Data Link Control (DLC)

CS44 Data Communication and Networking

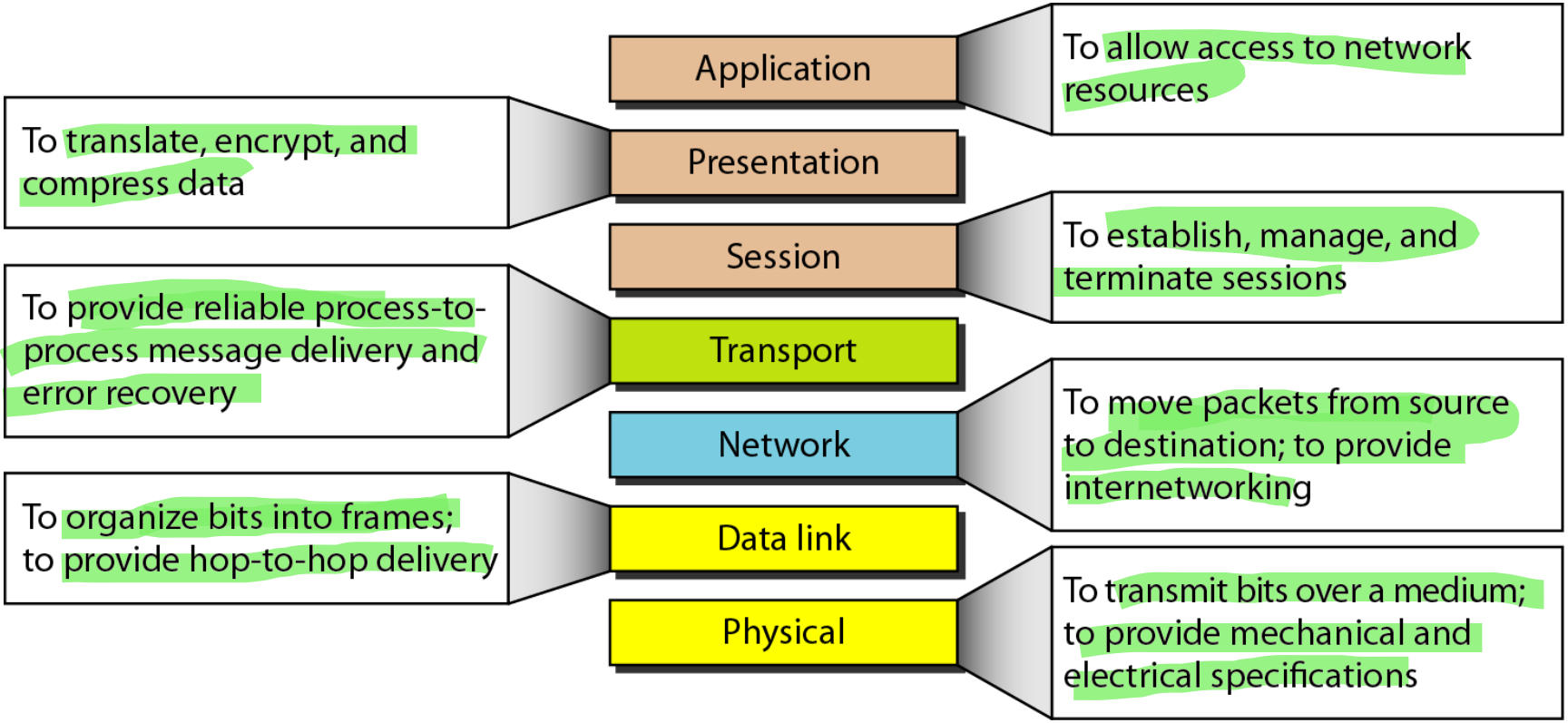
ML and Dr. SSC

Department of Computer Science and Engineering
Ramaiah Institute of Technology Bangalore

Syllabus

- Data-Link Control: Framing: Character-Oriented Framing, Bit-Oriented Framing. Error Control:
- Types of Errors, Block Coding: Error Detection, Hamming Distance, Linear Block Codes: Parity-
- Check Code. Cyclic Codes: Cyclic Redundancy Check, Point-to-Point Protocol, Media Access
- Control – Random Access- CSMA, CSMA/CD, CSMA/CA, Controlled access.

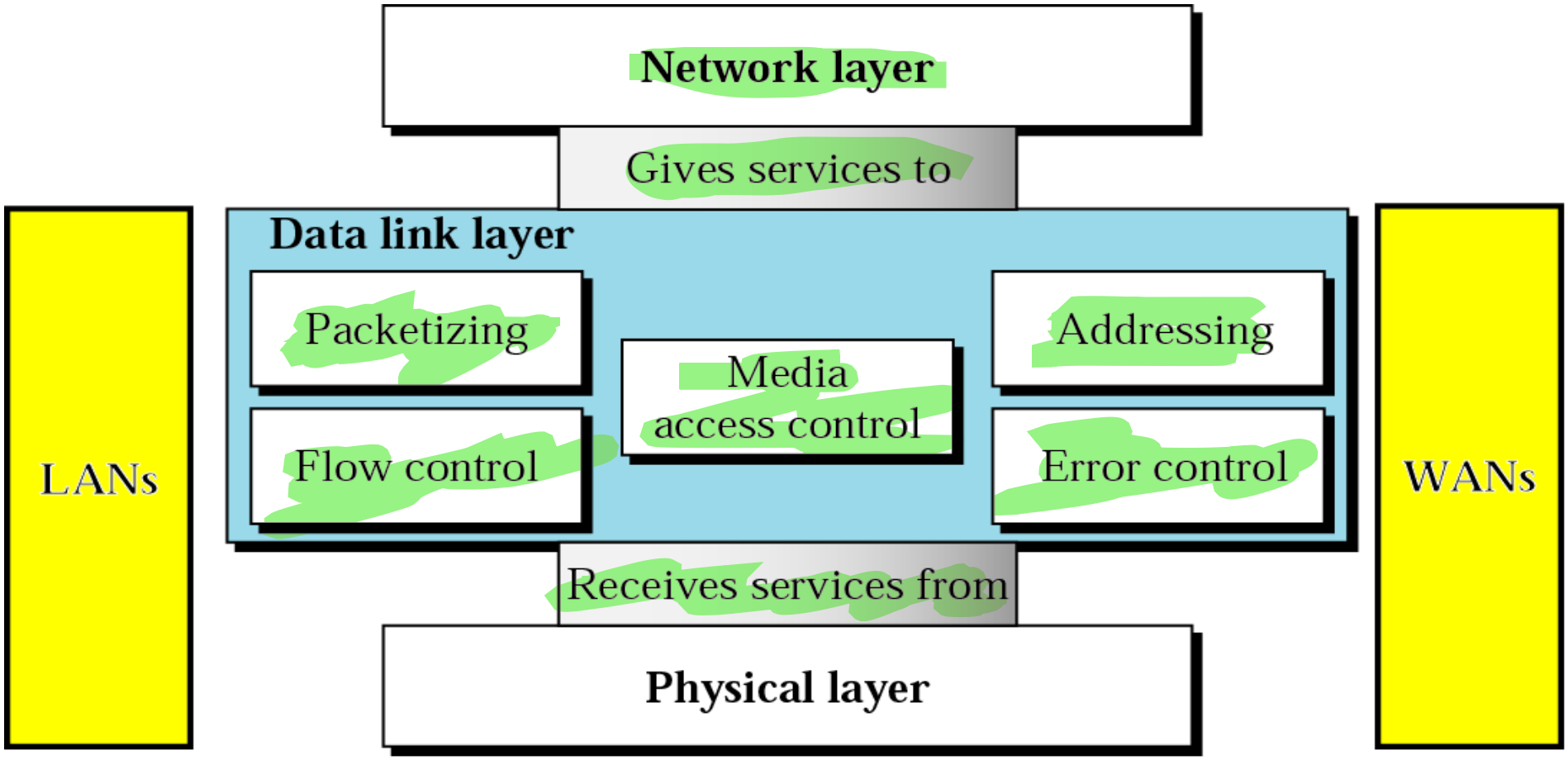
Summary of Layers



Revision

- Physical layer
  - Data transmission in the physical layer means moving bits in the form of a signal from the source to the destination.
  - The physical layer provides bit synchronization to ensure that the sender and receiver use the same bit durations and timing
- Data-link layer needs to pack bits into frames, so that each frame is distinguishable from another

Data Link Layer - functions



Data Link Layer Functions

- The data-link layer is divided into two sublayers
  - Data link control (DLC)
    - Deals with the design and procedures for communication between two adjacent nodes: node-to-node communication
    - DLC functions include
      - Framing
      - flow and error control
  - Media access control (MAC)
    - Deals with how to share the link
- To implement data link control, we need protocols that provide smooth and reliable transmission of frames between nodes
  - a set of rules that need to be implemented in software and run by the two nodes involved in data exchange at the data link layer

Framing

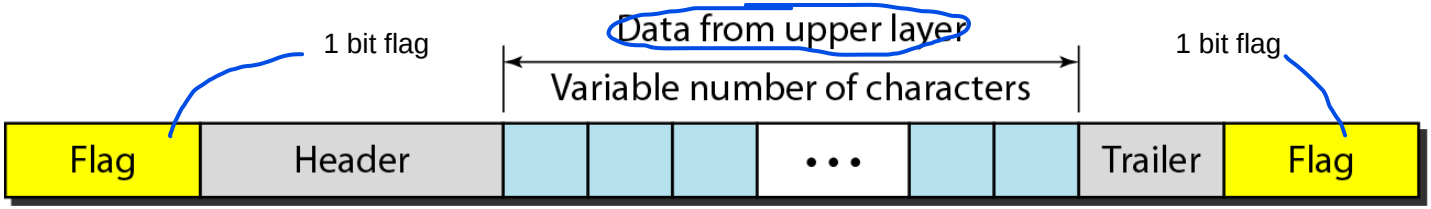
- *Framing in the data-link layer separates a message from one source to a destination* by adding a sender address and a destination address
  - The destination address defines where the packet is to go
  - The sender address helps the recipient acknowledge the receipt
- Why the whole message not packed in one frame?
  - A frame can be very large, making flow and error control very inefficient
  - When a message is carried in one very large frame, even a single-bit error would require the retransmission of the whole frame
- When a message is divided into smaller frames, a single-bit error affects only that small frame
  - Frame Size?
    - Fixed-Size Framing
    - Variable-Size Framing

Framing …

- Fixed-Size Framing
  - no need for defining the boundaries of the frames
  - the size itself can be used as a delimiter
  - An example of this type of framing is the ATM WAN, which uses frames of fixed size called *cells*
    - ATM: Asynchronous Transfer Mode(connection oriented, high-speed network technology that is used in both LAN and WAN over optical fiber and operates upto gigabit speed)
- Variable-Size Framing -
  - need a way to define the end of one frame and the beginning of the next
  - Historically, two approaches were used for this purpose:
    - a character-oriented *(or byte-oriented)* approach
    - a bit-oriented approach
  - prevalent in local-area networks

Character-Oriented Framing

- Data to be carried are 8-bit characters from a coding system such as ASCII
- The header (also multiples of 8 bits) normally carries
  - the source and destination addresses
  - other control information
  - the trailer, which carries error detection redundant bits
- To separate one frame from the next, an 8-bit (1-byte) flag is added at the beginning and the end of a frame
  - The flag composed of protocol-dependent special characters



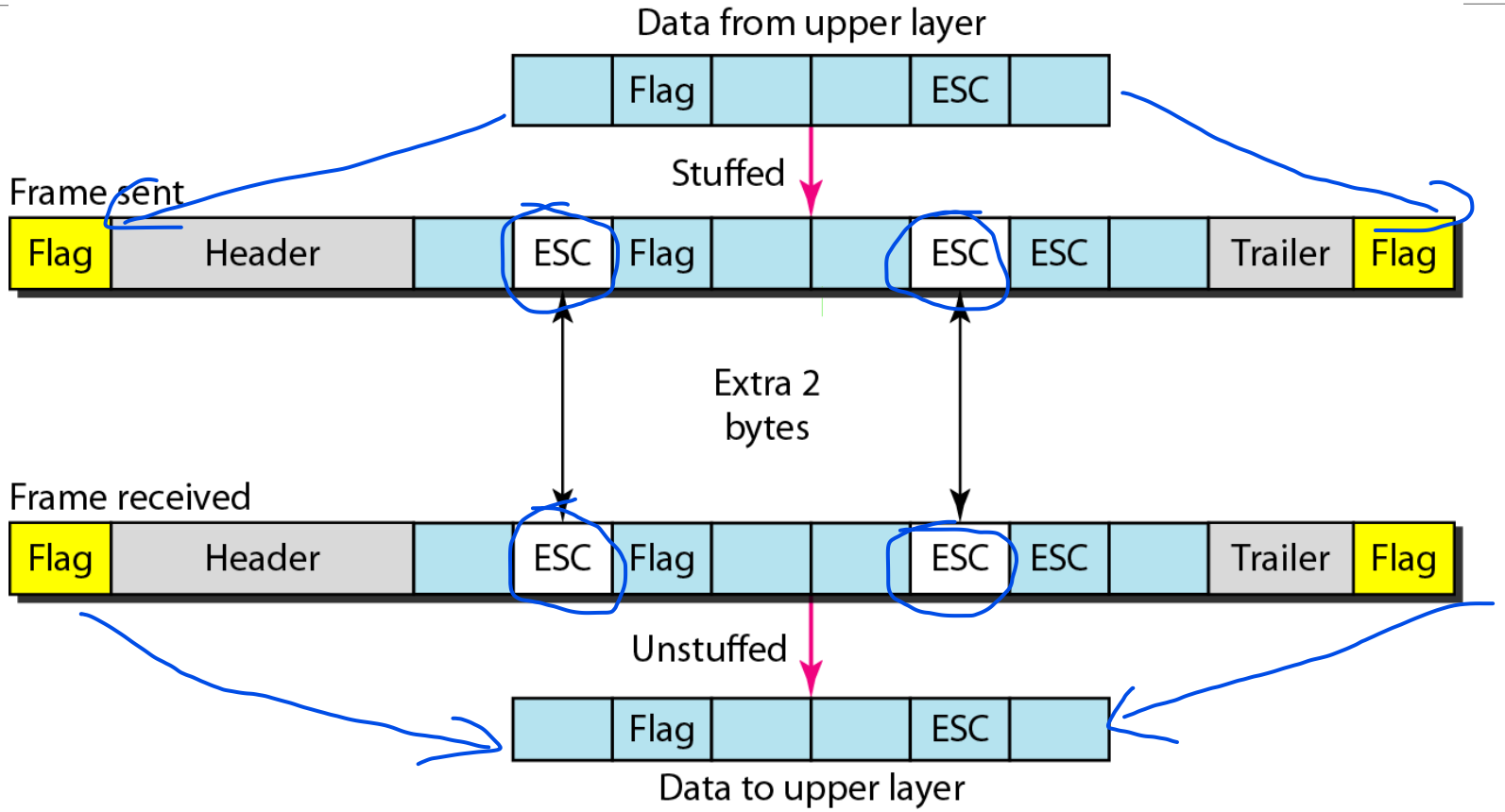
Problem with flag

- Text communication: flag could be selected to be any character not used for text communication
- Sending other types of information such as graphs, audio, and video: any character used for the flag could also be part of the information
- If flag is a part of the information, the receiver, when it encounters this pattern in the middle of the data, thinks it has reached the end of the frame

Solution to Flag Problem

- Add a byte-stuffing strategy to character-oriented framing
- In **byte stuffing (or character stuffing), a special** byte is added to the data section of the frame when there is a character with the same pattern as the flag
- The data section is stuffed with an extra byte
  - called the *escape character (ESC) and has a predefined bit pattern*
- *Whenever the* receiver encounters the ESC character, it removes it from the data section and treats the next character as data, not as a delimiting flag
- If the escape character is part of the text, an extra one is added to show that the second one is part of the text

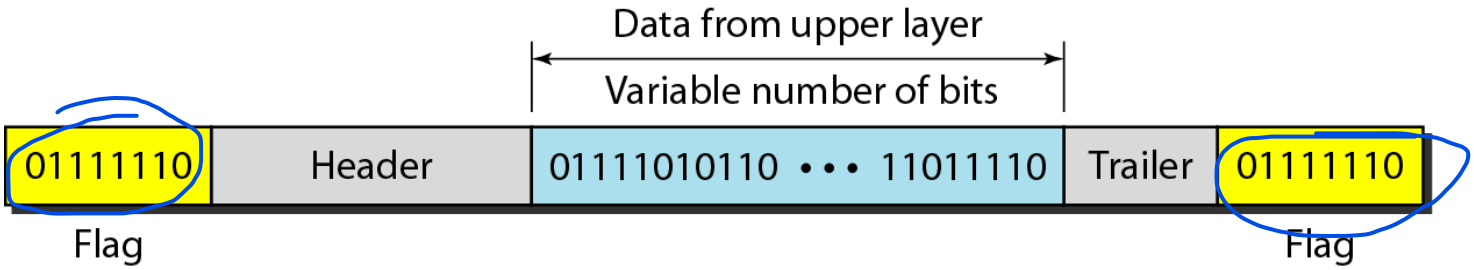
Byte stuffing and unstuffing



- The universal coding systems in use today, such as Unicode, have 16-bit and 32-bit characters that conflict with 8-bit characters
  - bit-oriented protocols

Bit-Oriented Framing

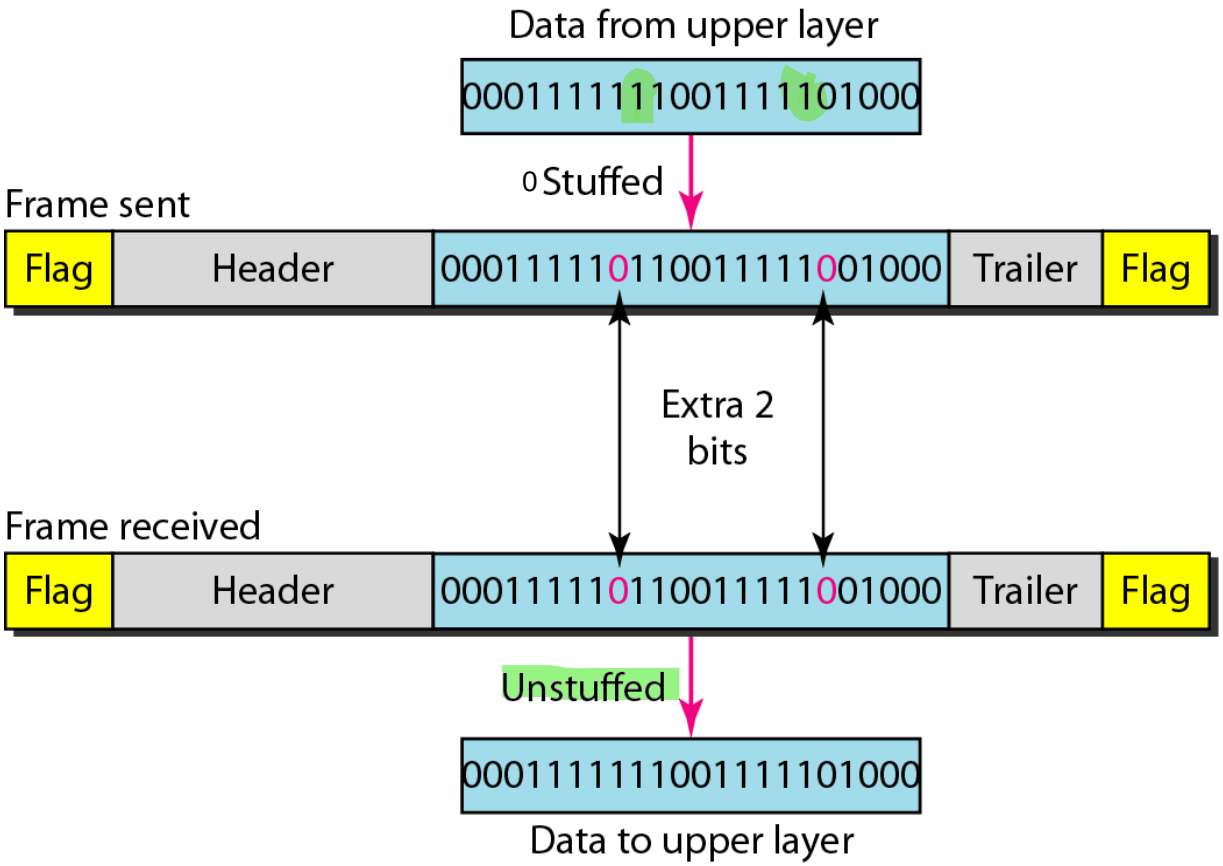
- Data section of a frame is a sequence of bits to be interpreted by the upper layer as text, graphic, audio, video, and so on
- In addition to headers (and possible trailers), need a delimiter to separate one frame from the other
  - Most protocols use a special 8-bit pattern flag, 01111110, as the delimiter to define the beginning and the end of the frame



Bit-Oriented Framing

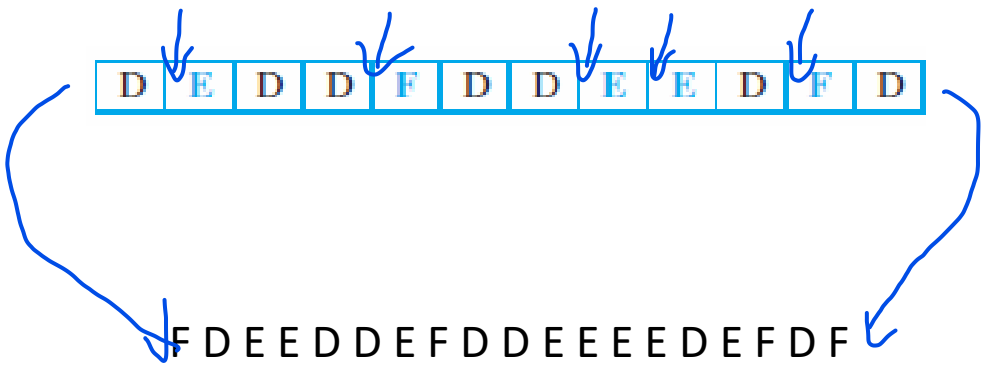
- Bit stuffing is the process of adding one extra 0 whenever five consecutive 1s follow a 0 in the data, so that the receiver does not mistake the pattern 0111110 for a flag
- If the flaglike pattern 01111110 appears in the data, it will change to 011111010 (stuffed) and is not mistaken for a flag by the receiver
  - The real flag 01111110 is not stuffed by the sender and is recognized by the receiver

Bit-Oriented Framing



Example 1

- Byte-stuff the following frame payload in which
  - E is the escape byte
  - F is the flag byte
  - D is a data byte
  - other than an escape or a flag character.



Example 2

- Unstuff the following frame payload in which
  - E is the escape byte
  - F is the flag byte
  - D is a data byte
  - other than an escape or a flag character.

E D F D D F E D D D

Example 3

- Bit-stuff the following frame payload:

00011111110011111101000111111110000111

00011111011001111100100011111011111010000111

Example 4

- Unstuff the following frame payload:

00011111000001111101110100111011111000001111

00011111000001111101110100111011111000001111
00011111000011111111010011101111100001111

Programming Examples

- Write and test a program that simulates the byte stuffing and byte unstuffing
- Write and test a program that simulates the bit stuffing and bit unstuffing

Error Control

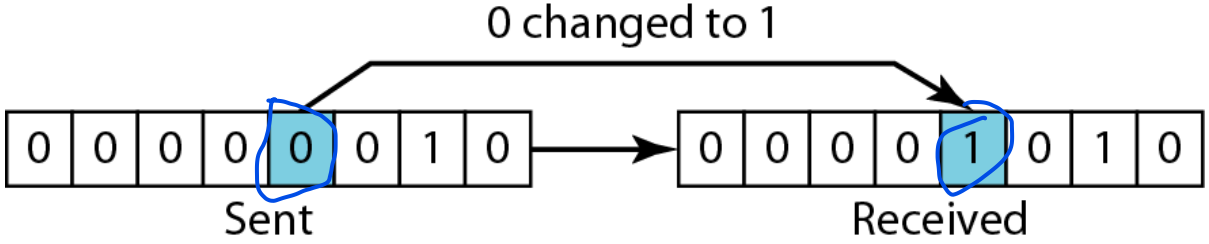
- At the data-link layer, corrupted frame needs to be corrected before it continues its journey to other nodes
- Detecting errors
  - block coding
  - cyclic codes – CRC
  - checksums
- Correcting errors
  - Forward error correction
    - Hamming distance
    - XORing of packets
    - interleaving chunks
    - compounding high and low resolutions packets
  - Automatic repeat request (ARQ)
- Most link-layer protocols simply discard the frame and let the upper-layer protocols handle the retransmission of the frame

Error Detection and Correction Issues

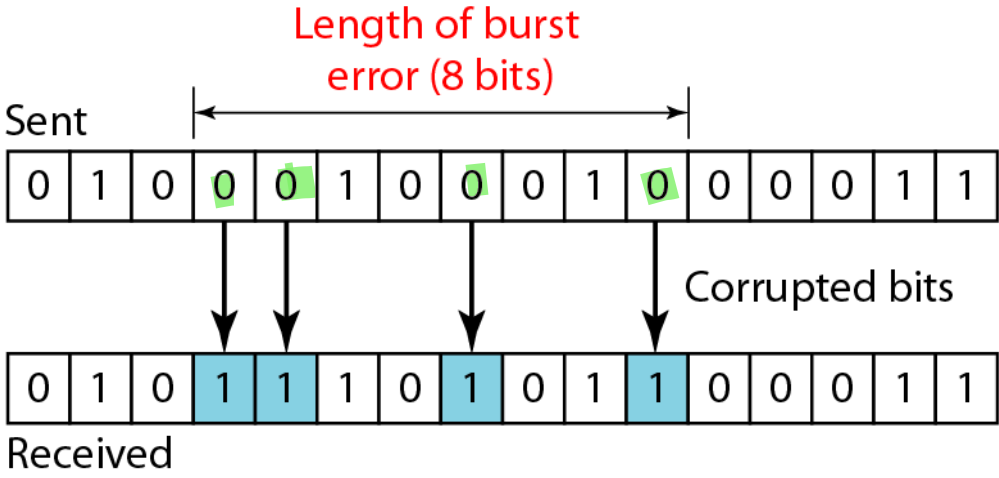
- Types of Errors
  - Single-bit and burst error
- Redundancy
  - some extra bits with our data
- Detection versus Correction
- Coding
  - block coding and convolution coding

Types of Errors

- Single-bit errors: only 1 bit of a given data unit is changed from 1 to 0 or from 0 to 1



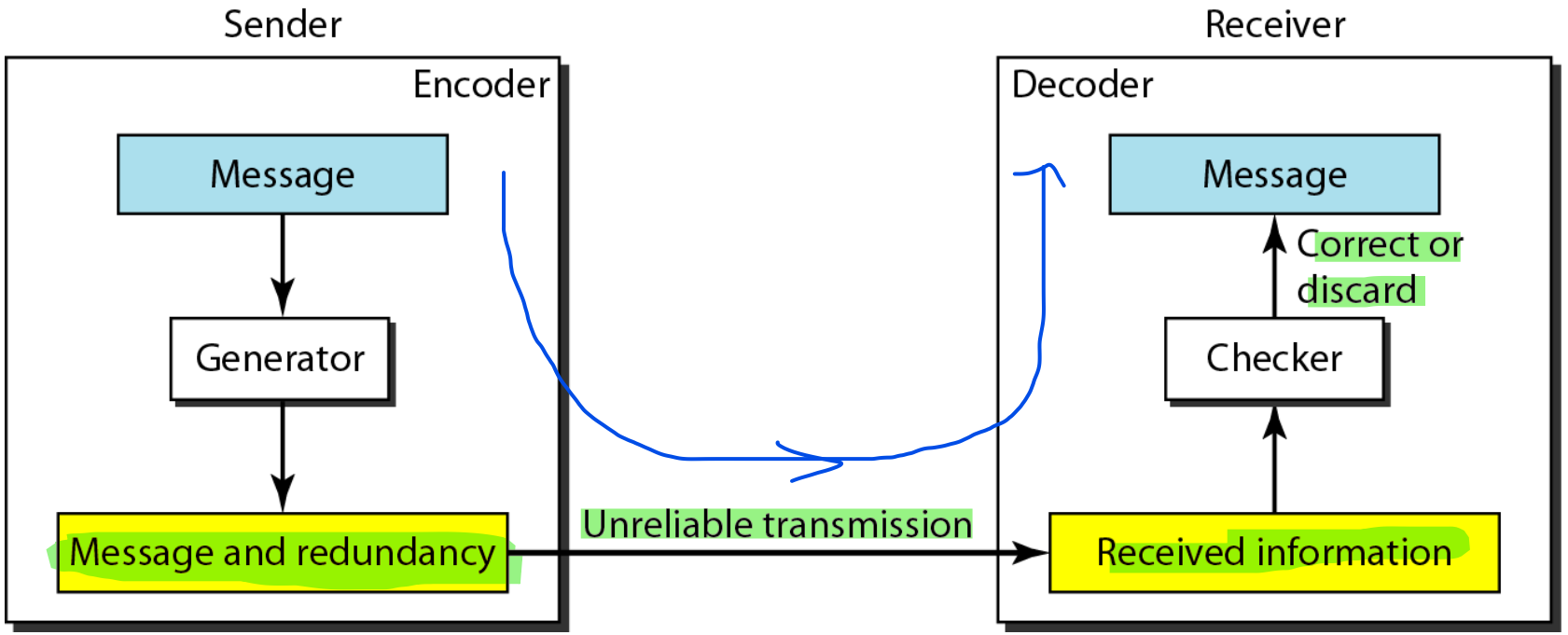
- Burst errors: 2 or more bits in the data unit is changed from 1 to 0 or from 0 to 1



- more likely to occur because the duration of the noise signal is normally longer than the duration of 1 bit
- The number of bits affected depends on the data rate and duration of noise
- Consider a noise of 1/100 second, 1 kbps data rate can affect 10 bits whereas 1 Mbps data rate affect 10,000 bits

Redundancy

- To detect or correct errors, redundant bits (some extra bits) of data must be added by sender and removed by receiver
  - allows the receiver to detect or correct corrupted bits



Coding

- Process of adding redundancy for error detection or correction creates a relationship between the redundant bits and the actual data bits
- Two types:
  - Block codes
    - Divides the data to be sent into a set of blocks
    - Extra information attached to each block
    - Memoryless
  - Convolutional codes
    - Treats data as a series of bits, and computes a code over a continuous series
    - The code computed for a set of bits depends on the current and previous input

Detection versus Correction

- correction of errors is more difficult than the detection
- error detection → only looking to see if any error has occurred not the number of corrupted bits
  - No difference between a single-bit error and a burst error
- error correction→ need to know the exact number of bits that are corrupted with their location in the message
  - 8-bit data unit
    - to correct a single error → need to consider eight possible error locations
    - to correct two errors → need to consider 28 (permutation of 8 by 2) possible error locations
  - Receiver find difficulty to correct 10 errors in a data unit of 1000 bits

Notes

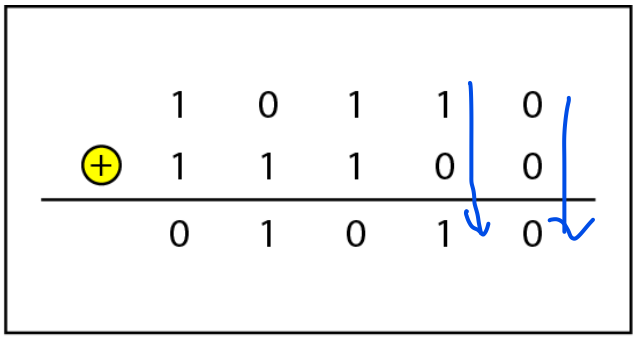
- An error-detecting/correcting code can detect/correct only the types of errors for which it is designed
  - Other types of errors may remain undetected.
- There is no way to detect/correct every possible error!

XOR Operation

- Main operation for computing error detection/correction codes
  - Simpler and faster than addition; no carry bit!
- Similar to modulo-2 addition

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

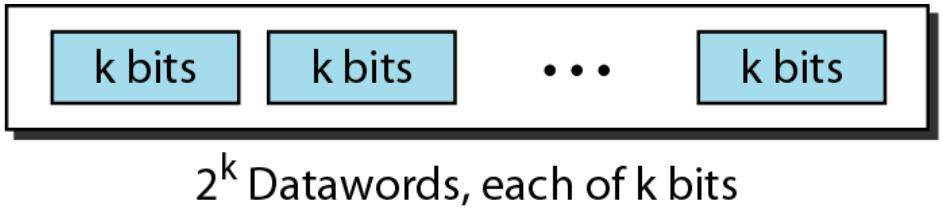
b. Two bits are different, the result is 1.



c. Result of XORing two patterns

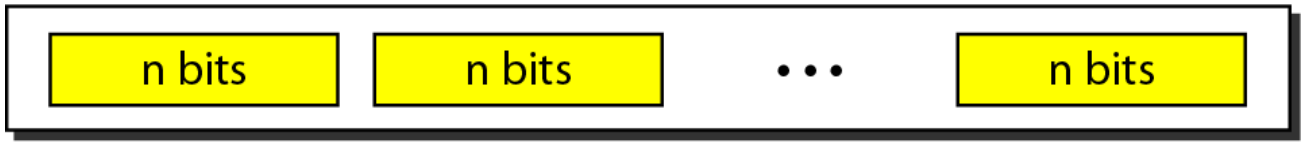
Block Coding

- Message is divided into *k*-bit blocks
  - Known as *datawords*
- *r* redundant bits are added
  - Blocks become $n=k+r$ bits
  - Known as *codewords*



2^k Datawords, each of k bits

- $2^n - 2^k$ codewords that are not used
  - invalid or illegal
- If the receiver receives an invalid codeword, this indicates that the data was corrupted during transmission.



2^n Codewords, each of n bits (only 2^k of them are valid)

Example: *4B/5B Block Coding*

Data	Code	Data	Code
0000	11110	1000	10010
0001	01001	1001	10011
0010	10100	1010	10110
0011	10101	1011	10111
0100	01010	1100	11010
0101	01011	1101	11011
0110	01110	1110	11100
0111	01111	1111	11101

$k = 4$

$n = 5$

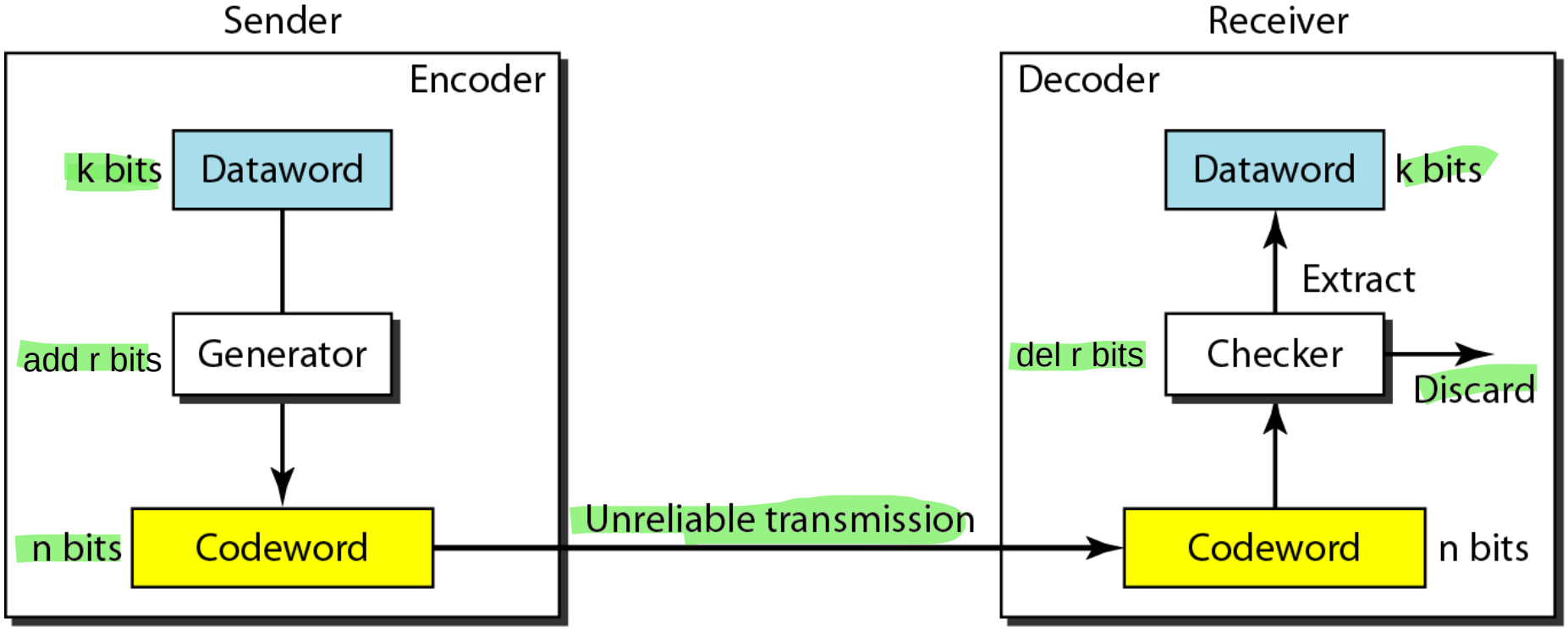
$r = 1$

How can errors be detected by using block coding?

- Two conditions the receiver follows to detect a change in the original codeword

1. The receiver has (or can find) a list of valid codewords
2. The original codeword has changed to an invalid one

Error Detection in Block Coding



if the codeword is corrupted during transmission but the received word still matches a valid codeword, the error remains undetected.

Error-detecting Code Example

- Let us assume that $k = 2$ and $n = 3$ r = 1
- *The list of datawords and codewords*

Dataword	*Codeword*	*Dataword*	*Codeword*
00	000	10	101
01	011	11	110

- sender encodes the dataword 01 as 011 and sends it to the receiver
  - receiver receives 011→ a valid codeword → extracts 01 dataword
  - receiver receives 111 →not a valid codeword → discarded
  - receiver receives 000→ a valid codeword → incorrectly extracts 00 dataword → Two corrupted bits have made the error undetectable

Hamming Distance

***Hamming Distance** between two words is the number of differences between corresponding bits.*

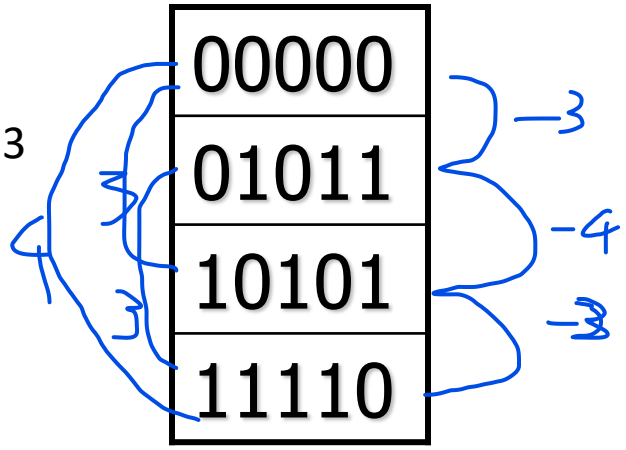
- Hamming distance between the received codeword and the sent codeword is the number of bits that are corrupted during transmission → *important for error* detection
- *if the Hamming* distance between the sent and the received codeword is not zero, the codeword has been corrupted during transmission
  - Found by XOR operation ($\oplus$) on the two words and count the number of 1s in the result
- *d(000, 011) is 2 because (000 $\oplus$ 011) is 011 (two 1s)*
- *d(10101, 11110) is 3 because (10101 $\oplus$ 11110) is 01011 (three 1s)*

Minimum Hamming Distance

The minimum Hamming distance is the smallest Hamming distance between all possible pairs in a set of codewords.

- Find the minimum Hamming Distance of the following codebook

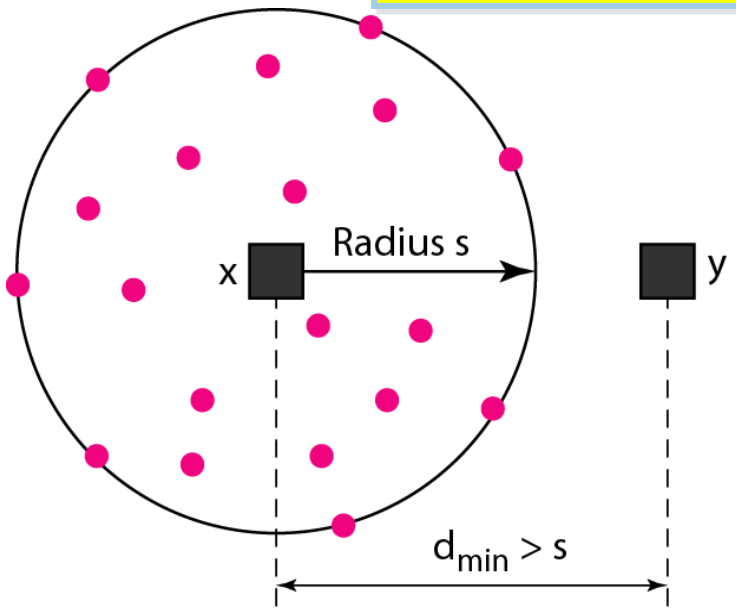
minimum Hamming Distance = 3

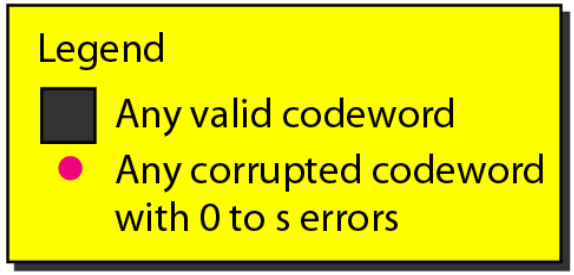


Detection Capability of Code

- To guarantee the ***detection*** of up to ***s***-bit errors, the minimum Hamming distance in a block code must be

$$d_{min} = s + 1$$





A code scheme with Hamming distance $d_{min} = 4$ *guarantees the detection of up to* three errors (*d = s + 1 or s = 3)*

Error-detecting Code Example

- Let us assume that *k = 2 and n = 3. The list of datawords and codewords*

Dataword	*Codeword*	*Dataword*	*Codeword*
00	000	10	101
01	011	11	110

Error-detecting Code Example

- Let us assume that $k = 2$ and $n = 3$. *The list of datawords and codewords*

Dataword	*Codeword*	*Dataword*	*Codeword*
00	000	10	101
01	011	11	110

- The minimum Hamming distance =2
  - guarantees detection of only a single error

First codeword	000	000	000	011	011	101
Second codeword	011	101	110	101	110	110
result	011	101	110	110	101	011
Hamming distance	2	2	2	2	2	2

- If the codeword 101 is sent and one error occurs, the received codeword does not match any valid codeword
- If two errors occur, however, the received codeword may match a valid codeword and the errors are not detected

Linear Block Codes

- Two types
  - *linear block codes – today's choice*
    - requires the knowledge of abstract algebra (particularly Galois fields)
    - a code in which the exclusive OR (addition modulo-2) of two valid codewords creates another valid codeword
  - *nonlinear block codes - not as* widespread because their structure makes theoretical analysis and implementation difficult

Dataword	*Codeword*	*Dataword*	*Codeword*
00	000	10	101
01	011	11	110

- This code in Table is a linear block code because the result of XORing any codeword with any other codeword is a valid codeword

First codeword	000	000	000	011	011	101
Second codeword	011	101	110	101	110	110
result	011	101	110	110	101	011
Valid/invalid	V	V	V	V	V	V

Minimum Distance for Linear Block Codes

- the number of 1s in the nonzero valid codeword with the smallest number of 1s
- So the minimum Hamming distance is $d_{min} = 2$.

Dataword	*Codeword*	*Dataword*	*Codeword*
00	000	10	101
01	011	11	110

First codeword	**000**	**000**	**000**	**011**	**011**	**101**
Second codeword	011	101	110	101	110	110
result	011	101	110	110	101	011
Hamming distance	2	2	2	2	2	2

Common Detection Methods

- Parity check
- Cyclic Redundancy Check
- Checksum

Parity Check

- Most common linear block code, least complex
- Single bit (*called the parity bit)* is added to a block
  - a *k-bit dataword is changed to an n-bit codeword* where *n = k + 1*
- Two schemes:
  - Even parity – Maintain even number of 1s
    - *Extra bit used to make the total* number of 1s in the codeword even
    - E.g., 1011 $\rightarrow$ 1011**1**
  - Odd parity – Maintain odd number of 1s
    - *Extra bit used to make the total* number of 1s in the codeword odd
    - E.g., 1011 $\rightarrow$ 1011**0**
- Minimum Hamming distance is $d_{min} = 2$
  - *means that the code is a single-bit error-detecting code*

Parity Check Examples

- A parity-check code *k = 2 and n = 3*

Dataword	codeword
00	00**0**
01	01**1**
10	10**1**
11	11**0**

- A parity-check code with *k = 4 and n = 5*

Dataword	*Codeword*	*Dataword*	*Codeword*
0000	00000	1000	10001
0001	00011	1001	10010
0010	00101	1010	10100
0011	00110	1011	10111
0100	01001	1100	11000
0101	01010	1101	11011
0110	01100	1110	11101
0111	01111	1111	11110

Example: *Parity Check*

Suppose the sender wants to send the word *world*. In ASCII the five characters are coded (with ***even parity***) as

1110111 1101111 1110010 1101100 1100100

The following shows the actual bits sent

1110111**0** 1101111**0** 1110010**0** 1101100**0** 1100100**1**

Receiver receives this sequence of words:

11111110 11011110 11101100 11011000 11001001

Which blocks are accepted? Which are rejected?

11111110 11011110 11101100 11011000 11001001

Example: *Parity Check*

Suppose the sender wants to send the word *world*. In ASCII the five characters are coded (with ***even parity***) as

1110111 1101111 1110010 1101100 1100100

The following shows the actual bits sent

1110111**0** 1101111**0** 1110010**0** 1101100**0** 1100100**1**

Receiver receives this sequence of words:

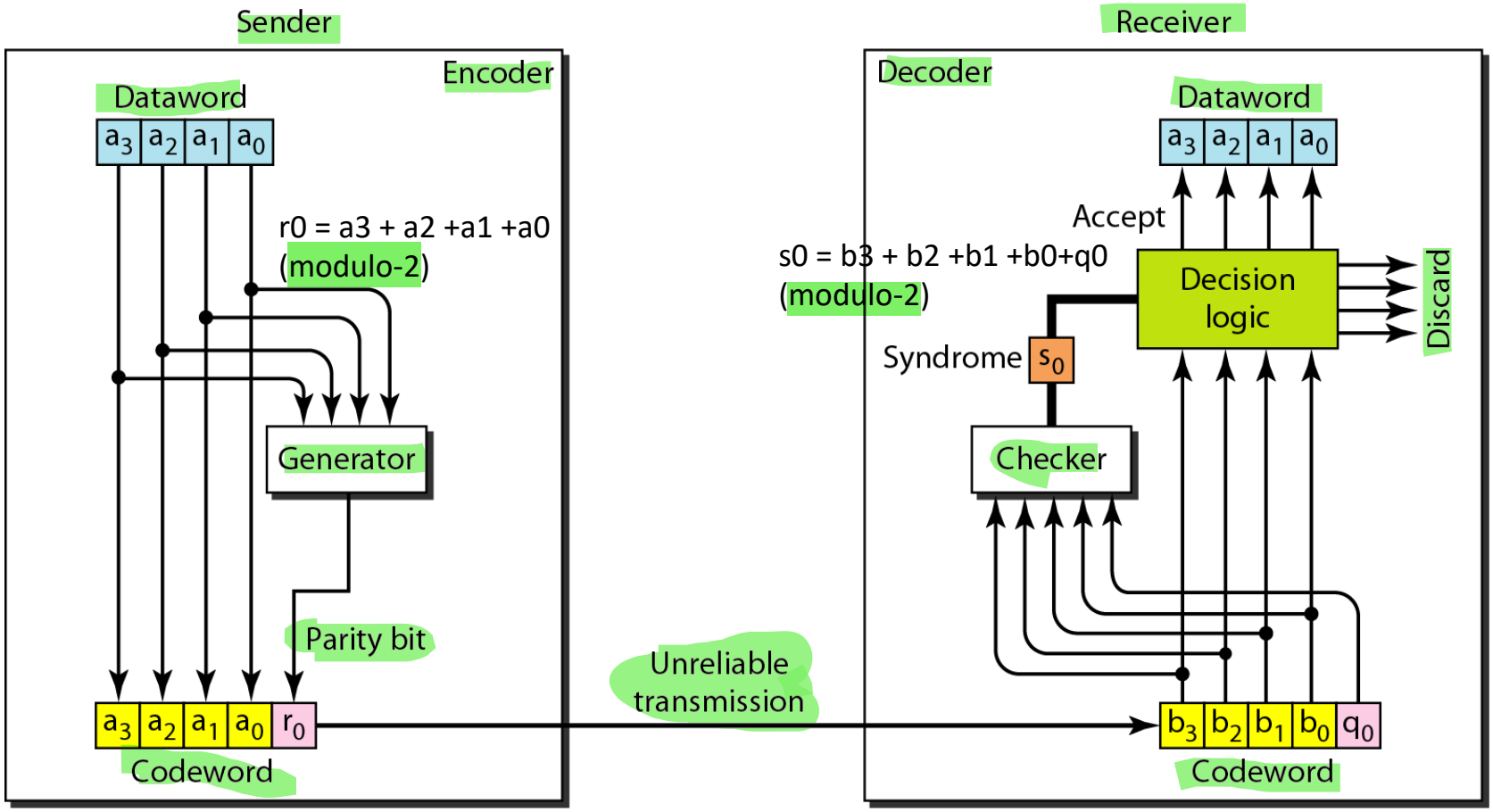
11111110 11011110 11101100 11011000 11001001

Which blocks are accepted? Which are rejected?

11111110 11011110 11101100 11011000 11001001

R A R A A

Parity-Check: Encoding/Decoding



Rule to make total number of 1s in the codeword is even:
If the number of 1s is even, r0=0 else r0=1

Rule of the decision logic analyzer : If s0=0, no detectable error in the received codeword (accept) else detectable error (discard)

Parity-Check: transmission scenarios

- Assume the sender dataword 1011 with codeword is 10111, which is sent to the receiver.

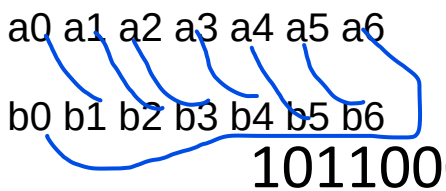
Receiver Scenario	received codeword	syndrome	data word	Note
No error occurs	10111	0	1011	-
One single-bit error - changes a1	10011	1	Not created	-
One single-bit error changes r0	10110	1	Not created	none of the dataword bits are corrupted still no dataword is created because the code is not sophisticated enough to show the position of the corrupted bit.
error changes r0 and a second error changes a3	00110	0	0011	dataword is wrongly created due to the syndrome value. The simple parity-check decoder cannot detect an even number of errors. The errors cancel each other out and give the syndrome a value of 0.
3-bits—a3, a2, and changed by errors	01011	1	Not created	shows that the simple parity check, guaranteed to detect one single error, can also find any odd number of errors.

A parity-check code can detect an odd number of errors.

Cyclic Codes

- Special linear block codes
- In a cyclic code, rotating a codeword always results in another codeword
  - n the rightmost equation, the last bit of the first word is wrapped around and becomes the first bit of the second word
- Example: Consider the bits in the first word *a0 to a6*, *and the bits in the second word b0 to b6*, *we can shift the bits by using the following*

b1 = a0
b2 = a1
b3 = a2
b4 = a3
b5 = a4
b6 = a5
b0 = a6



1011000 is a codeword → cyclically left-shift

Result is 0110001 → also a codeword

- Cyclic codes → correct errors

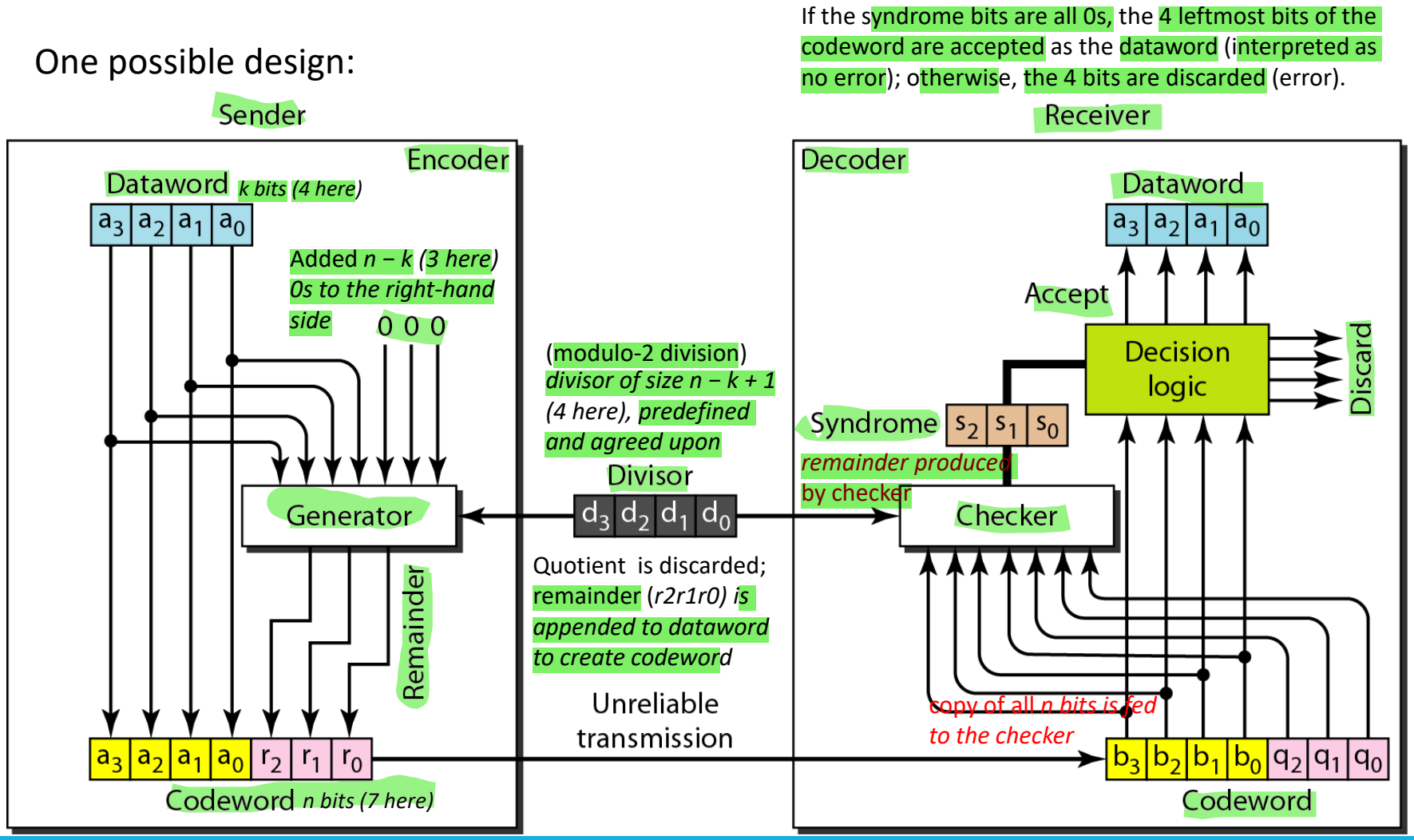
Cyclic Redundancy Check (CRC)

- A subset of cyclic codes
- Has both the linear and cyclic properties
- **Used in networks** such as LANs and WANs
- *A CRC code with C(7, 4)* n= 7, k = 4, r = 3

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

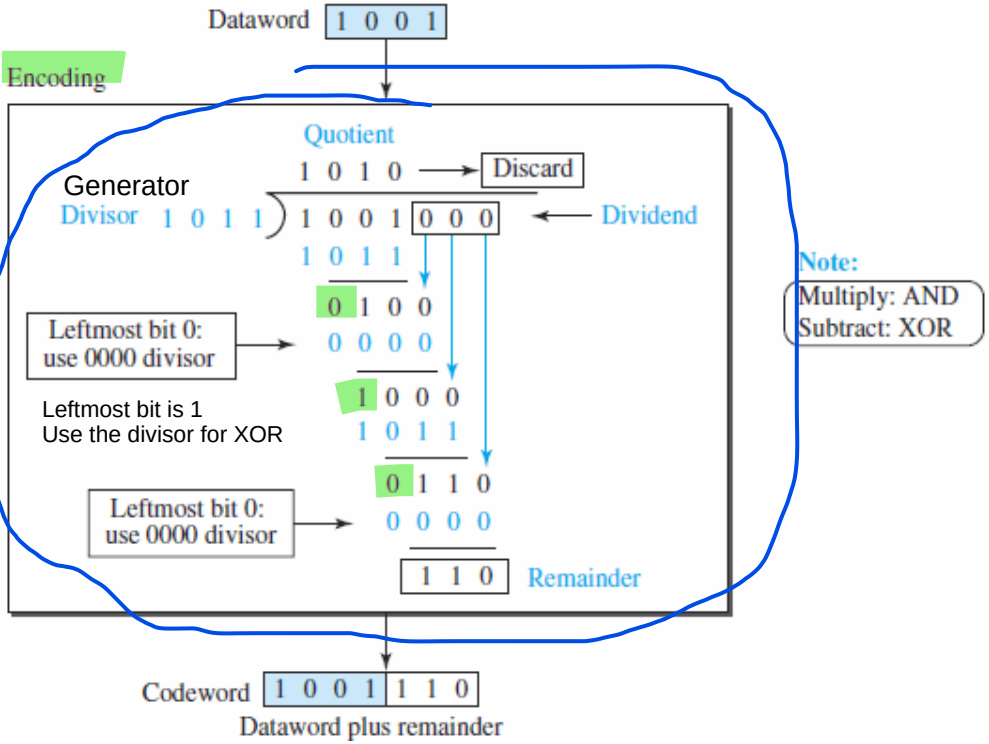
CRC Encoder/Decoder

One possible design:



CRC Generator

Division in CRC encoder:

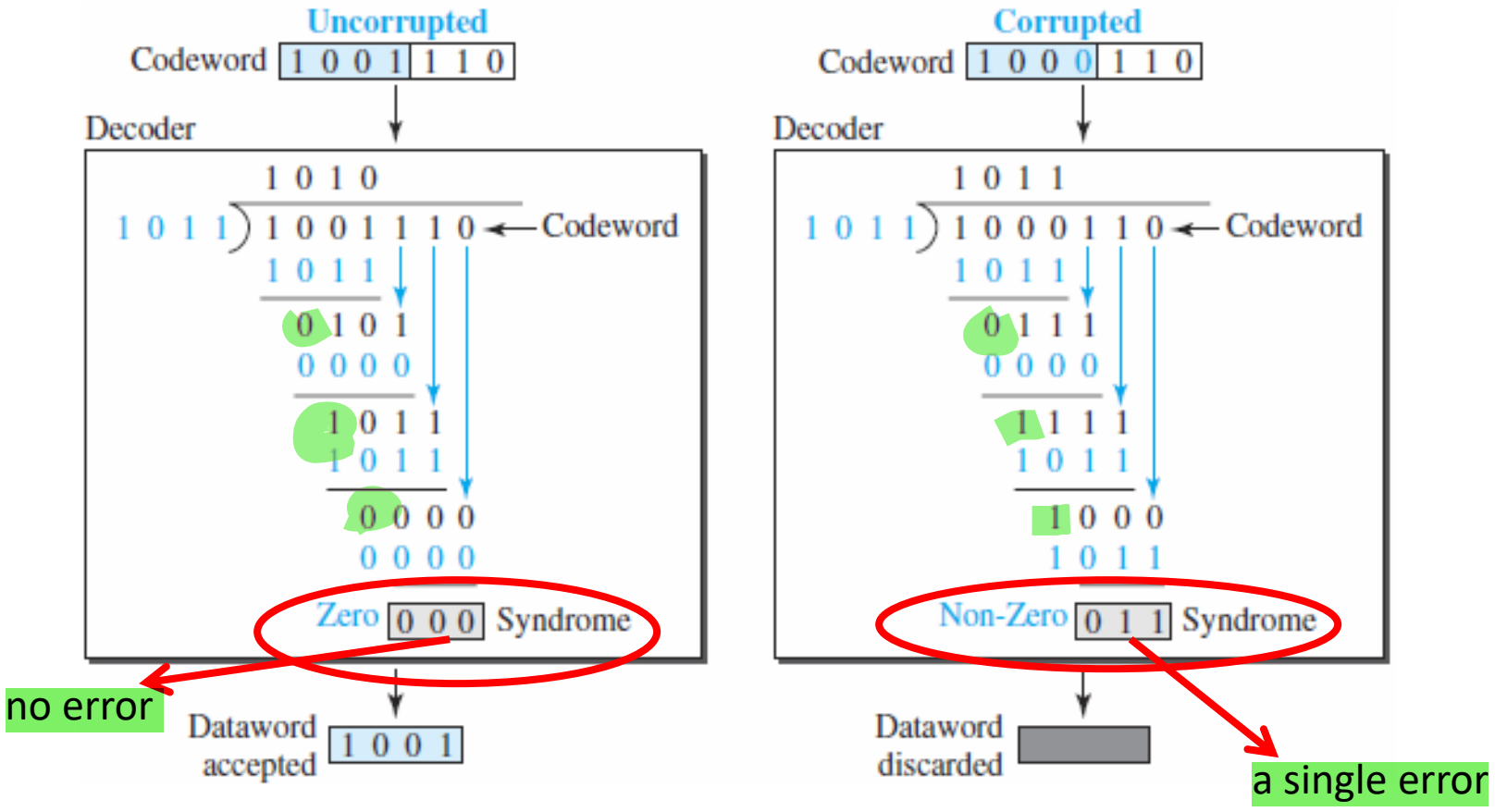


How the divisor 1011 is chosen?
Depends on the expectation we have from the code- we will discuss this in polynomials.

- Modulo-2 binary division
- Addition and subtraction are the same → use XOR operation to do both
- opy of the dEach step: a civisor is XORed with the 4 bits of the dividend
  - Result (remainder) is 3 bits (in this case), which is used for the next step after 1 extra bit is pulled down to make it 4 bits long
  - Note: If the leftmost bit of the dividend (or the part used in each step) is 0, the step cannot use the regular divisor; we need to use an all-0s divisor
  - When there are no bits left to pull down, we have a result
- 3-bit remainder forms the check bits (*r2, r1, and r0). They are appended to the dataword to create the* codeword

Checking CRC

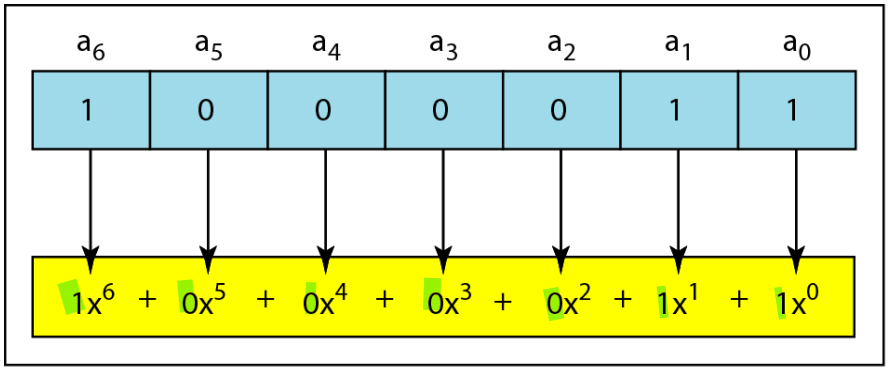
Division in the CRC decoder for two cases: division process as the encoder



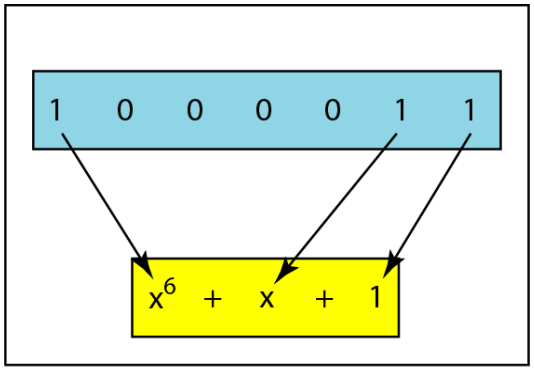
Remainder of the division is the syndrome

Polynomial Representation

- More common representation than binary form in cyclic codes
- Easy to analyze
- Divisor is commonly called *generator polynomial*
- Binary pattern can be represented as a polynomial with coefficients of 0 and 1
  - The power of each term shows the position of the bit; the coefficient shows the value of the bit.



a. Binary pattern and polynomial



b. Short form

Polynomial Operations

- Degree of a polynomial → highest power in the polynomial
  - *1 less* than the number of bits in the pattern
  - the degree of the polynomial $x^6 + x + 1$ *is* 6
    - The bit pattern in this case has 7 bits
- Adding and Subtracting Polynomials
  - Coefficients are only 0 and 1 (modulo-2)
    - Addition and subtraction are the same
    - Adding or subtracting is done by combining terms and deleting pairs of identical terms
      - adding $x^5 + x^4 + x^2$ *and* $x^6 + x^4 + x^2$ *gives* just $x^6 + x^5$
      - If we add three polynomials and we get x^2 three times, we delete a pair of them and keep the third
- Multiplying Terms: Just add the powers
  - For example, $x^3 \times x^4$ is x^7
- Dividing Terms: just subtract the power of the second term from the power of the first
  - For example, x^5/x^2 is x^3

Polynomial Operations …

- Multiplying Two Polynomials: done term by term
  - Each term of the first polynomial must be multiplied by all terms of the second
  - The result is then simplified, and pairs of equal terms are deleted

$$(x^5+x^3+x^2+x)(x^2+x+1) = x^7+x^6+x^5+x^5+x^4+x^3+x^4+x^3+x^2+x^3+x^2+x$$
$$= x^7+x^6+x^3+x$$

- Dividing One Polynomial by Another: same as the binary division → discussed for an encoder
  - Divide the first term of the dividend by the first term of the divisor to get the first term of the quotient
  - Multiply the term in the quotient by the divisor and subtract the result from the dividend
  - Repeat the process until the dividend degree is less than the divisor degree

Polynomial Operations …

- ***Shifting:*** A binary pattern is often shifted a number of bits to the right or left
- Shifting to the left means adding extra 0s as rightmost bits
  - accomplished by multiplying each term of the polynomial by x^m, *where m is the number of shifted bits*
- Shifting to the right means deleting some rightmost bits
  - *accomplished by* dividing each term of the polynomial by x^m
- Note: Polynomial representation do not have negative powers
- Also note that when we concatenate two bit patterns, we shift the first polynomial to the left and then add the second polynomial.

Shifting left 3 bits: 10011 becomes 10011000	$x^4 + x + 1$ becomes $x^7 + x^4 + x^3$
Shifting right 3 bits: 10011 becomes 10	$x^4 + x + 1$ becomes x

Criteria for Good 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 not divide $x^t + 1$, for t between 2 and n – 1
  4. It should have the factor x + 1.

Standard CRC Polynomials

- Some standard polynomials used by popular protocols for CRC generation along with the corresponding bit pattern

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**	LANs

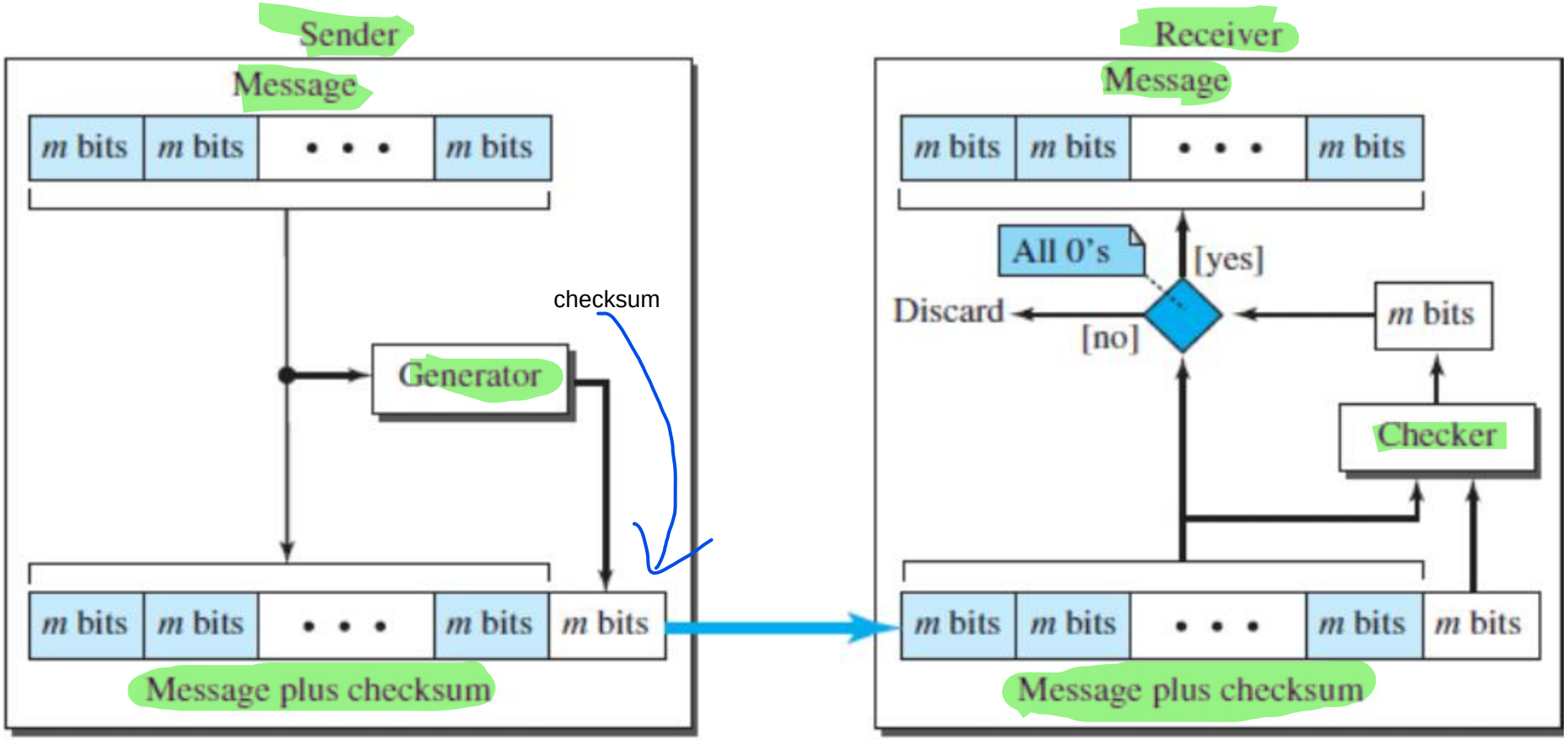
Advantages of Cyclic Codes

- A very good performance in detecting single-bit errors, double errors, an odd number of errors, and burst errors.
- Easily implemented in hardware and software
- Fast when implemented in hardware
- This has made cyclic codes a good candidate for many networks
- Other Cyclic Codes
  - Simple cyclic codes (discussed till now) - the check bits and syndromes can be calculated by simple algebra
  - Reed- Solomon code - More powerful polynomials that are based on abstract algebra involving Galois fields
    - used today for both detection and correction

Checksum

- An error-detecting technique that can be applied to a message of any length
- In the Internet, the checksum technique is mostly used at the network and transport layer rather than the data-link layer
- At the source
  - Message is first divided into *m-bit units*
  - *The generator then creates* an extra *m-bit unit called the checksum, which is sent with the message*
- *At the* destination
  - the checker creates a new checksum from the combination of the message and sent checksum
  - If the new checksum is all 0s, the message is accepted; otherwise, the message is discarded
- Note: in the real implementation, the checksum unit is not necessarily added at the end of the message; it can be inserted in the middle of the message

Checksum ...


Sender
Message
m bits
m bits
m bits
checksum
Generator
m bits
m bits
m bits
m bits
Message plus checksum
Receiver
Message
m bits
m bits
m bits
All 0's
[yes]
Discard
[no]
m bits
Checker
m bits
m bits
m bits
m bits
Message plus checksum

Checksum - simple example

- Suppose the message is a list of five 4-bit numbers that we want to send to a destination. In addition to sending these numbers, we send the sum of the numbers.
- For example, if the set of numbers is (7, 11, 12, 0, 6), we send (7, 11, 12, 0, 6, **36), where 36 is the sum of the original numbers.**
- The receiver adds the five numbers and compares the result with the sum
- If the two are the same, the receiver assumes no error, accepts the five numbers, and discards the sum
- Otherwise, there is an error somewhere and the message is not accepted.

Checksum - example

- Drawback - Each number can be written as a 4-bit word (each is less than 15) except for the sum
- One solution is to use **one's complement** arithmetic
  - In this arithmetic, we can represent unsigned numbers between 0 and $2^m - 1$ *using only m bits*
  - *If the number has more than m bits, the extra leftmost* bits need to be added to the *m rightmost bits*

One's complement arithmetic

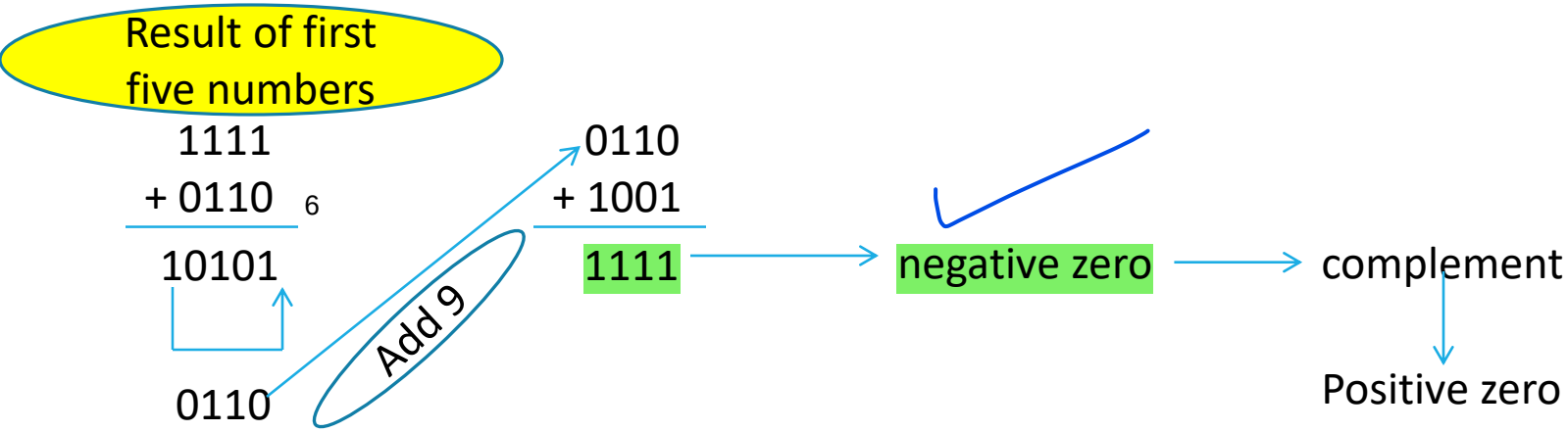
- Complement of a number → completing all bits (changing all 1s to 0s and all 0s to 1s)
  - same as subtracting the number from $2^m - 1$
- Two 0s: one positive and one negative, which are complements of each other
  - The positive zero has all m bits set to 0
  - The negative zero has all bits set to 1 → it is $2^m - 1$
- If we add a number with its complement, we get a negative zero (a number with all bits set to 1)

Checksum - example

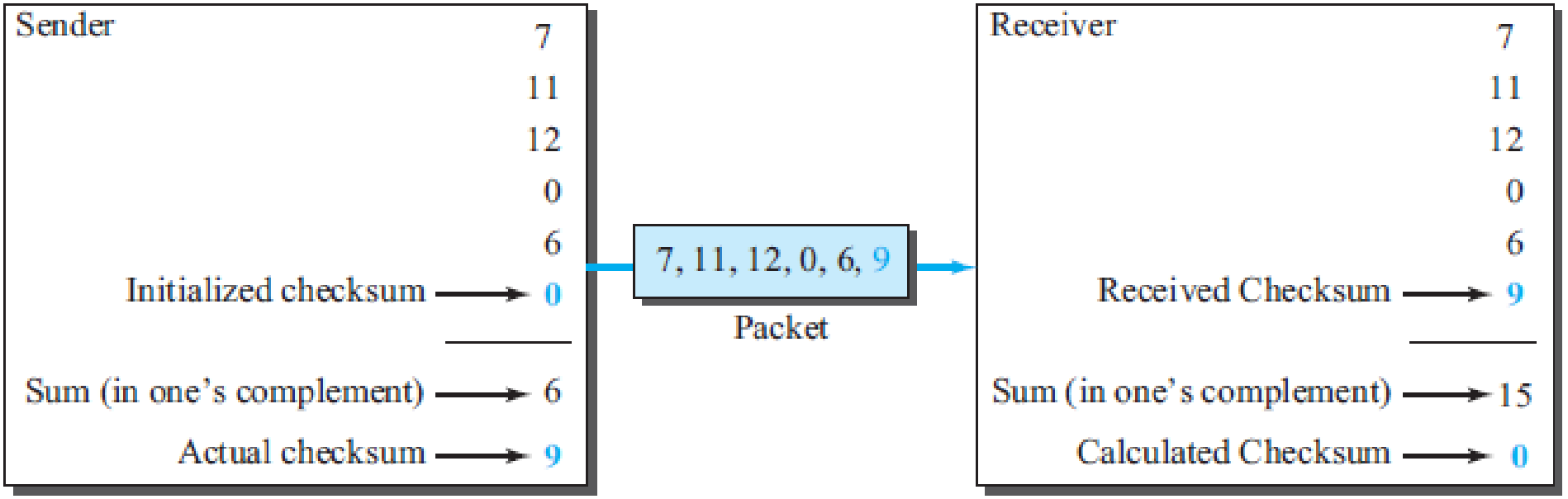
- How to make the receiver job easy?
- Send the complement of the sum→ checksum
- Eg. The sender adds all five numbers in one's complement to get the sum = 6
- The sender then complements the result to get the checksum = 9, which is 15 − 6.
- The sender sends the five data numbers and the checksum (7, 11, 12, 0, 6, 9)

Checksum - example

- If there is no corruption in transmission, the receiver receives (7, 11, 12, 0, 6, 9) and adds them in one's complement to get 15
- When the receiver adds all five numbers (including the checksum), if it gets a negative zero, the message is accepted; otherwise, it is rejected



Checksum - example



Point-to-Point Protocol

- Today, millions of Internet users who need to connect their home computers to the server of an Internet service provider use most common protocols for point-to-point access Point-to-Point Protocol (PPP)
- The majority of these users have a traditional modem connected to the Internet through a telephone line, which provides the services of the physical layer
- But to control and manage the transfer of data, there is a need for a point-to-point protocol at the data-link layer
- PPP is a byte-oriented protocol

PPP Services

Services Provided by PPP

- Format of the frame to be exchanged between devices
- How two devices can negotiate the establishment of the link and the exchange of data
- Accept payloads from several network layers (not only IP)
- Optional authentication
- The new version of PPP, called *Multilink PPP, provides connections over multiple links*
- Network address configuration
- useful when a home user needs a temporary network address to connect to the Internet

PPP Services

Services Not Provided by PPP

- No flow control - A sender can send several frames one after another with no concern about overwhelming the receiver
- A very simple mechanism for error control - A CRC field is used to detect errors
  - If the frame is corrupted, it is silently discarded; the upper-layer protocol needs to take care of the problem
    - Lack of error control and sequence numbering may cause a packet to be received out of order
- No a sophisticated addressing mechanism to handle frames in a multipoint configuration

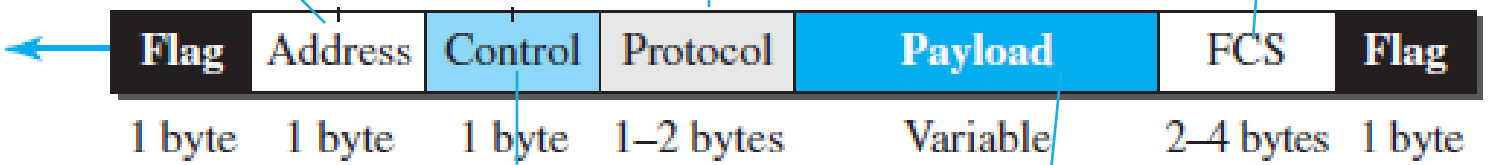
PPP frame format

- a character-oriented (or byte-oriented) frame
- starts and ends with a 1-byte flag with the bit pattern 01111110

a constant value and set to 11111111 (broadcast address)

defines what is being carried in the data field: either user data or other information

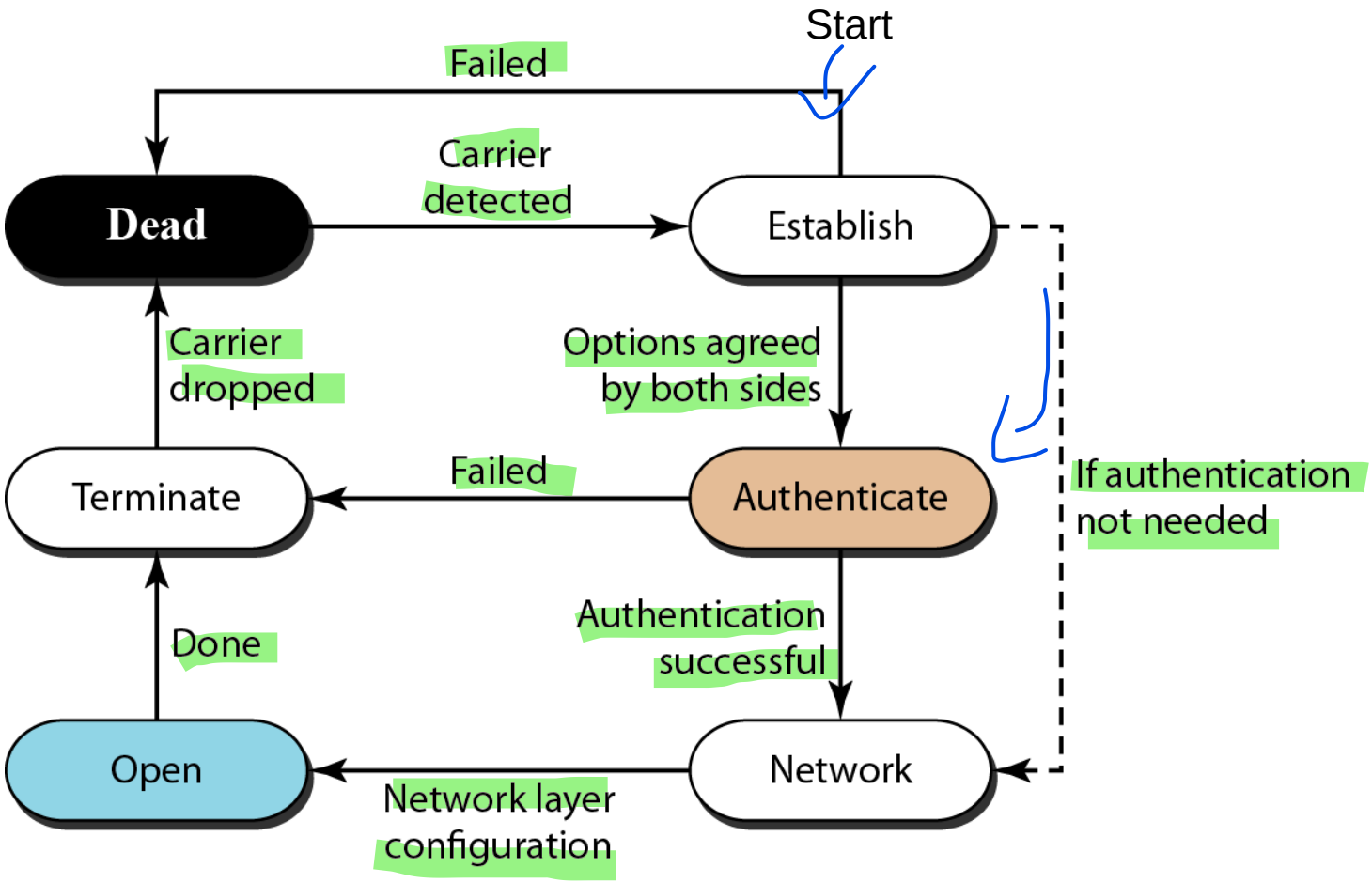
frame check sequence (FCS) is simply a 2-byte or 4-byte standard CRC



constant value 00000011 (imitating unnumbered frames in HDLC)

carries either the user data or other information (byte-stuffed with escape byte as 01111101) - maximum of 1500 bytes; but this can be changed during negotiation

Transition Phases (FSM)



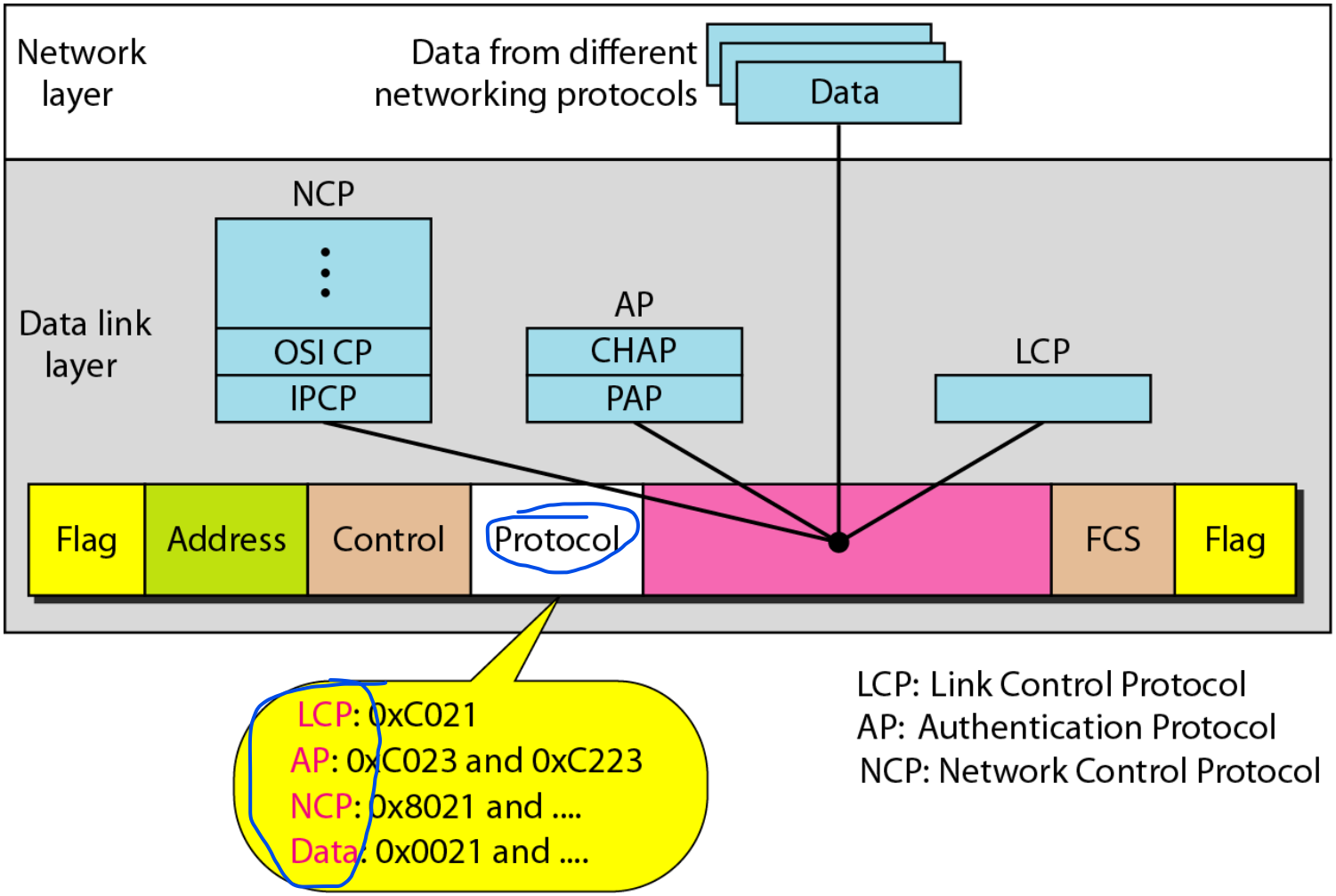
Transition Phases (FSM) …

- Starts with the dead state- no active carrier (at the physical layer) and the line is quiet
- When one of the two nodes starts the communication, the connection goes into the establish state
  - Negotiate options between the two parties through several packets exchange
    - Eg. If the two parties agree that they need authentication then the system needs to do authentication (an extra step); otherwise, the parties can simply start communication.
- Data transfer takes place in the open state
  - The connection remains in this state until one of the endpoints wants to terminate the connection
- System goes to the terminate state when one of the endpoints wants to terminate the connection
- The system remains in this state until the carrier (physical-layer signal) is dropped, which moves the system to the dead state again

Multiplexing in PPP

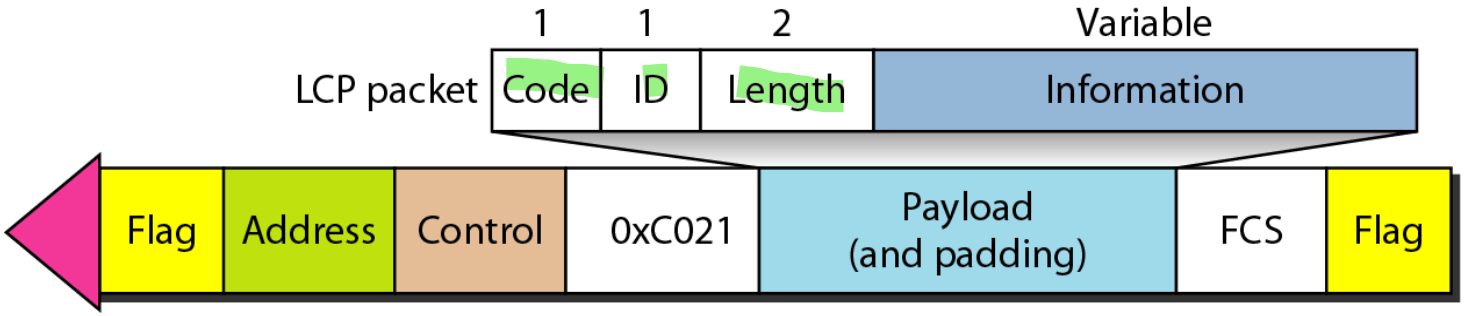
- Powerful PPP at a link-layer uses another set of protocols to
  - establish the link →Link Control Protocol (LCP)
  - authenticate the parties involved → two Authentication Protocols (APs)
  - carry the network-layer data → several Network Control Protocols (NCPs)
- PPP packet can carry data from one of these protocols in its data field
  - Data may also come from several different network layers

Multiplexing in PPP …



Multiplexing in PPP : LCP

- **Responsible for establishing, maintaining, configuring,** and terminating links
- Provides negotiation mechanisms to set options between the two endpoints to reach an agreement about the options before the link can be established
- All LCP packets are carried in the payload field of the PPP frame with the protocol field set to C021 in hexadecimal



Multiplexing in PPP : LCP …

- The code field defines the type of LCP packet → 11 types of packets

used for link configuration during the establish phase

used for link termination during the termination phase

used for link monitoring and debugging

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

Multiplexing in PPP : LCP …

- The ID field holds a value that matches a request with a reply
  - One endpoint inserts a value in this field, which will be copied into the reply packet
- The length field defines the length of the entire LCP packet
- The information field contains information, such as options, needed for some LCP packets
  - There are many options that can be negotiated between the two endpoints
  - Options are inserted in the information field of the configuration packets
    - Information field is divided into three fields: option type, option length, and option data

Multiplexing in PPP : LCP …

- *Common options*

Option	*Default*
Maximum receive unit (payload field size)	1500
Authentication protocol	None
Protocol field compression	Off
Address and control field compression	Off

Multiplexing in PPP : AP

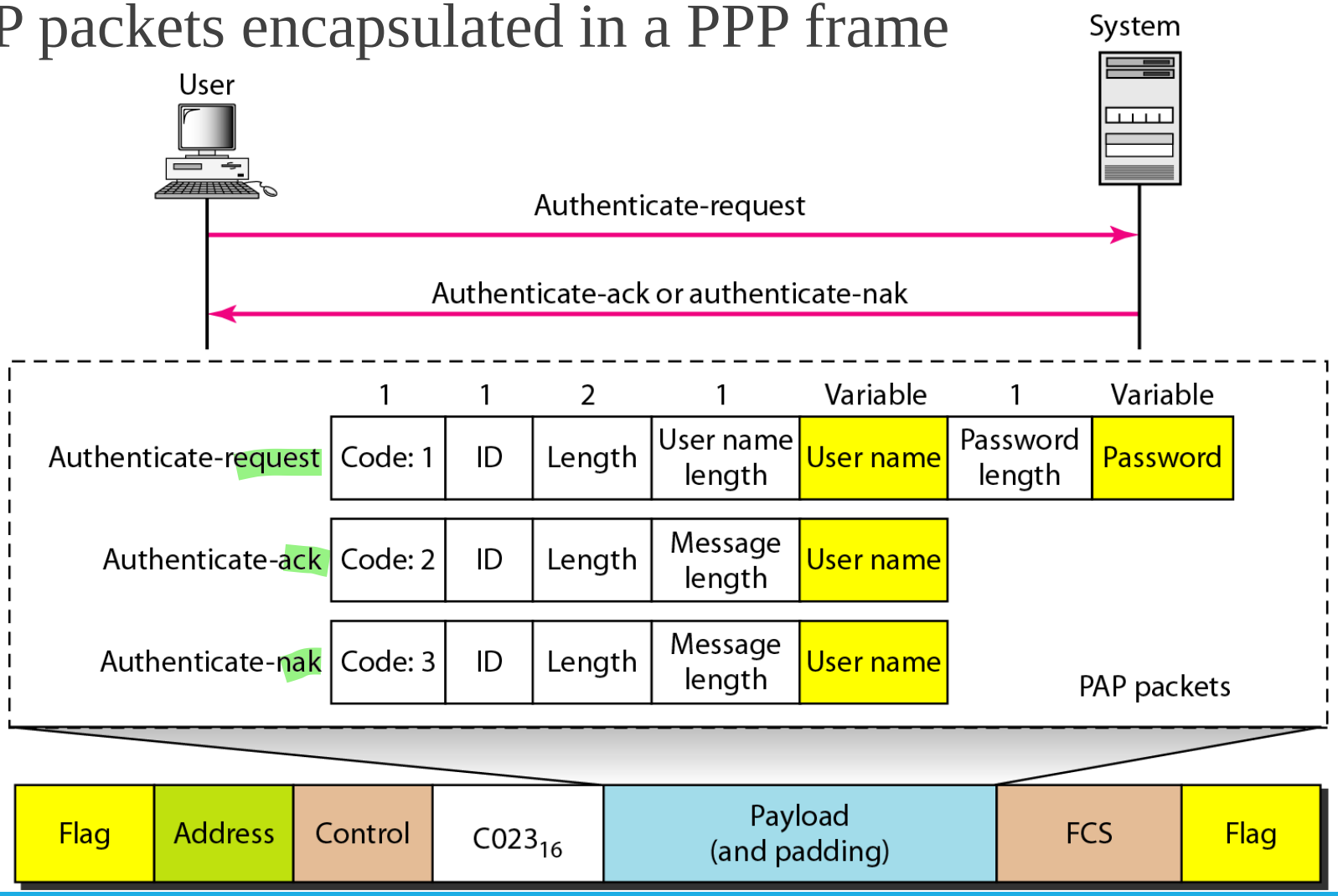
- Authentication plays a very important role in PPP because PPP is designed for use over dial-up links where verification of user identity is necessary
- *Authentication means validating* the identity of a user who needs to access a set of resources
- PPP has created two protocols for authentication used during the authentication phase:
  - Password Authentication Protocol
  - Challenge Handshake Authentication Protocol

Multiplexing in PPP : AP...

- Password Authentication Protocol (PAP) is a simple two-step authentication procedure
  a. The user who wants to access a system sends an authentication identification (usually the user name) and a password.
  b. The system checks the validity of the identification and password and either accepts or denies connection.
- Three types of packets used by PAP
  - authenticate-request - used by the user to send the user name and password
  - authenticate-ack - used by the system to allow access
  - authenticate-nak - used by the system to deny access
- When a PPP frame is carrying any PAP packets, the value of the protocol field is 0xC023

Multiplexing in PPP : AP...

- PAP packets encapsulated in a PPP frame



Multiplexing in PPP : AP...

- **Challenge Handshake Authentication Protocol (CHAP) is a three-way handshaking** authentication protocol
- Password is kept secret; it is never sent online.

a. The system sends the user a challenge packet containing a challenge value, usually a few bytes

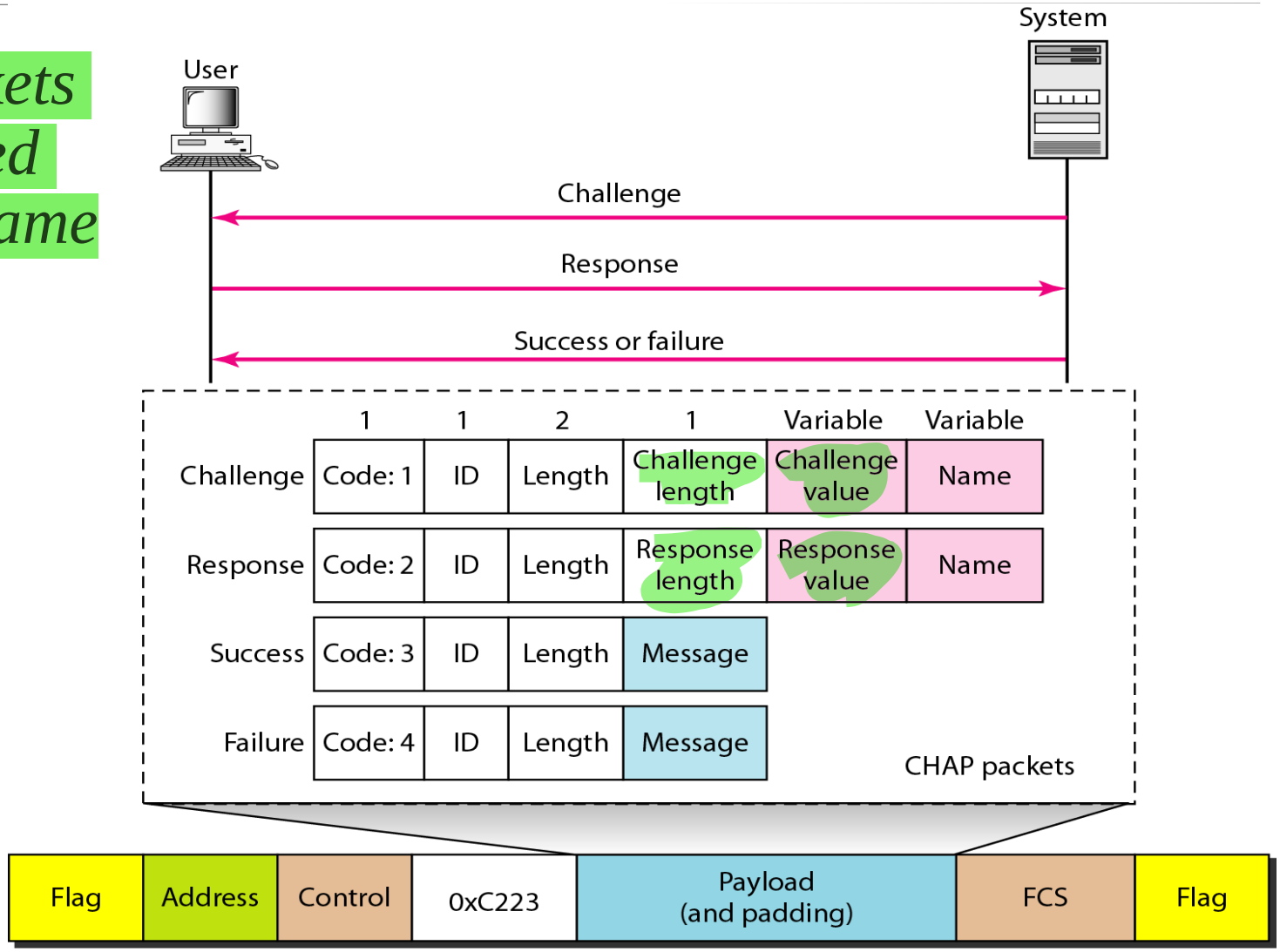
b. The user applies a predefined function that takes the challenge value and the user's own password and creates a result that is send as response packet to the system

c. The system also applies the same function to the password of the user (known to the system) and the challenge value to create a result
  - If the result created is the same as the result sent in the response packet, access is granted; otherwise, it is denied

- CHAP is more secure than PAP, especially if the system continuously changes the challenge value
- Even if the intruder learns the challenge value and the result, the password is still secret

Multiplexing in PPP : AP...

- *CHAP packets encapsulated in a PPP frame*



Multiplexing in PPP : AP…

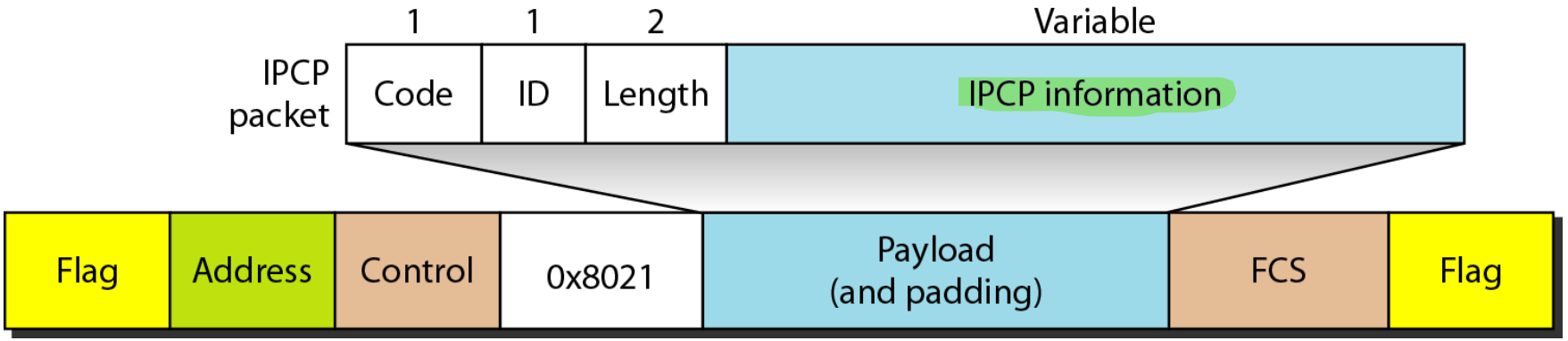
- CHAP packets are encapsulated in the PPP frame with the protocol value C223 in hexadecimal
- There are four CHAP packets:
  - Challenge - used by the system to send the challenge value
  - Response - used by the user to return the result of the calculation
  - Success - used by the system to allow access to the system
  - Failure - used by the system to deny access to the system

Multiplexing in PPP : NCPs

- PPP is a multiple-network-layer protocol
- PPP has defined a specific Network Control Protocol for each network protocol defined by the Internet, OSI, Xerox, DECnet, AppleTalk, Novel, and so on
  - For example, IPCP (Internet Protocol Control Protocol) configures the link for carrying IP data packets
  - Xerox CP does the same for the Xerox protocol data packets
- None of the NCP packets carry network-layer data - just configure the link at the network layer for the incoming data

Multiplexing in PPP : NCPs

- **Internet Protocol Control Protocol (IPCP)** configures the link used to carry IP packets in the Internet
- Value of the protocol field in the format of an IPCP packet is hexadecimal is 8021
  - The OSI Network Layer Control Protocol has a protocol field value of 8023
  - The Xerox NS IDP Control Protocol has a protocol field value of 8025; and so on.



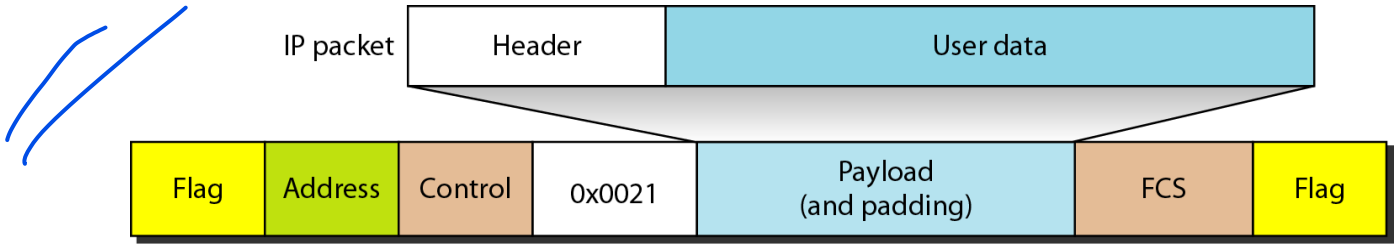
Multiplexing in PPP : NCPs

- IPCP defines seven packets, distinguished by their code values

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

Multiplexing in PPP : NCPs

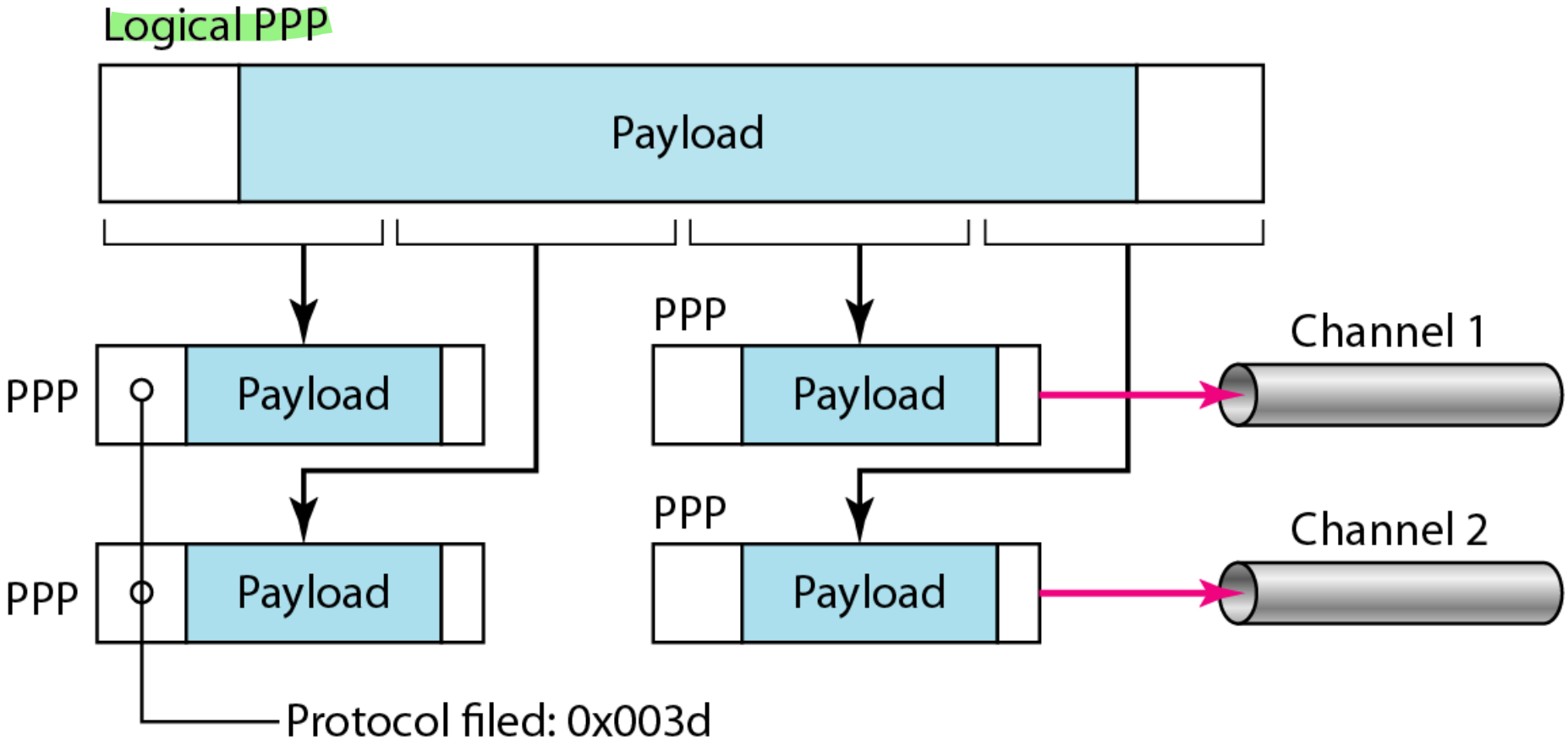
- After the network-layer configuration is completed by one of the NCP protocols, the users can exchange data packets from the network layer
- Different protocol fields for different network layers
  - For example, if PPP is carrying data from the IP network layer, the field value is 0021 (note that the three rightmost digits are the same as for IPCP)
  - If PPP is carrying data from the OSI network layer, the value of the protocol field is 0023
- *IP datagram encapsulated in a PPP frame*



Multilink PPP

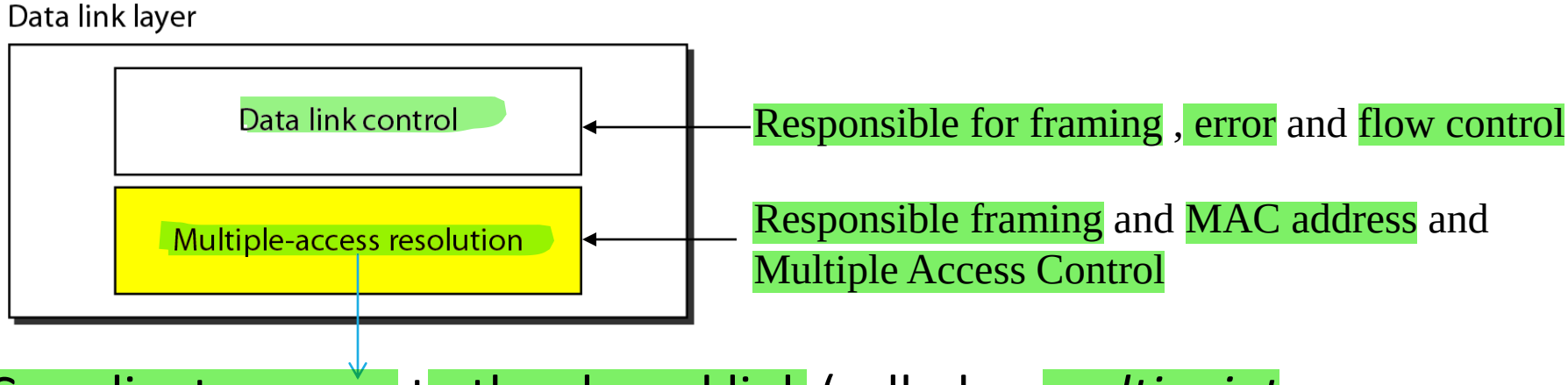
- The availability of multiple channels in a single point-to-point link motivated the development of Multilink PPP
- A logical PPP frame is divided into several actual PPP frames
- A segment of the logical frame is carried in the payload of an actual PPP frame
  - To show that the actual PPP frame is carrying a fragment of a logical PPP frame, the protocol field is set to $(003d)_{16}$
  - This new development adds complexity
    - For example, a sequence number needs to be added to the actual PPP frame to show a fragment's position in the logical frame

Multilink PPP



Data link layer - sublayers

- Data link layer divided into two functionality-oriented sublayers

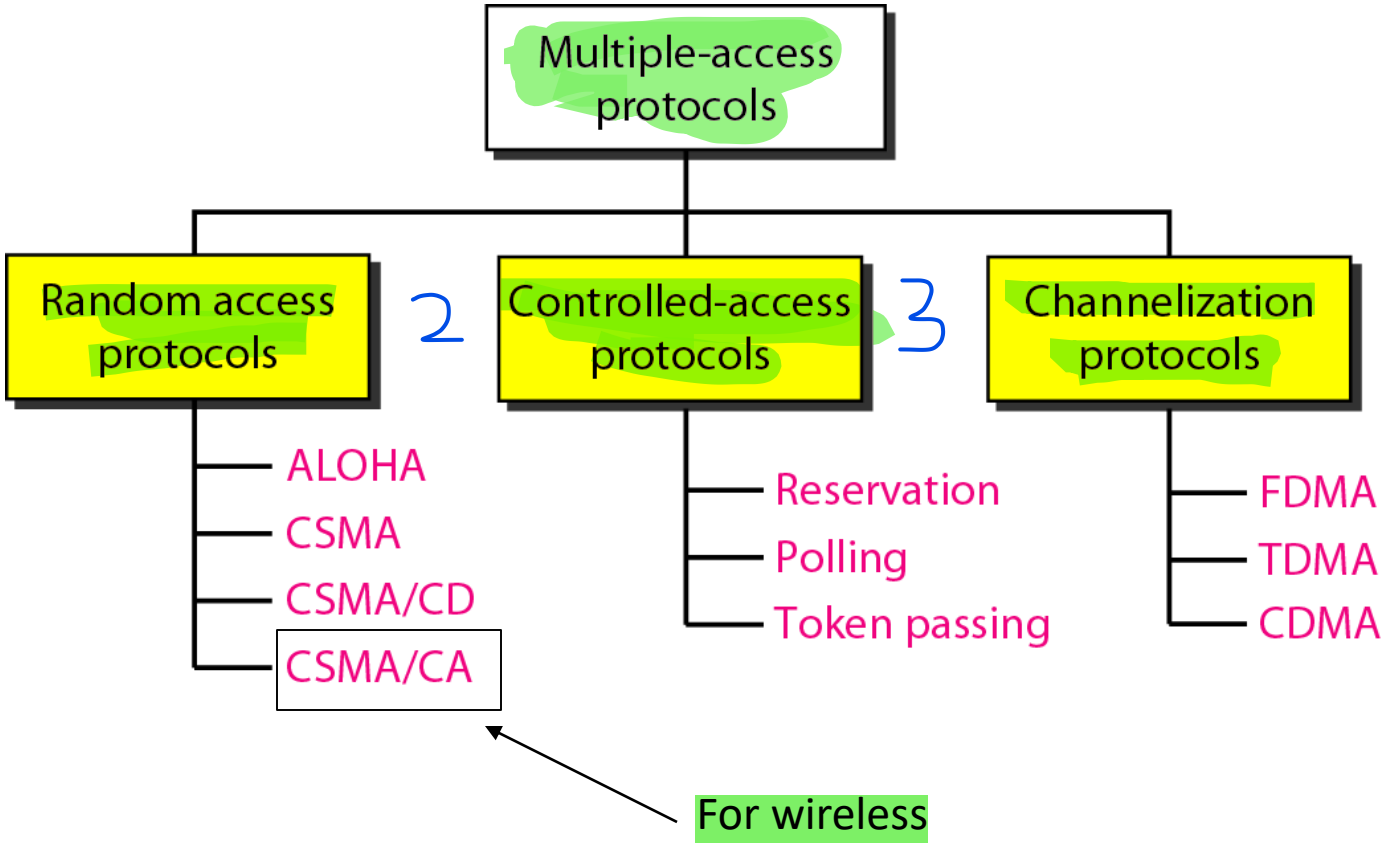


- Coordinate access to the shared link (called as *multipoint or broadcast link*) between nodes
  - **Broadcast link** used in LAN consists of multiple sending and receiving nodes connected to or use a single shared link

Multiple Access Control (MAC)

- **Problem**: When two or more nodes transmit at the same time, their frames will collide and the link bandwidth is **wasted** during collision
- how to determine who gets to use the channel when there is competition for it ?
  - How to coordinate the access of multiple sending/receiving nodes to the shared link**???**
- **Solution**: We need a **protocol** to coordinate the transmission of the active nodes
- These protocols are called **Medium or Multiple Access Control (MAC) Protocols** belong to a **sublayer** of the data link layer called **MAC** (Medium Access Control)
- What is expected from Multiple Access Protocols:
  - Main task is to **minimize collisions** in order to **utilize the bandwidth** by:
    - Determining **when** a station can use the link (medium)
    - **what** a station should do when the link is **busy**
    - **what** the station should do when it is involved in **collision**

Taxonomy of multiple-access protocols



Random Access

- Random Access (or contention) Protocols:
  - No station is superior to another station and none is assigned the control over another
  - A station with a frame to be transmitted can use the link directly based on a procedure defined by the protocol to make a decision on whether or not to send.
- Two features
  - No scheduled time for a station to transmit - Random Transmission
  - No rules specify which station should send next
    - Stations compete with one another to access the medium - contention methods

Evolution of random-access methods

- ALOHA -used a very simple procedure called multiple access (MA)
  - Pure ALOHA - each station sends a frame whenever it has a frame to send
  - Slotted ALOHA - divide the time into slots and force the station to send only at the beginning of the time slot
- Carrier sense multiple access (CSMA) - improved ALOHA
  - forces the station to sense the medium before transmitting
  - Two parallel evolved methods
    - carrier sense multiple access with collision detection (CSMA/CD)
      - tells the station what to do when a collision is detected
    - carrier sense multiple access with collision avoidance (CSMA/CA)
      - tries to avoid the collision

Carrier sense multiple access (CSMA)

- Minimize the chance of collision cannot eliminate it
  - increase the performance
- How to minimize chance of collision?
  - A station should sense the medium before trying to use it (space and time model)
    - Each station first listen to the medium (or check the state of the medium) before sending
      - Principle “sense before transmit” or “listen before talk”

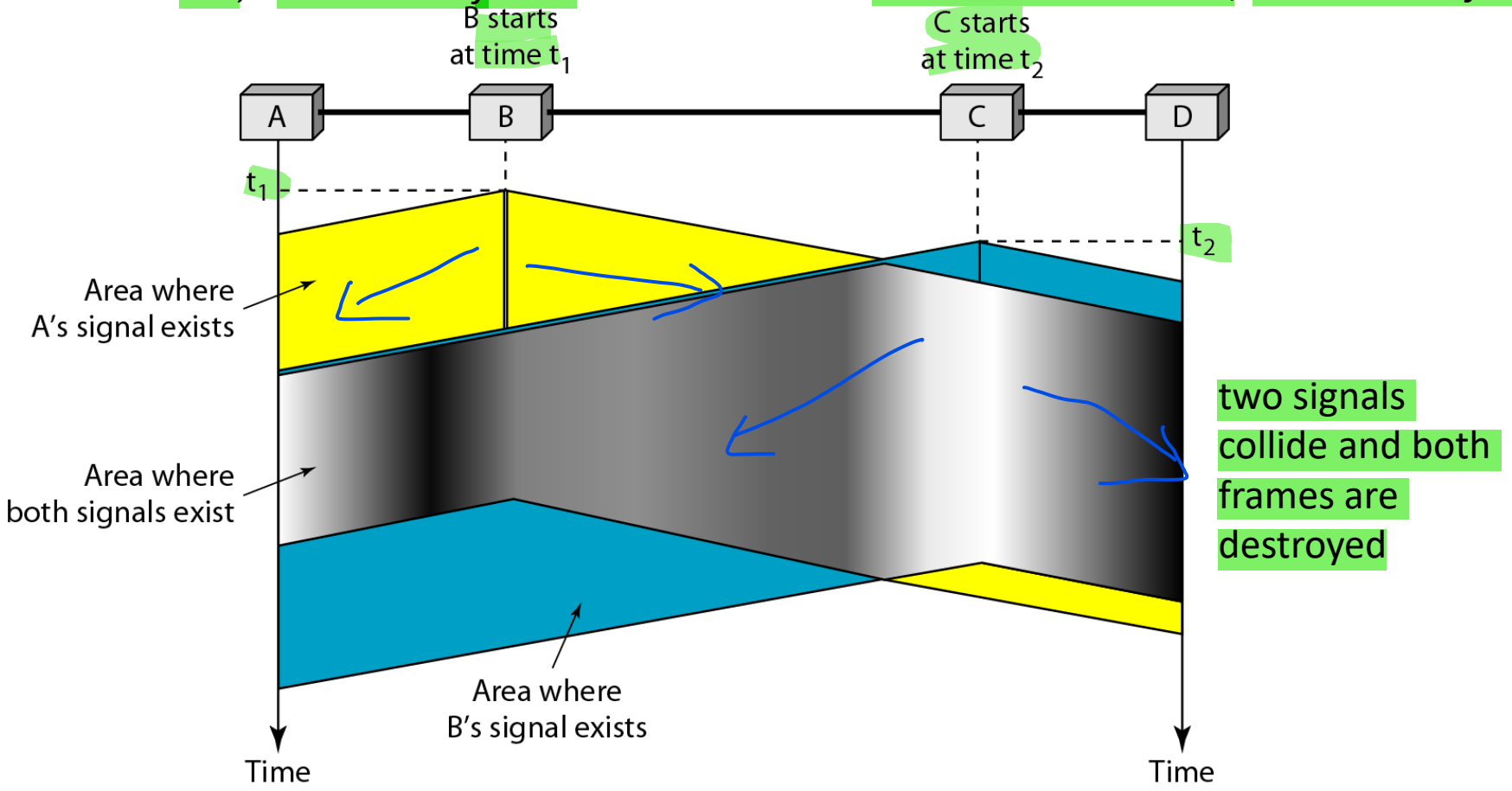
CSMA - space and time model

- Stations are connected to a shared channel
- The possibility of collision still exists because of propagation delay
  - when a station sends a frame, it still takes time (although very short) for the first bit to reach every station and for every station to sense it
    - a station may sense the medium and find it idle, only because the first bit sent by another station has not yet been received

CSMA - space and time model

station B senses the medium and finds it idle, so it sends a frame

station C senses the medium and finds it idle as the first bits from station B have not reached station C, *so it sends a frame*



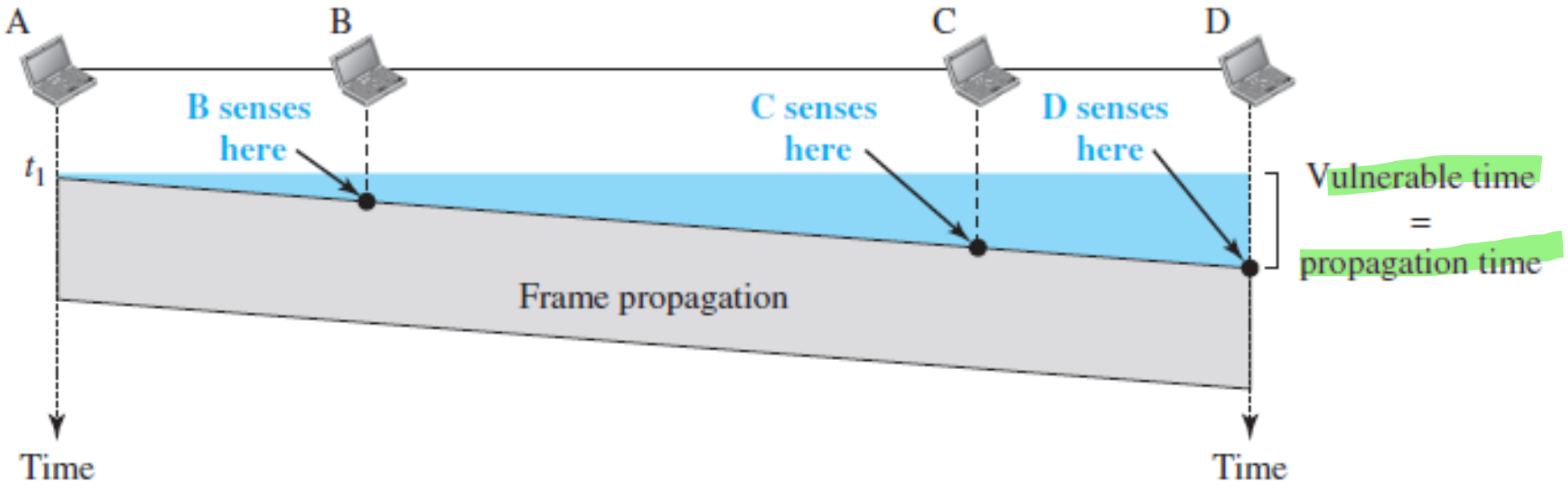
two signals collide and both frames are destroyed

CSMA - vulnerable time

- *Vulnerable time* = ***propagation time*** T_p
  - T_p is time needed for a signal to propagate from one end of the medium to the other
  - When a station sends a frame and any other station tries to send a frame during this time, a collision will result
    - If the first bit of the frame reaches the end of the medium, every station will already have heard the bit and will refrain from sending

CSMA - vulnerable time

- leftmost station, A, sends a frame at time t1, which reaches the rightmost station, D, at time t1 + T_p
- The gray area shows the vulnerable area in time and space.

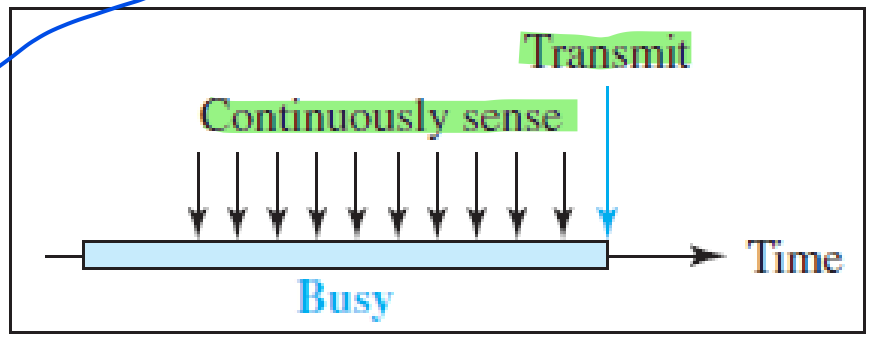


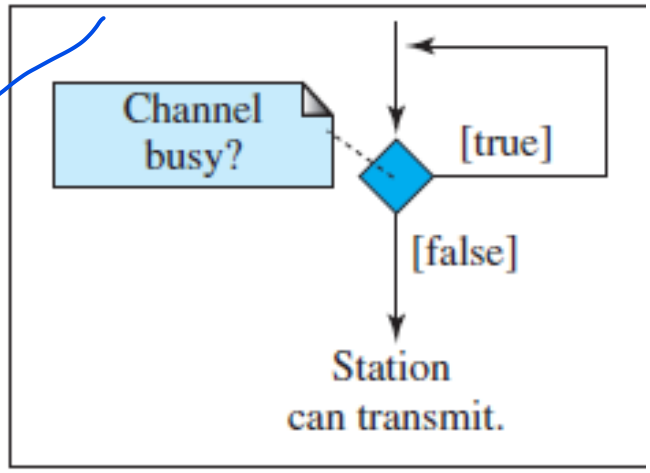
CSMA - Persistence Methods

- What should a station do if the channel is busy?
- What should a station do if the channel is idle?
- Three methods have been devised to answer these questions:
  - 1-persistent method
  - nonpersistent method
  - *p-persistent method*

CSMA - *1-Persistent*

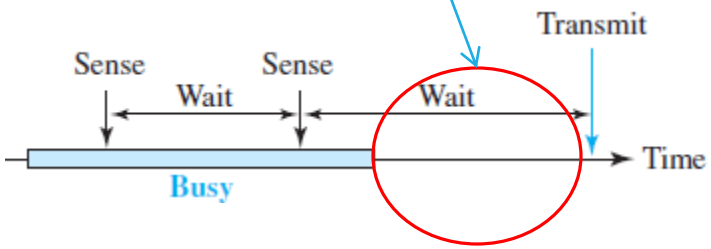
- *Simple and straightforward*
- *After the station* finds the line idle, it sends its frame immediately (with probability 1)
  - highest chance of collision because two or more stations may find the line idle and send their frames immediately
- Figure shows behavior when a station finds a channel busy

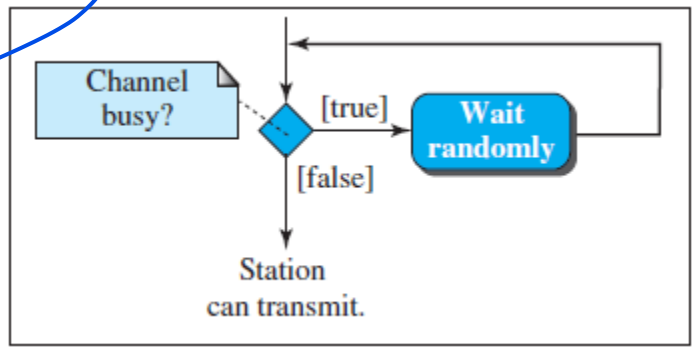




CSMA - Nonpersistent

- If a station that has a frame to send senses the line and the line is idle, it sends immediately
- If the line is not idle, it waits a random amount of time and then senses the line again
- Reduced the chance of collision because it is unlikely that two or more stations will wait the same amount of time and retry to send simultaneously
- Reduces the efficiency of the network because the medium remains idle when there may be stations with frames to send





CSMA - p-*Persistent*

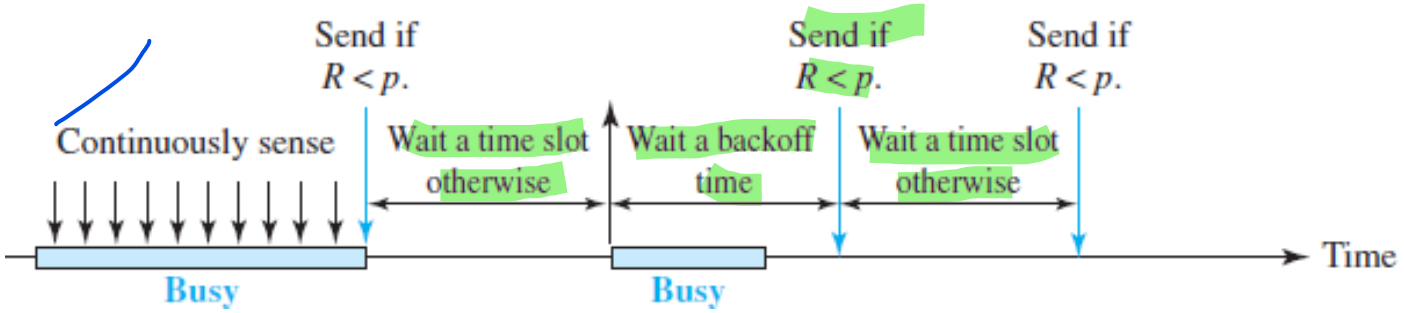
- Used if the channel has time slots with a slot duration equal to or greater than the maximum propagation time
- Combines the advantages of 1-persistent method and nonpersistent method
- Reduces the chance of collision and improves efficiency
- After the station finds the line idle it follows these steps:

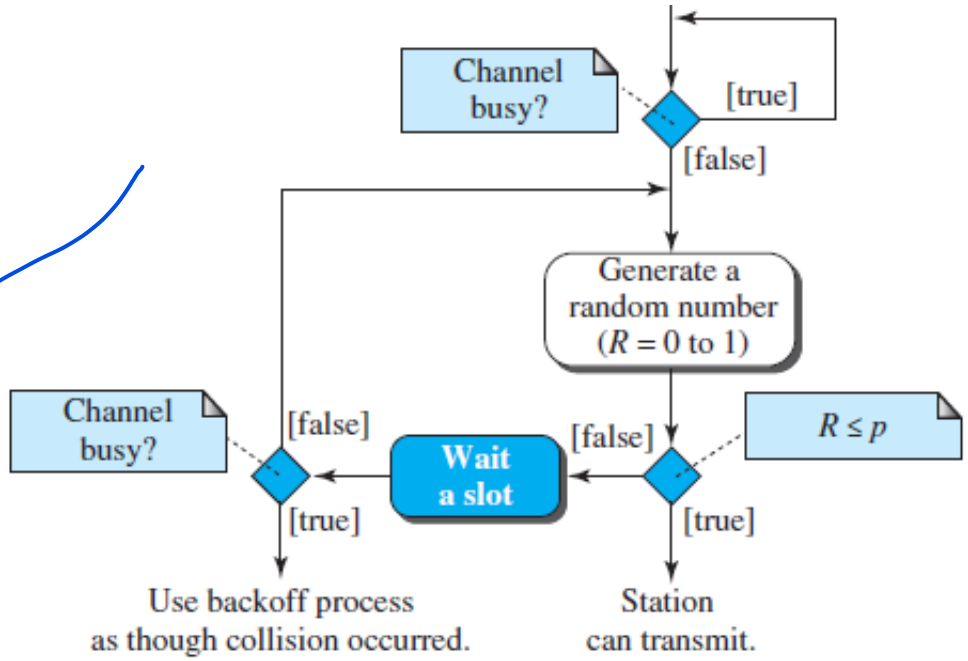
1. With probability p, the station sends its frame
2. With probability q = 1 − p, the station waits for the beginning of the next time slot and checks the line again

   a. If the line is idle, it goes to step 1.

   b. If the line is busy, it acts as though a collision has occurred and uses the backoff procedure

CSMA - p-*Persistent*



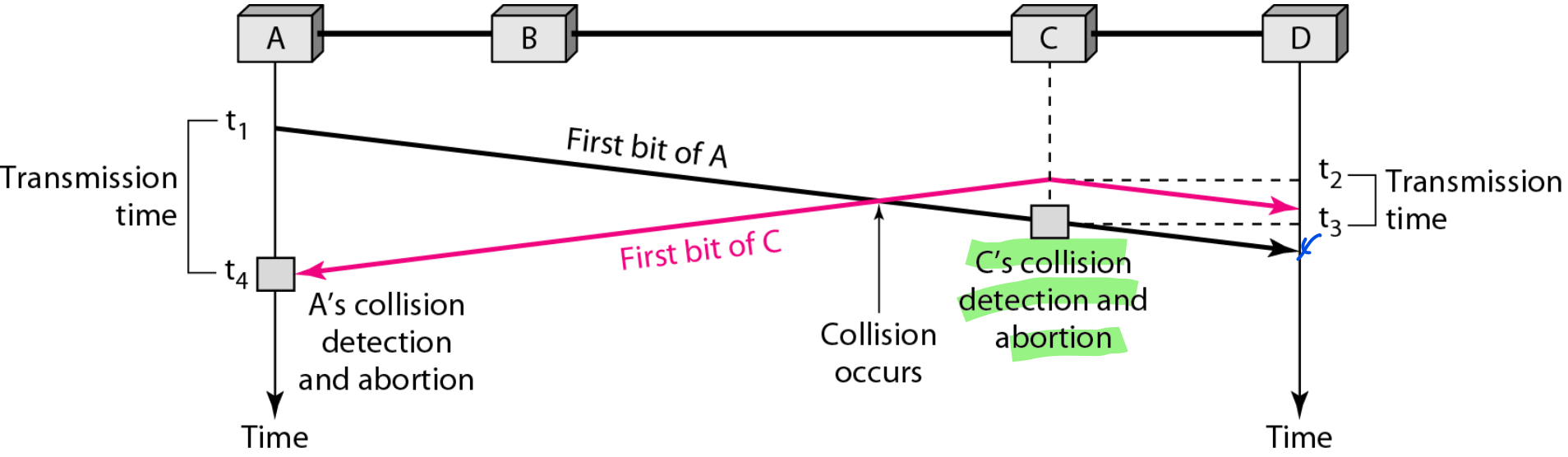


Carrier sense multiple access with collision detection (CSMA/CD)

- The CSMA method does not specify the procedure following a collision
- CSMA/CD augments the algorithm to handle the collision
  - A station monitors the medium after it sends a frame to see if the transmission was successful
  - If so, the station is finished
  - If there is a collision, the frame is sent again

CSMA/CD ...

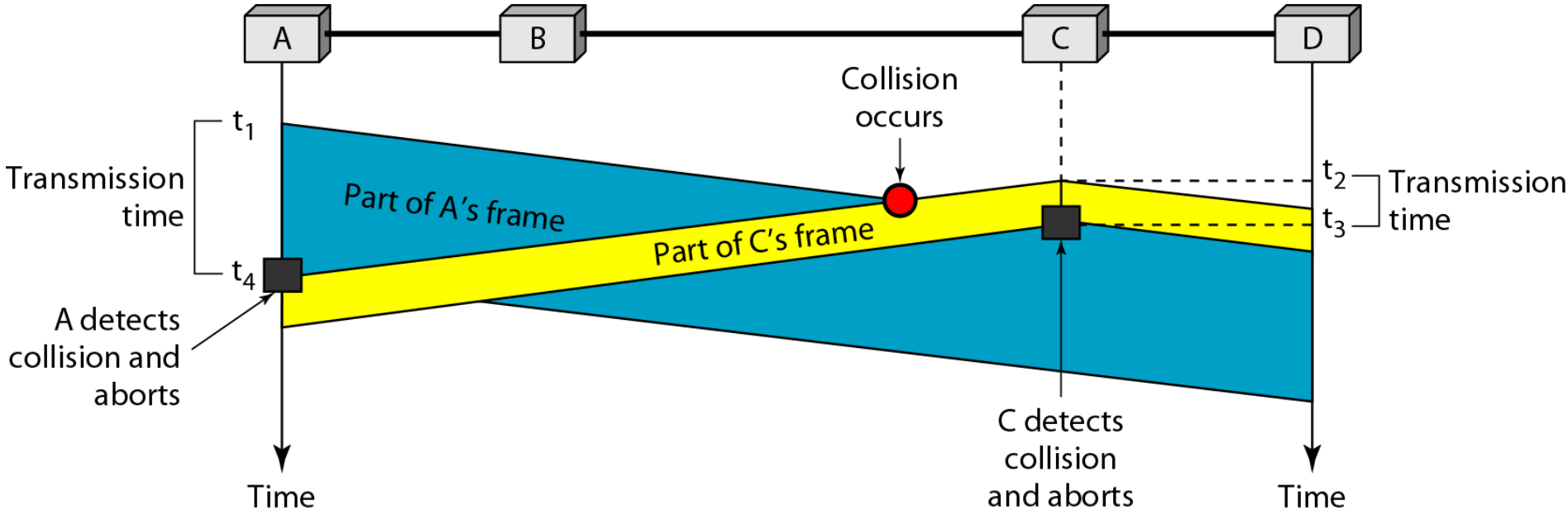
- Look at the first bits transmitted by the two stations involved in the collision
  - each station continues to send bits in the frame until it detects the collision



CSMA/CD …

- At time t1, station A has executed its persistence procedure and starts sending the bits of its frame
- At time t2, station C has not yet sensed the first bit sent by A
- Station C executes its persistence procedure and starts sending the bits in its frame, which propagate both to the left and to the right
- The collision occurs sometime after time t2
- Station C detects a collision at time t3 when it receives the first bit of A's frame
- Station C immediately (or after a short time, but we assume immediately) aborts transmission
- Station A detects collision at time t4 when it receives the first bit of C's frame - immediately aborts transmission
- A transmits for the duration t4 – t1; C transmits for the duration t3 – t2

CSMA/CD - Collision and abortion



CSMA/CD -*Minimum Frame Size*

- Need a restriction on the frame size
- Before sending the last bit of the frame, the sending station must detect a collision, if any, and abort the transmission
  - because the station, once the entire frame is sent, does not keep a copy of the frame and does not monitor the line for collision detection
  - The frame transmission time T_{fr} *must be at least two times the maximum propagation time* T_p
  - Worst-case scenario. If the two stations involved in a collision are the maximum distance apart, the signal from the first takes time T_p *to reach* the second, and the effect of the collision takes another time T_P *to reach the first*
  - *So the* requirement is that the first station must still be transmitting after $2T_p$

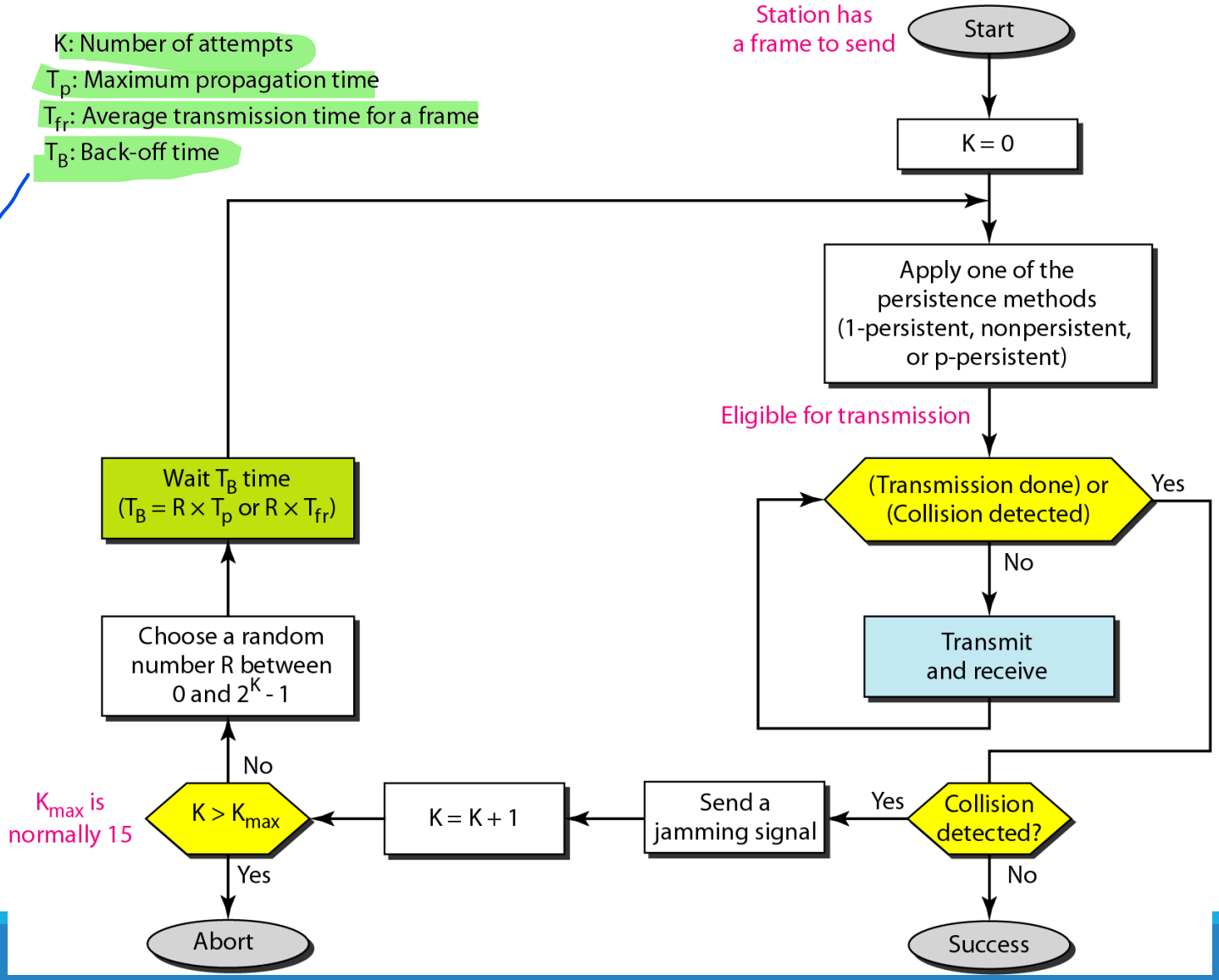
Tfr = 2*Tp

Mini Frame Size = Bandwidth (Mbps) * Tfr

CSMA/CD Example

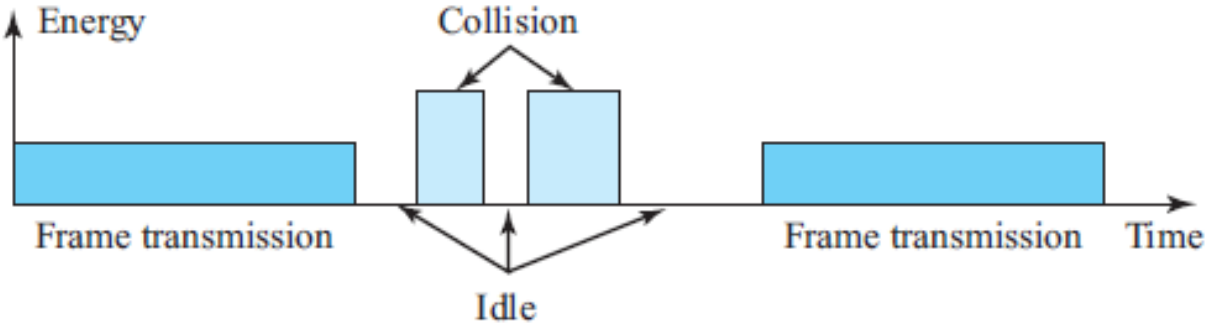
- A network using CSMA/CD has a bandwidth of 10 Mbps
- If the maximum propagation time (including the delays in the devices and ignoring the time needed to send a jamming signal is 25.6 µs, what is the minimum size of the frame?
- **Solution**
  - The minimum frame transmission time is $T_{fr} = 2 \times Tp = 51.2\ \mu s$
  - *The worst case, a* station needs to transmit for a period of 51.2 µs to detect the collision
  - The minimum size of the frame is 10 Mbps × 51.2 µs = 512 bits or 64 bytes
    - This is actually the minimum size of the frame for Standard Ethernet

Flow diagram for CSMA/CD



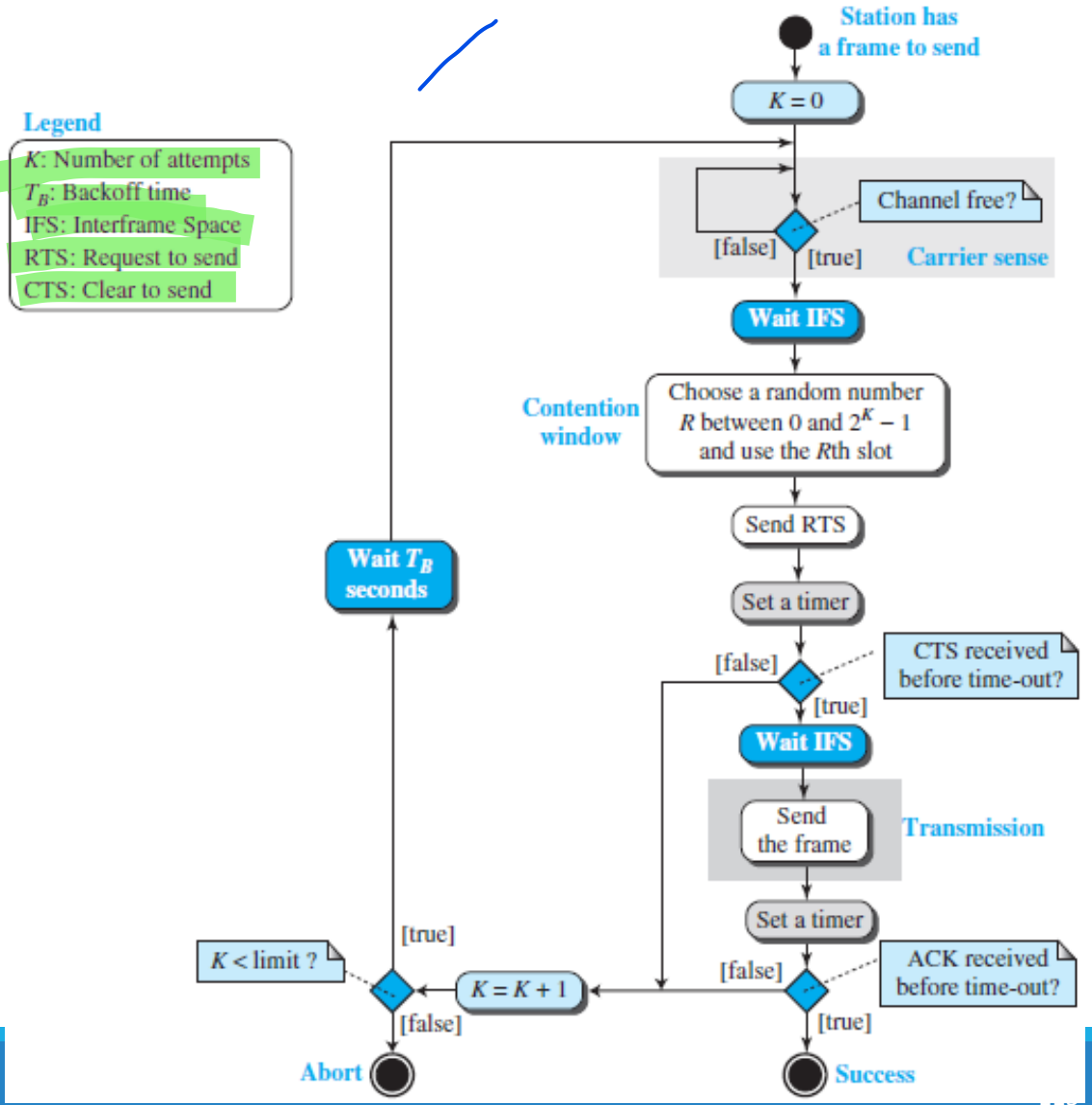
CSMA/CD - *Energy Level*

- The level of energy in a channel can have three values:
  - Zero - the channel is idle
  - Normal - station has successfully captured the channel and is sending its frame
  - Abnormal –there is a collision and the level of the energy is twice the normal level
- A station that has a frame to send or is sending a frame needs to monitor the energy level to determine if the channel is idle, busy, or in collision mode



Carrier sense multiple access with collision avoidance - CSMA/CA

- **Invented** for wireless networks
- Collisions are avoided through the use of CSMA/CA's three strategies:
  - the interframe space
  - the contention window
  - acknowledgments



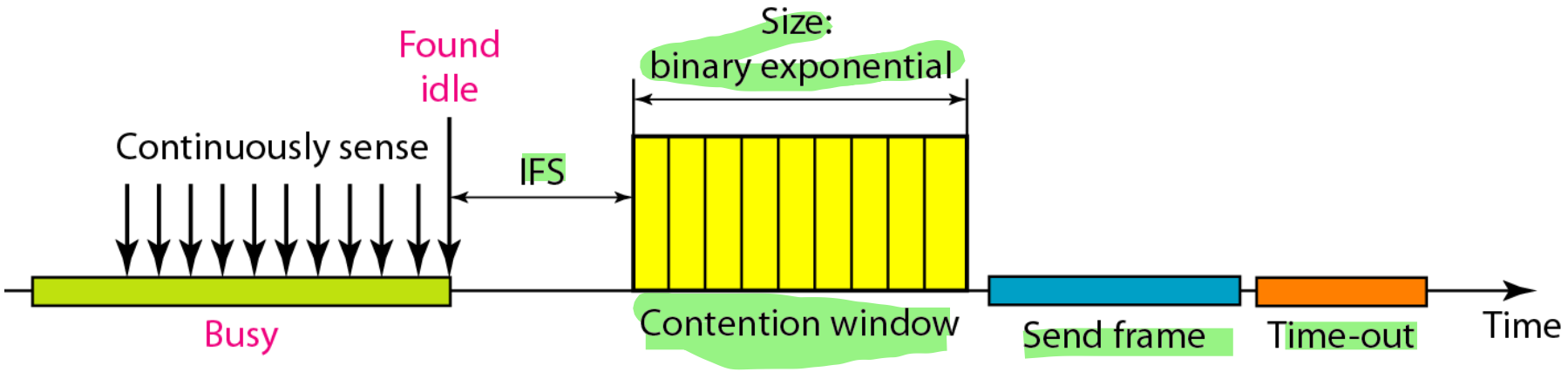
CSMA/CA- *Interframe Space (IFS)*

- *Avoids collisions by deferring transmission even* if the channel is found idle
- When an idle channel is found, the station waits for a period of *IFS* time
- IFS time allows the front of the transmitted signal by the distant station to reach this station
- After waiting an IFS time, if the channel is still idle, the station can send, but it still needs to wait a time equal to the contention window
- The IFS variable can also be used to prioritize stations or frame types
  - For example, a station that is assigned a shorter IFS has a higher priority

CSMA/CA - *Contention Window*

- ***An amount of time divided into*** slots
- A station that is ready to send chooses a random number of slots as its wait time
- The number of slots in the window changes according to the binary exponential backoff strategy
  - It is set to one slot the first time and then doubles each time the station cannot detect an idle channel after the IFS time
  - Very similar to the *p-persistent method except that a random outcome defines the* number of slots taken by the waiting station
- The station needs to sense the channel after each time slot
- However, if the station finds the channel busy, it does not restart the process; it just stops the timer and restarts it when the channel is sensed as idle
- This gives priority to the station with the longest waiting time

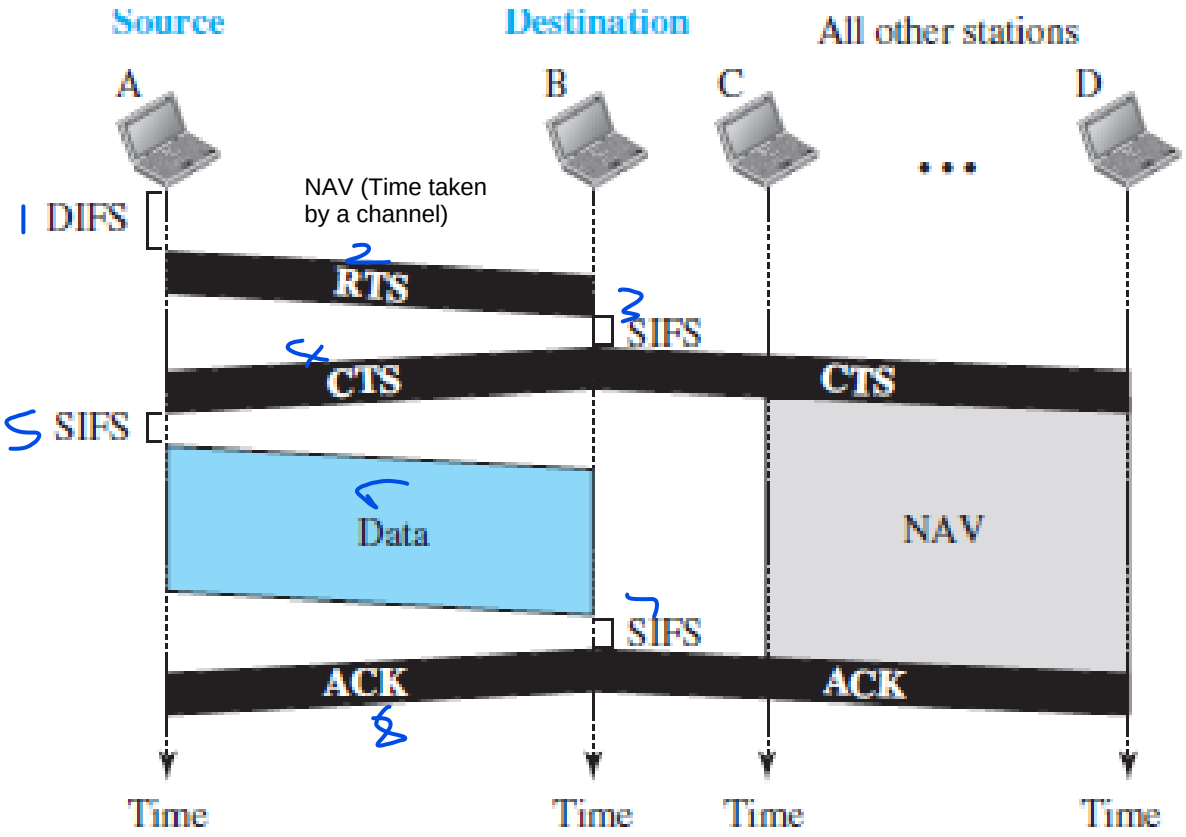
CSMA/CA - Timing



CSMA/CA - *Acknowledgment*

- With all these precautions, there still may be a collision resulting in destroyed data
- In addition, the data may be corrupted during the transmission
- The positive acknowledgment and the time-out timer can help guarantee that the receiver has received the frame

CSMA/CA -*Frame Exchange Time Line*



CSMA/CA -*Frame Exchange Time Line*

- Exchange of data and control frames in time.
  1. Before sending a frame, the source station senses the medium by checking the energy level at the carrier frequency
     a. The channel uses a persistence strategy with backoff until the channel is idle
     b. After the station is found to be idle, the station waits for a period of time called the *DCF interframe space (DIFS); then the station sends a control frame called* the *request to send (RTS).*
  2. After receiving the RTS and waiting a period of time called the *short interframe space (SIFS), the destination station sends a control frame, called the clear to send (CTS), to the source station. This control frame indicates that the destination* station is ready to receive data.
  3. The source station sends data after waiting an amount of time equal to SIFS.
  4. The destination station, after waiting an amount of time equal to SIFS, sends an acknowledgment to show that the frame has been received. Acknowledgment is needed in this protocol because the station does not have any means to check for the successful arrival of its data at the destination. On the other hand, the lack of collision is a kind of indication to the source that data have arrived.

CSMA/CA - *Network Allocation Vector*

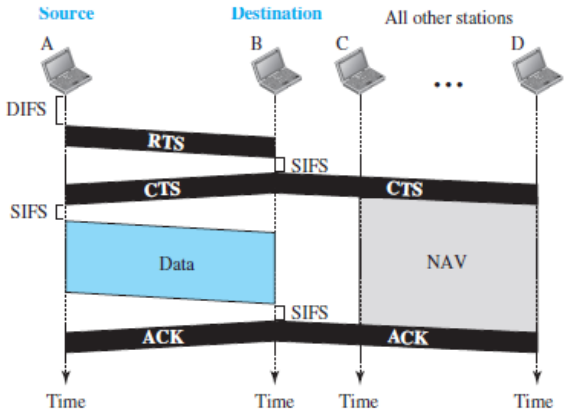
- How do other stations defer sending their data if one station acquires access?
- how is the *collision avoidance aspect of this protocol accomplished?*
- *Use key* feature *Network Allocation Vector (NAV):*
  - When a station sends an RTS frame, it includes the duration of time that it needs to occupy the channel
  - The stations that are affected by this transmission create a timer called a **network allocation vector (NAV) that shows how much time must pass before** these stations are allowed to check the channel for idleness
  - Each time a station accesses the system and sends an RTS frame, other stations start their NAV

CSMA/CA - *Collision During Handshaking*

- *handshaking period* - time when RTS or CTS control frames are in transition
- What happens if there is a collision during *handshaking period ?*
- *Indicates two or more stations may try to* send RTS frames at the same time which may collides
- No mechanism for collision detection, the sender assumes there has been a collision if it has not received a CTS frame from the receiver
- The backoff strategy is employed, and the sender tries again

CSMA/CA - Hidden-Station Problem

- Use of the handshake frames (RTS and CTS)
- The RTS message from B reaches A, but not C.
- However, because both B and C are within the range of A, the CTS message, which contains the duration of data transmission from B to A, reaches C
- Station C knows that some hidden station is using the channel and refrains from transmitting until that duration is over



Controlled access

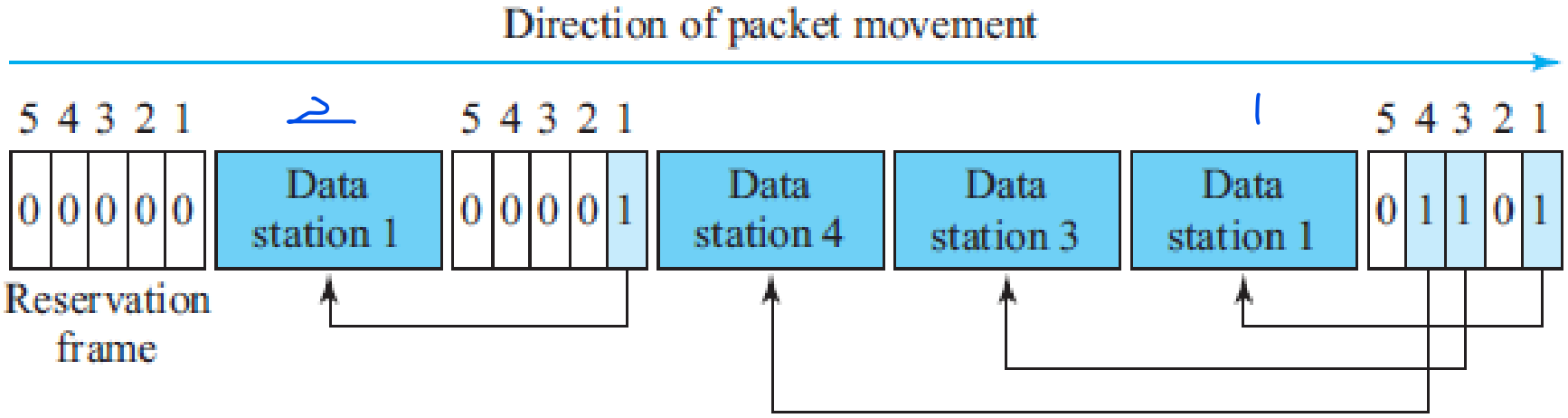
- **The stations consult one another to find which station has the right** to send
- A station cannot send unless it has been authorized by other stations
- Three controlled-access methods
  - **Reservation method**
  - **Polling**
  - **Token Passing**

Reservation method

- **A station needs to make a reservation before sending data**
- Time is divided into intervals
- A reservation frame precedes the data frames sent in each interval.
- If there are *N stations in the system, there are exactly N reservation minislots in the* reservation frame
- Each minislot belongs to a station - When a station needs to send a data frame, it makes a reservation in its own minislot
- The stations that have made reservations can send their data frames after the reservation frame

Reservation access method

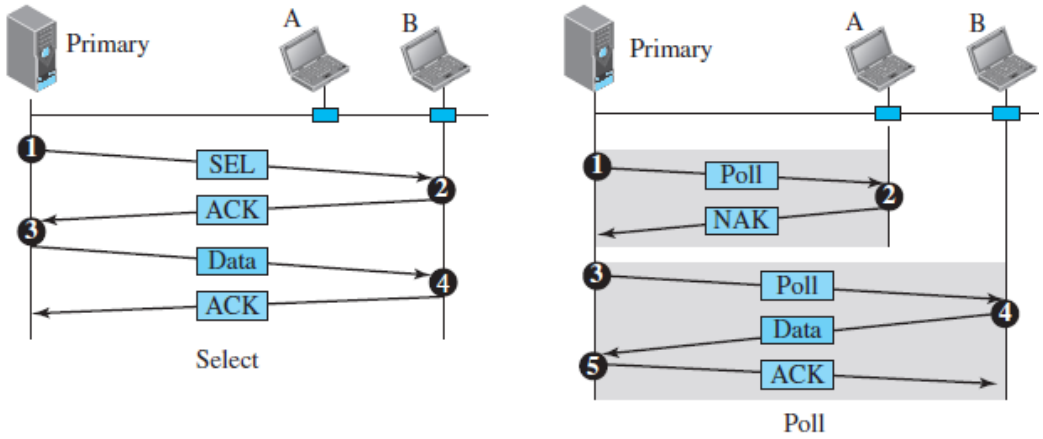
- Five stations and a five-minislot reservation frame
- In the first interval, only stations 1, 3, and 4 have made reservations
- In the second interval, only station 1 has made a reservation



Polling

- Works with topologies in which one device is designated as a primary station that controls the link and the other devices are secondary stations that follow its instructions
- All data exchanges must be made through the primary device even when the ultimate destination is a secondary device
- Primary device - determine which device is allowed to use the channel at a given time and always the initiator of a session
- uses poll and select functions to prevent collisions
- Drawback - if the primary station fails, the system goes down

Polling …



- *used whenever the primary device has something to send.*
- *Primary* does not know whether the target device is prepared to receive
- Primary alert the secondary about the upcoming transmission and wait for an acknowledgment of the secondary's ready status using select (SEL) frame
- One field of SEL includes the address of the intended secondary

- *Used by the primary device to solicit transmissions from the secondary* devices
- When the primary is ready to receive data, it must ask (poll) each device in turn if it has anything to send
- When the first secondary is approached, it responds either with a AK frame if it has nothing to send or with data if it does
- If the response is negative (a NAK frame), then the primary polls the next secondary in the same manner until it finds one with data to send
- When the response is positive (a data frame), the primary reads the frame and returns an acknowledgment (ACK frame), verifying its receipt

Token Passing

- **The stations in a network are organized in a logical ring where** each station has a *predecessor and a successor*
- *The predecessor* is the station which is logically before the station in the ring; the successor is the station which is after the station in the ring
- The current station is the one that is accessing the channel now
- The right to this access has been passed from the predecessor to the current station then to the successor when the current station has no more data to send

Token Passing

how is the right to access the channel passed from one station to another?

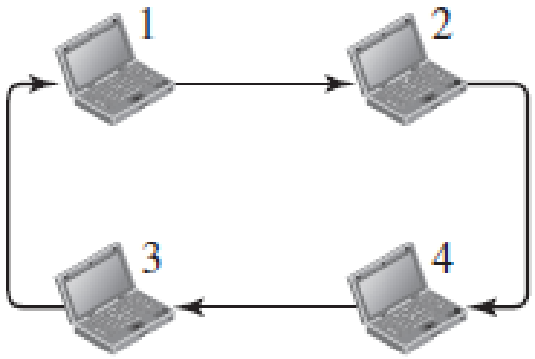
- A special packet called a ***token circulates through the ring***
- ***The possession*** of the token gives the station the right to access the channel and send its data
- When a station has some data to send, it waits until it receives the token from its predecessor
- It then holds the token and sends its data
- When the station has no more data to send, it releases the token, passing it to the next logical station in the ring
- The station cannot send data until it receives the token again in the next round
- When a station receives the token and has no data to send, it just passes the data to the next station

Token management

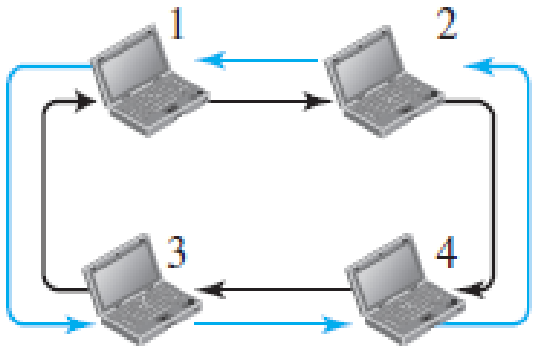
- Stations must be limited
- The token must be monitored to ensure it has not been lost or destroyed
  - For example, if a station that is holding the token fails, the token will disappear from the network
- Assign priorities to the stations and to the types of data being transmitted
- make low-priority stations release the token to high-priority stations

Logical Ring

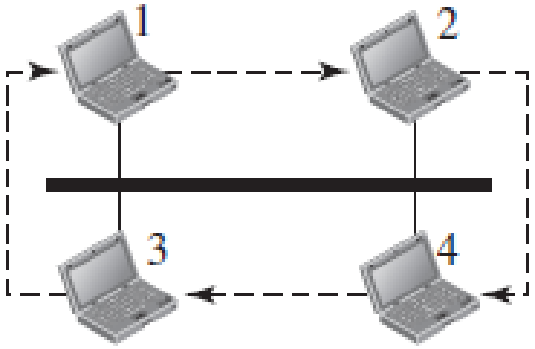
- Four different physical topologies that can create a logical ring



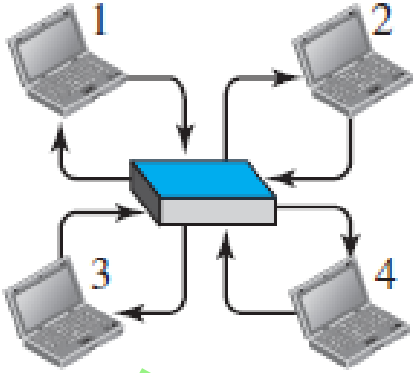
a. Physical ring



b. Dual ring



c. Bus ring



d. Star ring