Graphics Architectures and OpenCL

Michael Doggett
Department of Computer Science
Lund university

2024

Lectures

- Next Monday
  - 1st - Advanced Graphics Summit: Raytracing in Snowdrop: An Optimized Lighting Pipeline for Consoles
    - Quentin Kuenlin, Massive Entertainment
    - For the game, Avatar: Frontiers of Pandora
- 2nd - Vulkan, Animations and the Ray-Tracing pipeline
  - Gustaf Waldemarson, ARM

Overview

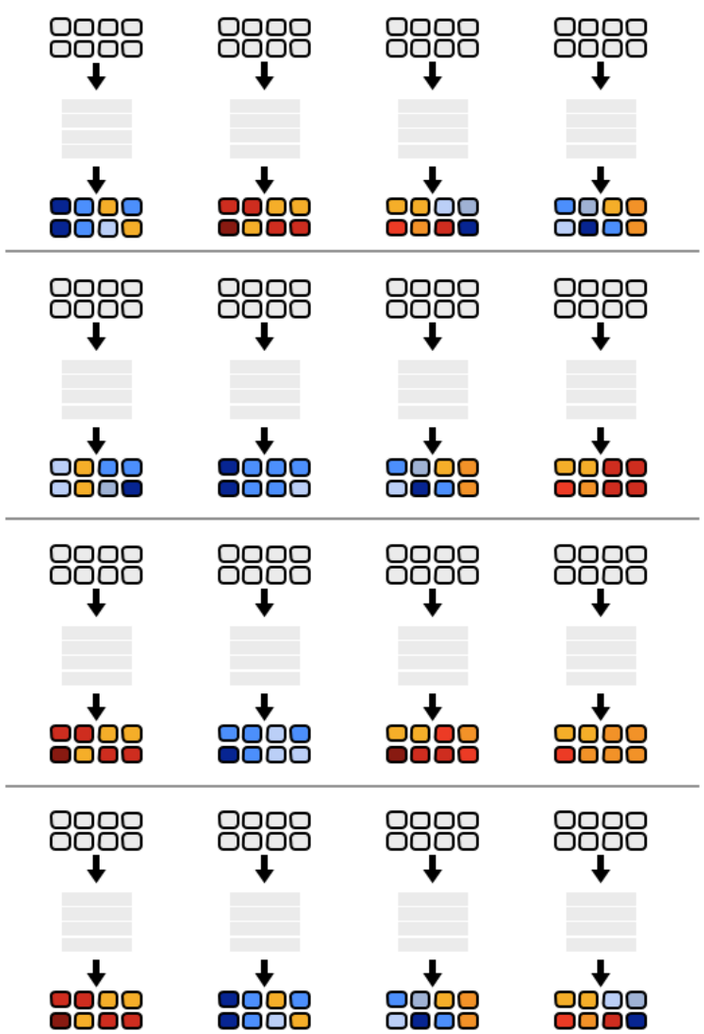
- Parallelism
- GPU Architecture - Radeon 5870
- Tiled Graphics Architectures
  - Important when Memory and Bandwidth limited
  - Different to Tiled Rasterization!
- Tessellation
- OpenCL
  - Programming the GPU without Graphics

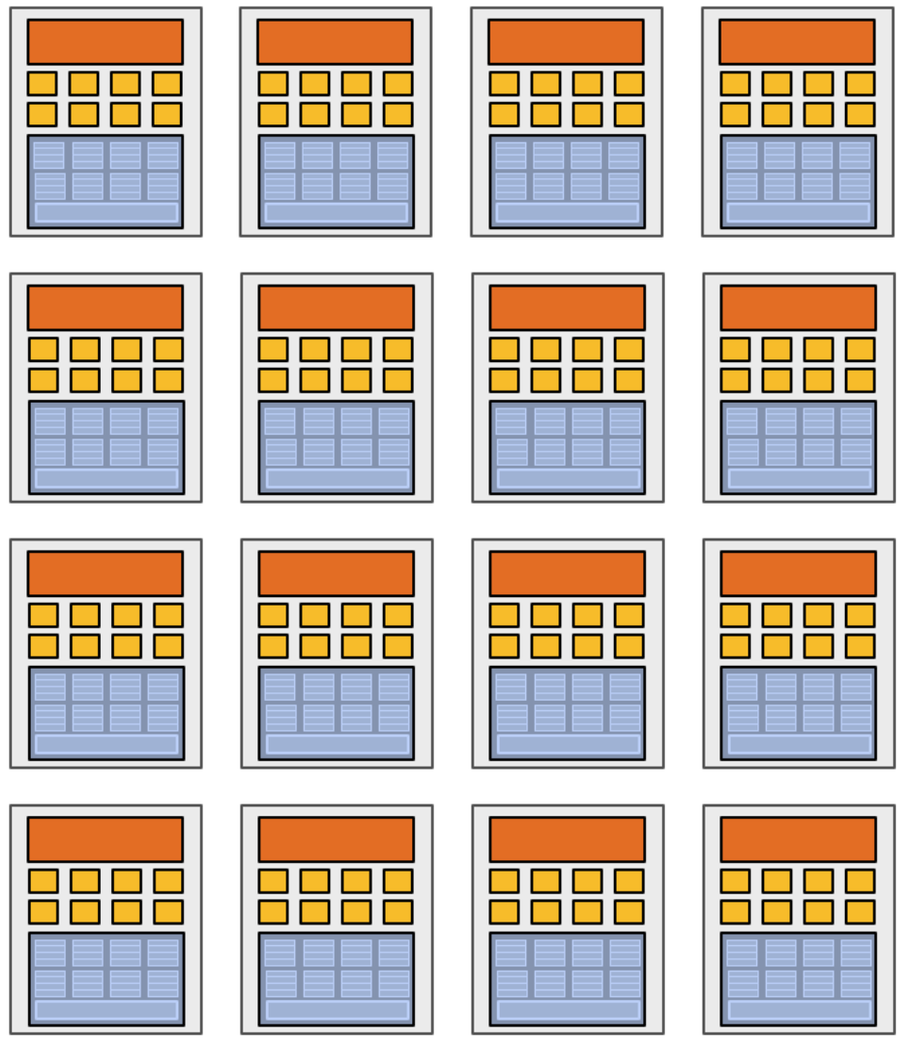
"Only 10% of our pixels require lots of samples for soft shadows, but determining which 10% is slower than always doing the samples. " by ID_AA_Carmack, Twitter, 2011-10-17

Parallelism

- GPUs do a lot of work in parallel
- Pipelining, SIMD and MIMD
- What work are they doing?

What's running on 16 unified shaders?
128 fragments in parallel





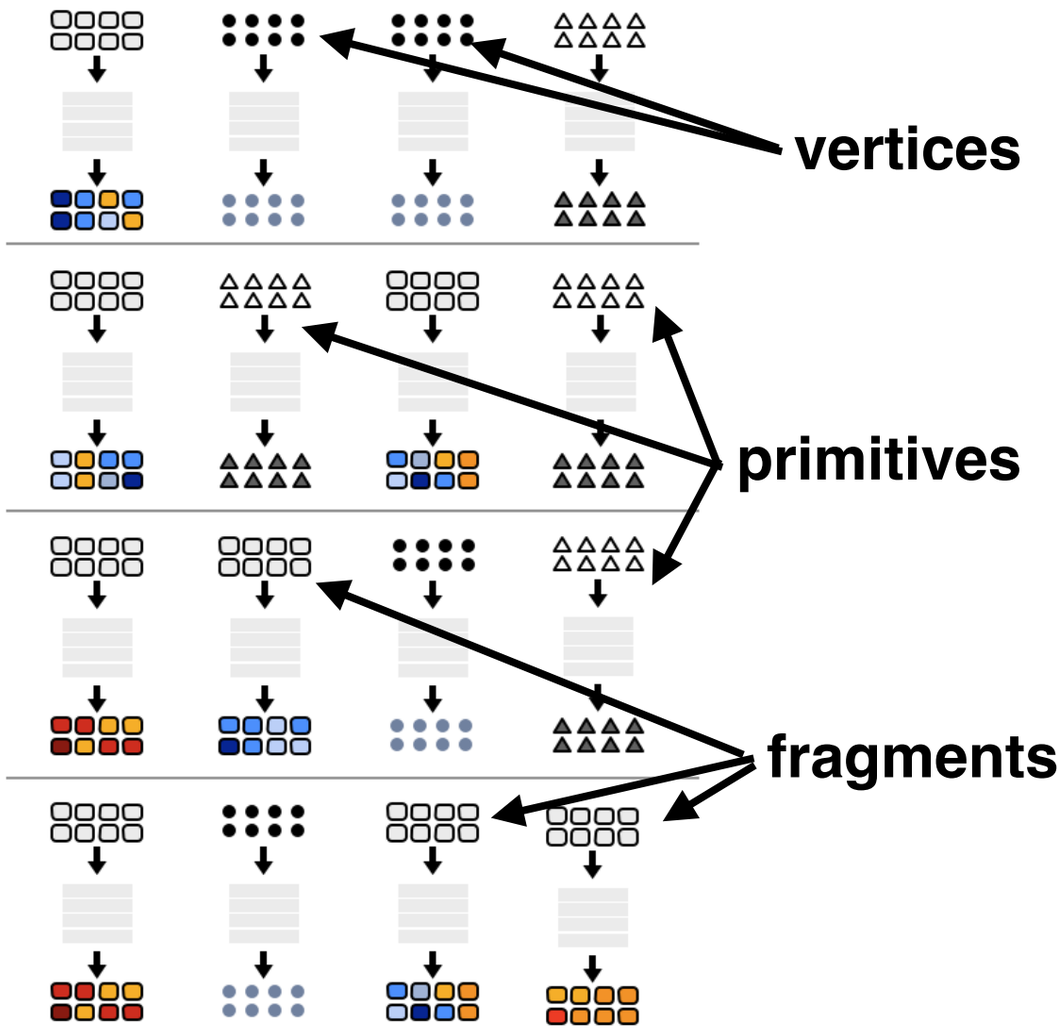
16 cores = 128 ALUs

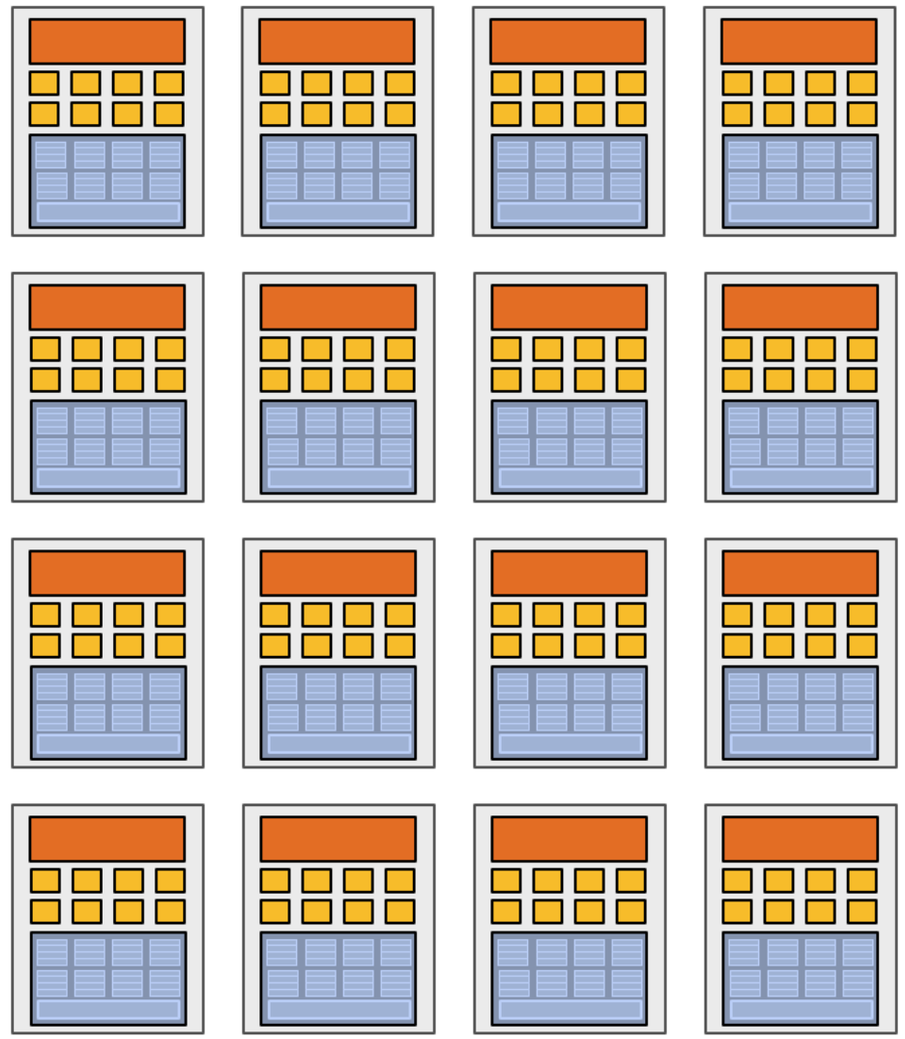
MIMD

16 simultaneous instruction streams

Slide courtesy Kayvon Fatahalian

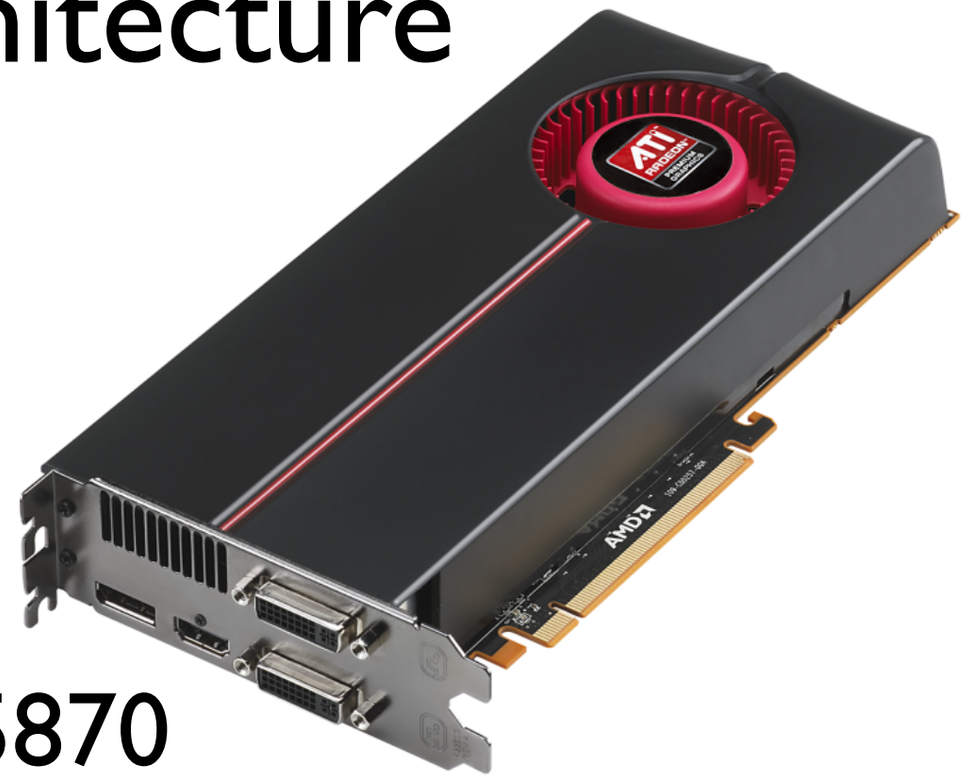
128 [vertices/fragments
primitives
OpenCL work items
CUDA threads] in parallel





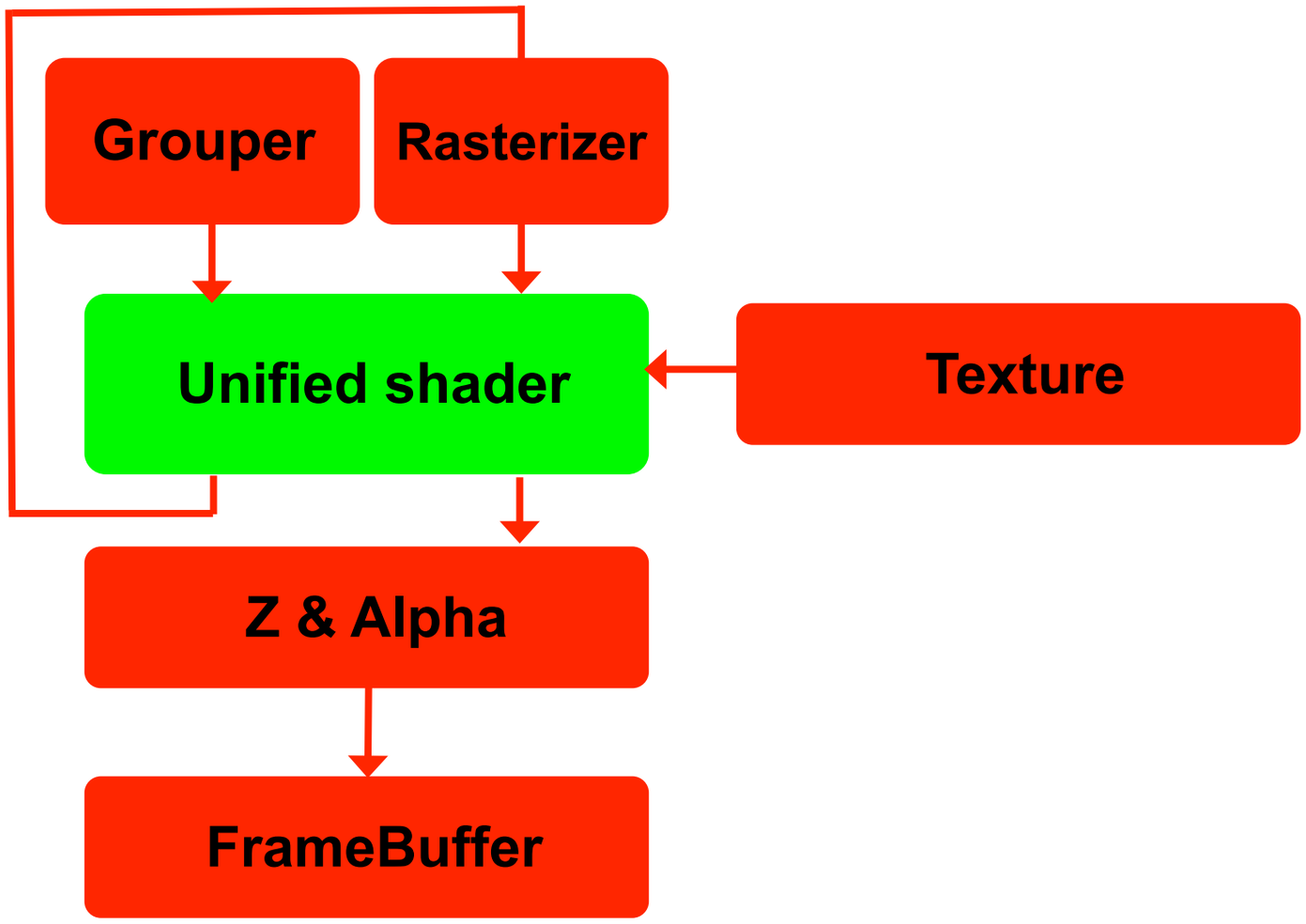
Slide courtesy Kayvon Fatahalian

Unified Shader Architecture



- Let's take the ATI Radeon 5870
  - From 2009
- What are the components of the programmable graphics hardware pipeline?

Unified Shader Architecture



GPU Architecture

ATI Radeon 5870

Shader Inputs

Unified Shader

Memory, Z & Alpha

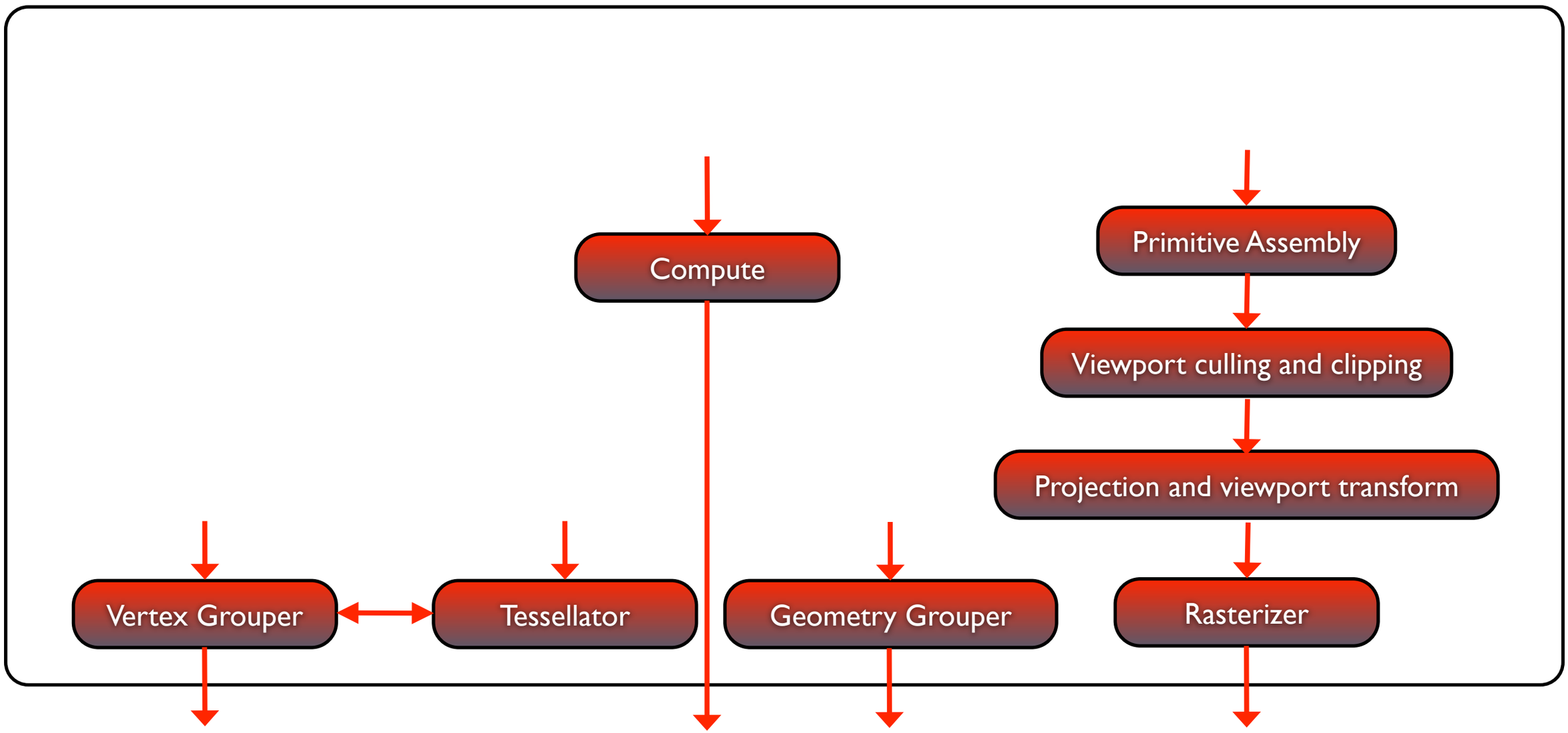
FrameBuffer

Shader Inputs

Unified Shader

Memory, Z & Alpha

Shader Inputs



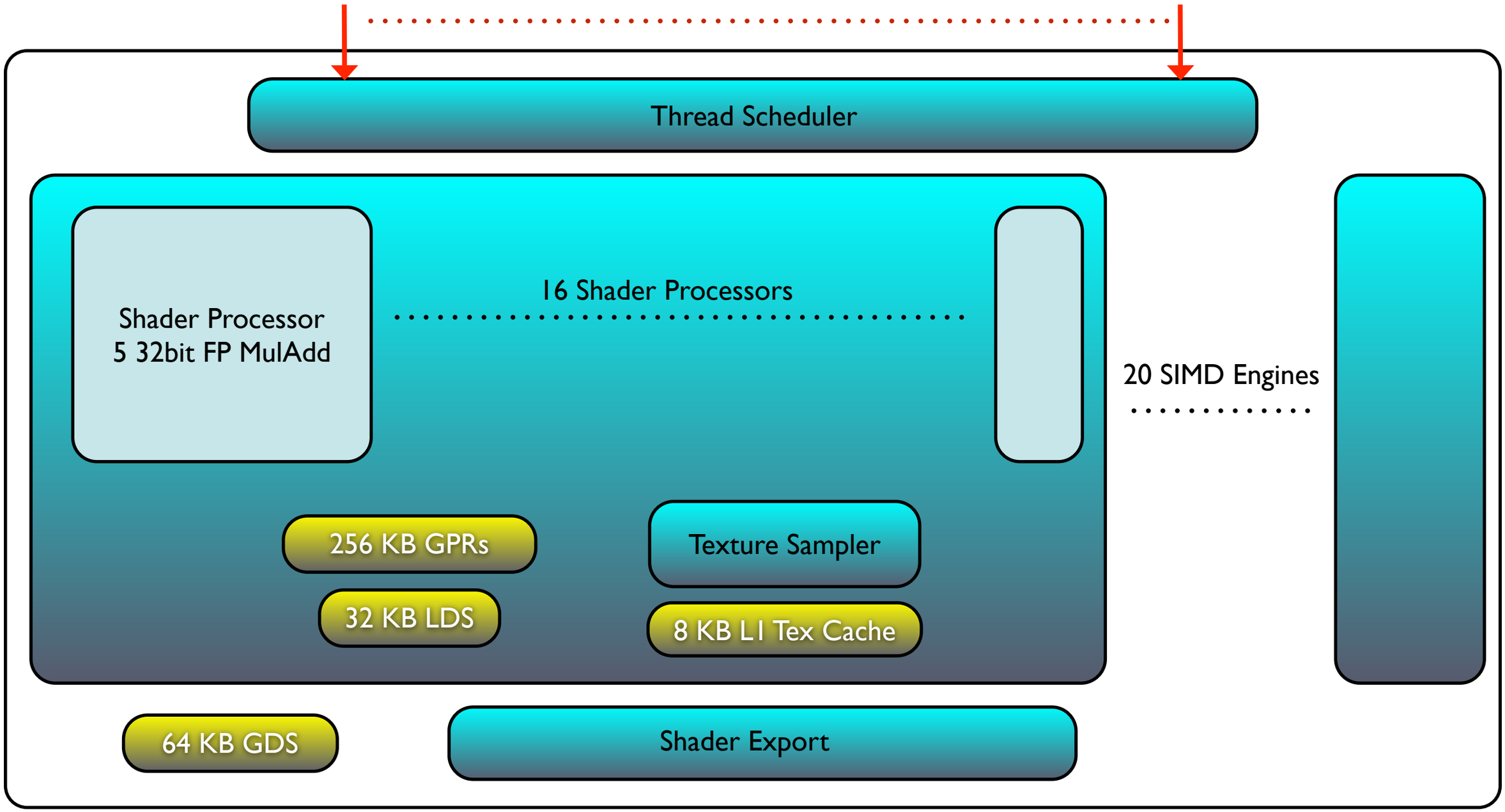
ATI Radeon 5870

Shader Inputs

Unified Shader

Memory, Z & Alpha

Unified Shader



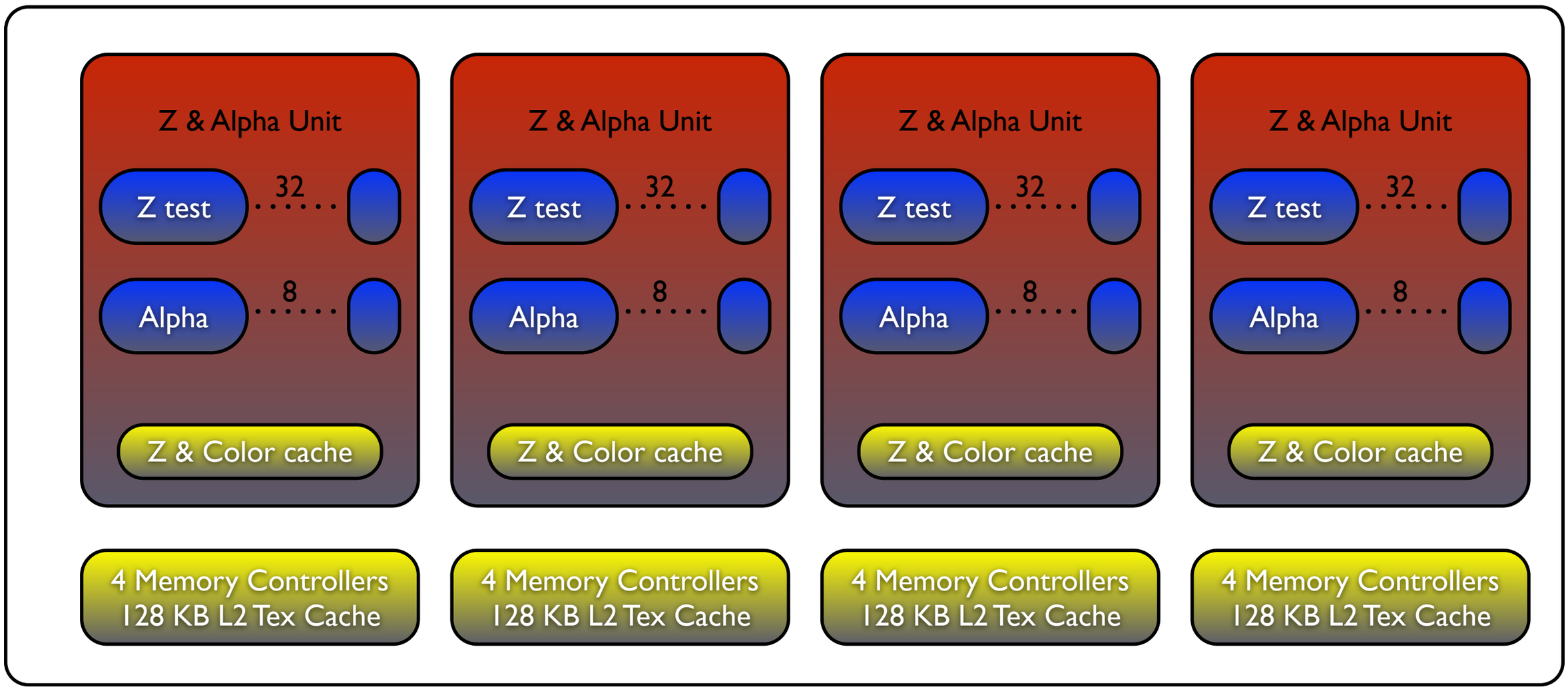
ATI Radeon 5870

Shader Inputs

Unified Shader

Memory, Z & Alpha

Memory, Z & Alpha



ATI Radeon 5870

Tiled Based Architectures

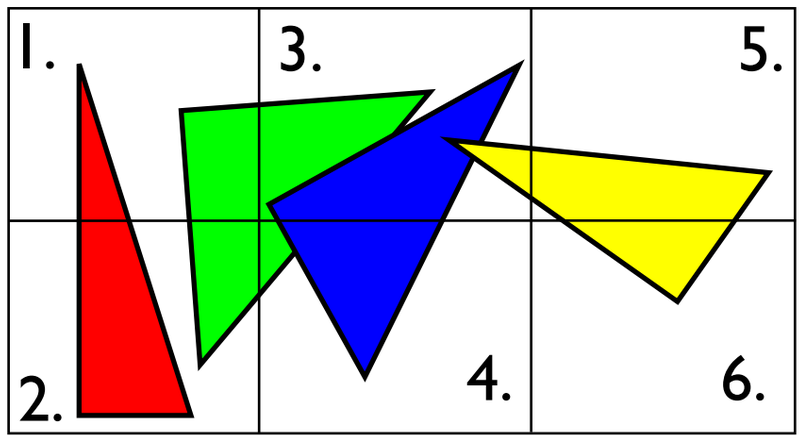
- There are two major styles of GPU architecture :
  - Traditional
    - One we work with and discuss
    - Dominant in Desktop (Nvidia, AMD, Intel)
  - Tiling based
    - Dominant in Mobile (ARM Mali, Qualcomm Adreno, Apple Ax)
    - Different to tile based rasterisation

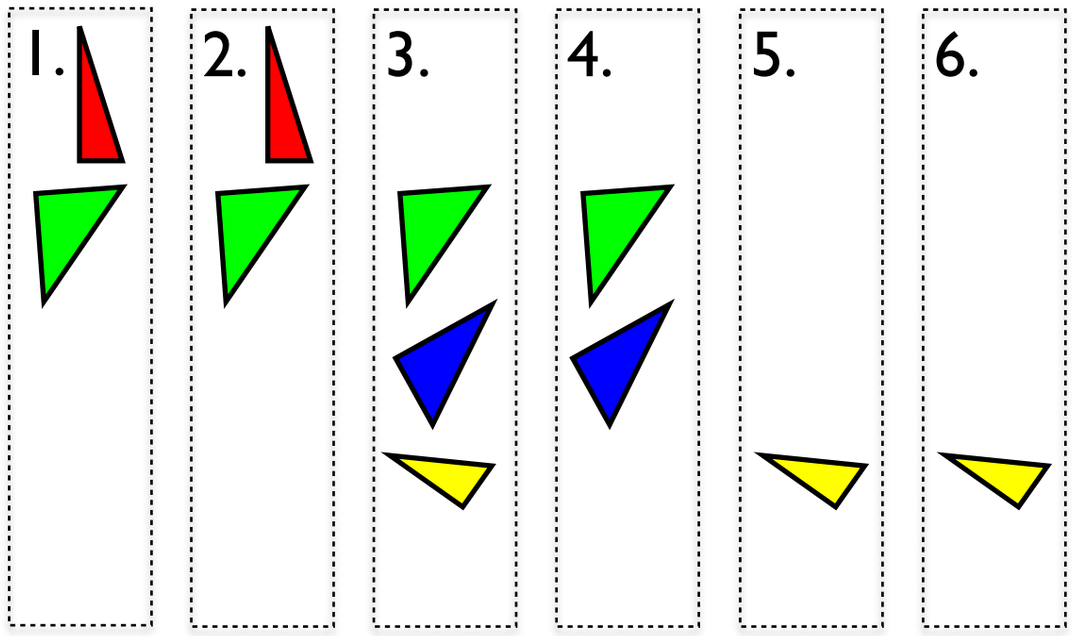
Examples of Tiled Based Architectures

- ARM Mali (Mobile)
- Imagination Technologies
  - Kyro II (PC graphics)
  - Sega Dreamcast (console)
  - PowerVR (Mobile)
  - Apple GPU
- Other notable ones
  - Original Intel Larrabee used a Software Tiled based rasterizer
    - (became Xeon Phi)

Tiled Based Rendering

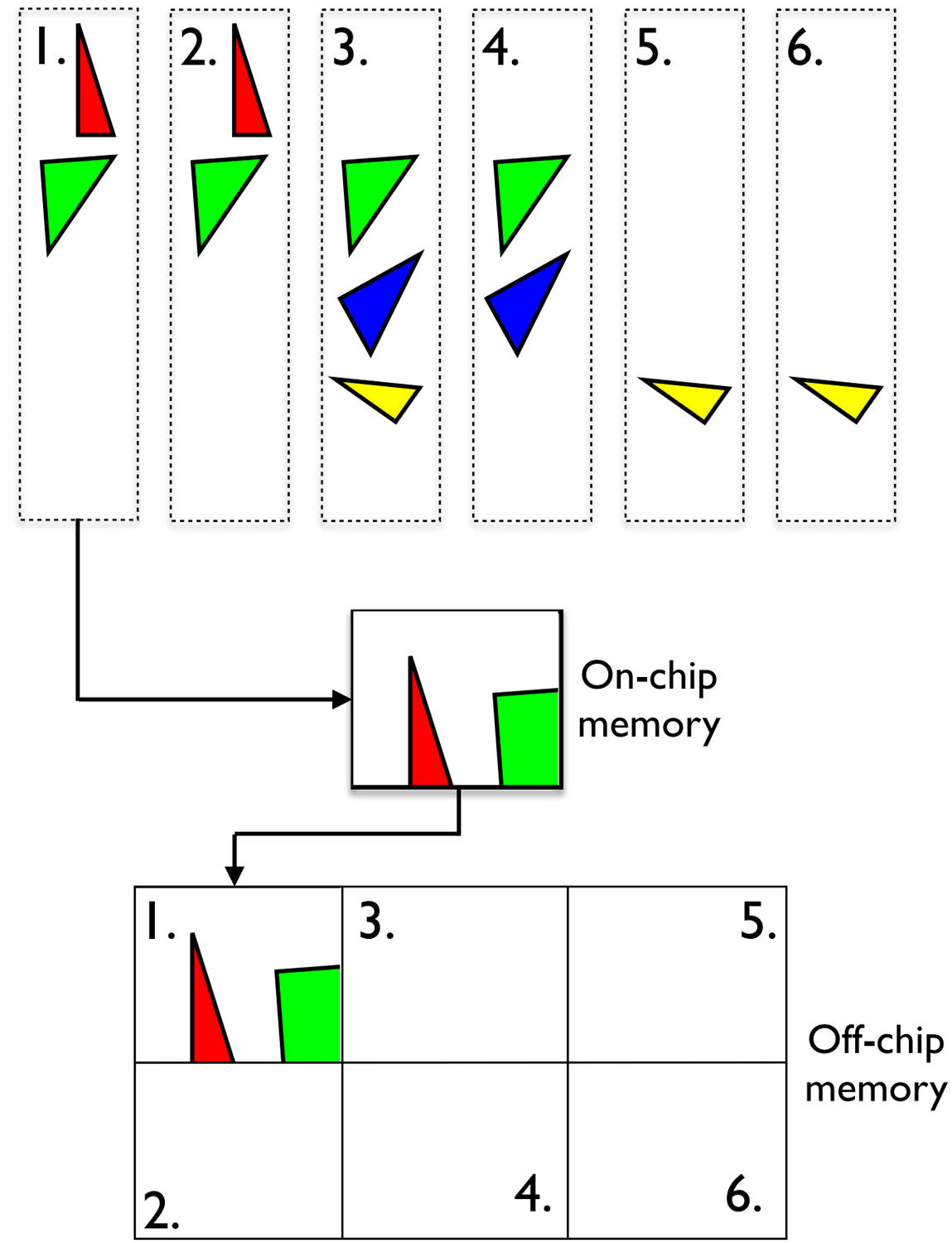
- Screen is divided into Tiles (e.g. 32x32 pixels)
- Run vertex shader to project triangles to the screen
- Create Tile Triangle Lists (or Bins)
- Each list contains pointers to all triangles in the tile





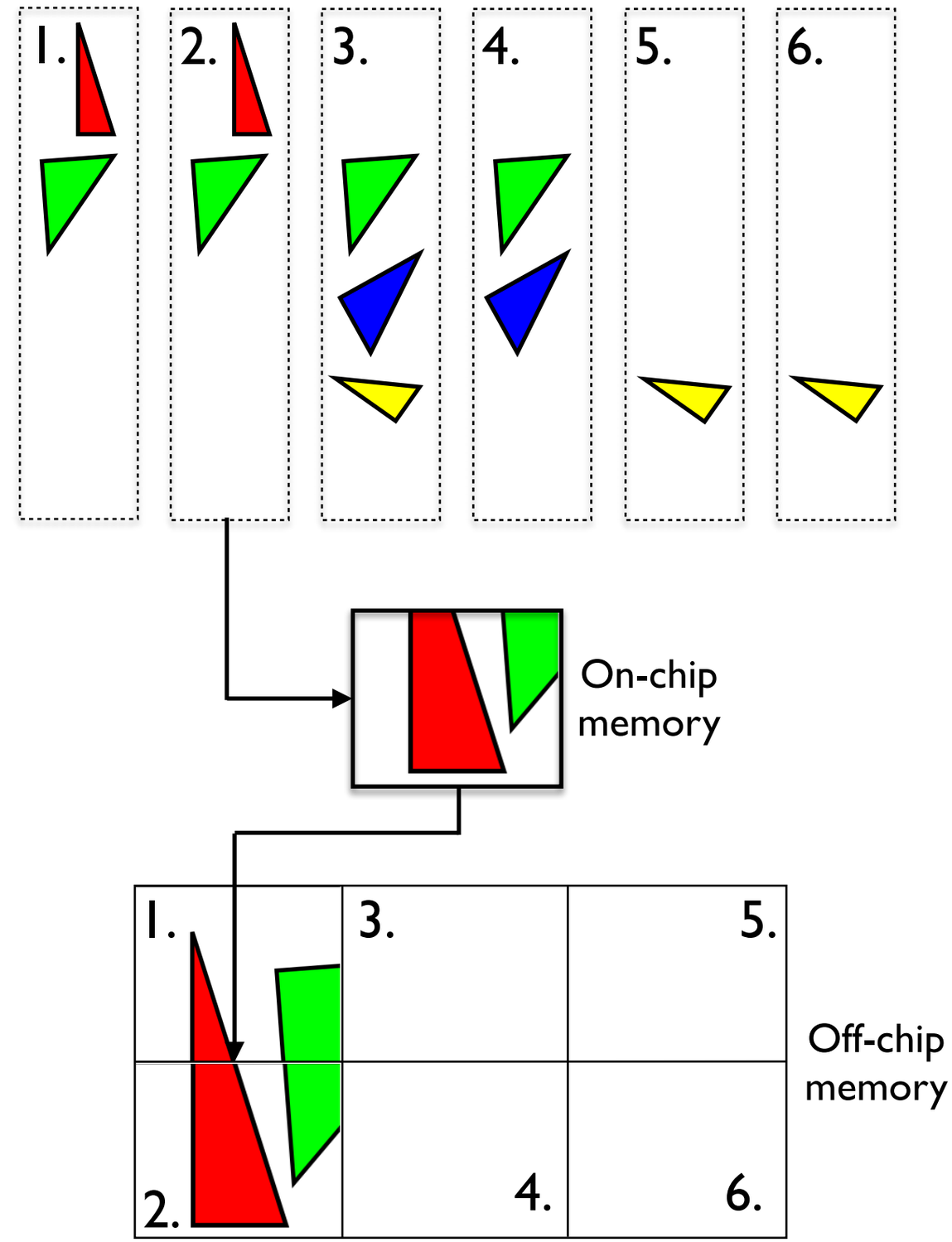
Tiled Based Rendering

- Take one tile and render all triangles in that tile list
- Use 1 tile sized on-chip memory
- After processing all triangles in the list
  - Copy tile memory, with Z and Color, from on-chip memory to main memory



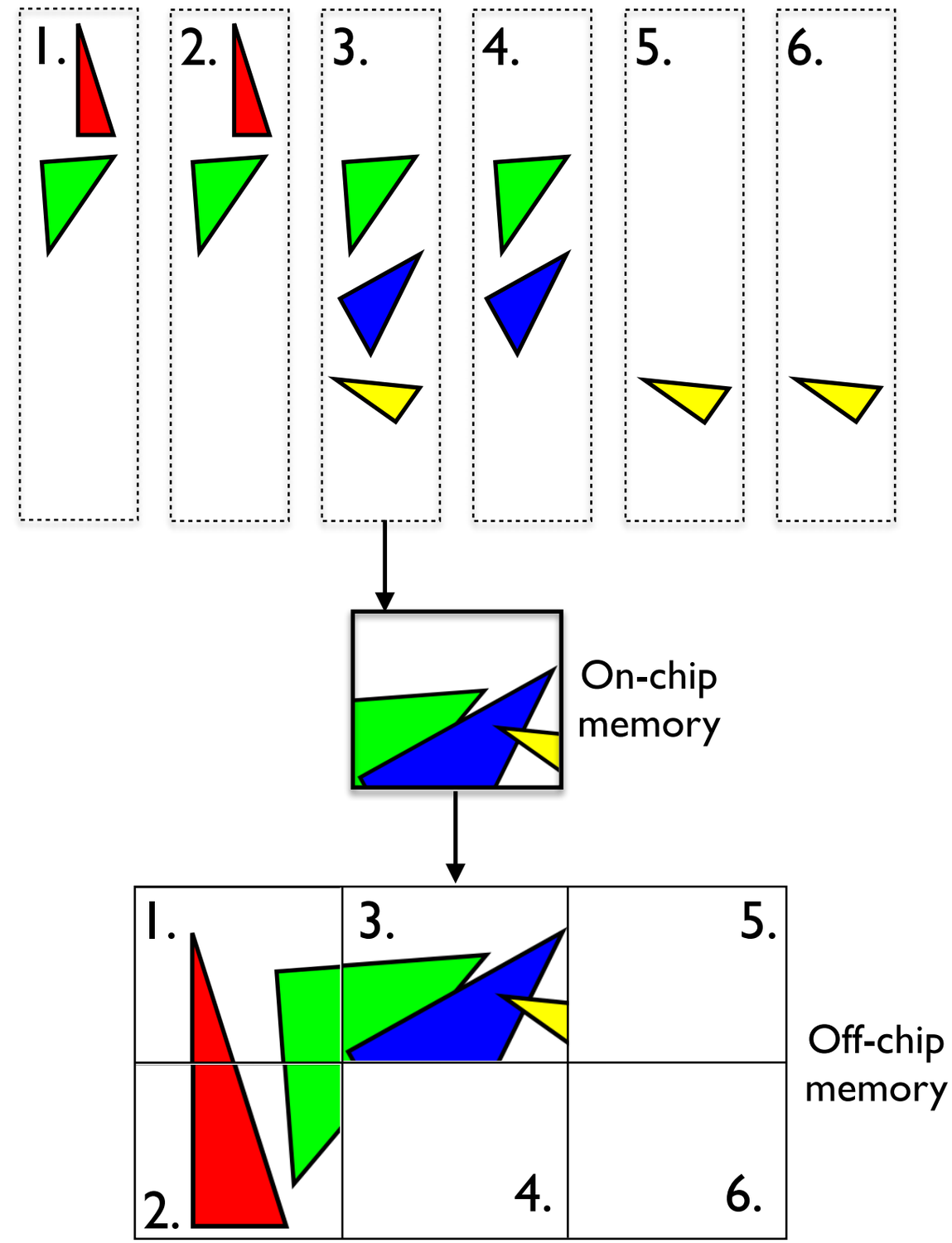
Tiled Based Rendering

- Take one tile and render all triangles in that tile list
- Use 1 tile sized on-chip memory
- After processing all triangles in the list
  - Copy tile memory, with Z and Color, from on-chip memory to main memory



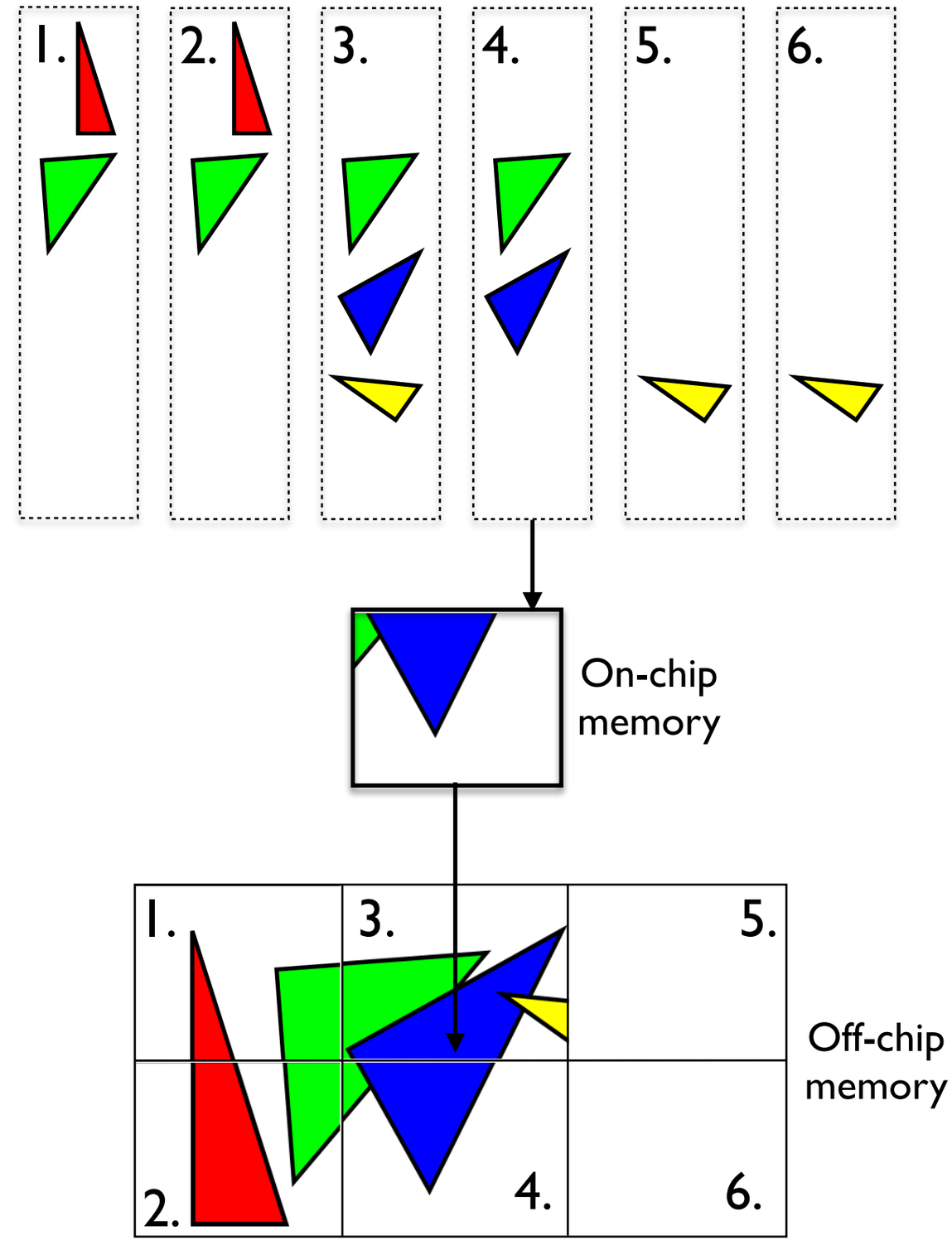
Tiled Based Rendering

- Take one tile and render all triangles in that tile list
- Use 1 tile sized on-chip memory
- After processing all triangles in the list
  - Copy tile memory, with Z and Color, from on-chip memory to main memory



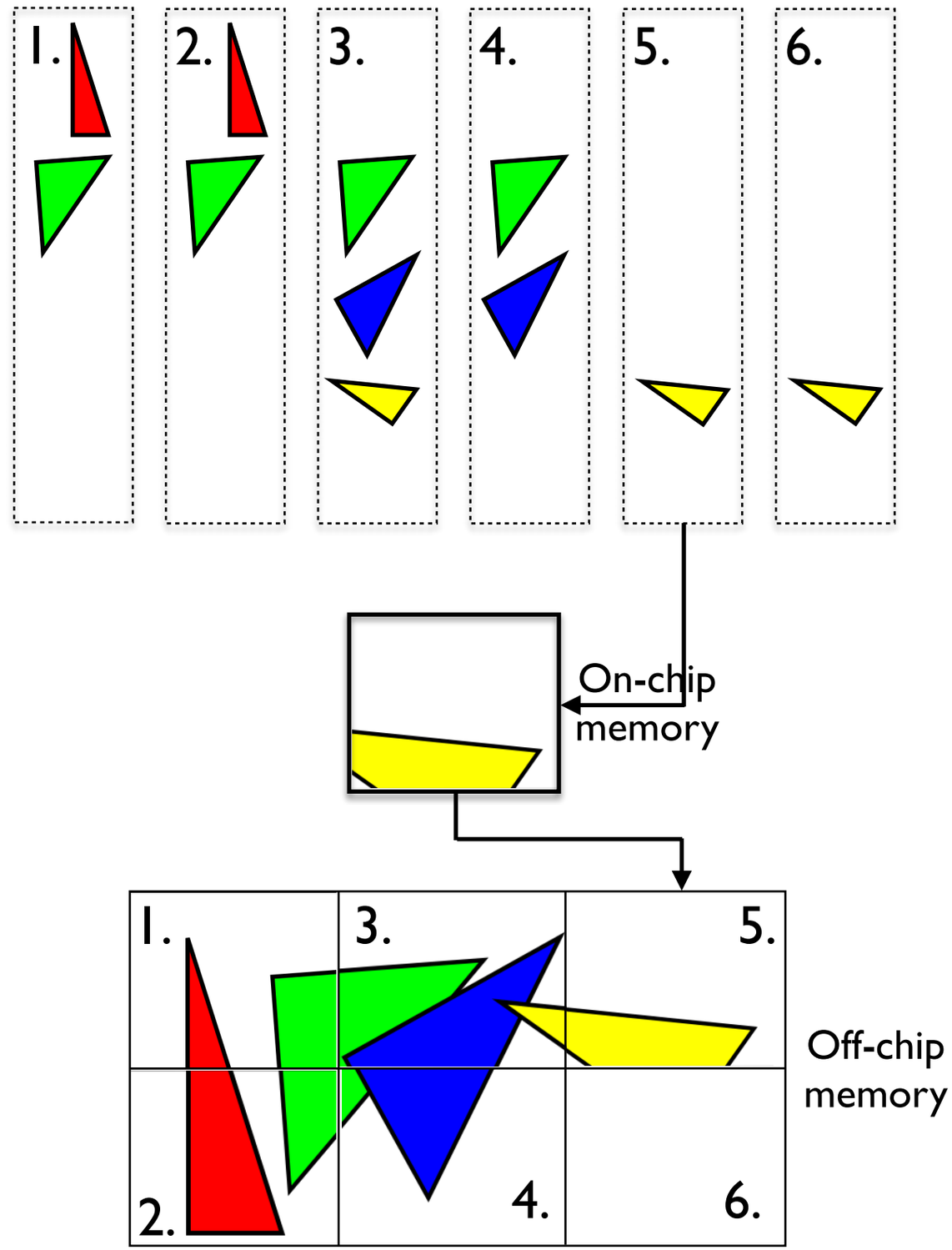
Tiled Based Rendering

- Take one tile and render all triangles in that tile list
- Use 1 tile sized on-chip memory
- After processing all triangles in the list
  - Copy tile memory, with Z and Color, from on-chip memory to main memory



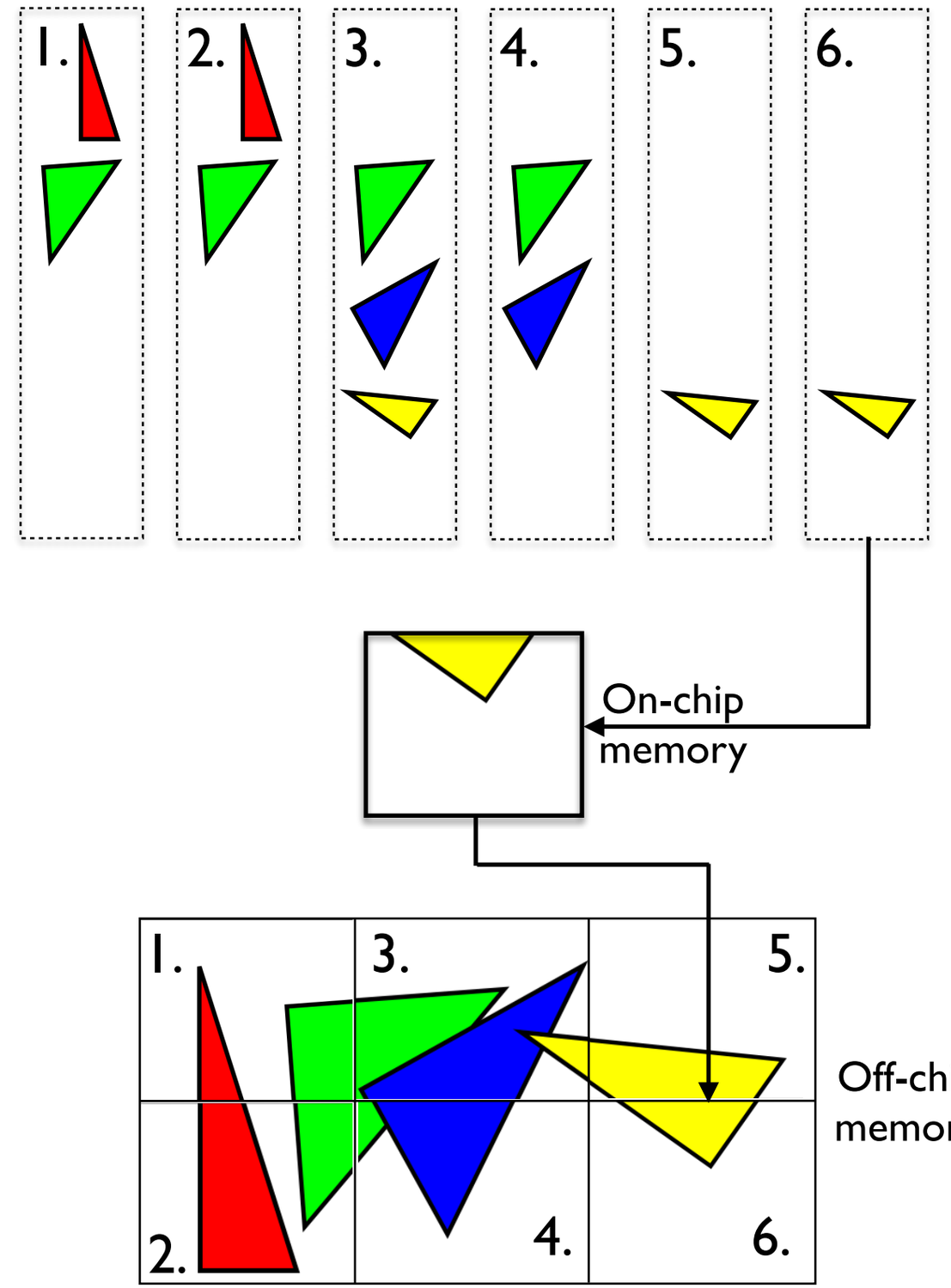
Tiled Based Rendering

- Take one tile and render all triangles in that tile list
- Use 1 tile sized on-chip memory
- After processing all triangles in the list
  - Copy tile memory, with Z and Color, from on-chip memory to main memory



Tiled Based Rendering

- Take one tile and render all triangles in that tile list
- Use 1 tile sized on-chip memory
- After processing all triangles in the list
  - Copy tile memory, with Z and Color, from on-chip memory to main memory



Mali Bifrost GPU

- Hierarchical tiling unit - 16x16, 32x32, 64x64 pixel sized bins
- Switch to Thread Level Parallelism (TLP) quad execution design
  - Better for compute

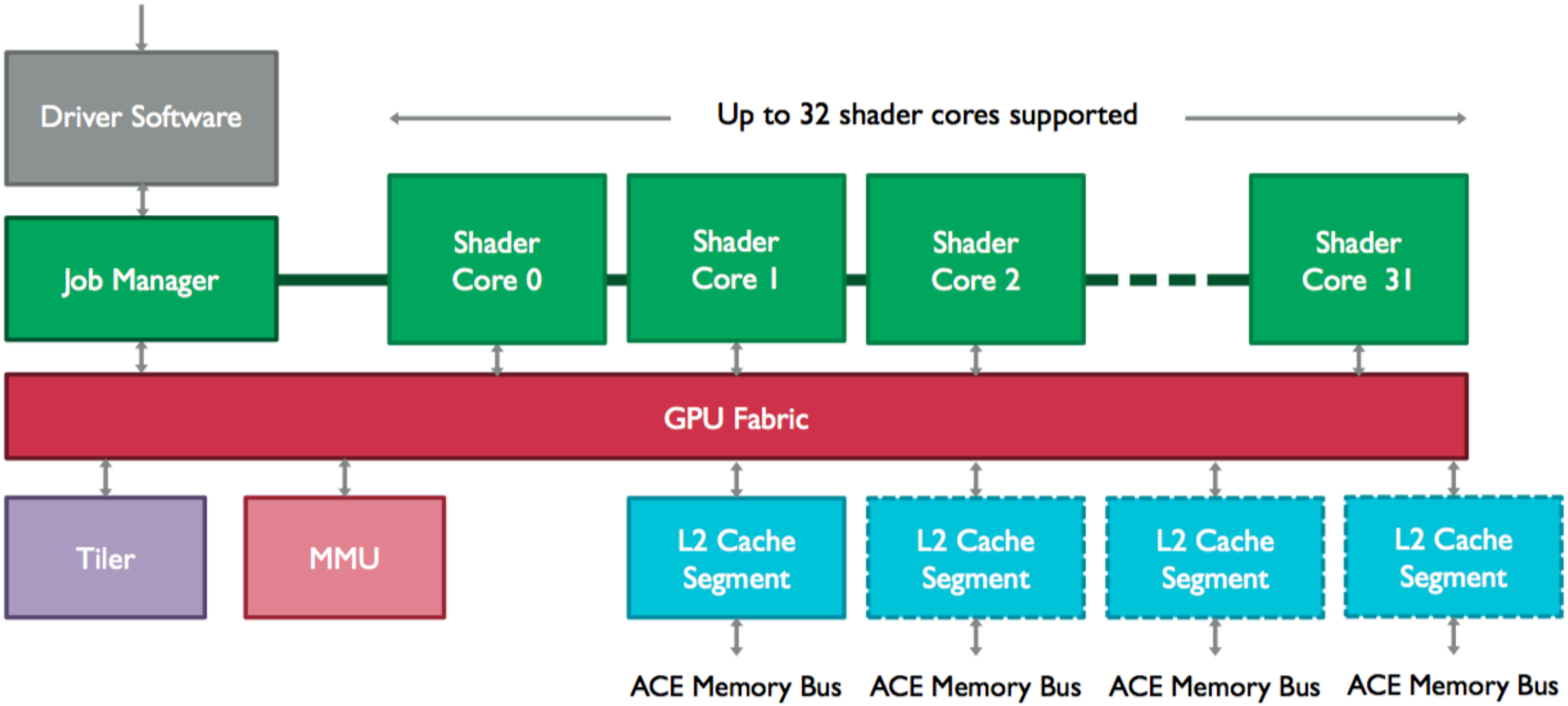


image courtesy Jem Davies,[23] "Bifrost GPU", 2016, © ARM

Mali-G71 Shader Core design

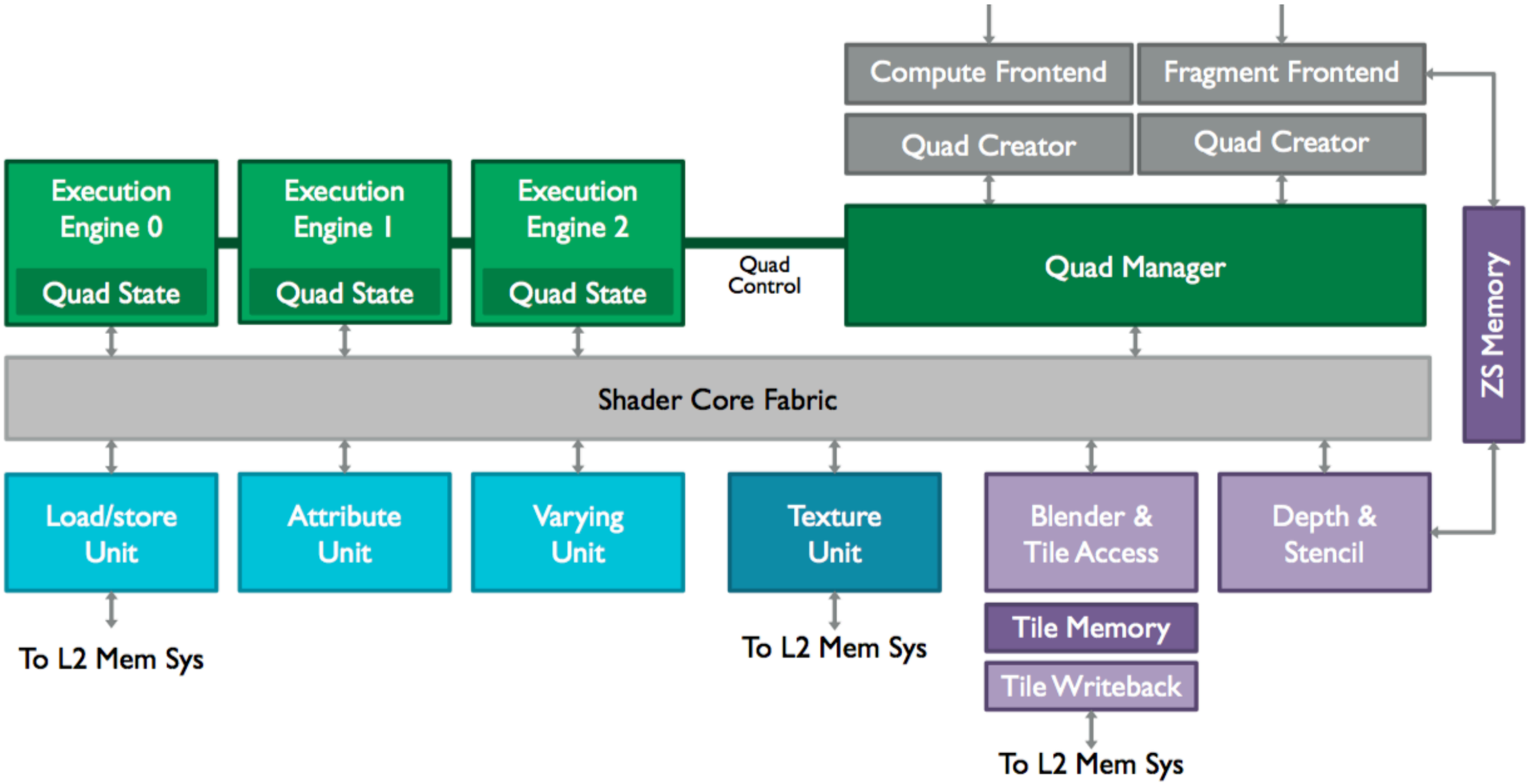


image courtesy Jem Davies, "Bifrost GPU", 2016, © ARM

Tiling : Advantages

- Easily parallelizable
- Allows access to frame buffer in a small high speed memory
- Z, Color accesses close to free
  - MSAA is almost free (5-10% of rendering time)
- Alpha Blending is significantly cheaper

Tiling : Disadvantages

- More on chip memory for sorting triangles into tiles/bins
- Increased bandwidth for triangle sorting
- State changes now happen for every tile (can take a lot of time)
- Triangles can be processed multiple times (can be solved with Mali hierarchical tiling)
- On chip memory size places hard limits on number of triangles
  - Overflow causes spilling and creates a big hit on performance
  - Too much geometry will flush whole pipeline (ParameterBuffer overflow)

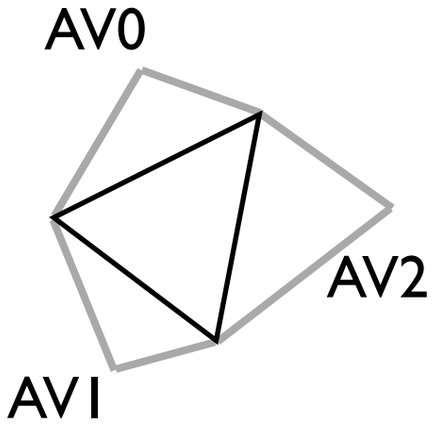
Comparing Tiling to the Traditional/Straight Pipeline

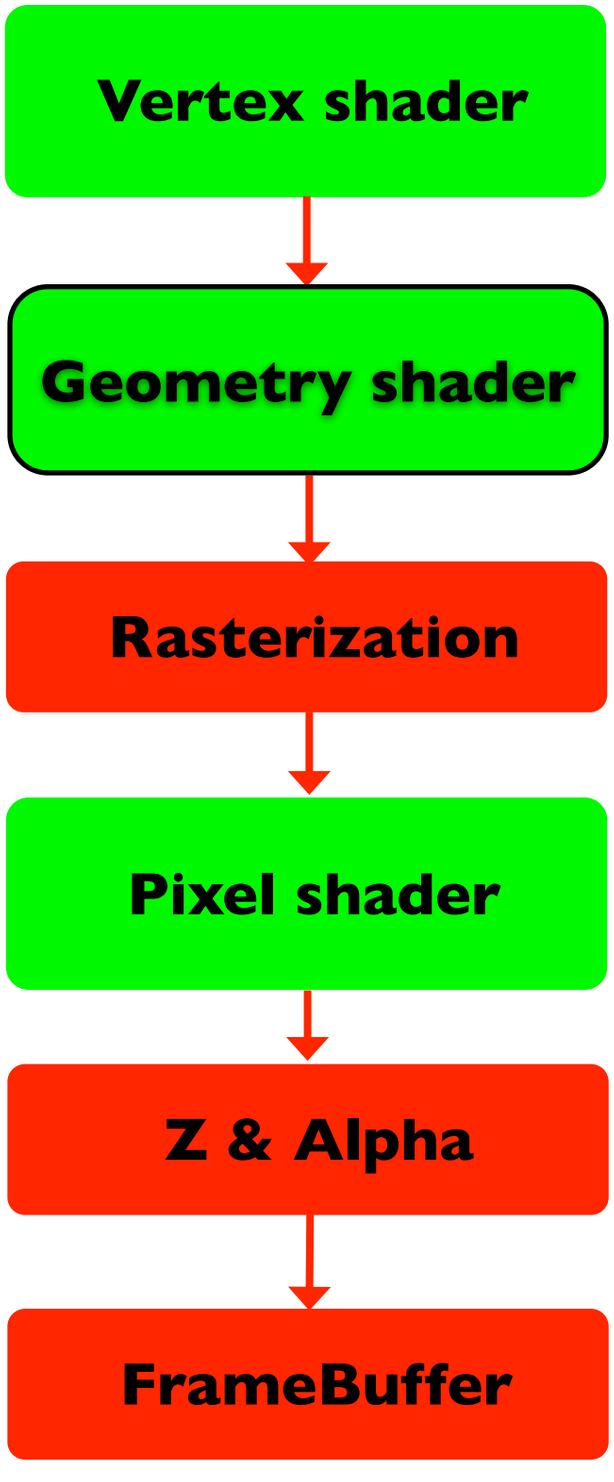
- Advantages of Straight Pipeline
  - Less complex pipeline
  - More predictable performance
    - Z Framebuffer compression techniques reduce off-chip bandwidth
    - State changes once for frame (but too many still a problem)
- Disadvantages
  - Impossible to have frame buffer in on chip memory
  - Accessing the frame buffer cost power and latency

Expanding the graphics pipeline

- Vertex shaders & Pixel shaders
- But wait there's more !!!
- Geometry Shaders
  - Triangle in, many triangles out
  - Like a triangle shader
- Tessellation (in OpenGL terminology)
  - Control Shader (DirectX Hull Shader)
  - Tessellator
  - Evaluation Shader (DirectX Domain Shader)

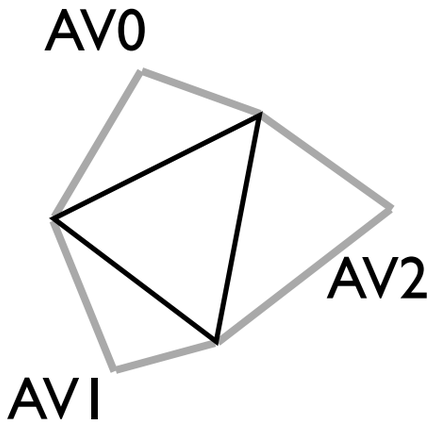
Geometry Shaders

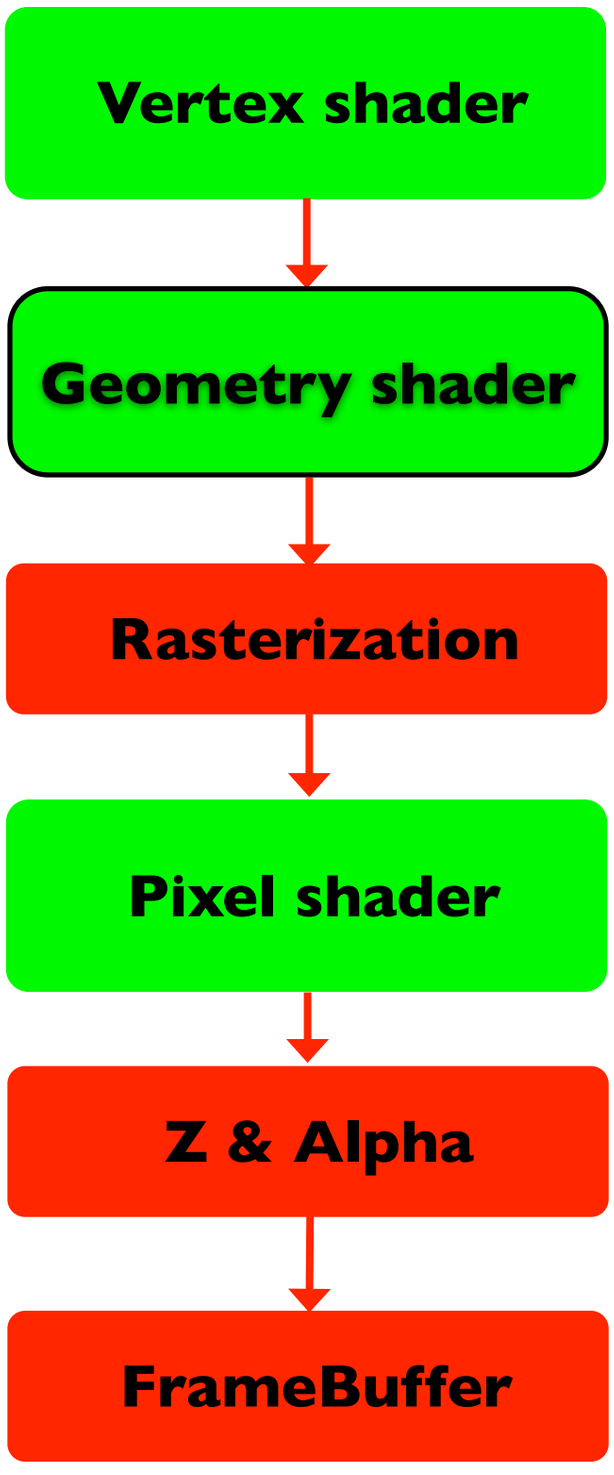




- Input and output can be points, lines, triangles
  - But lines and triangles can also have adjacency information (3 for triangles, 2 for lines)
  - Can think of it as the 'Triangle shader'
- 1:n ratio of input:output
  - 1 primitive in, many (or no) primitives out
  - Must set a **`max_vertices`** for number of outputs
  - Number of vertices, number of output components
  - Enables storing processed vertices into multiple buffers (framebuffers)

Geometry Shaders





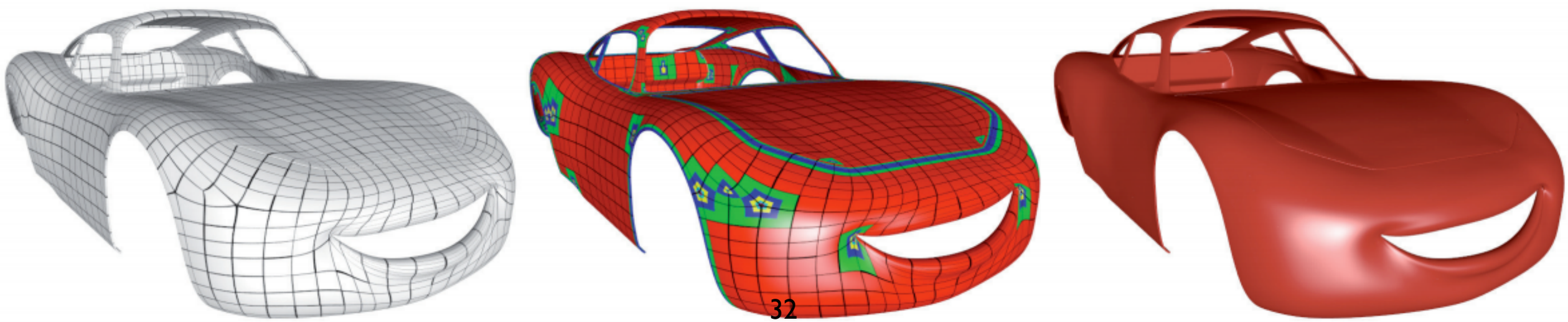
- Output lines and triangles are strips
  - If you want individual, you need **`EndPrimitive`**
- Layered rendering
  - Send primitives to specific layers of framebuffer
  - Good for rendering cube-based shadow mapping and environment maps
- DirectX 10, OpenGL 3.0 feature

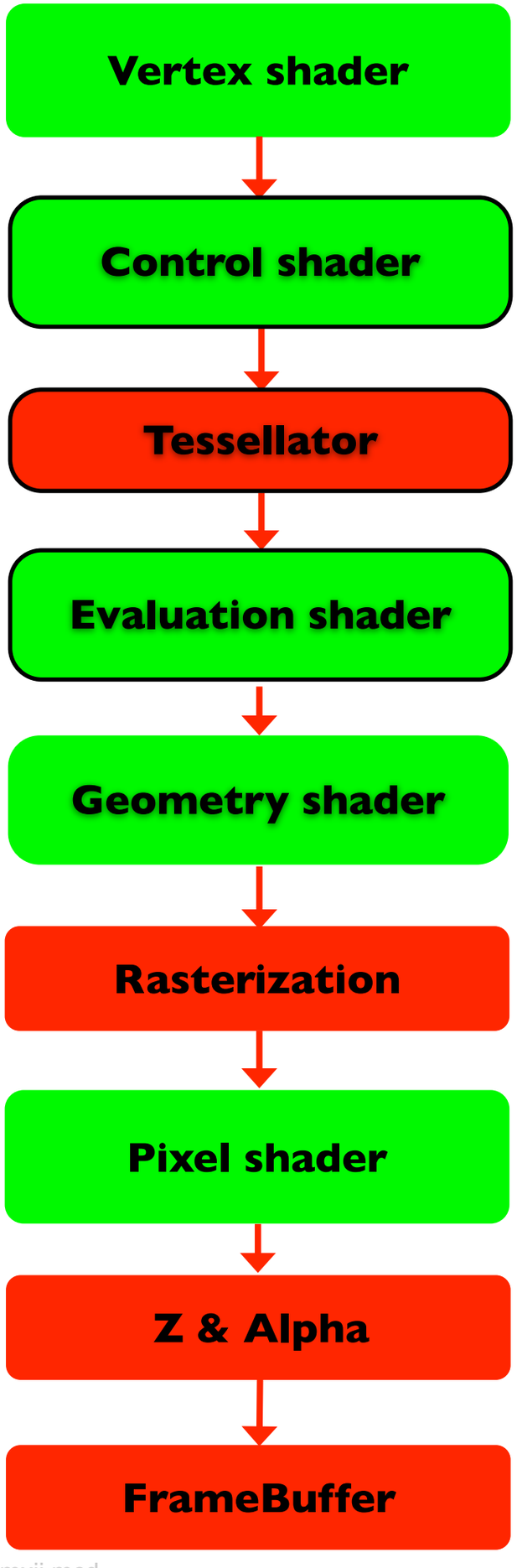
Geometry Shader Applications

- Single pass render to cube map
  - Using `gl_Layer`
- Point sprite expansion/generation
- Fur/Fin generation
- Marching cubes triangle generation
- Shadow volume extrusion
- Tessellation?

Tessellation

- Difference between games and film is geometric detail
- Film uses higher order surfaces (HOS)
  - Subdivision Surfaces
  - Bezier patches, NURBs
- Tessellation converts HOS into triangles

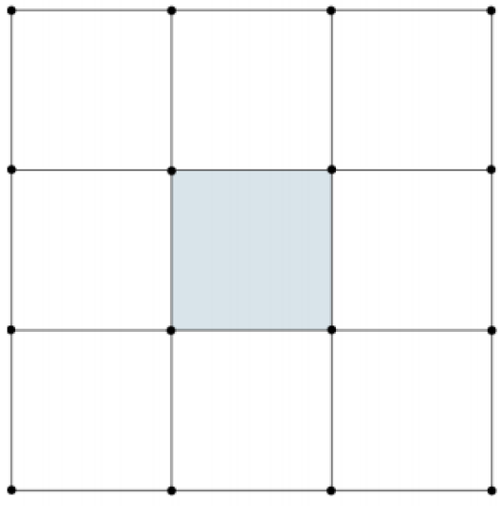




Tessellation

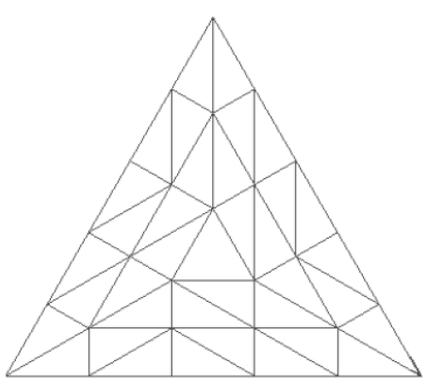
- Benefits
  - Higher order surfaces such as subdivision surfaces
  - Higher detailed surfaces using displaced mapping
  - View-dependent levels-of-detail
- Input is quad or triangle patches
- DirectX 11 (Hull and Domain shader), OpenGL 4.0 feature

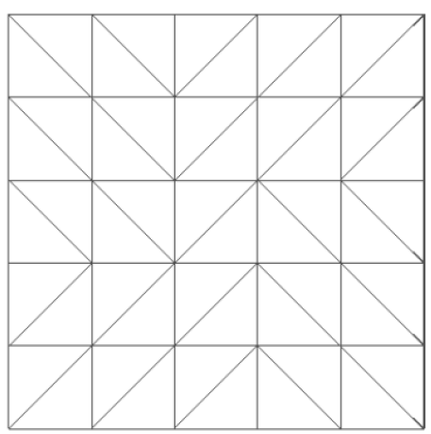
Tessellation



Regular Patch

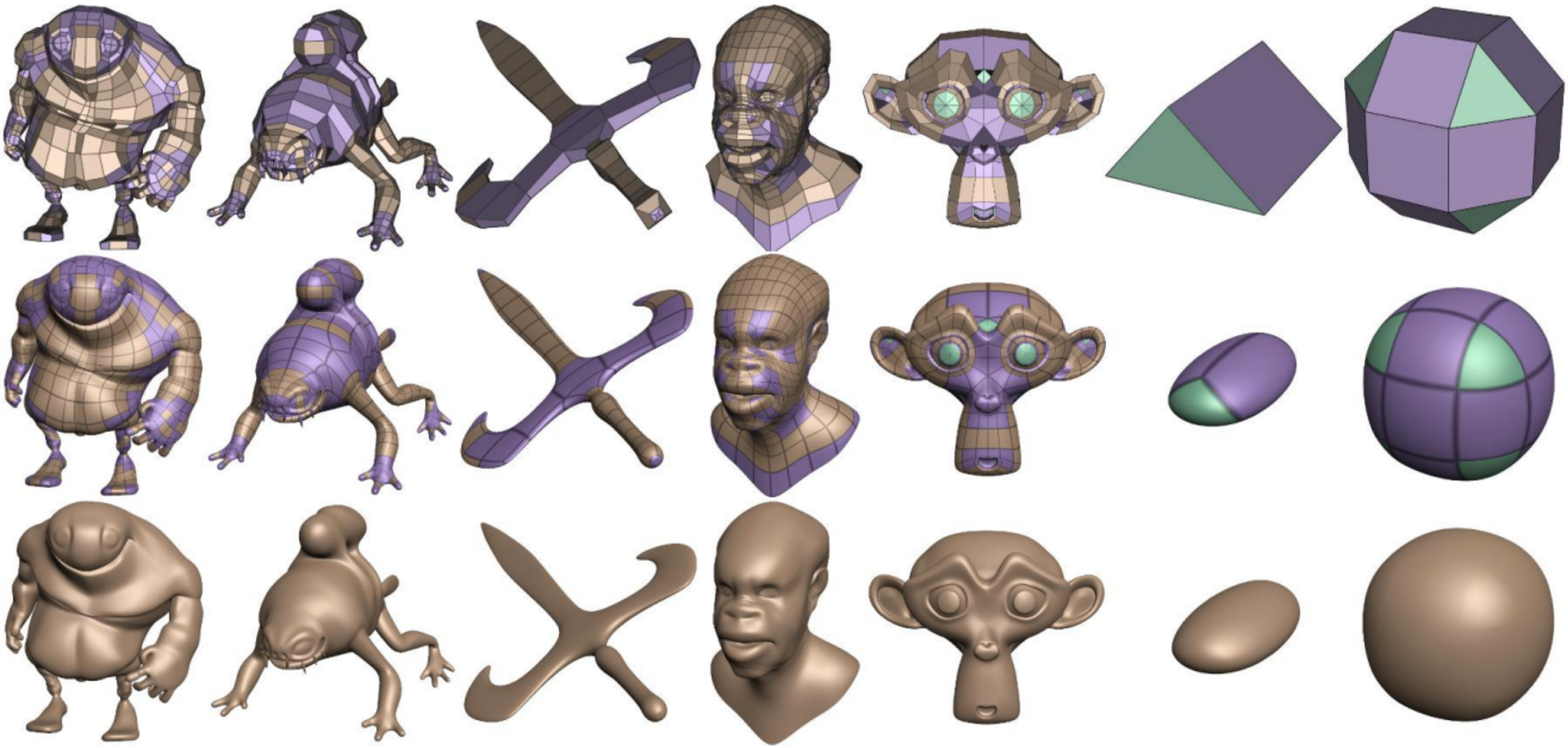
16 control points



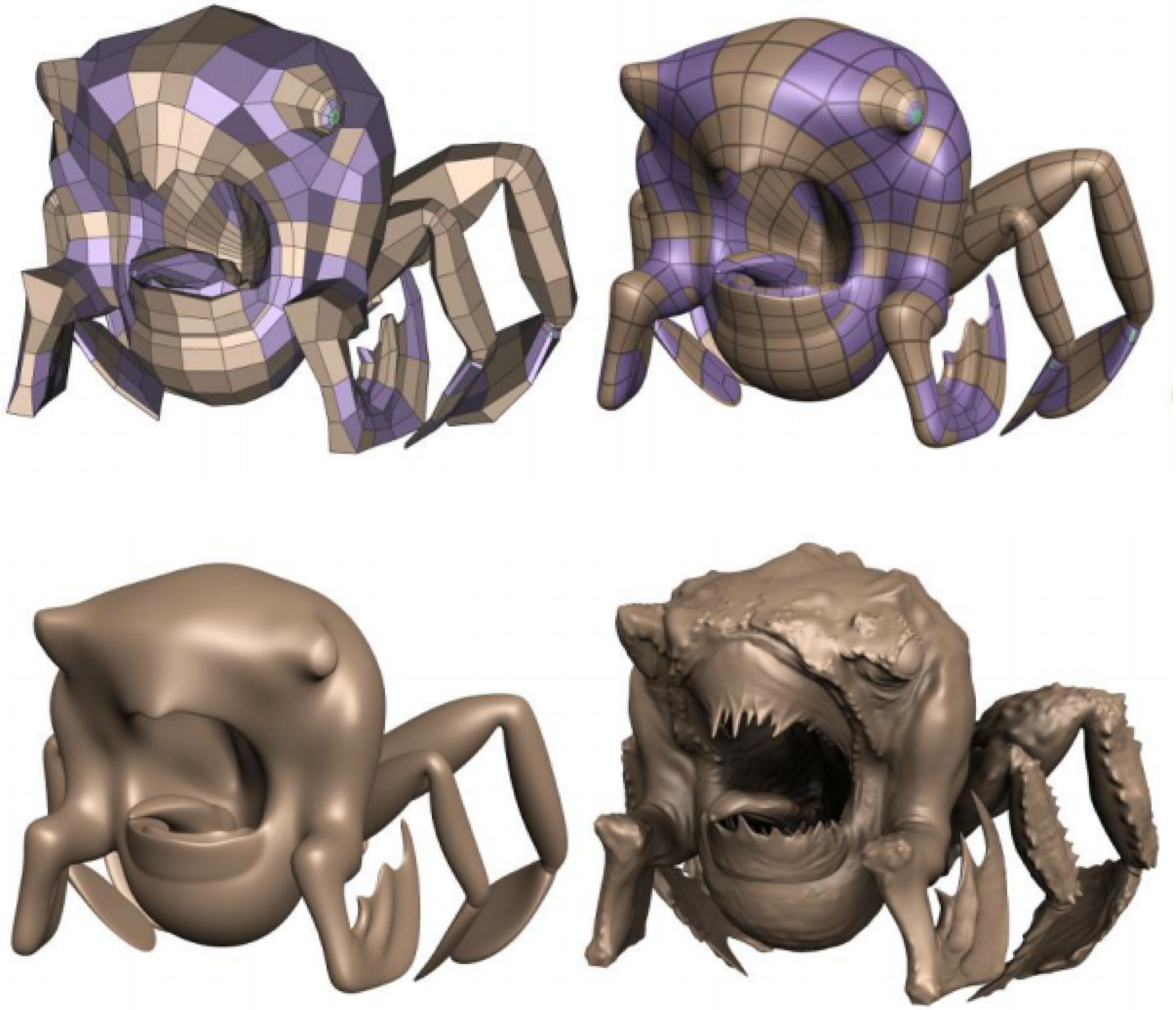


- Control shader
  - Transforms up to 32 control points of a patch (typically 16)
  - Calculates tessellation factor for patch edges in tessellator
- Tessellator
  - Outputs UV coordinates
- Evaluation shader
  - Calculates the vertex position of the output of tessellator in the patch
- More info and OpenGL code on LearnOpenGL.com
  - https://learnopengl.com/Guest-Articles/2021/Tessellation/Tessellation

Images courtesy Charles Loop, "Hardware Subdivision and Tessellation of Catmull-Clark Surfaces", GTC2010



Images courtesy Charles Loop, "Hardware Subdivision and Tessellation of Catmull-Clark Surfaces", GTC2010



displacement mapping

Images courtesy Charles Loop, "Hardware Subdivision and Tessellation of Catmull-Clark Surfaces", GTC2010


OpenCL

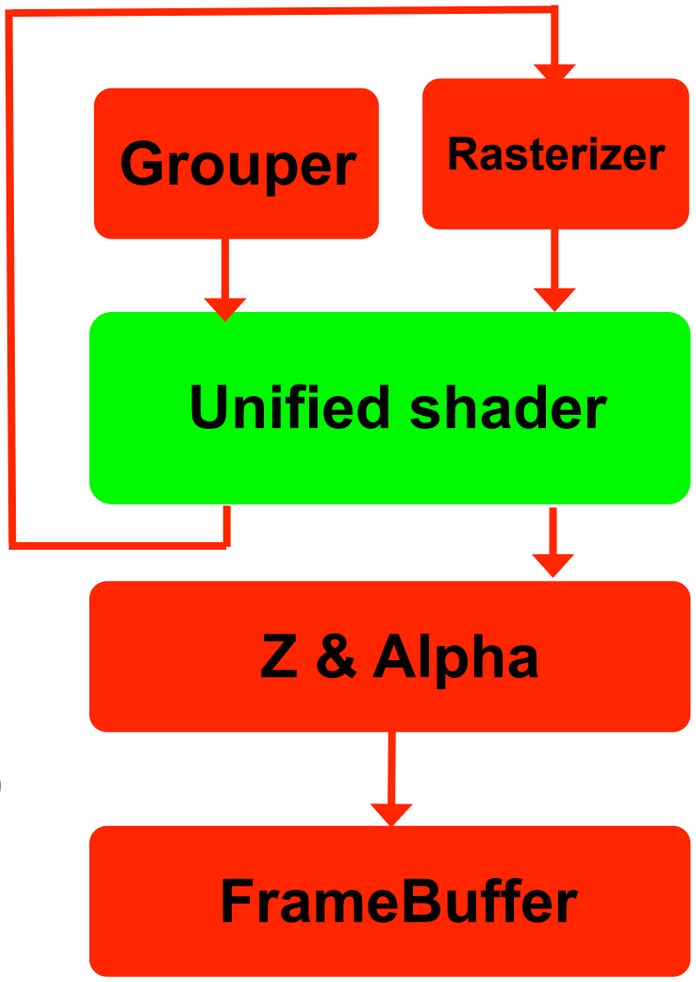
OpenCL

- OpenCL - Open Compute Language
- Application Programming Interface (API)
- Runs on GPUs
- Write programs (shaders) for GPU using OpenCL
- Industry standard between multiple companies

GPU as a parallel processor

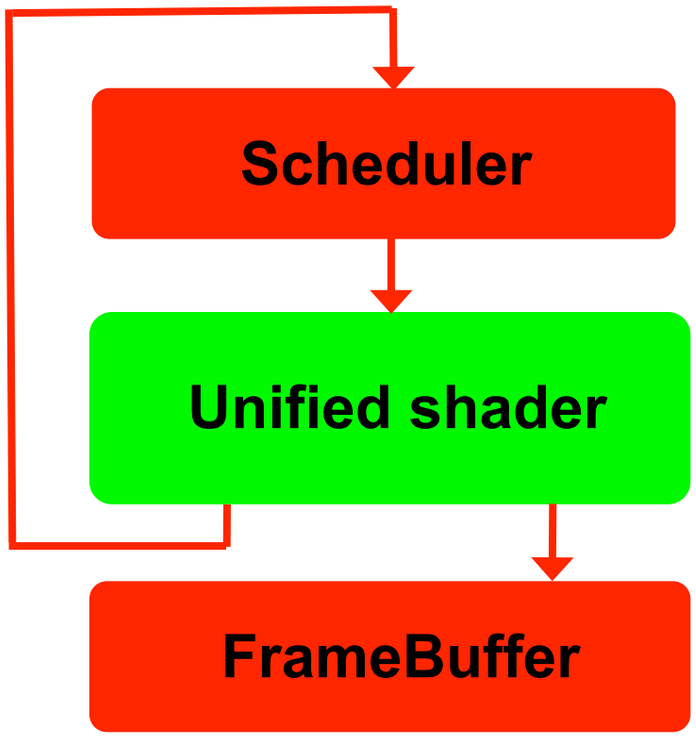
- GPU parallelism has increased significantly over time
  - Roughly 1000x more pixels/sec in 10 years
- People started to use this for more than just graphics
- How to enable using the GPU as a parallel processor?
  - How could we change OpenGL?

Designing OpenCL



- Take out all the graphics
  - Triangles, *textures*, Z testing, alpha blending
- Keep simple small programs (graphics shaders)
  - Compute **kernels**
  - Use the C programming model for kernels (ANSI C99)
  - Low-level abstraction
    - High-performance, but device independent
- CPUs are also parallel (multicore) so OpenCL works there too

Designing OpenCL



- Take out all the graphics
  - Triangles, *textures*, Z testing, alpha blending
- Keep simple small programs (graphics shaders)
  - Compute **kernels**
  - Use the C programming model for kernels (ANSI C99)
  - Low-level abstraction
    - High-performance, but device independent
- CPUs are also parallel (multicore) so OpenCL works there too

OpenCL platform

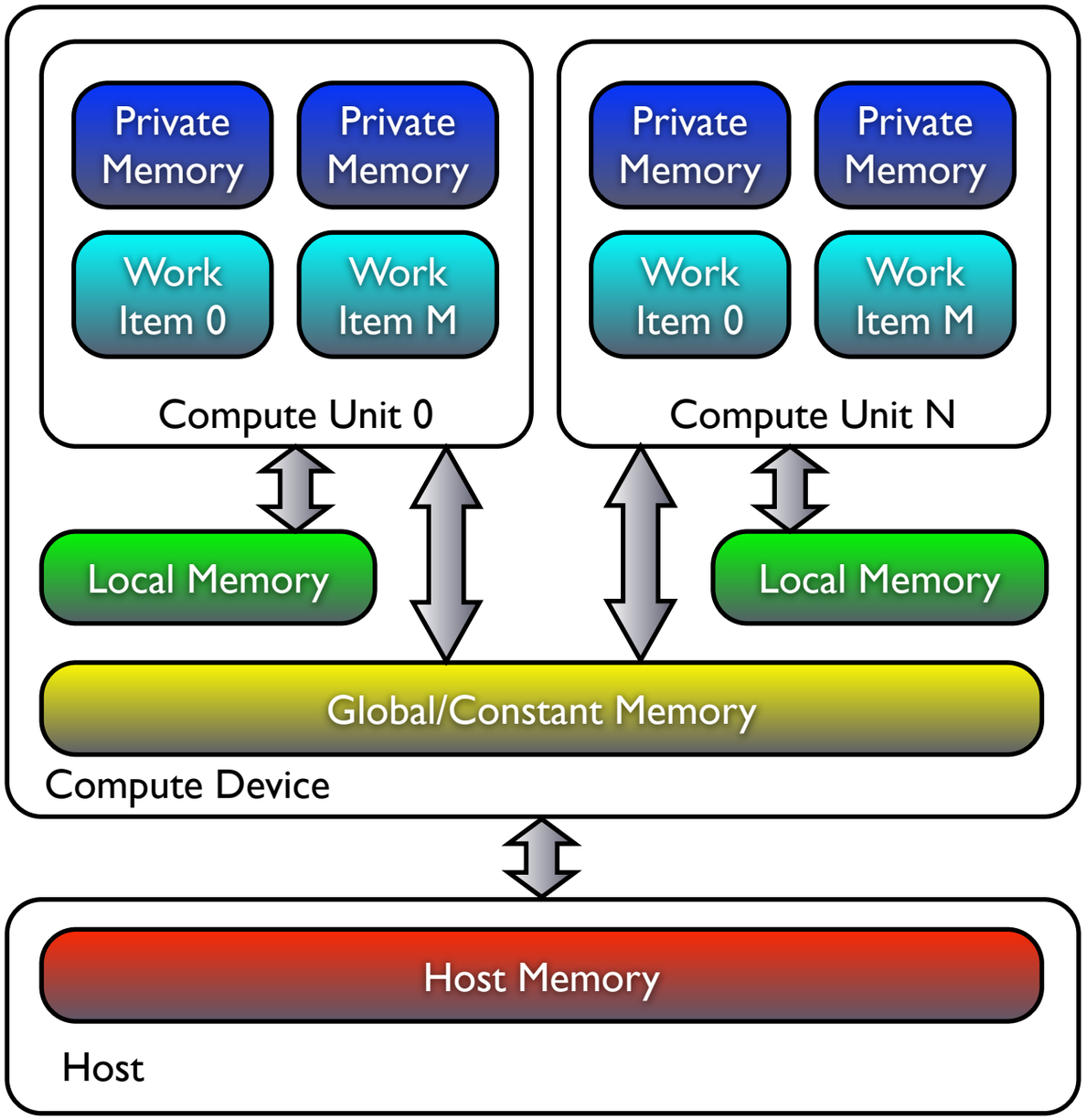
- Compute device may be a CPU, GPU or other processor
  - Compute device is made up of Compute Units
  - Compute Unit is made up of Processing Elements
  - Processing elements execute code as SIMD or SPMD (Single Program Multiple Data)
  - What could be added to GPUs for parallel programming and not just graphics?
    - GPU programs have a lot of dedicated memory(RAM) that only shaders use
    - Shaders cannot communicate with each other while running

How can kernels share data in OpenCL?

- Kernel **Local Memory**
- Allow kernel programs to share data while running
  - Nvidia's CUDA shared memory
  - Graphics groups pixels (shaders) with triangles
  - OpenCL groups work-items (*kernels/threads*) with work-groups (*blocks*)
  - Local Memory is where **work-items** in a **work-group** can share data

OpenCL Memory model

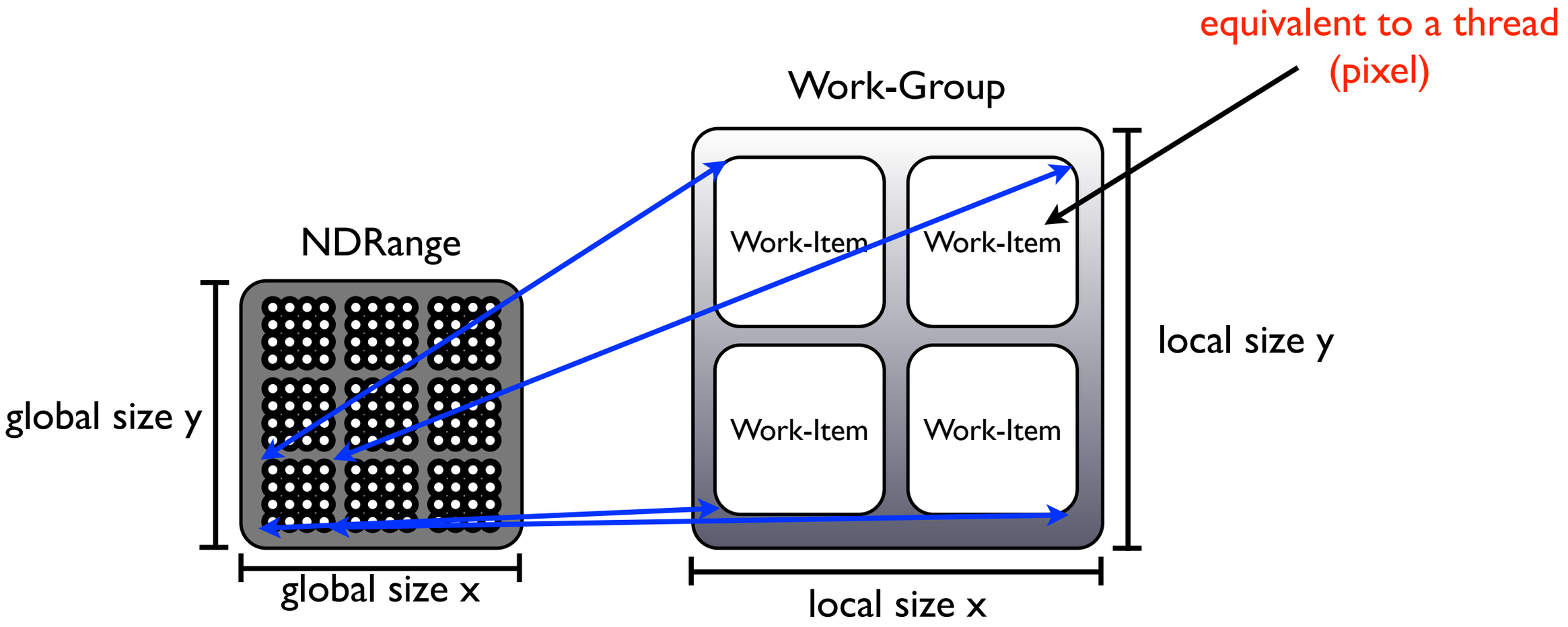
- Private Memory
  - Per work-item - registers
- Local Memory
  - Shared within a workgroup
- Global/Constant Memory
  - Visible to all workgroups
  - Read-only data (Constant)
  - Off-chip memory for GPU
    - GPU memory/framebuffer
- Host Memory
  - CPU's memory



How do you run commands in OpenCL?

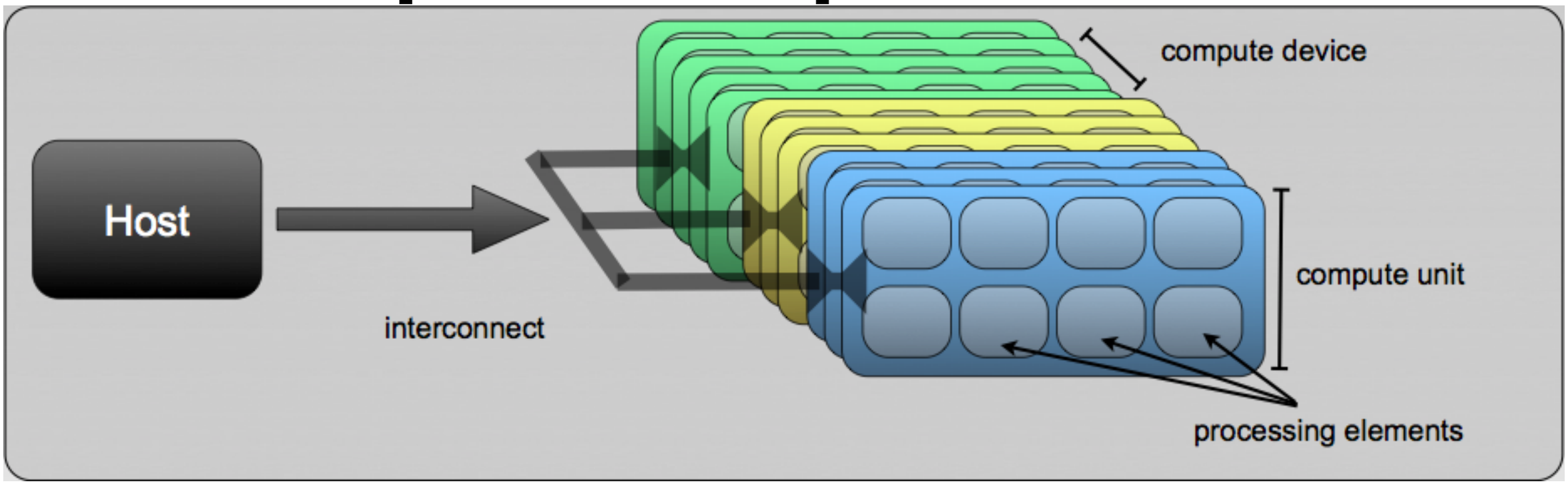
- Commands are submitted to a GPU or CPU queue
- Execution can be in-order or out-of-order
- Events are used for synchronization

Execution model



- Kernels are run over an N-Dimensional range (NDRange)
- Each work-item has a unique identifier (global-id)

OpenCL platform



- Compute device may be a CPU, GPU or other processor
  - Compute device is made up of Compute Units
  - Compute Unit is made up of Processing Elements
- Processing elements execute code as SIMD or SPMD

Summary

- OpenCL enables parallel programming
  - on both GPUs and CPUs
  - from different vendors
    - AMD (CPU&GPU), Nvidia (GPU), Intel (CPU&GPU), ARM(GPU)
  - on different operating systems
    - windows/linux/mac/android
  - At low-level (performance) with well known programming language

Further material on OpenCL and Compute

- Khronos web page
  - http://www.khronos.org/opencl/
- OpenGL 4.3 Compute Shaders
  - https://learnopengl.com/Guest-Articles/2022/Compute-Shaders/Introduction
- Nvidia sample code
  - https://developer.nvidia.com/opencl

Next

- Next Monday (9th)
  - 1st - Advanced Graphics Summit: Raytracing in Snowdrop: An Optimized Lighting Pipeline for Consoles
    - Quentin Kuenlin, Massive Entertainment
    - For the game, Avatar: Frontiers of Pandora
  - 2nd - Vulkan, Animations and the Ray-Tracing pipeline
    - Gustaf Waldemarson, ARM
- No Lecture next Thursday (12th), or following Monday (16th)
  - More time for projects!