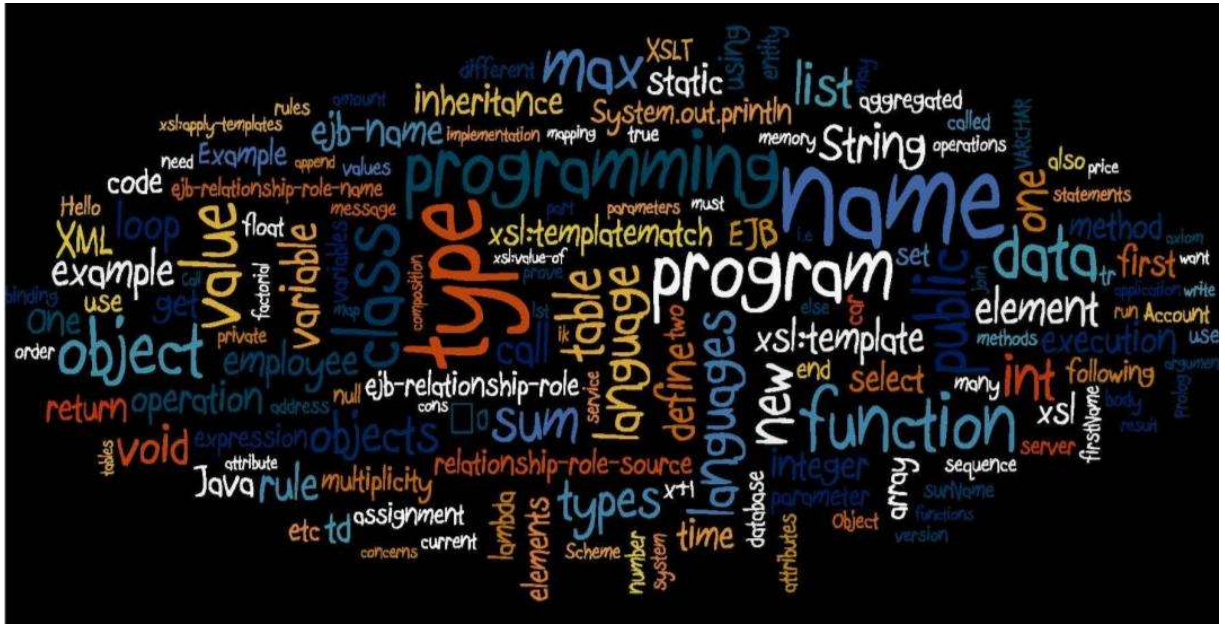
NUMBER SYSTEMS
SAFE PROGRAMMING

Computer Principles



for Programmers

Agenda

1. Base-10 programming in a binary world
2. Why do we care?

   because $(100)_{10}$ == $(01100100)_{2}$ $(huh?)_{00}$
3. How can we use it?

   without making mistakes
4. Why "Hexadecimal" numbers?

   because $(100)_{10}$ == $(64)_{16}$ is easier than binary

Activity

- Need a calculator during today's lecture
- Work with integer sizes
- Identify and solve an integer overflow bug.
- Work with colours in both Decimal and Hex RGB values.

Number Systems & Safe Programming

There are 10 types of people:
01 those who know binary
10 the other nine
11 those awaiting a ternary joke

Number Systems & Safe Programming

There are 10 types of people:
01 those who know ternary
02 those who don't
10 those expecting
another binary joke

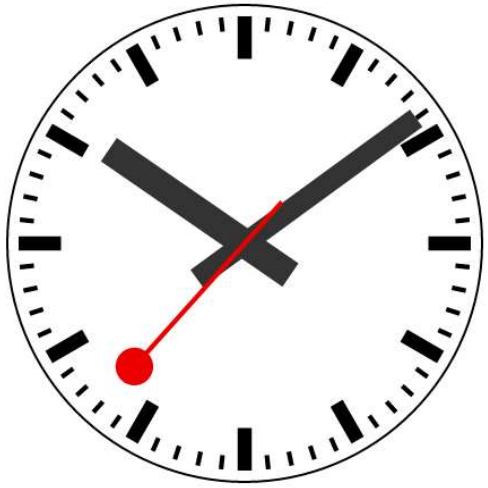
orders of magnitude of digital data

Decimal Value	Decimal Name	Sym -bol	SI prefix	Binary Value	Greek origin	d/load at 1 Gbps
1000^1	thousand	**k**	**kilo**	$2^{10} = 1024^1$	*chilioi* *"thousand"*	0.0000076 s
1000^2	million	**M**	**mega**	$2^{20} = 1024^2$	*mégas* *"great"*	0.0078125 s
1000^3	billion	**G**	**giga**	$2^{30} = 1024^3$	*gígas* *"giant"*	8 seconds
1000^4	trillion	**T**	**tera**	$2^{40} = 1024^4$	*teras* *"monster"*	2.28 hours
1000^5	quadrillion	**P**	**peta**	$2^{50} = 1024^5$	*pénte* *"five"*	97 days
1000^6	quintillion	**E**	**exa**	$2^{60} = 1024^6$	*héx* "six"	272.4 years
1000^7	sextillion	**Z**	**zetta**	$2^{70} = 1024^7$	*septem* "seven"	2,789 C
1000^8	septillion	**Y**	**yotta**	$2^{80} = 1024^8$	*októ* "eight"	285,616 M

1TB decimal = .90949TB binary 1TB binary = 1.0995TB decimal *but don't sweat it*

Humans

Analog Decimal



Computers

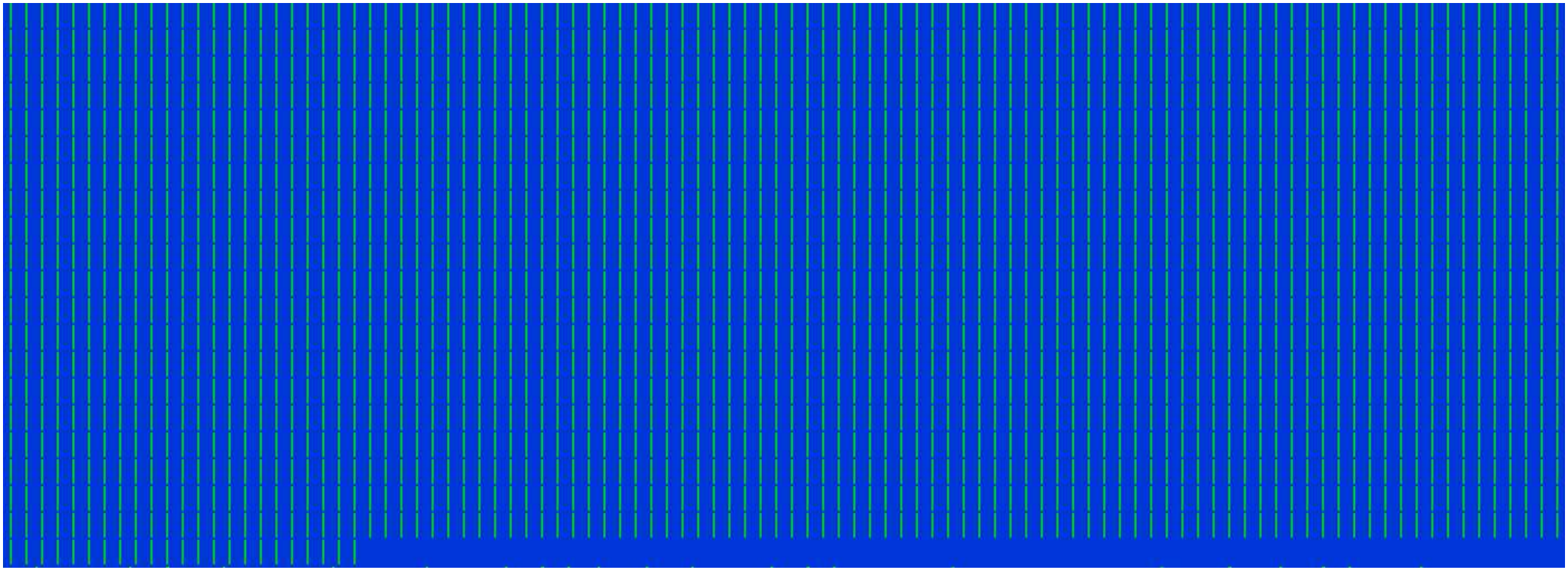
Digital Binary

	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
	128	64	32	16	8	4	2	1
H	○	○	○	○	●	○	●	○
M	○	○	○	○	●	○	○	●
S	○	○	●	○	○	●	○	●

Numbering Systems

How many ways can you represent 2023?

2023 toothpicks



Numbering Systems

How much is 2023?

- One, two, three, many.
- MMXXIII
- 00000111111100111
- 8:23 PM

Numbers, by themselves, are just…numbers. *Context matters*.

- The cardinal number 2,023 is different from the year 2023.
- Few or many depends on context: years, $$$, stars in galaxy
  2023 = two millennia $2,023 = 1 PC 0.000002% of stars *

* before James Webb Tel. found at lot more

8 bit byte $(8 \times 2)^2 = 2^8 = 256$

2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
128	64	32	16	8	4	2	1
0	1	0	1	0	1	0	1

Is **`01010101 char 'U'`** or **`decimal 85 ?`**

`if ('U' == 85)` is **`True`**.

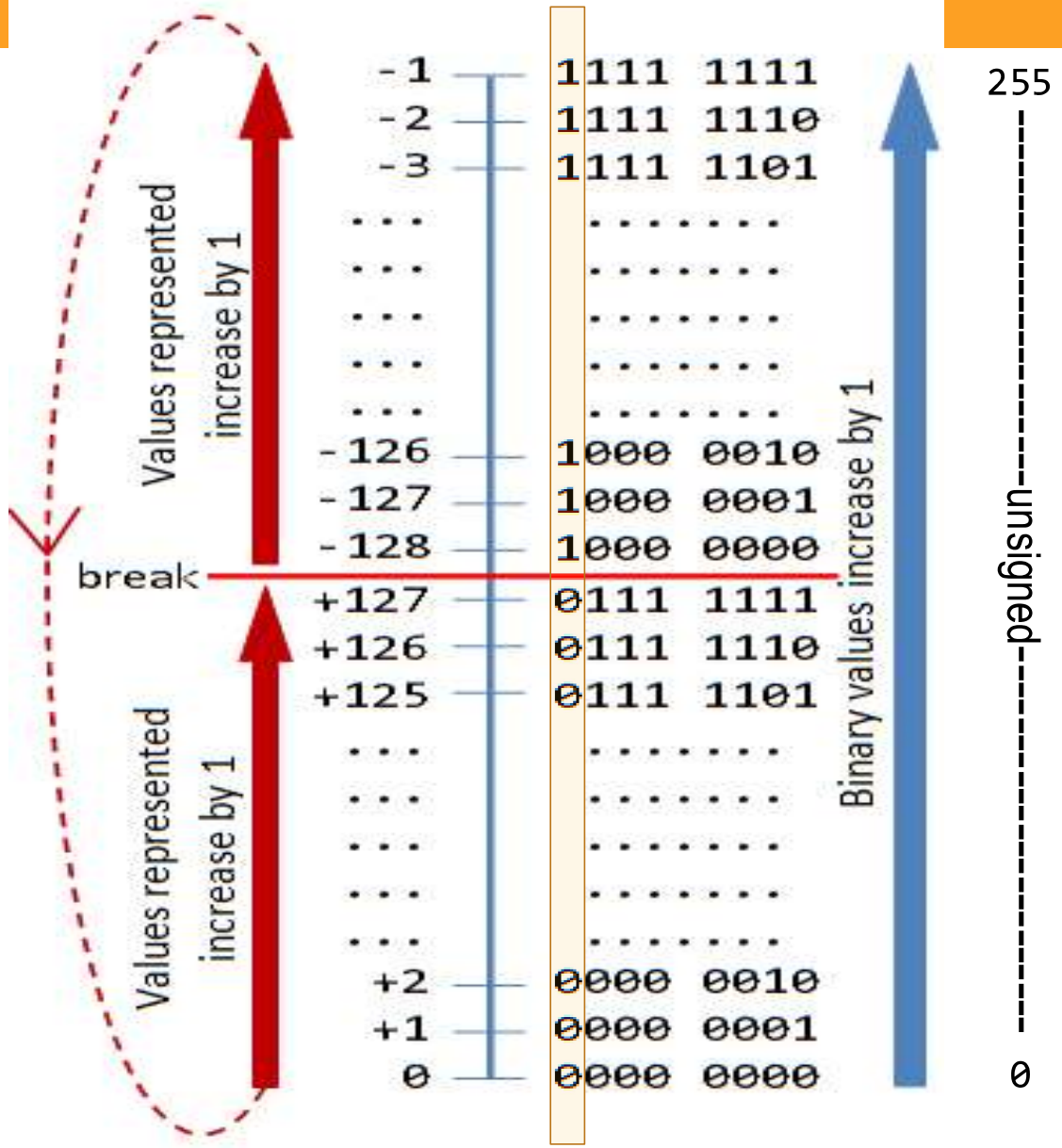
Only *binary* matters to computers.

Signed and unsigned within a single byte.

Integers

−/+ signed by default

* unsigned by request



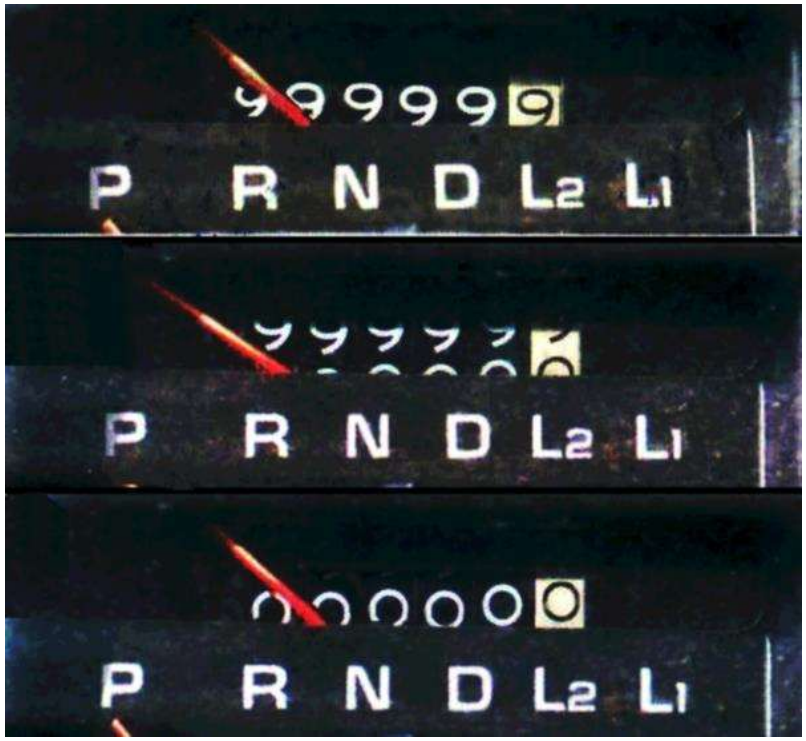
Binary Processing Cautions

What happens when you have too many?

add 1

to max =

overflow



Binary Processing Cautions

What happens when you have too many for the data type?

- *It doesn't matter.* Data type must hold more than enough.

But what happens when overflow *has* occurred?

- Overflow in C is ***undefined,*** *therefore unpredictable, and unstoppable -- program continues with binary values.*

unsigned	bits	signed
127	01111111	127
+1	00000001	+1
128	10000000	*–128**

unsigned	bits	signed
255	11111111	–1
+1	00000001	+1
*0**	00000000	0

- * overflow *

Binary is Binary, Context Matters

```
printf("'%c' ASCII %d", x, x);
```

unsigned/signed	unsigned	signed
'q' ASCII 113	'{' ASCII 123	'{' ASCII 123
'r' ASCII 114	'|' ASCII 124	'|' ASCII 124
's' ASCII 115	'}' ASCII 125	'}' ASCII 125
't' ASCII 116	'~' ASCII 126	'~' ASCII 126
'u' ASCII 117	'ᐧ' ASCII 127	'ᐧ' ASCII 127
'v' ASCII 118	'Ç' ASCII 128	'Ç' ASCII -128
'w' ASCII 119	'ü' ASCII 129	'ü' ASCII -127
'x' ASCII 120	'é' ASCII 130	'é' ASCII -126
'y' ASCII 121	'â' ASCII 131	'â' ASCII -125
'z' ASCII 122	'ä' ASCII 132	'ä' ASCII -124

Binary Processing Prudence

signed vs unsigned compares the *binary* values
results may be unexpected

unsigned	bits	signed	unsigned vs signed
255	11111111	-1	**`255 == -1`** is **`true`** ✘
			`255 > -1` is **`false`** ✘
			`255 < -1` is **`false`** ✓

- C# and Java do *not have unsigned integers* which are unsafe when compared with signed integers or when mixed in calculations.

➔ use *only* signed integer data types and

```
if (thisInt == thatInt) { /* will always be correct */ }
```

Bits and bit width
Know how many. Get a tattoo.

n* bits encode 2^n values = *bit width

- 8-bit `char` = 2^8 = 256 values (unsigned) **0 – 255**
- 16-bit `short` = 2^{16} = 65,536 values from **32,767** to **−32,768**
- 32-bit `long` = 2^{32} = 4,294,967,296 values ($65{,}536^2$)
  2,147,483,647 to **−2**,147,483,648
- 64-bit `long long` = 2^{64} = 1,2,3,*many* but slow to process

‼ `int short long` data types are `signed` by default, must declare `unsigned int` ***but don't.***

Binary Processing Problems

- *Gangnam Style* breaks YouTube, Dec. 2014
- PSY - GANGNAM STYLE(강남스타일)
- YouTube view counter goes to 64-bit width
- > 4.6B views since 15 July 2012

Binary Processing Problems

- ***If a counter tracks hundredths of a second, how many* days *would it take for an integer to overflow?***
  - SHORT_MAX (32,767) = almost right away…we can guess that.
  - LONG_MAX (2,147,483,647) = ⊞+R “calc”.
- The End is coming for UNIX: January 19, 2038 03:14:07
  - UNIX epoch 1970-01-01 00:00:00 + LONG_MAX seconds = The End
  - 32-bit Unix systems will go back in time 136.1 years due to overflow.
  - On 2033-05-17, timestamps will reach 2,000,000,000 seconds. Save the date for geek, nerd, and Unix user group parties!

Binary Processing Prudence

- **What will *never* overflow on a 32-bit platform**
  can *easily* overflow on a 16-bit platform
- INT_MAX **= 32,767 *or* 2,147,483,647** *platform*
  INT_MIN **= -32,768 *or* -2,147,483,648** *dependent*
- `INT_MIN <= x+y <= INT_MAX // primary effect`
- `INT_MIN - 1 = INT_MAX // side-effect`
  `INT_MAX + 1 = INT_MIN // side-effect`
- *Never* program for a side-effect. ***Never ever be clever.***

Binary Processing Prudence

C language **implementation** varies by compiler & platform

- C code could compile successfully but run differently.
- Data types have *minimum* sizes: platform's **`int`** width varies 8–64
- For *portability*, declare exact size/type you need `<stdint.h>`
  - int*nn*_t = minimum 100 times bigger than the biggest value expected.
- For *efficiency*, declare **`int`**
  - platform will use 16-bit or more but *know* the `sizeof(int)`
- Then *test*, **`test`**, **test**, *test*, **`test`** the edge cases
- **`gcc -std=c99 -pedantic -Wall -Wextra -Wshadow -Wconversion -Wstrict-overflow=5 -Wsign-conversion -Wformat=2 -Werror`**
  - examine compiler warning messages

Floats and Doubles — fractions

In binary,
what is the value between 0 and 1?

A float or a double, when your program is run,
Can cause you great trouble before it is done.
Values can muddle, though it all seems like fun,
They float like a bubble, between zero and one.

Floats and Doubles — *underflow*

```
                                                                  min. scale
1 / 1 = float  1.000000000000000000000000000000                    6 decimals
1 / 1 = double 1.000000000000000000000000000000000000000000000000 10 decimals

1 / 2 = float  0.500000000000000000000000000000
1 / 2 = double 0.500000000000000000000000000000000000000000000000

1 / 3 = float  0.33333334326744079589843750 0
1 / 3 = double 0.33333333333333314829616256247390000000000000000000

1 / 4 = float  0.250000000000000000000000000000
1 / 4 = double 0.250000000000000000000000000000000000000000000000
```

Floats and Doubles — *underflow*

```
                                                                  min. scale
1 / 5 = float  0.20000002980232238769531250                        6 decimals
1 / 5 = double 0.2000000000000001110223024625156                  10 decimals

1 / 8 = float  0.12500000000000000000000000000
1 / 8 = double 0.12500000000000000000000000000000

1 /10 = float  0.10000001490116119384765625
1 /10 = double 0.10000000000000005551115123125 78

1 /16 = float  0.06250000000000000000000000000
1 /16 = double 0.06250000000000000000000000000000
```

Floats and Doubles — fractions

In a binary system, what value is between 0 and 1?

- Every fractional value is an *approximation between 0 and 1*.
- **Not to be used for monetary or exact values in business**
  - DECIMAL data types are integers with implied decimal positions
    DECIMAL size 9.2 (precision.scale) contains 1,234,567.89
- BE VERY CAREFUL mixing integer and floating-point types
- Do not compare floats and doubles, only powers of 2 are ==
- In general, use doubles. Precision sacrifices scale as significant digits increase in value. The decimal point floats!
- Test edge cases of expected highest/lowest values.

Floats and Doubles — fractions

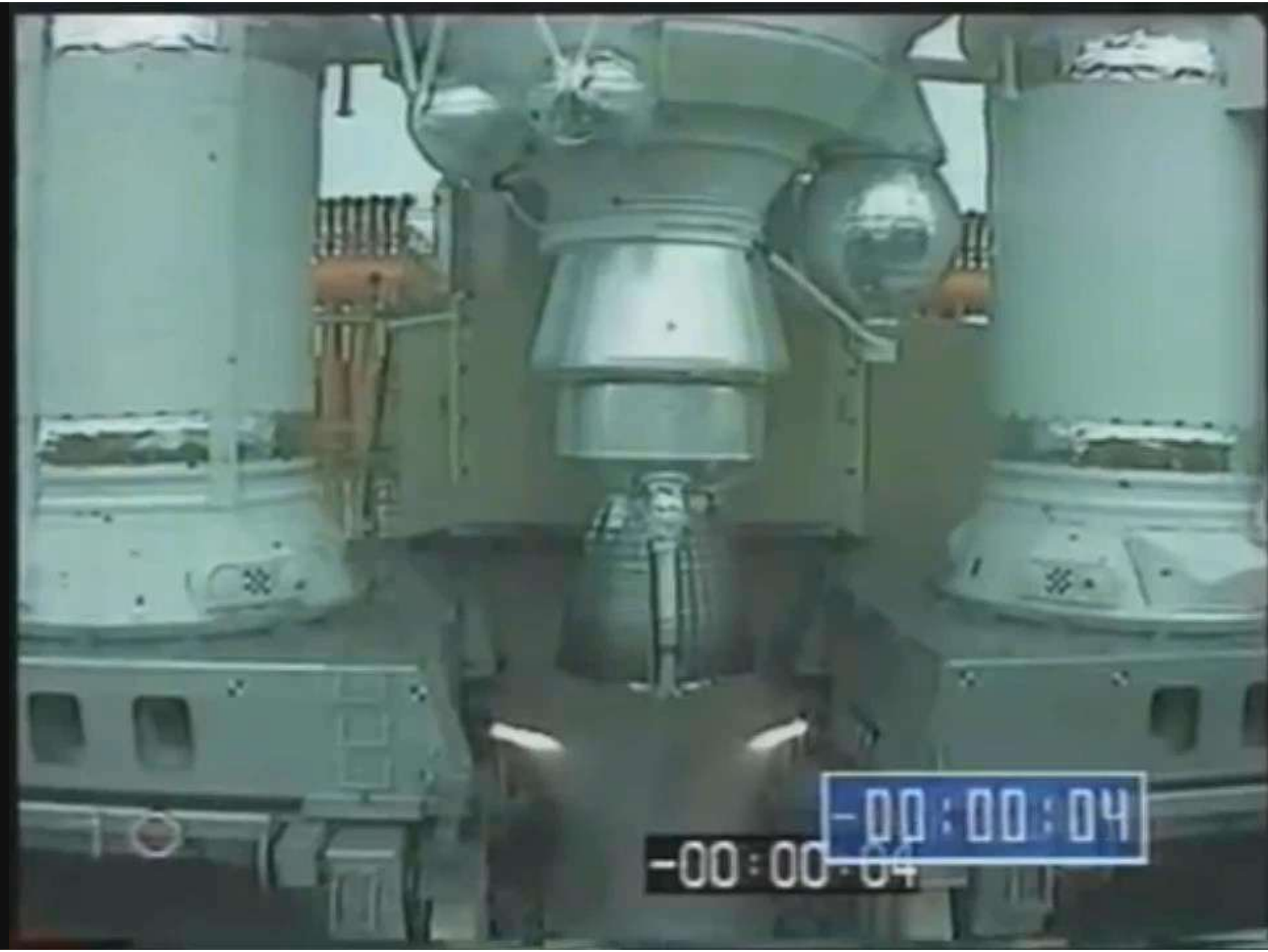
How many programmers does it take
to change a lightbulb?

1.00000000000001110223024625156540423631668

Never use a `float` or `double` when a `short` or `long` will do.

Ariane 5 rocket

- 10 years and $7B of development
- 1st flight June,1996
- Ariane 4 software
- Ariane 5 was faster
- 36 seconds of normal flight
- at 36.7 seconds… data conversion of **64-bit double to 16-bit integer**
- at 37 seconds… course correction
- at 39 seconds… $500M loss



Programming goes wrong on overflow

- Ariane 5 rocket self-destructs 40 sec after launch in 1996
  - casting a **64-bit `double`** to **16-bit `int`** needs 275B **`int`** variables
- For 6 hours, over 6000 people got a busy signal calling 911
  - 911 call routing stopped after a **counter reached maximum**
- Therac-25 radiation therapy device kills 6 (1985-1987)
  - **`error++; If (error != 0)`** means *do not irradiate patient*
    +1 to a byte 256 times equals zero. use **`isError`** set to **`true|false`**
- Toyota Sudden Unintended Acceleration kills ~400 (2002-2010)
  - data type **casting** altered values. 30% of **`switch`** without error traps

Error Traps

```
IF (true1) {
    do this;
}
ELSE IF (true2) {
    do that;
}
ELSE { /* error trap */
    unknownCondition = true;
}
...
IF (unknownCondition) {
    exit(86); // stop!!!!!!
}
```

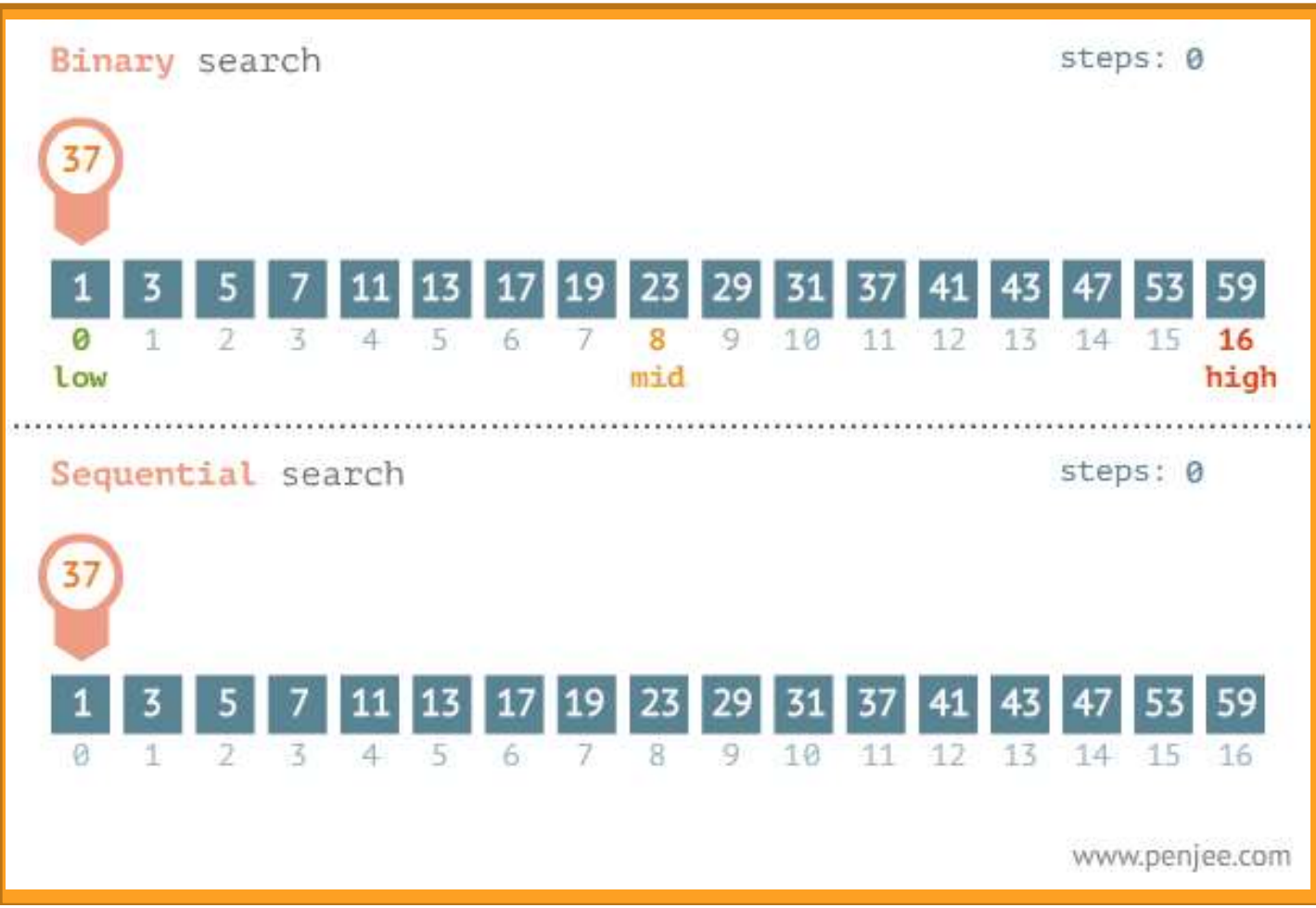
```
switch(char_variable) {
    case 'a':
    case 'A':
        do_this();
                // fall thru?!?!
    case 'b':
    case 'B':
        do_that();
        break; // don't forget!
    default:   // error trap
        exit(86); // stop!!!!!!
}
```

Defensive Programming

- Know your data! Use great care when mixing data types.
- Check for *type*_MAX and *type*_MIN. Overflow means unknown.
- Consider overflow for each intermediate value in calculations
  - `mid = (low + high) / 2; //` low + high can overflow before ÷ 2
  - Overflow can occur at *each* + − × operation
- Code for people, not for machines (compilers can optimize for efficiency)
  - Use comments; write code that reads easily.
- Use error traps: IF – THEN ends with unconditional ELSE
- Never ever be clever. Obfuscated C Code Contest

```
int main(int b,char**i){long long
n=B,a=I^n,r=(a/b&a)>>4,y=atoi(*++i),_=(((a^n/b)*(y>>T)|y>>S)&r)|(a^r);printf("%.8s\n",(char*)&_);}
```

Binary search → mid = (low + high) / 2 ; // *overflow bug*

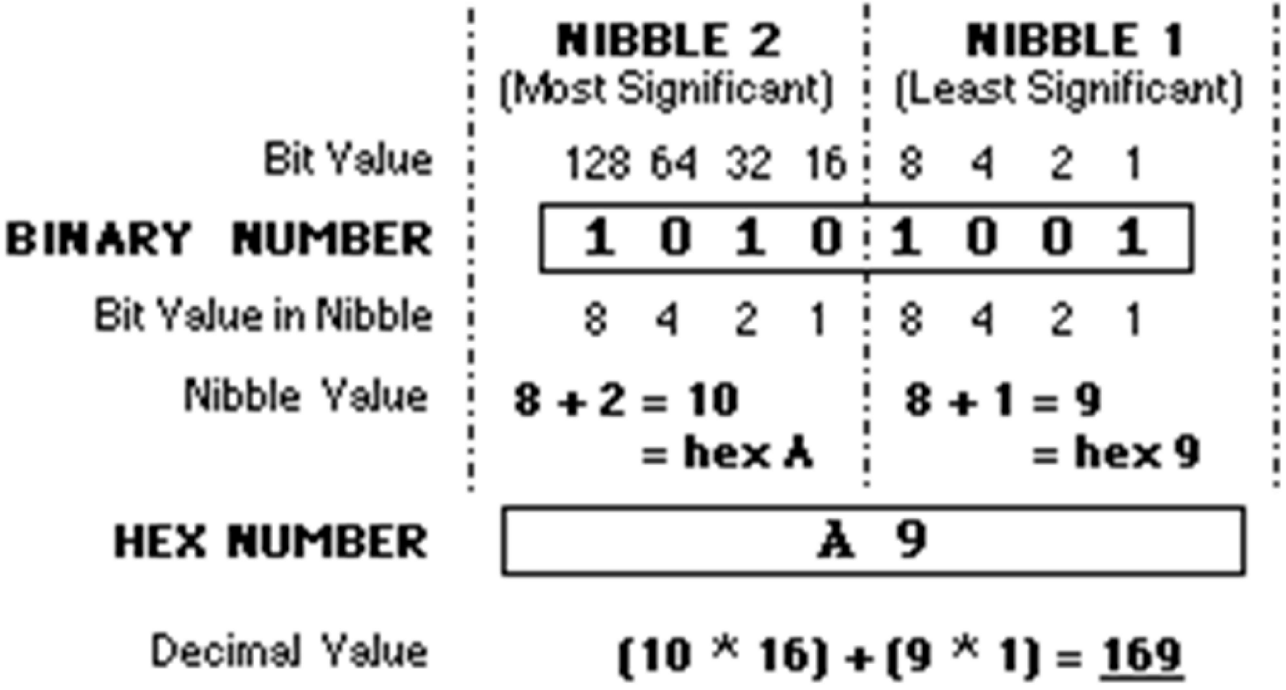



```
mid = (low + high) / 2
```
(*intermediate value*)

```
if (search > array[mid]){
    low = mid + 1
} else
if (search < array[mid]){
    high = mid - 1
} else {
  // search = array[mid]
}
```

Hexadecimal

- "Hex" is base 16
- Why? easier than binary but related
- 8 bit byte holds 256 values.
  16 * 16 = 256
  thus Hex



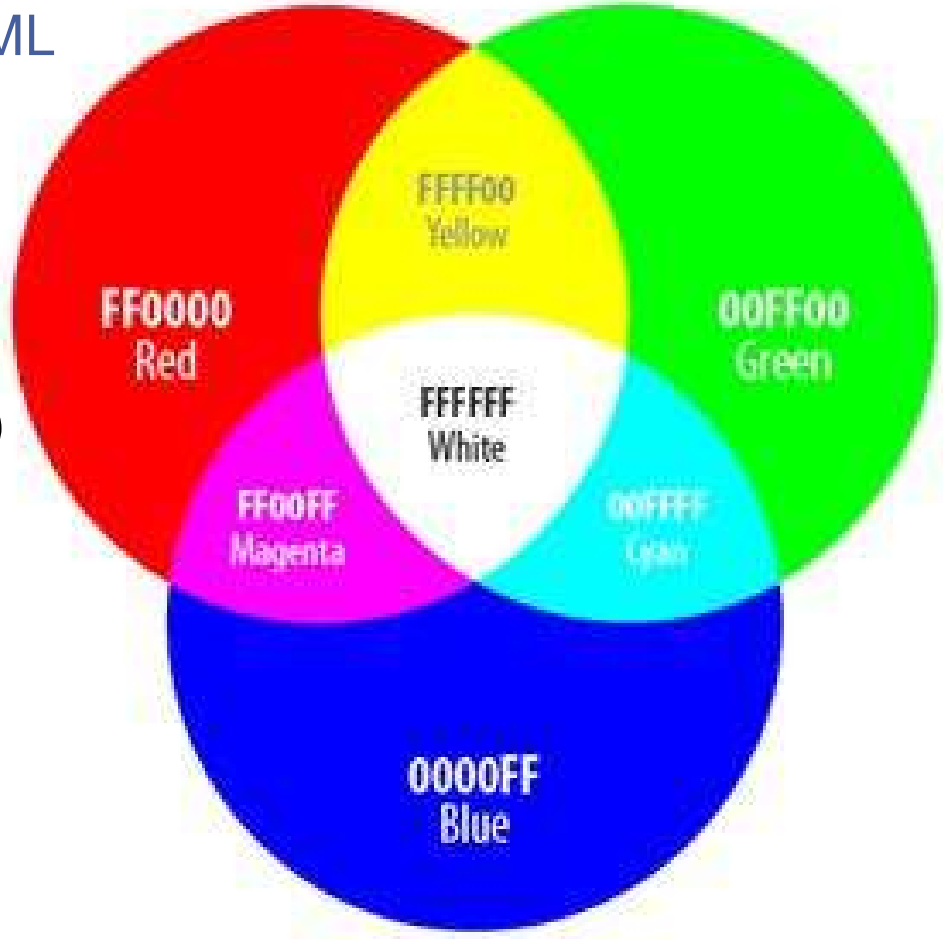
- 0–15: One – Nine, Able, Baker, Charlie, Dog, Easy, Fox

Hex

- **ASCII** more accurately, **US-ASCII**
  American Standard Code for Information Interchange
- **0000 1010 \n is 0x0A or Oh-Able**
- **0111 1111 $(127)_{10}$ is 0x7F or Seven-Fox**
- **1111 1111 $(255)_{10}$ is 0xFF or Fox-Fox**
- HTML & URLs use HEX values for special characters
- **0010 0000 $(32)_{10}$ is 0x20 or %20** = space char

HTML uses Hexadecimal color codes

- Hex codes of RGB triplets are used in HTML colour representations, as in: **#XXXXXX**
- Red, Green, and Blue values range from `00` (lowest intensity) to `FF` (highest intensity) in HEX
- `FFFFFF` is white (screen emits all colours) – presence of all visible wavelengths
- `00000` is black – absence of any visible wavelength
- eLearning: Hex numbers
  - eLearning Tutorials Login needs your Seneca credentials *and* a LinkedIn account



Notes

Not on the quiz
but worth having
an appreciation for…

Binary: Zero | One, On | Off, True | False

Leibniz (17th C) codifies the binary system

- 0 is the void, 1 is God.

Bit = **B**inary dig**it**

- unambiguous and unequivocal unit used to encode a message for transmission
- (Claude Shannon, 1948. Bell Labs)

Byte = 8 bits – the smallest unit of data

- 256 possible values { 0 – 255 }
- (W. Buchholz, 1956. IBM)

Why Binary?

Unambiguous and Unequivocal state:

- 0 or 1 for storage of a value in a **bit**
  - Electric current in RAM, magnetic polarity on HDD, magic in SSD
  - Computer switches/storage reliably hold only two possible states
- TRUE or FALSE to make a decision
  - hardware uses logic gates, software uses Boolean logic
  - All logic uses `True` | `False` → there is no *maybe*

Faster Processing:

- CPUs, ALUs, FPUs, GPUs run better in binary

Boolean Logic: Operator Truth Tables

If wishes were horses, beggars would ride.

Can you ride?

x — are wishes horses? **y** — have your own real horse?

x	NOT x
T	**F**
F	**T**

x	y	x AND y
T	T	**T**
T	F	**F**
F	T	**F**
F	F	**F**

x	y	x OR y
T	T	**T**
T	F	**T**
F	T	**T**
F	F	**F**

Safe, but *slow*, integer arithmetic

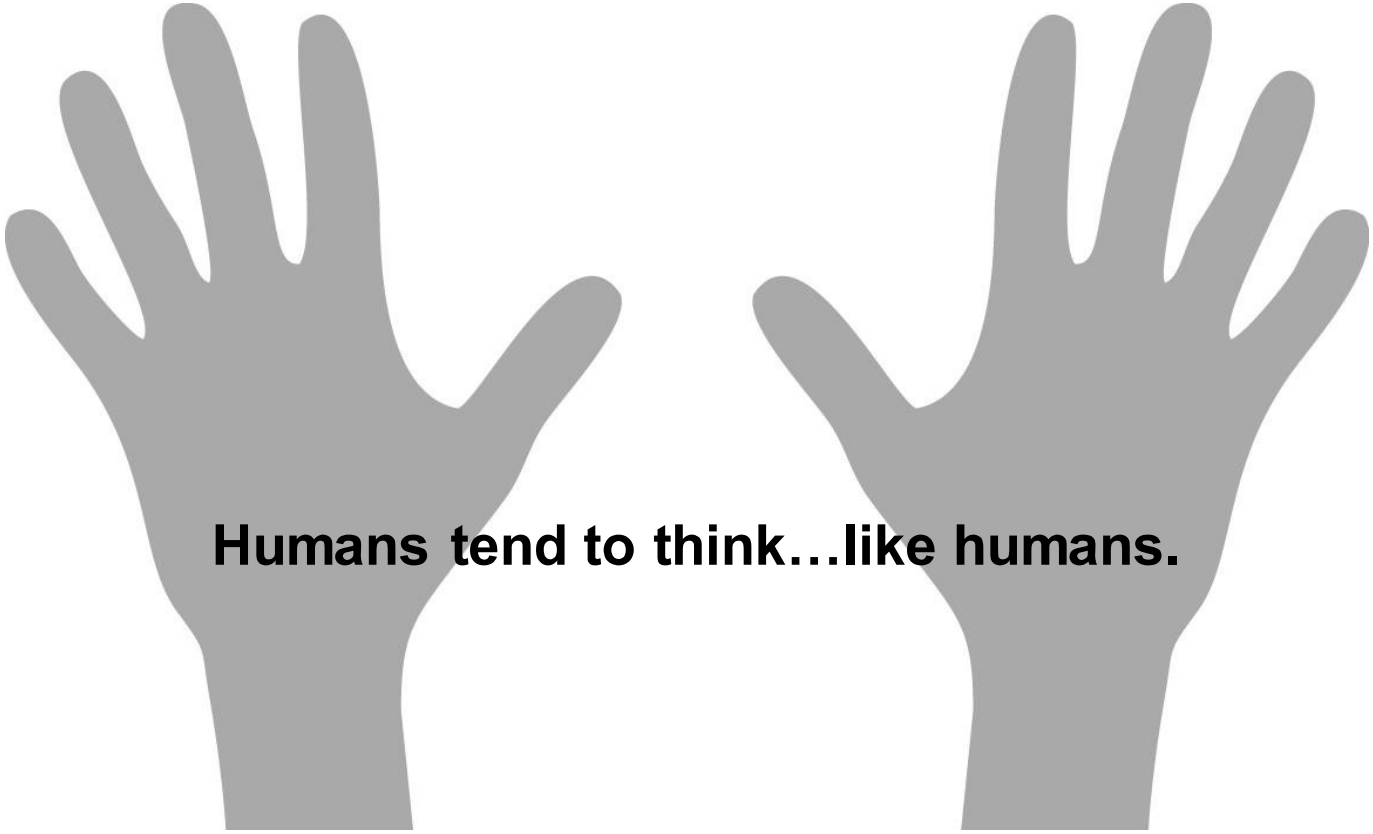
```
#include <limits.h>
int safe_add(int a, int b)
{
   if (a >= 0) {  // positive
      if (b > (INT_MAX - a)) {
         /* handle overflow */
       }
   } else {        // negative
      if (b < (INT_MIN - a)) {
         /* handle overflow */
      }
   }
   return a + b;
}
```

```
int safe_mult(int a, int b)
{ // efficiency: abs(a)>=abs(b)
   if (a == 0 || b == 0)
   {  return 0;  }
   int x = a;
   for (int i=2;i<=abs(b);i++)
   {
      x = safe_add(a,x);
   }
   if ( b < 0 ) { x = x * -1 }
   return x;
}
```

Bad Programming goes wrong

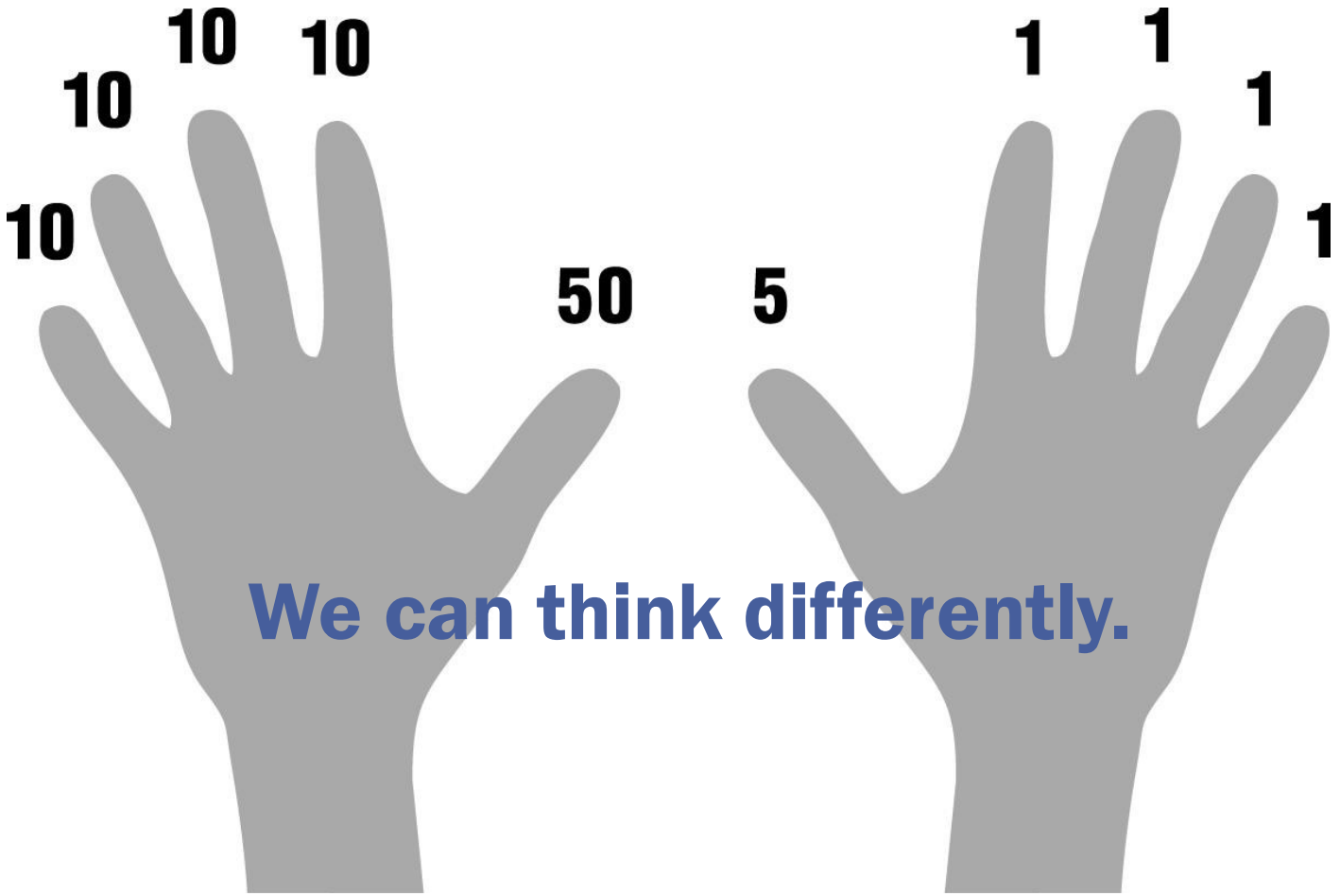
- Ping of Death crashes small computers/servers in 1995/6
  - **No validity checking** on fragmented IP packets allows reassembly to exceed max. packet length (16-bit value) causing buffer overflow
- Mars climate orbiter crashes in 1999
  - **USC/Imperial versus Metric** contest ended in a split decision
- Antarctica's ozone layer hole discovered 7 years late
  - NASA software **ignored values** outside range of expected measurements. Tell-tale data was ignored from 1978 – 1985.
- Lack of Random seeding could win you $1 million
  - Don't use **`rand()`** without first **seeding** with **`srand(NULL)`**

Numbering Systems

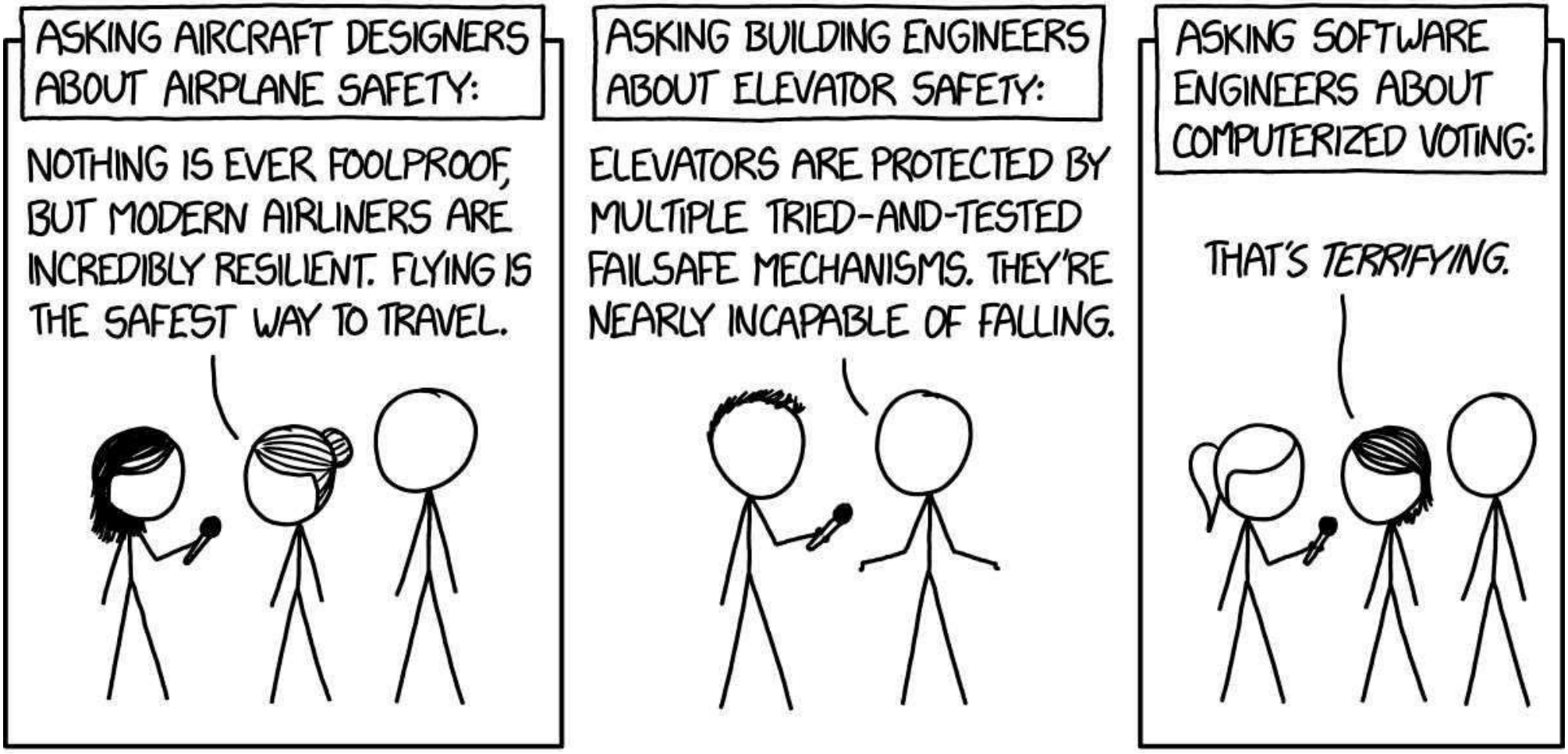


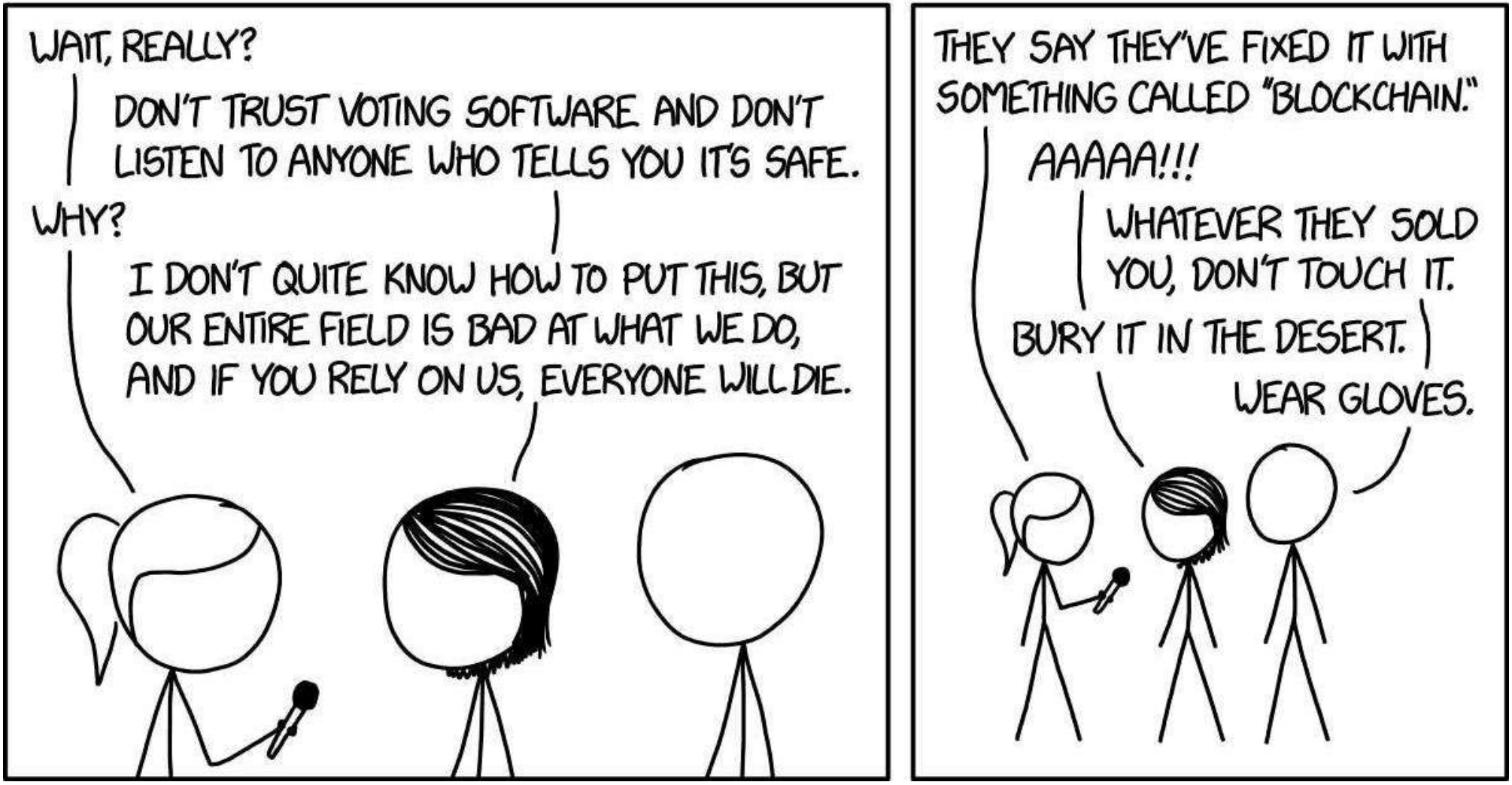
Humans tend to think…like humans.

Numbering Systems



We can think differently.


ASKING AIRCRAFT DESIGNERS ABOUT AIRPLANE SAFETY:
NOTHING IS EVER FOOLPROOF, BUT MODERN AIRLINERS ARE INCREDIBLY RESILIENT. FLYING IS THE SAFEST WAY TO TRAVEL.
ASKING BUILDING ENGINEERS ABOUT ELEVATOR SAFETY:
ELEVATORS ARE PROTECTED BY MULTIPLE TRIED-AND-TESTED FAILSAFE MECHANISMS. THEY'RE NEARLY INCAPABLE OF FALLING.
ASKING SOFTWARE ENGINEERS ABOUT COMPUTERIZED VOTING:
THAT'S TERRIFYING.


WAIT, REALLY?
DON'T TRUST VOTING SOFTWARE AND DON'T LISTEN TO ANYONE WHO TELLS YOU IT'S SAFE.
WHY?
I DON'T QUITE KNOW HOW TO PUT THIS, BUT OUR ENTIRE FIELD IS BAD AT WHAT WE DO, AND IF YOU RELY ON US, EVERYONE WILL DIE.
THEY SAY THEY'VE FIXED IT WITH SOMETHING CALLED "BLOCKCHAIN."
AAAAA!!!
WHATEVER THEY SOLD YOU, DON'T TOUCH IT.
BURY IT IN THE DESERT.
WEAR GLOVES.

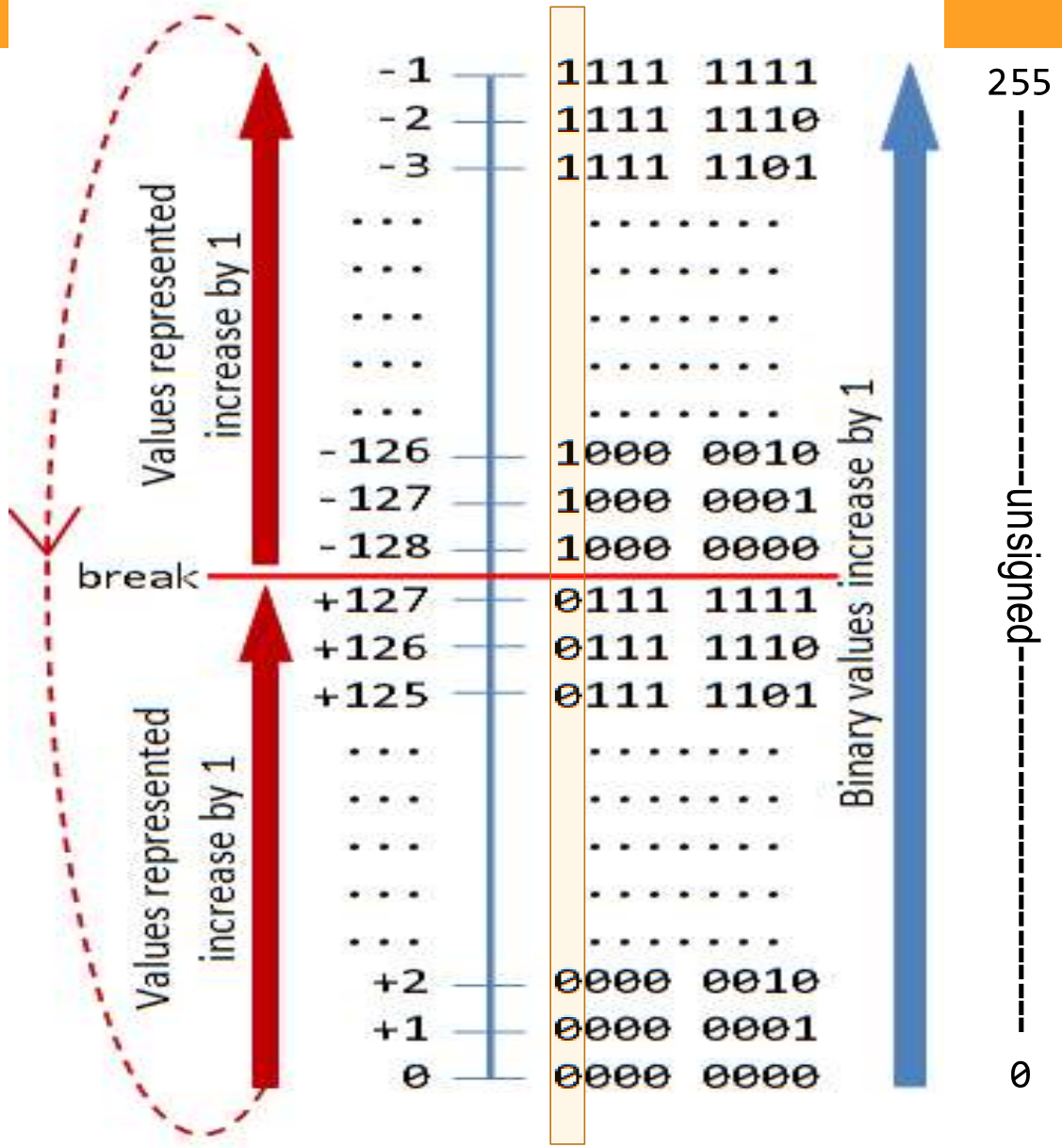
Two's Complement

```
+127   01111111   max positive
-128   10000000   min negative
= -1   11111111   correct

  +3   00000011   matching
  -3   11111101   pos and neg
=  0   00000000   always zero

+127   01111111   max positive
  +1   00000001   add one
-128   10000000   overflow

-128   10000000   min negative
  -1   11111111   sub one
+127   01111111   overflow
```



Floats and Doubles — integer precision

`float` Single-precision floating-point (32-bit base-2)

- Precision: 24 bits left & right of decimal, 6 decimals accuracy
- Precision limit: +/- 16,777,216 exact integer values, 2^{24}

`double` Double-precision floating-point (64-bit base-2)

- Precision: 53 bits left & right of decimal, 15 decimals accuracy
- Precision limit: +/- 9,007,199,254,740,992 (9 quadrillion)
- **`double`** calcs take longer to run but are safer than **`float`**
- Actual properties of float and double are **unspecified in C**

What is a Number System?

- The "information stored and processed inside a computer system is represented in machine language which is only about 0s and 1s."
- Therefore, the memory contents (which are "all digits and numbers") would be different from what we type (the strings of letters as an ex.) and we should have some ways to present them.
- A Number System is a "writing system for expressing numbers;" It is a "mathematical notation for representing numbers of a given set," using digits or other symbols in a consistent manner.
- A Number System could also referred to as "a numeral system" or "system of numeration." In this lecture, we will look at most common Number Systems.

Why do we need different number systems in computing? And Common Number Systems

- A computer system "only understands numbers." Therefore a computer "translates" all input including letters, words, special characters to numbers.
- While we, as human beings, use our "Decimal" number system with 10 (0 - 9) digits, a computer understands "Binary" number system that has only 2 (0 and 1) digits. Different number systems are just "different views to same numbers, but in different systems."
- To represent binary numbers more concisely, we use "Hexadecimal" number system that uses 10 digits (0 – 9) and 6 symbols (A – F).
- Each of these digits or symbols, represent different values "depending on their position in a given number." – more on this soon.

Common Number Systems – Value of Digits in a Number

- For each of the number systems mentioned, "the value of a digit in a number" is determined by:

1. "The digit itself." (As an example in decimal, each of the digits 0 - 9 have different values)

2. "Position of the digit." (As an example in decimal: 10s, 100s, 1000s, 10000s, etc.)

3. "The base of the number system." (which is its total number of digits.) Decimal, binary, and hexadecimal use bases 10, 2, and 16 respectively.

Common Number Systems – Decimal Number System

- Decimal Number System is the one which is "human preferred" and the one we use "in everyday life."

- "It has base 10 as it uses 10 digits (0 – 9)." The "values of digits increase from right to left:" 1s, 10s, 100s, 1000s, etc.

- "Each successive position represents a specific power of base 10." Take 1234 as an example. It has 4 in ones position, 3 in tens position, 2 in hundreds position, and 1 in thousands position and could be expressed as:

$$1234 = (1 \times 10^3) + (2 \times 10^2) + (3 \times 10^1) + (4 \times 10^0)$$
$$= (1 \times 1000) + (2 \times 100) + (3 \times 10) + (4 \times 1)$$

Binary number system

- Binary number system is "the one that a computer or machine understands."
- It uses only two digits: 0 and 1, which means "its base is 2" (or it's a Base 2 number system.)
- Like Decimal, "the values of digits increase from right to left" and "each successive position represents a specific power of base" (which is 2 in this case.)
- Take $(1001)_2$ as a binary number example. It could be expressed as:

$$(1001)_2 = (1 \times 2^3) + (0 \times 2^2) + (0 \times 2^1) + (1 \times 2^0)$$
$$= (1 \times 8) + (0 \times 4) + (0 \times 2) + (1 \times 1) = (9)_{10}$$

Importance of Binary number system

- "All data transfer, storage, and processing done by a microcomputer is performed digitally using binary" (base 2) codes.

- This binary system "translates every character entered in the computer into a set of 1's and 0's." As an example, computer represents the letter 'A' as 1000001 (or the code 65 in decimal.)

- The advantage of binary coding over other methods is that "a sequence of only two possible states is required to represent a character in the electronic circuits of the computer." Therefore, the smallest piece of information that needs to be stored in memory is a single binary digit.

Importance of Binary number system (Cont'd)

- A single binary digit is called a "bit." Different groupings of bits are used to represent different characters. A collection of eight bits is called a "byte."

- "One byte can represent any of 256 characters" (2^8 =256). As an example, the word "bit" would require a total of three bytes of memory, one byte for each character in the word.

- Since we are primarily concerned with how many characters the memory of a computer can hold, "memory size is referred to in units of bytes."

Importance of Binary number system (Cont'd), and How to convert from Decimal to Binary and vice versa

- In binary arithmetic, the power of 2 that is closest to 1000 is 2^{10} (2^{10} = 1024). Therefore, "in computer jargon, the prefix kilo stands for 1024." Frequently, the word kilobyte is abbreviated "kB". (Note the "k" is lower case because "K" in metric is reserved for the Kelvin temperature scale.)
- a chip with 256kB of RAM can store 262,144 characters in memory (256 X 1024 = 262,144).
- eLearning: binary and bits, binary numbers
  - eLearning Tutorials Login needs your Seneca credentials *and* a LinkedIn account since LinkedIn bought Lynda

Hexadecimal number system

- Hexadecimal (or base 16 number system) uses "16 digits: 0 to 9, and A to F." Letters A to F represent digits starting from 10 (A = 10, B = 11, C = 12, D = 13, E = 14, F = 15.)

- Like other number systems, "the values of digits increase from right to left" and "each successive position represents a specific power of base" (which is 16 in this case.)

- Take $(19FD)_{16}$ as a hexadecimal number example. It could be expressed as:

$$(19FD)_{16} = ((1 \times 16^3) + (9 \times 16^2) + (F \times 16^1) + (D \times 16^0))$$
$$= (1 \times 4096) + (9 \times 256) + (15 \times 16) + (13 \times 1) = (6653)_{10}$$