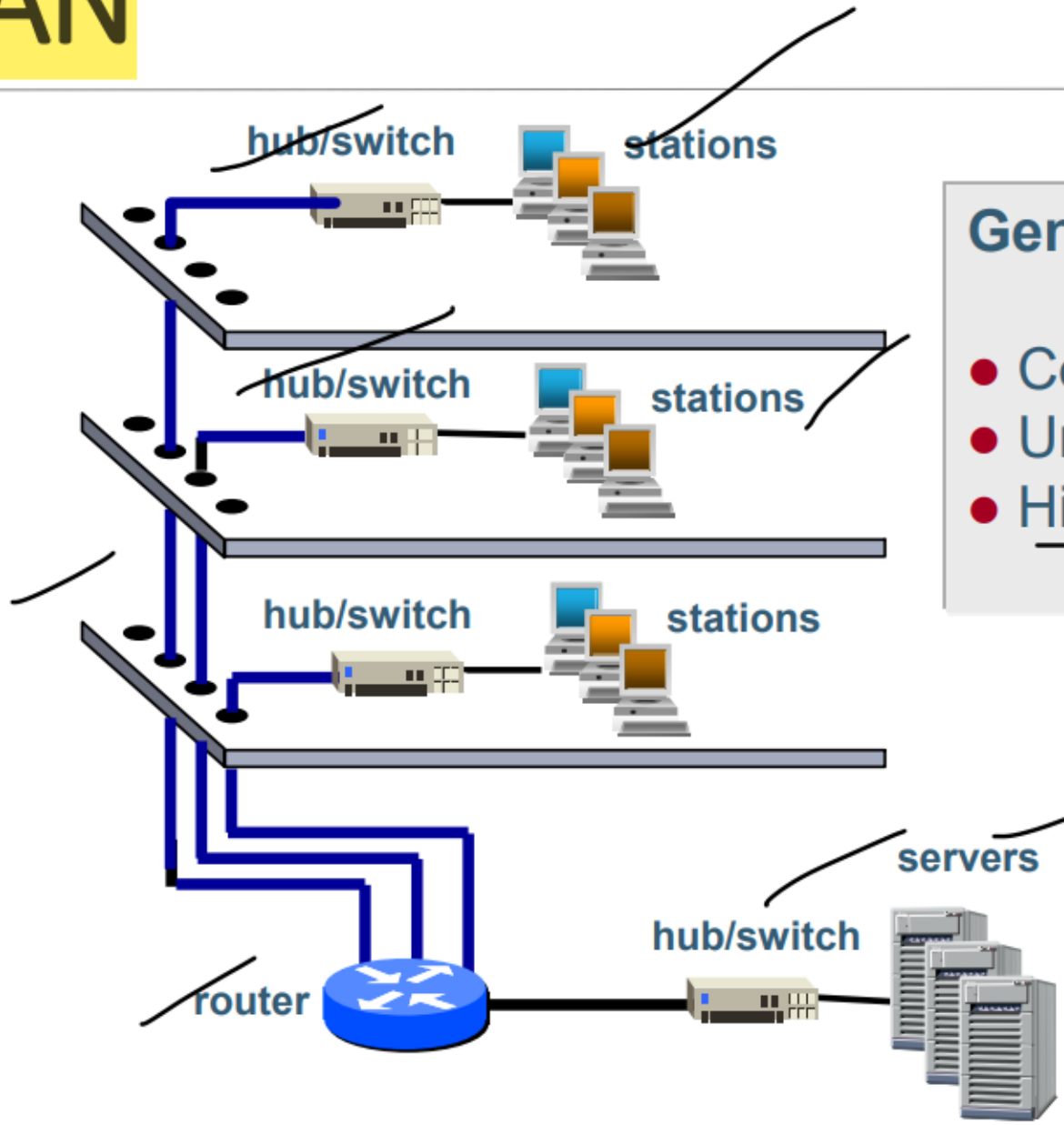






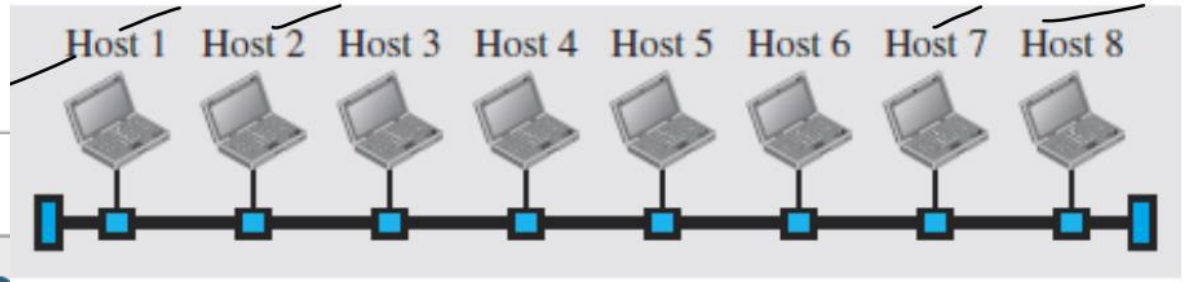


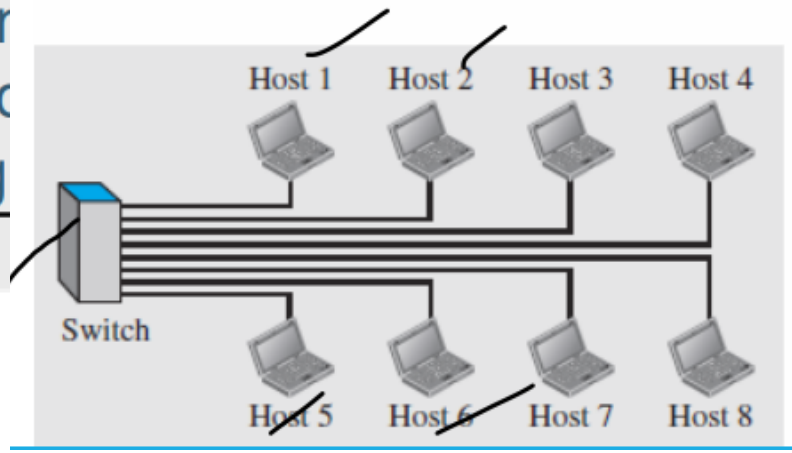
LAN

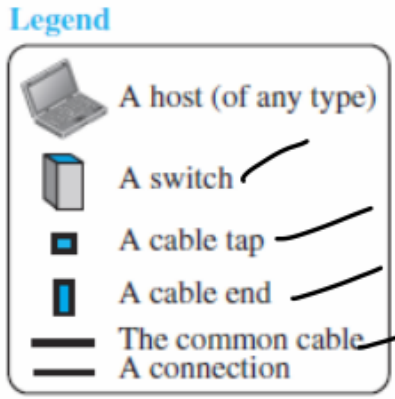


Gene

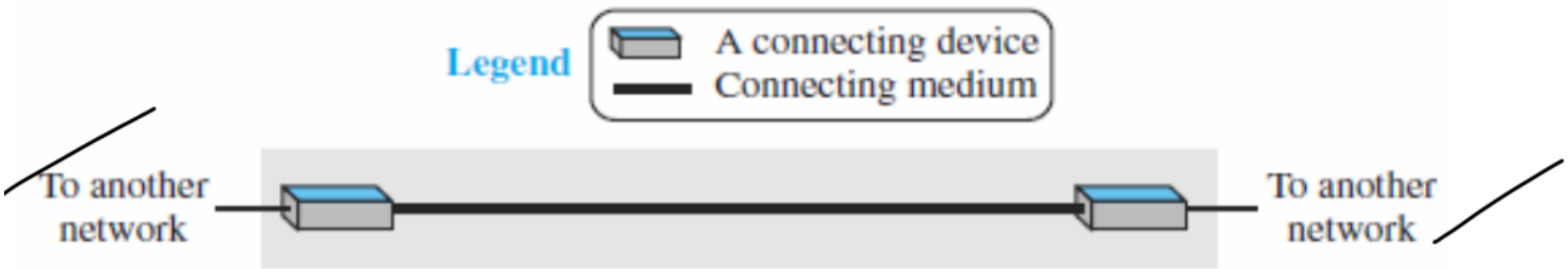
- Cor
- Unc
- Hig



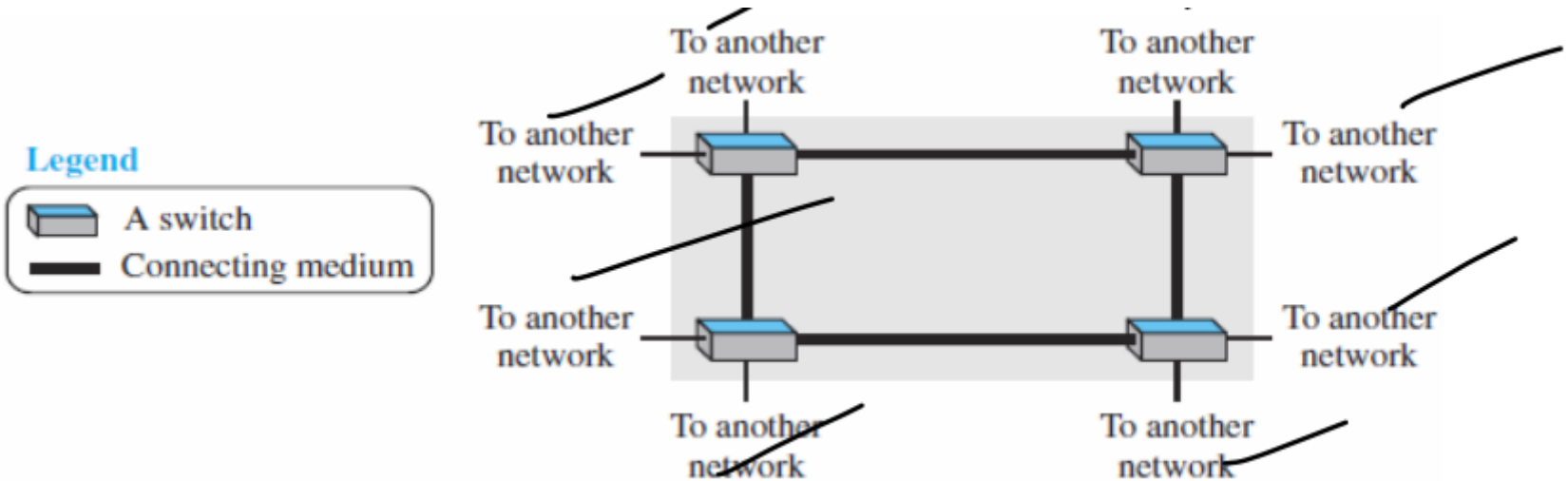


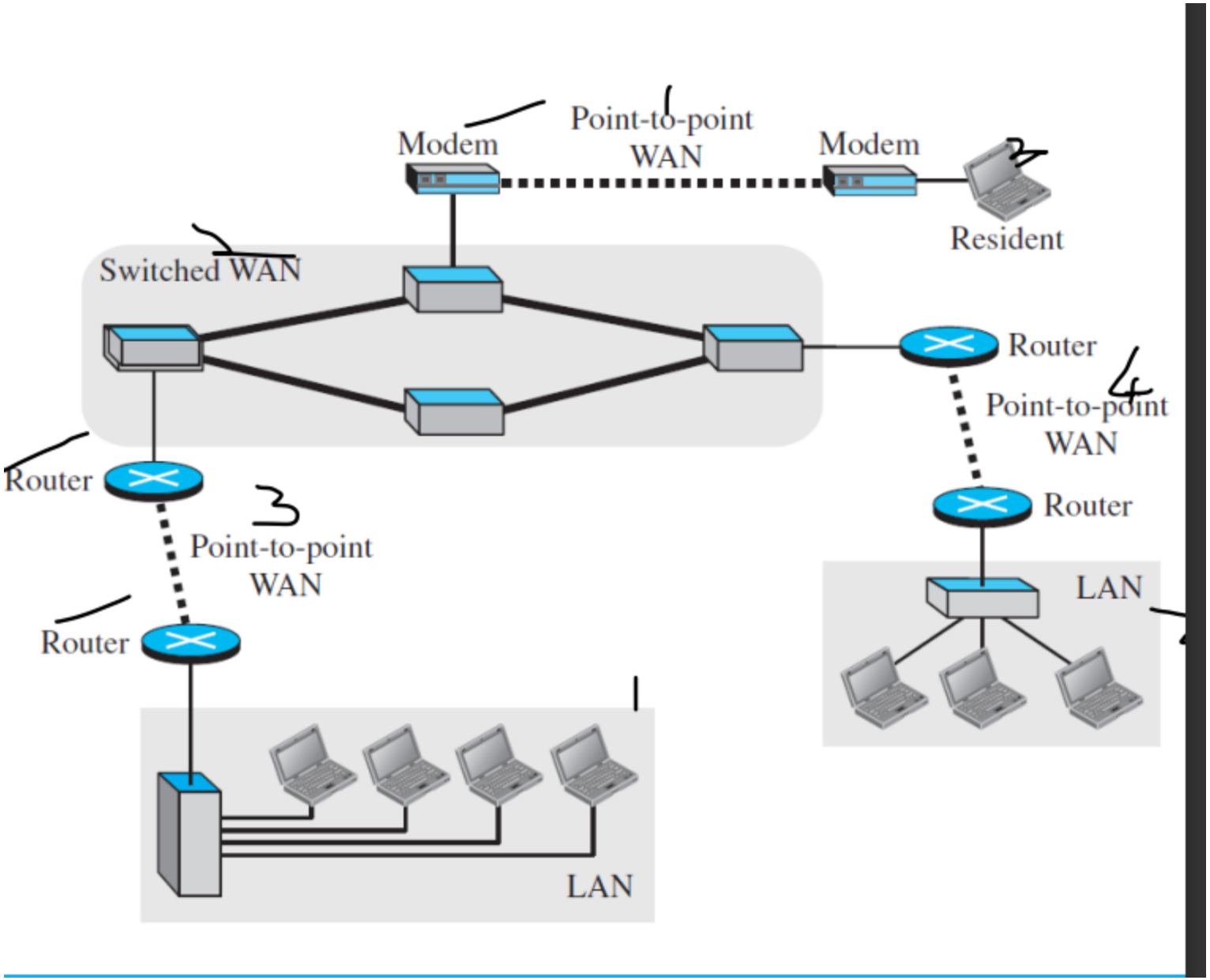


Point-to-Point WAN



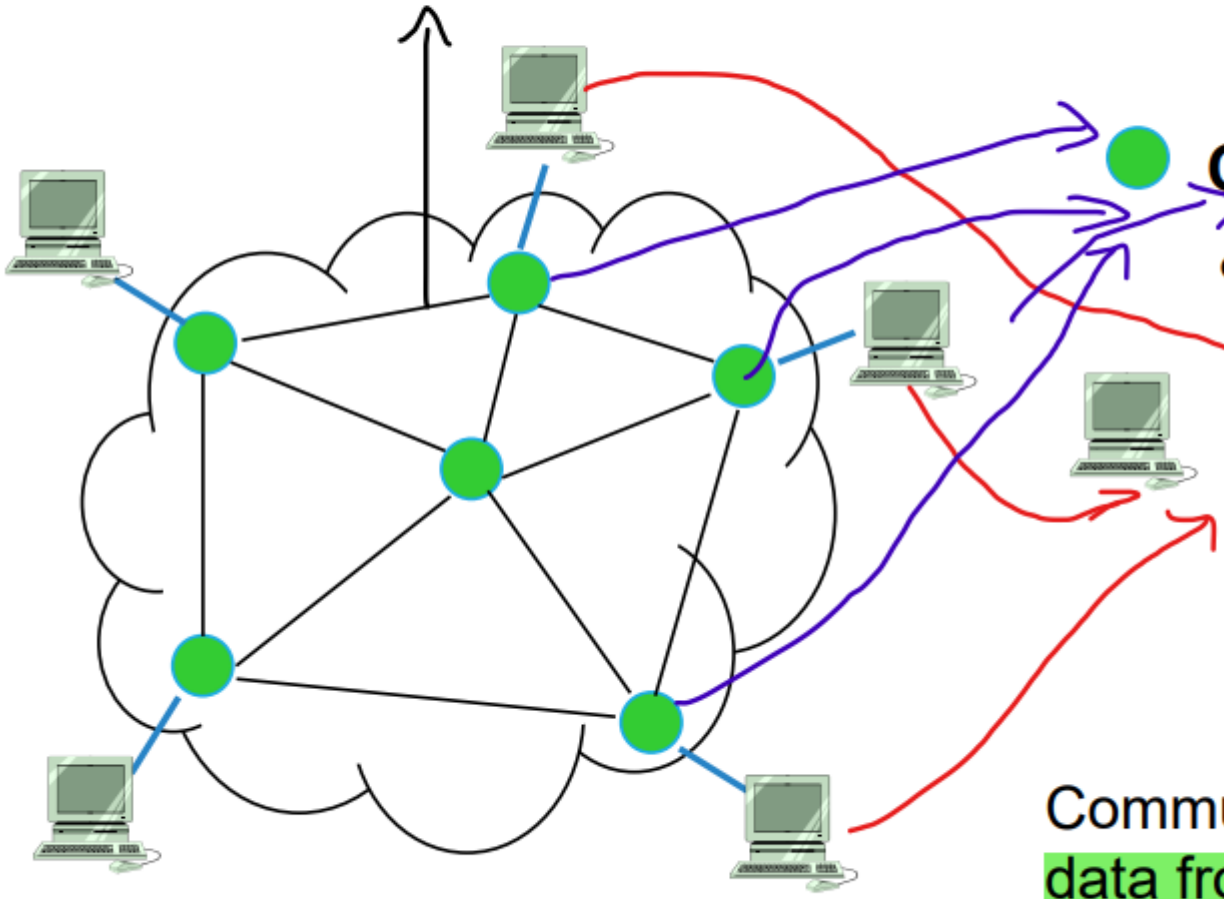
Switched WAN




Modem
Point-to-point
WAN
Modem
Resident
Switched WAN
Router
Point-to-point
WAN
Router
Router
Point-to-point
WAN
Router
LAN
LAN

Switching

Switching/Communication Node Network



Communication Network Node

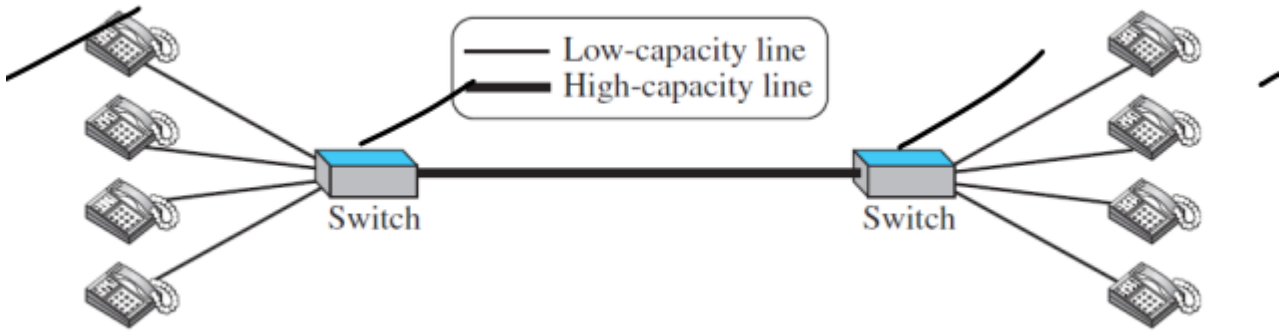
- provide switching facility (routing)

Network Station

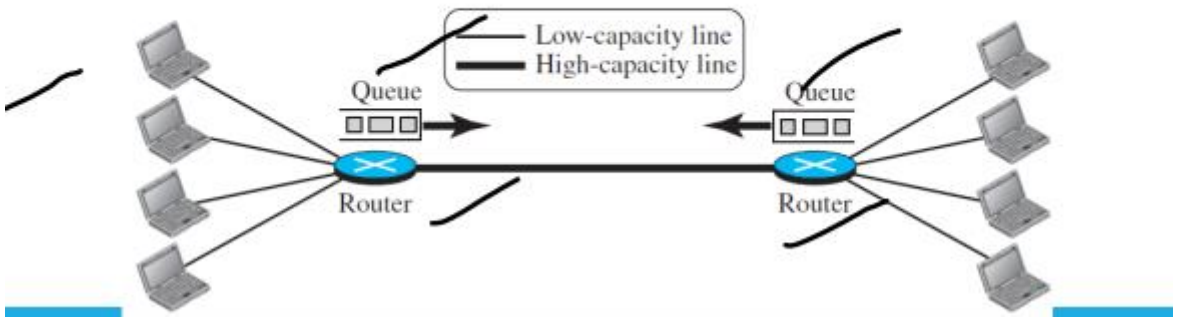
- end node (source &destination)

Communications is achieved by transmitting data from source to destination through a network of switching nodes

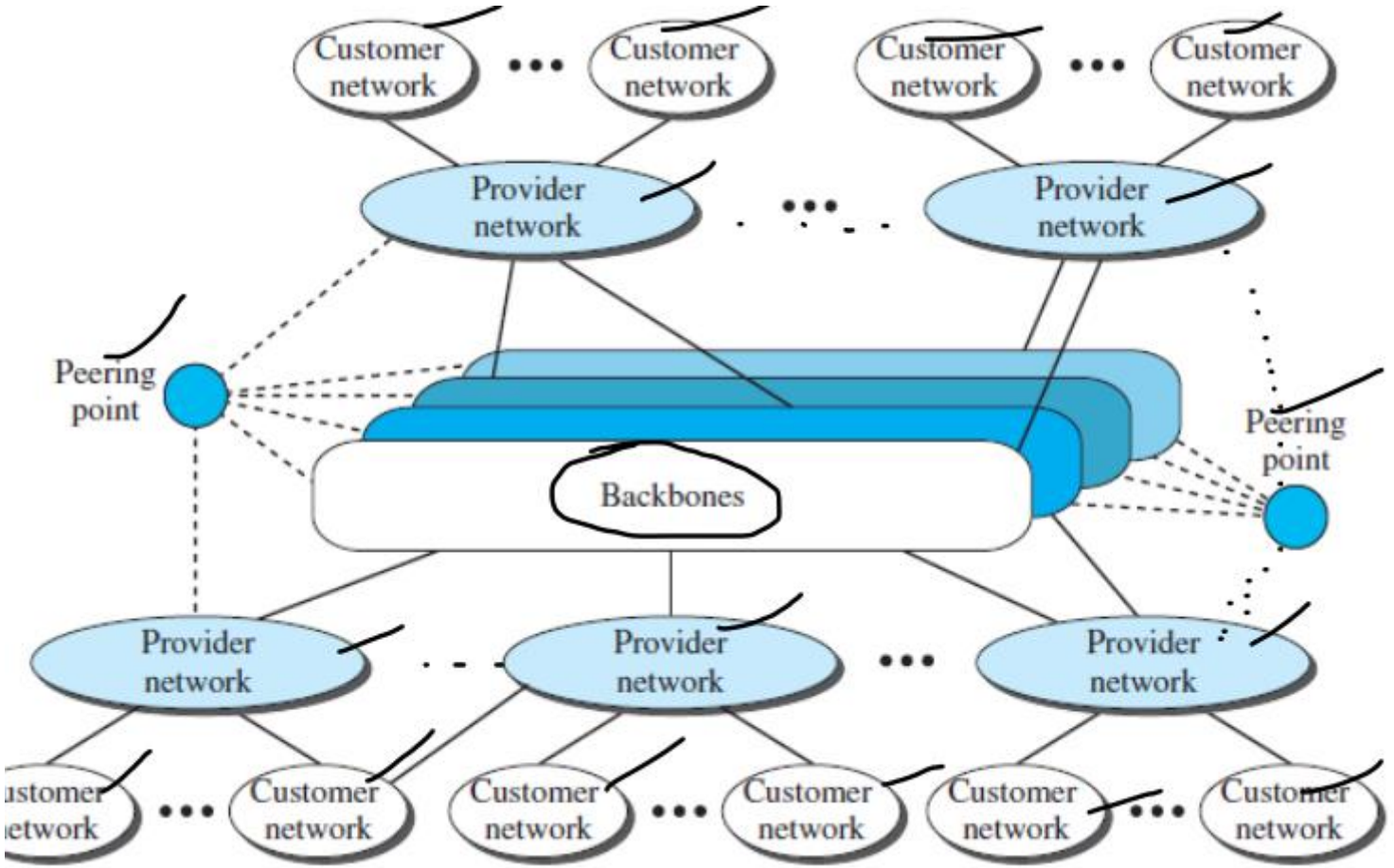
Circuit-Switched Network

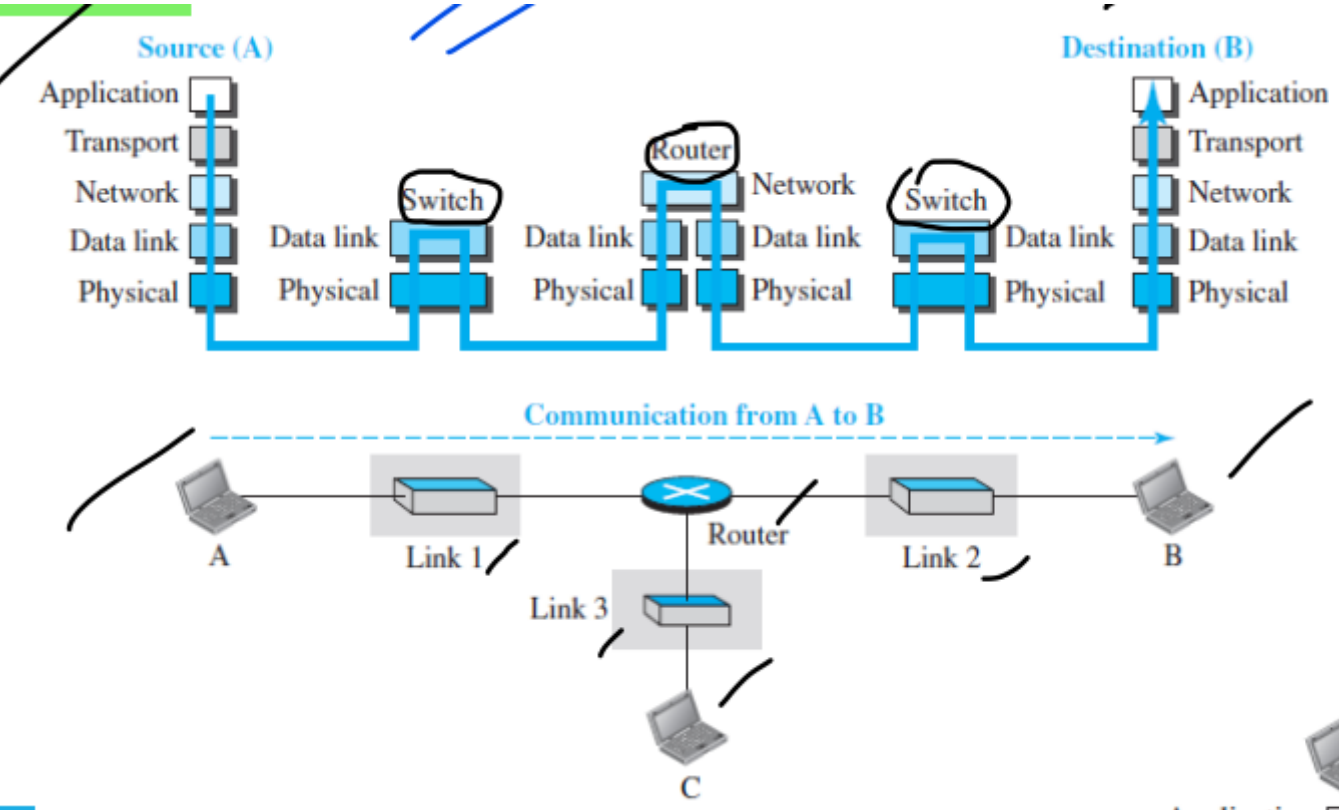


Packet-Switched Network



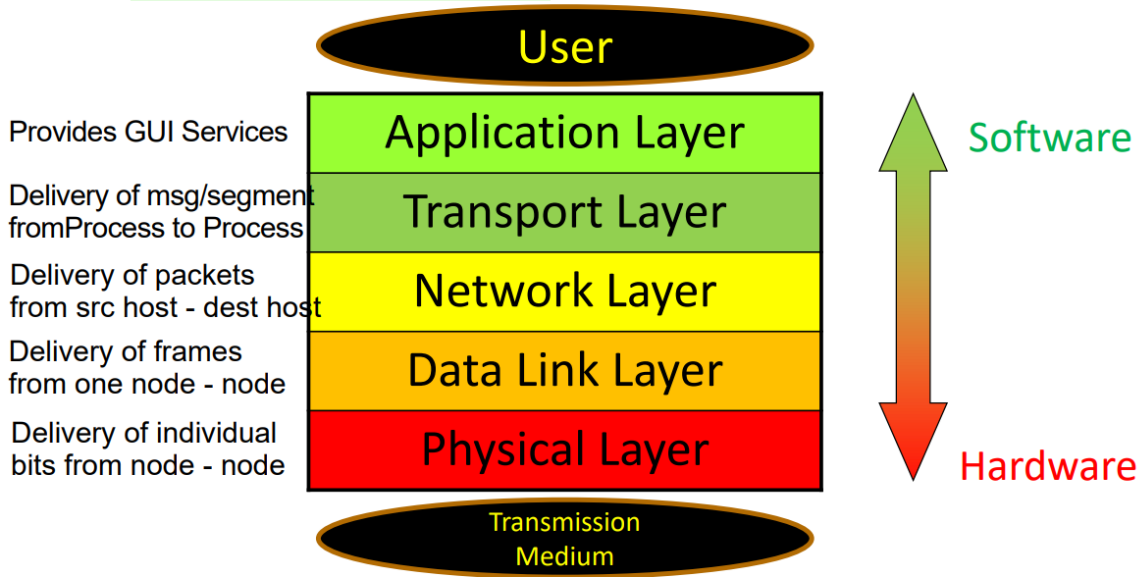
INTERNET

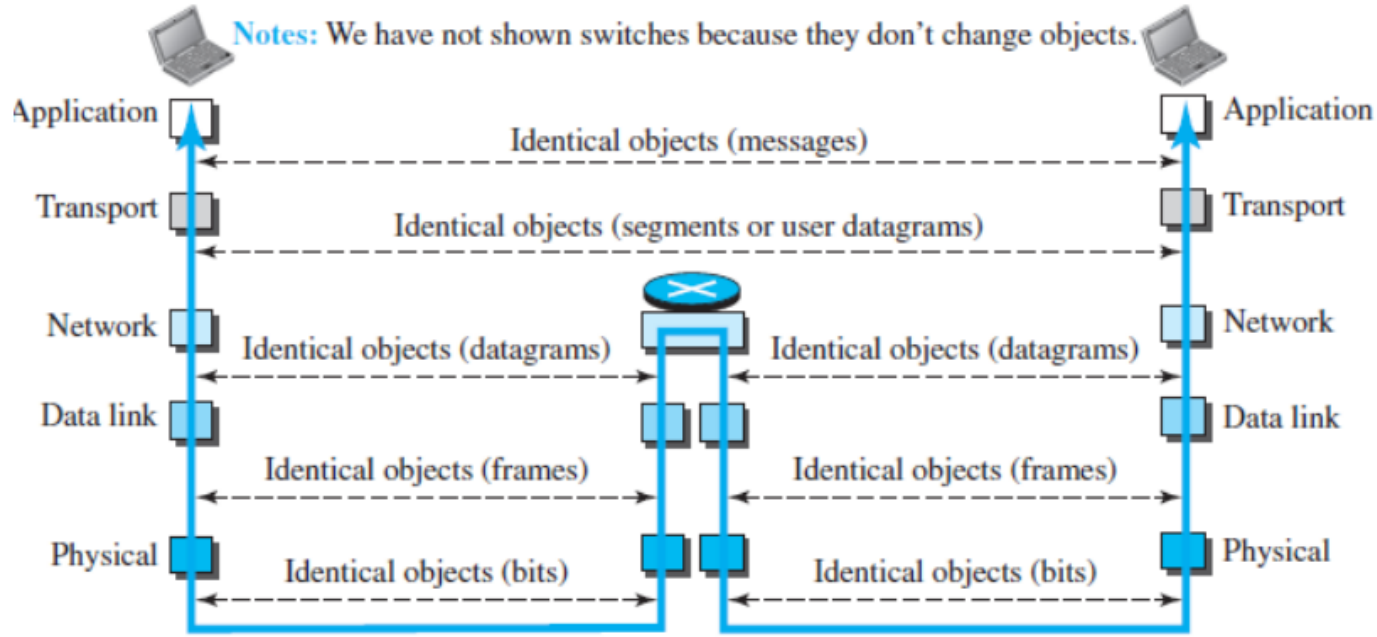

Customer network
Customer network
Customer network
Customer network
Provider network
Provider network
Peering point
Peering point
Backbones
Provider network
Provider network
Provider network
Customer network
Customer network
Customer network
Customer network
Customer network

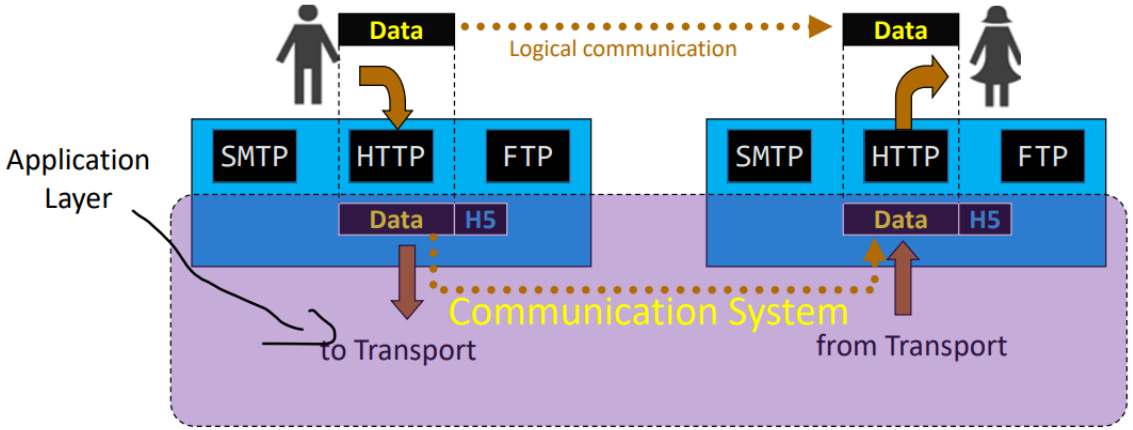


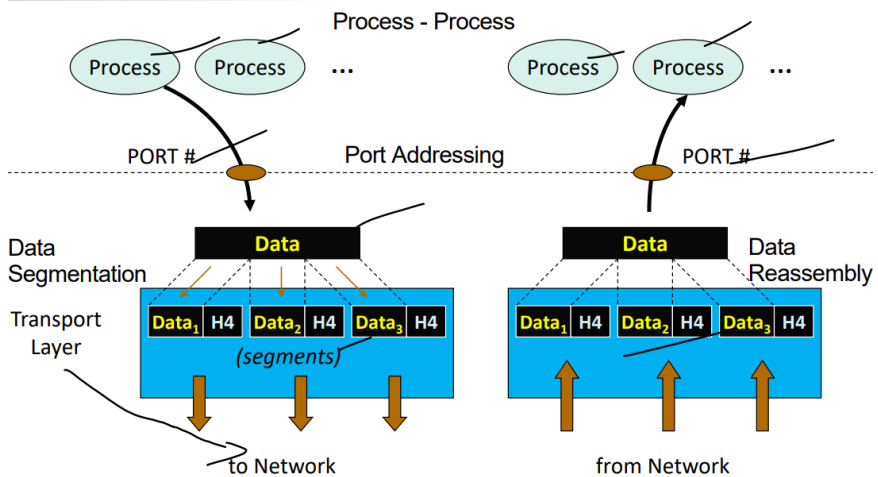
Internet Layer Model

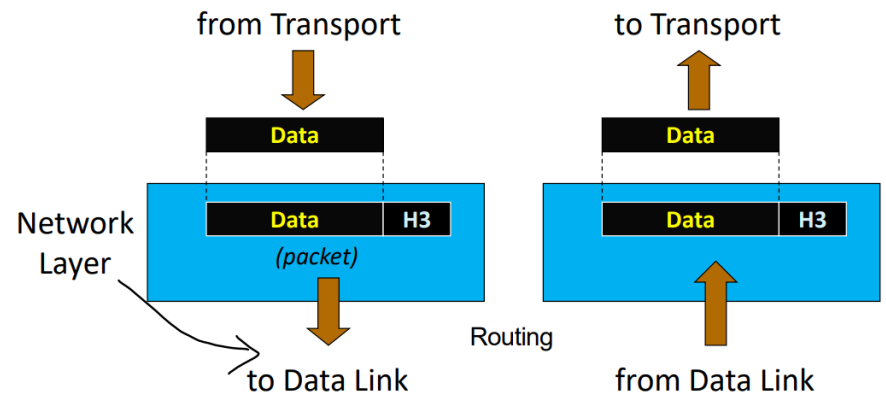
- The Internet Protocol Stack

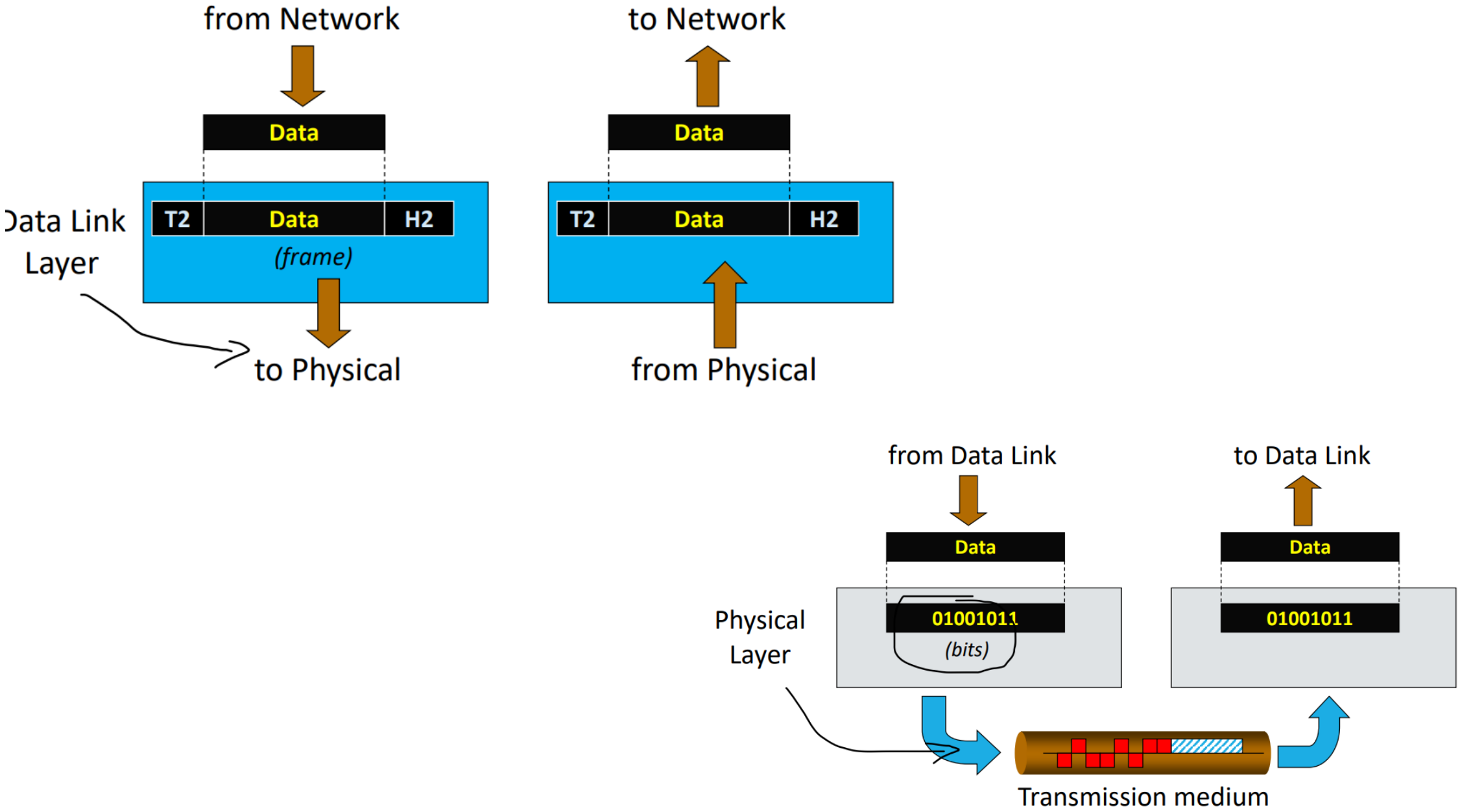


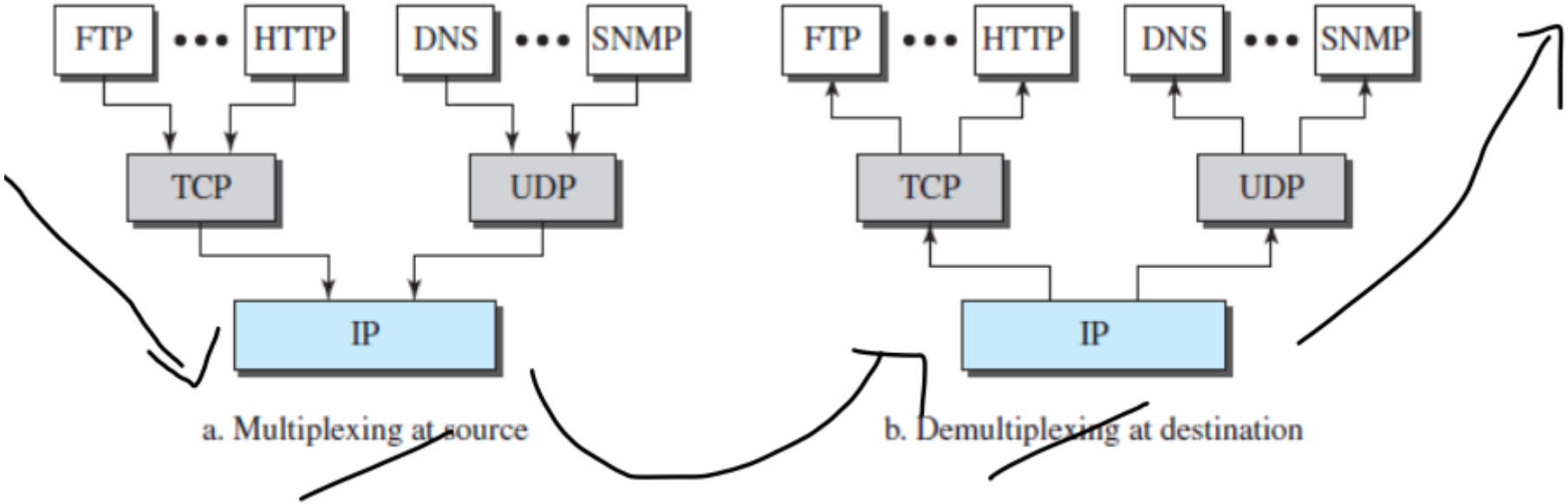


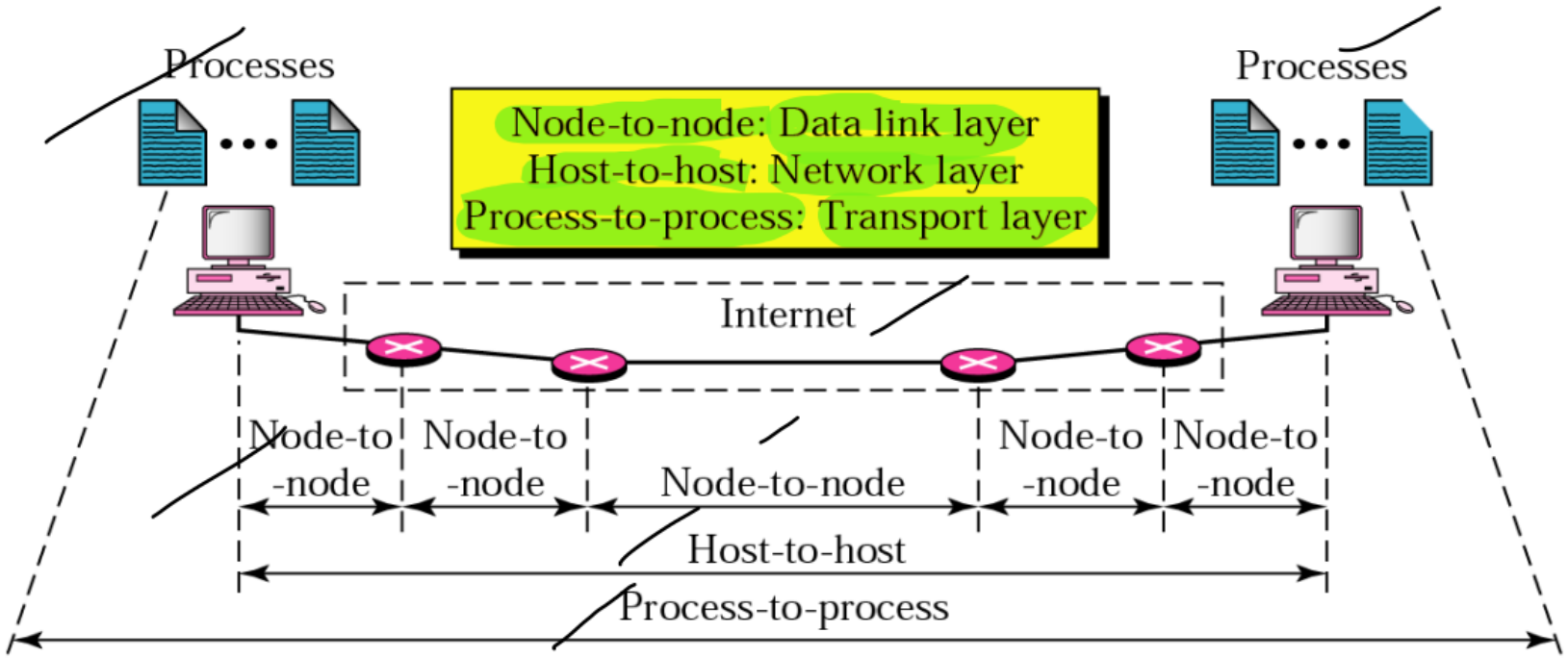

Data
Logical communication
Data
SMTP
HTTP
FTP
SMTP
HTTP
FTP
Application Layer
Data
H5
Data
H5
Communication System
to Transport
from Transport


Process - Process
Process
Process
...
Process
Process
...
PORT #
Port Addressing
PORT #
Data
Data
Data Segmentation
Data Reassembly
Transport Layer
Data1 H4
Data2 H4
Data3 H4
(segments)
Data1 H4
Data2 H4
Data3 H4
to Network
from Network


from Transport
to Transport
Data
Data
Network Layer
Data
H3
(packet)
Data
H3
Routing
to Data Link
from Data Link

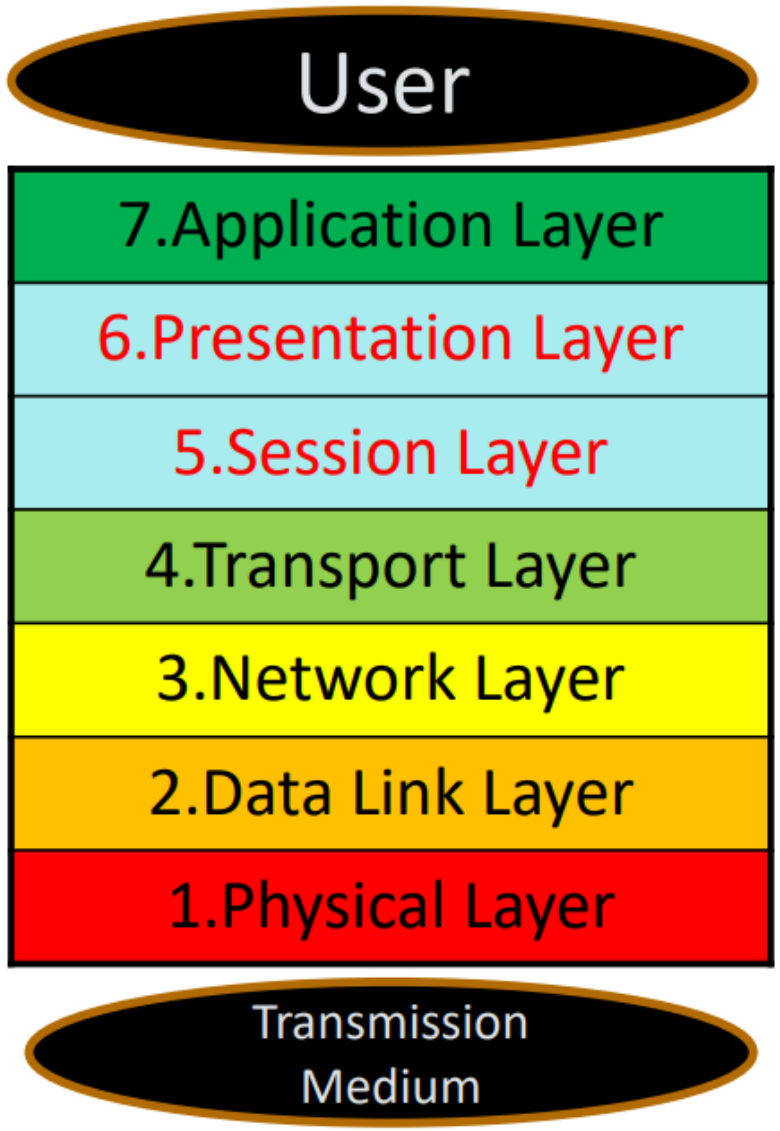

from Network
Data
Data Link Layer
T2
Data
H2
(frame)
to Physical
to Network
Data
T2
Data
H2
from Physical
from Data Link
Data
Physical Layer
01001011
(bits)
to Data Link
Data
01001011
Transmission medium


FTP
HTTP
DNS
SNMP
TCP
UDP
IP
FTP
HTTP
DNS
SNMP
TCP
UDP
IP
a. Multiplexing at source
b. Demultiplexing at destination


Processes
Node-to-node: Data link layer
Host-to-host: Network layer
Process-to-process: Transport layer
Processes
Internet
Node-to
-node
Node-to
-node
Node-to-node
Node-to
-node
Node-to
-node
Host-to-host
Process-to-process

OSI Model

Open Systems Interconnections



- OSI – ***O**pen **S**ystems **I**nterconnection*
- Developed by the ***I**nternational **S**tandards **O**rganizations* (ISO)
- Two additional layers
  - Presentation layer
  - Session layer

HTTP connections

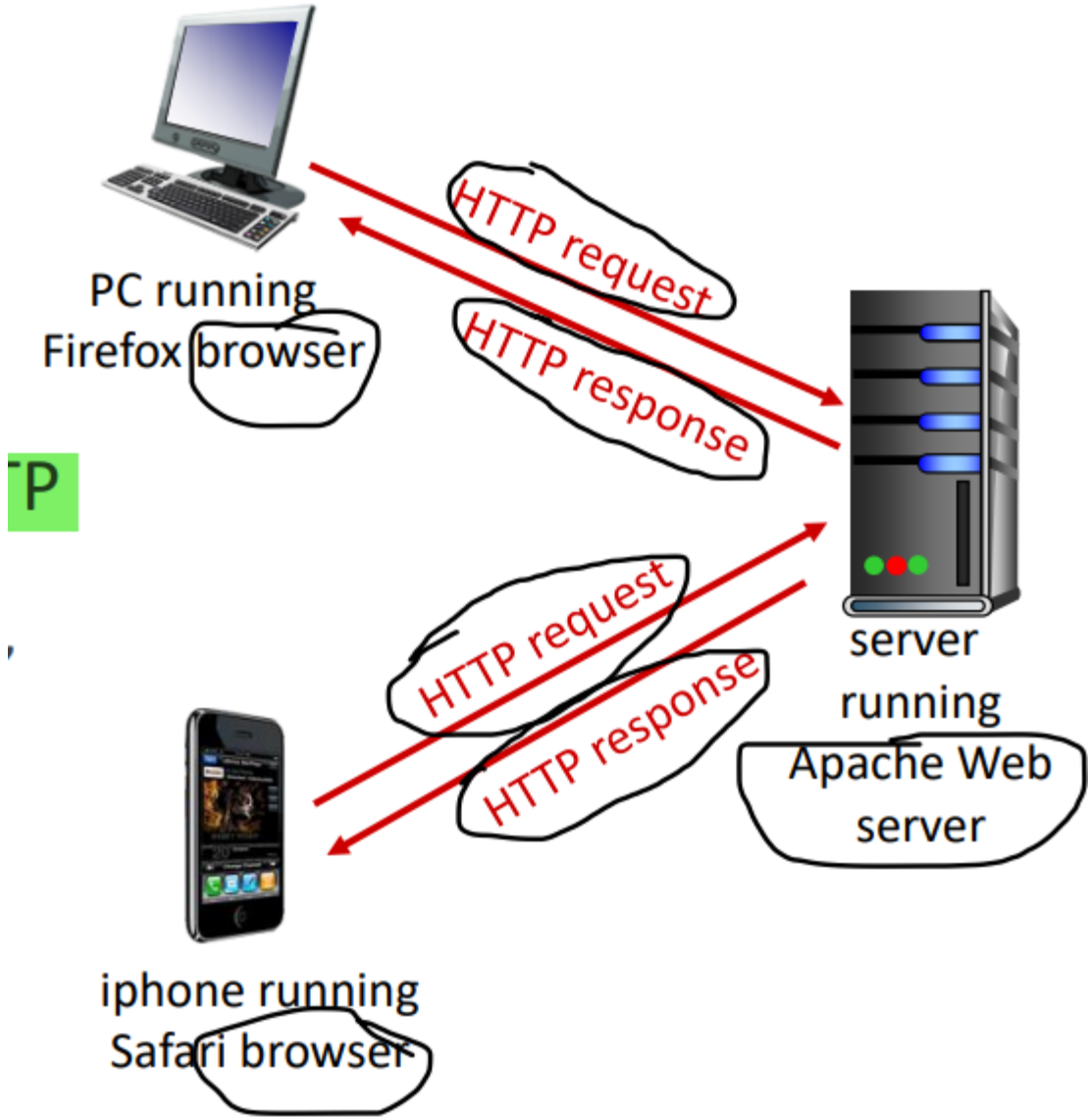
Non-persistent HTTP

1. TCP connection opened
2. at most one object sent over TCP connection
3. TCP connection closed

downloading multiple objects required multiple connections

Persistent HTTP

- TCP connection opened to a server
- multiple objects can be sent over *single* TCP connection between client, and that server
- TCP connection closed



Non-persistent HTTP

suppose user enters URL:
`www.someSchool.edu/someDepartment/home.index`

(contains text, references to 10 jpeg images)

1a. HTTP client initiates TCP connection to HTTP server (process) at www.someSchool.edu on port 80

1b. HTTP server at host www.someSchool.edu waiting for TCP connection at port 80. "accepts" connection, notifying client

time

2. HTTP client sends HTTP *request message* (containing URL) into TCP connection socket. Message indicates that client wants object someDepartment/home.index

3. HTTP server receives request message, forms *response message* containing requested object, and sends message into its socket

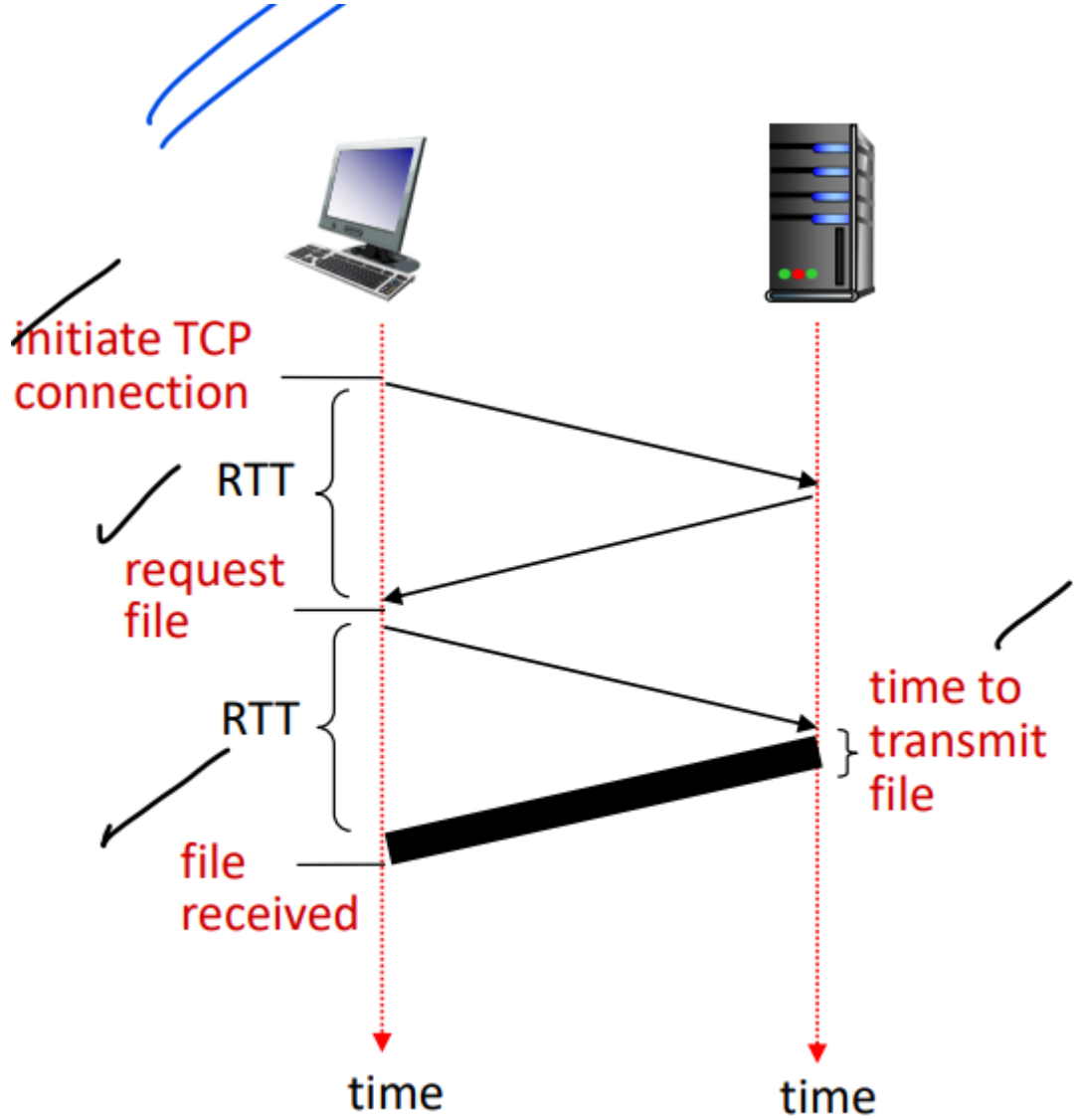
time

4. HTTP server closes TCP connection.

5. HTTP client receives response message containing html file, displays html. Parsing html file, finds 10 referenced jpeg objects

time

6. Steps 1-5 repeated for each of 10 jpeg objects



- HTTP request message:
  - ASCII (human-readable format)

request line (GET, POST, HEAD commands)

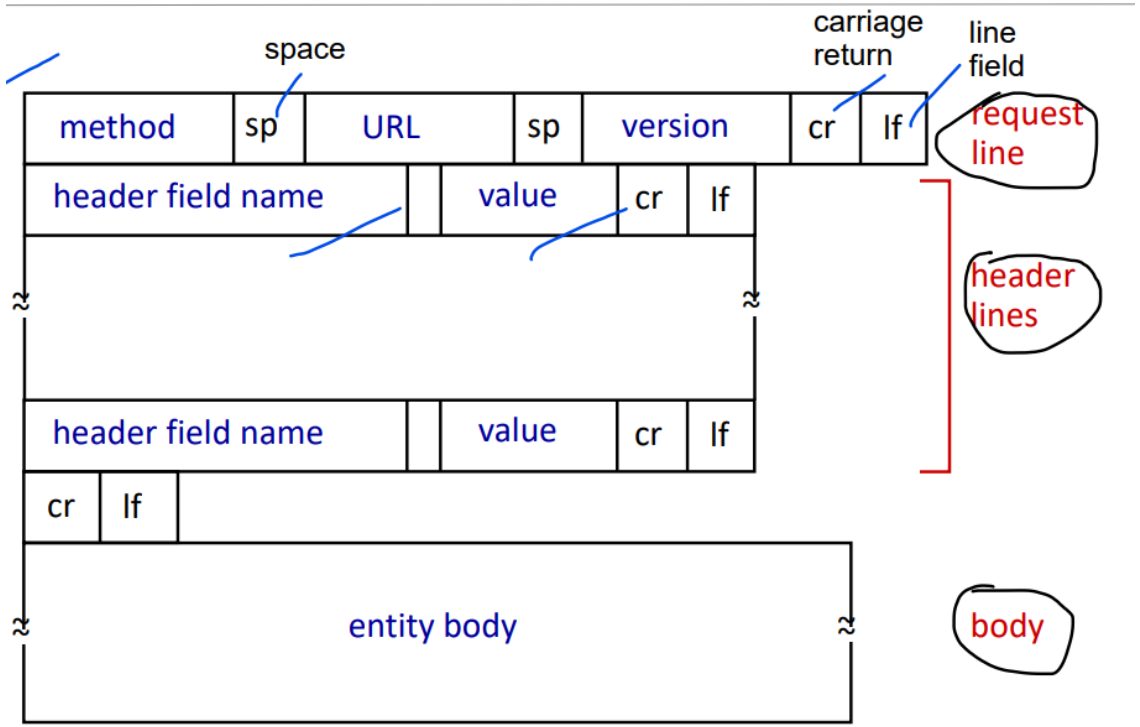
carriage return character

line-feed character

header lines

carriage return, line feed at start of line indicates end of header lines

```
GET /index.html HTTP/1.1\r\n
Host: www-net.cs.umass.edu\r\n
User-Agent: Firefox/3.6.10\r\n
Accept: text/html,application/xhtml+xml\r\n
Accept-Language: en-us,en;q=0.5\r\n
Accept-Encoding: gzip,deflate\r\n
Accept-Charset: ISO-8859-1,utf-8;q=0.7\r\n
Keep-Alive: 115\r\n
Connection: keep-alive\r\n
\r\n
```



HTTP response message

status line (protocol status code status phrase)

Protocol

Status Code

Status Phrase

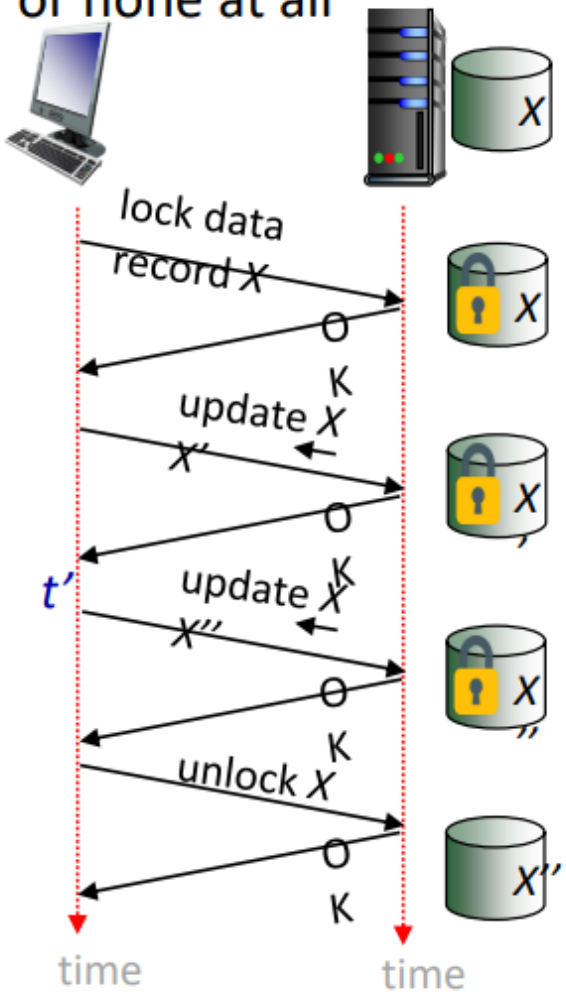
header lines

data, e.g., requested HTML file

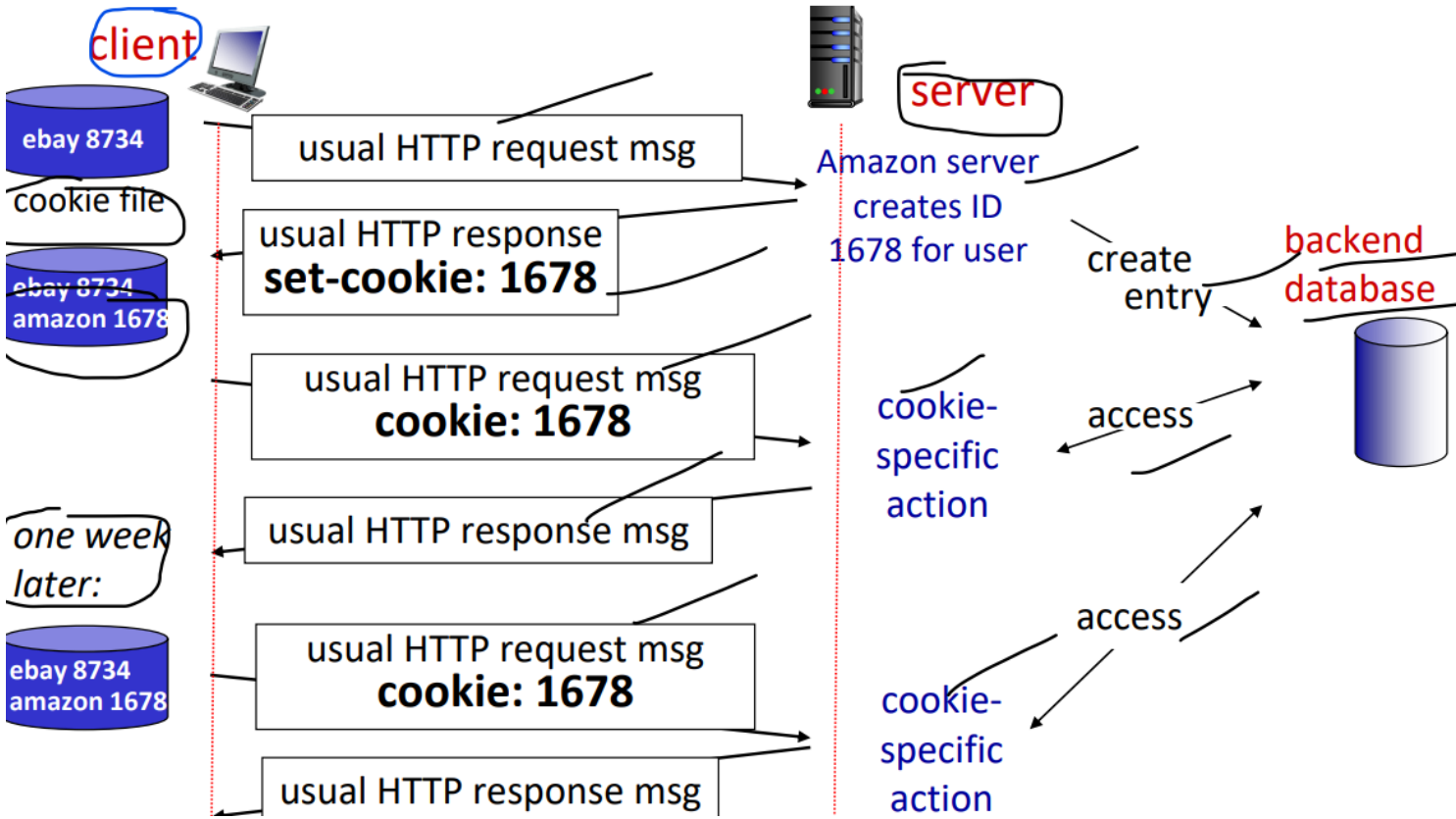
```
HTTP/1.1 200 OK\r\n
Date: Sun, 26 Sep 2010 20:09:20 GMT\r\n
Server: Apache/2.0.52 (CentOS)\r\n
Last-Modified: Tue, 30 Oct 2007 17:00:02
   GMT\r\n
ETag: "17dc6-a5c-bf716880"\r\n
Accept-Ranges: bytes\r\n
Content-Length: 2652\r\n
Keep-Alive: timeout=10, max=100\r\n
Connection: Keep-Alive\r\n
Content-Type: text/html; charset=ISO-8859-
   1\r\n
\r\n
data data data data data ...
```

tate: cookies

a stateful protocol: client makes two changes to X, or none at all



Maintaining user/server state: cookies



Conditional GET

Don't Get Object (
object is not modif

cli

Goal: don't send object if cache has up-to-date cached version

- no object transmission delay (or use of network resources)

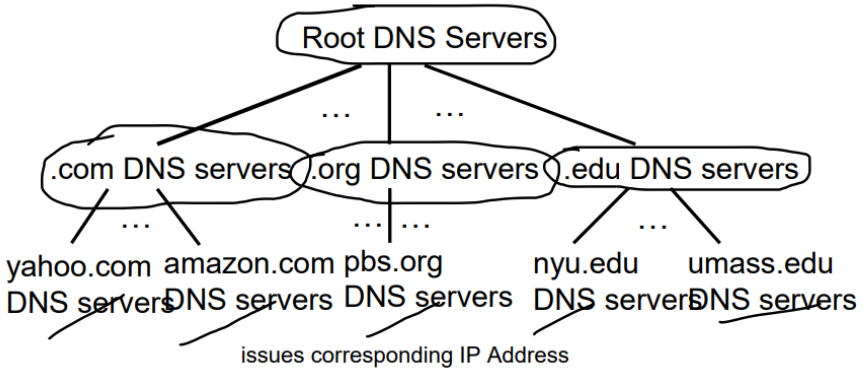
- *client:* specify date of cached copy in HTTP request

  If-modified-since: <date>

- *server:* response contains no object if cached copy is up-to-date:

  HTTP/1.0 304 Not Modified

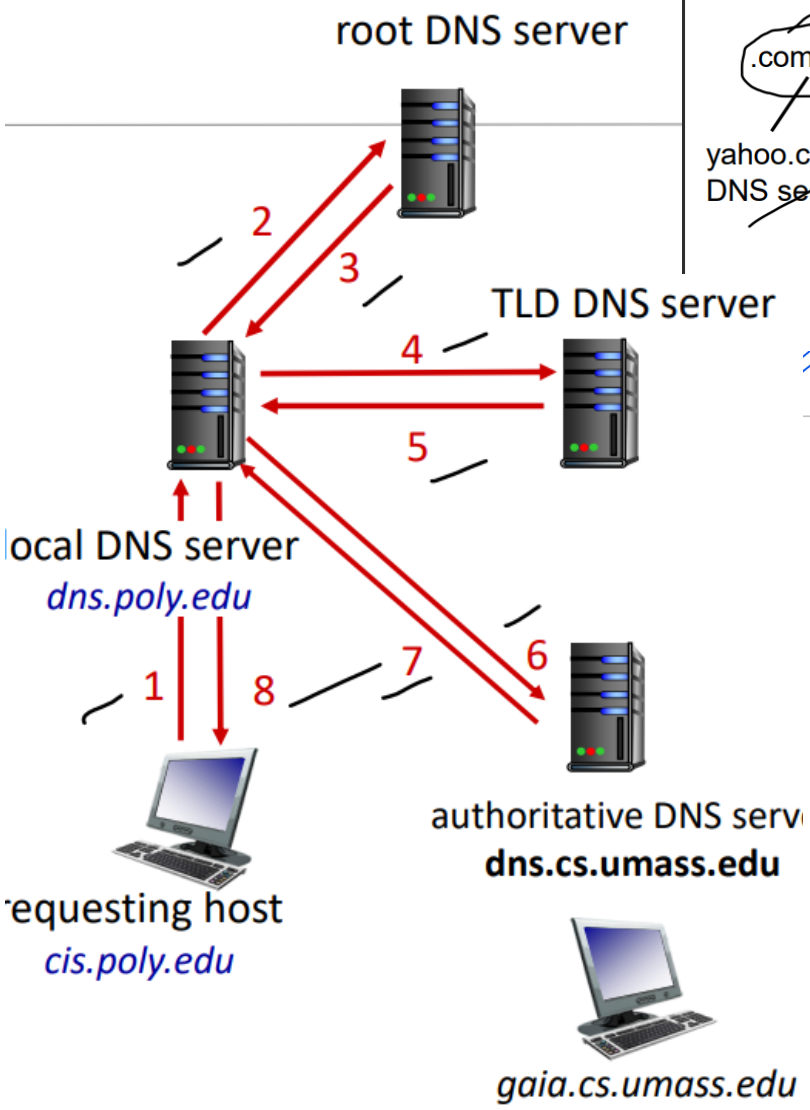
DNS: a distributed, hierarchical database



Root

Top Level Domain

Authoritative



DNS records

DNS: distributed db storing resource records (RR)

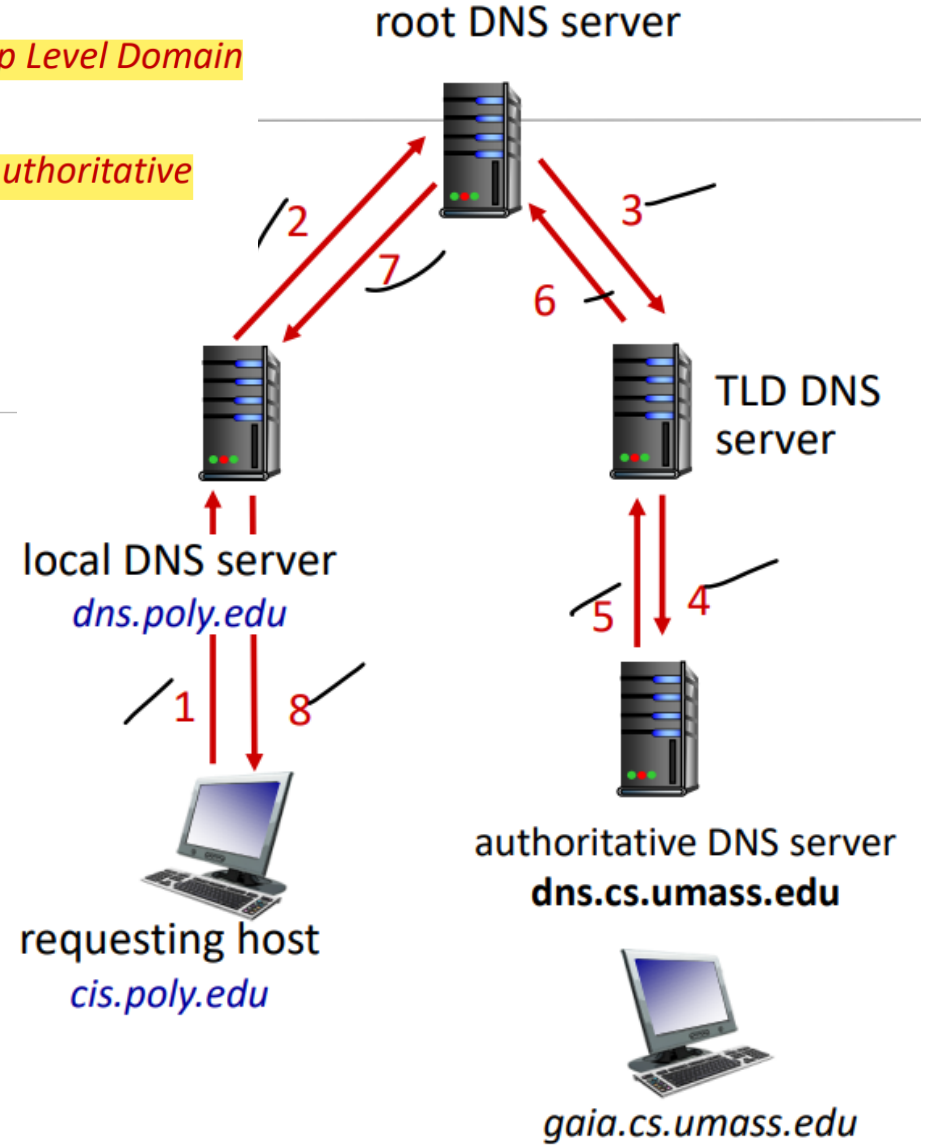
RR format: `(name, value, type, ttl)` time to last

type=A
- `name` is hostname
- `value` is IP address

type=NS
- `name` is domain (e.g., foo.com)
- `value` is hostname of authoritative name server for this domain

type=CNAME
- `name` is alias name for some "canonical" (the real) name
- `www.ibm.com` is really `servereast.backup2.ibm.com`
- `value` is canonical name

type=MX
- `value` is name of mailserver associated with `name`



msg header

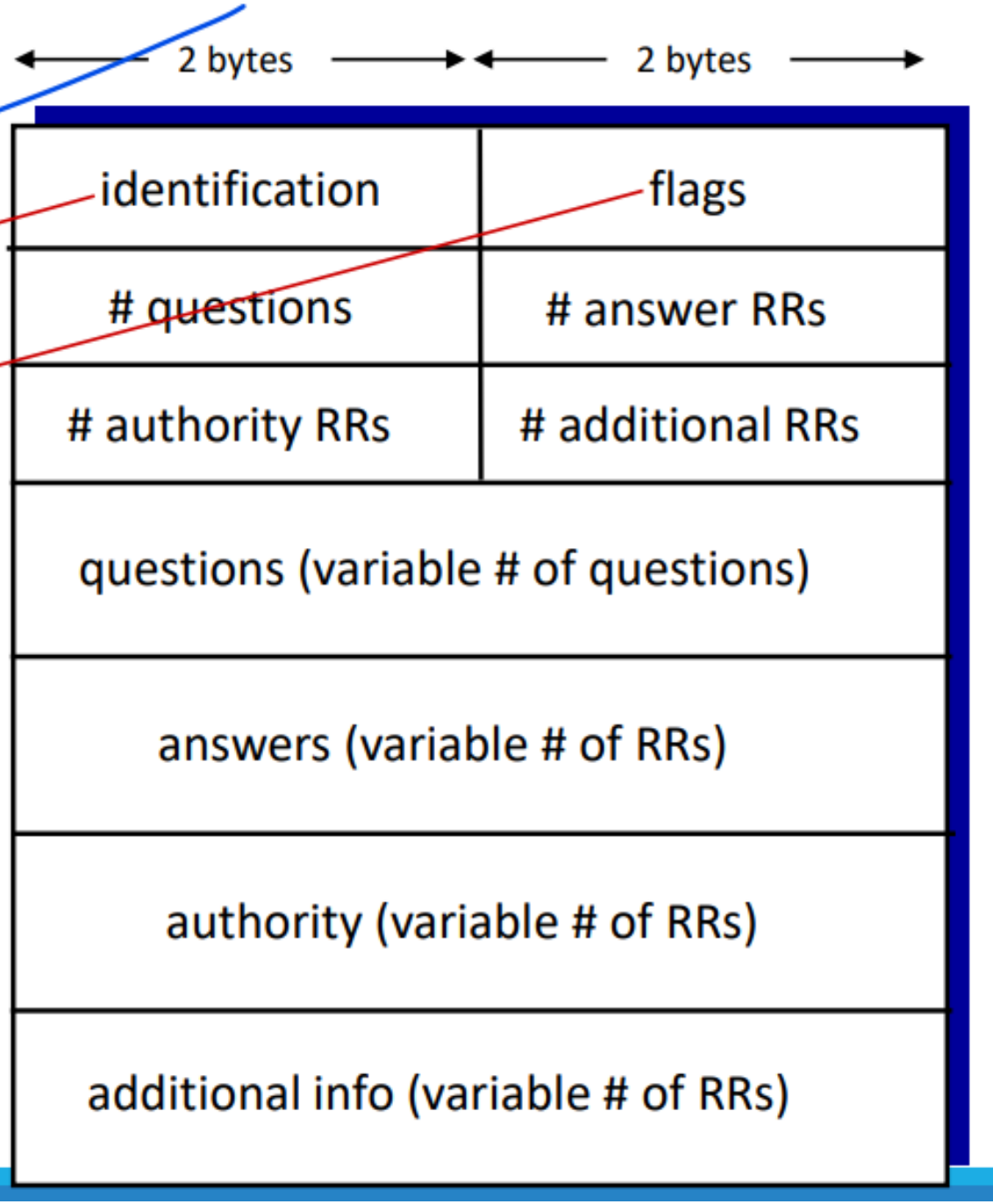
- identification: 16 bit # for query, reply to query uses same #
- flags:
  - query or reply
  - recursion desired
  - recursion available
  - reply is authoritative

name, type fields for a query

RRs in response to query

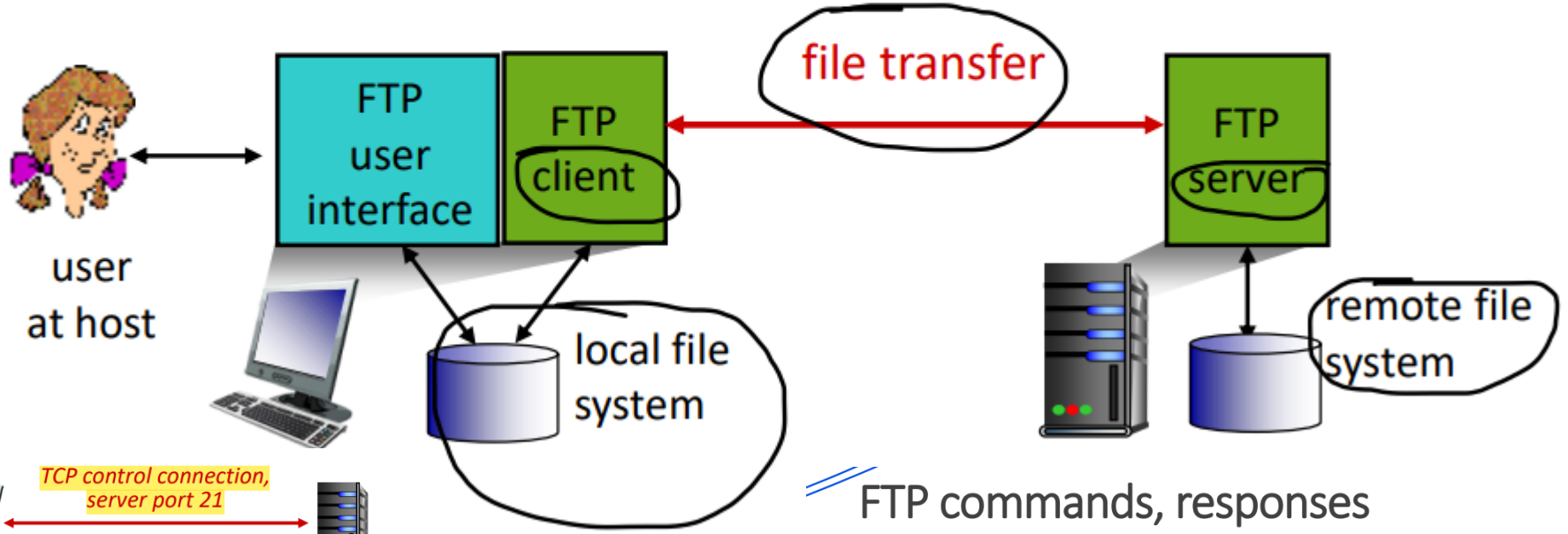
records for authoritative servers

additional "helpful" info that may be used

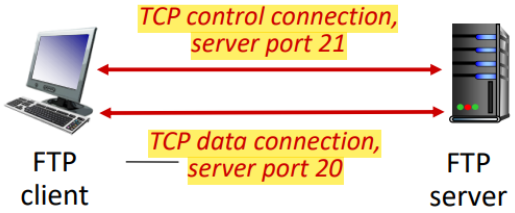


FTP: the file transfer protocol

Port 21



- FTP client contacts FTP server at port 21, using TCP
- client authorized over control connection
- client browses remote directory, sends commands over control connection
- when server receives file transfer command, *server* opens *2nd* TCP data connection (for file) *to* client
- after transferring one file, server closes data connection



- server opens another TCP data connection to transfer another file
- control connection: *"out of band"*
- FTP server maintains "state": current directory, earlier authentication

FTP commands, responses

sample commands:

- sent as ASCII text over control channel
- `USER username`
- `PASS password`
- `LIST` return list of file in current directory
- `RETR filename` retrieves (gets) file
- `STOR filename` stores (puts) file onto remote host

sample return codes

- status code and phrase (as in HTTP)
- `331 Username OK, password required`
- `125 data connection already open; transfer starting`
- `425 Can't open data connection`
- `452 Error writing file`

User Agent

- a.k.a. "mail reader"
- composing, editing, reading mail messages
- e.g., Outlook, Thunderbird, iPhone mail client
- outgoing, incoming messages stored on server

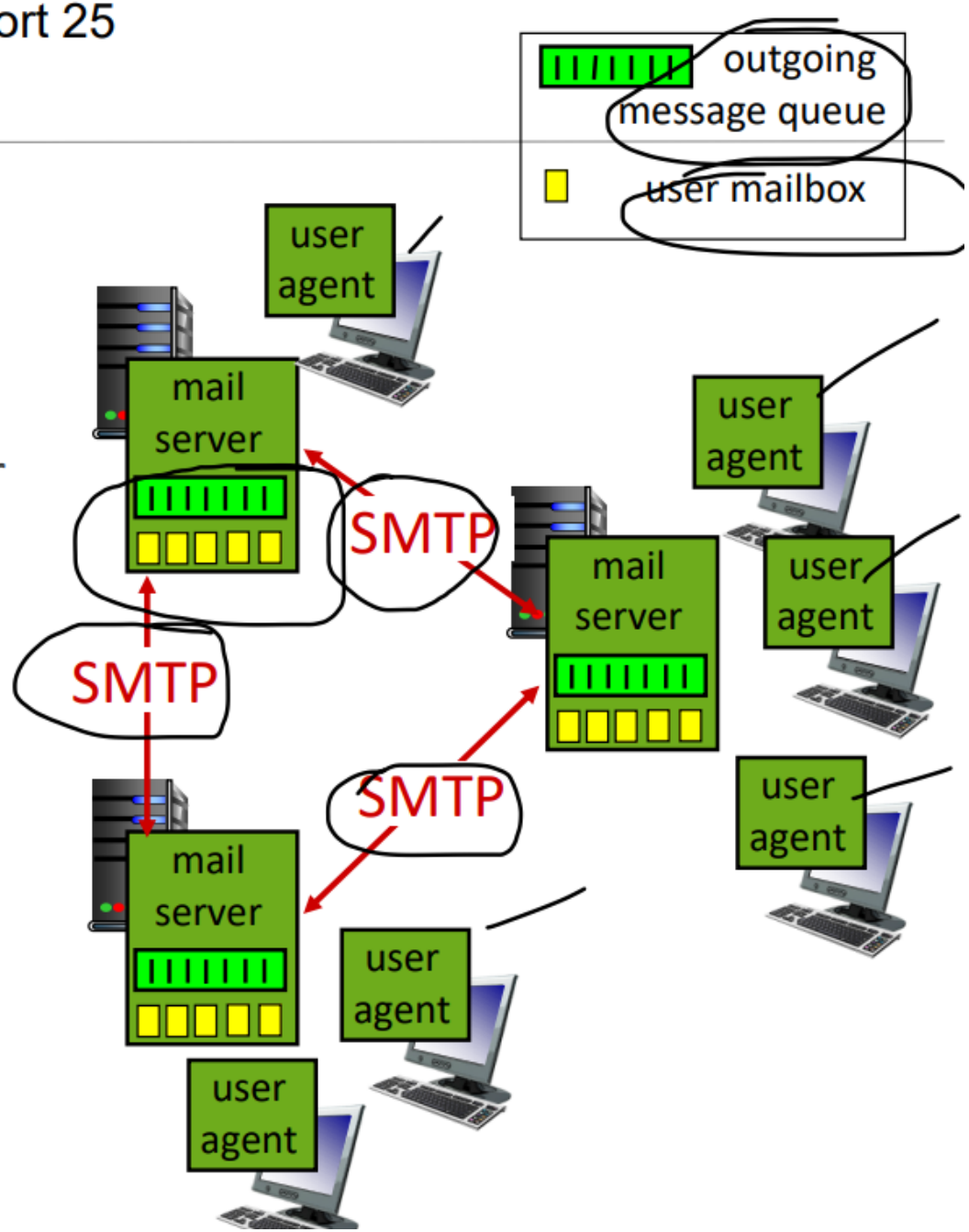
mail servers:

- *mailbox* contains incoming messages for user
- *message queue* of outgoing (to be sent) mail messages
- *SMTP protocol* between mail servers to send email messages
  - client: sending mail server
  - "server": receiving mail server

- three phases of transfer
  - handshaking (greeting)
  - transfer of messages
  - closure

il

Port 25



Scenario: Alice sends message to Bob

1) Alice uses UA to compose message "to" `bob@someschool.edu`

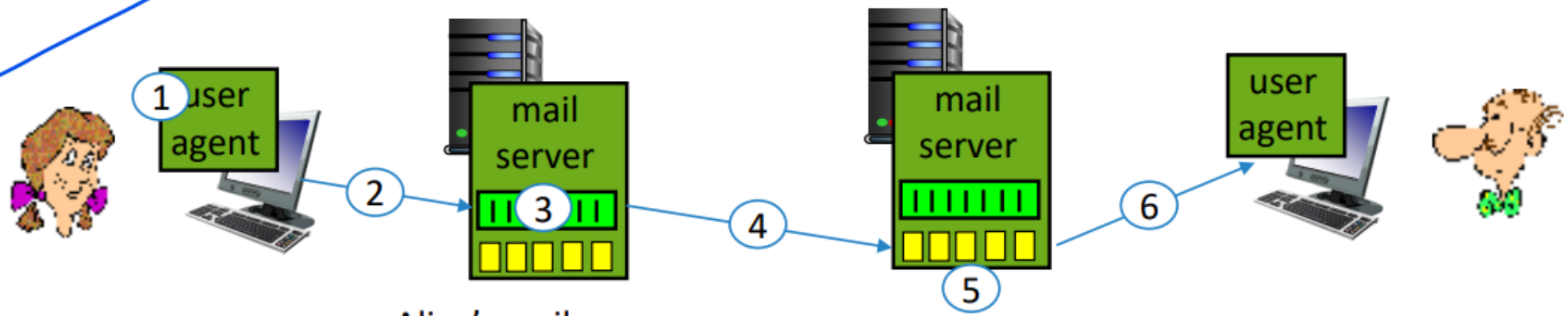
2) Alice's UA sends message to her mail server; message placed in message queue

3) client side of SMTP opens TCP connection with Bob's mail server

4) SMTP client sends Alice's message over the TCP connection

5) Bob's mail server places the message in Bob's mailbox

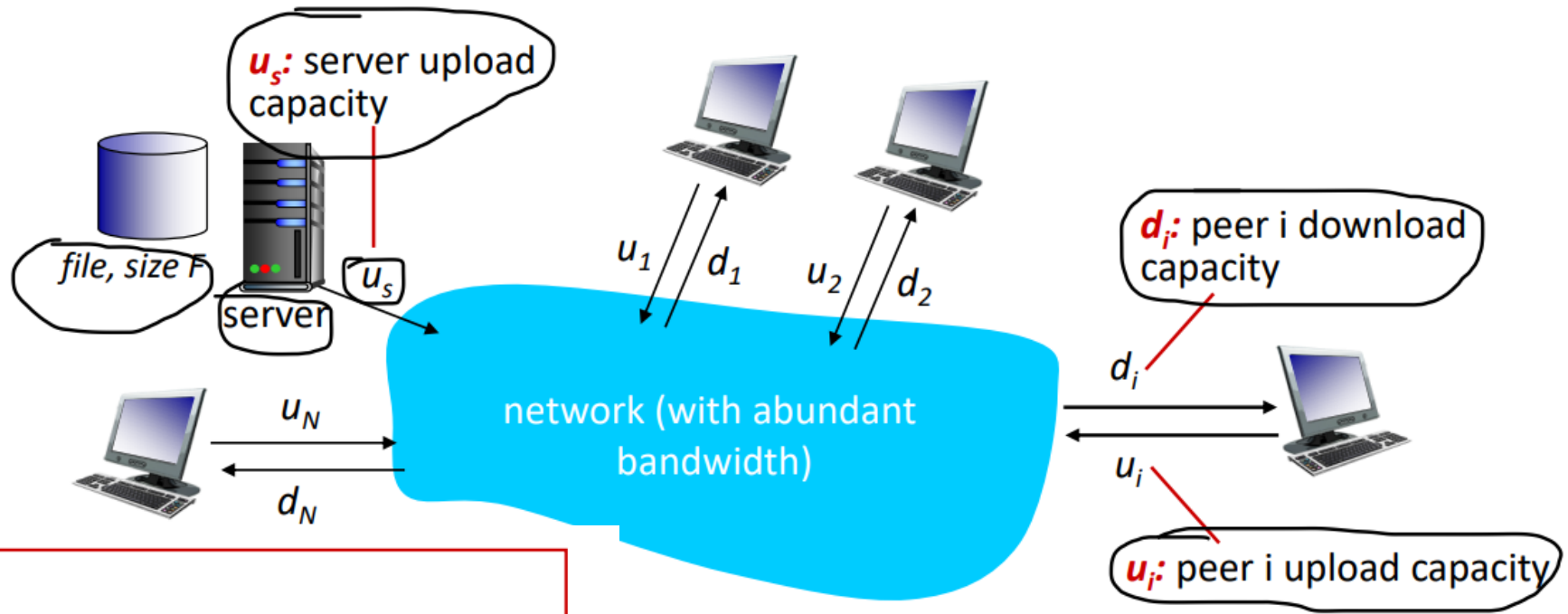
6) Bob invokes his user agent to read message



File distribution: client-server vs P2P

Question: how much time to distribute file (size *F*) from one server to *N peers*?

- peer upload/download capacity is limited resource



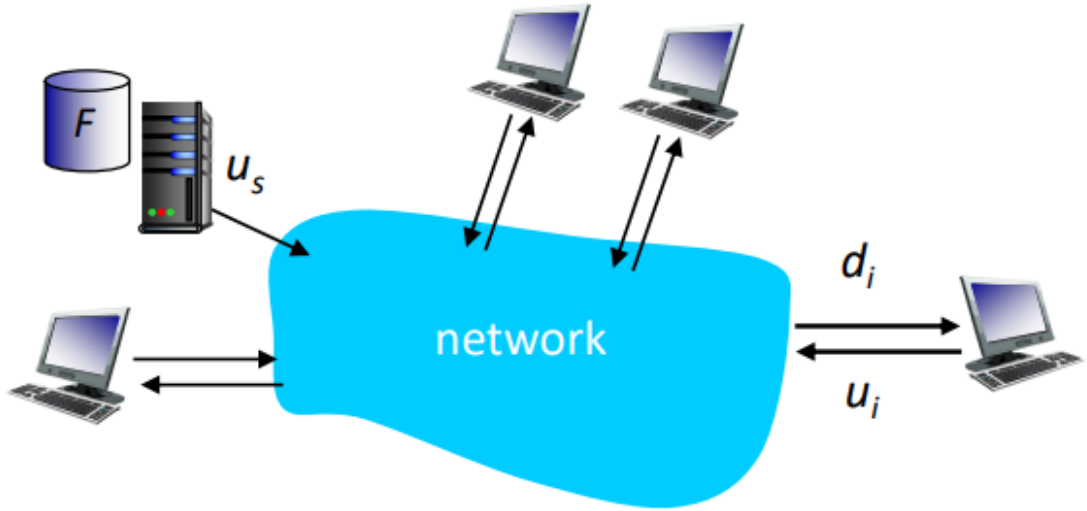
time to distribute F to N clients using client-server approach

$$D_{c\text{-}s} \geq max\{NF/u_s, F/d_{min}\}$$

increases linearly in N

File distribution time: P2P

- *server transmission:* must upload at least one copy
  - time to send one copy: F/u_s



- *client:* each client must download file copy
  - min client download time: F/d_{min}
- *clients:* as aggregate must download NF bits
  - max upload rate (limting max download rate) is $u_s + \Sigma u_i$

time to distribute F to N clients using P2P approach

$$D_{P2P} > max\{F/u_s, F/d_{min}, NF/(u_s + \Sigma u_i)\}$$

increases linearly in N …

… but so does this, as each peer brings service capacity

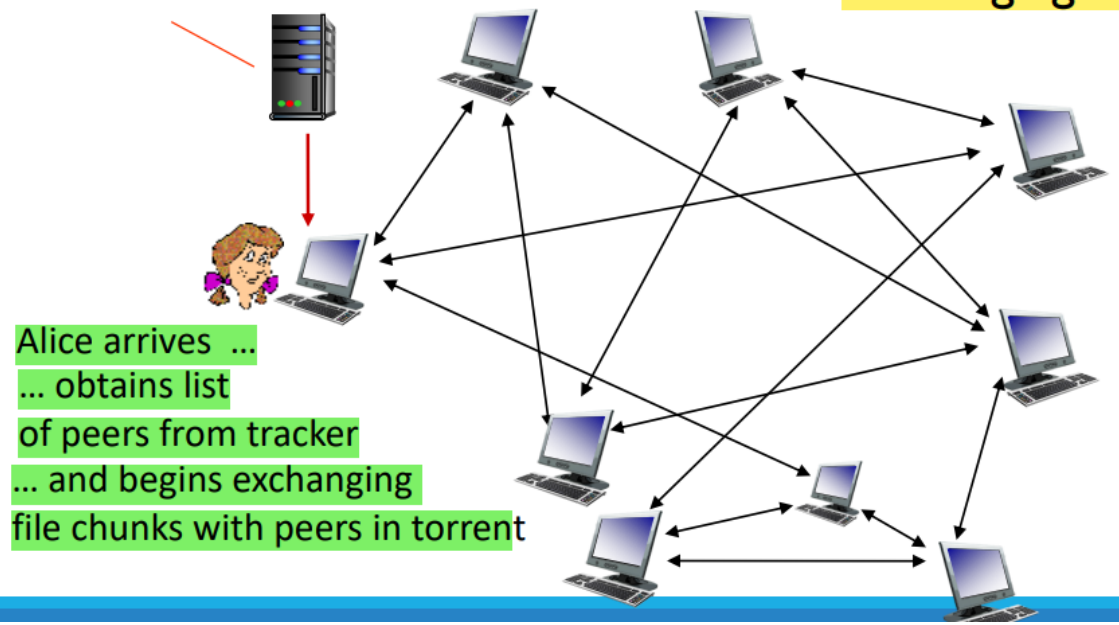
P2P file distribution: BitTorrent

- file divided into 256Kb chunks
- peers in torrent send/receive file chunks

Daymnnnnn!!

tracker: tracks peers participating in torrent

torrent: group of peers exchanging chunks of a file



Alice arrives ...
... obtains list of peers from tracker
... and begins exchanging file chunks with peers in torrent

requesting chunks:

- at any given time, different peers have different subsets of file chunks
- periodically, Alice asks each peer for list of chunks that they have
- Alice requests missing chunks from peers, rarest first

sending chunks: tit-for-tat

- Alice sends chunks to those four peers currently sending her chunks *at highest rate*
  - other peers are choked by Alice (do not receive chunks from her)
  - re-evaluate top 4 every10 secs
- every 30 secs: randomly select another peer, starts sending chunks
  - "optimistically unchoke" this peer
  - newly chosen peer may join top 4

Unit 2

Multiplexing/demultiplexing

The job of delivering the data in a transport-layer segment to the correct socket is called demultiplexing

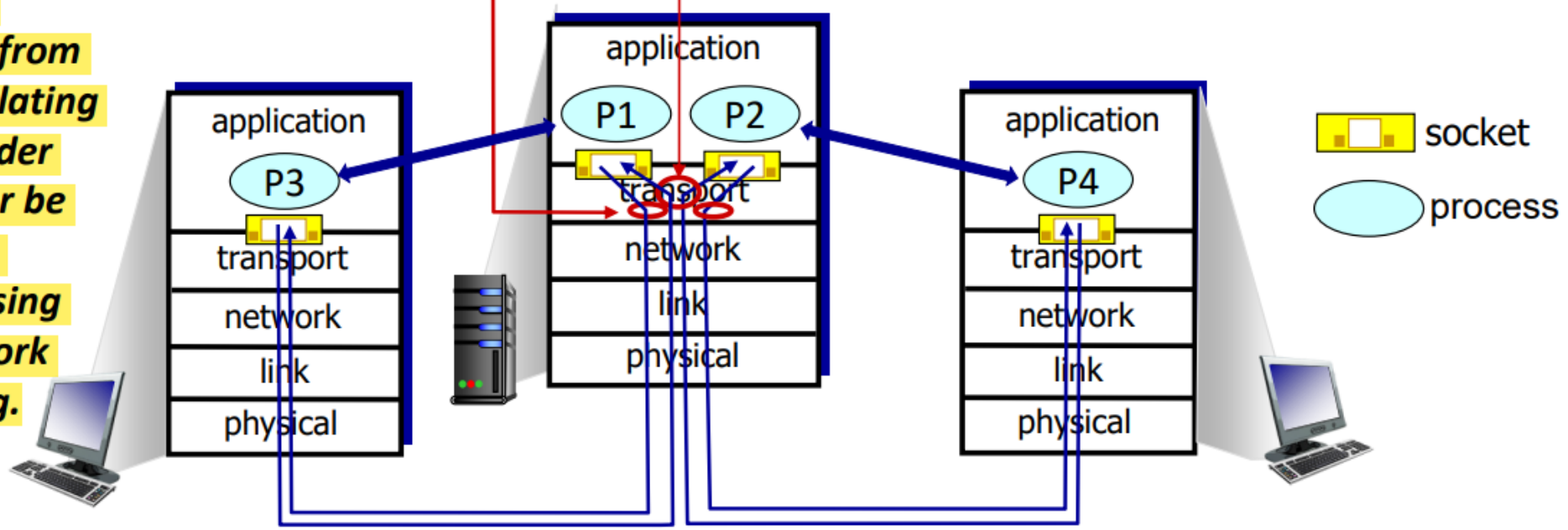
multiplexing at sender:

handle data from multiple sockets, add transport header (later used for demultiplexing)

demultiplexing at receiver:

use header info to deliver received segments to correct socket

The job of gathering data chunks at the source host from different sockets, encapsulating each data chunk with header information (that will later be used in demultiplexing) to create segments, and passing the segments to the network layer is called multiplexing.



Connectionless demultiplexing UDP

Recall:

- when creating socket, must specify *host-local* port #:

```
DatagramSocket mySocket1
 = new DatagramSocket(12534);
```

- when creating datagram to send into UDP socket, must specify
  - destination IP address
  - destination port #

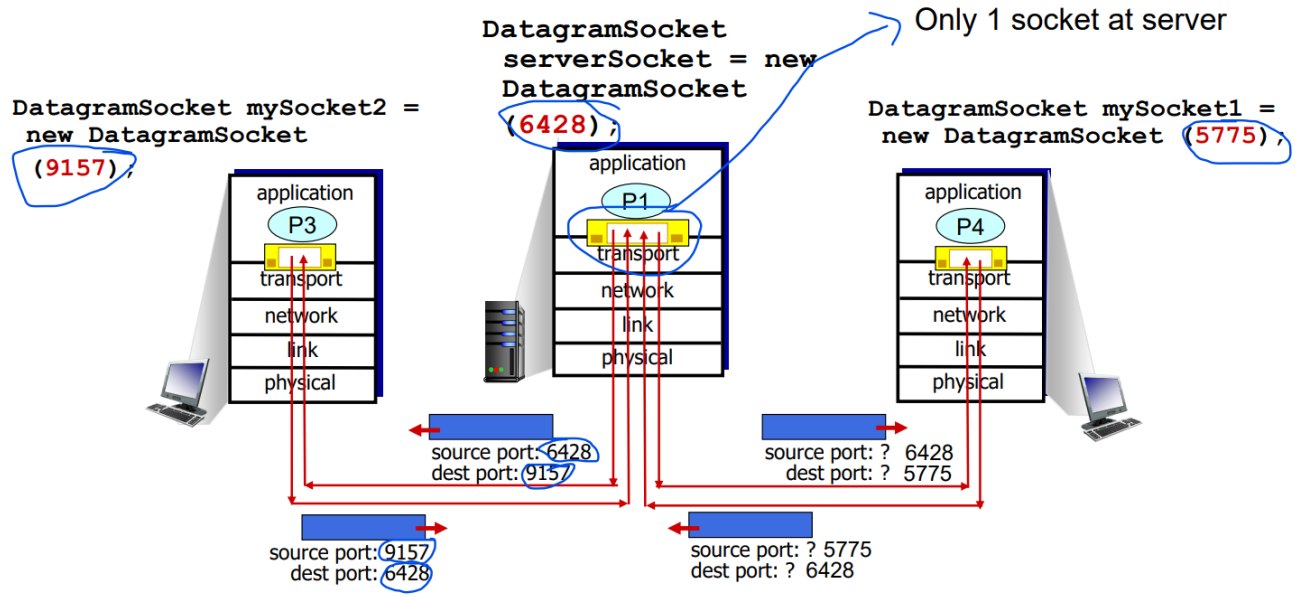
when receiving host receives *UDP* segment:

- checks destination port # in segment
- directs UDP segment to socket with that port #

IP/UDP datagrams with *same dest. port #,* but different source IP addresses and/or source port numbers will be directed to *same socket* at receiving ho

Simple
Faster
Transaction-oriented,
Non reliable
Un-secure
Connection-less
Stateless,
Less space

Connectionless demultiplexing: an example



Connection-oriented demultiplexing TCP

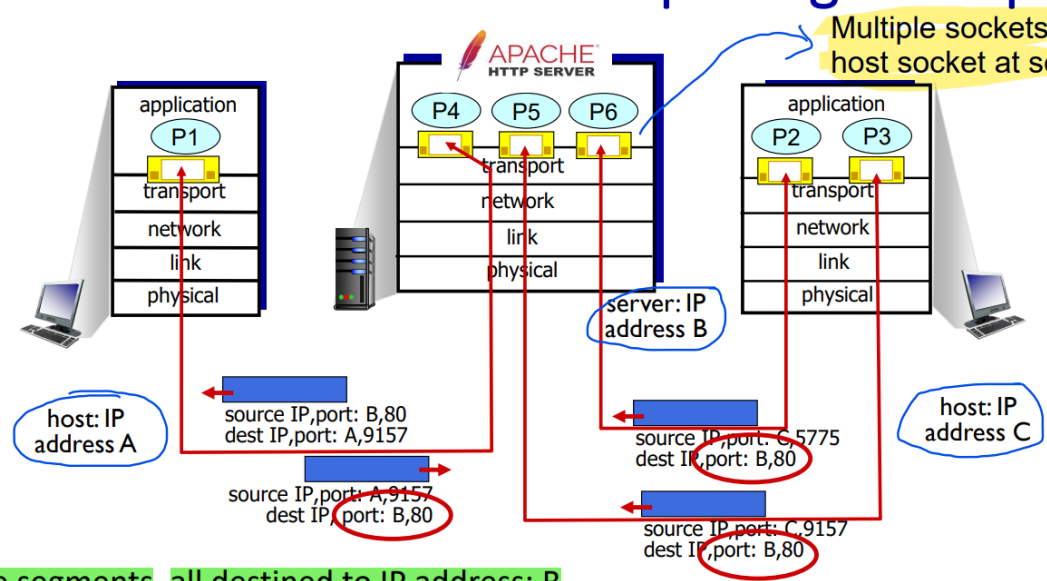
- TCP socket identified by 4-tuple:
  - source IP address
  - source port number
  - dest IP address
  - dest port number
- demux: receiver uses *all four values* (*4-tuple*) to direct segment to appropriate socket

- server may support many simultaneous TCP sockets:
  - each socket identified by its own 4-tuple
  - each socket associated with a different connecting client

4- Tuple

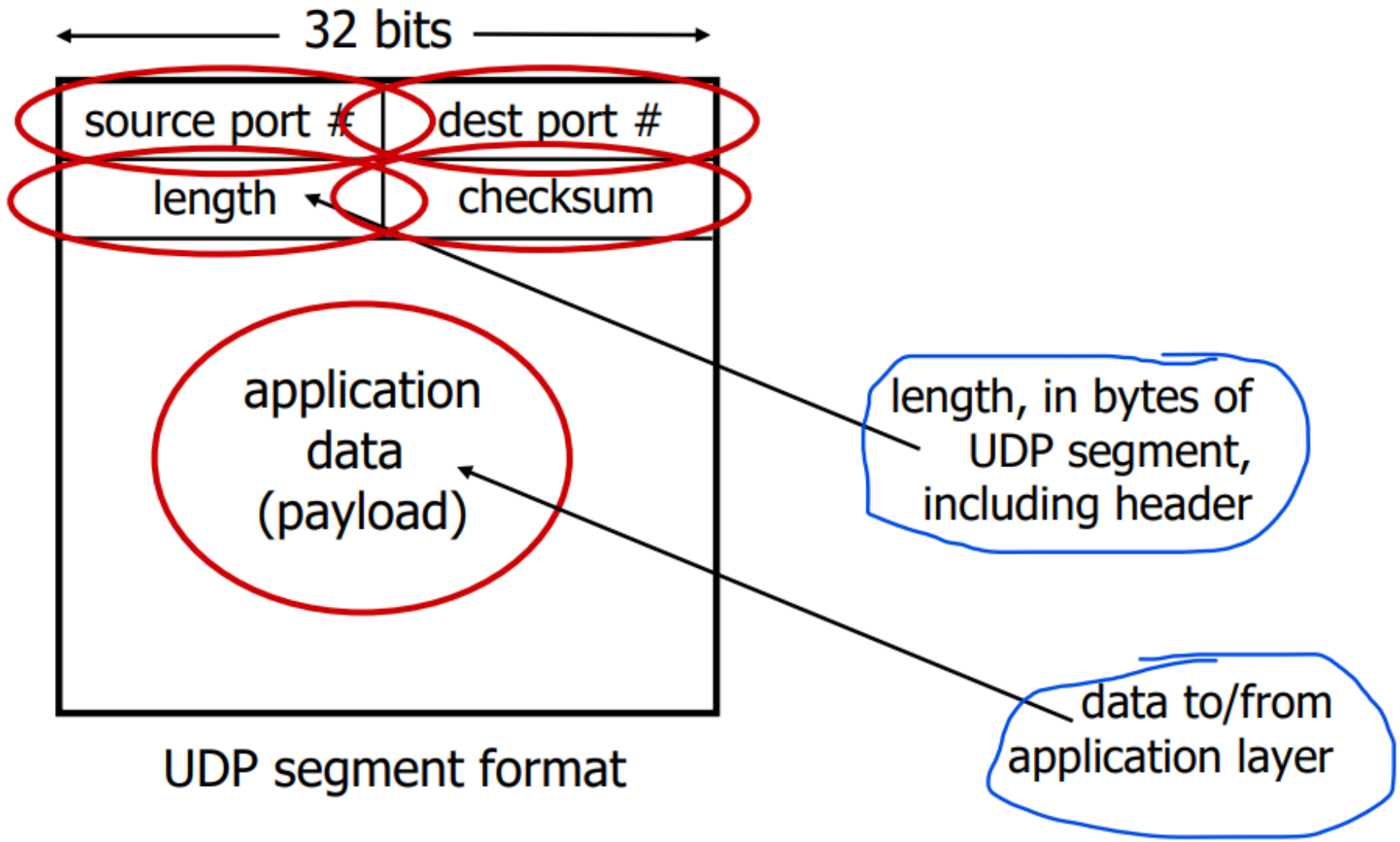
(Src IP, Src Port #, Dest Ip, Dest Port #)

Connection-oriented demultiplexing: example



Three segments, all destined to IP address: B, dest port: 80 are demultiplexed to *different* sockets

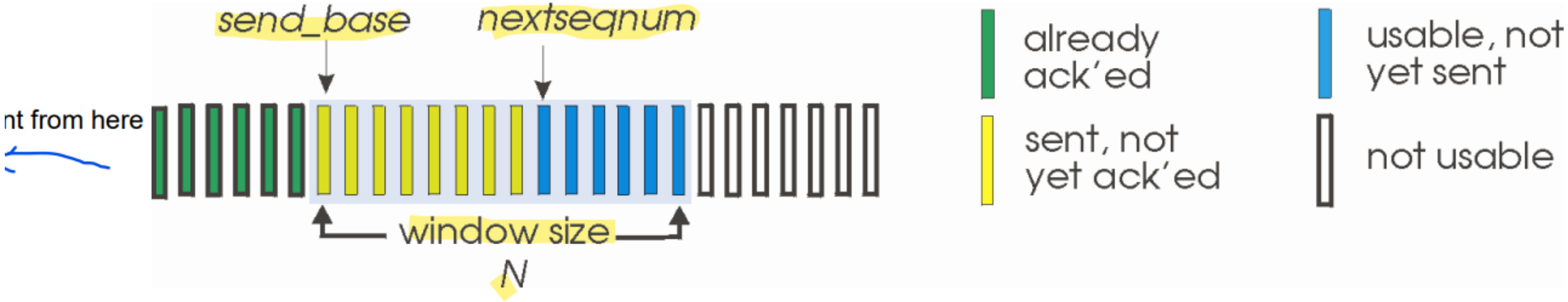
UDP segment header



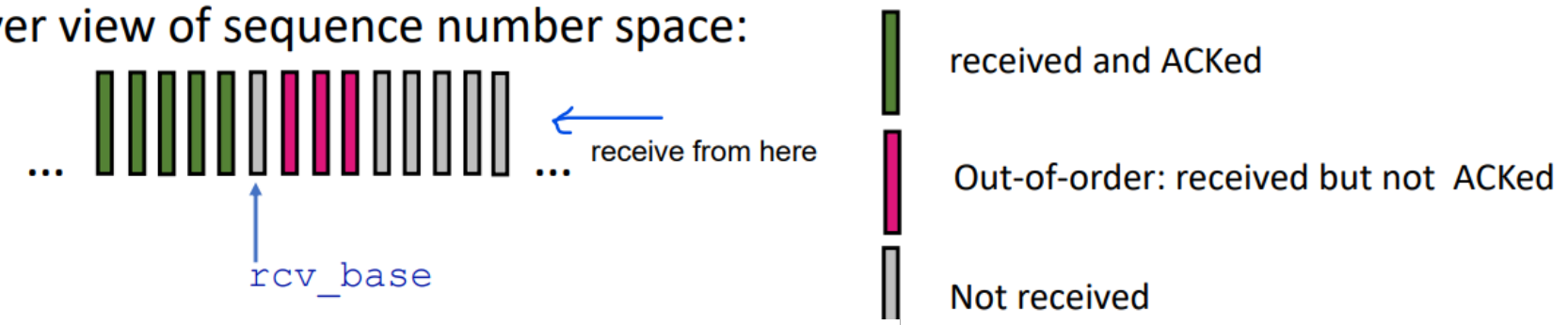
UDP segment format

Go-Back-N: sender

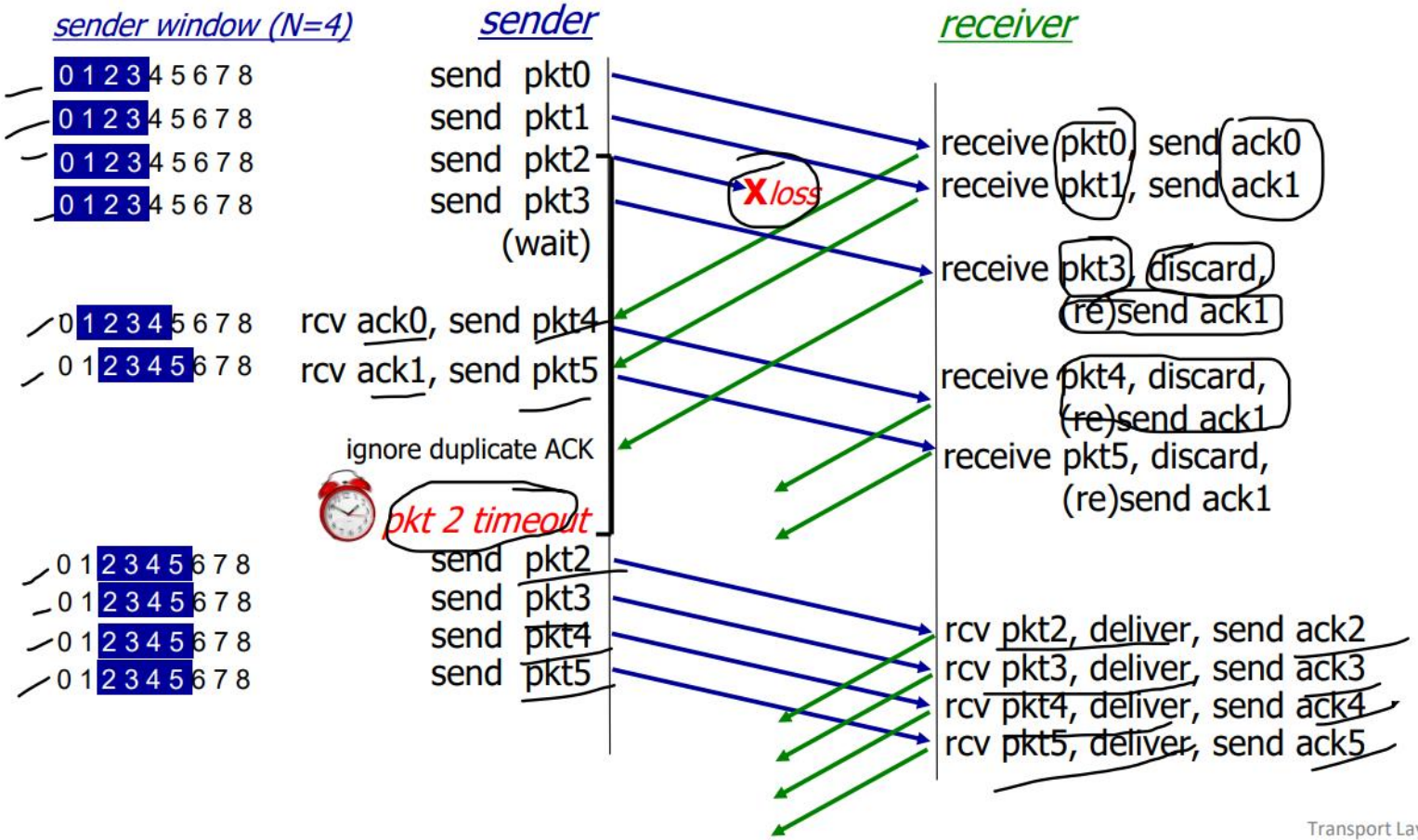
- sender: "window" of up to N, consecutive transmitted but unACKed pkts
  - k-bit seq # in pkt header



- *cumulative ACK:* ACK(*n*): ACKs all packets up to, including seq # *n*
  - on receiving ACK(*n*): move window forward to begin at *n+1*
- timer for oldest in-flight packet
- *timeout(n):* retransmit packet n and all higher seq # packets in window



Go-Back-N in action



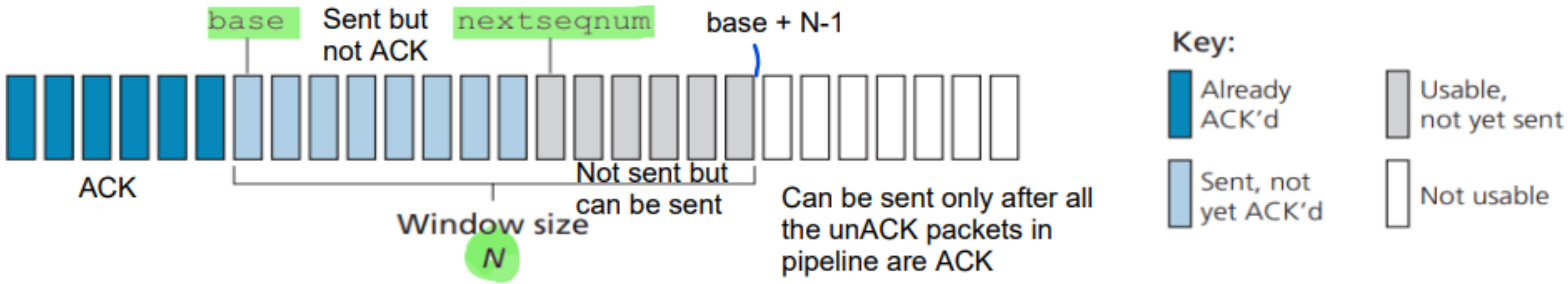
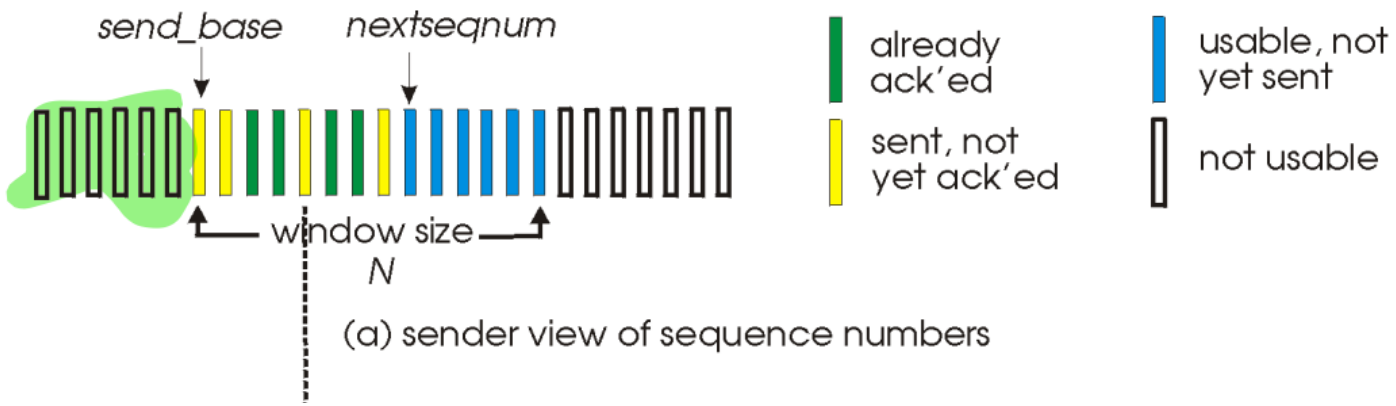


Figure 3.19 ♦ Sender's view of sequence numbers in Go-Back-N

- **Base-** to be the sequence number of the oldest unacknowledged packet
- **nextseqnum** - to be the smallest unused sequence number (that is, the sequence number of the next packet to be sent).
- Sequence numbers in the interval **[0,base-1]** correspond to packets that have already been transmitted and acknowledged.
- The interval **[base,nextseqnum-1]** corresponds to packets that have been sent but not yet acknowledged.
- Sequence numbers in the interval **[nextseqnum,base+N-1]** can be used for packets that can be sent immediately, should data arrive from the upper layer.
- sequence numbers **greater than or equal to base+N** cannot be used until an unacknowledged packet currently in the pipeline (specifically, the packet with sequence number base) has been acknowledged

Selective repeat: sender, receiver windows



sender

data from above:

- if next available seq # in window, send packet

timeout(*n*):

- resend packet *n*, restart timer

ACK(*n*) in [sendbase,sendbase+N]:

- mark packet *n* as received
- if n smallest unACKed packet, advance window base to next unACKed seq #

receiver

packet *n* in [rcvbase, rcvbase+N-1]

- send ACK(*n*)
- out-of-order: buffer
- in-order: deliver (also deliver buffered, in-order packets), advance window to next not-yet-received packet

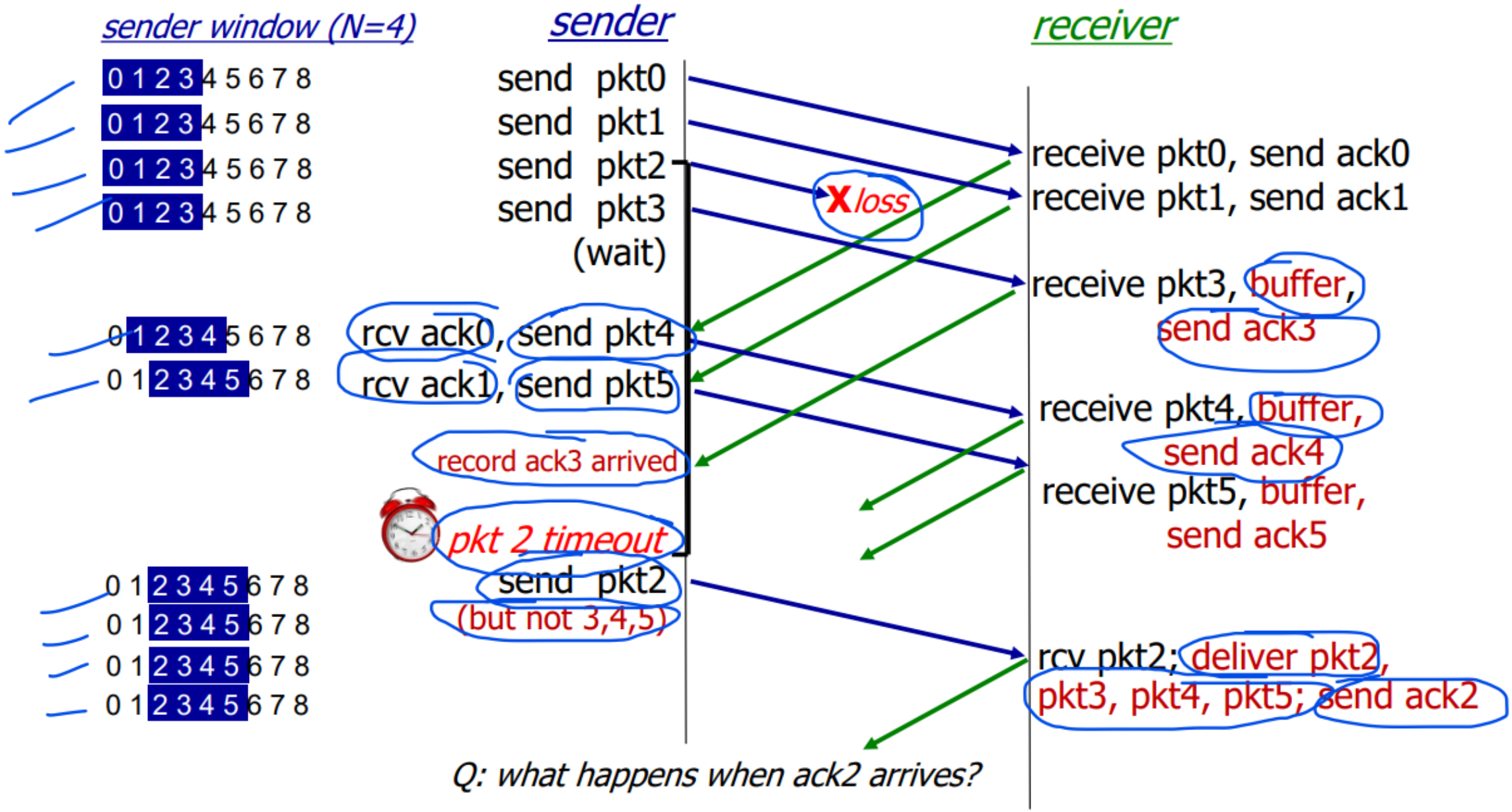
packet *n* in [rcvbase-N,rcvbase-1]

- ACK(*n*)

otherwise:

- ignore

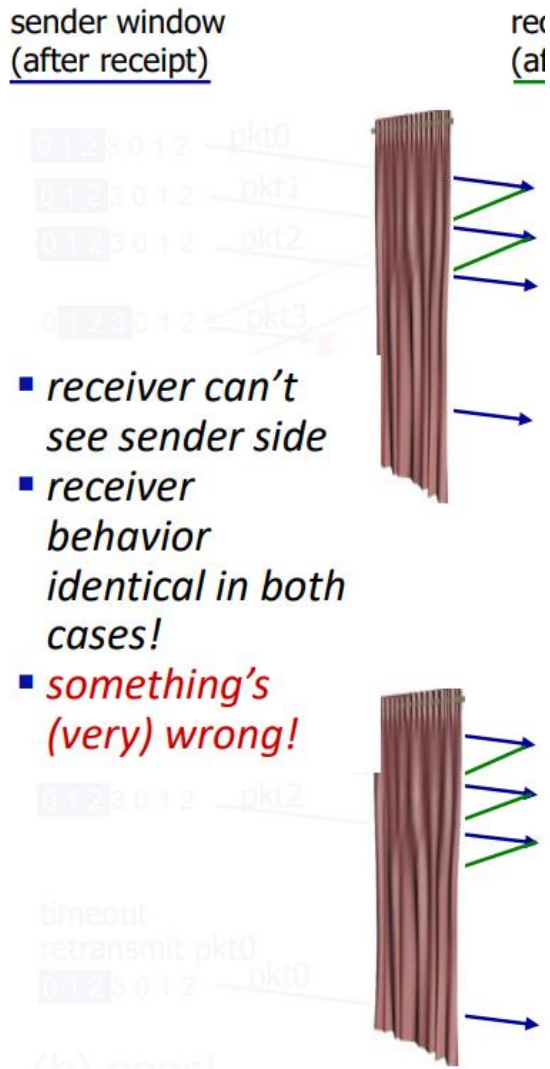
Selective Repeat in action

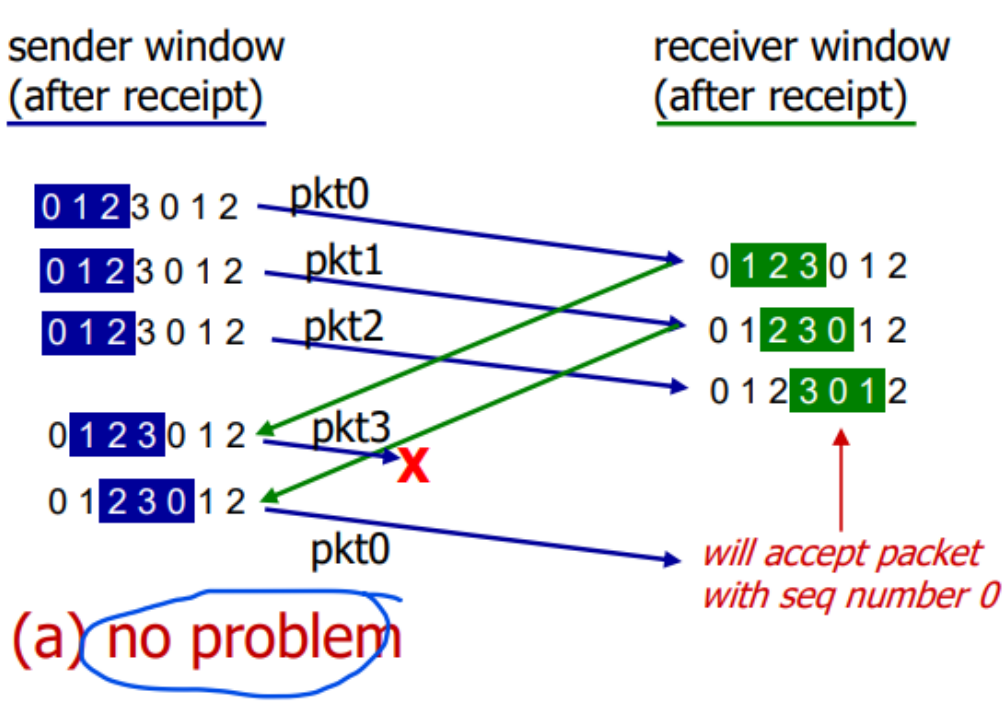


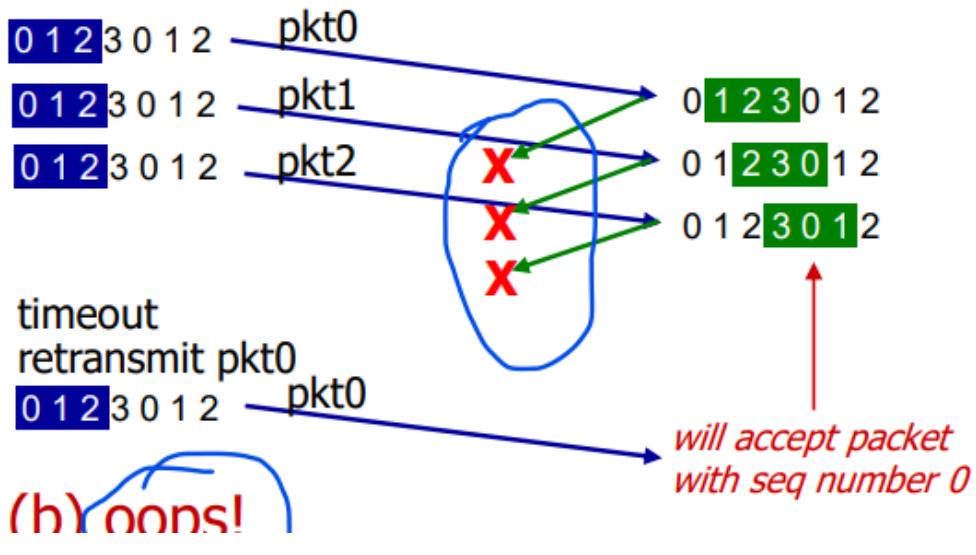
Selective repeat: a dilemma!

example:

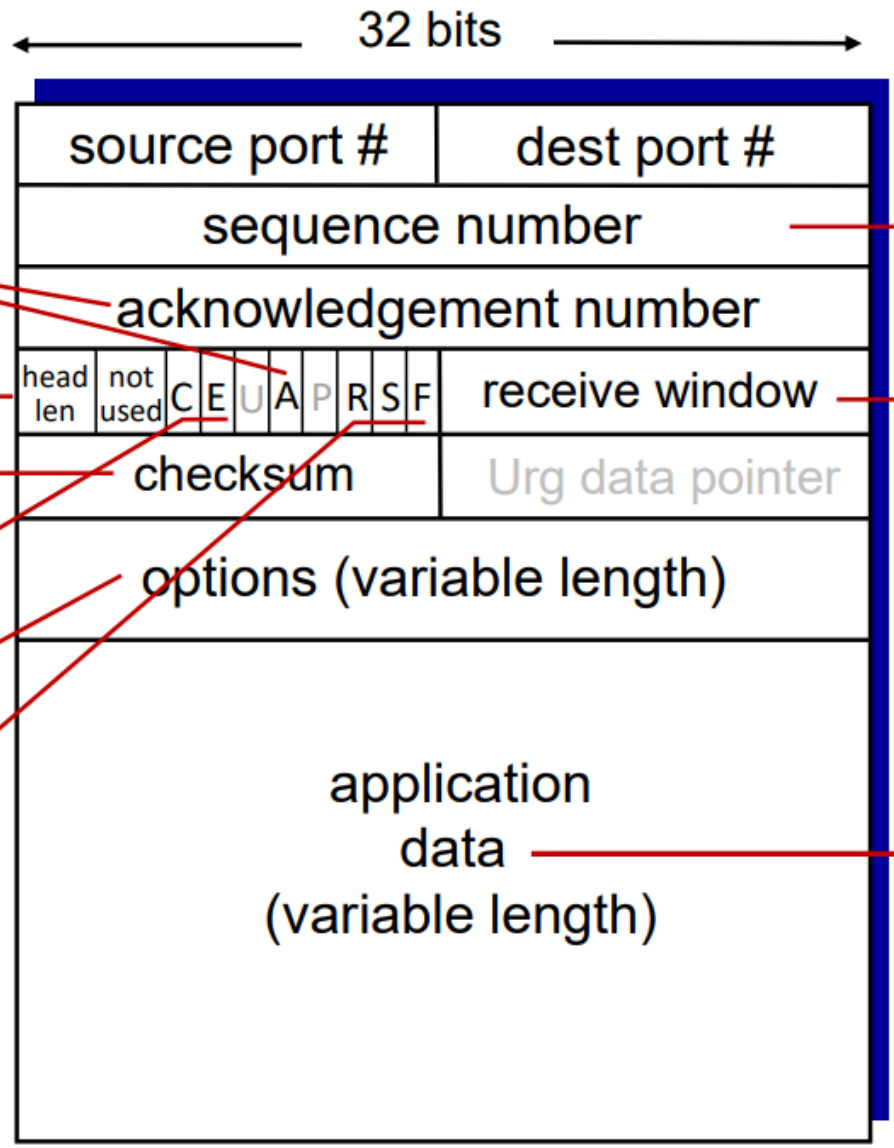
- seq #s: 0, 1, 2, 3 (base 4
- window size=3







TCP segment structure



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

length (of TCP header)

Internet **checksum**

C, E: congestion notification

TCP **options**

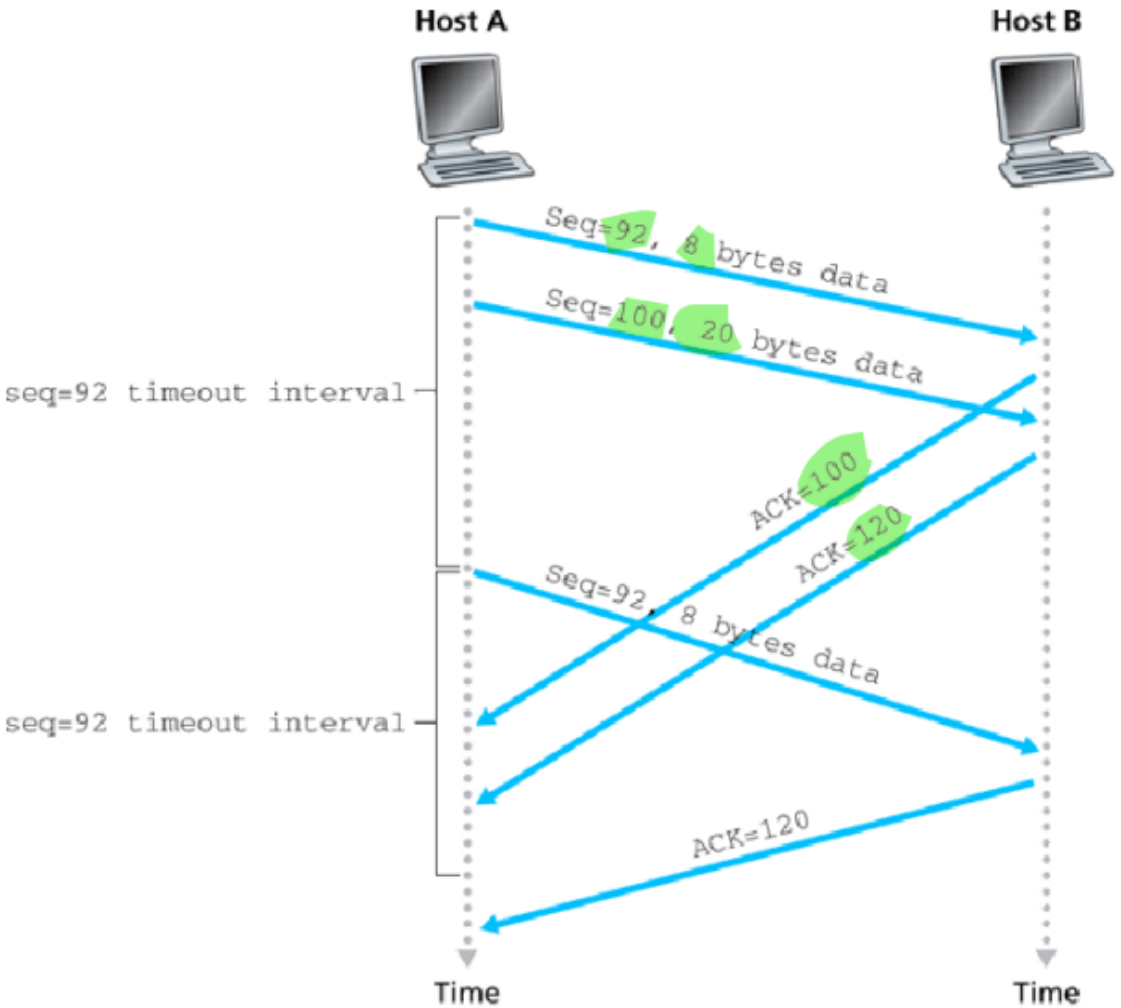
RST, SYN, FIN: connection management

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

flow control: # bytes receiver willing to accept

data sent by application into TCP socket

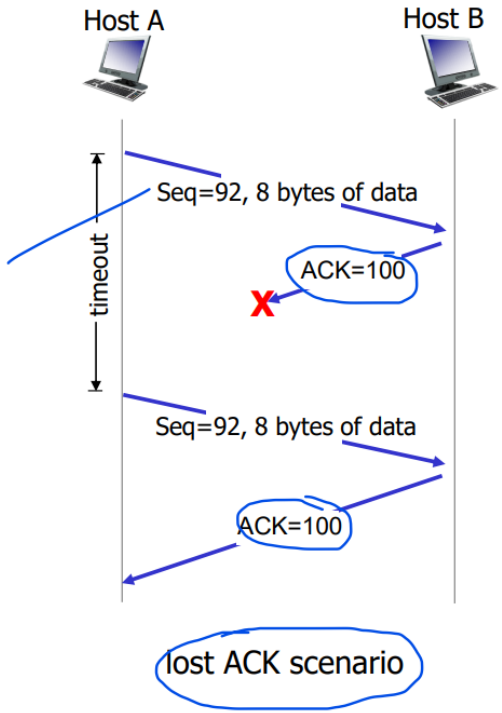
Cumulative Acknowledgement

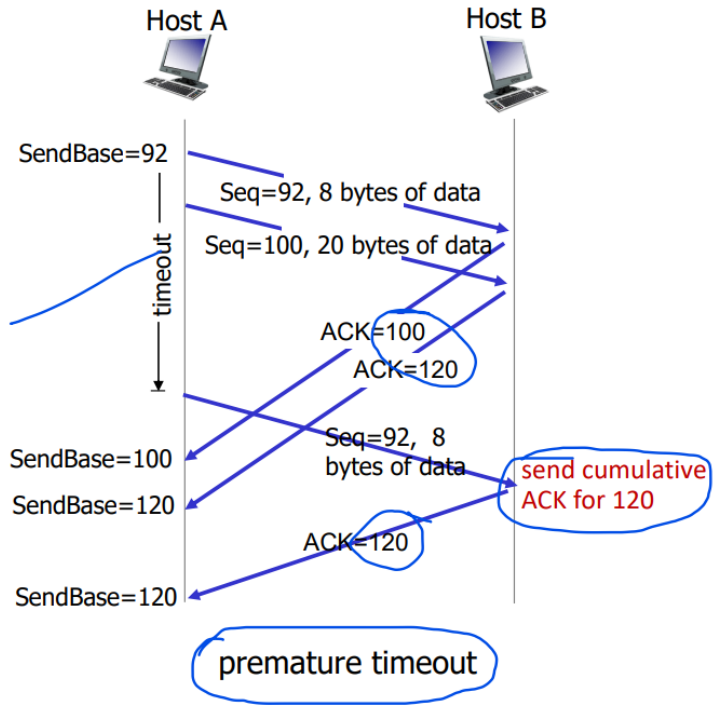


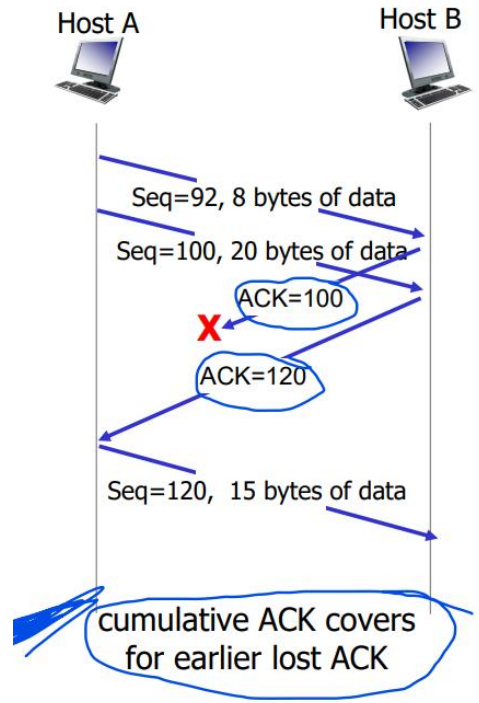
TCP is said to provide cumulative acknowledgments

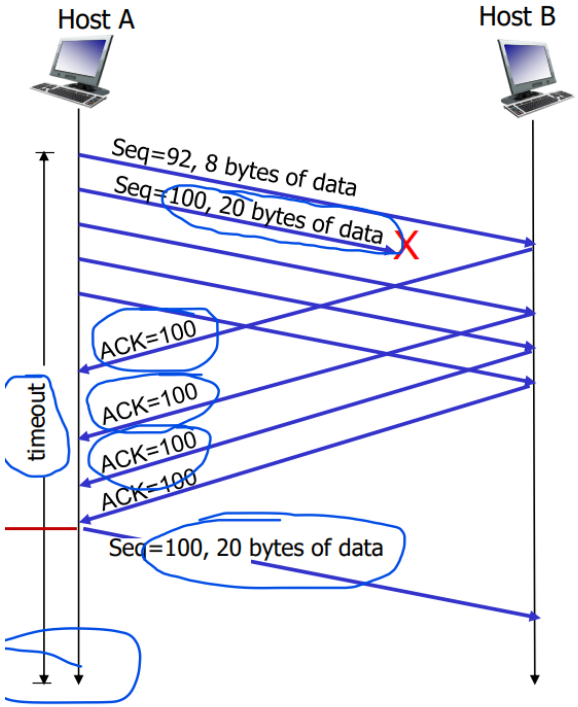
Next Seq # = ACK # =
Sent Seq # + No of Bytes sent

TCP: retransmission scenarios









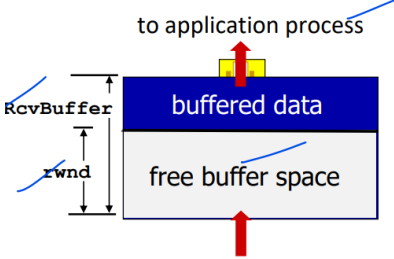
TCP fast retransmit

TCP fast retransmit

if sender receives 3 additional ACKs for same data ("triple duplicate ACKs"), resend unACKed segment with smallest seq #

- likely that unACKed segment lost, so don't wait for timeout

TCP Flow Control – reciever Side



TCP receiver-side buffering

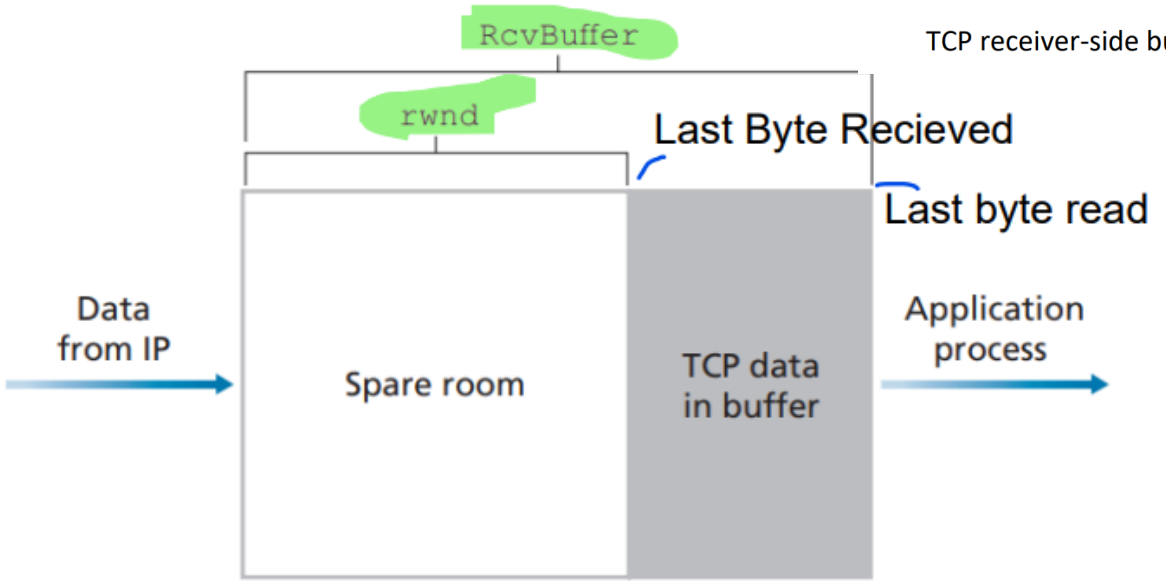


Figure 3.38 ♦ The receive window (rwnd) and the receive buffer (RcvBuffer)

- Tcp provides flow control by having sender maintain a variable called receive window **(rwnd).**

Last Byte Read- The number of last byte in data stream read from the buffer by the receiver application process.

Last Byte Rcvd: The number of last byte in data stream that has been arrived from the network and has been placed in the buffer.

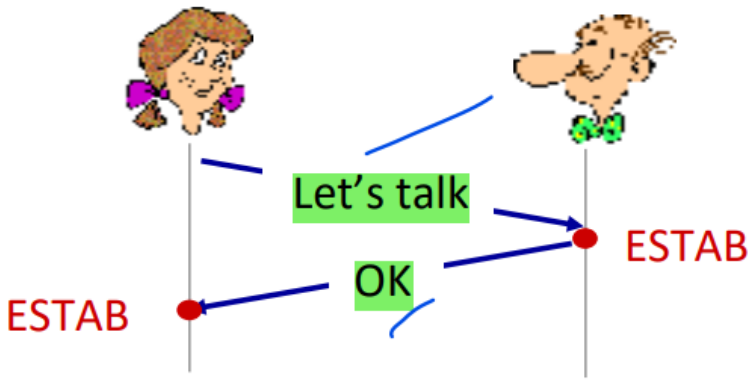
Rwnd = RcvBuffer – [**Last Byte Rcvd - Last Byte Read**]

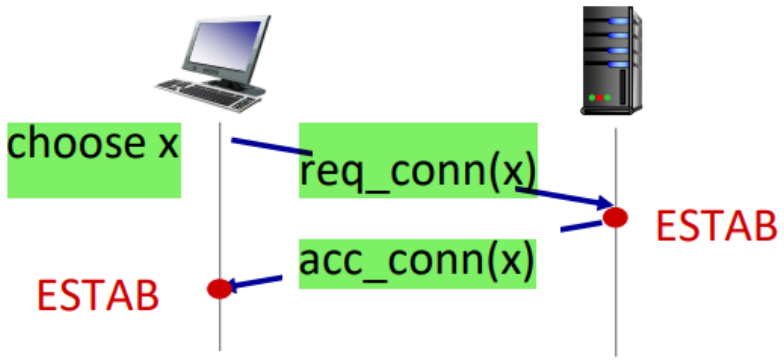
Initially ***rwnd=RcvBuffer***

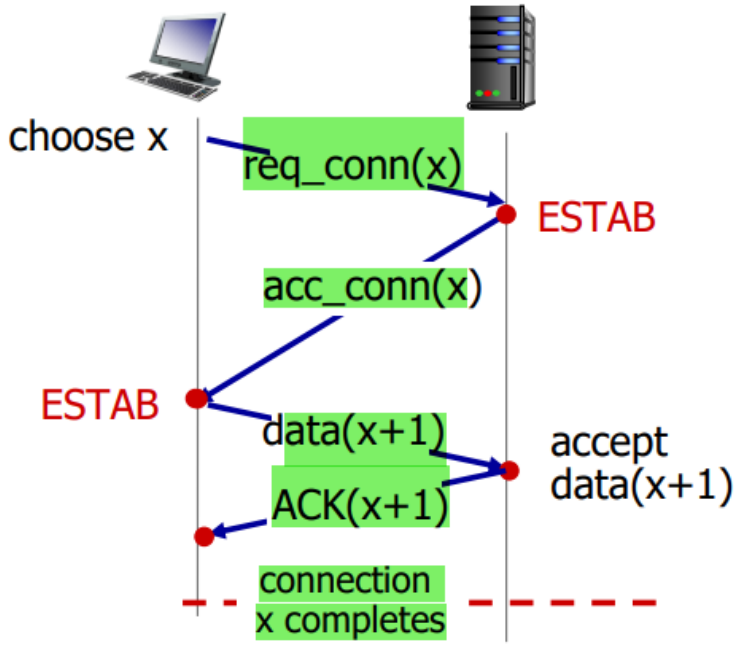
- To avoid Over flow:

Last Byte Rcvd- Last Byte Read < = RcvBuffer

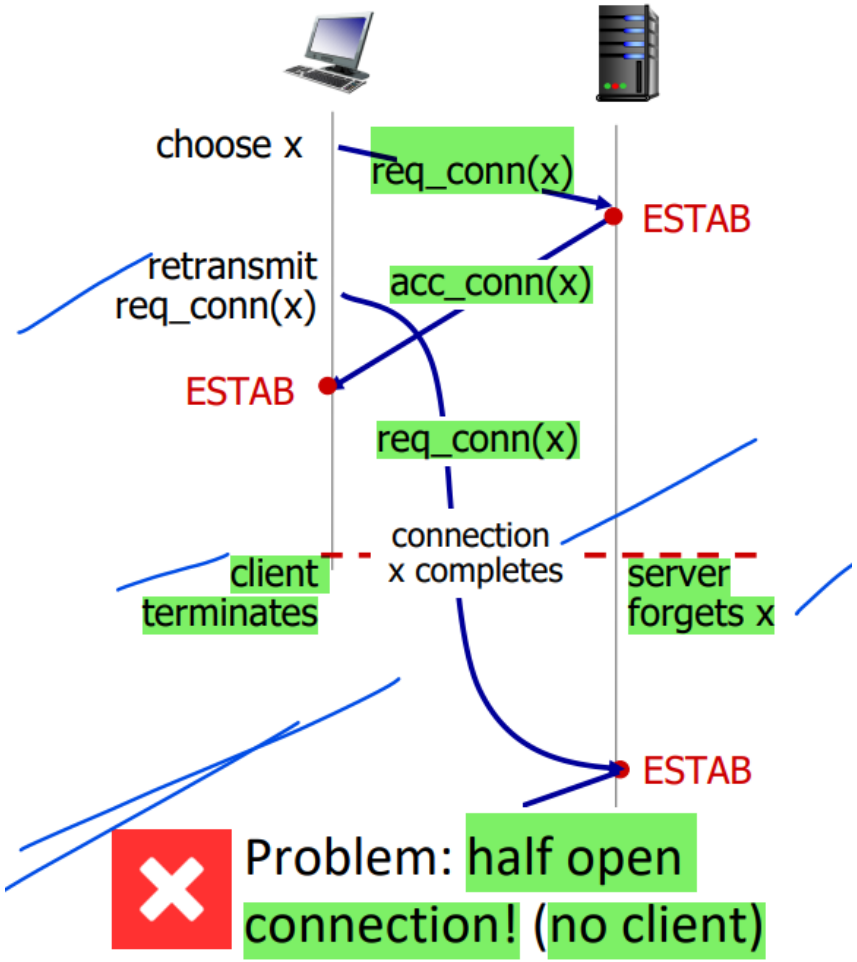
2-way handshake:







No problem!



Problem: half open connection! (no client)

TCP 3-way handshake

1. C: Accept?
2. S: Accepted!
3. C: Transmitting!

Starting a connection: SYN is used
Acknowledgement: ACK is used
Closing a connection: FIN is used

Client state

```
clientSocket = socket(AF_INET, SOCK_STREAM)
clientSocket.connect((serverName,serverPort))
```

Server state

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

LISTEN

choose init seq num, x
send TCP SYN msg

SYNSENT

SYNbit=1, Seq=x

choose init seq num, y
send TCP SYNACK
msg, acking SYN

LISTEN

SYN RCVD

SYNbit=1, Seq=y
ACKbit=1; ACKnum=x+1

received SYNACK(x)
indicates server is live;
send ACK for SYNACK;
this segment may contain
client-to-server data

ESTAB

ACKbit=1, ACKnum=y+1

received ACK(y)
indicates client is live

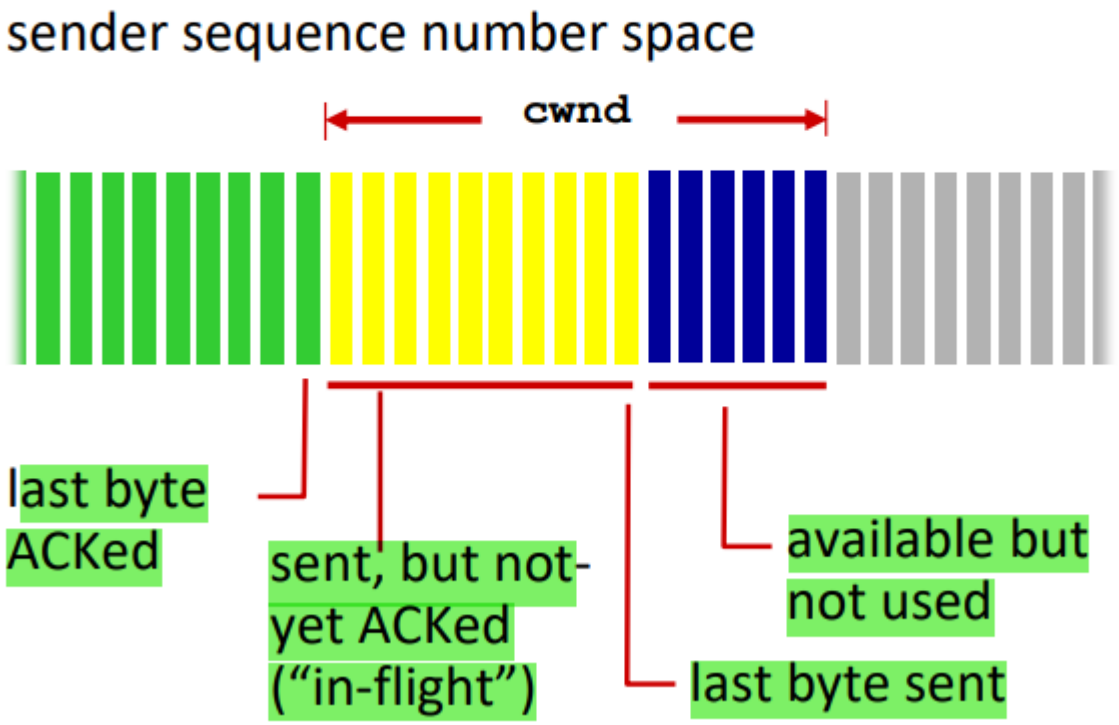
data transmission

FIN bit

FIN bit

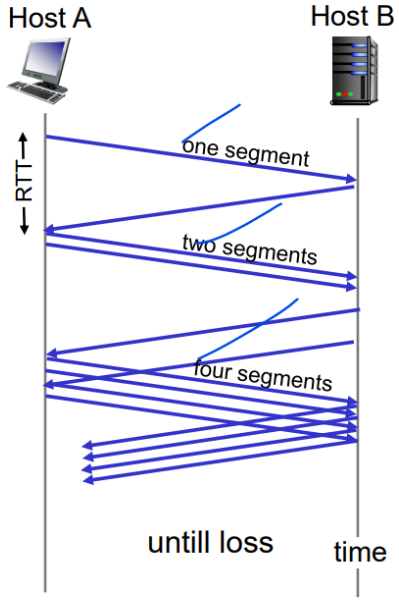
ESTAB

TCP congestion control:



TCP slow start

- when connection begins, increase rate exponentially until first loss event:
  - initially **cwnd** = 1 MSS
  - double **cwnd** every RTT
  - done by incrementing **cwnd** for every ACK received
- *summary:* initial rate is slow, but ramps up exponentially fast



LastByteSent - LastByteAcked ≤ min{cwnd, rwnd}

LastByteRecived - LastByteRead <= Rwnd/RcvBuffer

- 1. Lost segment implies congestion and hence the sender's rate should be decreased when a segment is lost.
- 3. Bandwidth Probing: *sstrsh(slow start threshold)*
- 2. An acknowledged segment indicates that the network is delivering the sender's segment to the receiver and hence the sender rate can be increased when an ACK arrives for the previously unacknowledged frame.

When should this exponential growth ends?

- 1. Lost segment (Time out):

Ssthresh = cwnd/2;

Slow Start THRESHold

- 2. when **cwnd >=ssthresh**
  - Then TCP enters Congestion Avoidance mode.

Q: when should the exponential increase switch to linear?

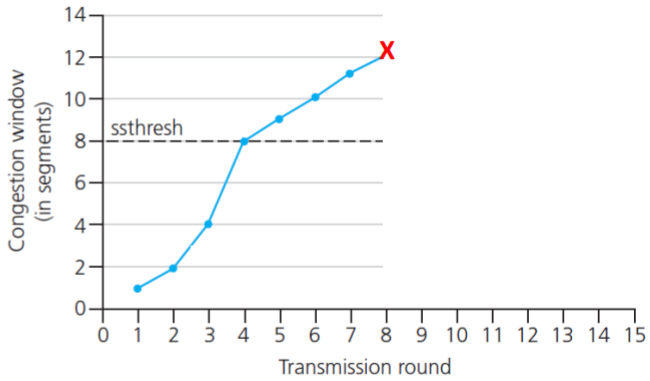
A: when **`cwnd`** gets to 1/2 of its value before timeout.

3. When three DUPACK are received:

Enter Fast recovery state :

Cwnd= sstresh+3MSS

Sstresh= cwnd/2



Implementation:

- variable **`ssthresh`**
- on loss event, **`ssthresh`** is set to 1/2 of **`cwnd`** just before loss event

- On entry on congestion state, value of cwnd will be half.
- TCP now adopts a more conservative approach I,e:

Cwnd=cwnd + MSS[MSS/cwnd]

This increases cwnd by 1/10 MSS for each iteration

- Congestion avoidance linear increase ends when timeout occurs or if 3 ACK duplicative is received.

TCP congestion control: AIMD

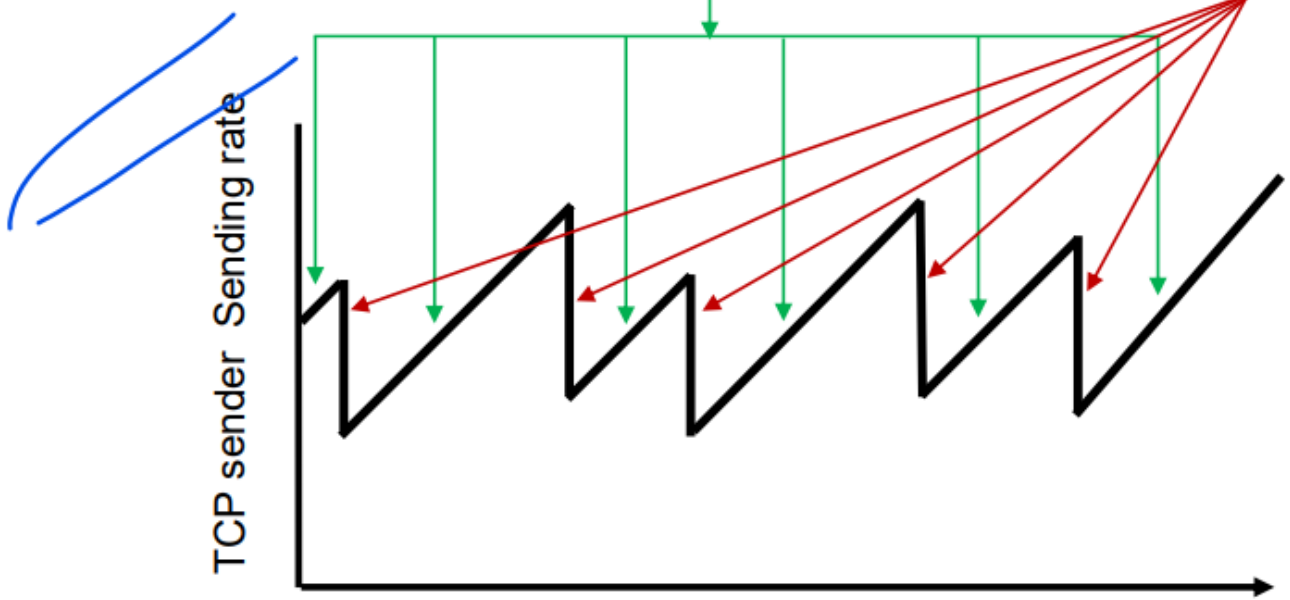
- *approach:* senders can increase sending rate until packet loss (congestion) occurs, then decrease sending rate on loss event

Additive Increase

increase sending rate by 1 maximum segment size every RTT until loss detected

Multiplicative Decrease

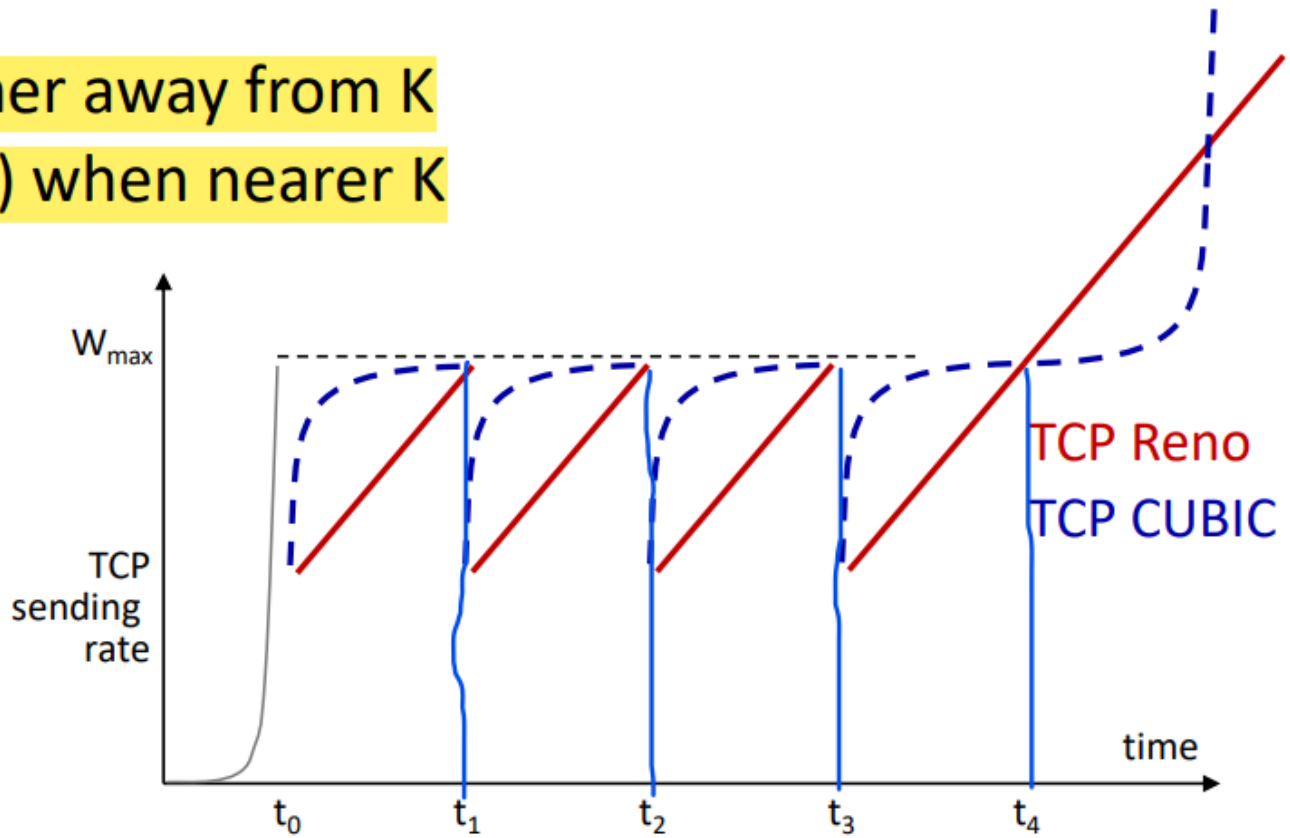
cut sending rate in half at each loss event



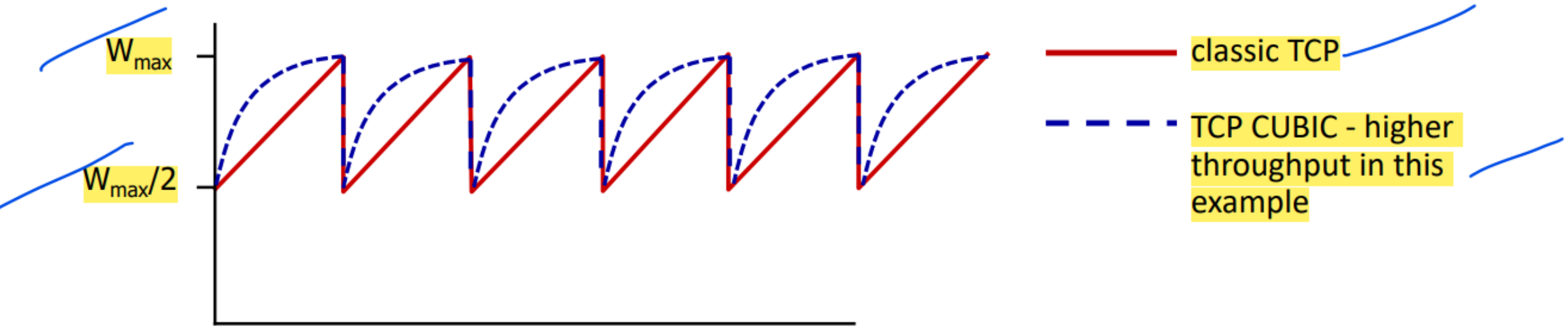
AIMD sawtooth behavior: *probing* for bandwidth

TCP CUBIC

- K: point in time when TCP window size will reach W_{max}
  - K itself is tuneable
- increase W as a function of the *cube* of the distance between current time and K
  - larger increases when further away from K
  - smaller increases (cautious) when nearer K
- TCP CUBIC default in Linux, most popular TCP for popular Web servers



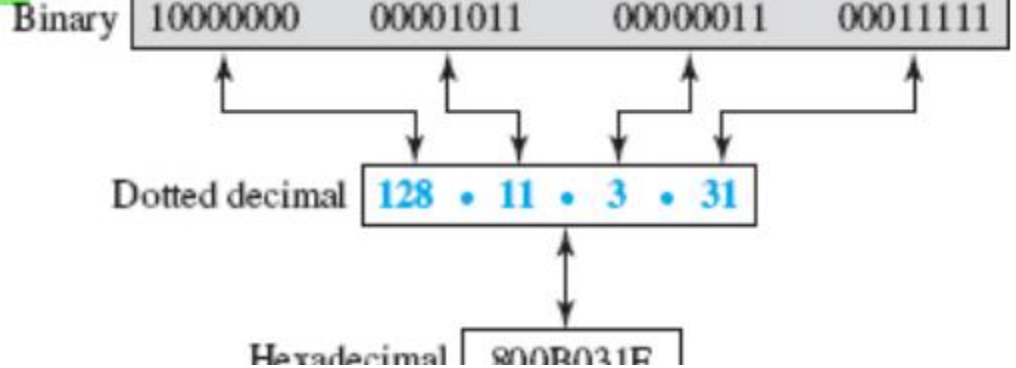
- after cutting rate/window in half on loss, initially ramp to to W_{max} *faster*, but then approach W_{max} more *slowly*

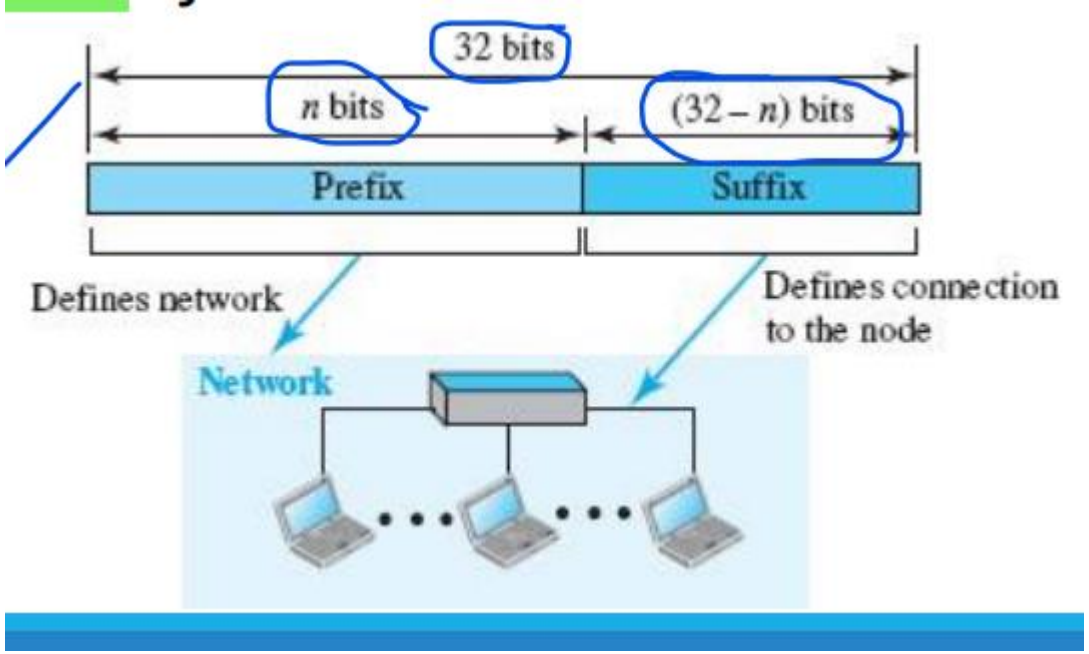


Multiplicative decrease detail: sending rate is

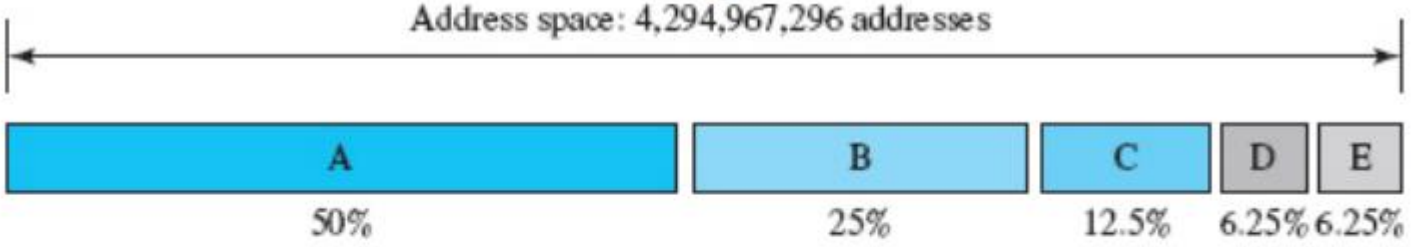
- Cut in half on loss detected by triple duplicate ACK (TCP Reno)
- Cut to 1 MSS (maximum segment size) when loss detected by timeout (TCP Tahoe)

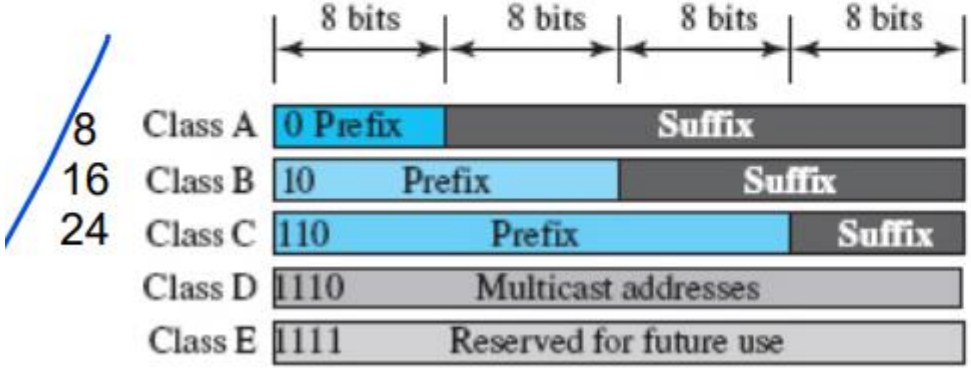
Unit 3





stand classless addressing.





Class	Prefixes	First byte
A	$n = 8$ bits	0 to 127
B	$n = 16$ bits	128 to 191
C	$n = 24$ bits	192 to 223
D	Not applicable	224 to 239
E	Not applicable	240 to 255

- The notation is informally referred to as *slash notation* and *formally as* **classless interdomain routing (CIDR, pronounced cider) strategy.**

Examples:
12.24.76.8/8
23.14.67.92/12
220.8.24.255/25

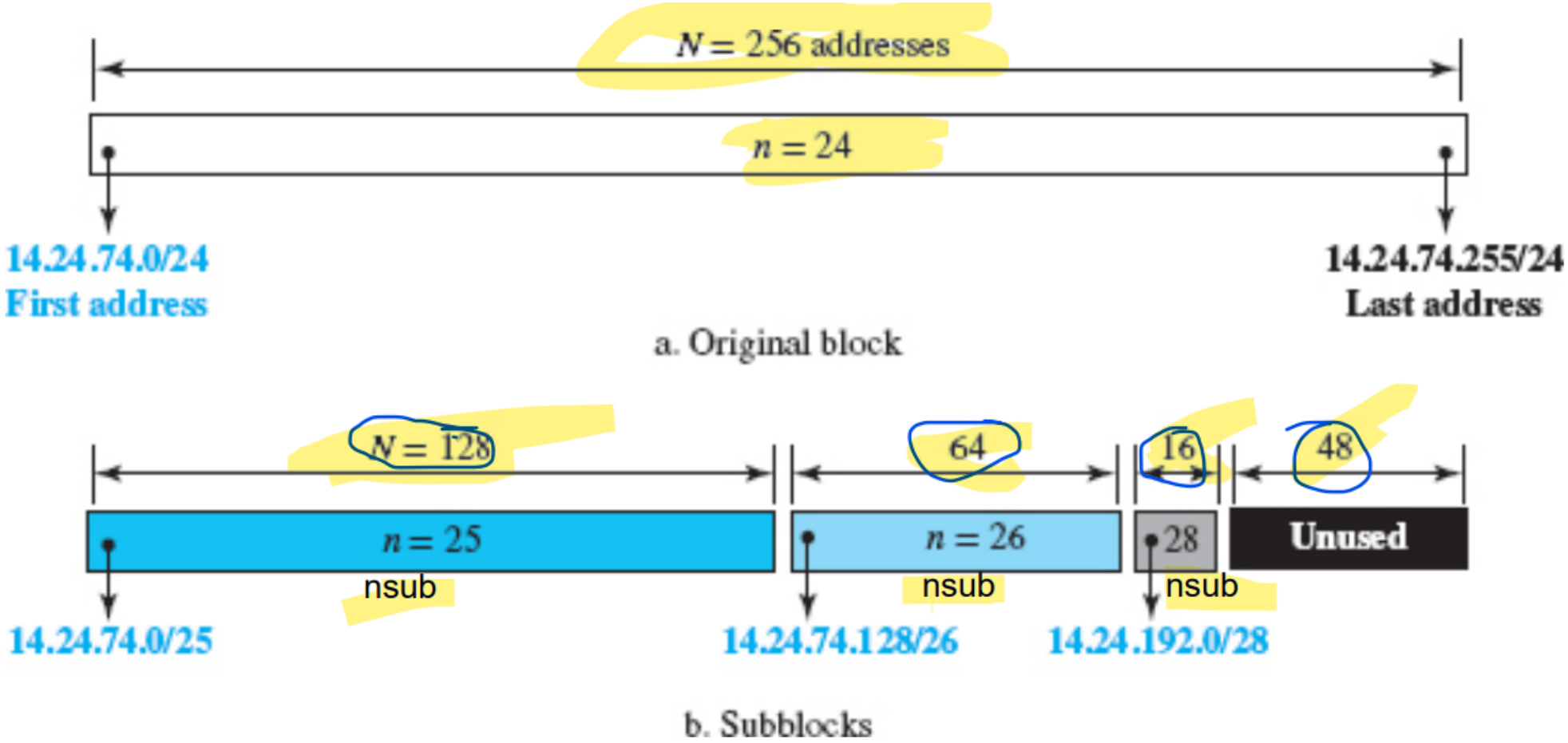
- the number of addresses
- the first address in the block -**keep the *n leftmost bits and set the (32 – n)***
- the last address- **keep the *n leftmost bits and set the (32 – n)*** rightmost bits all to 0s rightmost bits all to 1s

1. **The number of addresses in the block *N = NOT (Mask) + 1.***
2. **The first address in the block = (Any address in the block) AND (Mask).**
3. **The last address in the block = (Any address in the block) OR [(NOT (Mask)].**

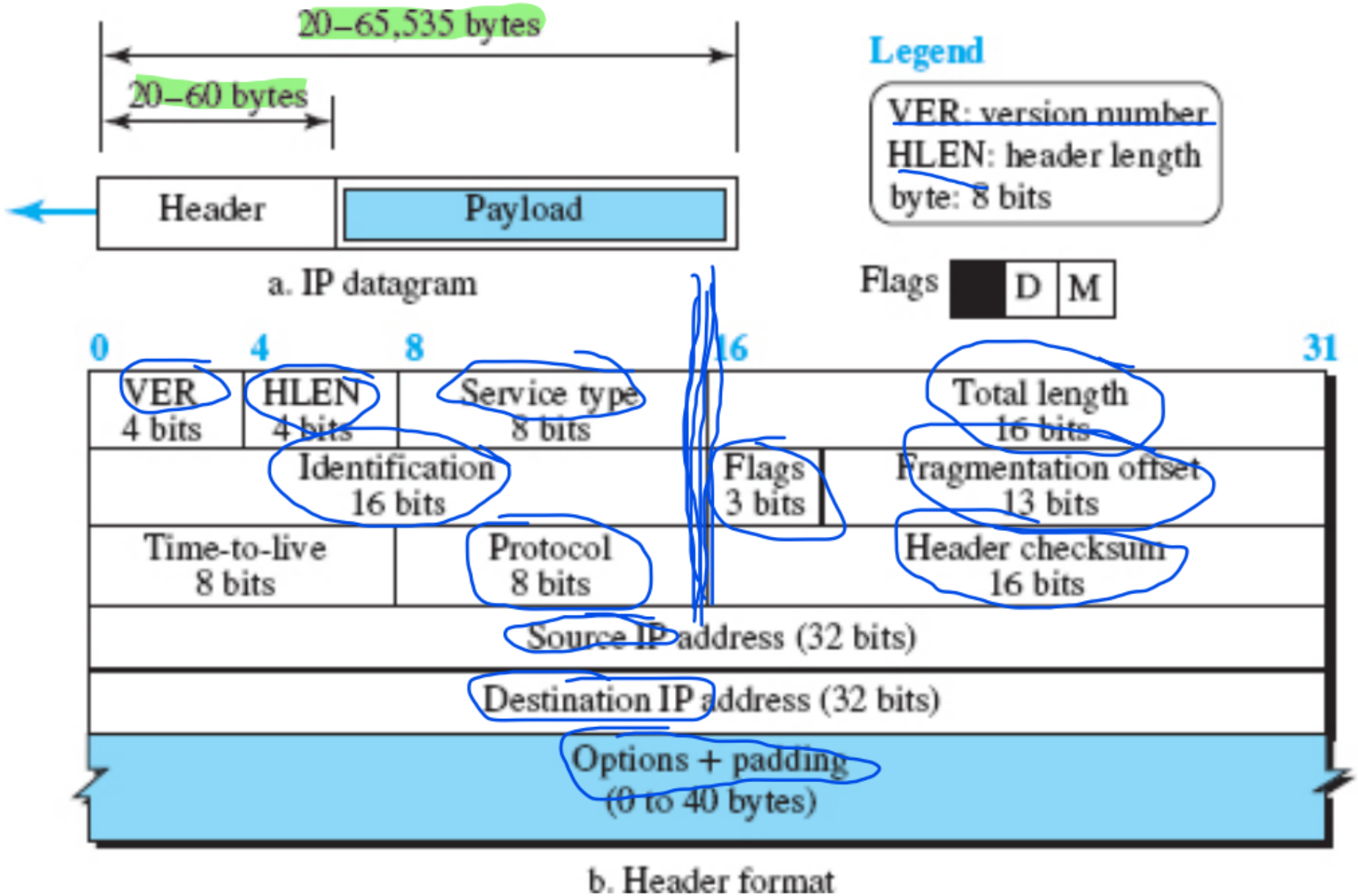
An organization is granted a block of addresses with the beginning address 14.24.74.0/24. The organization needs to have three subblocks of addresses to use in its three subnets: one subblock of 10 addresses, one subblock of 60 addresses, and one subblock of 120 addresses. Design the subblocks.

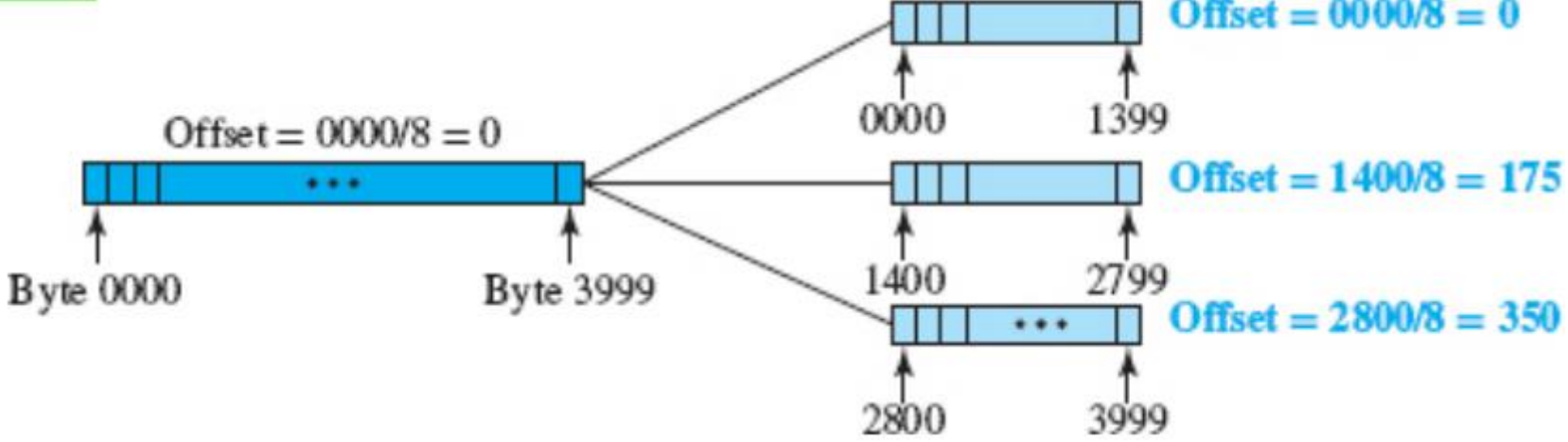
$n = 32 - \log_2 N$ n = prefix, N = no of blocks

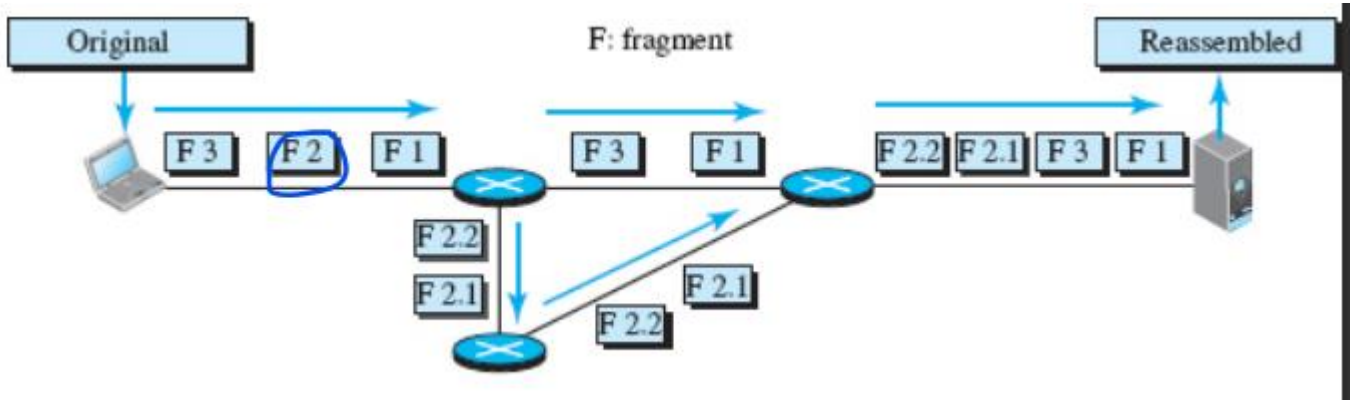
$nsub = 32 - \log_2 Nsub$



IPv4 Datagram Format



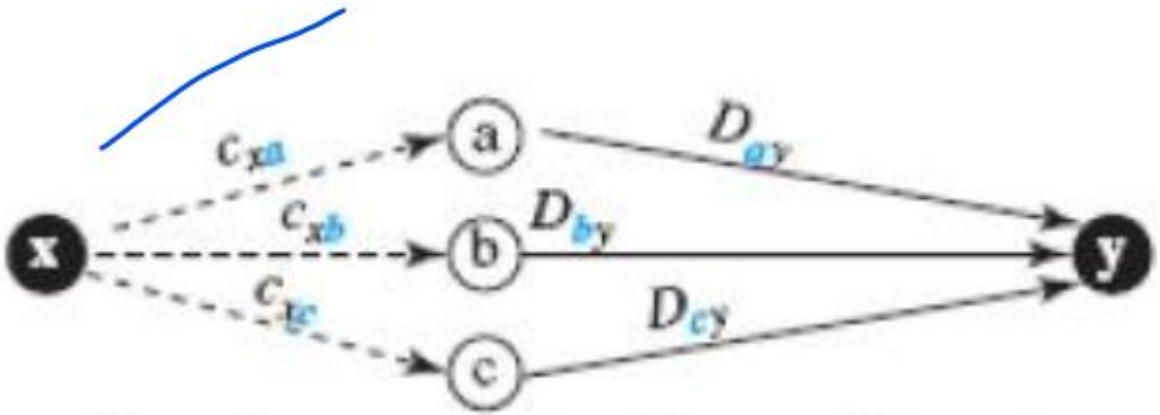

Offset = 0000/8 = 0
Byte 0000
Byte 3999
0000
1399
Offset = 0000/8 = 0
1400
2799
Offset = 1400/8 = 175
2800
3999
Offset = 2800/8 = 350


Original
F: fragment
Reassembled
F 3
F 2
F 1
F 3
F 1
F 2.2
F 2.1
F 3
F 1
F 2.2
F 2.1
F 2.1
F 2.2

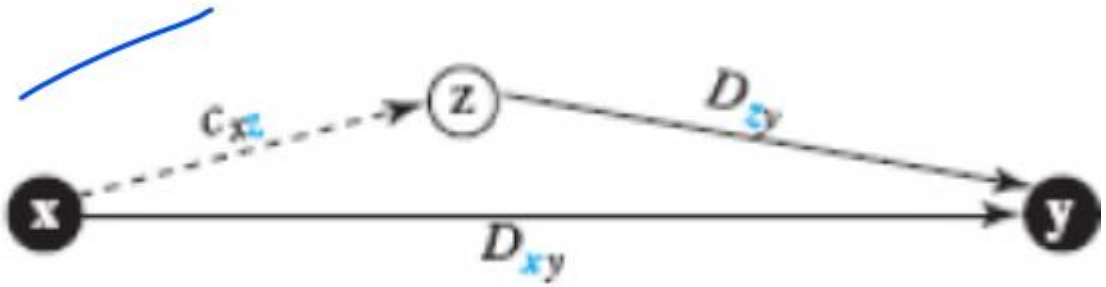
Bellman-Ford Equation

- $D_{xy} = \min \{(c_{xa} + D_{ay}), (c_{xb} + D_{by}), (c_{xc} + D_{cy}), \ldots\}$
- $D_{xy} = \min\{D_{xy}, (c_{xz} + D_{zy})\}$

- *Graphical idea behind Bellman-Ford equation*



a. General case with three intermediate nodes

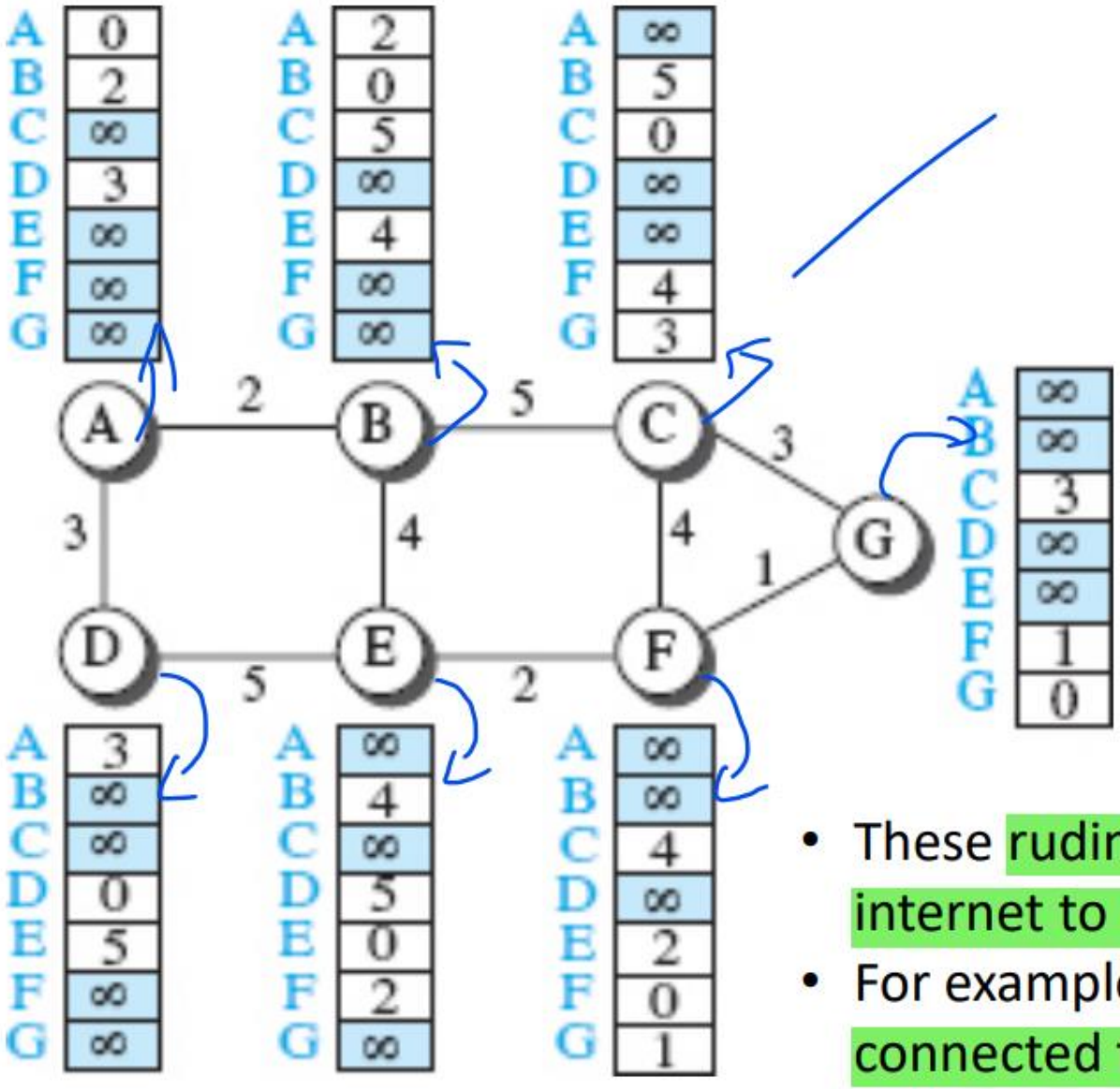


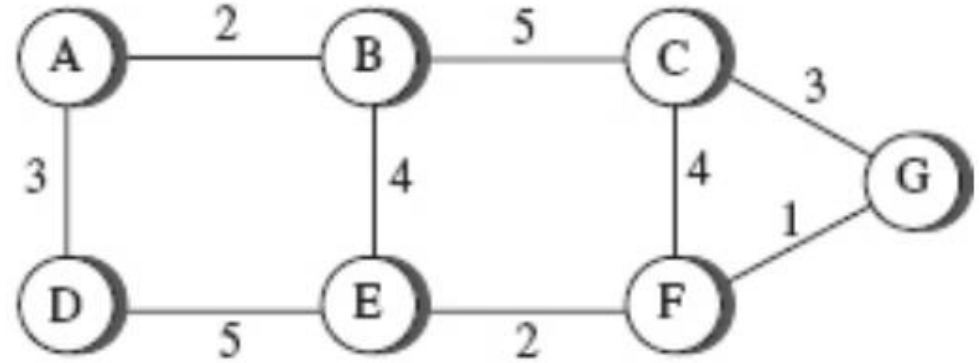
b. Updating a path with a new route

(a → y), (b → y), and (c → y) as previously established least-cost paths

(x → y) as the new least-cost path

First distance vector for an internet



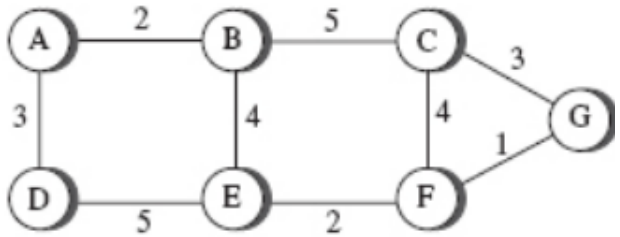


b. The weighted graph

- These rudimentary vectors cannot help the internet to effectively forward a packet.
- For example, node A thinks that it is not connected to node G because the corresponding cell shows the least cost of infinity.

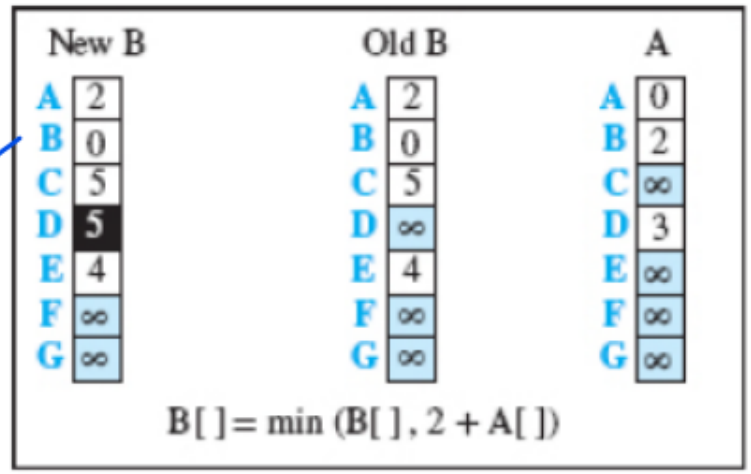
Updating distance vectors

- After each node has created its vector, it sends a copy of the vector to all its immediate neighbors
- After a node receives a distance vector from a neighbor, it updates its distance vector using the Bellman-Ford equation: $D_{xy} = min\{D_{xy}, (c_{xz} + D_{zy})\}$
- Two asynchronous events, happening one after another with some time in between.

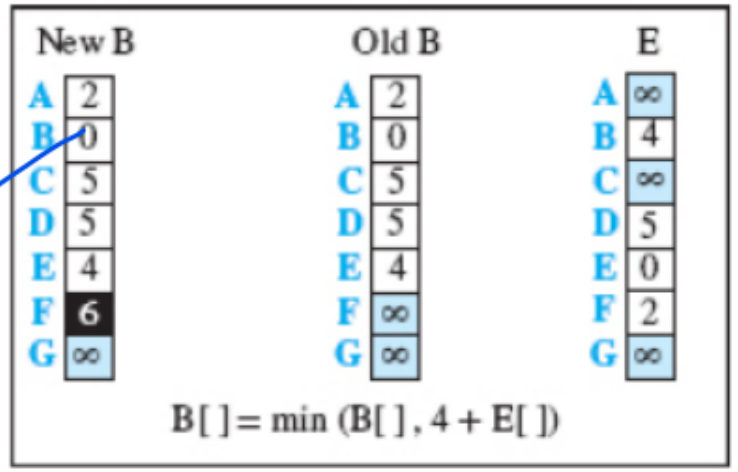


b. The weighted graph

X[]: the whole vector



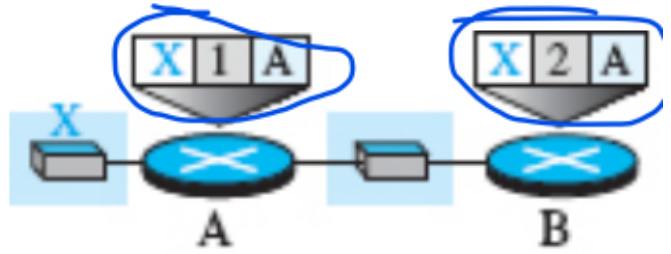
Node A has sent its vector to node B. Node B updates its vector using the cost $c_{BA} = 2$



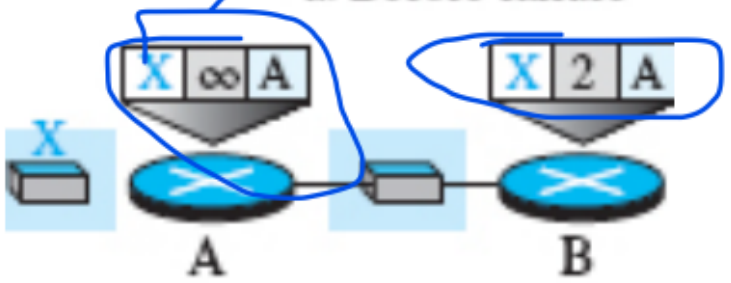
Node E has sent its vector to node B. Node B updates its vector using the cost $c_{EA} = 4$

Count to infinity - *Two-Node Loop*

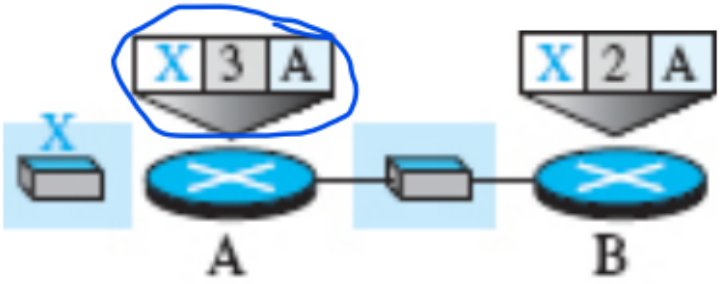
- *Both nodes A and B know how to reach node X*
- *Link between A and X fails. Node A changes its table.*
- *If A can send its table to B immediately, everything is fine.*
  - *However, the system becomes unstable if B sends its forwarding table to A before receiving A's forwarding table.*
  - *Node A receives the update and, assuming that B has found a way to reach X, immediately updates its forwarding table.*



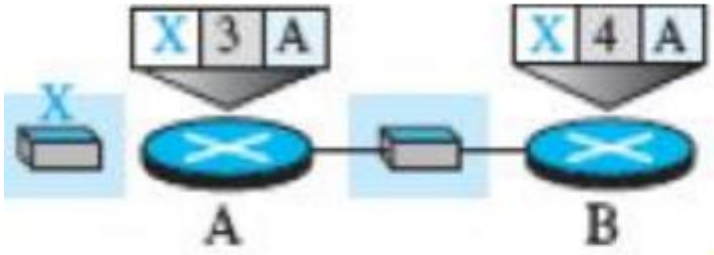
a. Before failure



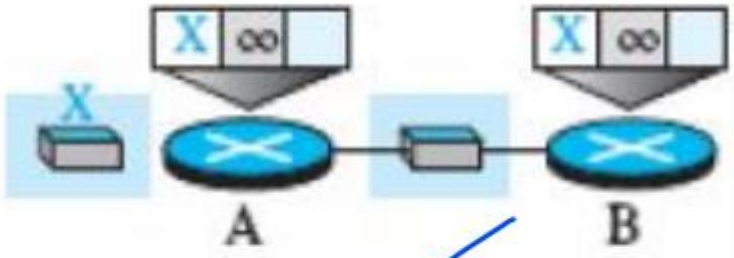
b. After link failure



c. After A is updated by B



d. After B is updated by A



e. Final result

- Now A sends its new update to B.
  - Now B thinks that something has been changed around A and updates its forwarding table.
  - The cost of reaching X increases gradually until it reaches infinity. At this moment, both A and B know that X cannot be reached.
- During this time the system is not stable.
  - Node A thinks that the route to X is via B; node B thinks that the route to X is via A.
  - If A receives a packet destined for X, the packet goes to B and then comes back to A.
  - Similarly, if B receives a packet destined for X, it goes to A and comes back to B.
  - Packets bounce between A and B, creating a two-node loop problem.

Solution to instability - *split horizon*

- Instead of flooding the table through each interface, each node sends only part of its table through each interface.
- If, according to its table, node B thinks that the optimum route to reach X is via A, it does not need to advertise this piece of information to A
  - the information has come from A (A already knows).
    - Taking information from node A, modifying it, and sending it back to node A is what creates the confusion.
  - Node A keeps the value of infinity as the distance to X - Later, when node A sends its forwarding table to B, node B also corrects its forwarding table.

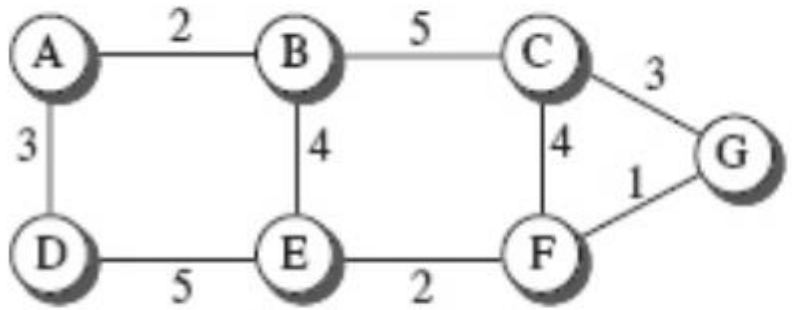
Poisoned Reverse

Split horizon problem:

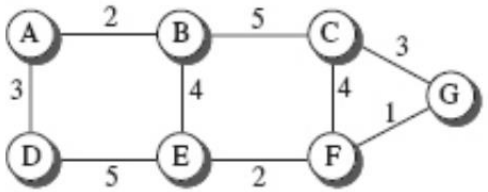
- Normally, the corresponding protocol uses a timer
- If there is no news about a route, the node deletes the route from its table
  - When node B in the previous scenario eliminates the route to X from its advertisement to A, node A cannot guess whether this is due to the split-horizon strategy (the source of information was A) or because B has not received any news about X recently
- Solution: poisoned reverse strategy
  - B can still advertise the value for X, but if the source of information is A, it can replace the distance with infinity as a warning: "Do not use this value; what I know about this route comes from you."

Link-State Routing

Formation of Least-Cost Trees



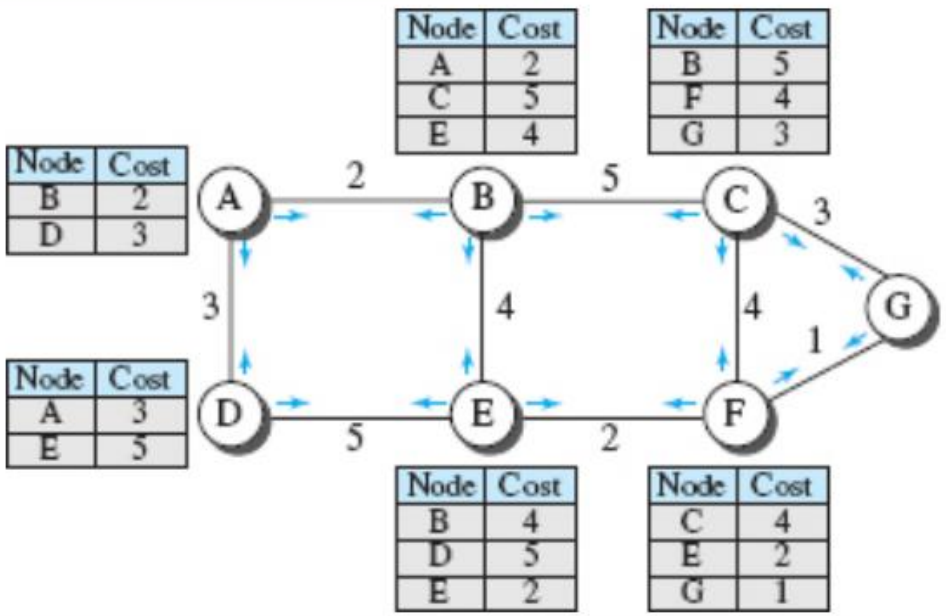
a. The weighted graph



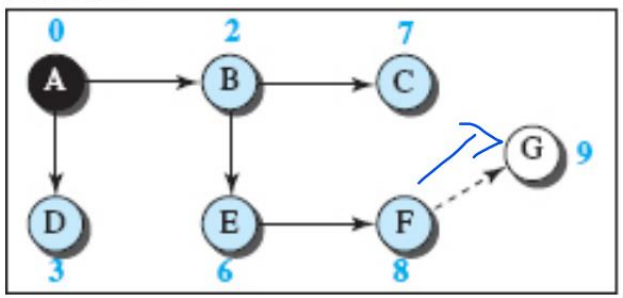
a. The weighted graph

	A	B	C	D	E	F	G
A	0	2	∞	3	∞	∞	∞
B	2	0	5	∞	4	∞	∞
C	∞	5	0	∞	∞	4	3
D	3	∞	∞	0	5	∞	∞
E	∞	4	∞	5	0	2	∞
F	∞	∞	4	∞	2	0	1
G	∞	∞	3	∞	∞	1	0

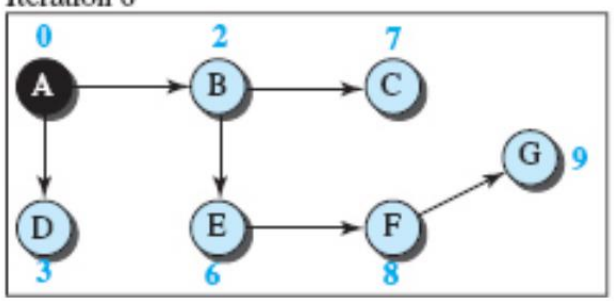
b. Link state database



Iteration 5



Iteration 6



Distance-vector routing algorithm	Link-state routing algorithm
Each router tells its neighbors what it knows about the whole internet	Each router tells the whole internet what it knows about its neighbors.

Routing Information Protocol (RIP)

- intradomain routing protocols based on the distance-vector routing algorithm
- RIP concept:
  - Hop count
  - Forwarding tables
  - Implementation – RIP messages, algorithm, timers
  - performance

Forwarding table for R1

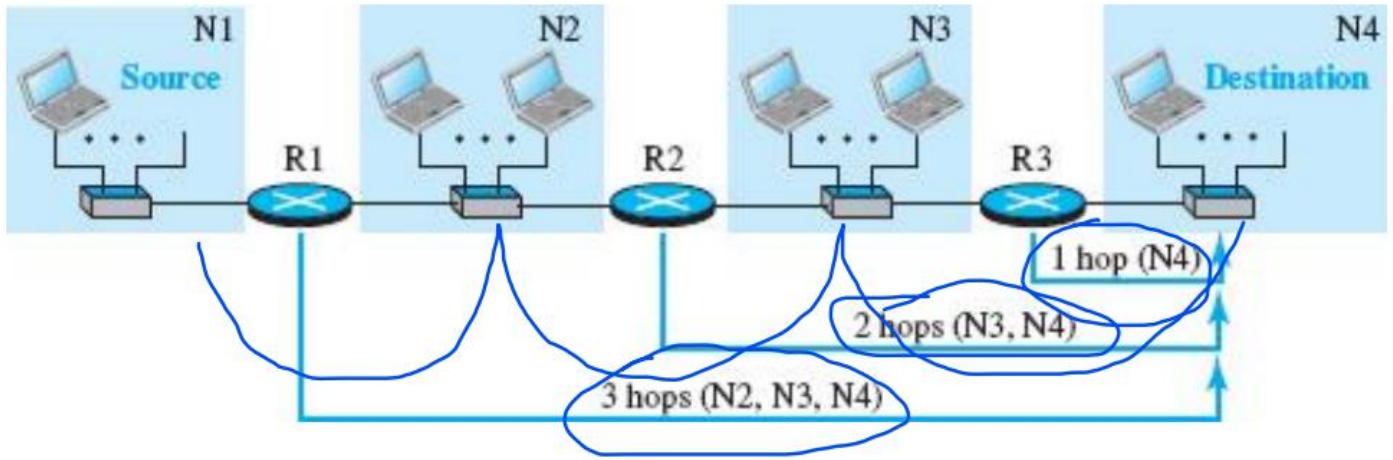
Destination network	Next router	Cost in hops
N1	——	1
N2	——	1
N3	R2	2
N4	R2	3

Forwarding table for R2

Destination network	Next router	Cost in hops
N1	R1	2
N2	——	1
N3	——	1
N4	R3	2

Forwarding table for R3

Destination network	Next router	Cost in hops
N1	R2	3
N2	R2	2
N3	——	1
N4	——	1



- In RIP, the maximum cost of a path can be 15, which means 16 is considered as infinity (no connection)

Open Shortest Path First (OSPF)

- **An intradomain routing protocol** like RIP, but it is based on the link-state routing protocol

OSPF concept:

- Metric
- Forwarding tables
- Areas
- Link state advertisement
- Implementation – OSPF messages, authentication, algorithm
- Performance

- If we use the hop count for OSPF, the tables will be exactly the same as RIP because both protocols use the shortest-path trees to define the best route from a source to a destination.

Forwarding table for R1

Destination network	Next router	Cost
N1	——	4
N2	——	5
N3	R2	8
N4	R2	12

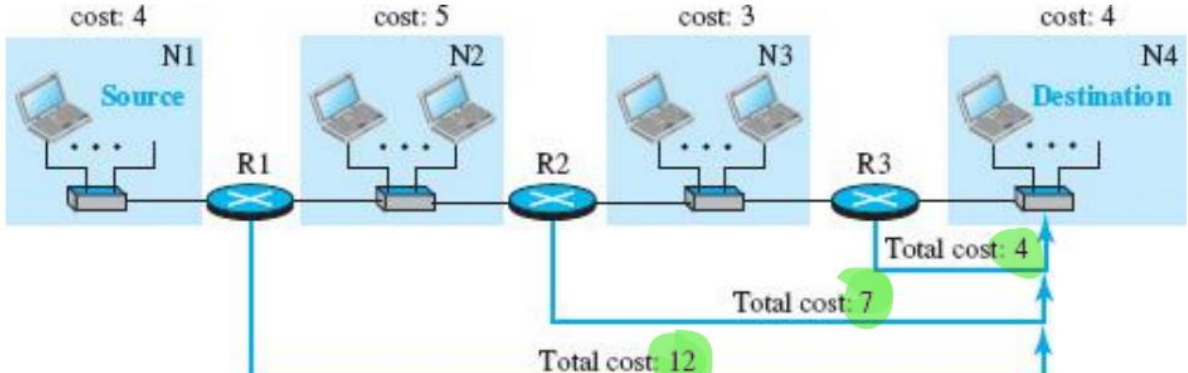
Forwarding table for R2

Destination network	Next router	Cost
N1	R1	9
N2	——	5
N3	——	3
N4	R3	7

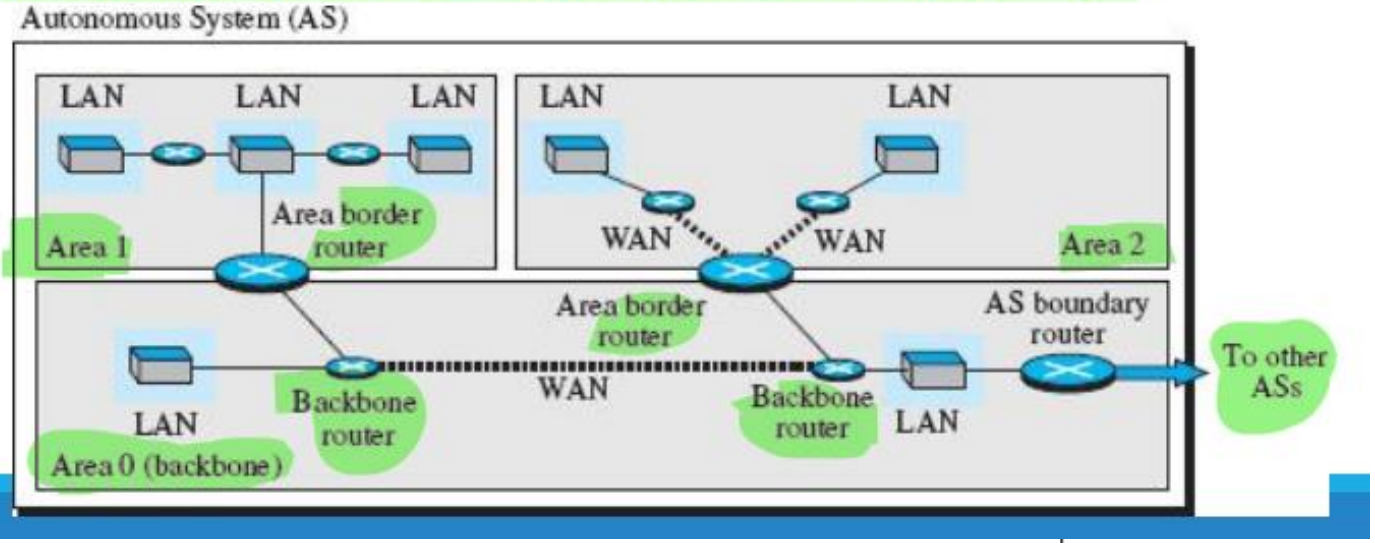
Forwarding table for R3

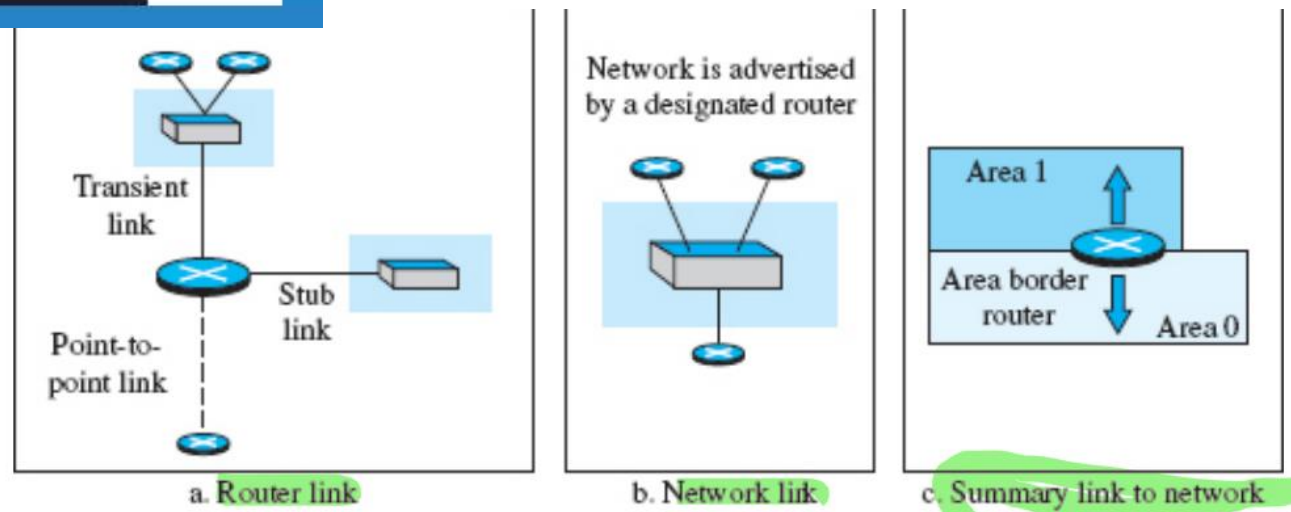
Destination network	Next router	Cost
N1	R2	12
N2	R2	8
N3	——	3
N4	——	4

- Each link (network) can be assigned a weight based on the throughput, round-trip time, reliability, and so on.
  - An administration can also decide to use the hop count as the cost
- An interesting point about the cost in OSPF is that different service types (TOSs) can have different weights as the cost.



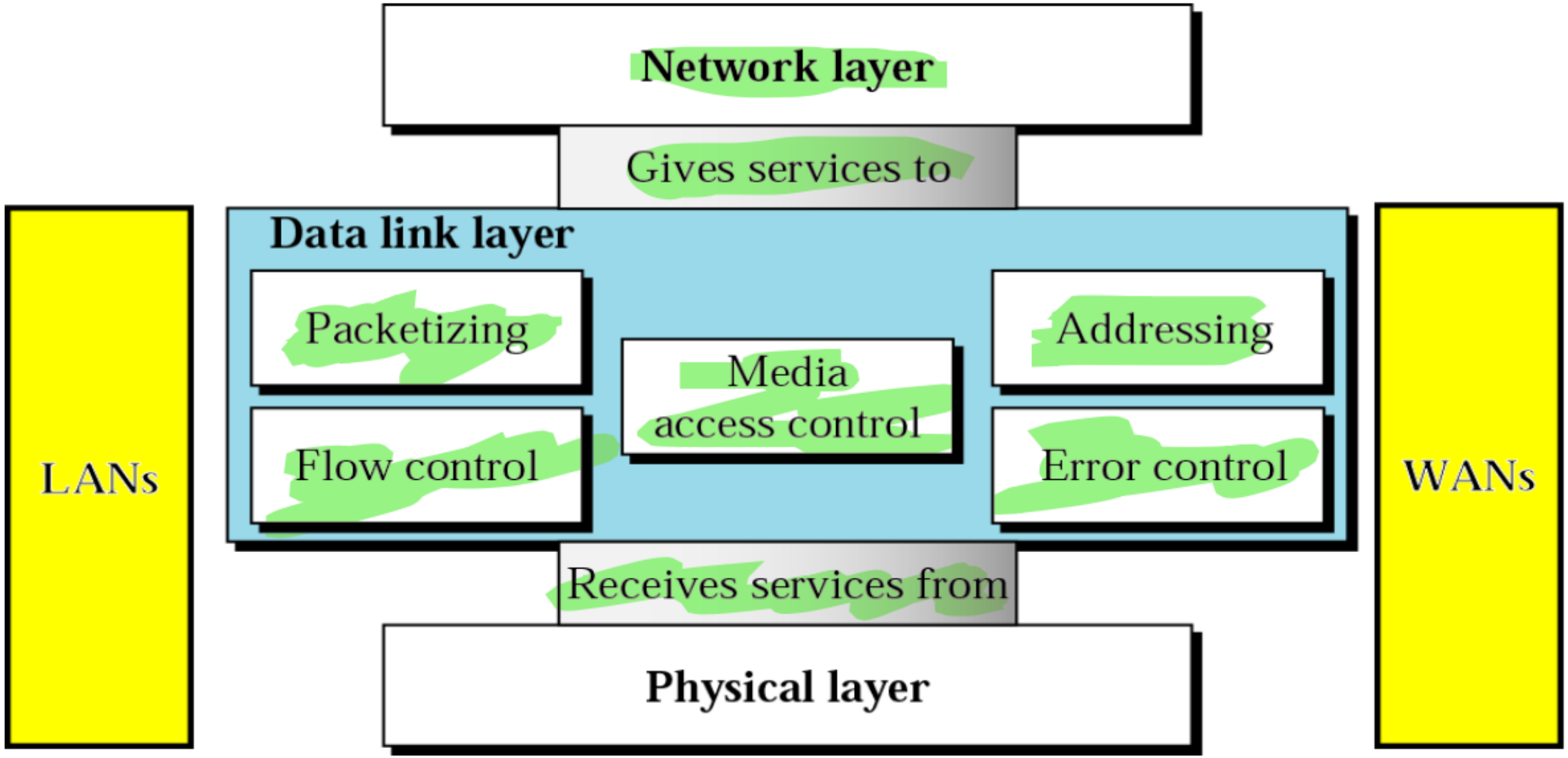
a small independent domain for flooding LSPs.





Unit 4

Data Link Layer - functions



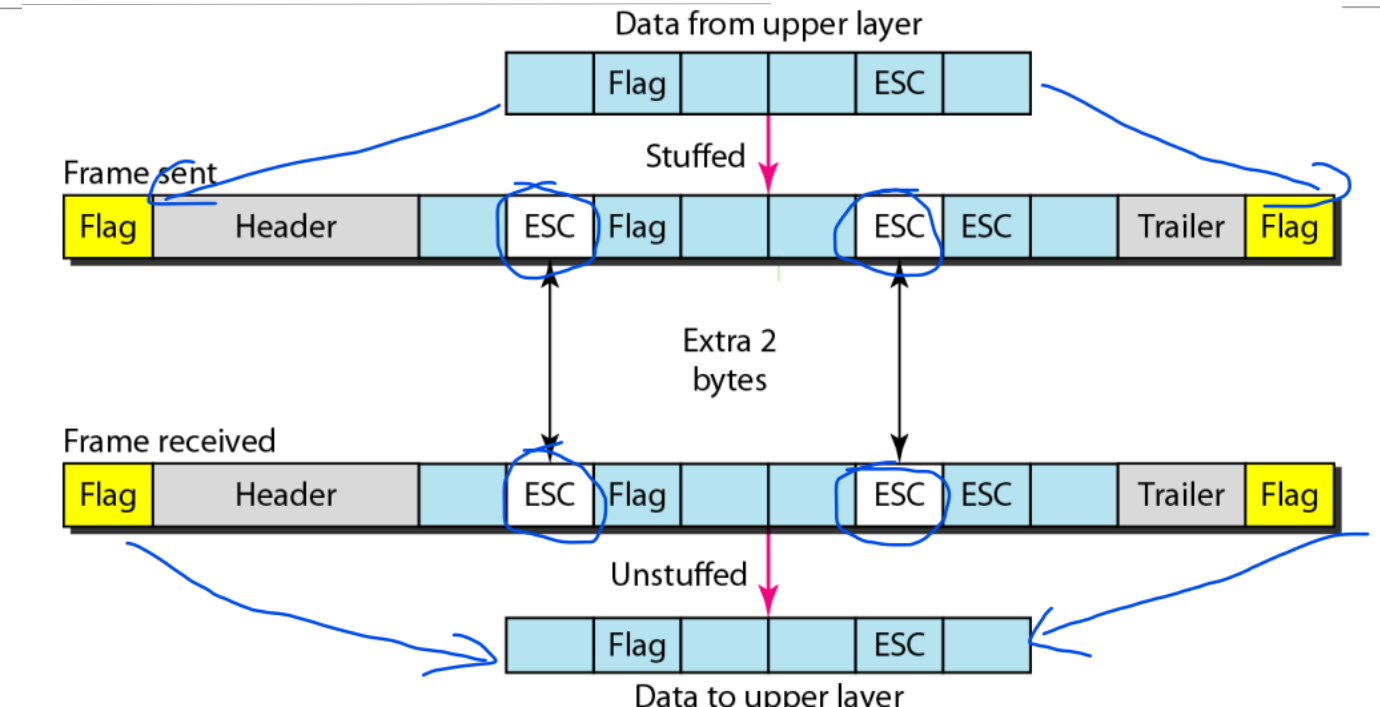
Fixed-Size Framing

- no need for defining the boundaries of the frames
- the size itself can be used as a delimiter

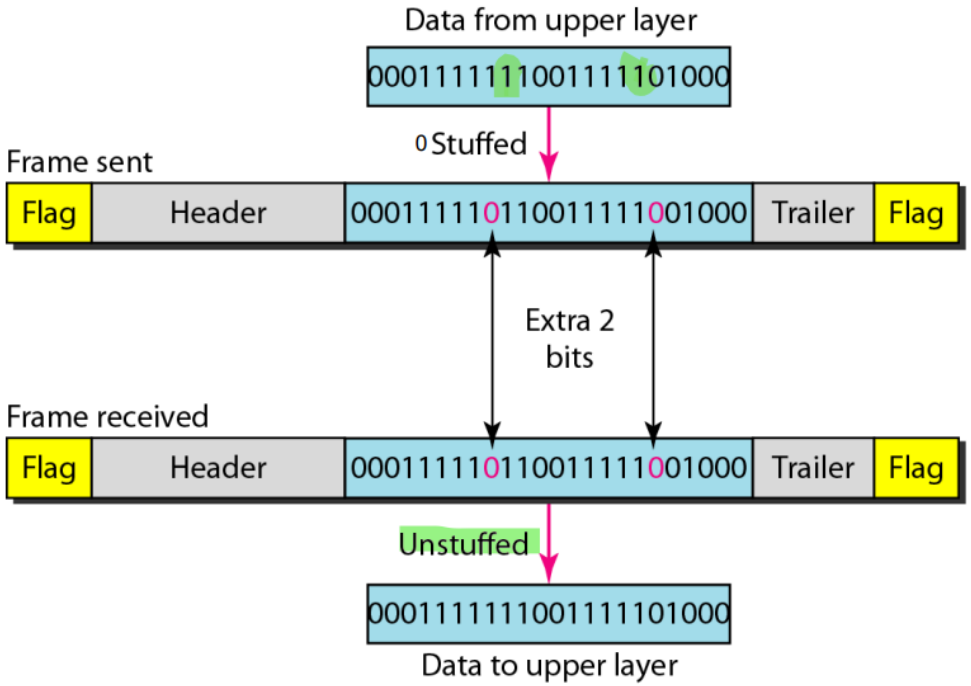
Variable-Size Framing -

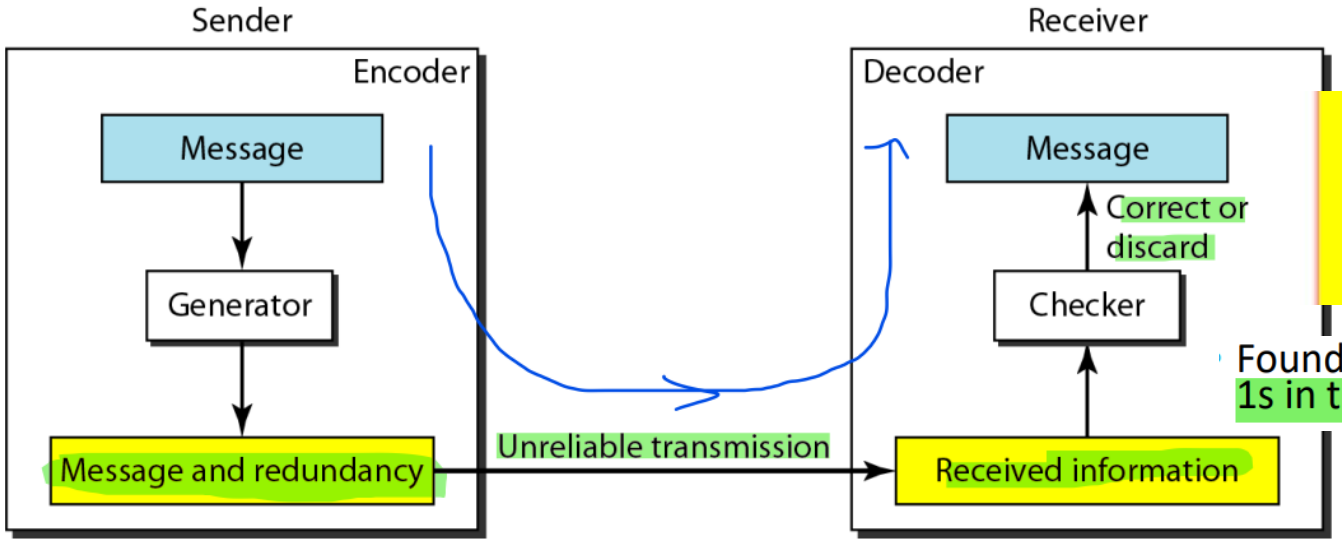
- need a way to define the end of one frame and the beginning of the next
- Historically, two approaches were used for this purpose:
  - a character-oriented *(or byte-oriented)* approach
  - a bit-oriented approach

Character-Oriented Framing

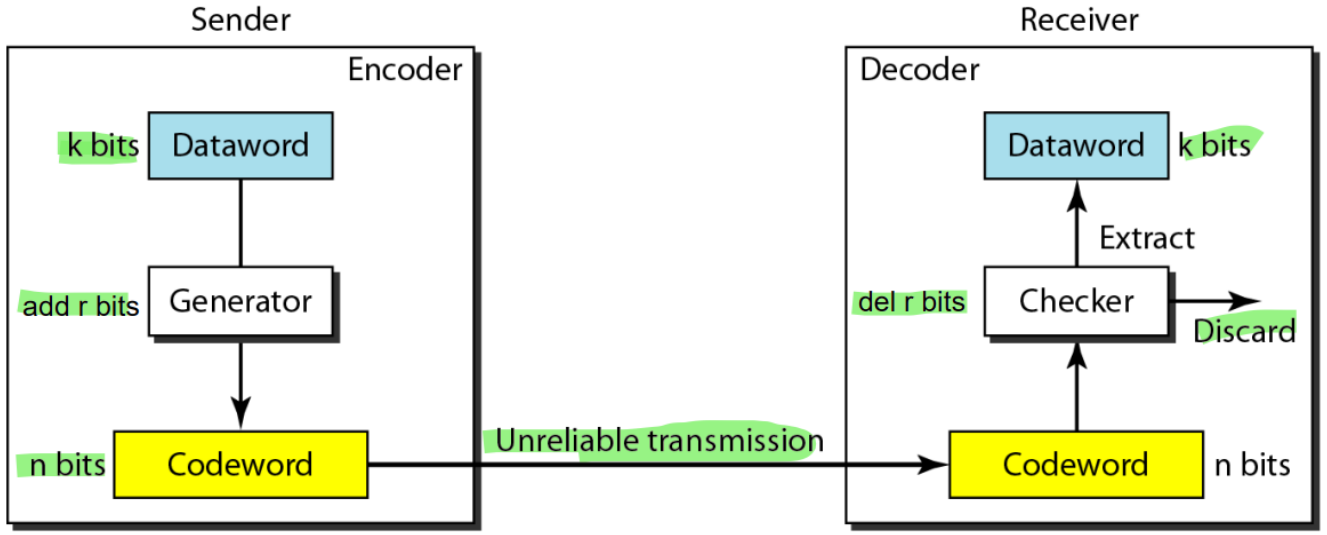


Bit-Oriented Framing





Error Detection in Block Coding



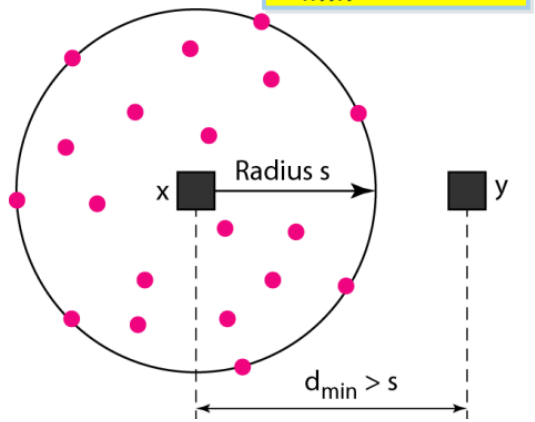
Minimum Hamming Distance

The minimum Hamming distance is the smallest Hamming distance between all possible pairs in a set of codewords.

Found by XOR operation ($\oplus$) on the two words and count the number of 1s in the result

minimum Hamming Distance = 3

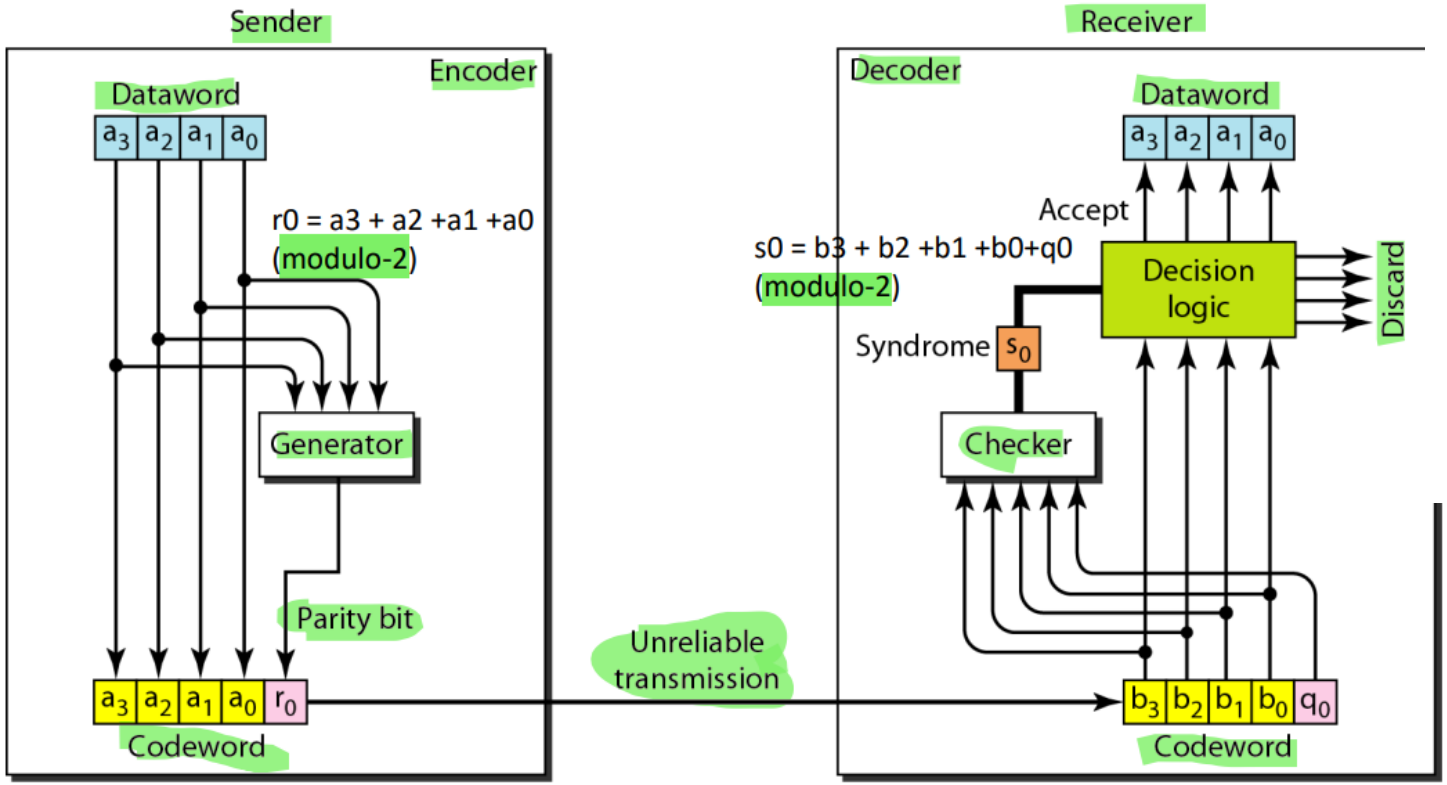
$$d_{min} = s + 1$$



00000
01011
10101
11110

3
4
3

Parity-Check: Encoding/Decoding



Rule to make total number of 1s in the codeword is even:
If the number of 1s is even, r0=0 else r0=1

Rule of the decision logic analyzer : If s0=0, no detectable error in the received codeword (accept)
else detectable error (discard)

Example: *Parity Check*

Suppose the sender wants to send the word ***world***. In ASCII the five characters are coded (with ***even parity***) as

1110111 1101111 1110010 1101100 1100100

The following shows the actual bits sent

1110111**0** 1101111**0** 1110010**0** 1101100**0** 1100100**1**

Receiver receives this sequence of words:

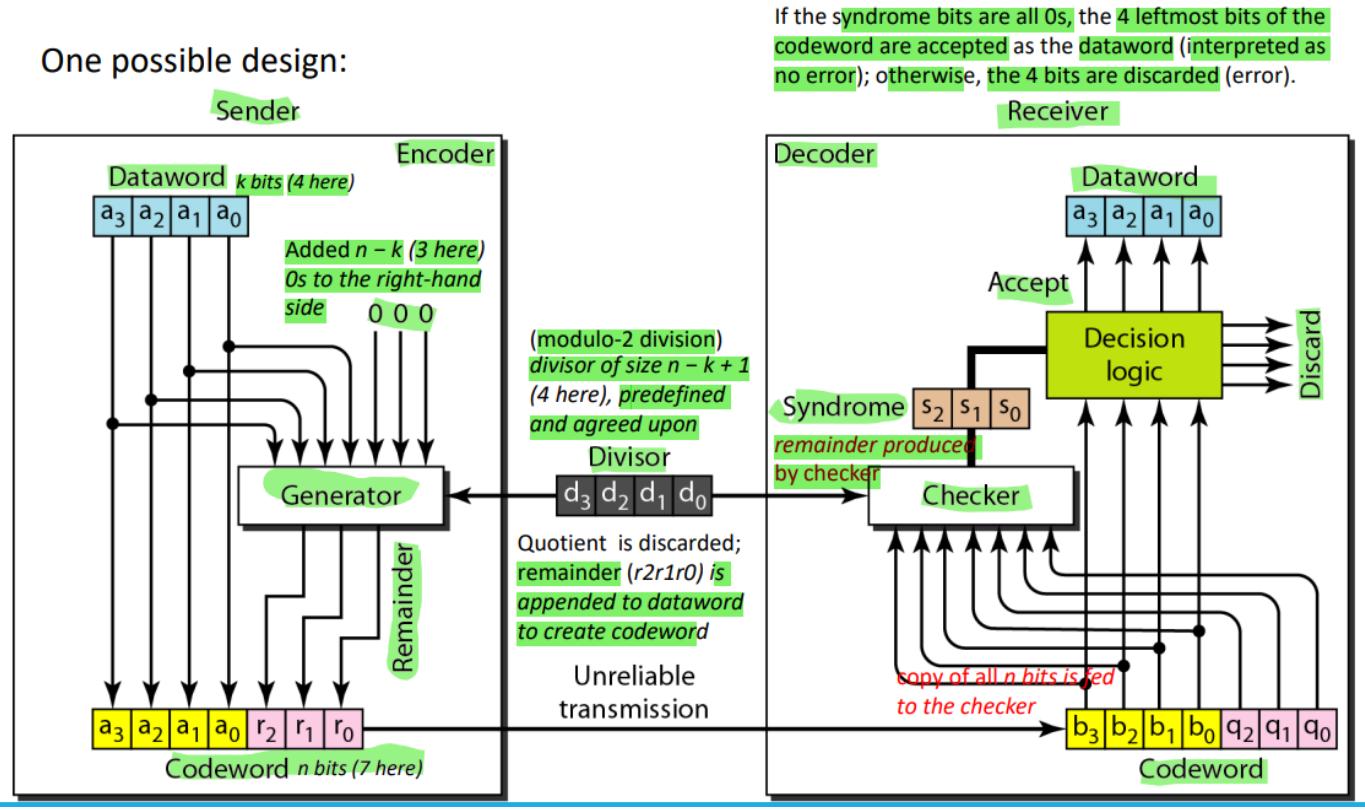
11111110 11011110 11101100 11011000 11001001

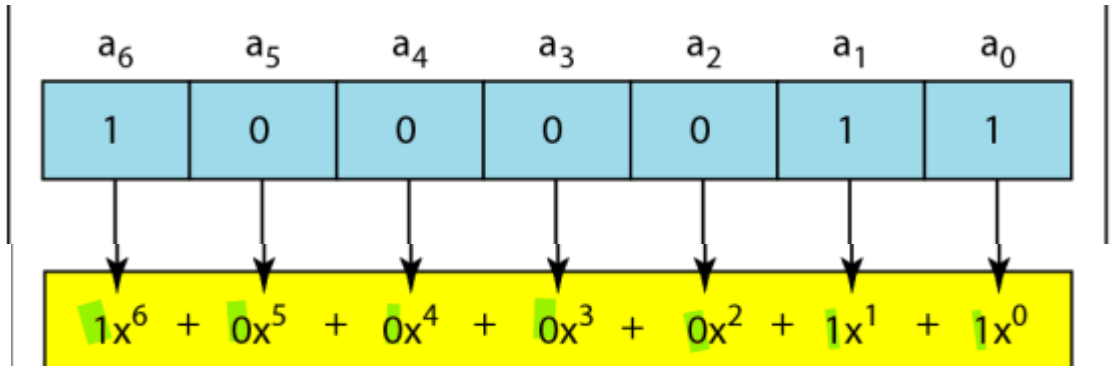
Which blocks are accepted? Which are rejected?

11111110 11011110 11101100 11011000 11001001

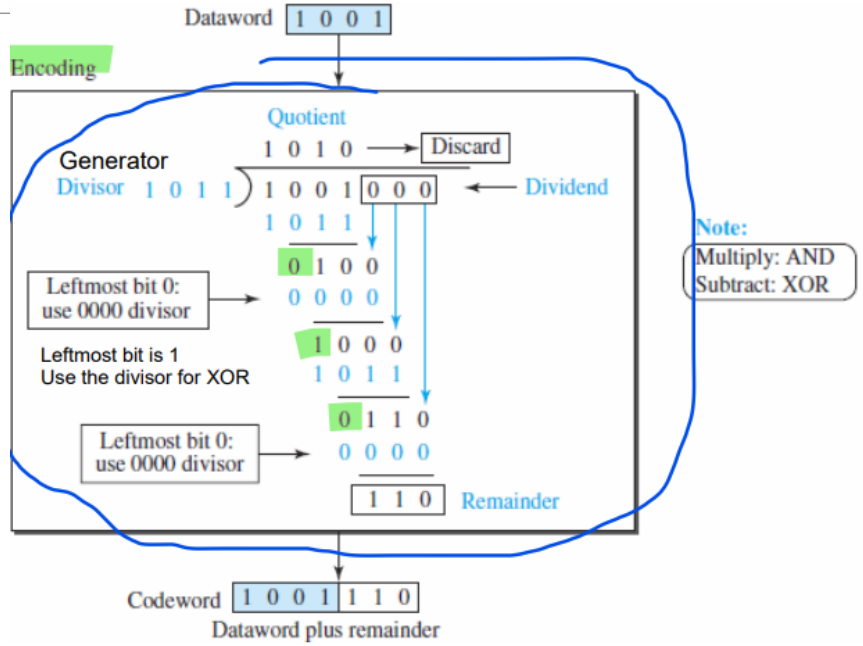
CRC Encoder/Decoder

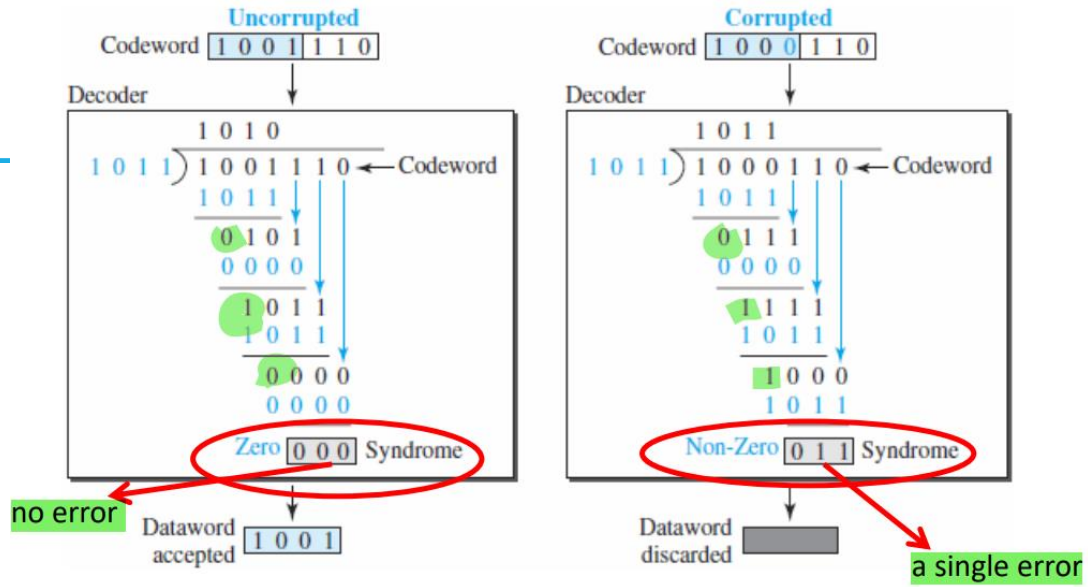
One possible design:





Division in CRC encoder:

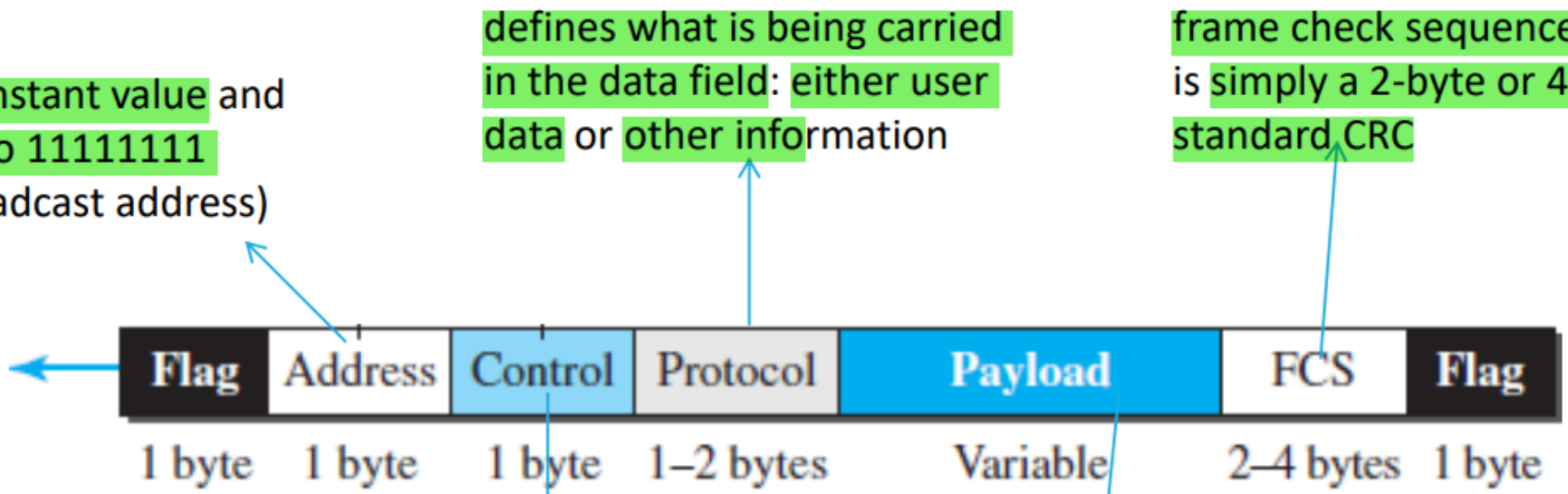




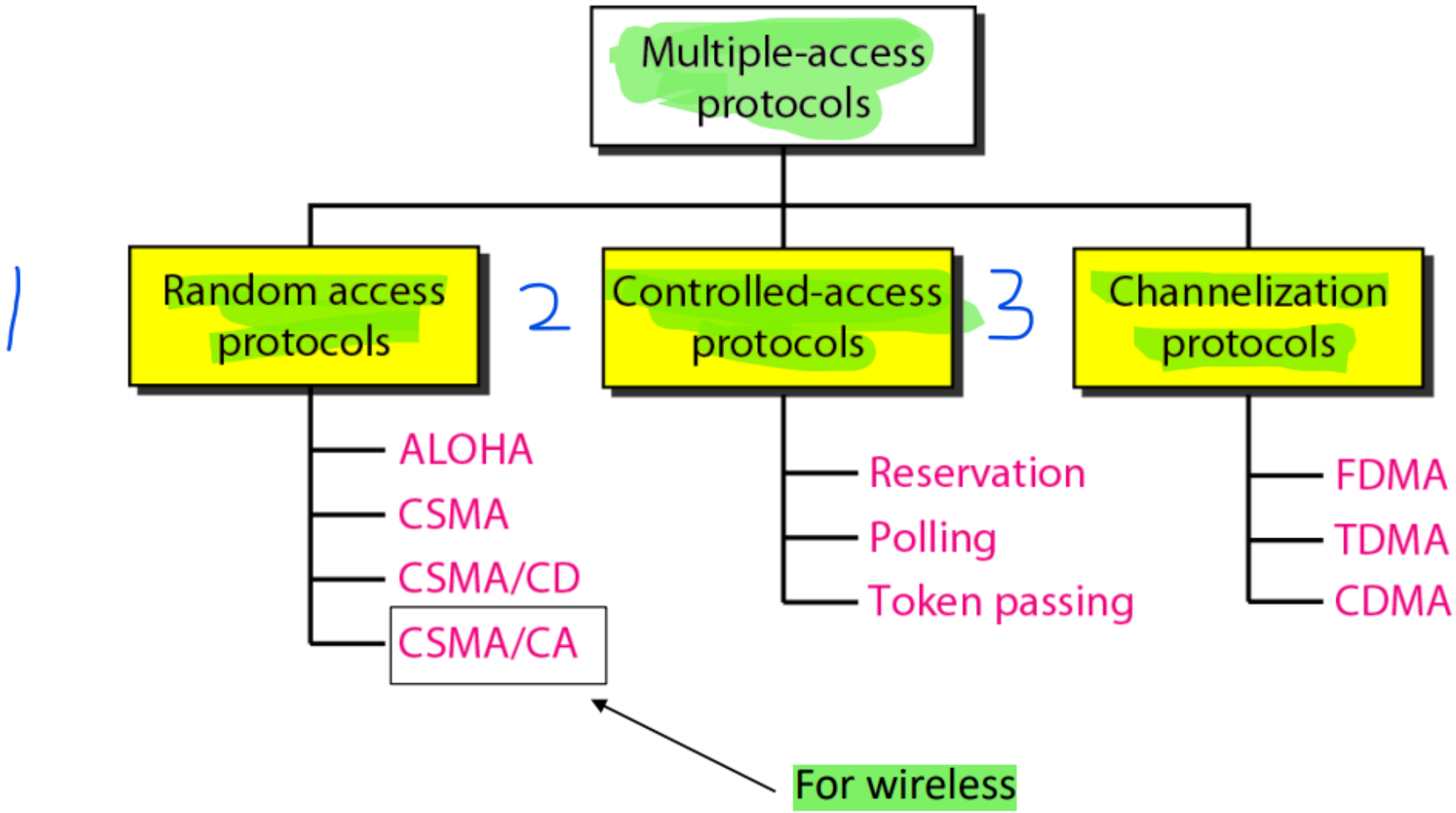
Remainder of the division is the syndrome

PPP frame format

- a character-oriented (or byte-oriented) frame
- starts and ends with a 1-byte flag with the bit pattern 01111110



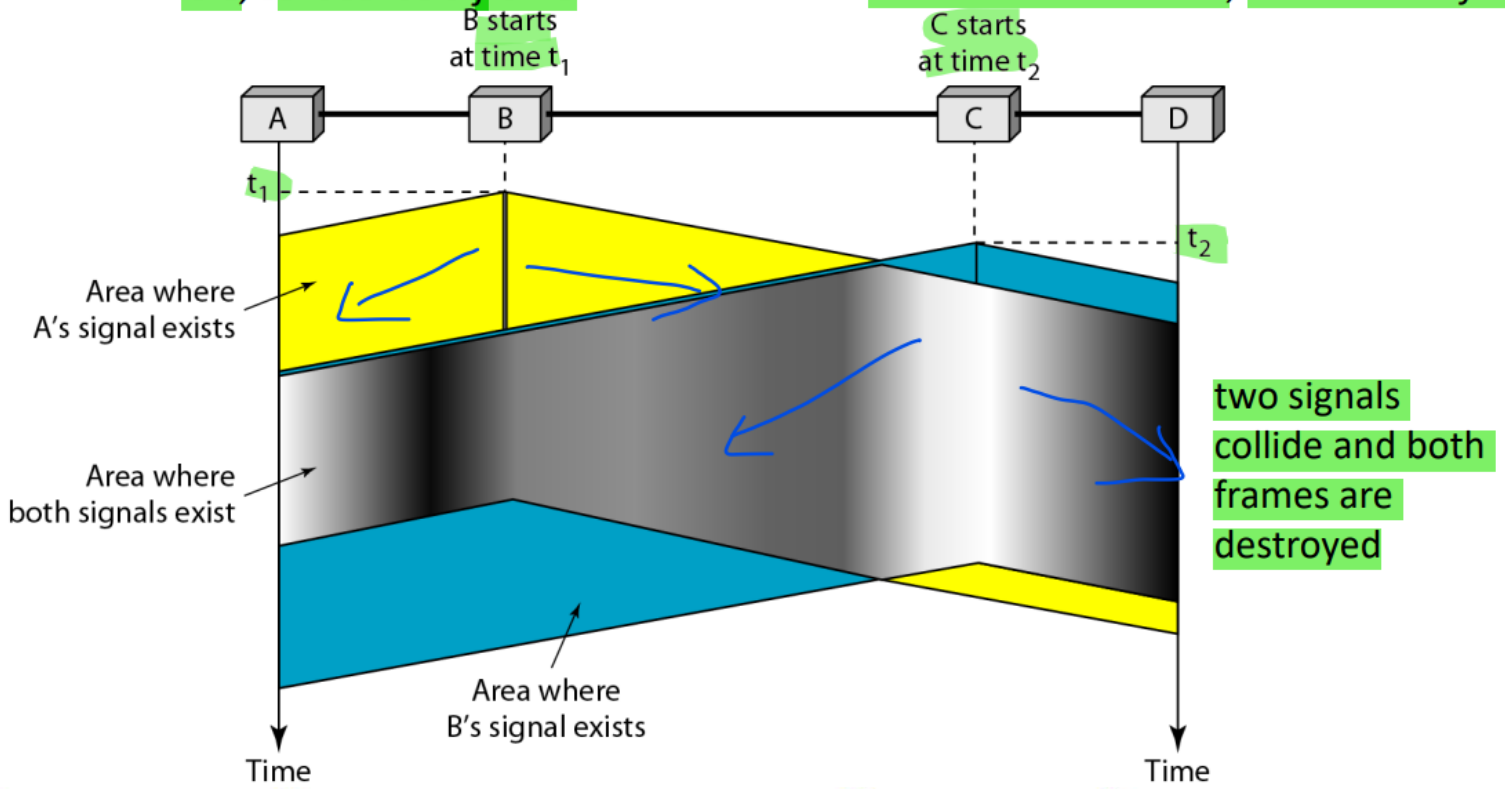
Taxonomy of multiple-access protocols



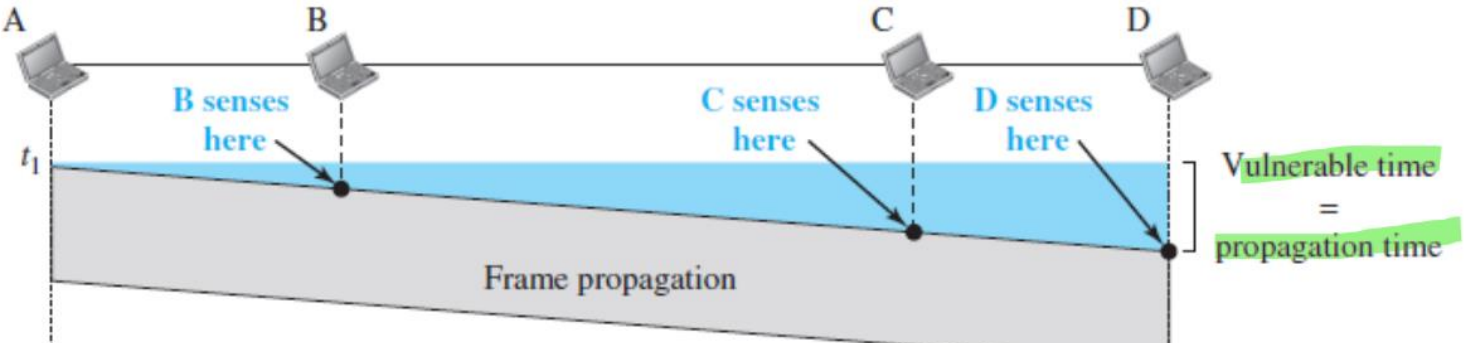
CSMA - space and time model

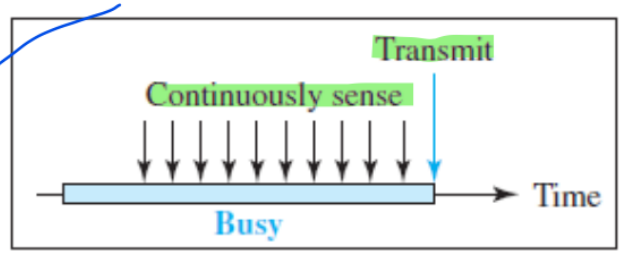
station B senses the medium and finds it idle, so it sends a frame

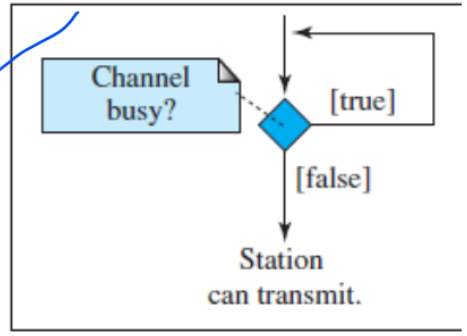
station C senses the medium and finds it idle as the first bits from station B have not reached station C, *so it sends a frame*

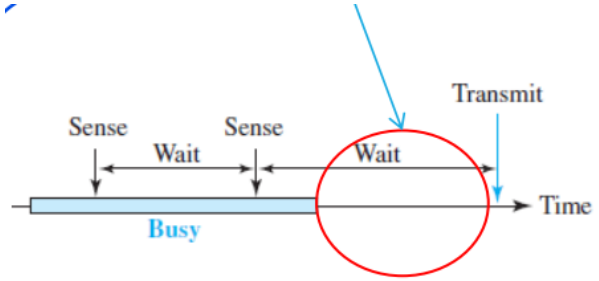


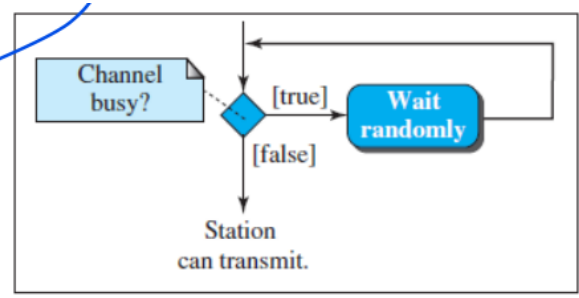
two signals collide and both frames are destroyed



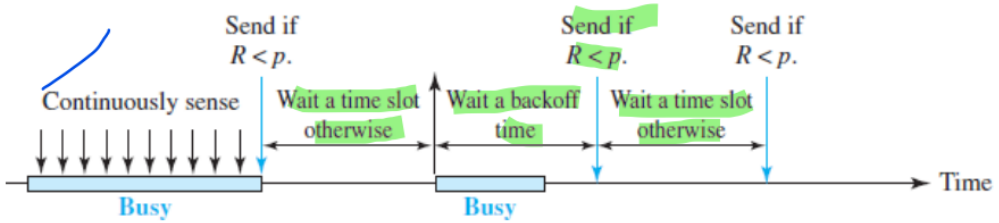


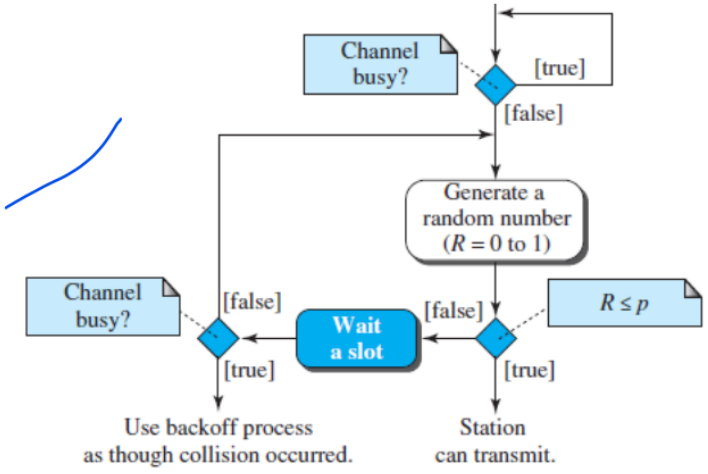




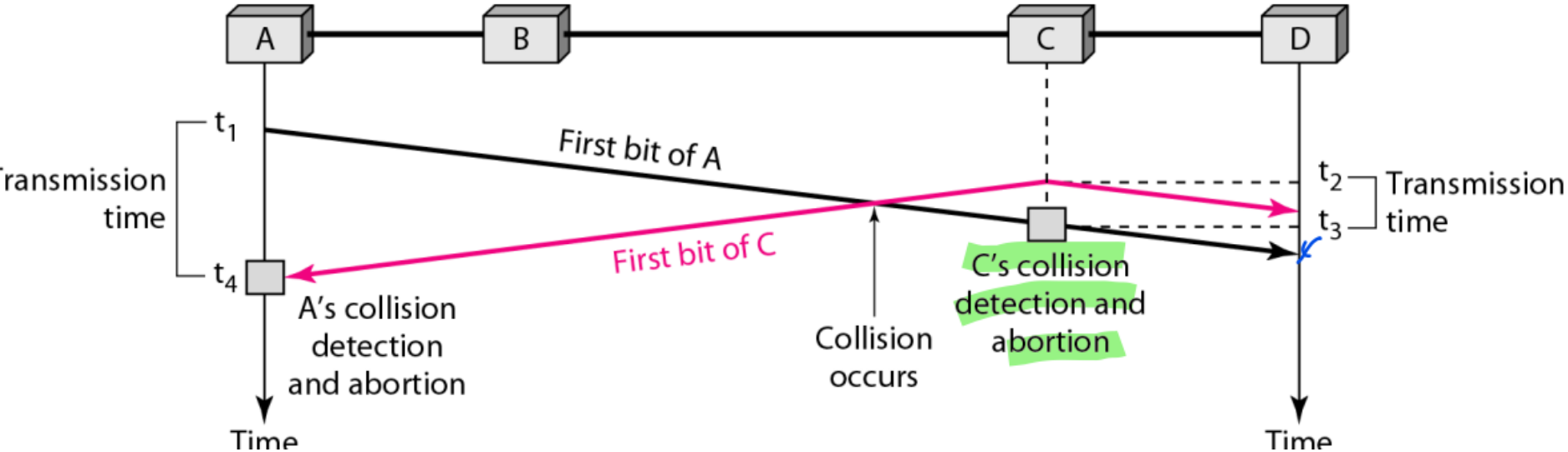


CSMA - p-*Persistent*



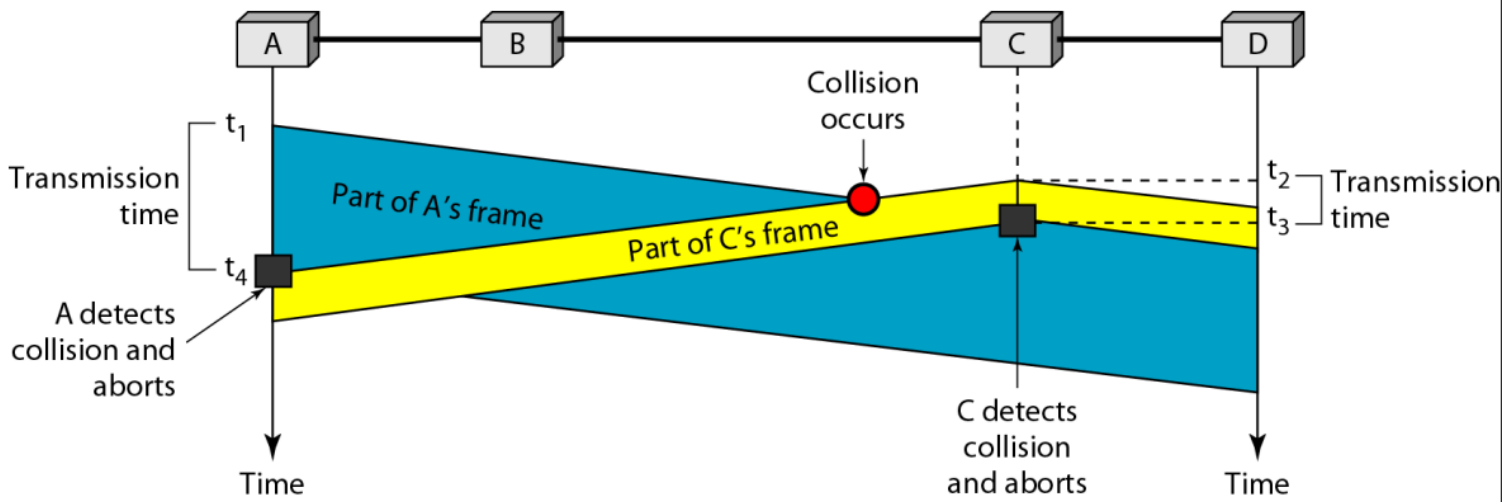


Carrier sense multiple access w collision detection (CSMA/CD)

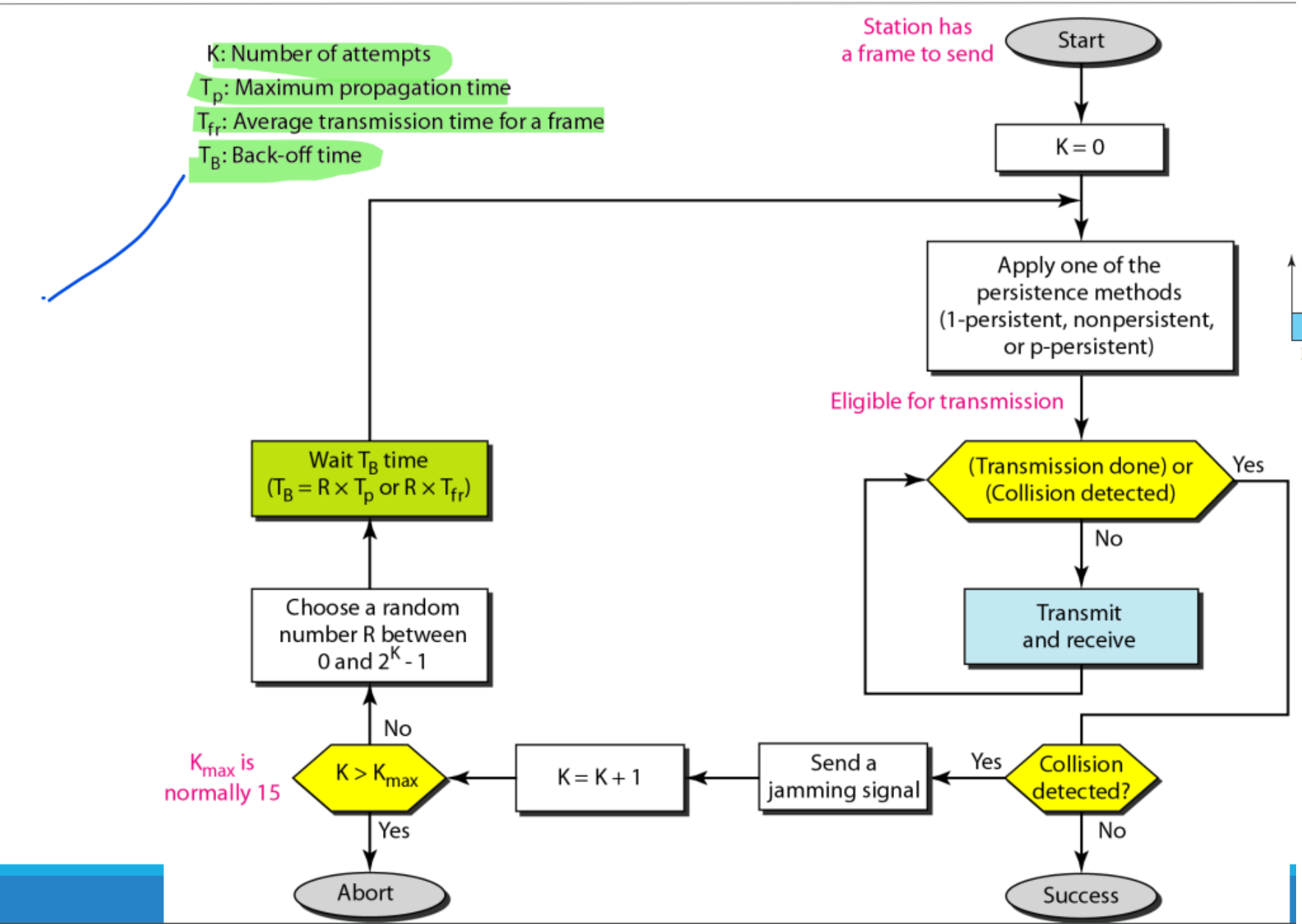


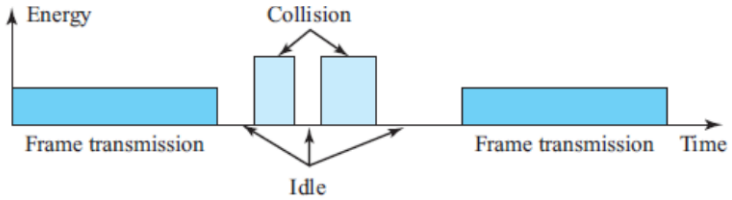
Tfr = 2*Tp

Mini Frame Size = Bandwidth (Mbps) * Tfr



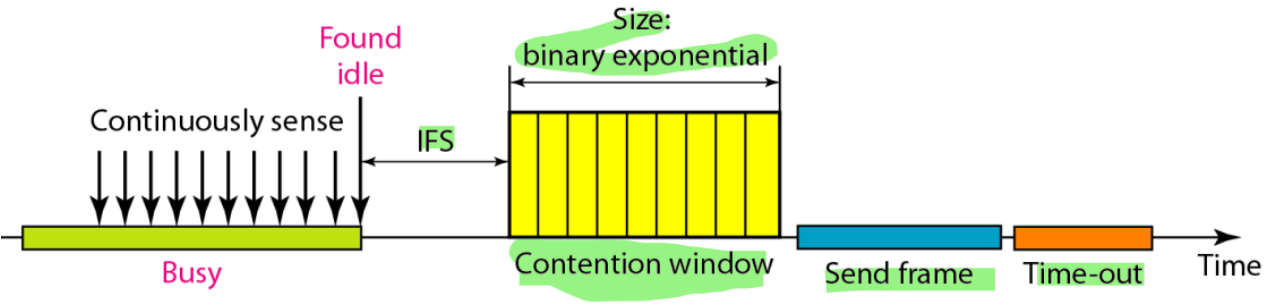
Flow diagram for CSMA/CD

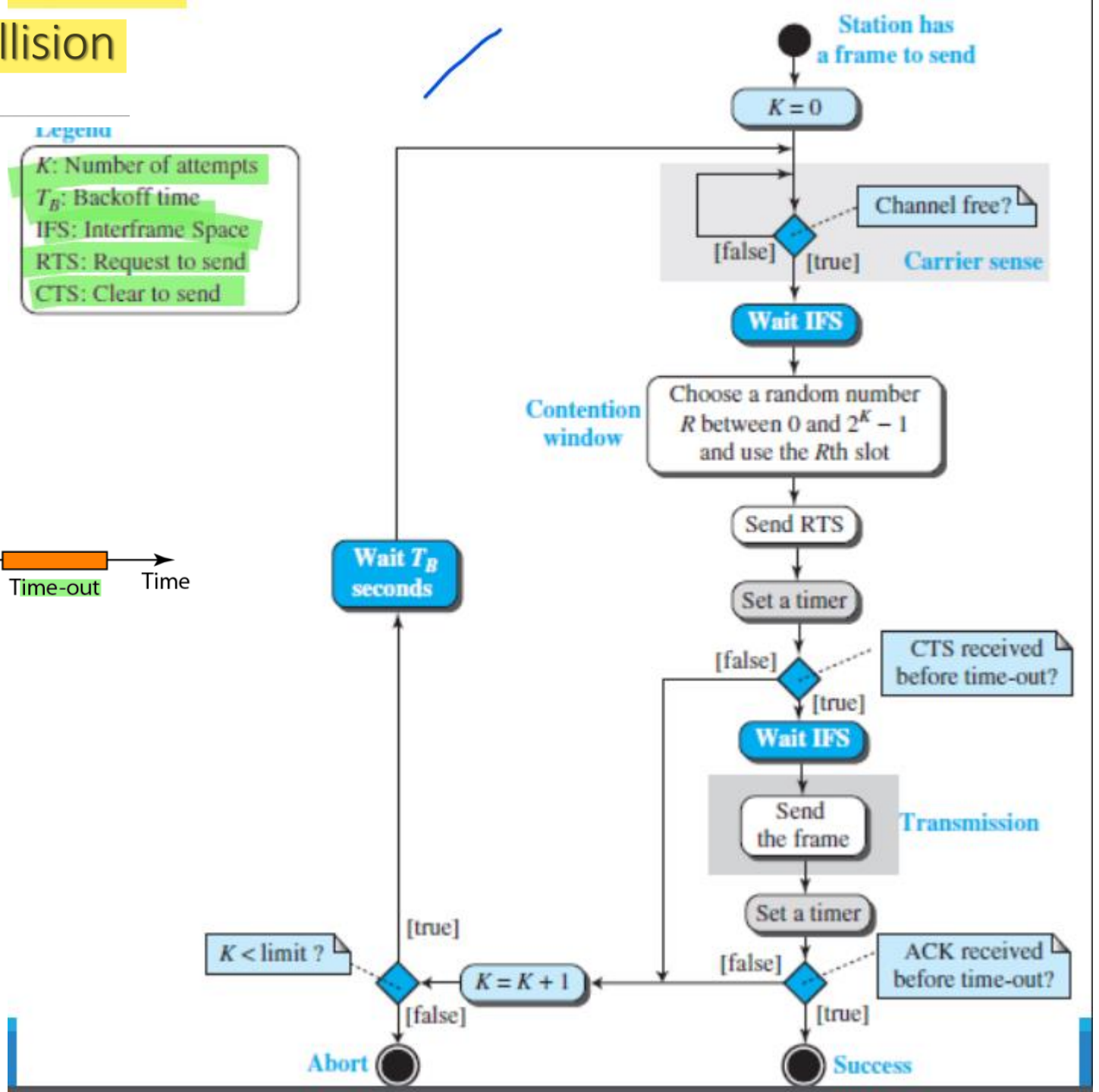




Carrier sense multiple access with collision avoidance - CSMA/CA

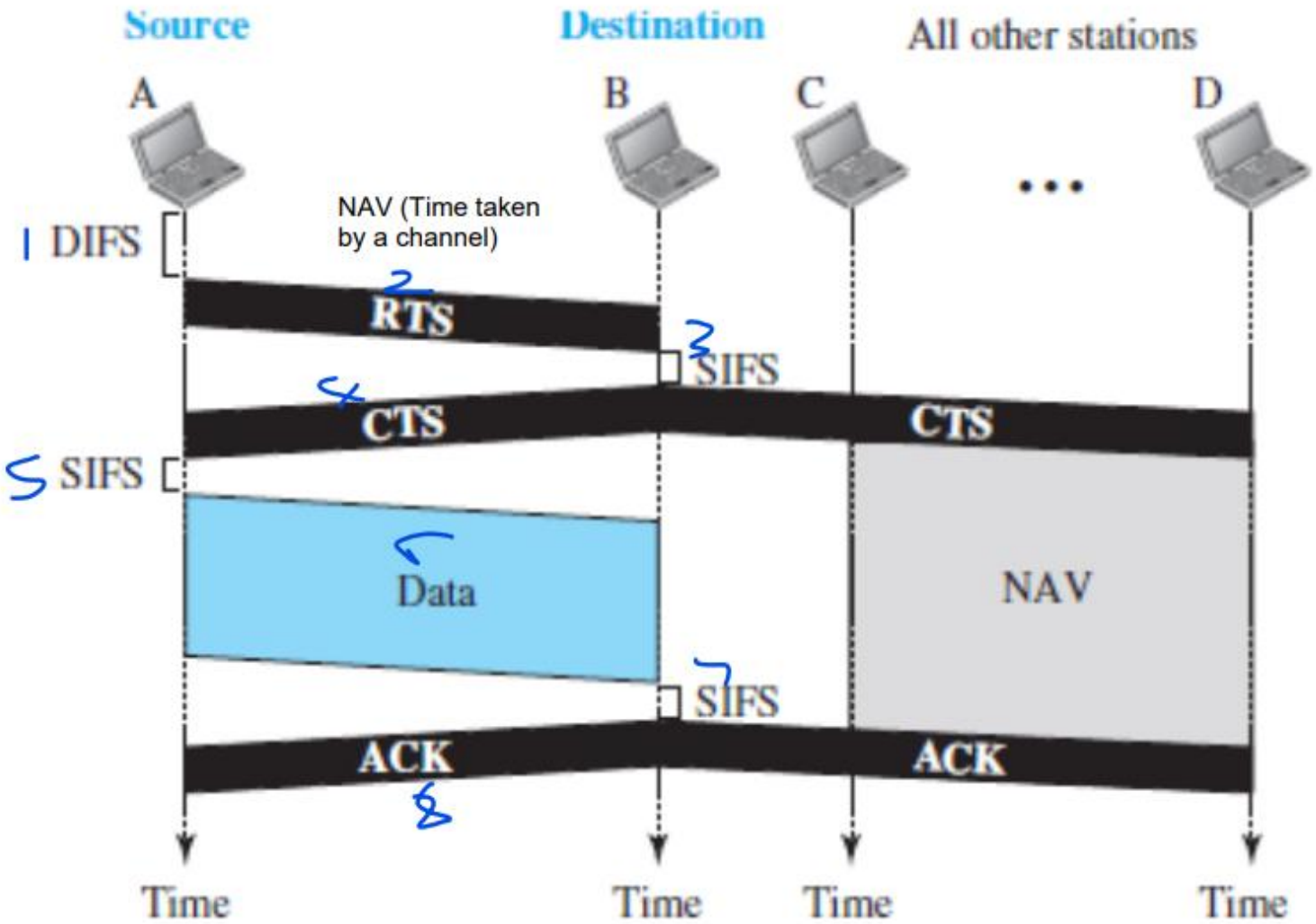
CSMA/CA - *Contention Window*





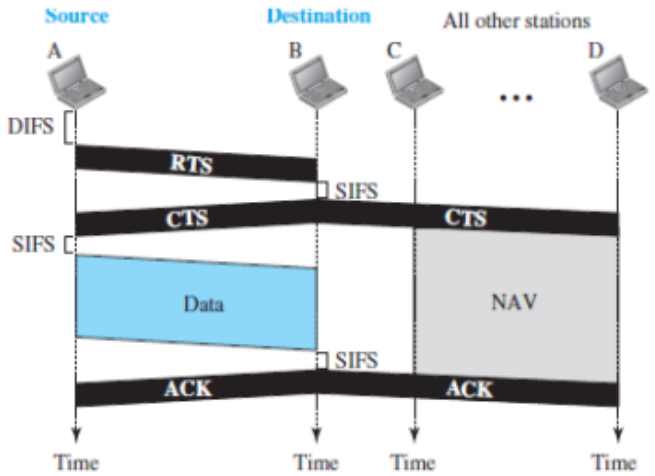
CSMA/CA -*Frame Exchange Time Line*

CSMA/CA - *Network Allocation Vector*

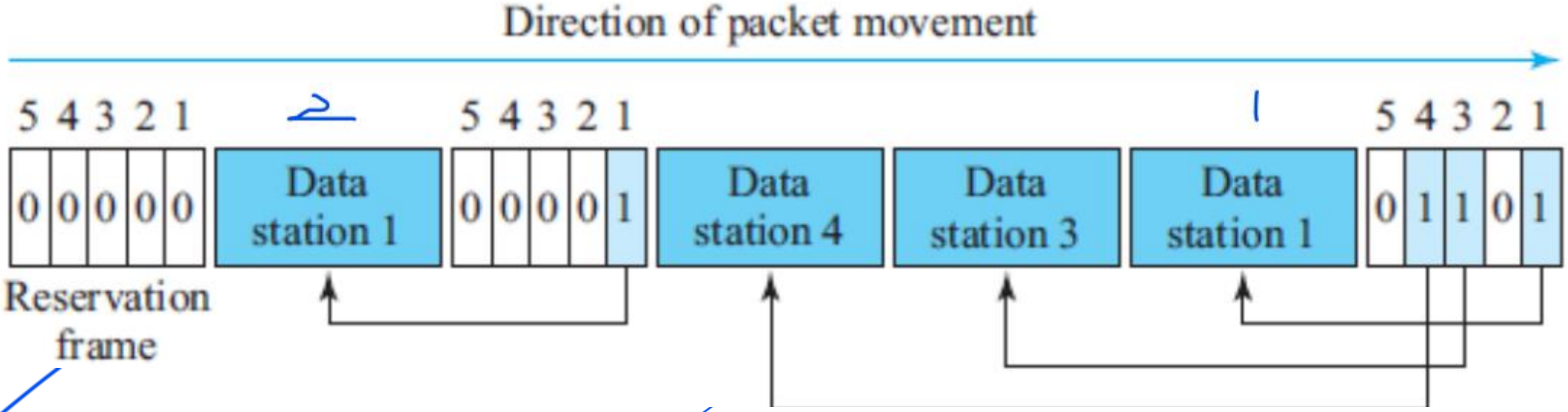


CSMA/CA - Hidden-Station Problem

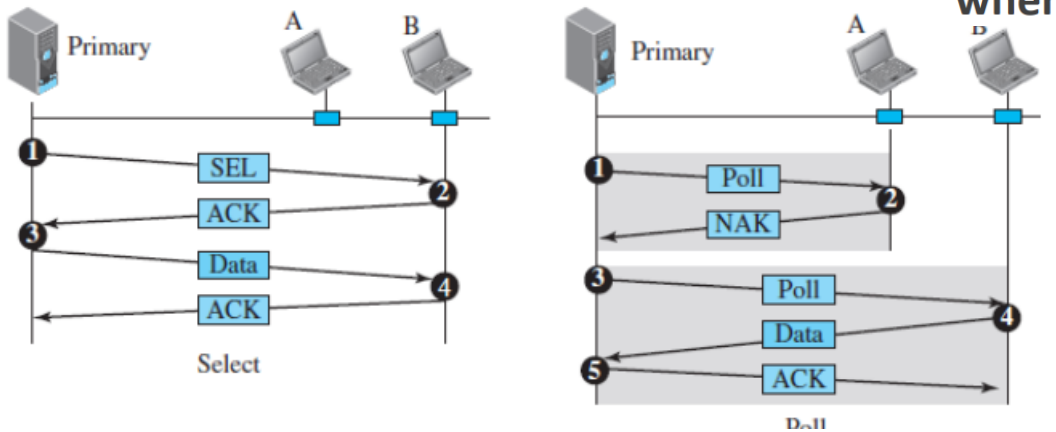
- Use of the handshake frames (RTS and CTS)
- The RTS message from B reaches A, but not C.
- However, because both B and C are within the range of A, the CTS message, which contains the duration of data transmission from B to A, reaches C
- Station C knows that some hidden station is using the channel and refrains from transmitting until that duration is over



Reservation method

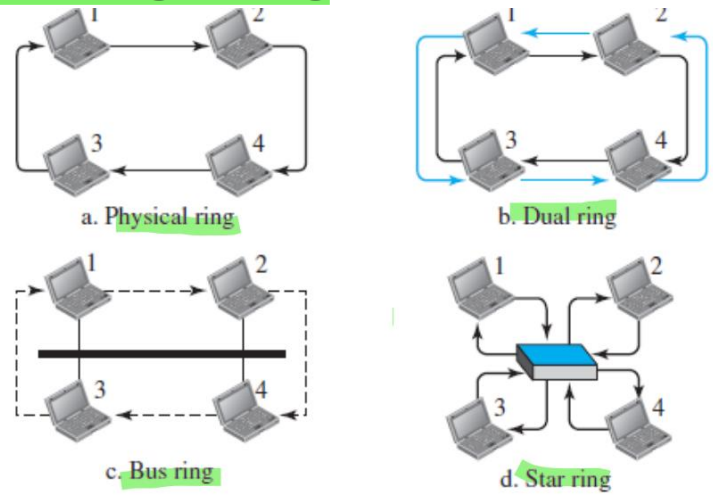


Polling



Token Passing

- The stations in a network are organized in a logical ring where each station has a *predecessor*



Unit 5

Frame Format

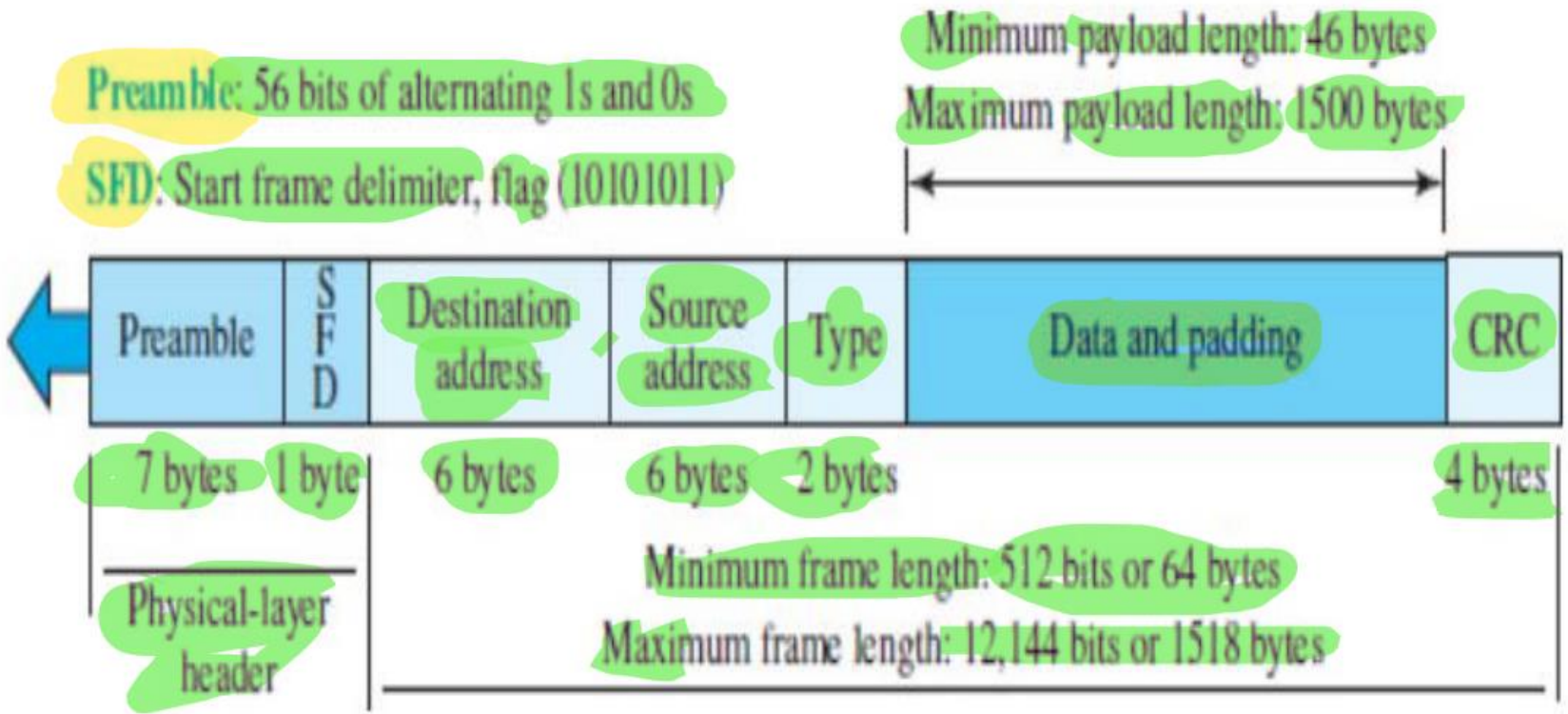


Figure 4.4 *Unicast and multicast addresses*

Example 4.2

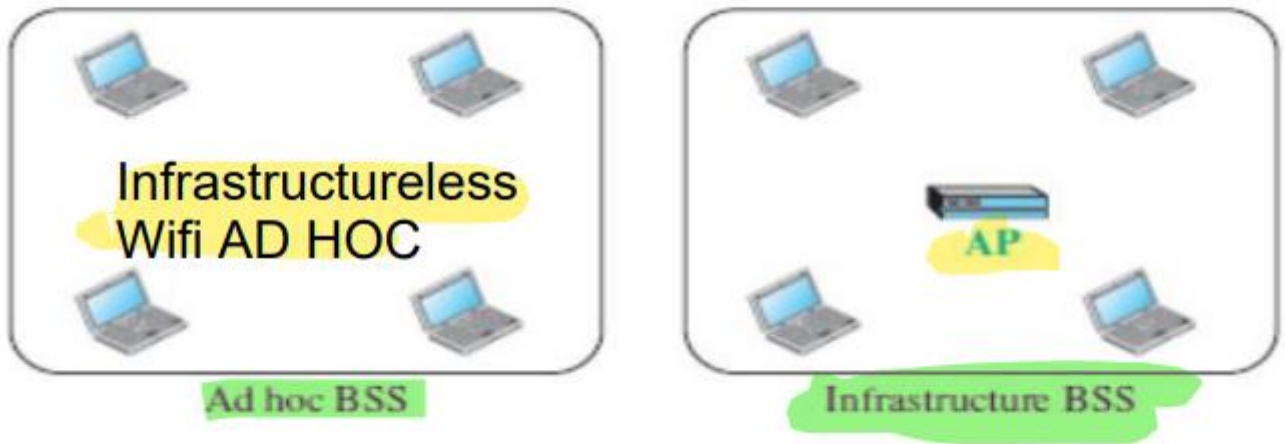
Define the type of the following destination addresses:

a. **4A:30:10:21:10:1A**

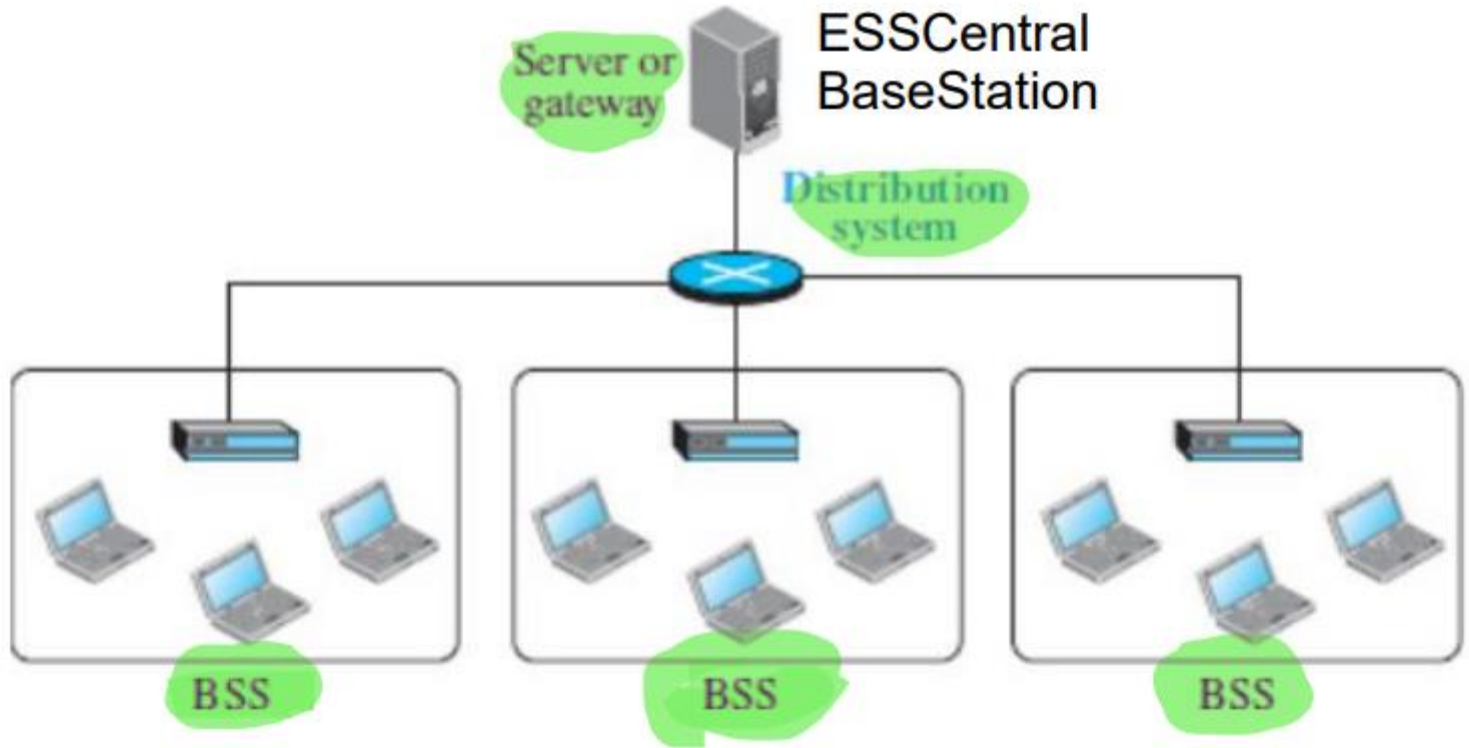
b. **47:20:1B:2E:08:EE**

c. **FF:FF:FF:FF:FF:FF**

convert the hex of 1st byte into binary
reverse it
UNICAST - if the first bit is 0
MULTICAST - if the first bit is 1
BROADCAST - all 48 bits / 6bytes are 1/FF

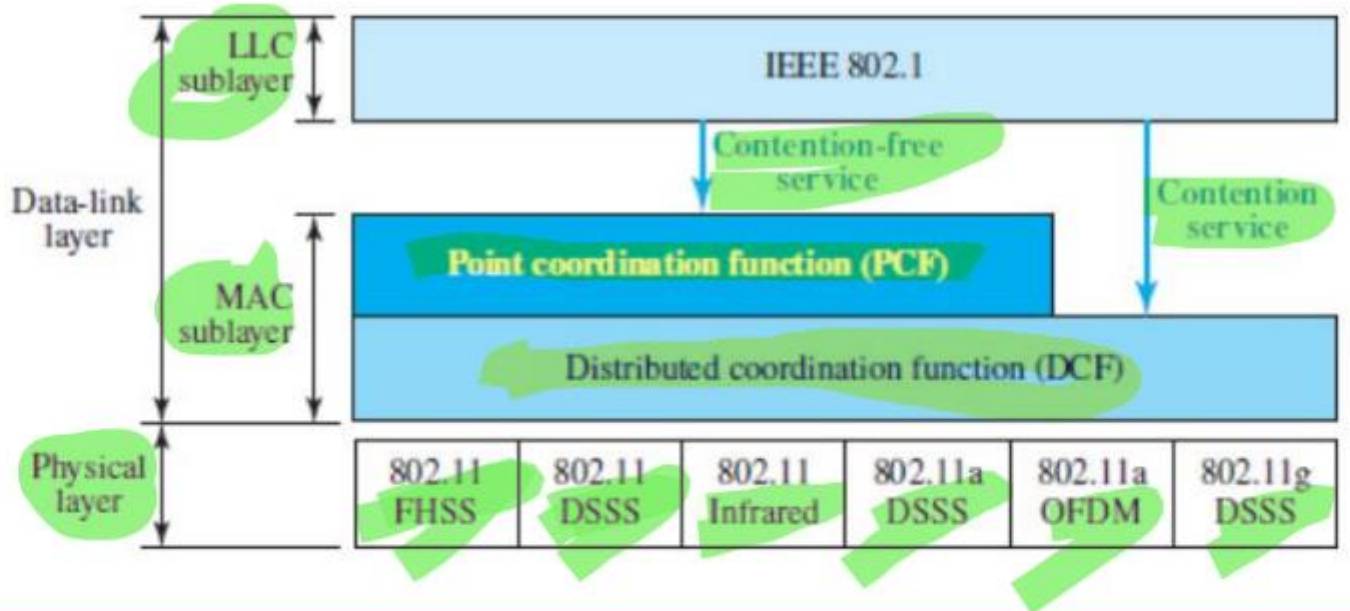

Infrastructureless
Wifi AD HOC
Ad hoc BSS
AP
Infrastructure BSS

Extended Service Set

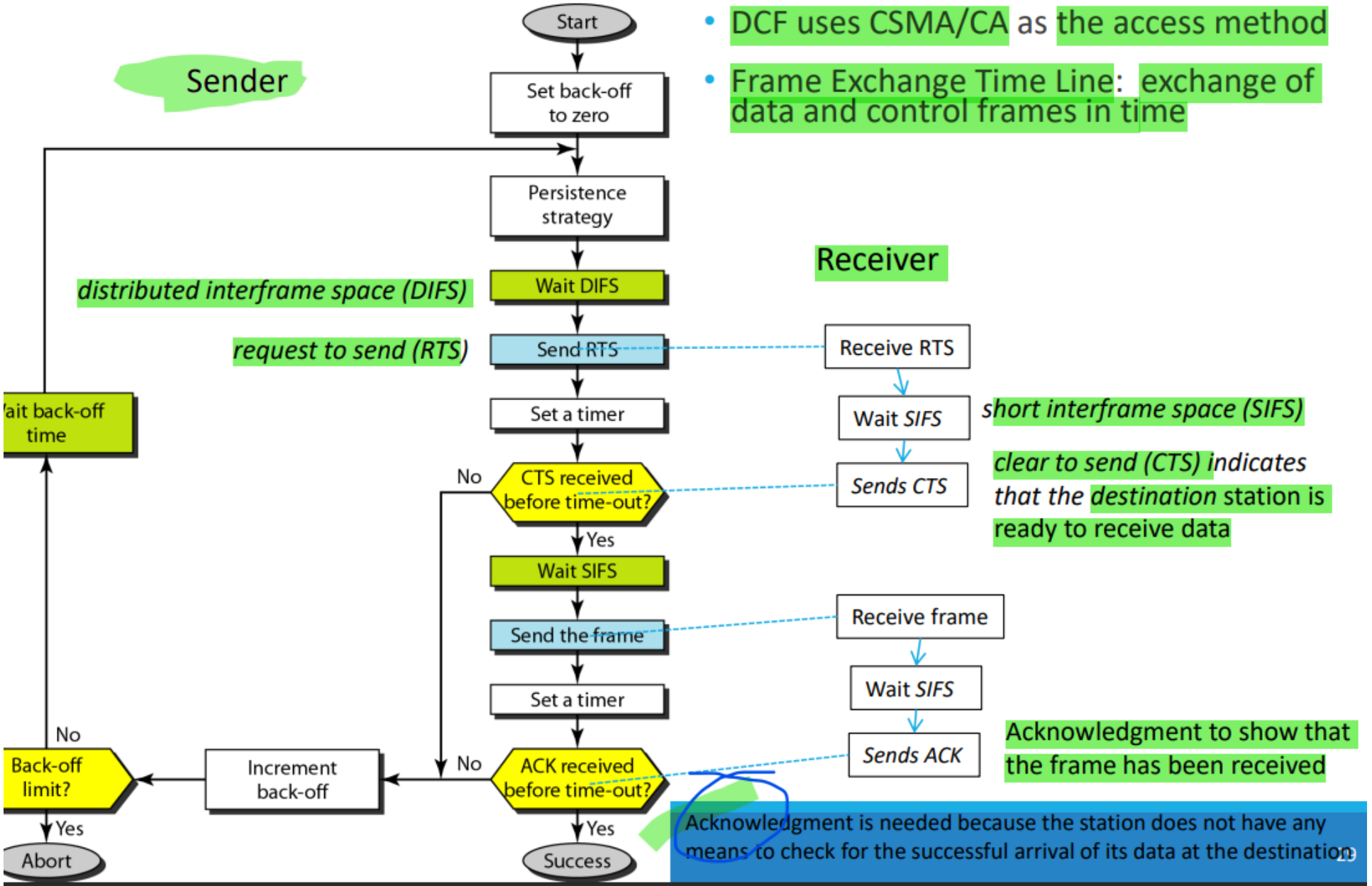


MAC Sublayer

Figure 4.9 *MAC layers in the IEEE 802.11 standard*



Distributed Coordination Function (DCF)

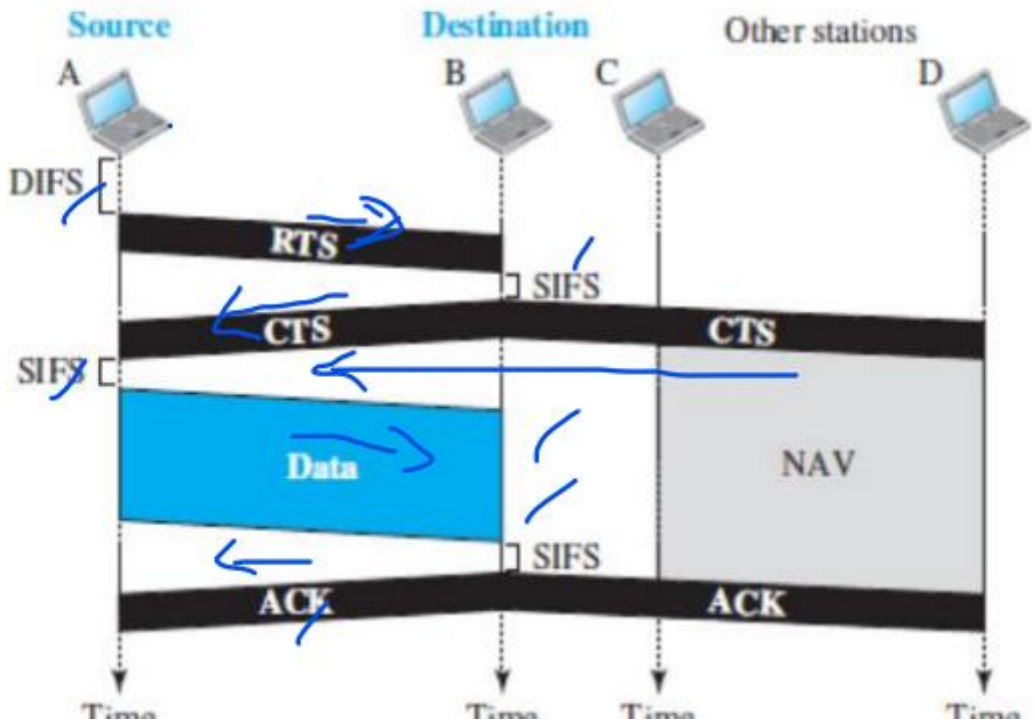


- DCF uses CSMA/CA as the access method
- Frame Exchange Time Line: exchange of data and control frames in time

Acknowledgment is needed because the station does not have any means to check for the successful arrival of its data at the destination

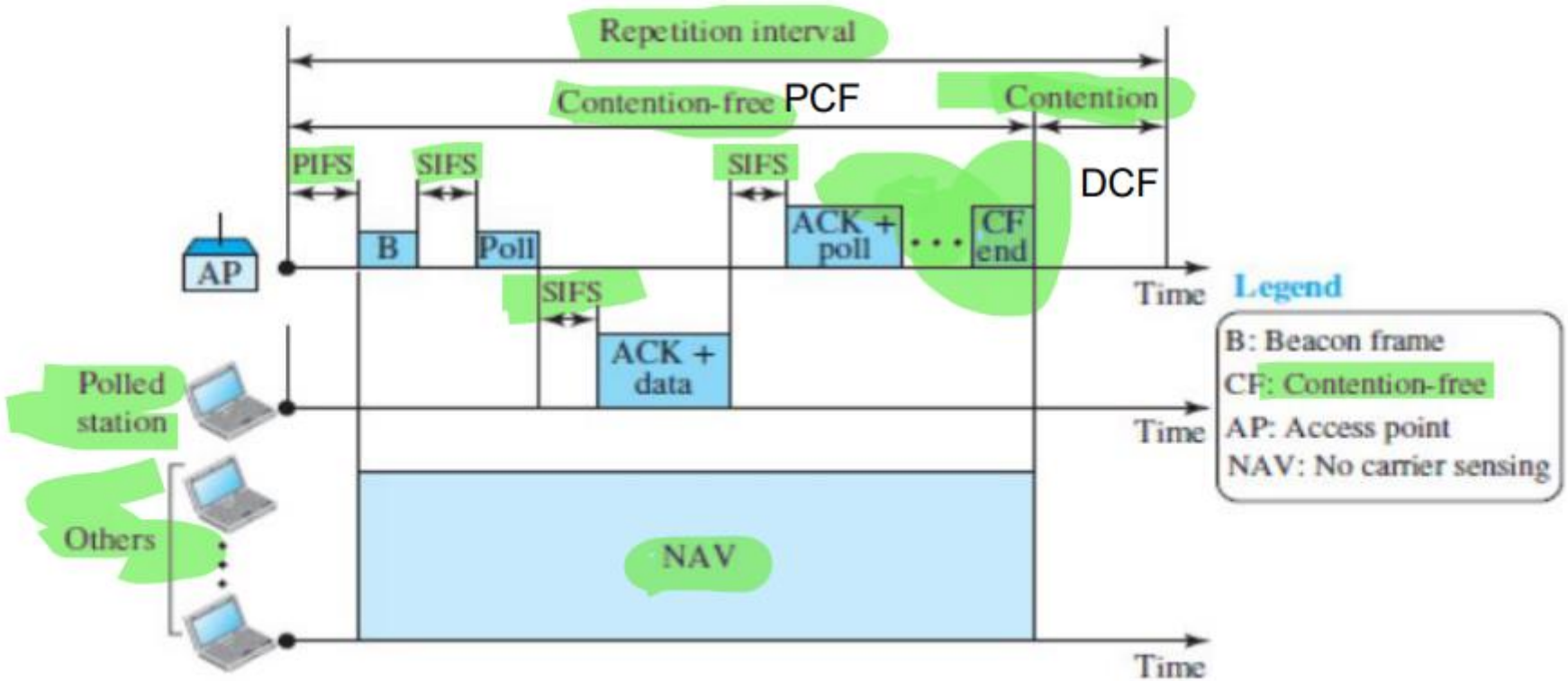
Distributed Coordination Function

Network Allocation Vector:



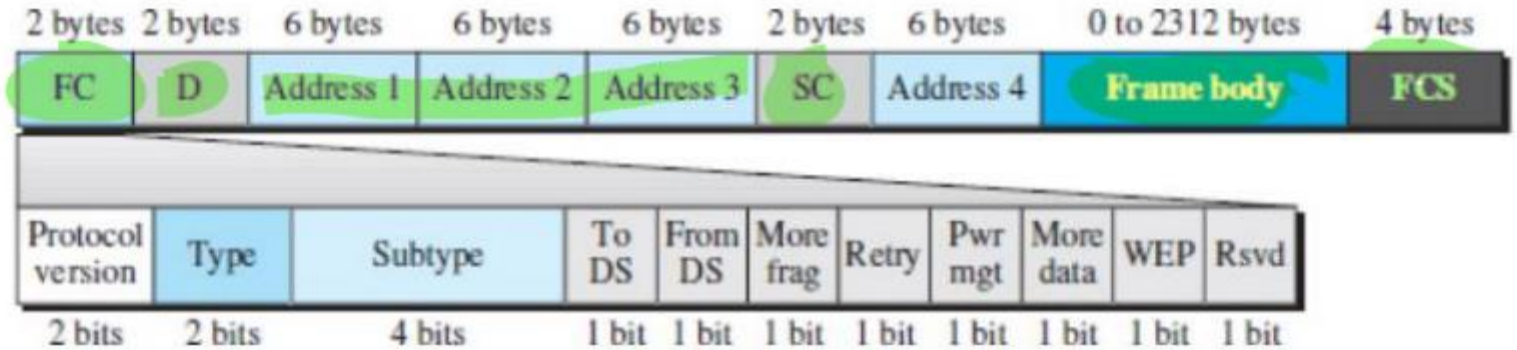
Point Coordination Function (PCF)

Figure 4.11 *Example of repetition interval*



Frame Format

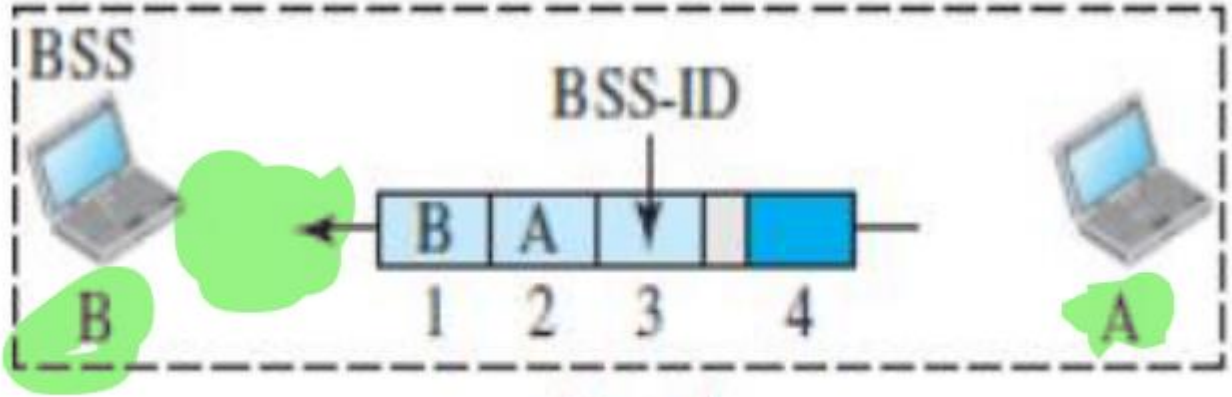
Figure 4.12 *Frame format*



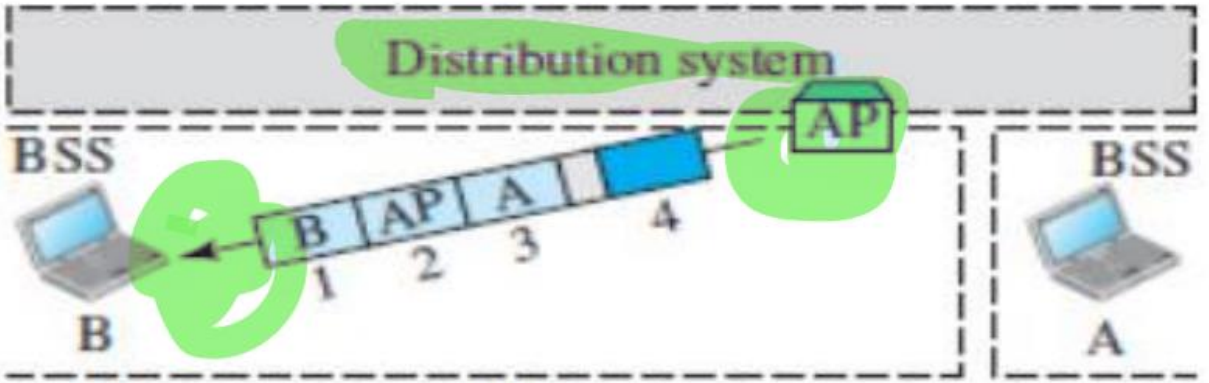
Addressing Mechanism

	To DS	From DS	Address 1 (Next device)	Address 2 (Previous Device)	Address 3 (Final Dest)	Address 4 (Original Src)	
DSB	0	0	Destination	Source	BSS ID	N/A	Station to Station
DSS	0	1	Destination	Sending AP	Source	N/A	AP to Station
RSD	1	0	Receiving AP	Source	Destination	N/A	Station to AP
RSDS	1	1	Receiving AP	Sending AP	Destination	Source	AP to AP

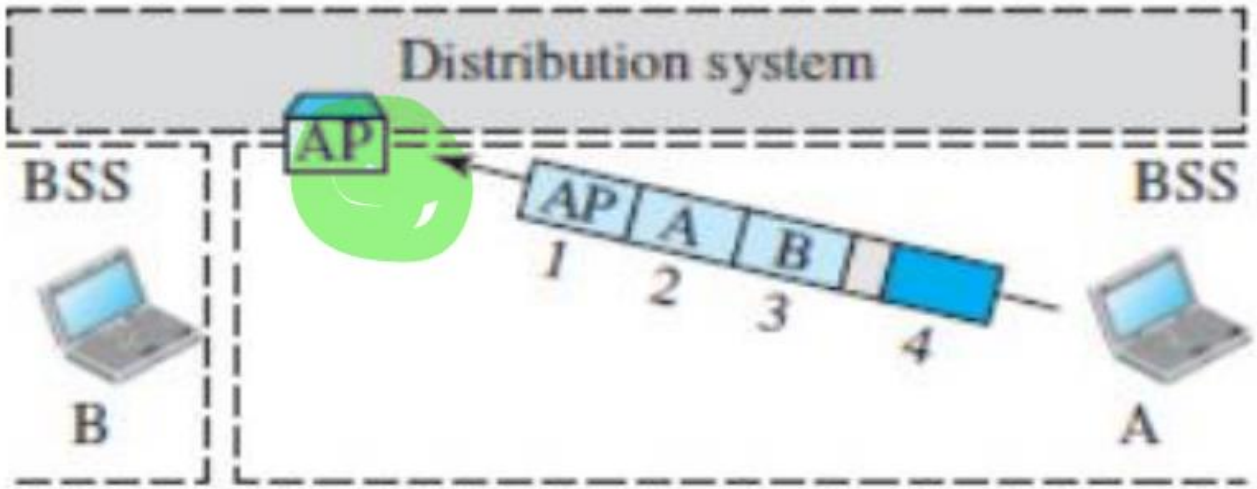
WIRELESS



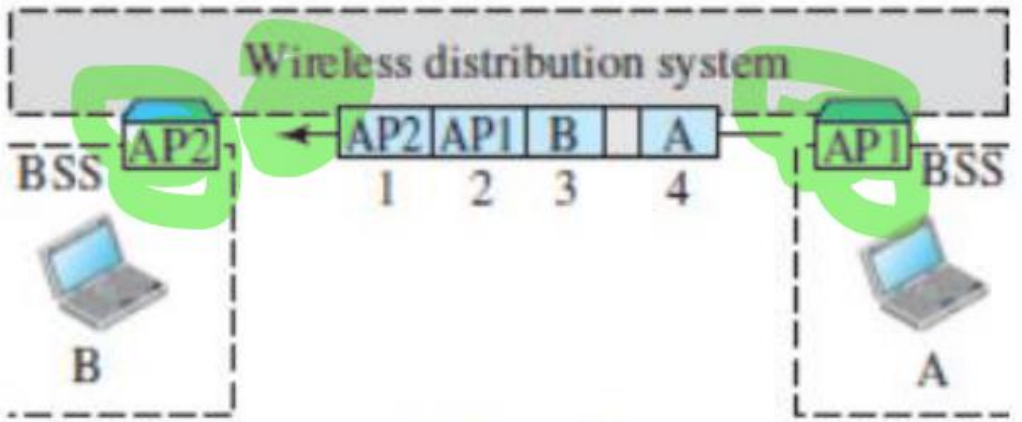
a. Case 1



b. Case 2



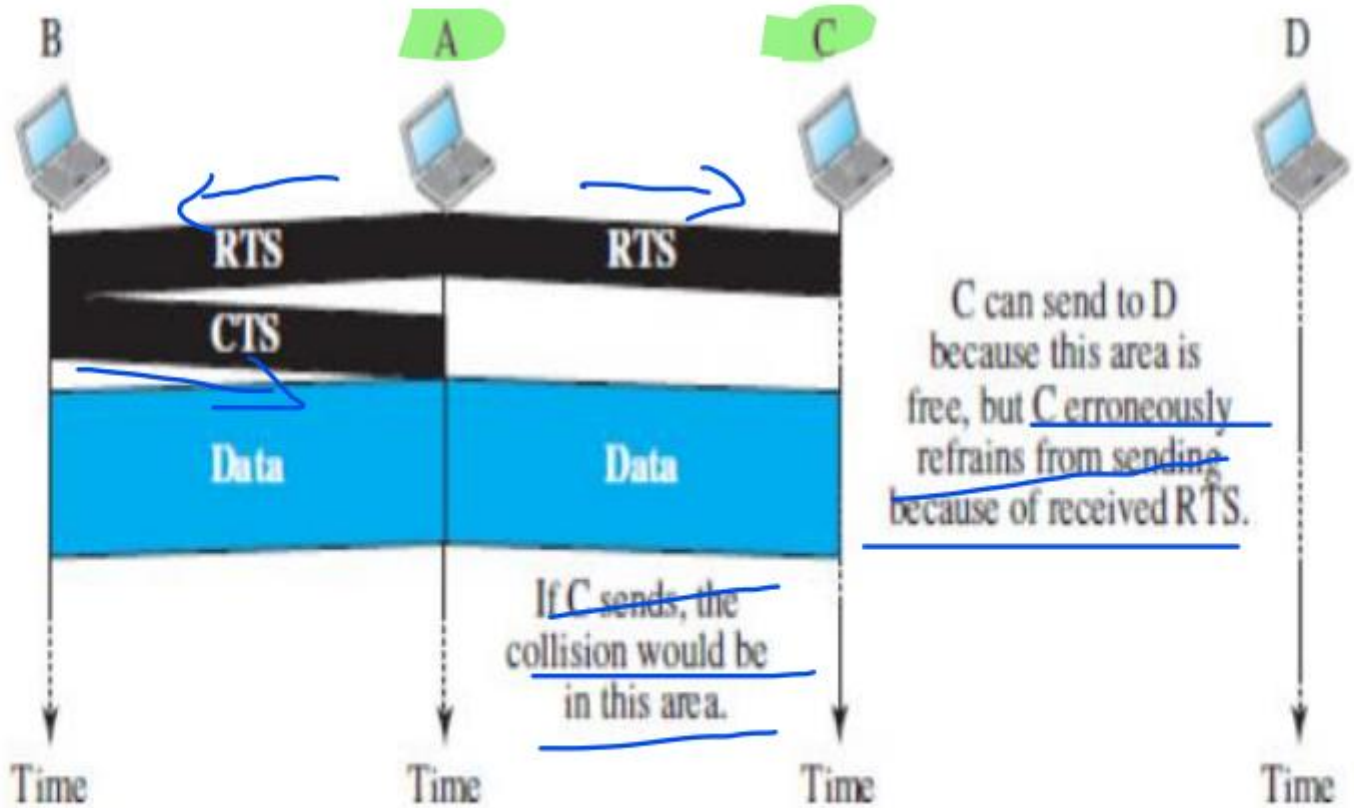
c. Case 3



d. Case 4

Exposed Station problem

Figure 4.15 *Exposed-station problem*



Exposed Station problem

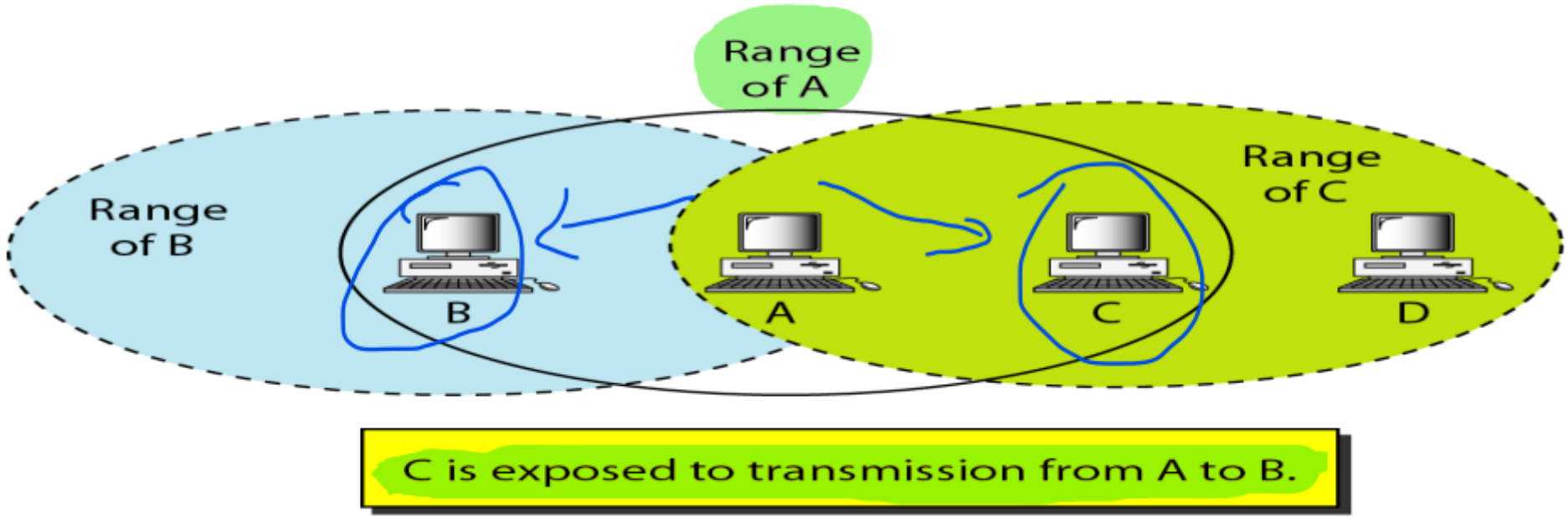
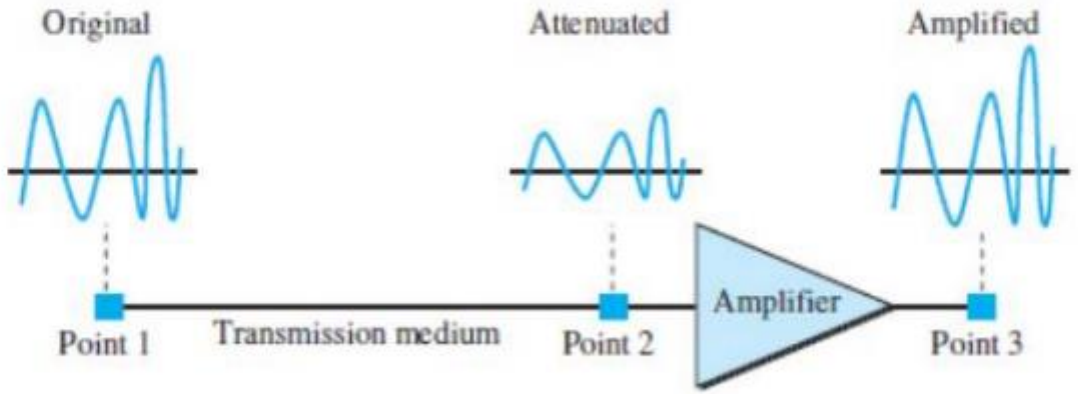
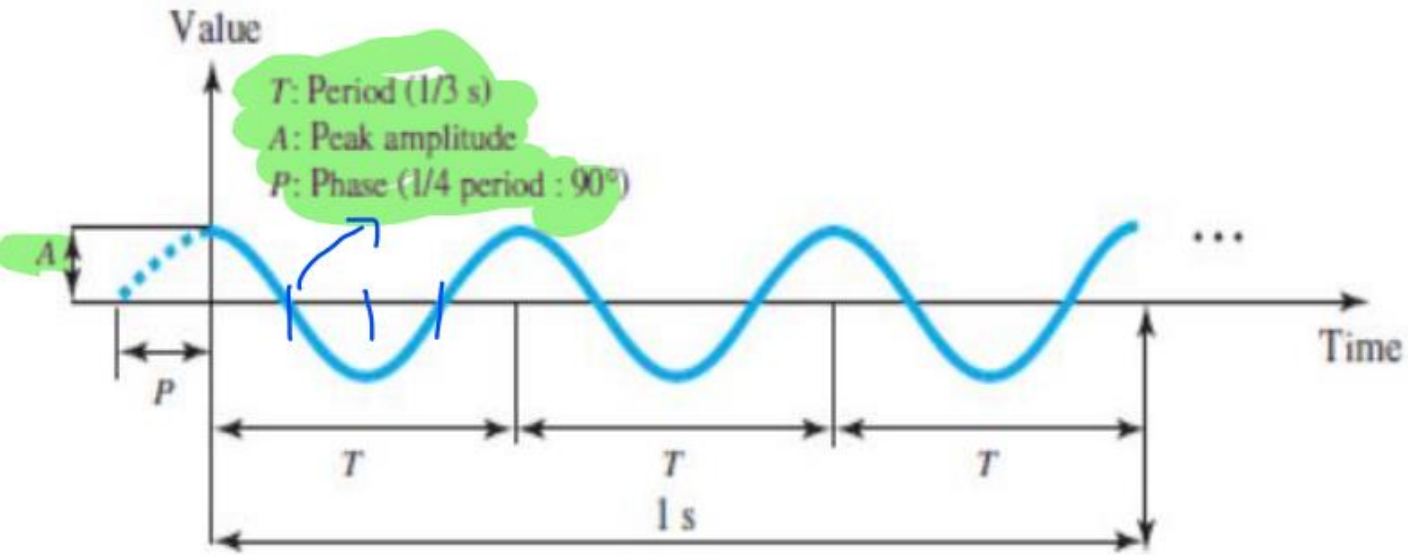


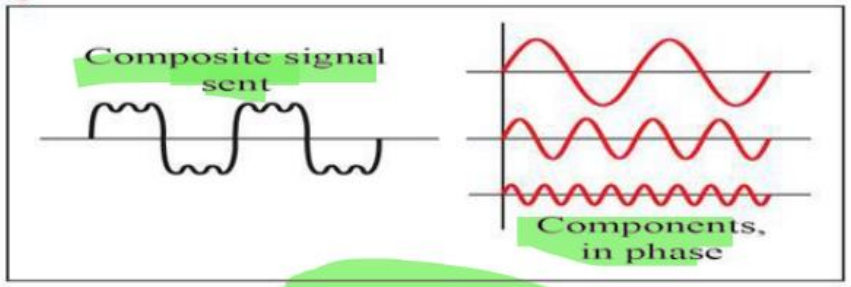
Figure 2.6 *Attenuation and amplification*



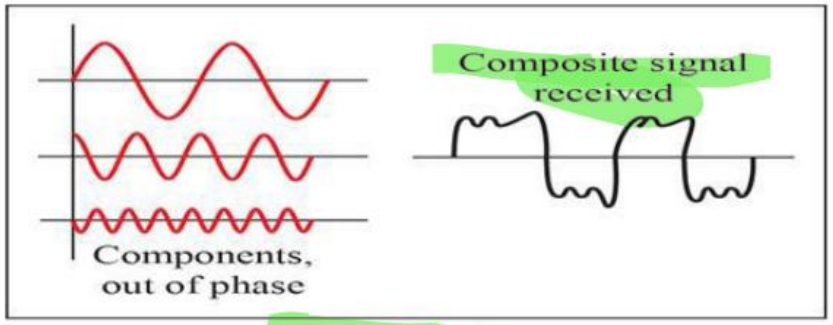


Distortion ...

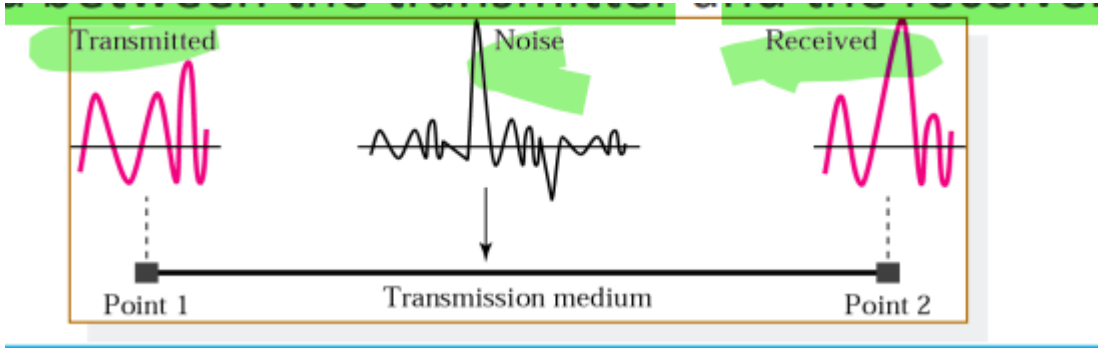
- If the delay is not exactly the same as the period duration, difference in phase
  - Signal components at the receiver have phases different from wha
  - The shape of the composite signal is therefore not the same



At the sender

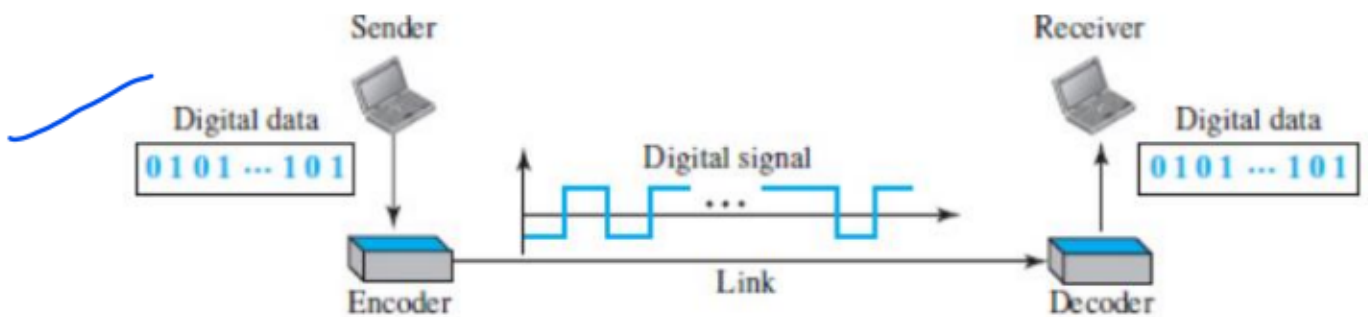


At the receiver

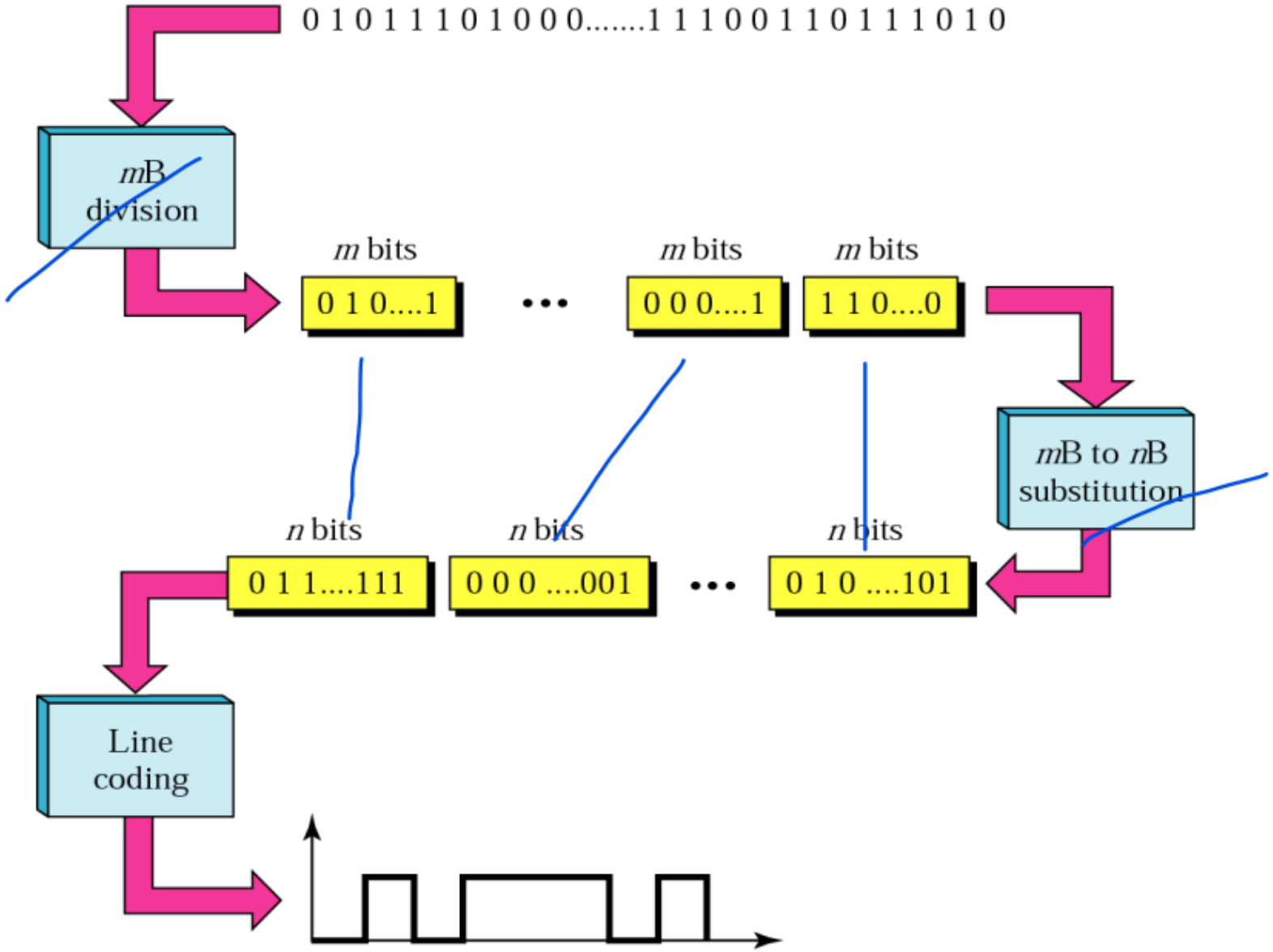


Digital-to-Digital Conversion

- The conversion involves three techniques: line coding, block coding, and scrambling.
- **Line coding** is the process of converting digital data to digital signals. We assume that data, in the form of text, numbers, graphical images, audio, or video, are stored in computer memorys as sequences of bits



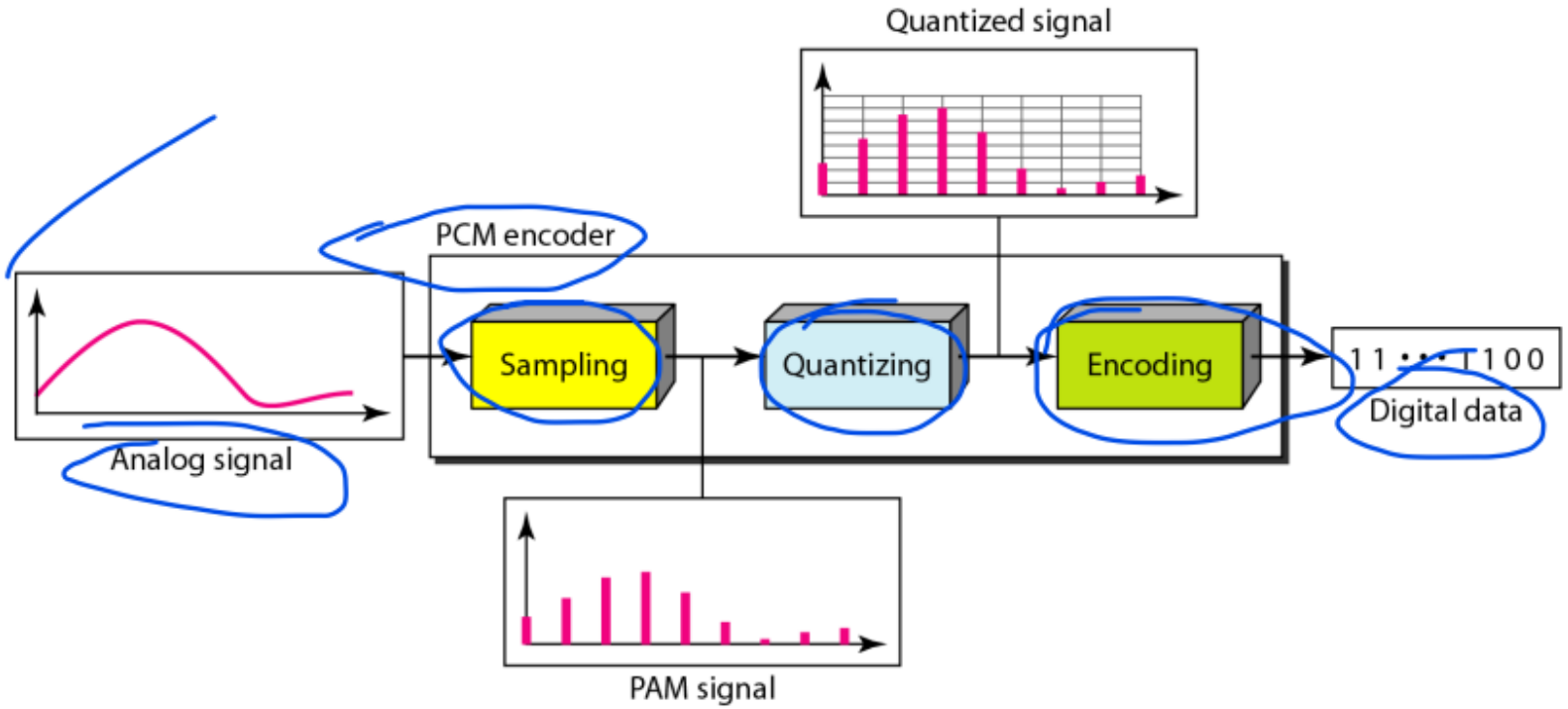
Block Coding



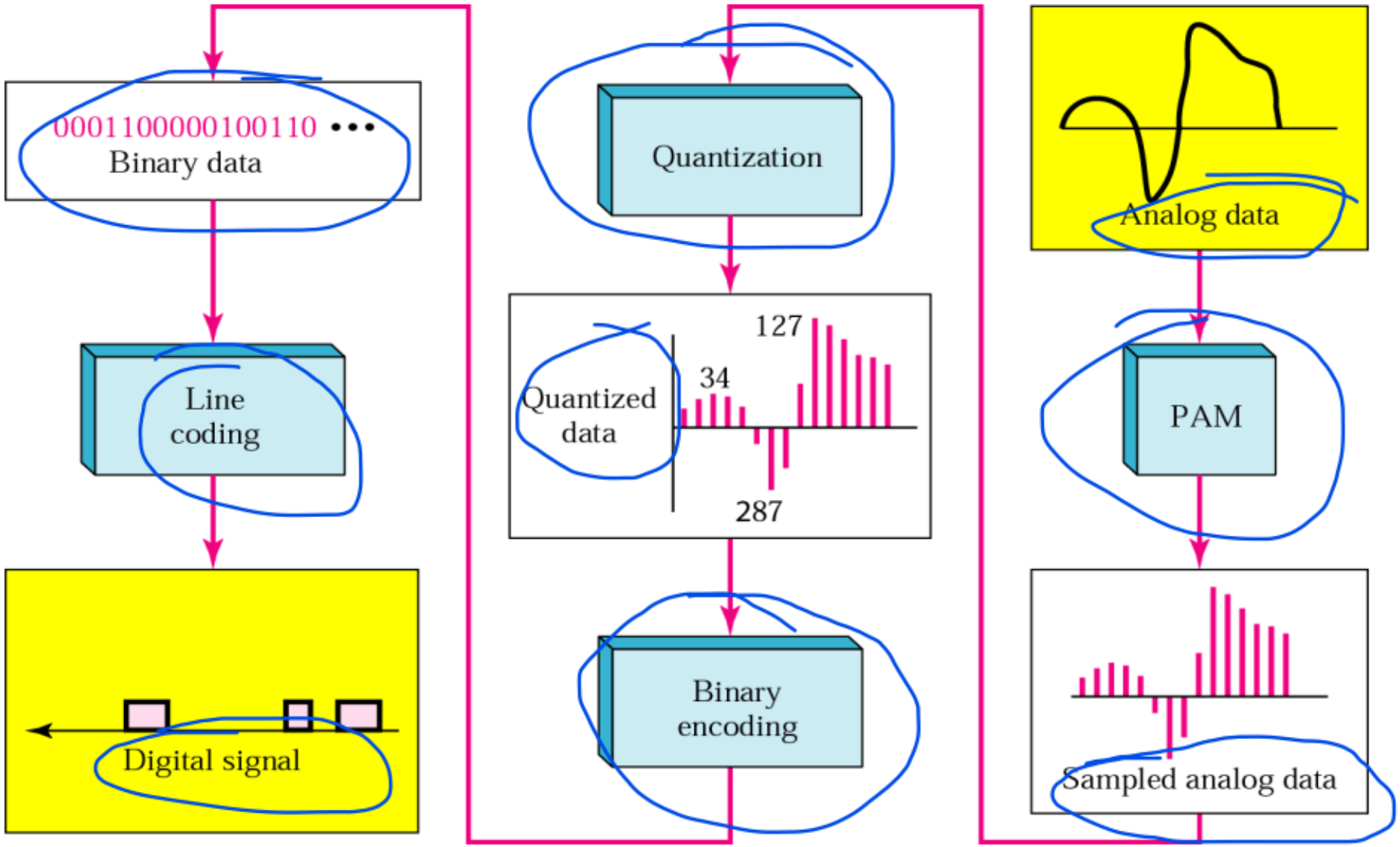
Analog to Digital Conversion

Pulse Code Modulation (PCM)

- Change an analog signal to digital data (digitization)
- A PCM encoder has three processes
  - Sampling - analog signal is sampled
  - Quantization- sampled signal is quantized
  - encoding - quantized values are encoded as streams of bits



PCM: The Whole Process

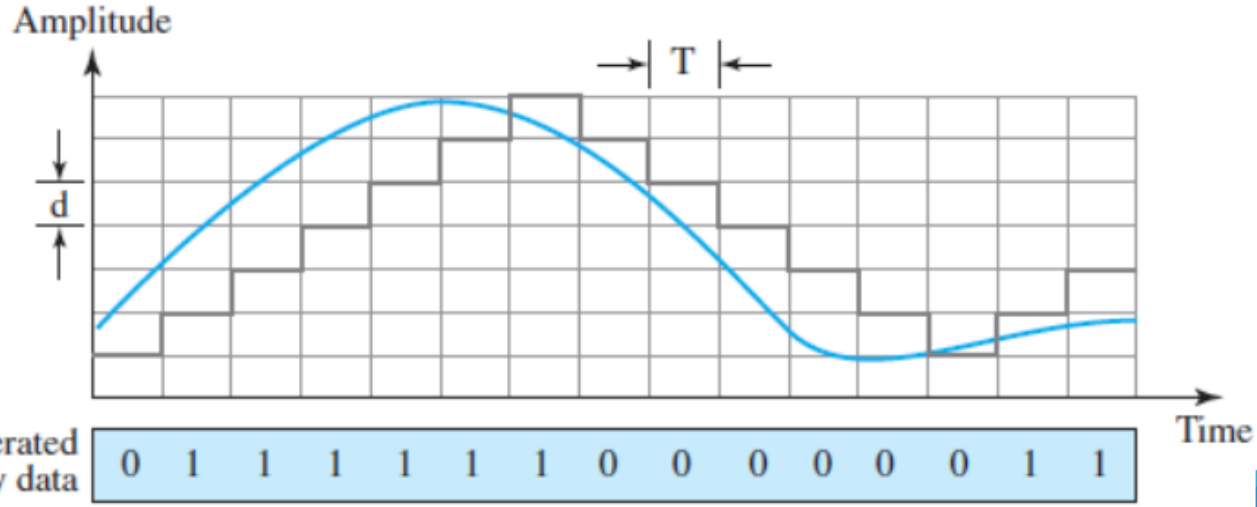


Delta Modulation (DM)

- PCM is a very complex technique
- Delta modulation reduce the complexity of PCM
- PCM finds the value of the signal amplitude for each sample whereas DM finds the change from the previous sample → bit
- DM → no code words; bits are sent one after another
  - Modulator is used at the sender site to create a stream of bits from an analog signal
  - Demodulator is used at receiver site to creates the analog signal from the received digital data
- DM is not perfect → Quantization error introduced in the process
  - The quantization error of DM is much less than that for PCM

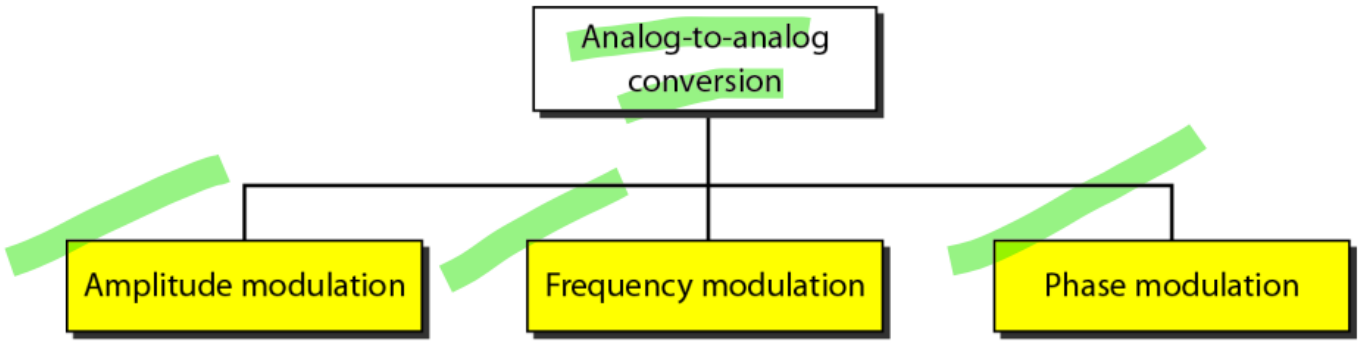
DM...

- The process records the small positive or negative chan called delta δ
  - If the delta is positive, the process records a 1
  - If it is negative, the process records a 0
- A better performance achieved if the value of δ is not f
  - Adaptive delta modulation - value of δ changes according to th amplitude of the analog signal.



Analog-to-analog conversion

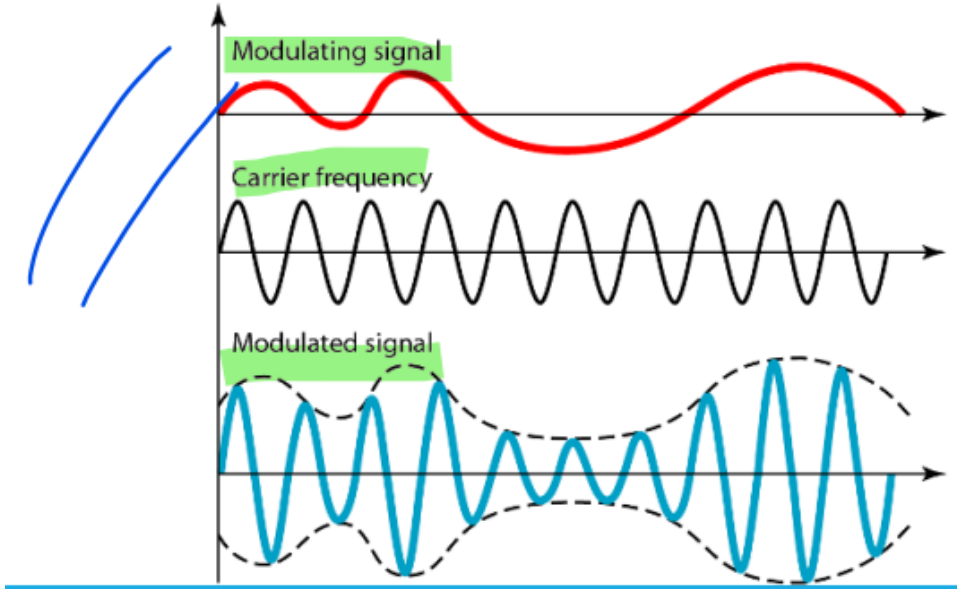
- Change an analog signal to a new analog signal with a smaller bandwidth
  - used when only a band-pass channel is available
- Three methods:
  - Amplitude modulation (AM) : amplitude of a carrier is changed based on the changes in the original analog signal
  - Frequency modulation (FM): the phase of a carrier is changed based on the changes in the original analog signal
  - Phase modulation (PM): the phase of a carrier signal is changed to show the changes in the original signal

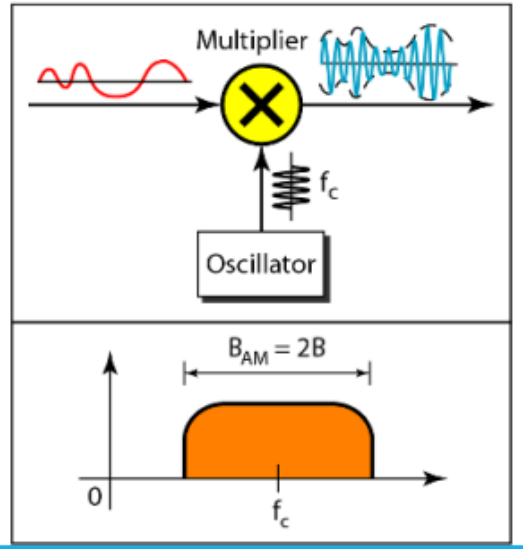


amplitude is modulated

Amplitude Modulation (AM)

- Amplitude of the carrier signal is modulated to follow the changing amplitudes of the modulating signal
  - Only the amplitude changes to follow variations in the information
  - The frequency and phase of the carrier remain the same
- The modulating signal is the envelope of the carrier
- The amplitude of the carrier signal changed according to the amplitude of the modulating signal using a simple multiplier





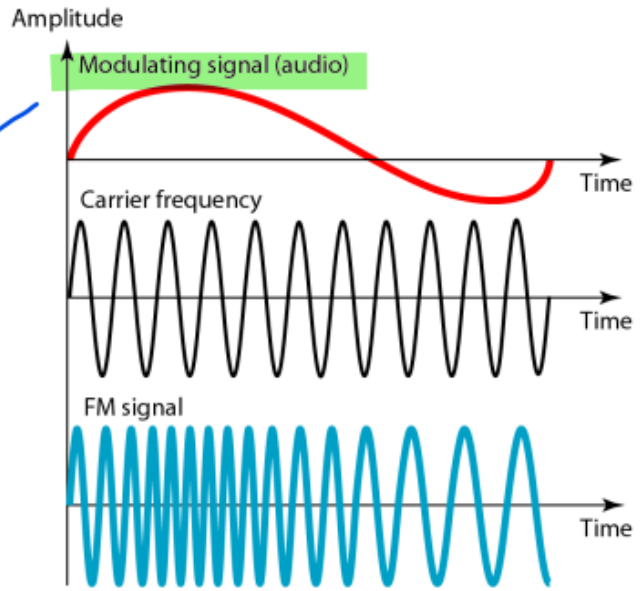
Frequency is modulated

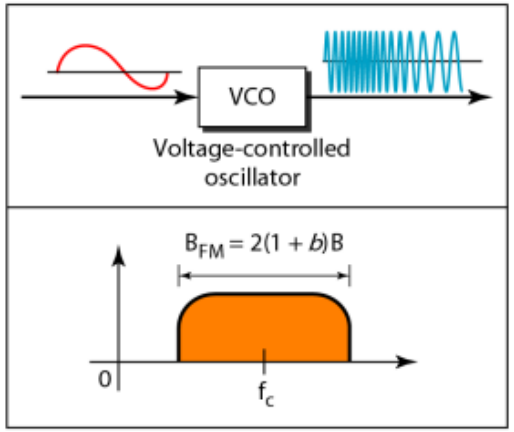
Frequency Modulation (FM)

- Frequency of the carrier signal is modulated to follow the changing voltage level (amplitude) of the modulating signal
  - Peak amplitude and phase of the carrier signal remain constant,
  - As the amplitude of the information signal changes, the frequency of the carrier changes correspondingly
- Implemented by using a voltage-controlled oscillator as with FSK
  - The frequency of the oscillator changes according to the input voltage which is the amplitude of the modulating signal

Beta = 4

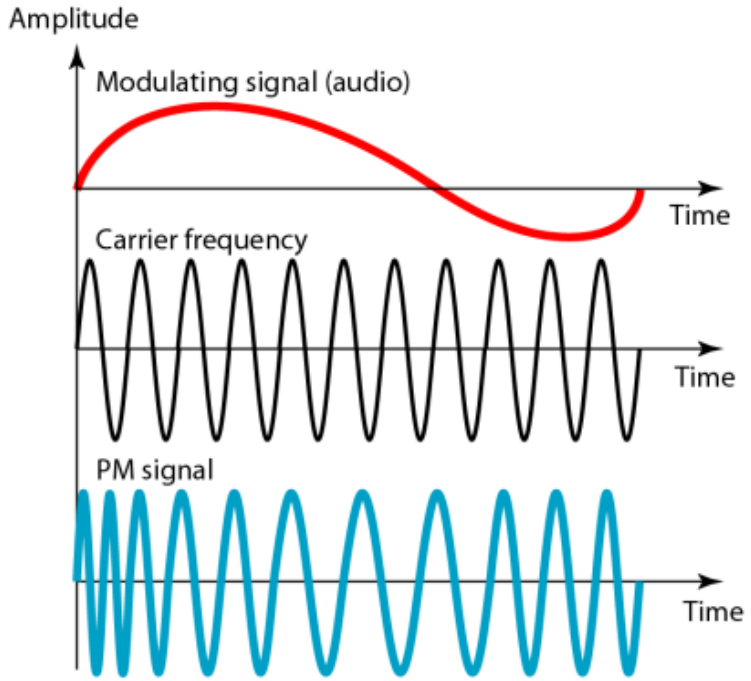
$$B_{PM} = 2(1 + \beta)B$$

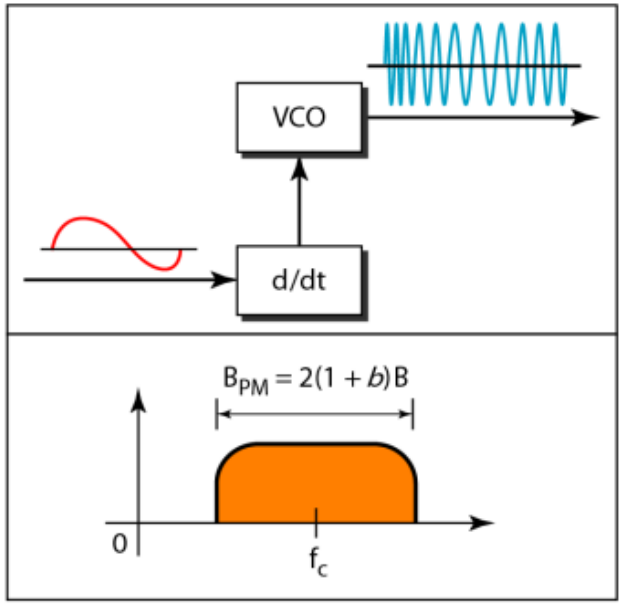




Phase Modulation (PM)

- PM is normally implemented by using a voltage-controlled oscillator along with a derivative
- Frequency of the oscillator changes according to the deriv of the input voltage, which is the amplitude of the modula signal.

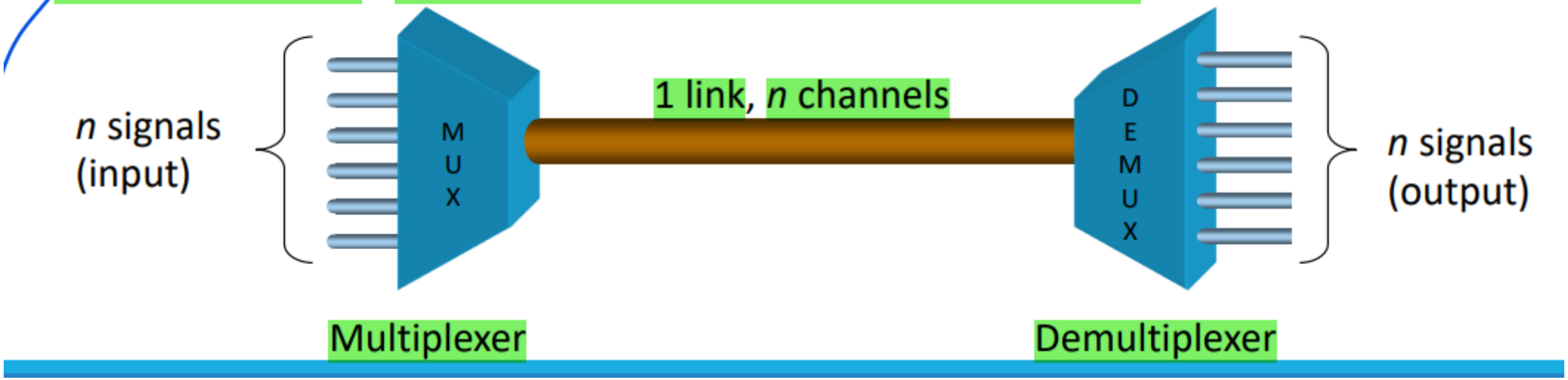




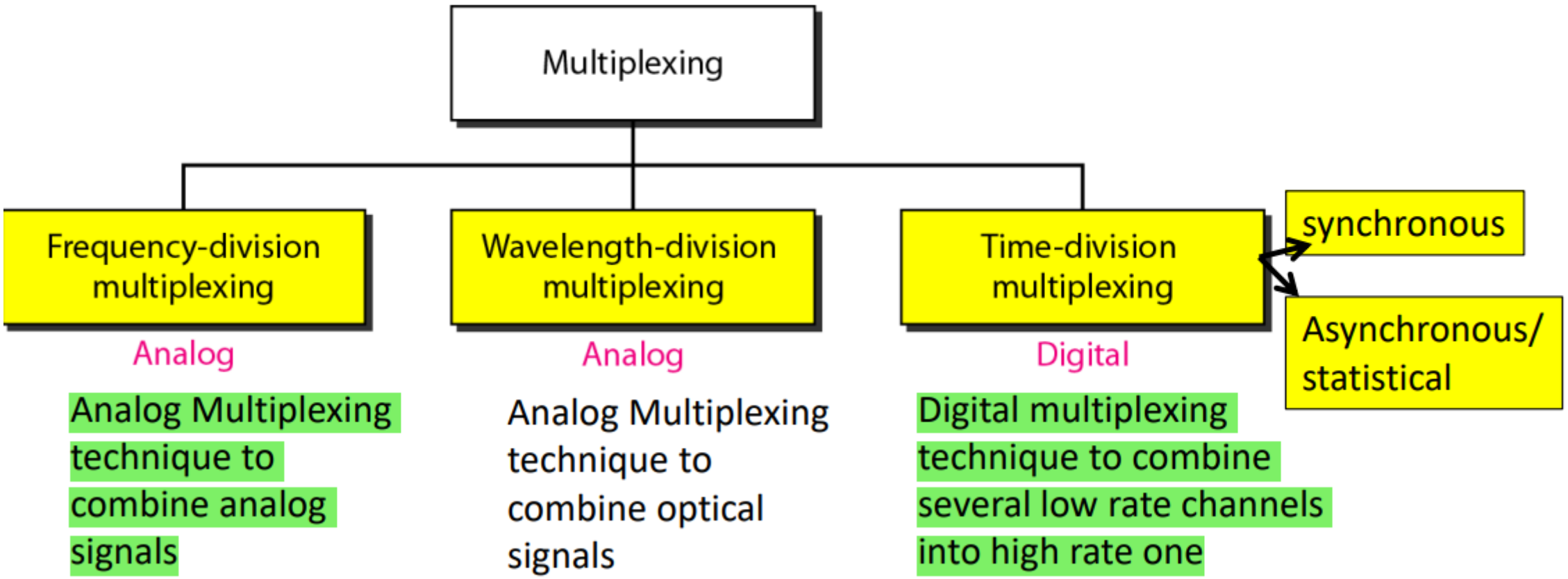
Beta = 3

$$B_{PM} = 2(1 + \beta)B$$

Shared Link: uses multiplexing technique

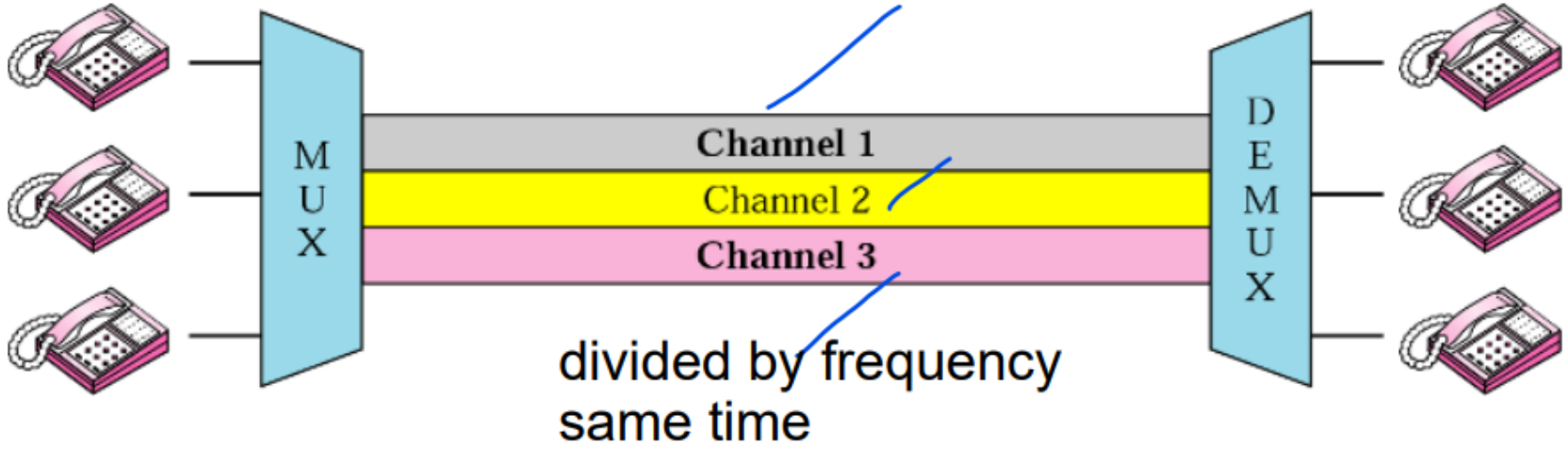


Multiplexing ...

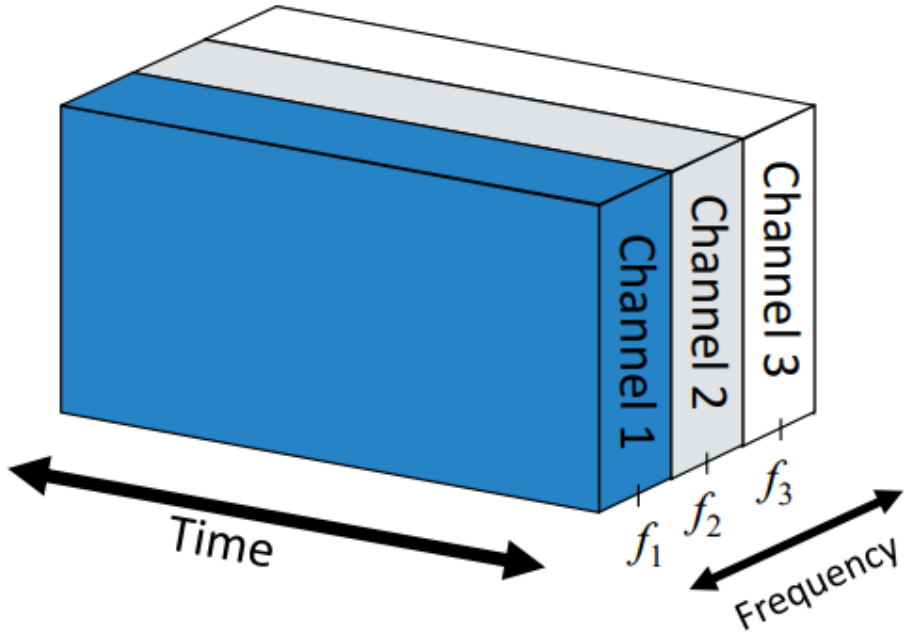


radio

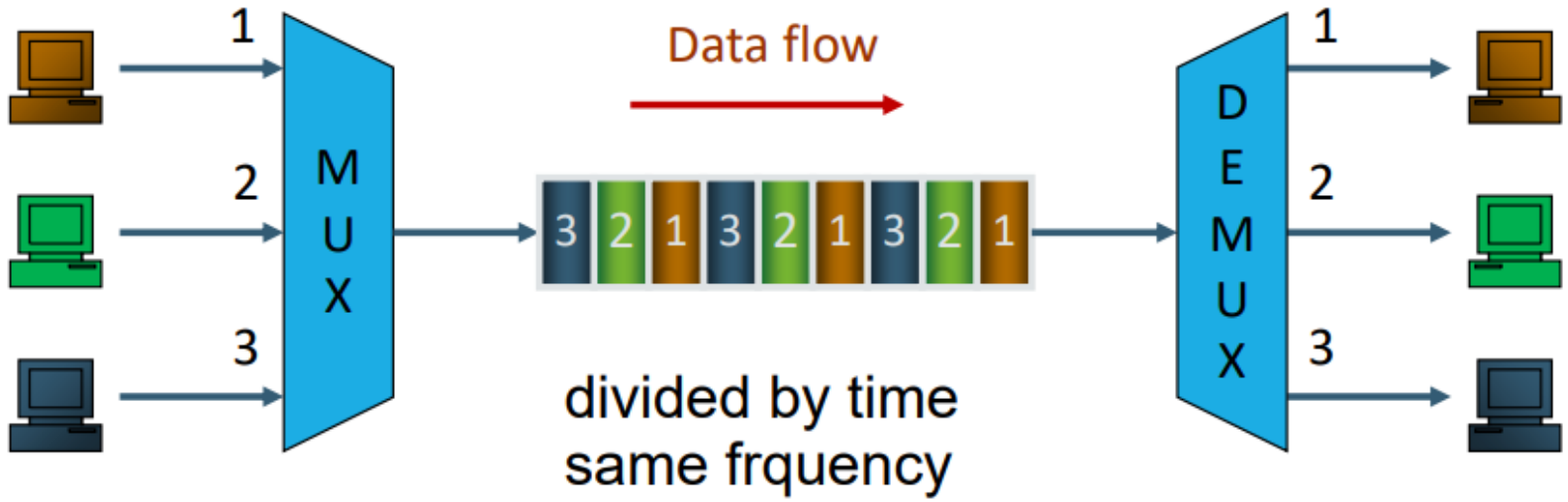
Conceptual View of FDM



divided by frequency
same time



Conceptual View of TDM



- Portions of signals 1, 2, and 3 occupy the link sequentially
- Data in a message from source 1 always go to one specific destination 1, 2, 3, or 4
- The delivery is fixed and unvarying, unlike switching