

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

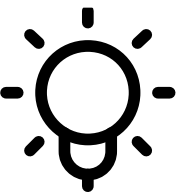
The first eight bytes of a PNG file always contain the following (decimal) values:

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: 01110101**1**1010111010001

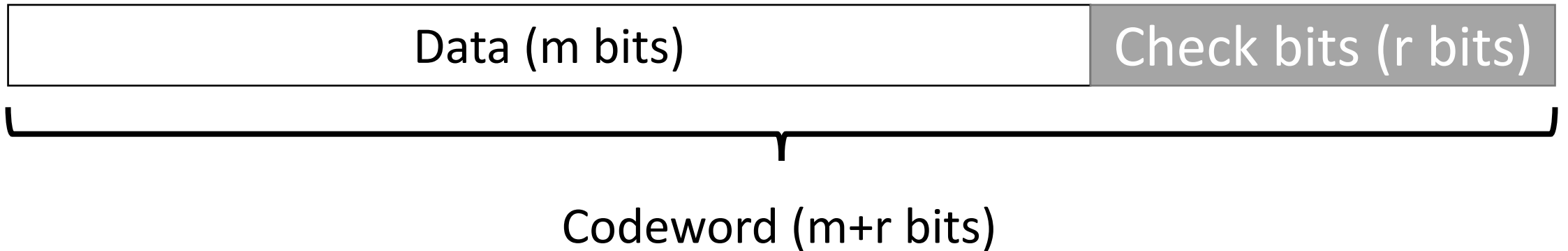


Somehow bit was flipped!

Adding redundant bits

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

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



Hamming distance

Number of bits that differ between two bit-strings

10001001

10**11**0001

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



Q: How to assess the quality of these codes?

Error detection

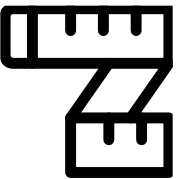
Linear, systematic block codes:

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

Block is $n = m + r$ bits large

Important code properties:

1. Code Ratio: $\frac{m}{n}$
2. How many / which kind of errors are reliably detected?



Q: How many bit errors
can be detected?

Parity

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

Q: How many bit errors
can be detected?

Parity

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 →

11100000010101110101001010001111000

Multiple parity bits

transmit order →

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

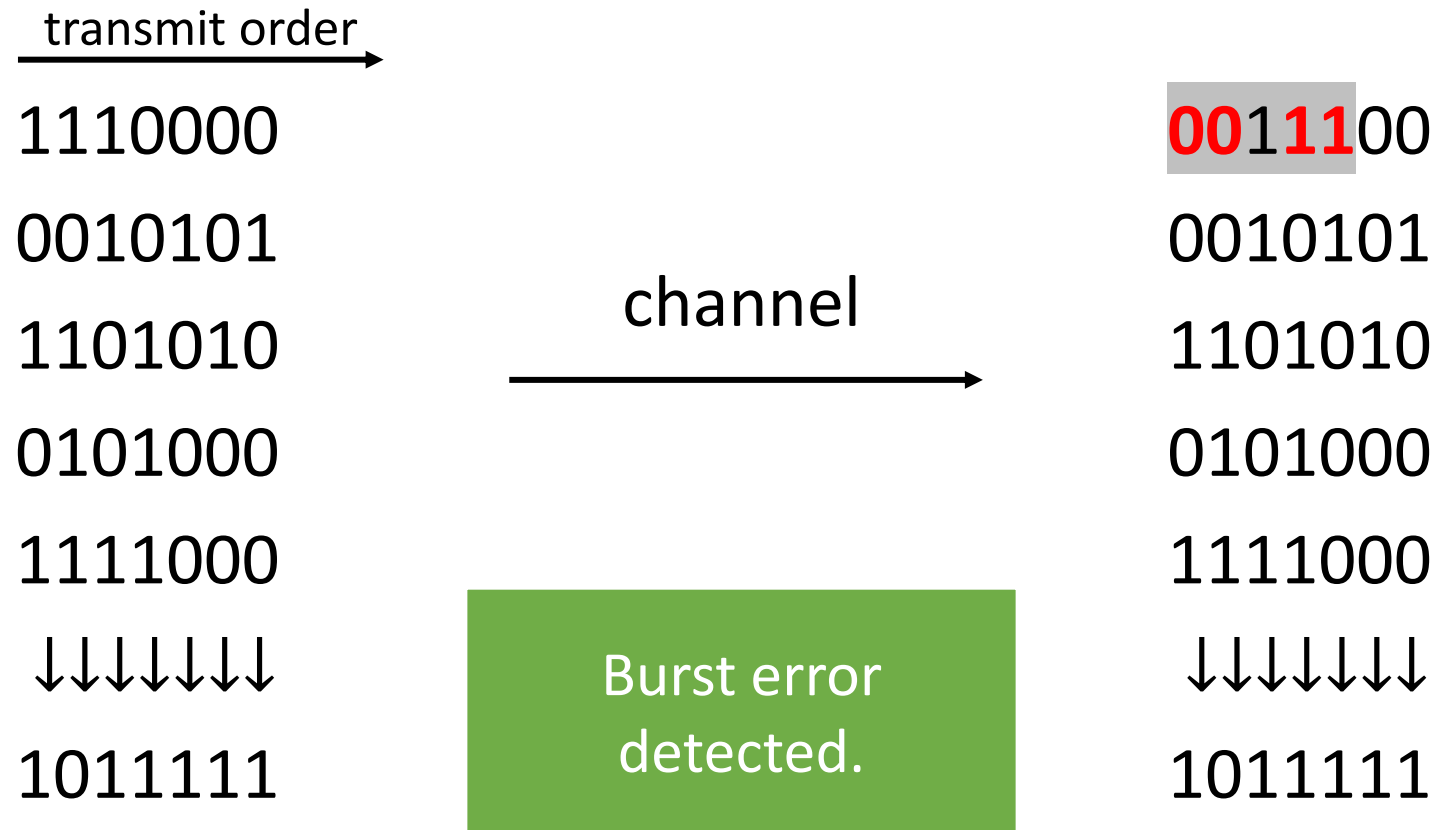
channel →

001100 → 1
0010101 → 1
1101010 → 0
0101000 → 0
1111000 → 0

Burst error not detected!



Burst errors



Checksums

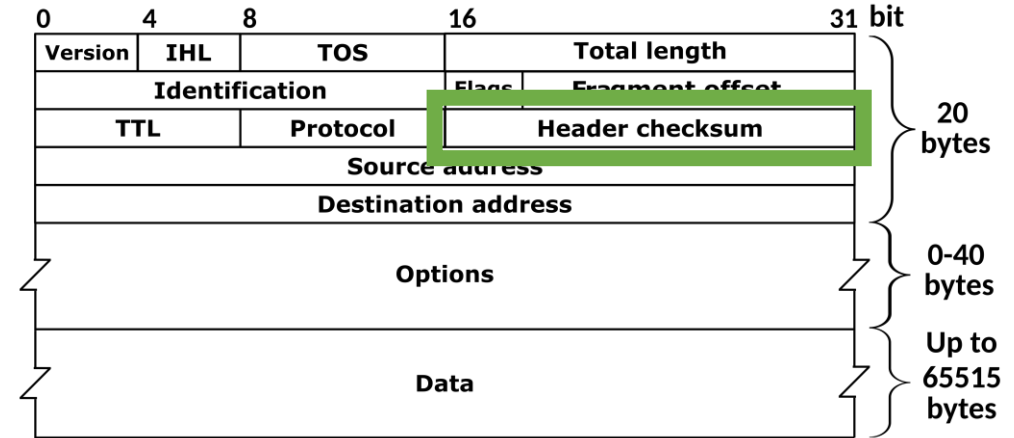
Q: Disadvantages?

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

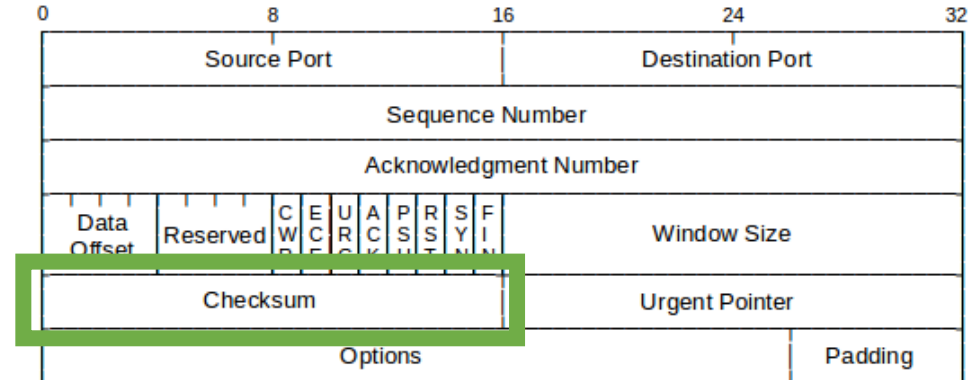
Example: Internet (i.e., IPv4) 16-bit one's complement header checksum

Properties:

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



Used in IPv4 (network layer)



Used in TCP (transport layer)

Checksum Example

Groups of
8 bits →
calculate
sum mod
256

transmit order →
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 →

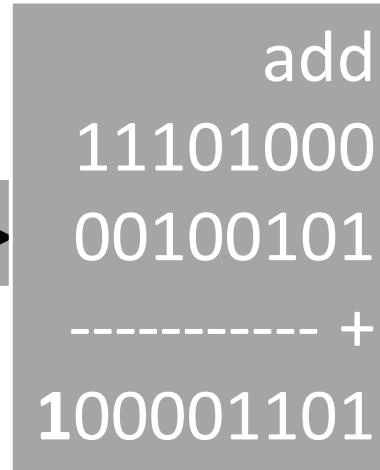
11101000

00100101

11011010

01010000

11111000



00001110

(carry added back: 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

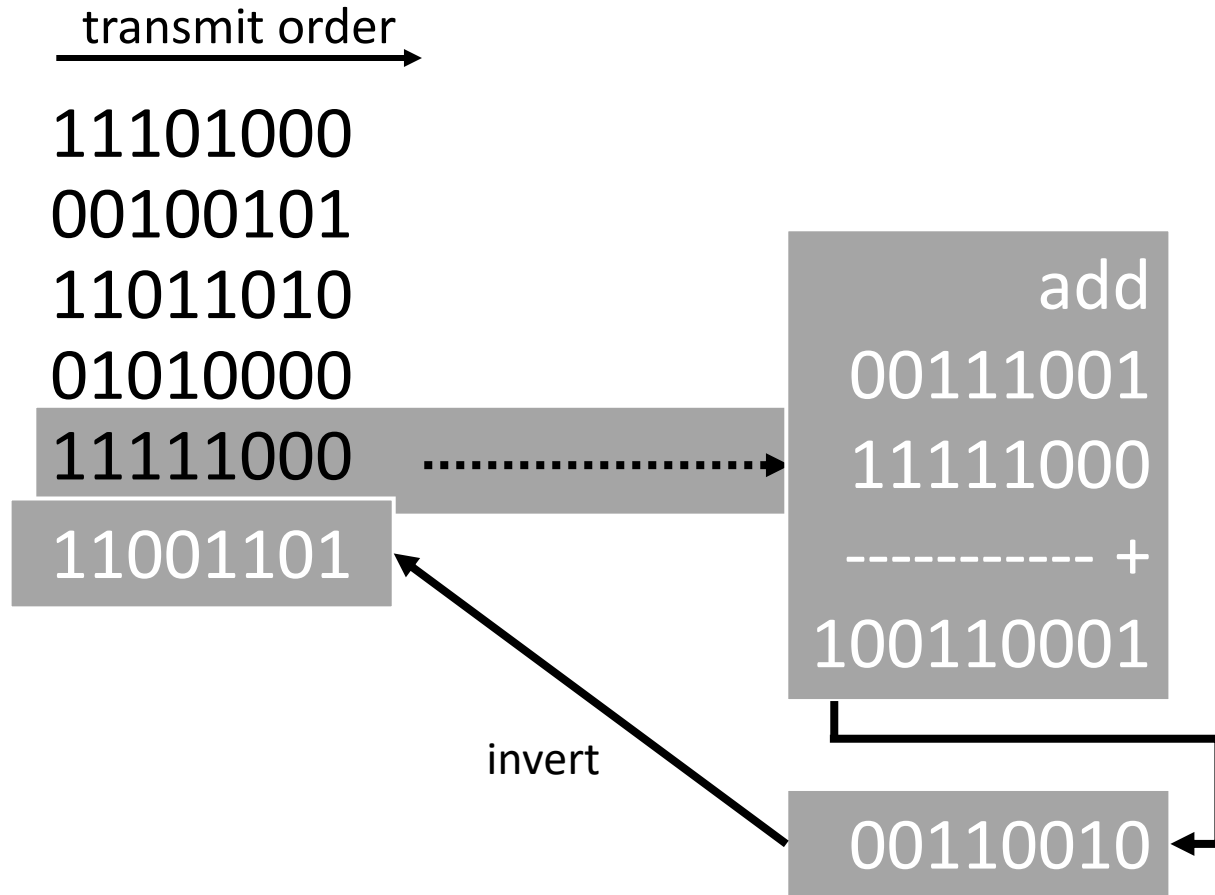
add
00111001
11111000
----- +
100110001

00110010

Internet checksum uses one's complement arithmetic

Checksum Example

Groups of
8 bits →
calculate
sum mod
256



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

Groups of
8 bits →
calculate
sum mod
256

transmit order →

11101000
00100101
11011010
01010000
11111000
11001101

channel →

11100000
00100101
11111010
01011000
11111000
10001101

(one's complement arithmetic)

+

11011111 = 223 (**error!**)

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
11011010
00100101
01010000
11111000
11001101



(one's complement arithmetic)

11001101
+
11111111 = 0 (no error)

Internet checksum uses one's complement arithmetic

Checksum Example #2

To send: AB A8 EF F3

Compute 8-bit (1 byte) word checksum:

AB

A8

EF

F3

—+ (1's complement)

??5

335

$$B(11)+8+F(15)+3=37$$

$$37 = 2 \times 16 + 5$$

$$= 0x25 \text{ (carry = 2)}$$

$$2+A(10)+A+E(14)+F=51$$

$$51 = 3 \times 16 + 3$$

$$= 0x33$$

Checksum Example #2

To send: AB A8 EF F3

Compute 8-bit (1 byte) word checksum:

AB

A8

EF

F3

—+ (1's complement)

38 (335 → 3 + 35 = 38)

$$B(11)+8+F(15)+3=37$$

$$37 = 2 \times 16 + 5$$

$$= 0x25 \text{ (carry = 2)}$$

$$2+A(10)+A+E(14)+F=51$$

$$51 = 3 \times 16 + 3$$

$$= 0x33$$

Checksum Example #2

To send: AB A8 EF F3

Compute 8-bit (1 byte) word checksum:

AB

A8

EF

F3

—+ (1's complement)

38 (335 → 3 + 35 = 38)

Checksum is inverse
of 38:

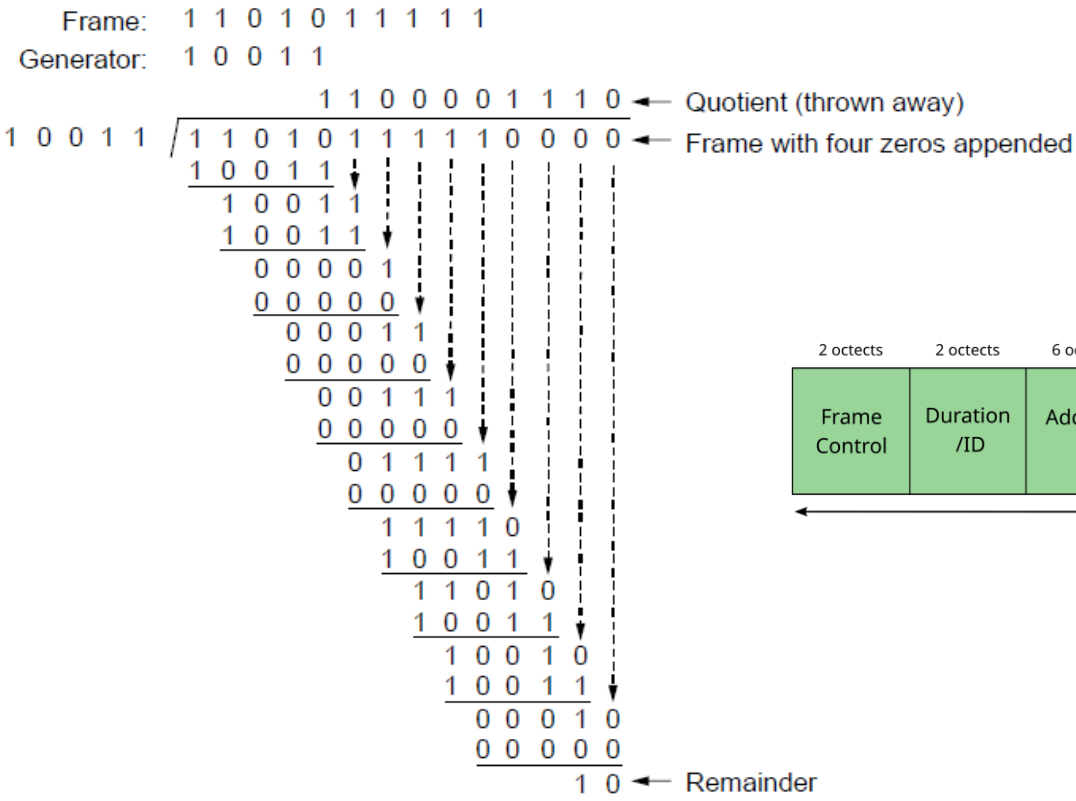
3 → C (15 - 3 = 12 = C)

8 → 7 (15 - 8 = 7 = 7)

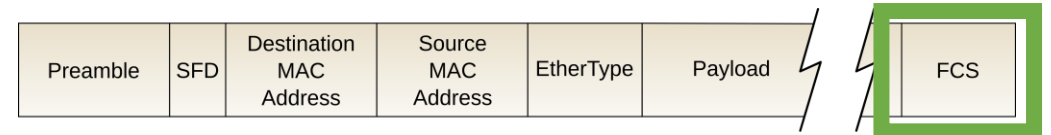
Checksum: C7

Lecture resumes at 14:18

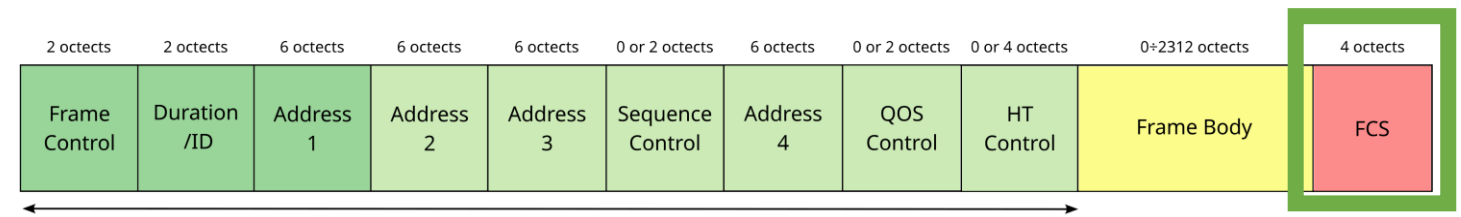
Cyclic Redundancy Check



Transmitted frame: 1 1 0 1 0 1 1 1 1 1 0 0 1 0 ← Frame with four zeros appended minus remainder



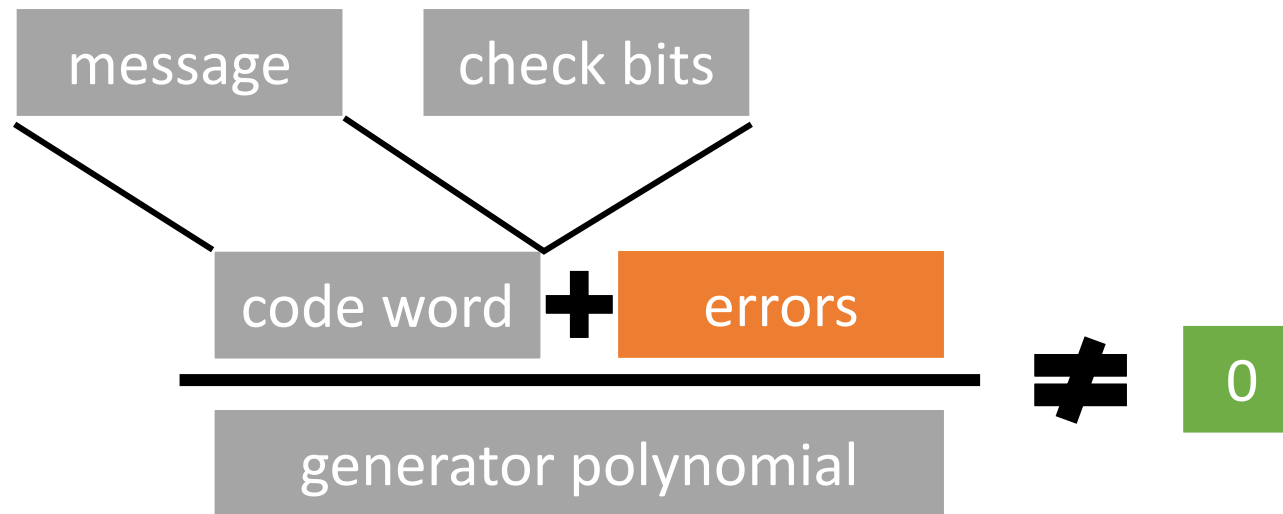
Used in Ethernet (link layer)



Used in WiFi (link layer)

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

Example

$$1 \times x^4 + 0 \times x^3 + 0 \times x^2 + 1 \times x^1 + 1 \times x^0$$

message: 110101010000

generator: 10011

Modulo 2 arithmetic.
No carries/borrows

Q: Consequences for
implementation?

$$x^4 + x + 1$$

10011010000

10011

10011

10011

0011

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

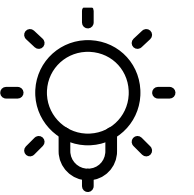
10011

0



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

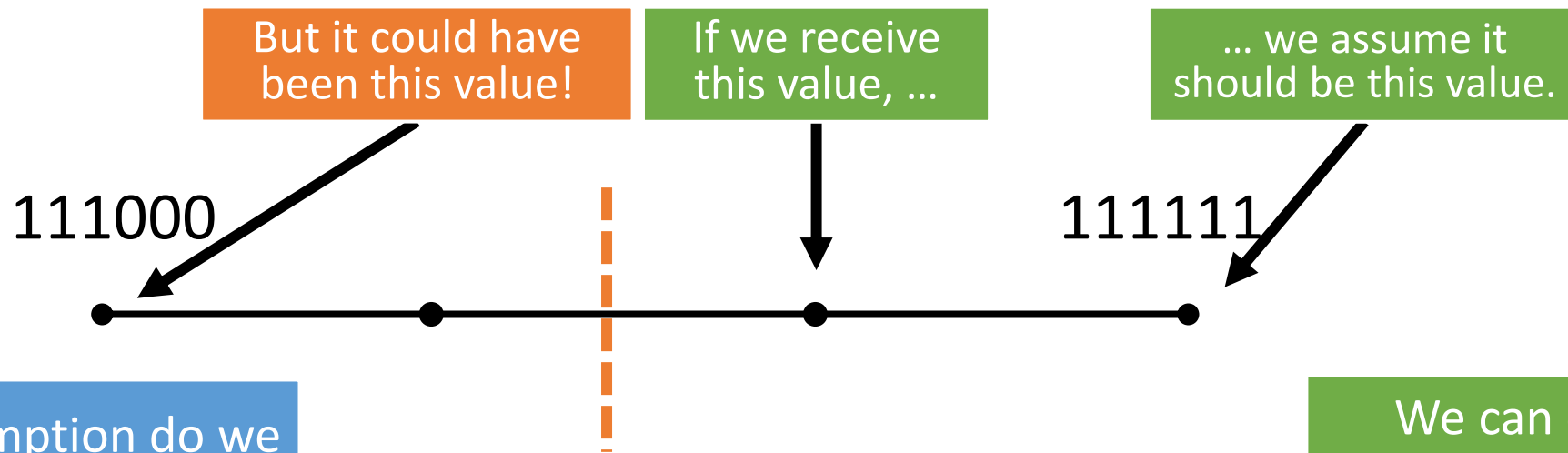
Error Correction



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.



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

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 →

1	1	1	0	0	0	0
0	0	1	0	1	0	1
1	1	0	1	0	1	0
0	1	0	1	0	0	0
1	1	1	1	0	0	0
↓	↓	↓	↓	↓	↓	↓
1	0	1	0	1	1	1

Error in fourth column!

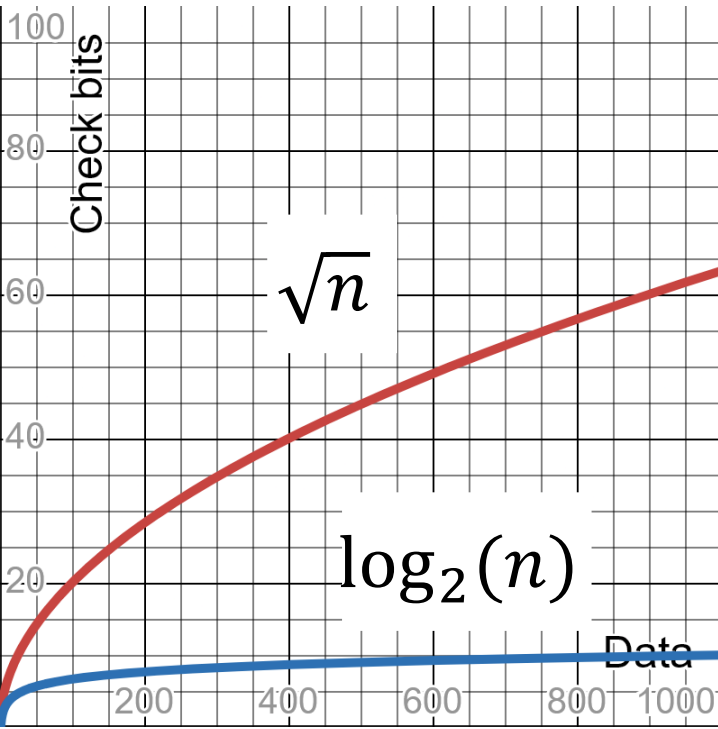
Multiple parity bits

receive order →

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

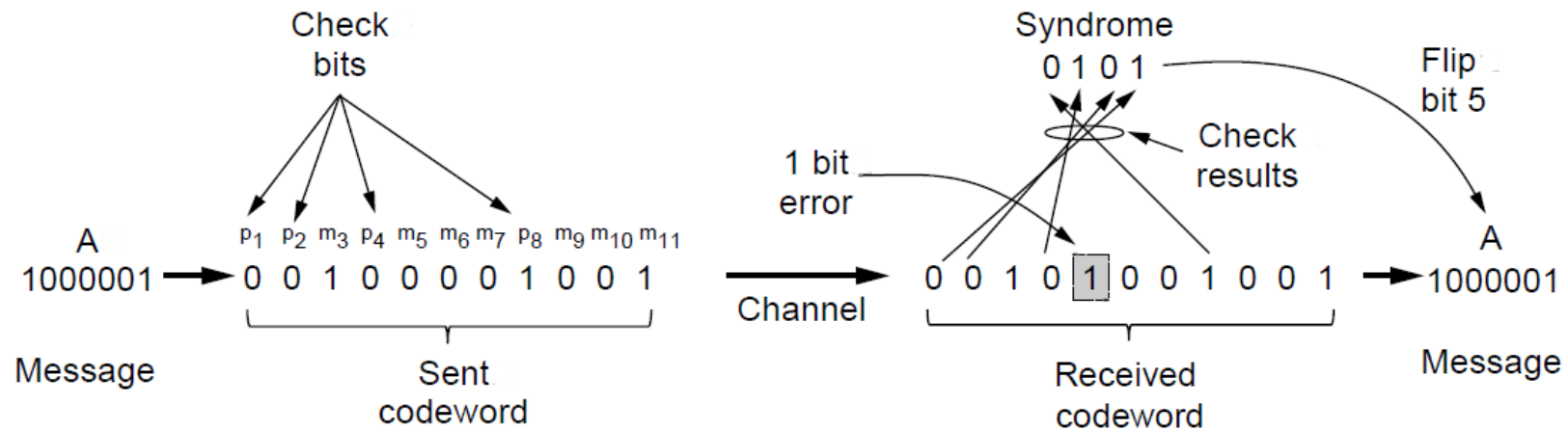
Error located!

- = 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

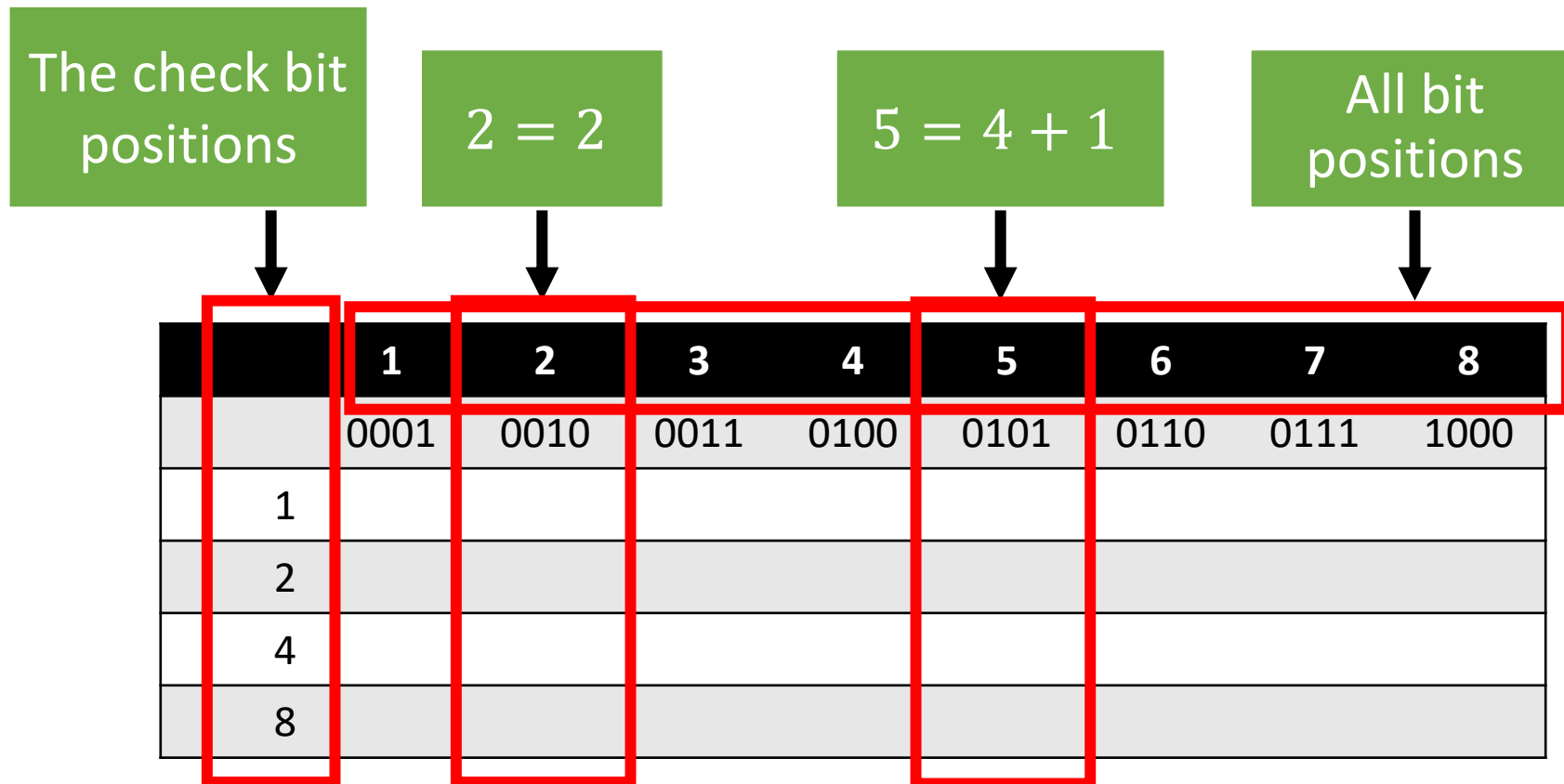
positions: 123456789...

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.



Hamming codes

An example

1. Expand all bit locations into powers of two.

	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

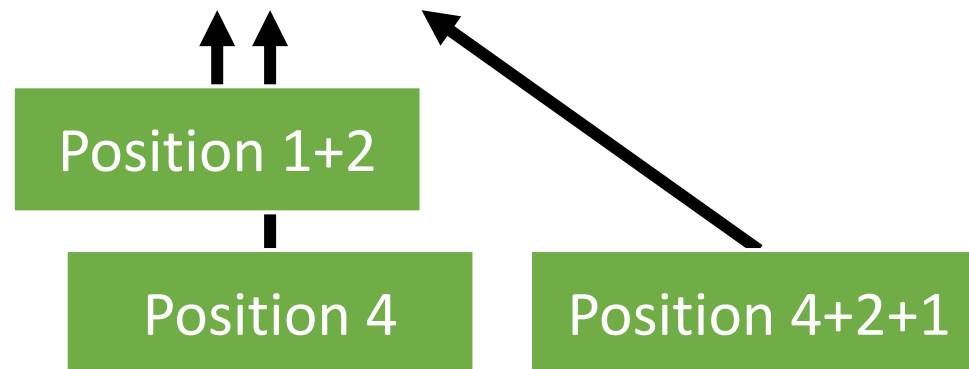
2. Calculate the parity bit using all bits that have 2^i in their expansion

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: 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** **101** **0101**

positions: **1**23456789...

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**0**1** **0**1**0**1

positions: **1**23456789...

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: 1**11**_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: 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

codeword: 11111**1**100101

positions: 123456789...

Computer *error syndrome*:

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



Single-bit error

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



Computer *error syndrome*:

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

codeword: 111111**1**100101

positions: 123456789...

Single-bit error



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
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

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

Computer Networks

X_400487

Lecture 3

Chapter 3: The Data Link Layer—Part 2



Lecturer: Jesse Donkervliet



Extra Slides

Convolutional codes

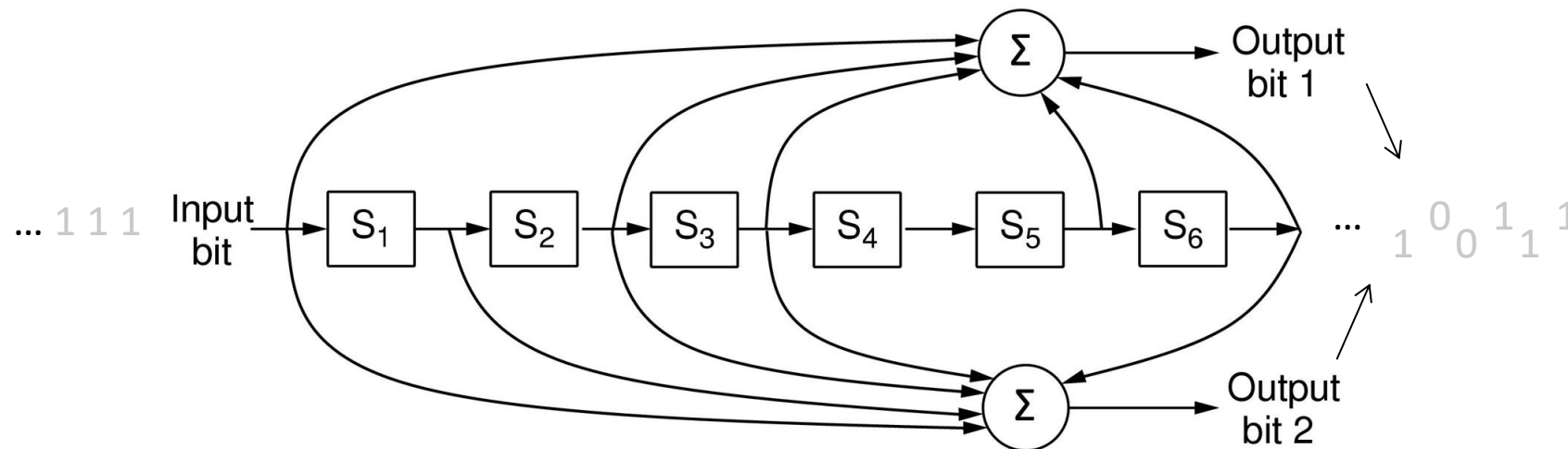
Different from *systematic* codes and *block* codes

Operates on a stream of bits, keeping internal state.

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

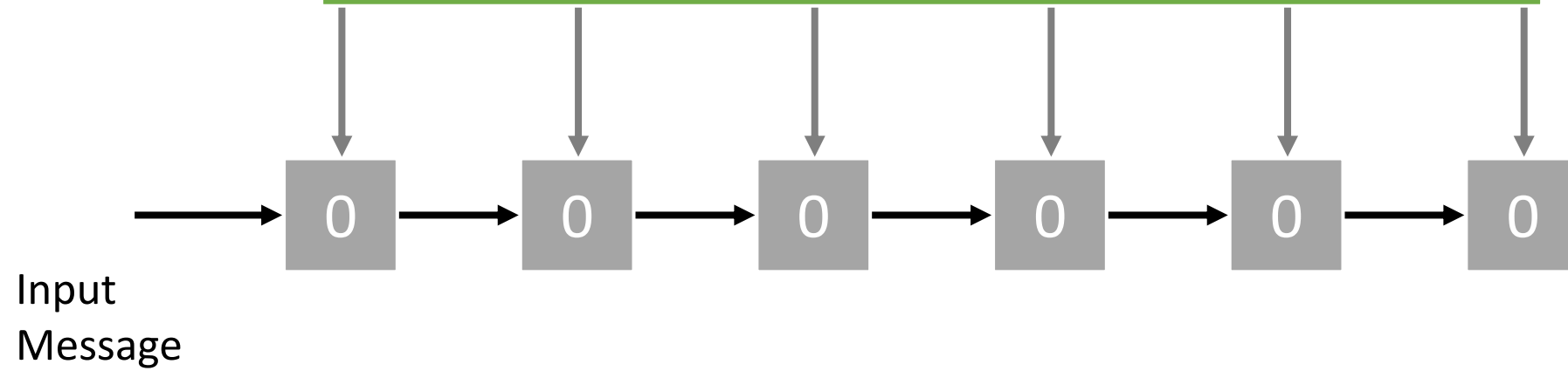
Bits are decoded with the Viterbi algorithm.

Determines most likely input for given output.

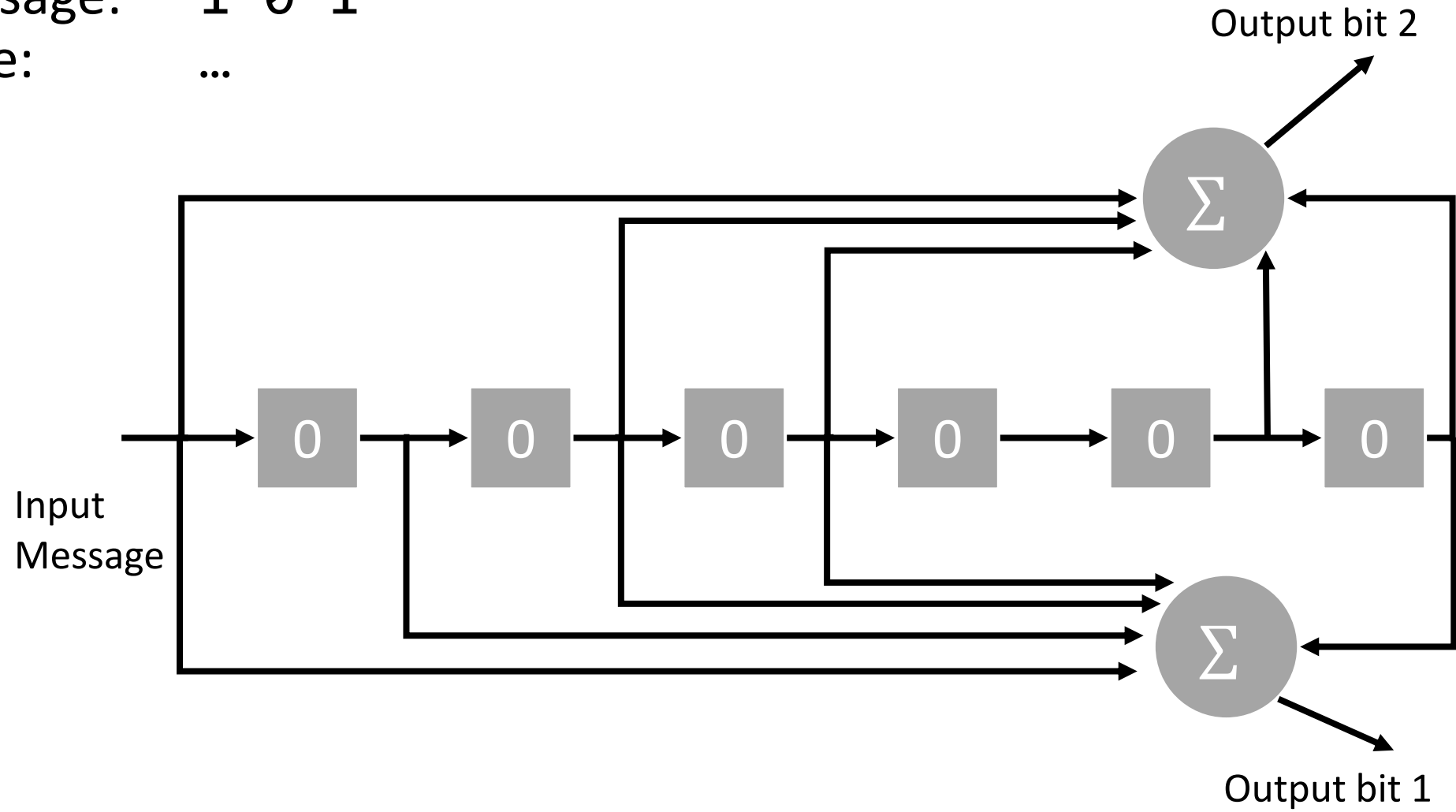


Popular NASA binary convolutional code (rate = $\frac{1}{2}$) used in 802.11 $k = 7$

Shift registers. Can be initialized with other values

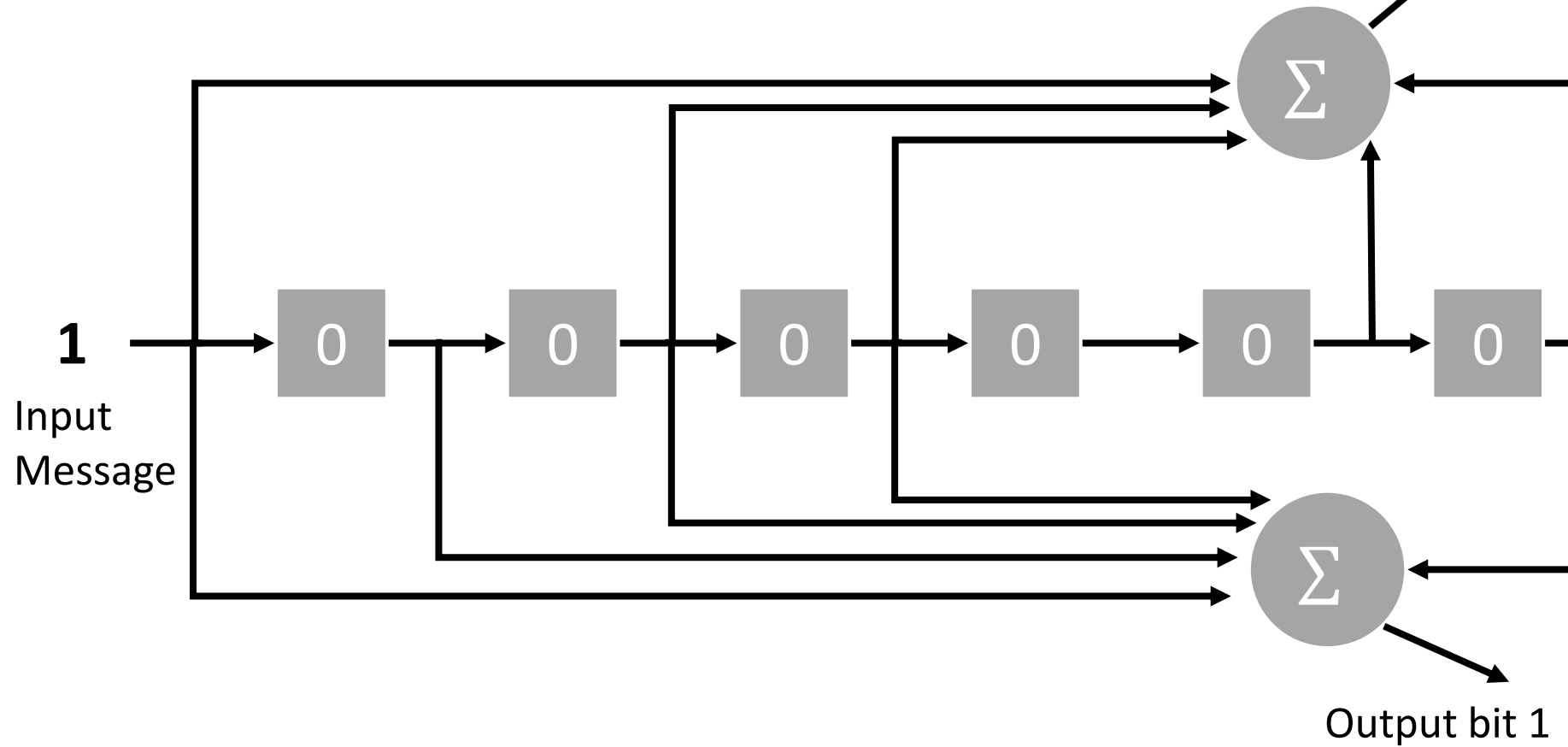


Message: 1 0 1
Code: ...



Message: 1 0 1
Code: 11...

$$1+0+0+0+0 = 1$$

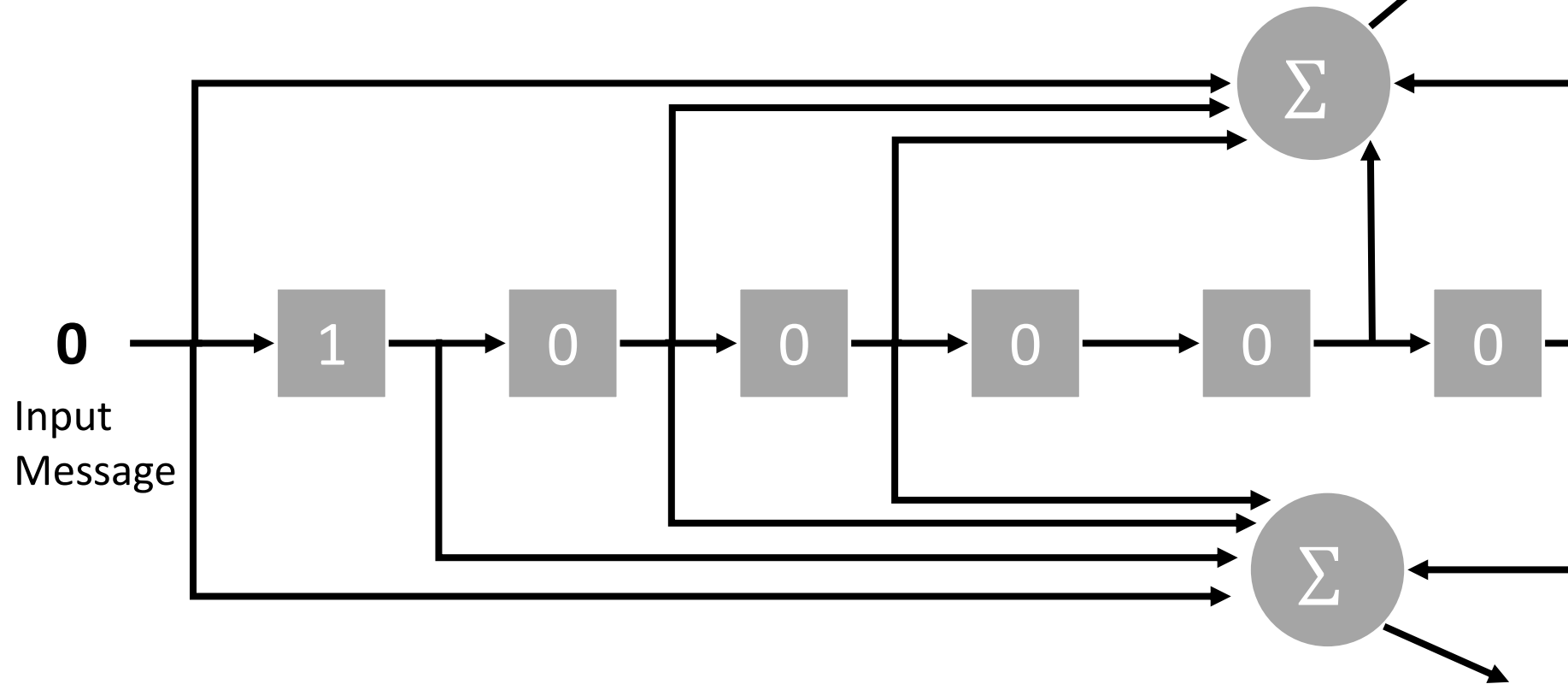


$$1+0+0+0+0 = 1$$

Message: 1 0 1
Code: 1101...

$$0+0+0+0+0 = 0$$

Output bit 1

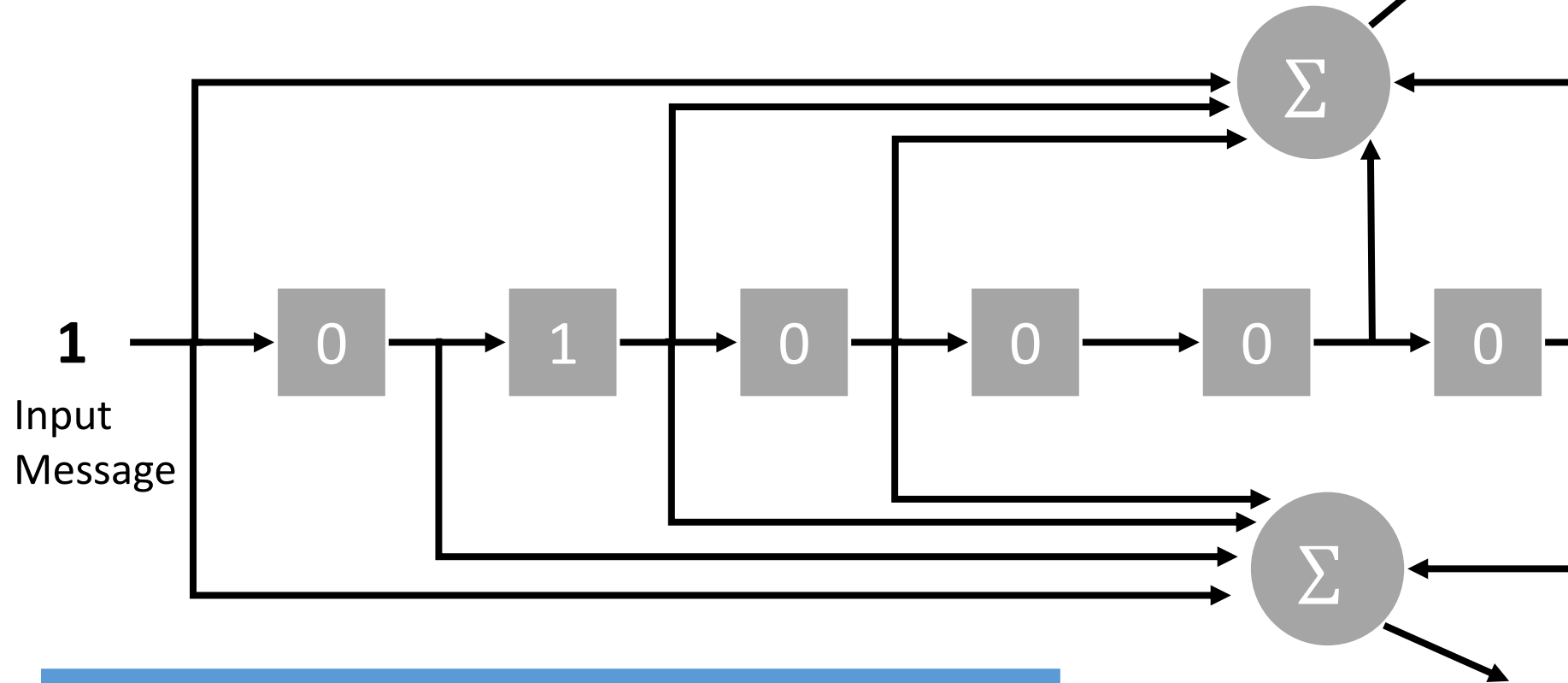


Output bit 2

$$0+1+0+0+0 = 1$$

Message: 1 0 1
Code: 110100

$$1+1+0+0+0 = 0$$



Q: Are we done dealing with errors?

$$1+0+1+0+0 = 0$$

$$1+0+1+0+0 = 0$$

Data Link Layer Summary

Framing (byte stuffing, bit stuffing, etc)

Guaranteed delivery **Q: When needed?**

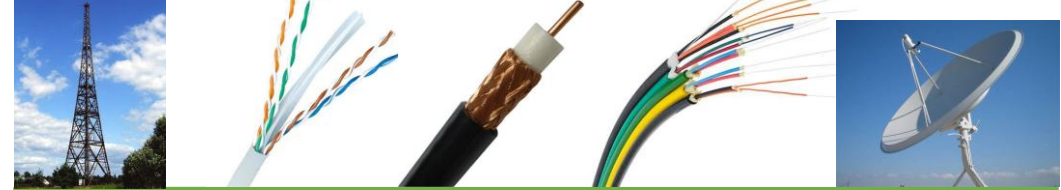
- Sequence numbers, acknowledgments, and retransmissions

Flow control **Q: When needed?**

- Stop-and-Wait
- Sliding Window

Error control **Q: When needed?**

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



A1: It depends on the physical medium



A2: It depends on the application

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

Q: Can you think of an example?