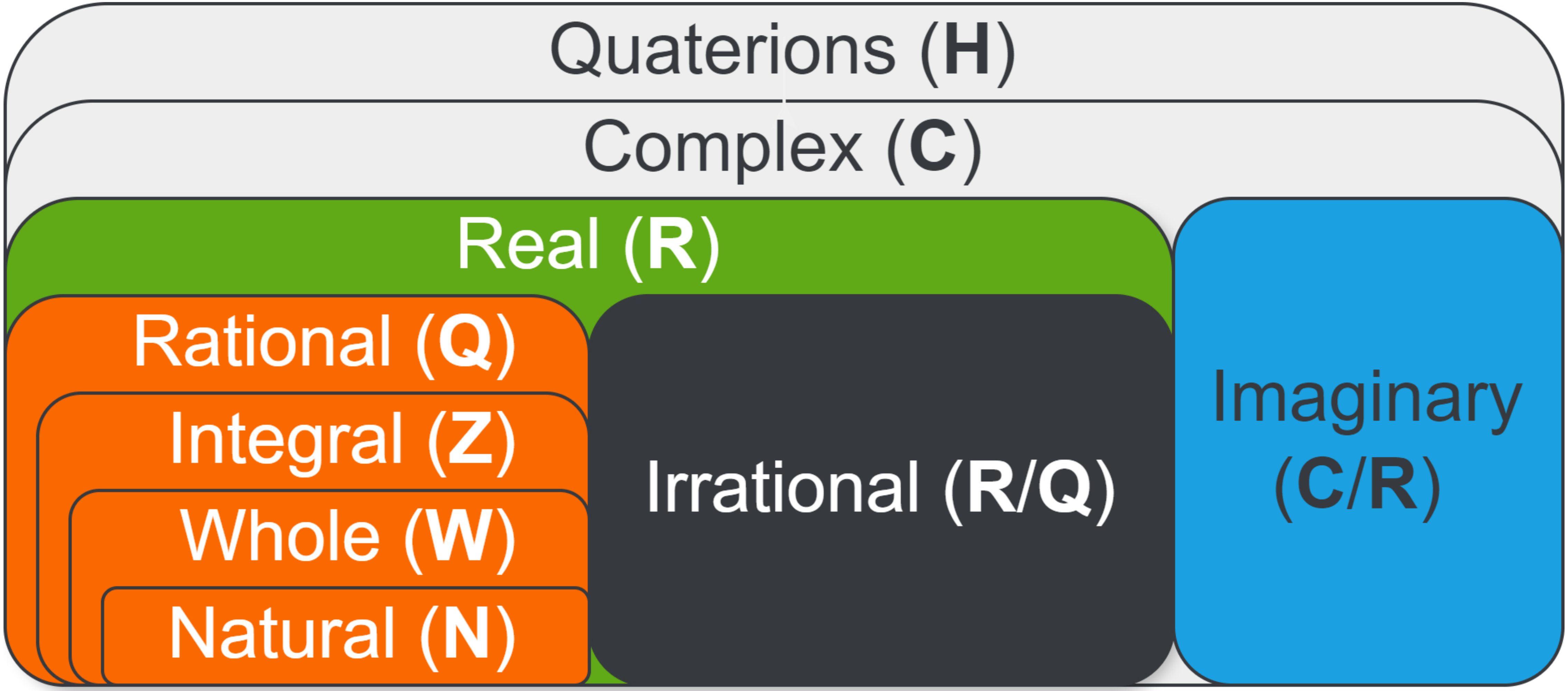
Numbers in computers

Some key-points about

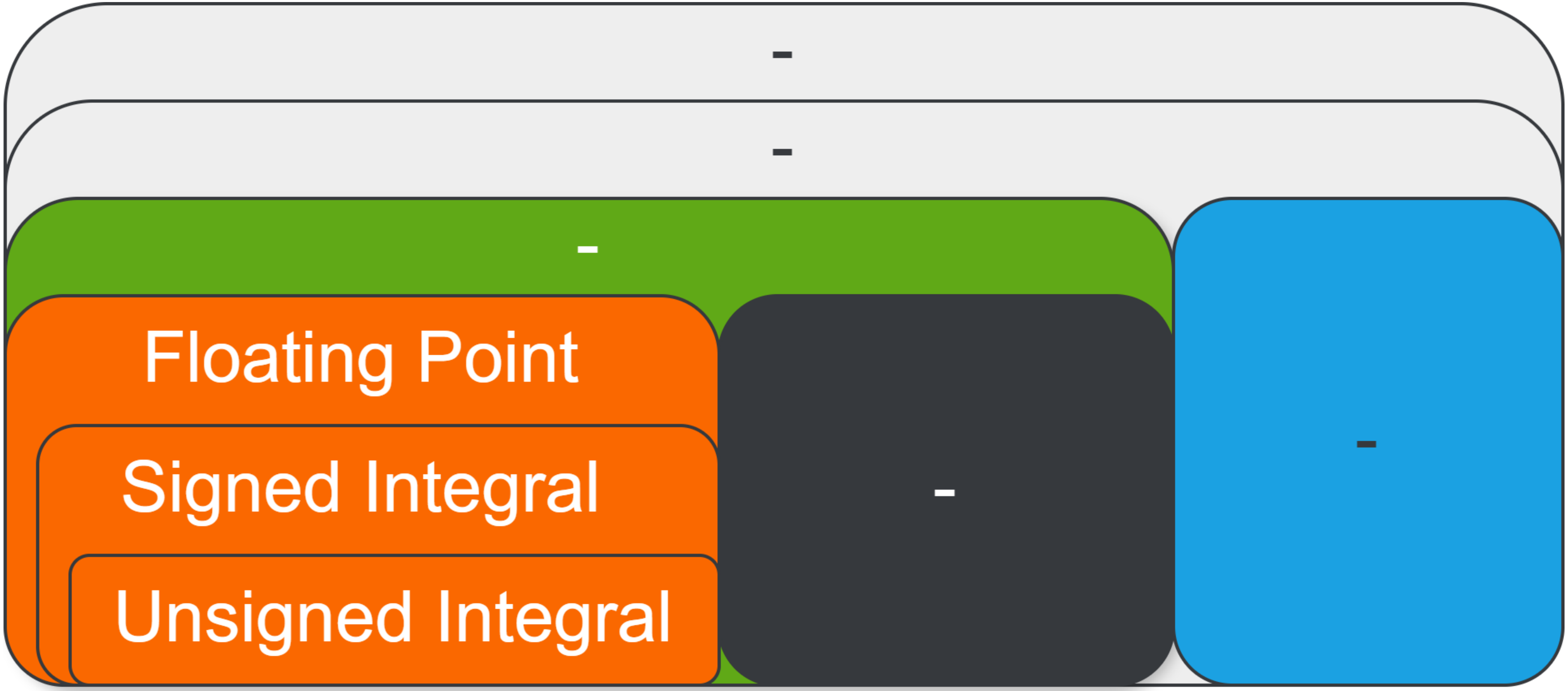
- There is no universal definition of a number. In that course, we will use something very naïve like:
  - An object used to denote an arithmetic value.
  - A basic math unit used to count, measure, and label things.
  - An abstraction to quantify some measurements.

- There are different sets (categories) of numbers.
- There are different representations (systems) of numbers.
- Numbers are a simple abstraction but are difficult in practice.

Sets we care about right now



What we are limited in reality



To be honest, look in the future

- It is even worse than on the previous slide:
  - All floating points cannot represent all real numbers
  - All signed integers cannot represent all integer numbers
  - All unsigned integers cannot represent all whole numbers

- Even more:
  - All floating points cannot represent all signed and unsigned integers
  - All signed integers cannot represent all unsigned integers

Numbers in computers

- Computers operate through the CPU.
- CPU performs computations with ALU and FPU.

- ALU and FPU operate through registers of different sizes.
- Registers size is limited. We refer to that size as width.

- That width limits how wide is range of numbers we can represent.
- If something cannot fit in the register, we emulate it ourselves.

Examples (1)

3								2								1								0							
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	1	0	1	1

3								2								1								0							
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	1	1	1	1	0	1	0	1

3								2								1								0							
0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	1	1	1	1	0	1	0	1

3								2								1								0							
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Examples (2)

3								2								1								0							
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	1	0	1	1

3								2								1								0							
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	1	1	1	1	0	1	0	1

3								2								1								0							
0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	1	1	1	1	0	1	0	1

3								2								1								0							
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Side note about endian

There are different ways to store byte sequences in memory, they are called endians. The two most common ways are little and big.

0x8D01	0x8D02	0x8D03	0x8D04	Addr
3	2	1	0	Little
0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 1	0 0 0 0 1 0 1 1	
0	1	2	3	Big
0 0 0 0 1 0 1 1	0 0 0 0 0 0 0 1	0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0	

Integers

- Most personal computers use 32 or 64-bit architecture.
- Due to that, the most widespread and commonly supported
  - uint8/int8 (or u8/i8, unsigned char/char) – 8 bits wide integers
  - uint16/int16 (or u16/i16, unsigned short/short) – 16 bits wide integers
  - uint32/int32 (or u32/i32, unsigned int/int) - 32 bits wide integers
  - uint64/int64 (or u64/i64, unsigned long/ long) – 64 bits wide integers

- For C and C++, unsigned long long/long long on Windows.
- Everything wider than 32/64 bits is emulated.

Here is our counter

- Let us start from a simple example. What if we have 4 digits and we want to describe some number with it? Here is our counter:

0000

Incrementing

- Let it be very simple, it is counter, so it is incrementable and decrementable, but let us focus on incrementing.

0008

Decrementing

- Let it be very simple, it is counter, so it is incrementable and decrementable, but let us focus on incrementing.

0005

Example

- With our counter we are able to represent whole numbers from 0 up to the 9999. Let us stick to one number:

2908

Example

- We can modify logic of our counter, and we will be able to count some fractions. We add imaginary dot between pair of digits.

29.08

Example

- With that imaginary modification we can represent, again, 9999 different rational numbers with precision (epsilon) of .01

29.08

Example

- Epsilon is crucial. It is the smallest representable non-zero value.
- It also defines difference between two closest numbers.

29.09

Example

- While we still can represent different 9999 numbers, only 99 of those 9999 are whole numbers. We may be unhappy with that.

99.99

Example

- Now we represent 999 whole numbers, but we can use rational numbers only with precision of .1. We may be unhappy with that.

999.9

Example

❖ Now we can represent numbers of .001 precision, but we can use only 9 whole numbers. We made the initial problem even worse.

9.999

Fixed precision arithmetic

- The concept shown on previous slide what basically is fixed precision is. We trade of between whole and rational part.

00.00

Fixed precision arithmetic

- We knew that computer operate with limited chunks of data. Due to that, our computer is similar in limitations to that counter.

00.00

We can do better

$$10^{0} \times 0.00$$

exponent

mantissa

We can do better

$$10^{0} \times 0.00$$

00.00

Everything is okay

$$10^{0} \times 0.61$$

00.61

Everything is okay

$$10^{0} \times 0.99$$

00.99

Everything is okay

$$10^{0} \times 9.99$$

09.99

Well, something is not okay

$$10^{1} \times 0.00$$

Multiple representation of 0 is not okay

00.00

And again – something is not okay

$$10^{2} \times 0.00$$

Multiple representation of 0 is not okay

00.00

And again – something is not okay

$$10^{3} \times 0.00$$

Multiple representation of 0 is not okay

00.00

And again – something is not okay

$$10^{9} \times 0.00$$

Multiple representation of 0 is not okay

00.00

Let us introduce scientific notation

$$10^{1} \times 1.00$$

10.00

And further

1

10 × 1.03

10.03

And further

$$10^{1} \times 1.67$$

10.67

And further

1

10 × 1.99

10.99

And further

$$10^{1} \times 4.85$$

40.85

And further

$$10^2 \times 5.74$$

507.40

And further

$$10^{3} \times 5.74$$

5074.00

And further

$$10^4 \times 5.74$$

50740.00

And further

$$10^5 \times 5.74$$

507400.00

And further

$$10^{9} \times 5.74$$

5074000000.00

And further

$$10^9 \times 9.99$$

9990000000.00

We can tradeoff whole for fraction

$$10^{0} \times 0.00 \times 10^{-3}$$

Simplify previous expression

biased exponent

$$10^{0-3} \times 0.00$$

mantissa

We can tradeoff whole for fraction

$$10^{-3} \times 1.00$$

00.01

The same but scaled

$$10^{-3} \times 1.03$$

0.0103

And again

$$10^{-3} \times 1.67$$

0.0167

And again

$$10^{1-3} \times 1.99$$

0.0199

And again

$$10^{-3} \times 4.85$$

0.0485

And again

$$10^{2-3} \times 5.74$$

0.574

And further

$$10^{33} \times 5.74$$

5.74

And again

$$10^{4-3} \times 5.74$$

57.40

And again

$$10^{5-3} \times 5.74$$

574.00

And further

$$10^{9\text{-}3} \times 5.74$$

5074000.00

And further

$$10^{9-3} \times 9.99$$

9990000.00

And for our computer

$$10^{9-3} \times 9.99$$

9990000.00

In binary

$$2^{00-11} \times 0.00$$

And again

$$2^{00-11} \times 0.11$$

And again

$$2^{00-11} \times 1.01$$

And again

$$2^{00-11} \times 1.11$$

And again

$$2^{01-11} \times 1.00$$

And again

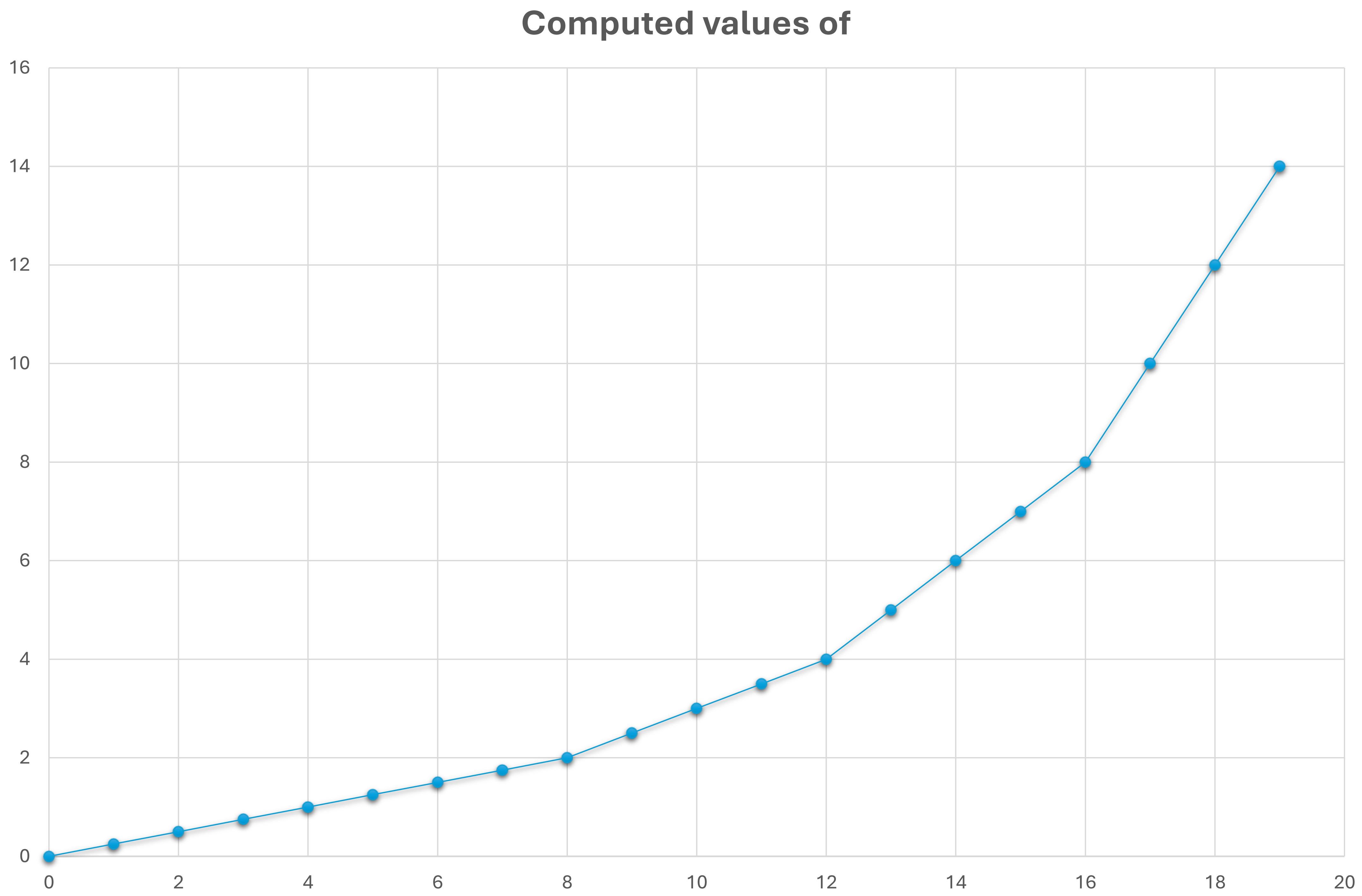
$$2^{01-11} \times 1.10$$

And again

$$2 \times 1.11^{11-11}$$

One final adjustment for memory

$$2^{11-11} \times 1.11$$


Computed values of
0
2
4
6
8
10
12
14
16
0
2
4
6
8
10
12
14
16
18
20

So, in reality

- Idea from previous slides is how floating-point work designed through the IEEE-754, the most widespread floating point implementation.
- There are 4 formats for it, all of them are named for how much space they occupy:

- 32-bits width – referred as single precision floating point, or float
- 64-bits width – referred as double precision floating point, or double

- 16-bits width – exotic , highly depending on platform
- 128-bits width – exotic, highly depending on platform

Single precision IEE-754 float

3								2								1								0							
S	E	E	E	E	E	E	E	E	M	M	M	M	M	M	M	M	M	M	M	M	M	M	M	M	M	M	M	M	M	M	M
8	7	6	5	4	3	2	1	0	-1	-2	-3	-4	-5	-6	-7	-8	-9	-10	-11	-12	-13	-14	-15	-16	-17	-18	-19	-20	-21	-22	-23

- That is a layout of our floating point in the little-endian model.
- We have 32 bits dedicated for that floating point in total.
- Indexes on the image show bit index before the “point”

- Orange – 1 bit for sign bit, denoted as *S*
- Blue – 8 bits for exponent, denoted as *E*, *and its width is* E_w
- Green - 23 bits for mantissa\significand as *M*, *and its width is* M_W

Single precision IEEE754 normal

3								2								1								0							
0	1	1	1	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	1	0	1

$$V = (-1)^{S_b} \times 2^{E_b - bias} \times \left(1 + \sum_{i=1}^{M_w} M_i \times \frac{1}{2^i}\right)$$

$$V = (-1)^{0_b} \times 2^{10000101_b - 127} \times \left(1 + \frac{1}{2^{23}} + \frac{1}{2^{21}} + \frac{1}{2^{17}} + \frac{1}{2^{16}}\right)$$

$$V = 1 \times 2^{133-127} \times 1.000233 = 64.015$$

Single precision IEEE754 normal

<table>
<tr><td colspan="8">3</td><td colspan="8">2</td><td colspan="8">1</td><td colspan="8">0</td></tr>
<tr><td>1</td><td>1</td><td>1</td><td>1</td><td>0</td><td>0</td><td>1</td><td>0</td><td>1</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>1</td><td>1</td><td>0</td><td>0</td><td>0</td><td>1</td><td>0</td><td>1</td></tr>
</table>

$$V = (-1)^{S_b} \times 2^{E_b - bias} \times (1 + \sum_{i=1}^{M_w} M_i \times \frac{1}{2^i})$$

$$V = (-1)^{1_b} \times 2^{10000101_b - 127} \times \left(1 + \frac{1}{2^{23}} + \frac{1}{2^{21}} + \frac{1}{2^{17}} + \frac{1}{2^{16}}\right)$$

$$V = -1 \times 2^{133-127} \times 1.000233 = -64.015$$

Single precision IEEE754 subnormal

3	2	1	0
0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0	1 1 0 0 0 1 0 1

$$V = (-1)^{S_b} \times 2^{1_b - bias} \times \left(0 + \sum_{i=1}^{M_w} M_i \times \frac{1}{2^i}\right)$$

$$V = (-1)^{0_b} \times 2^{1_b - 127} \times \left(0 + \frac{1}{2^{23}} + \frac{1}{2^{21}} + \frac{1}{2^{17}} + \frac{1}{2^{16}}\right)$$

$$V = 1 \times 2^{1-126} \times 0.000233 = 2.76056 \times 10^{-43}$$

Float vs Double

Parameter	Half	Single	Double	BFloat
Exponent Bit Count	5	8	11	8
Significand Bit Count	11	23	52	8
Sign Bit Count	1	1	1	1
Total Bit Count	16	32	64	16
Bias	15	127	1023	127
Epsilon	$\sim 10^{-4}$	$\sim 10^{-7}$	$\sim 10^{-16}$	$\sim 10^{-5}$

Special values (Zeroes)

<table>
<tr><td colspan="8">3</td><td colspan="8">2</td><td colspan="8">1</td><td colspan="8">0</td></tr>
<tr><td>S</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td></tr>
</table>

- There are two types of zero: positive and negative.
- Positive is denoted by all bits set to 0
- Negative is denoted by all bits except the signed one set to 0

Special values (Infinities)

3								2								1								0							
S	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

- There are two types of infinity: positive and negative.
- Positive is denoted by all exponent bit set to 1, sign bit set to 0, and all mantissa bits set to 0.
- Negative is denoted by all exponent bit set to 1, sign bit set to 1, and all mantissa bits set to 0.

Special values (NANs)

- There are a lot of different NaNs.
- NaNs are results of some undefined operations like dividing of 0 by 0, or Infinity by Infinity.
- They denoted by all exponent bit set to 1, sign bit set to 0, and at least one mantissa bit is set to 1.

Problems

- Compiler optimize away special values checks
- Compiler cannot optimize because of special values
- Rounding errors
- Precision lost errors
- Comparison with numbers in general
- Comparison with numbers of different magnitude
- Conversion from integers to floating points
- Undefined operations
- Serialization of values