THE END OF NUMERICAL ERROR

John L. Gustafson, Ph.D.

CTO, Ceranovo

Director at Massively Parallel Technologies, Etaphase, REX Computing, and Clustered Systems

johngustafson@earthlink.net

Why ask for 10^{18} flops per second?

Why ask for 10^{18} flops per second?

Henry Ford once said, “If I had asked my customers what they wanted, they would have said they wanted a faster horse.”

Why ask for 10^{18} flops per second?

Henry Ford once said, "If I had asked my customers what they wanted, they would have said they wanted a faster horse."

That is what we are doing now with supercomputing: asking for a faster horse, not what comes *after horses*.

Why ask for 10^{18} flops per second?

Henry Ford once said, “If I had asked my customers what they wanted, they would have said they wanted a faster horse.”

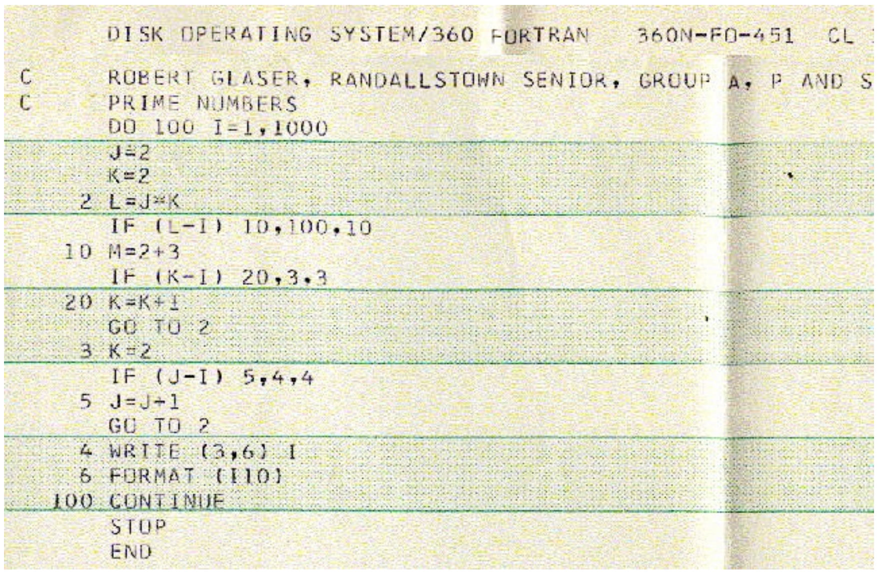
That is what we are doing now with supercomputing: asking for a faster horse, not what comes *after horses*.

We do not need 10^{18} sloppy operations per second that produce rounding errors of unknown size; we need a new foundation for computer arithmetic.

Analogy: Printing in 1970 vs. 2015

Analogy: Printing in 1970 vs. 2015

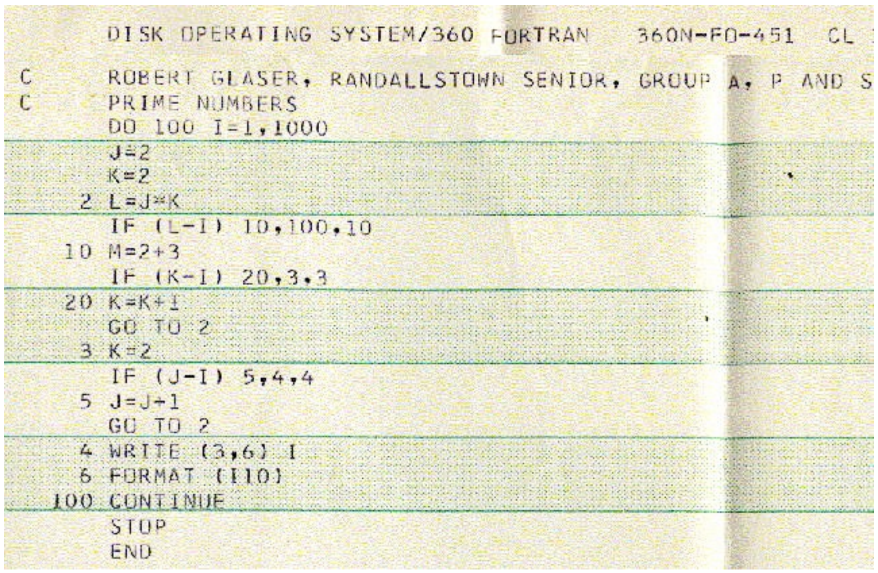
1970: 30 sec per page



```
      DISK OPERATING SYSTEM/360 FORTRAN    360N-FO-451   CL
C     ROBERT GLASER, RANDALLSTOWN SENIOR, GROUP A, P AND S
C     PRIME NUMBERS
      DO 100 I=1,1000
      J=2
      K=2
    2 L=J*K
      IF (L-I) 10,100,10
   10 M=2+3
      IF (K-I) 20,3,3
   20 K=K+1
      GO TO 2
    3 K=2
      IF (J-I) 5,4,4
    5 J=J+1
      GO TO 2
    4 WRITE (3,6) I
    6 FORMAT (I10)
  100 CONTINUE
      STOP
      END
```

Analogy: Printing in 1970 vs. 2015

1970: 30 sec per page

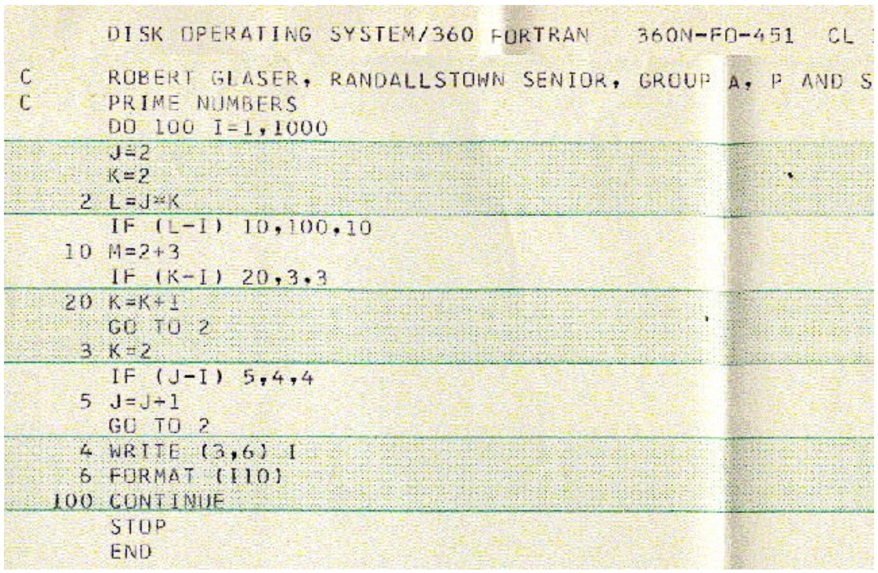


```
      DISK OPERATING SYSTEM/360 FORTRAN    360N-FO-451  CL
C     ROBERT GLASER, RANDALLSTOWN SENIOR, GROUP A, P AND S
C     PRIME NUMBERS
      DO 100 I=1,1000
      J=2
      K=2
    2 L=J*K
      IF (L-I) 10,100,10
   10 M=2+3
      IF (K-I) 20,3,3
   20 K=K+1
      GO TO 2
    3 K=2
      IF (J-I) 5,4,4
    5 J=J+1
      GO TO 2
    4 WRITE (3,6) I
    6 FORMAT (I10)
  100 CONTINUE
      STOP
      END
```

2015: 30 sec per page

Analogy: Printing in 1970 vs. 2015

1970: 30 sec per page



```
      DISK OPERATING SYSTEM/360 FORTRAN    360N-FO-451   CL
C     ROBERT GLASER, RANDALLSTOWN SENIOR, GROUP A, P AND S
C     PRIME NUMBERS
      DO 100 I=1,1000
      J=2
      K=2
    2 L=J*K
      IF (L-I) 10,100,10
   10 M=2+3
      IF (K-I) 20,3,3
   20 K=K+1
      GO TO 2
    3 K=2
      IF (J-I) 5,4,4
    5 J=J+1
      GO TO 2
    4 WRITE (3,6) I
    6 FORMAT (I10)
  100 CONTINUE
      STOP
      END
```

2015: 30 sec per page

Faster technology is for *better* prints,
not thousands of low-quality prints per second.
Why not do the same thing with computer arithmetic?

Big problems facing computing

Big problems facing computing

- Too much energy and power needed per calculation

Big problems facing computing

- Too much energy and power needed per calculation
- More hardware parallelism than we know how to use

Big problems facing computing

- Too much energy and power needed per calculation
- More hardware parallelism than we know how to use
- Not enough bandwidth (the “memory wall”)

Big problems facing computing

- Too much energy and power needed per calculation
- More hardware parallelism than we know how to use
- Not enough bandwidth (the “memory wall”)
- Rounding errors more treacherous than people realize

Big problems facing computing

- Too much energy and power needed per calculation
- More hardware parallelism than we know how to use
- Not enough bandwidth (the “memory wall”)
- Rounding errors more treacherous than people realize
- Rounding errors prevent use of parallel methods

Big problems facing computing

- Too much energy and power needed per calculation
- More hardware parallelism than we know how to use
- Not enough bandwidth (the “memory wall”)
- Rounding errors more treacherous than people realize
- Rounding errors prevent use of parallel methods
- Sampling errors turn physics simulations into guesswork

Big problems facing computing

- Too much energy and power needed per calculation
- More hardware parallelism than we know how to use
- Not enough bandwidth (the “memory wall”)
- Rounding errors more treacherous than people realize
- Rounding errors prevent use of parallel methods
- Sampling errors turn physics simulations into guesswork
- Numerical methods are hard to use, require experts

Big problems facing computing

- Too much energy and power needed per calculation
- More hardware parallelism than we know how to use
- Not enough bandwidth (the “memory wall”)
- Rounding errors more treacherous than people realize
- Rounding errors prevent use of parallel methods
- Sampling errors turn physics simulations into guesswork
- Numerical methods are hard to use, require experts
- IEEE floats give different answers on different platforms

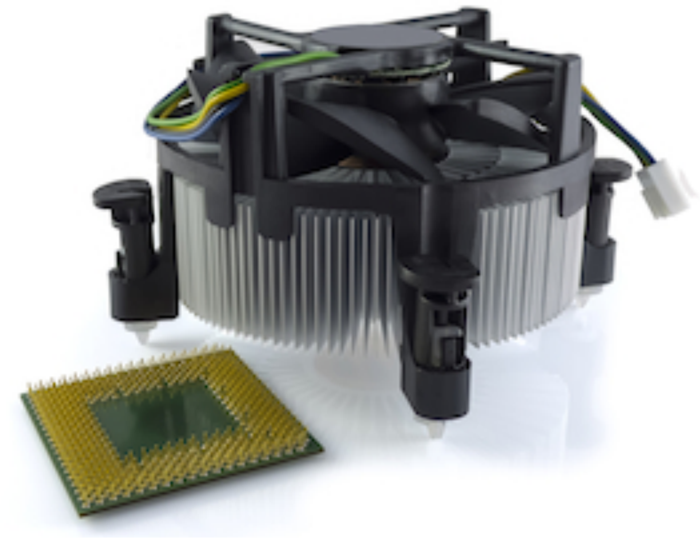
The ones *vendors* care most about

- Too much energy and power needed per calculation
- More hardware parallelism than we know how to use
- Not enough bandwidth (the "memory wall")
- Rounding errors more treacherous than people realize
- Rounding errors prevent use of parallel methods
- Sampling errors turn physics simulations into guesswork
- Numerical methods are hard to use, require experts
- IEEE floats give different answers on different platforms

Too much power and heat needed

Too much power and heat needed

- Huge heat sinks

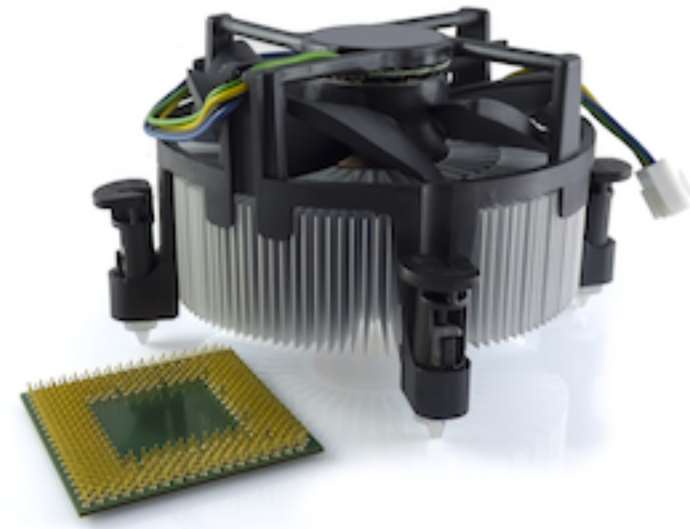


Too much power and heat needed

- Huge heat sinks
- 20 MW limit for exascale

Too much power and heat needed

- Huge heat sinks
- 20 MW limit for exascale
- Data center electric bills

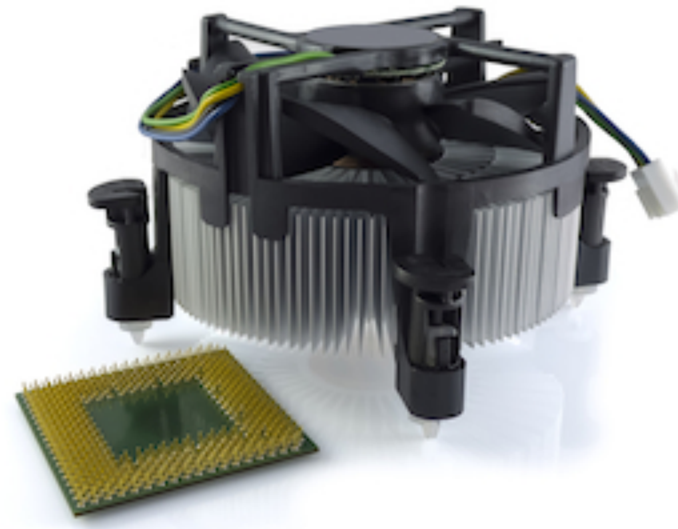


Too much power and heat needed

- Huge heat sinks
- 20 MW limit for exascale
- Data center electric bills
- Mobile device battery life

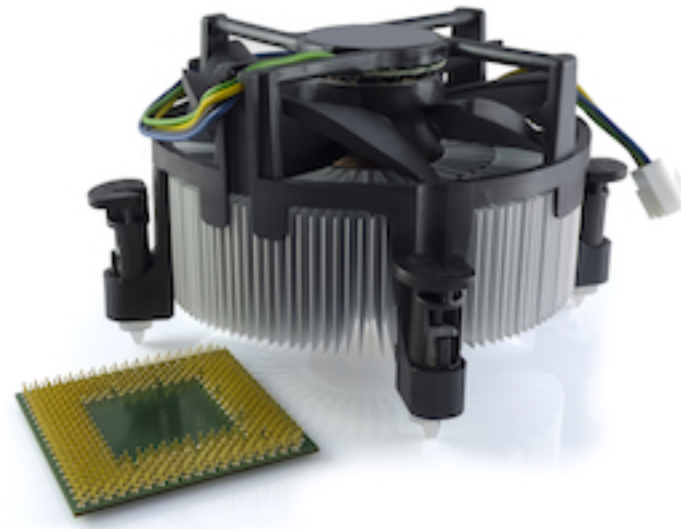
Too much power and heat needed

- Huge heat sinks
- 20 MW limit for exascale
- Data center electric bills
- Mobile device battery life
- Heat intensity means *bulk*



Too much power and heat needed

- Huge heat sinks
- 20 MW limit for exascale
- Data center electric bills
- Mobile device battery life
- Heat intensity means *bulk*
- *Bulk* increases *latency*



Too much power and heat needed

- Huge heat sinks
- 20 MW limit for exascale
- Data center electric bills
- Mobile device battery life
- Heat intensity means *bulk*
- *Bulk* increases *latency*
- *Latency* limits *speed*

More parallel hardware than we can use

- Huge clusters usually partitioned into 10s, 100s of cores
- Few algorithms exploit millions of cores except LINPACK
- *Capacity* is not a substitute for *capability*!

Not enough bandwidth ("Memory wall")

Operation	Energy consumed	Time needed
64-bit multiply-add	200 pJ	1 nsec
Read 64 bits from cache	800 pJ	3 nsec
Move 64 bits across chip	2000 pJ	5 nsec
Execute an instruction	7500 pJ	1 nsec
Read 64 bits from DRAM	***12000 pJ***	***70 nsec***

Not enough bandwidth (“Memory wall”)

Operation	**Energy consumed**	**Time needed**
64-bit multiply-add	200 pJ	1 nsec
Read 64 bits from cache	800 pJ	3 nsec
Move 64 bits across chip	2000 pJ	5 nsec
Execute an instruction	7500 pJ	1 nsec
Read 64 bits from DRAM	***12000 pJ***	***70 nsec***

Notice that 12000 pJ @ 3 GHz = 36 watts!

Not enough bandwidth (“Memory wall”)

Operation	**Energy consumed**	**Time needed**
64-bit multiply-add	200 pJ	1 nsec
Read 64 bits from cache	800 pJ	3 nsec
Move 64 bits across chip	2000 pJ	5 nsec
Execute an instruction	7500 pJ	1 nsec
Read 64 bits from DRAM	***12000 pJ***	***70 nsec***

Notice that 12000 pJ @ 3 GHz = 36 watts!

One-size-fits-all overkill 64-bit precision
wastes energy, storage, bandwidth

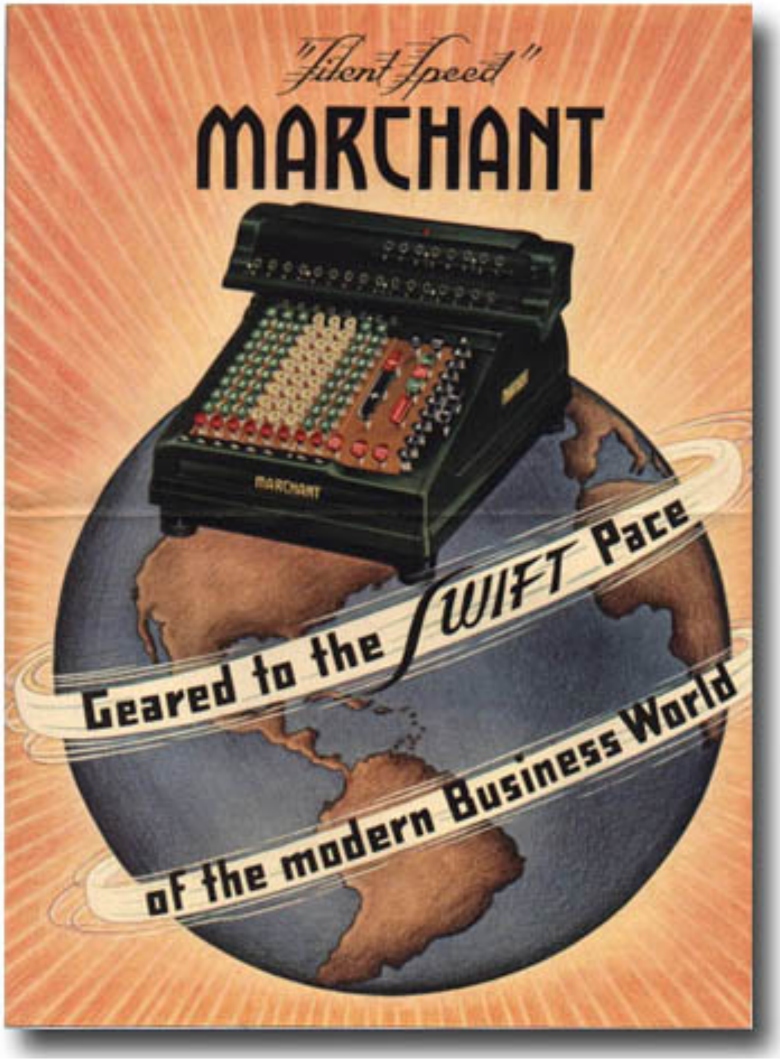
Happy 101st Birthday, Floating Point

1914: Torres y Quevedo proposes automatic computing with fraction & exponent.
2015: We still use a format designed for World War I hardware capabilities.

The “Original Sin” of Computer Math

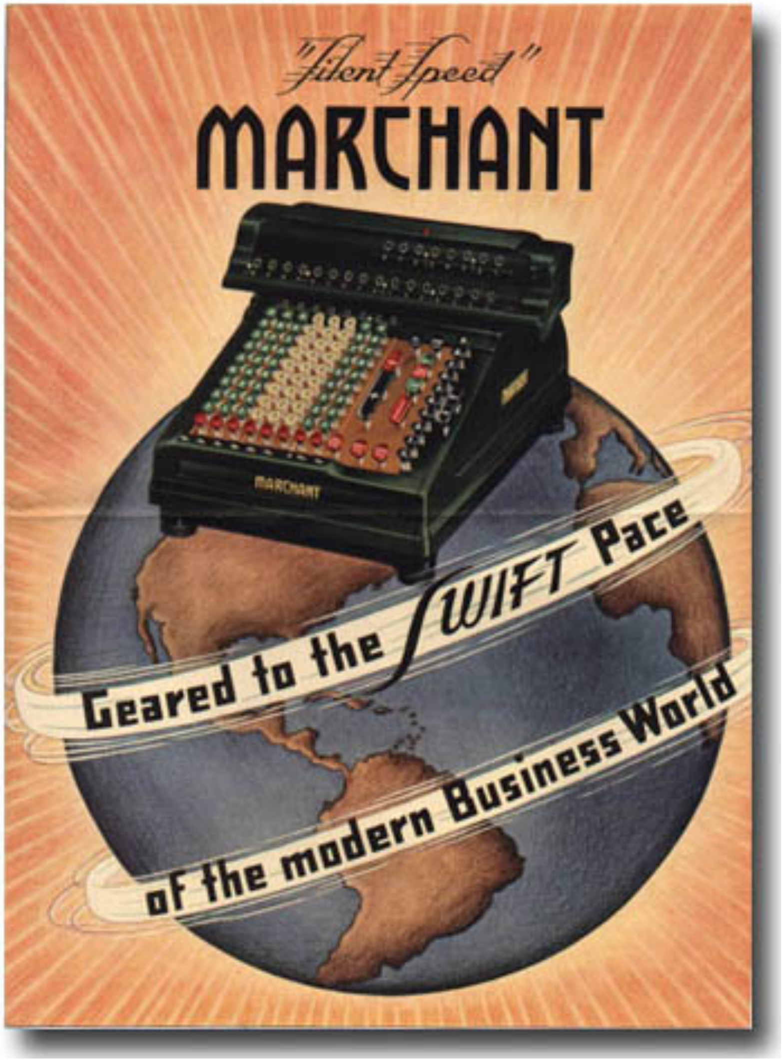
“The computer cannot give you the exact value, sorry. Use *this* value instead. It’s close.”

Floats worked for *visible* scratch work



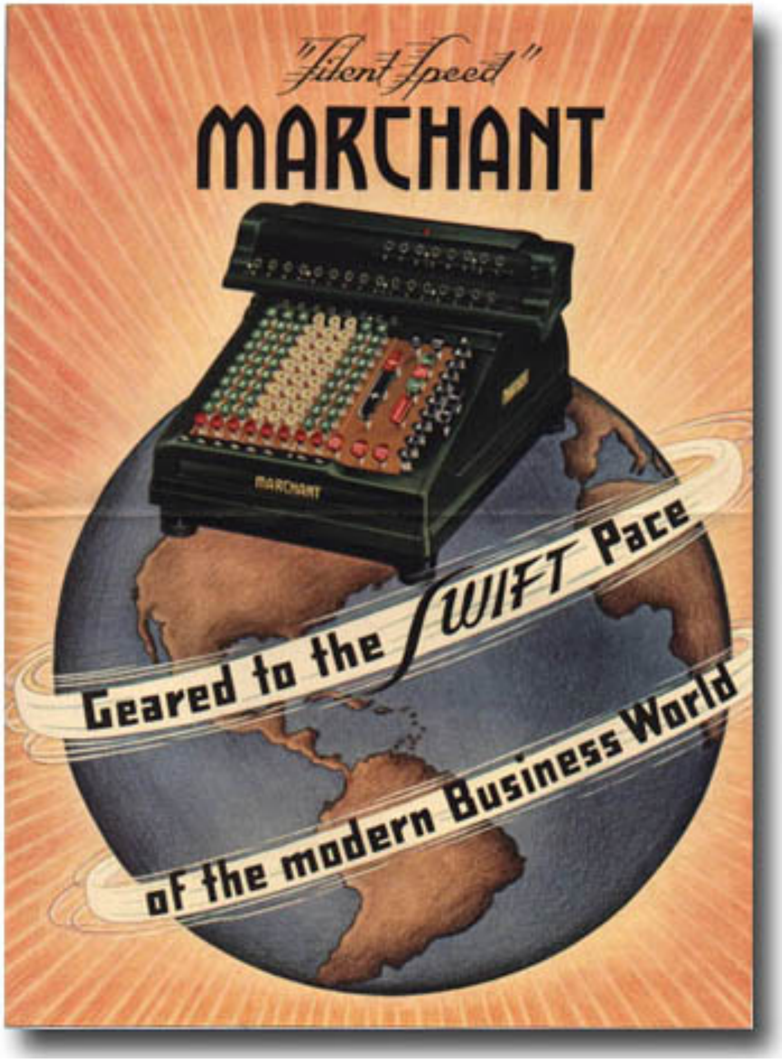
Floats worked for *visible* scratch work

- OK for *manual* calculations
  - Operator sees, remembers errors
  - Can head off overflow, underflow



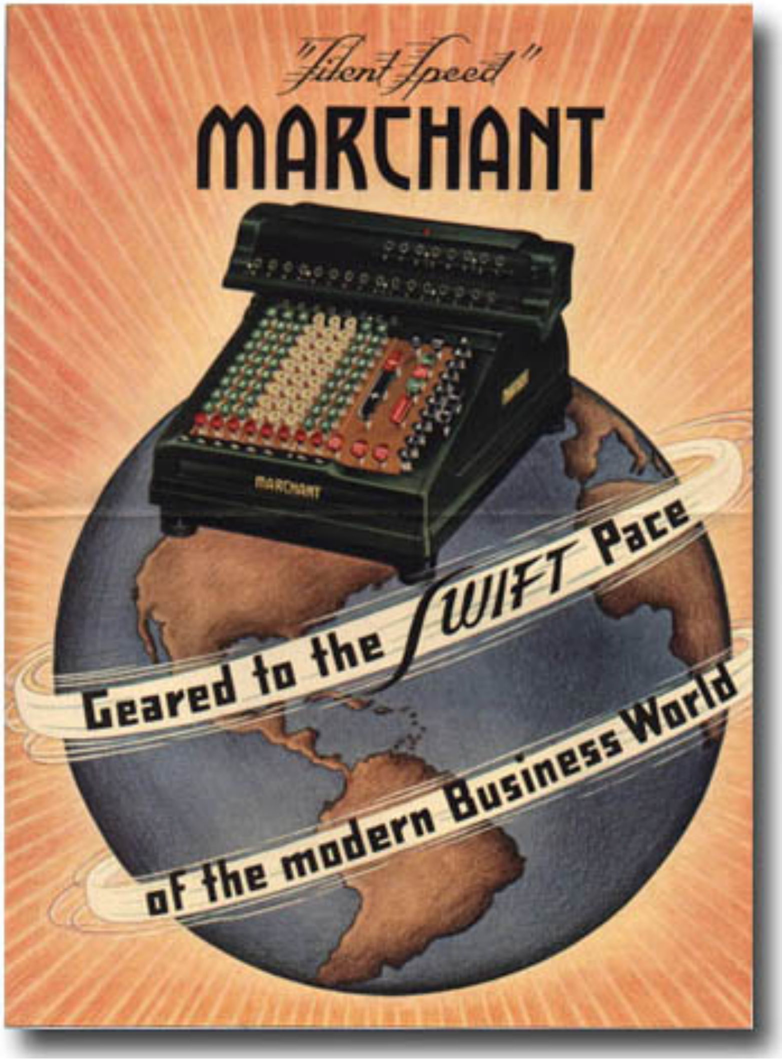
Floats worked for *visible* scratch work

- OK for *manual* calculations
  - Operator sees, remembers errors
  - Can head off overflow, underflow
- Automatic math *hides* all that



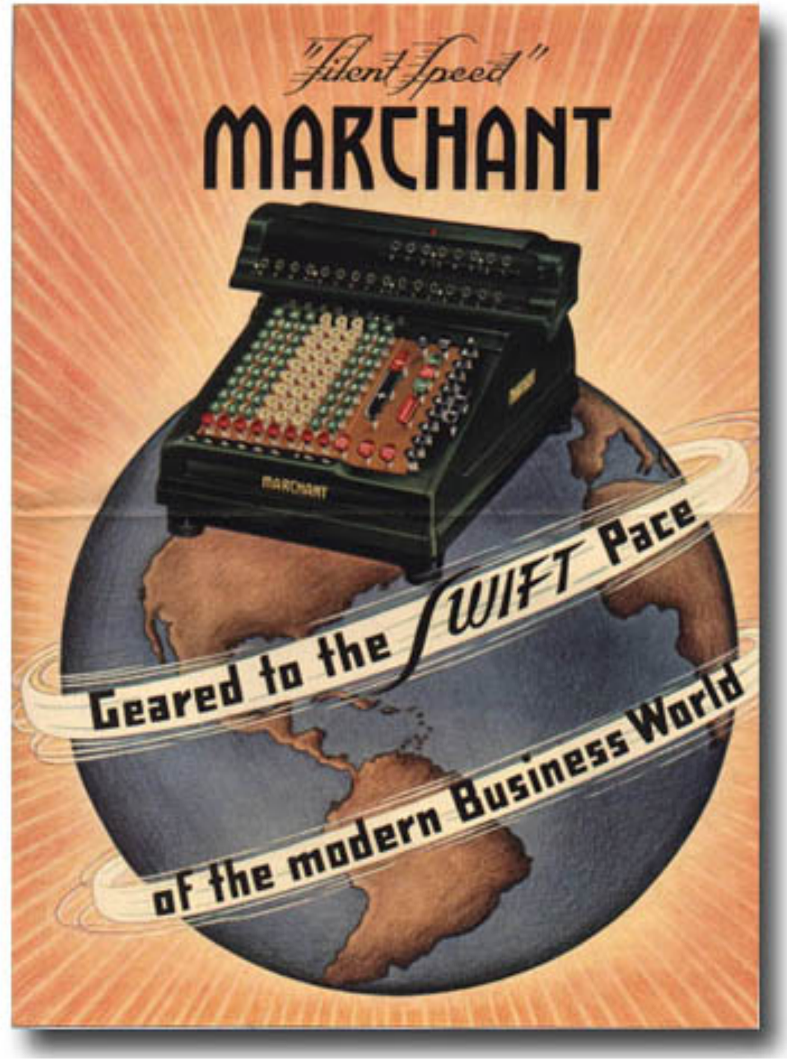
Floats worked for *visible* scratch work

- OK for *manual* calculations
  - Operator sees, remembers errors
  - Can head off overflow, underflow
- Automatic math *hides* all that
- Disobeys algebraic laws



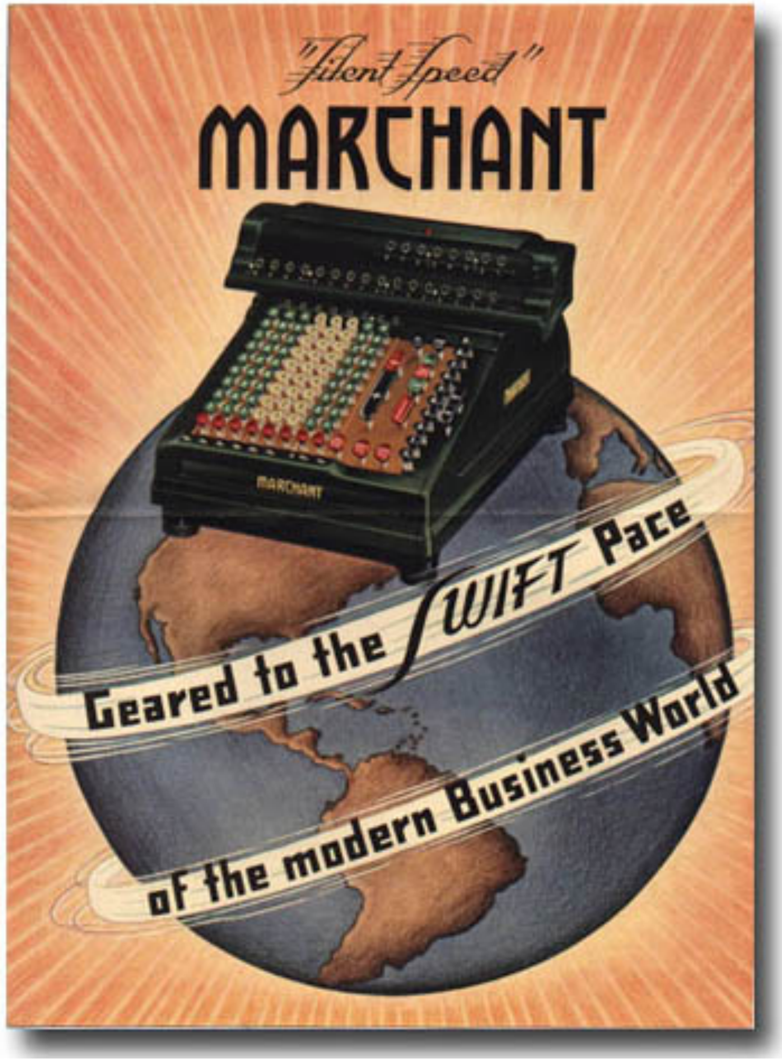
Floats worked for *visible* scratch work

- OK for *manual* calculations
  - Operator sees, remembers errors
  - Can head off overflow, underflow
- Automatic math *hides* all that
- Disobeys algebraic laws
- IEEE 754 “standard” is really the IEEE 754 *guideline*; optional rules spoil consistent results



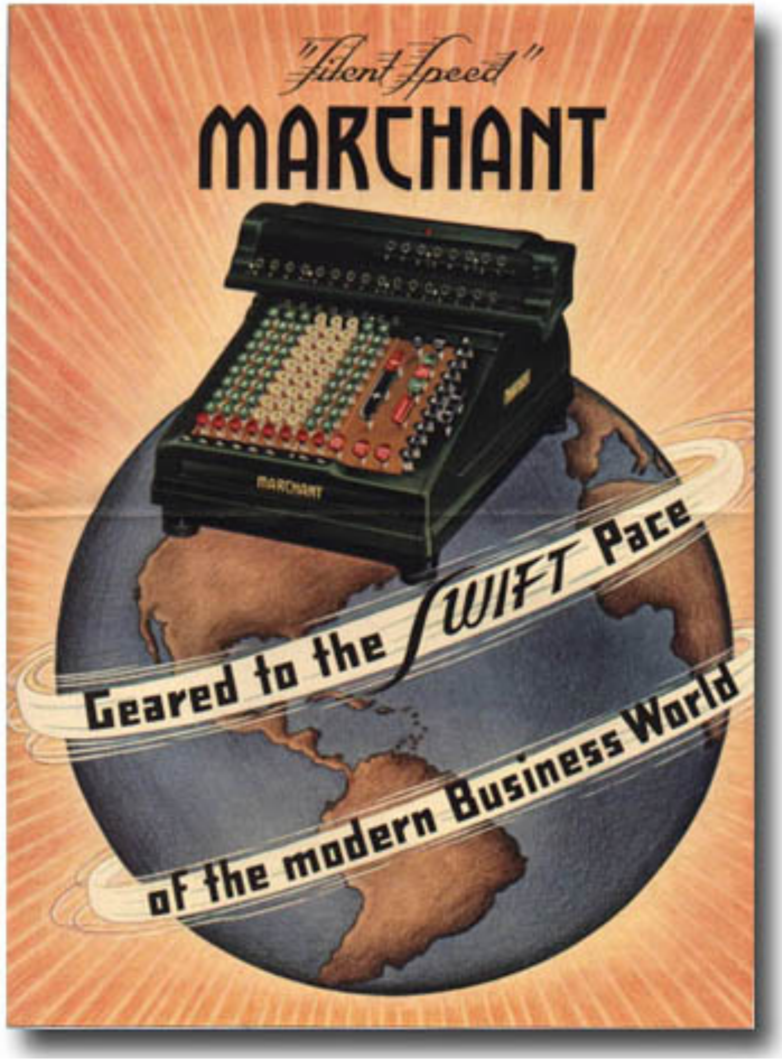
Floats worked for *visible* scratch work

- OK for *manual* calculations
  - Operator sees, remembers errors
  - Can head off overflow, underflow
- Automatic math *hides* all that
- Disobeys algebraic laws
- IEEE 754 "standard" is really the IEEE 754 *guideline*; optional rules spoil consistent results
- Wastes bit patterns as NaN values (NaN = Not a Number)

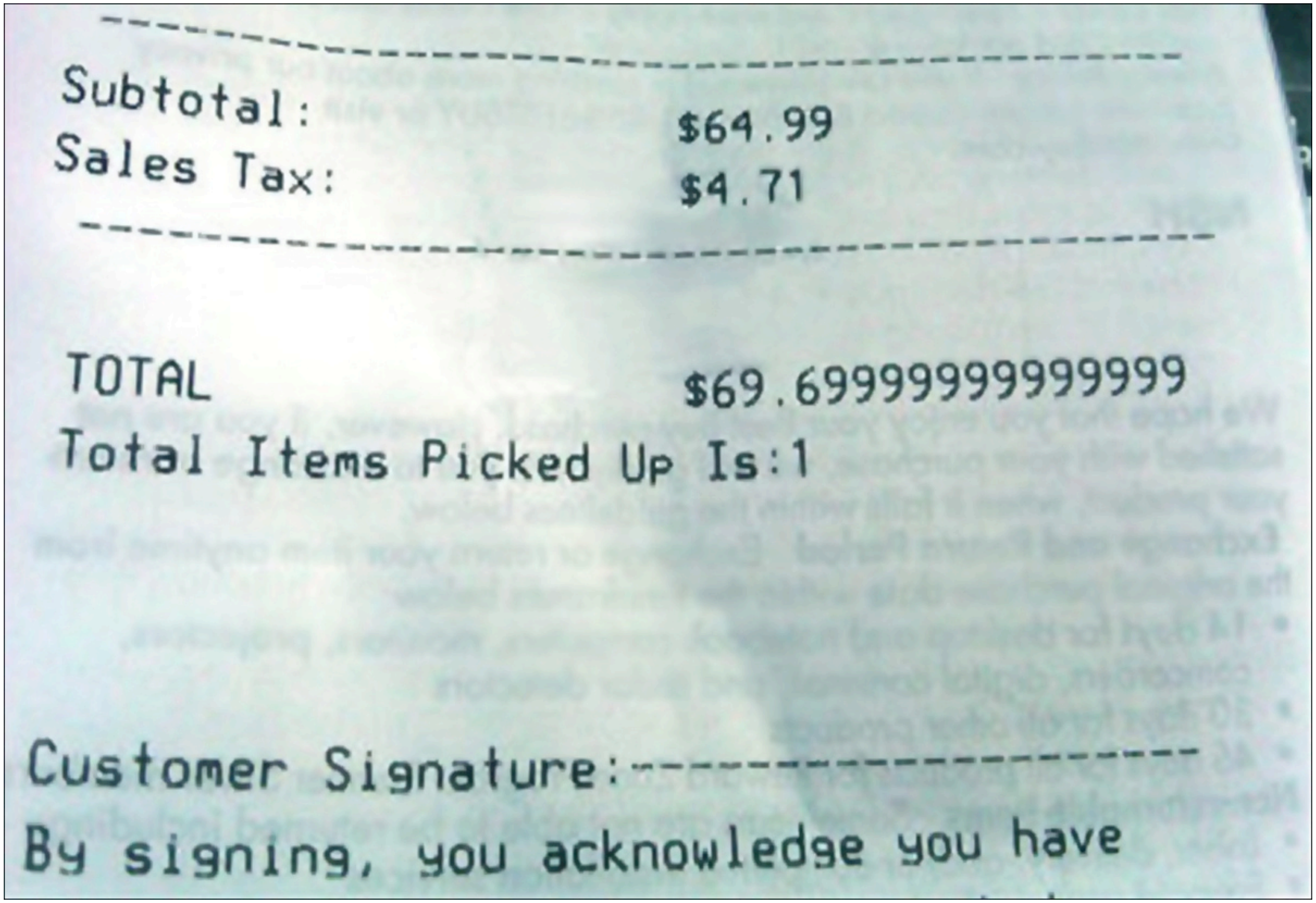


Floats worked for *visible* scratch work

- OK for *manual* calculations
  - Operator sees, remembers errors
  - Can head off overflow, underflow
- Automatic math *hides* all that
- Disobeys algebraic laws
- IEEE 754 “standard” is really the IEEE 754 *guideline*; optional rules spoil consistent results
- Wastes bit patterns as NaN values (NaN = Not a Number)
- No one sees processor “flags”



This is just… sad.



Floats *prevent use of parallelism*

Floats *prevent use of parallelism*

- No associative property for floats

Floats *prevent use of parallelism*

- No associative property for floats
- $(a + b) + (c + d)$ (parallel) $\neq ((a + b) + c) + d$ (serial)

Floats *prevent use of parallelism*

- No associative property for floats
- $(a + b) + (c + d)$ (parallel) $\neq ((a + b) + c) + d$ (serial)
- Looks like a "wrong answer"

Floats *prevent use of parallelism*

- No associative property for floats
- $(a + b) + (c + d)$ (parallel) $\neq ((a + b) + c) + d$ (serial)
- Looks like a “wrong answer”
- Programmers trust serial, reject parallel

Floats *prevent use of parallelism*

- No associative property for floats
- $(a + b) + (c + d)$ (parallel) $\neq ((a + b) + c) + d$ (serial)
- Looks like a "wrong answer"
- Programmers trust serial, reject parallel
- IEEE floats report rounding, overflow, underflow in *processor register bits that no one ever sees.*

A New Number Format: The Unum

- Universal numbers

A New Number Format: The Unum

- Universal numbers
- **Superset** of IEEE types, both 754 and 1788

A New Number Format: The Unum

- Universal numbers
- **Superset** of IEEE types, both 754 and 1788
- Integers→floats→unums

A New Number Format: The Unum

- Universal numbers
- **Superset** of IEEE types, both 754 and 1788
- Integers➔floats➔unums
- No rounding, no overflow to ∞, no underflow to zero

A New Number Format: The Unum

- Universal numbers
- **Superset** of IEEE types, both 754 and 1788
- Integers➔floats➔unums
- No rounding, no overflow to ∞, no underflow to zero
- They obey algebraic laws!

A New Number Format: The Unum

- Universal numbers
- **Superset** of IEEE types, both 754 and 1788
- Integers➔floats➔unums
- No rounding, no overflow to ∞, no underflow to zero
- They obey algebraic laws!
- Safe to parallelize

A New Number Format: The Unum

- Universal numbers
- **Superset** of IEEE types, both 754 and 1788
- Integers➔floats➔unums
- No rounding, no overflow to ∞, no underflow to zero
- They obey algebraic laws!
- Safe to parallelize
- *Fewer bits* than floats

A New Number Format: The Unum

- Universal numbers
- **Superset** of IEEE types, both 754 and 1788
- Integers→floats→unums
- No rounding, no overflow to ∞, no underflow to zero
- They obey algebraic laws!
- Safe to parallelize
- *Fewer bits* than floats
- But… they're *new*
- Some people don't like *new*

A New Number Format: The Unum

- Universal numbers
- **Superset** of IEEE types, both 754 and 1788
- Integers➔floats➔unums
- No rounding, no overflow to ∞, no underflow to zero
- They obey algebraic laws!
- Safe to parallelize
- *Fewer bits* than floats
- But… they're *new*
- Some people don't like *new*

"You can't boil the ocean."

—*Former Intel exec, when shown the unum idea*

A Key Idea: The Ubit

A Key Idea: The Ubit

We have *always* had a way of expressing reals correctly with a finite set of symbols.

A Key Idea: The Ubit

We have *always* had a way of expressing reals correctly with a finite set of symbols.

Incorrect: $\pi = 3.14$

Correct: $\pi = 3.14\cdots$

A Key Idea: The Ubit

We have *always* had a way of expressing reals correctly with a finite set of symbols.

Incorrect: $\pi = 3.14$

Correct: $\pi = 3.14\cdots$

The latter means 3.14 < pi < 3.15, a **true statement**.

Presence or absence of the "…" is the *ubit*, just like a sign bit. It is 0 if exact, 1 if there are more bits after the last fraction bit, not all 0s and not all 1s.

Three ways to express a big number

Avogadro's number: ~6.022×10^{23} atoms or molecules

Three ways to express a big number

Avogadro's number: ~6.022×10^{23} atoms or molecules

Sign-Magnitude Integer (80 bits):

0	**1111111100001010101001111010001010111111010100001001010011000000000000000000000**

sign

Lots of digits

Three ways to express a big number

Avogadro's number: ~6.022×10^{23} atoms or molecules

Sign-Magnitude Integer (80 bits):

0	1111111000010101010011110100010101111110101000010010100110000000000000000000000

sign

Lots of digits

IEEE Standard Float (64 bits):

0	10001001101	1111110000101010100111101000101011111101010000100011

sign exponent (scale)

fraction

Three ways to express a big number

Avogadro's number: ~6.022×10^{23} atoms or molecules

Sign-Magnitude Integer (80 bits):

0	1111111000010101010011110100010101111110101000010010100110000000000000000000000
sign	Lots of digits

IEEE Standard Float (64 bits):

0	10001001101	1111111000010101010011110100010101111110101000010011
sign	exponent (scale)	fraction

Unum (29 bits):

				utag	
0	11001101	111111100001	1	111	1011
sign	exp.	frac.	ubit	exp. size	frac. size

Self-descriptive "utag" bits track and manage uncertainty, exponent size, and fraction size

Fear of overflow wastes bits, time

- *Huge* exponents… why?
- Fear of overflow, underflow

Fear of overflow wastes bits, time

- *Huge* exponents… why?
- Fear of overflow, underflow
- Easier for hardware designer

Fear of overflow wastes bits, time

- *Huge* exponents… why?
- Fear of overflow, underflow
- Easier for hardware designer
- Universe size / proton size: 10^{40}

Fear of overflow wastes bits, time

- *Huge* exponents… why?
- Fear of overflow, underflow
- Easier for hardware designer
- Universe size / proton size: 10^{40}
- Single precision float range: 10^{83}

Fear of overflow wastes bits, time

- *Huge* exponents… why?
- Fear of overflow, underflow
- Easier for hardware designer
- Universe size / proton size: 10^{40}
- Single precision float range: 10^{83}
- Double precision float range: 10^{632}

Why unums use fewer bits than floats

- Exponent smaller by about 5 – 10 bits, typically
- Trailing zeros in fraction compressed away, saves ~2 bits
- Shorter strings for more common values
- Cancellation removes bits and the need to store them

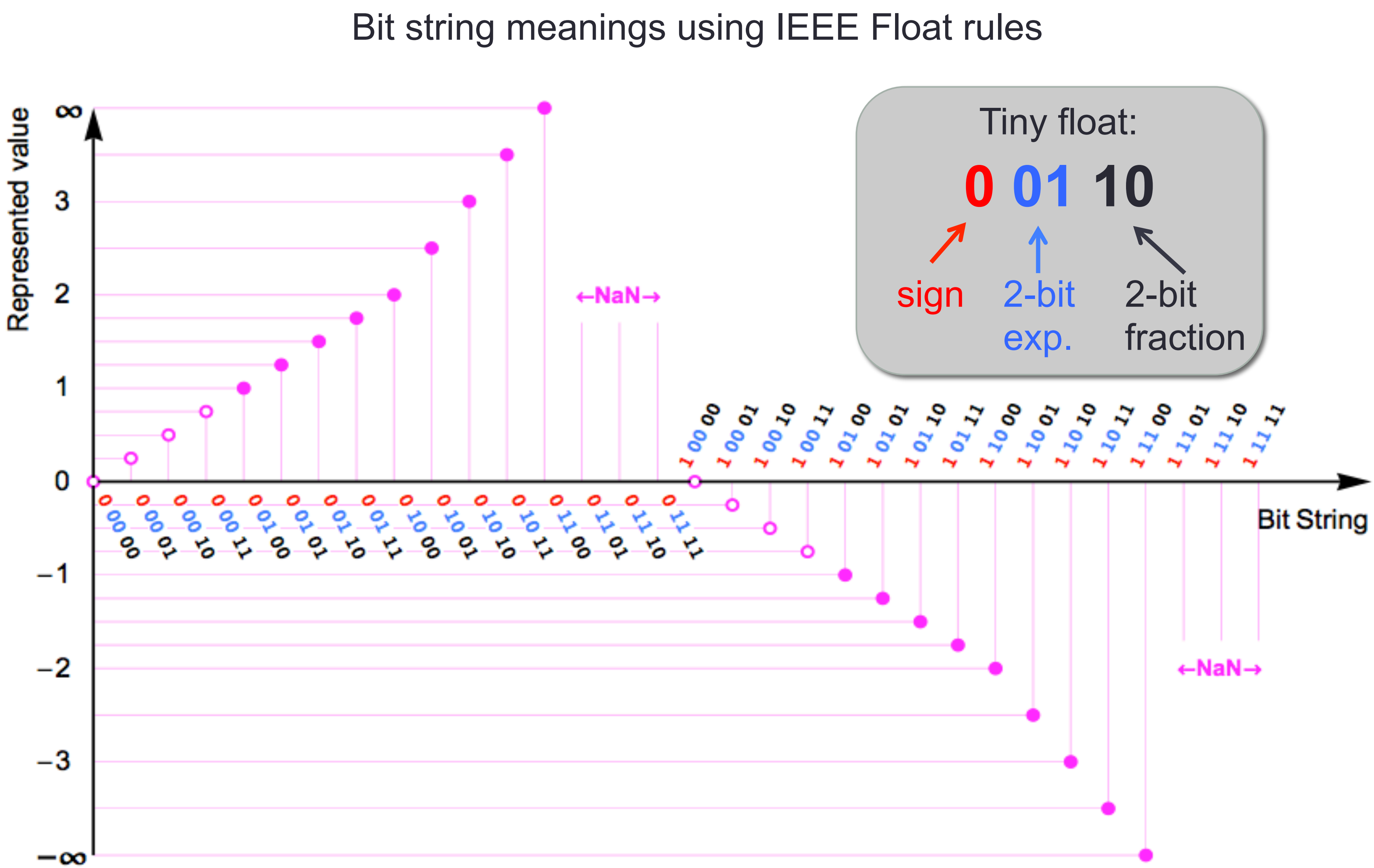
IEEE Standard Float (64 bits):

0	10001001101	1111111000010101010011110100010101111110101000010011

Unum (29 bits):

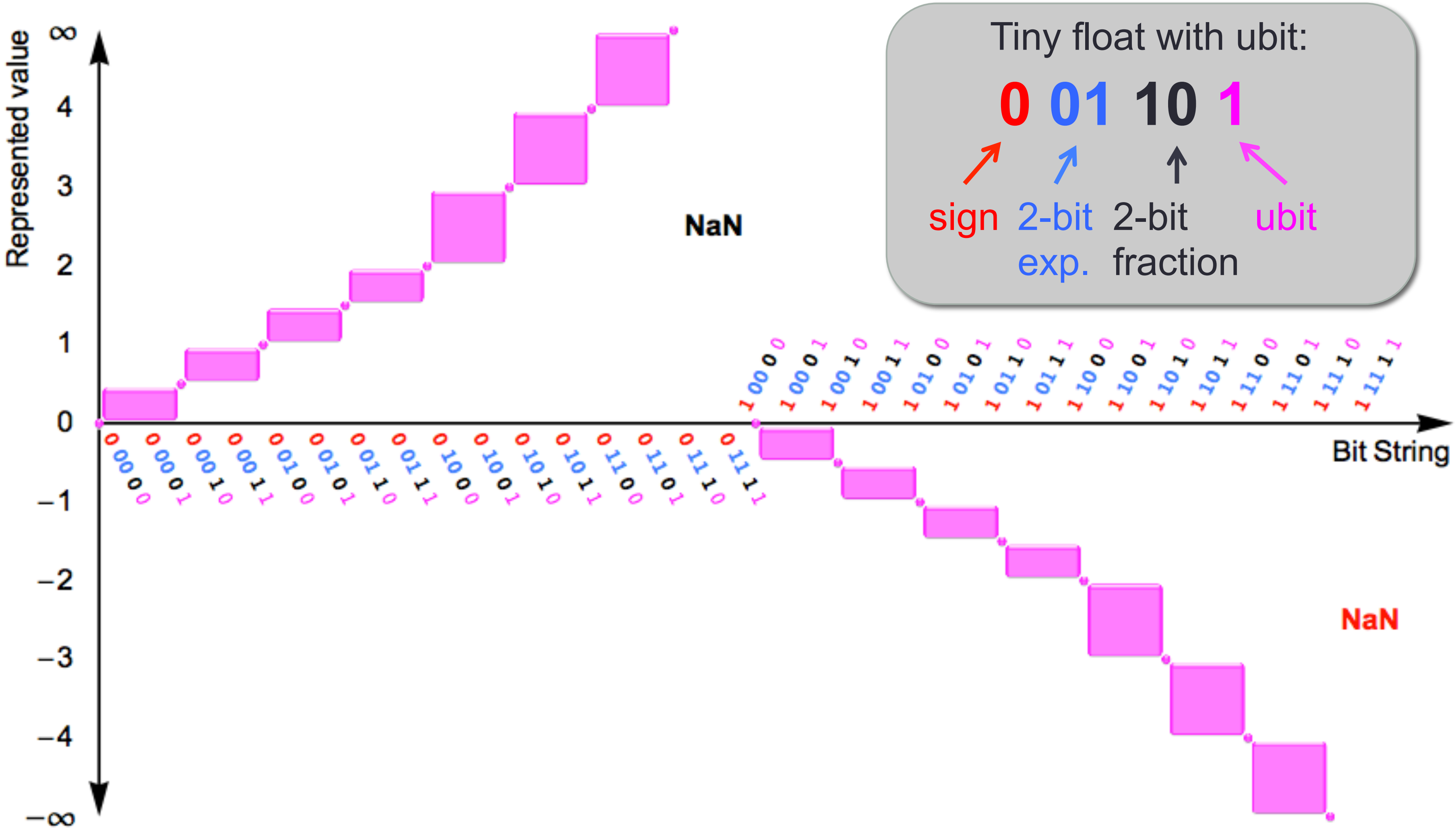
0	11001101	111111100001	1	111	1011

Value plot of tiny IEEE-style floats



Open *ranges*, as well as exact points

Complete representation of *all* real numbers using a finite number of bits



Cc

The Warlpiri unums

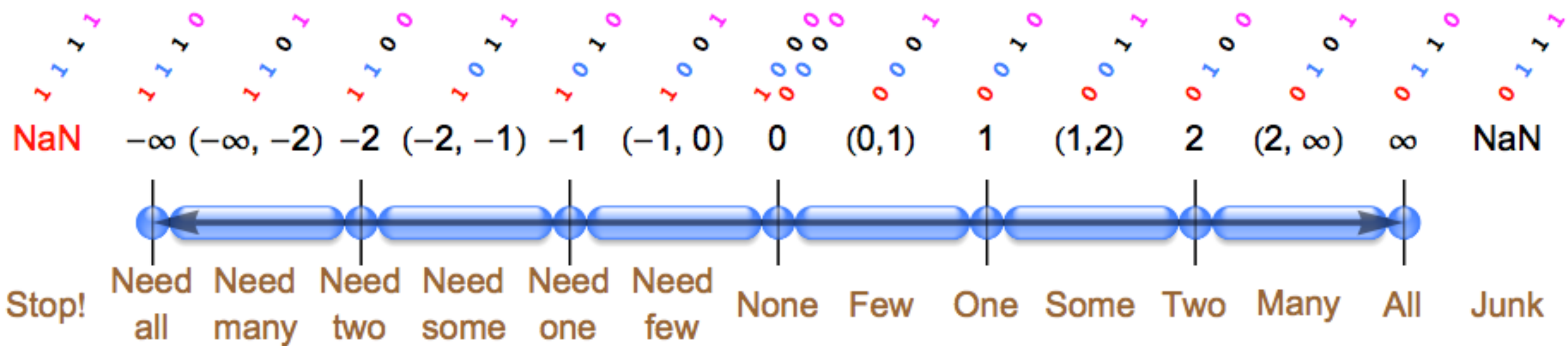
Before the aboriginal Warlpiri of Northern Australia had contact with other civilizations, their counting system was "One, two, many."

Maybe they were onto something.

The Warlpiri unums

Before the aboriginal Warlpiri of Northern Australia had contact with other civilizations, their counting system was "One, two, many."

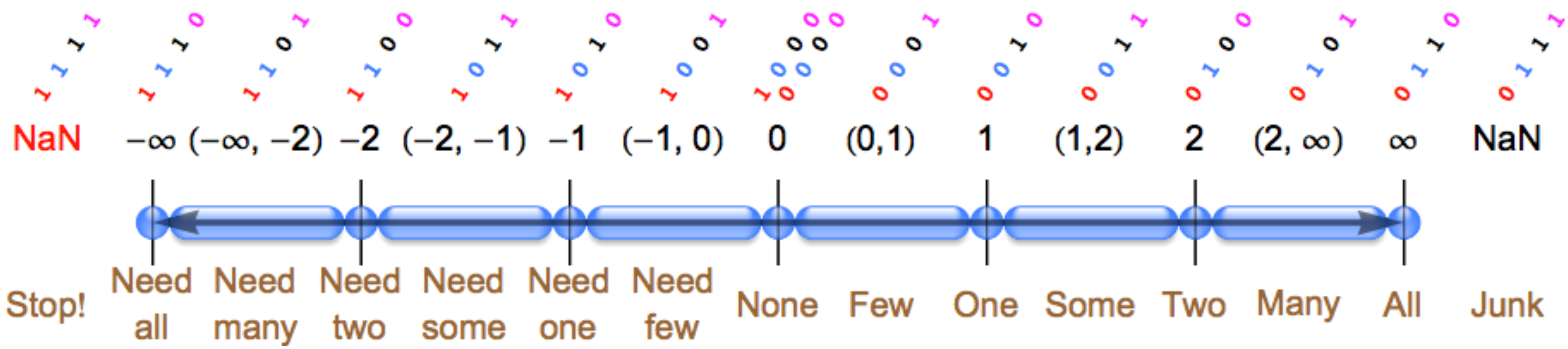
Maybe they were onto something.



The Warlpiri unums

Before the aboriginal Warlpiri of Northern Australia had contact with other civilizations, their counting system was “One, two, many.”

Maybe they were onto something.



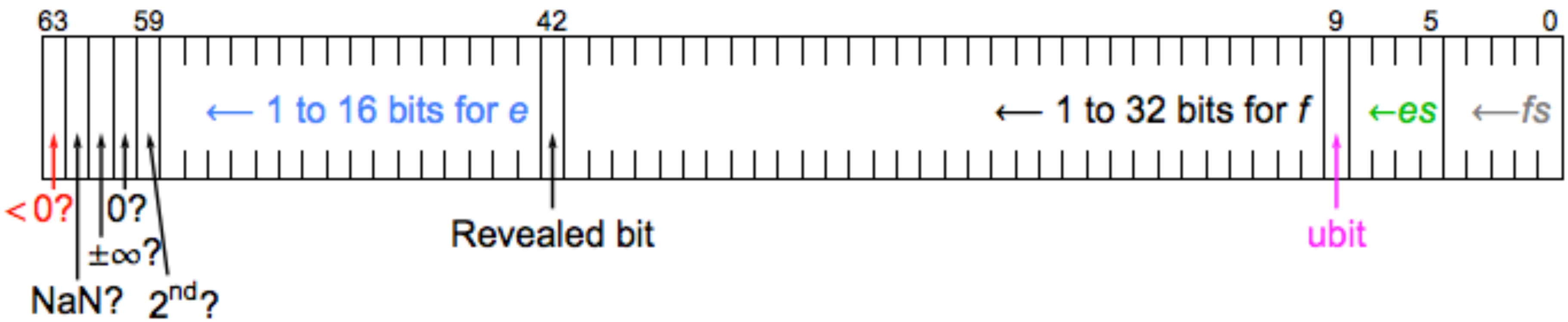
“Try everything” methods become *feasible*.

Fixed-size unums: faster than floats

- Warlpiri ubounds are one byte, but closed system for reals

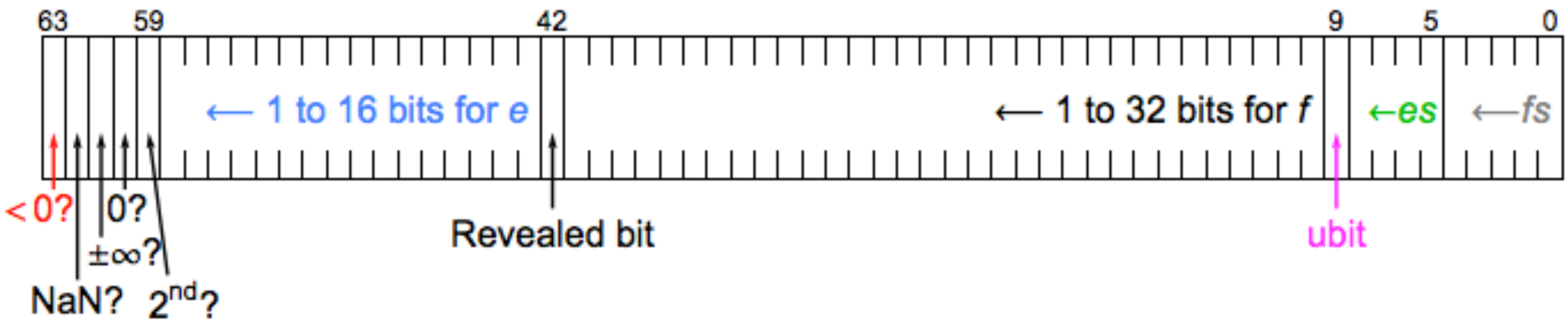
Fixed-size unums: faster than floats

- Warlpiri ubounds are one byte, but closed system for reals
- *Unpacked* unums pre-decode exception bits, hidden bit

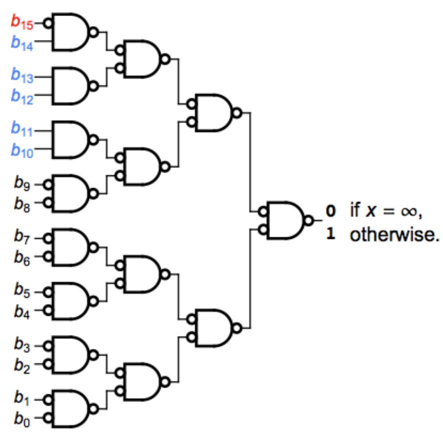


Fixed-size unums: faster than floats

- Warlpiri ubounds are one byte, but closed system for reals
- *Unpacked* unums pre-decode exception bits, hidden bit

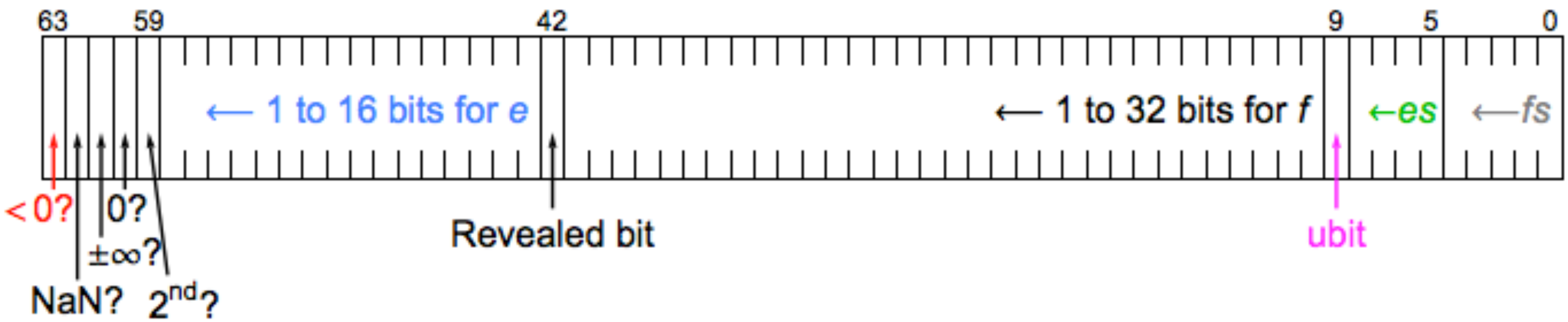


Circuit required for "IEEE half-precision float = ∞?"

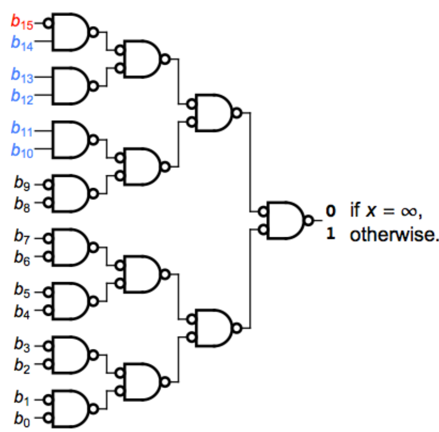


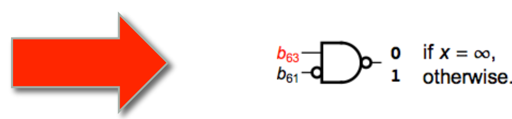
Fixed-size unums: faster than floats

- Warlpiri ubounds are one byte, but closed system for reals
- *Unpacked* unums pre-decode exception bits, hidden bit



Circuit required for "IEEE half-precision float = ∞?"





Circuit required for "unum = ∞?" (any precision)

Floating Point II: The Wrath of Kahan

Floating Point II: The Wrath of Kahan

- Berkeley professor William Kahan is the father of modern IEEE Standard floats

Floating Point II: The Wrath of Kahan

- Berkeley professor William Kahan is the father of modern IEEE Standard floats
- Also the authority on their many dangers

Floating Point II: The Wrath of Kahan

- Berkeley professor William Kahan is the father of modern IEEE Standard floats
- Also the authority on their many dangers
- Every idea to fix floats faces his tests that expose how new idea is *even worse*

Floating Point II: The Wrath of Kahan

- Berkeley professor William Kahan is the father of modern IEEE Standard floats
- Also the authority on their many dangers
- Every idea to fix floats faces his tests that expose how new idea is *even worse*

Working unum environment completed August 13, 2013.

Floating Point II: The Wrath of Kahan

- Berkeley professor William Kahan is the father of modern IEEE Standard floats
- Also the authority on their many dangers
- Every idea to fix floats faces his tests that expose how new idea is *even worse*

Working unum environment completed August 13, 2013.

Can unums survive the wrath of Kahan?

Typical Kahan Challenge (invented by J-M Müller)

"Define functions with: $E(0) = 1$, $E(z) = \frac{e^z - 1}{z}$. $Q[x] = \left| x - \sqrt{x^2 + 1} \right| - \frac{1}{x + \sqrt{x^2+1}}$. $H(x) = E(Q(x)^2)$.

Compute $H(x)$ for $x = 15.0, 16.0, 17.0, 9999.0$. Repeat with more precision, say using BigDecimal."

- Correct answer: (1, 1, 1, 1).

Typical Kahan Challenge (invented by J-M Müller)

"Define functions with: $E(0) = 1$, $E(z) = \frac{e^z - 1}{z}$. $Q[x] = \left| x - \sqrt{x^2 + 1} \right| - \frac{1}{x + \sqrt{x^2+1}}$. $H(x) = E(Q(x)^2)$.

Compute $H(x)$ for x = 15.0, 16.0, 17.0, 9999.0. Repeat with more precision, say using BigDecimal."

- Correct answer: (1, 1, 1, 1).
- IEEE 32-bit: (0, 0, 0, 0) **FAIL**

Typical Kahan Challenge (invented by J-M Müller)

"Define functions with: $E(0) = 1$, $E(z) = \frac{e^z - 1}{z}$. $Q[x] = \left| x - \sqrt{x^2 + 1} \right| - \frac{1}{x + \sqrt{x^2+1}}$. $H(x) = E(Q(x)^2)$.

Compute $H(x)$ for $x = 15.0, 16.0, 17.0, 9999.0$. Repeat with more precision, say using BigDecimal."

- Correct answer: (1, 1, 1, 1).
- IEEE 32-bit: (0, 0, 0, 0) **FAIL**
- IEEE 64-bit: (0, 0, 0, 0) **FAIL**

Typical Kahan Challenge (invented by J-M Müller)

"Define functions with: $E(0) = 1$, $E(z) = \frac{e^z - 1}{z}$. $Q[x] = \left| x - \sqrt{x^2+1} \right| - \frac{1}{x + \sqrt{x^2+1}}$. $H(x) = E(Q(x))^2$).

Compute $H(x)$ for $x = 15.0, 16.0, 17.0, 9999.0$. Repeat with more precision, say using BigDecimal."

- Correct answer: (1, 1, 1, 1).
- IEEE 32-bit: (0, 0, 0, 0) **FAIL**
- IEEE 64-bit: (0, 0, 0, 0) **FAIL**
- Myth: "Getting the same answer with increased precision means the answer is correct."

Typical Kahan Challenge (invented by J-M Müller)

"Define functions with: $E(0) = 1$, $E(z) = \frac{e^z - 1}{z}$. $Q[x] = \left| x - \sqrt{x^2 + 1} \right| - \frac{1}{x + \sqrt{x^2+1}}$. $H(x) = E(Q(x)^2)$.

Compute $H(x)$ for $x = 15.0, 16.0, 17.0, 9999.0$. Repeat with more precision, say using BigDecimal."

- Correct answer: (1, 1, 1, 1).
- IEEE 32-bit: (0, 0, 0, 0) **FAIL**
- IEEE 64-bit: (0, 0, 0, 0) **FAIL**
- Myth: "Getting the same answer with increased precision means the answer is correct."
- IEEE 128-bit: (0, 0, 0, 0) **FAIL**

Typical Kahan Challenge (invented by J-M Müller)

"Define functions with: $E(0) = 1$, $E(z) = \frac{e^z - 1}{z}$. $Q[x] = \left| x - \sqrt{x^2+1} \right| - \frac{1}{x + \sqrt{x^2+1}}$. $H(x) = E(Q(x)^2)$.

Compute $H(x)$ for $x = 15.0, 16.0, 17.0, 9999.0$. Repeat with more precision, say using BigDecimal."

- Correct answer: (1, 1, 1, 1).
- IEEE 32-bit: (0, 0, 0, 0) **FAIL**
- IEEE 64-bit: (0, 0, 0, 0) **FAIL**
- Myth: "Getting the same answer with increased precision means the answer is correct."
- IEEE 128-bit: (0, 0, 0, 0) **FAIL**
- Extended precision math packages: (0, 0, 0, 0) **FAIL**

Typical Kahan Challenge (invented by J-M Müller)

"Define functions with: $E(0) = 1$, $E(z) = \frac{e^z - 1}{z}$. $Q[x] = \left| x - \sqrt{x^2 + 1} \right| - \frac{1}{x + \sqrt{x^2+1}}$. $H(x) = E(Q(x)^2)$.

Compute $H(x)$ for $x = 15.0, 16.0, 17.0, 9999.0$. Repeat with more precision, say using BigDecimal."

- Correct answer: (1, 1, 1, 1).
- IEEE 32-bit: (0, 0, 0, 0) **FAIL**
- IEEE 64-bit: (0, 0, 0, 0) **FAIL**
- Myth: "Getting the same answer with increased precision means the answer is correct."
- IEEE 128-bit: (0, 0, 0, 0) **FAIL**
- Extended precision math packages: (0, 0, 0, 0) **FAIL**
- Interval arithmetic: Um, somewhere between –∞ and ∞.

Typical Kahan Challenge (invented by J-M Müller)

"Define functions with: $E(0) = 1$, $E(z) = \frac{e^z - 1}{z}$. $Q[x] = \left| x - \sqrt{x^2 + 1} \right| - \frac{1}{x + \sqrt{x^2+1}}$. $H(x) = E(Q(x))^2)$.

Compute $H(x)$ for $x = 15.0, 16.0, 17.0, 9999.0$. Repeat with more precision, say using BigDecimal."

- Correct answer: (1, 1, 1, 1).
- IEEE 32-bit: (0, 0, 0, 0) **FAIL**
- IEEE 64-bit: (0, 0, 0, 0) **FAIL**
- Myth: "Getting the same answer with increased precision means the answer is correct."
- IEEE 128-bit: (0, 0, 0, 0) **FAIL**
- Extended precision math packages: (0, 0, 0, 0) **FAIL**
- Interval arithmetic: Um, somewhere between –∞ and ∞. **EPIC FAIL**

Typical Kahan Challenge (invented by J-M Müller)

"Define functions with: $E(0) = 1$, $E(z) = \frac{e^z - 1}{z}$. $Q[x] = \left| x - \sqrt{x^2+1} \right| - \frac{1}{x + \sqrt{x^2+1}}$. $H(x) = E(Q(x))^2)$.

Compute $H(x)$ for $x = 15.0, 16.0, 17.0, 9999.0$. Repeat with more precision, say using BigDecimal."

- Correct answer: (1, 1, 1, 1).
- IEEE 32-bit: (0, 0, 0, 0) **FAIL**
- IEEE 64-bit: (0, 0, 0, 0) **FAIL**
- Myth: "Getting the same answer with increased precision means the answer is correct."
- IEEE 128-bit: (0, 0, 0, 0) **FAIL**
- Extended precision math packages: (0, 0, 0, 0) **FAIL**
- Interval arithmetic: Um, somewhere between –∞ and ∞. **EPIC FAIL**
- Unums, ***6-bit*** average size: (1, 1, 1, 1) **CORRECT**

Typical Kahan Challenge (invented by J-M Müller)

"Define functions with: $E(0) = 1$, $E(z) = \frac{e^z - 1}{z}$. $Q[x] = \left| x - \sqrt{x^2 + 1} \right| - \frac{1}{x + \sqrt{x^2+1}}$. $H(x) = E(Q(x)^2)$.

Compute $H(x)$ for x = 15.0, 16.0, 17.0, 9999.0. Repeat with more precision, say using BigDecimal."

- Correct answer: (1, 1, 1, 1).
- IEEE 32-bit: (0, 0, 0, 0) **FAIL**
- IEEE 64-bit: (0, 0, 0, 0) **FAIL**
- Myth: "Getting the same answer with increased precision means the answer is correct."
- IEEE 128-bit: (0, 0, 0, 0) **FAIL**
- Extended precision math packages: (0, 0, 0, 0) **FAIL**
- Interval arithmetic: Um, somewhere between $-\infty$ and ∞. **EPIC FAIL**
- Unums, ***6-bit*** average size: (1, 1, 1, 1) **CORRECT**

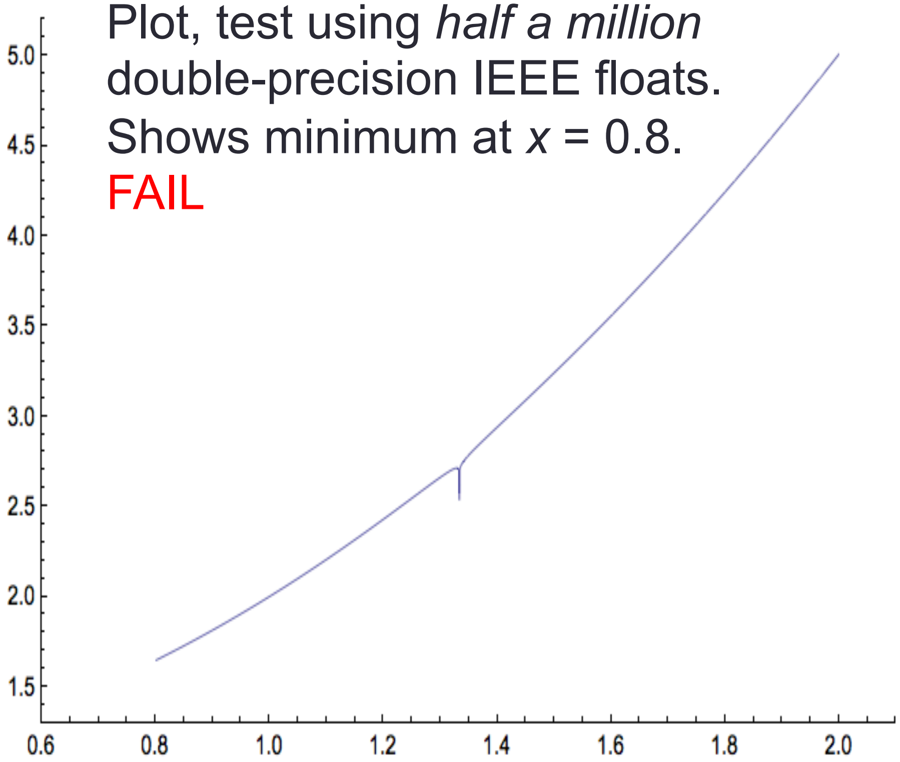
I have been unable to find a problem that "breaks" unum math.

Kahan's "Smooth Surprise"

Find minimum of $\log(|3(1-x)+1|)/80 + x^2 + 1$ in $0.8 \le x \le 2.0$

Kahan's "Smooth Surprise"

Find minimum of $\log(|3(1-x)+1|)/80 + x^2 + 1$ in $0.8 \le x \le 2.0$

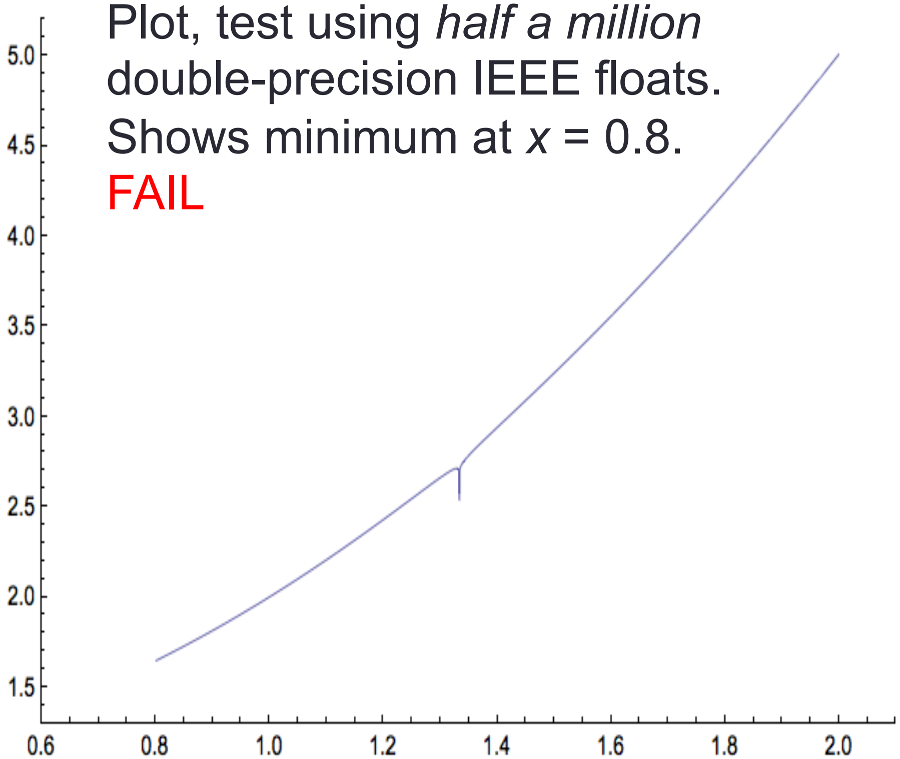


Plot, test using *half a million* double-precision IEEE floats. Shows minimum at $x = 0.8$.

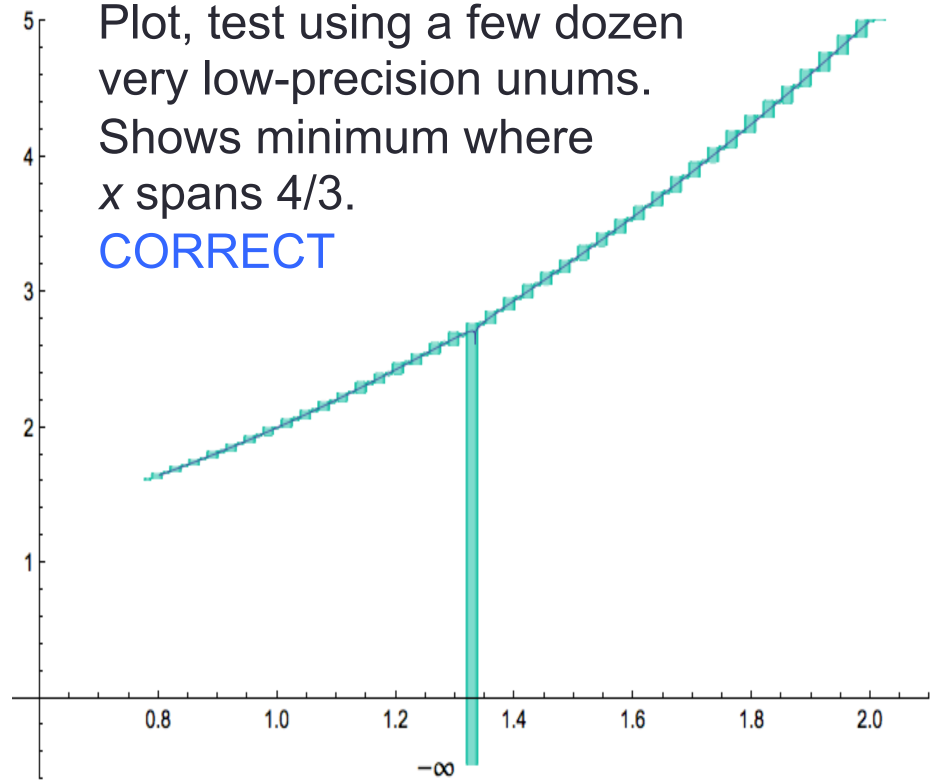
FAIL

Kahan's "Smooth Surprise"

Find minimum of $\log(|3(1-x)+1|)/80 + x^2 + 1$ in $0.8 \leq x \leq 2.0$



Plot, test using *half a million* double-precision IEEE floats. Shows minimum at $x = 0.8$.
FAIL



Plot, test using a few dozen very low-precision unums. Shows minimum where x spans 4/3.
CORRECT

Rump's Royal Pain

Compute $333.75y^6 + x^2(11x^2y^2 - y^6 - 121y^4 - 2) + 5.5y^8 + x/(2y)$
where $x = 77617$, $y = 33096$.

Rump's Royal Pain

Compute $333.75y^6 + x^2(11x^2y^2 - y^6 - 121y^4 - 2) + 5.5y^8 + x/(2y)$
where $x = 77617$, $y = 33096$.

- Using IBM (pre-IEEE Standard) floats, Rump got
  - 1.172603 in 32-bit precision

Rump's Royal Pain

Compute $333.75y^6 + x^2(11x^2y^2 - y^6 - 121y^4 - 2) + 5.5y^8 + x/(2y)$ where $x = 77617$, $y = 33096$.

- Using IBM (pre-IEEE Standard) floats, Rump got
  - 1.172603 in 32-bit precision
  - 1.1726039400531 in 64-bit precision

Rump's Royal Pain

Compute $333.75y^6 + x^2(11x^2y^2 - y^6 - 121y^4 - 2) + 5.5y^8 + x/(2y)$
where $x = 77617$, $y = 33096$.

- Using IBM (pre-IEEE Standard) floats, Rump got
  - 1.172603 in 32-bit precision
  - 1.1726039400531 in 64-bit precision
  - 1.172603940053178 in 128-bit precision

Rump's Royal Pain

Compute $333.75y^6 + x^2(11x^2y^2 - y^6 - 121y^4 - 2) + 5.5y^8 + x/(2y)$
where $x = 77617$, $y = 33096$.

- Using IBM (pre-IEEE Standard) floats, Rump got
  - 1.172603 in 32-bit precision
  - 1.1726039400531 in 64-bit precision
  - 1.172603940053178 in 128-bit precision
- Using IEEE double precision: 1.18059×10^{21}

Rump's Royal Pain

Compute $333.75y^6 + x^2(11x^2y^2 - y^6 - 121y^4 - 2) + 5.5y^8 + x/(2y)$
where $x = 77617$, $y = 33096$.

- Using IBM (pre-IEEE Standard) floats, Rump got
  - 1.172603 in 32-bit precision
  - 1.1726039400531 in 64-bit precision
  - 1.172603940053178 in 128-bit precision
- Using IEEE double precision: 1.18059×10^{21}
- **Correct answer: –0.827396059946821368…!**
  Didn't even get *sign* right

Rump's Royal Pain

Compute $333.75y^6 + x^2(11x^2y^2 - y^6 - 121y^4 - 2) + 5.5y^8 + x/(2y)$
where $x = 77617$, $y = 33096$.

- Using IBM (pre-IEEE Standard) floats, Rump got
  - 1.172603 in 32-bit precision
  - 1.1726039400531 in 64-bit precision
  - 1.172603940053178 in 128-bit precision
- Using IEEE double precision: 1.18059×10^{21}
- **Correct answer: –0.827396059946821 36⋯!**

  Didn't even get *sign* right

Unums: **Correct answer** to 23 decimals using an average of only ***75*** bits per number. Not even IEEE 128-bit precision can do that. Precision, range adjust *automatically*.

Some principles of unum math

Bound the answer as tightly as possible within the numerical environment, or admit defeat.

Some principles of unum math

Bound the answer as tightly as possible within the numerical environment, or admit defeat.

- No more guessing

Some principles of unum math

Bound the answer as tightly as possible within the numerical environment, or admit defeat.

- No more guessing
- No more “the error is $O(h^n)$” type estimates

Some principles of unum math

Bound the answer as tightly as possible within the numerical environment, or admit defeat.

- No more guessing
- No more "the error is $O(h^n)$" type estimates
- The smaller the bound, the greater the information

Some principles of unum math

Bound the answer as tightly as possible within the numerical environment, or admit defeat.

- No more guessing
- No more “the error is $O(h^n)$” type estimates
- The smaller the bound, the greater the information
- Performance is *information per second*, not ops per second

Some principles of unum math

Bound the answer as tightly as possible within the numerical environment, or admit defeat.

- No more guessing
- No more "the error is $O(h^n)$" type estimates
- The smaller the bound, the greater the information
- Performance is *information per second*, not ops per second
- Maximize information per second, per bit, per watt,…

Some principles of unum math

Bound the answer as tightly as possible within the numerical environment, or admit defeat.

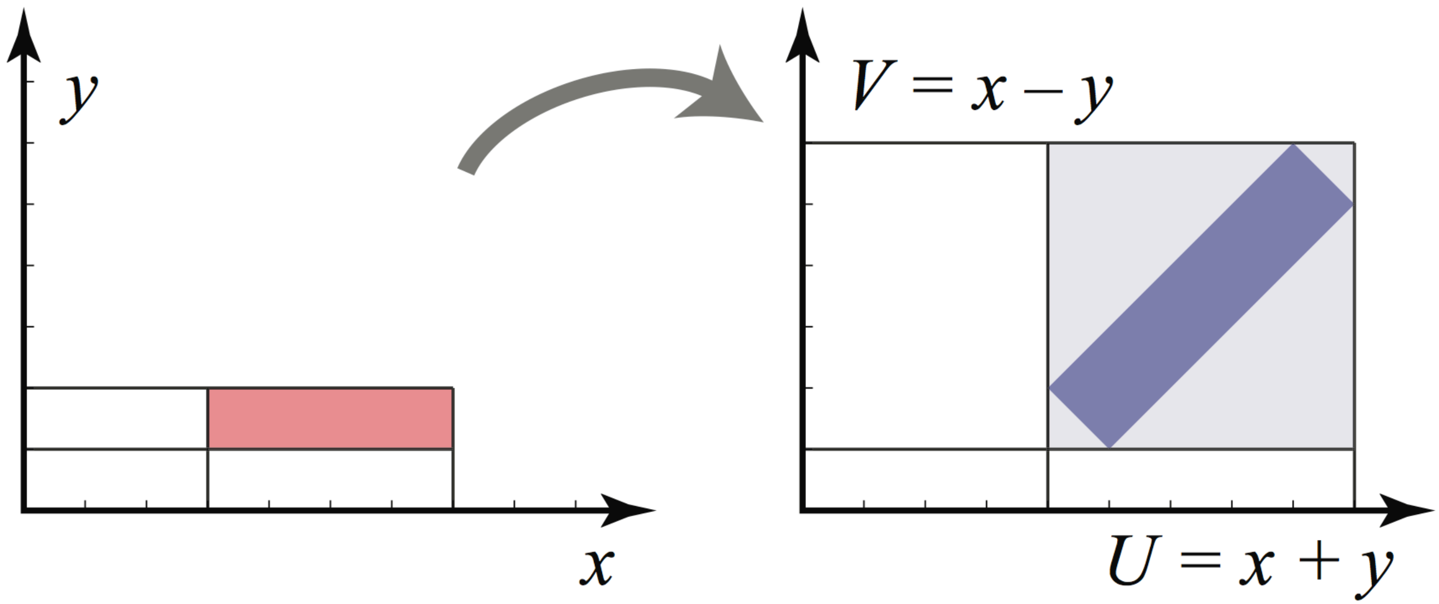
- No more guessing
- No more "the error is $O(h^n)$" type estimates
- The smaller the bound, the greater the information
- Performance is *information per second*, not ops per second
- Maximize information per second, per bit, per watt,…
- Fused operations are always distinct from non-fused, insuring **bitwise identical** results across platforms

Some principles of unum math

Bound the answer as tightly as possible within the numerical environment, or admit defeat.

- No more guessing
- No more "the error is $O(h^n)$" type estimates
- The smaller the bound, the greater the information
- Performance is *information per second*, not ops per second
- Maximize information per second, per bit, per watt,…
- Fused operations are always distinct from non-fused, insuring **bitwise identical** results across platforms
- Computer bears primary numerical analysis burden

Reason 1 why interval math hasn't displaced floats: The "Wrapping Problem"



Answer sets are complex shapes in general, but interval bounds are axis-aligned boxes, period.

No wonder interval bounds grow far too fast to be useful, in general!

Reason 2: The Dependency Problem

What wrecks interval arithmetic is simple things like

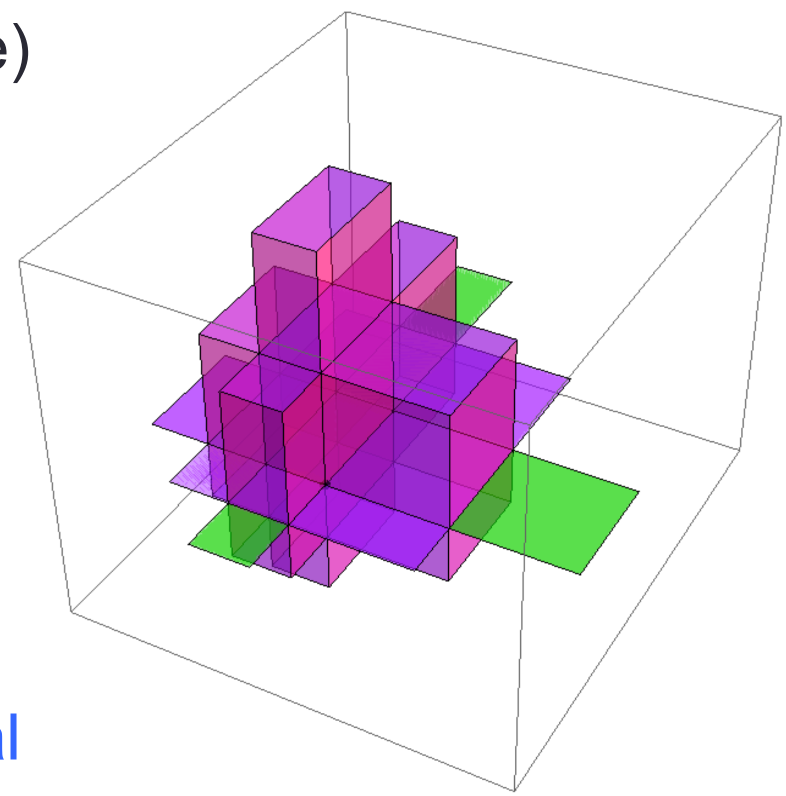
$$F(x) = x - x.$$

Should be 0, or maybe $[-\varepsilon, +\varepsilon]$. Say x is the interval [3, 4], then interval $x - x$ stupidly evaluates to [–1, +1], which doubles the uncertainty (interval width) and makes the interval solution far inferior to the point arithmetic method.

The unum architecture solves both drawbacks of traditional interval arithmetic.

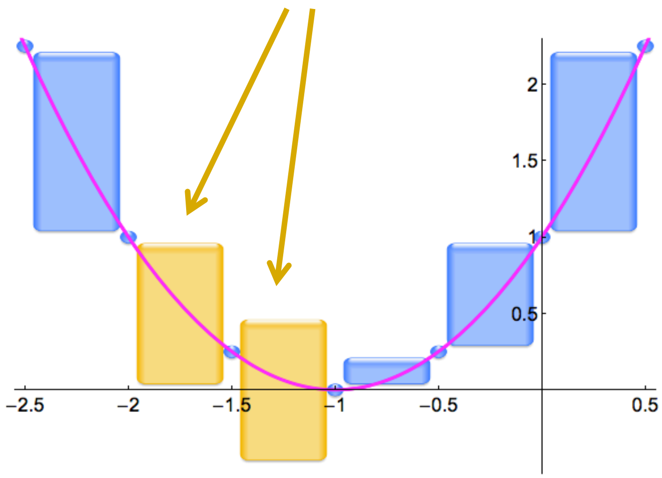
Uboxes and solution sets

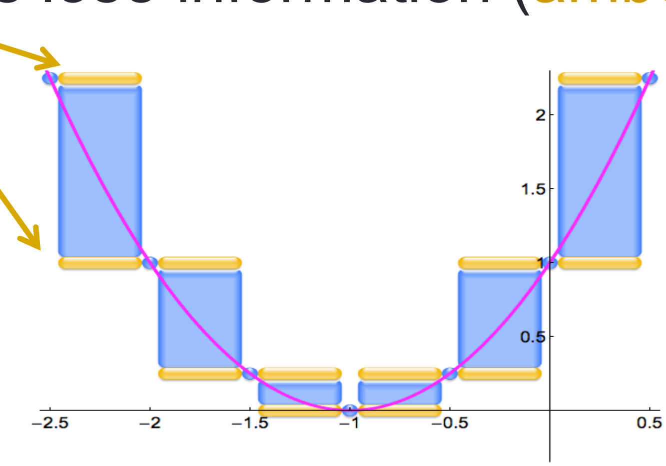
- A *ubox* is a multidimensional unum
- Exact or ULP-wide in each dimension (Unit in the Last Place)
- Sets of uboxes constitute a *solution set*
- One dimension per degree of freedom in solution
- Solves the main problems with interval arithmetic
- Super-economical for bit storage
- Massively data parallel in general



Polynomials: bane of classic intervals

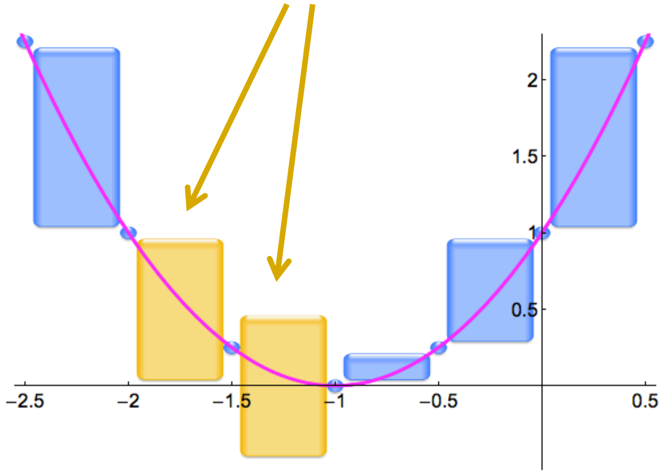
Dependency and closed endpoints lose information (amber)

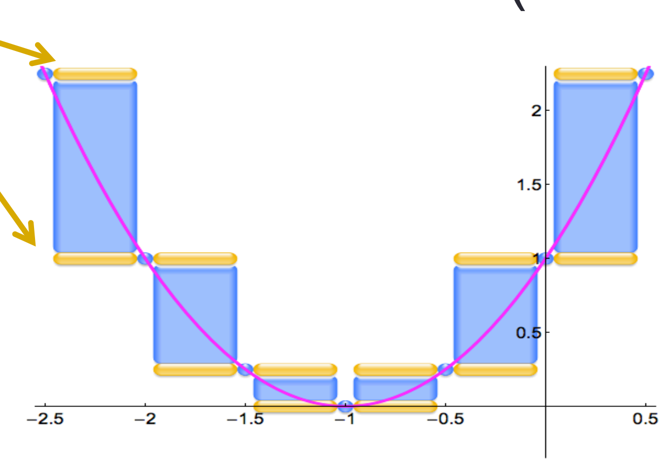


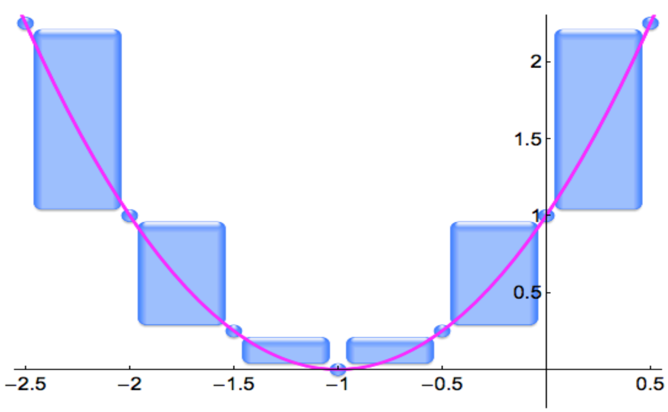


Polynomials: bane of classic intervals

Dependency and closed endpoints lose information (amber)



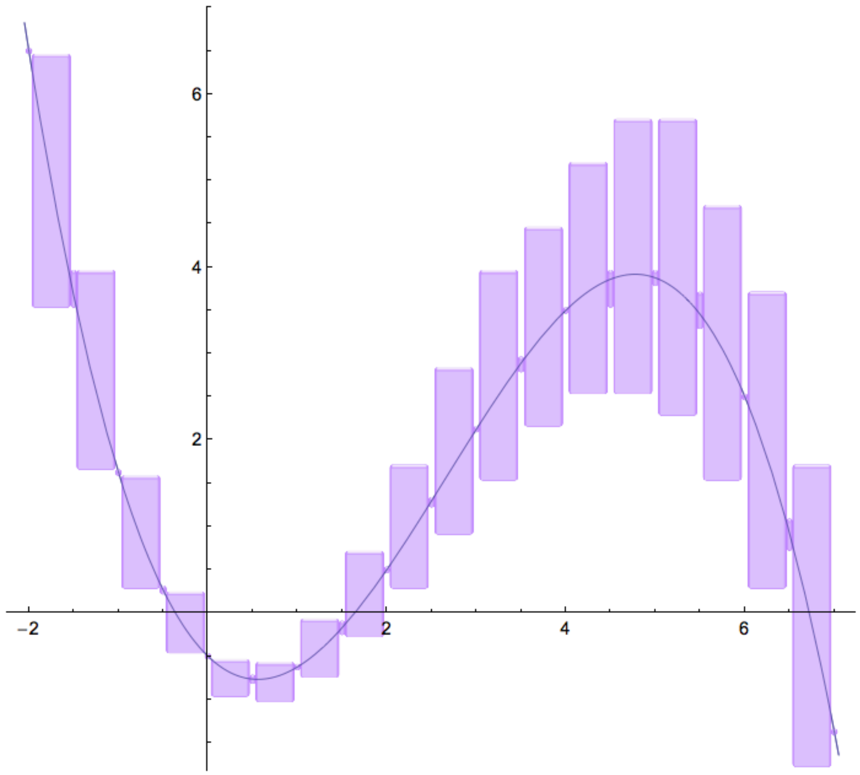




Unum polynomial evaluator loses *no* information.

Polynomial evaluation solved at last

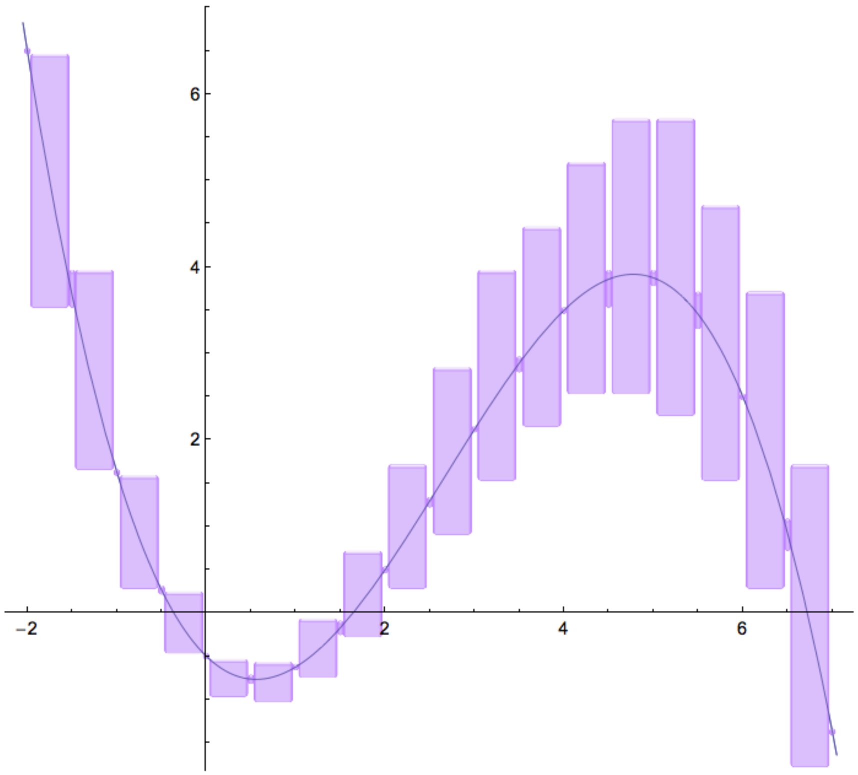
Mathematicians have sought this for at least 60 years.



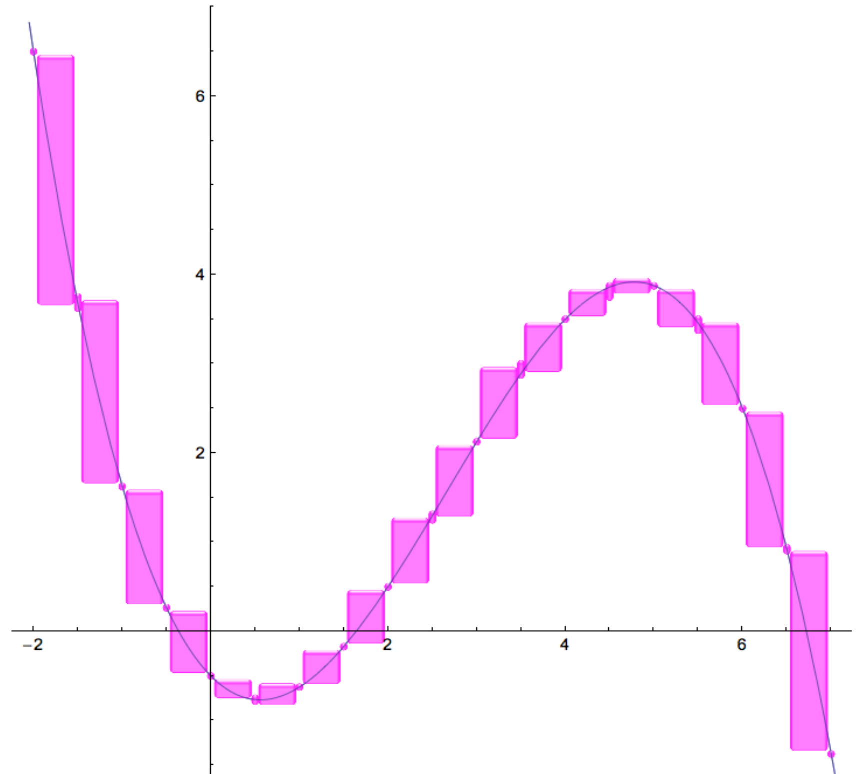
“Dependency Problem” creates sloppy range when input is an interval

Polynomial evaluation solved at last

Mathematicians have sought this for at least 60 years.



“Dependency Problem” creates sloppy range when input is an interval

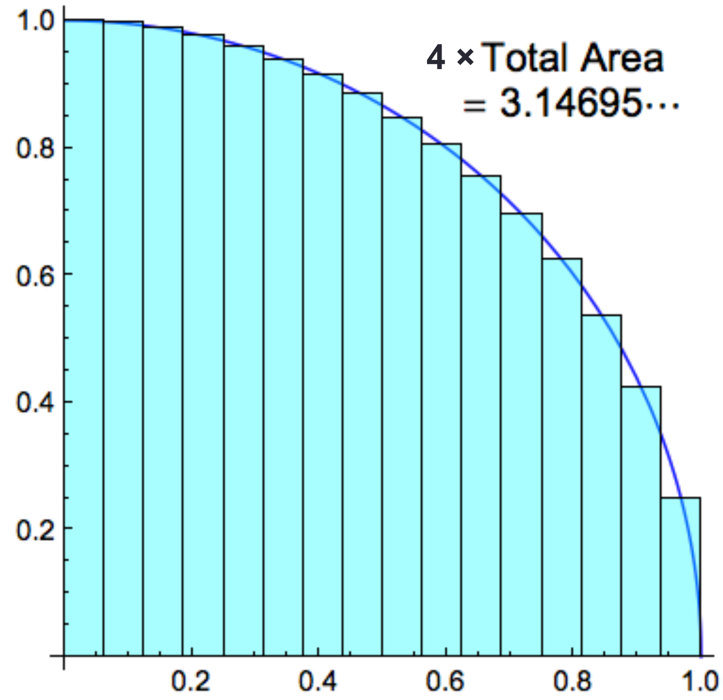


Unum evaluation refines answer to limits of the environment precision

The Deeply Unsatisfying Error Bounds of Classical Analysis

- Classical numerical texts teach this "error bound":

$$\text{Error} \le (b - a)\, h^2\, |f''(\xi)| / 24$$

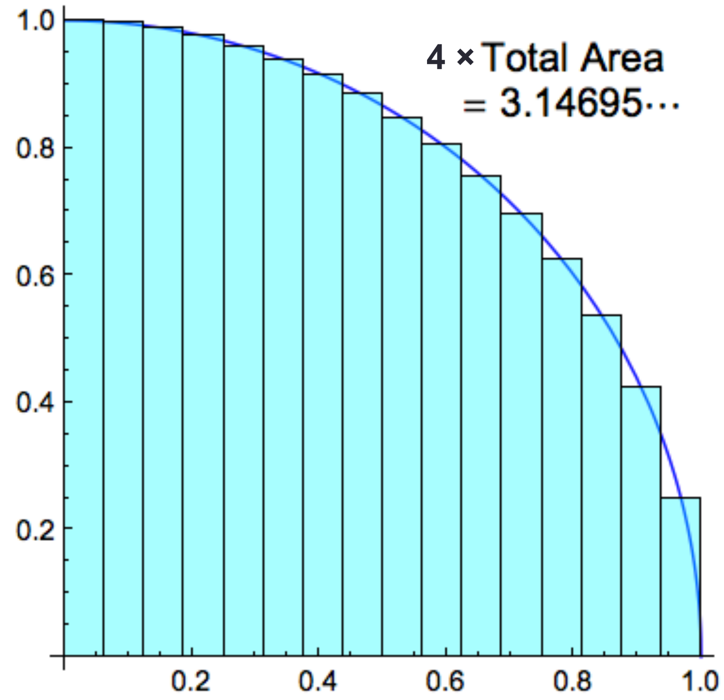


The Deeply Unsatisfying Error Bounds of Classical Analysis

- Classical numerical texts teach this "error bound":

$$\text{Error} \le (b - a)\, h^2\, |f''(\xi)| / 24$$

- What is f''? Where is ξ? What is the bound??

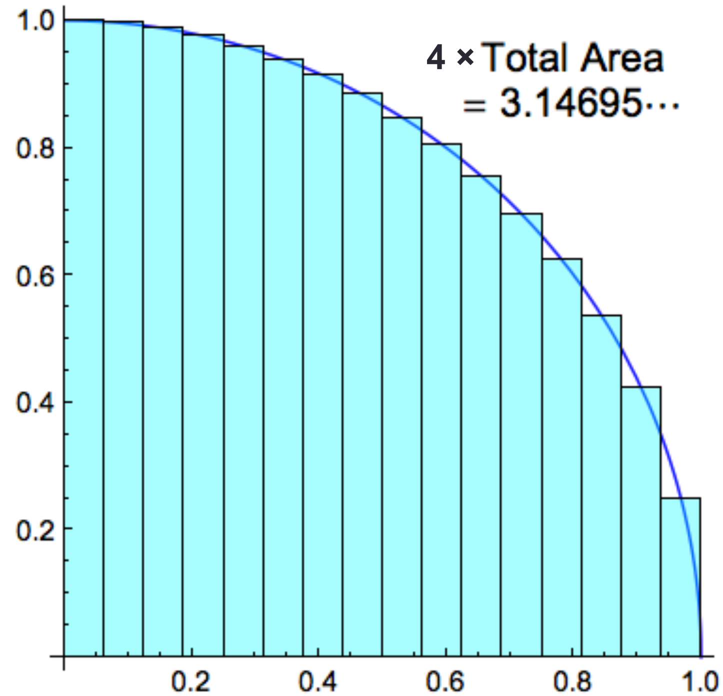


The Deeply Unsatisfying Error Bounds of Classical Analysis

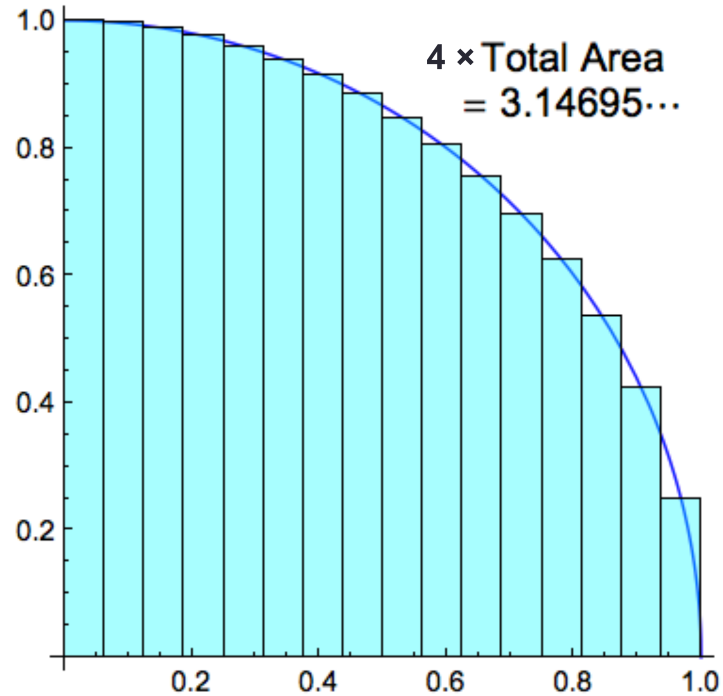
- Classical numerical texts teach this "error bound":

$$\text{Error} \leq (b - a)\, h^2\, |f''(\xi)| \,/\, 24$$

- What is f''? Where is ξ? What is the bound??
- Bound is often *infinite*, which means no bound at all



The Deeply Unsatisfying Error Bounds of Classical Analysis



- Classical numerical texts teach this "error bound":

$$\text{Error} \leq (b - a)\, h^2\, |f''(\xi)| \,/\, 24$$

- What is f''? Where is ξ? What is the bound??
- Bound is often *infinite*, which means no bound at all
- "Whatever it is, it's four times better if we make *h* half as big" creates supercomputing demand that *can never be satisfied*.

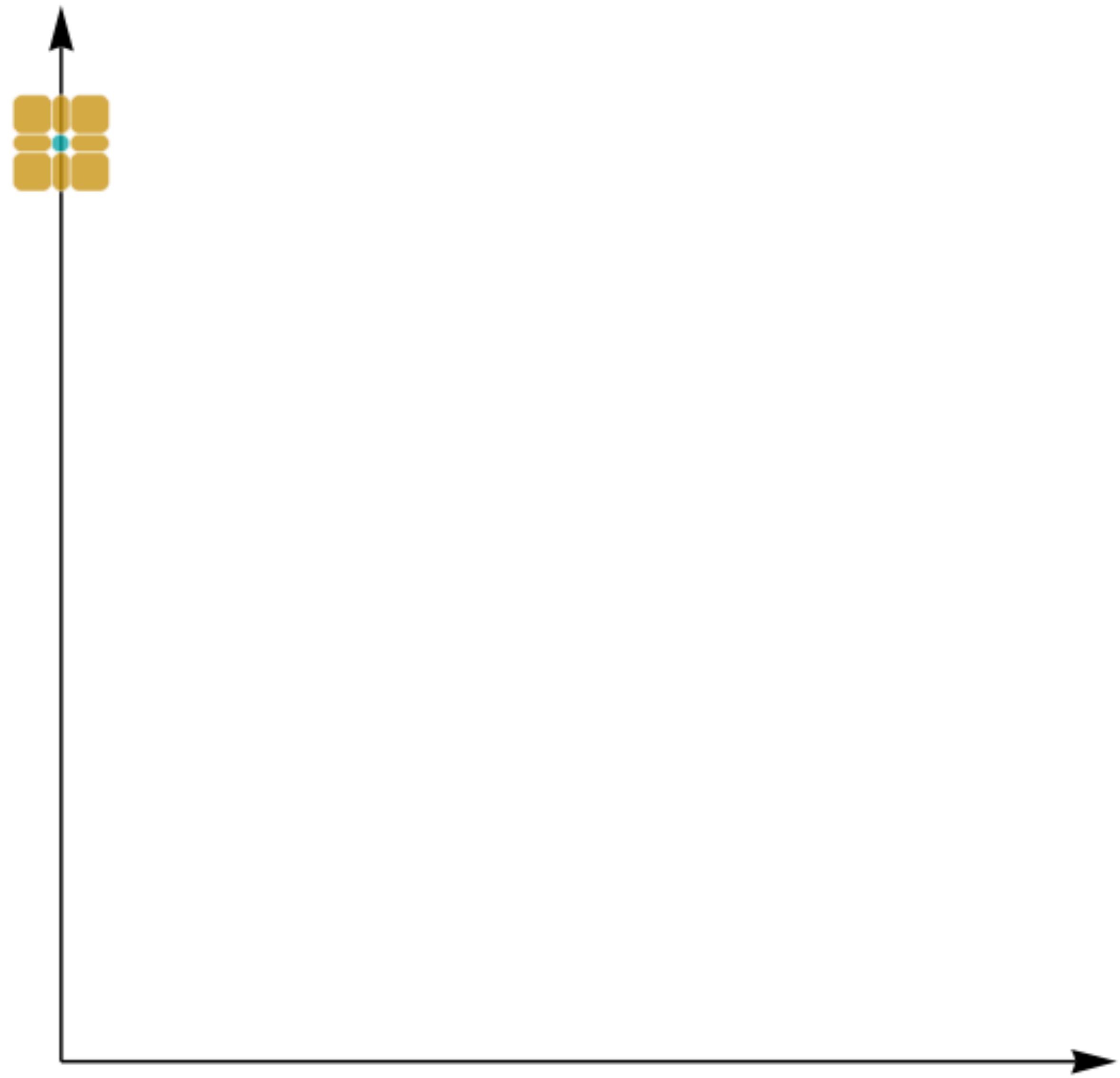
Quarter-circle example

- Suppose all we know is $x^2 + y^2 = 1$, and x and y are ≥ 0.
- Suppose we have at most ***2*** bits exponent, ***4*** bits fraction.

Task:
Bound the quarter circle area.
(i.e., bound the value of $\pi/4$)
Create the pixels for the shape.

Set is connected; need a seed

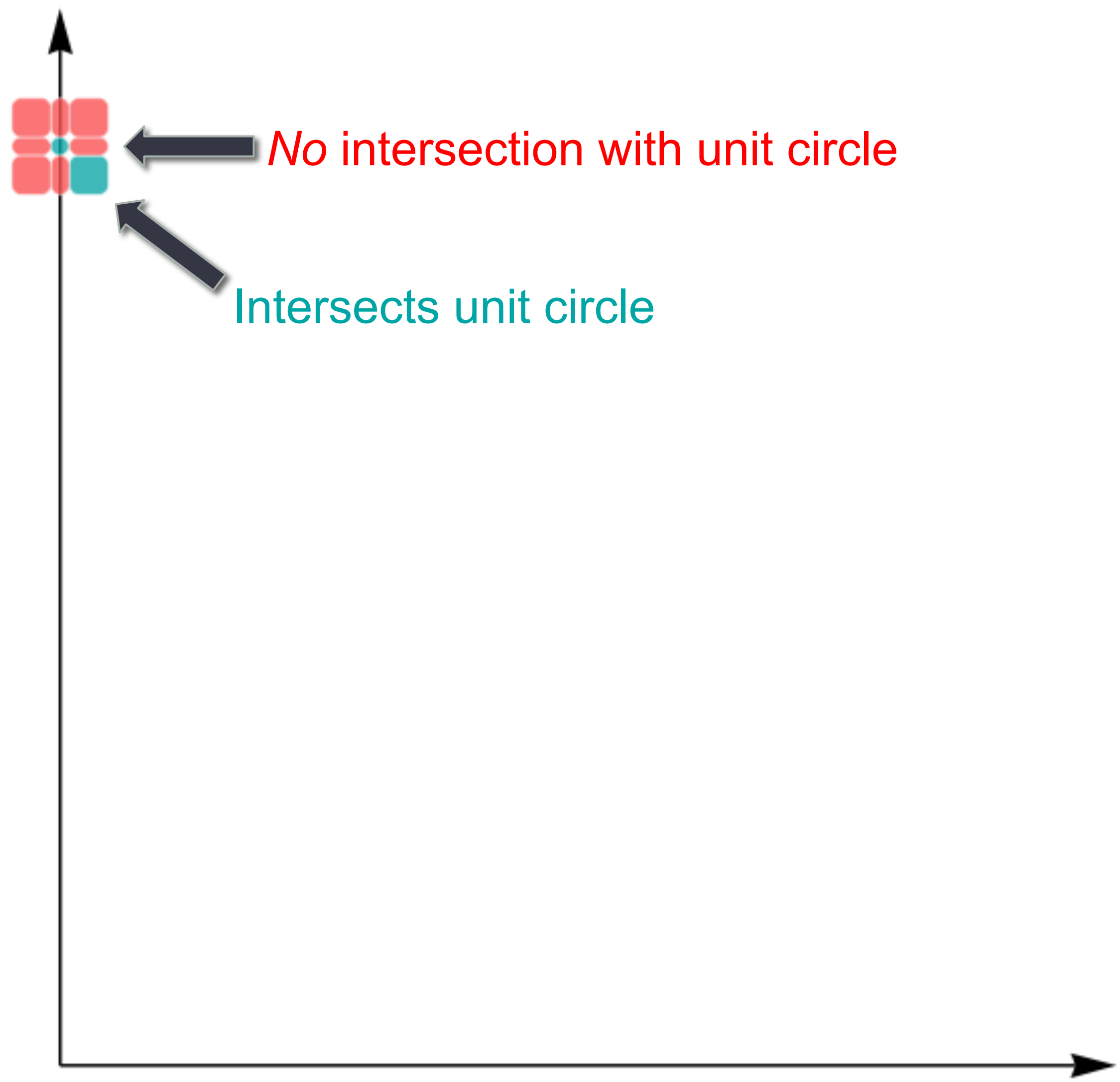
- We know $x = 0$, $y = 1$ works
- Find its 8 ubox neighbors in the plane
- Test $x^2 + y^2 = 1$, $x \geq 0$, $y \geq 0$
- Solution set is green
- Trial set is amber
- Failure set is red
- Stop when no more trials



Exactly one neighbor passes test

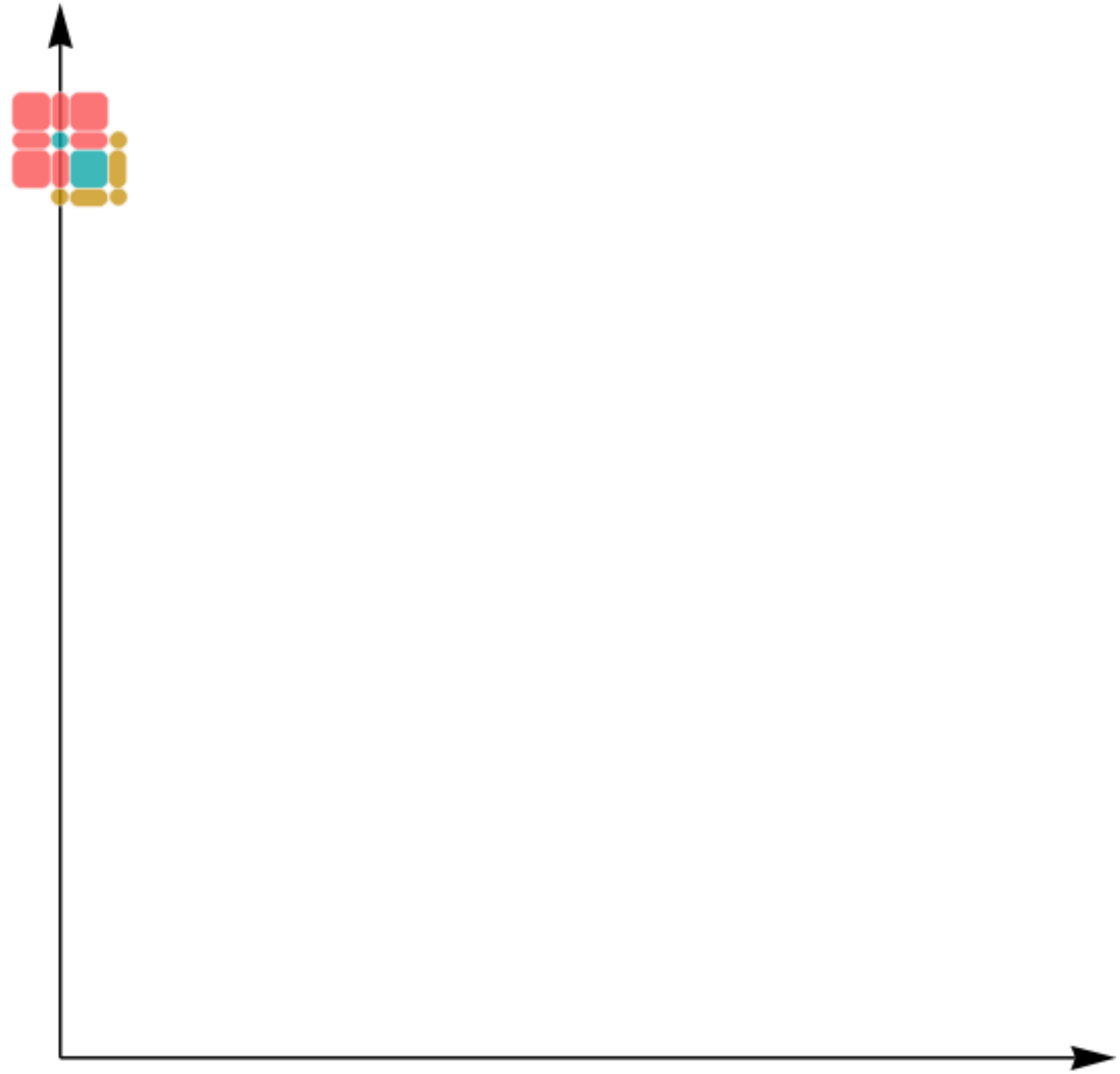
- Unum math automatically *excludes cases that floats would accept*
- Trials are neighbors of new solutions that
  - Are not already failures
  - Are not already solutions
- Note: *no* calculation of

$$y = \sqrt{1 - x^2}$$



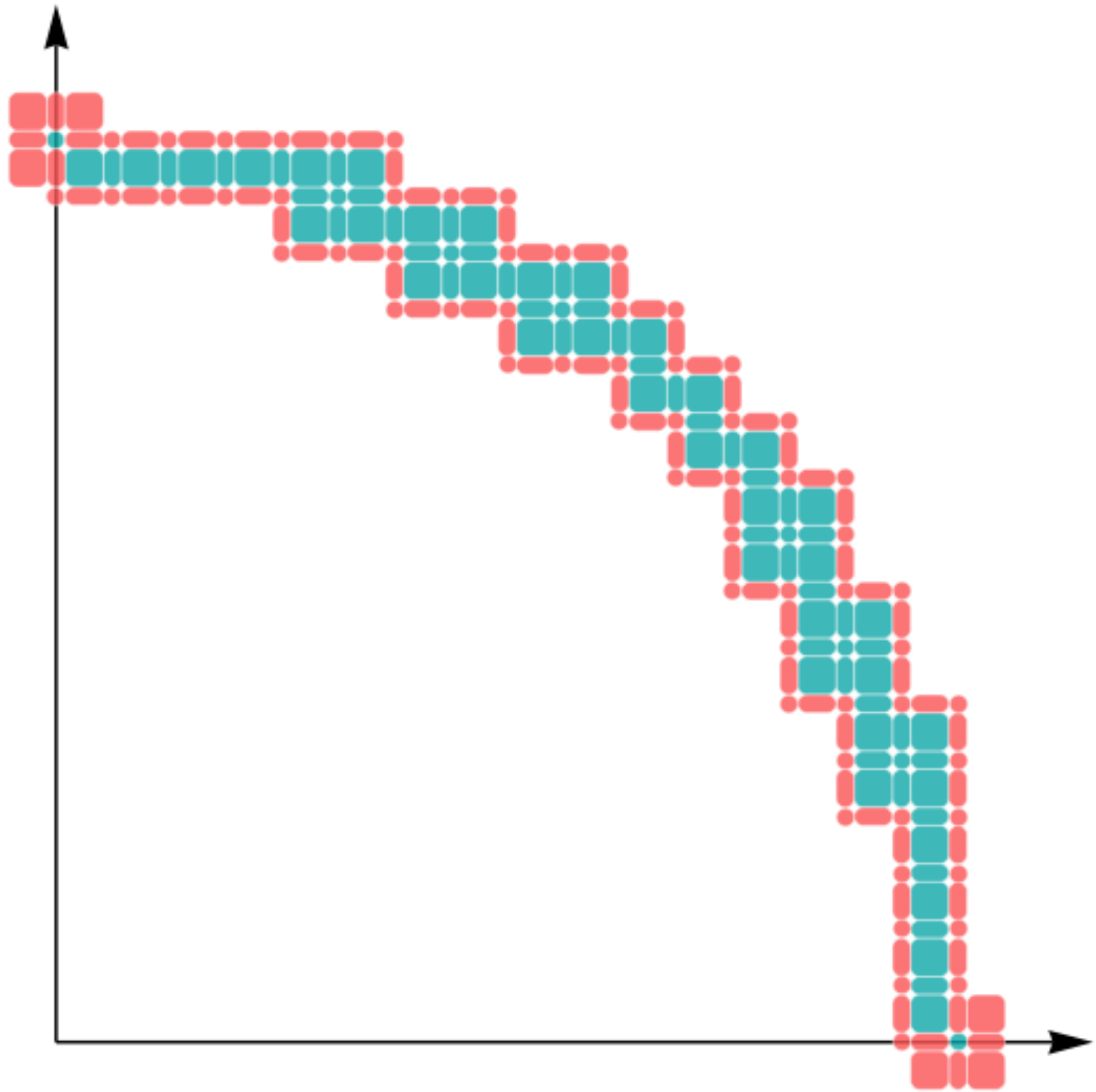
The new trial set

- Five trial uboxes to test
- Perfect, easy parallelism for multicore
- Each ubox takes only 15 to 23 bits
- Ultra-fast operations
- Skip to the final result…



The complete quarter circle

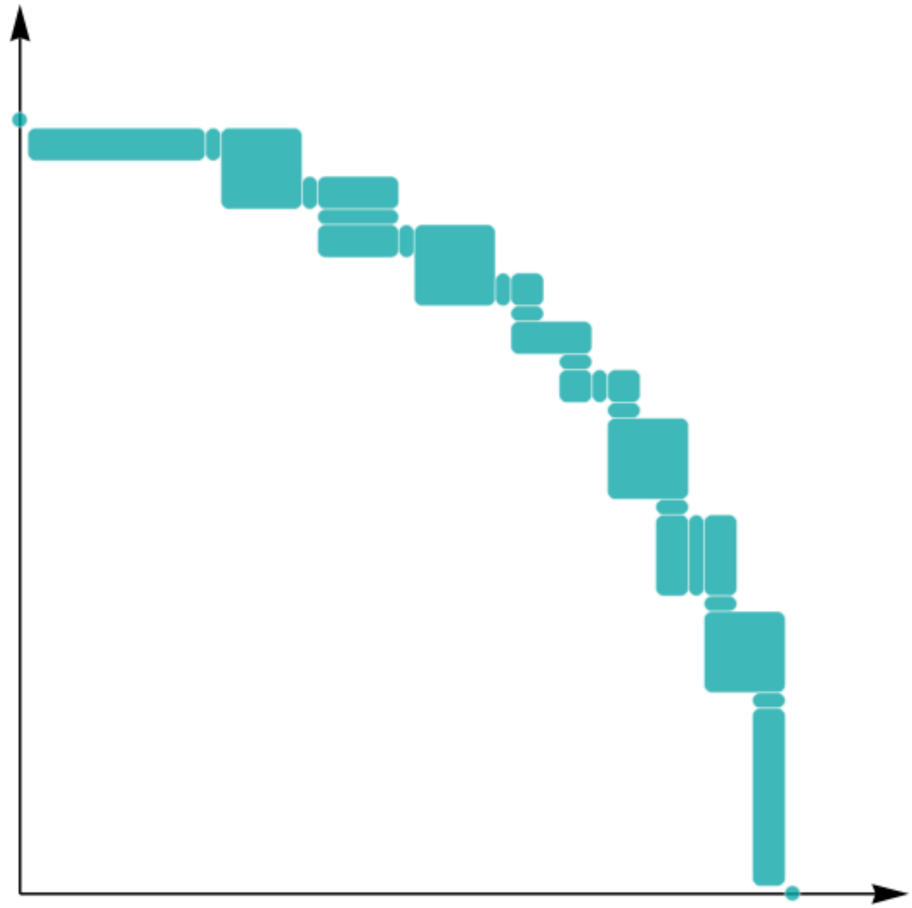
- Complete *solution*, to this finite precision
- *Information* = 1 / green area
- *Proves* value of π to 3% accuracy
- No calculus, no divides, and no square roots



Compressed Final Result

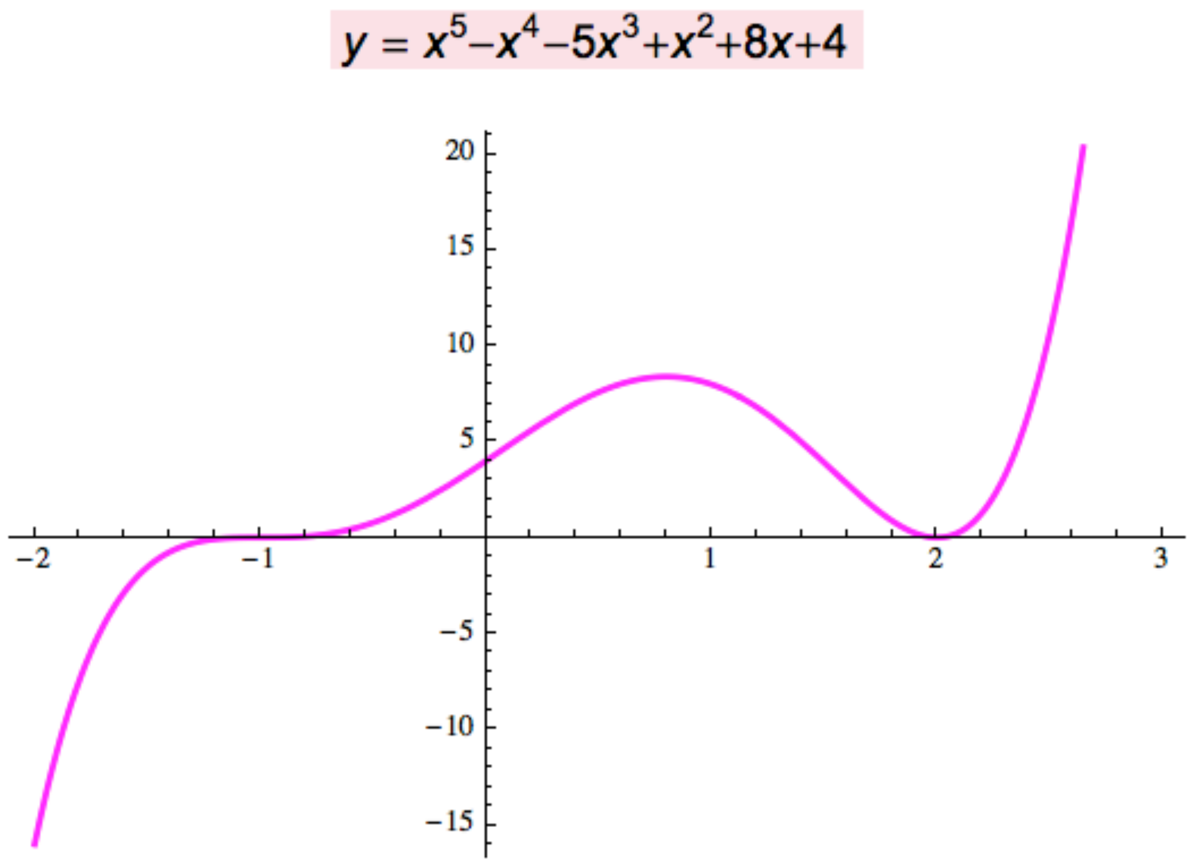
- Coalesce uboxes to largest possible ULP values
- *Lossless* compression
- Total data set: 603 bits!
- 6x faster graphics than current methods

Instead of ULPs being the source of error, they are the *atomic units of computation*



Fifth-degree polynomial roots

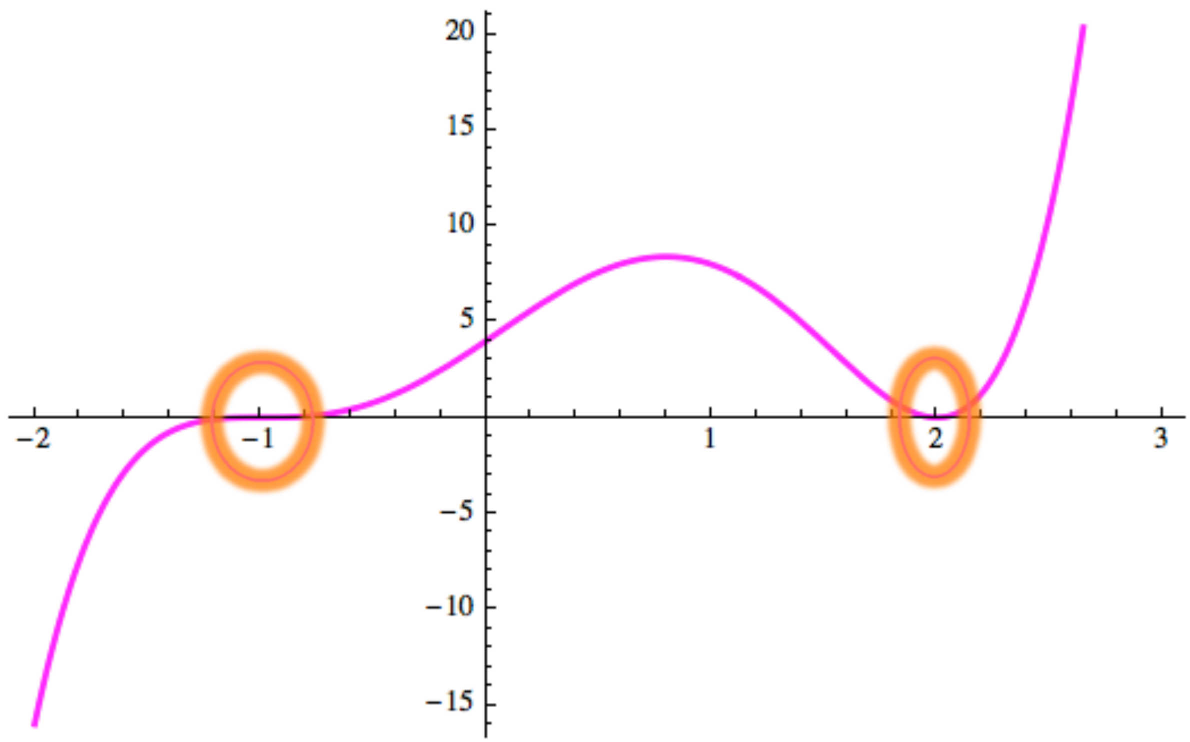
- Analytic solution: There isn't one.



Fifth-degree polynomial roots

- Analytic solution: There isn't one.
- Numerical solution: Huge errors from *underflow to zero*

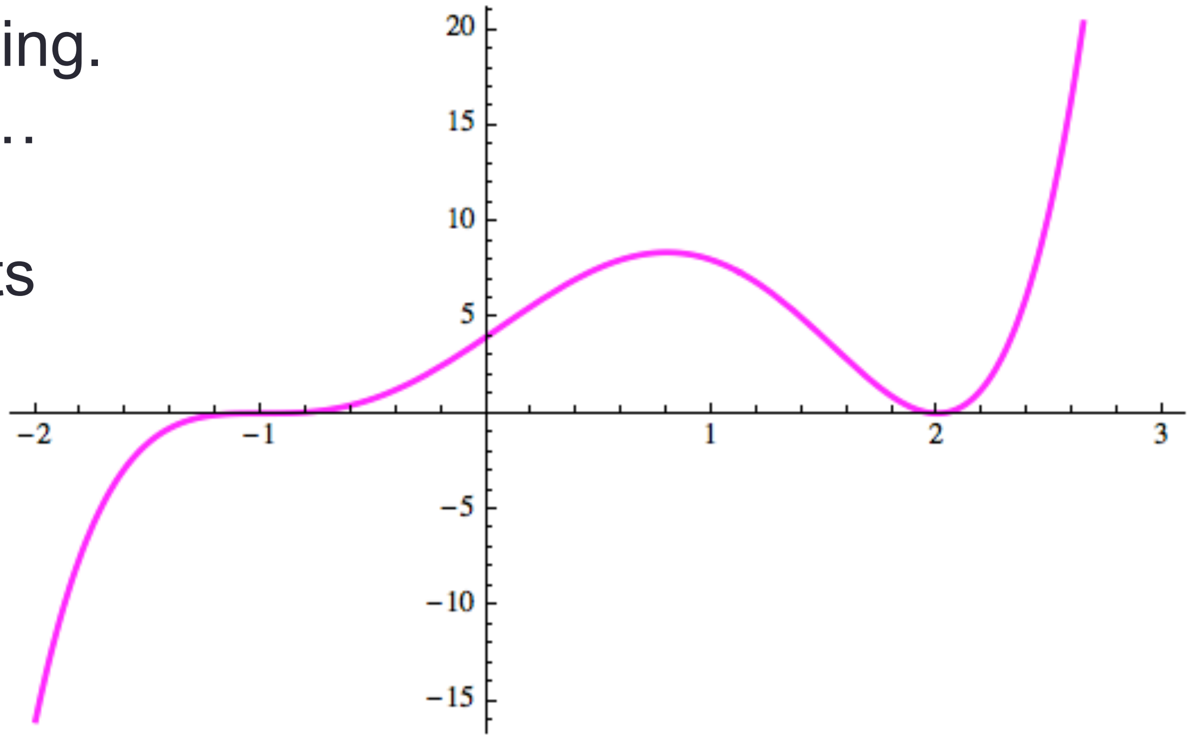
$$y = x^5 - x^4 - 5x^3 + x^2 + 8x + 4$$



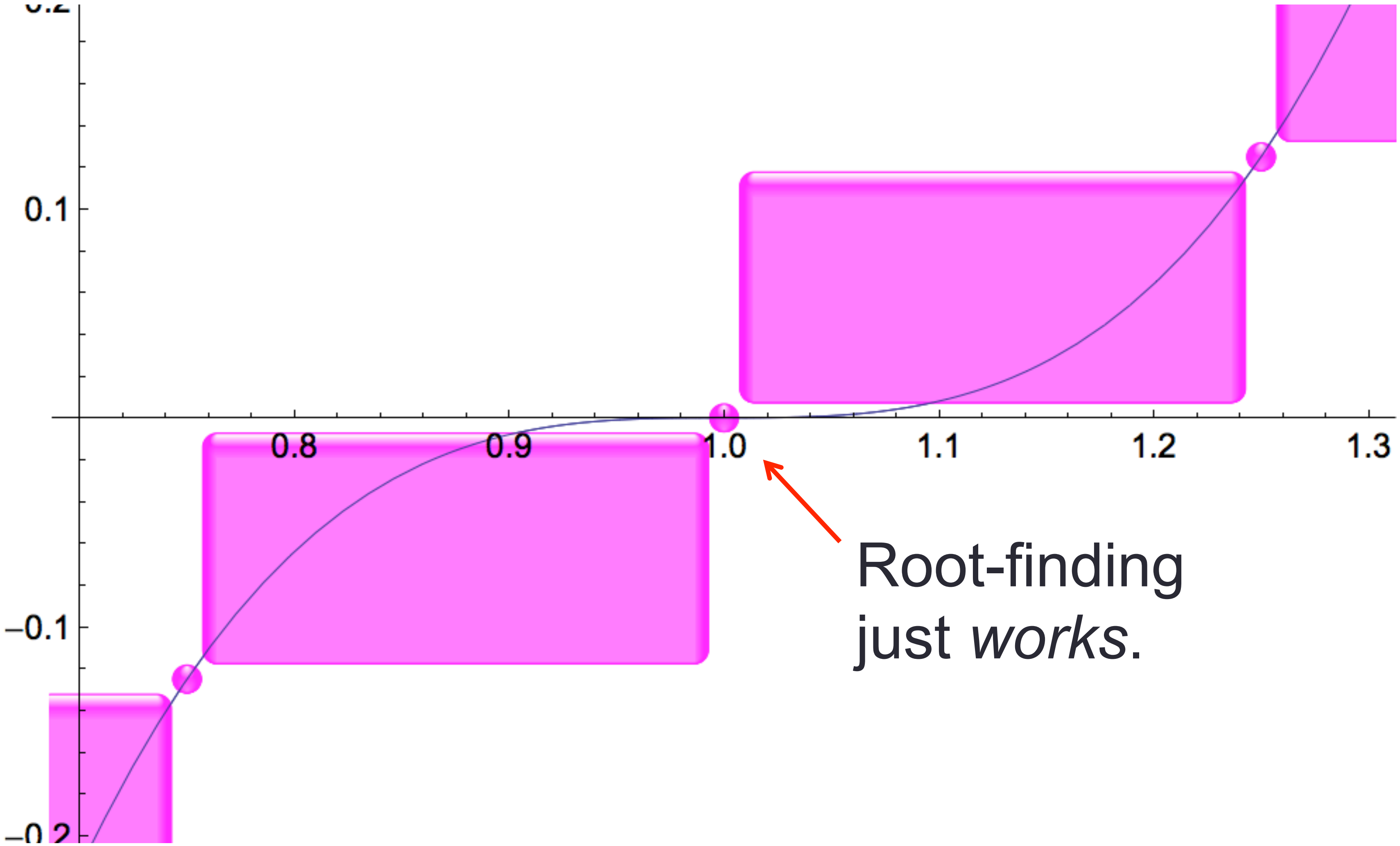
Fifth-degree polynomial roots

- Analytic solution: There isn't one.
- Numerical solution: Huge errors from *underflow to zero*
- Unums: quickly return $x = -1$, $x = 2$ as the exact solutions. No rounding. No underflow. Just… the *correct answer*. With as few as 4 bits for the operands!

$$y = x^5 - x^4 - 5x^3 + x^2 + 8x + 4$$



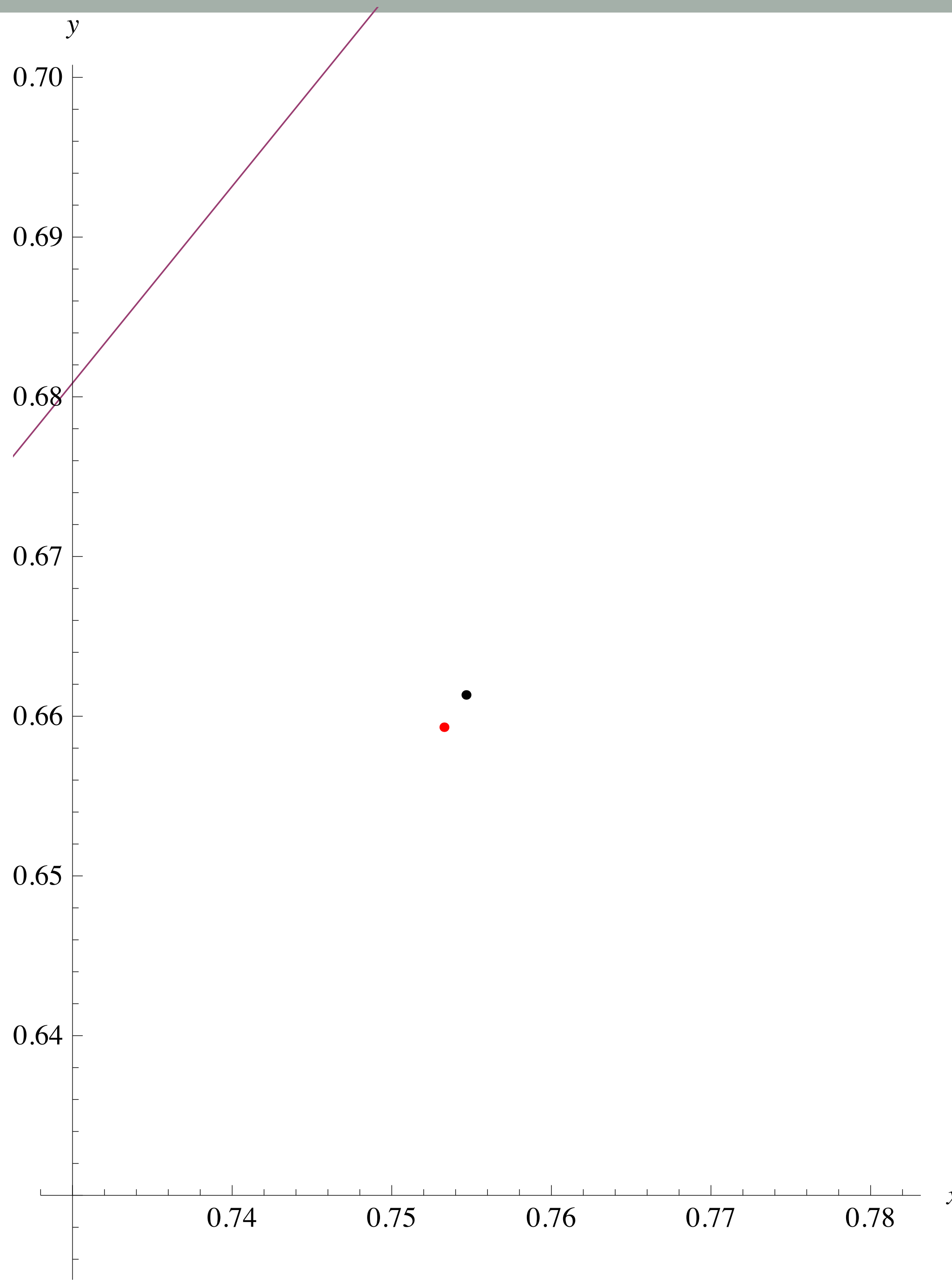
The power of open-closed endpoints



Root-finding just *works*.

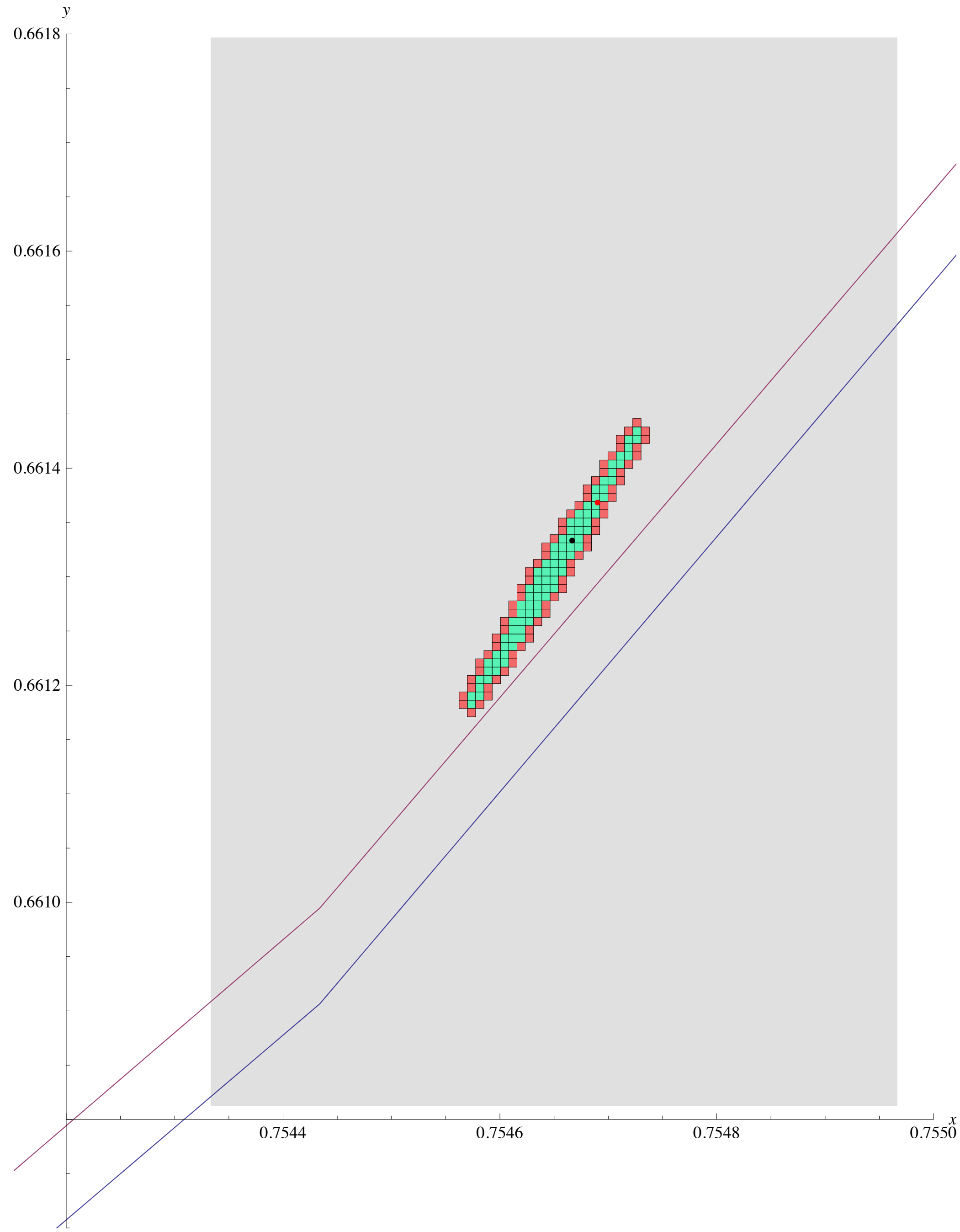
Linear solvers

- If the *A* and *b* values in *Ax=b* are rounded, the "lines" have width from uncertainty
- Apply a standard solver, and get the red dot as "the answer," *x*. A pair of floating-point numbers.
- Check it by computing *Ax* and see if it rigorously contains *b*. Yes, it does.
- Hmm… are there any *other* points that also work?



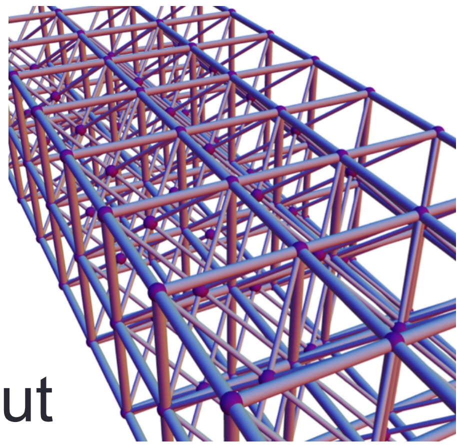
Float, Naïve Interval, and Ubox Solutions

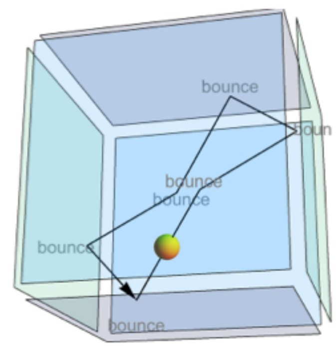
- Float solution (black dot) just gives *one of many* solutions; disguises instability
- Interval method (gray box) yields a bound too loose to be useful (naïve method)
- The ubox set (green) is the *best you can do for a given precision*
- Uboxes *reveal* ill-posed nature… yet provide solution anyway
- Works equally well on *nonlinear* problems!

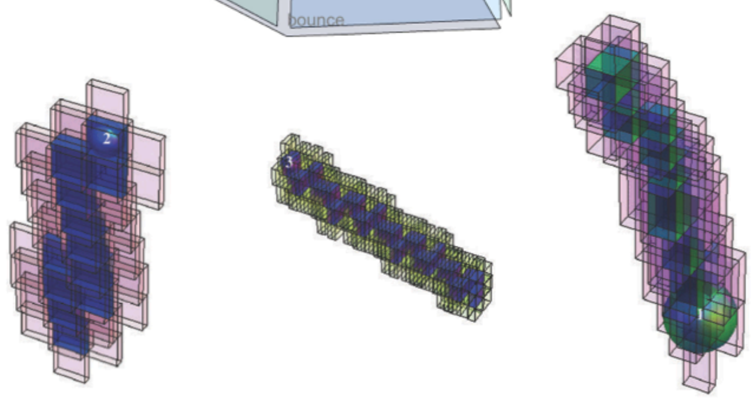


Other Apps with Ubox Solutions

- Photorealistic computer graphics
- *N*-body problems (!)
- Structural analysis
- Laplace's equation
- Perfect gas models without *statistical* mechanics







Imagine having **provable bounds** on answers for the first time, yet with easier programming, less storage, less bandwidth use, less energy/power demands, *and* abundant parallelism.

Revisiting the Big Challenges-1

Revisiting the Big Challenges-1

- *Too much energy and power needed per calculation*

Revisiting the Big Challenges-1

- *Too much energy and power needed per calculation*
  - Unums cut the main energy hog (memory transfers) by about 50%

Revisiting the Big Challenges-1

- *Too much energy and power needed per calculation*
  - Unums cut the main energy hog (memory transfers) by about 50%
- *More hardware parallelism than we know how to use*

Revisiting the Big Challenges-1

- *Too much energy and power needed per calculation*
  - Unums cut the main energy hog (memory transfers) by about 50%
- *More hardware parallelism than we know how to use*
  - Uboxes reveal vast sources of *data* parallelism, the easiest kind

Revisiting the Big Challenges-1

- *Too much energy and power needed per calculation*
  - Unums cut the main energy hog (memory transfers) by about 50%
- *More hardware parallelism than we know how to use*
  - Uboxes reveal vast sources of *data* parallelism, the easiest kind
- *Not enough bandwidth (the "memory wall")*

Revisiting the Big Challenges-1

- *Too much energy and power needed per calculation*
  - Unums cut the main energy hog (memory transfers) by about 50%
- *More hardware parallelism than we know how to use*
  - Uboxes reveal vast sources of *data* parallelism, the easiest kind
- *Not enough bandwidth (the "memory wall")*
  - More use of CPU transistors, fewer bits moved to/from memory

Revisiting the Big Challenges-1

- *Too much energy and power needed per calculation*
  - Unums cut the main energy hog (memory transfers) by about 50%
- *More hardware parallelism than we know how to use*
  - Uboxes reveal vast sources of *data* parallelism, the easiest kind
- *Not enough bandwidth (the "memory wall")*
  - More use of CPU transistors, fewer bits moved to/from memory
- *Rounding errors more treacherous than people realize*

Revisiting the Big Challenges-1

- *Too much energy and power needed per calculation*
  - Unums cut the main energy hog (memory transfers) by about 50%
- *More hardware parallelism than we know how to use*
  - Uboxes reveal vast sources of *data* parallelism, the easiest kind
- *Not enough bandwidth (the "memory wall")*
  - More use of CPU transistors, fewer bits moved to/from memory
- *Rounding errors more treacherous than people realize*
  - Unums eliminate rounding error, automate precision choice

Revisiting the Big Challenges-2

- *Rounding errors prevent use of multicore methods*

Revisiting the Big Challenges-2

- *Rounding errors prevent use of multicore methods*
  - Unums restore algebraic laws, eliminating the deterrent

Revisiting the Big Challenges-2

- *Rounding errors prevent use of multicore methods*
  - Unums restore algebraic laws, eliminating the deterrent
- *Sampling errors turn physics simulations into guesswork*

Revisiting the Big Challenges-2

- *Rounding errors prevent use of multicore methods*
  - Unums restore algebraic laws, eliminating the deterrent
- *Sampling errors turn physics simulations into guesswork*
  - Uboxes produce provable *bounds* on physical behavior

Revisiting the Big Challenges-2

- *Rounding errors prevent use of multicore methods*
  - Unums restore algebraic laws, eliminating the deterrent
- *Sampling errors turn physics simulations into guesswork*
  - Uboxes produce provable *bounds* on physical behavior
- *Numerical methods are hard to use, require expertise*

Revisiting the Big Challenges-2

- *Rounding errors prevent use of multicore methods*
  - Unums restore algebraic laws, eliminating the deterrent
- *Sampling errors turn physics simulations into guesswork*
  - Uboxes produce provable *bounds* on physical behavior
- *Numerical methods are hard to use, require expertise*
  - "Paint bucket" and "Try everything" are brute force general methods that need no expertise… not even calculus

Next steps

- Convert *Mathematica* prototype into a C library
  - Fixed-size types, plus lossless pack and unpack
  - Use existing integer data types (8-bit, 64-bit)
  - Python, Julia also strong candidates for language support

Next steps

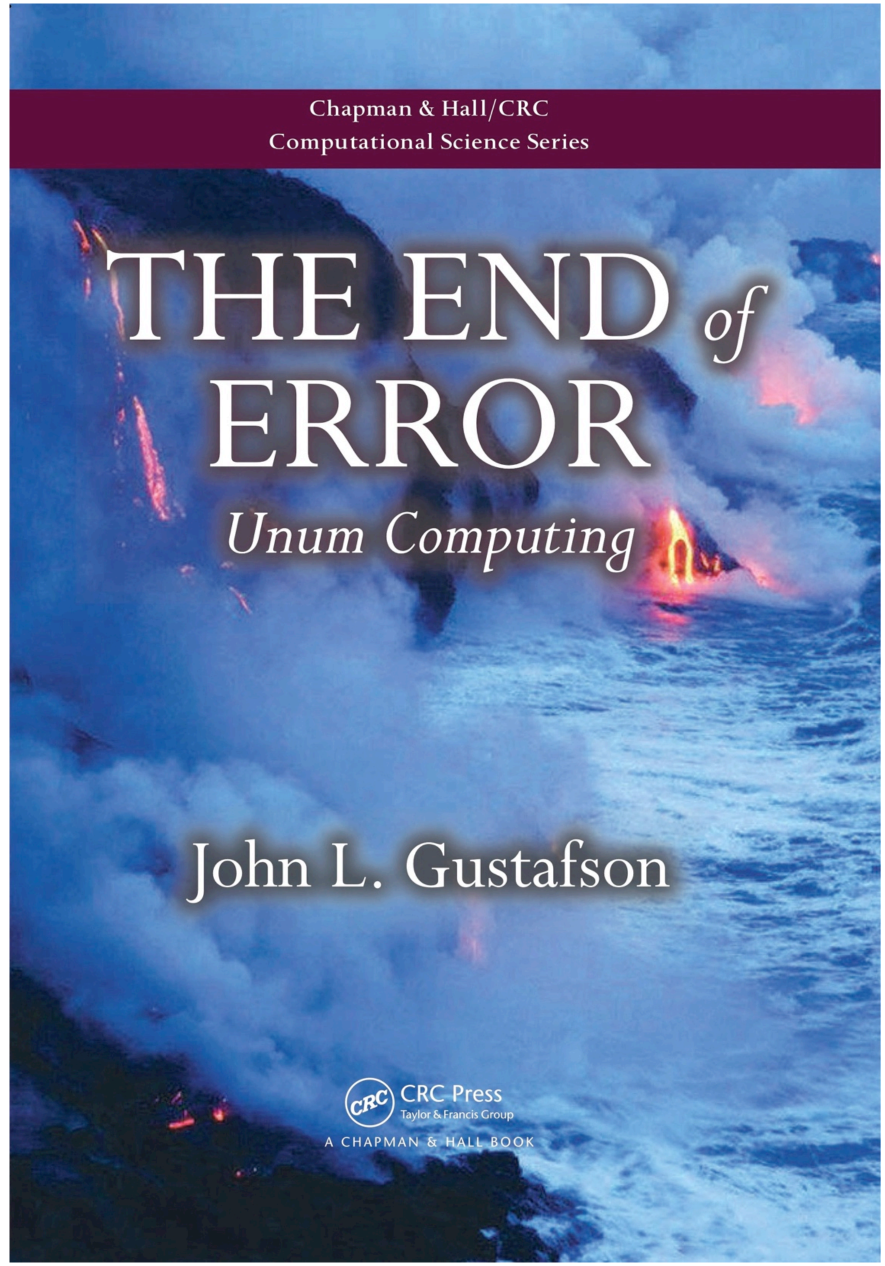
- Convert *Mathematica* prototype into a C library
  - Fixed-size types, plus lossless pack and unpack
  - Use existing integer data types (8-bit, 64-bit)
  - Python, Julia also strong candidates for language support
- Create FPGA from C version
  - Maximize the environment settings depending on FPGA size
  - Port to a Convey or other chip with user-definable ops
  - See if it accelerates codes while providing rigorous bounds

Next steps

- Convert *Mathematica* prototype into a C library
  - Fixed-size types, plus lossless pack and unpack
  - Use existing integer data types (8-bit, 64-bit)
  - Python, Julia also strong candidates for language support
- Create FPGA from C version
  - Maximize the environment settings depending on FPGA size
  - Port to a Convey or other chip with user-definable ops
  - See if it accelerates codes while providing rigorous bounds
- Custom VLSI processor
  - Initially with fixed-size data, plus lossless pack and unpack
  - Eventually, bit-addressed architecture with hardware memory management (similar to disc controller)

The End of Error

- A complete text on unums and uboxes is available from CRC Press as of Feb 2015: http://www.crcpress.com/product/isbn/9781482239867
- Aimed at *general* reader; for a formal version, see Kulisch
- Complete prototype environment is available as free *Mathematica* notebook through publisher



The End of Error

- A complete text on unums and uboxes is available from CRC Press as of Feb 2015: http://www.crcpress.com/product/isbn/9781482239867
- Aimed at *general* reader; for a formal version, see Kulisch
- Complete prototype environment is available as free *Mathematica* notebook through publisher

Thank you!