

# Computer Networks 1

## Lecture 4 - Framing

Fall 2022  
Joshua Reynolds

## Layer 2 - The Data Link Layer

Layer 2 is in charge of sending messages across physical connections.

It includes machines that receive a message on one connection, and send it out on the other.

The messages are called **datagrams**

# Abstracting the Physical Layer at the Data Link Layer

At the Data Link Layer, it doesn't matter how symbols are transmitted, received, or interpreted.

For any kind of connection, we care about:

- Error frequency
- Latency
- Bitrate
- Uptime

With this information, we can build communication between any two devices - regardless of how they are actually connected.

# Example: The Ethernet Protocol Family

Ethernet protocols are Data Link layer protocols

Ethernet protocols can be spoken over twisted pairs, coaxial cable, or fiber optics

It's all called Ethernet!

# Ethernet Autonegotiation

When you plug in a copper Ethernet cable in, how does it decide what bitrate it is going to transmit at? 10Mbps? 1Gbps?

Whenever an Ethernet port is idle, it sends a 100-200 ns pulse every 16ms called a Link Integrity Test (LIT).

When two ports hear each other's LIT for the first time, they enter autonegotiation, where each side declares the fastest speed it supports, and whether it can transmit and receive at the same time (duplex).

# 3-Way Rope Demo

Create a protocol to:

- Detect when someone connects to the 3-way rope

# 3-Way Rope Demo

Create a protocol to:

- Detect when someone connects to the 3-way rope
- Send a number between 1 and 5 to the other 2 people

# 3-Way Rope Demo

Create a protocol to:

- Detect when someone connects to the 3-way rope
- Send a number between 1 and 5 to the other 2 people
- Send a number between 1 and 5 to only one other person



# 3-Way Rope Demo

Create a protocol to:

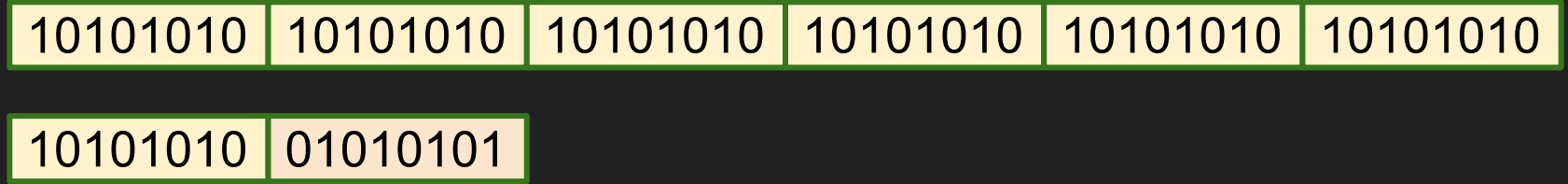
- Detect when someone connects to the 3-way rope
- Send a number between 1 and 5 to the other 2 people
- Send a number between 1 and 5 to only one other person
- Handle collisions when two or three people want to send at once

# Framing - How to tell when messages begin and end

Includes a chance to sync clocks in a self-clocking signal before important data flows

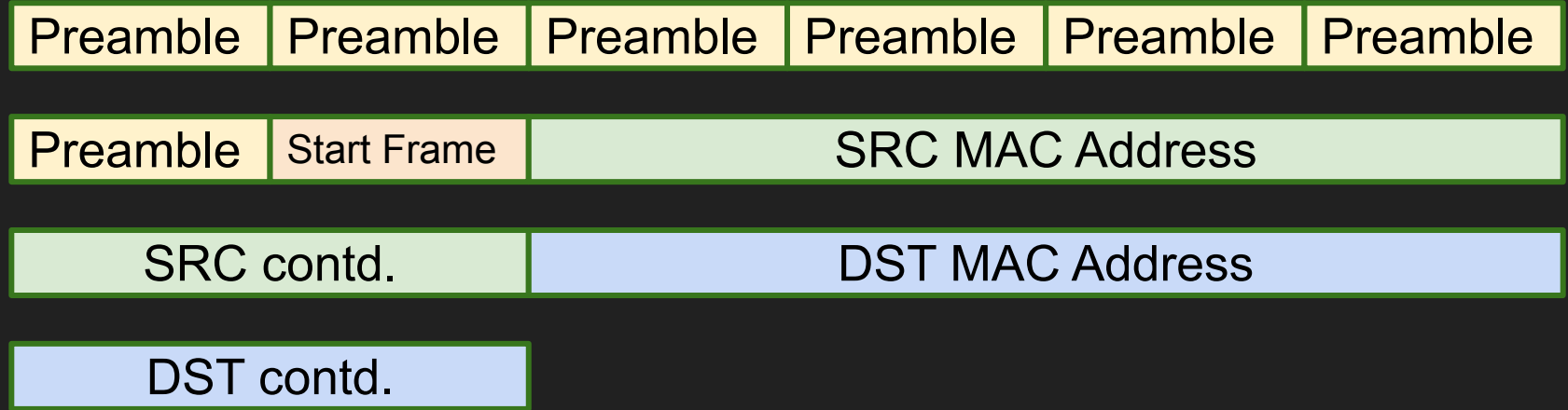
A **frame** is a sequence of bytes all sent together holding a chunk of information, and some error-correction.

# Ethernet Frames - Preamble



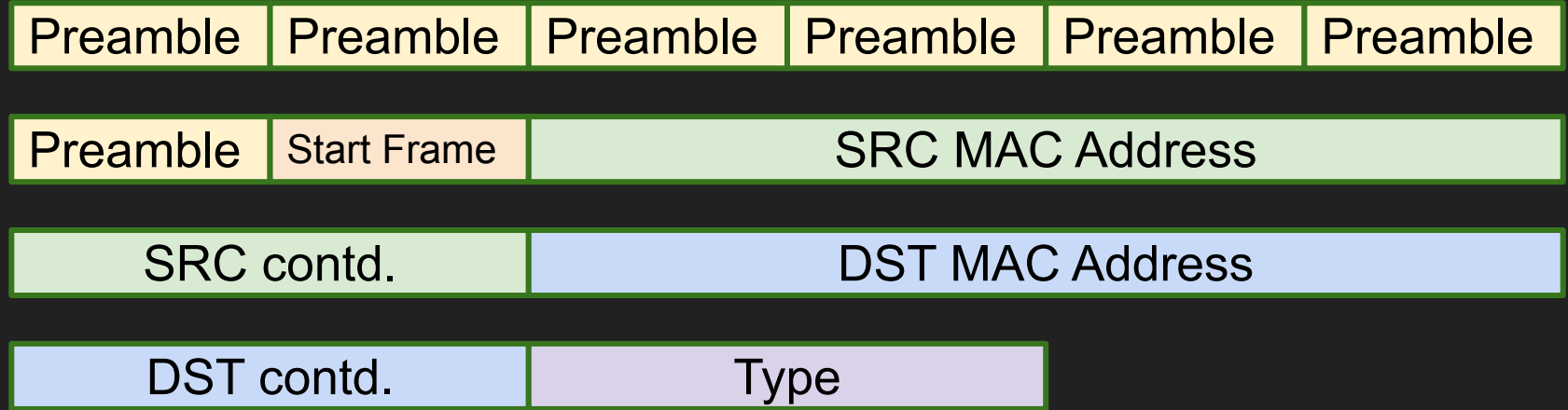
7 bytes of *10101010* followed by one byte of *01010101*  
called the **Start Frame Delimiter**

# Ethernet Frames - MAC Addresses



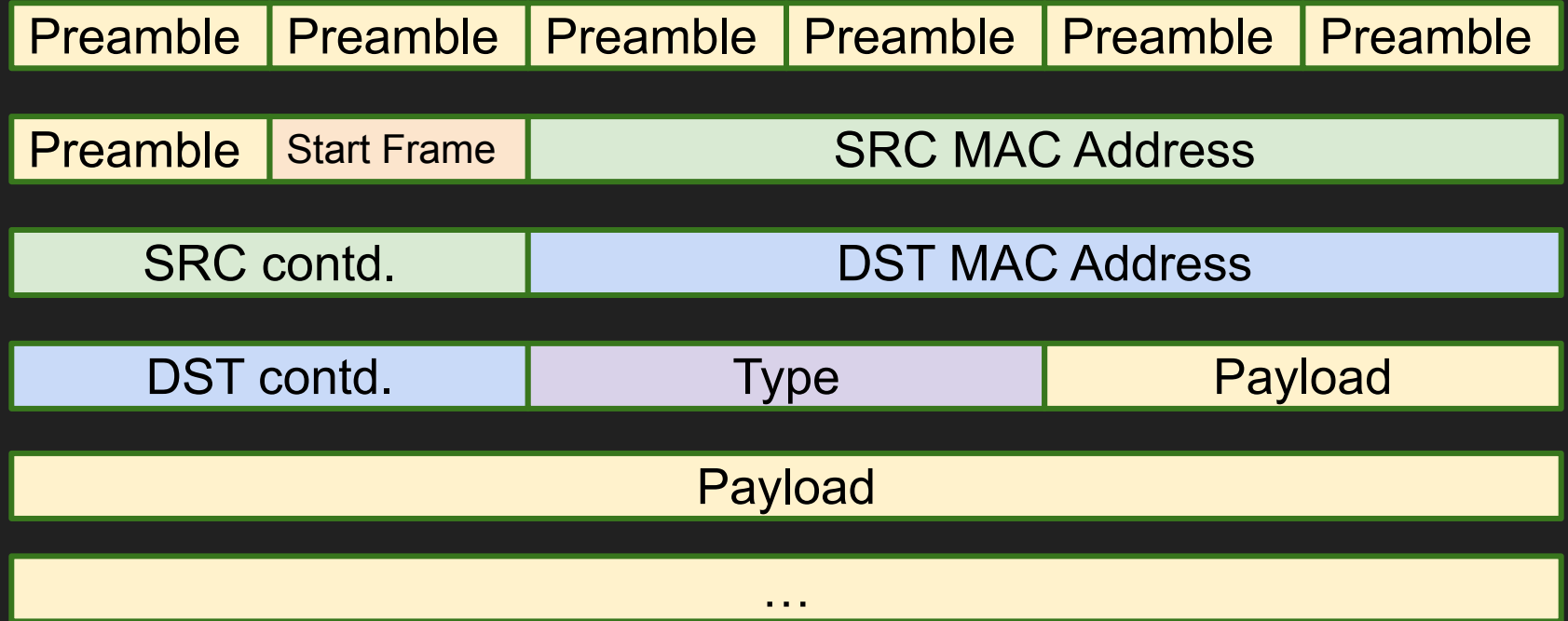
MAC addresses are 6 bytes long. We abbreviate a source address as SRC and a destination address as DST

# Ethernet Frames - Ethertype

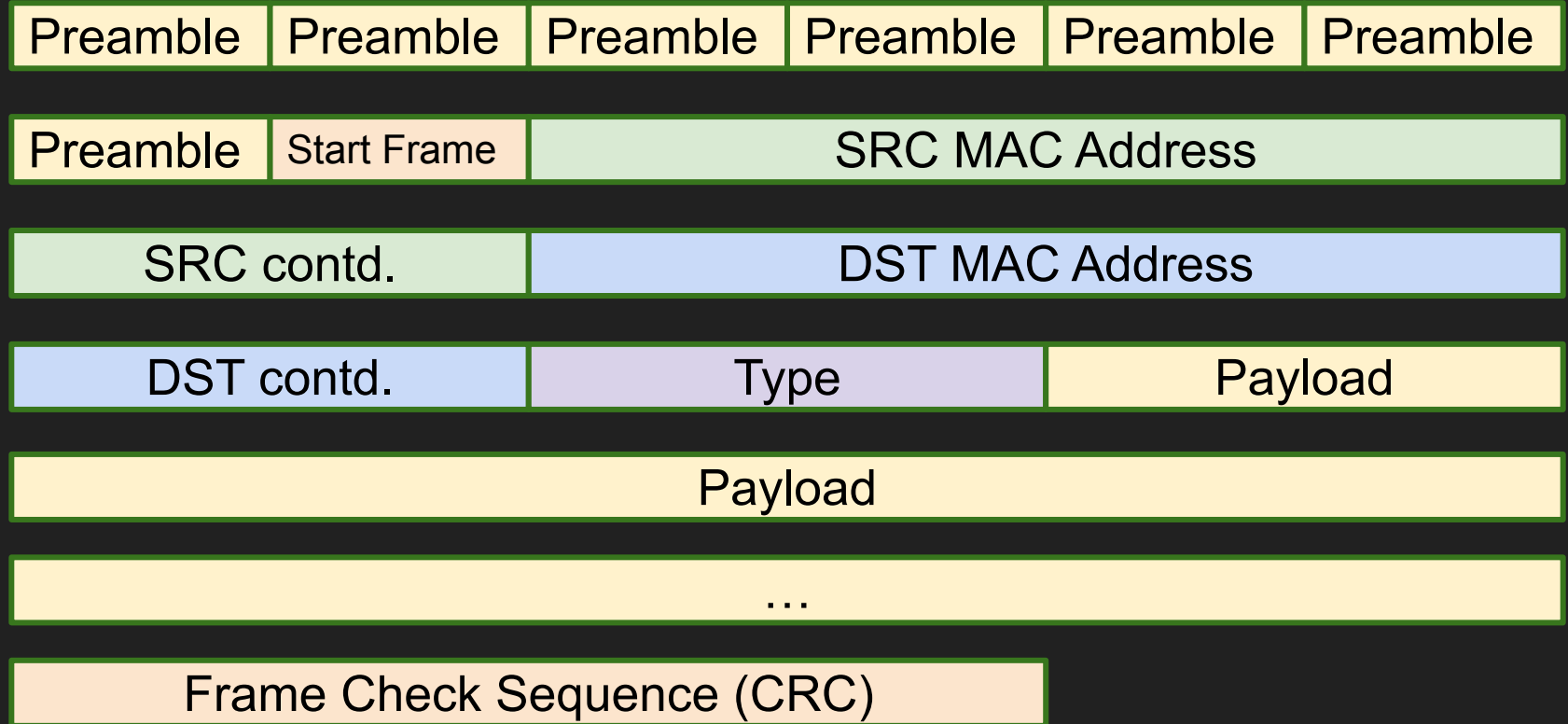


MAC addresses are 6 bytes long. We abbreviate a source address as SRC and a destination address as DST

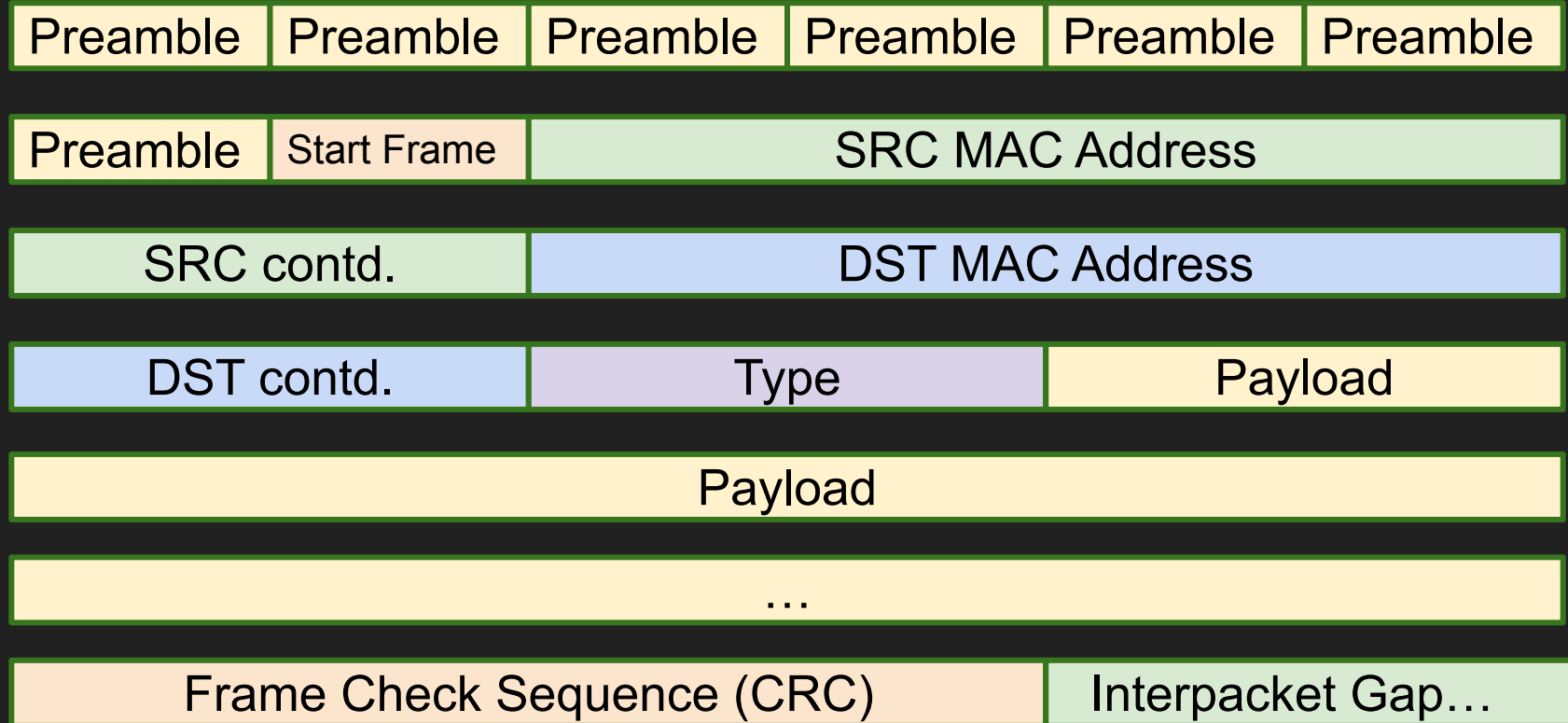
# Ethernet Frames - Payload



# Ethernet Frames - Error Detection CRC “Checksum”



# Ethernet Frames - Interpacket Gap (12 bytes)

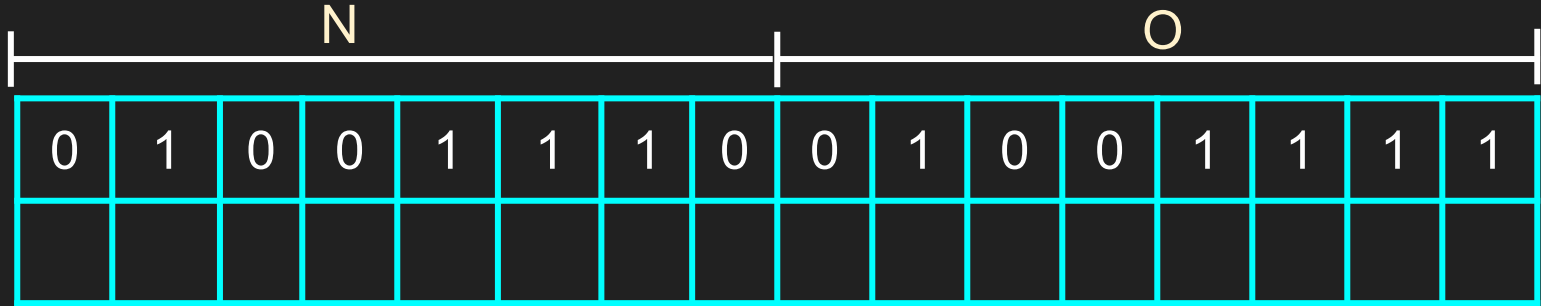




# Wireshark Ethernet Frame Demo - [wireshark.org](https://wireshark.org)



# Cyclic Redundancy Check (CRC)



# Cyclic Redundancy Check (CRC)

N								O							
0	1	0	0	1	1	1	0	0	1	0	0	1	1	1	1
-	$x^{12}$	-	-	$x^1_1$	$x^{10}$	$x^9$	-	-	$x^6$	-	-	$x^3$	$x^2$	$x^1$	$x^0$

# Cyclic Redundancy Check (CRC)

N								O								CRC		
0	1	0	0	1	1	1	0	0	1	0	0	1	1	1	1	0	0	0
-	$x^{17}$	-	-	$x^{14}$	$x^1_3$	$x^1_2$	-	-	$x^9$	-	-	$x^6$	$x^5$	$x^4$	$x^3$	-	-	-

Example Divisor Polynomial Used in GSM:

$$x^3 + x + 1$$

Ethernet Divisor Polynomial:

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

# Generating a 3-bit CRC Code for the Message “NO”

$$x^3 + x + 1 \overline{) x^{17} + x^{14} + x^{13} + x^{12} + x^9 + x^6 + x^5 + x^4 + x^3}$$

Goal:

Find the remainder

Special Rule:

Scalar addition and subtraction both work like XOR

# Generating a 3-bit CRC Code for the Message “NO”

$$\begin{array}{r} x^3 + x + 1 \quad | \quad x^{17} + x^{14} + x^{13} + x^{12} + x^9 + x^6 + x^5 + x^4 + x^3 \\ (x^{14}) \quad \quad \quad \underline{x^{17} + x^{15} + x^{14}} \end{array}$$

Goal:

Find the remainder

Special Rule:

Scalar addition and subtraction both work like XOR

# Generating a 3-bit CRC Code for the Message “NO”

$$\begin{array}{r} x^3 + x + 1 \quad | \quad x^{17} + x^{14} + x^{13} + x^{12} + x^9 + x^6 + x^5 + x^4 + x^3 \\ (x^{14}) \quad \quad \quad x^{17} + x^{15} + x^{14} \\ \hline x^{15} + x^{13} + x^{12} + x^9 + x^6 + x^5 + x^4 + x^3 \end{array}$$

Goal:

Find the remainder

Special Rule:

Scalar addition and subtraction both work like XOR

# Generating a 3-bit CRC Code for the Message “NO”

$$\begin{array}{r}
 x^3 + x + 1 \overline{) x^{17} + x^{14} + x^{13} + x^{12} + x^9 + x^6 + x^5 + x^4 + x^3} \\
 \underline{(x^{14}) \quad x^{17} + x^{15} x^{14}} \\
 x^{15} + x^{13} + x^{12} + x^9 + x^6 + x^5 + x^4 + x^3 \\
 \underline{(x^{12}) \quad x^{15} + x^{13} + x^{12}} \\
 x^9 + x^6 + x^5 + x^4 + x^3
 \end{array}$$

Goal:

Find the remainder

Special Rule:

Scalar addition and subtraction both work like XOR



# Generating a 3-bit CRC Code for the Message “NO”

$$\begin{array}{r}
 x^3 + x + 1 \overline{) x^{17} + x^{14} + x^{13} + x^{12} + x^9 + x^6 + x^5 + x^4 + x^3} \\
 \underline{(x^{14}) \quad x^{17} + x^{15} + x^{14}} \phantom{+ x^{13} + x^{12} + x^9 + x^6 + x^5 + x^4 + x^3} \\
 \phantom{x^3 + x + 1 \overline{)} x^{15} + x^{13} + x^{12} + x^9 + x^6 + x^5 + x^4 + x^3} \\
 \underline{(x^{12}) \quad x^{15} + x^{13} + x^{12}} \phantom{+ x^9 + x^6 + x^5 + x^4 + x^3} \\
 \phantom{x^3 + x + 1 \overline{)} x^9 + x^6 + x^5 + x^4 + x^3} \\
 \phantom{x^3 + x + 1 \overline{)} x^9 + x^7 + x^6} \\
 \hline
 \phantom{x^3 + x + 1 \overline{)} }
 \end{array}$$

Goal:

Find the remainder

Special Rule:

Scalar addition and subtraction both work like XOR

# Generating a 3-bit CRC Code for the Message “NO”

$$\begin{array}{r}
 x^3 + x + 1 \overline{) x^{17} + x^{14} + x^{13} + x^{12} + x^9 + x^6 + x^5 + x^4 + x^3} \\
 \underline{(x^{14}) \quad x^{17} + x^{15} x^{14}} \\
 x^{15} + x^{13} + x^{12} + x^9 + x^6 + x^5 + x^4 + x^3 \\
 \underline{(x^{12}) \quad x^{15} + x^{13} + x^{12}} \\
 x^9 + x^6 + x^5 + x^4 + x^3 \\
 \underline{(x^6) \quad x^9 + x^7 + x^6} \\
 x^7 + x^5 + x^4 + x^3
 \end{array}$$

Goal:

Find the remainder

Special Rule:

Scalar addition and subtraction both work like XOR

# Generating a 3-bit CRC Code for the Message “NO”

$$\begin{array}{r}
 x^3 + x + 1 \overline{) x^{17} + x^{14} + x^{13} + x^{12} + x^9 + x^6 + x^5 + x^4 + x^3} \\
 \underline{(x^{14}) \quad x^{17} + x^{15} + x^{14}} \phantom{+ x^{13} + x^{12} + x^9 + x^6 + x^5 + x^4 + x^3} \\
 \phantom{x^{17} + } x^{15} + x^{13} + x^{12} + x^9 + x^6 + x^5 + x^4 + x^3 \\
 \underline{(x^{12}) \quad x^{15} + x^{13} + x^{12}} \phantom{+ x^9 + x^6 + x^5 + x^4 + x^3} \\
 \phantom{x^{17} + x^{15} + } x^9 + x^6 + x^5 + x^4 + x^3 \\
 \underline{(x^6) \quad x^9 + x^7 + x^6} \phantom{+ x^5 + x^4 + x^3} \\
 \phantom{x^{17} + x^{15} + x^{13} + } x^7 + x^5 + x^4 + x^3 \\
 \underline{(x^4) \quad x^7 + x^5 + x^4} \phantom{+ x^3} \\
 \phantom{x^{17} + x^{15} + x^{13} + x^7 + } x^3
 \end{array}$$

Goal:

Find the remainder

Special Rule:

Scalar addition and subtraction both work like XOR

# Generating a 3-bit CRC Code for the Message “NO”

$$\begin{array}{r}
 x^3 + x + 1 \overline{) x^{17} + x^{14} + x^{13} + x^{12} + x^9 + x^6 + x^5 + x^4 + x^3} \\
 \underline{(x^{14}) \quad x^{17} + x^{15} x^{14}} \\
 x^{15} + x^{13} + x^{12} + x^9 + x^6 + x^5 + x^4 + x^3 \\
 \underline{(x^{12}) \quad x^{15} + x^{13} + x^{12}} \\
 x^9 + x^6 + x^5 + x^4 + x^3 \\
 \underline{(x^6) \quad x^9 + x^7 + x^6} \\
 x^7 + x^5 + x^4 + x^3 \\
 \underline{(x^4) \quad x^7 + x^5 + x^4} \\
 x^3 \\
 \underline{(x^0) \quad x^3 + x + 1}
 \end{array}$$

Goal:

Find the remainder

Special Rule:

Scalar addition and subtraction both work like XOR

# Generating a 3-bit CRC Code for the Message “NO”

$$\begin{array}{r}
 x^3 + x + 1 \overline{) x^{17} + x^{14} + x^{13} + x^{12} + x^9 + x^6 + x^5 + x^4 + x^3} \\
 \underline{(x^{14}) \quad x^{17} + x^{15} x^{14}} \\
 x^{15} + x^{13} + x^{12} + x^9 + x^6 + x^5 + x^4 + x^3 \\
 \underline{(x^{12}) \quad x^{15} + x^{13} + x^{12}} \\
 x^9 + x^6 + x^5 + x^4 + x^3 \\
 \underline{(x^6) \quad x^9 + x^7 + x^6} \\
 x^7 + x^5 + x^4 + x^3 \\
 \underline{(x^4) \quad x^7 + x^5 + x^4} \\
 x^3 \\
 \underline{(x^0) \quad x^3 + x + 1}
 \end{array}$$

Goal:

Find the remainder

Special Rule:

Scalar addition and subtraction both work like XOR

What? This works because addition and subtraction are the same :)

# Generating a 3-bit CRC Code for the Message “NO”

$$\begin{array}{r}
 x^3 + x + 1 \overline{) x^{17} + x^{14} + x^{13} + x^{12} + x^9 + x^6 + x^5 + x^4 + x^3} \\
 \underline{(x^{14}) \quad x^{17} + x^{15} x^{14}} \\
 x^{15} + x^{13} + x^{12} + x^9 + x^6 + x^5 + x^4 + x^3 \\
 \underline{(x^{12}) \quad x^{15} + x^{13} + x^{12}} \\
 x^9 + x^6 + x^5 + x^4 + x^3 \\
 \underline{(x^6) \quad x^9 + x^7 + x^6} \\
 x^7 + x^5 + x^4 + x^3 \\
 \underline{(x^4) \quad x^7 + x^5 + x^4} \\
 x^3 \\
 \underline{(x^0) \quad x^3 + x + 1} \\
 x + 1
 \end{array}$$

Goal:

Find the remainder

Special Rule:

Scalar addition and subtraction both work like XOR

# Generating a 3-bit CRC Code for the Message “NO”

$$\begin{array}{r}
 x^3 + x + 1 \overline{) x^{17} + x^{14} + x^{13} + x^{12} + x^9 + x^6 + x^5 + x^4 + x^3} \\
 \underline{(x^{14}) \quad x^{17} + x^{15} x^{14}} \\
 x^{15} + x^{13} + x^{12} + x^9 + x^6 + x^5 + x^4 + x^3 \\
 \underline{(x^{12}) \quad x^{15} + x^{13} + x^{12}} \\
 x^9 + x^6 + x^5 + x^4 + x^3 \\
 \underline{(x^6) \quad x^9 + x^7 + x^6} \\
 x^7 + x^5 + x^4 + x^3 \\
 \underline{(x^4) \quad x^7 + x^5 + x^4} \\
 x^3 \\
 \underline{(x^0) \quad x^3 + x + 1} \\
 \boxed{x + 1}
 \end{array}$$

Remainder →

Goal:  
Find the remainder

Special Rule:  
Scalar addition and subtraction both work like XOR

# Generating a 3-bit CRC Code for the Message “NO”

$$\begin{array}{r}
 x^3 + x + 1 \overline{) x^{17} + x^{14} + x^{13} + x^{12} + x^9 + x^6 + x^5 + x^4 + x^3} \\
 \underline{(x^{14}) \quad x^{17} + x^{15} x^{14}} \\
 x^{15} + x^{13} + x^{12} + x^9 + x^6 + x^5 + x^4 + x^3 \\
 \underline{(x^{12}) \quad x^{15} + x^{13} + x^{12}} \\
 x^9 + x^6 + x^5 + x^4 + x^3 \\
 \underline{(x^6) \quad x^9 + x^7 + x^6} \\
 x^7 + x^5 + x^4 + x^3 \\
 \underline{(x^4) \quad x^7 + x^5 + x^4} \\
 x^3 \\
 \underline{(x^0) \quad x^3 + x + 1} \\
 \boxed{x + 1}
 \end{array}$$

Remainder →

Goal:

Find the remainder

Special Rule:

Scalar addition and subtraction both work like XOR

CRC: 011



# A 3-bit Cyclic Redundancy Check (CRC)

N								O								CRC		
0	1	0	0	1	1	1	0	0	1	0	0	1	1	1	1	0	1	1
-	$x^{17}$	-	-	$x^{14}$	$x^1_3$	$x^1_2$	-	-	$x^9$	-	-	$x^6$	$x^5$	$x^4$	$x^3$	-	$x^1$	$x^0$

1. Put 011 in the CRC and transmit
2. When the receiver gets this, they also divide by  $x^3+x+1$
3. We made it so the remainder is 0
4. If the remainder is not zero, there was an error

This is really easy to calculate if we use bit shifts and XOR operators to represent polynomial division

# Efficiently working out a CRC with bit shifts and XOR

Message: "CRC" Polynomial: 0x13

# Efficiently working out a CRC with bit shifts and XOR

Message: "CRC" Polynomial: 0x13

CRC in 8-bit ASCII: 010000110101001001000011

# Efficiently working out a CRC with bit shifts and XOR

Message: "CRC" Polynomial: 0x13

CRC in 8-bit ASCII: 010000110101001001000011

Polynomial in binary:

# Efficiently working out a CRC with bit shifts and XOR

Message: "CRC" Polynomial: 0x13

CRC in 8-bit ASCII: 010000110101001001000011

Polynomial in binary: 10011

# Efficiently working out a CRC with bit shifts and XOR

Message: "CRC" Polynomial: 0x13

CRC in 8-bit ASCII: 010000110101001001000011

Polynomial in binary: 10011

CRC Bit Length Must Be:

# Efficiently working out a CRC with bit shifts and XOR

Message: "CRC" Polynomial: 0x13

CRC in 8-bit ASCII: 010000110101001001000011

Polynomial in binary: 10011

CRC Bit Length Must Be (aka the max length of the remainder):



# Efficiently working out a CRC with bit shifts and XOR

Message: "CRC" Polynomial: 0x13

CRC in 8-bit ASCII: 010000110101001001000011

Polynomial in binary: 10011

CRC Bit Length Must Be (aka the max length of the remainder): 4 bits

# Efficiently working out a CRC with bit shifts and XOR

Message: "CRC" Polynomial: 0x13

CRC in 8-bit ASCII: 010000110101001001000011

Polynomial in binary: 10011

CRC Bit Length Must Be (aka the max length of the remainder): 4 bits

Message with 4 empty bits ready to hold CRC in binary:

# Efficiently working out a CRC with bit shifts and XOR

Message: "CRC" Polynomial: 0x13

CRC in 8-bit ASCII: 010000110101001001000011

Polynomial in binary: 10011

CRC Bit Length Must Be (aka the max length of the remainder): 4 bits

Message with 4 empty bits ready to hold CRC in binary:

0100001101010010010000110000


# Efficiently working out a CRC with bit shifts and XOR

Algorithm: while the message is still longer than 4 bits:

1. If the first bit is 1, xor by the polynomial padded to the full length with zeros
2. Shift left until the first bit is 1

## Efficiently working out a CRC with bit shifts and XOR

10011 | 0100001101010010010000110000

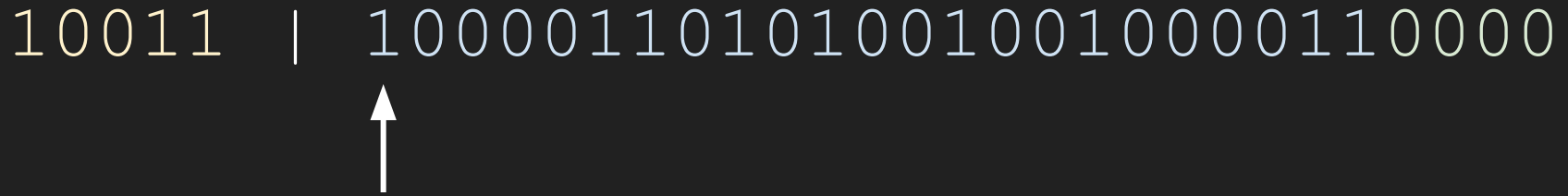


Algorithm: while the message is still longer than 4 bits:

1. If the first bit is 1, xor by the polynomial padded to the full length with zeros
2. **Shift left until the first bit is 1**

## Efficiently working out a CRC with bit shifts and XOR

10011 | 100001101010010010000110000



Algorithm: while the message is still longer than 4 bits:

1. **If the first bit is 1, xor by the polynomial padded to the full length with zeros**
2. Shift left until the first bit is 1

# Efficiently working out a CRC with bit shifts and XOR

```
10011 | 100001101010010010000110000  
      ⊕ 10011000000000000000000000
```

Algorithm: while the message is still longer than 4 bits:

1. If the first bit is 1, xor by the polynomial padded to the full length with zeros
2. Shift left until the first bit is 1

## Efficiently working out a CRC with bit shifts and XOR

10011 | 000111101010010010000110000

Algorithm: while the message is still longer than 4 bits:

1. If the first bit is 1, xor by the polynomial padded to the full length with zeros
2. **Shift left until the first bit is 1**



## Efficiently working out a CRC with bit shifts and XOR

10011 | 111101010010010000110000

Algorithm: while the message is still longer than 4 bits:

1. **If the first bit is 1, xor by the polynomial padded to the full length with zeros**
2. Shift left until the first bit is 1

## Efficiently working out a CRC with bit shifts and XOR

10011 | 111101010010010000110000  
 $\oplus$  100110000000000000000000

Algorithm: while the message is still longer than 4 bits:

1. **If the first bit is 1, xor by the polynomial padded to the full length with zeros**
2. Shift left until the first bit is 1

## Efficiently working out a CRC with bit shifts and XOR

10011 | 011011010010010000110000

Algorithm: while the message is still longer than 4 bits:

1. If the first bit is 1, xor by the polynomial padded to the full length with zeros
2. **Shift left until the first bit is 1**

## Efficiently working out a CRC with bit shifts and XOR

10011 | 11011010010010000110000

Algorithm: while the message is still longer than 4 bits:

1. **If the first bit is 1, xor by the polynomial padded to the full length with zeros**
2. Shift left until the first bit is 1

# Efficiently working out a CRC with bit shifts and XOR

```
10011 | 11011010010010000110000
      ⊕ 100110000000000000000000
```

Algorithm: while the message is still longer than 4 bits:

1. If the first bit is 1, xor by the polynomial padded to the full length with zeros
2. Shift left until the first bit is 1

## Efficiently working out a CRC with bit shifts and XOR

10011 | 01000010010010000110000

Algorithm: while the message is still longer than 4 bits:

1. **If the first bit is 1, xor by the polynomial padded to the full length with zeros**
2. Shift left until the first bit is 1

## Efficiently working out a CRC with bit shifts and XOR

$$\begin{array}{r} 10011 \quad | \quad 10000100100100000110000 \\ \oplus 100110000000000000000000000000000 \end{array}$$

Algorithm: while the message is still longer than 4 bits:

1. **If the first bit is 1, xor by the polynomial padded to the full length with zeros**
2. Shift left until the first bit is 1

## Efficiently working out a CRC with bit shifts and XOR

10011 | 0001110010010000110000

Algorithm: while the message is still longer than 4 bits:

1. If the first bit is 1, xor by the polynomial padded to the full length with zeros
2. Shift left until the first bit is 1



# Efficiently working out a CRC with bit shifts and XOR

$$\begin{array}{r} 10011 \quad | \quad 1110010010000110000 \\ \oplus 1001100000000000000000 \end{array}$$

Algorithm: while the message is still longer than 4 bits:

1. If the first bit is 1, xor by the polynomial padded to the full length with zeros
2. Shift left until the first bit is 1

## Efficiently working out a CRC with bit shifts and XOR

10011 | 0111110010000110000

Algorithm: while the message is still longer than 4 bits:

1. If the first bit is 1, xor by the polynomial padded to the full length with zeros
2. Shift left until the first bit is 1

## Efficiently working out a CRC with bit shifts and XOR

$$\begin{array}{r} 10011 \quad | \quad 111110010000110000 \\ \oplus 10011000000000000000 \end{array}$$

Algorithm: while the message is still longer than 4 bits:

1. If the first bit is 1, xor by the polynomial padded to the full length with zeros
2. Shift left until the first bit is 1

## Efficiently working out a CRC with bit shifts and XOR

10011 | 11000010000110000

Algorithm: while the message is still longer than 4 bits:

1. If the first bit is 1, xor by the polynomial padded to the full length with zeros
2. Shift left until the first bit is 1

## Efficiently working out a CRC with bit shifts and XOR

$$\begin{array}{r} 10011 \quad | \quad 11000010000110000 \\ \oplus 1001100000000000000 \end{array}$$

Algorithm: while the message is still longer than 4 bits:

1. If the first bit is 1, xor by the polynomial padded to the full length with zeros
2. Shift left until the first bit is 1

# Efficiently working out a CRC with bit shifts and XOR

10011 | 1011010000110000

Algorithm: while the message is still longer than 4 bits:

1. If the first bit is 1, xor by the polynomial padded to the full length with zeros
2. Shift left until the first bit is 1

## Efficiently working out a CRC with bit shifts and XOR

10011 | 1011010000110000

$\oplus$  1001100000000000

Algorithm: while the message is still longer than 4 bits:

1. If the first bit is 1, xor by the polynomial padded to the full length with zeros
2. Shift left until the first bit is 1

# Efficiently working out a CRC with bit shifts and XOR

10011 | 10110000110000

Algorithm: while the message is still longer than 4 bits:

1. If the first bit is 1, xor by the polynomial padded to the full length with zeros
2. Shift left until the first bit is 1



## Efficiently working out a CRC with bit shifts and XOR

10011 | 10110000110000

$\oplus$  10011000000000

Algorithm: while the message is still longer than 4 bits:

1. If the first bit is 1, xor by the polynomial padded to the full length with zeros
2. Shift left until the first bit is 1

# Efficiently working out a CRC with bit shifts and XOR

$$\begin{array}{r} 10011 \quad | \quad 101000110000 \\ \oplus 1001100000000 \end{array}$$

Algorithm: while the message is still longer than 4 bits:

1. If the first bit is 1, xor by the polynomial padded to the full length with zeros
2. Shift left until the first bit is 1

## Efficiently working out a CRC with bit shifts and XOR

10011 | 1110110000

$\oplus$  1001100000

Algorithm: while the message is still longer than 4 bits:

1. If the first bit is 1, xor by the polynomial padded to the full length with zeros
2. Shift left until the first bit is 1

# Efficiently working out a CRC with bit shifts and XOR

10011 | 111010000

$\oplus$  100110000

Algorithm: while the message is still longer than 4 bits:

1. If the first bit is 1, xor by the polynomial padded to the full length with zeros
2. Shift left until the first bit is 1

## Efficiently working out a CRC with bit shifts and XOR

10011 | 11100000

$\oplus$  10011000

Algorithm: while the message is still longer than 4 bits:

1. If the first bit is 1, xor by the polynomial padded to the full length with zeros
2. Shift left until the first bit is 1

# Efficiently working out a CRC with bit shifts and XOR

10011 | 1111000

$\oplus$  1001100

Algorithm: while the message is still longer than 4 bits:

1. If the first bit is 1, xor by the polynomial padded to the full length with zeros
2. Shift left until the first bit is 1

# Efficiently working out a CRC with bit shifts and XOR

10011 | 110100

$\oplus$  100110

Algorithm: while the message is still longer than 4 bits:

1. If the first bit is 1, xor by the polynomial padded to the full length with zeros
2. Shift left until the first bit is 1

# Efficiently working out a CRC with bit shifts and XOR

10011 | 10010

$\oplus$  10011

Algorithm: while the message is still longer than 4 bits:

1. If the first bit is 1, xor by the polynomial padded to the full length with zeros
2. Shift left until the first bit is 1



## Efficiently working out a CRC with bit shifts and XOR

10011 | 0001 ← REMAINDER 0x01

Algorithm: while the message is still longer than 4 bits:

1. If the first bit is 1, xor by the polynomial padded to the full length with zeros
2. Shift left until the first bit is 1

## Efficiently working out a CRC with bit shifts and XOR

10011 | 0001  $\leftarrow$  REMAINDER 0x01

SEND: **0100001101010010010000110001**

Algorithm: while the message is still longer than 4 bits:

1. If the first bit is 1, xor by the polynomial padded to the full length with zeros
2. Shift left until the first bit is 1

# Pseudocode for CRC Activity

```
def CRC(msg, polynomial):
```

Notice: You could start calculating a CRC while the message is still coming in!

Notice: You could start calculating a CRC while the message is still coming in!

Practice Problem:

- Msg = “?”
- Polynomial = 0x3

Notice: You could start calculating a CRC while the message is still coming in!

Practice Problem:

- Msg = “?”
- Polynomial = 0x3
- To send: 001111110

# Ethernet Collision Avoidance (CSMA/CD)

## Carrier Sense Multiple Access Collision Avoidance Algorithm:

1. Build a frame to transmit
2. If someone is currently transmitting
  - a. Wait till they stop
3. Transmit your frame
  - a. If there is no collision detected, you are done
4. If there is a collision, **jam** the transmission so everyone knows a collision happened.
5. Wait a **random amount of time\***, then goto step 2

# “Exponential Backoff” in Ethernet Collision Avoidance

If everyone tries to retransmit soon, we will just collide again.

If we ask who is going to go next, our questions could collide!

## **Exponential backoff algorithm:**

1. Let  $m = 0$ ,  $n = \text{bitrate}^{-1}$  (seconds per bit)
2. While the transmission still hasn't gone through:
  - a. Let  $k = \text{random integer between } 0 \text{ and } (2^m - 1), \text{ inclusive}$
  - b. Wait for  $k * n * 512$  seconds
  - c. Try transmitting again
  - d.  $m += 1$



# WiFi Framing

WiFi also uses a preamble and start-of-frame delimiter pattern

Uses a 16-bit CRC

# Wifi Frame Types

- **Management Frames**
  - Beacons, Authentication (passwords and keys), Probes
- **Control Frames**
  - ACK - I acknowledge that I got what was sent
    - **Block ACK** acknowledges many frames at once, for efficiency
  - RTS - Request to send, are you available to listen?
  - CTS - Cleared to send, everyone be quiet while they send
- **Data Frames**
  - Data (the actual data being sent)
  - Quality of Service (QoS) “set urgency level”

# WiFi Beacons and Probes

About every 100ms, a wireless **Access Point** declares the WiFi networks it has available in a beacon frame.

Each one has a Service Set Identifier (SSID)

It advertises which WiFi protocols are supported, and what kind of encryption is supported

Devices can also ask if a particular access point can see it by sending a “probe”

Your phone will periodically ask for wifi networks it has seen before

## 802.11 WiFi BSSID

WiFi uses 6-byte BSSIDs instead of 6-byte MAC addresses.

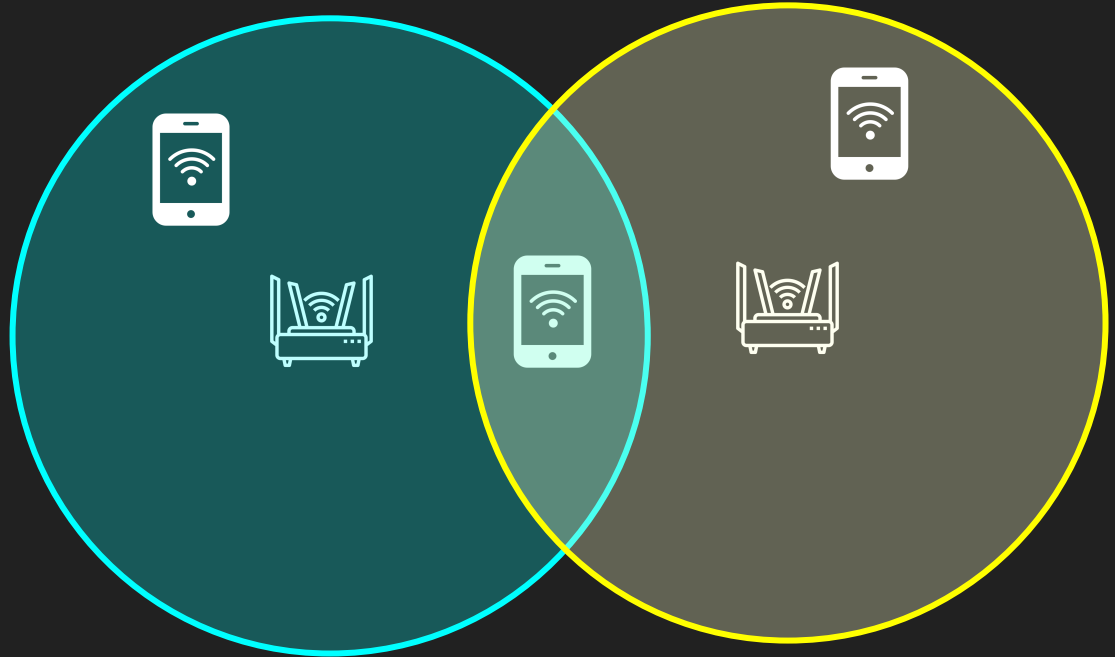
In practice, they are basically the same.

Like with MAC addresses, they can be changed with software

# 802.11 WiFi Multiple Access with Collision Avoidance

If I want to send something:

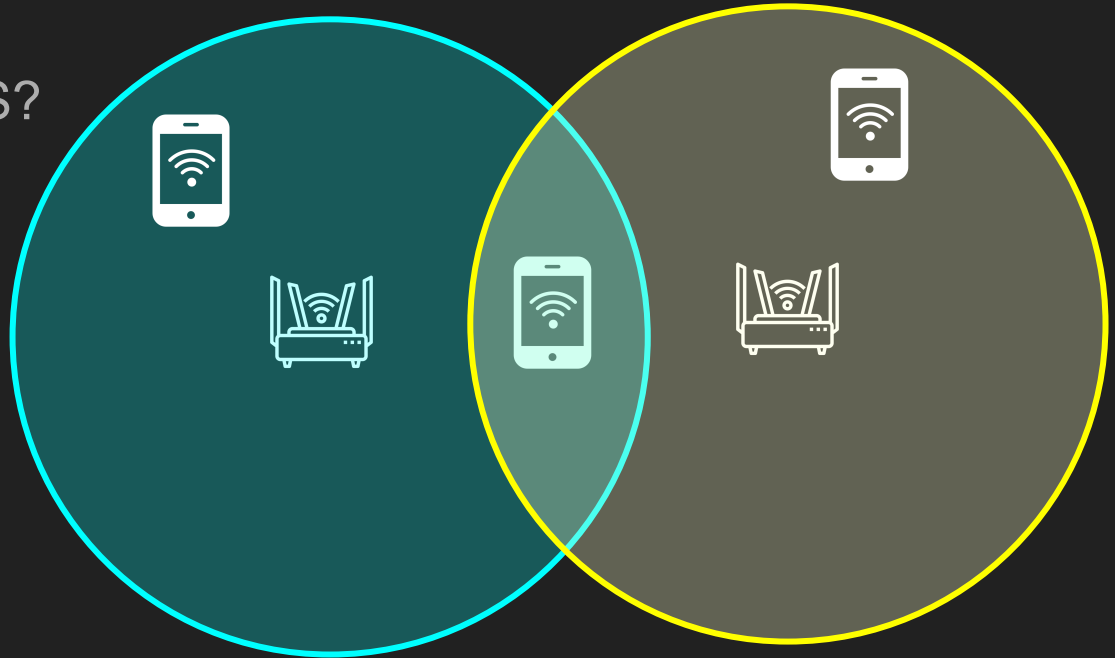
1. I send a short RTS.
2. I wait for a CTS reply
3. I send my data
4. I wait for an ACK



# 802.11 WiFi Multiple Access with Collision Avoidance

What if...

- I hear a RTS and a CTS?
- I hear only a RTS?
- I hear only a CTS?



## 802.11 WiFi Encryption

Wired Equivalent Privacy (**WEP**) - Broken and should be turned off

WiFi Protected Access (**WPA**) - Uses weaker Temporal Key Integrity Protocol (TKIP)

**WPA2** - Uses strong 256-bit Advanced Encryption Standard (AES) symmetric encryption

**WPA3** - Adds “perfect forward secrecy”. Secrets are safe even if later messages are decrypted. New and upcoming.

# WiFi Access Points - Hard Handoff

It is the mobile device's job to disconnect and join the stronger signal when it detects a better access point.



# Cell Networks - Soft Handover

During a transition between cell towers, both towers broadcast the phone call

Is it the towers' job to manage the handover and give good service

# Image Credits

Router by Studio 365 from Noun Project



Smartphone by IconHome from Noun Project