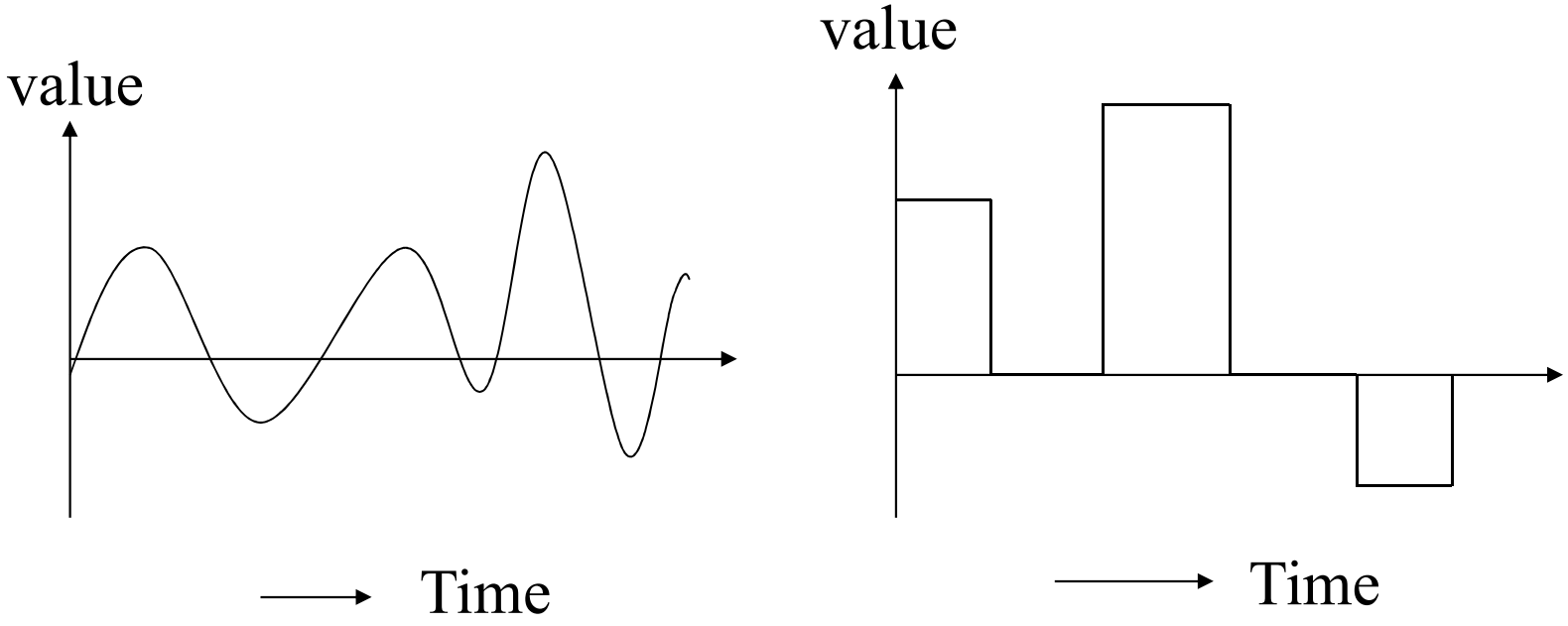
Physical Layer

- Function: To move data in the form of electromagnetic signals across a transmission medium.
- Transmission media work by conducting energy along a physical path.
- Design issues largely deal with the electrical, mechanical and physical transmission medium.

Analog and Digital

- Both data and the signals which represent them can be either analog or digital.
- Analog data: Information that is continuous.

  Ex: Analog clock.
- Digital data: Information containing discrete values.
  Ex: Digital clock
- Analog signal: A signal having infinitely many levels of intensity over a period of time.
- Digital signal: A signal having only limited number of defined values.

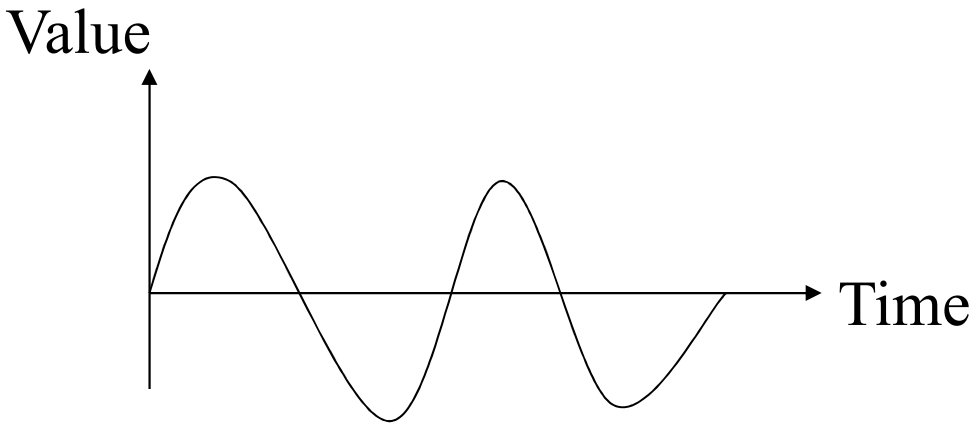


Analog signal

Digital signal

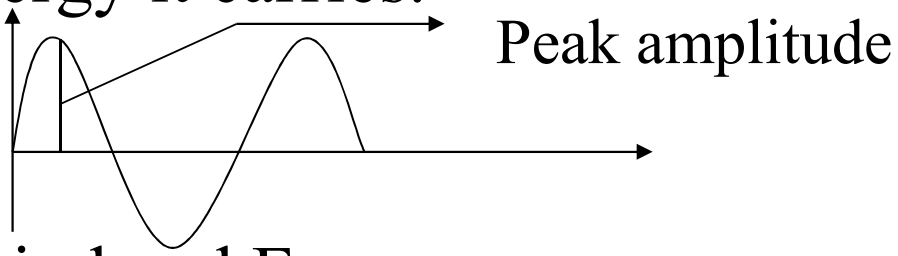
Analog and digital signals can be periodic (exhibits a pattern within a measurable time ,"period" and repeats it over subsequent identical periods. Completion of one full pattern is called cycle) and nonperiodic.

- In data communication Periodic Analog Signal and Non periodic Digital Signal are used.
- Periodic analog Signals → Simple, Composite
- Simple Periodic Analog Signal:Ex. Sine Wave



A Sine Wave is represented by 3 parameters: Peak amplitude, Frequency and Phase

- Peak Amplitude: It is the absolute value of the highest intensity of the signal proportional to the energy it carries.



Period and Frequency:

Period (T): amount of time in seconds, a signal needs to complete one cycle.

Frequency(F): Refers to number of periods in one second.

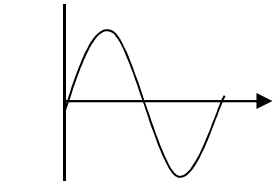
F = 1/ T

- Electromagnetic signals are oscillating waveforms i:e they fluctuate continuously above and below the mean energy level.

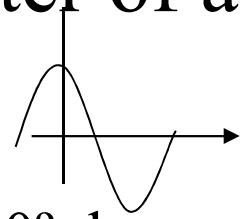
Hence, frequency is also rate of change w.r.t. time.

- Phase: It describes the position of the waveform relative to time 0. It indicates the status of the first cycle.
- Measured in degrees or radians

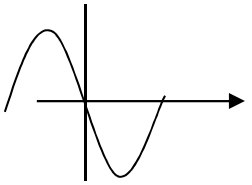
  (360º is 2∏ rad)
- A phase shift of 360º corresponds to a shift of a complete period, a phase shift of 180º corresponds to a shift of one half of a period and a phase shift of 90º corresponds to a shift of one quarter of a period.



0º degrees



90º degrees

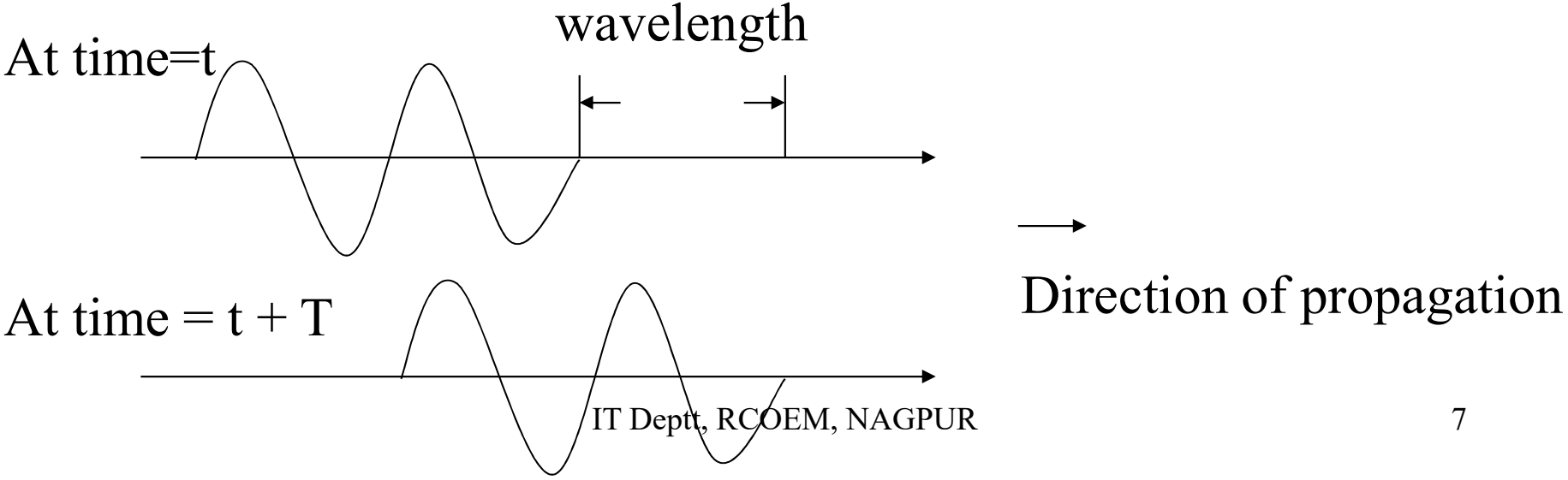


180º degrees

- Wavelength: It's the distance traveled by a simple signal in one period. Wavelength binds the period or frequency of a simple sine wave to the propagation speed of the medium.
- Wavelength = propagation speed * period

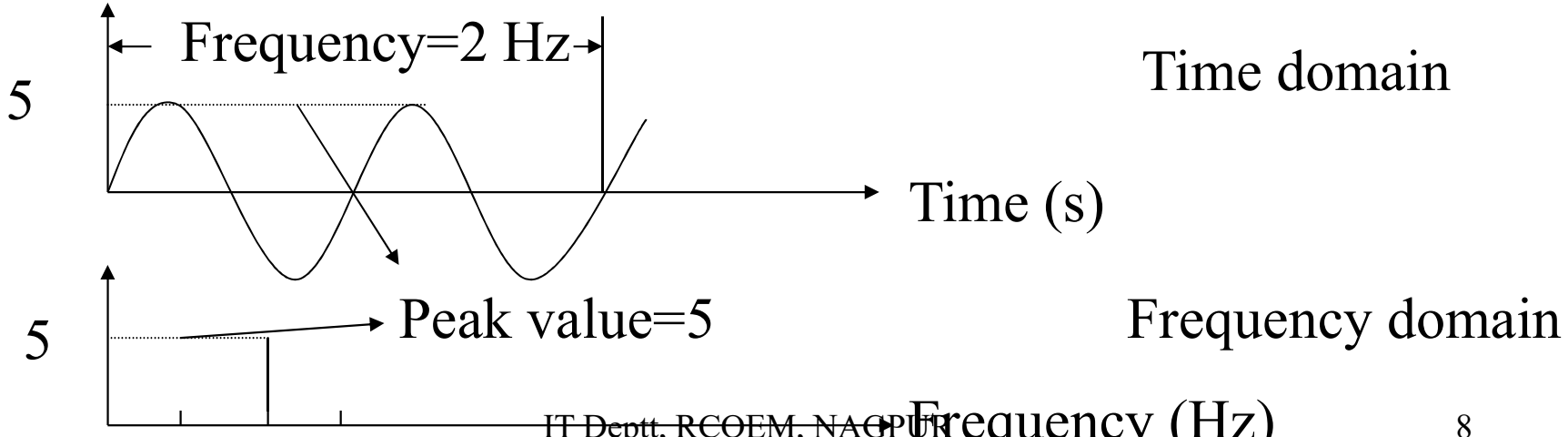
  = propagation speed / frequency

  $\lambda = c/f$ (measured in micrometers)

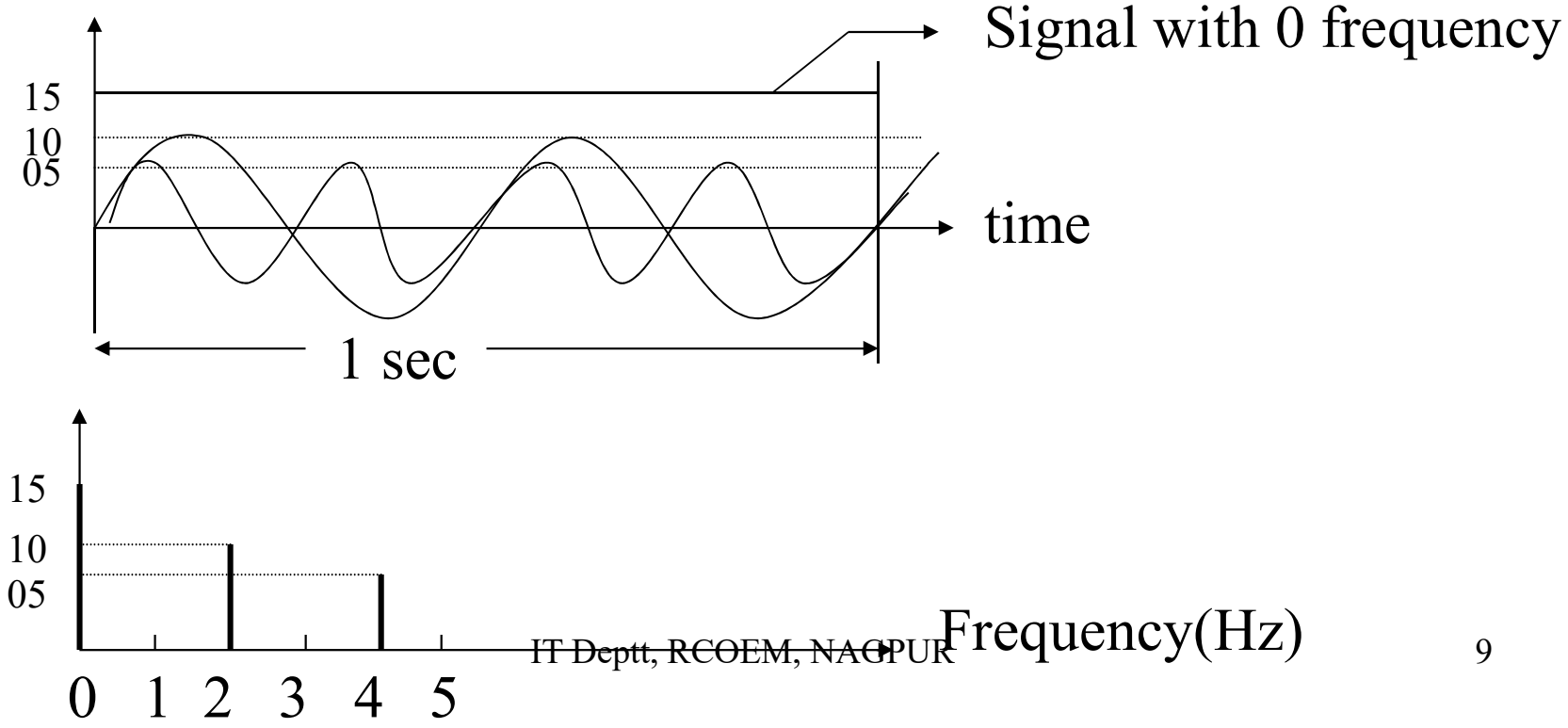


Time and Frequency domains

- The time domain plot shows the changes in signal amplitude w.r.t. time.
- To show the relationship between amplitude and frequency, we can use Frequency domain plot. It deals with only peak value and the frequency.



- Advantages of Frequency domain plot:
  - ✓Complete sine wave is just denoted by a spike. Hence, easy to see values of frequency and amplitude.
  - ✓More compact and useful when dealing with more than one sine wave (composite signal).



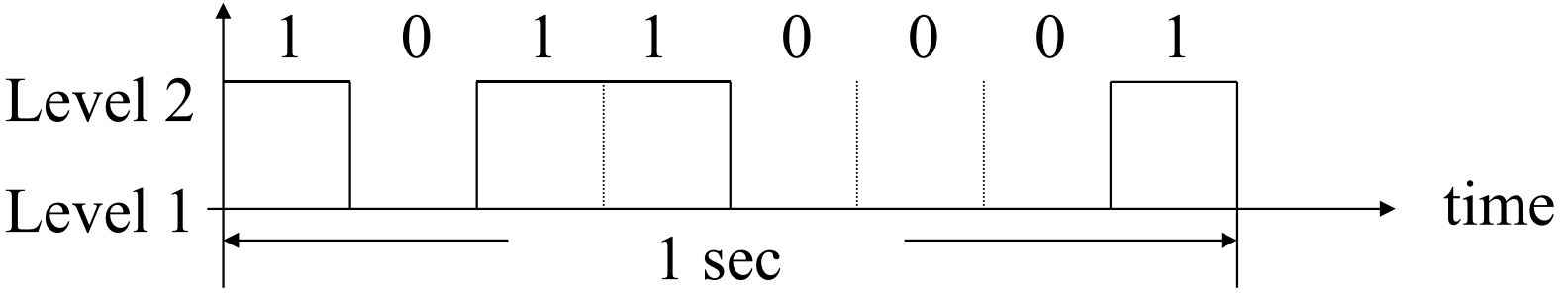
- Single sine wave is not sufficient for sending data. We need composite(consisting of many sine waves) signal to communicate data.
- Composite signals can be periodic or non periodic.
- A periodic composite signal can be decomposed into a series of simple sine waves with discrete frequencies.

  A non periodic composite signal can be decomposed into a combination of an infinite number of simple sine waves with continuous frequencies.

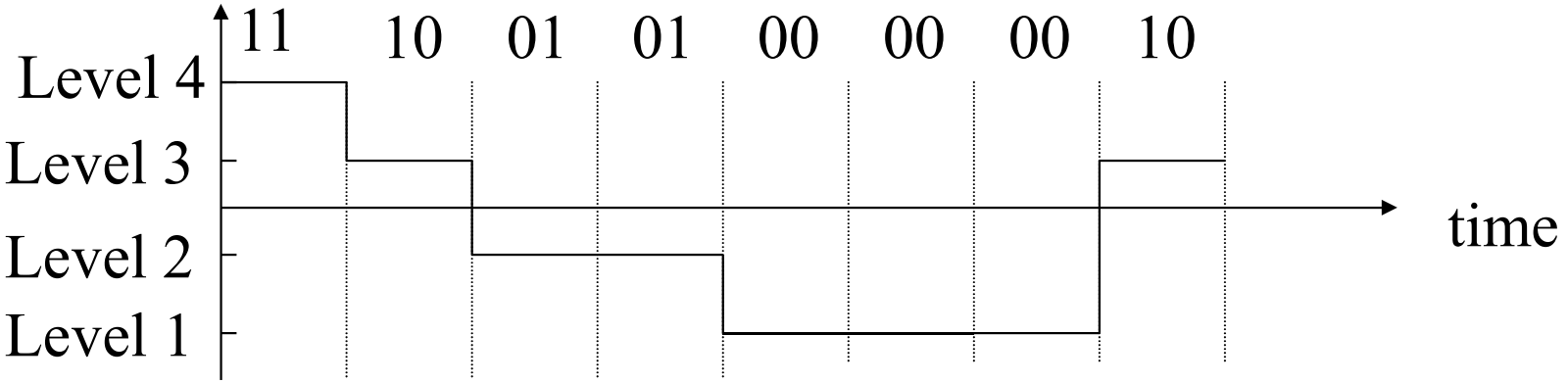
- The range of frequencies contained in a composite signal is its Bandwidth.

  It is basically difference between highest frequency and lowest frequency in a composite signal.
- Digital Signals:
  - ✓ Information can be represented using digital signal. (‘1’ encoded as +ve voltage and ‘0’ encoded as zero voltage).
  - ✓ A digital signal can have more than two levels. In this case we can send more than one bit for each level.

- Ex:



A signal with 2 signal levels, 8 bits sent in 1 sec



A signal with 4 signal levels, 16 bits sent in 1 sec

In general if signal has L levels, each level needs $\log_2 L$ bits.

- Bit Rate: Most digital signals are non periodic, thus frequency and period are not appropriate characteristics. Bit Rate (instead of frequency) is used to describe the digital signals.

  It is the number of bits sent in one second. Expressed in ‘bps’.

- Bit Length: It is the distance one bit occupies in the transmission medium.

  Bit Length=

  propagation speed * bit duration

Transmission Impairment

- Transmission media are not perfect. Imperfection causes signal impairment.

  Three causes of impairment:

  Attenuation, Distortion and Noise

  ✓ Attenuation: It means loss of energy. A Signal (simple or composite) loses some of its energy in overcoming resistance of the medium.

  Amplifiers are used to amplify and restore the signal. To show the loss or gain of strength of a signal, the unit used is decibel (dB). It measures the relative strengths of two signals or one signal at two different points.

$dB=10 \log_{10} P_2/P_1$ or $dB=20 \log_{10} V_2/V_1$

- ✓ Distortion: occurs when signal changes its form and shape.

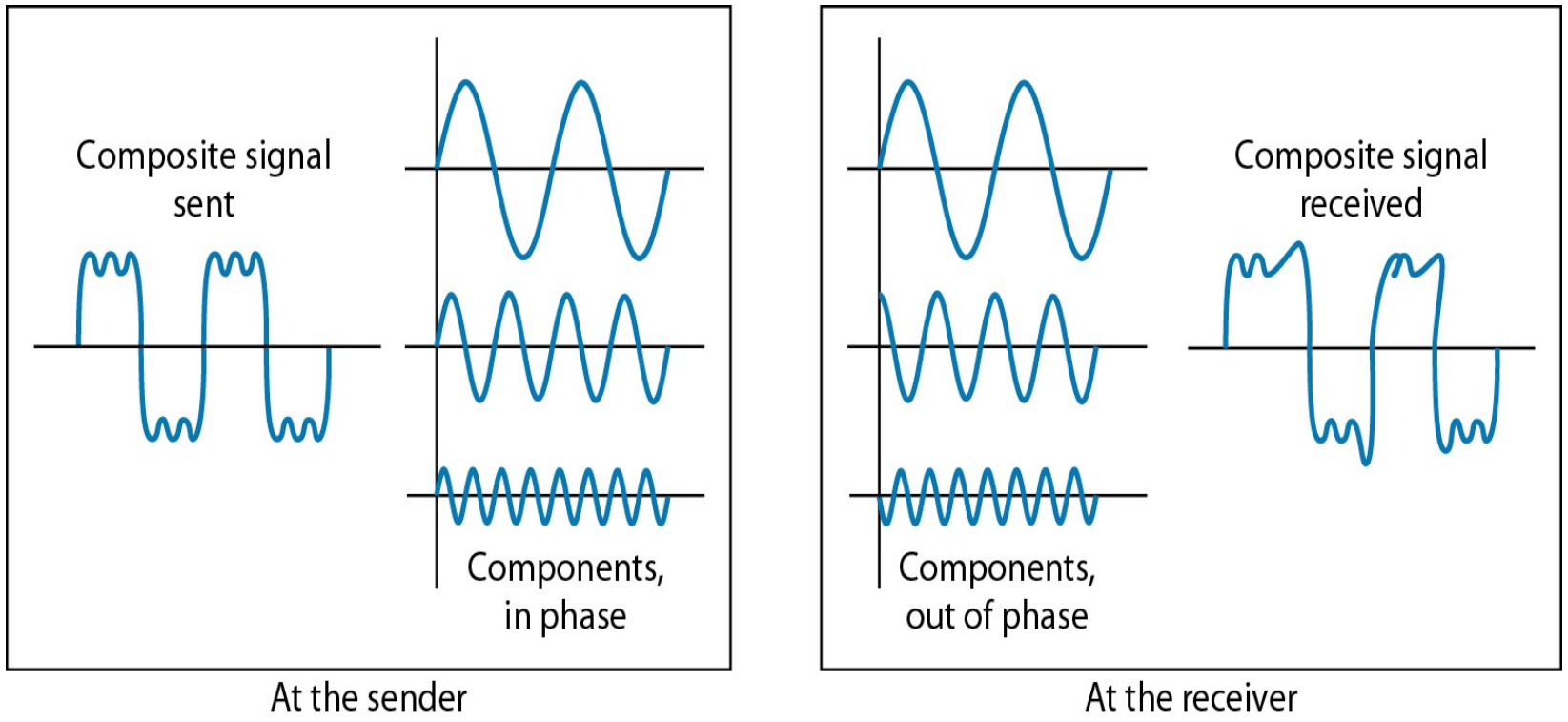
It can occur in a composite signal made of different frequencies.

Each signal component has its own propagation speed through the medium and hence its own delay in arriving at the destination

Differences in delay may create differences in phase if the delay is not exactly the same as period duration!!!

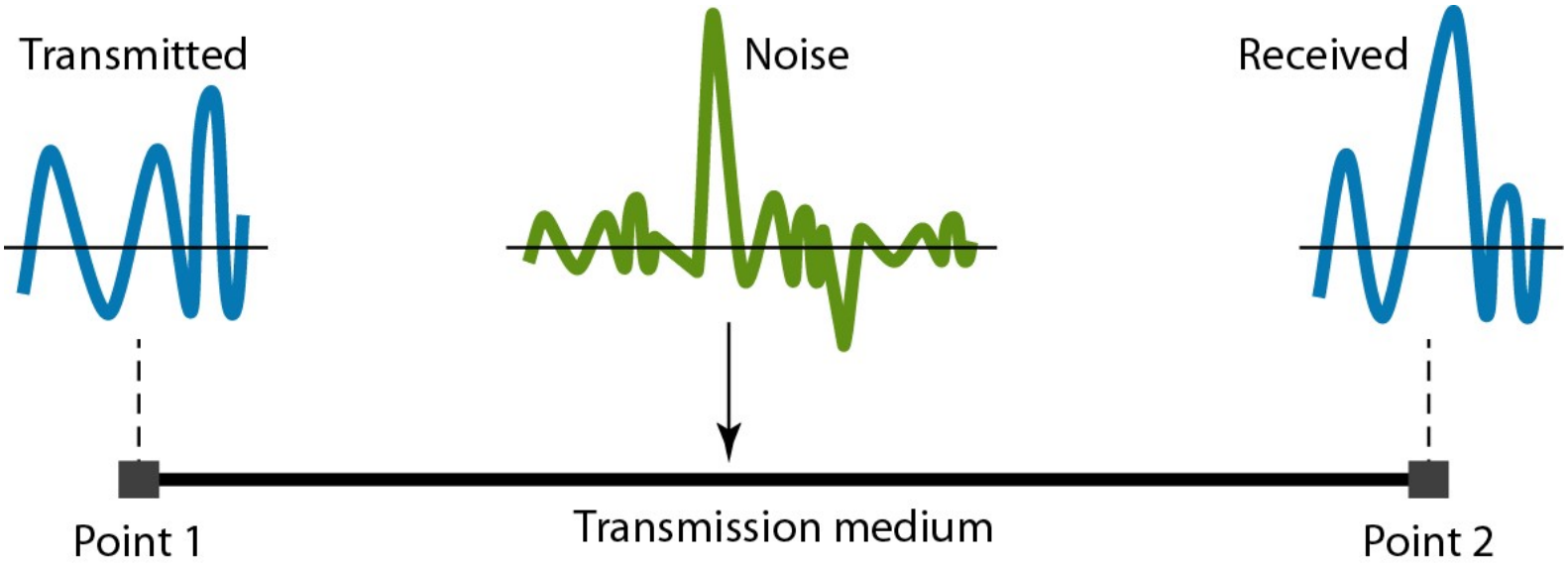
The shape of the received signal then is not same as that send.

Distortion


Composite signal
sent
Components,
in phase
At the sender
Composite signal
received
Components,
out of phase
At the receiver

Noise

- ✓Its an unwanted signal
- ✓Several types: Thermal noise, Induced noise, Cross talk and Impulse noise



Signal to noise ratio(SNR) is the ratio of what is wanted (signal) To what is not wanted (Noise),

SNR= avg. signal power/ avg. noise power

- Since SNR is the ratio of powers, it is expressed in decibels:

$$SNR_{dB}=10\log_{10}SNR$$

Ex: The power of a signal is 10 mW and the power of the noise is 1 μW; what are the values of SNR and SNR_{dB} ?

Solution: The values of SNR and SNR_{dB} can be calculated as follows:

$$SNR = \frac{10{,}000\ \mu W}{1\ mW} = 10{,}000$$

$$SNR_{dB} = 10 \log_{10} 10{,}000 = 10 \log_{10} 10^4 = 40$$

What is the SNR and SNR_{dB} for noiseless channel?

- DATA RATE LIMITS

A very important consideration in data communications is how fast we can send data, in bits per second, over a channel. Data rate depends on three factors:

1. The bandwidth available
2. The level of the signals we use
3. The quality of the channel (the level of noise)

Two theoritical formulas were developed to calculate the data rate:

By Nyquist for noiseless channel and

By Shannon for noisy channel

- Nyquist bit rate formula for noiseless channel defines theoretical maximum bit rate:

  Bit Rate= 2*bandwidth*$\log_2 L$

  L- No of signal levels used to represent data.
- Increasing the levels of signal may reduce the reliability of the system.
- Consider a noiseless channel with a bandwidth of 3000 Hz transmitting a signal with two signal levels. The maximum bit rate can be calculated as

- Noiseless channel is ideal. Channels are always noisy.
- Shannon capacity gives the theoretical highest data rate for noisy channel.

  capacity(bits/sec)= bandwidth*$\log_2$(1+SNR)
- What happens if SNR is 0?.

  Ex: The signal-to-noise ratio is often given in decibels. Assume that SNR_{dB} = 36 and the channel bandwidth is 2 MHz. Find the theoretical channel capacity.

$$SNR_{dB} = 10 \log_{10} SNR \rightarrow SNR = 10^{SNR_{dB}/10} \rightarrow SNR = 10^{3.6} = 3981$$

$$C = B \log_2 (1+ SNR) = 2 \times 10^6 \times \log_2 3982 = 24 \text{ Mbps}$$

- Both limits: In practice we need to use both methods to find the limits and signal levels!!
- The Shannon capacity gives us the upper limit; the Nyquist formula tells us how many signal levels we need.
- Ex: We have a channel with a 1-MHz bandwidth. The SNR for this channel is 63. What are the appropriate bit rate and signal level?

Solution:

First, we use the Shannon formula to find the upper limit.

$$C = B \log_2 (1 + \text{SNR}) = 10^6 \log_2 (1 + 63) = 10^6 \log_2 64 = 6 \text{ Mbps}$$

The Shannon formula gives us 6 Mbps, the upper limit. For better performance we choose something lower, 4 Mbps, for example. Then we use the Nyquist formula to find the number of signal levels.

$$4 \text{ Mbps} = 2 \times 1 \text{ MHz} \times \log_2 L \quad \rightarrow \quad L = 4$$

PERFORMANCE

One important issue in networking is the performance of the network- how good is it?

*In networking, we use the term **bandwidth** in two contexts.*

- The first, bandwidth in hertz, refers to the range of frequencies in a composite signal or the range of frequencies that a channel can pass.

- The second, bandwidth in bits per second, refers to the speed of bit transmission in a channel or link.

- **Throughput:** It's a measure of how fast actually we can send the data through the network.
- Appears to be same as bandwidth?? .

  Not exactly, bandwidth is the potential measurement of a link and throughput is the actual measurement of how fast we can send the data.
  Ex: A network with bandwidth of 10 Mbps can pass only an average of 12,000 frames per minute with each frame carrying an average of 10,000 bits. What is the throughput of this network?
  Solution:
  We can calculate the throughput as

$$\text{Throughput} = \frac{12{,}000 \times 10{,}000}{60} = 2 \text{ Mbps}$$

- **Latency** (Delay): It defines how long it takes for an entire message to completely arrive at the destination from the time the first bit is sent out from the source.
- It is made up of four components:

Propagation time(PT), transmission time(TT), queuing time(QT) and processing delay.

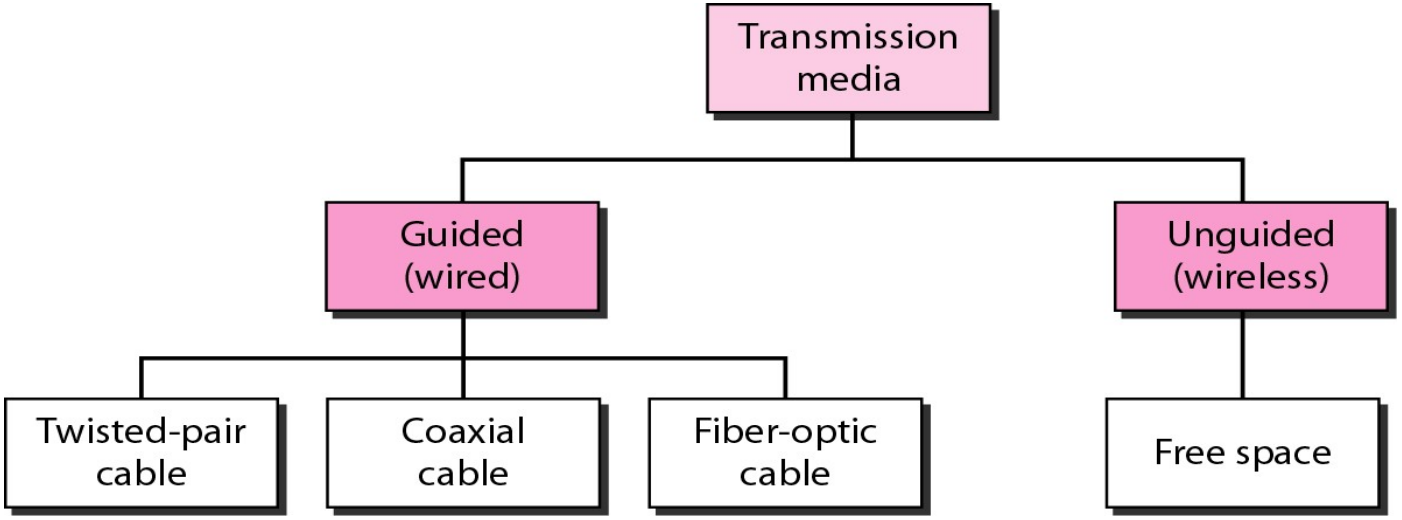
PT= Distance/Propagation speed

TT= Message size/Bandwidth

QT= Time needed by the intermediate or end device to hold message before it can be processed.

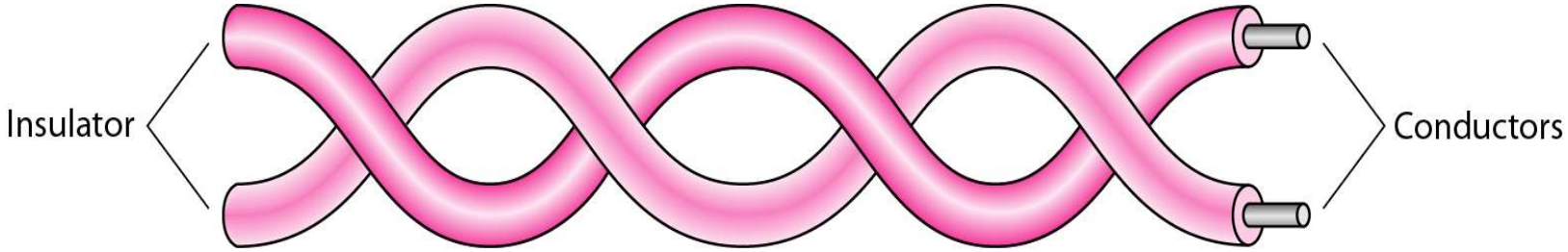
Transmission Media

- Basically used to transport a raw bit stream from one machine to another.

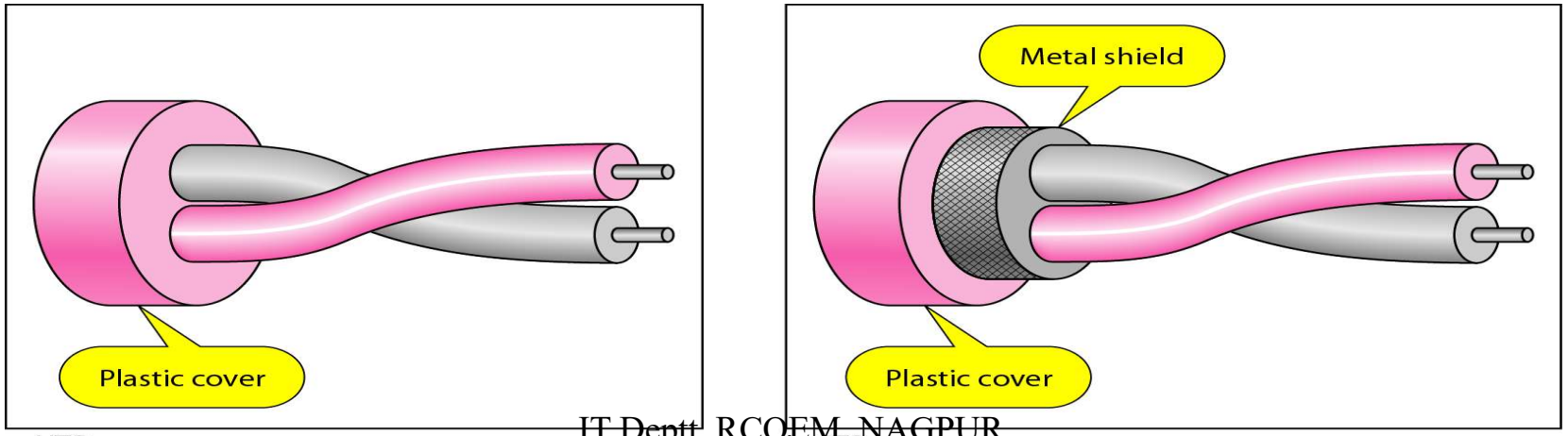


- Magnetic media can provide bandwidth better than any media!!!

- Guided Media:
  - ✓ Twisted pair:
    - o Oldest and most commonly used.
    - o Consists of two insulated copper wires, 1 mm thick twisted together. One wire is used to carry signals to receiver and the other is used as ground reference. The receiver uses the difference between the two.



- Interference and crosstalk affects both wires and create unwanted signals.
- If two wires are parallel, the effect of unwanted signals is not same in both wires. Twisting ensures that both are equally affected by external influences. The unwanted signals are mostly cancelled out.
- Two types of twisted pair cables:

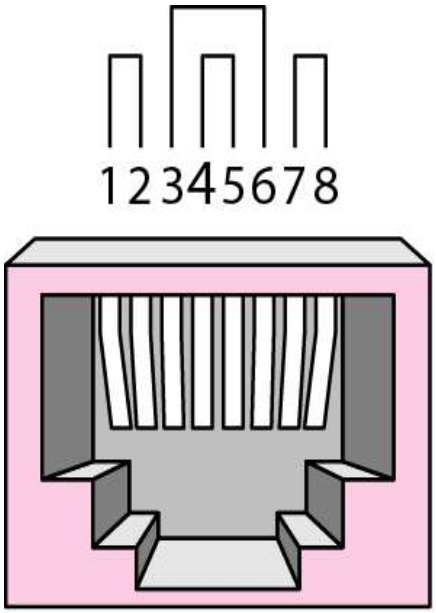


a. UTP b. STP

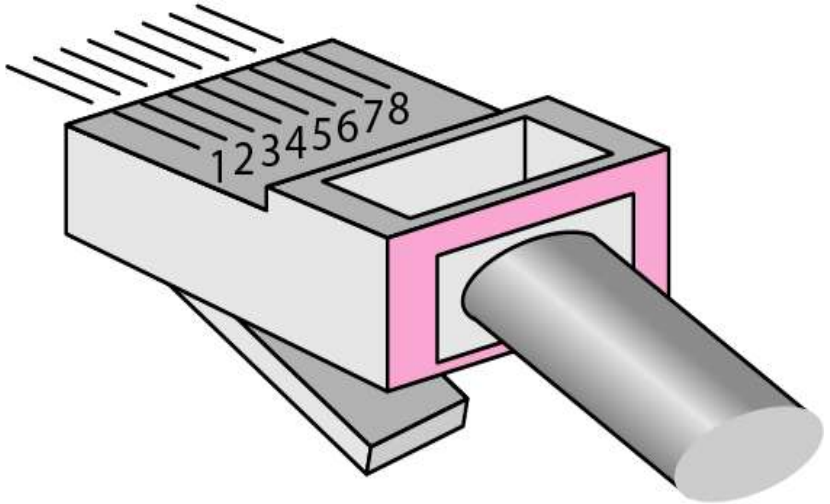
Categories of unshielded twisted-pair cables

Category	*Specification*	*Data Rate (Mbps)*	*Use*
1	Unshielded twisted-pair used in telephone	< 0.1	Telephone
2	Unshielded twisted-pair originally used in T-lines	2	T-1 lines
3	Improved CAT 2 used in LANs	10	LANs
4	Improved CAT 3 used in Token Ring networks	20	LANs
5	Cable wire is normally 24 AWG with a jacket and outside sheath	100	LANs
5E	An extension to category 5 that includes extra features to minimize the crosstalk and electromagnetic interference	125	LANs
6	A new category with matched components coming from the same manufacturer. The cable must be tested at a 200-Mbps data rate.	200	LANs
7	Sometimes called SSTP (shielded screen twisted-pair). Each pair is individually wrapped in a helical metallic foil followed by a metallic foil shield in addition to the outside sheath. The shield decreases the effect of crosstalk and increases the data rate.	600	LANs

- Connectors: Most common twisted pair connector is RJ45.



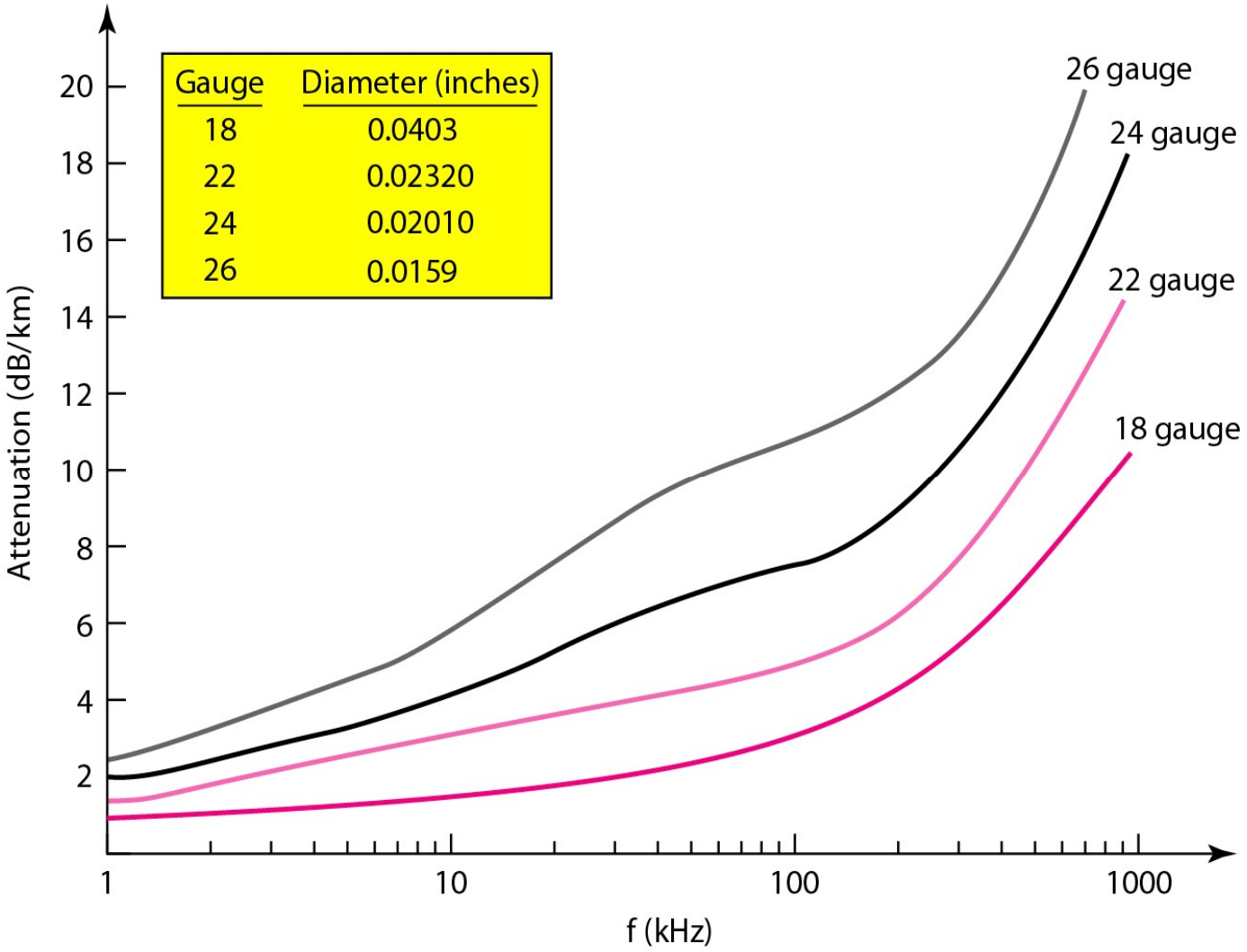
RJ-45 Female



RJ-45 Male

- Performance: Twisted pair cable can pass wide range of frequencies.

UTP performance



Recent cabling standard for LAN

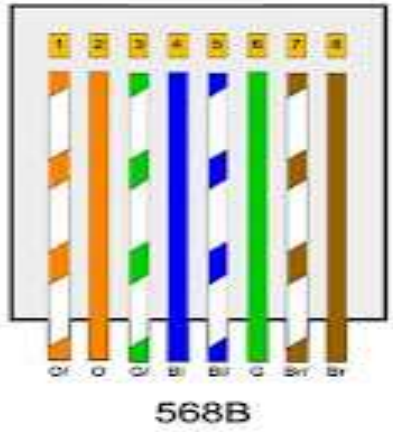
- CAT 5 cable
  - ✓ Frequency: 100MHz (Speed 100Mbps)
  - ✓ Cable segment length of 100 meters maximum
  - ✓ Uses 2 twisted pairs of wires (four) out of four such pairs, for sending and receiving data
- CAT 5e cable
  - ✓ Improved specification to the very popular Category 5 cable. The improvement was in noise reduction
  - ✓ speeds increased to 350 Mbit/s over 100 meters
  - ✓ All eight wires (4 pairs) are used. Cat 5e introduced and optimized encoding scheme that allows up to 50-meter lengths of Cat 5e cable to provide at or near Gigabit Ethernet (1000BASE-T) speeds.

- CAT 6 cable
  - ✓ Gigabit Ethernet (1000BASE-T)
  - ✓ The 6 cable uses thicker-gauge wire to attain the higher frequencies, it has increased shielding, and more pair twists per inch to reduce signal noise and interference.
  - ✓ The new tighter specifications introduced with Cat 6 cabling guarantee that 100-meter runs of Category 6 are capable of 1000 Mbit/s transfer speeds.
  - ✓ As with 5e reducing the cable length can achieve higher speeds than the category types design goal so 10-Gigabit Ethernet speeds can be achieved when reducing cable lengths to less than 50 meters.

Ethernet Cable Wiring Specification

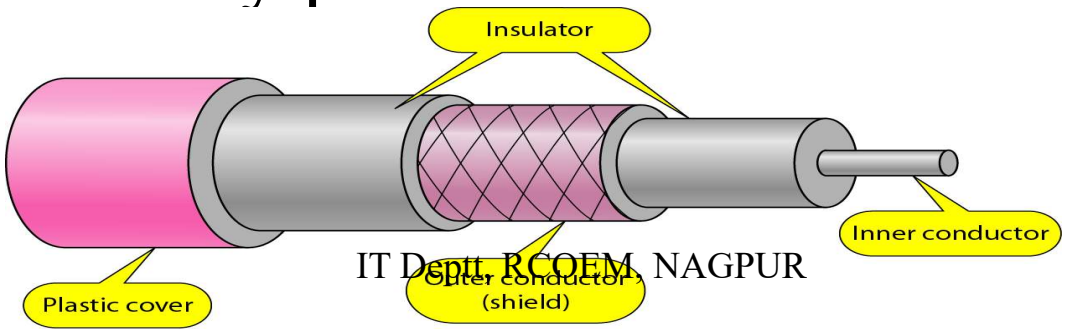
- All Ethernet cables use either the T568A or T568B wiring standard

Orange/White
Orange
Green/White
Blue
Blue/White
Green
Brown/White
Brown



Coaxial cable

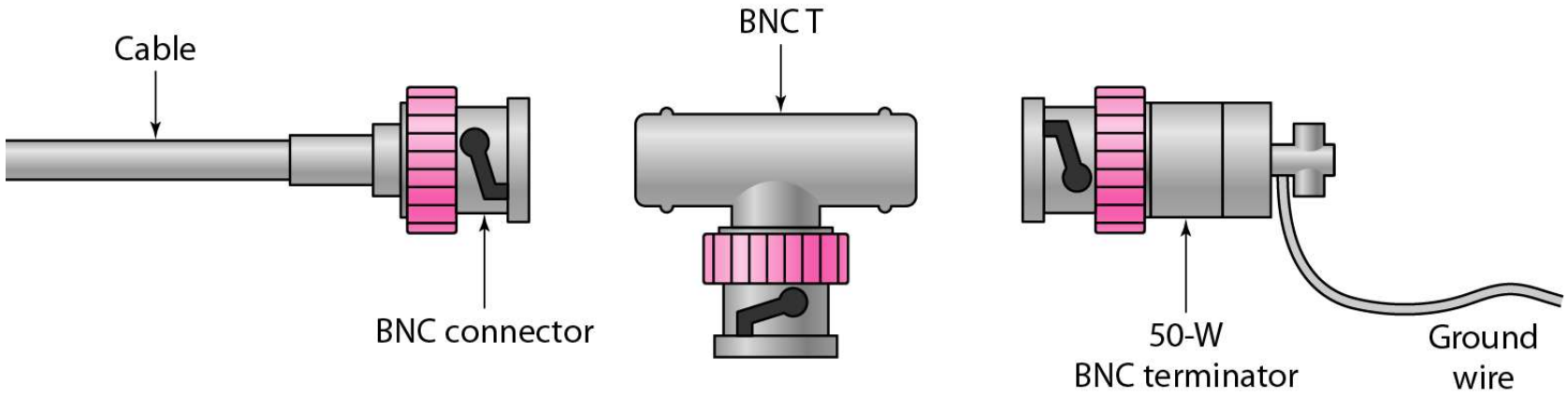
- Coaxial cable (or coax) carries signals of higher frequencies than UTP.
- Consists of a central core conductor of solid or stranded wire enclosed in an insulating sheath, which in turn is enclosed in an outer conductor of metal foil, braid or combination of two. This outer conductor is also enclosed in an insulating sheath and the whole cable is protected by plastic cover.



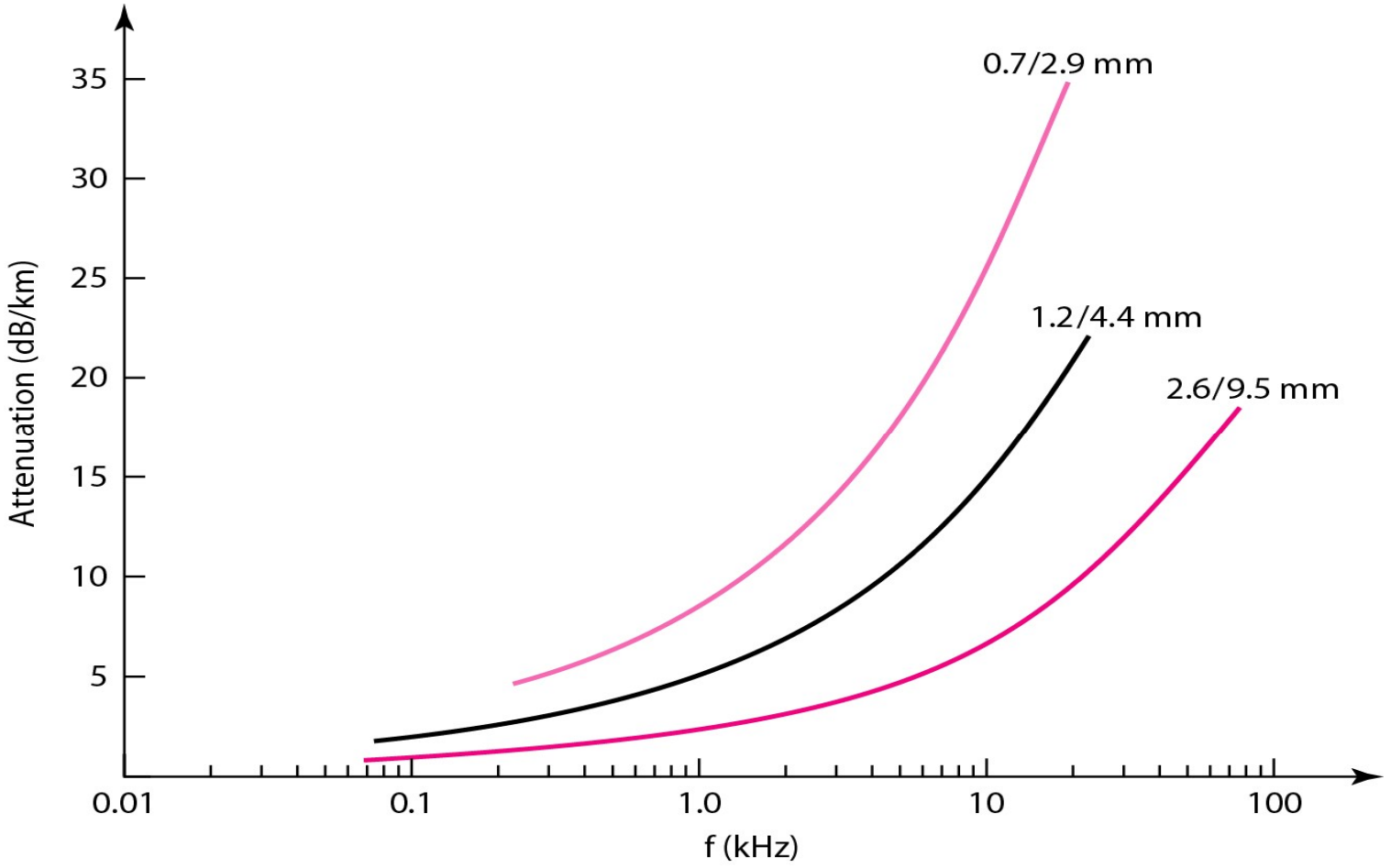
Categories of coaxial cables

Category	*Impedance*	*Use*
RG-59	75 Ω	Cable TV
RG-58	50 Ω	Thin Ethernet
RG-11	50 Ω	Thick Ethernet

BNC (Bayone Neill Concelman) connectors

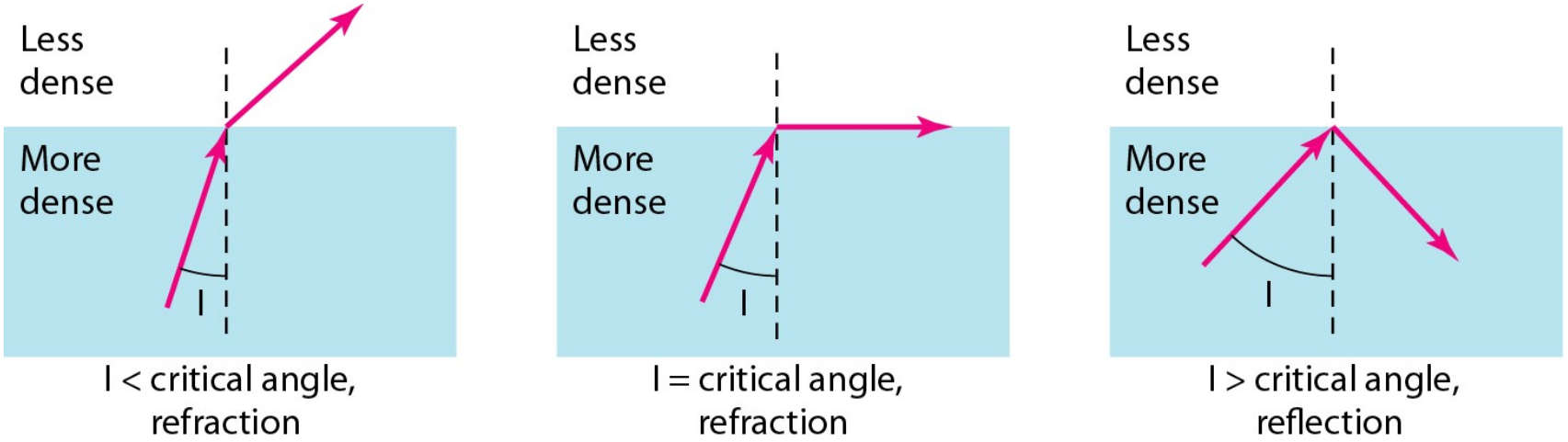


- Performance of coaxial cable:

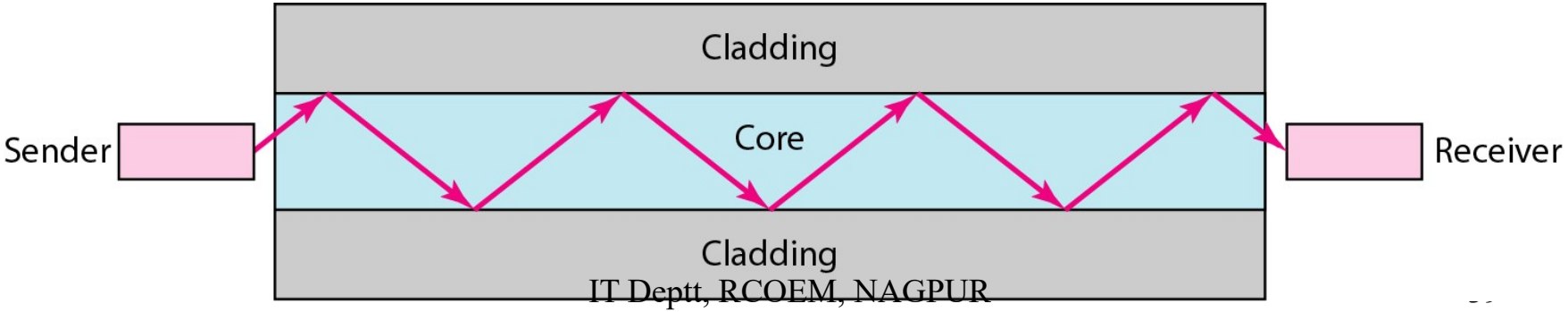


Fiber Optic cable

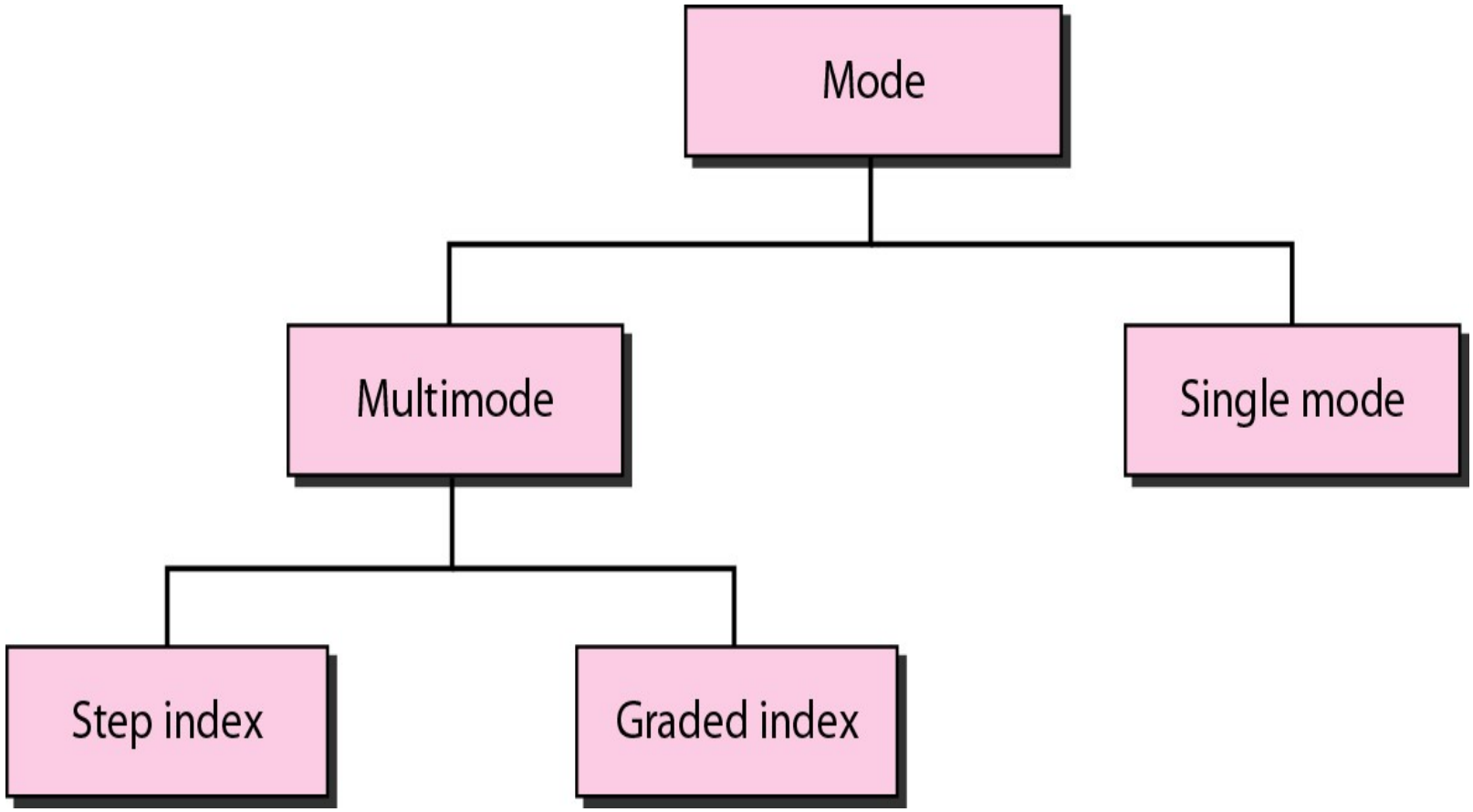
Bending of light ray:



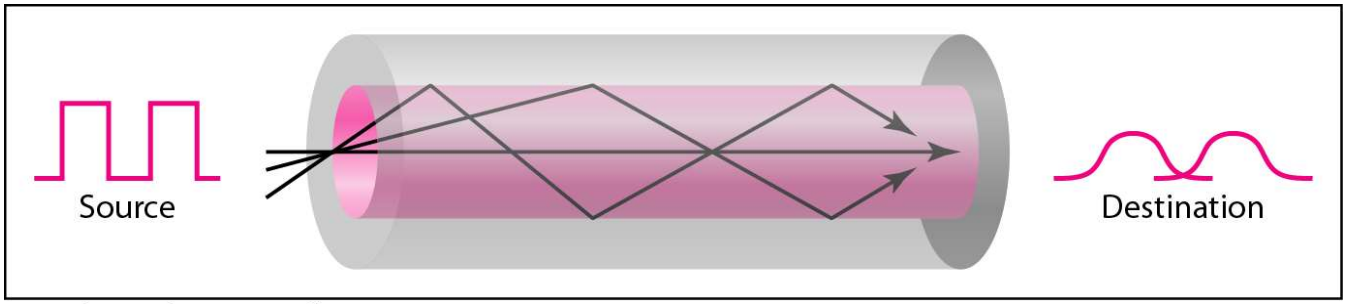
Optical fiber:



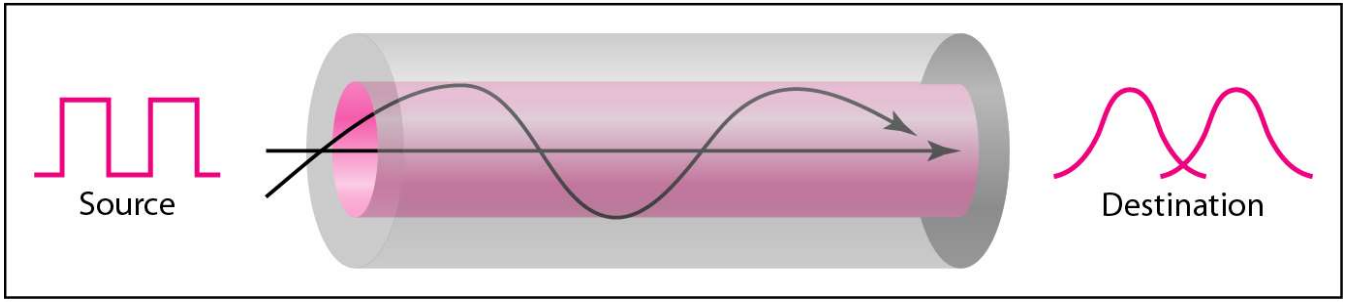
- Propagation modes:



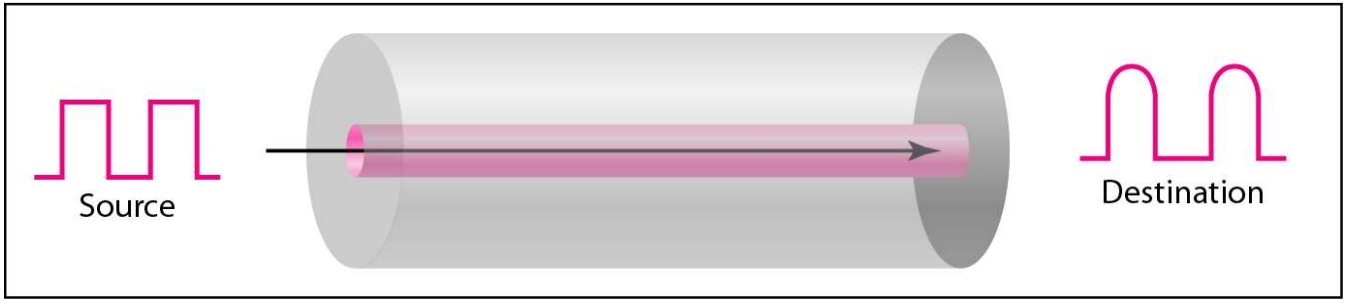
- *Modes*



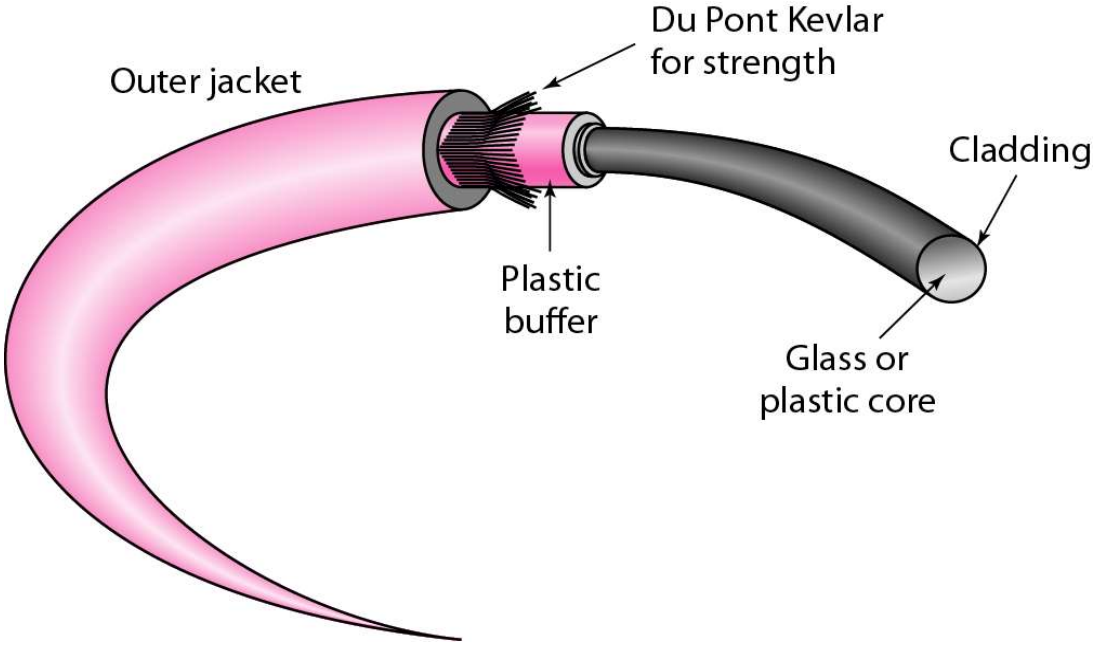
a. Multimode, step index



b. Multimode, graded index



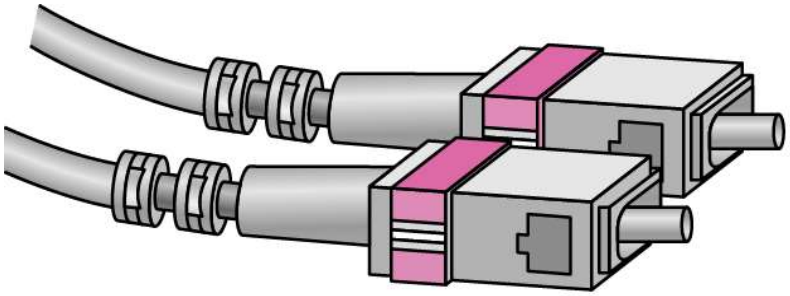
c. Single mode



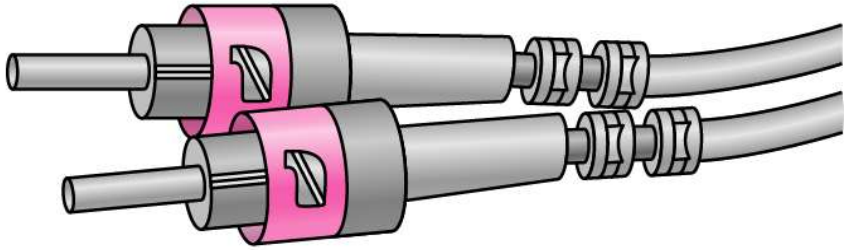
Fiber types

Type	*Core (μm)*	*Cladding (μm)*	*Mode*
50/125	50.0	125	Multimode, graded index
62.5/125	62.5	125	Multimode, graded index
100/125	100.0	125	Multimode, graded index
7/125	7.0	125	Single mode

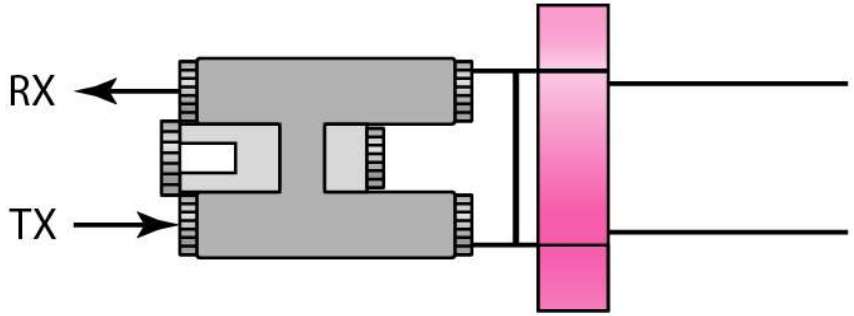
Fiber-optic cable connectors



SC connector

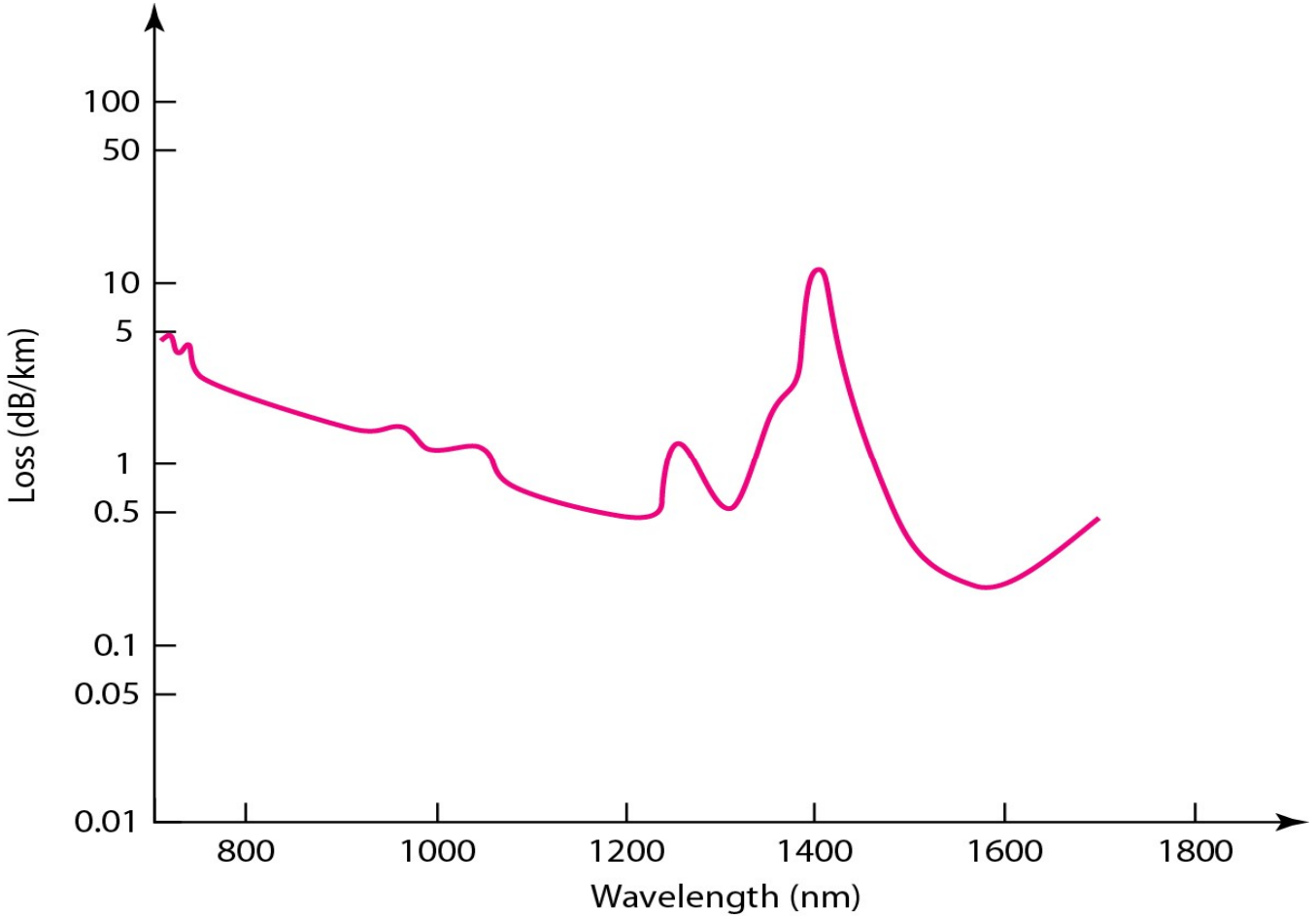


ST connector



MT-RJ connector

Optical fiber performance



Applications : Used in backbone networks.

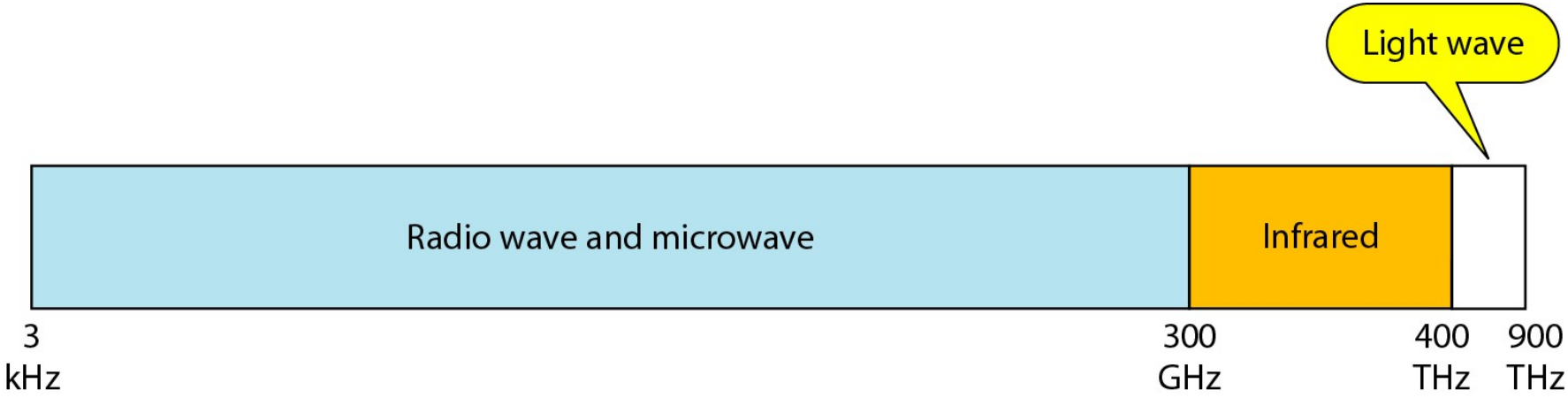
Combination of fiber optic cable and coaxial cable is also used.

- Advantages :
  - ✓ Higher bandwidth
  - ✓ Less signal attenuation
  - ✓ Immunity to electromagnetic noise
  - ✓ Light weight
  - ✓ Greater immunity to tapping

- Disadvantages :
  - ✓ Installation and maintenance
  - ✓ Unidirectional
  - ✓ Costly

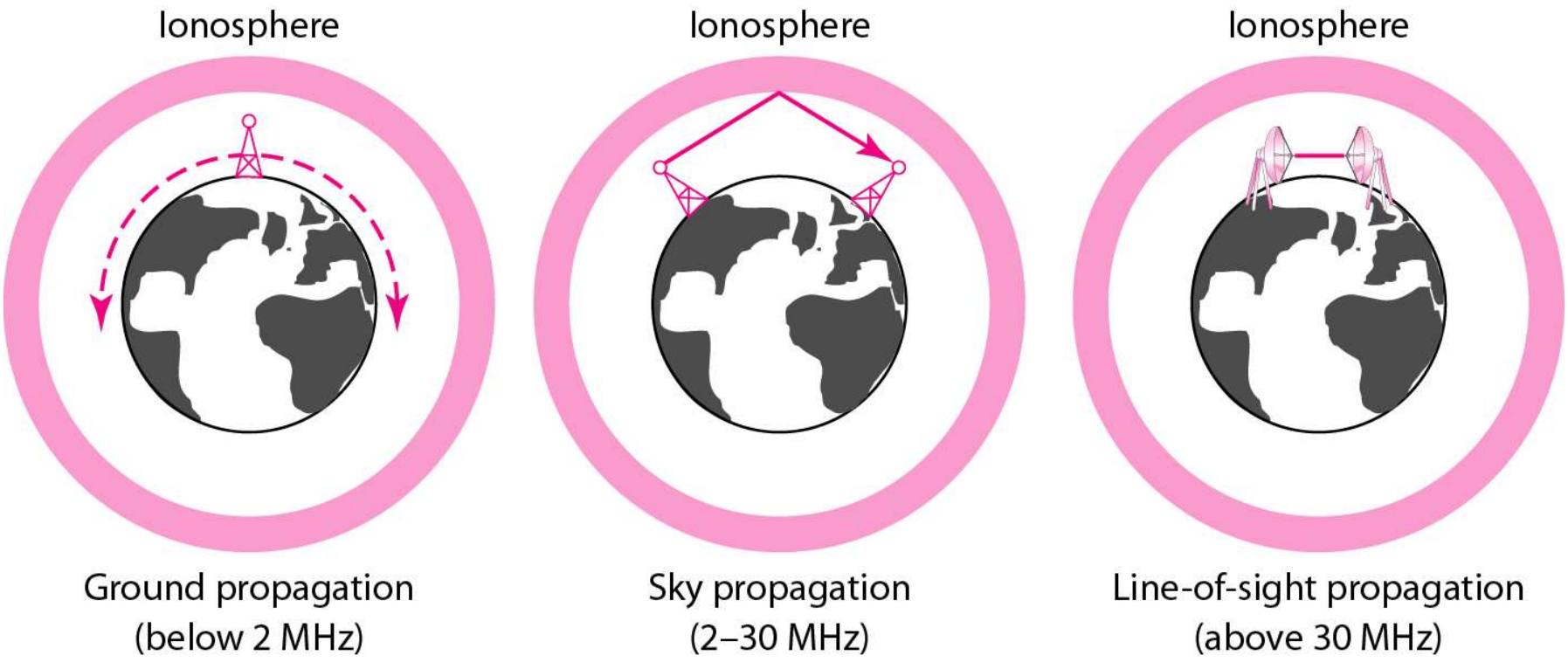
Unguided Media: WIRELESS

- Unguided media transport electromagnetic waves without using physical conductor.
- Electromagnetic spectrum used for wireless communication:



- Unguided signals can travel from source to destination in several ways: ground, sky and Line of sight propagation

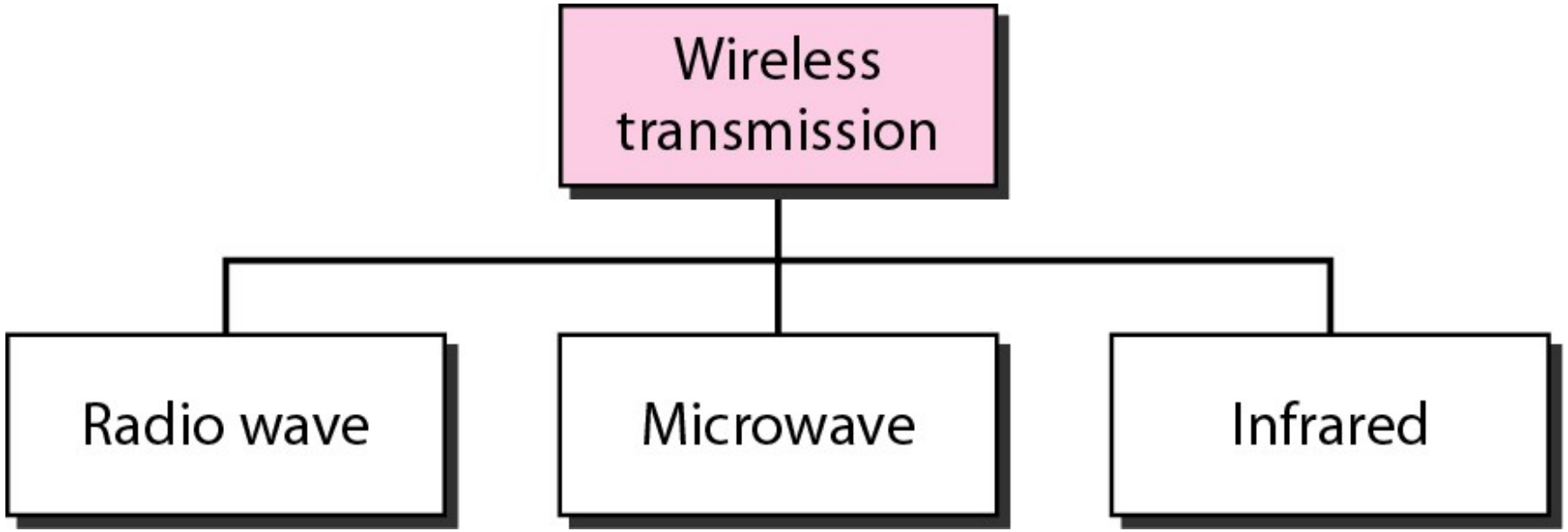
Propagation Methods



Frequency Bands

Band	*Range*	*Propagation*	*Application*
VLF (very low frequency)	3–30 kHz	Ground	Long-range radio navigation
LF (low frequency)	30–300 kHz	Ground	Radio beacons and navigational locators
MF (middle frequency)	300 kHz–3 MHz	Sky	AM radio
HF (high frequency)	3–30 MHz	Sky	Citizens band (CB), ship/aircraft communication
VHF (very high frequency)	30–300 MHz	Sky and line-of-sight	VHF TV, FM radio
UHF (ultrahigh frequency)	300 MHz–3 GHz	Line-of-sight	UHF TV, cellular phones, paging, satellite
SHF (superhigh frequency)	3–30 GHz	Line-of-sight	Satellite communication
EHF (extremely high frequency)	30–300 GHz	Line-of-sight	Radar, satellite

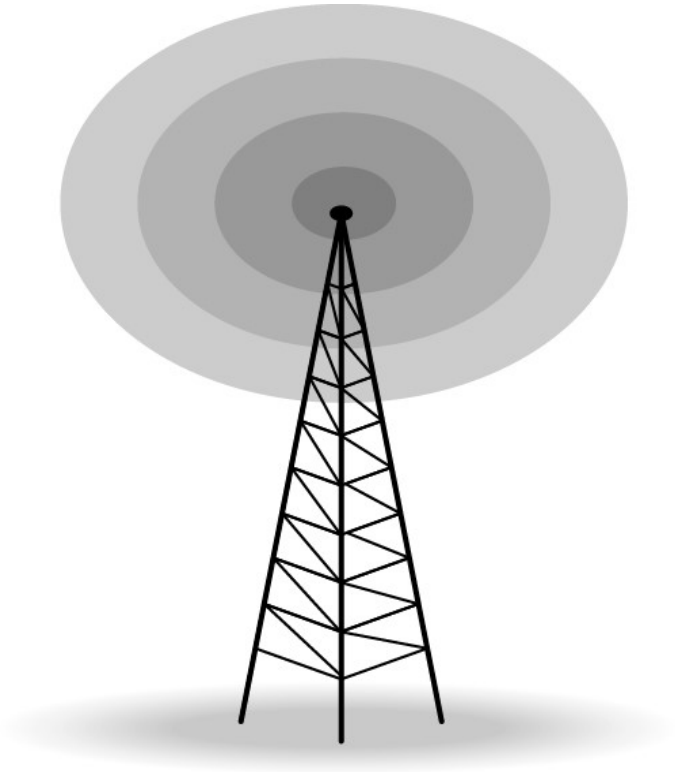
Wireless transmission waves



Radio Waves

- Frequency range: 3KHz to 1GHz
- They are Omnidirectional
  - ✓ Alignment of antennas not required
  - ✓ Interference problem
- Radio waves with low and medium frequency can penetrate walls
- Band is narrow hence lower data rate
- Radio waves propogating in sky mode can travel long distances and hence are used for multicast communications, such as radio and television, cordless phones and paging systems
- Omnidirectional antennas are required

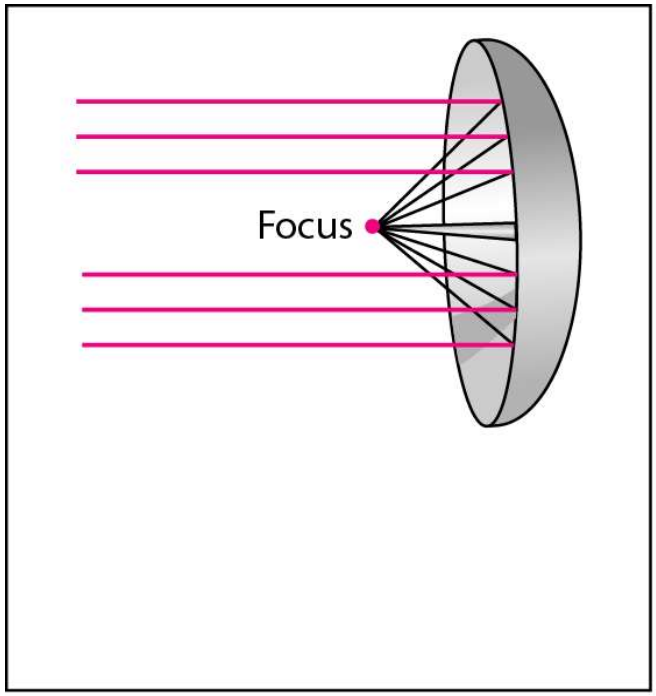
Omnidirectional antenna



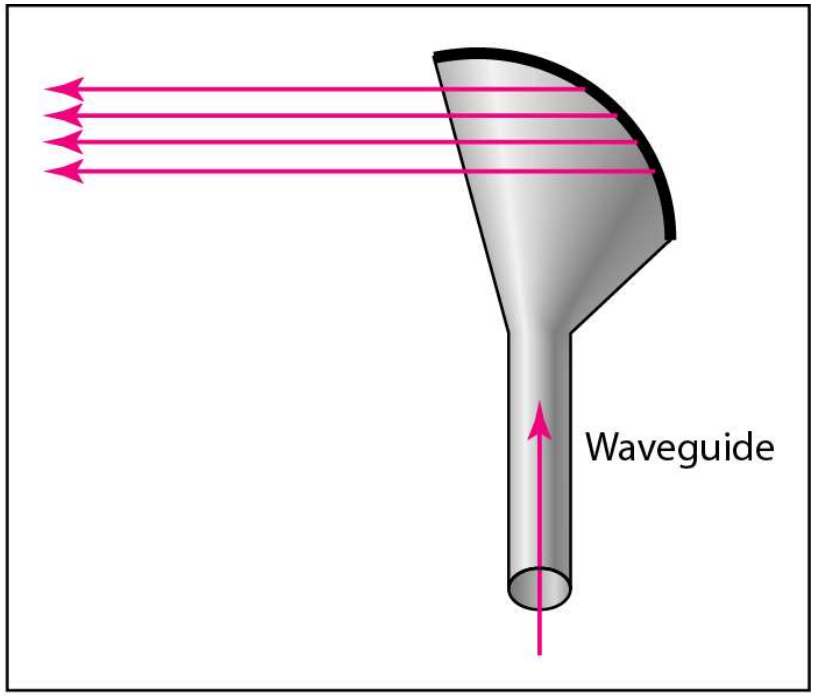
Microwaves

- Electromagnetic waves with frequency:
  1 to 300 GHz
- They are Unidirectional (antennas have to be aligned)
- No interference with other antennas
- High frequency waves cannot penetrate walls
- Propagation is line of sight
- Frequency Band is wider
- Microwaves are used for unicast communication such as cellular telephones, satellite networks and wireless LANs.
- Unidirectional antennas are required

Unidirectional antennas



a. Dish antenna



b. Horn antenna

Infrared waves

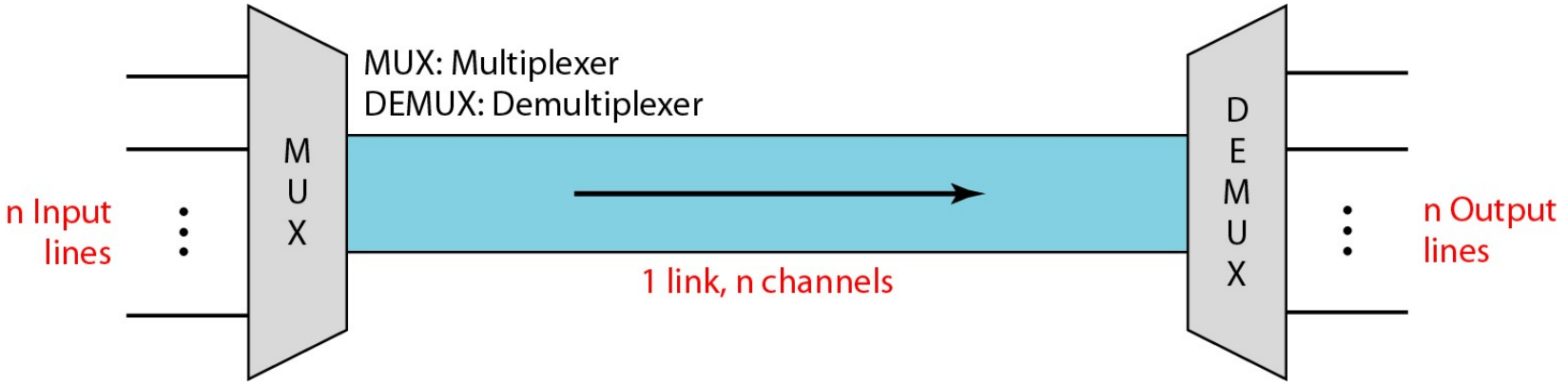
- Frequencies: 300 GHz to 400 THz
- Infrared signals can be used for short-range communication in a closed area using line-of-sight propagation
- High frequency cannot penetrate walls
- Cannot be used outside
- Higher data rates are possible
- Infrared Data Association (IrDA) proposes standard for communication between small devices

Multiplexing

- Whenever the bandwidth of a medium linking two devices is greater than the bandwidth needs of the devices, the link can be shared.
- Multiplexing is the set of techniques that allows the simultaneous transmission of multiple signals across a single data link. As data and telecommunications use increases, so does traffic.

Thus, multiplexing maximizes utilization of bandwidth.

- In multiplexed system, n lines share the bandwidth of one link.



- Three basic multiplexing techniques:
  1. Frequency division Multiplexing (Analog)
  2. Wavelength division Multiplexing (Analog) and
  3. Time division Multiplexing (Digital)

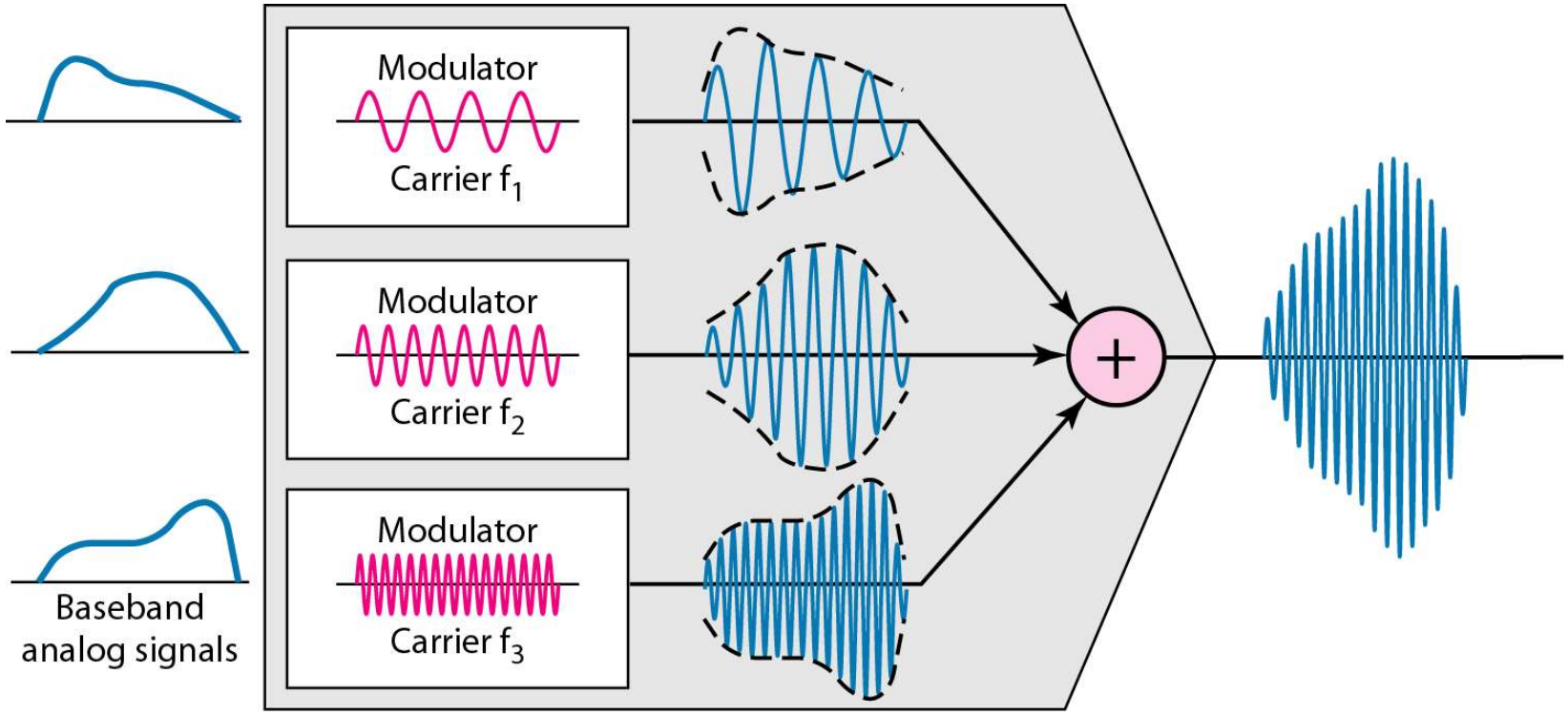
Frequency Division Multiplexing

- FDM is an analog multiplexing technique that combines analog signals.
- It can be applied when the bandwidth of a link (in Hz) is greater than the combined bandwidth of the signals to be transmitted.
- In FDM, signals generated by each sending device modulate different carrier frequency. These modulated signals are then combined into a single composite signal that can be transported by the link.

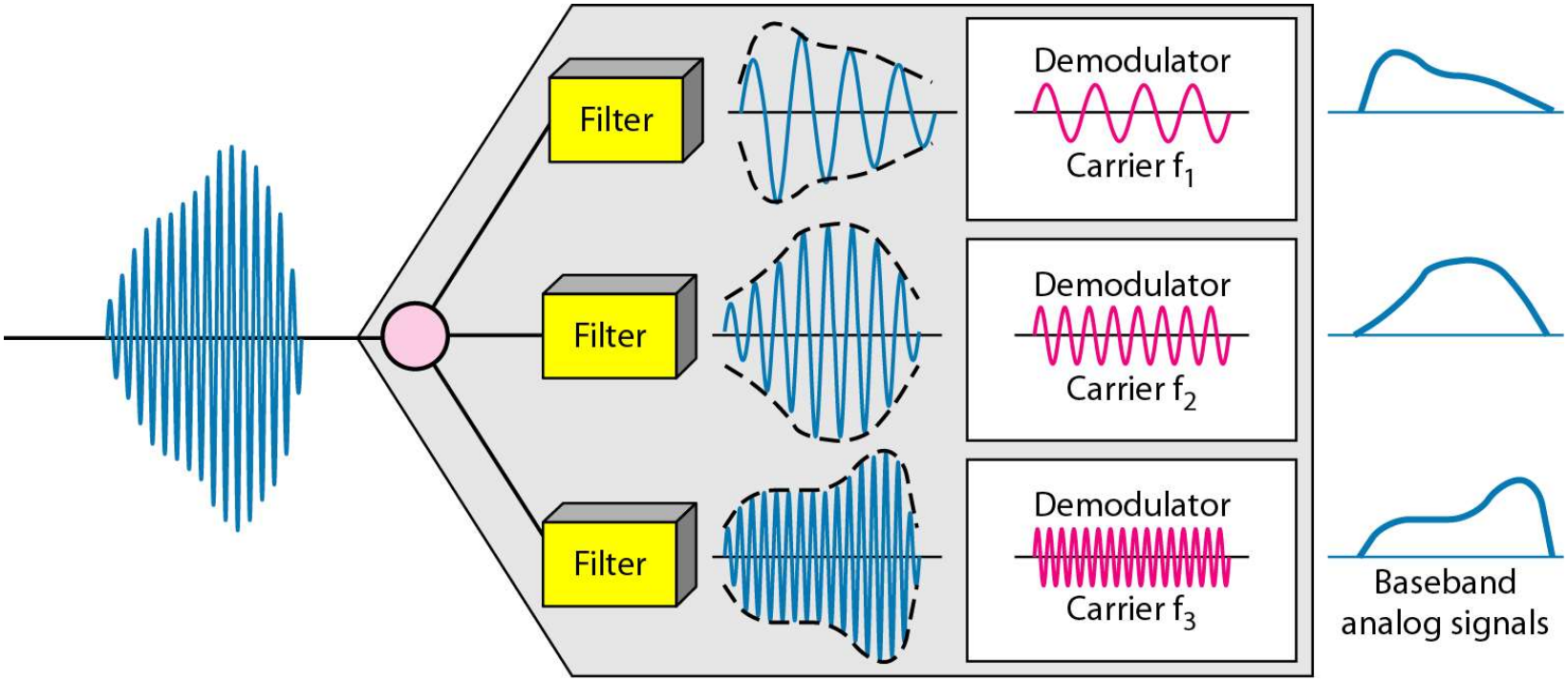
Frequency-division multiplexing

- Channels are separated by strips of unused bandwidth called Guard bands, to prevent the signals from overlapping.

FDM process: Multiplexing

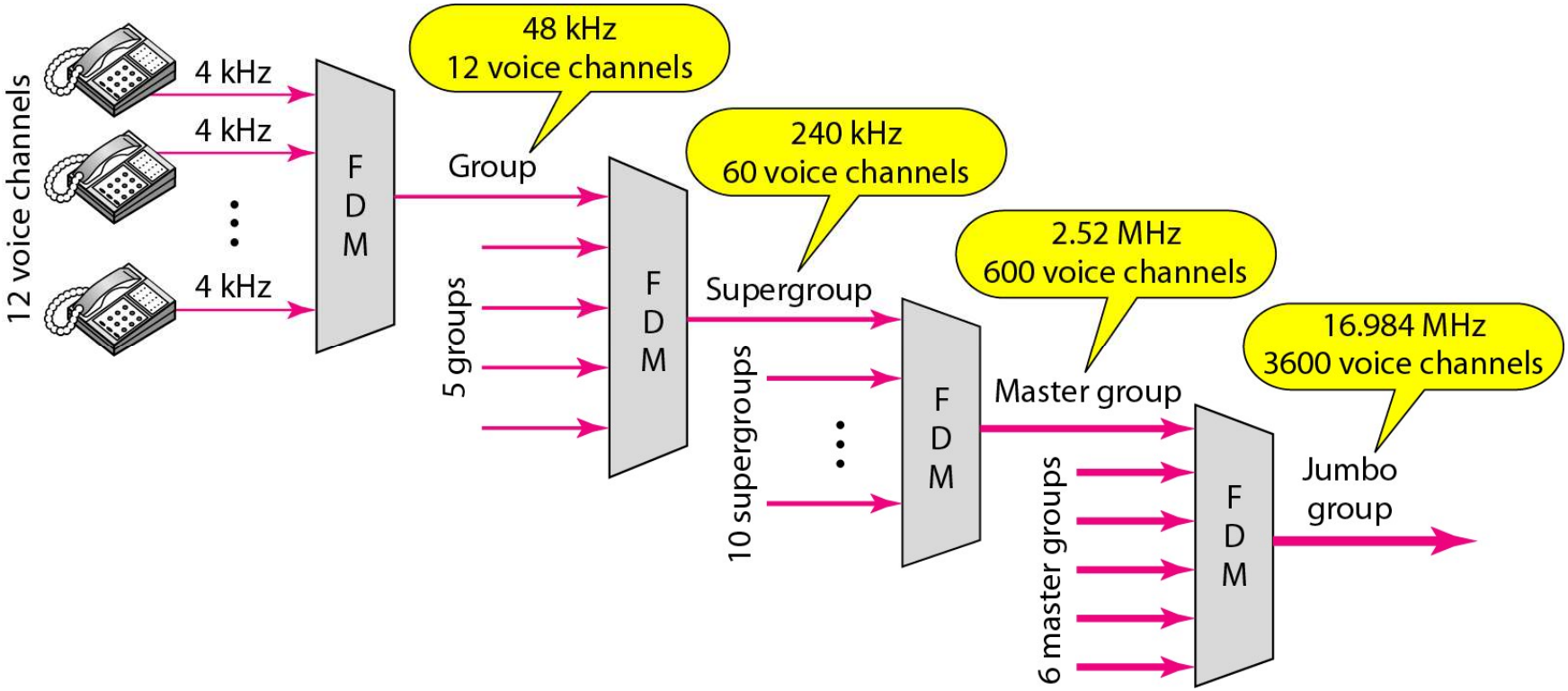


FDM process: Demultiplexing



- To maximize the efficiency of their infrastructure Telephone companies use multiplexing.
  for ex the hierarchical system at AT&T

Analog hierarchy

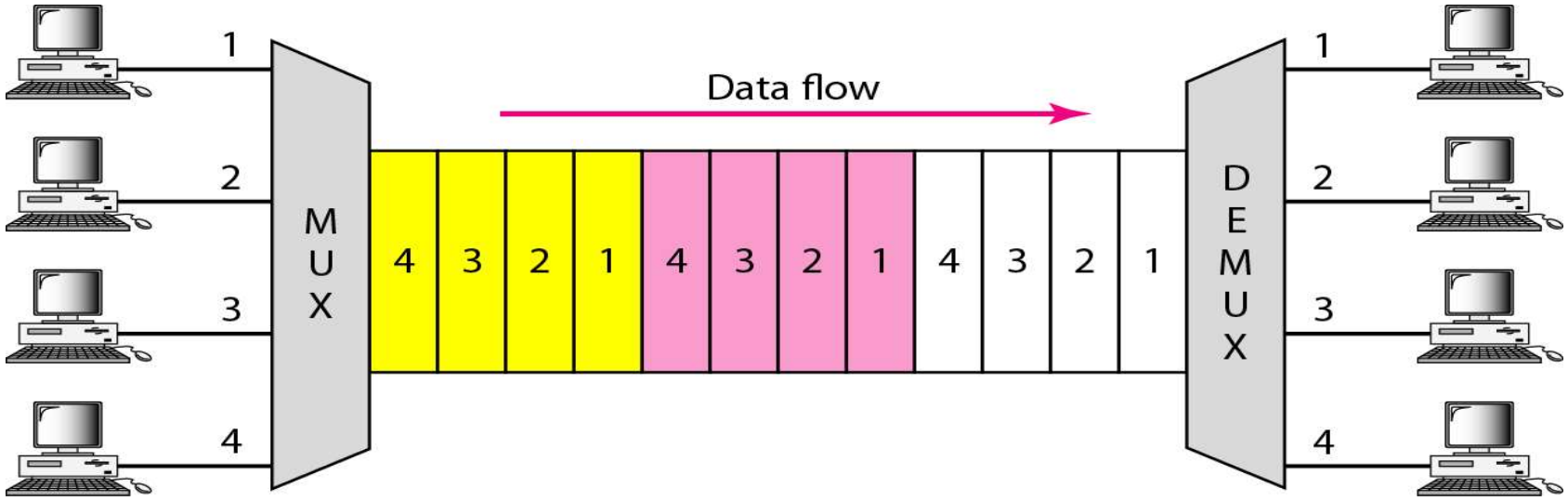


Applications: AM and FM Radios and TV broadcasting

Time division Multiplexing

- It's a digital process that allows several connections to share the high bandwidth of a link.
- Instead of sharing bandwidth of a link, as in FDM, time is shared.
- Each connection occupies portion of time in the link.
- Thus, TDM is a multiplexing technique for combining several low-rate channels into a high rate one.

TDM



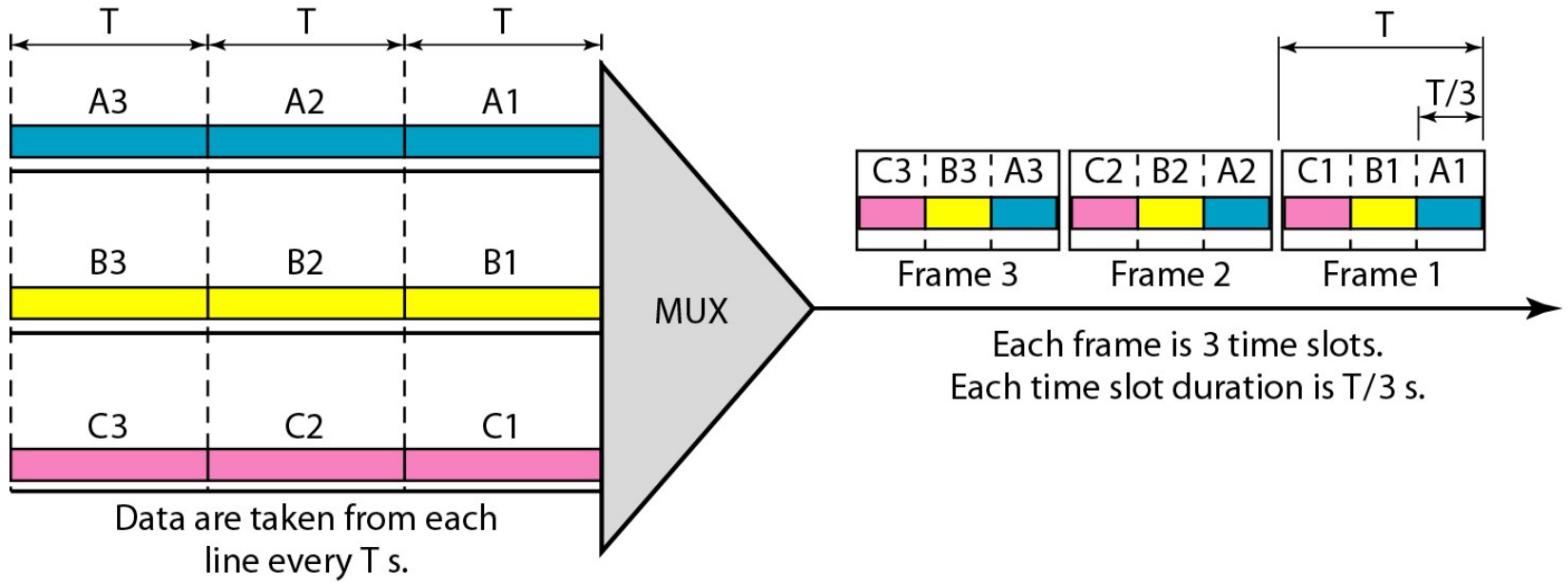
Two schemes in TDM:
Synchronous TDM and Statistical TDM

Synchronous TDM

- Data flow of each input connection is divided into units, where each unit occupies one input time slot.
- Each input becomes one output unit and occupies one output time slot.
- The duration of output time slot is n times shorter than the duration of an input time slot.

thus, if input time slot is T sec, the output time slot is t/n sec's, where n is the number of connections.

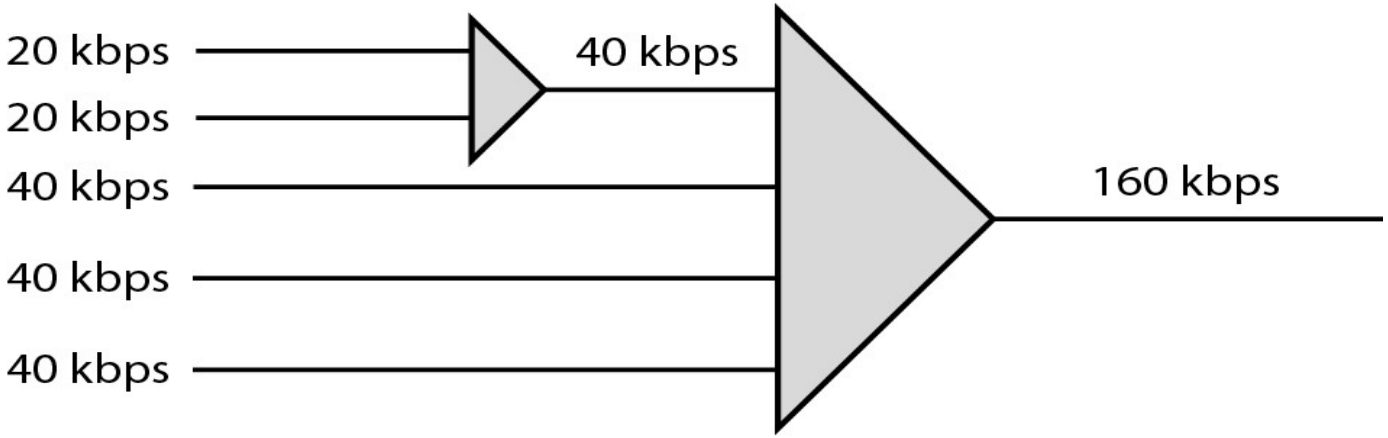
Synchronous time-division multiplexing



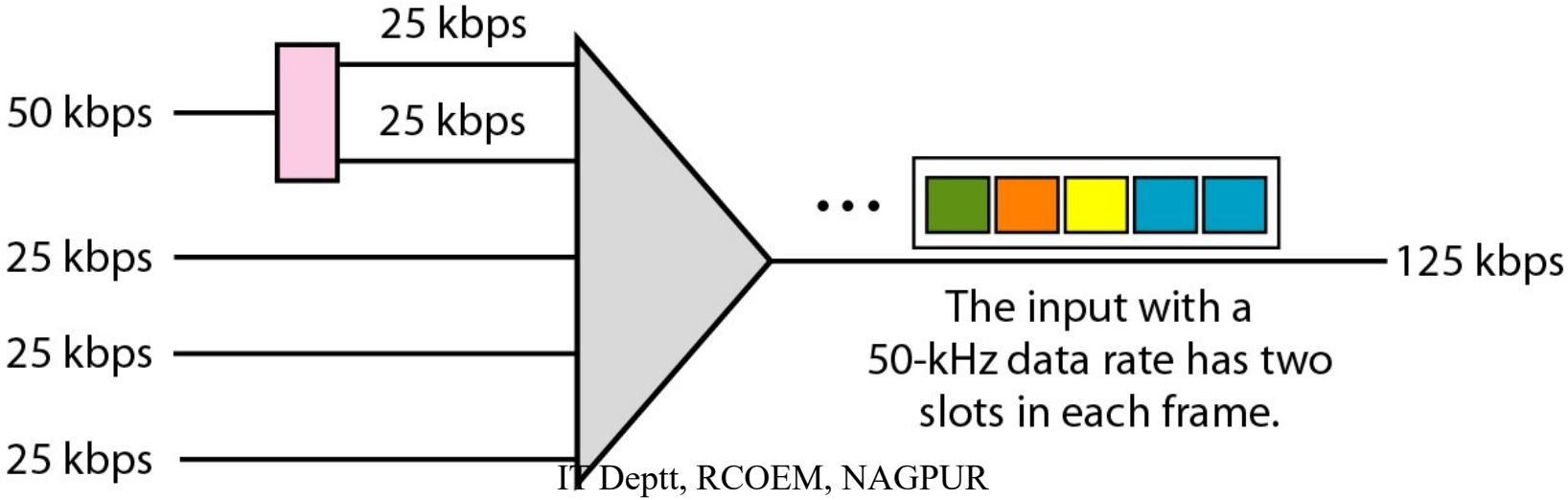
- A round of data units from each input connection is collected into a frame with duration T sec's.
- The data rate of the output link should be n times the data rate of the connection to guarantee flow of data.
- Assumption: Data rates of all the input lines is same.

- Drawbacks of Scheme:
  - ✓ The slot in output frame may be empty if the source has nothing to send.
  - ✓ Input data rates may vary.
- If data rates are not same, three strategies are used: multilevel multiplexing, multiple slot allocation and pulse stuffing.
- Multilevel multiplexing is used when the data rate of an input is a multiple of other.

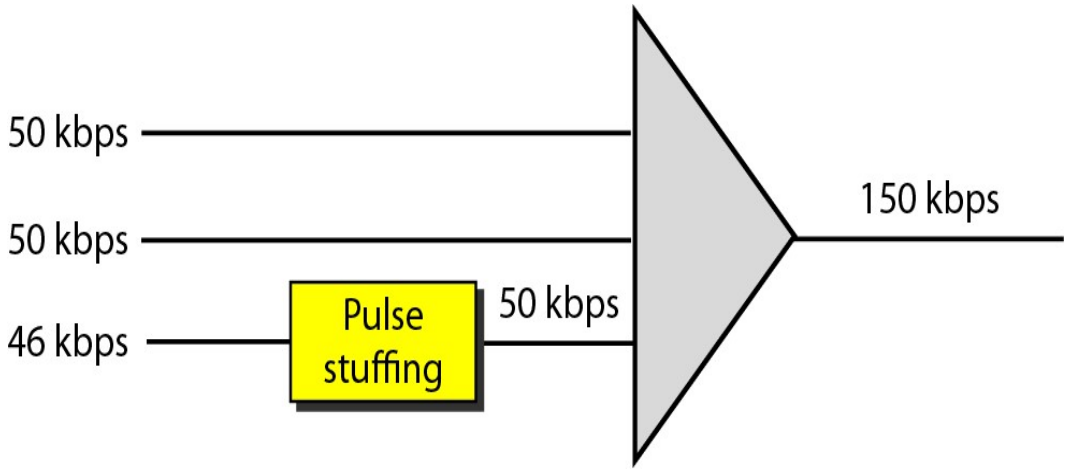
Multilevel multiplexing



- Multiple-slot allocation allocates more than one slot in a frame to a single input line.

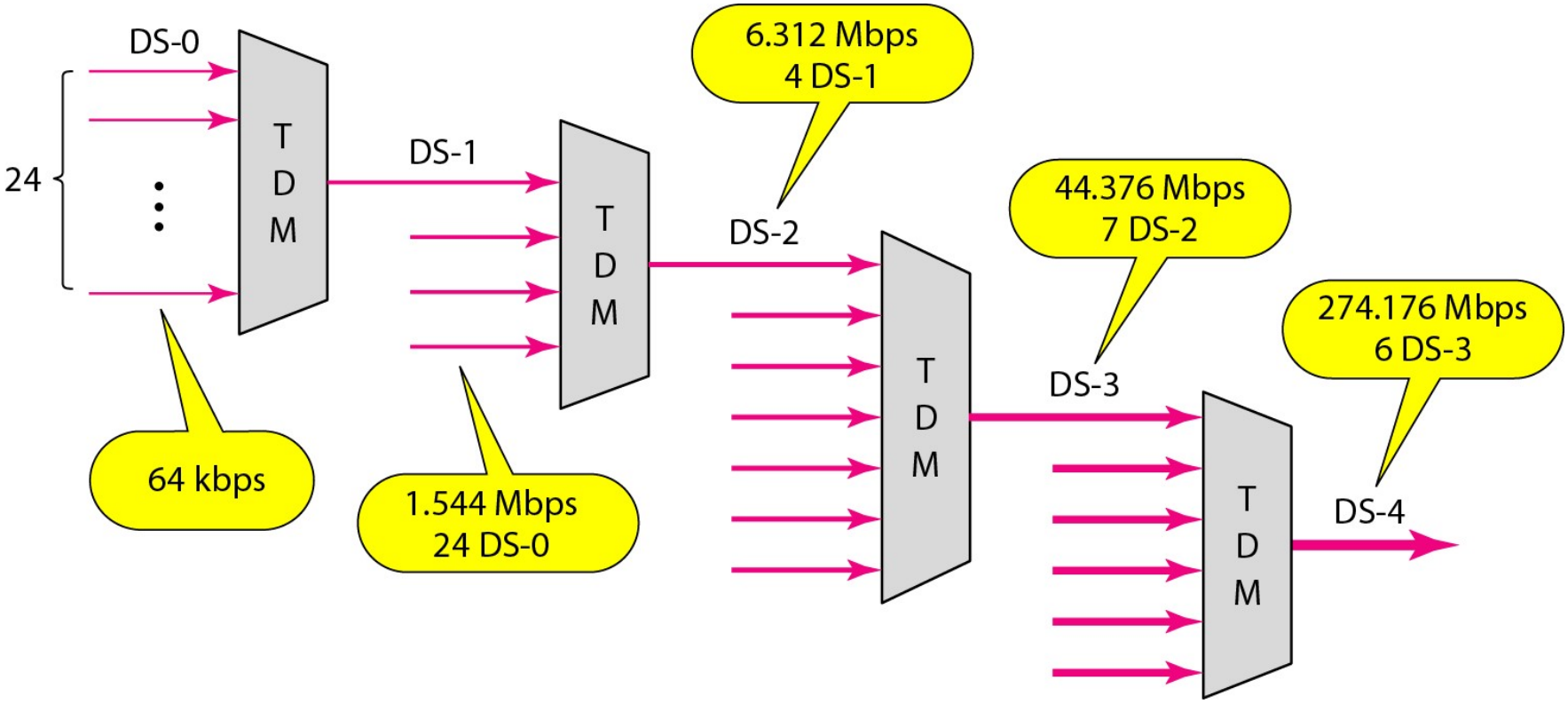


- Pulse stuffing is used if the bit rates of sources are not multiple integers of each other.
- The highest input data rate is considered as dominant rate and the dummy bits are added to the input lines with lower data rates.



- Major issue in TDM: Frame synchronization

Digital hierarchy



Statistical TDM

- Slots are dynamically allocated to improve bandwidth efficiency
- Only when an input line has slots worth of data to send it is given a slot in the output frame
- Thus, the number of slots in frame on the output line can be less than the number of input lines

- A slot on the output line carries address as well as data in STDM (not req. in Sync TDM)
- Slot size is important
- No Synchronization required
- The capacity of the output link is normally less than the sum of the capacities of each input channel. Capacity is based on the statistics of the load for each channel!!!

DATA LINK LAYER

- Deals with algorithms for achieving reliable, efficient communication between two adjacent machines at data link layer.
- Requires hardware as well as software !!
- Design issues:
  - Services provided
  - Framing
  - Error control
  - Flow control

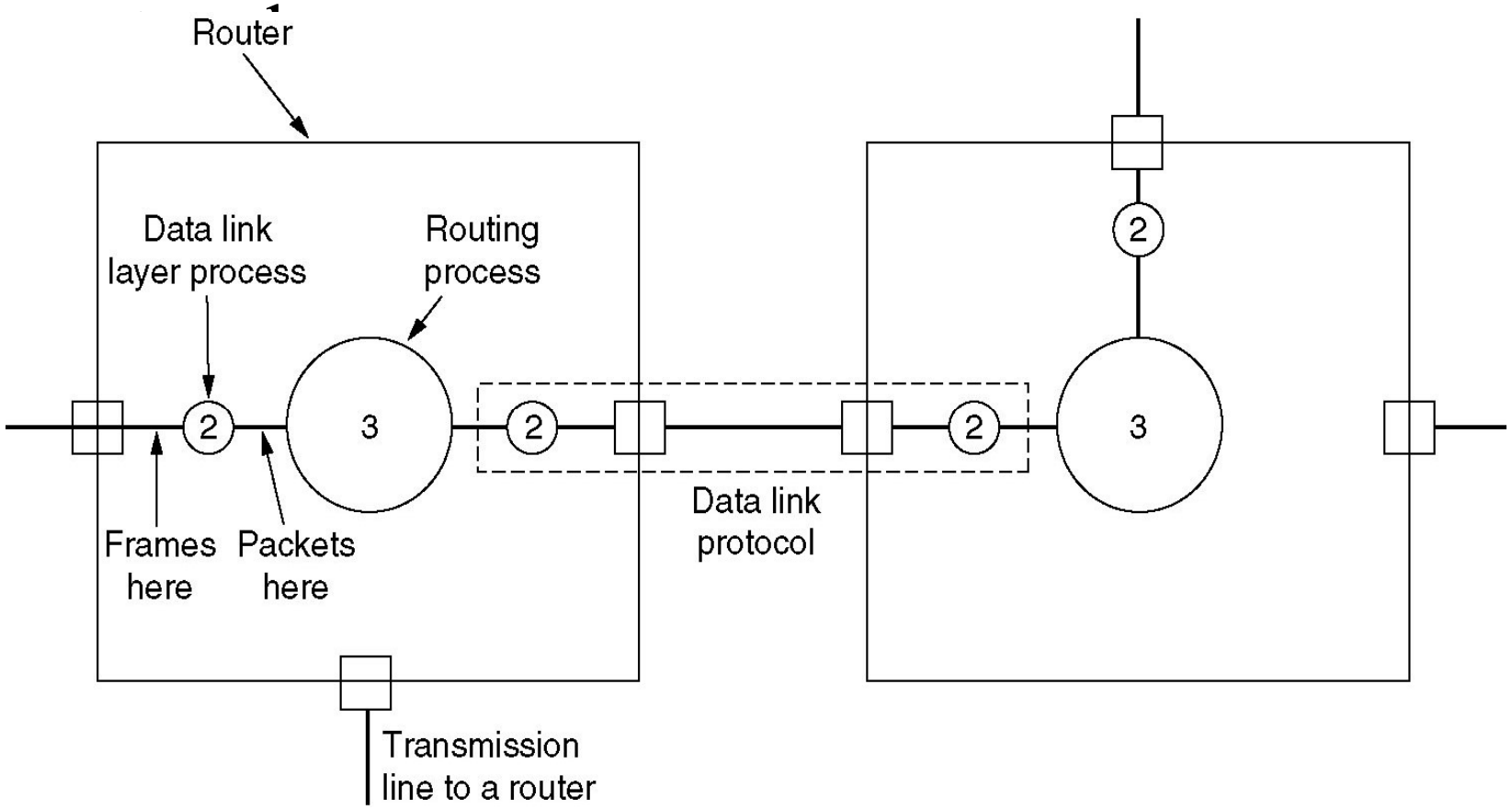
- Services provided to network layer:

✓ Principle service: Transferring data from network layer on source machine to that on destination machine.

✓ Three services: Unacknowledged Connectionless service(UCS), Acknowledged Connectionless service(ACS) and Acknowledged Connection Oriented service(ACOS).

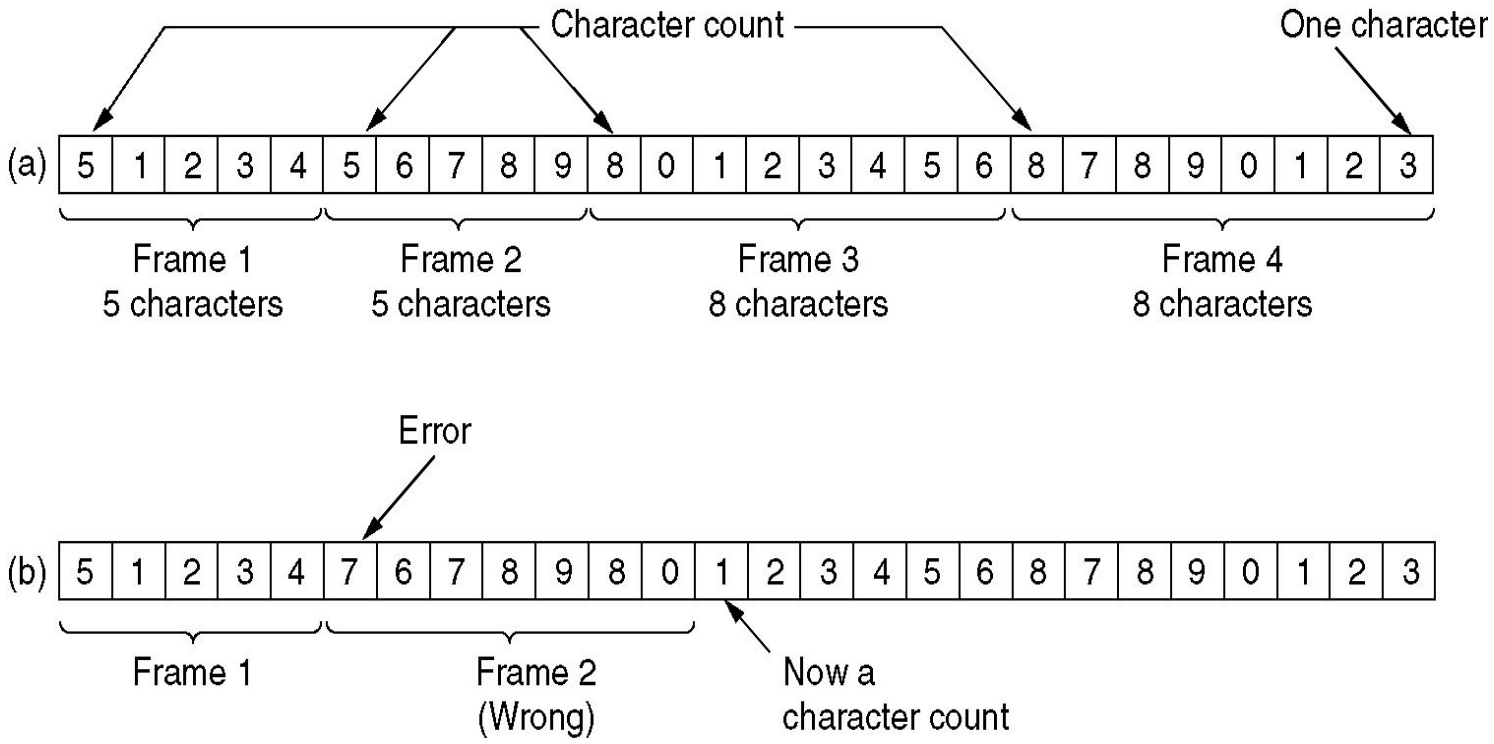
- UCS is appropriate when error rate is very low
- ACS is appropriate when unreliable channels are used.
- ACOS provides absolute reliability.

- Placement of data link layer



- Framing:
  - ✓ DLL uses the services provided by the physical layer.
  - ✓ Data provided by the physical layer is not guaranteed to be error free. Hence, detects and corrects the errors if any.
  - ✓ Breaks the data stream into discrete frames and computes / recomputes the checksum for each frame.
  - ✓ Different methods for framing:
    - o Character count.
    - o Starting and ending characters.
    - o Starting and ending flags.
    - o Physical layer coding violation.

- Character count:
  - ✓ Uses a field in the header to specify the number of characters in the frame.



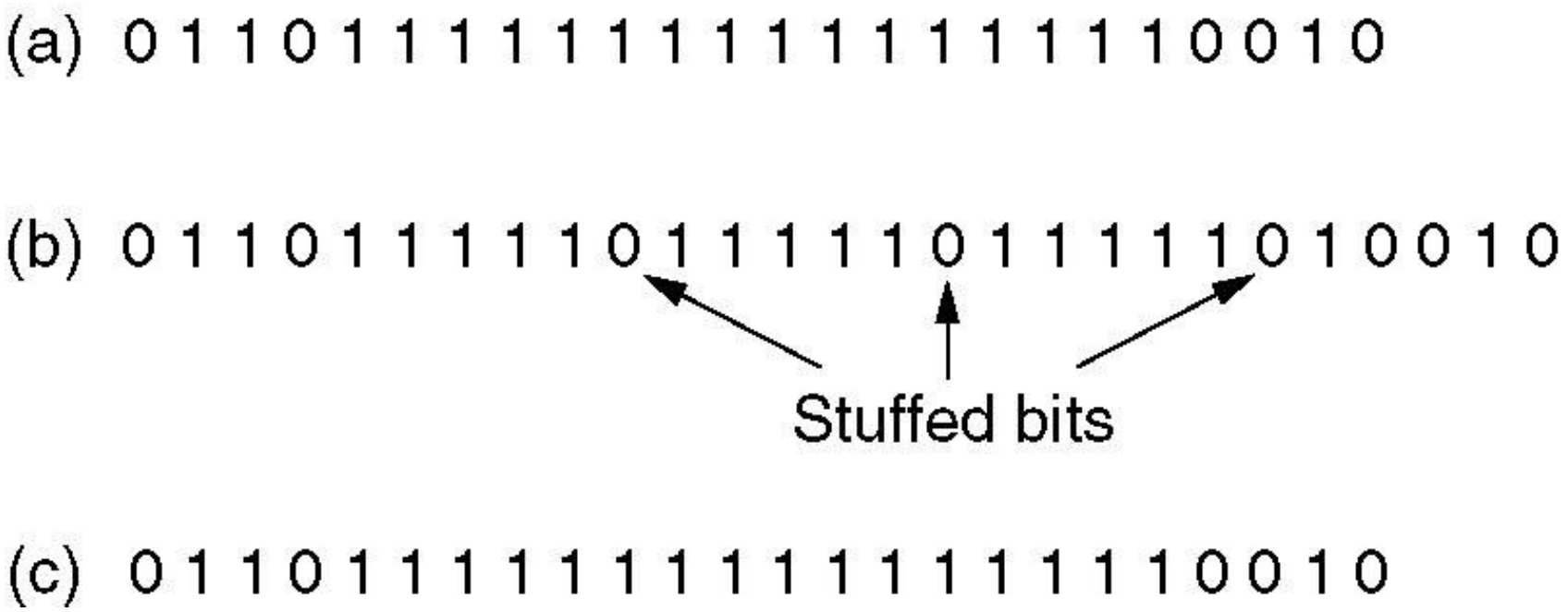
A character stream. (a) Without errors. (b) With one error.

- Starting and Ending characters with character stuffing:
  - ✓ Here each frame starts with ASCII character sequence DLE(Data Link Escape), STX(Start of Text) and ends with DLE, ETX(End of Text).
  - ✓ Since ASCII sequence is embedded in data, problem comes when data itself contains this sequence!!.
  - ✓ Solution:

    Sender's data link layer inserts an ASCII DLE character just before each accidental DLE character in the data. This DLE can be removed by the receiving end. This is called "character stuffing".
  - ✓ Disadvantage:

    - Closely tied to 8 bit character ASCII code.

- ✓ Need: A Technique which allows data frames to contain an arbitrary number of bits and allows character codes with an arbitrary number of bits per character.
- ✓ New Technique:
  - o Each frame begins and ends with a special bit pattern, 01111110, called flag byte.
  - o Whenever the senders data link layer encounters five consecutive one's in data, it automatically stuff's a 0 bit into outgoing bit stream.
  - o When receiver see's five consecutive 1 bits followed by 0, it destuff's the 0 bit.



Bit stuffing

(a) The original data.

(b) The data as they appear on the line.

(c) The data as they are stored in receiver's memory after destuffing.

- ✓ With bit stuffing the boundary between the two frames is unambiguously recognized by the flag pattern.
- ✓ If receiver looses track, it has to simply scan the flag sequences, since they only occur at frame boundary.

- Physical layer coding violation:
  - ✓ Only applicable to networks in which the encoding on the physical medium contains some redundancy.

- Error control

o Issues:

  ✓ How to ensure delivery of frames to n/w layer at destination in proper order.

  One solution : positive / negative acknowledgements.

  ✓ Frames can vanish completely b'coz of noise bursts.

  Solution: Timers, Sequence numbers.

- Flow control

o Issues:

  ✓ Fast sender, slow receiver.

  Solution: Feedback

Error Detection and Correction

- Vulnerable aspects of communication:
  - ✓ Local loops and wireless communication.
- Errors: single bit errors and burst errors.

  Burst errors are harder to detect and correct than isolated errors.
- Error correcting and detecting codes:
  - ✓ Two strategies:
    - o Include enough redundant information in each block of data to enable receiver to deduce what transmitted character must have been.(Error correction)
    - o To include only enough redundancy to allow the receiver to deduce that an error has occurred. (Error detection)

- Normally the frame consists of

  m- data bits, r- redundant or check bits, thus, the total length n=m+r.

  The n bit unit is referred to as n-bit codeword.
- Given two codeword's it possible to determine how many corresponding bits differ. (Ex-or them and count number of 1's). The number of bit positions in which two codewords differ is called the Hamming distance.
- Significance: If two codewords are a hamming distance d apart then it will require d single bit errors to convert one into another.

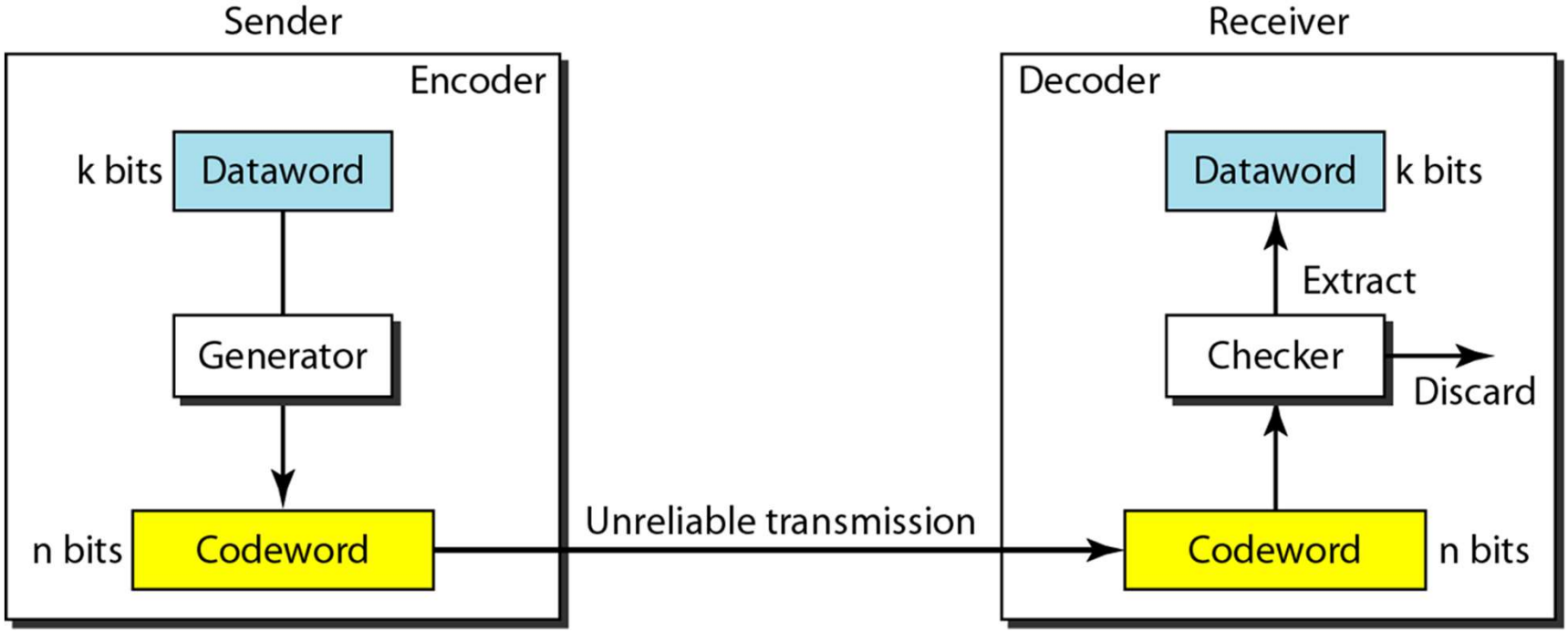
Error Detection using block codes

- If following conditions are met the receiver can detect the a change in original code word:

  i) The receiver has (or can find) a list of valid code words

  ii) The original code word has changed to invalid one

Process of error detection in Block Coding



- Note: The received code word may be accepted, rejected or accepted with errors

- **Remember**: An error detecting code can detect only the type of errors for which it is designed
- Example: let the data block size be k=2 bits and code word size be n=3 bits (so, n = k+r)

Let the data words and corresponding code words be:

Datawords	*Codewords*
00	000
01	011
10	101
11	110

- Assume that the sender encodes the data word 01 as 011 and sends it to receiver. Following cases can occur:

  i) Receiver receives 011, a valid code word,
  and extracts 01 from it.

  ii) Code word is corrupted and received as 111 (single bit error), the receiver discards the data as invalid

  iii) Code word is corrupted and received as 000 (two bit error, but legal code word!!), receiver incorrectly extracts data as 00.

Conclusion : This strategy can only detect 1 bit error

1-bit error correction

- More redundant bits are required for error correction
- Remember: Detection comes before correction
- For example to correct one bit 3 bits are required. For example:

Dataword	*Codeword*
00	00000
01	01011
10	10101
11	11110

- Assume that the sender encodes the data word 01 as 01011 and sends it to receiver. Code word is corrupted and is received as 01001 (1 bit change). Receiver finds that it is not a valid code word. Receiver uses following strategy to correct this one bit error:

  i) Compares the code word received with valid code words to see in how many bits it differs from each of the valid code words. Any valid code word differing with received one by only **one bit** is selected as the probable code word

  ii) Extract the data word from the identified valid code word

- Given an algorithm for computing the check bits it is possible to construct the complete list of legal codewords and from the list find two codewords whose hamming distance is minimum. This distance is the hamming distance of the complete code.
- To detect d errors, we need a distance d+1 code.
- Whenever receiver sees an invalid codeword it can tell that error has occurred.
- To correct d errors, we need a distance 2d+1 code.

Error correction/detection using hamming code

- All bit positions that are powers of two are used as check or parity bits. (positions 1, 2, 4, 8, 16, etc.)
- All other bit positions are for the data to be encoded. (positions 3, 5, 6, 7, 9, 10, 11, 12, etc.)
- Each check bit forces the parity of some collection of bits including itself, to be even (or odd). A data bit at position k is checked by the check bits occurring in its expansion using powers of 2. Example: data bit at 11 is checked by 1,2 and 8.
- Consider the 7-bit data word "0110101". Let **d** signify data bits and **p** signify parity bits.

- Data to send: 0 1 1 0 1 0 1
- Generation of Code word:

1	2	3	4	5	6	7	8	9	10	11
P1	P2		P3				P4			

P1	P2	D1	P3	D2	D3	D4	P4	D5	D6	D7

Data Bits	Position	Check bits	Parity bits and the data bits controlled by them
D1	3	1+2	P1: D1,D2,D4,D5,D7
D2	5	1+4	P2: D1,D3,D4,D6,D7
D3	6	2+4	P3: D2,D3,D4
D4	7	1+2+4	P4: D5,D6,D7
D5	9	1+8	
D6	10	2+8	
D7	11	1+2+8	

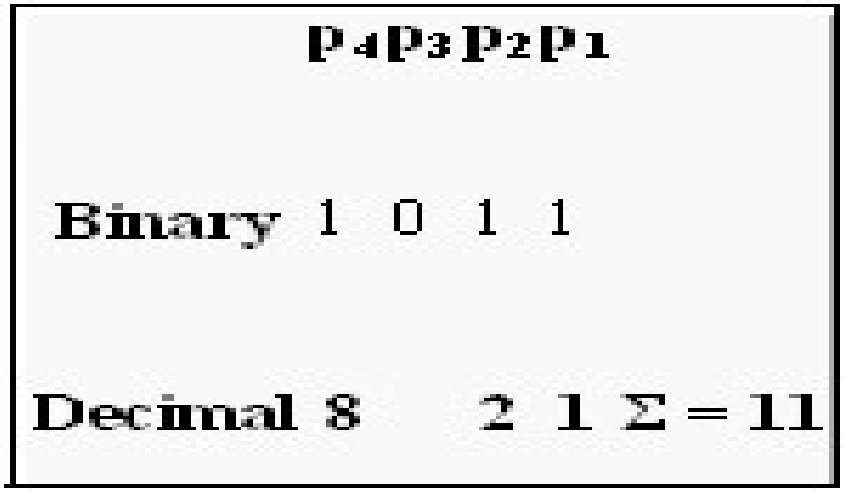
- Firstly the data bits are inserted into their appropriate positions and the parity bits calculated in each case using *even* parity.
- First step: sending side

	p_1	p_2	d_1	p_3	d_2	d_3	d_4	p_4	d_5	d_6	d_7
Data word (without parity):			**0**		**1**	**1**	**0**		**1**	**0**	**1**
p_1	**1**		0		1		0		1		1
p_2		**0**	0			1	0			0	1
p_3				**0**	1	1	0				
p_4								**0**	1	0	1
Data word (with parity):	**1**	**0**	0	**0**	1	1	0	**0**	1	0	1

- Second step: The new data word (with parity bits) is now "10001100101". We now assume the final bit gets corrupted and turned from 1 to 0. Our new data word is "10001100100“.

	p_1	p_2	d_1	p_3	d_2	d_3	d_4	p_4	d_5	d_6	d_7	Parity check	Parity bit
Received data word:	**1**	**0**	**0**	**0**	**1**	**1**	**0**	**0**	**1**	**0**	**0**	**1**	
p_1	**1**		0		1		0		1		0	**Fail**	**1**
p_2		**0**	0			1	0			0	0	**Fail**	**1**
p_3				**0**	1	1	0					Pass	0
p_4								**0**	1	0	0	**Fail**	**1**

- The final step is to evaluate the value of the parity bits :



- Error correcting codes are used when the channel is simplex.
- Usual method: Error detection followed by retransmission.

Polynomial code

- It is based on treating bit strings as representation of polynomials with coefficients 0 and 1.
- Ex: 110001=> $x^5+x^4+x^0$
- Polynomial arithmetic is done modulo 2. No carries or borrows used.
  Addition/subtraction are identical to Ex-OR.
- Sender and receiver agree on a generator polynomial G(x) in advance. Both high and low order bits of G(x) must be one.

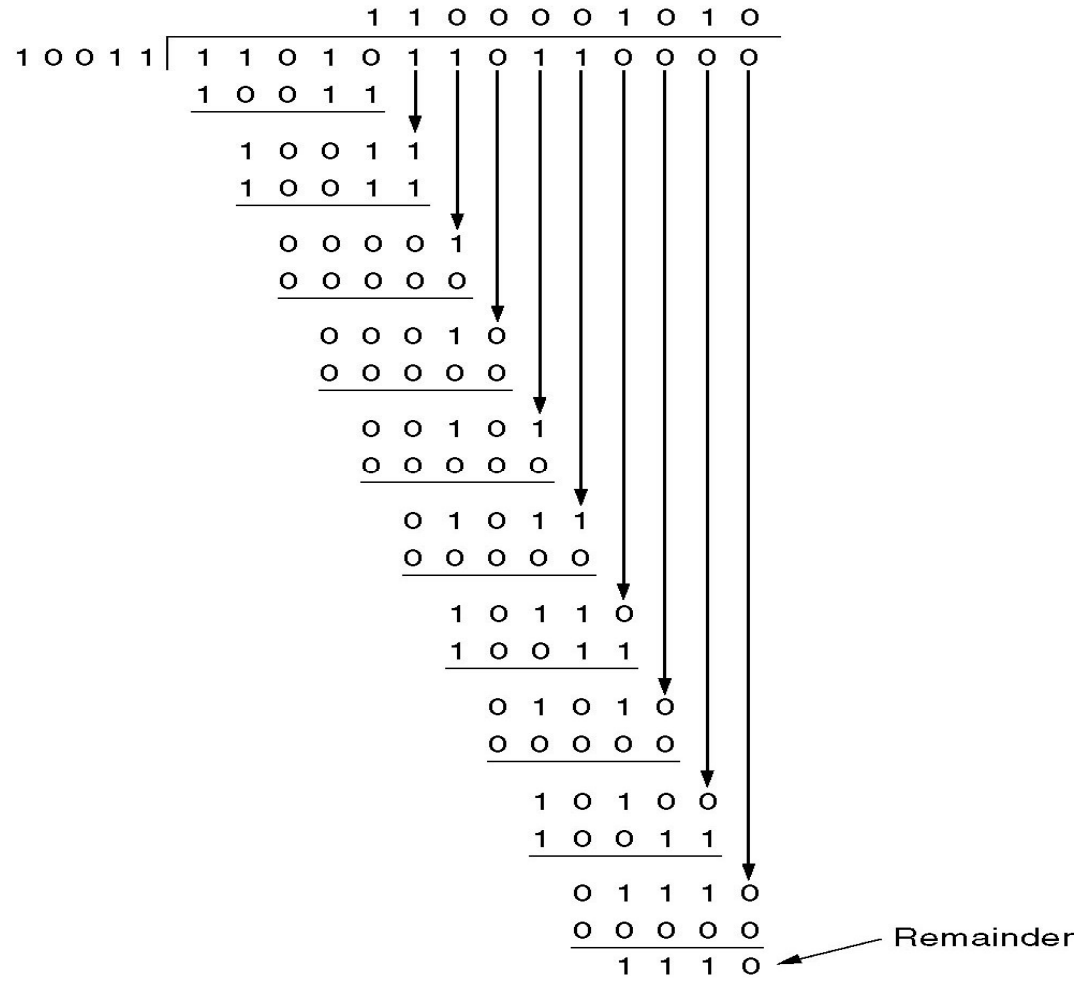
- To compute the checksum for the frame with m bits, corresponding to polynomial M(x), the frame must be longer than the generator polynomial.
- Idea is to append the checksum to the end of the frame in such a way that the polynomial represented by the checksummed frame is divisible by G(x).
- The receiver divides the checksummed frame by G(x). If there is a remainder, error has occurred.

- Steps:
  - ✓ Let r be the degree of G(x). Append r zero bits to low order end of the frame. It now contains m+r bits and corresponds to polynomial $x^r M(x)$.
  - ✓ Divide the bit string corresponding to $x^r M(x)$ by G(x) using modulo2 division.
  - ✓ Subtract the remainder from the bit string corresponding to $x^r M(x)$ using modulo 2 subtraction. The result is the checksummed frame to be transmitted, T(x).

Frame : 1 1 0 1 0 1 1 0 1 1

Generator: 1 0 0 1 1

Message after 4 zero bits are appended: 1 1 0 1 0 1 1 0 1 1 0 0 0 0



Transmitted frame: 1 1 0 1 0 1 1 0 1 1 1 1 1 0

- G(x): 1011
- Frame: 1001

- Brief analysis:
  - ✓ Let T(x) + E(x) be received by the receiver.

    E(x) --> Error, Each 1 bit in E(x)

    corresponds to a bit that has been inverted.
  - ✓ If there are k 1 bits in E(x), k single bit errors have occurred.
  - ✓ Upon receiving checksummed frame receiver divides it by G(x)i:e [T(x)+E(x)]/G(x)= E(x)/G(x).
  - ✓ Those errors that happen to correspond to polynomial containing G(x) as factor will slip, all others are caught.
- Three polynomials (International standard)

  CRC-12- $x^{12}+x^{11}+x^3+x^2+x^1+1$

  CRC-16- $x^{16}+x^{15}+x^2+1$ and

  CRC- CCITT- $x^{16}+x^{12}+x^5+1$

Elementary Data Link Protocols

- Assumptions on communication model:
  - ✓ Physical, data link and network layers are independent processes.
  - ✓ Basic service is reliable connection oriented service and the sender is assumed to have infinite data to send and never have to wait for data to be produced.
  - ✓ Data received from network layer is encapsulated by data link layer in the frame. Std. Library functions are used to send and receive data from physical layer.

  (to_physical_ layer and from_physical_layer)
  - ✓ The data link layer waits for something to happen using function, wait_for_event(&event).

- Protocol Definitions:

```
#define MAX_PKT 1024                                  /* determines packet size in bytes */

typedef enum {false, true} boolean;                   /* boolean type */
typedef unsigned int seq_nr;                          /* sequence or ack numbers */
typedef struct {unsigned char data[MAX_PKT];} packet;/*    packet definition */
typedef enum {data, ack, nak} frame_kind;             /* frame_kind definition */

typedef struct {                                      /* frames are transported in this layer */
  frame_kind kind;                                    /* what kind of a frame is it? */
  seq_nr seq;                                         /* sequence number */
  seq_nr ack;                                         /* acknowledgement number */
  packet info;                                        /* the network layer packet */
} frame;
```

Some definitions needed in the protocols to follow.

```
/* Wait for an event to happen; return its type in event. */
void wait_for_event(event_type *event);

/* Fetch a packet from the network layer for transmission on the channel. */
void from_network_layer(packet *p);

/* Deliver information from an inbound frame to the network layer. */
void to_network_layer(packet *p);

/* Go get an inbound frame from the physical layer and copy it to r. */
void from_physical_layer(frame *r);

/* Pass the frame to the physical layer for transmission. */
void to_physical_layer(frame *s);

/* Start the clock running and enable the timeout event. */
void start_timer(seq_nr k);

/* Stop the clock and disable the timeout event. */
void stop_timer(seq_nr k);

/* Start an auxiliary timer and enable the ack_timeout event. */
void start_ack_timer(void);

/* Stop the auxiliary timer and disable the ack_timeout event. */
void stop_ack_timer(void);

/* Allow the network layer to cause a network_layer_ready event. */
void enable_network_layer(void);

/* Forbid the network layer from causing a network_layer_ready event. */
void disable_network_layer(void);

/* Macro inc is expanded in-line: Increment k circularly. */
#define inc(k) if (k < MAX_SEQ) k = k + 1; else k = 0
```

Unrestricted Simplex Protocol

```
/* Protocol 1 (utopia) provides for data transmission in one direction only, from
   sender to receiver.  The communication channel is assumed to be error free,
   and the receiver is assumed to be able to process all the input infinitely quickly.
   Consequently, the sender just sits in a loop pumping data out onto the line as
   fast as it can. */

typedef enum {frame_arrival} event_type;
#include "protocol.h"

void sender1(void)
{
  frame s;                                  /* buffer for an outbound frame */
  packet buffer;                            /* buffer for an outbound packet */

  while (true) {
      from_network_layer(&buffer);          /* go get something to send */
      s.info = buffer;                      /* copy it into s for transmission */
      to_physical_layer(&s);                /* send it on its way */
  }                                 /        * Tomorrow, and tomorrow, and tomorrow,
                                             Creeps in this petty pace from day to day
                                             To the last syllable of recorded time
                                                 - Macbeth, V, v */
}

void receiver1(void)
{
  frame r;
  event_type event;                         /* filled in by wait, but not used here */

  while (true) {
      wait_for_event(&event);               /* only possibility is frame_arrival */
      from_physical_layer(&r);              /* go get the inbound frame */
      to_network_layer(&r.info);            /* pass the data to the network layer */
  }
}
```

Simplex Stop-and-Wait Protocol

```
/* Protocol 2 (stop-and-wait) also provides for a one-directional flow of data from
   sender to receiver. The communication channel is once again assumed to be error
   free, as in protocol 1. However, this time, the receiver has only a finite buffer
   capacity and a finite processing speed, so the protocol must explicitly prevent
   the sender from flooding the receiver with data faster than it can be handled. */

typedef enum {frame_arrival} event_type;
#include "protocol.h"

void sender2(void)
{
  frame s;                                  /* buffer for an outbound frame */
  packet buffer;                            /* buffer for an outbound packet */
  event_type event;                         /* frame_arrival is the only possibility */

  while (true) {
      from_network_layer(&buffer);          /* go get something to send */
      s.info = buffer;                      /* copy it into s for transmission */
      to_physical_layer(&s);                /* bye bye little frame */
      wait_for_event(&event);               /* do not proceed until given the go ahead */
  }
}

void receiver2(void)
{
  frame r, s;                               /* buffers for frames */
  event_type event;                         /* frame_arrival is the only possibility */
  while (true) {
      wait_for_event(&event);               /* only possibility is frame_arrival */
      from_physical_layer(&r);              /* go get the inbound frame */
      to_network_layer(&r.info);            /* pass the data to the network layer */
      to_physical_layer(&s);                /* send a dummy frame to awaken sender */
  }
}
```

A Simplex Protocol for a Noisy Channel

```
/* Protocol 3 (par) allows unidirectional data flow over an unreliable channel. */

#define MAX_SEQ 1                                   /* must be 1 for protocol 3 */
typedef enum  {frame_arrival, cksum_err, timeout} event_type;
#include "protocol.h"

void sender3(void)
{
  seq_nr next_frame_to_send;                        /* seq number of next outgoing frame */
  frame s;                                          /* scratch variable */
  packet buffer;                                    /* buffer for an outbound packet */
  event_type event;

  next_frame_to_send = 0;                           /* initialize outbound sequence numbers */
  from_network_layer(&buffer);                      /* fetch first packet */
  while (true) {
       s.info = buffer;                             /* construct a frame for transmission */
       s.seq = next_frame_to_send;                  /* insert sequence number in frame */
       to_physical_layer(&s);                       /* send it on its way */
       start_timer(s.seq);                          /* if answer takes too long, time out */
       wait_for_event(&event);                      /* frame_arrival, cksum_err, timeout */
       if (event == frame_arrival) {
            from_physical_layer(&s);                /* get the acknowledgement */
            if (s.ack == next_frame_to_send) {
                 stop_timer(s.ack);                 /* turn the timer off */
                 from_network_layer(&buffer);       /* get the next one to send */
                 inc(next_frame_to_send);           /* invert next_frame_to_send */
            }
       }
  }
}
```

```
void receiver3(void)
{
  seq_nr frame_expected;
  frame r, s;
  event_type event;

  frame_expected = 0;
  while (true) {
       wait_for_event(&event);                  /* possibilities: frame_arrival, cksum_err */
       if (event == frame_arrival) {            /* a valid frame has arrived. */
            from_physical_layer(&r);            /* go get the newly arrived frame */
            if (r.seq == frame  expected) {     /* this is what we have been waiting for. */
                 to_network_layer(&r.info);     /* pass the data to the network layer */
                 inc(frame_expected);           /* next time expect the other sequence nr */
            }
            s.ack = 1 – frame_expected;         /* tell which frame is being acked */
            to_physical_layer(&s);              /* send acknowledgement */
       }
  }
}
```

A positive acknowledgement with retransmission protocol.

Sliding Window Protocols

- A One-Bit Sliding Window Protocol
- A Protocol Using Go Back N
- A Protocol Using Selective Repeat

Sliding Window Protocols

- Each outbound frame must contain a *sequence number.* With *n* bits for the sequence number field,

  maxseq = $\mathbf{2^n}$ - 1 and the numbers range from 0 to maxseq.
- *Sliding window* : sender has a window of frames and maintains a list of consecutive sequence numbers for frames that it is permitted to send without waiting for ACKs.
- Receiver has a window that is a list of frame sequence numbers it is permitted to accept.
- *Note – sending and receiving windows do NOT have to be the same size.*
- Windows can be fixed size or dynamically growing and shrinking.

- Host is oblivious, message order at transport level is maintained.
- *sender's window* : *frames sent but not yet ACKed.*
  - New packets from the Host cause the upper edge inside sender window to be incremented.
  - ACKed frames from the receiver cause the lower edge inside window to be incremented.
- All frames in the sender's window must be saved for possible retransmission and we need one timer per frame in the window.

- If the maximum sender window size is B, the sender needs B buffers.
- If the sender window gets full (i.e., reaches its maximum window size, the protocol must shut off the Host (the network layer) until buffers become available.

Receiver window

- Frames received with sequence numbers outside the *receiver window* are not accepted.
- The receiver window size is normally *static*.

  The set of acceptable sequence numbers is rotated as "acceptable" frames arrive.

a receiver window size = 1 ➔ *the protocol only accepts frames in order.*

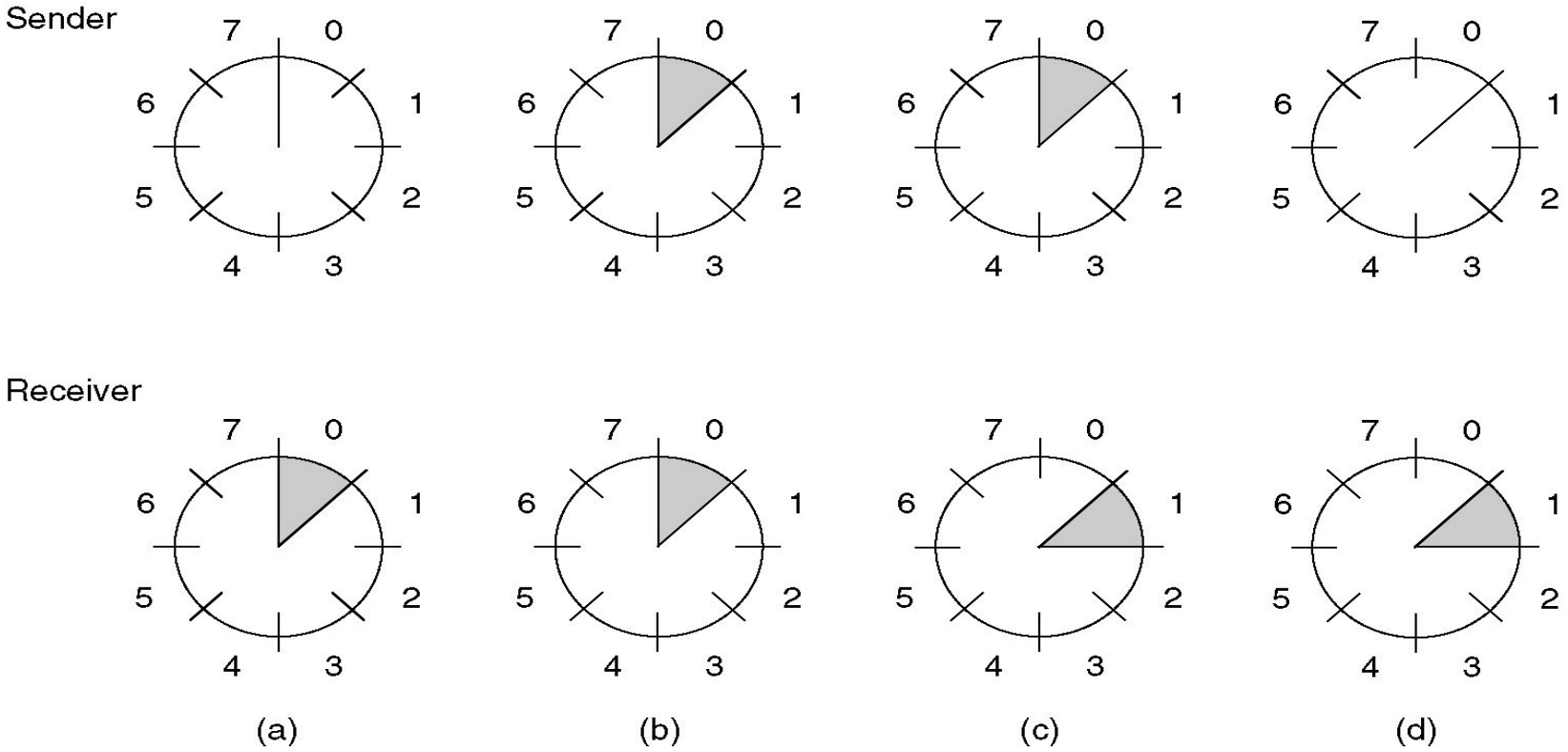
There is referred to as **Go Back N**.

Standard Ways to ACK

- ACK sequence number indicates *the last frame successfully received.*

- OR -

- ACK sequence number indicates *the next frame the receiver expects to receive.*
- *Both of these can be strictly individual ACKs or represent cumulative ACK.*
- Cumulative ACK is the most common technique.



A sliding window of size 1, with a 3-bit sequence number.

(a) Initially.

(b) After the first frame has been sent.

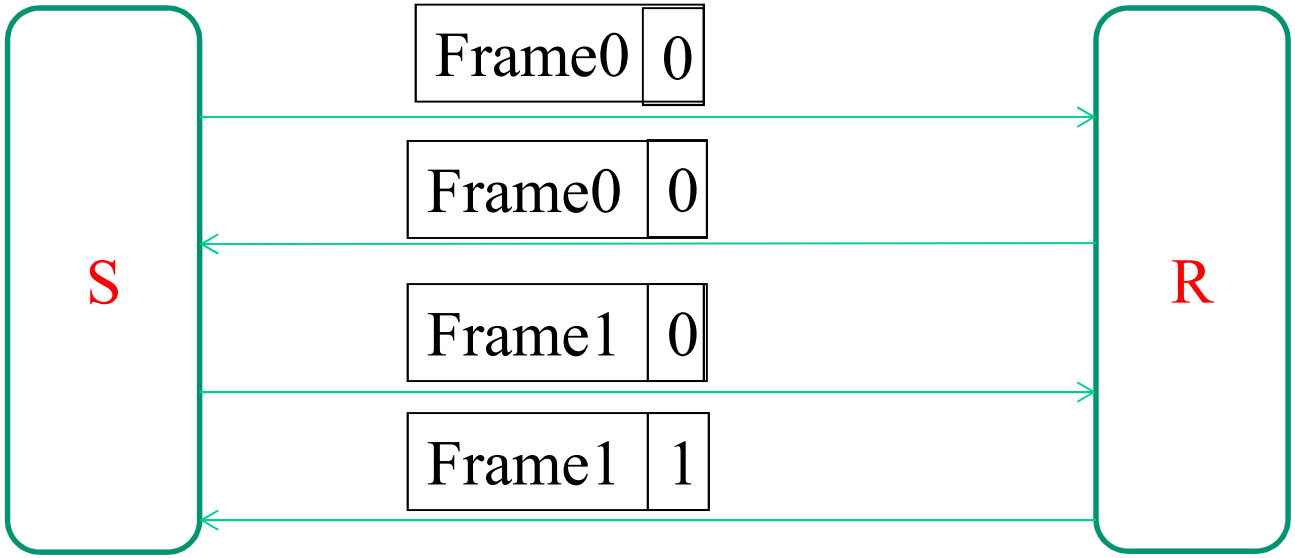
(c) After the first frame has been received.

(d) After the first acknowledgement has been received.

Piggybacking

- A separate acknowledgement has an overhead,
  "Bandwidth is consumed by them during transmission"
  If 'b' is the bandwidth requirement of data, then b/2 bandwidth is used by acknowledgements!!!

- In bidirectional data transfer, Piggybacking makes efficient utilization of bandwidth by allowing acknowledments for the frames received in next new data frame through an additional field in header!!!



Limitation of Simple Stop and wait protocol

- The sender sends one frame and waits for acknowledgement

  Problem: Underutilization of bandwidth with delay

Example:

Bandwidth: 50kbps

RTT: 500msecs

Frame size: 1000 bits

- Transmission time: 20msecs
- Time at which complete frame arrives at receiver: 270 msecs (250 + 20)
- Assume negligible processing and extremely short Ack frame,

  Time at which Ack is received: 520 msecs

  (270 + 250)
- Sender is blocked for 96% of time (500/520)
- Observation: Long transit time, high bandwidth and short frame length are not good for efficiency!!!

Important concepts

- **Transmission Delay (Tt)** – Time to transmit the packet from host to the outgoing link. If B is the Bandwidth of the link and D is the Data Size to transmit

$$Tt = D/B$$

- **Propagation Delay (Tp)** – It is the time taken by the first bit transferred by the host onto the outgoing link to reach the destination. It depends on the distance d and the wave propagation speed s of the medium.

$$Tp = d/s$$

- **Efficiency** – It is defined as the ratio of total useful time to the total cycle time of a packet. For stop and wait protocol,

Total cycle time = Tt(data) + Tp(data) + Tt(ack) + Tp(ack)

(Since ack are very small Tt(ack) can be ignored)

= Tt(data) + Tp(data) + Tp(ack)

= Tt + 2*Tp

= 1+ 2Tp / Tt = 1+ 2a (a = Tp/Tt)

Efficiency = Tt / (Tt + 2Tp) = 1 / (1+2a)

- **Effective Bandwidth(EB) or Throughput** – Number of bits sent per second.

  EB = Data Size(L) / Total Cycle time (Tt + 2*Tp)

  Multiplying and dividing by Bandwidth (B),

  = (1/(1+2a)) * B [Using a = Tp/Tt]

  = Efficiency * Bandwidth

- **Capacity of link** – If a channel is Full Duplex, then bits can be transferred in both the directions and without any collisions. Number of bits a channel can hold at maximum is its capacity. For Simplex channel:

  Capacity = Bandwidth(B) * Propagation(Tp)

  For Full Duplex channels,

  Capacity = 2*Bandwidth(B) * Propagation(Tp)

Problem

- Compute approximate optimal window size when packet size is 53 bytes, RTT is 60 msec and bandwidth is 155 Mbps.

- Transmission delay (T_t)

  = Packet size / Bandwidth

  = 53 bytes / 155 Mbps

  = (53 x 8 bits) / (155 x 10^6 bits per sec)

  = 2.735 μsec

- Propagation delay (T_p)

  = Round Trip Time / 2

  = 60 msec / 2

  = 30 msec

- $a = T_p / T_t$

  $= 30$ msec $/ 2.735$ μsec

  $= 10968.921$

- Optimal window size

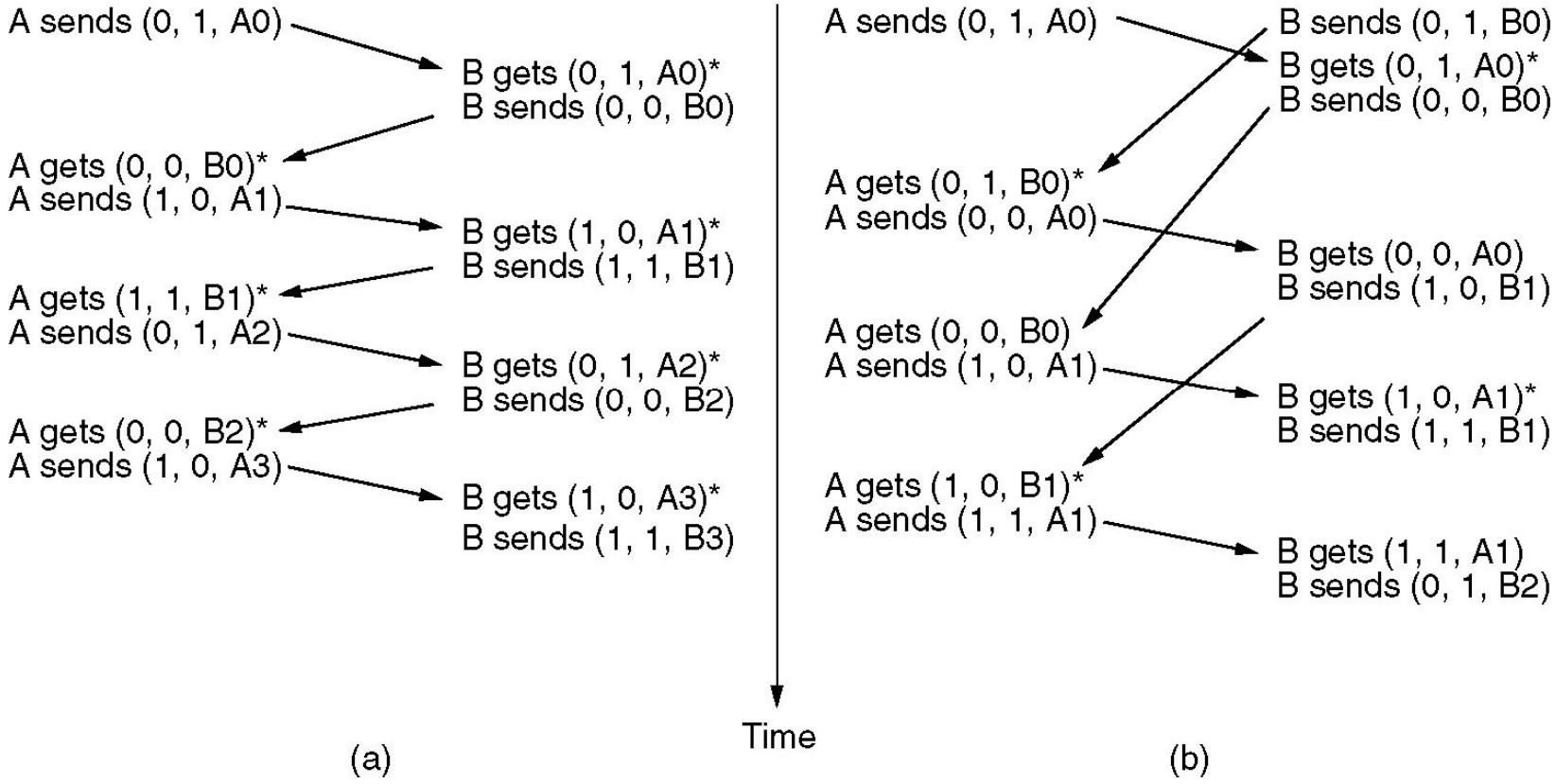
  $= 1 + 2a = 1 + 2 \times 10968.921 = 21938.84$

- Thus, approximate optimal window size = 21938 frames.

A One-Bit Sliding Window Protocol

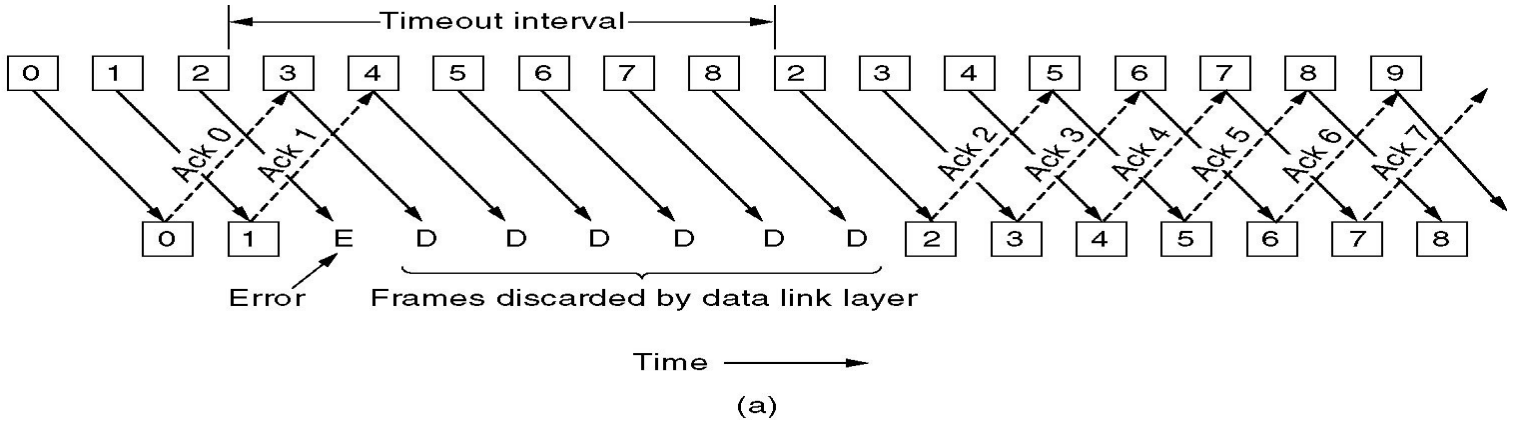
```
/* Protocol 4 (sliding window) is bidirectional. */
#define MAX_SEQ 1                                   /* must be 1 for protocol 4 */
typedef enum {frame_arrival, cksum_err, timeout} event_type;
#include "protocol.h"
void protocol4 (void)
{
  seq_nr next_frame_to_send;                        /* 0 or 1 only */
  seq_nr frame_expected;                            /* 0 or 1 only */
  frame r, s;                                       /* scratch variables */
  packet buffer;                                    /* current packet being sent */
  event_type event;

  next_frame_to_send = 0;                           /* next frame on the outbound stream */
  frame_expected = 0;                               /* frame expected next */
  from_network_layer(&buffer);                      /* fetch a packet from the network layer */
  s.info = buffer;                                  /* prepare to send the initial frame */
  s.seq = next_frame_to_send;                       /* insert sequence number into frame */
  s.ack = 1 – frame_expected;                       /* piggybacked ack */
  to_physical_layer(&s);                            /* transmit the frame */
  start_timer(s.seq);                               /* start the timer running */
```

```
while (true) {
    wait_for_event(&event);                 /* frame_arrival, cksum_err, or timeout */
    if (event == frame_arrival) {           /* a frame has arrived undamaged. */
        from_physical_layer(&r);            /* go get it */

        if (r.seq == frame_expected) {      /* handle inbound frame stream. */
            to_network_layer(&r.info);      /* pass packet to network layer */
            inc(frame_expected);            /* invert seq number expected next */
        }

        if (r.ack == next_frame_to_send) {  /* handle outbound frame stream. */
            stop_timer(r.ack);              /* turn the timer off */
            from_network_layer(&buffer);    /* fetch new pkt from network layer */
            inc(next_frame_to_send);        /* invert senderís sequence number */
        }
    }
    s.info = buffer;                        /* construct outbound frame */
    s.seq = next_frame_to_send;             /* insert sequence number into it */
    s.ack = 1 – frame_expected;             /* seq number of last received frame */
    to_physical_layer(&s);                  /* transmit a frame */
    start_timer(s.seq);                     /* start the timer running */
  }
}
```

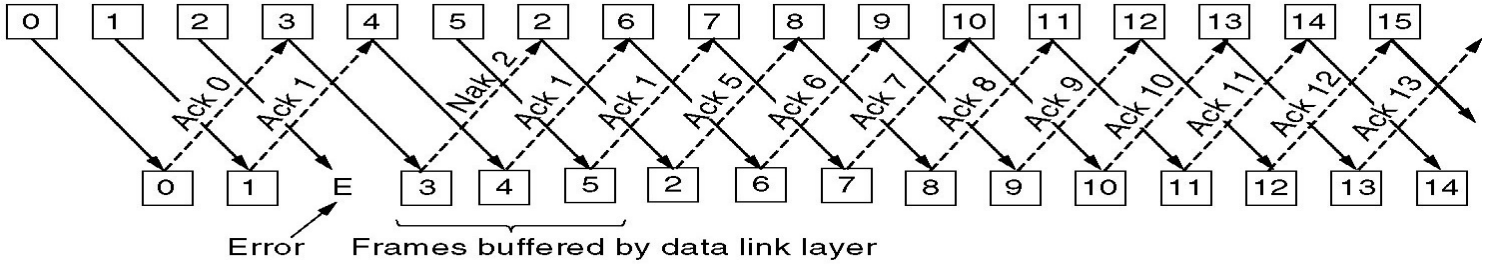


Two scenarios for protocol 4. (a) Normal case. (b) Abnormal case. The notation is (seq, ack, packet number). An asterisk indicates where a network layer accepts a packet.

A Protocol Using Go Back N



(a)



(b)

Pipelining and error recovery. Effect on an error when

(a) Receiver's window size is 1.

(b) Receiver's window size is large.

S W P Using Go Back N

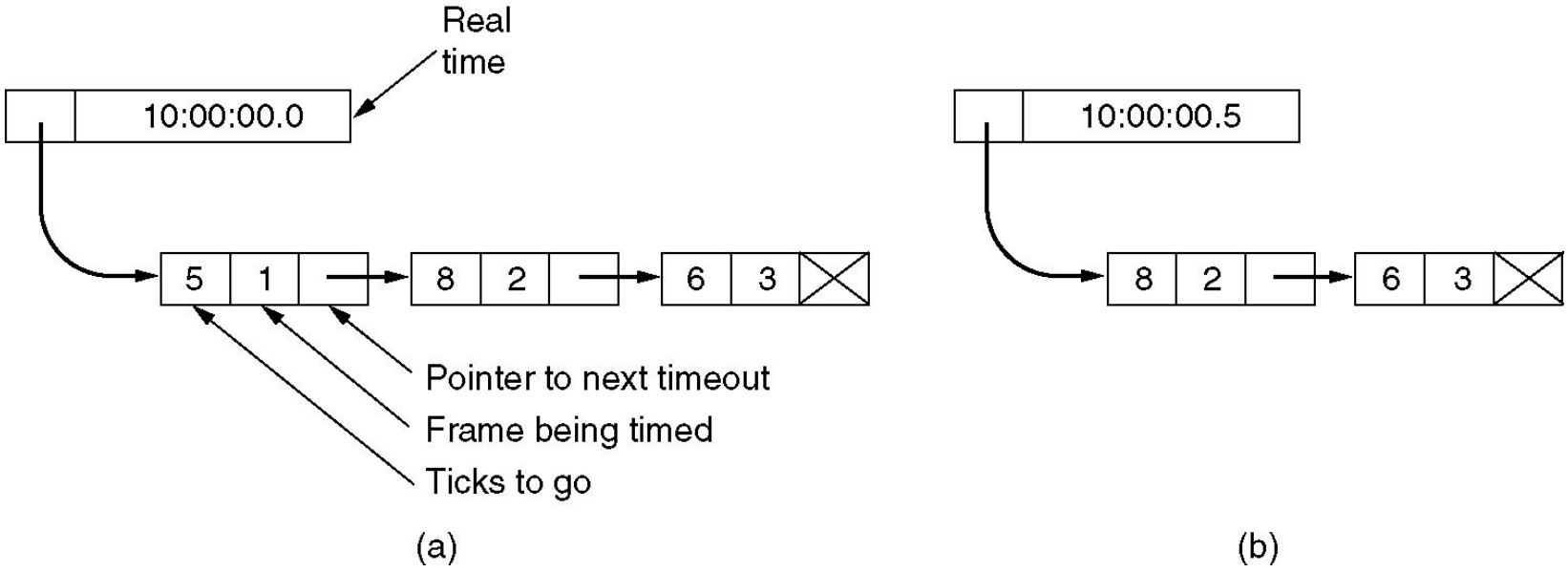
```
/* Protocol 5 (pipelining) allows multiple outstanding frames. The sender may transmit up
   to MAX_SEQ frames without waiting for an ack. In addition, unlike the previous protocols,
   the network layer is not assumed to have a new packet all the time. Instead, the
   network layer causes a network_layer_ready event when there is a packet to send. */

#define MAX_SEQ 7                       /* should be 2^n – 1 */
typedef enum {frame_arrival, cksum_err, timeout, network_layer_ready} event_type;
#include "protocol.h"

static boolean between(seq_nr a, seq_nr b, seq_nr c)
{
/* Return true if a <=b < c circularly; false otherwise. */
  if (((a <= b) && (b < c)) || ((c < a) && (a <= b)) || ((b < c) && (c < a)))
       return(true);
    else
       return(false);
}

static void send_data(seq_nr frame_nr, seq_nr frame_expected, packet buffer[ ])
{
/* Construct and send a data frame. */
  frame s;                              /* scratch variable */

  s.info = buffer[frame_nr];            /* insert packet into frame */
  s.seq = frame_nr;                     /* insert sequence number into frame */
  s.ack = (frame_expected + MAX_SEQ) % (MAX_SEQ + 1);/* piggyback ack */
  to_physical_layer(&s);                /* transmit the frame */
  start_timer(frame_nr);                /* start the timer running */
}
```

```
void protocol5(void)
{
  seq_nr next_frame_to_send;          /* MAX_SEQ > 1; used for outbound stream */
  seq_nr ack_expected;                /* oldest frame as yet unacknowledged */
  seq_nr frame_expected;              /* next frame expected on inbound stream */
  frame r;                            /* scratch variable */
  packet buffer[MAX_SEQ + 1];         /* buffers for the outbound stream */
  seq_nr nbuffered;                   /* # output buffers currently in use */
  seq_nr i;                           /* used to index into the buffer array */
  event_type event;

  enable_network_layer();             /* allow network_layer_ready events */
  ack_expected = 0;                   /* next ack expected inbound */
  next_frame_to_send = 0;             /* next frame going out */
  frame_expected = 0;                 /* number of frame expected inbound */
  nbuffered = 0;                      /* initially no packets are buffered */
```

```
while (true) {
  wait_for_event(&event);              /* four possibilities: see event_type above */

  switch(event) {
    case network_layer_ready:          /* the network layer has a packet to send */
         /* Accept, save, and transmit a new frame. */
         from_network_layer(&buffer[next_frame_to_send]); /* fetch new packet */
         nbuffered = nbuffered + 1;  /* expand the sender's window */
         send_data(next_frame_to_send, frame_expected, buffer);/* transmit the frame */
         inc(next_frame_to_send);    /* advance sender's upper window edge */
         break;

    case frame_arrival:                /* a data or control frame has arrived */
         from_physical_layer(&r);      /* get incoming frame from physical layer */

         if (r.seq == frame_expected) {
              /* Frames are accepted only in order. */
              to_network_layer(&r.info);  /* pass packet to network layer */
              inc(frame_expected);   /* advance lower edge of receiver's window */
          }
```

```
            /* Ack n implies n – 1, n – 2, etc.  Check for this. */
            while (between(ack_expected, r.ack, next_frame_to_send)) {
                /* Handle piggybacked ack. */
                nbuffered = nbuffered    1; /* one frame fewer buffered */
                stop_timer(ack_expected); /* frame arrived intact; stop timer */
                inc(ack_expected);      /* contract sender's window */
            }
            break;

      case cksum_err: break;              /* just ignore bad frames */

      case timeout:                       /* trouble; retransmit all outstanding frames */
            next_frame_to_send = ack_expected;    /* start retransmitting here */
            for (i = 1; i <= nbuffered; i++) {
                send_data(next_frame_to_send, frame_expected, buffer);/* resend 1 frame */
                inc(next_frame_to_send);  /* prepare to send the next one */
            }

    }

    if (nbuffered < MAX_SEQ)
            enable_network_layer();
    else
            disable_network_layer();
  }
}
```

Sliding Window Protocol Using Go Back N



Simulation of multiple timers in software.

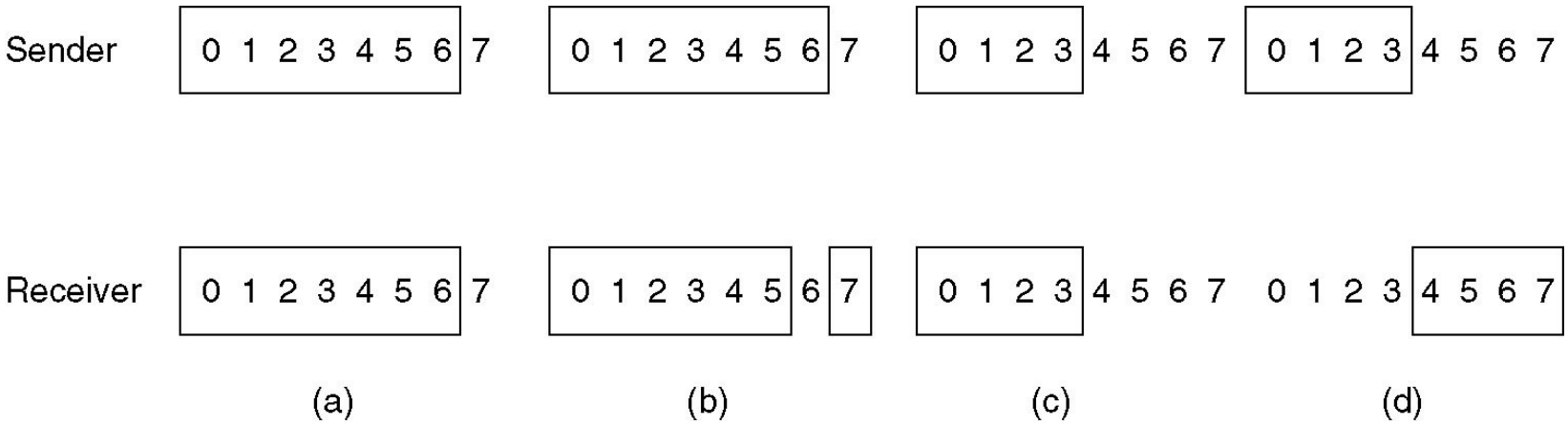
A Sliding Window Protocol Using Selective Repeat

```
/* Protocol 6 (nonsequential receive) accepts frames out of order, but passes packets to the
   network layer in order. Associated with each outstanding frame is a timer. When the timer
   expires, only that frame is retransmitted, not all the outstanding frames, as in protocol 5. */

#define MAX_SEQ 7                                       /* should be 2^n – 1 */
#define NR_BUFS ((MAX_SEQ + 1)/2)
typedef enum {frame_arrival, cksum_err, timeout, network_layer_ready, ack_timeout} event_type;
#include "protocol.h"
boolean no_nak = true;                                  /* no nak has been sent yet */
seq_nr oldest_frame = MAX- SEQ + 1;                     /* initial value is only for the simulator */

static boolean between(seq_nr a, seq_nr b, seq_nr c)
{
/* Same as between in protocol5, but shorter and more obscure. */
  return ((a <= b) && (b < c)) II ((c < a) && (a <= b)) II ((b < c) && (c < a));
}

static void send_frame(frame_kind fk, seq_nr frame_nr, seq_nr frame_expected, packet buffer[ ])
{
/* Construct and send a data, ack, or nak frame. */
  frame s;                                              /* scratch variable */

  s.kind = fk;                                          /* kind == data, ack, or nak */
  if (fk == data) s.info = buffer[frame_nr % NR_BUFS];
  s.seq = frame_nr;                                     /* only meaningful for data frames */
  s.ack = (frame_expected + MAX_SEQ) % (MAX_SEQ + 1);
  if (fk == nak) no_nak = false;                        /* one nak per frame, please */
  to_physical_layer(&s);                                /* transmit the frame */
  if (fk == data) start_timer(frame_nr % NR_BUFS);
  stop_ack_timer();                                     /* no need for separate ack frame */
}
```

```
void protocol6(void)
{
  seq_nr ack_expected;                          /* lower edge of sender's window */
  seq_nr next_frame_to_send;                    /* upper edge of sender's window + 1 */
  seq_nr frame_expected;                        /* lower edge of receiver's window */
  seq_nr too_far;                               /* upper edge of receiver's window + 1 */
  int i;                                        /* index into buffer pool */
  frame r;                                      /* scratch variable */
  packet out_buf[NR_BUFS];                      /* buffers for the outbound stream */
  packet in_buf[NR_BUFS];                       /* buffers for the inbound stream */
  boolean arrived[NR_BUFS];                     /* inbound bit map */
  seq_nr nbuffered;                             /* how many output buffers currently used */
  event_type event;

  enable_network_layer();                       /* initialize */
  ack_expected = 0;                             /* next ack expected on the inbound stream */
  next_frame_to_send = 0;                       /* number of next outgoing frame */
  frame_expected = 0;
  too_far = NR_BUFS;
  nbuffered = 0;                                /* initially no packets are buffered */
  for (i = 0; i < NR_BUFS; i++) arrived[i] = false;
```

```
while (true) {
  wait_for_event(&event);                          /* five possibilities: see event_type above */
  switch(event) {
    case network_layer_ready:                      /* accept, save, and transmit a new frame */
         nbuffered = nbuffered + 1;                /* expand the window */
         from_network_layer(&out  buf[next_frame_to_send % NR_BUFS]); /* fetch new packet */
         send_frame(data, next_frame_to_send, frame_expected, out_buf);/* transmit the frame */
         inc(next_frame_to_send);                  /* advance upper window edge */
         break;

    case frame_arrival:                            /* a data or control frame has arrived */
         from_physical_layer(&r);                  /* fetch incoming frame from physical layer */
         if (r.kind == data) {
              /* An undamaged frame has arrived. */
              if ((r.seq != frame_expected) && no_nak)
                 send_frame(nak, 0, frame_expected, out_buf); else start_ack_timer();
              if (between(frame_expected, r.seq, too_far) && (arrived[r.seq%NR_BUFS] == false)) {
                   /* Frames may be accepted in any order. */
                   arrived[r.seq % NR_BUFS] = true;      /* mark buffer as full */
                   in_buf[r.seq % NR_BUFS] = r.info;     /* insert data into buffer */
                   while (arrived[frame_expected % NR_BUFS]) {
                        /* Pass frames and advance window. */
                        to_network_layer(&in_buf[frame_expected % NR_BUFS]);
                        no_nak = true;
                        arrived[frame_expected % NR_BUFS] = false;
                        inc(frame_expected);     /* advance lower edge of receiver's window */
                        inc(too_far);            /* advance upper edge of receiver's window */
                        start_ack_timer();       /* to see if a separate ack is needed */
                   }
              }
         }
```

```
            if((r.kind==nak) && between(ack_expected,(r.ack+1)%(MAX_SEQ+1),next frame to send))
                  send_frame(data, (r.ack+1) % (MAX_SEQ + 1), frame_expected, out_buf);

            while (between(ack_expected, r.ack, next_frame_to_send)) {
                  nbuffered = nbuffered    1;           /* handle piggybacked ack */
                  stop_timer(ack_expected % NR_BUFS);    /* frame arrived intact */
                  inc(ack_expected);                     /* advance lower edge of sender's window */
            }
            break;
      case cksum_err:
            if (no_nak) send_frame(nak, 0, frame_expected, out_buf);/* damaged frame */
            break;
      case timeout:
            send_frame(data, oldest_frame, frame_expected, out_buf);/* we timed out */
            break;
      case ack_timeout:
            send_frame(ack,0,frame_expected, out_buf);     /* ack timer expired; send ack */
    }
    if (nbuffered < NR_BUFS) enable_network_layer(); else disable_network_layer();
  }
}
```



(a) Initial situation with a window size seven.

(b) After seven frames sent and received, but not acknowledged.

(c) Initial situation with a window size of four.

(d) After four frames sent and received, but not acknowledged.