DTU

09/11/23

Real-Time Visual and Machine Learning Systems

Agenda – Module 1 part B

- Where were we?
- A Whirlwind Introduction to GPUs
- The Memory Hierarchy and the GPU
- Back to the Computational Graph
- Computational Graph Definition
- Macro Fusion
- Micro Fusion
- Tying it all together
- Using the framework
- Exercises/Hand-in

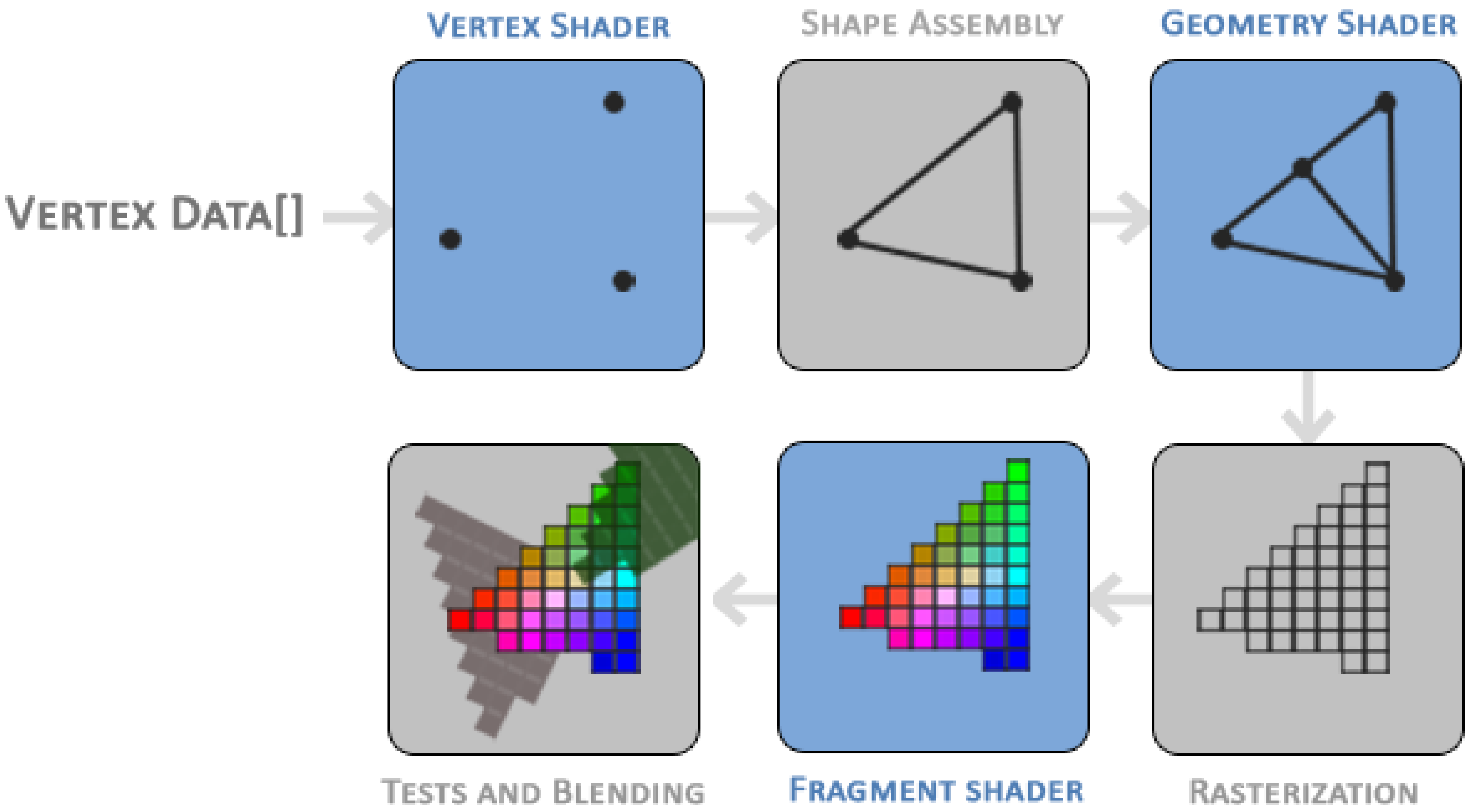
Computational Graphs – Where were we?

- Memory Hierarchies in Hardware
- Memory Hierarchies in Software
- Memory Allocations and Data Structures
- Smart Pointers
- Graph Structures
- Garbage Collectors
- Computational Graphs
- Exercise

A Whirlwind Introduction to GPUs

Matador mix?

A Whirlwind Introduction to GPUs – Graphics Origins



Geometry Shader is optional - [Link]

A Whirlwind Introduction to GPUs – APIs

Name	Platforms	Capability
OpenGL	Windows, Linux, (Mac)	Legacy graphics & compute
Vulkan	Windows, Linux, (Mac)	Graphics & compute
DirectX11	Windows, Xbox	Graphics & compute
DirectX12	Windows, Xbox	Graphics & compute
Metal	Mac	Graphics & compute
WebGL	Web	Legacy graphics
WebGL 2.0	Web	Legacy graphics & compute
WebGPU	Chrome, (Web)	Graphics & compute
CUDA	Nvidia GPU's	Compute
OpenCL (+ ROCm)	GPU's, (-Mac?), CPU, FPGA	Compute

A Whirlwind Introduction to GPUs – Shading Languages

Name	Platforms
GLSL	OpenGL, WebGL, Vulkan
HLSL	DirectX12
CUDA C++ and CUDA Fortran	Nvidia
C++ for OpenCL / SYCL	Linux, Windows, (Mac?)
MSL	Mac
WGSL	WebGPU
SPIR-V	Intermediate Representation

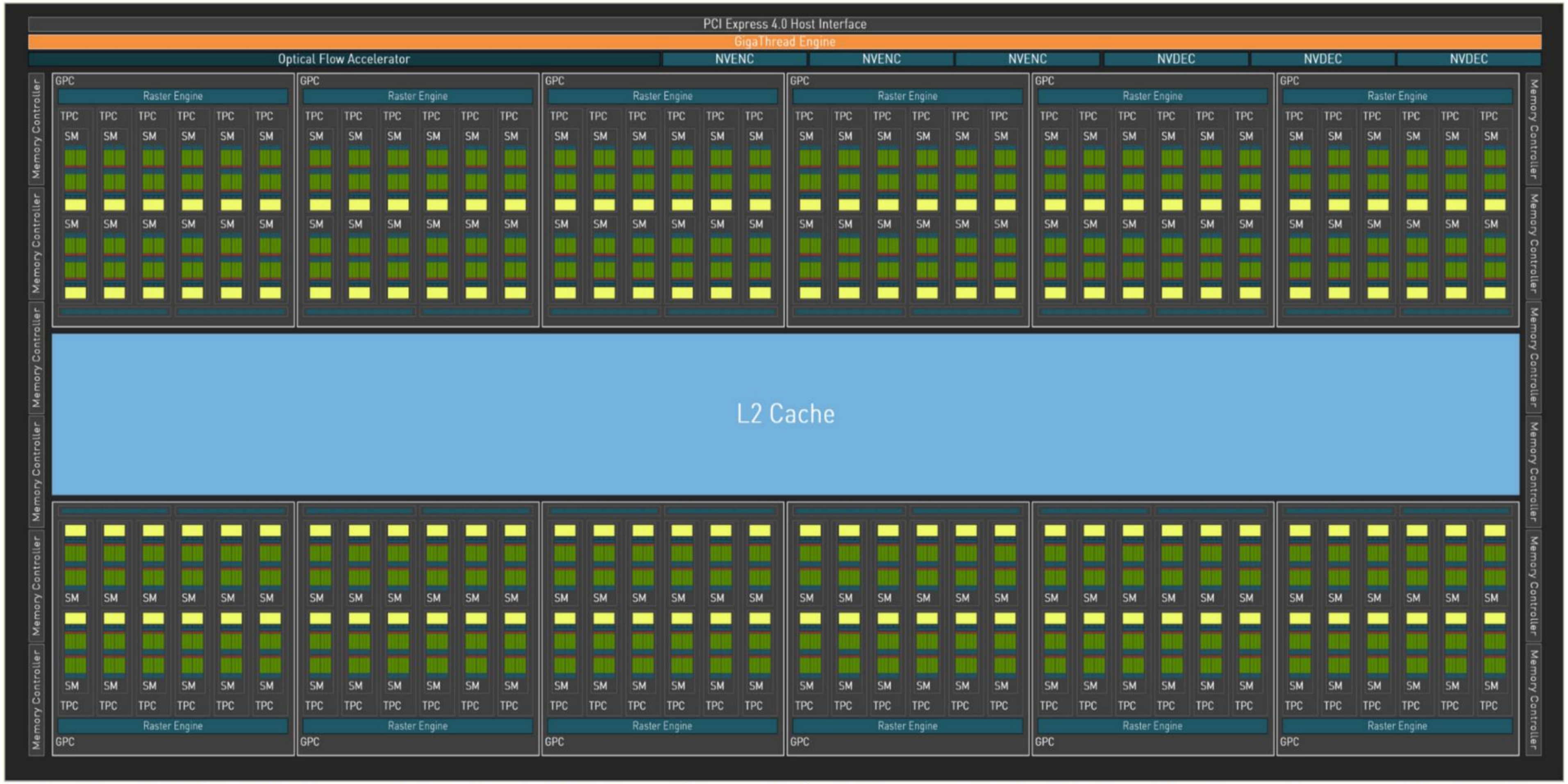
A Whirlwind Introduction to GPUs – H100 Architecture

[Link]

A Whirlwind Introduction to GPUs – H100 Architecture

Link

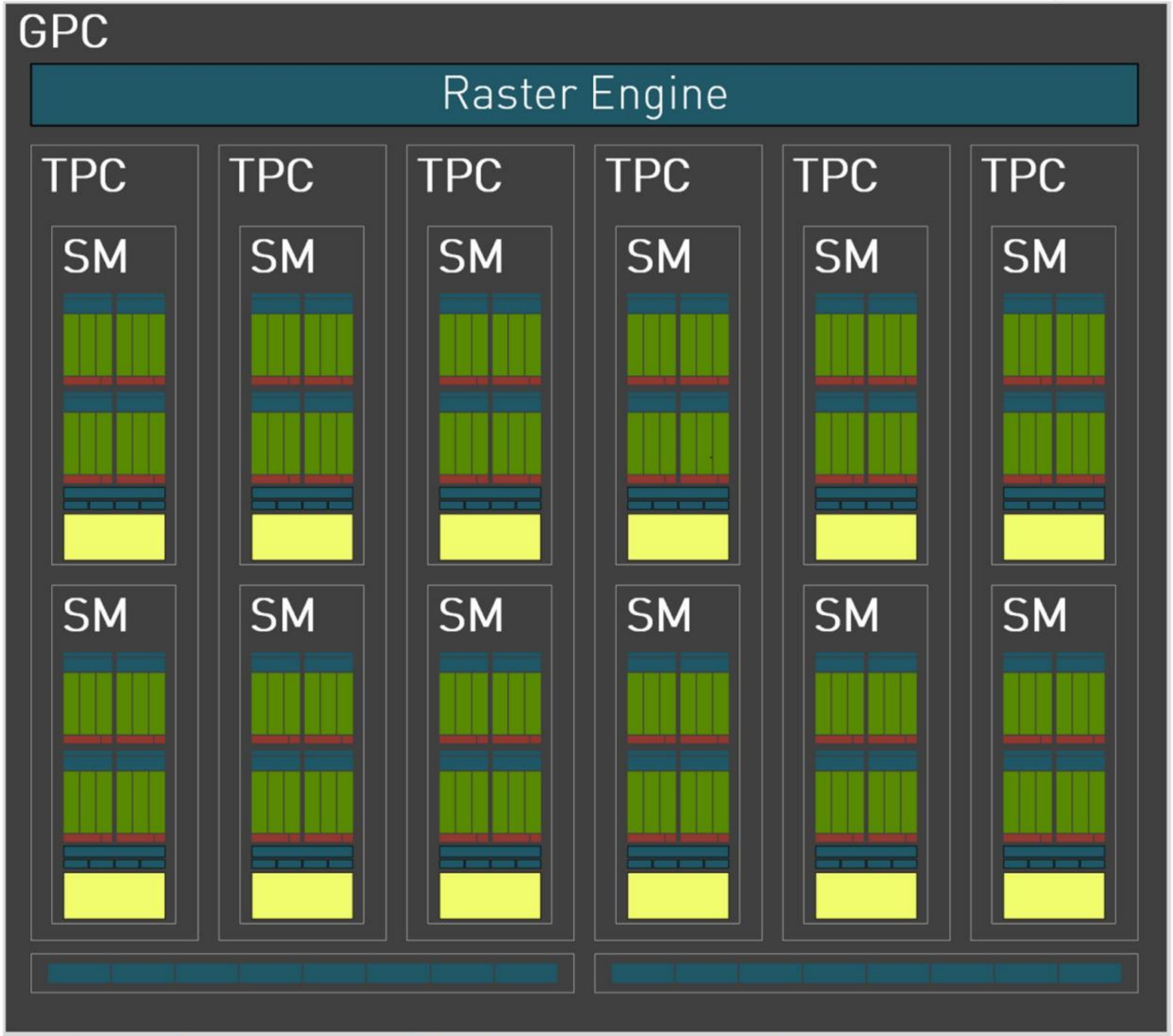
A Whirlwind Introduction to GPUs – 4090 Architecture



Note: The AD102 GPU also includes 288 FP64 Cores (2 per SM) which are not depicted in the above diagram. The FP64 TFLOP rate is 1/64th the TFLOP rate of FP32 operations. The small number of FP64 Cores are included to ensure any programs with FP64 code operate correctly, including FP64 Tensor Core code.

[Link]

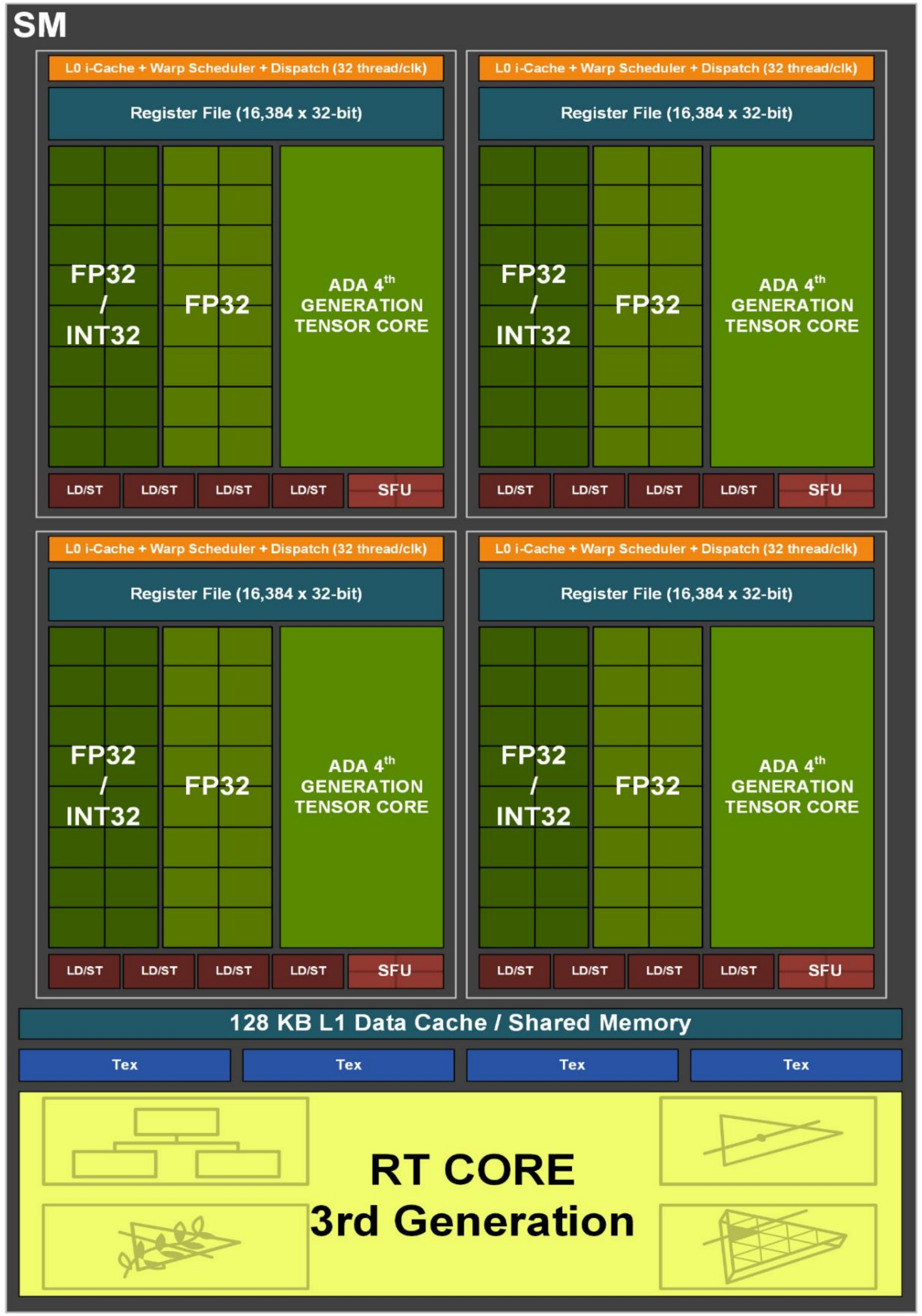
A Whirlwind Introduction to GPUs – 4090 Architecture



Ada GPC with Raster Engine, 6 TPCs, 12 SMs, and 16 ROPs (8 per ROP partition).

Link

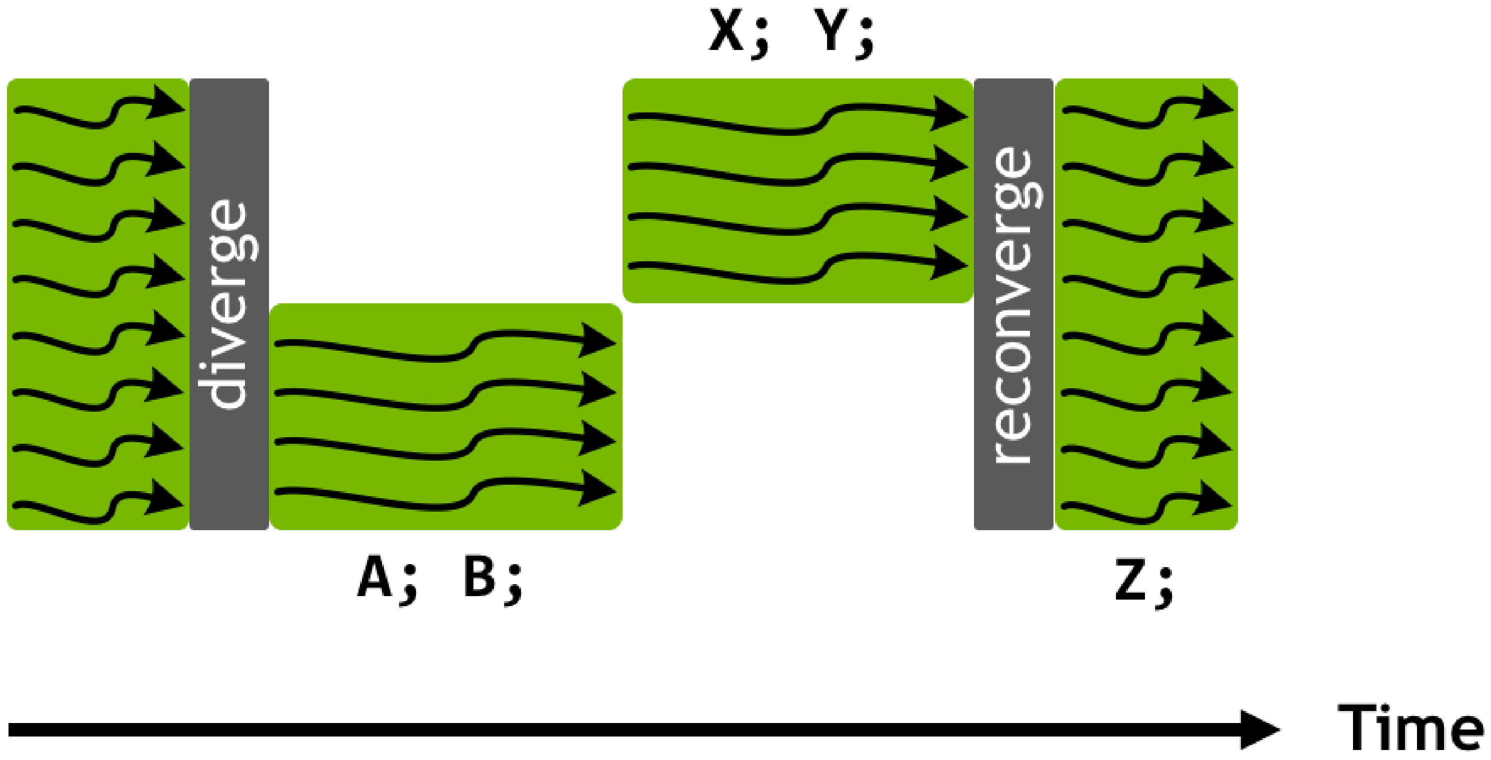
A Whirlwind Introduction to GPUs – 4090 Architecture



Link

A Whirlwind Introduction to GPUs - Architecture

```
if (threadIdx.x < 4) {
    A;
    B;
} else {
    X;
    Y;
}
Z;
```



Link

A Whirlwind Introduction to GPUs – Hello GPU

Let’s look at this link

A Whirlwind Introduction to GPUs – Shared Memory

A programmable L1 cache

Requires a different mindset and structure

Pre-step where you load the needed data

Synchronization

Your function, just working on shared memory

[Link](#)

```
// In general it needs to be verified how we handle odd sizes
// This function should only ever be launched for a single workgroup
@compute @workgroup_size(32, 1, 1)
fn single_pass_sum(
    @builtin(workgroup_id) group_id: vec3<u32>,
    @builtin(local_invocation_id) local_id: vec3<u32>,
    ) {
    let tid: u32 = local_id.x;
    // In this first section we can use all 32 threads
    var elements_left: u32 = sum_uniform.element_count;
    var i: u32 = tid;
    var sum_value: f32 = 0.0;
    // How do we handle the odd case?
    while (BLOCK_SIZE < elements_left) {
        sum_value += data[i];
        elements_left -= BLOCK_SIZE;
        i += BLOCK_SIZE;
    }
    if(tid < elements_left) {
        sum_value += data[i];
    }

    shared_data[tid] = sum_value;
    workgroupBarrier();

```

A Whirlwind Introduction to GPUs – Shared Memory

```
if (tid == 0u) {
    var sum_value: f32 = 0.0;
    var index: u32 = 0u;
    while (index < BLOCK_SIZE) {
        sum_value += shared_data[index];
        index++;
    }
    output[0] = sum_value;
}
```

Exercises & Break

Warm up, by playing around with the gpu_add project and changing some stuff.

Could you create a second input buffer and use it in the calculation?

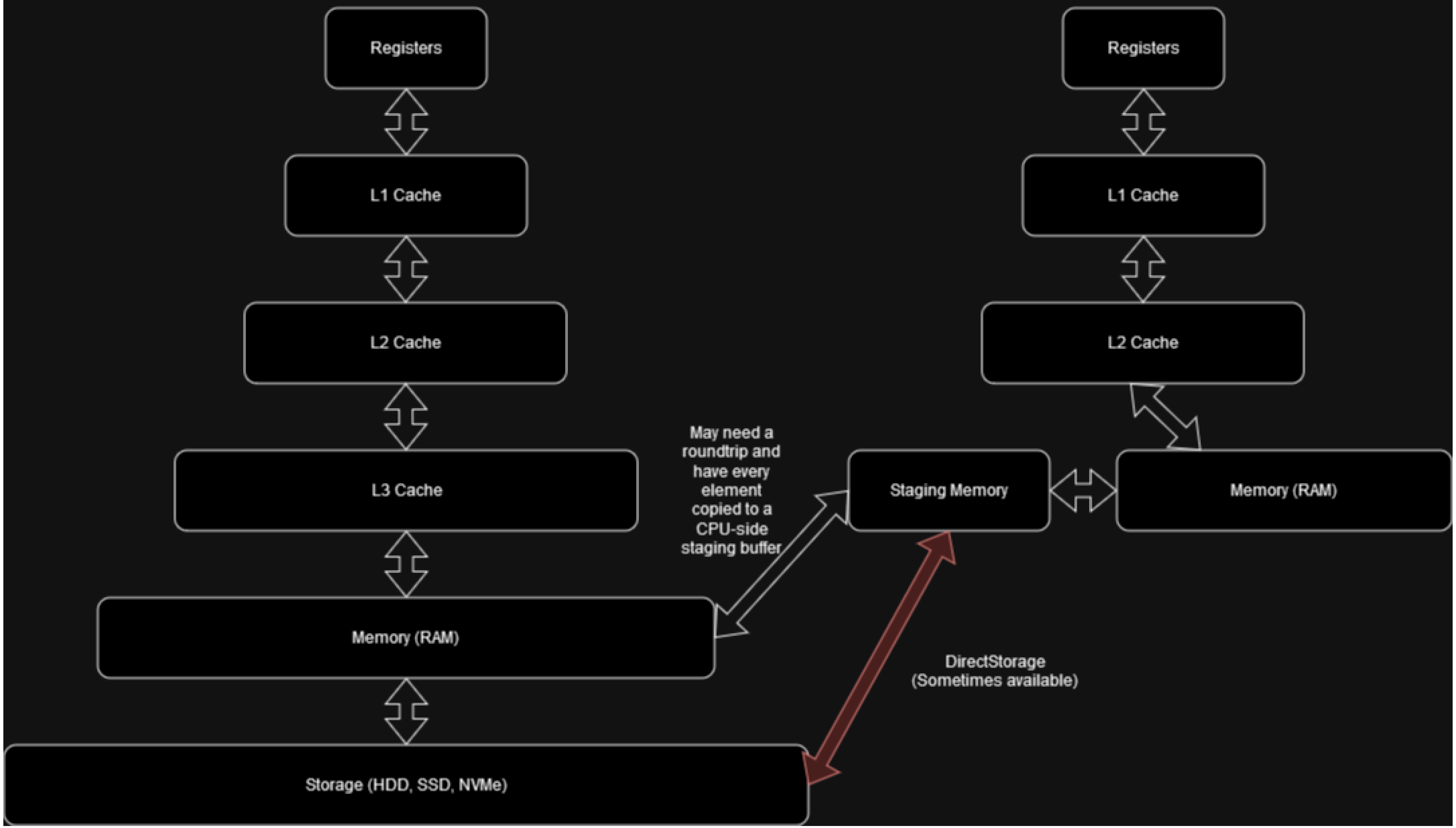
Could you make a simple convolution example where you load a convolution kernel and a range of data into shared memory and compute a 1D convolution filter?

The Memory Hierarchy and the GPU

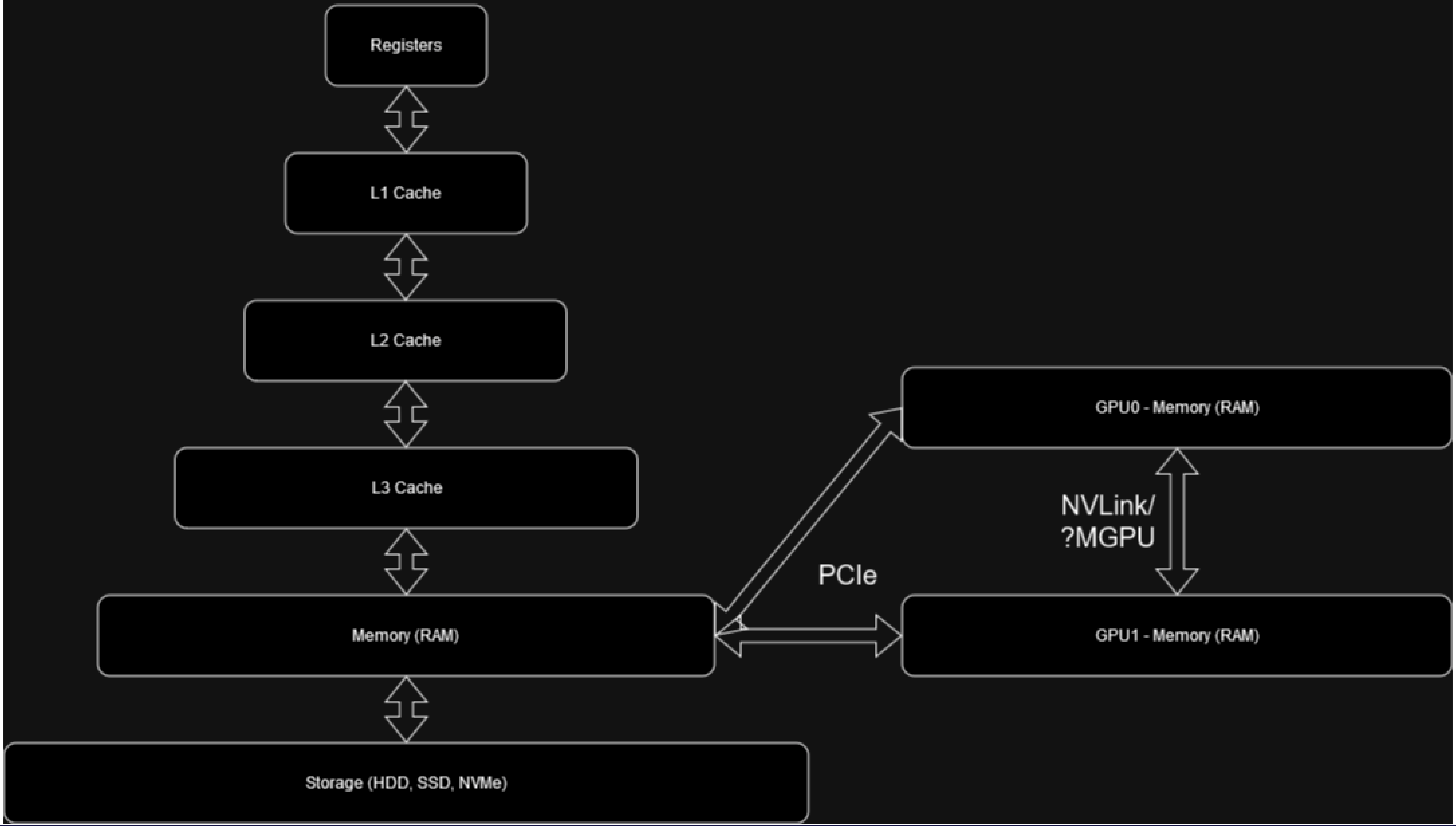
A memory hierarchy friend for your memory hierarchy

Integrated GPU’s – Lowering the cost of transfers

CPU-GPU Hierarchy Simplified

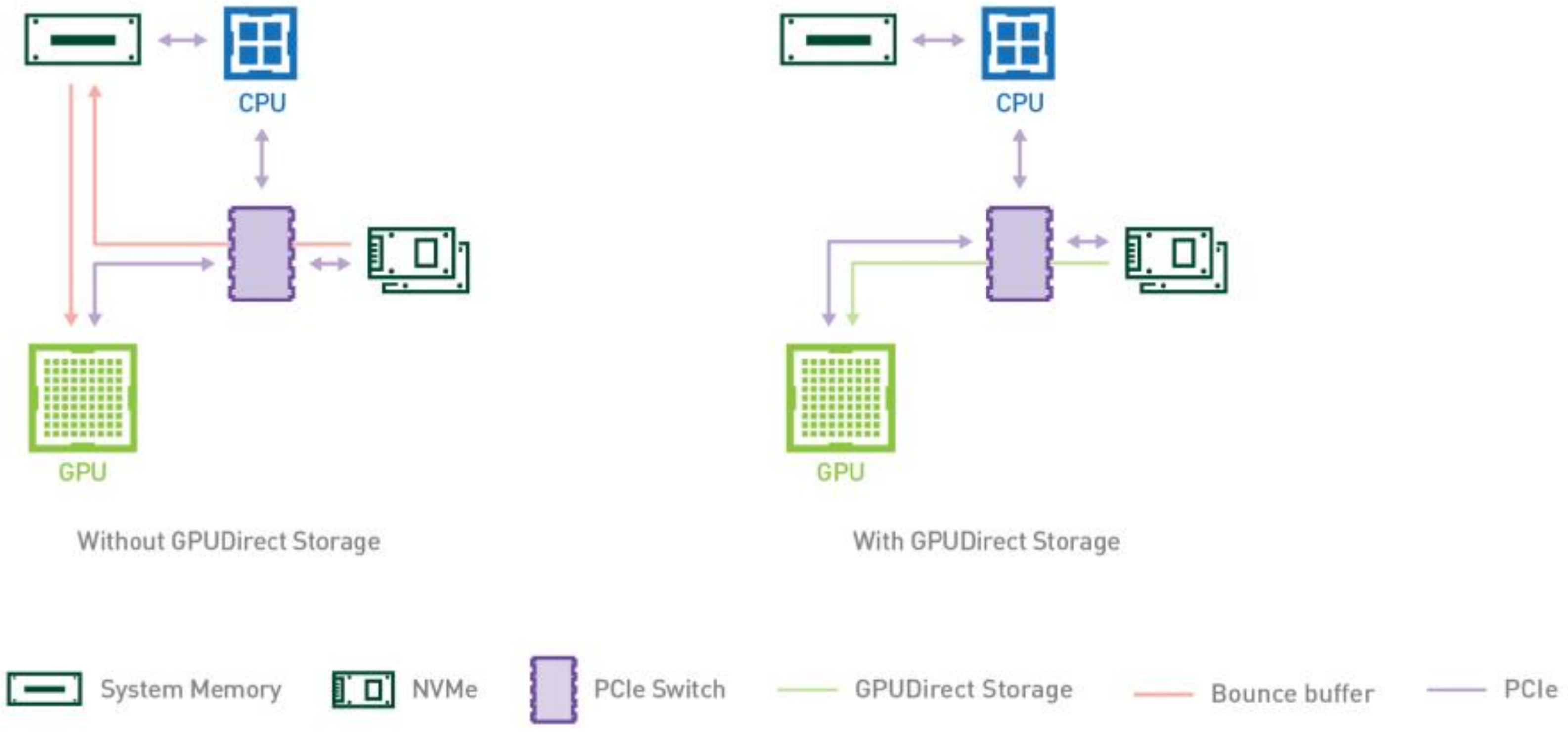


Dual GPU Hierarchy Simplified



The Memory Hierarchy and the GPU

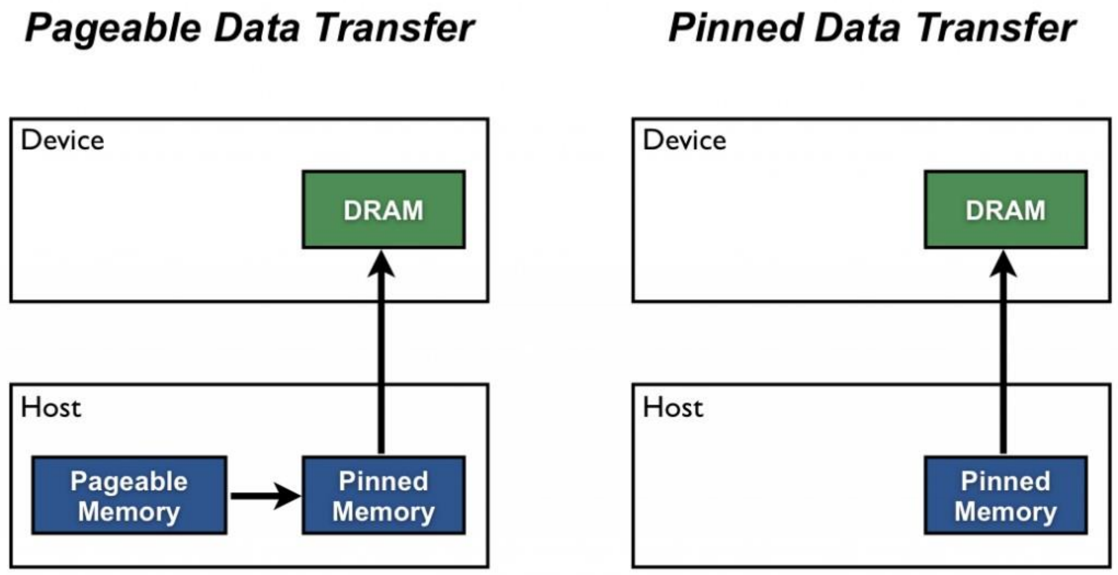
GPUDirect



The Memory Hierarchy and the GPU - Transfers

Pinned memory – we have a segment of memory on the Host (CPU) side which is guaranteed to be read-only for the GPU. Pinned memory.

Staging memory – A part of the GPU memory which is visible from the CPU, internally transferred to usable memory inside the GPU, but no longer Host visible



The Memory Hierarchy and the GPU - Registers

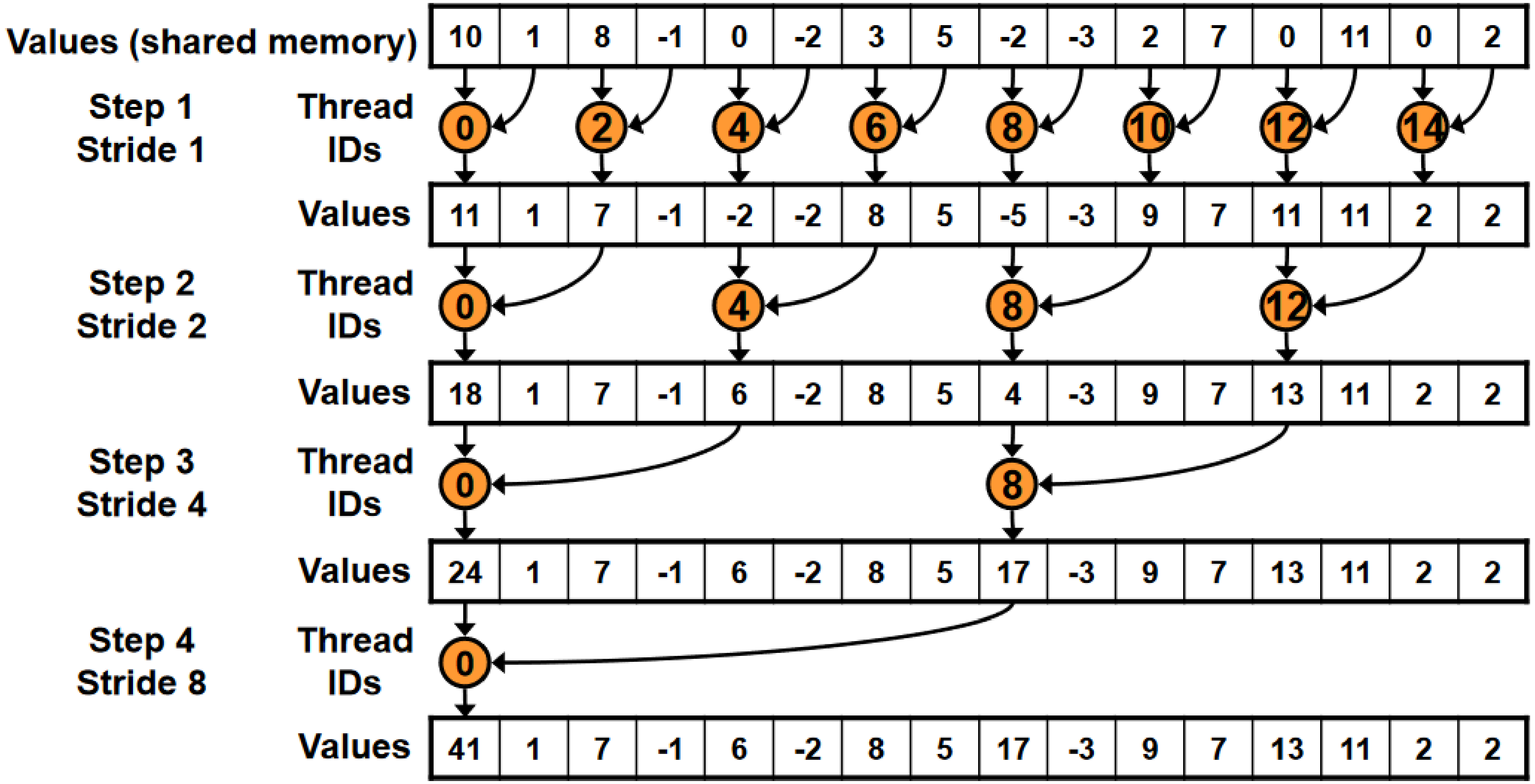
Register File – for swapping memory from different threads

Register Pressure – if each thread uses too much memory to contain it all in the register file,
we need to launch fewer threads

Link

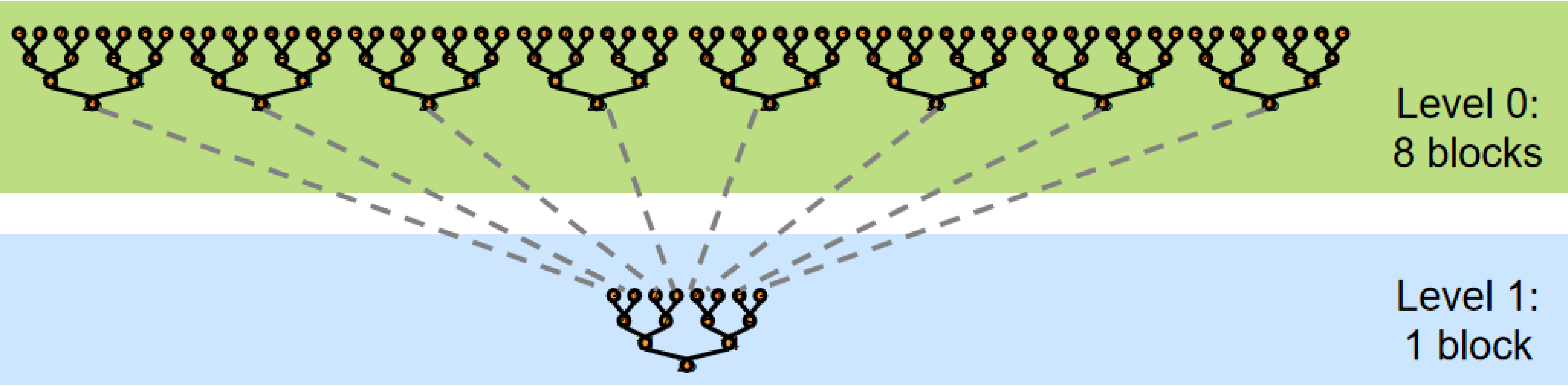
A Whirlwind Introduction to GPUs – The Next Level

[Tree Reductions]()



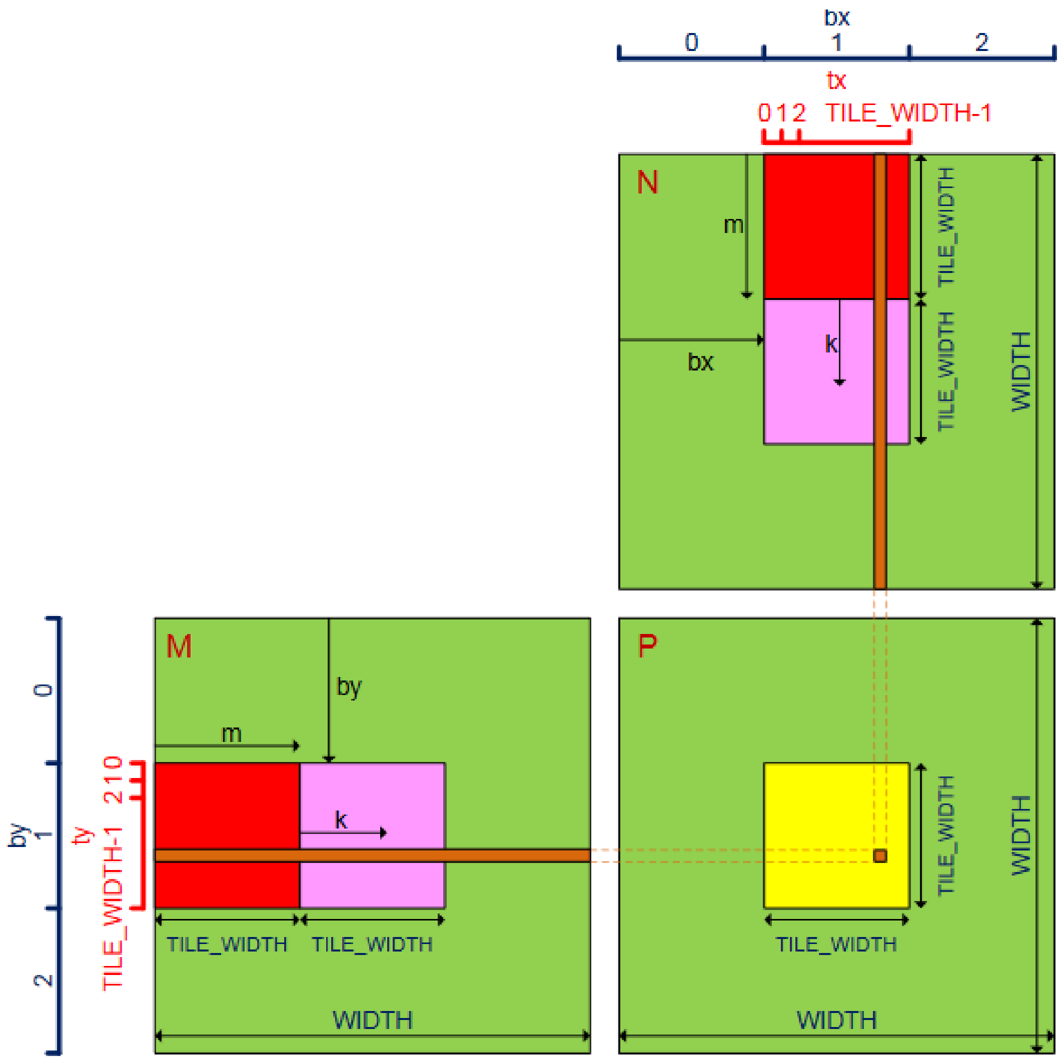
A Whirlwind Introduction to GPUs – The Next Level

Tree Reductions

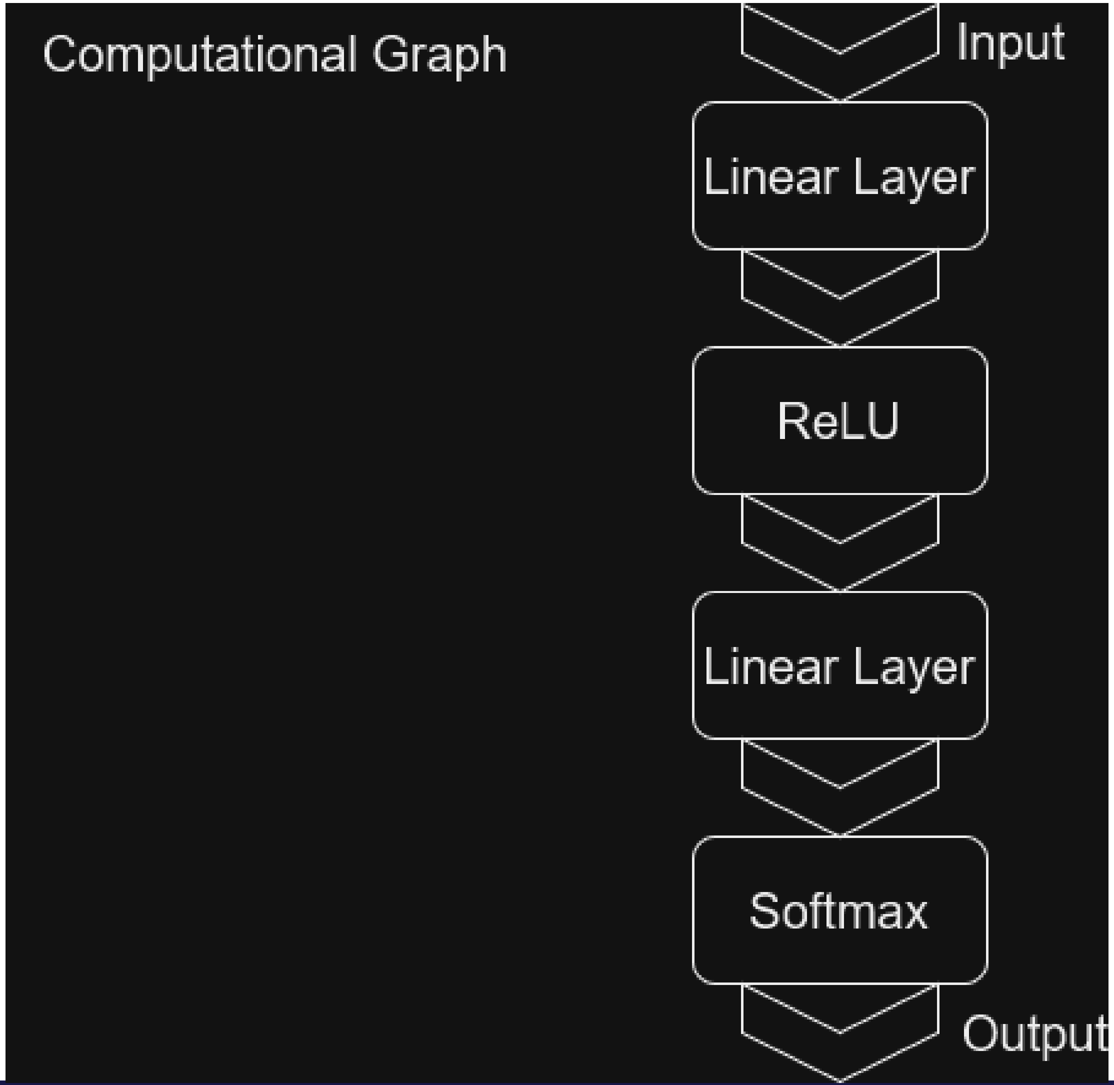


A Whirlwind Introduction to GPUs – The Next Level

[Tiled Matrix Multiplication]()



Back to the Computational Graph



Computational Graph Definition

```rust
    let input: Tensor2D = Tensor2D::new(0.5, size, size);
    let mut graph: Vec<GraphOperator> = vec![GraphOperator::HostToDevice { input }];

    let mut rng: ChaCha8Rng = ChaCha8Rng::seed_from_u64((depth * size) as u64);
    for _ in 0..depth {
        let weights: Tensor2D = Tensor2D::new(0.5, size, size);
        let bias: Tensor2D = Tensor2D::new(0.1, size, size);
        graph.push(GraphOperator::LinearLayer { weights, bias });

        let layer_type: usize = rng.gen_range(0..2);
        if layer_type == 1 {
            graph.push(GraphOperator::ReLU);
        }
    }
    match graph[graph.len() - 1] {
        GraphOperator::ReLU => {}
        _ => graph.push(GraphOperator::ReLU),
    };

    graph.push(GraphOperator::Softmax);
    graph.push(GraphOperator::DeviceToHost);

    let mut out: Tensor2D = Tensor2D::new(0.0, size, size);
```

Kernel Fusion

Swap out operator pairs, for fused operator versions

If Linear->ReLU, swap out for LinearReLU operator

Does not scale well due to the combinatorial explosion

Micro Fusion

Programs are just strings; we can compile new ones!

Op-codes

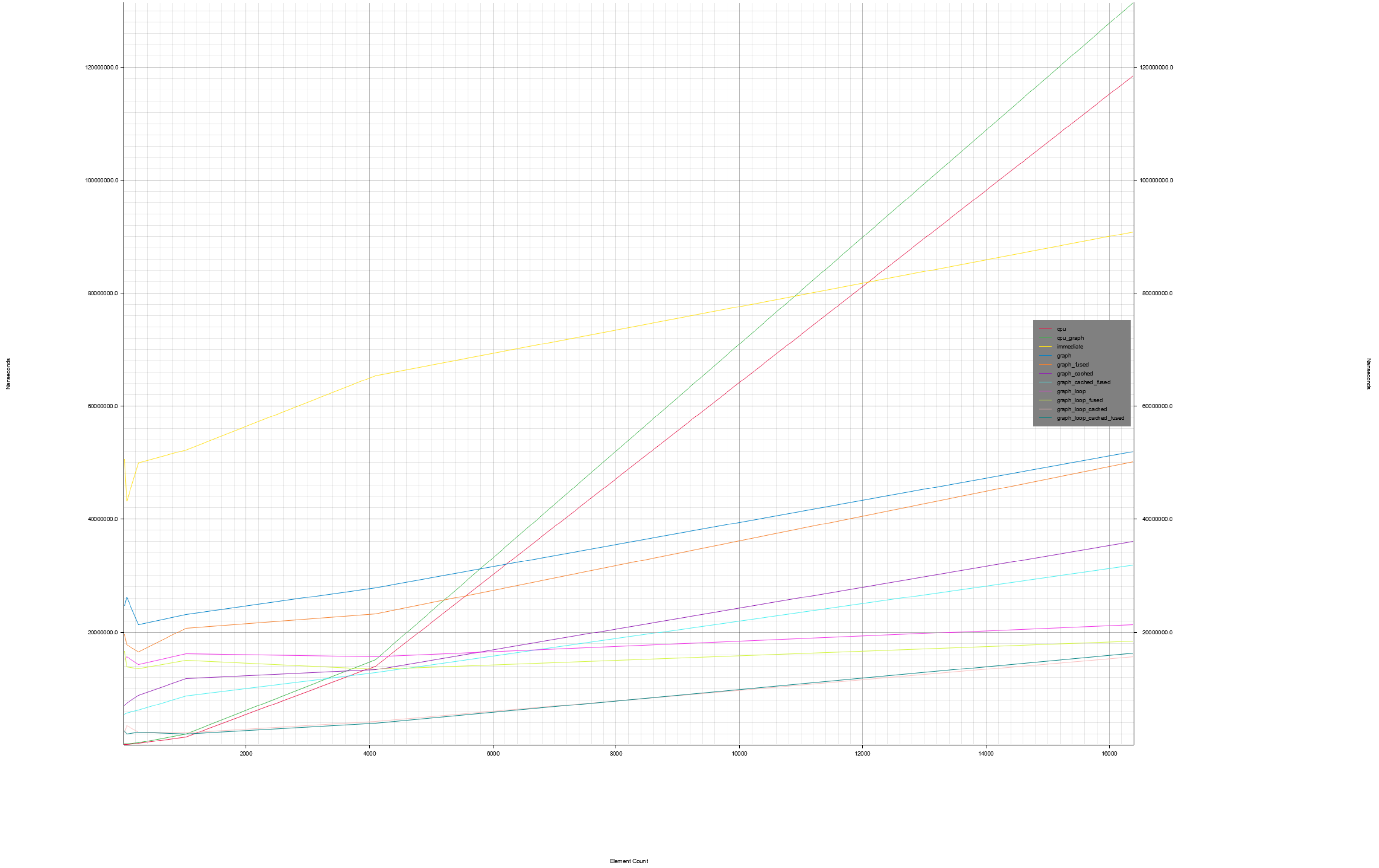
Scales better but is hard to develop for

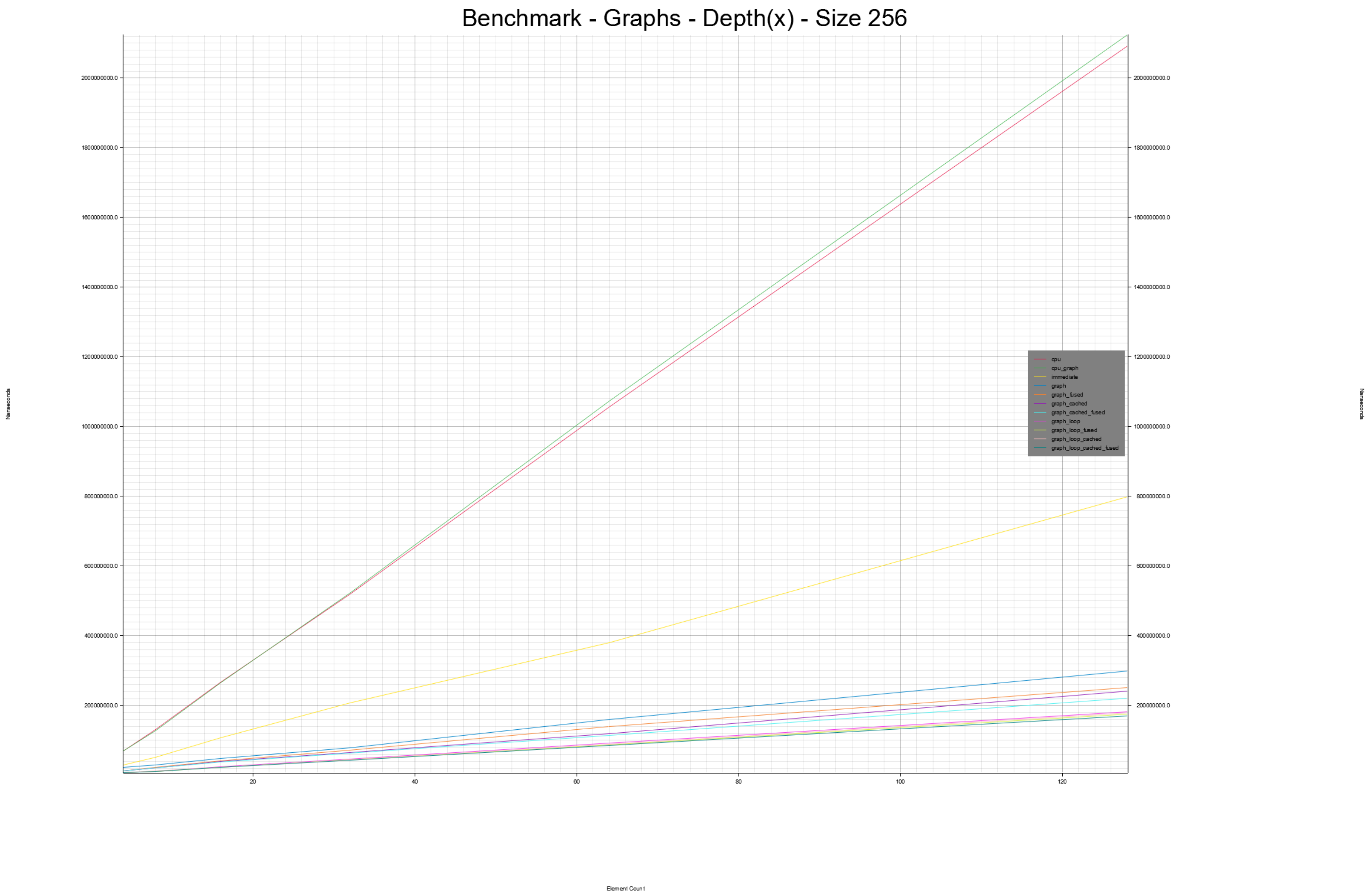
Or… op-code annotation and parsing?

Tying it all together

- CPU
- Immediate GPU
- GPU Graph
- GPU Graph in a loop

Benchmark - Graphs - Size(x) - Depth 64




Benchmark - Graphs - Depth(x) - Size 256
Nanoseconds
2000000000.0
1800000000.0
1600000000.0
1400000000.0
1200000000.0
1000000000.0
800000000.0
600000000.0
400000000.0
200000000.0
20
40
60
80
100
120
Element Count
cpu
cpu_graph
immediate
graph
graph_fused
graph_cached
graph_cached_fused
graph_loop
graph_loop_fused
graph_loop_cached
graph_loop_cached_fused

Using the framework

Link or find it in the-guide::m1_memory_hierarchies::code::computational_graphs

Exercises/Hand-in

You can do one or more from the list, but I recommend doing a tiled matrix-matrix multiplication version of the linear layer shader and adding it to the benchmark graph.