Chapter 3

The Data Link Layer

Prepared by :

Dr. Adel Soudani & Dr. Mznah Al-Rodhaan

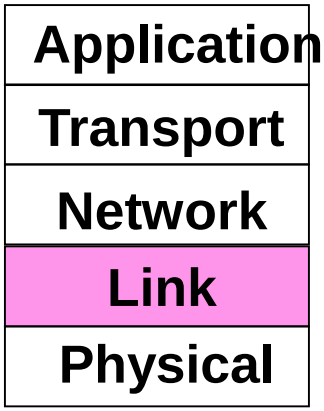
The Data Link Layer

- Data Link Layer Design Issues
- Error Detection and Correction
- Elementary Data Link Protocols
- Sliding Window Protocols
- Example Data Link Protocols

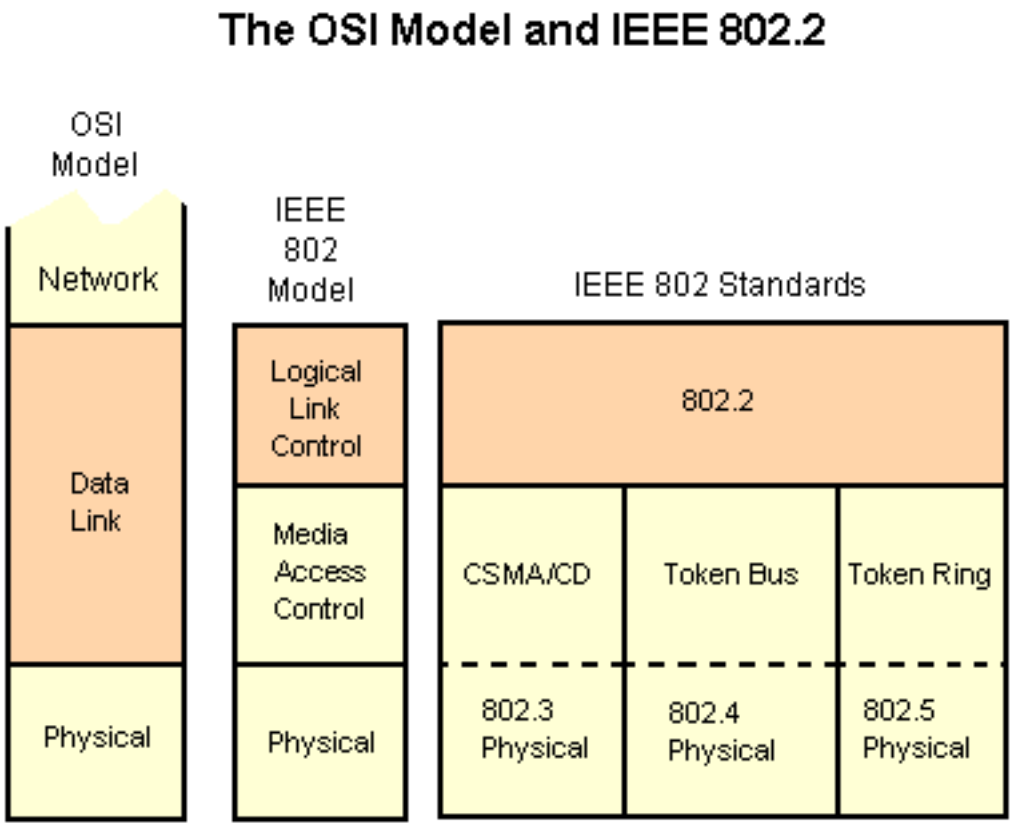
The Data Link Layer

Responsible for delivering frames of information over a single link

- Handles transmission errors and regulates the flow of data



The Data Link Layer



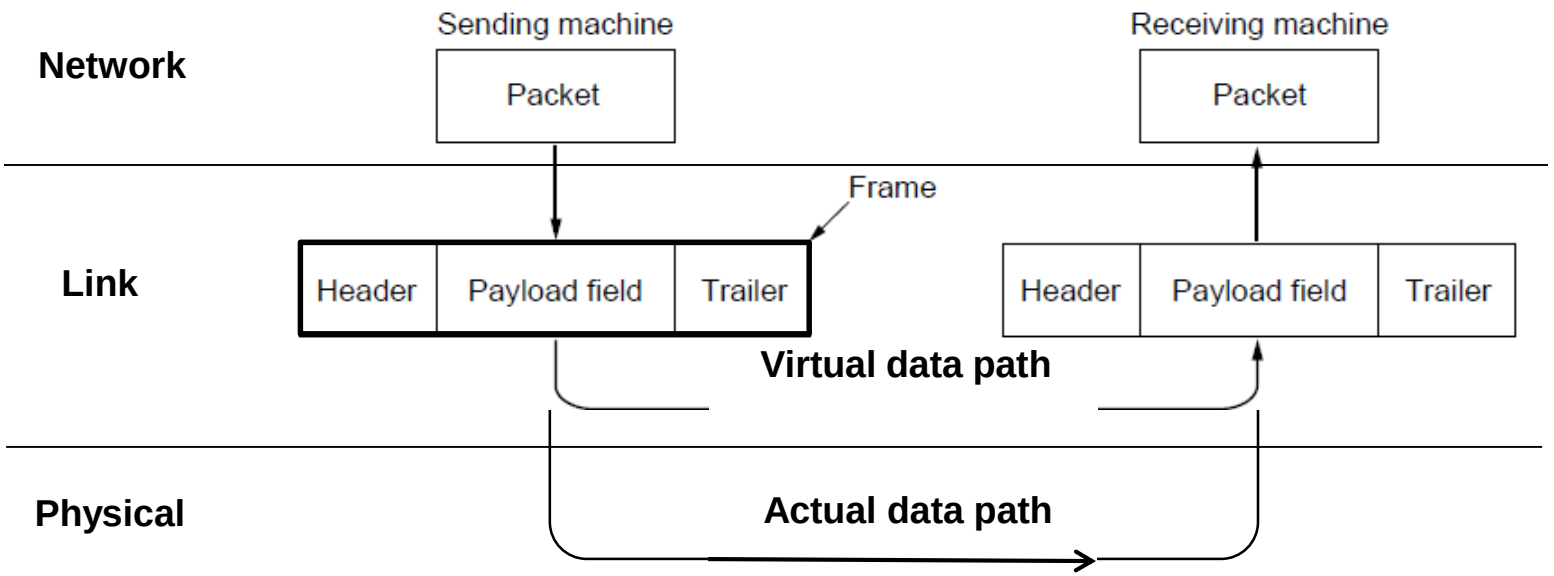
Data Link Layer Design Issues

- Frames »
- Possible services »
- Framing methods »
- Error control »
- Flow control »

Framing

Frames

Link layer accepts packets from the network layer, and encapsulates them into frames that it sends using the physical layer; reception is the opposite process



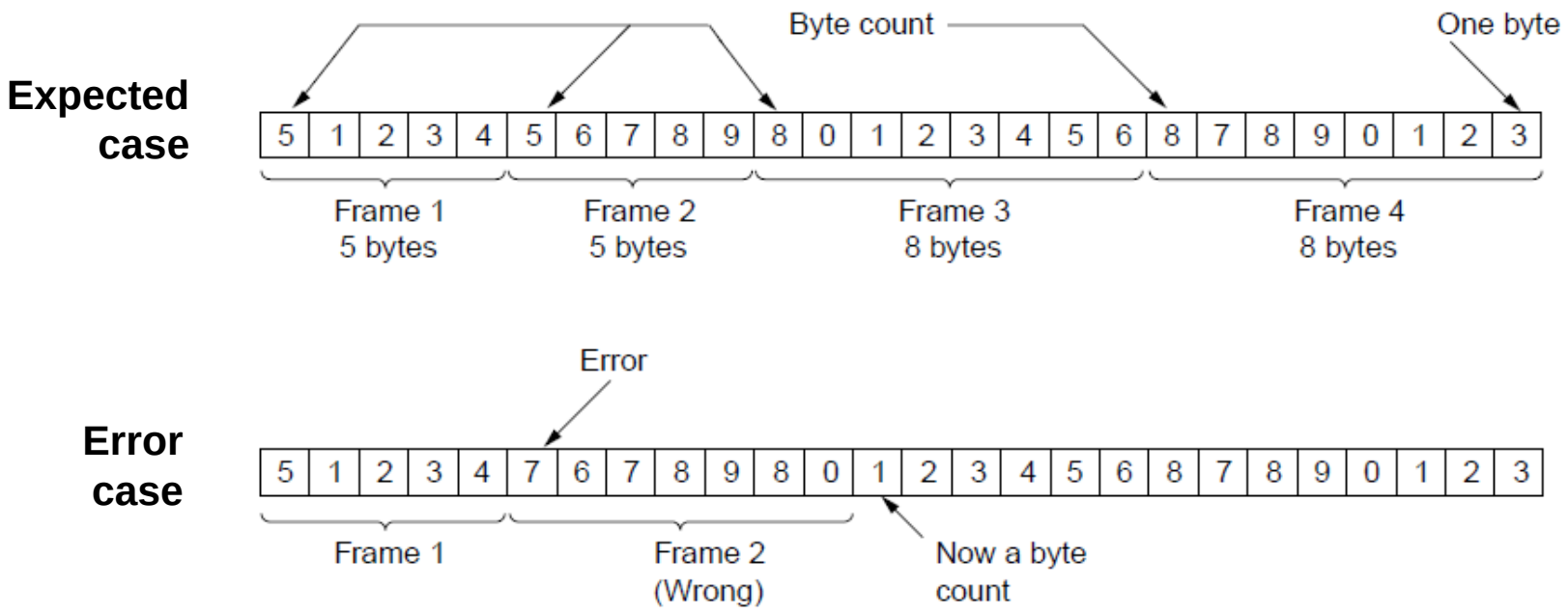
Framing Methods

- Byte count »
- Flag bytes with byte stuffing »
- Flag bits with bit stuffing »
- Physical layer coding violations
  - Use non-data symbol to indicate frame

Framing – Byte count

Frame begins with a count of the number of bytes in it

- Simple, but difficult to resynchronize after an error

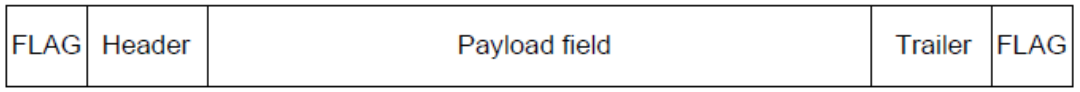


Framing – Byte stuffing

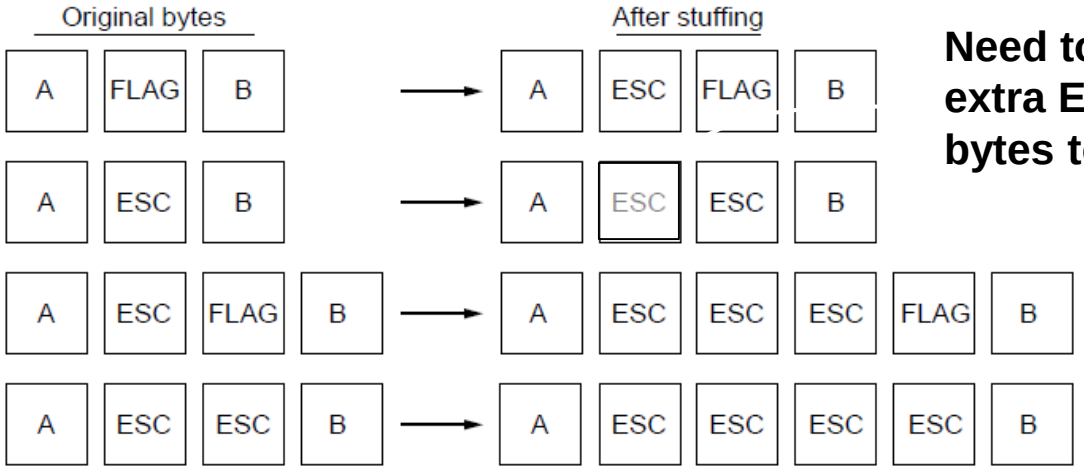
Special flag bytes delimit frames; occurrences of flags in the data must be stuffed (escaped)

- Longer, but easy to resynchronize after error

Frame format



Stuffing examples

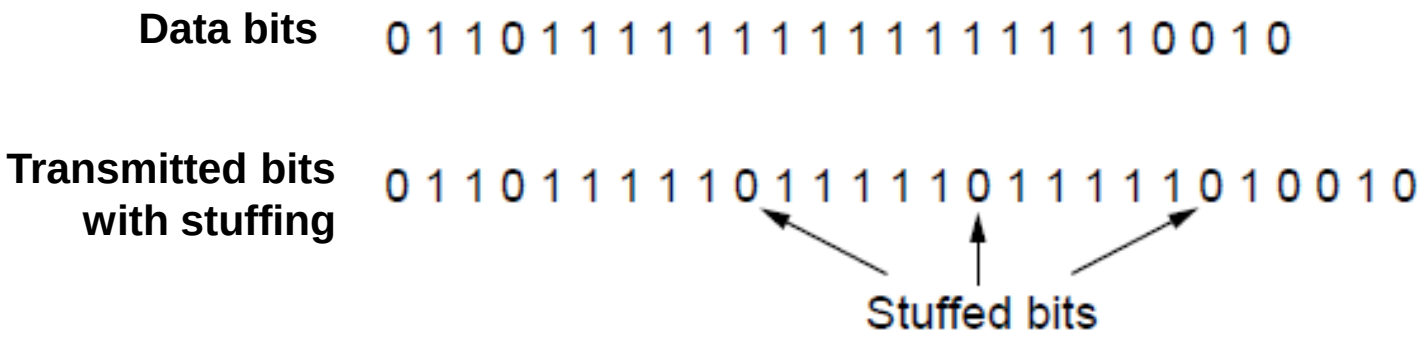


Need to escape extra ESCAPE bytes too!

Framing – Bit stuffing

Stuffing done at the bit level:

- Frame flag has six consecutive 1s (01111110)
- On transmit, after five 1s in the data, a 0 is added
- On receive, a 0 after five 1s is deleted



Error Control

Error control repairs frames that are received in error

- Requires errors to be detected at the receiver
- Typically retransmit the unacknowledged frames
- Timer protects against lost acknowledgements

Detecting errors and retransmissions are next topics.

Flow Control

Prevents a fast sender from out-pacing a slow receiver

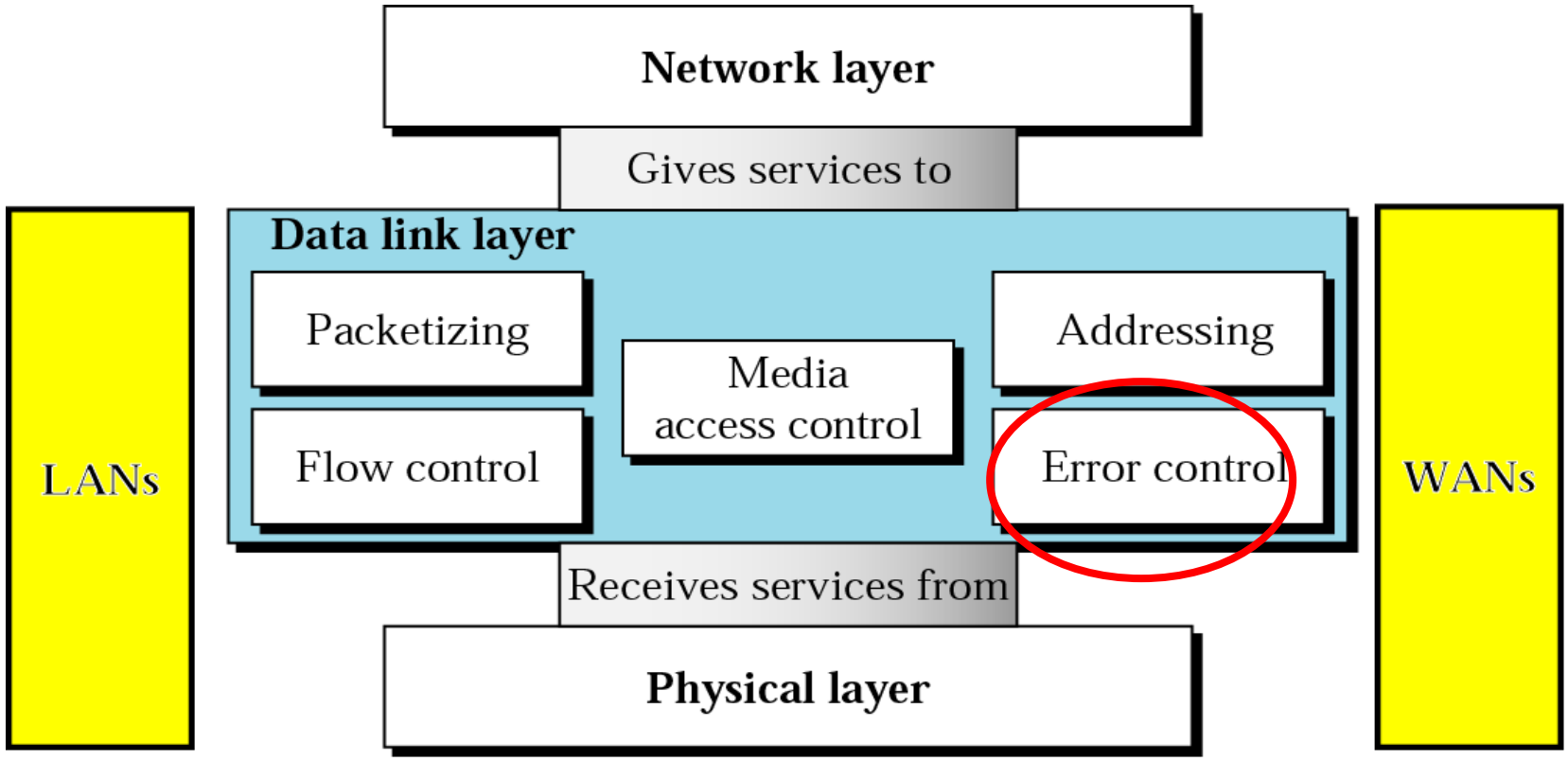
- Receiver gives feedback on the data it can accept
- Rare in the Link layer as NICs run at "wire speed"
  - Receiver can take data as fast as it can be sent

Flow control is a topic in the Data Link and Transport layers.

Lecture 2

Error Control

Data Link Layer



3.1 INTRODUCTION

Topics discussed in this section:

Types of Errors

Redundancy

Detection Versus Correction

Detection

- **Parity Check**
- **Cyclic Redundancy Check (CRC)**
- **Checksum**

Corrections

- **Retransmission**
- **Forward Error Correction**
- **Burst Error Correction**

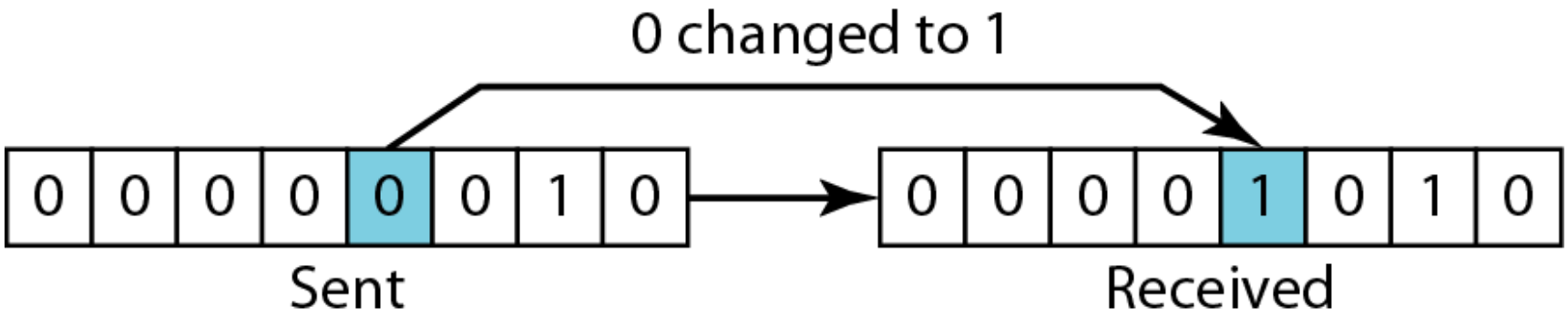
Data can be corrupted during transmission.

Some applications require that errors be detected and corrected.

Types of Errors

1. Single-bit error

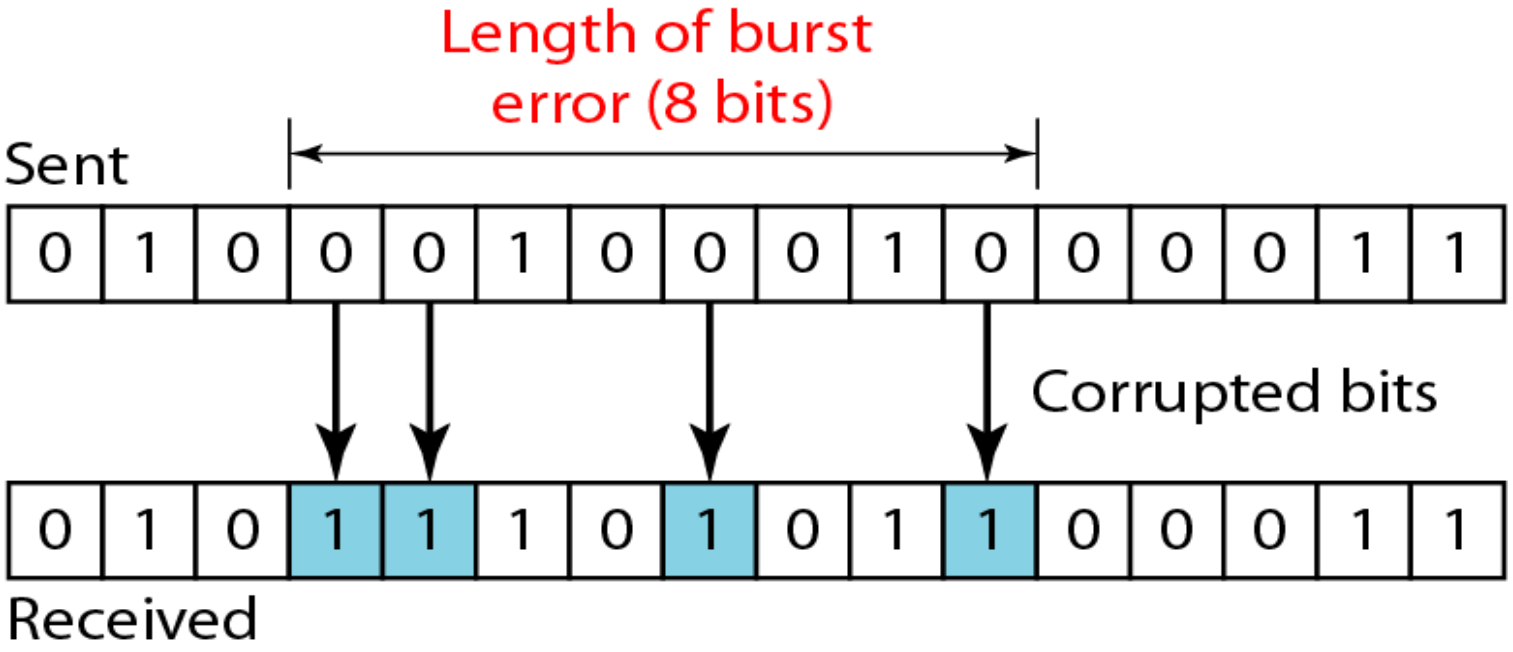
In a single-bit error, only 1 bit in the data unit has changed.



Types of Errors

2. Burst error

A burst error means that 2 or more bits in the data unit have changed.

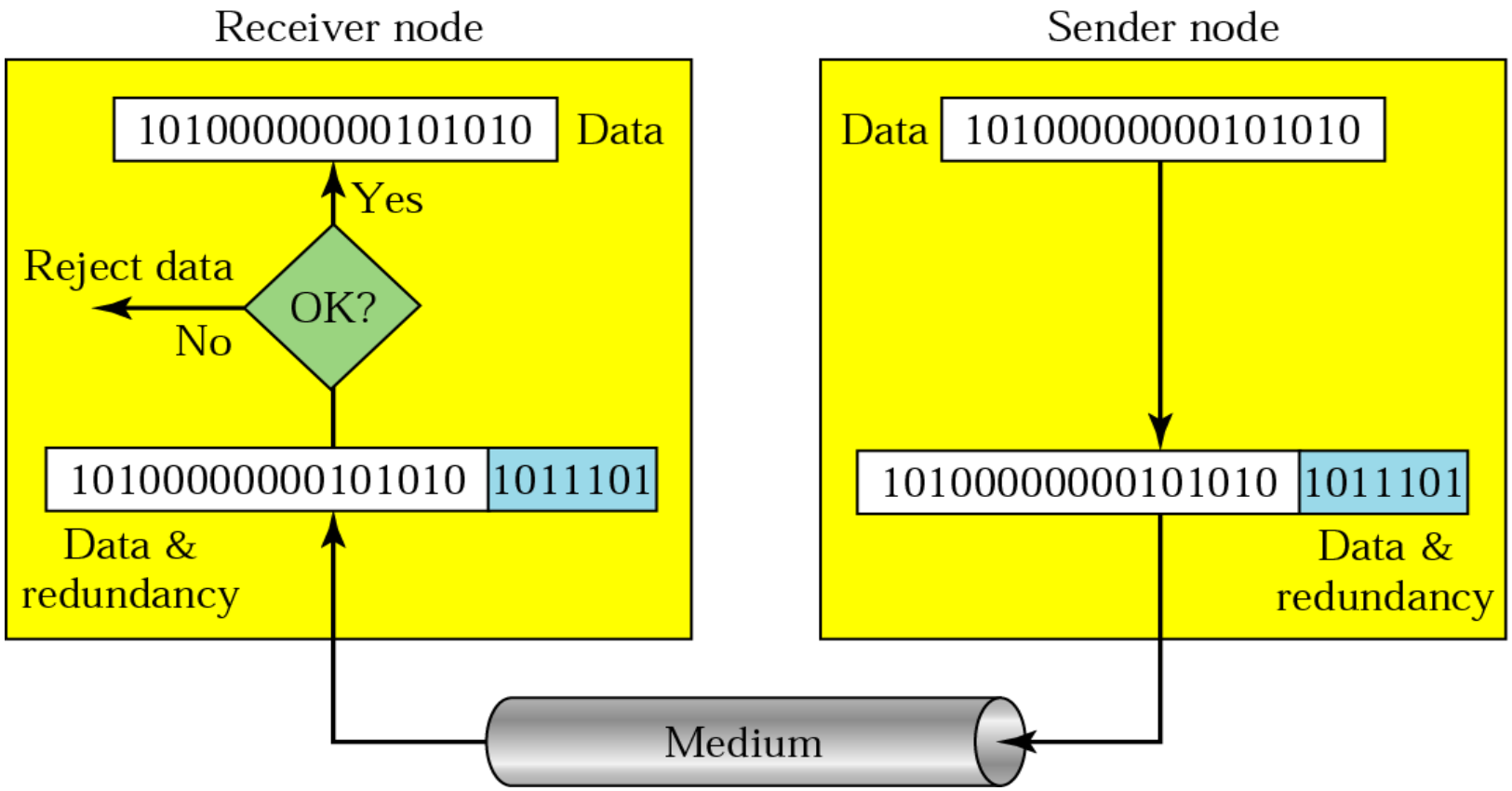


Redundancy

Error detection uses the concept of redundancy, which means adding extra bits for detecting errors at the destination.

To detect or correct errors, we need to send extra (redundant) bits with data.

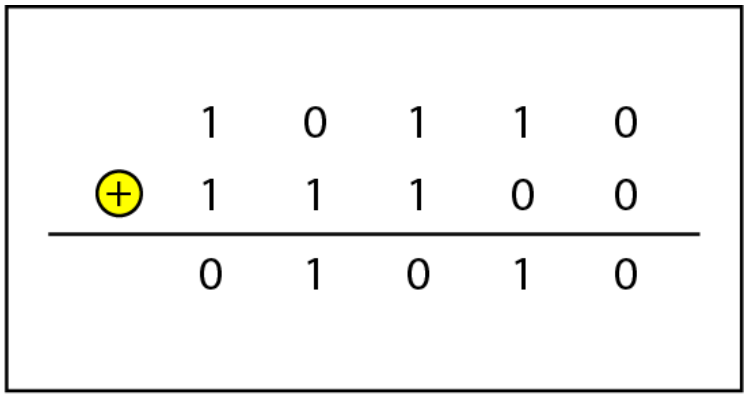
Redundancy


Receiver node
Sender node
1010000000101010
Data
Yes
Reject data
OK?
No
1010000000101010
1011101
Data &
redundancy
Data
1010000000101010
1010000000101010
1011101
Data &
redundancy
Medium

XORing of two single bits or two words

a. Two bits are the same, the result is 0.

b. Two bits are different, the result is 1.



c. Result of XORing two patterns

3.2 Block Coding

We divide the message into blocks, each of m bits, called datawords. We add r redundant bits to each block to make the length

$n = m + r$.

The resulting n-bit blocks are called codewords.

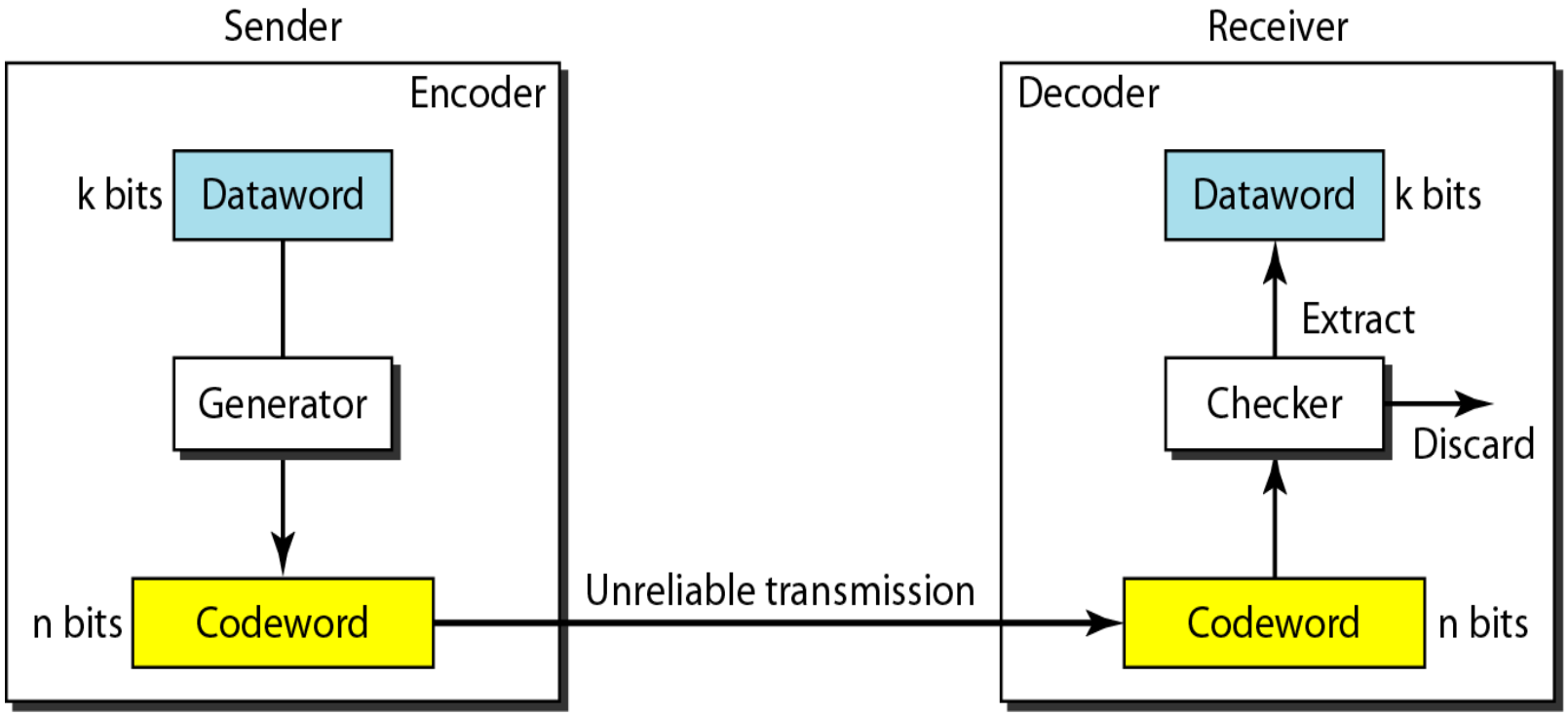
$2^r \geq m+r+1$

Topics discussed in this section:

Error Detection

Error Correction

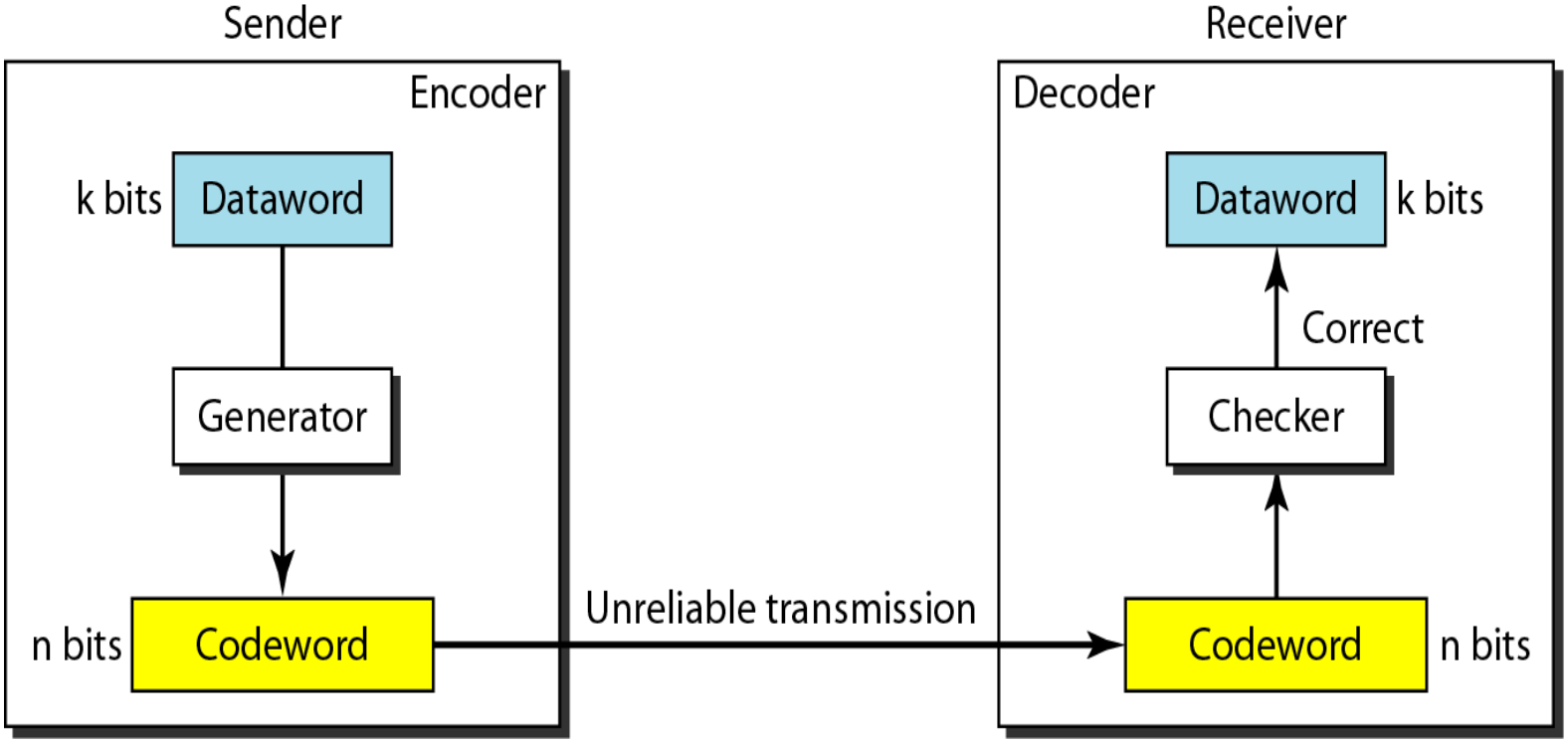
Error Detection



Solution

Retransmission

Error Correction

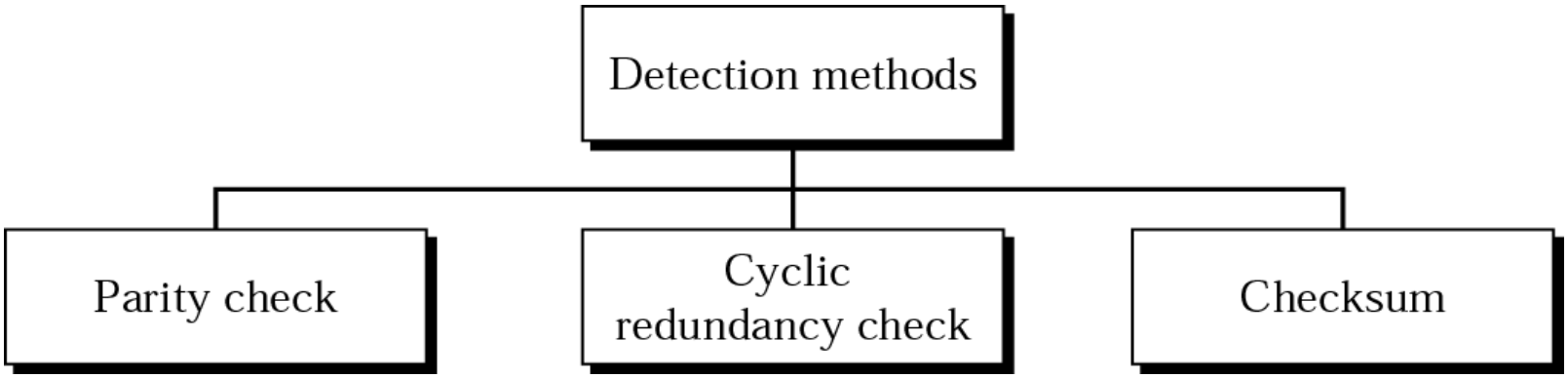


Solution

1. **Forward Error Correction**
2. **Burst Error Correction**

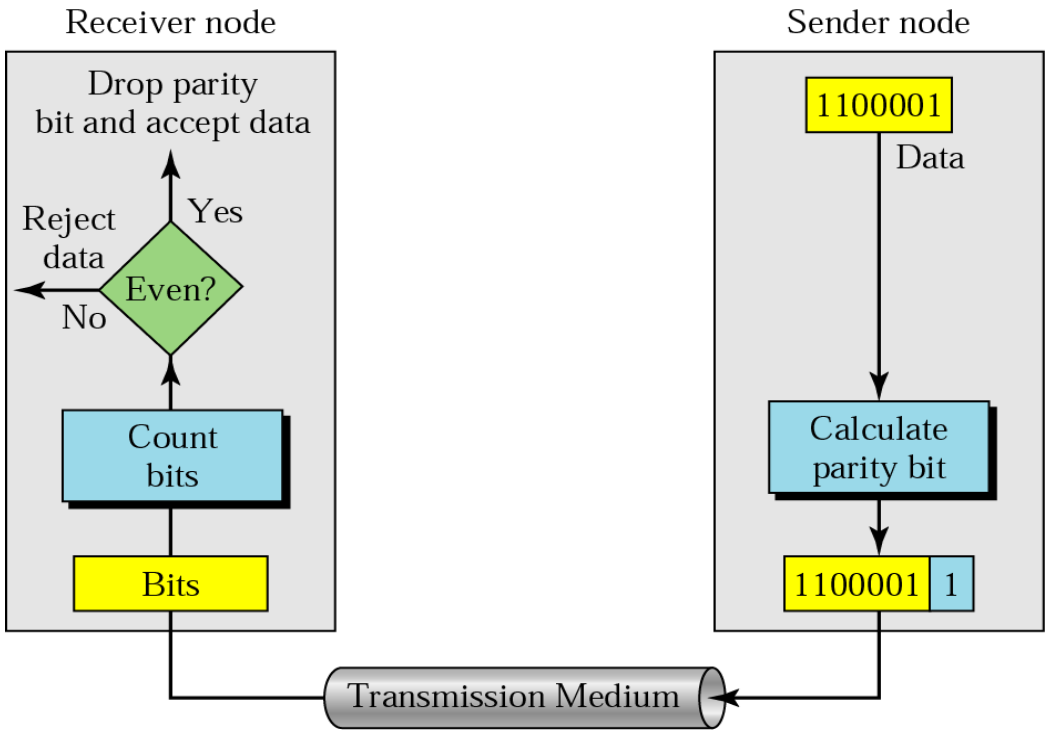
3.3 Linear Block Coding

Error Detection



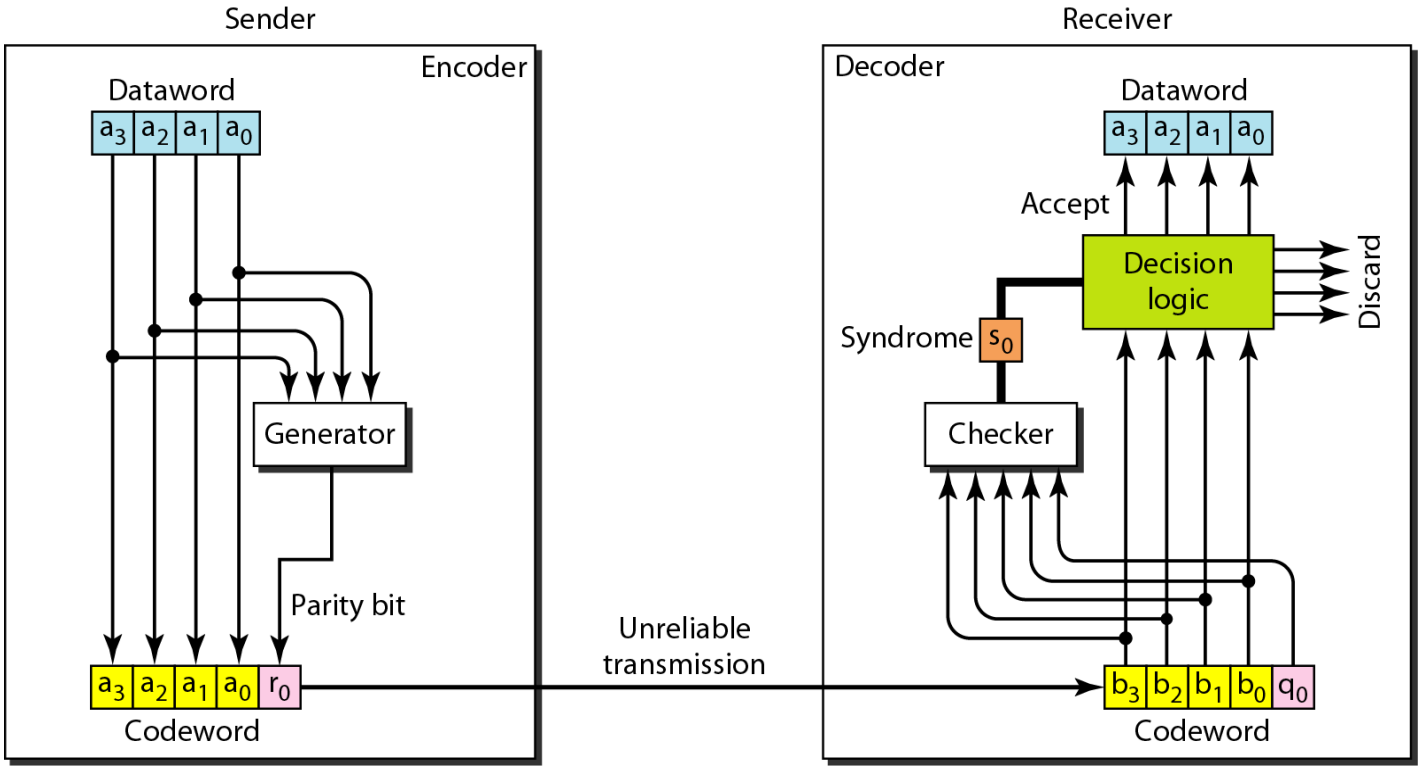
Simple Parity Check code

Even-parity concept



Odd-parity concept ?

Parity Check

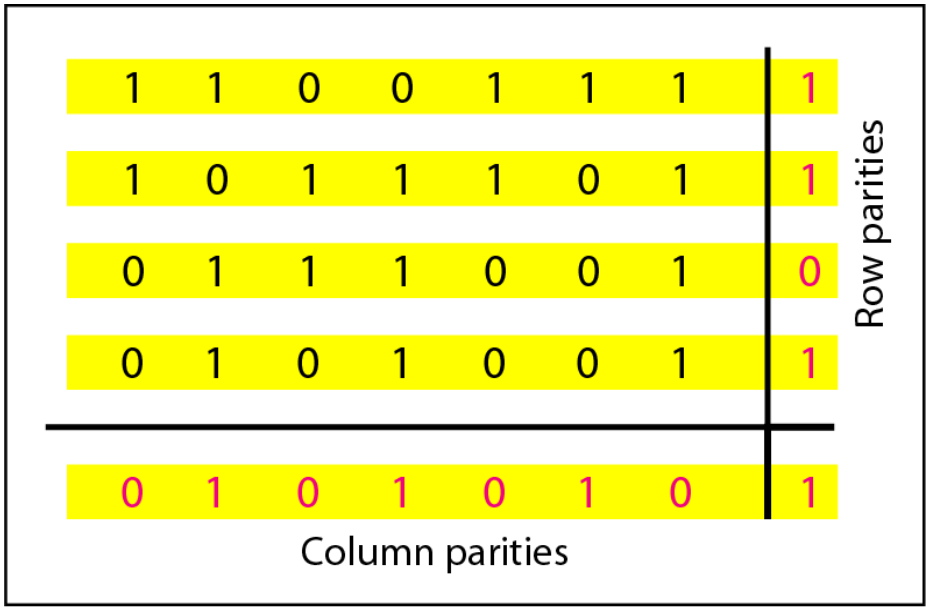


Simple parity check can detect all single-bit errors.

It can detect burst errors only if the total number of errors in each data unit is odd for even-parity.

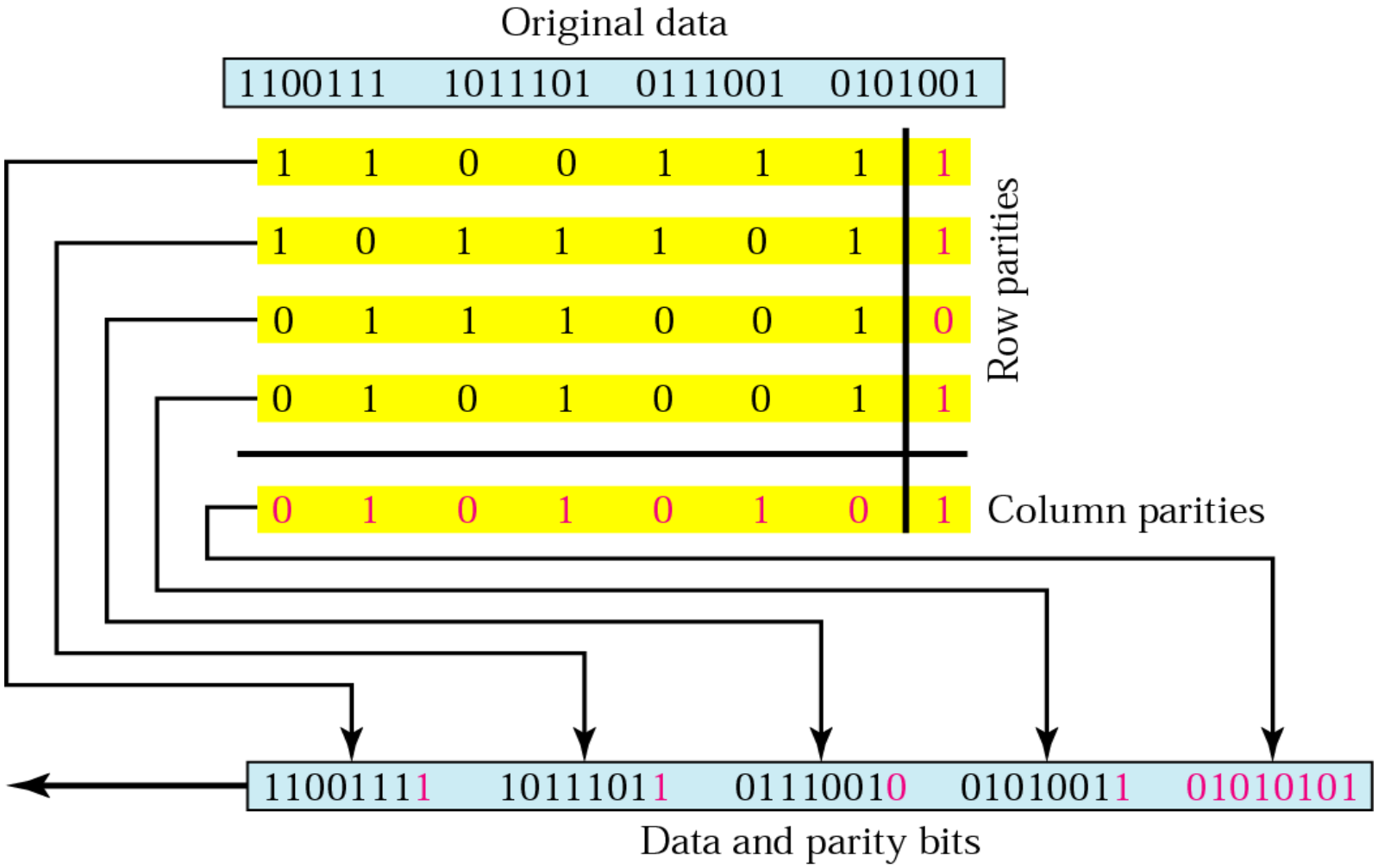
In two-dimensional parity check, a block of bits is divided into rows and a redundant row of bits is added to the whole block.

Two-dimensional parity-check code



a. Design of row and column parities

Two-dimensional parity



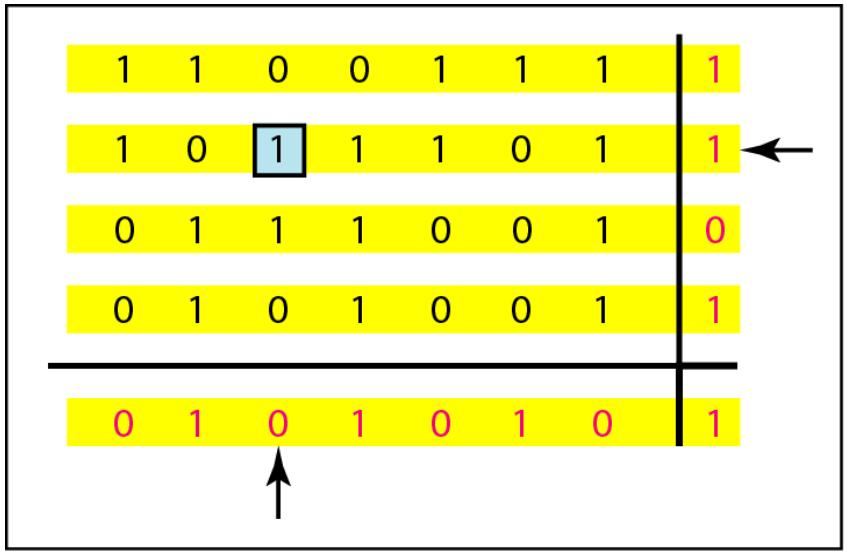
Sent

1	**0**	**1**	**0**	**1**	**0**	**0**	**1**
0	0	1	1	1	0	0	1
1	1	0	1	1	1	0	1
1	1	1	0	0	1	1	1
1	0	1	0	1	0	1	0

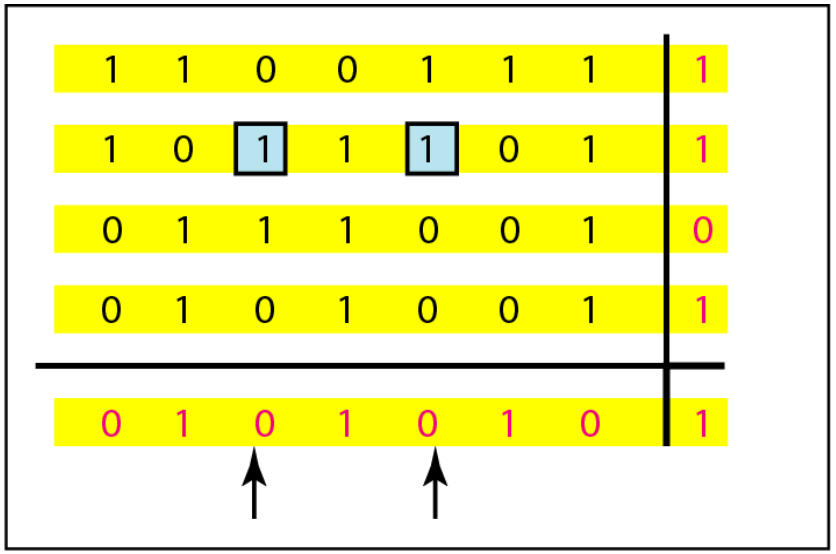
Received

1	**0**	**1**	**0**	**1**	**0**	**0**	**1**	**0**
0	0	1	1	1	0	0	1	0
1	1	0	1	1	1	0	1	0
1	1	1	0	0	1	1	1	0
1	0	1	0	1	0	1	0	0
0	0	0	0	0	0	0	0	0

Two-dimensional parity-check code

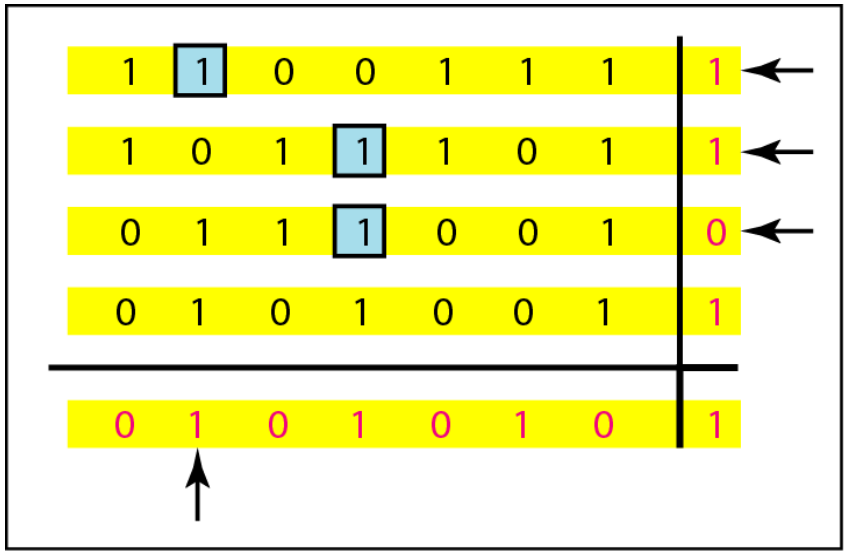


b. One error affects two parities

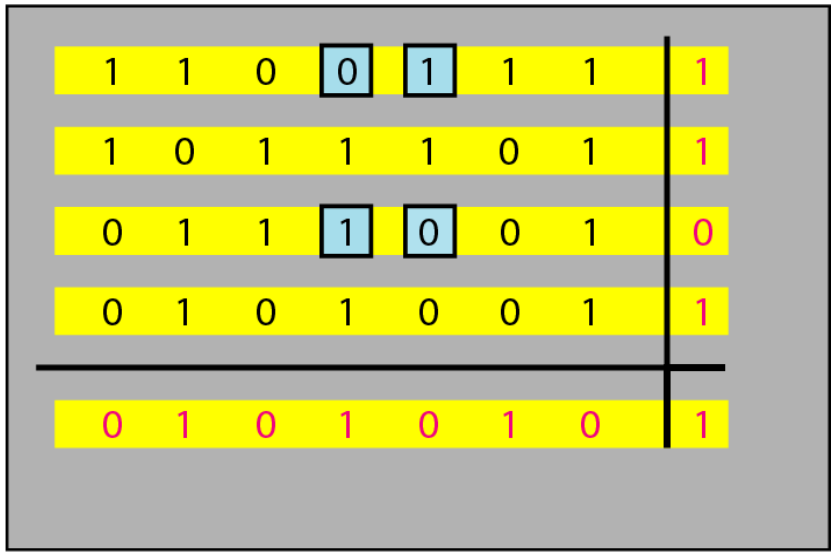


c. Two errors affect two parities

Two-dimensional parity-check code



d. Three errors affect four parities



e. Four errors cannot be detected

Lecture 3

Error Detection and Correction

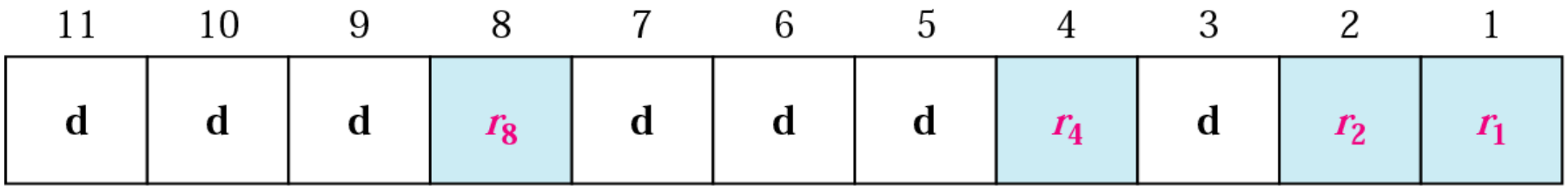
Hamming Code

1. Hamming codes provide for FEC using a “Block Parity”
   i.e, instead of one parity bit send a block of parity bits
2. Allows correction of single bit errors
3. This is accomplished by using more than one parity bit
4. Each computed on different combination of bits in the data

Note: Study the Hamming code from the slides.

Redundancy Bits Calculation

Parity Bits

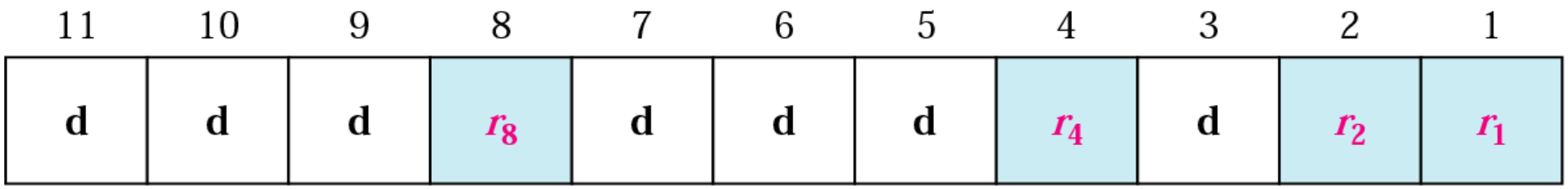


Decimal	Binary
3	0011
5	0101
6	0110
7	0111
9	1001
10	1010
11	1011

Uses four (4) parity bits

- Bit Position 1 (r_1): Bits 3, 5, 7, 9, 11
- Bit Position 2 (r_2): Bits 3, 6, 7, 10, 11
- Bit Position 4 (r_3): Bits 5, 6, 7
- Bit Position 8 (r_4): Bits 9, 10, 11

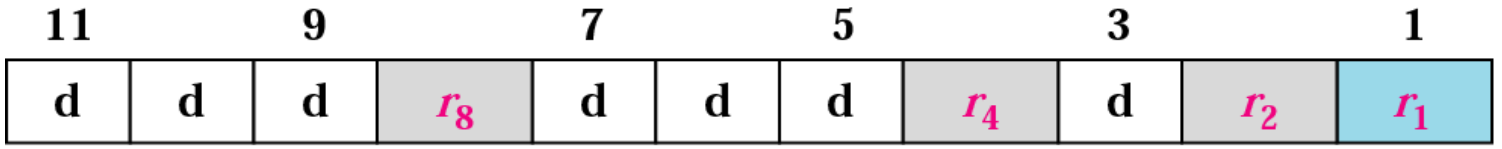
Redundancy Bits Calculation



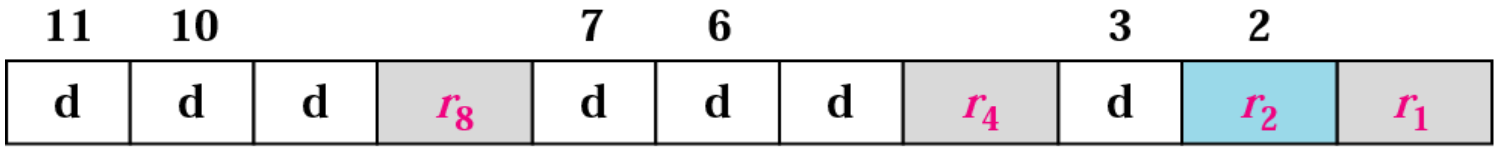
- Position 1: check 1 bit, skip 1 bit, ..etc. (1,3,5,7,9,11,13,15,...)
- Position 2: check 2 bits, skip 2 bits, ..etc. (2,3,6,7,10,11,14,15,...)
- Position 4: check 4 bits, skip 4 bits, ..etc. (4,5,6,7,12,13,14,15,20,21,22,23,...)
- Position 8: check 8 bits, skip 8 bits..etc. (8-15,24-31,40-47,...)
- Position 16: check 16 bits, skip 16 bits, ..etc. (16-31,48-63,80-95,...)

Redundancy Bits Calculation

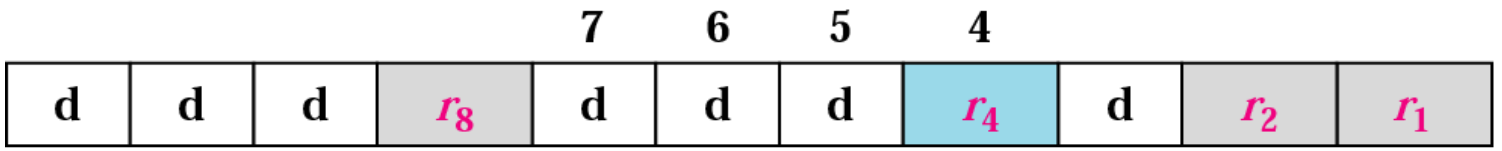
r_1 will take care of these bits.



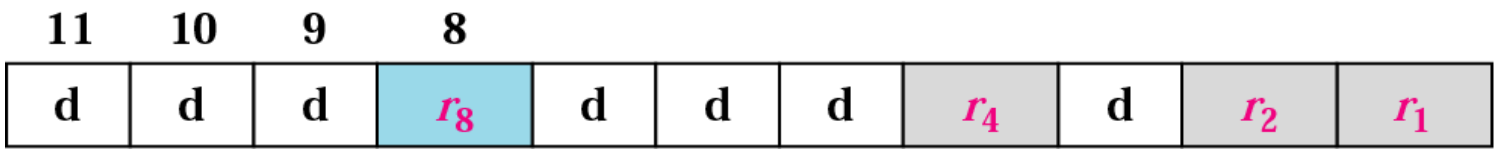
r_2 will take care of these bits.



r_4 will take care of these bits.

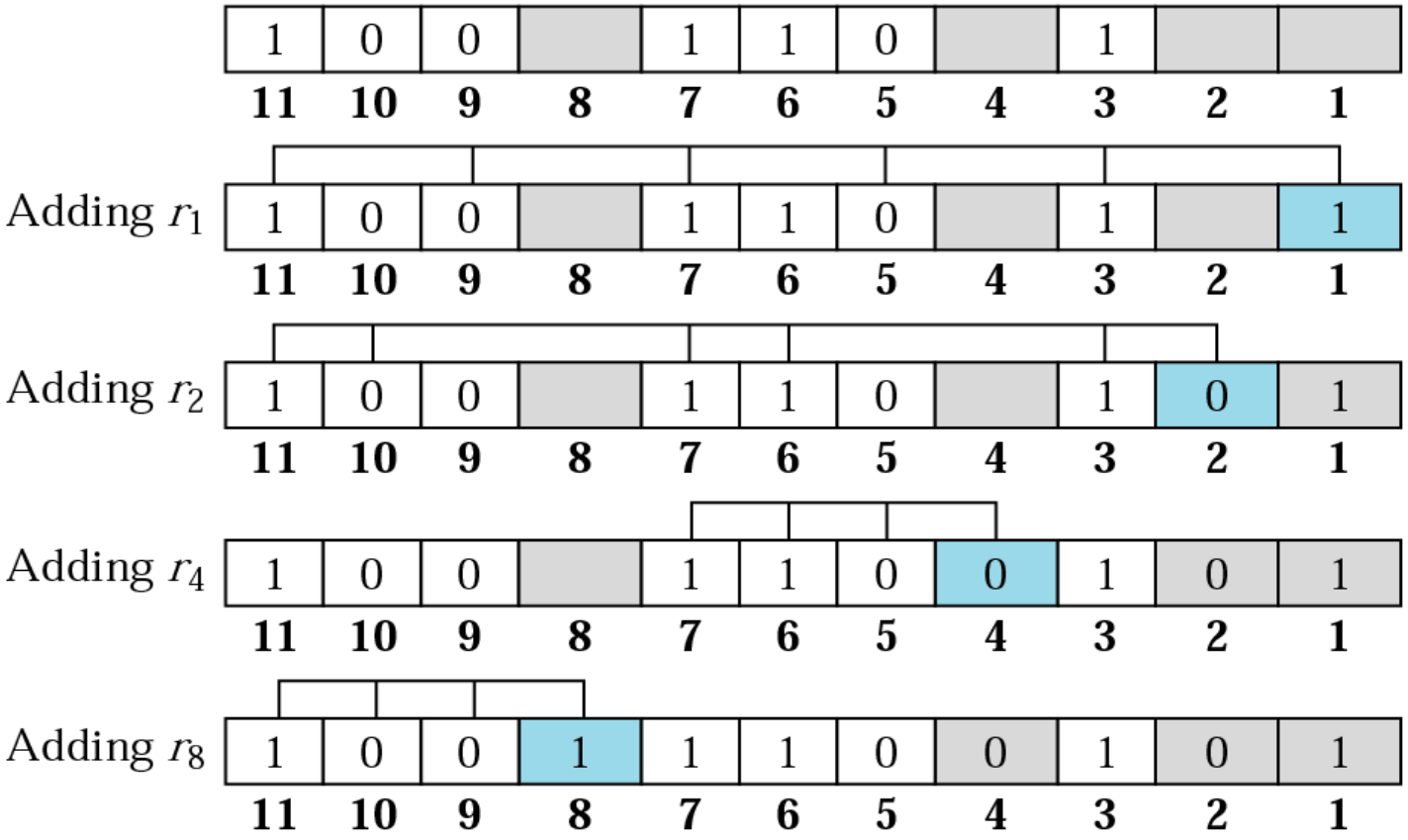


r_8 will take care of these bits.



Example

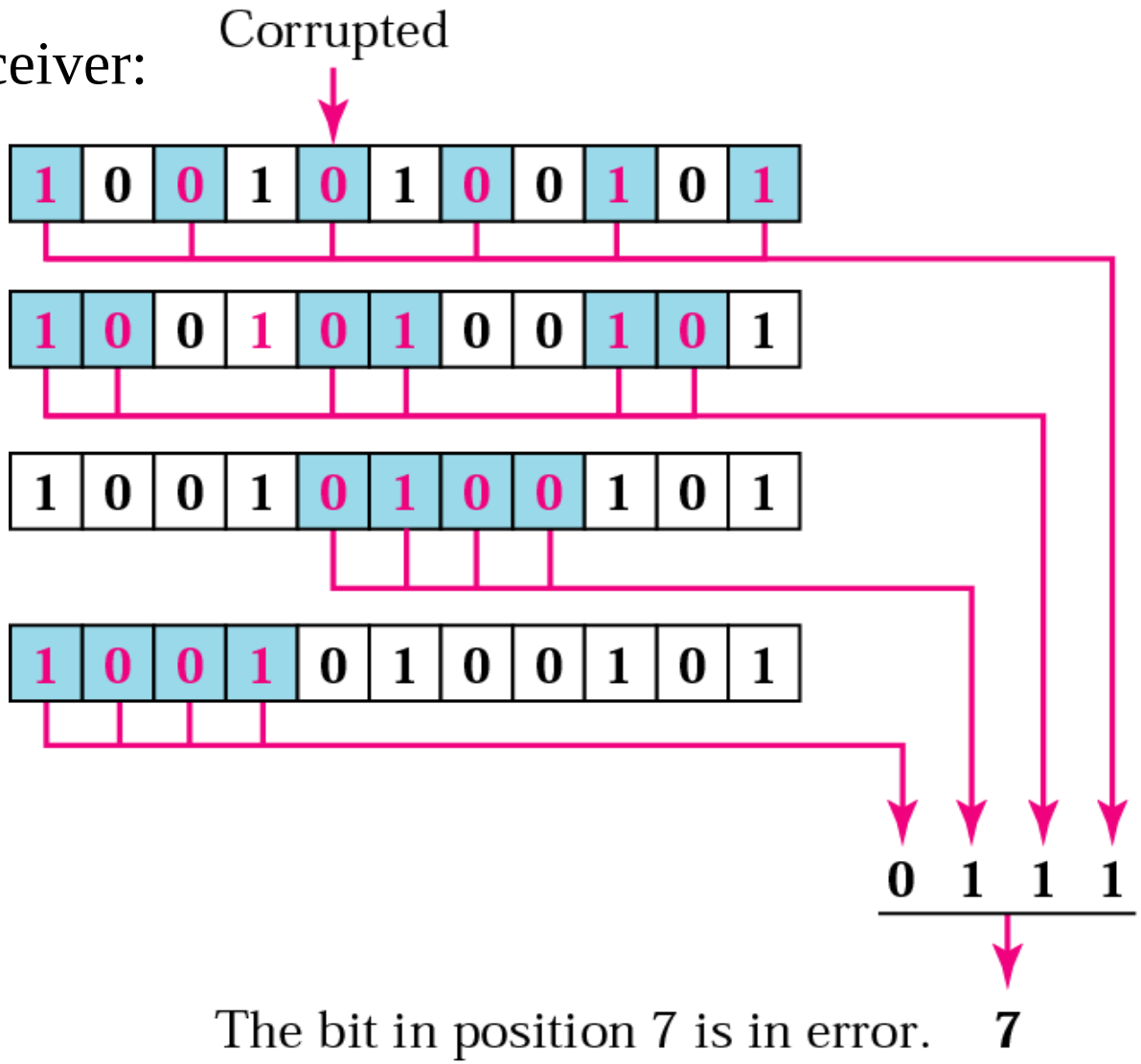
Redundancy bits calculation using even-parity (at the sender)



So the sender will send 10011100101

Error Correction using Hamming Code

At the receiver:

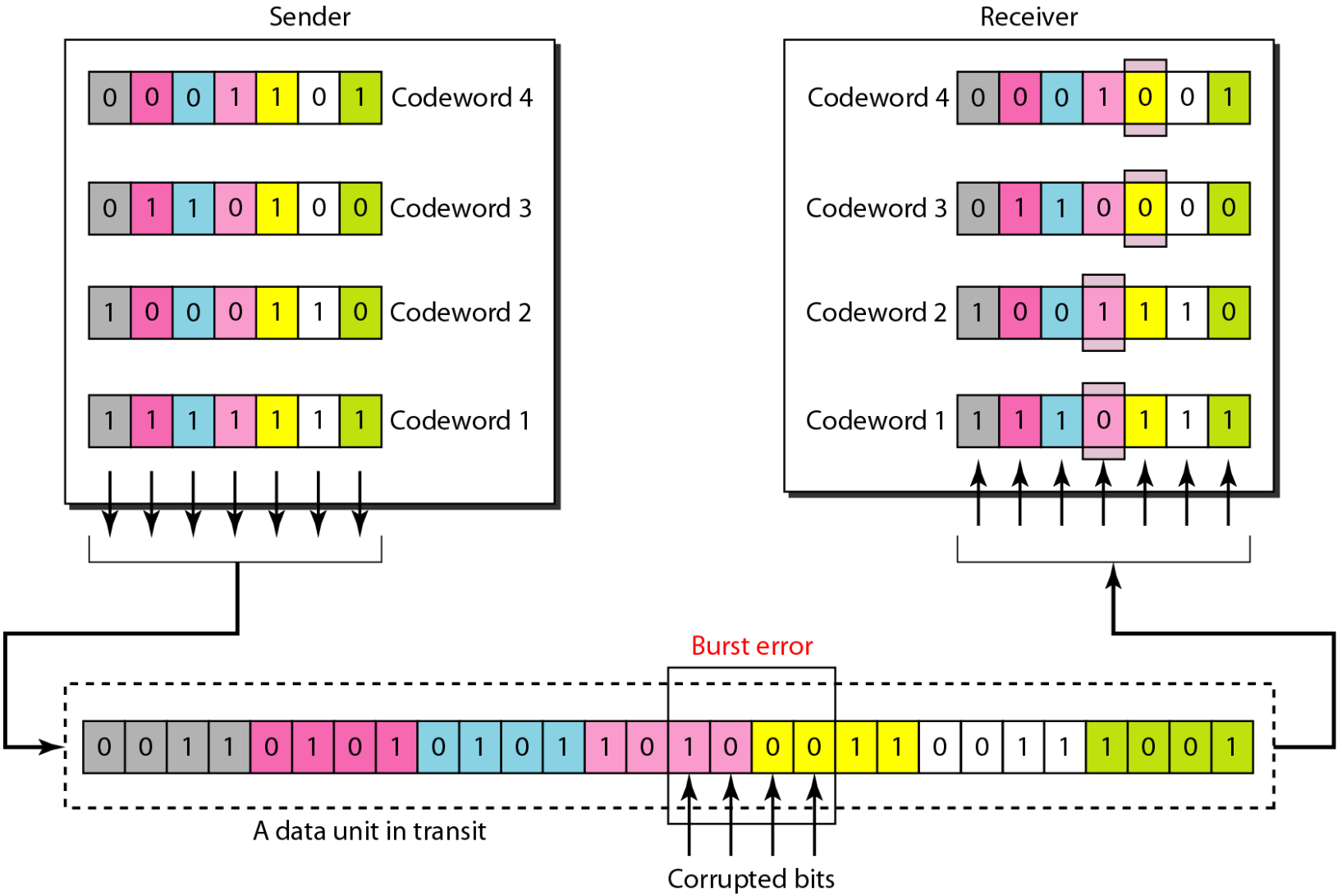


So the corrected data will be 10011100101

Error Correction using Hamming Code

Hamming code Increases overhead in both data transmitted and processing time

Burst error correction using Hamming code



Lecture 4

Cyclic codes

3.4 CYCLIC CODES

In a cyclic code, if a codeword is cyclically shifted (rotated), the result is another codeword.

Cyclic codes

- **Cyclic Redundancy Check**
- **Checksum**

Topics discussed in this section:

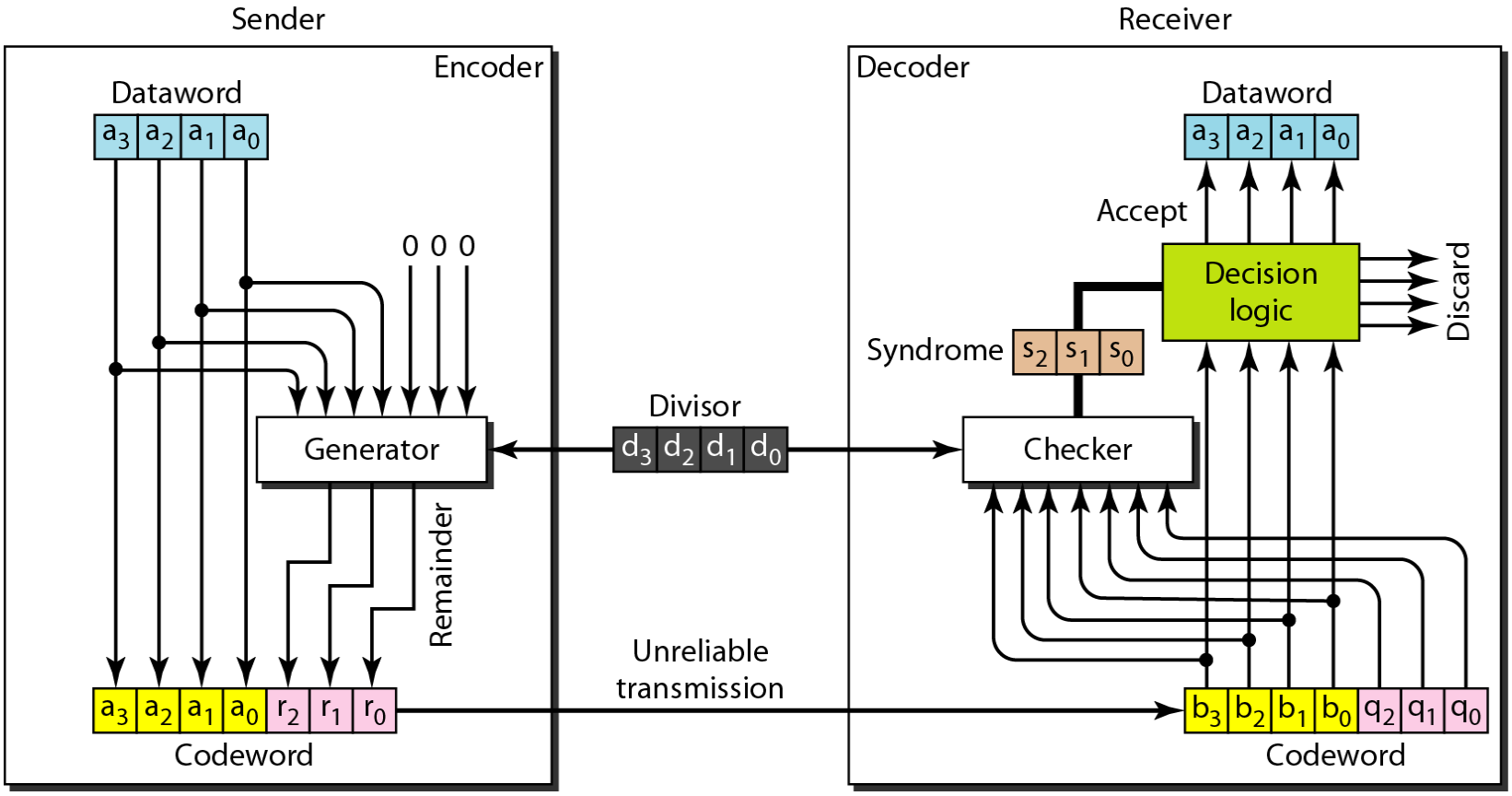
Cyclic Redundancy Check

Generator and Checker

Polynomials

Checksum

CRC generator and checker



A CRC code with C(7, 4)

Dataword	*Codeword*	*Dataword*	*Codeword*
0000	0000000	1000	1000101
0001	0001011	1001	1001110
0010	0010110	1010	1010011
0011	0011101	1011	1011000
0100	0100111	1100	1100010
0101	0101100	1101	1101001
0110	0110001	1110	1110100
0111	0111010	1111	1111111

Binary Division

- **Find the quotient and remainder when 1111101 is divided by 1101 in modulo 2 arithmetic.**
  - As with traditional division, we note that the dividend is divisible once by the divisor.
  - We place the divisor under the dividend and perform modulo 2 subtraction.

```
        1
   ------------
1101)1111101
     1101
     ----
     0010
```

Binary Division

- **Find the quotient and remainder when 1111101 is divided by 1101 in modulo 2 arithmetic…**
  - Now we bring down the next bit of the dividend.
  - We see that 00101 is not divisible by 1101. So we place a zero in the quotient.

```
          10
     ------
1101)1111101
     1101
     ----
     00101
      0000
```

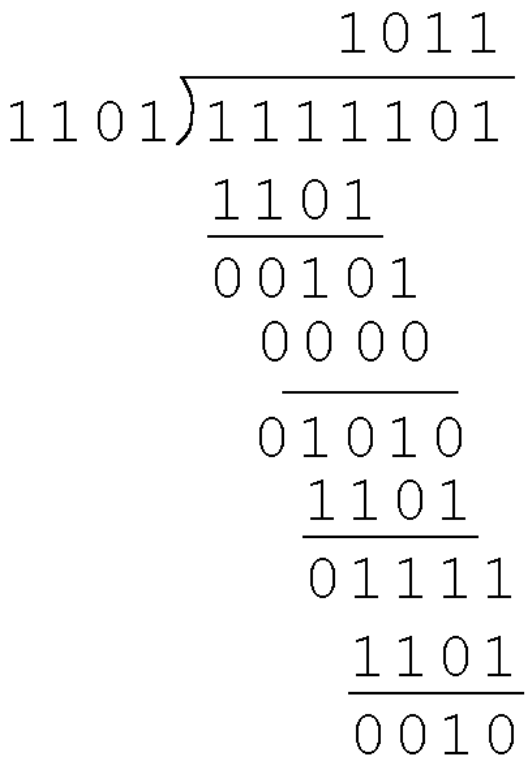
Binary Division

- **Find the quotient and remainder when 1111101 is divided by 1101 in modulo 2 arithmetic…**
  - 1010 is divisible by 1101 in modulo 2.
  - We perform the modulo 2 subtraction.

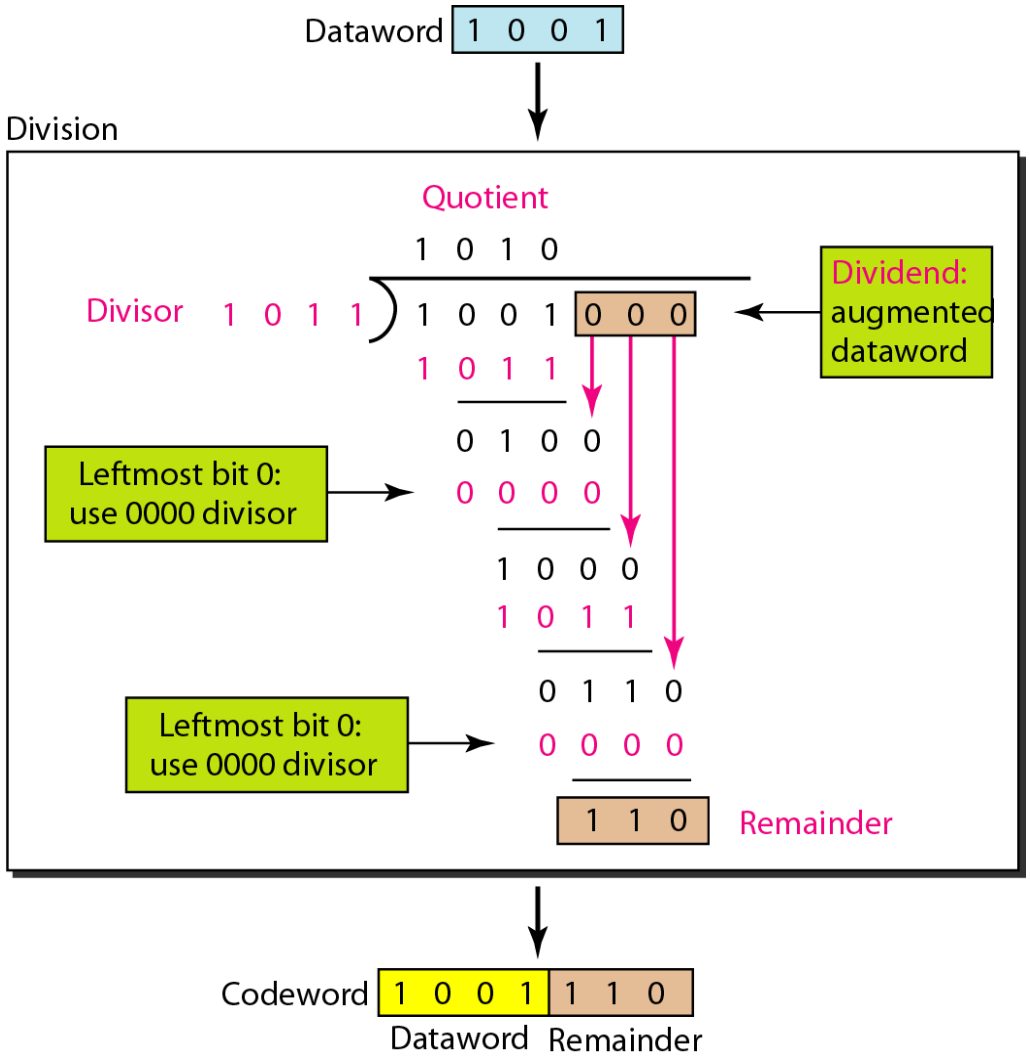
```
          101
      ------------
1101 ) 1111101
       1101
       -----
       00101
         0000
         ----
         01010
          1101
          ----
          0111
```

Binary Division

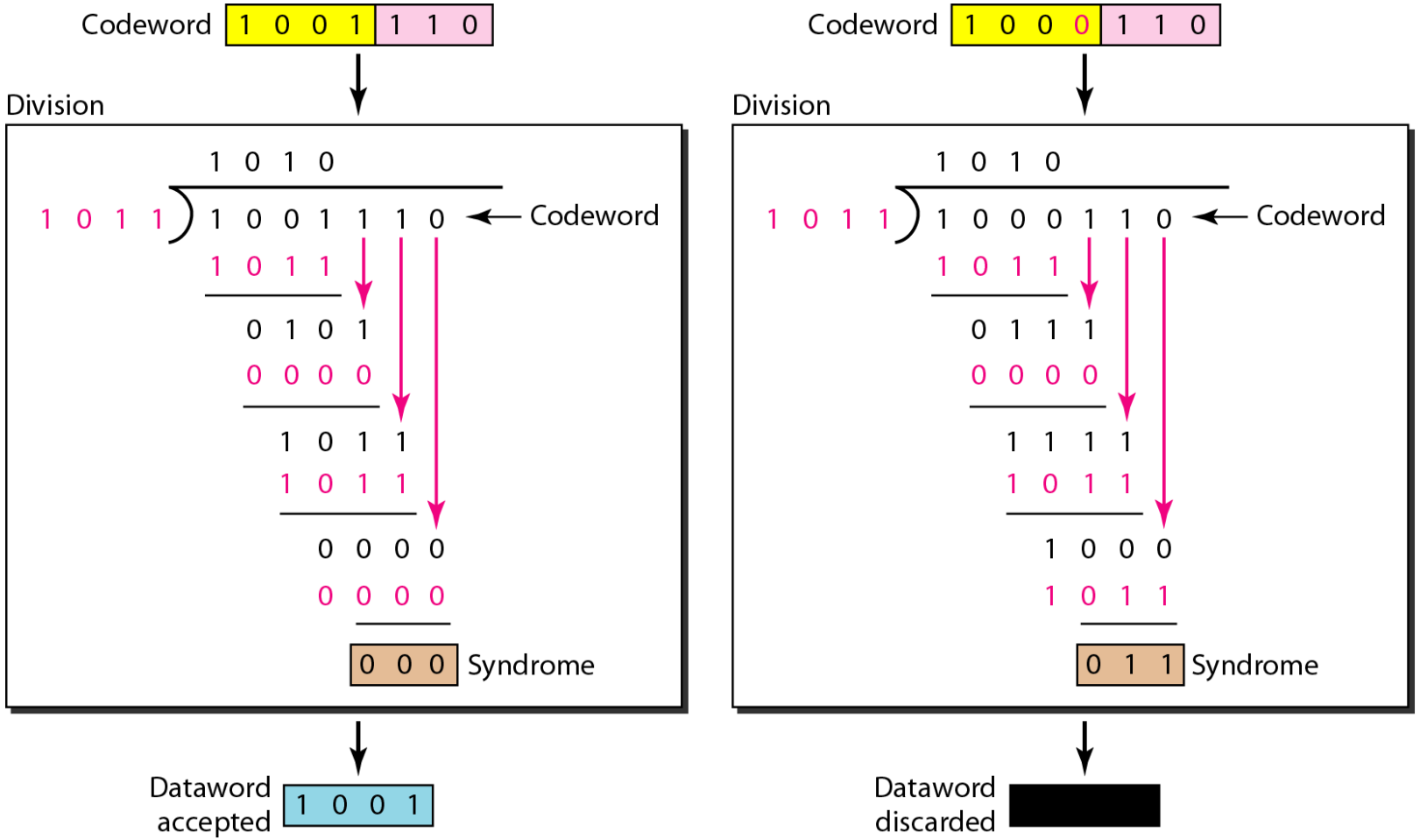
- **Find the quotient and remainder when 1111101 is divided by 1101 in modulo 2 arithmetic…**
  - We find the quotient is 1011, and the remainder is 0010.
- **This procedure is very useful to us in calculating CRC syndromes.**



Division in CRC encoder



Division in the CRC decoder for two cases



At the sender side

- **Suppose we want to transmit the information string: 1111101.**
- **The receiver and sender decide to use the polynomial pattern, 1101.**
- **The information string is shifted left by one position less than the number of positions in the divisor.**
- **The remainder is found through modulo 2 division (at right) and added to the information string: 1111101000 + 111 = 1111101111.**

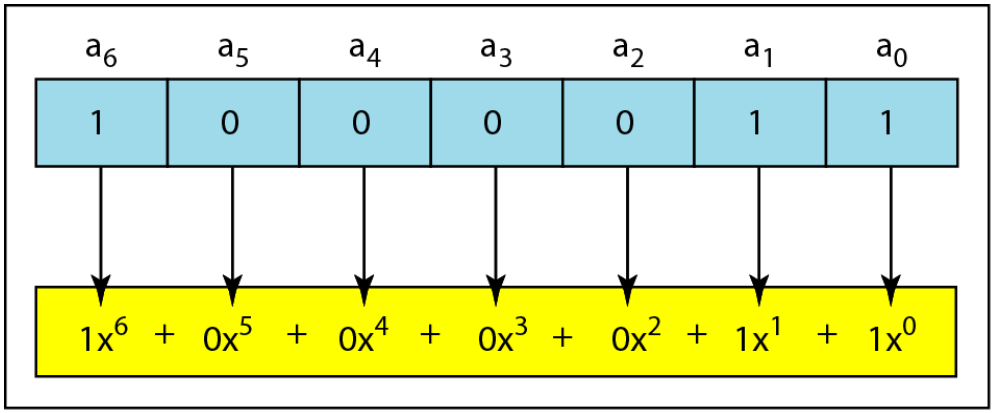
```
          1011011
     ------------
1101 ) 1111101000
       1101
       ----
       001010
          1101
          ----
          01111
            1101
            ----
            001000
               1101
               ----
               01010
                 1101
                 ----
                 0111
```

At the receiver side

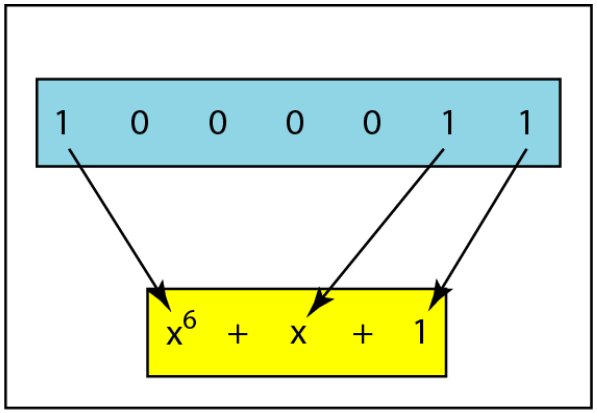
- **If no bits are lost or corrupted, dividing the received information string by the agreed upon pattern will give a remainder of zero.**
- **We see this is so in the calculation at the right.**
- **Real applications use longer polynomials to cover larger information strings.**

```
          1011011
      ------------
1101 ) 1111101111
       1101
       ----
       001010
          1101
          ----
          01111
            1101
            ----
            001011
               1101
               ----
               01101
                 1101
                 ----
                 0000
```

A polynomial to represent a binary word



a. Binary pattern and polynomial



b. Short form

Note:

The divisor 1101 corresponds to the polynomial: $X^3 + X^2 + 1.$

The divisor 1011 corresponds to the polynomial: $X^3 + X^1 + 1.$

The divisor in a CRC is normally called the generator polynomial or simply the generator.

polynomial generator

A good polynomial generator needs to have the following characteristics:

1. **It should have at least two terms.**
2. **The coefficient of the term x^0 should be 1.**
3. **It should have the factor $x + 1$.**

polynomial generator

It is obvious that we cannot choose **x** (binary 10) or **$x^2 + x$** (binary 110) as the polynomial because both are divisible by x to guarantee that all burst errors of length ≤ degree of the polynomial are detected.

However, we can choose **x + 1** (binary 11) or **$x^2 + 1$** (binary 101) because it is divisible by x + 1 (binary division) to guarantee that all burst of odd numbers are detected.

Example 1

The CRC-12

$$x^{12} + x^{11} + x^3 + x + 1$$

which has a degree of 12, will detect all burst errors affecting an odd number of bits, will detect all burst errors with a length less than or equal to 12, and will detect, 99.97 percent of the time, burst errors with a length of 12 or more.

If the generator has more than one term and the coefficient of x^0 is 1, all single errors can be caught.

A generator that contains a factor of $x + 1$ can detect all odd-numbered errors.

Standard polynomials

Name	Polynomial	Application
CRC-8	$x^8 + x^2 + x + 1$	**ATM header**
CRC-10	$x^{10} + x^9 + x^5 + x^4 + x^2 + 1$	**ATM AAL**
CRC-16 (ITU-16)	$x^{16} + x^{12} + x^5 + 1$	**HDLC**
CRC-32 (ITU-32)	$x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} + x^{10} + x^8 + x^7 + x^5 + x^4 + x^2 + x + 1$	**LANs**

Lecture 5

Checksum

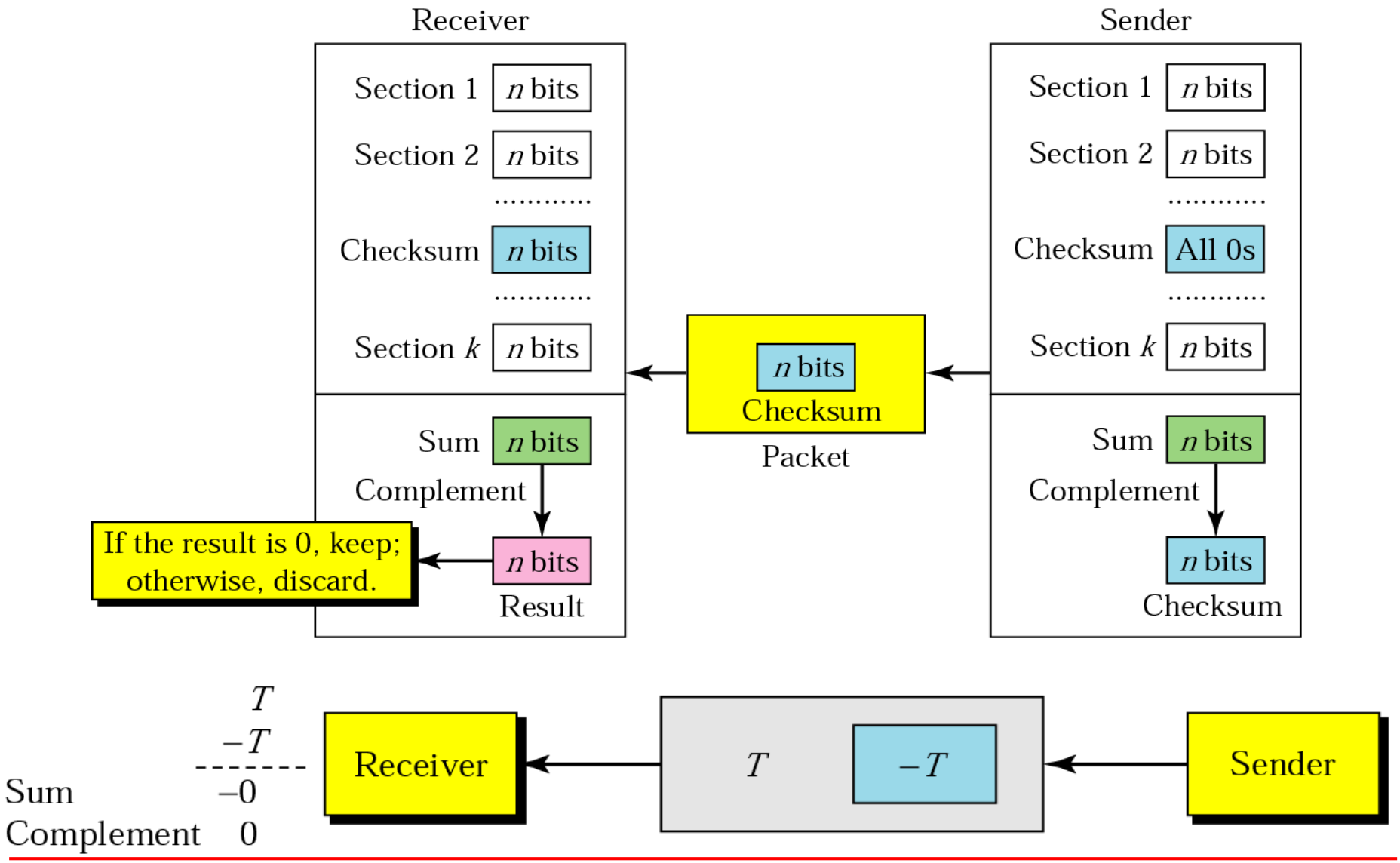
3.5 CHECKSUM

Checksum is an error detection method. The checksum is used in the Internet by several protocols although not at the data link layer. However, we briefly discuss it here to complete our discussion on error checking

Topics discussed in this section:

Idea
One's Complement
Internet Checksum

CHECKSUM


Receiver
Sender
Section 1
n bits
Section 2
n bits
Checksum
n bits
Section k
n bits
All 0s
Sum
Complement
Result
Checksum
Packet
If the result is 0, keep;
otherwise, discard.
T
–T
Sum
Complement
–0
0
Receiver
Sender

CHECKSUM

The sender follows these steps:

- The unit is divided into k sections, each of n bits.
- All sections are added using one's complement to get the sum.
- The sum is complemented and becomes the checksum.
- The checksum is sent with the data.

CHECKSUM

The receiver follows these steps:

- The unit is divided into k sections, each of n bits.
- All sections are added using one's complement to get the sum.
- The sum is complemented.
- If the result is zero, the data are accepted: otherwise, rejected.

Example 2

Suppose the following block of 16 bits is to be sent using a checksum of 8 bits.

10101001 00111001

The numbers are added using one's complement

	10101001
	00111001

Sum	11100010
Checksum	**00011101**

The pattern sent is 10101001 00111001 **00011101**

Example 3

Now suppose the receiver receives the pattern sent in Example 2 and there is no error. 10101001 00111001 00011101

When the receiver adds the three sections, it will get all 1s, which, after complementing, is all 0s and shows that there is no error.

	10101001
	00111001
	00011101
Sum	11111111
Complement	**00000000** means that the pattern is OK.

Example 4

Now suppose there is a burst error of length 5 that affects 4 bits.

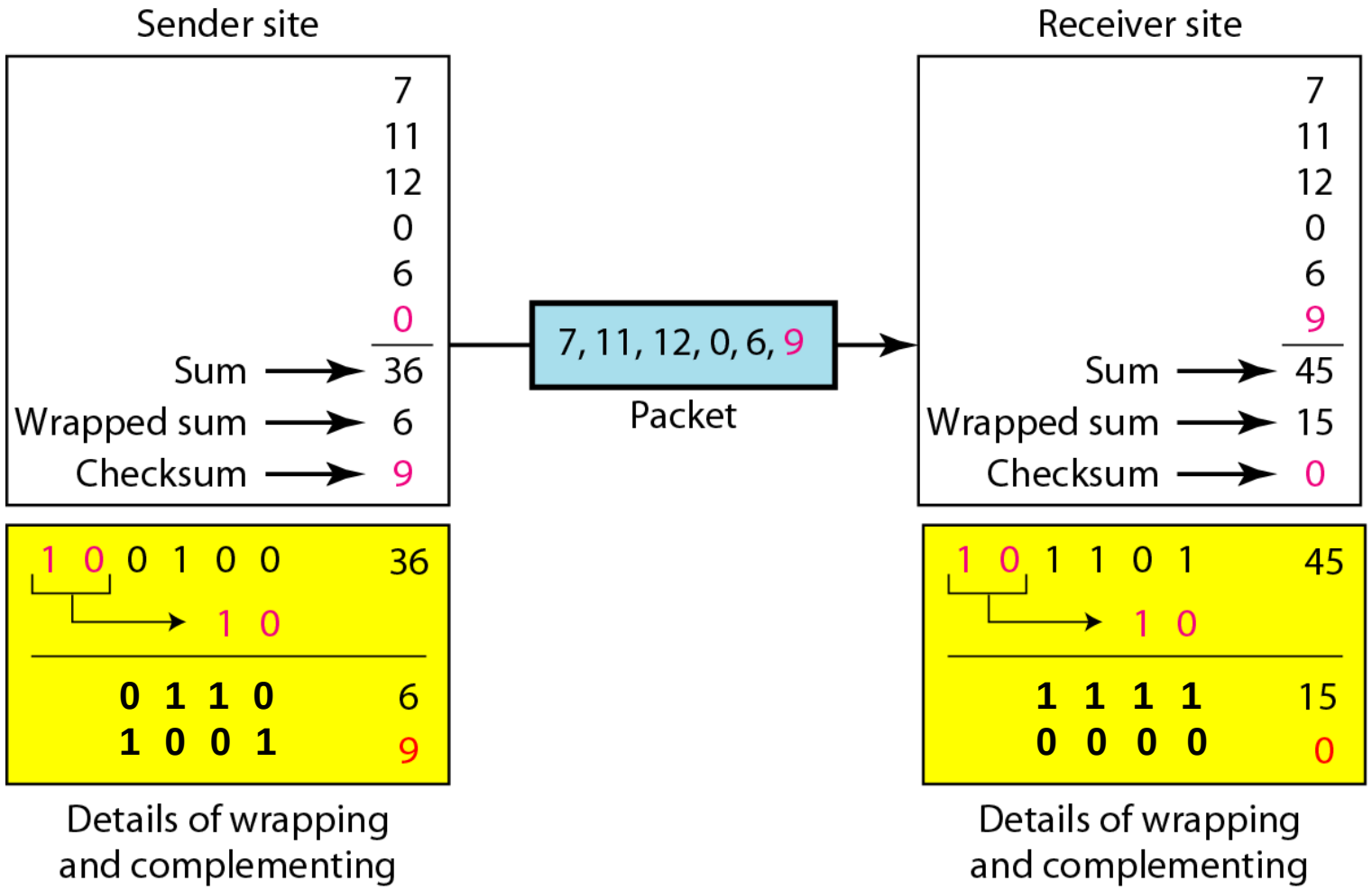
10101***111 11***111001 00011101

When the receiver adds the three sections, it gets

	10101111	
	11111001	
	00011101	
Partial Sum	**1** 11000101	
Carry	**1**	
Sum	11000110	
Complement	**00111001**	**the pattern is corrupted.**

Example 5

If we need to send the set of numbers (7, 11, 12, 0, 6), we

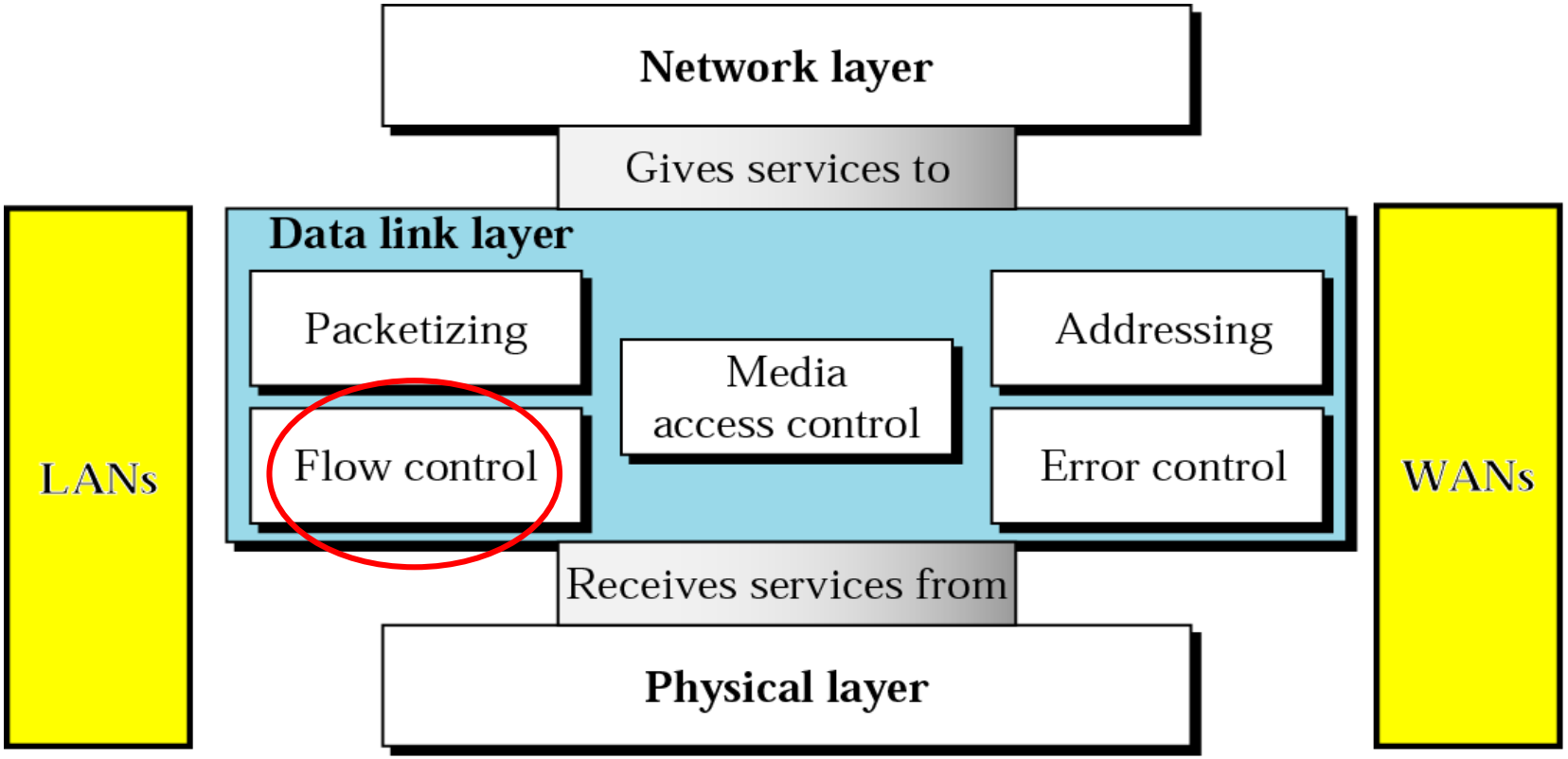


An error-detecting code can detect only the types of errors for which it is designed; other types of errors may remain undetected.

Lecture 6

Flow Control

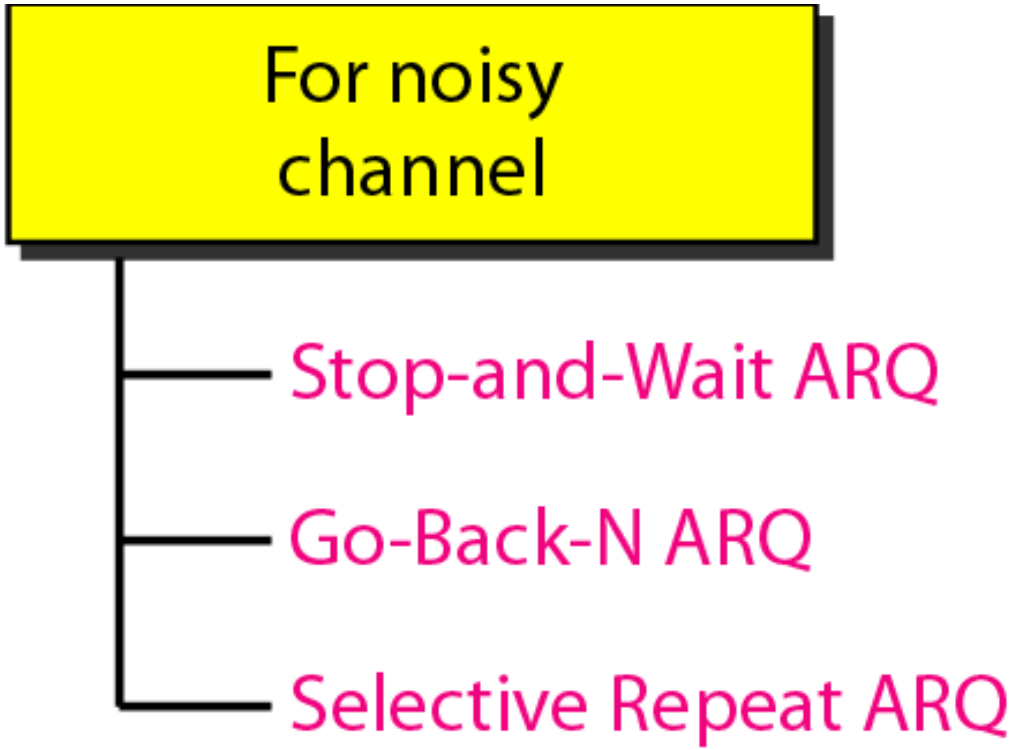
Data Link Layer (DLL)



Flow and Error Control

- The most important responsibilities of the data link layer are flow control and error control. Collectively, these functions are known as data link control.
  - Flow control refers to a set of procedures used to restrict the amount of data that the sender can send before waiting for acknowledgment.
  - Error control in the data link layer is based on automatic repeat request (ARQ), which is the retransmission of data.

Protocols


For noisy channel
Stop-and-Wait ARQ
Go-Back-N ARQ
Selective Repeat ARQ

Noisy Channels

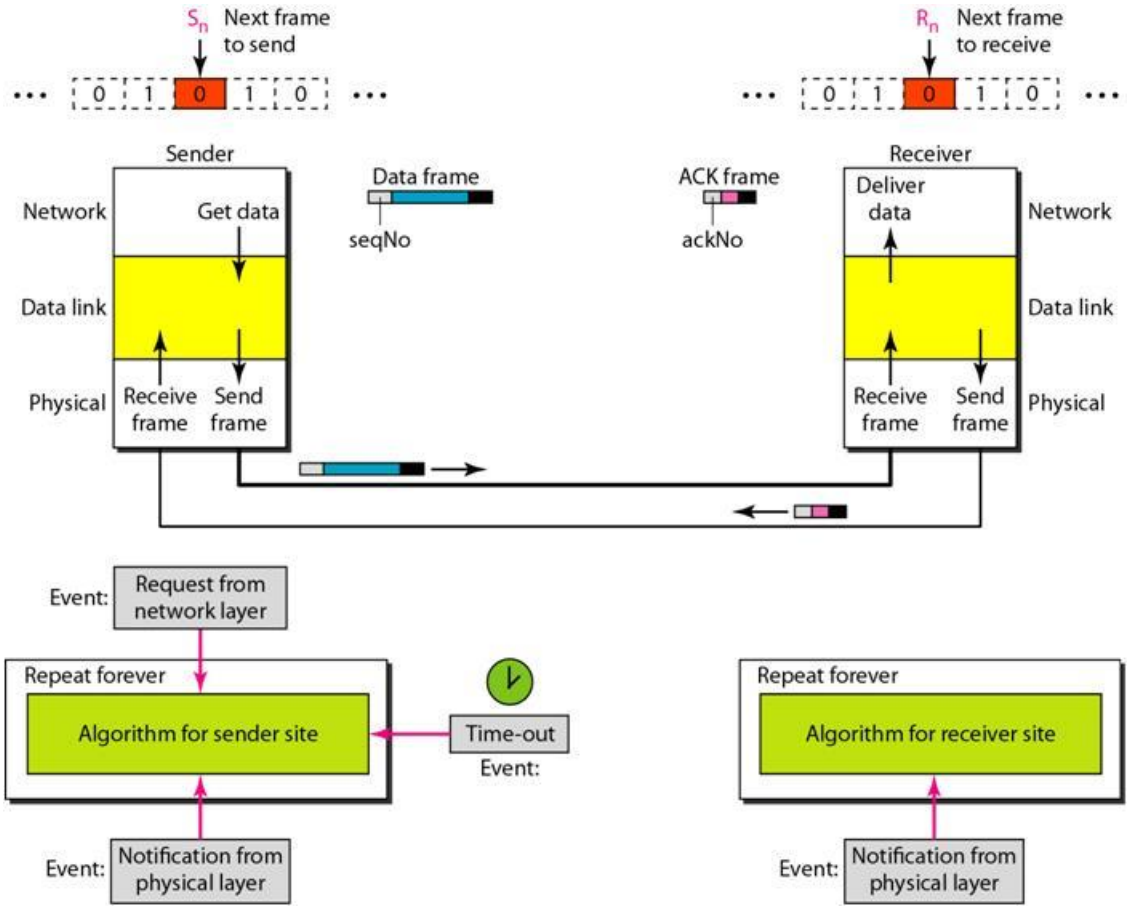
We discuss three protocols in this section that use error control.

- Stop & Wait Automatic Repeat Request
- Go-Back-N Automatic Repeat Request
- Selective Repeat Automatic Repeat Request

Stop-and-Wait Automatic Repeat Request

- Simplest flow and error control mechanism
- The sending device keeps a copy of the last frame transmitted until it receives an acknowledgement
  - Identification of duplicate transmission
- A damaged or lost frame is treated in the same way at the receiver.
- A control variable is needed at both sides.
- Timers introduced at the sender side.
- Positive ACK sent only for frames received safe & sound to define the next expected frame.

Stop-and-Wait



Stop-and-Wait ARQ

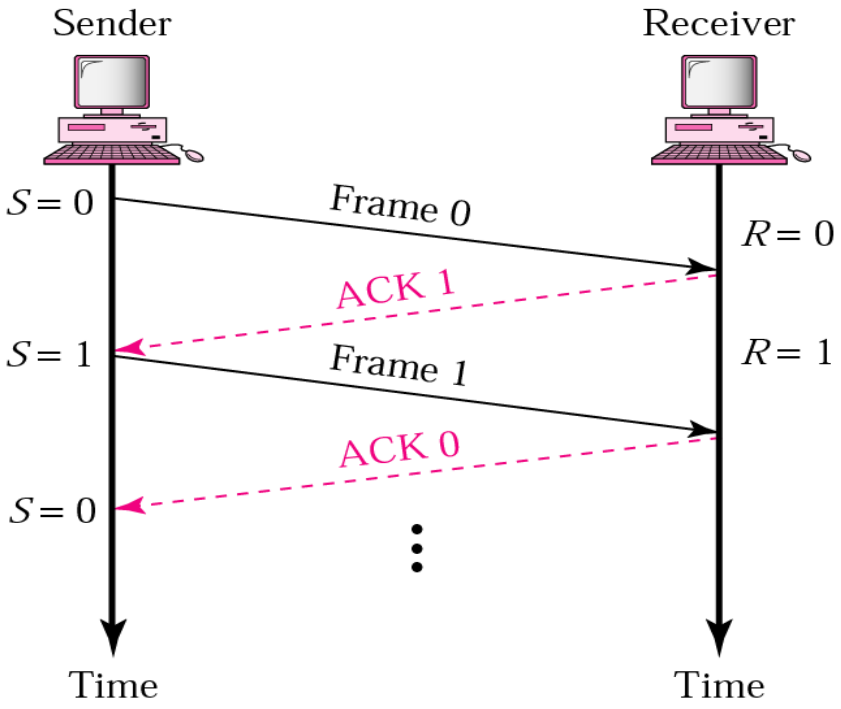
Cases of operations:

1. Normal operation
2. The frame is lost
3. The ACK is lost
4. The ACK is delayed

Stop-and-Wait ARQ

Normal operation

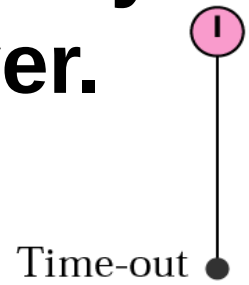
- The sender will not send the next piece of data until it is sure that the current one is correctly received
- sequence number is necessary to check for duplicated packets
- No NACK – when packet is corrupted – duplicate ACKs instead

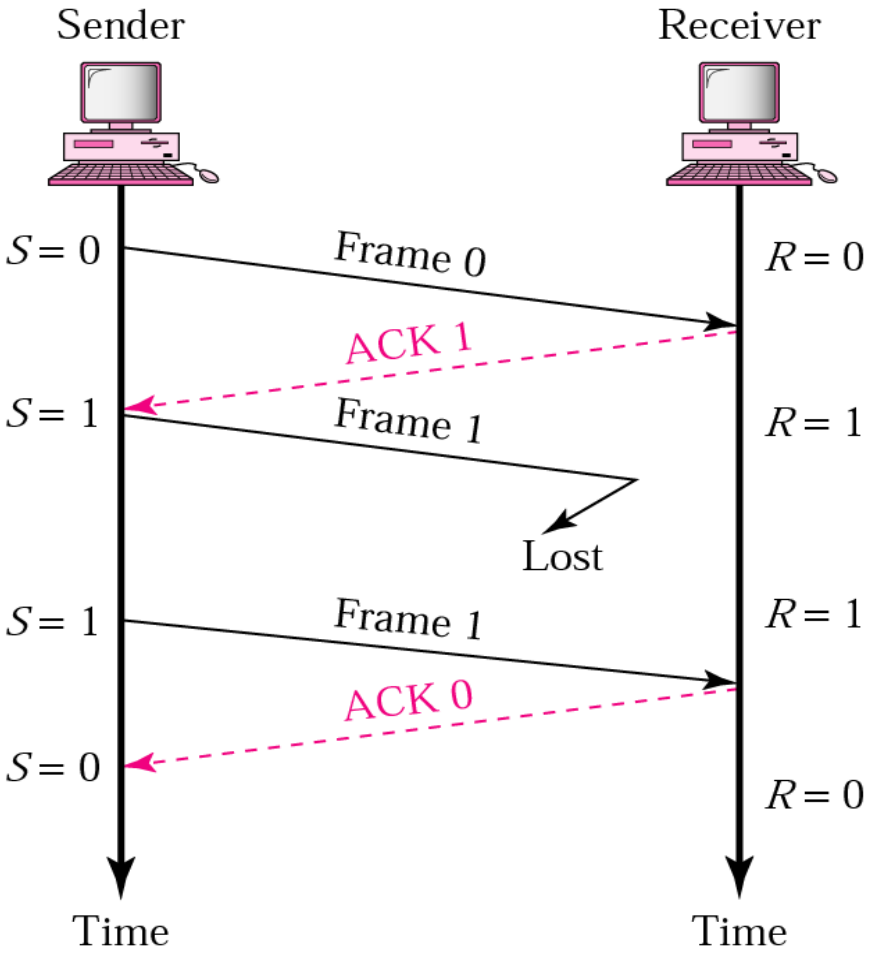


Stop-and-Wait ARQ

Lost or damaged frame

A damaged or lost frame is treated in the same way at the receiver.

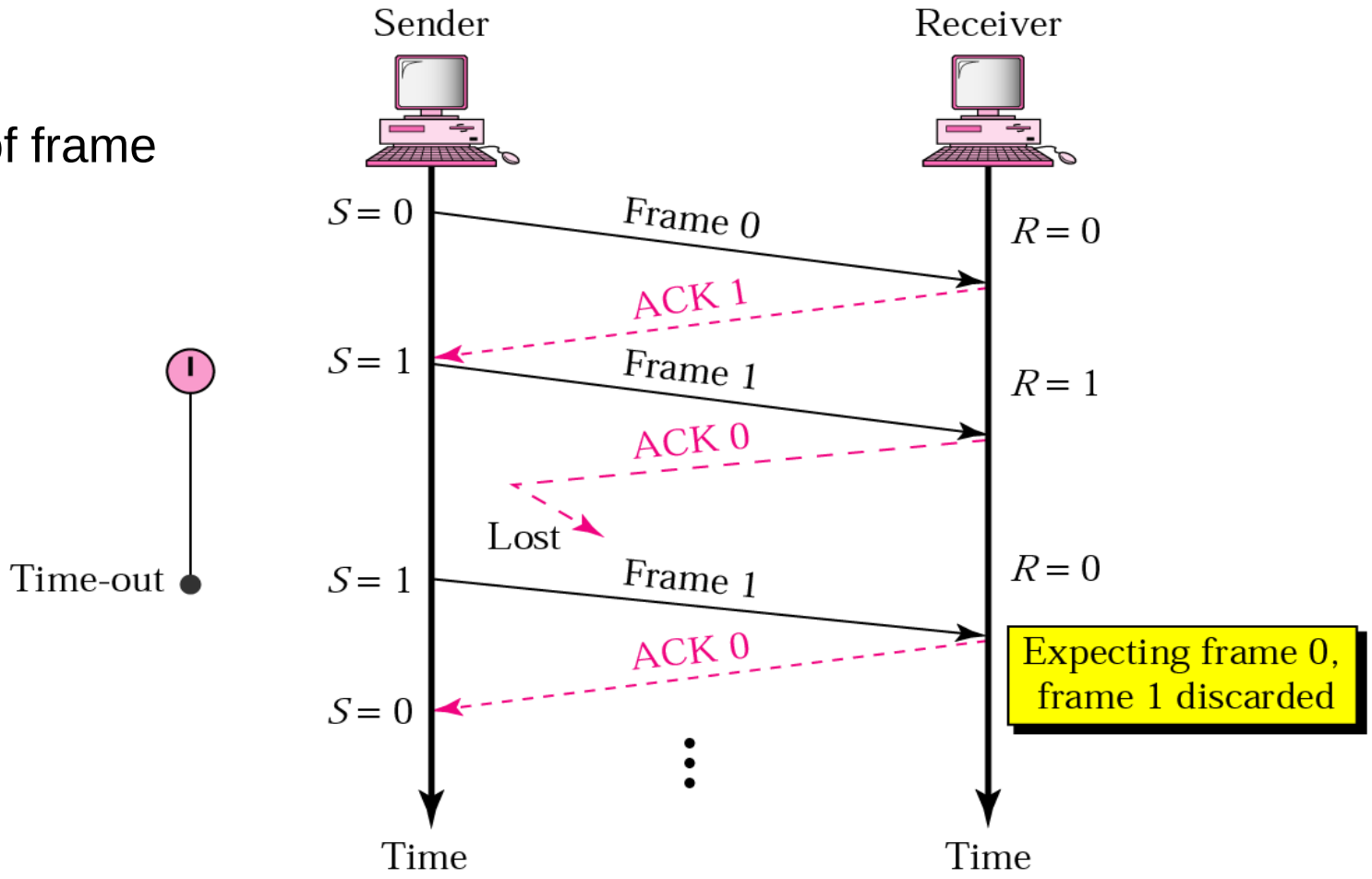




Stop-and-Wait ARQ

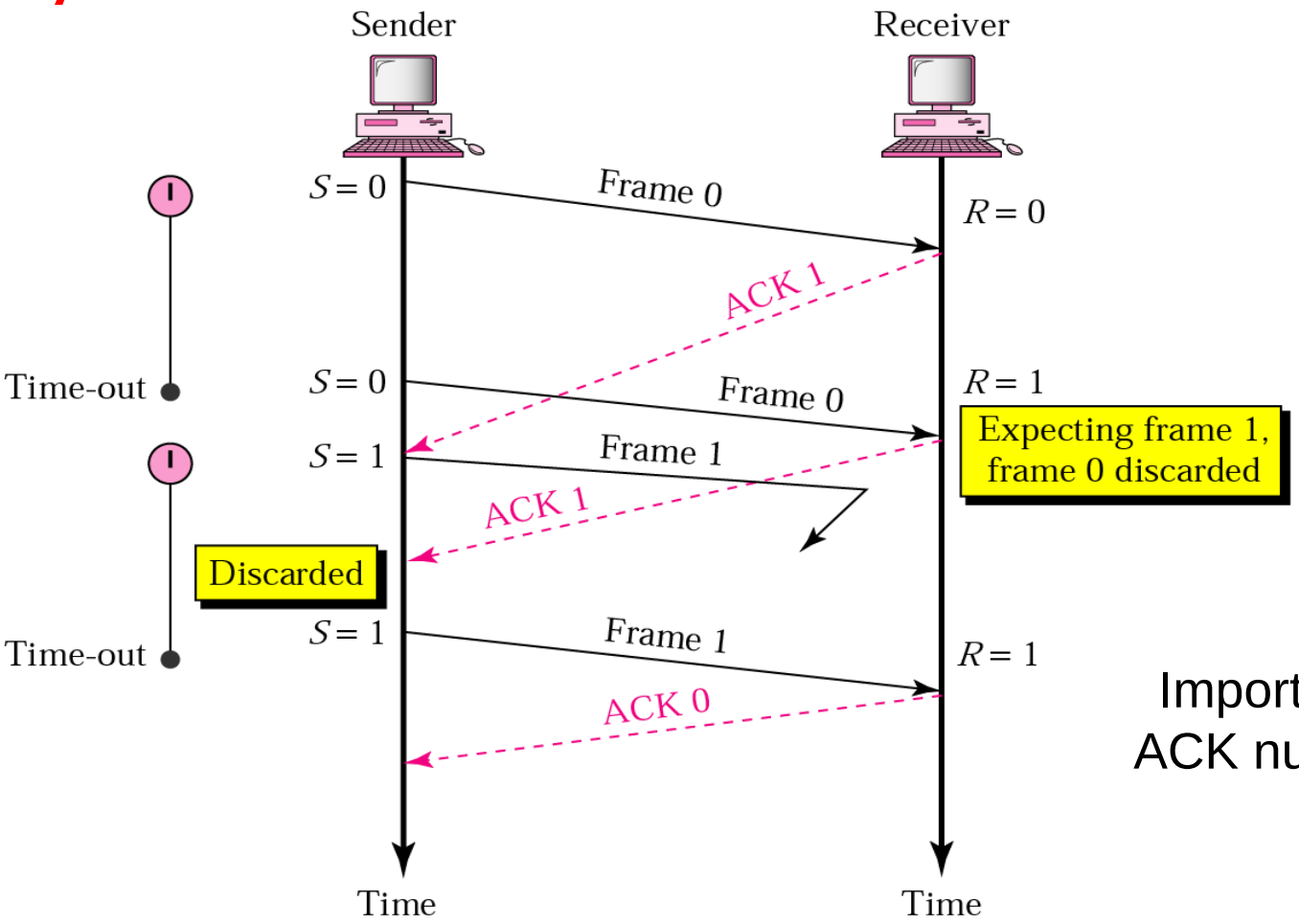
Lost ACK

Importance of frame numbering



Stop-and-Wait ARQ

Delayed ACK

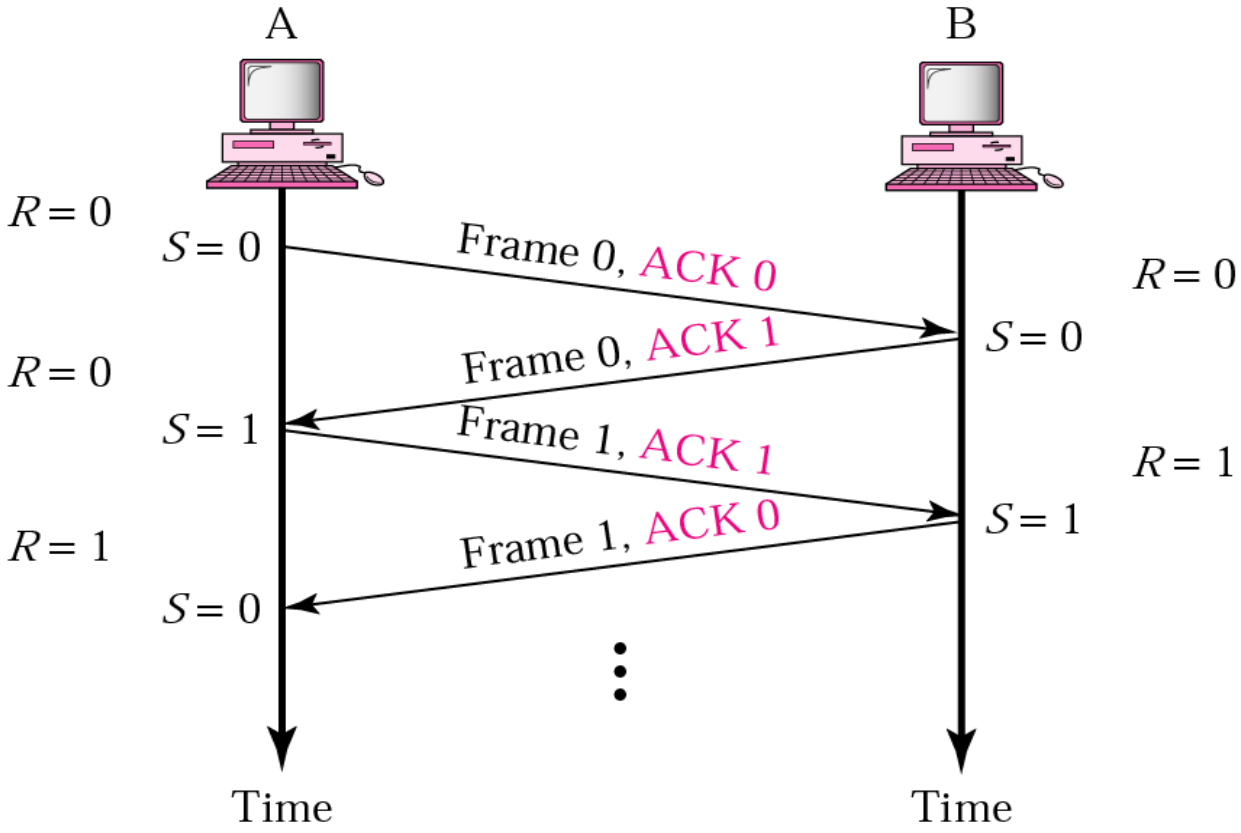


Importance of ACK numbering

Duplex Stop-and-Wait ARQ

Piggybacking

- combine data with ACK (less overhead saves bandwidth)



Efficiency

The bandwidth-delay product defines the number of bits that can fill the link.

bandwidth-delay product = Bandwidth * Delay.

Example

Assume that, in a Stop-and-Wait ARQ system, the bandwidth of the line is 1 Mbps, and 1 bit takes 20 ms to make a round trip. What is the bandwidth-delay product?

If the system data frames are 1000 bits in length, what is the utilization percentage of the link?

Solution

The bandwidth-delay product is

$$(1 \times 10^6) \times (20 \times 10^{-3}) = 20{,}000 \text{ bits}$$

the utilization percentage of the link is 1000/20,000 = 5%

Stop-and-Wait ARQ

- After each frame sent the host must wait for an ACK
  - inefficient use of bandwidth
- To improve efficiency ACK should be sent after multiple frames
- Alternatives: Sliding Window protocols
  - Go-back-*N* ARQ
  - Selective Repeat ARQ

Lecture 7

Go-back-N

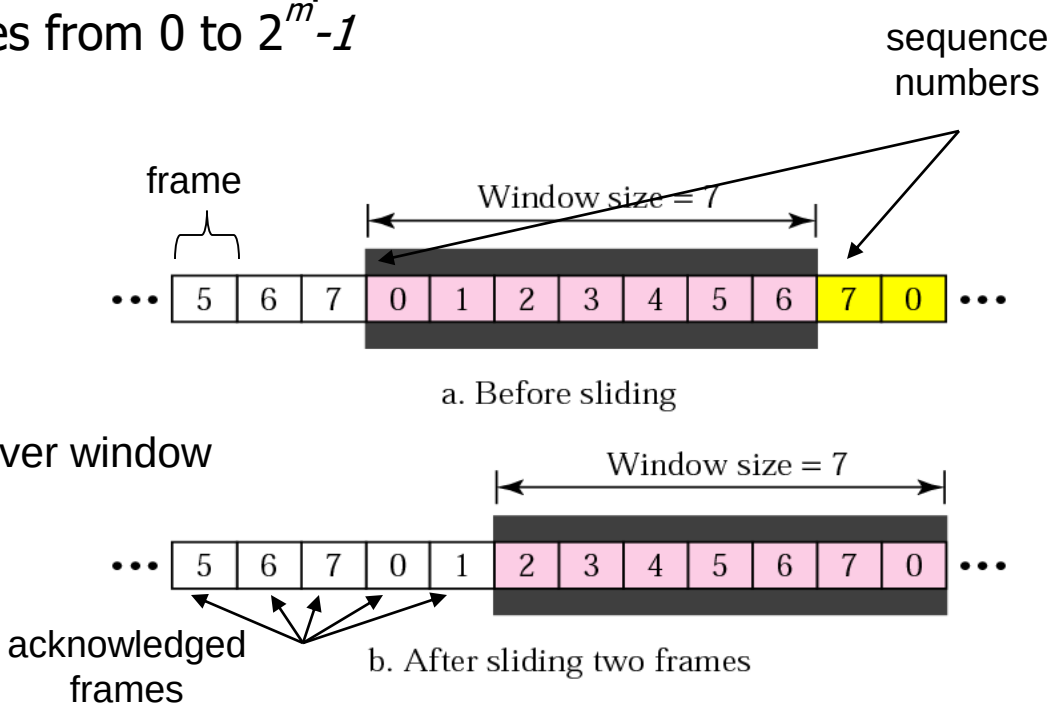
Pipelining

- One task begins before the other one ends
  - increases efficiency in transmission
- There is no pipelining in Stop-and-Wait ARQ

Sliding Window Protocols

- Sequence numbers
  - Sent frames are numbered sequentially
  - Sequence number is stored in the header
    - if the number of bits to store the sequence number is *m* than sequence number goes from 0 to $2^m - 1$

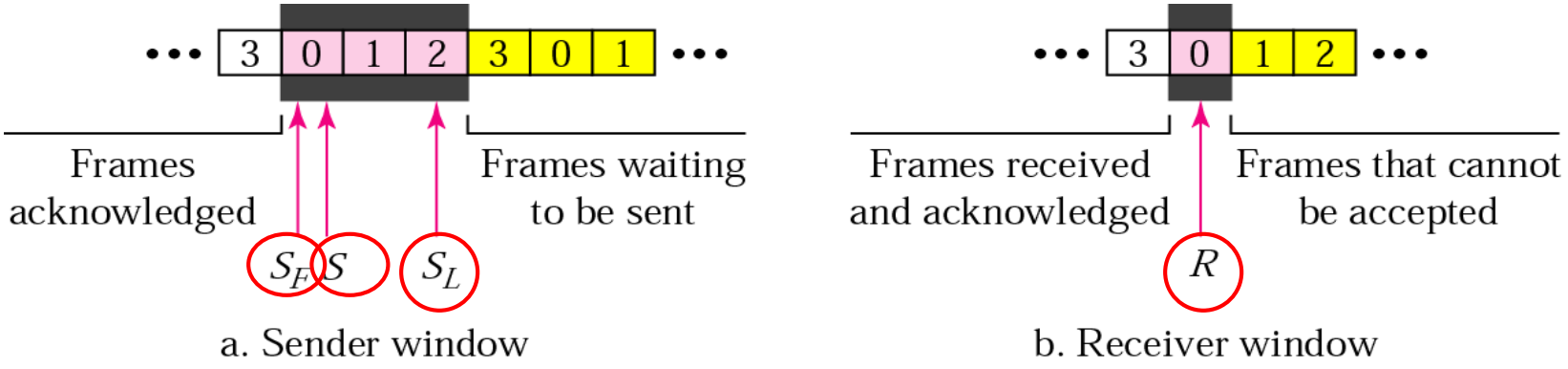
- Sliding window
  - ❑ to hold the unacknowledged outstanding frames
  - ❑ In Go-back-N ARQ the receiver window size always 1



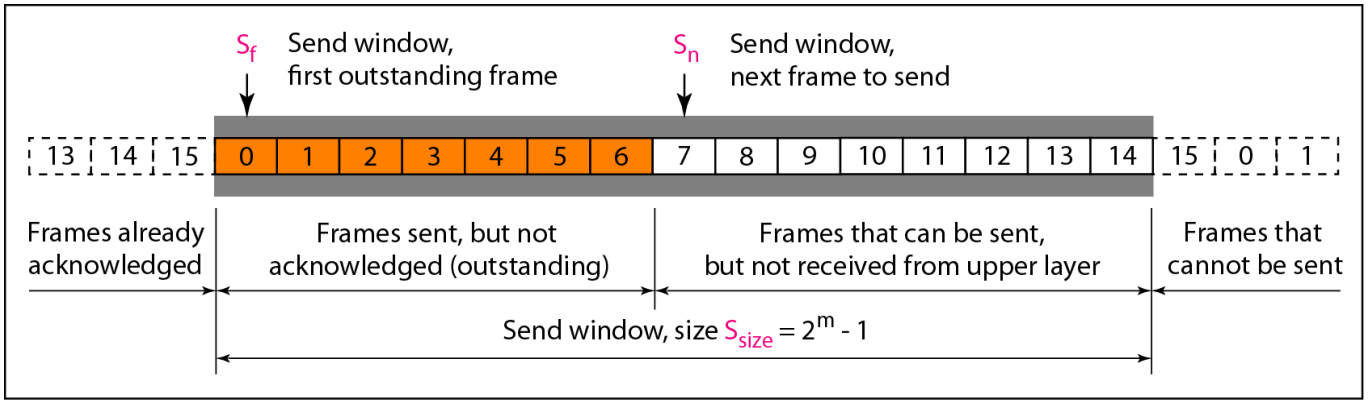
Go-back-*N*

Control variables

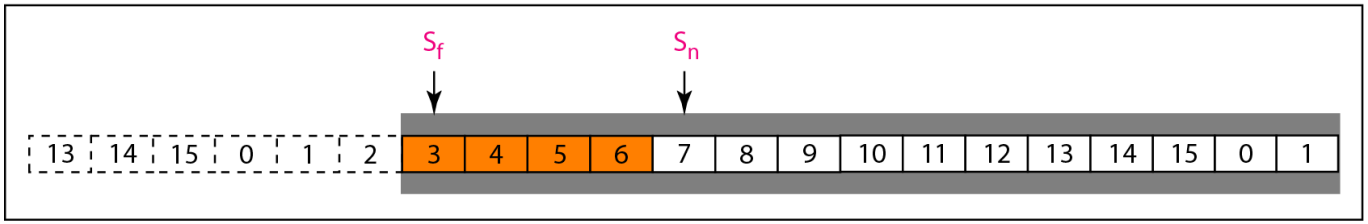
- *S*– holds the sequence number of the recently sent frame
- S_F – holds sequence number of the first frame in the window
- S_L – holds the sequence number of the last frame *or* S_n – holds the sequence number of the next frame to be send
- *R* – sequence number of the frame expected to be received



Go-back-*N*

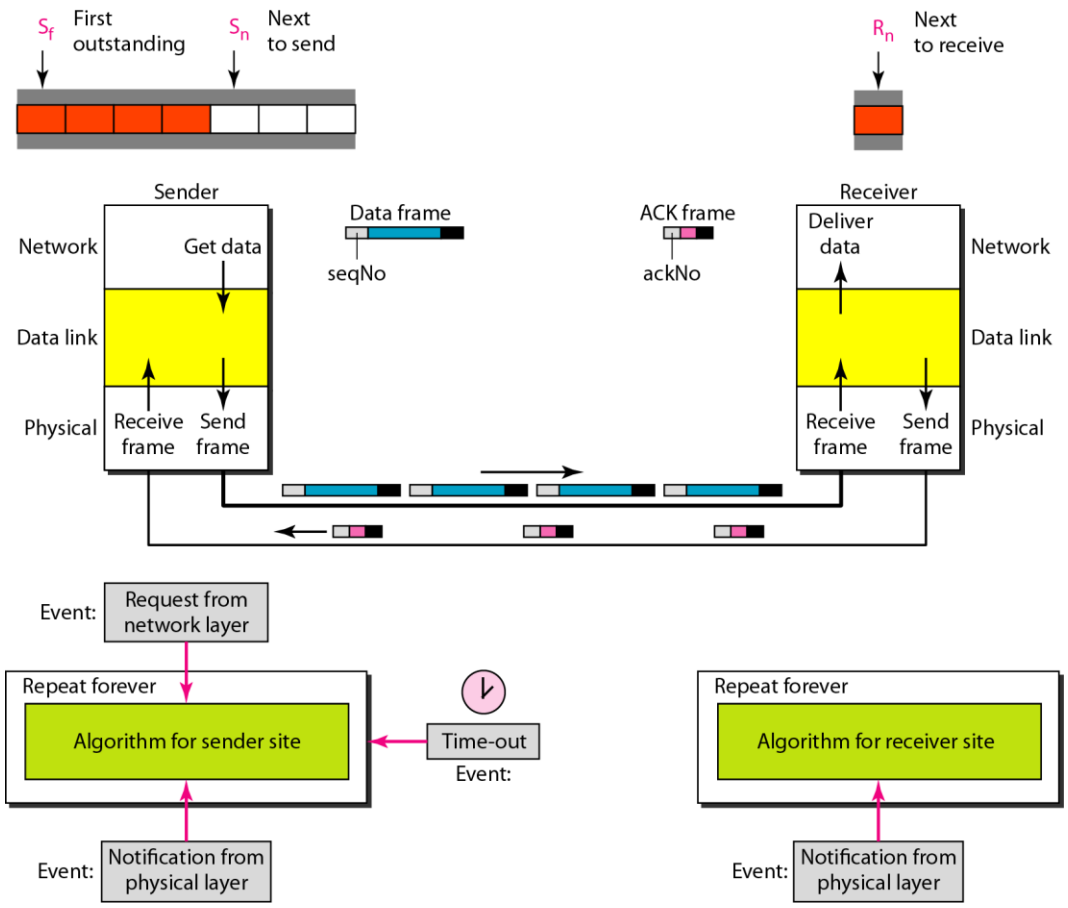


a. Send window before sliding



b. Send window after sliding

Go-back-*N*


S_f First outstanding
S_n Next to send
R_n Next to receive
Sender
Receiver
Network
Get data
Deliver data
Data frame
seqNo
ACK frame
ackNo
Data link
Physical
Receive frame
Send frame
Event:
Request from network layer
Repeat forever
Algorithm for sender site
Time-out
Event:
Algorithm for receiver site
Notification from physical layer

The name of Go-back-*N*: why?

Re-sending frame

when the frame is damaged the sender **goes back** and resends a set of frames starting from the last one acknowledged; the number of retransmitted frames is ***N***

Example:

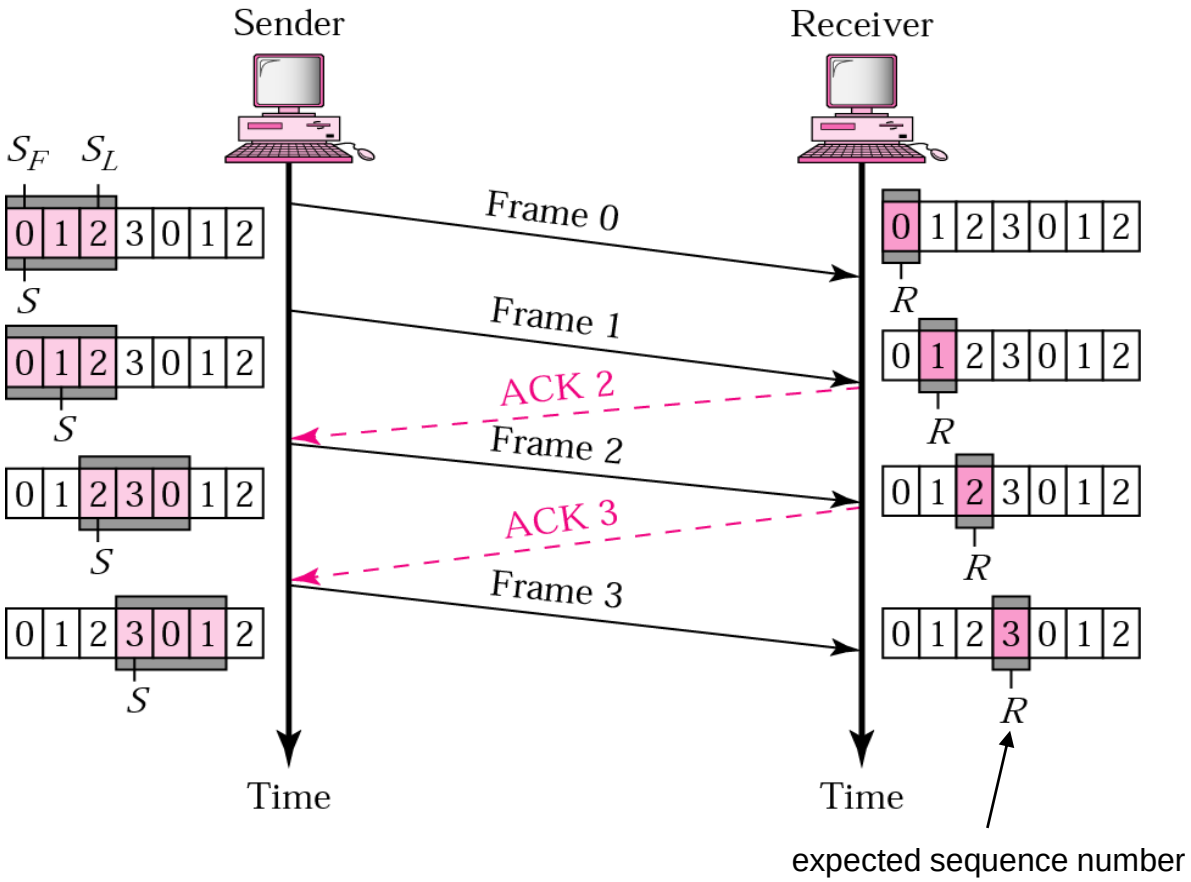
The window size is 4.

A sender has sent frame 6 and the timer expires for frame 3 (ACK3 is not received). The sender goes back and resends the frames 3, 4, 5 and 6.

Go-back-*N*

normal operation

- How many frames can be transmitted without acknowledgment?
- ACK1 – not necessary if ACK2 is sent
  - Cumulative ACK



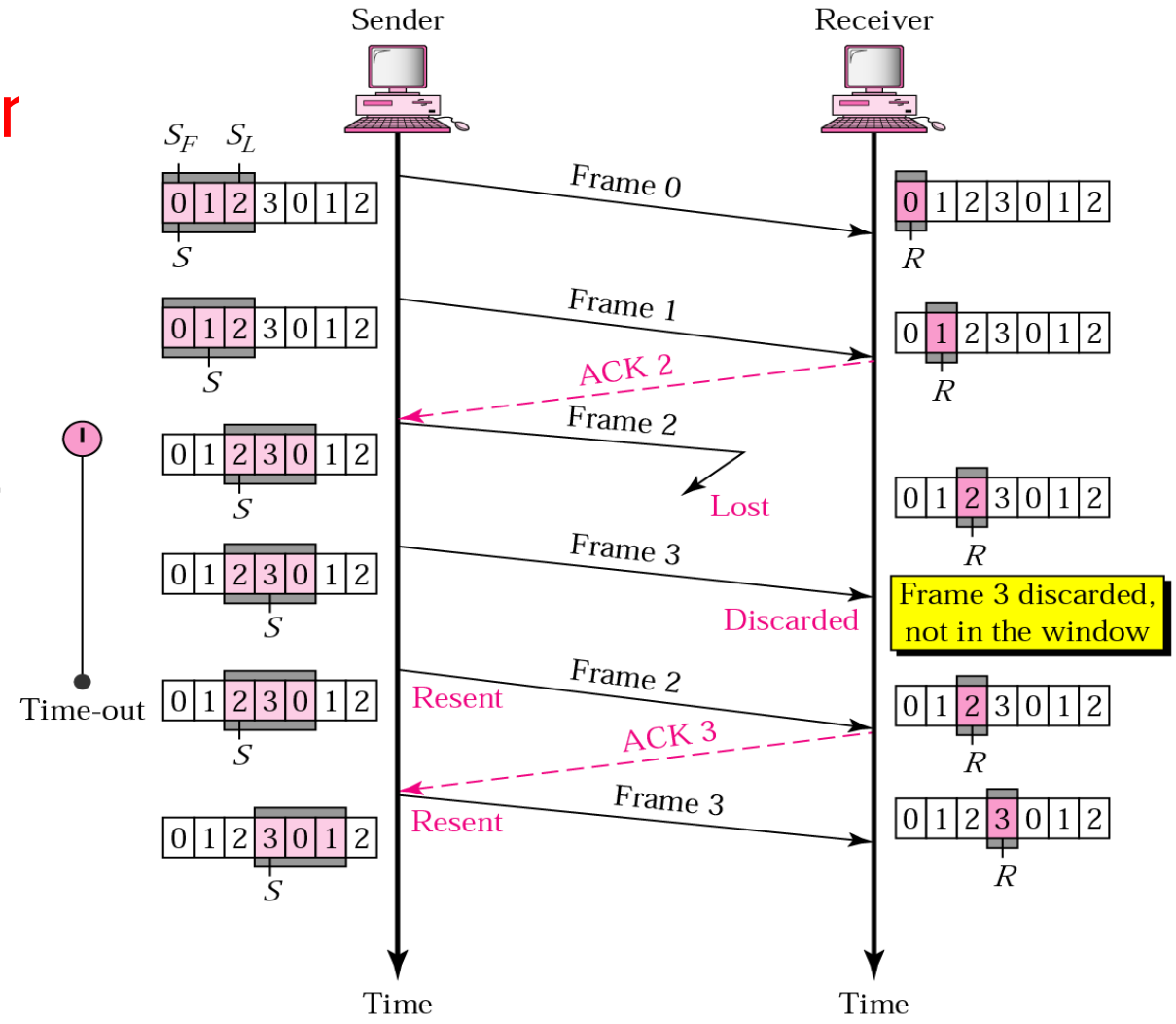
Go-back-*N*

damaged or lost frar

Damaged frames are discarded!

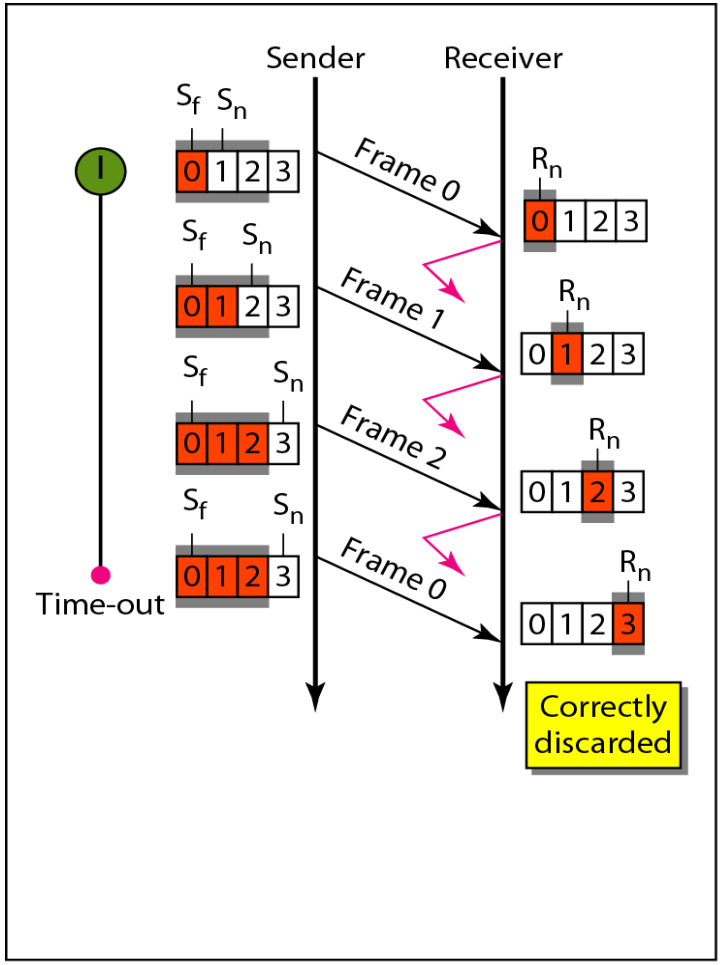
Why are correctly received out of order packets are not buffered?

What is the disadvantage of this?

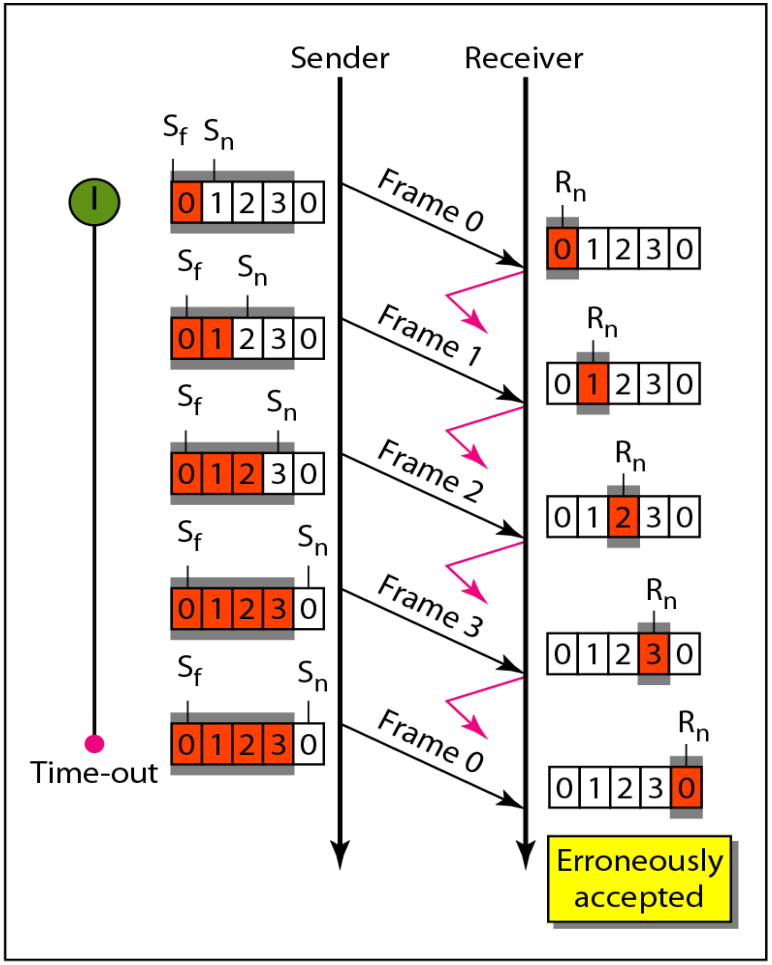


Go-back-*N*

sender window size



a. Window size < 2^m



b. Window size = 2^m

In the Go-Back-N Protocol, the sequence numbers are modulo 2^m, where m is the size of the sequence number field in bits (Number of bits needed to represent one sequence number).

Go-back-*N*

- Inefficient
  - all out of order received packets are discarded
- This is a problem in a noisy link
  - many frames must be retransmitted -> bandwidth consuming
- Solution
  - re-send only the damaged frames
- Selective Repeat ARQ
  - avoid unnecessary retransmissions

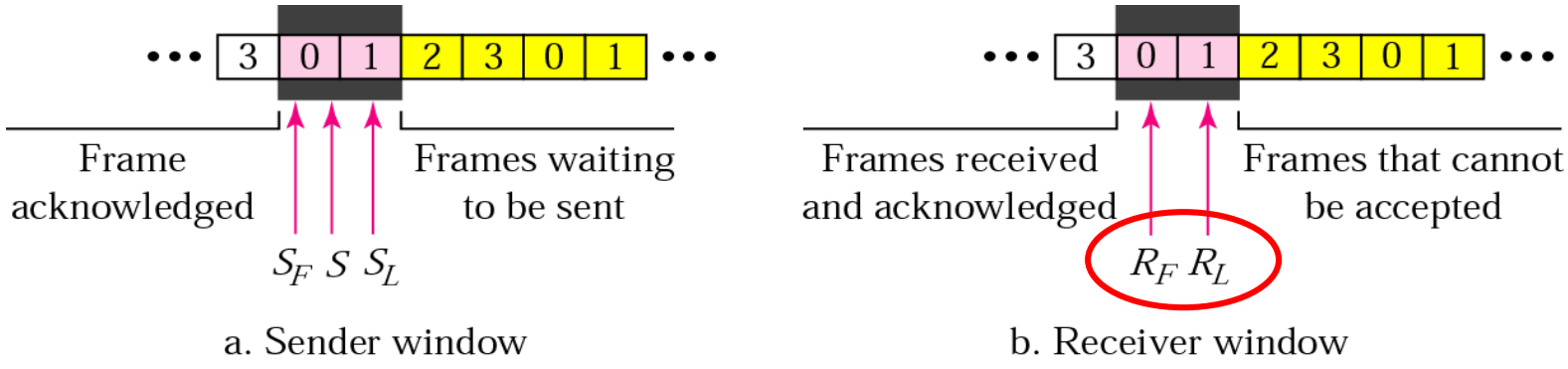
Stop-and-Wait ARQ is a special case of Go-Back-N ARQ in which the size of the send window is 1.

Lecture 8

Selective Repeat

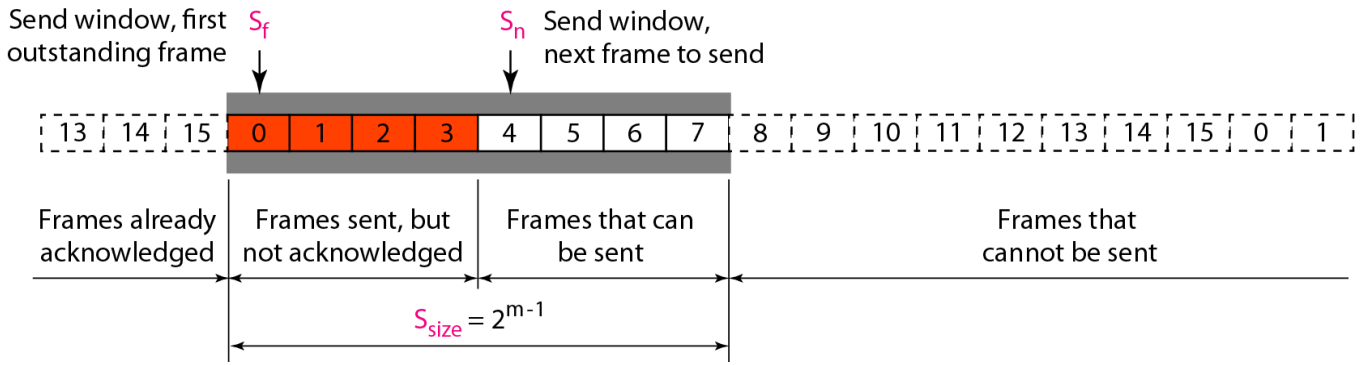
Selective Repeat ARQ

- Processing at the receiver more complex
- The window size is reduced to one half of 2^m
- Both the transmitter and the receiver have the same window size
- Receiver expects frames within the range of the sequence numbers

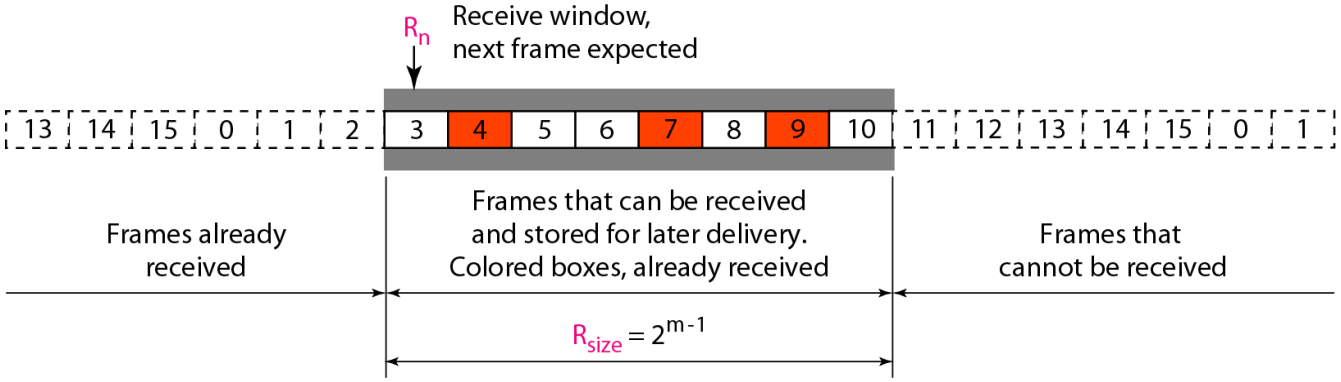


Selective Repeat ARQ

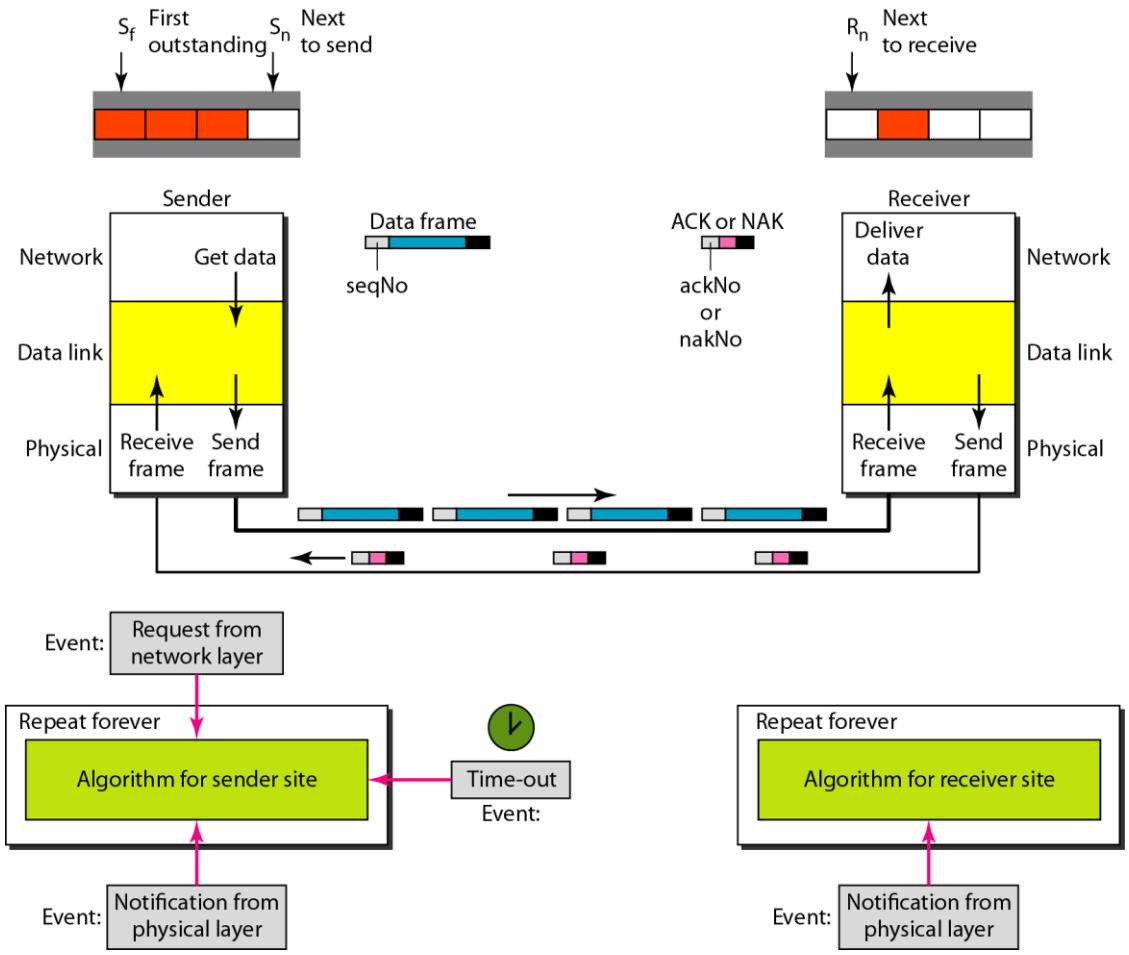
- Sender window size



- **Receive window size**

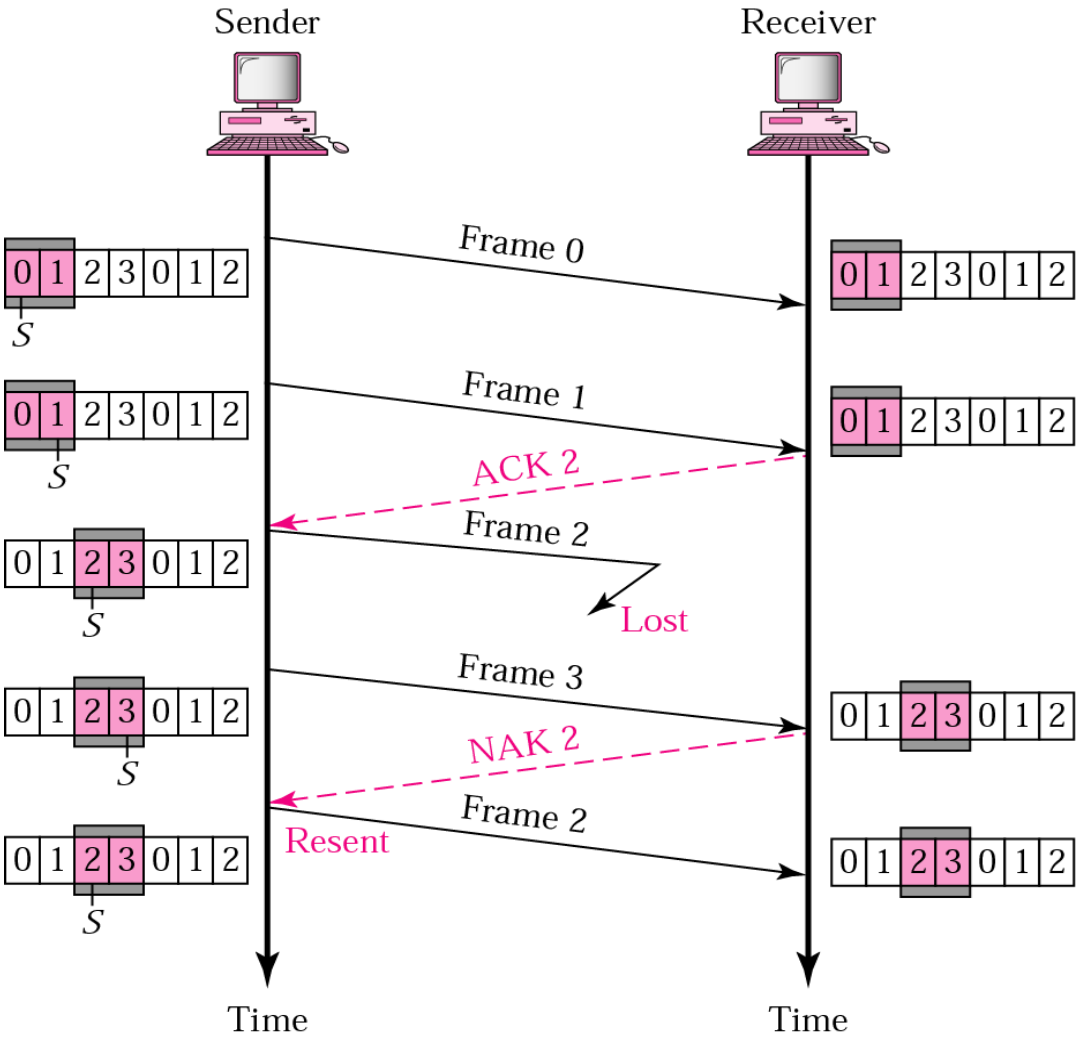


Selective Repeat ARQ



Selective Repeat ARQ

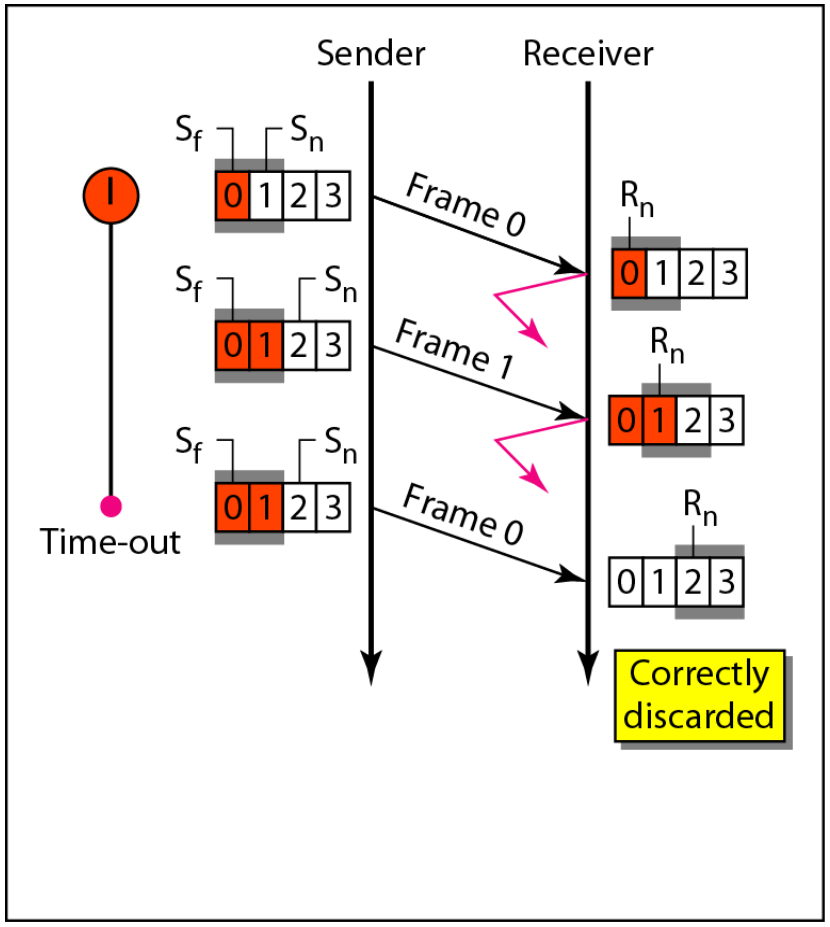
lost frame



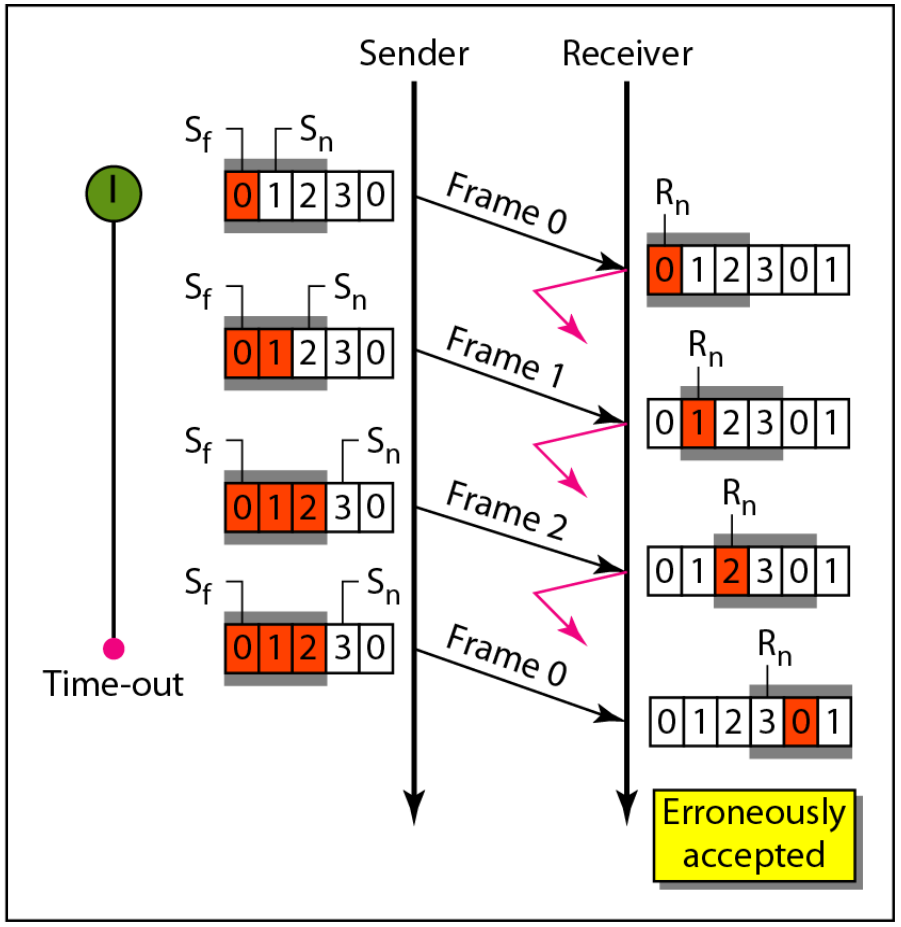
Note:
retransmission triggered with NACK and not with expired timer.

Selective Repeat ARQ

Sender window size



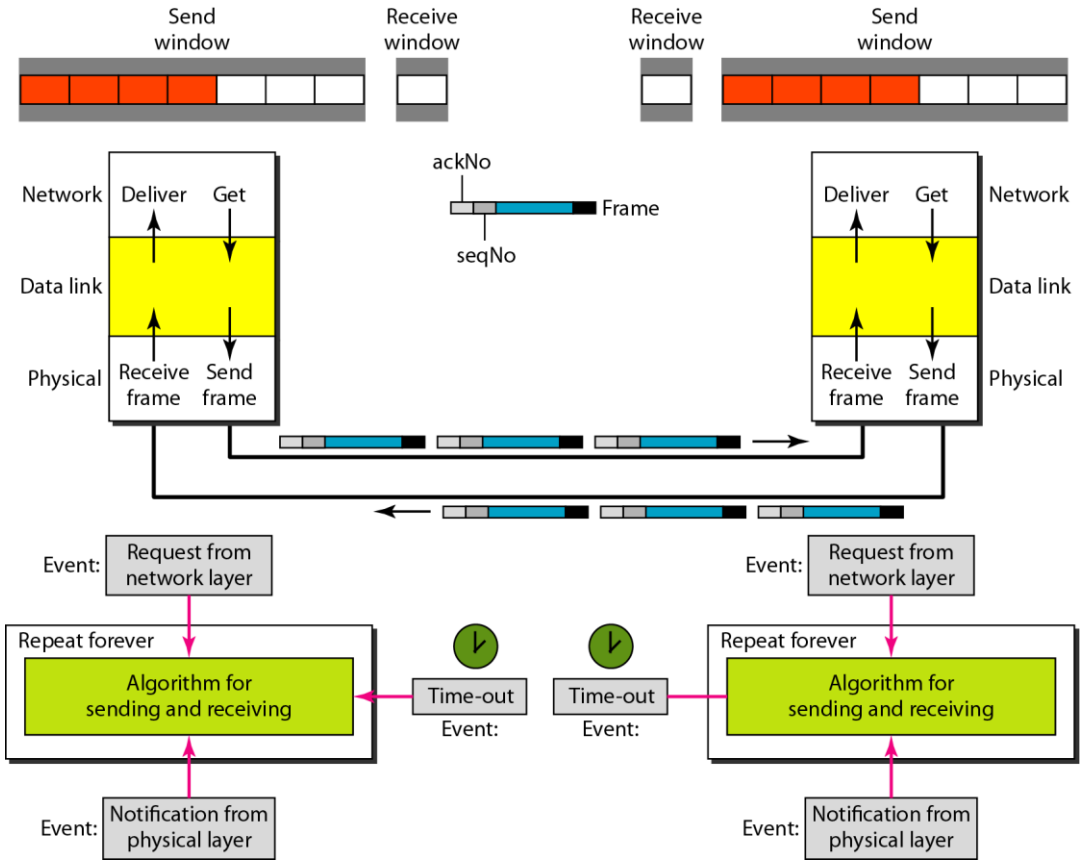
a. Window size = 2^{m-1}



b. Window size > 2^{m-1}

Piggybacking

- To improve the efficiency of the bidirectional protocols
- Piggybacking in Go-Back-N ARQ



Lecture 9

High-level Data Link Control

HDLC

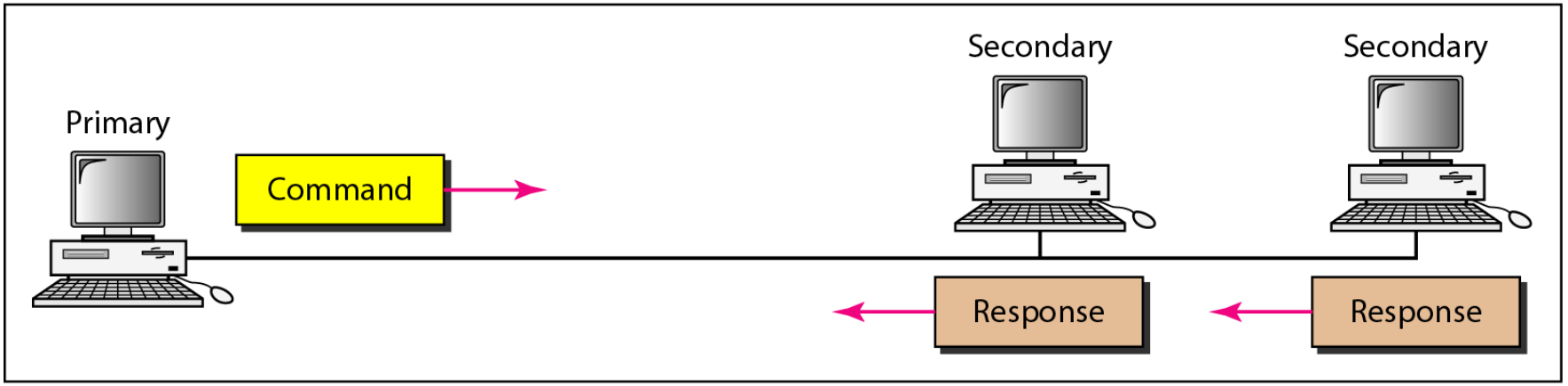
High-level Data Link Control (HDLC) ***supports half-duplex and full-duplex communication over point-to-point and multipoint links. It implements the ARQ mechanisms we discussed in this chapter.***

Topics discussed in this section:

- **Configurations and Transfer Modes**
  - **Normal respond mode (NRM)**
  - **Asynchronous balanced mode (ABM)**
- **Frames**
- **Control Field**

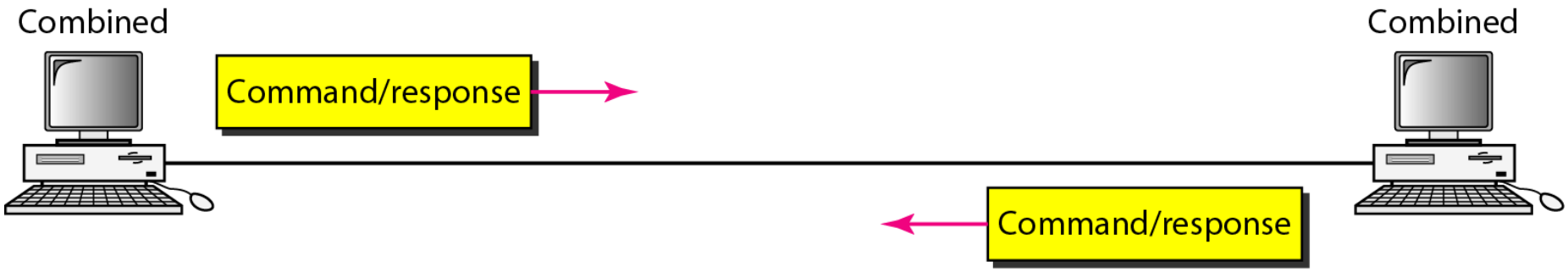
Normal response mode

a. Point-to-point



b. Multipoint

Asynchronous balanced mode (ABM)



ABM is used over point-to-point links.

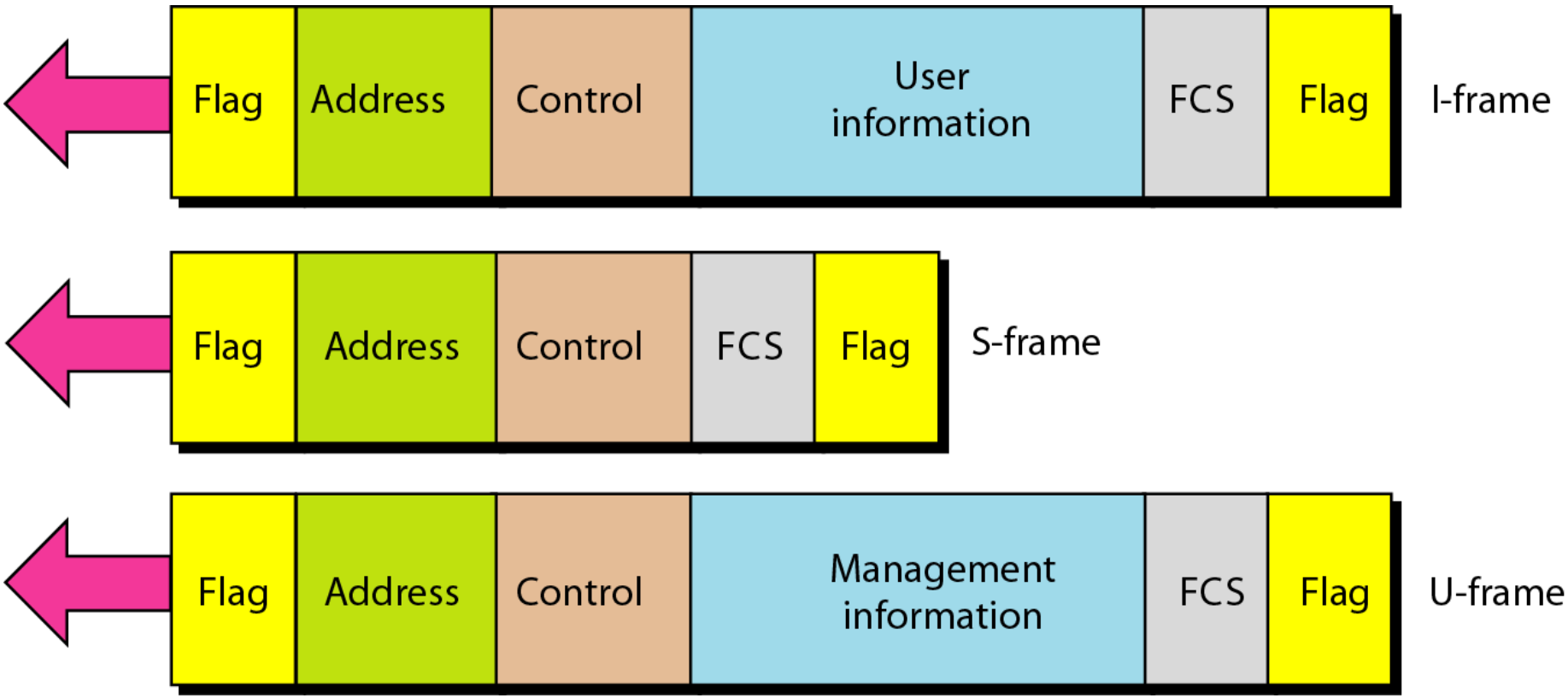
Frames

HDLC frames:

1. **Information frames (I-frame).**
2. **Supervisory frames (S-frame).**
3. **Unnumbered frames (U-frame).**

Frames

HDLC frames

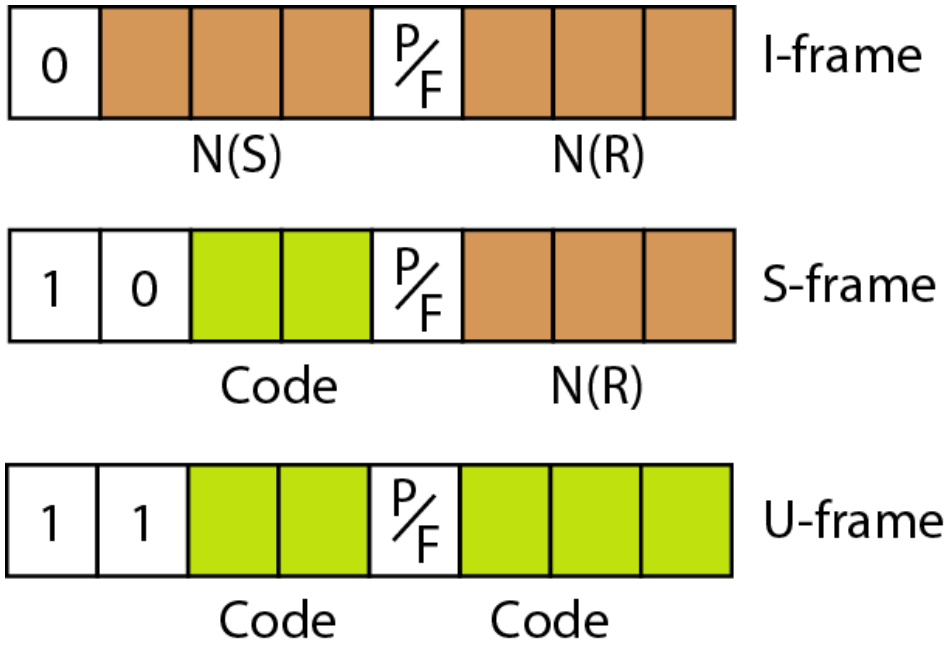


Frames

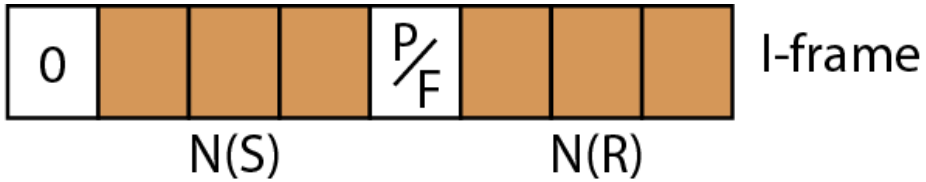
HDLC frames

1. **Flag: 8-bit ; 01111110 that identifies the beginning and end of the frame**
2. **FCS: frame check sequence is error detection field**
3. **Address: contain the address of the secondary station. It can be one byte or more . If the address is more than one bytes , all bytes end with zero except the last byte ends with ones.**

Control field format for the different frame types



Control field format for the I-frame

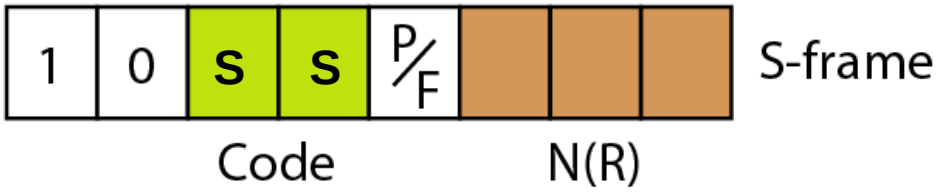


N(s): Sequence number of the frame in travel

N(R): the value of ACK when piggybacking is used.

P/F: (poll/Final)

Control field format for S-frame



1. **SS=00 RR - Receiver Ready to accept more I-frames**
2. **SS=01 REJ - Go-Back-N retransmission request for an I-frame**
3. **SS=10 RNR - Receiver Not Ready to accept more I-frames**
4. **SS=11 SREJ - Selective retransmission request for an I-frame**

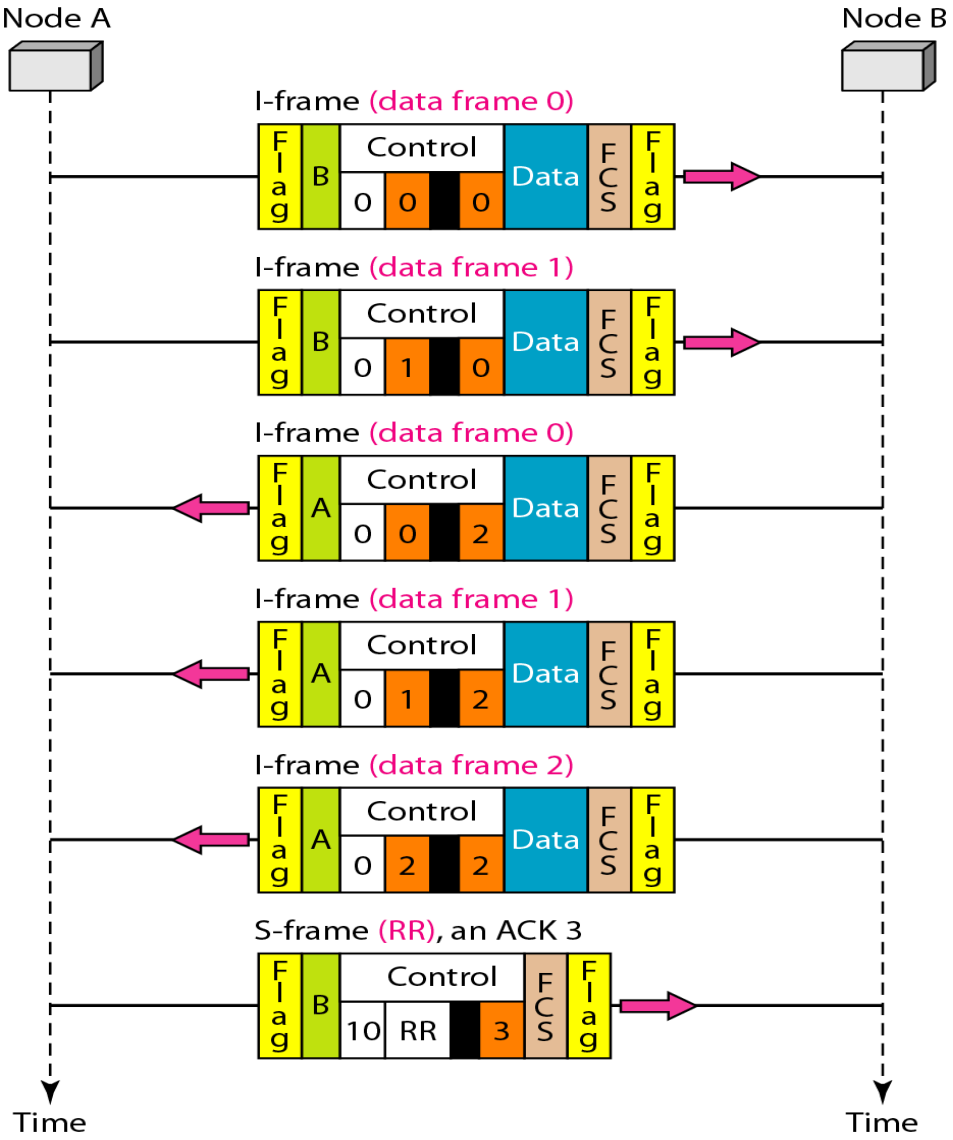
U-frame control command and response

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

Example

The figure in the next slide shows an exchange using piggybacking. Node A begins the exchange of information with an I-frame numbered 0 followed by another I-frame numbered 1. Node B piggybacks its acknowledgment of both frames onto an I-frame of its own. Node B's first I-frame is also numbered 0 [N(S) field] and contains a 2 in its N(R) field, acknowledging the receipt of A's frames 1 and 0 and indicating that it expects frame 2 to arrive next. Node B transmits its second and third I-frames (numbered 1 and 2) before accepting further frames from node A.

Example of piggybacking without error



Its N(R) information, therefore, has not changed: B frames 1 and 2 indicate that node B is still expecting A's frame 2 to arrive next. Node A has sent all its data. Therefore, it cannot piggyback an acknowledgment onto an I-frame and sends an S-frame instead. The RR code indicates that A is still ready to receive. The number 3 in the N(R) field tells B that frames 0, 1, and 2 have all been accepted and that A is now expecting frame number 3.

Figure 11.31 *Example of piggybacking with error*