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

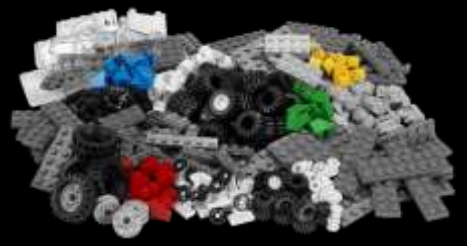
Analogy

Different Valid Approaches

(booring…)

Fixed-function OpenGL

(cool... Messes-up the bedroom)



Modern AZDO OpenGL with Programmable Shaders

(adult supervision required!)

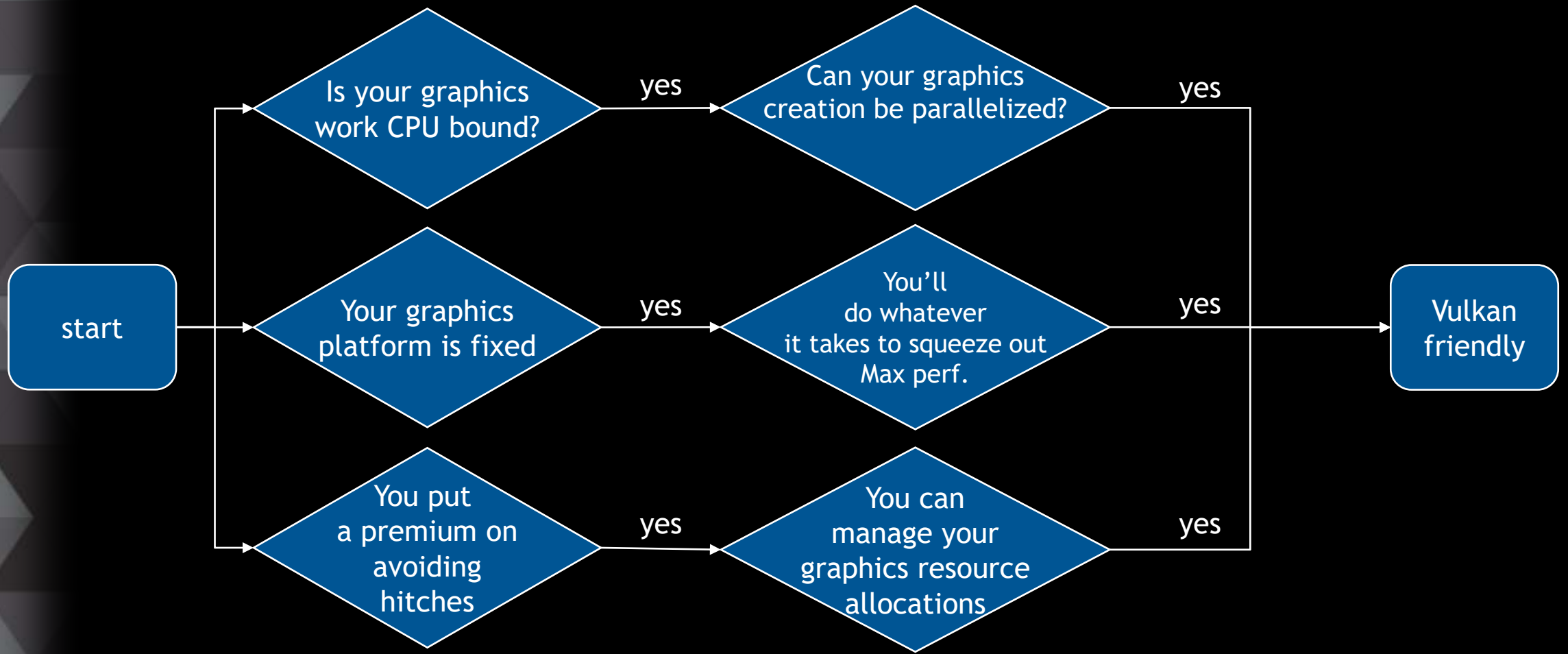
Vulkan

WHAT IS VULKAN ?

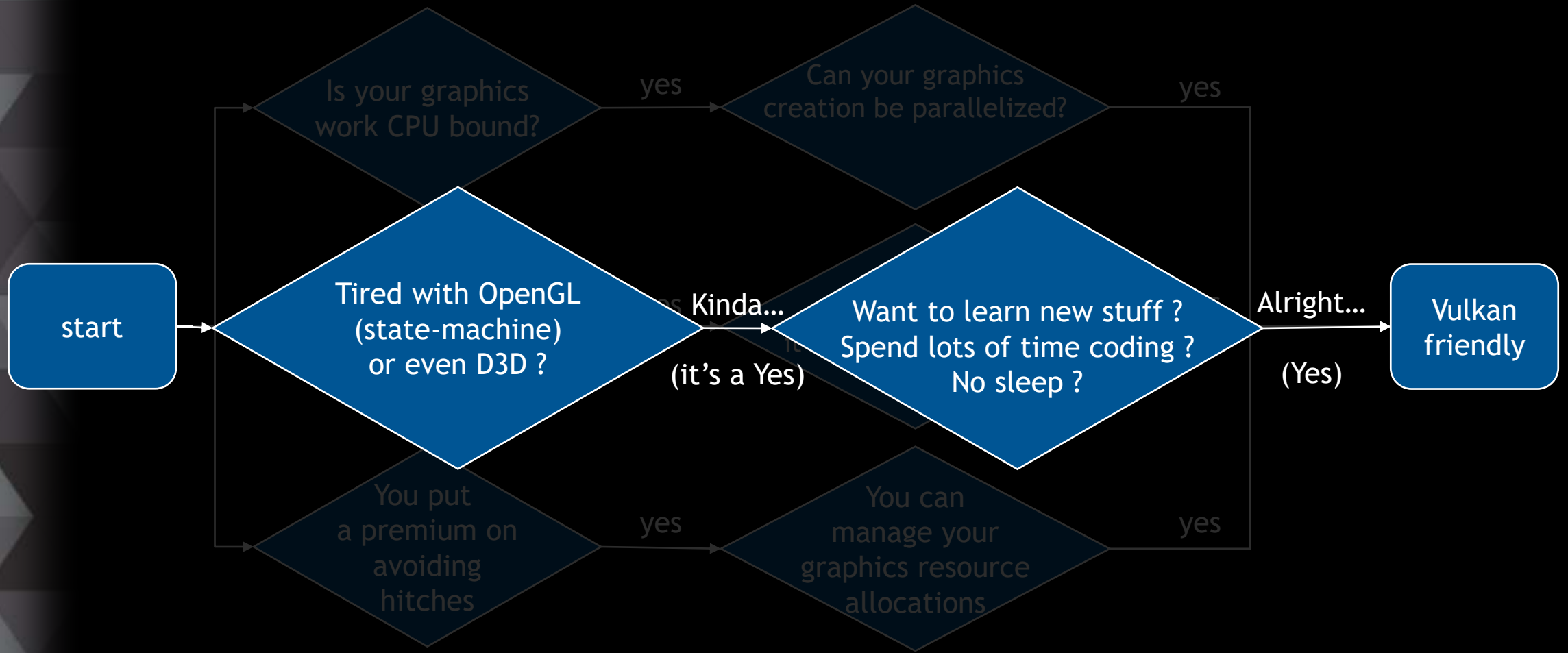
…It's a modern API

- 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



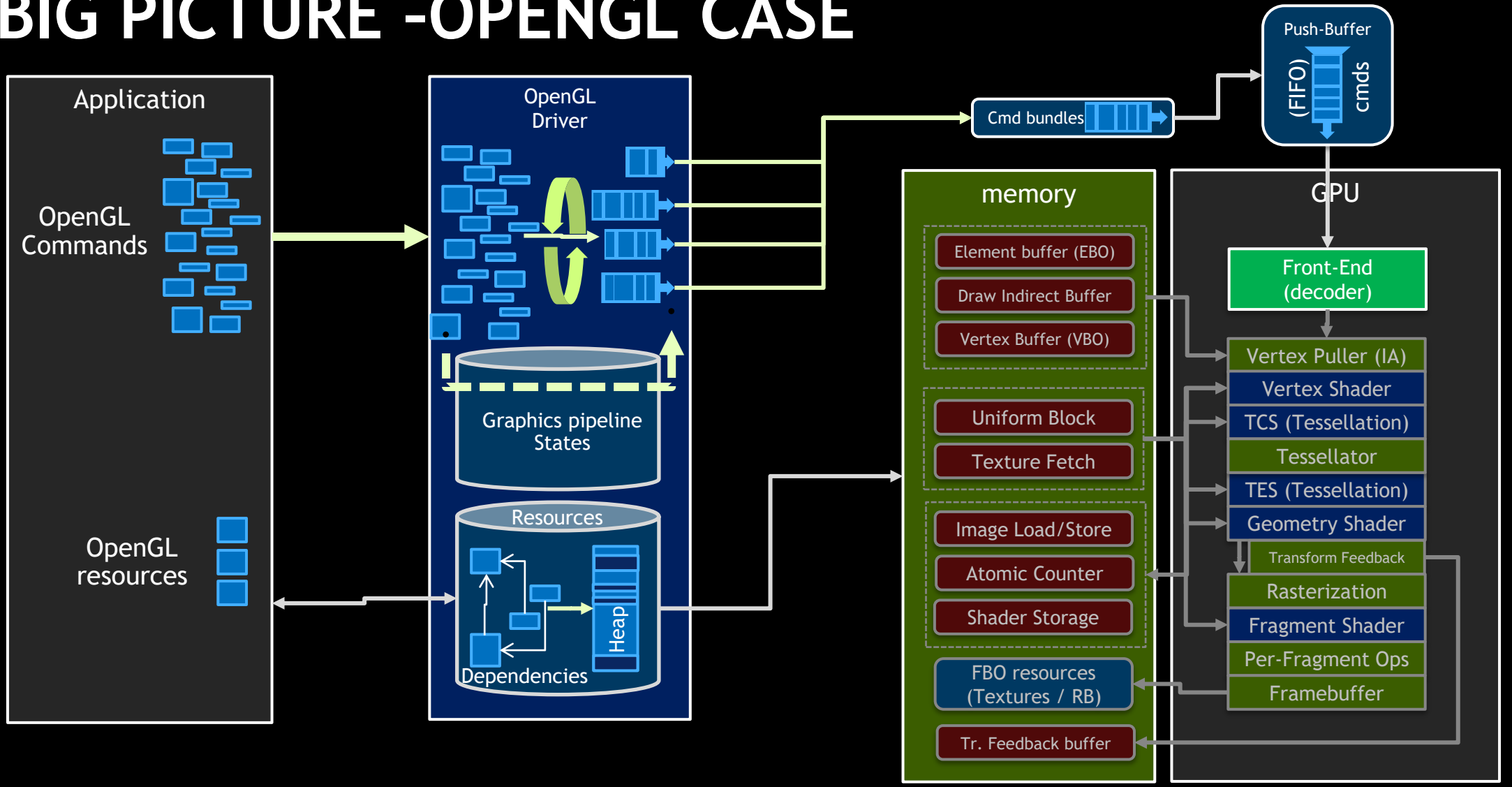
Unlikely to Benefit

Scenarios to Reconsider Coding to Vulkan

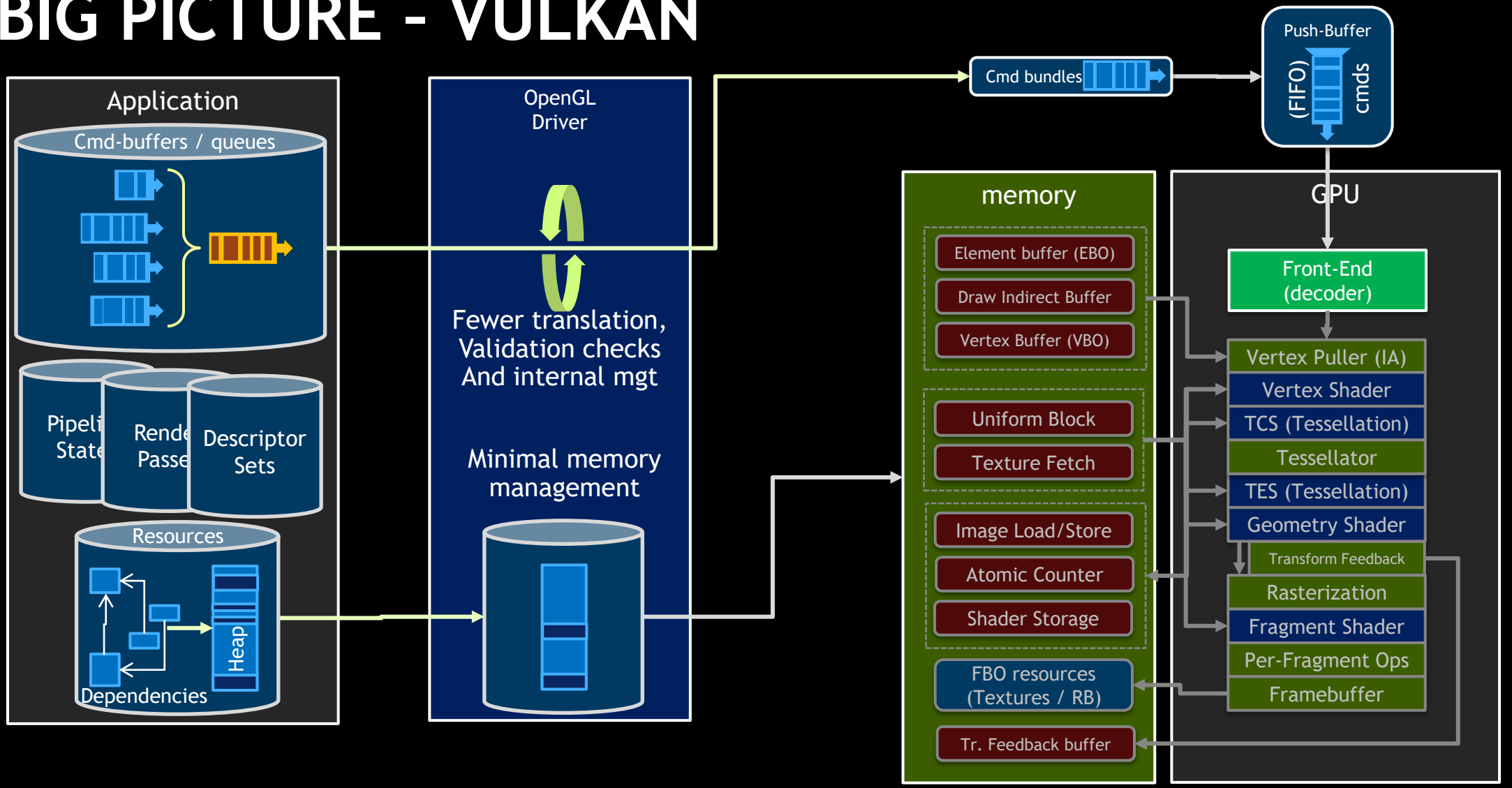
1. Need for compatibility to pre-Vulkan platforms
2. Heavily GPU-bound application
3. Heavily CPU-bound application due to non-graphics work
4. Single-threaded application, unlikely to change
5. App can target middle-ware engine, avoiding 3D graphics API dependencies
    - Consider using an engine targeting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

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

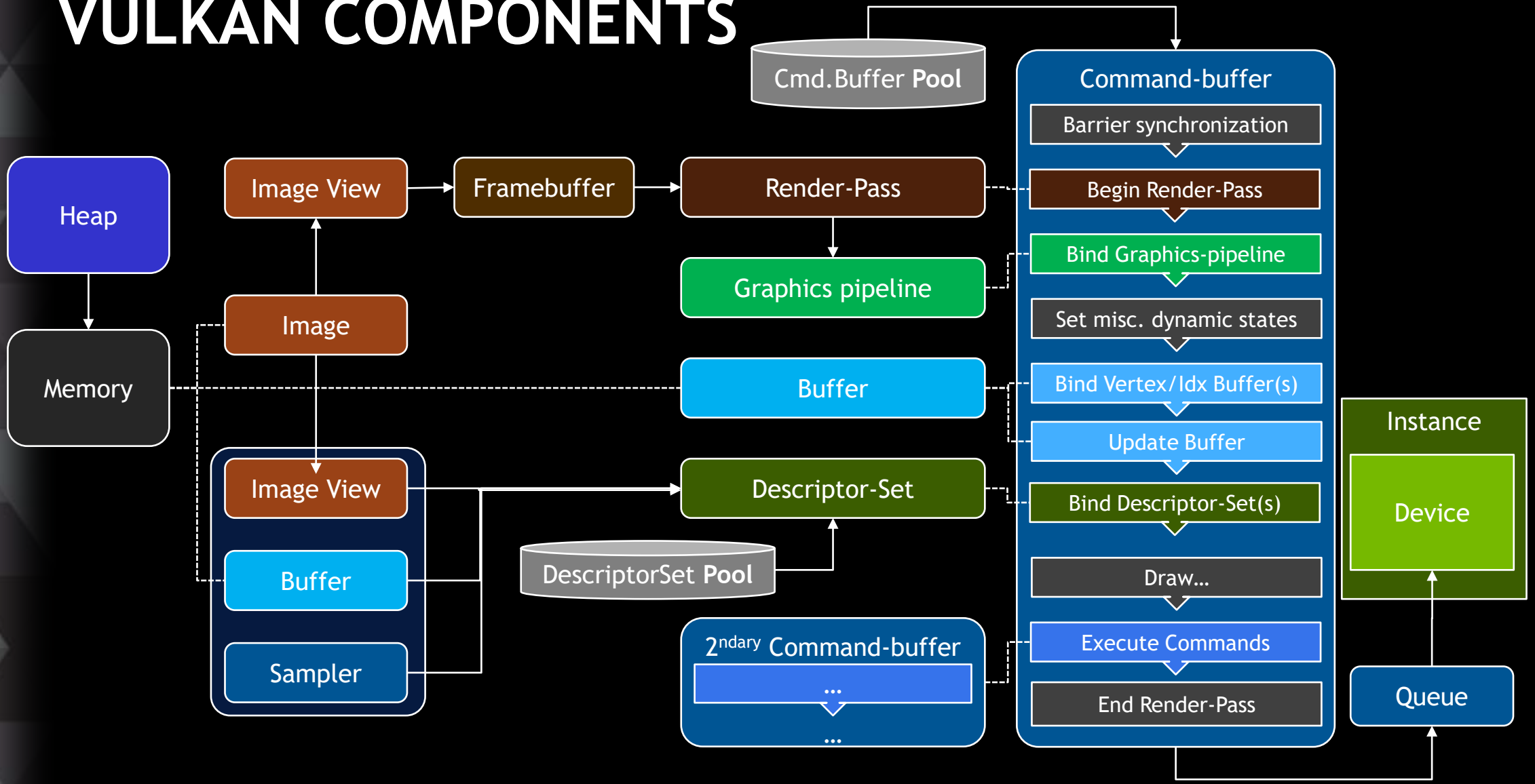
BIG PICTURE -OPENGL CASE



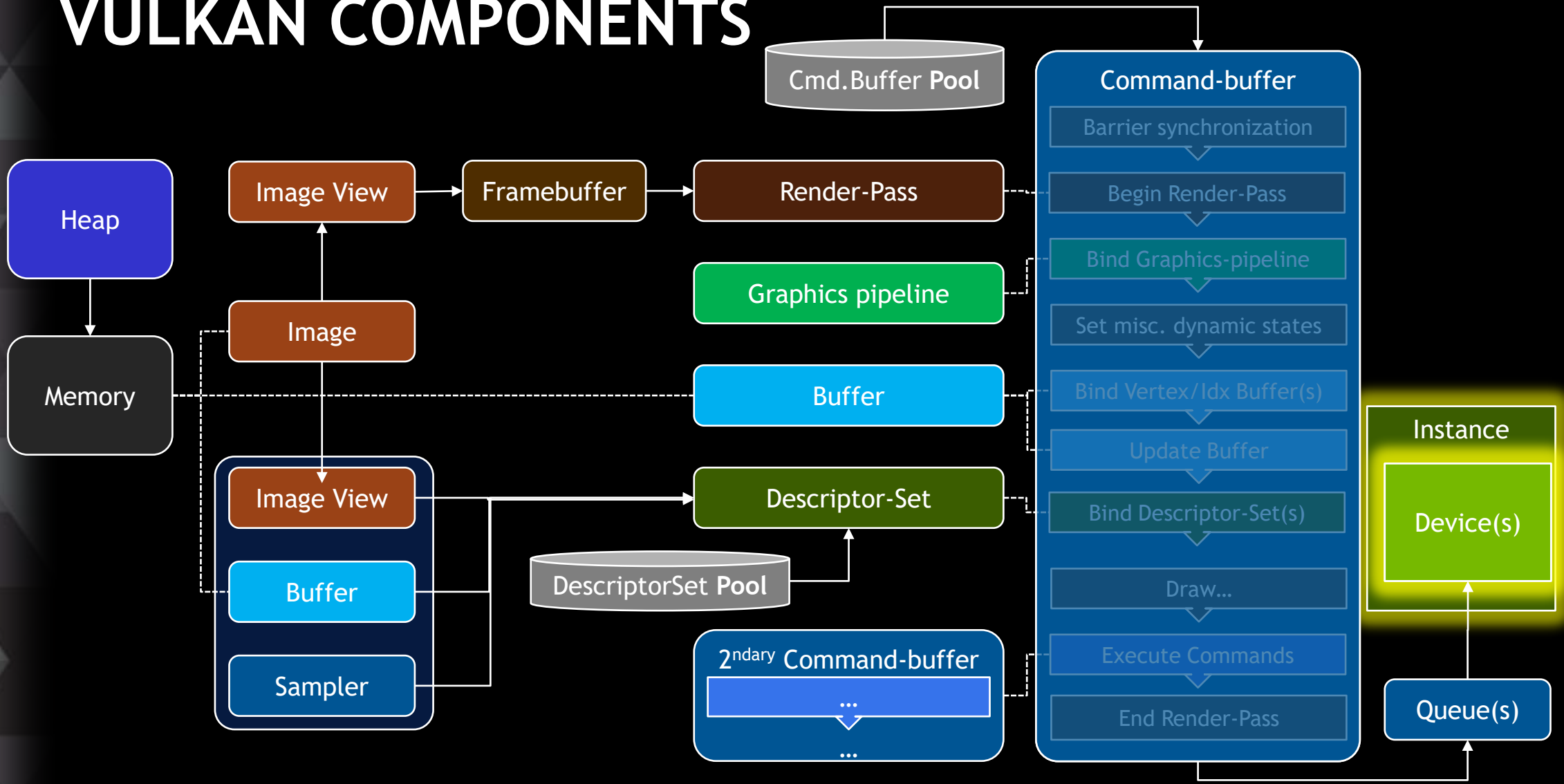
BIG PICTURE - VULKAN



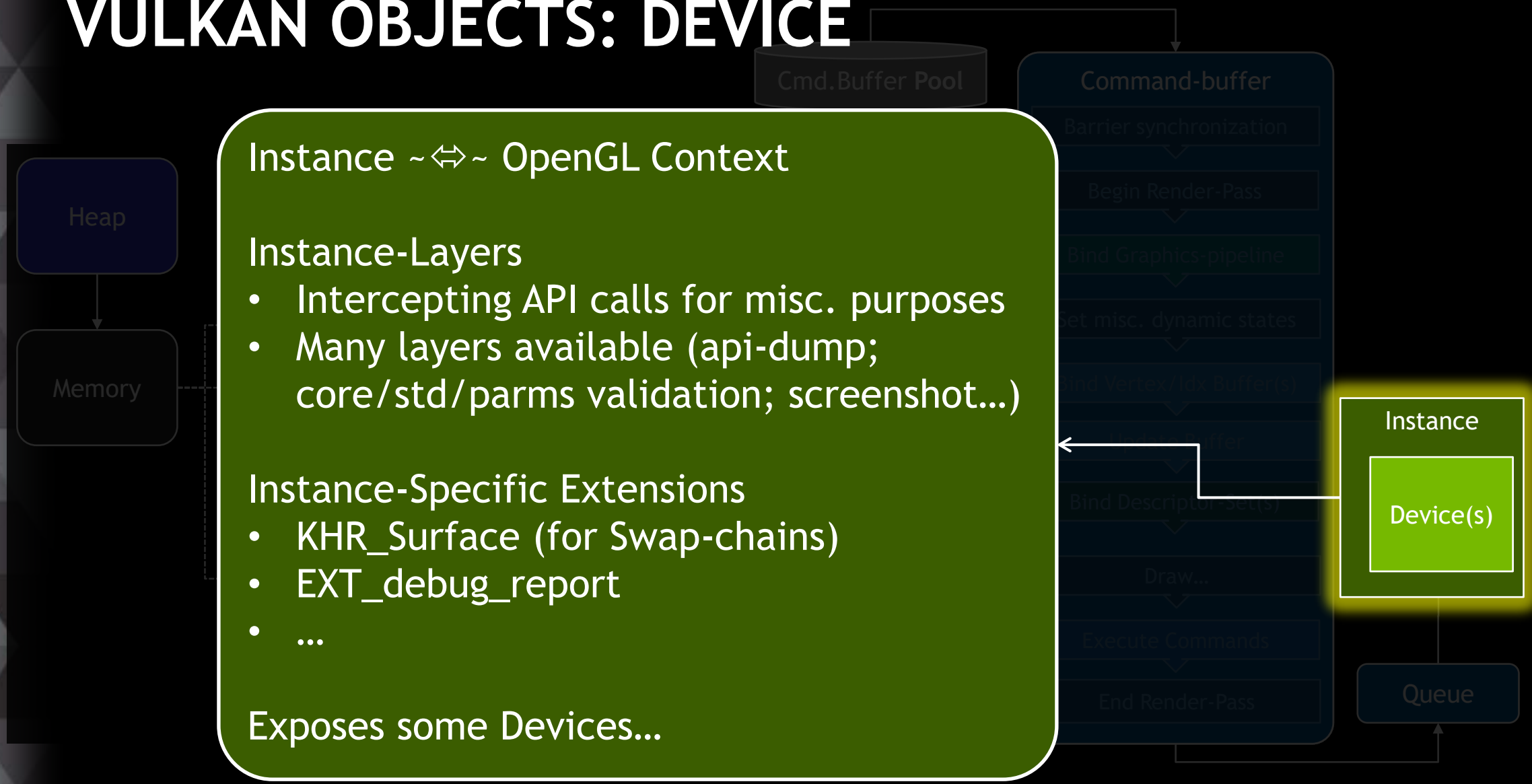
VULKAN COMPONENTS



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…

HOW DOES IT LOOK ?

Instance creation

```
result = vkEnumerateInstanceLayerProperties(&count, NULL);
assert(result == VK_SUCCESS);
if(count > 0)
{
    instance_layers.resize(count);
    result = vkEnumerateInstanceLayerProperties(&count, &instance_layers[0]);
    //...
```

Layer properties

```
result = vkEnumerateInstanceExtensionProperties(NULL, &count, NULL);
assert(result == VK_SUCCESS);
instance_extensions.resize(count);
extension_names.resize(count);
result = vkEnumerateInstanceExtensionProperties(NULL, &count, &instance_extensions[0]);
```

Extension properties

HOW DOES IT LOOK ?

Instance creation

```
appInfo.pApplicationName = "...";
appInfo.applicationVersion = 1;
appInfo.pEngineName = "...";
appInfo.engineVersion = 1;
appInfo.apiVersion = VK_MAKE_VERSION(1, 0, 0);
instanceInfo.flags = 0;
instanceInfo.pApplicationInfo = &appInfo;
instanceInfo.enabledLayerCount = instance_validation_layers_sz;
instanceInfo.ppEnabledLayerNames = instance_validation_layers_sz > 0 ? &instance_validation_layers[0] : NULL;
instanceInfo.enabledExtensionCount = (uint32_t)extension_names.size();
instanceInfo.ppEnabledExtensionNames = &extension_names[0];
result = vkCreateInstance(&instanceInfo, NULL, &m_instance);
```

Get Instance-Extension’s functions

```
m_CreateDebugReportCallback = (PFN_vkCreateDebugReportCallbackEXT) vkGetInstanceProcAddr(m_instance, "vkCreateDebugReportCallbackEXT");
m_DestroyDebugReportCallback = (PFN_vkDestroyDebugReportCallbackEXT) vkGetInstanceProcAddr(m_instance, "vkDestroyDebugReportCallbackEXT");
```

etc

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

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

HOW DOES IT LOOK ?

Device creation – enumerate physical devices; gather properties

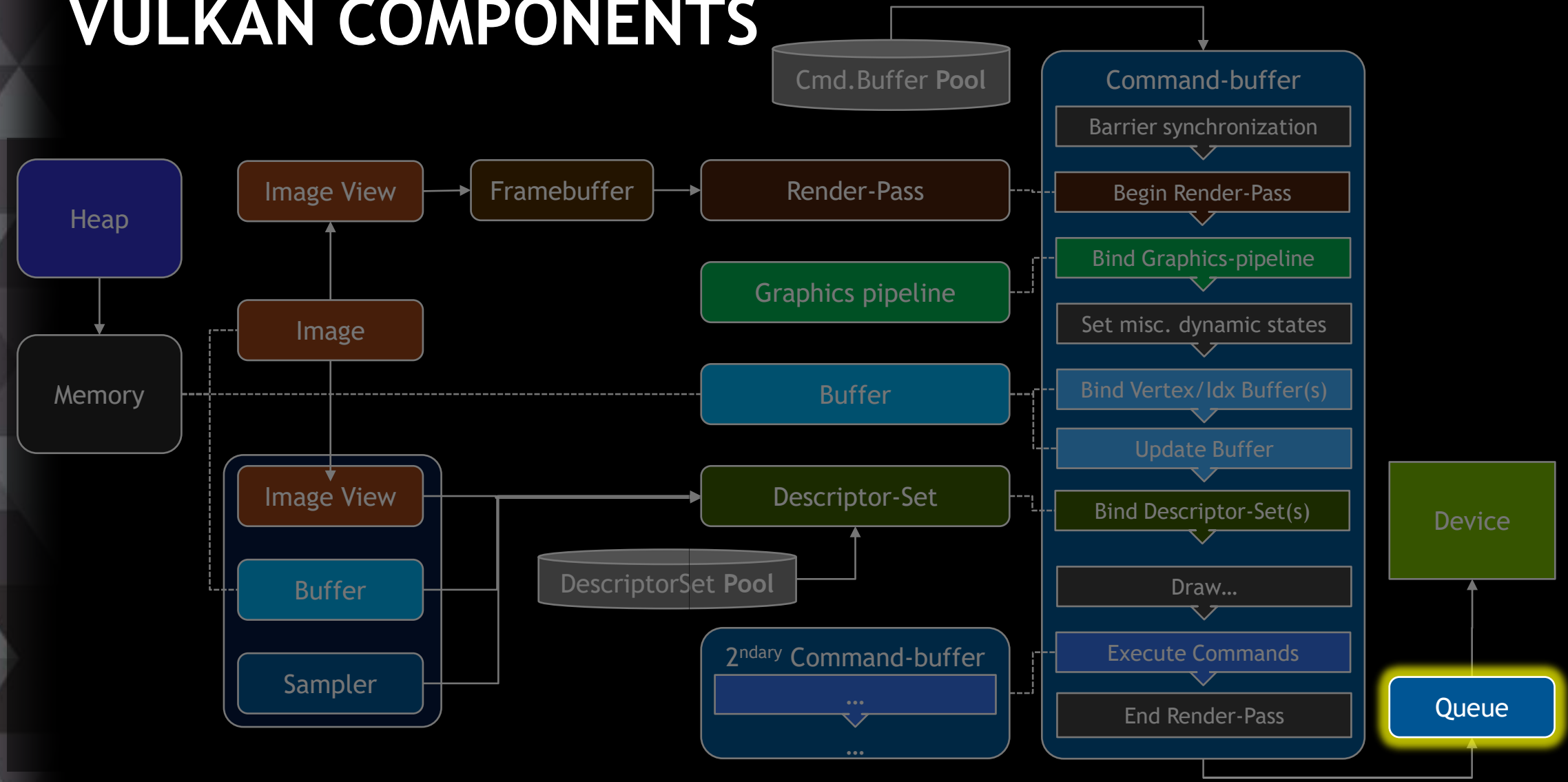
```
result = vkEnumeratePhysicalDevices(m_instance, &count, NULL);
physical_devices.resize(count);
result = vkEnumeratePhysicalDevices(m_instance, &count, &physical_devices[0]);
for(int j=0; j<physical_devices.size(); j++)
{
    vkGetPhysicalDeviceProperties(physical_devices[j], &m_gpu.properties);
    //...
    vkGetPhysicalDeviceMemoryProperties(physical_devices[j], &m_gpu.memoryProperties);
    //...
    result = vkEnumerateDeviceLayerProperties(physical_devices[j], &count, NULL);
    if(count > 0)
    {
        device_layers.resize(count);
        result = vkEnumerateDeviceLayerProperties(physical_devices[j], &count, &device_layers[0]);
        //...
    }
    result = vkEnumerateDeviceExtensionProperties(physical_devices[j], NULL, &count, NULL);
    device_extensions.resize(count);
    result = vkEnumerateDeviceExtensionProperties(physical_devices[j], NULL, &count, &device_extensions[0]);
    //...
}
```

HOW DOES IT LOOK ?

Device creation – Create the device (!)

```
queueInfo.queueFamilyIndex = queueFamilyIndex;
queueInfo.queueCount =  m_gpu.queueProperties[queueFamilyIndex].queueCount;
devInfo.queueCreateInfoCount = 1;
devInfo.pQueueCreateInfos = &queueInfo;
devInfo.enabledLayerCount = instance_validation_layers_sz;
devInfo.ppEnabledLayerNames = instance_validation_layers_sz > 0 ? &instance_validation_layers[0] : NULL;
devInfo.enabledExtensionCount = (uint32_t)extension_names.size();
devInfo.ppEnabledExtensionNames = &(extension_names[0]);
result = ::vkCreateDevice(m_gpu.device, &devInfo, NULL, &m_device);
```

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”

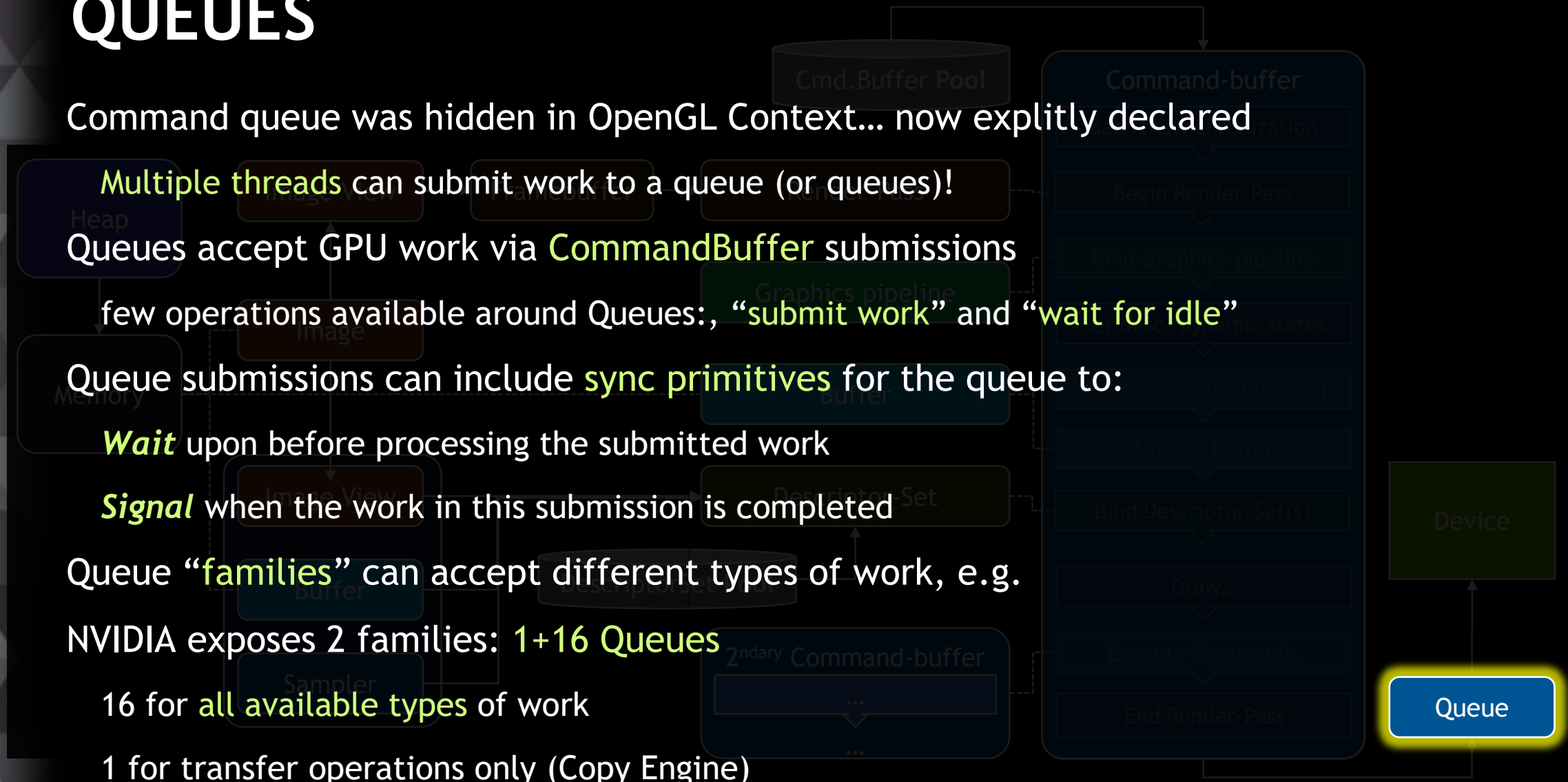
Queue submissions can include sync primitives for the queue to:

- ***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)



HOW DOES IT LOOK ?

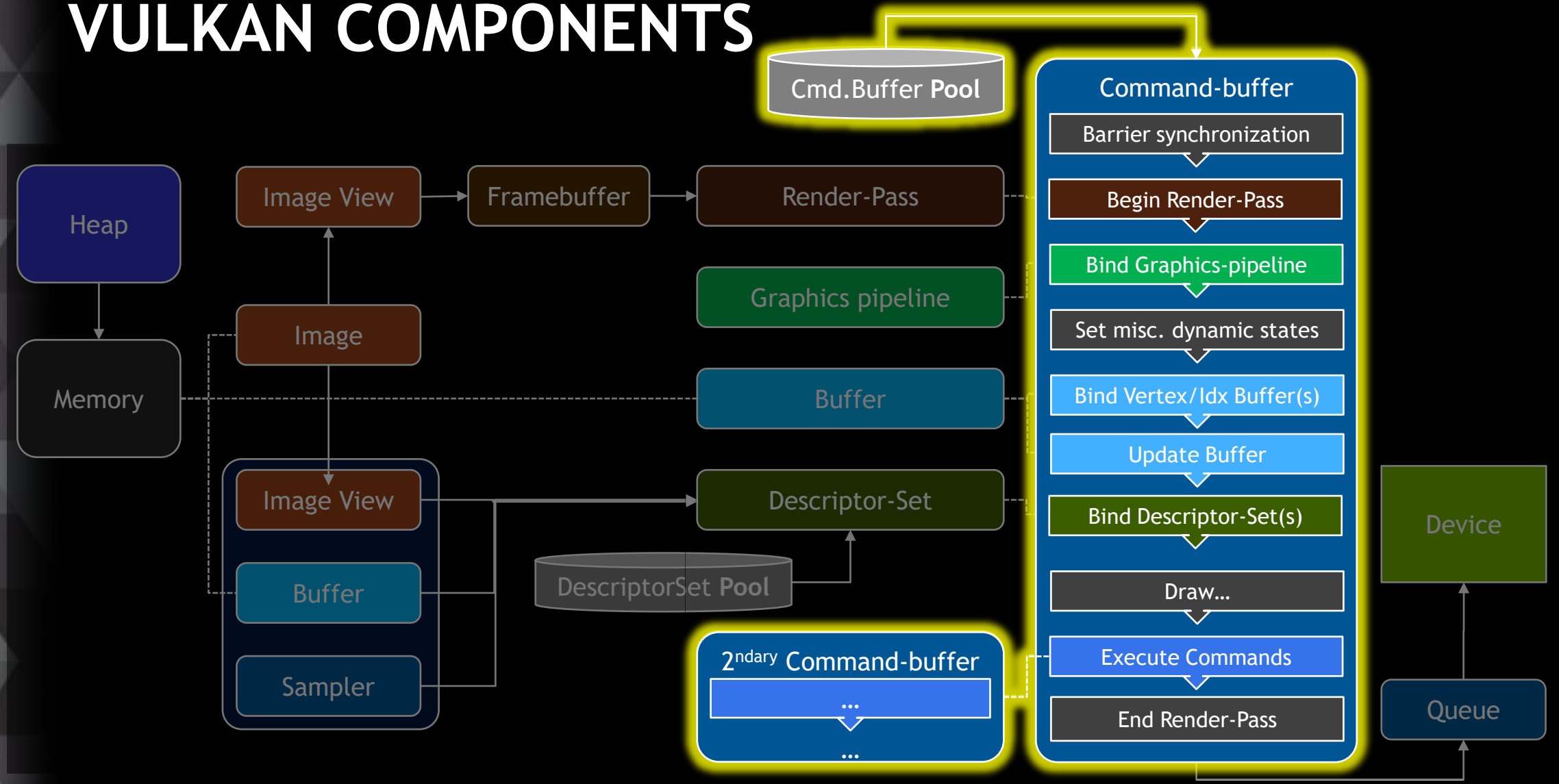
Queue(s)

```
// at Init time (after Device creation...)
vkGetDeviceQueue(m_device, 0, 0, &m_queue);
```

```
::VkSubmitInfo submitInfo = { VK_STRUCTURE_TYPE_SUBMIT_INFO, NULL };
submitInfo.waitSemaphoreCount = 0;
submitInfo.pWaitSemaphores = NULL;
submitInfo.commandBufferCount = 1;
submitInfo.pCommandBuffers = &cmd;
submitInfo.signalSemaphoreCount = 0;
submitInfo.pSignalSemaphores = NULL;

CHECK(::vkQueueSubmit(m_queue, 1, &submitInfo, VK_NULL_HANDLE) );
```

```
vkDeviceWaitIdle(m_device);
```

VULKAN COMPONENTS



SYNCHRONIZATION

events and **barriers**

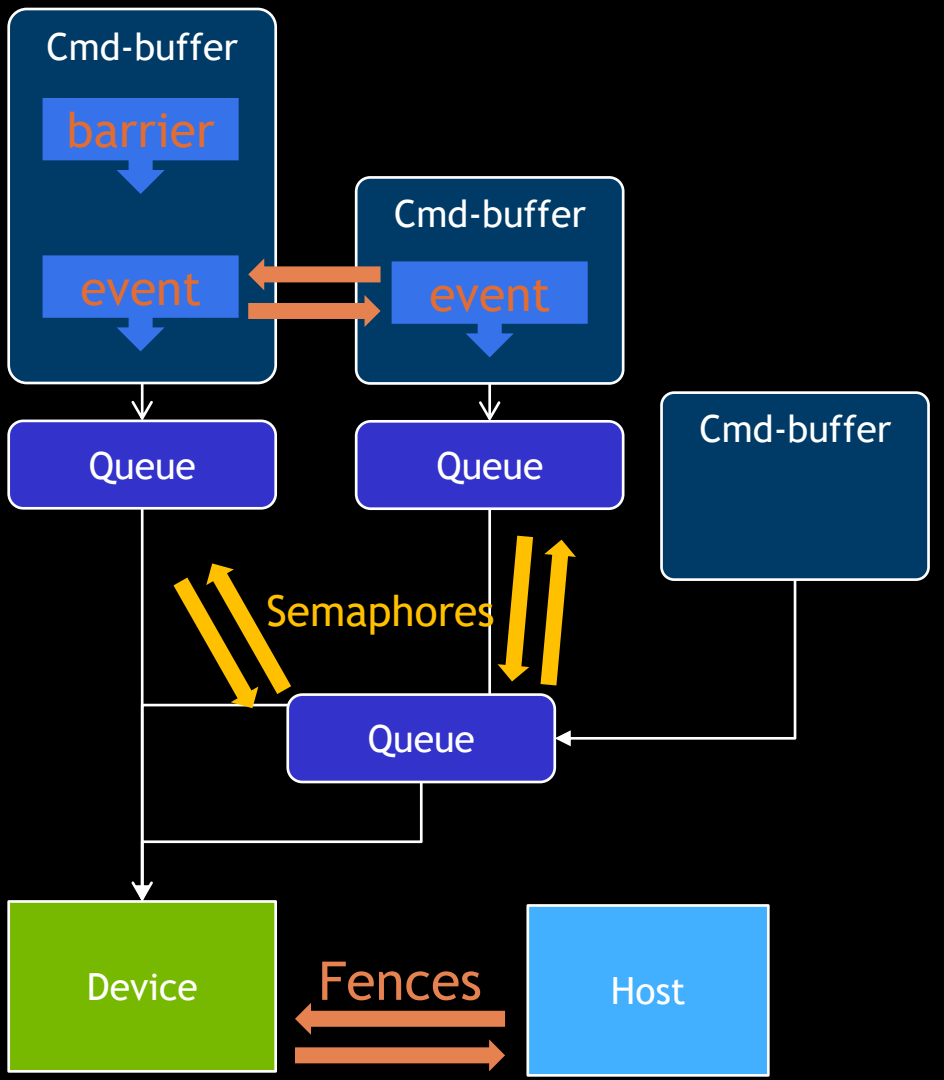
used to synchronize work within a command buffer or sequence of command buffers submitted to a single queue

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)

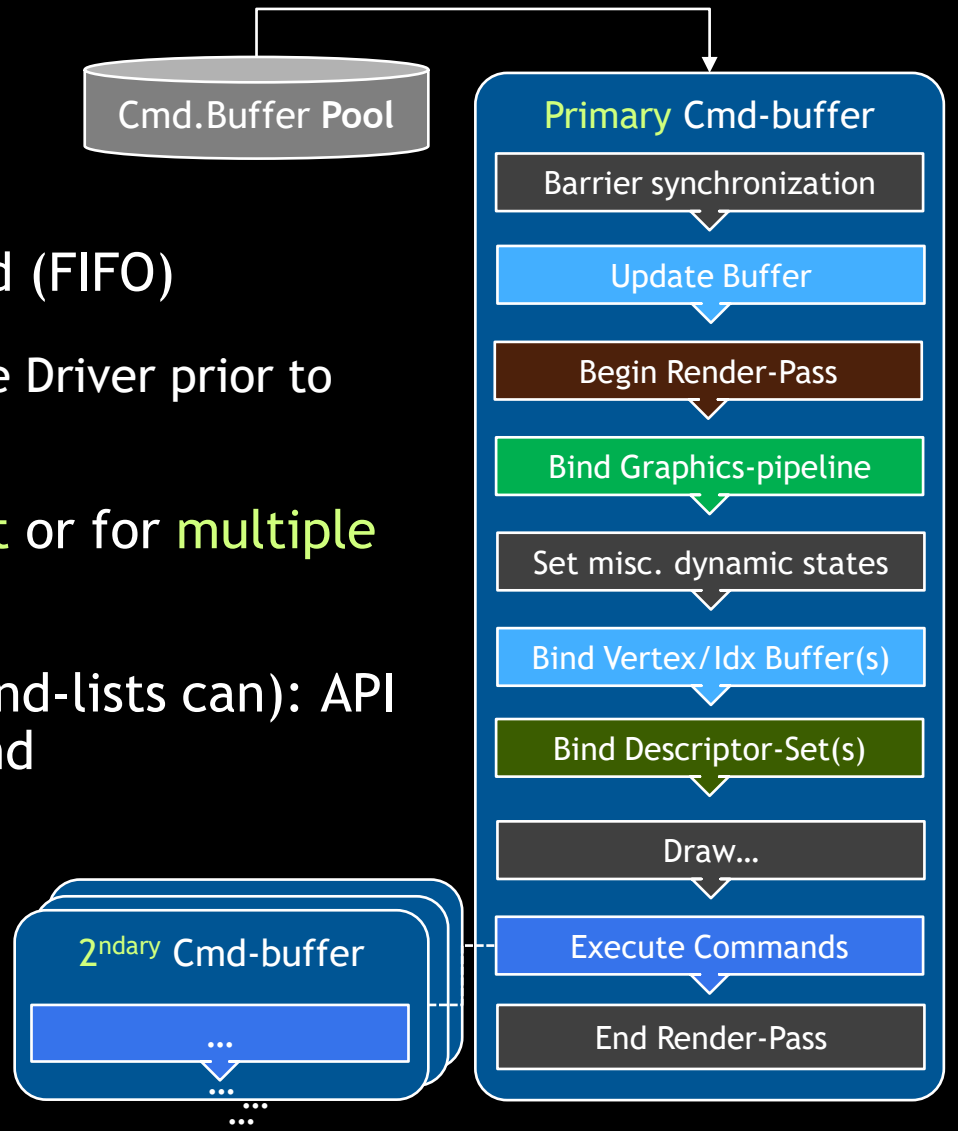
Minor translation & optimization from the Driver prior to sending to the GPU

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

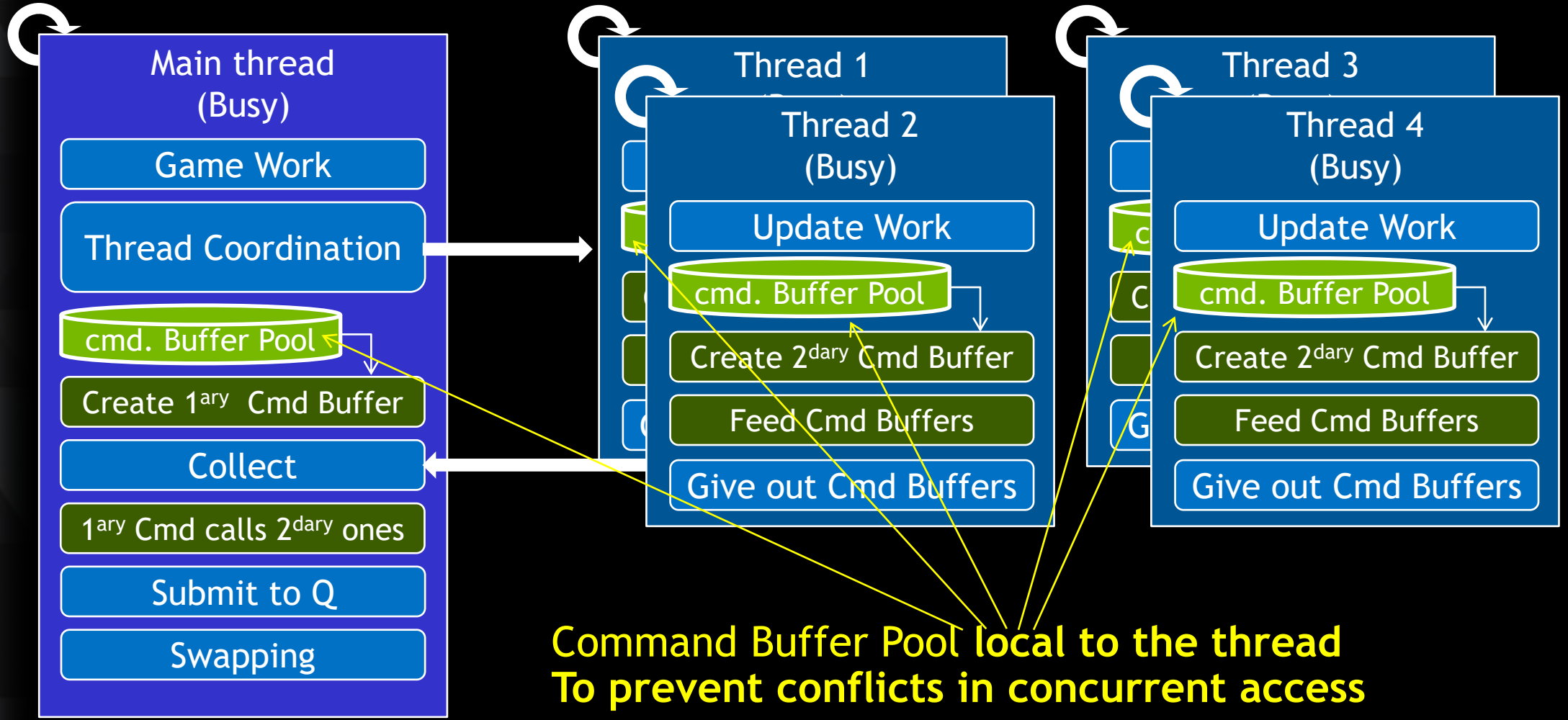
Cannot Cmd-Buffers from GPU (command-lists can): API calls to vkCmd…() between Begin & End

Multi-threading friendly (!)

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



COMMAND-BUFFERS AND MULTI-THREADING



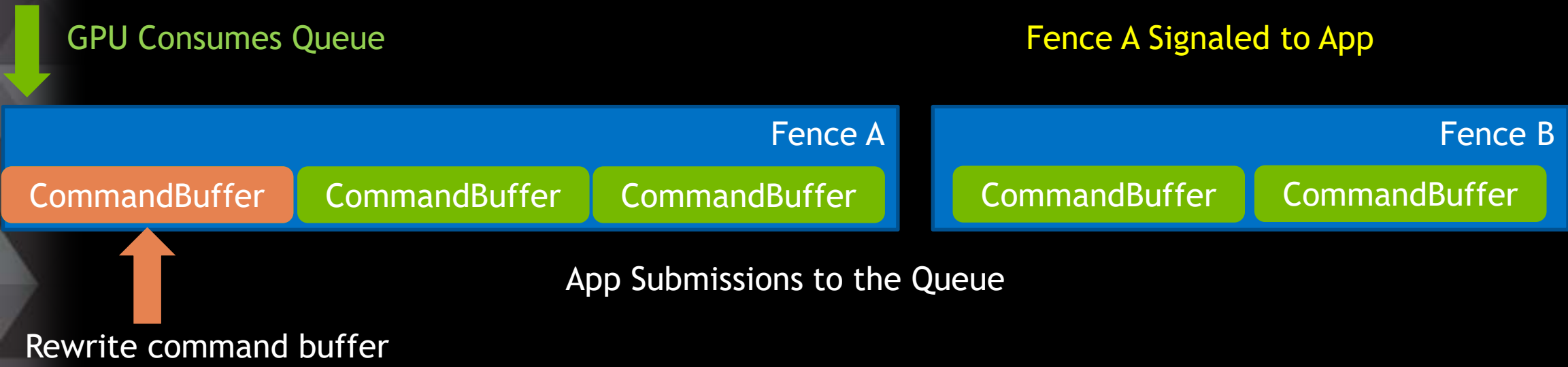
Command Buffer Pool **local to the thread**
To prevent conflicts in concurrent access

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



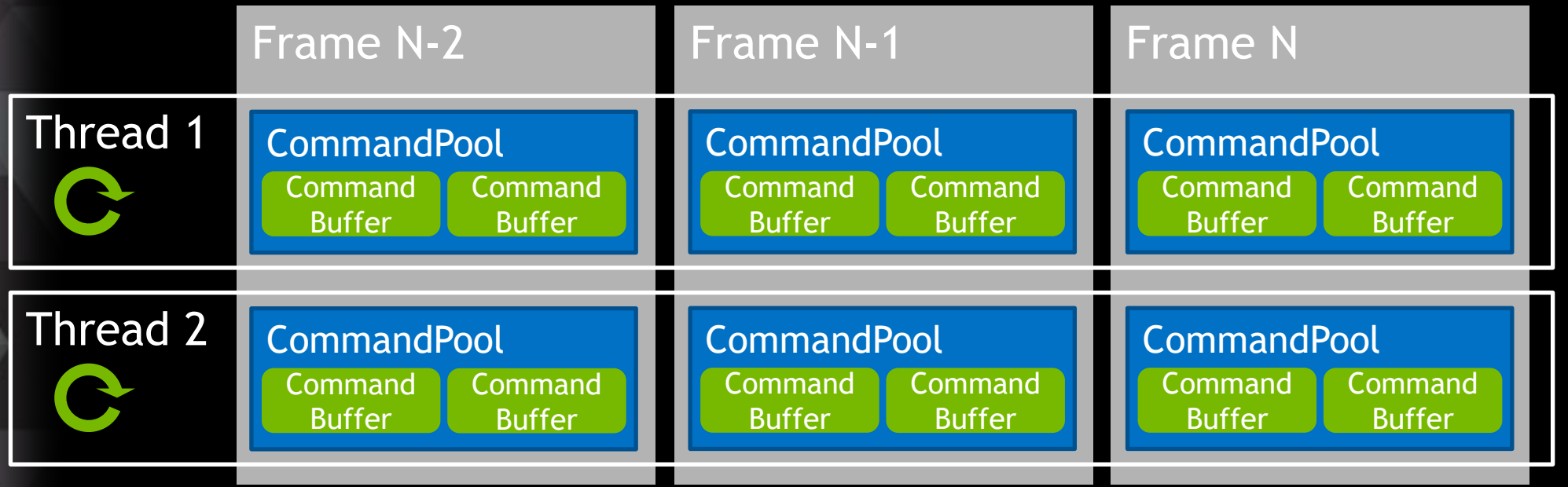
THREADS AND COMMAND POOLS

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



HOW DOES IT LOOK ?

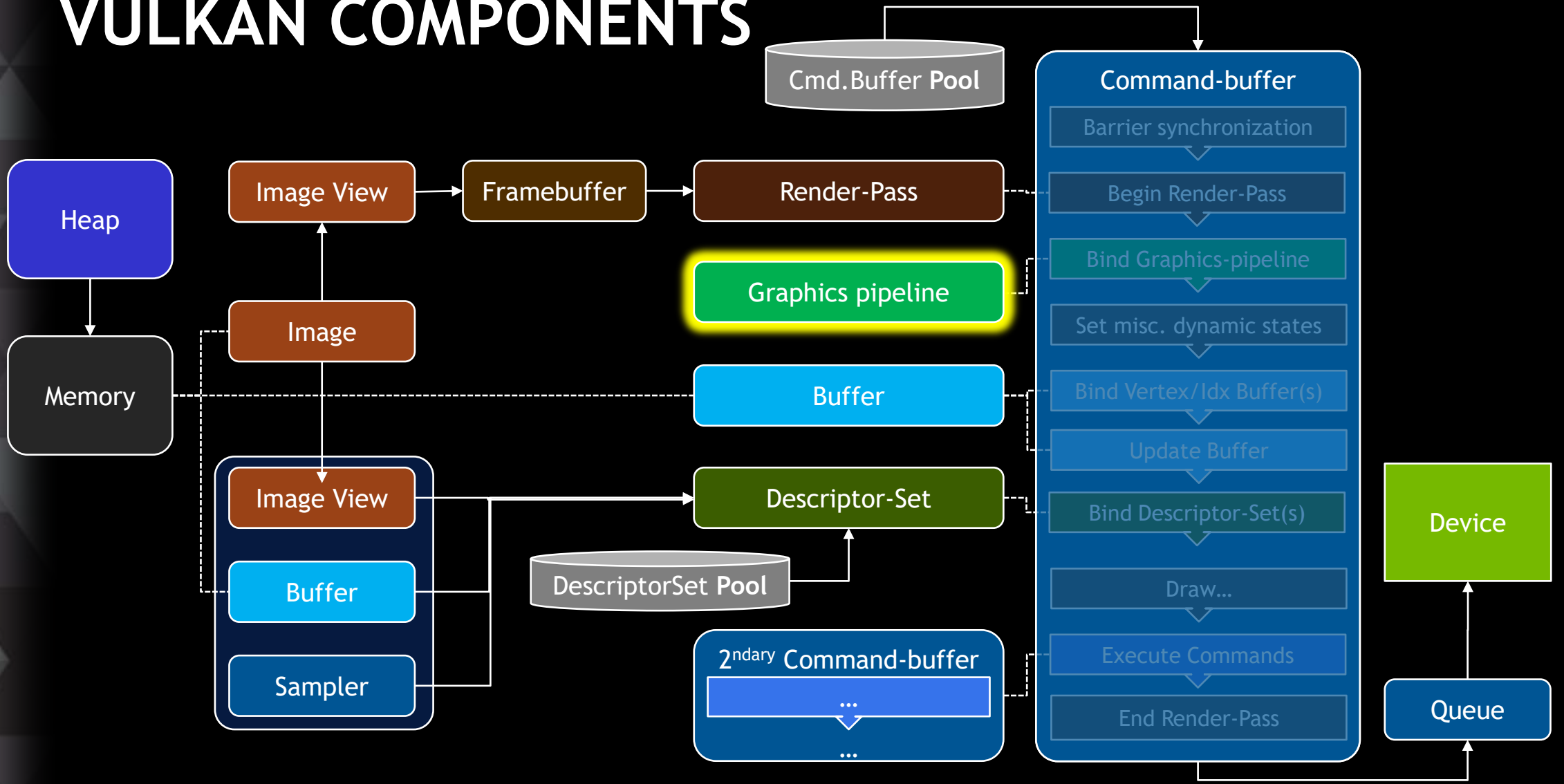
Command-Buffers

```
m_perThreadData = new PerThreadData;
VkCommandPoolCreateInfo cmdPoolInfo = { VK_STRUCTURE_TYPE_COMMAND_POOL_CREATE_INFO };
cmdPoolInfo.queueFamilyIndex = 0;
cmdPoolInfo.flags = VK_COMMAND_POOL_CREATE_RESET_COMMAND_BUFFER_BIT;
result = vkCreateCommandPool(nvk.m_device, &cmdPoolInfo, NULL, &m_perThreadData->m_cmdPoolStatic);
```

```
nvk.vkResetCommandPool(m_perThreadData->m_curCmdPoolDynamic, 0);
m_cmdScene = nvk.vkAllocateCommandBuffer(m_perThreadData->m_curCmdPoolDynamic, true);
nvk.vkBeginCommandBuffer(m_cmdScene, false, NVK::VkCommandBufferInheritanceInfo(renderPass, 0, framebuffer, VK_FALSE, 0, 0) );
vkCmdUpdateBuffer   (m_cmdScene, m_matrix.buffer, 0, sizeof(g_globalMatrices), (uint32_t*)&g_globalMatrices);
vkCmdBeginRenderPass(m_cmdScene,
    NVK::VkRenderPassBeginInfo(
    renderPass, framebuffer, viewRect,
        NVK::VkClearValue(NVK::VkClearColorValue(0.0f, 0.0f, 0.0f, 1.0f))
                          (NVK::VkClearDepthStencilValue(1.0, 0))),
    VK_SUBPASS_CONTENTS_INLINE );
vkCmdSetViewport( m_cmdScene, 0, 1, NVK::VkViewport(0.0, 0.0, w, h, 0.0f, 1.0f) );
vkCmdSetScissor(  m_cmdScene, 0, 1, NVK::VkRect2D(0.0, 0.0, w, h ) );

vkCmdExecuteCommands(m_cmdScene, 1, &m_cmdBufferBgnd);
//...
vkCmdEndRenderPass(m_cmdScene);
vkEndCommandBuffer(m_cmdScene);
```

Note: helpers (structs wrappers) to use Constructors & functors for compact data declaration
See **NVK.h/cpp** in https://github.com/nvpro-samples

VULKAN COMPONENTS



GRAPHICS PIPELINE

Snapshot of all States

Including Shaders

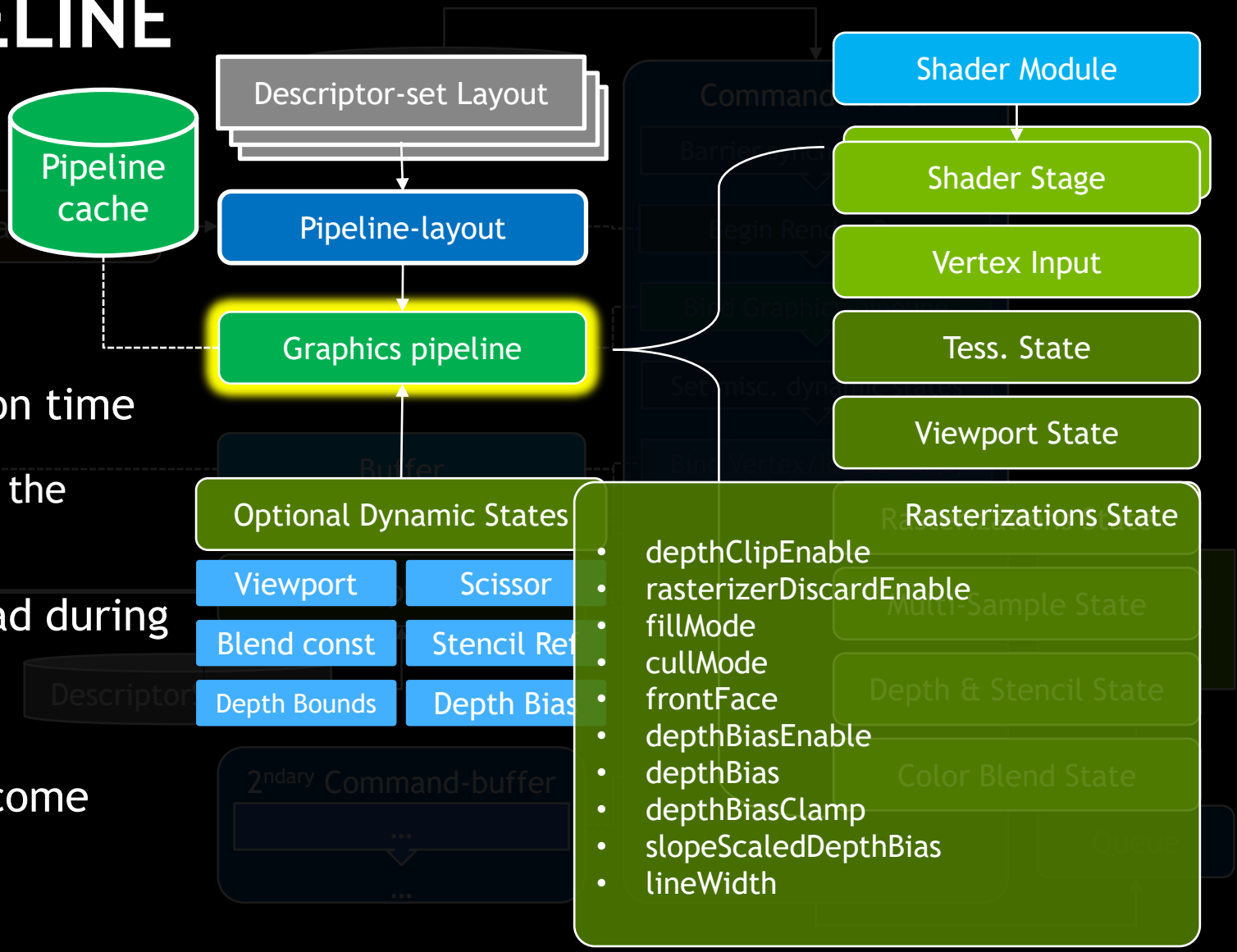
Pre-compiled & Immutable

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

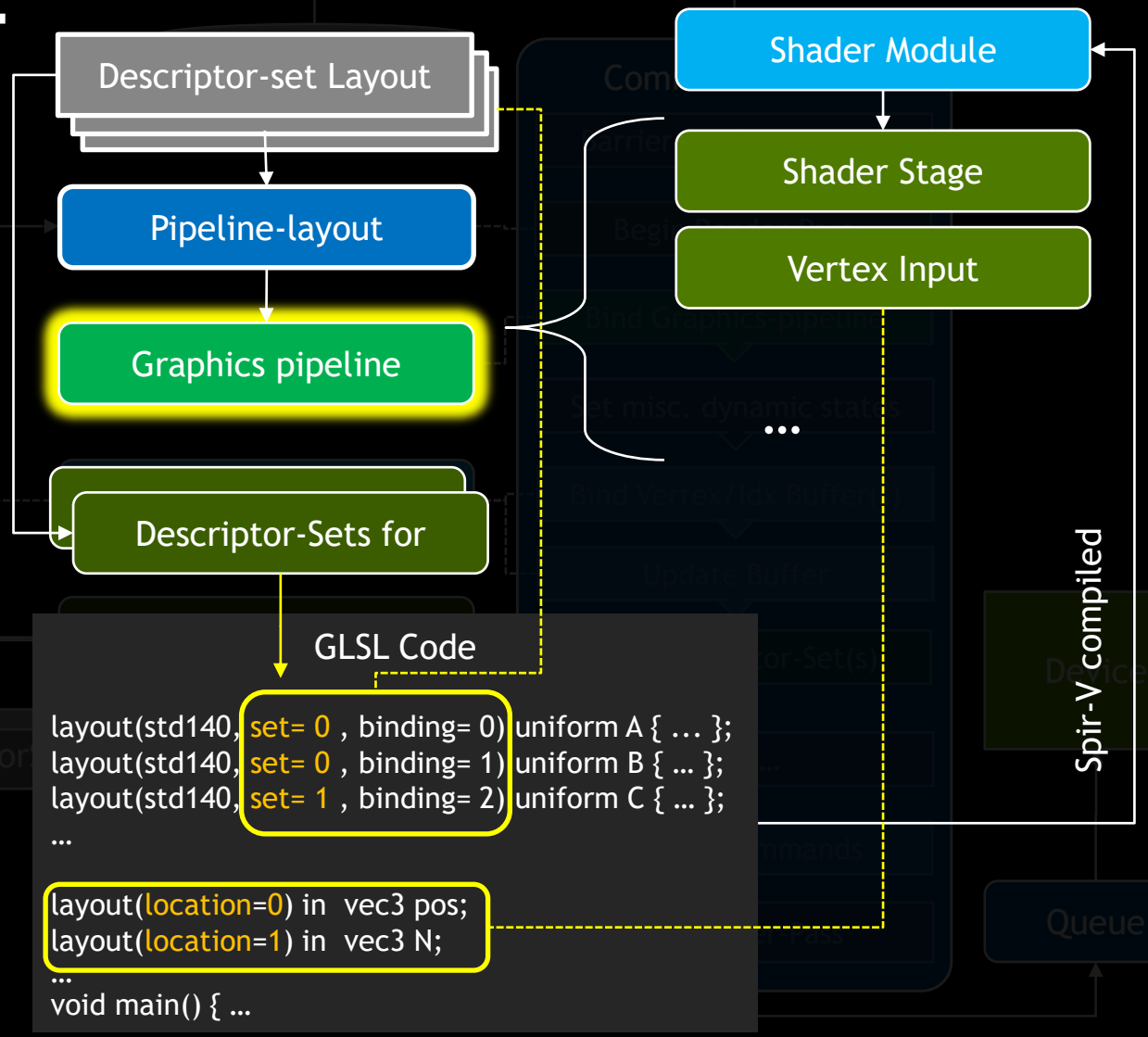
No “introspection”, so everything known & prepared in advance

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…)



HOW DOES IT LOOK ?

Graphics pipeline - setup

```
m_descriptorSetLayouts[DSET_GLOBAL] = nvk.vkCreateDescriptorSetLayout(
    NVK::VkDescriptorSetLayoutCreateInfo(NVK::VkDescriptorSetLayoutBinding
     (BINDING_MATRIX, VK_DESCRIPTOR_TYPE_UNIFORM_BUFFER, 1, VK_SHADER_STAGE_VERTEX_BIT) // BINDING_MATRIX
     //(0, VK_DESCRIPTOR_TYPE_UNIFORM_BUFFER, 1, VK_SHADER_STAGE_FRAGMENT_BIT) // BINDING_LIGHT
     (BINDING_CUBETEX, VK_DESCRIPTOR_TYPE_COMBINED_IMAGE_SAMPLER, 1, VK_SHADER_STAGE_FRAGMENT_BIT)
) );
m_descriptorSetLayouts[DSET_OBJECT] = nvk.vkCreateDescriptorSetLayout(
    NVK::VkDescriptorSetLayoutCreateInfo(NVK::VkDescriptorSetLayoutBinding
     (BINDING_MATRIXOBJ, VK_DESCRIPTOR_TYP
     (BINDING_MATERIAL , VK_DESCRIPTOR_TYP
) );

  m_pipelineLayout     = nvk.vkCreatePipe
```

```
NVK::VkPipelineDynamicStateCreateInfo dynamicStateCreateInfo(
        NVK::VkDynamicState
            (VK_DYNAMIC_STATE_VIEWPORT)
            (VK_DYNAMIC_STATE_SCISSOR)
            (VK_DYNAMIC_STATE_LINE_WIDTH)
            (VK_DYNAMIC_STATE_DEPTH_BIAS)
    );
NVK::VkPipelineRasterizationStateCreateInfo vkPipelineRasterStateCreateInfo(
    VK_TRUE, /*depthClipEnable*/       VK_FALSE, //rasterizerDiscardEnable
    VK_POLYGON_MODE_FILL, /*fillMode*/ VK_CULL_MODE_NONE,  //cullMode
    VK_FRONT_FACE_COUNTER_CLOCKWISE,  //frontFace
    VK_TRUE, //depthBiasEnable
    0.0, /*depthBias*/ 0.0, //depthBiasClamp
    0.0, /*slopeScaledDepthBias*/ 1.0 /*lineWidth*/ );
```

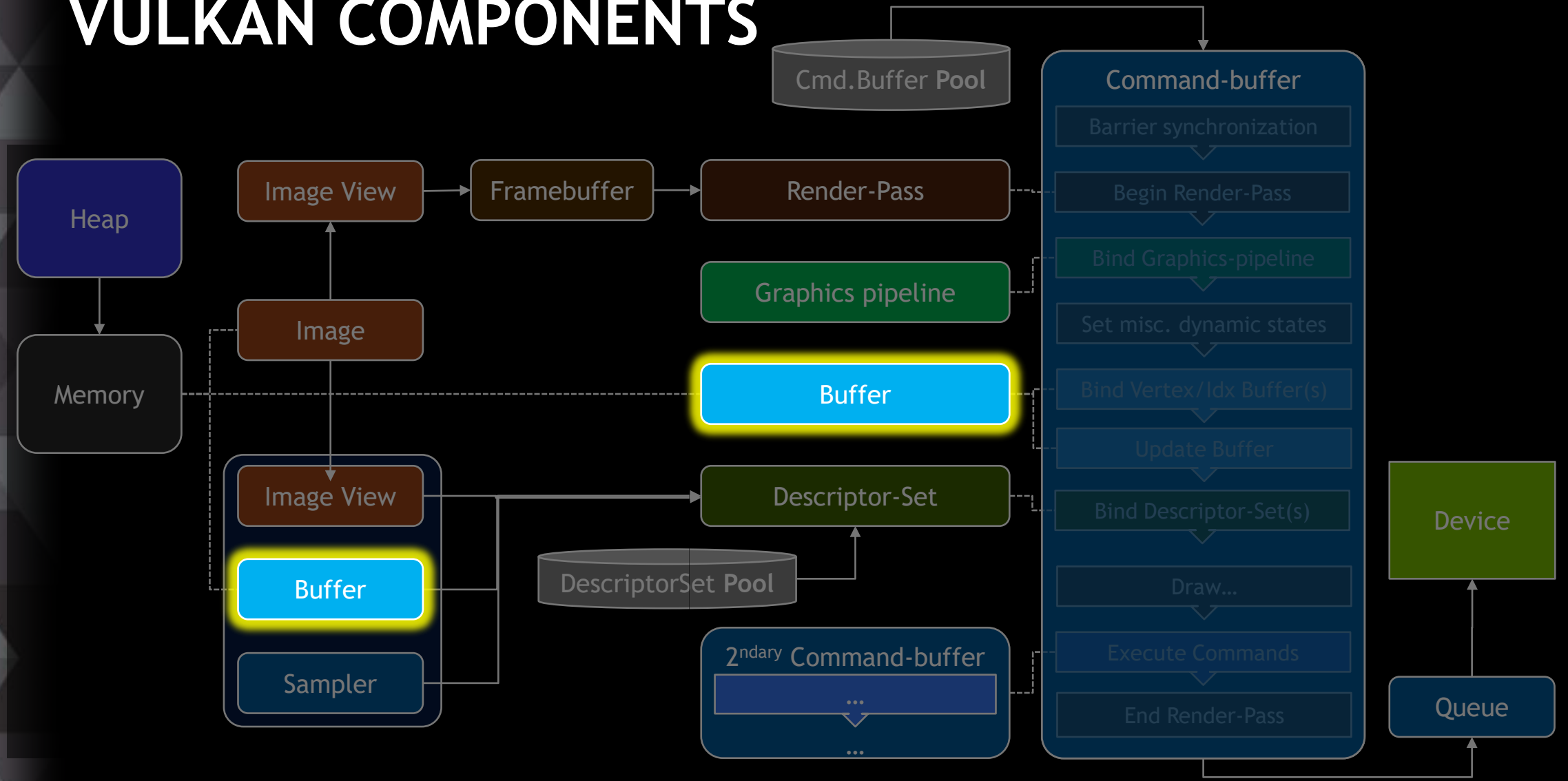
HOW DOES IT LOOK ?

Graphics pipeline cr

```
m_pipelineGrid = nvk.vkCreateGraphicsPi
    (m_pipelineLayout,0)
    (NVK::VkPipelineVertexInputStateCrea
        NVK::VkVertexInputBindingDescri
        NVK::VkVertexInputAttributeDesc
    ) )
    (NVK::VkPipelineInputAssemblyStateC
    (NVK::VkPipelineShaderStageCreateIn
        VK_SHADER_STAGE_VERTEX_BIT, nvk
    (vkPipelineViewportStateCreateInfo)
    (vkPipelineRasterStateCreateInfo)
    (vkPipelineMultisampleStateCreateIn
    (NVK::VkPipelineShaderStageCreateIn
        VK_SHADER_STAGE_FRAGMENT_BIT, nv
    (vkPipelineColorBlendStateCreateInf
    (vkPipelineDepthStencilStateCreateI
    (dynamicStateCreateInfo)
    );
```

```
m_pipelineCubeBgnd = nvk.vkCreateGraphicsPipeline(NVK::VkGraphicsPipelineCreateInfo
    (m_pipelineLayout, m_scenePass)
    (NVK::VkPipelineVertexInputStateCreateInfo(
        NVK::VkVertexInputBindingDescription    (0/*binding*/, sizeof(Attribs)/*stride*/, VK_VERTEX_INPUT_RATE_VERTEX),
        NVK::VkVertexInputAttributeDescription  (0/*location*/, 0/*binding*/, VK_FORMAT_R32G32B32_SFLOAT, 0/*offsetbytes*/)
                                               ,(1/*location*/, 0/*binding*/, VK_FORMAT_R32G32_SFLOAT, 12/*offsetbytes*/)
    ) )
    (NVK::VkPipelineInputAssemblyStateCreateInfo(VK_PRIMITIVE_TOPOLOGY_TRIANGLE_LIST, VK_FALSE) )
    (NVK::VkPipelineShaderStageCreateInfo(
        VK_SHADER_STAGE_VERTEX_BIT, nvk.vkCreateShaderModule(spv_GLSL_bgnd_vert.c_str(), spv_GLSL_bgnd_vert.size()), "main") )
    (vkPipelineViewportStateCreateInfo)
    (vkPipelineRasterStateCreateInfo)
    (NVK::VkPipelineRasterizationStateCreateInfo(
        VK_FALSE,            //depthClipEnable
        VK_FALSE,           //rasterizerDiscardEnable
        VK_POLYGON_MODE_FILL, //fillMode
        VK_CULL_MODE_NONE,  //cullMode
        VK_FRONT_FACE_COUNTER_CLOCKWISE,  //frontFace
        VK_FALSE,           //depthBiasEnable
        0.0,                //depthBias
        0.0,                //depthBiasClamp
        0.0,                //slopeScaledDepthBias
        1.0                 //lineWidth
    ))
    (vkPipelineMultisampleStateCreateInfo)
    (NVK::VkPipelineShaderStageCreateInfo(
        VK_SHADER_STAGE_FRAGMENT_BIT, nvk.vkCreateShaderModule(spv_GLSL_bgnd_frag.c_str(), spv_GLSL_bgnd_frag.size()), "main") )
    (vkPipelineColorBlendStateCreateInfo)
    (NVK::VkPipelineDepthStencilStateCreateInfo(
        VK_FALSE,                   //depthTestEnable
        VK_FALSE,                   //depthWriteEnable
        VK_COMPARE_OP_ALWAYS,       //depthCompareOp
        VK_FALSE,                   //depthBoundsTestEnable
        VK_FALSE,                  //stencilTestEnable
        NVK::VkStencilOpState(), NVK::VkStencilOpState(), //front, back
        0.0f, 1.0f                  //minDepthBounds, maxDepthBounds
    ))
    (dynamicStateCreateInfo)
    );
```

VULKAN COMPONENTS



BUFFERS

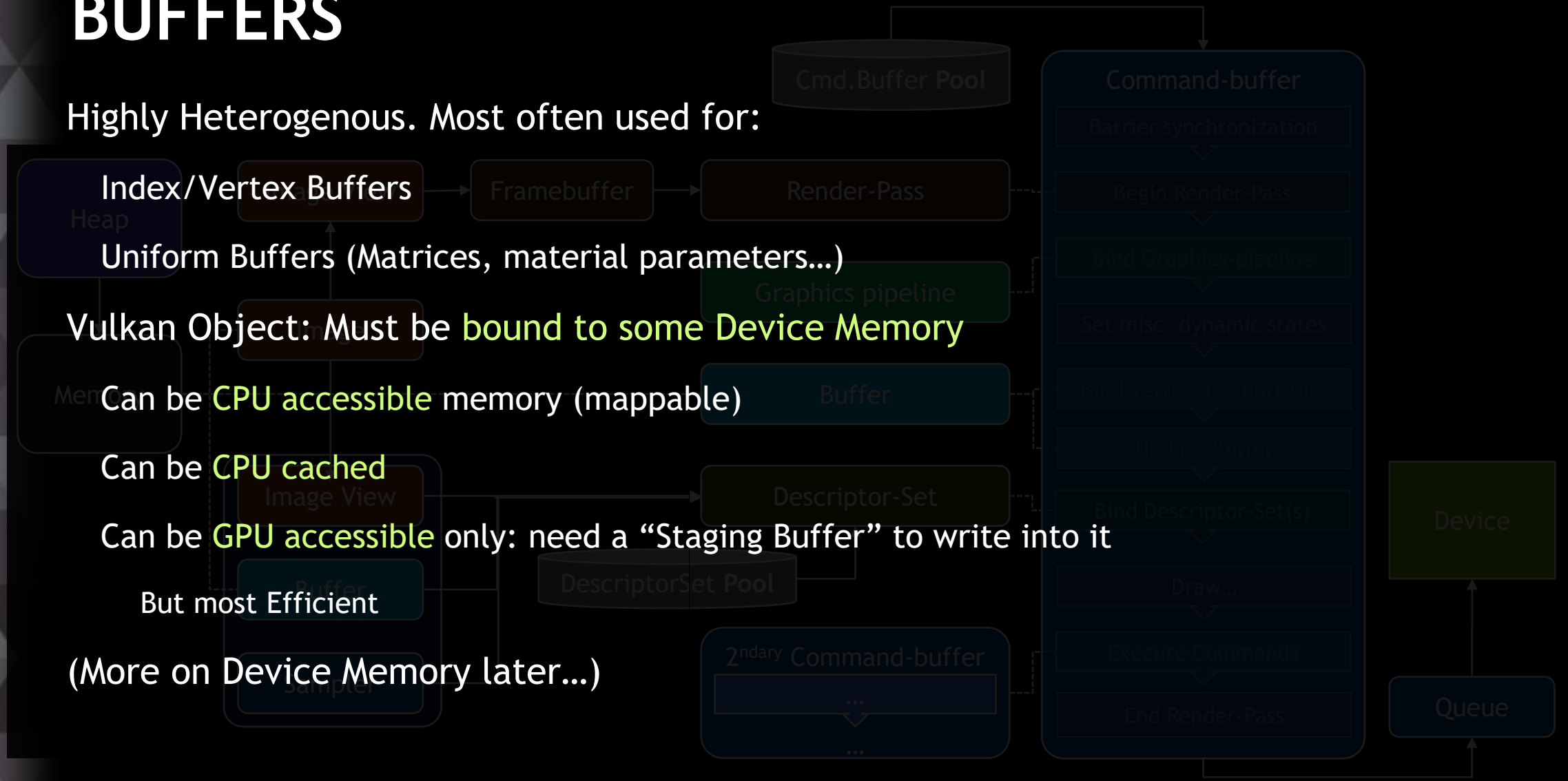
Highly Heterogenous. Most often used for:

- 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

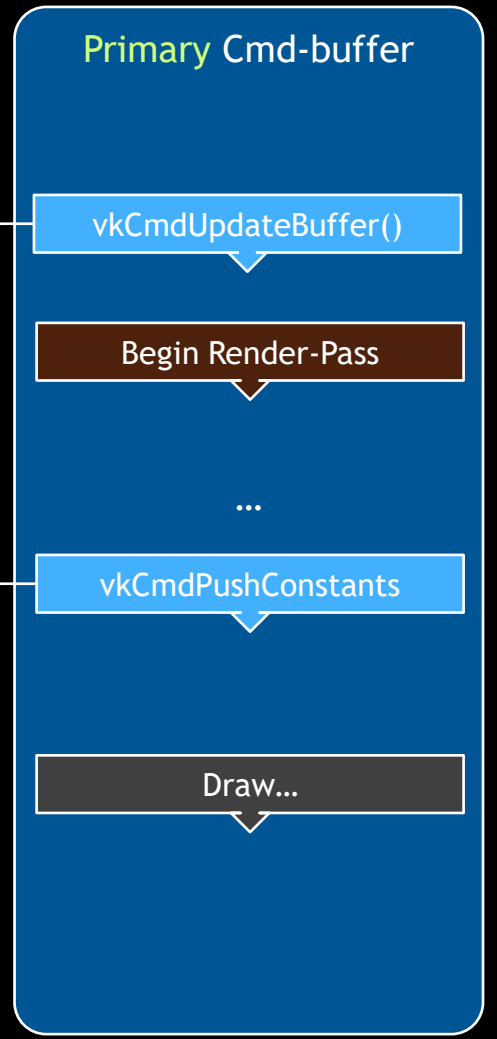
```
layout(set=0 , binding = 2 ) uniform MyBuffer {
   mat4 mW;
…
```

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



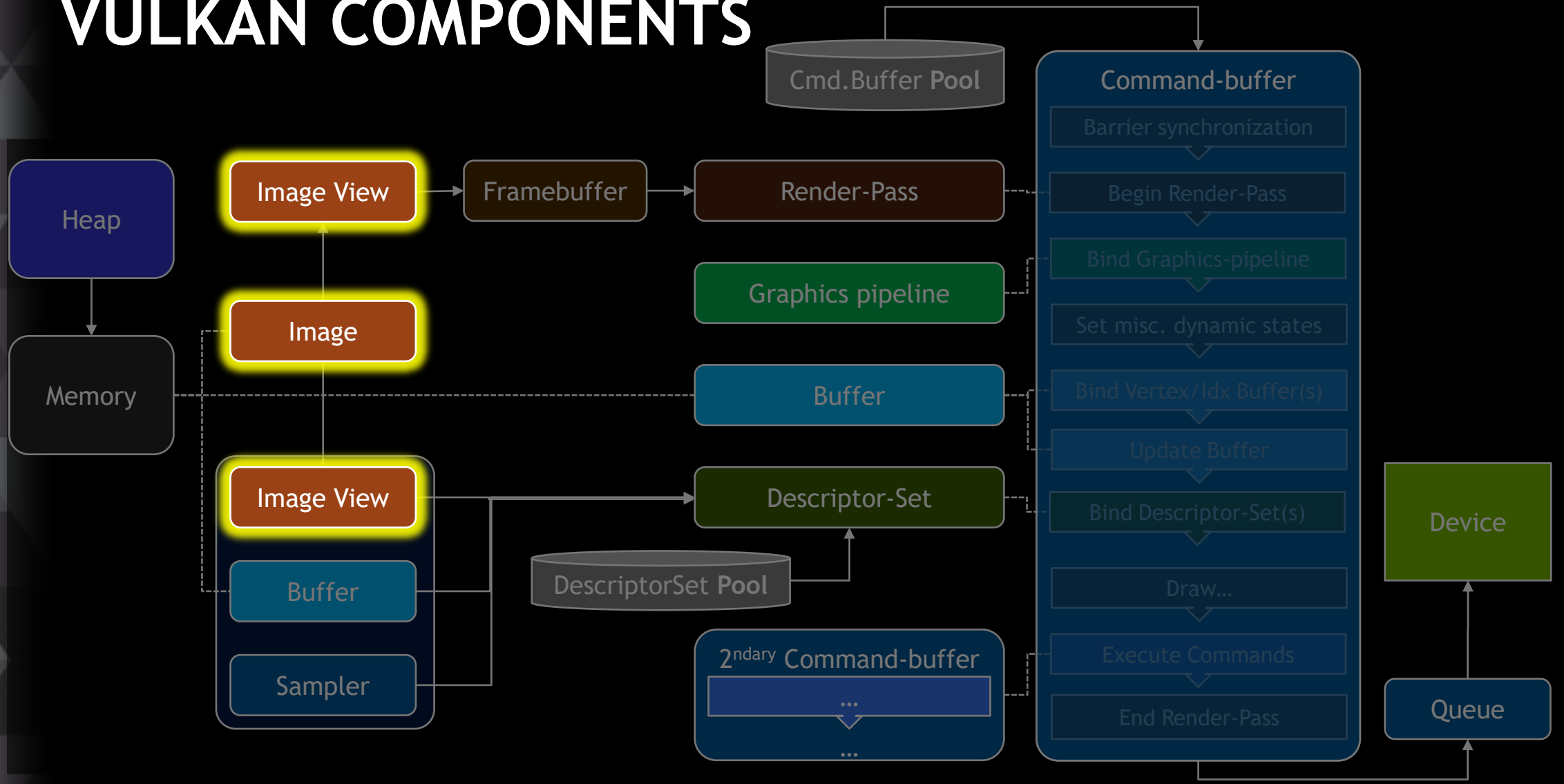
HOW DOES IT LOOK ?

Buffers – copy data in a buffer

- Create Buffer
- Create staging buffer
- Bind objects to some memory
- Command buffer for copy
- Enqueue command buffer
- Note: could be put with other graphic commands for better efficiency

```
VkBuffer buffer; // may not be accessible from host
VkBufferCreateInfo bufferInfo(size, usage);
CHECK(::vkCreateBuffer(m_device, &bufferInfo, NULL, &buffer) );
bufferMem = allocMemAndBindObject(buffer, memProps);
// Create staging buffer: accessible from host
VkBuffer bufferStage;
VkBufferCreateInfo bufferStageInfo(size, VK_BUFFER_USAGE_TRANSFER_SRC_BIT);
CHECK(::vkCreateBuffer(m_device, &bufferStageInfo, NULL, &bufferStage) );
// Allocate and bind to the buffer
::VkDeviceMemory bufferStageMem;
bufferStageMem = allocMemAndBindObject(bufferStage, (::VkFlags)VK_MEMORY_PROPERTY_HOST_VISIBLE_BIT);
vkMemcpy(bufferStageMem, data, size);
::VkCommandBuffer cmd;
cmd = vkAllocateCommandBuffer(cmdPool, true);
vkBeginCommandBuffer(cmd, true);
    vkCmdCopyBuffer(cmd, bufferStage, buffer, size, offset);
vkEndCommandBuffer(cmd);

::VkSubmitInfo submitInfo = { VK_STRUCTURE_TYPE_SUBMIT_INFO, NULL };
submitInfo.waitSemaphoreCount = 0;
submitInfo.pWaitSemaphores = NULL;
submitInfo.commandBufferCount = 1;
submitInfo.pCommandBuffers = &cmd;
submitInfo.signalSemaphoreCount = 0;
submitInfo.pSignalSemaphores = NULL;

CHECK(::vkQueueSubmit(m_queue, 1, &submitInfo, VK_NULL_HANDLE) );
vkDeviceWaitIdle(m_device);
// release stuff
vkFreeCommandBuffer(cmdPool, cmd);
vkDestroyBuffer(bufferStage);
::vkFreeMemory(m_device, bufferStageMem, NULL);
```

VULKAN COMPONENTS



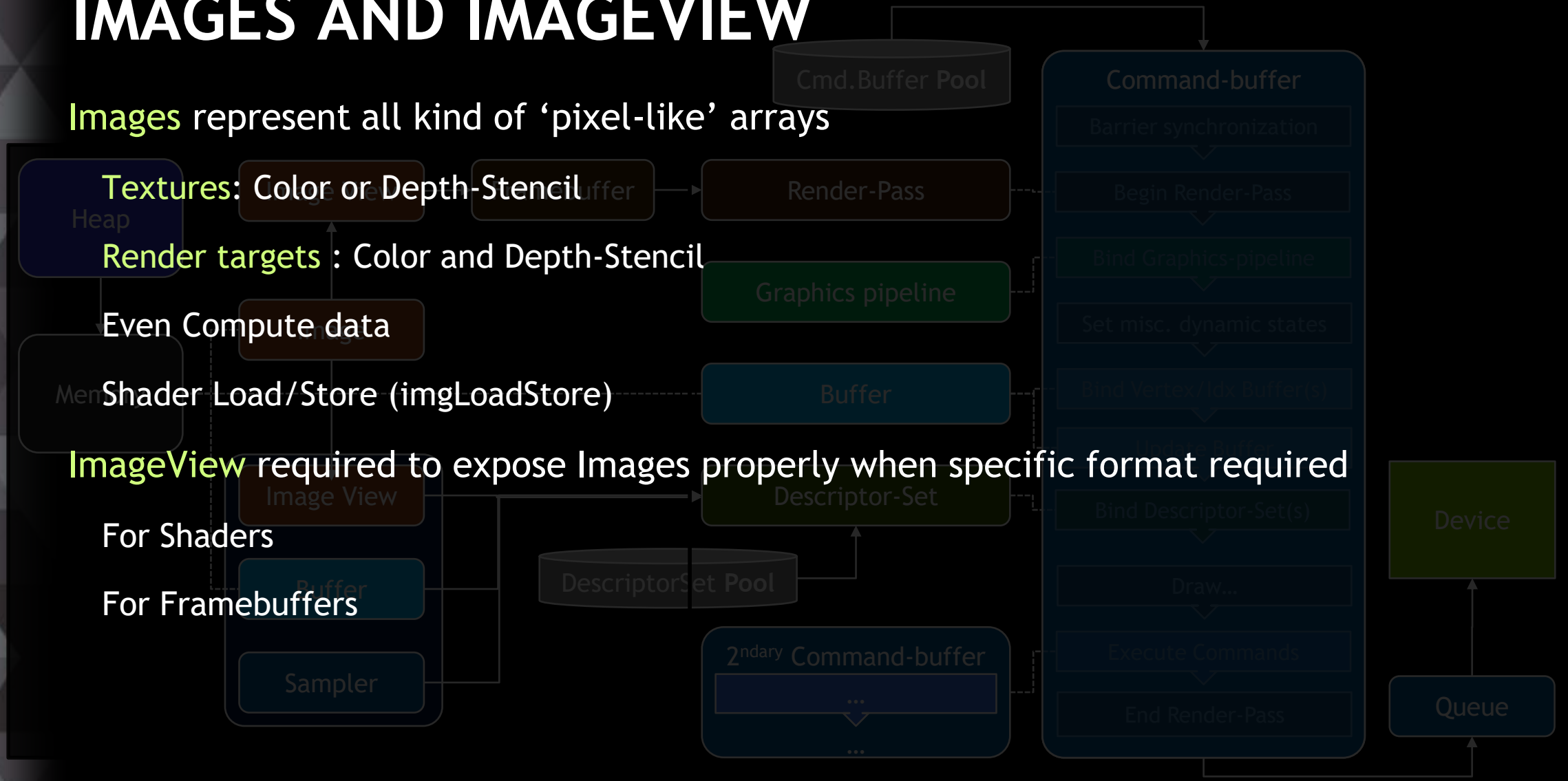
IMAGES AND IMAGEVIEW

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

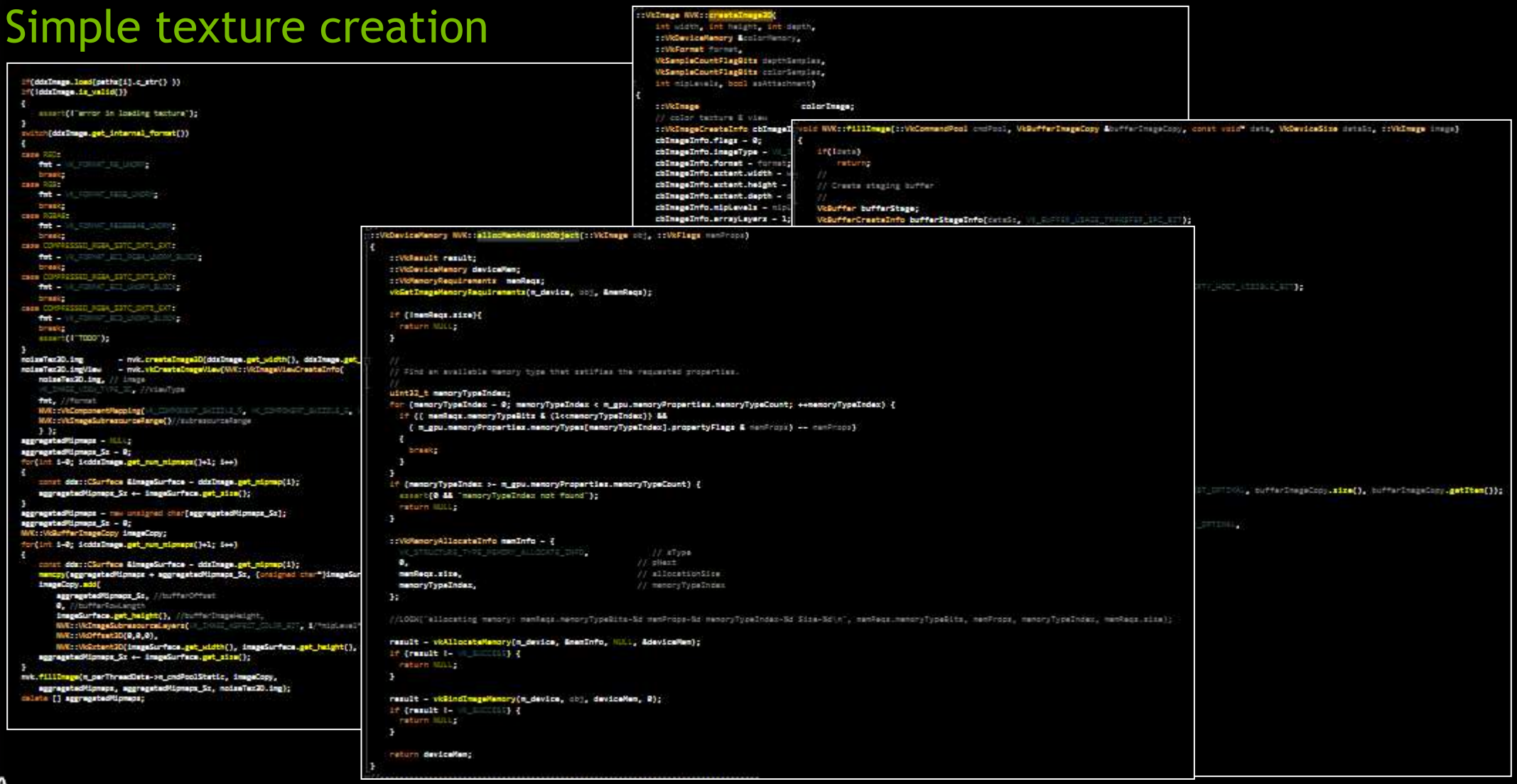
Way more complex than OpenGL !

- 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

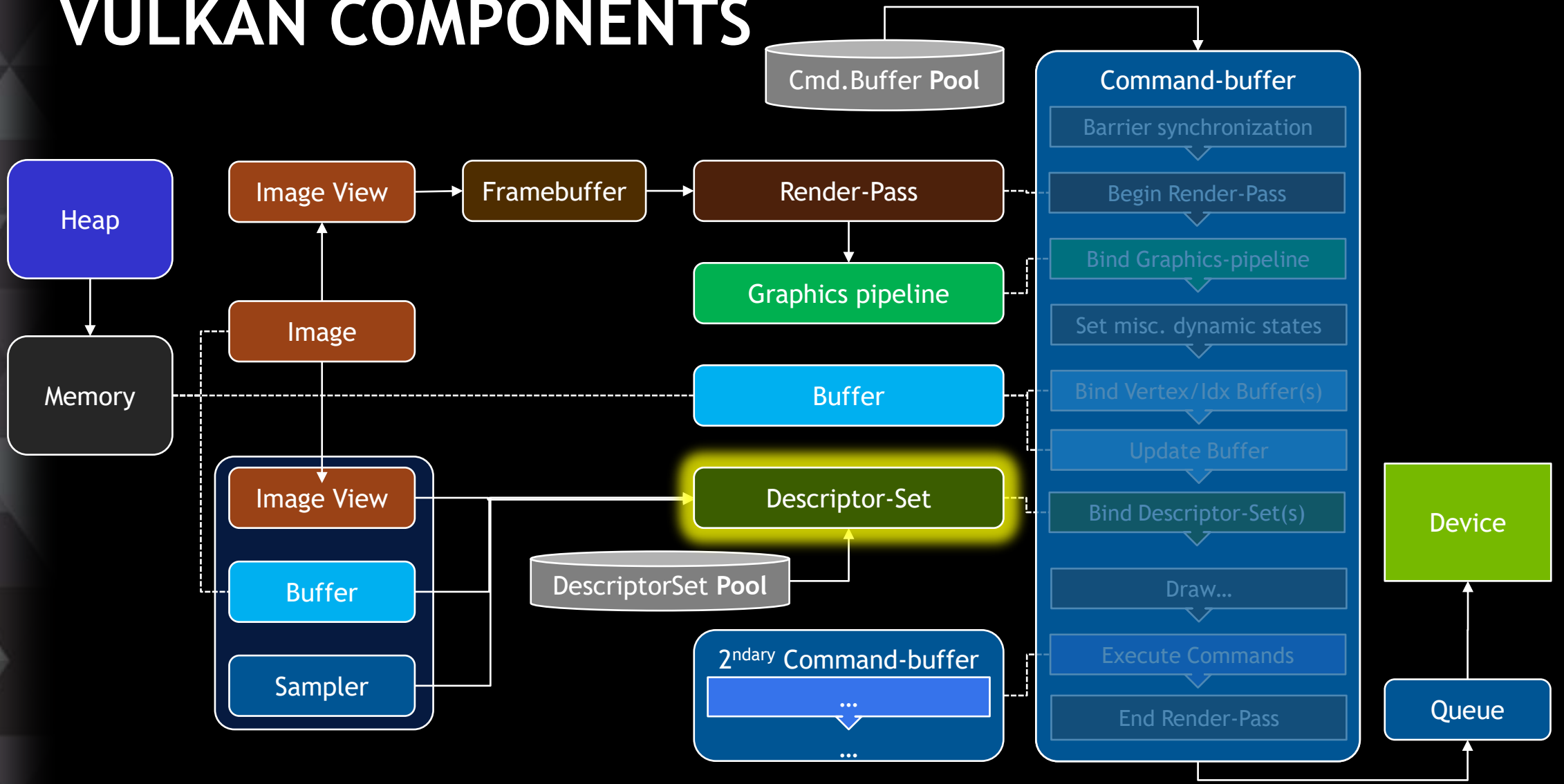
- Create staging buffer + bind memory + copy data in it
- Use command-buffer to copy to the image: layers and mipmaps
  - Layout transition of image for copy
  - vkCmdCopyBufferToImage
  - Layout transition of image for use by shader
- Enqueue command buffer and execute

HOW DOES IT LOOK ?

Simple texture creation



VULKAN COMPONENTS



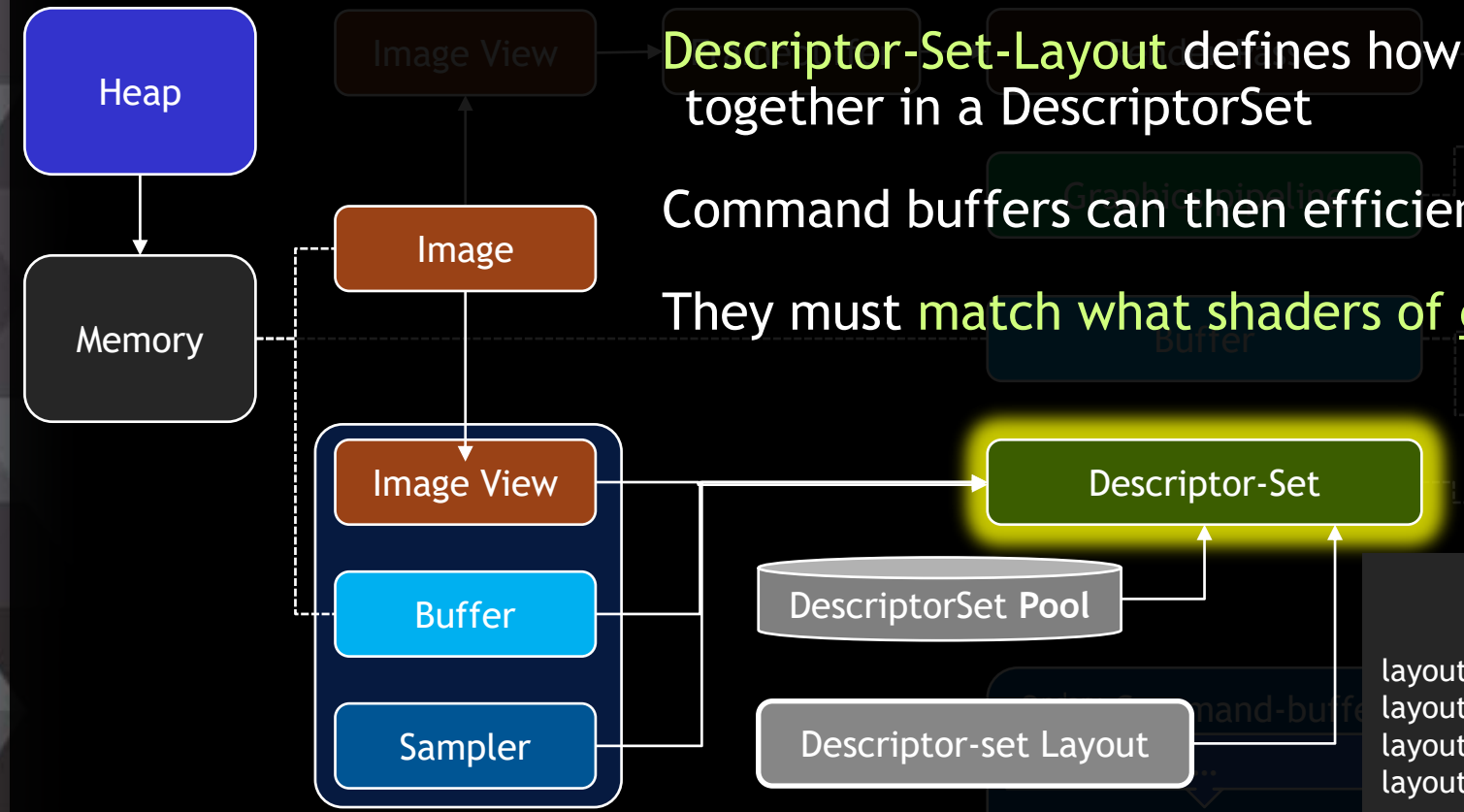
DESCRIPTOR-SET

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

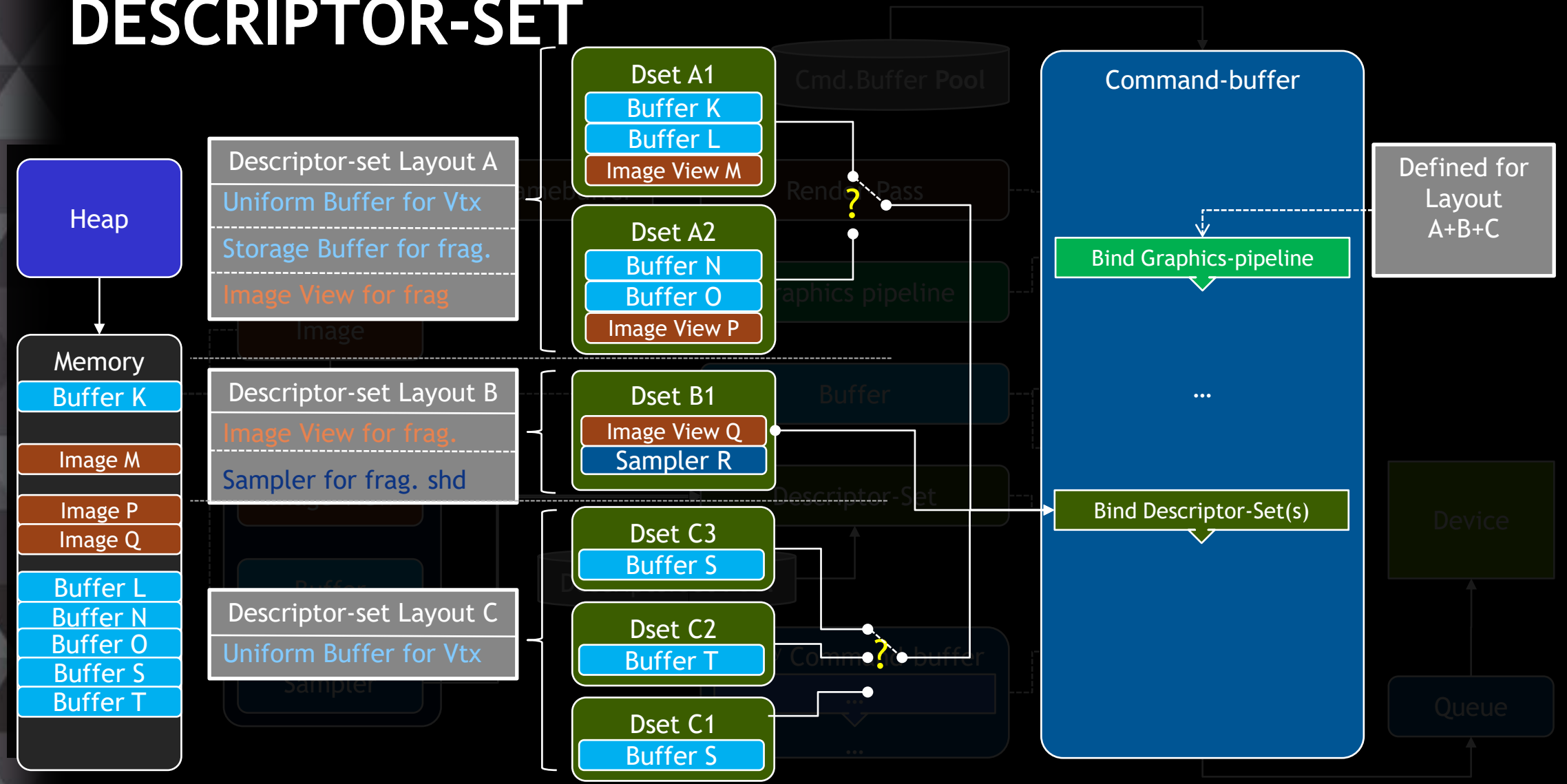
They must match what shaders of each stage expect !



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



HOW DOES IT LOOK ?

Descriptor Set - initialization

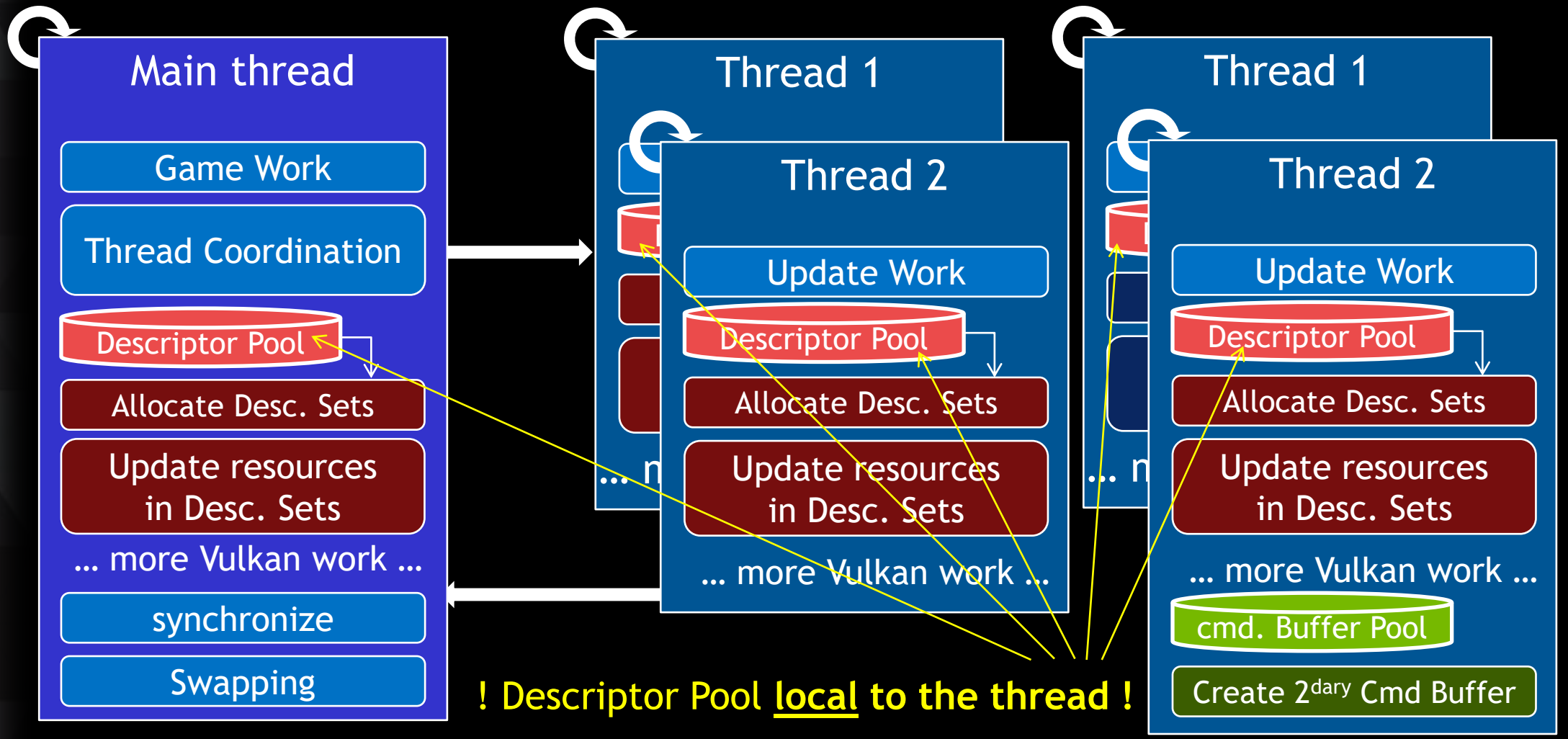
```
m_descriptorSetLayouts[DSET_GLOBAL] = nvk.vkCreateDescriptorSetLayout(
    NVK::VkDescriptorSetLayoutCreateInfo(NVK::VkDescriptorSetLayoutBinding
     (BINDING_MATRIX, VK_DESCRIPTOR_TYPE_UNIFORM_BUFFER, 1, VK_SHADER_STAGE_VERTEX_BIT) // BINDING_MATRIX
     (BINDING_CUBETEX, VK_DESCRIPTOR_TYPE_COMBINED_IMAGE_SAMPLER, 1, VK_SHADER_STAGE_FRAGMENT_BIT)
) );


m_descPool = nvk.vkCreateDescriptorPool(NVK::VkDescriptorPoolCreateInfo(
    10/*maxSets*/, NVK::VkDescriptorPoolSize(VK_DESCRIPTOR_TYPE_UNIFORM_BUFFER, 5)
                                            (VK_DESCRIPTOR_TYPE_UNIFORM_BUFFER_DYNAMIC, 5)
                                            (VK_DESCRIPTOR_TYPE_COMBINED_IMAGE_SAMPLER, 5))
);

nvk.vkAllocateDescriptorSets(NVK::VkDescriptorSetAllocateInfo
    (m_descPool, 1, m_descriptorSetLayouts), &m_descriptorSetGlobal);


NVK::VkDescriptorBufferInfo descBuffer = NVK::VkDescriptorBufferInfo(m_matrix.buffer, 0, m_matrix.Sz);
nvk.vkUpdateDescriptorSets(NVK::VkWriteDescriptorSet
    (m_descriptorSetGlobal, BINDING_MATRIX, 0, descBuffer,                VK_DESCRIPTOR_TYPE_UNIFORM_BUFFER)
    (m_descriptorSetGlobal, BINDING_CUBETEX,  0, cubeTextureSamplerAndView, VK_DESCRIPTOR_TYPE_COMBINED_IMAGE_SAMPLER)
    );
```

HOW DOES IT LOOK ?

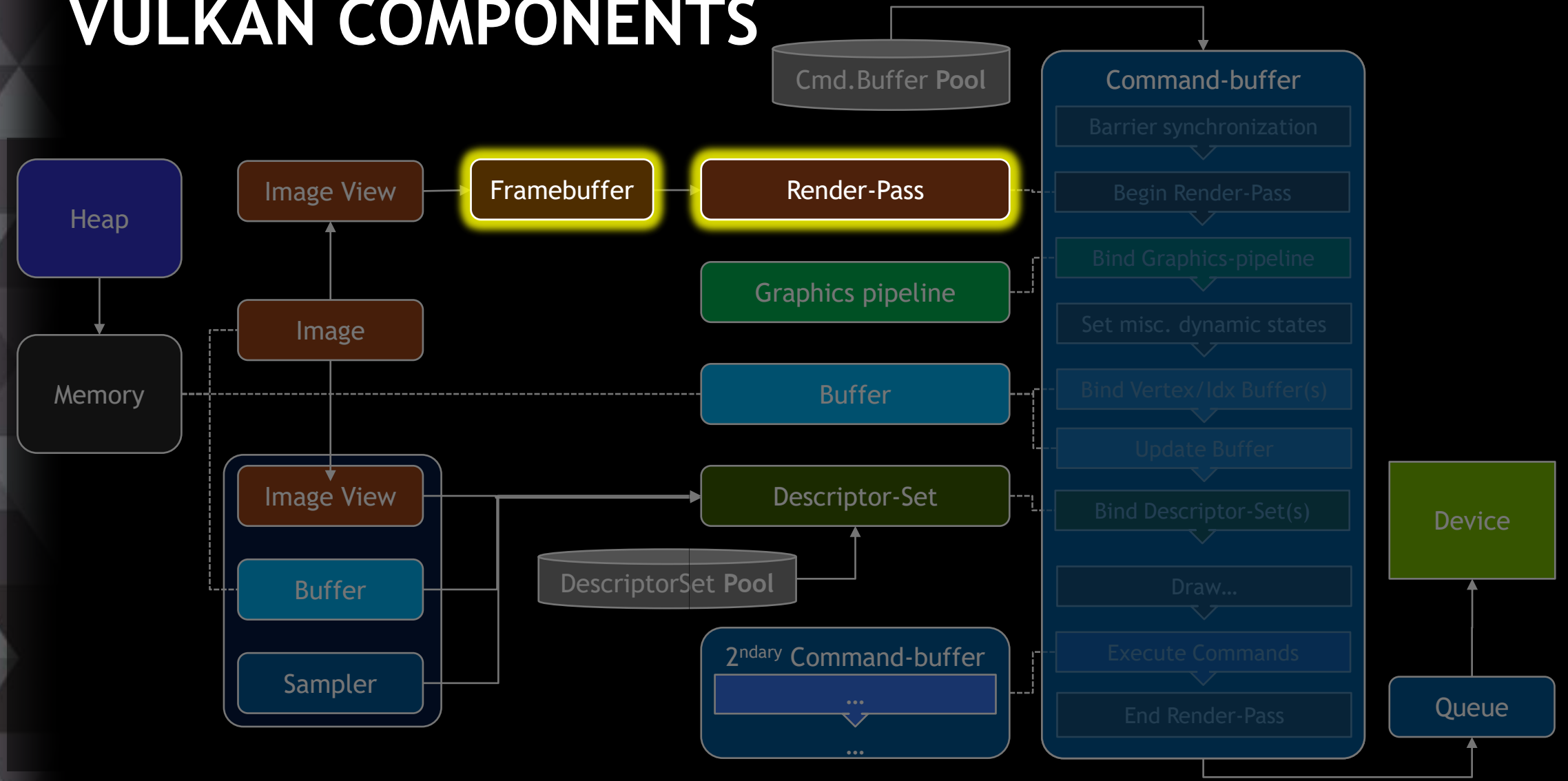
Descriptor Set - using it

```
nvk.vkBeginCommandBuffer(m_cmdBufferGrid, false,
    NVK::VkCommandBufferInheritanceInfo(renderPass, 0, framebuffer,  0/*occlusionQueryEnable*/,
                                                                     0/*queryFlags*/, 0/*pipelineStatistics*/) );
{
    vkCmdBindPipeline(m_cmdBufferGrid, VK_PIPELINE_BIND_POINT_GRAPHICS, m_pipelineGrid);
    vkCmdSetDepthBias(m_cmdBufferGrid, 0.0f, 0.0f, 0.0f); // offset raster
    vkCmdSetLineWidth(m_cmdBufferGrid, lineWidth);
    VkDeviceSize vboffsets[1] = {0};
    vkCmdBindVertexBuffers(m_cmdBufferGrid, 0, 1, &m_gridBuffer.buffer, vboffsets);

    vkCmdBindDescriptorSets(m_cmdBufferGrid, VK_PIPELINE_BIND_POINT_GRAPHICS, m_pipelineLayout,
                            DSET_GLOBAL, 1, &m_descriptorSetGlobal, 0, NULL);

    vkCmdDraw(m_cmdBufferGrid, GRIDDEF * 4, 1, 0, 0);
}
vkEndCommandBuffer(m_cmdBufferGrid);
```

DESCRIPTOR SETS AND MULTI-THREADING

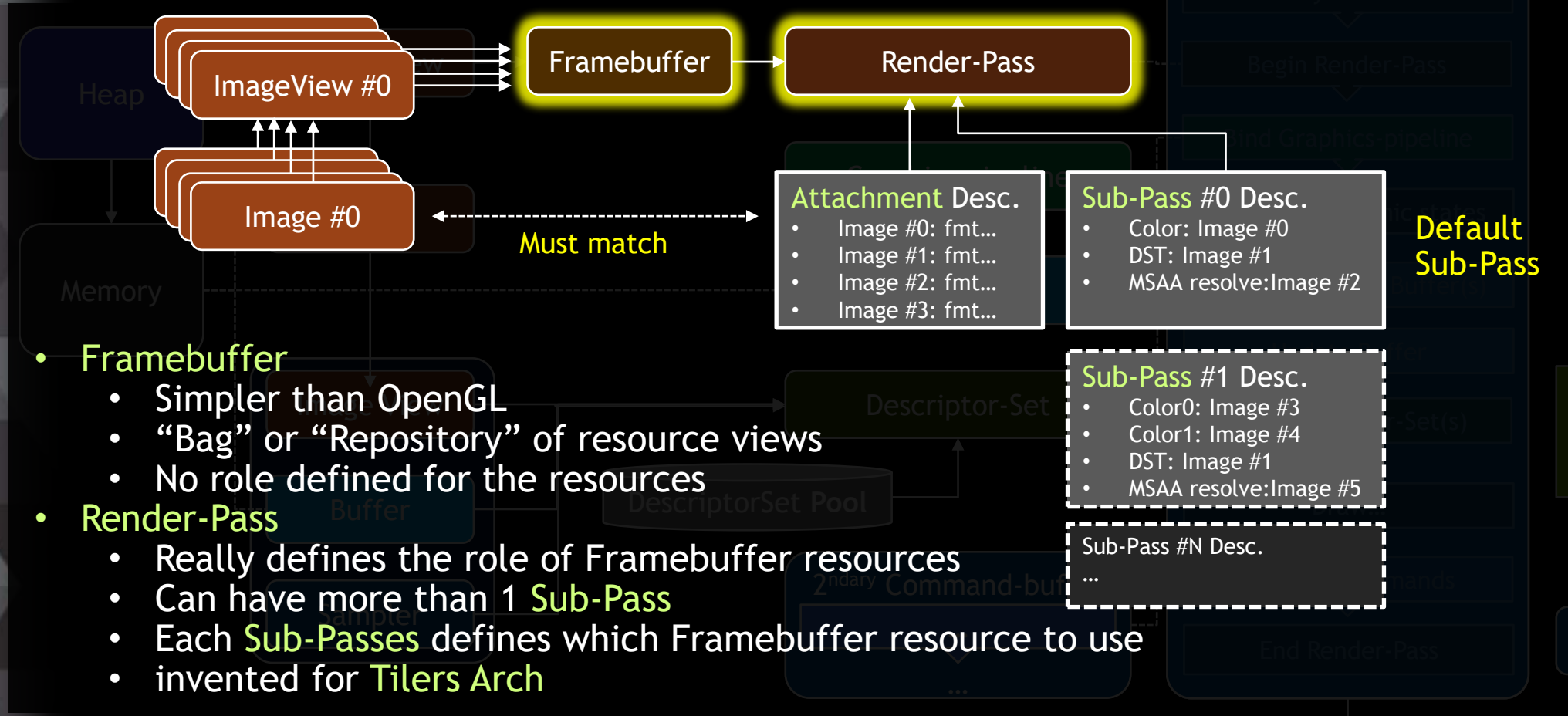


VULKAN COMPONENTS



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

HOW DOES IT LOOK ?

Renderpass Creation (‘compacted’ notation for Vulkan structures)

```
NVK::VkRenderPassCreateInfo rpinfo = NVK::VkRenderPassCreateInfo(
    NVK::VkAttachmentDescription
        (   VK_FORMAT_R8G8B8A8_UNORM, (VkSampleCountFlagBits)MSAA,                          //format, samples
            VK_ATTACHMENT_LOAD_OP_CLEAR, VK_ATTACHMENT_STORE_OP_STORE,                      //loadOp, storeOp
            VK_ATTACHMENT_LOAD_OP_DONT_CARE, VK_ATTACHMENT_STORE_OP_DONT_CARE,              //stencilLoadOp, stencilStoreOp
            VK_IMAGE_LAYOUT_COLOR_ATTACHMENT_OPTIMAL, VK_IMAGE_LAYOUT_COLOR_ATTACHMENT_OPTIMAL //initialLayout, finalLayout
        )
        (   VK_FORMAT_D24_UNORM_S8_UINT, (VkSampleCountFlagBits)MSAA,
            VK_ATTACHMENT_LOAD_OP_CLEAR, VK_ATTACHMENT_STORE_OP_STORE,
            VK_ATTACHMENT_LOAD_OP_CLEAR, VK_ATTACHMENT_STORE_OP_STORE,
            VK_IMAGE_LAYOUT_DEPTH_STENCIL_ATTACHMENT_OPTIMAL, VK_IMAGE_LAYOUT_DEPTH_STENCIL_ATTACHMENT_OPTIMAL
        )
        (   VK_FORMAT_R8G8B8A8_UNORM, (VkSampleCountFlagBits)1,                             //format, samples
            VK_ATTACHMENT_LOAD_OP_DONT_CARE, VK_ATTACHMENT_STORE_OP_DONT_CARE,              //loadOp, storeOp
            VK_ATTACHMENT_LOAD_OP_DONT_CARE, VK_ATTACHMENT_STORE_OP_DONT_CARE,              //stencilLoadOp, stencilStoreOp
            VK_IMAGE_LAYOUT_COLOR_ATTACHMENT_OPTIMAL, VK_IMAGE_LAYOUT_COLOR_ATTACHMENT_OPTIMAL //initialLayout, finalLayout
        ),
    // Easy way
    NVK::VkSubpassDescription
    (   VK_PIPELINE_BIND_POINT_GRAPHICS,                                                    //pipelineBindPoint
        NVK::VkAttachmentReference(),                                                       //inputAttachments
        NVK::VkAttachmentReference(0/*attachment*/, VK_IMAGE_LAYOUT_COLOR_ATTACHMENT_OPTIMAL/*layout*/),       //colorAttachments
        NVK::VkAttachmentReference(2/*attachment*/, VK_IMAGE_LAYOUT_COLOR_ATTACHMENT_OPTIMAL/*layout*/),       //resolveAttachments
        NVK::VkAttachmentReference(1/*attachment*/, VK_IMAGE_LAYOUT_DEPTH_STENCIL_ATTACHMENT_OPTIMAL/*layout*/),//depthStencilAttachment
        NVK::Uint32Array(),                                                                 //preserveAttachments
        0                                                                                   //flags
    ),
    NVK::VkSubpassDependency(/*NONE*/)
);
m_scenePass     = nvk.vkCreateRenderPass(rpinfo);
```

HOW DOES IT LOOK ?

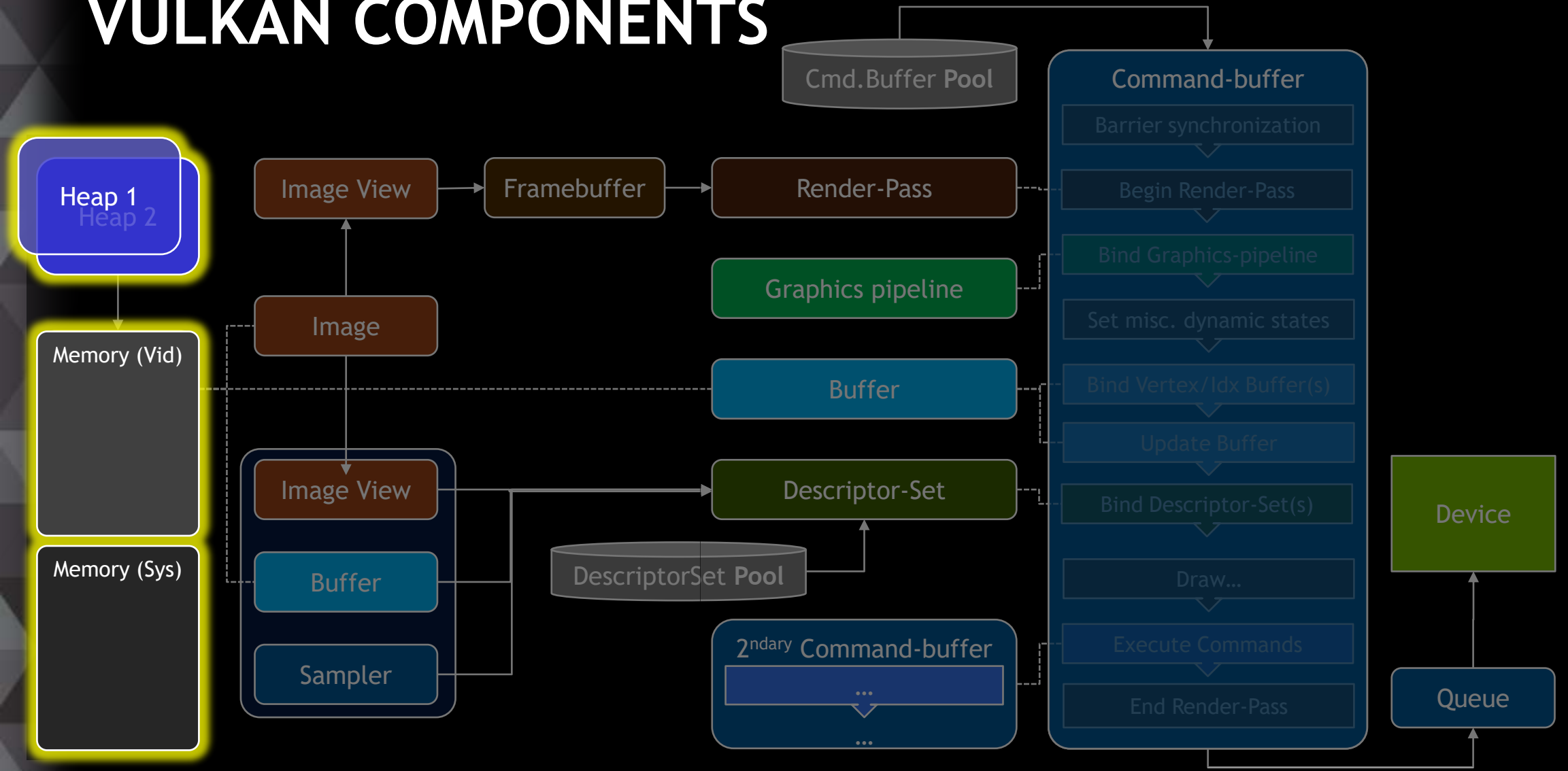
Framebuffer Creation and use

```
m_colorImage[i].img      = nvk.createImage2D(width, height, m_colorImage[i].imgMem, VK_FORMAT_R8G8B8A8_UNORM, VK_SAMP
m_colorImage[i].imgView  = nvk.vkCreateImageView(NVK::VkImageViewCreateInfo(
    m_colorImage[i].img,      /*image*/        VK_IMAGE_VIEW_TYPE_2D, /*viewType*/      VK_FORMAT_R8G8B8A8_UNORM, /*
    NVK::VkComponentMapping(), /*channels*/    NVK::VkImageSubresourceRange() /*subresourceRange*/ ) );
//...
```

```
m_framebuffers[i] = nvk.vkCreateFramebuffer(
    NVK::VkFramebufferCreateInfo
    (   m_scenePass,            //renderPass
        width, height, 1,       //width, height, layers
    (m_colorImageMS[i].imgView) )
    (m_DSTImageMS[i].imgView)
    (m_colorImage[i].imgView)
);
```

```
nvk.vkBeginCommandBuffer(m_cmdScene, false,
    NVK::VkCommandBufferInheritanceInfo(renderPass, 0, framebuffer, VK_FALSE, 0, 0) );
vkCmdBeginRenderPass(m_cmdScene,
    NVK::VkRenderPassBeginInfo(     renderPass, framebuffer, viewRect,
                                    NVK::VkClearValue(NVK::VkClearColorValue(0.0f, 0.0f, 0.0f, 1.0f))
                                    (NVK::VkClearDepthStencilValue(1.0, 0))), VK_SUBPASS_CONTENTS_INLINE );
vkCmdSetViewport( m_cmdScene, 0, 1, NVK::VkViewport(0.0, 0.0, w, h, 0.0f, 1.0f) );
vkCmdSetScissor(  m_cmdScene, 0, 1, NVK::VkRect2D(0.0, 0.0, w, h ) );
```

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)

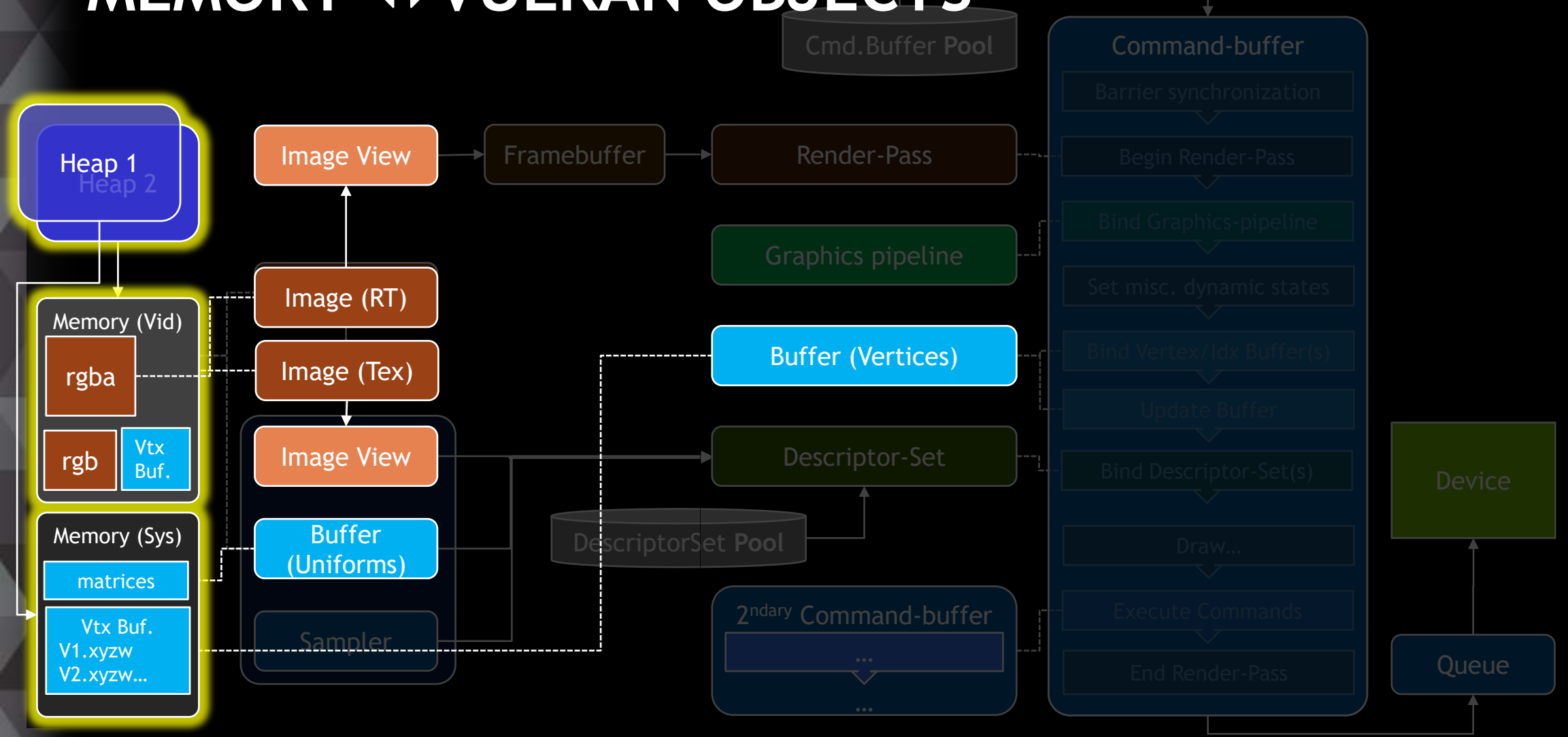
heap 1: size:17,911Mb (System Memory of my PC)

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

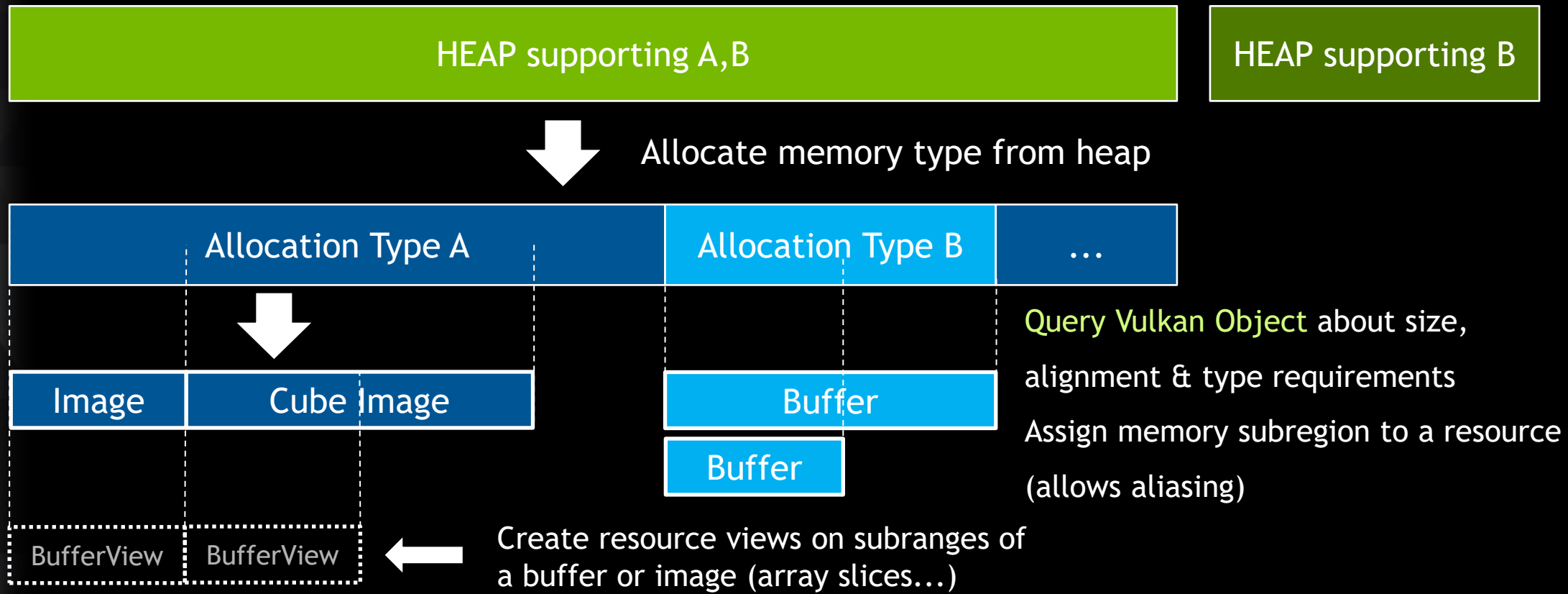
Tegra: Adds one more:
HOST_VISIBLE "NON-Coherent"

MEMORY ⇔VULKAN OBJECTS

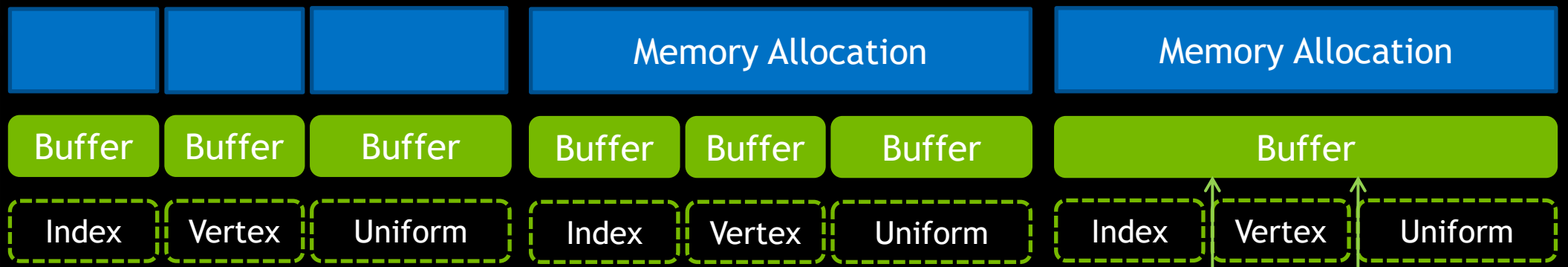


RESOURCE MANAGEMENT

Allocation and Sub allocation



RESOURCE MANAGEMENT



Not. So. Good.

Better...

#HappyGPU

Bind same buffer with Offsets > 0

1 buffer can have many types of data

HOW DOES IT LOOK ?

Binding Memory to Objects – Simple use-case

Mem properties previously gathered at init time

```
vkGetPhysicalDeviceMemoryProperties(physical_devices[j], &m_gpu.memoryProperties);
```

```
typedef struct VkPhysicalDeviceMemoryProperties {
    uint32_t        memoryTypeCount;
    VkMemoryType    memoryTypes[VK_MAX_MEMORY_TYPES];
    uint32_t        memoryHeapCount;
    VkMemoryHeap    memoryHeaps[VK_MAX_MEMORY_HEAPS];
} VkPhysicalDeviceMemoryProperties;
```

```
vkGetBufferMemoryRequirements(m_device, obj, &memReqs);

// Find an available memory type that satifies 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;
}

::VkMemoryAllocateInfo memInfo = { VK_STRUCTURE_TYPE_MEMORY_ALLOCATE_INFO, 0, memReqs.size, memoryTypeIndex };
result = vkAllocateMemory(m_device, &memInfo, NULL, &deviceMem);

result = vkBindBufferMemory(obj, deviceMem, 0);
```

SHADERS

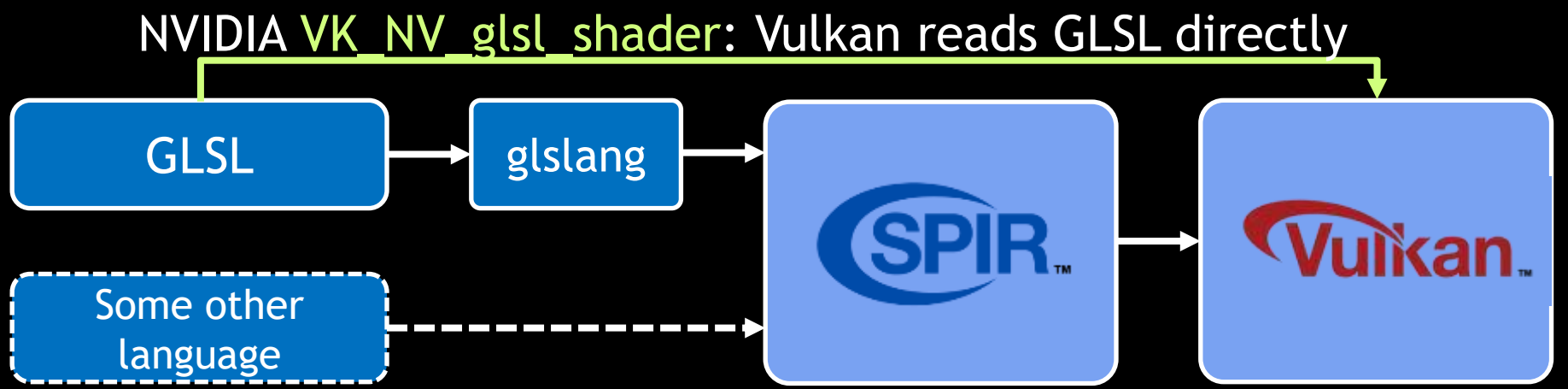
Vulkan uses SPIR-V passed directly to the driver

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

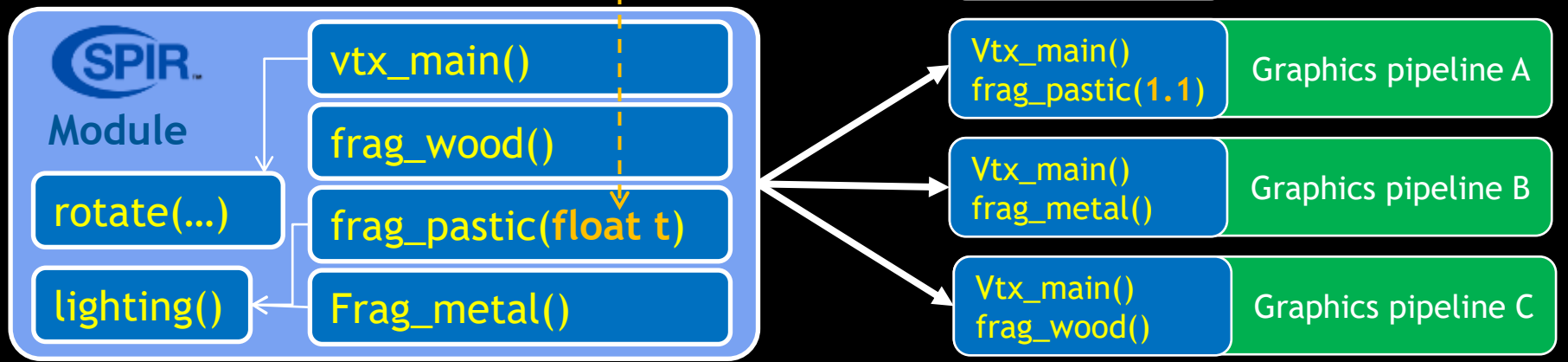
Multiple entry points can be defined in a single Spir-V shader-module

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

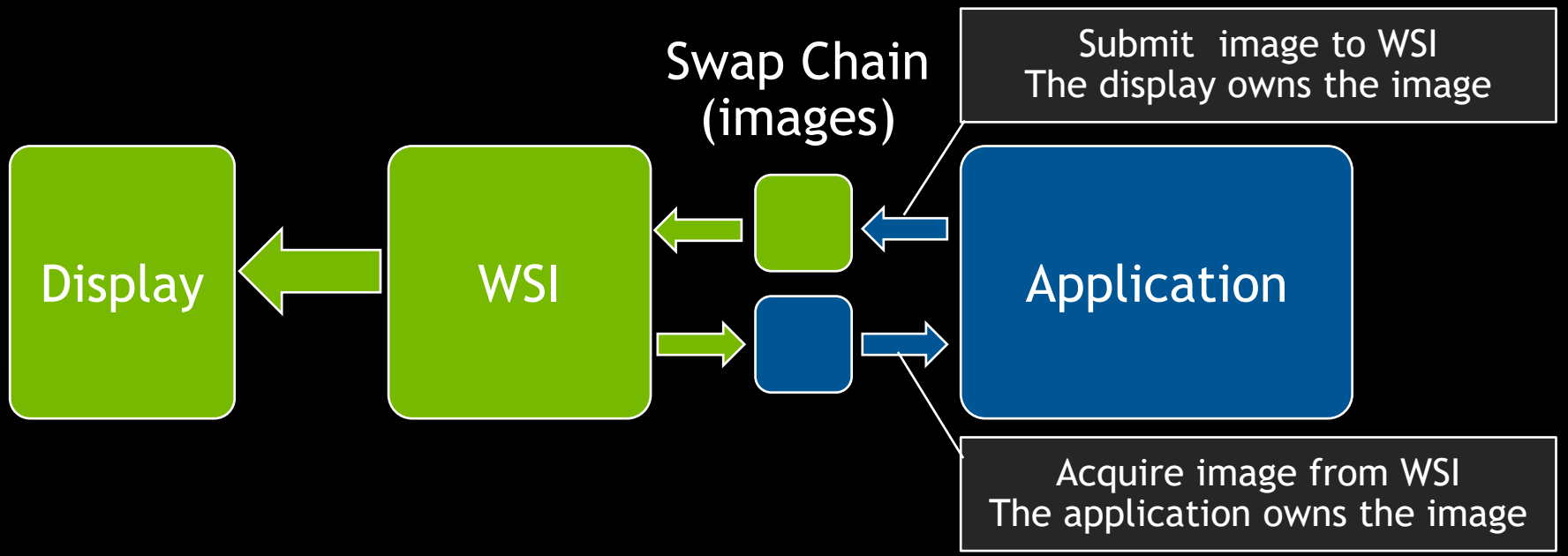


VULKAN WINDOW SYSTEM INTEGRATION (WSI)

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

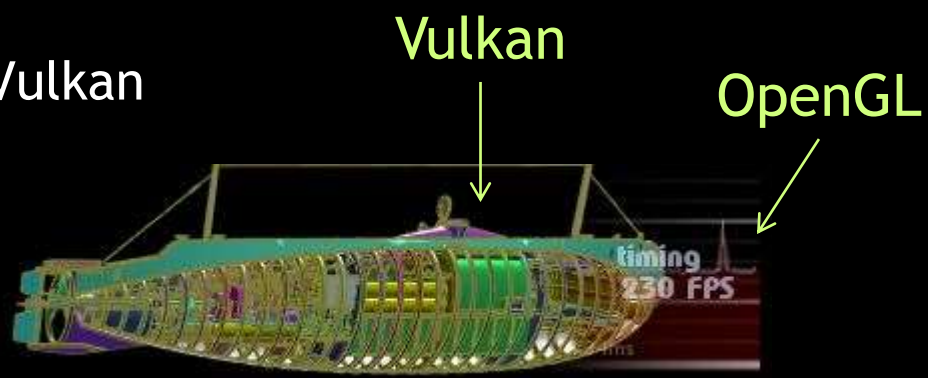
Create Vulkan Device

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



HOW DOES IT LOOK ?

GL_NV_draw_vulkan_image

Initialize…

```
glWaitVkSemaphoreNV = (PFNGLWAITVKSEMAPHORENVPROC)NVPWindow::sysGetProcAddress("glWaitVkSemaphoreNV");
glSignalVkSemaphoreNV = (PFNGLSIGNALVKSEMAPHORENVPROC)NVPWindow::sysGetProcAddress("glSignalVkSemaphoreNV");
glSignalVkFenceNV = (PFNGLSIGNALVKFENCENVPROC)NVPWindow::sysGetProcAddress("glSignalVkFenceNV");
glDrawVkImageNV = (PFNGLDRAWVKIMAGENVPROC)NVPWindow::sysGetProcAddress("glDrawVkImageNV");
if(glDrawVkImageNV == NULL)
{
    LOGE("couldn't find entry points to blit Vulkan to OpenGL back-buffer (glDrawVkImageNV...)");
    nvk.DestroyDevice();
    m_bValid = false;
    return false;
}
VkSemaphoreCreateInfo semCreateInfo = { VK_STRUCTURE_TYPE_SEMAPHORE_CREATE_INFO };
m_semOpenGLReadDone = nvk.vkCreateSemaphore();
m_semVKRenderingDone = nvk.vkCreateSemaphore();
```

HOW DOES IT LOOK ?

GL_NV_draw_vulkan_image

```
nvk.vkQueueSubmit( NVK::VkSubmitInfo(
    1, &m_semOpenGLReadDone, //pWaitSemaphores
    NULL,
    1, &m_cmdScene,          //pCommandBuffers
    1, &m_semVKRenderingDone //pSignalSemaphores
    ),
VK_NULL_HANDLE);
```

Submit commands to Queue with semaphores

Backbuffer Blit

```
void RendererVk::blitToBackbuffer()
{
    float w = m_viewRect.extent.width;
    float h = m_viewRect.extent.height;

    // Wait for the queue of Our VK rendering to signal m_semVKRenderingDone so we know the image is ready
    glWaitVkSemaphoreNV((GLuint64)m_semVKRenderingDone);

    // Blit the image
    glDrawVkImageNV((GLuint64)m_colorImage[m_currentImage].img, 0, 0,0,w,h, 0, 0,1,1,0);

    // Signal m_semOpenGLReadDone to tell the VK rendering queue that it can render the next one
    glSignalVkSemaphoreNV((GLuint64)m_semOpenGLReadDone);
}
```

PRE-REQUISITES TO WORK WITH VULKAN

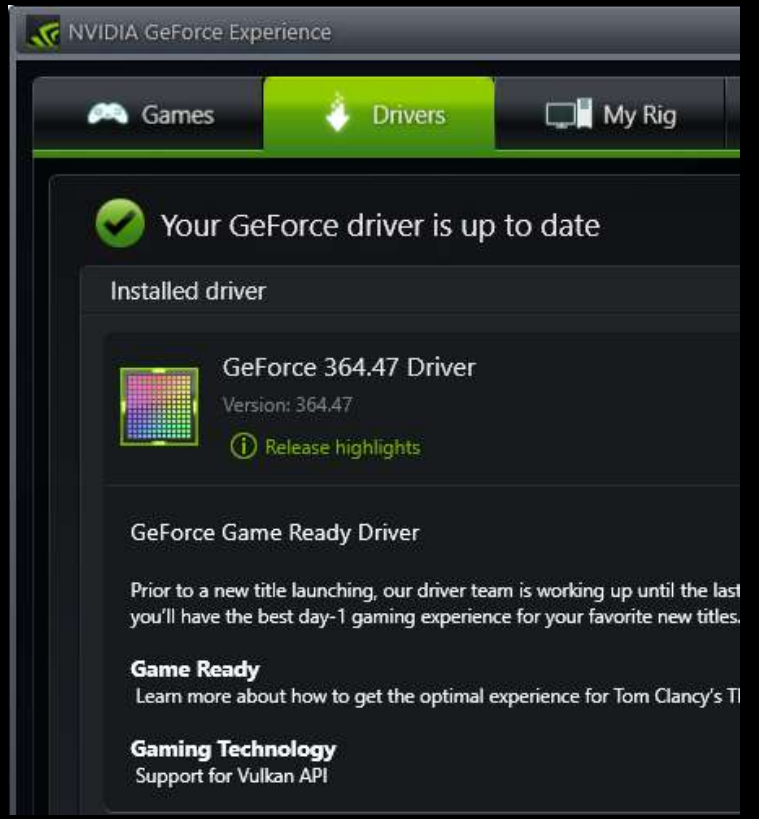
Lunar-G (http://lunarg.com/)

- 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

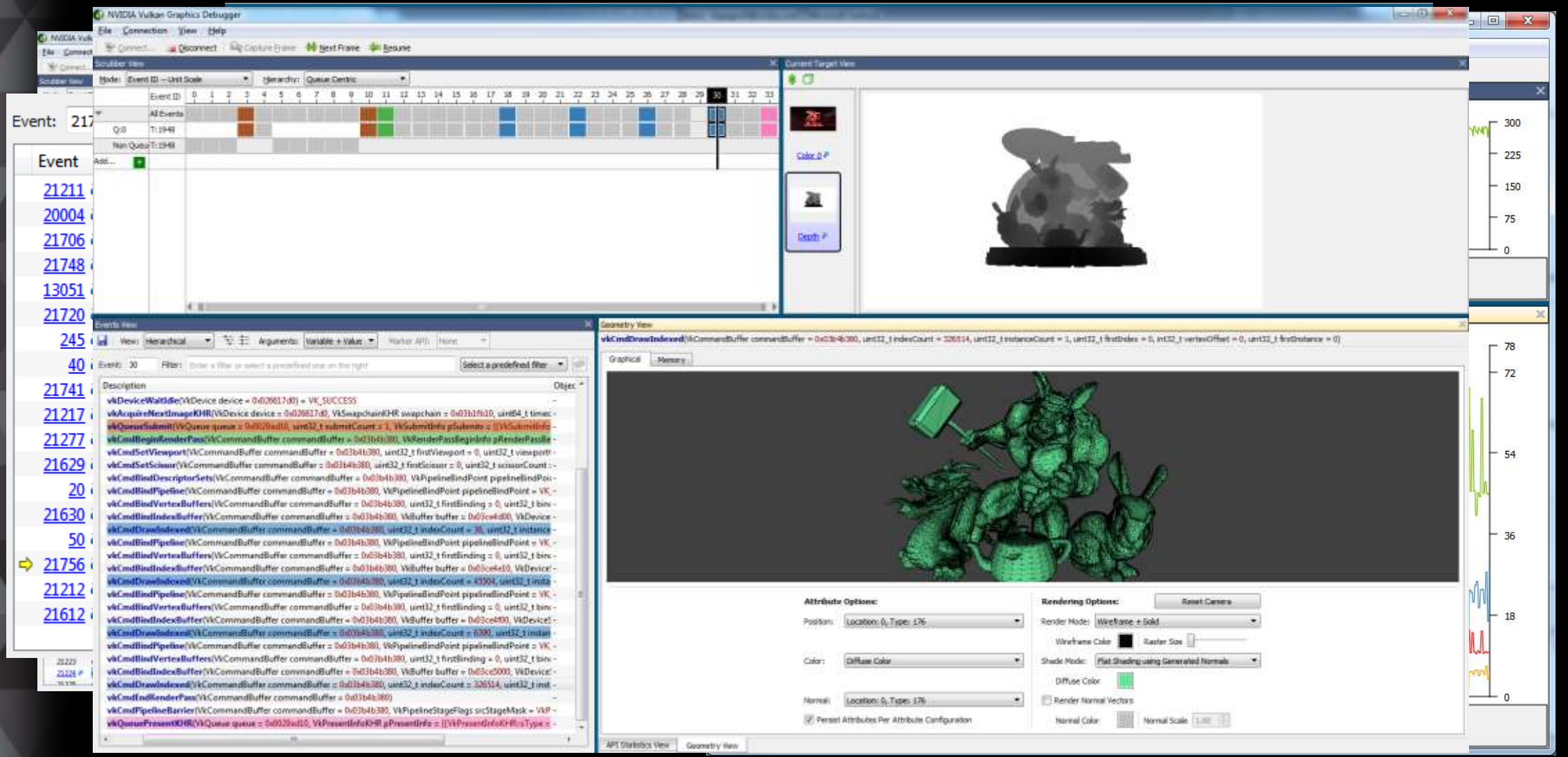
All Host memories are "Coherent" (except one for Tegra)

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

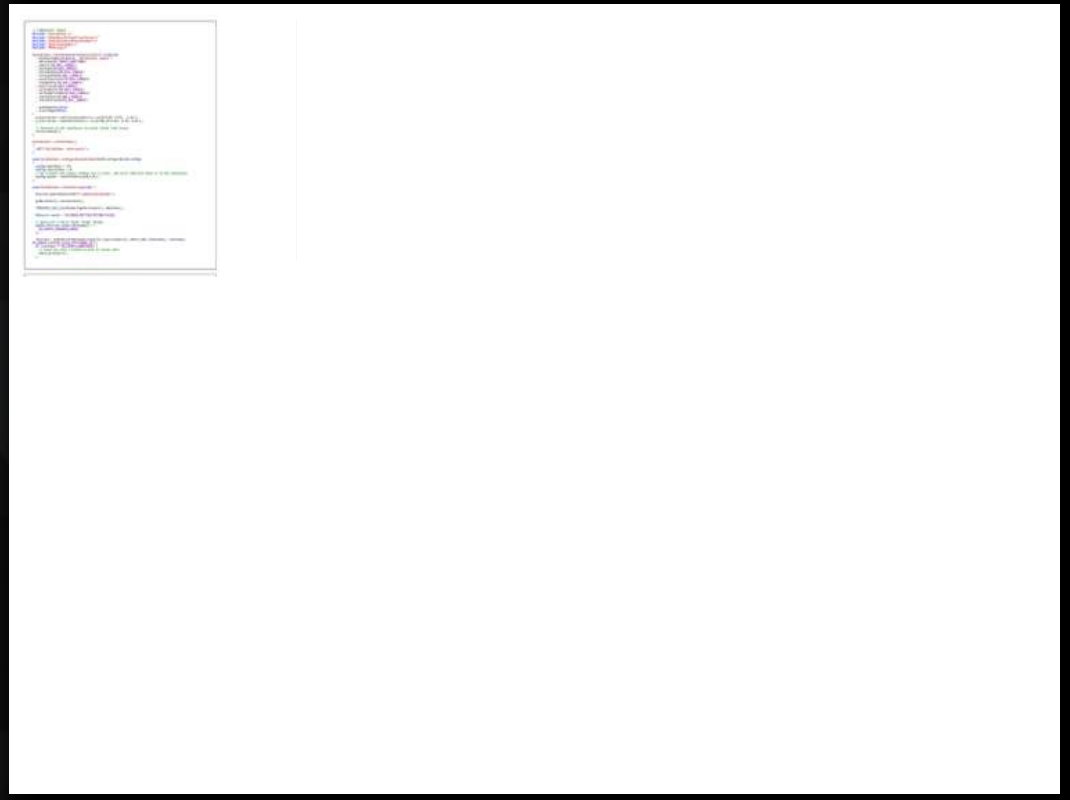
- Type safety
- Syntactic sugar
- Lightweight layer; Keeps you closer to the real Vulkan

Hand-coded ‚high level‘ layer

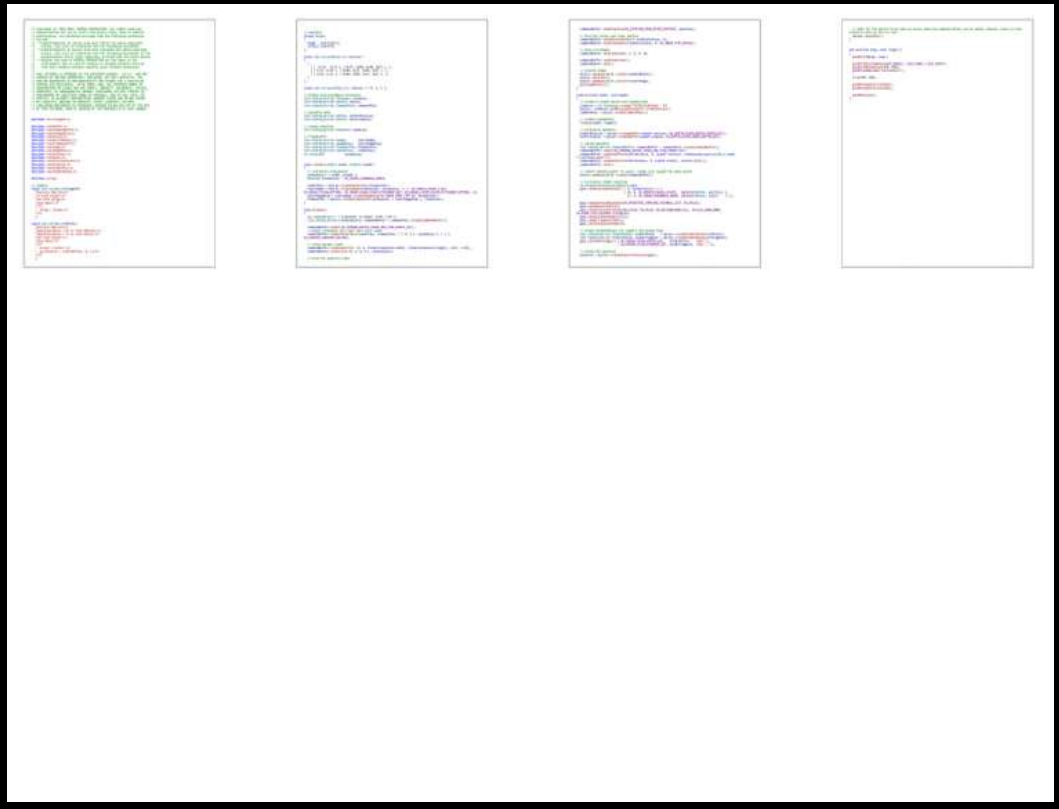
- Reduce code complexity
- Exception safety, resource lifetime tracking, …
- Closer dependency with VkCpp internal implementations

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

PRESENTED BY NVIDIA.