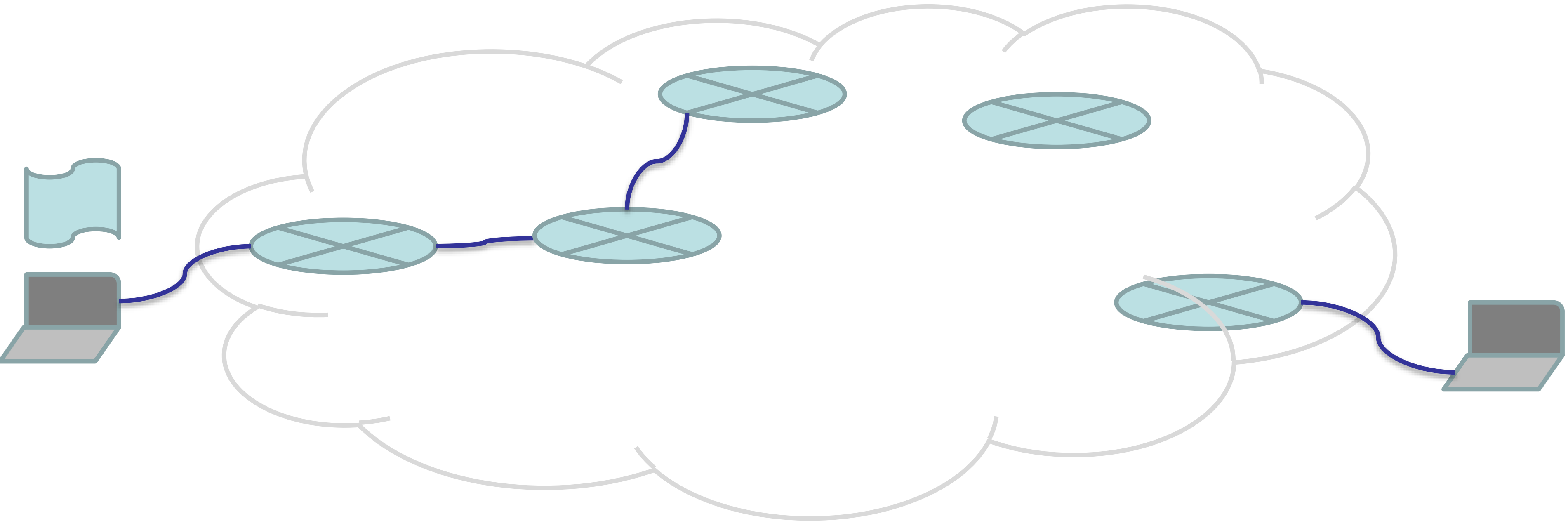
COSC264
Introduction to Computer Networks and the Internet

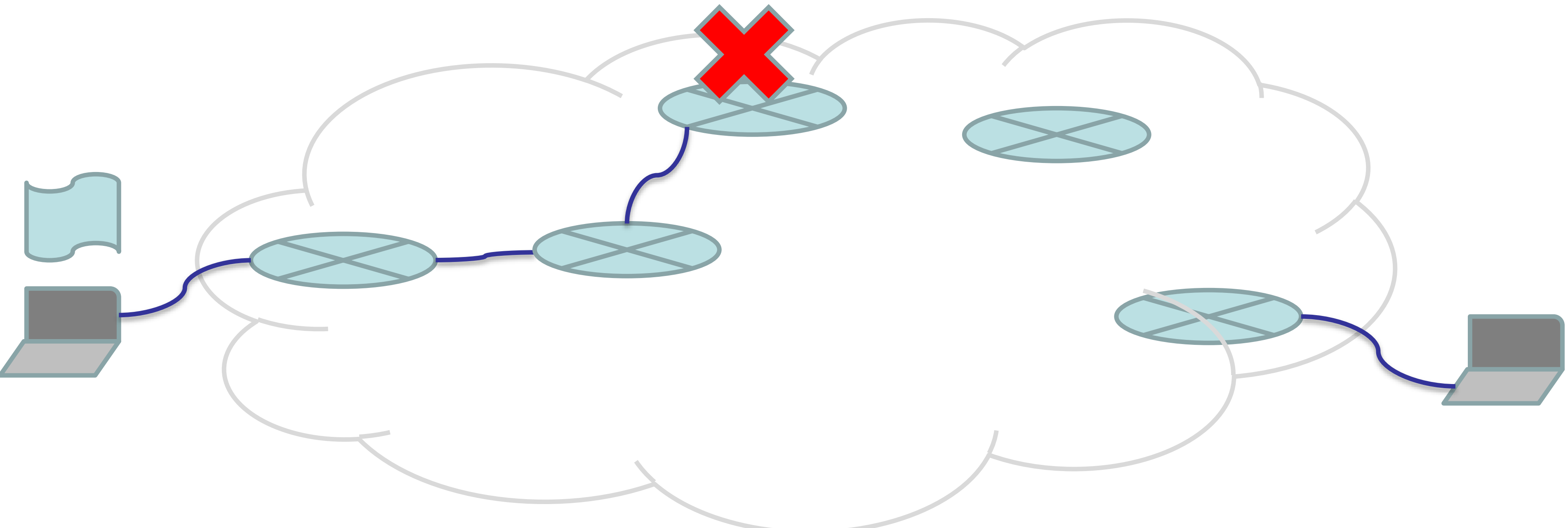
Reliable data transfer: Error Detection and Correction

Dr. Barry Wu
Wireless Research Centre
University of Canterbury
barry.wu@canterbury.ac.nz

The journey of a packet



The journey of a packet



Problems	Solutions
Bit error	Error detection and correction
Buffer overflow	Flow control and congestion control
Lost packet	Acknowledgement and retransmission (ARQ)
Out of order	Acknowledgement and retransmission (ARQ)

Outline

- Error Detection
  - Parity check
    - Parity bit / 2-D Parity check
  - Internet checksum
  - Cyclic Redundancy Check (CRC)
- Forward Error Correction
  - Block Code Principles
- Summary

Error Detection

- With error-detection coding, redundancy is added to a packet so that:
  - Certain error patterns can be detected reliably
  - Other error patterns can be detected with high probability
  - No information about the position of errors in a packet can be inferred (and hence *no correction can be done*)

Outline

- **Error Detection**
  - **Parity check**
    - Parity bit / 2-D Parity check
  - Internet checksum
  - Cyclic Redundancy Check (CRC)
- Forward Error Correction
  - Block Code Principles
- Summary

Parity Check

- The simplest error **detecting** scheme is to append a parity bit to the end of a block of data
  - Even parity

Parity bit

10000101 1

10010101 1

  - Odd parity

Parity bit

10010101 1

10000101 1

Parity check

- If any even number of bits are inverted due to error, an undetected error occurs

Parity bit

10000101 1

10010001 1

Parity Check (2)

- **How?** data stream is subdivided into small blocks, e.g. bytes
- Even parity: one additional bit is appended to a byte so that total # of 1’s in data and parity bits is even, e.g.:
  - 0100 1101 becomes 0100 1101 0
  - 0101 1101 becomes 0101 1101 1
- Odd parity: similar, but total number of 1’s is odd
- Properties of parity check codes:
  - All odd numbers of bit errors are reliably detected
  - All even numbers of bit errors are reliably not detected
- Parity check codes have traditionally been used on serial comms. interfaces,
  - e.g. RS-232 is a standard for serial communication transmission of data
  - where a parity bit has been appended to each byte

Parity Check (3)

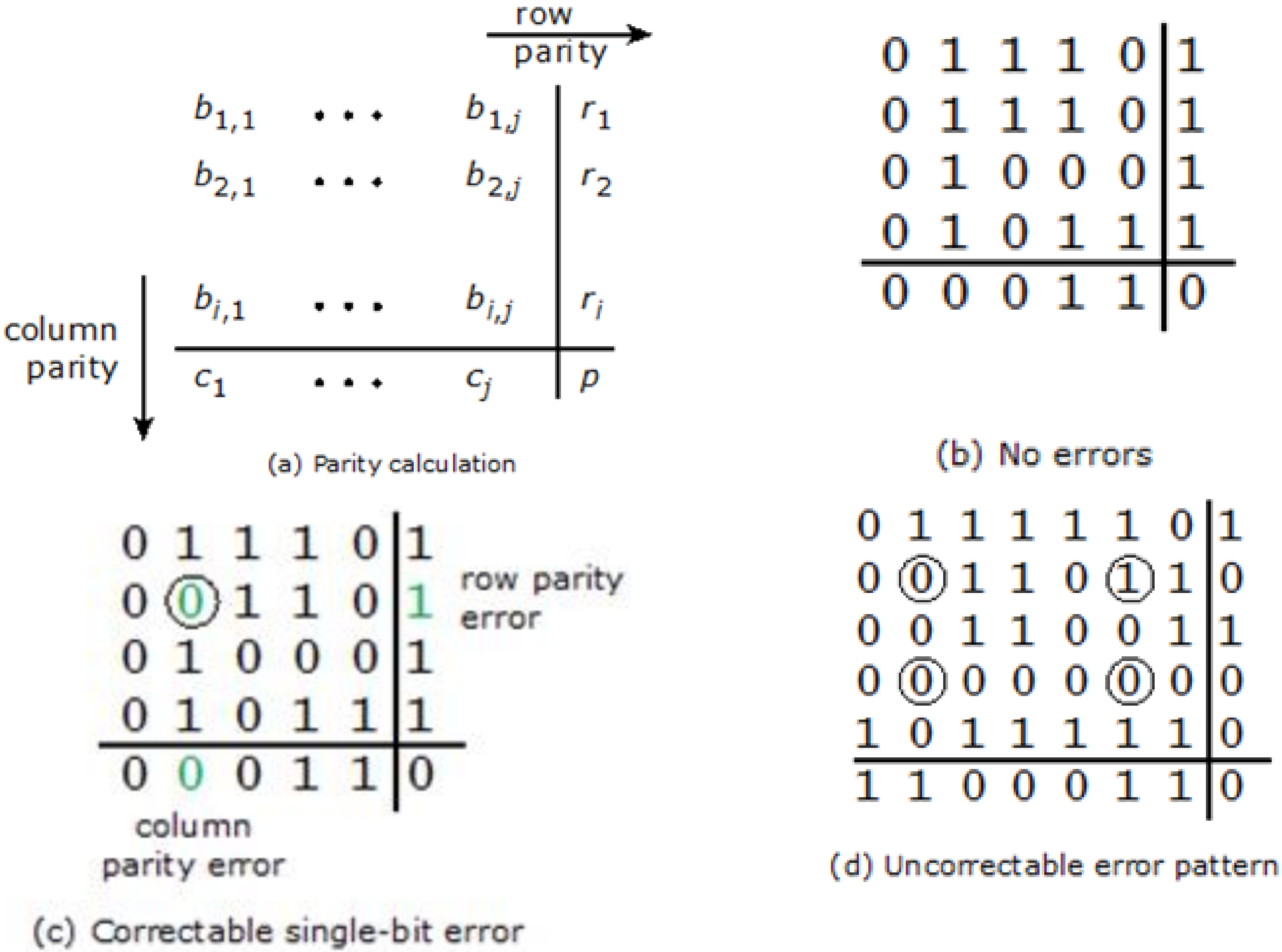
- Example 1
  - Even parity is used
  - 01011101?
  - 010111011
- Example 2
  - Odd parity is used
  - 01011101?
  - 010111010

- Q: what is the problem with error detection using the parity bit?
- A: if two bits are in error then the parity check fails

Parity check (4)

- The two-dimensional (2D) parity scheme is more robust than the single parity bit.
- The string of data bits to be checked is arranged in a two-dimensional array.
- Appended to each row i is an even parity bit r_i for that row, and appended to each column j is an even parity bit c_j for that column.
- An overall parity bit p completes the matrix. Thus the error-detecting code consists of $i + j + 1$ parity bits.

Parity Check (5)



(a) Parity calculation

(b) No errors

(c) Correctable single-bit error

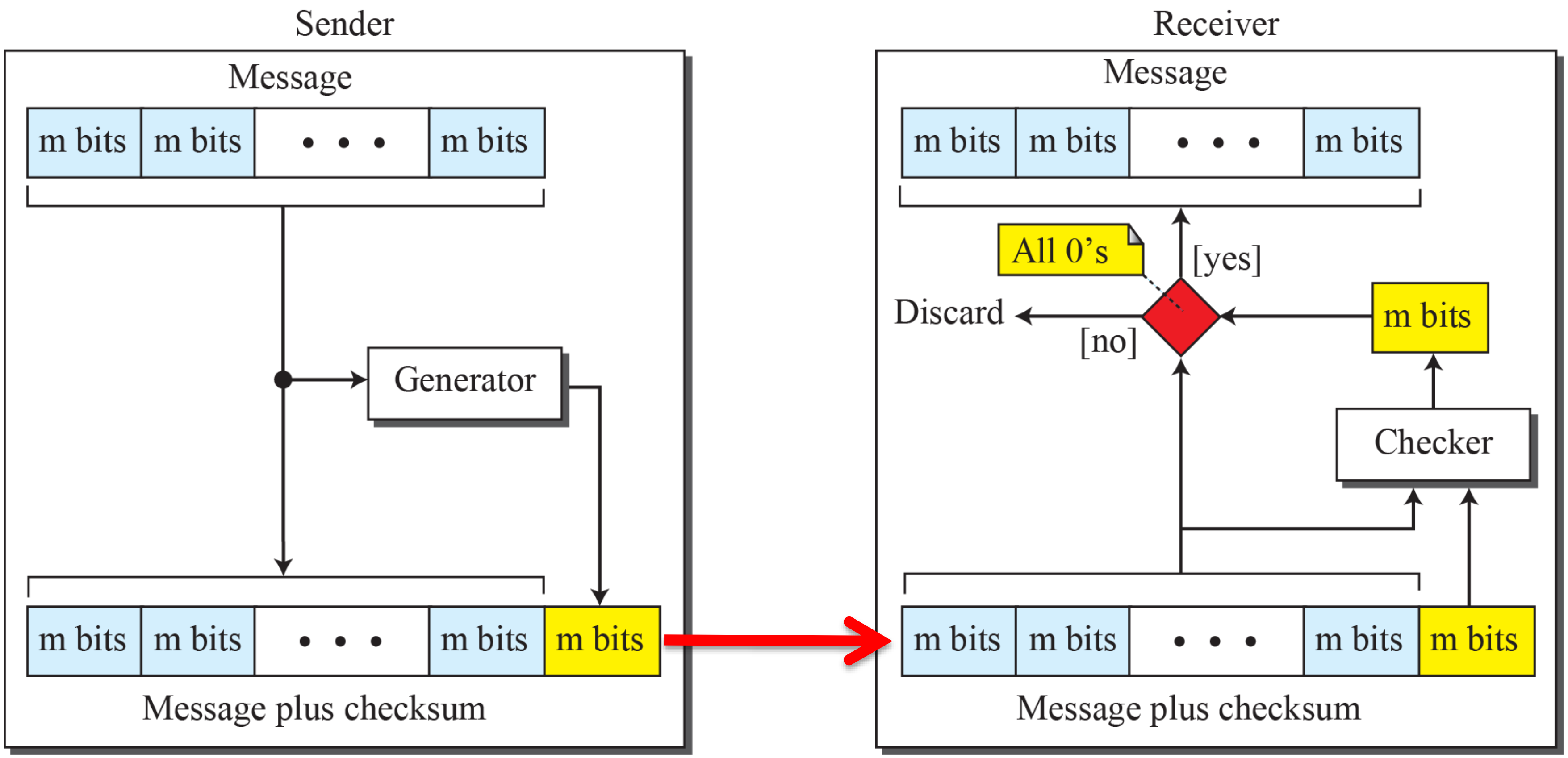
(d) Uncorrectable error pattern

Outline

- Introduction
  - Learning objectives
  - Background
  - Types of errors
- Error Detection
  - Parity check
    - Parity bit / 2-D Parity check
  - **Internet checksum**
  - Cyclic Redundancy Check (CRC)
- Forward Error Correction
  - Block Code Principles
- Summary

The Internet Checksum

- Checksum is 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.
  - including IP, *TCP*, and *UDP* *(to be covered in TCP/UDP module)*



The Internet Checksum (2)

- The idea of the traditional checksum is simple.
- We show this using a 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 not accepted.

The Internet Checksum (3)

- The calculation makes use of:
  - Ones-complement addition
    1. The two numbers are treated as unsigned binary integers and added
    2. If there is a carry out of the leftmost bit, add 1 to the sum (*end-around carry*)

ex1.

```
  0011
 +1100
---------
  1111
```

ex2.

```
   1101
 + 1011
---------
  11000
 +     1
-------------
   1001
```

  - Ones-complement operation on a set of binary digits
    - o Replace 0 digits with 1 digits and 1 digits with 0 digits

The Internet Checksum (4)

- Typically, the checksum is included as a field in the header of a protocol data unit, such as in IP datagram.
- To compute the checksum,
  - the checksum field is first set to all *zeros*.
  - The checksum is then calculated by performing the *ones-complement addition* of all the words in the header, and then taking the *ones-complement operation* of the result.
- This result is placed in the *checksum* field.
- To verify a checksum,
  - the ones-complement sum is computed over the same set of octets, including the checksum field. If the result is all 1 bits (- 0 in ones-complement arithmetic), the check succeeds.

The Internet Checksum (5)

- Consider a header that consists of 10 octets
  - with the checksum in the last two octets with the following content (in hexadecimal)

00 01 F2 03 F4 F5 F6 F7 00 00

Checksum calculation by sender

Partial sum	0001 F203 F204
Partial sum	F204 F4F5 1E6F9
Carry	E6F9 1 E6FA
Partial sum	E6FA F6F7 1DDF1
Carry	DDF1 1 DDF2
Ones complement of the result	220D

Checksum calculation by receiver

Partial sum	0001 F203 F204
Partial sum	F204 F4F5 1E6F9
Carry	E6F9 1 E6FA
Partial sum	E6FA F6F7 1DDF1
Carry	DDF1 1 DDF2
Partial sum	DDF2 220D FFFF

The result is a value of all ones, which verifies that no errors have been detected.

The Internet Checksum (6)

- provides greater error-detection capability than a parity bit or two-dimensional parity scheme
  - but is considerably *less* effective than the cyclic redundancy check (CRC), discussed next.
- The primary reason for its adoption in Internet protocols is ***efficiency***.
  - Most of these protocols are implemented in software and the Internet checksum, involving simple addition and comparison operations, causes very little overhead.
- It is assumed that
  - at the lower link level, a strong error-detection code such as CRC is used,
  - and so the Internet checksum is simply an additional end-to-end check for errors.

Outline

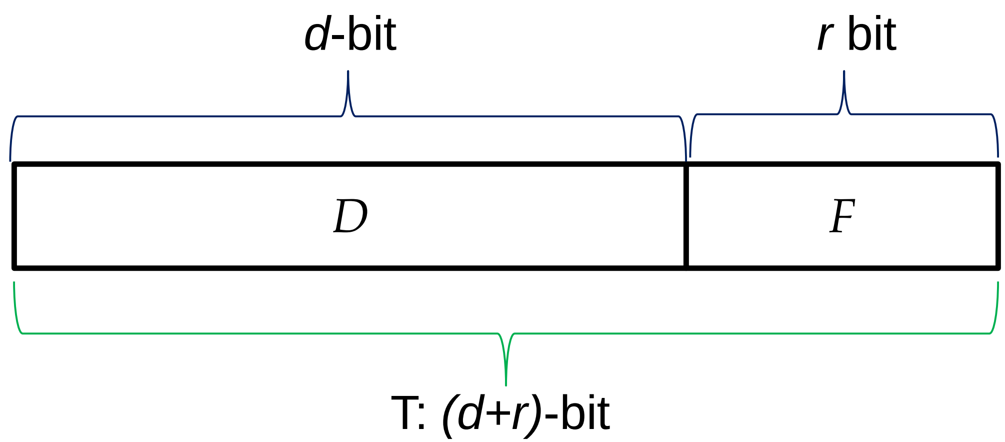
- Introduction
  - Learning objectives
  - Background
  - Types of errors
- Error Detection
  - Parity check
    - Parity bit / 2-D Parity check
  - Internet checksum
  - **Cyclic Redundancy Check (CRC)**
- Forward Error Correction
  - Block Code Principles
- Summary

Cyclic Redundancy Check (CRC)

- One of the most common and powerful **error-detecting codes**
- *d*-bit data to be sent;
- Sender and receiver first agree on an *r*+1 pattern (generator); leftmost bit is 1;
- Sender chooses *r* additional bits appending the *d*-bit data such that the *d*+*r* bits data is divisible by the generator using modulo-2 arithmetic;
- Receiver divides the incoming *d*+*r* bits data by the generator
  - If there is no remainder, assume there is no error.

CRC (3)

- Now define
  - T = $(d+r)$-bit frame to be transmitted
  - D = d-bit block of data
  - F = r additional bits
  - G = pattern of $r+1$ bits; this is the predetermined divisor



- We would like T/P (using modulo-2 arithmetic) to have no reminder.

CRC (2)

- Modulo-2 arithmetic
  - Addition and subtraction: Uses binary addition with no *carries, just XOR*

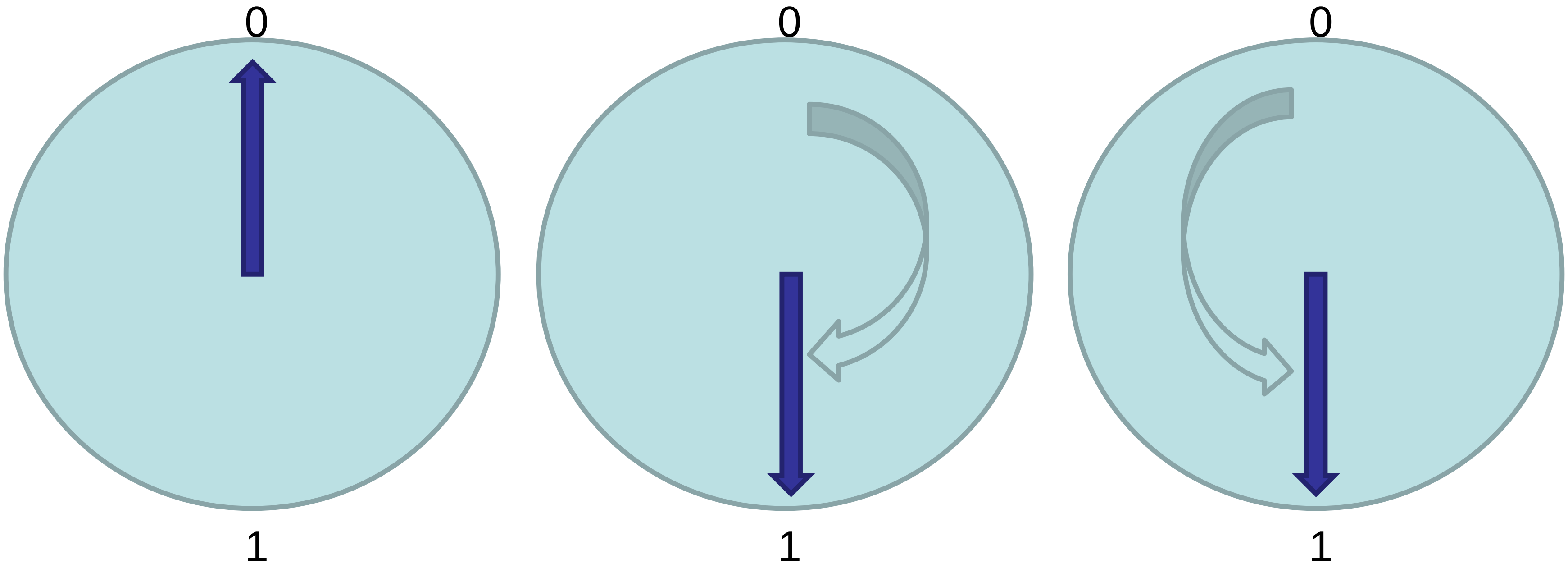
```
 1111
+1010
--------
 0101
```

```
 1111
-0101
--------
 1010
```

  - Multiplication and division: the same as in base-2 arithmetic, but add or sub is done without carries or borrows (XOR).

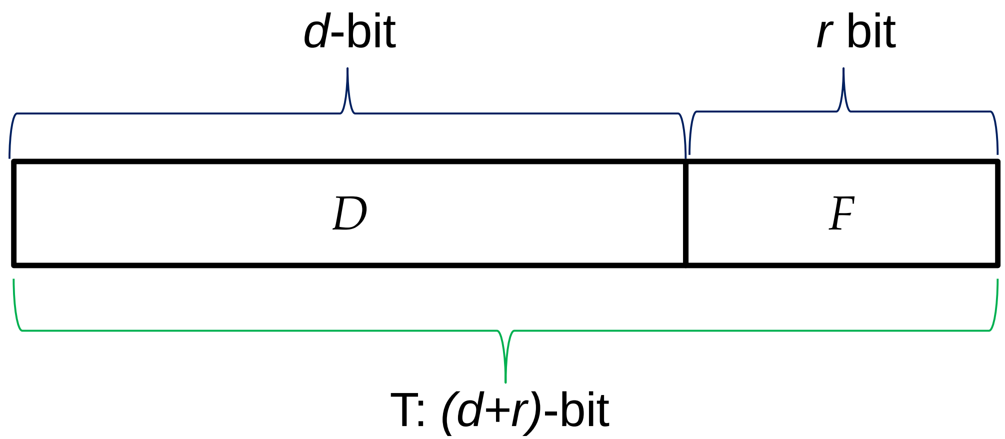
```
   11001
    X 11
---------
   11001
  11001
--------------
  101011
```

```
     __1100____
10  11001
    10
    010
     10
      00
      00
       1
```


0
1
0
1
0
1

How to generate the *r* bits

- The (*d*+*r*) bits data is $D * 2^r$ XOR F;
- Then
  - $D * 2^r$ XOR F = nG;
  - $D * 2^r$ = nG XOR F;
  - F = remainder of $(D * 2^r)/G$;



Source: [KR3]

CRC (5)

```
                          1 1 0 1 0 1 0 1 1 0
              ___________________________________
G ──▶ 1 1 0 1 0 1 / 1 0 1 0 0 0 1 1 0 1 0 0 0 0 0  ◀── D*2^r
                    1 1 0 1 0 1
                    -------------------------------
                      1 1 1 0 1 1
                      1 1 0 1 0 1
                      -----------------------------
                          1 1 1 0 1 0
                          1 1 0 1 0 1
                          -------------------------
                              1 1 1 1 1 0
                              1 1 0 1 0 1
                              ---------------------
                                  1 0 1 1 0 0
                                  1 1 0 1 0 1
                                  -----------------
                                    1 1 0 0 1 0
                                    1 1 0 1 0 1
                                    ---------------
                                      0 1 1 1 0  ◀── R
```

4. The remainder is added to 2^5D to give T= 1010001101**01110**
5. If there are no errors, the receiver receives *T* intact (i.e., no damage). The received frame is divided by *G*:

   (see the next page)

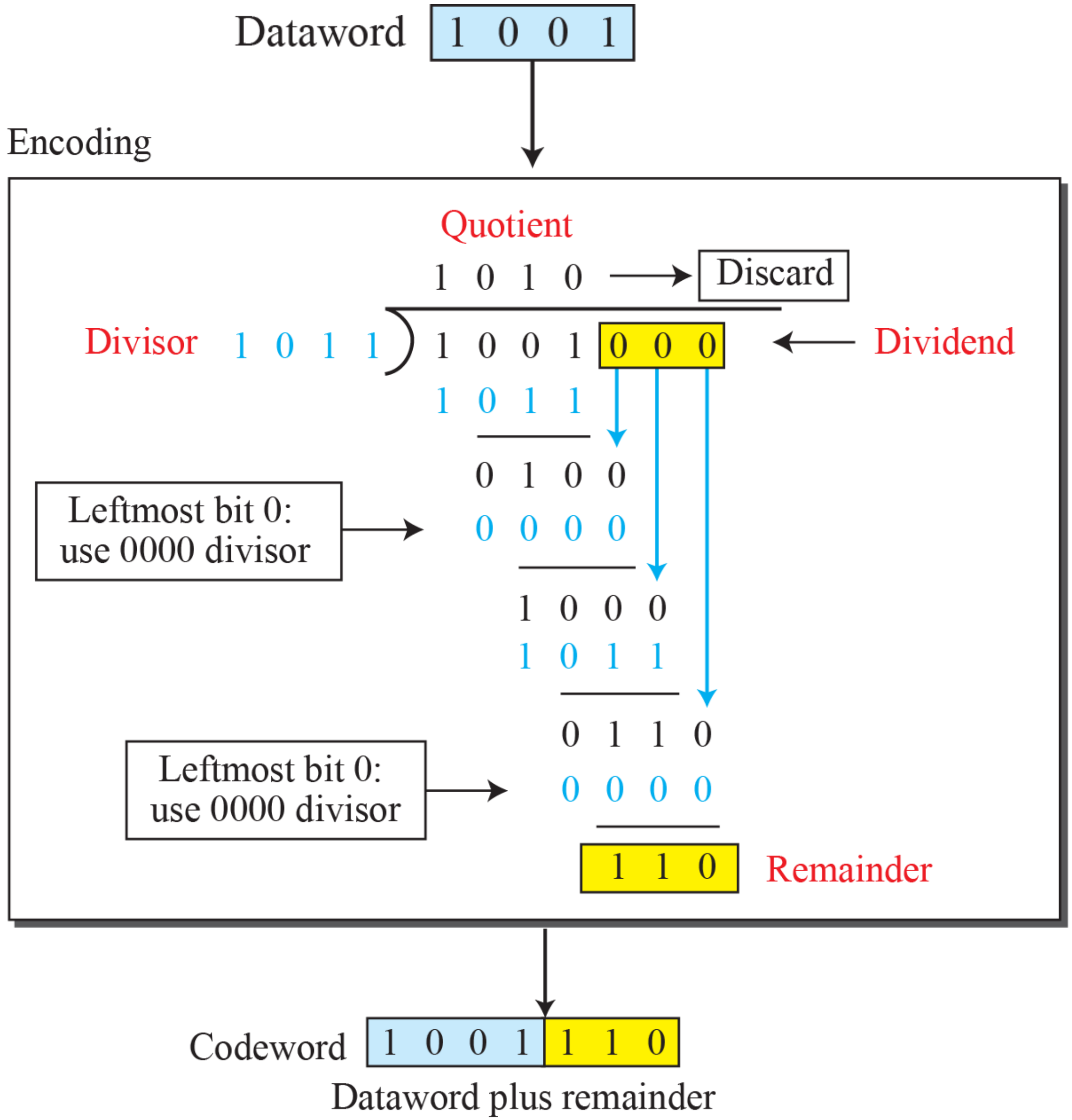
CRC (6)

```
                      1 1 0 1 0 1 0 1 1 0
           ____________________________________
G → 1 1 0 1 0 1 / 1 0 1 0 0 0 1 1 0 1 0 1 1 1 0  ← T
                  1 1 0 1 0 1
                  -----------
                    1 1 1 0 1 1
                    1 1 0 1 0 1
                    -----------
                      1 1 1 0 1 0
                      1 1 0 1 0 1
                      -----------
                          1 1 1 1 1 0
                          1 1 0 1 0 1
                          -----------
                              1 0 1 1 1 1
                              1 1 0 1 0 1
                              -----------
                                1 1 0 1 0 1
                                1 1 0 1 0 1
                                -----------
                                          0  ← R
```

- Because there is no remainder, it is assumed that there have been no errors

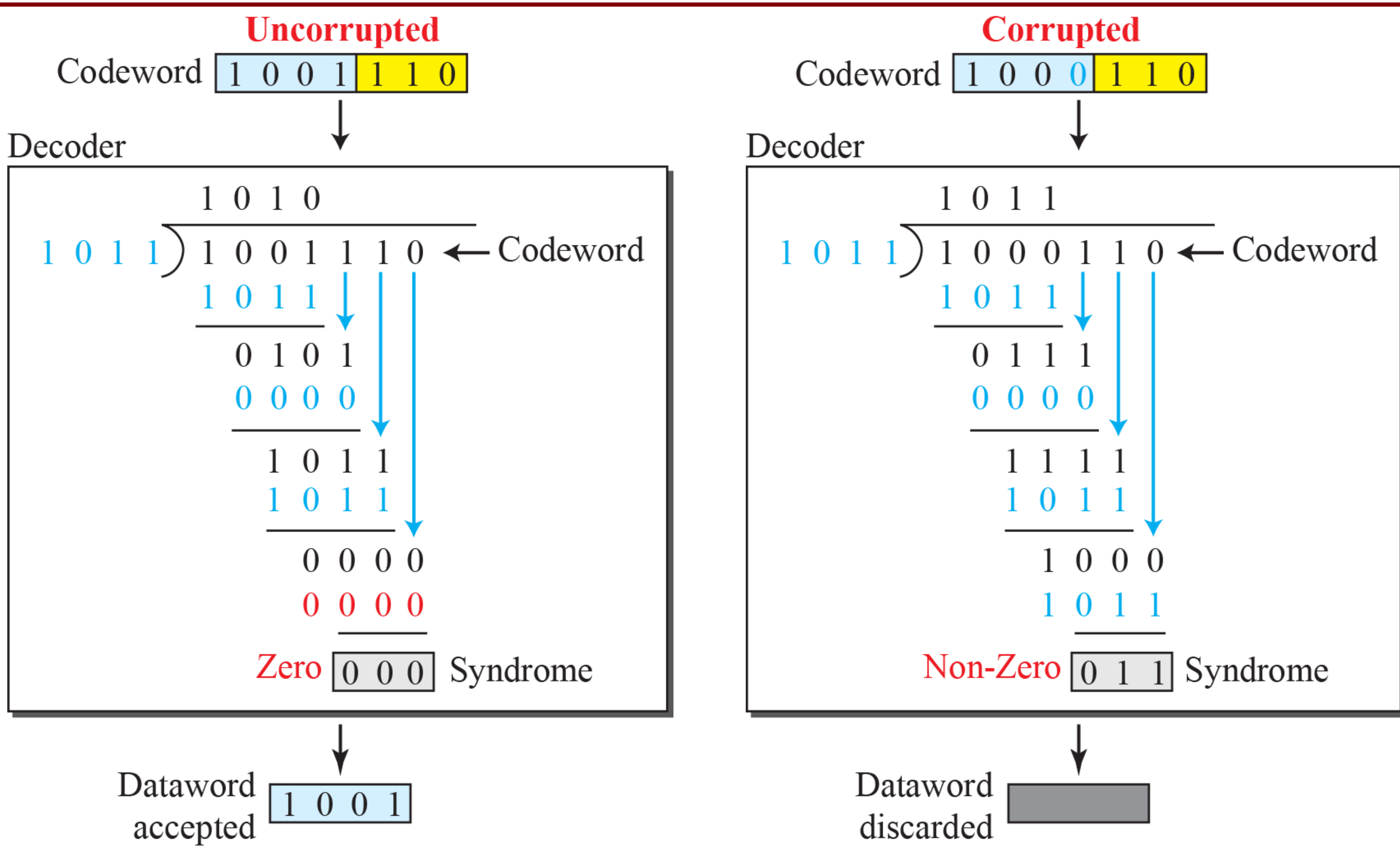
Figure 10.6: Division in CRC encoder

dividend ÷ divisor = *quotient*



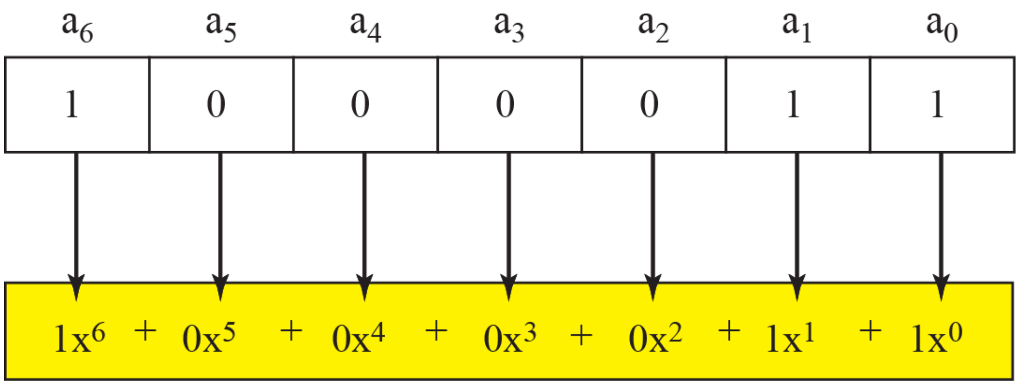
Note:
Multiply: AND
Subtract: XOR

Figure 10.7: Division in the CRC decoder for two cases

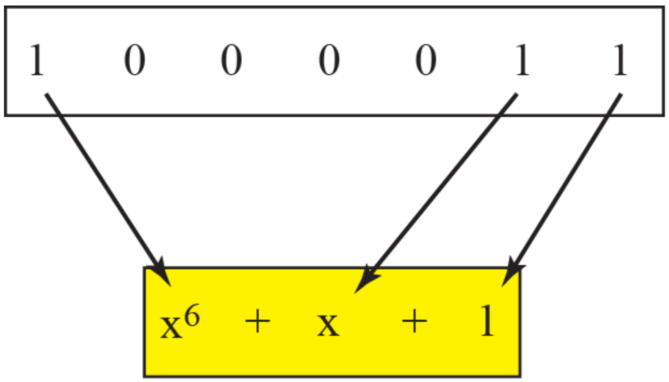


Binary vs. polynomials

- A better way to understand ***cyclic codes*** and how they can be analyzed is to represent them as ***polynomials***.
- In practice, all commonly used CRCs employ the Galois field of two elements, GF(2).
  - The two elements are usually called 0 and 1, comfortably matching computer architecture; A pattern of 0s and 1s 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.

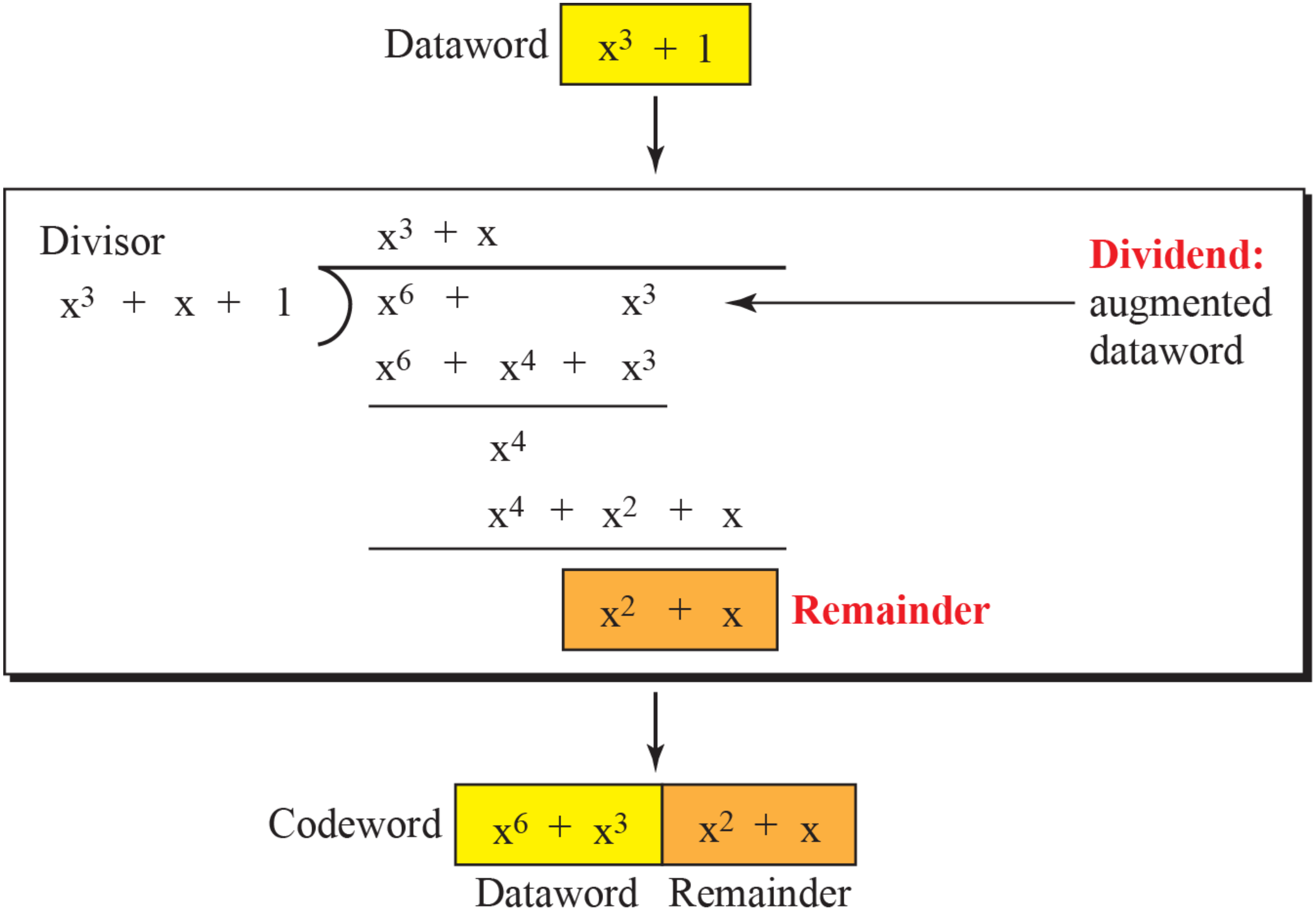


a. Binary pattern and polynomial



b. Short form

CRC division using polynomials



Example using a polynomials

- Message D= $X^7 + X^4 + X^3 + X^1$, 10011010
- $2^{n-k}D = 10001101\textcolor{red}{000}$
- P = 1101

1 1 1 1 1 0 0 1

P → 1 1 0 1 ⟌ 1 0 0 1 1 0 1 0 **0 0 0** ← T

1 1 0 1

1 0 0 1

1 1 0 1

1 0 0 0

1 1 0 1

1 0 1 1

1 1 0 1

1 1 0 0

1 1 0 1

1 0 0 0

1 1 0 1

1 0 1 ← *R*

CRC – Some Standard Polynomials

- Four versions of *P(X)* are widely used.

CRC-12	$X^{12}+X^{11}+X^3+X^2+X+1$
CRC-16	$X^{16}+X^{15}+X^2+1$
CRC-CCITT	$X^{16}+X^{12}+X^5+1$
CRC-32	$X^{32}+X^{26}+X^{23}+X^{22}+X^{16}+X^{12}+X^{11}+X^{10}+X^8+X^7+X^5+X^4+X^2+X+1$

  - CRC-12: for transmission of streams of 6-bit characters and generates a 12-bit frame check sequence (FCS)
  - CRC-16 and CRC-CCITT: are popular for 8-bit characters and result in a 16-bit FCS; High-Level Data Link Control (**HDLC**)
  - CRC-32: is specified as an option in some point-to-point synchronous transmission standards and is used in **IEEE 802 LAN** standards
  - An example: http://srecord.sourceforge.net/crc16-ccitt.html#overview

Outline

- Introduction
  - Learning objectives
  - Background
  - Types of errors
- Error Detection
  - Parity check
    - Parity bit / 2-D Parity check
  - Internet checksum
  - Cyclic Redundancy Check (CRC)
- **Forward Error Correction**
  - **Block Code Principles**
- Summary

Detection vs. Correction

- In error detection, we are only looking to see if any error has occurred.
  - The answer is a simple yes or no.
  - We are not even interested in the number of corrupted bits.
  - A single-bit error is the same for us as a burst error.
- The **correction** of errors is more difficult than the detection.
- In error correction, we need to know the exact number of bits that are corrupted and, more importantly, their location in the message.

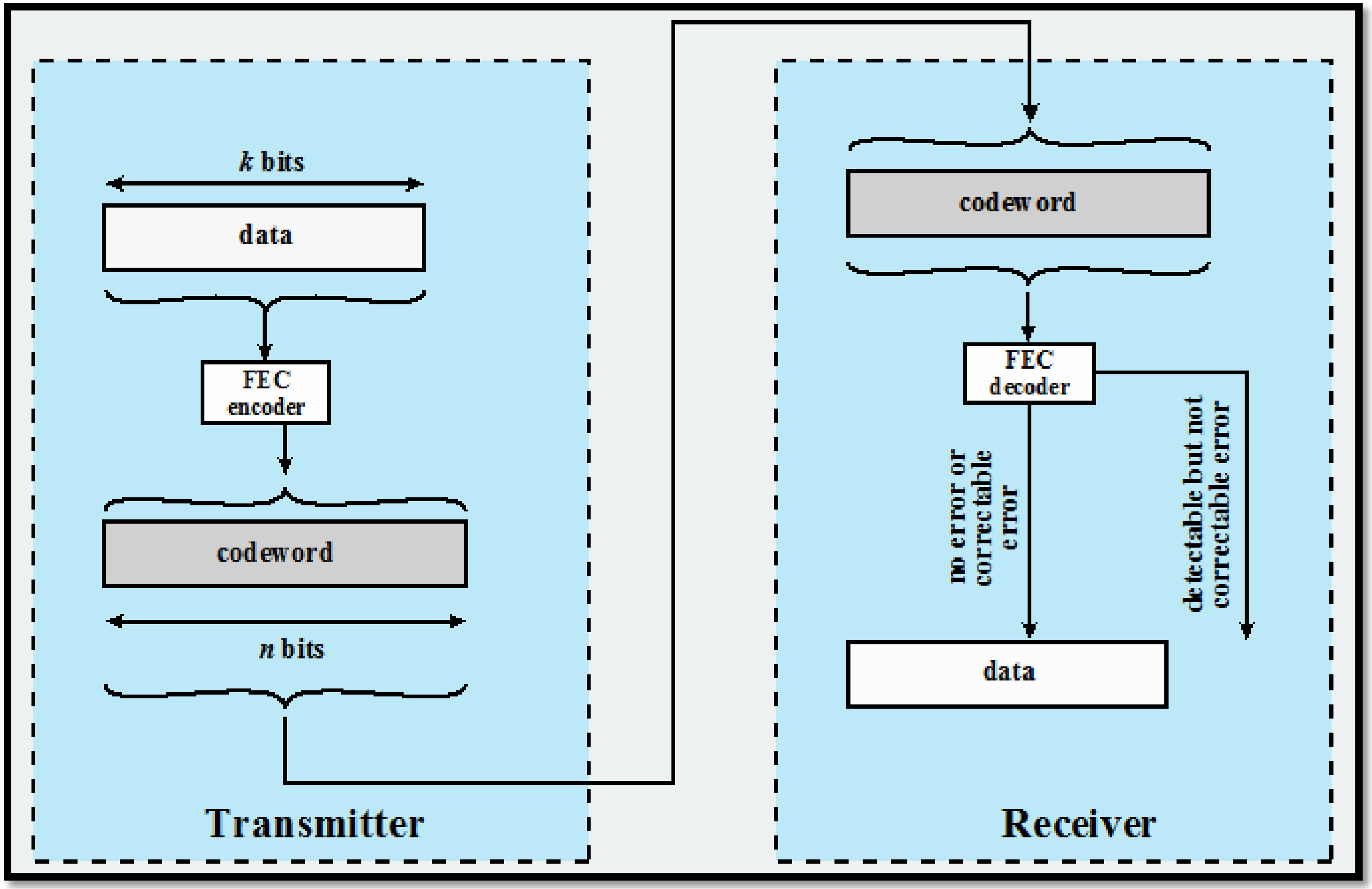
Forward Error Correction (FEC)

- Correction of detected errors usually requires data blocks to be ***retransmitted***
- Not appropriate for wireless applications:
  - The bit error rate (BER) on a wireless link can be quite high, which would result in a large number of retransmissions
  - Propagation delay is very long compared to the transmission time of a single frame
- **Need to correct errors on basis of bits received**

Codeword
• On the transmission end each k-bit block of data is mapped into an n-bit block ($n > k$) using a forward error correction (FEC) encoder

FEC Process

- In block coding, we divide our message into blocks, each of k bits, called *datawords*. We add *r* redundant bits to each block to make the length *n* = *k* + *r*. The resulting *n*-bit blocks are called *codewords*.



FEC Process (2)

- During transmission, the signal is subject to impairments, which may produce bit errors in the signal.
- At the receiver, the incoming signal is demodulated to produce a bit string that is similar to the original codeword but may contain errors.

One of four possible outcomes:

1. **No errors:**
   - If there are no bit errors, the input to the FEC decoder is identical to the original codeword, and the decoder produces the original data block as output.
2. **Detectable, correctable errors:**
   - For certain error patterns, it is possible for the decoder to detect and correct those errors.
   - Thus, even though the incoming data block differs from the transmitted codeword, the FEC decoder is able to map this block into the original data block.
3. **Detectable, not correctable errors:**
   - For certain error patterns, the decoder can detect but not correct the errors.
   - In this case, the decoder simply reports an uncorrectable error.
4. **Undetectable errors:**
   - For certain, typically rare, error patterns, the decoder does not detect the error and maps the incoming n-bit data block into a *k*-bit block that differs from the original k-bit block.

An example

- Let us assume that *k* = 2 and *n* = 3. Table 10.1 shows the list of datawords and codewords.

Table 10.1: A code for error detection in Example 10.1

Datawords	*Codewords*	*Datawords*	*Codewords*
00	000	10	101
01	011	11	110

Assume the sender encodes the dataword 01 as 011 and sends it to the receiver. Consider the following cases:

1. The receiver receives 011. It is a valid codeword. The receiver extracts the dataword 01 from it.
2. The codeword is corrupted during transmission, and 111 is received (the leftmost bit is corrupted). This is not a valid codeword and is discarded.
3. The codeword is corrupted during transmission, and 000 is received (the right two bits are corrupted). This is a valid codeword. The receiver incorrectly extracts the dataword 00. Two corrupted bits have made the error undetectable.

Block Code Principles

- **Hamming distance**
  - $d(v_1, v_2)$ between two n –bit binary sequences v_1 and v_2 is the number of bits in which v_1 and v_2 disagree
- Redundancy of the code
  - The ratio of redundant bits to data bits $(n-k)/k$
- Code rate
  - The ratio of data bits to total bits k/n
  - Is a measure of how much additional bandwidth is required to carry data at the same data rate as without the code

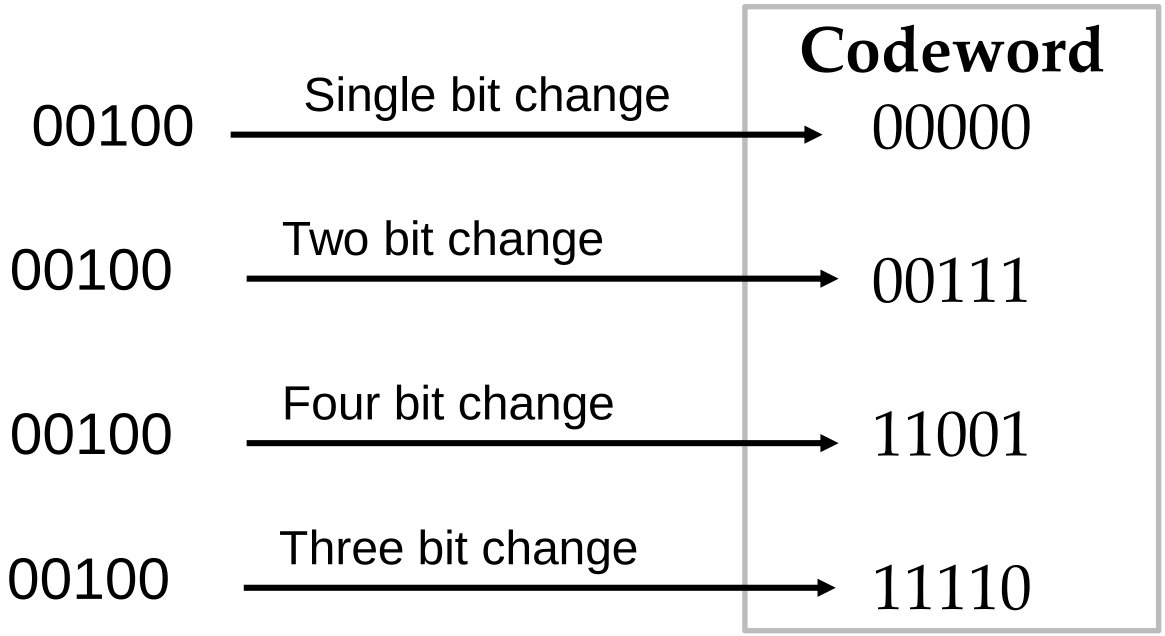
Block Code Principles (2)

- Hamming distance
  - E.g. v_1=011011, v_2=110001, $d(v_1,v_2) = 3$
- Suppose we wish to transmit blocks of data of length *k* bits.
  - Instead of transmitting each block as *k* bits, we map each *k*-bit sequence into a unique *n*-bit codeword.
  - E.g. For *k = 2* and *n=5*, we can make the following assignment:

Data Block (dataword)	Codeword
00	00000
01	00111
10	11001
11	11110

Block Code Principles (3)

- A codeword block is received with the bit pattern 00100
  - This is not a valid codeword



Block Code Principles (3)

- If an invalid codeword is received, then valid codeword is closest to it (*minimum* distance) is selected
  - This will only work if there is a unique valid codeward at a minimum distance from each invalid codeword
  - e.g., it is not true that for every invalid codeword there is one and only one valid codeword at a minimum distance
  - There are $2^5 = 32$ possible codwords of which 4 are valid, leaving 28 invalid codewords.

Block Code Principles (4)

Invalid codeword	Minimum Distance	Valid Codeword	Invalid Codeword	Minimum Distance	Valid Codeword
00001	1	00000	10000	1	00000
00010	1	00000	10001	1	11001
00011	1	00111	10010	2	**00000 or 11110**
00100	1	00000	10011	2	**00111 or 11001**
00101	1	00111	10100	2	**00111 or 11001**
00110	1	00111	10101	2	**00111 or 11001**
01000	1	00000	10110	1	11110
01001	1	11001	10111	1	00111
01010	2	**00000 or 11110**	11000	1	11001
01011	2	**00111 or 11001**	11010	1	11110
01100	2	**00000 or 11110**	11011	1	11001
01101	2	**00111 or 11001**	11100	1	11110
01110	1	11110	11101	1	11001
01111	1	00111	11111	1	11110

Block Code Principles (5)

- There are eight cases in which an invalid codeword is at a distance *2* from two different valid codewords.
  - Thus, if one such invalid codeword is received, and error in *2* bits could have caused it and the receiver has no way to choose between the two alternatives.
  - An error is detected but cannot be corrected
- This code is capable of correcting all single-bit errors but cannot correct double-bit errors

Summary

- Error Detection
  - Parity check
  - Internet checksum
  - Cyclic Redundancy Check (CRC)
- Forward Error Correction
  - Block Code Principles

References

- [KR3] James F. Kurose, Keith W. Ross, *Computer networking: a top-down approach featuring the Internet*, 3rd edition.
- [PD5] Larry L. Peterson, Bruce S. Davie, *Computer networks: a systems approach*, 5th edition
- [TW5] Andrew S. Tanenbaum, David J. Wetherall, *Computer network*, 5th edition
- [LHBi]Y-D. Lin, R-H. Hwang, F. Baker, *Computer network: an open source approach*, International edition

Acknowledgements

- All slides are developed based on slides from the following two sources:
  - Dr DongSeong Kim’s slides for COSC264, University of Canterbury;
  - Prof Aleksandar Kuzmanovic’s lecture notes for CS340,Northwestern University, https://users.cs.northwestern.edu/~akuzma/classes/CS340-w05/lecture_notes.htm