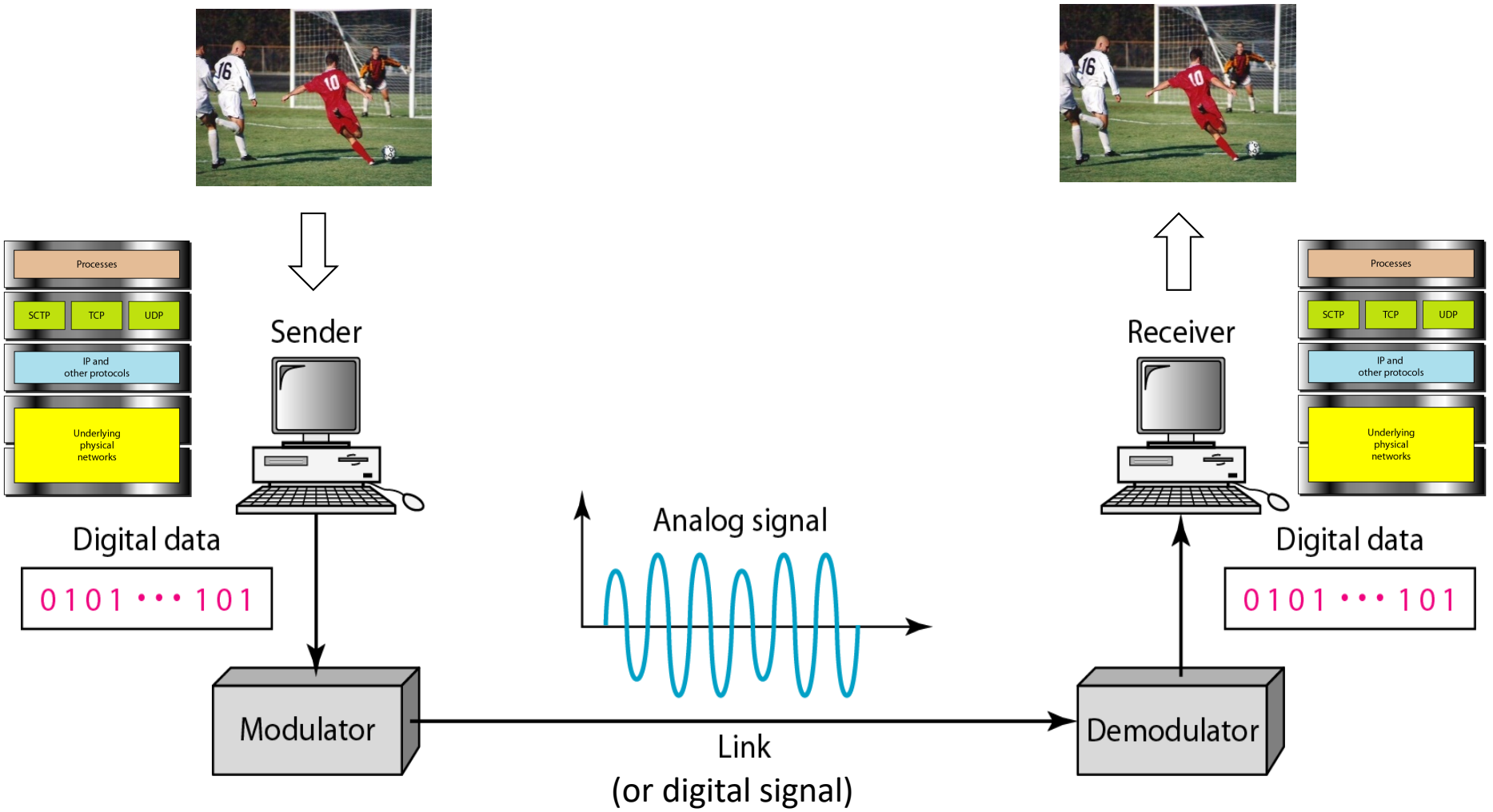
ETSF15 – Communication Systems and Networks

Data Link Layer
-1-
Error and Flow Control

2019, Lecture #3

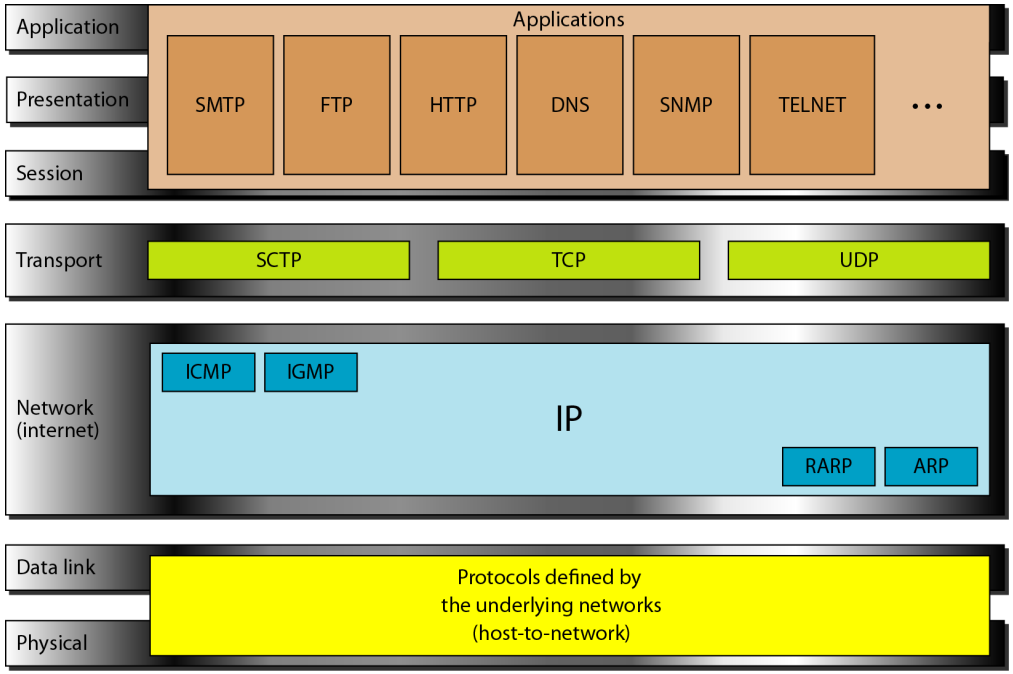
Kaan Bür

Previously on ETSF15



Data link layer

- Principles of digital communications
  - From electrical signals to bits to packets
- **Using the physical infrastructure**
  - **Network access**
- Finding your way
  - Addressing, routing
- Making use of it all
  - Applications



Data Link Layer

- Medium Access Control
  - Access to network
- Logical Link Control
  - Node-to-node error and flow control

Link layer protocols

Link layer protocols

- Error detection
  - All errors must be detected
- Error correction
  - Receiver must get correct data
- Flow control
  - Receiver must not be overloaded

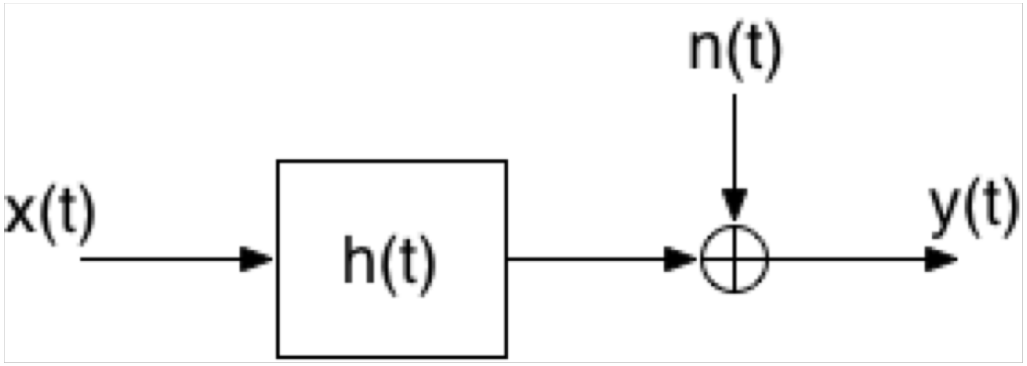
Data Link Layer (1)

Logical Link Control Sublayer

- Error detection and correction
  [S6.1-6]
- Data link control, go-back-N
  [S7.1-2]
- High-level data link control protocol
  [S7.3]

**[Kihl & Andersson: 4.1, 4.2, 4.3, 4.4]*

Error



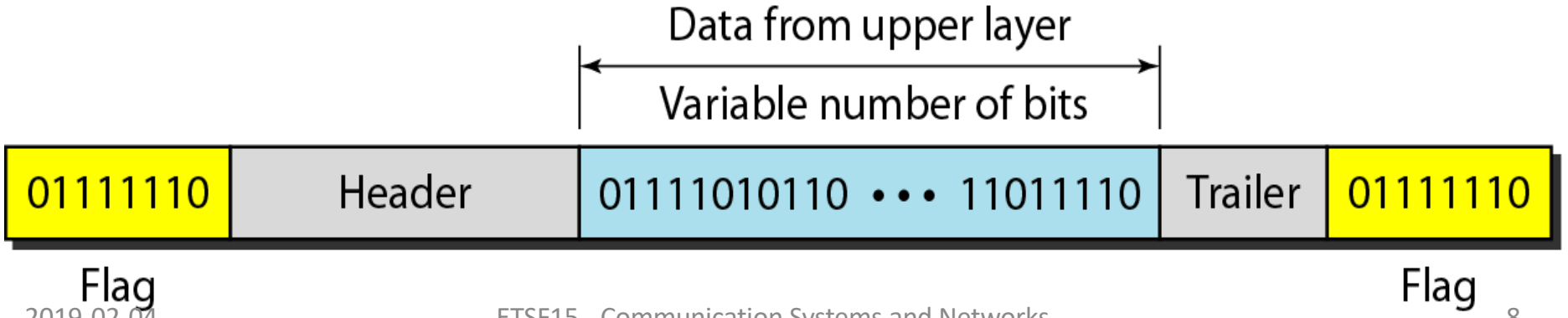
$$x(t) : \ldots 01100101001000011110 \ldots$$

$$y(t) = x(t) \text{ for all t?}$$

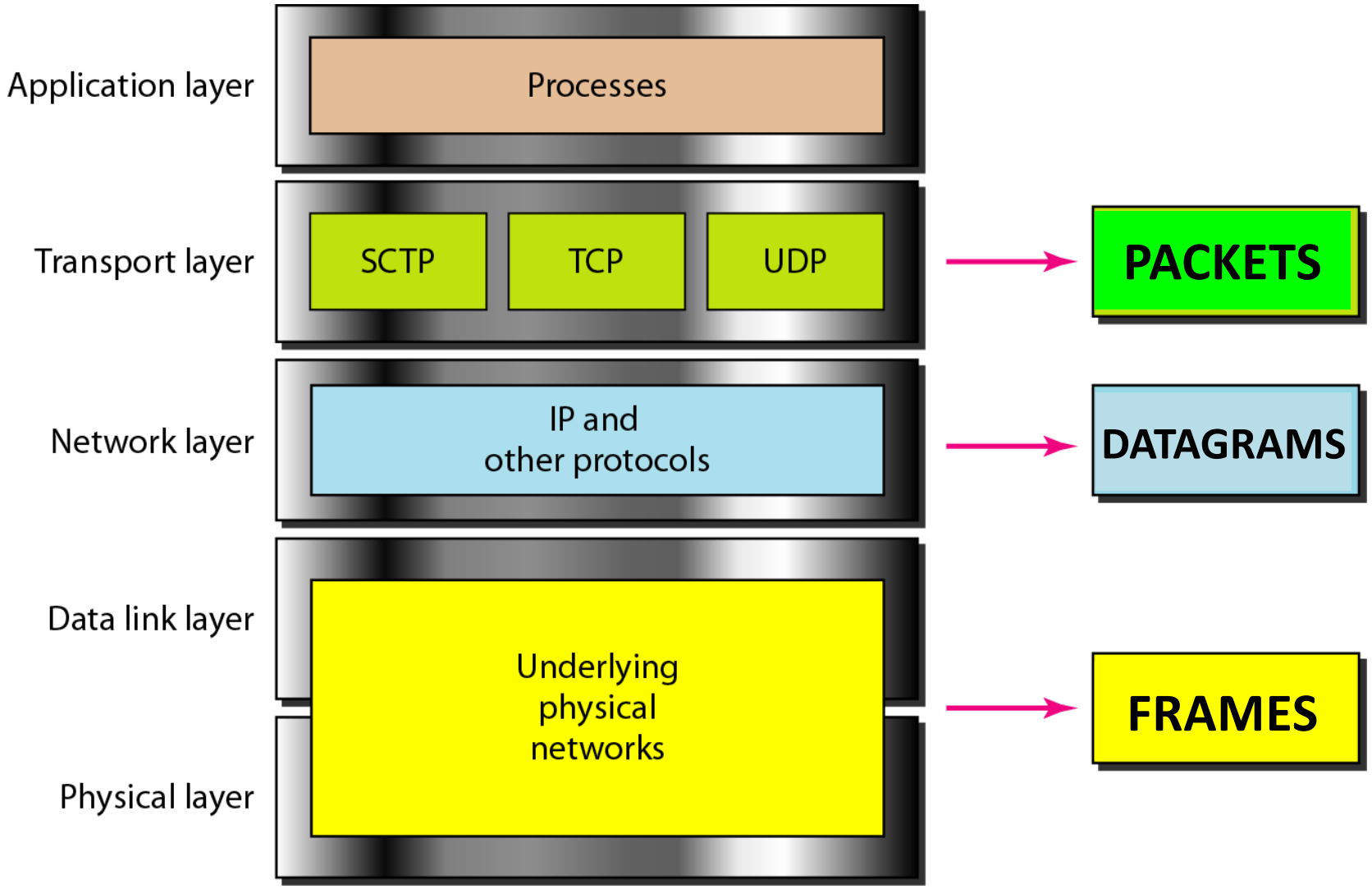
- How to find errors in transmitted data
- What to do with those errors

Framing

- Physical layer → bitstream
- Link layer → frames
- We need logical transmission units
  - Synchronisation points
  - Switching between users
  - Error handling

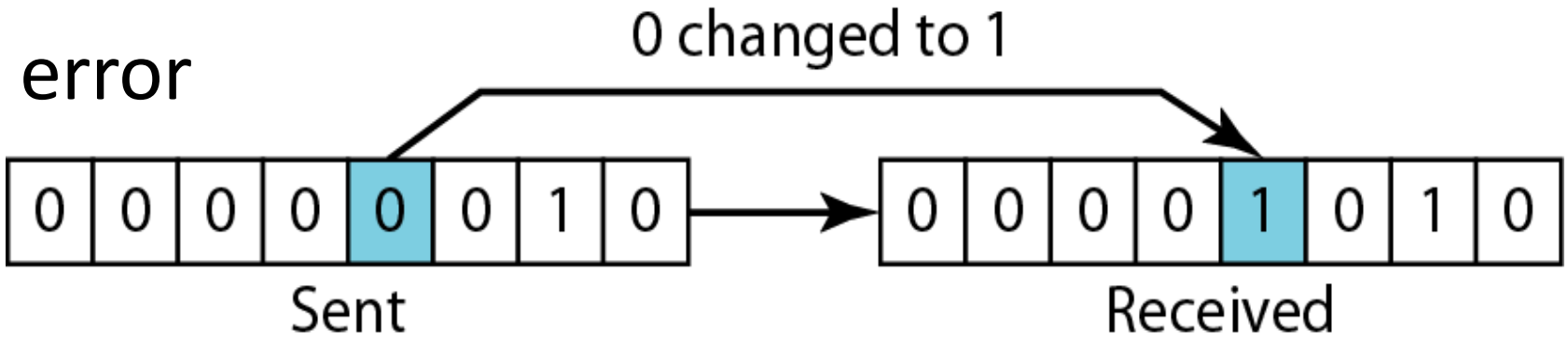


TCP/IP model and data units

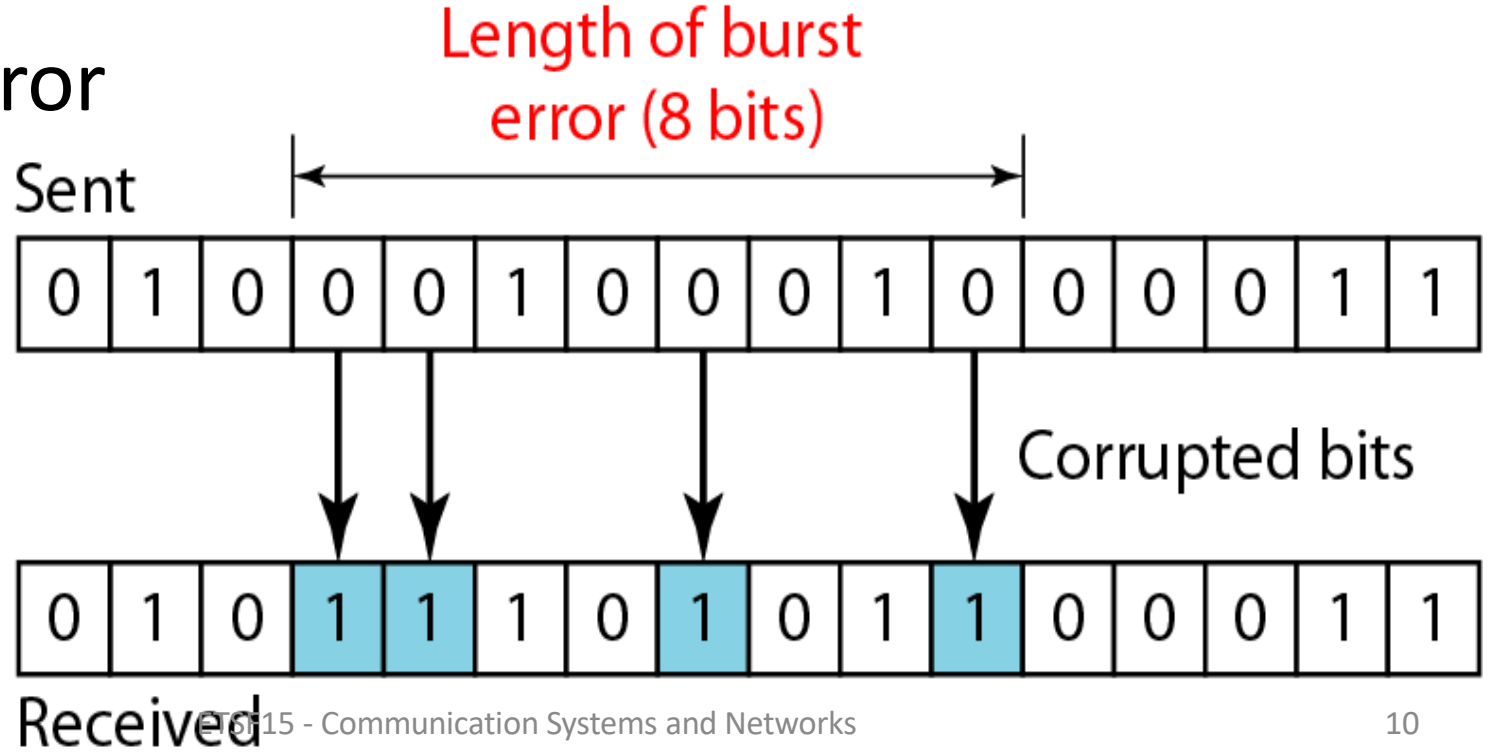


Error types

- Bit error

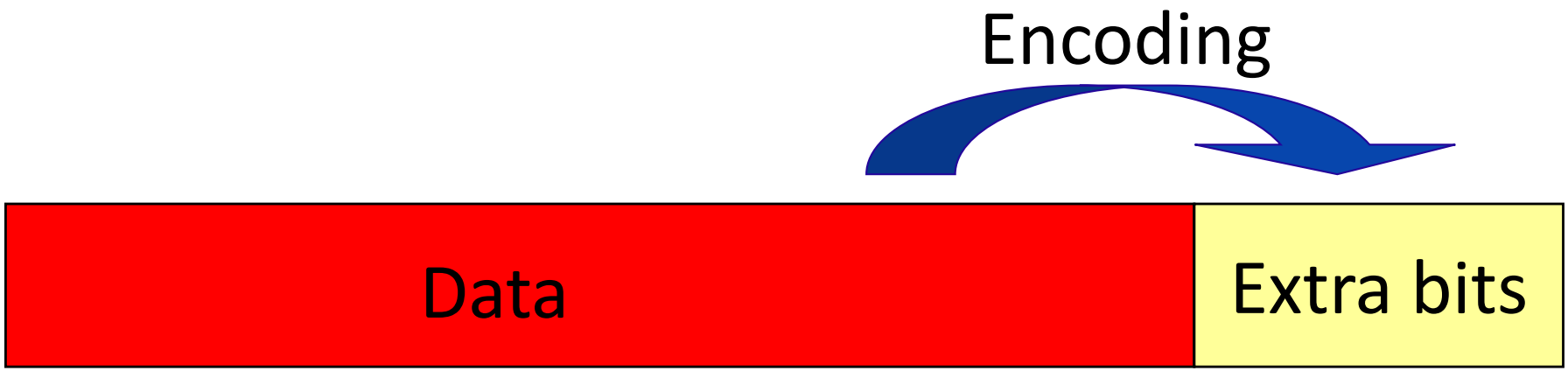


- Burst error

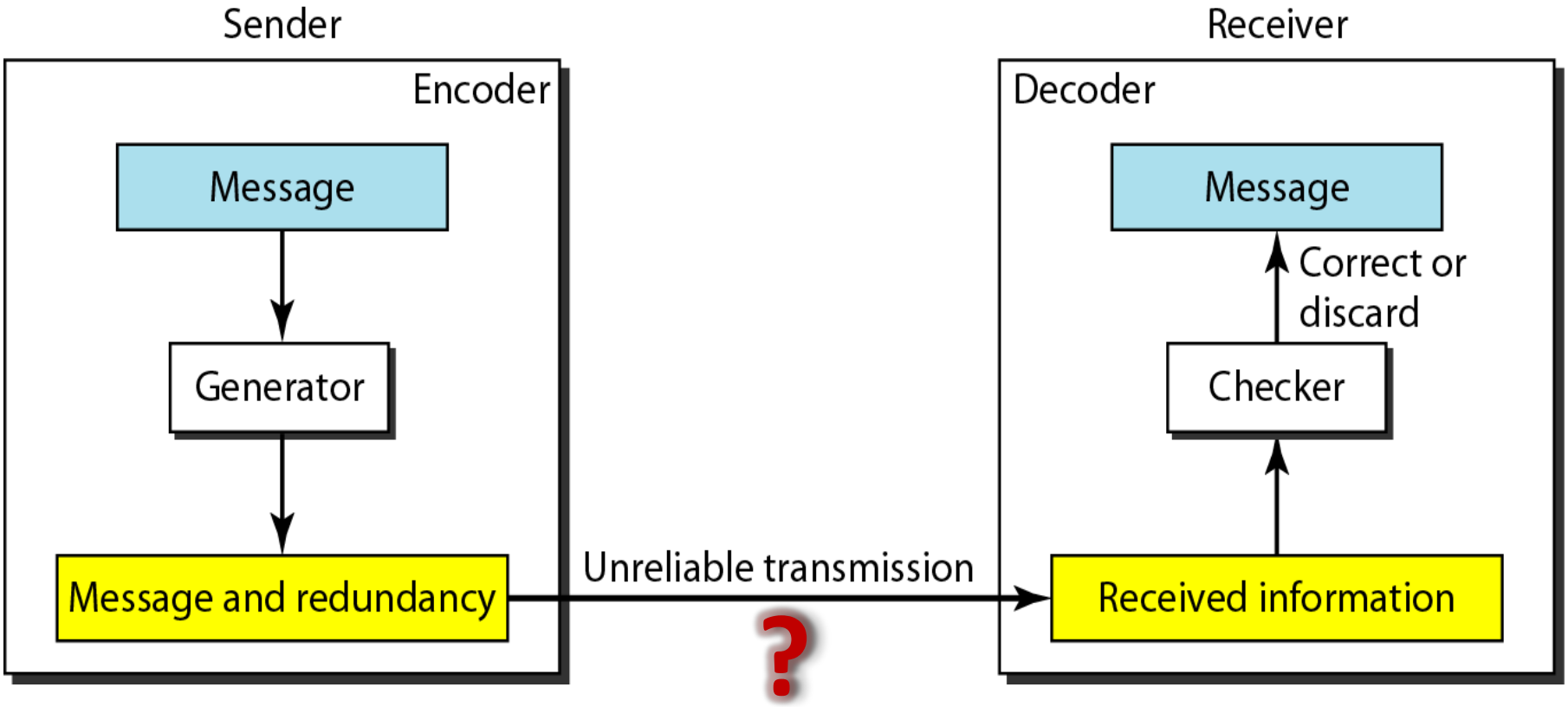


Error control

- Data assumed error-free by higher layers
  - Errors occur at lower layers (physical)
  - Job for LLC layer
- Extra (redundant) bits added to data
  - Generated by an encoding scheme from data



Error detection process

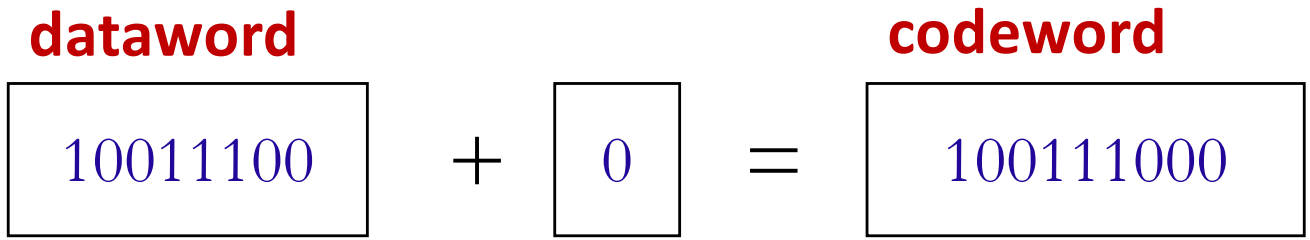


Error detection schemes

- Simple parity-check code
- Cyclic Redundancy Check (CRC)
- Checksum

Simple Parity-Check Code

- Extra bit added to make the total number of 1s in the codeword
  - Even → even parity
  - Odd → odd parity



- Problem?
  - Can detect an odd number of errors

Block coding

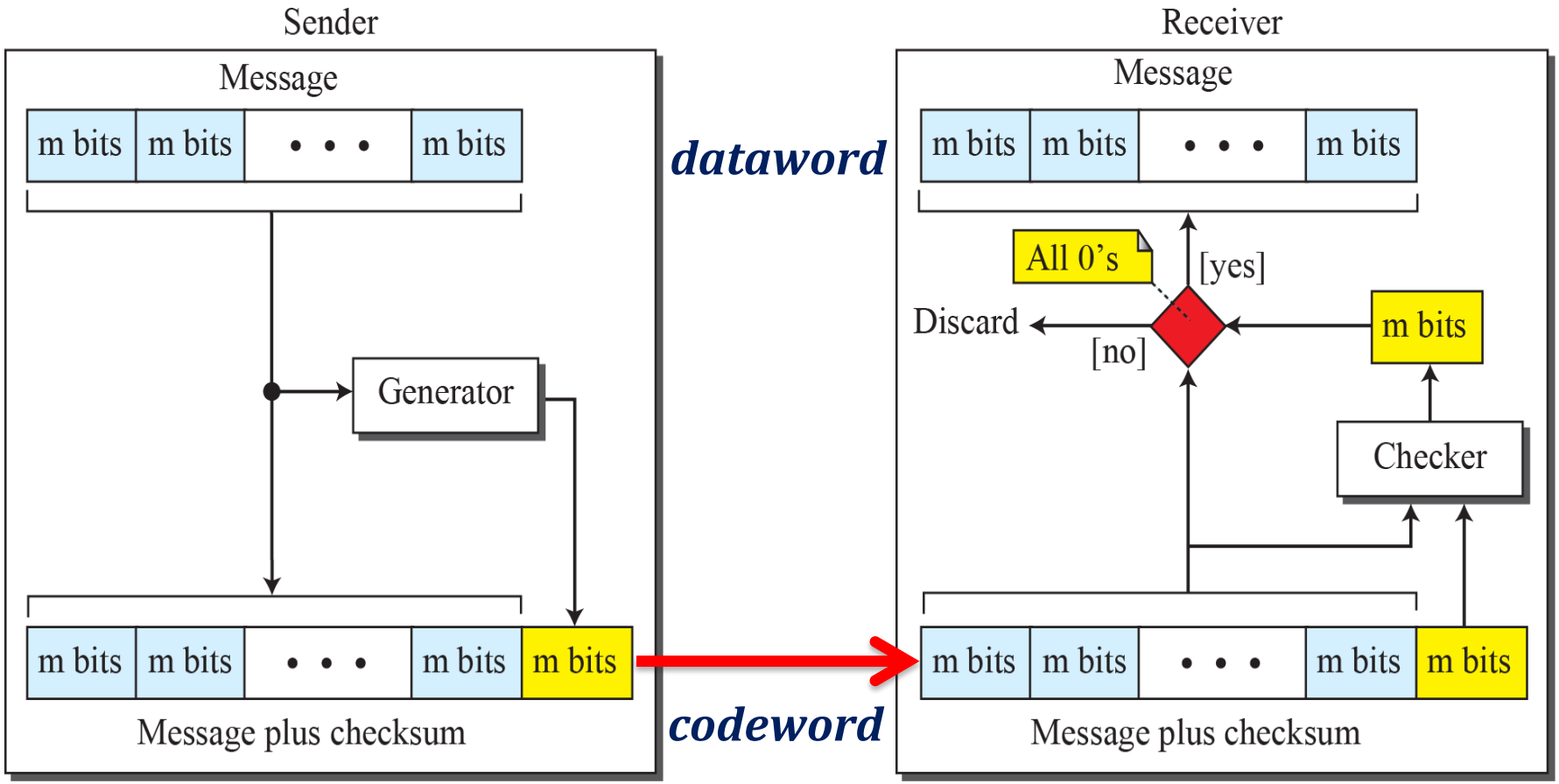
- Divide the message into *k*-bit blocks, called **datawords**.
- Add *r* redundant bits to each block. The resulting *n*-bit blocks (*n*=*k*+*r*) are called **codewords**.
- The code rate is *R*=*k*/*n*.

Checksum

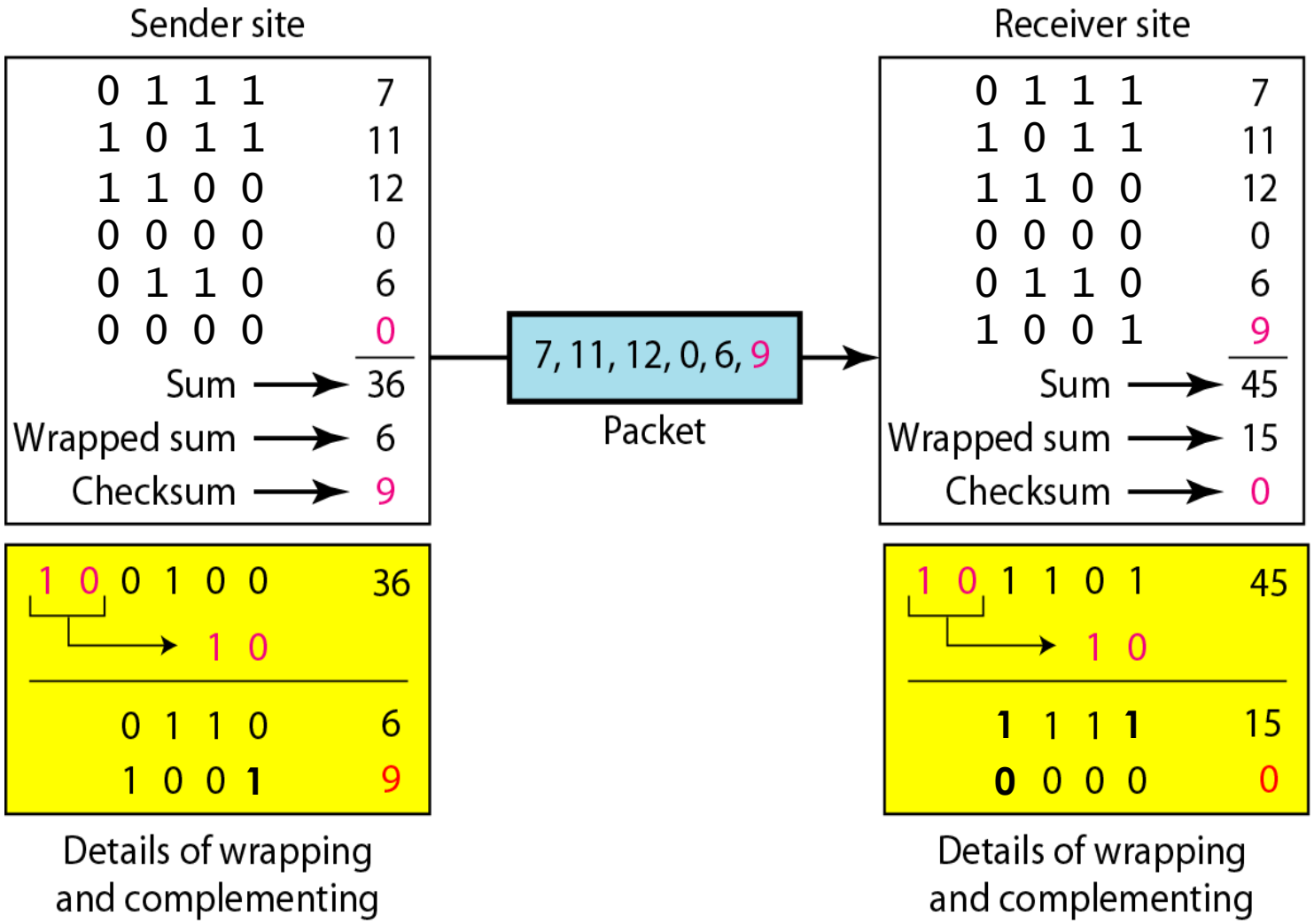
- The checksum is used in the Internet by several protocols although not at the data link layer.

- The main principle is to divide the data into segments of n bits. Then add the segments and use the sum as redundant bits.

Checksum process

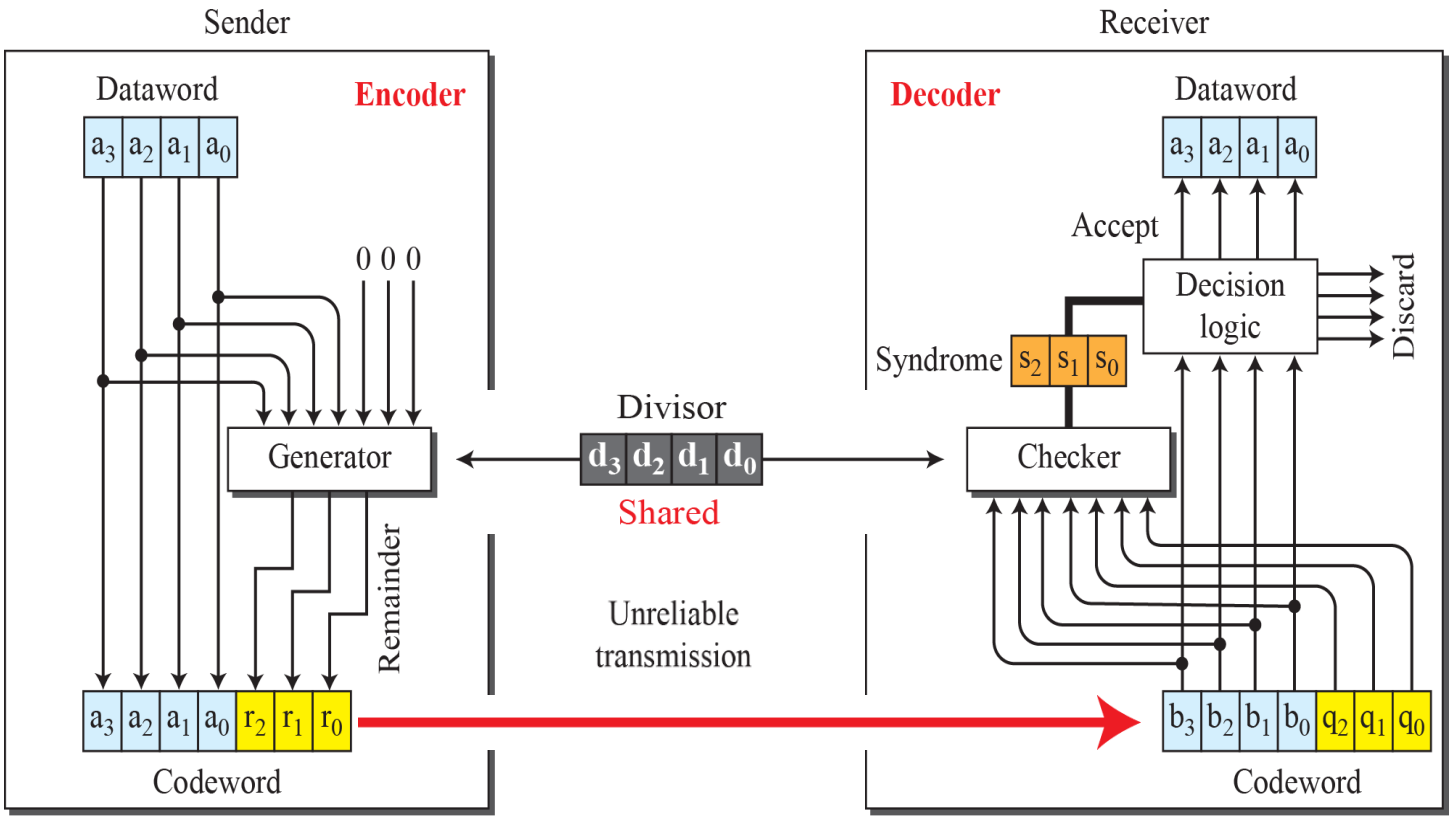


Example: Checksum



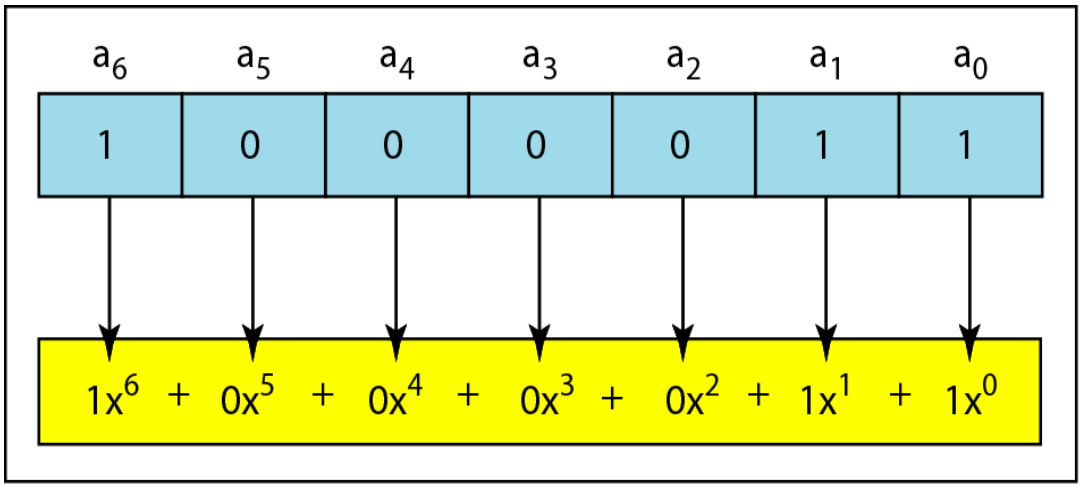
Cyclic Redundancy Check (CRC)

- Predefined shared *divisor* to calculate codeword

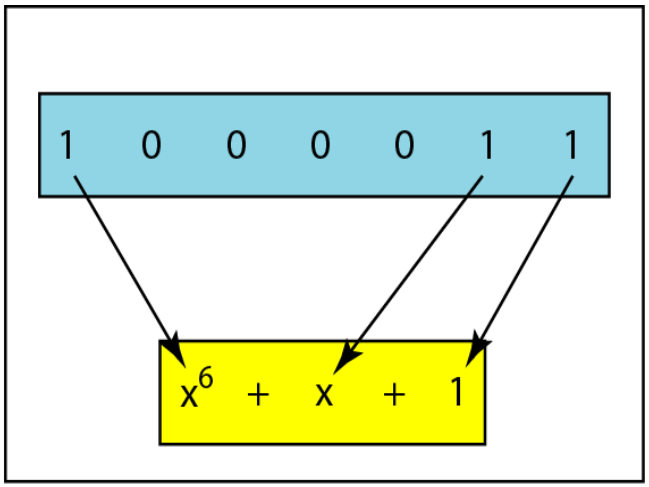


CRC: Polynomial representation

- The dataword of *k* bits is represented by a polynomial, *d(x)*.
- The degree of the polynomial is *k*-1.



a. Binary pattern and polynomial



b. Short form

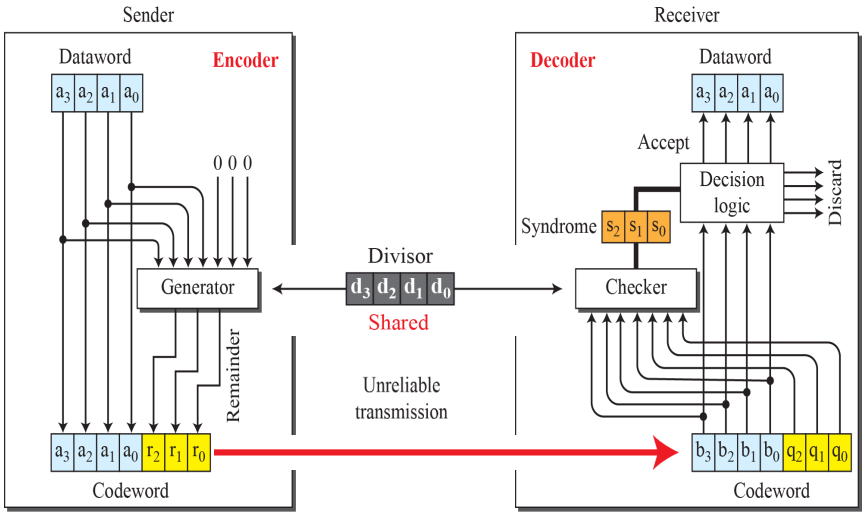
CRC: The principle

- **Objective**: Send a dataword ***d(x)*** of k bits represented by a polynomial of degree ***k*-1**.
- **Given**: Generator polynomial ***g(x)*** of degree ***m***.
- **Find**: Remainder polynomial ***r(x)*** such that:

  $\mathbf{c(x) = d(x) \cdot x^m + r(x)}$

  can be divided by g(x) without remainder.
- Codeword ***c(x)*** will then be sent to the receiver.
- ***r(x)*** has degree ***m*-1** or less, and CRC has ***m*** bits.

CRC: How it works



- Sender:
  1. Generate $b(x) = d(x) \cdot x^m$
  2. Divide b(x) by g(x) to find r(x)
  3. Send c(x) = b(x) + r(x)
- Receiver:
  1. Divide c'(x) = c(x) + e(x) by g(x)
  2. Check remainder r'(x) → if 0 data correct, c(x) = c'(x)
  3. Remove CRC bits from codeword to get dataword

Example: CRC derivation

- For dataword **1001**, derive CRC using generator **1011**.

- Data polynomial: $d(x) = x^3+1$
- Generator polynomial: $g(x) = x^3+x+1$
- Dividend: $b(x) = d(x)\cdot x^3 = x^6+x^3$
- Codeword polynomial: $c(x) = d(x)\cdot x^3 + r(x)$
- CRC polynomial: $r(x) = ?$

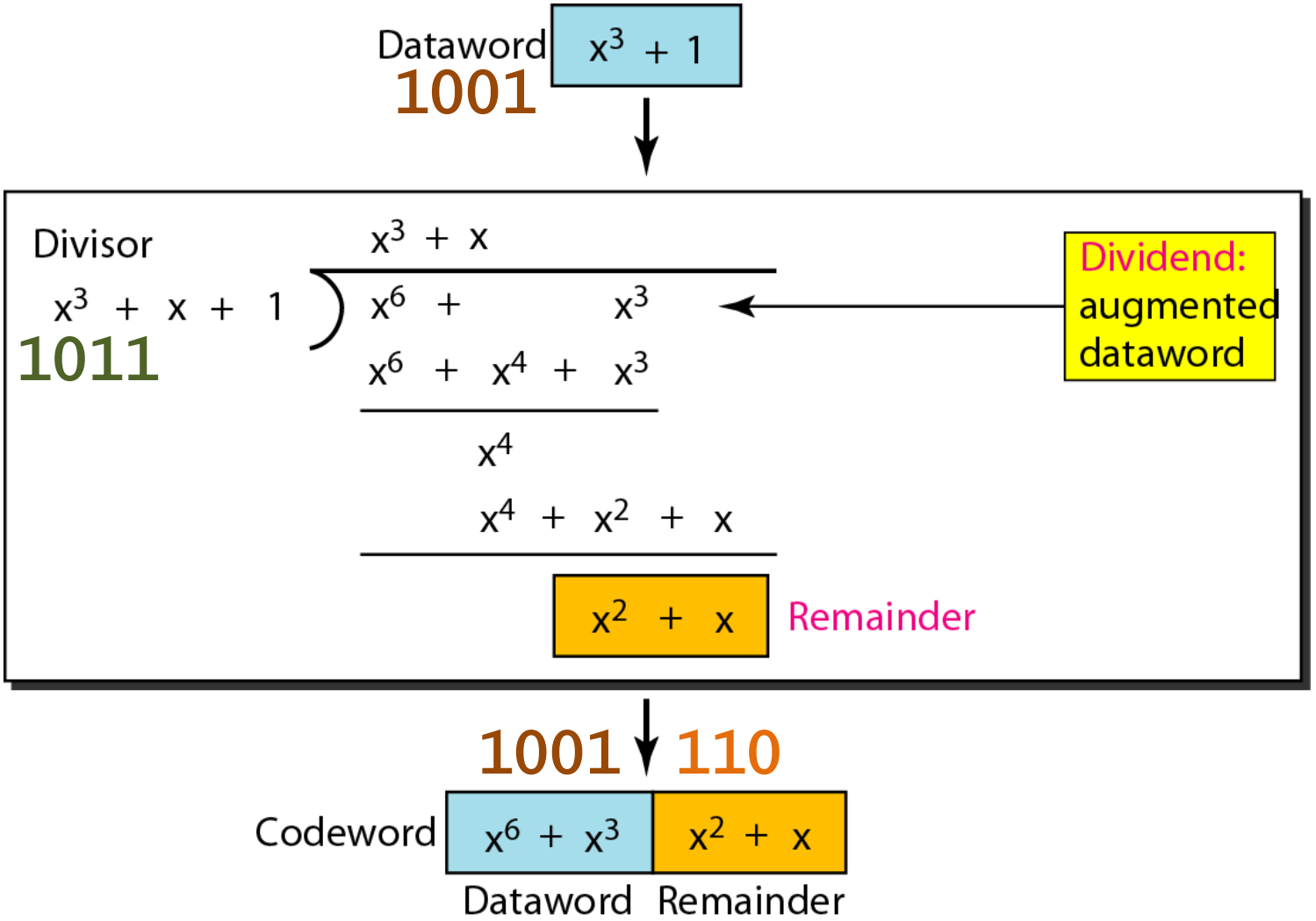
Modulo 2 arithmetic

$\oplus$	0	1
0	0	1
1	1	0

$\otimes$	0	1
0	0	0
1	0	1

$$1 \oplus 1 = 0$$
$$1 - 1 = 0$$

Example: CRC derivation



Some standard CRC polynomials

Name	*Polynomial*	*Used in*
CRC-8	$x^8 + x^2 + x + 1$ **100000111**	ATM header
CRC-10	$x^{10} + x^9 + x^5 + x^4 + x^2 + 1$ **11000110101**	ATM AAL
CRC-16	$x^{16} + x^{12} + x^5 + 1$ **1000100000100001**	HDLC
CRC-32	$x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} + x^{10} + x^8 + x^7 + x^5 + x^4 + x^2 + x + 1$ **10000010011000001000111011011011 1**	LANs

Error Correction

Two alternative ideas

- Forward Error Correction (FEC)
  - Send each bit multiple times
  - Decode according to majority decision
- Retransmission
  - Resend the entire frame

See you in 15’ :)

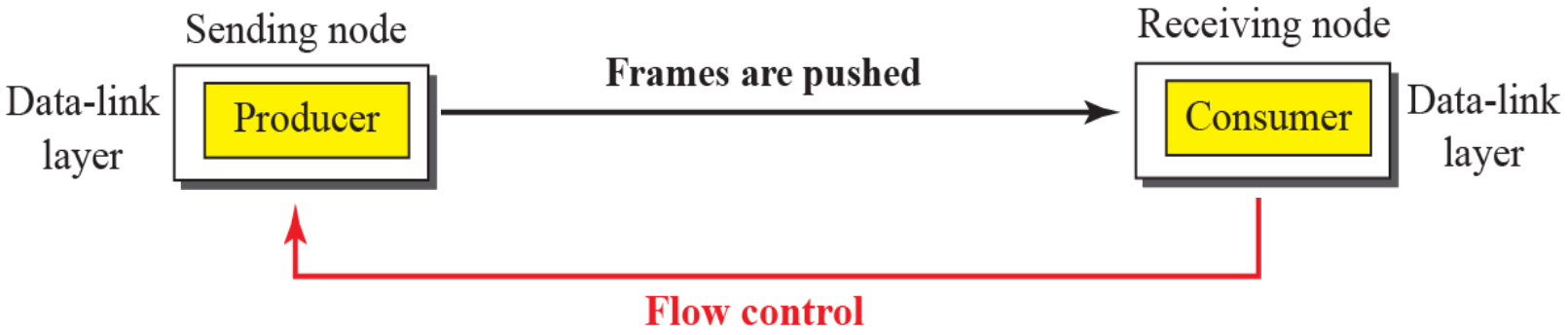
- After the break
  - Data link control
  - HDLC

Error and flow control

- The basic principle in error and flow control is that the receiver **acknowledges** all correctly received packets.

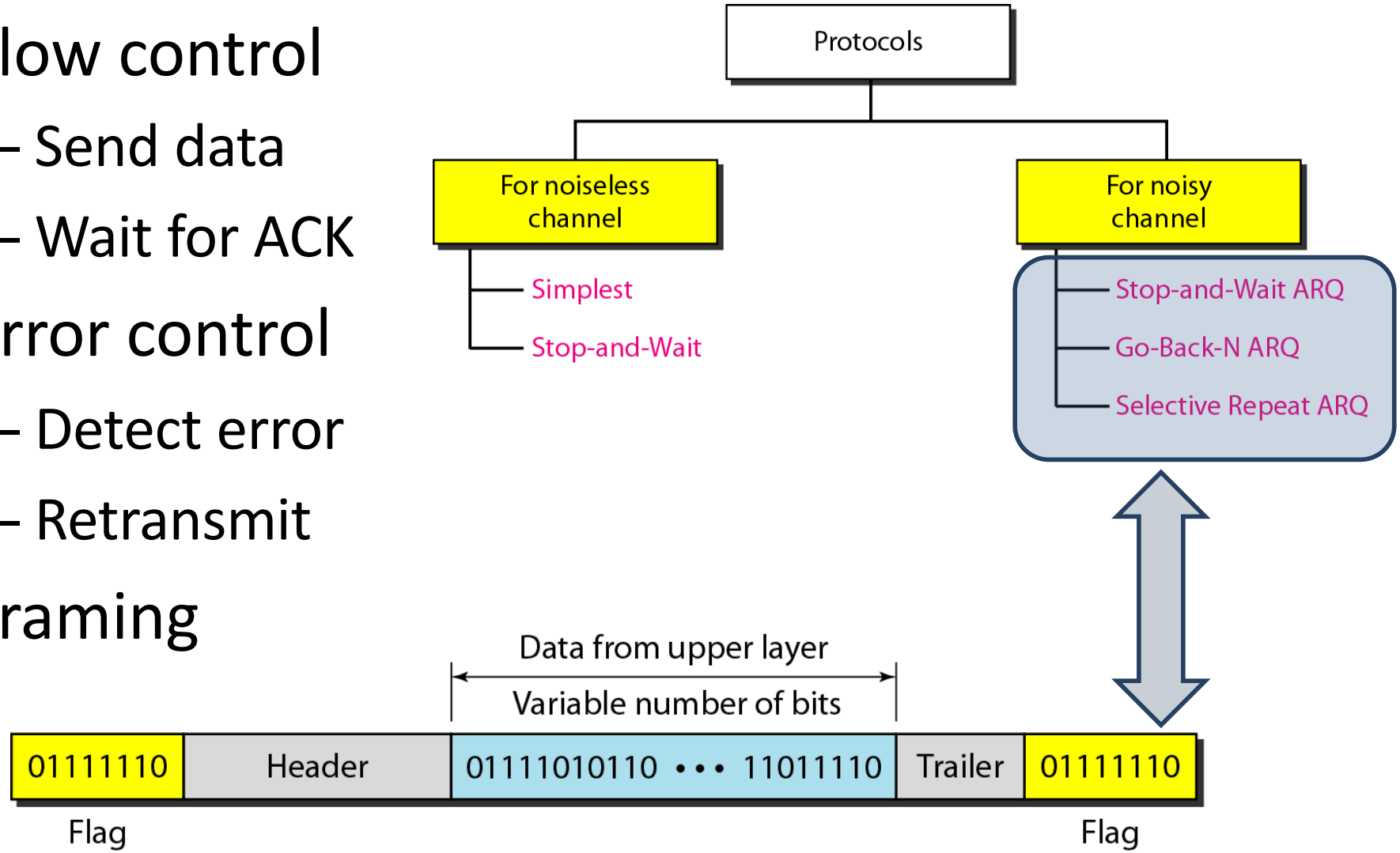
The need for flow control

- The receiver must be able to handle all recieved frames. If the transmission rate is too high, the receiver may become overloaded and drop frames due to full buffers.



Data link control protocols

- Flow control
  - Send data
  - Wait for ACK
- Error control
  - Detect error
  - Retransmit
- Framing

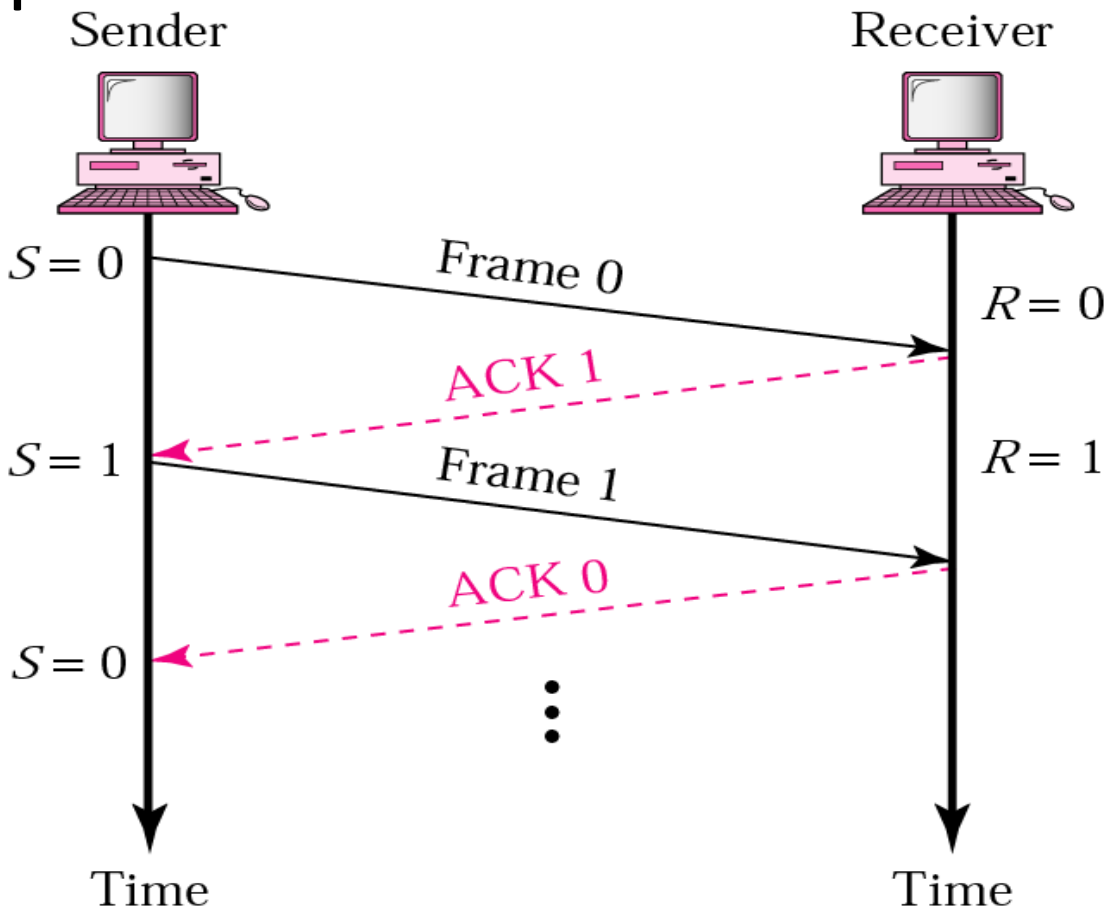


Stop-and-wait ARQ

- Send and wait
  - Keep time
  - Wait for ACK
  - Retransmit
- Automatic repeat request
  - Frames (SEQ++)
  - Acknowledgements (SEQ+1)
  - Mismatch = problem!

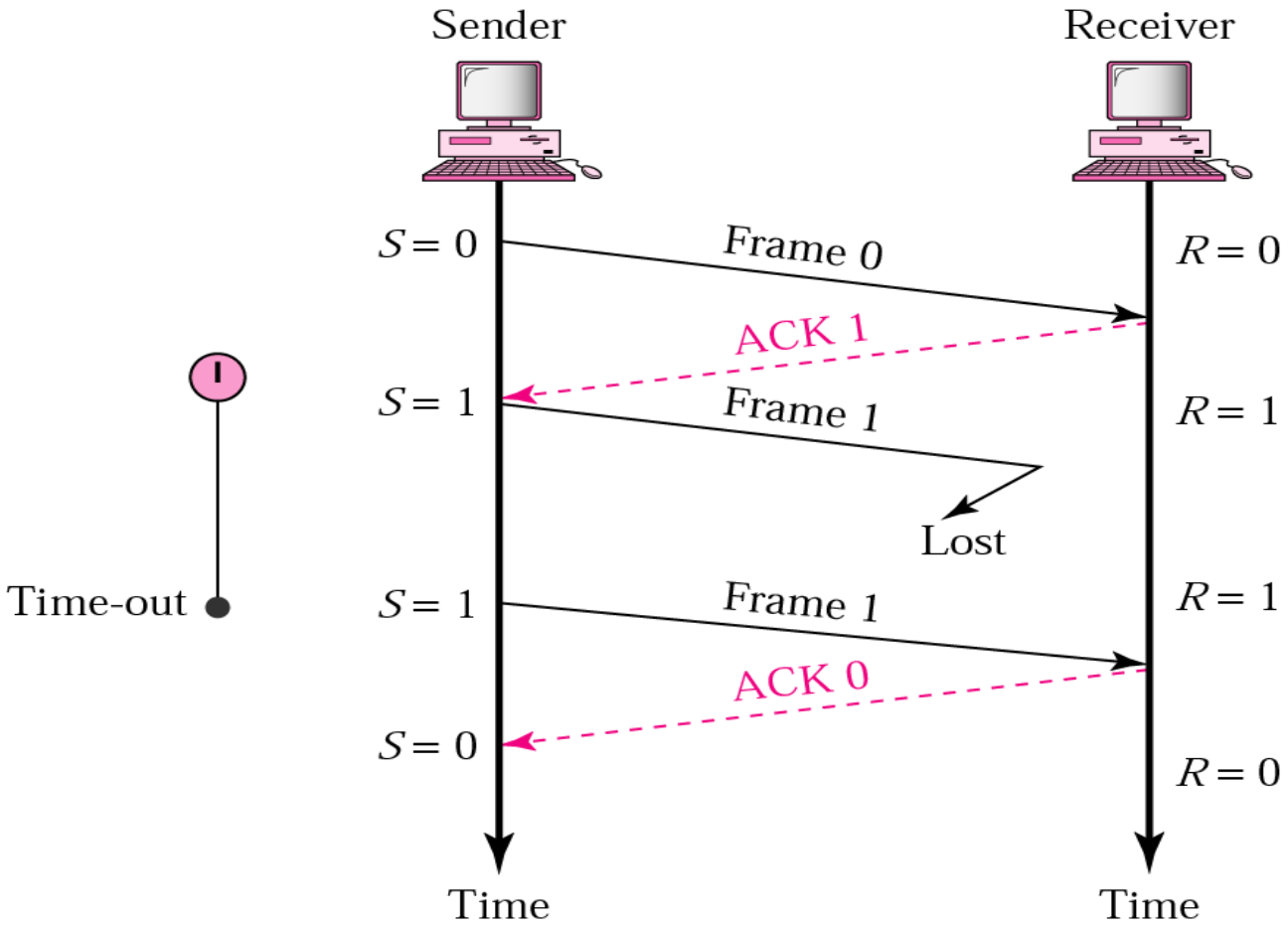
Stop-and-wait ARQ

- Normal operation



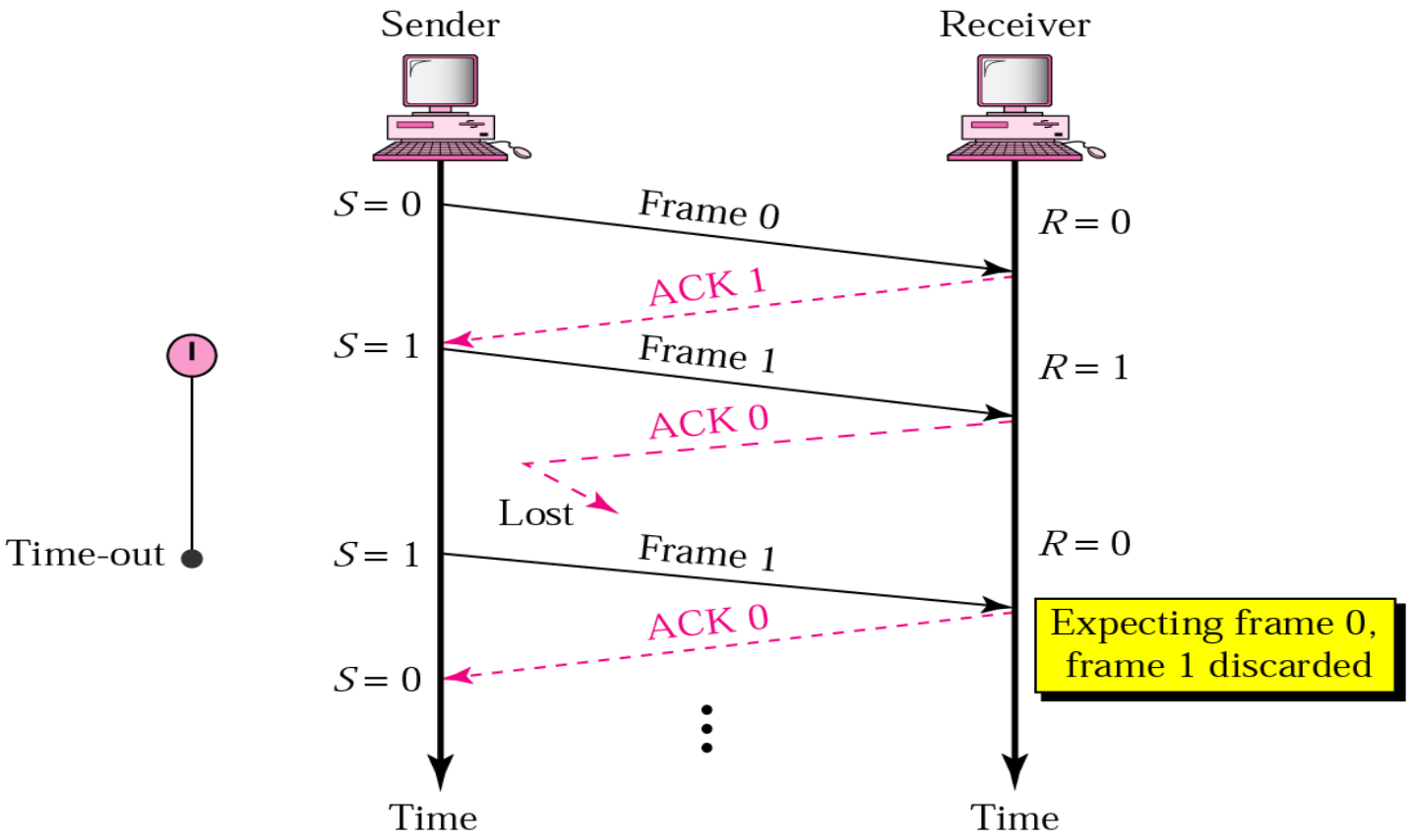
Stop-and-wait ARQ

- Frame lost



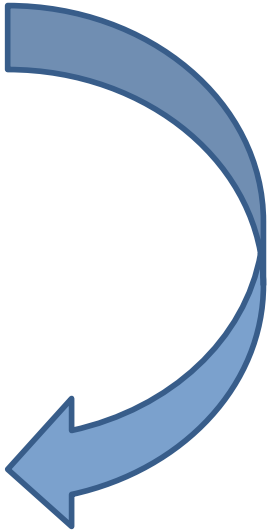
Stop-and-wait ARQ

- ACK lost



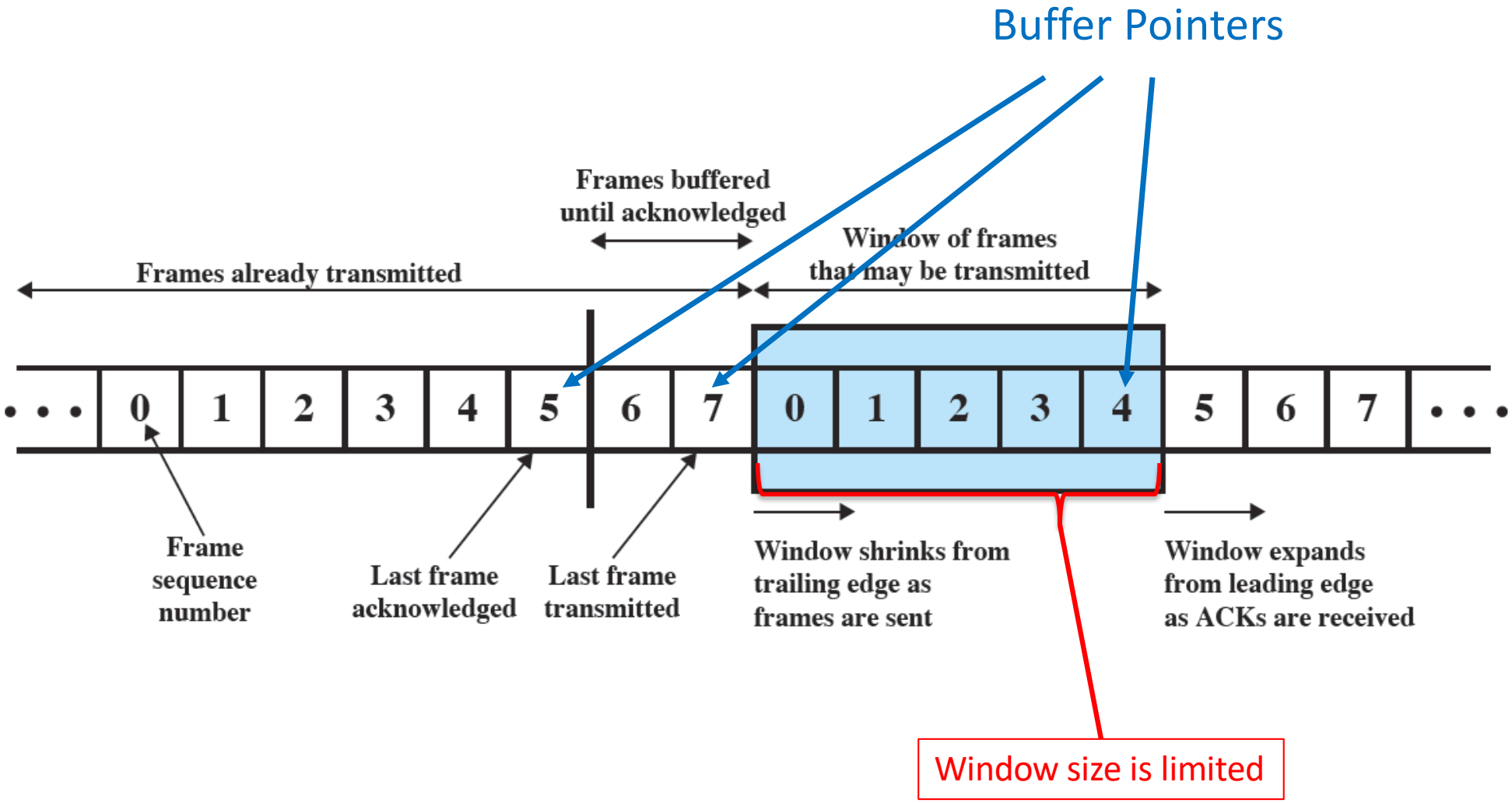
Stop-and-wait ARQ inefficiency

- Too much waiting
- Solution
  - Keep the pipe full
  - But not too full



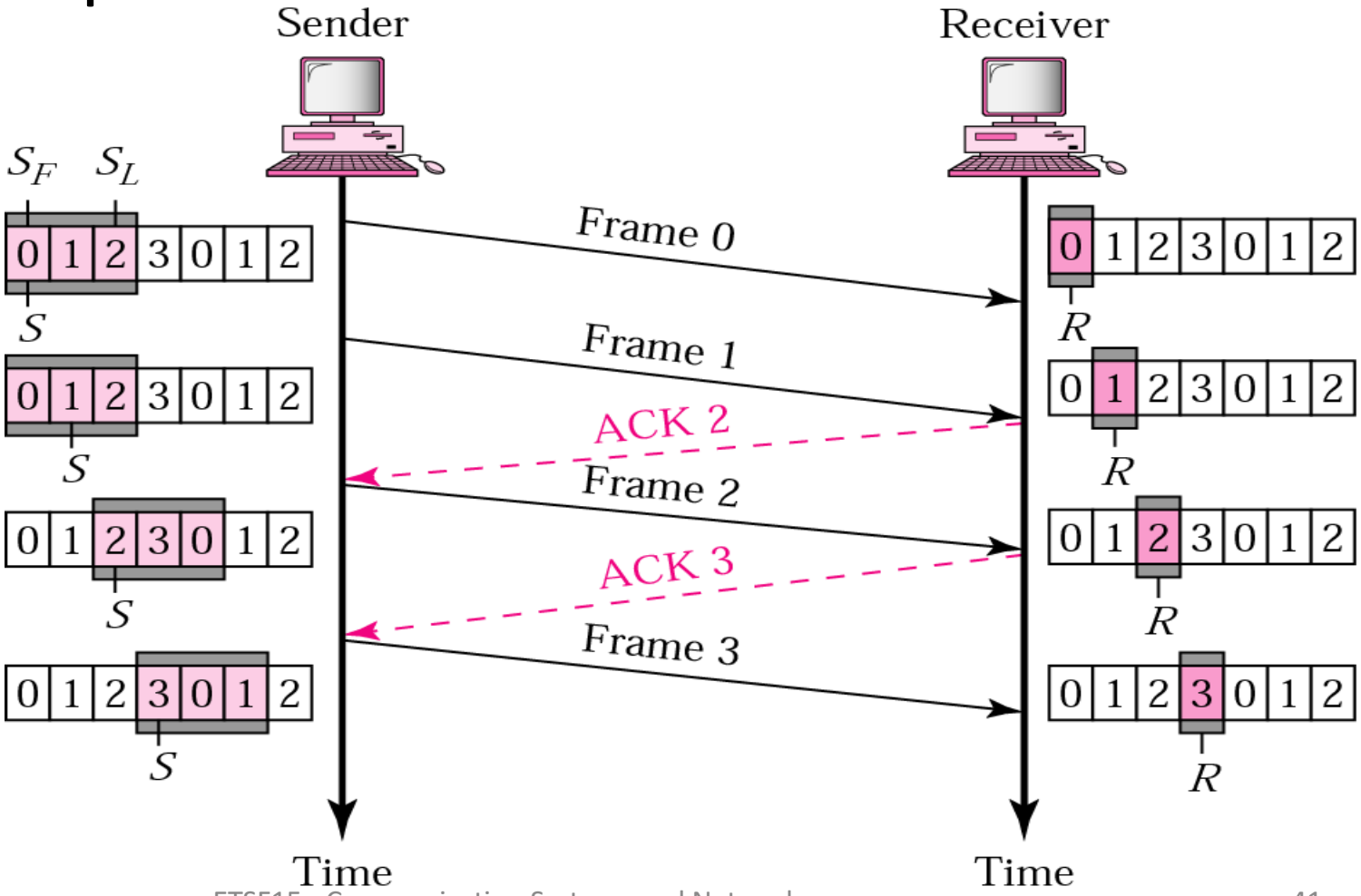
- Sliding window
  - Size matters
  - Window size < 2^m

Sliding window



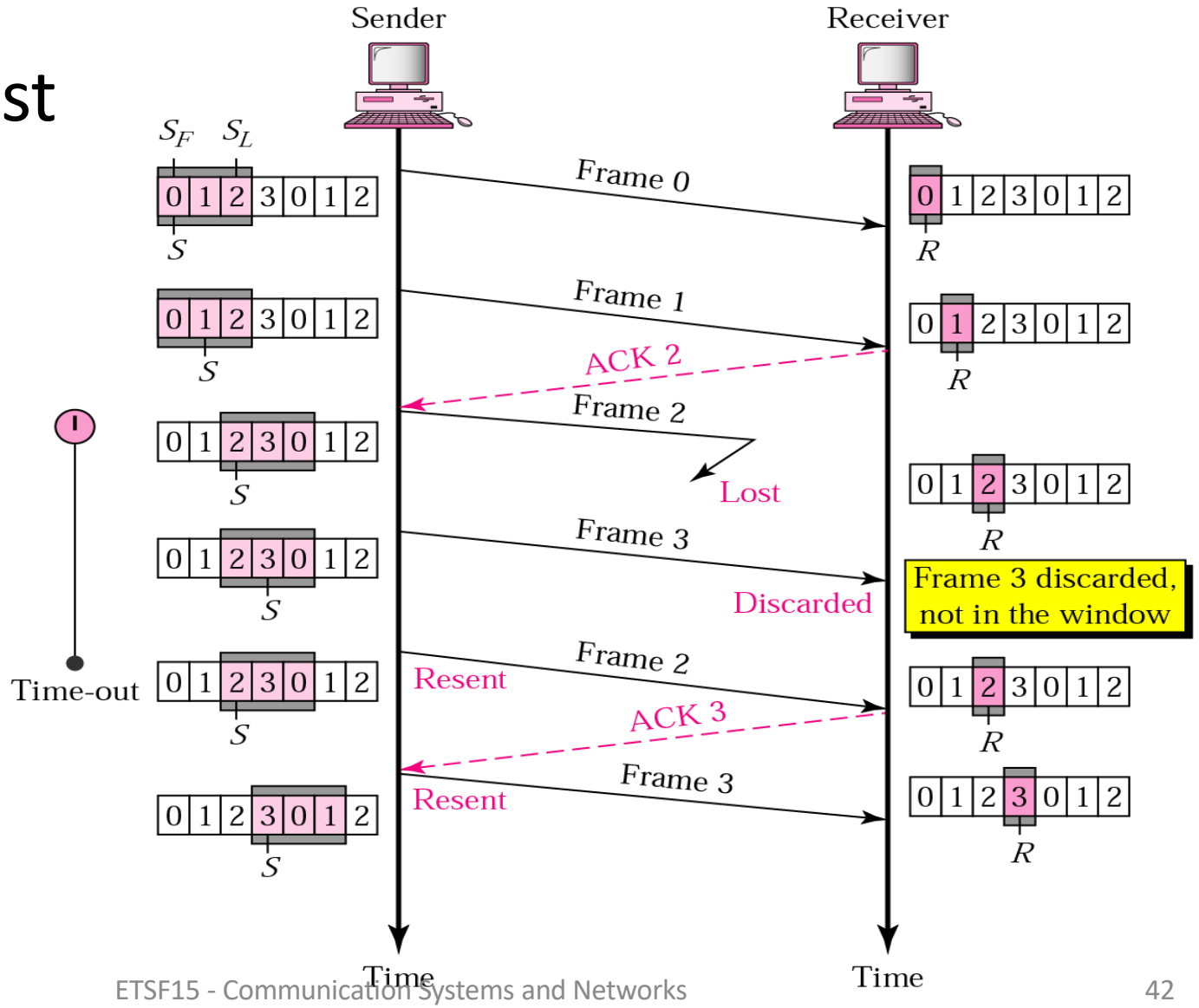
Go-back-N ARQ

- Normal operation

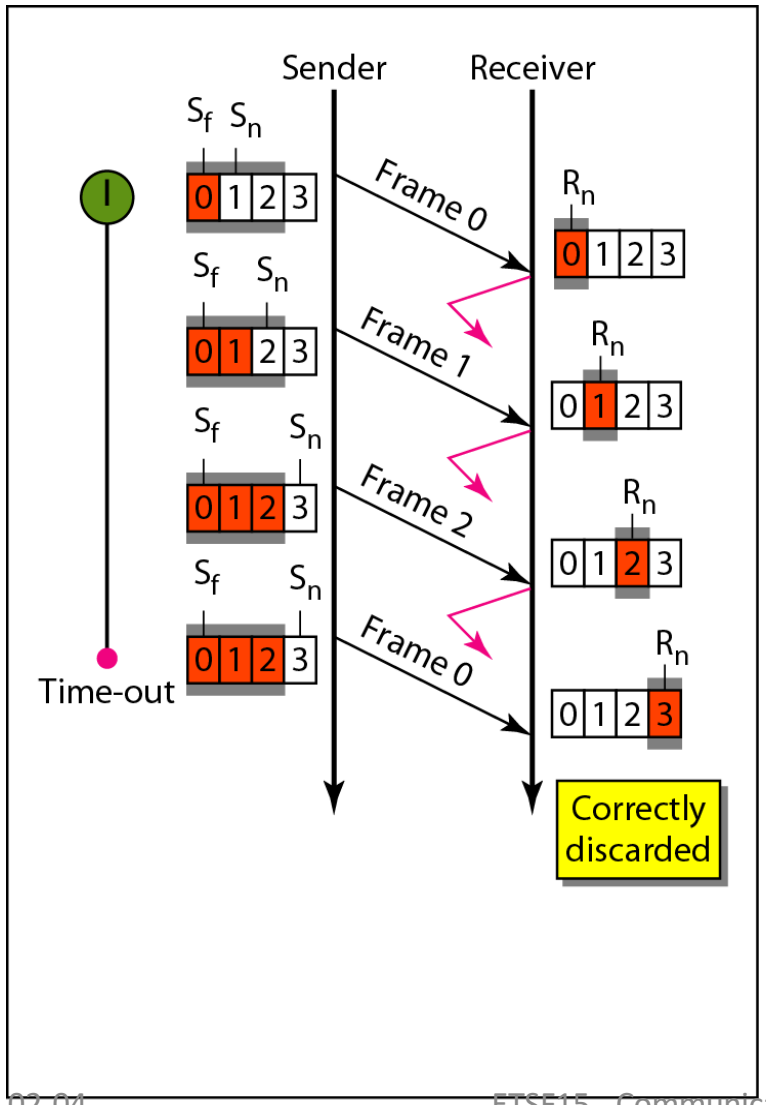


Go-back-N ARQ

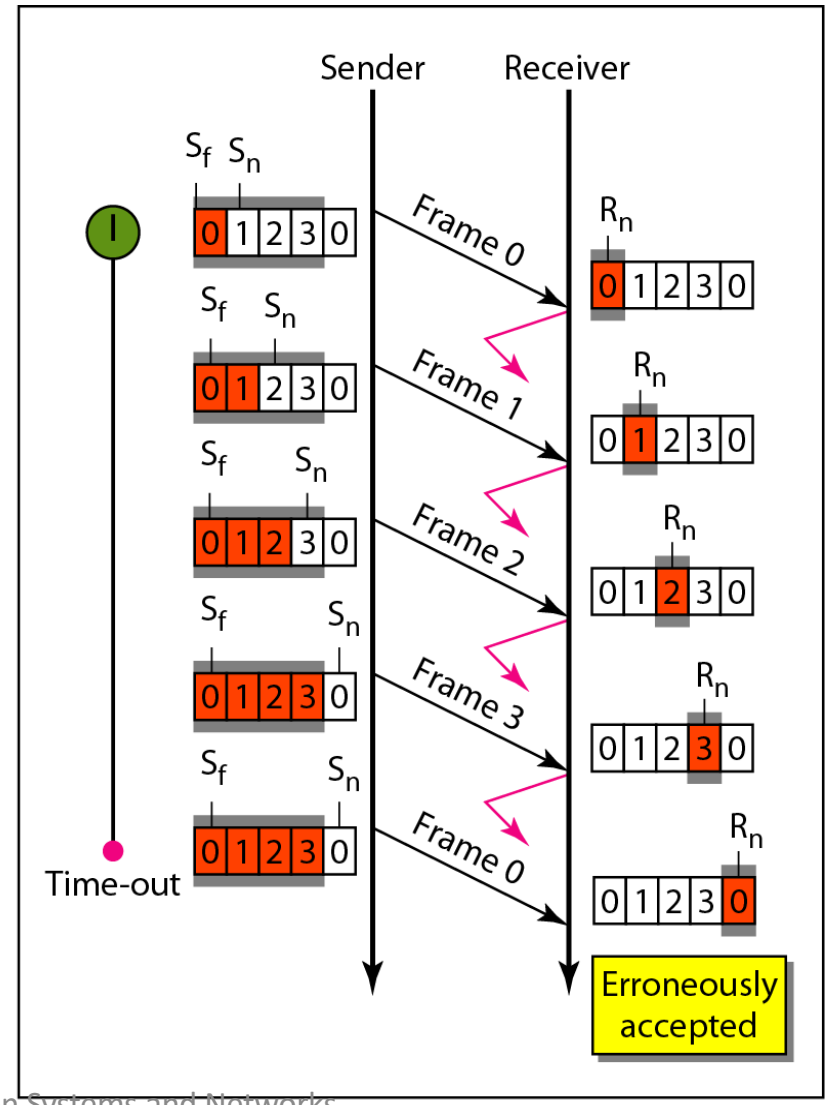
- Frames lost



Go-back-N ARQ window size



a. Window size < 2^m

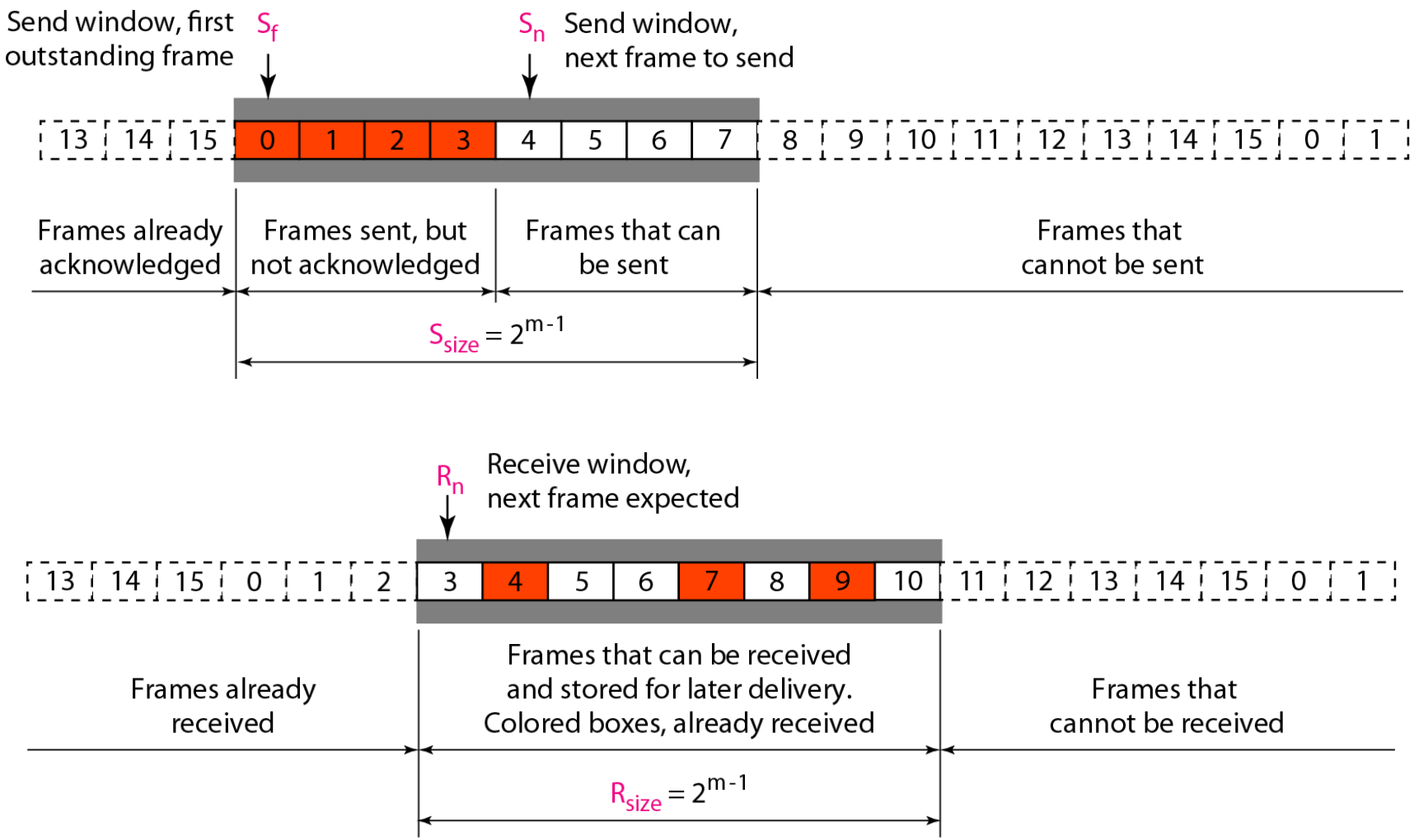


b. Window size = 2^m

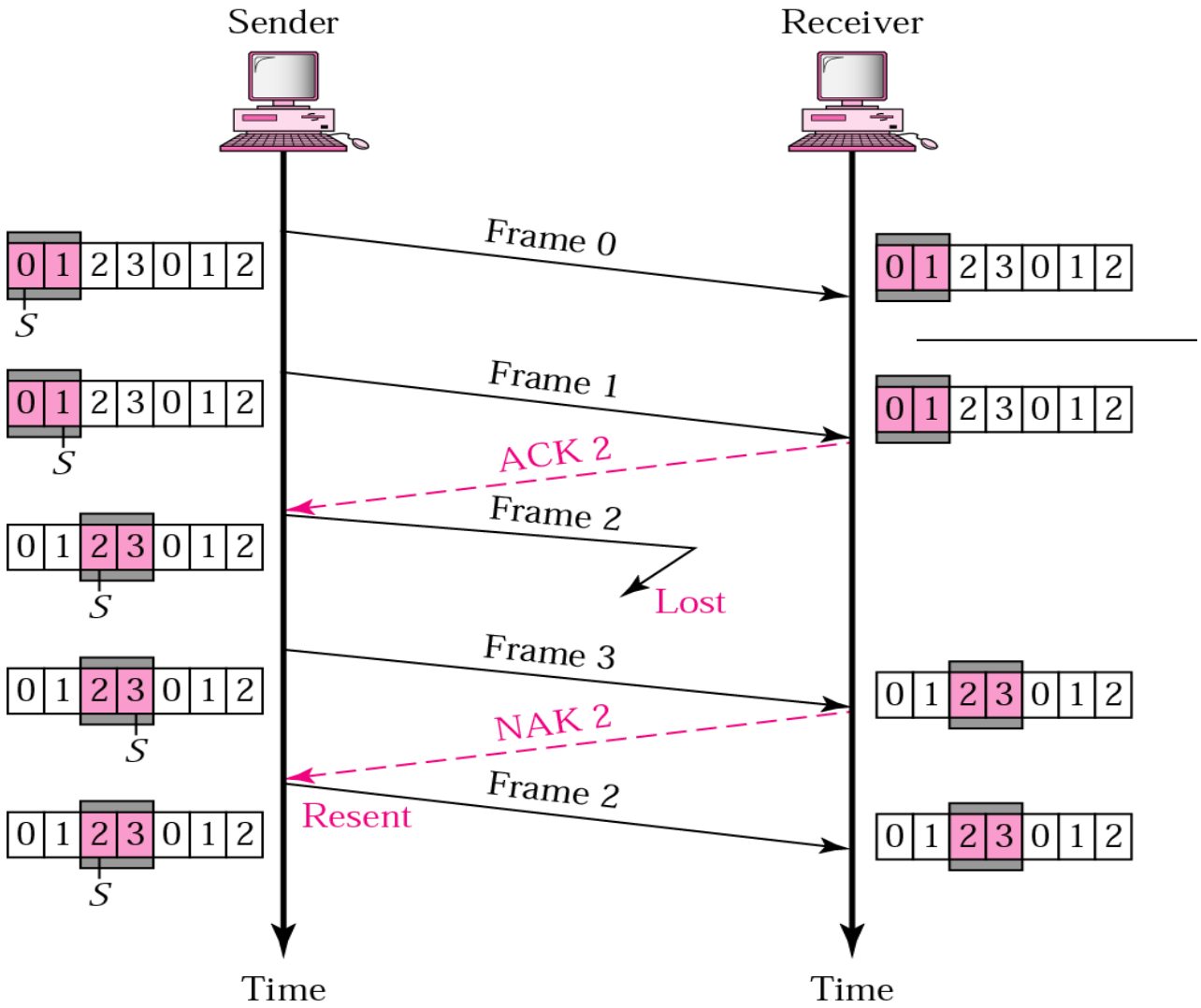
Selective repeat ARQ

- Why?
  - Too many retransmissions
- What if?
  - Just send lost frames
- Higher efficiency
  - Higher receiver complexity

Windows again

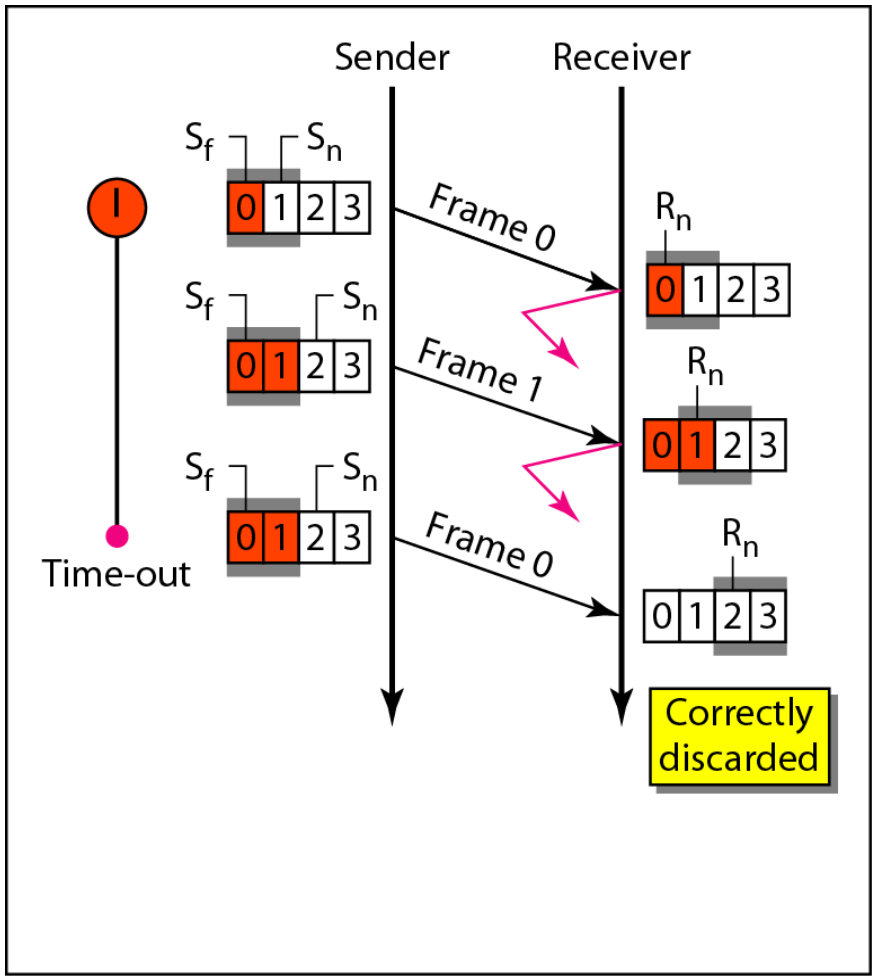


Selective Repeat ARQ

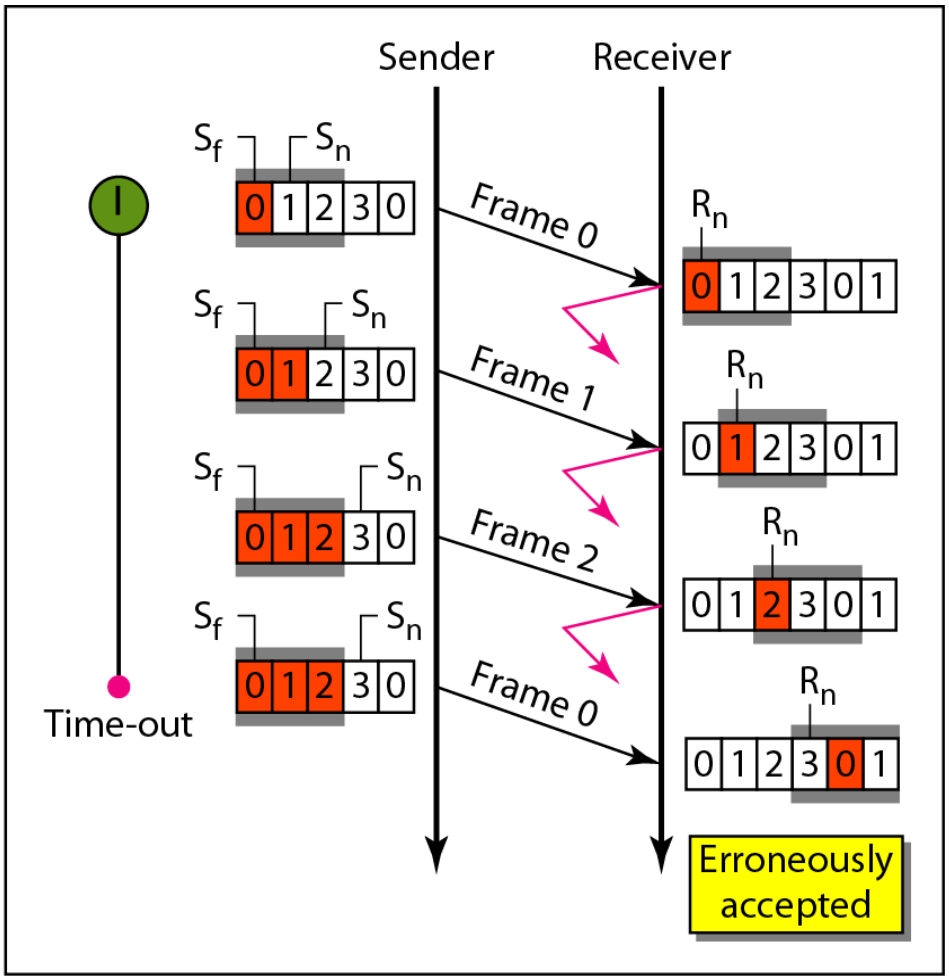


Note: Receiver sliding window size

Selective repeat ARQ window size

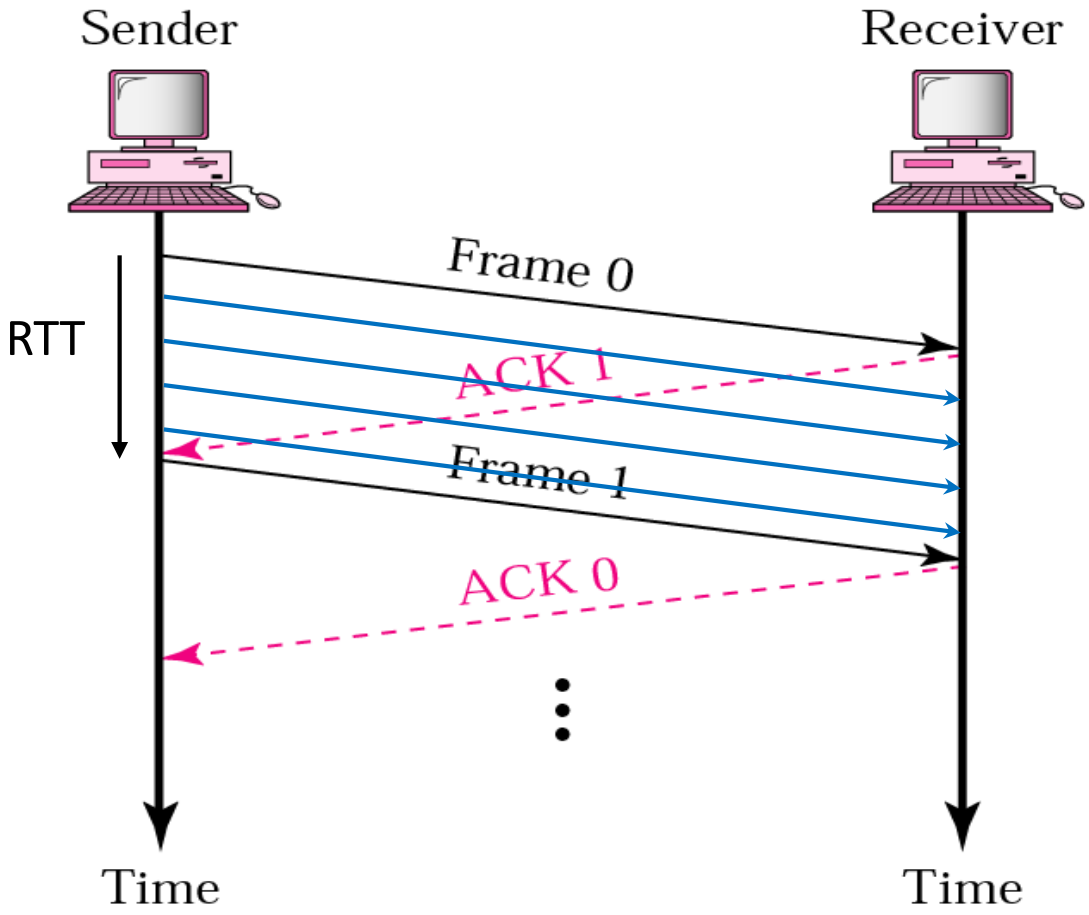


a. Window size = 2^{m-1}



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

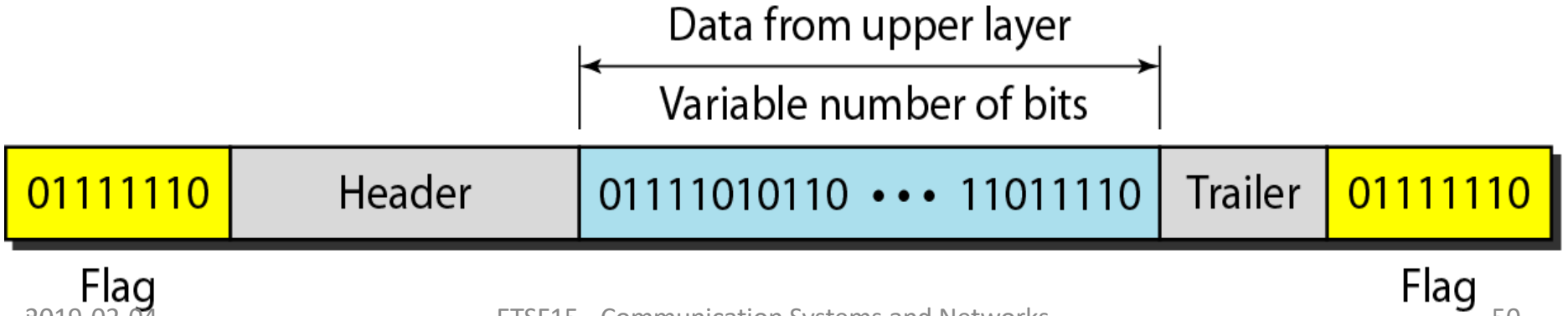
Flow control



- Send frames until ACK for first frame is expected → Round-trip time (RTT)
- Sliding window size = f(RTT)

Framing

- Header:
  - Sequence and ACK numbers
  - More to come …
- Trailer
  - CRC
- Flags?

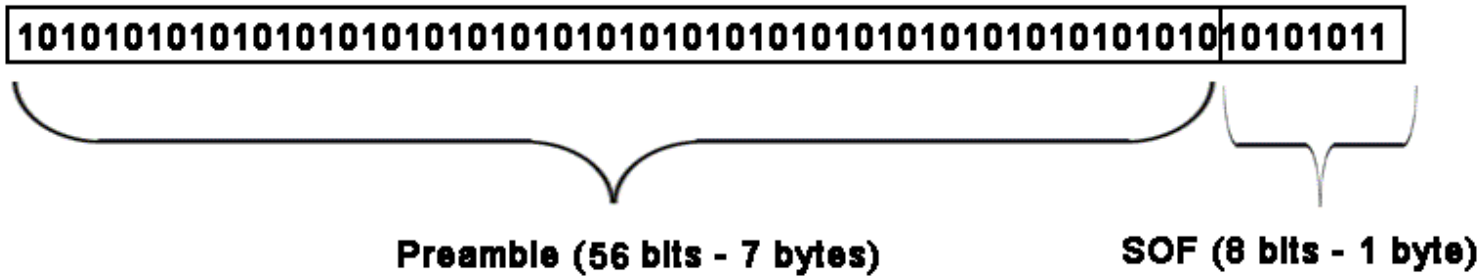


Synchronisation

Preamble and start flag	Frame	End flag

Receiver has to synch to signal of a frame

Ethernet

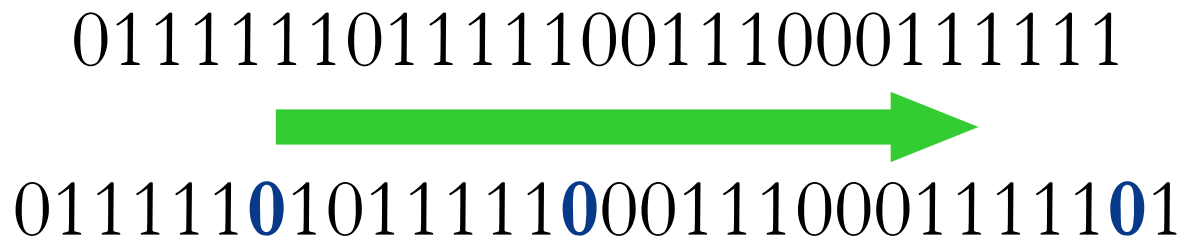


Finding the flags

- Corrolate incoming bit pattern with known flag
- End flag: we have a problem!
  - End flag bit pattern = data bit pattern?

Bit stuffing

- Given: Flag = 01111110
- Task: Avoid 6 consecutive 1's in payload
- Solution:
  - Sender: In payload add a 0 after 5 consecutive 1's
  - Receiver: Remove bit following 5 consecutive 1's



One link layer protocol: HDLC

- HDLC = High-level Data Link Control

Flag = 01111110

16 or 32 bits CRC

Go-back-N or Selective-repeat ARQ

Summary: Data Link Layer (1)

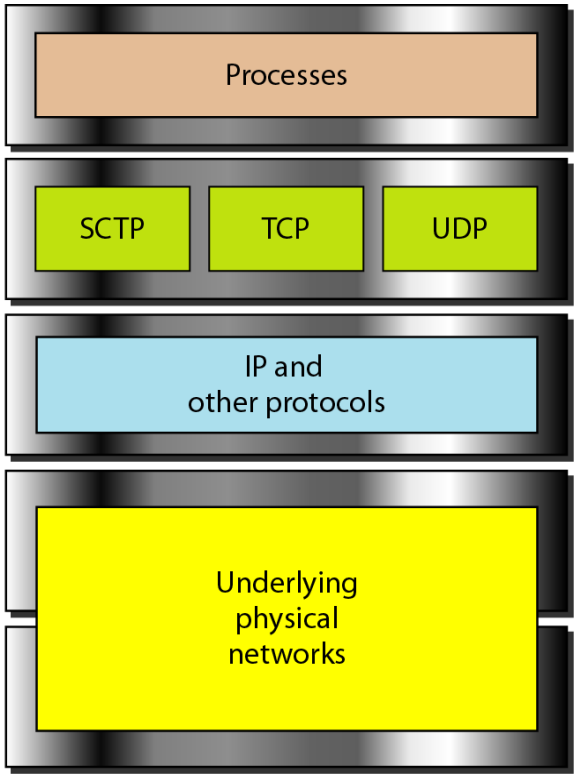
Logical Link Control Sublayer

- Frames
- Error control
  - Detection and correction
- Flow control
  - Stop and wait, go back N, selective repeat
- High-level data link control protocol

Next week: Data Link Layer (2)

Medium Access Control Sublayer

- Access methods
  [S11.1-2]
- Ethernet
  [S12.1-2]
- Wireless local area networks
  [S13.1-3]



**[Kihl & Andersson: 5.1-6]*