Programming the Cell Processor

A simple raytracer from pseudo-code to spu-code

Michael Ellerman

Linux on Cell/Power Kernel Team
IBM OzLabs
Australia Development Lab

Overview

Cell Processor

Raytracing

Optimisation strategies

Bling

Summary

The Cell Processor

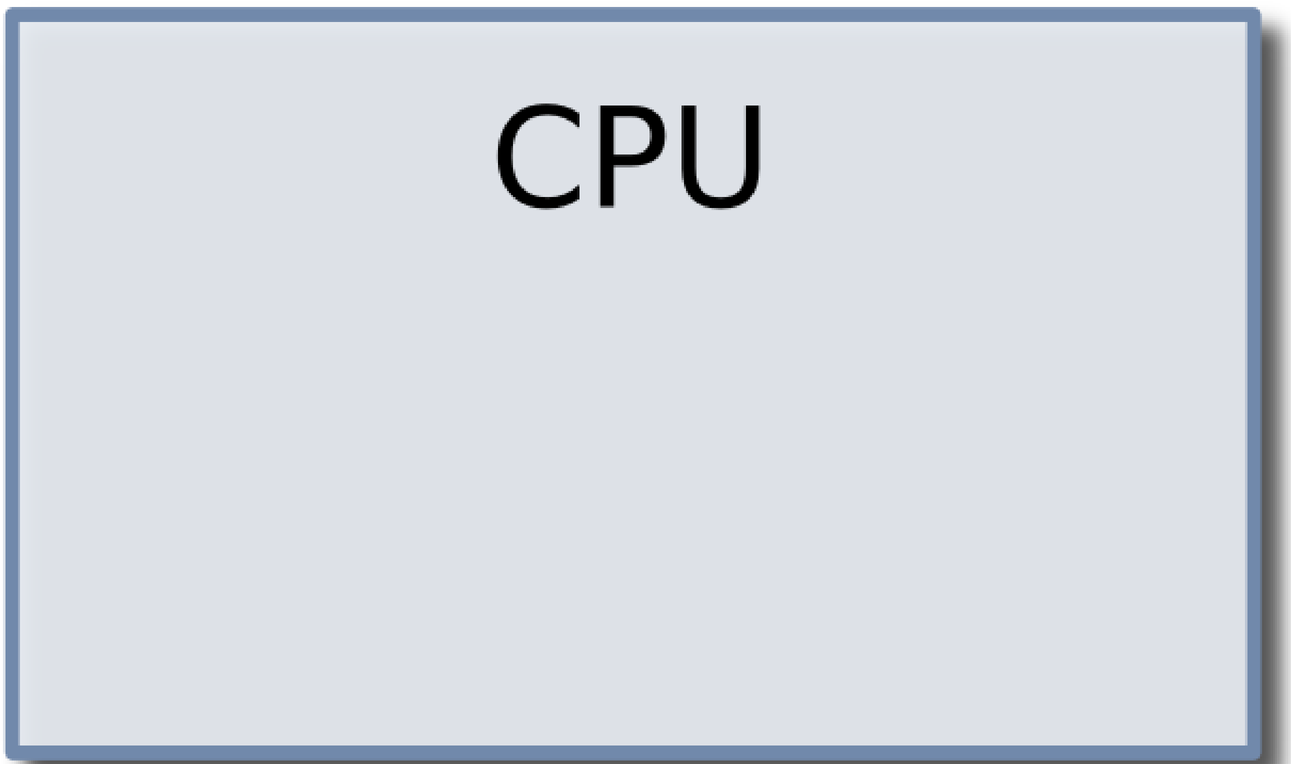
The Cell Broadband Engine® Processor

An implementation of the Cell Broadband Engine® Architecture

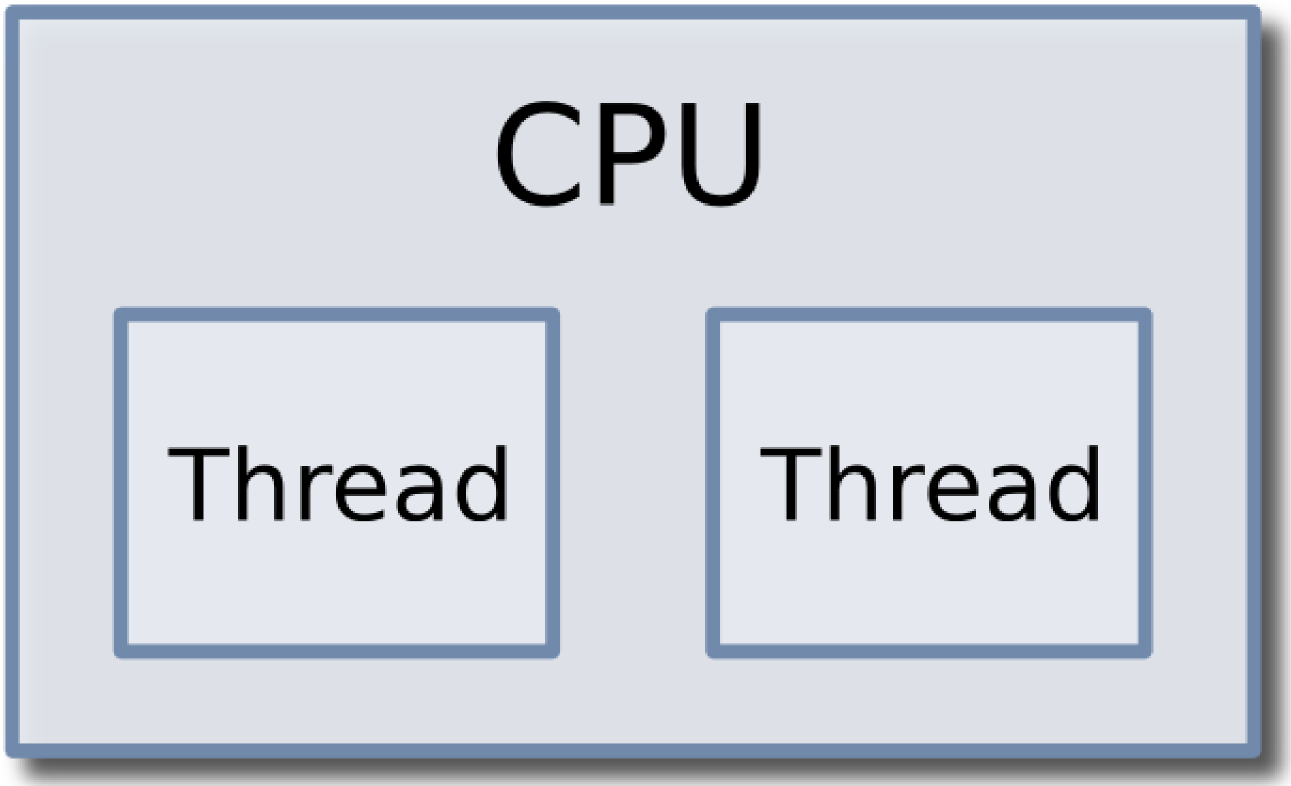
Cell Broadband Engine is a trademark of Sony Computer Entertainment Inc.

Why is Cell interesting?

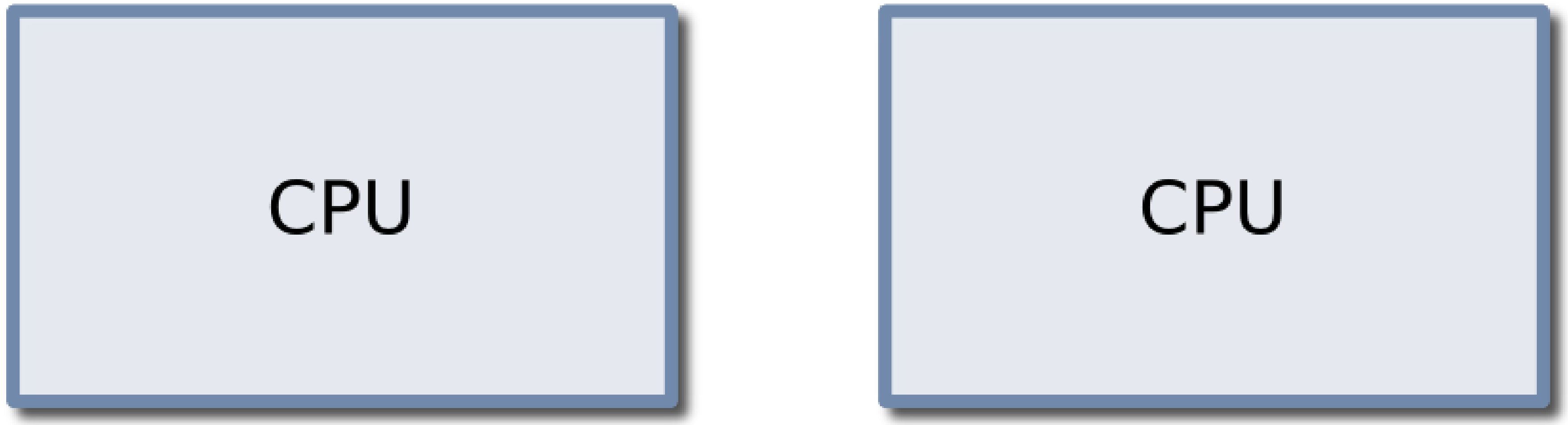
The good old days



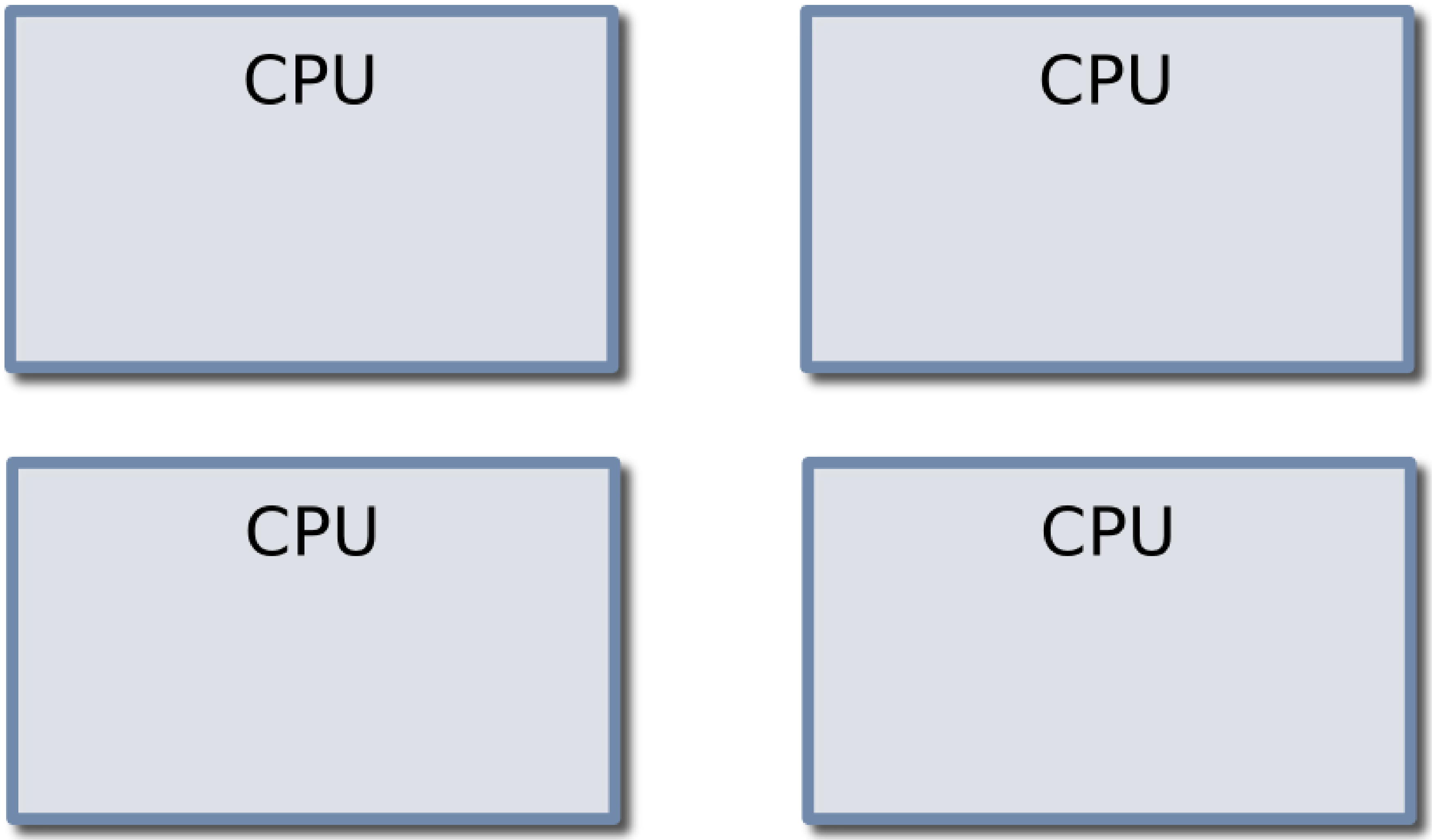
Early 2000's



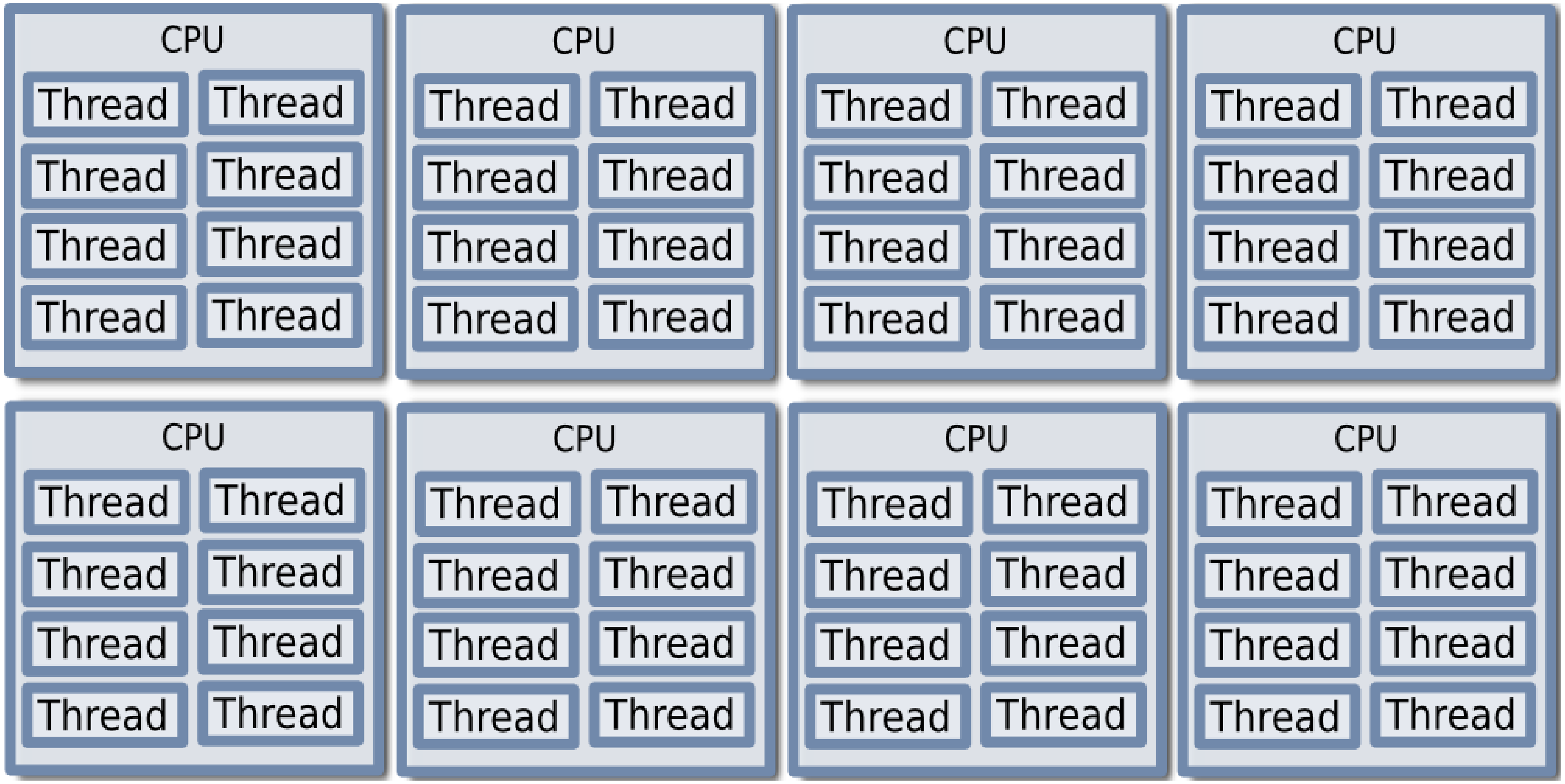
About now



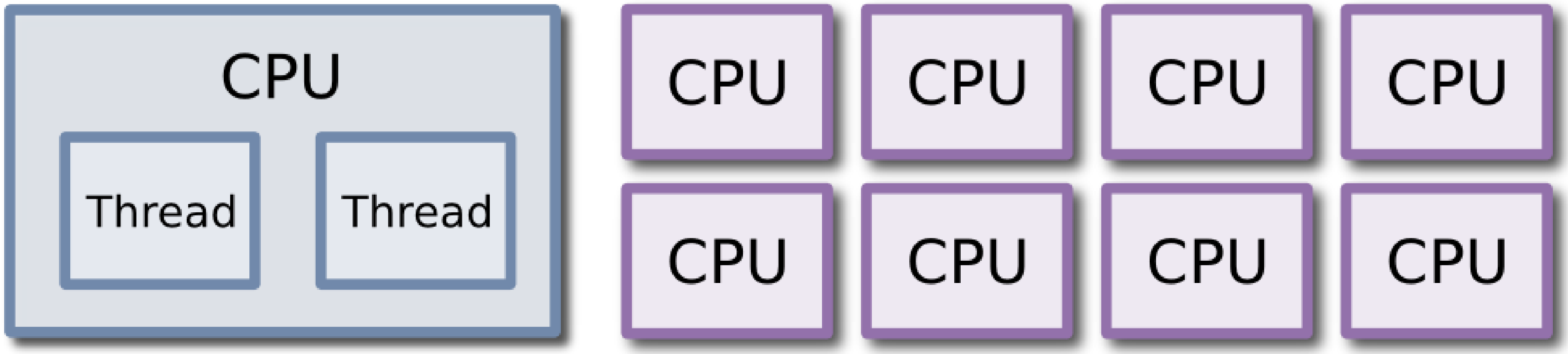
Soonish



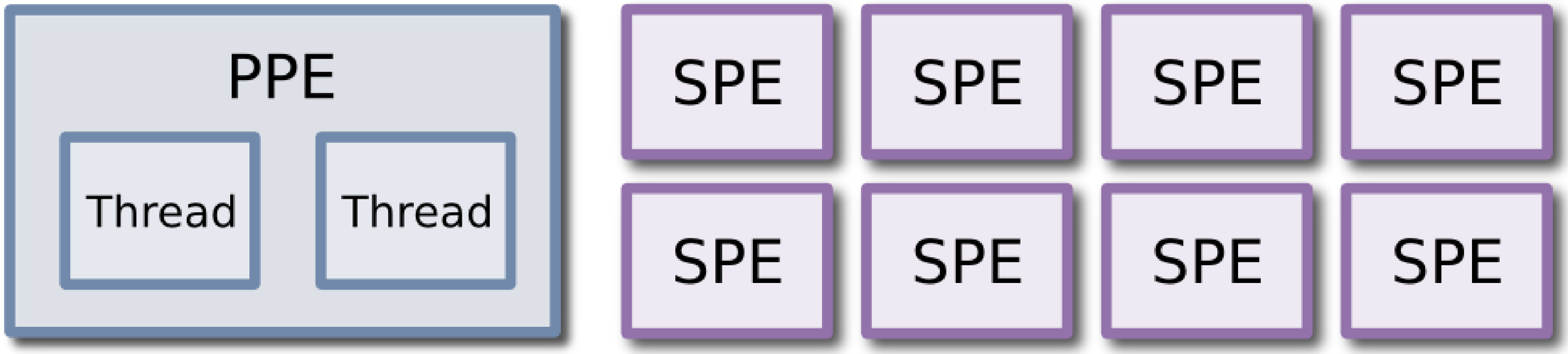
2015: In your laptop?



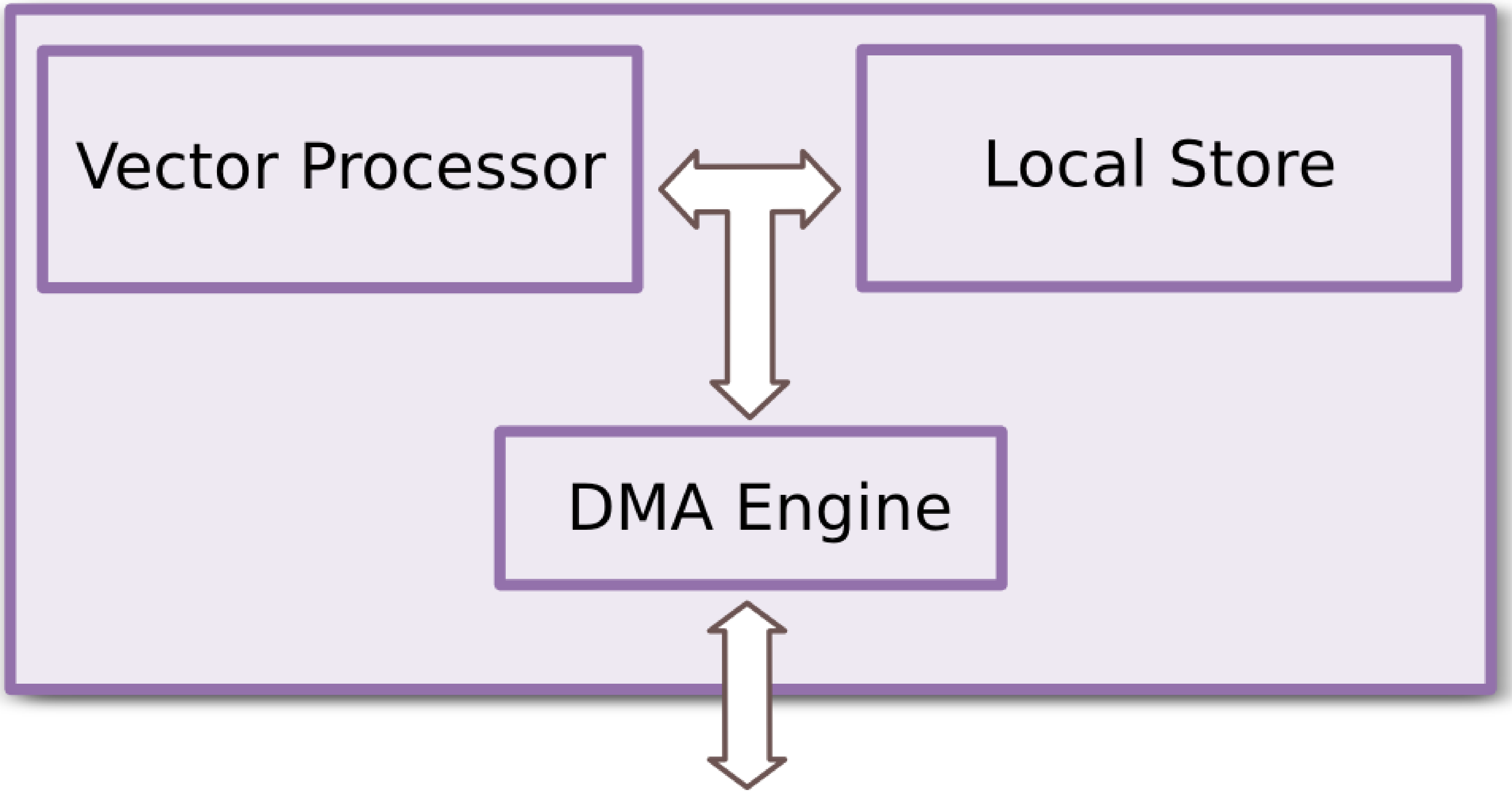
The Cell



The Cell



SPEs are more than CPUs

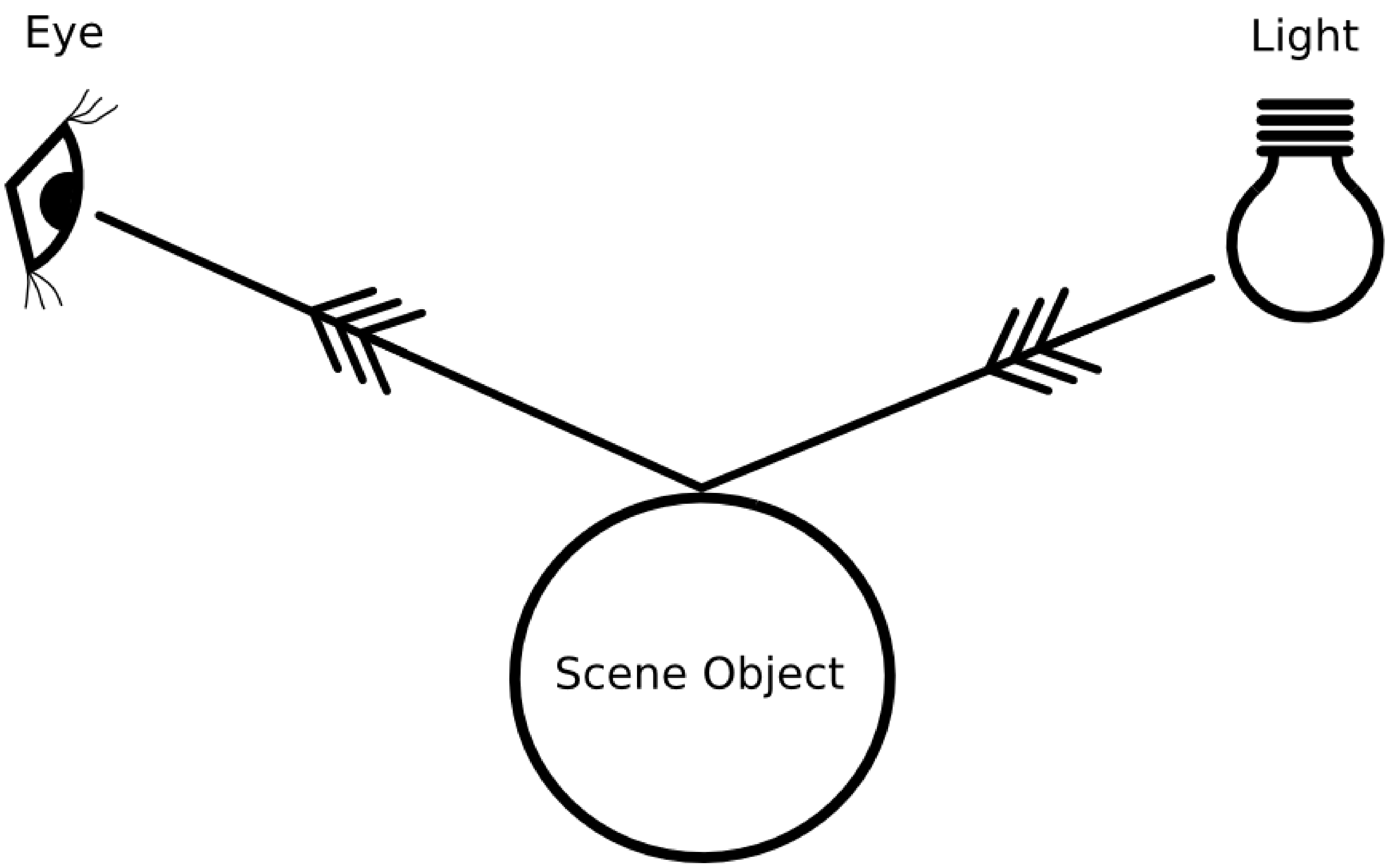


There will be no CPU

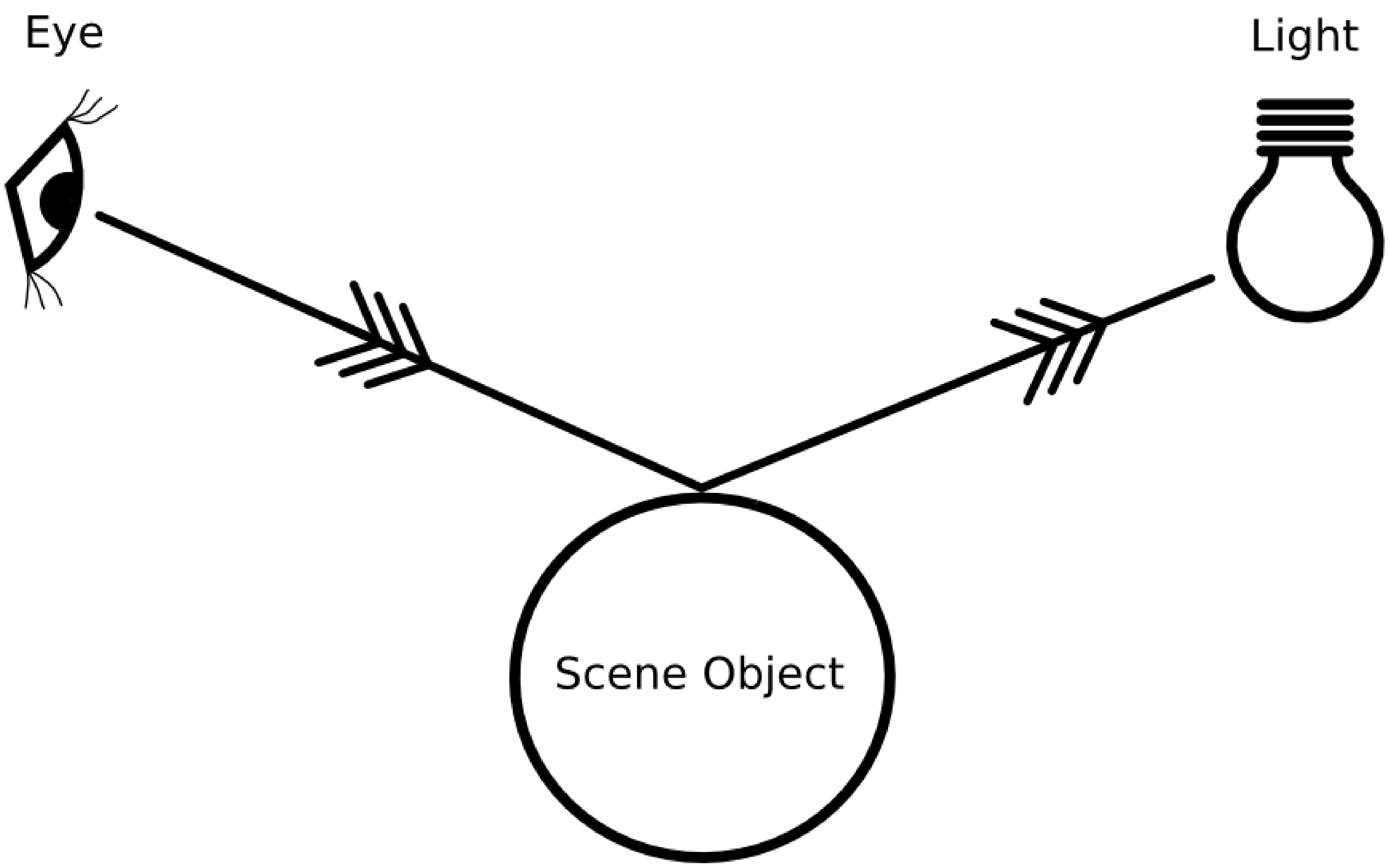
So what's it like to program?

A simple raytracer

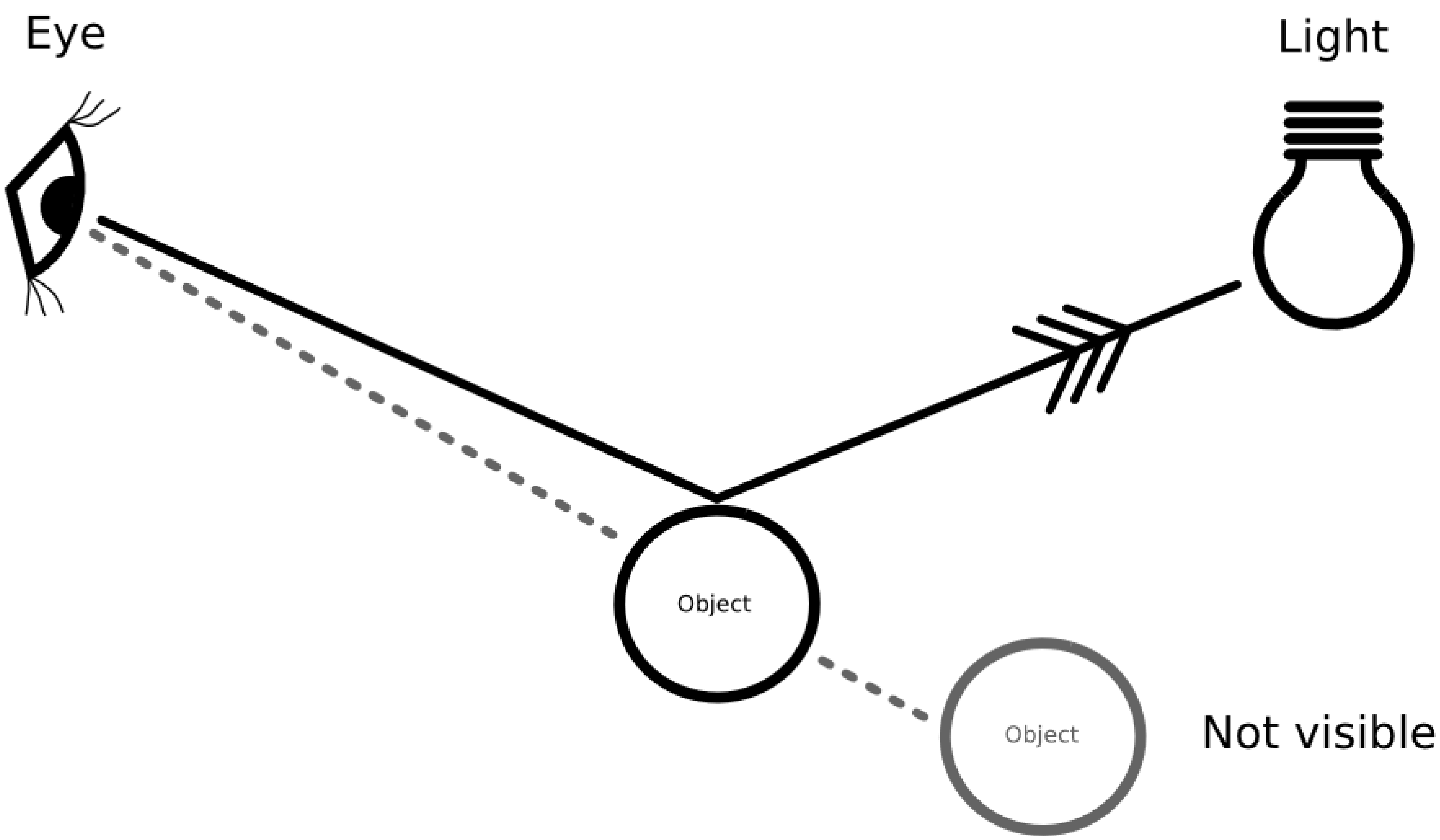
Raytracing #1



Raytracing #2



Raytracing #3



A raytracer in 7 lines

```
for each pixel:
    hit = Nothing
    for each object:
        if ray hits object:
            if object closer than hit:
                hit = object
    pixel = hit.colour
```

It's not quite that simple

```
pixel = hit.colour
```

Actually more like this:

```
pixel = lighting_function(hit)
```

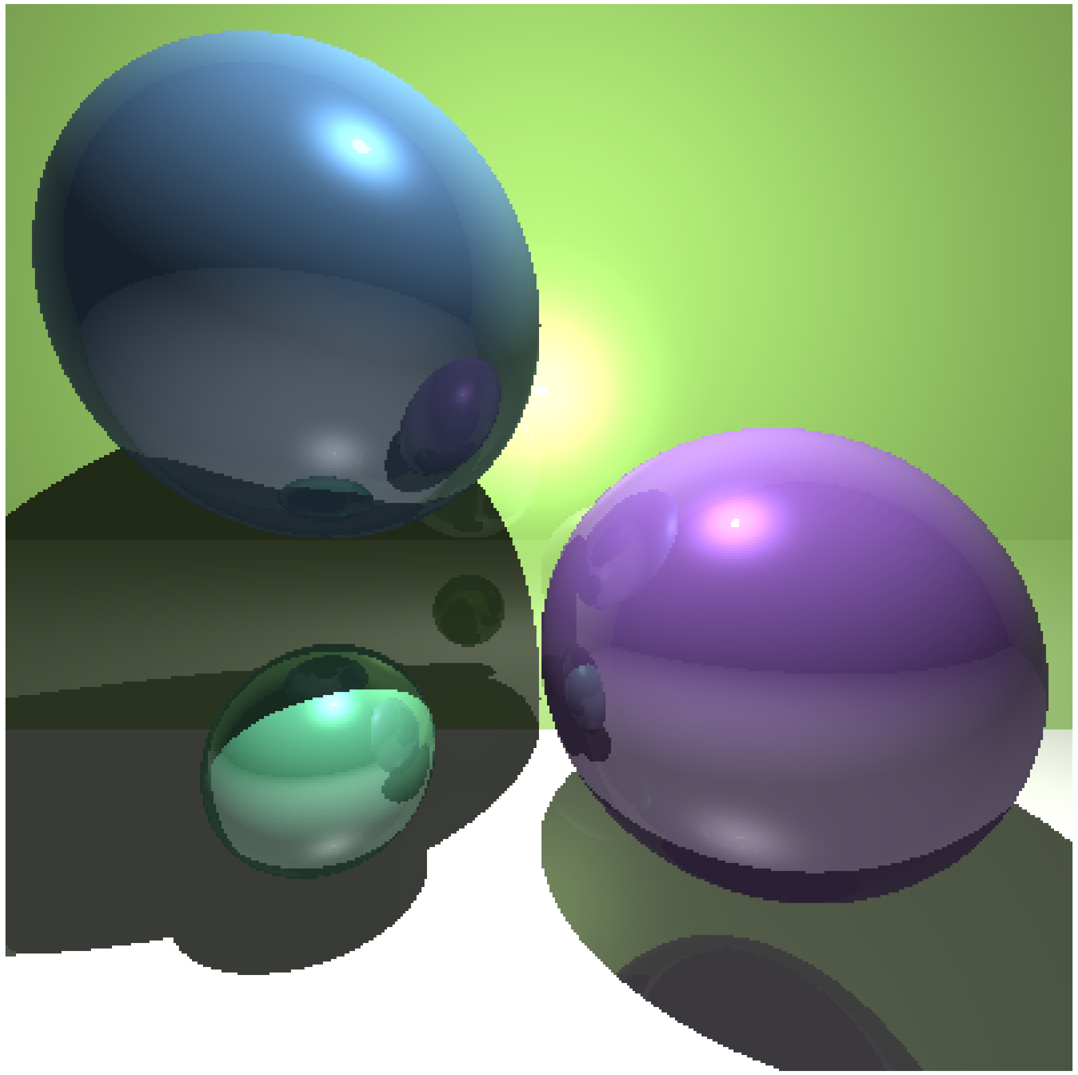
A raytracer on Cell

- It's a new instruction set
- C, C++, Fortran, Ada?
- **C** - close to the metal
- I don't know Fortran

Python prototype first

- Get the algorithms right first
- Python ~= pseudo code
- Library routines for vectors

3 mins 48 s @ 400x400



How to parallelise on Cell?

- 6 SPUs on PS3
- 16 SPUs on IBM QS2x Blades
- One SPU thread per pixel?
- Split hit detection and lighting?
- Each SPU renders 1/nth of the rows?

Thread creation and switch is costly, synchronisation is hard

How to parallelise on Cell?

- By rows: each SPU renders 1/n rows
- For large scenes rectangles would be better - object locality
- Adaptive partitioning
- **Open question IMHO**

PPE Structure

```
load_scene()

for 0 to num_spus:
    threads[i] = spawn_spu_thread(i)

for 0 to num_spus:
    wait_for(threads[i])

save_image()
```

SPU Structure

```
dma_scene_data_from_ppe()

raytrace_scene()

dma_image_to_ppe()
```

This will appear to work, but ..

Let's do some math

- 854 x 480 = 409,920 pixels
- 409,920 x 3 (RGB) ~= 1.2 MB
- 1.2 MB ÷ 6 (SPUs) ~= **200 KB**
- SPU program is **~70 KB**
- How big was local store again?

SPU Structure

```
dma_scene_data_from_ppe()

for each row:

    raytrace_row()

    dma_row_image_to_ppe()
```

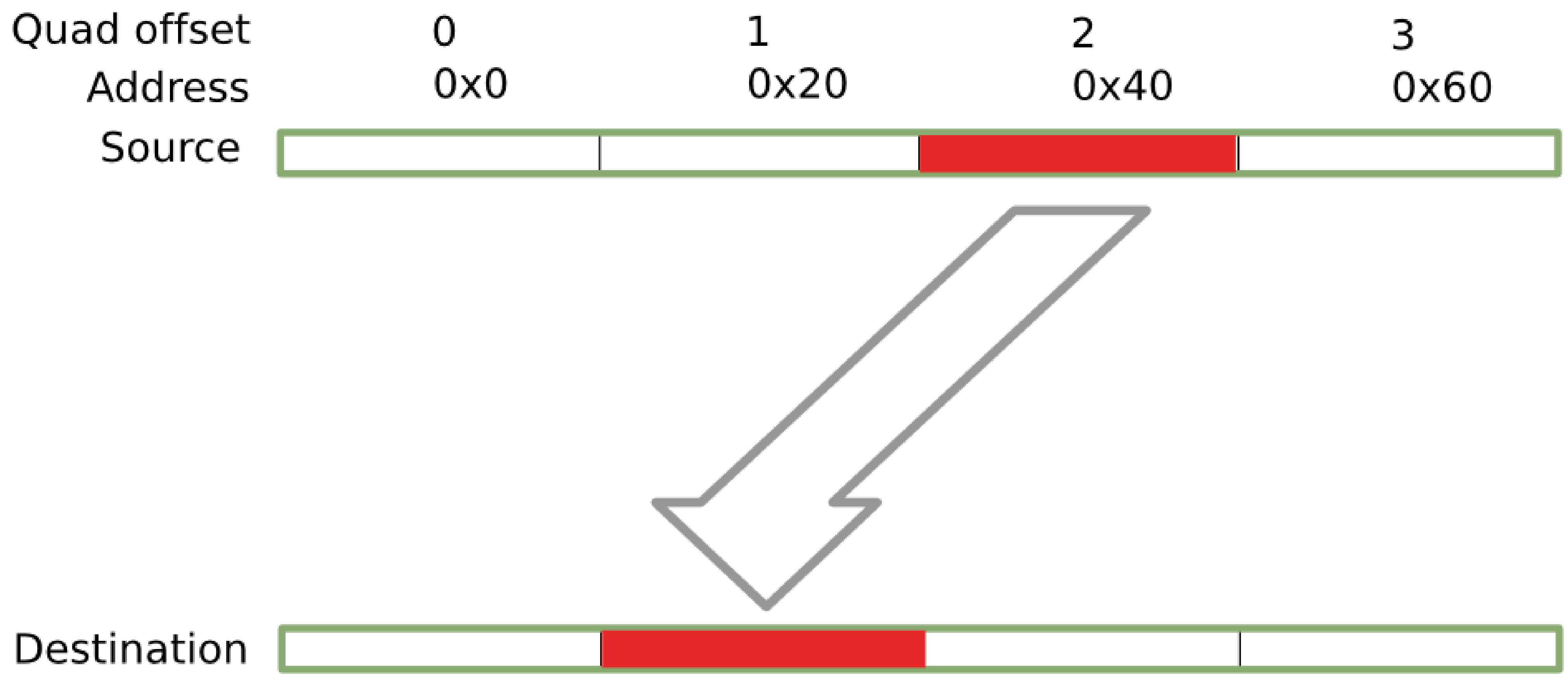
Meet the MFC

- MFC: Memory Flow Controller
- DMA engine in each SPE
- Up to 16 DMAs in flight
- Scatter/Gather support

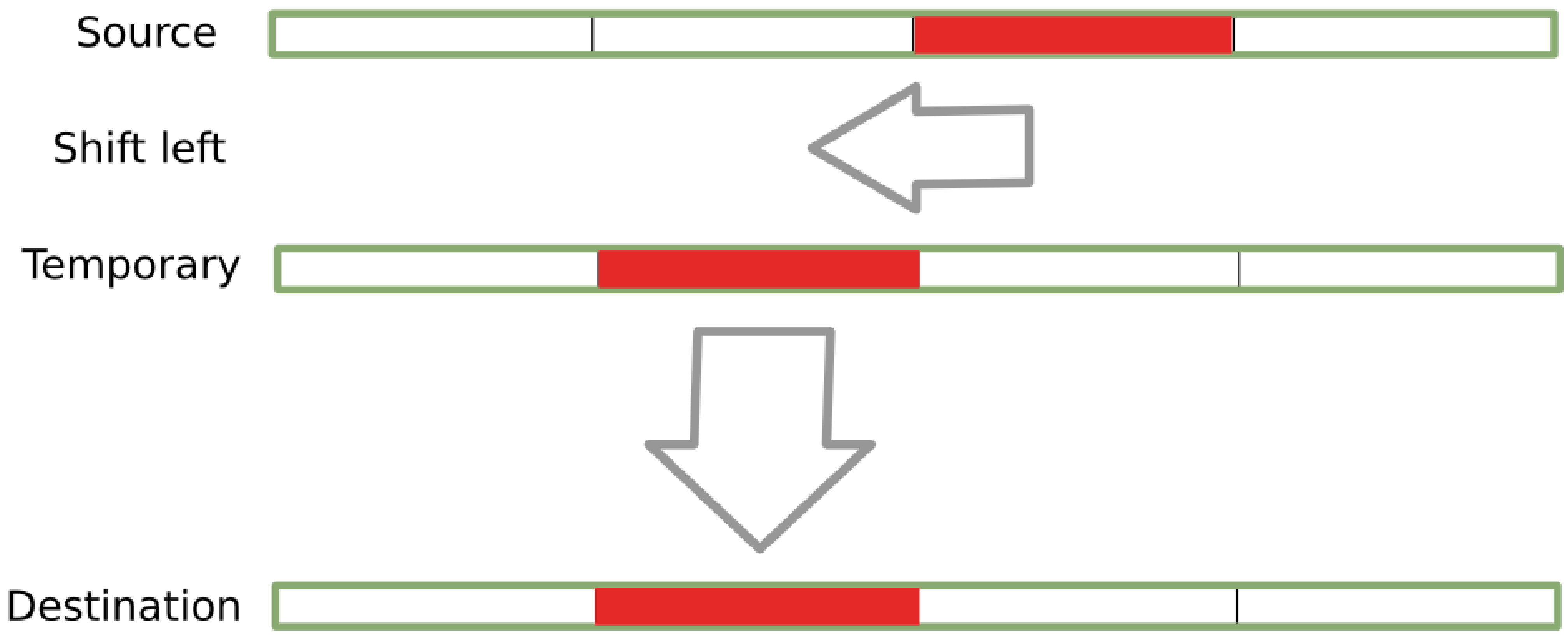
Power at a cost

- DMA sizes 1, 2, 4, 8, 16 bytes or a multiple of 16 bytes upto 16KB
- Must be naturally aligned
- 128 bytes is optimal (cache line)
- Quadword offsets must match

Quadword offsets?



Quadword offsets?



The dreaded "Bus Error"

- Received when DMA goes wrong
- Todo: better error reporting
- 6 SPUs, 80 rows each, 0x320A0 pixels
- 8 SPUs, 60 rows each, 0x25878 pixels

Bad design decision #1

```
struct pixel {
    char r, g, b;
};
```

- 3 bytes!
- Saves alpha byte we don't use
- 1/4 less memory use is good right?

Alignment matters

- 3 byte pixels give weird quadword offsets
- Shift every quadword before DMA'ing
- Shift every quadword as we store pixels

But it's 25% more DMA traffic?

- 1080p, 1920 x 1080 = 2,073,600 pixels
- 3 Bpp = 6,220,800 B = 0.0006s
- 4 Bpp = 8,294,400 B = 0.0008s
- Can DMA **1,250 frames/second**

Alignment & size

- Data structures need to be aligned
- And an appropriate size

```
struct thingo {
    int a, b, c;    /* 32-bit */
    uint_32t pad;
};
```

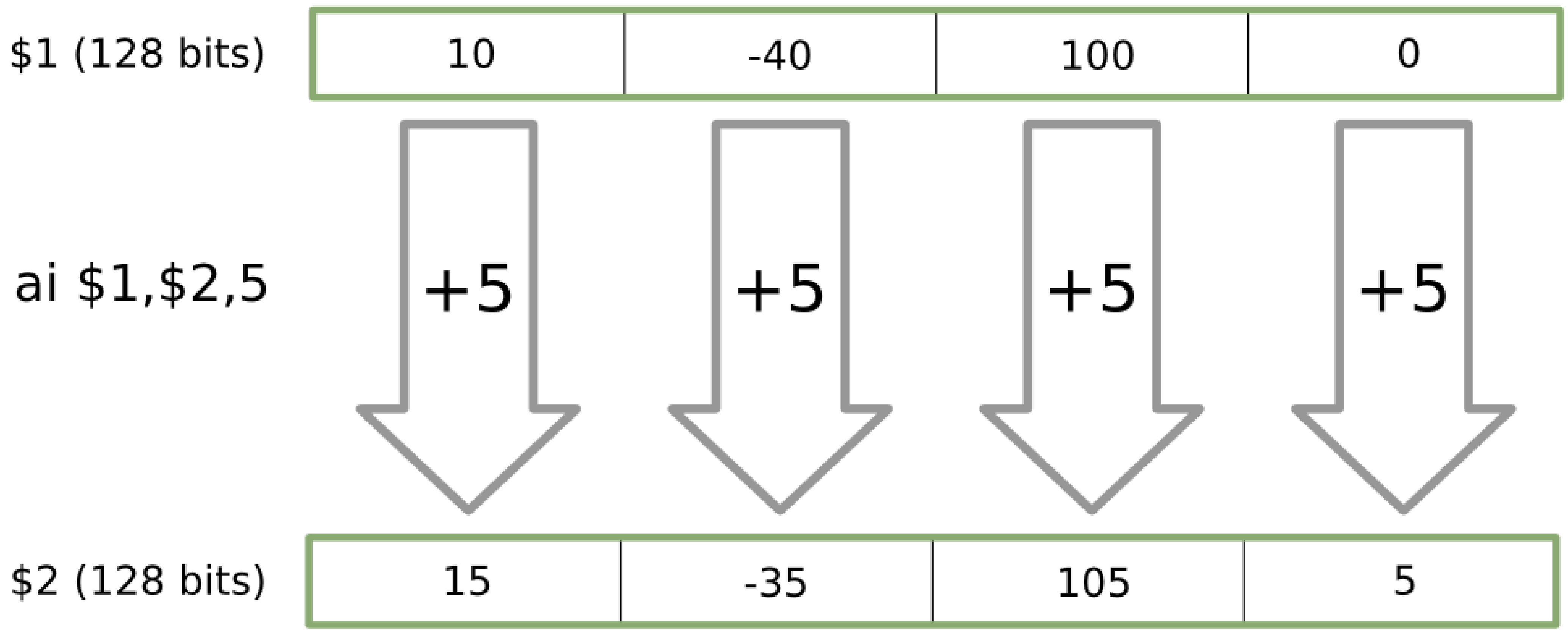
Raytracer core is all 3D vector math

I won't bore you with the details

Vector Registers

- 128-bit wide registers
- 4 floats (single precision)
- 2 doubles (double precision)
- 4 ints/unsigned ints
- 16 chars (bytes)

SIMD 101



Vector Registers

- Each SPU has 128 128-bit registers
- 512 floats in flight (in theory)
- Compiler will use them, it has to
- Can help the compiler out though

Vectorising

```
struct vector {
    x, y, z, w;
} vec;
```

Replace with:

```
vec_float4 vec;
```

A little more raytracing theory

Ray / object intersections

O.O.P

```
struct primitive {
    int type;
    union {
        struct plane plane;
        struct sphere sphere;
    } data;
}
```

OOPs!

```
float primitive_intersect(struct primitive *p,
                          struct ray *ray)
{
    switch (p->type) {
    case PLANE:
        return plane_intersect(p, ray);
    case SPHERE:
        return sphere_intersect(p, ray);
    }
}
```

Branches

- SPUs have no branch prediction
- Missed branches cost 18-19 cycles
- Can't statically predict this branch
- ~50% of the time we'll take the wrong path

No Branches

- Move the test up
- Loop through all spheres, then all planes
- Inside the loop we know what we're dealing with

SPU timing tool

- Part of IBM SDK
- Estimate of execution pattern
- Dual issues
- Stalls

SPU timing tool output

```
000265 0 -----567890                    fm      $80,$4,$79
000271 0       -----123456              fm      $81,$80,$2
000272 0             234567             fnms    $5,$4,$80,$24
000278 0              -----890123       fma     $78,$5,$81,$80
000284 0                    -----456789 fs      $75,$77,$78
```

```
000205 0D      567890                   fs      $19,$68,$19
000205 1D      567890                   lqd     $34,48($30)
000206 0D       678901                  fm      $36,$5,$5
000206 1D       6789                    shufb   $39,$13,$49,$63
000207 0D        789012                 fs      $58,$68,$10
000207 1D        7890                   shufb   $15,$15,$48,$63
```

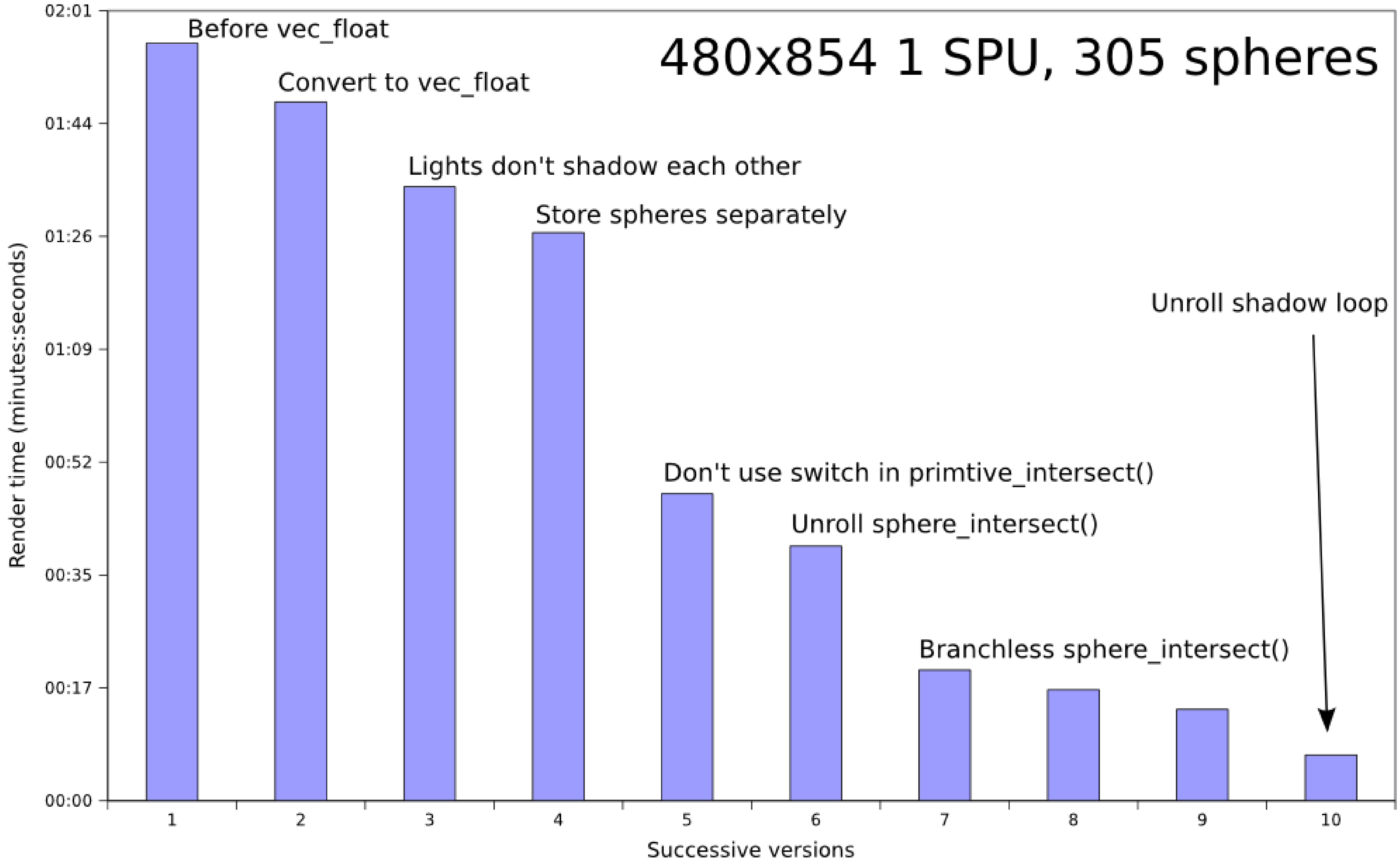
Unroll your loops

- Reduces loop management overhead
- More code in the loop body
- Compiler has more chance to schedule
- Not pretty code

AOS vs SOA

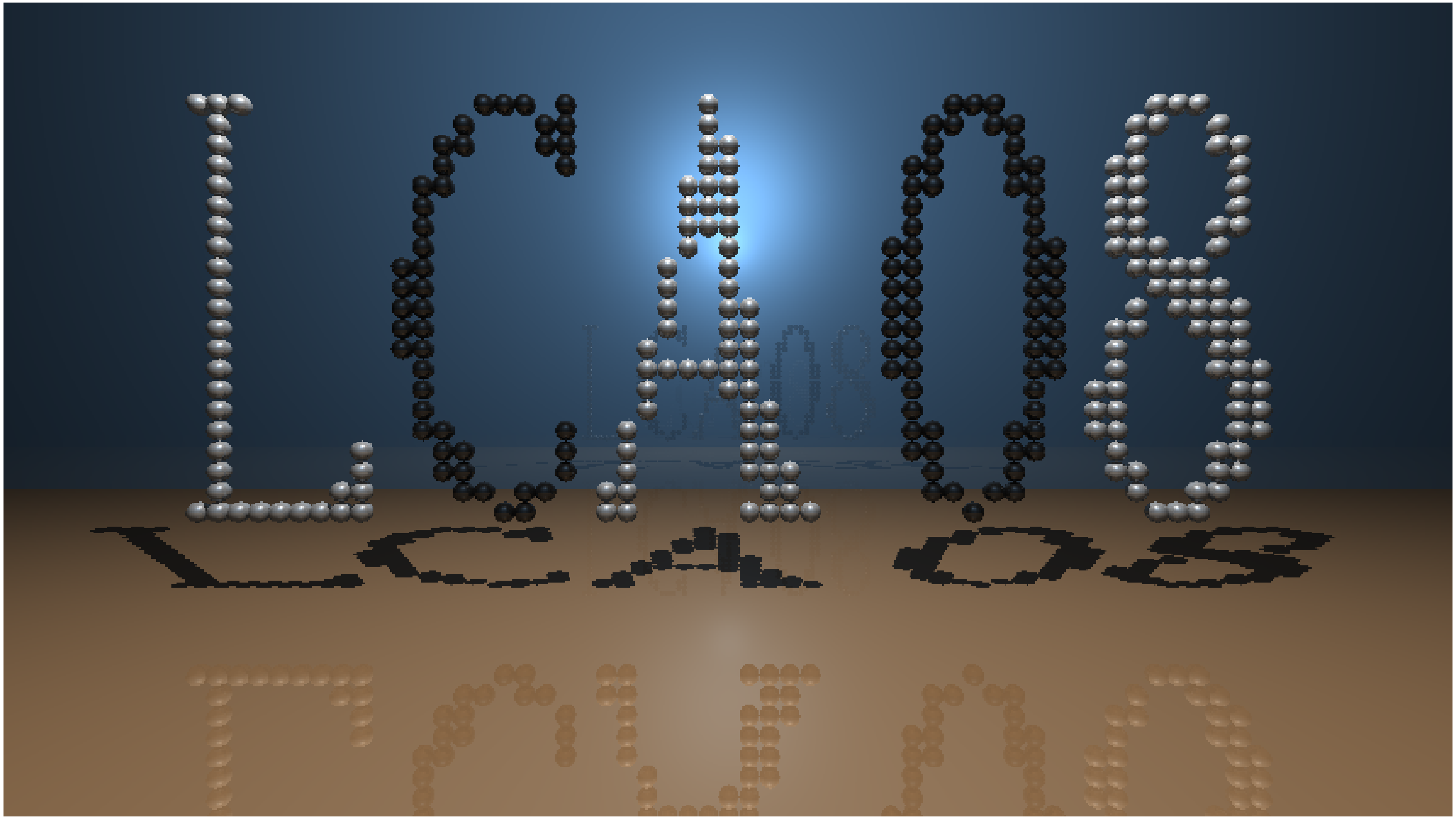
- Array of Structures
- Structure of arrays
- Column vs row vectors
- AOS is intutive, SOA is faster
- Can convert between quite quickly

This slide accidentally left blank


480x854 1 SPU, 305 spheres
Before vec_float
Convert to vec_float
Lights don't shadow each other
Store spheres separately
Don't use switch in primtive_intersect()
Unroll sphere_intersect()
Branchless sphere_intersect()
Unroll shadow loop
Render time (minutes:seconds)
02:01
01:44
01:26
01:09
00:52
00:35
00:17
00:00
1
2
3
4
5
6
7
8
9
10
Successive versions

Bling

292 spheres at 854x480 on 6 SPEs in ~0.65s per frame


LCA08

Random thoughts

- Code quality vs speed
- Single source base?
- Compilers could get better
- Real issues with debugging optimised code

A complex raytracer?

- Space partitioning approach
- Scenes larger than Local Store
- Object caching, DMA prefetching
- More complex lighting
- Dynamic code loading

IBM iRT

- ~300,000 polygon models in real time
- Runs on PS3 and QS2x blades
- Linear scaling across multiple machines
- Several man years of effort
- Awesome

Props to ..

- Jk for his SVGs
- Everyone at OzLabs
- The Böblingen crowd
- **Meg**

Links

- IBM iRT: http://www.alphaworks.ibm.com/tech/irt
- IBM Cell SDK: http://www.ibm.com/developerworks/power/cell/
- My Blog: http://michael.ellerman.id.au/blog

Legal

- This work represents the view of the authors and does not necessarily represent the view of IBM.
- Linux is a registered trademark of Linus Torvalds.
- Cell Broadband Engine is a trademark of Sony Computer Entertainment Inc.
- PLAYSTATION is a trademark of Sony Computer Entertainment Inc.
- Other company, product, and service names may be trademarks or service marks of others.

Questions?