AN ENERGY-EFFICIENT AND MASSIVELY PARALLEL APPROACH TO VALID NUMERICS

John L. Gustafson

ICRAR Seminar

john.gustafson@nus.sg.edu

2 June 2016

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.

Big problems facing computing

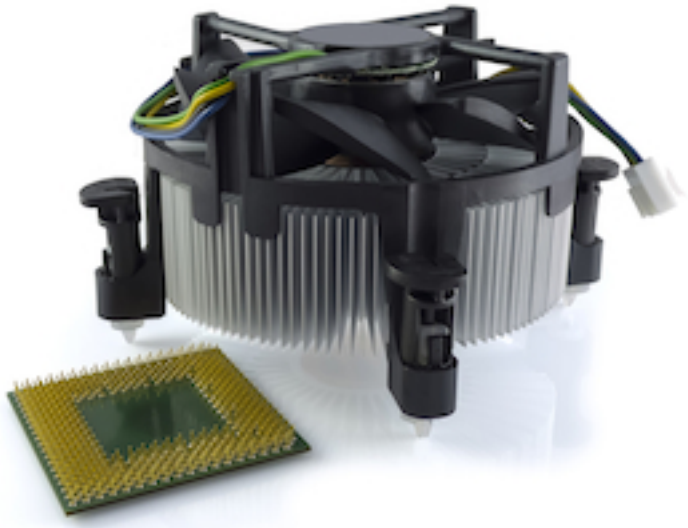
- Too much energy and power needed per calculation
- More hardware parallelism than we know how to use
- Not enough bandwidth (the “memory wall”)
- Rounding errors more treacherous than people realize
- Rounding errors prevent use of parallel methods
- Sampling errors turn physics simulations into guesswork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

- 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***

Notice that 12000 pJ at 3 GHz = 36 watts!

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

Happy 101th 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.”

A Key Idea: The Ubit

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

Incorrect: $\pi = 3.14$

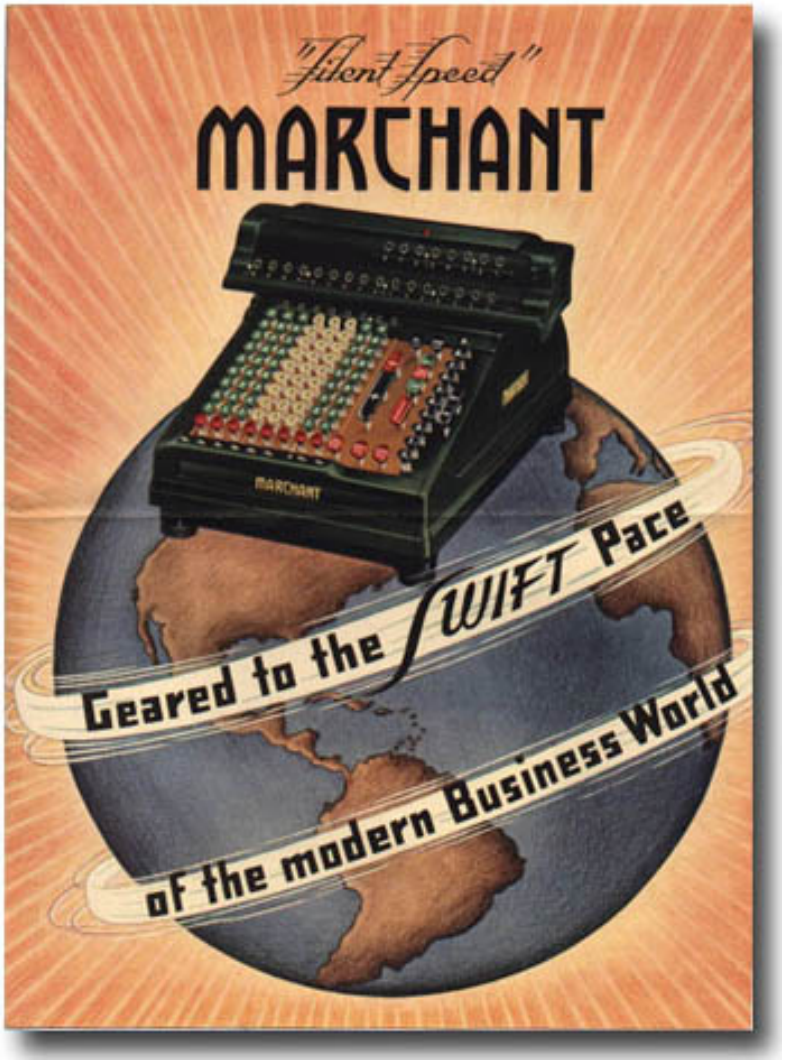
Correct: $\pi = 3.14\cdots$

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.

Floats designed for *visible* scratch work

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



Analogy: Printing in 1970 vs. 2015

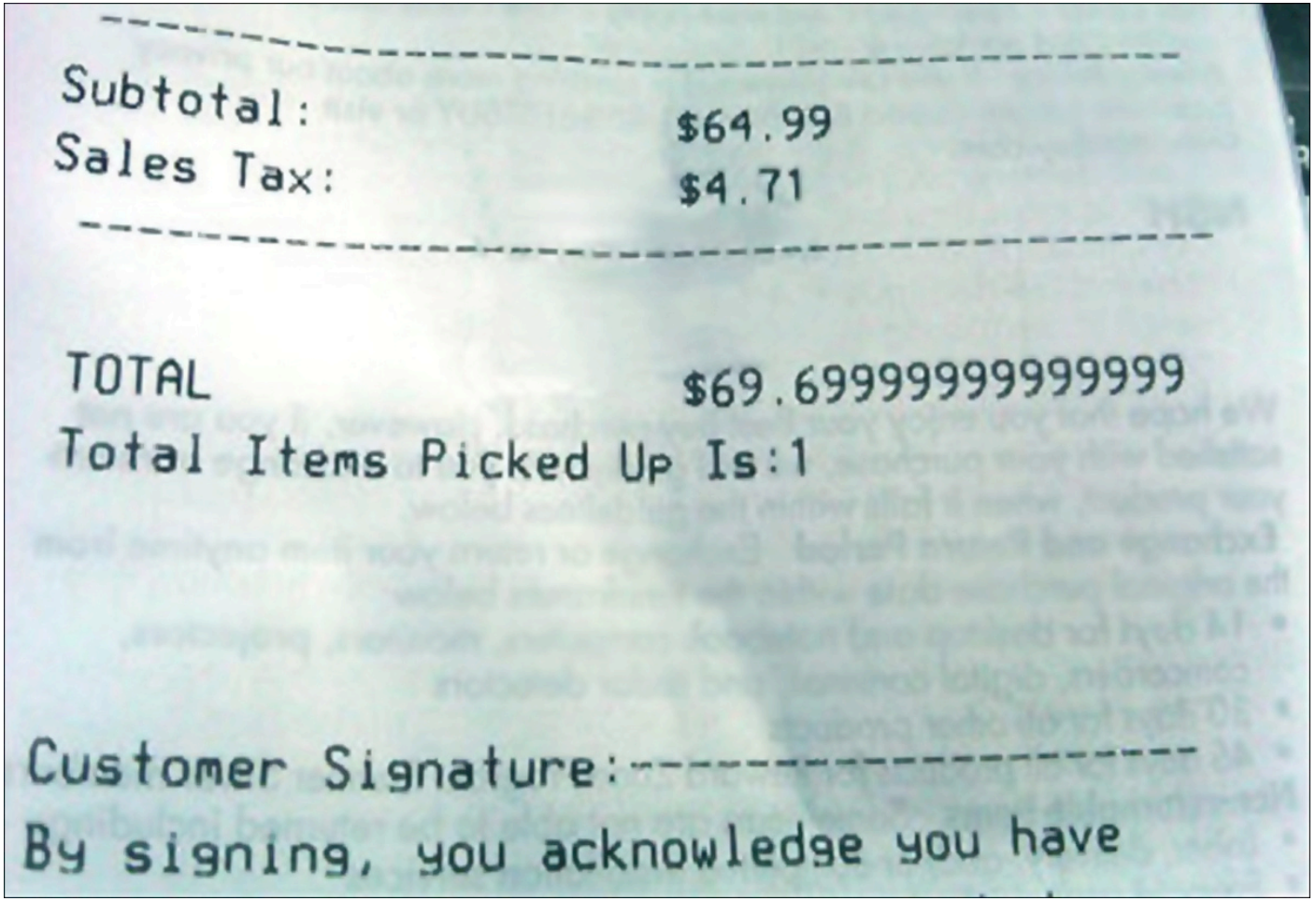
1970: 30 sec per page

```
      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

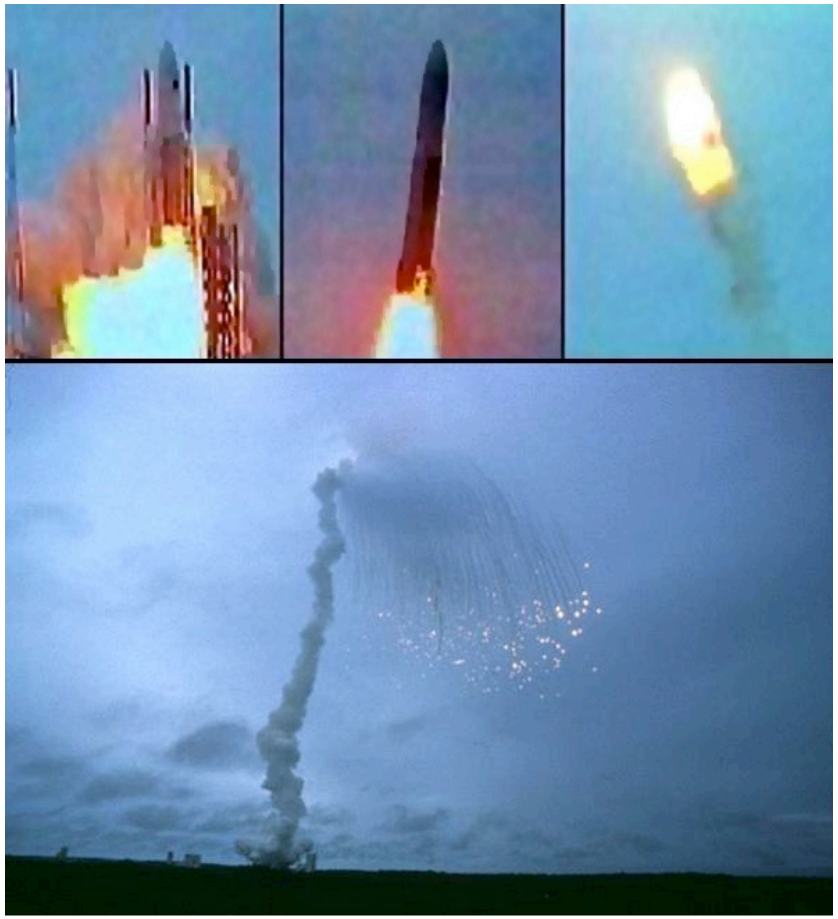
Faster technology is for *better* prints,
not thousands of low-quality prints per second.
Why not do the same thing with computer arithmetic?

This is just… sad.



Float Disaster: The Ariane 5

- 64-bit float measured speed
- 16-bit guidance system; *oops*
- $0.7 billion gone in seconds



Why do *programmers* have to manage storage sizes when computers are much better at doing it?

When rounding error killed 38 people

- First Gulf War
- Left on for 100 hours
- Fraction “crowded out” by integer part
- Guidance off by 0.43 seconds
- Patriot missed Scud
- Scud struck barracks

The Sleipner Oil Platform Disaster

- Float error in structural analysis; collapsed to ocean floor
- ~$1 billion loss (in 2015 dollars)

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
- **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*

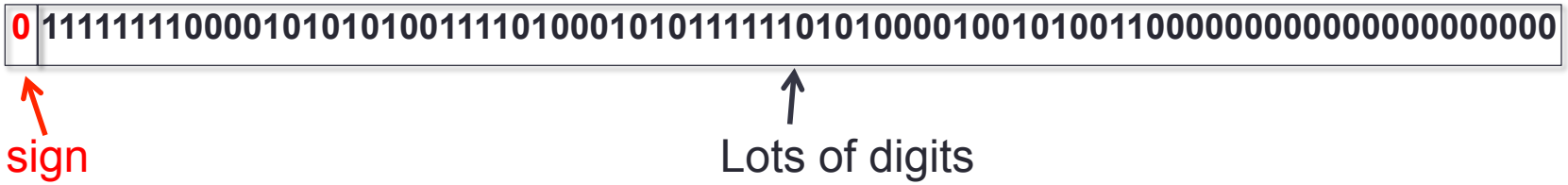
"You can't boil the ocean."

—Former Intel exec, when shown the unum idea

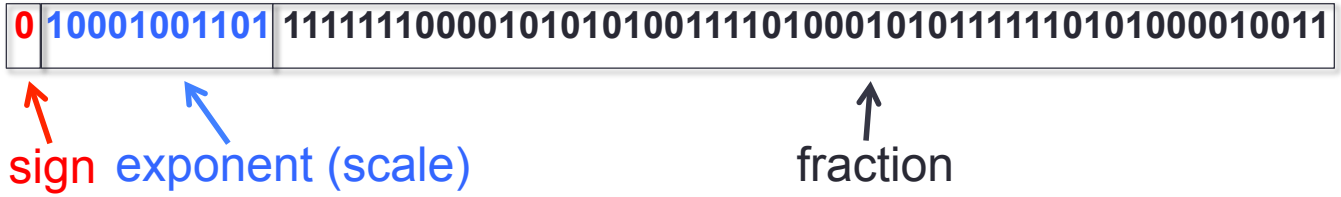
Three ways to express a big number

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

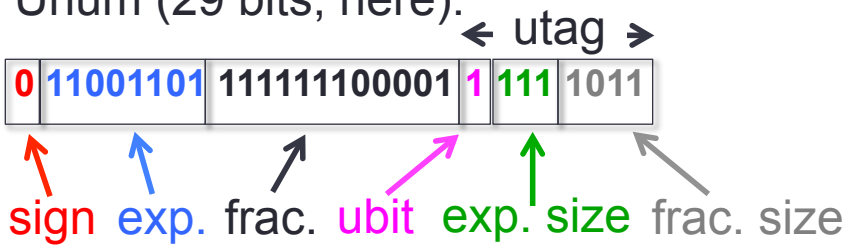
Sign-Magnitude Integer (80 bits):



IEEE Standard Float (64 bits):



Unum (29 bits, here):



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
- 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):

Unum (29 bits) (for example):

A Typesetting Analogy

```
The biggest
objection to unums
is likely to come
from the fact that
they are variable in
size, at least when
they are stored in
packed form.
```

18-point Courier
(fixed width font)

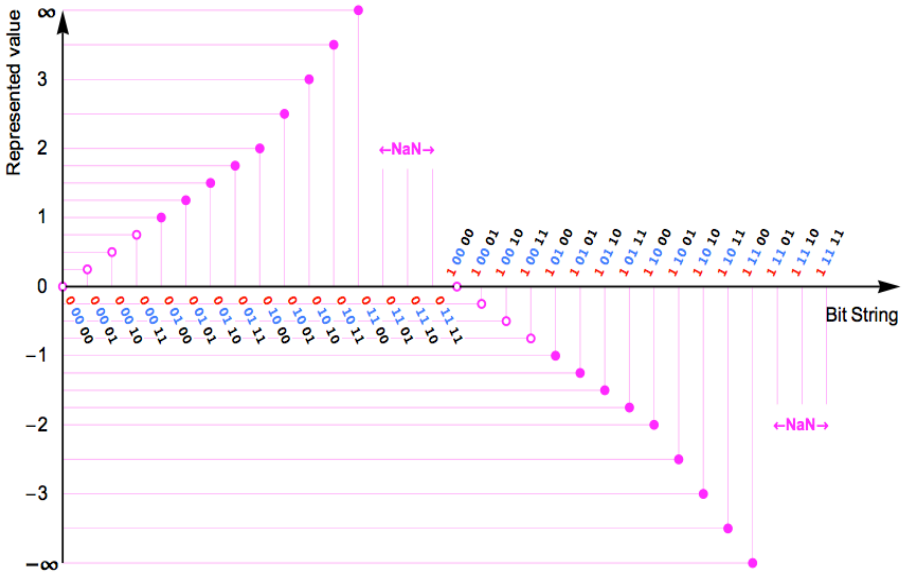
The biggest objection to unums is likely to come from the fact that they are variable in size, at least when they are stored in packed form.

18-point Times
(variable width font)

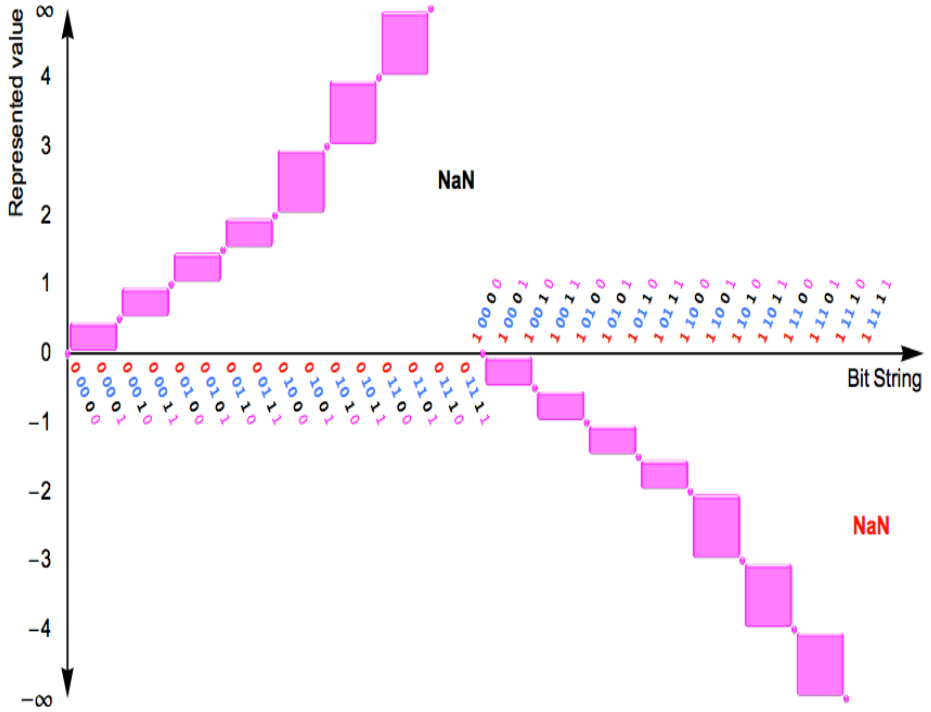
This shows the price of “One Size Fits All.”

Open *ranges*, as well as exact points

Bit string meanings
using IEEE Float rules



Bit string meanings
in unum format

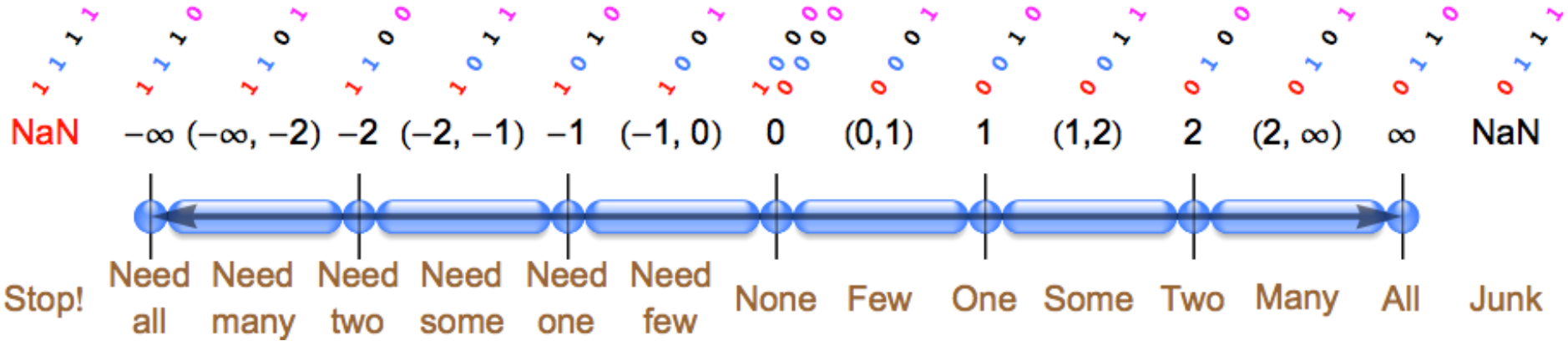


Complete representation of *all* real numbers using a finite number of bits

The Warlpiri unums

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

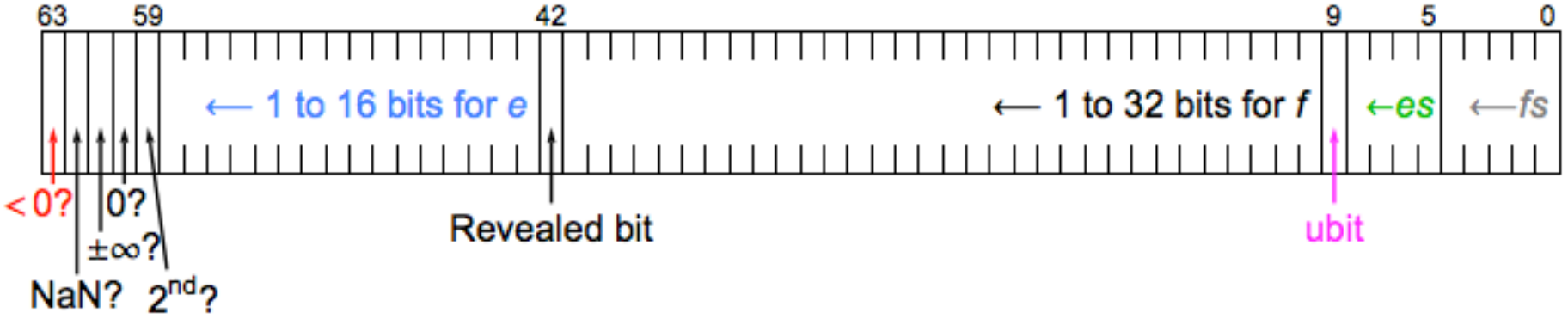
Maybe they were onto something.



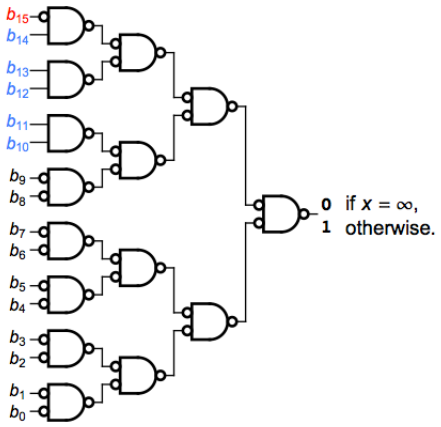
"Try everything" methods become feasible.

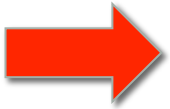
Fixed-size unums: faster than floats

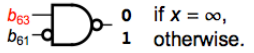
- Warlpiri ubounds are one byte, but closed system for reals
- Unpacked unums *pre-decode* exception bits, hidden bit



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







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

Floating Point II: The Wrath of Kahan

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

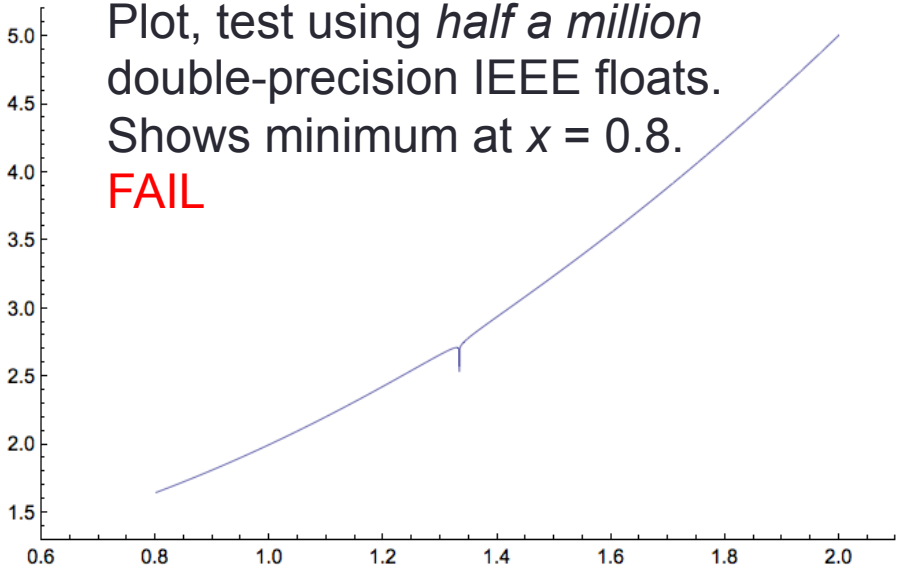
Compute $H(x)$ for $x = 15.0, 16.0, 17.0, 9999.0$. Repeat with more precision, say using BigDecimal."

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

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

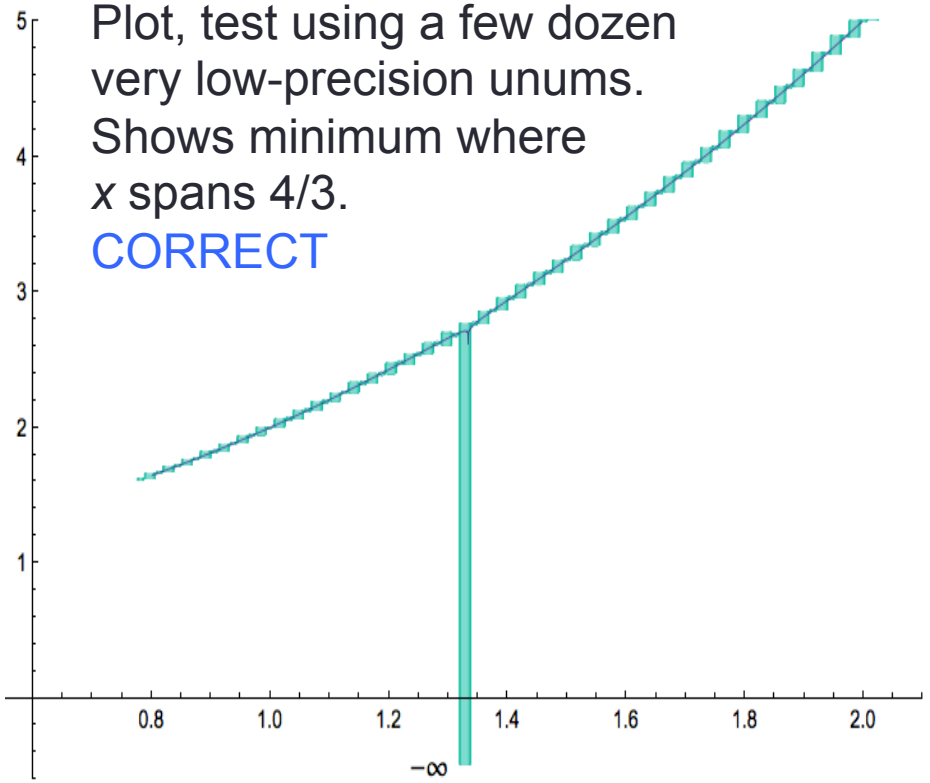
Kahan's "Smooth Surprise"

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



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

Even if you test *every float in the range,* it will fail to detect the minimum!



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 $x = 77617$, $y = 33096$.

- Using IBM (pre-IEEE Standard) floats, Rump got
  - 1.172603 in 32-bit precision
  - 1.1726039400531 in 64-bit precision
  - 1.17260394005317**8** in 128-bit precision
- **Correct answer: –0.827396059946821**36⋯**!**

  Didn't even get *sign* right

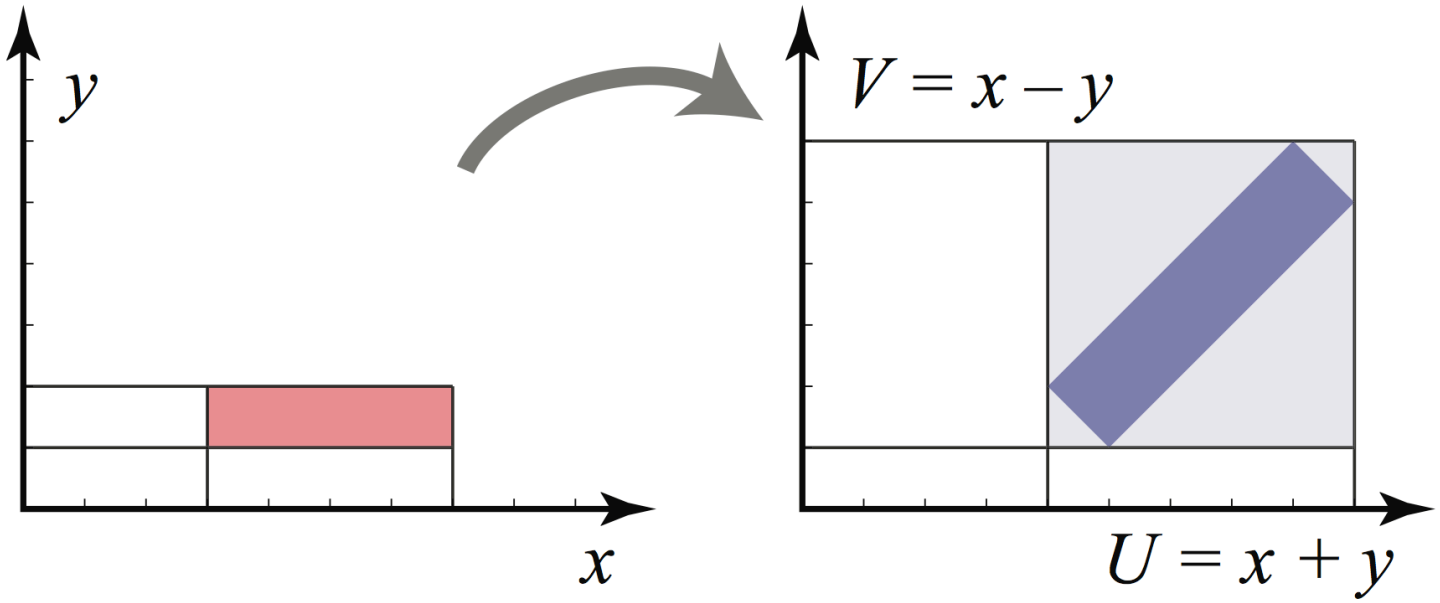
Unums: **Correct answer** to 23 decimals using an average of only ***75*** bits per number. Not even IEEE 128-bit precision can do that. Precision, range adjust *automatically*.

Some principles of unum math

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

- 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*
- Maximize information per bit
- Fused operations are always explicitly distinct from their non-fused versions, so results are **bitwise identical** across platforms

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.

Accuracy on a 32-bit budget: a contest

For fair comparison of different number systems, put them all on a strict diet of 32 bits, and compare their accuracy.

Say we want to compute $$\cos(\frac{2\pi}{3})\left(\frac{e-2.7}{\pi-(\sqrt{2}+\sqrt{3})}\right)$$

and we try

- 32-bit IEEE floats
- Interval arithmetic [a,b] where a and b are 16-bit floats
- Unum arithmetic with *average* size less than 32 bits

Contest results: Floats

Correct answer, to 20 decimals: 1.95665009183592105270…

32-bit IEEE floats: 1.95669615268707275390625

- 5 decimals correct
- Off by *486* ULPs
- No indication of what the accuracy is

The nominal accuracy of 32-bit IEEE floats is 7+ decimals.

Contest results: Intervals

16-bit IEEE floats have 3+ decimal accuracy and a dynamic range of about 9 orders of magnitude.

Example of an interval bound:

$e = 2.71828\cdots$ is represented by [2.716796875, 2.71875].

Result given by interval math:

[0.99853515625, 5.009765625]

????

- *No* decimals correct
- A rigorous bound, but *uselessly pessimistic*

Contest results: Unums

Using 9 bits in the self-descriptive utag, unums can take anywhere from 12 to 78 bits, but we can cap it at 44 bits.

With an *average* length of 31.5 bits, the unum result is the open interval

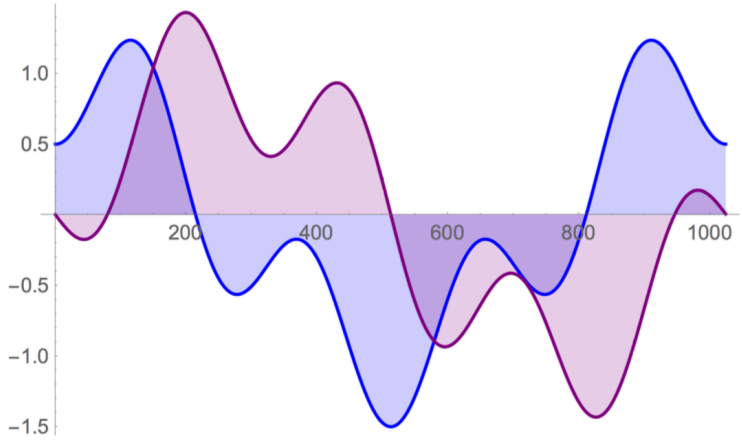
(1.9566497802734375, 1.956650257110595703125)

This bound is *sixty times more accurate* than the float result and it **guarantees the bound**.

With 32 bits as the *maximum* size permitted, we get an average size of 22.8 bits, and the interval

1.953125, 1.9609375)

An Experiment that should interest SKA: Fast Fourier Transforms with unums



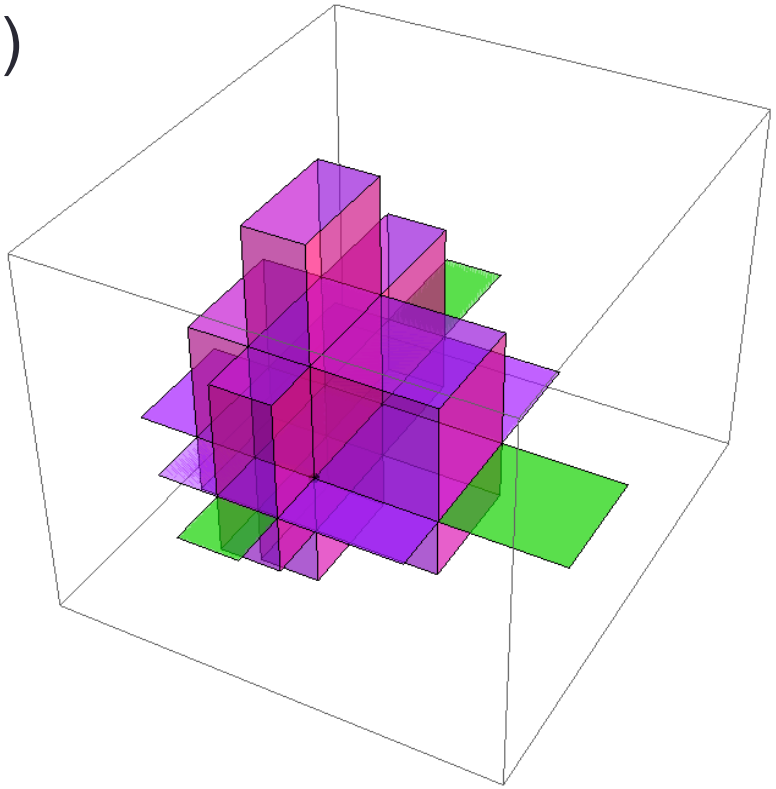
- Standard test: Complex 1024-point FFT, in-place.
- 12-bit input data, like A-to-D convertors produce
- Utag adds 6 bits to each value, but…

Numbers moved	178 152
Unum bits moved	4 043 801
Average bits per number	**22.7**

Bounds, not guesses!

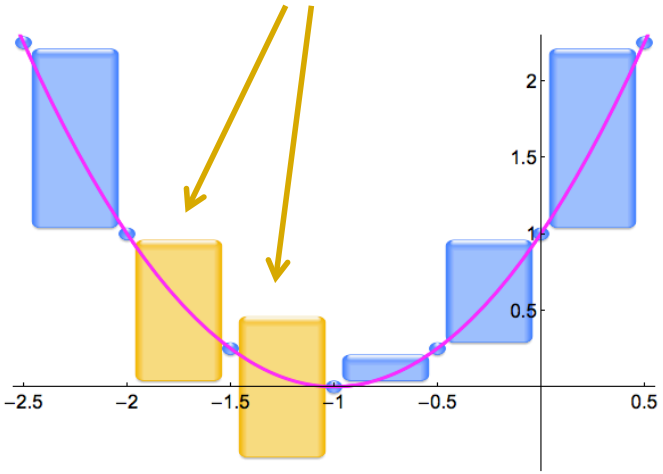
Uboxes and solution sets

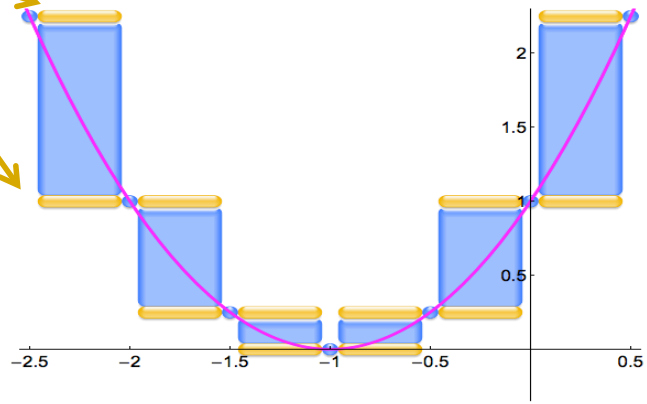
- A *ubox* is a multidimensional unum
- Exact or ULP-wide in each dimension (Unit in the Last Place)
- Sets of uboxes constitute a *solution set*
- One dimension per degree of freedom in solution
- Solves the main problems with interval arithmetic
- Super-economical for bit storage
- Data parallel in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

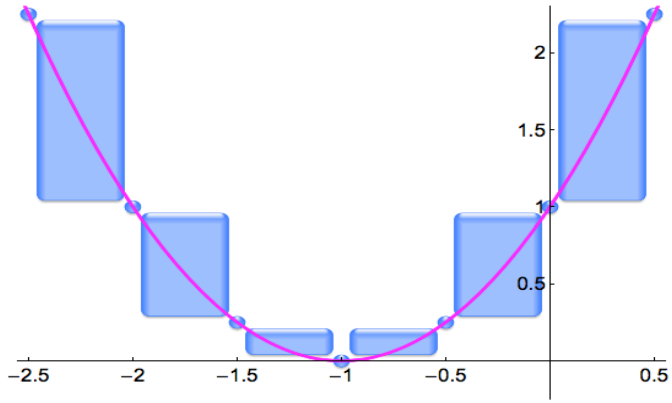


Polynomials: bane of classic intervals

Dependency and closed endpoints lose information (amber)



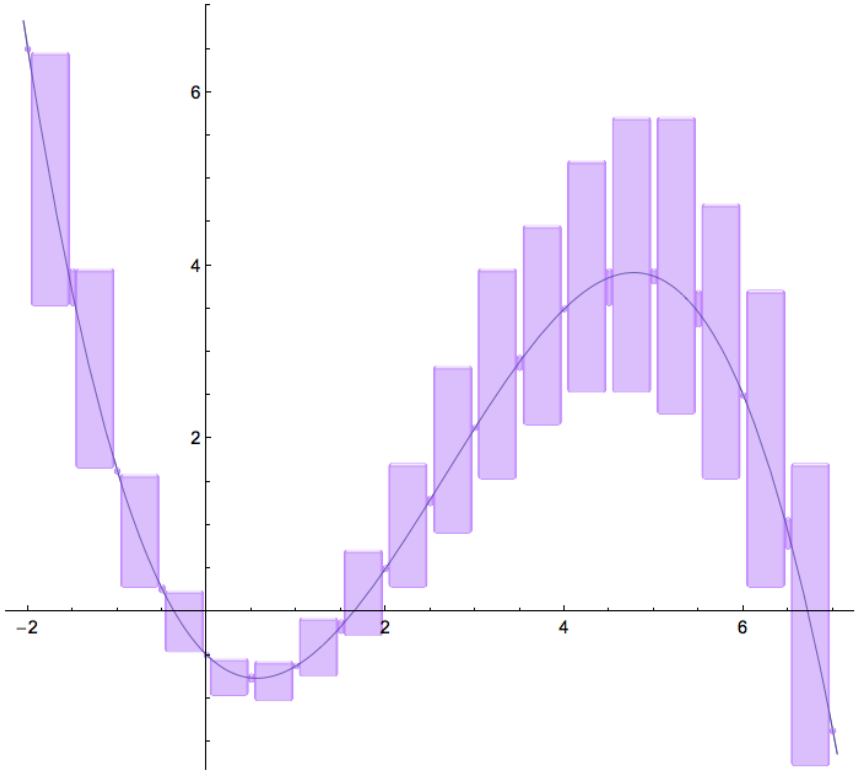




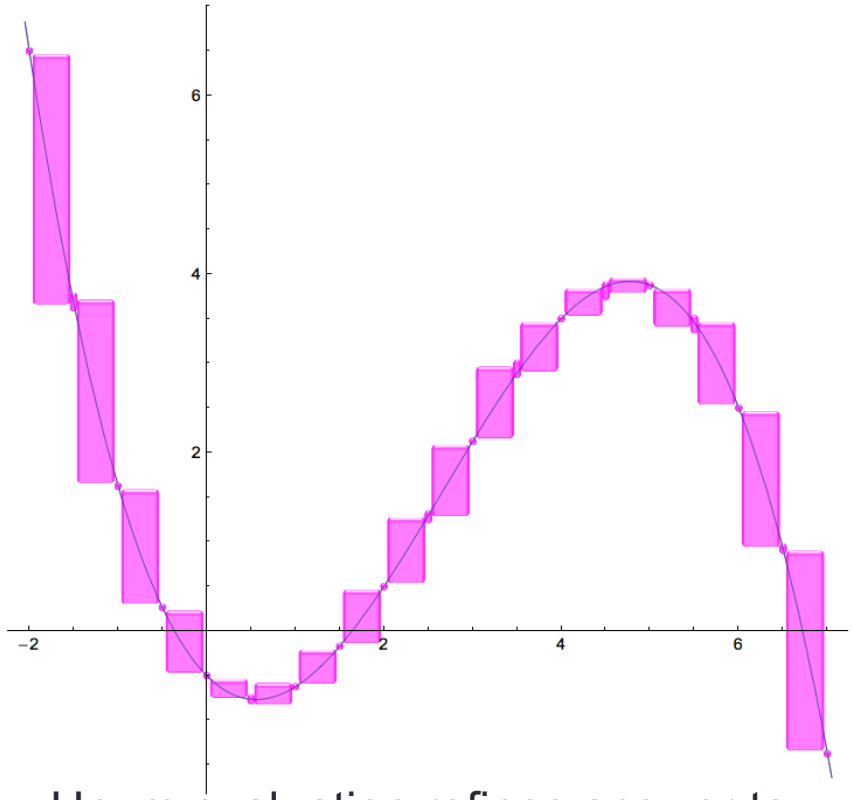
Unum polynomial evaluator loses *no* information.

Polynomial evaluation solved at last

Mathematicians have sought this for at least 60 years.



"Dependency Problem" creates sloppy range when input is an interval

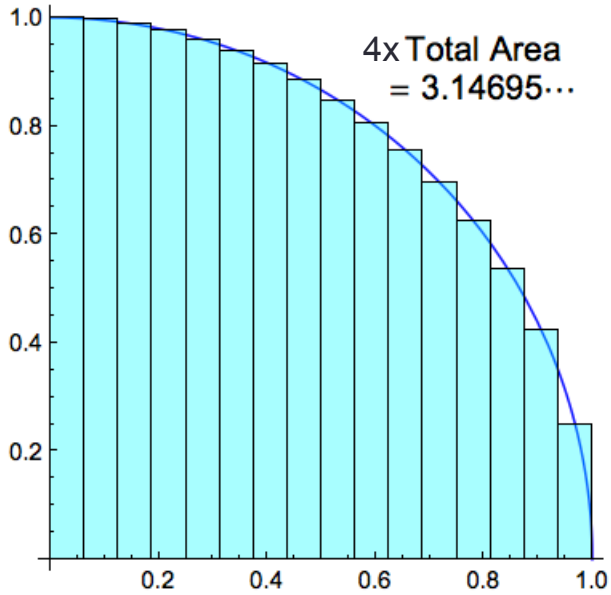


Unum evaluation refines answer to limits of the environment precision

The Deeply Unsatisfying Error Bounds of Classical Analysis

- Classical numerical texts teach this "error bound":

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



- 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 demand for supercomputing that *cannot be satisfied*.

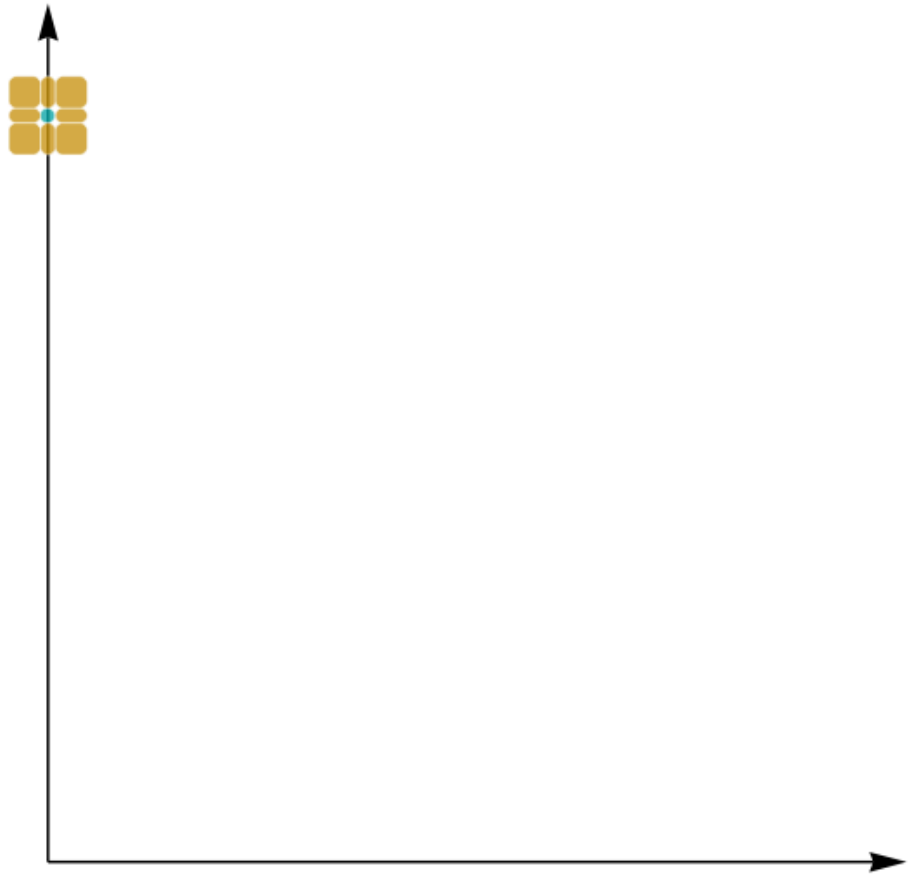
Quarter-circle example

- 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

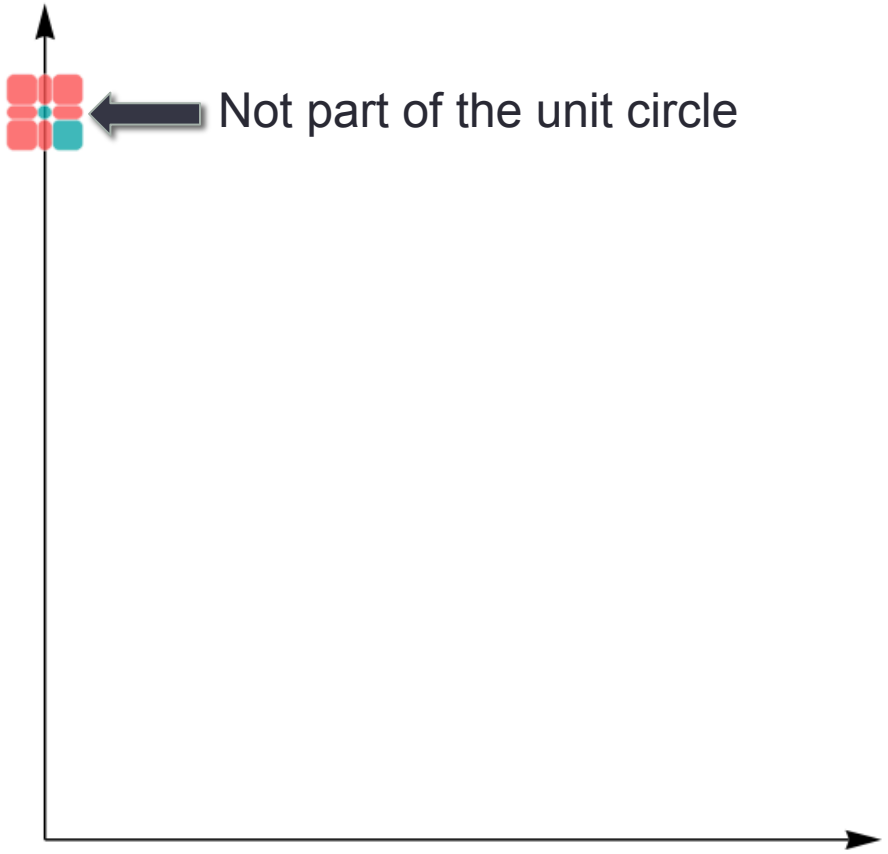
- We know $x = 0$, $y = 1$ works
- Find its eight ubox neighbors in the plane
- Test $x^2 + y^2 = 1$, $x \geq 0$, $y \geq 0$
- Solution set is green
- Trial set is amber
- Failure set is red
- Stop when no more trials



Exactly one neighbor passes

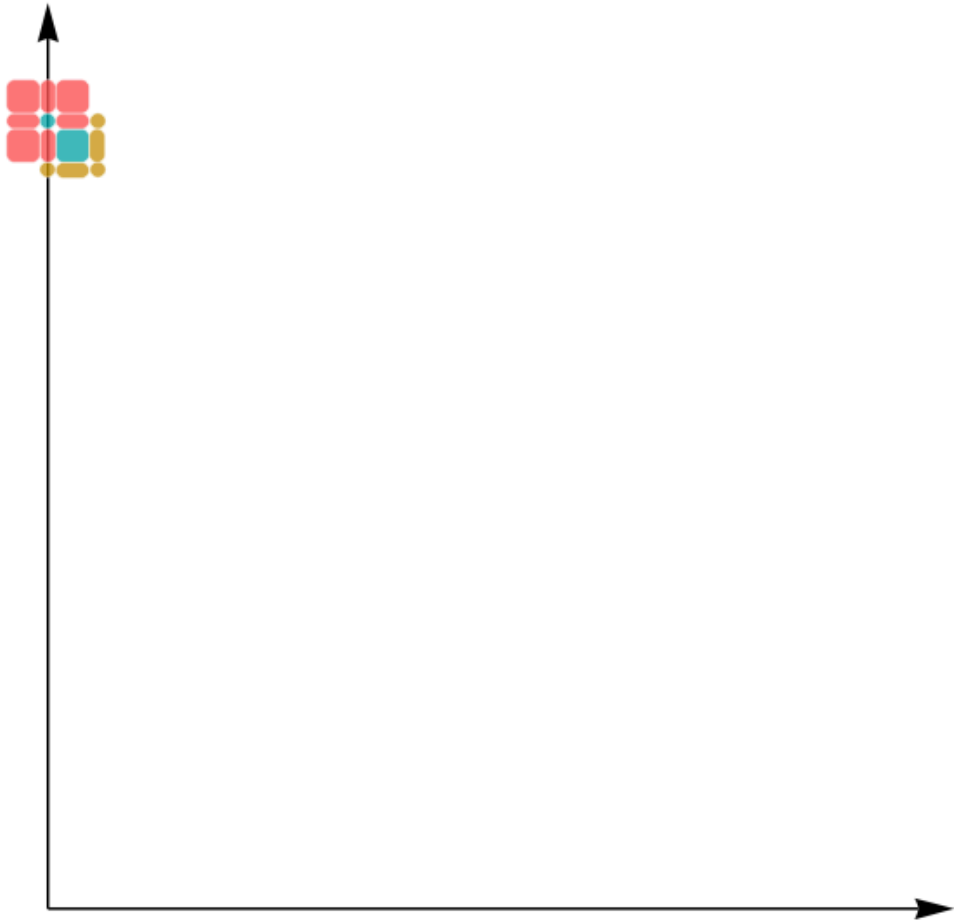
- Unum math automatically excludes cases that floats would accept
- Trials are neighbors of new solutions that
  - Are not already failures
  - Are not already solutions
- Note: *no* calculation of

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



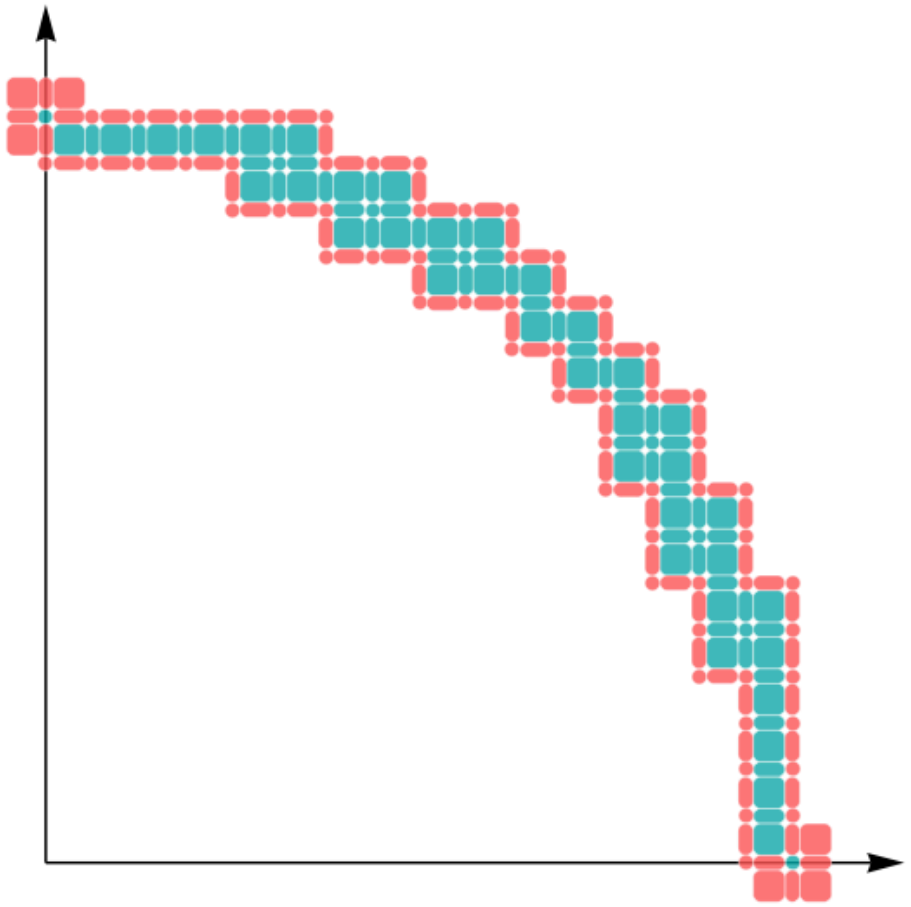
The new trial set

- 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

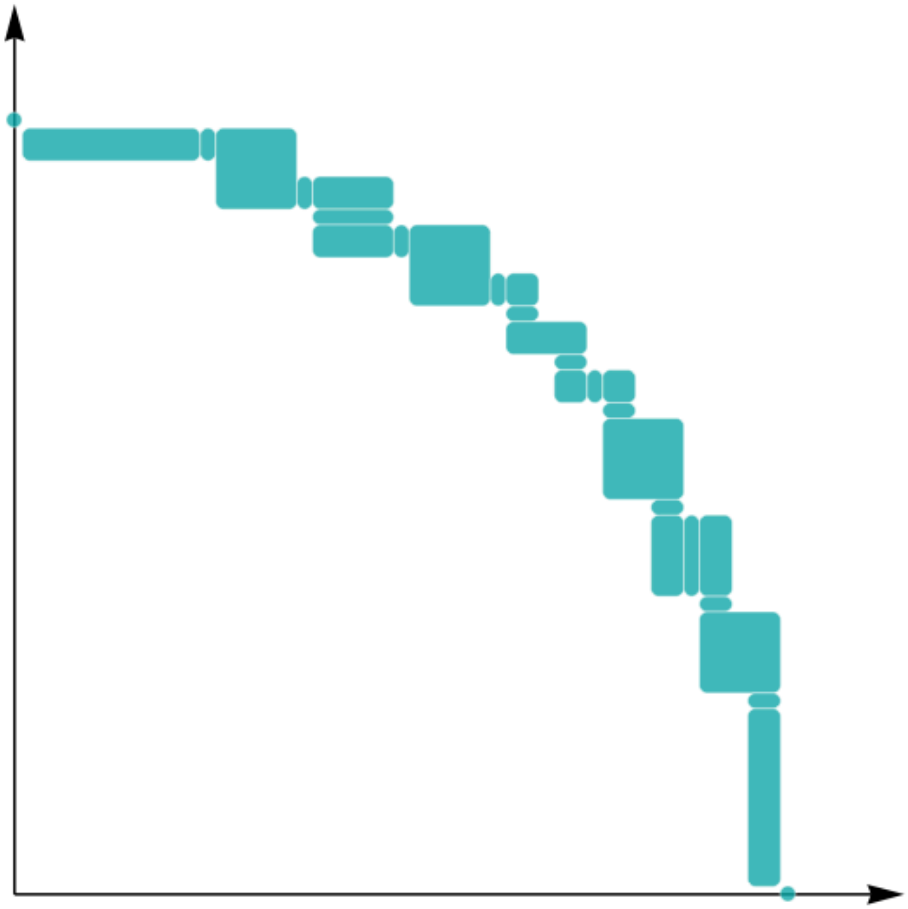
- The complete *solution*, to this finite precision level
- *Information* is reciprocal of green area
- Use to find area under arc, bounded above and below
- *Proves* value of π to an accuracy of 3%
- No calculus needed, or divides, or square roots



Compressed Final Result

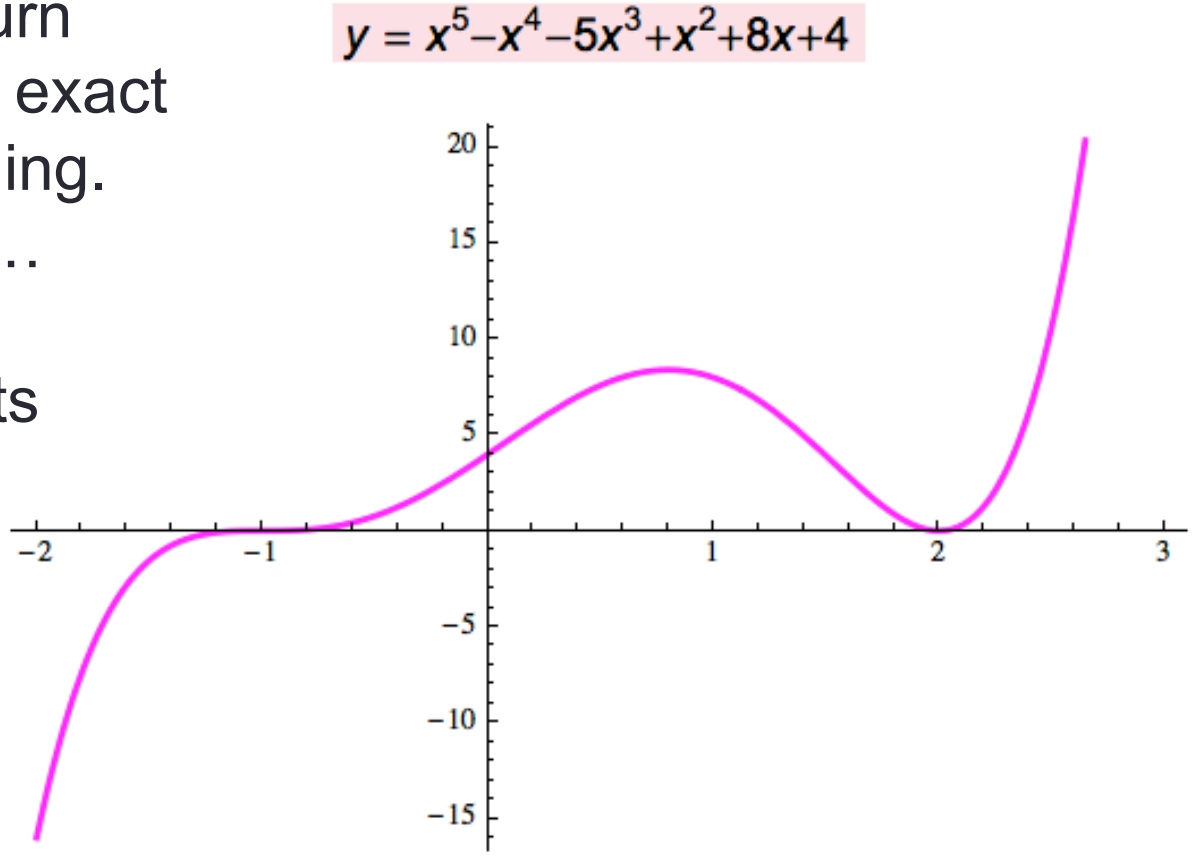
- Coalesce uboxes to largest possible ULP values
- *Lossless* compression
- Total data set: 603 bits!
- 6x faster graphics than current methods

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

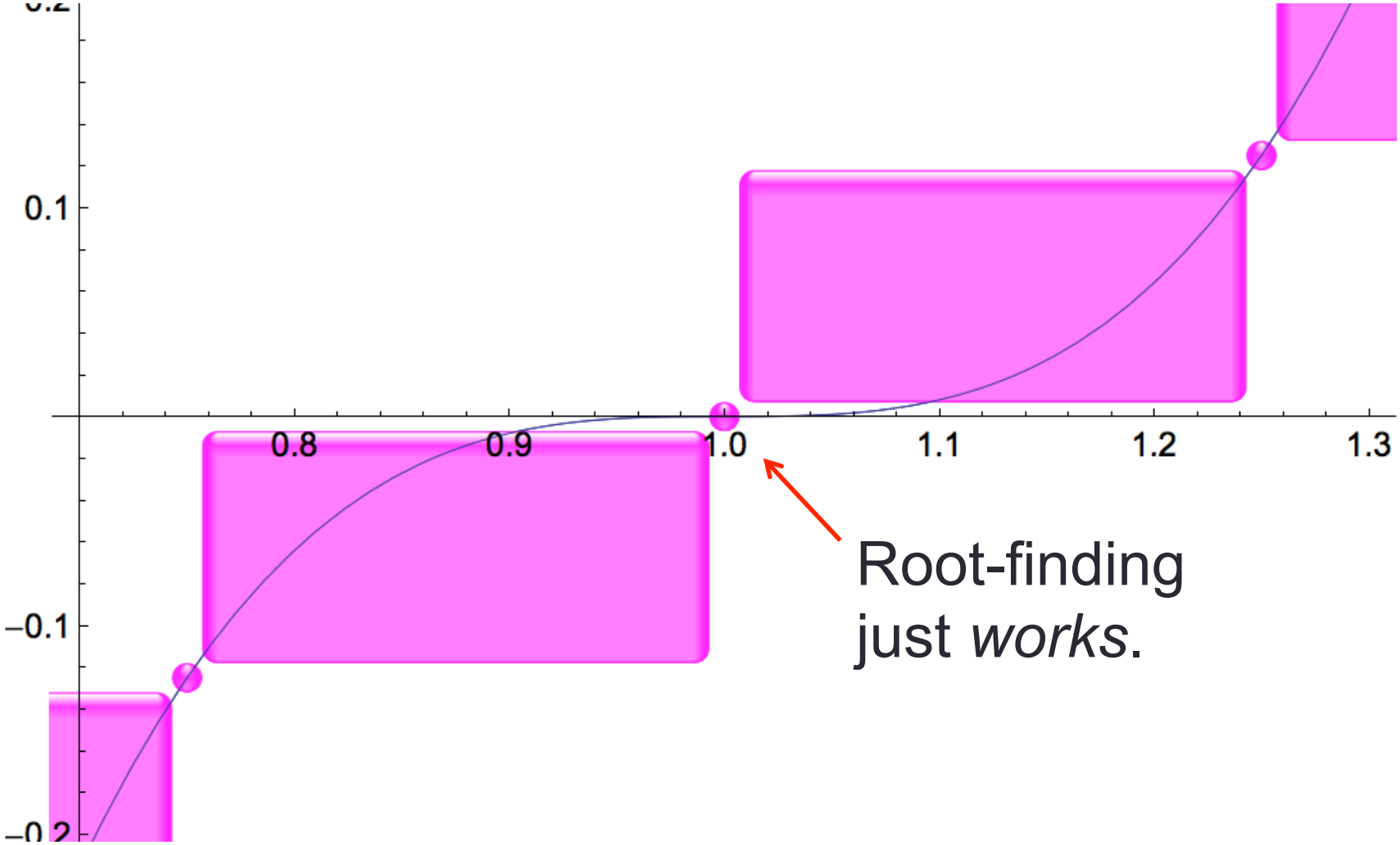


Fifth-degree polynomial roots

- Analytic solution: There isn't one.
- 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!

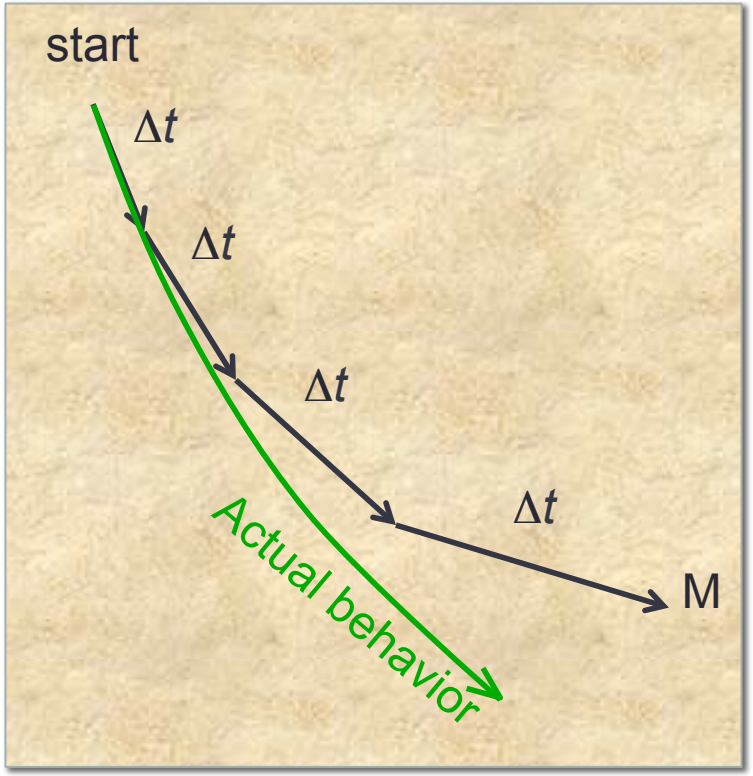


The power of open-closed endpoints



Classical Numerical Analysis

- Time steps
  - Use position to estimate force
  - Use force to estimate acceleration
  - Update the velocity
  - Update the position
  - Lather, rinse, repeat
- Accumulates rounding and sampling error, both unknown
- ***Cannot be done in parallel***

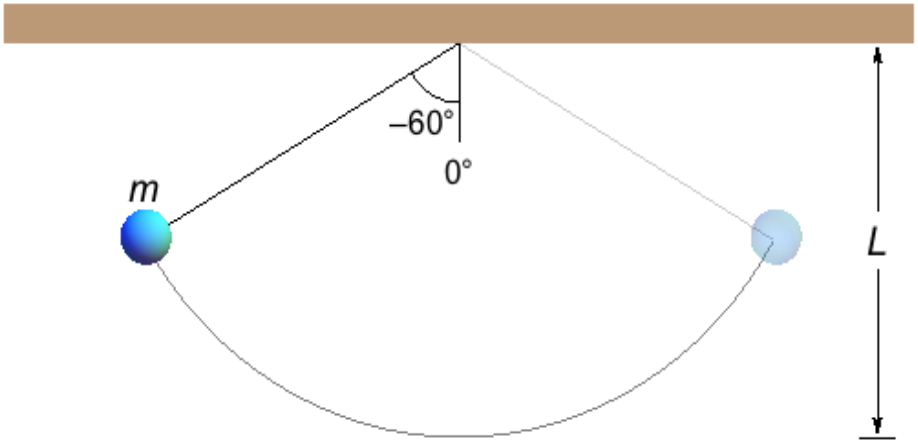


Pendulums Done Right

- Physics teaches us it's a harmonic oscillator with period

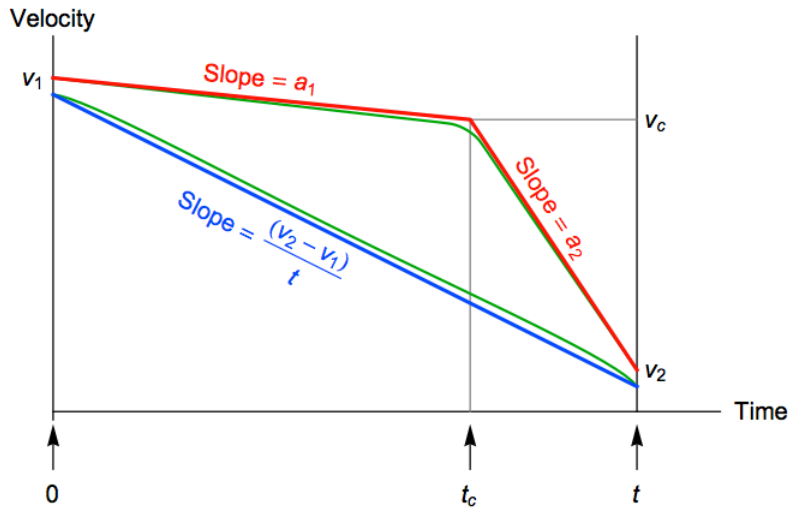
$$2\pi\sqrt{\frac{g}{L}}$$

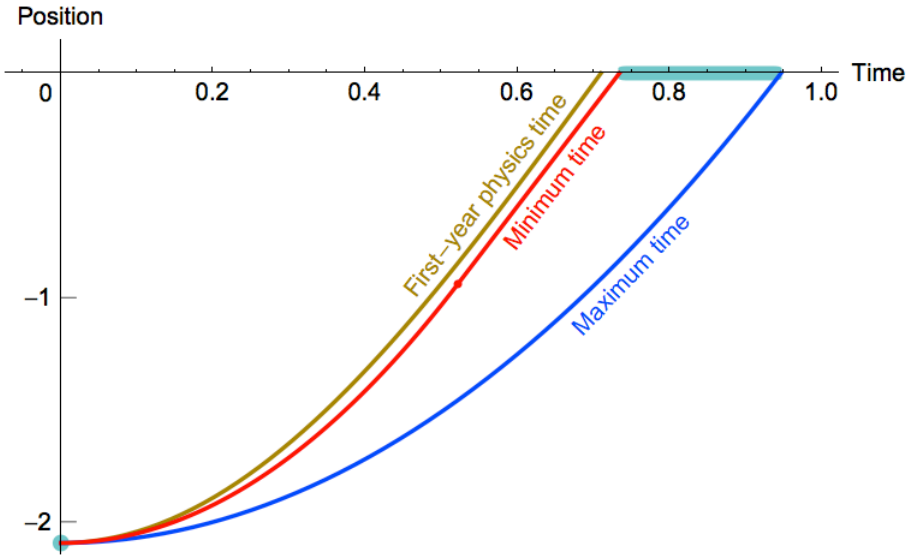
- Force-fits nonlinear ODE into linear ODE for which calculus works.
- WRONG answer



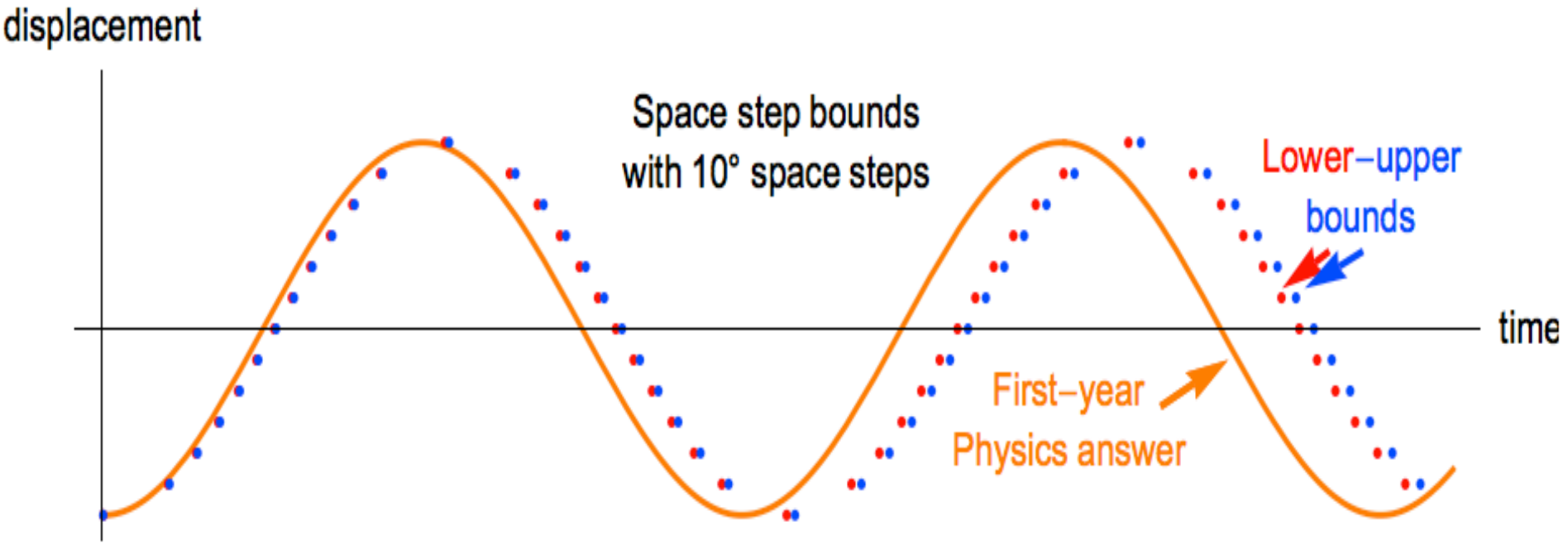
A New Type of Parallelism

- *Space* steps, not time steps
- Acceleration, velocity bounded in any given space interval
- Find traversal time as a function of space step (2D ubox)
- Massively parallel!
- *No rounding error*
- *No sampling error*
- Obsoletes existing ODE methods





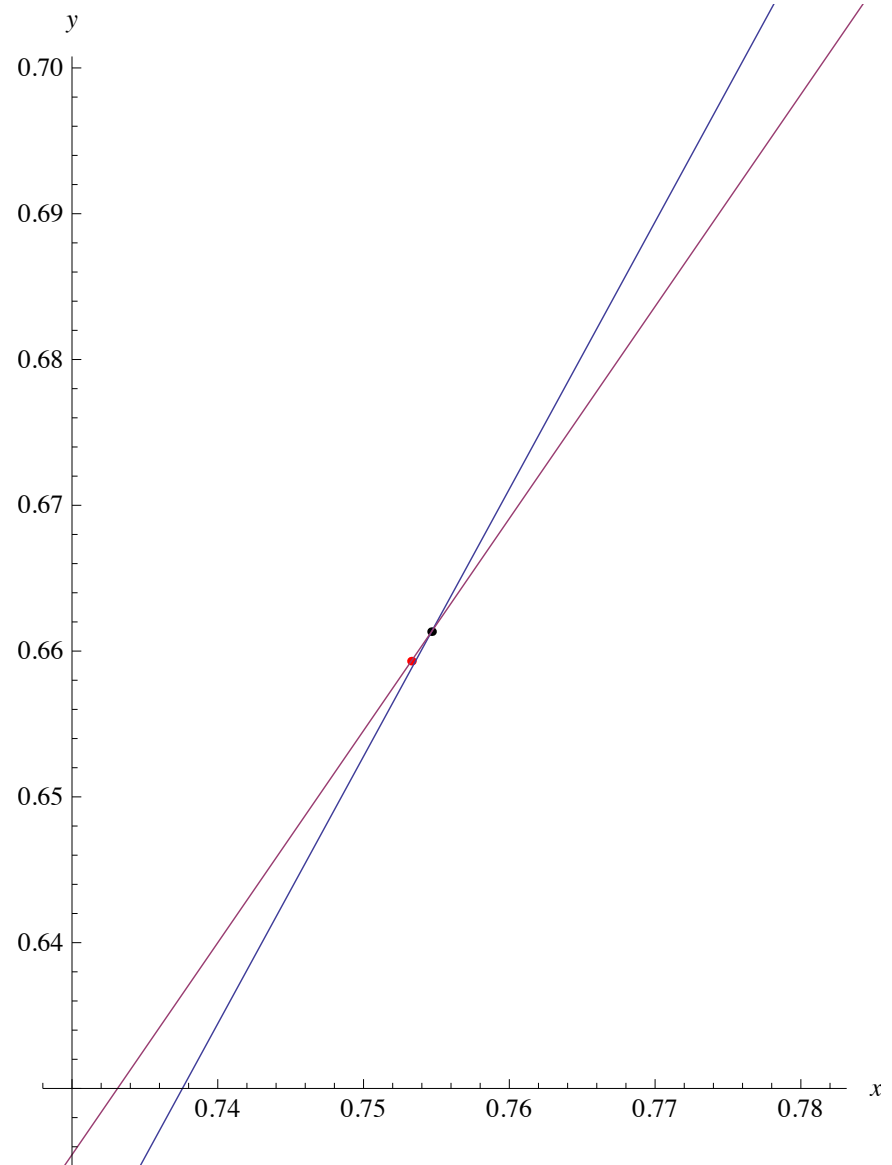
Physical Truth vs. Force-Fit Solution



Traditional approach bends the problem to fit known solution methods, gets wrong answer

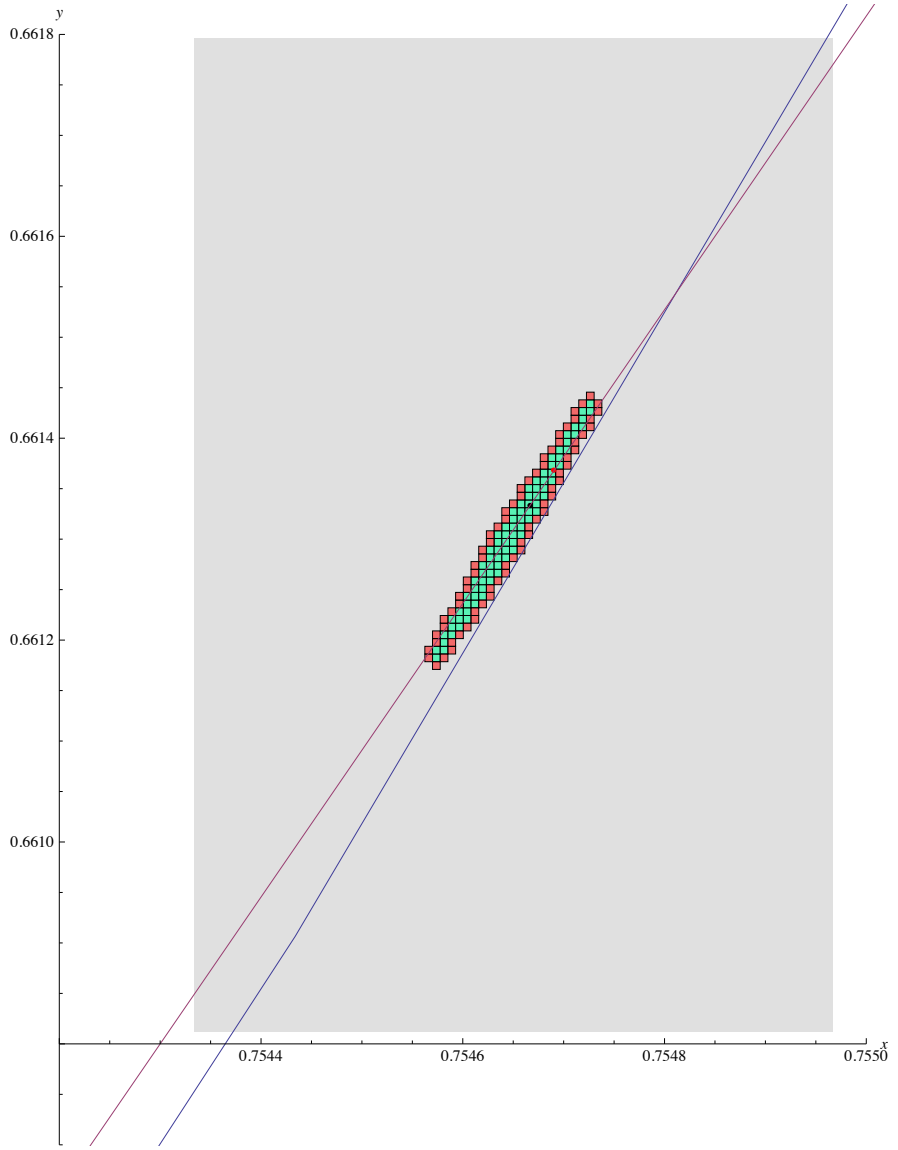
Uboxes for linear solvers

- 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?



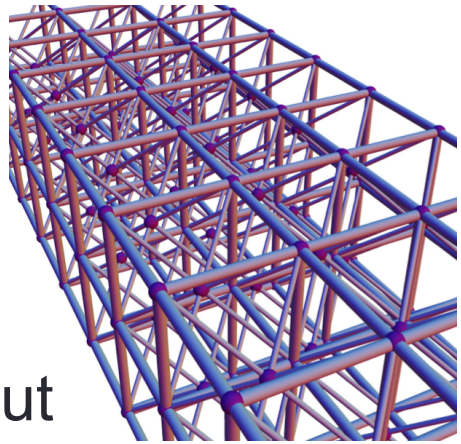
Float, Interval, and Ubox Solutions

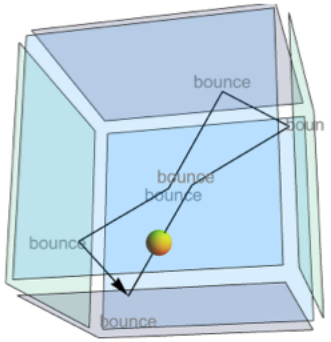
- Point solution (black dot) just gives *one of many* solutions; disguises answer instability
- Interval method (gray box) yields a bound too loose to be useful
- The ubox set (green) is the *best you can do for a given precision*
- Uboxes *reveal* ill-posed nature… yet provide solution anyway
- Works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

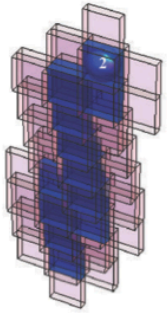


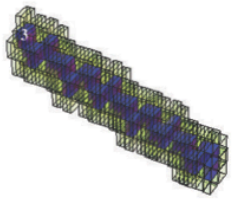
Other Apps with Ubox Solutions

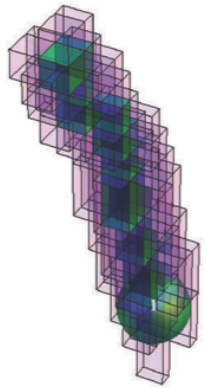
- Photorealistic computer graphics
- *N*-body problems
- Structural analysis
- Laplace's equation
- Perfect gas models without *statistical* mechanics











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

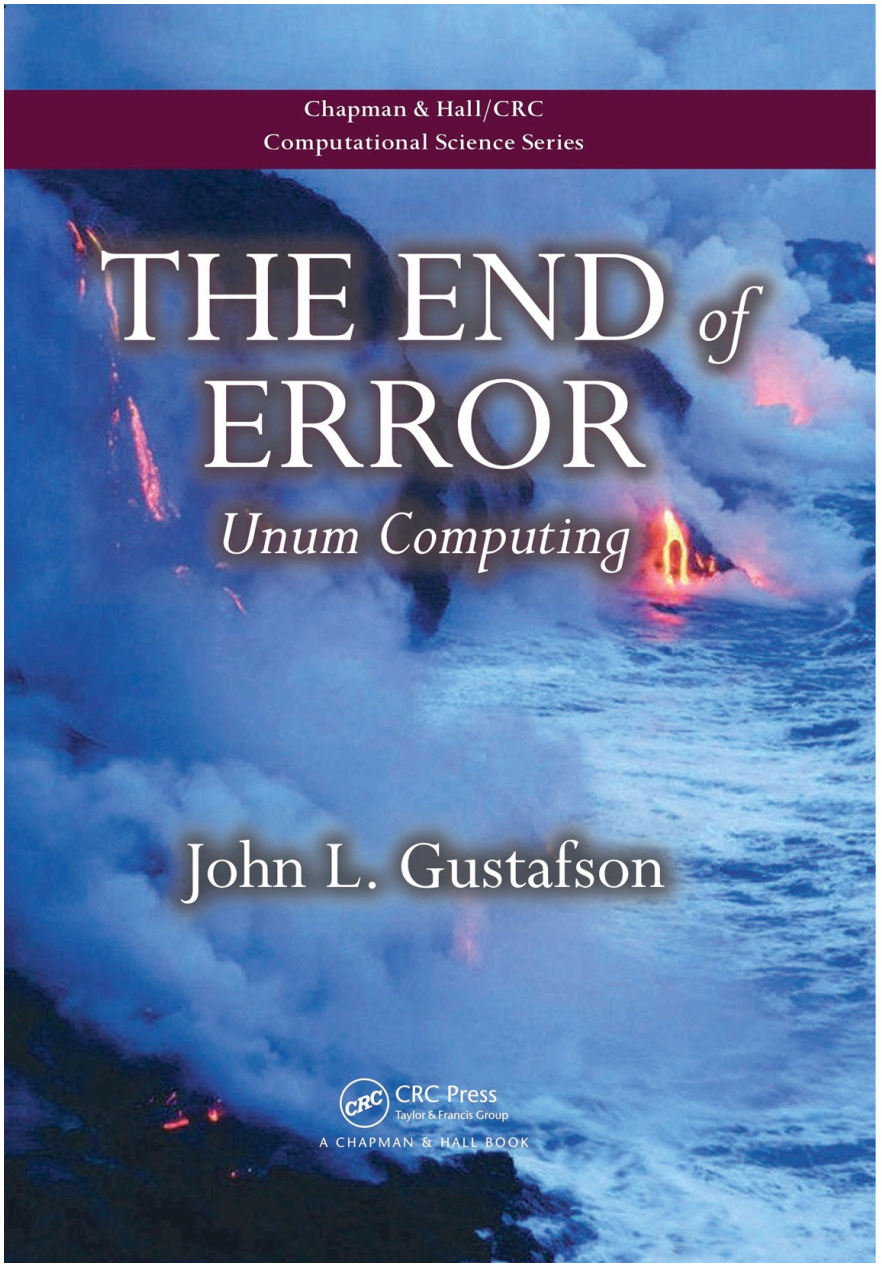
- *Too much energy and power needed per calculation*
  - Unums cut the main energy hog (memory transfers) by about 50%
- *More hardware parallelism than we know how to use*
  - Uboxes reveal vast sources of *data* parallelism, the easiest kind
- *Not enough bandwidth (the "memory wall")*
  - More use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

Revisiting the Big Challenges-2

- *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

The End of Error

- A complete text on unums and uboxes was published Feb 2015 (CRC Press)
- Aimed at *general* reader; mathematicians hate its casual, accessible style
- Complete prototype environment is available as free *Mathematica* notebook through publisher
- Amazon #1 Best Seller in Number Systems; they keep running out of copies



Work in Progress

- Unum data types are now in Julia (funded by A*STAR)
  - Alan Edelman, MIT
  - Viral Shah, Julia Computing
  - Deepak Vinchhi, Julia Computing
  - Isaac Yonemoto, REX Computing
  - Jeffrey Sarnoff, Diadem Special Projects
  - Job Van Der Zwan, julia-users@googlegroups.com
  - Thomas Breloff, Cointegrated Technologies
  - 61 repository results on github, too many to list here
- A Python version of the prototype environment exists: https://github.com/dpsanders/pyunum

Work in Progress

- Lawrence Livermore National Lab
  - David Jefferson
  - Charles Reynolds
  - Robin Goldstone
  - Dan Quinlan
  - Markus Schordan
  - 5 others
- University of California Santa Barbara
  - Carlos Maltzahn is creating open source community for unums; student hired
- Karlsruhe Institute of Technology
  - Ulrich Kulisch has formalized unum definition, proposed fixed-size implementation
- IEEE
  - 2015 President Tom Conte has urged creation of a unum IEEE Standard

Work in Progress

- Convert *Mathematica* prototype into a C library
  - Elavarasi Manogaran and Himeshi DeSilva, NUS
  - Basic arithmetic + – * / working as of last week. 2 million ops per second.
- FPGA version
  - A*STAR intends to fund and staff one of several efforts
- 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)
  - Fundamentally different kind of processor design, incl. integer ops
  - REX Computing will put unums in next version of their CPU
  - A different unum approach discovered in February 2016 is super-fast and **hardware-friendly**. Hear about it in tomorrow's talk!

The ocean is already starting to boil. Thank you!