“THE GREAT DEBATE”: UNUM ARITHMETIC POSITION STATEMENT

Prof. John L. Gustafson

A*STAR-CRC and National University of Singapore

July 12, 2016

ARITH23, Santa Clara California

Why this debate?

Why this debate?

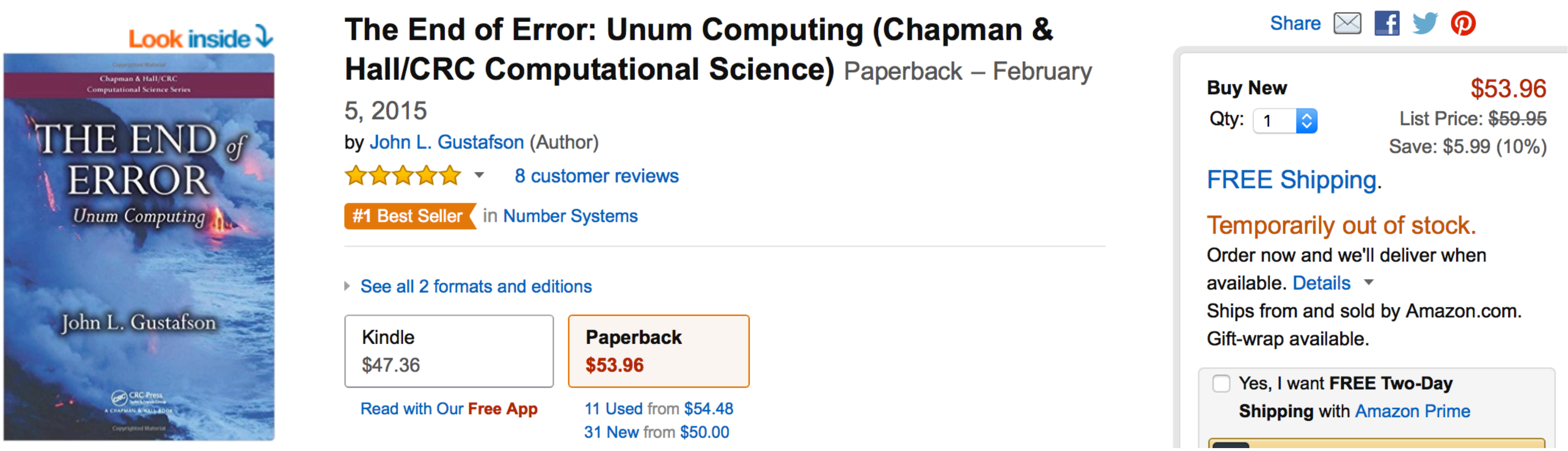
- *The End of Error* had dozens of reviewers, including David Bailey, Horst Simon, Gordon Bell, John Gunnels…

Why this debate?

- *The End of Error* had dozens of reviewers, including David Bailey, Horst Simon, Gordon Bell, John Gunnels…
- Kahan has had the manuscript since November 2013 but ceased email conversation about its content in July 2014

Why this debate?

- *The End of Error* had dozens of reviewers, including David Bailey, Horst Simon, Gordon Bell, John Gunnels…
- Kahan has had the manuscript since November 2013 but ceased email conversation about its content in July 2014
- Then this happened (Amazon.com):



The End of Error: Unum Computing (Chapman & Hall/CRC Computational Science) Paperback – February 5, 2015

by John L. Gustafson (Author)

★★★★★ 8 customer reviews

#1 Best Seller in Number Systems

See all 2 formats and editions

Kindle	Paperback
$47.36	$53.96
Read with Our Free App	11 Used from $54.48 31 New from $50.00

Share

Buy New $53.96

Qty: 1

List Price: ~~$59.95~~

Save: $5.99 (10%)

FREE Shipping.

Temporarily out of stock.

Order now and we'll deliver when available. Details

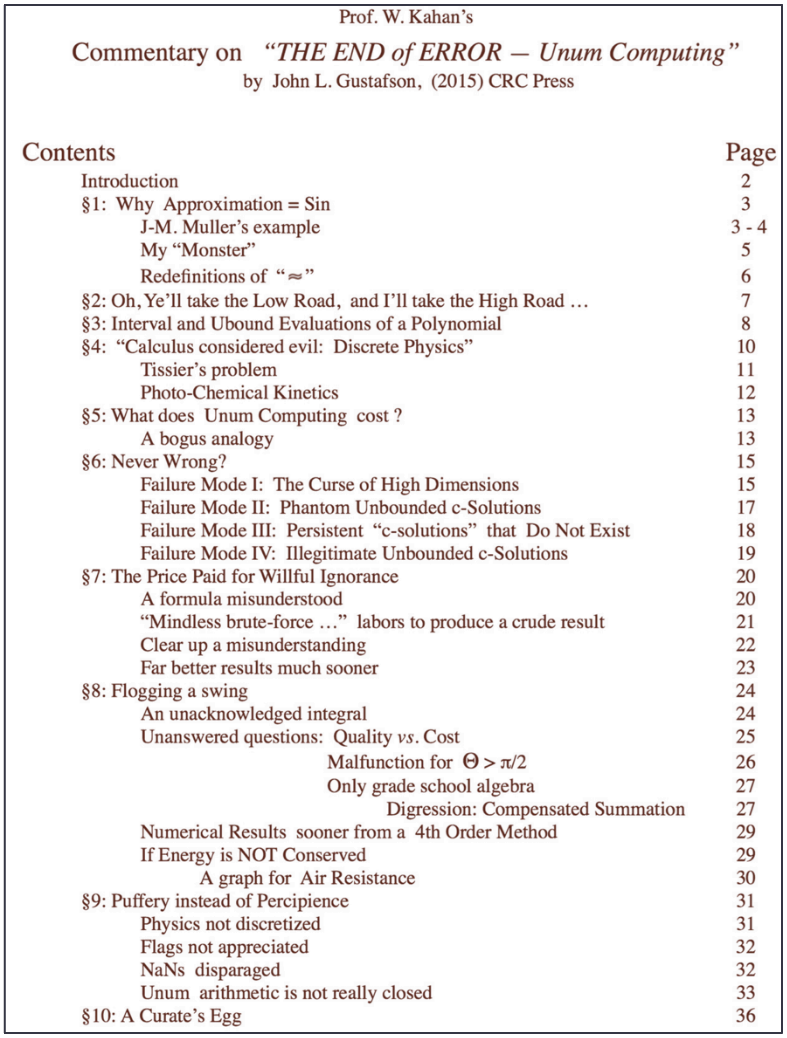
Ships from and sold by Amazon.com.

Gift-wrap available.

□ Yes, I want **FREE Two-Day Shipping** with Amazon Prime

The Wrath of Kahan: A Bitter Blog

- Kahan no longer submits papers to journals.



Prof. W. Kahan's

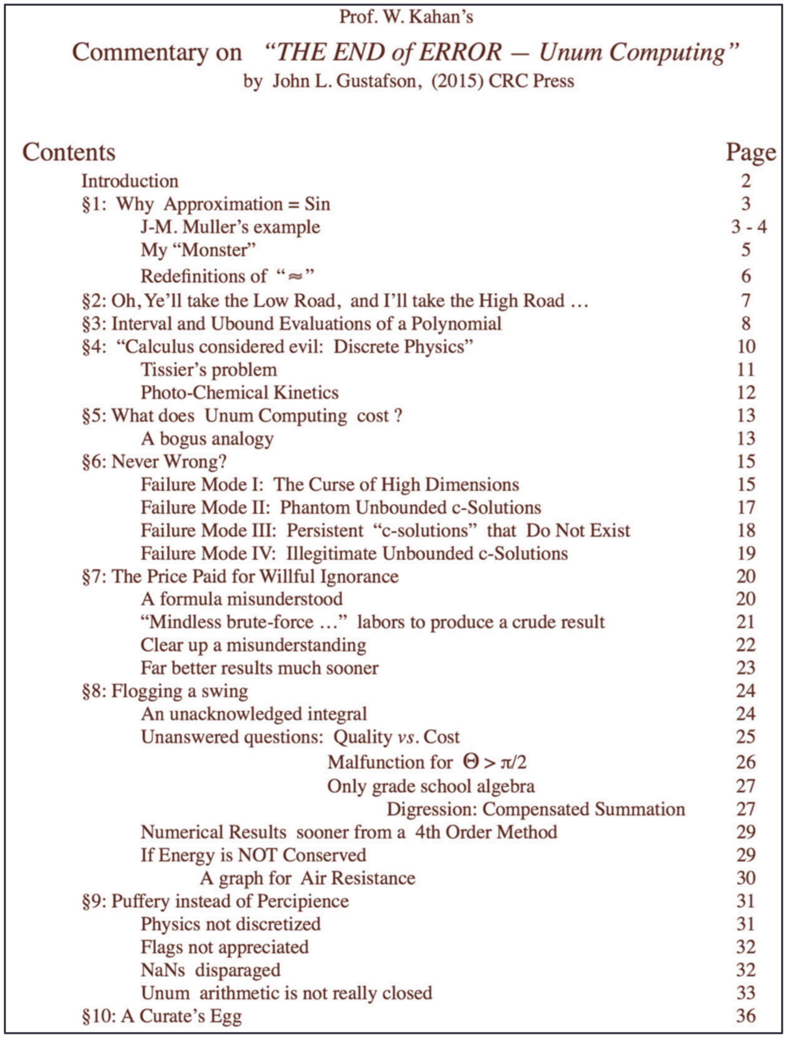
Commentary on *"THE END of ERROR — Unum Computing"*

by John L. Gustafson, (2015) CRC Press

Contents | Page

The Wrath of Kahan: A Bitter Blog

- Kahan no longer submits papers to journals.
- Instead he prepares diatribes and blogs them, as “work in progress.”



Prof. W. Kahan’s

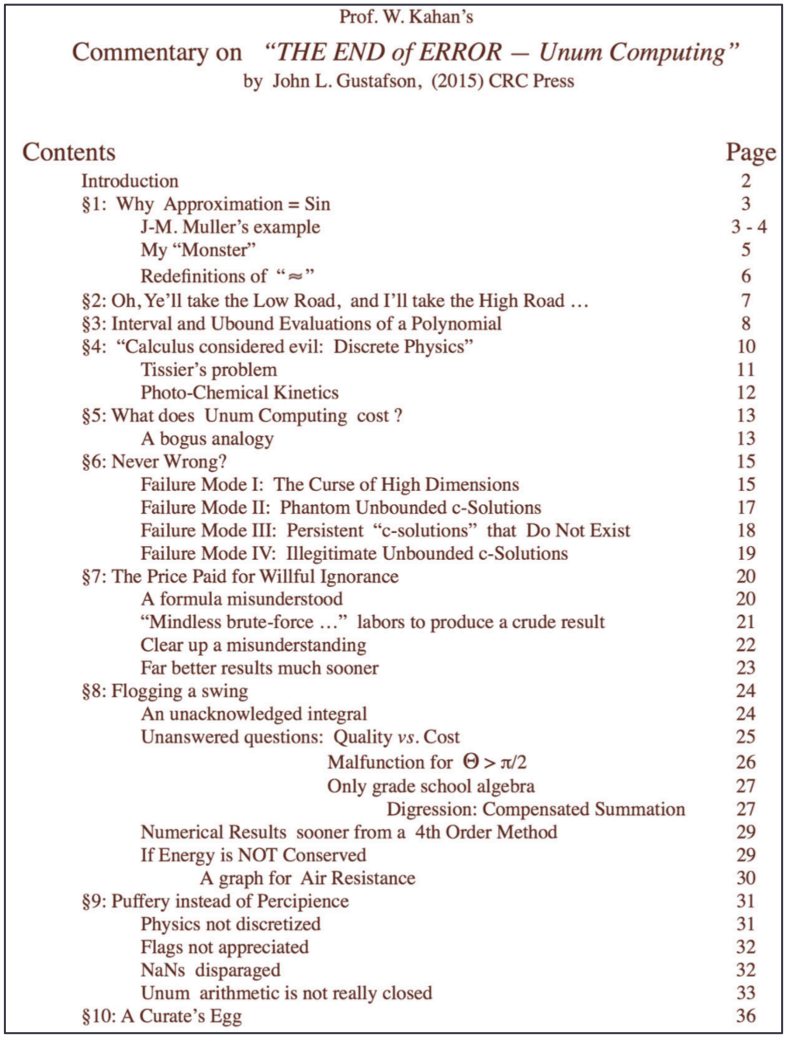
Commentary on *“THE END of ERROR — Unum Computing”*

by John L. Gustafson, (2015) CRC Press

Contents Page

The Wrath of Kahan: A Bitter Blog

- Kahan no longer submits papers to journals.
- Instead he prepares diatribes and blogs them, as “work in progress.”
- This issue is too important to be left to the bickering of two old men.



Prof. W. Kahan’s

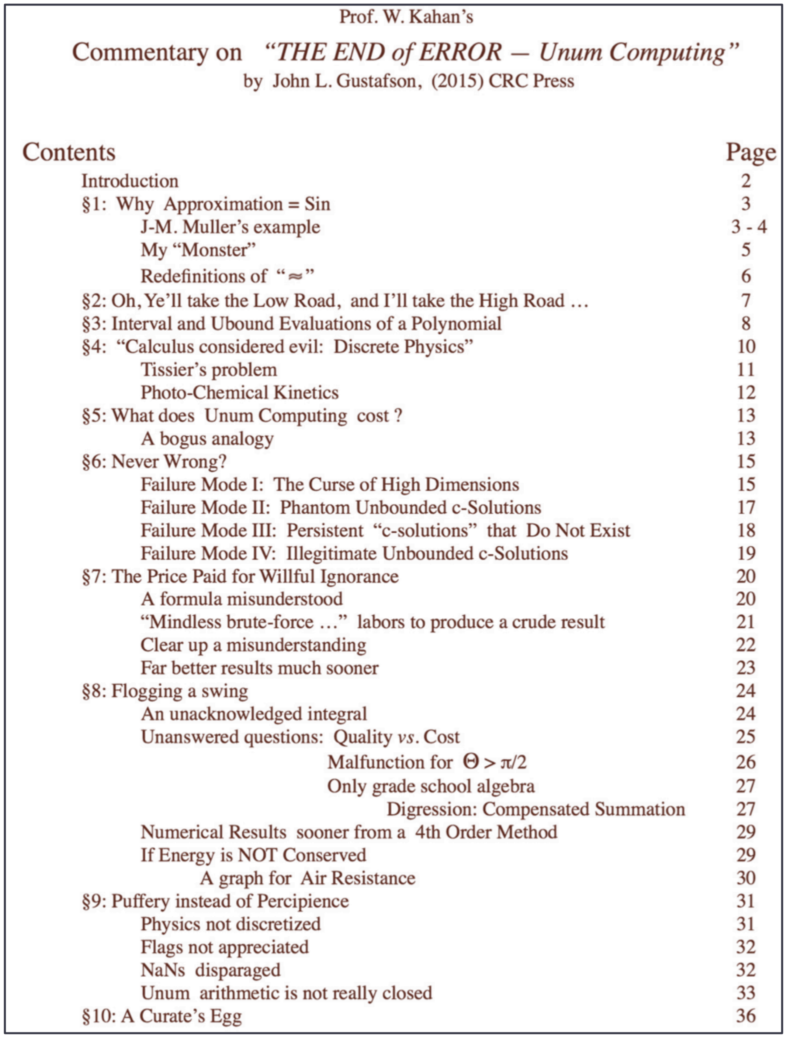
Commentary on *“THE END of ERROR — Unum Computing”*

by John L. Gustafson, (2015) CRC Press

Contents — Page

The Wrath of Kahan: A Bitter Blog

- Kahan no longer submits papers to journals.
- Instead he prepares diatribes and blogs them, as “work in progress.”
- This issue is too important to be left to the bickering of two old men.
- He was kind enough to share with me the 38-page attack he wants to post about *The End of Error: Unum Arithmetic.*



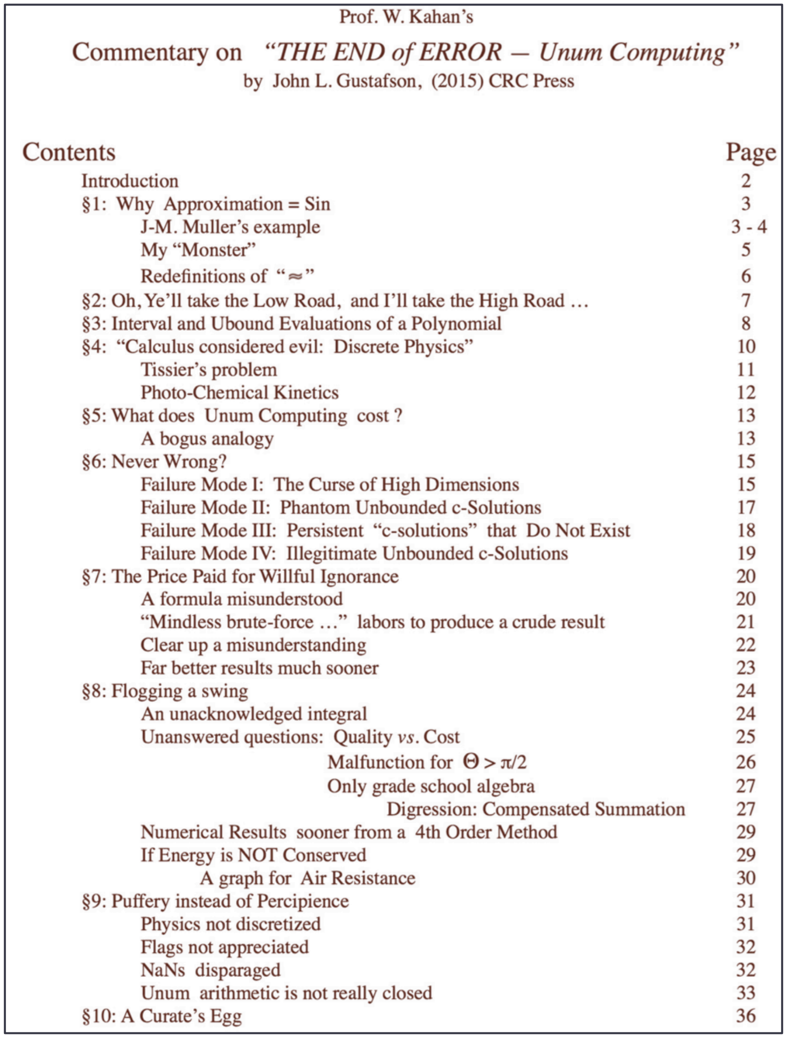
Prof. W. Kahan’s

Commentary on *“THE END of ERROR — Unum Computing”*
by John L. Gustafson, (2015) CRC Press

Contents Page

The Wrath of Kahan: A Bitter Blog

- Kahan no longer submits papers to journals.
- Instead he prepares diatribes and blogs them, as “work in progress.”
- This issue is too important to be left to the bickering of two old men.
- He was kind enough to share with me the 38-page attack he wants to post about *The End of Error: Unum Arithmetic.*
- I will respond in part here.



Prof. W. Kahan’s

Commentary on *“THE END of ERROR — Unum Computing”*

by John L. Gustafson, (2015) CRC Press

Contents Page

“Variable bit size is too expensive”

“Variable bit size is too expensive”

- The utag serves as a linked-list pointer for packing

“Variable bit size is too expensive”

- The utag serves as a linked-list pointer for packing
- “Chapter 7: Fixed-size unum storage” pp. 93–102

“Variable bit size is too expensive”

- The utag serves as a linked-list pointer for packing
- “Chapter 7: Fixed-size unum storage” pp. 93–102
- Energy/power savings still possible with unpacked form

“Variable bit size is too expensive”

- The utag serves as a linked-list pointer for packing
- “Chapter 7: Fixed-size unum storage” pp. 93–102
- Energy/power savings still possible with unpacked form
- Here is an example Kahan calls “a bogus analogy”:

“Variable bit size is too expensive”

- The utag serves as a linked-list pointer for packing
- “Chapter 7: Fixed-size unum storage” pp. 93–102
- Energy/power savings still possible with unpacked form
- Here is an example Kahan calls “a bogus analogy”:

Courier, 16 point

> “Unums offer the same trade-off versus floats as variable-width versus fixed-width typefaces: Harder for the design engineer and more logic for the computer, but superior for *everyone else* in terms of usability, compactness, and overall cost.” (page 193)

“Variable bit size is too expensive”

- The utag serves as a linked-list pointer for packing
- “Chapter 7: Fixed-size unum storage” pp. 93–102
- Energy/power savings still possible with unpacked form
- Here is an example Kahan calls “a bogus analogy”:

Courier, 16 point

“Unums offer the same trade-off versus floats as variable-width versus fixed-width typefaces: Harder for the design engineer and more logic for the computer, but superior for *everyone else* in terms of usability, compactness, and overall cost.” (page 193)

Times, 16 point

“Unums offer the same trade-off versus floats as variable-width versus fixed-width typefaces: Harder for the design engineer and more logic for the computer, but superior for *everyone else* in terms of usability, compactness, and overall cost.” (page 193)

Willful Misunderstanding

“Bunkum! Gustafson has confused the way text is printed, or displayed on today’s bit-mapped screens, with the way text is stored in files and in DRAM memory by word-processor software. …Text stored in variable-width characters **would occupy more DRAM memory**, not less, as we shall see.” *(boldface mine)*

Willful Misunderstanding

"Bunkum! Gustafson has confused the way text is printed, or displayed on today's bit-mapped screens, with the way text is stored in files and in DRAM memory by word-processor software. …Text stored in variable-width characters **would occupy more DRAM memory**, not less, as we shall see." *(boldface mine)*

Kahan may be **unique** in his misreading. Other readers understand that variable width saves *display space* at the cost of more computing. The analogy is that unums save *storage space* at the cost of more computing.

Willful Misunderstanding

"Bunkum! Gustafson has confused the way text is printed, or displayed on today's bit-mapped screens, with the way text is stored in files and in DRAM memory by word-processor software. …Text stored in variable-width characters **would occupy more DRAM memory**, not less, as we shall see." *(boldface mine)*

Kahan may be **unique** in his misreading. Other readers understand that variable width saves *display space* at the cost of more computing. The analogy is that unums save *storage space* at the cost of more computing.

The "willful misunderstanding" technique: Misread a statement so **it becomes one that can be shown wrong.**

Willful Misunderstanding

"Bunkum! Gustafson has confused the way text is printed, or displayed on today's bit-mapped screens, with the way text is stored in files and in DRAM memory by word-processor software. …Text stored in variable-width characters **would occupy more DRAM memory**, not less, as we shall see." *(boldface mine)*

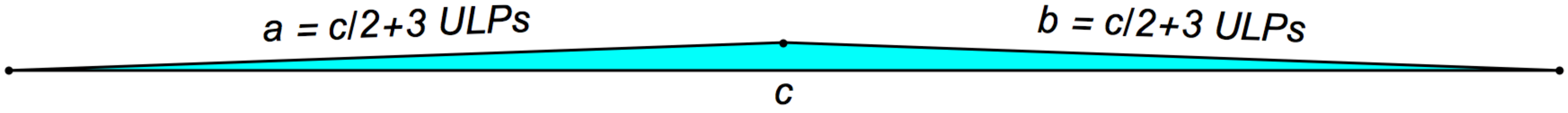
Kahan may be **unique** in his misreading. Other readers understand that variable width saves *display space* at the cost of more computing. The analogy is that unums save *storage space* at the cost of more computing.

The "willful misunderstanding" technique: Misread a statement so **it becomes one that can be shown wrong.**

Now imagine ***38 pages*** of similar attacks on things that were also not said.

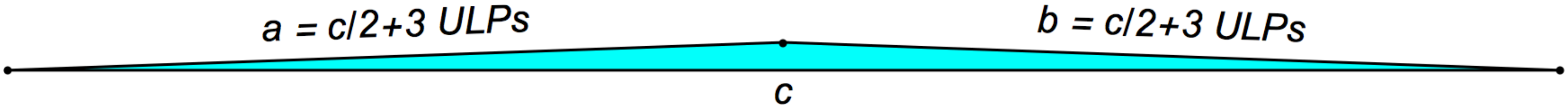
Let's try a classic Kahan example

Find the area of a triangle with sides *a*, *b*, *c* where *a* and *b* are only 3 ULPs longer than half the length of *c*.



Let’s try a classic Kahan example

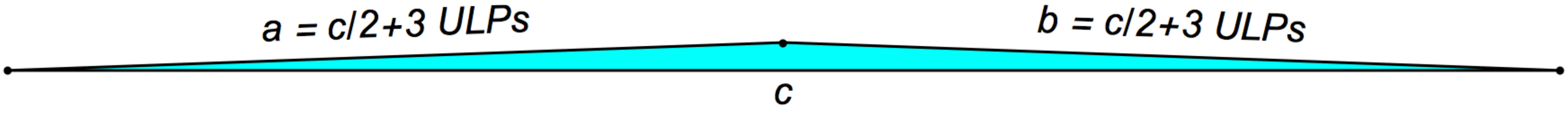
Find the area of a triangle with sides *a*, *b*, *c* where *a* and *b* are only 3 ULPs longer than half the length of *c*.



Try the formula Area = $\sqrt{s(s-a)(s-b)(s-c)}$ where $s = \dfrac{a+b+c}{2}$

Let's try a classic Kahan example

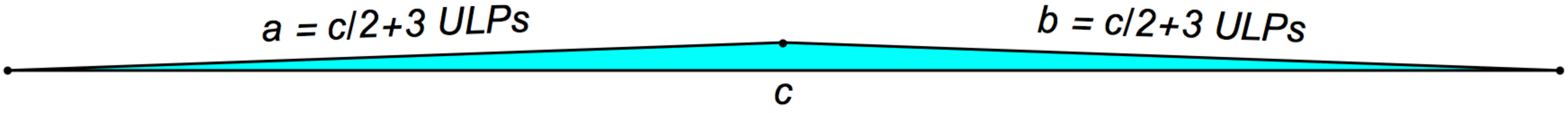
Find the area of a triangle with sides *a*, *b*, *c* where *a* and *b* are only 3 ULPs longer than half the length of *c*.



Try the formula Area = $\sqrt{s(s-a)(s-b)\underline{(s-c)}}$ where $s = \dfrac{a+b+c}{2}$

Let’s try a classic Kahan example

Find the area of a triangle with sides *a*, *b*, *c* where *a* and *b* are only 3 ULPs longer than half the length of *c*.

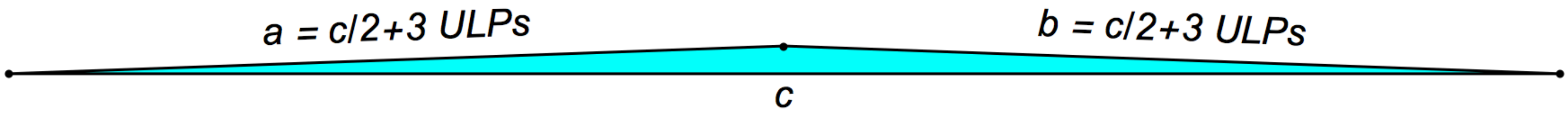


Try the formula Area = $\sqrt{s(s-a)(s-b)\underline{(s-c)}}$ where $s = \dfrac{a+b+c}{2}$

IEEE Quad Precision (128 bits, 34 decimals): Let $a = b = 7/2 + 3\cdot 2^{-111}$, $c = 7$.

Let’s try a classic Kahan example

Find the area of a triangle with sides *a*, *b*, *c* where *a* and *b* are only 3 ULPs longer than half the length of *c.*



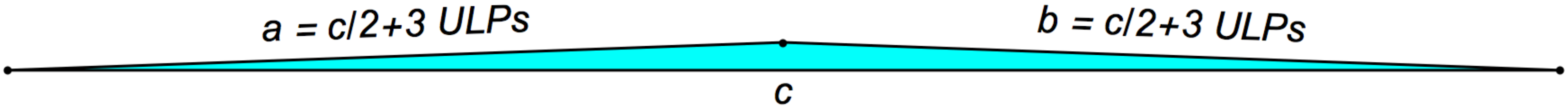
Try the formula Area = $\sqrt{s(s-a)(s-b)\underline{(s-c)}}$ where $s = \dfrac{a+b+c}{2}$

IEEE Quad Precision (128 bits, 34 decimals): Let *a* = *b* = 7/2 + 3·2^{-111}, *c* = 7.

If *c* is 7 light years long, 3 ULPs is ~1/200 the diameter of a proton. The **correct** area is about 55 times the surface area of the earth. To 34 decimals:

Let’s try a classic Kahan example

Find the area of a triangle with sides *a*, *b*, *c* where *a* and *b* are only 3 ULPs longer than half the length of *c*.



Try the formula Area = $\sqrt{s(s-a)(s-b)\underline{(s-c)}}$ where $s = \dfrac{a+b+c}{2}$

IEEE Quad Precision (128 bits, 34 decimals): Let $a = b = 7/2 + 3 \cdot 2^{-111}$, $c = 7$.

If *c* is 7 light years long, 3 ULPs is ~1/200 the diameter of a proton. The **correct** area is about 55 times the surface area of the earth. To 34 decimals:

$3.147842048749004252358852654945507\cdots \times 10^{-16}$ square light years.

Quad-precision float result

Quad-precision float result

- IEEE Quad float gets *1 digit* right: $3.634814908423321347259205161580577\cdots \times 10^{-16}$.

Quad-precision float result

- IEEE Quad float gets *1 digit* right: 3.634814908423321347259205161580577$^{\cdots}$×10^{-16}.
- Error is about 15 percent, or 252 *peta*-ULPs.

Quad-precision float result

- IEEE Quad float gets *1 digit* right:
  $3.634814908423321347259205161580577^{\cdots}\times 10^{-16}$.
- Error is about 15 percent, or 252 *peta*-ULPs.
- **Result does not admit any error, nor bound it.**

Quad-precision float result

- IEEE Quad float gets *1 digit* right:
  $3.634814908423321347259205161580577\cdots\times10^{-16}$.
- Error is about 15 percent, or 252 *peta*-ULPs.
- **Result does not admit any error, nor bound it.**
- Kahan's approach: Sort the sides so $a \geq b \geq c$ and rewrite the formula as

$$Area = \frac{\sqrt{(a+(b+c))(c-(a-b))(c+(a-b))(a+(b-c))}}{4}$$

Quad-precision float result

- IEEE Quad float gets *1 digit* right: 3.634814908423321347259205161580577···×10^–16.
- Error is about 15 percent, or 252 *peta*-ULPs.
- **Result does not admit any error, nor bound it.**
- Kahan's approach: Sort the sides so *a* ≥ *b* ≥ *c* and rewrite the formula as

$$Area = \frac{\sqrt{(a+(b+c))(c-(a-b))(c+(a-b))(a+(b-c))}}{4}$$

This is within 11 ULPs of the correct area, but it takes **hours** to figure out such an approach.

It also uses twice as many operations, but that's not the issue: it's the *people cost* of the approach.

Unum approach to the thin triangle

Unum approach to the thin triangle

- Use no more than 128 bits per number, but *adjustable*

Unum approach to the thin triangle

- Use no more than 128 bits per number, but *adjustable*
- Exponent can be 1 to 16 bits (wider range than quad)

Unum approach to the thin triangle

- Use no more than 128 bits per number, but *adjustable*
- Exponent can be 1 to 16 bits (wider range than quad)
- Fraction can be 1 to 128 bits, plus the hidden bit (higher precision than quad)

Unum approach to the thin triangle

- Use no more than 128 bits per number, but *adjustable*
- Exponent can be 1 to 16 bits (wider range than quad)
- Fraction can be 1 to 128 bits, plus the hidden bit (higher precision than quad)
- Result is a *rigorous bound* accurate to 31 decimals:

Unum approach to the thin triangle

- Use no more than 128 bits per number, but *adjustable*
- Exponent can be 1 to 16 bits (wider range than quad)
- Fraction can be 1 to 128 bits, plus the hidden bit (higher precision than quad)
- Result is a *rigorous bound* accurate to 31 decimals:

$3.14784204890042523588526549455070\cdots\times 10^{-16} < \text{Area} <$
$3.14784204890042523588526549455139\cdots\times 10^{-16}$

Unum approach to the thin triangle

- Use no more than 128 bits per number, but *adjustable*
- Exponent can be 1 to 16 bits (wider range than quad)
- Fraction can be 1 to 128 bits, plus the hidden bit (higher precision than quad)
- Result is a *rigorous bound* accurate to 31 decimals:

$3.14784204890042523588526549455070\cdots\times 10^{-16} < \text{Area} <$
$3.14784204890042523588526549455139\cdots\times 10^{-16}$

The size of that bound is the area of a square 8 nanometers on a side.

No need to rewrite the formula.

Summary of comparison

Format Capabilities	Quad-precision IEEE floats	Unums, {4,7} environment
Dynamic Range	~6.5×10^{-4966} to 1.2×10^{4932}	~8.2×10^{-9903} to ~2.8×10^{9864}
Precision	~34.0 decimal digits	~38.8 decimal digits

Summary of comparison

Format Capabilities	**Quad-precision IEEE floats**	**Unums, {4,7} environment**
Dynamic Range	~6.5×10^{-4966} to 1.2×10^{4932}	~8.2×10^{-9903} to ~2.8×10^{9864}
Precision	~34.0 decimal digits	~38.8 decimal digits

Results on thin triangle	**Quad-precision IEEE floats**	**Unums, {4,7} environment**
Maximum bits used	128	128
Average bits used	128	**90**
Result	Area **=** $3.64814908423321347259205161580577\times10^{-16}$	$3.14784204874900425235885265494550 7\times10^{-16}$ **<** Area **<** $3.14784204874900425235885265494551 4\times10^{-16}$
Type of information loss	Invisible error, very hard to debug	Rigorous bound, easy to debug if needed
Error / bound size	~4×10^{15} meters^2	~6×10^{-17} meters^2

Another “Rewrite it this way” example

From my book, to show why round-to-nearest might not be random and how unums can self-manage accuracy:

```
#include < stdio.h >
float sumtester () {
        float sum; int i;
        sum = 0.0;
        for (i = 0; i < 1000000000; i++) {sum = sum + 1.0;}
        printf (“%f\n”, sum);
}
```

Another “Rewrite it this way” example

From my book, to show why round-to-nearest might not be random and how unums can self-manage accuracy:

```
#include < stdio.h >
float sumtester () {
        float sum; int i;
        sum = 0.0;
        for (i = 0; i < 1000000000; i++) {sum = sum + 1.0;}
        printf (“%f\n”, sum);
}
```

In trying to count to a billion, IEEE floats (32-bit) produce 16777216.

Another “Rewrite it this way” example

From my book, to show why round-to-nearest might not be random and how unums can self-manage accuracy:

```
#include < stdio.h >
float sumtester () {
        float sum; int i;
        sum = 0.0;
        for (i = 0; i < 1000000000; i++) {sum = sum + 1.0;}
        printf (“%f\n”, sum);
}
```

In trying to count to a billion, IEEE floats (32-bit) produce 16777216.

“Compensated Summation will be illustrated by application to a silly sum Gustafson uses on p. 120 to justify what unums do as intervals do, namely, convey numerical uncertainty via their widths.”

Another “Rewrite it this way” example

From my book, to show why round-to-nearest might not be random and how unums can self-manage accuracy:

```
#include < stdio.h >
float sumtester () {
        float sum; int i;
        sum = 0.0;
        for (i = 0; i < 1000000000; i++) {sum = sum + 1.0;}
        printf (“%f\n”, sum);
}
```

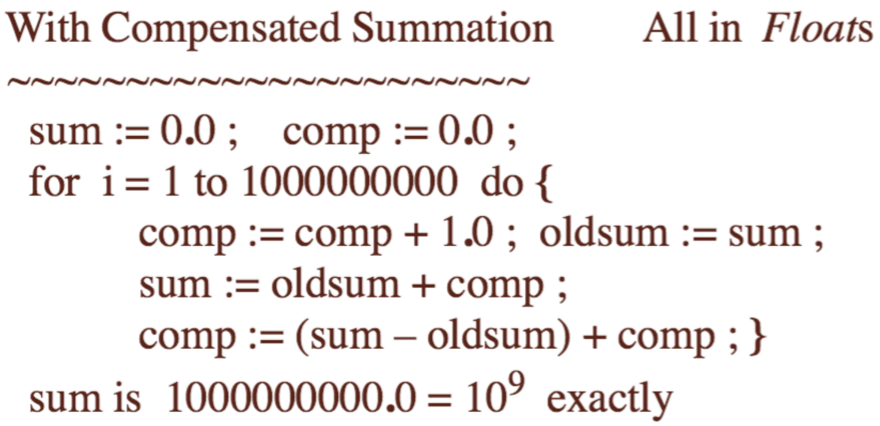
In trying to count to a billion, IEEE floats (32-bit) produce 16777216.

“Compensated Summation will be illustrated by application to a silly sum Gustafson uses on p. 120 to justify what unums do as intervals do, namely, convey numerical uncertainty via their widths.”

(Misreading. Actually, the example was to show how unums can automatically adjust range and precision to get the exact answer.)

Let’s try Kahan’s suggestion for $\sum_{i=1}^{n} 1$

*Screen shot from Kahan’s paper, n = **10⁹**:*



With Compensated Summation   All in *Floats*

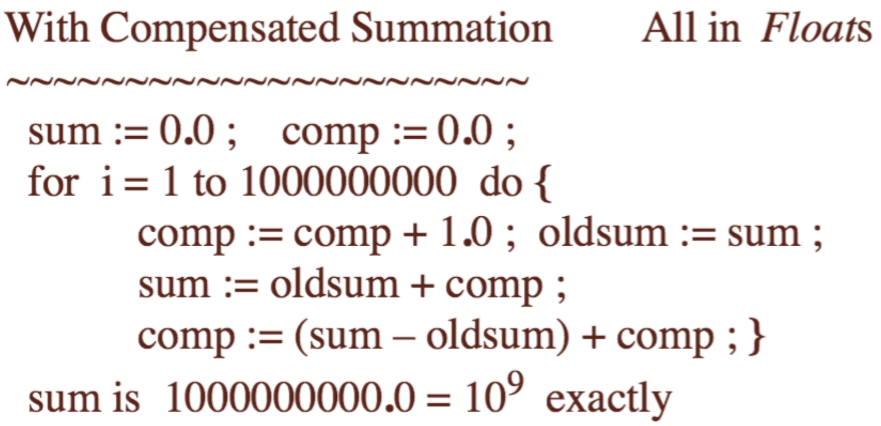
~~~~~~~~~~~~~~~~~~~~~~~

```
sum := 0.0 ;    comp := 0.0 ;
for  i = 1 to 1000000000  do {
         comp := comp + 1.0 ;  oldsum := sum ;
         sum := oldsum + comp ;
         comp := (sum – oldsum) + comp ; }
sum is  1000000000.0 = 10^9  exactly
```
~~~~~~~~~~~~~~~~~~~~~~~

Let’s try Kahan’s suggestion for $\sum_{i=1}^{n} 1$

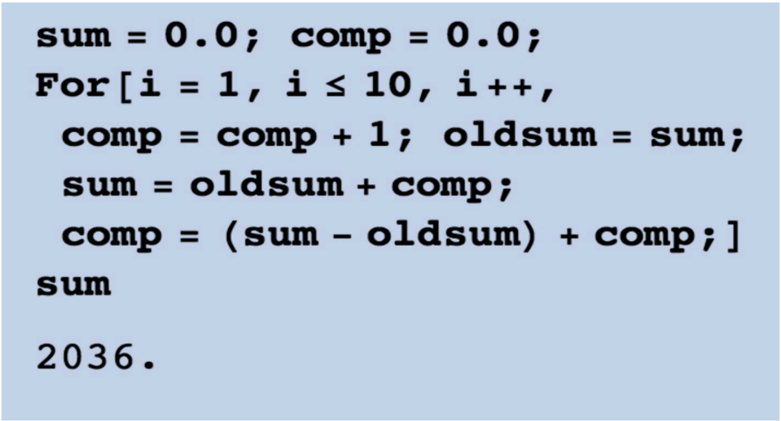
Screen shot from Kahan’s paper, n = ***10^9^*** —

Screen shot from Kahan’s paper, n = $\mathbf{10^9}$*:*



```
With Compensated Summation        All in Floats
~~~~~~~~~~~~~~~~~~~~~~~~~~~~
 sum := 0.0 ;    comp := 0.0 ;
 for  i = 1 to 1000000000  do {
          comp := comp + 1.0 ;  oldsum := sum ;
          sum := oldsum + comp ;
          comp := (sum – oldsum) + comp ; }
 sum is  1000000000.0 = 10^9  exactly
```

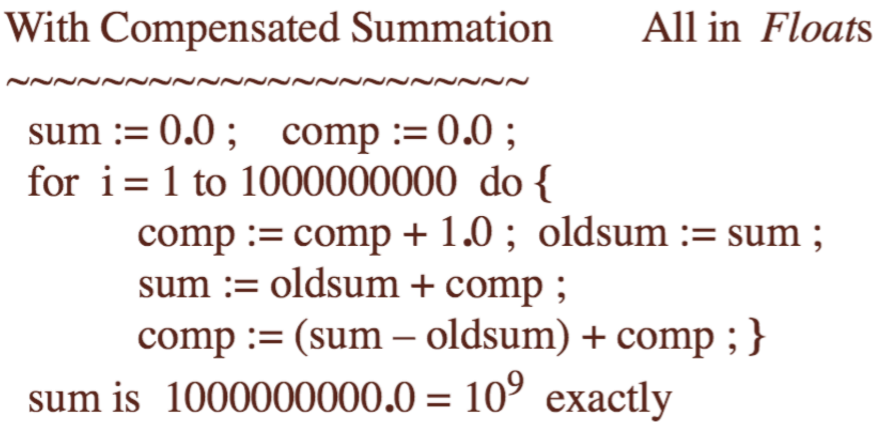
Screen shot from Mathematica test for sum up to n = ***10***



```
sum = 0.0;  comp = 0.0;
For[i = 1, i ≤ 10, i++,
  comp = comp + 1;  oldsum = sum;
  sum = oldsum + comp;
  comp = (sum - oldsum) + comp;]
sum

2036.
```

Let’s try Kahan’s suggestion for $\sum_{i=1}^{n} 1$

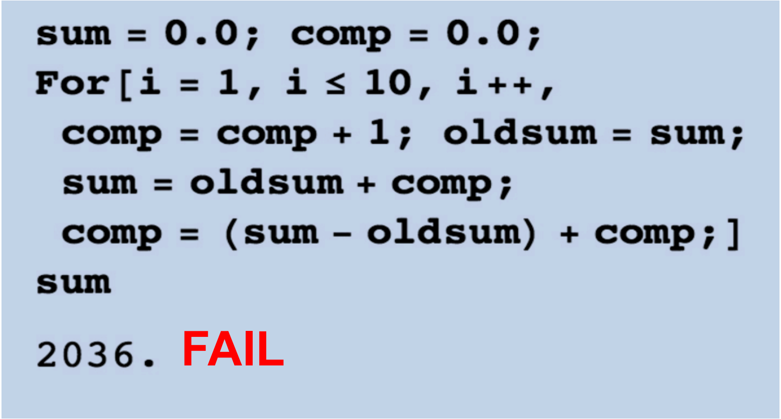
*Screen shot from Kahan’s paper, n = **10⁹**:*



With Compensated Summation   All in *Floats*

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
  sum := 0.0 ;    comp := 0.0 ;
  for  i = 1 to 1000000000  do {
          comp := comp + 1.0 ;  oldsum := sum ;
          sum := oldsum + comp ;
          comp := (sum – oldsum) + comp ; }
  sum is  1000000000.0 = 10^9  exactly
```

*Screen shot from Mathematica test for sum up to n = **10***

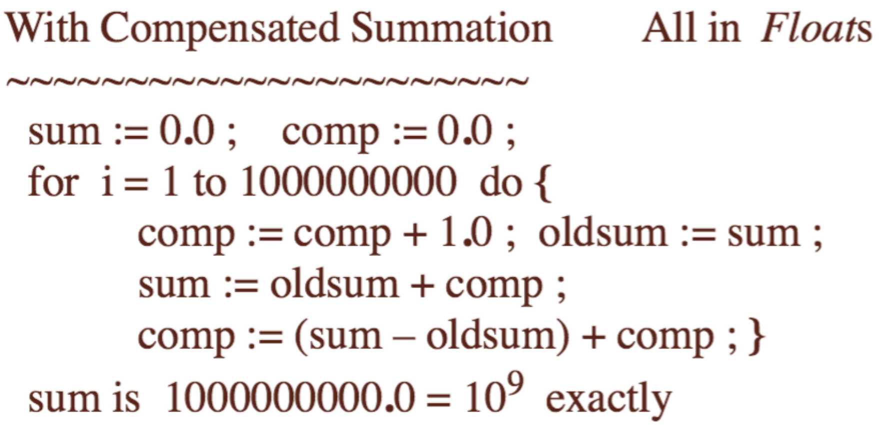


```
sum = 0.0;  comp = 0.0;
For[i = 1, i ≤ 10, i++,
  comp = comp + 1;  oldsum = sum;
  sum = oldsum + comp;
  comp = (sum - oldsum) + comp;]
sum

2036. FAIL
```

(Attempting to sum to 10^9 gives NaN.)

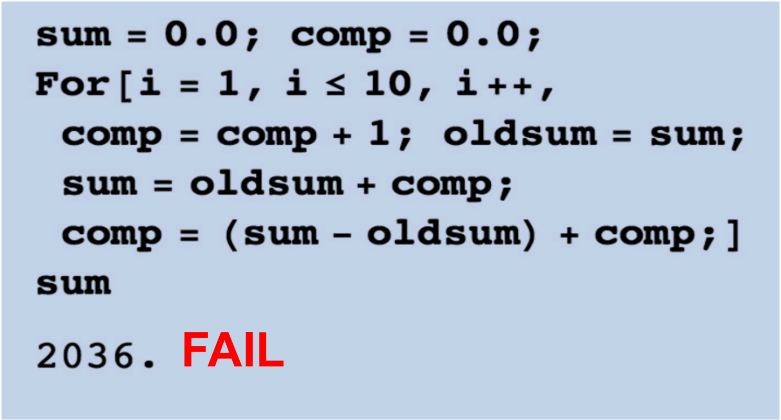
Let’s try Kahan’s suggestion for $\sum_{i=1}^{n} 1$

Screen shot from Kahan’s paper, n = ***10***^{9}*:*



```
With Compensated Summation        All in  Floats
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
 sum := 0.0 ;    comp := 0.0 ;
 for  i = 1 to 1000000000  do {
          comp := comp + 1.0 ;  oldsum := sum ;
          sum := oldsum + comp ;
          comp := (sum – oldsum) + comp ; }
 sum is  1000000000.0 = 10^9  exactly
```

Screen shot from Mathematica test for sum up to n = ***10***



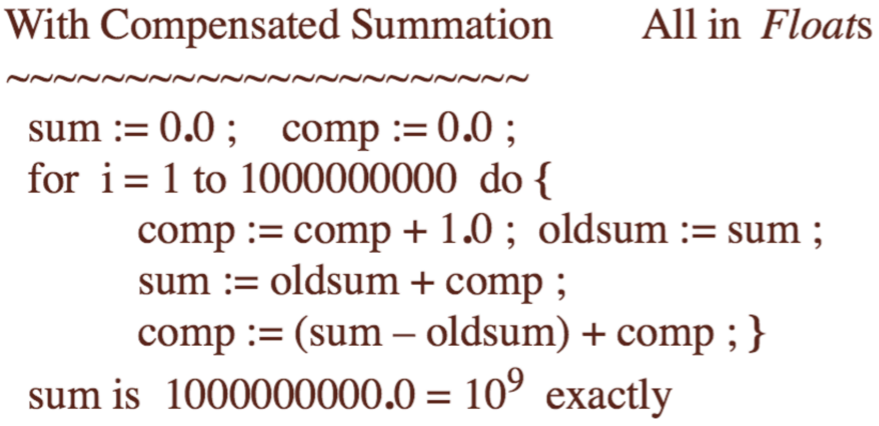
```
sum = 0.0; comp = 0.0;
For[i = 1, i ≤ 10, i++,
  comp = comp + 1; oldsum = sum;
  sum = oldsum + comp;
  comp = (sum - oldsum) + comp;]
sum

2036. FAIL
```

(Attempting to sum to 10^{9} gives NaN.)

- Rewriting code to compensate for rounding is very *error-prone*; **even Kahan didn’t get it right**.

Let’s try Kahan’s suggestion for $\sum_{i=1}^{n} 1$

*Screen shot from Kahan’s paper, n = **10⁹**:*

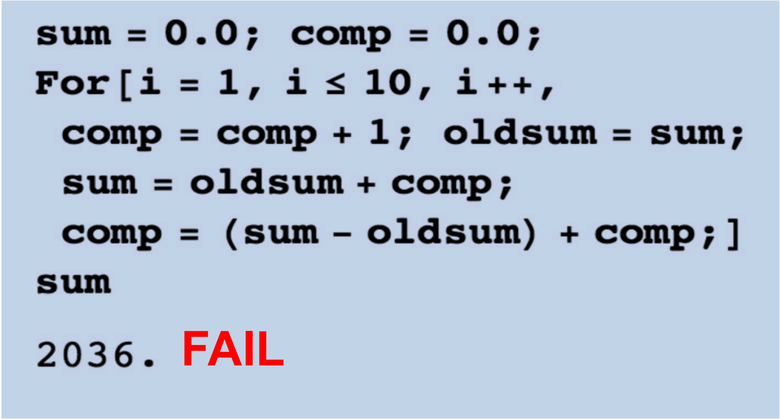


With Compensated Summation   All in *Floats*

~~~~~~~~~~~~~~~~~~~~~~~~

```
sum := 0.0 ;    comp := 0.0 ;
for  i = 1 to 1000000000  do {
         comp := comp + 1.0 ;  oldsum := sum ;
         sum := oldsum + comp ;
         comp := (sum – oldsum) + comp ; }
sum is  1000000000.0 = 10^9  exactly
```

*Screen shot from Mathematica test for sum up to n = **10***



```
sum = 0.0;  comp = 0.0;
For[i = 1, i ≤ 10, i++,
  comp = comp + 1;  oldsum = sum;
  sum = oldsum + comp;
  comp = (sum - oldsum) + comp;]
sum

2036.
```

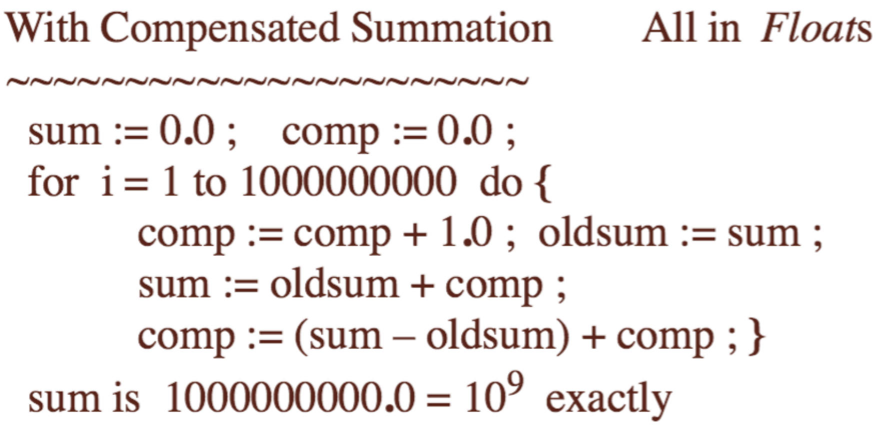
FAIL

(Attempting to sum to 10^9 gives NaN.)

- Rewriting code to compensate for rounding is very *error-prone*; **even Kahan didn’t get it right**.
- Approach uses much more human coding effort and three times as many bits to produce a wildly wrong answer.
~~~~~~~~~~~~~~~~~~~~~~~~

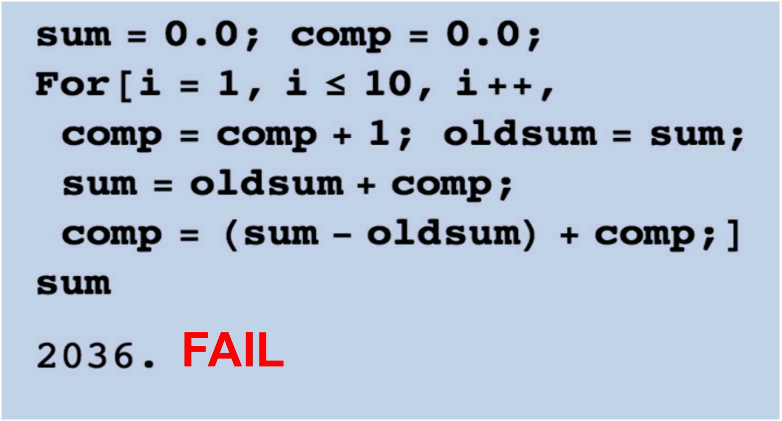
Let’s try Kahan’s suggestion for $\sum_{i=1}^{n} 1$

Screen shot from Kahan’s paper, n = ***10^9****:*



```
With Compensated Summation        All in  Floats
~~~~~~~~~~~~~~~~~~~~~~~~~~~~
  sum := 0.0 ;    comp := 0.0 ;
  for  i = 1 to 1000000000  do {
          comp := comp + 1.0 ;  oldsum := sum ;
          sum := oldsum + comp ;
          comp := (sum – oldsum) + comp ; }
  sum is  1000000000.0 = 10^9  exactly
```

Screen shot from Mathematica test for sum up to n = ***10***



```
sum = 0.0;  comp = 0.0;
For[i = 1, i ≤ 10, i++,
  comp = comp + 1;  oldsum = sum;
  sum = oldsum + comp;
  comp = (sum - oldsum) + comp;]
sum

2036.
```

FAIL

(Attempting to sum to 10^9 gives NaN.)

- Rewriting code to compensate for rounding is very *error-prone*; **even Kahan didn’t get it right**.
- Approach uses much more human coding effort and three times as many bits to produce a wildly wrong answer.
- Examples like this need to be *tested*, not merely *asserted*.

Kahan’s “Monster” Revisited

Verbatim:

```
Real variables      x, y, z ;
Real Function       T(z) := { If z = 0 then 1 else (exp(z) – 1 )/z } ;
Real Function       $Q(y) := | y - \sqrt{}(y^2 + 1) | - 1/( y + \sqrt{}(y^2 + 1) )$ ;
Real Function       $G(x) := T( Q(x)^{2} )$ ;
        For Integer n = 1 to 9999 do Display{ n , G(n) } end do.
```

“$G(x) := T(Q(x)^{2})$ ends up wrongly as 0 instead of 1 . Almost always.”

Kahan’s “Monster” Revisited

Verbatim:

```
Real variables      x, y, z ;
Real Function       T(z) := { If z = 0 then 1 else (exp(z) – 1 )/z } ;
Real Function       Q(y) := | y – $\sqrt{y^2 + 1}$ | – 1/( y + $\sqrt{y^2 + 1}$ ) ;
Real Function       G(x) := T( $Q(x)^2$ ) ;
     For Integer n = 1 to 9999 do Display{ n , G(n) } end do.
```

“G(x) := T($Q(x)^2$) ends up wrongly as 0 instead of 1 . Almost always.”

- Unums got exactly 1, but used “≈” (intersection test) instead of “=”.

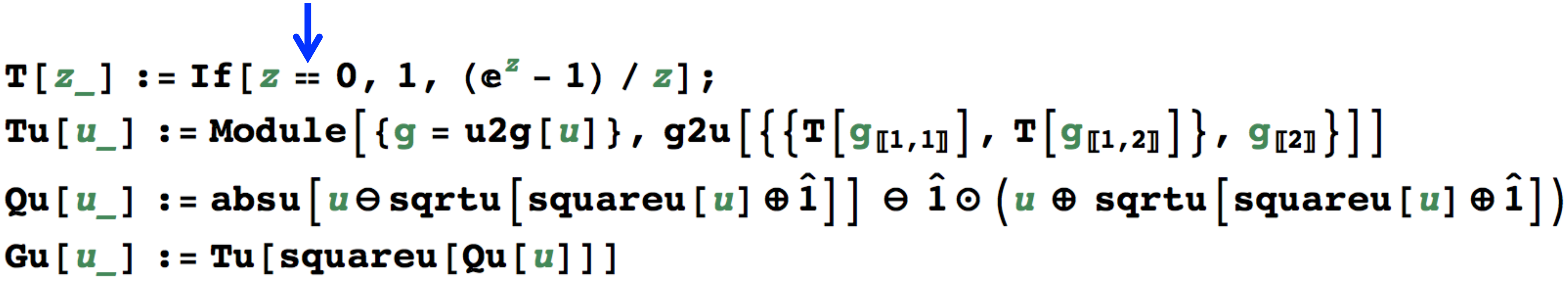
Kahan's "Monster" Revisited

Verbatim:

Real variables x, y, z ;
Real Function T(z) := { If z = 0 then 1 else (exp(z) – 1)/z } ;
Real Function Q(y) := | y – √(y^2 + 1) | – 1/(y + √(y^2 + 1)) ;
Real Function G(x) := T(Q(x)^2) ;
For Integer n = 1 to 9999 do Display{ n , G(n) } end do.

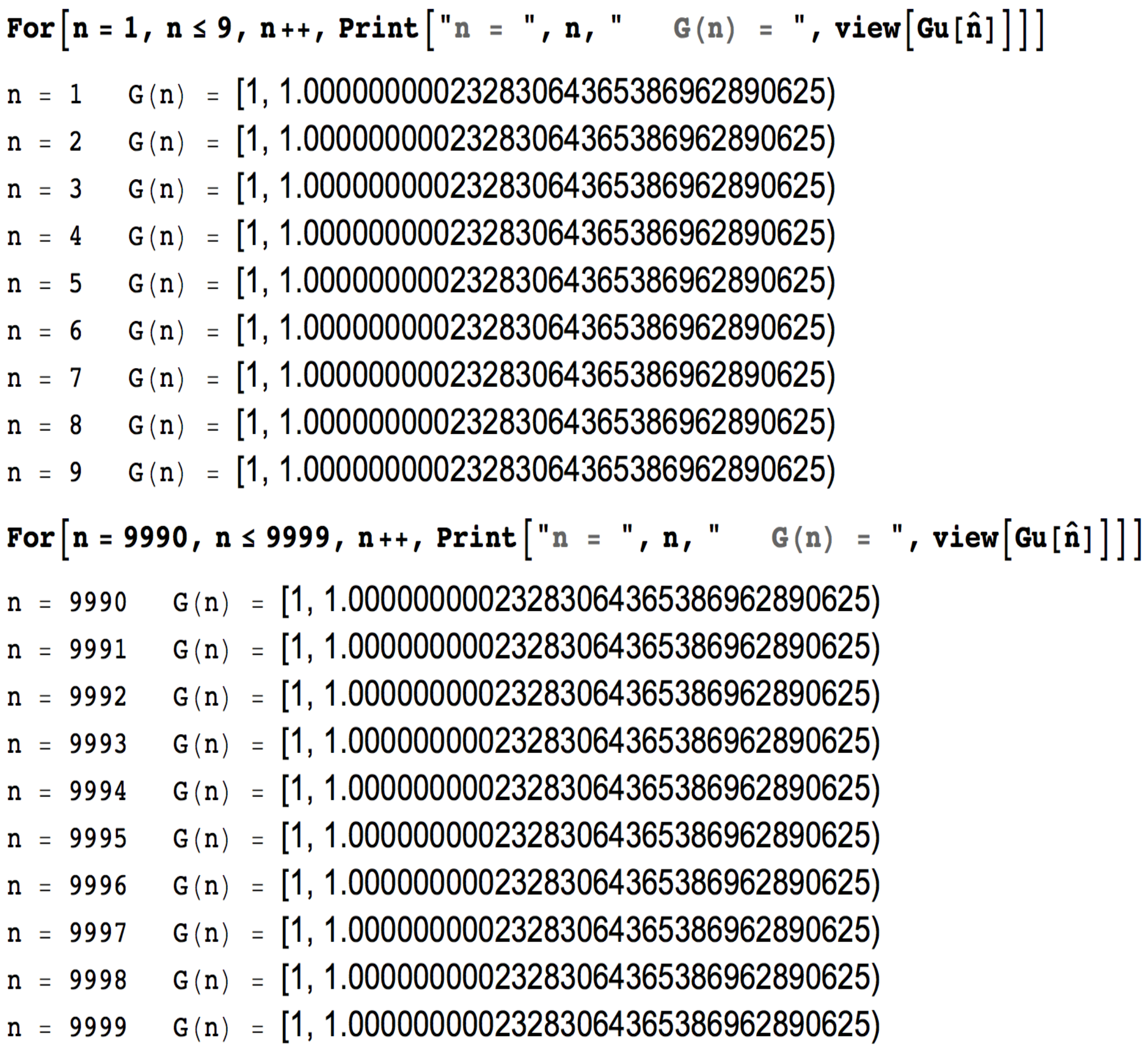
"G(x) := T(Q(x)^2) ends up wrongly as 0 instead of 1 . Almost always."

- Unums got exactly 1, but used "≈" (intersection test) instead of "=".
- Kahan cried "Foul!" so here is a unum version with exactly the specified equality test, which he says will break unums:



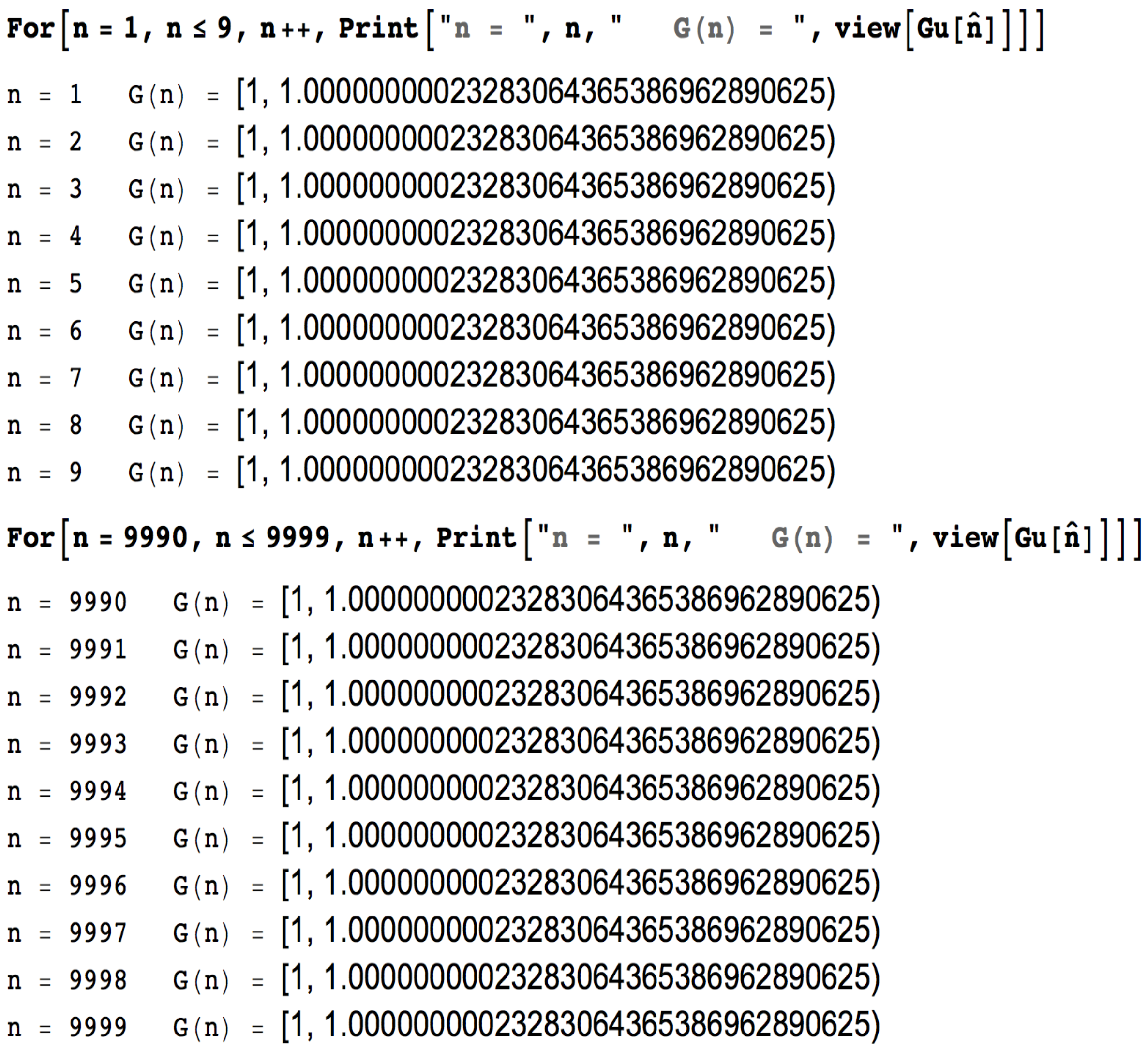
```
T[z_] := If[z == 0, 1, (ⅇ^z - 1) / z];
Tu[u_] := Module[{g = u2g[u]}, g2u[{{T[g[[1,1]]], T[g[[1,2]]]}, g[[2]]}]]
Qu[u_] := absu[u ⊖ sqrtu[squareu[u] ⊕ 1̂]] ⊖ 1̂ ⊙ (u ⊕ sqrtu[squareu[u] ⊕ 1̂])
Gu[u_] := Tu[squareu[Qu[u]]]
```

The result of the “=“ unum version



```
For[n = 1, n ≤ 9, n++, Print["n = ", n, "    G(n) = ", view[Gu[n̂]]]]

n = 1    G(n) = [1, 1.00000000023283064365386962890625)
n = 2    G(n) = [1, 1.00000000023283064365386962890625)
n = 3    G(n) = [1, 1.00000000023283064365386962890625)
n = 4    G(n) = [1, 1.00000000023283064365386962890625)
n = 5    G(n) = [1, 1.00000000023283064365386962890625)
n = 6    G(n) = [1, 1.00000000023283064365386962890625)
n = 7    G(n) = [1, 1.00000000023283064365386962890625)
n = 8    G(n) = [1, 1.00000000023283064365386962890625)
n = 9    G(n) = [1, 1.00000000023283064365386962890625)

For[n = 9990, n ≤ 9999, n++, Print["n = ", n, "    G(n) = ", view[Gu[n̂]]]]

n = 9990    G(n) = [1, 1.00000000023283064365386962890625)
n = 9991    G(n) = [1, 1.00000000023283064365386962890625)
n = 9992    G(n) = [1, 1.00000000023283064365386962890625)
n = 9993    G(n) = [1, 1.00000000023283064365386962890625)
n = 9994    G(n) = [1, 1.00000000023283064365386962890625)
n = 9995    G(n) = [1, 1.00000000023283064365386962890625)
n = 9996    G(n) = [1, 1.00000000023283064365386962890625)
n = 9997    G(n) = [1, 1.00000000023283064365386962890625)
n = 9998    G(n) = [1, 1.00000000023283064365386962890625)
n = 9999    G(n) = [1, 1.00000000023283064365386962890625)
```

The result of the "=" unum version



```
For[n = 1, n ≤ 9, n++, Print["n = ", n, "    G(n) = ", view[Gu[n̂]]]]

n = 1    G(n) = [1, 1.00000000023283064365386962890625)
n = 2    G(n) = [1, 1.00000000023283064365386962890625)
n = 3    G(n) = [1, 1.00000000023283064365386962890625)
n = 4    G(n) = [1, 1.00000000023283064365386962890625)
n = 5    G(n) = [1, 1.00000000023283064365386962890625)
n = 6    G(n) = [1, 1.00000000023283064365386962890625)
n = 7    G(n) = [1, 1.00000000023283064365386962890625)
n = 8    G(n) = [1, 1.00000000023283064365386962890625)
n = 9    G(n) = [1, 1.00000000023283064365386962890625)

For[n = 9990, n ≤ 9999, n++, Print["n = ", n, "    G(n) = ", view[Gu[n̂]]]]

n = 9990    G(n) = [1, 1.00000000023283064365386962890625)
n = 9991    G(n) = [1, 1.00000000023283064365386962890625)
n = 9992    G(n) = [1, 1.00000000023283064365386962890625)
n = 9993    G(n) = [1, 1.00000000023283064365386962890625)
n = 9994    G(n) = [1, 1.00000000023283064365386962890625)
n = 9995    G(n) = [1, 1.00000000023283064365386962890625)
n = 9996    G(n) = [1, 1.00000000023283064365386962890625)
n = 9997    G(n) = [1, 1.00000000023283064365386962890625)
n = 9998    G(n) = [1, 1.00000000023283064365386962890625)
n = 9999    G(n) = [1, 1.00000000023283064365386962890625)
```

Result: tight bounds, [1, 1+ε).

Never zero.

All Kahan had to do was try it. He has all my prototype code at his fingertips.

He did not *test* any of his assertions about what he thought unum arithmetic would do, but *preferred to speculate* that it would fail.

Kahan's *Unum-Targeted* Variation

```
Real Function $G^{o}(x) := T( Q(x)^2 + (10.0^{-300})^{10000\cdot(x+1)} )$ ;
For Integer n = 1 to 9999 do Display{ n , $G^{o}(n)$ } end do.
```

"Without roundoff, the ideal value $G^{o}(x) \approx 1.0$ for all real x . Rounded floating-point gets 0.0 almost always for all practicable precisions. What, if anything, does Unum Computing get for $G^{\circ}(n)$? And how long does it take? It cannot be soon nor simply 1.0 ."

Kahan’s *Unum-Targeted* Variation

Real Function Gº *(*x*)* := T*(* Q*(*x*)*^{2} + (10.0^{-300})$^{10000\cdot(x+1)}$ *)* ;
For Integer n = 1 to 9999 do Display{ n , Gº *(*n*)* } end do.

“Without roundoff, the ideal value Gº (x) ≈ 1.0 for all real x . Rounded floating-point gets 0.0 almost always for all practicable precisions. What, if anything, does Unum Computing get for G°(n*)* ? And how long does it take? It cannot be soon nor simply 1.0 .”

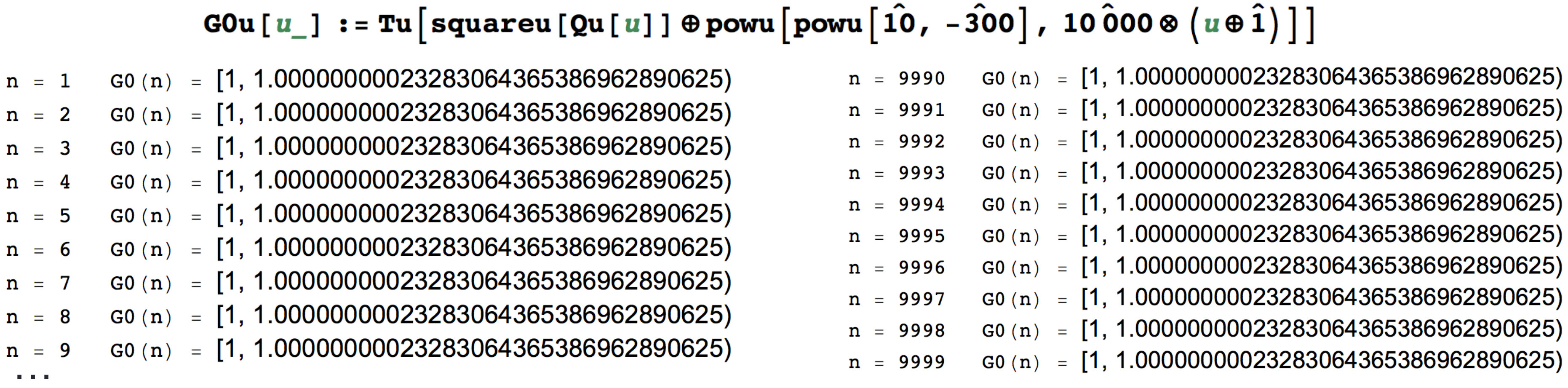
Surprise. Unums handled this without a hiccup. Quickly.

Kahan’s *Unum-Targeted* Variation

Real Function G^{o} *(x)* := T(Q*(*x*)*^{2} + $(10.0^{-300})^{10000\cdot(x+1)}$ *)* ;
For Integer n = 1 to 9999 do Display{ n , G^{o} *(*n*)* } end do.

“Without roundoff, the ideal value G^{o} (x) ≈ 1.0 for all real x . Rounded floating-point gets 0.0 almost always for all practicable precisions. What, if anything, does Unum Computing get for G^{o}*(*n*)* ? And how long does it take? It cannot be soon nor simply 1.0 .”

Surprise. Unums handled this without a hiccup. Quickly.



```
G0u[u_] := Tu[squareu[Qu[u]] ⊕ powu[powu[1̂0, -3̂00], 10 0̂00 ⊗ (u ⊕ 1̂)]]
```

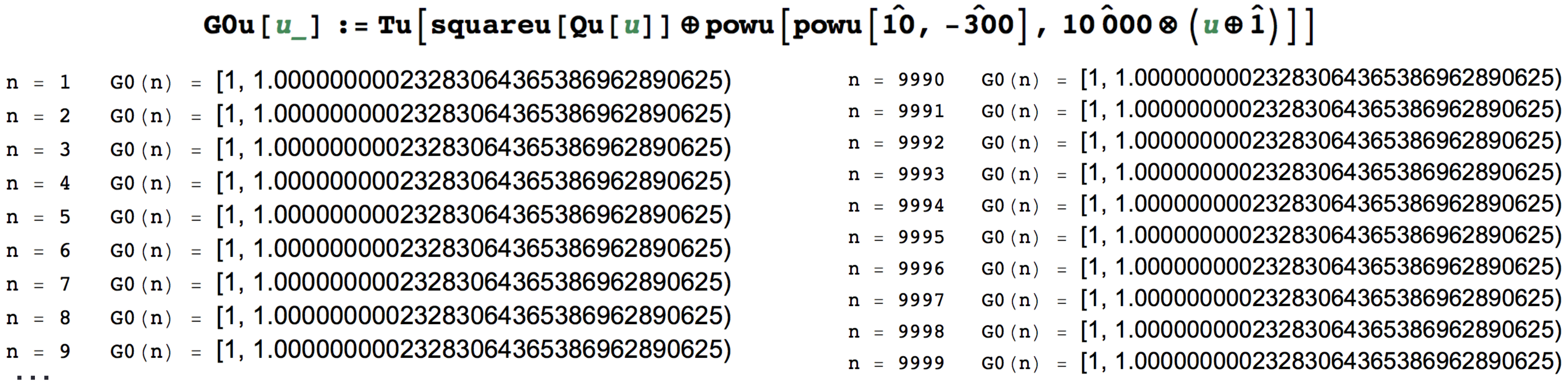
```
n = 1    G0(n) = [1, 1.00000000023283064365386962890625)
n = 2    G0(n) = [1, 1.00000000023283064365386962890625)
n = 3    G0(n) = [1, 1.00000000023283064365386962890625)
n = 4    G0(n) = [1, 1.00000000023283064365386962890625)
n = 5    G0(n) = [1, 1.00000000023283064365386962890625)
n = 6    G0(n) = [1, 1.00000000023283064365386962890625)
n = 7    G0(n) = [1, 1.00000000023283064365386962890625)
n = 8    G0(n) = [1, 1.00000000023283064365386962890625)
n = 9    G0(n) = [1, 1.00000000023283064365386962890625)
 ...
n = 9990    G0(n) = [1, 1.00000000023283064365386962890625)
n = 9991    G0(n) = [1, 1.00000000023283064365386962890625)
n = 9992    G0(n) = [1, 1.00000000023283064365386962890625)
n = 9993    G0(n) = [1, 1.00000000023283064365386962890625)
n = 9994    G0(n) = [1, 1.00000000023283064365386962890625)
n = 9995    G0(n) = [1, 1.00000000023283064365386962890625)
n = 9996    G0(n) = [1, 1.00000000023283064365386962890625)
n = 9997    G0(n) = [1, 1.00000000023283064365386962890625)
n = 9998    G0(n) = [1, 1.00000000023283064365386962890625)
n = 9999    G0(n) = [1, 1.00000000023283064365386962890625)
```

Kahan’s *Unum-Targeted* Variation

Real Function G° (x) := T(Q$(x)^2$ + $(10.0^{-300})^{10000\cdot(x+1)}$) ;
For Integer n = 1 to 9999 do Display{ n , G° (n) } end do.

“Without roundoff, the ideal value G° (x) ≈ 1.0 for all real x . Rounded floating-point gets 0.0 almost always for all practicable precisions. What, if anything, does Unum Computing get for G°(n) ? And how long does it take? It cannot be soon nor simply 1.0 .”

Surprise. Unums handled this without a hiccup. Quickly.



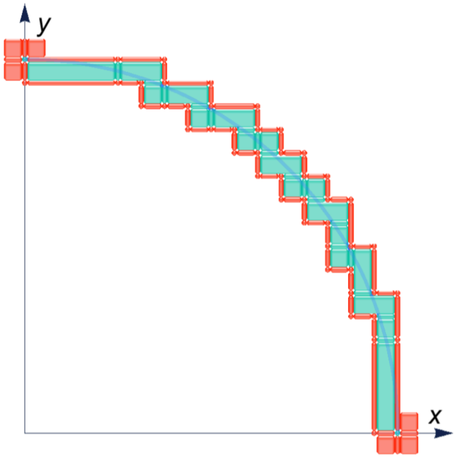
```
G0u[u_] := Tu[squareu[Qu[u]] ⊕ powu[powu[1̂0̂, -3̂0̂0̂], 1̂0̂0̂0̂0̂ ⊗ (u ⊕ 1̂)]]
```

```
n = 1    G0(n) = [1, 1.00000000023283064365386962890625)     n = 9990   G0(n) = [1, 1.00000000023283064365386962890625)
n = 2    G0(n) = [1, 1.00000000023283064365386962890625)     n = 9991   G0(n) = [1, 1.00000000023283064365386962890625)
n = 3    G0(n) = [1, 1.00000000023283064365386962890625)     n = 9992   G0(n) = [1, 1.00000000023283064365386962890625)
n = 4    G0(n) = [1, 1.00000000023283064365386962890625)     n = 9993   G0(n) = [1, 1.00000000023283064365386962890625)
n = 5    G0(n) = [1, 1.00000000023283064365386962890625)     n = 9994   G0(n) = [1, 1.00000000023283064365386962890625)
n = 6    G0(n) = [1, 1.00000000023283064365386962890625)     n = 9995   G0(n) = [1, 1.00000000023283064365386962890625)
n = 7    G0(n) = [1, 1.00000000023283064365386962890625)     n = 9996   G0(n) = [1, 1.00000000023283064365386962890625)
n = 8    G0(n) = [1, 1.00000000023283064365386962890625)     n = 9997   G0(n) = [1, 1.00000000023283064365386962890625)
n = 9    G0(n) = [1, 1.00000000023283064365386962890625)     n = 9998   G0(n) = [1, 1.00000000023283064365386962890625)
 ...                                                          n = 9999   G0(n) = [1, 1.00000000023283064365386962890625)
```

Kahan’s “infinitesimal” (his term) becomes unum (0, ε).

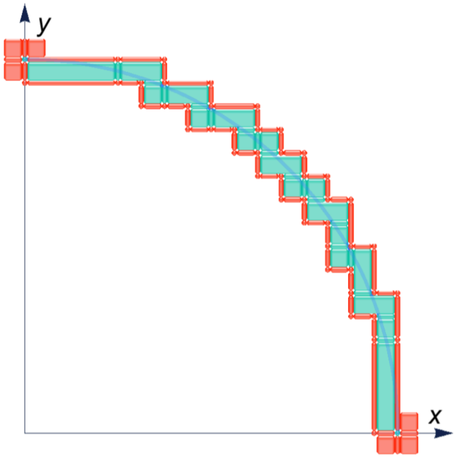
An Inconvenient Infinity

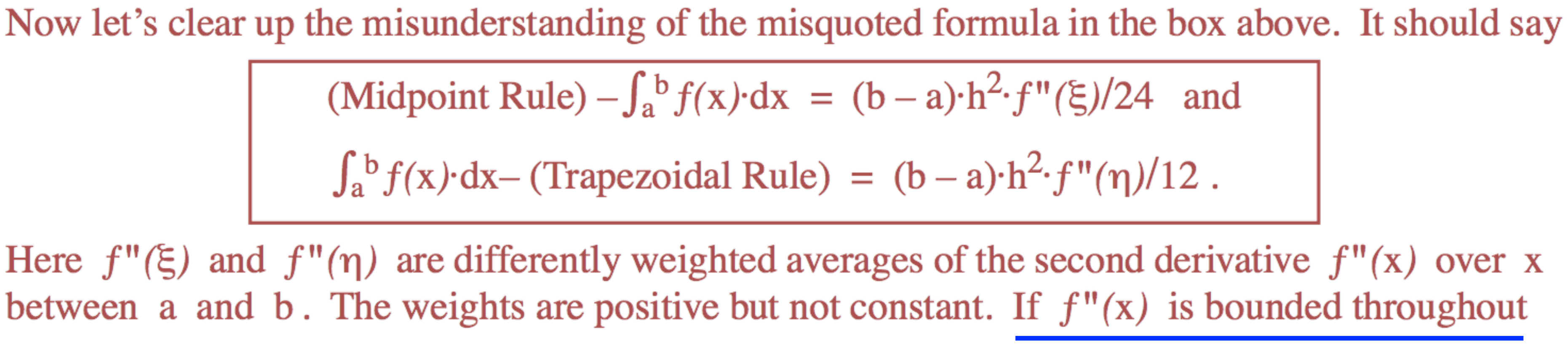
My example of quarter-circle integration takes $O(n)$ time for n subdivisions, and produces $O(1/n)$ size rigorous bounds. Works on any continuous function.



An Inconvenient Infinity

My example of quarter-circle integration takes $O(n)$ time for n subdivisions, and produces $O(1/n)$ size rigorous bounds. Works on any continuous function.





Now let’s clear up the misunderstanding of the misquoted formula in the box above. It should say

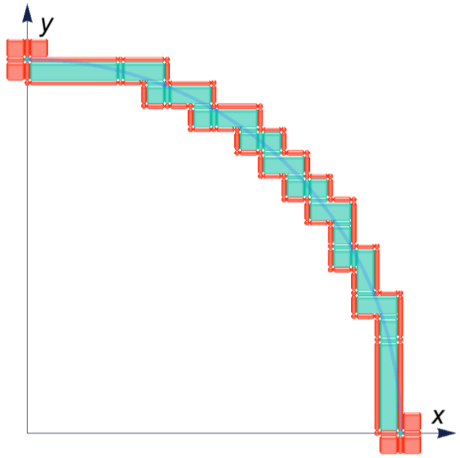
$$\text{(Midpoint Rule)} - \int_a^b f(x)\cdot dx \;=\; (b-a)\cdot h^2\cdot f''(\xi)/24 \quad \text{and}$$

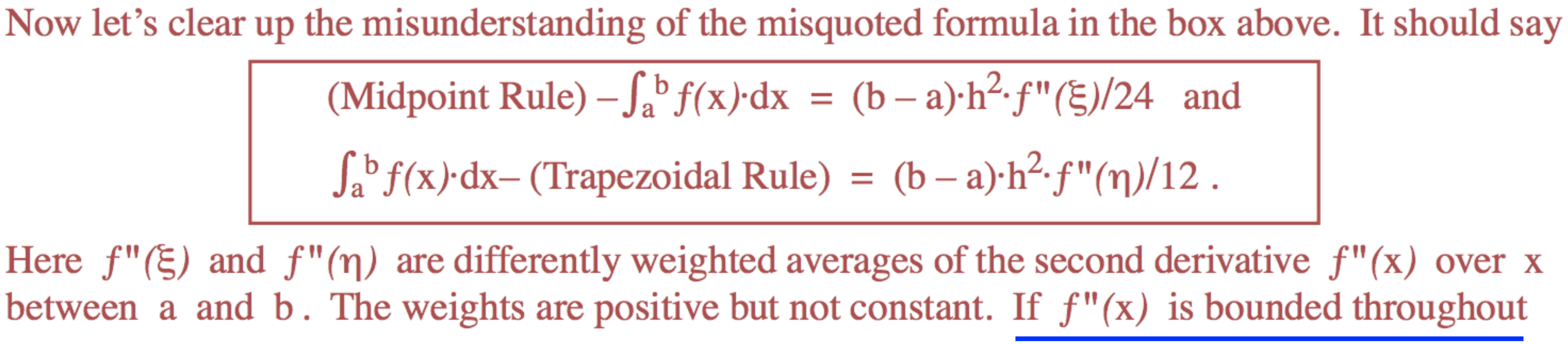
$$\int_a^b f(x)\cdot dx - \text{(Trapezoidal Rule)} \;=\; (b-a)\cdot h^2\cdot f''(\eta)/12\ .$$

Here $f''(\xi)$ and $f''(\eta)$ are differently weighted averages of the second derivative $f''(x)$ over x between a and b. The weights are positive but not constant. If $f''(x)$ is bounded throughout

An Inconvenient Infinity

My example of quarter-circle integration takes $O(n)$ time for n subdivisions, and produces $O(1/n)$ size rigorous bounds. Works on any continuous function.





Now let's clear up the misunderstanding of the misquoted formula in the box above. It should say

$$\boxed{\begin{array}{c}\text{(Midpoint Rule)} - \int_a^b f(x)\cdot dx \;=\; (b-a)\cdot h^2\cdot f''(\xi)/24 \quad \text{and} \\ \int_a^b f(x)\cdot dx - \text{(Trapezoidal Rule)} \;=\; (b-a)\cdot h^2\cdot f''(\eta)/12\ .\end{array}}$$

Here $f''(\xi)$ and $f''(\eta)$ are differently weighted averages of the second derivative $f''(x)$ over x between a and b. The weights are positive but not constant. If $f''(x)$ is bounded throughout

But $f''(x)$ is *not* bounded throughout. ***Kahan uses the formula anyway!***

Also, Kahan says my method is $O(n^2)$.
Willful misunderstanding. Obviously not true (see figure above).

Too many mistakes to cover here…

The book claims it ends all error.

Too many mistakes to cover here…

The book claims it ends all error.

It does not. A *specific kind* of error.

Too many mistakes to cover here…

The book claims it ends all error.

It does not. A *specific kind* of error.

Unums are tarted intervals.

Too many mistakes to cover here…

The book claims it ends all error.

It does not. A *specific kind* of error.

Unums are tarted intervals.

Unums *subsume* floats and intervals.
This is an environment, not just a format.

Too many mistakes to cover here…

The book claims it ends all error.

It does not. A *specific kind* of error.

Unums are tarted intervals.

Unums *subsume* floats and intervals.
This is an environment, not just a format.

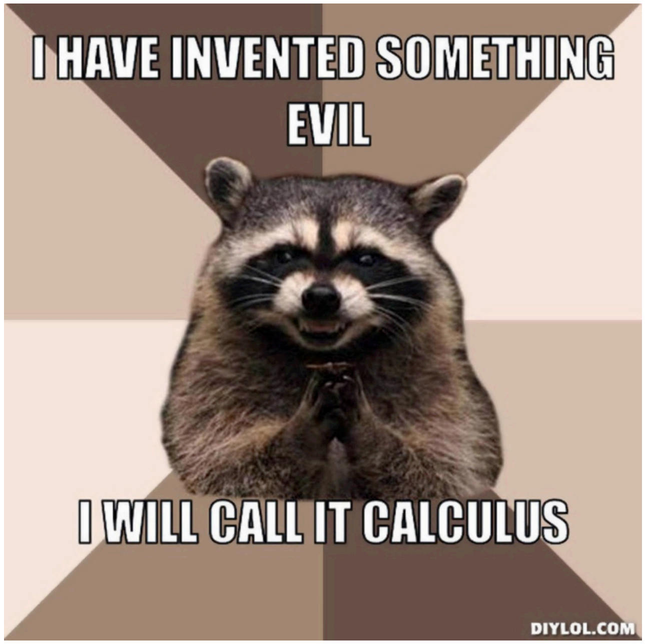
Gustafson regards calculus as “evil.”
He is not joking.

Too many mistakes to cover here…

The book claims it ends all error.	It does not. A *specific kind* of error.
Unums are tarted intervals.	Unums *subsume* floats and intervals. This is an environment, not just a format.
Gustafson regards calculus as "evil." He is not joking.	Good grief. A raccoon meme from DIY LOL, and he thinks I'm *not* joking?

Too many mistakes to cover here…

The book claims it ends all error.	It does not. A *specific kind* of error.
Unums are tarted intervals.	Unums *subsume* floats and intervals. This is an environment, not just a format.
Gustafson regards calculus as “evil.” He is not joking.	Good grief. A raccoon meme from DIY LOL, and he thinks I’m *not* joking?



Too many mistakes to cover here…

The book claims it ends all error.	It does not. A *specific kind* of error.
Unums are tarted intervals.	Unums *subsume* floats and intervals. This is an environment, not just a format.
Gustafson regards calculus as "evil." He is not joking.	Good grief. A raccoon meme from DIY LOL, and he thinks I'm *not* joking?
That's not "grade school" math!	

Too many mistakes to cover here…

The book claims it ends all error.	It does not. A *specific kind* of error.
Unums are tarted intervals.	Unums *subsume* floats and intervals. This is an environment, not just a format.
Gustafson regards calculus as “evil.” He is not joking.	Good grief. A raccoon meme from DIY LOL, and he thinks I’m *not* joking?
That’s not “grade school” math!	12th grade is a grade. So is 11th grade.

Too many mistakes to cover here…

The book claims it ends all error.	It does not. A *specific kind* of error.
Unums are tarted intervals.	Unums *subsume* floats and intervals. This is an environment, not just a format.
Gustafson regards calculus as "evil." He is not joking.	Good grief. A raccoon meme from DIY LOL, and he thinks I'm *not* joking?
That's not "grade school" math!	12th grade is a grade. So is 11th grade.
Unums will cost *thousands* of extra transistors!	

Too many mistakes to cover here…

The book claims it ends all error.	It does not. A *specific kind* of error.
Unums are tarted intervals.	Unums *subsume* floats and intervals. This is an environment, not just a format.
Gustafson regards calculus as "evil." He is not joking.	Good grief. A raccoon meme from DIY LOL, and he thinks I'm *not* joking?
That's not "grade school" math!	12th grade is a grade. So is 11th grade.
Unums will cost *thousands* of extra transistors!	Which will cost *thousandths* of a penny. The year is 2016, not 1985.

Too many mistakes to cover here…

The book claims it ends all error.	It does not. A *specific kind* of error.
Unums are tarted intervals.	Unums *subsume* floats and intervals. This is an environment, not just a format.
Gustafson regards calculus as "evil." He is not joking.	Good grief. A raccoon meme from DIY LOL, and he thinks I'm *not* joking?
That's not "grade school" math!	12th grade is a grade. So is 11th grade.
Unums will cost *thousands* of extra transistors!	Which will cost *thousandths* of a penny. The year is 2016, not 1985.
His approach is very inefficient; here's a faster one that usually works.	

Too many mistakes to cover here…

The book claims it ends all error.	It does not. A *specific kind* of error.
Unums are tarted intervals.	Unums *subsume* floats and intervals. This is an environment, not just a format.
Gustafson regards calculus as "evil." He is not joking.	Good grief. A raccoon meme from DIY LOL, and he thinks I'm *not* joking?
That's not "grade school" math!	12th grade is a grade. So is 11th grade.
Unums will cost *thousands* of extra transistors!	Which will cost *thousandths* of a penny. The year is 2016, not 1985.
His approach is very inefficient; here's a faster one that usually works.	I'm not interested in methods that *usually* work. We have plenty of those.

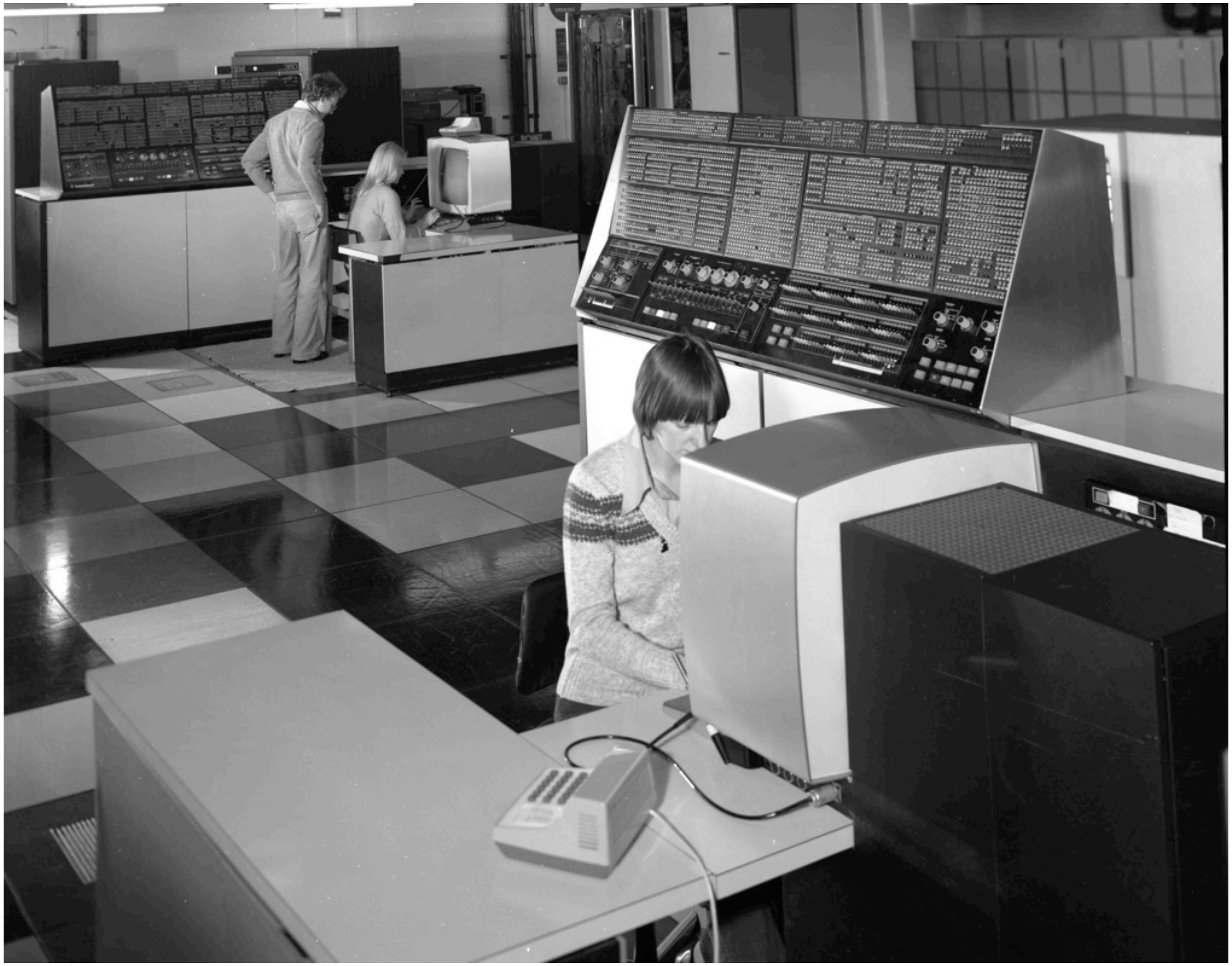
Too many mistakes to cover here…

The book claims it ends all error.	It does not. A *specific kind* of error.
Unums are tarted intervals.	Unums *subsume* floats and intervals. This is an environment, not just a format.
Gustafson regards calculus as “evil.” He is not joking.	Good grief. A raccoon meme from DIY LOL, and he thinks I’m *not* joking?
That’s not “grade school” math!	12th grade is a grade. So is 11th grade.
Unums will cost *thousands* of extra transistors!	Which will cost *thousandths* of a penny. The year is 2016, not 1985.
His approach is very inefficient; here’s a faster one that usually works.	I’m not interested in methods that *usually* work. We have plenty of those.
Gustafson suffers from a misconception about floating point shared by Von Neumann.	

Too many mistakes to cover here…

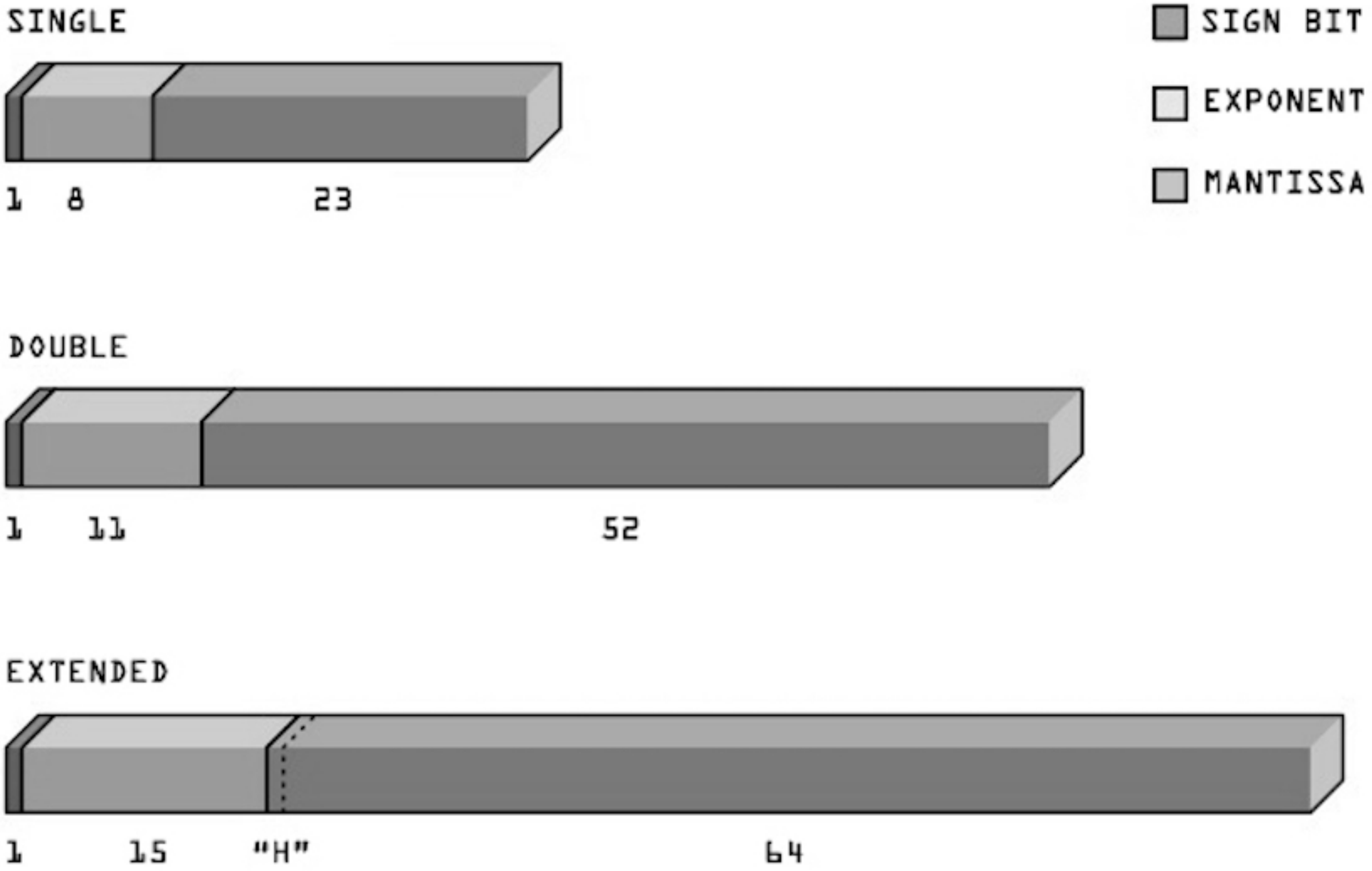
The book claims it ends all error.	It does not. A *specific kind* of error.
Unums are tarted intervals.	Unums *subsume* floats and intervals. This is an environment, not just a format.
Gustafson regards calculus as “evil.” He is not joking.	Good grief. A raccoon meme from DIY LOL, and he thinks I’m *not* joking?
That’s not “grade school” math!	12th grade is a grade. So is 11th grade.
Unums will cost *thousands* of extra transistors!	Which will cost *thousandths* of a penny. The year is 2016, not 1985.
His approach is very inefficient; here’s a faster one that usually works.	I’m not interested in methods that *usually* work. We have plenty of those.
Gustafson suffers from a misconception about floating point shared by Von Neumann.	It pleases me very much to share misconceptions with John von Neumann.

COMPUTERS THEN

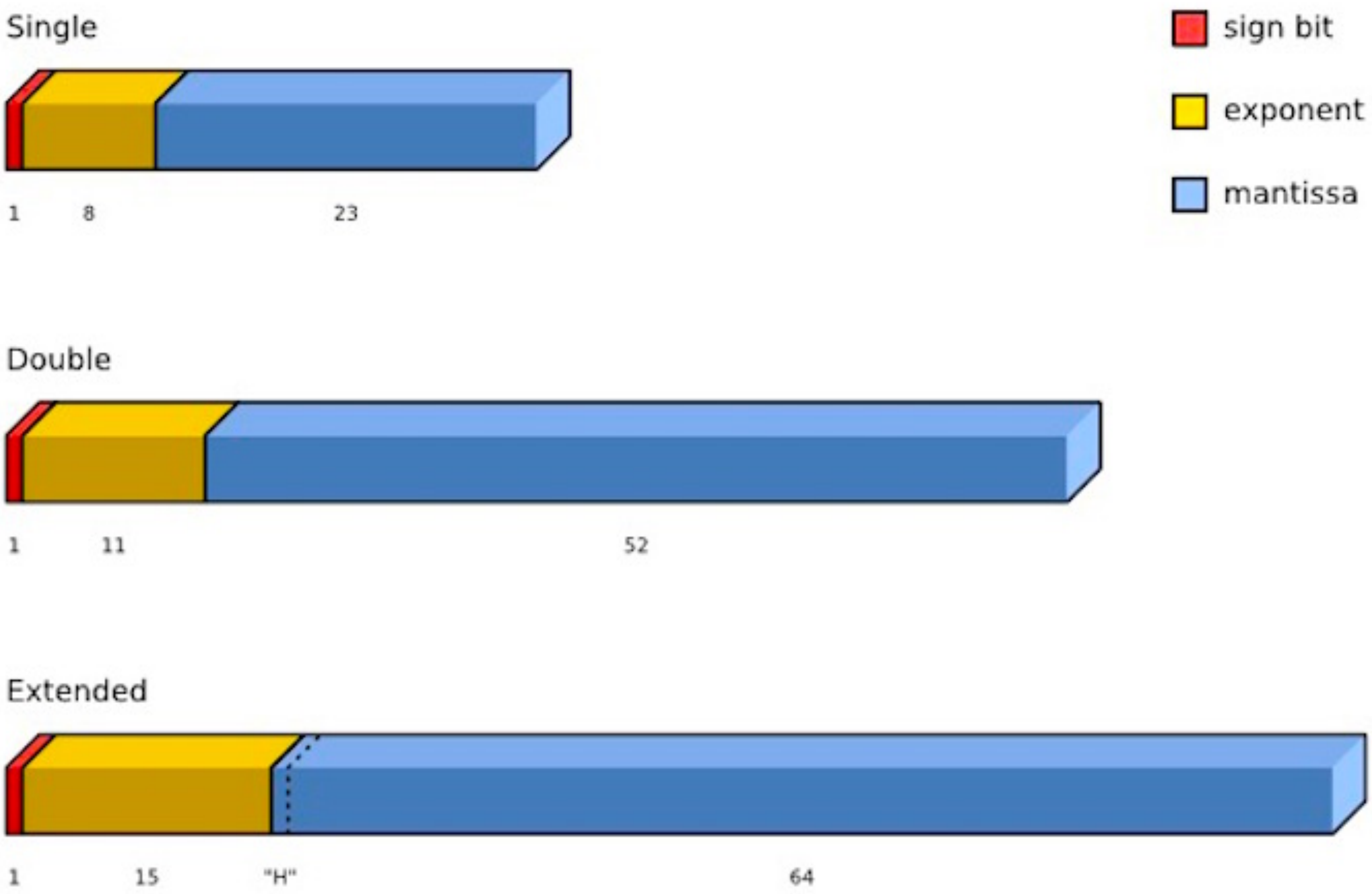


COMPUTERS NOW

ARITHMETIC THEN



ARITHMETIC NOW



Kahan’s biggest blind spot of all

Remember: There is nothing floats can do that unums cannot. ■

The last line of my book, p. 413, and emphasized throughout

Kahan’s biggest blind spot of all

Remember: There is nothing floats can do that unums cannot. ■

The last line of my book, p. 413, and emphasized throughout

- Unums are a *superset* of IEEE floats. Not an “alternative.”

Kahan’s biggest blind spot of all

Remember: There is nothing floats can do that unums cannot. ■

The last line of my book, p. 413, and emphasized throughout

- Unums are a *superset* of IEEE floats. Not an “alternative.”
- We need not throw away float algorithms that work well

Kahan’s biggest blind spot of all

> Remember: There is nothing floats can do that unums cannot. ■

The last line of my book, p. 413, and emphasized throughout

- Unums are a *superset* of IEEE floats. Not an “alternative.”
- We need not throw away float algorithms that work well.
- Rounding can be *requested*, not forced on users. **Unums end the error of mandatory, invisible substitution of incorrect exact values for correct answers.**

Kahan’s biggest blind spot of all

Remember: There is nothing floats can do that unums cannot. ■

The last line of my book, p. 413, and emphasized throughout

- Unums are a *superset* of IEEE floats. Not an “alternative.”
- We need not throw away float algorithms that work well.
- Rounding can be *requested*, not forced on users. **Unums end the error of mandatory, invisible substitution of incorrect exact values for correct answers.**
- Float methods are a good way to deal with “The Curse of High Dimensions” in many cases, like getting a starting answer for *Ax* = *b* linear systems in polynomial time.

WK's Dysphemisms, Insults, and Rants about *The End of Error: Unum Computing*

Bunkum!

Lies

liar

Bogus

perverse

crude

WK’s Dysphemisms, Insults, and Rants about *The End of Error: Unum Computing*

foolish

Bunkum!

Lies

Flogging

misunderstandings

liar

tarted

Bogus

perverse

crude

incorrigibly unrealistic

WK's Dysphemisms, Insults, and Rants about *The End of Error: Unum Computing*

foolish Puffery Bunkum!

Lies Flogging faux

misunderstandings

seductive ***liar***

tarted Bogus

folly perverse exaggerated

crude

incorrigibly unrealistic

WK's Dysphemisms, Insults, and Rants about *The End of Error: Unum Computing*

foolish

Puffery

Bunkum!

snide

Mere hyperbole

Lies

Flogging

faux

unfair

misunderstandings

seductive

liar

misconceptions

tarted

Bogus

folly

silly

exaggerated

perverse

crude

misguided

incorrigibly unrealistic

WK’s Dysphemisms, Insults, and Rants about *The End of Error: Unum Computing*

foolish Puffery Bunkum! *snide*

Mere hyperbole

Lies Flogging faux

unfair

misunderstandings

seductive ***liar***

misconceptions

tarted Bogus

folly perverse silly exaggerated

crude

misguided incorrigibly unrealistic

Invective worked for Donald Trump, but… is this really the right way to discuss *mathematics*?

“THE LORD OF THE REALS…
DOES NOT SHARE POWER.”