
DTU

09/11/23

Real-Time Visual and Machine Learning Systems

Projects and January

Projects can be done whenever you want

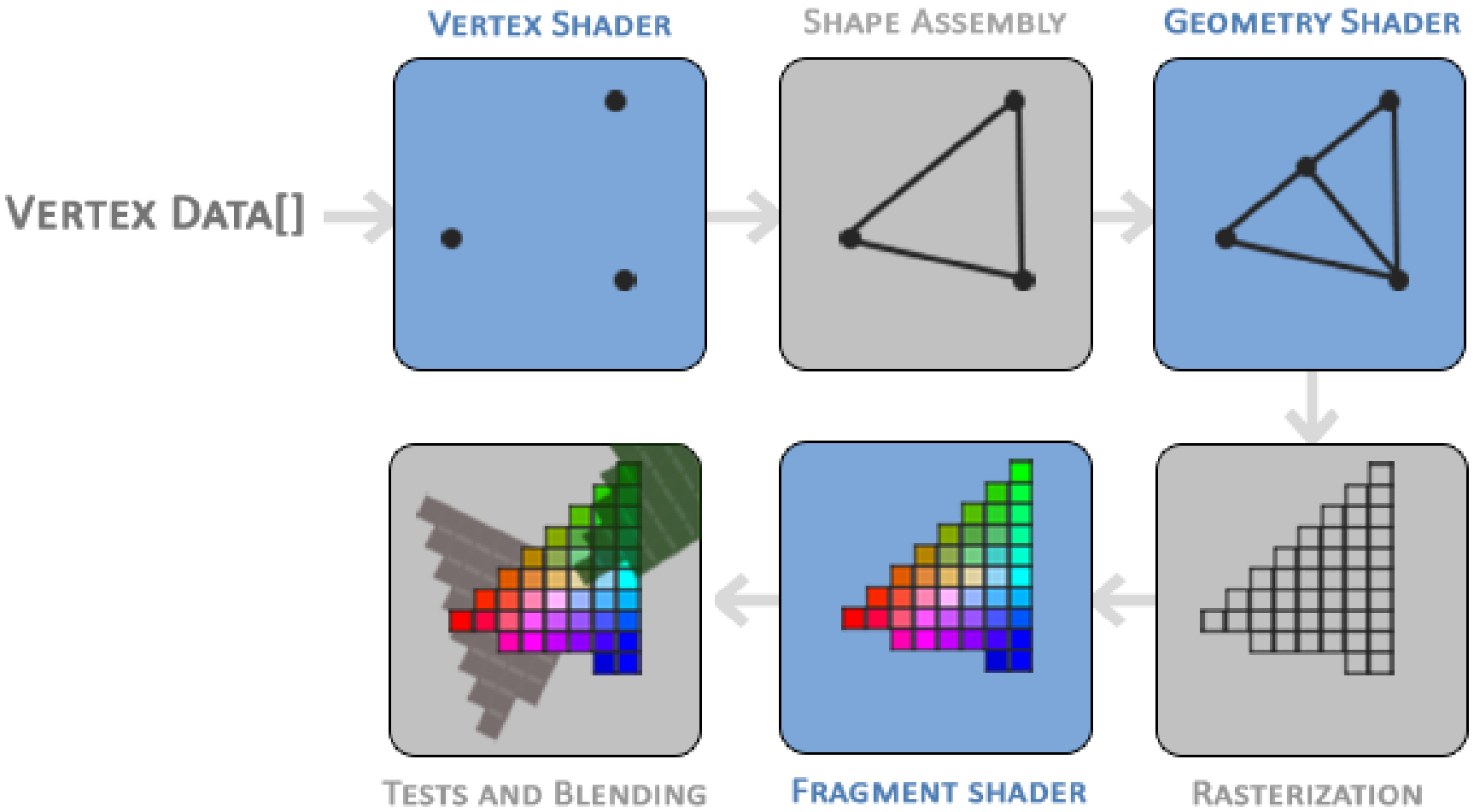
Start thinking about a project

We can talk it through and approve it during the last December lecture

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 – 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

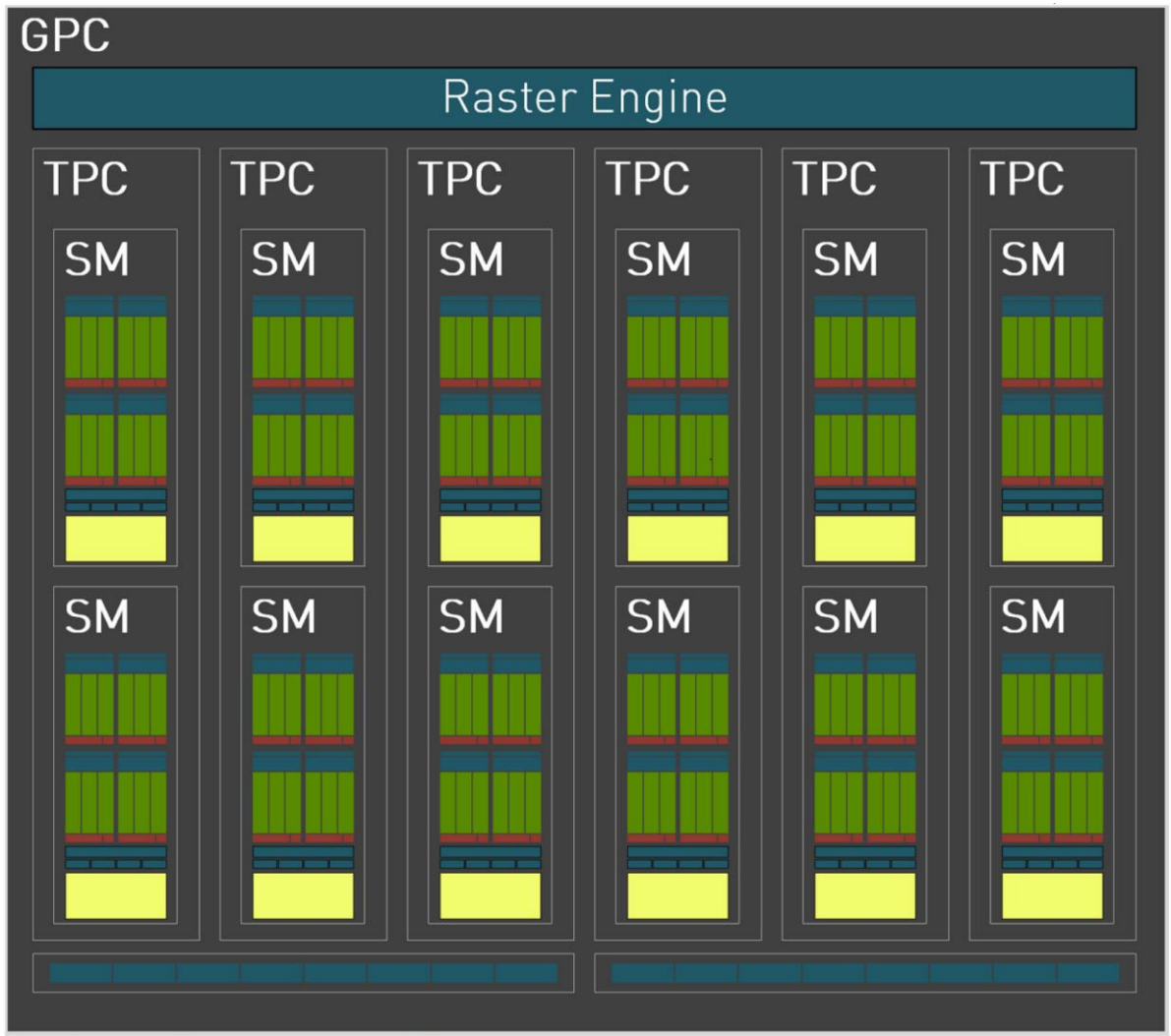
Link

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

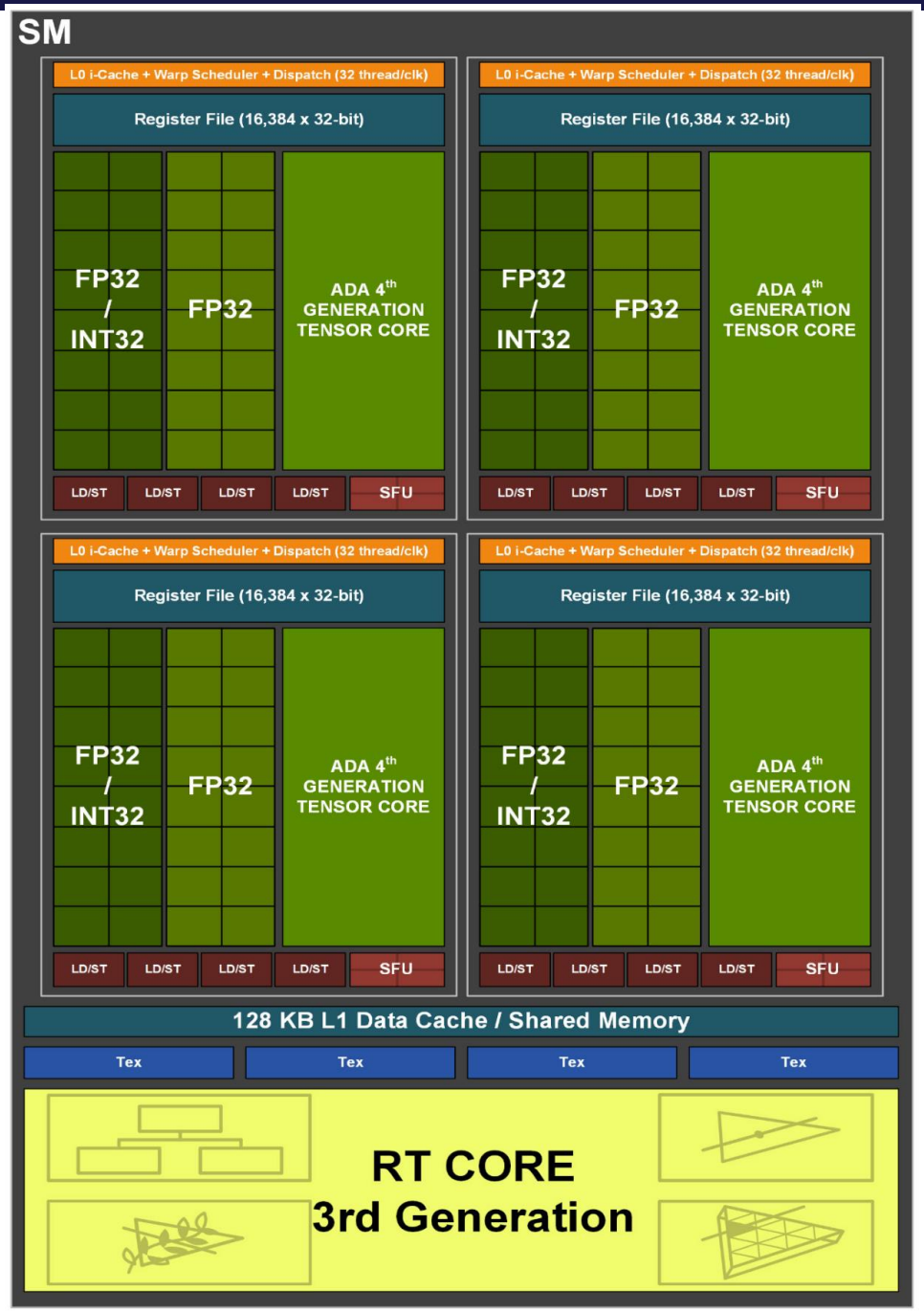
A Whirlwind Introduction to GPUs – 4090 Architecture



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

Link

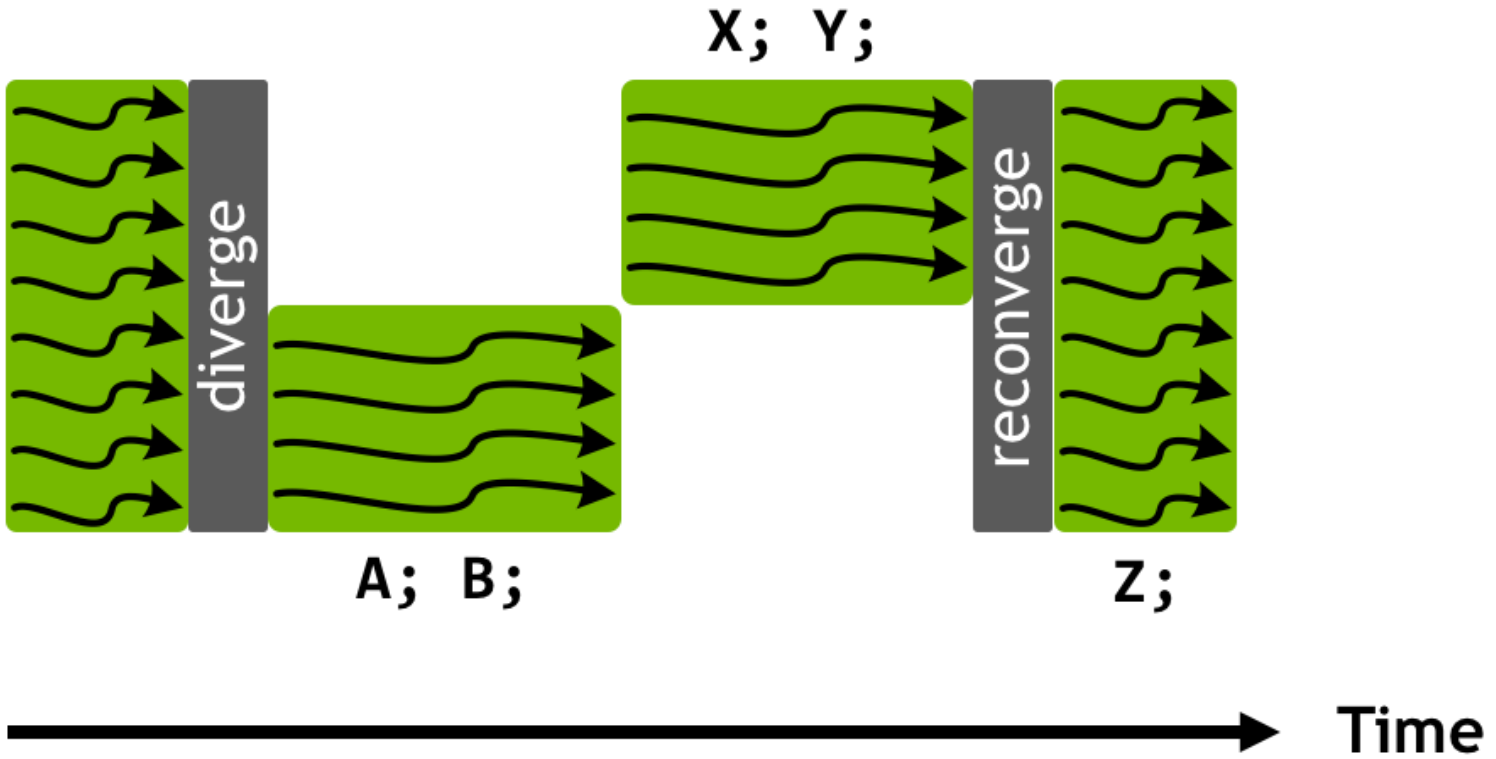
A Whirlwind Introduction to GPUs – 4090 Architecture



Link

A Whirlwind Introduction to GPUs - Architecture

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



Link

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 – Hello GPU

Let’s look at this link

Exercises & Break

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

Could you create a third input buffer and use it in the calculation, such as multiplication?

A Whirlwind Introduction to GPUs – Shared Memory

A programmable L1 cache

Requires a different mindset and structure

Add a step to your code where you load the needed data into shared memory

Synchronization

Your function, working from shared memory

Link

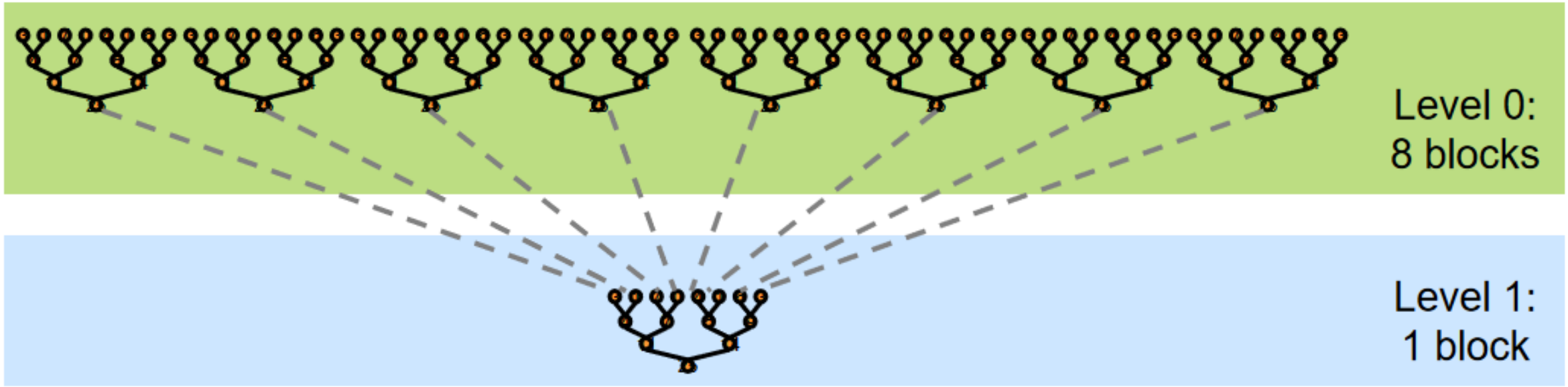
```
159  // In general it needs to be verified how we handle odd sizes
160  // This function should only ever be launched for a single workgroup
161  @compute @workgroup_size(32, 1, 1)
162  fn single_pass_sum(
163      @builtin(workgroup_id) group_id: vec3<u32>,
164      @builtin(local_invocation_id) local_id: vec3<u32>,
165      ) {
166      let tid: u32 = local_id.x;
167      // In this first section we can use all 32 threads
168      var elements_left: u32 = sum_uniform.element_count;
169      var i: u32 = tid;
170      var sum_value: f32 = 0.0;
171      // How do we handle the odd case?
172      while (BLOCK_SIZE < elements_left) {
173          sum_value += data[i];
174          elements_left -= BLOCK_SIZE;
175          i += BLOCK_SIZE;
176      }
177      if(tid < elements_left) {
178          sum_value += data[i];
179      }
180
181      shared_data[tid] = sum_value;
182      workgroupBarrier();
183
```

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;
}
```

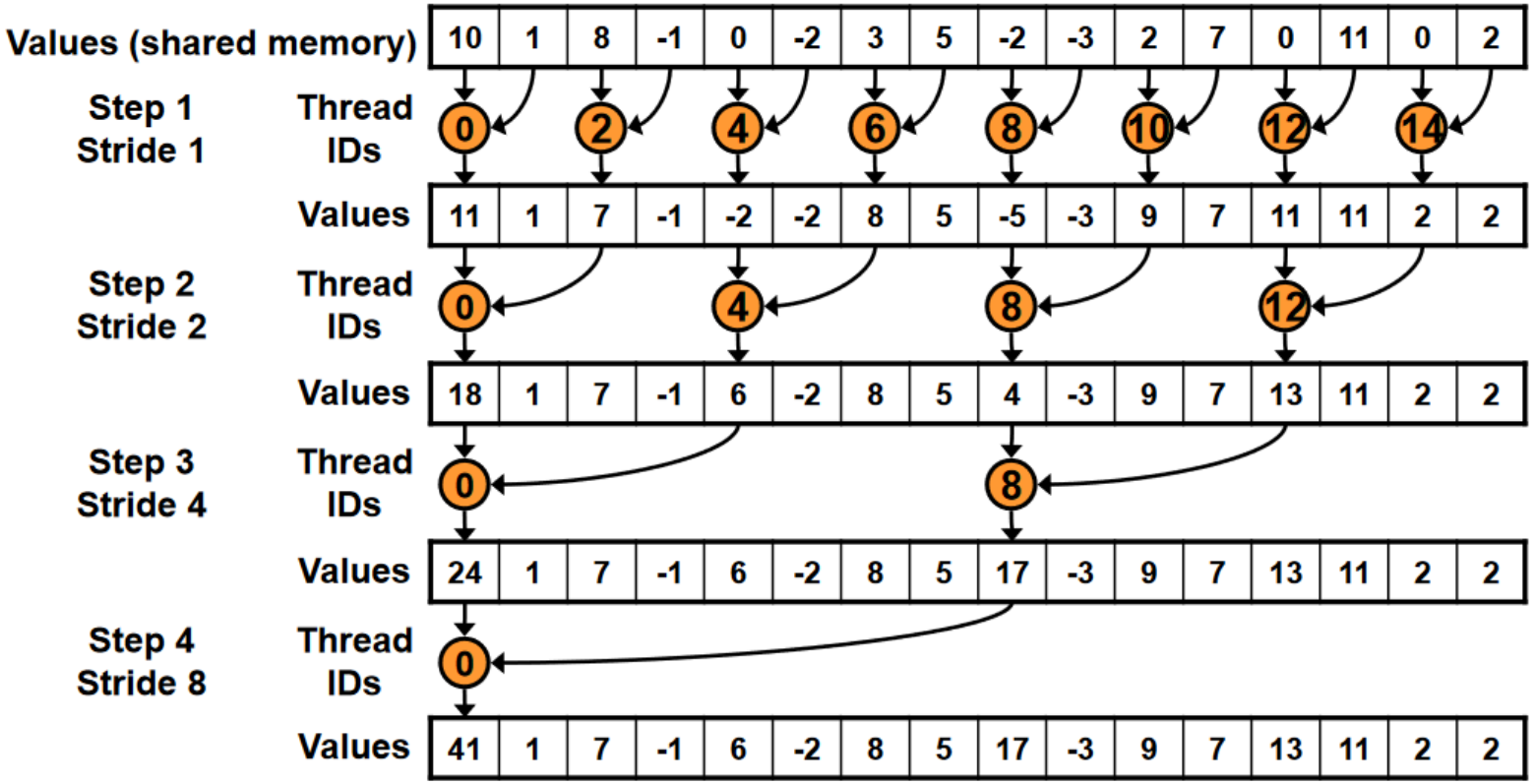
A Whirlwind Introduction to GPUs – The Next Level

Tree Reductions



A Whirlwind Introduction to GPUs – The Next Level

Tree Reductions



The Memory Hierarchy and the GPU

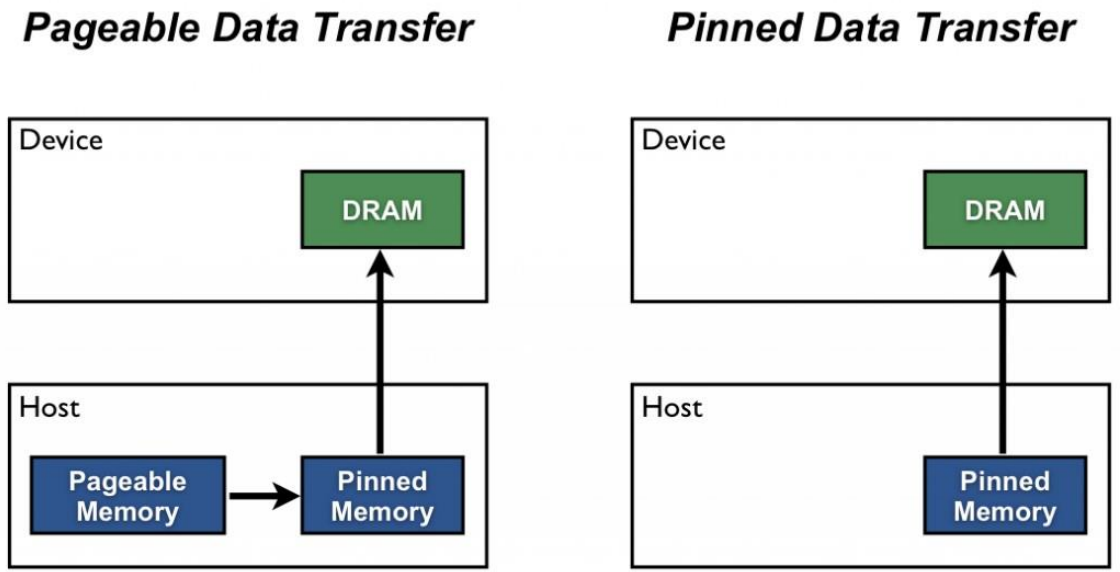
A memory hierarchy friend for your memory hierarchy

Integrated GPU’s – Lowering the cost of transfers

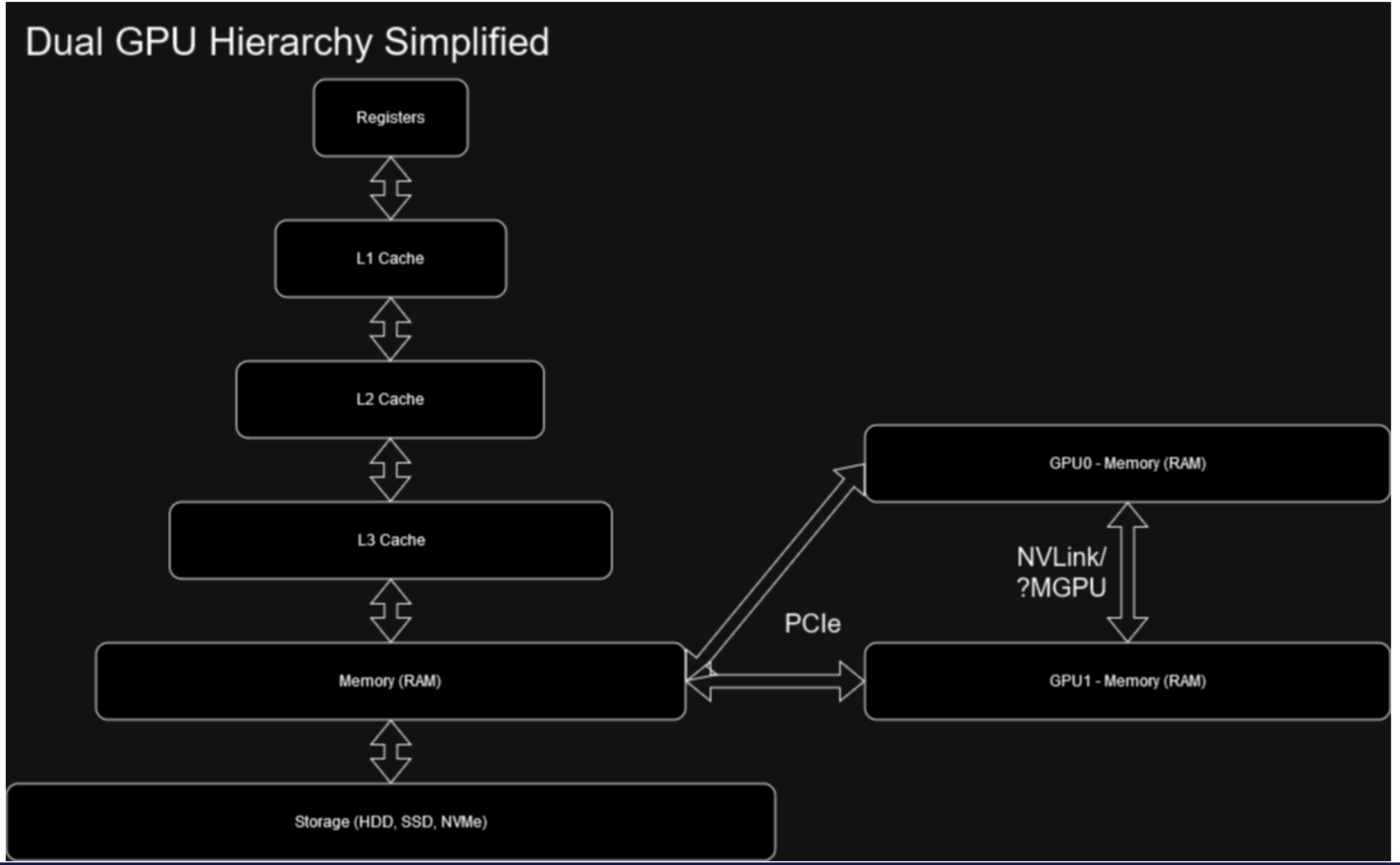
The Memory Hierarchy and the GPU - Transfers

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



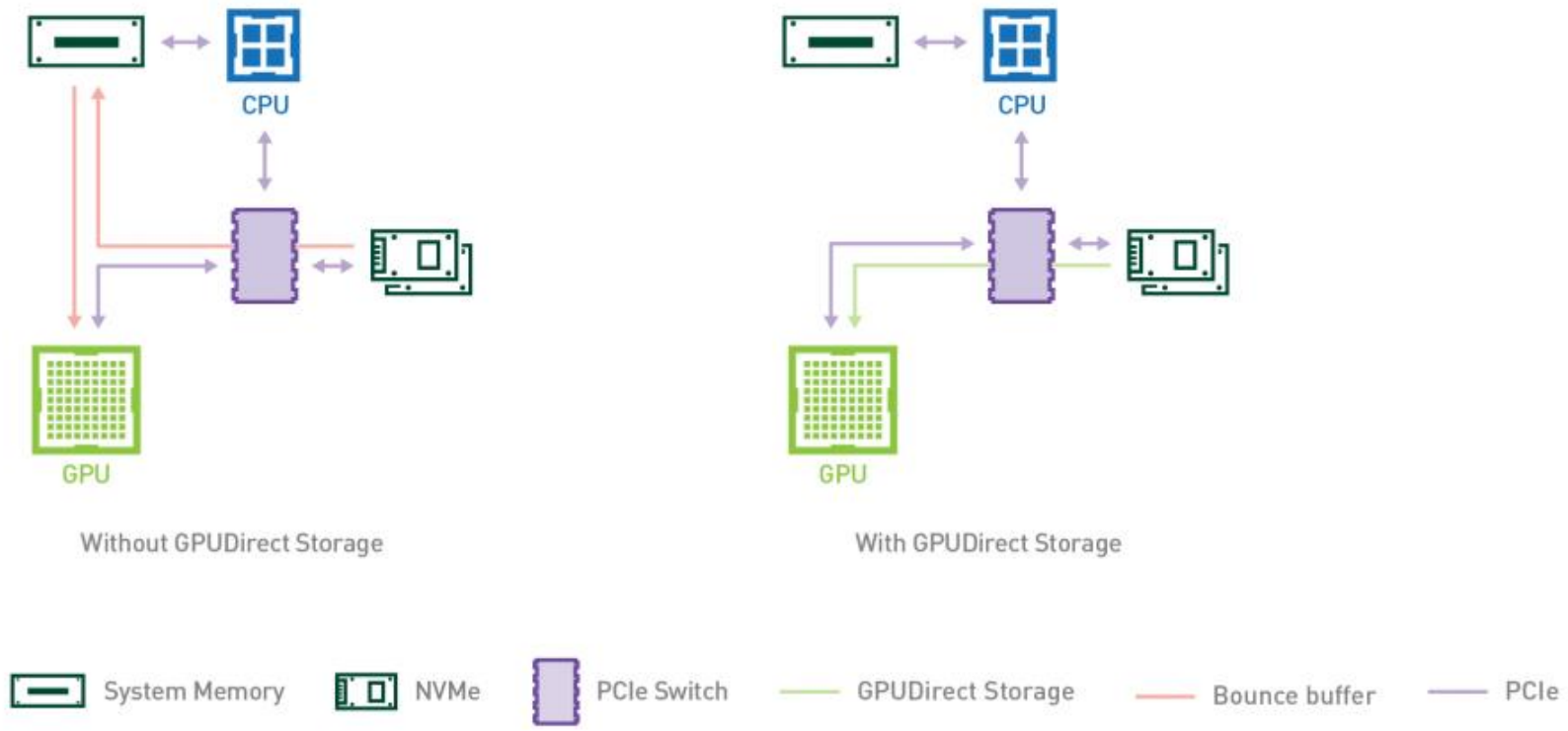
Dual GPU Hierarchy Simplified

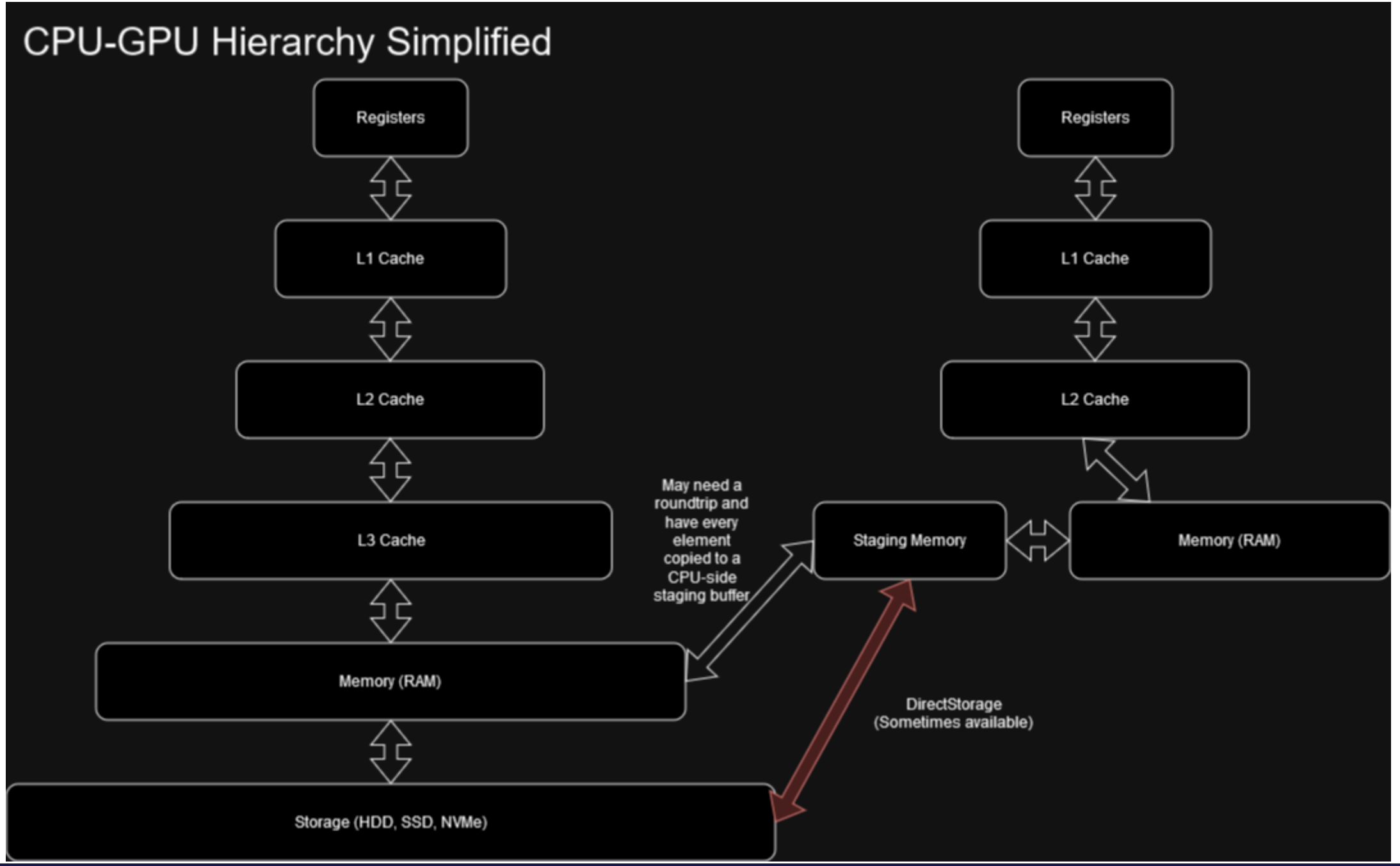


The Memory Hierarchy and the GPU

[GPUDirect]

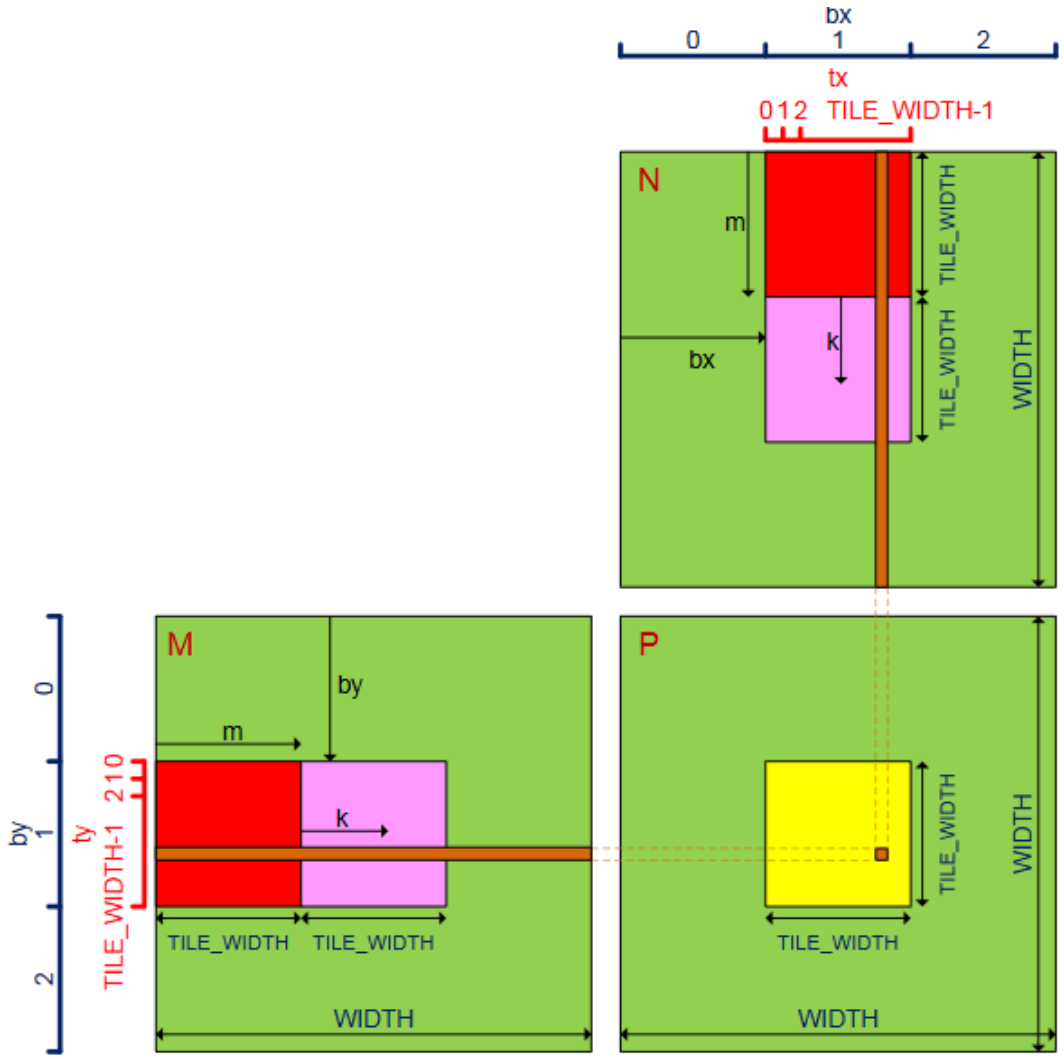
HPC systems love a few big files over lots of small ones



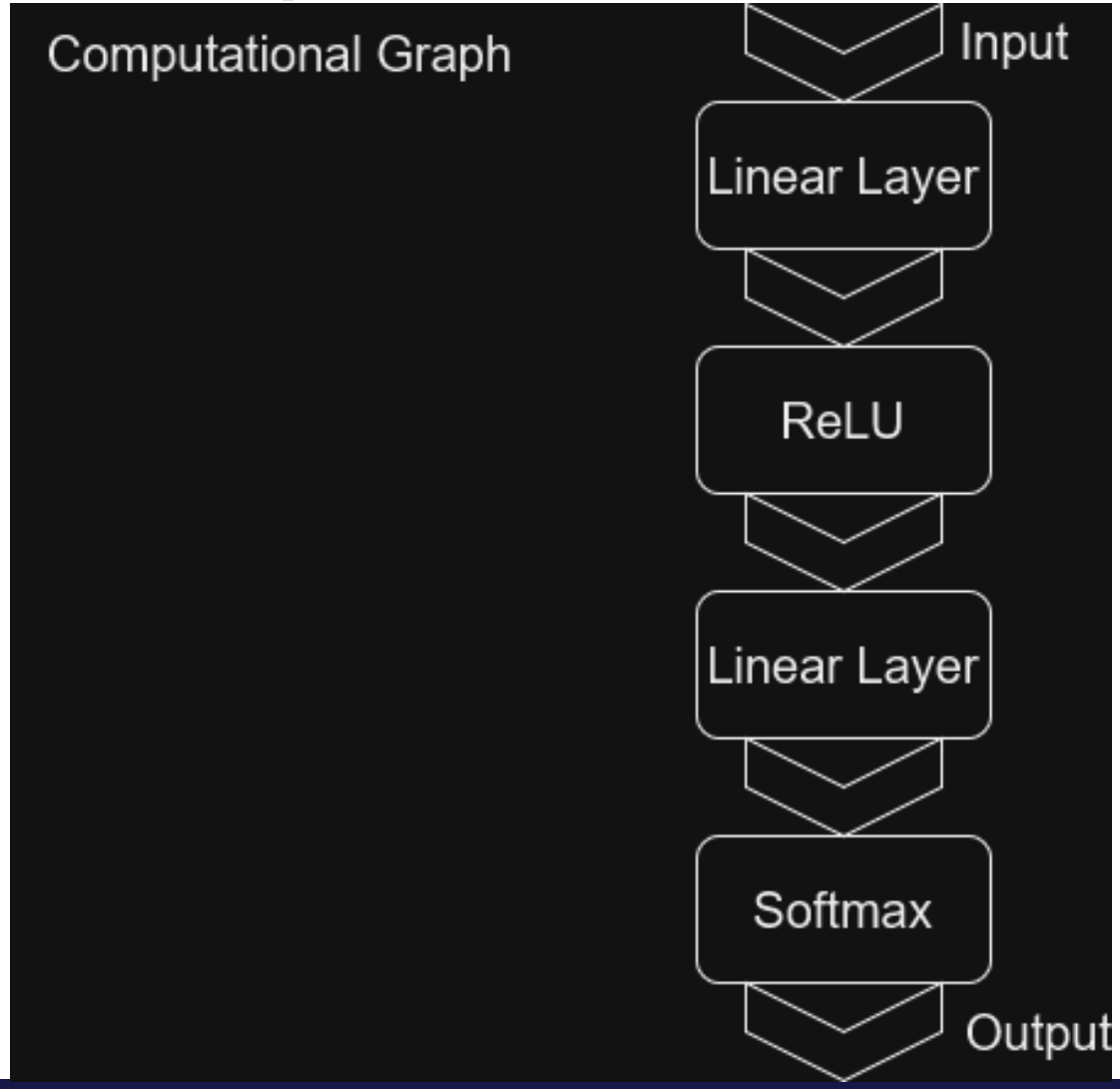

CPU-GPU Hierarchy Simplified
Registers
L1 Cache
L2 Cache
L3 Cache
Memory (RAM)
Storage (HDD, SSD, NVMe)
May need a roundtrip and have every element copied to a CPU-side staging buffer
Registers
L1 Cache
L2 Cache
Staging Memory
Memory (RAM)
DirectStorage
(Sometimes available)

A Whirlwind Introduction to GPUs – The Next Level

[Tiled Matrix Multiplication]



Back to the Computational Graph



Computational Graph Definition

```
159     let input: Tensor2D = Tensor2D::new(0.5, size, size);
160     let mut graph: Vec<GraphOperator> = vec![GraphOperator::HostToDevice { input }];
161
162     let mut rng: ChaCha8Rng = ChaCha8Rng::seed_from_u64((depth * size) as u64);
163     for _ in 0..depth {
164         let weights: Tensor2D = Tensor2D::new(0.5, size, size);
165         let bias: Tensor2D = Tensor2D::new(0.1, size, size);
166         graph.push(GraphOperator::LinearLayer { weights, bias });
167
168         let layer_type: usize = rng.gen_range(0..2);
169         if layer_type == 1 {
170             graph.push(GraphOperator::ReLU);
171         }
172     }
173     match graph[graph.len() - 1] {
174         GraphOperator::ReLU => {}
175         _ => graph.push(GraphOperator::ReLU),
176     };
177
178     graph.push(GraphOperator::Softmax);
179     graph.push(GraphOperator::DeviceToHost);
180
181     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

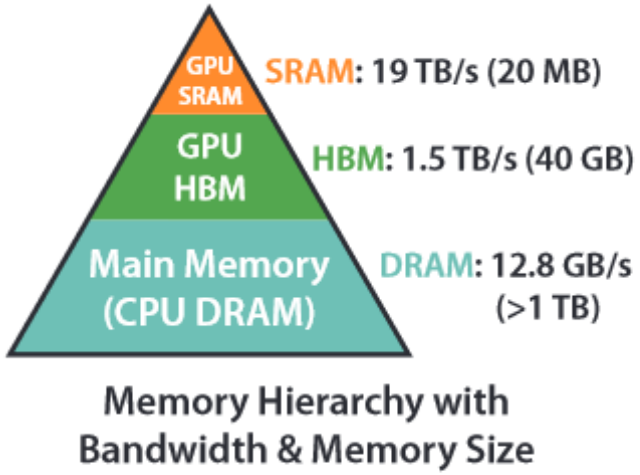
Or… op-code annotation and parsing?

Tying it all together

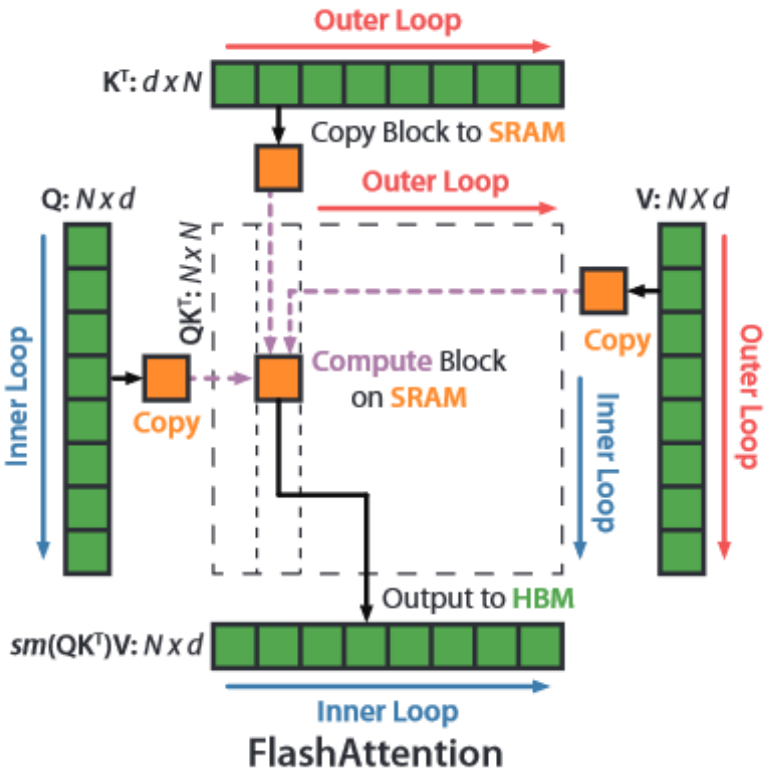
- CPU
- Immediate GPU
- GPU Graph
- GPU Graph in a loop

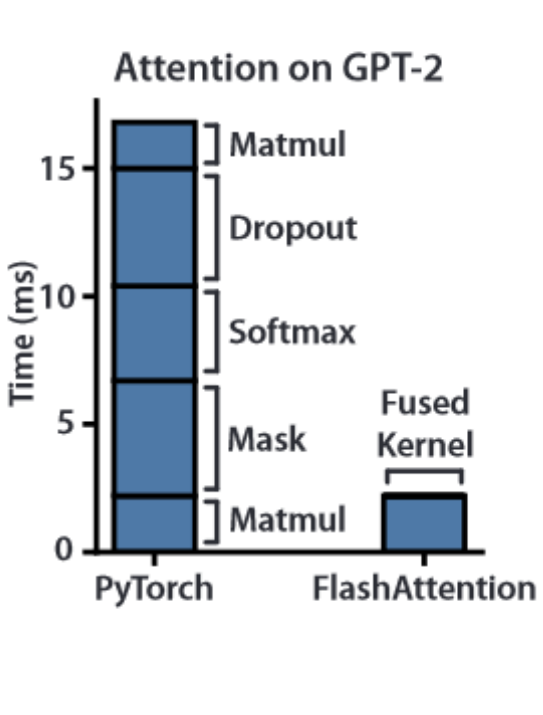
Tying it all together

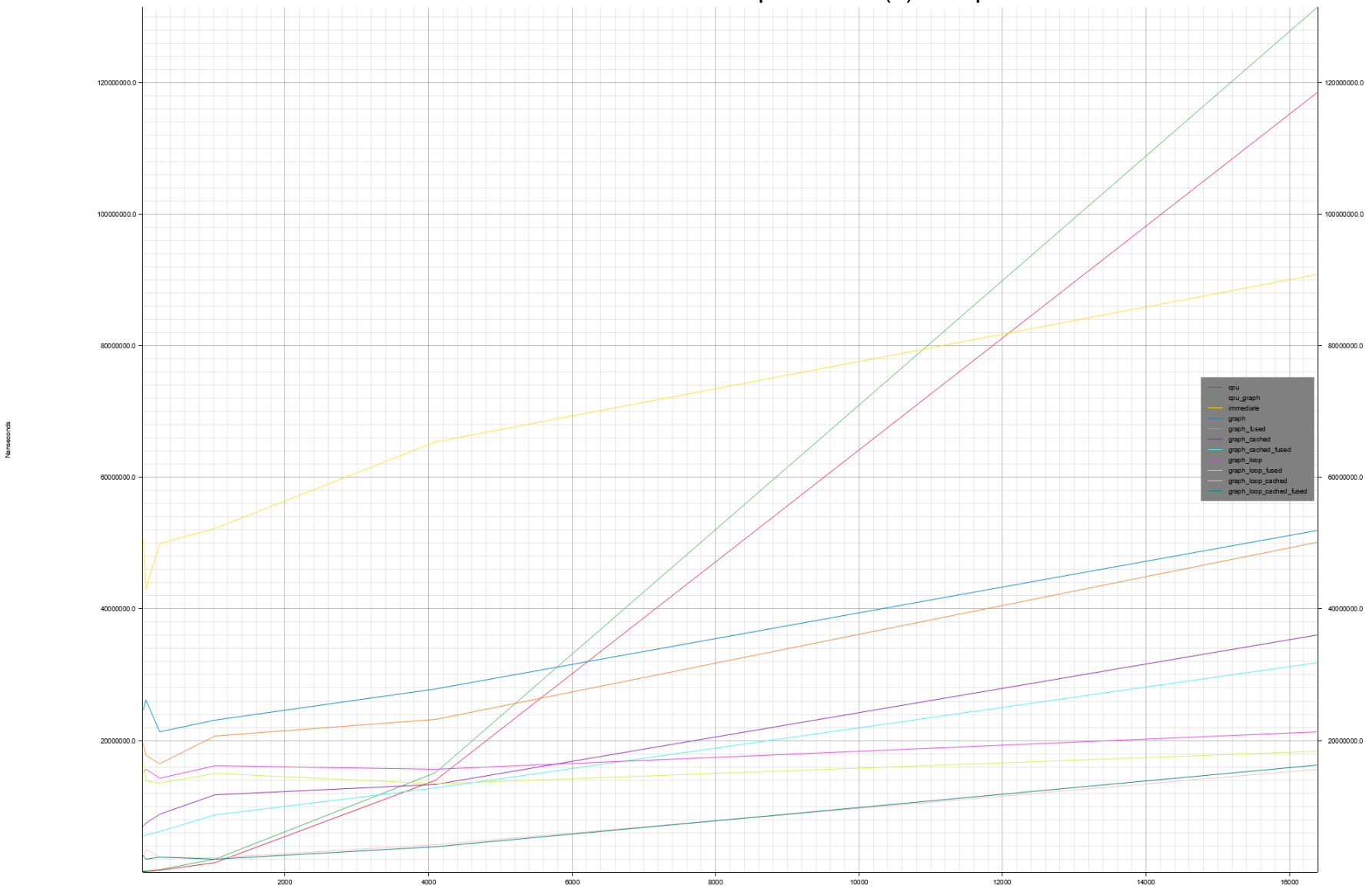
- Tiled Matrix Multiplication and Kernel Fusion are the main drivers of Flash Attention

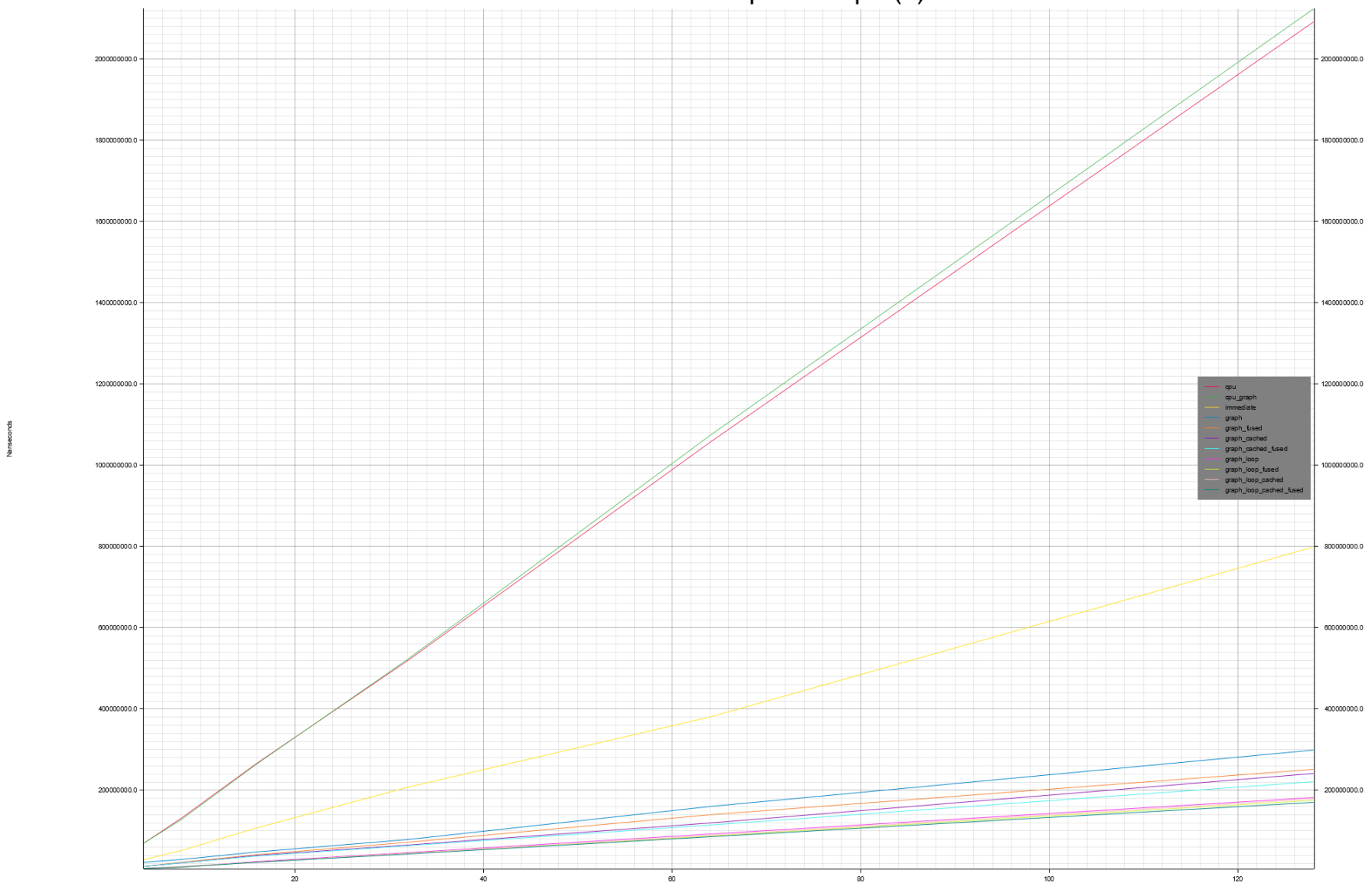


Memory Hierarchy with Bandwidth & Memory Size






Benchmark - Graphs - Size(x) - Depth 64
Nanoseconds
Element Count
120000000.0
100000000.0
80000000.0
60000000.0
40000000.0
20000000.0
2000
4000
6000
8000
10000
12000
14000
16000
cpu
cpu_graph
immediate
graph
graph_fused
graph_cached
graph_cached_fused
graph_loop
graph_loop_fused
graph_loop_cached
graph_loop_cached_fused


Benchmark - Graphs - Depth(x) - Size 256
Nanoseconds
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

Exercises/Hand-in

You will find the hand-in template in **m1_memory_hierarchies::code::gpu_hand_in**

Go through the code. The vector addition program will be there as a reference.

1. Create the GitHub repo for your hand-ins, make it private
2. Invite me to your repo
3. Commit the gpu_hand_in project, with no changes from your side as a baseline
4. Complete the tasks described in the convolution.rs and matrix_multiplication.rs files
5. Make sure your code is correct before optimizing
6. When optimizing, ensure that your code is correct every step of the way
7. Latest deadline is when you hand in your project, you are free to do the hand-ins in the order that works best for you, I will introduce the second hand-in next week