Data Link Layer

Kumkum Saxena

Functions of data link layer

1. Provides a well defined service interface to the network layer.

The data link layer offers three types of services.

Unacknowledged connectionless service

- Here, the data link layer of the sending machine sends independent frames to the data link layer of the receiving machine.
- The receiving machine does not acknowledge receiving the frame.
- No logical connection is set up between the host machines.
- Error and data loss is not handled in this service.
- This is applicable in Ethernet services and voice communications.

Functions of data link layer

Acknowledged connectionless service

- Here, no logical connection is set up between the host machines, but each frame sent by the source machine is acknowledged by the destination machine on receiving.
- If the source does not receive the acknowledgment within a stipulated time, then it resends the frame. This is used in Wifi (IEEE 802.11) services.

Acknowledged connection-oriented service

- This is the best service that the data link layer can offer to the network layer.
- A logical connection is set up between the two machines and the data is transmitted along this logical path.
- The frames are numbered, that keeps track of loss of frames and also ensures that frames are received in correct order.

Functions of data link layer

The service has three distinct phases −

- Set up of connection – A logical path is set up between the source and the destination machines. Buffers and counters are initialized to keep track of frames.
- Sending frames – The frames are transmitted.
- Release connection – The connection is released, buffers and other resources are released.

It is appropriate for satellite communications and long-distance telephone circuits.

Functions of data link layer

2. Determines how the bits of the physical layer are grouped into frames (framing)

The Data Link Layer is the second layer in the OSI model, above the Physical Layer, which ensures that the error free data is transferred between the adjacent nodes in the network. It breaks the datagram passed down by above layers and converts them into frames ready for transfer. This is called Framing.

Framing

2 types of framing

Fixed-Size Framing

- Frames can be of fixed or variable size.
- In fixed-size framing, there is no need for defining the boundaries of the frames; the size itself can be used as a delimiter

Variable size framing

- The contents of each frame are encapsulated between a pair of

reserved characters or bytes for frame synchronization.

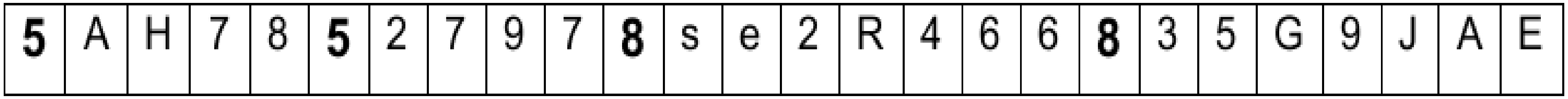
There are four methods:

1. Character count.
2. Flag bytes with byte stuffing.
3. Starting and ending flags, with bit stuffing.
4. Physical layer coding violations.

Framing

1. **Character count**

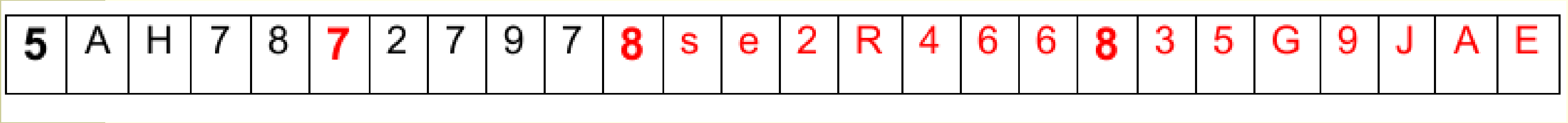
- The first framing method that is character count method uses a field in the header to specify the number of characters in the frame. When the data link layer at the destination sees the character count, it knows how many characters follow and also can come to know where the frame will end.
- There are four frames in this figure with size 5, 5, 8 and 8. The first frame has data A, H, 7 and 8…..



- There is one problem with this method. The count can be garbled by a transmission error

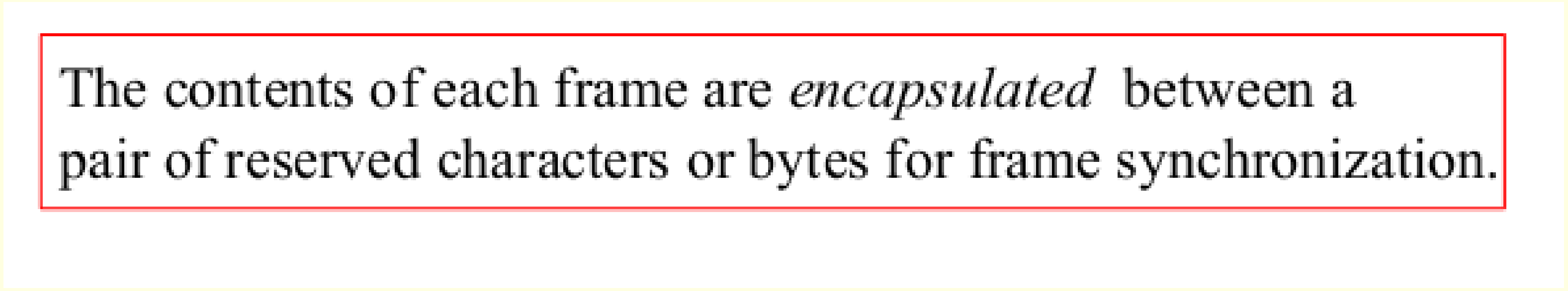
Functions of data link layer

- If the character count of 5 in the second frame becomes 7, the destination will get out of synchronization and will be unable to locate the start of the next frame.
- For this reason the character count method is rarely used

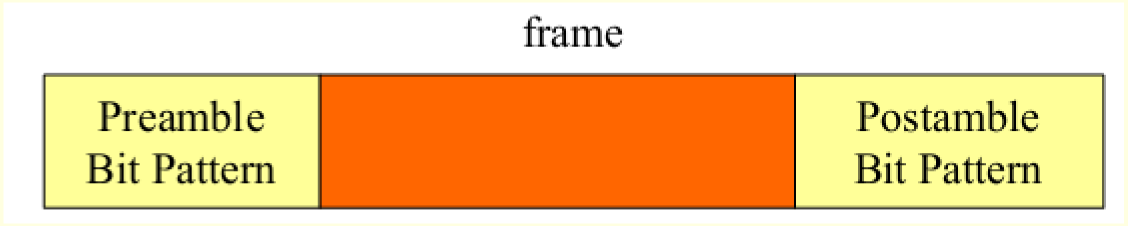


5	A	H	7	8	**7**	2	7	9	7	**8**	s	e	2	R	4	6	6	**8**	3	5	G	9	J	A	E

Alternative to character count



The contents of each frame are *encapsulated* between a pair of reserved characters or bytes for frame synchronization.



Framing

2. Flag bytes with byte stuffing.

- The second framing method is flag bytes insertion using byte stuffing. This is also referred to as character stuffing.
- ASCII characters are used as framing delimiters for example DLE STX (start of the frame) and DLE ETX(End of the frame).
- If however, binary data is being transmitted then there exists a possibility of the characters DLE STX and DLE ETX occurring in the data. Since this can interfere with the framing, a technique called character stuffing is used. The sender's data link layer inserts an ASCII DLE character just before the DLE character in the data. The receiver's data link layer removes this DLE before this data is given to the network layer. However character stuffing is closely associated with 8-bit characters and this is a major hurdle in transmitting arbitrary sized characters. The problem occurs when these character patterns occur within the data.

Framing

- This framing mechanism uses the special ASCII characters 0x10 (called DLE or Data Link Escape), 0x02 (STX, Start of Text), 0x03 (ETX, End of Text). Each packet is framed as between "DLE STX" and "DLE ETX" as follows:

```
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5
|   DLE          |    STX      | Payload (DLEs byte-stuffed) |  ... | DLE | ETX |
```

- So, when sending a packet on a stream, just transmit "DLE STX", then the payload of bytes (the content of the packet), and finally "DLE ETX".
- When sending the payload data, each DLE character in the data must be doubled (i.e., DLE becomes DLE DLE).

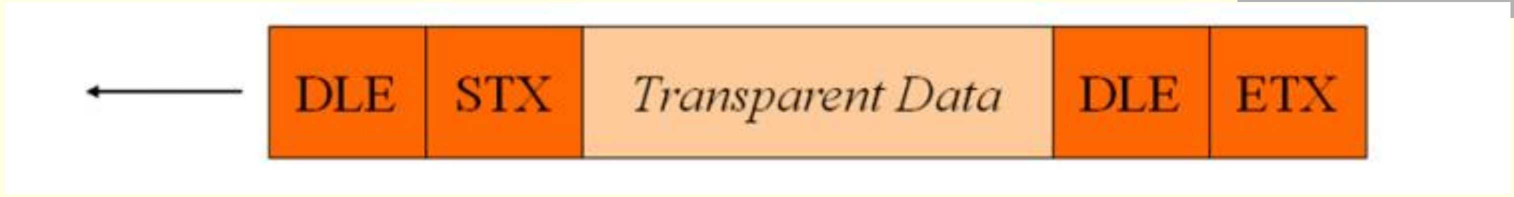
Framing

- Solution to this problem is to stuff an extra DLE into the data stream just before each occurrence of an “accidental” DLE in the data stream. The data link layer on the receiving end unstuffs the DLE before giving the data to the network layer as shown in this figure. The flag byte DLE and STX are inserted at the start and DLE ETX bytes are inserted in the end.

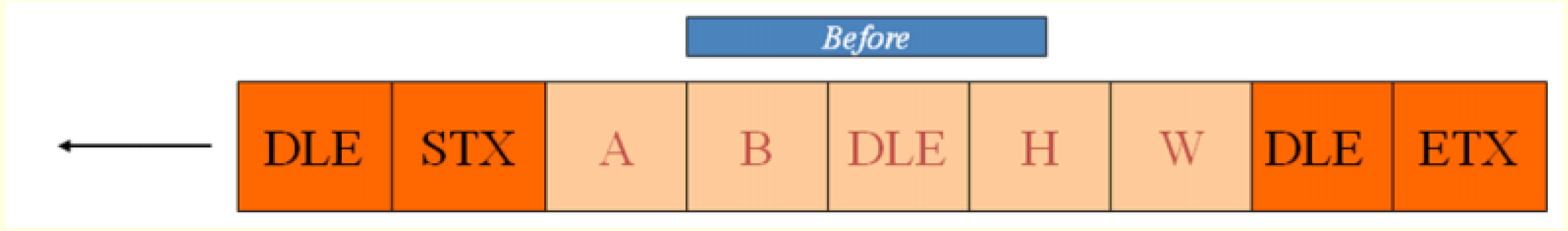
Framing

- If you want to transmit the payload "0x10“.
- The final packet is 0x10 0x02 0x10 0x10 0x10 0x03 (or DLE STX DLE DLE DLE ETX).
- A receiver must therefore look for packet start (i.e. DLE STX), read the payload and remove the second DLE if two consecutive ones are received (de-stuffing), and continue buffering until DLE ETX is found.

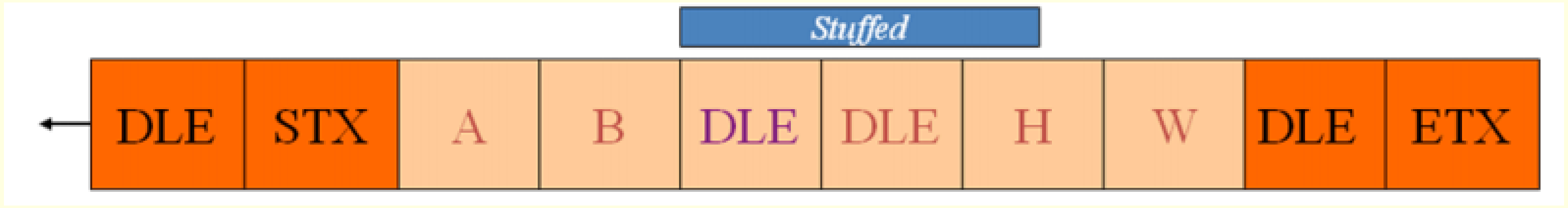
Framing



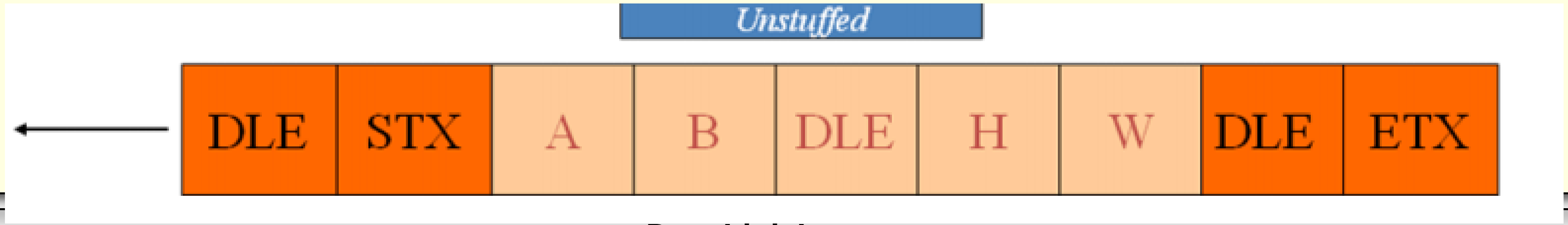
DLE
STX
Transparent Data
DLE
ETX



Before
DLE
STX
A
B
DLE
H
W
DLE
ETX



Stuffed
DLE
STX
A
B
DLE
DLE
H
W
DLE
ETX



Unstuffed
DLE
STX
A
B
DLE
H
W
DLE
ETX

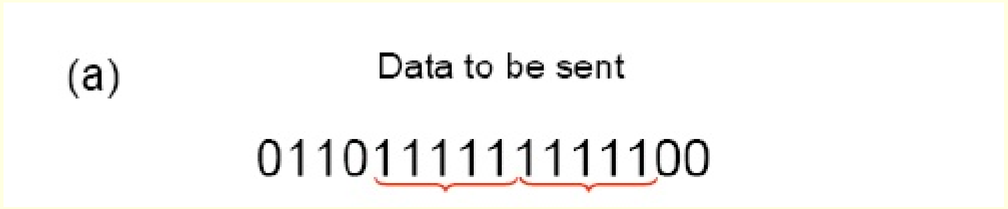
Framing

3. Bit Stuffing

- In this method each frame begins and ends with a special bit pattern called a flag byte which is 01111110.
- In this technique also there is a possibility that this particular pattern of 01111110 may appear in data itself.
- In order to take care of this problem, whenever sender data link layer encounters five consecutive ones in the data stream, it automatically stuff a 0 bit into the outgoing stream.
- When the receiver sees five consecutive incoming ones followed by a 0 bit , it automatically destuffs the 0 bit before sending the data to the network layer.

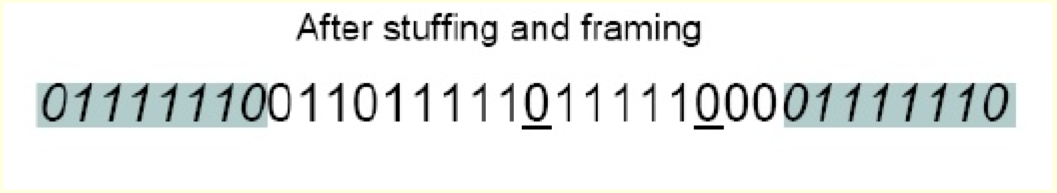
Framing

- Eg :1



(a) Data to be sent

$0110\underbrace{11111}\underbrace{11111}00$

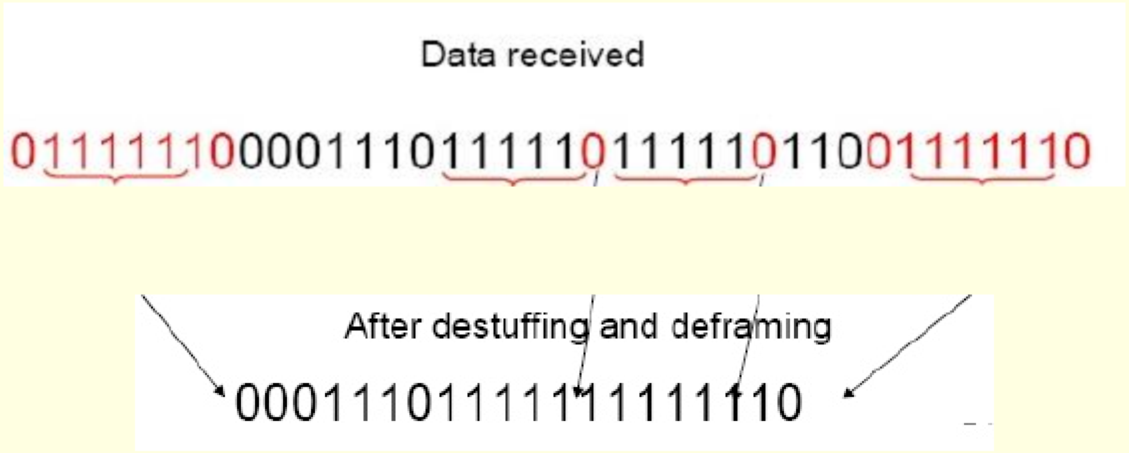


After stuffing and framing

*01111110*011011111$\underline{0}$11111$\underline{0}$00*01111110*

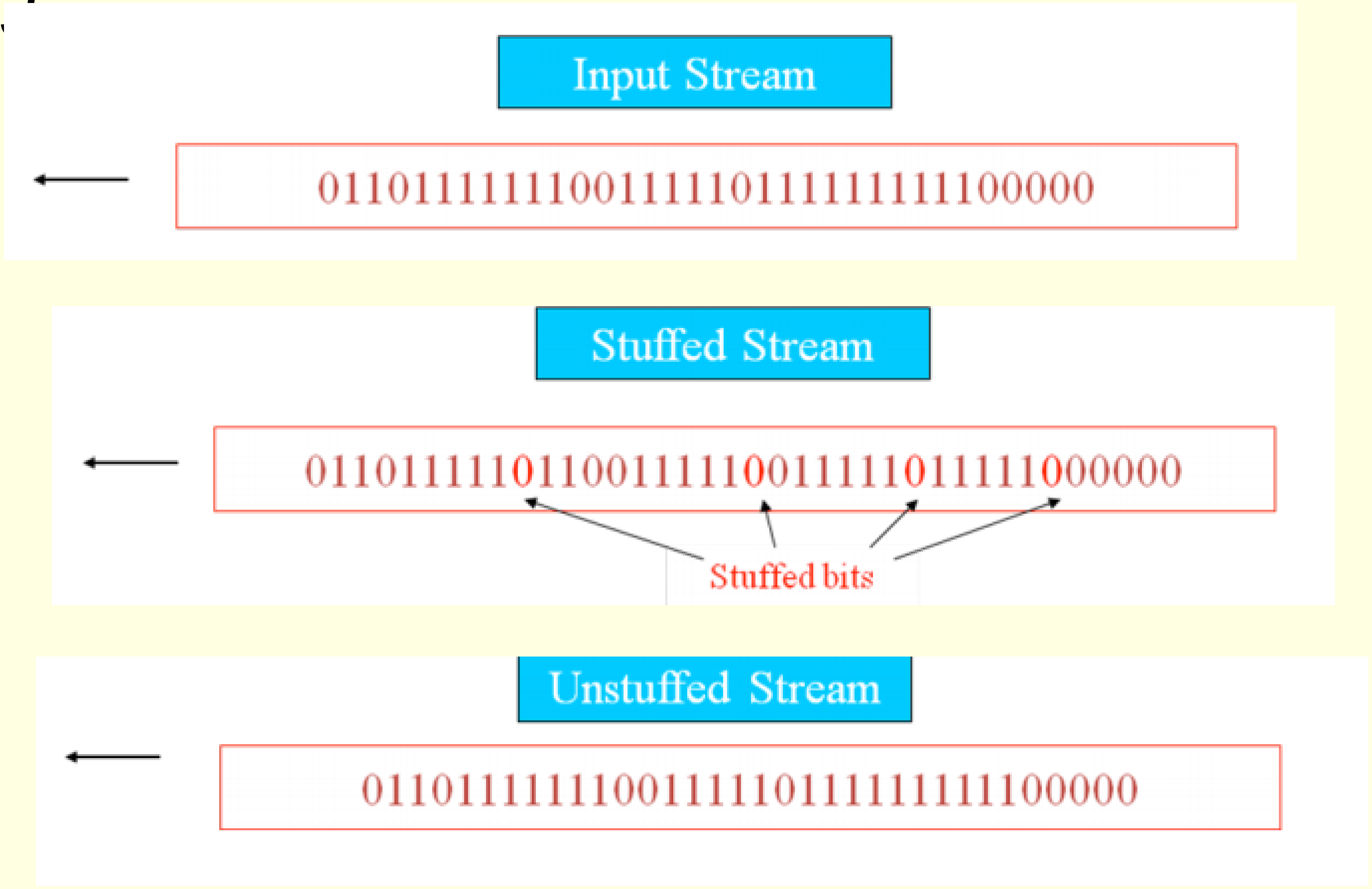
Framing

Eg:2


Data received
01111110000111011111011111011001111110
After destuffing and deframing
0001110111111111110

Framing

- Eg: 3


Input Stream
0110111111100111110111111111100000
Stuffed Stream
01101111101100111110011111011111000000
Stuffed bits
Unstuffed Stream
0110111111100111110111111111100000

Error detection and correction

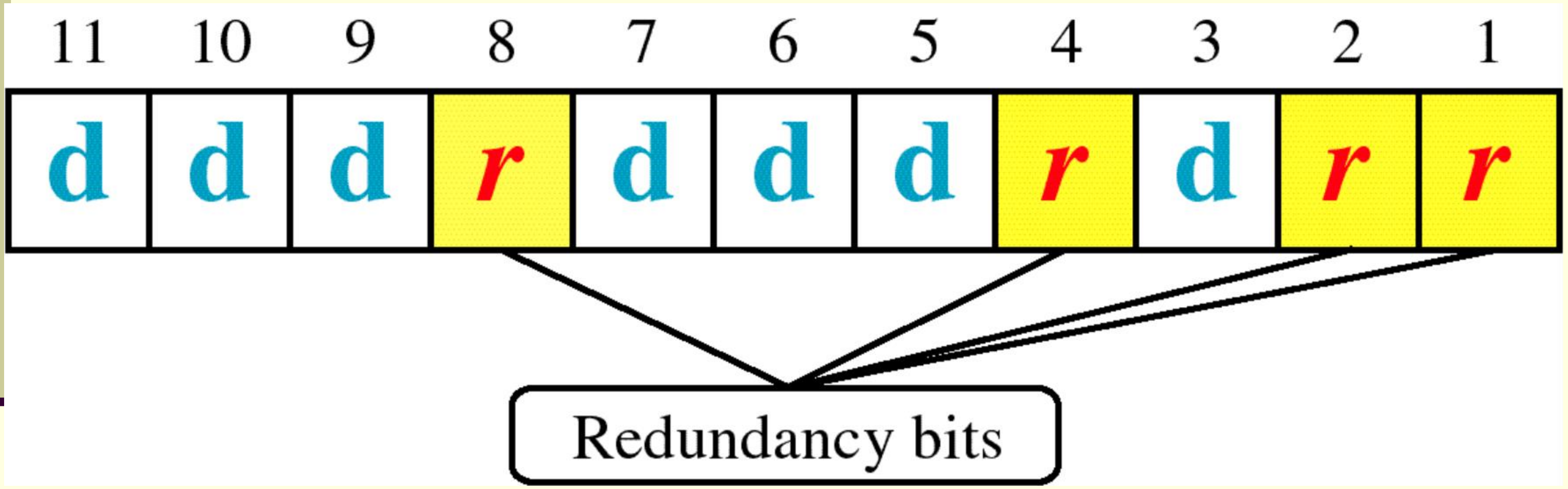
- Networks must be able to transfer data from one device to another with complete accuracy.
- Data can be corrupted during transmission.
- For reliable communication, errors must be detected and corrected.
- **Error detection and correction** are implemented either at the **data link layer** or the **transport layer** of the OSI model.
- Error Detection and Correction method----→ Hamming Code(single bit error correction and double bit error detection)

Error Detection Methods

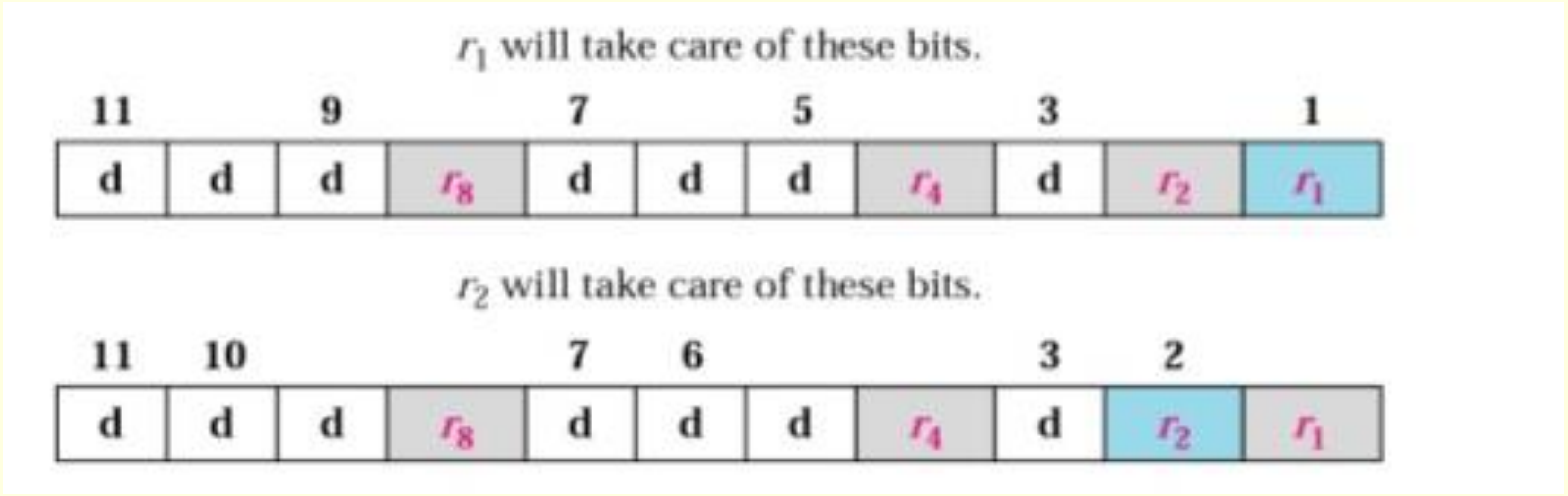
Error Detection Methods :

- CRC
- Checksum

Hamming Code

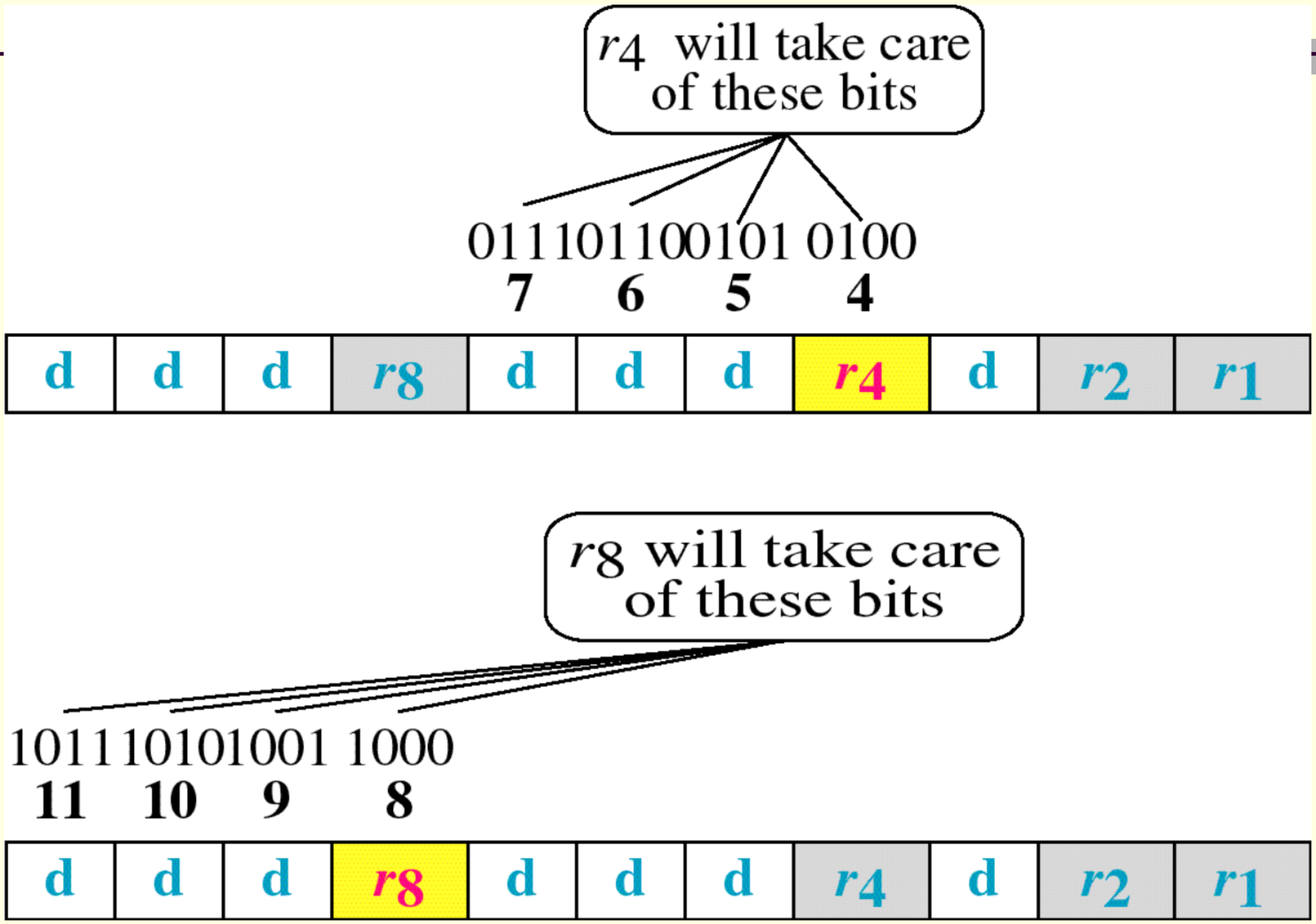


Hamming Code

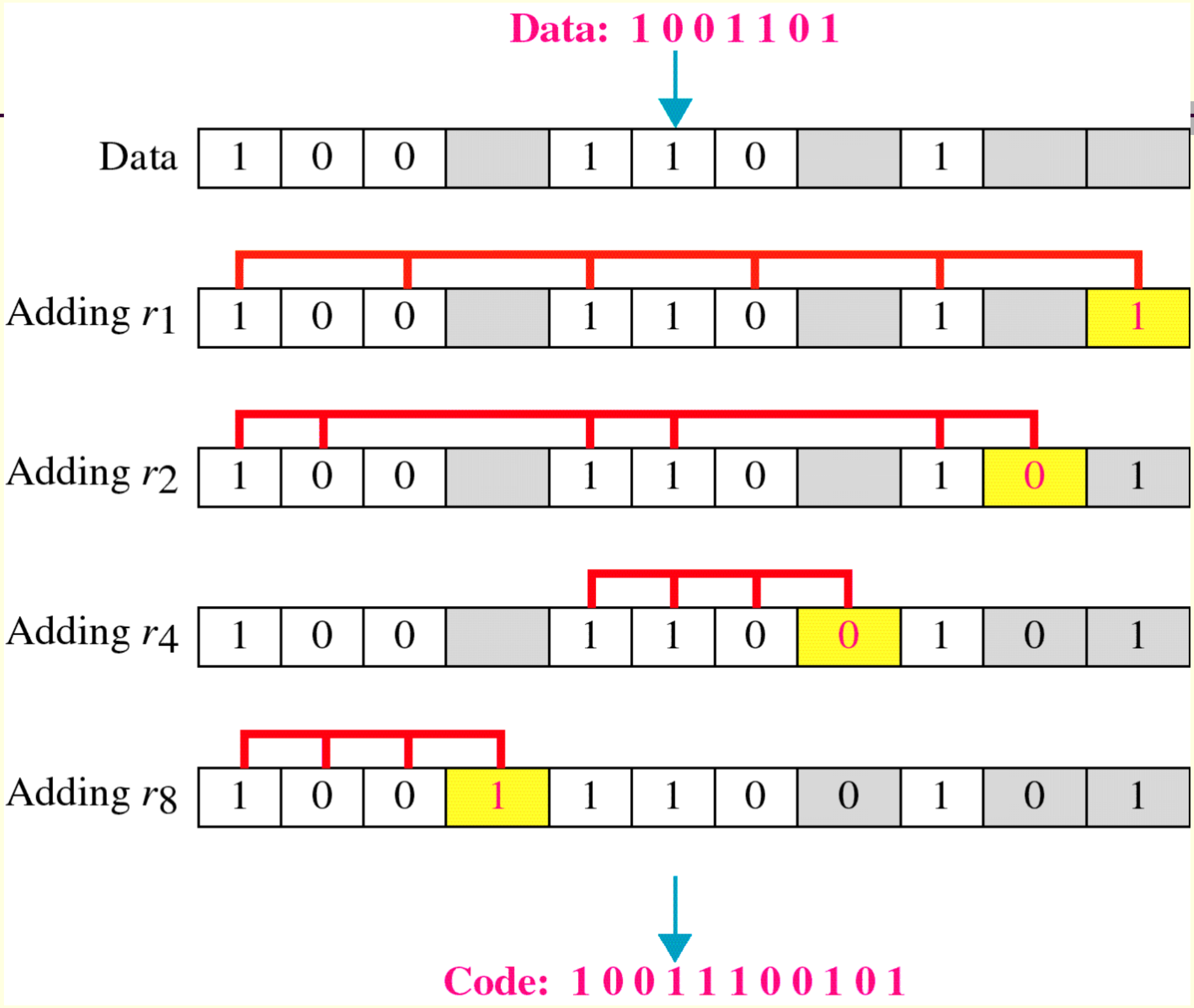


r_1 will take care of these bits.
11
9
7
5
3
1
d
d
d
r_8
d
d
d
r_4
d
r_2
r_1
r_2 will take care of these bits.
11
10
7
6
3
2

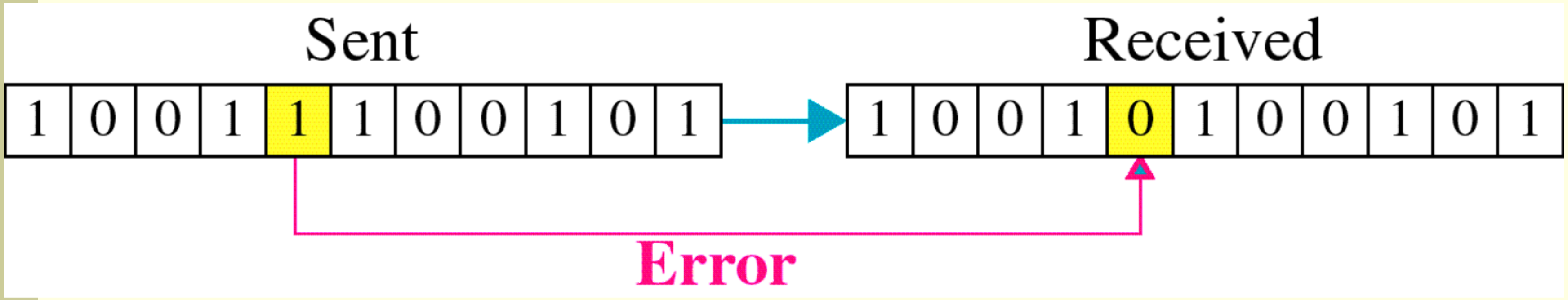
Hamming Code


r4 will take care of these bits
0111 1011 1001 01 0100
7 6 5 4
d d d r8 d d d r4 d r2 r1
r8 will take care of these bits
1011 1010 1001 1000
11 10 9 8
d d d r8 d d d r4 d r2 r1

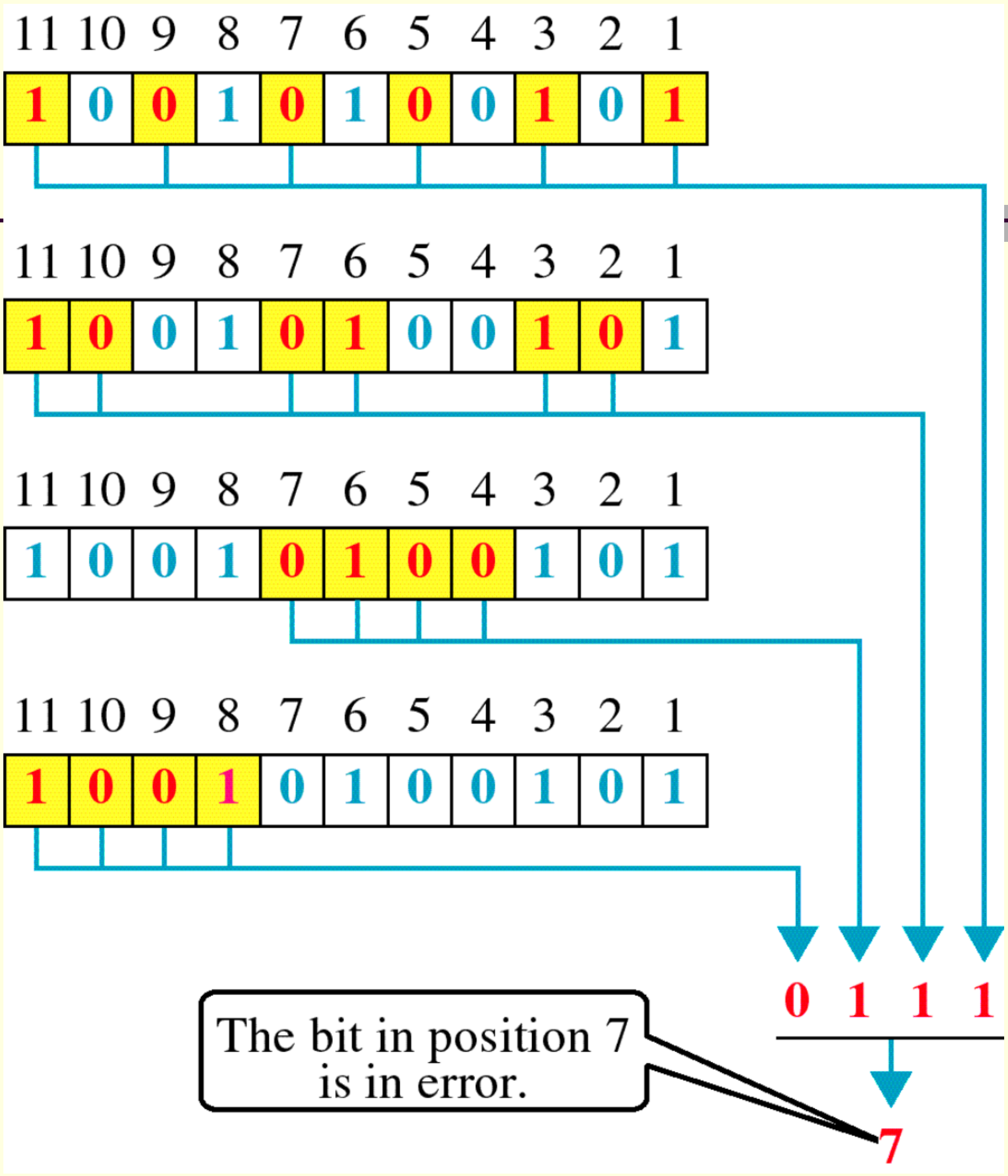
Example of Hamming Code



Single-bit error



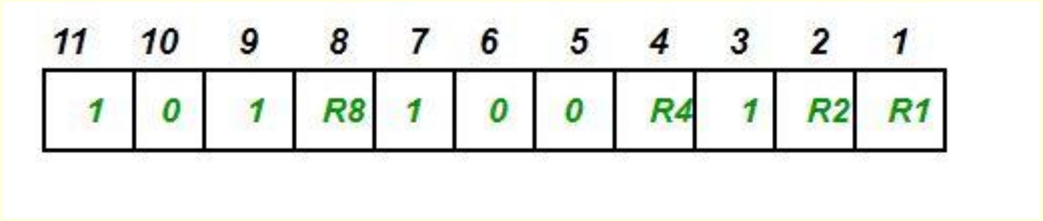
Error Detection



→ r

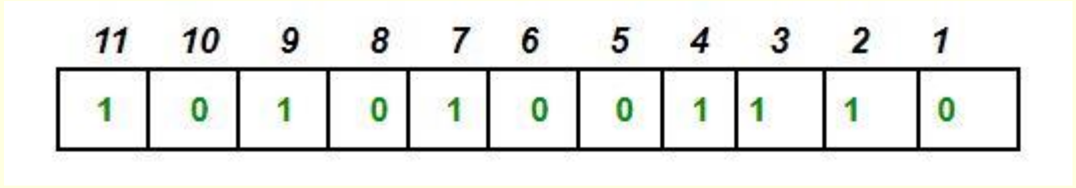
Hamming code

- Eg: 2
- Suppose the data to be transmitted is 1011001, the bits will be placed as follows:



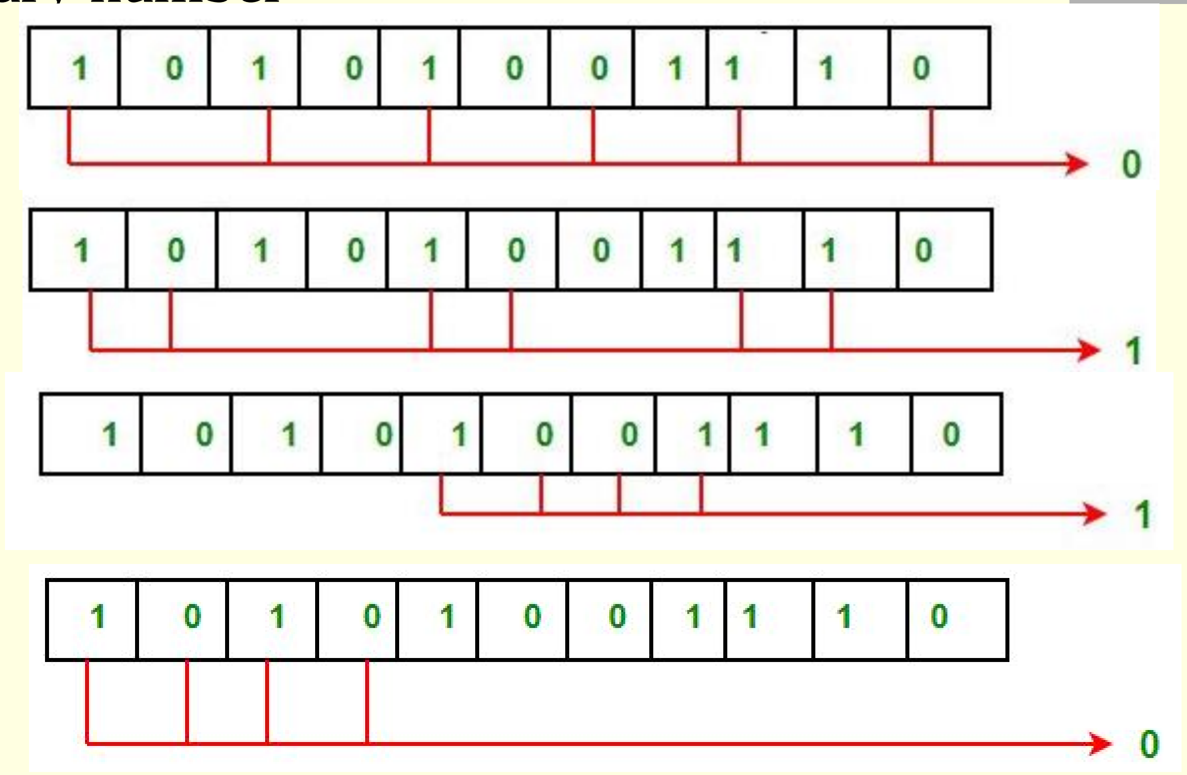
11	10	9	8	7	6	5	4	3	2	1
1	0	1	R8	1	0	0	R4	1	R2	R1

Thus, the data transferred is:



11	10	9	8	7	6	5	4	3	2	1
1	0	1	0	1	0	0	1	1	1	0

- **Error detection and correction –**
  Suppose in the above example the 6th bit is changed from 0 to 1 during data transmission, then it gives new parity values in the binary number



- The bits give the binary number as 0110 whose decimal representation is 6. Thus, the bit 6 contains an error. To correct the error the 6th bit is changed from 1 to 0.

Hamming Code

Double bit error detection

- The Hamming code can be modified to detect double bit errors by adding a parity bit as the MSB, which is the XOR of all other bits.
- Suppose that the message 1100101 needs to be encoded using even parity Hamming code.
- Hence, the message sent will be, 11000101100 (after adding the redundant or parity bits).
- After adding P = XOR(1,1,0,0,0,1,0,1,1,0,0) = 1, the new codeword to be sent will be **1**11000101100.

Hamming Code

- At the receiver's end, error detection is done as shown in the following table .
- Note : r- value is the binary value obtained after checking the redundant bit positions

r-value	P value	Conclusion
r=0	P(sender)=P(receiver)	No error
r=0	P(sender)≠P(receiver)	Error in P bit, Data can be sent to the uppers layers after removing all check bits.
r≠0	P(sender)≠P(receiver)	Single bit error occurred that can be corrected by reversing the bit value at the bit position given by value r
r≠0	P(sender)=P(receiver)	Double bit error detected that cannot be corrected.

Hamming Code

Eg :

- Suppose that the message 1100101 needs to be encoded using even parity Hamming code.
- Hence, the message sent will be, 11000101100 (after adding the redundant or parity bits).
- After adding P = XOR(1,1,0,0,0,1,0,1,1,0,0) = 1, the new codeword to be sent will be **1**11000101100.
- Assuming 2 bits are changed(last 2 lsb bits changed to 11)
- Received code word – **1**1100010111**11**
- New parity -1
- r≠0
- Hence   Double bit error detected

Cyclic Redundancy Check

- CRC or Cyclic Redundancy Check is a method of detecting errors in communication channel.
- Given a k-bit frame or message, the transmitter generates an n-bit sequence, known as a *frame check sequence (FCS)*, so that the resulting frame, consisting of (k+n) bits
- Bit sequences can be written as polynomials with the coefficients 0 and 1.
- A frame with k bits is considered as a polynomial of degree k−1.
- The most significant bit is the coefficient of x^{k-1} the next bit is the coefficient of x^{k-2}. . . Example: The bit sequence 10011010 corresponds to this polynomial:

$$\begin{aligned} M(x) &= 1*x^7+0*x^6+0*x^5+1*x^4+1*x^3+0*x^2+1*x^1+0*x^0 \\ &= x^7+x^4+x^3+x^1 \end{aligned}$$

- Sending and receiving messages can be imagined as an exchange of polynomials

Cyclic Redundancy Check

- The Data Link Layer protocol specifies a generator polynomial C(x). Generator Polynomial is available on both sender and receiver side.
- *C(x) is a polynomial of degree k*
- If e.g. *C(x) = x3 + x2 + x0*
- *= 1101, then k = 3*
- Therefore, the generator polynomial is of degree 3
- The degree of the generator polynomial is equal to the number of bits minus one.
- If for a frame, the CRC need to be calculated, n 0 bits are appended to the frame
- n corresponds to the degree of the generator polynomial

Cyclic Redundancy Check

Generator polynomial:	100110

- The generator polynomial has 6 digits
  - Therefore, five 0 bits are appended

Frame (payload):	10101
Frame with appended 0 bits:	1010100000

Cyclic Redundancy Check

Sender Side (Generation of Encoded Data from Data and Generator Polynomial (or Key)):

- The binary data is first augmented by adding n zeros in the end of the data. (n- degree of the generator polynomial)
- Use ***modulo-2 binary division*** to divide binary data by the key and store remainder of division.
- Append the remainder at the end of the data to form the encoded data and send the same.

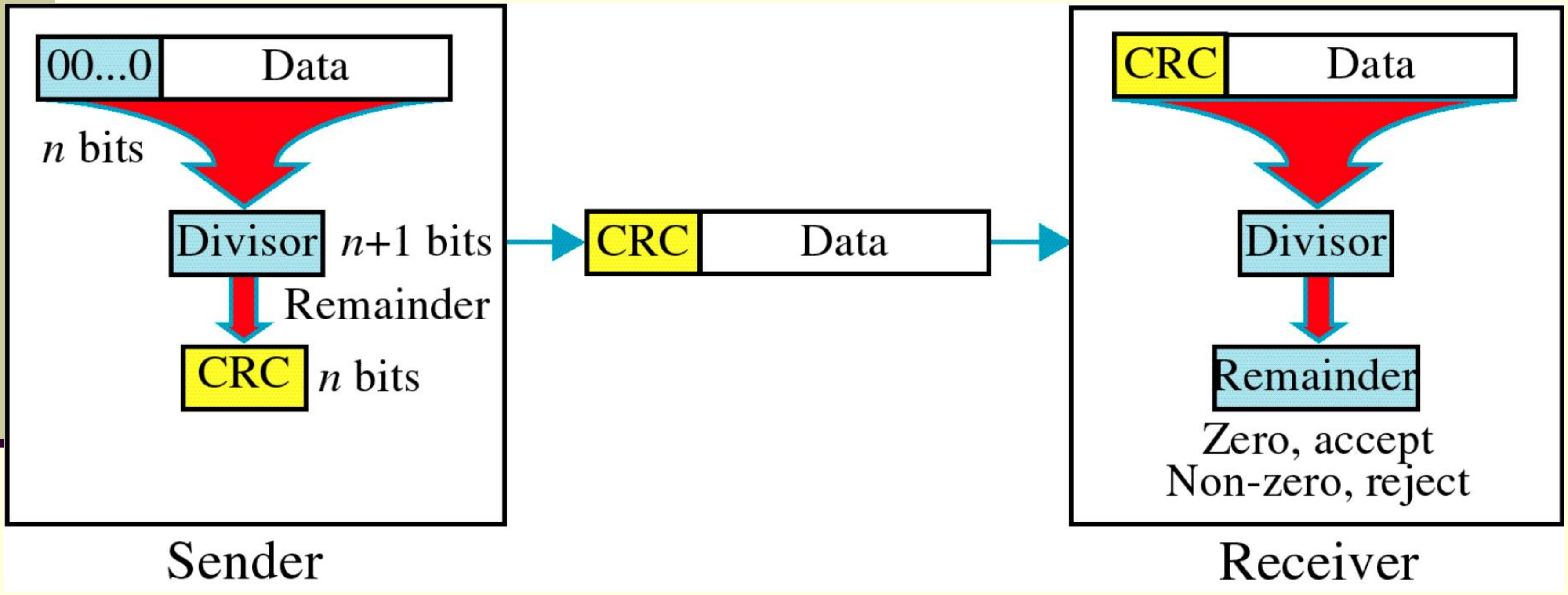
Receiver Side (Check if there are errors introduced in transmission)

Perform modulo-2 division again and if remainder is 0, then there are no errors.

Cyclic Redundancy Check

- **Modulo 2 Division:**
  The process of modulo-2 binary division is the same as the familiar division process we use for decimal numbers. Just that instead of subtraction, we use XOR here.
- In each step, a copy of the divisor is XORed with the n+1 bits of the dividend (or key).
- The result of the XOR operation (remainder) is n bits, which is used for the next step after 1 extra bit is pulled down to make it n+1 bits long.
- When there are no bits left to pull down, we have a result. The n-bit remainder which is appended at the sender side.

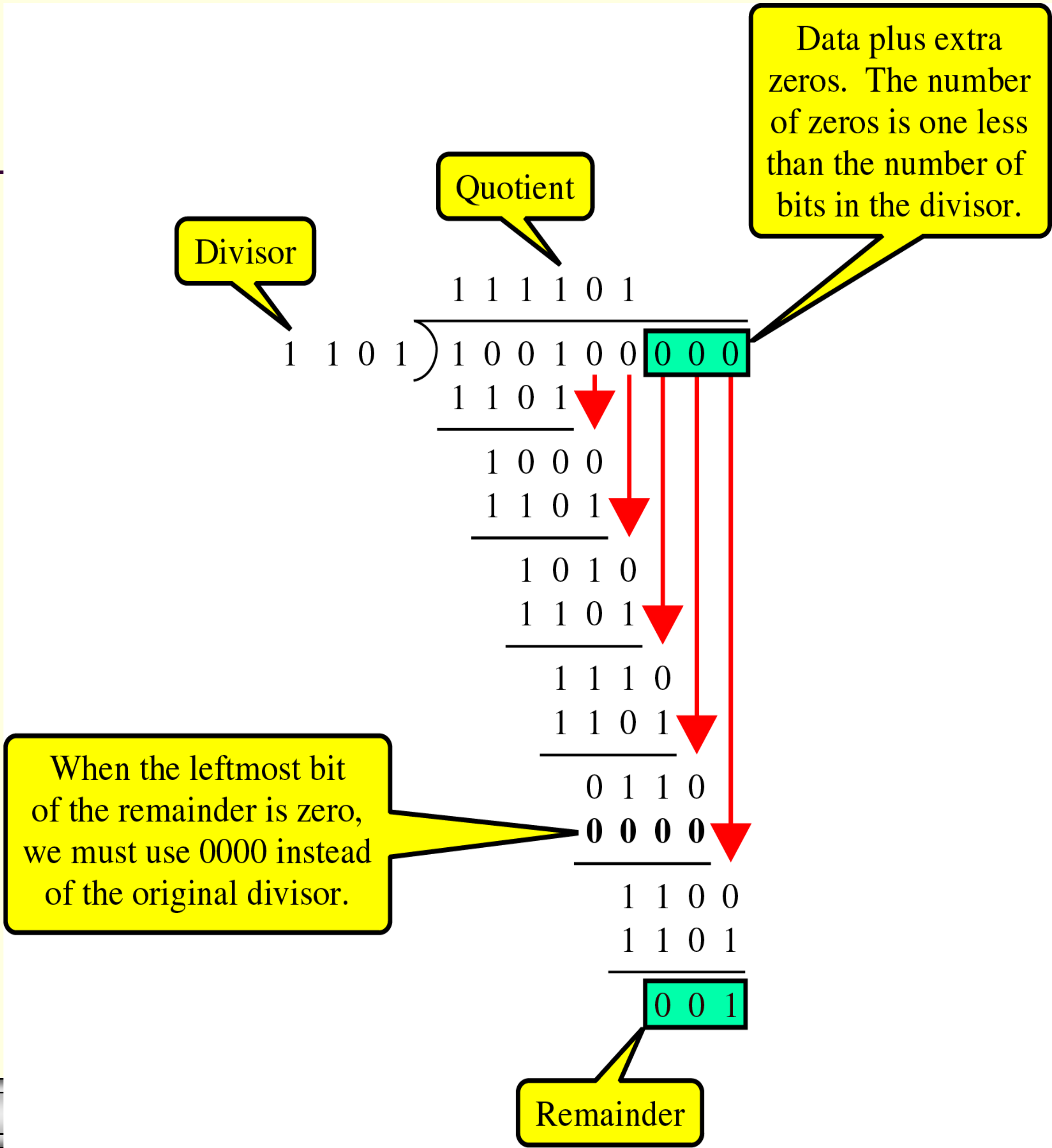
Cyclic Redundancy Check
CRC

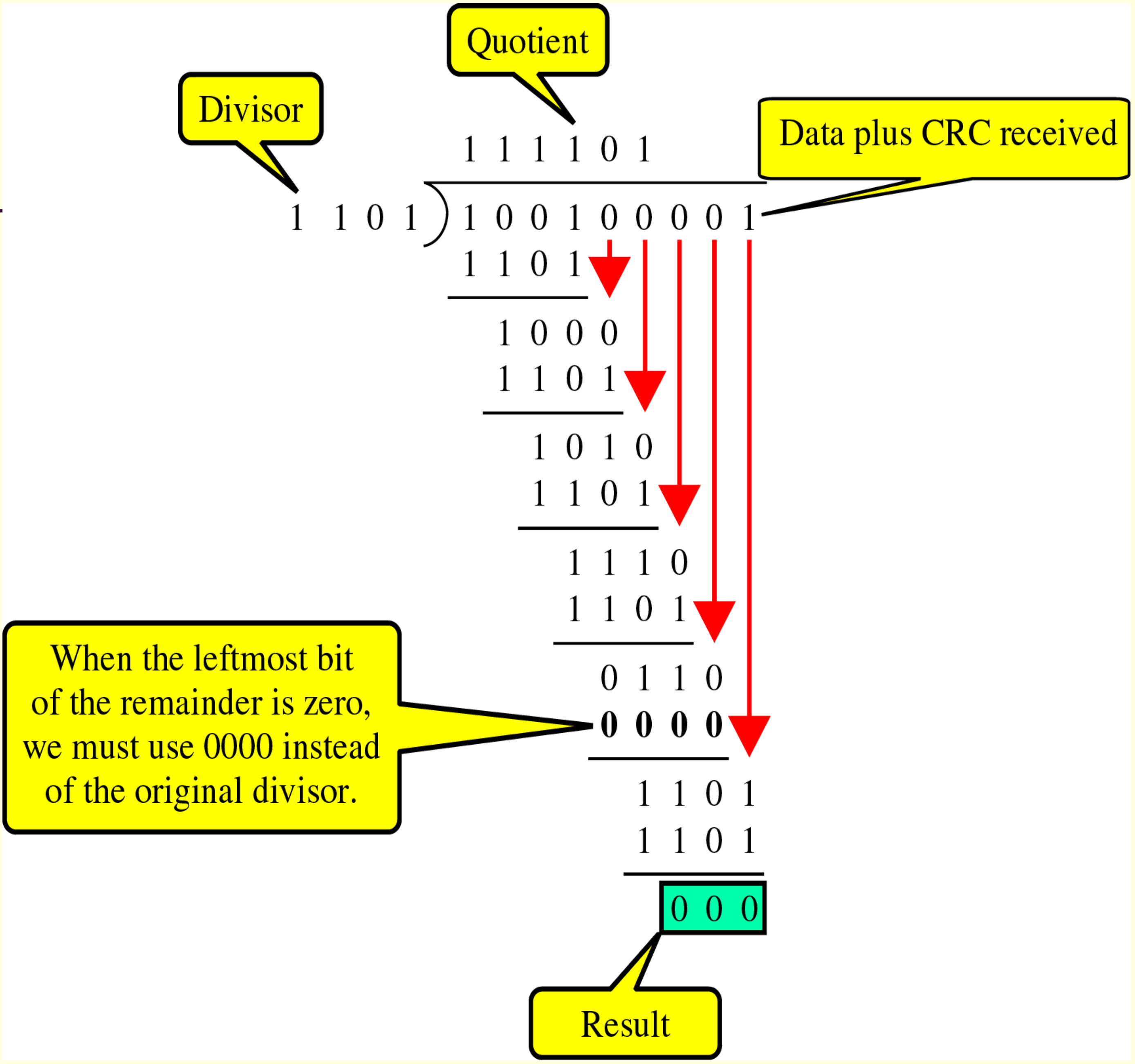


CRC generator

uses modular-2 division.

Bi
in
CR


Data plus extra zeros. The number of zeros is one less than the number of bits in the divisor.
Quotient
Divisor
1 1 1 1 0 1
1 1 0 1
1 0 0 1 0 0 0 0 0
1 1 0 1
1 0 0 0
1 1 0 1
1 0 1 0
1 1 0 1
1 1 1 0
1 1 0 1
0 1 1 0
0 0 0 0
1 1 0 0
1 1 0 1
0 0 1
When the leftmost bit of the remainder is zero, we must use 0000 instead of the original divisor.
Remainder

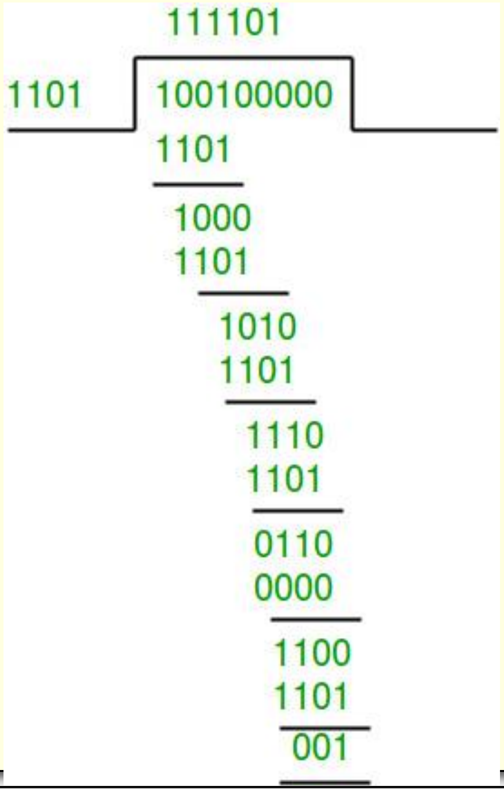

Quotient
Divisor
Data plus CRC received
1 1 1 1 0 1
1 1 0 1
1 0 0 1 0 0 0 0 1
1 1 0 1
1 0 0 0
1 1 0 1
1 0 1 0
1 1 0 1
1 1 1 0
1 1 0 1
0 1 1 0
0 0 0 0
1 1 0 1
1 1 0 1
0 0 0
When the leftmost bit of the remainder is zero, we must use 0000 instead of the original divisor.
Result

Cyclic Redundancy Check

- Therefore, the remainder is all zeros. Hence, the data received has no error.

Example 2: (Error in transmission)

- Data word to be sent - 100100
- Key - 1101

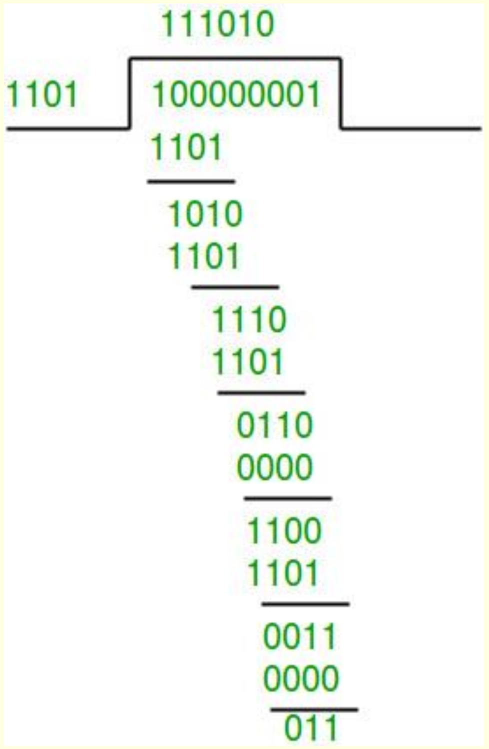


Cyclic Redundancy Check

- Therefore, the remainder is 001 and hence the code word sent is 100100001.

Receiver Side

- Let there be error in transmission media
- Code word received at the
- receiver side - 100000001



```
           111010
1101 | 100000001
       1101
        1010
        1101
         1110
         1101
          0110
          0000
           1100
           1101
            0011
            0000
             011
```

Cyclic Redundancy Check

- Since the remainder is not all zeroes, the error is detected at the receiver side.

Cyclic Redundancy Check

Eg :3

original message

1010000

Generator polynomial

x^3+1

$1.x^3+0.x^2+0.x^1+1.x^0$

CRC generator

1001 4-bit

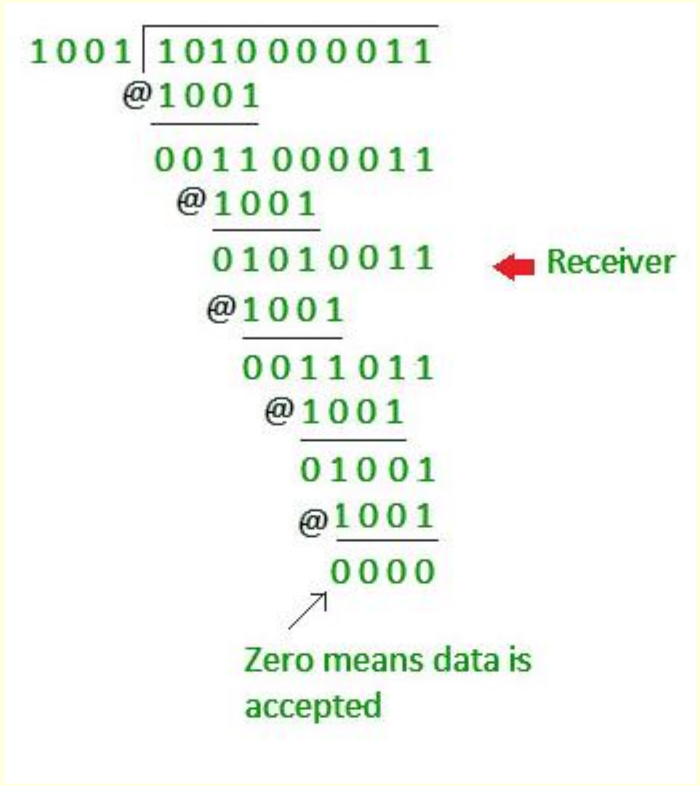
Sender

```
1001 | 1010000000
       1001
       ----
       001100
         1001
         ----
         01010
          1001
          ----
          001100
             1001
             ----
             01010
            @1001
             ----
             0011
```

Message to be transcribed

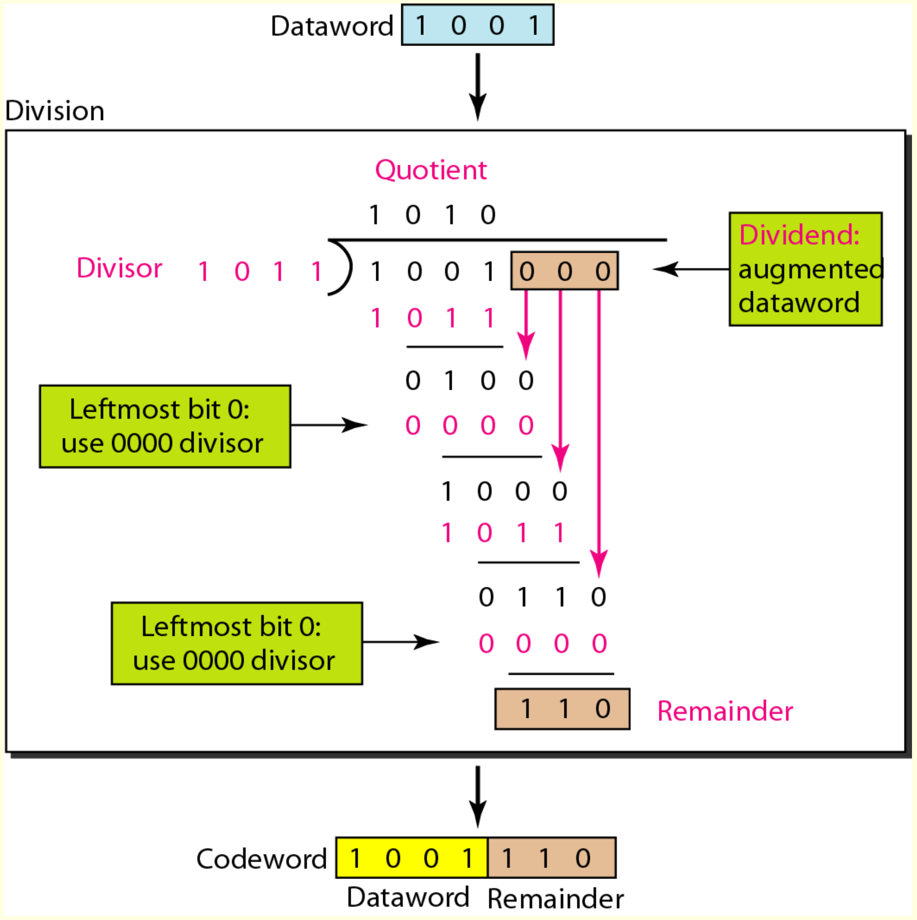
```
1010000000
      +011
----------
1010000011
```

Cyclic Redundancy Check

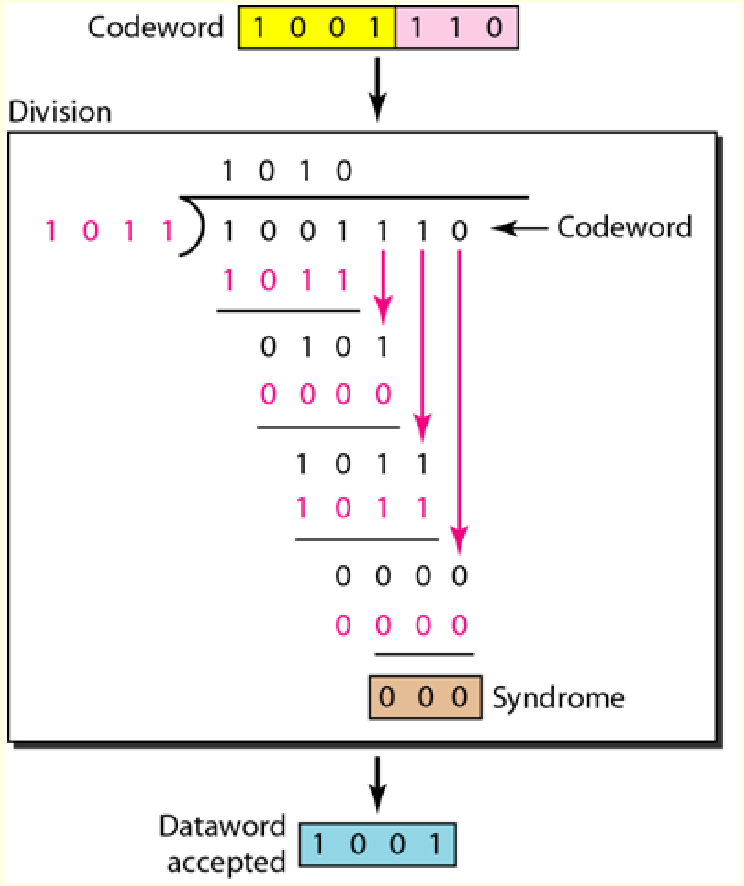


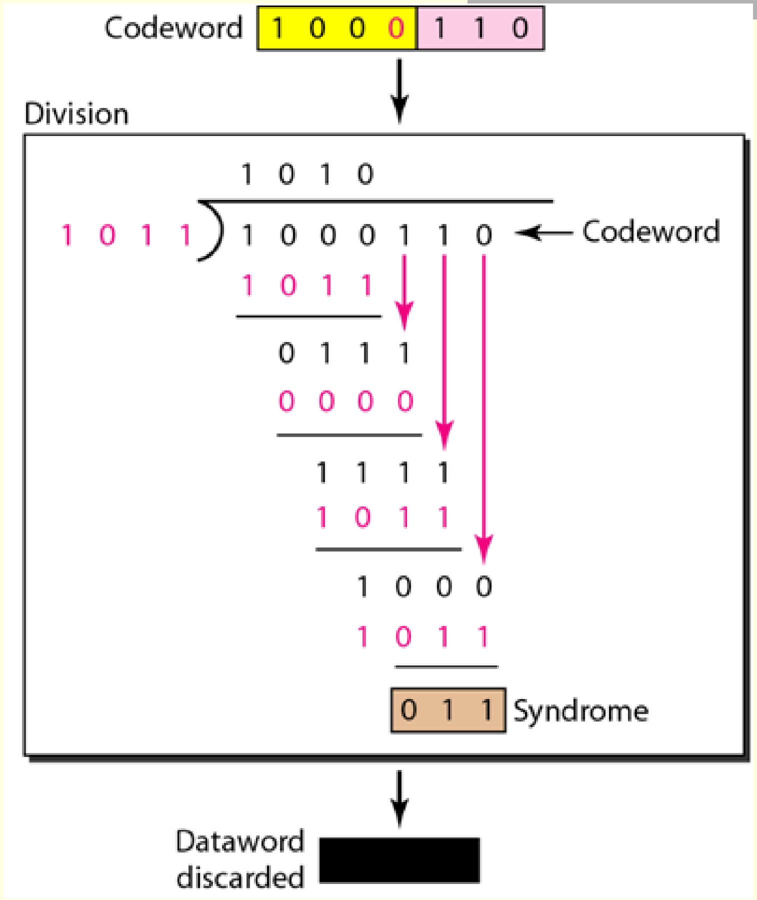
Cyclic Redundancy Check

- Ex:4



Cyclic Redundancy Check





CRC

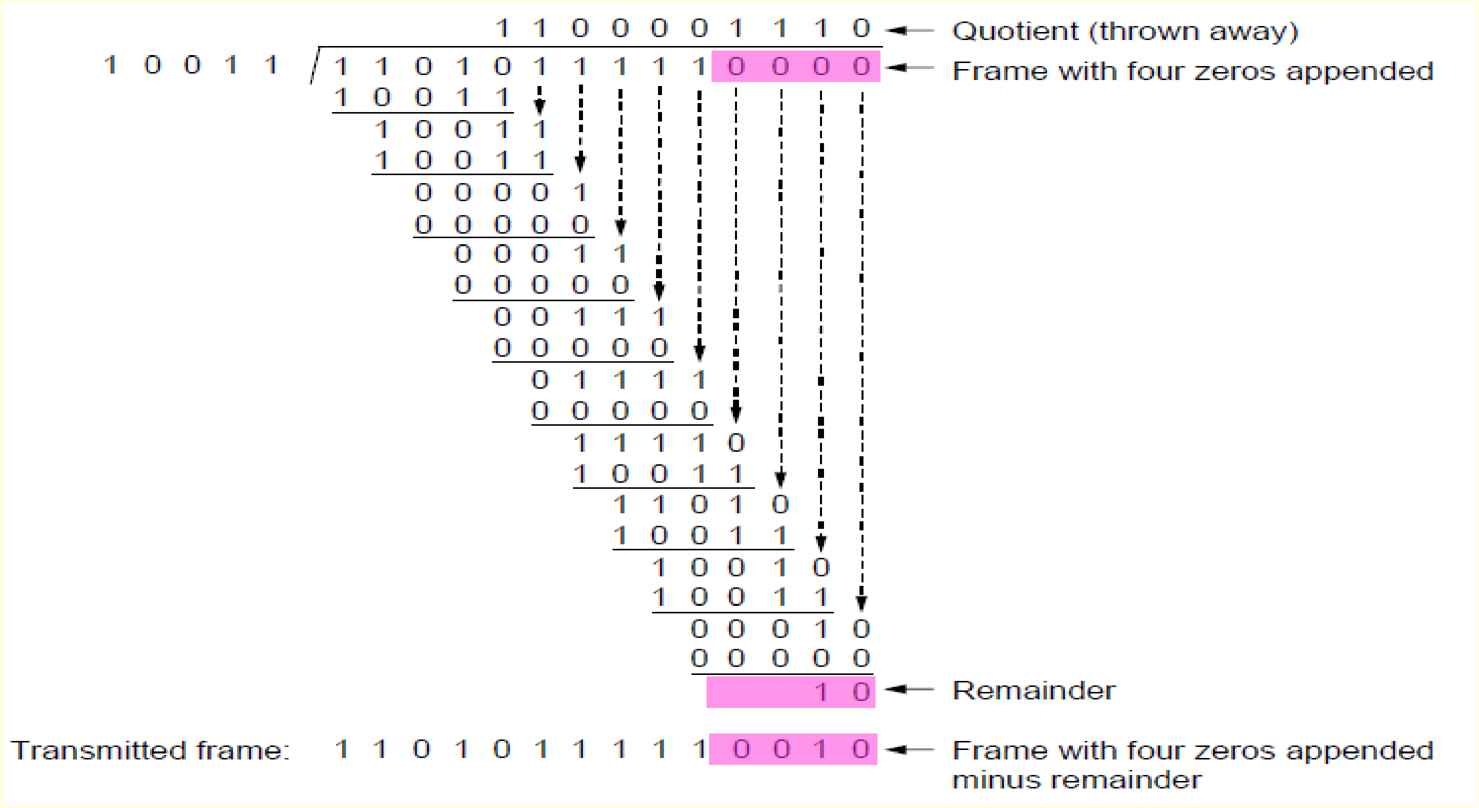
- Ex:

Data bits:
1101011111

$$10011 \overline{)\,1\ 1\ 0\ 1\ 0\ 1\ 1\ 1\ 1\ 1}$$

Check bits:
$C(x)=x^4+x^1+1$
C = 10011
k = 4

CRC

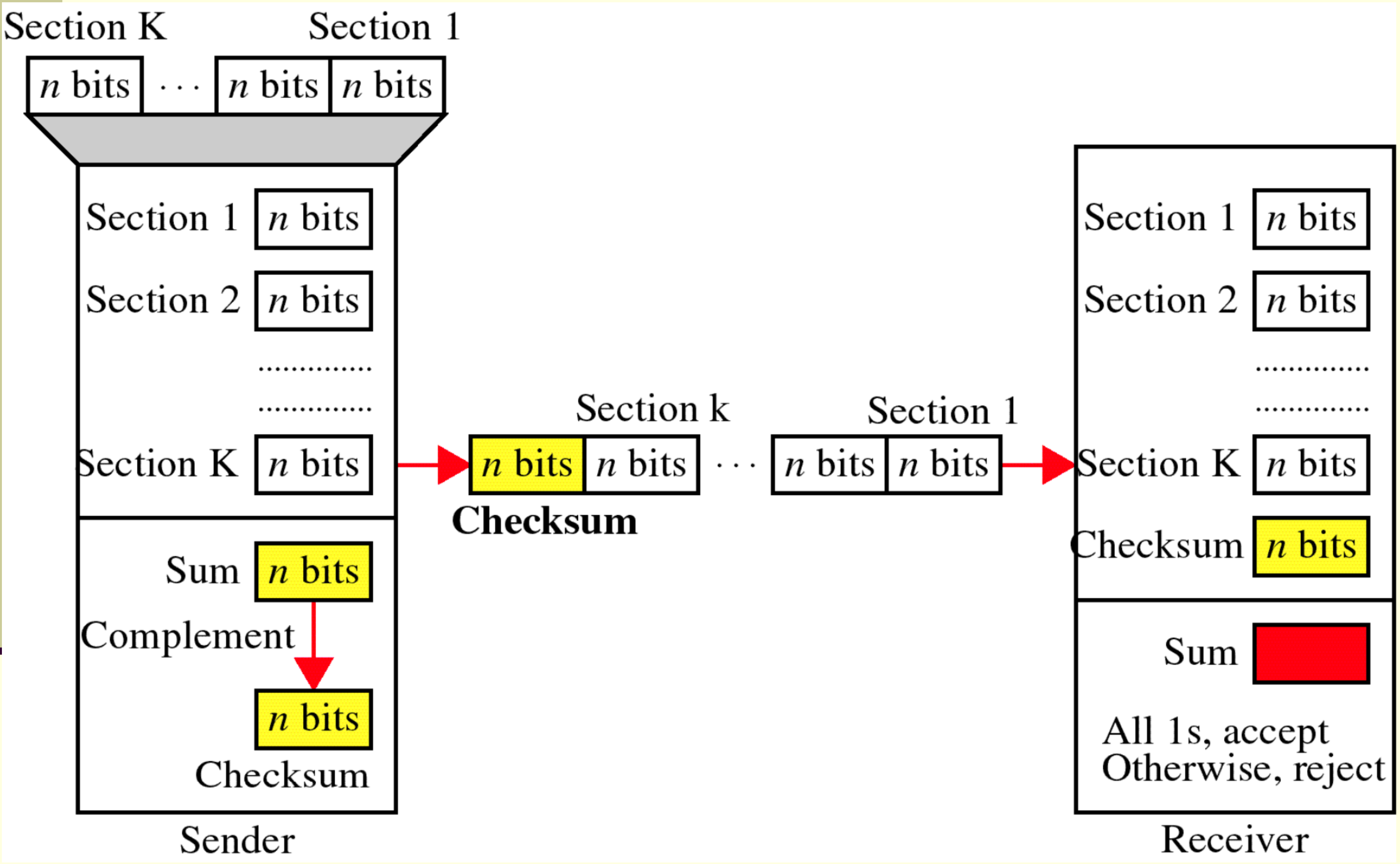


```
                         1 1 0 0 0 0 1 1 1 0
1 0 0 1 1  /  1 1 0 1 0 1 1 1 1 1 0 0 0 0
              1 0 0 1 1
                1 0 0 1 1
                1 0 0 1 1
                  0 0 0 0 1
                  0 0 0 0 0
                    0 0 0 1 1
                    0 0 0 0 0
                      0 0 1 1 1
                      0 0 0 0 0
                        0 1 1 1 1
                        0 0 0 0 0
                          1 1 1 1 0
                          1 0 0 1 1
                            1 1 0 1 0
                            1 0 0 1 1
                              1 0 0 1 0
                              1 0 0 1 1
                                0 0 0 1 0
                                0 0 0 0 0
                                      1 0

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

Quotient (thrown away)
Frame with four zeros appended
Remainder
Frame with four zeros appended minus remainder

Checksum


Section K
Section 1
n bits
n bits
n bits
Section 1
Section 2
Section K
Sum
Complement
Checksum
Sender
Section k
Section 1
Checksum
Section 1
Section 2
Section K
Checksum
Sum
All 1s, accept
Otherwise, reject
Receiver

Checksum

At sender side,

- If m bit checksum is used, the data unit to be transmitted is divided into segments of m bits.
- All the m bit segments are added.
- The result of the sum is then complemented using 1's complement arithmetic.
- The value so obtained is called as **checksum**.
- The data along with the checksum value is transmitted to the receiver.

Checksum

At receiver side,

- If m bit checksum is being used, the received data unit is divided into segments of m bits.
- All the m bit segments are added along with the checksum value.
- The value so obtained is complemented and the result is checked.

Case-01: Result = 0

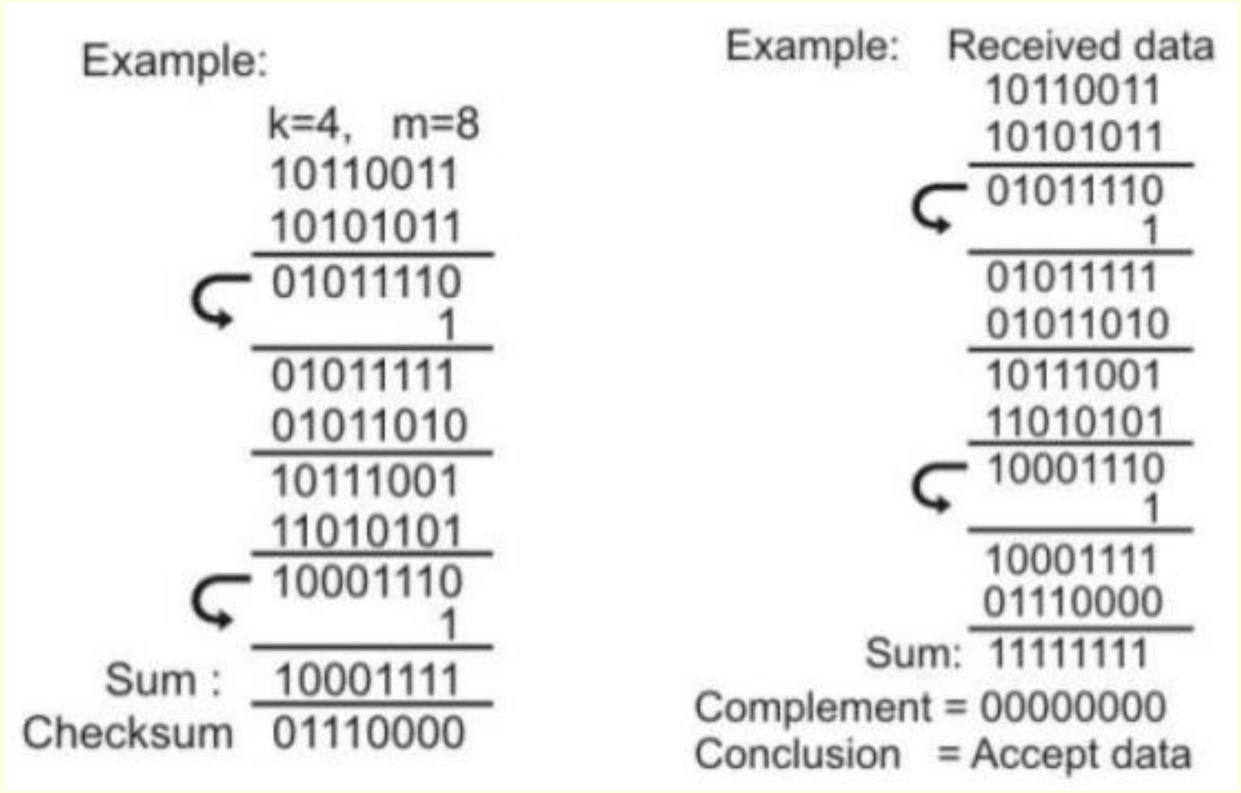
- Receiver assumes that no error occurred in the data during the transmission.
- Receiver accepts the data.

Case-02: Result ≠ 0

- Receiver assumes that error occurred in the data during the transmission.
- Receiver discards the data and asks the sender for retransmission.

Checksum

- Eg :
- **10110011 10101011 01011010 11010101**



```
Example:
            k=4,  m=8
             10110011
             10101011
            ---------
          ⤹  01011110
                    1
            ---------
             01011111
             01011010
            ---------
             10111001
             11010101
            ---------
          ⤹  10001110
                    1
            ---------
   Sum :     10001111
            ---------
Checksum     01110000
```

```
Example:    Received data
             10110011
             10101011
            ---------
          ⤹  01011110
                    1
            ---------
             01011111
             01011010
            ---------
             10111001
             11010101
            ---------
          ⤹  10001110
                    1
            ---------
             10001111
             01110000
            ---------
      Sum:   11111111
Complement = 00000000
Conclusion   = Accept data
```

Checksum

- Eg:2

(at a sender)

Original data : 10101001 00111001

10101001

00111001

--------------

11100010 Sum

00011101 Checksum

10101001 00111001 00011101

Checksum

- Example (at a receiver)

Received data : 10101001 00111001 00011101

10101001

00111001

00011101

11111111 ← Sum

00000000 ← Complement

Checksum

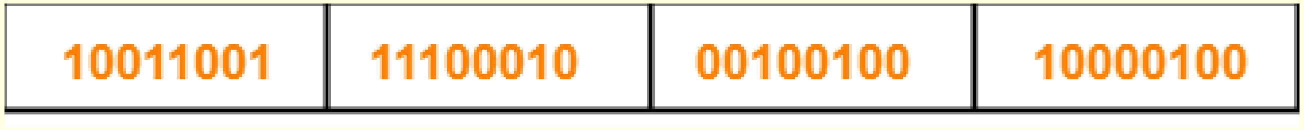
Ex 3

Consider the data unit to be transmitted is-

10011001 11100010 00100100 10000100

Consider 8 bit checksum is used.

- **<u>Step-01:</u>**
- At sender side,
- The given data unit is divided into segments of 8 bits as-



10011001	11100010	00100100	10000100

Checksum

Now, all the segments are added and the result is obtained as-
10011001 + 11100010 + 00100100 + 10000100 = 1000100011

- Since the result consists of 10 bits, so extra 2 bits are wrapped around.
- 00100011 + 10 = 00100101 (8 bits)
- Now, 1's complement is taken which is 11011010.
- Thus, checksum value = 11011010

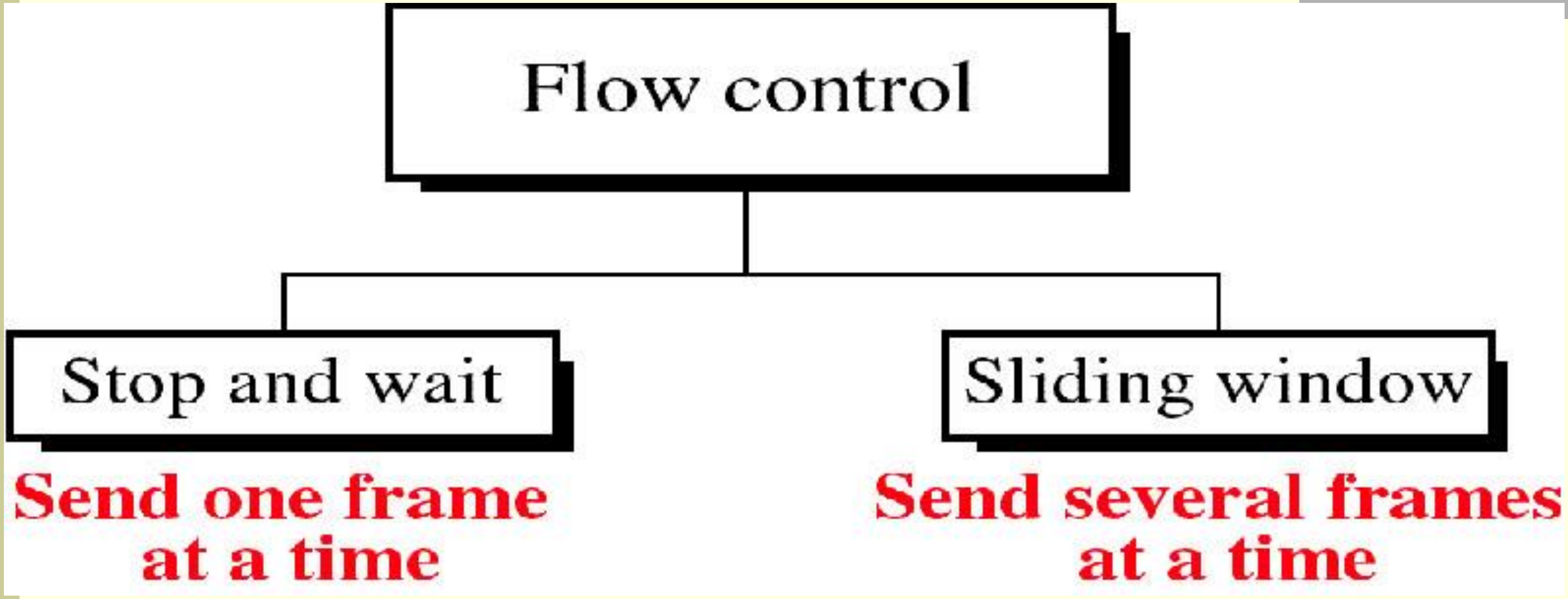
Performance

- The checksum detects all errors involving an odd number of bits.
- It detects most errors involving an even number of bits.
- If one or more bits of a segment are damaged and the corresponding bit or bits of opposite value in a second segment are also damaged, the sums of those columns will not change and the receiver will not detect a problem.

Flow Control

- Receiver has a limited speed at which it can process incoming data and a limited amount of memory in which to store incoming data.
- Receiver must inform the sender before the limits are reached and request that the transmitter to send fewer frames or stop temporarily.
- Flow control coordinates the amount of data that can be sent before receiving acknowledgement
- It is one of the most important functions of data link layer.
- Flow control is a set of procedures that tells the sender how much data it can transmit before it must wait for an acknowledgement from the receiver.

Flow Control



Stop & Wait Protocol

Types of Stop and Wait Protocol:

- An Unrestricted Simplex Protocol
- A Simplex Stop-and-Wait Protocol
- A Simplex Protocol for a Noisy Channel

1. An Unrestricted Simplex Protocol

- Data are transmitted in one direction only
- The transmitting (Tx) and receiving (Rx) hosts are always ready
- Infinite buffer space is available
- No errors occur; i.e. no damaged frames and no lost frames (perfect channel)
- The sender is in an infinite while loop just pumping data out onto the line as fast as it can.

Stop & Wait Protocol

- The body of the loop consists of three actions:

1 Go fetch a packet from the network layer,

2 Construct an outbound frame and

3. Send it on the way.

- Only the info field of the frame is used by this protocol, because the other fields have to do with error and flow control and there are no errors and flow control restrictions here.
- The receiver waits for something to happen, the only possibility being the arrival of an undamaged frame.
- It passes to the network layer and the data link layer settles back to wait for the next frame.

Stop & Wait Protocol

2. A simplex stop-and-wait protocol

- In this protocol we assume that data are transmitted in one direction only
- No errors occur (perfect channel)
- The receiver can only process the received information at a finite rate

These assumptions imply that the transmitter cannot send frames at a rate faster than the receiver can process them.

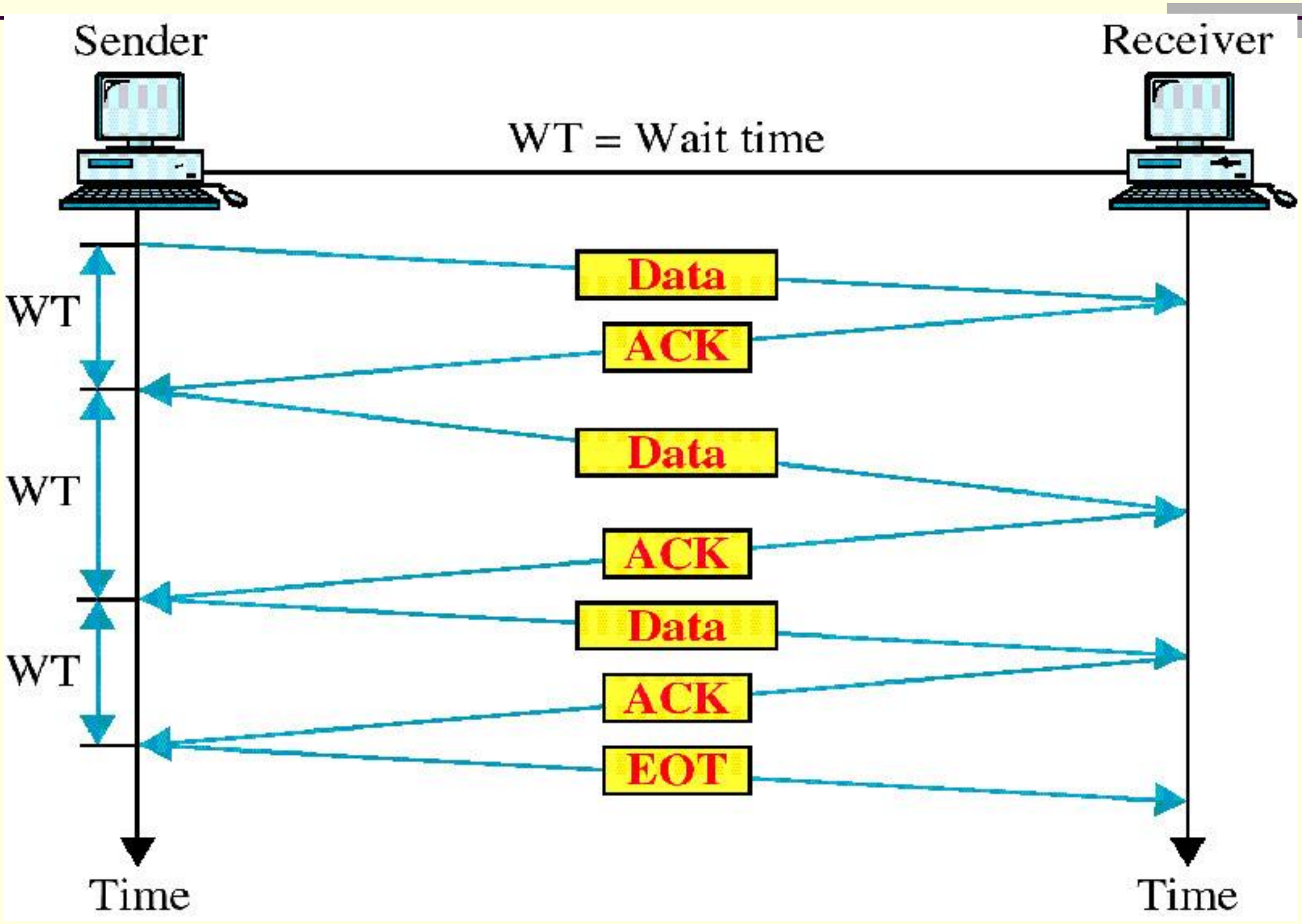
Stop & Wait Protocol

- The problem here is how to prevent the sender from flooding the receiver.
- A general solution to this problem is to have the receiver provide some sort of feedback to the sender.
- The process could be as follows:

The receiver send an acknowledge frame back to the sender telling the sender that the last received frame has been processed.

Permission to send the next frame is granted. The sender, after having sent a frame, must wait for the acknowledge frame from the receiver before sending another frame. **This protocol is known as stop-and-wait.**

Stop & Wait Protocol



Stop & Wait Protocol

3.A simplex protocol for a noisy channel

- In this protocol the unreal "error free" assumption in protocol 2 is dropped. Frames may be either damaged or lost completely.
- The sender would send a frame and the receiver would send an ACK frame only if the frame is received correctly. If the frame is in error the receiver simply ignores it; the transmitter would time out and would retransmit it.
- One fatal flaw with the above scheme is that if the ACK frame is lost or damaged, duplicate frames are accepted at the receiver without the receiver knowing it.

Stop & Wait Protocol

- Imagine a situation where the receiver has just sent an ACK frame back to the sender saying that it correctly received and already passed a frame to its host.
- However, the ACK frame gets lost completely, the sender times out and retransmits the frame.
- There is no way for the receiver to tell whether this frame is a retransmitted frame or a new frame, so the receiver accepts this duplicate happily and transfers it to the host. The protocol thus fails in this aspect.
- To overcome this problem it is required that the receiver be able to distinguish a frame that it is seeing for the first time from a retransmission.

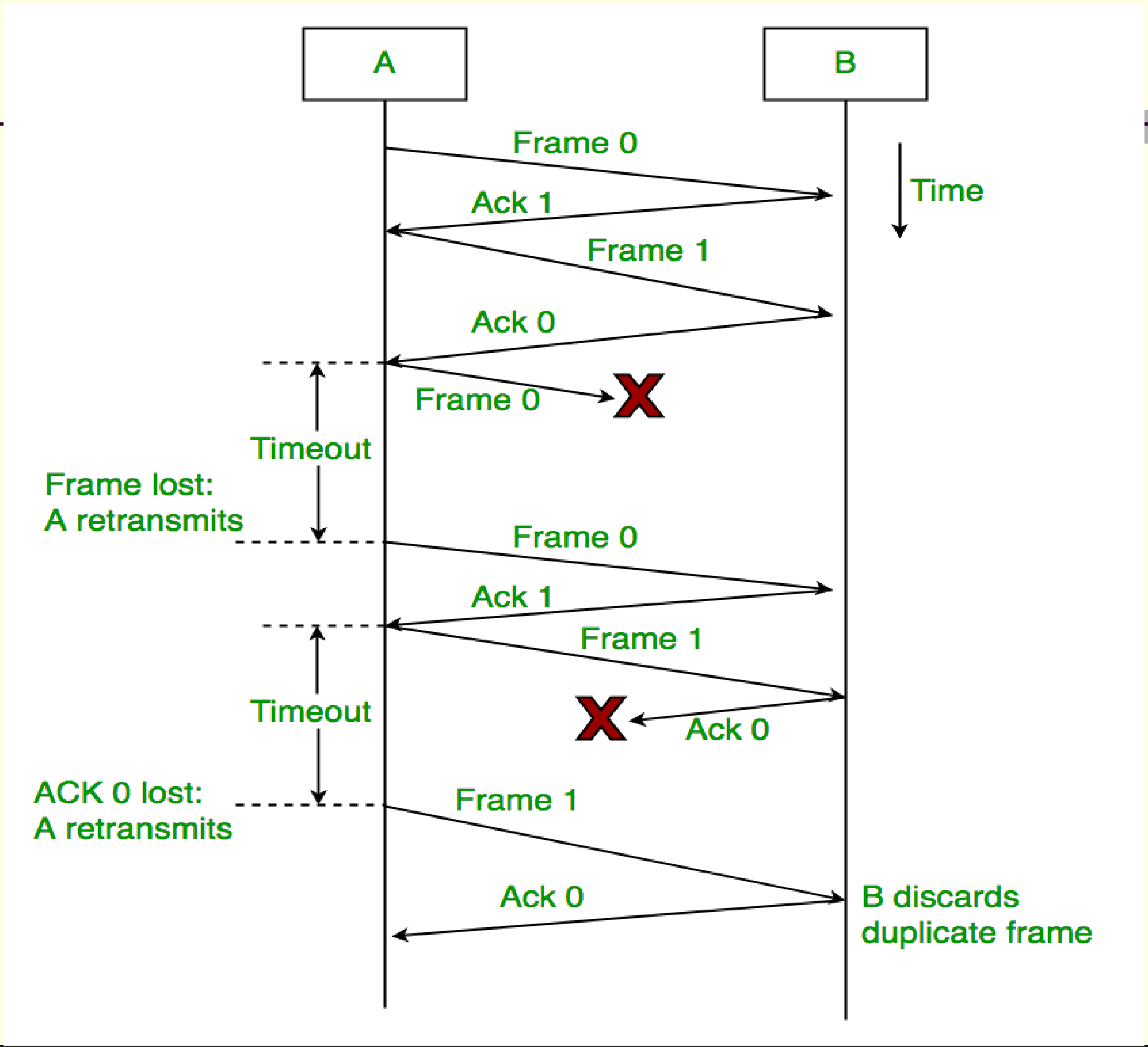
Stop & Wait Protocol

- One way to achieve this is to have the sender put a sequence number in the header of each frame it sends. The receiver then can check the sequence number of each arriving frame to see if it is a new frame or a duplicate to be discarded.
- The receiver needs to distinguish only 2 possibilities: a new frame or a duplicate; a 1-bit sequence number is sufficient. At any instant the receiver expects a particular sequence number. Any wrong sequence numbered frame arriving at the receiver is rejected as a duplicate. A correctly numbered frame arriving at the receiver is accepted, passed to the host, and the expected sequence number is incremented by 1

Stop & Wait Protocol

- 1) Sender A sends a data frame or packet with sequence number 0.
  2) Receiver B, after receiving data frame, sends and acknowledgement with sequence number 1 (sequence number of next expected data frame or packet)
  There is only one bit sequence number that implies that both sender and receiver have buffer for one frame or packet only.

Stop & Wait Protocol



Stop & Wait Protocol

Drawback of Stop and wait protocol(A Simplex Stop-and-Wait Protocol and A Simplex Protocol for a Noisy Channel)

1. Very Inefficient. At a moment , only one frame is in transition.
2. The sender has to wait at least one round trip time before sending the next frame.

Sliding window protocol

Assumptions:

1. Data is transmitted in both the directions.
2. Requires full duplex communication channel.
3. No separate ACK sent.
4. Uses the concept of piggybacking.(The technique of temporarily delaying the outgoing ACK so they can be hooked onto the next outgoing data frame is known as piggybacking).
5. Noisy channel.

Sliding window protocol

3 types of sliding window protocol

1.One-bit sliding window protocol

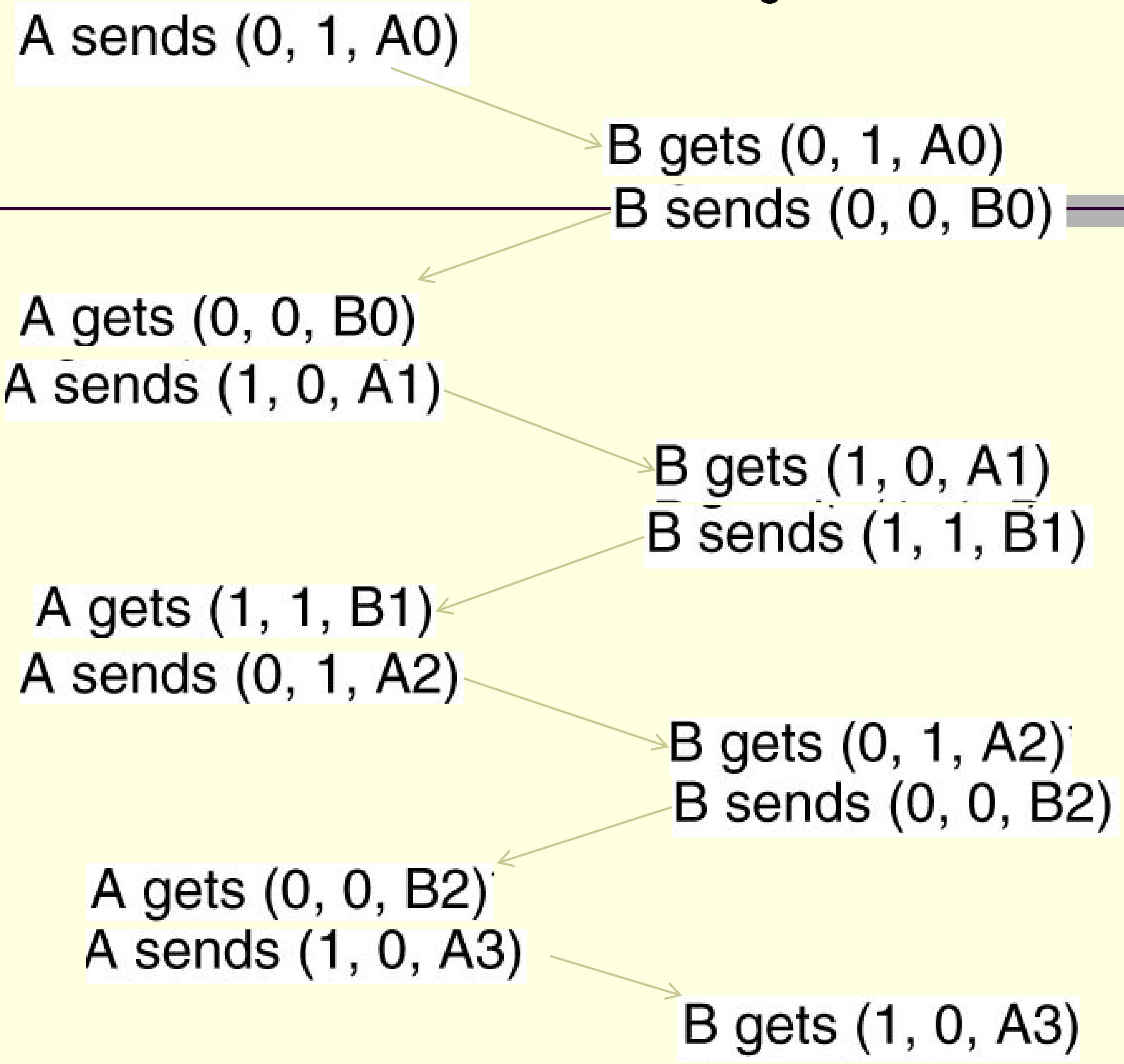
2.Go-back n sliding window protocol

3.Selective repeat sliding window protocol

One bit Sliding window

- Sender transmits one frame and waits for it's ACK, before sending the next frame
- Uses stop & wait .
- Window size is 1.
- Each end simultaneously acts as both sender and receiver .
- Sender sends frames as tuples.
- The ack no indicates the last frame received without error.

If one of them goes first

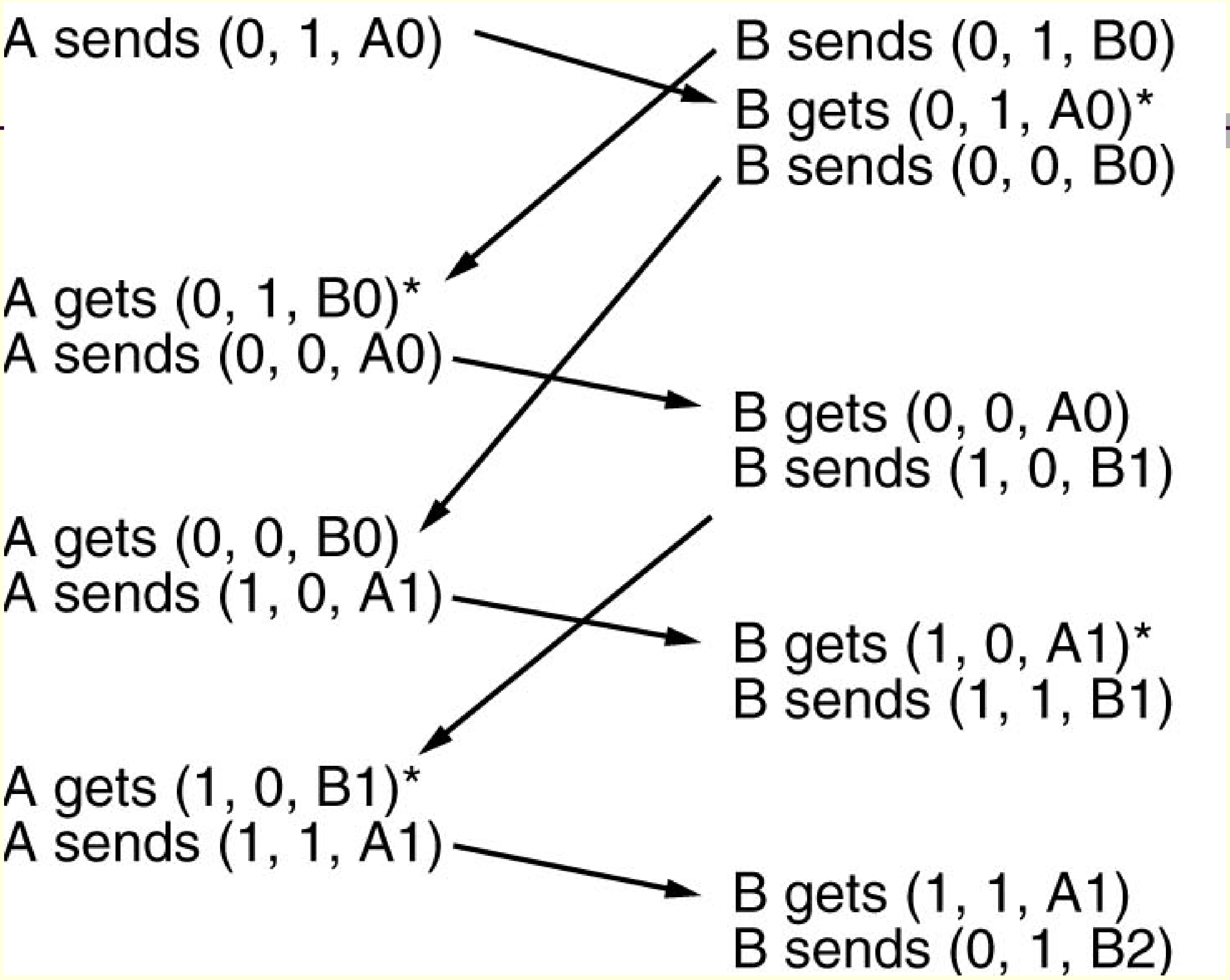


One bit Sliding window

If both send simultaneously

- A sends 0,1 B sends 0,1
  B gets 0,1 (got 0, but no ack of my 0)
  B **unnecessarily re-sends** 0,0
- A gets 0,1 (got 0, but no ack of my 0)
  A **unnecessarily re-sends** 0,0
- B gets 0,0 (already have 0, **nothing to pass to NB**, but this is good ack of my 0)
  B sends 1,0
- A gets 0,0 (already have 0, **nothing to pass to NA**, but this is good ack of my 0)
  A sends 1,0
- B gets 1,0 (got 1, but no ack of my 1) B **unnecessarily re-sends** 1,1
- A gets 1,0 (got 1, but no ack of my 1) A **unnecessarily re-sends** 1,1
- B gets 1,1 (already have 1, **nothing to pass to NB**, but this is good ack of my 1) B sends 0,1

- Problem if both send initial packet at same time.



Duplication of packets

One bit Sliding window

Drawback:

- Inefficient- Sender waits until an ACK arrives from the receiver.
- Sends only one frame at a time

Sliding Window

- **Goal**- Keep transmission medium busy.
- **Proposed scheme**

  Permit the sender to send more than one frame while waiting for the first ACK. This technique is called as pipelining.
- Sender maintains a **sending window**.
- Receiver maintains a **receiving window**.
- Sending window – Sequence nos of frames that have been sent but not yet acknowledged.
- Receiving window – Sequence nos of frames the receiver can accept
- Eg: 3-bit sequence no- frames are numbered from 0-7. If the sender has more than 8 frames--→0 1 2 3 4 5 6 7 0 1 2 3 4 5 6 7 0 1 2……

Sliding Window

- Max window size- 2^n (n- no of bits in the sequence no)
- Eg: $\boxed{0\ 1\ 2\ 3\ 4\ 5\ 6\ 7}$ 0 1 2 3….

  Sending window
- i.e sender can send 2^n frames without waiting for an ACK.
- When ACK for a frame comes in – Lower edge of the window is advanced by 1
- When a frame is sent upper edge of the window is advanced by 1.

Issue for a noisy channel

- Frame in the middle of a long stream is damaged/lost.
- What should the receiver do with all the correct frames following the damaged one ?
- **Constraint** : Receiving Data link layer has to hand packets to the network layer in sequence

Sliding Window

Two approaches to deal with errors in the presence of pipelining :

1. Go back n
2. Selective Repeat

Go back n:

- Discard all subsequent frames following the damaged frames sending no ACKs.
- Eventually the sender times out and retransmits all the unacknowledged frames in order starting with the damaged or lost one.

Operation :

Assume no of bits in sequence no 2 .

Window size : 4 frames

Sliding Window(Go back n)

- Sending window

$\boxed{0\ 1\ 2\ 3}$ 0 1 2 3……

- Sender has sent 4 frames 0 to 3 and waits for an ACK.
- ACK0 and ACK1 received by the sender

$\boxed{2\ \ 3\ \ 0\ 1}$ 2 3 0 1

Now sent 0 1

- Frame 2 lost not received but received 3 0 1
- Receiver discards 3 0 and 1

Note

A receiving station does not acknowledge each received frame explicitly . If a sending station received an ack for frame j and later receives an ack for frame k it assumes all frames between j and k have been received correctly.

Sliding Window(Go back n)

- Adv: Reduces no. of acks and lessens the n/w traffic.
- What should be the max sender's and receivers window size ?
- Assume k ---→ no. of bits in sequence no.
- Frames are numbered from 0 to 2^k-1
- Window size cannot be larger than 2^k

Case1 : Window size larger than 2^k

- Let k=3
- Assume window size =9
- Sender's window---→0 1 2 3 4 5 6 7 0 1 2 3……
- Problem --→

Sender receives an ACK0 . Does not know if it was for first/last frame.

Sliding Window(Go back n)

Window size must be less than or equal to 2^k

Case 2 : Window size = 2^k

1. At time t1 A sends frames 0-7 to B.
2. B receives each one correctly .
3. At time t2, B sends ACK for the most recent received frame ACK7.
4. **ACK gets lost**
5. Next frame expected by B is frame numbered 0.
6. A does not receive ACK and hence retransmits frames 0 through 7 at time t3.
7. At t4 B receives frame 0. Sequence no matches with the one it is expecting. Hence B accepts it.

P**rotocol fails(Since B has accepted a duplicate and not a new frame)**

Sliding Window(Go back n)

Reason

- 2 consecutive windows contain the same sequence no.
- Solution reduce the sender's window size by 1

Case 3 : Window size less than 2^k

Sender's window -→ $\boxed{0\ 1\ 2\ 3\ 4\ 5}$ 6 7 0 1 2 3 4….

1. A sends frames 0 through 6 at t1.
2. B receives each one correctly .
3. At time t2, B sends ACK for the most recent received frame ACK6.
4. **ACK gets lost**
5. Next frame expected by B is frame numbered 7.
6. A does not receive ACK and hence retransmits frames 0 through 6 at time t3.

Sliding Window(Go back n)

7. B receives frame 0 through 6 at t4 .

8. B expects frame 7.

9.Hence ignores them.

10. Eventually B sends another ACK6 which A receives and A advances it's window to include frame 7 0 1 2 3 4 5 and the protocol continues

Summary

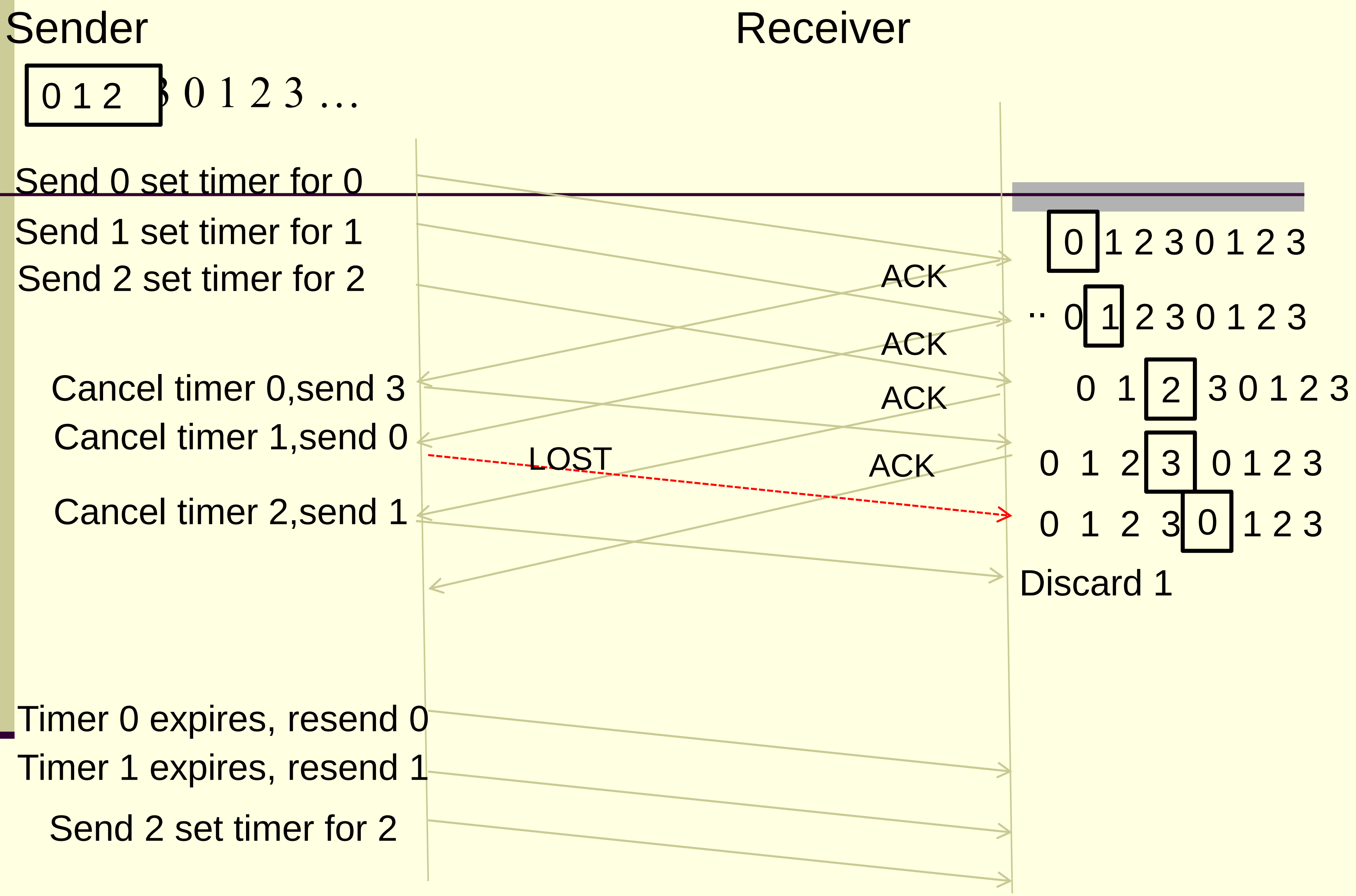
- **Sender's window size – strictly less than 2^k**

(1 less than MAXSEQ)

- What should be the receiver's window size?

Frames are always received in order.

Hence Receiver's window size should not be grater than 1


Sender
Receiver
0 1 2 3 0 1 2 3 …
Send 0 set timer for 0
Send 1 set timer for 1
Send 2 set timer for 2
Cancel timer 0,send 3
Cancel timer 1,send 0
Cancel timer 2,send 1
Timer 0 expires, resend 0
Timer 1 expires, resend 1
Send 2 set timer for 2
ACK
ACK
ACK
ACK
LOST
0 1 2 3 0 1 2 3
·· 0 1 2 3 0 1 2 3
0 1 2 3 0 1 2 3
0 1 2 3 0 1 2 3
0 1 2 3 0 1 2 3
Discard 1

11.5 Go-Back-N

- **Setup –Receiver sliding window**
- The size of the receiver window is always **1** and points to the **next expected frame** number to arrive
- This means that frames should arrive **in order**
- If the expected frame is received <u>without errors</u>, the receiver window slides over the **next sequence number**.
- **Operation**
- The receiver sends a positive ACK if a frame has arrived **without error** and **in order** (with the expected sequence number)
- Receiver <u>does not have</u> to acknowledge each individual frame received correctly and in order.
- Receiver can send **cumulative ACK** for several frames (**ACK 4 acknowledges frames (0,1,2,3,4))**
- If the frame is <u>**damaged or out-of-order**</u>, the receiver **discards it** (and **stay *silent***) and also **discards all subsequent frames** until it receives the one expected.
  - In this case, **no** ACK will be transmitted
- If the sender <u>timer expires</u> before receiving an ACK, it will **resend *ALL* frames beginning with the one expired until the last one sent** (Go-Back-N).

Selective repeat

Drawback of Go back n

- Receiver discards all the correct frames transmitted after the bad one.
- Channel bandwidth wasted on retransmitted frames.

Alternative strategy :

Allow the receiver to accept and buffer the frames following a damaged/lost one.

Called Selective repeat

- Since the receiving station is not required to receive frames in order, the receiving window size is greater than 1.
- Note: A frame arriving out of order can be received as long as it is in the receiving window.

Selective repeat

- If an arriving frame is in the receiving window it is buffered and not given to the n/w layer until all the predecessor have also arrived

Ex: Assume a 3-bit sequence no.

- Sender sends up to seven frames before waiting for an ACK.
- Sender's window $\boxed{0\ 1\ 2\ 3\ 4\ 5\ 6}$ 7 0 1 2 3 4….
- Receiver's window $\boxed{0\ 1\ 2\ 3\ 4\ 5\ 6}$ 7 0 1 2 3 4….

1. Sender transmits frames 0 through 6
2. All 7 frames are correctly received hence receiver sends ACK6
3. Receiver advances it's window to allow the receipt of next set of sequence nos. 0 1 2 3 4 5 6 $\boxed{7\ 0\ 1\ 2\ 3\ 4\ 5}$ 6 7 0….
4. ACK is lostSender times out and retransmits frame from 0 through 6

Selective repeat

6. Receiver receives frames 0-6 but accepts frames 0 1 2 3 4 5 since it lies within it's window.

   But frames 0-5 being duplicates should have been rejected by the receiving station.

   Hence the protocol fails.

PROBLEM

After the receiver advanced it's window, the new range of valid

sequence nos. overlapped with the old one.

Consequently the following batch of frames might be either duplicates

or new ones(if all ACKS were received).

The receiver has no way of distinguishing these 2 cases.

Selective repeat

For a 3 –bit sequence no sending and receiving window should be 4.

- Sender Receiver

$\boxed{0\ 1\ 2\ 3}$ 4 5 6 7 0 1 2 3 4…. $\boxed{0\ 1\ 2\ 3}$ 4 5 6 7 0 1 2 3 4….

$\longrightarrow$

$\boxed{0\ 1\ 2\ 3}$ 4 5 6 7 0 1 2 3 4…. 0 1 2 3 $\boxed{4\ 5\ 6\ 7}$ 0 1 2 3 4….

$\longleftarrow$

ACK 3

ACK 3 Lost

Sender times out and retransmits frames 0 through 3

$\boxed{0\ 1\ 2\ 3}$ 4 5 6 7 0 1 2 3 4…. 0 1 2 3 $\boxed{4\ 5\ 6\ 7}$ 0 1 2 3 4….

$\xrightarrow[\textbf{0 1 2 3}]{}$

Receiver discards frames 0 1 2 3 and sends ACK3

0 1 2 3 $\boxed{4\ 5\ 6\ 7}$ 0 1 2 3 4…. 0 1 2 3 $\boxed{4\ 5\ 6\ 7}$ 0 1 2 3 4….

Selective repeat

	One bit sliding window	Go back n	Selective repeat
Max Sender's Window size	1	$2^k -1$	2^{k-1}
Max receiver's window size	1	1	2^{k-1}