Lecture 1

Real Numbers on a Finite Machine

Virginia Tech Department of Computer Science
Fall 2026 • Sumeet Khatri

Outline

01	**Describe a floating-point number**	Identify sign, base, significand, precision, and exponent.
02	**Read IEEE 754 formats**	Understand the binary16, binary32, binary64, and binary128 formats.
03	**Reason about finite-precision effects**	Explain gaps, overflow, underflow, rounding, and special values.
04	**Quantify approximation error**	Use absolute error, relative error, unit roundoff, and machine epsilon.

Exact arithmetic meets a finite machine

Exactly representable

$2 + 2 = 4$

$4 \times 8 = 32$

Usually approximate

$(\sqrt{3})^2 \approx 3$

not necessarily exactly 3

Why?

A computer stores only a finite set of finite-digit numbers. Most real numbers must be represented by a nearby machine number.

$\sqrt{3^2} - 3 \approx -4.44 \times 10^{-16}$ (binary64 experiment)

Machine arithmetic models real arithmetic

Real-number world

Infinitely many numbers

Potentially infinite digit strings

Exact definitions such as $(\sqrt{3})^2 = 3$

Machine world

A finite set of representable values

Fixed precision and exponent range

Approximate real-number calculations

round-off error = error caused by finite-digit arithmetic

Scientific notation

$$37.21829 = 3.721829 \times 10^{1}$$

$$0.002271828 = 2.271828 \times 10^{-3}$$

$$3000527.11059 = 3.00052711059 \times 10^{6}$$

General p-digit decimal form

$$(-1)^{s}(d_1.d_2d_3\ldots d_p)_{10} \times 10^{k}$$

Binary uses place value too!

Binary digits Only 0 and 1 are allowed.

$$(0110)_2 = 0\cdot 2^3 + 1\cdot 2^2 + 1\cdot 2^1 + 0\cdot 2^0 = 6$$

$$(1101.1011)_2 = 13 + 1/2 + 1/8 + 1/16 = 13.6875$$

Binary scientific notation

$$(1101.1011)_2 = (1.1011011)_2 \times 2^3$$

Convert an integer by repeated division

1

Divide by 2

Record the integer quotient.

2

Record the remainder

The remainder is always 0 or 1.

3

Repeat

Stop when the quotient reaches 0.

4

Read upward

Remainders from last to first form the binary number.

Next: convert $(13)_{10}$

Convert $(13)_{10}$ to binary

Division	Quotient	Remainder
13 ÷ 2	6	1
6 ÷ 2	3	0
3 ÷ 2	1	1
1 ÷ 2	0	1

Read remainders from bottom to top

1 1 0 1

$$(13)_{10} = (1101)_2$$

Convert a fraction by repeated multiplication

1

Multiply by 2

Keep both the integer part and remaining fraction.

2

Record the integer part

Each recorded integer is a binary digit.

3

Repeat

Stop when the remaining fraction reaches 0.

4

Read downward

Integer parts from first to last form the fraction.

Next: convert $(0.625)_{10}$

Convert 0.625_{10} to binary

Multiplication	Integer part	Remaining fraction
$0.625 \times 2 = 1.25$	1	0.25
$0.25 \times 2 = 0.5$	0	0.5
$0.5 \times 2 = 1.0$	1	0

Read integer parts from top to bottom

. 1 0 1

$$(0.625)_{10} = (0.101)_2$$

Combine the integer and fractional parts

Integer part

$(13)_{10} = (1101)_2$

+

Fractional part

$(0.625)_{10} = (0.101)_2$

$$(13.625)_{10} = (1101.101)_2$$

A floating-point number has five ingredients

$$(-1)^s(d_1 . d_2 d_3 \cdots d_p)_b \times b^k$$

s	b	p	d_i	k
sign	**base**	**precision**	**significand digits**	**exponent**
0 for positive; 1 for negative	2 for binary machines	fixed number of significand digits	d_1 is nonzero for a normal number	$k_{min} \le k \le k_{max}$

The triple (p, k_{min}, k_{max}) defines the format.

Normal and subnormal numbers

Normal

$$(-1)^s(d_1 . d_2 d_3 \cdots d_p)_b \times b^k$$

$$d_1 \in \{1,2,...,b-1\} \quad k_{\min} \le k \le k_{\max}$$

Subnormal

$$(-1)^s(0.d_2 d_3 \cdots d_p)_b \times b^{k_{\min}}$$

Leading digit is 0
Exponent is fixed at $k_{\min}$
Note: the digits cannot all be equal to zero at the same time! (By convention, the number 0 is not included; but IEEE formats include it separately.)

Subnormals fill the gap between zero and the smallest normal number.

A toy decimal floating-point system

Base	Precision	Minimum exponent	Maximum exponent
$b = 10$	$p = 3$	$k_{\min} = -2$	$k_{\max} = 2$

Normal: $(-1)^s(d_1 . d_2 d_3)_{10} \times 10^k$

Subnormal: $(-1)^s(0.d_2 d_3)_{10} \times 10^{-2}$

The toy system represents a discrete range

Positive subnormals

0.0001, 0.0002, …, 0.0099

Spacing: 0.0001
Smallest positive value: 0.0001

Positive normals

0.0100, 0.0101, …, 999

Spacing grows with the exponent
Largest positive value: 999

Every positive value also has a negative counterpart.

Normal or subnormal?

0.0123 is normal

$0.0123 = 1.23 \times 10^{-2}$

The leading digit is nonzero, and $k = -2$ is permitted.

0.0023 is subnormal

$0.0023 = 0.23 \times 10^{-2}$

Writing 2.30×10^{-3} is not allowed because $k = -3$ is outside the allowed range of exponents.

The format—not algebra alone—decides which representation is legal.

Floating-point systems leave gaps

Near zero, representable values are tightly spaced.

Farther away, the gaps become much larger.

Real numbers between the dots are not exactly represented by our floating-point system

Now do the same thing in binary!

$$b = 2, \quad p = 3, \quad k_{\min} = -2, \quad k_{\max} = 2$$

Subnormal values

$$(0.d_2d_3)_2 \times 2^{-2}, \quad d_2, d_3 \in \{0,1\}$$

1/16, 2/16, 3/16

Normal values

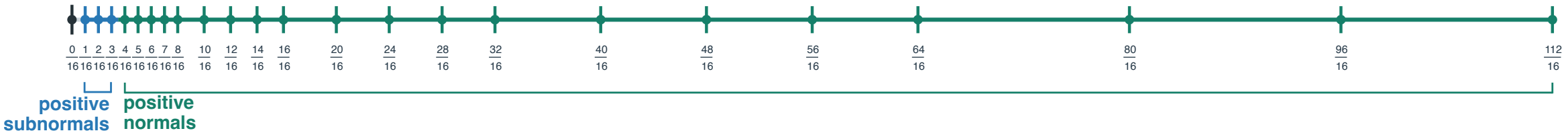
$$(d_1 . d_2d_3)_2 \times 2^{k}, \quad d_1, d_2, d_3 \in \{0,1\}$$

4/16, 5/16, 6/16, 7/16, 8/16,
10/16, 12/16, …, 112/16

The smallest positive normal is 4/16.

Spacing doubles whenever the exponent increases.

Number line is not uniform



A fixed number of significand digits creates non-uniform spacing.

Three outcomes for an unrepresented result

Overflow

$|x|$ exceeds the largest representable value.

Typical result: $\pm\infty$ or an exception

Underflow

$|x|$ is smaller than the representable range.

Typical result: a subnormal value or 0

Rounding

x lies inside the range but between two machine numbers.

Choose a nearby representable value

Finite range + finite precision ⇒ a rule is needed for every calculation.

IEEE 754 standardizes common binary formats

The current standard is IEEE 754-2019.

Format	Sign bits	Significand bits	Exponent bits	Precision	Approx. decimal digits
binary16 (half)	1	10	5	11 bits	3-4
binary32 (single)	1	23	8	24 bits	7
binary64 (double)	1	52	11	53 bits	16
binary128 (quad)	1	112	15	113 bits	34

The leading 1 of a normal binary significand is implicit, so fraction bits + 1 = precision.

IEEE 754 encodes special values

Signed zero

+0 and −0 preserve the direction from which an underflowed value approached zero.

Signed infinity

+∞ and −∞ distinguish positive and negative overflow or division by signed zero.

$3/(+0) = +\infty$
$3/(-0) = -\infty$

NaN

NaN means “Not a Number.”

$\infty - \infty = \text{NaN}$
$0 \times \infty = \text{NaN}$
$0/0 = \text{NaN}$

binary32 packs sign, exponent, and significand into 32 bits

$b=2$, $p=24$, $k_{\min}=-126$, $k_{\max}=127$

Normal significand: $(1.d_1d_2\ldots d_{23})_2$

The exponent range is 127-(-126)=254

But we add the two special values 0 and infinity.

This leads to $254+2=256=2^8$ — this leads to 8 bits for the exponent.

binary32 includes special values

+0	0	00000000	00000000000000000000000
−0	1	00000000	00000000000000000000000
+∞	0	11111111	00000000000000000000000
−∞	1	11111111	00000000000000000000000
NaN	0	11111111	10000000000000000000000

First convert the magnitude to normalized binary

Start

52.234375_{10}

Convert

110100.001111_2

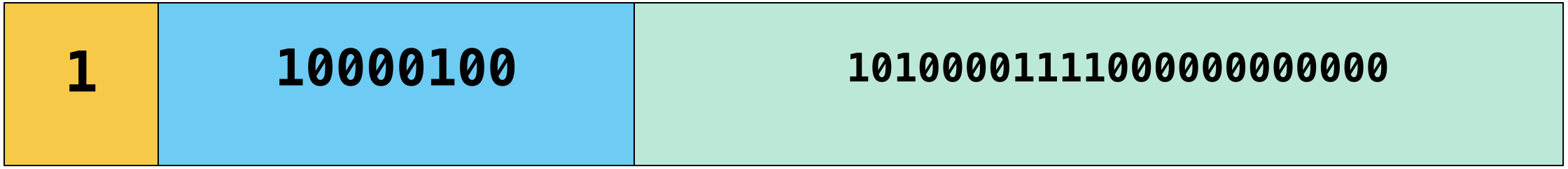
Normalize

$1.10100001111_2 \times 2^5$

sign bit = 1 • unbiased exponent = 5 • fraction begins 10100001111

binary32 stores a biased exponent: 5 + 127 = 132 = 10000100_2

Assemble the three binary32 fields



sign	5 + 127 = 132	remaining significand bits

$$-52.234375 \;\mapsto\; 1 \mid 10000100 \mid 10100001111000000000000$$

Error can be measured in absolute or relative terms

Actual error

$\varepsilon = x - \hat{x}$

Keeps the sign of the discrepancy.

Absolute error

$\varepsilon_{\text{abs}} = |x - \hat{x}|$

Carries the same units as x.

Relative error

$\varepsilon_{\text{rel}} = |x - \hat{x}| \,/\, |x|$

Measures error against the scale of x.

Relative error is often the more meaningful measure of accuracy.

Example

$$x = 0.3000 \times 10^{1} = 3.0 \qquad \hat{x} = 0.3100 \times 10^{1} = 3.1$$

Actual $\varepsilon = 3.0 - 3.1 = -0.1$

Absolute $\varepsilon_{\text{abs}} = |-0.1| = 0.1$

Relative $\varepsilon_{\text{rel}} = 0.1 / 3.0 = 0.03333\ldots$

Relative error ≈ 3.33%

Scaling x changes absolute error—but not relative error

Case	Exact x	Approximation $\hat{x}$	Absolute error	Relative error
a	0.3000×10^{1}	0.3100×10^{1}	0.1	0.03333…
b	0.3000×10^{-3}	0.3100×10^{-3}	0.1×10^{-4}	0.03333…
c	0.3000×10^{4}	0.3100×10^{4}	0.1×10^{3}	0.03333…

The absolute errors vary by seven orders of magnitude.

The relative error is identical in every row.

Same absolute error, different quality

Approximation 1

$x_1 = 1.333$
$\hat{x}_1 = 1.334$

absolute error = 0.001
relative error ≈ 0.00075

Good approximation

Approximation 2

$x_2 = 0.001$
$\hat{x}_2 = 0.002$

absolute error = 0.001
relative error = 1

100% relative error

Significant digits describe a numeral; precision describes a format

$$3.721498 \times 10^{-5}$$

most significant

least significant

7 significant digits

A property of this written normalized numeral.

Precision p

A fixed property of a floating-point format; it does not by itself indicate accuracy.

Chopping discards every digit after position p

$$x = d_1.d_2 \ldots d_p \; d_{p+1} \; d_{p+2} \ldots \times 10^k$$

keep | discard

$$\mathrm{fl}(x) = d_1.d_2 \ldots d_p \times 10^k$$

Chopping always moves toward zero in magnitude.

Rounding inspects the next digit

If the next digit is 0–4

Round down

Keep the first p digits exactly as they are.

If the next digit is 6–9

Round up

Add 1 to the final retained digit d_p.

A next digit of 5 is a midpoint and needs a tie-breaking rule.

Ties-to-even chooses the even endpoint

0.365 to two decimal places

Candidates: 0.36 and 0.37

6 is even ⇒ choose 0.36

0.475 to two decimal places

Candidates: 0.47 and 0.48

8 is even ⇒ choose 0.48

0.365 → 0.36 0.475 → 0.48

Balanced tie-breaking reduces systematic upward or downward bias across large datasets.

Five-digit chopping and rounding disagree on π

$$\pi = 3.14159265\ldots$$

Chop after five digits

3.1415

Ignore the sixth digit.

Round to five digits

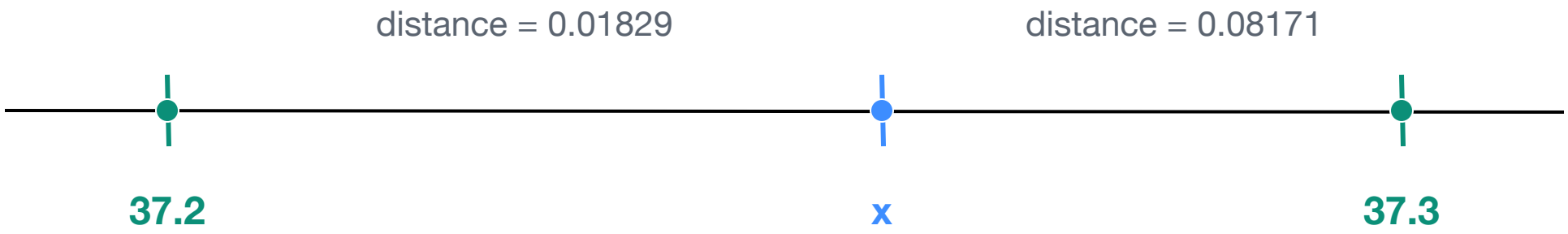
3.1416

The sixth digit is 9, so round up.

Both are five-digit floating-point approximations; the rule determines fl(π).

Choose the nearest decimal toy number

$$x = 37.21829$$



$$\mathrm{fl}(37.21829) = 37.2$$

At a midpoint, parity decides

$x = 37.25$

Midpoint between 37.2 and 37.3

Last retained digits: 2 and 3
2 is even ⇒ fl(x) = 37.2

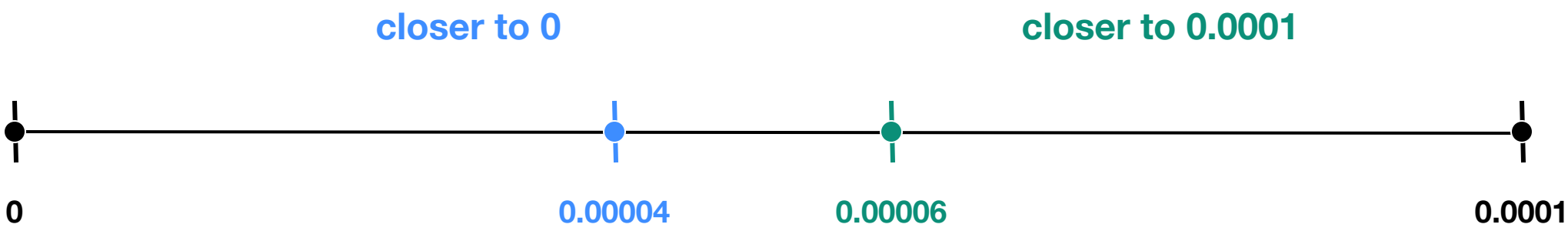
$x = 0.00235$

Midpoint between 0.0023 and 0.0024

Last retained digits: 3 and 4
4 is even ⇒ fl(x) = 0.0024

Ties-to-even is about the final retained digit—not the discarded 5.

Near zero: zero or the first subnormal



fl(0.00004) = 0

Rounds to zero

fl(0.00006) = 0.0001

Rounds up to the first subnormal

Binary rounding still means “choose the nearest point”

x = 0.1

Neighbors: 1/16 = 0.0625 and 2/16 = 0.125

Distances: 0.0375 and 0.025

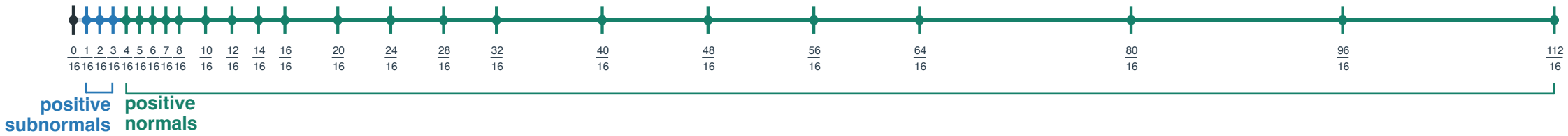
fl(0.1) = 2/16 = 0.125

x = 0.2

Neighbors: 3/16 = 0.1875 and 4/16 = 0.25

Distances: 0.0125 and 0.05

fl(0.2) = 3/16 = 0.1875



Binary ties use the last stored bit

9/32 = 0.28125

Midpoint between 4/16 and 5/16

$4/16 = (1.00)_2 \times 2^{-2}$ → last bit 0
$5/16 = (1.01)_2 \times 2^{-2}$ → last bit 1

fl(9/32) = 4/16 = 1/4

11/32 = 0.34375

Midpoint between 5/16 and 6/16

$5/16 = (1.01)_2 \times 2^{-2}$ → last bit 1
$6/16 = (1.10)_2 \times 2^{-2}$ → last bit 0

fl(11/32) = 6/16 = 3/8

Even in binary means the final retained bit is 0.

Nearest rounding has a uniform relative-error bound

For a normal-range x rounded to the nearest representable value:

$$\frac{|x - \mathrm{fl}(x)|}{|x|} \leq u := \frac{1}{2} b^{-p+1}$$

$$\mathrm{fl}(x) = x(1 + \delta), \quad |\delta| \leq u$$

u is the **unit roundoff**: the worst-case relative error introduced by one correctly rounded operation.

Proof. Consider the exponent value $k \in \{k_{\min}, k_{\min}+1, \ldots, k_{\max}\}$, with $k_{\min}$ and $k_{\max}$ specified by the floating-point system, such that $b^k \leq |x| \leq b^{k+1}$. (This value is simply $k = \lfloor \log_b |x| \rfloor$.) Within this exponent range, consecutive p-digit floating-point numbers are separated by $\Delta = b^{k-p+1}$. Because $\mathrm{fl}(x)$ is a nearest floating-point number, it holds that

$$|x - \mathrm{fl}(x)| \leq \frac{\Delta}{2} = \frac{1}{2} b^{k-p+1}. \tag{48}$$

At an exact midpoint, the rounds-to-even rule chooses one of the two endpoints, but the error is still exactly $\frac{\Delta}{2}$. Therefore, using the fact that $|x| \geq b^k$,

$$\frac{|x - \mathrm{fl}(x)|}{|x|} \leq \frac{\frac{1}{2} b^{k-p+1}}{|x|} \tag{49}$$

$$\leq \frac{\frac{1}{2} b^{k-p+1}}{b^k} \tag{50}$$

$$= \frac{1}{2} b^{-p+1} \tag{51}$$

$$= u, \tag{52}$$

as desired. Now, define

$$\delta = \frac{\mathrm{fl}(x) - x}{x}. \tag{53}$$

Then, $\mathrm{fl}(x) - x = x\delta$, which implies that $\mathrm{fl}(x) = x(1+\delta)$. Moreover, using the steps shown above, we have that $|\delta| = \left| \frac{\mathrm{fl}(x)-x}{x} \right| \leq u$. This concludes the proof. ■

Unit roundoff vs. machine epsilon

Unit roundoff

$u = \frac{1}{2} b^{1-p}$

Worst-case relative error for rounding to nearest floating-point number.

Machine epsilon

$\varepsilon_{\text{machine}} = b^{1-p} = 2u$

Distance from 1 to the next larger floating-point number.

Format	p	$\varepsilon_{\text{machine}}$
binary32	24	2^{-23}
binary64	53	2^{-52}

Summary

01	**Representation**	A floating-point format stores only discrete points from the real line.
02	**Range**	Exponent limits create overflow, underflow, normals, and subnormals.
03	**Rounding**	Arithmetic usually returns a nearby representable value; ties-to-even avoids directional bias.
04	**Error**	Relative error and unit roundoff connect local representation choices to algorithmic accuracy.

Next question: how do these local errors propagate through an algorithm?