Introduction to Vulkan

A High Level Walk Through Modern Graphics API Concepts

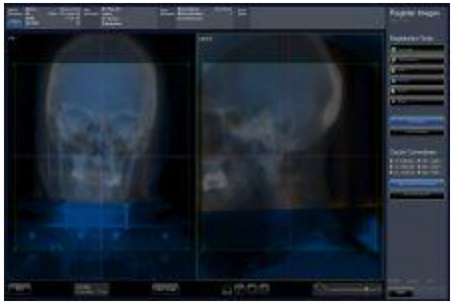
Dan Ginsburg – Upsample Software, LLC

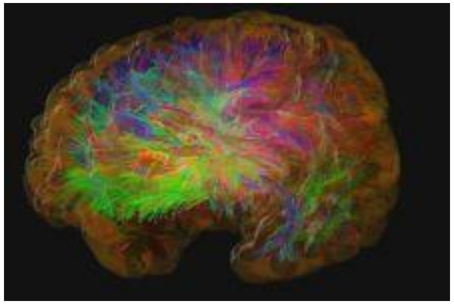
UMASS BOSTON

Overview

- About Me
- A Bit of Graphics APIs History
- Why Vulkan?
- Key Modern Graphics API Concepts
- Vulkan Resources

Stuff I’ve worked on…





Many possible career paths for graphics programmers…

- Drivers
  - **NVIDIA, AMD, Qualcomm, Intel, ARM, Apple, ImgTec, etc.**
- GPU Tools, GPU Research
  - **Same companies as above, + game engines/devs**
- Rendering
  - **Game developers/engines**
- Many other fields that use graphics/GPUs
  - **Medical Devices**
  - **Avionics/Automotive**
  - **User Interfaces**
  - **CAD**
  - **…**

A Bit of Graphics API History

How did WebGL come to be?

Let’s go back 15 years…2006

What is OpenGL ES?

- OpenGL-based API for embedded systems
- Removed redundant and expensive functionality
- Kept as compatible with OpenGL as possible
- Added features needed for embedded systems

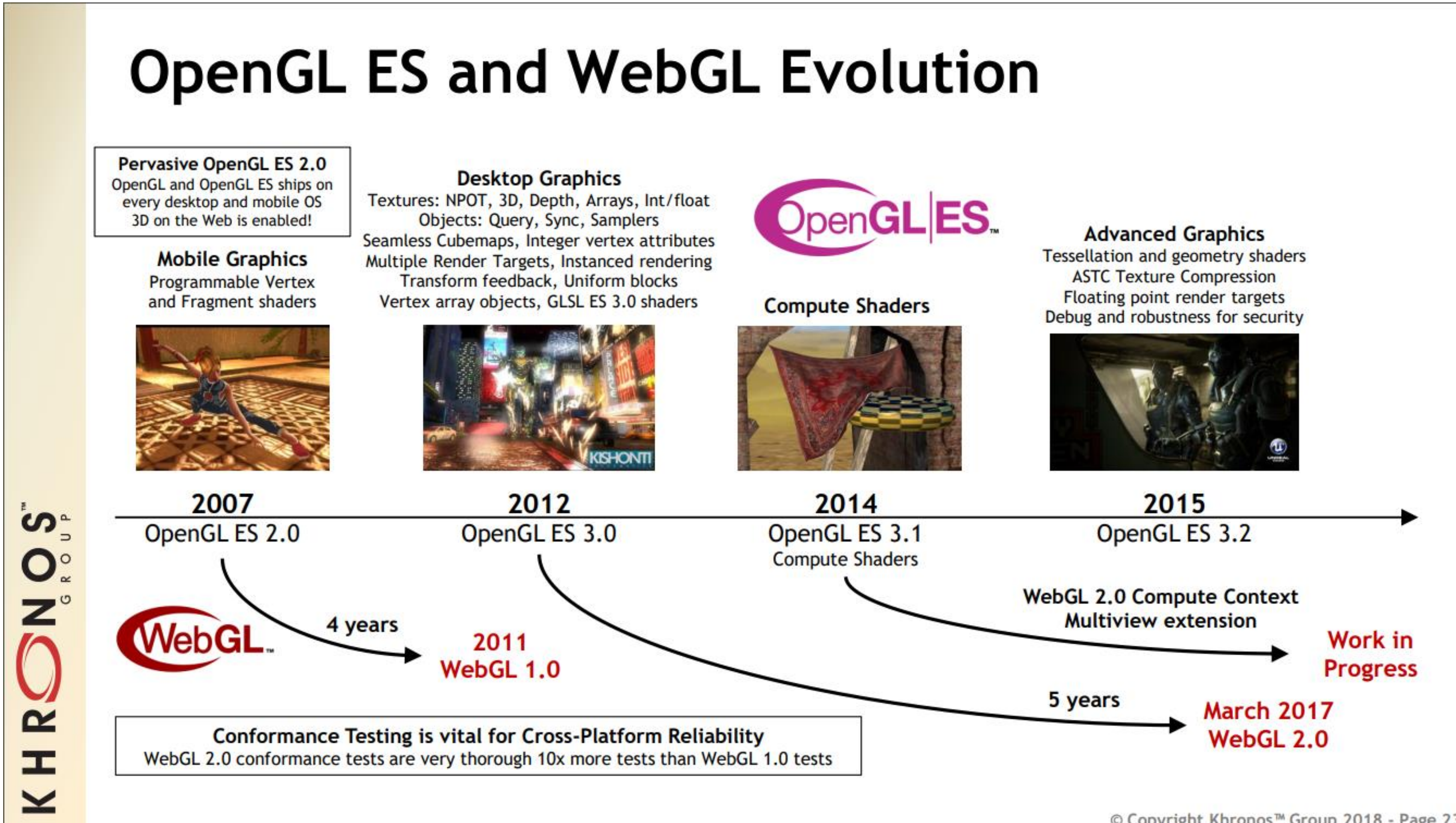
What is OpenGL ES?

- OpenGL-based API for embedded systems
- Removed redundant and expensive functionality
- Kept as compatible with OpenGL as possible
- Added features needed for embedded systems

OpenGL ES History

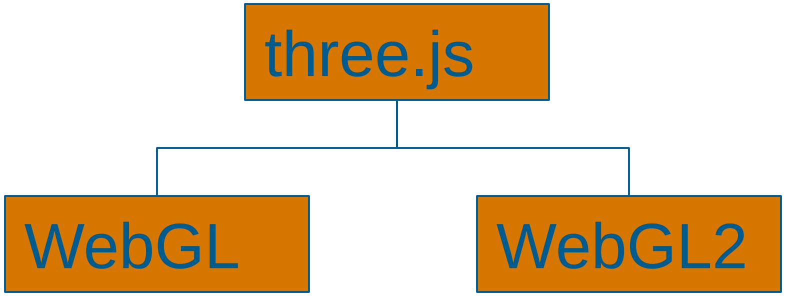
- OpenGL ES 1.0
  - Basic 3D functionality
- OpenGL ES 1.1+
  - Comprehensive set of fixed-function 3D
  - Backwards compatible
- OpenGL ES 2.0
  - Shader only
  - Not backwards compatible

iPhone 3G S – June 2008

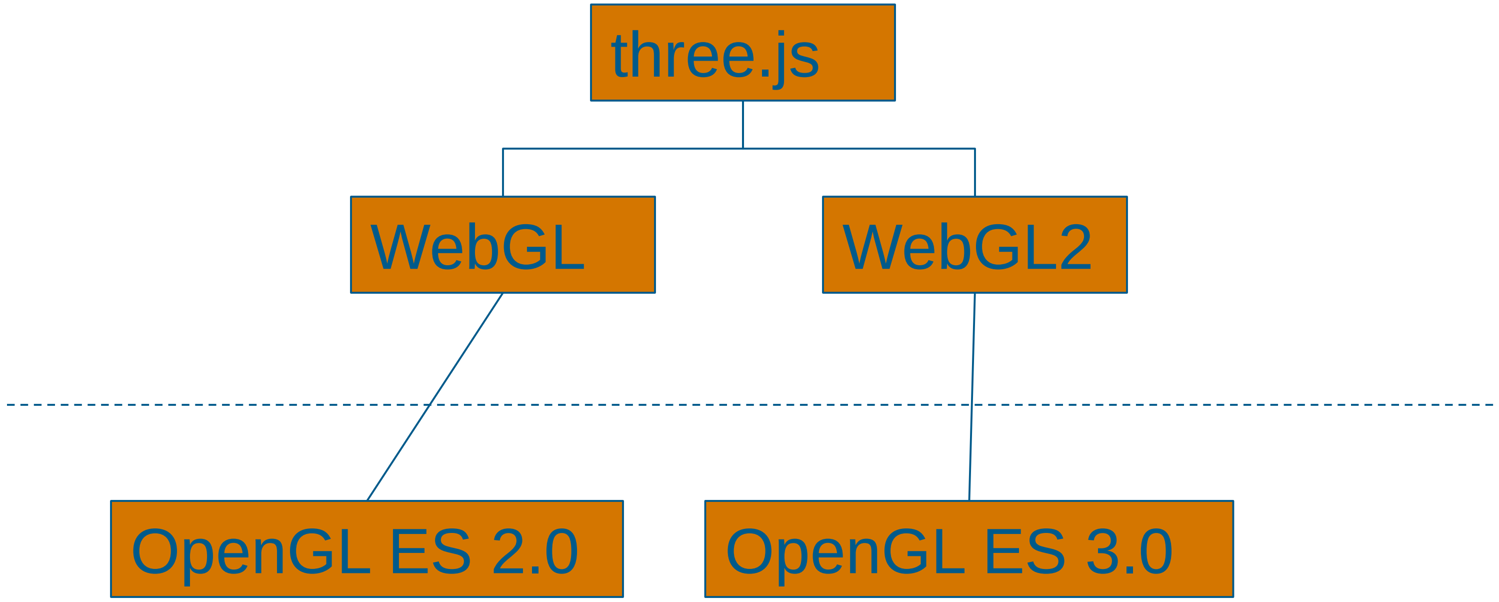


Source:
https://www.web3d.org/sites/default/files/attachment/node/2326/edit/49_HwanyongLee_KhronosLiaisonReport_Seoul2019January24.pdf

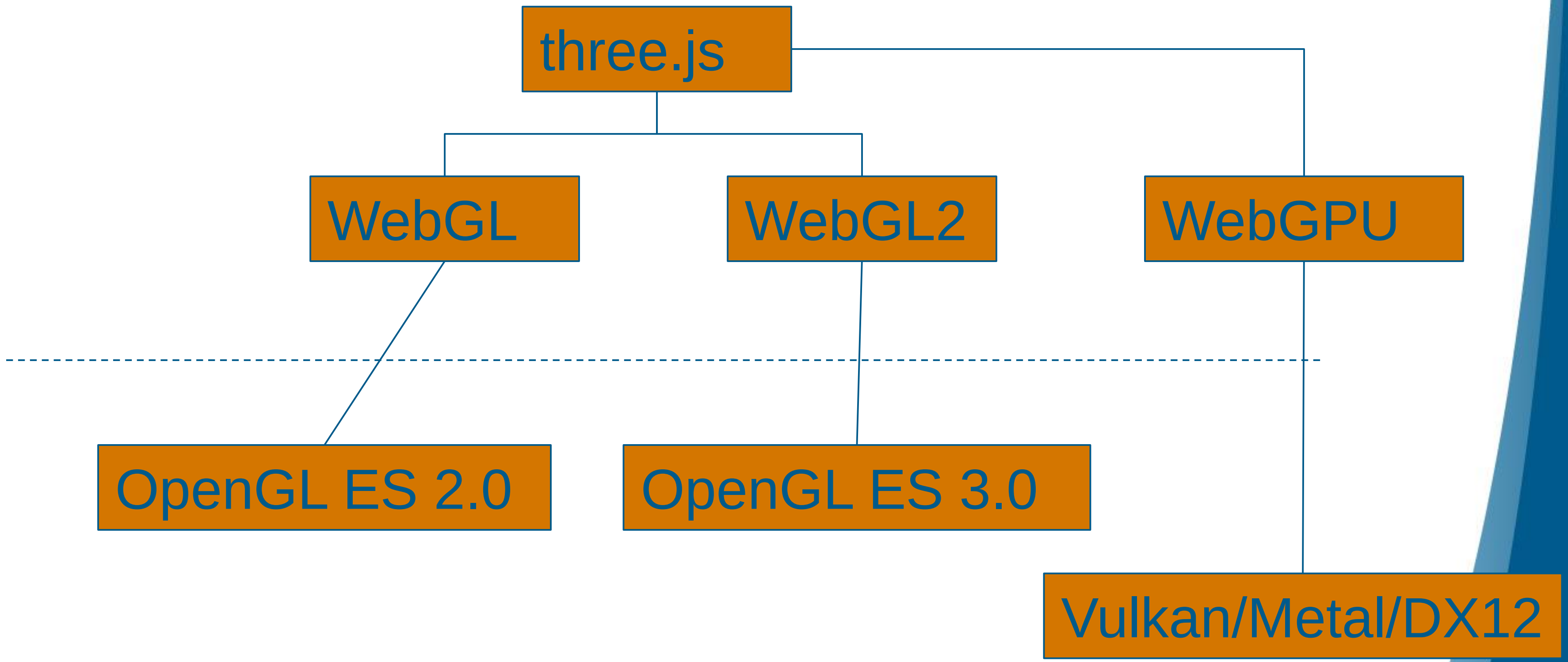
three.js Web APIs



three.js Web APIs

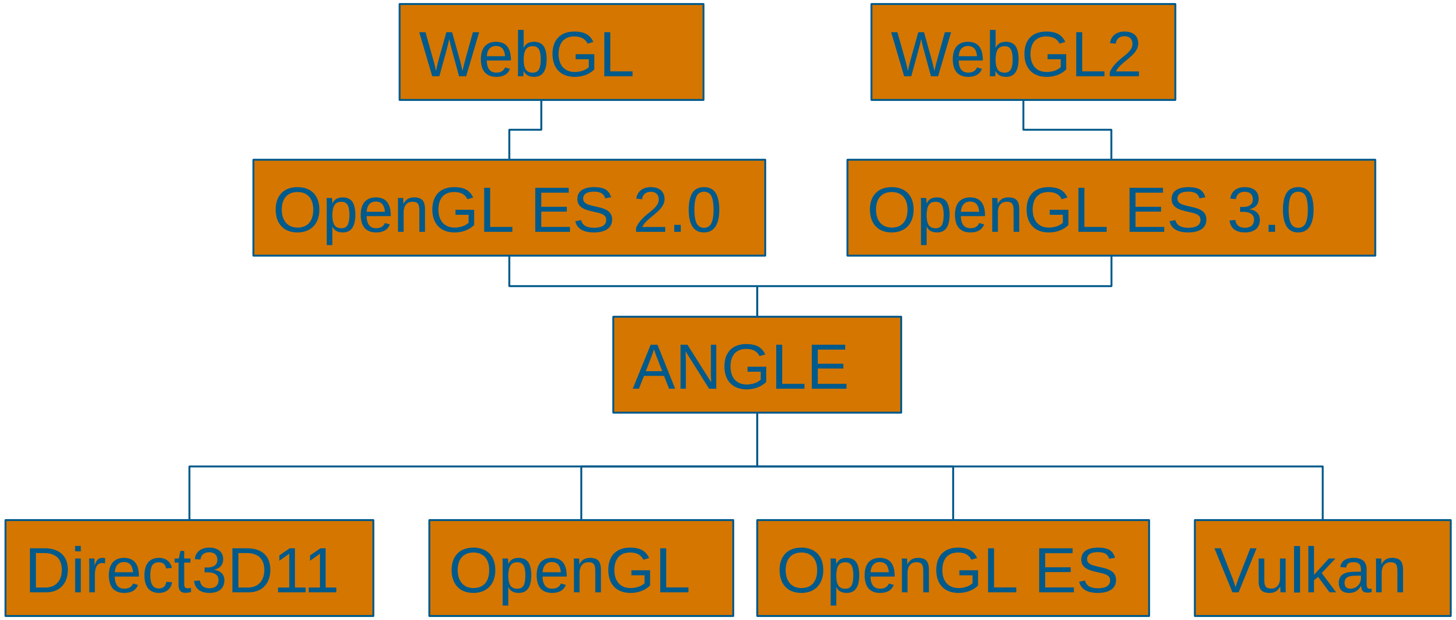


three.js Web APIs Future

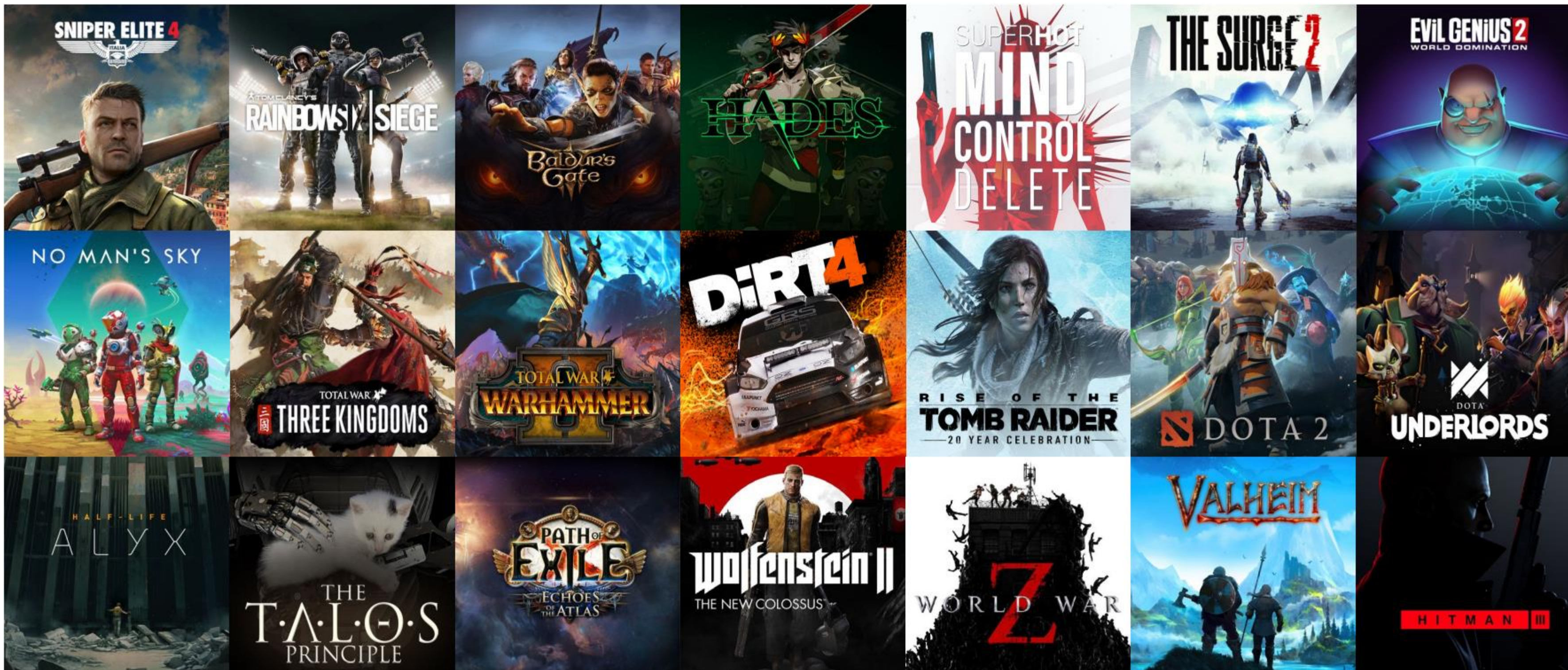


Chrome WebGL Implementation

- ANGLE: https://github.com/google/angle



Why Vulkan?



SNIPER ELITE 4
TOM CLANCY'S RAINBOW SIX SIEGE
Baldur's Gate
HADES
SUPERHOT MIND CONTROL DELETE
THE SURGE 2
EVIL GENIUS 2
WORLD DOMINATION
NO MAN'S SKY
TOTAL WAR
THREE KINGDOMS
TOTAL WAR WARHAMMER II
DiRT4
RISE OF THE
TOMB RAIDER
20 YEAR CELEBRATION
DOTA 2
DOTA
UNDERLORDS
HALF-LIFE
ALYX
THE
T·A·L·O·S
PRINCIPLE
PATH OF EXILE
ECHOES OF THE ATLAS
Wolfenstein II
THE NEW COLOSSUS
WORLD WAR Z
VALHEIM
HITMAN III

OpenGL/OpenGL ES Submission Model

- One Thread Submits to the API

OpenGL/OpenGL ES Submission Model

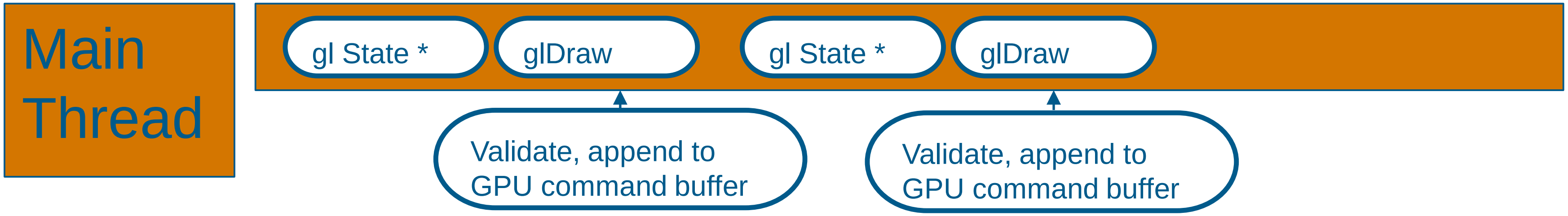
- One Thread Submits to the API

OpenGL/OpenGL ES Submission Model

- One Thread Submits to the API

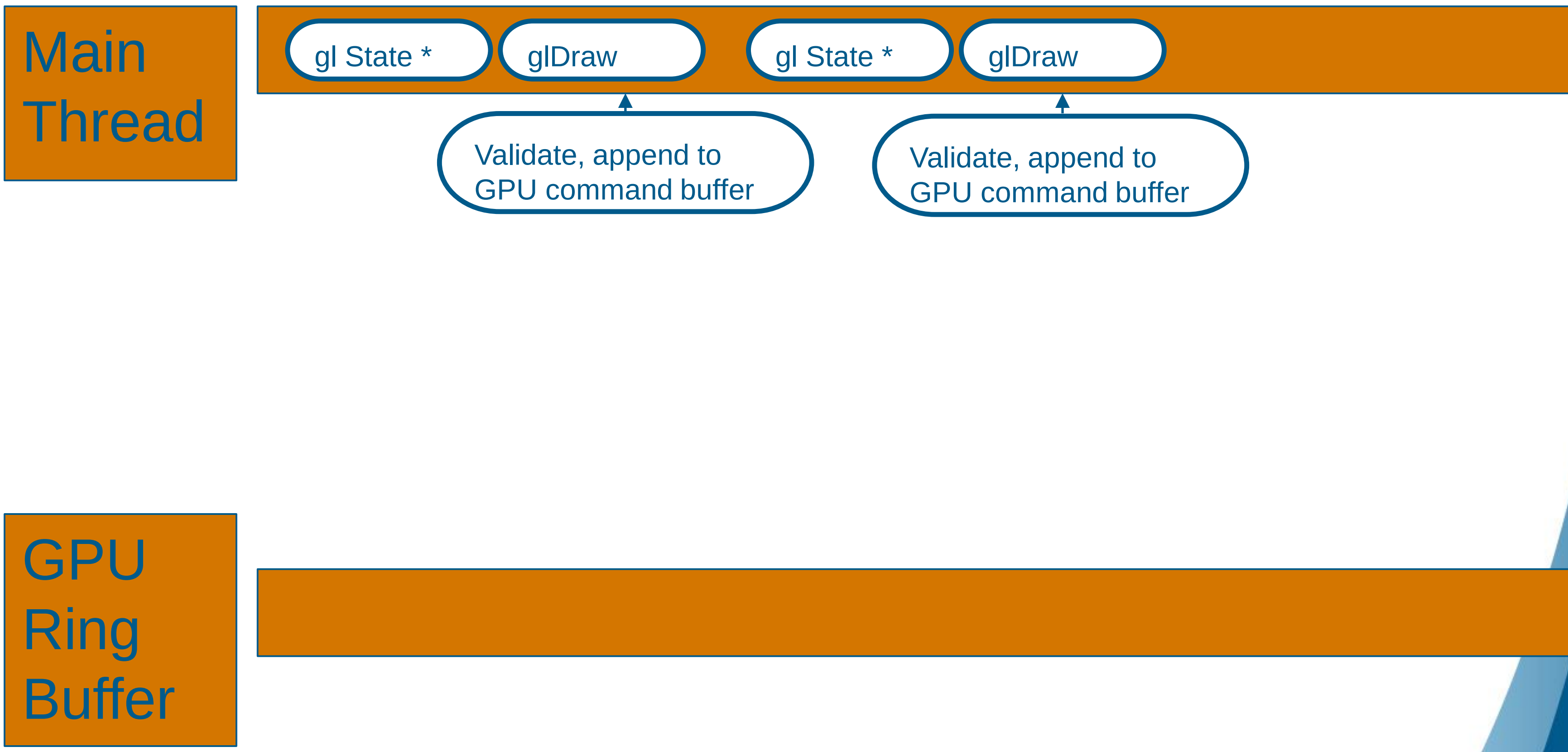
OpenGL/OpenGL ES Submission Model

- One Thread Submits to the API



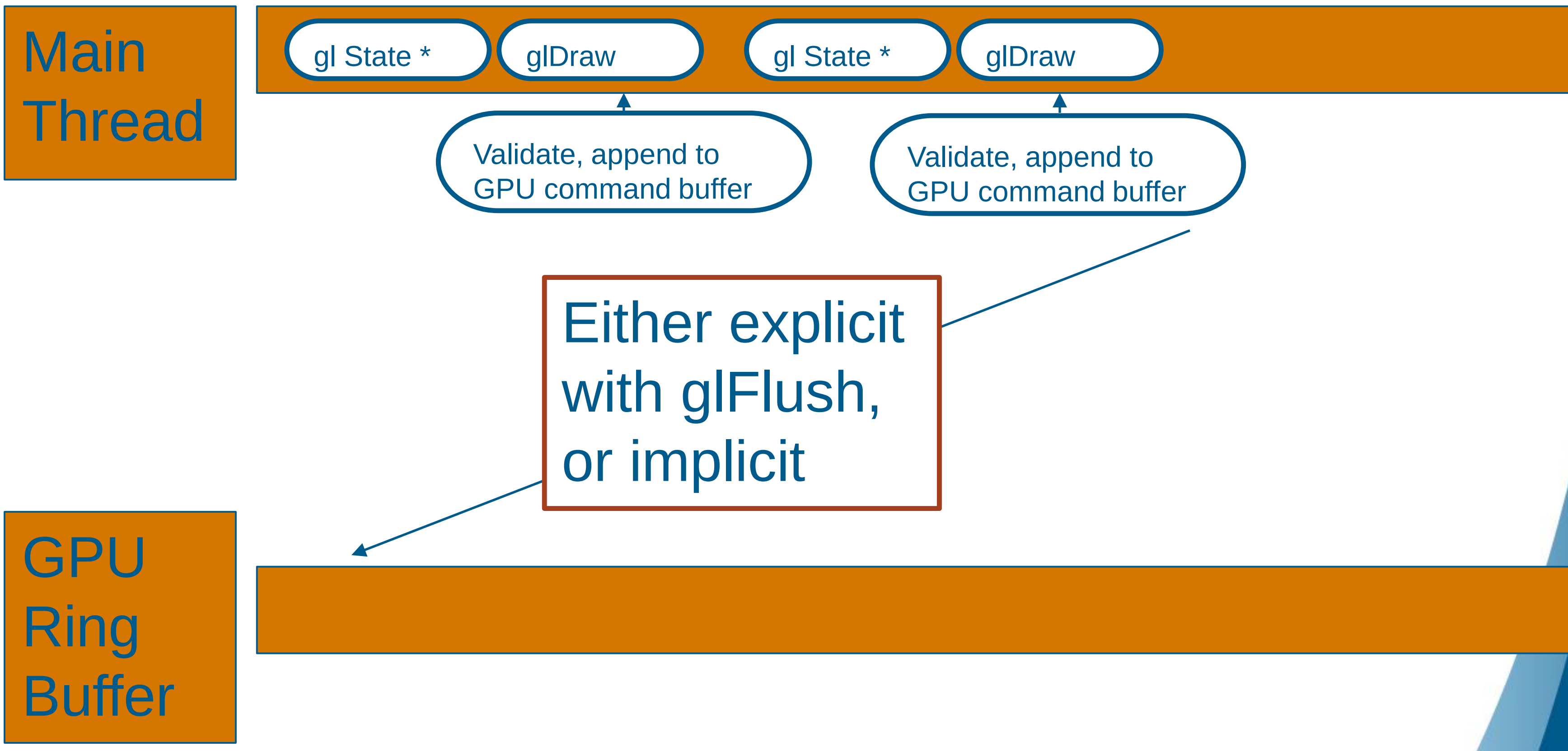
OpenGL/OpenGL ES Submission Model

- One Thread Submits to the API



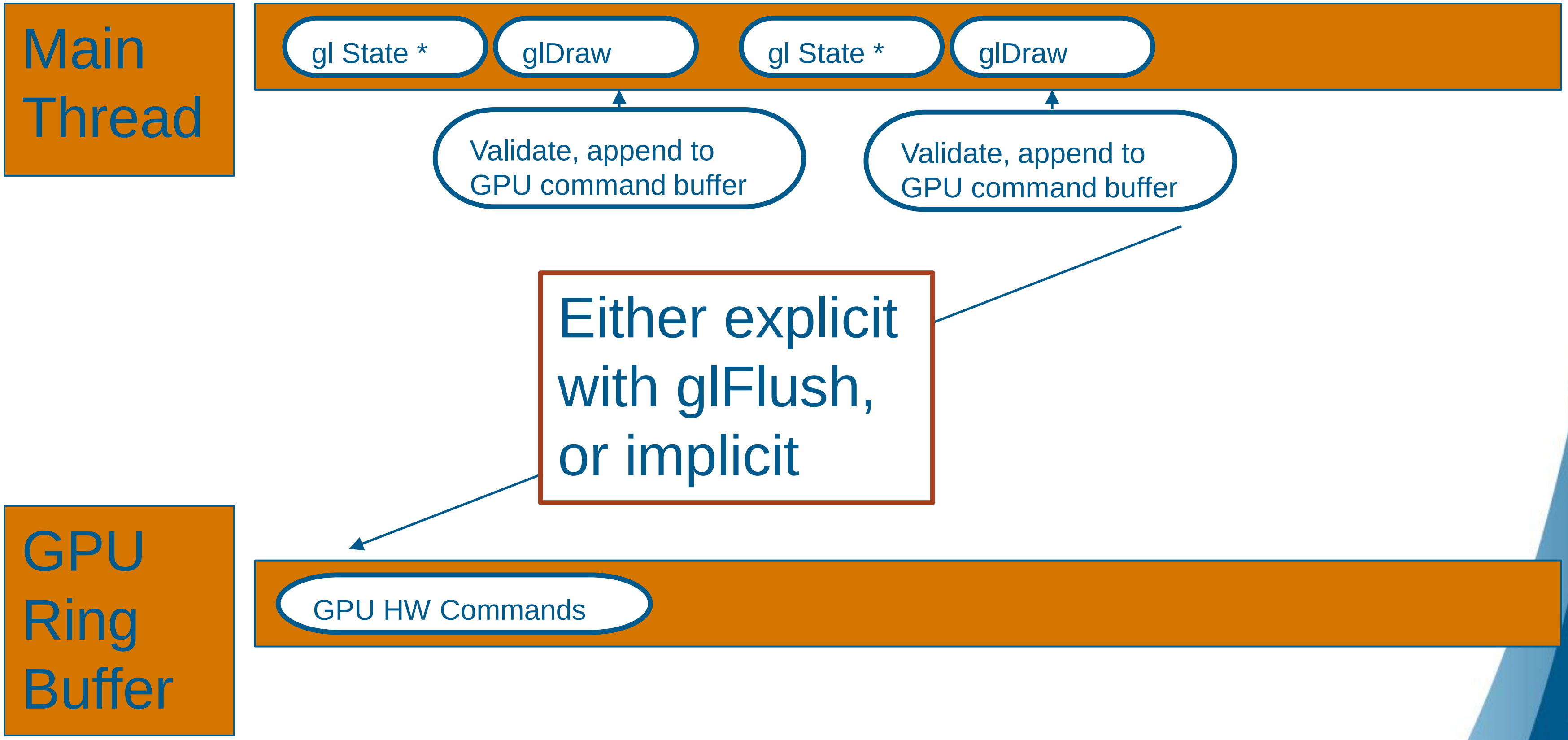
OpenGL/OpenGL ES Submission Model

- One Thread Submits to the API



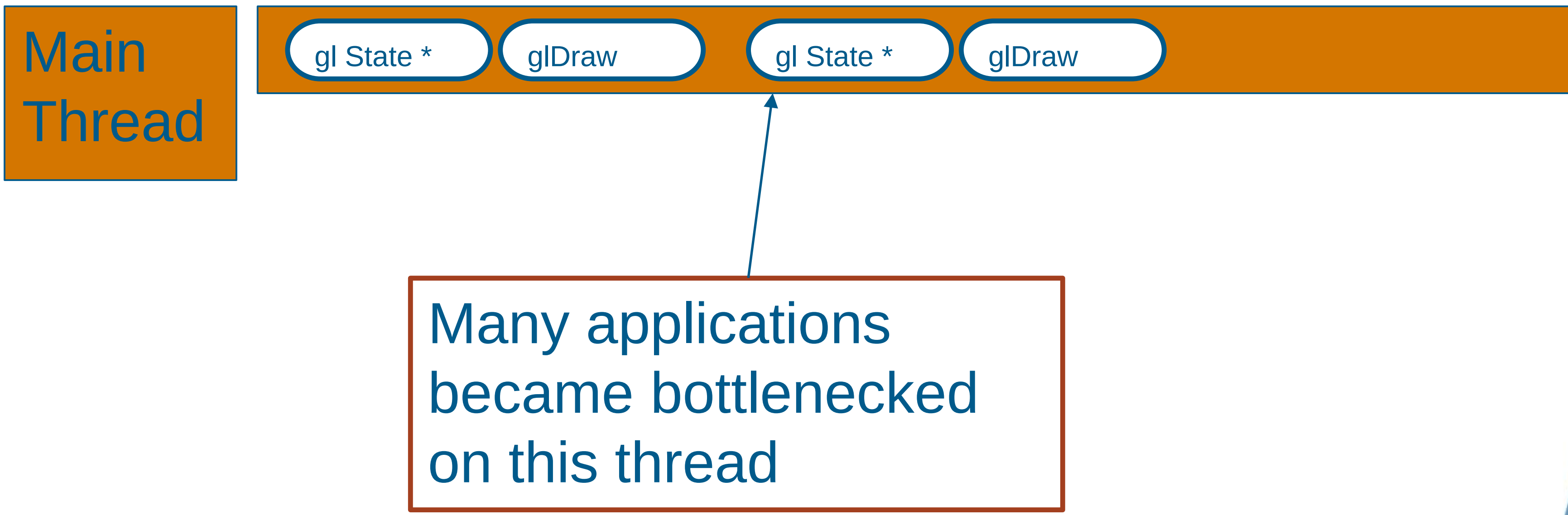
OpenGL/OpenGL ES Submission Model

- One Thread Submits to the API

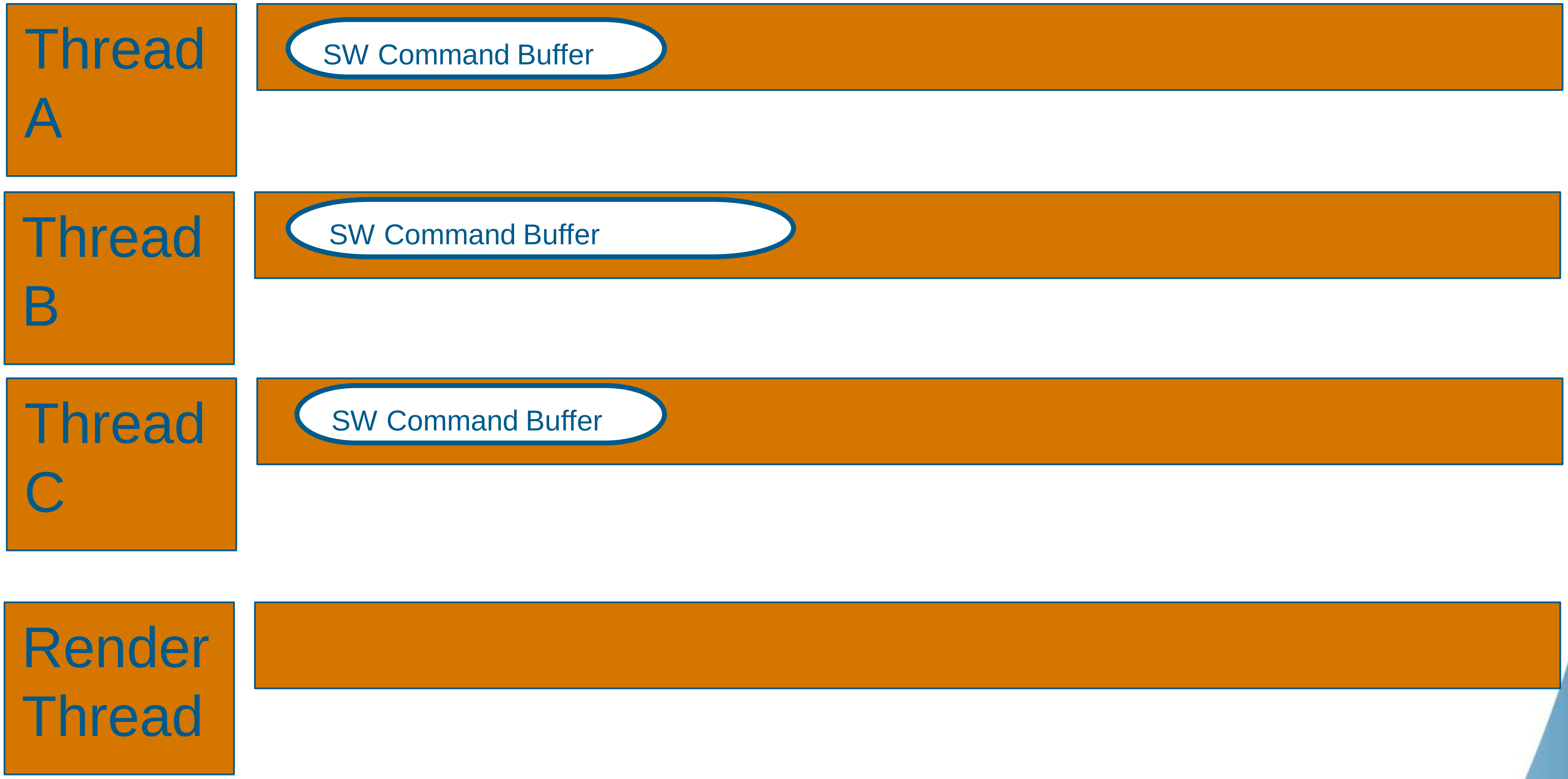


OpenGL/OpenGL ES Submission Model

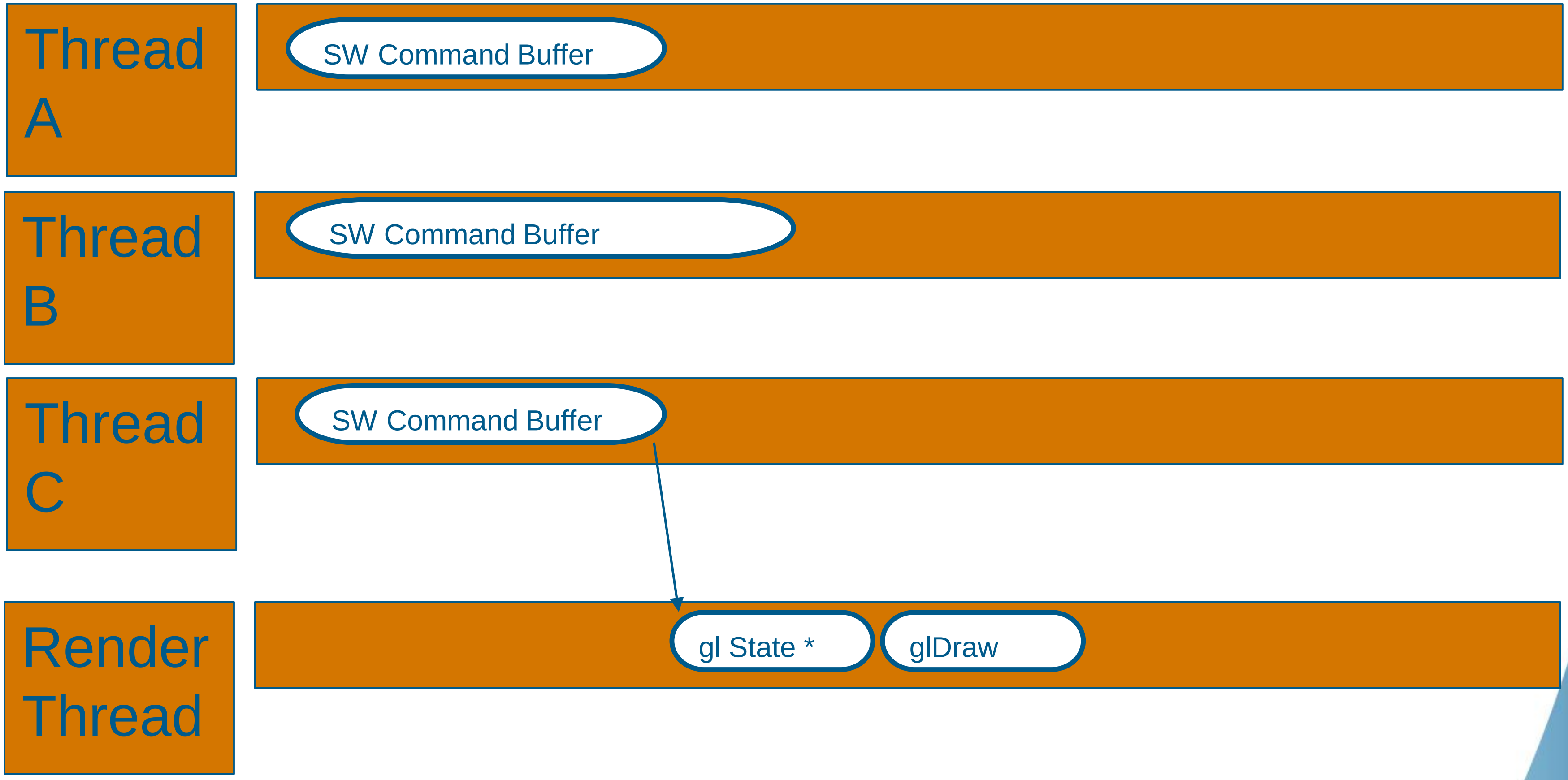
- One Thread Submits to the API



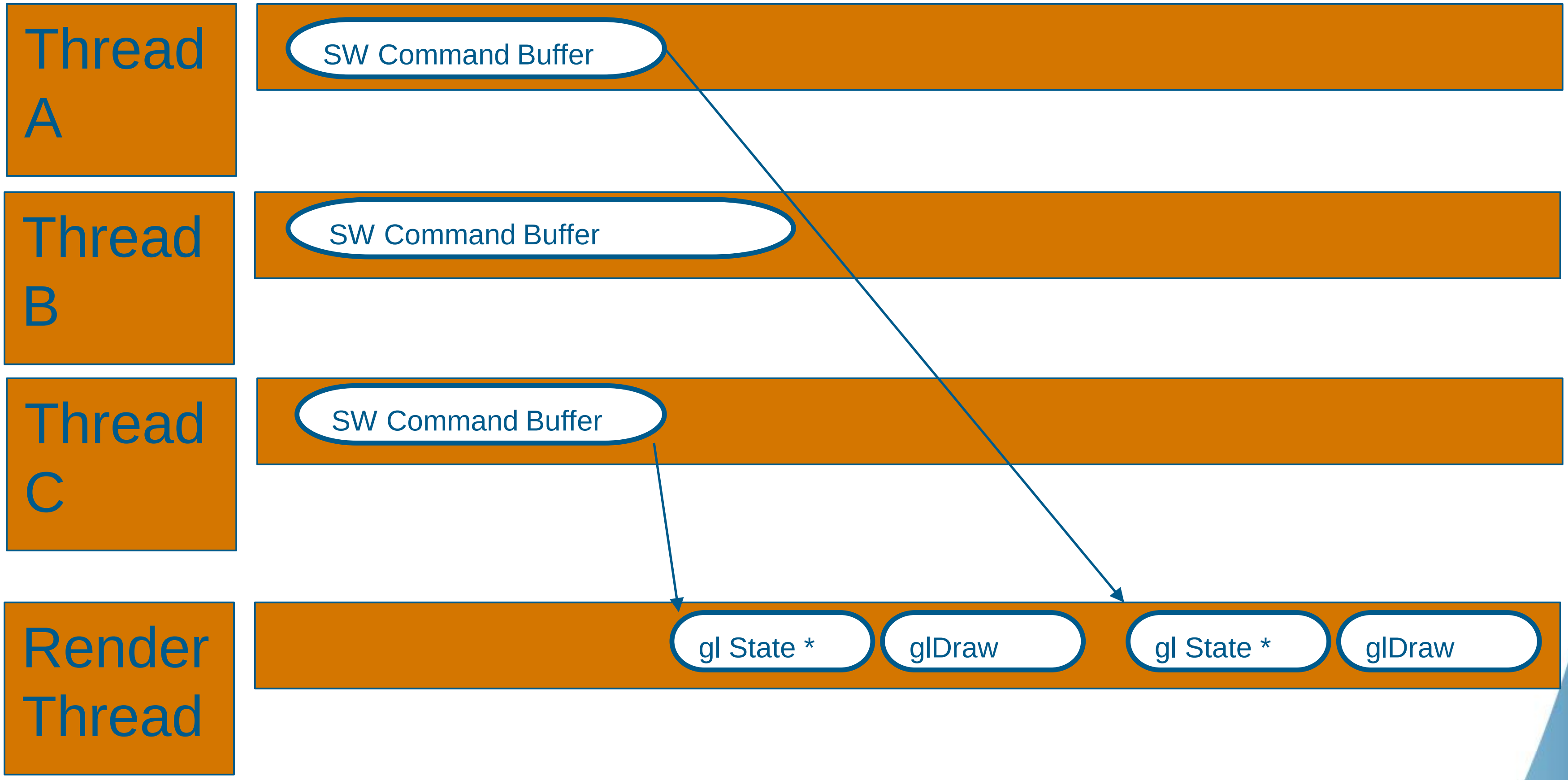
Mulithreading OpenGL/OpenGLES



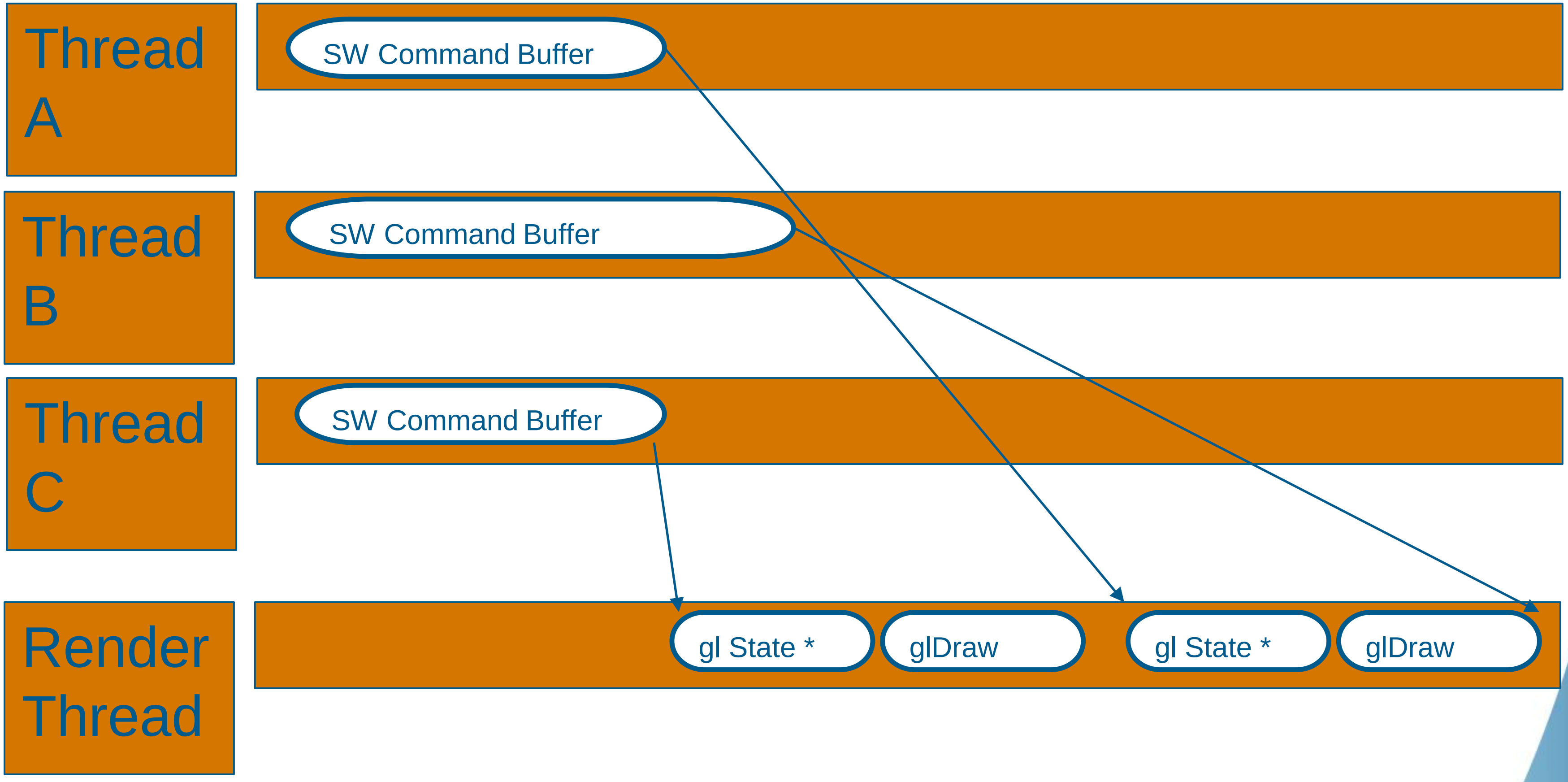
Mulithreading OpenGL/OpenGLES



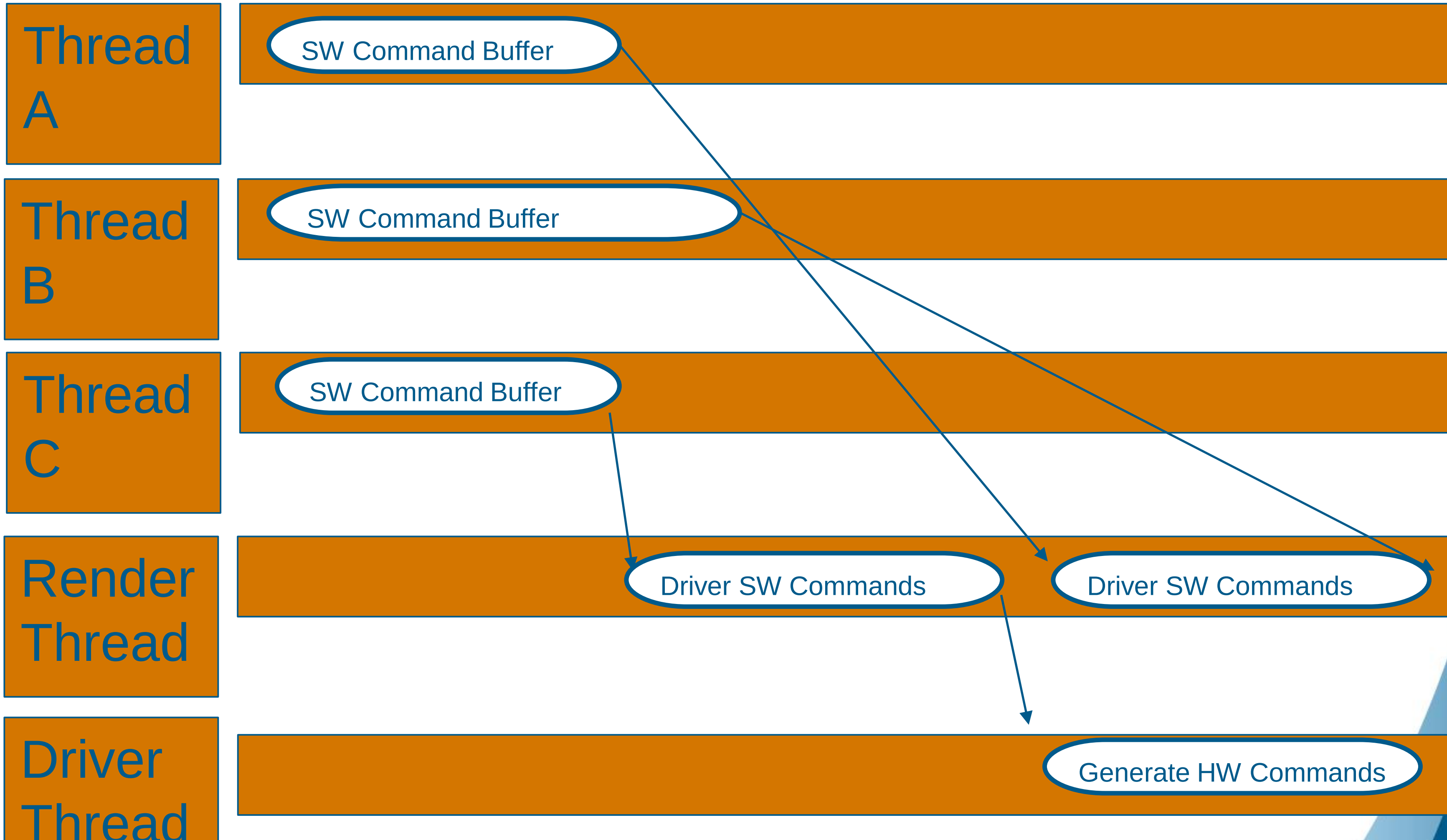
Mulithreading OpenGL/OpenGLES



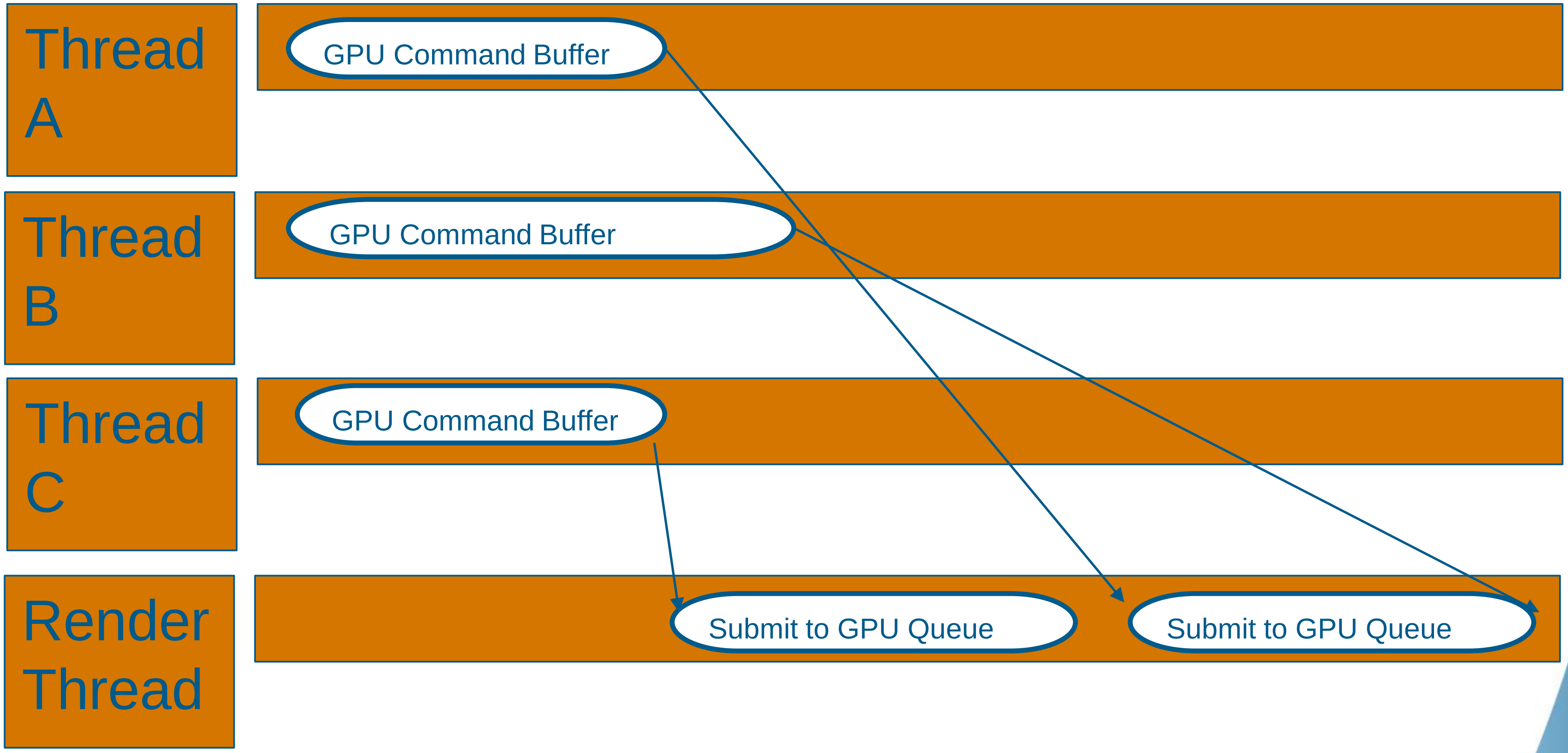
Mulithreading OpenGL/OpenGLES



Mulithreading OpenGL/OpenGLES



Vulkan Allows Multithreaded GPU Command Buffer Recording



OpenGL/OpenGL ES Problems

- State based recompiles
  - **Lead to frame time inconsistency (“hitches”)**

OpenGL/OpenGL ES Problems

- State based recompiles
  - **Lead to frame time inconsistency (“hitches”)**
- Expensive error checking
  - **Driver has to return GL_ERROR codes, expensive**

OpenGL/OpenGL ES Problems

- State based recompiles
  - **Lead to frame time inconsistency (“hitches”)**
- Expensive error checking
  - **Driver has to return GL_ERROR codes, expensive**
- Synchronization tracking
  - **When do caches flush?**
  - **When are images done being rendered to?**

OpenGL/OpenGL ES Problems

- State based recompiles
  - **Lead to frame time inconsistency (“hitches”)**
- Expensive error checking
  - **Driver has to return GL_ERROR codes, expensive**
- Synchronization tracking
  - **When do caches flush?**
  - **When are images done being rendered to?**
- Memory Allocation
  - **Where should resources get stored?**
  - **What to do if oversubscribed?**

OpenGL/OpenGL ES Problems

- State based recompiles
  - **Lead to frame time inconsistency ("hitches")**
- Expensive error checking
  - **Driver has to return GL_ERROR codes, expensive**
- Synchronization tracking
  - **When do caches flush?**
  - **When are images done being rendered to?**
- Memory Allocation
  - **Where should resources get stored?**
  - **What to do if oversubscribed?**
- Shader compilation
  - **Each driver needs to parse GLSL/ESSL – error prone**

OpenGL/OpenGL ES Problems

- State based recompiles
  - **Lead to frame time inconsistency (“hitches”)**
- Expensive error checking
  - **Driver has to return GL_ERROR codes, expensive**
- Synchronization tracking
  - **When do caches flush?**
  - **When are images done being rendered to?**
- Memory Allocation
  - **Where should resources get stored?**
  - **What to do if oversubscribed?**
- Shader compilation
  - **Each driver needs to parse GLSL/ESSL – error prone**
- Tile Based Renderers (TBR – Mobile)
  - **Massive driver heuristics/heroics**

OpenGL/OpenGL ES Problems – Vulkan Solutions

- State based recompiles
  - **Vulkan: Pipeline State Objects**
- Expensive error checking
  - **Vulkan: Move error checking to layers**
- Synchronization tracking
  - **Vulkan: Explicit synchronization**
- Memory Allocation
  - **Vulkan: Explicit Memory Allocation**
- Shader compilation
  - **Vulkan: SPIR-V**
- Tile Based Renderers (TBR – Mobile)
  - **Vulkan: RenderPasses**

UMASS BOSTON

Should I use Vulkan or GL?

- Probably yes:
  - **CPU perf/high draw call throughput**
  - **Multithreaded**
  - **Frame consistency**
  - **Explicit control**
  - **Fancy GPU features (i.e. async compute)**
  - **Targeting Mobile TBRs**
- Maybe not:
  - **Fully GPU bound in shading work**
    - Vulkan may make little/no difference other than power consumption
  - **Need quick results**

Vulkan API Concepts

Vulkan Graphics API Concepts

- Command Buffer
- Queue
- Synchronization Primitives
- Shaders
- Pipeline State Object
- Descriptor Sets
- RenderPass
- Barrier
- Memory Allocation

Command Buffer

- Per-thread command recorder
- Primary or secondary
  - **Secondary for parallelizing within a renderpass (more on this later)**
- No state inheritance across boundaries
- Submit to queue

Queues

- Abstraction for hardware command queue
- vkQueueSubmit
  - **Submit command buffer(s) to GPU**
- Common case: one queue
- Other possibilities:
  - **Graphics and compute queue (async compute)**
  - **Transfer Queue**

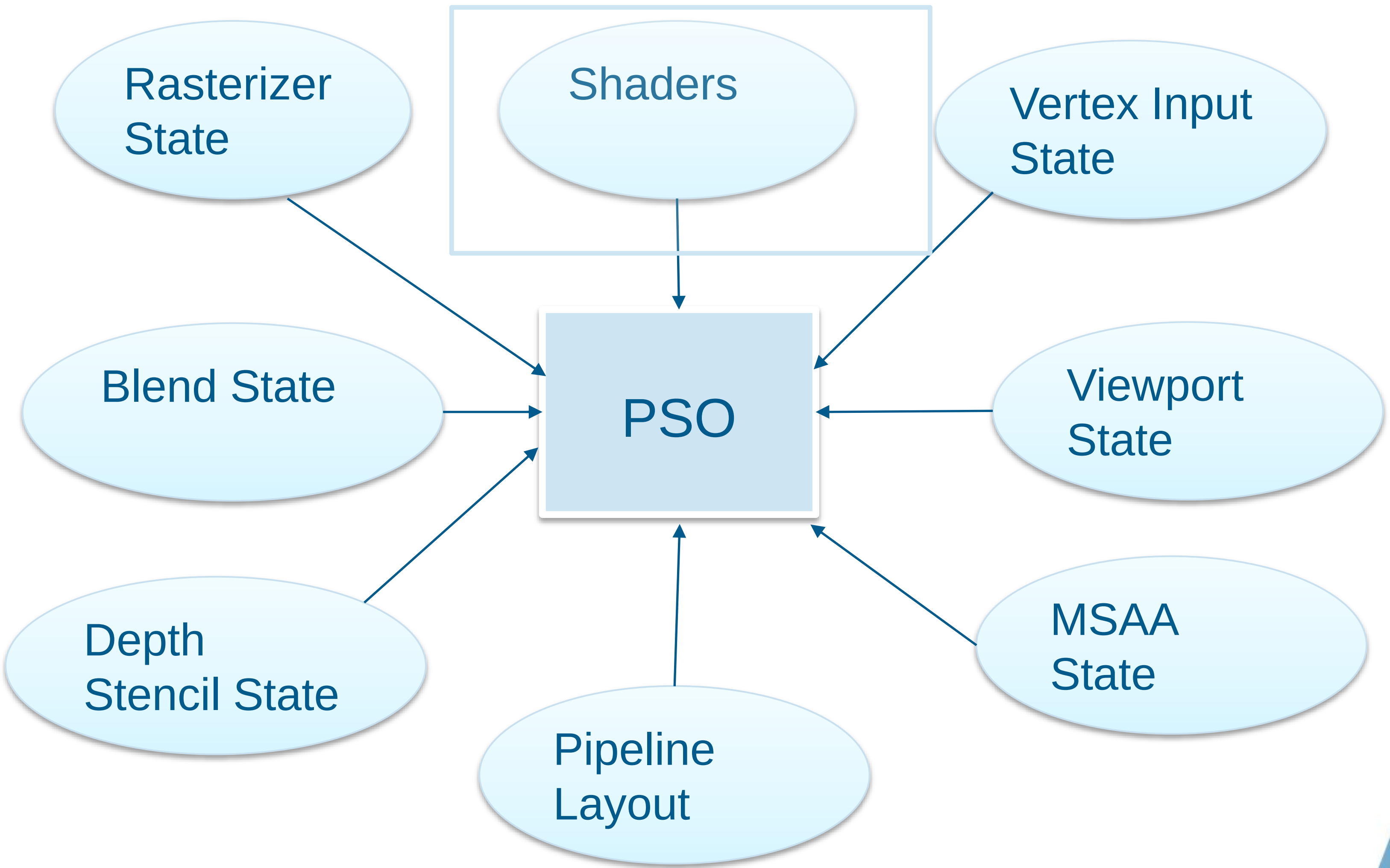
Synchronization Primitives

- Fences
  - **Determine when queue work has finished on CPU**
- Semaphore
  - **Wait for work on the GPU**
  - **Synchronize across multiple submits**
  - **Synchronize across multiple queues**
- Events
  - **Wait inside a command buffer**
  - **Signal from GPU or CPU**

Shaders

- SPIR-V
  - **Intermediate Representation (IR)**
  - **Single Static Assignment (SSA)**
- Lots of high level languages to choose from
  - **GLSL**
  - **HLSL**
  - **OpenCL C (sort of)**

Pipeline State Object (PSO)



Descriptor Sets

- Inputs to the PSO
  - **Textures**
  - **Uniforms**
  - **Storage Buffers**
  - **Etc.**
- Descriptor Set Layout
  - **Describes layout of shader inputs**
- Pipeline Layout
  - **Combines descriptor sets to define all inputs to PSO**
- Descriptor Set
  - **Populated with bindings to resources**

RenderPass

- First we should talk about tile based renderers (TBR)…

Tile Based Renderers

- Mobile GPUs have limited external memory bandwidth
  - **Slow**
  - **Uses a lot of battery**
- Mobile GPUs have fast internal memory
  - **Qualcomm Adreno calls this GMEM**
  - **ARM Mali and Apple GPUs have similar ideas**
  - **(some desktop GPUs are becoming tile based too)**
- Breaks frame up into bins or tiles

Tile Based Renderer

Tile Based Renderer

Tile Based Renderer

Rendering in two passes

- Binning Pass
  - **Run position portion of vertex shader to compute clip space position**
  - **Store which bin each vertex is in**
- For each tile
  - **Load tile with initial value**
    - LOAD – restore previous contents
    - DON’T_CARE – ignore previous contents
    - CLEAR – clear to solid color
  - **Process primitives matching that bin**
  - **Shade fragments, blend, etc.**
  - **Store tile results**
    - STORE – store to external memory
    - DON’T_CARE – throw away

Back to RenderPasses

- RenderPass defines this process
  - **What attachments are we rendering to?**
  - **What should the initial value of tile memory be?**
  - **What should we do with the results?**
- Immediate Mode Renderers
  - **This is basically BindRenderTarget/DiscardRenderTarget, but can still be useful**
- This is an overly simple explanation
  - **RenderPasses can actually represent frame graph**
    - Subpasses
    - Subpass dependencies

Barriers

- Express data dependency
- Examples
  - **Render to Texture, Bind as Shader Resource**
    - Image Barrier: COLOR_ATTACHMENT_OPTIMAL -> SHADER_READ_ONLY_OPTIMAL
    - Makes sure outstanding writes finish before reads
  - **Copy to Buffer, Read as Vertex Buffer**
    - Memory Barrier: TRANSFER_DST -> VERTEX_INPUT
- Pretty hard to get right, recommend reading this:
  - **http://themaister.net/blog/2019/08/14/yet-another-blog-explaining-vulkan-synchronization/**

Memory Allocation

- Vulkan Driver provides heaps and memory types
- Query to figure out which heaps can be used for allocation
- Allocate and bind memory to resource
- Application does all pooling
  - **You have to write your own memory manager**
  - **Or just use VMA: https://github.com/GPUOpen-LibrariesAndSDKs/VulkanMemoryAllocator**

Vulkan Getting Started

- Bad news:
  - **It’s hard**
- Good news:
  - **There are tons of resources**
  - **It’s all open source**

Vulkan Resources

- Vulkan SDK (Windows/Linux/macOS):
  - **https://vulkan.lunarg.com/**
  - **Validation Layers**
  - **Shader Compilers**
    - glslang (GLSL or HLSL SM3-5.1 -> SPIR-V)
    - DXC (HLSL SM6+)
  - **SPIRV-Tools**
  - **spirv-cross**
    - SPIR-V -> MSL/HLSL/GLSL
  - **MoltenVK (Vulkan on Metal)**
    - macOS + iOS
    - Dota Underlords shipped on it!
- What about Android?
  - **Part of NDK**
  - **Supported since Android 7**

Vulkan Repos

- glslang
  - **https://github.com/KhronosGroup/glslang**
- SPIR-V Tools
  - **https://github.com/KhronosGroup/SPIRV-Tools**
- Vulkan Validation Layers
  - **https://github.com/KhronosGroup/Vulkan-ValidationLayers**
- DirectXShaderCompiler (SM6 HLSL)
  - **https://github.com/microsoft/DirectXShaderCompiler**
- MoltenVK
  - **https://github.com/KhronosGroup/MoltenVK**

Other Useful Resources

- Vulkan Samples
  - https://github.com/KhronosGroup/Vulkan-Samples
- Sascha Willems Examples
  - https://github.com/SaschaWillems/Vulkan
- VMA (Vulkan Memory Allocator)
  - https://github.com/GPUOpen-LibrariesAndSDKs/VulkanMemoryAllocator
- Vulkan GPU Database
  - http://vulkan.gpuinfo.org/

Questions?

dginsburg@upsamplesoftware.com