Introduction to Computer Networks

Framing and Error Handling

CS640

https://pages.cs.wisc.edu/~mgliu/CS640/S26/index.html

Ming Liu
mgliu@cs.wisc.edu

Outline

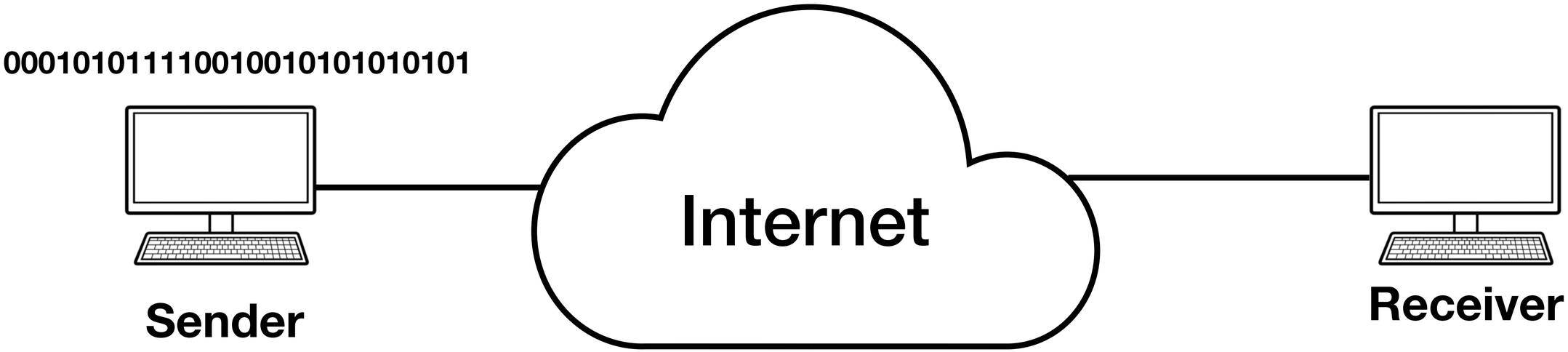
- Last
  - Encoding
- Today
  - Framing and Error Handling
- Announcements
  - Lab1 due on Feb 10th
  - No class this Thursday

Recap

- Key Questions
  - What are the requirements of a communication link?
  - How can we represent bits on the link?
  - How can we reliably propagate bits across the link?

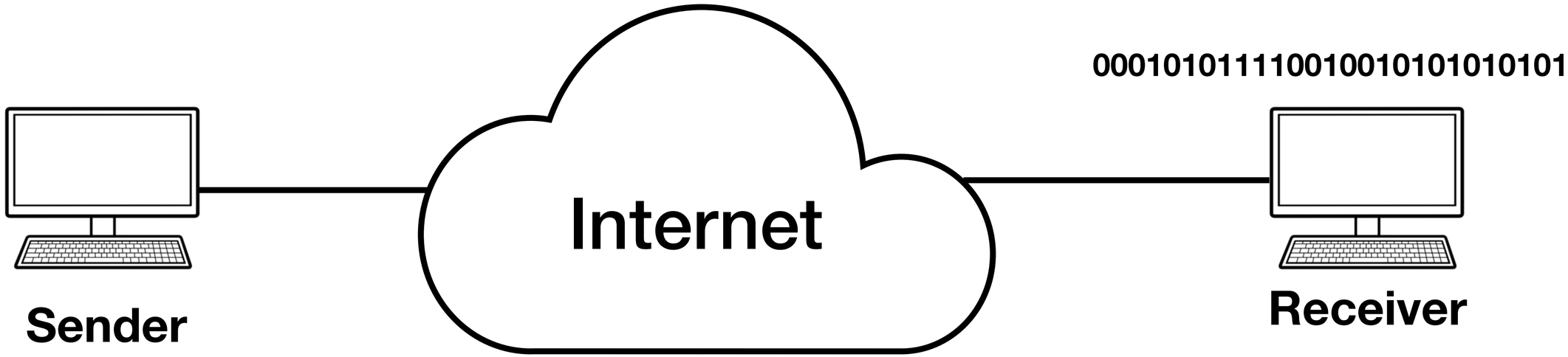
- Terminology
  - Signal
  - NRZ, NRZI, Manchester, and 4B/5B
  - Baud rate

Reliable Bitstream Transmission @Physical Layer

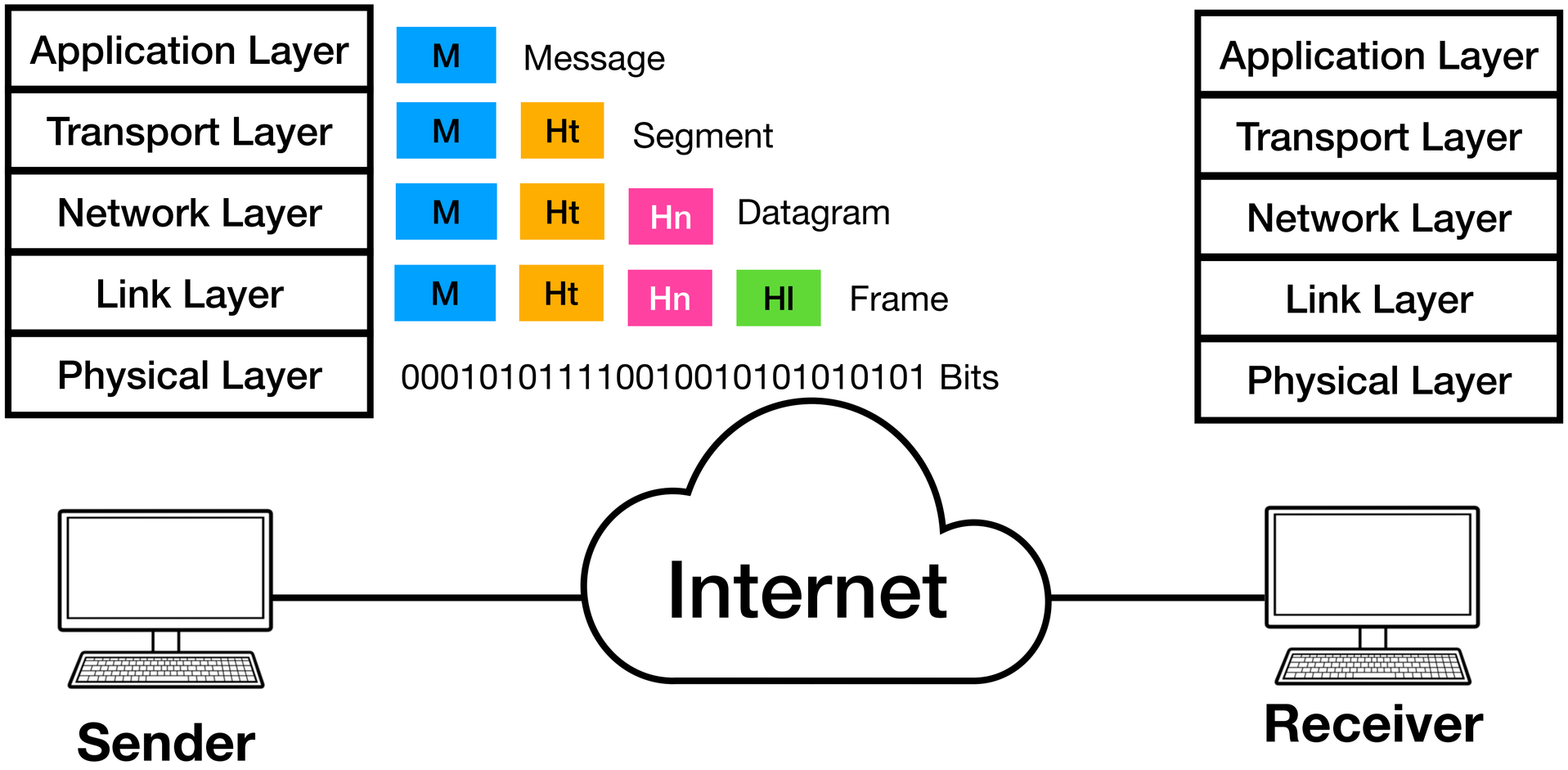


Recap: Data as **Packet**

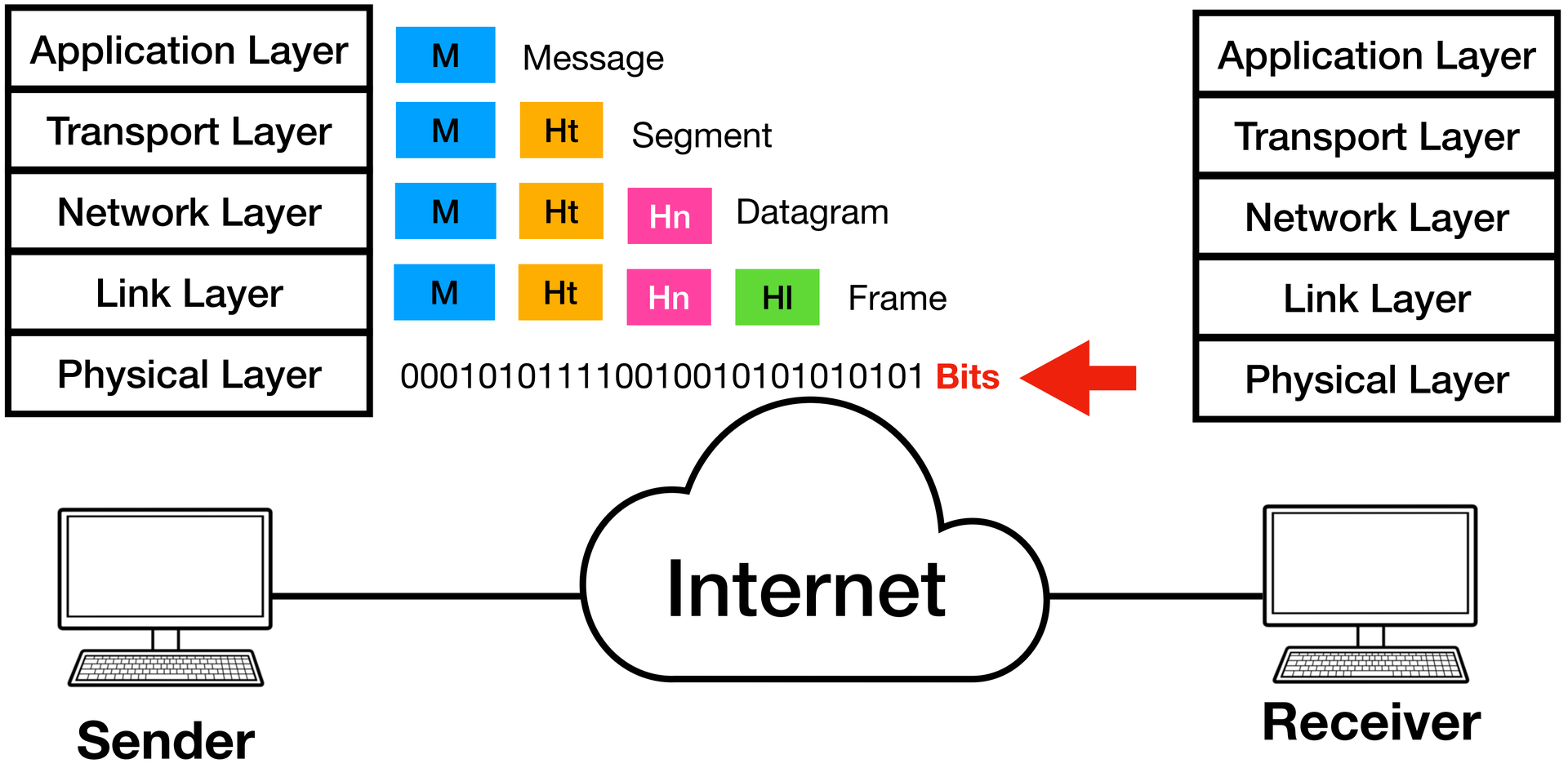
- A packet is the smallest unit of data that traverses the network
  - Consist of header and payload
  - The sender divides data and encapsulates them as packets
  - The receiver decapsulates packets and rebuilds the data



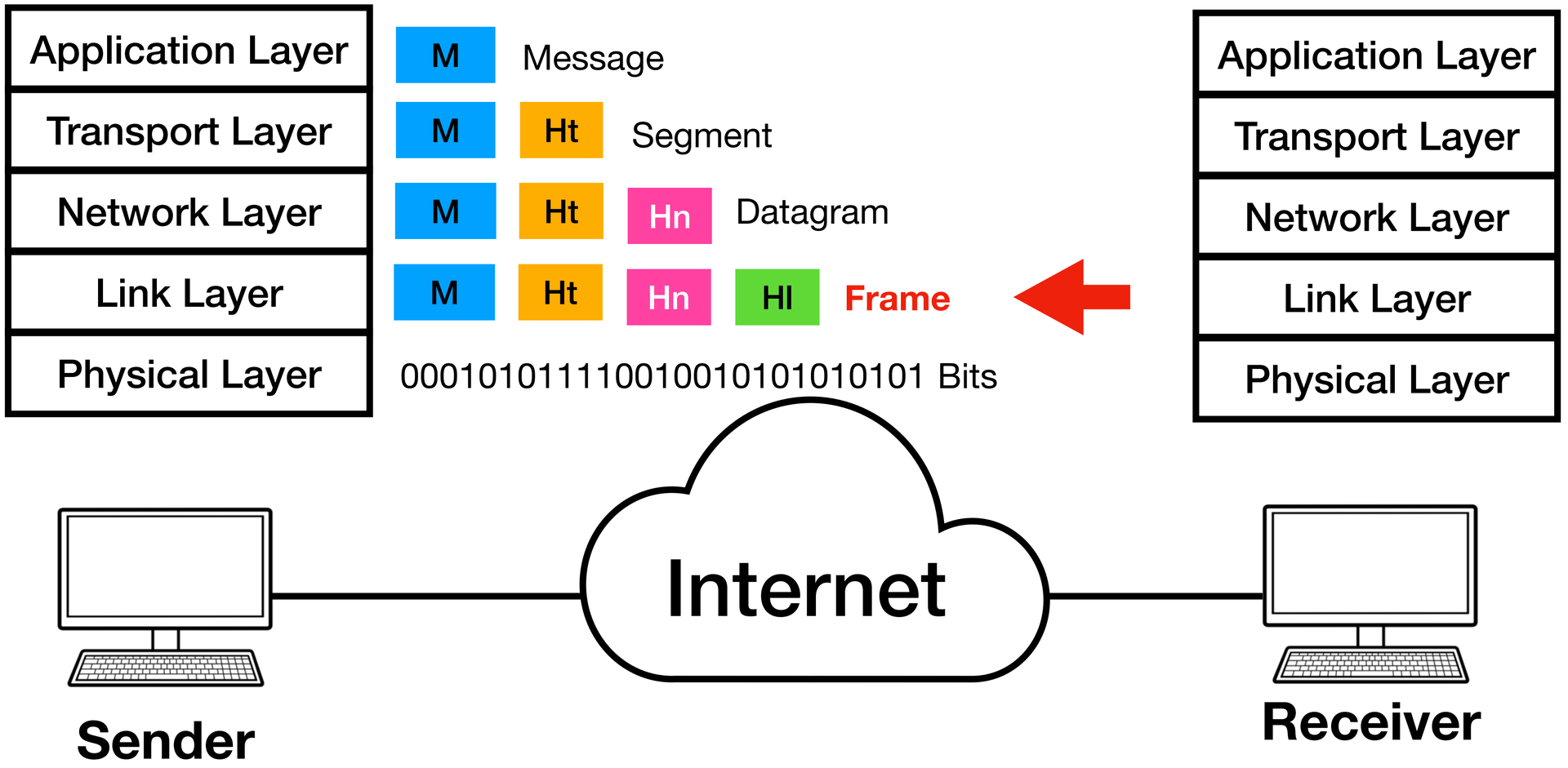
Recap: Data Encapsulation across Layers



Recap: Data Encapsulation across Layers



Recap: Data Encapsulation across Layers



How can we identify a frame from bit streams?

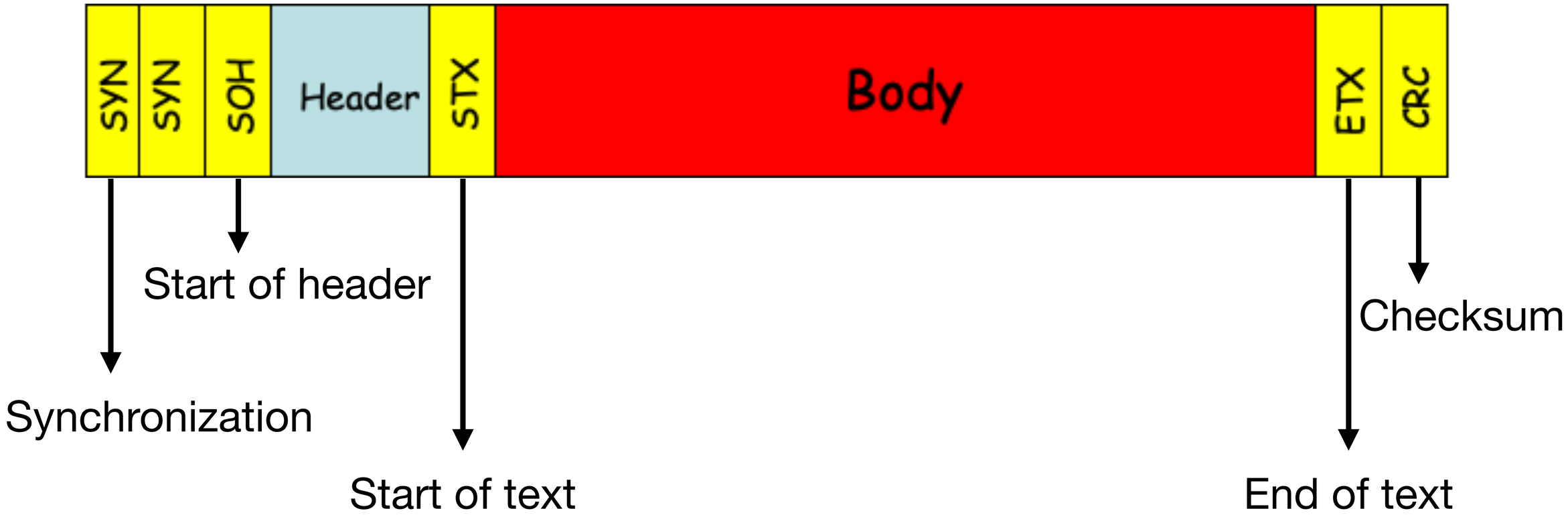
Why is it hard?

- One should correctly identify the start and end of a frame
  - Easily be hidden from data bits

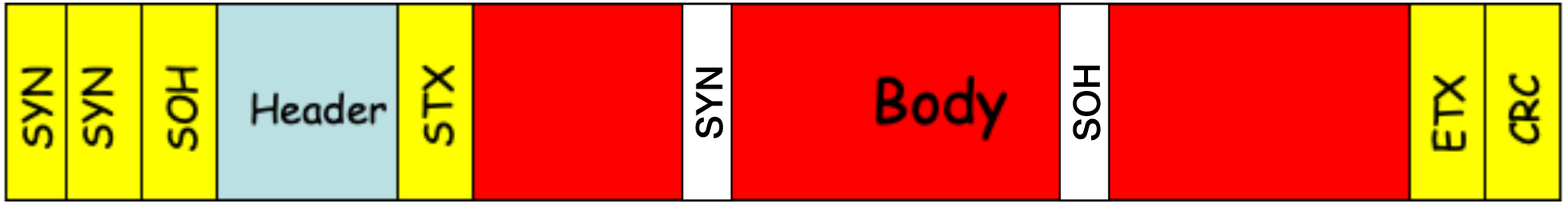
- A frame has a variable length
  - Only the maximum length is fixed

Technique #1: Byte Stuffing

- Mark the frame with special characters
  - Used by the Binary Synchronous Communication protocol (BISYNC)
  - IBM invented it in 1967 for its mainframes, e.g., System/360

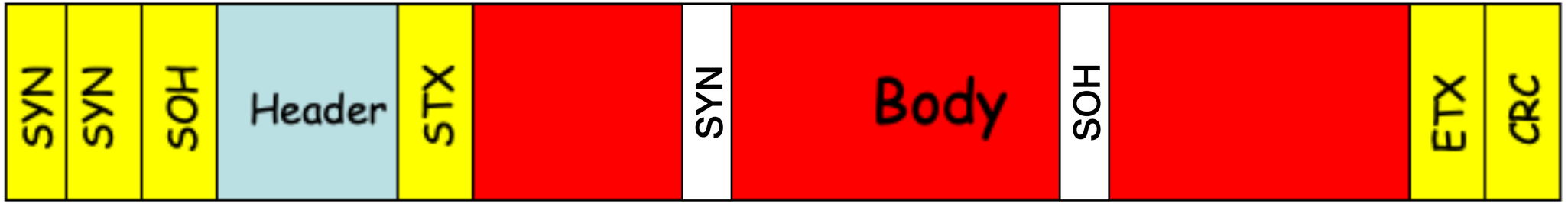


Can we send this data?



Handle Special Characters

- Use an escape character (DLE)
  - SYN = DLE + SYN

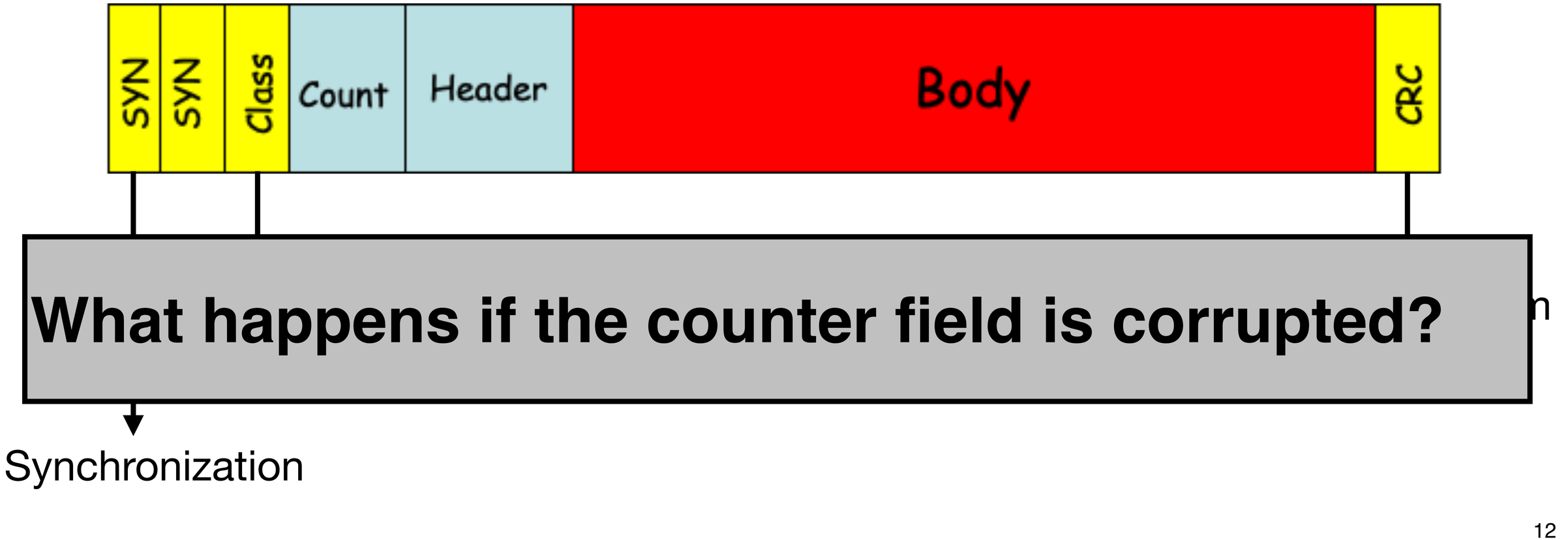


Technique #2: Byte Counting

- Encode the number of bytes into a frame
  - Used by the Digital Data Communication Message Protocol (DDCMP)
  - DEC invented it in 1974 for its DECnet

Technique #2: Byte Counting

- Encode the number of bytes into a frame
  - Used by the Digital Data Communication Message Protocol (DDCMP)
  - DEC invented it in 1974 for its DECnet

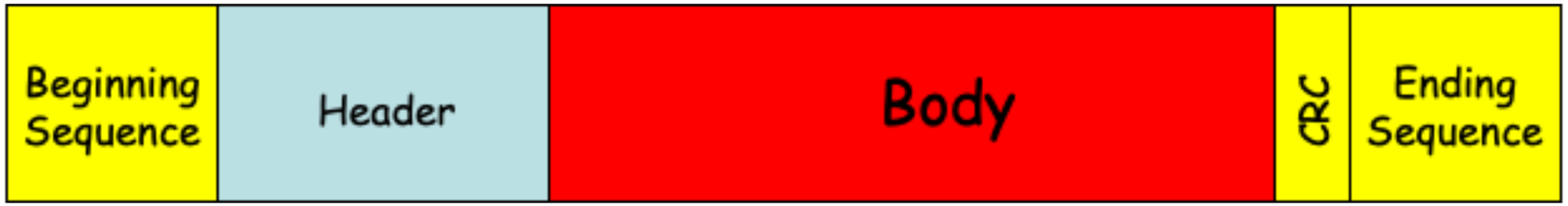


The Receiver Needs to Check Frame Integrity

- Check the checksum field first

Technique #3: Bit Stuffing

- Mark the frame with special bit sequences
  - Used by the High-Level Data Link Control protocol (HDLC)
  - IBM invented it in the 1970s for its system network architecture



How Bit Stuffing Works

- The sender and receiver agree on a special flag
  - For example, **01111110**

How Bit Stuffing Works

- The sender and receiver agree on a special flag
  - For example, **01111110**

- Sender Logic: stuff the bitstream
  - Insert a **0** before transmitting the next bit if transmitting 5 consecutive **1**s

How Bit Stuffing Works

- The sender and receiver agree on a special flag
  - For example, **01111110**
- Sender Logic: stuff the bitstream
  - Insert a **0** before transmitting the next bit if transmitting 5 consecutive **1**s
- Receiver Logic: unstuff the bitstream
  - When receiving 5 consecutive **1**s,
  - If the next bit is **1**, this is the frame marker or an error
  - If the next bit is **0**, remove it

An Exercise: Encoding + Framing

A sender and receiver use 4B/5B encoding and bit stuffing in their networking stacks. Suppose (a) the maximum frame length is 512 bytes; (b) the agreed bit sequence is 01111110; (c) we use high and low signals. Please show how signals traverse across the wire when the sender sends an “OK?” byte stream to the receiver.

O —> 0x4F —> 01001111
K —> 0x4B —> 01001011
? —> 0x3F —> 00111111

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

An Exercise: Encoding + Framing

A sender and receiver use 4B/5B encoding and bit stuffing in their networking stacks. Suppose (a) the maximum frame length is 512 bytes; (b) the agreed bit sequence is 01111110; (c) we use high and low signals. Please show how signals traverse across the wire when the sender sends an “OK?” byte stream to the receiver.

O —> 0x4F —> 01001111
K —> 0x4B —> 01001011

Data	Code	Data	Code
0000	11110	1000	10010
0001	01001	1001	10011
0010	10100	1010	10110
0111	01111	1111	11101

When the number of bits is less than 4, we should pad the data with leading zeros to complete a 4-bit block.

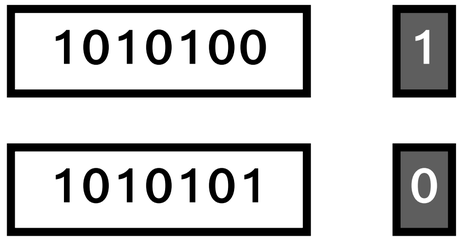
How do we handle frame errors?

How do we handle frame errors?

Data redundancy!

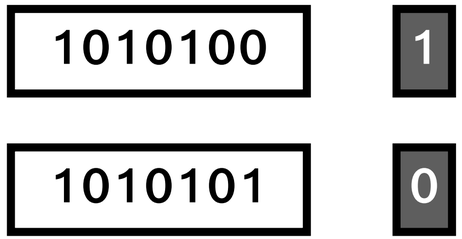
Technique #1: Parity Bit

- Even parity
  - Append a parity bit to 7 bits of data to make an even number of 1’s
  - Odd parity is defined accordingly

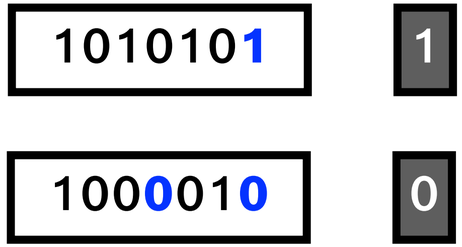


Technique #1: Parity Bit

- Even parity
  - Append a parity bit to 7 bits of data to make an even number of 1’s
  - Odd parity is defined accordingly



- Limitations
  - 1 in 8 bits of overheads
  - Can only detect a single error



Technique #2: 2-D Parity

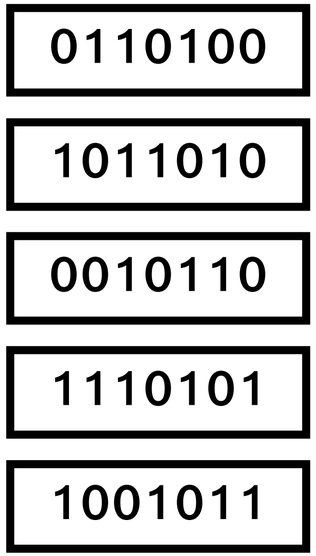
- Apply parity bit across two dimensions
  - Take the even parity as an example

Bitstream of a frame: **01101001011010001011011101011001011**

How 2-D Parity Works

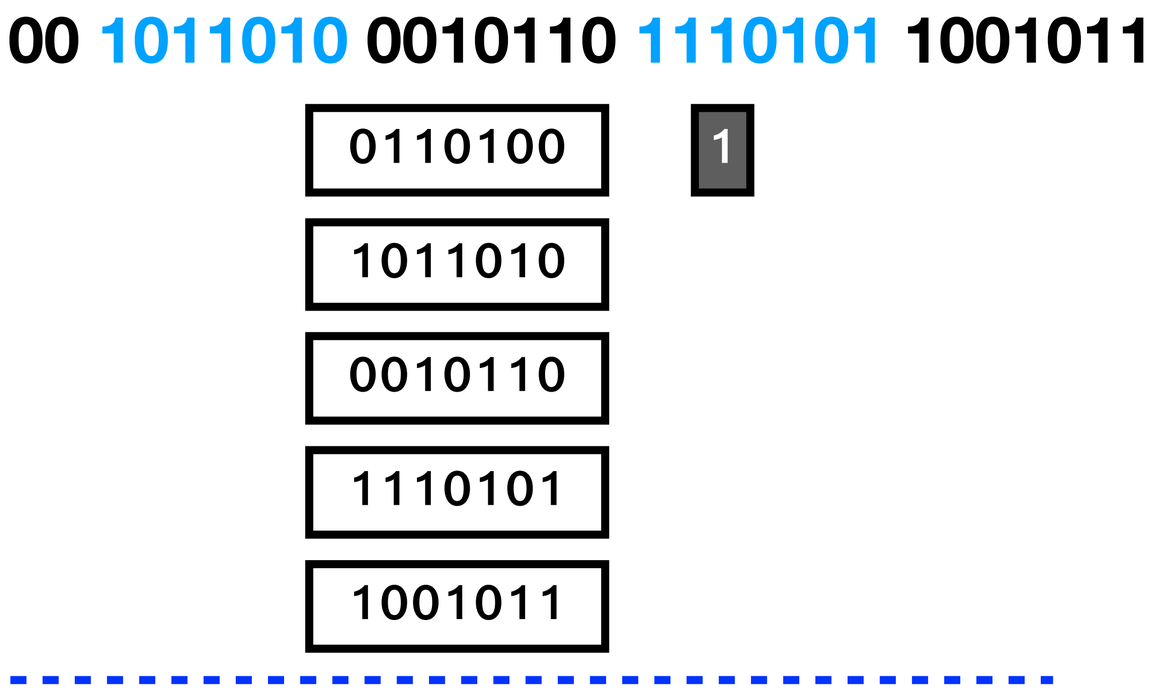
- Step 1: divide the bitstream into a sequence of 7-bit blocks

0110100 1011010 0010110 1110101 1001011



How 2-D Parity Works

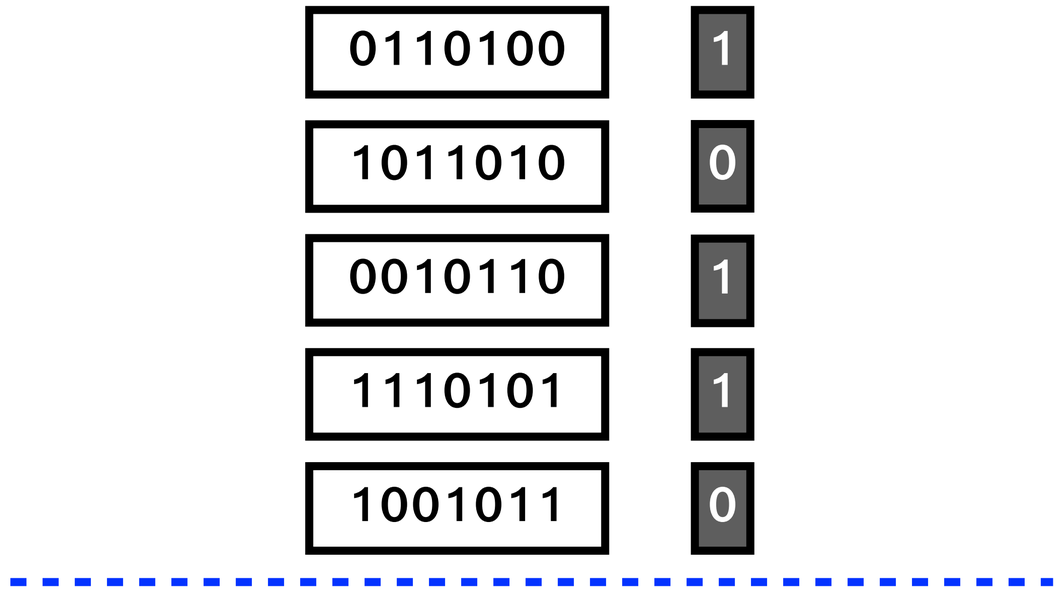
- Step 1: divide the bitstream into a sequence of 7-bit blocks
- Step 2: make each byte even parity



How 2-D Parity Works

- Step 1: divide the bitstream into a sequence of 7-bit blocks
- Step 2: make each byte even parity

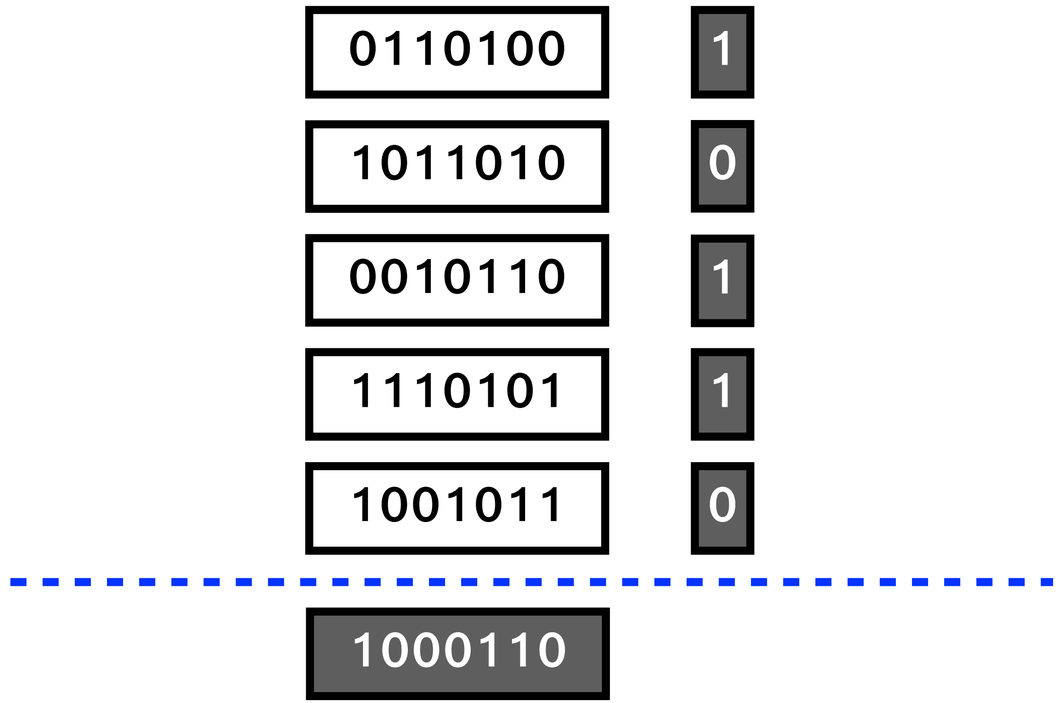
0110100 1011010 0010110 1110101 1001011



How 2-D Parity Works

- Step 1: divide the bitstream into a sequence of 7-bit blocks
- Step 2: make each byte even parity
- Step 3: add a parity byte for all bytes of the packet

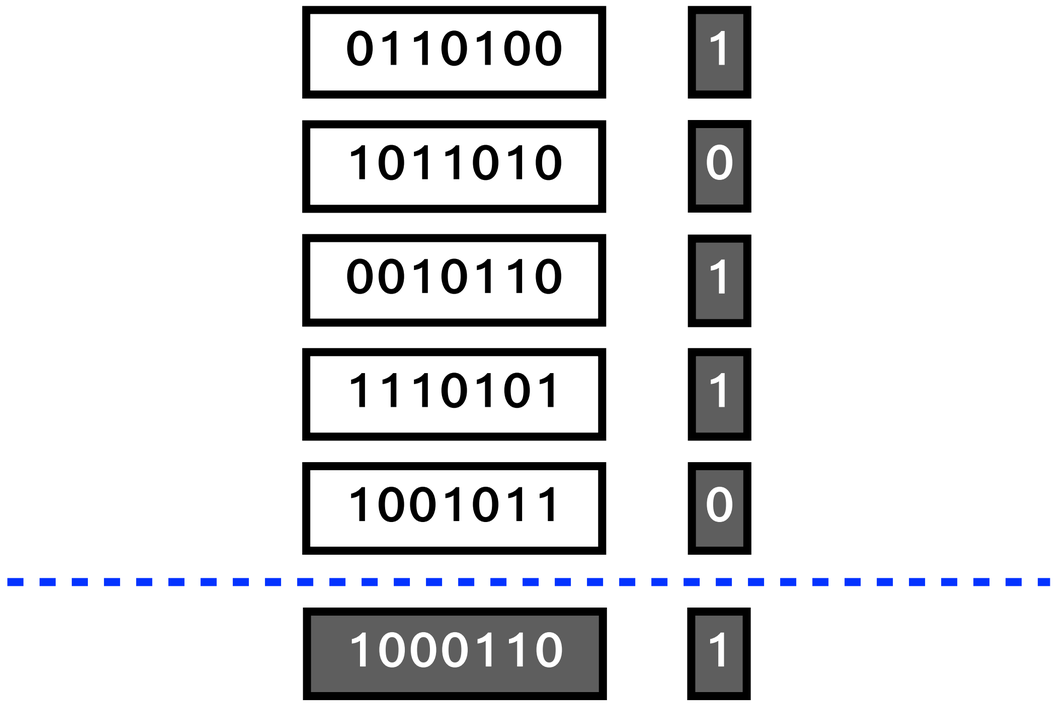
0110100 1011010 0010110 1110101 1001011



How 2-D Parity Works

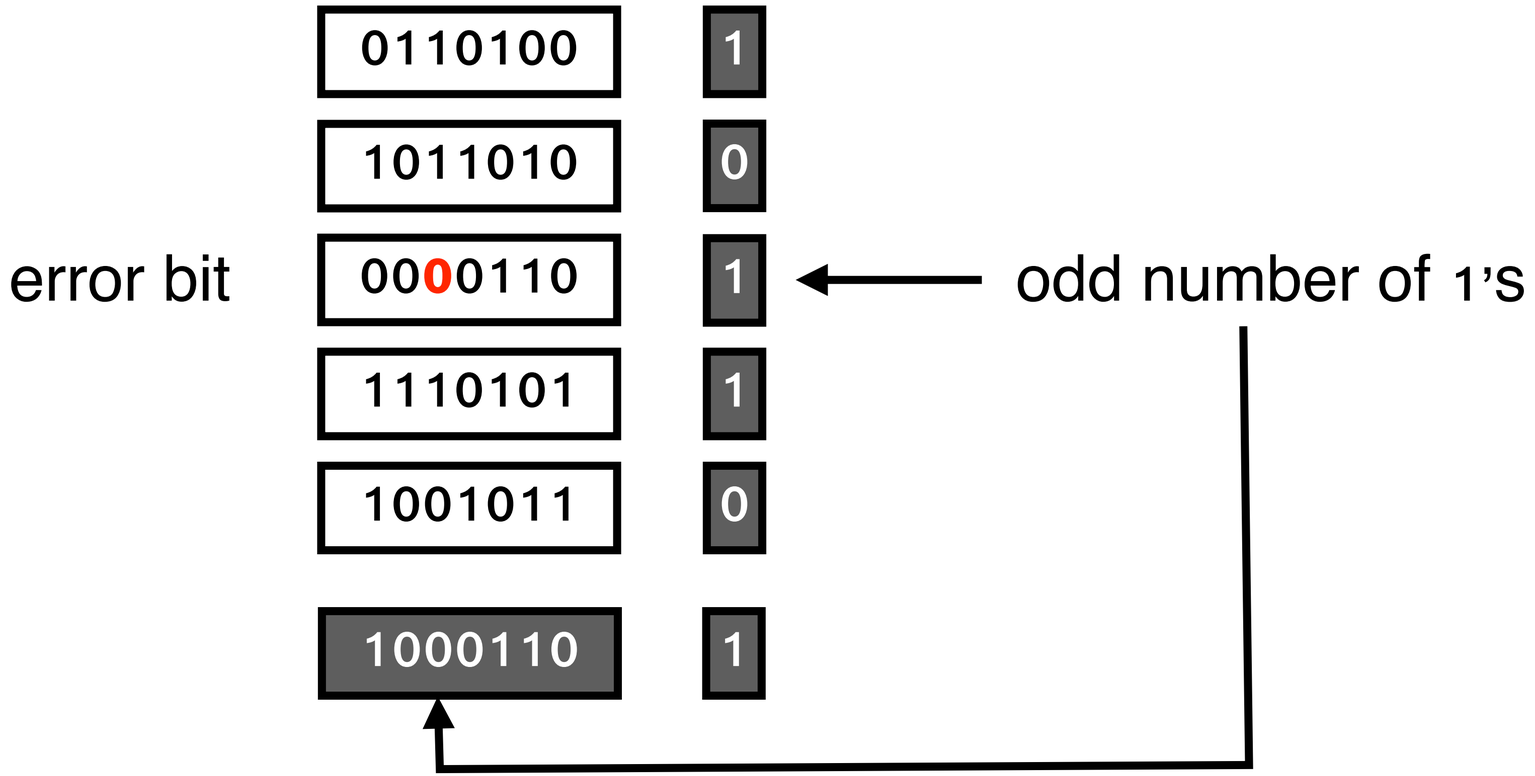
- Step 1: divide the bitstream into a sequence of 7-bit blocks
- Step 2: make each byte even parity
- Step 3: add a parity byte for all bytes of the packet

0110100 1011010 0010110 1110101 1001011



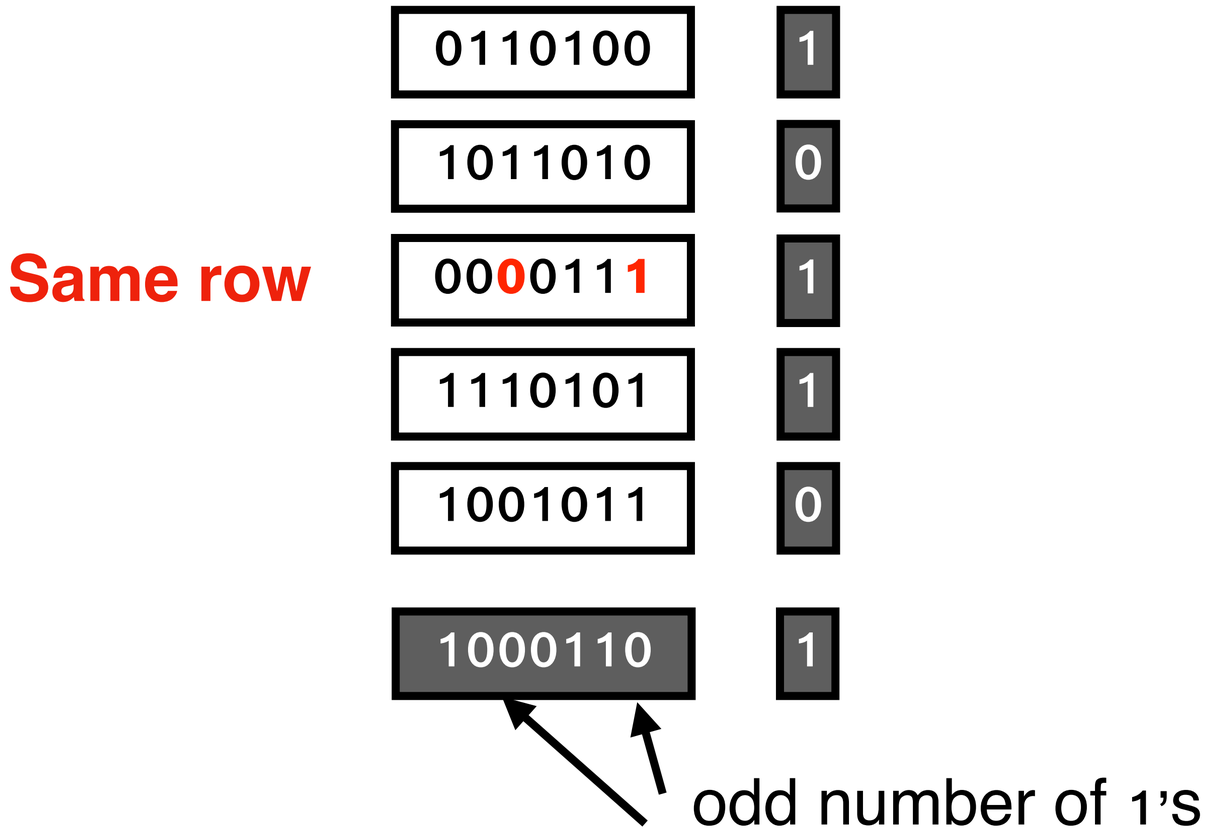
2-D Parity Capabilities

- An 1-bit error can be detected and corrected



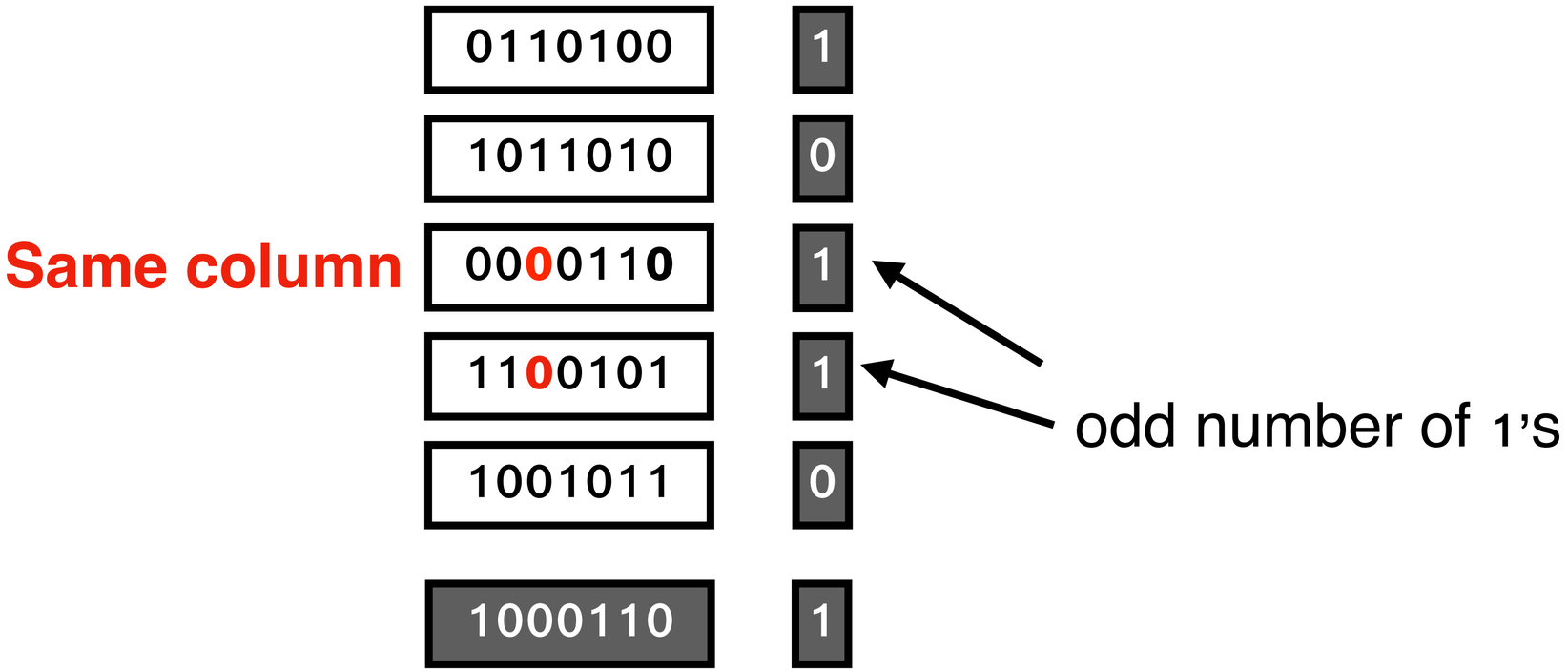
2-D Parity Capabilities

- An 1-bit error can be detected and corrected
- A 2-bit error can be detected



2-D Parity Capabilities

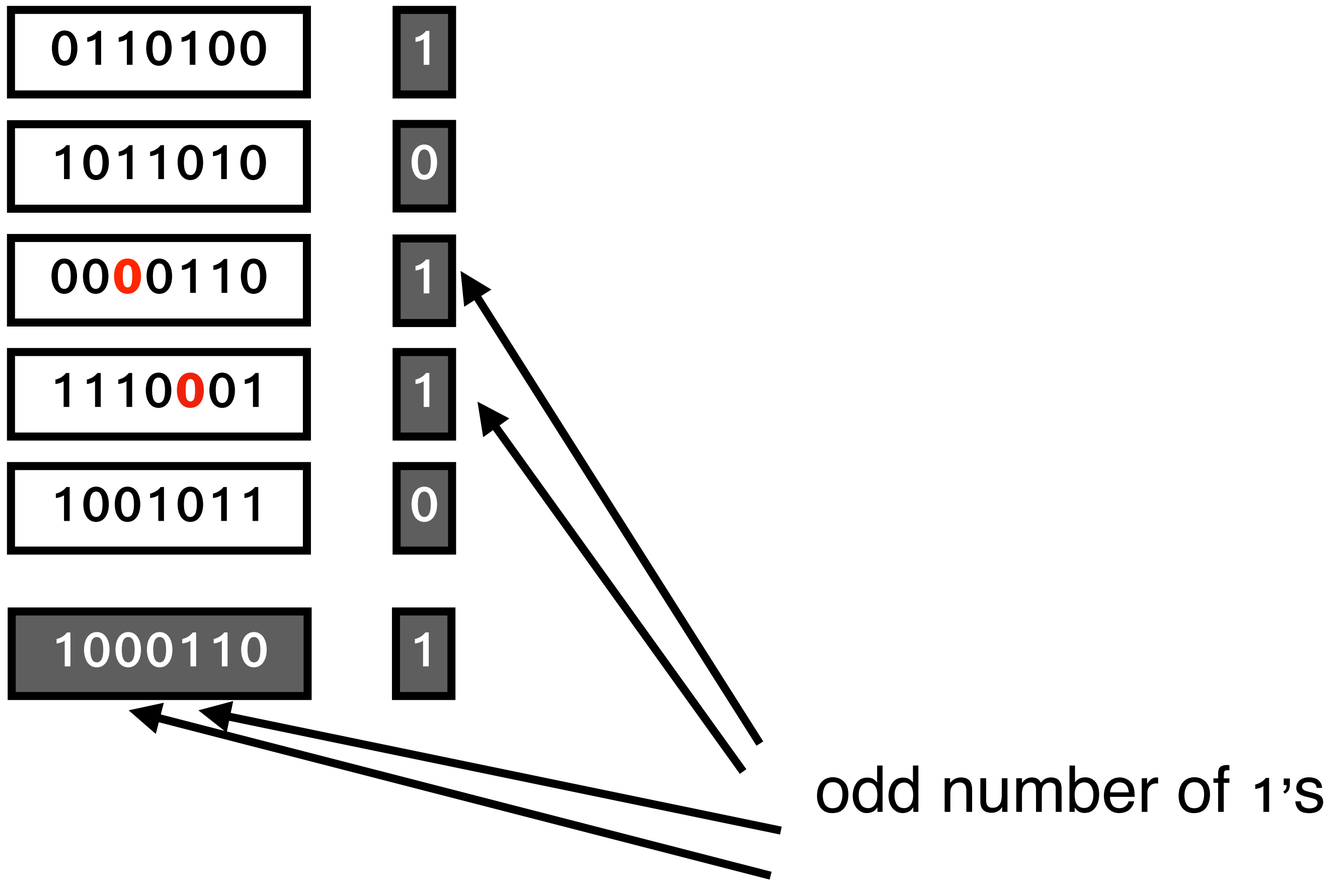
- An 1-bit error can be detected and corrected
- A 2-bit error can be detected



2-D Parity Capabilities

- An 1-bit error can be detected and corrected
- A 2-bit error can be detected

Different row and column



Technique #3: Internet Checksum

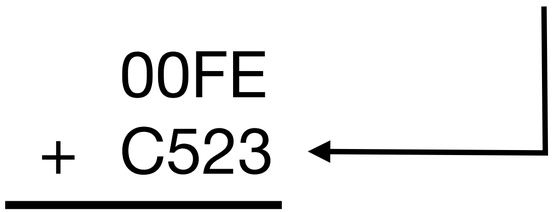
- Checksum = add up all the words of a frame
  - Ones complement arithmetic
  - 1101 + 1001 = 0111

- Workflow
  - #1: The sender divides the frame into a sequence of 16-bit data words
  - #2: The sender calculates the checksum and attaches it to the frame
  - #3: The receiver receives the frame and performs the same calculation
  - #4: The receiver performs one more calculation to figure out if its valid

An Example

Bytestream of a frame:

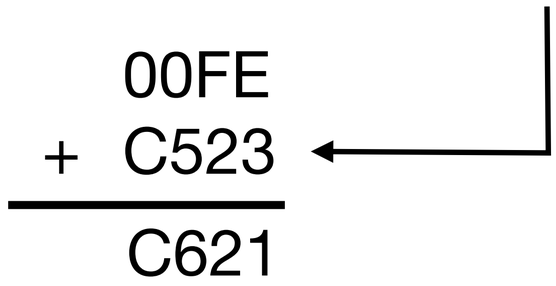
00FE C523 FDA1 D68A AF02



An Example

Bytestream of a frame:

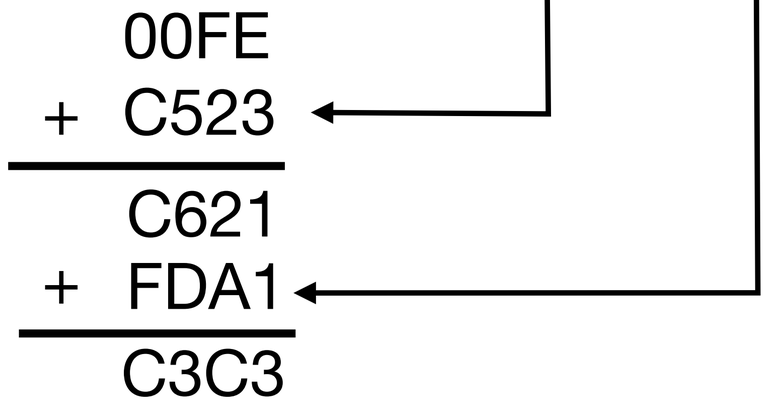
00FE C523 FDA1 D68A AF02



An Example

Bytestream of a frame:

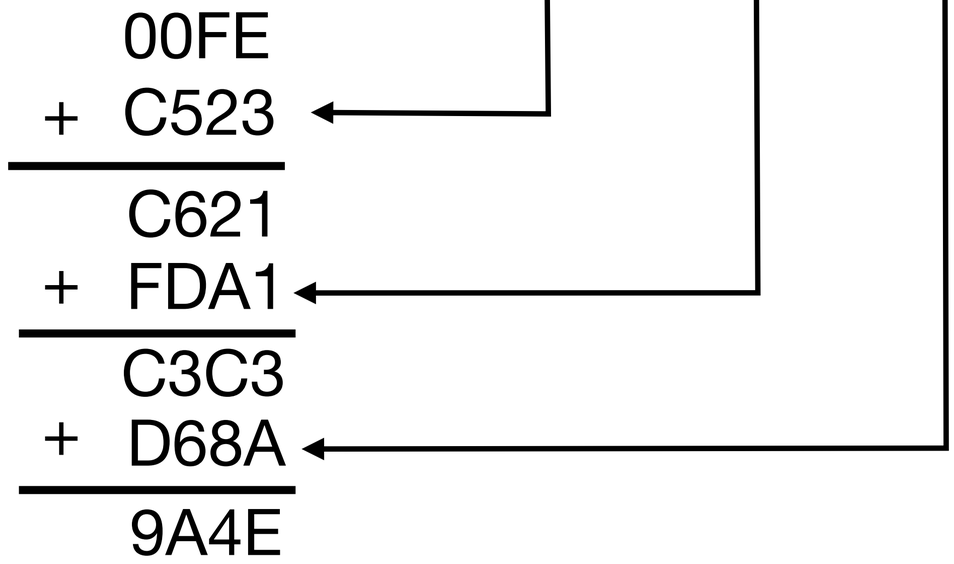
00FE C523 FDA1 D68A AF02



An Example

Bytestream of a frame:

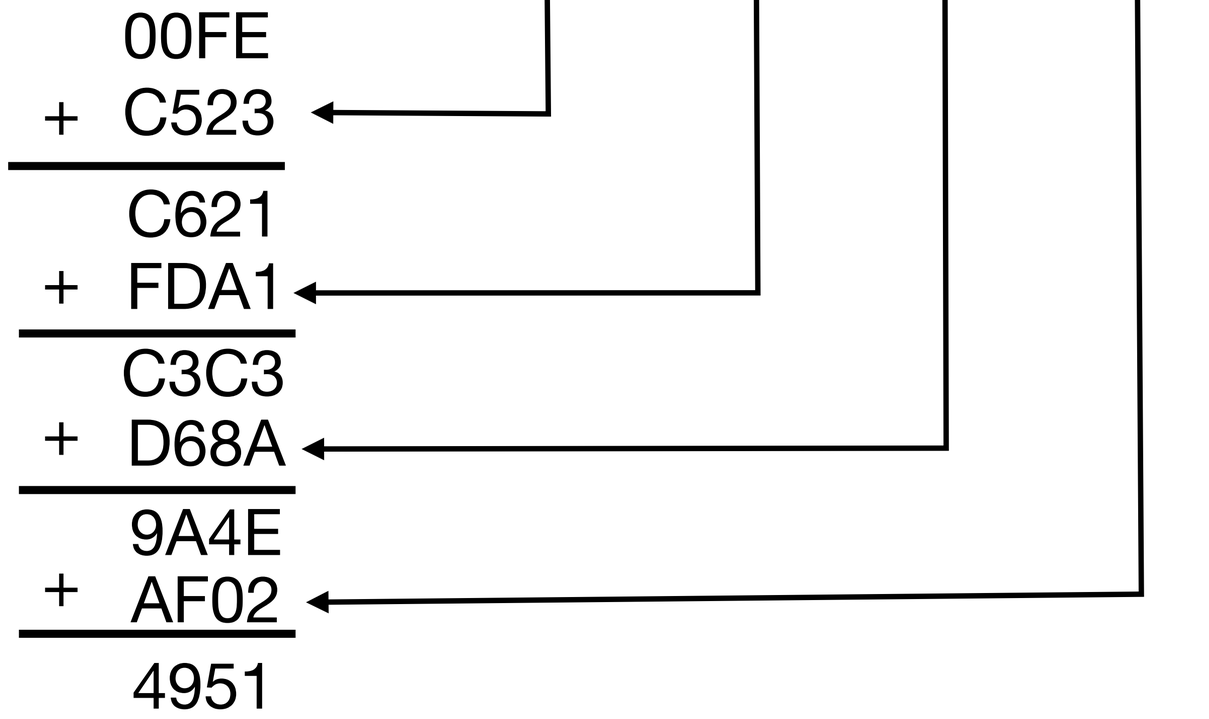
00FE C523 FDA1 D68A AF02



An Example

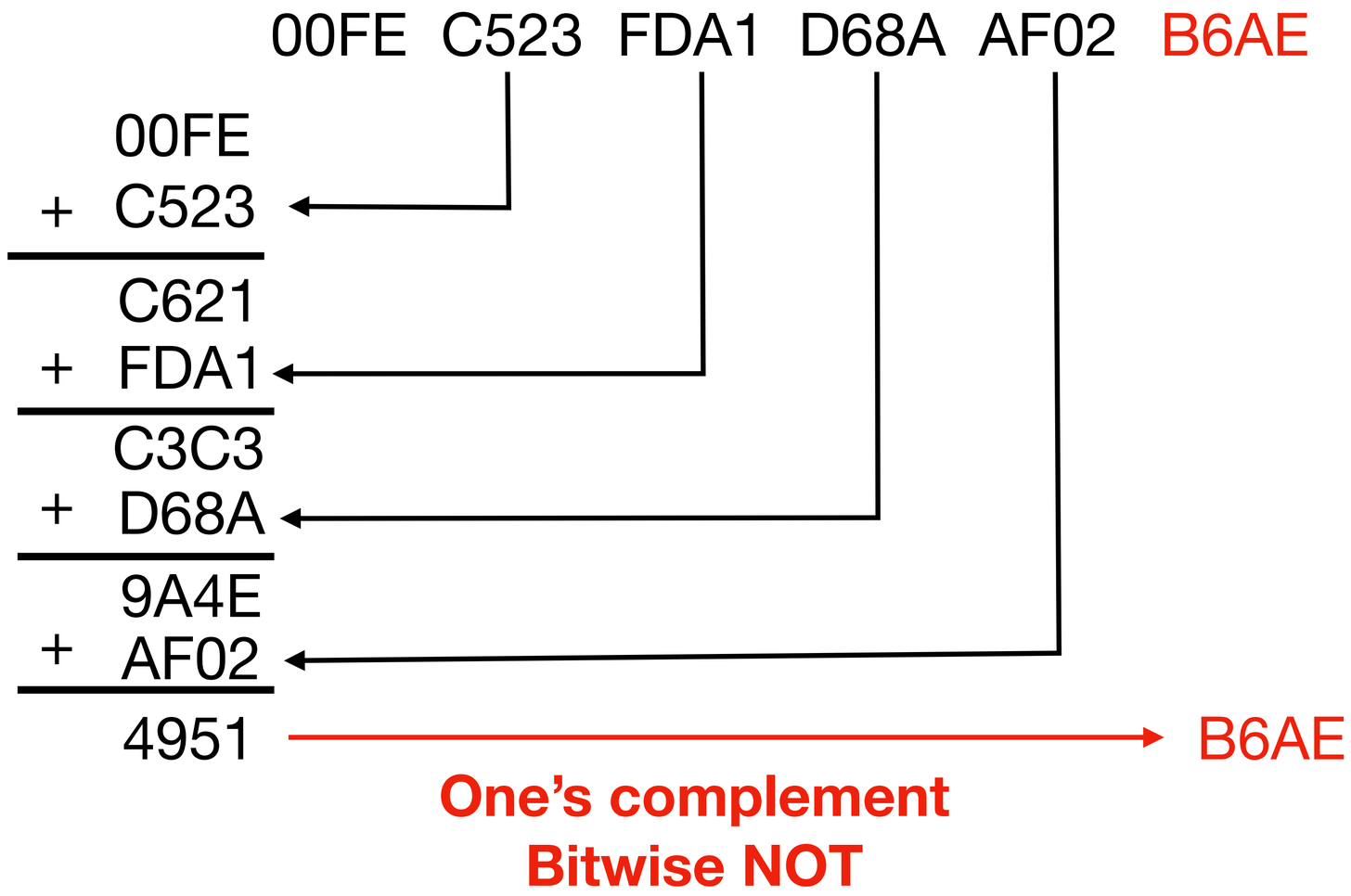
Bytestream of a frame:

00FE C523 FDA1 D68A AF02



An Example

Bytestream of a frame:



Internet Checksum Discussion

- Simple but not robust
  - 16 redundant bits for the whole frame
  - Easy to implement in the software
  - Concurrent errors without hurting the sum cannot be detected

Other Techniques

- Cyclic Redundancy Check (CRC)
  - Widely used codes with error detection properties
  - Ethernet frame check sequence (CRC-32)
  - Hardware friendly: K-bit shift registers and XOR gates

- High-level ideas
  - #1: View an (n+1)-bit frame as an n-degree polynomial M(x)
  - #2: The sender and receiver agree on the same divisor polynomial C(x)
  - #3: Error detection is performed by polynomial arithmetics

Summary

- Today
  - Framing and Error Handling

- Next lecture
  - L2 Switching