ECE 1150: Computer Networks

Data Link Layer – Error & Flow Control

Mai Abdelhakim, PhD

ECE Department

Swanson School of Engineering

University of Pittsburgh

Revisit Data Link Layer Functions

- **Framing**
  - Encapsulate datagram into frame, adding header (e.g. address), trailer
  - Format/encode for physical layer
- **Addressing**
  - MAC address used locally on a link
- **Channel access (if shared medium) (MAC)**
  - Control who uses channel
- **Reliable delivery between adjacent nodes**
  - Error Detection
    - receiver detects presence of errors
  - Error Correction
    - Receiver asks for retransmission if error occurs
  - Flow Control
    - Pacing between adjacent sending and receiving nodes

Error Control

- Errors occur in transmissions
  - **Imperfections** of the physical **medium**
    - Noise from power spikes, outages etc.
  - **Collisions**

- Networks should be designed with:
  - Error prevention
  - Error detection
  - Error correction

Impact of error on throughput

- Let P be the probability that a frame will be corrupted (e.g. due to collisions)
  - ➔1–P is the probability of successful transmission

- Throughput = (1-P) Useful data / (Delay when attempt is successful)

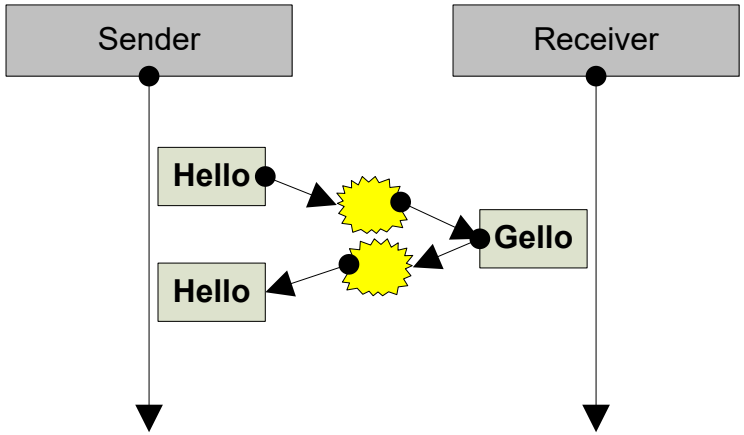
Objectives of the rest of this unit

- Explain the how errors control strategies at the link layer
  - Error detection
  - Error correction
  - Flow control

Slide modified from Agrawal

Error correction in human communication

- Some human error-correction techniques
  - Receiver **reads back** on telephone
    - Credit card number, phone number etc.
      - high overhead, error cancellation
  - **Redundant data**
    - Don't just say tomorrow
      - Say tomorrow, Friday, Oct. 28,



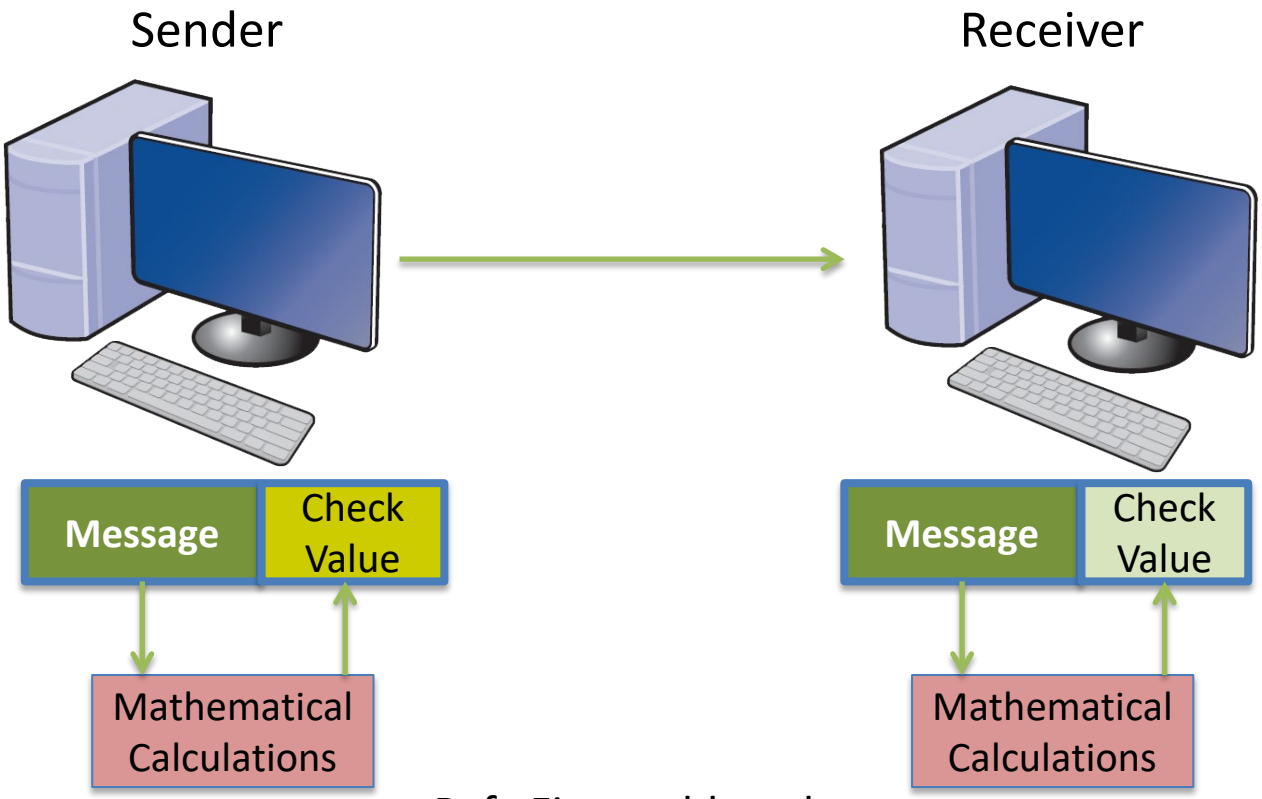
Slide modified from Agrawal

Error correction in computer communication by adding metadata

- The general approach to error detection in data communications is to **add some meta-data to the original data**
  - The meta-data is generated from the data itself
  - The receiver use the meta-data to check if error exists

Error Detection

- Both sender and receiver calculate check value
- Receiver tests whether the check values match



Ref.: Fitzgerald et al.

Q_Metadata

- Can metadata be of any size?

Slide modified from Agrawal

Error Detection– Simple Example for Illustration

- Say we want to send **HELLO**
  - We could code it (in decimal) as: **8 5 12 12 15** (location in the alphabet)
  - Assume we can add only **one digit as meta-data**
    - A simple meta-data would be to add all the digits till you get a single digit
    - 8 + 5 + 12 + 12 + 15 = 52 (two digits) ➔ add them
    - 5 + 2 = **7** (got one digit)
  - Send **8 5 12 12 15** **7** (7 appended to the HELLO)

Slide modified from Agrawal

Error Correction Problems

- If everything goes well, the receiver will get
  - **8 5 12 12 15 7**
    - It knows that the 7 is the "meta-data"
    - It regenerated the meta data from the data:
      - 8 + 5 + 12 + 12 + 15 = 52
      - 5 + 2 = **7 (matches the meta-data appended in the message)**
    - Hence, data is received correct (no error detected)

- If error occurs and the receiver gets: "GELLO": 7 5 12 12 15 7 -> it can detect error
  - 7+5+12+12+15=51, **5+1=6 (not 7)** = > **error**

- But this scheme is **too naïve – errors may go undetected**
  - What if we receive **8 5 11 13 15 7**
  - Or 2 2 11 13 15 7 ➔ sum (without meta-data)= 43 => sum = 7

Error Detection Schemes – Applied on Bits!

- Parity check
- Checksum
- Cyclic redundancy check (CRC)

Error Detection: Parity Check

- Parity check
  - 1-bit check value
  - Based on the number of 1's in the message
  - Two techniques:
    - Even parity: total number of 1's with remains even
    - Odd parity: total number of 1's remains odd
  - Simple, but also some errors may go undetected

Example (Even Parity)

Character: 'A'
Binary: 01000001

Add Parity Bit: 0
(to make total even)

Character: 'C'
Binary: 01000011

Parity Bit = 1
'C' has odd number of ones (3 ones), if we use even parity we add 1 to make the total even (4 instead of 3)

Tophat

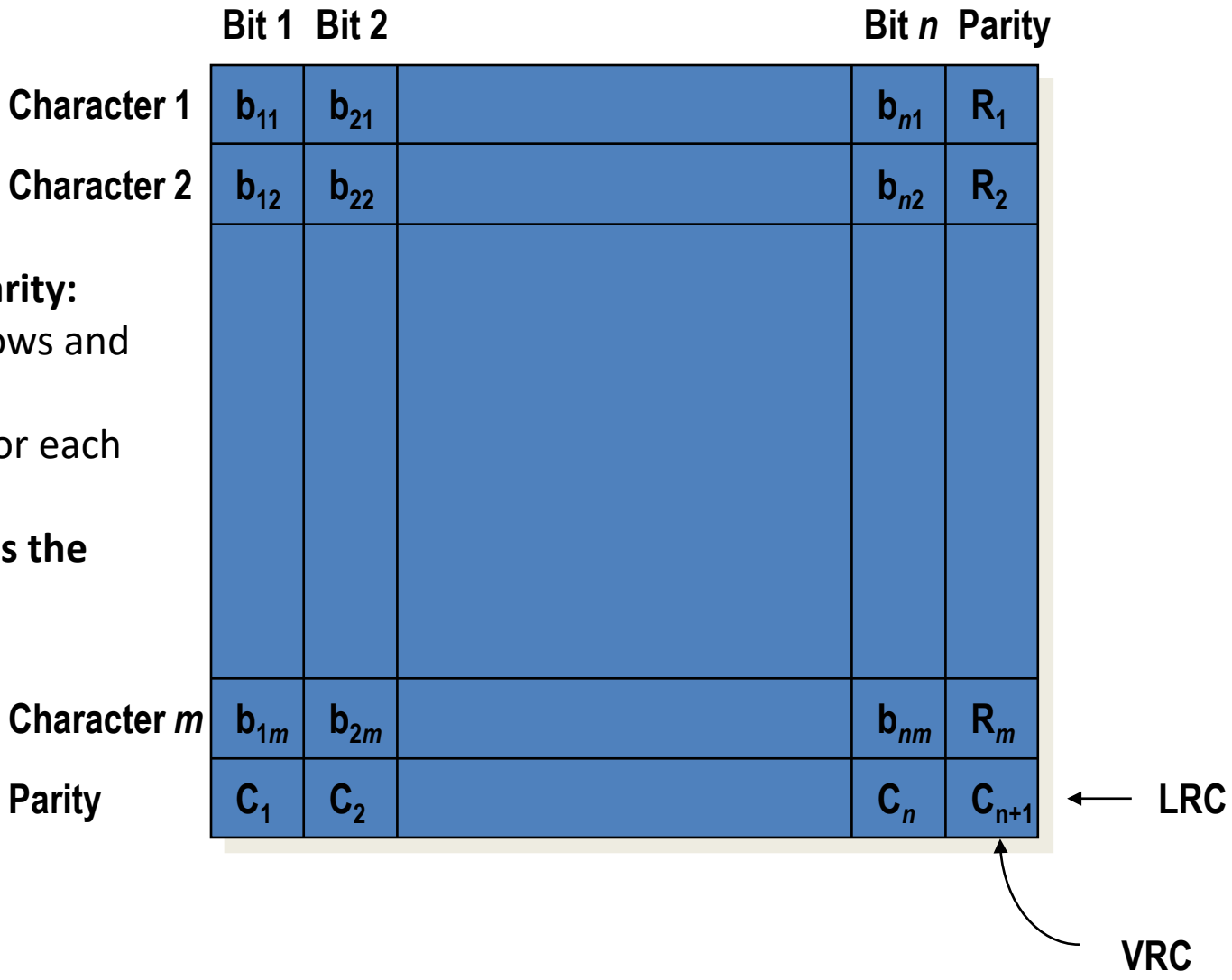
Q_Odd parity

If we are using odd parity, then for the sequence 01000011 we append

A	bit '0'
B	bit '1'

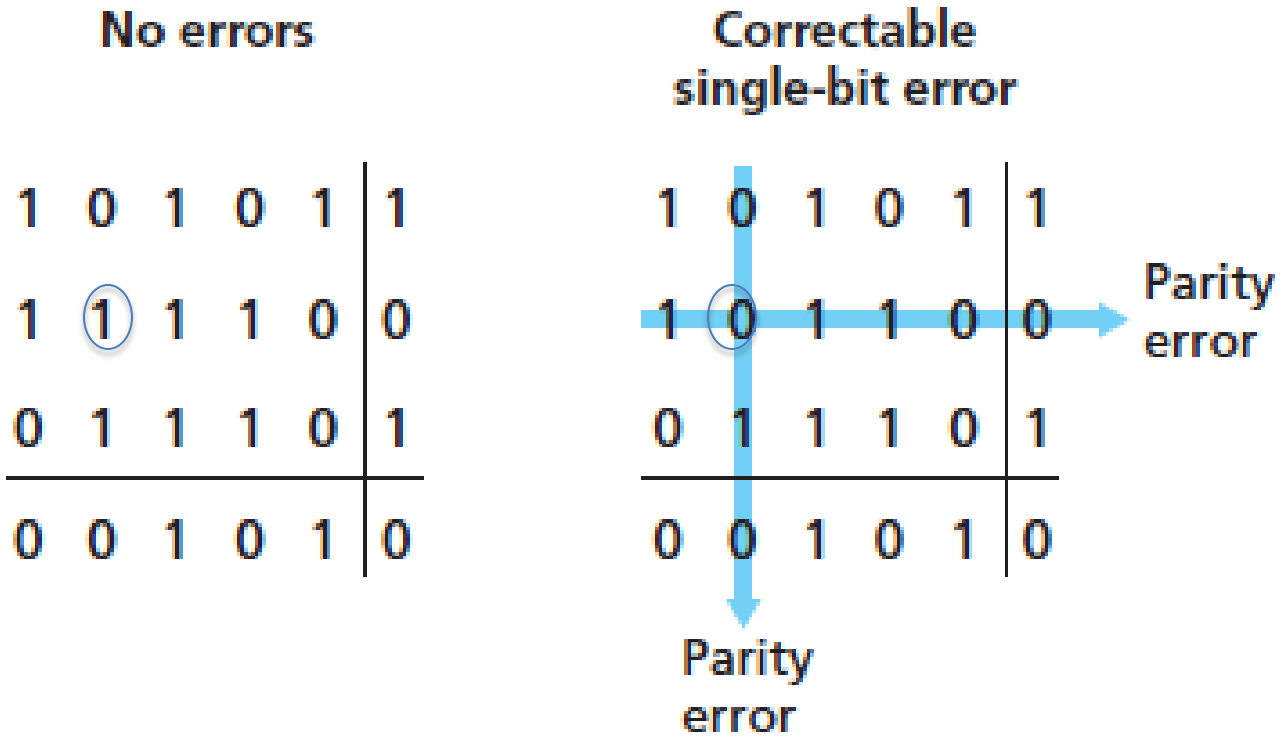
Error Detection: Vertical and Longitudinal Parity Checks

- **Two-dimensional parity:** data is divided into rows and columns
- Parity is computed for each row and column
- **Can identify where is the error**



Error Detection: Vertical and Longitudinal Parity Checks - Example

- Identify errors and correct them
  - Example: Even parity is used for each row and each column



Error Detection: Checksum

- 1-byte (typically) is used as check value
- Checksum algorithms vary in the creation of check values
- Common approach: adds the decimal representation of each character (8 bits), then divide by 256
  - Append the remainder to data

Error Detection Schemes

- Parity check
- Checksum
- Cyclic redundancy check (CRC)

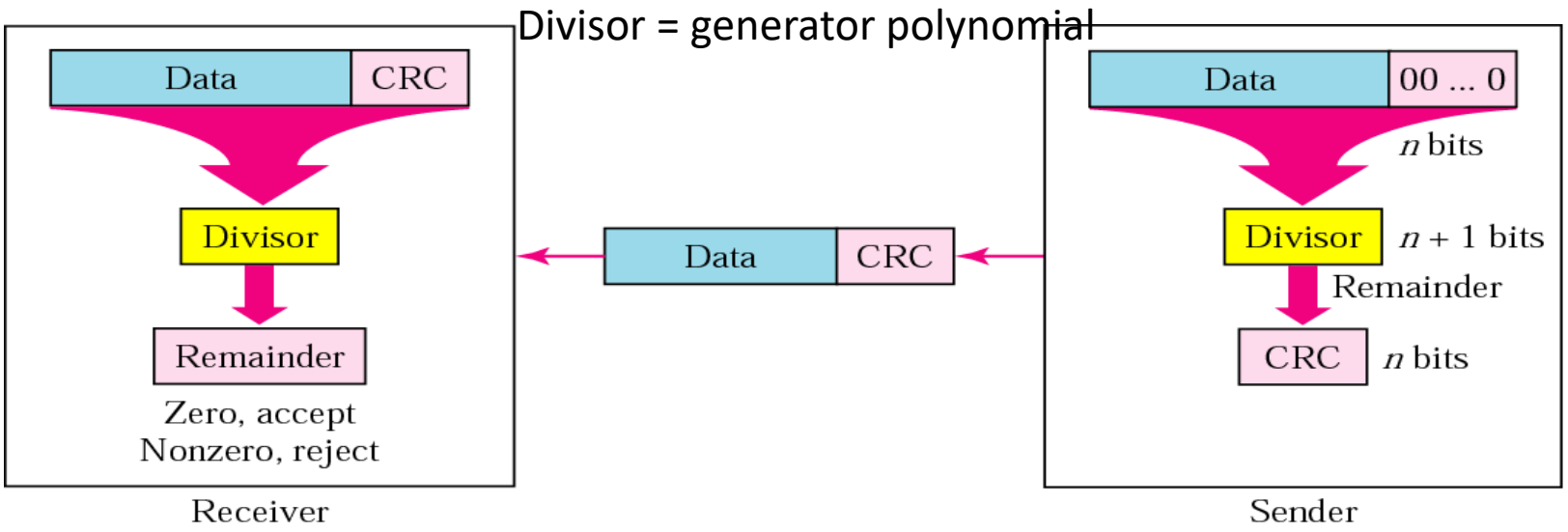
Polynomials

- Each bit sequence can be represented as a polynomial
  - $[b_{n-1} \ldots b_2\, b_1 b_0] \Rightarrow m(x) = b_{n-1}\, x^{n-1} + \ldots + b_1\, x^1 + b_0\, x^0$
  - Note that $x^0 = 1$
  - Example: represent 11000101 in a polynomial
    - 11000101 => $x^7 + x^6 + x^2 + 1$
    - *8 bits* ➔ *Degree of polynomial (largest power of x) is = 7*

- When the generator polynomial g = [1 1 0 1] =>
  - $g(x) = x^3 + x^2 + 1$,
  - Generator polynomial is $g(x)$ of degree $n = 3$

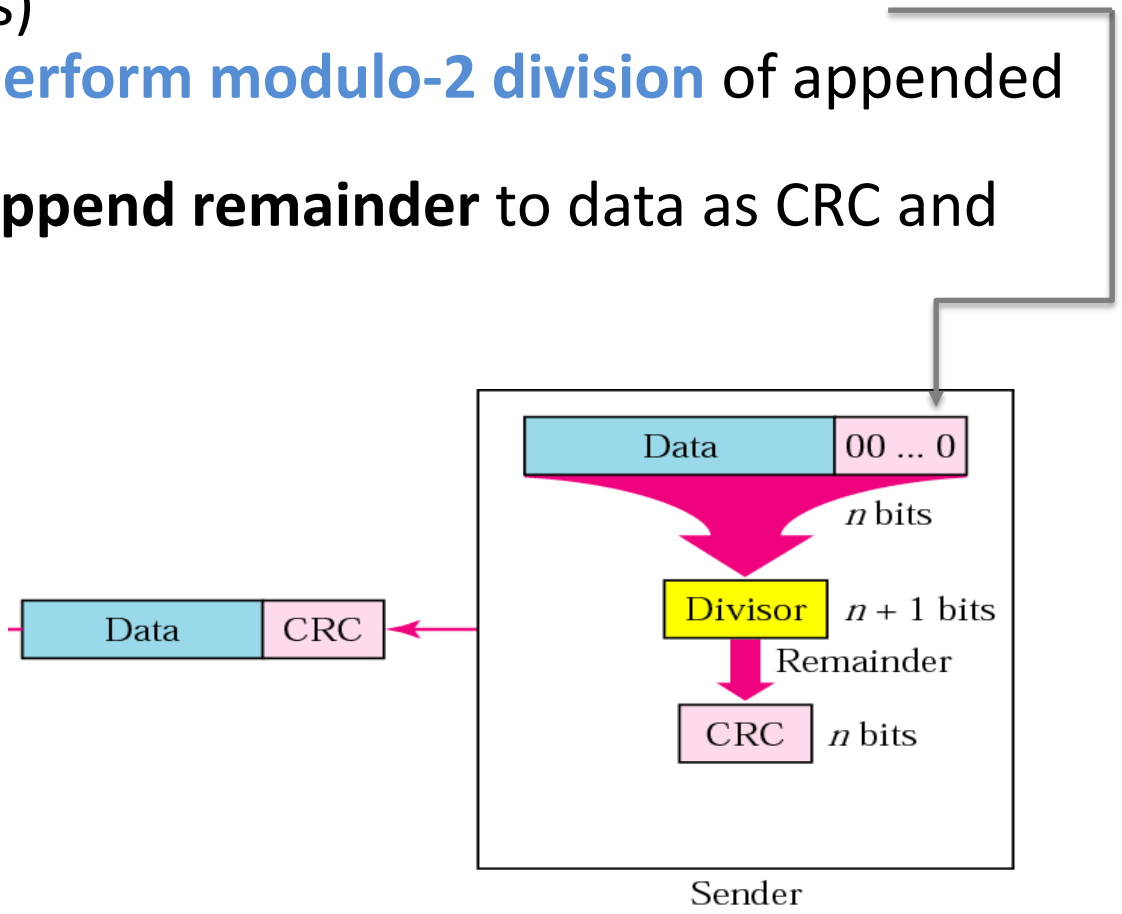
Error Detection: CRC

- Treats message as a single binary number
- Sender and receiver must **agree** on a **pattern** of n+1 bits, known as **generator polynomial or divisor**
  - n is degree of the polynomial
- **CRC** is the **remainder** with data is divided by divisor
- International standards define: **8, 12, 16** and **32 bits** CRC
  - degree of polynomial= number of bits appended



CRC Sender Procedure

- **Step 1:** User data is dividend, technology specifies a **divisor**
- **Step 2:** At sender, **add n zeros** to the **end** of the data (divisor has n+1 bits)
- **Step 3:** At sender, **perform modulo-2 division** of appended data with divisor
- **Step 4:** At sender, **append remainder** to data as CRC and send to receiver



CRC Receiver Procedure

- Step 5: At receiver, perform modulo-2 division of entire frame (data & CRC) with same divisor
  - Divisor is known because it is specified by technology
- Step 6: At receiver, if remainder = 0, accept data (no errors).
  - Else remainder ≠ 0, error detected => reject data

Error Detection: CRC

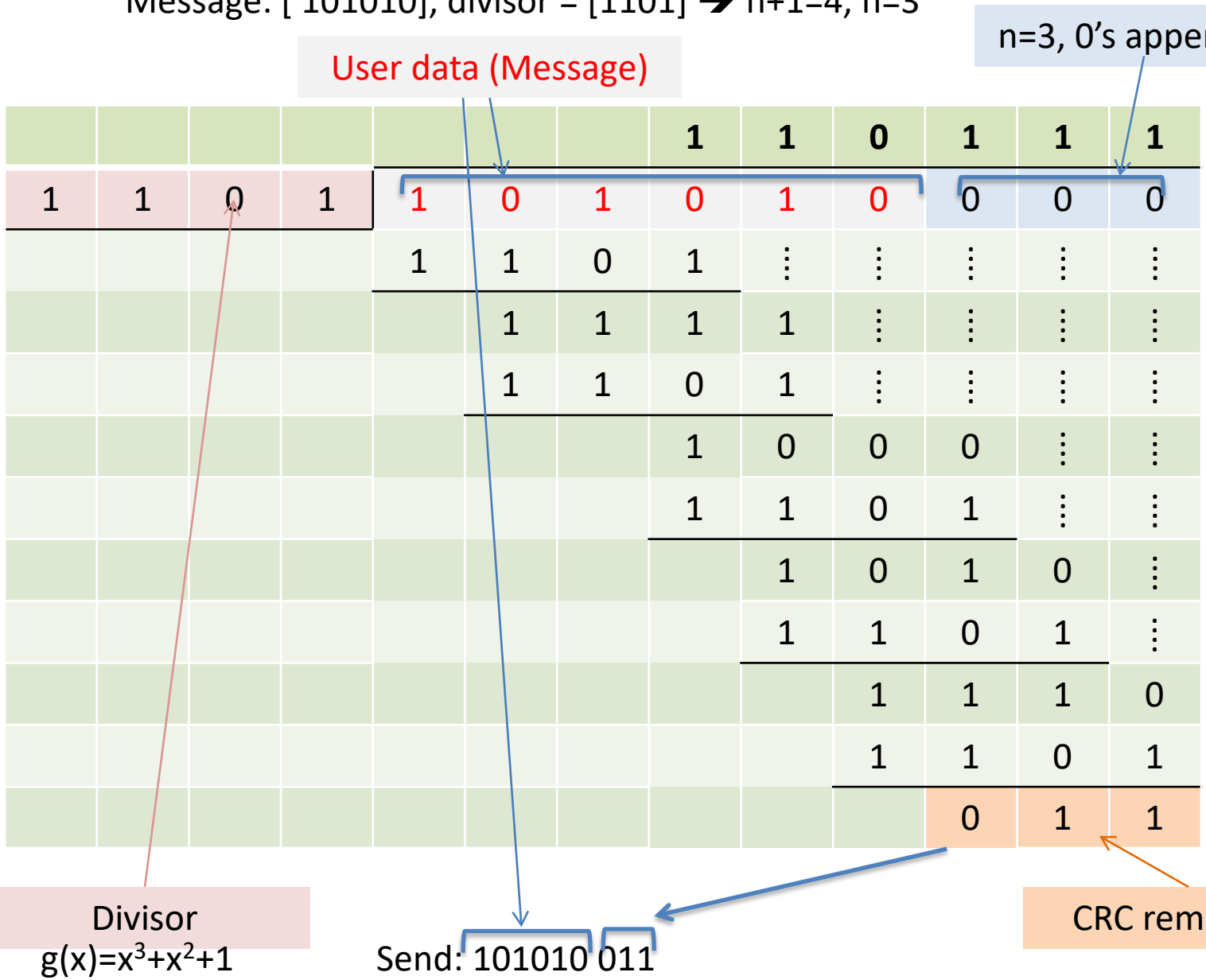
- Divisions in the CRC operations are modulo-2:
  - Like regular long division, however addition and subtraction and equivalent to bitwise XOR (Exclusive OR)
    - 0 – 0 = 0
    - 0 – 1 = 1
    - 1 – 0 = 1
    - 1 – 1 = 0

Example

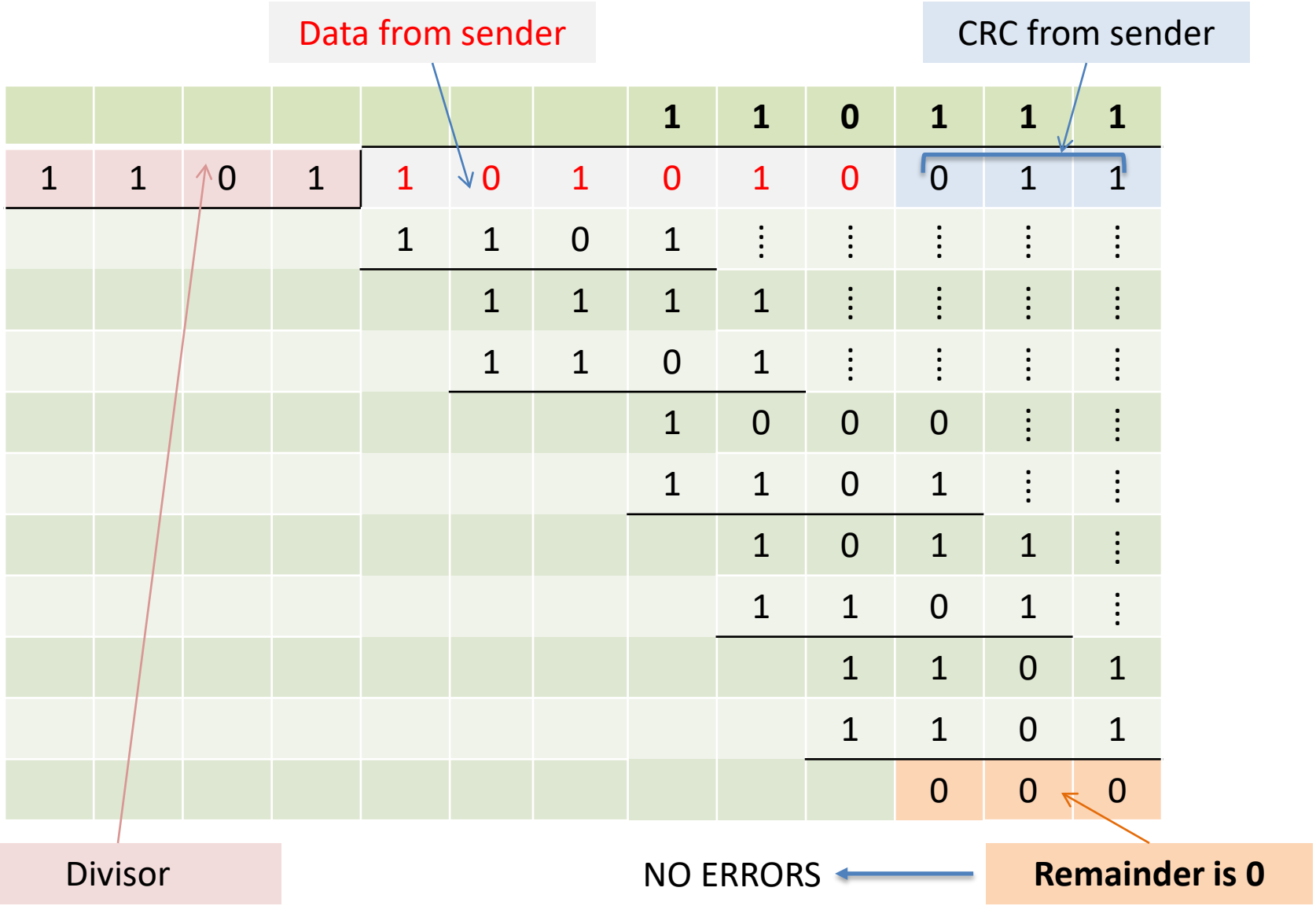
- Example: Message: [101010],
  divisor =g = [1101],
  get CRC bits.

CRC – Sender Operation

Message: [101010], divisor = [1101] ➔ n+1=4, n=3



CRC – Receiver Operation



Another Method: Get CRC using Polynomials

- Generator polynomial: g = [1 1 0 1] => $g(x)=x^3+x^2+1$
  - Degree: n =3
- Append n zeros to the message then get polynomial representation
  - Message|000: [1 0 1 0 1 0 0 0 0] => m(x) = ?
    - OR we can get the polynomial of message then multiply by x^n
- Divide by g(x)
- The remainder is the CRC = x+1 => [0 1 1]
  - It should be n bits, so we add zero at the most significant bit
- Similarly, the received sequence with no errors is:
  [101010 011] => r(x)=?
  - When you divide by g(x) the remainder will be zero. Try it.

Slide modified from Agrawal

Cyclic redundancy check (CRC)

- Most commercial data communication technologies use CRC
- CRC-32 is used in Ethernet
  - CRC-32: generator polynomial of degree 32. This also means that 32 CRC bits will be appended to data
- The generator polynomial is

$$g(x) = 1 + x + x^2 + x^4 + x^5 + x^7 + x^8 + x^{10} + x^{11} + x^{12} + x^{16} + x^{22} + x^{23} + x^{26} + x^{32}$$

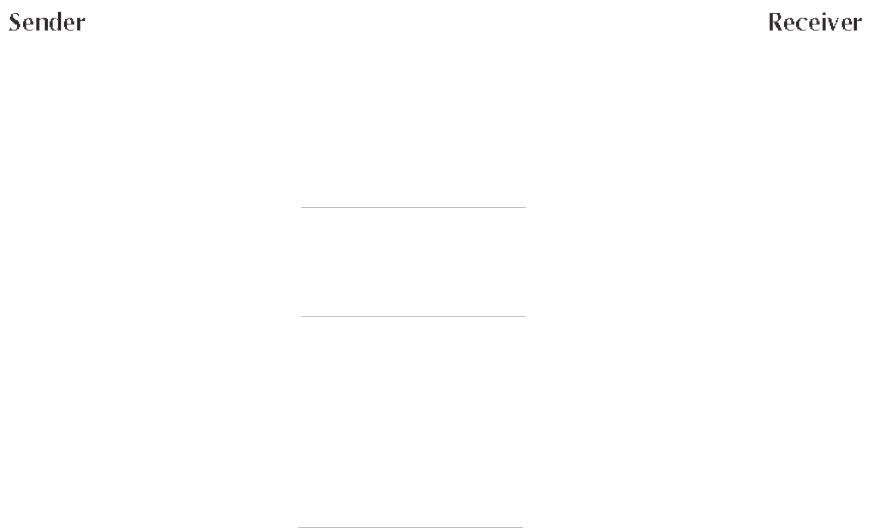
See also: http://www.xilinx.com/support/documentation/application_notes/xapp209.pdf

Error Correction

- Once detected, errors must be corrected
- Error correction techniques
  - **Retransmission** (or backward error correction)
    - Retransmission is simple and most common
    - **Automatic Repeat reQuest (ARQ)**
      - **Stop-and-wait ARQ**
      - **Continuous ARQ**
  - Forward error correction
    - Receiving device can correct messages without retransmission.. Instead use channel coding techniques (think of repetition code)

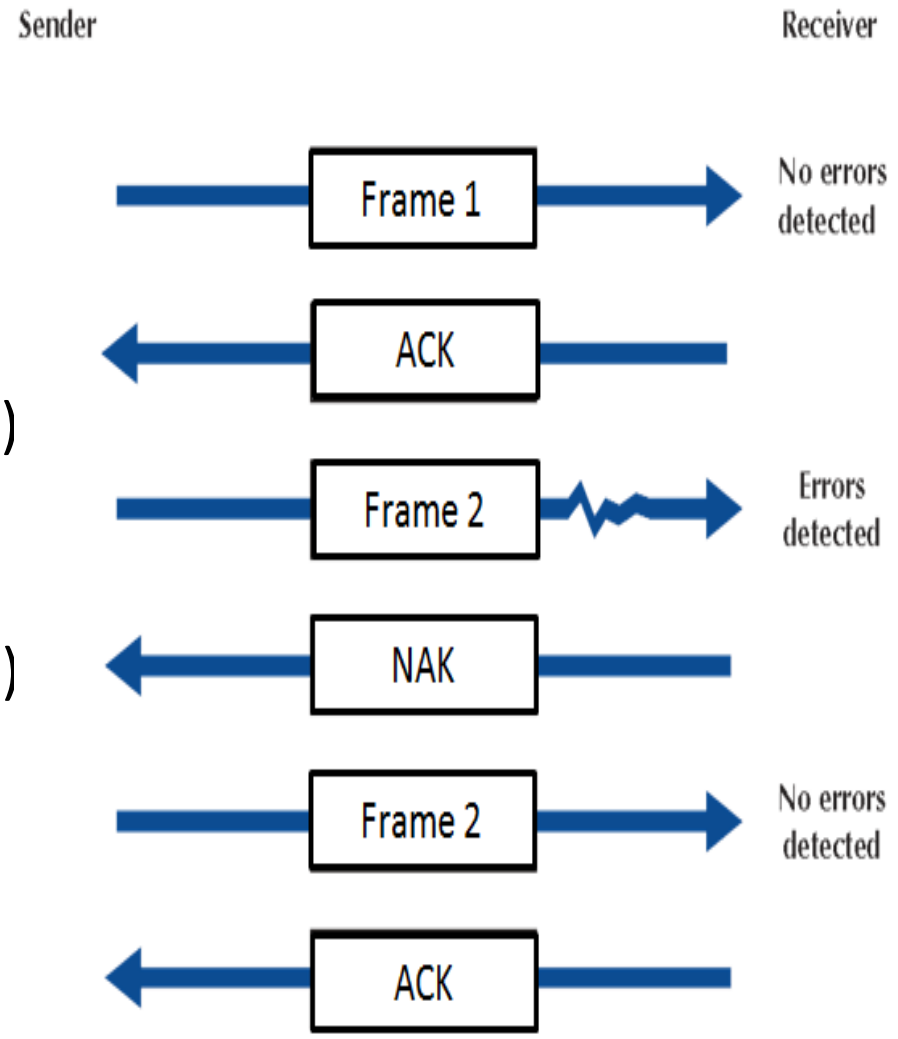
Error Correction: Stop-and-Wait ARQ

- Stop-and-wait ARQ
  1. Receiver receives frame and sends:
     - Acknowledgement (ACK) <u>if no error</u>
     - Negative acknowledgement (NAK) <u>if error</u>
  2. If NAK, sender re-sends data



Error Correction: Stop-and-Wait ARQ

- Stop-and-wait ARQ
  1. Receiver receives frame and sends:
     - Acknowledgement (ACK) <u>if no error</u>
     - Negative acknowledgement (NAK) <u>if error</u>
  2. If NAK, sender re-sends data

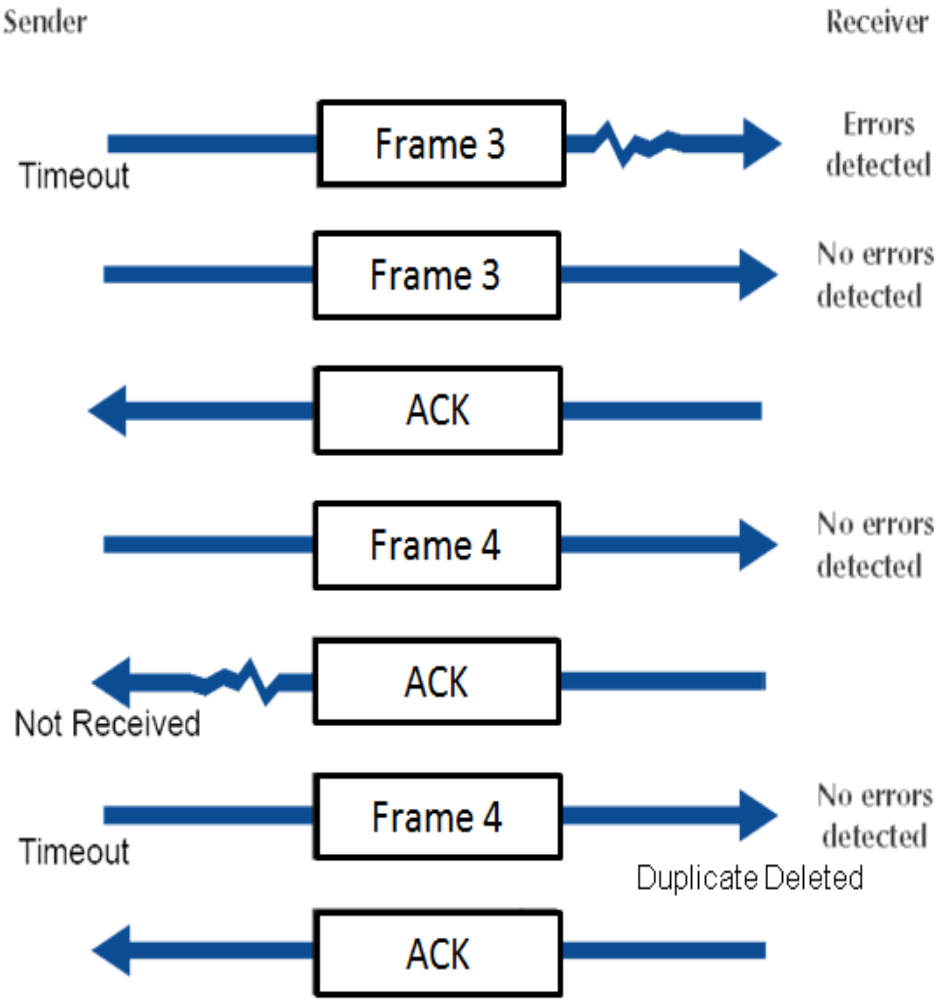


Error Correction: Stop-and-Wait ARQ

- Stop-and-wait ARQ
  3. If no ACK or NAK, Sender retransmits frame after “timeout”
  4. If no ACK, sender re-sends data, receiver sends ACK and deletes duplicate frame

Error Correction: Stop-and-Wait ARQ

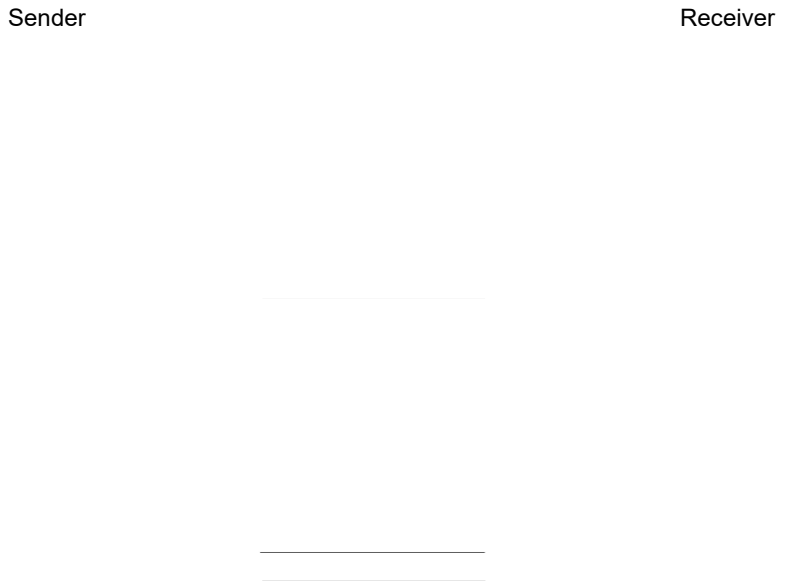
- Stop-and-wait ARQ
  3. If no ACK or NAK, Sender retransmits frame after "timeout"
  4. If no ACK, sender re-sends data, receiver sends ACK and deletes duplicate frame



Error Correction

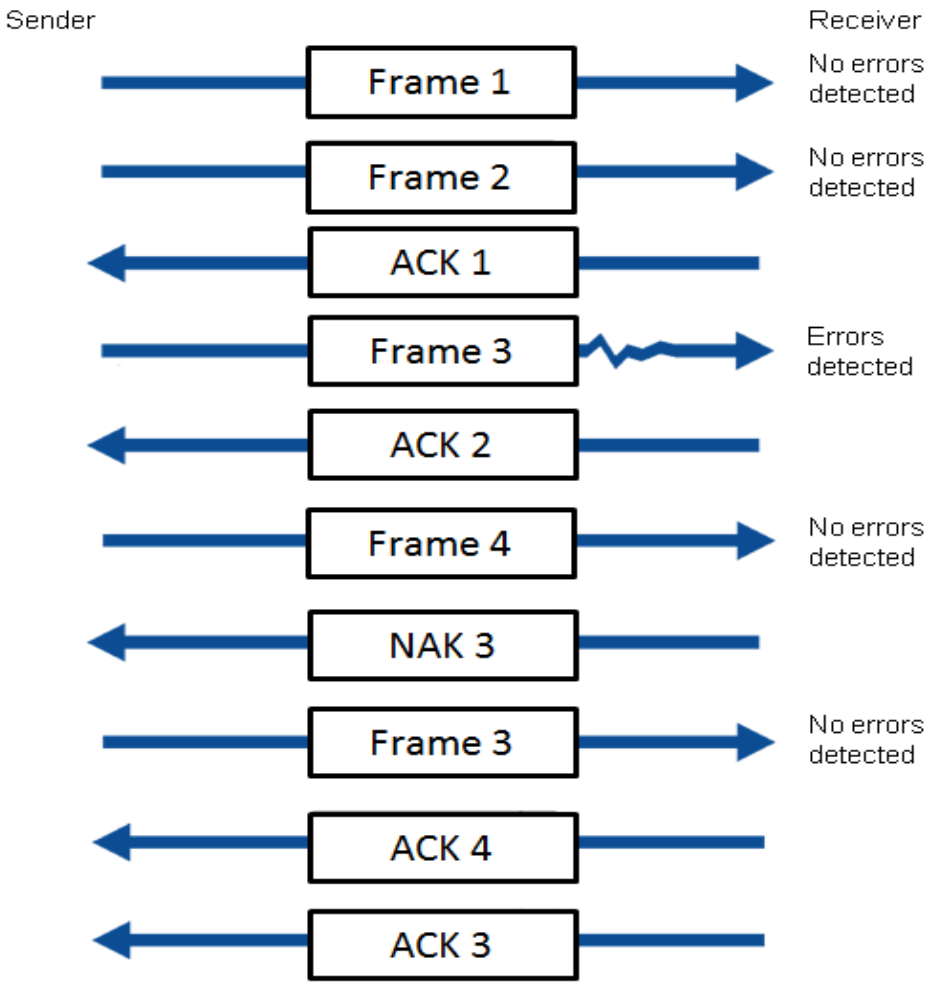
Here, window size is 2

- Continuous ARQ
  - Sender does not wait for ACKs before sending more data
  - "**Sliding window**" is number of frames allowed to be unacknowledged by receiver
    - We send more than one frames before receiving an ACK
  - Number of frames that can be sent depends on window size
    - Window size is **agreed upon by sender and receiver at the beginning**



Error Correction: Continuous ARQ

- Continuous ARQ
  - Sender does not wait for ACKs before sending more data
  - “Sliding window” is number of frames allowed to be unacknowledged by receiver
  - Agreed upon by sender and receiver at the beginning



Here, window size is 2

Tophat

Q_Throughput of ARQ

We expect that throughput using stop-and-wait ARQ

A	is higher than that can be obtained from continuous ARQ
B	is lower than that can be obtained from continuous ARQ

- ARQ is useful in flow control by limiting the number of packets received

Sliding Window

- If an error occurs
  - Transmitter resends everything since the error (Go-Back-N)
  - Only the packet that has errors (Selective Retransmission)

Sliding Window (Go-Back-N)

Key Takeaways

- Important function at Data Link layer is error control
- Error detection
  - Append extra bits for error detection
  - Parity, checksum, CRC
  - CRC is commonly used and is more robust
- Error Correction through retransmission
  - Stop and wait ARQ
  - Continuous ARQ
    - Define sliding window, which is maximum number of frames that can be transmitted before an ACK is received
    - Implement flow control by defining limit on the window size