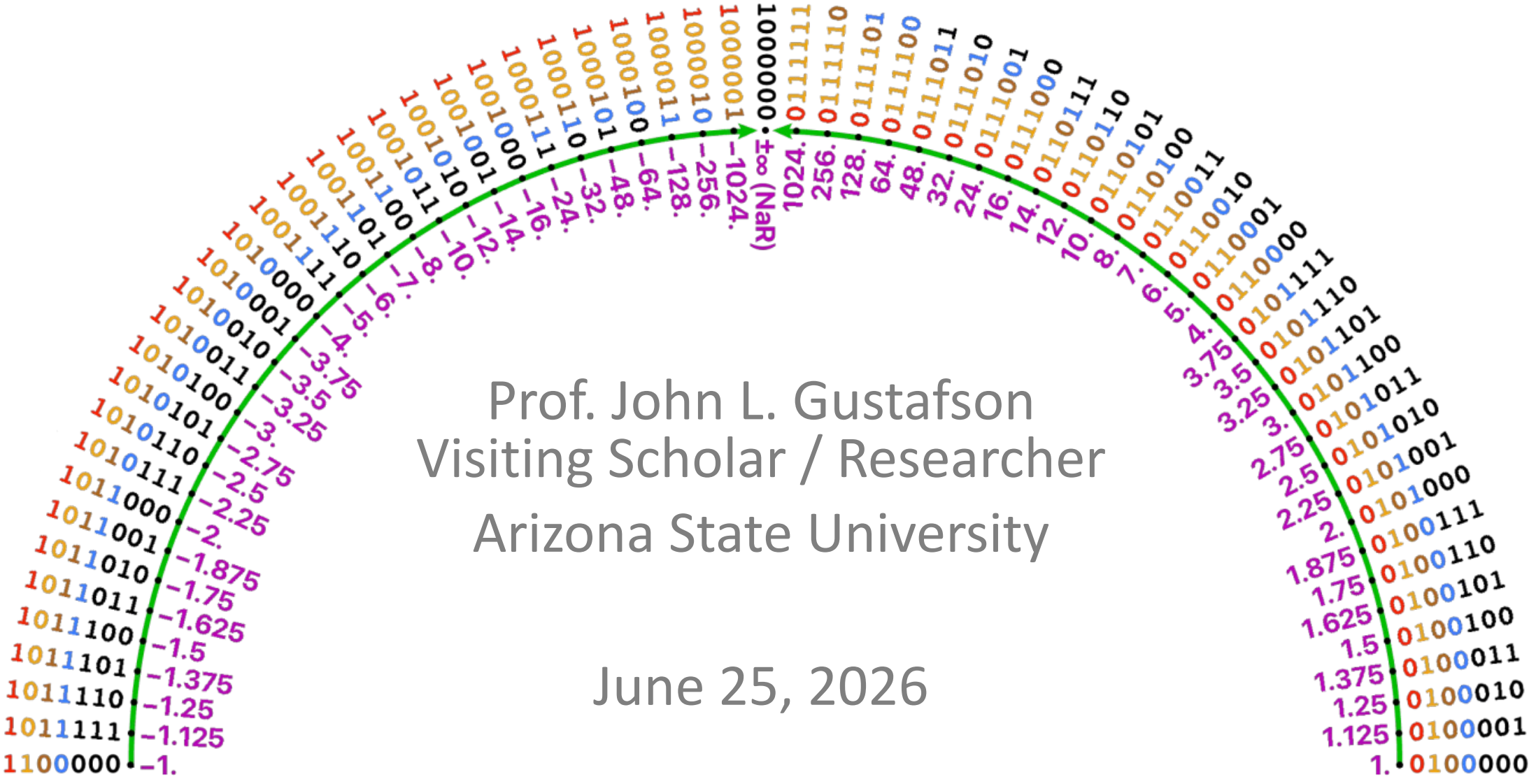
Every Bit Counts: Posit Computing for Energy-Efficient HPC and AI



Prof. John L. Gustafson
Visiting Scholar / Researcher
Arizona State University

June 25, 2026

Thesis

The slowdown of Moore's law and the rise of AI (Machine Learning) have made us realize that IEEE floating-point format (Intel, 1977) is long overdue for replacement.

IEEE 754 floats waste storage, bandwidth, energy, and power.

The *posit* format (2017) is a new way to compute with real numbers, designed mathematically to meet engineering goals. Gains of 4× possible by using fewer bits, but getting better answers.

This is a watershed, a revolution. And it is well underway.

Energy cost of a 64-bit IEEE float multiplication, in 2026:

Task	Energy, nanojoules
All data in registers	0.002 to 0.02
All data in L1 or L2 cache	0.1 to 0.4
All data in L3 cache	0.2 to 1.
All data in HBM3 or HBM4 DRAM (like GPU stacks)	0.6 to 1.5
All data in DDR5 DRAM (like server memory)	3 to 5.
All data in a different cabinet in a cluster	~20.

The multiplication operation itself	**0.005**

Data motion is 10× to 100× the energy cost of *arithmetic*.

Obvious conclusion: **Use fewer bits to get the work done.**

The Game of “Twenty Questions”

Parlor game from the 1800s, later made into a radio show in the US.

Goal: Identify a mystery object with no more than 20 yes-no questions.

Notice that yes-no answers are *bits*.

Five Questions:

“I’m thinking of an integer between 1 and 30.”

“Is it sixteen or larger?”	**No.**	0••••
“Is it eight or larger?”	**Yes.**	01•••
“Is it twelve or larger?”	**No.**	010••
“Is it ten or larger?”	**Yes.**	0101•
“Is it eleven?”	**Yes, you win.**	01011

Eleven in binary

This is how Analog-to-Digital convertors work.

But what if the game starts with "I'm thinking of a real number."

What should be your strategy?

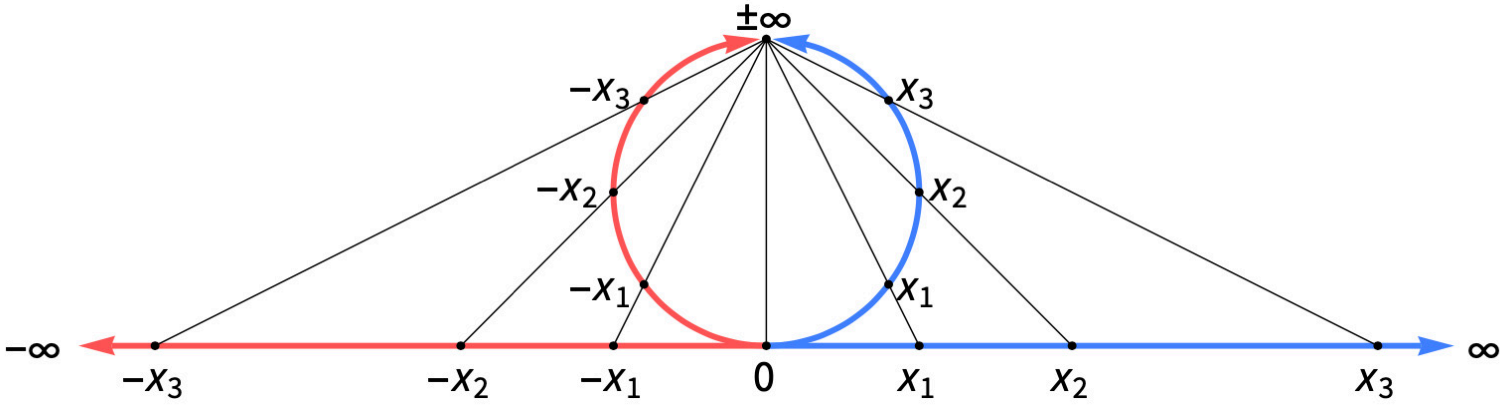
I often hear this line of reasoning:

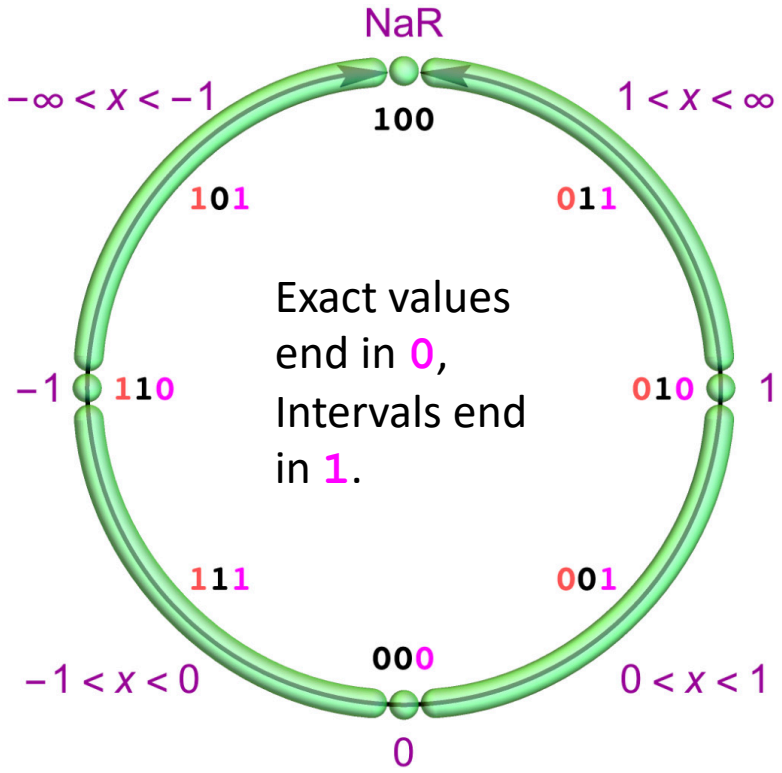
> "You cannot represent all the real numbers on a computer because there are an infinite number of them, and computers only have a finite amount of memory."

This is a fallacy!

On the next slide, I will do it in three bits.

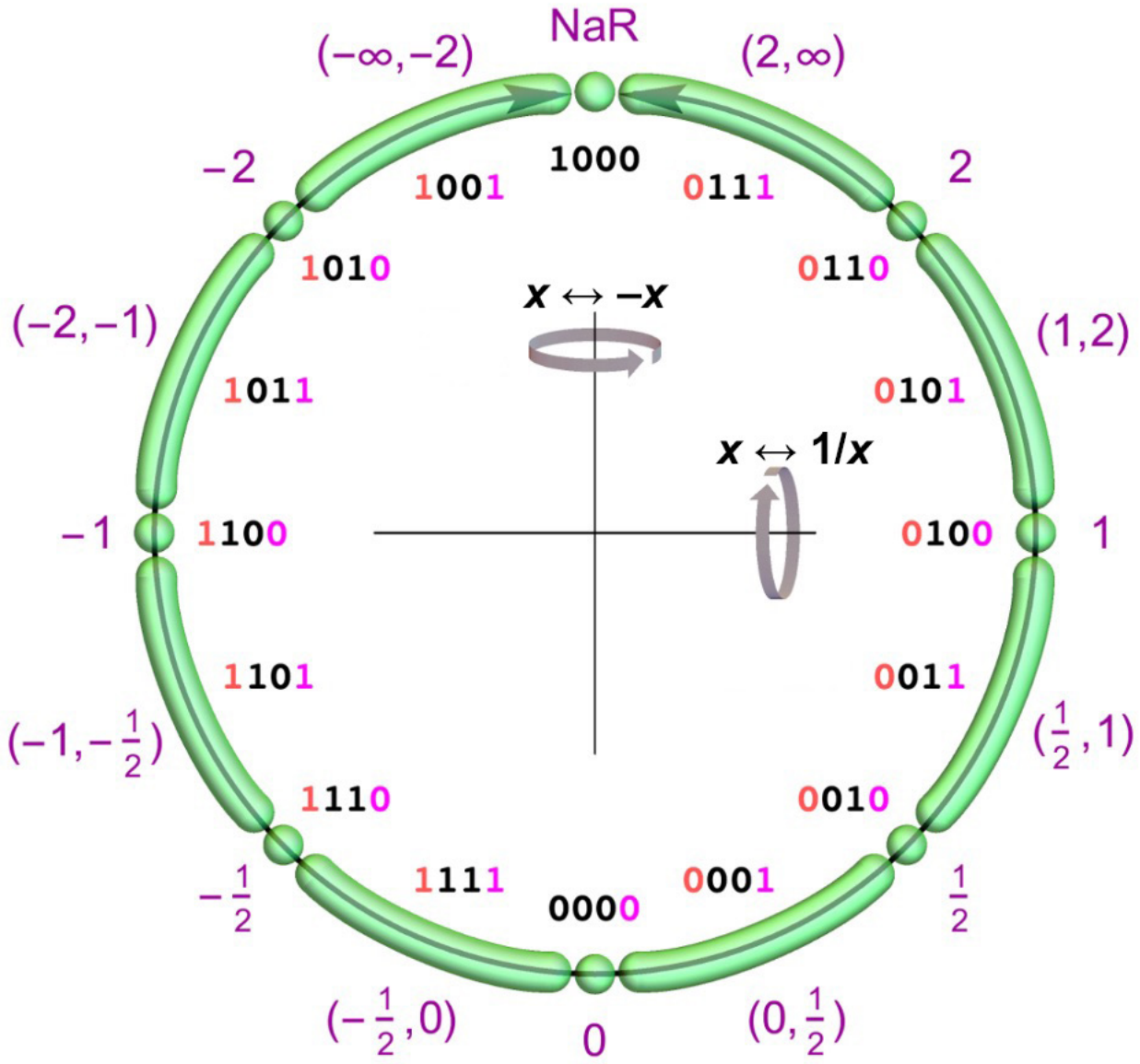
The projective reals, and 3-bit *universal numbers*





000	0
001	Real x such that 0 < x < 1
010	1
011	Real *x* such that 1 < *x* < ∞
100	Anything that is Not a Real (NaR)
101	Real *x* such that −∞ < *x* < −1
110	−1
111	Real *x* such that −1 < *x* < 0

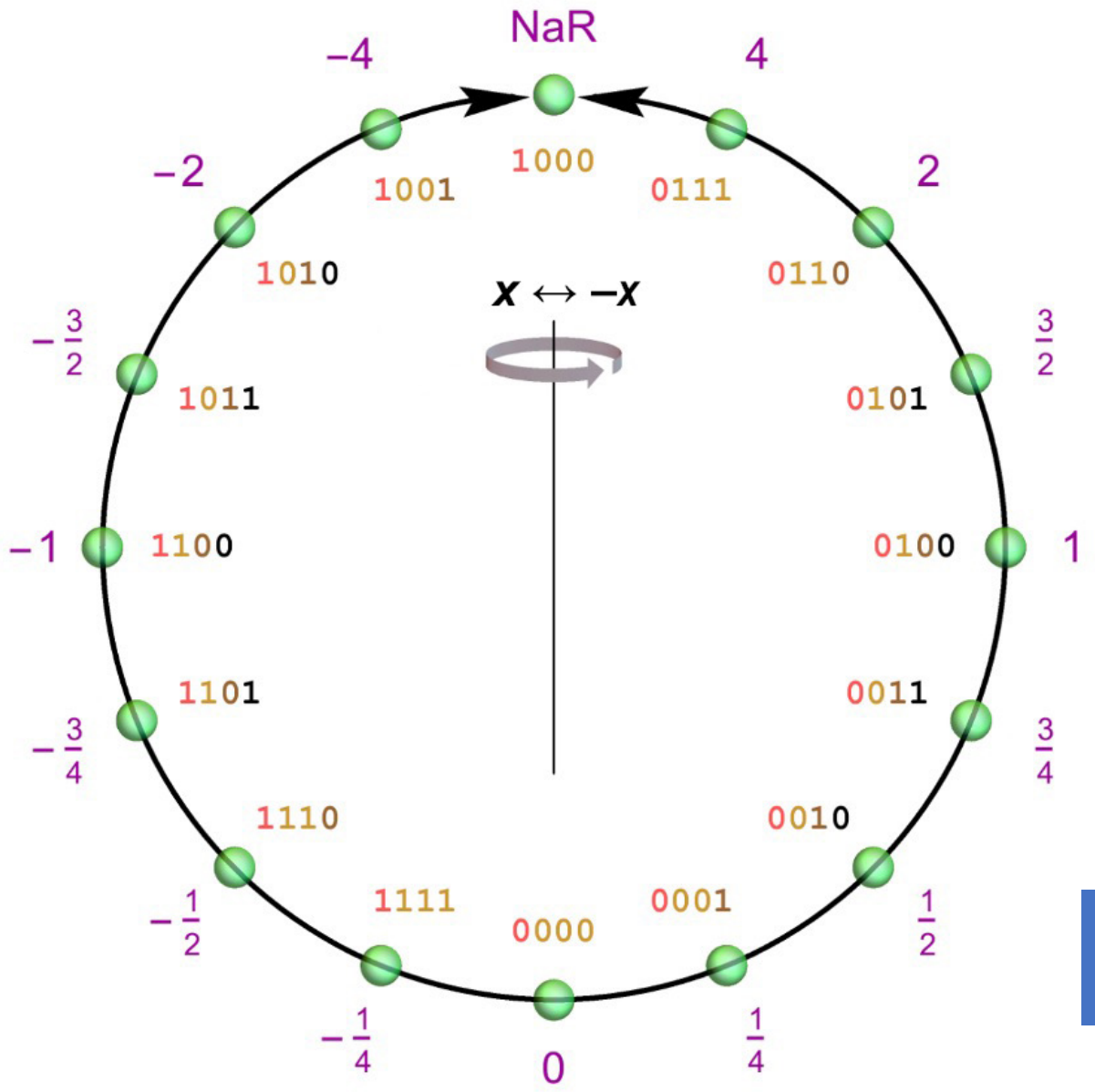
Append a bit for more precision.



We did not have to use 2 at the northwest point. More about that later.

Reciprocation is easy and exact, like negating (2's complement integer)

That's the *interval* form. The rounded form is a *posit*.



Now reciprocals are only perfect for powers of 2.

Appending a **0** bit does not change the represented value.

Appending a **1** bit:

- Between NaR and *x*, insert 2*x*.
- Between 0 and *x*, insert *x*/2.
- Otherwise, between *x* and *y*, insert (*x* + *y*) / 2.

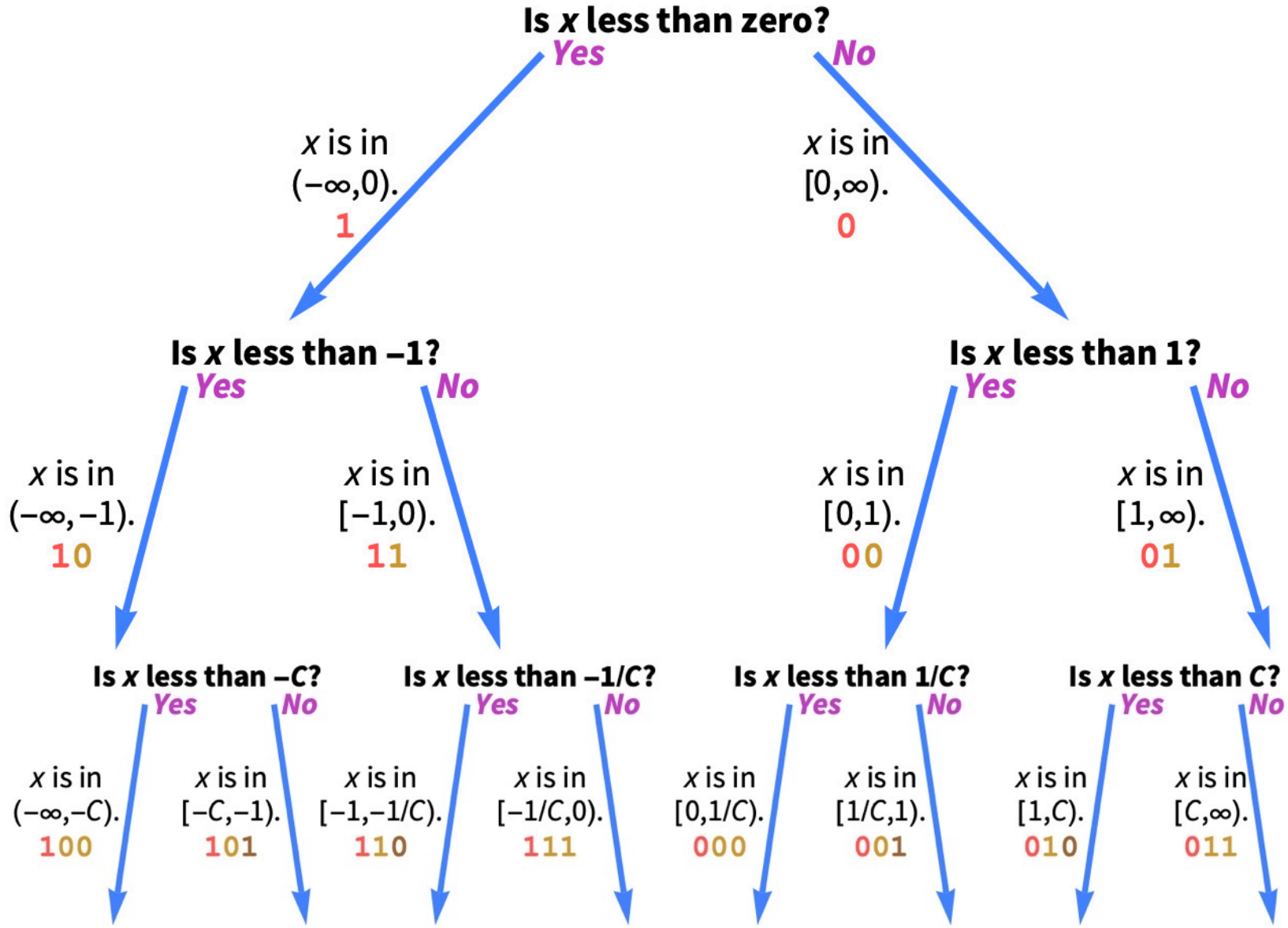
These are 4-bit posits.

posit | ˈpäzət |

noun *Philosophy*

a statement that is made on the assumption that it will prove to be true.

Posits work like “Twenty Questions” for Real Values



Regime format lets us decode the bit string.

Like a decimal, assume infinitely many zeros after the last expressed digit. Just like 3.75 is the same as 3.7500000⋯ where “3.75” is explicit digits and the zeros are *ghost digits*.

Bit string|Ghost bits	Run Length, k	Integer meaning r: $-k$ if first bit is 0, else $k-1$
000|00⋯	∞	−∞
001|00⋯	2	−2
01•|00⋯	1	−1
10•|00⋯	1	0
110|00⋯	2	1
111|00⋯	3	2

Like tally marks, but signed!

Now we can decode posit format in a one-liner.

$$x = ((1 - 3\,S) + F) \times 2^{scale}$$

where

S is the value of the sign bit,

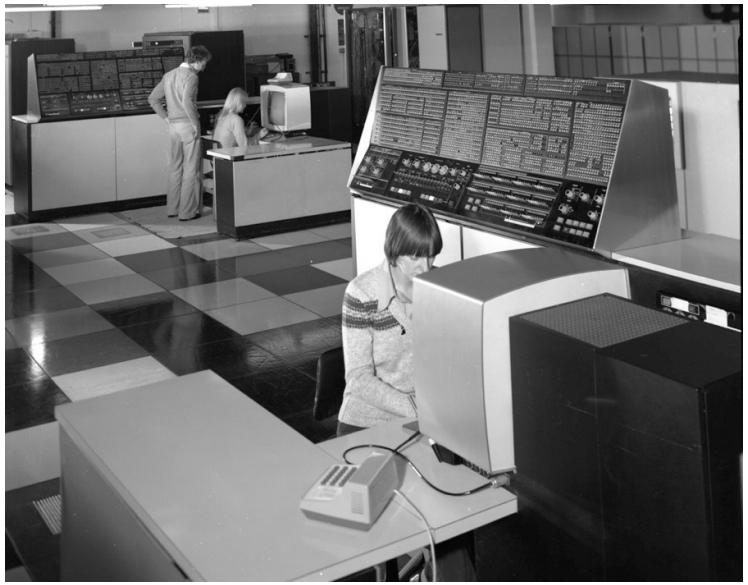
F is the value of the fraction bits, and

scale is $(-1)^S \times (r + S)$.

Tapered accuracy results. Extreme magnitudes have fewer fraction bits. No exception cases!

IEEE Floats are *Weapons of Math Destruction*

- No guarantee of **repeatable** or **portable** behavior
- Computations change when parallelized
- Low 32-bit accuracy forces use of 64-bit floats
- “Negative zero” and “positive zero” nonsense
- Illogical: $X \neq X$, and $a = b$ does not mean $f(a) = f(b)$
- Poor handling of overflow, underflow, errors
- Dynamic ranges ill-matched to application needs



IEEE floats were designed, *ad hoc*, for 1970s engineering limitations

The Dirty Secret in "Hardware" for IEEE Standard

Test the exponent bits for all 0 bits or all 1 bits…

6% of bit patterns

Trap to **software or microcode**, ~200 clocks!

$$\text{Value} = \begin{cases} \text{if } E = 31, \begin{cases} \text{if } F = 0, (-1)^S \times \infty \\ \text{else}, \begin{cases} \text{if } F \geq 0.5, \text{Quiet NaN} \\ \text{else, Signaling NaN} \end{cases} \end{cases} \\ \text{else if } E = 0, \begin{cases} \text{if } F \neq 0, \text{Subnormal}, (-1)^S \times 2^{E-14} \times (0 + F) \\ \text{else}, \begin{cases} \text{if } S = 0, \text{“Positive zero”} \\ \text{else, “Negative zero”} \end{cases} \end{cases} \\ \text{Else, a “normal” float. } (-1)^S \times 2^{E-15} \times (1 + F) \end{cases}$$

Why linear algebra isn't portable with floats

$$\begin{bmatrix} \times & \times & \times & \times \\ 2^{26} & 2 & 1 & -2^{27} \\ \times & \times & \times & \times \\ \times & \times & \times & \times \end{bmatrix} \cdot \begin{bmatrix} \times & 2^{28} & \times & \times \\ \times & 1 & \times & \times \\ \times & -1 & \times & \times \\ \times & 2^{27} & \times & \times \end{bmatrix} = \begin{bmatrix} \times & \times & \times & \times \\ \times & c_{22} & \times & \times \\ \times & \times & \times & \times \\ \times & \times & \times & \times \end{bmatrix}$$

- All values are exact in IEEE single precision.
- Dot product requires summing 2^{54}, 2, –1, and -2^{54}.
- Use IEEE double precision to be "safe."
- Parallelization changes the order of summation.
- There are 15 different ways to sum up four numbers.
  What they never taught you in Computer Science…

Possible Rounded Values with 64-bit Floats

$$(2^{54}+2)+(-1-2^{54}) \stackrel{?}{=} 0$$
$$(2^{54}-1)+(2-2^{54}) \stackrel{?}{=} 2$$
$$(2^{54}-2^{54})+(2-1) \stackrel{?}{=} 1 \text{ (correct)}$$
$$((2^{54}+2)-2^{54})-1 \stackrel{?}{=} -1$$

A parallelized version of the dot product can get *four* different values depending on summation order.

Fix the IEEE 754 Standard?

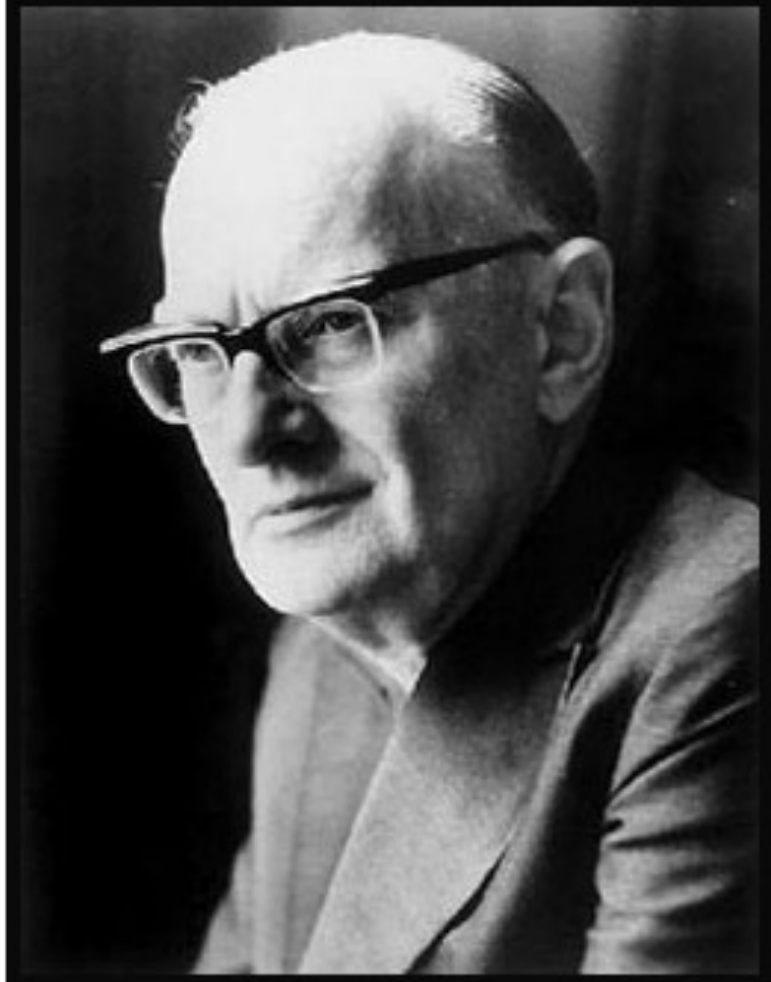
Can we get help from “the father of IEEE 754,” Bill Kahan? Probably not.

“Linguistically legislated exact reproducibility is unenforceable.”

“Faster matrix multiplication is usually too valuable to forego for unneeded exact reproducibility.”*

*How Java’s Floating-Point Hurts Everyone Everywhere”

The solution: Form a *new*, *alternative* standard.



If an elderly but distinguished scientist says that something is possible, he is almost certainly right; but if he says that it is impossible, he is very probably wrong.

(Arthur C. Clarke)

izquotes.com

11 Design Principles for NGA

1. Distribution of representable values closely resembles the distribution needed for exact calculations.
2. No redundant bit patterns. Every bit counts.
3. No hidden “modes” of operation. Source program and data suffice to determine every bit of the output.
4. All math operations must be correctly rounded.
5. It should be easy to change precisions, like it is for integers.
6. Error results must always propagate through to the final output.

11 Design Principles for NGA (cont.)

7. It should be simple to build in hardware, and fast.

In the "nice to have" category:

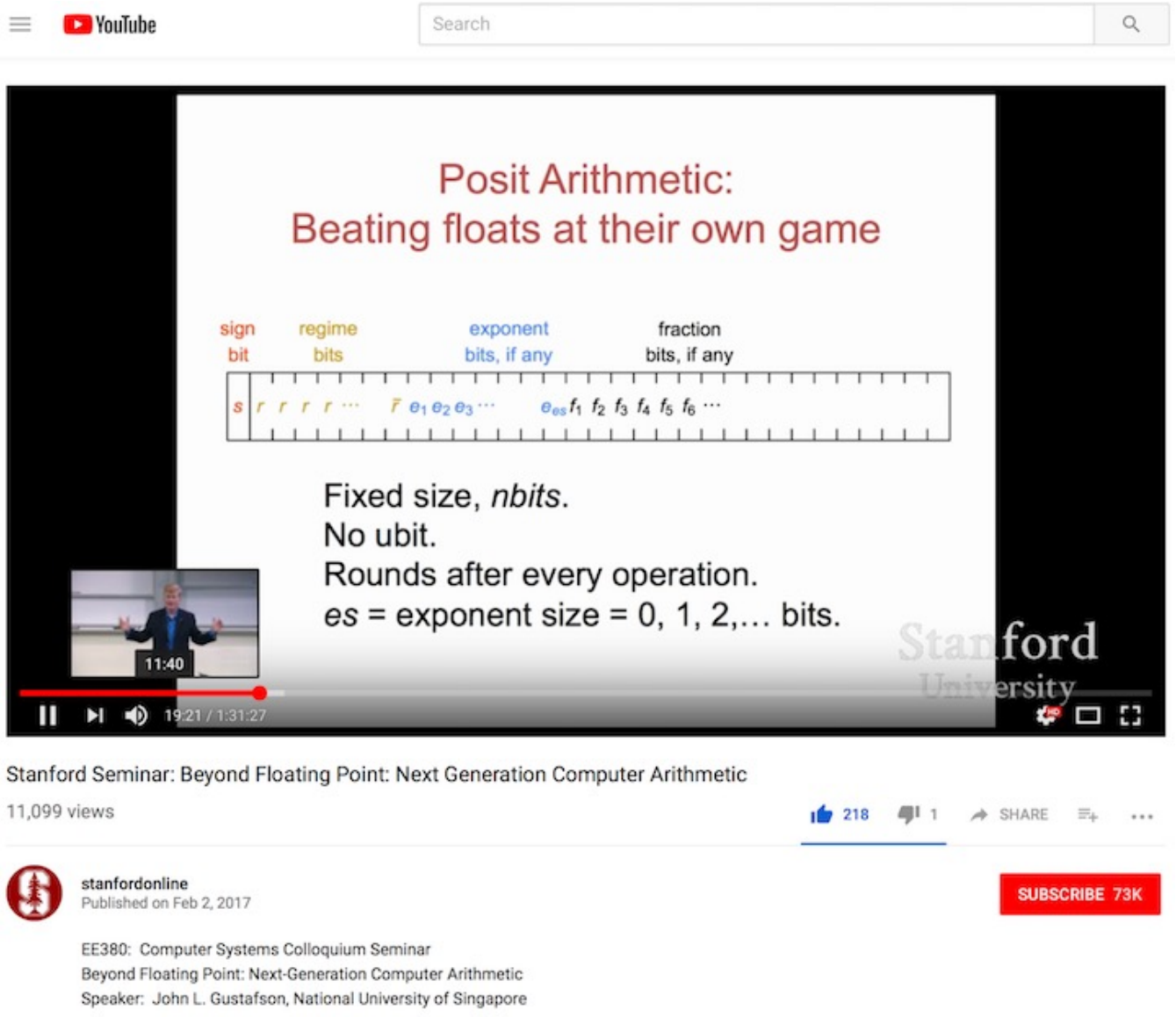
8. It should be easy to scale to any number of bits.
9. Real numbers should be ordered like integers with the same bit strings. (No extra hardware for comparison operations).
10. Negating a real should be the same as negating it as an integer.
11. The results of application computations should be close to what they would be if infinite precision is used.

Note: IEEE Std 754 floats fail ***every one of these goals***.

Posits were introduced February 2, 2017.

Stanford's EE380 Seminars are put on YouTube and usually get ~300 views.

This talk has 31,296 views so far, and rising.
Near the end, Dr. Isaac Yonemoto did a live demo of successful Machine Learning using **8-bit** posits.

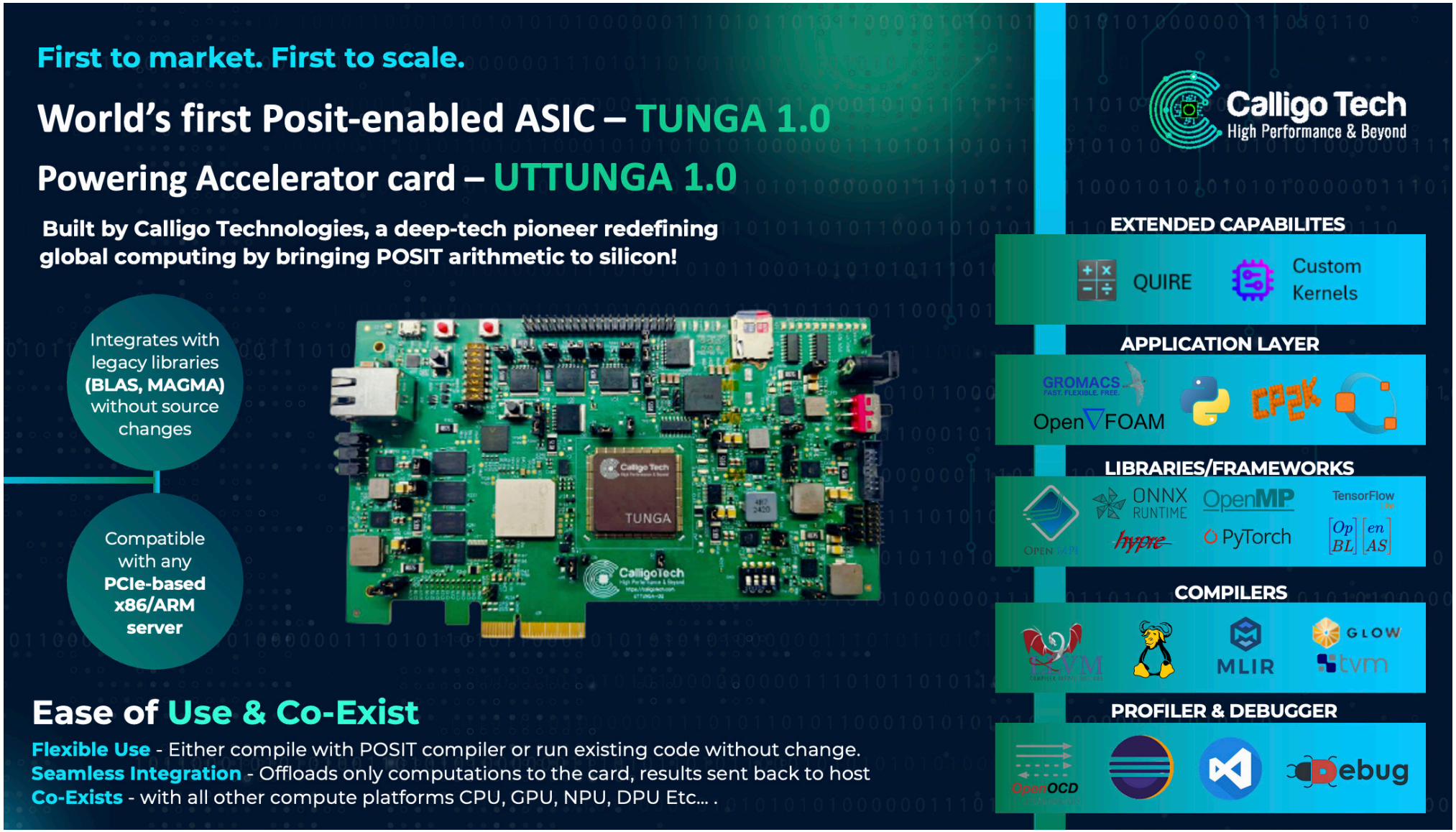


The revolution is underway.

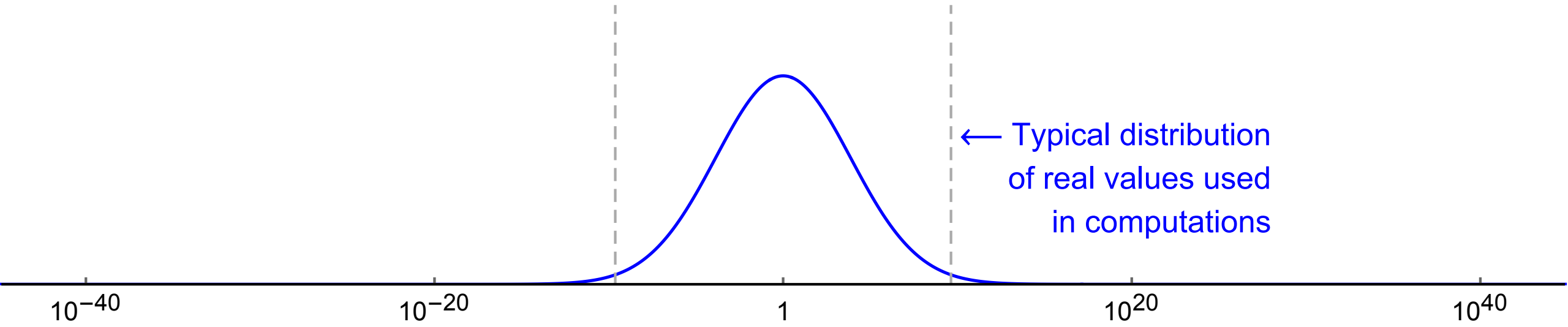
Interest from industry

Interest from universities, labs

Calligo Tech has pioneered posits hardware

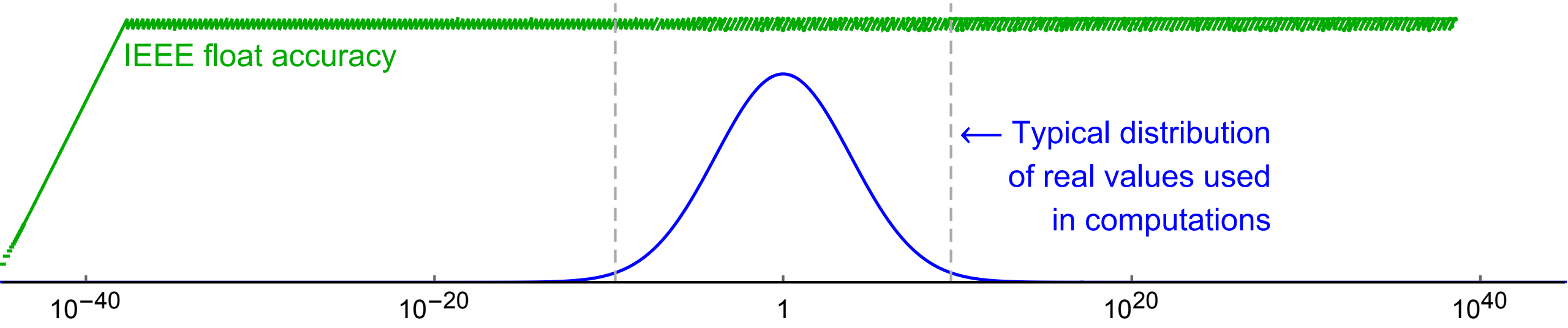


What reals should we seek to represent?



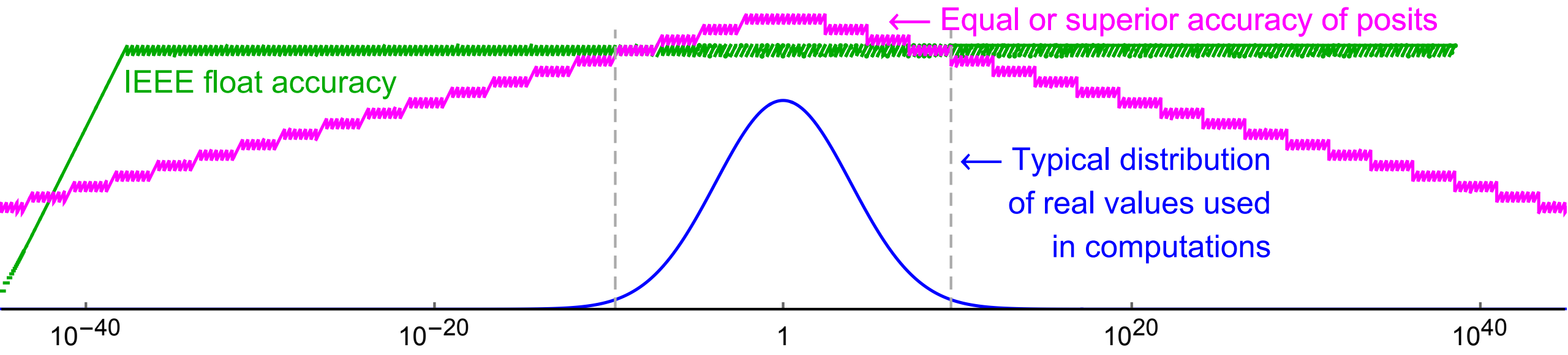
Studies show *very* rare use of values outside 10^{-13} to 10^{13}.
Central Limit Theorem says exponents distribute as a bell curve.

IEEE floats have about 7 decimals accuracy, flat except on the left



This shows 32-bit float accuracy. Dynamic range of 83 decades.
For 64-bit floats, exponent range is even sillier: *631* decades.
Is *flat* accuracy over a huge range really what we need?

Posits provide concise *tapered accuracy*

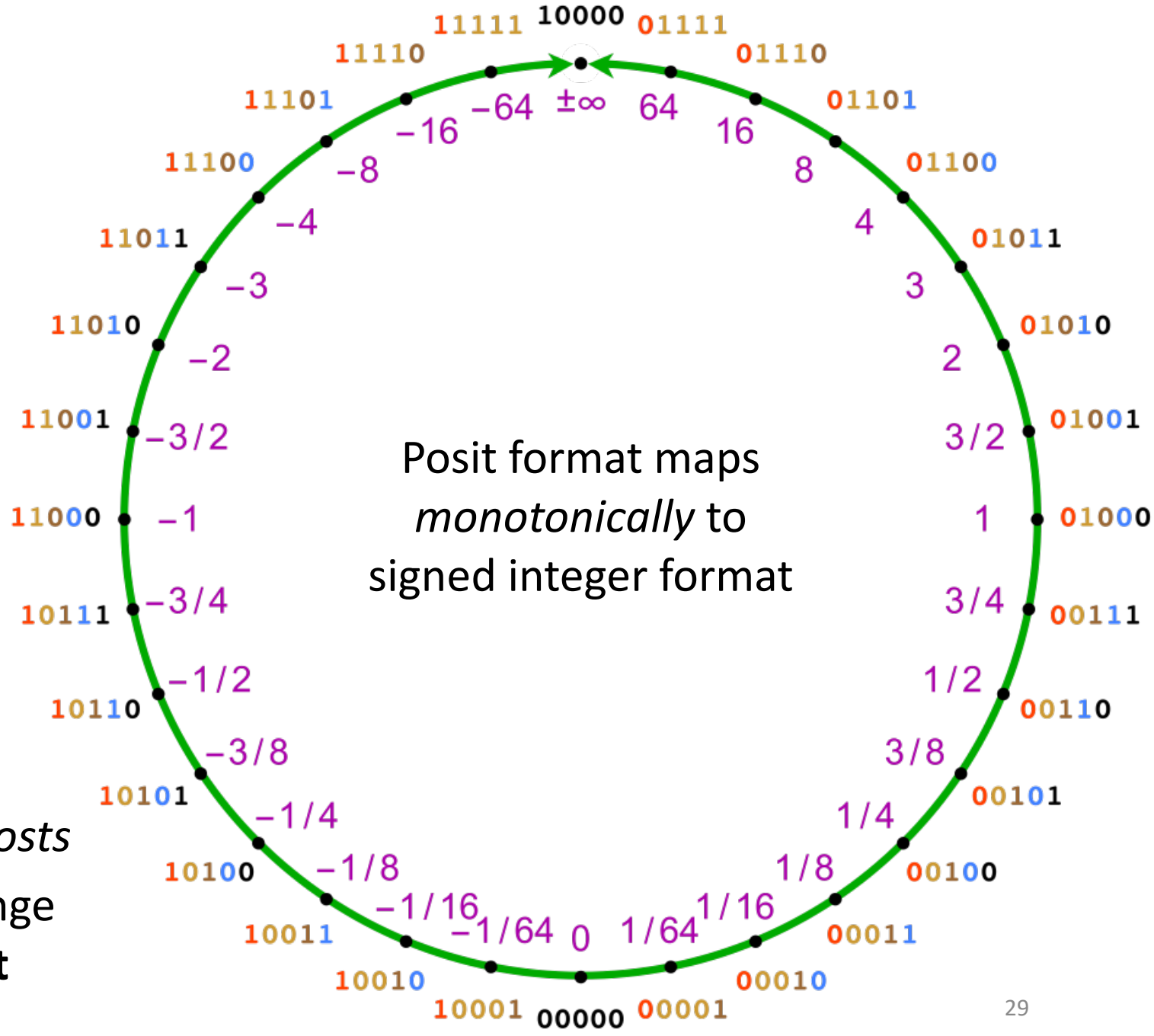


Posits have same or better accuracy on the vast majority of calculations, yet have *greater* dynamic range.

This is only *one* major advantage of posits over floats.

Posit arithmetic

- Better accuracy with fewer bits
- Bitwise **repeatability** and **portability** across systems
- Clean, mathematical design
- 32-bit posits can often replace 64-bit floats
- 16-bit posits can often replace 32-bit floats
- Reduces *energy*, *power*, *storage*, *bandwidth*, and *programming costs*
- Like parallel computing, the change will be a lot of work, but **worth it**



The *quire* reduces need for "double precision"

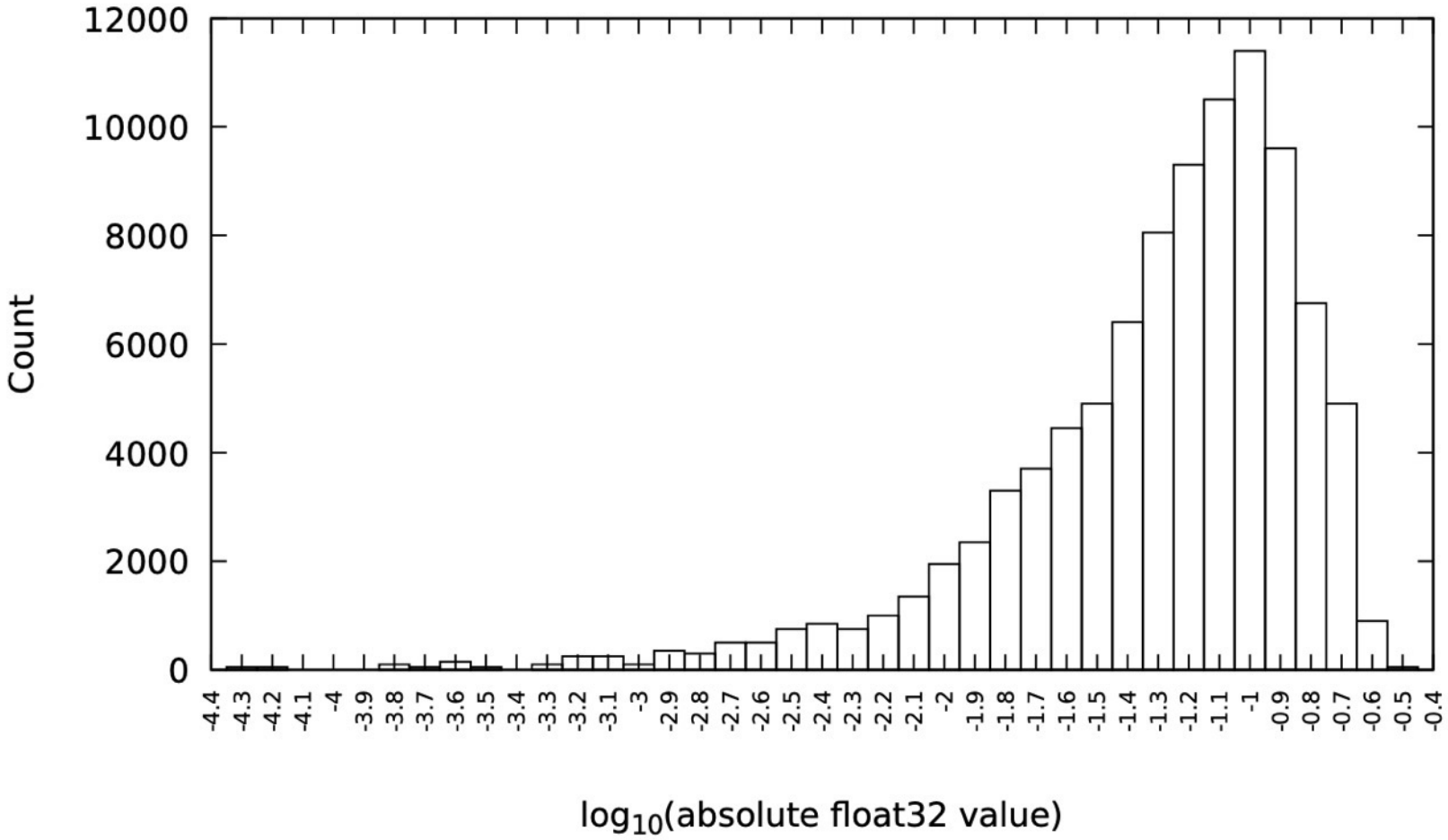
- Every *n*-bit posit environment has a 16*n*-bit **fixed-point accumulator**, the *quire*
- Exact dot products guaranteed up to ~2 billion-long vectors (2^{31})
- Exact signed sum accumulation guaranteed to even longer lists (3×10^{45} for 32-bit)
- Linear algebra is more accurate, **gives identical answers in parallel and serial**
- Strassen matrix multiplication is both faster *and* numerically safer for first time
- Think of it as a vector of 16 integers. Hardware can be quite fast (Berkeley).

Most of the reasons programmers use mixed-precision (especially double precision) disappear because of the quire.

16-bit formats tested for Deep Learning

- bfloat16 (“brainfloat”): 8 exponent bits, proposed by Google
- IEEE16: 754 Standard binary16 with 5 exponent bits
- IEEE16_6: Follows IEEE rules but with 6 exponent bits
- IEEE16_7: Follows IEEE rules but with 7 exponent bits
- DLFloat: 6 exponent bits, proposed by IBM;
  different rounding rules, no subnormals, other simplifications
- posit16_*es* : 16-bit posit with *es* exponent bits, *es* = 1, 2, 3;
  the old Standard used *es* = 1, new Standard is *es* = 2

Distribution of values needed for AI



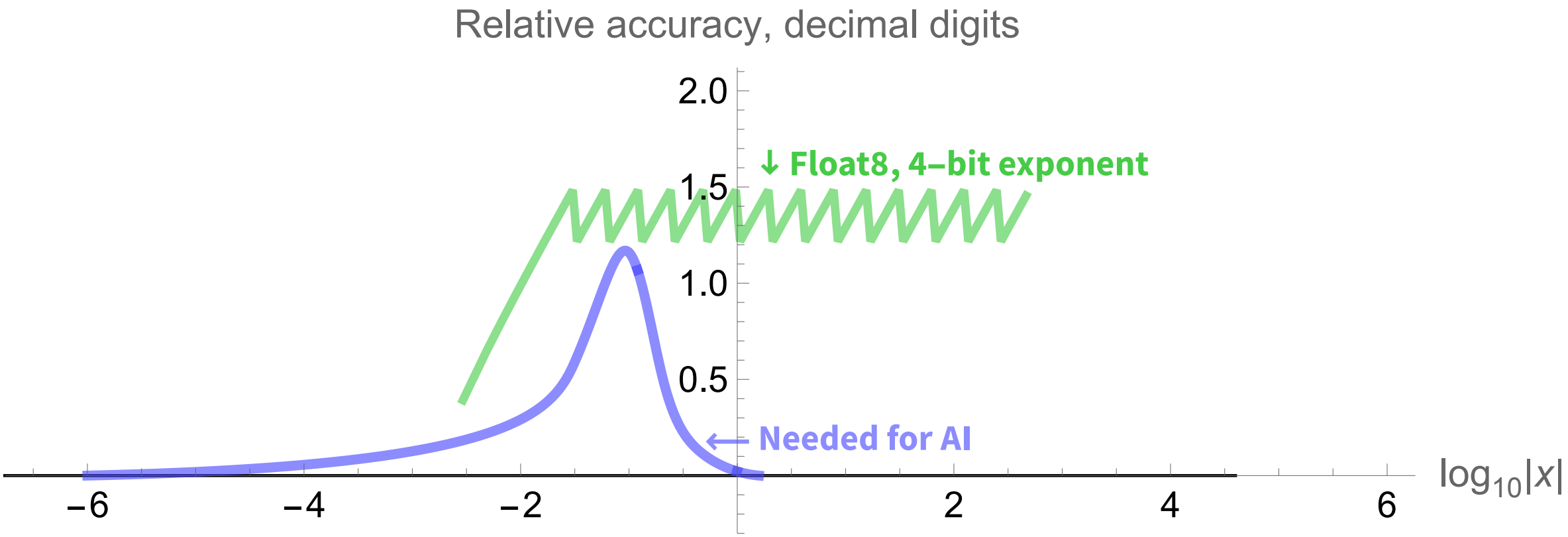
Posits can match this distribution almost perfectly.

Machine Learning Results – Top-1 Accuracy

Dataset/ Model	cifar10/ convnet	cifar100/ NIN	Imagenet/ Squeezenet	Imagenet/ Alexnet	Imagenet/ Resnet-18
IEEE32 "gold standard"	78.70%	56.06%	56.04%	57.04%	67.88%
Google bfloat16	76.02%	0.96%	0.32%	52.40%	61.88%
IEEE16 (Standard)	73.02%	NaN	0.00%	53.08%	NaN
IEEE_6	78.56%	46.28%	54.72%	46.48%	68.00%
IEEE_7	78.74%	NaN	0.24%	9.96%	67.52%
DLFloat16	77.96%	45.48%	54.24%	46.56%	47.60%
Posit16_2 (Standard)	77.74%	53.92%	56.80%	53.60%	67.64%
Posit16_3	79.72%	53.74%	56.48%	53.16%	67.60%

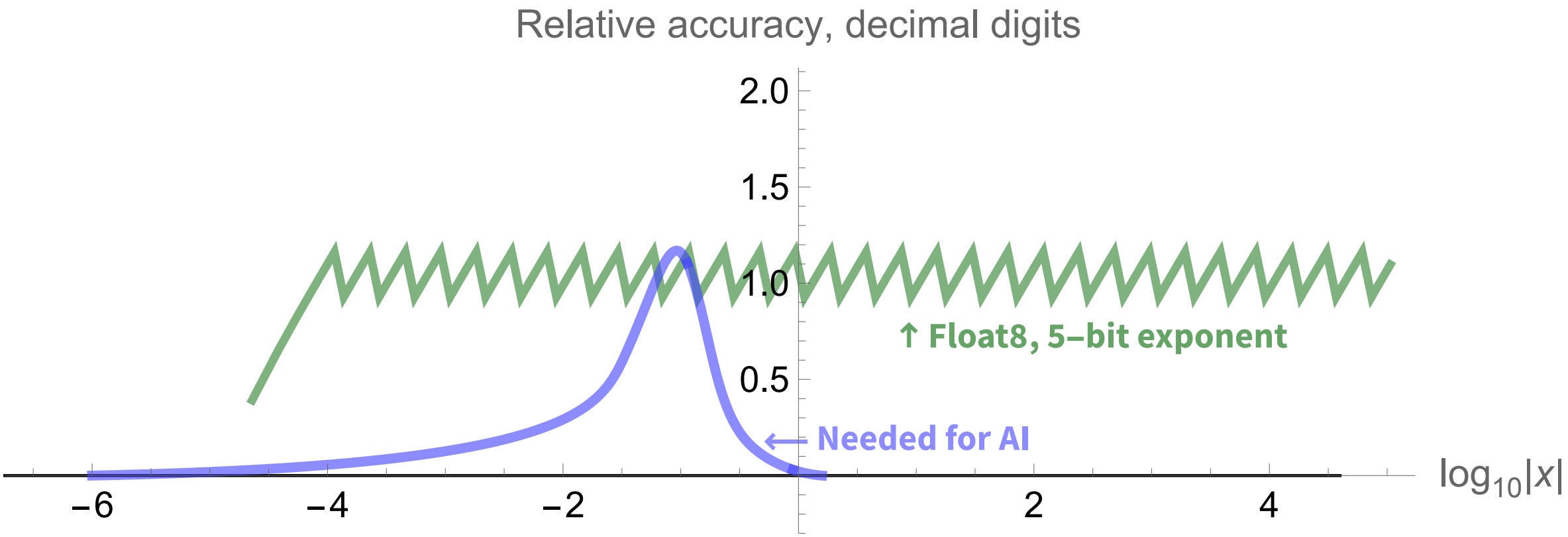
Standard posit16 performs consistently well
across different networks and datasets.
16-bit posits sometimes do better than **32-bit** IEEE floats!

8-bit floats with 4 exponent bits for AI



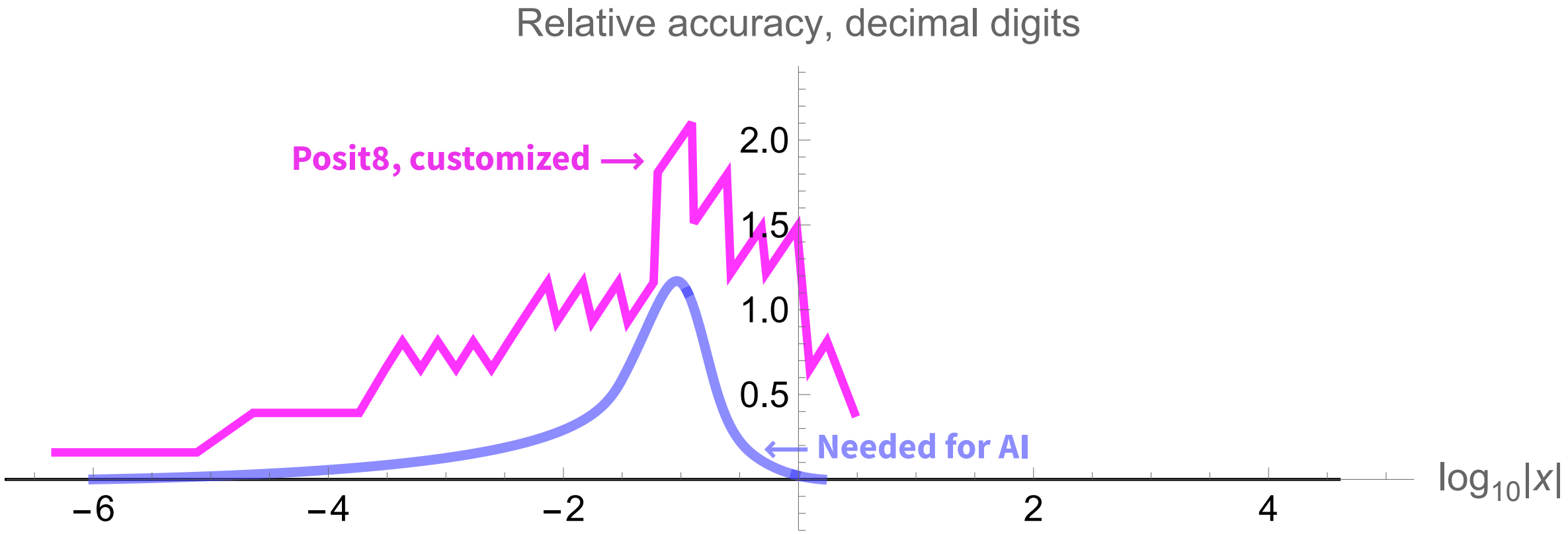
Insufficient dynamic range, even if you shift left by scaling.

Increase FP8 exponent bits to 5 for more range



Insufficient accuracy where it is most needed.

Posit8 distribution can be customized.



Near-perfect fit for both training and inference.

Generalized 8-bit posits can train MNIST-LeNet

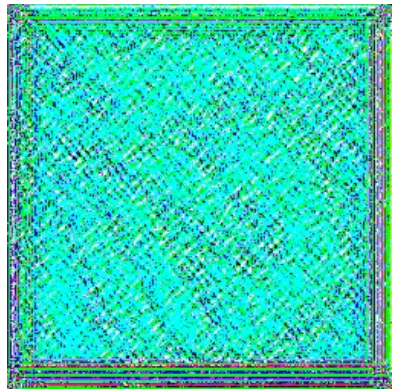
Format	Accuracy
Float32 (IEEE 754) Baseline	98.70%
Posit8, *es* = 1, *ebias* = –2	97.14%
Posit8, *es* = 1, *ebias* = –2, *rs* = 6	97.76%
Float8 with scaling (*es* = 4)	86.42%
Float8 with scaling (*es* = 5)	69.56%

Same experiment setup as with 16-bit posits and floats: All weights, activations, and gradients are maintained in 8 bits

ESRGAN (Enhance resolution)



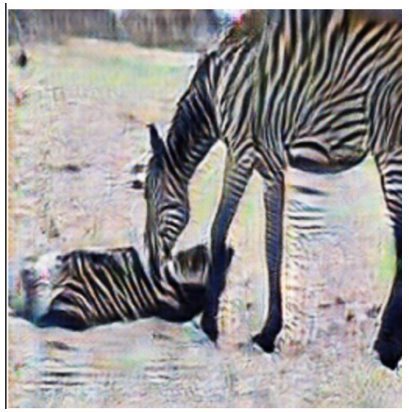




Horse2zebra



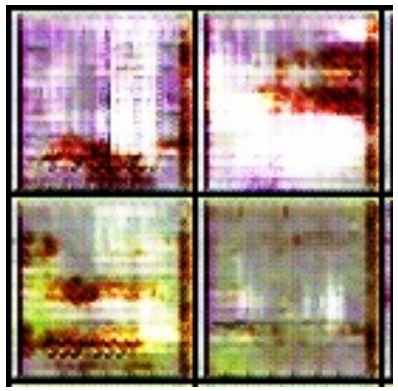




DCGAN Bedroom

Input vector in latent space (e.g. [0.1,0.23,…]



Input **FP32** **FP16** **P8** **FP8**

Training results

	FP32	FP16	P8+	P8	FP8
CGAN-MNIST	67.31	64.58	70	75.41	423.28
DCGAN-Lsun	43.12	49	44.82	49.54	318.07
Horse2zebra	77.2	64.3	61.1	70.1	75.63
Summer2winter	77.6	78.52	77.12	75.67	84.01
Zebra2horse	134.2	134.5	138.36	148.49	138.13
Winter2summer	75	73.46	72.79	74.75	77.16
ESRGAN (PSNR)	24.67	24.69	24.74	24.53	4.62

Normal GAN (FID) lower is better.
ESRGAN (PSNR-dB): higher is better.

FP16: Nvidia mixed precision training APEX (FP16 and FP32)
P8+: Posits with loss scaling and exponent bias
P8 : Posits without loss scaling or exponent bias
FP8: IBM's hybrid training float8 (*es* = 4, 5) mixed with FP(1,6,9)

11 Design Principles for NGA

1. Distribution of representable values closely resembles the distribution needed for exact calculations. ✓
2. No redundant bit patterns. Every bit counts. ✓
3. No hidden “modes” of operation. Source program and data suffice to determine every bit of the output. ✓
4. All math operations must be correctly rounded. ✓
5. It should be easy to change precisions, like it is for integers. ✓
6. Error results must always propagate through to the final output. ✓

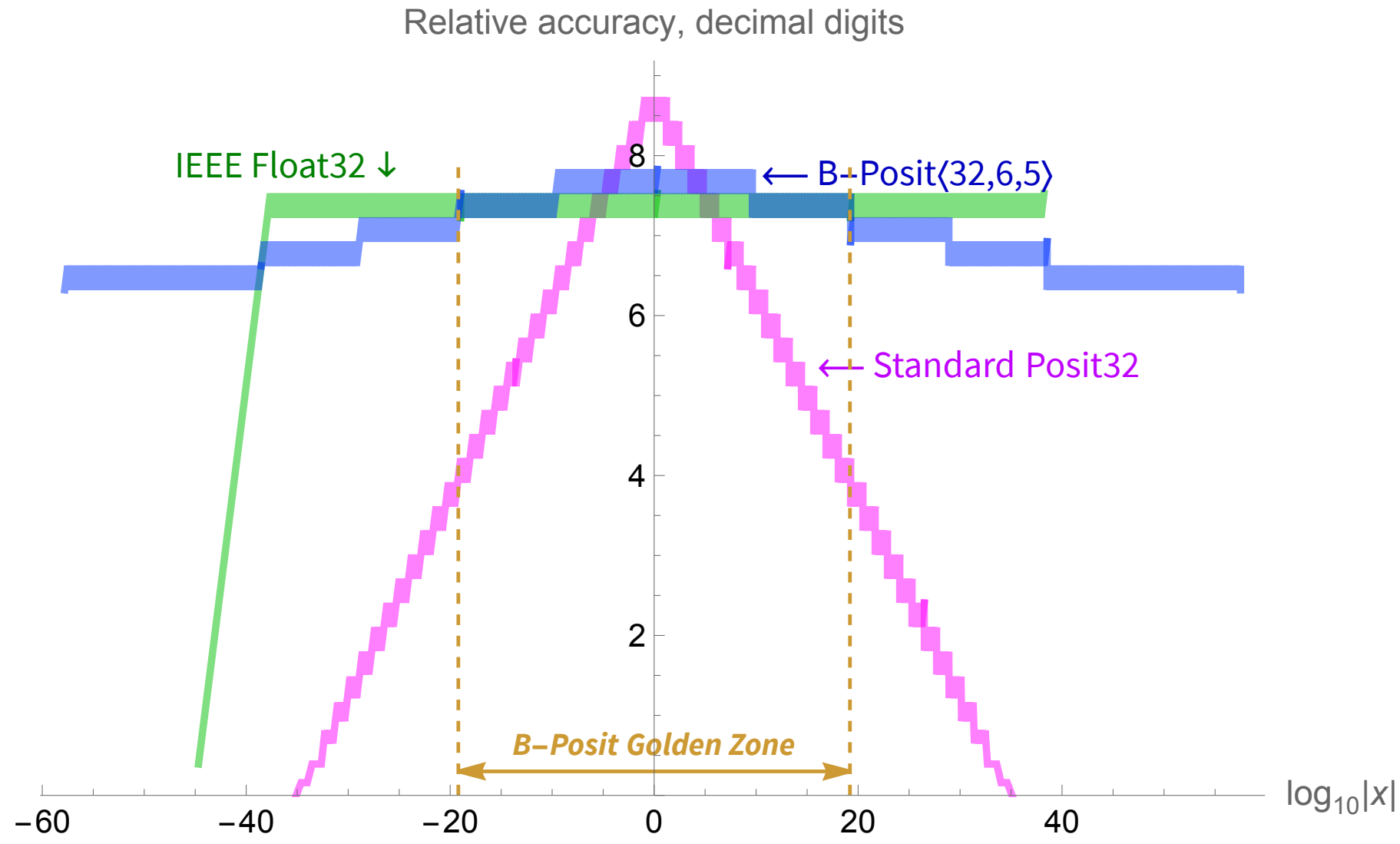
11 Design Principles for NGA (cont.)

7. It should be simple to build in hardware, and fast. ✓

In the "nice to have" category:

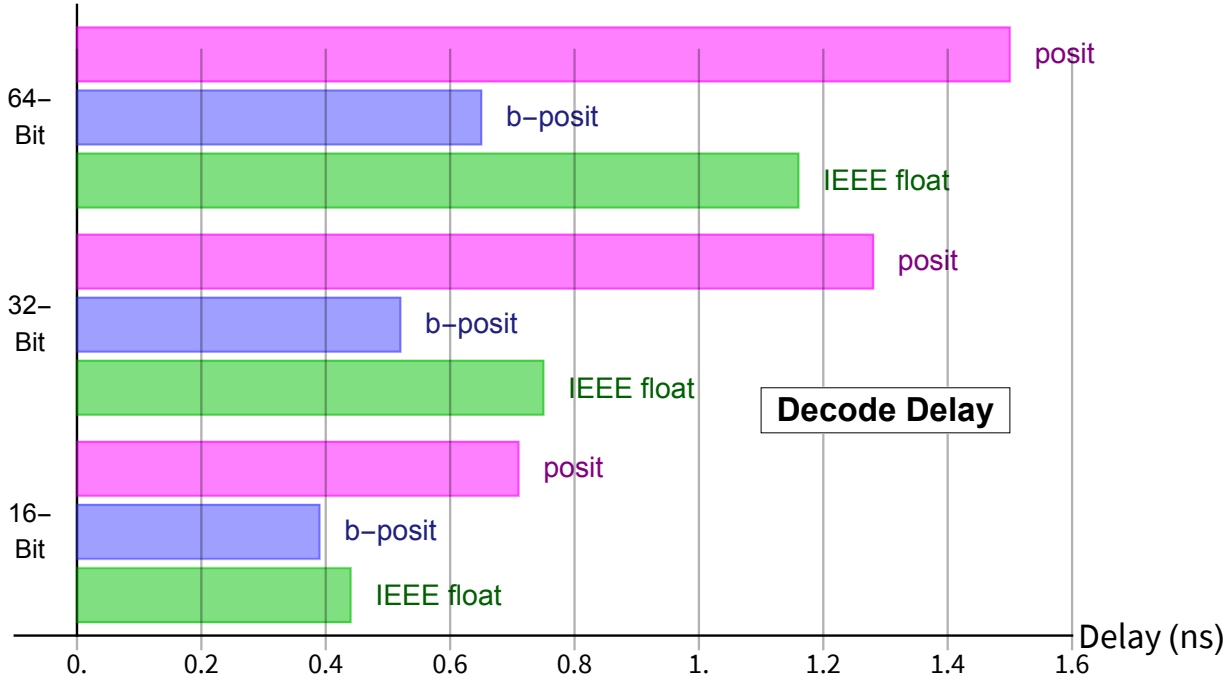
8. It should be easy to scale to any number of bits. ✓
9. Real numbers should be ordered like integers with the same bit strings. (No extra hardware for comparison operations). ✓
10. Negating a real should be the same as negating it as an integer. ✓
11. The results of application computations should be close to what they would be if infinite precision is used. an improvement…

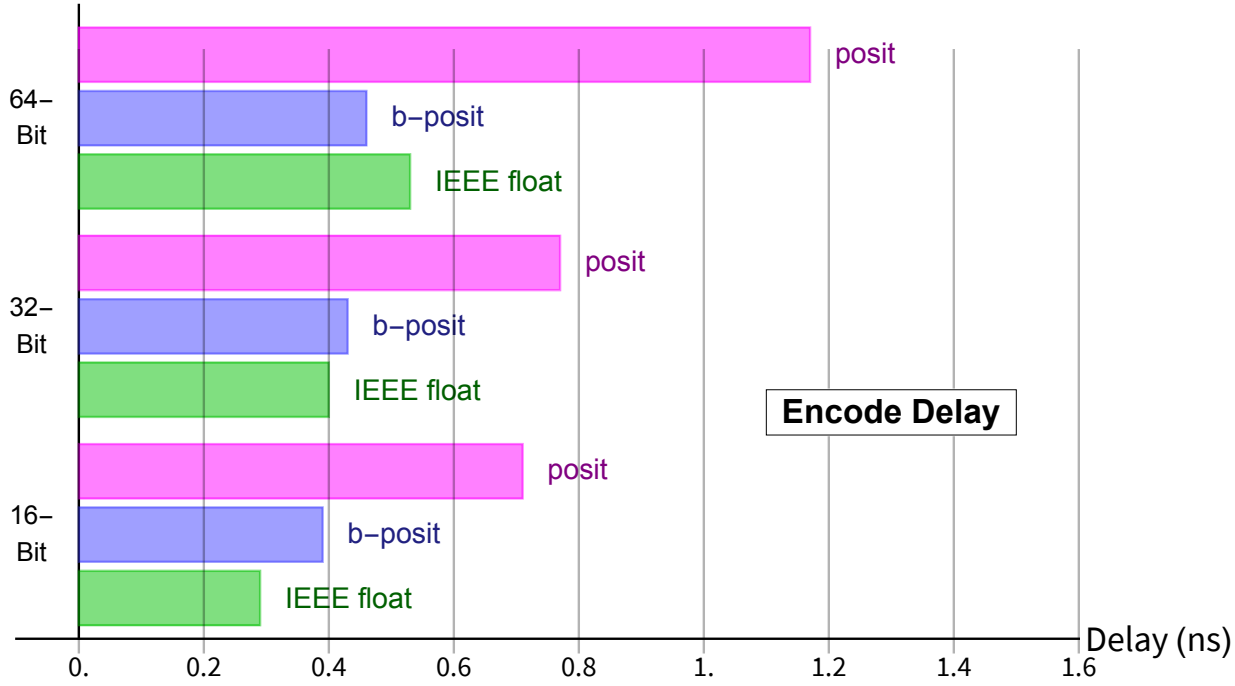
A Recent Breakthrough: The B-Posit



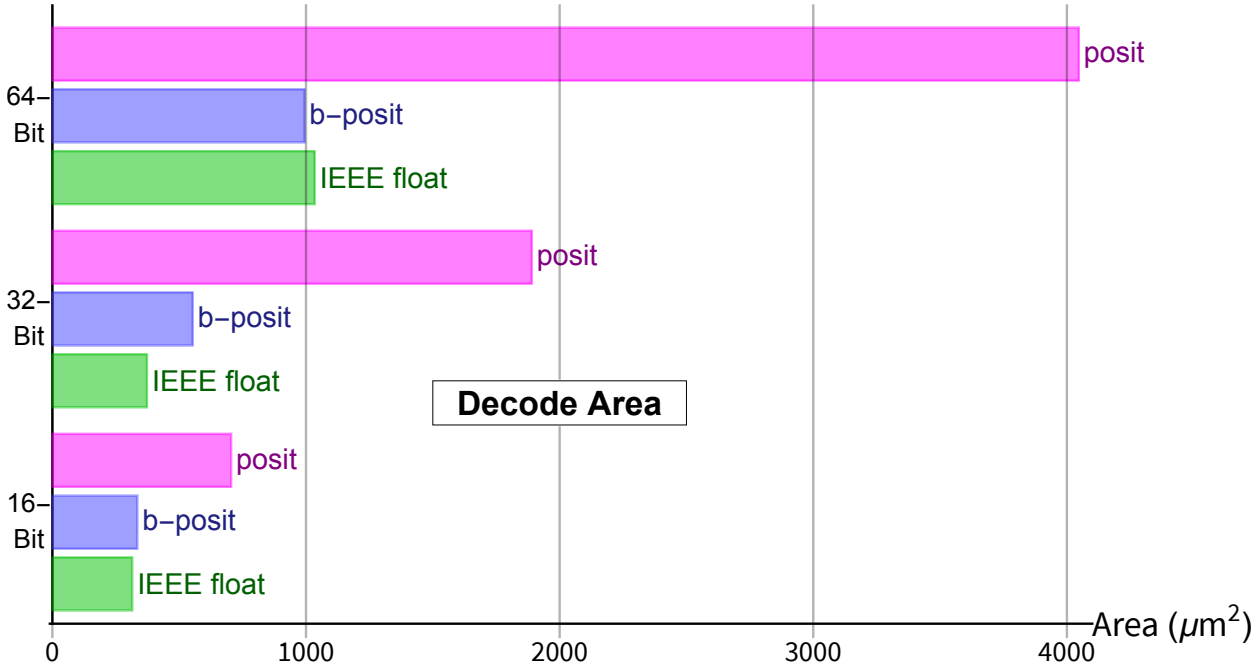
- Limit the maximum regime size
- Superior dynamic range
- Superior accuracy over vast range (Golden Zone)
- Greatly simplified hardware
- Lower latency

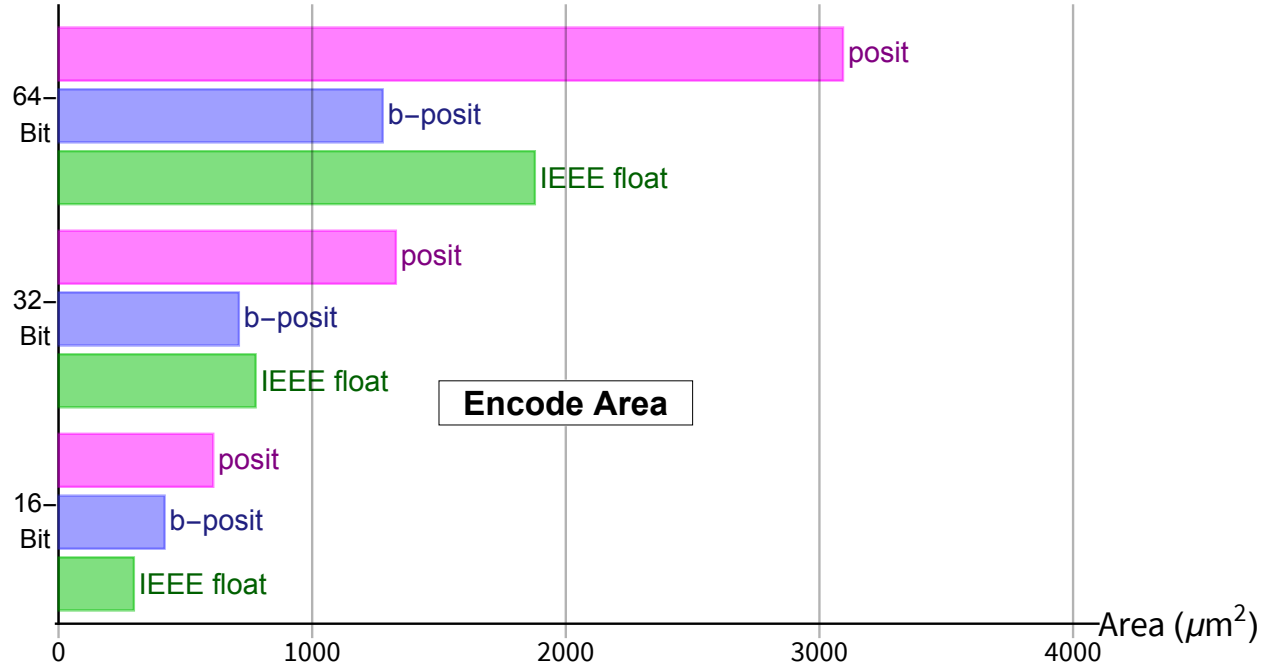
Worst-Case Decode-Encode Time





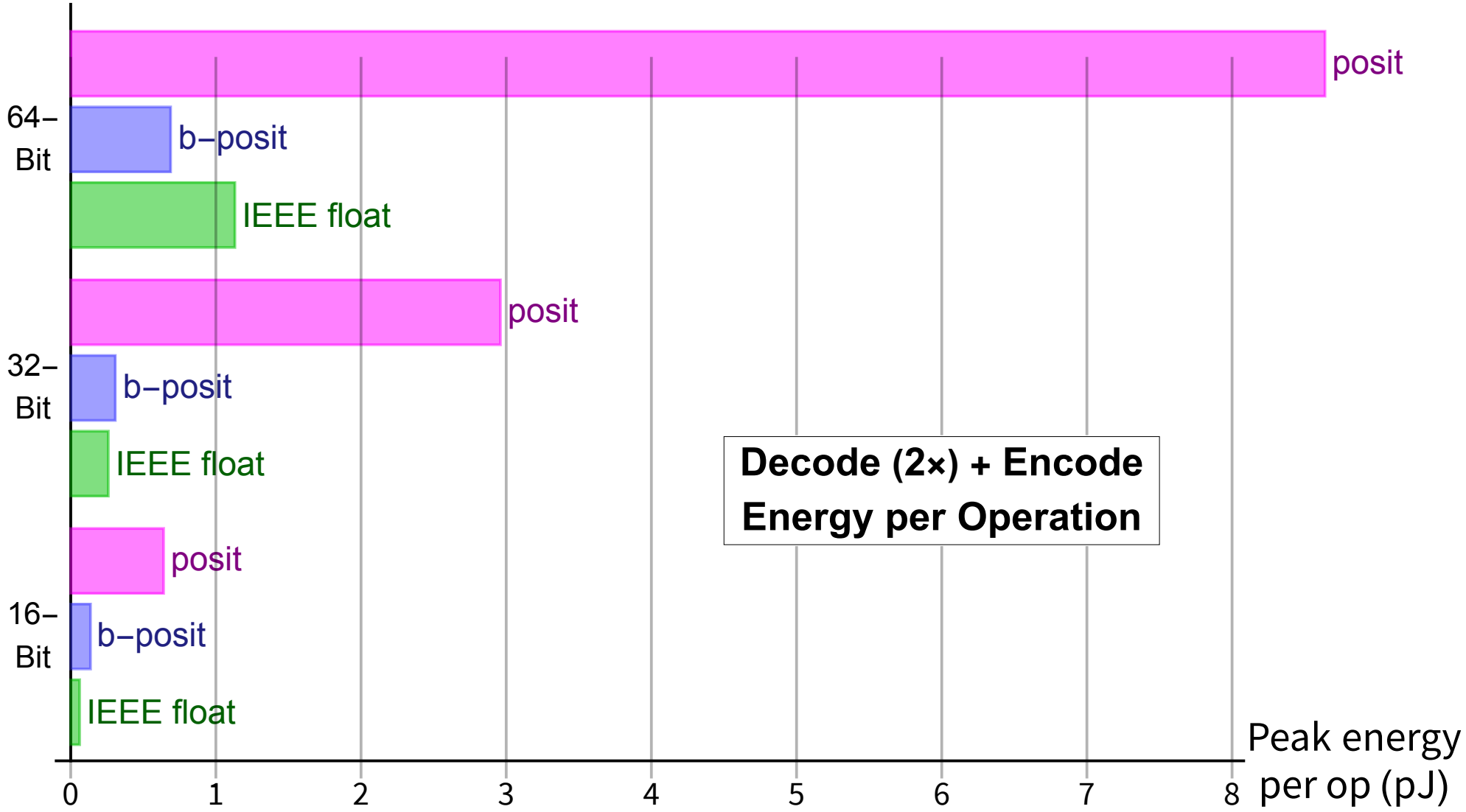
Decode-Encode Chip Area for Same Precision





These are early results. More improvements likely in the future.

Worst-Case Power Consumption

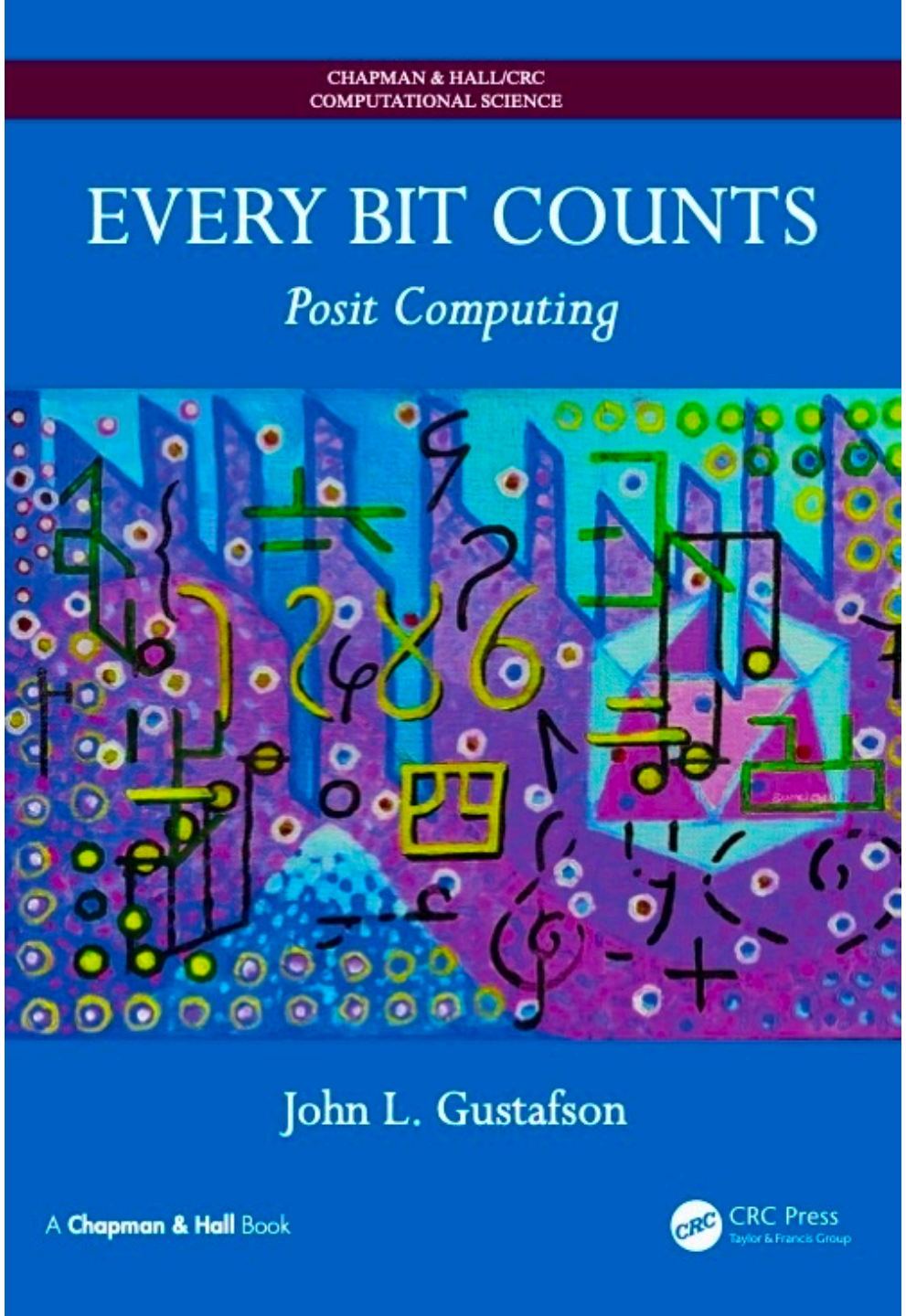


Shameless Plug…

9 years in the making

464 pages, all in full color

Shows how to customize real number formats to any workload, including AI.



Summary

- Posits can allow *smaller precisions* to achieve acceptable accuracy, making posit arithmetic cheaper, faster, more power/energy efficient, smaller chip area.
- Bitwise reproducibility for real number programs is achievable for the first time.
- Low precision posits work well for AI apps.
- Posit adoption has caught fire, esp. on RISC-V

Thank you!
For more information on posits:
www.posithub.org


TESLA
Roadster Sport
ARIZONA
POSIT