# [301] Bits and Memory

Tyler Caraza-Harter

# Ones and Zeros

**Computers “speak” ones and zeros (called bits)**

# Ones and Zeros

**Computers “speak” ones and zeros (called bits)**

**what about…**

# Ones and Zeros

**Computers “speak” ones and zeros (called bits)**

**what about…**

hello, world

**text**

# Ones and Zeros

**Computers “speak” ones and zeros (called bits)**

**what about…**

hello, world

**text**

3.14159…

**numbers**

# Ones and Zeros

**Computers “speak” ones and zeros (called bits)**

**what about…**

hello, world

**text**

**images**

3.14159…

**numbers**

# Ones and Zeros

**Computers “speak” ones and zeros (called bits)**

**what about…**

hello, world

**text**

**video**

3.14159…

**numbers**

**images**

# Ones and Zeros

**Computers “speak” ones and zeros (called bits)**

**what about…**

hello, world

**text**

**music**

**images**

**video**

3.14159…

**numbers**

# Ones and Zeros

Computers "speak" ones and zeros (called bits)

what about...

Intuitively, people used to believe all of these different types of information were **fundamentally different**

# Information Theory

Claud Shannon invented information theory in 1948, with bits as basic unit of information.

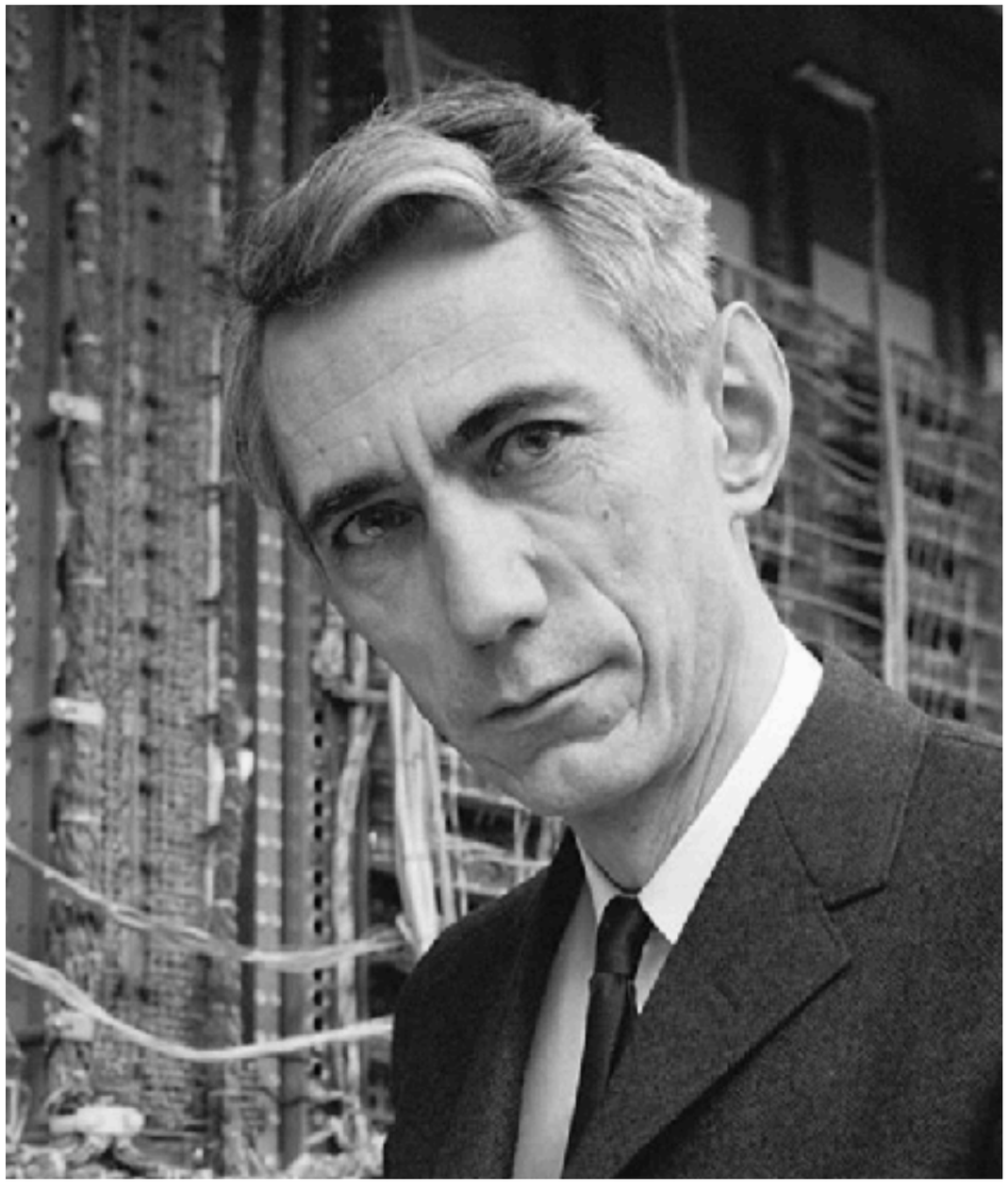
*Before Shannon, there was precious little sense of information as an idea, a measurable quantity, an object fitted out for hard science.*

*Before Shannon, information was a telegram, a photograph, a paragraph, a song.*

*After Shannon, information was entirely abstracted into bits.*

*The sender no longer mattered, the intent no longer mattered, the medium no longer mattered, not even the meaning mattered: a phone conversation, a snatch of Morse telegraph, a page from a detective story, were all brought under a common code.*

~ **A Mind at Play**, by Jimmy Soni and Rob Goodman



# Learning Objectives

What is a bit?  A byte?  How to quantify them?

How can we use simple bits to represent everything?

How are bits stored in memory?

How can we store instructions in memory?

# Today's Topics

Binary

- Bits, bytes
- Quantifying bytes

Memory Organization

Data representation

# What is a bit?

A bit is just a **0** or a **1**

# Why the name “bit”?

In the decimal number system, with have ten “digits”

- 0, 1, 2, 3, 5, 6, 7, 8, 9

# Why the name “bit”?

In the decimal number system, with have ten “digits”

- 0, 1, 2, 3, 5, 6, 7, 8, 9

In a “binary” system

- Two possibilities: 0 or 1

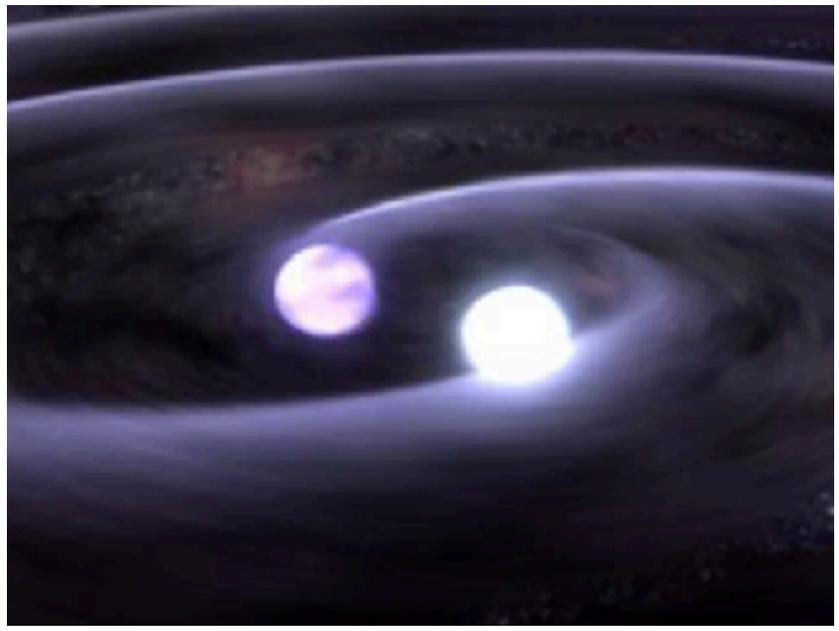
# Why the name “bit”?

In the decimal number system, with have ten “digits”

- 0, 1, 2, 3, 5, 6, 7, 8, 9

In a “binary” system

- Two possibilities: 0 or 1
- Binary is general word beyond scope of computing



**Binary Start System**

**https://www.space.com/22509-binary-stars.html**

# Why the name “bit”?

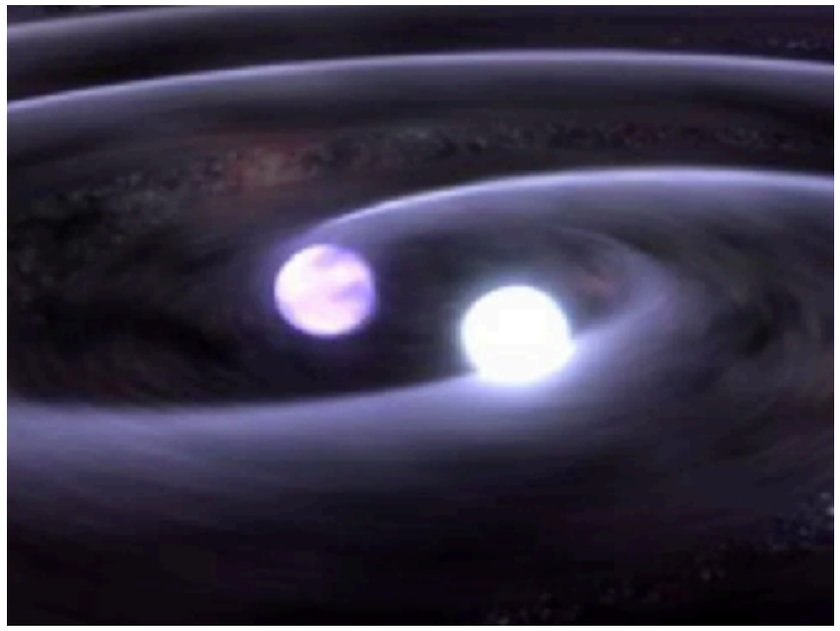
In the decimal number system, with have ten “digits”

- 0, 1, 2, 3, 5, 6, 7, 8, 9

In a “binary” system

- Two possibilities: 0 or 1
- Binary is general word beyond scope of computing

A “bit” is short for “binary digit”



**Binary Start System**

**https://www.space.com/22509-binary-stars.html**

# Why use bits in computing?

Why not decimal?  (make it easier for humans)

Or something else, like 12?  (divisible by many things)

# Why use bits in computing?

Why not decimal?  (make it easier for humans)

Or something else, like 12?  (divisible by many things)

**Reason 1: elegance**

- 2 different symbols in the minimum possible

# Why use bits in computing?

Why not decimal?  (make it easier for humans)

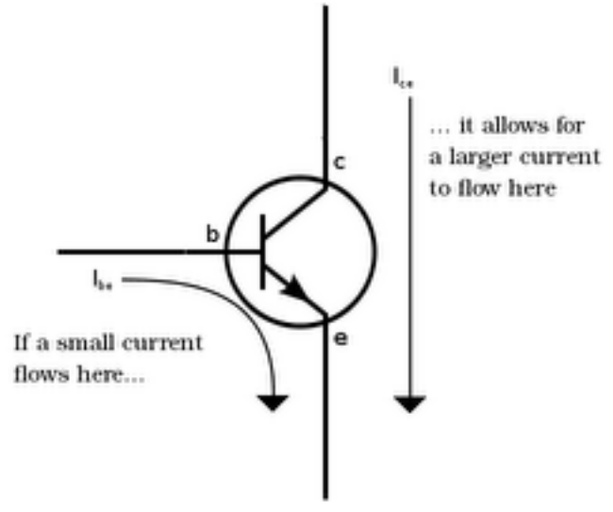
Or something else, like 12?  (divisible by many things)

**Reason 1: elegance**

- 2 different symbols in the minimum possible

**Reason 2: electronics**

- computers are built from transistors and capacitors
- 0/1 maps nicely to on/off, open/closed, charged/uncharged




https://www.build-electronic-circuits.com/how-transistors-work/

# What if you want more than 2 values?

Use more bits!

**1** bit gives **2** values: 0, 1

**2** bits gives **4** values: 00, 01, 10, 11

**3** bits gives **8** values: 000, 001, 010, 011, 100, 101, 110, 111

**N** bits gives $\mathbf{2^N}$ values

# What is a byte?

A byte is just 8 bits

# What is a byte?

A byte is just 8 bits

For example: 10101010

# What is a byte?

A byte is just 8 bits

For example: 10101010

*How many different byte values are there?*

# Today's Topics

Binary

- Bits, bytes
- Quantifying bytes

Memory Organization

Data representation

# Computers can process large numbers of bytes!

Metric system commonly used

- Just like a kilogram is 1000 grams, a kilobyte is 1000 bytes
- Abbreviate:

  kB (kilobyte, thousand), MB (megabyte, million),
  GB (gigabyte, billion), TB (terabyte, trillion)

# Computers can process large numbers of bytes!

Metric system commonly used

- Just like a kilogram is 1000 grams, a kilobyte is 1000 bytes
- Abbreviate:

  kB (kilobyte, thousand), MB (megabyte, million),
  GB (gigabyte, billion), TB (terabyte, trillion)

These metrics are used for speed and space

- Capacity: my hard drive can store 500 GB of data
- Throughput: I can save 100 MB of data per second to by hard drive

# Computers can process large numbers of bytes!

Metric system commonly used

- Just like a kilogram is 1000 grams, a kilobyte is 1000 bytes
- Abbreviate:

  kB (kilobyte, thousand), MB (megabyte, million),
  GB (gigabyte, billion), TB (terabyte, trillion)

These metrics are used for speed and space

- Capacity: my hard drive can store 500 GB of data
- Throughput: I can save 100 MB of data per second to by hard drive (sometimes written MB/s or MBps)

# Computers can process large numbers of bytes!

Metric system commonly used

- Just like a kilogram is 1000 grams, a kilobyte is 1000 bytes
- Abbreviate:

  kB (kilobyte, thousand), MB (megabyte, million),
  GB (gigabyte, billion), TB (terabyte, trillion)

These metrics are used for speed and space

- Capacity: my hard drive can store 500 GB of data
- Throughput: I can save 100 MB of data per second to by hard drive (sometimes written MB/s or MBps)

**What is the fastest you could fill up this hard drive, if it starts empty?**

# Gotcha 1: powers of 2

In computing, we often use 1024 instead of 1000

- 1024 is a “round number” in binary, because it is $2^{10}$
- Just like 1000 is round in decimal because it is 10

# Gotcha 1: powers of 2

In computing, we often use 1024 instead of 1000

- 1024 is a “round number” in binary, because it is $2^{10}$
- Just like 1000 is round in decimal because it is 10

For example

- So a kilobyte might mean 1024 bytes instead of 1000 bytes
- A megabyte might mean 1,048,576 bytes (instead of 1,000,000)

# Gotcha 1: powers of 2

In computing, we often use 1024 instead of 1000

- 1024 is a “round number” in binary, because it is $2^{10}$
- Just like 1000 is round in decimal because it is 10

For example

- So a kilobyte might mean 1024 bytes instead of 1000 bytes
- A megabyte might mean 1,048,576 bytes (instead of 1,000,000)
- Note: the difference in interpretations is relatively small

# Gotcha 2: “b” vs “B”

What is faster, 80 Mbps, or 20 MBps?

- A little “b” means bits
- A capital “B” means bytes

# Gotcha 2: “b” vs “B”

What is faster, 80 Mbps, or 20 MBps?

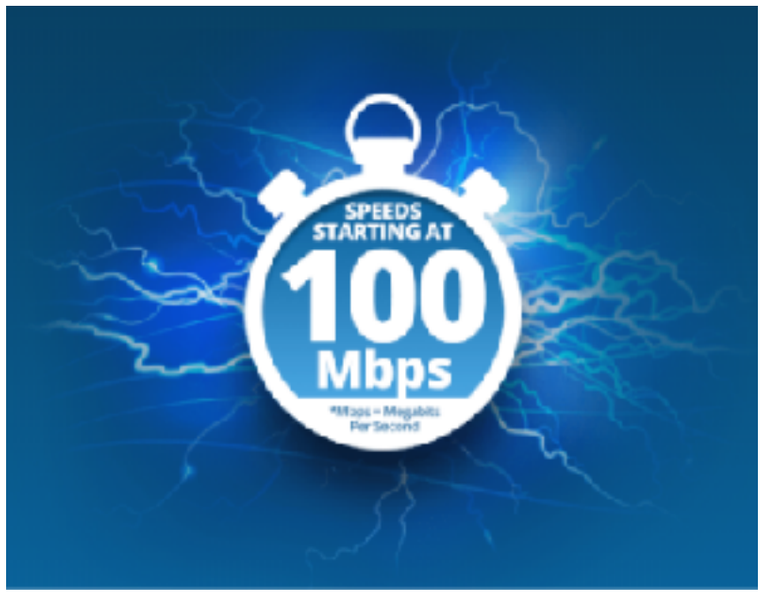
- A little “b” means bits
- A capital “B” means bytes
- 1 byte = 8 bits, so 80 Mbps is actually 10 MBps

# Gotcha 2: “b” vs “B”

What is faster, 80 Mbps, or 20 MBps?

- A little “b” means bits
- A capital “B” means bytes
- 1 byte = 8 bits, so 80 Mbps is actually 10 MBps

Bits are often used for networking, and bytes for storage




This suggests you could download 12.5 MB of data per second.

Example ad from https://www.spectrum.com/internet.html

# Today's Topics

Binary

Memory Organization

Data representation

- Numbers
- Colors and images
- Text

# RAM: Random Access Memory

Memory cell

- circuit that stores 1 bit of information
- often based on capacitors
- low voltage is 0, high voltage is 1

# RAM: Random Access Memory

Memory cell

- circuit that stores 1 bit of information
- often based on capacitors
- low voltage is 0, high voltage is 1

Memory a lot of memory cells!

- Typically billions (for gigabytes of memory)

lots of tiny capacitors

**these are built in, and don't actually look like this in the chip**

# RAM: Random Access Memory

Memory cell

- circuit that stores 1 bit of information
- often based on capacitors
- low voltage is 0, high voltage is 1

Memory a lot of memory cells!

- Typically billions (for gigabytes of memory)
- How to find the right bits you need to access?

# RAM: Random Access Memory

Memory cell

- circuit that stores 1 bit of information
- often based on capacitors
- low voltage is 0, high voltage is 1

Memory a lot of memory cells!

- Typically billions (for gigabytes of memory)
- How to find the right bits you need to access?

**give groups of bits/bytes addresses**

# Addresses

| bits in memory | voltage |
|---|---|
| … | … |
| 0 | low |
| 0 | low |
| 0 | low |
| 0 | low |
| 1 | high |
| 1 | high |
| 1 | high |
| 0 | low |
| 1 | high |
| 0 | low |
| 1 | high |
| 0 | low |
| 1 | high |
| 0 | low |
| 1 | high |
| … | … |

# Addresses

| addresses | bits in memory | voltage |
|---|---|---|
| | … | … |
| 150 → | 0 | low |
| | 0 | low |
| | 0 | low |
| | 0 | low |
| | 1 | high |
| | 1 | high |
| | 1 | high |
| | 1 | high |
| 151 → | 0 | low |
| | 1 | high |
| | 0 | low |
| | 1 | high |
| | 0 | low |
| | 1 | high |
| | 0 | low |
| | 1 | high |
| | … | … |

# Addresses

| addresses | bits in memory | voltage |
|---|---|---|
| | … | … |
| 150 → | 0 | low |
| | 0 | low |
| | 0 | low |
| | 0 | low |
| | 1 | high |
| | 1 | high |
| | 1 | high |
| 151 → | 0 | low |
| | 1 | high |
| | 0 | low |
| | 1 | high |
| | 0 | low |
| | 1 | high |
| | 0 | low |
| | 1 | high |
| | … | … |

**request bits at address 150**

# Addresses

| addresses | bits in memory | voltage |
|---|---|---|
| | … | … |
| 150 → | 0 | low |
| | 0 | low |
| | 0 | low |
| | 0 | low |
| | 1 | high |
| | 1 | high |
| | 1 | high |
| | 1 | high |
| 151 → | 0 | low |
| | 1 | high |
| | 0 | low |
| | 1 | high |
| | 0 | low |
| | 1 | high |
| | 0 | low |
| | 1 | high |
| | … | … |

**request bits at address 150**

**get back 00001111**

# Today's Topics

Binary

Memory Organization

Data representation

- Numbers
- Colors and images
- Text
- Instructions

# Data representation

## Challenge

*How can we use ones and zeros (many of them) to represent a wide variety of data?*

# Today's Topics

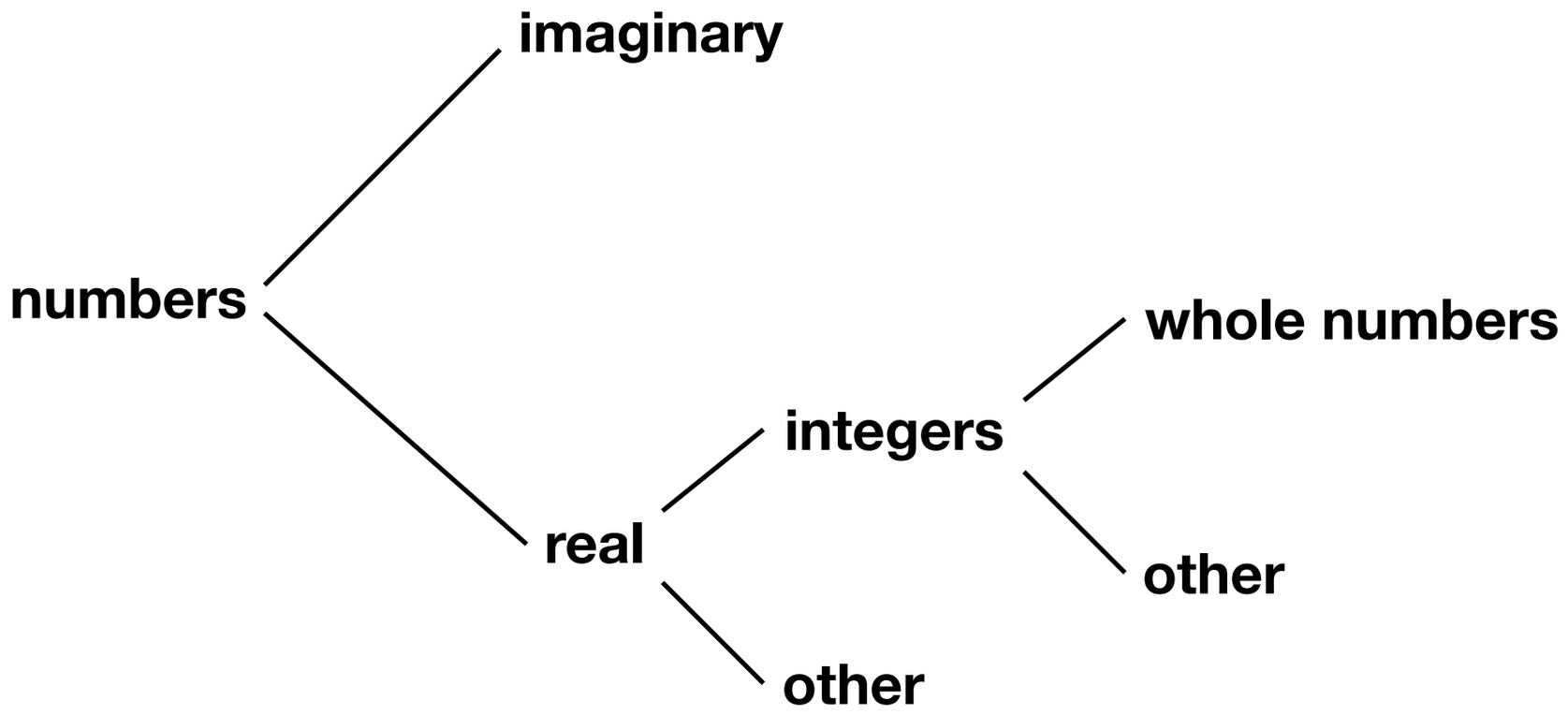
Binary

Memory Organization

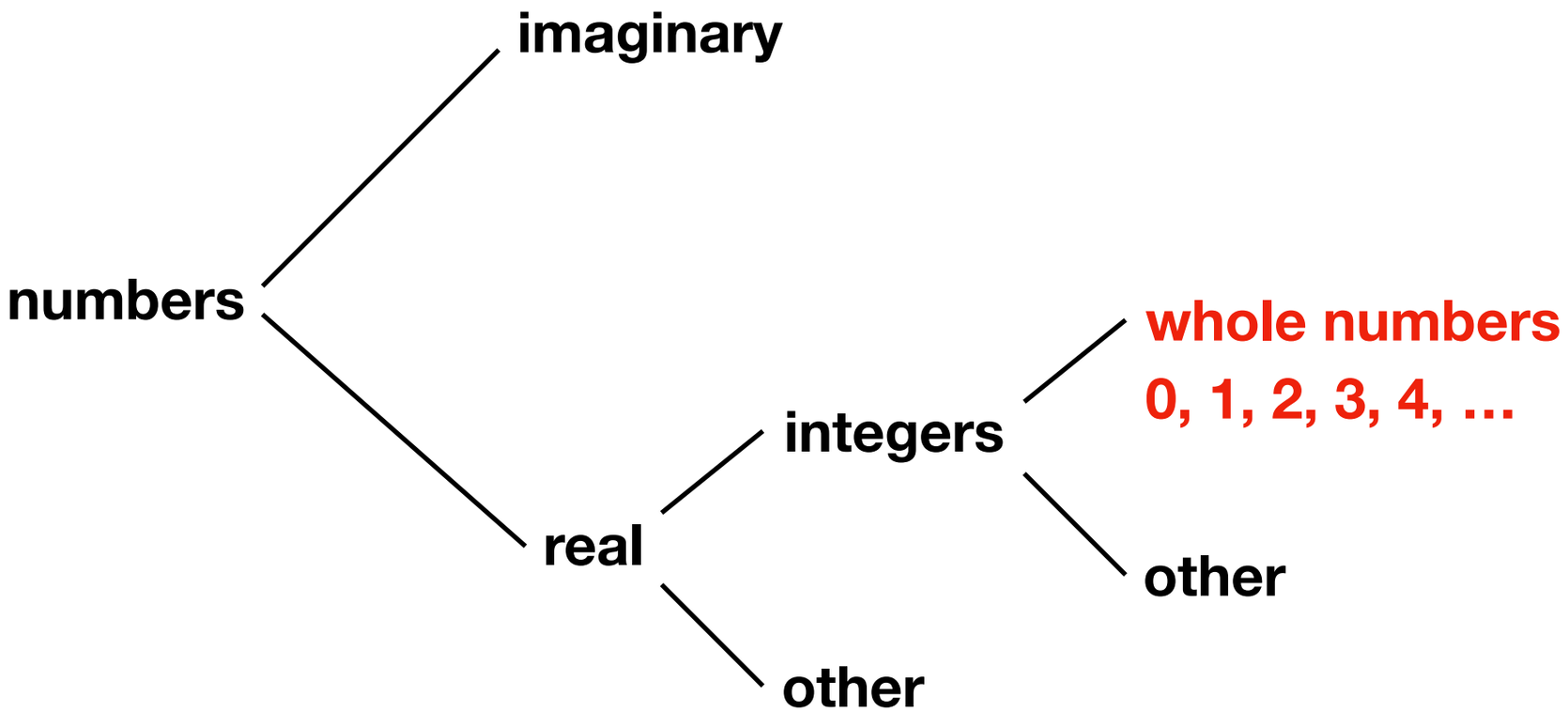
Data representation

- Numbers
- Colors and images
- Text
- Instructions

# Types of numbers



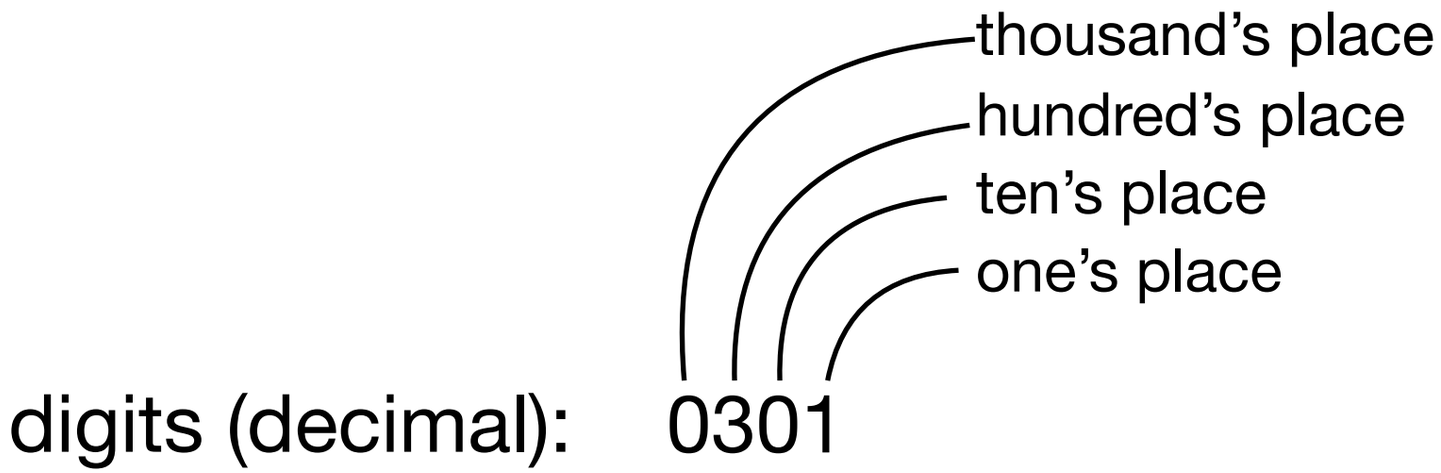
# Types of numbers



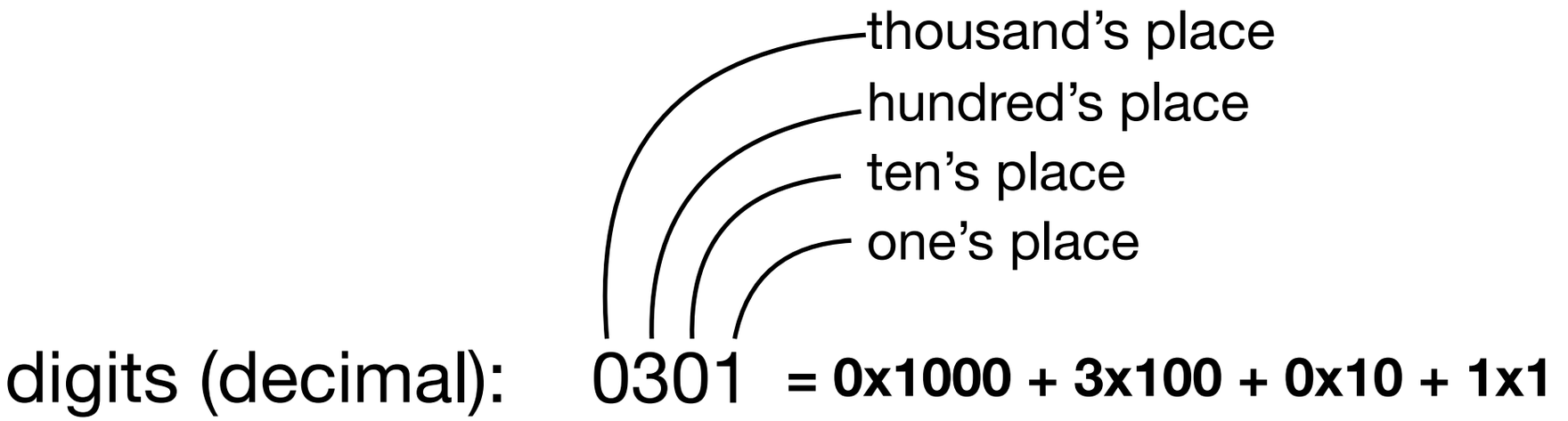
# Base ten vs two example

digits (decimal):  0301

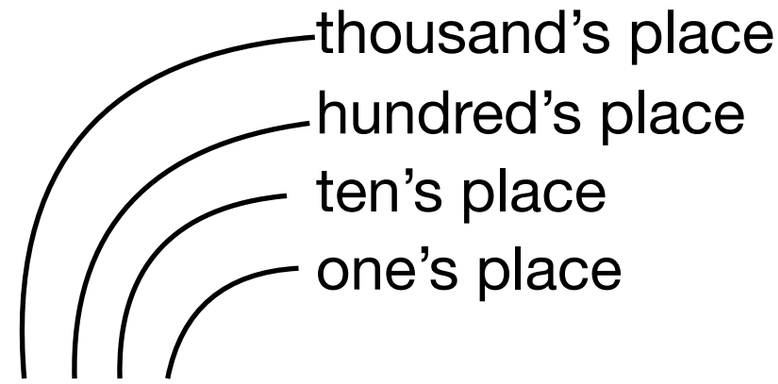
# Base ten vs two example



# Base ten vs two example



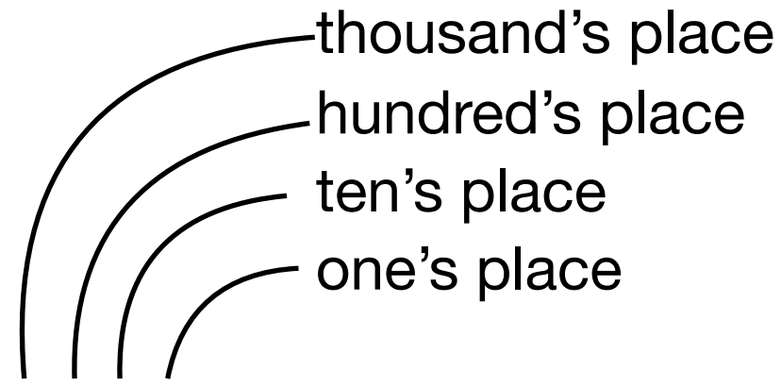
# Base ten vs two example




Step up by factors of **ten**
because we have **ten** digits

digits (decimal): 0301 **= 0x1000 + 3x100 + 0x10 + 1x1**

# Base ten vs two example



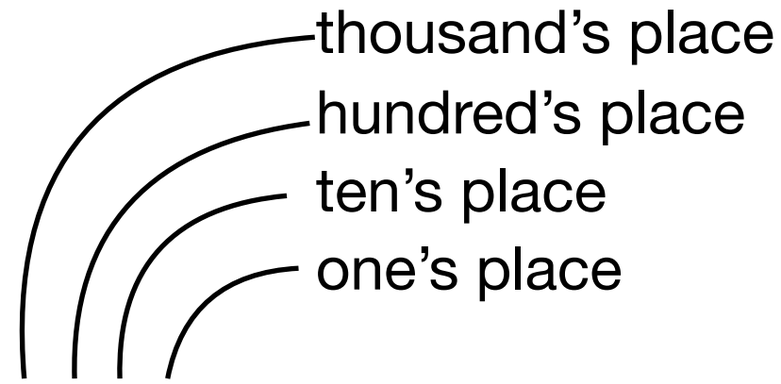

Step up by factors of **ten**
because we have **ten** digits

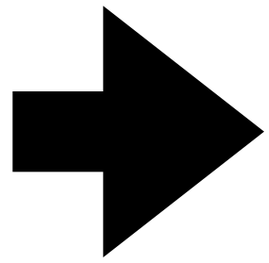
digits (decimal): 0301 **= 0x1000 + 3x100 + 0x10 + 1x1**

bits (binary): 1011

Number only uses 0's and 1's
(NOT thousand and eleven)

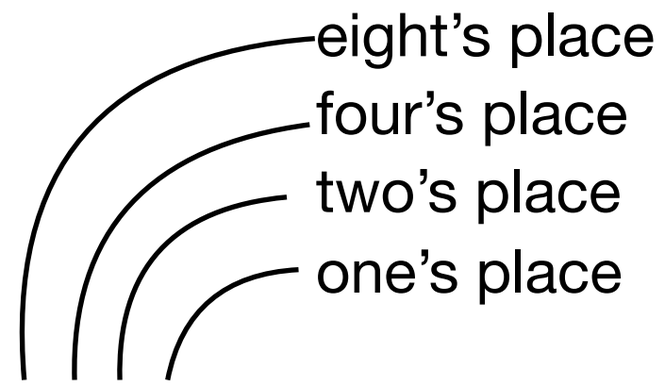
# Base ten vs two example



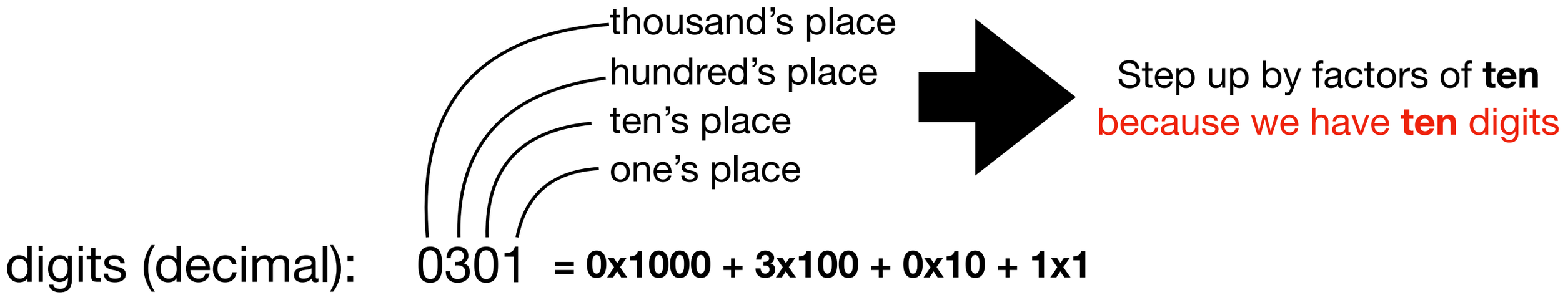
Step up by factors of **ten**
because we have **ten** digits

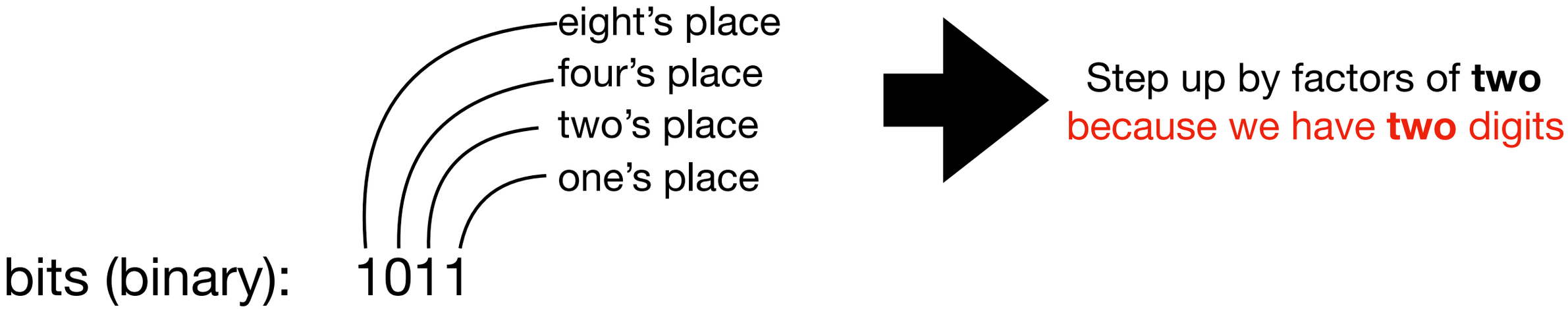
digits (decimal): 0301 **= 0x1000 + 3x100 + 0x10 + 1x1**



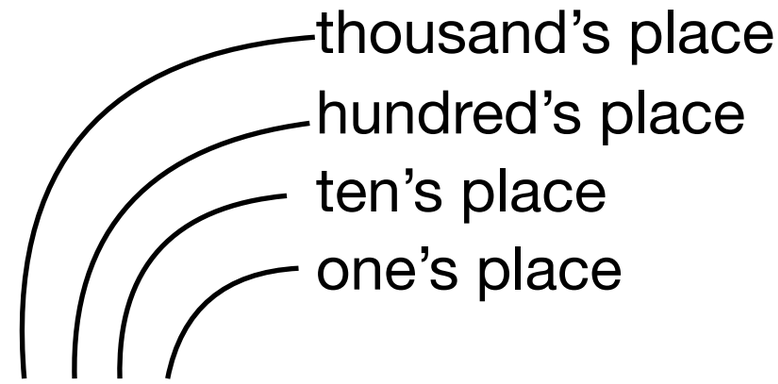

bits (binary): 1011

# Base ten vs two example



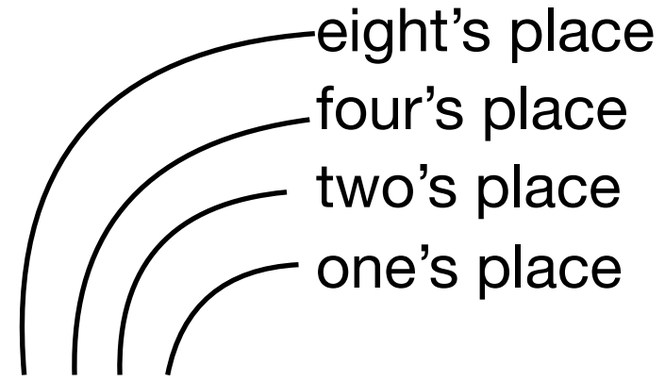



# Base ten vs two example




Step up by factors of **ten**
because we have **ten** digits

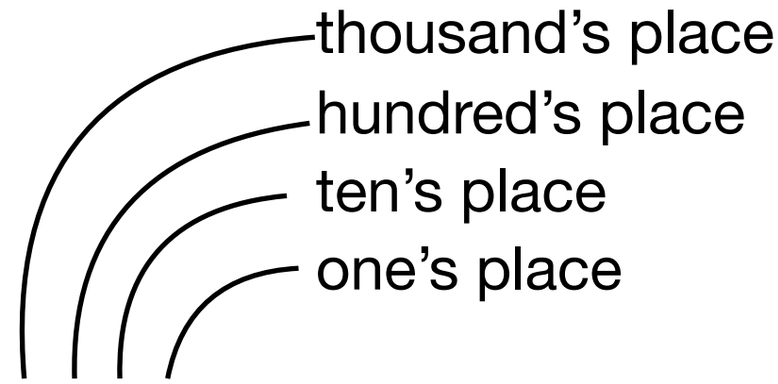
digits (decimal): 0301 **= 0x1000 + 3x100 + 0x10 + 1x1**




Step up by factors of **two**
because we have **two** digits

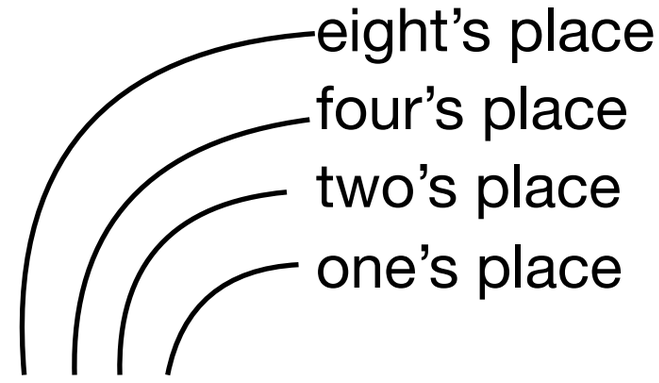
bits (binary): 1011 **= 1x8 + 0x2 + 1x2 + 1x1**

# Base ten vs two example




Step up by factors of **ten**
because we have **ten** digits

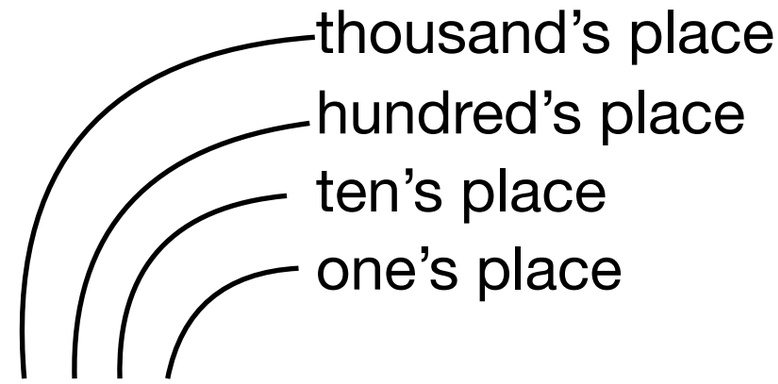
digits (decimal): 0301 **= 0x1000 + 3x100 + 0x10 + 1x1**




Step up by factors of **two**
because we have **two** digits

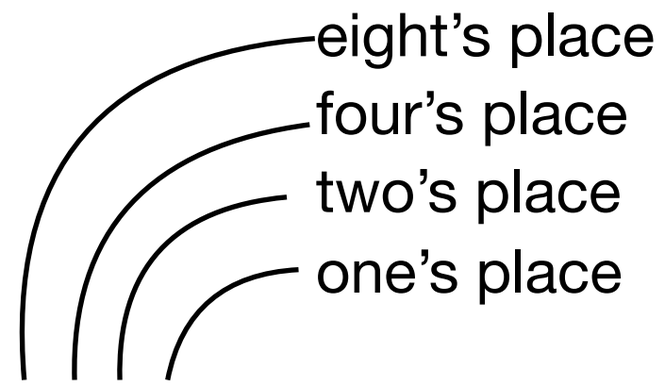
bits (binary): 1011 **= 1x8 + 0x2 + 1x2 + 1x1 = eleven**

# Base ten vs two example




Step up by factors of **ten**
because we have **ten** digits

digits (decimal): 0301 **= 0x1000 + 3x100 + 0x10 + 1x1**

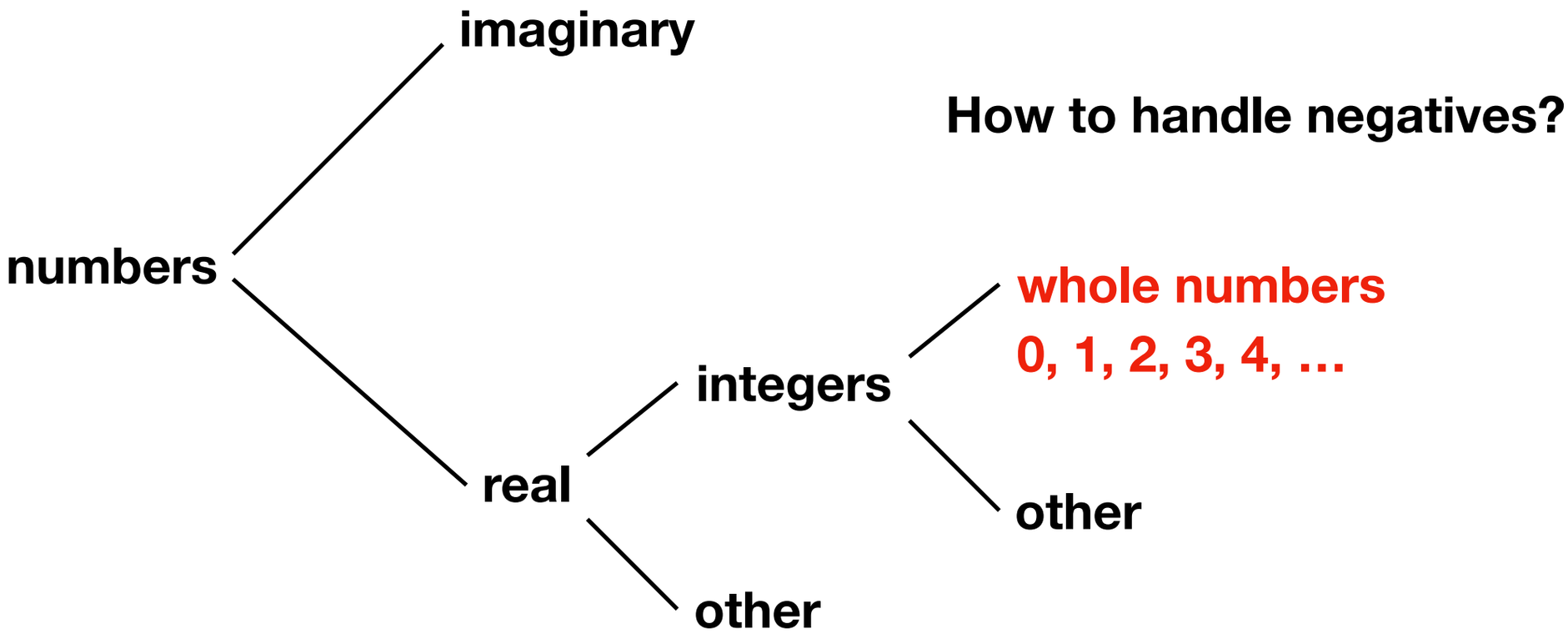



Step up by factors of **two**
because we have **two** digits

bits (binary): 1011 **= 1x8 + 0x2 + 1x2 + 1x1 = eleven**

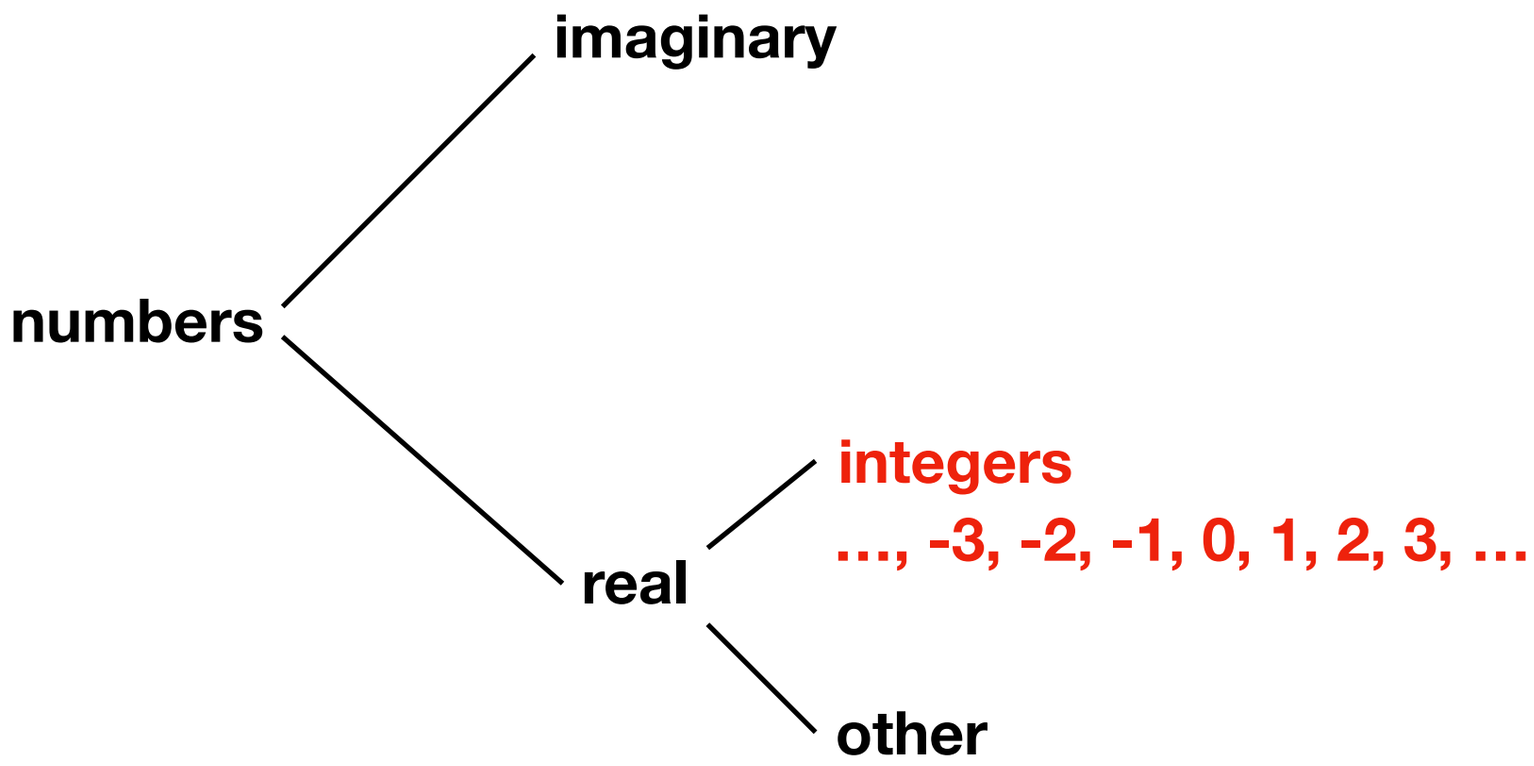
**What decimal number is represented by these bits?**
**11001**

# Types of numbers




**How to handle negatives?**

# Types of numbers



# Negative binary numbers

eleven: 1011

# Negative binary numbers

eleven: 1011

negative eleven: -1011

# Negative binary numbers

eleven: 1011

negative eleven: -1011

**Problem: computers know “0” and “1”**
**What is this “-” thing?**

# Negative binary numbers

eleven: 1011

negative eleven: -1011

**Problem: computers know “0” and “1”**
**What is this “-” thing?**

**Discuss approaches to representing negative integers with only 0’s and 1’s with your neighbor**

# One representation scheme

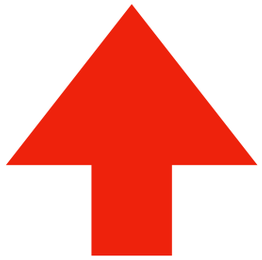
**01011 is eleven**

**11011 is negative eleven**

# One representation scheme

**01011 is eleven**

**11011 is negative eleven**



**main number**

# One representation scheme

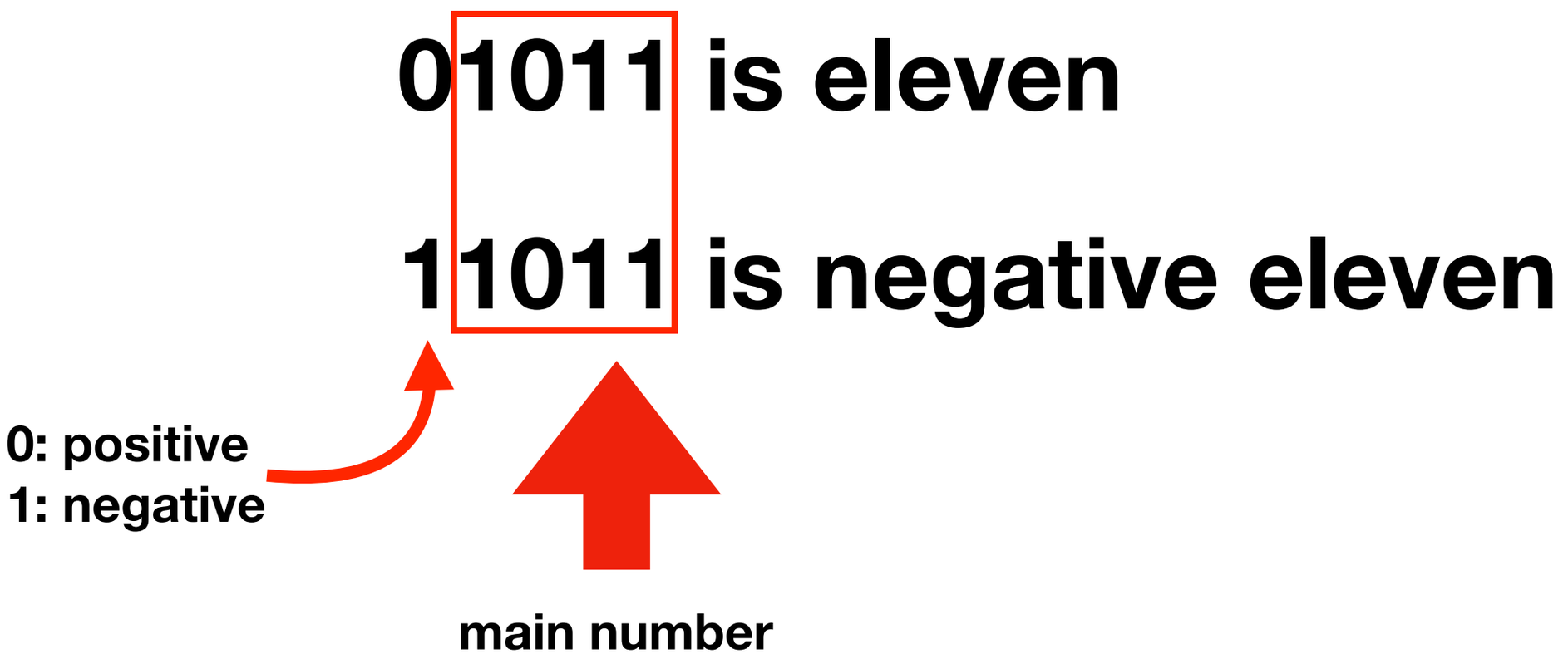
**01011 is eleven**

**11011 is negative eleven**

**0: positive**
**1: negative**

**main number**

# One representation scheme




One oddity: **0**0000 and **1**0000 are two different ways to say the same number, zero.  In practice, most computers use a slightly more complicated representation.

# Today's Topics

Binary

Memory Organization

Data representation

- Numbers
- Colors and images
- Text
- Instructions

# Representing color

All colors of light are a combination of red, green, and blue

- Of varying intensities
- Sometimes abbreviated RGB
- Maximum intensity of all three is white

# Representing color

All colors of light are a combination of red, green, and blue

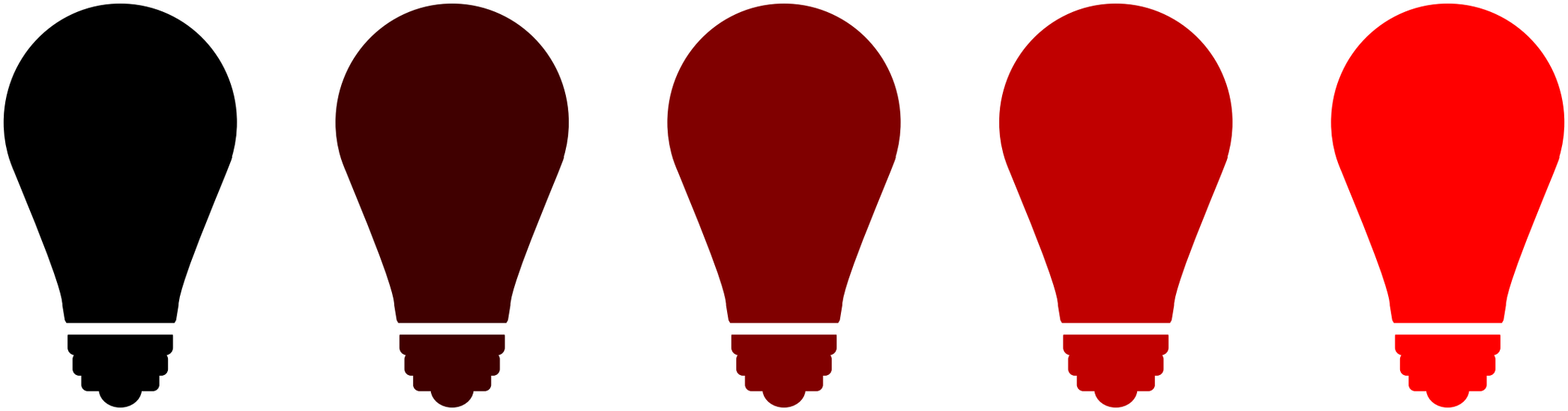
- Of varying intensities
- Sometimes abbreviated RGB
- Maximum intensity of all three is white

**How to represent intensity of one of these primary colors?**

# Red intensity

Strategy

- Strategy: use whole numbers (already know how to represent these as bits)
- Small number: less red
- Big number: more red



# Back to colors

Strategy

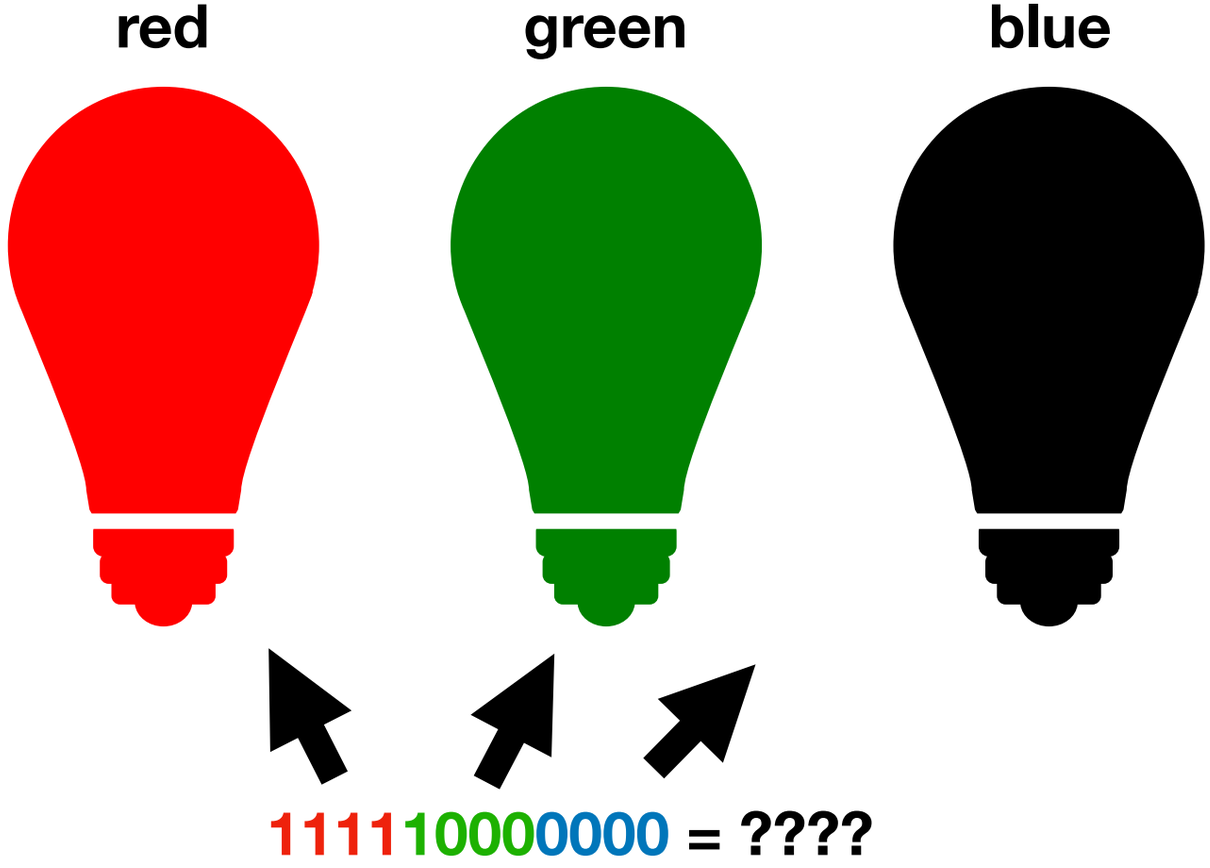
- Encode final color as mix of three intensities

**111110000000 = ????**

# Back to colors

Strategy

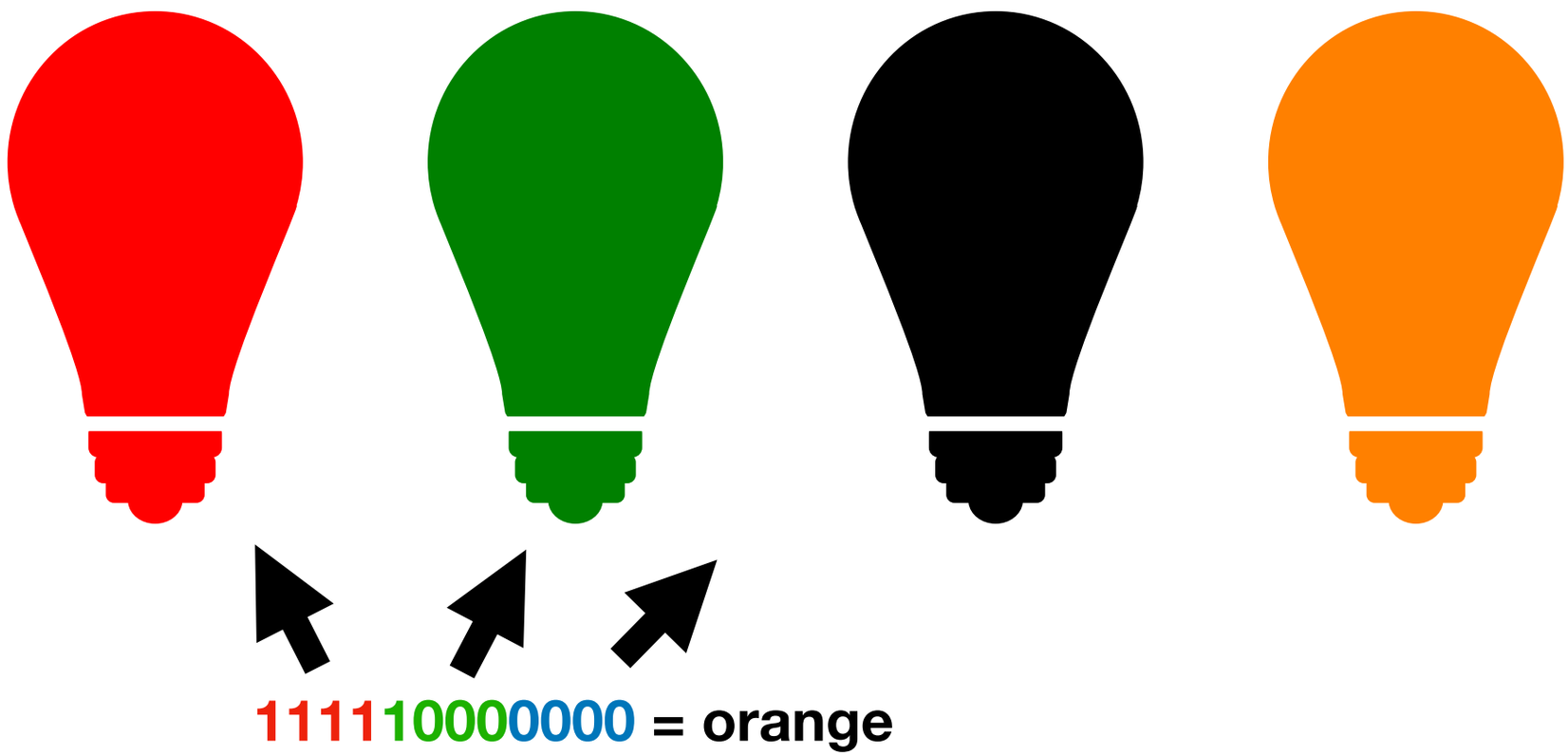
- Encode final color as mix of three intensities



# Back to colors

Strategy

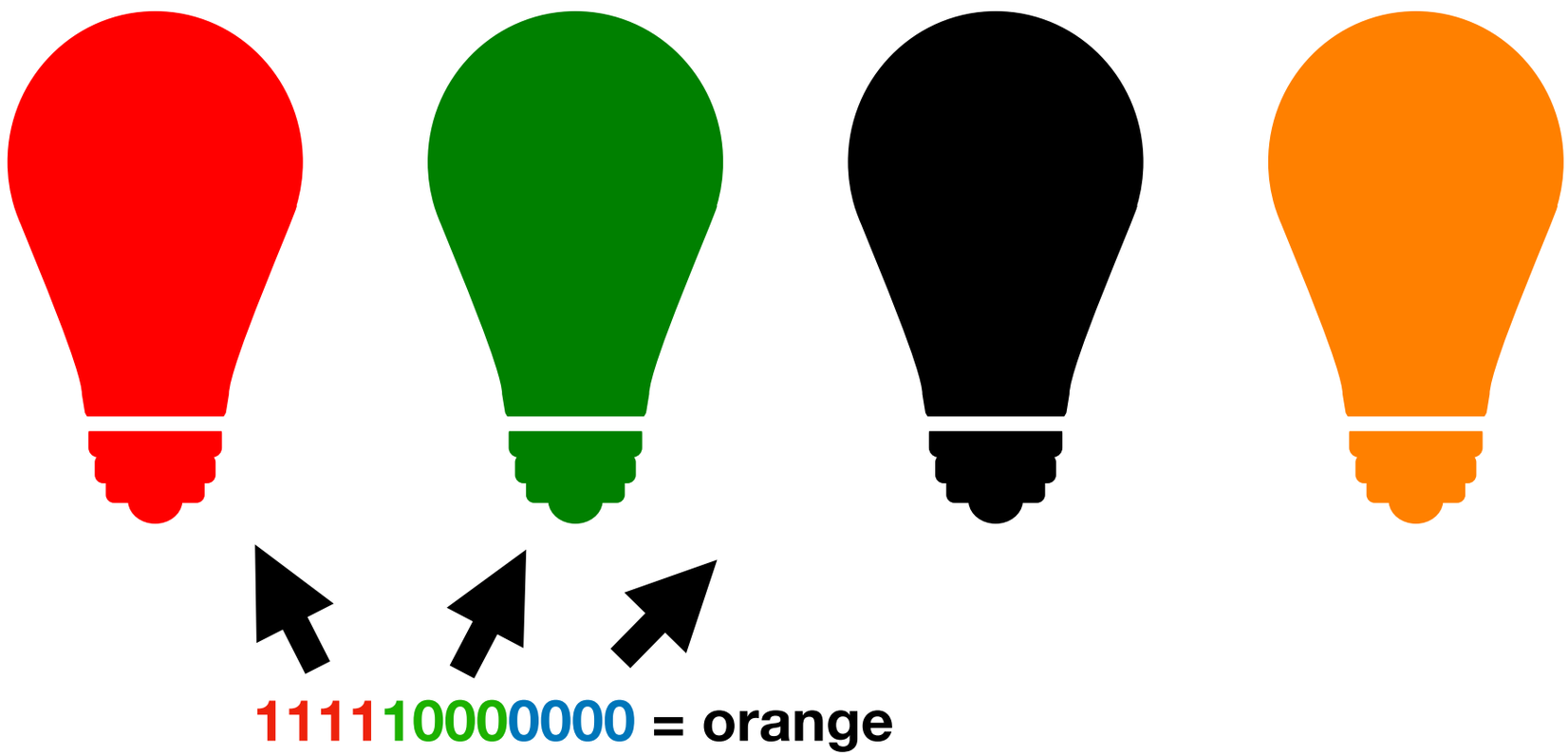
- Encode final color as mix of three intensities



# Back to colors

Strategy

- Encode final color as mix of three intensities
- We can represent many colors with 12 bits now



# What about images?

Strategy

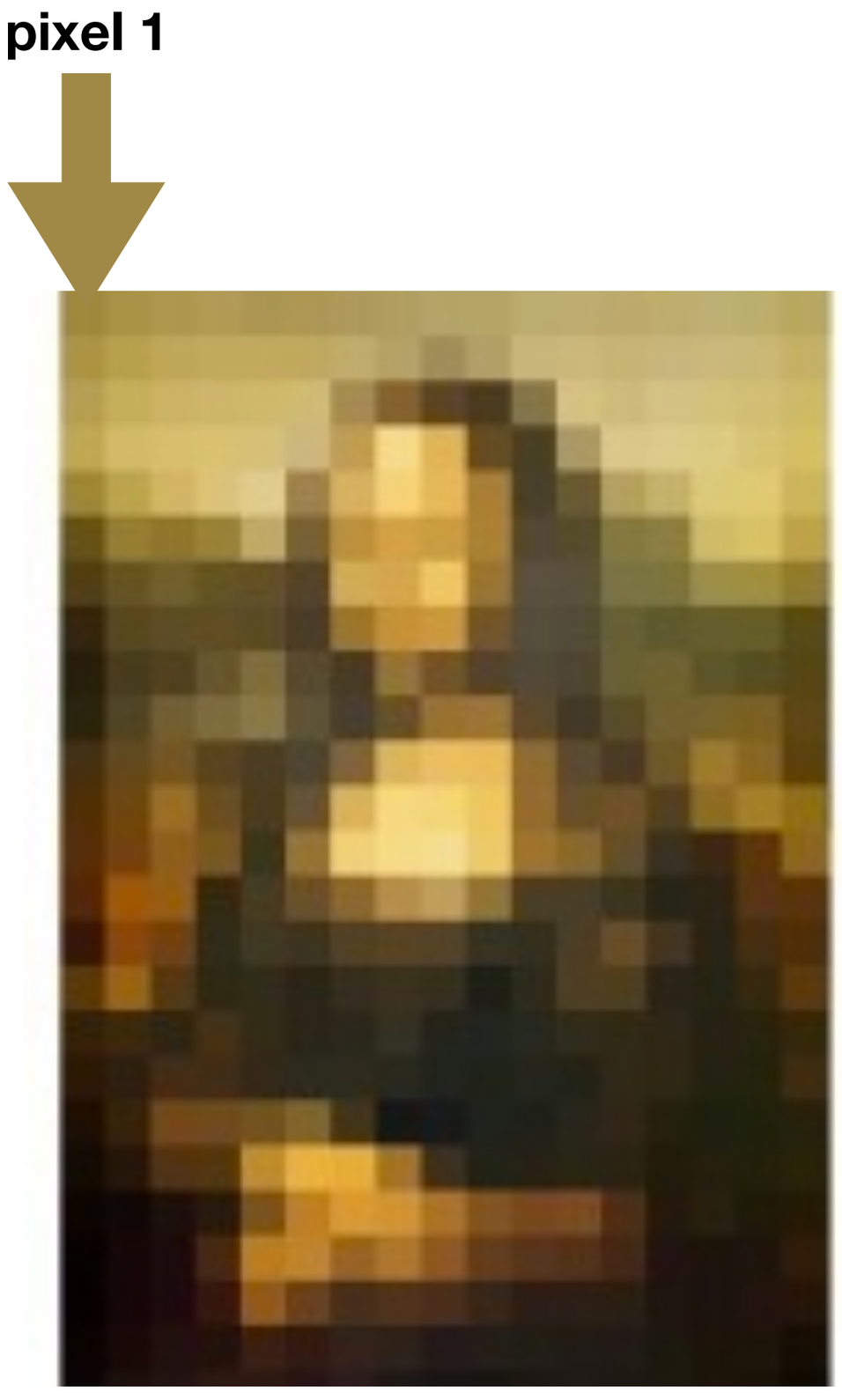
- Break into a bunch of small squares, called pixels
- Record color of each pixel

# What about images?

101010000100

Strategy

- Break into a bunch of small squares, called pixels
- Record color of each pixel



# What about images?

101010000100 101010010100

Strategy

- Break into a bunch of small squares, called pixels
- Record color of each pixel

# What about images?

101010000100 101010010100 101010010101

Strategy

- Break into a bunch of small squares, called pixels
- Record color of each pixel

# What about images?

101010000100 101010010100 101010010101 …

Strategy

- Break into a bunch of small squares, called pixels
- Record color of each pixel

# A note on breaking down problems

Problem: how to represent images as 1’s and 0’s?

- Can we make it simpler?

# A note on breaking down problems

Problem: how to represent images as 1's and 0's?

- Can we make it simpler?

We can represent **images** as a bunch of **color values**

We can represent a **color value** as a three **integers**

We can represent an **integer** as a bunch of **bits**

# A note on breaking down problems

Problem: how to represent images as 1's and 0's?

- Can we make it simpler?

We can represent **images** as a bunch of **color values**

We can represent a **color value** as a three **integers**

We can represent an **integer** as a bunch of **bits**

Chain these representations together to represent images as bits!

# Today's Topics

Binary

Memory Organization

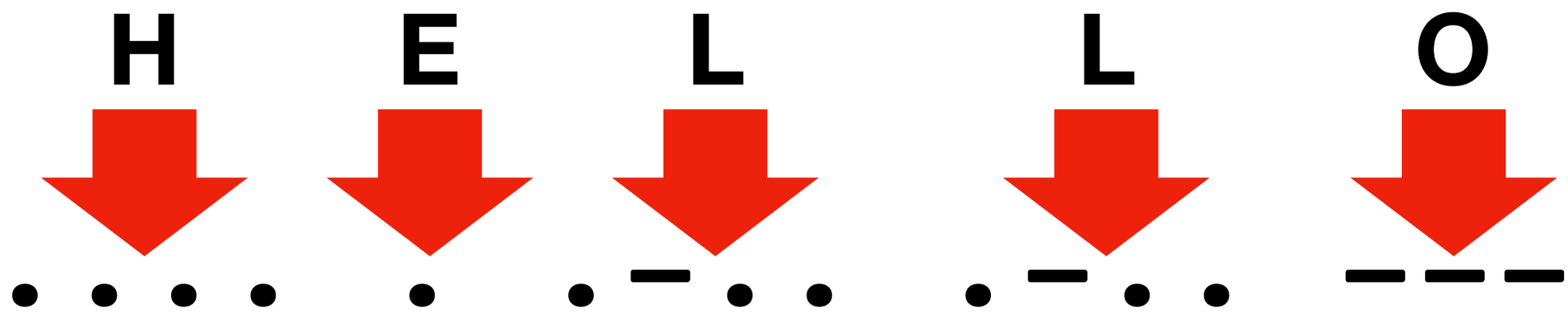
Data representation

- Numbers
- Colors and images
- Text
- Instructions

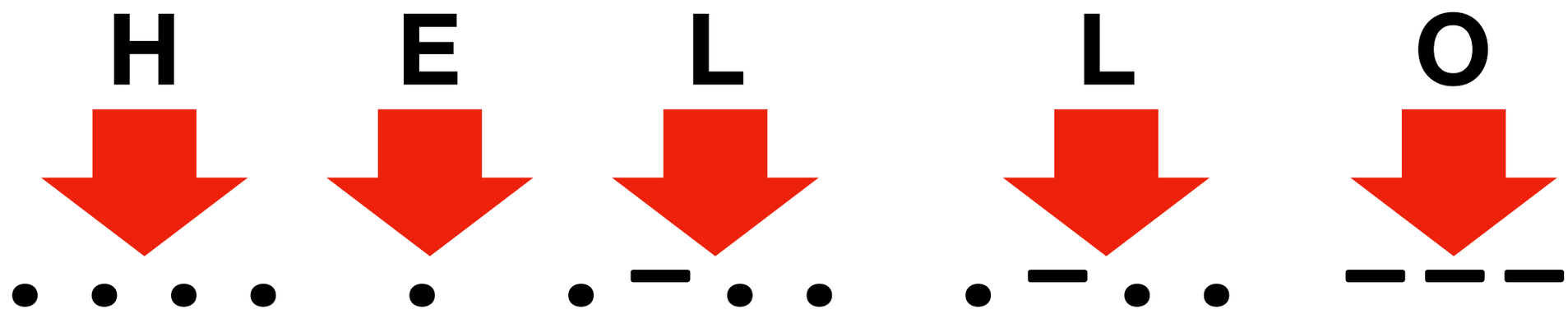
# How can we encode text, like “Hello World”, using just bytes?

# Morse Code

# Morse Code

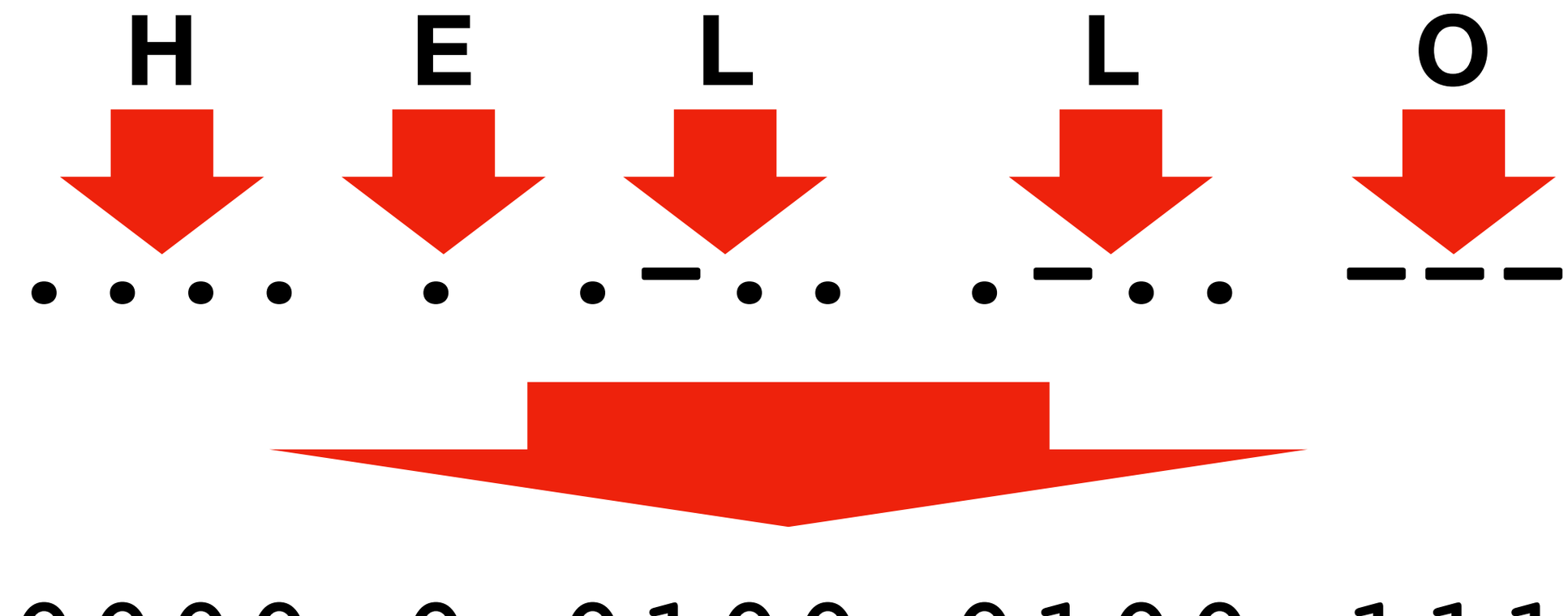


# Morse Code



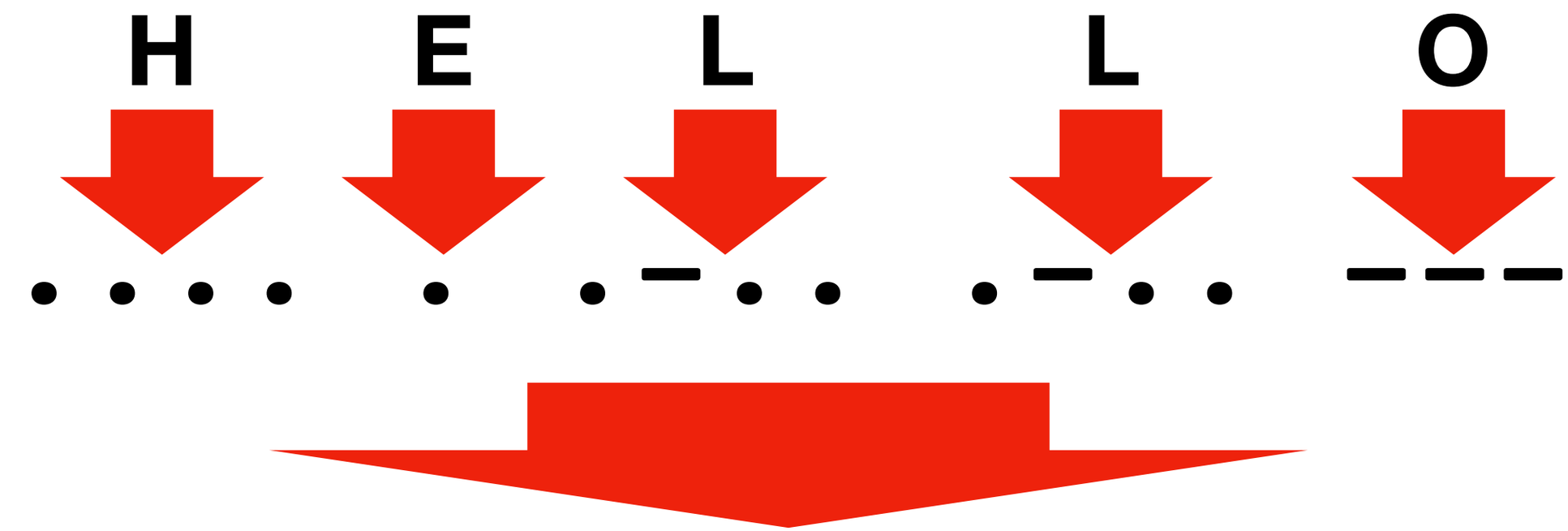

**We can represent text as dots and dashes.**
**Any ideas on how we can further encode dots and dashes as bits?**

# Morse Code




0000 0 0100 0100 111

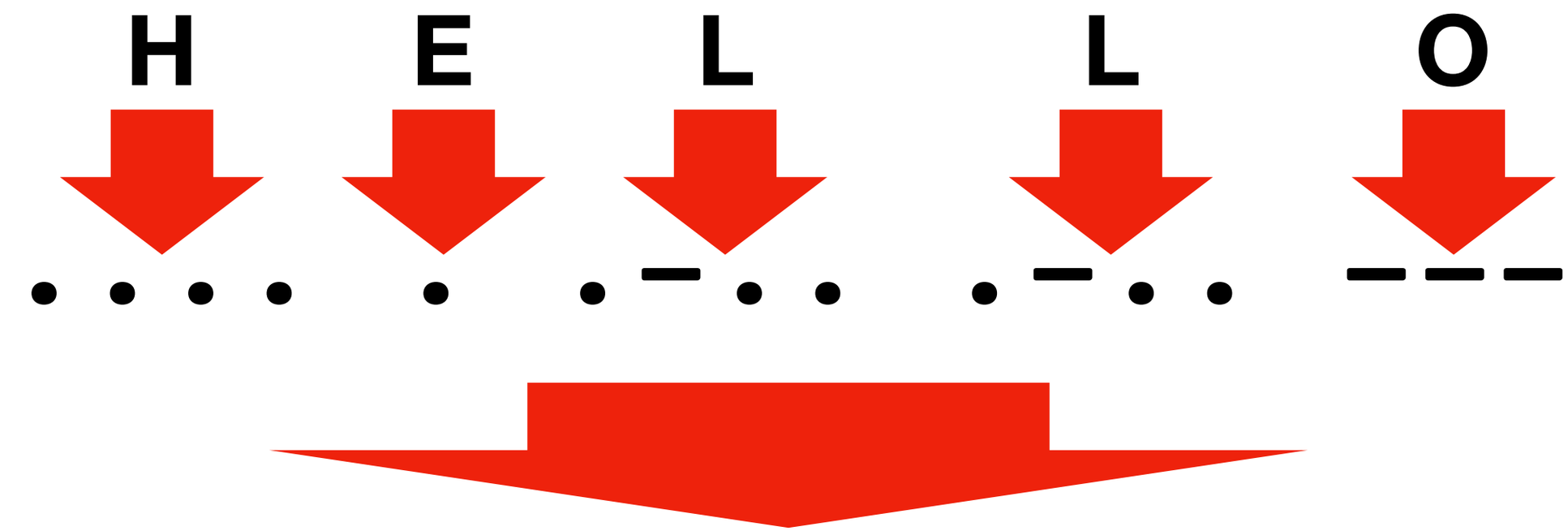
# Morse Code

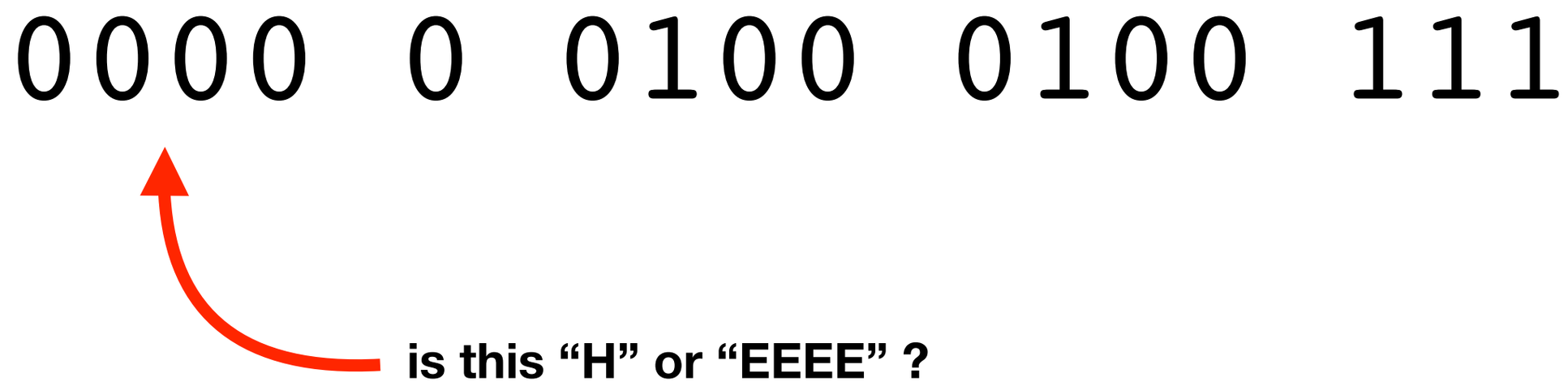



```
0000 0 0100 0100 111
```

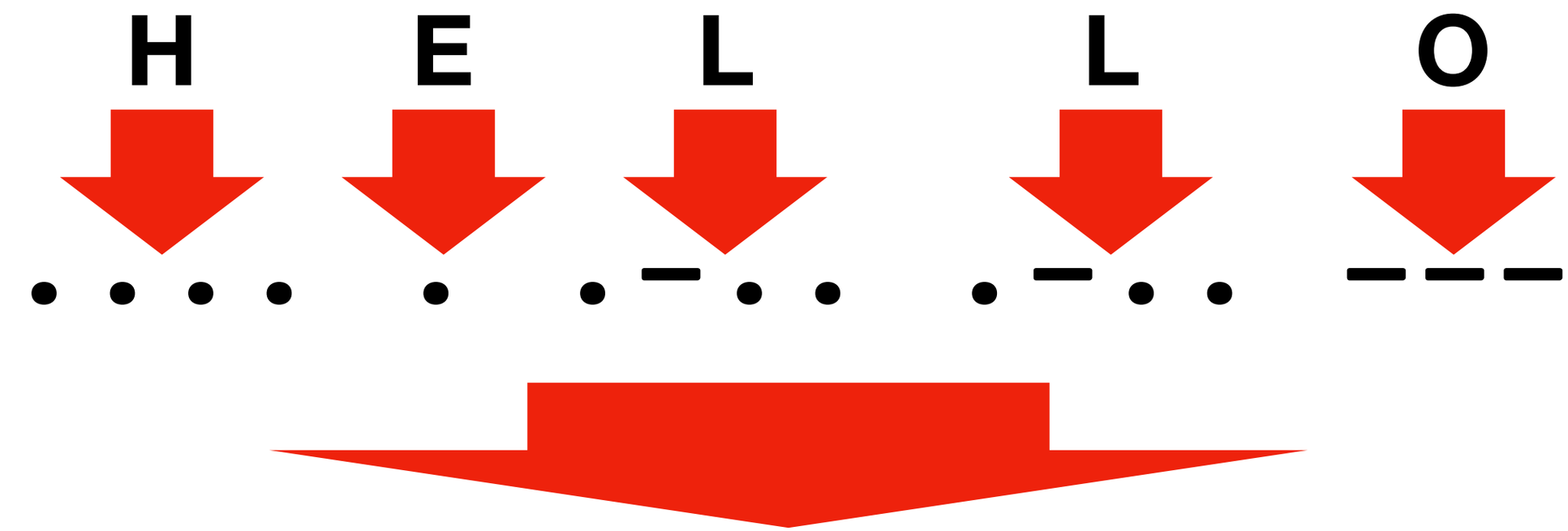
**What’s wrong with this?**

# Morse Code



# Morse Code




0000 0 0100 0100 111

**is this “H” or “EEEE” ?**

**Morse code depends on spaces, which aren’t bits!**
**(similar to problem we encountered with “-”)**

# A popular encoding: ASCII

American Standard Code for Information Interchange

- We know how to encode integers as bits

# A popular encoding: ASCII

American Standard Code for Information Interchange

- We know how to encode integers as bits
- Assign every English letter (upper and lower case), digit, and various other symbols a number between 0 and 127
- To save text as bits, convert each character to a number, then convert that number to bits

| Dec | Hx | Oct | Char | Dec | Hx | Oct | Html | Chr | Dec | Hx | Oct | Html | Chr | Dec | Hx | Oct | Html | Chr |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 000 | NUL (null) | 32 | 20 | 040 |   | Space | 64 | 40 | 100 | @ | @ | 96 | 60 | 140 | ` | ` |
| 1 | 1 | 001 | SOH (start of heading) | 33 | 21 | 041 | ! | ! | 65 | 41 | 101 | A | A | 97 | 61 | 141 | a | a |
| 2 | 2 | 002 | STX (start of text) | 34 | 22 | 042 | " | " | 66 | 42 | 102 | B | B | 98 | 62 | 142 | b | b |
| 3 | 3 | 003 | ETX (end of text) | 35 | 23 | 043 | # | # | 67 | 43 | 103 | C | C | 99 | 63 | 143 | c | c |
| 4 | 4 | 004 | EOT (end of transmission) | 36 | 24 | 044 | $ | $ | 68 | 44 | 104 | D | D | 100 | 64 | 144 | d | d |
| 5 | 5 | 005 | ENQ (enquiry) | 37 | 25 | 045 | % | % | 69 | 45 | 105 | E | E | 101 | 65 | 145 | e | e |
| 6 | 6 | 006 | ACK (acknowledge) | 38 | 26 | 046 | & | & | 70 | 46 | 106 | F | F | 102 | 66 | 146 | f | f |
| 7 | 7 | 007 | BEL (bell) | 39 | 27 | 047 | ' | ' | 71 | 47 | 107 | G | G | 103 | 67 | 147 | g | g |
| 8 | 8 | 010 | BS (backspace) | 40 | 28 | 050 | ( | ( | 72 | 48 | 110 | H | H | 104 | 68 | 150 | h | h |
| 9 | 9 | 011 | TAB (horizontal tab) | 41 | 29 | 051 | ) | ) | 73 | 49 | 111 | I | I | 105 | 69 | 151 | i | i |
| 10 | A | 012 | LF (NL line feed, new line) | 42 | 2A | 052 | * | * | 74 | 4A | 112 | J | J | 106 | 6A | 152 | j | j |
| 11 | B | 013 | VT (vertical tab) | 43 | 2B | 053 | + | + | 75 | 4B | 113 | K | K | 107 | 6B | 153 | k | k |
| 12 | C | 014 | FF (NP form feed, new page) | 44 | 2C | 054 | , | , | 76 | 4C | 114 | L | L | 108 | 6C | 154 | l | l |
| 13 | D | 015 | CR (carriage return) | 45 | 2D | 055 | - | - | 77 | 4D | 115 | M | M | 109 | 6D | 155 | m | m |
| 14 | E | 016 | SO (shift out) | 46 | 2E | 056 | . | . | 78 | 4E | 116 | N | N | 110 | 6E | 156 | n | n |
| 15 | F | 017 | SI (shift in) | 47 | 2F | 057 | / | / | 79 | 4F | 117 | O | O | 111 | 6F | 157 | o | o |
| 16 | 10 | 020 | DLE (data link escape) | 48 | 30 | 060 | 0 | 0 | 80 | 50 | 120 | P | P | 112 | 70 | 160 | p | p |
| 17 | 11 | 021 | DC1 (device control 1) | 49 | 31 | 061 | 1 | 1 | 81 | 51 | 121 | Q | Q | 113 | 71 | 161 | q | q |
| 18 | 12 | 022 | DC2 (device control 2) | 50 | 32 | 062 | 2 | 2 | 82 | 52 | 122 | R | R | 114 | 72 | 162 | r | r |
| 19 | 13 | 023 | DC3 (device control 3) | 51 | 33 | 063 | 3 | 3 | 83 | 53 | 123 | S | S | 115 | 73 | 163 | s | s |
| 20 | 14 | 024 | DC4 (device control 4) | 52 | 34 | 064 | 4 | 4 | 84 | 54 | 124 | T | T | 116 | 74 | 164 | t | t |
| 21 | 15 | 025 | NAK (negative acknowledge) | 53 | 35 | 065 | 5 | 5 | 85 | 55 | 125 | U | U | 117 | 75 | 165 | u | u |
| 22 | 16 | 026 | SYN (synchronous idle) | 54 | 36 | 066 | 6 | 6 | 86 | 56 | 126 | V | V | 118 | 76 | 166 | v | v |
| 23 | 17 | 027 | ETB (end of trans. block) | 55 | 37 | 067 | 7 | 7 | 87 | 57 | 127 | W | W | 119 | 77 | 167 | w | w |
| 24 | 18 | 030 | CAN (cancel) | 56 | 38 | 070 | 8 | 8 | 88 | 58 | 130 | X | X | 120 | 78 | 170 | x | x |
| 25 | 19 | 031 | EM (end of medium) | 57 | 39 | 071 | 9 | 9 | 89 | 59 | 131 | Y | Y | 121 | 79 | 171 | y | y |
| 26 | 1A | 032 | SUB (substitute) | 58 | 3A | 072 | : | : | 90 | 5A | 132 | Z | Z | 122 | 7A | 172 | z | z |
| 27 | 1B | 033 | ESC (escape) | 59 | 3B | 073 | ; | ; | 91 | 5B | 133 | [ | [ | 123 | 7B | 173 | { | { |
| 28 | 1C | 034 | FS (file separator) | 60 | 3C | 074 | < | < | 92 | 5C | 134 | \ | \ | 124 | 7C | 174 | | | \| |
| 29 | 1D | 035 | GS (group separator) | 61 | 3D | 075 | = | = | 93 | 5D | 135 | ] | ] | 125 | 7D | 175 | } | } |
| 30 | 1E | 036 | RS (record separator) | 62 | 3E | 076 | > | > | 94 | 5E | 136 | ^ | ^ | 126 | 7E | 176 | ~ | ~ |
| 31 | 1F | 037 | US (unit separator) | 63 | 3F | 077 | ? | ? | 95 | 5F | 137 | _ | _ | 127 | 7F | 177 | &#127; | DEL |

Source: www.LookupTables.com

**http://www.asciitable.com/**

# A popular encoding: ASCII

American Standard Code for Information Interchange

- We know how to encode integers as bits
- Assign every English letter (upper and lower case), digit, and various other symbols a number between 0 and 127
- To save text as bits, convert each character to a number, then convert that number to bits

**Problem: to be generally useful for text in many languages, 127 is not nearly enough.**

**¡We need more characters!**

# Unicode

A numbered set of characters for many languages

- The number assigned a character is its “code point”
- Has 137,439 characters
- Supports 146 different scripts
  (a script is the set of characters used in a language; different languages may share the same script)
- Many of the changes from Python 2 to Python 3 are related to making unicode the default

# Unicode

A numbered set of characters for many languages

- The number assigned a character is its “code point”
- Has 137,439 characters
- Supports 146 different scripts
  (a script is the set of characters used in a language; different languages may share the same script)
- Many of the changes from Python 2 to Python 3 are related to making unicode the default

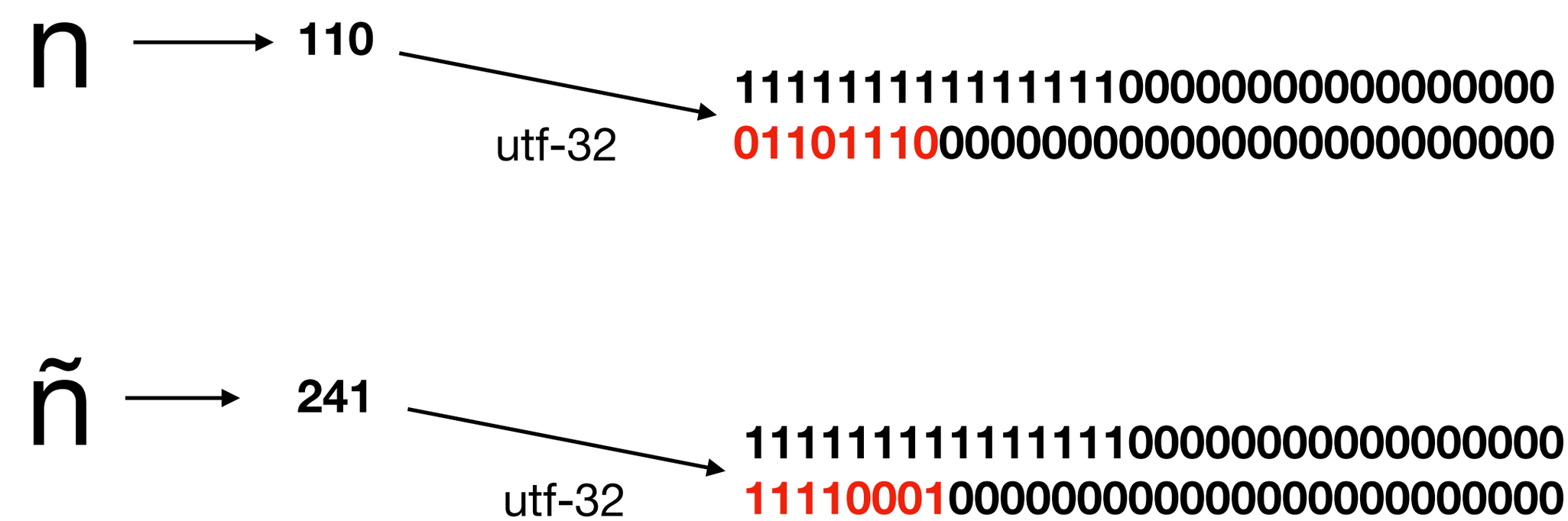
n has code point 110

ñ has code point 241

# Code points to bits

Unicode has multiple ways to convert code points to bits

- Useful for compatibility/efficiency

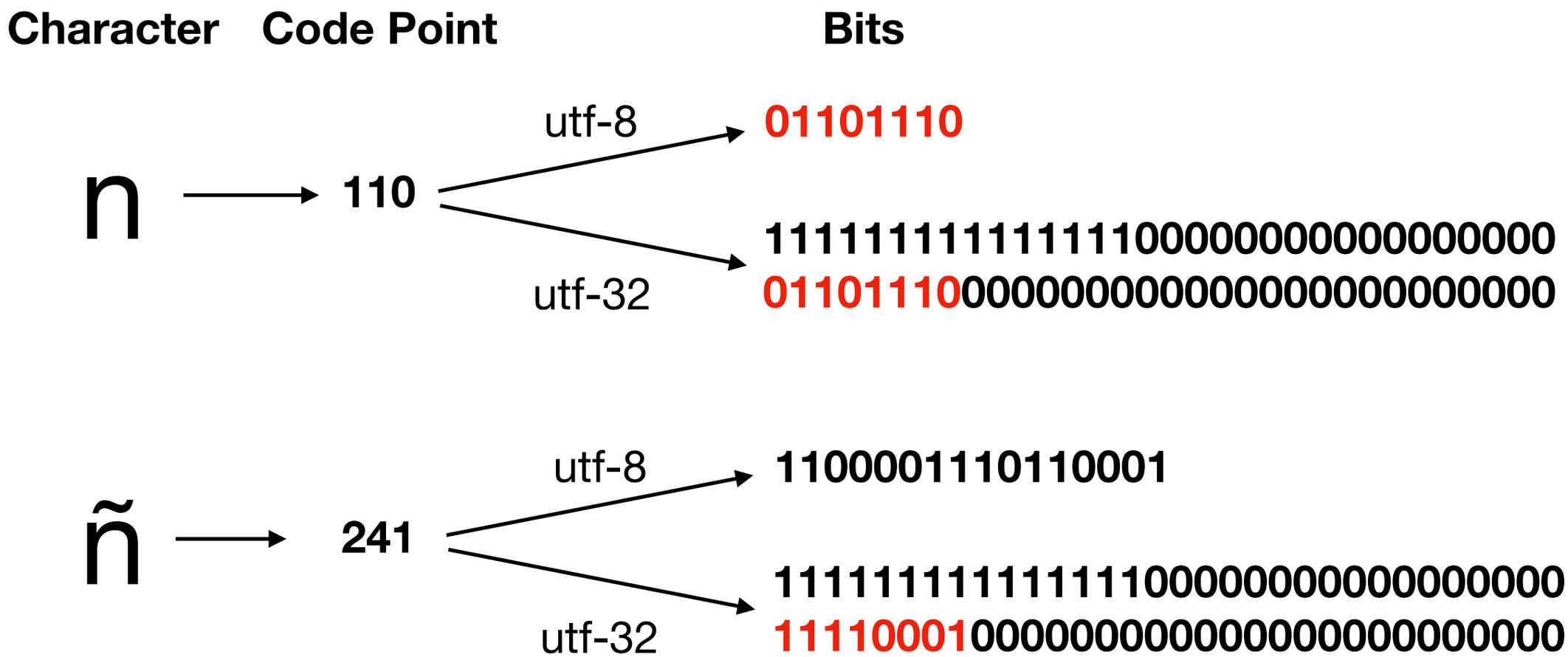



utf-32 is a way to convert code points to bits. If you use the same number of bits for every code point, you need lots of bits (because there are many code points)

# Code points to bits

Unicode has multiple ways to convert code points to bits

- Useful for compatibility/efficiency




utf-8 uses shorter sequences of bits for some characters and longer sequences for others. Depending on what characters you use, this may save bits.

# Today's Topics

Binary

Memory Organization

Data representation

- Numbers
- Colors and images
- Text
- Instructions

# Storing code in memory

Programs are also represented in memory

A program is just a bunch of instructions

- Instructions are things like add, subtract, multiply, compare

# Storing code in memory

Programs are also represented in memory

A program is just a bunch of instructions

- Instructions are things like add, subtract, multiply, compare

Encoding strategy

- Give each instruction a number, for example:

1: add
2: multiply
3: compare
…

# Storing code in memory

Programs are also represented in memory

A program is just a bunch of instructions

- Instructions are things like add, subtract, multiply, compare

Encoding strategy

- Give each instruction a number, for example:
  1: add
  2: multiply
  3: compare
  …
- We already know how to represent numbers as bits, so we can also represent the instructions that make up a program as bits

# Conclusion

Today we learned about

- Binary numbers
- How RAM stores binary numbers
- Units for expressing capacity and throughput
- All information can be represented with bits!
- Encoding strategies for numbers, images, text, and programs
  - the details don't matter for this class.  Just get a sense how to break down the problem of converting different types of data to ones and zeros