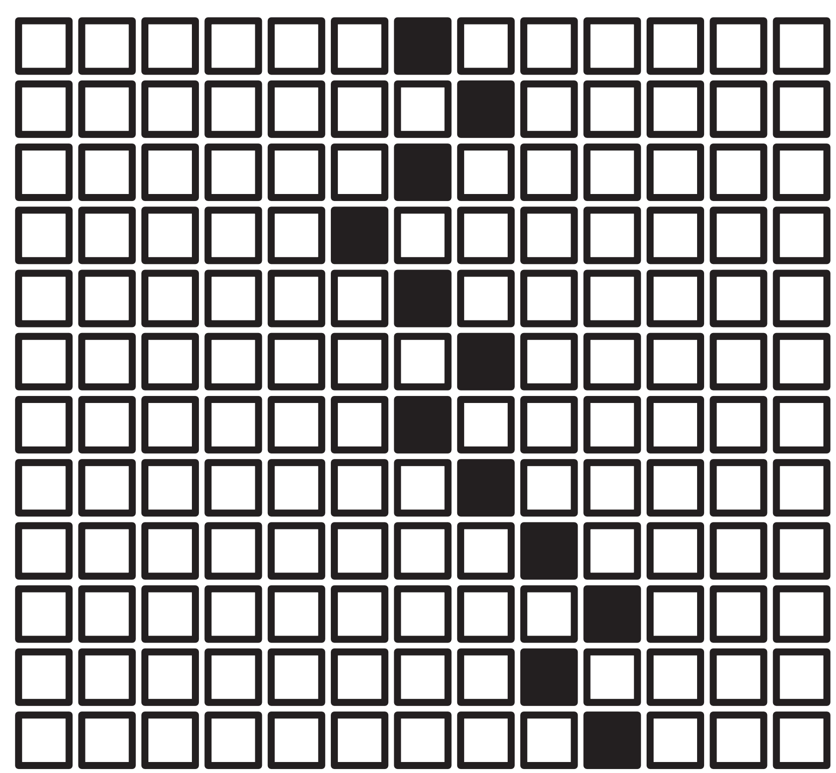
Floating-point numbers

Phys 750 Lecture 5

Random walk CA

- Activate a single cell at site $i = 0$
- For all subsequent times steps, let the active site wander to $i := i \pm 1$ with equal probability



Random walk CA

- Q: If we run this model M times, how often is the activated cell found at position i after 25, 100 , 400, 1600 , 4800 steps?
- Empirical test: let's allocate storage for a histogram:

```
unsigned long int hist25[51];
unsigned long int hist100[201];
unsigned long int hist400[801];
unsigned long int hist1600[3201];
unsigned long int hist4800[9601];
```

N steps

2N+1 elements

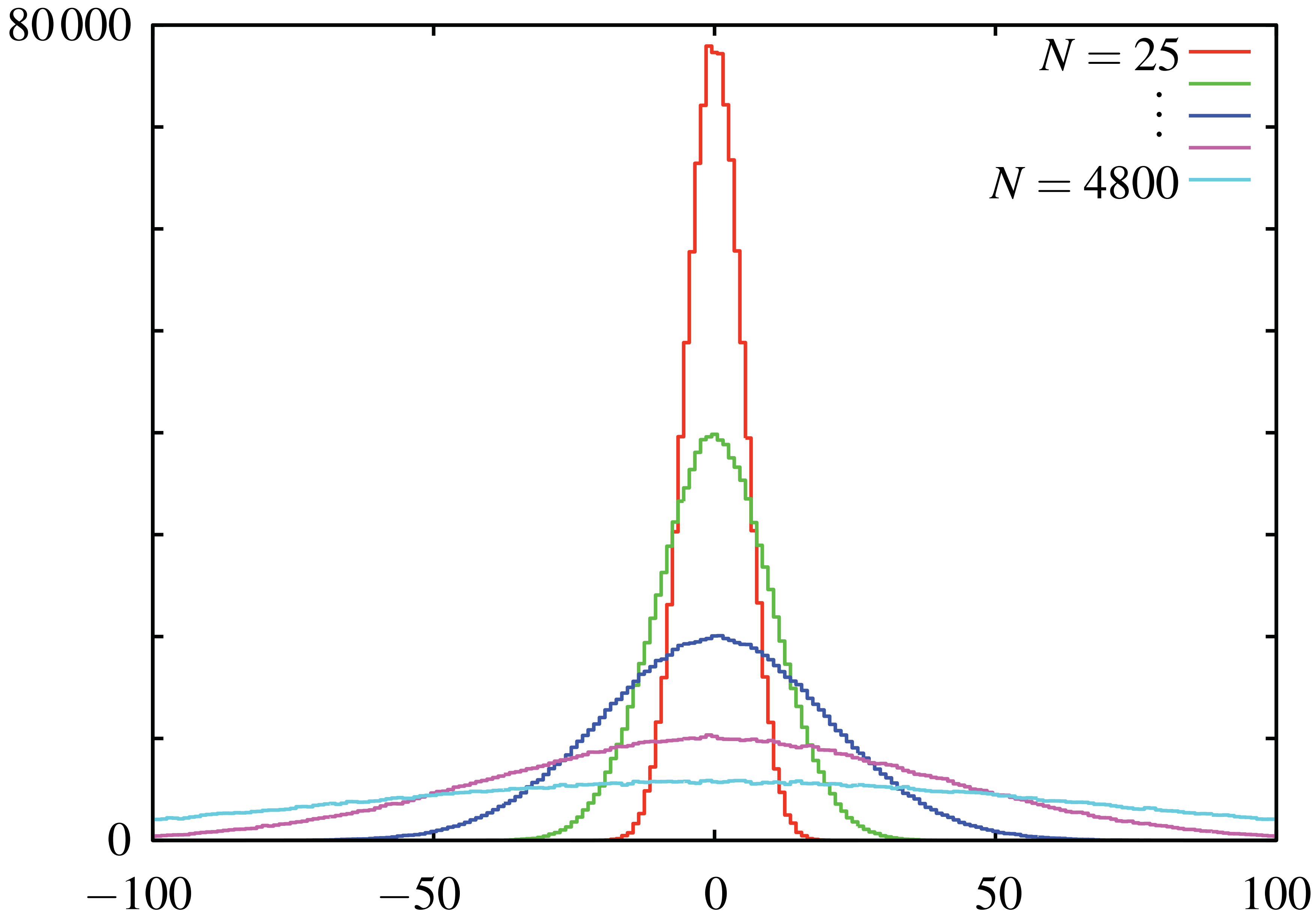
Random walk CA

- Then accumulate values in the arrays:

```
int main()
{
   for (unsigned long int m = 0; m < M; ++m)
   {
      int x = 0;
      for (int n = 0; n <= 4800; ++n)
        {
            if (R() < 0.5) ++x; else --x;
            if (n == 24 or n == 25) ++hist25[x+25];
            else if (n == 99 or n == 100) ++hist100[x+100];
            else if (n == 399 or n == 400) ++hist400[x+400];
            else if (n == 1599 or n == 1600) ++hist1600[x+1600];
            else if (n == 4799 or n == 4800) ++hist4800[x+4800];
        }
    }
```

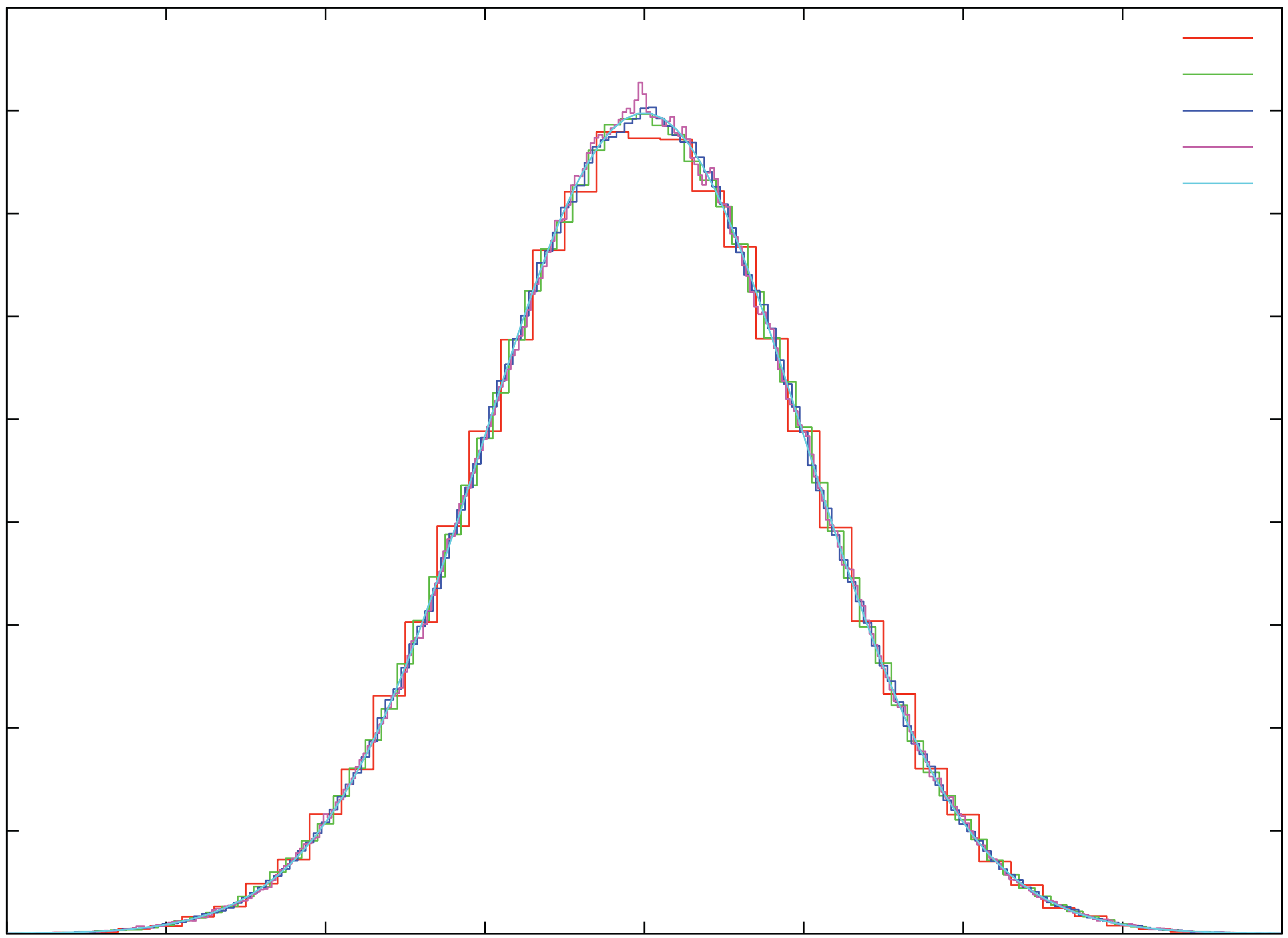
watch the offset!

Position histograms



- Results for $M = 500\,000$

Asymptotic distribution



$\times N$

$\times \frac{1}{N}$

after rescaling, collapses onto

$$2M \times \frac{e^{-x^2/2\sigma}}{\sqrt{\pi\sigma}}$$

Asymptotic distribution

- In the double limit $N, M \to \infty$, the rescaled histogram is a perfect gaussian (normal distribution)
- Amazingly, a smooth, continuous distribution can result from a limiting sequence of discrete histograms
- Analogue of coarse-graining
- Rescaling implicitly turns integers into fractions; suggests that we can use rational numbers to cover the real line

Floating-point numbers

- Floating-point numbers have the form

sign

$$\pm f \times b^e$$

exponent

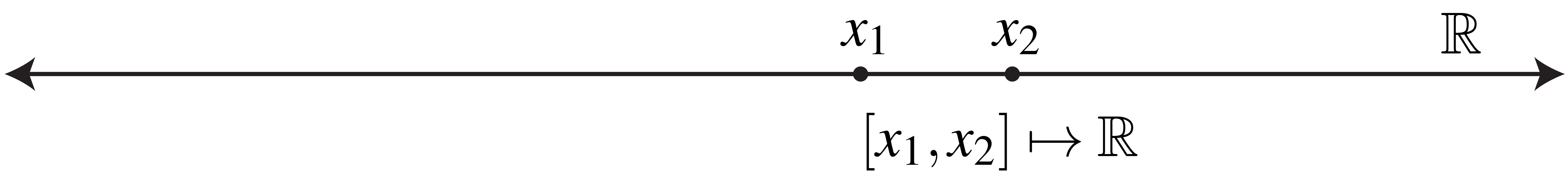
fraction (or significand)

base

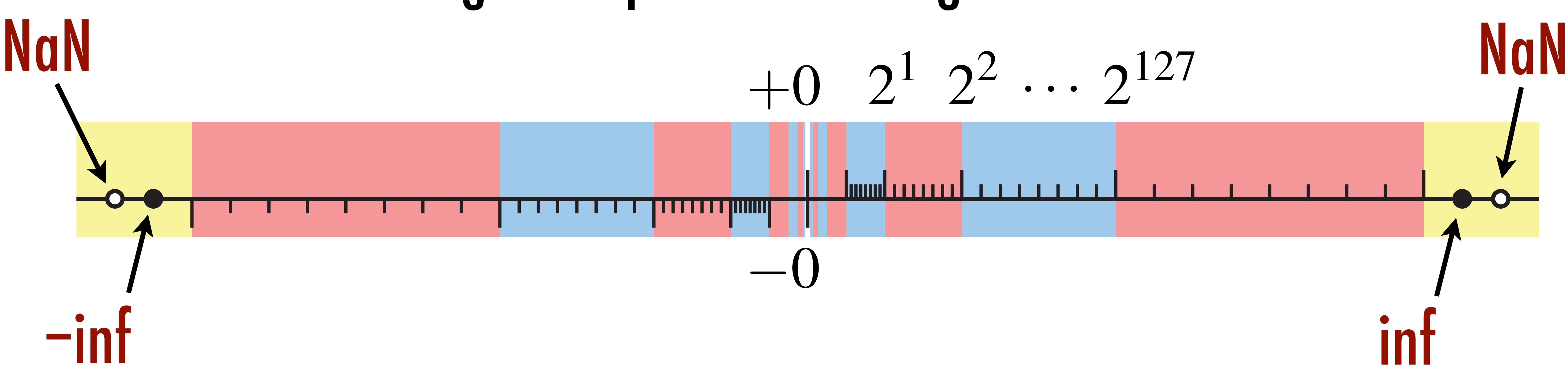
- The adjustable radix point allows for calculation over a wide range of magnitudes
- Floating-point numbers are limited by the number of bits used to represent the fraction and exponent

Floating-point numbers

- Real line is dense and uncountably infinite:



- FP scheme gives a partial covering:

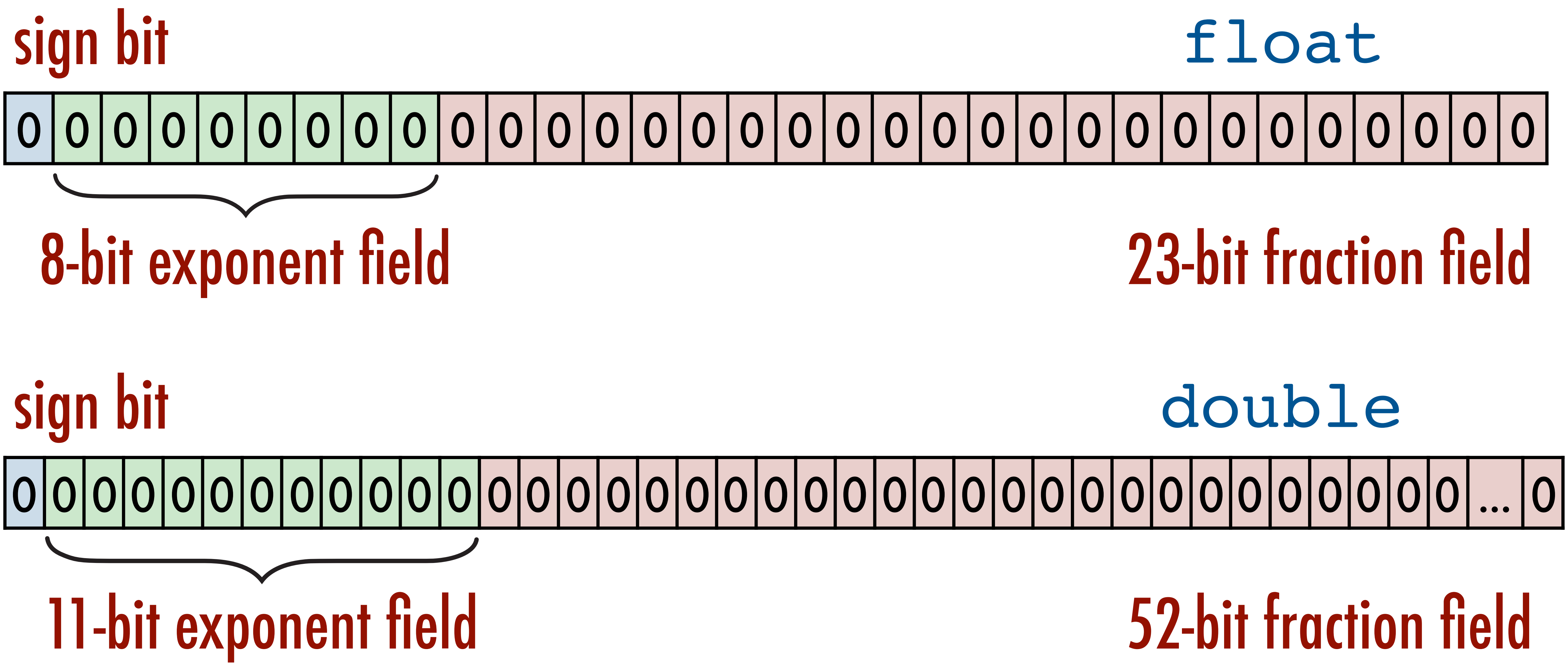


Floating-point numbers

- Finite representation that manages to span many orders of magnitude
- A sort of finite-precision scientific notation, with the significant and exponent encoded in fixed width binary
- Equal number of uniformly spaces values in each interval $[2^n, 2^{n+1})$
- Relies on special values (+0, −0, inf, −inf, NaN)

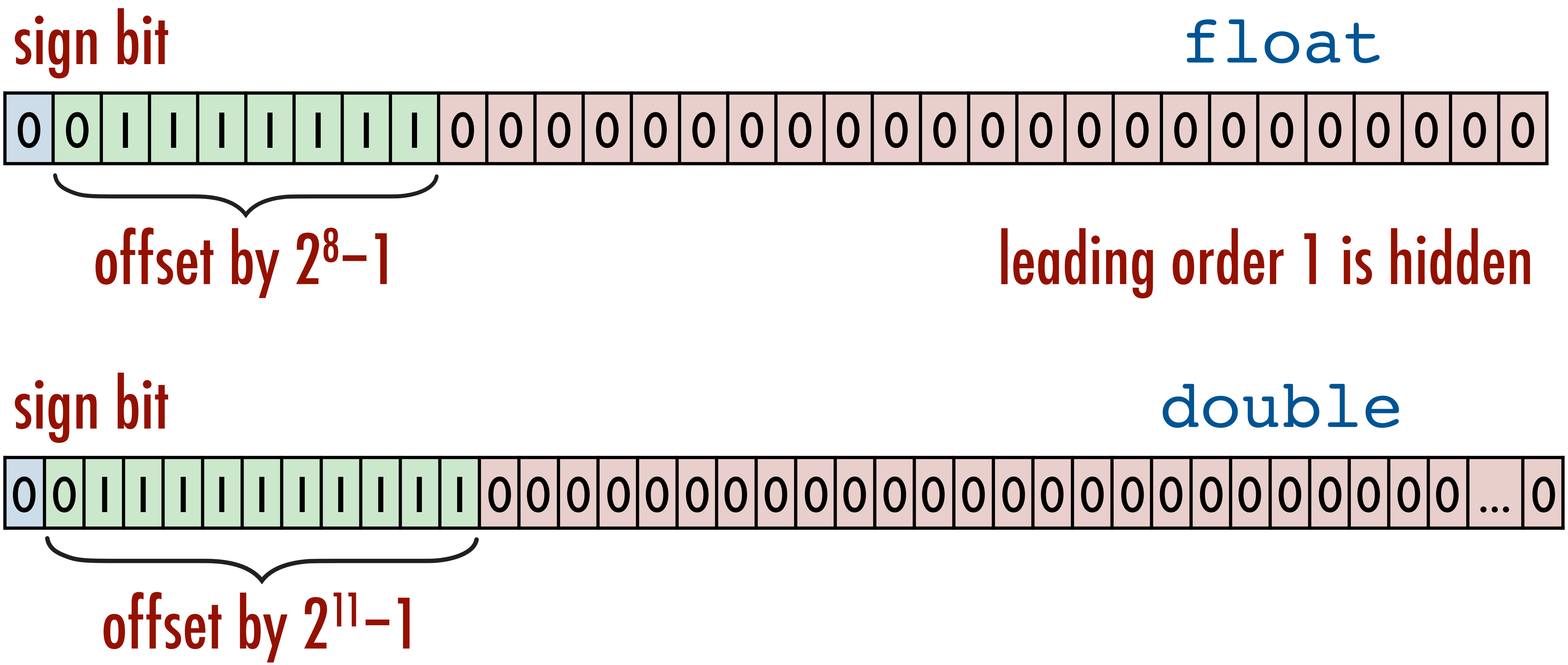
Floating point types

- Intel architecture follows the IEEE 754 standard



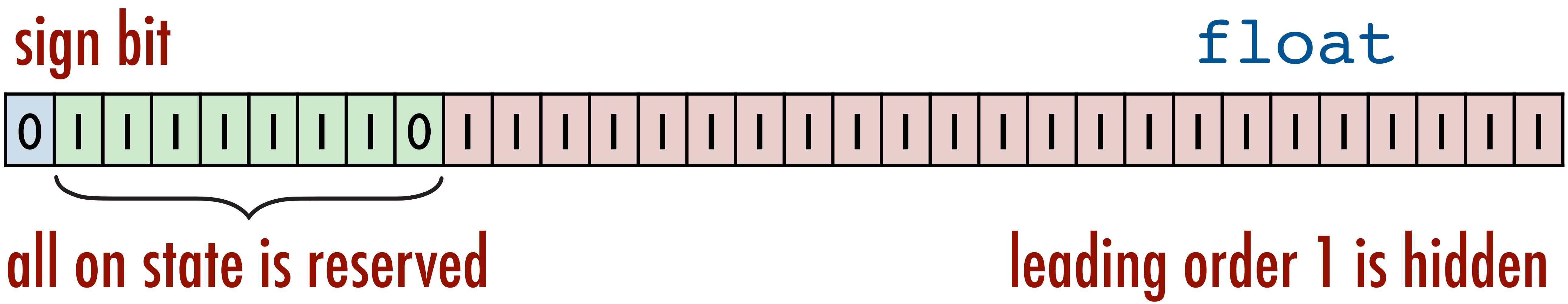
Floating point types

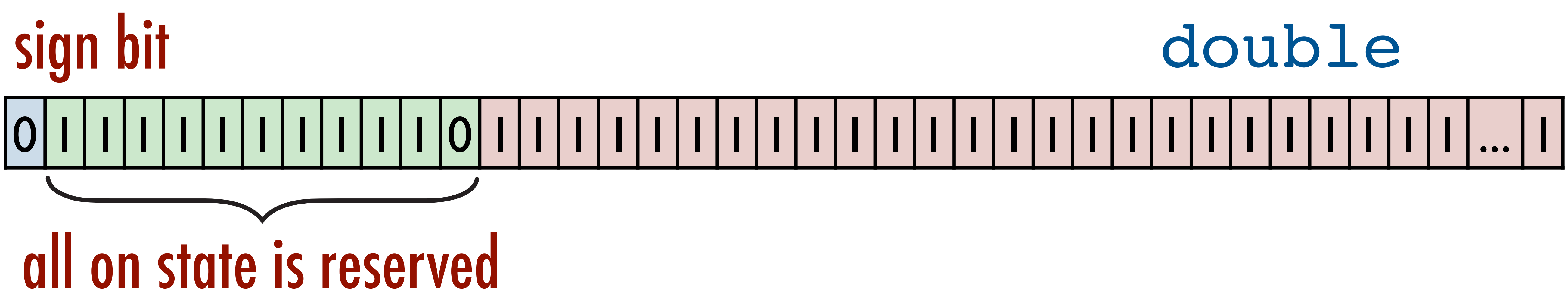
- Representation of unity:



Floating point types

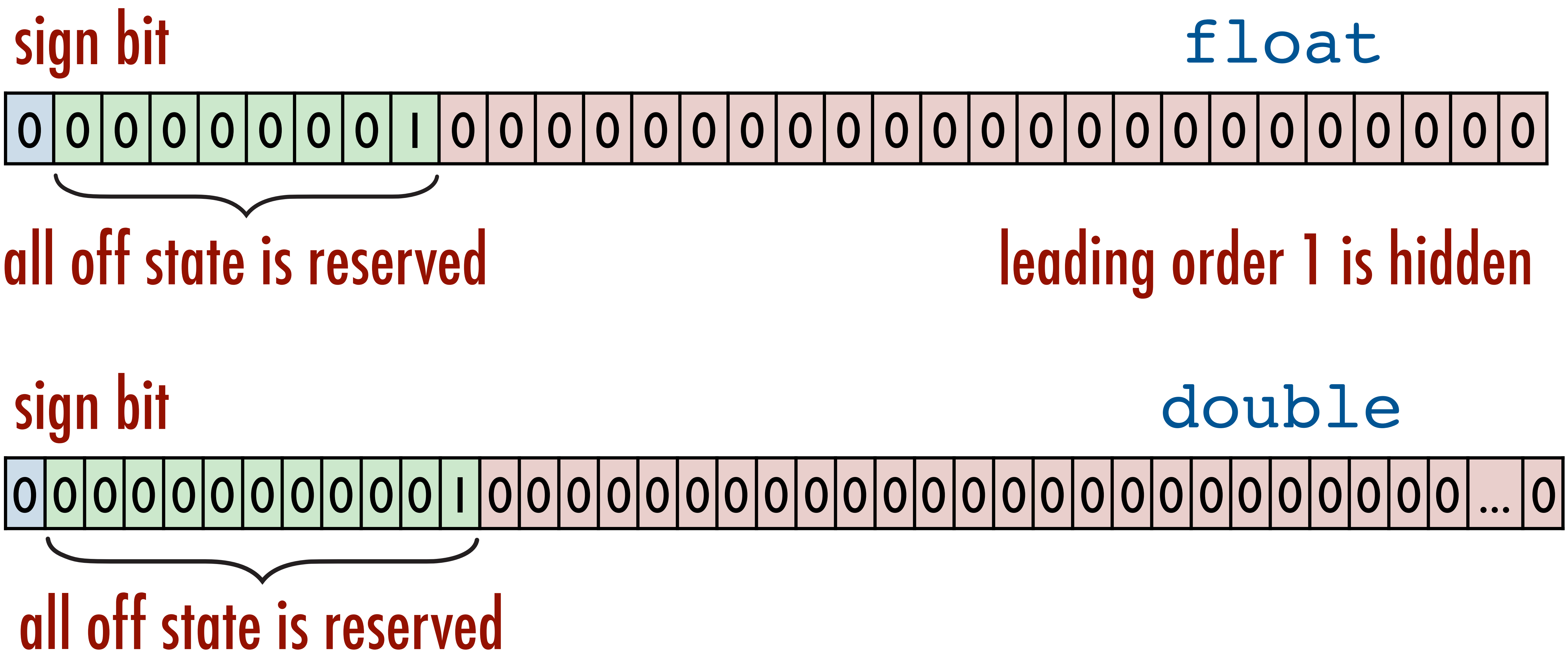
- Largest positive number:





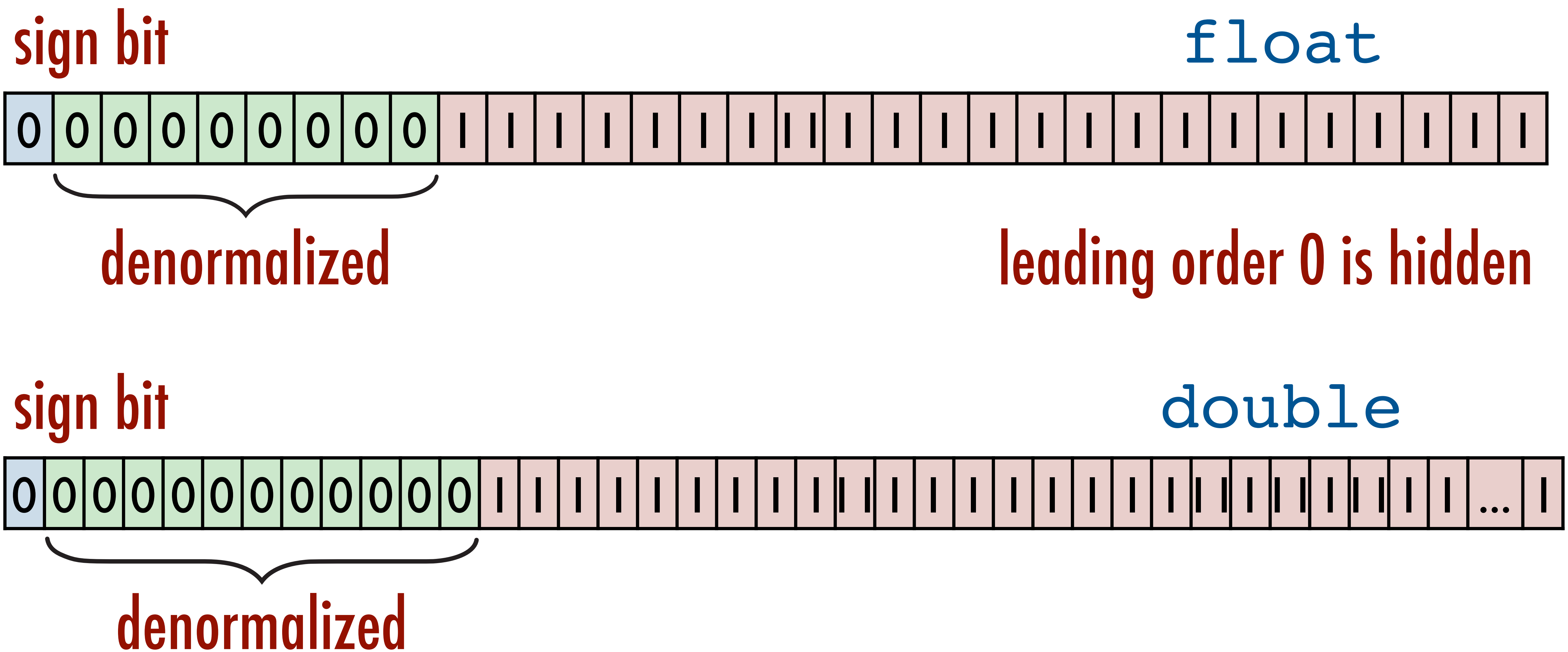
Floating point types

- Smallest positive non-denormalized number:



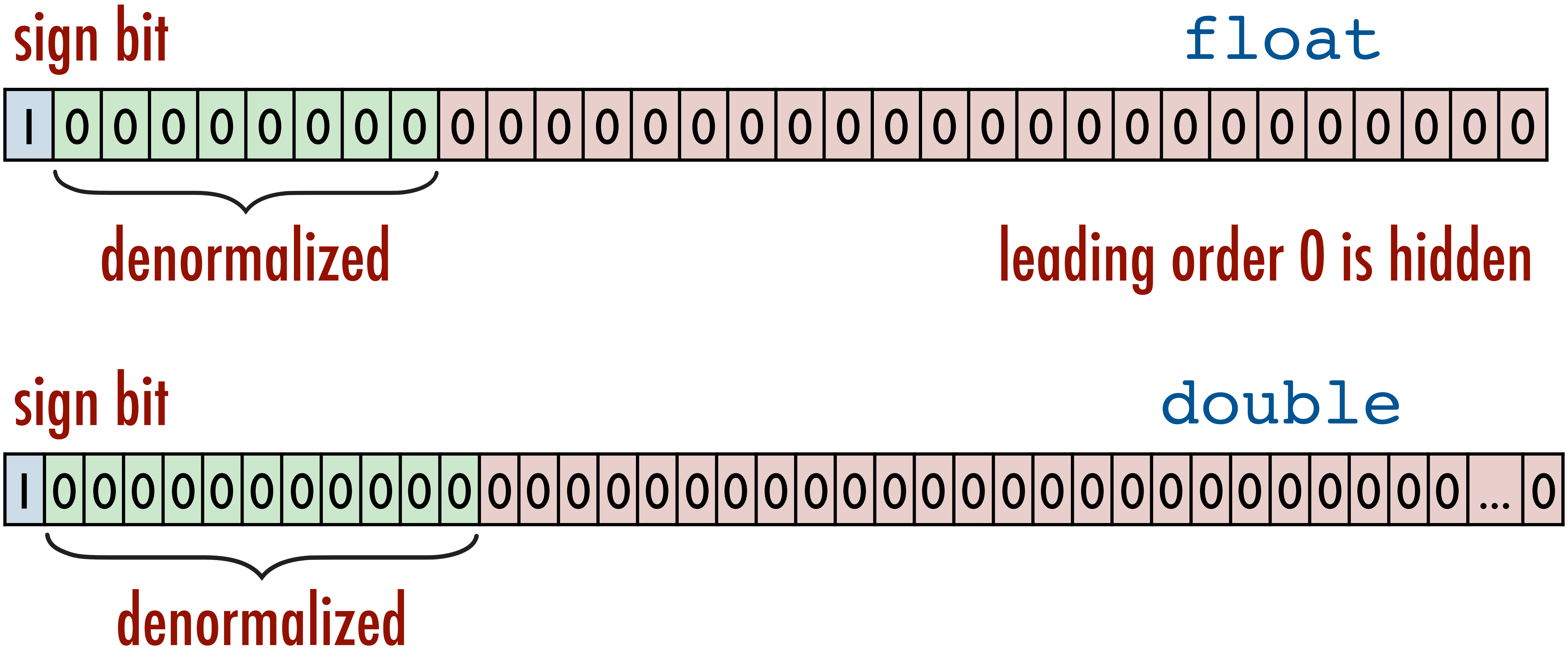
Floating point types

- Largest positive denormalized number:



Floating point types

- Negative zero:



Accuracy of FP arithmetic

- FP arithmetic is by its nature inexact
- Important always to think about accuracy: should we believe the computer's final answer?
- FP multiplication is relatively safe
- FP subtraction of nearly-equal quantities (or addition of equal magnitude, opposite sign quantities) can dramatically increase the relative error

Potential dangers

- FP operations can yield both “overflow” and “underflow”
- Additional notes on the class web site will explore the Infinity (Inf) and Not-a-Number (NaN) error states

Potential dangers

- Associativity breaks down: $(u+v)+w \neq u+(v+w)$
- The following 8-digit decimal floating point operation has a 5% relative error depending on the order in which operations are performed:

$$
\begin{aligned}
(11111113. + -11111111.) + 7.5111111 &= 2.0000000 + 7.511111 \\
&= 9.511111
\end{aligned}
$$

$$
\begin{aligned}
11111113. + (-11111111. + 7.511111) &= 11111113. + -11111103. \\
&= 10.000000
\end{aligned}
$$

Potential dangers

- The distributive law

$$u \times (v + w) \neq (u \times v) + (u \times w)$$

can also fail badly:

$$\begin{aligned} 20000.000 \times (-6.0000000 + 6.0000003) &= 20000.000 \times 0.00000030000000 \\ &= 0.0060000000 \end{aligned}$$

$$\begin{aligned} (20000.000 \times -6.0000000) + (20000.000 \times 6.0000003) &= -120000.00 + 120000.01 \\ &= .01000000 \end{aligned}$$

Potential dangers

- It can easily occur that $2(u^2+v^2) < (u+v)^2$
- Hence, variance is not guaranteed to be positive
- Naively calculating the standard deviation can lead to your taking the square root of a negative number

$$\sigma = \frac{1}{n}\sqrt{n\sum_{k=1}^{n} x_k^2 - \left(\sum_{k=1}^{n} x_k\right)^2}$$

Potential dangers

- Many common mathematical relations no longer hold ...

$$(x+y)(x-y) = x^2 - y^2$$

$$\sin^2\theta + \cos^2\theta = 1$$

“Carefully written programs”

- technical meaning: programs that are numerically correct
- this is very difficult to guarantee!

“Carefully written programs”

1. $(x+y)/2$
2. $x/2+y/2$
3. $x+((y-x)/2)$
4. $y+((x-y)/2)$

- Which formula should we use to compute the average of x and y?

"Carefully written programs"

1. $(x+y)/2$
2. $x/2+y/2$
3. $x+((y-x)/2)$
4. $y+((x-y)/2)$

May raise an overflow if x and y have the same sign

“Carefully written programs”

1. $(x+y)/2$
2. $x/2+y/2$
3. $x+((y-x)/2)$
4. $y+((x-y)/2)$

May degrade accuracy but is safe from overflows

“Carefully written programs”

1. $(x+y)/2$
2. $x/2+y/2$
3. $x+((y-x)/2)$
4. $y+((x-y)/2)$

May raise an overflow if x and y have opposite signs

“Carefully written programs”

- you want functions that are robust
- give some thought to the rare or extreme cases that may cause your function to misbehave
- avoid overflows and underflows
- avoid undefined operations, e.g., $\sqrt{-1},\ \frac{0}{0}$

"Carefully written programs"

- Example: roots of the quadratic equation, $ax^2 + bx + c$
- According to the usual formula, $x_{1,2} = \dfrac{-b \pm \sqrt{b^2 - 4ac}}{2a}$
- Problem can arise if $b^2 \gg |4ac|$ so that $\sqrt{b^2 - 4ac} \approx |b|$
- Cancellation can lead to catastrophic loss of significant digits: $\dfrac{b \pm |b|}{2a} \approx \dfrac{0}{0}$

"Carefully written programs"

- One possible workaround: use exact algebraic manipulations on a per case basis

$$x_1 = \frac{-b + \sqrt{b^2 - 4ac}}{2a} = \frac{-2c}{b + \sqrt{b^2 - 4ac}}$$

$$x_2 = \frac{-b - \sqrt{b^2 - 4ac}}{2a} = \frac{2c}{-b + \sqrt{b^2 - 4ac}}$$

cancellation

no cancellation

“Carefully written programs”

```
#include <cassert>
#include <cmath>
using std::sqrt; // square root
using std::fabs; // absolute value

void quadratic_roots(double a, double b, double c,
                     double &x1, double &x2)
{
   const double X2 = b*b-4*a*c;
   assert(X2 >= 0.0);
   const double X = sqrt(X2);
   const double Ym = -b-X;
   const double Yp = -b+X;
   const double Y = (fabs(Ym) > fabs(Yp) ? Ym : Yp);

   x1 = 2*c/Y;
   x2 = Y/(2*a);
}
```

“Carefully written programs”

- Example: norm of a complex number

$$z = x + iy,\ |z| = \sqrt{x^2 + y^2}$$

- Avoid possible overflow when squaring terms:

$$|z| = x\sqrt{1 + r^2},\ r = \frac{y}{x},\ \text{ if } |y| < |x|$$
$$|z| = y\sqrt{1 + r^2},\ r = \frac{x}{y},\ \text{ if } |x| < |y|$$

"Carefully written programs"

- Evaluation by nested polynomials (Horner's scheme)

$$
\begin{aligned}
f(x) &= a_0 + a_1 x + a_2 x^2 + a_3 x^3 + \cdots a_N x^N \\
&= a_0 + x(a_1 + x(a_2 + x(a_3 + \cdots x(a_{N-1} + x a_N)\cdots)))
\end{aligned}
$$

```c
double eval_poly(const double f[], double x, int n)
{
   double val = f[--n];
   do
   {
      val *= x;
      val += f[--n];
   } while (n != 0);
   return val;
}
```

“Carefully written programs”

```
#include <iostream>
using std::cout;
using std::endl;

int main()
{
   // f(x) = 1 + 20x + 9x^2 - 3x^3
   //           + 5x^4 + 2x^5 + x^6
   double f[7] = { 1, 20, 9, -3, 5, 2, 1 };
   for (int i = 0; i <= 1000; ++i)
   {
      const double x = (i-500)*3.0/500;
      cout << x << "\t" << eval_poly(f,x,7) << endl;
   }
   return 0;
}
```

"Carefully written programs"

- Example: the sinc function $\frac{\sin(x)}{x}$
- possible problems as $x \to 0$
- workaround: explicit power series expansion

$$\frac{\sin(x)}{x} = \frac{x - \frac{x^3}{3!} + \frac{x^5}{5!} + \cdots}{x} = 1 - \frac{x^2}{3!} + \frac{x^4}{5!} + \cdots$$

with sensible cutoff

Arbitrary precision arithmetic

- scheme for performing operations on integers and rational numbers with no rounding, e.g.,

$$\frac{2153}{9932} + \frac{871}{7362} = \frac{12250579}{36559692}$$

- available in symbolic manipulation environments (Maple, Mathematica) and “bignum” libraries
- implemented in software; limited by system memory