Siggraph 2016

VULKAN AND NVIDIA: THE ESSENTIALS

Tristan Lorach Manager of Developer Technology Group, NVIDIA US

7/25/2016

PRESENTED BY

NVIDIA.

ANALOGY ON GRAPHIC APIS

(getting ready for my 7 years old son's questions on my job…)

Car Toy

Lego Kit

Derby Kit

Analogy

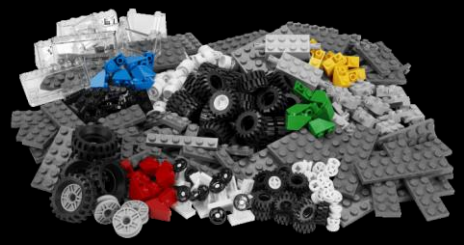
Different Valid Approaches

(booring…)

(cool... Messes-up the bedroom)

(adult supervision required!)

Fixed-function OpenGL



Modern AZDO OpenGL with Programmable Shaders

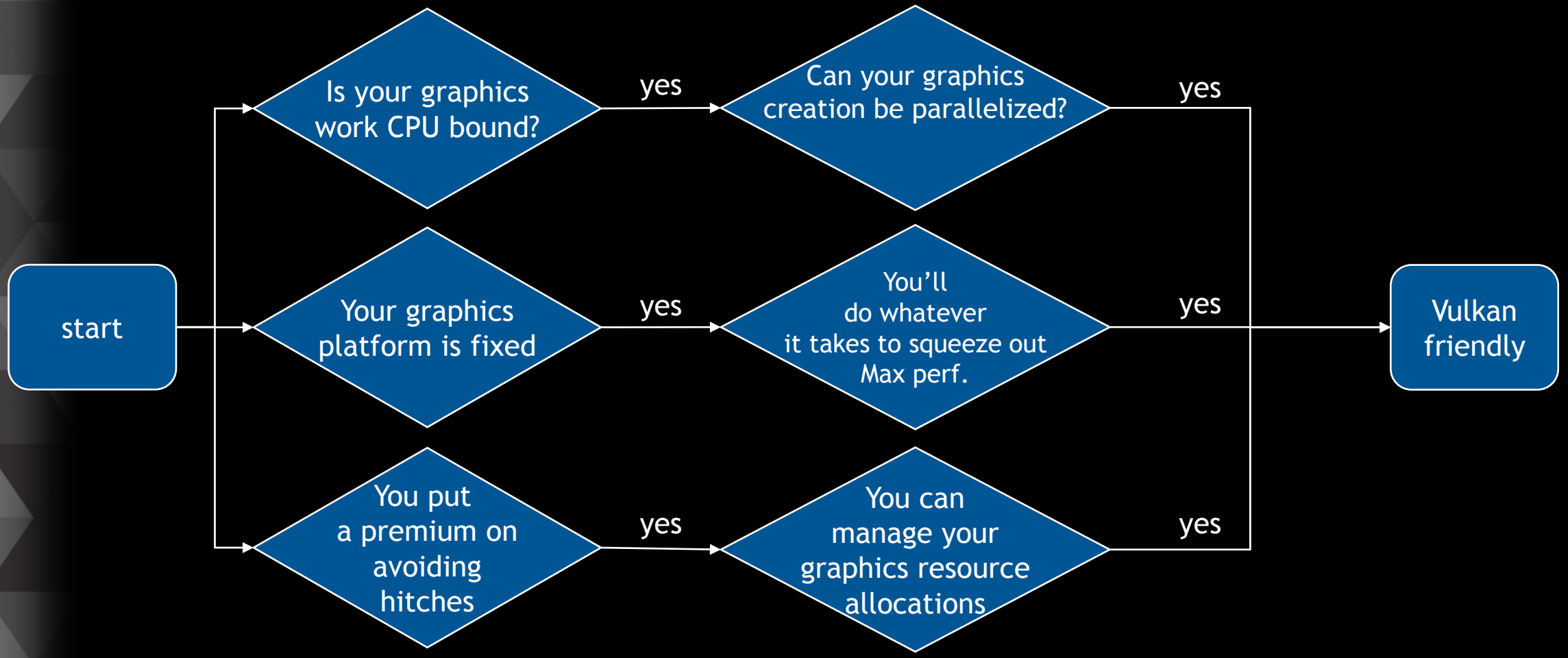
Vulkan

WHAT IS VULKAN ?

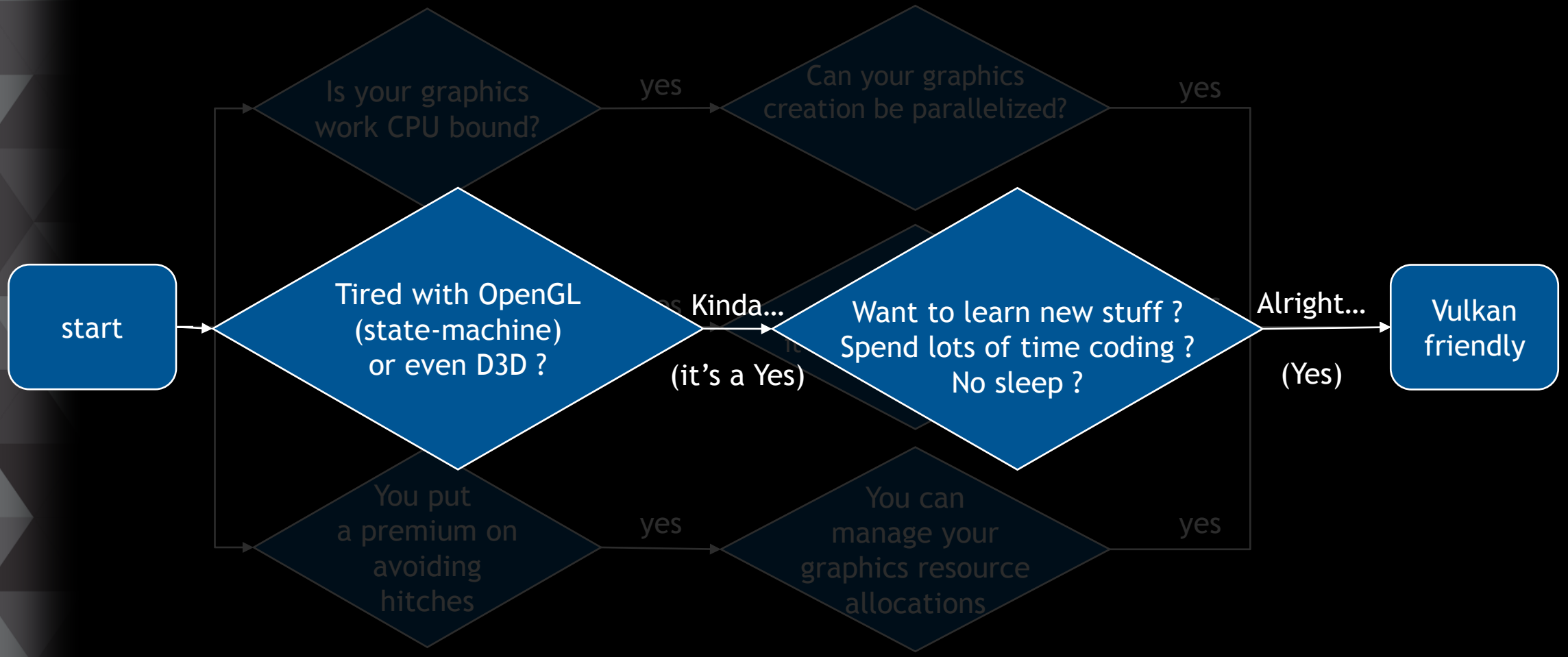
…It's a modern API

- Designed and maintained by Khronos Group
- Designed for high performance on rendering and compute
- [Extremely] low level : no more "baby-sitting" from our driver
  - Manage yourself memory, resource updates; batching; scheduling…
- [Extremely] verbose : Lots of structures to fill with parameters
- close to DX12 design…
- Opposite of OpenGL: Multi-threading friendly : Vulkan will especially shine if multi-threading used
- But still generic enough to work on many HW vendors & platforms

Beneficial Vulkan Scenarios



Beneficial Vulkan Scenarios



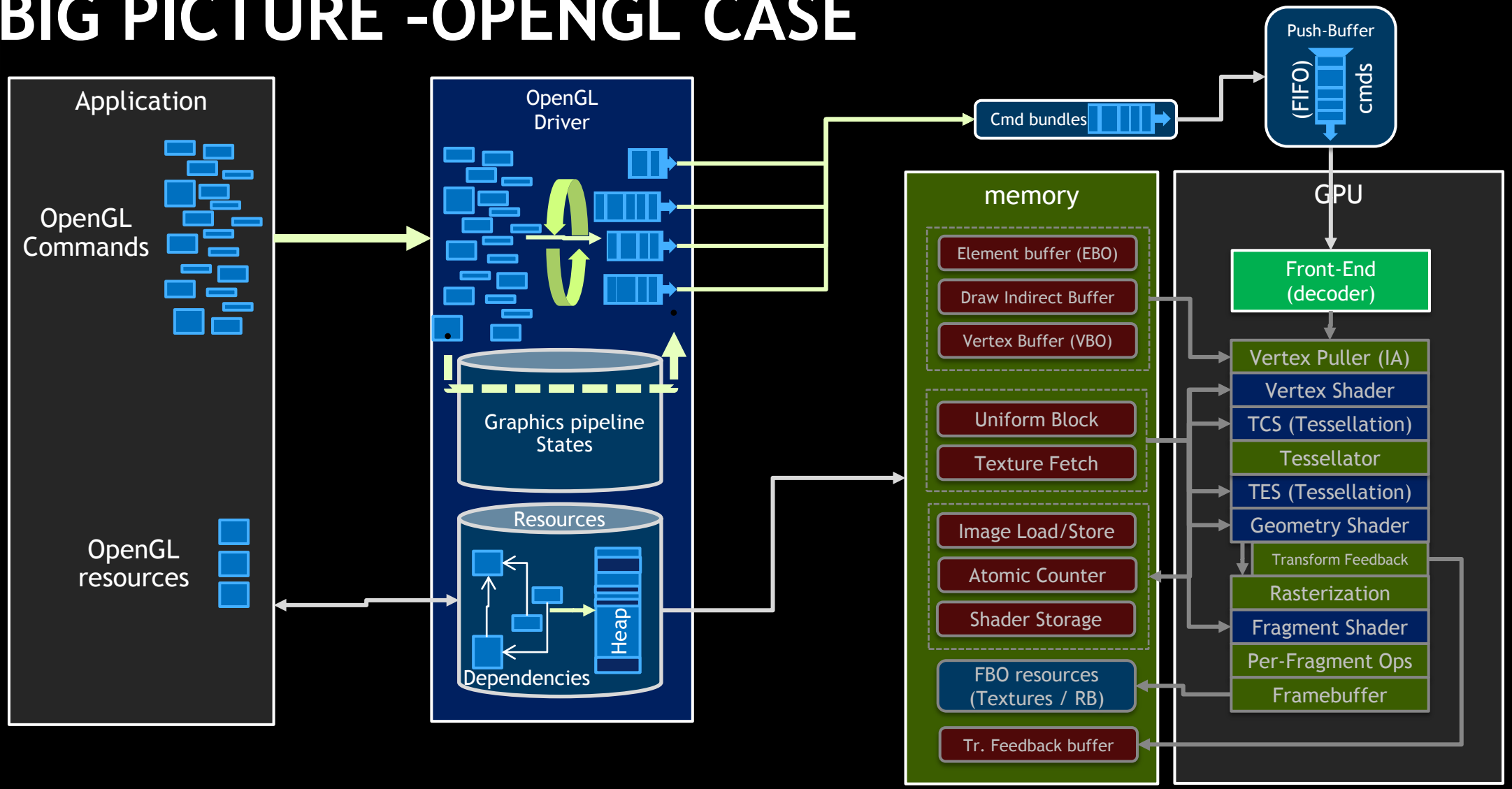
Unlikely to Benefit

Scenarios to Reconsider Coding to Vulkan

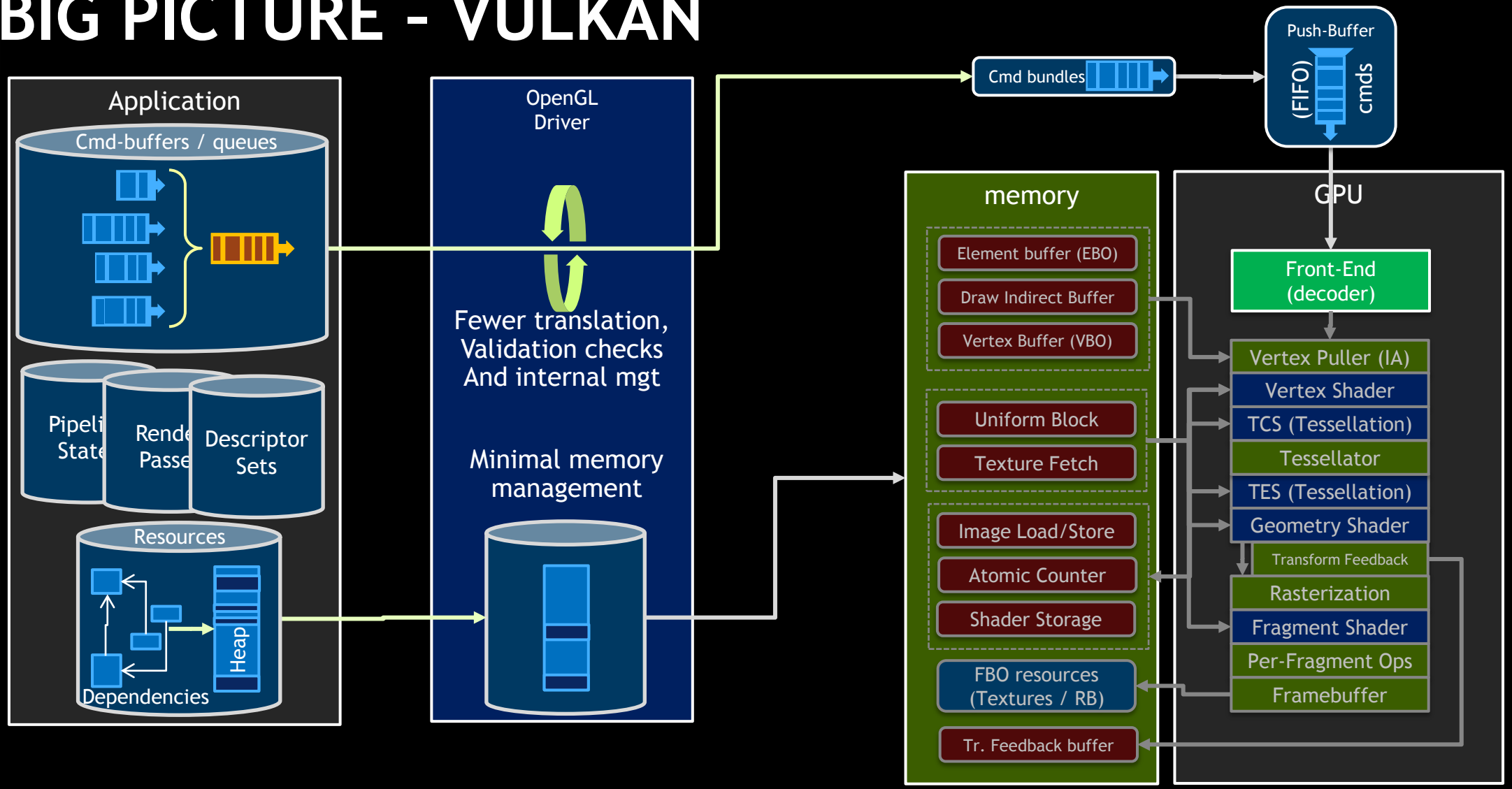
1. Need for compatibility to pre-Vulkan platforms
2. Heavily GPU-bound application
3. Heavily CPU-bound application due to non-graphics work
4. Single-threaded application, unlikely to change
5. App can target middle-ware engine, avoiding 3D graphics API dependencies
    - Consider using an engine targeting Vulkan, instead of dealing with Vulkan yourself

Good News in any case: NVIDIA OpenGL driver is great and will always be there !

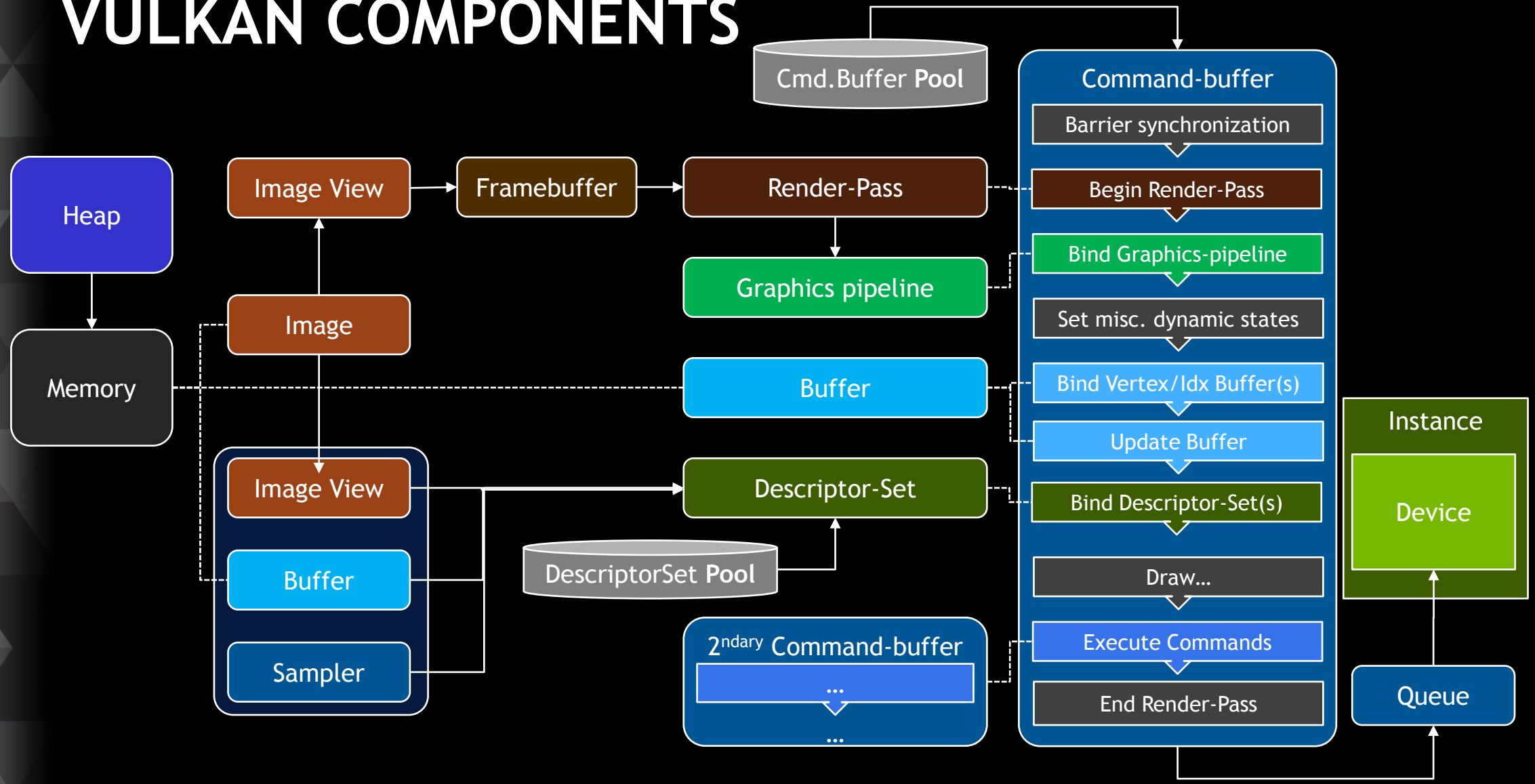
BIG PICTURE -OPENGL CASE



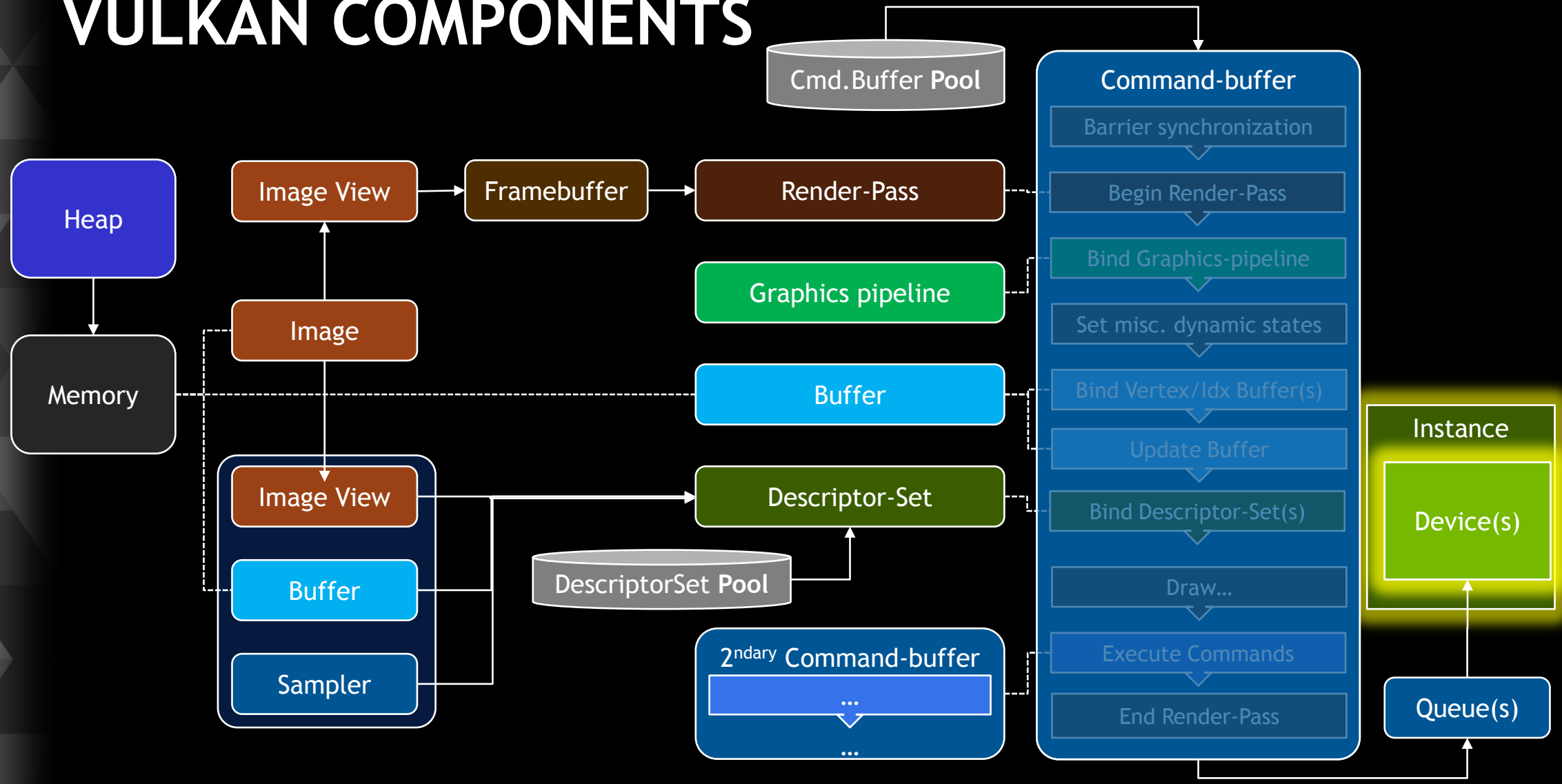
BIG PICTURE – VULKAN



VULKAN COMPONENTS



VULKAN COMPONENTS



VULKAN OBJECTS: DEVICE

Instance ~⇔~ OpenGL Context

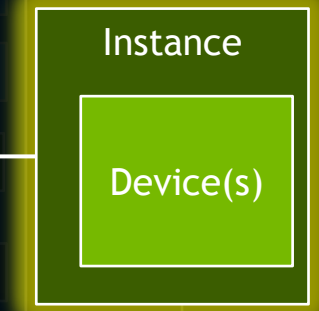
Instance-Layers

- Intercepting API calls for misc. purposes
- Many layers available (api-dump; core/std/parms validation; screenshot…)

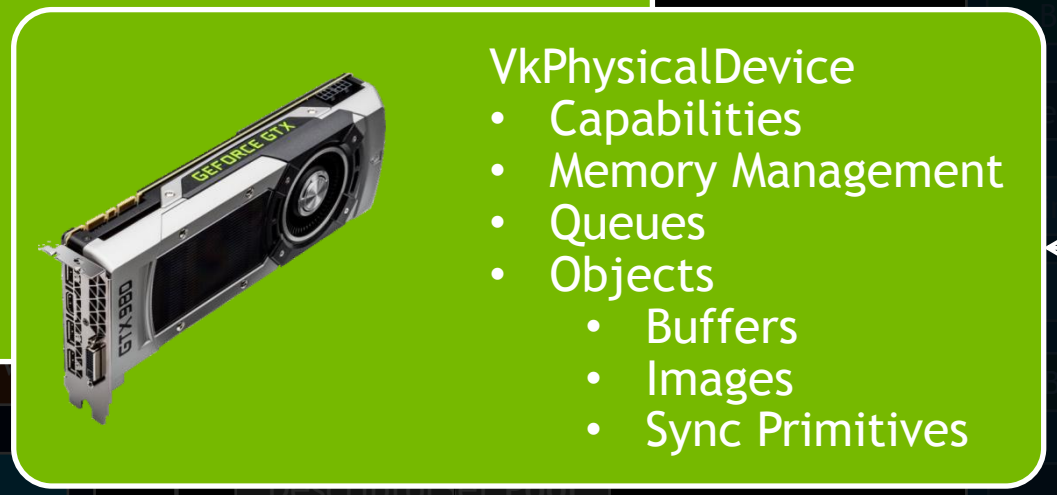
Instance-Specific Extensions

- KHR_Surface (for Swap-chains)
- EXT_debug_report
- …

Exposes some Devices…



VULKAN OBJECTS: DEVICE



VkPhysicalDevice
- Capabilities
- Memory Management
- Queues
- Objects
  - Buffers
  - Images
  - Sync Primitives

Can have many …

Device(s)

NVIDIA'S VULKAN CAPABILITIES

Properties listed from Physical Device

NVIDIA is almost full featured

- Top to bottom: from GeForce, Quadro down to Tegra

Check http://vulkan.gpuinfo.org/listreports.php

NVIDIA'S VULKAN CAPABILITIES

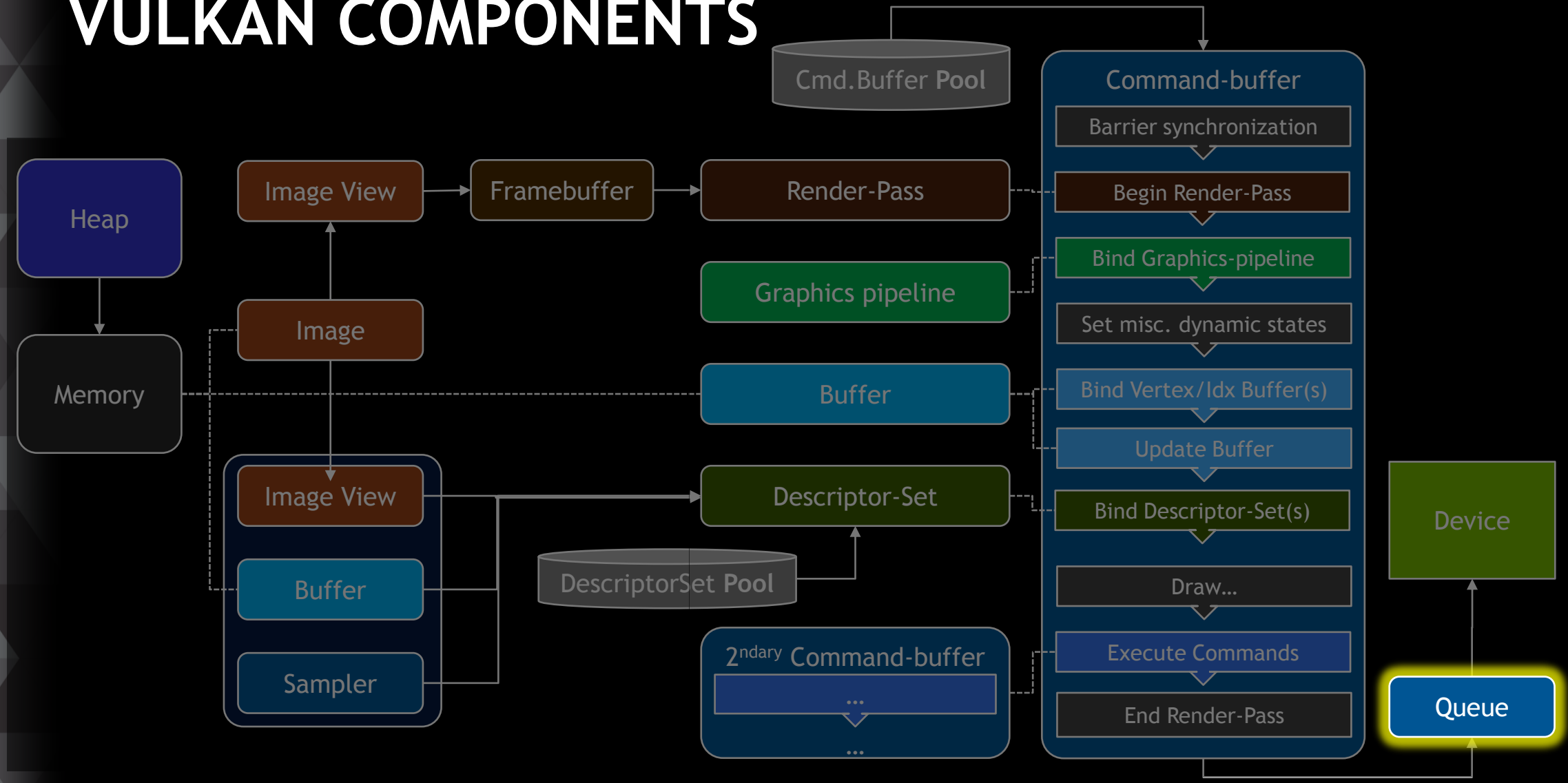
GeForce GTX 980

Feature	Value
alphaToOne	true
depthBiasClamp	true
depthBounds	true
depthClamp	true
drawIndirectFirstInstance	true
dualSrcBlend	true
fillModeNonSolid	true
fragmentStoresAndAtomics	true
fullDrawIndexUint32	true
geometryShader	true
imageCubeArray	true
independentBlend	true
inheritedQueries	true
largePoints	true
logicOp	true
multiDrawIndirect	true
multiViewport	true
occlusionQueryPrecise	true
pipelineStatisticsQuery	true
robustBufferAccess	true
sampleRateShading	true
samplerAnisotropy	true
shaderClipDistance	true
shaderCullDistance	true
shaderFloat64	true
shaderImageGatherExtended	true
shaderInt16	false
shaderInt64	true
shaderResourceMinLod	true
shaderResourceResidency	true
shader	
shader	
shader	
shader	
shader	
shader	
shader	
shader	
shader	
sparse	
sparse	
sparse	
sparse	
sparse	
sparse	
sparse	
sparse	
sparse	
tessella	
texture	
texture	
texture	
variabl	
vertexP	
wideLin	

Tegra X1 & K1

Feature	Report 3	Report 78
device	NVIDIA NVIDIA Tegra X1	NVIDIA NVIDIA Tegra K1
version	361.0.0 (1.0.2)	361.0.0 (1.0.2)
os	android 6.0 (arm)	android 6.0.1 (arm)
alphaToOne	true	true
depthBiasClamp	true	true
depthBounds	true	true
depthClamp	true	true
drawIndirectFirstInstance	true	true
dualSrcBlend	true	true
fillModeNonSolid	true	true
fragmentStoresAndAtomics	true	true
fullDrawIndexUint32	true	true
geometryShader	true	true
imageCubeArray	true	true
independentBlend	true	true
inheritedQueries	true	true
largePoints	true	true
logicOp	true	true
multiDrawIndirect	true	true
multiViewport	true	true
occlusionQueryPrecise	true	true
pipelineStatisticsQuery	true	true
robustBufferAccess	true	true
sampleRateShading	true	true
samplerAnisotropy	true	true
shaderClipDistance	true	true
shaderCullDistance	true	true
shaderFloat64	true	true
shaderImageGatherExtended	true	true
shaderInt16	false	false
shaderInt64	true	true
shaderResourceMinLod	true	false
shaderResourceResidency	true	false
shaderSampledImageArrayDynamicIndexing	true	true
shaderStorageBufferArrayDynamicIndexing	true	true
shaderStorageImageArrayDynamicIndexing	true	true
shaderStorageImageExtendedFormats	true	true
shaderStorageImageMultisample	true	true
shaderStorageImageReadWithoutFormat	true	false
shaderStorageImageWriteWithoutFormat	true	true
shaderTessellationAndGeometryPointSize	true	true
shaderUniformBufferArrayDynamicIndexing	true	true
sparseBinding	true	true
sparseResidency16Samples	true	false
sparseResidency2Samples	true	true
sparseResidency4Samples	true	true
sparseResidency8Samples	true	true
sparseResidencyAliased	true	true
sparseResidencyBuffer	true	true
sparseResidencyImage2D	true	true
sparseResidencyImage3D	true	true
tessellationShader	true	true
textureCompressionASTC_LDR	true	true
textureCompressionBC	true	true
textureCompressionETC2	true	true
variableMultisampleRate	true	true
vertexPipelineStoresAndAtomics	true	true
wideLines	true	true

VULKAN COMPONENTS



QUEUES

Command queue was hidden in OpenGL Context… now explitly declared

- Multiple threads can submit work to a queue (or queues)!

Queues accept GPU work via CommandBuffer submissions

- few operations available around Queues:, “submit work” and “wait for idle”

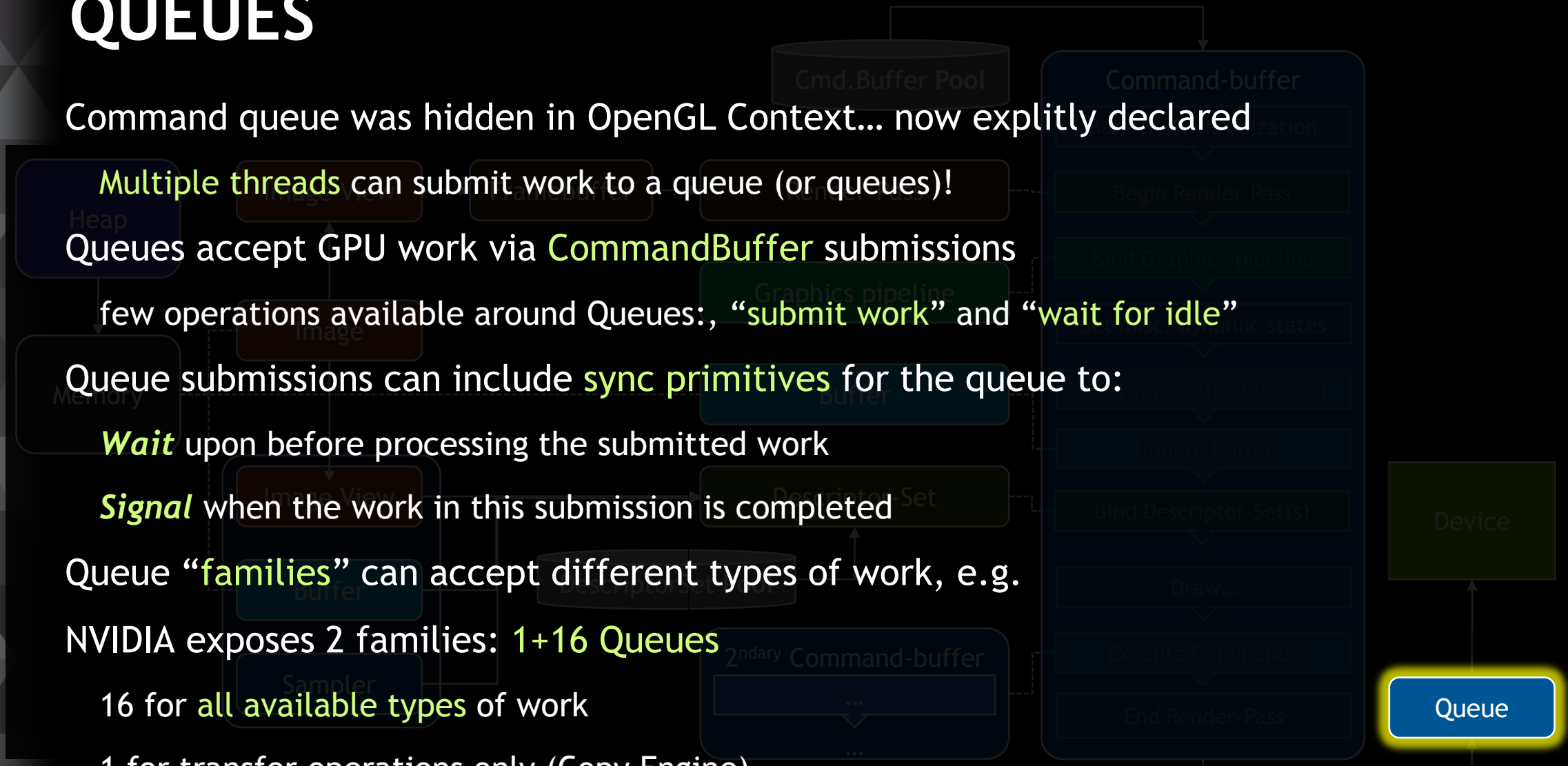
Queue submissions can include sync primitives for the queue to:

- ***Wait*** upon before processing the submitted work
- ***Signal*** when the work in this submission is completed

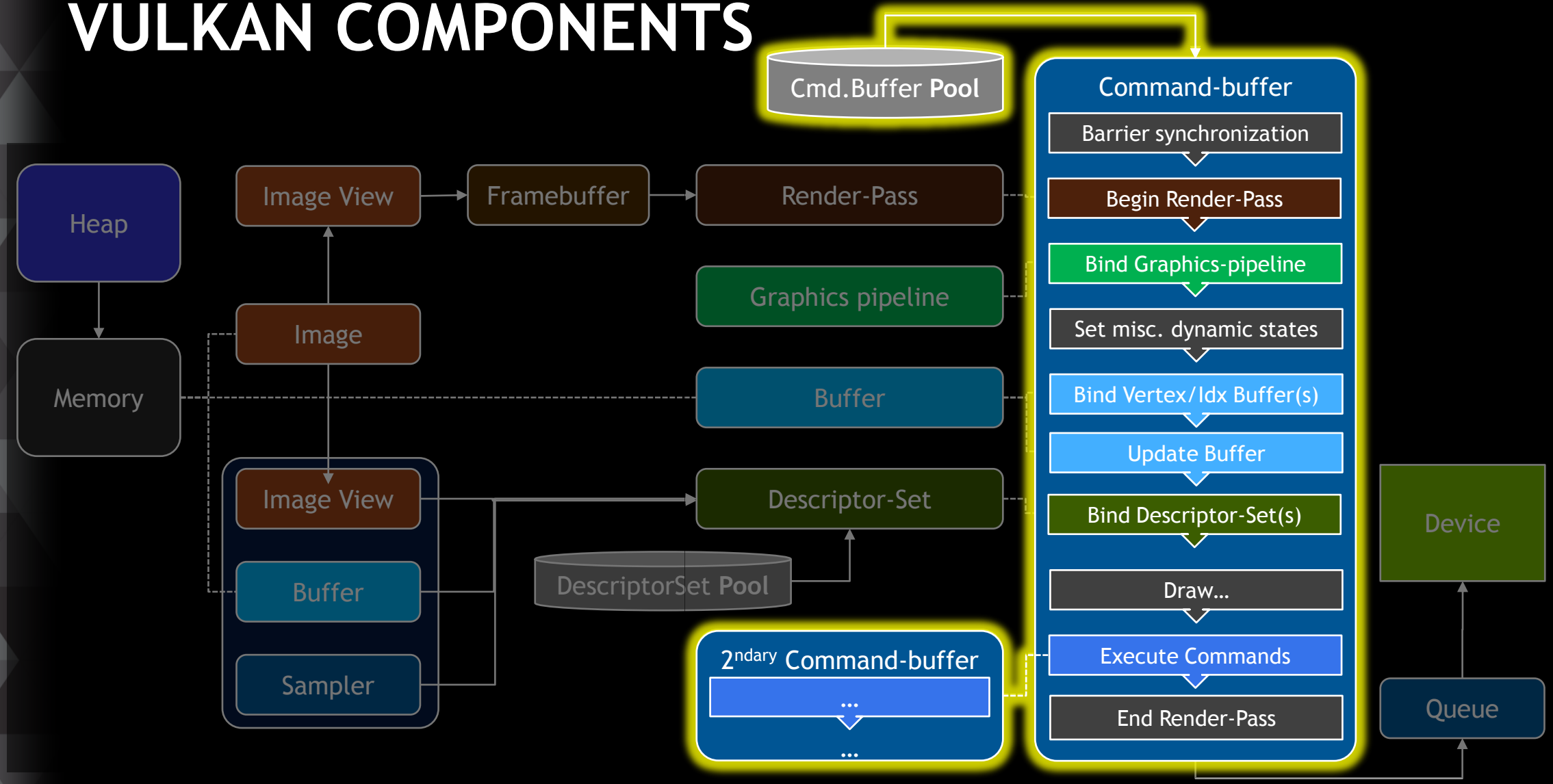
Queue “families” can accept different types of work, e.g.

NVIDIA exposes 2 families: 1+16 Queues

- 16 for all available types of work
- 1 for transfer operations only (Copy Engine)



VULKAN COMPONENTS



SYNCHRONIZATION

events and barriers

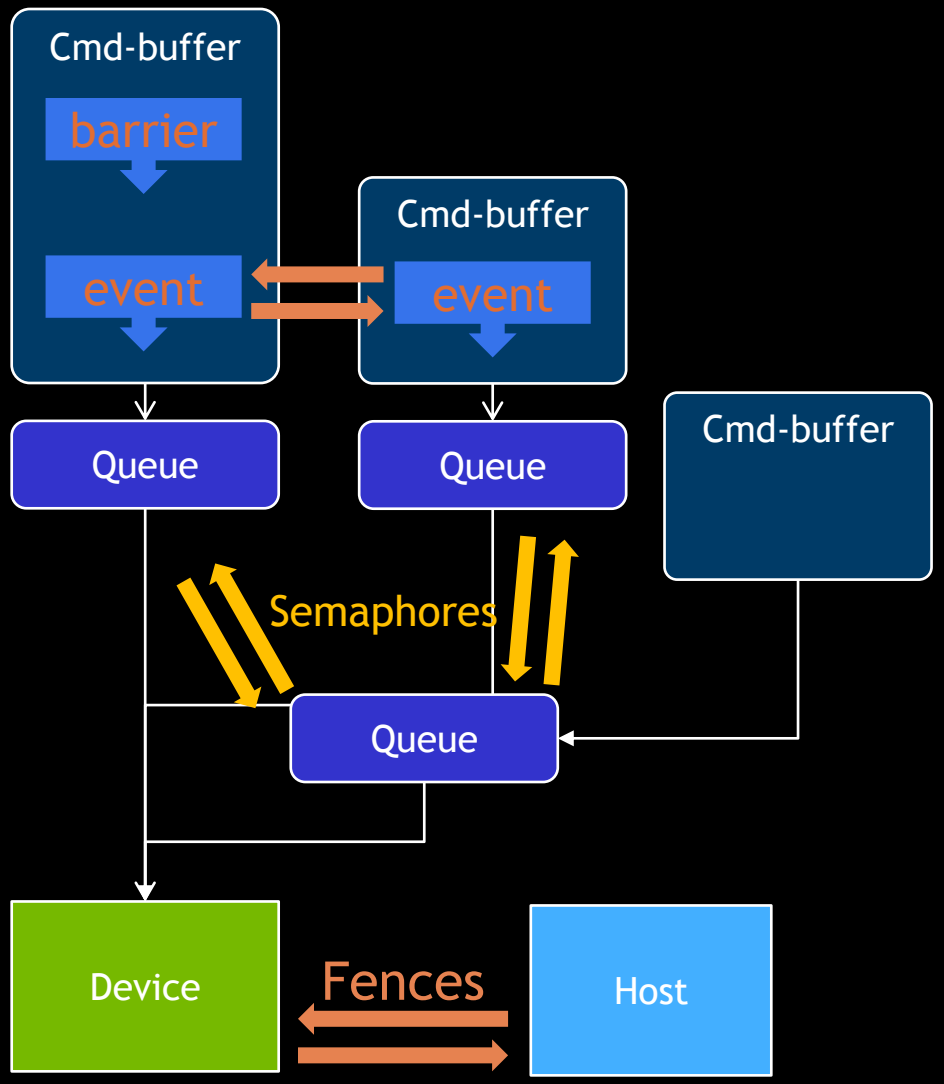
used to synchronize work within a command buffer or sequence of command buffers submitted to a single queue

semaphores

used to synchronize work across queues or across coarse-grained submissions to a single queue

fences

used to synchronize work between the device and the host.



COMMAND-BUFFERS

Vulkan Rendering ➔ Command-Buffers

Close to what GPU will get at Front-End (FIFO)

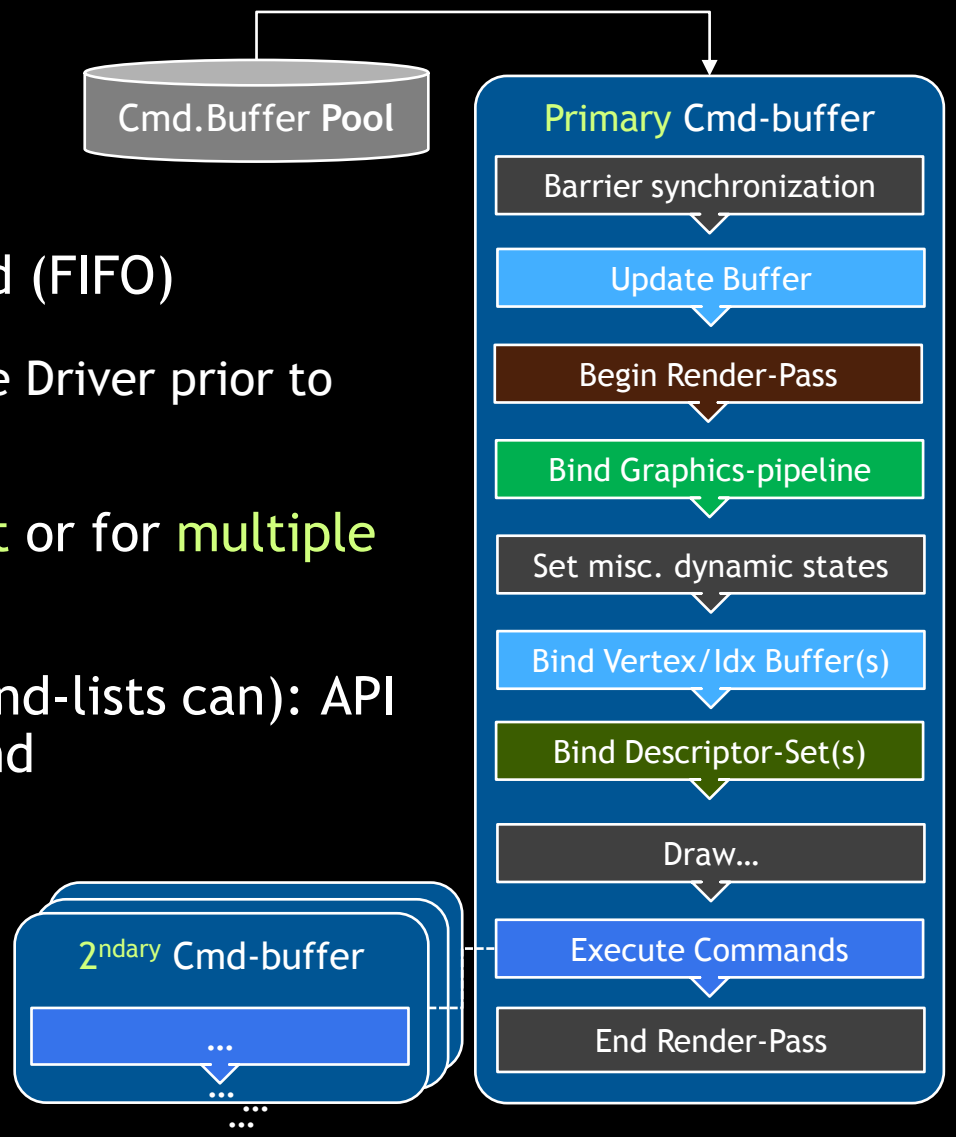
- Minor translation & optimization from the Driver prior to sending to the GPU

Each can be created either for one shot or for multiple frames/submissions

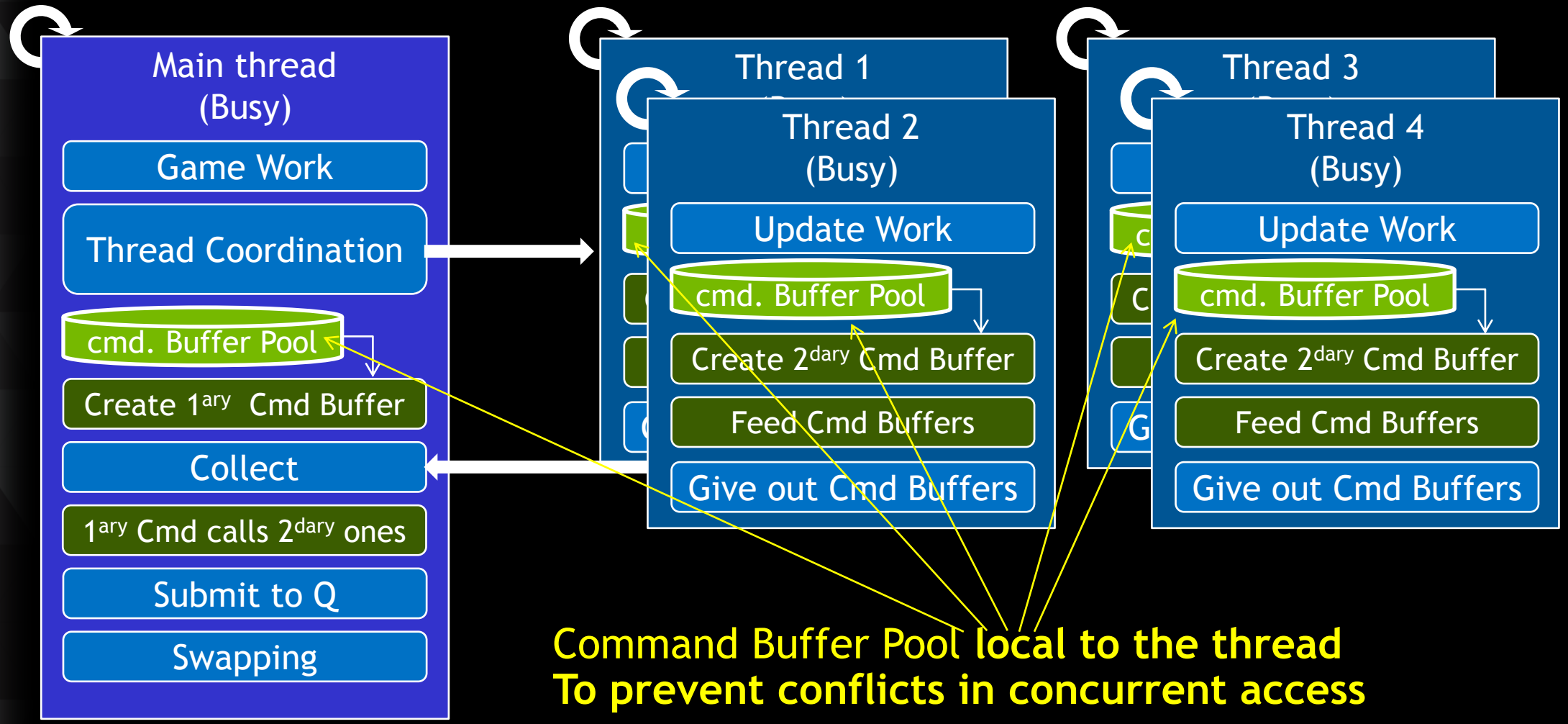
Cannot Cmd-Buffers from GPU (command-lists can): API calls to vkCmd…() between Begin & End

Multi-threading friendly (!)

Primary Cmd-Buffer can call many 2ndary Cmd-Buffers



COMMAND-BUFFERS AND MULTI-THREADING



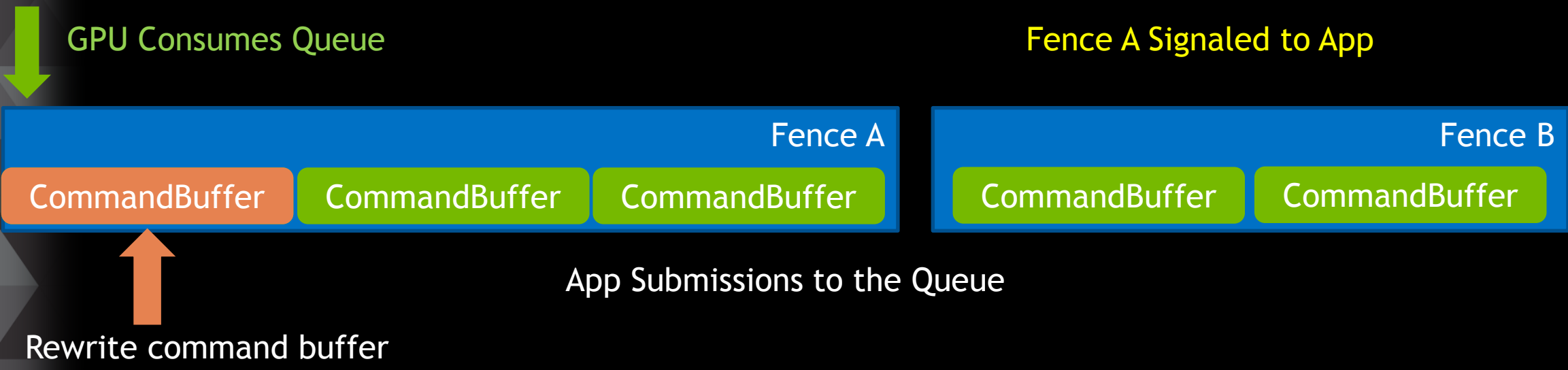
Command Buffer Pool **local to the thread**
To prevent conflicts in concurrent access

COMMAND BUFFER THREAD SAFETY

Must not recycle a CommandBuffer for rewriting until it is no longer in flight (In flight == GPU still consuming it on its side)

But we can't flush the queue each frame: would break parallelism !

VkFences can be provided with a queue submission to test when a command buffer is ready to be recycled



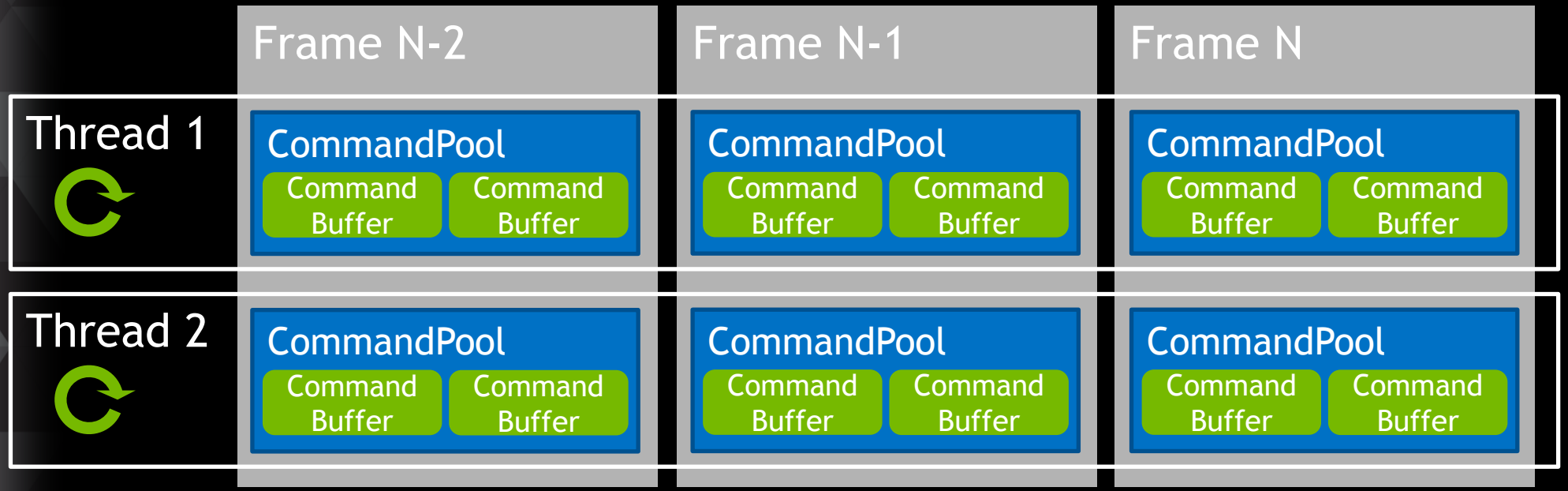
THREADS AND COMMAND POOLS

Threads can have more than 1 Command Pool

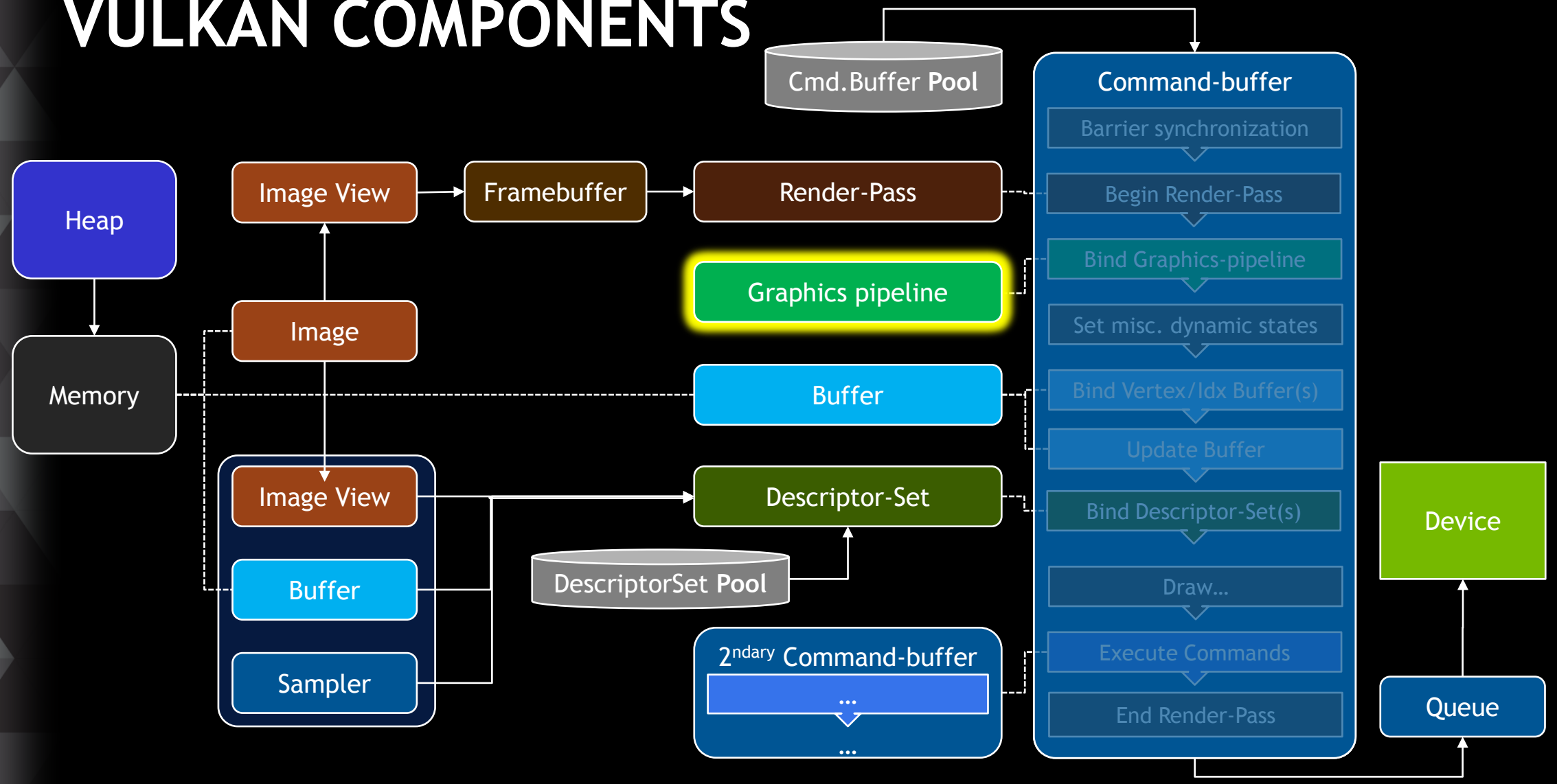
Ring-buffer: One Command-Pool per Frame

when the frame is no longer in flight (Using Fences):

simply reset the whole Pool



VULKAN COMPONENTS



GRAPHICS PIPELINE

Snapshot of all States

Including Shaders

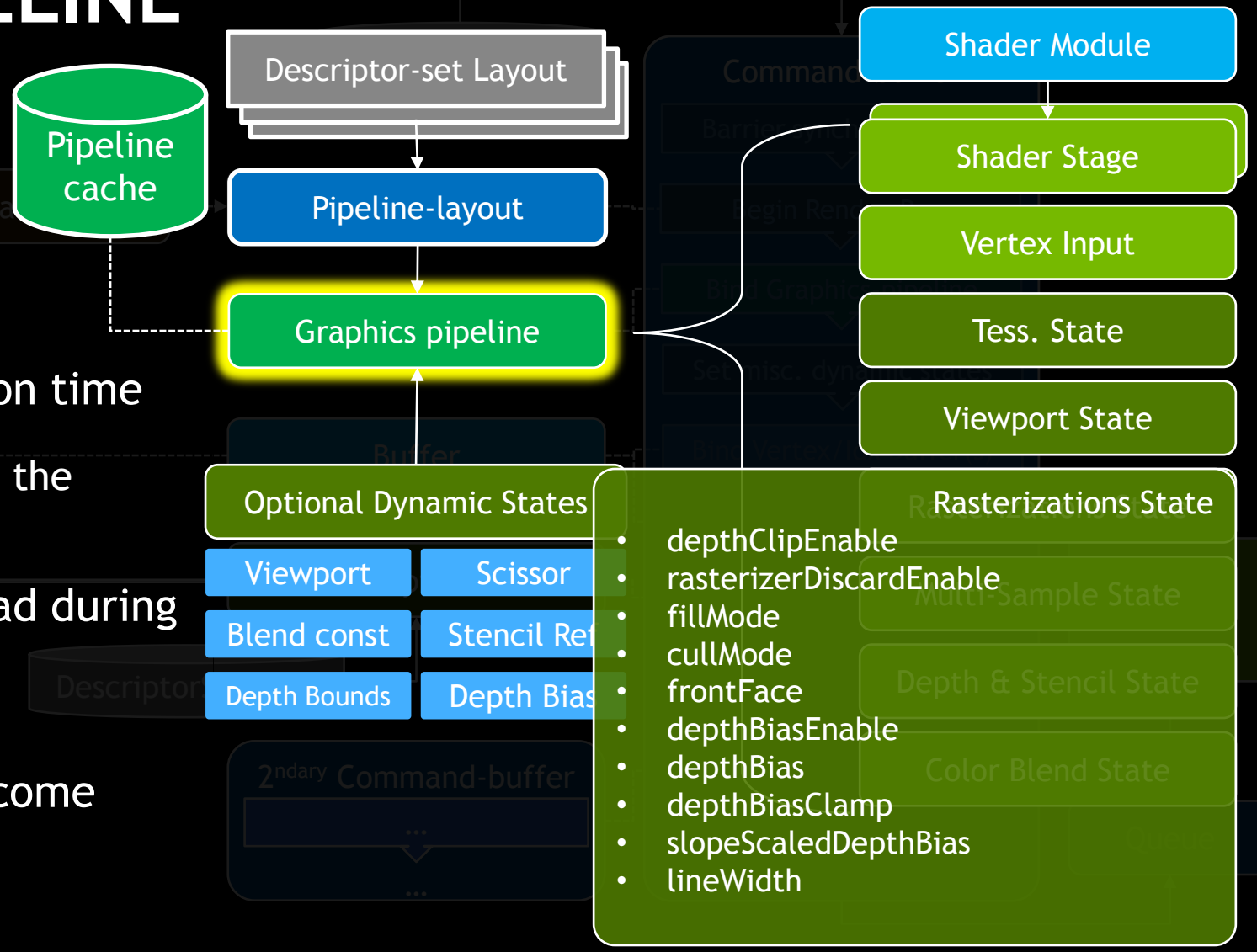
Pre-compiled & Immutable

Ideally: done at Initialization time

Ok at render-time *if* using the Pipeline-Cache

Prevents validation overhead during rendering loop

Some Render-states can be excluded from it: they become "Dynamic" States



GRAPHICS PIPELINE

Graphics Pipeline must be consistent with shaders

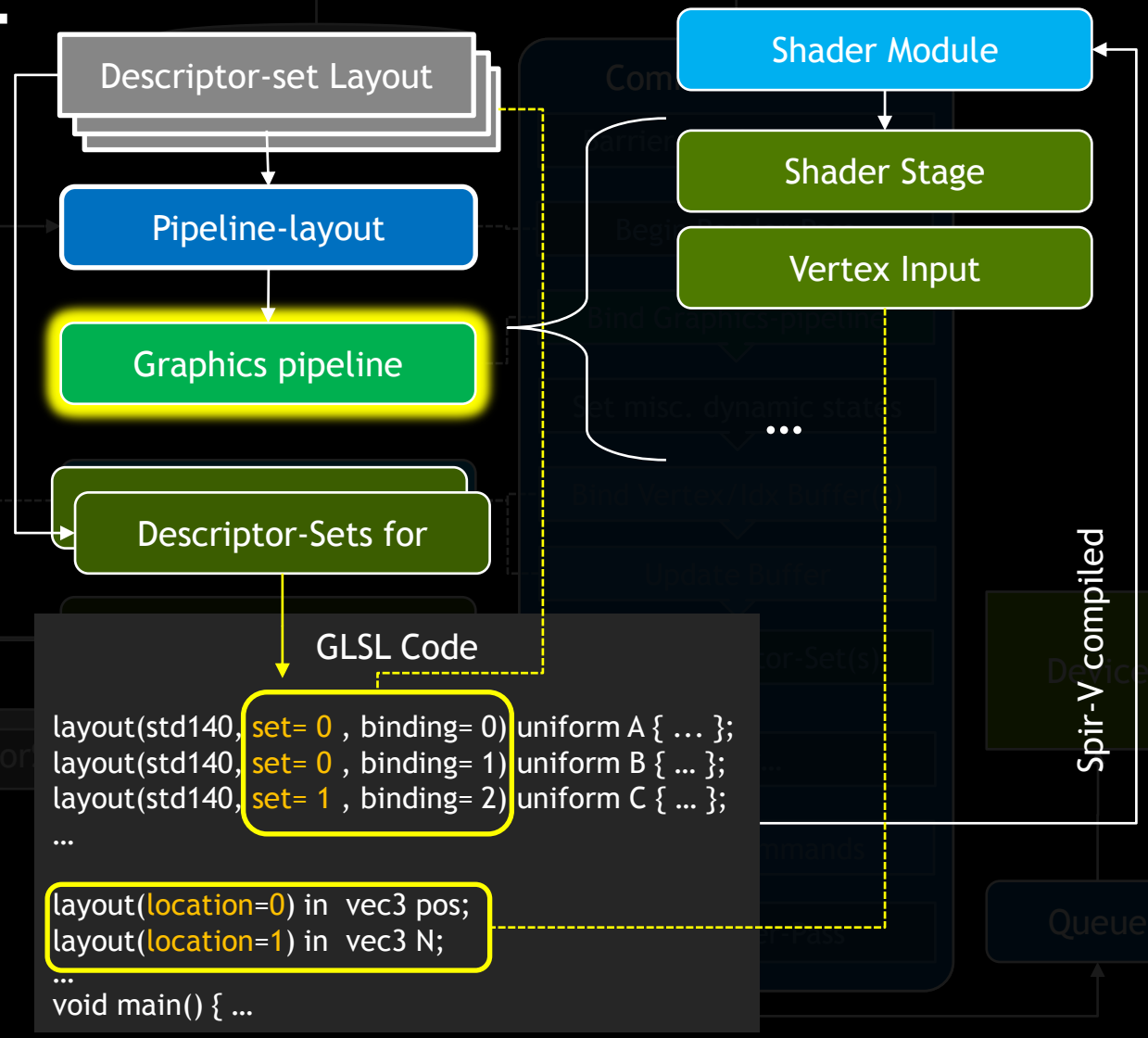
No "introspection", so everything known & prepared in advance

Vertex Input:

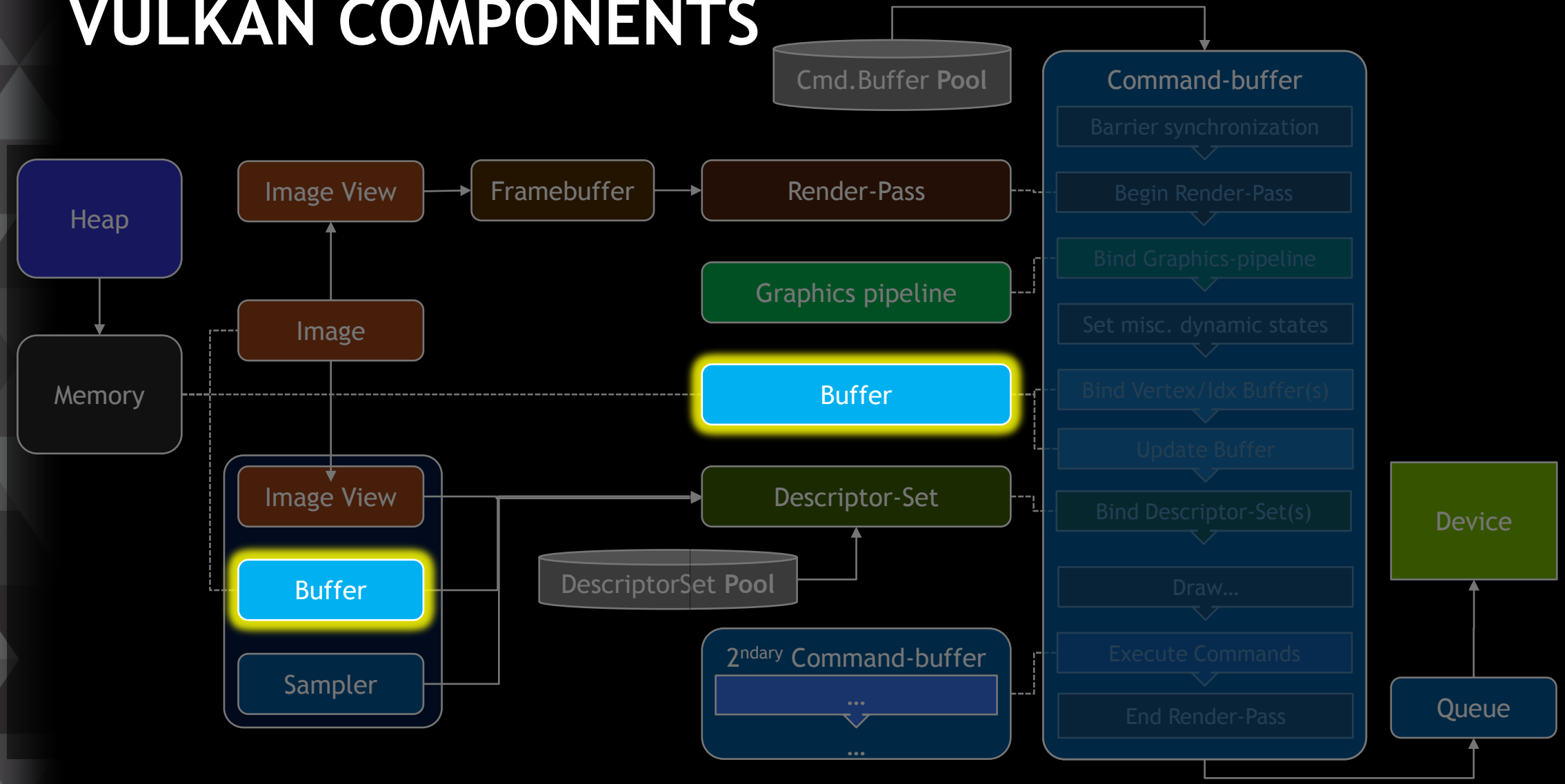
tells how Attributes: Locations are attached to which Vertex Buffer at which offset

Pipeline Layout:

Tells how to map Sets and Bindings for the shaders at each stage (Vtx, Fragment, Geom…)



VULKAN COMPONENTS



BUFFERS

Highly Heterogenous. Most often used for:

- Index/Vertex Buffers
- Uniform Buffers (Matrices, material parameters…)

Vulkan Object: Must be bound to some Device Memory

- Can be CPU accessible memory (mappable)
- Can be CPU cached
- Can be GPU accessible only: need a “Staging Buffer” to write into it
  - But most Efficient

(More on Device Memory later…)

COMMAND-BUFFERS: UPDATE/PUSH CONSTANTS

2 more ways to update constants/uniforms for Shaders from the Command-Buffer

Update-Buffer: prior to Render-Pass: can target any Buffer bound by Descriptor Sets

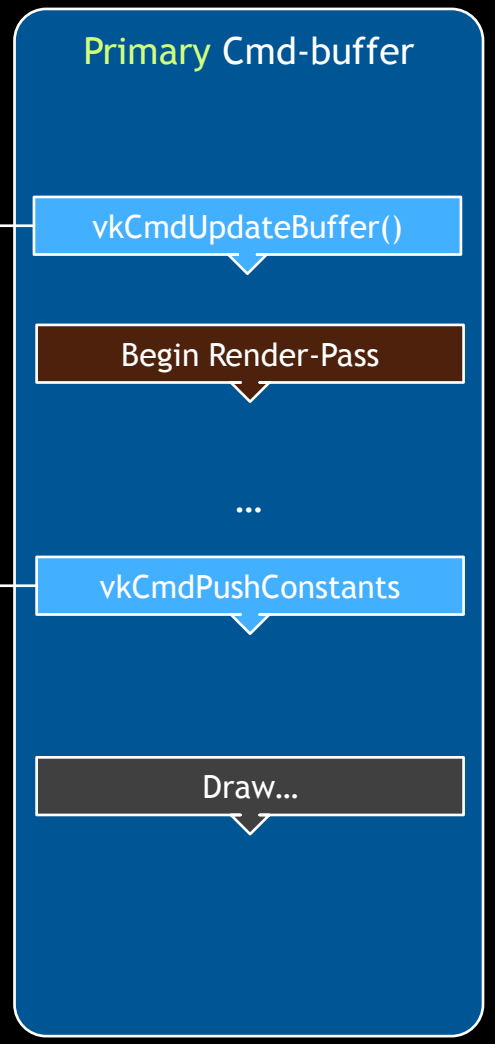
```
layout(set=0 , binding = 2 ) uniform MyBuffer {
   mat4 mW;
…
```

Push-Constants: targets a dedicated section in GLSL/SpirV

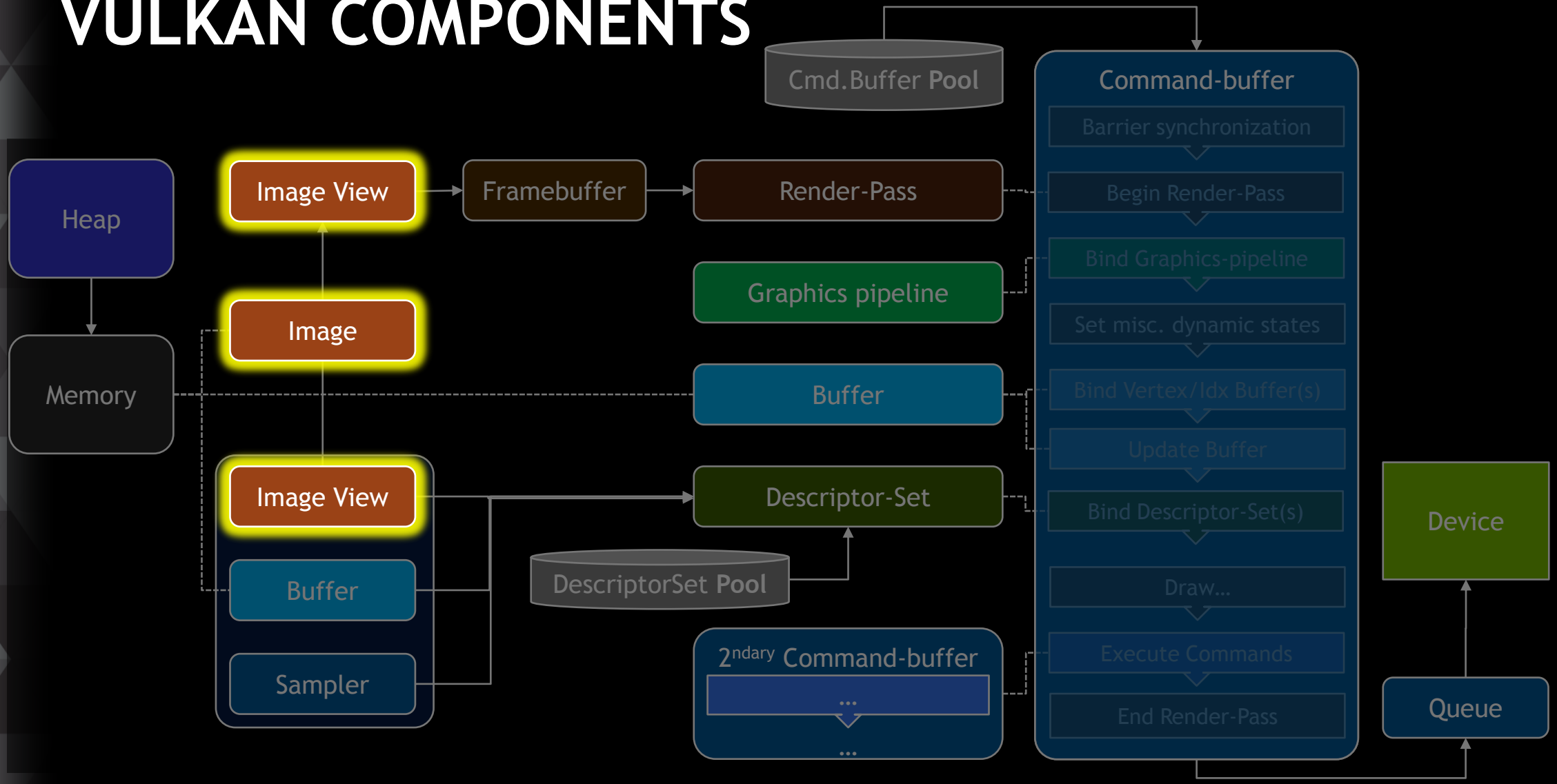
```
layout(push_constant) uniform objectBuffer {
   mat4 matrixObject;
   vec4 diffuse;
} object;
```

New values appended “in-band”: in the Command-Buffer

Efficient; but good for small amount of values



VULKAN COMPONENTS



IMAGES AND IMAGEVIEW

Images represent all kind of ‘pixel-like’ arrays

- Textures: Color or Depth-Stencil
- Render targets : Color and Depth-Stencil
- Even Compute data
- Shader Load/Store (imgLoadStore)

ImageView required to expose Images properly when specific format required

- For Shaders
- For Framebuffers

HOW DOES IT LOOK ?

Simple texture creation

Way more complex than OpenGL !

- Load image
- Create an Image (1D/2D/3D/Cube…)
- Create an Image-View
- Aggregate layers/mipmap layers info (offsets, sizes) in a structure (VkBufferImageCopy)
- Aggregate layers & mipmap data to contiguous memory

- Create staging buffer + bind memory + copy data in it
- Use command-buffer to copy to the image: layers and mipmaps
  - Layout transition of image for copy
  - vkCmdCopyBufferToImage
  - Layout transition of image for use by shader
- Enqueue command buffer and execute

HOW DOES IT LOOK ?

Simple texture creation

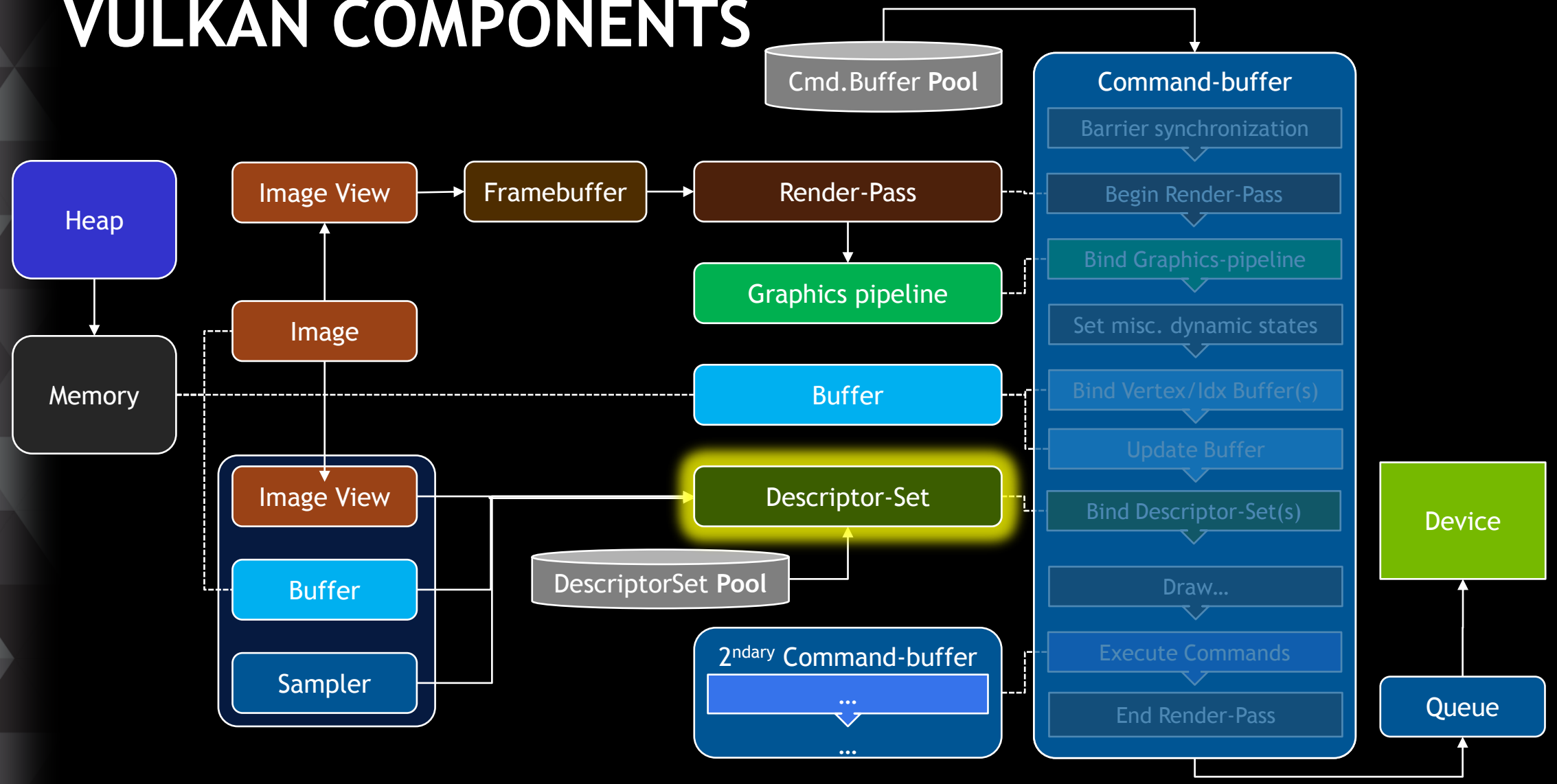
```
if(ddxImage.load(paths[i].c_str() ))
if(!ddxImage.is_valid())
{
    assert(!"error in loading texture");
}
switch(ddxImage.get_internal_format())
{
case RED:
    fmt = VK_FORMAT_R8_UNORM;
    break;
case RGB:
    fmt = VK_FORMAT_R8G8B8_UNORM;
    break;
case RGBA8:
    fmt = VK_FORMAT_R8G8B8A8_UNORM;
    break;
case COMPRESSED_RGBA_S3TC_DXT1_EXT:
    fmt = VK_FORMAT_BC1_RGBA_UNORM_BLOCK;
    break;
case COMPRESSED_RGBA_S3TC_DXT3_EXT:
    fmt = VK_FORMAT_BC2_UNORM_BLOCK;
    break;
case COMPRESSED_RGBA_S3TC_DXT5_EXT:
    fmt = VK_FORMAT_BC3_UNORM_BLOCK;
    break;
    assert(!"TODO");
}
noiseTex2D.img       = nvk.createImage2D(ddxImage.get_width(), ddxImage.get_
noiseTex2D.imgView   = nvk.vkCreateImageView(NVK::VkImageViewCreateInfo(
    noiseTex2D.img, // image
    VK_IMAGE_VIEW_TYPE_2D, //viewType
    fmt, //format
    NVK::VkComponentMapping(VK_COMPONENT_SWIZZLE_R, VK_COMPONENT_SWIZZLE_G,
    NVK::VkImageSubresourceRange()//subresourceRange
    ) );
aggregatedMipmaps = NULL;
aggregatedMipmaps_Sz = 0;
for(int i=0; i<ddxImage.get_num_mipmaps()+1; i++)
{
    const ddx::CSurface &imageSurface = ddxImage.get_mipmap(i);
    aggregatedMipmaps_Sz += imageSurface.get_size();
}
aggregatedMipmaps = new unsigned char[aggregatedMipmaps_Sz];
aggregatedMipmaps_Sz = 0;
NVK::VkBufferImageCopy imageCopy;
for(int i=0; i<ddxImage.get_num_mipmaps()+1; i++)
{
    const ddx::CSurface &imageSurface = ddxImage.get_mipmap(i);
    memcpy(aggregatedMipmaps + aggregatedMipmaps_Sz, (unsigned char*)imageSur
    imageCopy.add(
        aggregatedMipmaps_Sz, //bufferOffset
        0, //bufferRowLength
        imageSurface.get_height(), //bufferImageHeight,
        NVK::VkImageSubresourceLayers(VK_IMAGE_ASPECT_COLOR_BIT, i/*mipLevel*
        NVK::VkOffset3D(0,0,0),
        NVK::VkExtent3D(imageSurface.get_width(), imageSurface.get_height(),
    aggregatedMipmaps_Sz += imageSurface.get_size();
}
nvk.fillImage(m_perThreadData->m_cmdPoolStatic, imageCopy,
    aggregatedMipmaps, aggregatedMipmaps_Sz, noiseTex2D.img);
delete [] aggregatedMipmaps;
```

```
::VkImage NVK::createImage2D(
    int width, int height, int depth,
    ::VkDeviceMemory &colorMemory,
    ::VkFormat format,
    VkSampleCountFlagBits depthSamples,
    VkSampleCountFlagBits colorSamples,
    int mipLevels, bool asAttachment)
{
    ::VkImage                    colorImage;
    // color texture & view
    ::VkImageCreateInfo cbImage
    cbImageInfo.flags = 0;
    cbImageInfo.imageType = VK_
    cbImageInfo.format = format;
    cbImageInfo.extent.width =
    cbImageInfo.extent.height =
    cbImageInfo.extent.depth = d
    cbImageInfo.mipLevels = mip
    cbImageInfo.arrayLayers = 1;
```

```
void NVK::fillImage(::VkCommandPool cmdPool, VkBufferImageCopy &bufferImageCopy, const void* data, VkDeviceSize dataSz, ::VkImage image)
{
    if(!data)
        return;
    //
    // Create staging buffer
    //
    VkBuffer bufferStage;
    VkBufferCreateInfo bufferStageInfo(dataSz, VK_BUFFER_USAGE_TRANSFER_SRC_BIT);
```

```
::VkDeviceMemory NVK::allocMemAndBindObject(::VkImage obj, ::VkFlags memProps)
{
    ::VkResult result;
    ::VkDeviceMemory deviceMem;
    ::VkMemoryRequirements  memReqs;
    vkGetImageMemoryRequirements(m_device, obj, &memReqs);

    if (!memReqs.size){
      return NULL;
    }

    //
    // Find an available memory type that satisfies the requested properties.
    //
    uint32_t memoryTypeIndex;
    for (memoryTypeIndex = 0; memoryTypeIndex < m_gpu.memoryProperties.memoryTypeCount; ++memoryTypeIndex) {
      if (( memReqs.memoryTypeBits & (1<<memoryTypeIndex)) &&
        ( m_gpu.memoryProperties.memoryTypes[memoryTypeIndex].propertyFlags & memProps) == memProps)
      {
        break;
      }
    }
    if (memoryTypeIndex >= m_gpu.memoryProperties.memoryTypeCount) {
      assert(0 && "memoryTypeIndex not found");
      return NULL;
    }

    ::VkMemoryAllocateInfo memInfo = {
      VK_STRUCTURE_TYPE_MEMORY_ALLOCATE_INFO,            // sType
      0,                                                 // pNext
      memReqs.size,                                      // allocationSize
      memoryTypeIndex,                                   // memoryTypeIndex
    };

    //LOGW("allocating memory: memReqs.memoryTypeBits=%d memProps=%d memoryTypeIndex=%d Size=%d\n", memReqs.memoryTypeBits, memProps, memoryTypeIndex, memReqs.size);

    result = vkAllocateMemory(m_device, &memInfo, NULL, &deviceMem);
    if (result != VK_SUCCESS) {
      return NULL;
    }

    result = vkBindImageMemory(m_device, obj, deviceMem, 0);
    if (result != VK_SUCCESS) {
      return NULL;
    }

    return deviceMem;
}
```

VULKAN COMPONENTS



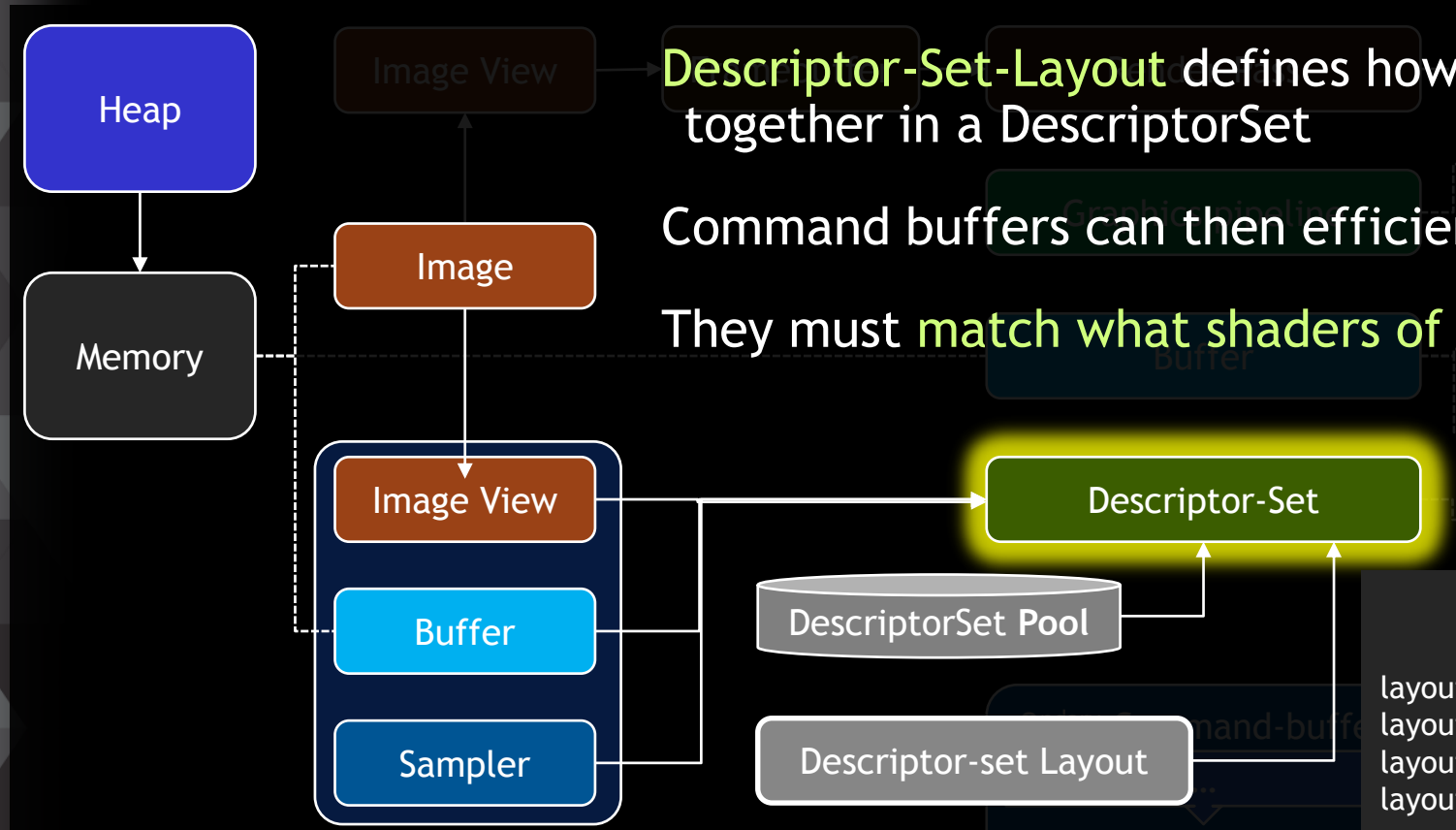
DESCRIPTOR-SET

Each DescriptorSet holds references to some resources

Descriptor-Set-Layout defines how resources must be put together in a DescriptorSet

Command buffers can then efficiently bind any or them

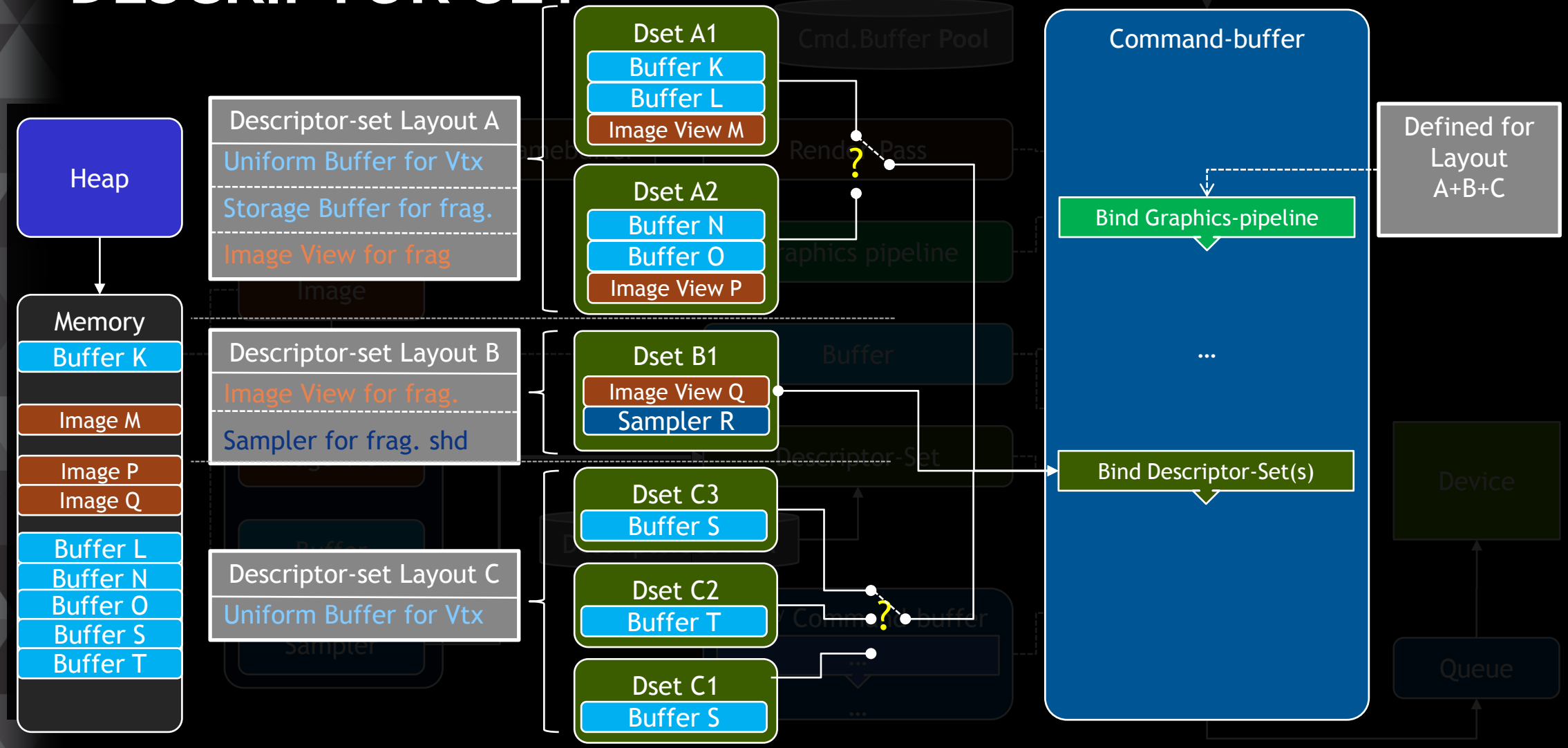
They must match what shaders of each stage expect !



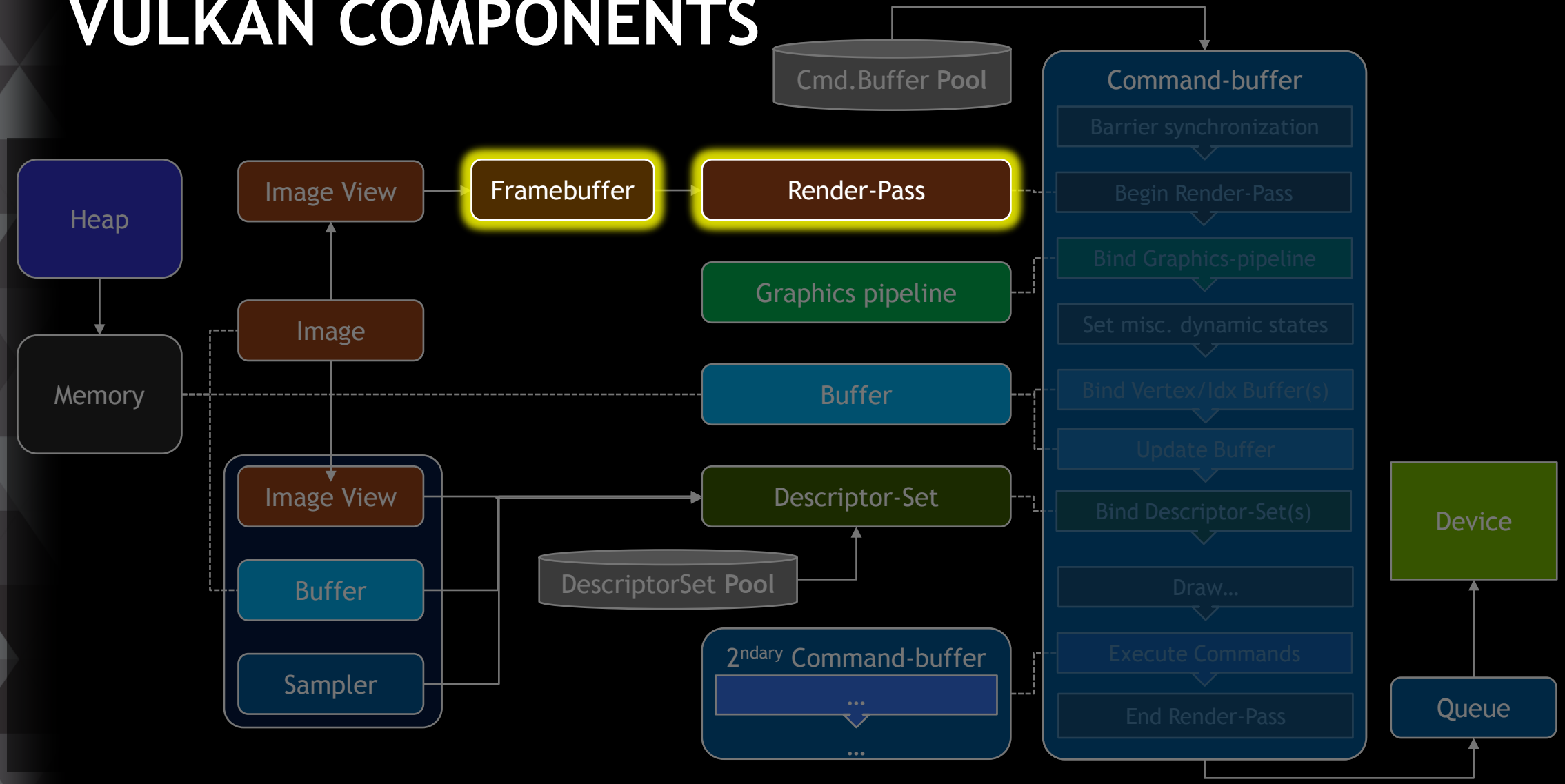
GLSL Code

```
layout(std140, set= 0 , binding= 0) uniform A { … };
layout(std140, set= 0 , binding= 1) uniform B { … };
layout(std140, set= 1 , binding= 2) uniform C { … };
layout(set=0, binding=3) uniform sampler2D tex;
…
void main() { …
```

DESCRIPTOR-SET

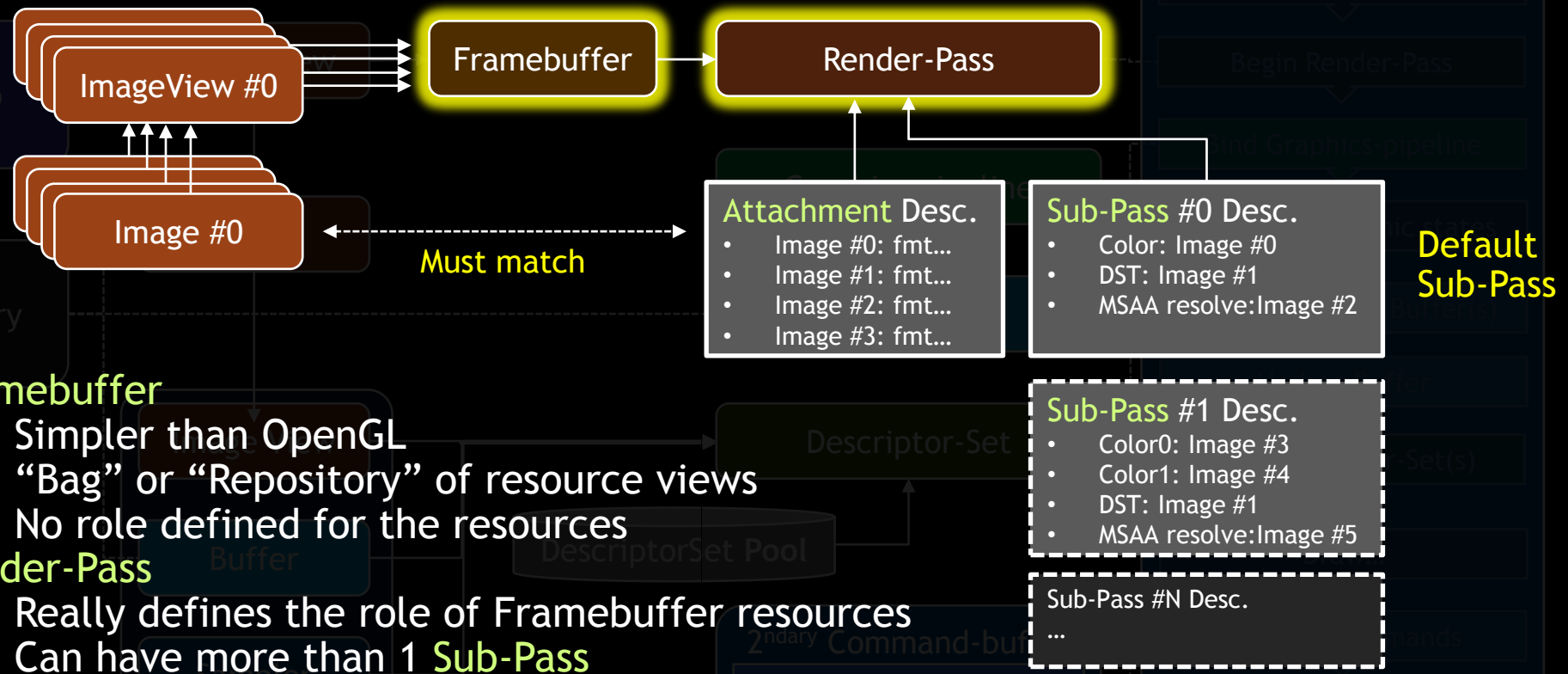


VULKAN COMPONENTS



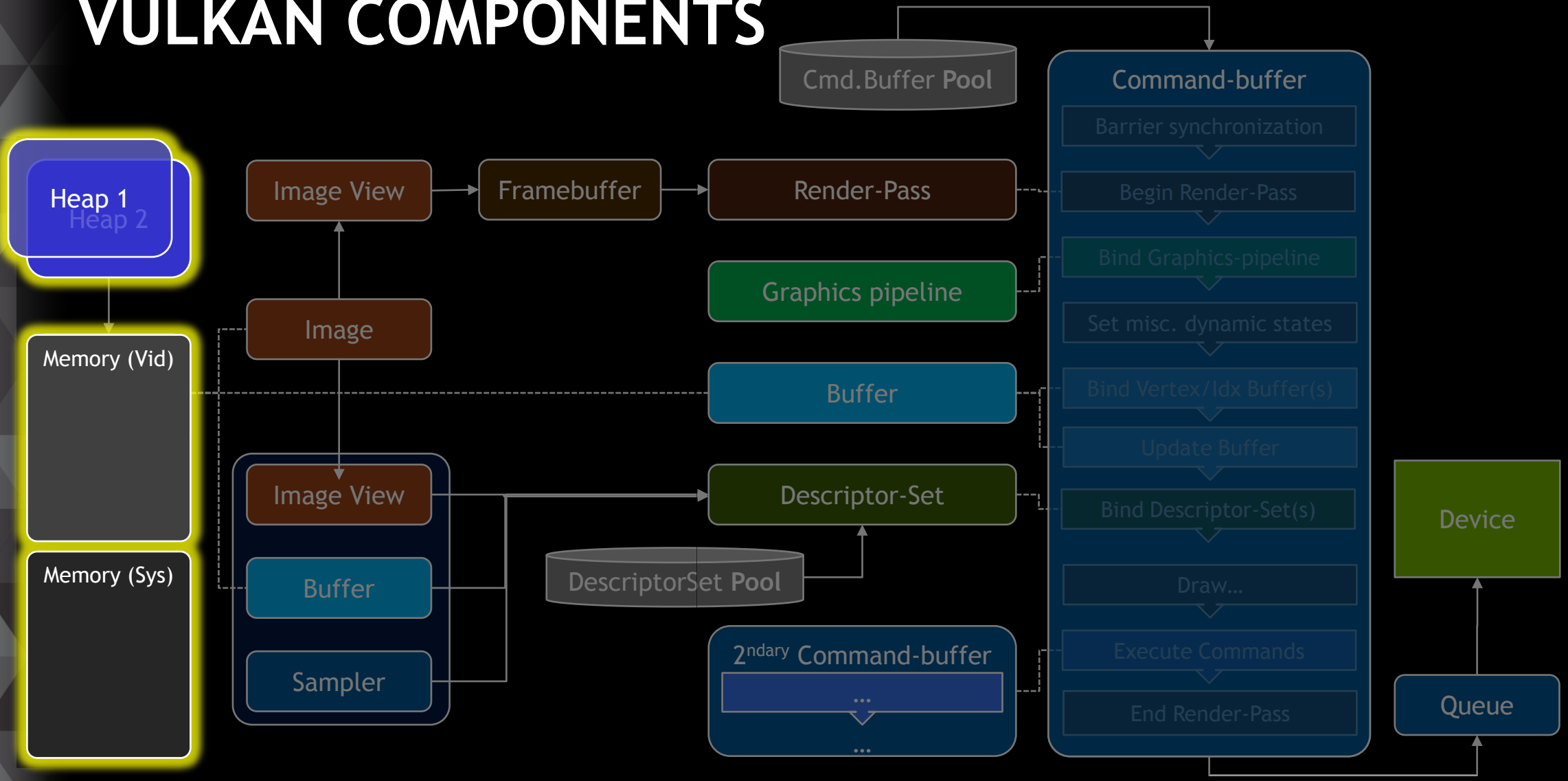
VULKAN COMPONENTS

Can use many if compatibles



- Framebuffer
  - Simpler than OpenGL
  - “Bag” or “Repository” of resource views
  - No role defined for the resources
- Render-Pass
  - Really defines the role of Framebuffer resources
  - Can have more than 1 Sub-Pass
  - Each Sub-Passes defines which Framebuffer resource to use
  - invented for Tilers Arch

VULKAN COMPONENTS



MEMORY ⇔VULKAN OBJECTS

Vulkan Objects *referring to buffer(s) of data* need binding to memory

Vertex/Index Buffers; Uniform Buffers; Images/Textures…

Vulkan Device exposes various Memory Heaps - Example:

heap 0: size:12,288Mb (Video Memory of my K6000)

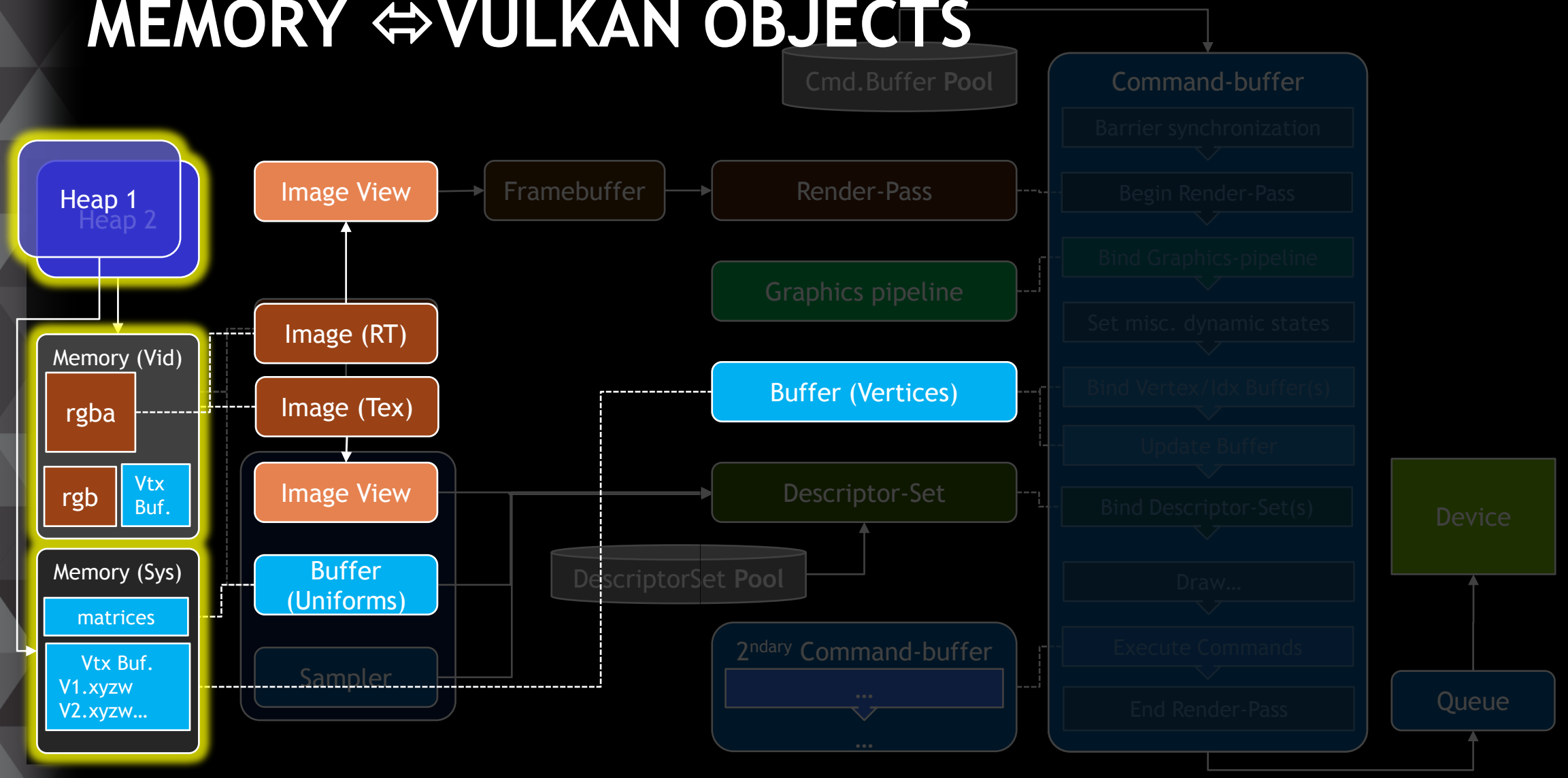
heap 1: size:17,911Mb (System Memory of my PC)

And various Memory Types from these Heaps. Example:

Mem.Type	Heap	Flags
0	1 (sys.mem)	-
1	0 (Video)	DEVICE_LOCAL
2	1 (sys.mem)	HOST_VISIBLE | HOST_COHERENT
3	1 (sys.mem)	HOST_VISIBLE | HOST_COHERENT | HOST_CACHED

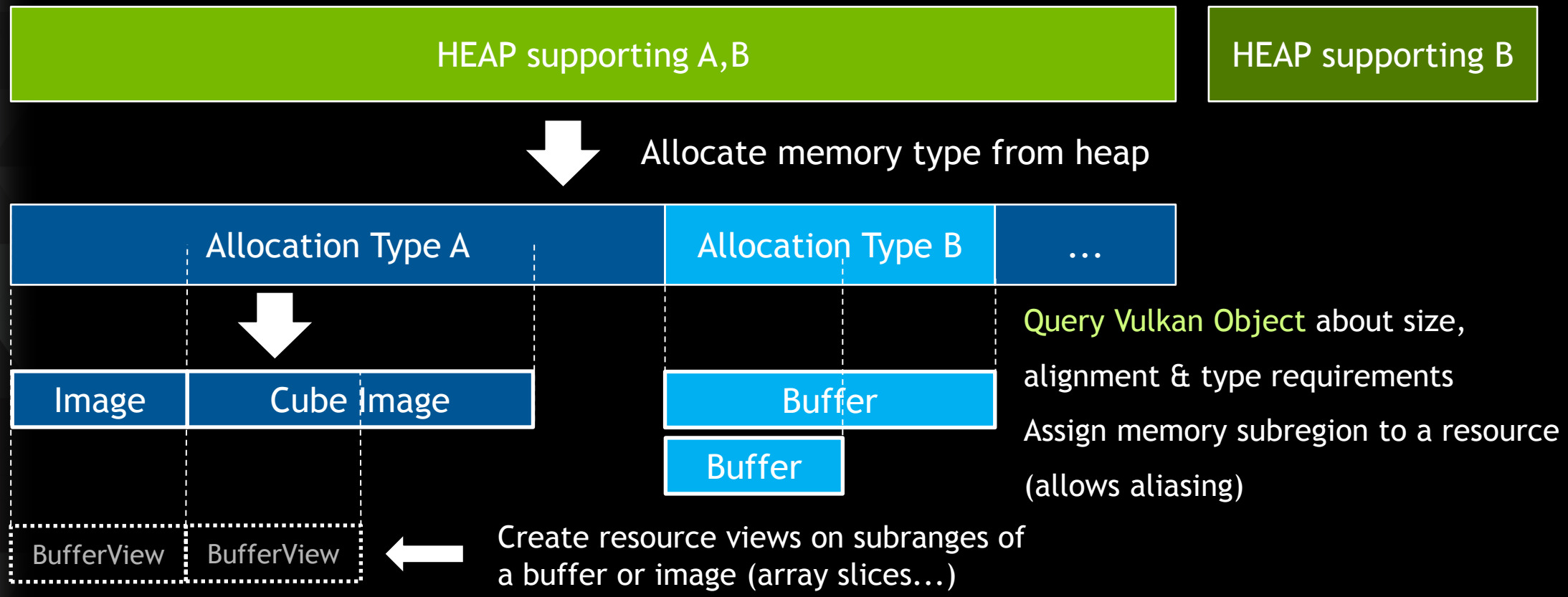
Tegra: Adds one more: HOST_VISIBLE “NON-Coherent”

MEMORY ⇔VULKAN OBJECTS

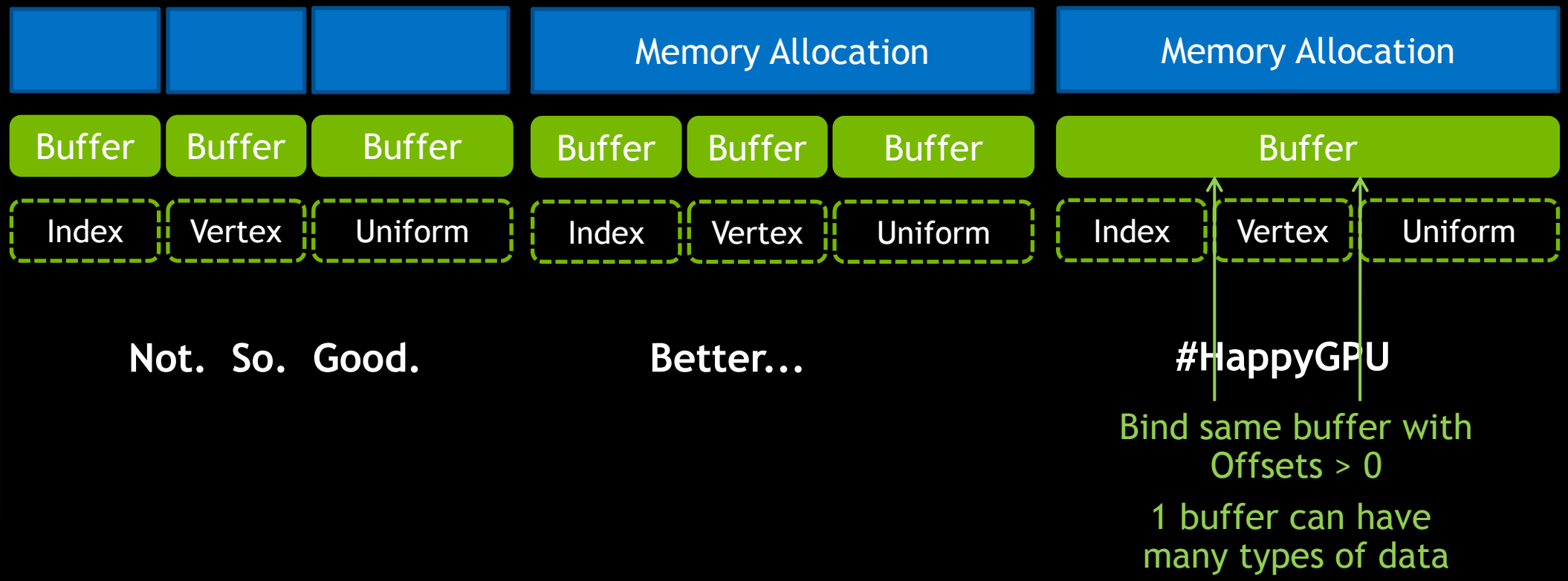


RESOURCE MANAGEMENT

Allocation and Sub allocation



RESOURCE MANAGEMENT

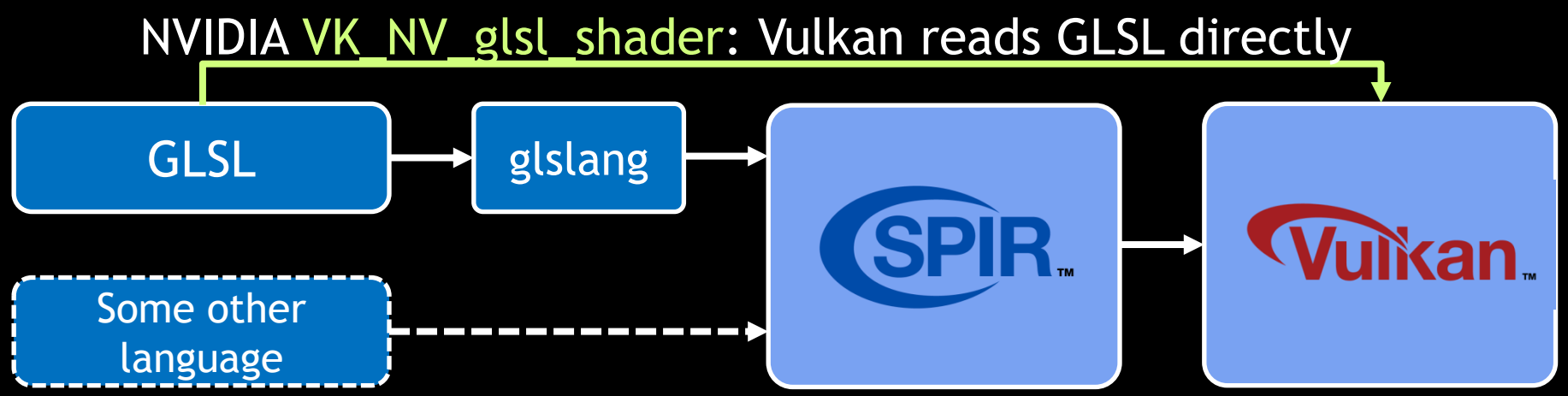


SHADERS

Vulkan uses SPIR-V passed directly to the driver

- Can be compiled from GLSL Via glslang or LunarG's glslangValidator; Google ShaderC
- theoretically other languages could be compiled to Spir-V…
- Libraries available to compile GLSL to Spir-V from the application

NVIDIA allows to compile GLSL directly



SHADERS

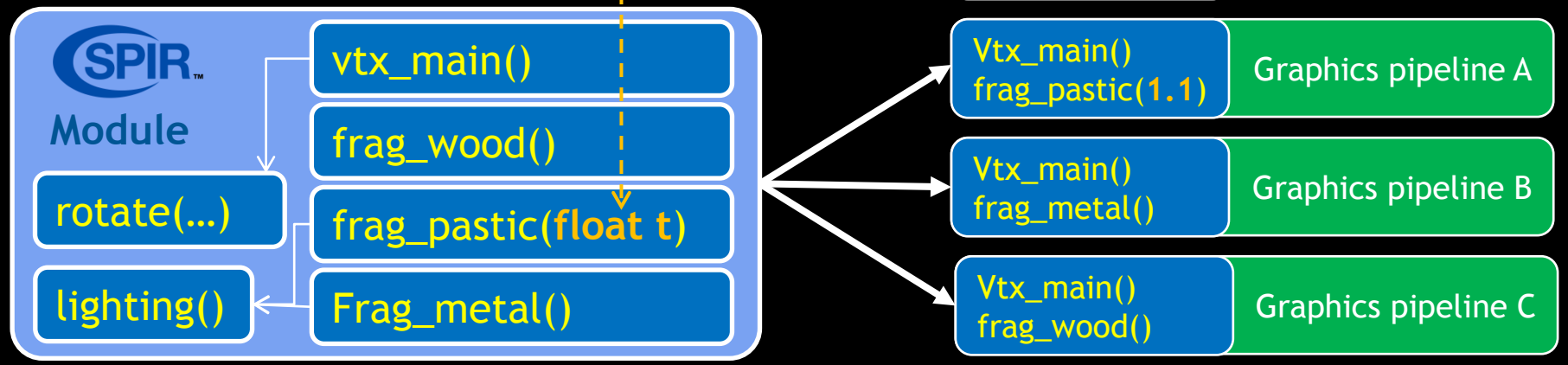
Multiple entry points can be defined in a single Spir-V shader-module

Prevents redundant code: shader module used by many Graphics-Pipelines

Specialization Constants: early setup of constants for shaders in given Graphics-Pipeline

Allows sharing snippets of code : easier to share common shader code

Warning: Current GLSL → Spir-V compilers
Don't support this feature, yet
But part of the API & Spir-V
Will happen soon

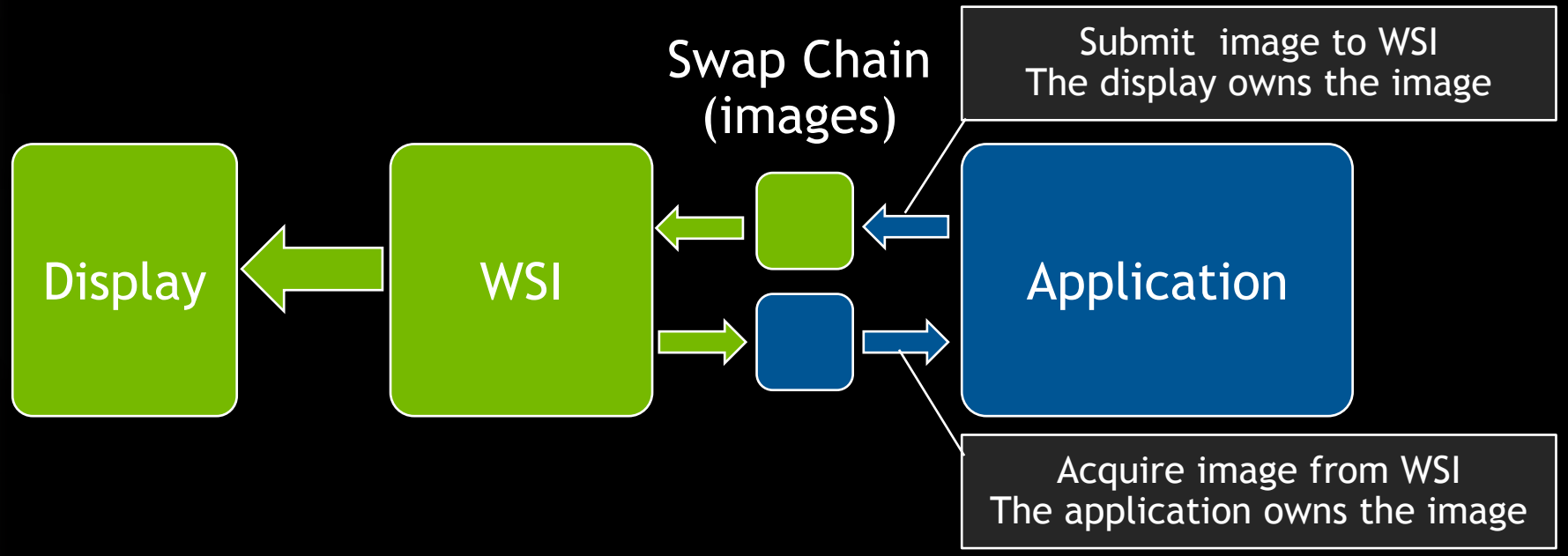


VULKAN WINDOW SYSTEM INTEGRATION (WSI)

WSI manages the ownership of images via a swap chain

One image is presented while the other is rendered to

WSI is a Vulkan Extension



NVIDIA OPENGL ⇔VULKAN INTEROP

Alternative to WSI: GL_NV_draw_vulkan_image

Create an OpenGL Context and all the usual things

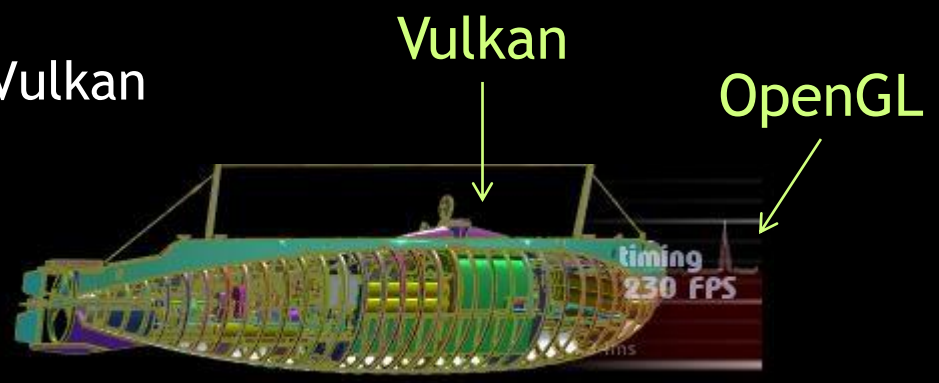
Create Vulkan Device

Rendering Loop involves both OpenGL and Vulkan

- Blit the Vulkan image to OpenGL backbuffer: glDrawVkImageNV
- Extra care on synchronization (Semaphores)

Bonus: Mix OpenGL rendering (UI overlay…) with Vulkan

- Allows smooth transition in projects



PRE-REQUISITES TO WORK WITH VULKAN

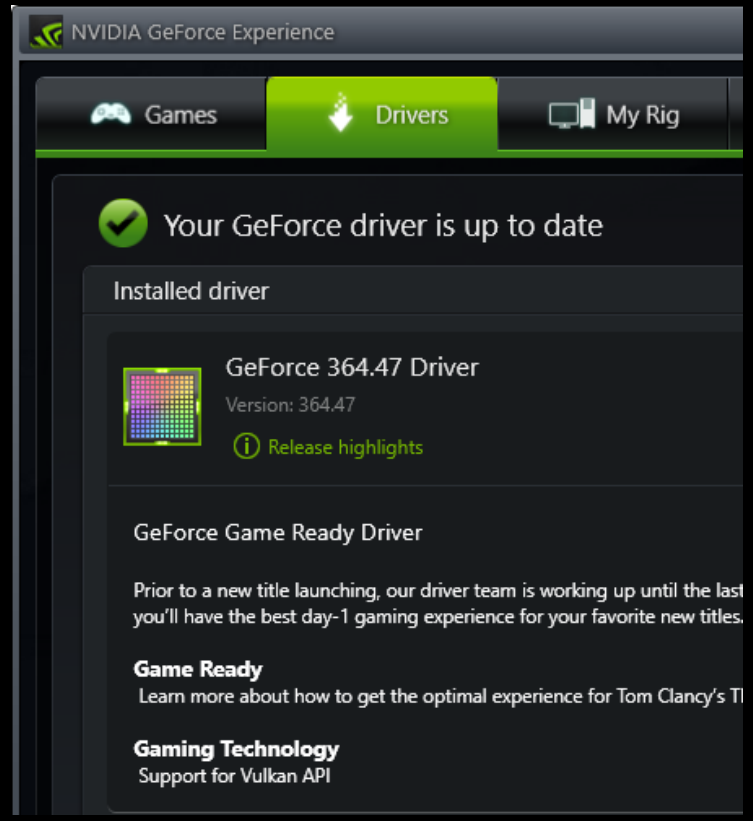
Lunar-G (http://lunarg.com/)

- Vulkan Loader (+Source code)
- Tools: Spir-V compiler for GLSL code and other libraries
- Layers: intermediate code invoked by Vulkan API functions to help debug
- Vulkan Includes

Drivers:

- GeForce Experience
- https://developer.nvidia.com/vulkan-driver

NVIDIA resources: https://developer.nvidia.com/Vulkan



RECAP' ON NVIDIA-SPECIFIC FEATURES

Compatible GPUs for Vulkan: Kepler and Higher; Shield Tablet; Shield Android TV

VK_NV_glsl_shader : GLSL can be directly sent to Vulkan

VK_NV_dedicated_allocation : more efficient memory usage

GL_NV_draw_vulkan_image can replace WSI

16 Queues. All available for all kind of use; 1 Queue for Copy-Engine only

3 frames (max) in flight with WSI

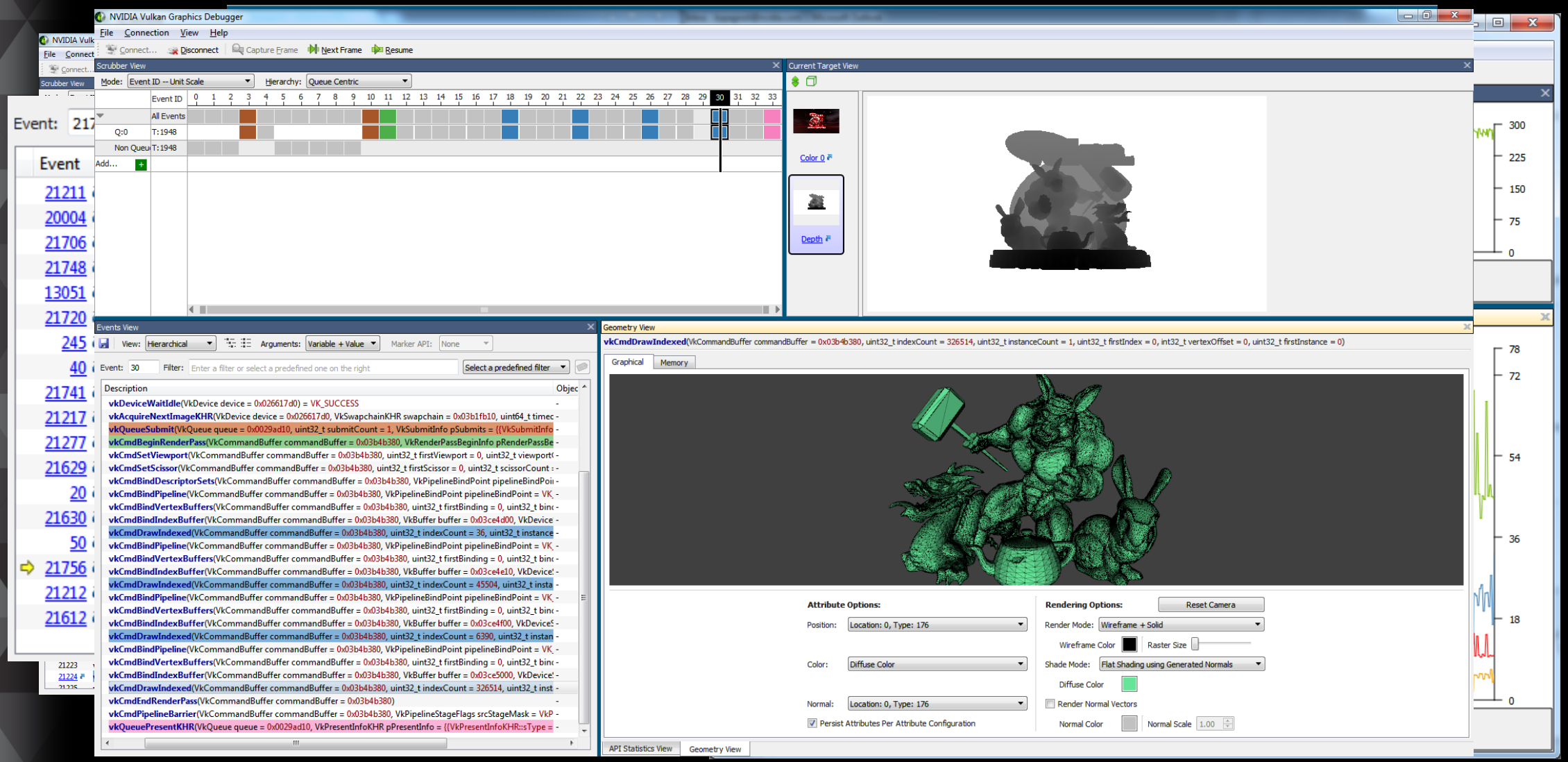
All Host memories are "Coherent" (except one for Tegra)

Layout transitions don't exist in our HW (VK_IMAGE_LAYOUT_GENERAL)

Linear-Tiling only for 2D non-mipmapped textures… please avoid (bad performance)

Shaders never need re-compilation due to states in Graphics-pipeline

NSIGHT FOR VULKAN



RECAP' ON VULKAN PHILOSOPHY

Validate as much as possible up-front (DescriptorSets; Pipelines…)

The driver doesn't waste time on figuring-out how to set things-up

Reuse existing patterns of Graphics-Pipelines: cached pipelines

Know your application: Taylor Vulkan design according to it

Know your memory usage: You are in charge of optimal sub-allocations

Explicit multi-threading for graphics: Application's responsibility

Explicit Resource updates: Either through [non]Coherent buffers; or Queue-Based DMA transfers

FEW WORDS ON VKCPP PROJECT

C++11 to the rescue

- Open-Source Project of a C++11 overlay for Vulkan: became Khronos-offcial (!)
- Simplify Vulkan usage by
  - reducing risk of errors, i.e. type safety, automatic initialization of sType, …
  - Reduce #lines of written code, i.e. constructors, initalizer lists for arrays, …
  - Add utility functions for common tasks (suballocators, resource tracking, …)

http://on-demand.gputechconf.com/gtc/2016/events/vulkanday/Vulkan_C++.pdf

https://developer.nvidia.com/open-source-vulkan-c-api

https://www.khronos.org/news/permalink/khronos-introduces-vulkan-hpp-open-source-vulkan-c-api

VKCPP PROJECT

Two C++ based layers

Autogenerated ‚low-level‘ layer using vulkan.xml

- Type safety
- Syntactic sugar
- Lightweight layer; Keeps you closer to the real Vulkan

Hand-coded ‚high level‘ layer

- Reduce code complexity
- Exception safety, resource lifetime tracking, …
- Closer dependency with VkCpp internal implementations

NATIVE VULKAN VS. VKCPP CODE

Native Vulkan: ~750 lines

vkCPP: ~200 lines




REFERENCES

Vulkan info from NVIDIA:

https://developer.nvidia.com/Vulkan

https://developer.nvidia.com/vulkan-graphics-api-here

Samples + Source code in OpenGL and Vulkan:

https://github.com/nvpro-samples

Other:

https://gameworks.nvidia.com

https://developer.nvidia.com/designworks

http://vulkan.gpuinfo.org/listreports.php

https://developer.nvidia.com/open-source-vulkan-c-api

THANK YOU

HTTPS://DEVELOPER.NVIDIA.COM/DESIGNWORKS

TLORACH@NVIDIA.COM