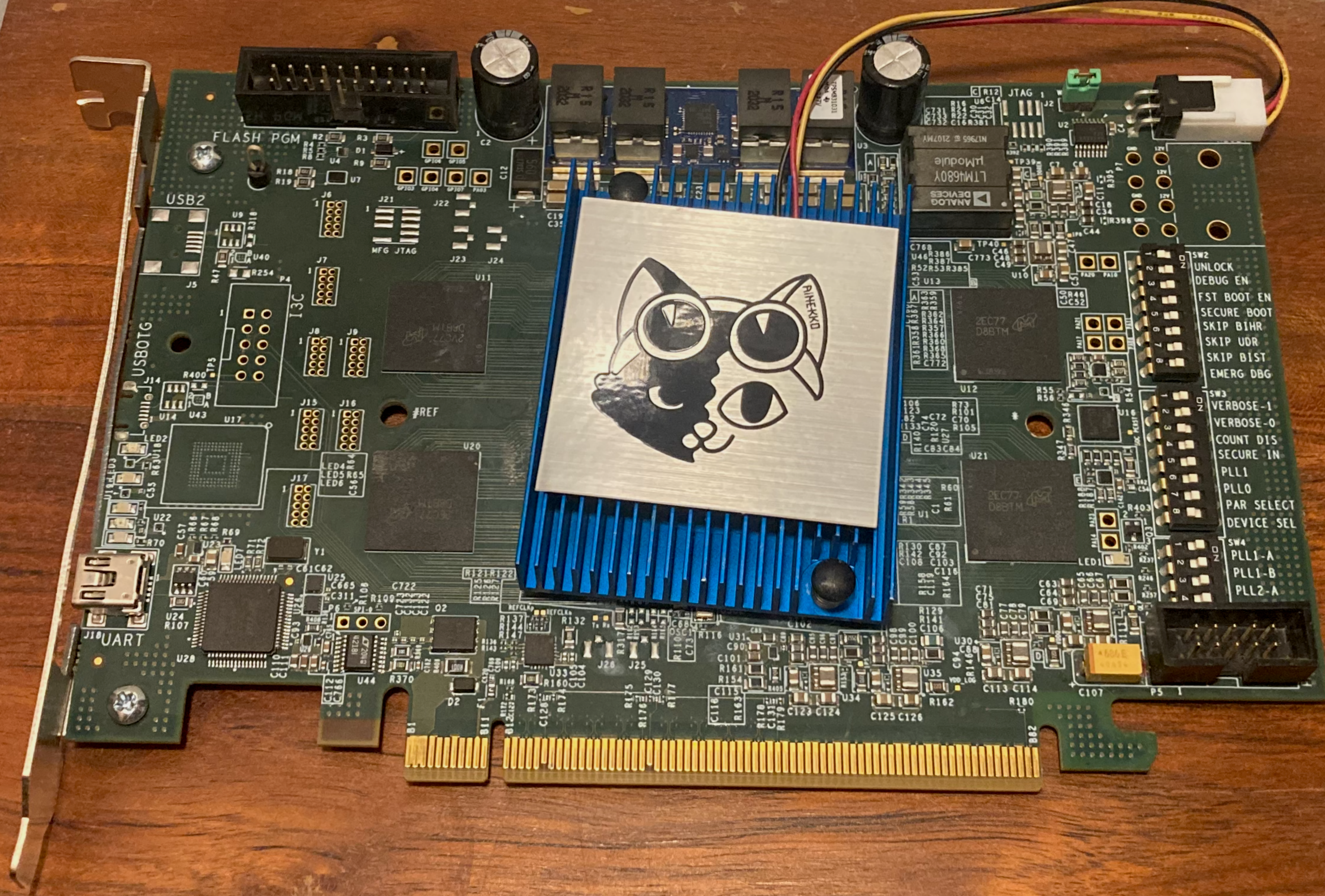


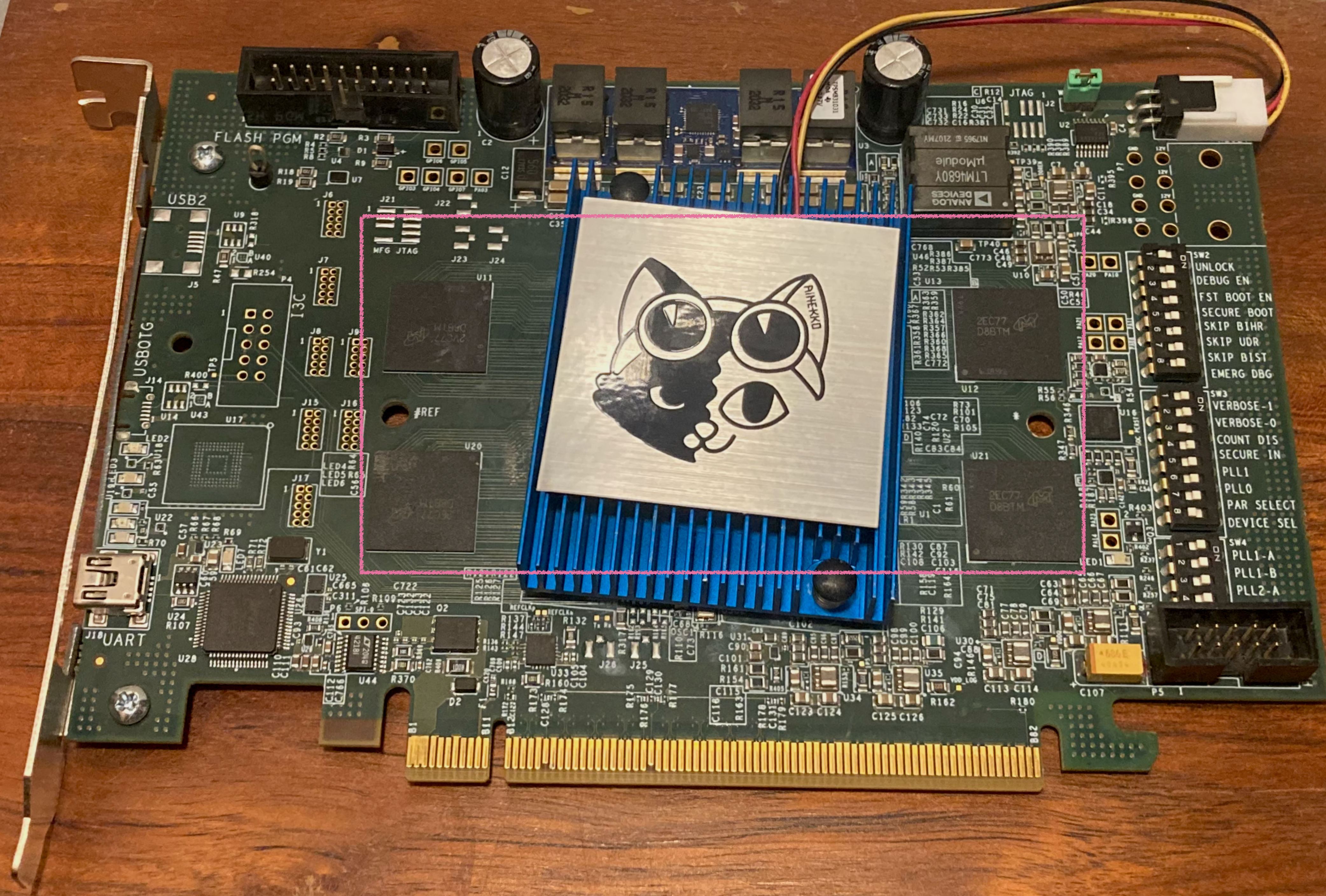
Zero to matmul with the ET-SoC-1

FOSDEM 2026, AI Plumbers devroom

Zero to matmul with the ET-SoC-1

FOSDEM 2026, AI Plumbers devroom







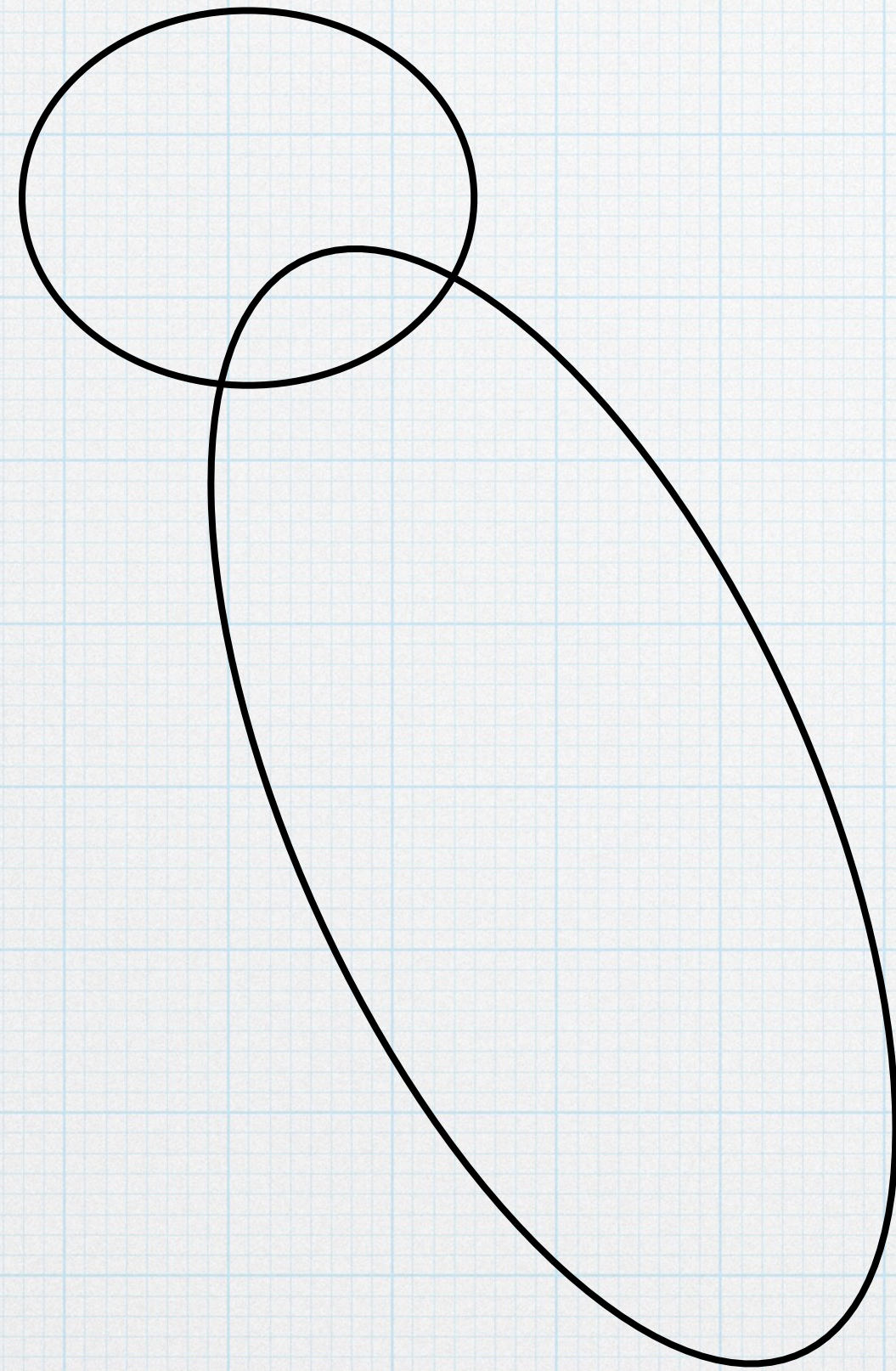
- ▶ **1093 RISC-V cores (RV64IMFC plus extensions)**
- ▶ **Minus 5 special cores (4 maxions, 1 service processor)**
- ▶ **Minus 32 cores for consistent yield purposes (*)**
- ▶ **Minus 32 cores for default firmware (*)**
- ▶ **Leaves us with 1024 “minion” cores to play with**

- ▶ **Software stack? On GitHub**
- ▶ **Firmware? On GitHub**
- ▶ **Manuals? On GitHub**
- ▶ **Emulator? On GitHub**
- ▶ **RTL? Hopefully soon on GitHub**

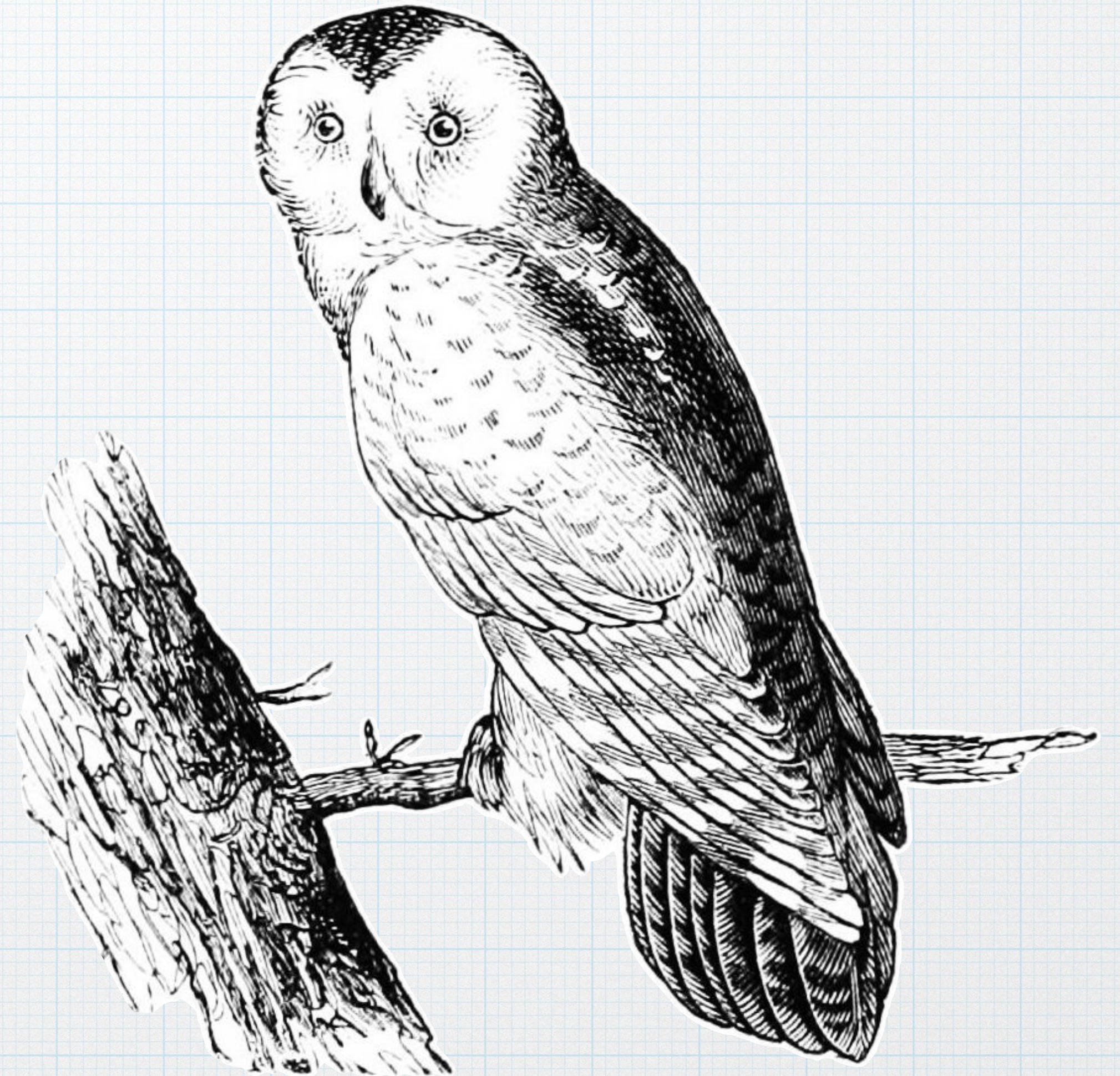


- ▶ **1024 “minion” cores**
- ▶ **At 650 MHz (*)**
- ▶ **With 8 vector lanes per core (custom SIMD, not RVV)**
- ▶ **And every FMA instruction is two fp32 operations**
- ▶ **... so 10.6 TFLOP/s of fp32 in theory?**

How to draw an owl

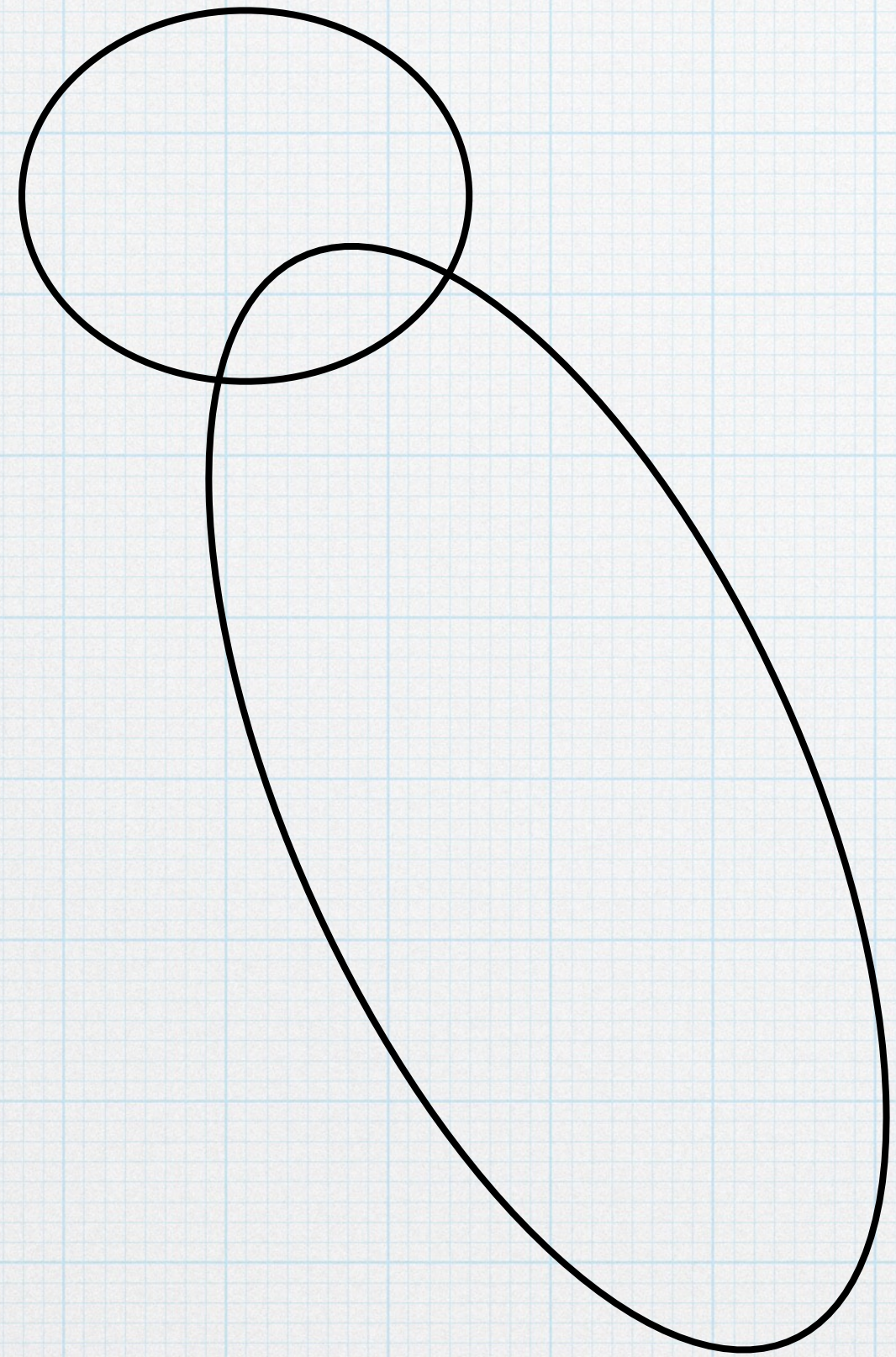


1. Draw some circles

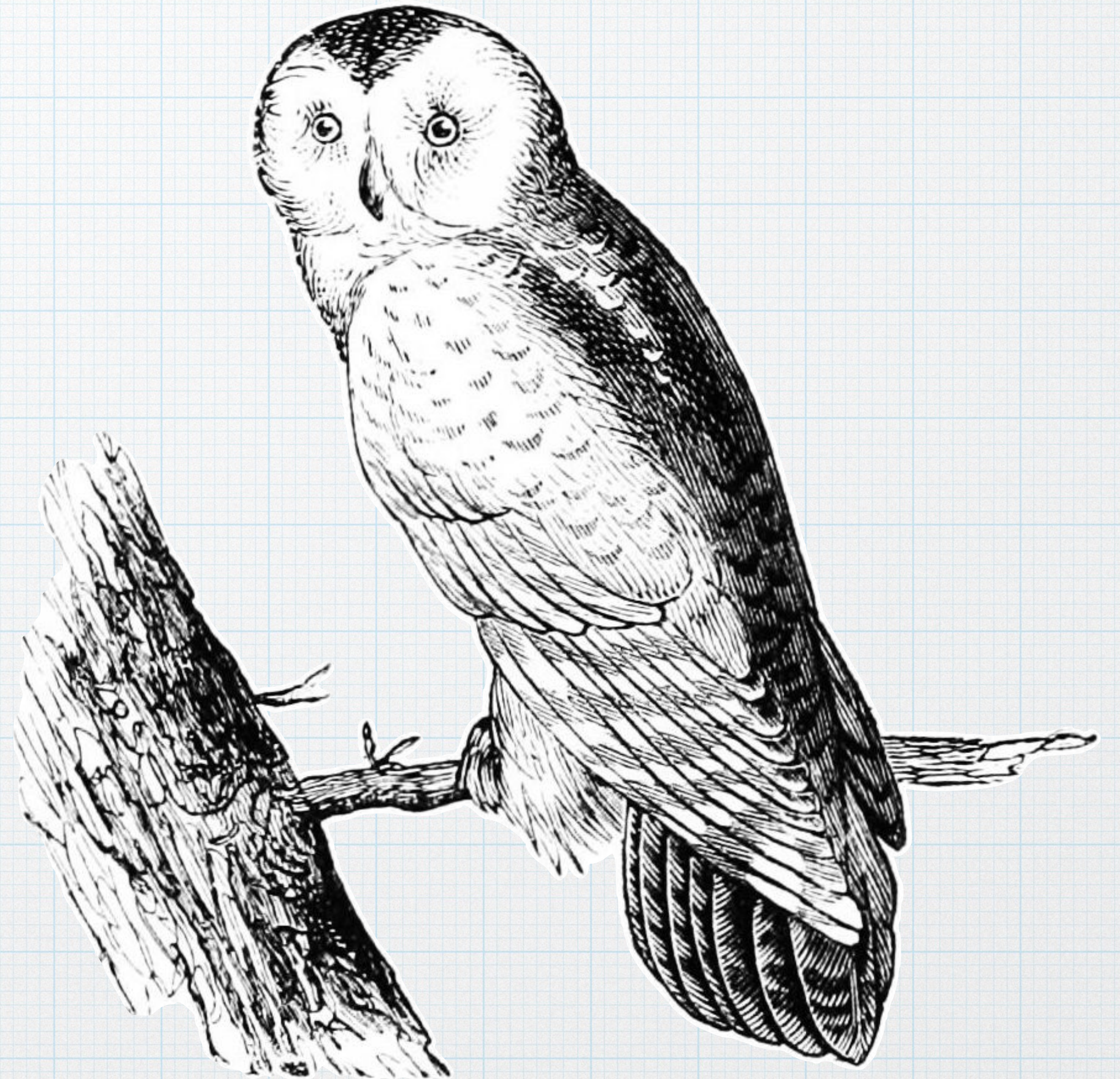


2. Draw the rest of the owl

How to draw an owl in the age of AI

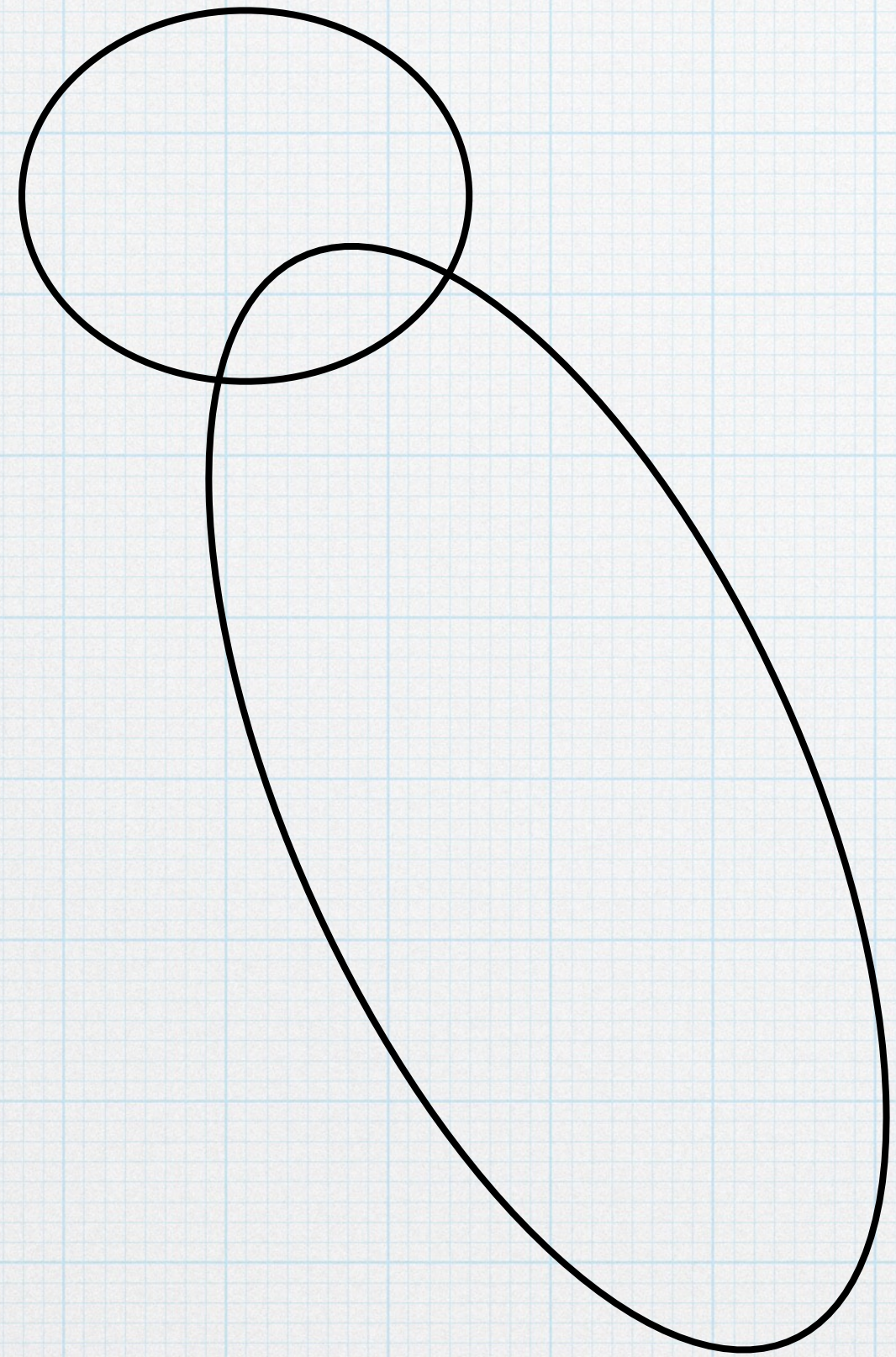


1. Draw some circles

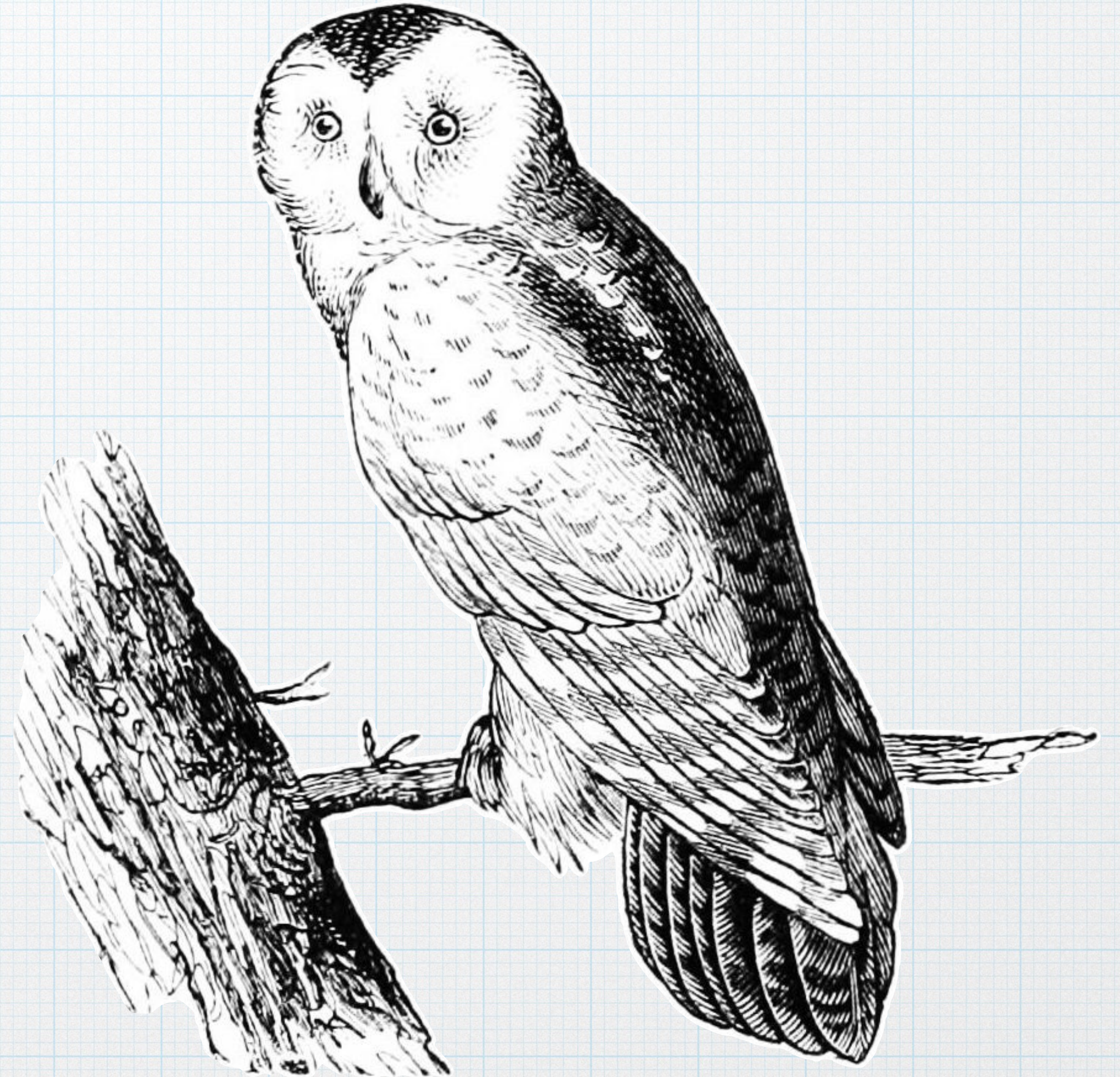


2. Use img2img diffusion model

How to create a tensor library (PyTorch, GGML, tinygrad, etc)



1. Figure out matmuls



2. Figure out the rest of the library


```
struct matrix { fp32 x[512][512]; };
```

```
void matmul(matrix* a, matrix* b, matrix* c) {  
    for (int i = 0; i < 512; ++i) {  
        for (int j = 0; j < 512; ++j) {  
            fp32 x = 0;  
            for (int k = 0; k < 512; ++k) {  
                x += a->x[i][k] * b->x[k][j];  
            }  
            c->x[i][j] = x;  
        }  
    }  
}
```


7 MFLOP/s

... only 1 ½ million times slower than my 10.6 TFLOP/s goal

Act 1: Going Parallel

mhartid is threadIdx
and blockIdx
(mostly)


```

void matmul(matrix* a, matrix* b, matrix* c) {
    int i = csr_read(mhartid);
    if (i < 512) {
        for (int j = 0; j < 512; ++j) {
            fp32 x = 0;
            for (int k = 0; k < 512; ++k) {
                x += a->x[i][k] * b->x[k][j];
            }
            c->x[i][j] = x;
        }
    }
}

```

```

void matmul(matrix* a, matrix* b, matrix* c) {
    int j = csr_read(mhartid);
    for (int i = 0; i < 512; ++i) {
        if (j < 512) {
            fp32 x = 0;
            for (int k = 0; k < 512; ++k) {
                x += a->x[i][k] * b->x[k][j];
            }
            c->x[i][j] = x;
        }
    }
}

```



```

int i = csr_read(mhartid);
if (i < 512) {
    for (int j = 0; j < 512; ++j) {
        fp32 x = 0;
        for (int k = 0; k < 512; ++k) {
            x += a->x[i][k] * b->x[k][j];
        }
        c->x[i][j] = x;
    }
}

```

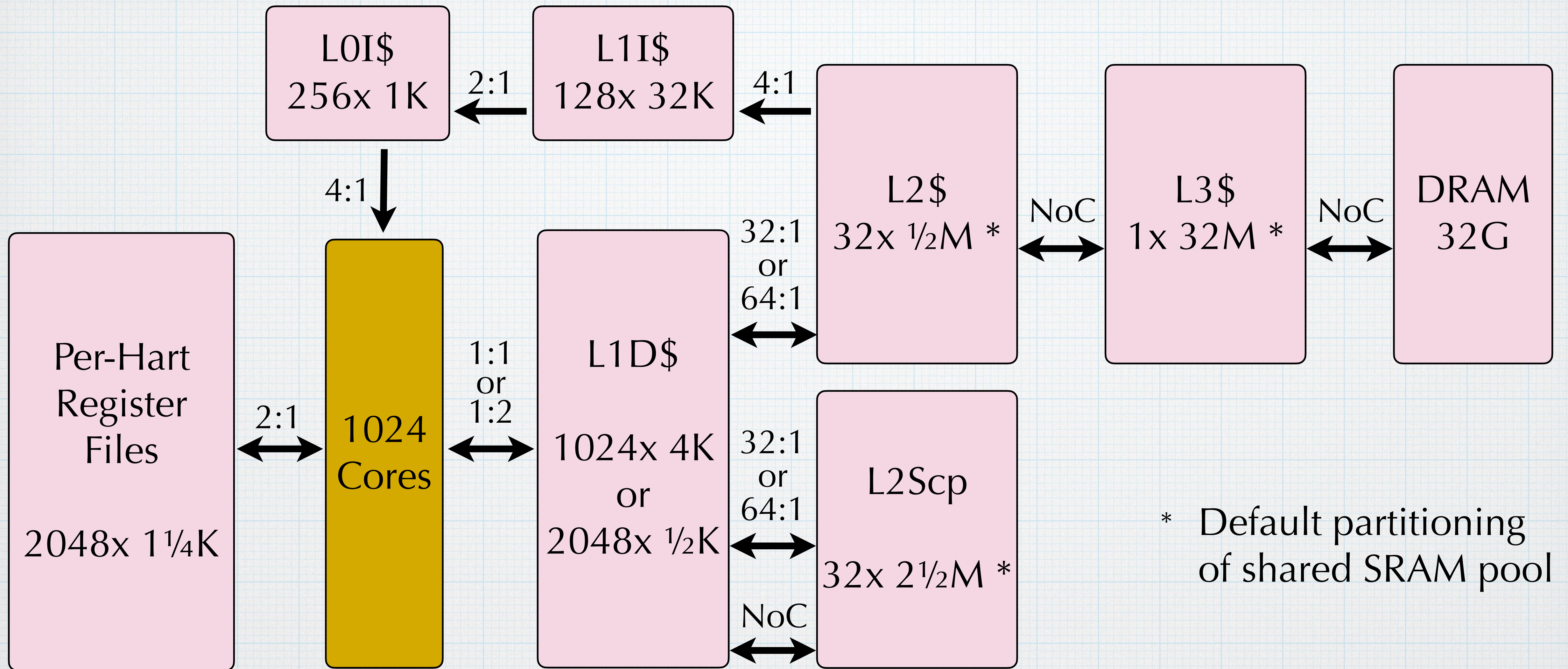
Correct result

```

int j = csr_read(mhartid);
for (int i = 0; i < 512; ++i) {
    if (j < 512) {
        fp32 x = 0;
        for (int k = 0; k < 512; ++k) {
            x += a->x[i][k] * b->x[k][j];
        }
        c->x[i][j] = x;
    }
}

```

Wrong result
Eh?



Up to 2465 caches, and no coherency between them



In hart #3's L1D\$:

? ₀	? ₁	? ₂	c[i][3]	? ₄	? ₅	? ₆	? ₇	? ₈	? ₉	? ₁₀	? ₁₁	? ₁₂	? ₁₃	? ₁₄	? ₁₅
----------------	----------------	----------------	---------	----------------	----------------	----------------	----------------	----------------	----------------	-----------------	-----------------	-----------------	-----------------	-----------------	-----------------

In hart #4's L1D\$:

? ₁₆	? ₁₇	? ₁₈	? ₁₉	c[i][4]	? ₂₁	? ₂₂	? ₂₃	? ₂₄	? ₂₅	? ₂₆	? ₂₇	? ₂₈	? ₂₉	? ₃₀	? ₃₁
-----------------	-----------------	-----------------	-----------------	---------	-----------------	-----------------	-----------------	-----------------	-----------------	-----------------	-----------------	-----------------	-----------------	-----------------	-----------------

What we eventually want in their upstream L2\$:

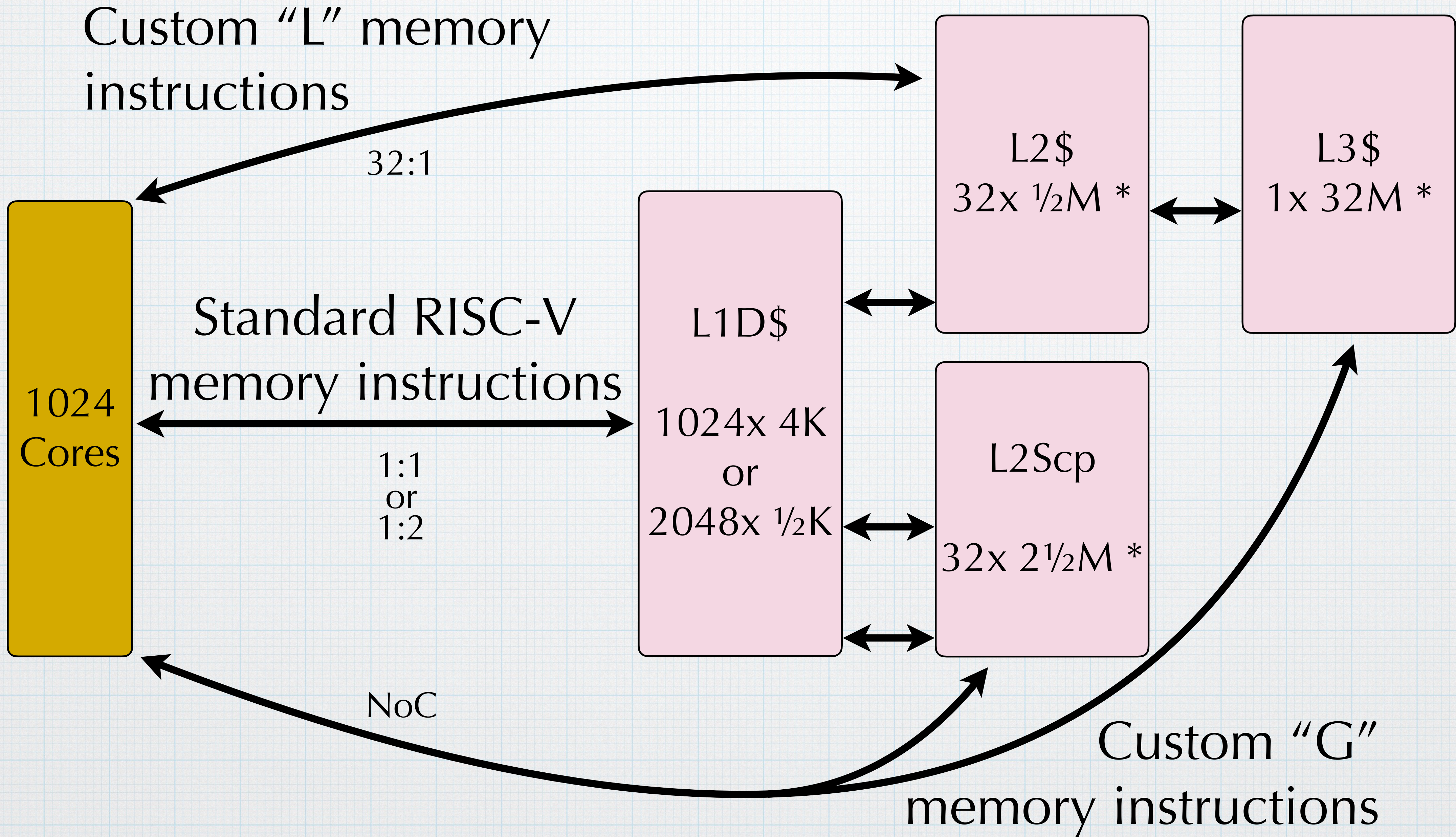
? ₃₂	? ₃₃	? ₃₄	c[i][3]	c[i][4]	? ₃₇	? ₃₈	? ₃₉	? ₄₀	? ₄₁	? ₄₂	? ₄₃	? ₄₄	? ₄₅	? ₄₆	? ₄₇
-----------------	-----------------	-----------------	---------	---------	-----------------	-----------------	-----------------	-----------------	-----------------	-----------------	-----------------	-----------------	-----------------	-----------------	-----------------

But if hart #3 flushes first, L2\$ ends up with:

? ₀	? ₁	? ₂	c[i][3]	? ₄	? ₅	? ₆	? ₇	? ₈	? ₉	? ₁₀	? ₁₁	? ₁₂	? ₁₃	? ₁₄	? ₁₅
----------------	----------------	----------------	---------	----------------	----------------	----------------	----------------	----------------	----------------	-----------------	-----------------	-----------------	-----------------	-----------------	-----------------

Then hart #4 flushes to L2\$, totally overwriting it:

? ₁₆	? ₁₇	? ₁₈	? ₁₉	c[i][4]	? ₂₁	? ₂₂	? ₂₃	? ₂₄	? ₂₅	? ₂₆	? ₂₇	? ₂₈	? ₂₉	? ₃₀	? ₃₁
-----------------	-----------------	-----------------	-----------------	---------	-----------------	-----------------	-----------------	-----------------	-----------------	-----------------	-----------------	-----------------	-----------------	-----------------	-----------------




```

int i = csr_read(mhartid);
if (i < 512) {
    for (int j = 0; j < 512; ++j) {
        fp32 x = 0;
        for (int k = 0; k < 512; ++k) {
            x += a->x[i][k] * b->x[k][j];
        }
        c->x[i][j] = x;
    }
}

```

Correct result

```

int j = csr_read(mhartid);
for (int i = 0; i < 512; ++i) {
    if (j < 512) {
        fp32 x = 0;
        for (int k = 0; k < 512; ++k) {
            x += a->x[i][k] * b->x[k][j];
        }
        // c->x[i][j] = x;
        store_l(c->x[i][j], x);
    }
}

```

Correct result

3.4 GFLOP/s

... 3 thousand times slower than my goal


```
int id = csr_read(mhartid);  
int i = id / 4;  
int jN = 128;  
int j0 = (id % 4) * jN;  
for (int j = j0; j < j0 + jN; ++j) {  
    fp32 x = 0;  
    for (int k = 0; k < 512; ++k) {  
        x += a->x[i][k] * b->x[k][j];  
    }  
    c->x[i][j] = x;  
}
```


13.1 GFLOP/s

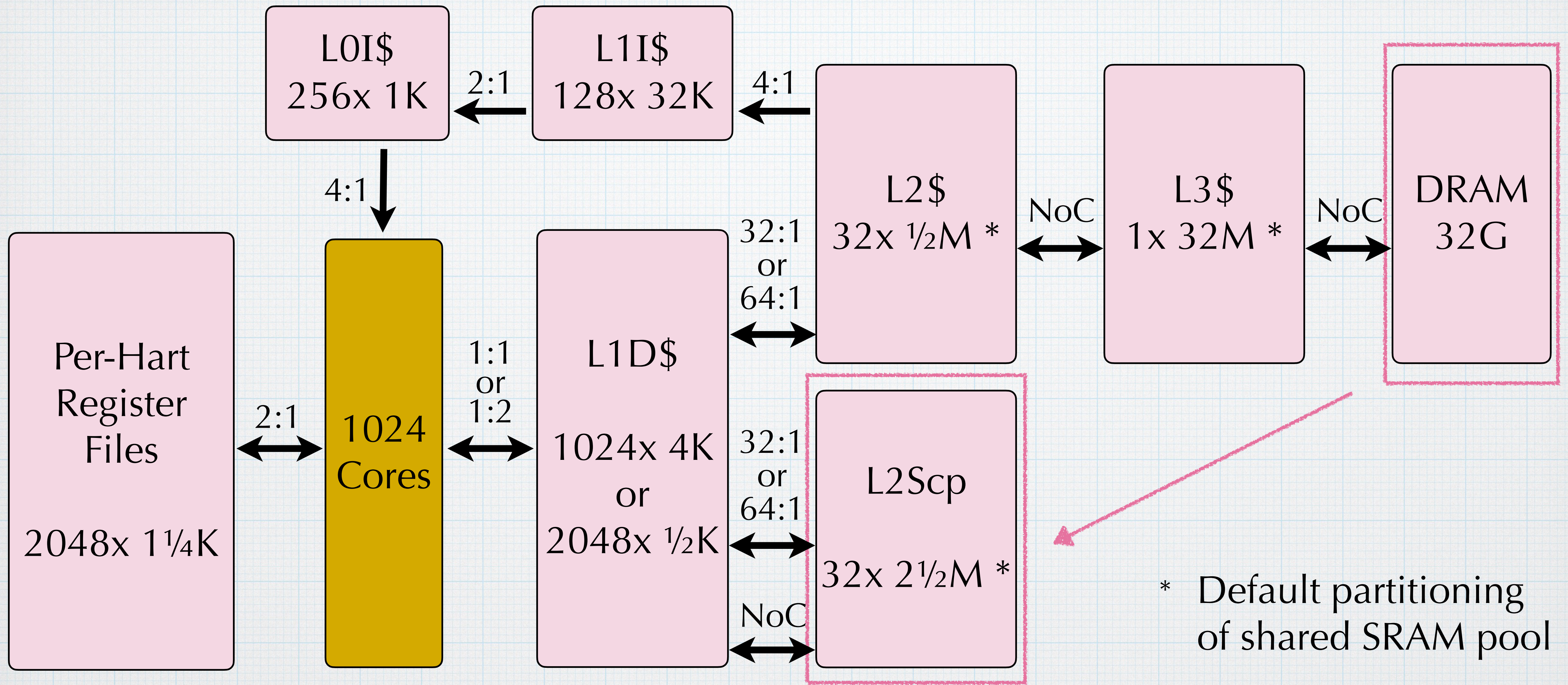
... 809 times slower than my goal


```
int id = csr_read(mhartid);
int i = id / 4;
int jN = 128;
int j0 = (id % 4) * jN;
for (int j = j0; j < j0 + jN; j += 8) {
    fp32x8 x = BCAST(0);
    for (int k = 0; k < 512; ++k) {
        x += BCAST(a->x[i][k]) * LD8(b->x[k][j]);
    }
    ST8(c->x[i][j], x);
}
```


104 GFLOP/s

... 102 times slower than my goal

Act 2: Being Clever



312 GFLOP/s

... 34 times slower than my goal


```
int id = csr_read(mhartid);
int i = id / 4;
int jN = 128;
int j0 = (id % 4) * jN;
for (int j = j0; j < j0 + jN; ++j) {
    fp32 x = 0;
    for (int k = 0; k < 512; ++k) {
        x += a->x[i][k] * b->x[k][j];
    }
    c->x[i][j] = x;
}
```



```
k_loop:
    flw      f1, 0(t1)      # Load a[i][k]
    flw      f2, 0(t2)      # Load b[k][j]
    addi     t1, t1, 4       # ++k on a ptr
    fsgnjx.s f3, f1, f2
    add      t2, t2, t3      # ++k on b ptr
    fmadd.s  f0, f1, f2, f0  # += and *
    bne      t1, t0, k_loop  # k < 512?
```


k_loop:

flw	f1, 0(t1)	# Load a[i][k]
flw	f2, 0(t2)	# Load b[k][j]
addi	t1, t1, 4	# ++k on a ptr
fsgnjx.s	f3, f1, f2	Eh?
add	t2, t2, t3	# ++k on b ptr
fmadd.s	f0, f1, f2, f0	# += and *
bne	t1, t0, k_loop	# k < 512?


```
int id = csr_read(mhartid);
int i = id / 4;
int jN = 128;
int j0 = (id % 4) * jN;
for (int j = j0; j < j0 + jN; j += 8) {
    fp32x8 x = BCAST(0);
    for (int k = 0; k < 512; ++k) {
        x += BCAST(a->x[i][k]) * LD8(b->x[k][j]);
    }
    ST8(c->x[i][j], x);
}
```


k_loop:

aif.fbc.ps	f1, 0(t1)	# BCAST a[i][k]
aif.flw.ps	f2, 0(t2)	# LD8 b[k][j]
addi	t1, t1, 4	# ++k on a ptr
aif.fsgnjsx.ps	f3, f1, f2	
add	t2, t2, t3	# ++k on b ptr
aif.fmadd.ps	f0, f1, f2, f0	# += and *
bne	t1, t0, k_loop	# k < 512?

► **14.3% FMAs**

► **28.5% Loads**

► **57.2% Other**


```

int id = csr_read(mhartid);
int i  = (id / 8) * 2;
int jN = 64;
int j0 = (id % 8) * jN;
for (int j = j0; j < j0 + jN; j += 16){
    fp32x8 x00 = BCAST(0);
    fp32x8 x01 = BCAST(0);
    fp32x8 x10 = BCAST(0);
    fp32x8 x11 = BCAST(0);
    for (int k = 0; k < 512; k += 2) {
        fp32x8 a00 = BCAST(a->x[i+0][k+0]);
        fp32x8 a01 = BCAST(a->x[i+0][k+1]);
        fp32x8 a10 = BCAST(a->x[i+1][k+0]);
        fp32x8 a11 = BCAST(a->x[i+1][k+1]);
        ...
    }
}

```

```

...
fp32x8 b00 = LD8(b->x[k+0][j+0]);
fp32x8 b01 = LD8(b->x[k+0][j+8]);
fp32x8 b10 = LD8(b->x[k+1][j+0]);
fp32x8 b11 = LD8(b->x[k+1][j+8]);
x00 = x00 + a00 * b00 + a01 * b10;
x01 = x01 + a00 * b01 + a01 * b11;
x10 = x10 + a10 * b00 + a11 * b10;
x11 = x11 + a10 * b01 + a11 * b11;
}
ST8(c->x[i+0][j+0], x00);
ST8(c->x[i+0][j+8], x01);
ST8(c->x[i+1][j+0], x10);
ST8(c->x[i+1][j+8], x11);

```


Was 1x8

Now 2x16

► **14.3% FMAs** ► **32% FMAs**

► **28.5% Loads** ► **32% Loads**

► **57.2% Other** ► **36% Other**

1.64 TFLOP/s

... 6½ times slower than my goal

Was 2x16

Going to 4x32

► 14.3% FMAs ► 32% FMAs ► 62.8% FMAs

► 28.5% Loads ► 32% Loads ► 31.4% Loads

► 57.2% Other ► 36% Other ► 5.8% Other

2.94 TFLOP/s

... 3.6 times slower than my goal

Now 4x32

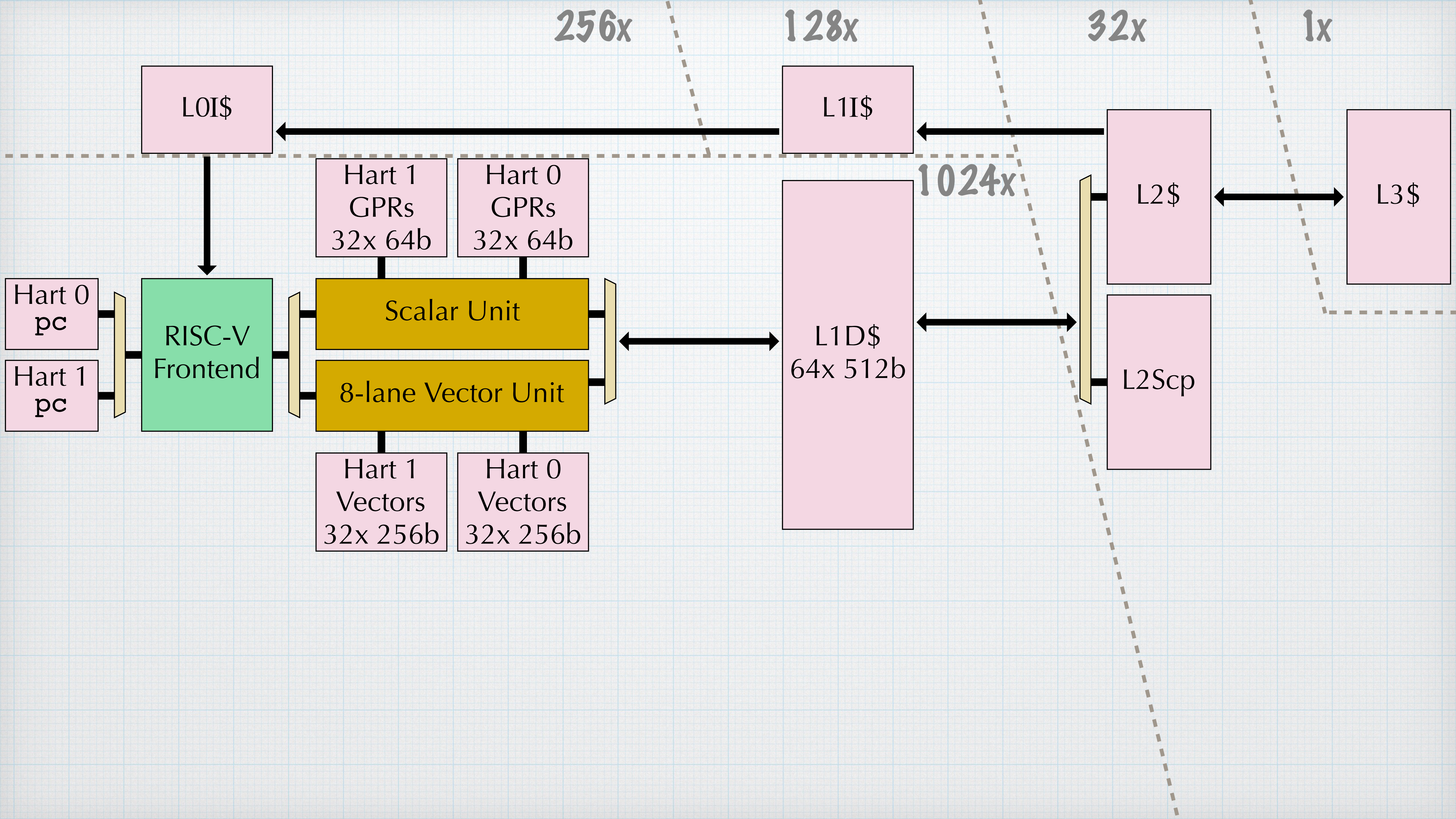
Somehow?

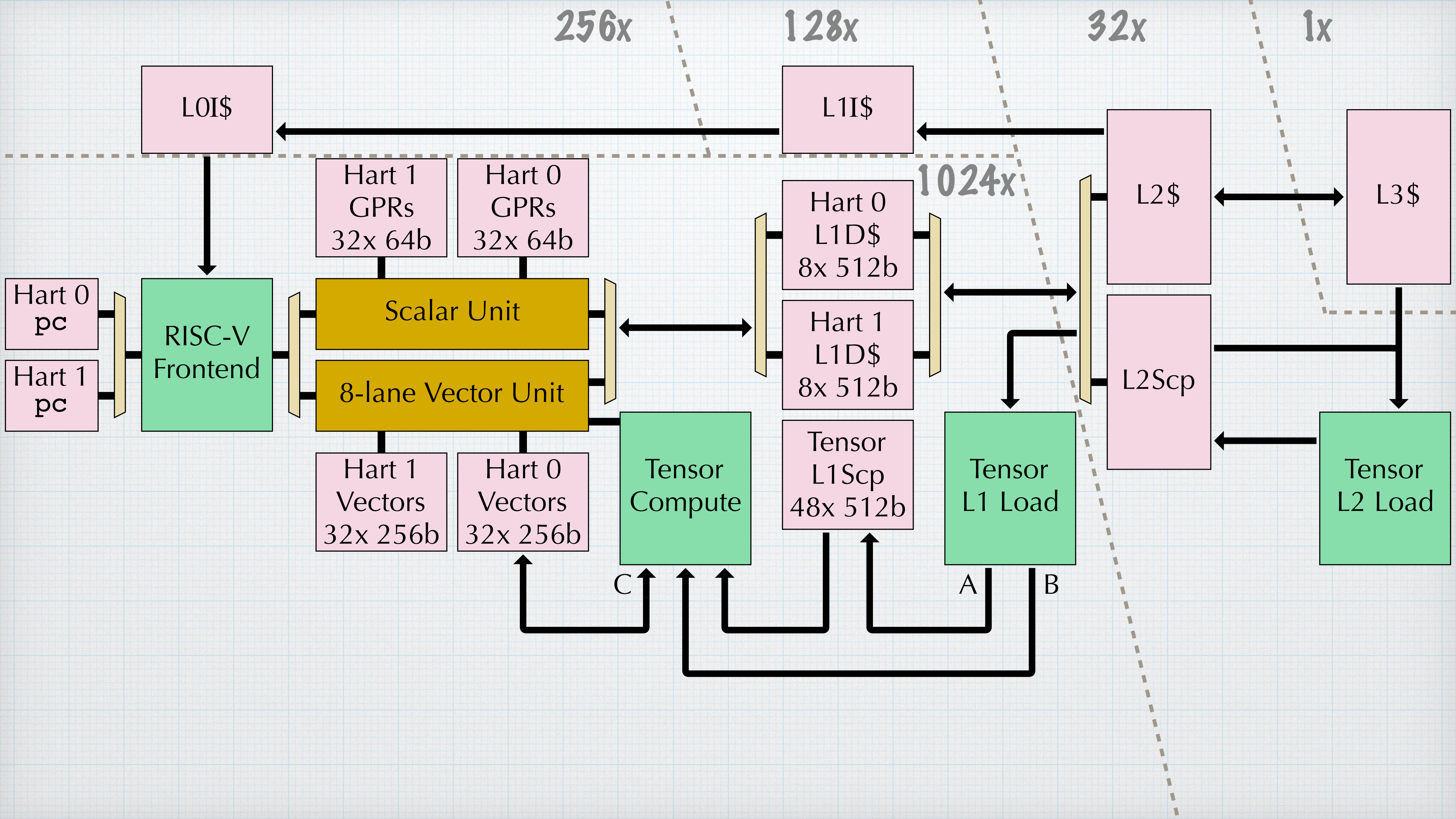
► 14.3% FMAs ► 32% FMAs ► 62.8% FMAs ► 100% FMAs

► 28.5% Loads ► 32% Loads ► 31.4% Loads ► 20% Loads

► 57.2% Other ► 36% Other ► 5.8% Other ► 5% Other

Act 3: Magic Hardware






```
int id = csr_read(mhartid);  
if (id & 1) return;  
int i = ((id / 2) / 32) * 16;  
int j = ((id / 2) % 32) * 16;  
...
```



```

...
register u64 stride __asm("x31") = offsetof(matrix, x[1][0]);
for (int k = 0; k < 512; k += 16) {
    __asm volatile(
        "csrw aif.tensor_load, %[load_a_cmd]\n"
        "csrw aif.tensor_load, %[load_b_cmd]\n"
        "csrwi aif.tensor_wait, %[wait_a_cmd]\n"
        "csrw aif.tensor_fma,  %[fma_cmd]\n"
        "csrwi aif.tensor_wait, %[wait_fma_cmd]\n"
        : "+r" (stride)
        : [load_a_cmd] "r" (
            , [load_b_cmd] "r" ((1ull << 52) | 15 | (uintptr_t)&b->x[k][j])
            , [wait_a_cmd] "K" (0)
            , [fma_cmd] "r" ((3ull << 55) | (15ull << 51) | (15ull << 47)
                | ( 1ull << 20) | (k == 0))
            , [wait_fma_cmd] "K" (7)
            : "memory", "f0", "f1", ... "f30", "f31"
        );
}
...

```


...

```
__asm volatile(  
    "csrw aif.tensor_store, %[store_cmd]\n"  
    : "+r" (stride)  
    : [store_cmd] "r" ((3ull << 55) | (15ull << 51)  
                        | (uintptr_t)&c->x[i][j])  
    : "memory", "f0", "f1", ... "f30", "f31"  
);
```


7.05 TFLOP/s

... still 50% slower than my goal

Calculate i/j

{ Tensor Load A
Tensor FMA

{ Tensor Load A
Tensor FMA

...

{ Tensor Load A
Tensor FMA

{ Tensor Load A
Tensor FMA

Tensor Store



Calculate i/j
Tensor Load A

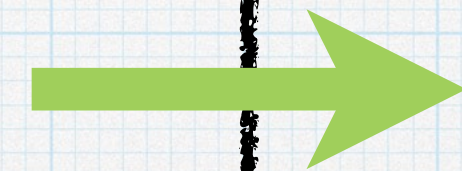
{ Tensor FMA
Tensor Load A

{ Tensor FMA
Tensor Load A

...

{ Tensor FMA
Tensor Load A

Tensor FMA
Tensor Store



Calculate i/j
Tensor Load A

Tensor FMA, Tensor Load A

Tensor FMA, Tensor Load A

...

Tensor FMA, Tensor Load A

Tensor FMA
Tensor Store


```

int id = csr_read(mhartid);
if (id & 1) return;
int i = ((id / 2) / 32) * 16;
int j = ((id / 2) % 32) * 16;
register u64 stride __asm("x31") = offsetof(matrix, x[1][0]);
int k = 0;
u64 fma_cmd = (3ull << 55) | (15ull << 51) | (15ull << 47)
               | ( 1ull << 20) | 1;

__asm volatile(
    "csrw aif.tensor_load, %[load_a_cmd]\n"
    : "+r" (stride)
    : [load_a_cmd] "r" (15 | (fma_cmd << 49) | (uintptr_t)&a->x[i][k])
    : "memory", "f0", "f1", ... "f30", "f31"
);
...

```



```

...
for (; k < 496; k += 16) {
    u64 next_fma_cmd = fma_cmd ^ 0x100;
    __asm volatile(
        "csrw aif.tensor_load, %[load_b_cmd]\n"
        "csrwi aif.tensor_wait, %[wait_a_cmd]\n"
        "csrw aif.tensor_fma,  %[fma_cmd]\n"
        "csrw aif.tensor_load, %[load_a_cmd]\n"
        : "+r" (stride)
        : [load_b_cmd] "r" ( (1ull << 52) | 15 | (uintptr_t)&b->x[k][j])
        , [wait_a_cmd] "K" (0)
        , [fma_cmd]     "r" (fma_cmd)
        , [load_a_cmd] "r" ((next_fma_cmd << 49) | 15 | (uintptr_t)&a->x[i][k+16])
        : "memory", "f0", "f1", ... "f30", "f31"
    );
    fma_cmd = next_fma_cmd & ~1ull;
}
...

```



```
...
__asm volatile(
    "csrw aif.tensor_load, %[load_b_cmd]\n"
    "csrwi aif.tensor_wait, %[wait_a_cmd]\n"
    "csrw aif.tensor_fma, %[fma_cmd]\n"
    "csrw aif.tensor_store, %[store_cmd]\n"
    : "+r" (stride)
    : [load_b_cmd] "r" ((1ull << 52) | 15 | (uintptr_t)&b->x[k][j])
    , [wait_a_cmd] "K" (0)
    , [fma_cmd] "r" (fma_cmd)
    , [store_cmd] "r" ((3ull << 55) | (15ull << 51) | (uintptr_t)&c->x[i][j])
    : "memory", "f0", "f1", ... "f30", "f31"
);
```


10.25 TFLOP/s

... sufficiently close to my goal