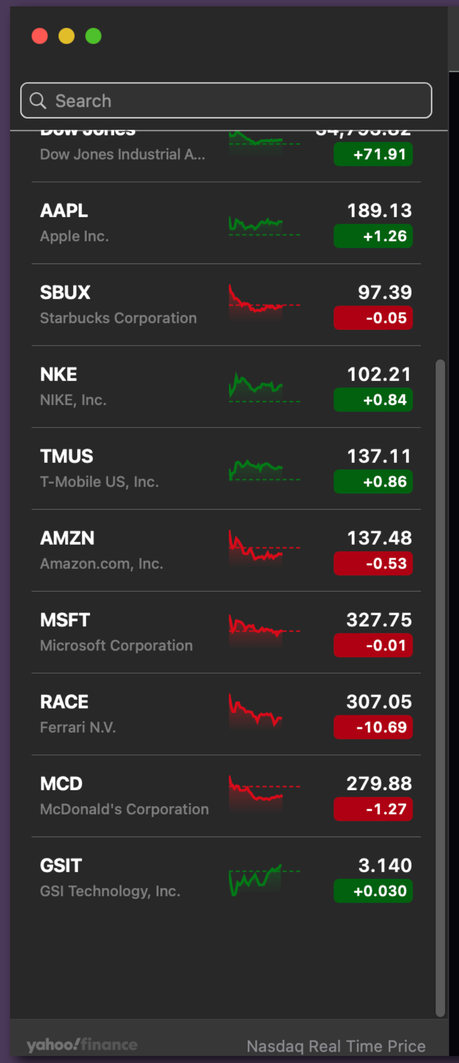
BRIAN BECKMAN, 1 SEPTEMBER 2023

ASSOCIATIVE COMPUTING FOR HDC

APU: ASSOCIATIVE PROCESSING UNIT

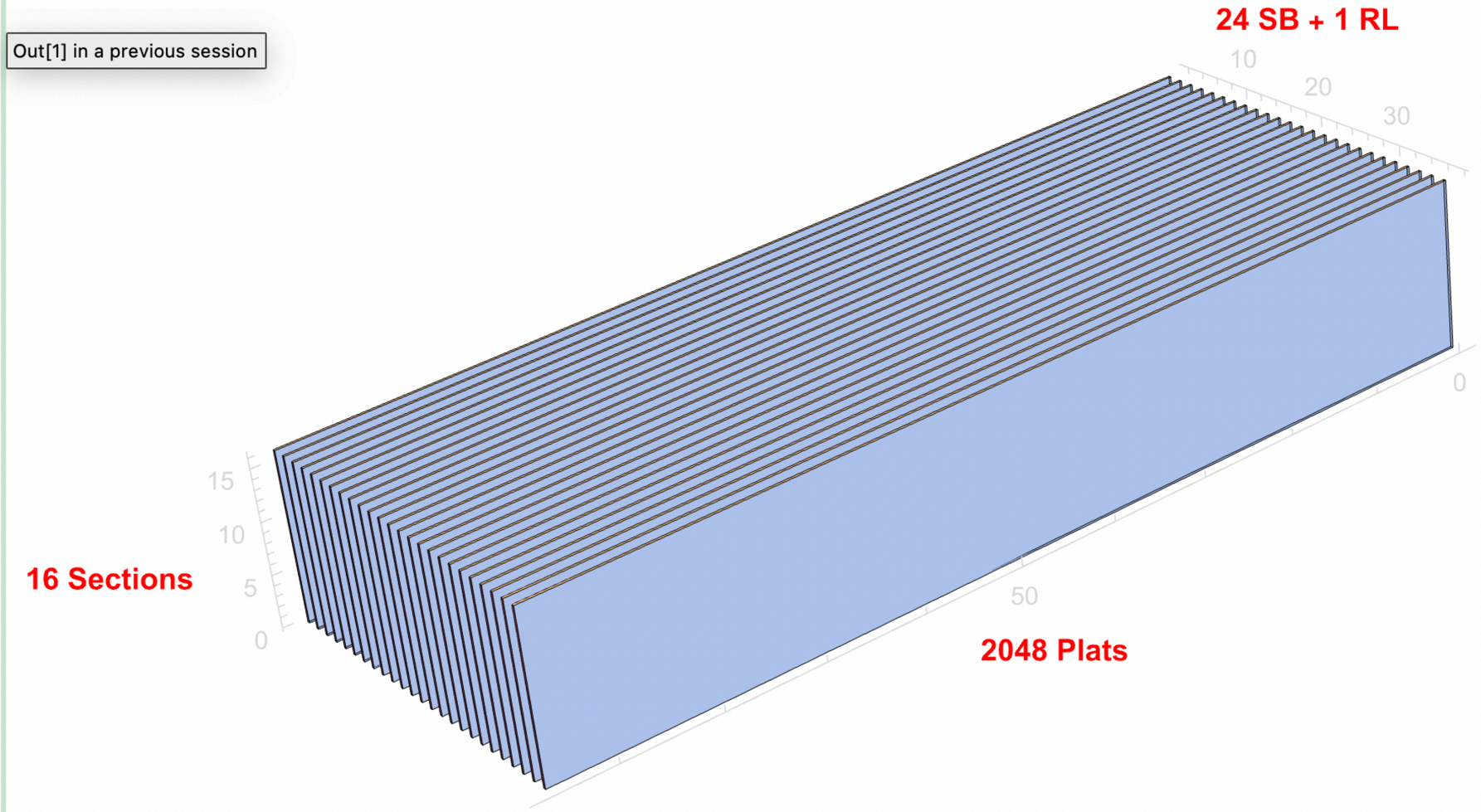
SRAM (LOW-POWER) COMPUTE-IN-MEMORY

- **GSI**
  - established vendor of rad-hardened SRAM
- **Not a startup**
  - on NASDAQ
- **APU is a new venture for this 25-year-old company**



- 3-D Memory Model: *plats* (matrix columns) *sections* (matrix rows) per half bank
- Each Section holds four 8192-K BHVs per core; 4 cores/chip
- SIMD ops on up to 6144 BHVs at once
- We have emulators in Mathematica, numpy, and C++

Out[1] in a previous session



64 copies of that, for about 50 Mibi bits:

```
16 * 2048 * 24 * 64
```

```
50 331 648
```

APU: ASSOCIATIVE PROCESSING UNIT

SRAM (LOW-POWER) COMPUTE-IN-MEMORY

- **Gemini-I: ~6144 bit-vectors of length 8192 in 4 cores of each chip**
- **Rich Boolean Logic ISA (Instruction-Set Architecture)**
  - Based on discovery that pulling multiple bitlines in standard SRAM yields NOR
- **Belex: Optimizing Assembler Embedded in Python (soon to be FOSS)**
  - Register Allocation, Spill-Restore, Instruction Scheduling, Inlining, Peephole, …
- **Copperhead: Application Programming in LPython (FOSS compiler, C-speed)**
- **Tartan: Higher-Level Belex Libraries for Linear Algebra in GF(2)**

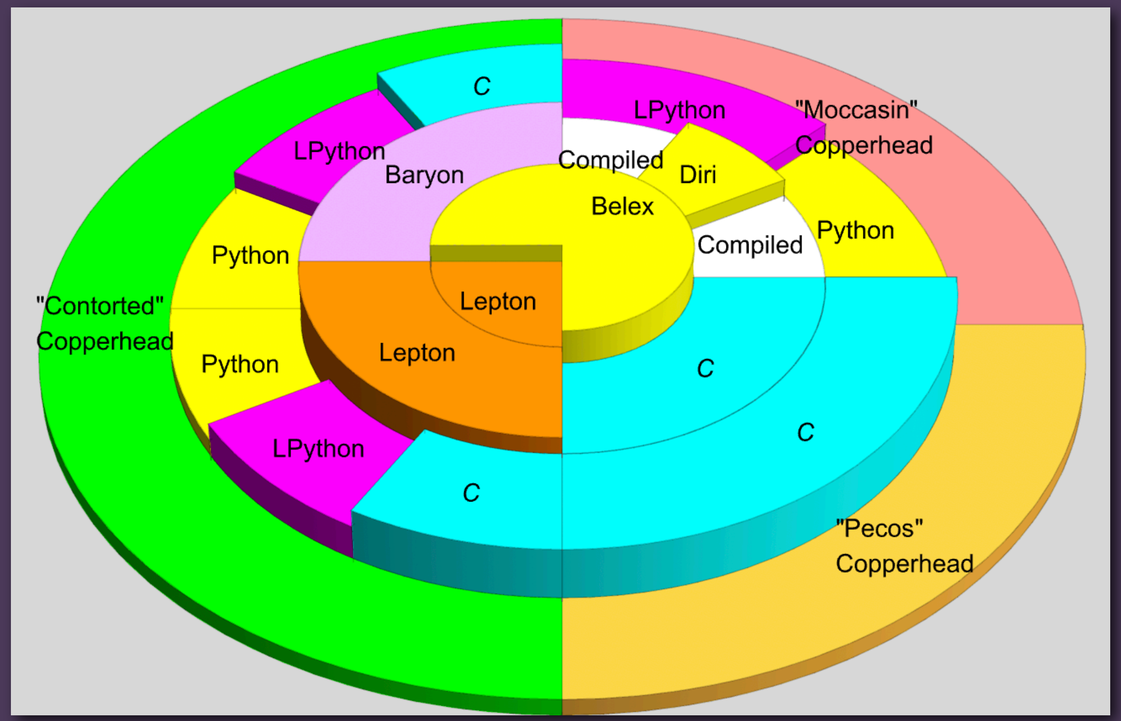
APU: ASSOCIATIVE PROCESSING UNIT

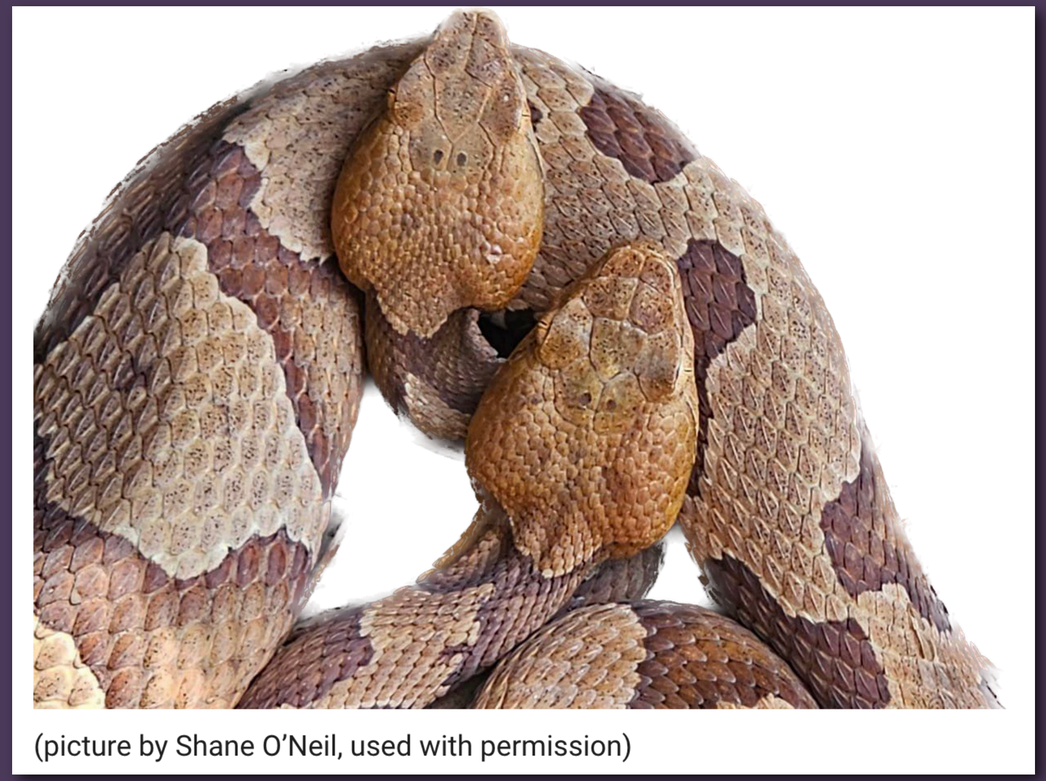
SRAM (LOW-POWER) COMPUTE-IN-MEMORY

- **Gemini-I: 500 MHz, 28 nm, PCIe, COTS SKU**
  - Vector Math, Similarity Search, Synthetic Aperture Radar (SAR)
- **Gemini-II: Faster Clock (1.4 GHz, 14 nm)**
  - More L1 memory (384 VRs)
  - Smaller, Cheaper, Faster, Simpler boards
  - Taping out Now, 2023
- **Gemini-III: 5 nm; Much more Main Memory (late 2024)**

SOFTWARE STACK

- Python-based SDK with full ecosystem: debugging, visualization, etc.
- “Copperhead”: Chip emulators, compiled and interpreted Python modes





(picture by Shane O’Neil, used with permission)

GRANDMOTHER EXAMPLE

Dylon gave me a little fix and I now have the grandmother example running 100% in the APU, 2476 cycles.

This does TRAINING and INFERENCE of the following:

15 examples of ((M and F) imply G)

((x mother of y) and (y father of z) imply (x grandmother of z))

1 query: Anna is mother of Bill and Bill is father of Cid

answer: Anna is grandmother of Cid

Full ML inference with tiny training set 100% in APU.

BELEX: ASSEMBLY IN PYTHON

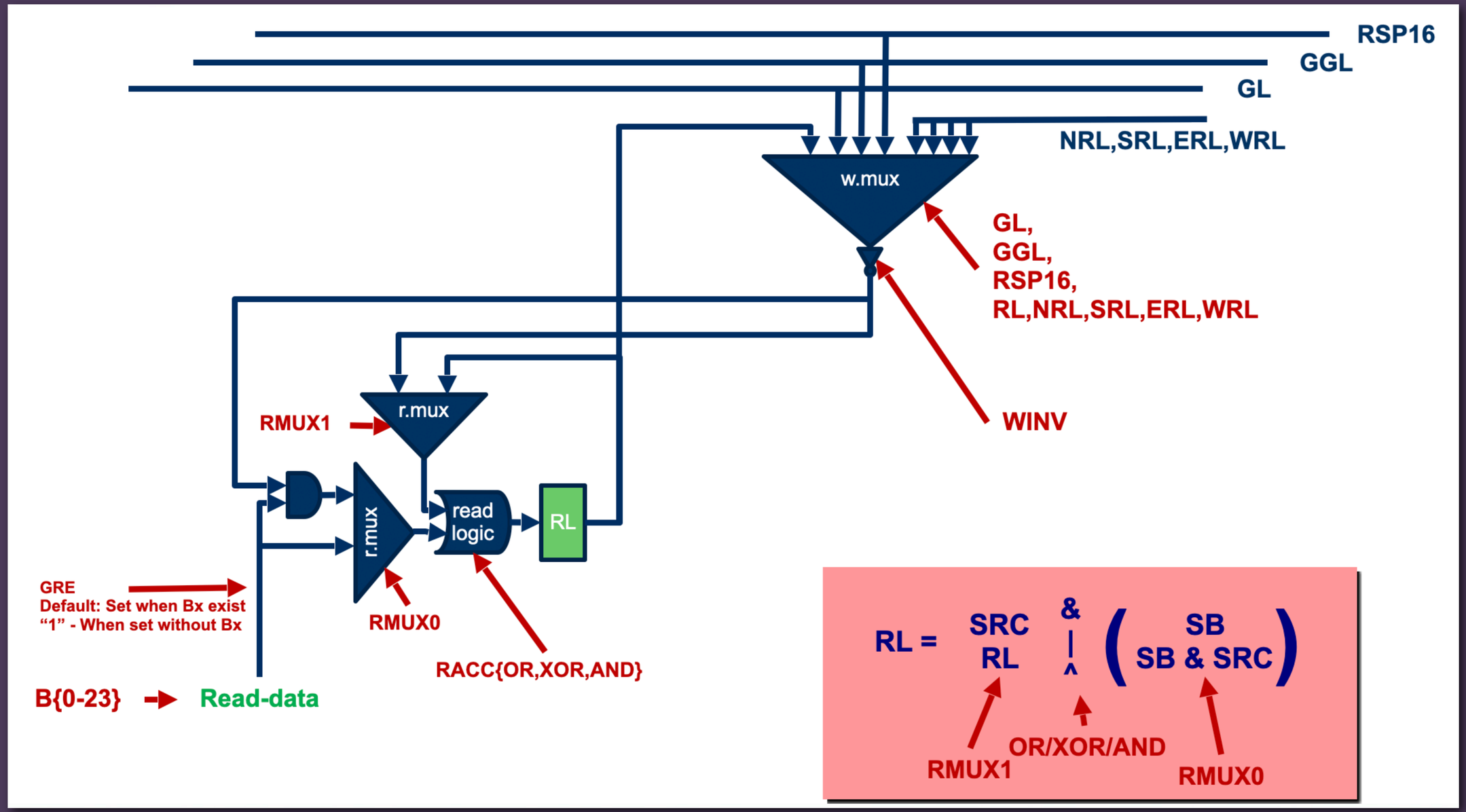
- Each logic op is SIMD per core on 32K bits (4 BHV-8192s), MIMD on 4 cores

```
105  @belex_apl
106  def hdc_ternary_majority_const_section (
107          Belex,
108          dst : VR,
109          a   : VR,
110          b   : VR,
111          c   : VR,
112          sec : Section) -> None:
113      """In hdc, ADD is majority. This routine works
114      only if all VRs are different.
115      """
116      RL[sec]  <= a() & b()
117      RL[sec]  <= b() & c() | RL()
118      RL[sec]  <= c() & a() | RL()
119      dst[sec] <= RL()
```

Rich Boolean Logic in

SRAM is inherently low-power
(no refresh cycles, low retention current and voltage)

LARGE SET OF HARDWARE COMMANDS



LARGE SET OF HARDWARE COMMANDS

FOUR PARALLEL LANES FOR DATA-INDEPENDENT COMMANDS

To count the number of commands, the number of SB notations is $24 + 24^2 + 24^3 = 14\,424$ because there may be up to three RNs in an SB notation. The number of SRC notations is 16 including inverses. The number of section masks is 65 536.

Command	section	destination	operator	source	count
1, 2	msk	RL	:=	{0, 1}	131 072
3-5, 10-15, 18-20	msk	RL	{ := , |= , &= , ^= }	{SB, SRC, SB & SRC}	64 284 000 256
6, 7	msk	RL	:=	SB{ | , ^ }SRC	30 249 320 448
8, 9	msk	RL	:=	{~SB & SRC, SB &~SRC}	22 686 990 336
16, 17	msk	RL	&=	{~SB, ~SRC}	7 562 330 112
undoc	msk	SB	:=	SRC	1 048 576
undoc	msk	{GL, GGL, RSP16}	:=	RL	196 608

(5)

The numbers are only approximate because not all combinations are allowed, but the machine supports roughly 100 billion different commands.

LARGE SET OF HARDWARE COMMANDS

FOUR PARALLEL LANES FOR DATA-INDEPENDENT COMMANDS

```
5.1 WRITE LOGIC
~~~~~~~~~~~~~~~

  in shorter, BNF-style notation

  <SRC> is one of (INV_)?[GL, GGL, RSP16, RL [NEWS]RL]
                  NOTA BENE: <SRC> does NOT include SB!

  As many as three VRs may be written in one clock via the
  following SB notation:

  msk: SB[x]       = <SRC>, e.g., SB[9] = RL
  msk: SB[x, y]    = <SRC>, e.g., SB[3, 14] = GL
  msk: SB[x, y, z] = <SRC>, e.g., SB[1, 2, 3] = WRL

  where x, y, z are each one of RN_REG_0 .. RN_REG_15.

  SB[x] is shorthand for SB[x, x, x],
  SB[x, y] is shorthand for SB[x, y, y]
```

EXAMPLE: CARRY-PREDICTION ADDER

FOUR PARALLEL LANES FOR DATA-INDEPENDENT COMMANDS

```
with apl_commands("instruction 1"):
    RL["0xFFFF"] <= x()
with apl_commands("instruction 2"):
    RL["0xFFFF"] ^= y()
    GGL["014589CD"] <= RL()
with apl_commands("instruction 3"):
    x_xor_y["0xFFFF"] <= RL()
with apl_commands("instruction 4"):
    cout1["048C"] <= RL()
    cout1["159D"] <= GGL()
    RL["26AE"] <= x_xor_y() & GGL()
    RL["014589CD"] <= x() & y()
with apl_commands("instruction 5"):
    cout1["26AE"] <= RL()
    RL["37BF"] <= x_xor_y() & NRL()
    RL["159D"] |= x_xor_y() & NRL()
    RL["26AE"] <= x() & y()
with apl_commands("instruction 6"):
```

HDC: CONSTANT VR VERSUS CONSTANT SECTION

```
lex_apl
 hdc_ternary_majority_const_section (
      Belex,
      dst : VR,
      a   : VR,
      b   : VR,
      c   : VR,
      sec : Section) -> None:
 """In hdc, ADD is majority. This rou
 only if all VRs are different.
 """
 RL[sec]  <= a() & b()
 RL[sec]  <= b() & c() | RL()
 RL[sec]  <= c() & a() | RL()
 dst[sec] <= RL()
```

```
@belex_apl
def hdc_ternary_majority_const_vr (
        Belex,
        dst : VR, d : Section,
        src : VR, a : Section,
                  b : Section,
                  c : Section) -> None:
    """Compute majority for three sections in a
    single VR and propagate it to a section d in
    dst, a different VR. d may not equal any of the
    other sections as src[d] is scratch space. That
    fits with our assumption that at most 15
    sections of a single VR are live, with one
    reserved for scratch.
    """
    RL[a] <= src()
    GL[a] <= RL()
    RL[b] <= src() & GL()
    GL[b] <= RL()
    RL[d] <= GL()

    RL[b] <= src()
    GL[b] <= RL()
    RL[c] <= src() & GL()
    GL[c] <= RL()
    RL[d] |= GL()
```

BELEX PYTEST OF APU HDC LIBRARY

```
test_hdc_hamming_minimal_tuning PASSED                                   [  5%]
test_hdc_mul_const_section PASSED                                        [ 10%]
test_hdc_hamming_minimal PASSED                                          [ 15%]
test_hdc_grandmother_example_baryonic PASSED                             [ 21%]
test_hdc_mul_const_vr PASSED                                             [ 26%]
test_hdc_add_const_section PASSED                                        [ 31%]
test_hdc_groundpound_all_sections PASSED                                 [ 36%]
test_hdc_9_maj_via_3_const_section PASSED                                [ 42%]
test_bytes_section_round_trip PASSED                                     [ 47%]
test_hdc_distributivity_const_section PASSED                             [ 52%]
test_hdc_fma_const_vr PASSED                                             [ 57%]
test_hdc_ternary_fma_const_section PASSED                                [ 63%]
test_hdc_15_maj_const_vr_two_phases_explicit_sections PASSED             [ 68%]
test_hdc_15_maj_const_vr_two_phases PASSED                               [ 73%]
test_hdc_9_maj_via_3_const_vr PASSED                                     [ 78%]
test_hdc_hamming PASSED                                                  [ 84%]
test_hdc_15_maj_const_vr PASSED                                          [ 89%]
test_hdc_distributivity_const_vr PASSED                                  [ 94%]
test_hdc_add_const_vr PASSED                                             [100%]
```

BELEX BARYON TESTS (COMPILED-MODE EMULATOR)

```
[DEBUG] Total number of failures: 0
[DEBUG] Total number of failures: 0
[DEBUG] Total number of failures: 0
[DEBUG] Total number of failures: 0
[DEBUG] Total number of failures: 0
[DEBUG] Total number of failures: 0
[DEBUG] Total number of failures: 0
[DEBUG] Total number of failures: 0
[DEBUG] Total number of failures: 0
[DEBUG] Total number of failures: 0
[DEBUG] Total number of failures: 0
[DEBUG] Total number of failures: 0
[DEBUG] Total number of failures: 0
[DEBUG] Total number of failures: 0
[DEBUG] Total number of failures: 0
belex-test -t baryon --shuffle tests/test_belex_hdc_library.py  76.56s user
grep --color=auto --exclude-dir={.bzr,CVS,.git,.hg,.svn,.idea,.tox} -i fail
```

GRANDMOTHER: INSTRUMENTED GROUND TRUTH

```
# relation utility
dump(rel_subject := Perm.random(), file_prefix: 'bhv_rel_subject_')
dump(rel_object := ~rel_subject, file_prefix: 'bhv_rel_object_')

# relations
# (SEGVIO dumping the VanillaBHV object; just dump the data)
mother_of = BHV.rand()
dump(mother_of.data, file_prefix: 'bhv_mother_of_')
father_of = BHV.rand()
dump(father_of.data, file_prefix: 'bhv_father_of_')
grandmother_of = BHV.rand()
dump(grandmother_of.data, file_prefix: 'bhv_gran_of_')

# extractors
mother_of_mother = rel_subject(mother_of)
dump(mother_of_mother.data, file_prefix: 'bhv_mofmo_')
mother_of_child = rel_object(mother_of)
dump(mother_of_child.data, file_prefix: 'bhv_mofch_')
```

- Adam's Implementation in BHV

```
id_6yghlglo
pf
q6
d_hnb47mp1
q
8rzrwmz
k53
uu5
imes_maj_mxy_f
xy_fyz_gran_ru
iebreaker_gran
22  bhv_gen_0_person_x_gran_rule_
22  bhv_gen_0_person_y_gran_rule_
22  bhv_gen_0_person_z_gran_rule_
22  bhv_gen_10_gxz_times_maj_mxy_
22  bhv_gen_10_maj_mxy_fyz_gran_r
22  bhv_gen_10_maj_tiebreaker_gra
22  bhv_gen_10_person_x_gran_rule
22  bhv_gen_10_person_y_gran_rule
```

- ~400 Files

```
#           _       _   _
#   _ _ ___| |__ _| |_(_)___ _ _  ___
#  | '_/ -_) / _` |  _| / _ \ ' \(_-<
#  |_| \___|_\__,_|\__|_\___/_||_/__/

# Let m, f, g be random bit-vectors representing relations
# (mother, father, and grandmother).

mother_of = bhv_from_session_file_prefix('bhv_mother_of_') \
    or BHV.rand()
diri.hb[relations_VR, :, mother_section] = \
    section_wise_from_bytes(mother_of.data)
# Check that all the round-tripping is done correctly.
mother_of_bools = diri.hb[relations_VR, :, mother_section]
mother_of_bytes = bytes_from_section_array(mother_of_bools, BHV_DATA_BY
mother_of_check = mother_of.data
assert mother_of_bytes == mother_of_check
```

- Belex Implementation

GRANDMOTHER EXAMPLE

```
# Method

Let $G_{particular}$ be the proposition "Anna is the
grandmother of Cid."

Let M be "Anna is the mother of Bill." Let F be "Bill is the
father of Cid." Let $H_{particular}$ be M and F.
Don't connect $H_{particular}$ to $G_{particular}$, yet.

Let $T_{random}$ be the conjunction (the and) of several
transitivity statements -- "'X is the mother of Y' and
'Y is the father of Z' imply 'X is the grandmother of Z'," for
randomly selected persons X, Y, and Z.

Test "if $T_{random}$ implies $H_{particular}$,
then $G_{particular} obtains, statistically at least."
```

GRANDMOTHER EXAMPLE IN PYTEST AND BARYON

```
fragment: hdc_grandmother_apply_rel; instruction: 2502; variable: dest_vr (least signif
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
[1 0 0 0 1 0 1 0 0 1 0 1 1 1 1 0 0 1 1 0 1 0 0 0 1 1 1 0 0 1 0 1]
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]

not unrelated: grandmother_calculation_checks_out = True, related_by_hand = False
PASSED
```

```
[DEBUG] Actual Values (out): plats [0, 8)
[DEBUG] - - - - - - - -
[DEBUG] 0 1 2 3 4 5 6 7
[DEBUG] - - - - - - - -
[DEBUG] 3 6 0 6 6 5 D 6
[DEBUG] 0 8 9 6 7 F 7 9
[DEBUG] 1 F 4 D 1 1 0 7
[DEBUG] 7 3 0 8 0 A 6 3
[DEBUG] - - - - - - - -

[DEBUG] Total number of failures: 0
[PASSED] hdc_grandmother_example_baryonic, EXAMPLE_0 (num_instructions=2305, num_commands=2501)
gdl_context_free()
gdl_exit()
gsi_sim_destroy_simulator()
[SUCCESS] hdc_grandmother_example_baryonic (num_instructions=2305, num_commands=2501)

~/tmp/belex-tests ~/tmp/belex-tests
~/tmp/belex-tests
Done.
belex-test -t baryon --no-generate   0.80s user 0.18s system 198% cpu 0.492 total
```

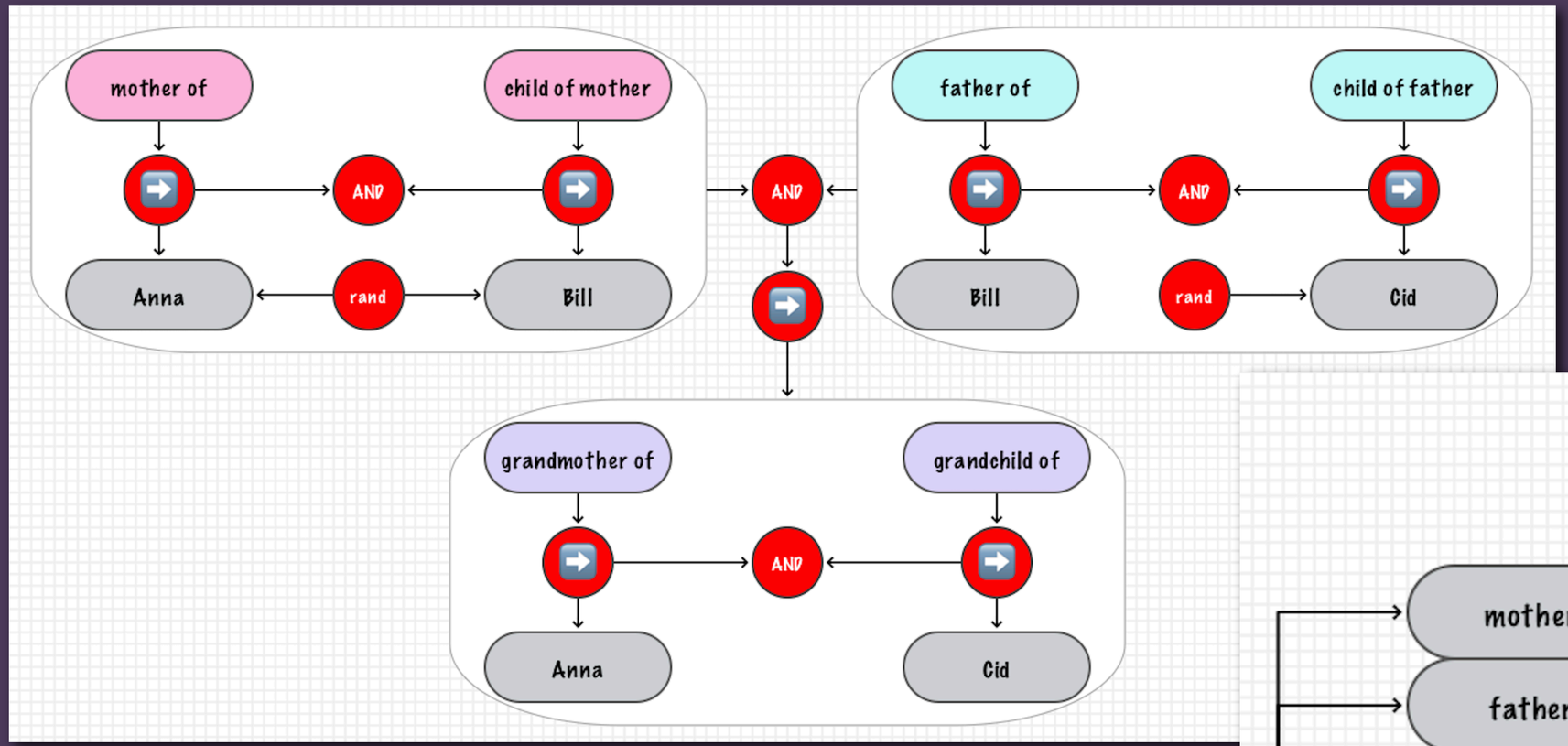
APPLY REL FOR MOTHER, FATHER, GRANDMOTHER

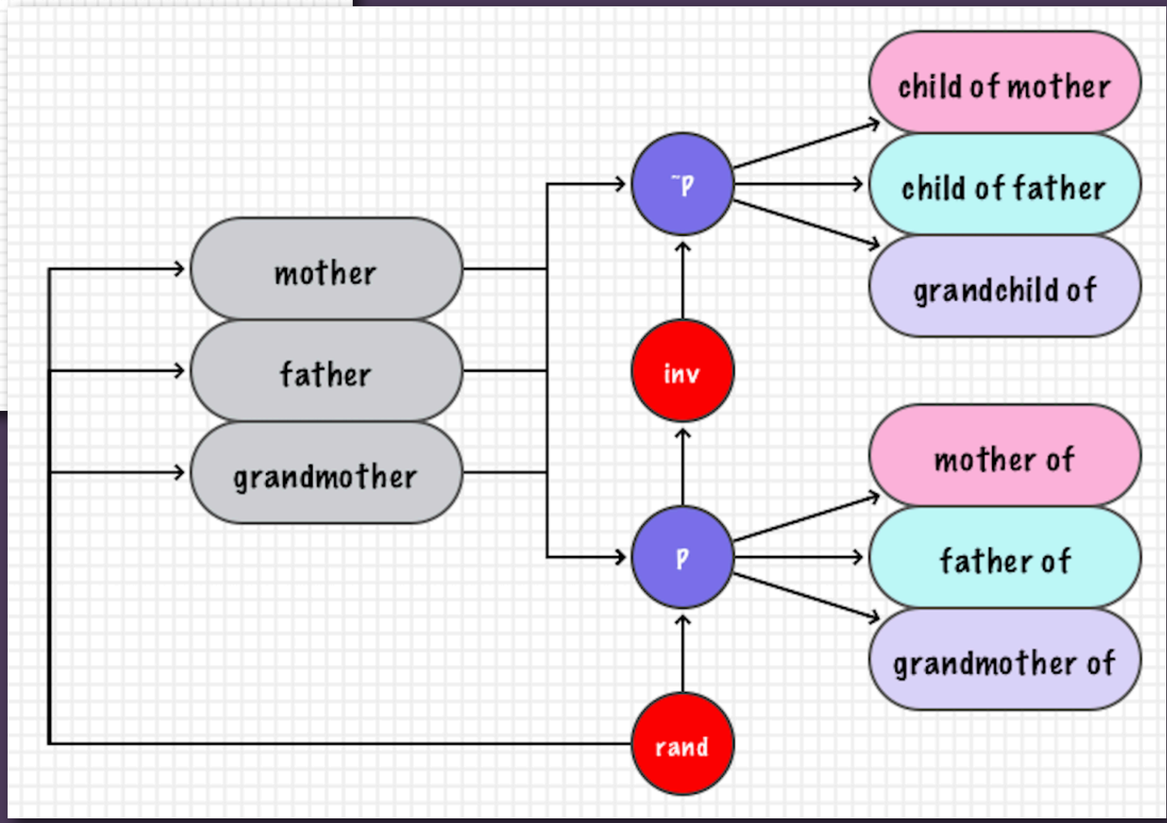
```
Let P be a random permutation of bit-vectors.

With "=>" as hmul and "/\" as hsum:

Let P(g) assert that g is unidirectional relation such that
(P(g) => x) asserts that x is a g of something.

Let P^{-1}(g) assert the other direction of g, such that
(P^{-1}(g) => y) asserts that something is a g of y.

Let (P(g) => x) /\ (P^{-1}(g) => y) assert that x is a g of y.

Implementation:

Let P(g) and P^{-1}(g) be computed externally to the call of
this routine. Call them rel_subj and rel_obj, here.
```

BASE CASE — SUPERVISORY EXAMPLES

```
Let Anna, Bill, Cid be random bit-vectors representing
particular persons.

Let m, f, g be random bit-vectors representing relations
(mother, father, and grandmother).

### Supervisory Examples

Let G = (P(g) => Anna) /\ (P^{-1}(g) => Cid)   (direct).

Let M = (P(m) => Anna) /\ (P^{-1}(m) => Bill).
Let F = (P(f) => Bill) /\ (P^{-1}(f) => Cid).
Let H = (M /\ F)                                (transitivity).

> two expressions of the same fact, but they're not yet
  linked. Don't expect H and G to be related, yet.
```

SOME PEOPLE PREFER DIAGRAMS





TRAINING SET AND FINAL TEST

```
Let Xi, Yi, Zi, be random-bit-vector training examples.

Let Mi = (P(m) => Xi) /\ (P^{-1}(m) => Yi).
Let Fi = (P(f) => Yi) /\ (P^{-1}(f) => Zi).
Let Gi = (P(g) => Xi) /\ (P^{-1}(g) => Zi)      (Gi != G)

> Sum a collection of such transitivity examples:

Let T = hsum( [(Mi /\ Fi) => Gi for i in range(15)] )

### Final Test

> Collect a bit-vector representing the assertion that
  randomly trained examples imply the specific transitive case.

Let G' = (T => H)

Check that G' obtains statistically:

G ~~ G'    (related within 6 sigmas, 1 in a billion)
```

GRANDMOTHER — REGISTER ALLOCATION BY HAND

15 TRAINING EXAMPLES IN SECTIONS [0..E] PLUS FINAL TEST IN F

	VR 0	VR 1	VR 2	VRs 3 .. 7	VR 9
	rels	dest	persons	tiebreakers	averaging
sec 0	mother	dest	person_x	tb_1	avg_0
sec 1	father	scratch	person_y	tb_2	avg_1
sec 2	gran	scratch_2	person_z	tb_3	avg_2
sec 3	m of m	scratch_3		tb_4	avg_3
sec 4	m of c	mxy		etc.	avg_4
sec 5	f of f	fyz			avg_5
sec 6	f of c	gxz			avg_6
sec 7	g of g	scratch_4			avg_7
sec 8	g of c	scratch_5			avg_8
sec 9		scratch_6			avg_9
sec A	x	scratch_7			avg_A
sec B	y	scratch_8			avg_B
sec C		scratch_9			avg_C
sec D	subj				avg_D
sec E	obj				avg_E
sec F					avg_F

example	anna	bill	cid
section	VR 10	VR 11	VR 12
0			
1			
2			
3			
4			
5			
6			
7			
8			
9			
A			
B			
C			
D			
E			
F			

GRANDMOTHER IMPROVEMENTS

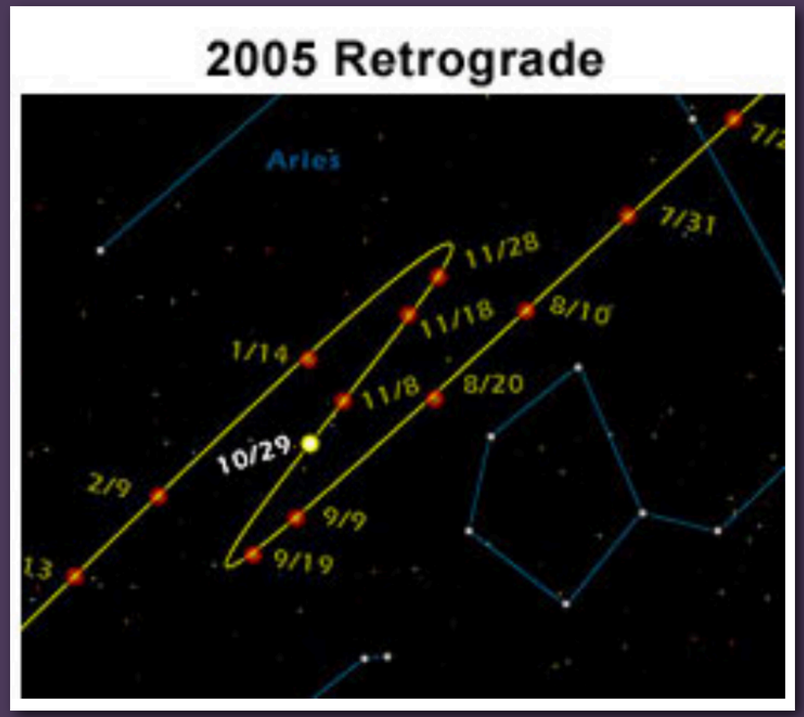
- **Constant-Section version would be much faster**
- **Automate Section Allocation**
  - We already do VR allocation and coalescing — extend the idea to Sections
- **Figure out how to distribute 16 simultaneous calculations (MPI-style?)**
- **It's I/O-bound — do more calculations per I/O**

BRIAN BECKMAN, 1 SEPTEMBER 2023

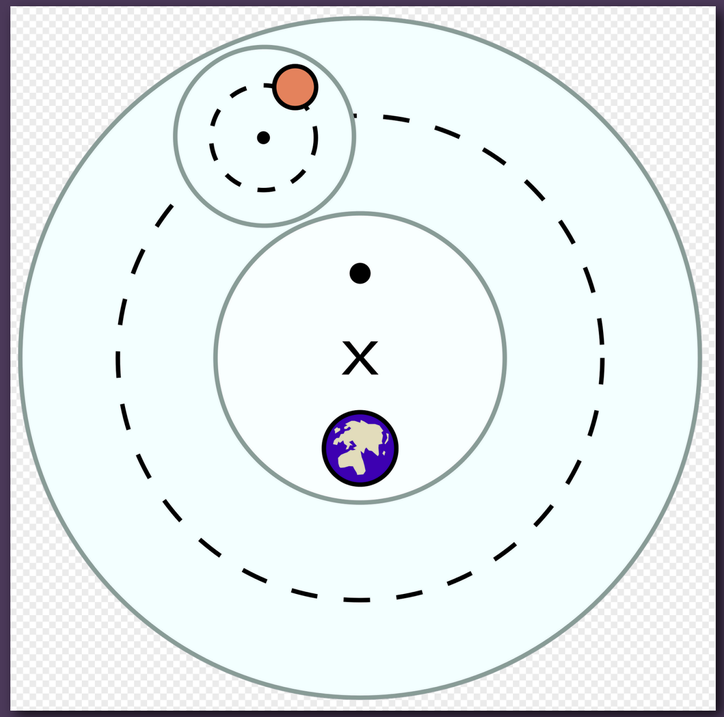
HDC: COPERNICAN REVOLUTION FOR AI?

PTOLEMAIC ASTRONOMY

- **Premise: Planets 1. go in circles 2. around the Earth**
  - because 1. God is perfect, only creates perfect circles 1. God said Earth is special.
- **Problem: the planets sometimes go backwards in the sky!**



- **Solution: Planets go in circles within circles!**
  - (And maybe not exactly around Earth's center)
  - Like a Taylor Series, can approximate any orbit



COPERNICUS, BRAHE, KEPLER, NEWTON, EINSTEIN

LET'S HAVE A BETTER MODEL

- **Ellipses around the Sun account for all the new, precise observations**
  - The model has fewer parameters!
  - Survives Newton's reduction to Gravitation
  - Survives (with a few more well-organized parameters) Einstein's corrections
- **Occam's Razor says smaller, better organized, more predictive is better**
  - Doesn't say Ptolemy is *wrong*!
  - Taylor Series aren't *wrong*, they're just inefficient when alternatives exist

PTOLEMAIC FUNCTION APPROXIMATORS

HTTPS://WWW.YOUTUBE.COM/WATCH?V=TKWXA7CVFR8

- Neural networks are Universal Function Approximators
- They model everything as layers of floating-point matrices — one hammer for all nails
- Profligate use of metaphorical epicycles (memory is cheap, free data are everywhere)
- But, if we know more about a class of functions, we can devise (much) more efficient approximations

COPERNICUS, BRAHE, KEPLER, NEWTON, EINSTEIN

LET'S HAVE A ~~SMALLER~~ MORE STRUCTURED MODEL

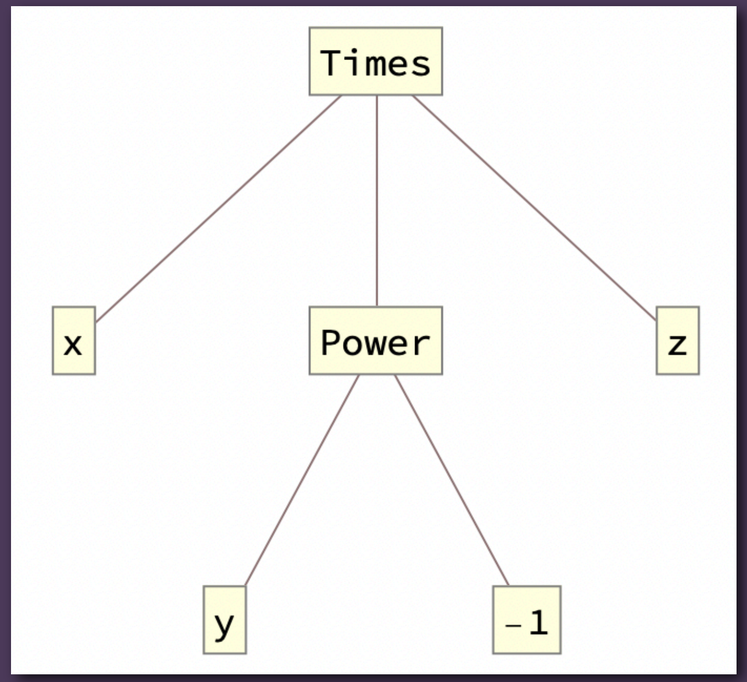
- **Knowing more structure means you don't need to know more data.**
  - Einstein's model predicts all details of planetary motion, plus much more!
  - It's survived every fair attempt to refute it.
  - It's smaller, better organized, and more predictive.
  - But, it's not Universal — it can't explain Quantum phenomena.
- Can HDC be to Neural Networks as CBKNE is to Ptolemy?
  - For certain problems, I think so, and grandly so, orders of magnitude more efficient.
  - Is HDC Universal, in the sense "can it solve all current AI problems" ?

DIGRESSION: WHY DIDN'T I SAY "SIMPLER"

"SIMPLE" ISN'T MEASURED THE SAME WAY BY EVERYONE

- Which of the following two is "simpler?

$$x/yz$$



DIGRESSION: WHY DIDN'T I SAY "SIMPLER"

IT'S NOT SO EASY TO SAY

- Some say this is "simpler"
  BUT, it is ambiguous

$$x/yz$$

- Programming languages

$$xz/y$$

- American Mathematical Society et al.

$$x/(yz)$$

- Some say this is "simpler"
  BUT, it is bigger