Computer Networks X_400487

Lecture 3

Chapter 3: The Data Link Layer—Part 2

Lecturer: Jesse Donkervliet

Data Link Layer — Roadmap

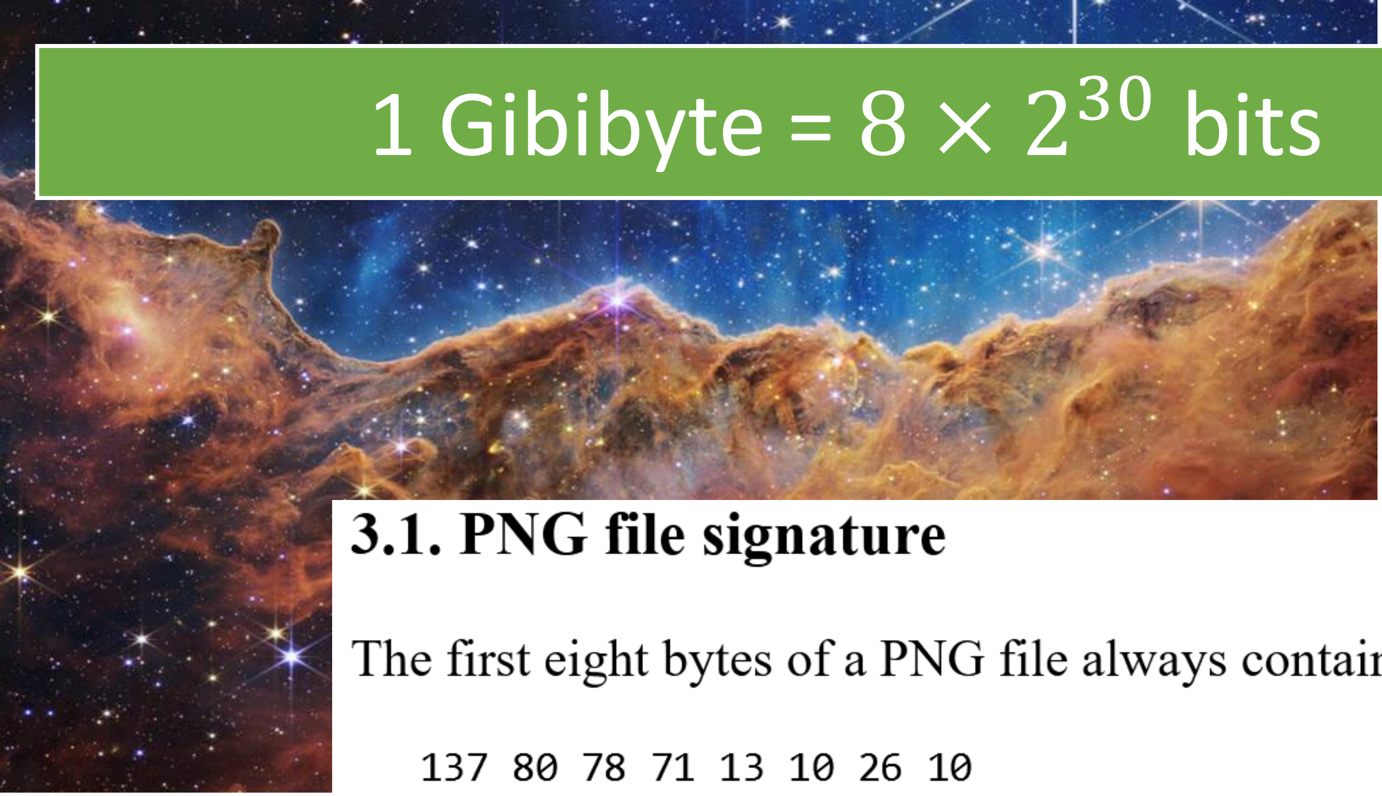
Part 1

- Framing
- Flow Control
- Guaranteed Delivery
- Sliding Window Protocols

Part 2

- **Error detection**
- **Error correction**

1 Gibibyte = 8×2^{30} bits



3.1. PNG file signature

The first eight bytes of a PNG file always contain the following (decimal) values:

```
137 80 78 71 13 10 26 10
```

A **single bit flip** can break these images

Error Detection

Detecting errors in received frames

Q: What causes these bit flips?

Data at sender: 0111010101010111010001

Data at receiver: 0111010111010111010001

Somehow bit was flipped!

Adding redundant bits

For a message of m bits,
send an extra r redundant bits.

Send $m + r$ to the receiver. ⟵ Systematic code

Data (m bits)	Check bits (r bits)

Codeword (m+r bits)

Hamming distance

Number of bits that differ between two bit strings.

10001001
10110001

Three bit flips required to change from one sequence to the other.

Adding redundant bits increases the distance between valid bit strings!

Hamming distance 3.

How many errors can we detect?

Consider code words:

000111

111111

000000

Q: How many single-bit errors can we detect?

Hamming distance 3,
so we can detect $3 - 1 = 2$ single-bit errors.

000111 111111

Error detection

Q: How to assess the quality of these codes?

Linear, systematic block codes:

1. Parity
2. Checksums
3. Cyclic Redundancy Checks (CRCs)

Block is $n = m + r$ bits large

Important code properties:

1. Code Rate: $\frac{m}{n}$
2. Number of errors reliably detected: N

Parity

Q: How many bit errors can be detected?

Add single bit such that:

- The sum of the data bits modulo 2 is 0.
- The number of 1's is even.

Send example: 1110000 → 1110000**1**

Receive example: 11010101 ← Error detected!

Easy to detect an *odd* number of errors.

Parity

Q: How many bit errors can be detected?

Add single bit such that:

- The sum of the data bits modulo 2 is 1.
- The number of 1's is odd.

Send example: 1110000 → 1110000**0**

Receive example: 11010100 ← Error detected!

Easy to detect an *odd* number of errors.

Multiple parity bits

transmit order →

1110000010101110101001010001111000

Multiple parity bits

transmit order $\longrightarrow$

1110000 → 1

0010101 → 1

1101010 → 0

0101000 → 0

1111000 → 0

Multiple single-bit errors detected.

Burst errors

transmit order

1110000 → 1
0010101 → 1
1101010 → 0
0101000 → 0
1111000 → 0

channel

Burst error not detected!

0011100 → 1
0010101 → 1
1101010 → 0
0101000 → 0
1111000 → 0

Burst errors

transmit order →

1110000		0011100
0010101	channel →	0010101
1101010		1101010
0101000		0101000
1111000		1111000
↓↓↓↓↓↓↓	Burst error detected.	↓↓↓↓↓↓↓
1011111		1011111

Checksums

Q: Disadvantages?

Checksum treats data as N-bit words and adds N check bits that are the modulo 2^N sum of the words.

Example: Internet 16-bit one's complement checksum.

Properties:

- Improved error detection over parity bits.
- Detects bursts up to N errors.
- Vulnerable to systematic errors, e.g., added zeros.

Checksum Example

transmit order →

Groups of 8 bits → calculate *sum mod 256*

11101000
00100101
11011010
01010000
11111000

add
00000000
11101000
----------- +
11101000

Internet checksum uses one's complement arithmetic

Checksum Example

Groups of 8 bits → calculate *sum mod 256*

transmit order →

11101000
00100101
11011010
01010000
11111000

add

11101000
00100101
----------- +
100001101

00001110

(one's complement arithmetic)

Internet checksum uses one's complement arithmetic

Checksum Example

Groups of 8 bits → calculate *sum mod 256*

transmit order →

11101000
00100101
11011010
01010000
11111000

add
00111001
11111000
---------- +
100110001

00110010

Internet checksum uses one's complement arithmetic

Checksum Example

transmit order

Groups of 8 bits → calculate *sum mod 256*

11101000
00100101
11011010
01010000
11111000
11001101

add
00111001
11111000
----------- +
100110001

00110010

invert

Internet checksum uses one's complement arithmetic

Checksum Example

Groups of 8 bits → calculate *sum mod 256*

transmit order →

11101000
00100101
11011010
01010000
11111000
11001101

channel →

11101000
00100101
11011010
01010000
11111000
11001101

(one's complement arithmetic)

\+

11111111 = 0 (no error)

Q: Why do we get 0?

Internet checksum uses one's complement arithmetic

Checksum Example

transmit order →

Groups of 8 bits → calculate *sum mod 256*

Sent		Received
11101000		11100000
00100101		00100101
11011010	channel →	11111010
01010000		01011000
11111000		11111000
11001101		10001101

(one's complement arithmetic)

+

11011111 = 223 (***error!***)

Internet checksum uses one's complement arithmetic

Checksum Example

transmit order →

Groups of 8 bits → calculate *sum mod 256*

Sent		Received
11101000		11101000
00100101	channel →	**11011010**
11011010		**00100101**
01010000		01010000
11111000		11111000
11001101		11001101

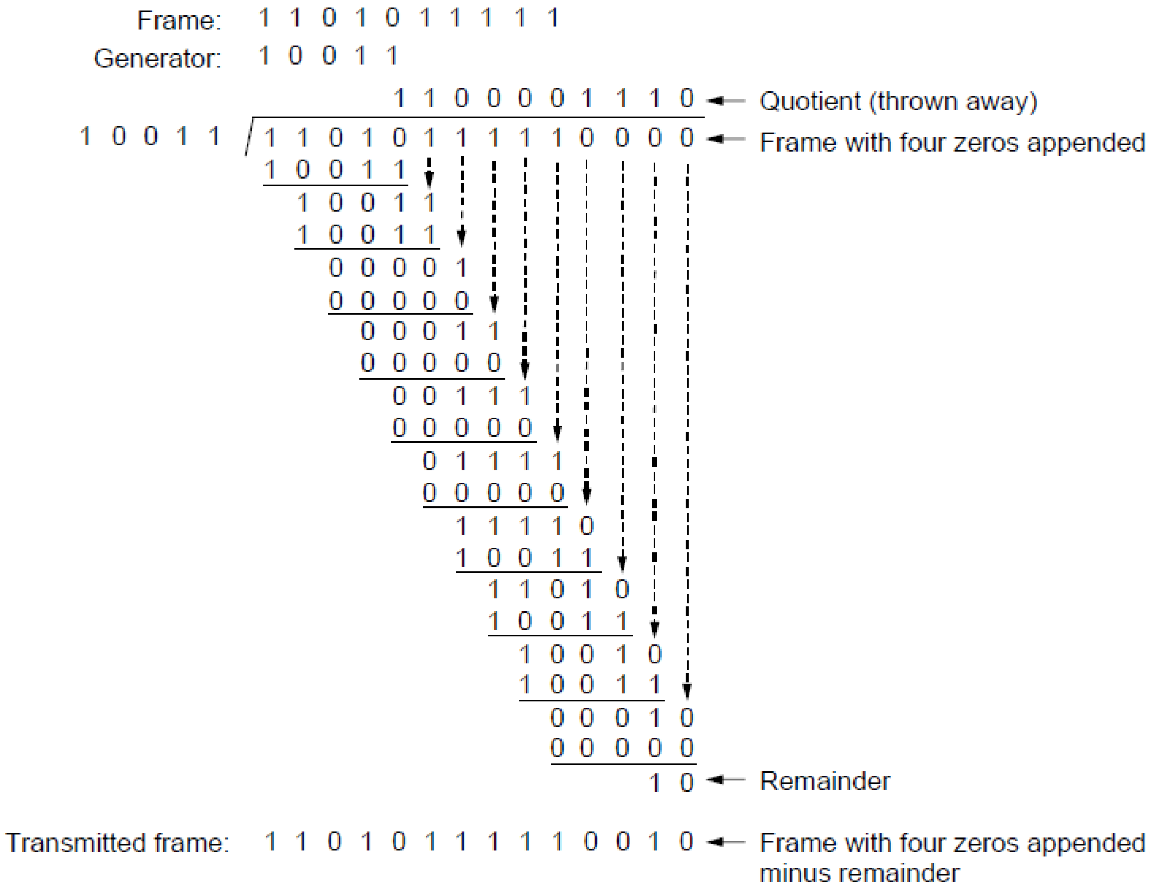
(one's complement arithmetic)

+

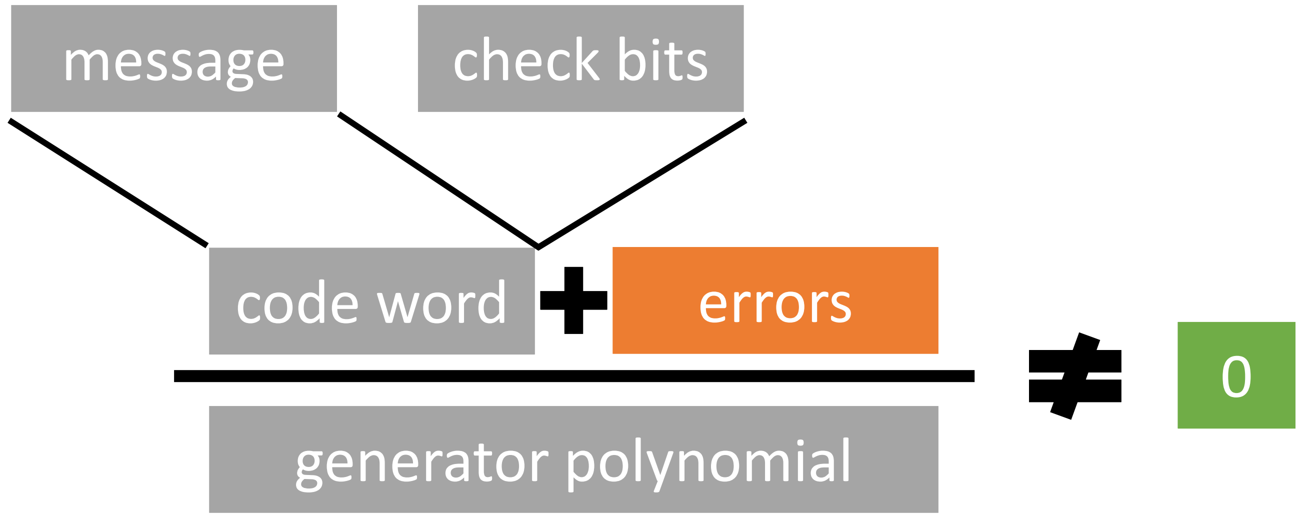
11111111 = 0 (no error)

Internet checksum uses one's complement arithmetic

Cyclic Redundancy Check



Cyclic Redundancy Check
The concept



Cyclic Redundancy Check
Properties and practice

Sender and receiver agree upon polynomial in advance

Example: Ethernet’s 33-bit polynomial is:
$x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} + x^{10} + x^{8} + x^{7} + x^{5} + x^{4} + x^{2} + x^{1} + 1$

CRC is computed with simple shift/XOR circuits

Because we use modulo 2 arithmetic.

Stronger detection than checksums:

1. Can detect all double bit errors, odd bit errors
2. Detect all burst errors $\leq r$ bits (in example, $r = 32$)
3. Not vulnerable to systematic errors
4. …

Cyclic Redundancy Check

```
Frame:      1 1 0 1 0 1 1 1 1 1
Generator:  1 0 0 1 1

                    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
```

Cyclic Redundancy Check Example

Sender adds CRC

$\mathbf{1} \times x^4 + \mathbf{0} \times x^3 + \mathbf{0} \times x^2 + \mathbf{1} \times x^1 + \mathbf{1} \times x^0$

message: 110101010000

generator: 10011

$x^4 + x + 1$

```
110101010000
10011
 10011010000
 10011
      10000
      10011
       0011
```

Modulo 2 arithmetic.
No carries/borrows

Q: Consequences for implementation?

Message: 11010101, CRC: 0011,
Codeword: 110101010011

$$\frac{110101010011}{10011} = 0$$

Cyclic Redundancy Check Example

Receiver checks for errors

message: 11010101**0011**

generator: 10011

```
  10011010011
  10011
        10011
        10011
            0
```

$$\frac{110101010011}{10011} = 0\text{, no error}$$

Error Correction

How many errors can we *correct*?

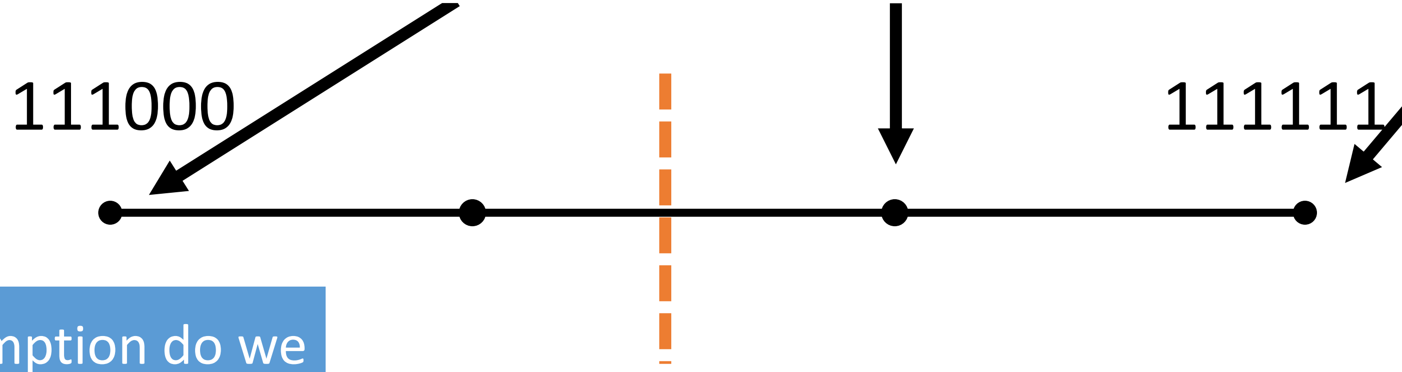
Consider a code with hamming distance n.

We have seen that we can **detect** $n-1$ single-bit errors.

But it could have been this value!

If we receive this value, …

… we assume it should be this value.



Q: What assumption do we need for this approach to work in practice?

Conputer Nrtworls

We can **correct** errors with $\lfloor\frac{n-1}{2}\rfloor$ changes.

Error correction

1. Hamming codes
2. Binary convolutional codes
3. Reed-Solomon codes
4. Low-Density Parity Check codes
5. … (many others)

Multiple parity bits

receive order →

1110000 → 1
0010101 → **0**
1101010 → 0
0101000 → 0
1111000 → 0

Error in second row!

Multiple parity bits

receive order →

1110000
0010101
1101010
0101000
1111000
↓↓↓↓↓↓↓
1010111

Error in fourth column!

Multiple parity bits

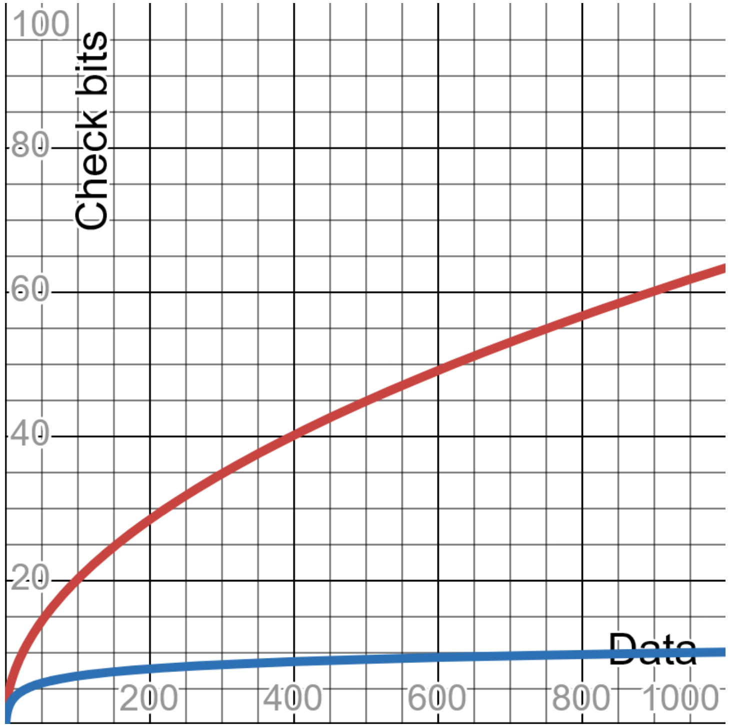
receive order →

1110000 → 1
0010101 → **0**
1101010 → 0
0101000 → 0
1111000 → 0
↓↓↓↓↓↓↓
1010111

Error located!

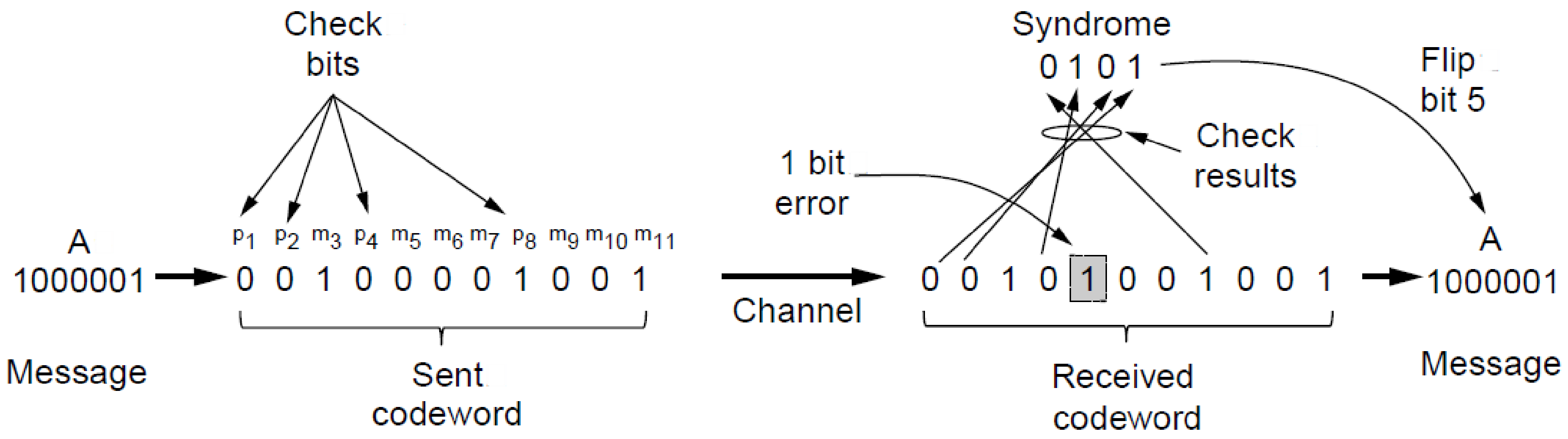
= Row+column parity bits

= Hamming code



Hamming codes

Minimum number of check bits such that *all* 1-bit errors can be *corrected*!



Example of an (11, 7) Hamming code correcting a single-bit error.

Hamming codes
An example

Use bit-locations that are a power of 2 as check bits.
Use the remaining positions for the message.

message: 11010101

codeword: __1_101_0101

positions: 123456789…

Hamming codes
An example

Use bit-locations that are a power of 2 as check bits.
Use the remaining positions for the message.

message: `1 101 0101`

codeword: `__1_101_0101`

positions: `123456789…`

1. Expand all bit locations into powers of two.
2. Decide the value of each check bit in position 2^i by calculating the parity function over all bits that have 2^i in their expansion.

Hamming codes
An example

1. Expand all bit locations into powers of two.

The check bit positions

$2 = 2$

$5 = 4 + 1$

All bit positions

	1	2	3	4	5	6	7	8
	0001	0010	0011	0100	0101	0110	0111	1000
1								
2								
4								
8								

Hamming codes
An example

1. Expand all bit locations into powers of two.

$2 = 2$

$5 = 4 + 1$

	1	2	3	4	5	6	7	8
	0001	0010	0011	0100	0101	0110	0111	1000
1	X		X		X		X	
2		X	X			X	X	
4				X	X	X	X	
8								X

Hamming codes
An example

2. Calculate the parity bit using all bits that have 2^i in their expansion

Check bit 1 is a parity bit calculated using bits 1, 3, 5, and 7

Check bit 4 is a parity bit calculated using bits 4, 5, 6, and 7

	1	2	3	4	5	6	7	8
	0001	0010	0011	0100	0101	0110	0111	1000
1	X		X		X		X	
2		X	X			X	X	
4				X	X	X	X	
8								X

Hamming codes
An example

2. Calculate the parity bit using all bits that have 2^i in their expansion

Use bit-locations that are a power of 2 as check bits.
Use the remaining positions for the message.

message: 1 101 0101

codeword: __1_101_0101

positions: 123456789…

Position 1+2

Position 4

Position 4+2+1

Hamming codes
An example

Use bit-locations that are a power of 2 as check bits.
Use the remaining positions for the message.

message: 1 101 0101

codeword: _ _ **1** _ **1** 0 **1** _ **0** 1 **0** 1

positions: **1** 2 3 4 5 6 7 8 9…

Hamming codes
An example

Use bit-locations that are a power of 2 as check bits.
Use the remaining positions for the message.

message: `  1 101 0101`

codeword: `1_1_101_0101`

positions: `123456789…`

Hamming codes
An example

Use bit-locations that are a power of 2 as check bits.
Use the remaining positions for the message.

message: 1 101 0101

codeword: 1 _ **_** **1** _ 1 **01** _ 0 **10** 1

positions: 1**2**3456789…

Hamming codes
An example

Use bit-locations that are a power of 2 as check bits.
Use the remaining positions for the message.

message: `  1 101 0101`

codeword: `111_101_0101`

positions: `123456789…`

Hamming codes
An example

Use bit-locations that are a power of 2 as check bits.
Use the remaining positions for the message.

message: 1 101 0101

codeword: 111110100101

positions: 123456789…

Hamming codes
Error correction

Use bit-locations that are a power of 2 as check bits.
Use the remaining positions for the message.

message: 1 101 0101

Single-bit error

codeword: 11111**1**100101

positions: 123456789…

Computer *error syndrome*:

Check bit 1 = parity of bits 1, 3, 5, 7, 9, 11

Hamming codes
Error correction

Use bit-locations that are a power of 2 as check bits.
Use the remaining positions for the message.

message: `1 101 0101`

Single-bit error

codeword: `11111`**1**`100101`

positions: `123456789…`

Computer *error syndrome*:

Check bit 1 = parity of bits <u>1, 3, 5, 7, 9, 11</u> = 0

Hamming codes
Error correction

Use bit-locations that are a power of 2 as check bits.
Use the remaining positions for the message.

message: 1 101 0101

Single-bit error

codeword: 11111**1**100101

positions: 123456789…

Computer *error syndrome*:

Check bit 1 = parity of bits 1, 3, 5, 7, 9, 11 = 0

Check bit 2 = parity of bits 2, 3, 6, 7, 10, 11 = 1

Hamming codes
Error correction

Use bit-locations that are a power of 2 as check bits.
Use the remaining positions for the message.

message: 1 101 0101

Single-bit error

codeword: 11111**1**100101

Error at location:
110 (binary)

positions: 123456789…

Computer *error syndrome*:

Check bit 1 = parity of bits 1, 3, 5, 7, 9, 11 = 0

Check bit 2 = parity of bits 2, 3, 6, 7, 10, 11 = 1

Check bit 4 = parity of bits 4, 5, 6, 7, 12 = 1

Hamming codes
Error correction

Use bit-locations that are a power of 2 as check bits.
Use the remaining positions for the message.

message: 1 101 0101

codeword: 11111**1**100101

positions: 123456789…

Single-bit error

Error at location: 110 (binary)

Computer *error syndrome*:

Check bit 1 = parity of bits 1, 3, 5, 7, 9, 11 = 0

Check bit 2 = parity of bits 2, 3, 6, 7, 10, 11 = 1

Check bit 4 = parity of bits 4, 5, 6, 7, 12 = 1

Convolutional codes

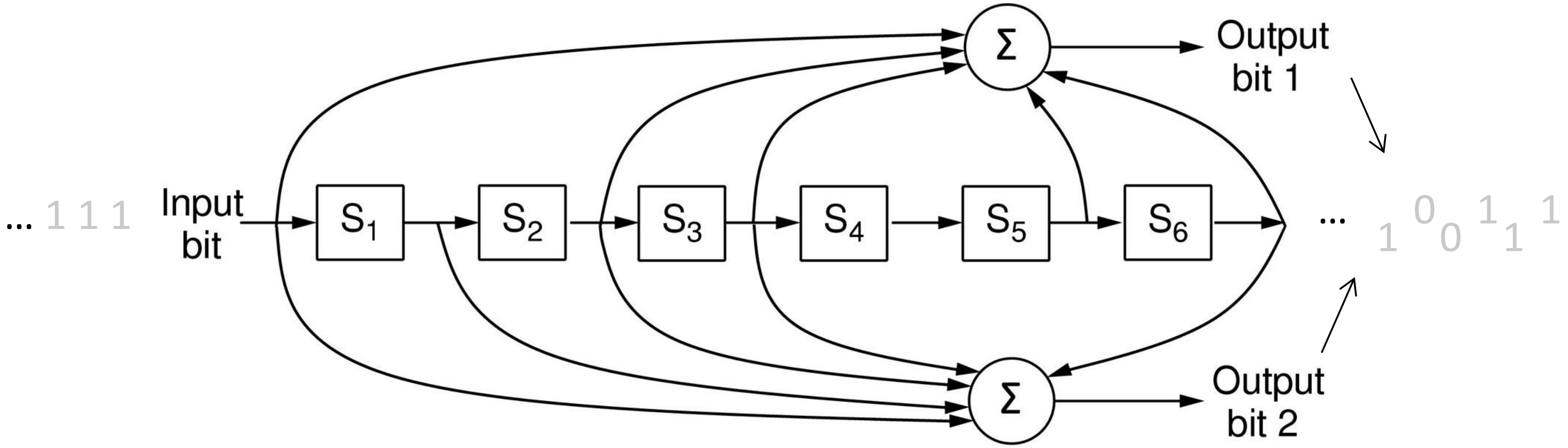
Different from *systematic* codes and *block* codes

Operates on a stream of bits, keeping internal state.

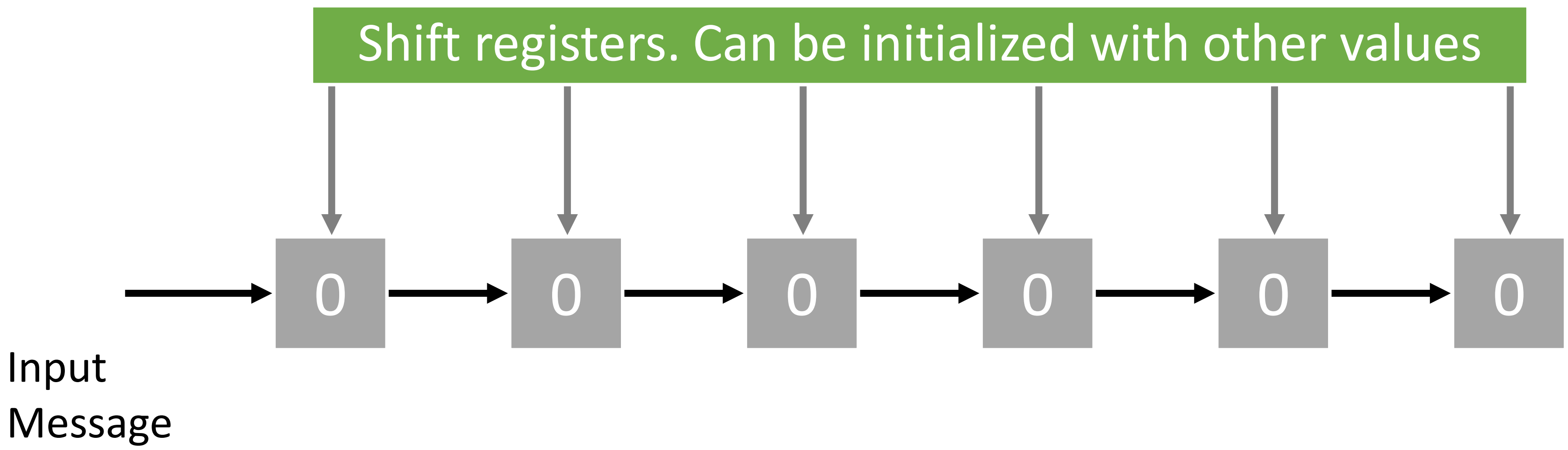
Output stream is a function of last k preceding input bits.

Bits are decoded with the Viterbi algorithm.

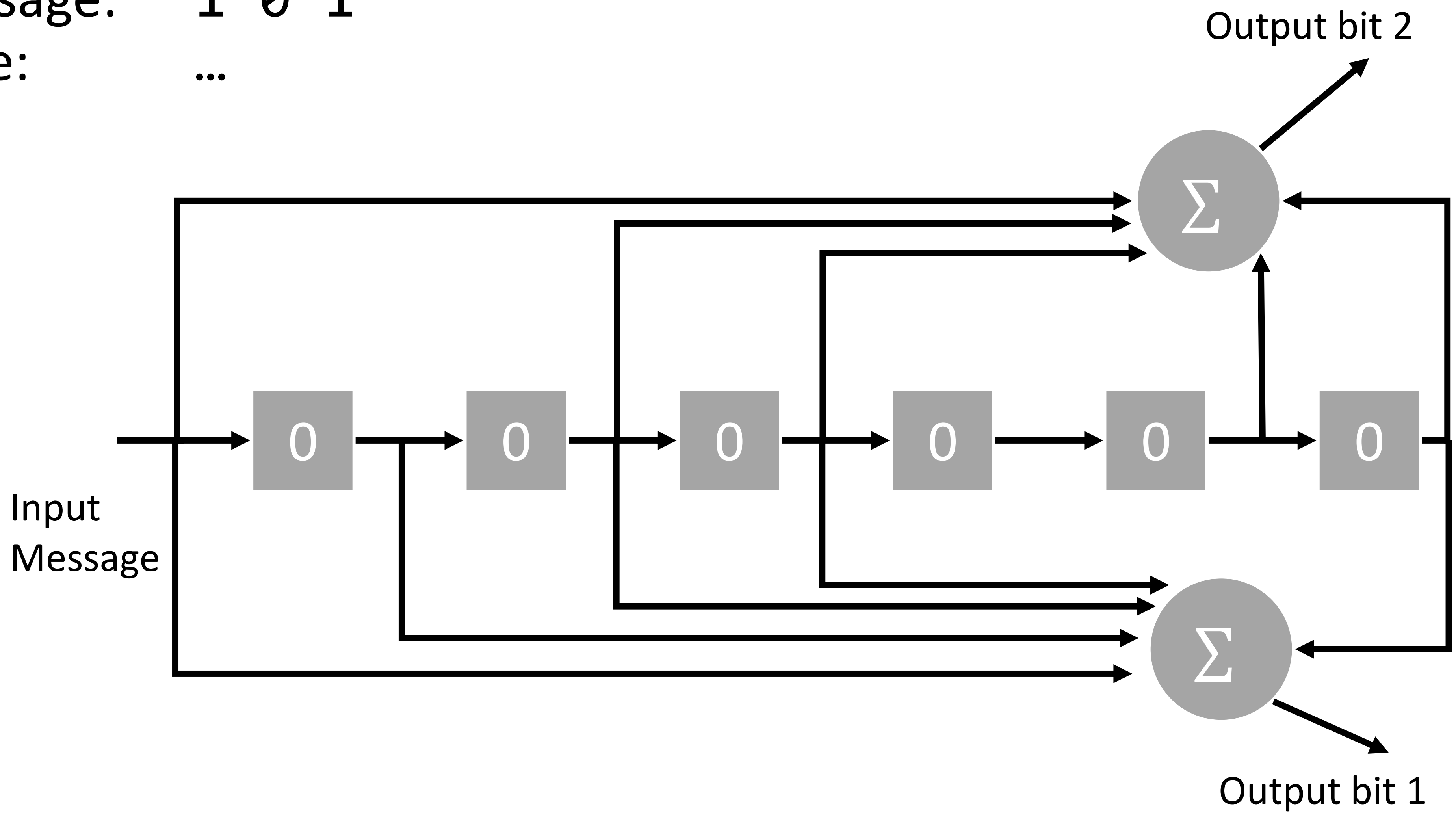
Determines most likely input for given output.



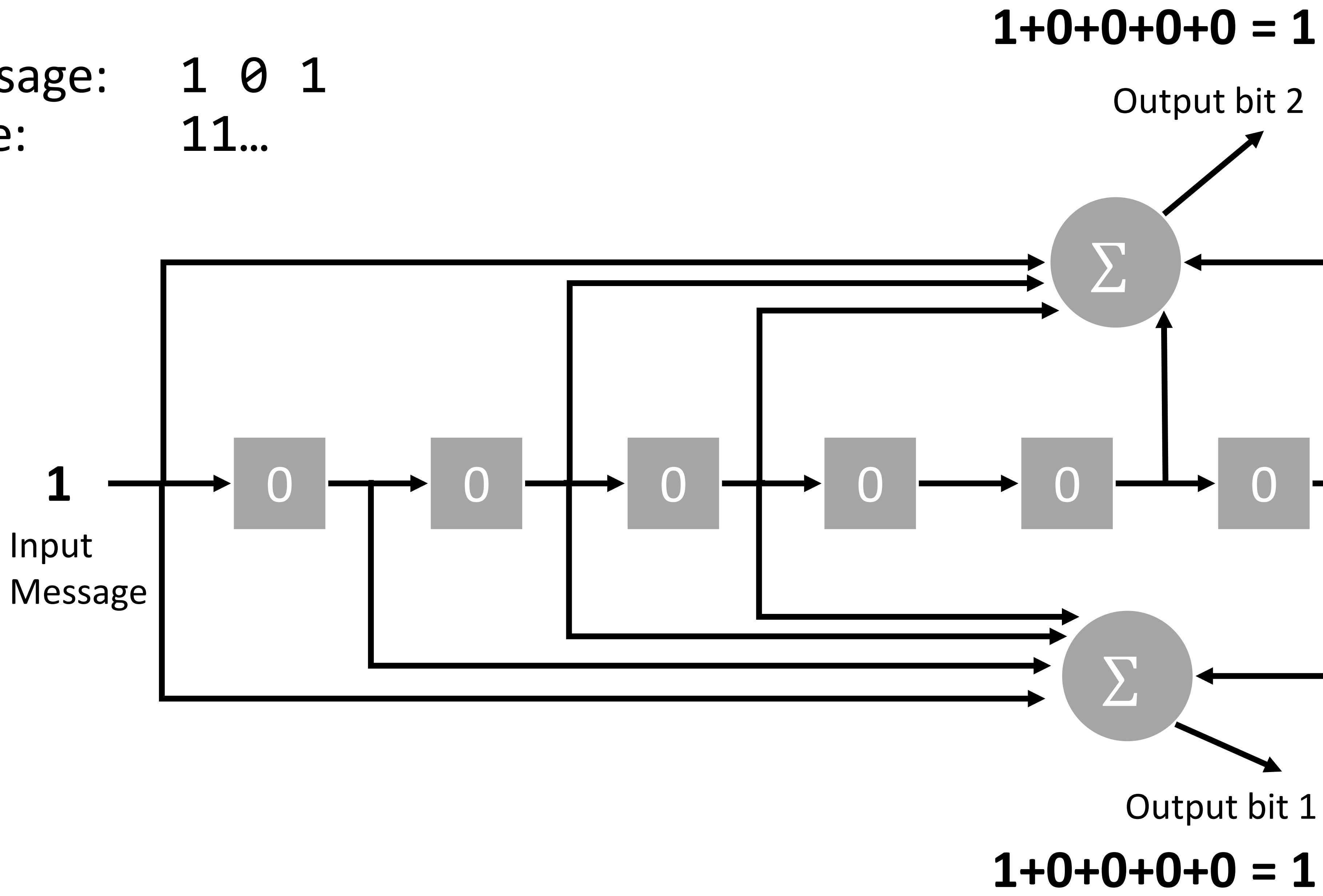
Popular NASA binary convolutional code (rate = ½) used in 802.11 $k = 7$


Shift registers. Can be initialized with other values
0
0
0
0
0
0
Input
Message

Message: `1 0 1`
Code: …

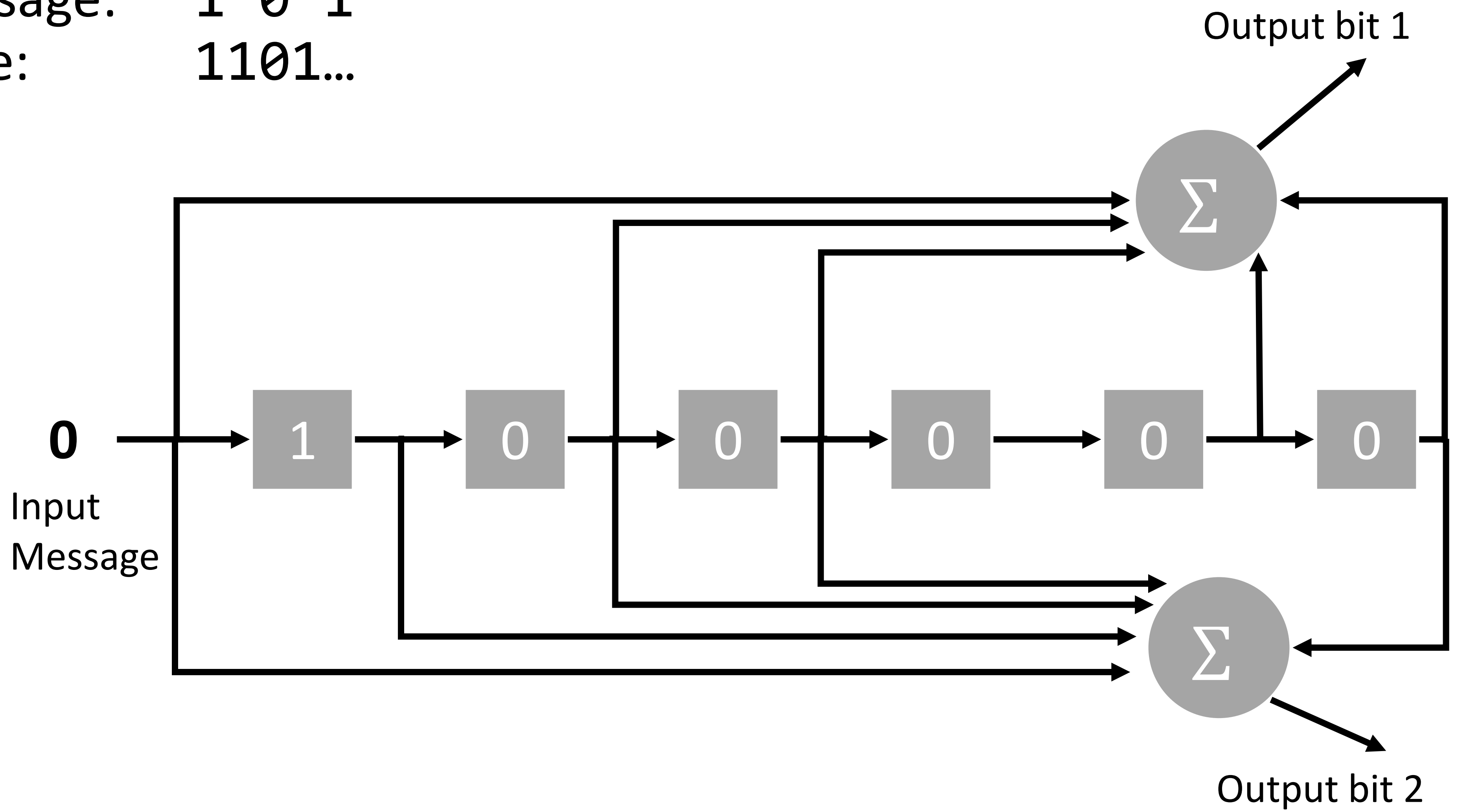


Message: 1 0 1
Code: 11…



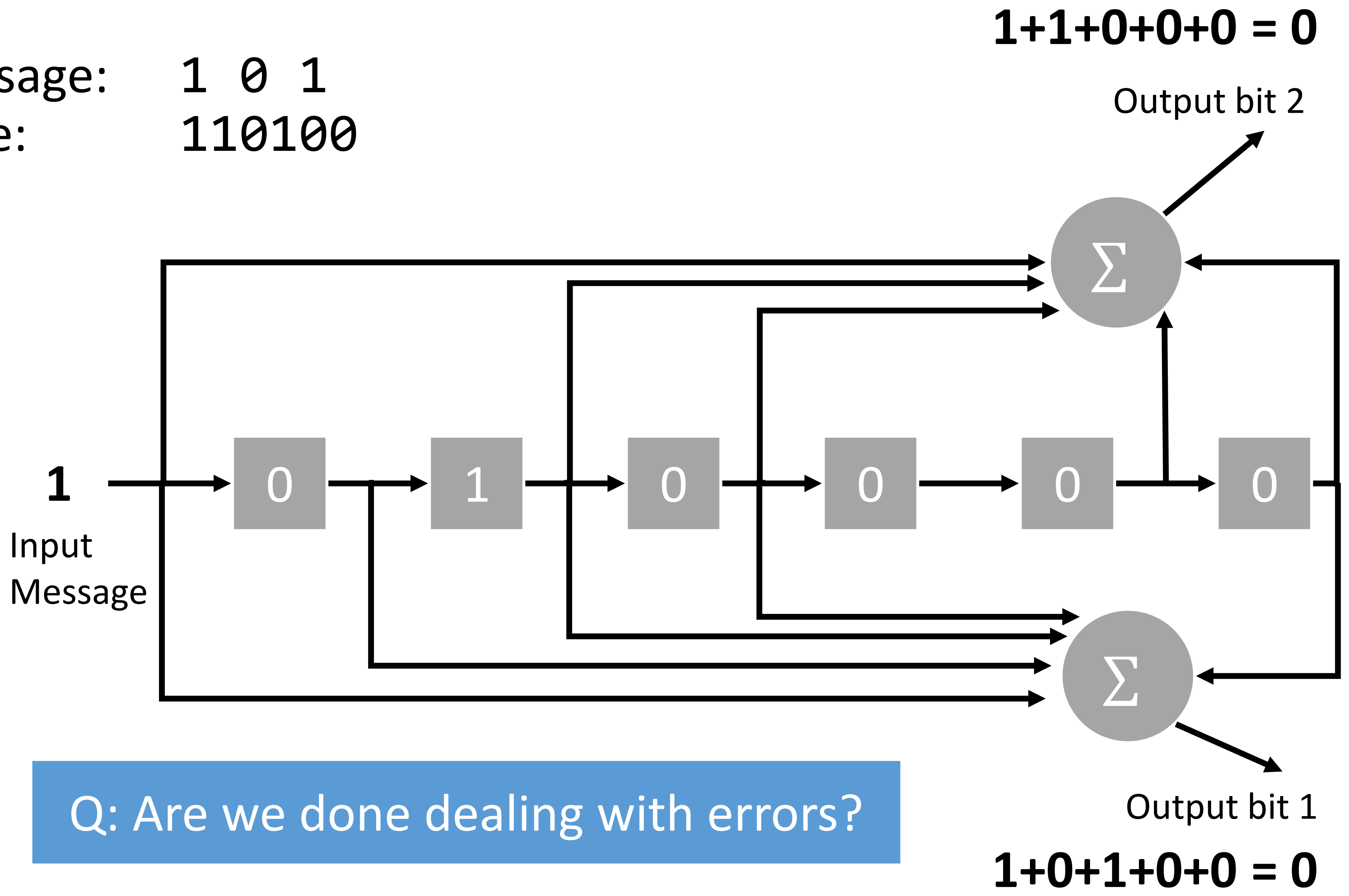
Message: 1 0 1
Code: 1101…

0+0+0+0+0 = 0



0+1+0+0+0 = 1

Message: 1 0 1
Code: 110100



Q: Are we done dealing with errors?

Data Link Layer Summary

A1: It depends on the physical medium

A2: It depends on the application

Framing (byte stuffing, bit stuffing, etc)

Guaranteed delivery Q: When needed?

- Sequence numbers, acknowledgments, and retransmissions

Flow control Q: When needed?

- Stop-and-Wait
- Sliding Window

A3: We may want to address the problem in a higher layer

Q: Can you think of an example?

Error control Q: When needed?

- Detection (e.g., Parity bit, Checksum, CRC, …)
- Correction (e.g., Hamming Code, Convolutional Code, …)