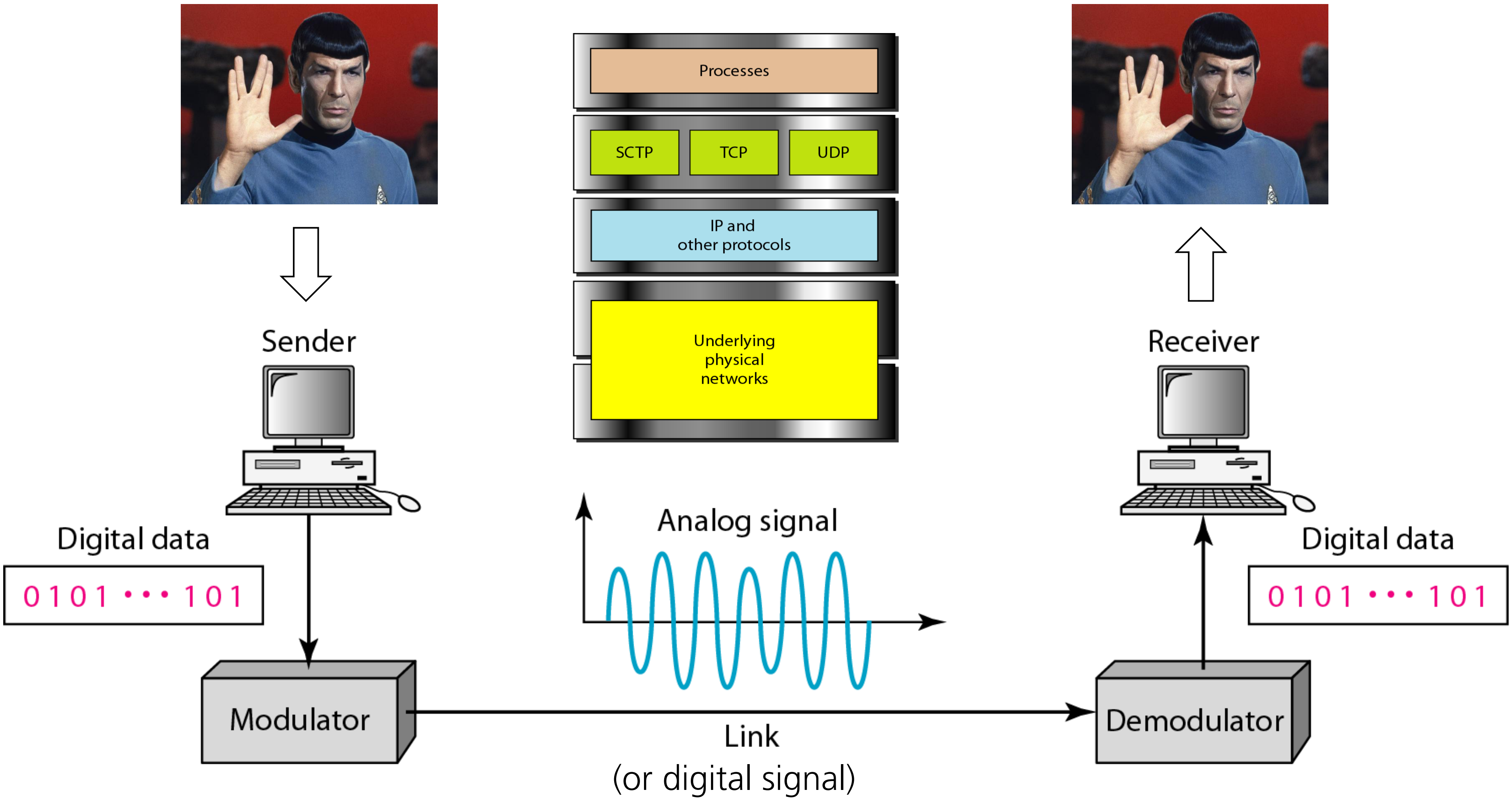
EITF25 Internet--Techniques and Applications

Stefan Höst

L4 Data link (part 1)

LUND UNIVERSITY

Previously on EITF25



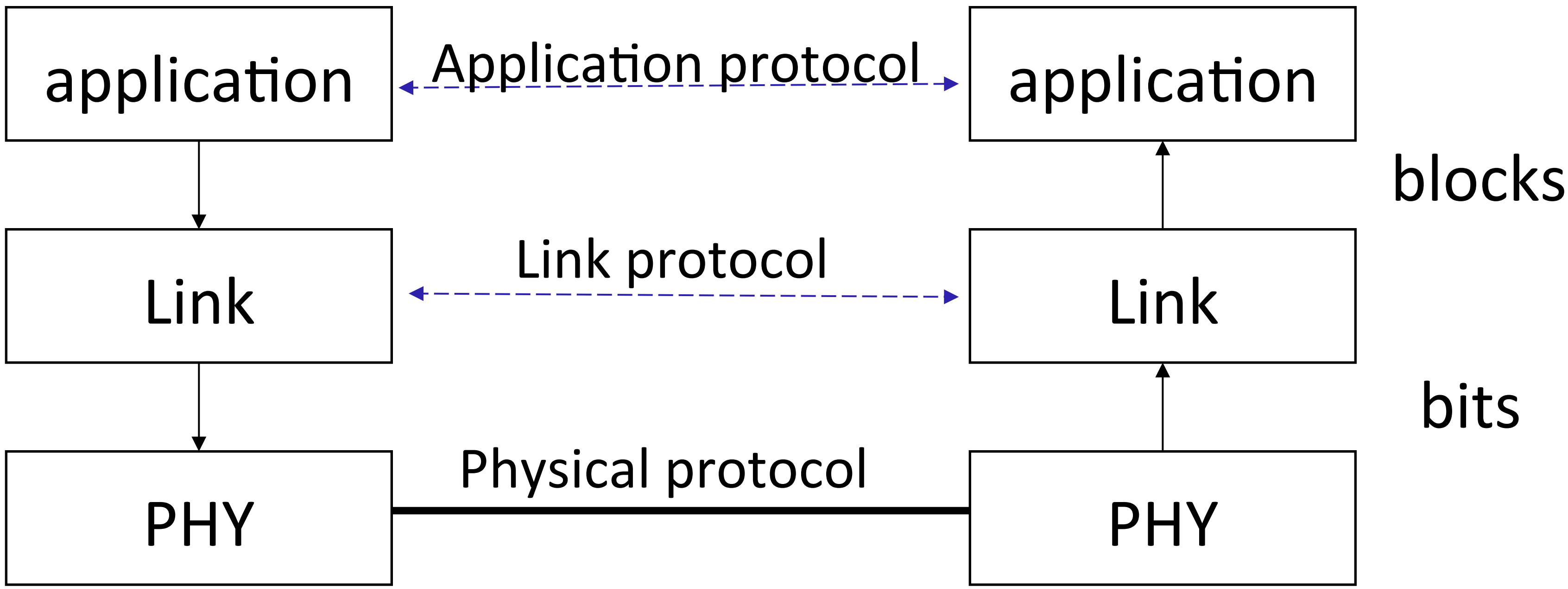
Data Link Layer

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

Link layer protocols

Link Layer Protocols

The sender and receiver uses a link layer protocol that provides error control for data that is sent on a physical link.



Framing

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

Bit stuffing

Flag = 01111110

Avoid having the flag pattern (01111110) in the data:

Transmitter

- After five consecutive 1s insert a 0

Receiver

- After five consecutive 1s delete next bit

01111110	header	101010010101010.........100101010110	trailer	01111110

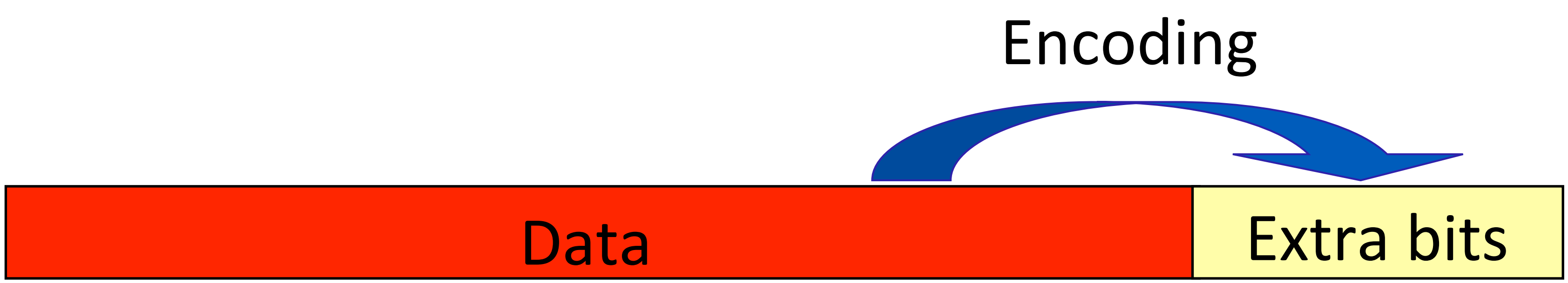
Bit stuffing

Example

- Data = 01001111111101011111001001...
- Data_{BS}= 01001111101110101111100001001...

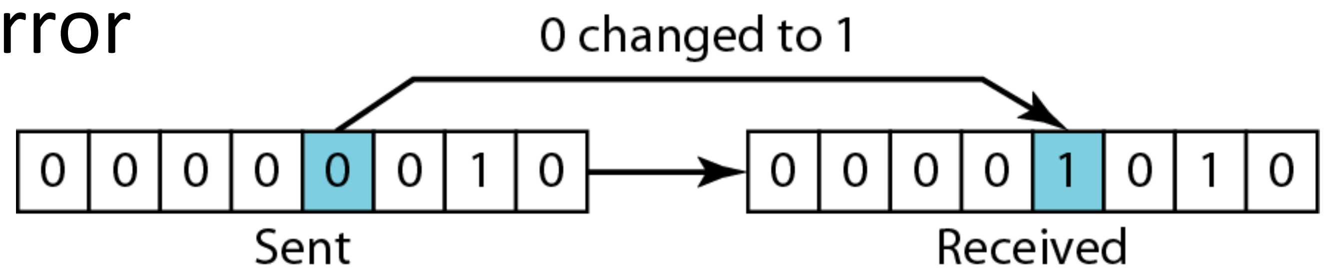
Error control

- Data assumed error-free by higher layers
    - Errors occur at lower layers (physical)
- Extra (redundant) bits added to data
    - Generated by an encoding scheme from data
    - Used to resolve (detect or correct) errors

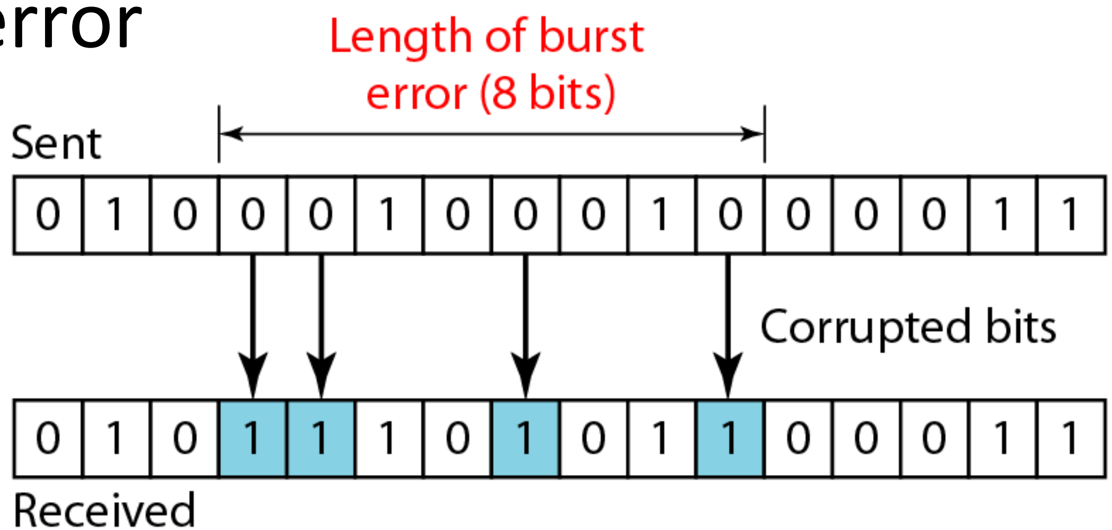


Typical Error Types

- Bit error



- Burst error



Error control

- Error detection: the aim is to detect errors
  - The transmission protocol decides what to do about erroneous packages
- Error correction: the aim is to correct errors
  - Roughly half as many errors can be corrected as can be detected.
- In most communication systems both error detection and error correction occur

Error detection schemes

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

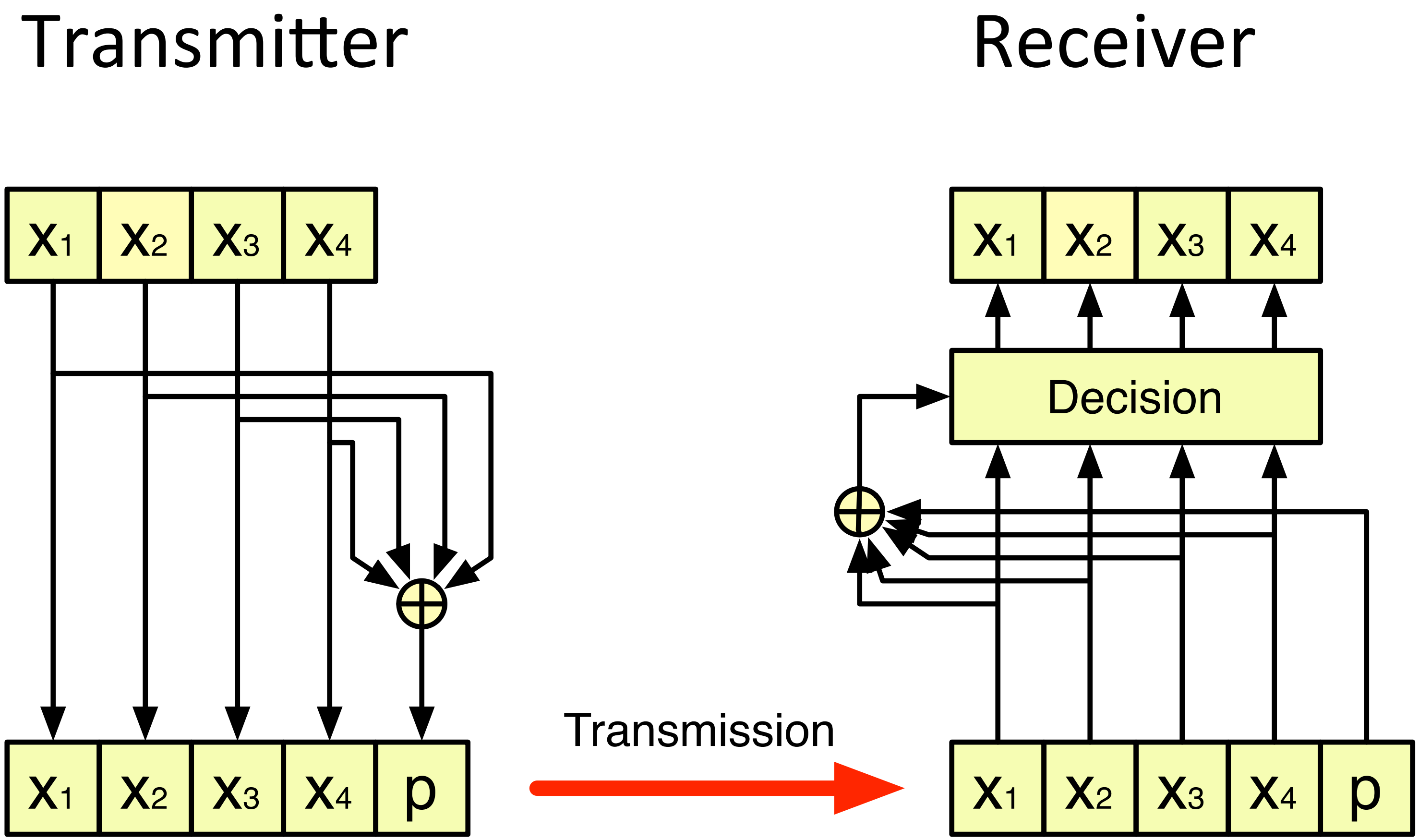
Simple Parity-Check Code

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

dataword				codeword
10011100	+	0	=	100111000

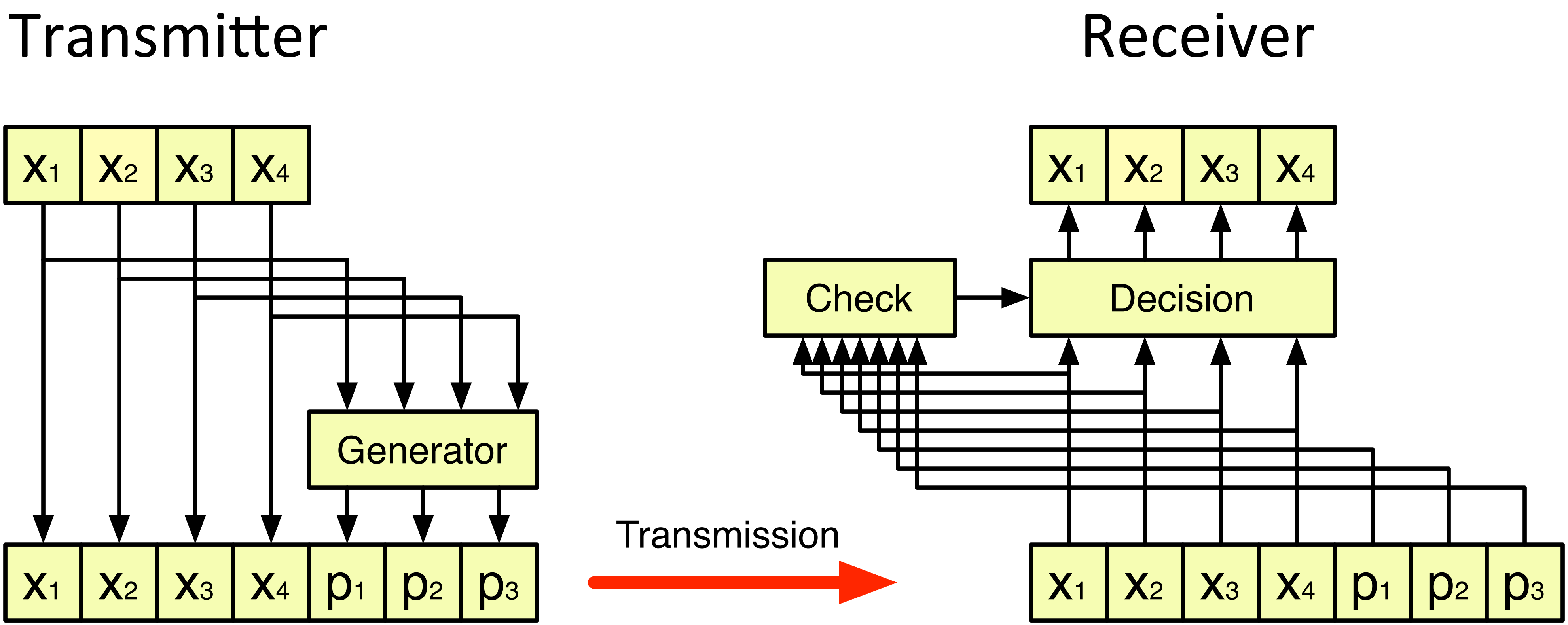
- Can detect an odd number of errors

Simple Parity-Check Code



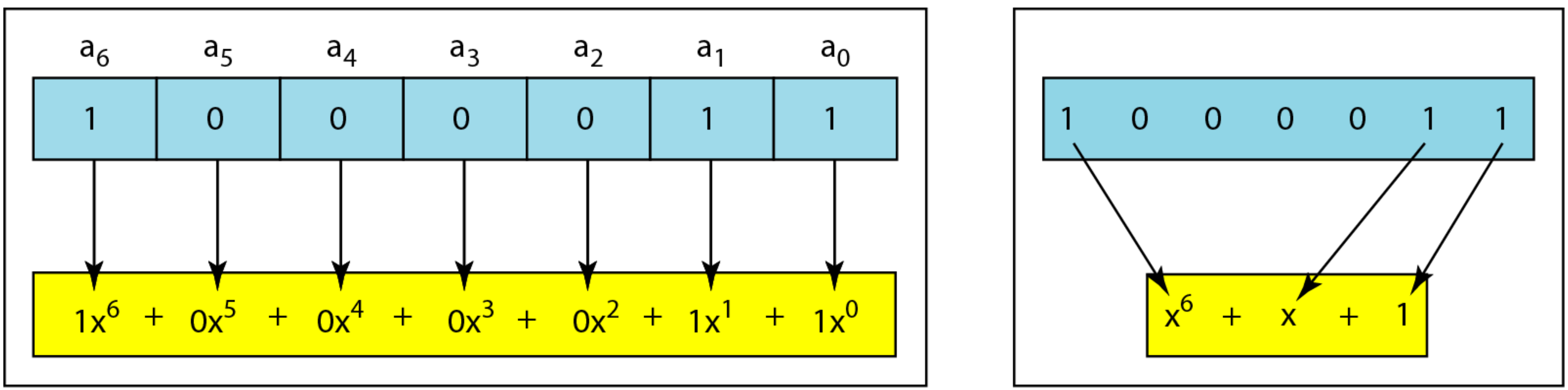
Cyclic Redundancy Check (CRC)

Generalise to more (independent) parity bits



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:

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

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: $d(x) = x^3+1$

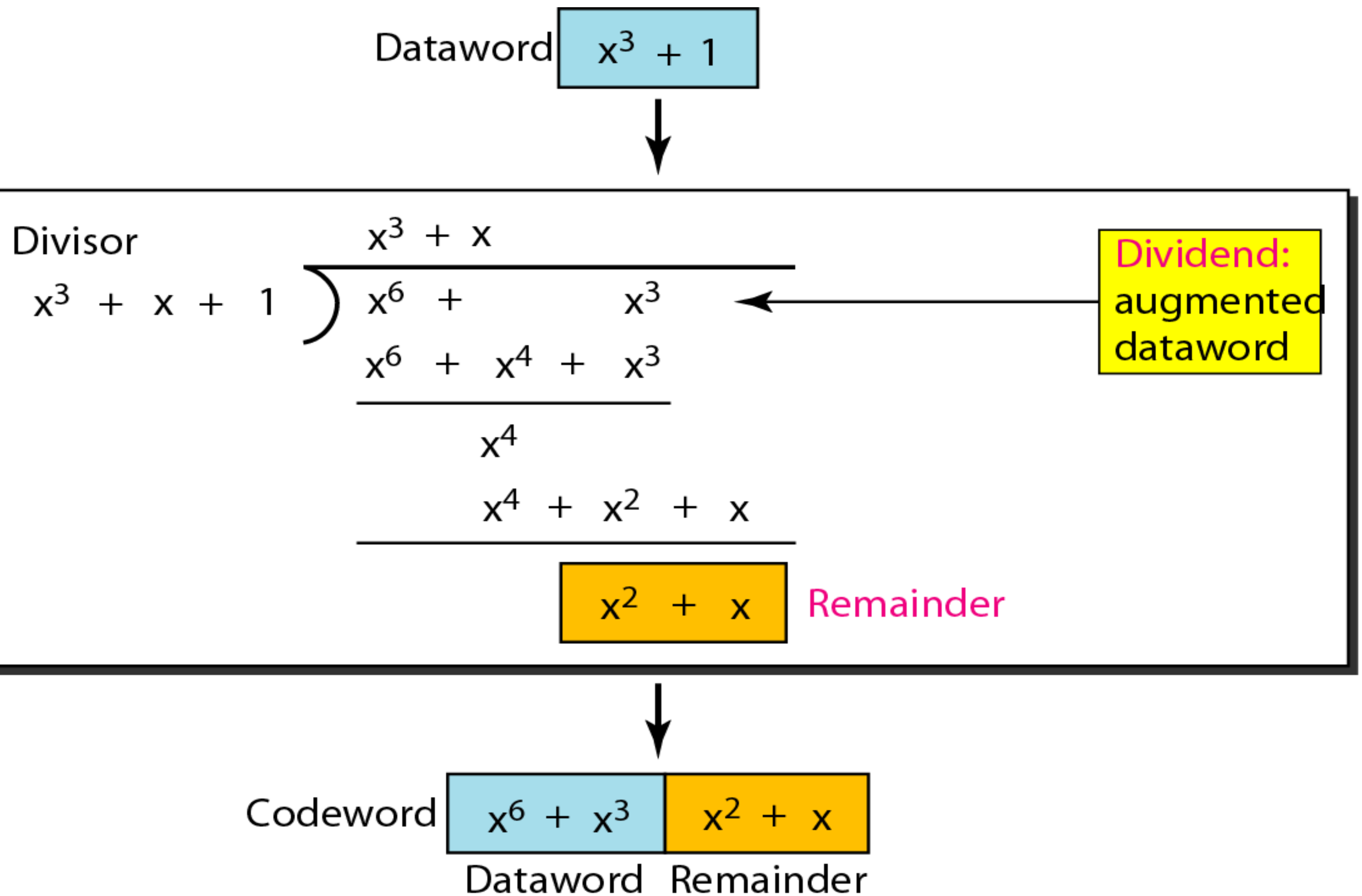
Generator polynomial: $g(x) = x^3+x+1$

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: $r(x) = ?$

Example: CRC derivation



CRC: Some theory

- The CRC polynomial is the reminder of the division

$$\frac{d(x)x^{n-k}}{g(x)}$$

- Thus $d(x)x^{n-k} = g(x)z(x) + r(x)$
  or, equivalently $c(x) = d(x)x^{n-k} + r(x) = g(x)z(x)$

> A polynomial *c(x)* with *deg<n* is a codeword if and only if *g(x)* divides *c(x).*

Error detection capabilities

- Single errors: $e(x)=x^i$ is not divisable by $g(x)$
- Double errors: $e(x)=x^j+x^i=x^i(x^{j-i}+1)$
  - Use primitive polynomial $p(x)$ with $deg=L$. Then if $n-1<2^L-1$ it is not divisable and all double errors will be detected
- If $x+1 \mid g(x)$ all odd error patterns will be detected
- In practice, set $g(x)=(x+1)\cdot p(x)$

Some standard CRC polynomials

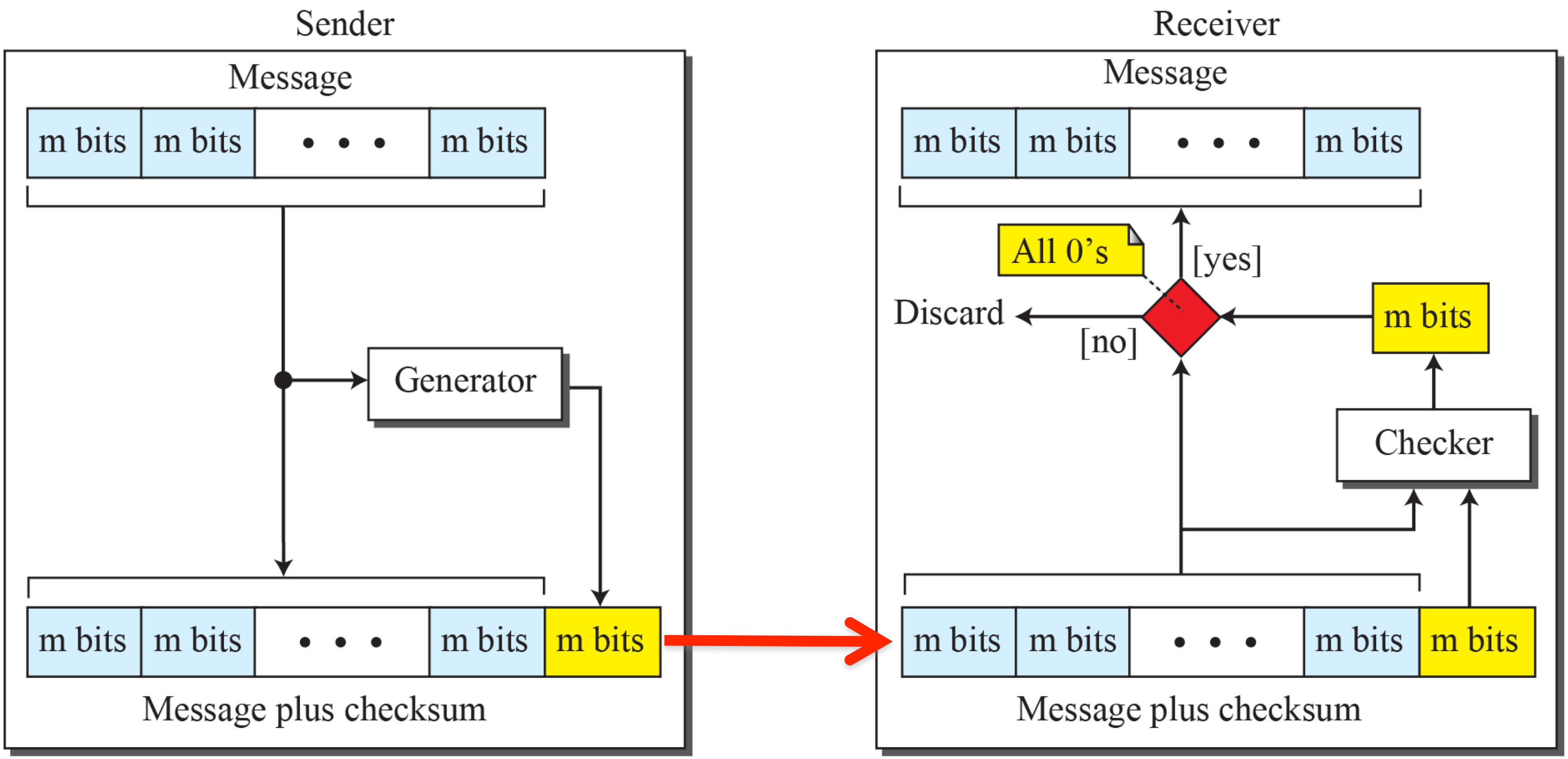
Name	*Polynomial*	*Used in*
CRC-8	$x^8 + x^2 + x + 1$ **100000111**	ATM header
CRC-10	$x^{10} + x^9 + x^5 + x^4 + x^2 + 1$ **11000110101**	ATM AAL
CRC-16	$x^{16} + x^{12} + x^5 + 1$ **10001000000100001**	HDLC
CRC-32	$x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} + x^{10} + x^8 + x^7 + x^5 + x^4 + x^2 + x + 1$ **100000100110000010001110110110111**	LANs

Checksum

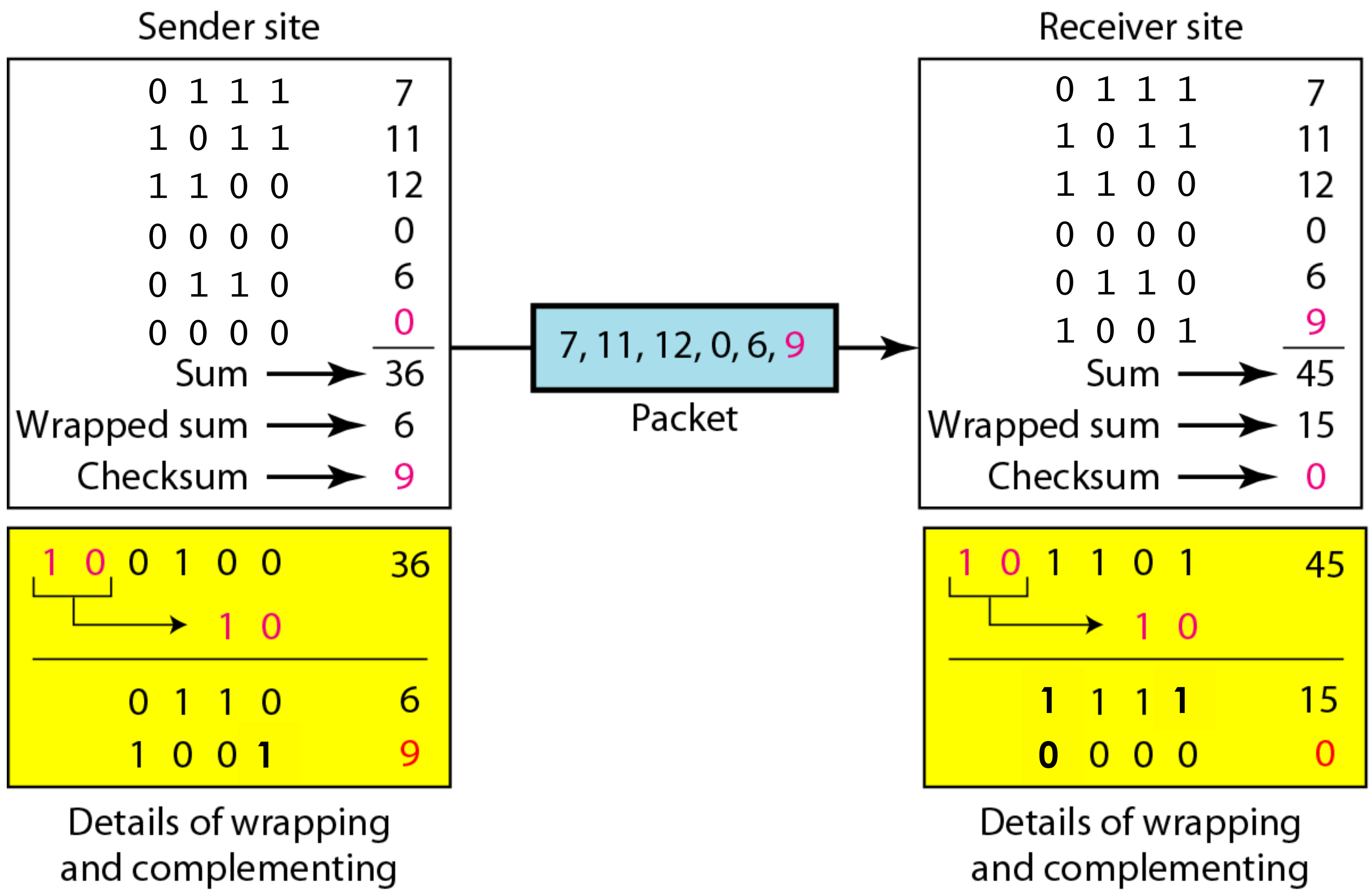
- The checksum is used in the Internet by several protocols although not at the data link layer.

- 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



Forward Error Correction (FEC)

Two simple examples

- Repetition code
- Concatenated parity check

Repetition code

Transmitter

- For each bit, transmit three copies

Receiver

- Decode acording to majority decision
- If one error uccured this will be corrected

Repetition code, Example

Transmitter

- $d = 0 \Rightarrow c = 000$

Channel

- One error: $e = 010 \Rightarrow y = 010$

Receiver

- $y = 010 \Rightarrow \hat{c} = 000 \Rightarrow \hat{d} = 0$

Vertical and horisontal parity

Encoding

- Let *d* be a binary matrix.
- Add parity bits for each row and column

Decoding

- If one error, it can be found from the parity bits
- Two or three errors can be detected (but not always corrected)

V+H parity, Example

Encoding

$$d = \begin{array}{cccc} 1 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 \end{array} \Rightarrow c = \begin{array}{cccc|c} 1 & 1 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 \\ \hline 1 & 1 & 1 & 1 & 0 \end{array}$$

Channel

One error

$$y = \begin{array}{cccc|c} 1 & 1 & 0 & 1 & 1 \\ 0 & 0 & \color{red}{\mathbf{0}} & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 \\ \hline 1 & 1 & 1 & 1 & 0 \end{array}$$

V+H parity, Example

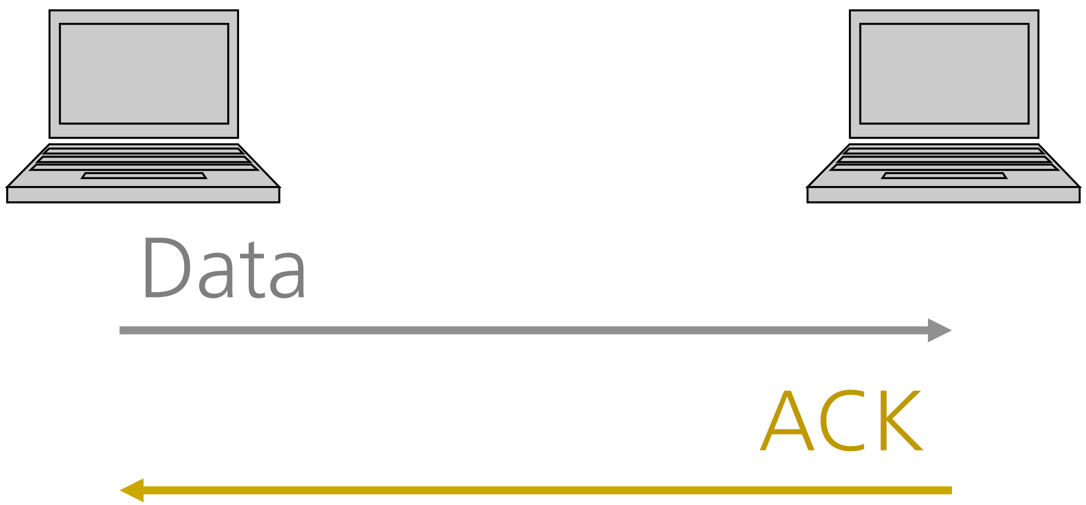
Decoding

Two parity bits wrong. Points at one position.

$$y = \begin{array}{cccc|c} 1 & 1 & 0 & 1 & 1 \\ 0 & 0 & \color{red}{\mathbf{0}} & 1 & \color{blue}{\mathbf{0}} \\ 0 & 0 & 0 & 1 & 1 \\ \hline 1 & 1 & \color{blue}{\mathbf{1}} & 1 & 0 \end{array} \Rightarrow \hat{c} = \begin{array}{cccc|c} 1 & 1 & 0 & 1 & 1 \\ 0 & 0 & \color{red}{\mathbf{1}} & 1 & \color{blue}{\mathbf{0}} \\ 0 & 0 & 0 & 1 & 1 \\ \hline 1 & 1 & \color{blue}{\mathbf{1}} & 1 & 0 \end{array} \Rightarrow \hat{d} = \begin{array}{cccc} 1 & 1 & 0 & 1 \\ 0 & 0 & \color{red}{\mathbf{1}} & 1 \\ 0 & 0 & 0 & 1 \end{array}$$

Error and flow control

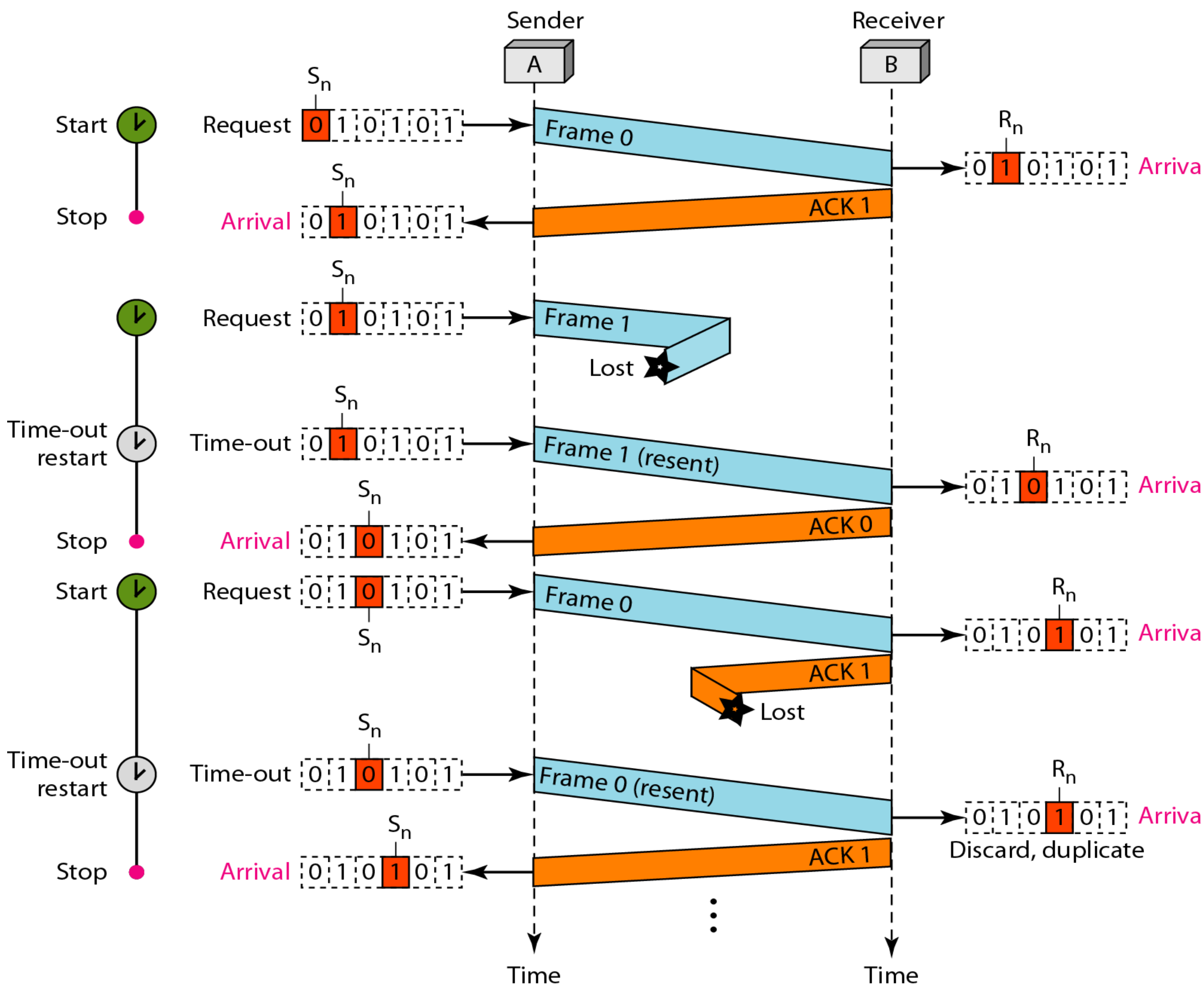
The basic principle in error and flow control is that the receiver **acknowledges** all correctly received packets.



Stop-and-wait ARQ

- The receiver sends an ACK for every data packet that is correctly received.
- The sender transmits the next packet when it has received an ACK for the previous one.
- The sender uses a time-out for each packet. If the time-out expires (i.e. no ACK has arrived), the packet is retransmitted.
- Packets are identified with a sequence number, alternating between 0 and 1. ACK is labeled with the next packet

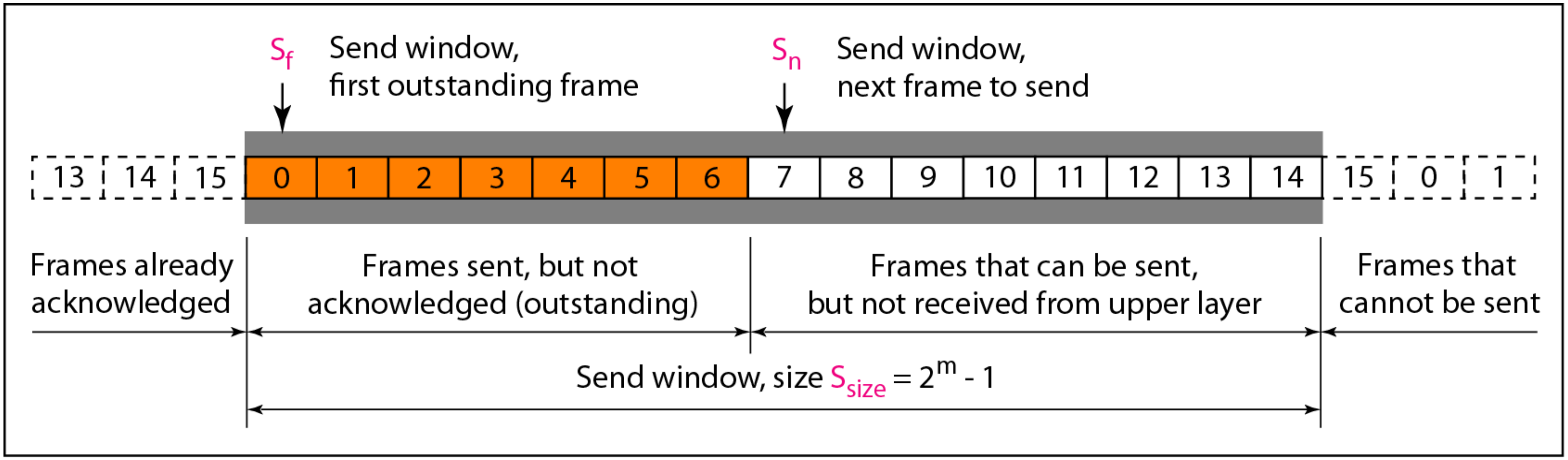
Stop-and-wait ARQ flow diagram



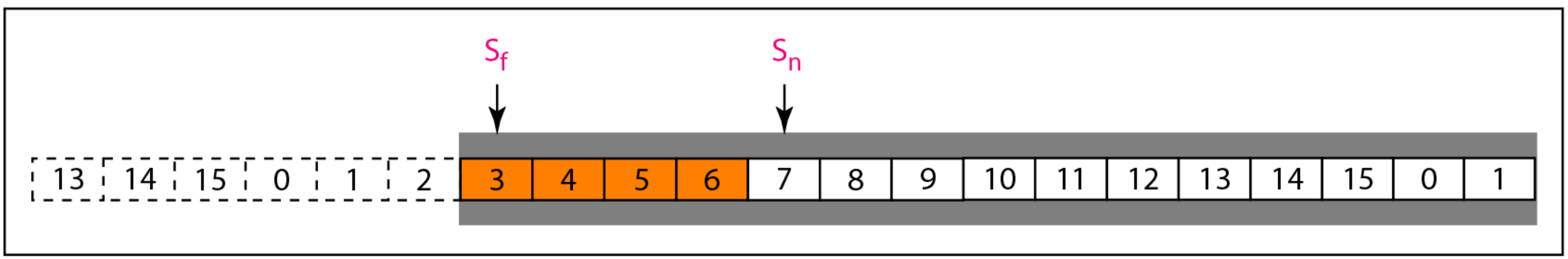
Stop-and-wait ARQ inefficiency

- Too much waiting
- Solution
  - Keep the pipe full
  - But not too full
- Sliding window
  - Size matters
  - Window size $< 2^m$

Sliding window at sender

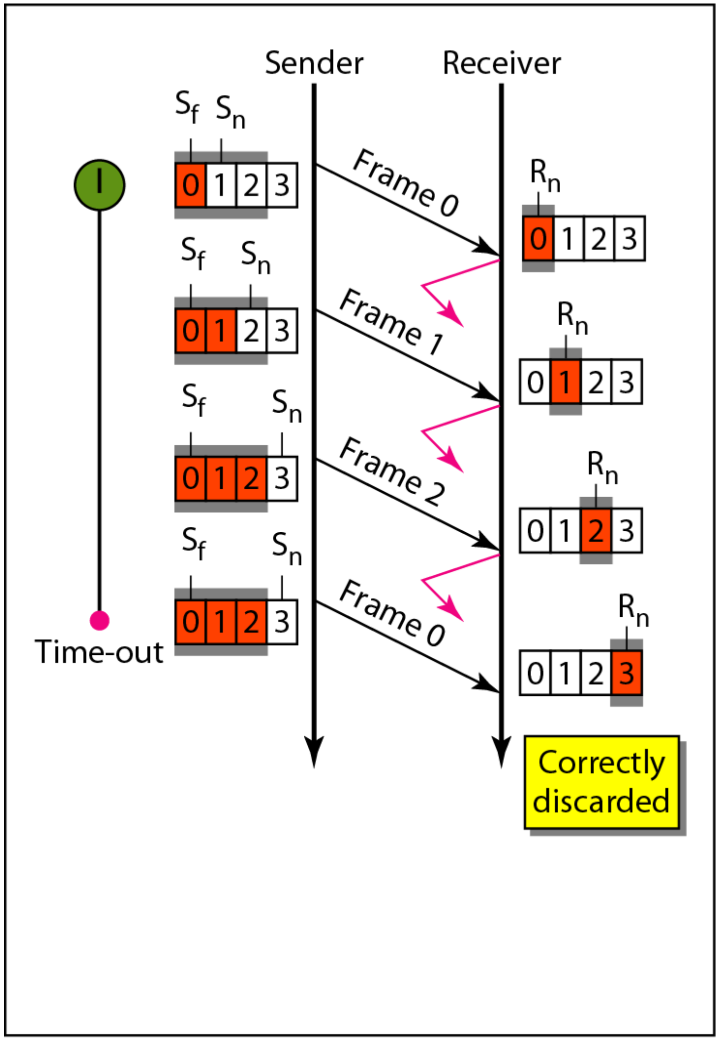


a. Send window before sliding

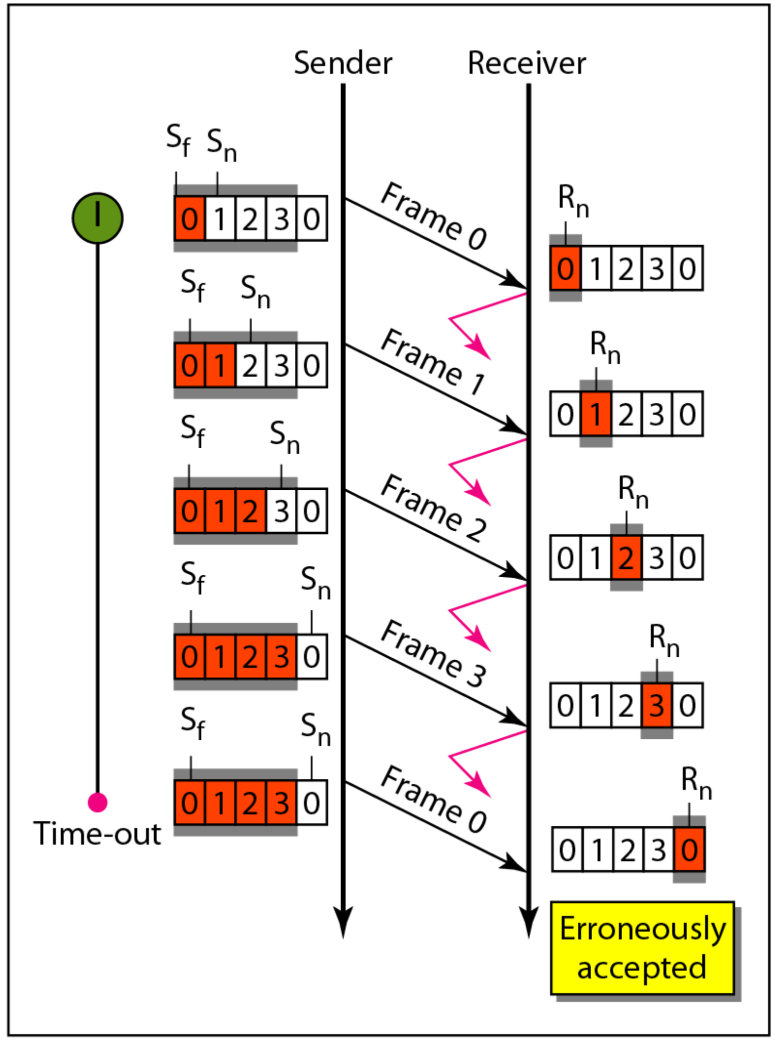


b. Send window after sliding

Importance of window size

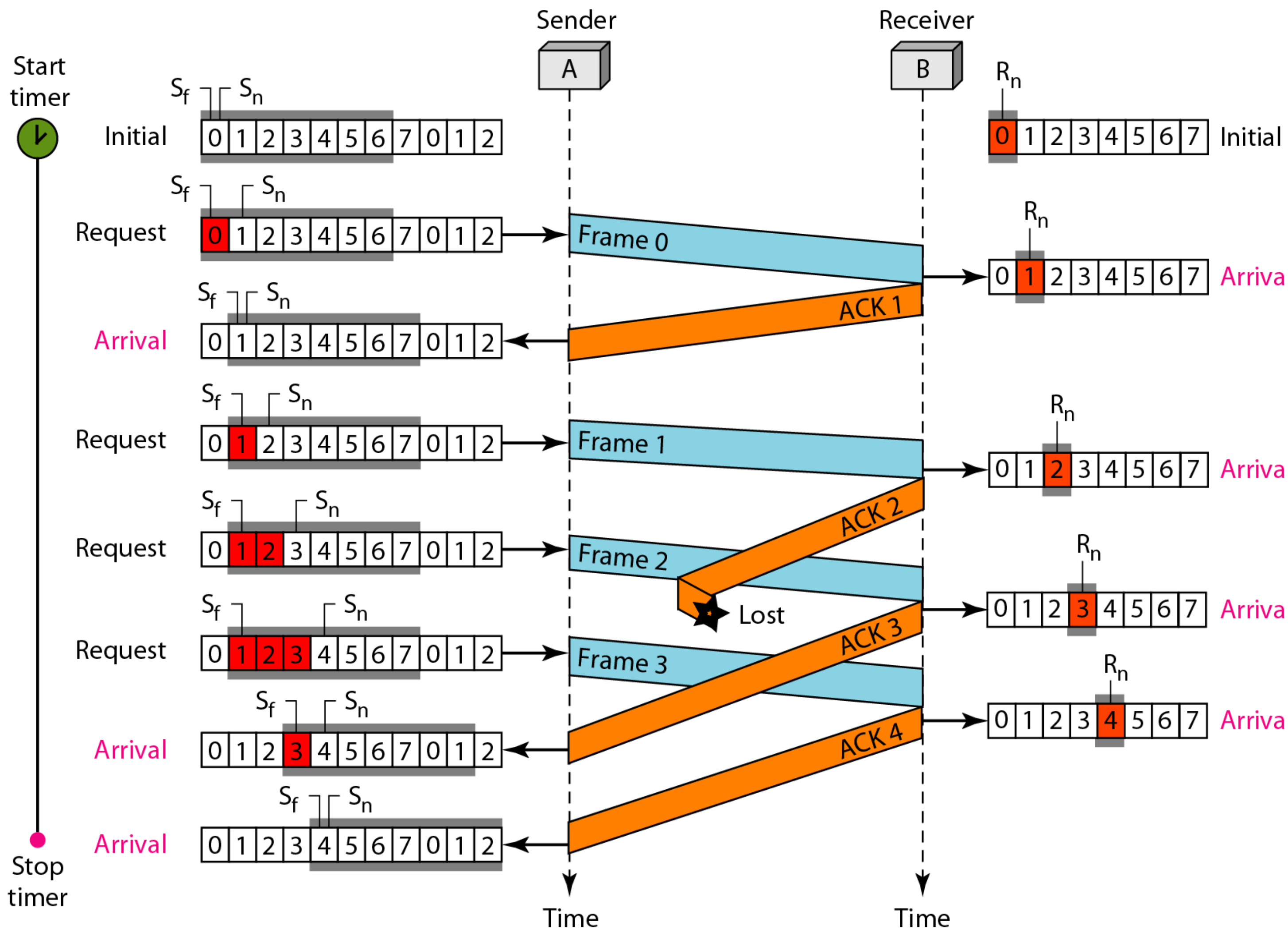


a. Window size < 2^m

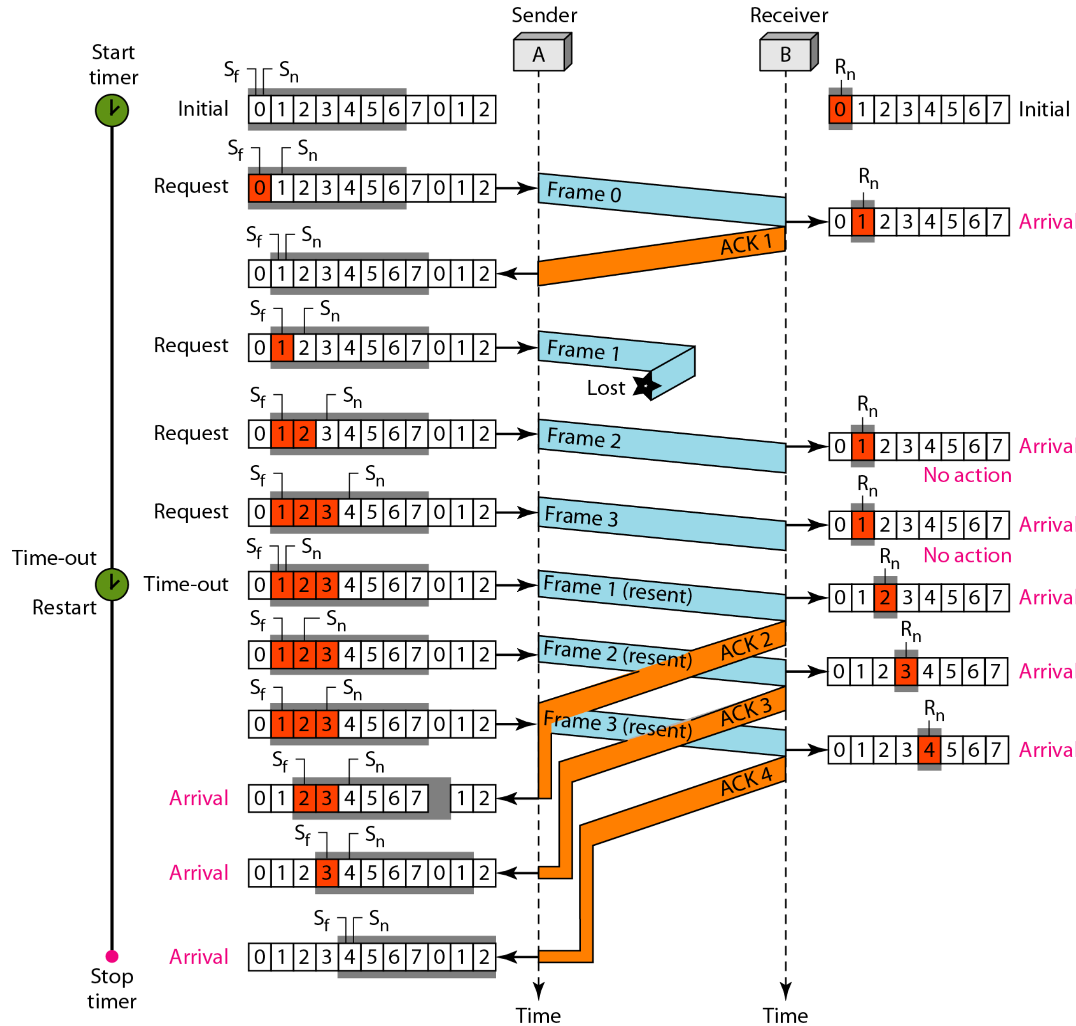


b. Window size = 2^m

Go-back-N, example (lost ACK)



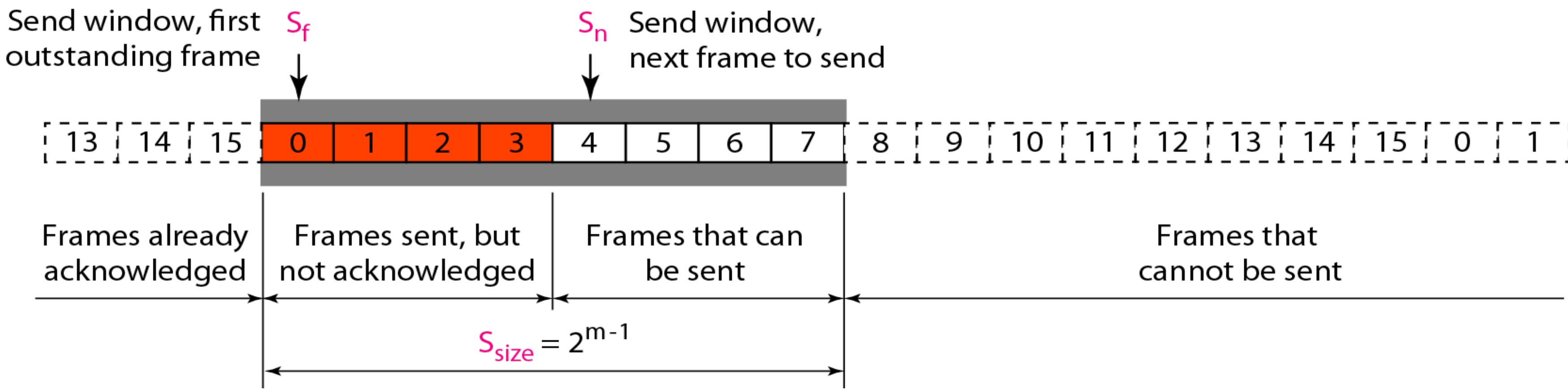
Go-back-N, example (lost data)

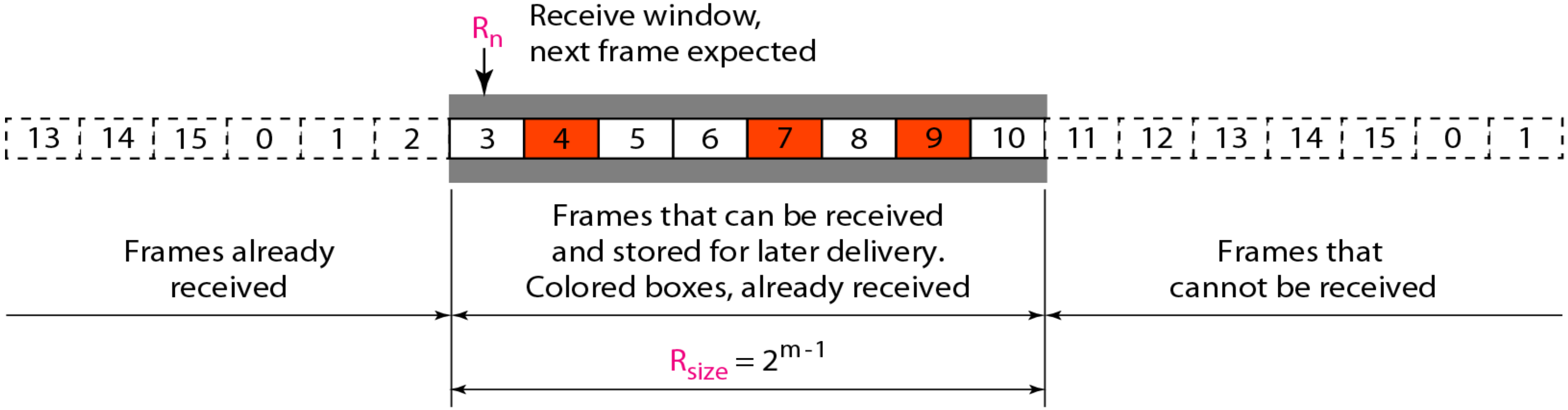


Selective repeat ARQ

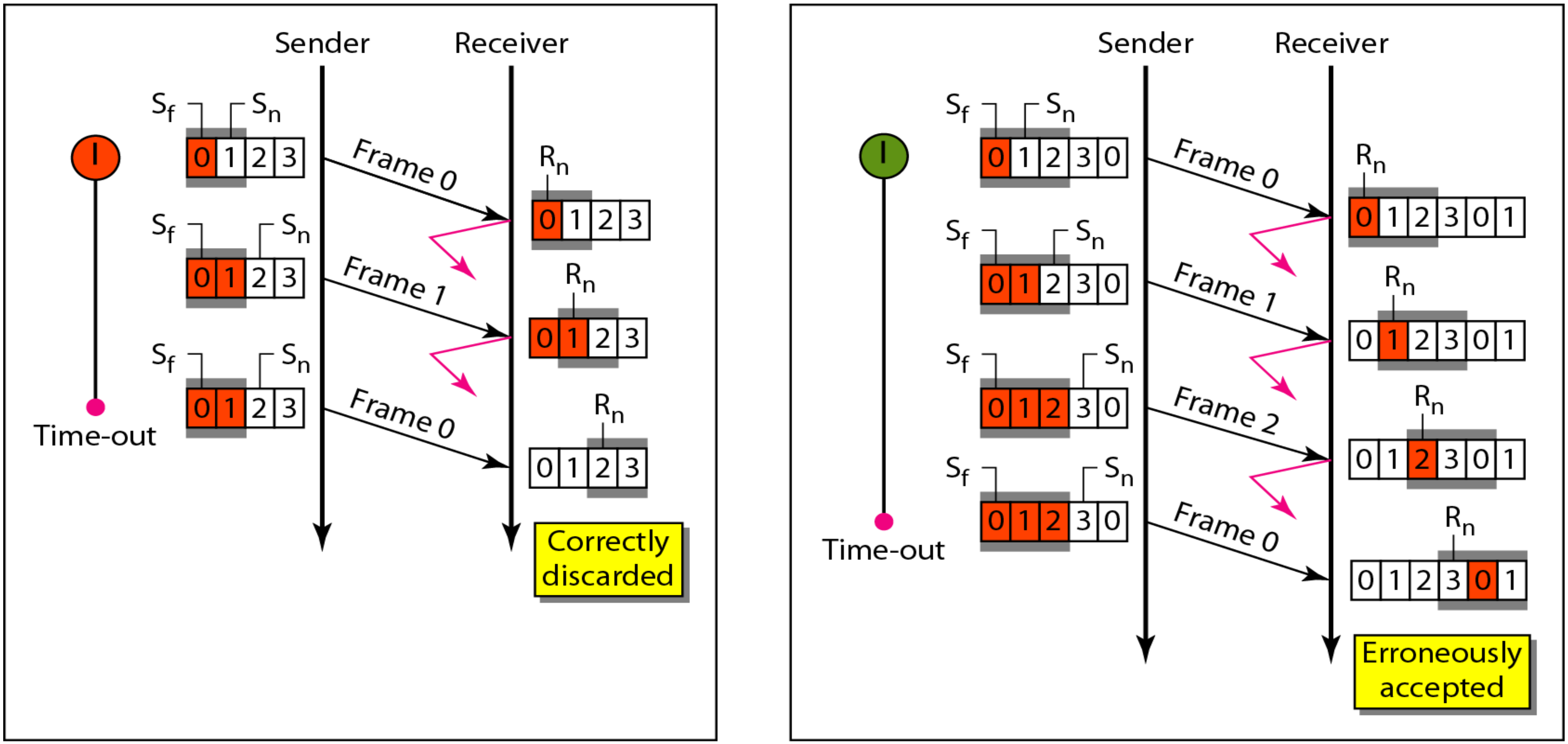
- Why?
  - Too many retransmissions
- What if?
  - Just send lost frames
- Higher efficiency
  - Higher receiver complexity

Windows again





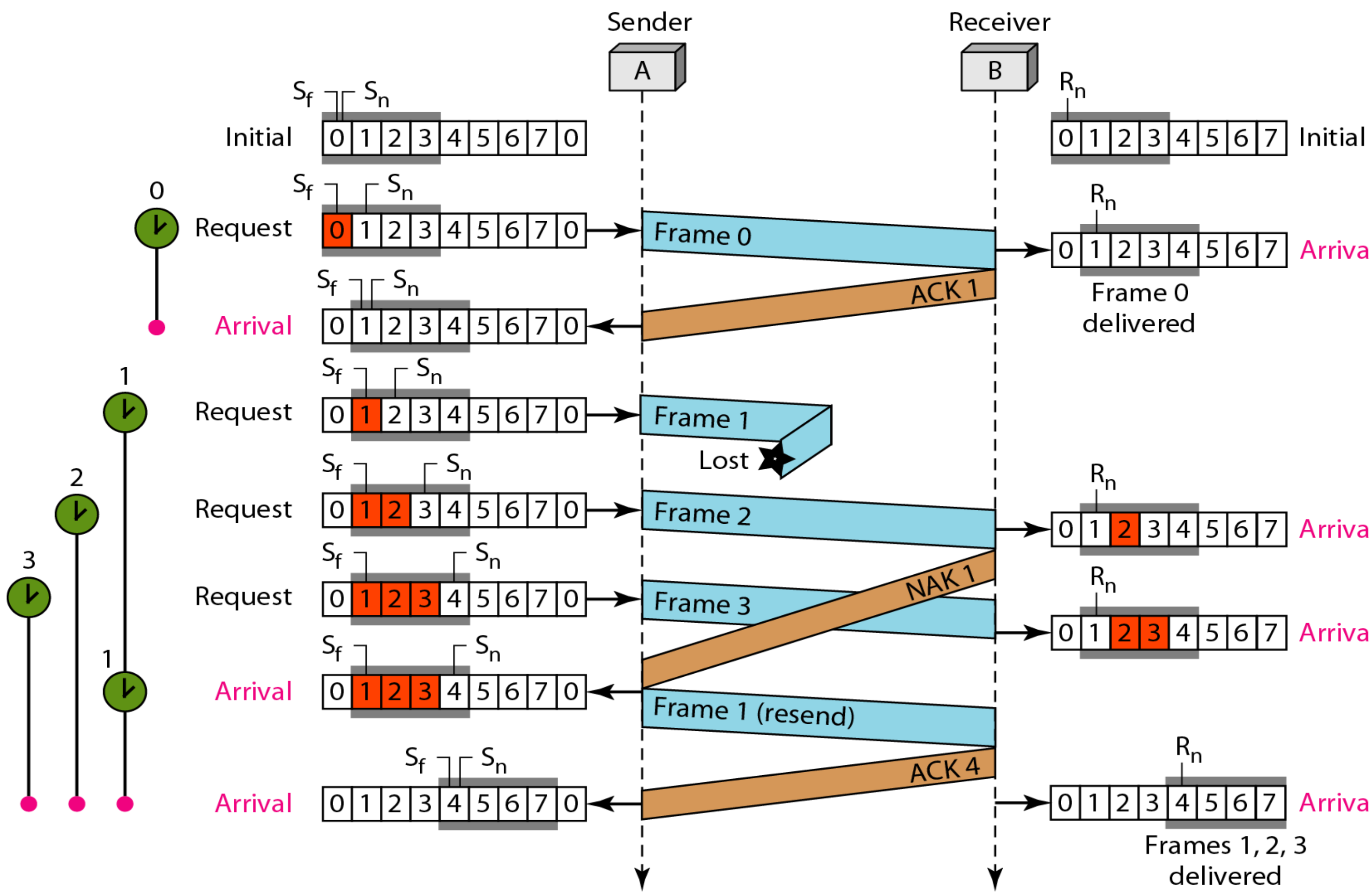
Selective repeat ARQ window size



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

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

Selective repeat ARQ flow diagram



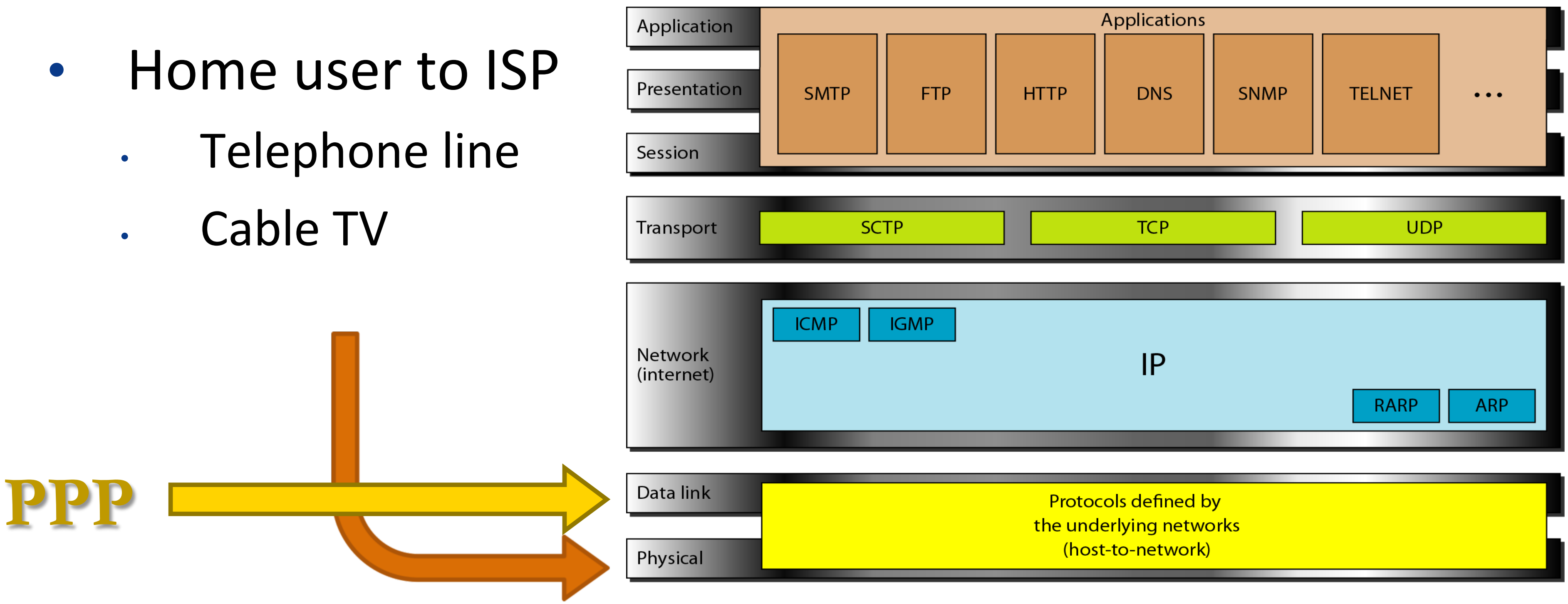
Piggyback

- Often the traffic goes both ways.
  - Use the transmitted packages to send ACK
  - Let S_n and R_n be transmitted and received sequense numbers. Use frame as below.

Flag	S_n	R_n	Data	CRC	Flag

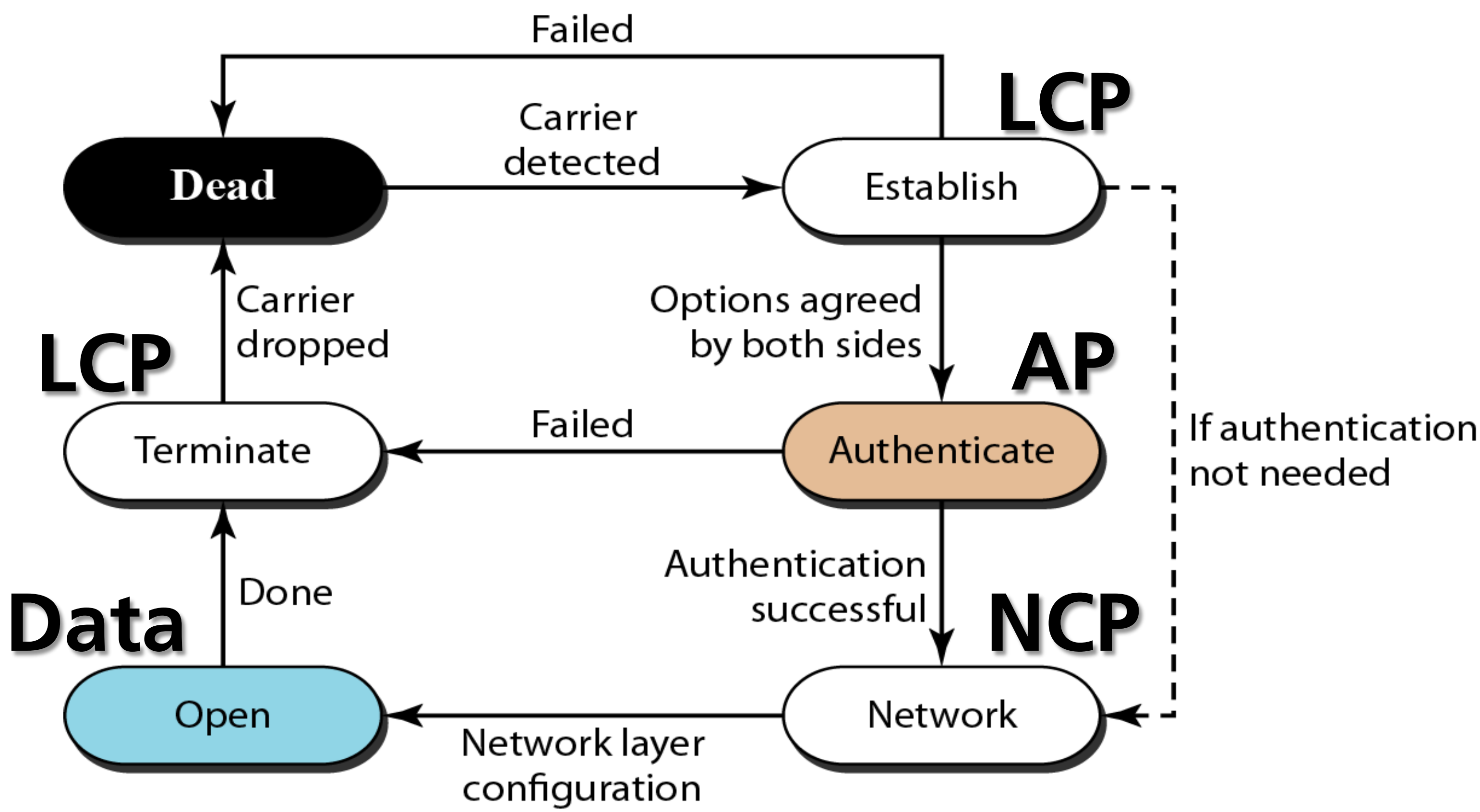
Point-to-point protocol (PPP)

- Direct connection between two nodes
  - Internet access
  - Home user to ISP
    - Telephone line
    - Cable TV



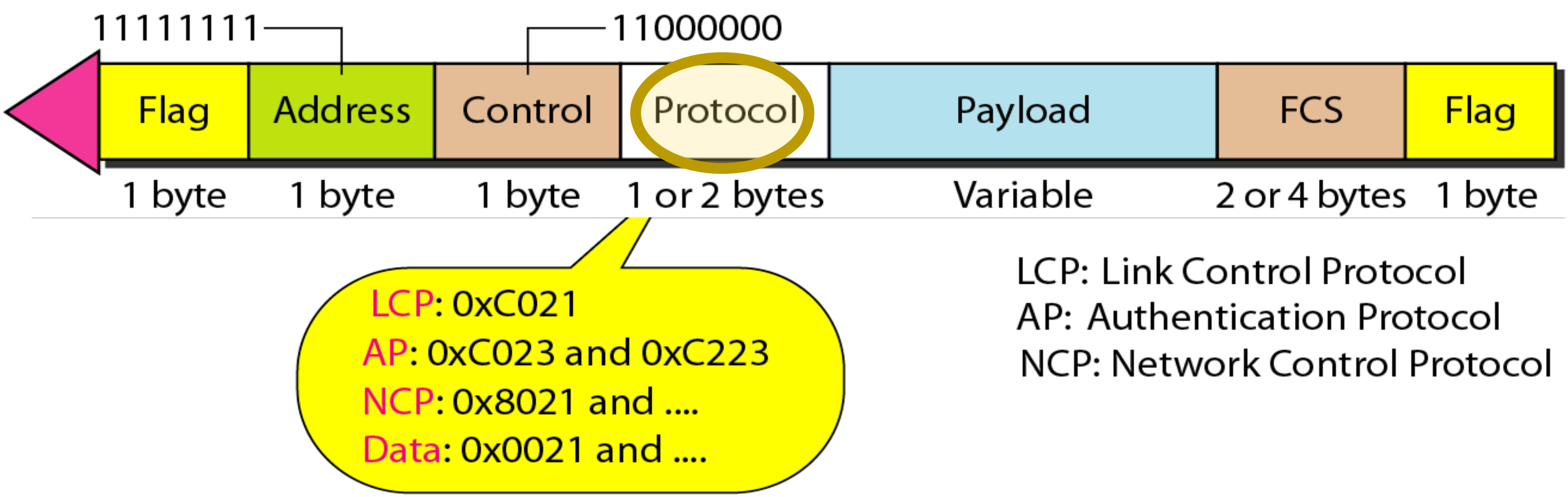
State transitions in PPP

- We need more protocols



PPP frame format

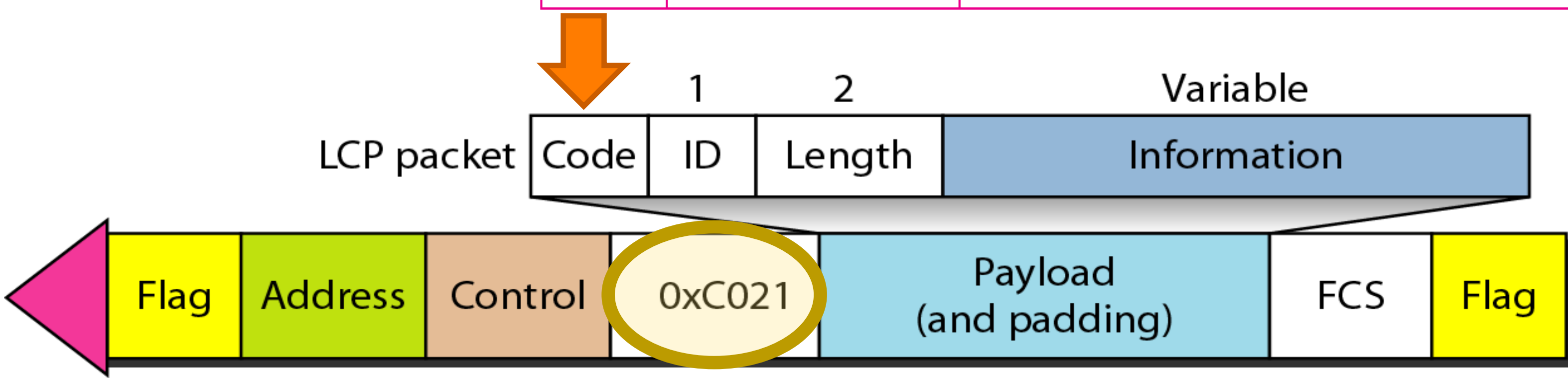
- Support for several (sub)protocols
- Address & control not used
- Maximum payload 1500 bytes



Link control protocol (LCP)

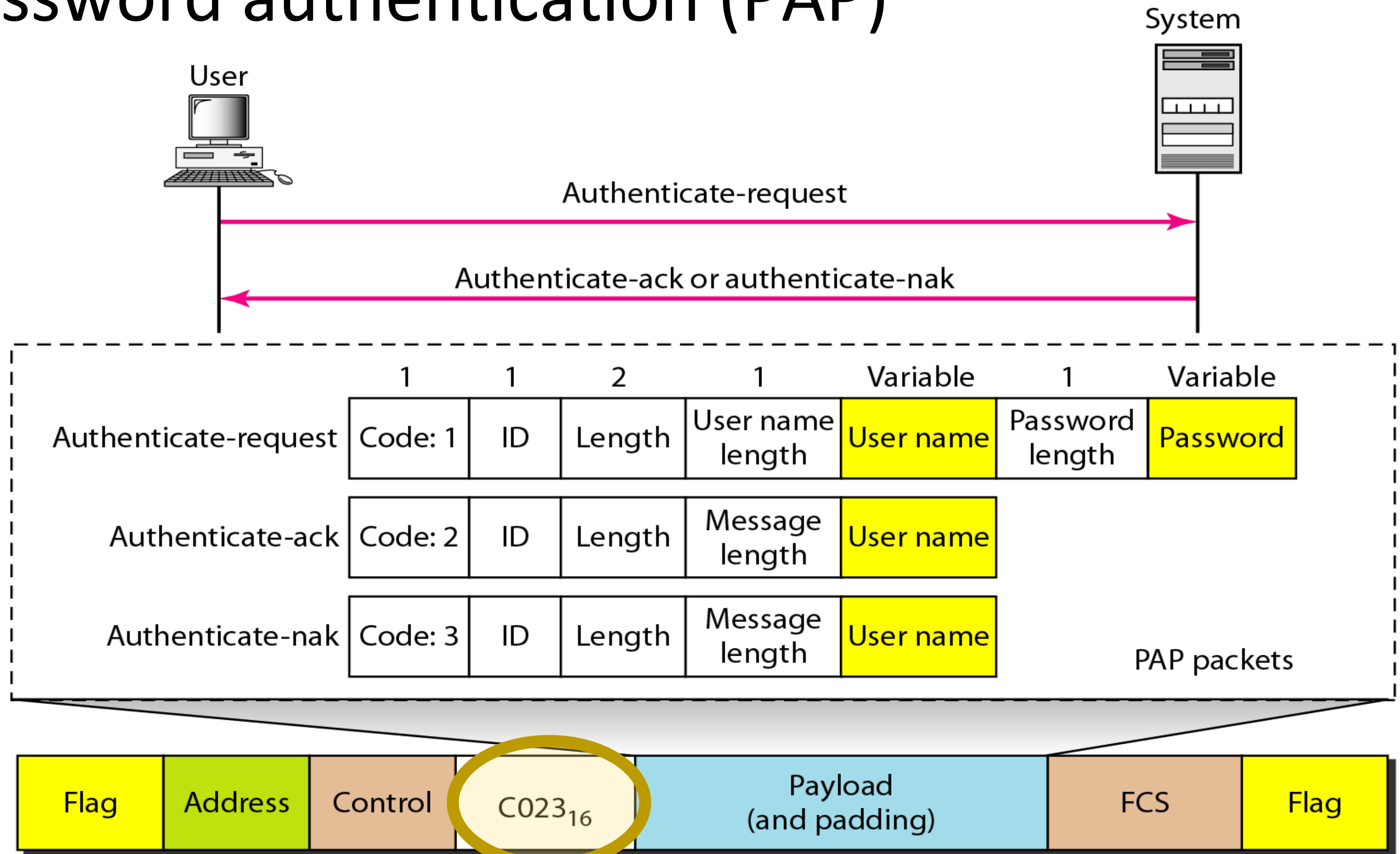
- Establish
- Configure
- Terminate

Code	*Packet Type*	*Description*
0x01	Configure-request	Contains the list of proposed options and their values
0x02	Configure-ack	Accepts all options proposed
0x03	Configure-nak	Announces that some options are not acceptable
0x04	Configure-reject	Announces that some options are not recognized
0x05	Terminate-request	Request to shut down the line
0x06	Terminate-ack	Accept the shutdown request
0x07	Code-reject	Announces an unknown code
0x08	Protocol-reject	Announces an unknown protocol
0x09	Echo-request	A type of hello message to check if the other end is alive
0x0A	Echo-reply	The response to the echo-request message
0x0B	Discard-request	A request to discard the packet



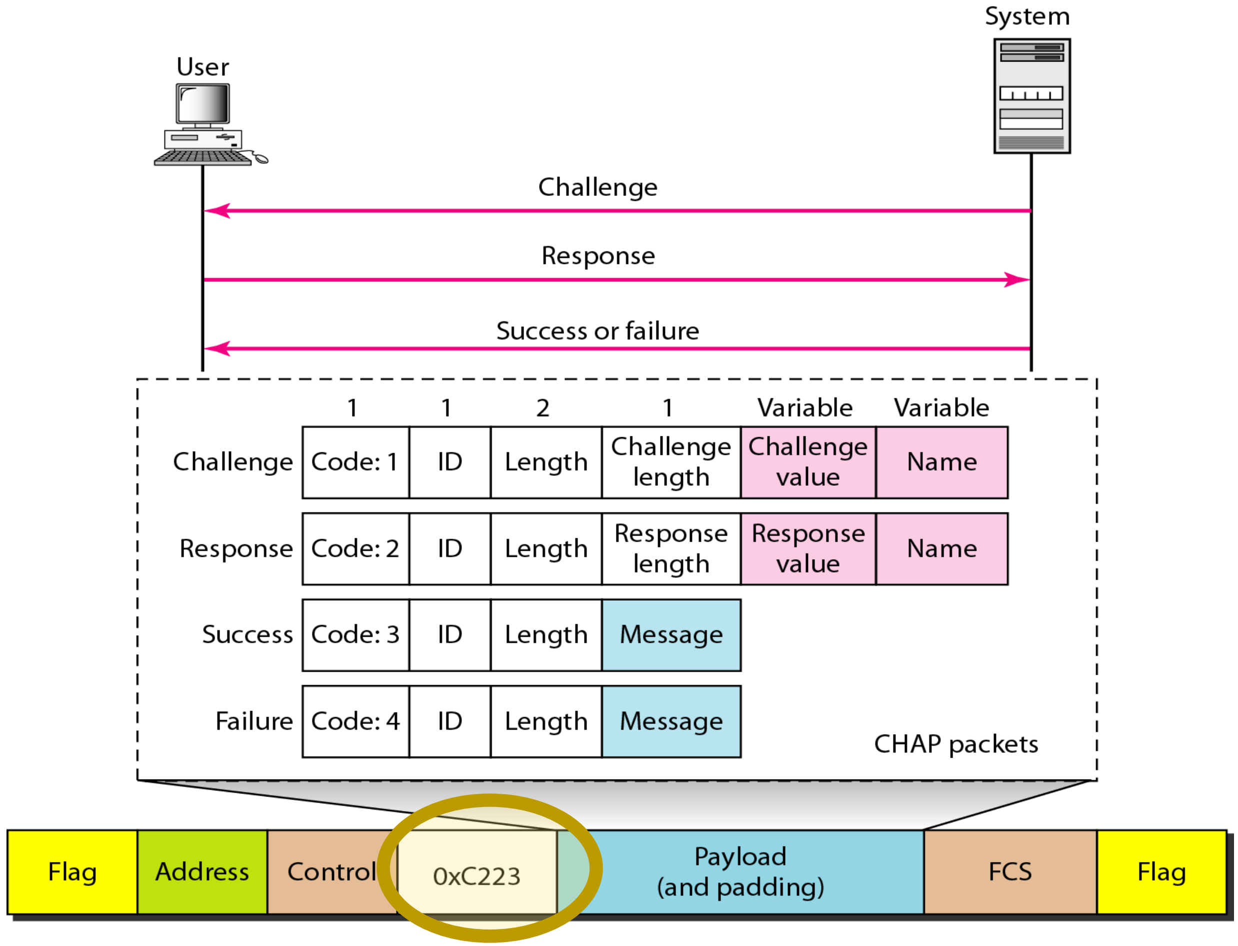
Authentication protocols (AP)

- Password authentication (PAP)



Authentication protocols (AP)

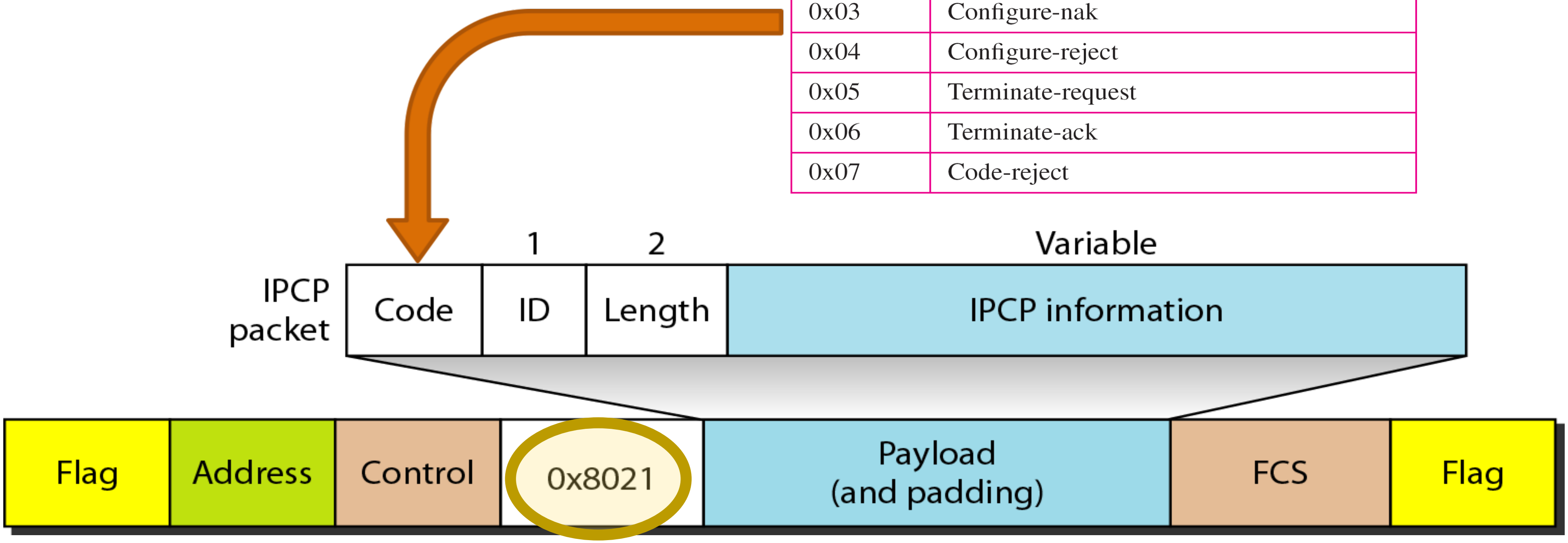
- Challenge handshake authentication (CHAP)



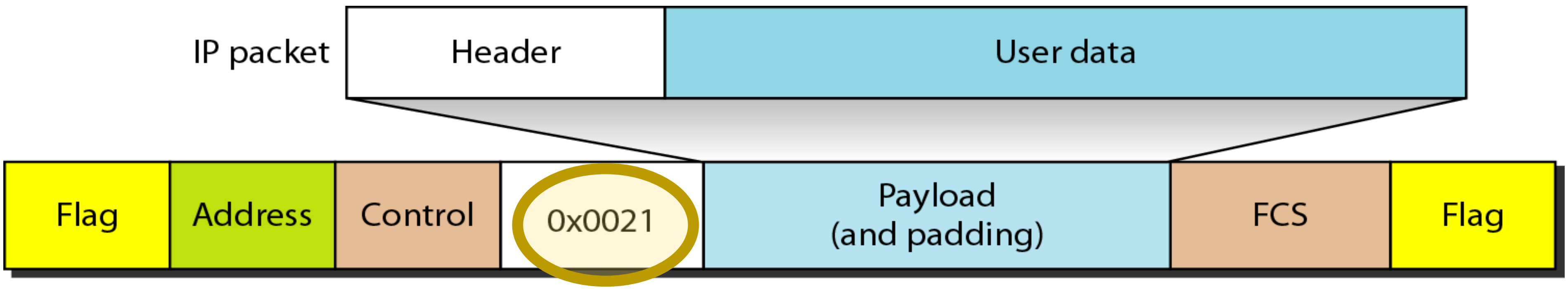
Network control protocols (NCP)

- Preparations for the network layer
  - IPCP for Internet

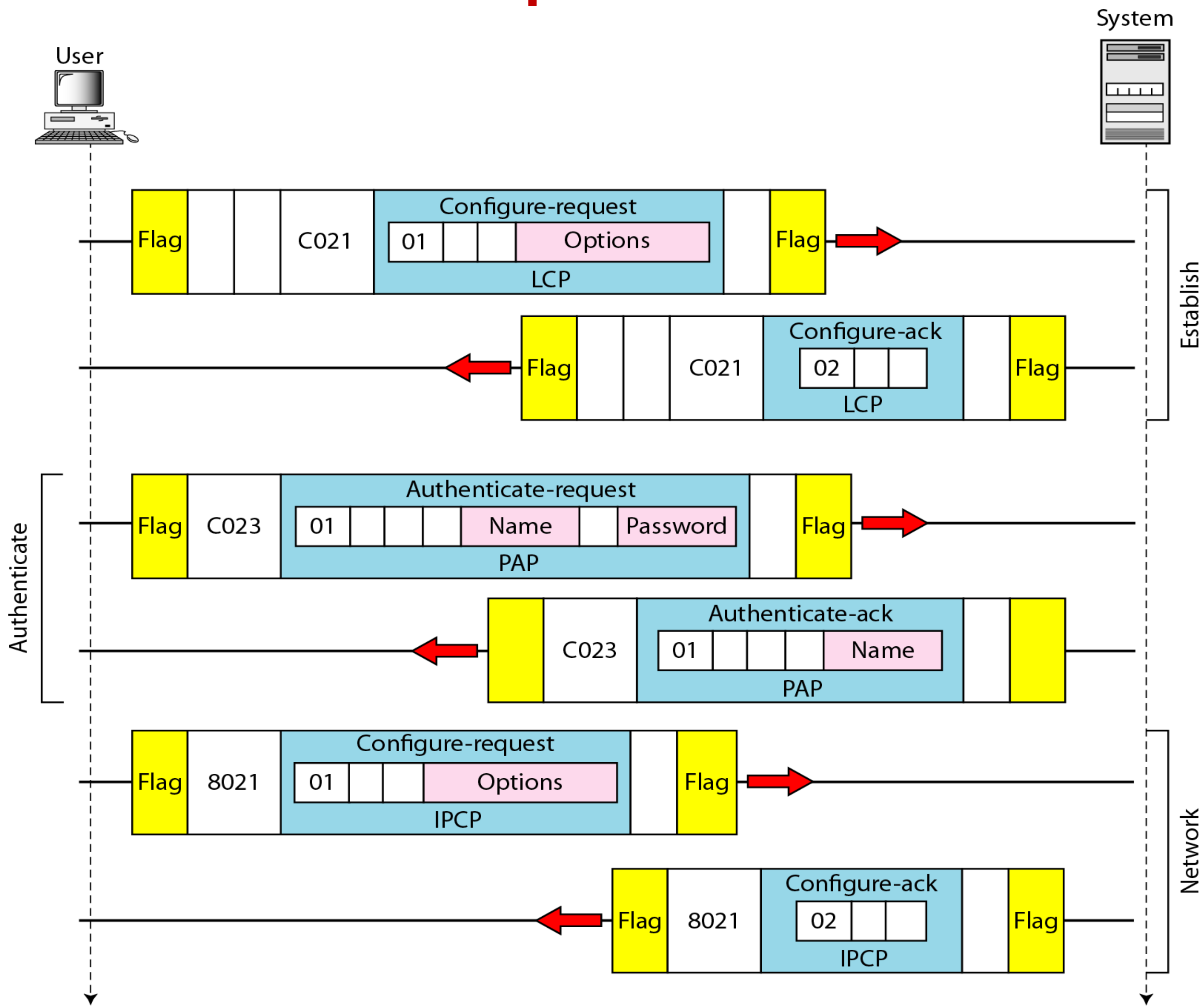
Code	*IPCP Packet*
0x01	Configure-request
0x02	Configure-ack
0x03	Configure-nak
0x04	Configure-reject
0x05	Terminate-request
0x06	Terminate-ack
0x07	Code-reject



IP datagram encapsulation in PPP



PPP session example



PPP session example (cont'd)