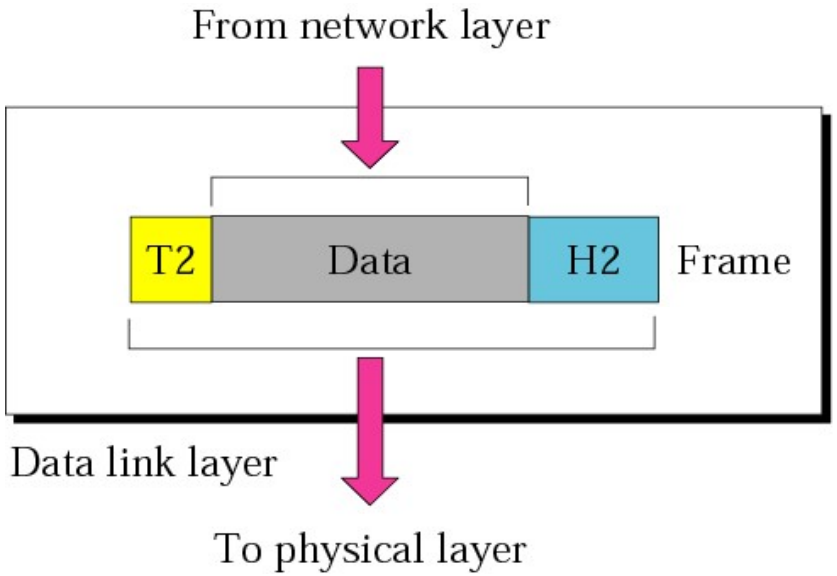
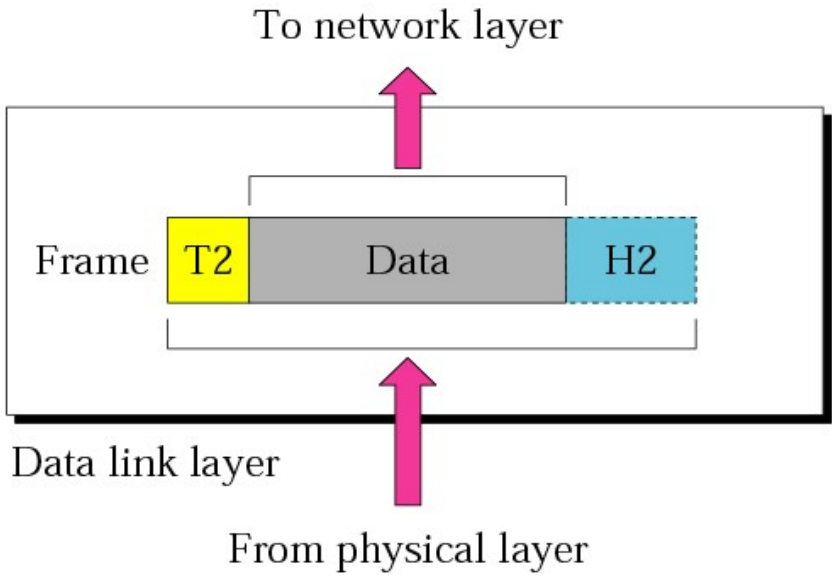
UNIT II

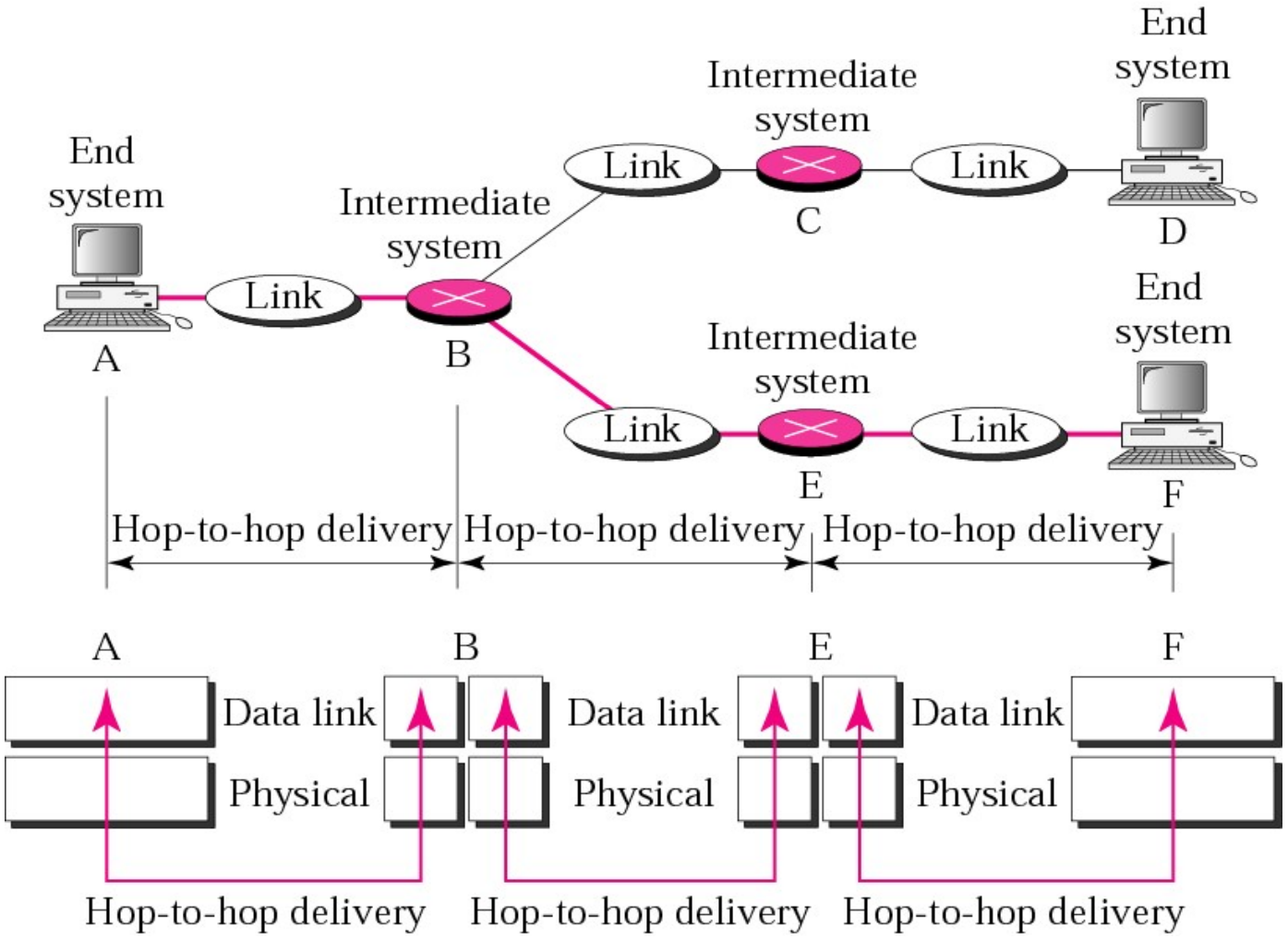
- **Data link layer**: Design issues in data link layer: framing, flow control, error control, Error Detection and Correction: Parity, CRC checksum, Hamming code, Flow Control: Sliding Window Protocols, Applications: Data link layer protocols HDLC, PPP.

Data Link layer cont.





Hop-to-Hop delivery



UNIT II

Data Link Layer

The **data link layer** transforms the physical layer, a raw transmission facility, to a reliable link.

• Responsible for node-to-node (hop-to-hop) communication.

•Specific responsibilities of the data link layer include ***framing, addressing, flow control, error control, and media access control***.

- *Framing: The data link layer divides the stream of bits received from the network* layer into manageable data units called **frames**.
- Addressing: Frames are to be distributed to different systems, the data link adds the header to the frame to define the receiver.

Flow Control: If the rate at which the data are absorbed by the receiver is less than the rate at which data are produced in the sender, the data link layer imposes a flow control mechanism to avoid overwhelming the receiver.

Error Control: The data link layer also adds reliability to the physical layer by adding mechanisms **to detect and retransmit damaged, duplicate, or lost frames**.

Access Control: When two or more devices are connected to the same link, data link layer protocols are necessary to determine which device has control over the link at any given time.

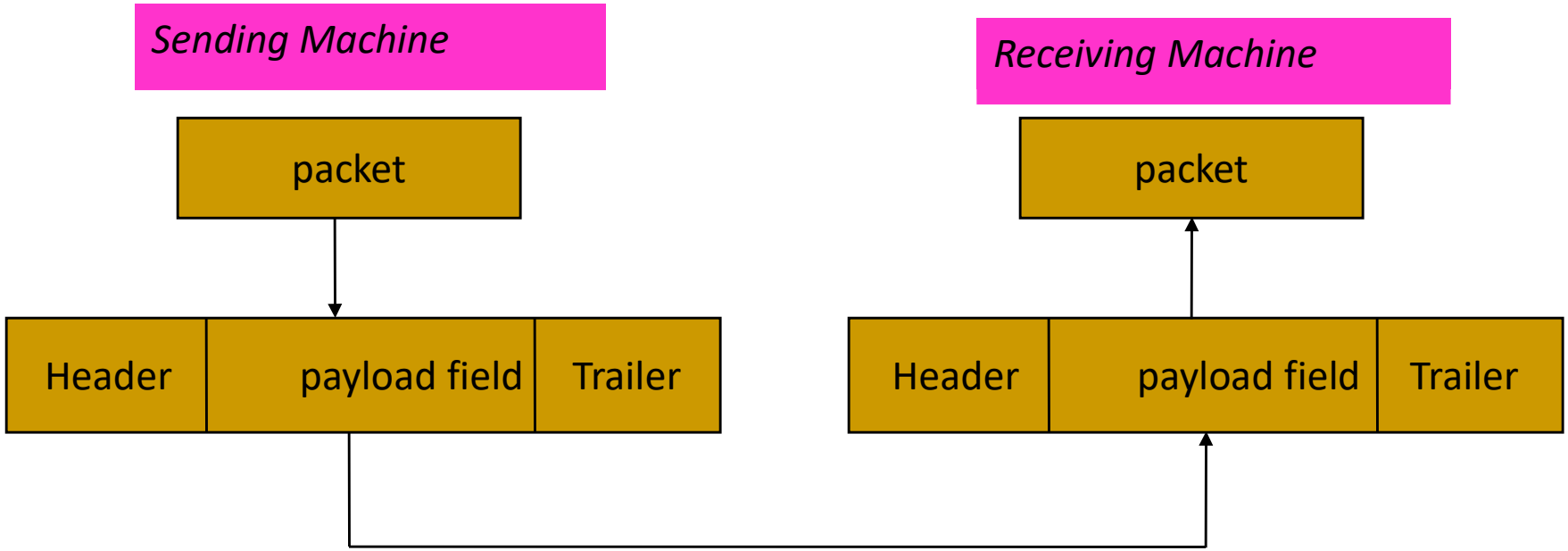
Data link layer:

Data-Link Layer *has responsibility of transferring frames from one node to adjacent node over a link*

- ***DATA LINK LAYER DESIGN ISSUES:***
  - *Data link layer has a number of specific functions to carry out. These functions include*
    - *Provides a well-defined service interface to the network layer.*
    - *Determines how the bits of the physical layer are grouped into frames (framing).*
    - *Deals with transmission errors*
    - *Regulates the flow of frames.*

Data link layer-Introduction

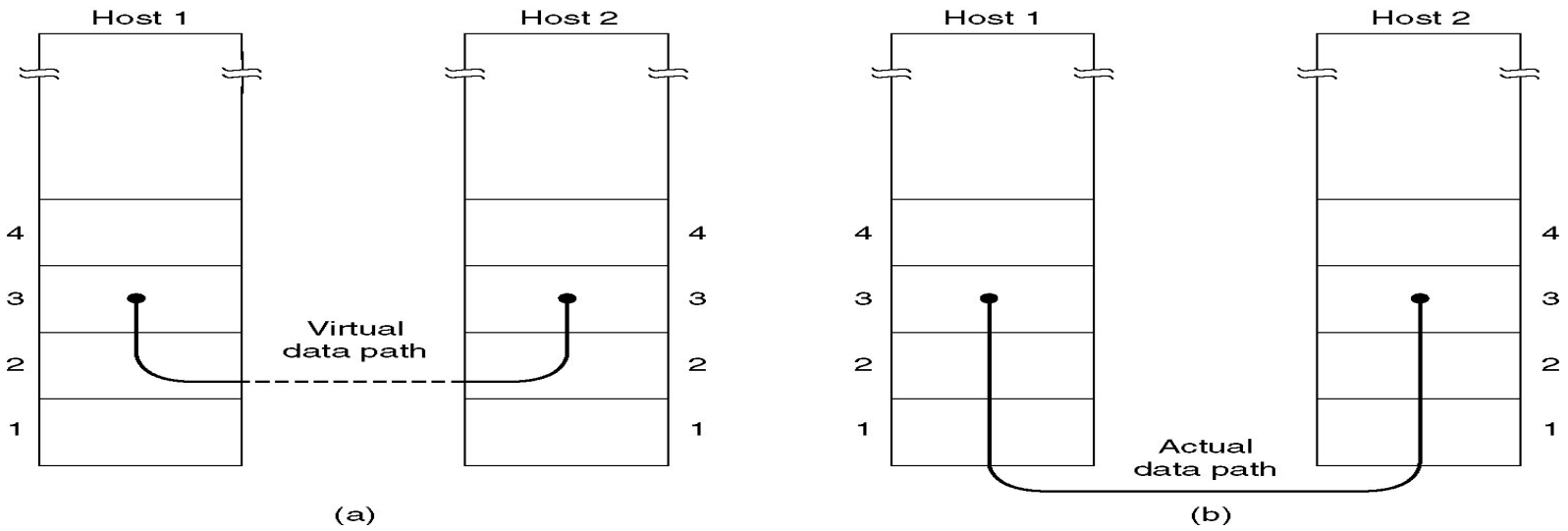
- *The data link layer takes the packets it gets from the network layer and encapsulates them into frames for transmission .*
- *Each frame contains frame header ,a payload field for holding the packet ,and a frame trailer*



Relationship between packets and frames

Data link layer-Introduction

- *Services Provided to the Network layer:*
  - *The function of the data link layer is to provide services to the network layer*
  - *The principal service is transferring data from the network layer on the source machine to the network layer on the destination machine.*



(a) Virtual *Communication* *(b)Actual communication*

Types of services provided to the Network Layer

- *Unacknowledged Connectionless service*
- *Acknowledged Connectionless service*
- *Acknowledged Connection-Oriented service*

Unacknowledged Connectionless service:

- *It consists of having the source machine send independent frames to the destination machine without having the destination acknowledge them.*
- *No connection is established beforehand or released afterward*
- *If a frame is lost due to noise on the line, no attempt is made to recover it in the data link layer.*
- *Appropriate for voice, where delay is worse than bad data.*
- *Most of the LANs use unacknowledged connection less service in the data link layer*

- *Acknowledged Connectionless service*
  - *when this service is offered, there are still no logical connections used, but each frame sent is individually acknowledged.*
  - *In this way ,the sender knows whether a frame has arrived correctly.*
  - *If it has not arrived within a specified time interval it can be sent again.*
  - *This service is used over unreliable channels ,such as wireless systems.*

Acknowledged Connection-Oriented service

- *The most sophisticated service the data link layer can provide to the network layer is connection-oriented service.*
- *The source and destination machines establish a connection before any data are transferred*
- *Each frame sent over the connection is numbered ,and the data link layer guarantees that each frame is received exactly once and that all frames are received in the right order*
- *When connection oriented service is used ,transfers go through three distinct phases.*
  - *1) Connection established*
  - *2) Data transferred*
  - *3) connection Released*

Framing:

- *In order to provide service to the network layer the data link layer must use the service provided to it by the physical layer.*
- *Physical layer is used to accept the raw bit stream and attempt to deliver it to the destination*
- *The bit stream is not guaranteed to be error free*
- *The number of bits received may be less than ,equal to ,or more than the number of bits transmitted, and they may have different values*
- *It is up to the data link layer to detect and if necessary ,correct errors.*
- *The usual approach is for the data link layer to break the bit stream up into discrete frames and compute the checksum for each frame.*

Framing:

- *When a frame arrives at the destination ,the checksum is recomputed.*
- *If the newly computed checksum is different from the one contained in the frame, the data link layer knows that an error has occurred and take steps to deal with it.*
  - *e.g., discarding the bad frame and possibly also sending back an error report*
- *Breaking the bit stream up into frames is more difficult than it at first appears, One way to achieve this framing is to insert time gaps between frames, much like the spaces between words in ordinary text.*

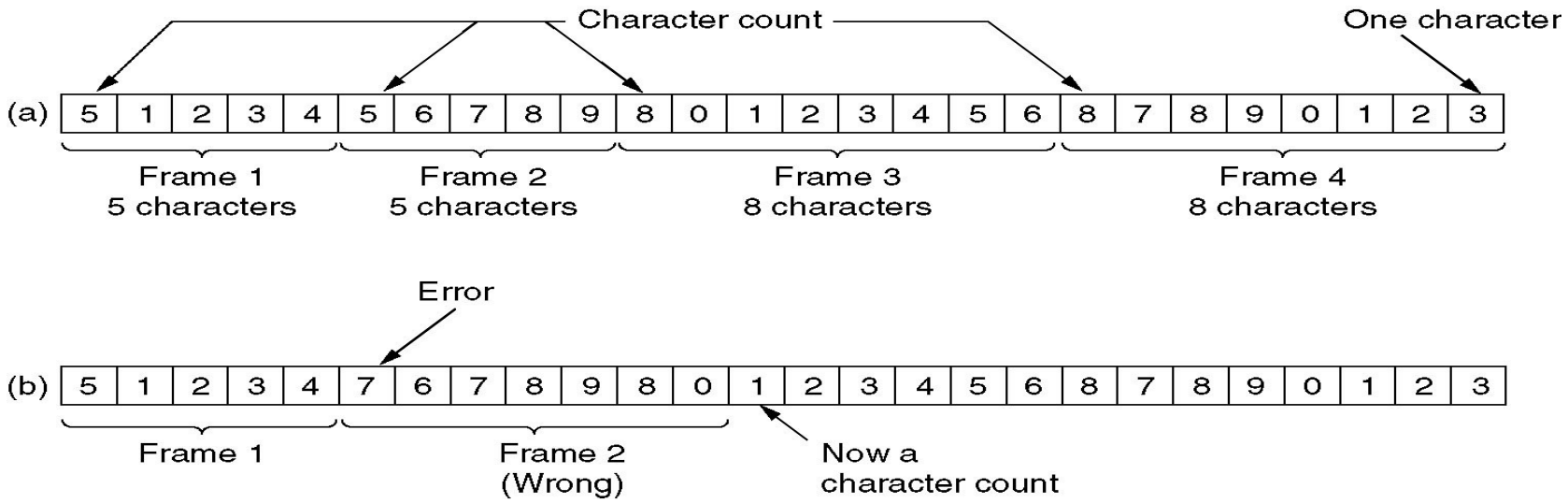
Methods:

1) character count

2) Flag bytes with byte stuffing

3) starting and ending flags, with bit stuffing

Character count:



A character stream a) with out errors b) with one error

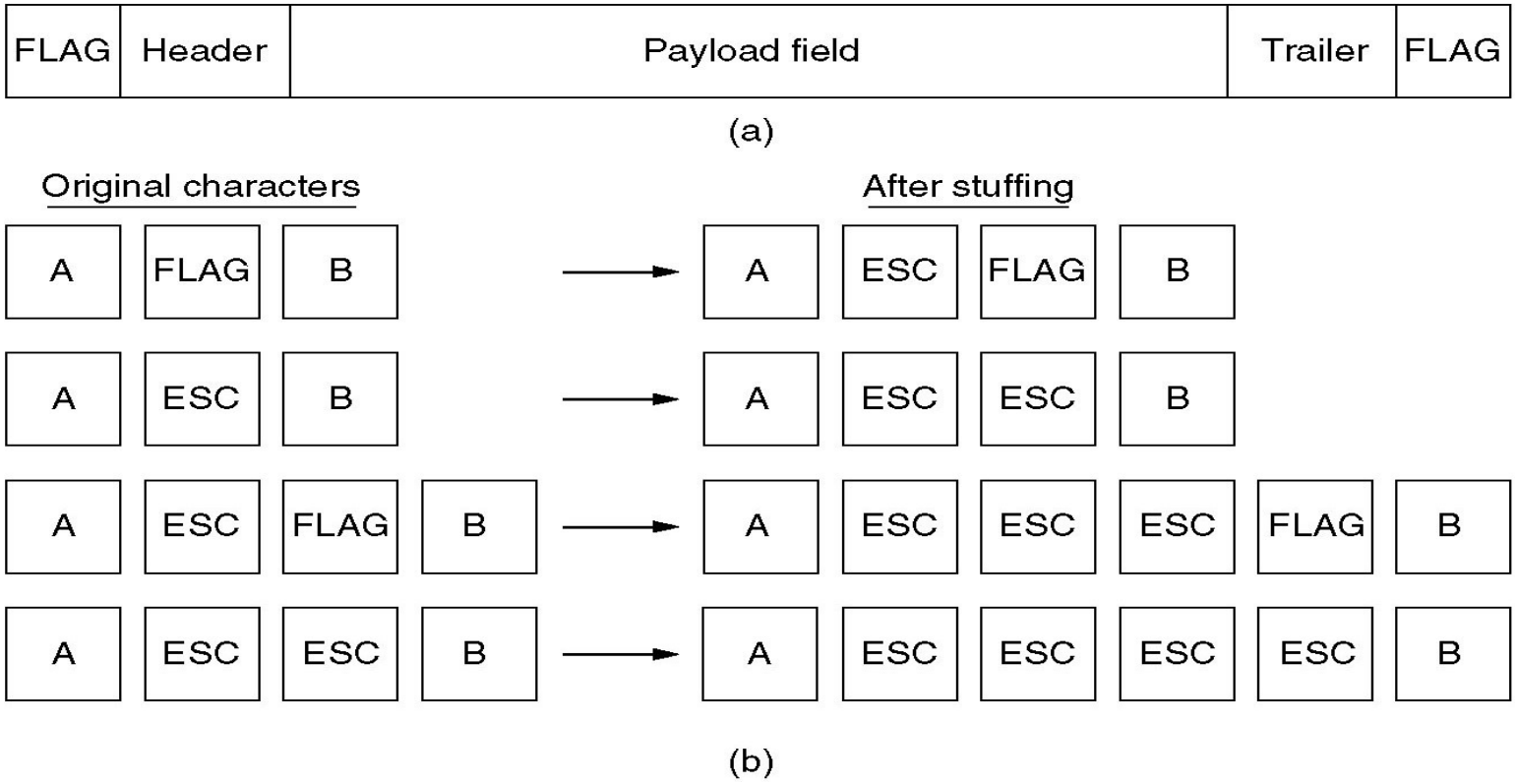
- *The first framing method uses a field in the header to specify the number of characters in the frame.*

- *When the data link layer in the destination sees the character count ,it knows how many characters follow and hence where the end of the frame is.*

Character count:

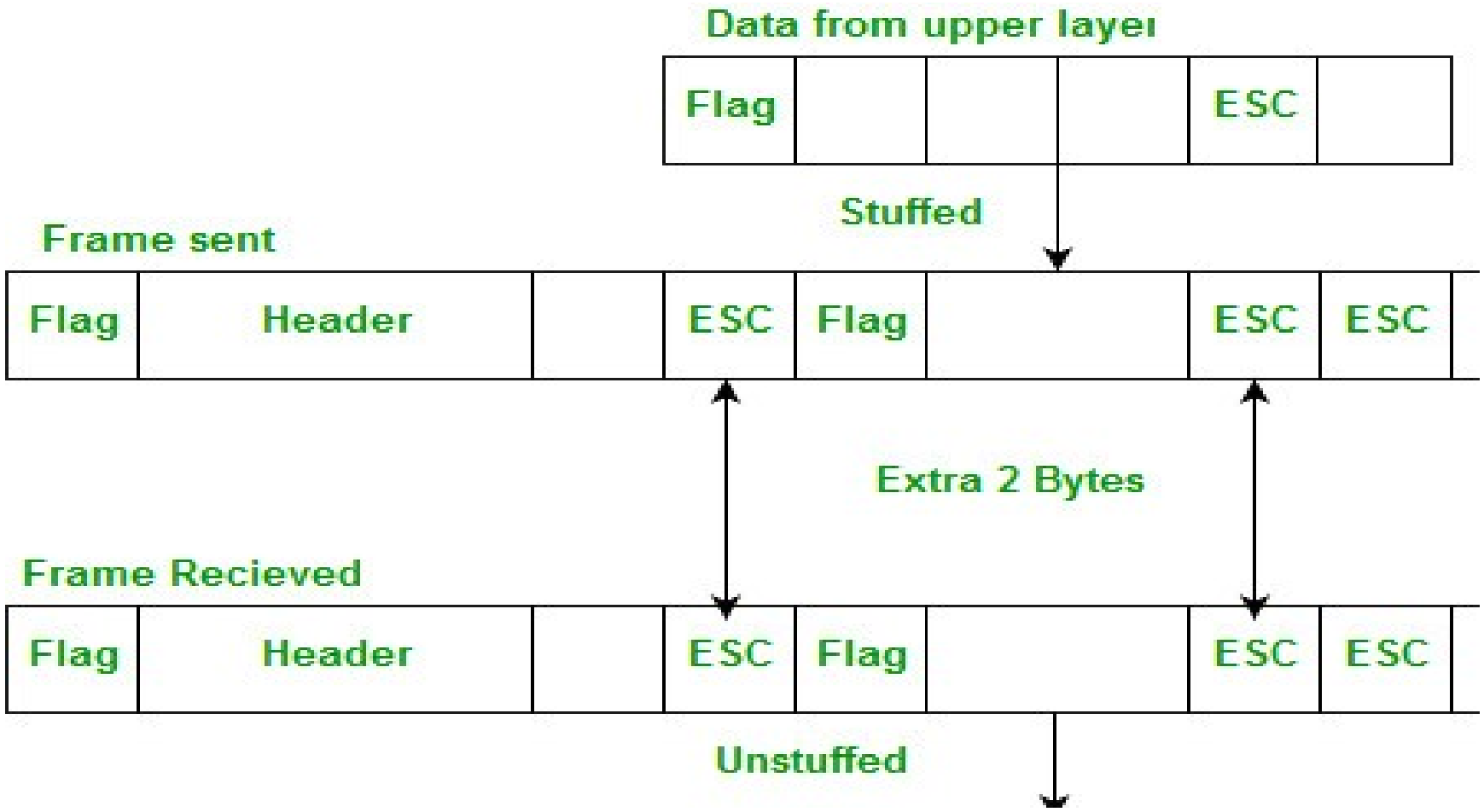
- *The trouble with this algorithm is that the count can be garbled by a transmission error.*
- *For example, if the character count of 5 in the second frame becomes a 7,the destination will get out of synchronization and will be unable to locate the start of the next frame.*
- *Disadvantages:*
  - *The count can be garbled by the transmission error*
  - *Resynchronization is not possible Even if with checksum, the receiver knows that the frame is bad there is no way to tell where the next frame starts.*
  - *Asking for retransmission doesn't help either because the start of the retransmitted frame is not known*
  - *No longer used*

Character Stuffing/Byte Stuffing



(a) A frame delimited by flag bytes.
(b) Four examples of byte sequences before and after stuffing.

Character Stuffing/Byte Stuffing

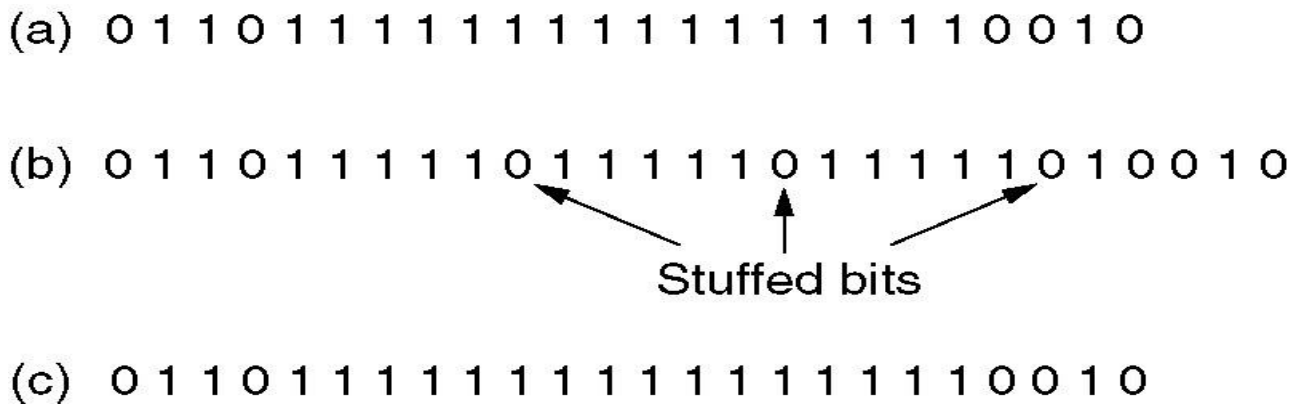


- *The second transmission method gets around the problem of Resynchronization called character stuffing.*
- *Most protocols used the same byte ,called flag byte, as both the starting and ending delimiter as FLAG.*
- *Two consecutive flag bytes indicate the end of one frame and start of the next one.*

Starting and ending flags, with bit stuffing

- *This new technique allows data frames to contain an arbitrary number of bits and allow character codes with an arbitrary number of bits per character.*
- *Each frame begins and ends with a special bit pattern,0111110 called a flag byte.*
- *whenever the senders data link layer encounters five consecutive ones in the data , it automatically stuffs a 0 bit into the outgoing bit stream.*
- *whenever the receiver sees five consecutive incoming 1 bits, followed by a 0 bit ,it automatically destuffs (i.e deletes) the 0 bit.*
- *If the flag pattern 01111110,this flag is transmitted as 011111010 but stored in the receiver memory as 01111110.*

starting and ending flags, with bit stuffing



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.

Error Control:

- *Next problem: How to make sure all frames are eventually delivered to the network layer at the destination, and in the proper order.*
- *The usual way to ensure reliable delivery is to provide the sender with some feedback about what is happening at the other end of the line.*
- *The protocol calls for the receiver to send back special control frames bearing positive and negative acknowledgements about the incoming frames*

Error Control:

- *If the sender receives a positive acknowledgement about a frame ,it knows the frame has arrived safely.*
- *on the other hand, a negative acknowledgement means that something has gone wrong, and the frame must be transmitted again.*
- *An additional complication comes from hardware troubles.*
- *Managing the Timers and sequence numbers so as to ensure that each frame is ultimately passed to the network layer at the destination exactly once.*

Error Control:

- How to ensure that the frames are delivered
  - Positive ACK
  - Negative ACK
- *Maintaining Timers for Error Control: When a sender transmits a frame, it generally also starts a timer .*
- *The timer is set to expire after an interval long enough for the frame to reach the destination, be processed there ,and have the acknowledgement to propagate back to the sender.*
- *Normally, the frame will be correctly received and the acknowledgement will get back before the timer runs out ,in which case the timer will be cancelled.*
- *However, if either the frame or the acknowledgement is lost ,the timer will go off ,alerting the sender to a potential problem .the obvious solution is to just transmit the frame again.*
- *Sequence number is used to recognize the duplicate packet.*

Flow control:

- *A sender that systematically wants to transmit frames faster than the receiver can accept them..*
- *When the sender is running on a fast (or lightly loaded) computer and the receiver is running on a slow (or heavily loaded machine)*
- *The sender keeps pumping the frames out at a higher rate until the receiver is completely swamped.*
- *Even if the transmission is error free ,at a certain point the receiver will simply be unable to handle the frames as they arrive and will start to lose someone*

To Prevent above situation two approaches are used

- *Feedback- based flow control*
- *Rate-based flow control*

Feedback- based flow control ,the receiver sends back information to the sender giving it permission to send more data.

Rate-based flow control , the protocol has built- in mechanism that limits the rate at which senders may transmits data, without using feedback – based flow control

Error Detection and Correction methods

Error Detection and correction:

- *In some cases it is sufficient to detect an error and in some, it requires the errors to be corrected also.*

- *For e.g.*
  - *On a reliable medium : Error Detection is sufficient where the error rate is low and asking for retransmission after Error Detection would work efficiently*
  - *In contrast, on an unreliable medium : Retransmission after Error Detection may result in another error and still another and so on. Hence Error Correction is desirable.*

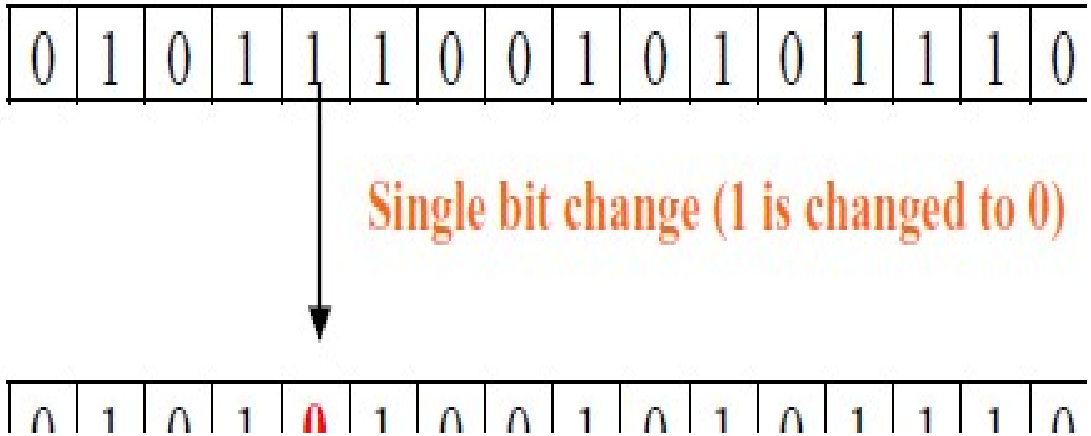
- *Data can be corrupted during transmission some applications require that error be detected and corrected.*
- *ERROR: whenever bits flow from one point to another, they are subject to unpredictable changes because of interference this interference can change the shape of the signal.*
- *Types of ERRORS:*

1) single bit error

2) burst error

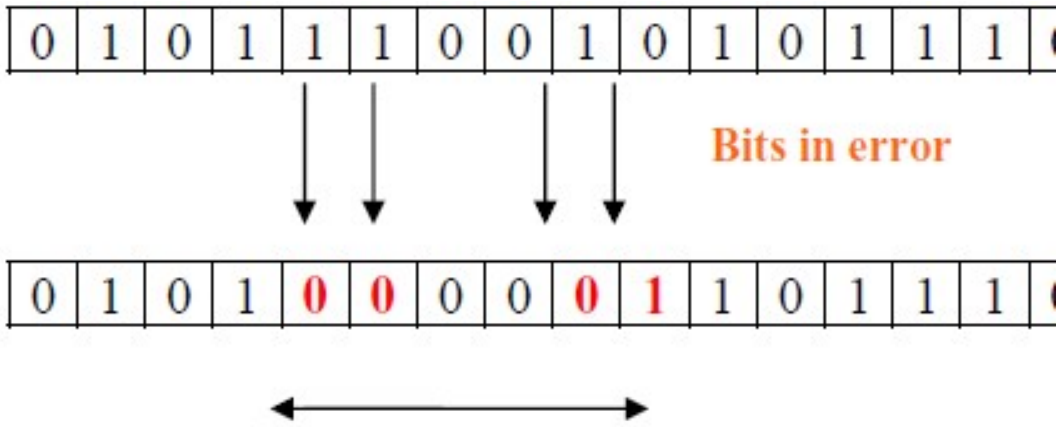
Single Bit Error

- *The term single-bit error means that only one bit of given data unit (such as a byte, character, or data unit) is changed from 1 to 0 or from 0 to 1.*



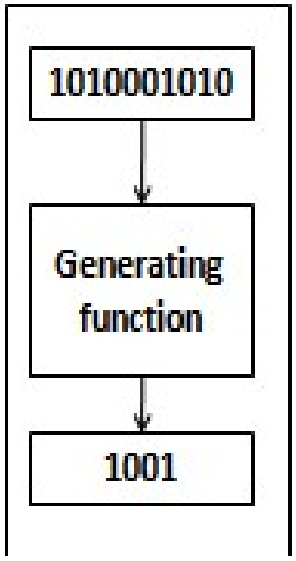
Burst Error

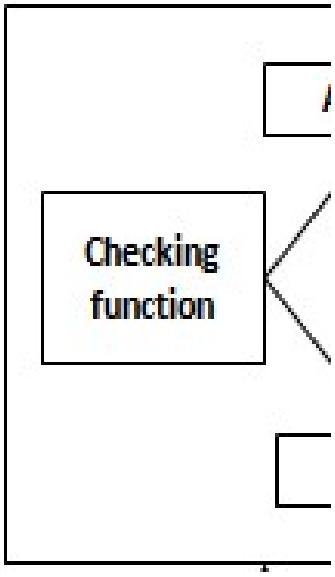
- *The term burst error means that two or more bits in the data unit have changed from 0 to 1 or vice-versa.*
- *Note that burst error doesn't necessary means that error occurs in consecutive bits.*
- *The length of the burst error is measured from the first corrupted bit to the last corrupted bit. Some bits in between may not be corrupted.*



Error Detection :

- *They are few codes which are used to detect errors, not to prevent the occurrence of error or not to correct the occurred error.*
- *Solution – a shorter group of data is appended to the end of the data unit . This technique is called* ***Redundancy****.*



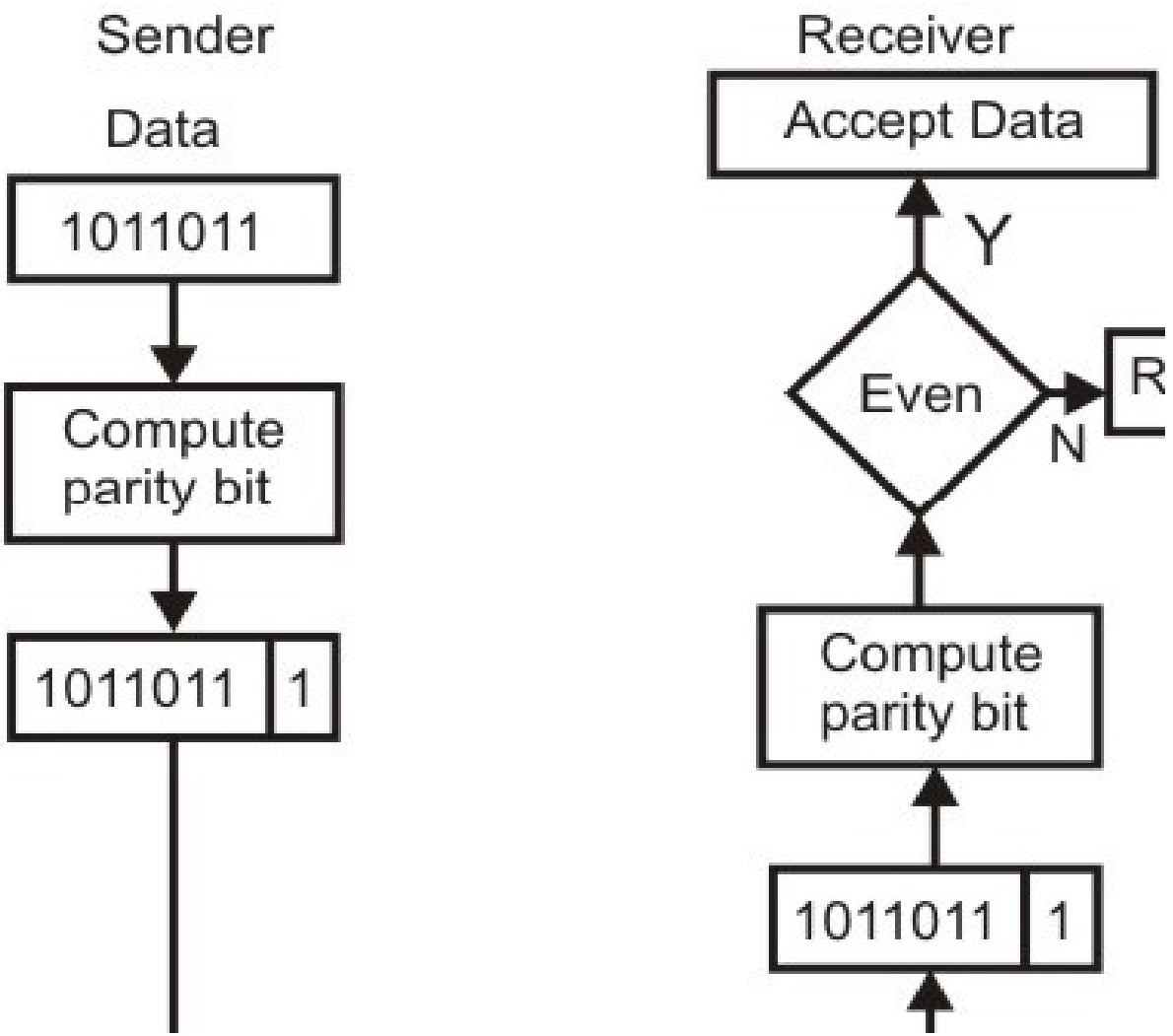


Error Detecting Codes

- *Basic approach used for error detection is the use of redundancy, where additional bits are added to facilitate detection and correction of errors. Popular techniques are:*
- *Simple Parity check*
- *Two-dimensional Parity check*
- *Checksum*
- *Cyclic redundancy check*

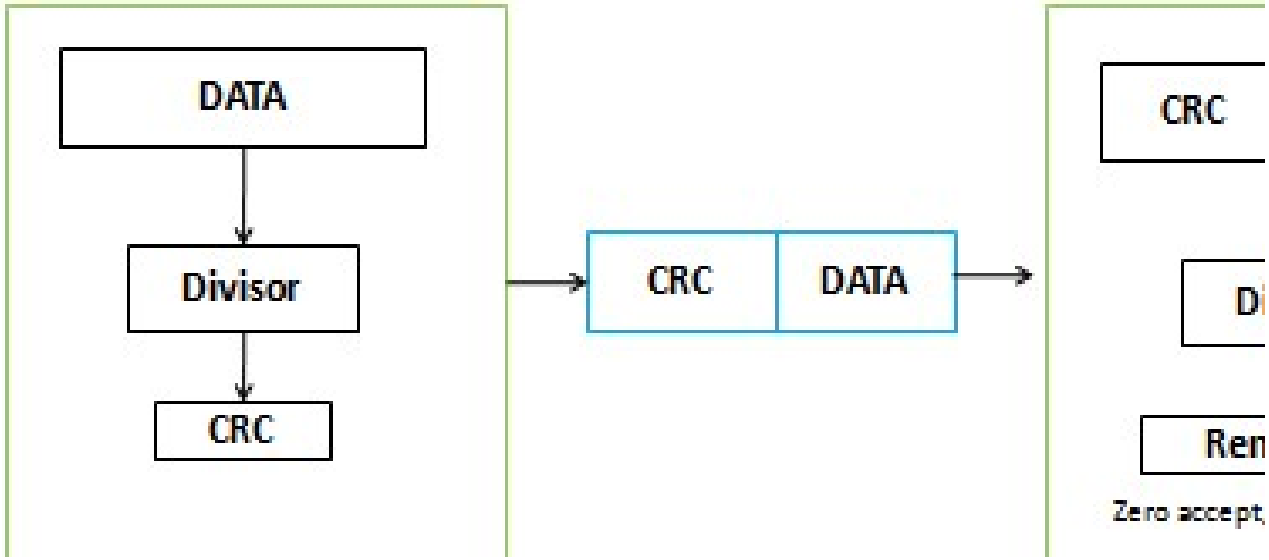
Simple Parity Checking/One-dimension Parity Check/Vertical Redundancy Check(Vrc):

- *.It is most common and least expensive mechanism for error detection.*
- *It is also called as **Parity check**.*
- *Append a single bit at the end of data block such that the number of ones is even*
  → Even Parity (odd parity is similar)
  0110011 → 01100110
  0110001 → 01100011
- ***Performance**: Detects all odd-number errors in a data block*


Sender
Data
1011011
Compute parity bit
1011011
1
Receiver
Accept Data
Y
Even
N
R
Compute parity bit
1011011
1

CRC(Cyclic Redundancy Check)

- *It is based on the binary division*
- *Polynomial code are used for generating check bits in the form of cyclic redundancy check.*

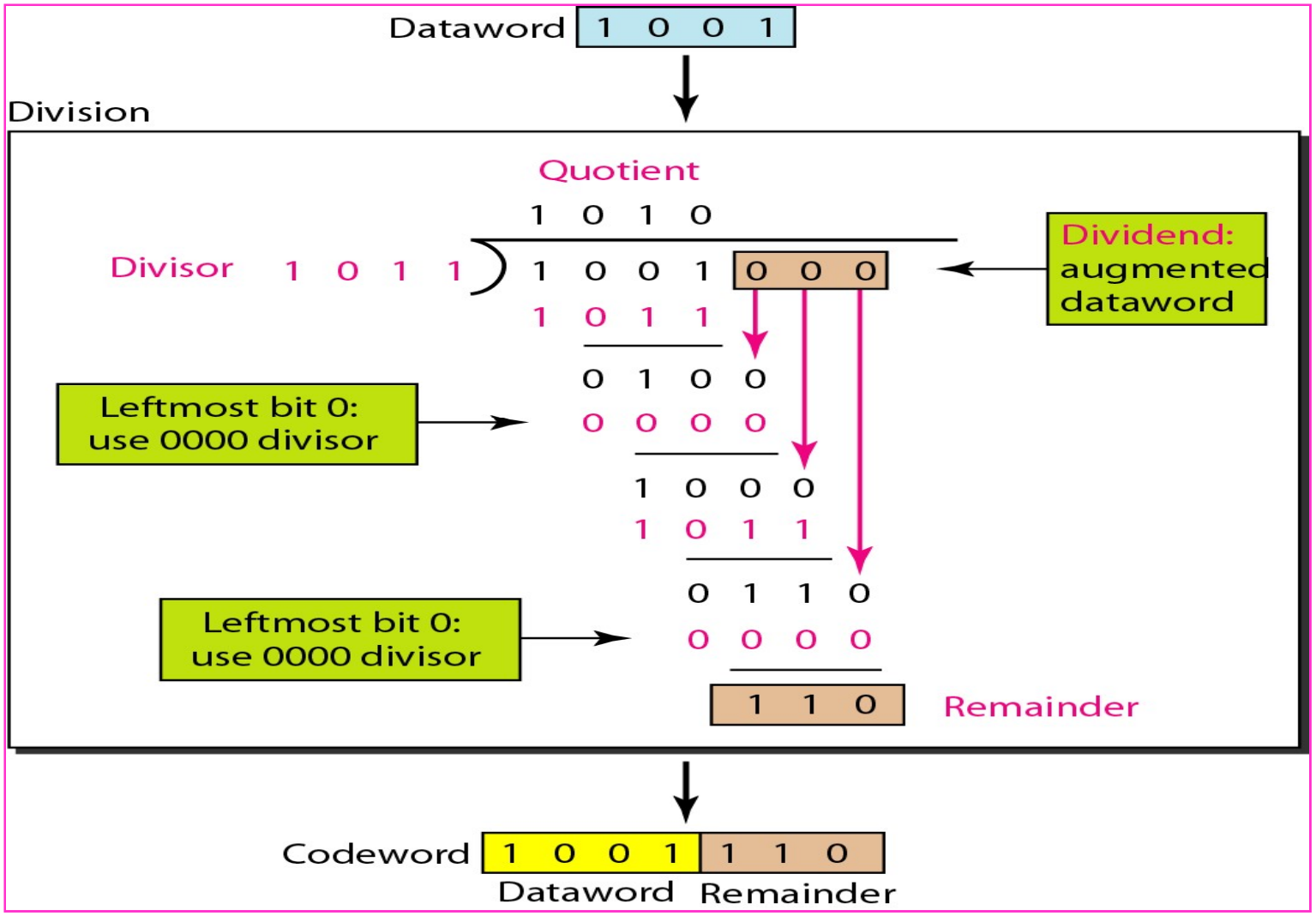


- *When the polynomial code method is employed ,the sender and receiver must agree upon a generator polynomial G(x) .*
- *In advance both higher and lower order bits of generator must be 1.*
- *To compute the check sum for some frame with m bits ,corresponding to the polynomial M(x), the frame must be longer than the generator polynomial.*
- *The idea is to append a checksum to the end of the frame in such a way that the polynomial represented by the check summed frame is divisible by G(x).*
- *When the receiver gets the check summed frame, it tries dividing it by G(x). If there is a remainder ,there has been a transmission error otherwise no error.*

Algorithm:

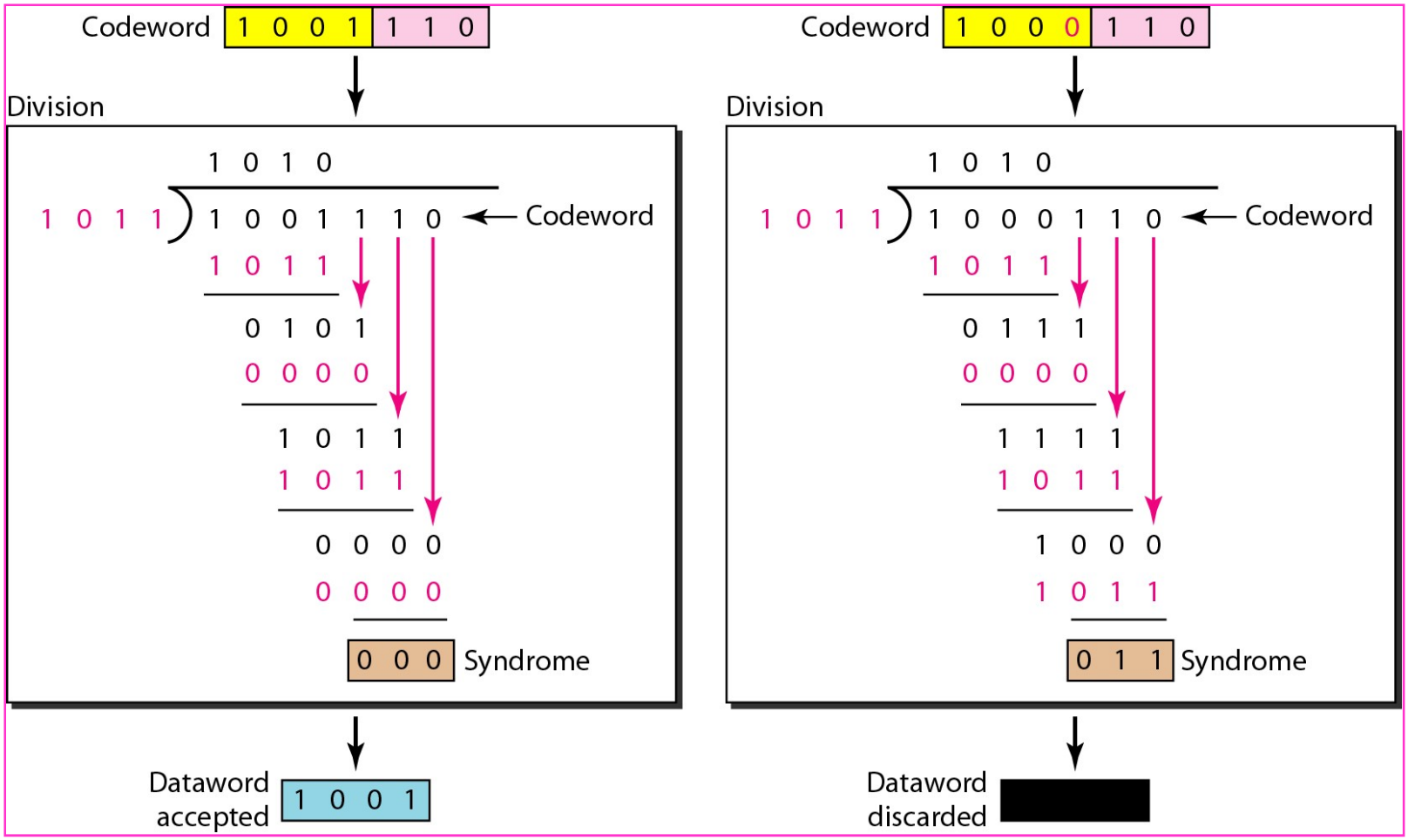
- *Let r be the degree of G(x)*
- *Append r zero bits to the low-order end of the frame ,so it now contains m+r bits and corresponds to the polynomial $x^r M(x)$.*
- *Divide the bit string corresponding to G(x) in to the bit string corresponding to $X^r M(x)$ using modulo 2 division.*
- *subtract the remainder (which is always r or fewer bits) from the bit string correspond to $x^r M(x)$ using modulo 2 subtraction.*
- *the result is the check summed frame to be transmitted .call its polynomial T(x).*

Calculation of the polynomial code checksum:



Frame: 1001 Generator : 1011

Division in the CRC decoder for two cases:



Performance of CRC:

- *CRC is a very effective error detection technique. If the divisor is chosen according to the previously mentioned rules, its performance can be summarized as follows:*
- *CRC can detect all single-bit errors*
- *CRC can detect all double-bit errors (three 1's)*
- *CRC can detect any odd number of errors (X+1)*
- *CRC can detect all burst errors of less than the degree of the polynomial.*
- *CRC detects most of the larger burst errors with a high probability.*
- *For example CRC-12 detects 99.97% of errors with a length 12 or more*

Error Correcting codes

- *The techniques that we have discussed so far can detect errors, but do not correct them.*

Error Correction can be handled in two ways.

- *One is when an error is discovered; the receiver can have the sender retransmit the entire data unit. This is known as backward error correction.*
- *In the other, receiver can use an error-correcting code, which automatically corrects certain errors. This is known as forward error correction.*

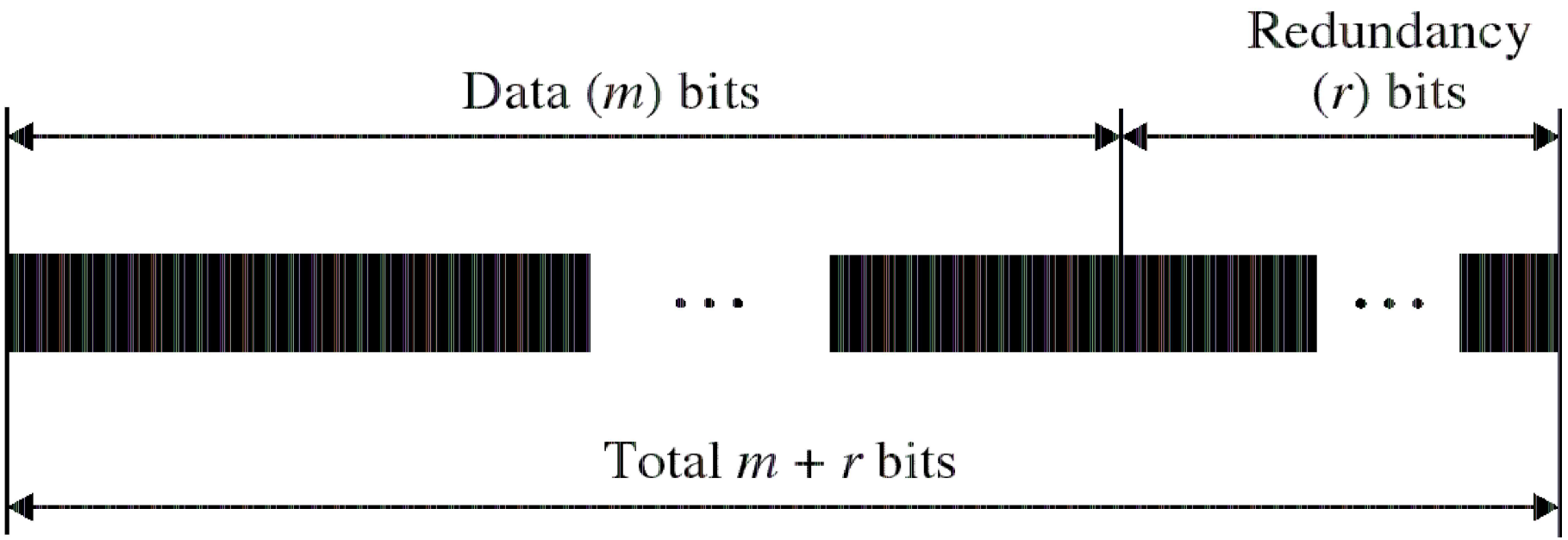
Error correcting codes:

- *Forward Error Correction: FEC is the only error correction scheme that actually detects and corrects transmission errors at the receive end with out calling for retransmission.*
  - *Ex: Hamming code*
- *No of bits in hamming code is dependent on the no. of bits in the data character by using the relation.*
  - $2^n >= m+n+1$
  - *Where n=no. of hamming bits*
  - *m= no. of bits in the data character*

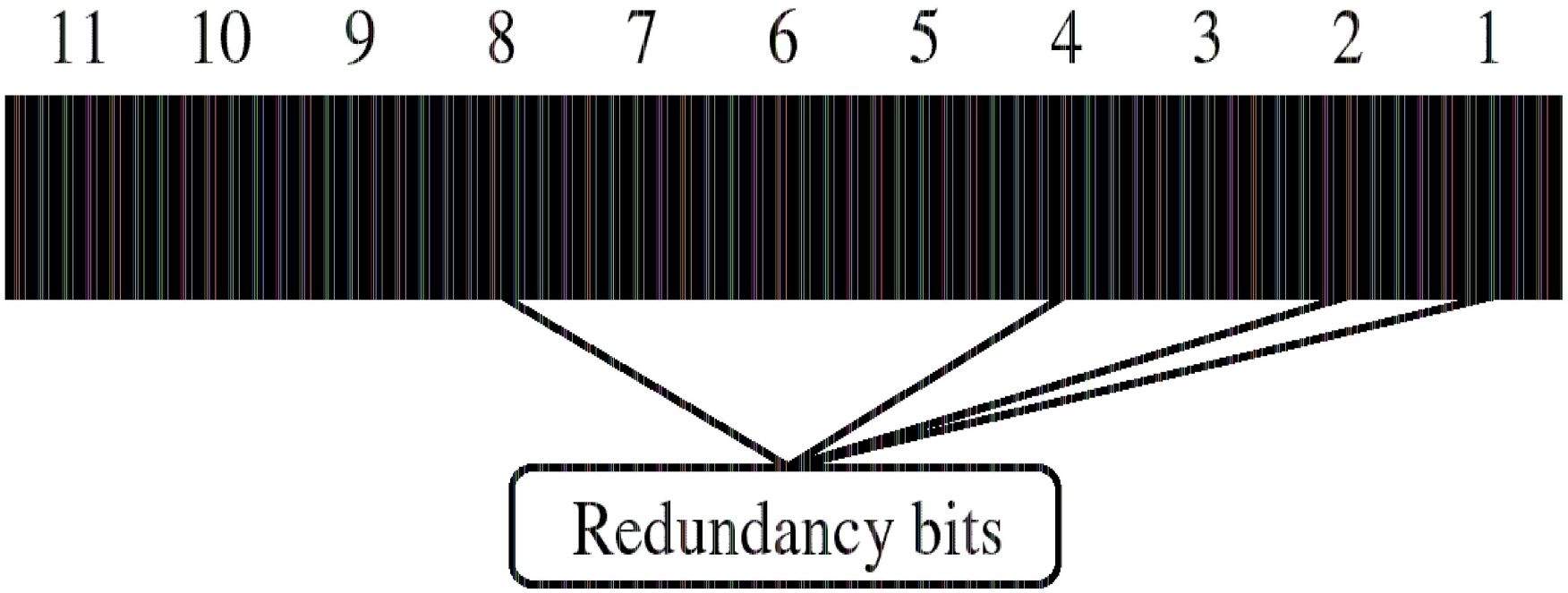
Error correcting codes:

- *No. of bits in hamming code*
- $2^n >= m+n+1$
  - *Take n=4*
  - $2^4 = 16 >= 12+4+1$
    - *16>=17*
  - *Take n=5*
    - $2^5 = 32 >= 12+5+1 = 18$
      - *32>=18 ---- 5 bits are required for hamming code.*
      - *12 bits of data +5 bits of code=17 bits of data stream.*

Error correcting codes:



Hamming Code

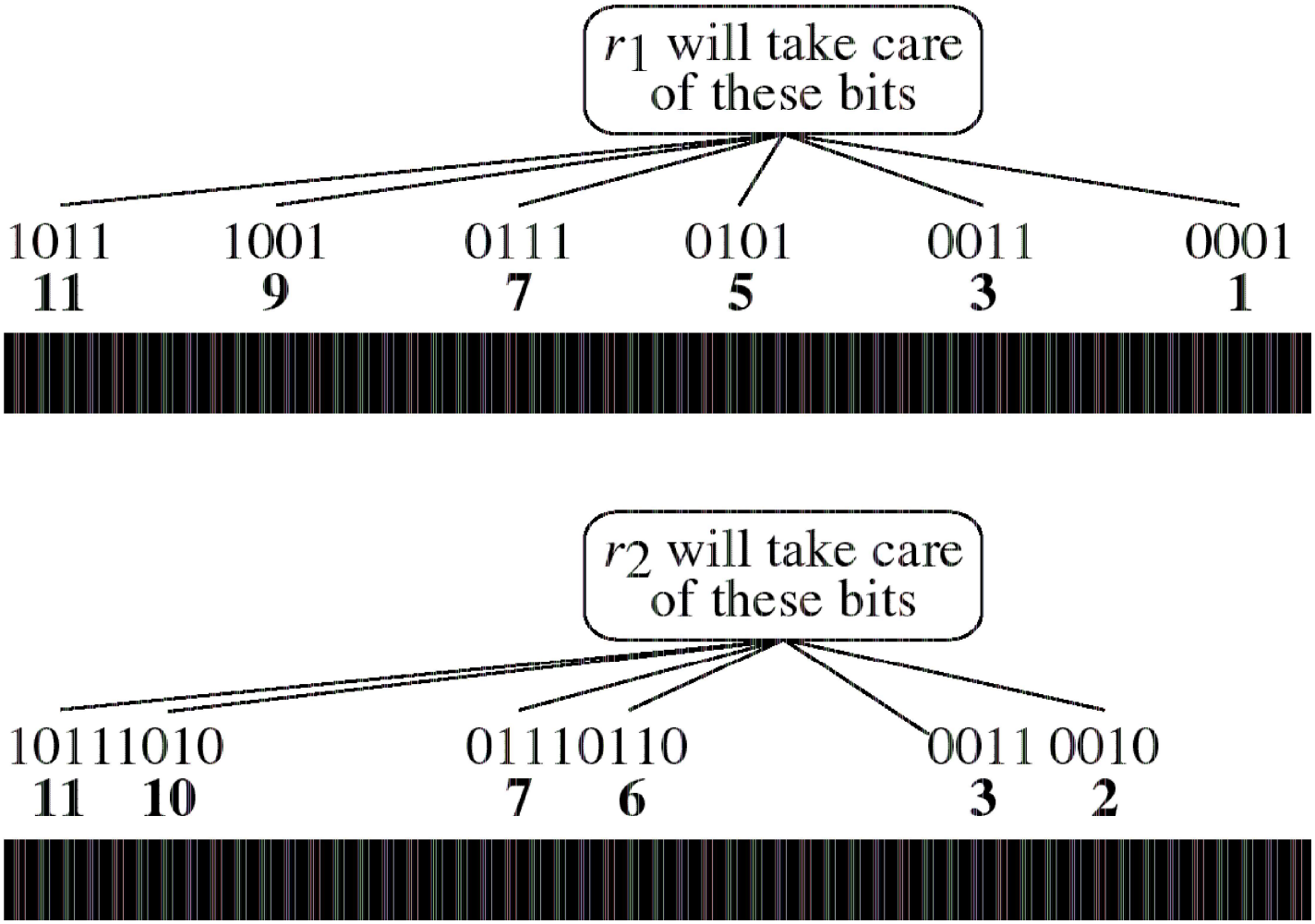


Calculation of redundancy bits for 7 bit data

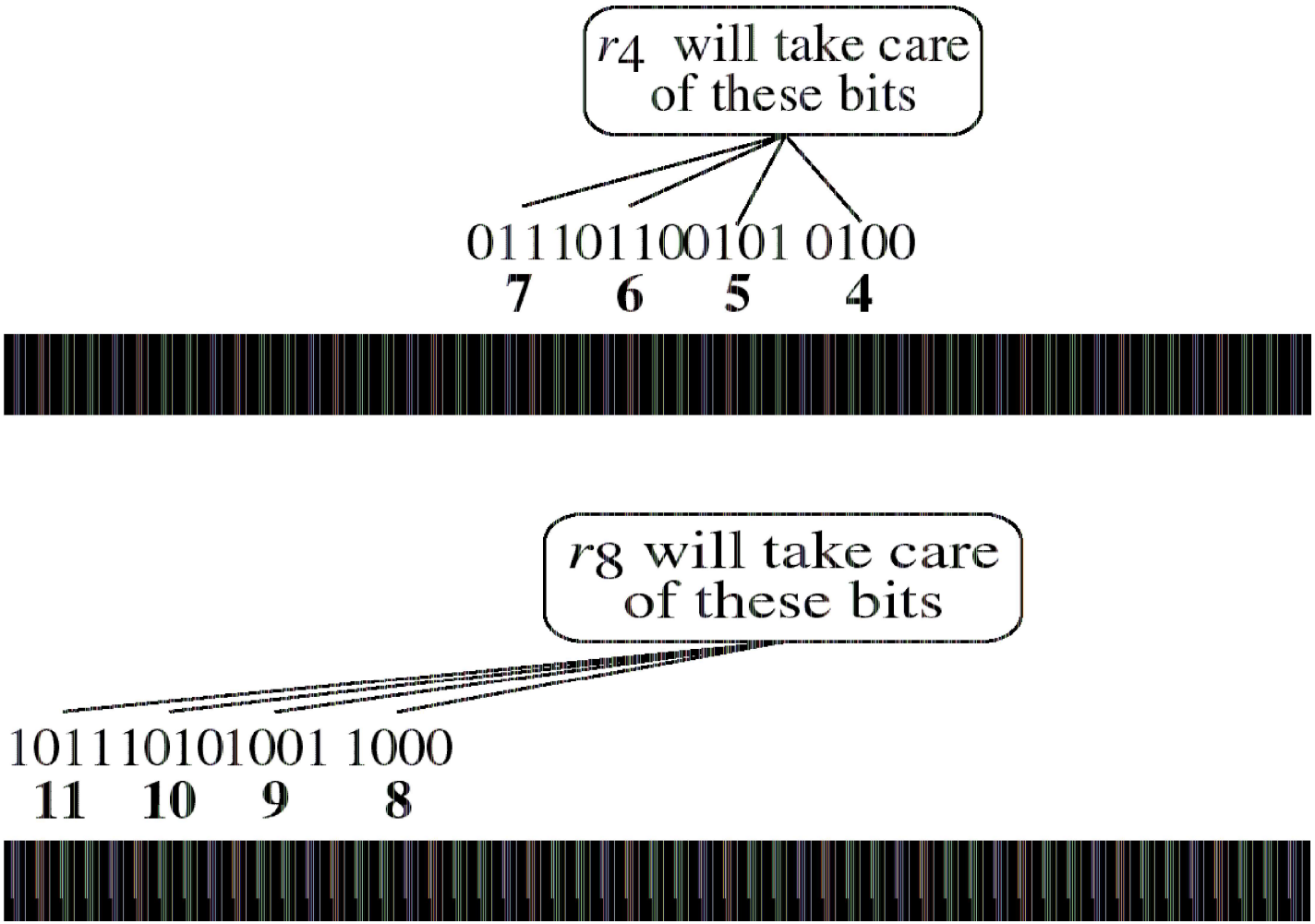
- R1- r1 d3 d5 d7 d9 d11
- R2-r2 d3 d6 d7 d10 d11
- R4-r4 d5 d6 d7
- R8-r8 d9 d10 d11

Note: these bits are fixed.

Hamming Code



Hamming Code



Error position	Positic c3
0 (no error)	0 (
1	0 (
2	0
3	0
4	1 (
5	1 (
6	1
7	1

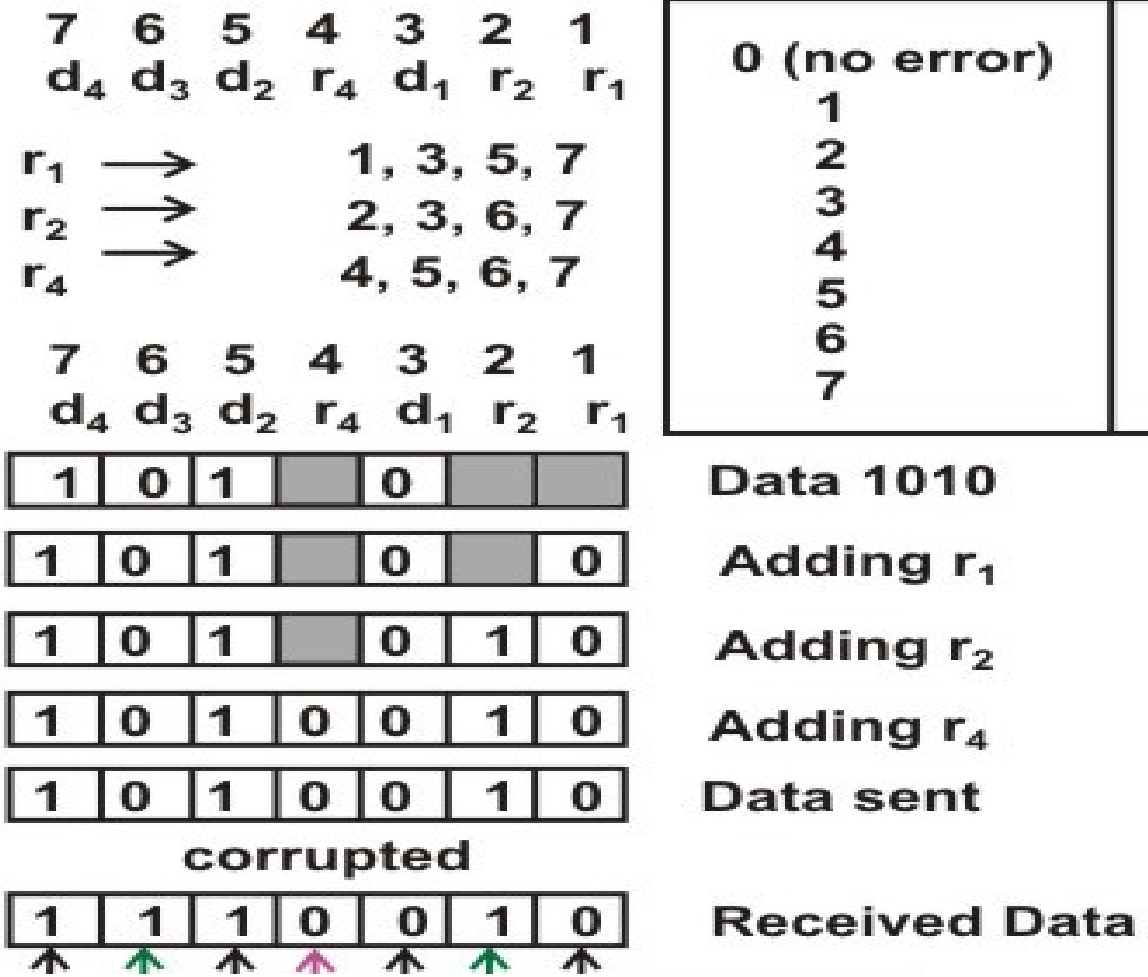


Figure 9-19

Hamming Code

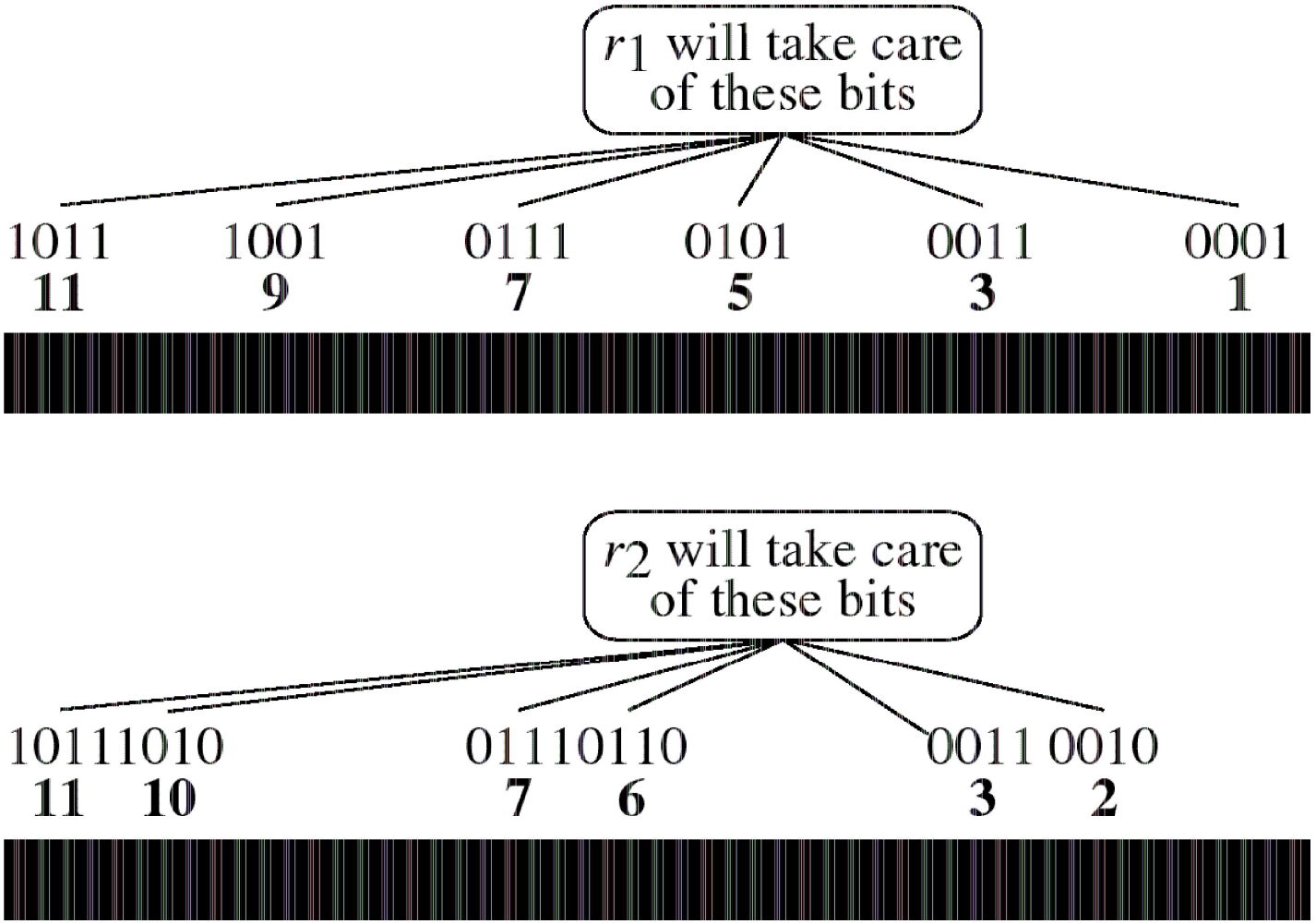


Figure 9-19-continued

Hamming Code

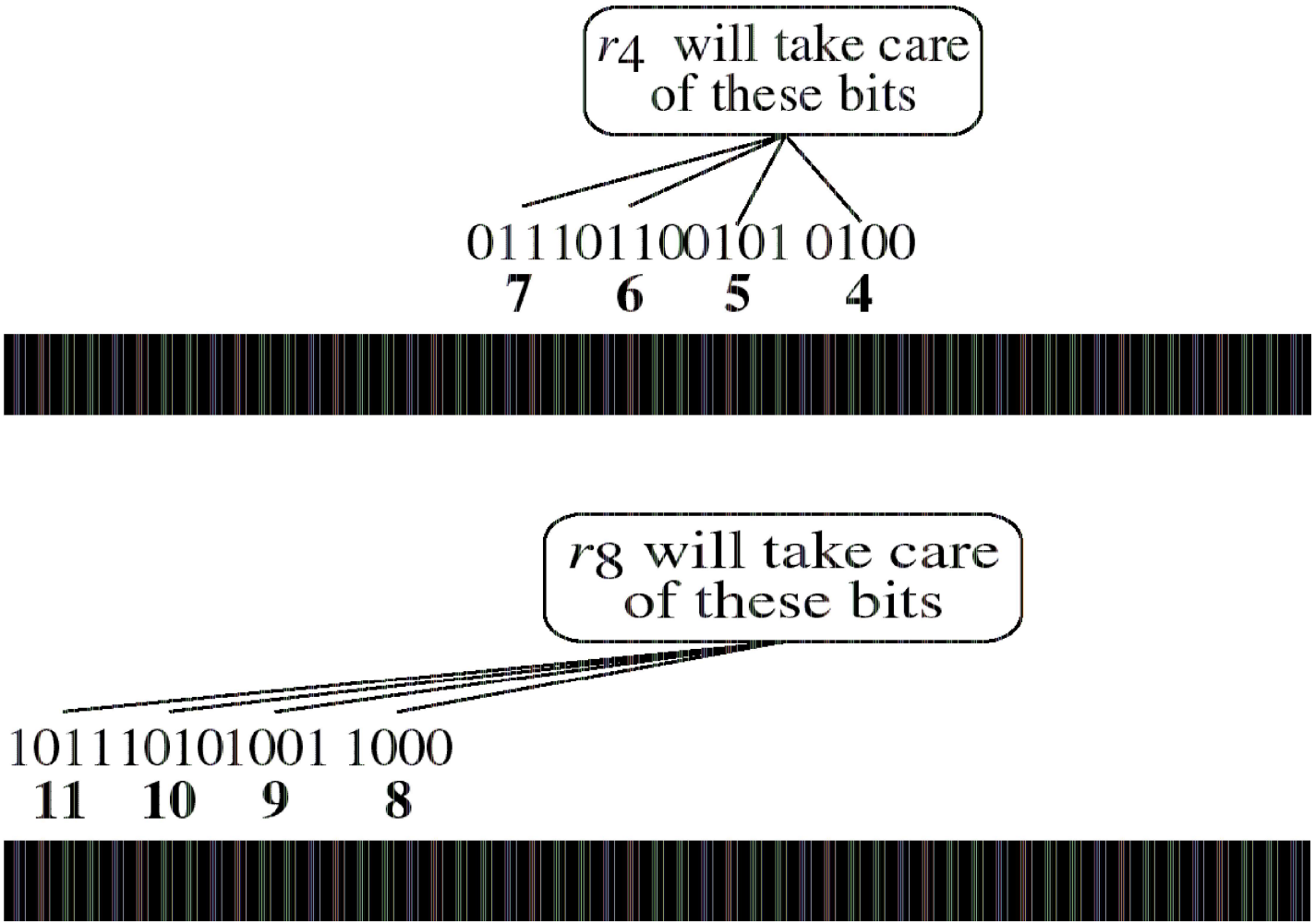


Figure 9-20

Example of Hamming Code

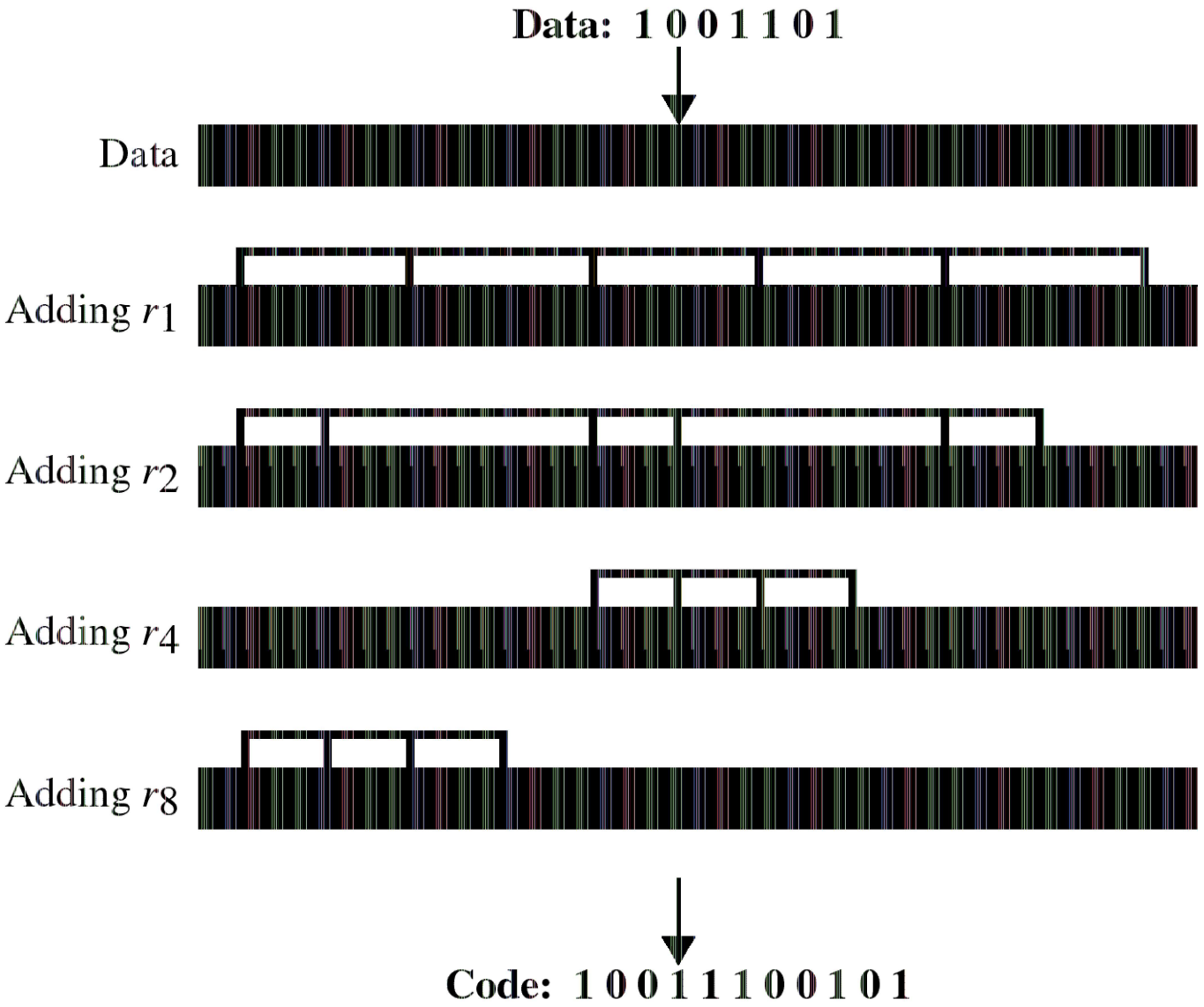


Figure 9-21

Single-bit error

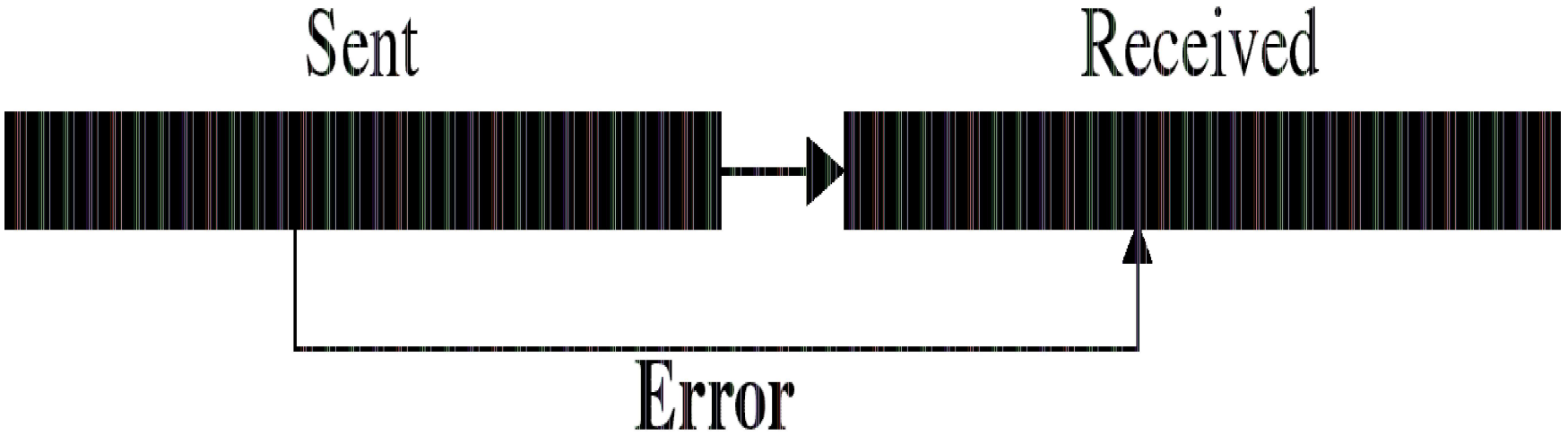
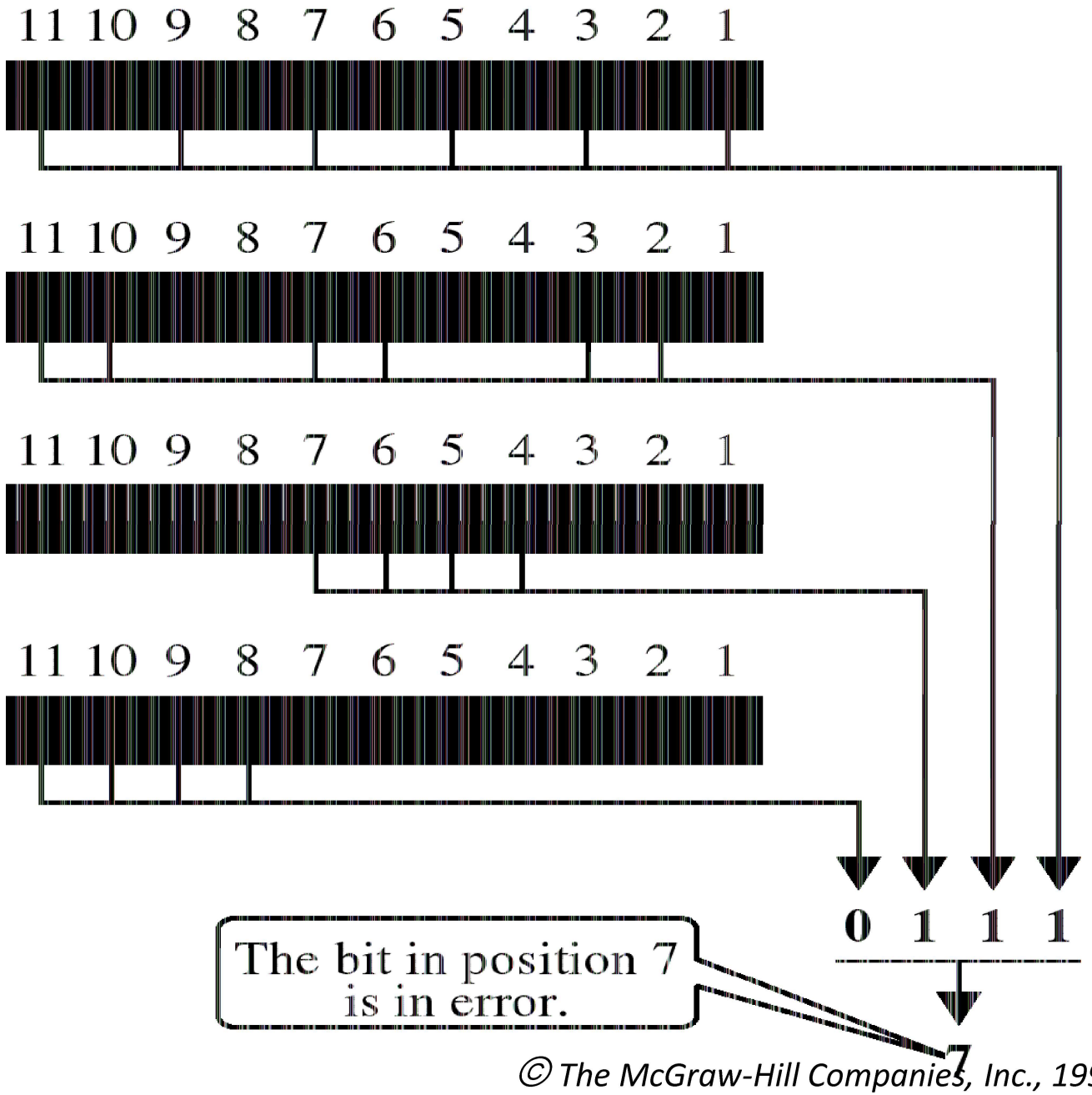


Figure 9-22

Error Detection



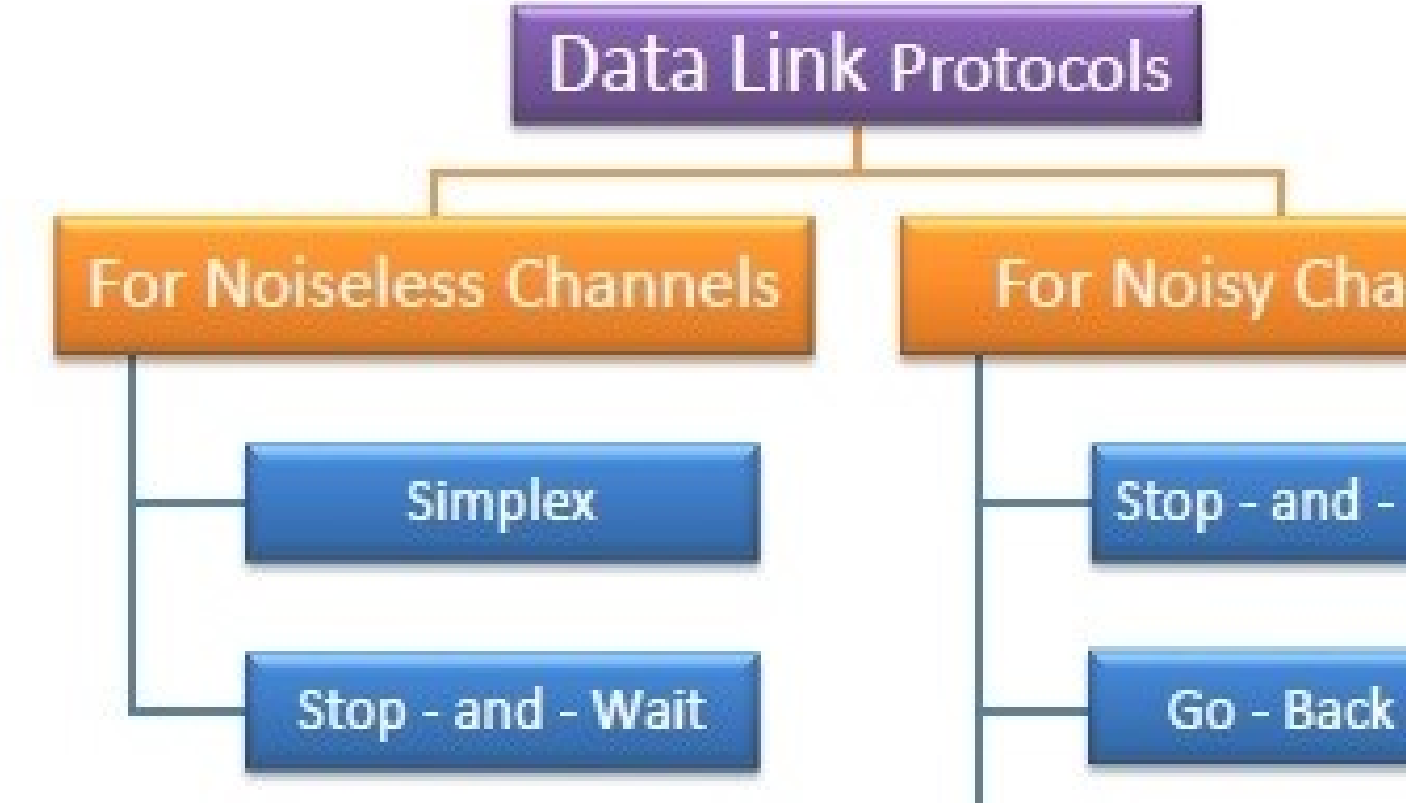
- Above figure shows how hamming code is used for correction for 4-bit numbers (d4d3d2d1) with the help of three redundant bits (r3r2r1).
- For the example data 1010, first r1 (0) is calculated considering the parity of the bit positions, 1, 3, 5 and 7. Then the parity bits r2 is calculated considering bit positions 2, 3, 6 and 7. Finally, the parity bits r4 is calculated considering bit positions 4, 5, 6 and 7 as shown.
- If any corruption occurs in any of the transmitted code 1010010, the bit position in error can be found out by calculating r3r2r1 at the receiving end.
- For example, if the received code word is 1110010, the recalculated value of r3r2r1 is 110, which indicates that bit position in error is 6, the decimal value of 110.

Data Link Layer Protocols

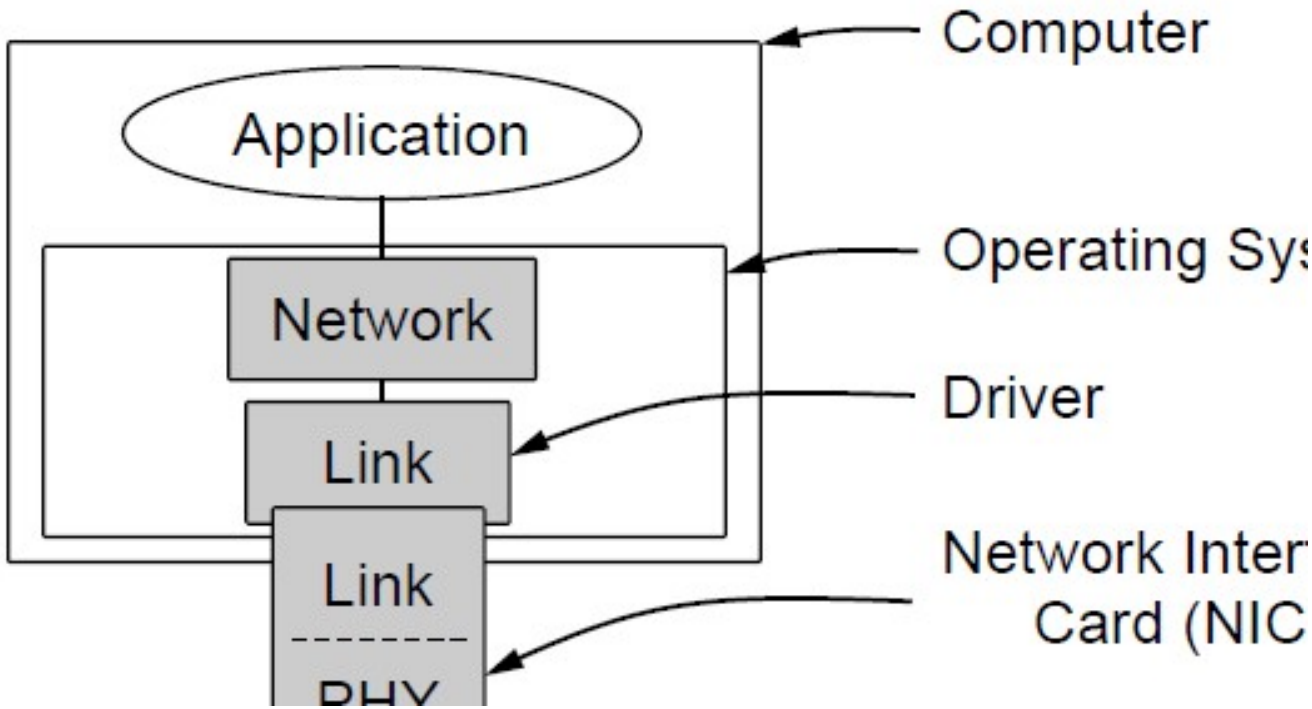
Flow Control in Data Link Layer

FLOW CONTROL

- Flow control **coordinates the amount of data that can be sent before receiving an acknowledgment** and is one of the most important duties of the data link layer.
- In most protocols, flow control is a set of procedures that tells the sender how much data it can transmit before it must wait for an acknowledgment from the receiver.
- The flow of data must **not be allowed to overwhelm the receiver**.
- Any receiving device has a limited speed at which it can process incoming data and a limited amount of memory in which to store incoming data.
- The receiving device must be able to inform the sending device before those limits are reached and to request that the transmitting device send fewer frames or stop temporarily.


Data Link Protocols
For Noiseless Channels
For Noisy Cha
Simplex
Stop - and -
Stop - and - Wait
Go - Back

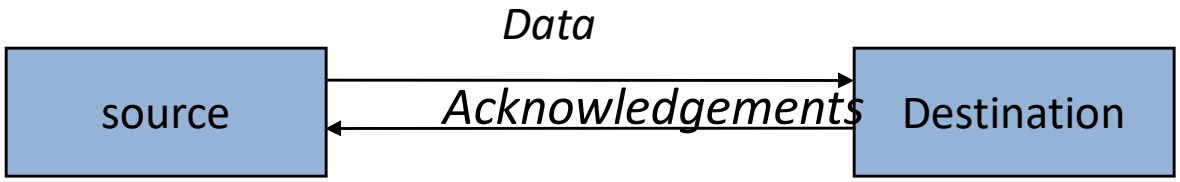
Elementary Data Link Protocols



Implementation of the physical, data link, and network layers.

Sliding window protocols:

- In the Previous protocols ,Data frames were transmitted in one direction only.
- In most practical situations ,there is a need for transmitting data in both directions.
- One way of achieving full-duplex data transmission is to have two separate communication channels and each one for simplex data traffic (in different directions)
- we have two separate physical circuits ,each with a “forward” channel (for data) and a “reverse” channel(for acknowledgements).

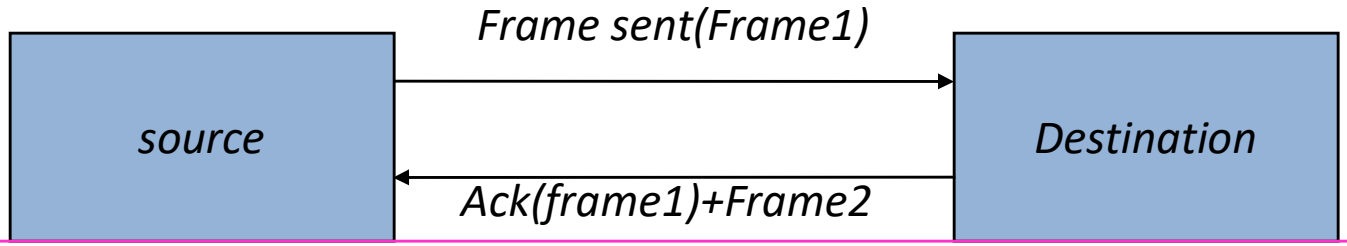


Sliding window protocols:

- Disadvantage: The bandwidth of the reverse channel is almost entirely wasted.
- A better idea is to use the same circuits for data in both directions
- In this model the data frames from A and B are intermixed with the acknowledgement frames from A to B.
- Kind: kind field in the header of an incoming frame, the receiver can tell whether the frame is data or acknowledgements.

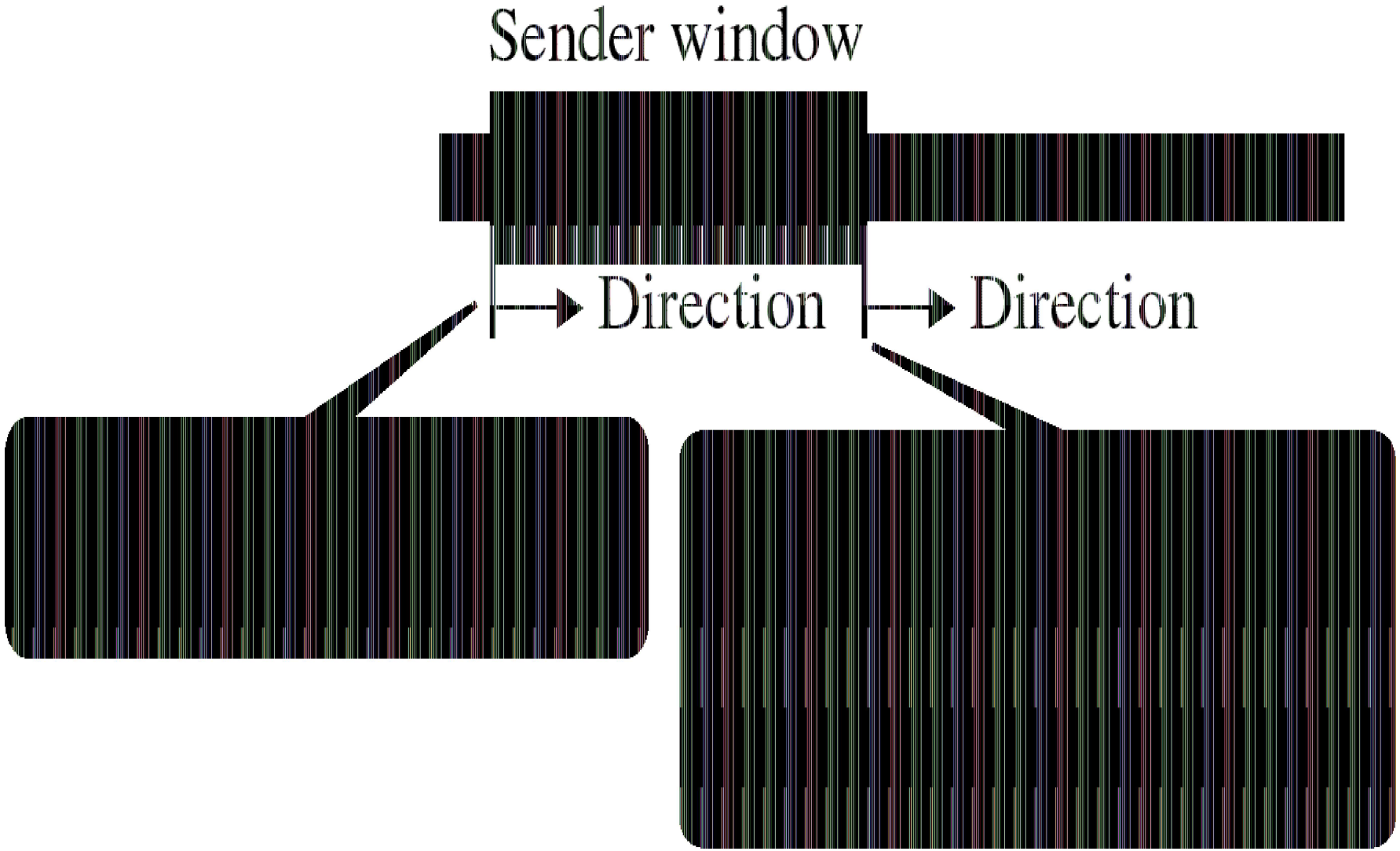
Sliding window protocols:

- **Piggybacking:**



- When a data frame arrives, instead of immediately sending a separate control frame, the receiver restrains itself and waits until the network layer passes it the next packet.
- The acknowledgement is attached to the outgoing data frame
- Disadv: This technique is temporarily delaying outgoing acknowledgements.
- If the data link layer waits longer than the senders timeout period, the frame will be retransmitted.

Sender Sliding Window



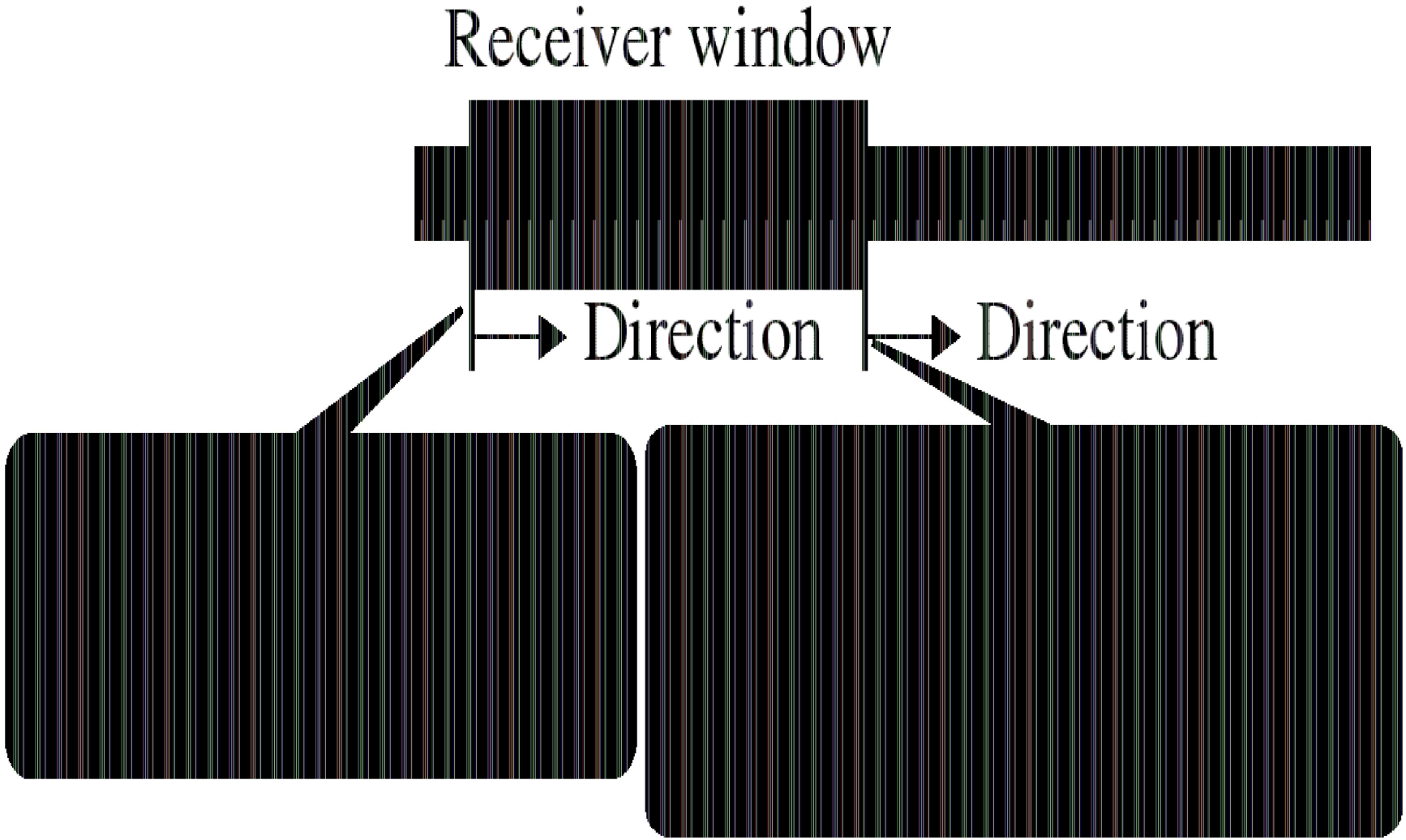
Sliding Window Protocol (Duplex)

- At any instant of time, the sender maintains a set of sequence numbers corresponding to frames it is permitted to send – this includes
  - frames not yet sent
  - Frames sent but not yet acknowledged .. May have to be sent again

  This corresponds to the sender's window. When a packet comes from the Network Layer ..it is appended in the window and when an ack arrives for a frame..it is deleted from the buffer(and from the window)

Figure 10-13

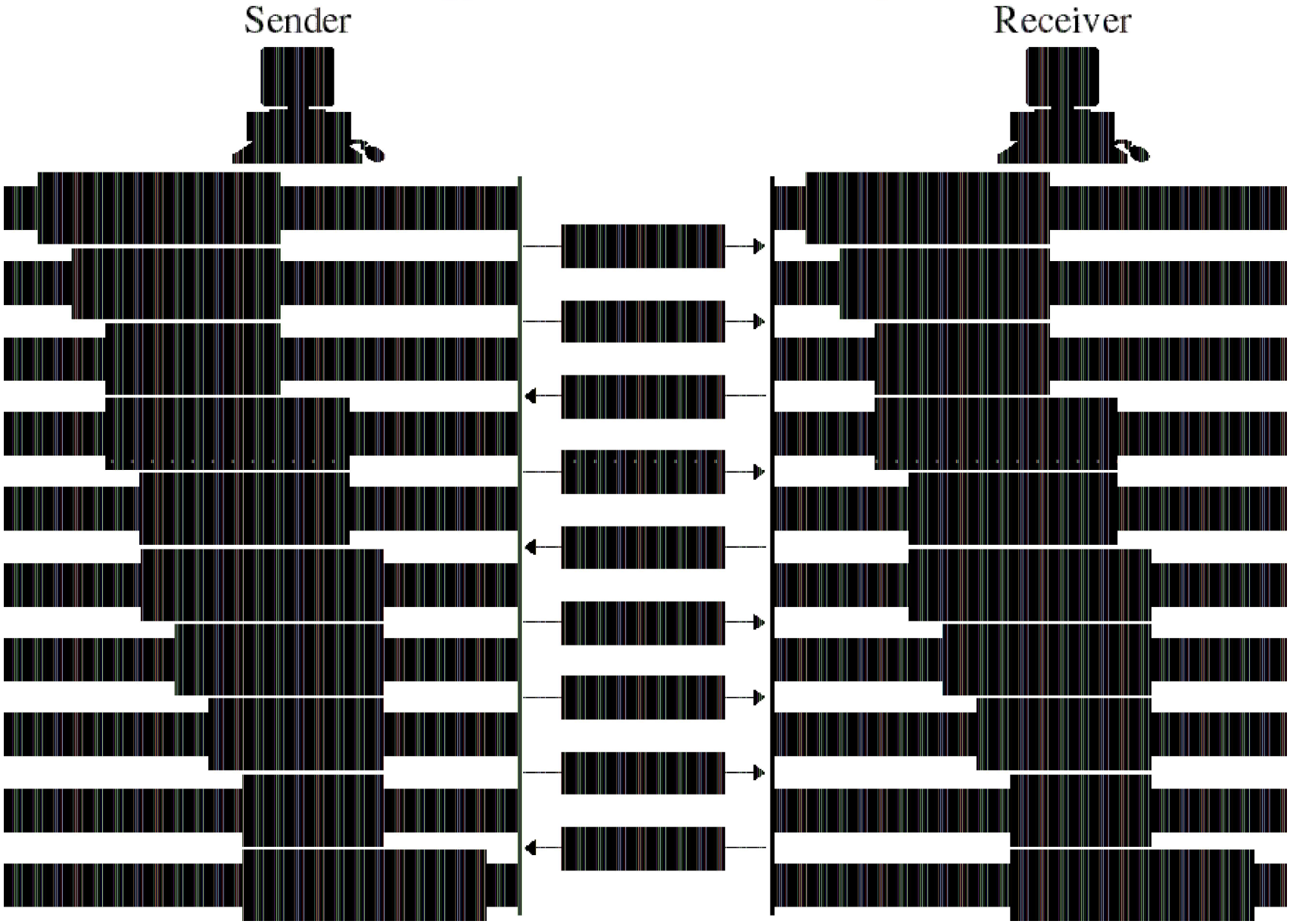
Receiver Sliding Window



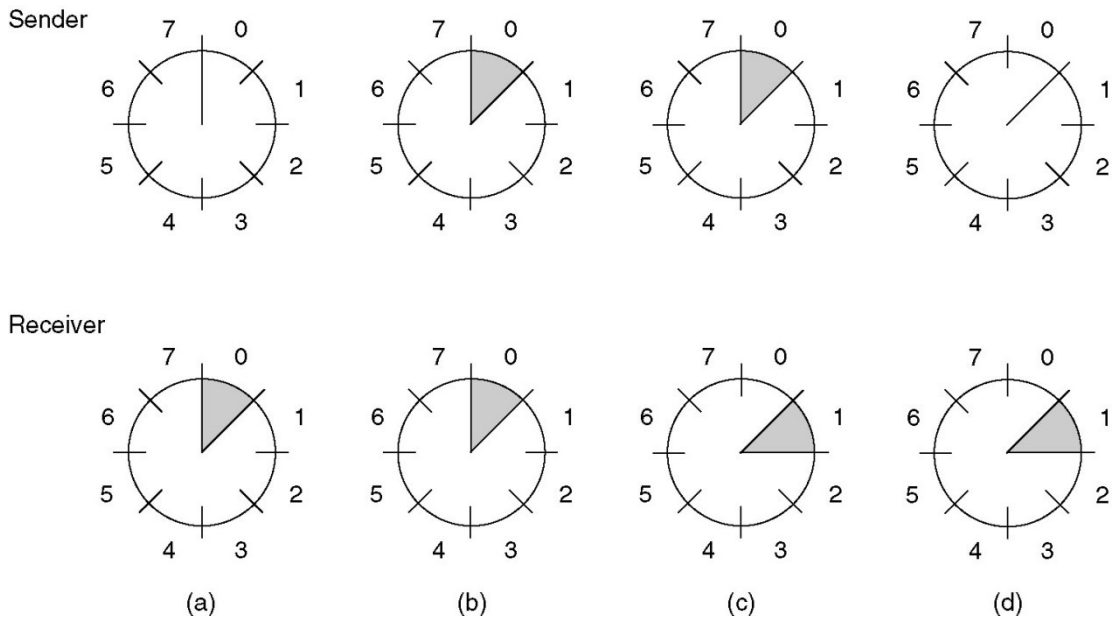
Sliding Window Protocol(Duplex)

- Similarly, the receiver also maintains a set of sequence numbers corresponding to frames it is permitted to accept.
  - When a frame arrives, its seq number is checked..if it falls in the window it is accepted else discarded .. Once an ack is sent for a frame it is deleted from the window so that duplicate frames are not accepted.
- Sliding window gives the Data Link Layer the flexibility to accept the frames in any order but they must be handed over to the Network Layer in order.

Sliding Window Example



Sliding Window Protocols



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.

Sliding Window Protocols

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

Sliding window protocols:

- 1-bit sliding window protocol:
  - Window size 1.
  - Stop-and-wait.(ARQ)
  - Must get ack before can send next frame.
  - Both machines are sending and receiving.
  - Problem: Only one frame will be in transit.

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 */
```

Continued →

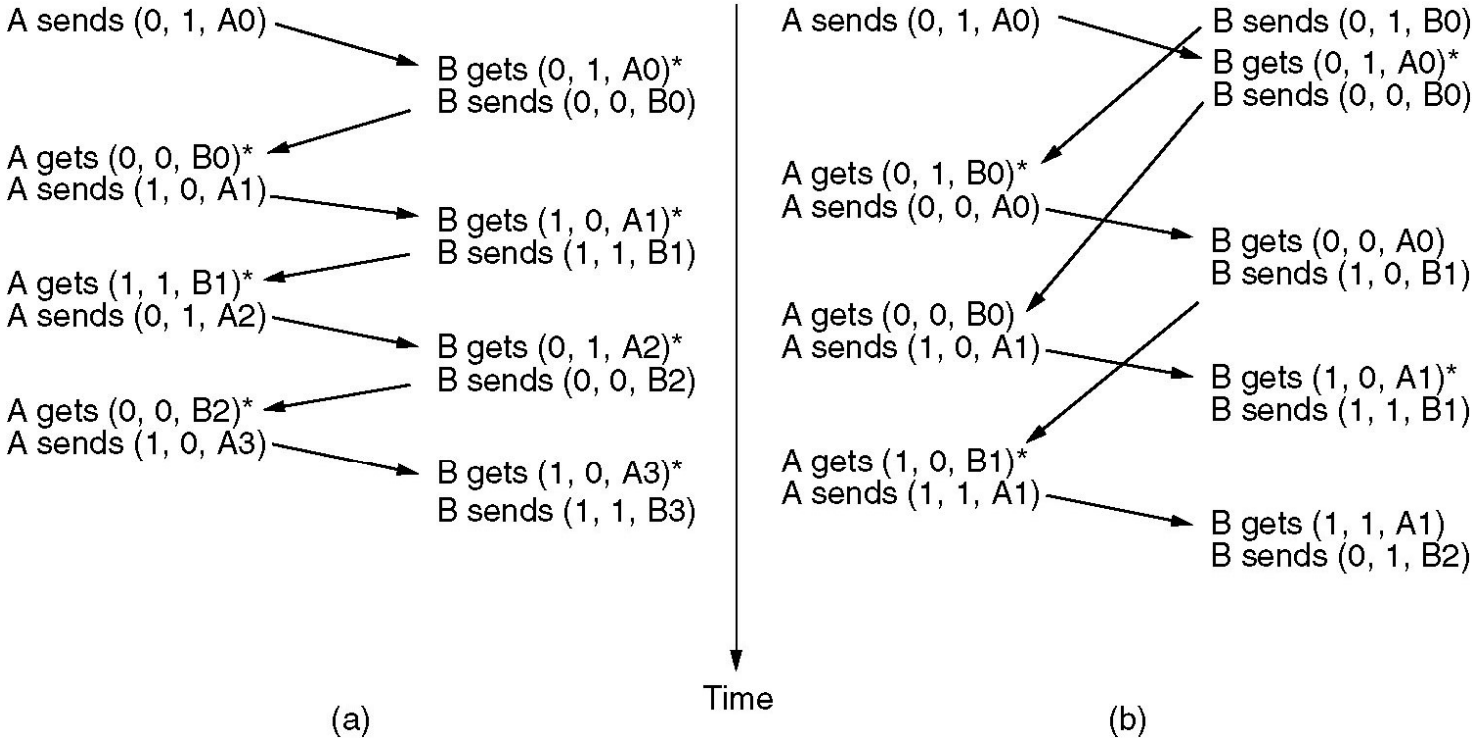
A One-Bit Sliding Window Protocol (ctd.)

```
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 */
  }
}
```

A One-Bit Sliding Window Protocol (2)



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

Bad scene : duplicates without transmission error

- When B sends before receiving a frame from A.

Long propagation delays

- Consider a 50 kbps satellite channel with a round trip delay of 500 msec (10^-3 sec).
- To send a frame of size 1000 bits ..it takes = 1000/50kbps = 20msec.
- At t=0, the sender sends the first bit and at

  t = 20 msec it sends the last bit.
- At t= 270 msec, the frame completely reaches the receiver.
- Not before t= 520 msec , the ack arrives back at the receiver (it will be later than this). This means that the sender was blocked for 500 msec out of 520 msec.

Contd..

- That means
  - Long propagation delays
  - High bandwidth
  - And, short frame length

  Is bad in terms of channel efficiency.

  A large window size is needed at the sender's end whenever the bandwidth and round trip delay is large.

Pipelining

- Imagine in the above example, if the sender could send more frames without waiting for the ack for the first one, the channel could have been utilized more efficiently. This technique is called pipelining.

Pipelining contd…

Let

- Channel capacity is b bits/sec
- Frame size l bits,
- Round trip propagation time is R
- The time required to transmit a single frame is l/b.
- So, in stop n wait, the line is busy for l/b time and idle for R time, thus the line utilization is

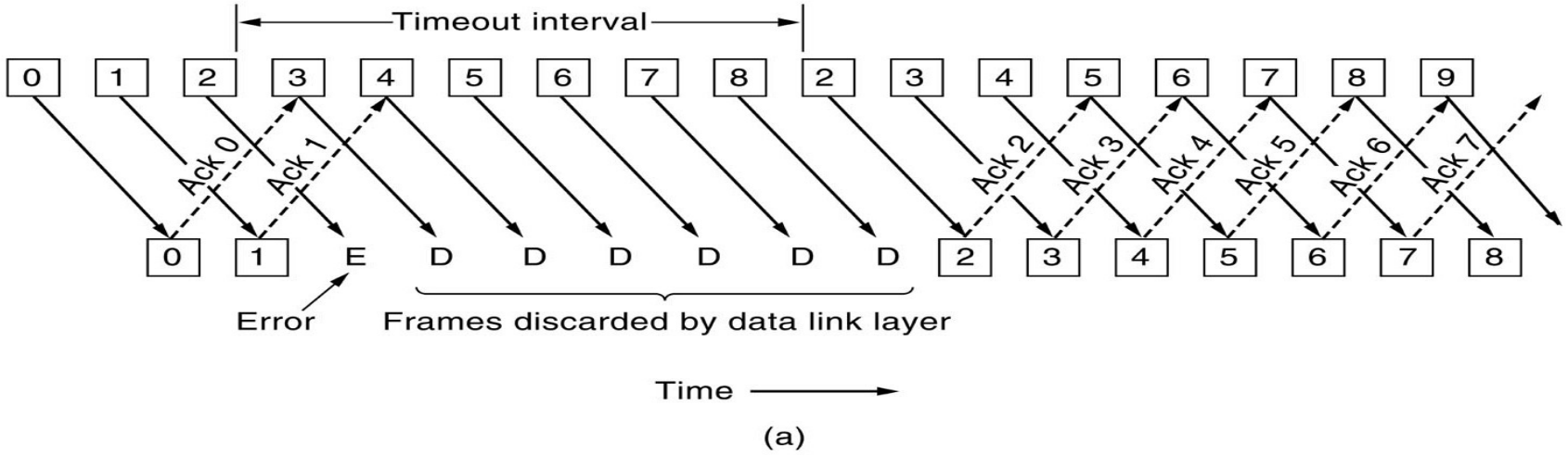
  $l/(l + bR)$.

  If $l < bR$, the efficiency will be less than 50%.

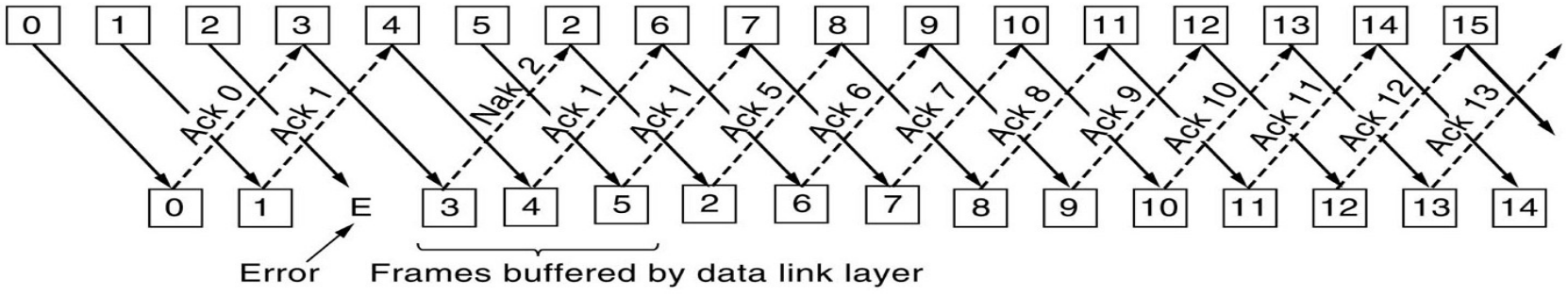
Pipelining over noisy channels

- What if one or more frames is errored.
- Two options :
  - Throw away all the subsequent frames and go back and start resending from the damaged frame. - Go back N Protocol
  - Buffer the subsequent frames and wait for the damaged frame to be resent, once that comes handover the frames in proper order to the Network Layer-Selective Repeat Protocol

A protocol using Go Back N and selective repeat:



(a)



(b)

A protocol using Go Back N:

Go Back n, is for the receiver simply to discard all subsequent frames, sending no acknowledgements for the discarded frames. This strategy corresponds to a receive window of size 1.

The data link layer refuses to accept any frame except the next one it must give to the network layer.

Points: fig (a)

1) frame 0 and 1 correctly received and Acknowledged

2) Frame 2 is damaged or lost ,The sender continuous to send frames until the timer for frame 2 expires.

3) Then it backs up to frame 2 and starts all over with it ,sending 2,3,4 etc. all over again.

Selective Repeat:

- When selective repeat is used ,a bad frame that is received is discarded. But good frames after it are buffered.
- when the senders timeout ,only the oldest unacknowledged frame is retransmitted.
- If that frames arrives correctly ,the receiver can deliver it to the network layer in sequence all the frames it has buffered.
- fig (b) :
  - 1) Frames 0 and 1 correctly received and acknowledged.
  - 2) Frame 2 is lost, when frame 3 arrives at the receiver ,the data link layer notices that it has a missed frame. so it sends back a NAK for 2 but buffers 3
  - 3) when frame 4 and 5 arrive ,they are buffered by the data link layer instead of being passed to the network layer
  - 4)The NAK 2 gets back to the sender which immediately resends frame 2.

Comparison

- Trade off between the bandwidth utilization and the buffer space in the receiver.

Sliding Window Protocol 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 */
}
```

Continued →

Sliding Window Protocol Using Go Back N

```
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 */
```

Continued →

Sliding Window Protocol Using Go Back N

```
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 */
          }
```

Continued →

Sliding Window Protocol Using Go Back N

```
            /* 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();
 }
}
```

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)) || ((c < a) && (a <= b)) || ((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 */
}
```

Continued →

A Sliding Window Protocol Using Selective Repeat (2)

```
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;
```

Continued →

A Sliding Window Protocol Using Selective Repeat (2)

```
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 */
                   }
              }
         }
```

Continued →

A Sliding Window Protocol Using Selective Repeat (4)

```
            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();
  }
}
```

		Stop and Wait protocol	Sliding Window protocol
1	Mechanism	In Stop and Wait protocol, sender sends single frame and waits for acknowledgment from the receiver.	In Sliding window protocol, sender sends multiple frames at a time and retransmits the damaged, lost frames.
2	Efficiency	Stop and Wait protocol is less efficient.	Sliding Window protocol is more efficient than Stop and Wait protocol.
3	Window Size	Sender's window size in Stop and Wait protocol is 1.	Sender's window size in Sliding Window protocol varies from 1 to n.
4	Sorting	Sorting of frames is not needed.	Sorting of frames is needed. Because Network Layer accepts frames in order only
5	Duplex	Stop and Wait protocol is half duplex in nature.	Sliding Window protocol is full duplex in nature.
6	Propagation delay	Long	Short
7	Link utilisation	Poor	Better

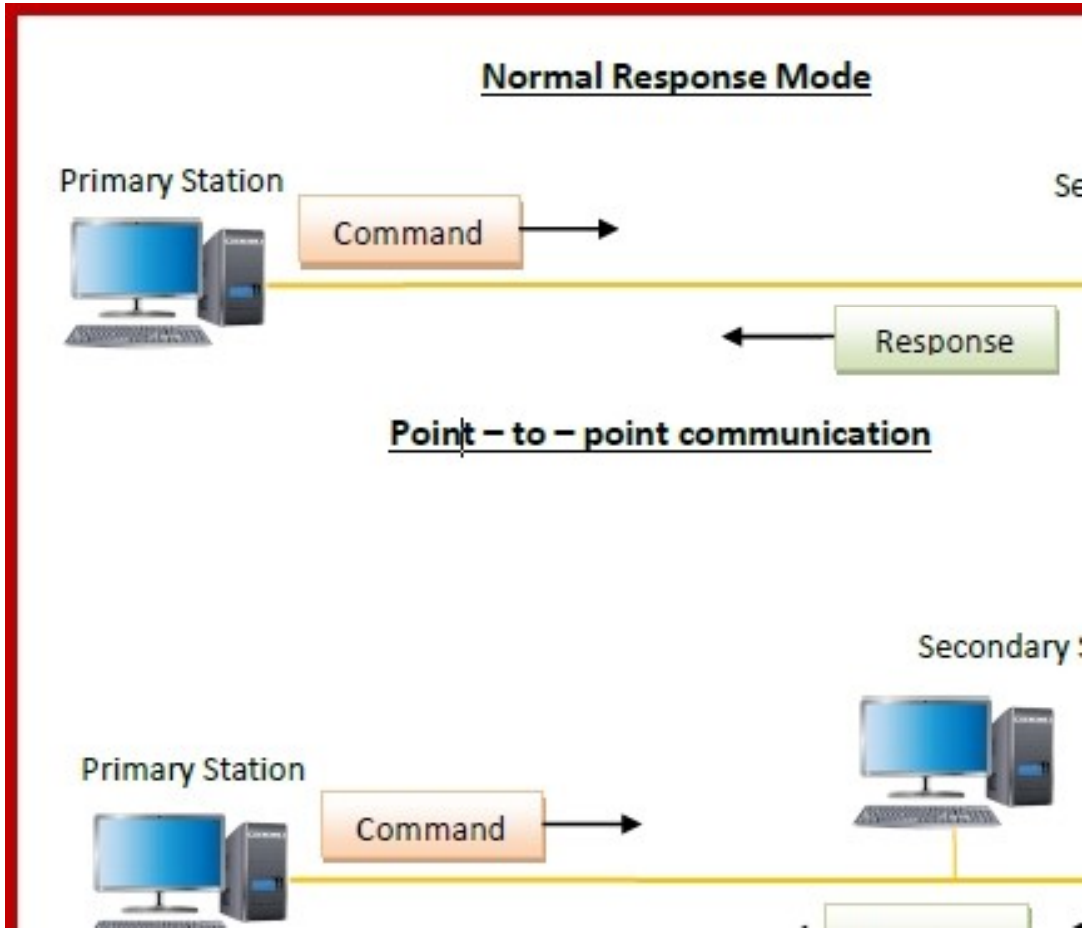
Comparison

	Go-Back-N	Selective Repeat
Basic	Retransmits all the frames that sent after the frame which is damaged or lost.	Retransmits only those frames that are lost or damaged
Bandwidth Utilization	If error rate is high, it wastes a lot of bandwidth.	Comparatively less bandwidth is wasted in retransmitting.
Complexity	Less complicated.	More complex as it require to apply extra logic and sorting and storage, at sender and receiver.
Sorting	Sorting is neither required at sender side nor at receiver side.	Receiver must be able to sort as it has to maintain the sequence of the frames.

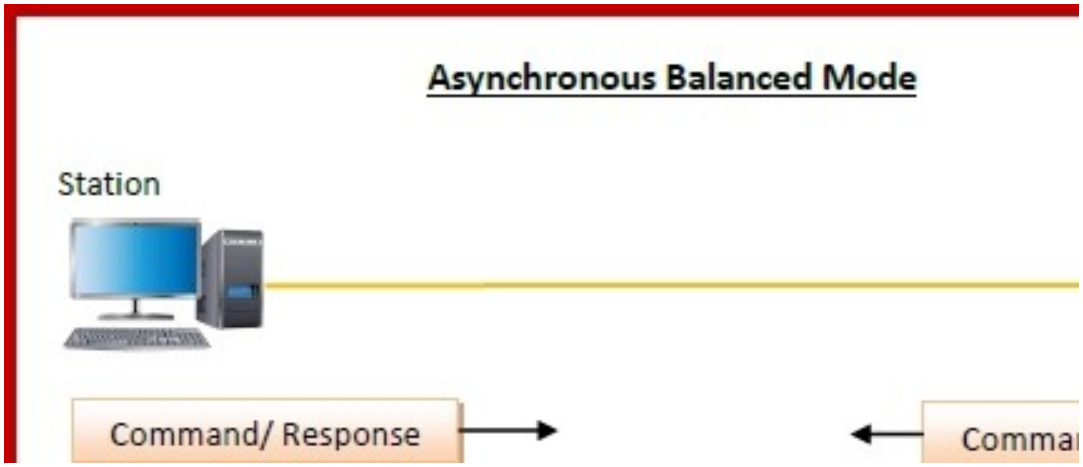
HDLC-High Level Data Link Control

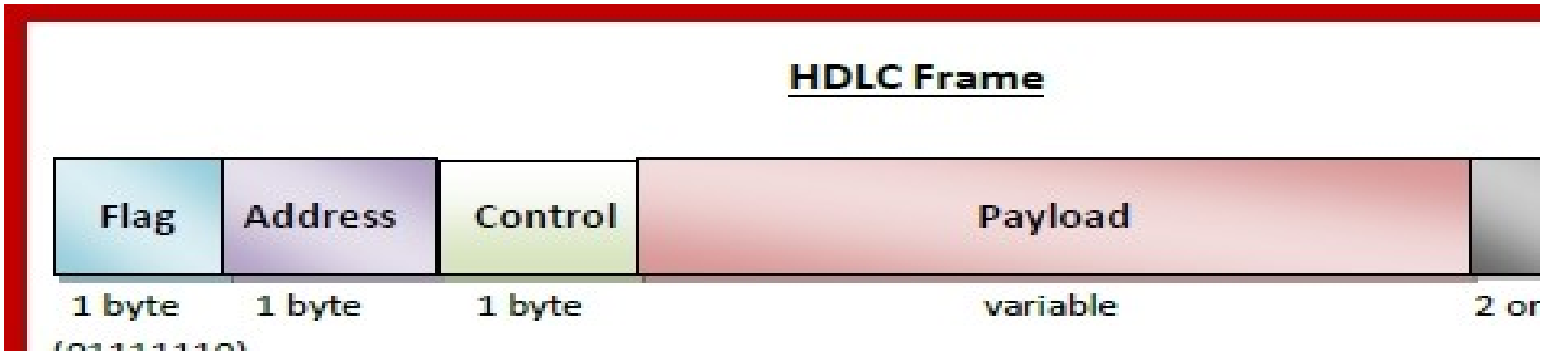
- *HDLC is a bit oriented protocol that supports both half-duplex and full-duplex communication over point to point & multipoint link.*
- Since it is a data link protocol, data is organized into frames. A frame is transmitted via the network to the destination that verifies its successful arrival
- *For any HDLC communications session, one station is designated as primary and the other is secondary.*
- **Transfer Modes-** HDLC supports two types of transfer modes, normal response mode and asynchronous balanced mode.

Normal Response Mode (NRM) – Here, two types of stations are there, a primary station that send commands and secondary station that can respond to received commands. It is used for both point - to - point and multipoint communications.



Asynchronous Balanced Mode (ABM) – Here, the configuration is balanced, i.e. each station can both send commands and respond to commands. It is used for only point - to - point communications.





Flag – It is an 8-bit sequence that marks the beginning and the end of the frame. The bit pattern of the flag is 01111110.

Address – It contains the address of the receiver. If the frame is sent by the primary station, it contains the address(es) of the secondary station(s). If it is sent by the secondary station, it contains the address of the primary station. The address field may be from 1 byte to several bytes.

Control – It is 1 byte containing flow and error control information.

Payload – This carries the data from the network layer. Its length may vary from one network to another.

FCS – It is a 2 byte or 4 bytes frame check sequence for error detection. The standard code used is CRC (cyclic redundancy code)

High level data link control:

- *Frame Formats- 3 types*:
- *I-Frame (Information Frame)- is used to carry user data and control information related with user data*
- *S-Frame (Supervisory Frame)- is used to transport control information.*
- *U-Frame(Unnumbered Frame)- reserved for system management*

HDLC frame types

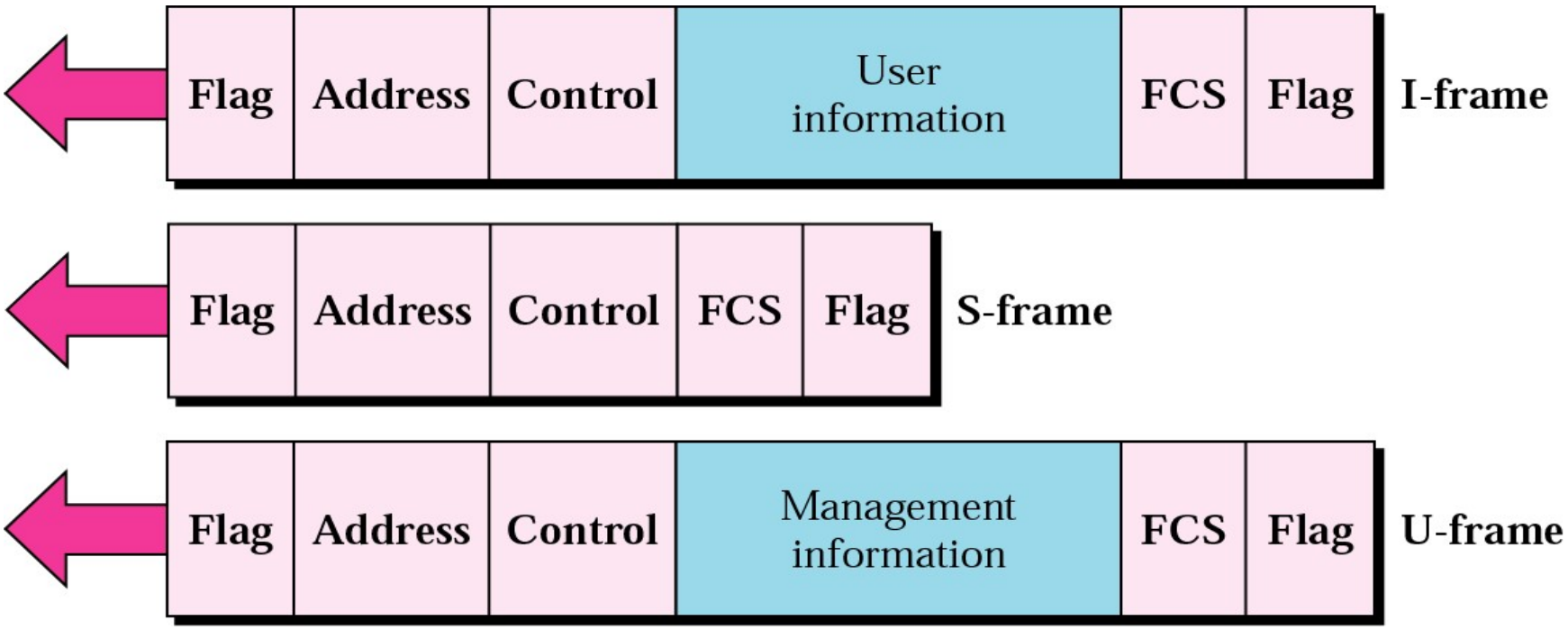


Figure 11-17

HDLC Flag Field

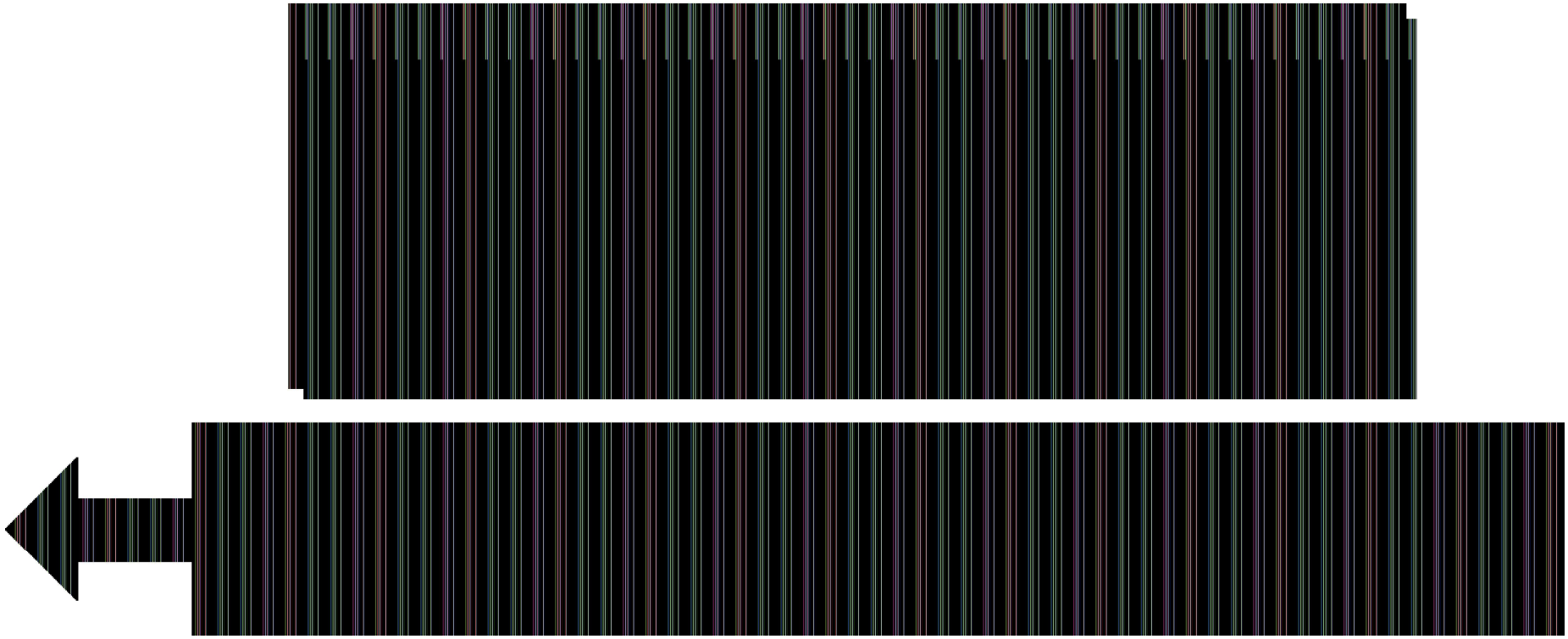


Figure 11-19

HDLC Address Field

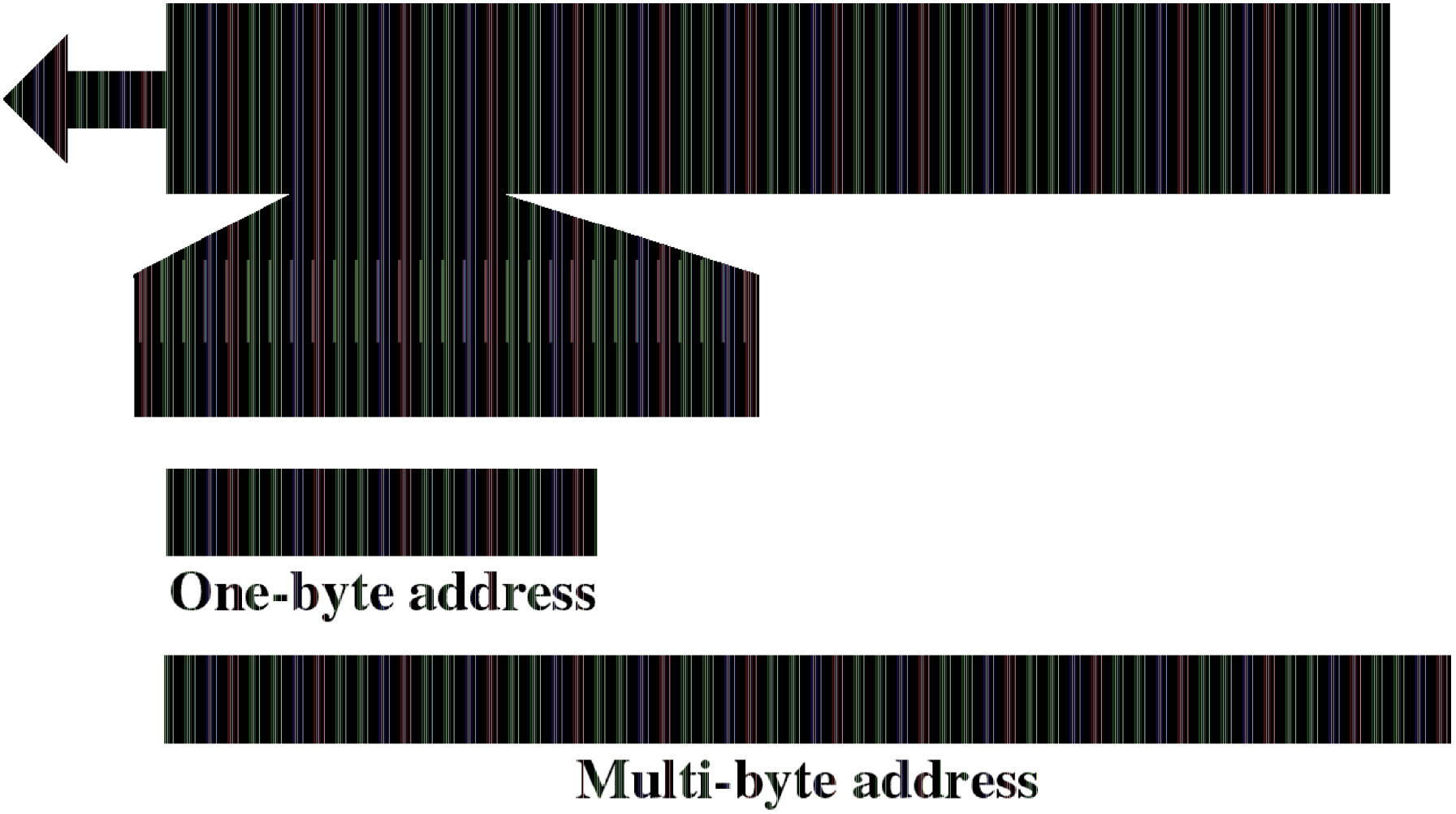
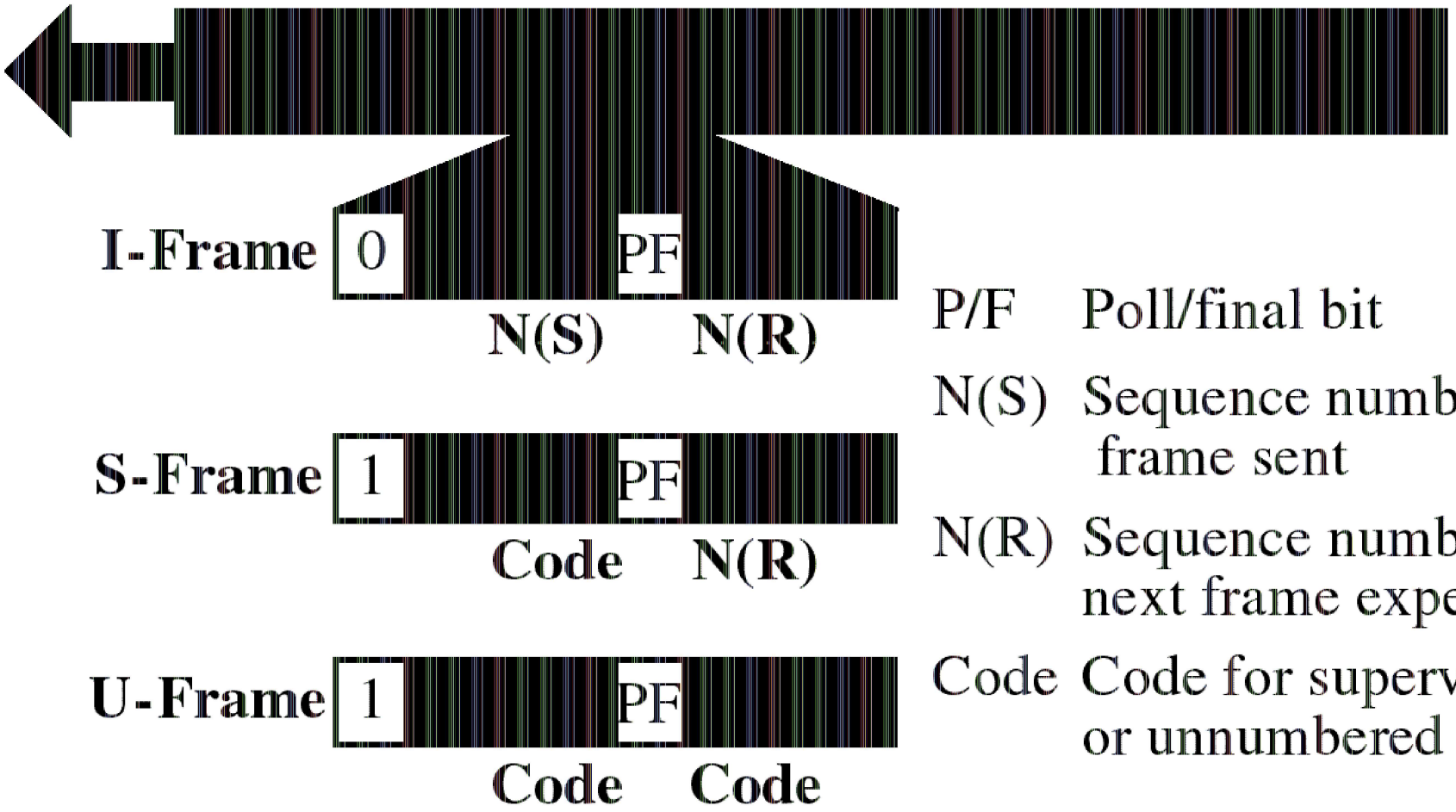


Figure 11-20

HDLC Control Field



P/F Poll/final bit

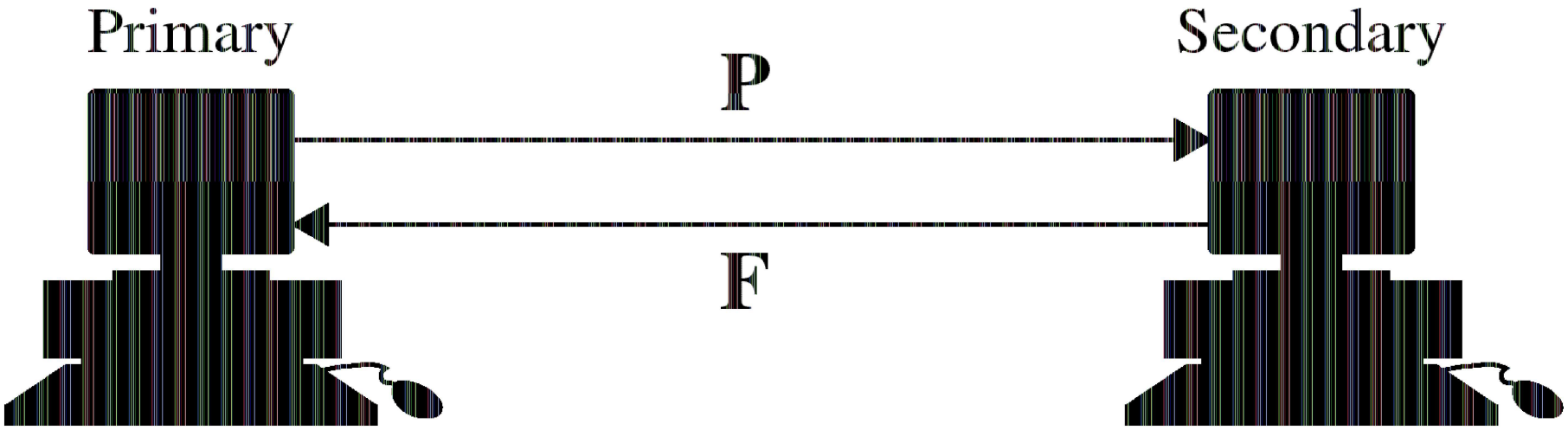
N(S) Sequence number of frame sent

N(R) Sequence number of next frame expected

Code Code for supervisory or unnumbered frame

Figure 11-21

Poll/Final



The *P/F field is a single bit with a dual purpose. It has meaning* only when it is set (bit = 1) and can mean poll or final. It means *poll when the frame* is sent by a primary station to a secondary (when the address field contains the address of the receiver). It means *final when the frame is sent by a secondary to a primary* (when the address field contains the address of the sender).

Figure 11-22

HDLC Information Field

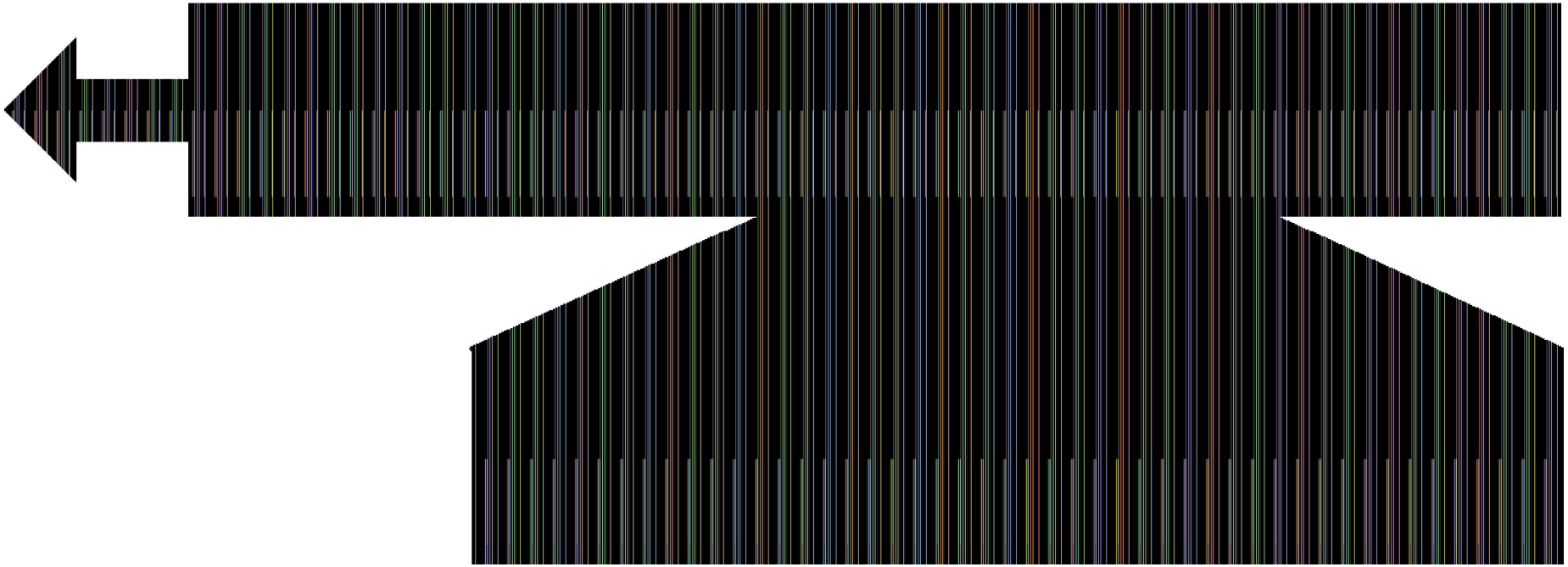


Figure 11-23

HDLC FCS Field

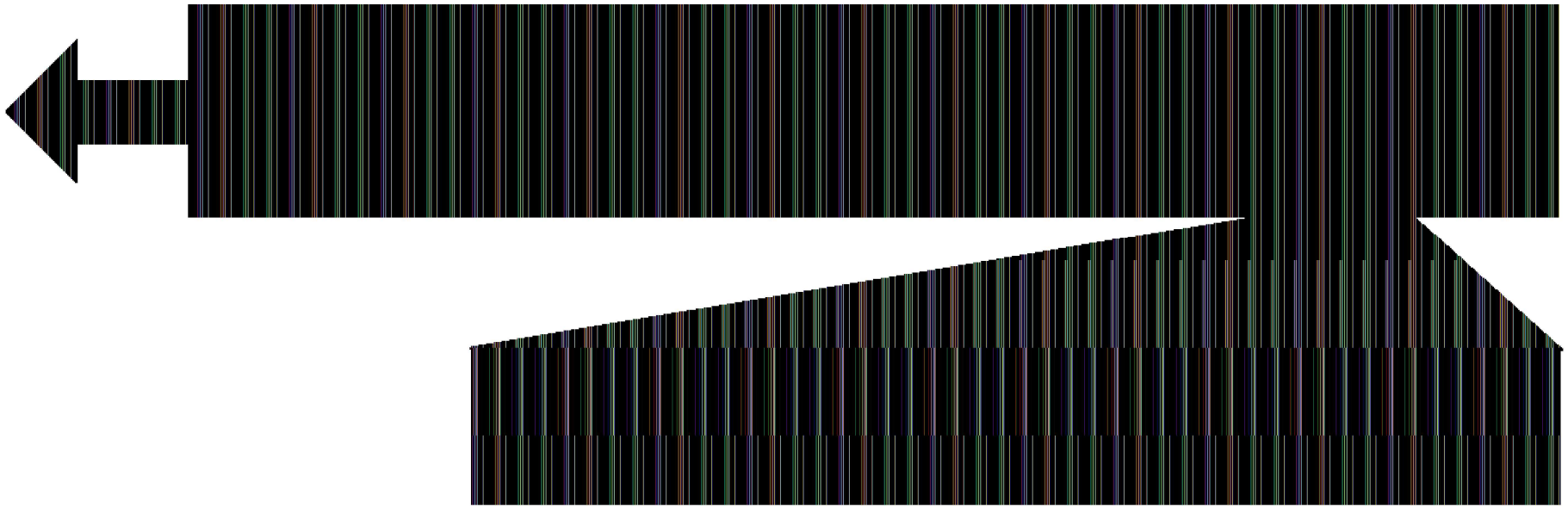
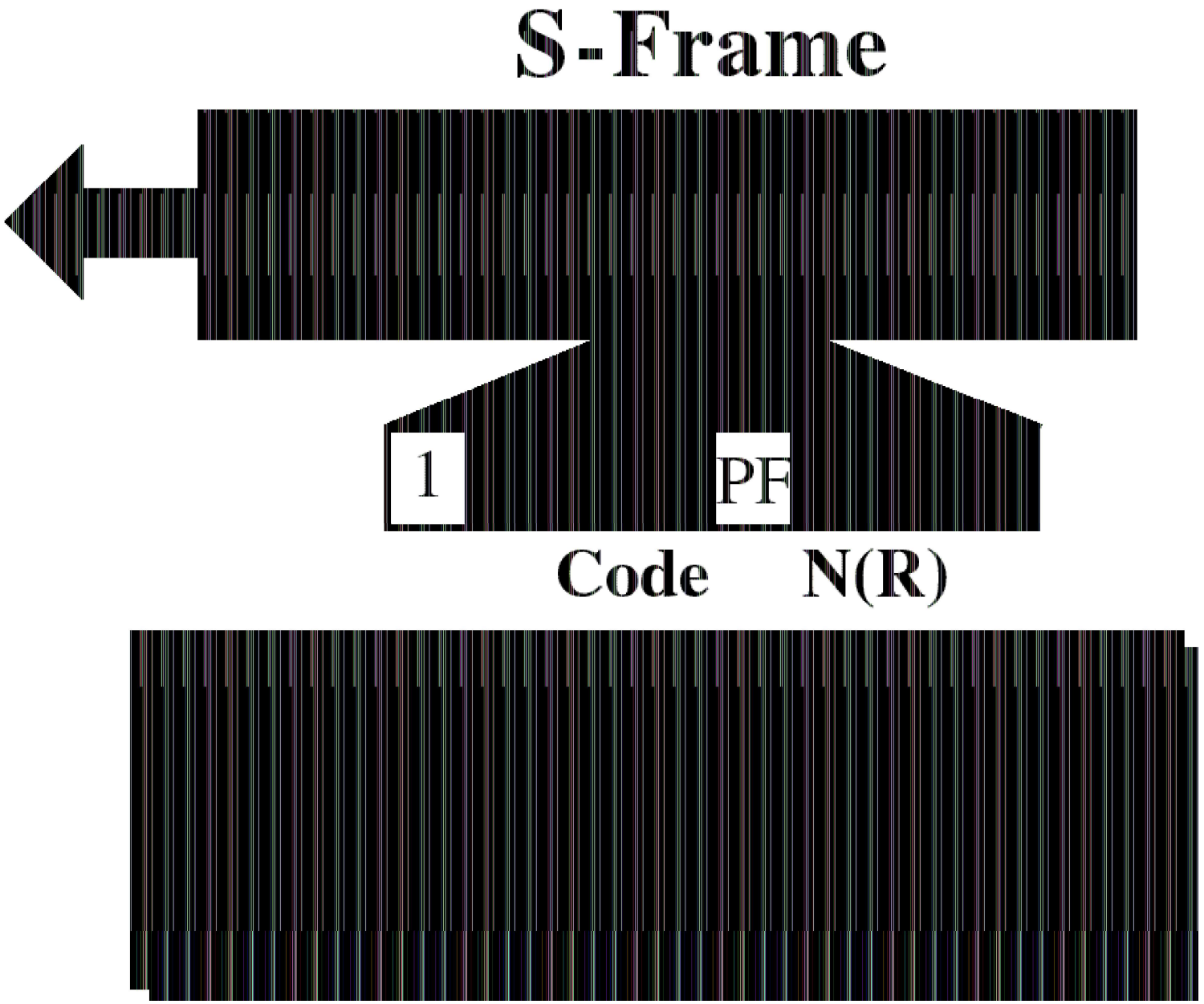


Figure 11-24

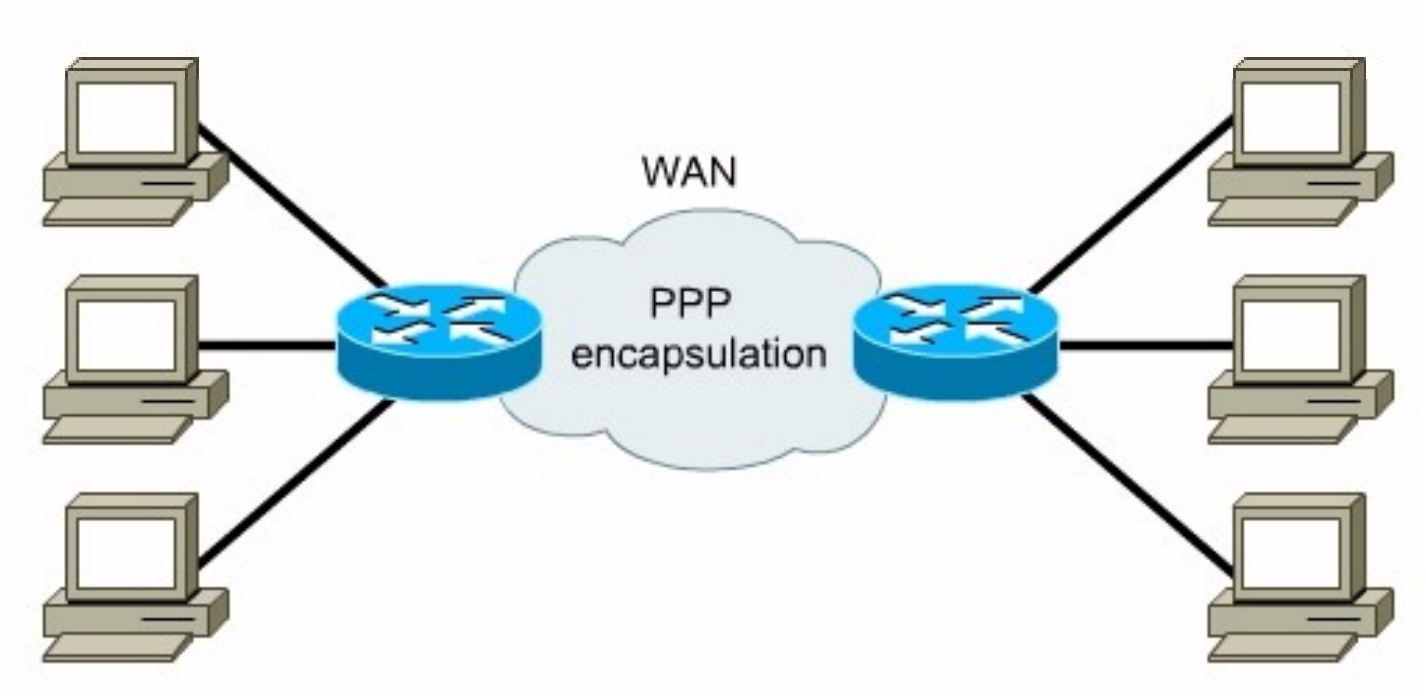


- Control field for U-frame

Code	*Command*	*Response*	*Meaning*
00 001	SNRM		Set normal response mode
11 011	SNRME		Set normal response mode, extended
11 100	SABM	**DM**	Set asynchronous balanced mode or **disconnect mode**
11 110	SABME		Set asynchronous balanced mode, extended
00 000	UI	**UI**	Unnumbered information
00 110		**UA**	**Unnumbered acknowledgment**
00 010	DISC	**RD**	Disconnect or **request disconnect**
10 000	SIM	**RIM**	Set initialization mode or **request information mode**
00 100	UP		Unnumbered poll
11 001	RSET		Reset
11 101	XID	**XID**	Exchange ID
10 001	FRMR	**FRMR**	Frame reject

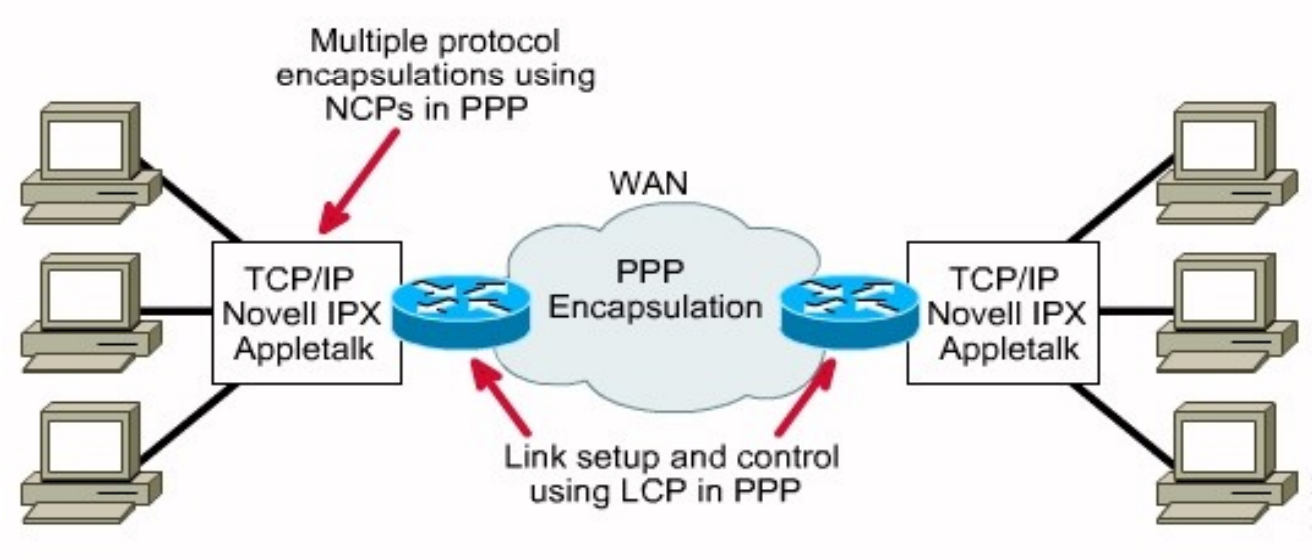
Point-to-Point Protocol (PPP)

- The Internet needs a point-to-point protocol for a variety of purposes, including router-to-router traffic and home user-to-ISP traffic.



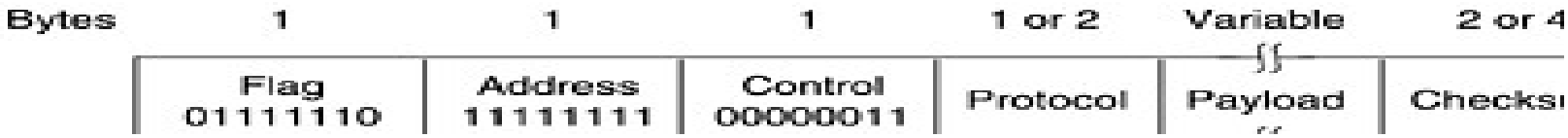
PPP Layer Function

In order to move data between any two nodes or routers, a data path must be established, and flow control procedures must be in place to ensure delivery of data.



- PPP provides three features:
  - A framing method that unambiguously delineates the end of one frame and the start of the next one. The frame format also handles error detection.
  - A link control protocol(LCP) for bringing lines up, testing them, negotiating options, and bringing them down again gracefully when they are no longer needed.
  - Network Control Protocol (NCP) to negotiate network-layer options in a way that is independent of the network layer protocol to be used.

Frame Format



All PPP frames begin with the standard HDLC flag byte (01111110), which is byte stuffed if it occurs within the payload field.

The Address field, which is always set to the binary value 11111111 to indicate that all stations are to accept the frame.

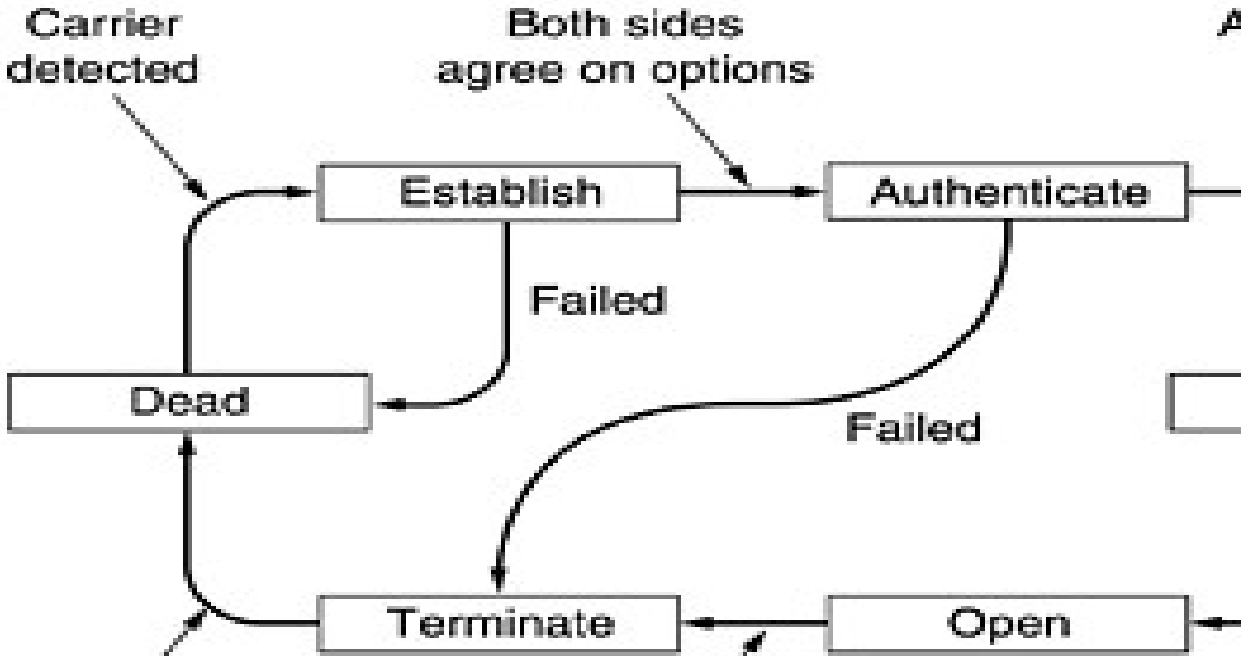
The Address field is followed by the Control field, the default value of which is 00000011. This value indicates an unnumbered frame. In other words, PPP does not provide reliable transmission using sequence numbers and acknowledgements as the default

The fourth PPP field is the Protocol field. Its job is to tell what kind of packet is in the Payload field. Protocols starting with a 0 bit are network layer protocols such as IP, IPX, OSI CLNP, XNS. Those starting with a 1 bit are used to negotiate other protocols.
These include LCP and a different NCP for each network layer protocol supported.
The default size of the Protocol field is 2 bytes, but it can be negotiated down to 1 byte using LCP.

The Payload field is variable length, up to some negotiated maximum. If the length is not negotiated using LCP during line setup, a default length of 1500 bytes is used.

After the Payload field comes the Checksum field, which is normally 2 bytes, but a 4-byte checksum can be negotiated.

A simplified phase diagram for bringing a line up and down



- The protocol starts with the line in the DEAD state, which means that no physical layer carrier is present and no physical layer connection exists.
- After physical connection is established, the line moves to ESTABLISH.
- At that point LCP option negotiation begins, which, if successful, leads to AUTHENTICATE. Now the two parties can check on each other's identities if desired.
- When the NETWORK phase is entered, the appropriate NCP protocol is invoked to configure the network layer.
- If the configuration is successful, OPEN is reached and data transport can take place.
- When data transport is finished, the line moves into the TERMINATE phase, and from there, back to DEAD when the carrier is dropped.

- LCP negotiates data link protocol options during the ESTABLISH phase. The LCP protocol is not actually concerned with the options themselves, but with the mechanism for negotiation. It provides a way for the initiating process to make a proposal and for the responding process to accept or reject it, in whole or in part. It also provides a way for the two processes to test the line quality to see if they consider it good enough to set up a connection. Finally, the LCP protocol also allows lines to be taken down when they are no longer needed.

The LCP frame types.

Name	Direction	Descriptio
Configure-request	I → R	List of proposed option
Configure-ack	I ← R	All options are accepte
Configure-nak	I ← R	Some options are not a
Configure-reject	I ← R	Some options are not r
Terminate-request	I → R	Request to shut the lin
Terminate-ack	I ← R	OK, line shut down
Code-reject	I ← R	Unknown request rece
Protocol-reject	I ← R	Unknown protocol requ
Echo-request	I → R	Please send this frame

- The four Configure- types allow the initiator (I) to propose option values and the responder (R) to accept or reject them. In the latter case, the responder can make an alternative proposal or announce that it is not willing to negotiate certain options at all. The options being negotiated and their proposed values are part of the LCP frames.

- The Terminate- codes shut a line down when it is no longer needed. The Code-reject and Protocol-reject codes indicate that the responder got something that it does not understand. This situation could mean that an undetected transmission error has occurred, but more likely it means that the initiator and responder are running different versions of the LCP protocol. The Echo- types are used to test the line quality. Finally, Discard-request help debugging. If either end is having trouble getting bits onto the wire, the programmer can use this type for testing. If it manages to get through, the receiver just throws it away, rather than taking some other action that might confuse the person doing the testing.

- The options that can be negotiated include setting the maximum payload size for data frames, enabling authentication and choosing a protocol to use, enabling line-quality monitoring during normal operation, and selecting various header compression options. There is little to say about the NCP protocols in a general way. Each one is specific to some network layer protocol and allows configuration requests to be made that are specific to that protocol. For IP, for example, dynamic address assignment is the most important possibility.