Floating-Point Numbers

- Marc Andrysco

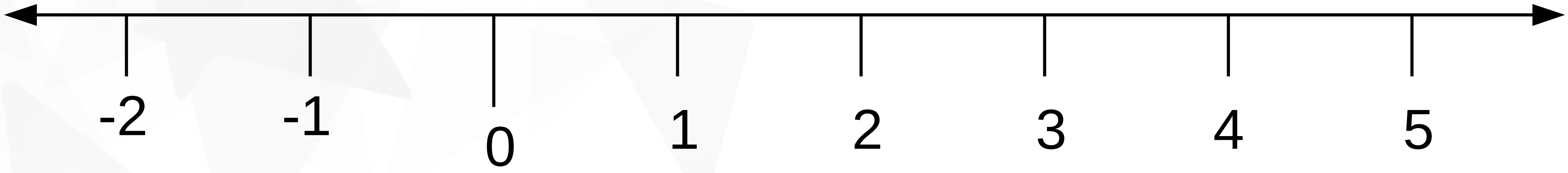
Overview

- Formats
  - Scientific-notation
  - IEEE-754 representation
- Rounding
  - Boundaries
  - Rounding Error
- Practical Considerations

Floating-Point Format

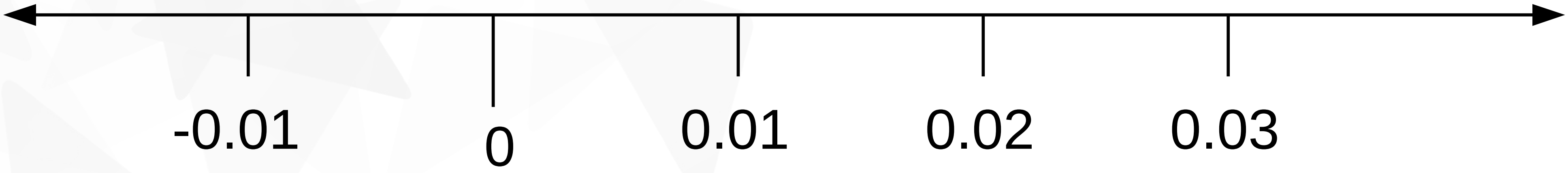
- Integers
- Fixed-point numbers
- Floating-point numbers
- Range and Precision

Integers



- Take up whole number spots on the number line
- Space between is always 1
- 32-bit integers have 2^{32} unique points
  - Signed range [-2147483648, 2147483647]
- The number line can wrap around the globe

Fixed-Point Numbers



- Integers scaled by a constant factor
- Space between is constant
- Useful for fixed decimal values like currency
- 32-bit numbers have 2^{32} unique points
  - Example range [-21474836.48, 21474836.47]

Scientific Notation

- Contains a “floating” decimal point
  - 1.2×10^{-2} same as 0.012×10^{0}
- Easy to represent large and small values
  - 1.6162×10^{-35}, 6.02×10^{23}
- Made up of two pieces

    $1.2\ 10^{-2}$

  - Significand (or fraction or mantissa)
  - Exponent

Normal Form

0.01616×10^{-33} 1.616×10^{-35} 1616.0×10^{-38}

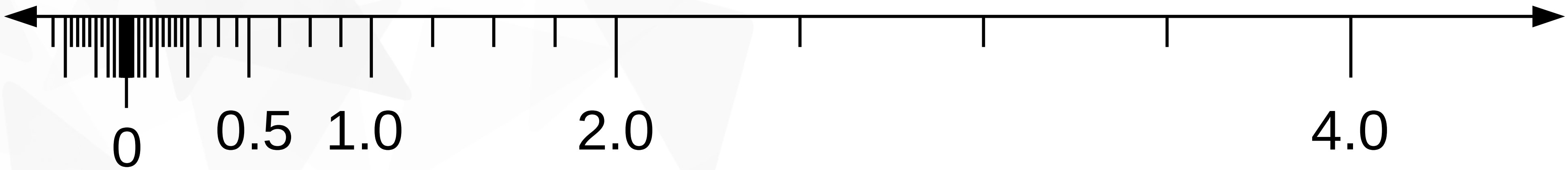
- Non-zero, single digit left of the decimal
  - Called normal or normalized
- Non-conforming forms are “denomalized”
  - All denormalized numbers can be made normal ✢

✢ Except zero, infinity...

Fixed Precision

- Fix the number of of digits
  - Always use normal form
- Ex: 3 significand digits, 2 exponent digits
  - Maximum number 9.99×10^{99}
  - Minimum (non-zero) 1.00×10^{-99}
- Missing zero!
  - It has no normal form

Floating-Point Range



- Space between floats vary
- Allows for an enormous range
  - Max value of 9.99×10^{99}
    - 2.3×10^{73} larger than the observable universe
    - Next smaller value, 9.98×10^{99} is a difference of 10^{97}
  - Min value of 1.00×10^{-99}
    - 1.6×10^{64} times smaller than Planck length

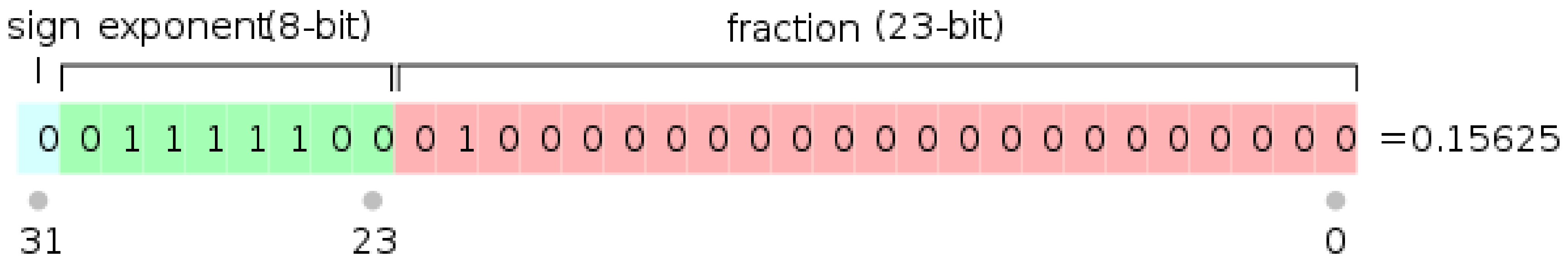
IEEE-754 Floats

- IEEE-754 standard
- Special values
- Subnormal numbers

IEEE-754 Standard

- IEEE-754 specifies standard formats for floats
  - Binary32: “single-precision” ; `float` in C
  - Binary64: “double-precision” ; `double` in C
- Like scientific-notation, uses binary
  - $1.0101110001_b \times 2^{101_b}$
  - Normal forms, first digit must be 1

Single-Precision Float

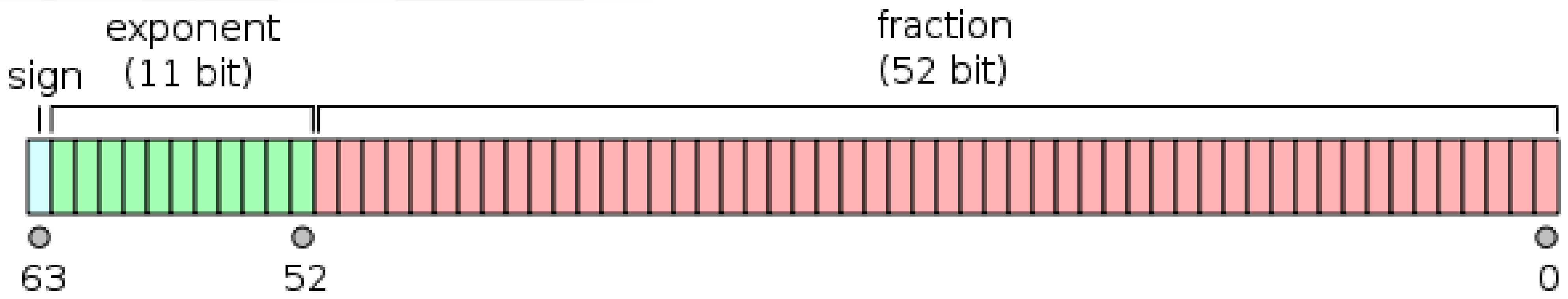


- 1-bit sign, 8-bit exponent, 23-bit fraction ✢
- Maximum: $3.402823466 \times 10^{38}$
- Minimum: $1.175494351 \times 10^{-38}$ ❈

✢ Special forms for other numbers (zero, infinity...)

❈ That's sometimes lie

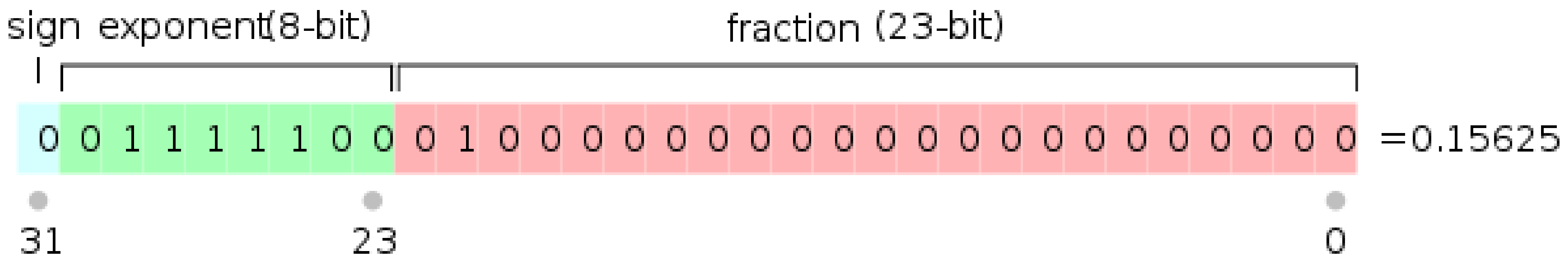
Double-Precision Float



- 1-bit sign, 11-bit exponent, 52-bit fraction ⸸
- Maximum: $1.7976931348623157 \times 10^{308}$
- Minimum: $2.2250738585072014 \times 10^{-308}$

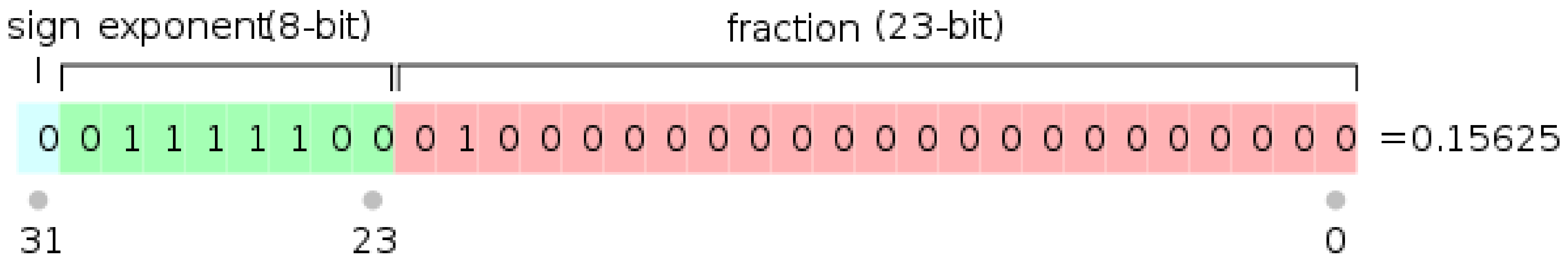
⸸ Same exceptions apply

Float Representation



- Fraction starts with 1 (ex: 1.0100...)
- Exponent is biased by 127
  - Example as above: 01111100_b – 127 = 124 – 127 = -3
  - Makes floats better ordered

Special Values



- Zero – all exponent and fraction bits are zero
  - Includes negative zero: -0.0
- Infinity – all exponent and fraction bits are one
  - Includes negative infinity: -Inf
- NaN – exponent is all ones, fraction is non-zero
  - Have many NaN representations including -NaN

Subnormal Numbers

- Occurs when all exponent bits are zero
  - No longer given that the first fraction bit is one
  - Takes the form $0.000110_b \times 2^{-126}$
- Loss of precision
- Extreme loss of performance
- Frequently omitted (e.g. GPUs)

Subnormal Rationale

- Single-precision floats

  $1.175\times10^{-38} \rightarrow 1.401\times10^{-45}$

- Double-precision floats

  $2.225\times10^{-308} \rightarrow 4.940\times10^{-324}$

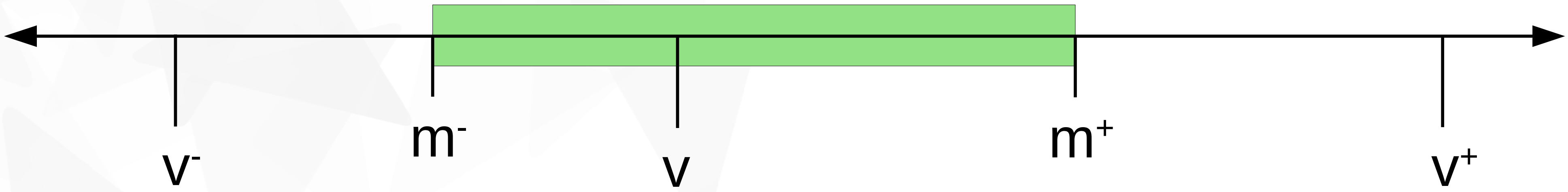
- Subtraction property

```
if a != b
  y = 1 / (a - b)
```

Rounding

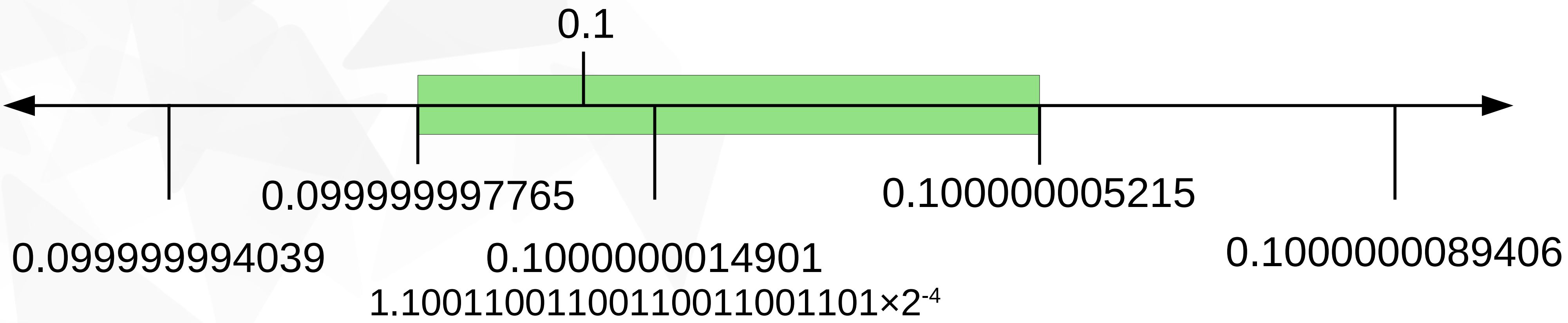
- Boundaries
- Rounding Modes
- Printing and Scanning

Boundaries



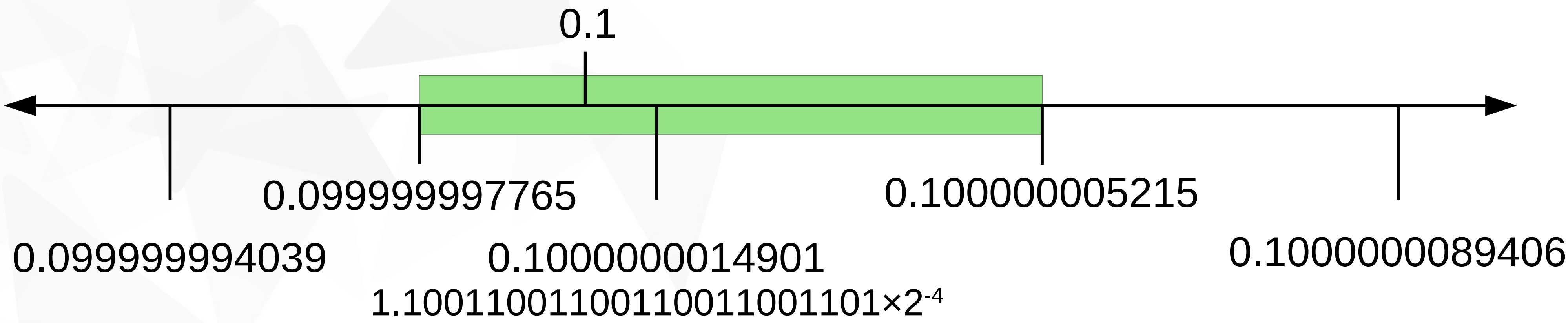
- Predecessor and successor (v^-, v^+)
- Surrounded by midpoints (m^-, m^+)
  - Also called boundaries
- Between midpoints map to v

Rounding



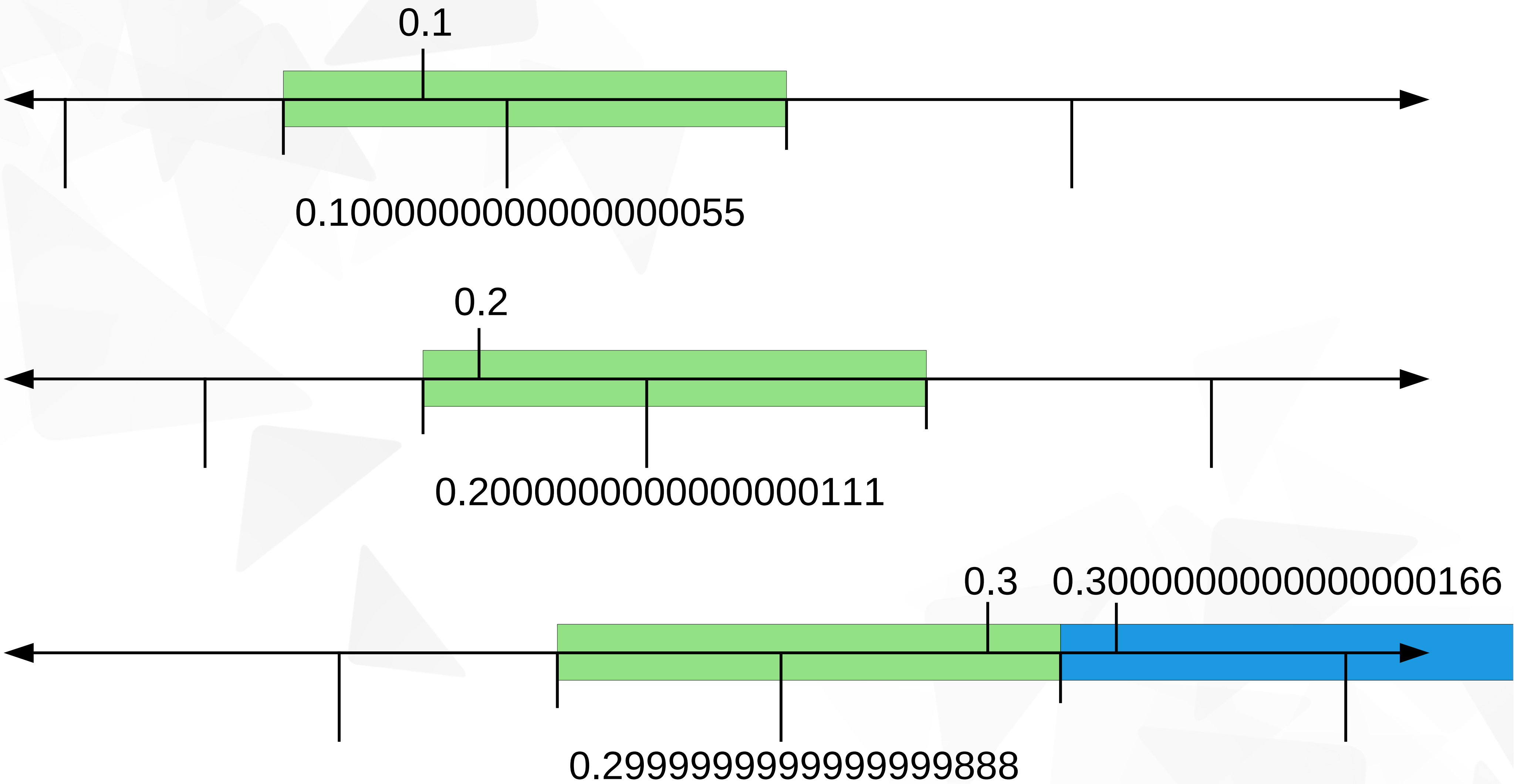
- Poor mapping between real → floats
- 0.1 maps to nearest float
  - $1.10011001100110011001101 \times 2^{-4}$
- Boundaries round to even

Printing Floats



- Which real number to print out
  - Select any value between boundaries
- Pick shortest

Rounding Error (0.1 + 0.2)



Rounding Error

- Finite number of digits
  - Cannot sum big with small

$1.23\times10^4 + 4.56\times10^0$

$= 1.230456\times10^4$

Truncate to fit format

$\approx 1.23\times10^4$

Primarily affects add/sub

```
  123  .
+    4.56
  --------
  12304.56
  --------
  123  .
```

Reduce Rounding Error

- Sum similar numbers first
- Harmonic series 1/1 + 1/2 + 1/3 + 1/4 …
- Sum the first N terms
  - Add small terms first
- At 10,000,000 iterations
  - Forward 15.4037
  - Backward 16.6860
  - Real 16.6953

Practical Considerations

- Catastrophic Cancellation
- Alternate Formats
- Using `printf`

Catastrophic Cancellation

- `x^2 - y^2` can give very poor results

  `x=900.2`, `y=900.1` (4 digit format)

- `x^2 = 810360.04 → 8.103e5`
- `y^2 = 810180.01 → 8.106e5`

  `x^2 - y^2 = 1.000e2`

- Actual answer is 180.03
- We expected to get `1.800e2`

Fixing Cancellation

- Rewrite `x^2 - y^2`

  as `(x - y)(x + y)`

  `900.2 - 900.1 = 0.1`

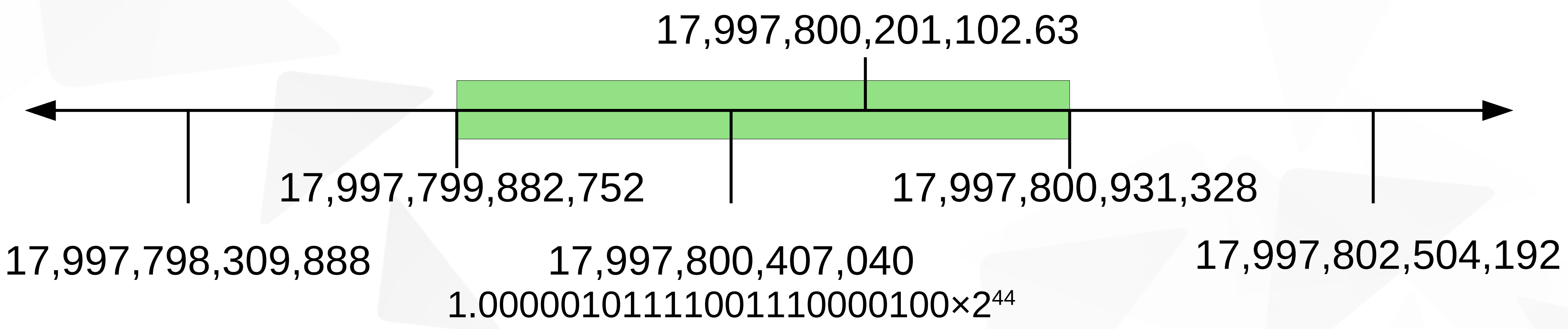
  `900.2 + 900.1 = 1800.1 → 1.800e3`

  `0.1 × 1.800e2 = 1.800e2`

- Exactly the correct answer

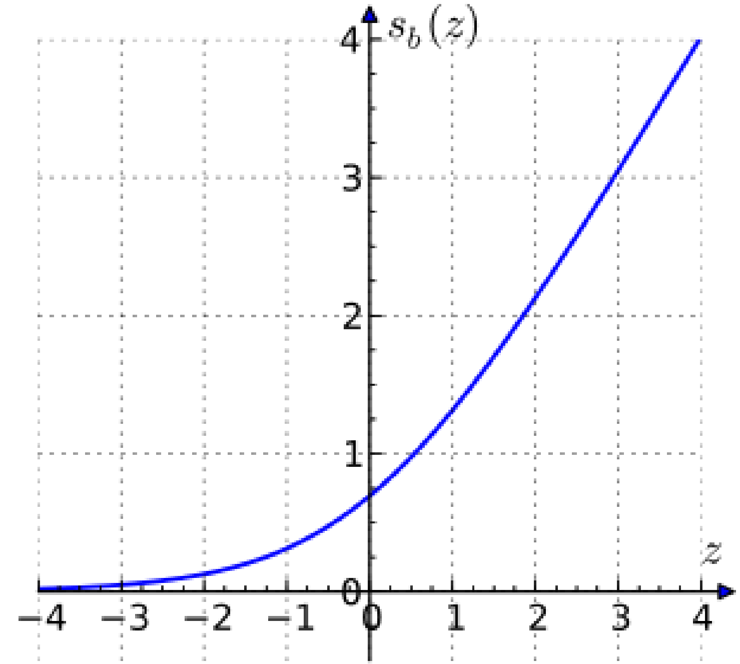
Alternate Formats

- Fixed-point
  - Exact spacing between values
  - Good for currency or time



Logarithmic Number System

- Fixed base, variable exponent
  - b^n
- Used in music (cent)
  - $b=2^{1/12000}$
- An octave
  - double the frequency
  - 12000 cents

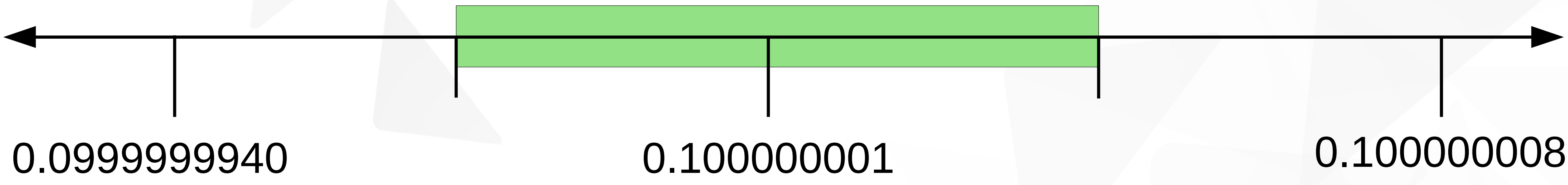


Printing Floats

- `%f` – Fixed number after decimal
- `%e` – Always use exponential notation
- `%g` – Chooses between `%f` and `%e`
- Use `%g` for debugging
- Use `%e` for data logging

Printing Floats

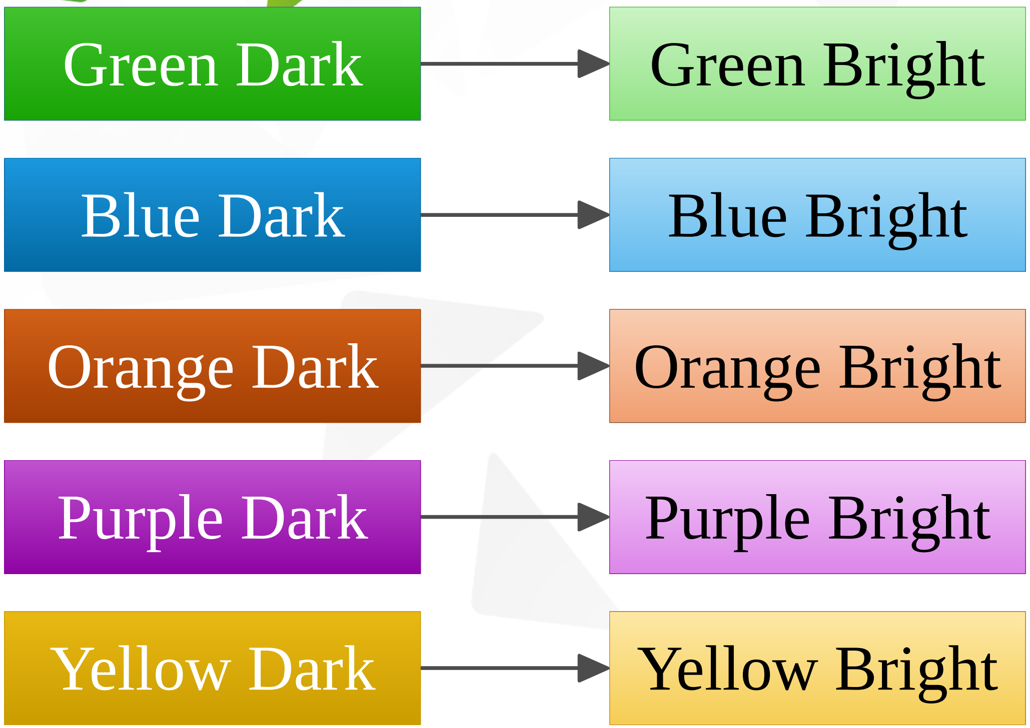
- `%.17e` – Print 17 digits
  - 17 digits for double
  - 9 digits for single
- `%#g` – Always print decimal point
- `%a` – Hexadecimal float



Thanks

Pre-defined Shapes

Here are some pre-defined shapes for your convenience: copy the shapes, copy their formatting, or use the LibO styles.




```
You may add your
code examples, XML
statements, or debug
output here ;-)
```

Section Header Example

You may add additional text here ...

Thank you …

- … for using this template!
- … for supporting LibreOffice!

Normalized Math

- Multiplication is easy
  - $y \times 2^z = a \times 2^b \times c \times 2^d$
  - $y = a \times c$
  - $z = b + d$

$3.1 \times 2^4 \times 2.6 \times 2^{-2}$

$3.1 \times 2.6 = 8.06$

$4 + -2 = 4$

8.06×2^2

Normalized Math

- Addition is harder
  - $y\times2^z = a\times2^b + c\times2^d$ — $3.1\times2^4 + 2.6\times2^3$
  - $y\times2^z = a\times2^b + c'\times2^b$ — $3.1\times2^4 + 0.26\times2^4$
  - $y = a + c'$ — $3.1 + 0.26 = 3.36$
  - $z = b$ — $4 = 4$

3.36×2^4

Denormalized Math

- All operations are harder
  - Results may be outside representable range
  - Multiplication yields wildly denormalized results
  - Much slower on modern CPUs (Intel, AMD)
- Frequently omitted
  - GPUs round denormals to zero