Error Detection

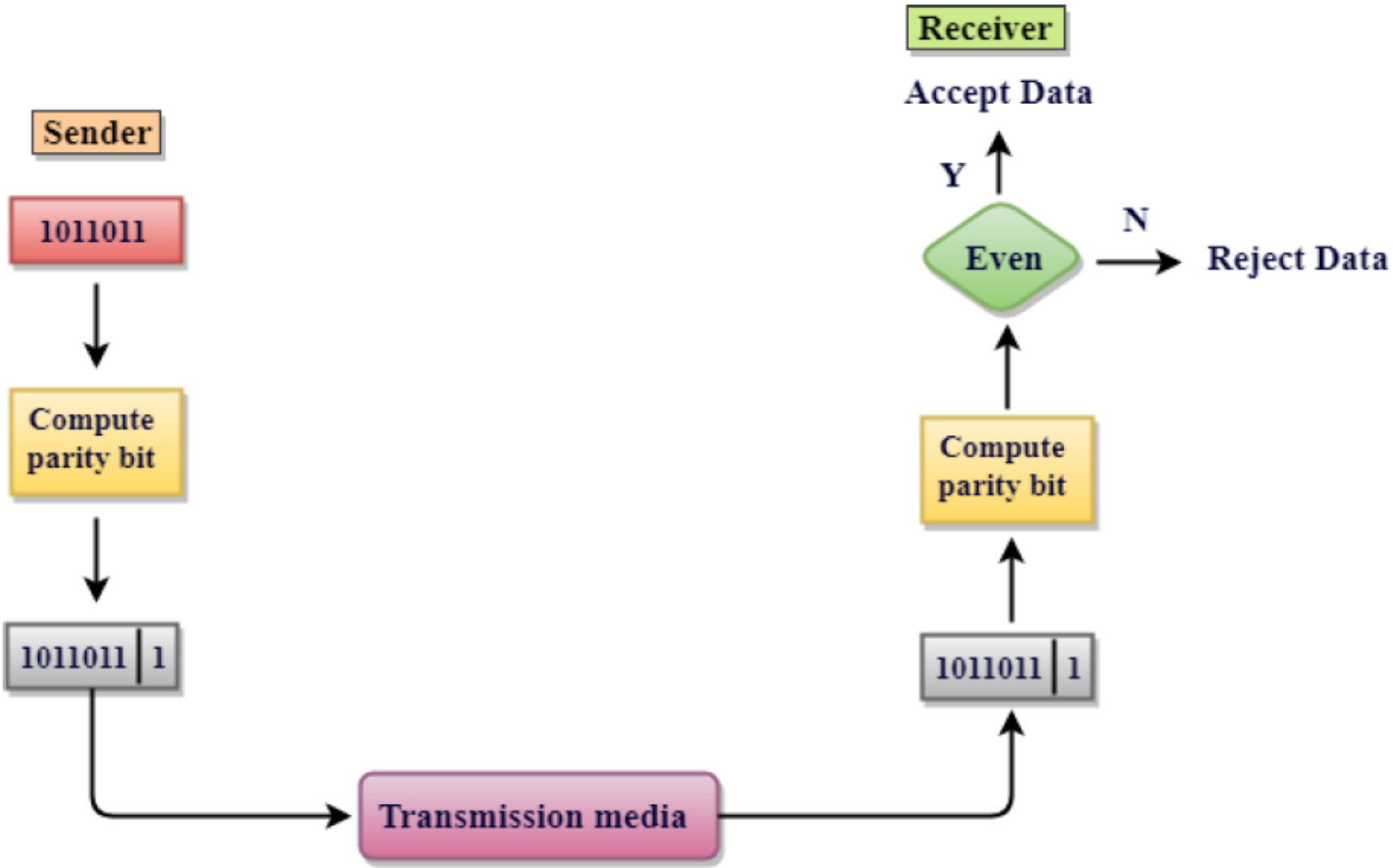
Overview

- Some bits may be received in error due to noise. How do we detect this?
  - Parity
  - Checksums
  - CRCs
- Detection will let us fix the error, for example, by retransmission (later)

Simple Error Detection-Parity Bit

- Take D data bits, add 1 check bit that is the sum of the check bits
  - Sum is modulo 2 or XOR

Parity Bit(2)

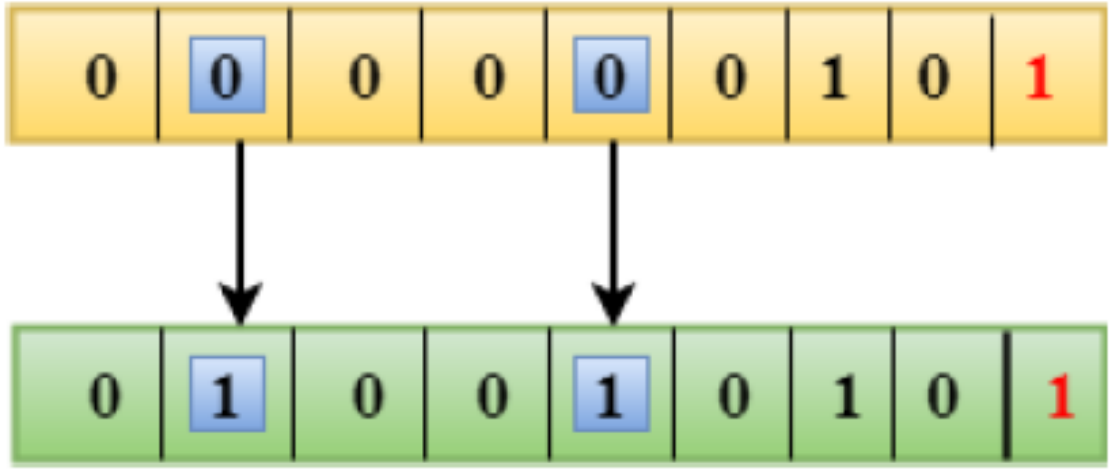

Receiver
Accept Data
Y
N
Even
Reject Data
Sender
1011011
Compute parity bit
1011011 | 1
Compute parity bit
1011011 | 1
Transmission media

Parity Bit(3)

- How well does parity work?
  - —What is the distance of the code?
  - —How many errors will it detect/correct?
- What about larger errors?

Drawbacks of Single Parity Checking

- It can only detect single-bit errors which are very rare.
- If two bits are interchanged, then it cannot detect the errors.



Two-Dimensional Parity Check

- Organizes the data in the form of a table
- Parity check bits are computed for each row, which is equivalent to the single-parity check.
- In Two-Dimensional Parity check, a block of bits is divided into rows, and the redundant row of bits is added to the whole block.
- At the receiving end, the parity bits are compared with the parity bits computed from the received data.

Two-Dimensional Parity Check(2)

Drawbacks Of 2D Parity Check

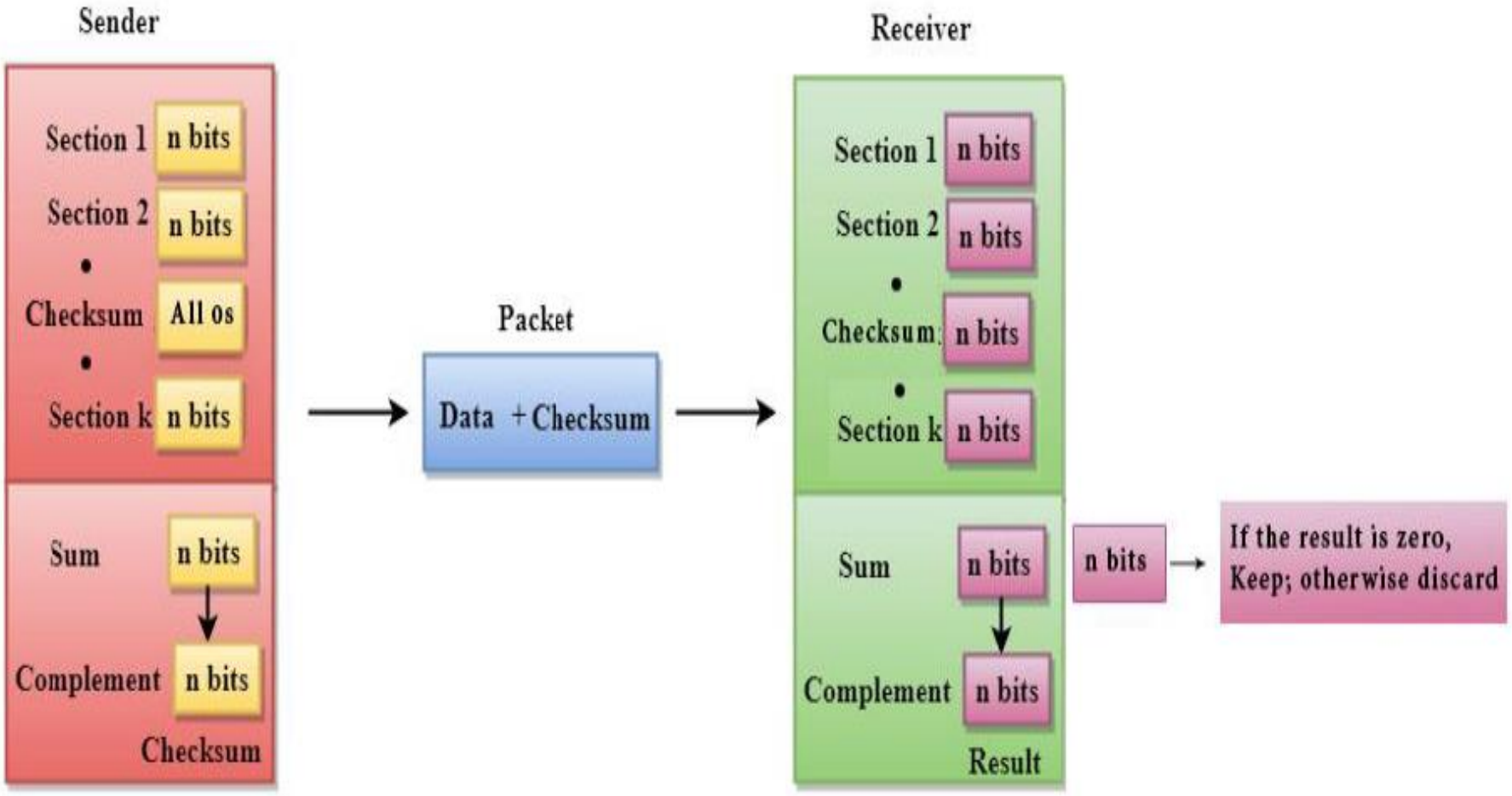
- If two bits in one data unit are corrupted and two bits exactly the same position in another data unit are also corrupted, then 2D Parity checker will not be able to detect the error.
- This technique cannot be used to detect the 4-bit errors or more in some cases.

Checksums

- Idea: sum up data in N-bits words
  - —Widely used in, e.g., TCP/IP/UDP

- Stronger protection than parity

Checksums(2)


Sender
Section 1
n bits
Section 2
n bits
Checksum
All 0s
Section k
n bits
Sum
n bits
Complement
n bits
Checksum
Packet
Data + Checksum
Receiver
Section 1
n bits
Section 2
n bits
Checksum
n bits
Section k
n bits
Sum
n bits
Complement
n bits
Result
n bits
If the result is zero,
Keep; otherwise discard

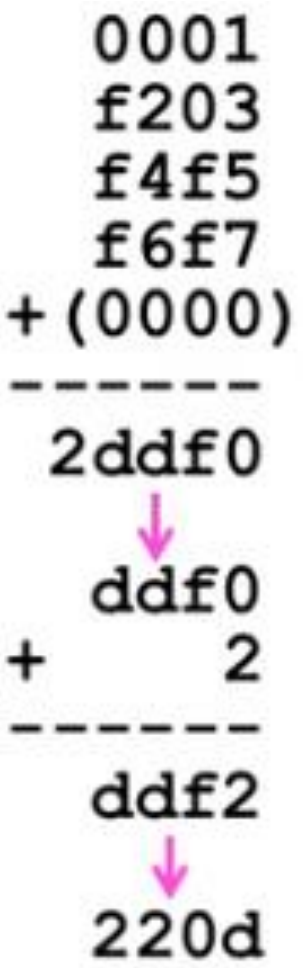
Internet Checksum

- Sum is defined in 1s complement arithmetic (must add back carries)
- And it's negative sum

Internet Checksum(2)

Sending:

1. Arrange data in 16 bit words
2. Put zero in checksum position, add

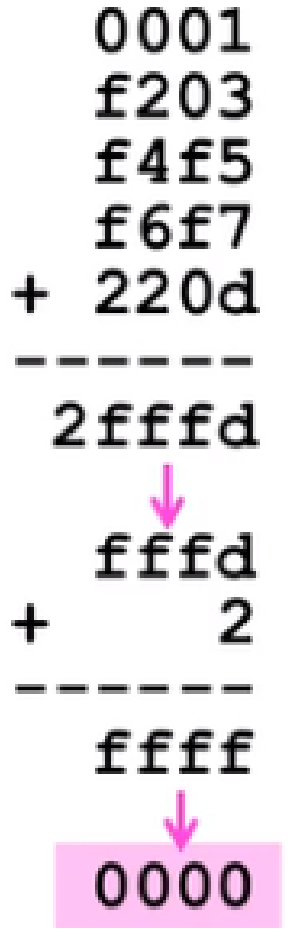


3. Add any carryover back to get 16 bits
4. Negate (complement) to get sum

Internet Checksum(3)

Receiving:

1. Arrange data in 16 bit words
2. Checksum will be non-zero, add



3. Add any carryover back to get 16 bits
4. Negate (complement) and check it is zero

Internet Checksum(4)

- How well does the checksum work?
  - —What is the distance of the code?
  - —How many errors will it detect/correct?
- What about larger errors?

Cyclic Redundancy Check (CRC)

- Even stronger protection
  - —Given n data bits, generate k check bits such that the n+k bits are evenly divisible by a generator C

- Example with numbers:
  - —n=302, k=one digit, C=3

CRCs(2)

- The catch:
  - It's based on mathematics of finite fields, in which "numbers" represent polynomials
  - E.g., 10011010 is $x^7 + x^4 + x^3 + x^1$
- What this means:
  - We work with binary values and operate using modulo 2 arithmetic.

CRCs(3)

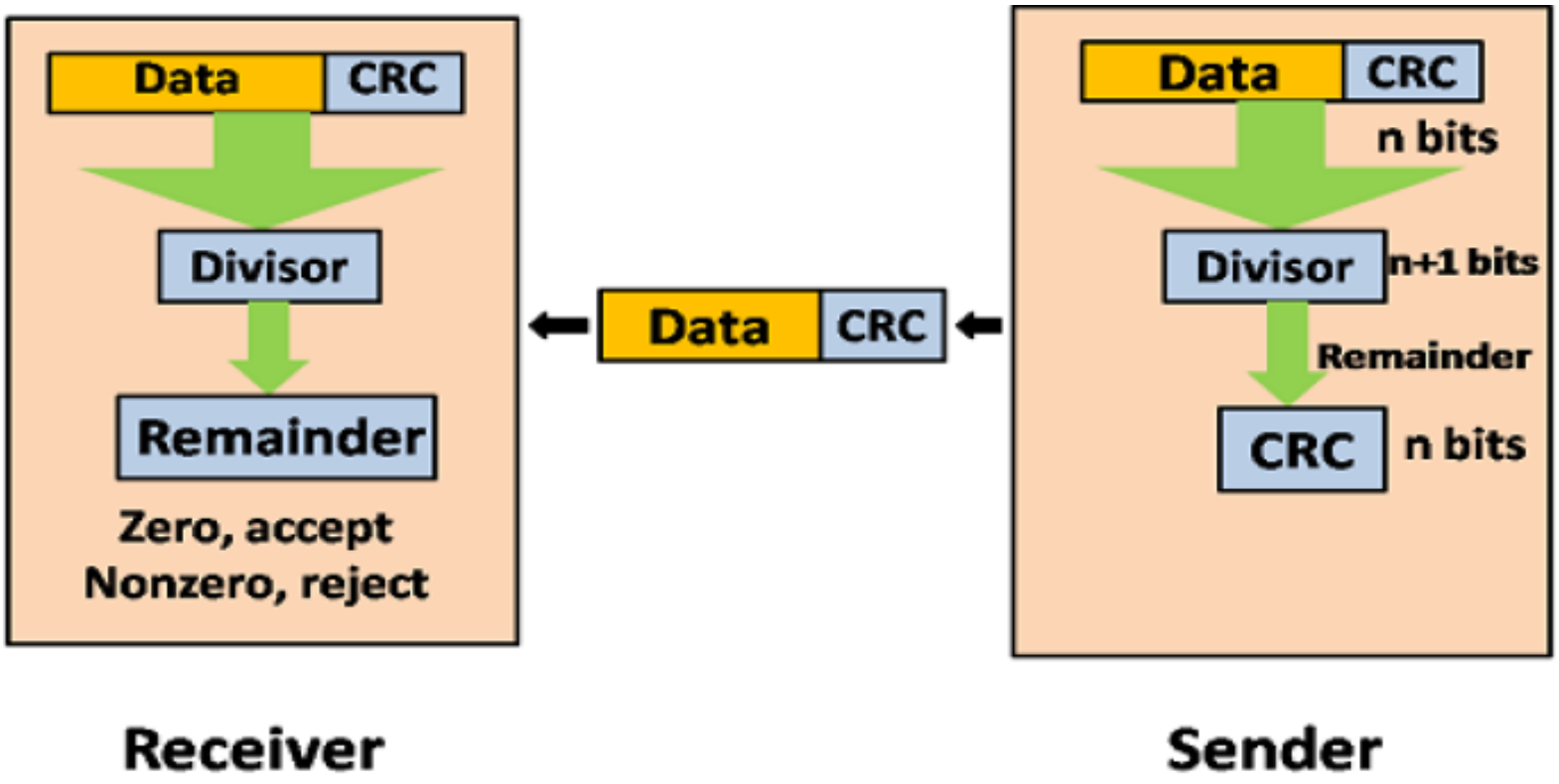
- Send procedure:

1. Extend the n data bits with k zeros
2. Divide by the generator value C
3. Keep remainder, ignore quotient
4. Adjust k check bits by remainder

- Receive Procedure:

1. Divide and check for zero remainder

CRCs(4)


Data
CRC
Divisor
Remainder
Zero, accept
Nonzero, reject
Data
CRC
Data
CRC
n bits
Divisor
n+1 bits
Remainder
CRC
n bits
Receiver
Sender

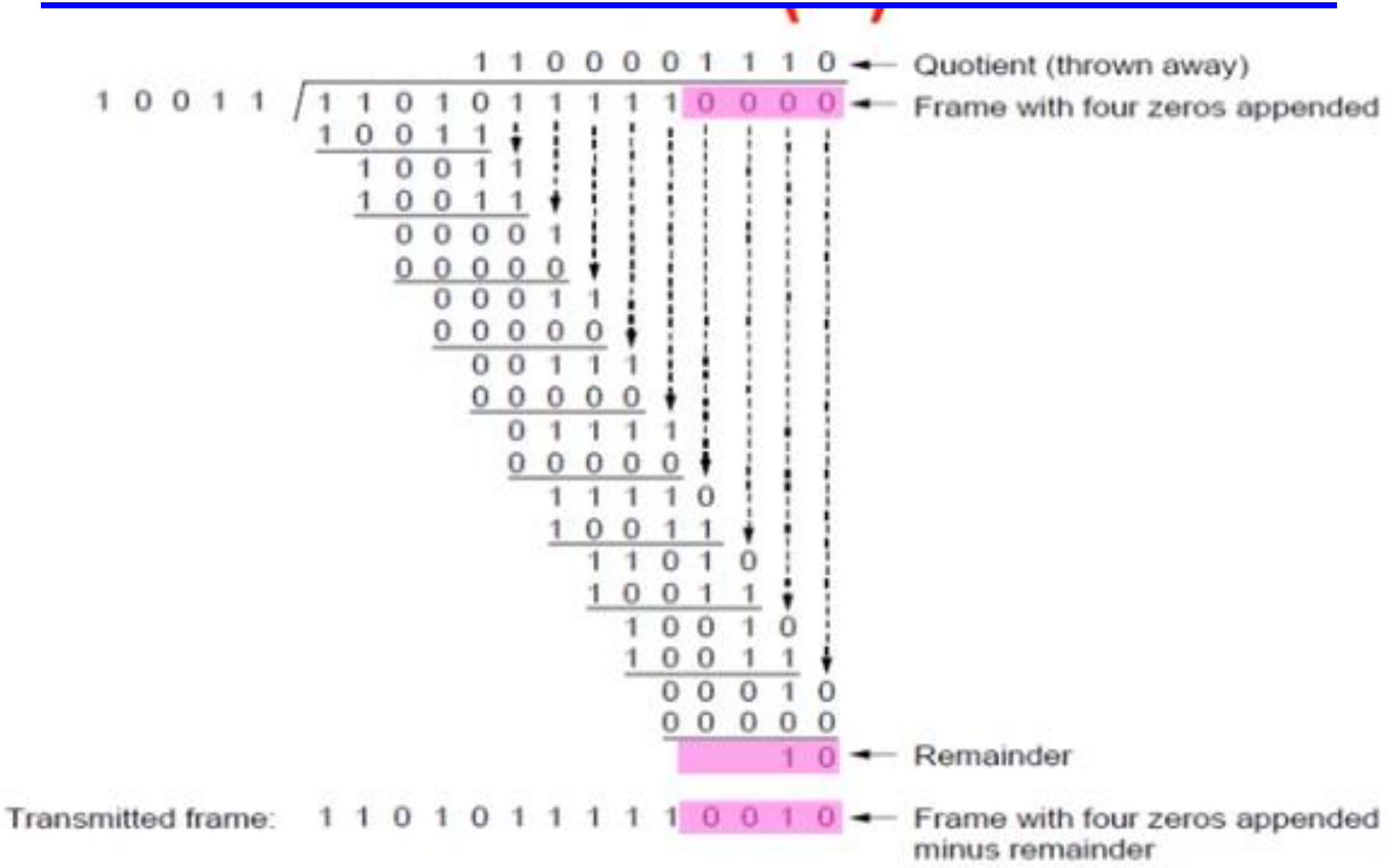
CRCs(5)

Data bits:
1101011111

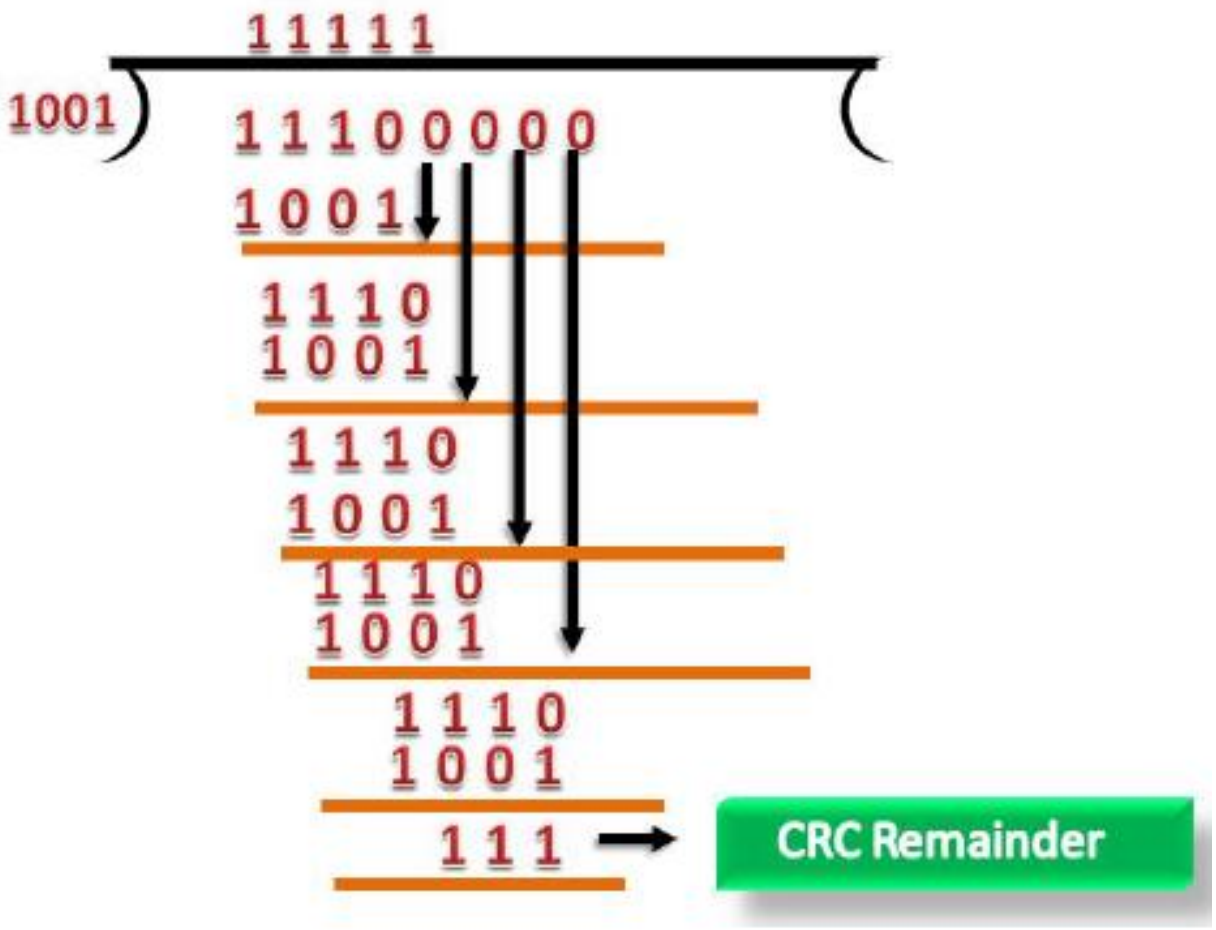
Check bits:
$C(x)=x^4+x^1+1$
C = 10011
k = 4

$$10011 \overline{)\,1\ 1\ 0\ 1\ 0\ 1\ 1\ 1\ 1\ 1}$$

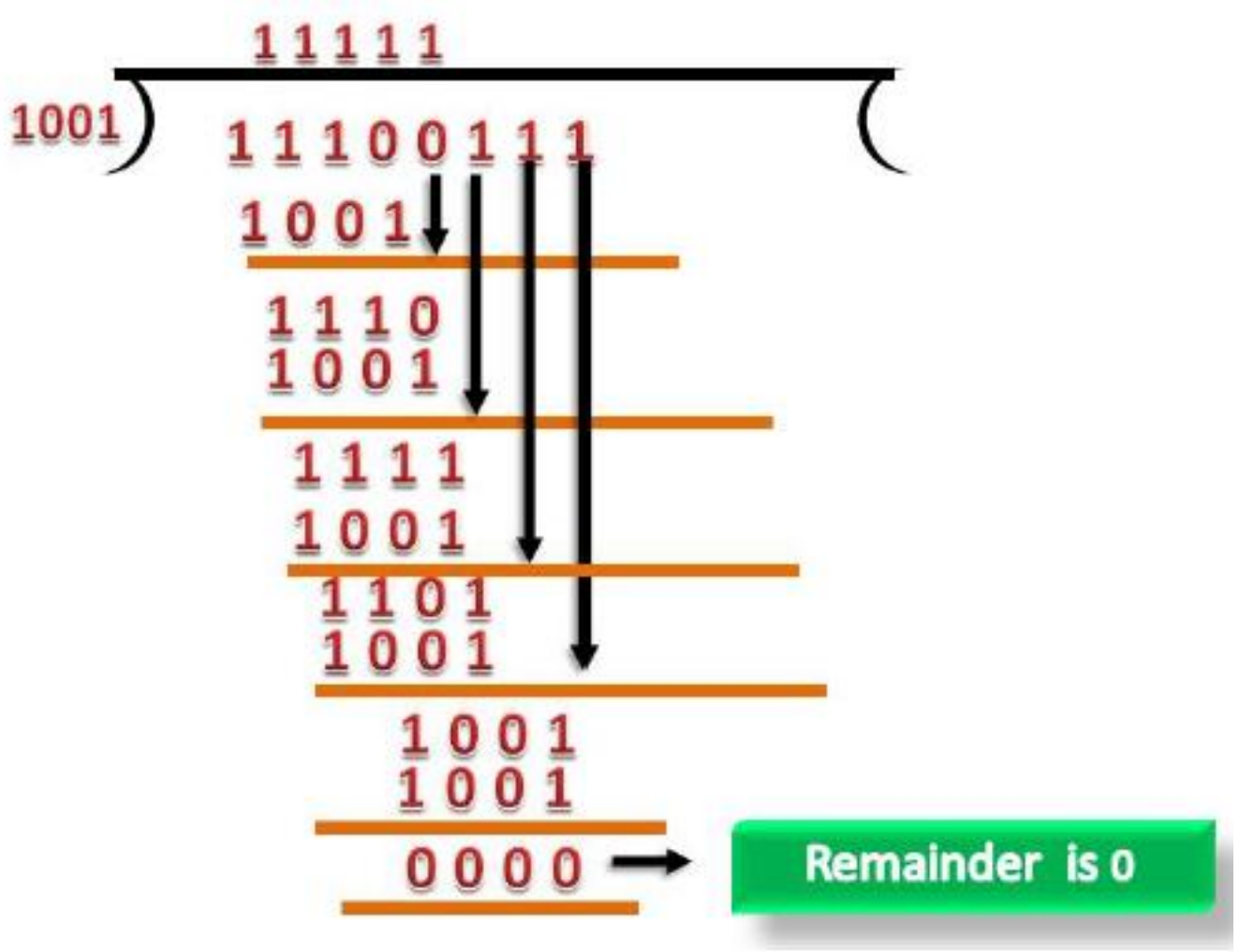
CRCs(6)


1 1 0 0 0 0 1 1 1 0 ← Quotient (thrown away)
1 0 0 1 1 | 1 1 0 1 0 1 1 1 1 1 0 0 0 0 ← Frame with four zeros appended
1 0 0 1 1
1 0 0 1 1
1 0 0 1 1
0 0 0 0 1
0 0 0 0 0
0 0 0 1 1
0 0 0 0 0
0 0 1 1 1
0 0 0 0 0
0 1 1 1 1
0 0 0 0 0
1 1 1 1 0
1 0 0 1 1
1 1 0 1 0
1 0 0 1 1
1 0 0 1 0
1 0 0 1 1
0 0 0 1 0
0 0 0 0 0
1 0 ← Remainder
Transmitted frame: 1 1 0 1 0 1 1 1 1 1 0 0 1 0 ← Frame with four zeros appended minus remainder

CRCs Example



CRCs Example (2)



CRCs Example (3)

Information: (1,1,0,0) $\longrightarrow$ $i(x) = x^3 + x^2$

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

Encoding: $x^3 i(x) = x^6 + x^5$

$$
\begin{array}{r}
x^3 + x^2 + x \\
\hline
x^3 + x + 1 \;)\; x^6 + x^5 \\
x^6 + \quad x^4 + x^3 \\
\hline
x^5 + x^4 + x^3 \\
x^5 + \quad x^3 + x^2 \\
\hline
x^4 + \quad x^2 \\
x^4 + \quad x^2 + x \\
\hline
x
\end{array}
$$

$$
\begin{array}{r}
1110 \\
\hline
1011 \;)\; 1100000 \\
1011 \\
\hline
1110 \\
1011 \\
\hline
1010 \\
1011 \\
\hline
010
\end{array}
$$

Transmitted codeword:

$b(x) = x^6 + x^5 + x$

$\longrightarrow \underline{b} = (1,1,0,0,0,1,0)$

[redacted]

CRCs(7)

- Protection depend on generator
  - —Standard CRC-32 is 10000010011000001000111011011011 1
- Properties:
  - —HD=4, detect up to triple bit errors
  - —Also odd number of errors
  - —And burst of up to k bit of errors
  - —Not vulnerable to systematic errors like checksums

Error detection in Practice

- CRCs are widely used on links
  - —Ethernet, 802.11, ADSL, Cable …
- Checksum used in Internet
  - —IP, TCP, UDP … but it is weak
- Parity
  - —Is little used