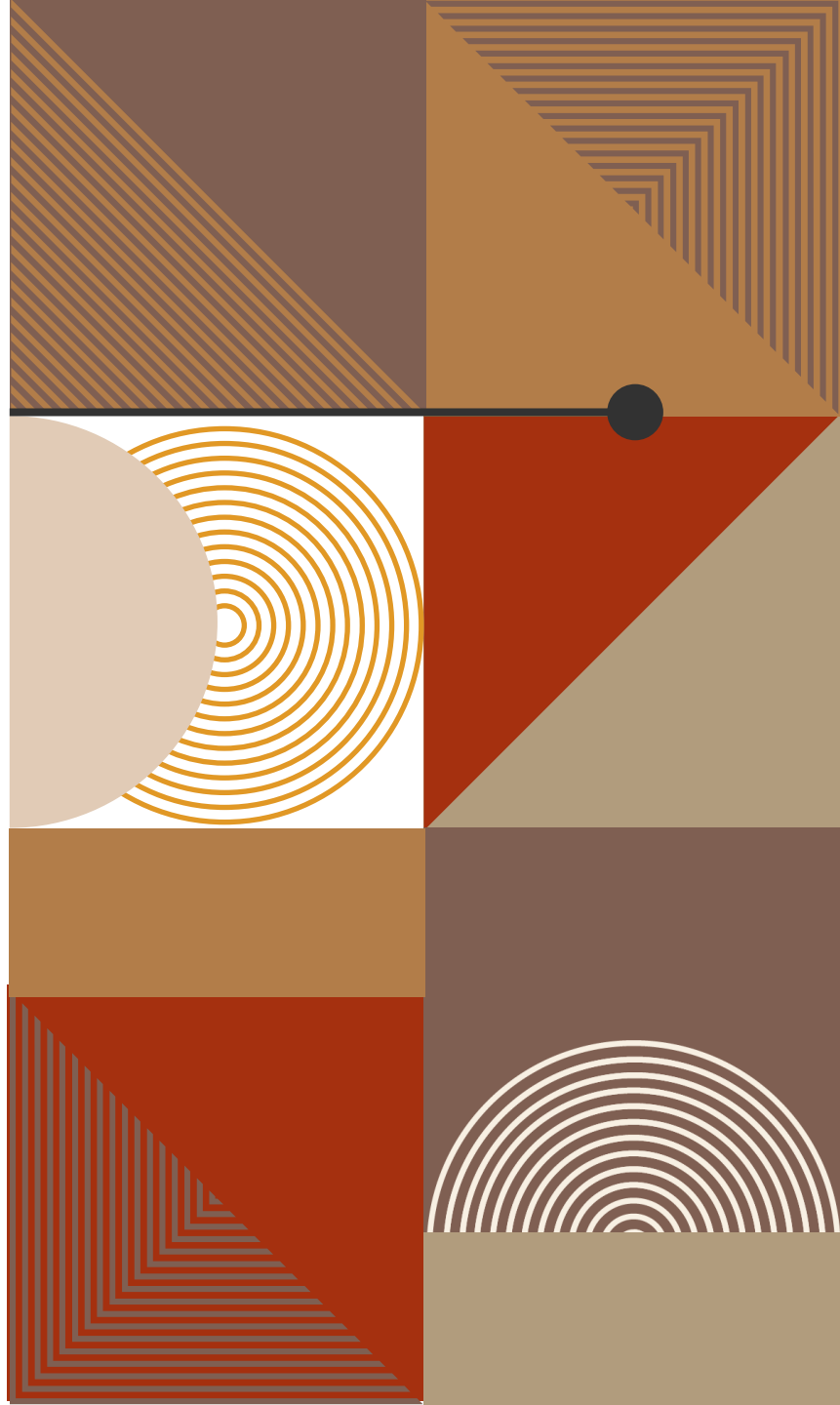
VULKAN

Where making a triangle takes 1000 lines of code

Reem Alghamdi

VULKAN OVERVIEW

- A Low-level, verbose abstraction API of the GPU
- High performance
- Suitable for **general** purpose or **graphics** computation
- Cross platform: windows, linux, android, mac, iOS, switch
- Tradeoff: very **verbose**, but very **efficient**

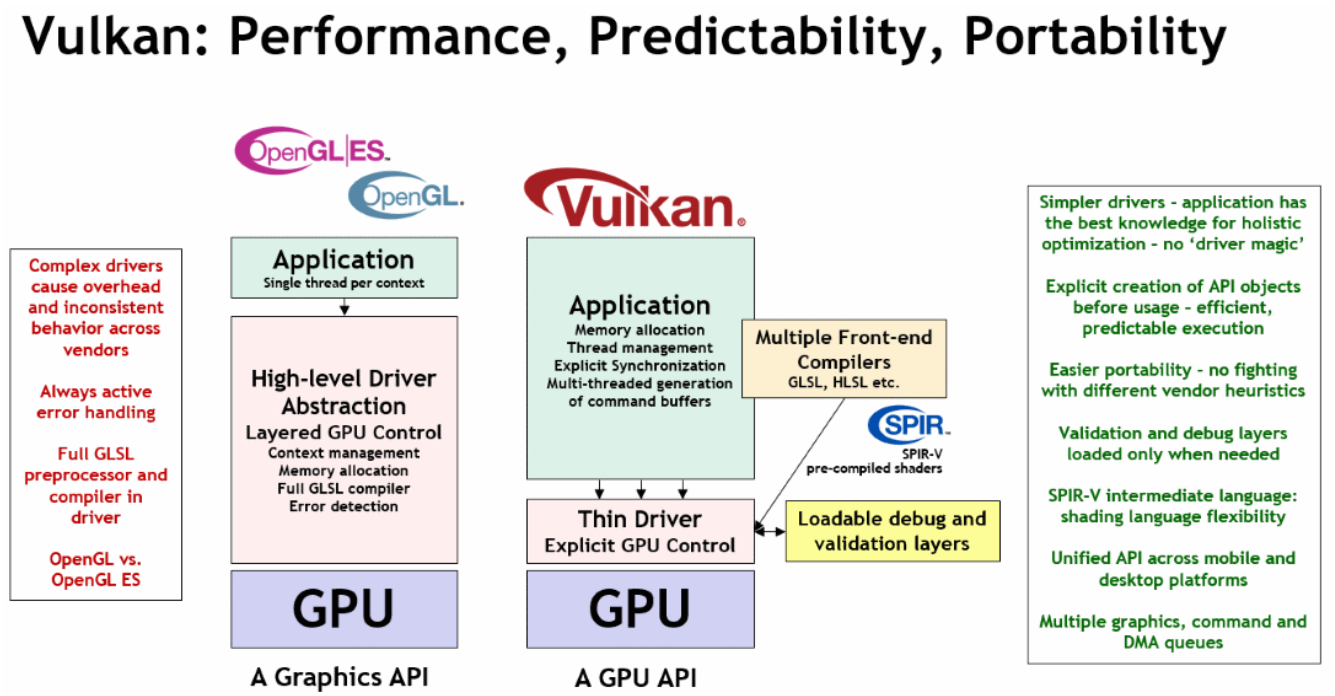


VULKAN IS PLATFORM AGNOSTIC

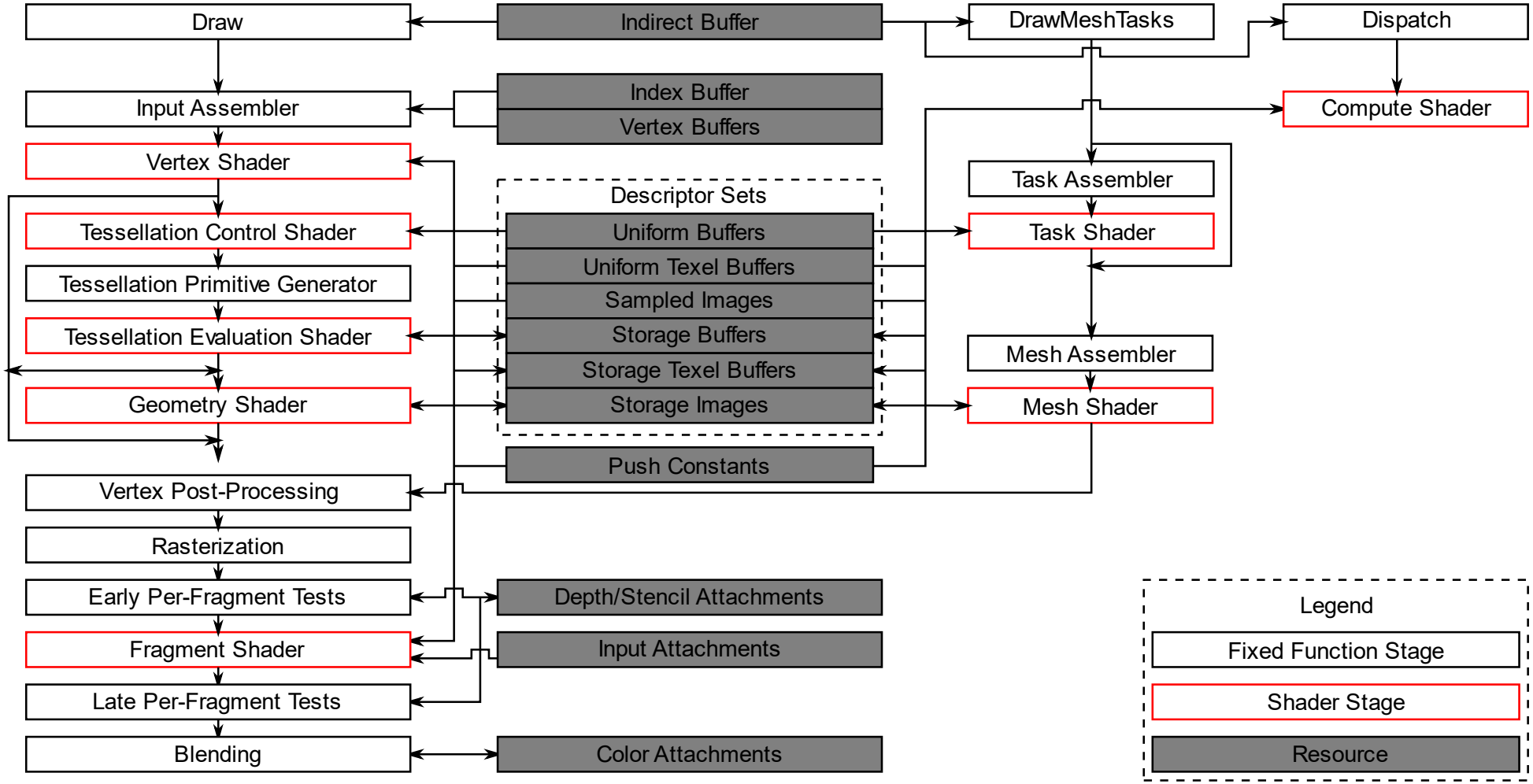
- No windowing system
- Must use extensions to present images
- GLFW

- No default shading language!
- Receive SPIR-V bytecode
- Write in GLSL/HLSL
- Then compile to SPIR-V

THE DRIVER WORK LESS, AT YOUR EXPENSE

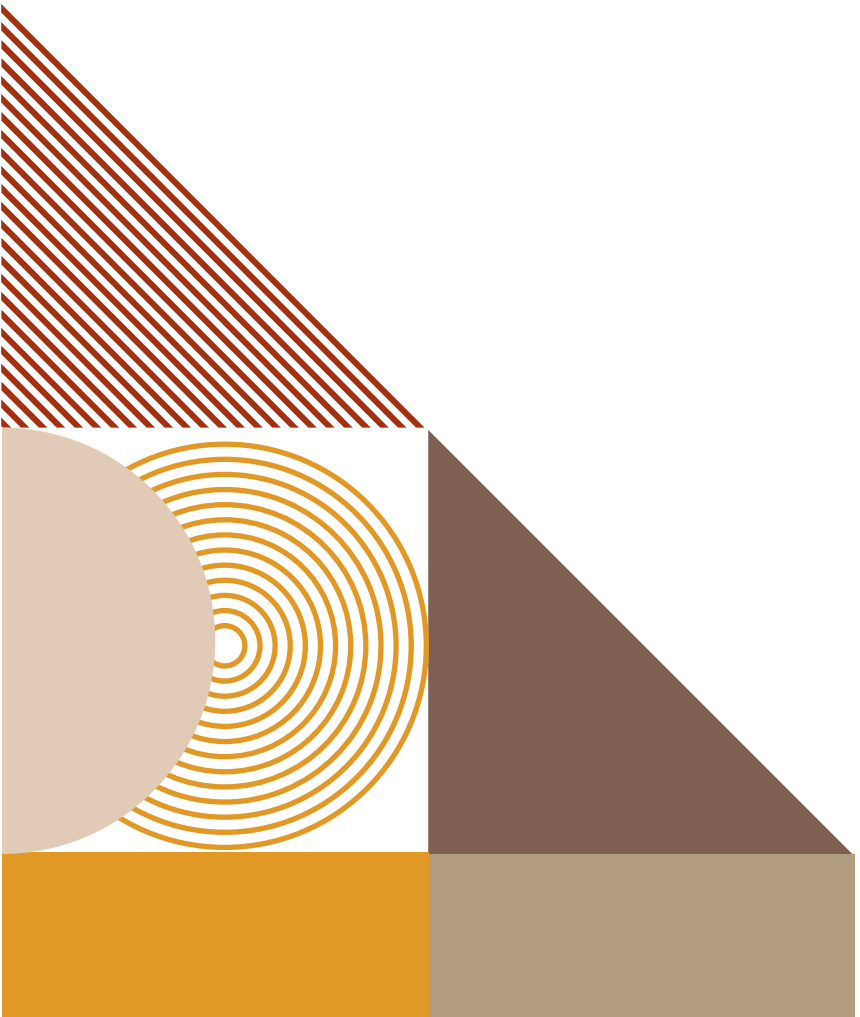


GENERAL PURPOSE AND GRAPHICS



MEMORY TYPES

CUDA	Vulkan
Global Memory	Storage Buffers
Constant Memory	Uniform Buffers
Shared Memory	Shared memory
Texture Memory	Images and Samples
Local Memory	Local Memory



API CONCEPTS

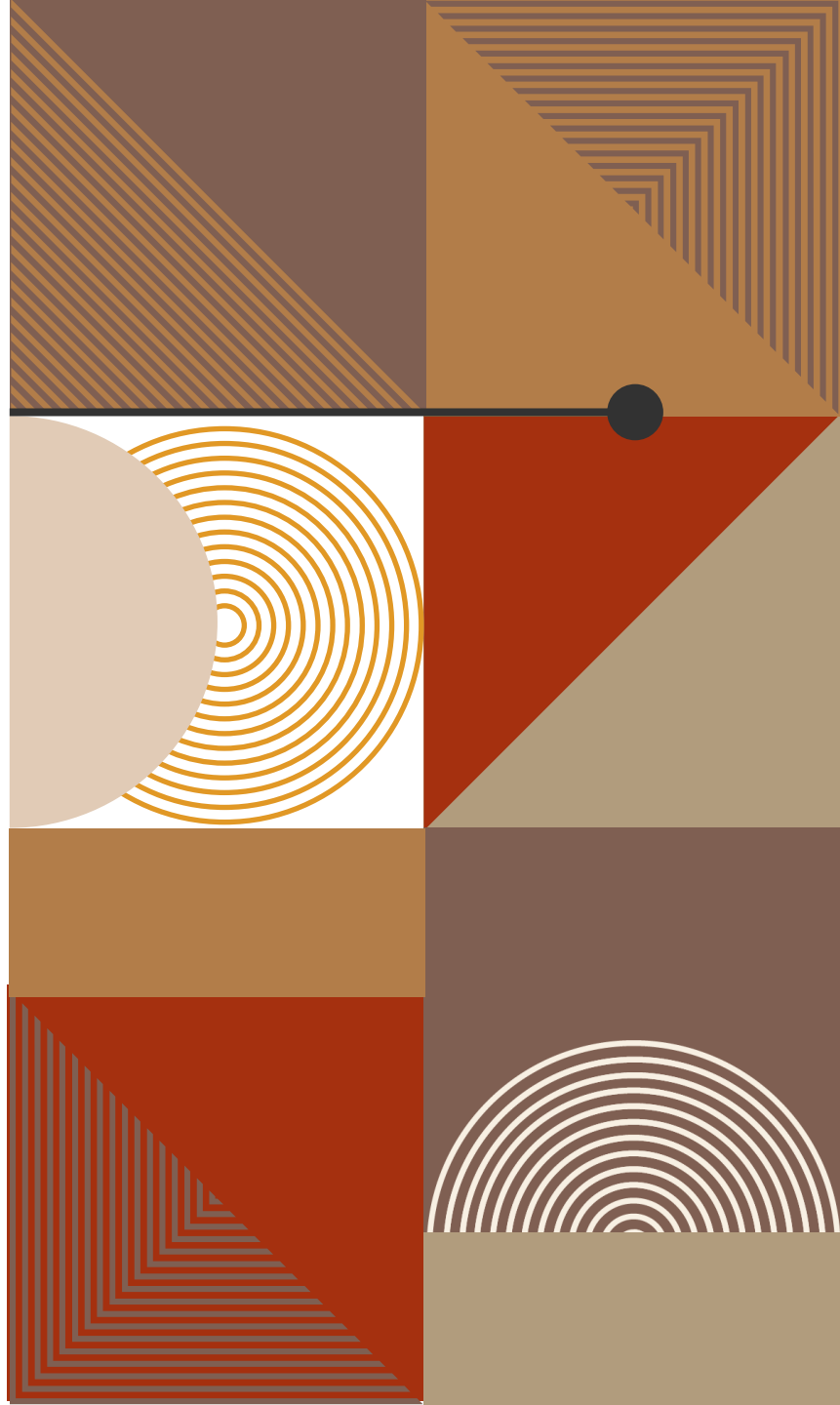
And Best Practices

VALIDATION LAYER

Vulkan philosophy is minimal driver overhead

- Barely any error checks by the API by default

A problem with an API as verbose as Vulkan!



VALIDATION LAYER

Validation layer: optional component hooked into Vulkan calls to perform additional operations:

- Checking the values of parameters against the specification to detect misuse
- Tracking creation and destruction of objects to find resource leaks
- Checking thread safety by tracking the threads that calls originate from
- Logging every call and its parameters to the standard output
- Tracing Vulkan calls for profiling and replaying

STRUCTS OF INFORMATION

```
// fill out struct to specify settings
VkXXCreateInfo createInfo{};
createInfo.sType = VK_XX_CREATE_INFO;
createInfo.pNext = nullptr;
createInfo.foo = 0;
createInfo.bar = ...;
createInfo.pAnotherStruct = &anotherStruct;

// create pointer
if (vkCreateXX(&createInfo, nullptr, &xx) != VK_SUCCESS) {
        throw std::runtime_error("failed to create XX!");
}
// ......
// delete pointer later
vkDestroyXX(xx, nullptr);
```

STRUCTS OF INFORMATION

```
// fill out struct to specify settings
VkXXCreateInfo createInfo{};
createInfo.sType = VK_XX_CREATE_INFO;
createInfo.pNext = nullptr;
createInfo.foo = 0;
createInfo.bar = ...;
createInfo.pAnotherStruct = &anotherStruct;
```

Initialize all variables and structs

```
// create pointer
if (vkCreateXX(&createInfo, nullptr, &xx) != VK_SUCCESS) {
        throw std::runtime_error("failed to create XX!");
}
// ......
// delete pointer later
vkDestroyXX(xx, nullptr);
```

STRUCTS OF INFORMATION

```
// fill out struct to specify settings
VkXXCreateInfo createInfo{};
createInfo.sType = VK_XX_CREATE_INFO;
createInfo.pNext = nullptr;
createInfo.foo = 0;
createInfo.bar = ...;
createInfo.pAnotherStruct = &anotherStruct;
```

There are no default values in Vulkan!

```
// create pointer
if (vkCreateXX(&createInfo, nullptr, &xx) != VK_SUCCESS) {
        throw std::runtime_error("failed to create XX!");
}
// ......
// delete pointer later
vkDestroyXX(xx, nullptr);
```

STRUCTS OF INFORMATION

```
// fill out struct to specify settings
VkXXCreateInfo createInfo{};
createInfo.sType = VK_XX_CREATE_INFO;
createInfo.pNext = nullptr;
createInfo.foo = 0;
createInfo.bar = ...;
createInfo.pAnotherStruct = &anotherStruct;

// create pointer
if (vkCreateXX(&createInfo, nullptr, &xx) != VK_SUCCESS) {
        throw std::runtime_error("failed to create XX!");
}
// ......
// delete pointer later
vkDestroyXX(xx, nullptr);
```

Check function return values

STRUCTS OF INFORMATION

```
// fill out struct to specify settings
VkXXCreateInfo createInfo{};
createInfo.sType = VK_XX_CREATE_INFO;
createInfo.pNext = nullptr;
createInfo.foo = 0;
createInfo.bar = ...;
createInfo.pAnotherStruct = &anotherStruct;

// create pointer
if (vkCreateXX(&createInfo, nullptr, &xx) != VK_SUCCESS) {
        throw std::runtime_error("failed to create XX!");
}
// ......
// delete pointer later
vkDestroyXX(xx, nullptr);
```

Call cleanup functions where appropriate

QUERY AND ENUMERATION

```
// get number of elements
uint32_t count;
vkEnumerateXXs(&count, nullptr);
// fill out array
std::vector<VkXXs> XXs(count);
vkEnumerateXXs(&count, XXs.data());
```

STAGING BUFFERS

```
VkDeviceSize bufferSize = myDataSize;
// create staging buffer
VkBuffer stagingBuffer;
VkDeviceMemory stagingBufferMemory;
createBuffer(bufferSize, stagingBuffer, stagingBufferMemory, stagingFlags);
// create our buffer
createBuffer(bufferSize, myDataBuffer, myDataBufferMemory, myBufferFlags);

// copy data from CPU to GPU (staging buffer)
void* data;
vkMapMemory(device, stagingBufferMemory, 0, bufferSize, 0, &data);
memcpy(data, vertices.data(), (size_t) bufferSize);
vkUnmapMemory(device, stagingBufferMemory);

// copy data from GPU host visible to GPU local (our buffer)
copyBetweenBuffersCommand(stagingBuffer, myDataBuffer, bufferSize);

// clean up staging buffer
vkDestroyBuffer(device, stagingBuffer, nullptr);
vkFreeMemory(device, stagingBufferMemory, nullptr);
```

STAGING BUFFERS

```
VkDeviceSize bufferSize = myDataSize;
// create staging buffer
VkBuffer stagingBuffer;
VkDeviceMemory stagingBufferMemory;
createBuffer(bufferSize, stagingBuffer, stagingBufferMemory, stagingFlags);
// create our buffer
createBuffer(bufferSize, myDataBuffer, myDataBufferMemory, myBufferFlags);
```

Create buffers. Set flags appropriately for the wanted memory type

```
// copy data from CPU to GPU (staging buffer)
void* data;
vkMapMemory(device, stagingBufferMemory, 0, bufferSize, 0, &data);
memcpy(data, vertices.data(), (size_t) bufferSize);
vkUnmapMemory(device, stagingBufferMemory);

// copy data from GPU host visible to GPU local (our buffer)
copyBetweenBuffersCommand(stagingBuffer, myDataBuffer, bufferSize);

// clean up staging buffer
vkDestroyBuffer(device, stagingBuffer, nullptr);
vkFreeMemory(device, stagingBufferMemory, nullptr);
```

STAGING BUFFERS

```
VkDeviceSize bufferSize = myDataSize;
// create staging buffer
VkBuffer stagingBuffer;
VkDeviceMemory stagingBufferMemory;
createBuffer(bufferSize, stagingBuffer, stagingBufferMemory, stagingFlags);
// create our buffer
createBuffer(bufferSize, myDataBuffer, myDataBufferMemory, myBufferFlags);

// copy data from CPU to GPU (staging buffer)
void* data;
vkMapMemory(device, stagingBufferMemory, 0, bufferSize, 0, &data);
memcpy(data, vertices.data(), (size_t) bufferSize);
vkUnmapMemory(device, stagingBufferMemory);

// copy data from GPU host visible to GPU local (our buffer)
copyBetweenBuffersCommand(stagingBuffer, myDataBuffer, bufferSize);

// clean up staging buffer
vkDestroyBuffer(device, stagingBuffer, nullptr);
vkFreeMemory(device, stagingBufferMemory, nullptr);
```

Copy from CPU to GPU in a temporary staging buffer

STAGING BUFFERS

```
VkDeviceSize bufferSize = myDataSize;
// create staging buffer
VkBuffer stagingBuffer;
VkDeviceMemory stagingBufferMemory;
createBuffer(bufferSize, stagingBuffer, stagingBufferMemory, stagingFlags);
// create our buffer
createBuffer(bufferSize, myDataBuffer, myDataBufferMemory, myBufferFlags);

// copy data from CPU to GPU (staging buffer)
void* data;
vkMapMemory(device, stagingBufferMemory, 0, bufferSize, 0, &data);
memcpy(data, vertices.data(), (size_t) bufferSize);
vkUnmapMemory(device, stagingBufferMemory);

// copy data from GPU host visible to GPU local (our buffer)
copyBetweenBuffersCommand(stagingBuffer, myDataBuffer, bufferSize);

// clean up staging buffer
vkDestroyBuffer(device, stagingBuffer, nullptr);
vkFreeMemory(device, stagingBufferMemory, nullptr);
```

GPU command to copy from temporary buffer to our buffer

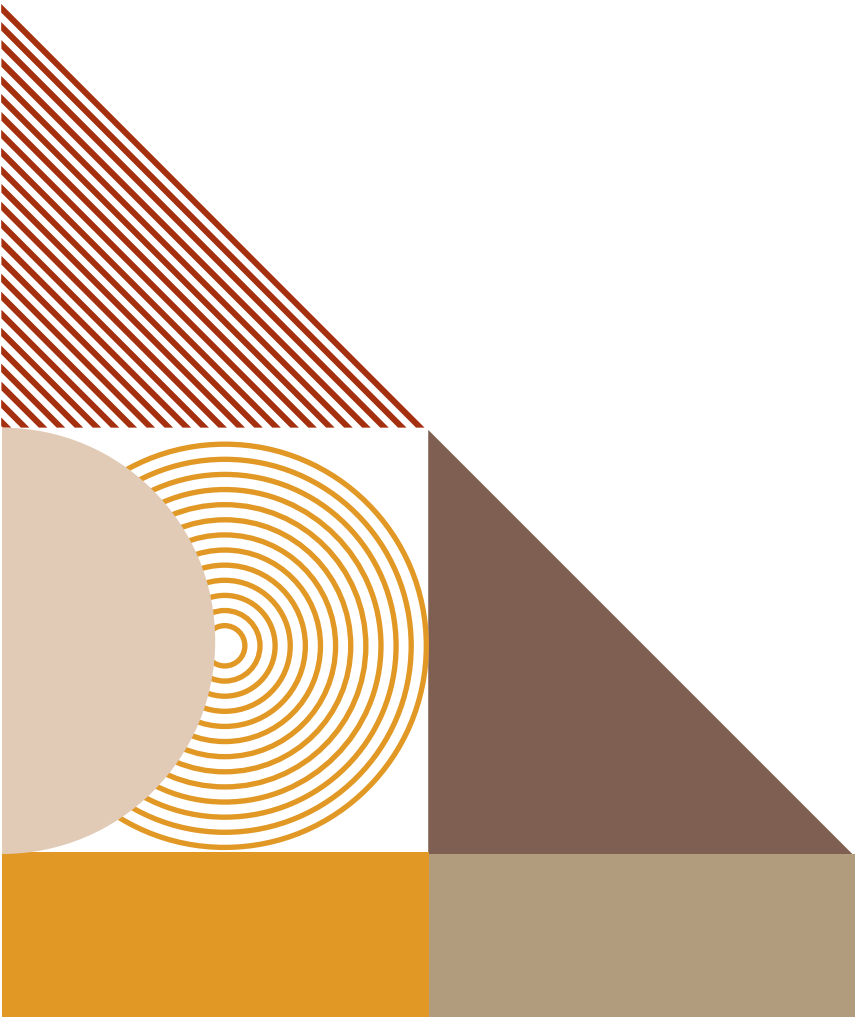
STAGING BUFFERS

```
VkDeviceSize bufferSize = myDataSize;
// create staging buffer
VkBuffer stagingBuffer;
VkDeviceMemory stagingBufferMemory;
createBuffer(bufferSize, stagingBuffer, stagingBufferMemory, stagingFlags);
// create our buffer
createBuffer(bufferSize, myDataBuffer, myDataBufferMemory, myBufferFlags);

// copy data from CPU to GPU (staging buffer)
void* data;
vkMapMemory(device, stagingBufferMemory, 0, bufferSize, 0, &data);
memcpy(data, vertices.data(), (size_t) bufferSize);
vkUnmapMemory(device, stagingBufferMemory);

// copy data from GPU host visible to GPU local (our buffer)
copyBetweenBuffersCommand(stagingBuffer, myDataBuffer, bufferSize);

// clean up staging buffer
vkDestroyBuffer(device, stagingBuffer, nullptr);
vkFreeMemory(device, stagingBufferMemory, nullptr);
```

Cleanup unneeded resources

SYNCHRONIZATION

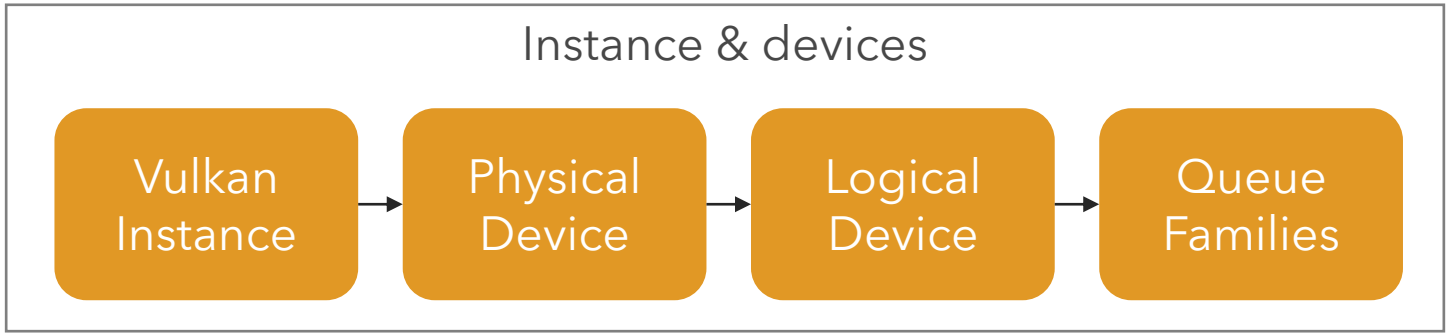
- Vulkan does not manage synchronization.
  - You must explicitly synchronize resources, commands, ..etc!
- Various synchronization mechanisms at different control levels
  - Fences: sync between CPU and GPU
  - Semaphores: GPU sync between command queues
  - Barriers: GPU sync within the command buffer
  - waitIdle: wait until (device, queue) is free

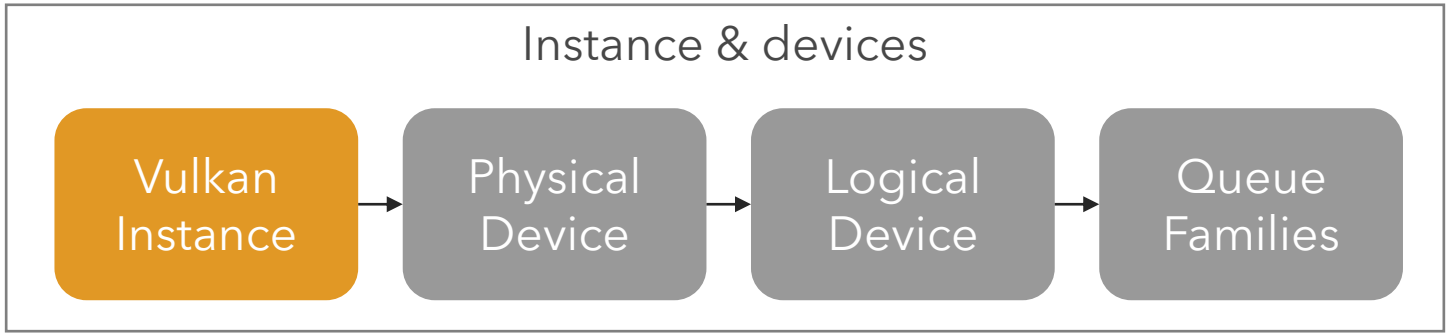


API OVERVIEW

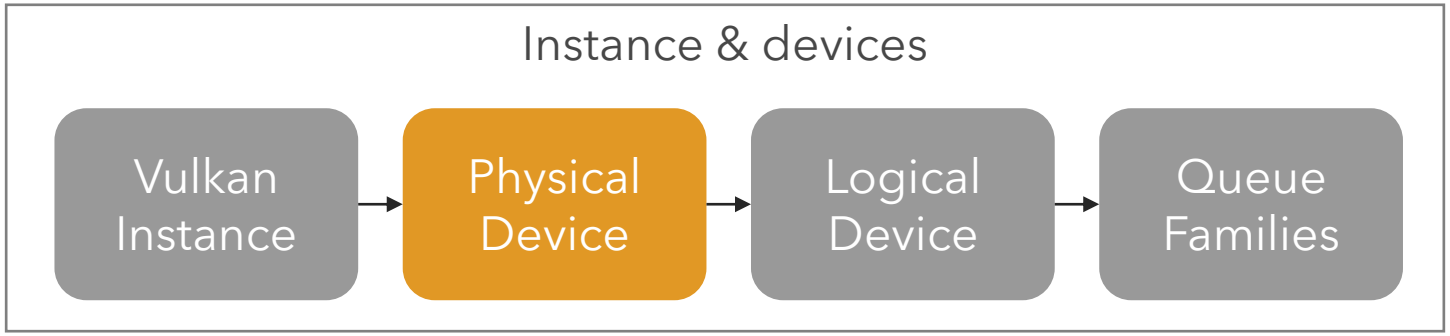
Let’s draw a triangle

Instance & devices

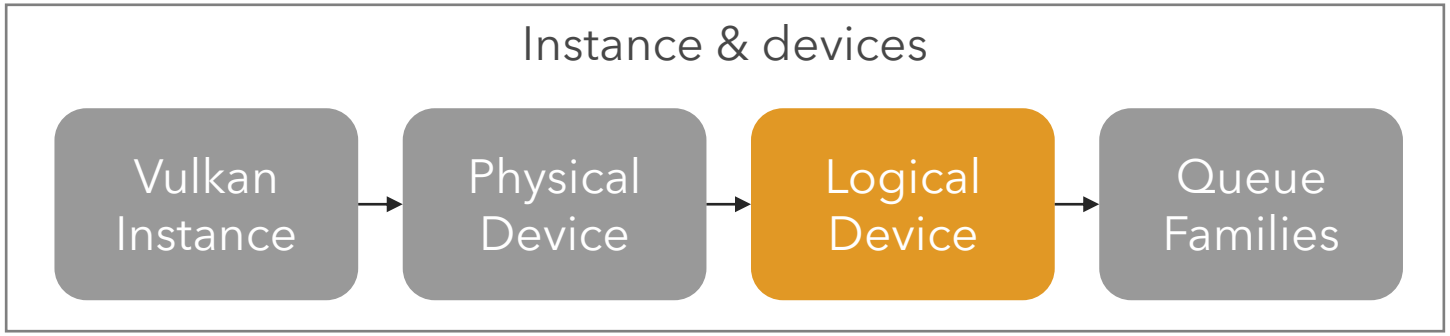

Instance & devices
Vulkan Instance
Physical Device
Logical Device
Queue Families



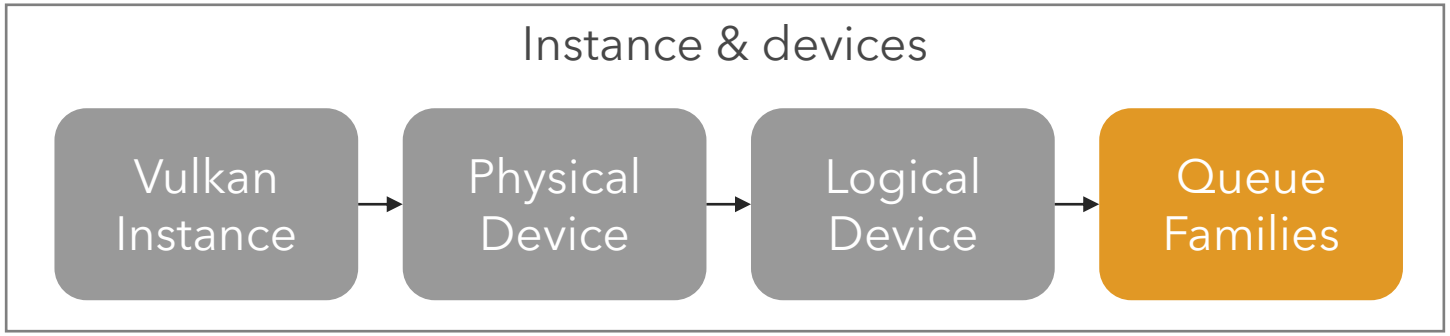
- Specify application name, Vulkan version
- Check GLFW extension support
- Enable validation layers for debugging



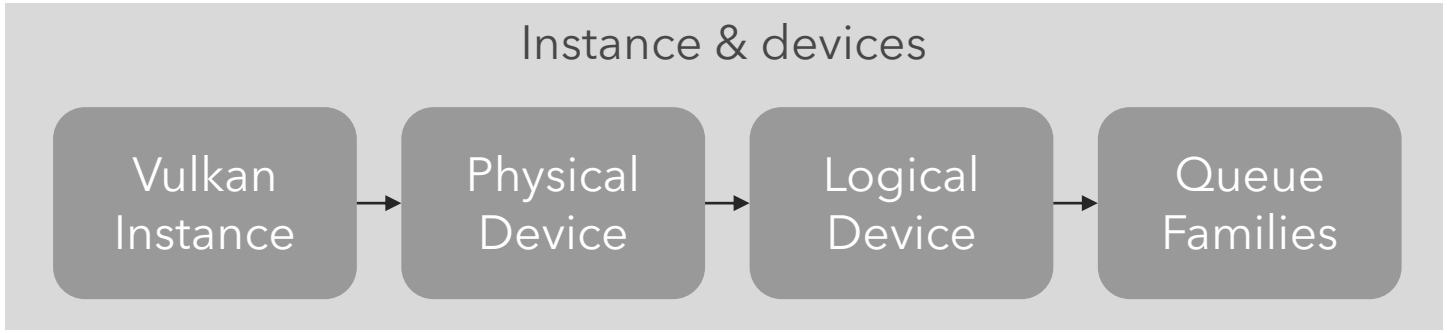
- Enumerate through devices available and select based on criteria
- e.g, extension support, memory size, queue support

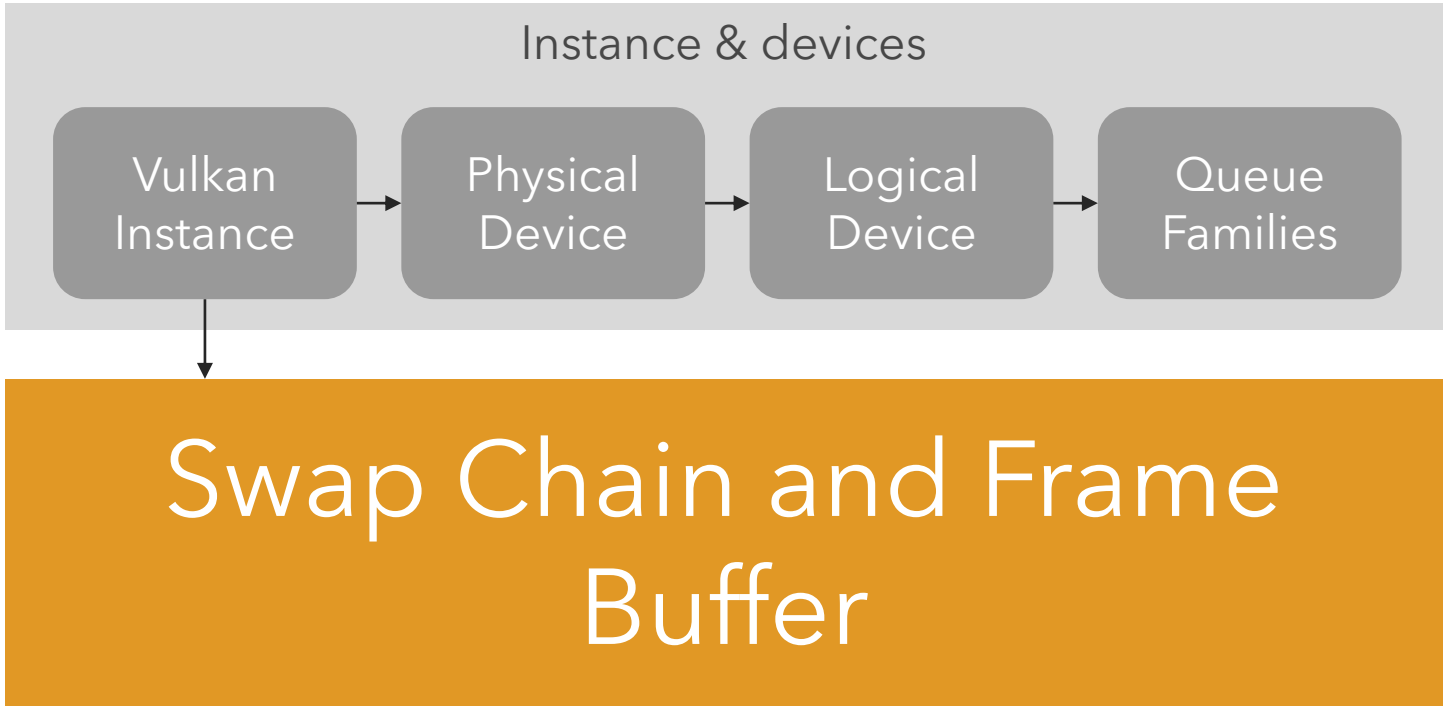


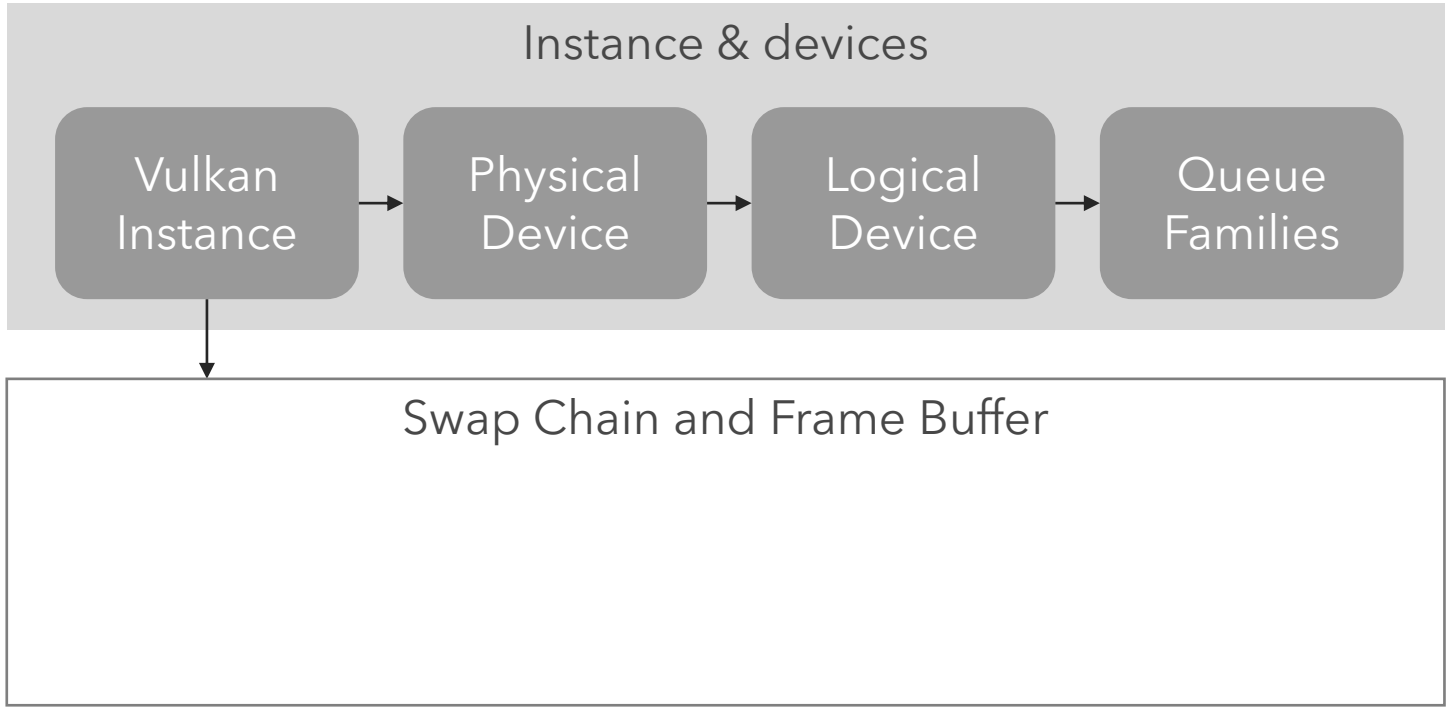
Specify the enabled features, extensions, layers, queue families

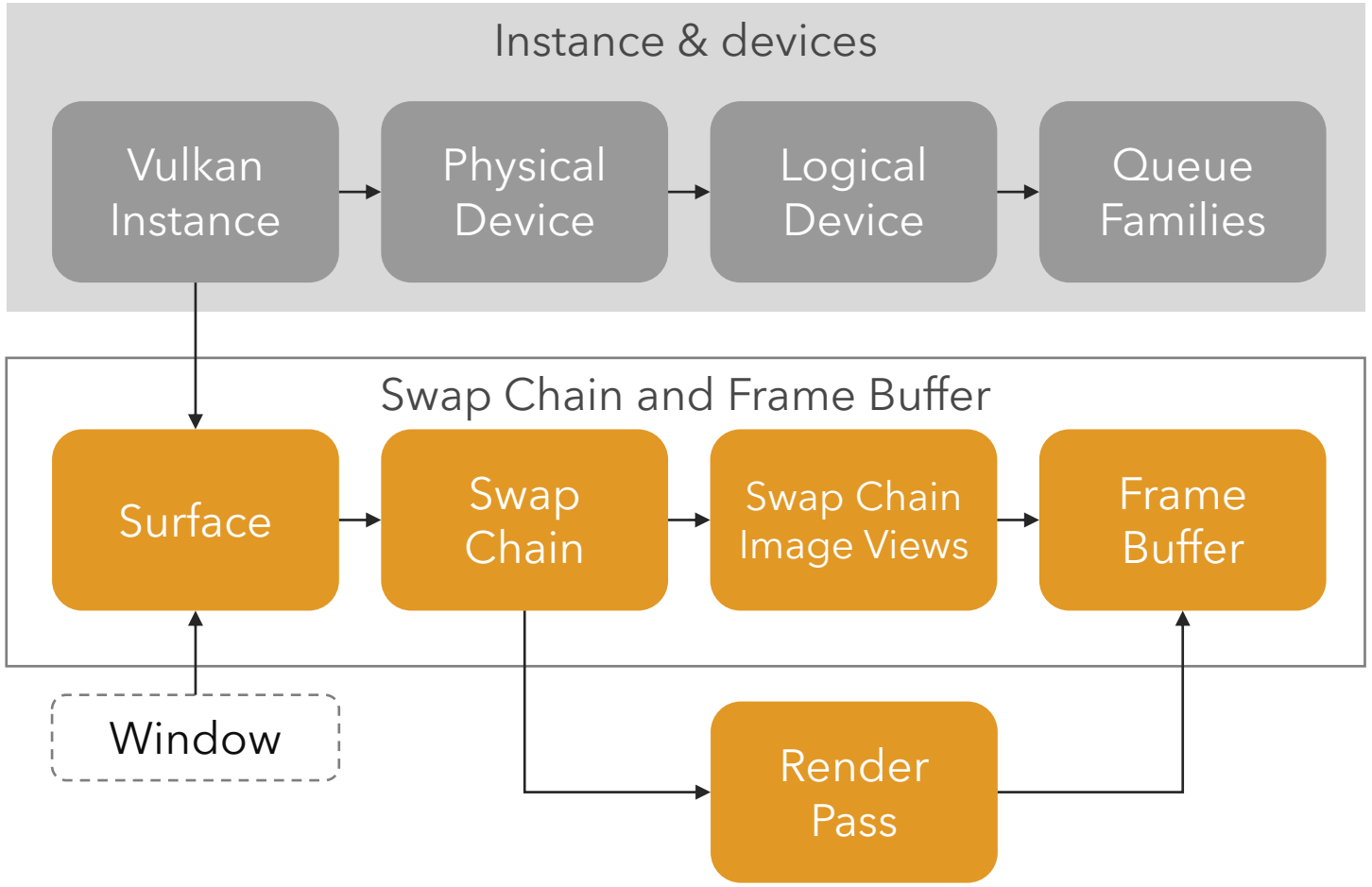


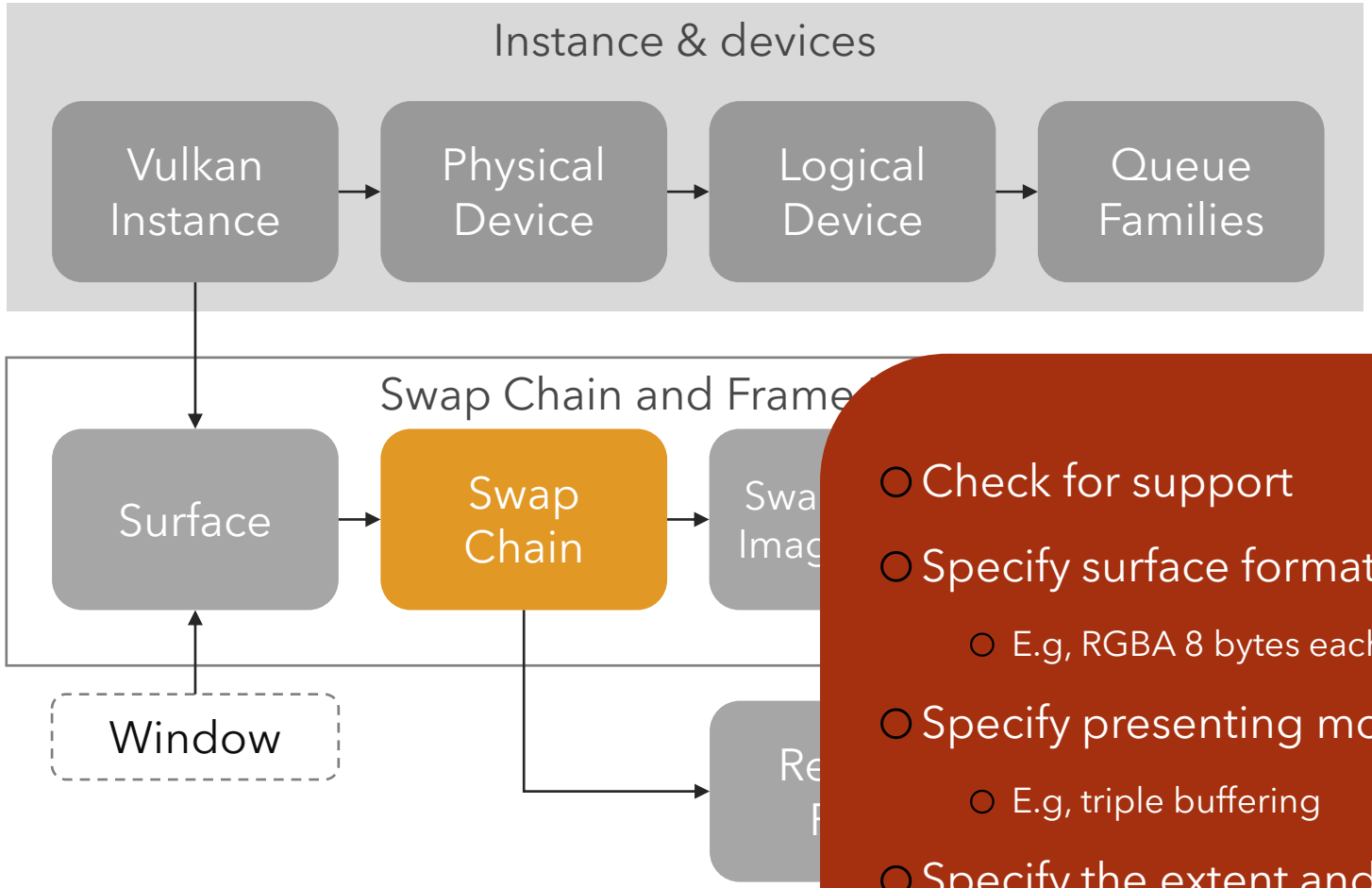
Graphics, presenting, compute


Instance & devices
Vulkan Instance
Physical Device
Logical Device
Queue Families

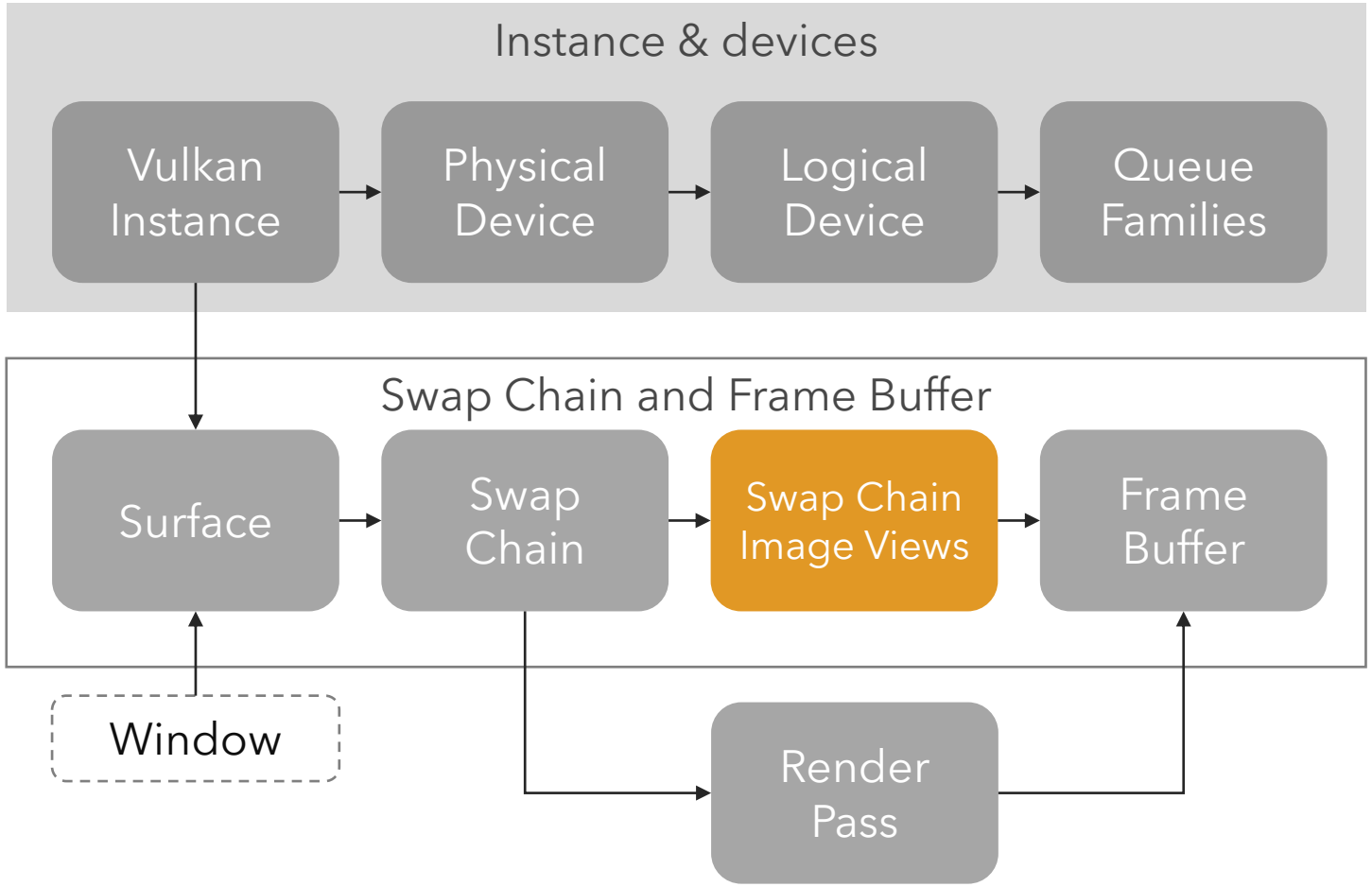

Instance & devices
Vulkan Instance
Physical Device
Logical Device
Queue Families
Swap Chain and Frame Buffer


Instance & devices
Vulkan Instance
Physical Device
Logical Device
Queue Families
Swap Chain and Frame Buffer

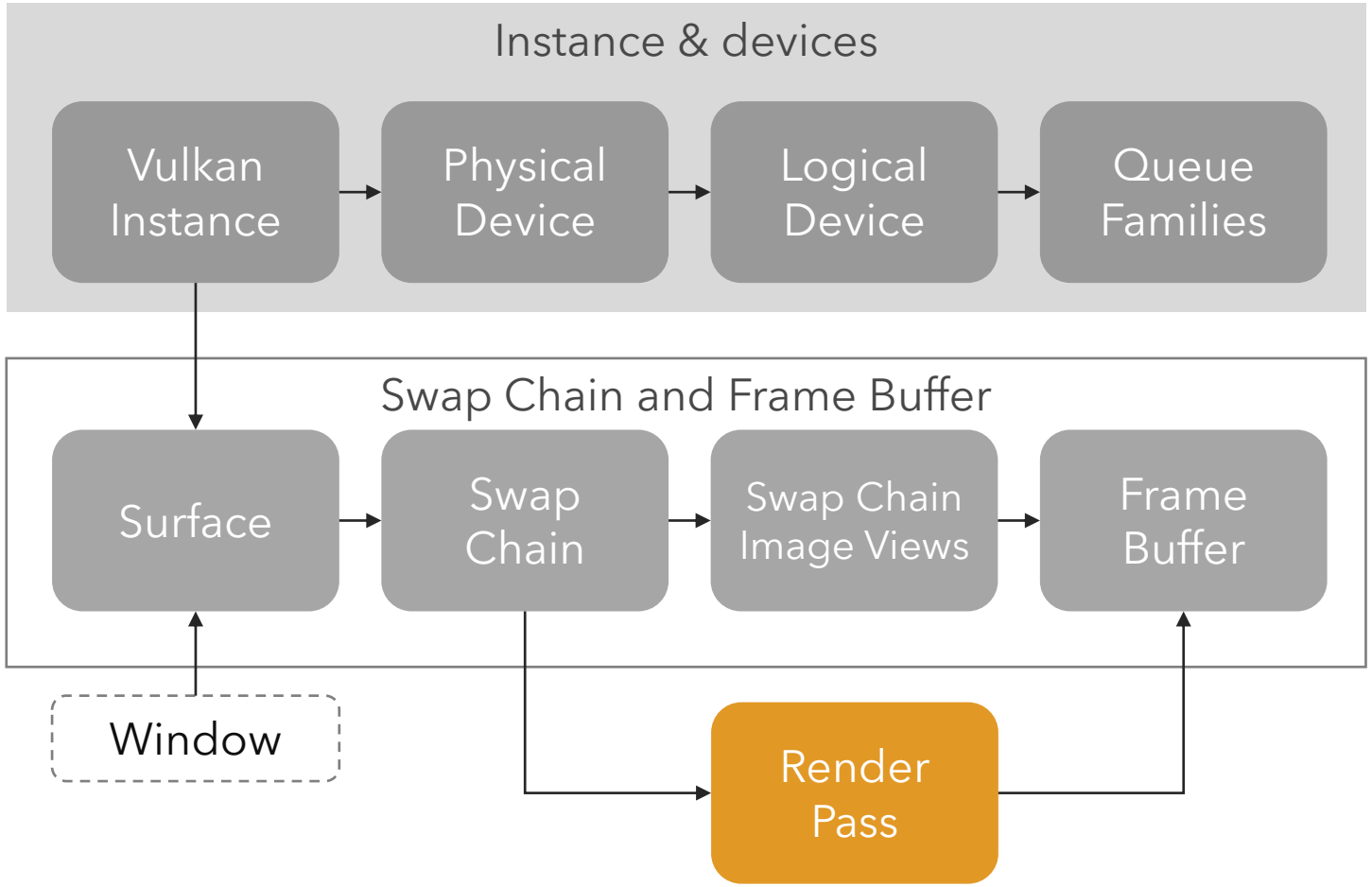

Instance & devices
Vulkan Instance
Physical Device
Logical Device
Queue Families
Swap Chain and Frame Buffer
Surface
Swap Chain
Swap Chain Image Views
Frame Buffer
Window
Render Pass



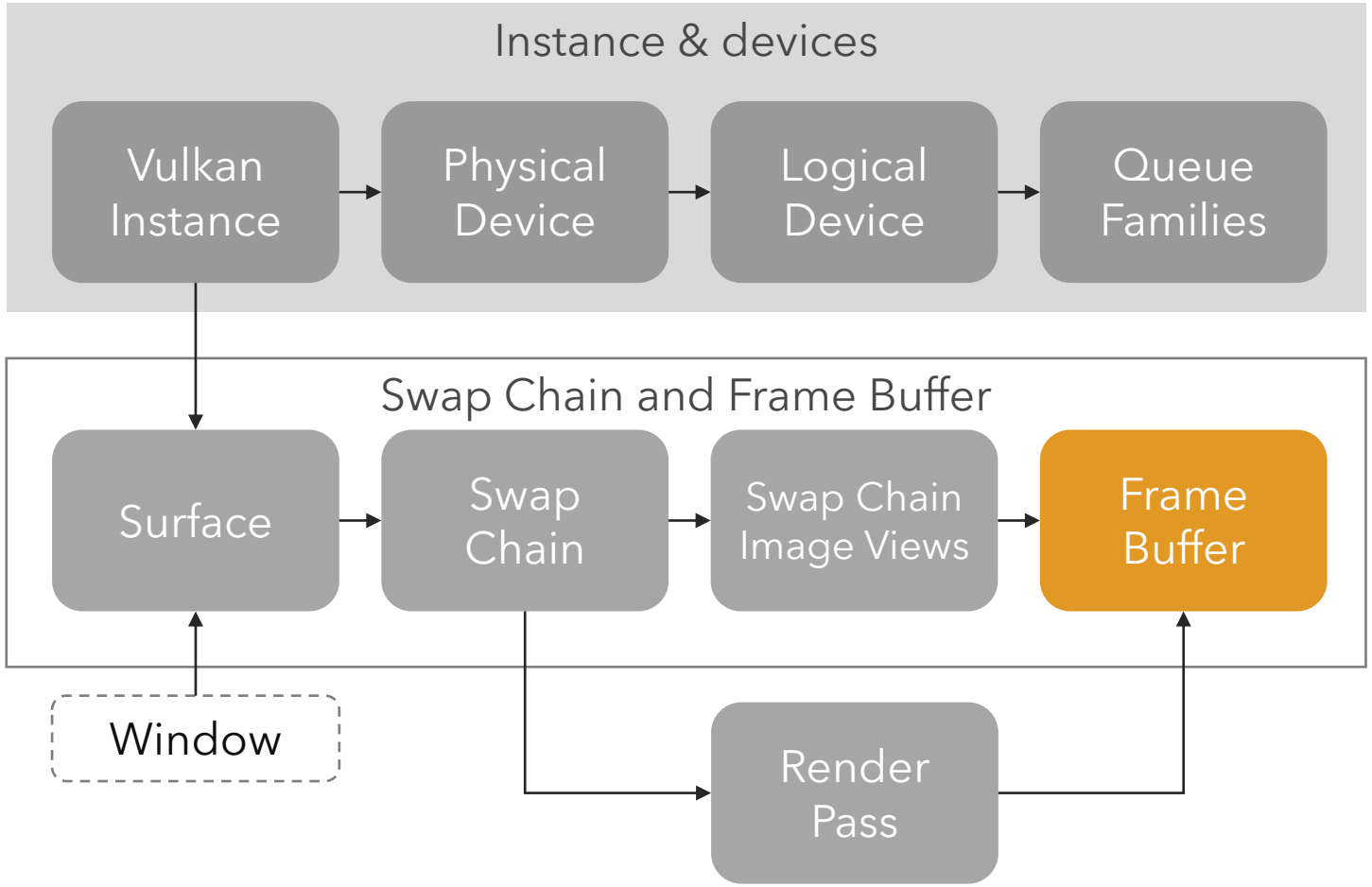
- Check for support
- Specify surface format and color space
  - E.g, RGBA 8 bytes each, sRGB color space
- Specify presenting mode:
  - E.g, triple buffering
- Specify the extent and screen resolution
  - Dynamic states: can be updated without the need to recreate the pipeline



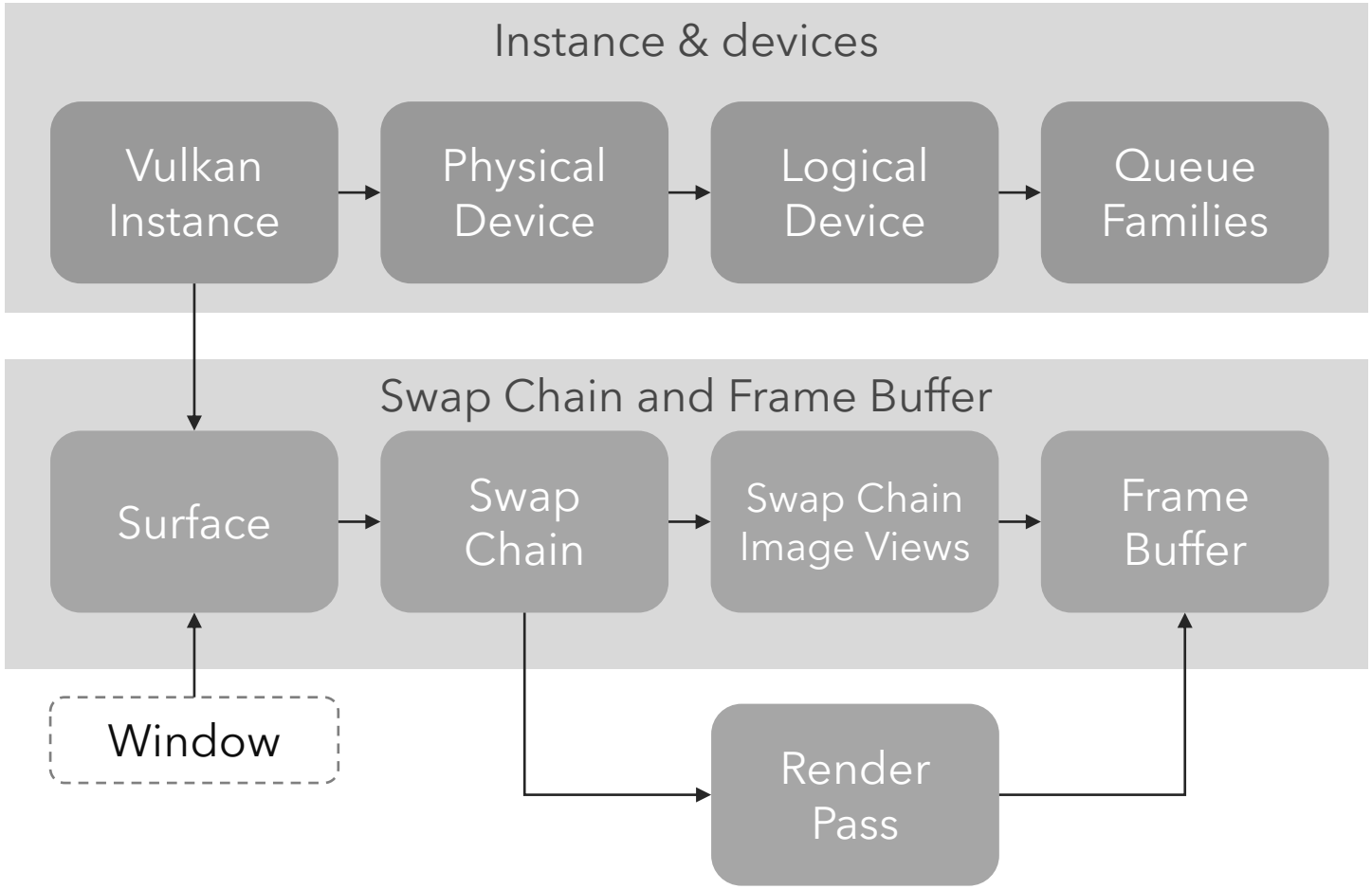
Specify swap chain image, format, and mip levels.. Etc



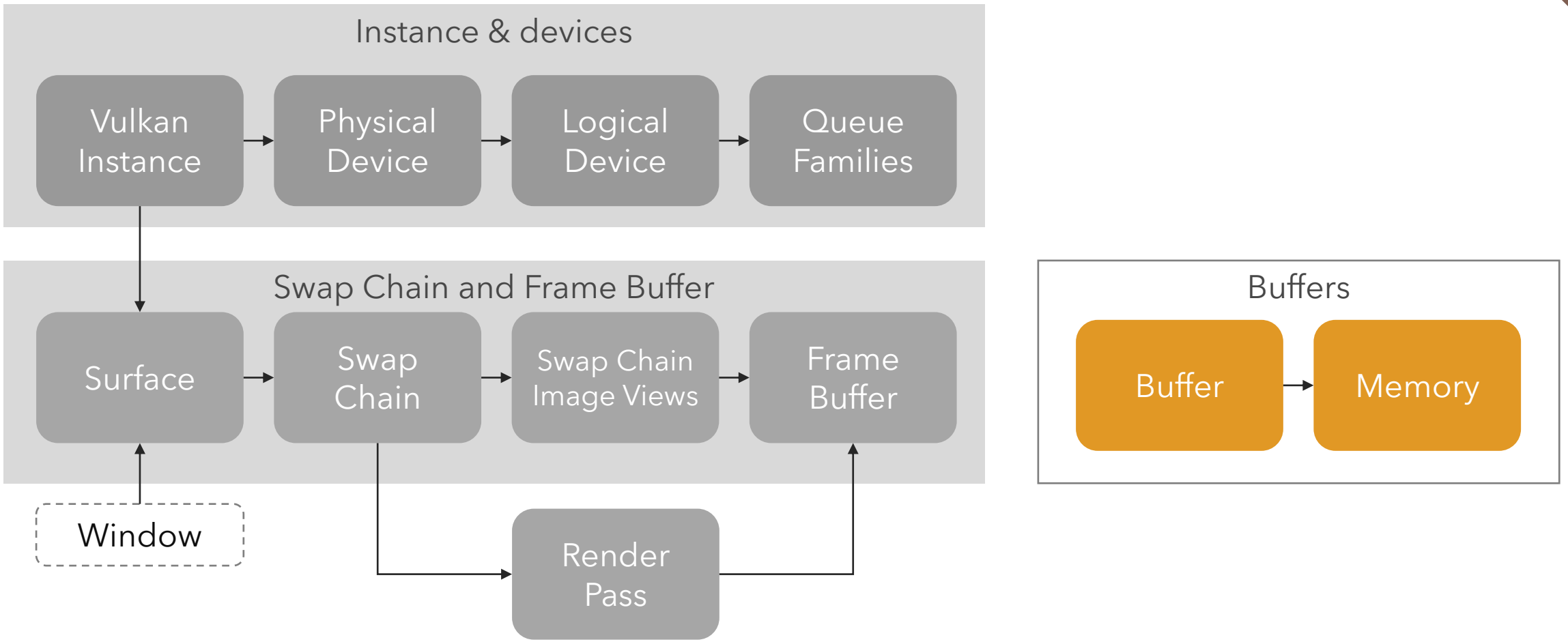
- Specify swap chain format
- Load and store operations
- Attachments

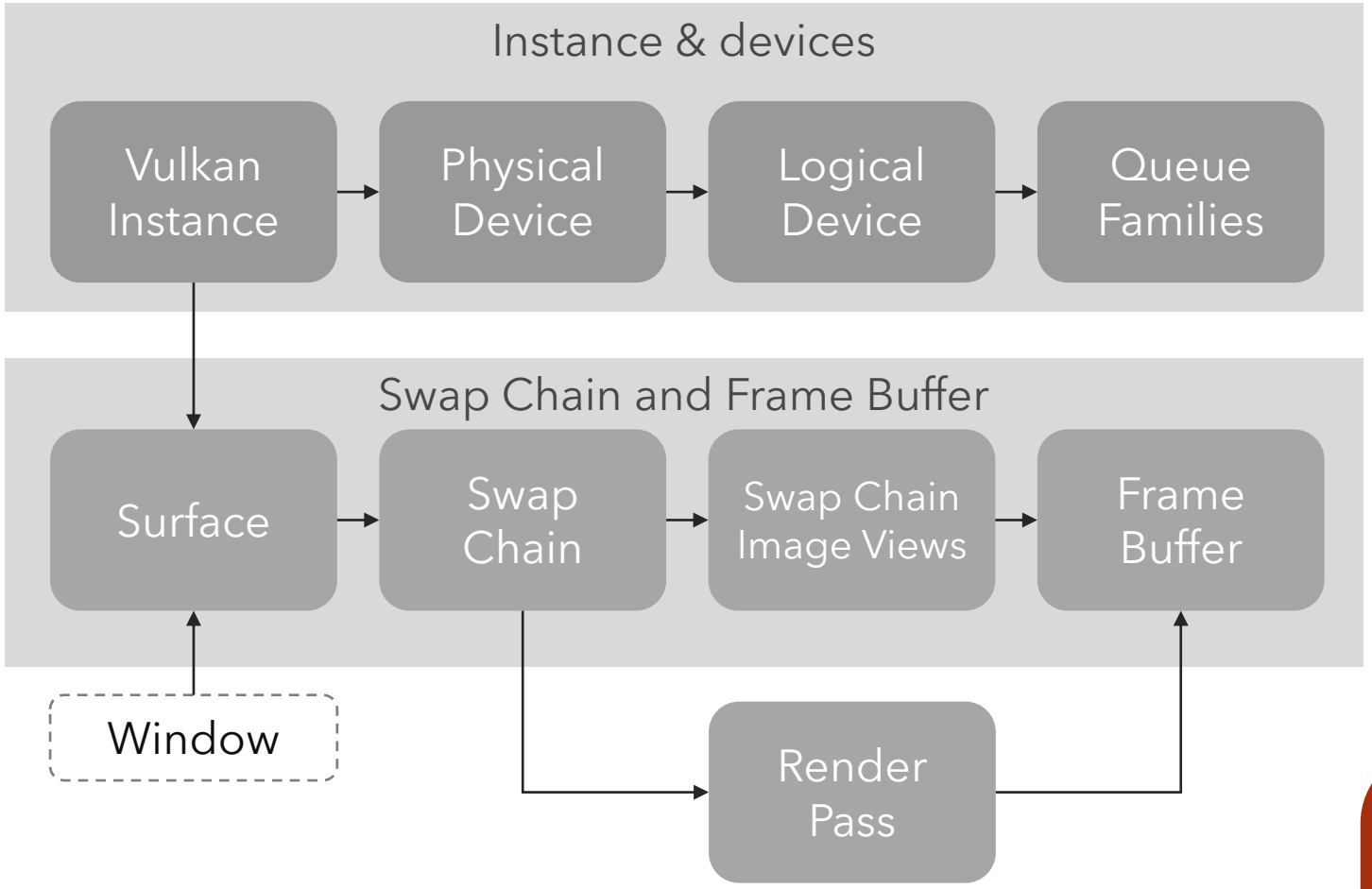


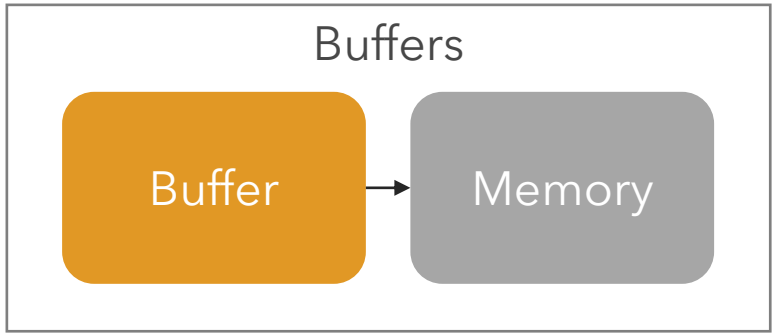
- Specify the swap chain image view as an attachment
- Specify the render pass
- Extent and layers


Instance & devices
Vulkan Instance
Physical Device
Logical Device
Queue Families
Swap Chain and Frame Buffer
Surface
Swap Chain
Swap Chain Image Views
Frame Buffer
Window
Render Pass

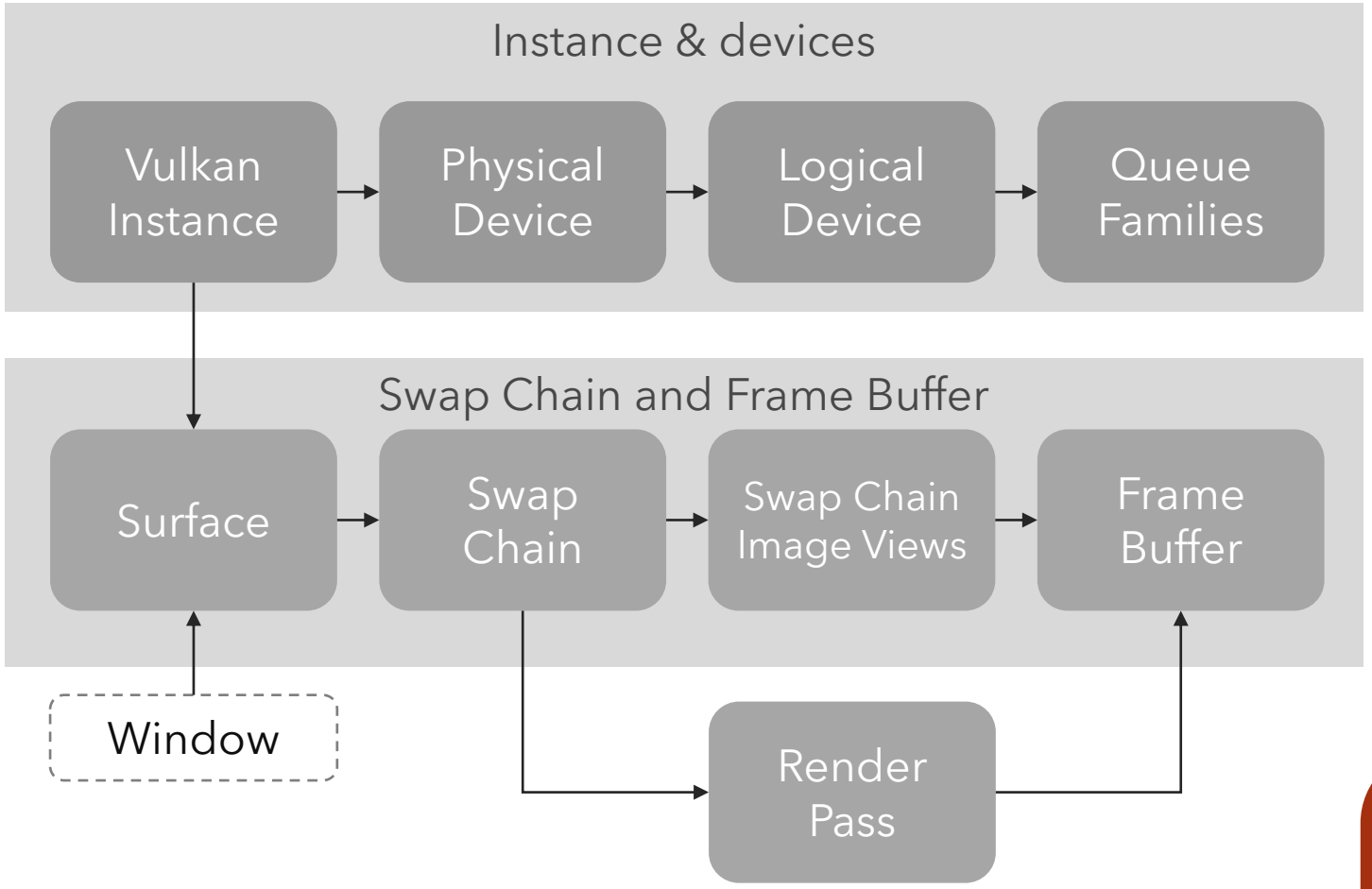
Buffers

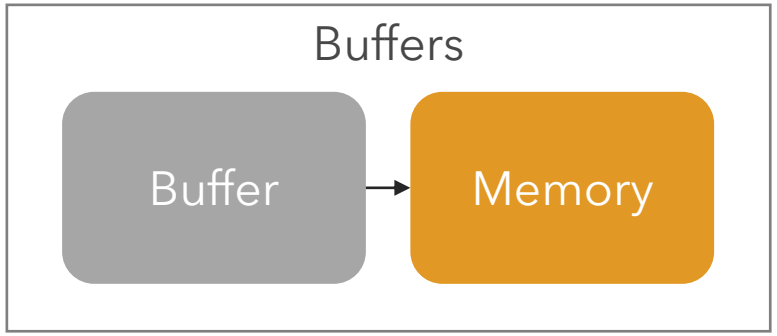

Instance & devices
Vulkan Instance
Physical Device
Logical Device
Queue Families
Swap Chain and Frame Buffer
Surface
Swap Chain
Swap Chain Image Views
Frame Buffer
Window
Render Pass
Buffers
Buffer
Memory





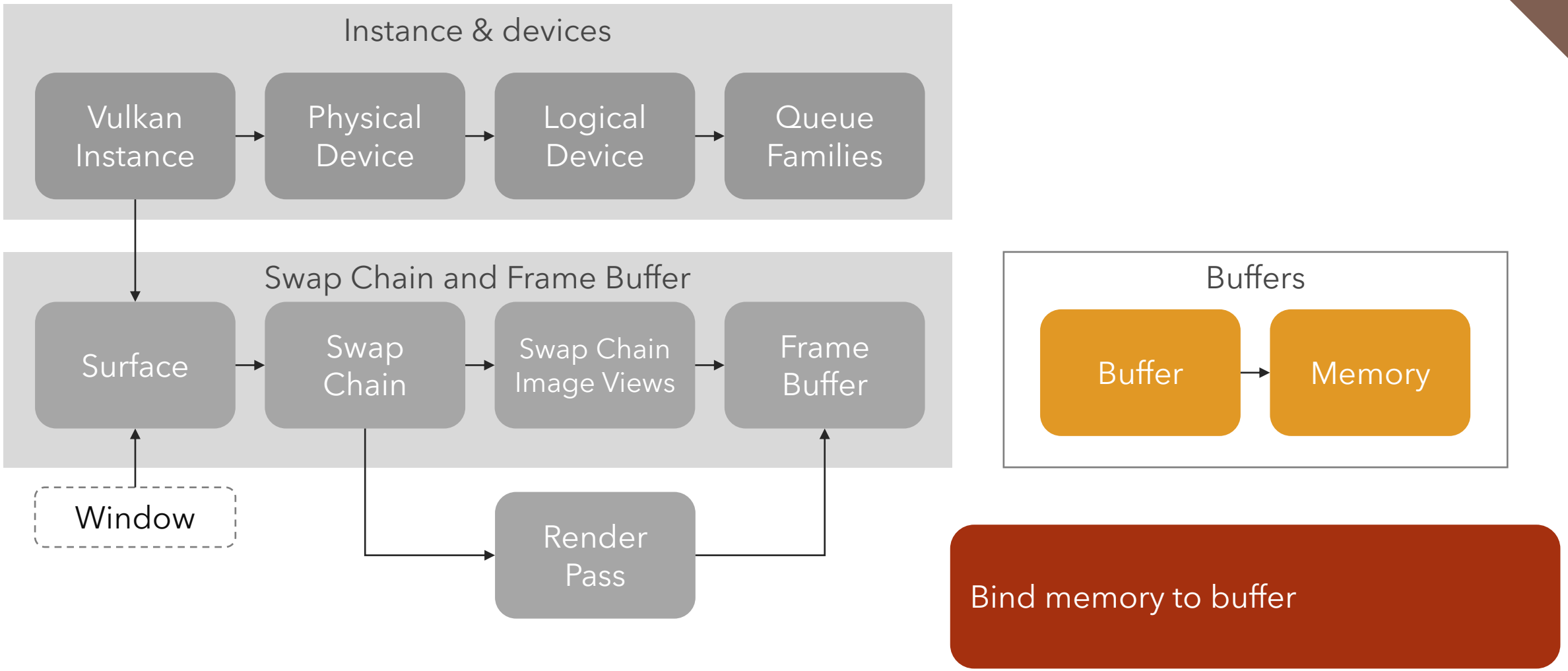
- Specify size
- Specify usage: eg vertex, index, uniform, transfer, ..etc

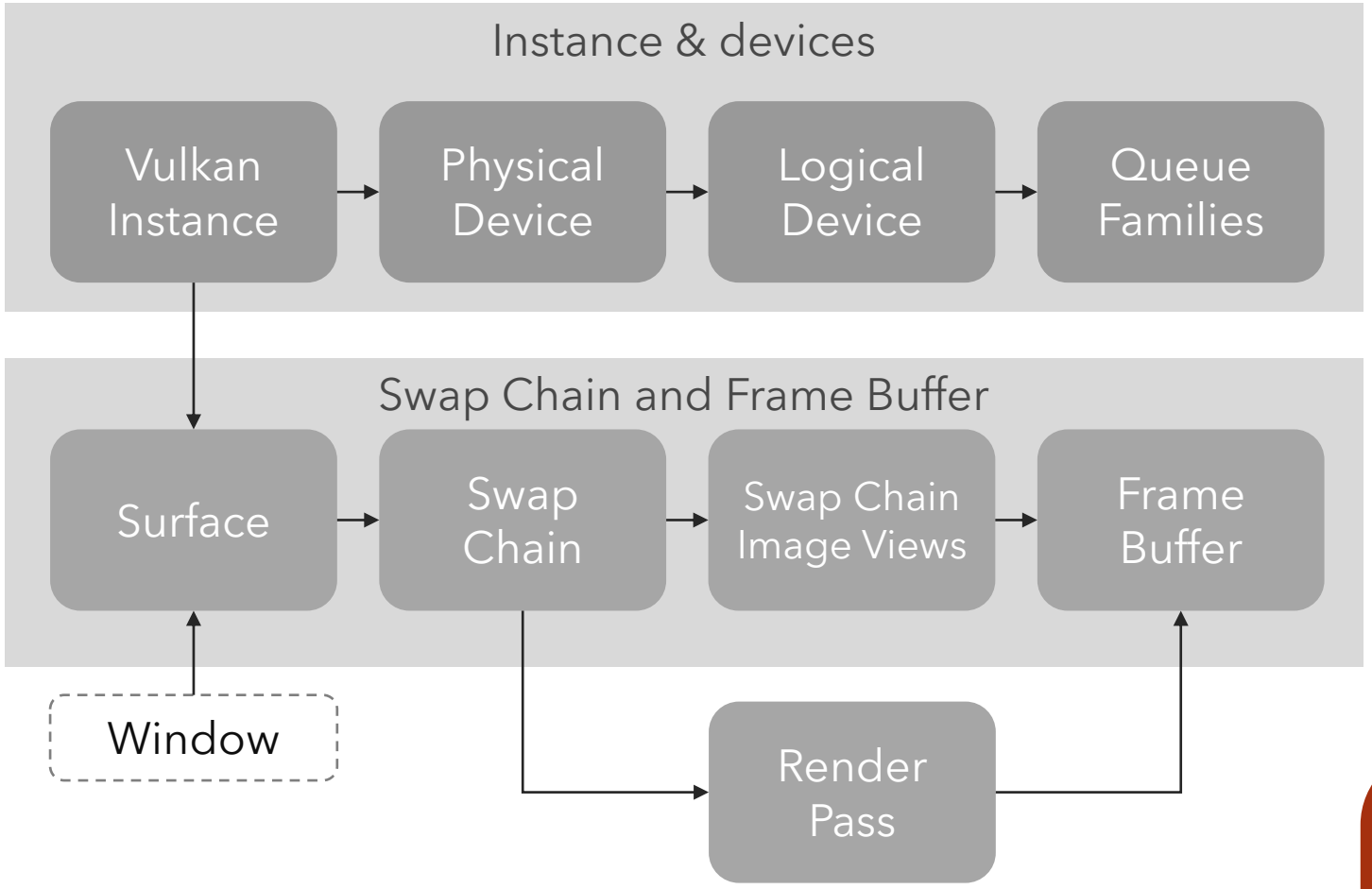


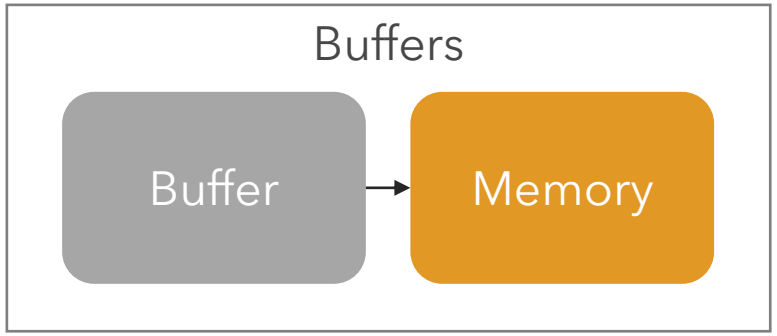


Query for suitable memory type

Allocate memory

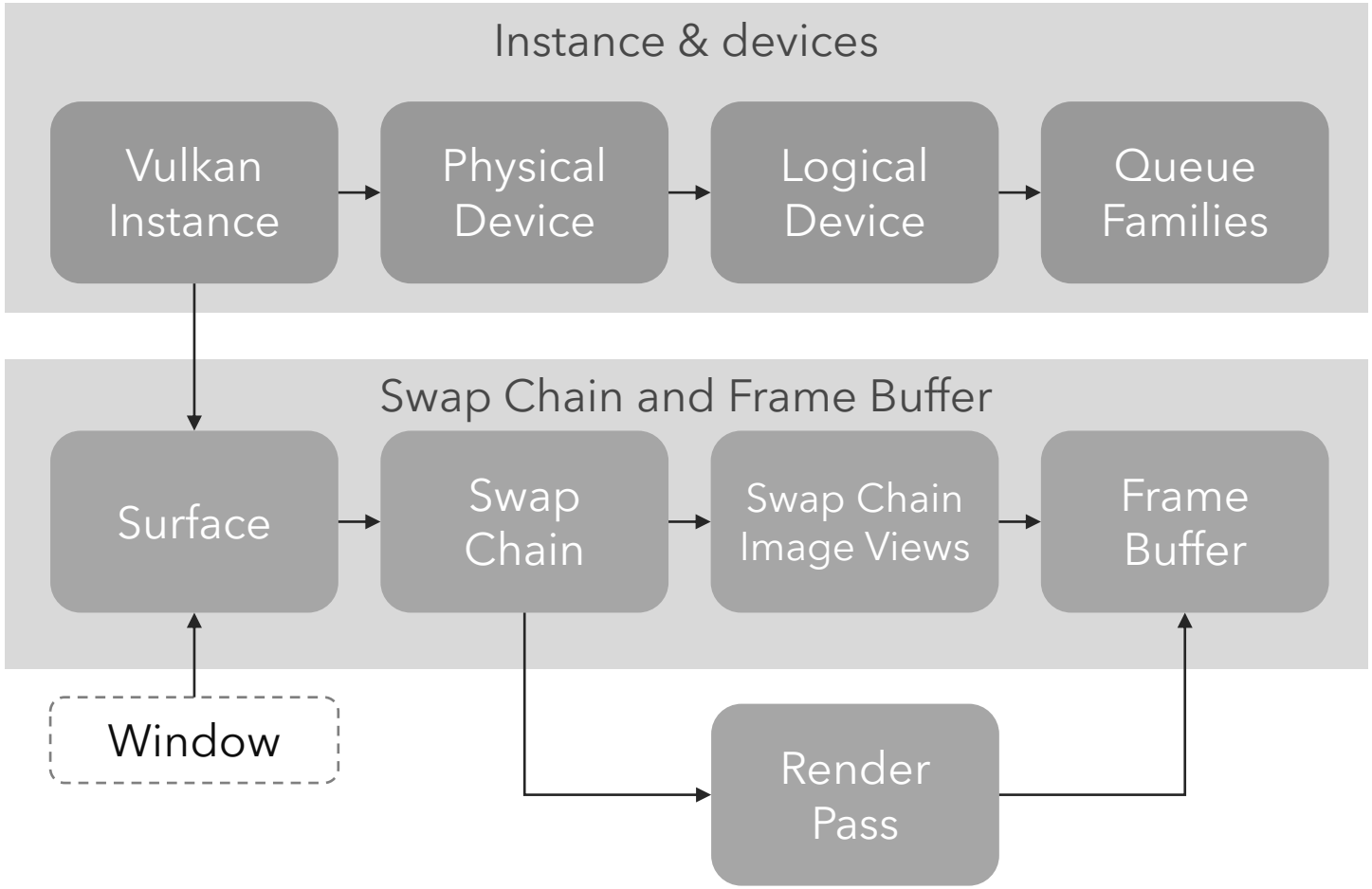

Instance & devices
Vulkan Instance
Physical Device
Logical Device
Queue Families
Swap Chain and Frame Buffer
Surface
Swap Chain
Swap Chain Image Views
Frame Buffer
Window
Render Pass
Buffers
Buffer
Memory
Bind memory to buffer

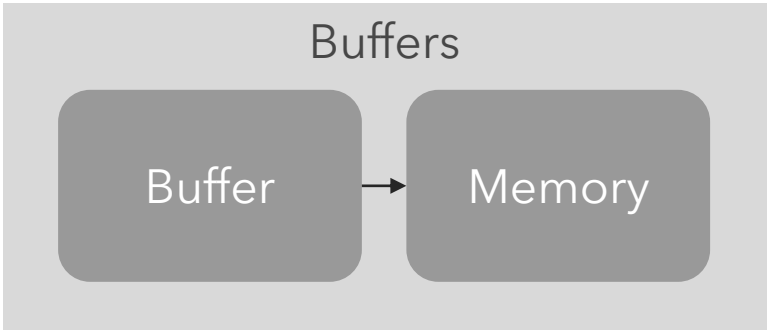




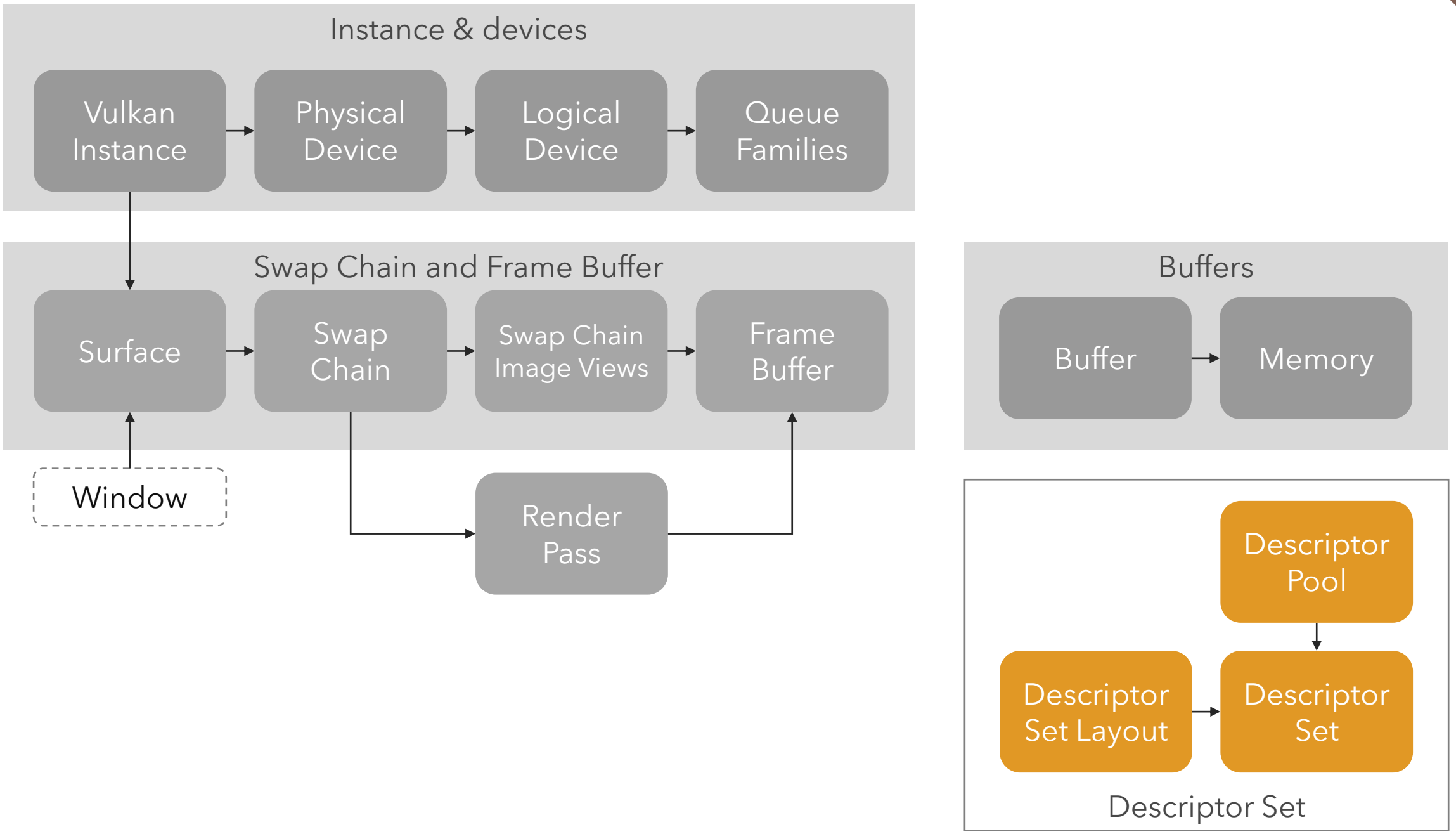
To copy from CPU to GPU:

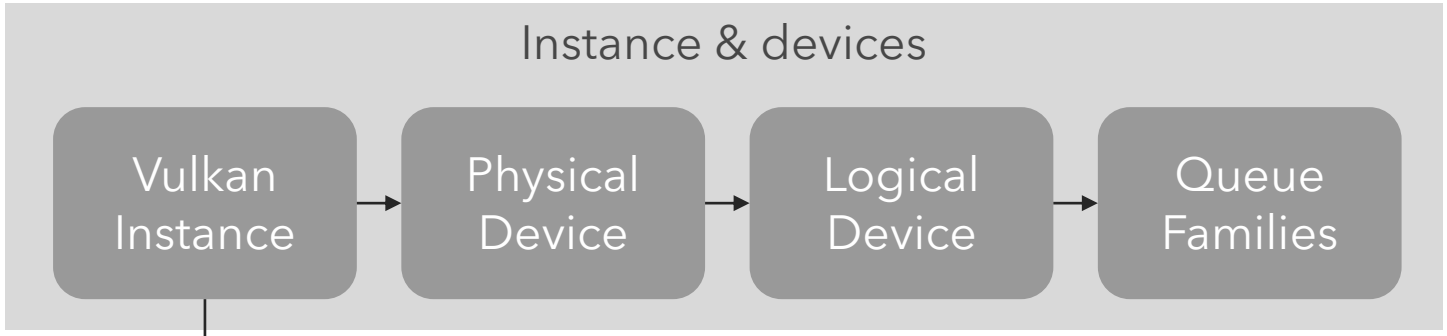
- Map GPU memory to a pointer
- Copy local data to the pointer
- Unmap the GPU memory from the pointer

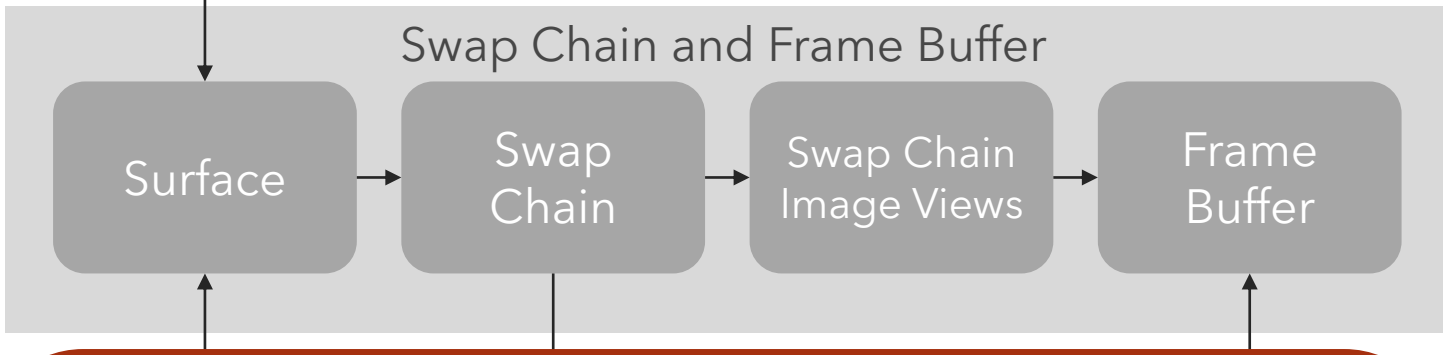


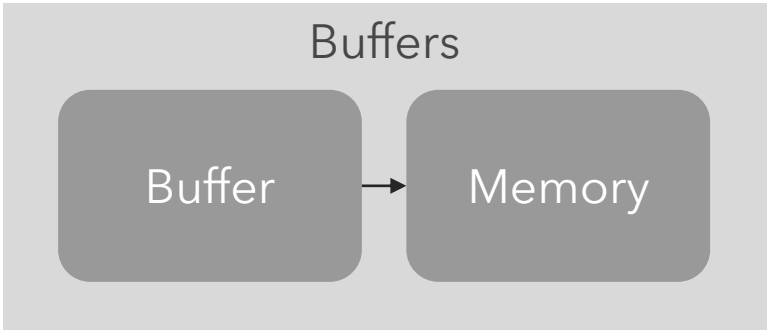


Descriptor Set

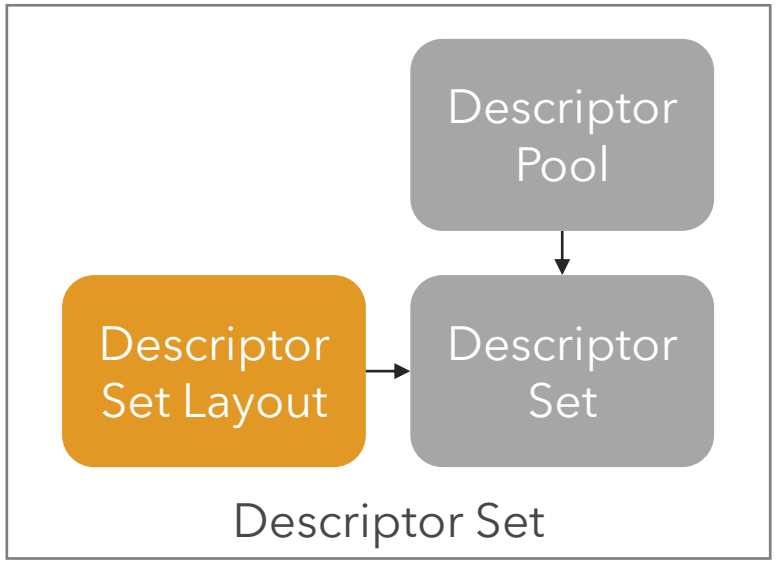

Instance & devices
Vulkan Instance
Physical Device
Logical Device
Queue Families
Swap Chain and Frame Buffer
Surface
Swap Chain
Swap Chain Image Views
Frame Buffer
Window
Render Pass
Buffers
Buffer
Memory
Descriptor Pool
Descriptor Set Layout
Descriptor Set
Descriptor Set

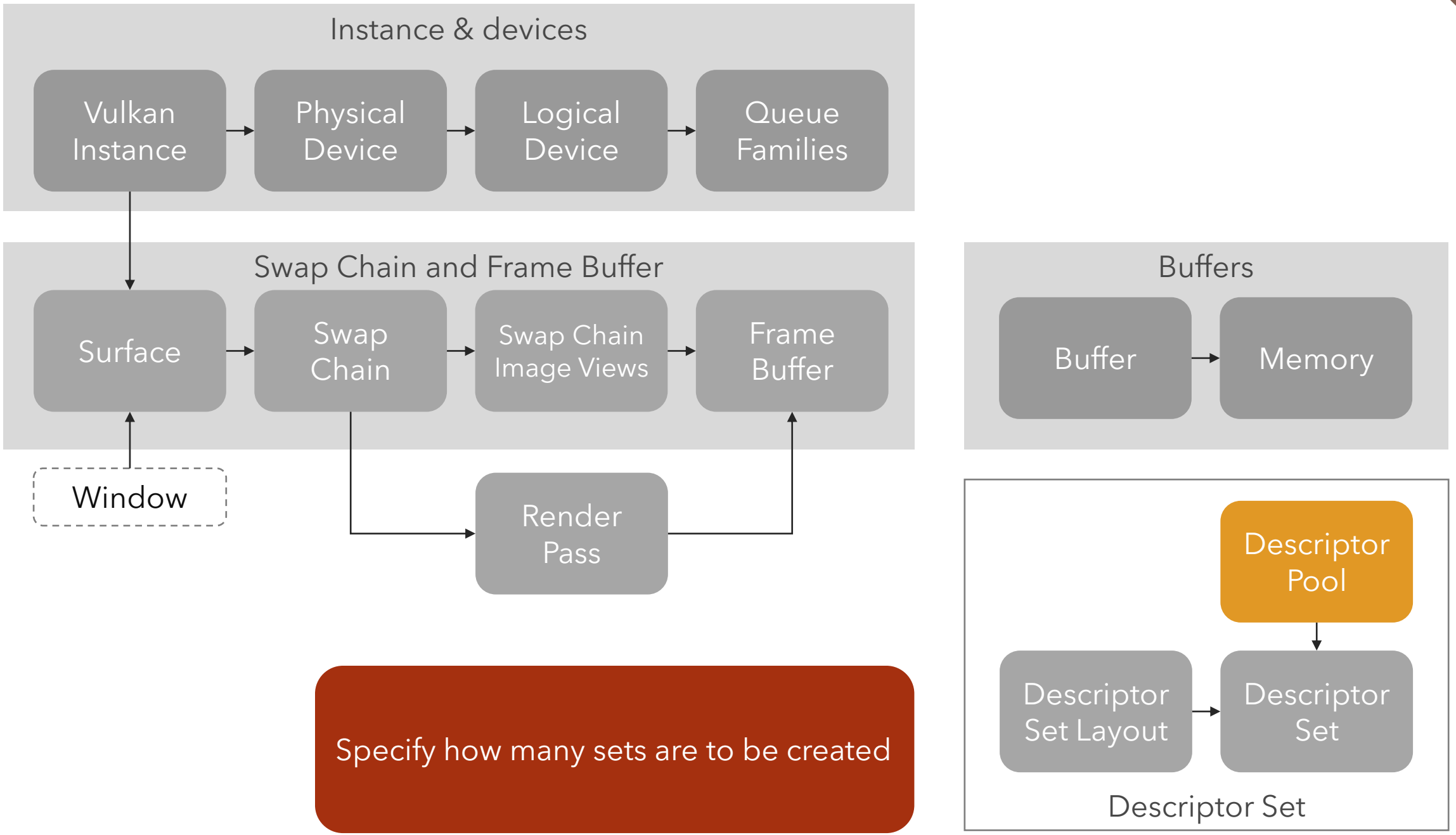


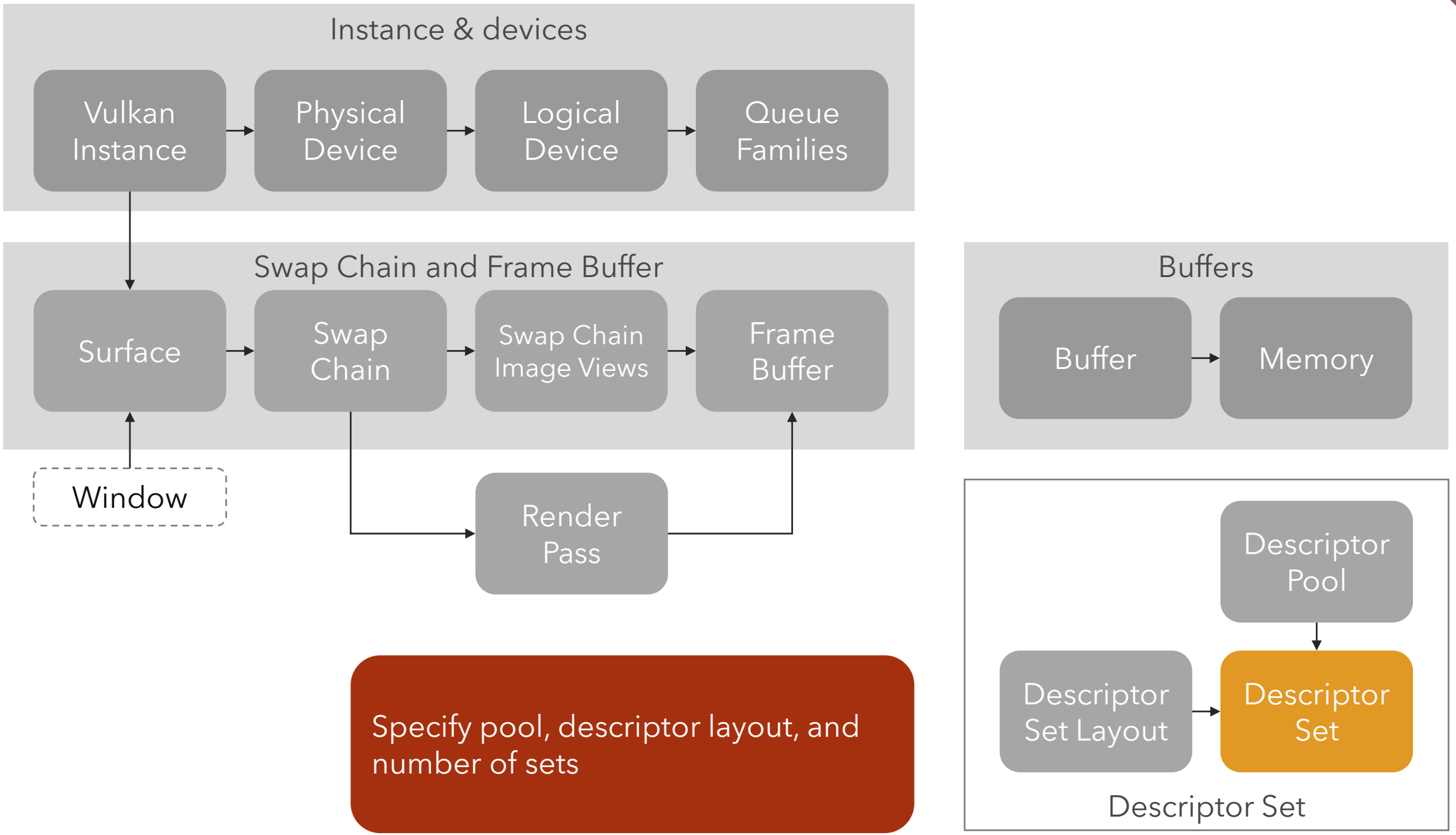


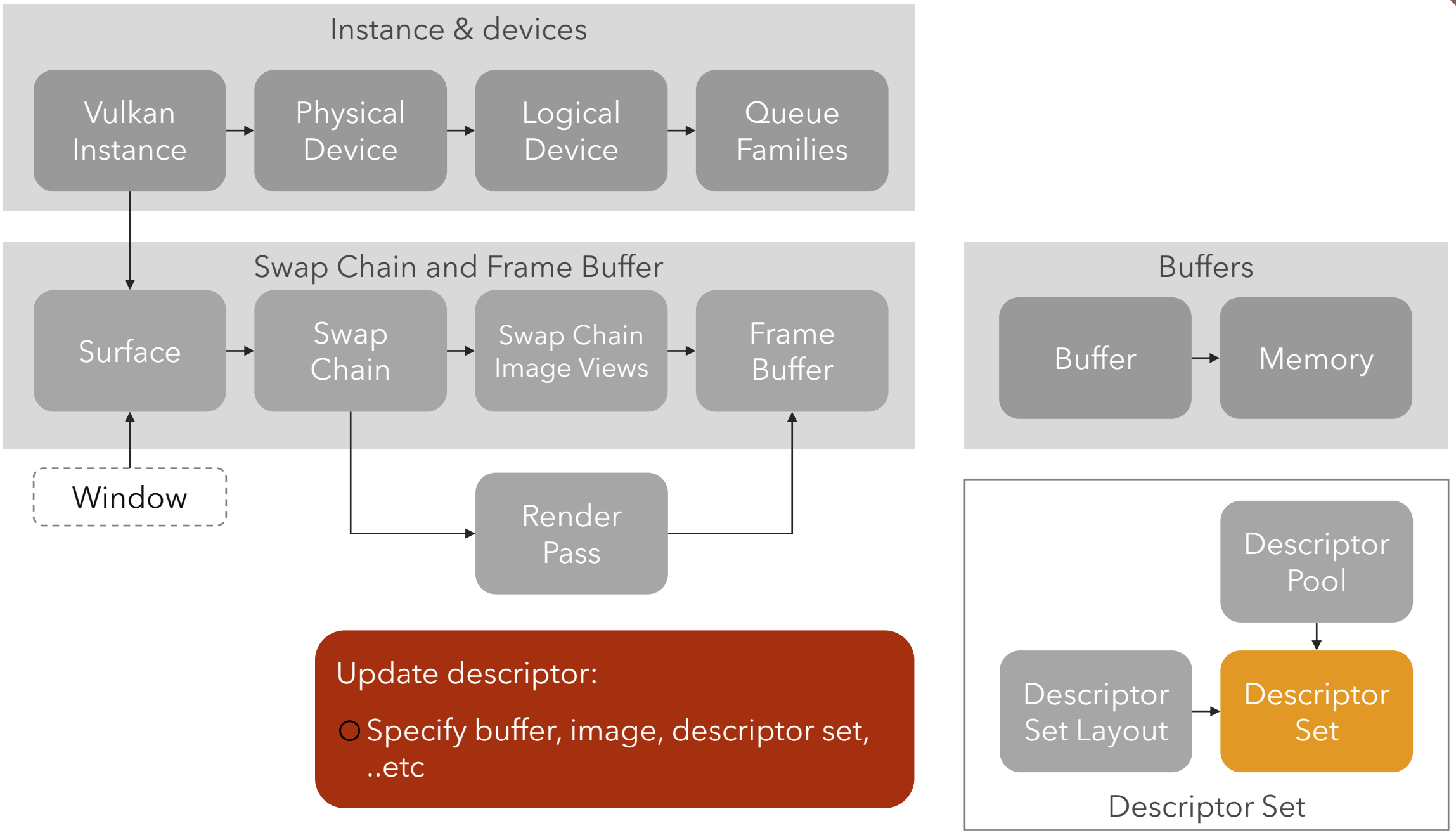


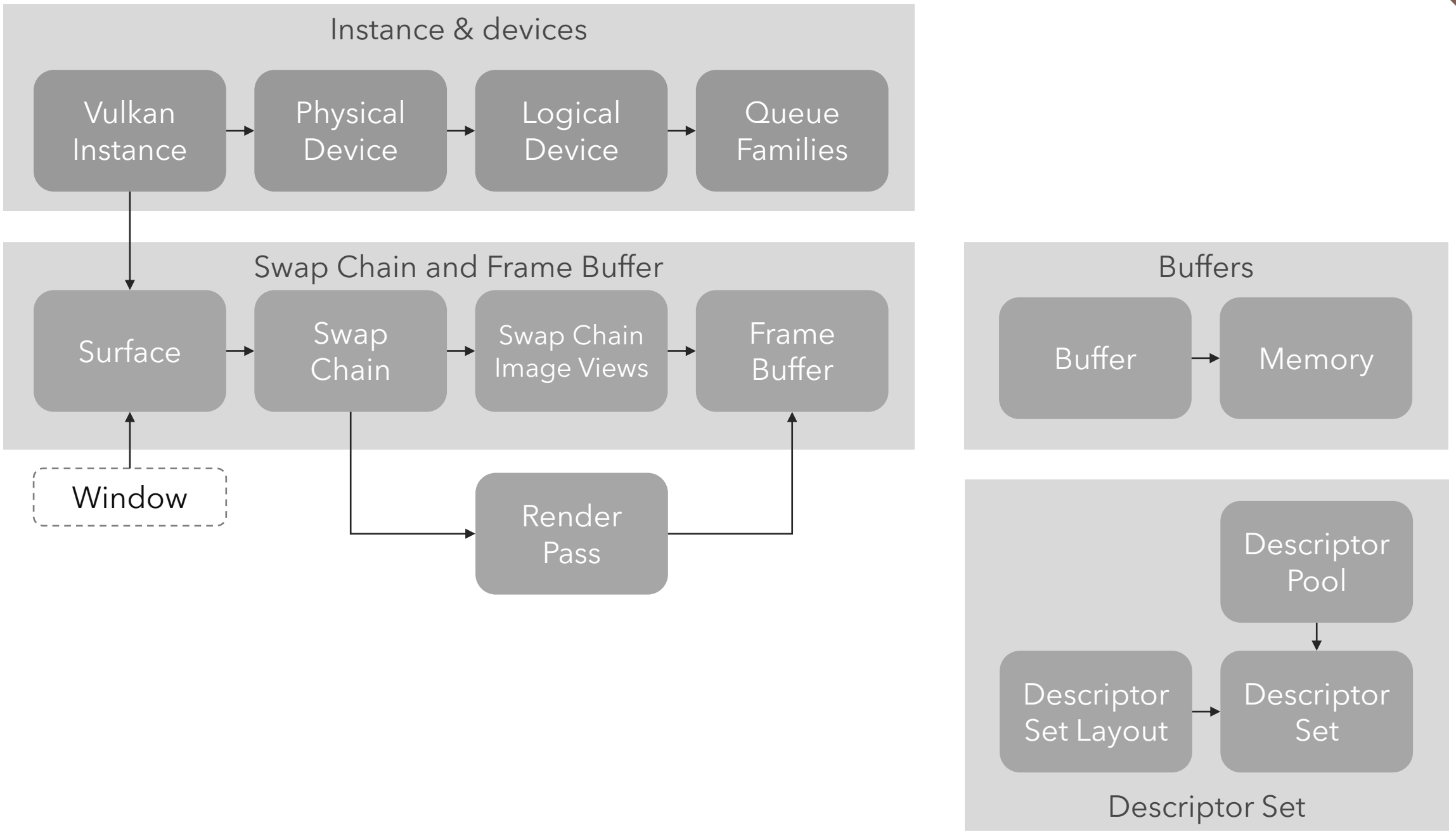
- Specify the binding number (for access inside shaders)
- Specify type: uniform, storage buffer, images
- Specify shading stage: vertex, fragment, compute
- Specify how many bindings are inside the set

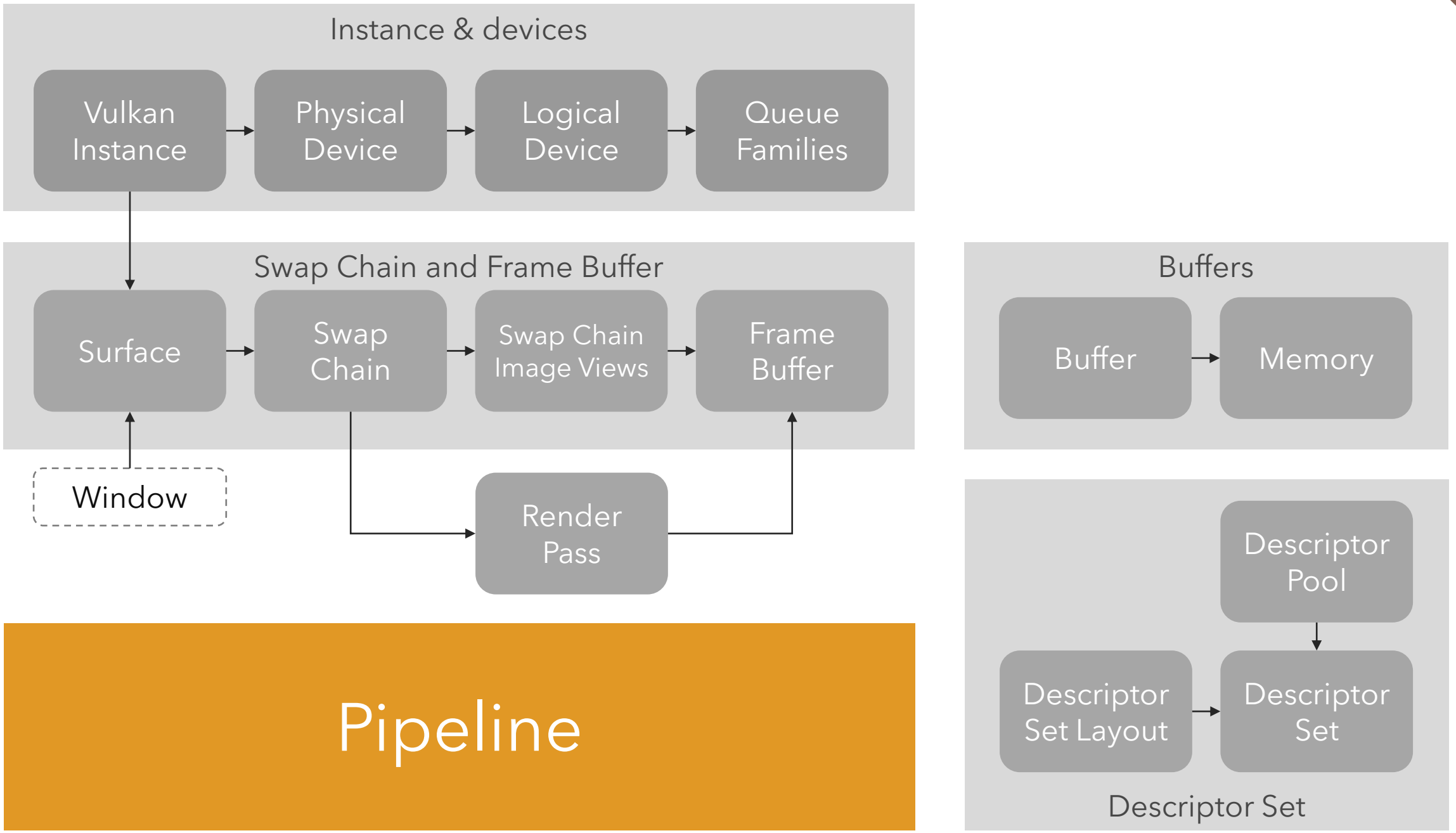


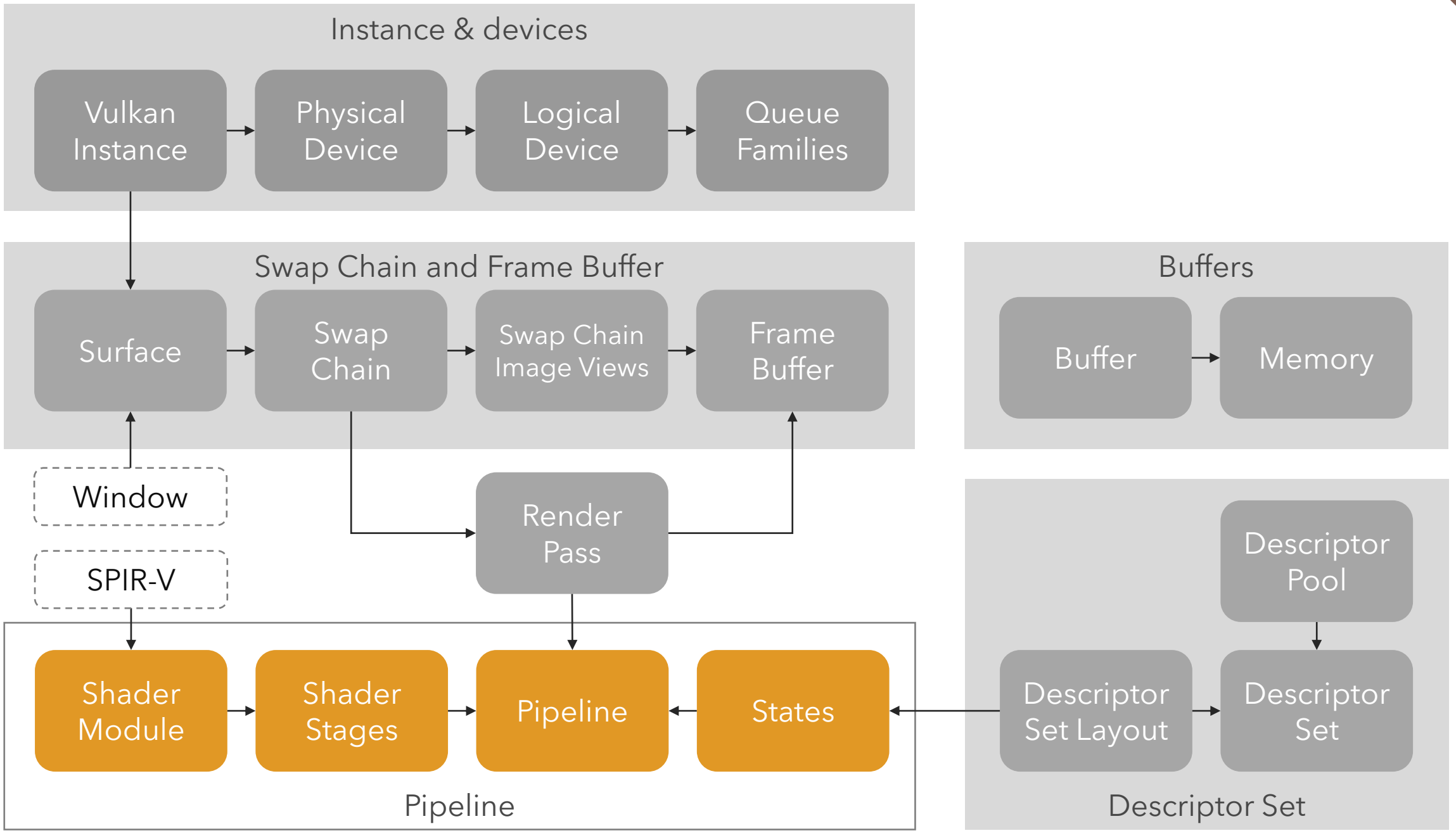

Instance & devices
Vulkan Instance
Physical Device
Logical Device
Queue Families
Swap Chain and Frame Buffer
Surface
Swap Chain
Swap Chain Image Views
Frame Buffer
Window
Render Pass
Buffers
Buffer
Memory
Descriptor Pool
Descriptor Set Layout
Descriptor Set
Descriptor Set
Specify how many sets are to be created

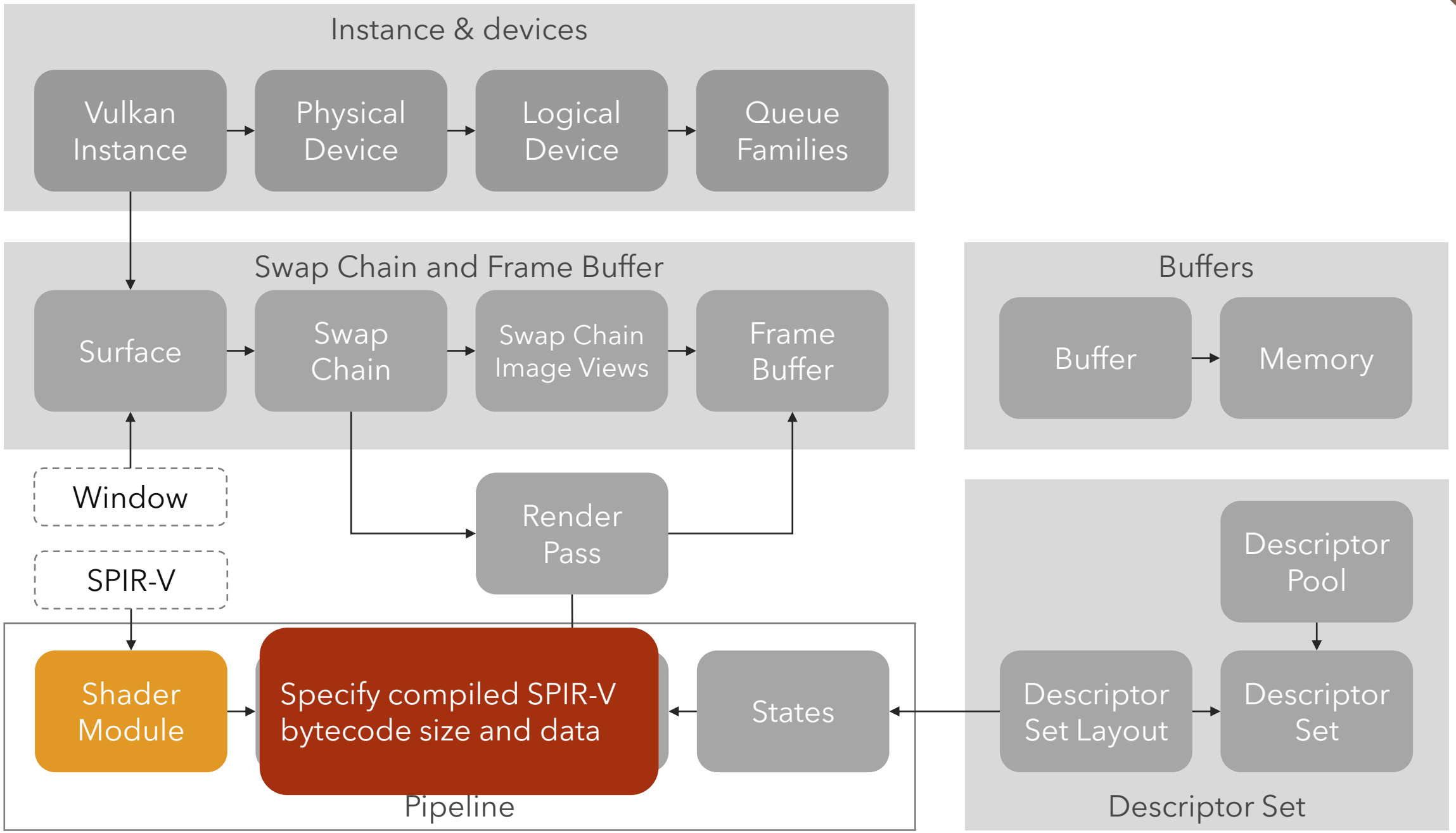

Instance & devices
Vulkan Instance
Physical Device
Logical Device
Queue Families
Swap Chain and Frame Buffer
Surface
Swap Chain
Swap Chain Image Views
Frame Buffer
Window
Render Pass
Buffers
Buffer
Memory
Descriptor Pool
Descriptor Set Layout
Descriptor Set
Descriptor Set
Specify pool, descriptor layout, and number of sets

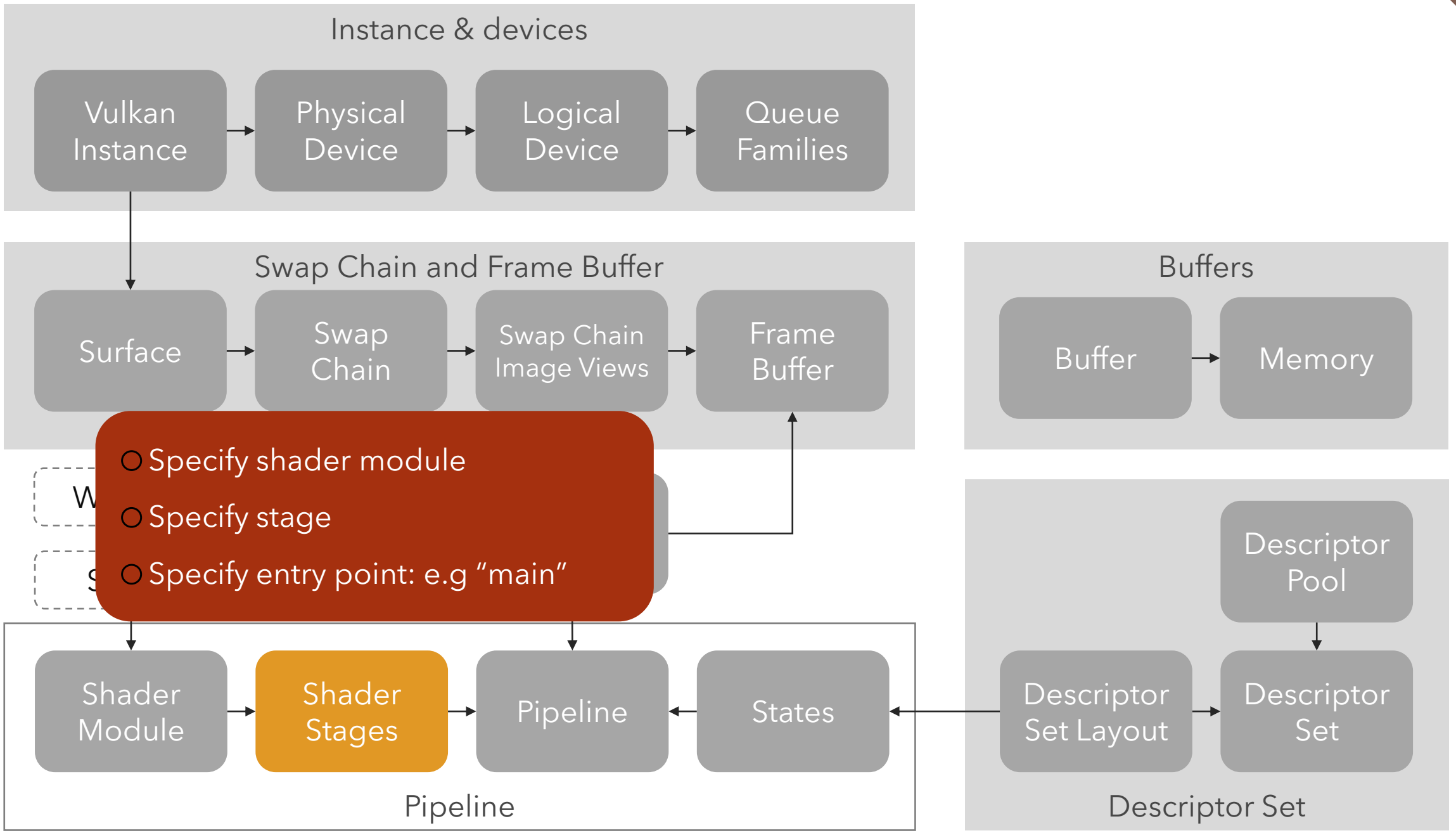

Instance & devices
Vulkan Instance
Physical Device
Logical Device
Queue Families
Swap Chain and Frame Buffer
Surface
Swap Chain
Swap Chain Image Views
Frame Buffer
Window
Render Pass
Buffers
Buffer
Memory
Descriptor Pool
Descriptor Set Layout
Descriptor Set
Descriptor Set
Update descriptor:
Specify buffer, image, descriptor set, ..etc


Instance & devices
Vulkan Instance
Physical Device
Logical Device
Queue Families
Swap Chain and Frame Buffer
Surface
Swap Chain
Swap Chain Image Views
Frame Buffer
Window
Render Pass
Buffers
Buffer
Memory
Descriptor Pool
Descriptor Set Layout
Descriptor Set
Descriptor Set

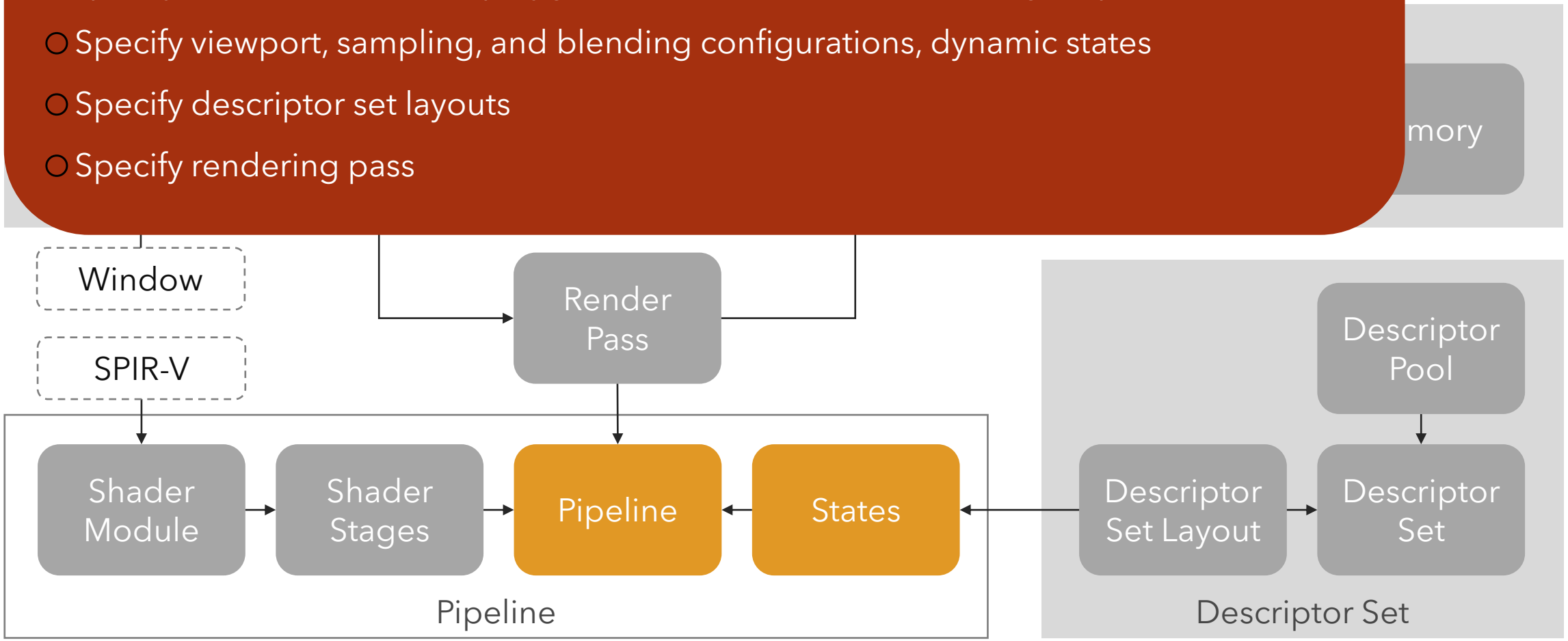

Instance & devices
Vulkan Instance
Physical Device
Logical Device
Queue Families
Swap Chain and Frame Buffer
Surface
Swap Chain
Swap Chain Image Views
Frame Buffer
Window
Render Pass
Pipeline
Buffers
Buffer
Memory
Descriptor Pool
Descriptor Set Layout
Descriptor Set
Descriptor Set

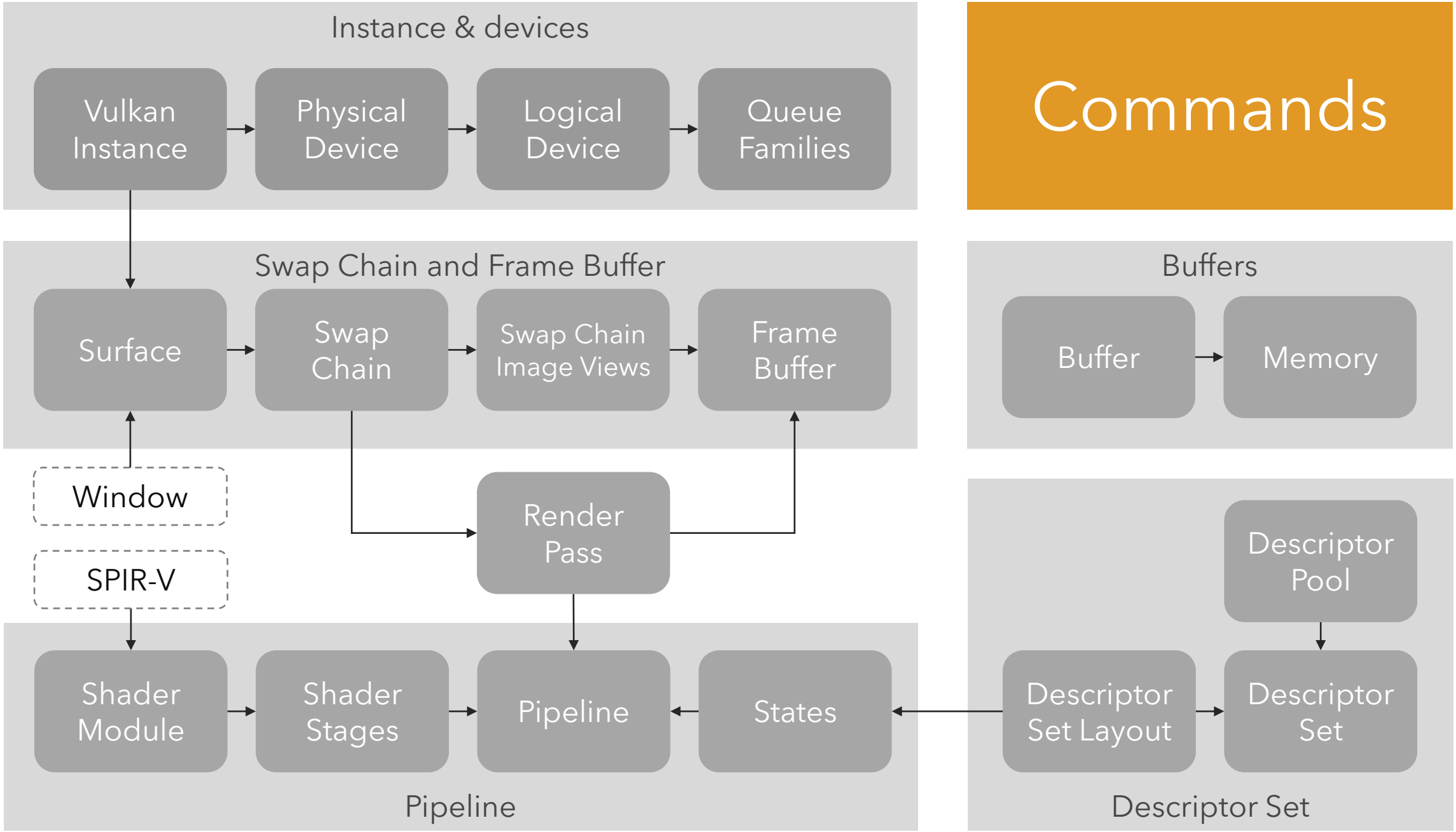

Instance & devices
Vulkan Instance
Physical Device
Logical Device
Queue Families
Swap Chain and Frame Buffer
Surface
Swap Chain
Swap Chain Image Views
Frame Buffer
Buffers
Buffer
Memory
Window
SPIR-V
Render Pass
Descriptor Pool
Shader Module
Shader Stages
Pipeline
States
Descriptor Set Layout
Descriptor Set
Pipeline
Descriptor Set

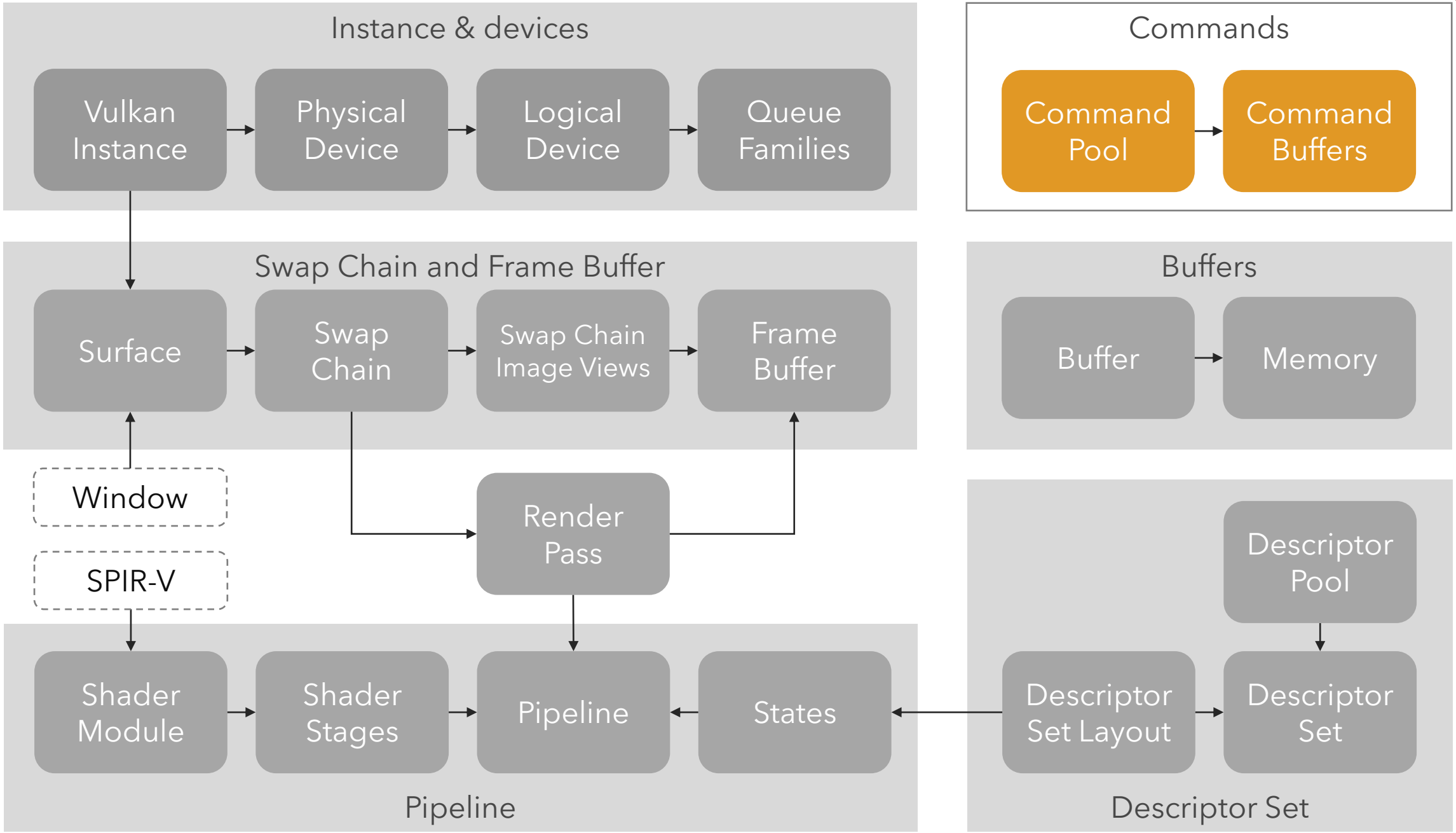

Instance & devices
Vulkan Instance
Physical Device
Logical Device
Queue Families
Swap Chain and Frame Buffer
Surface
Swap Chain
Swap Chain Image Views
Frame Buffer
Buffers
Buffer
Memory
Window
SPIR-V
Render Pass
Descriptor Pool
Shader Module
Specify compiled SPIR-V bytecode size and data
States
Descriptor Set Layout
Descriptor Set
Pipeline
Descriptor Set

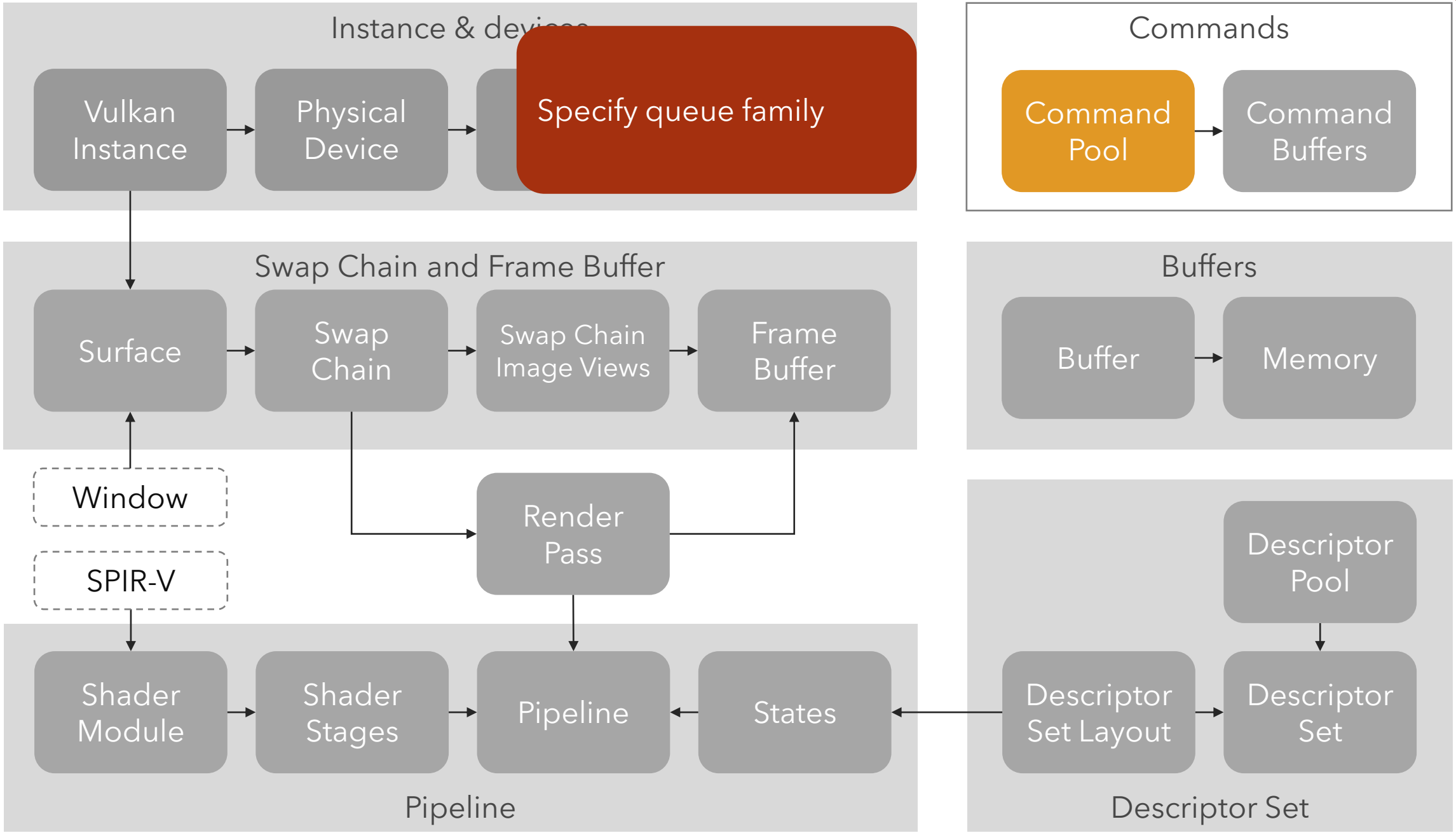

Instance & devices
Vulkan Instance
Physical Device
Logical Device
Queue Families
Swap Chain and Frame Buffer
Surface
Swap Chain
Swap Chain Image Views
Frame Buffer
Buffers
Buffer
Memory
Specify shader module
Specify stage
Specify entry point: e.g “main”
Descriptor Pool
Shader Module
Shader Stages
Pipeline
States
Descriptor Set Layout
Descriptor Set
Pipeline
Descriptor Set

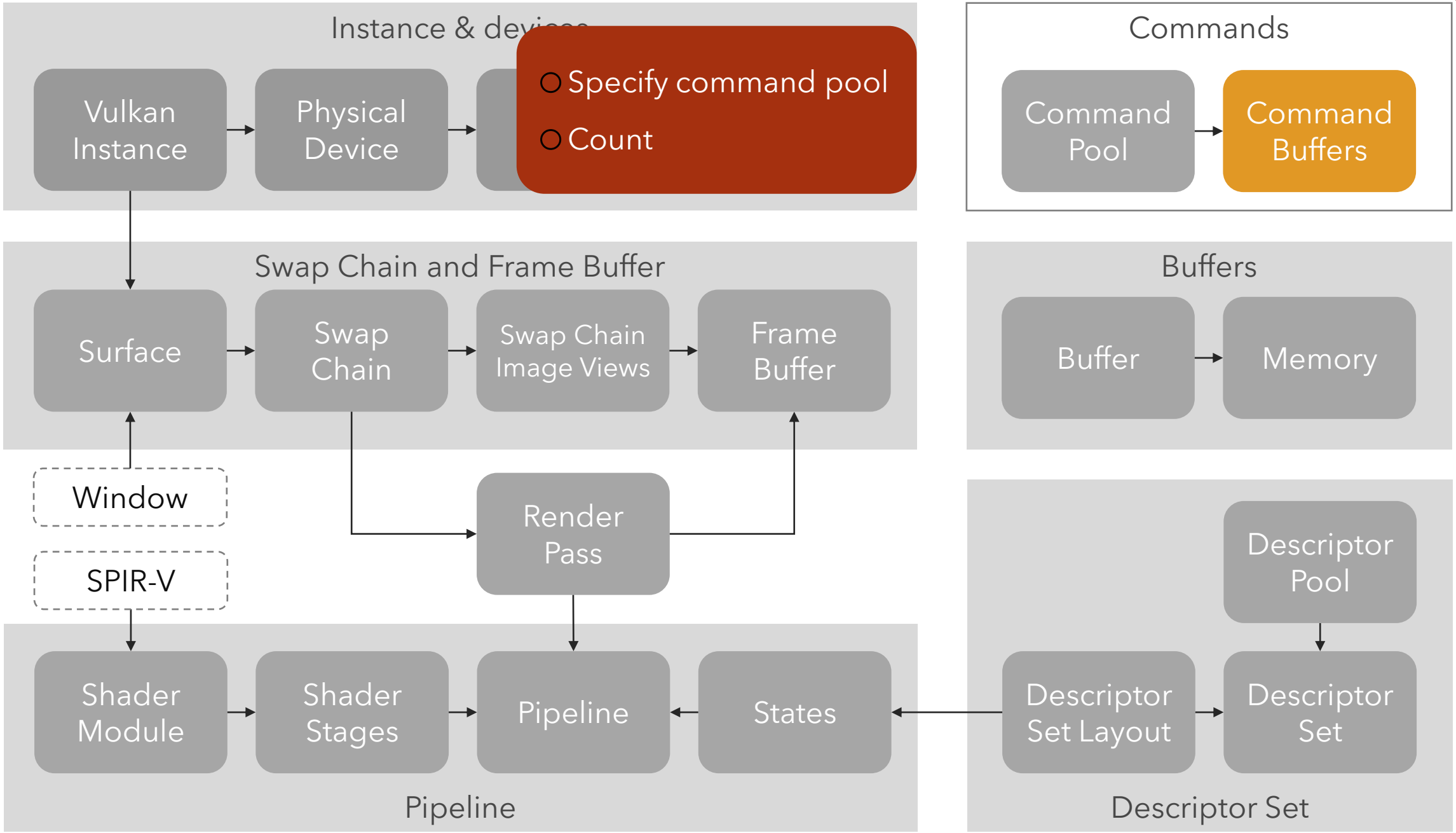
- Specify shader stages
- Specify input assembly states: binding description and attributes
- Specify topology: lines, triangles, points
- specify rasterization states: polygon mode (fill/wireframe), culling, depth
- Specify viewport, sampling, and blending configurations, dynamic states
- Specify descriptor set layouts
- Specify rendering pass

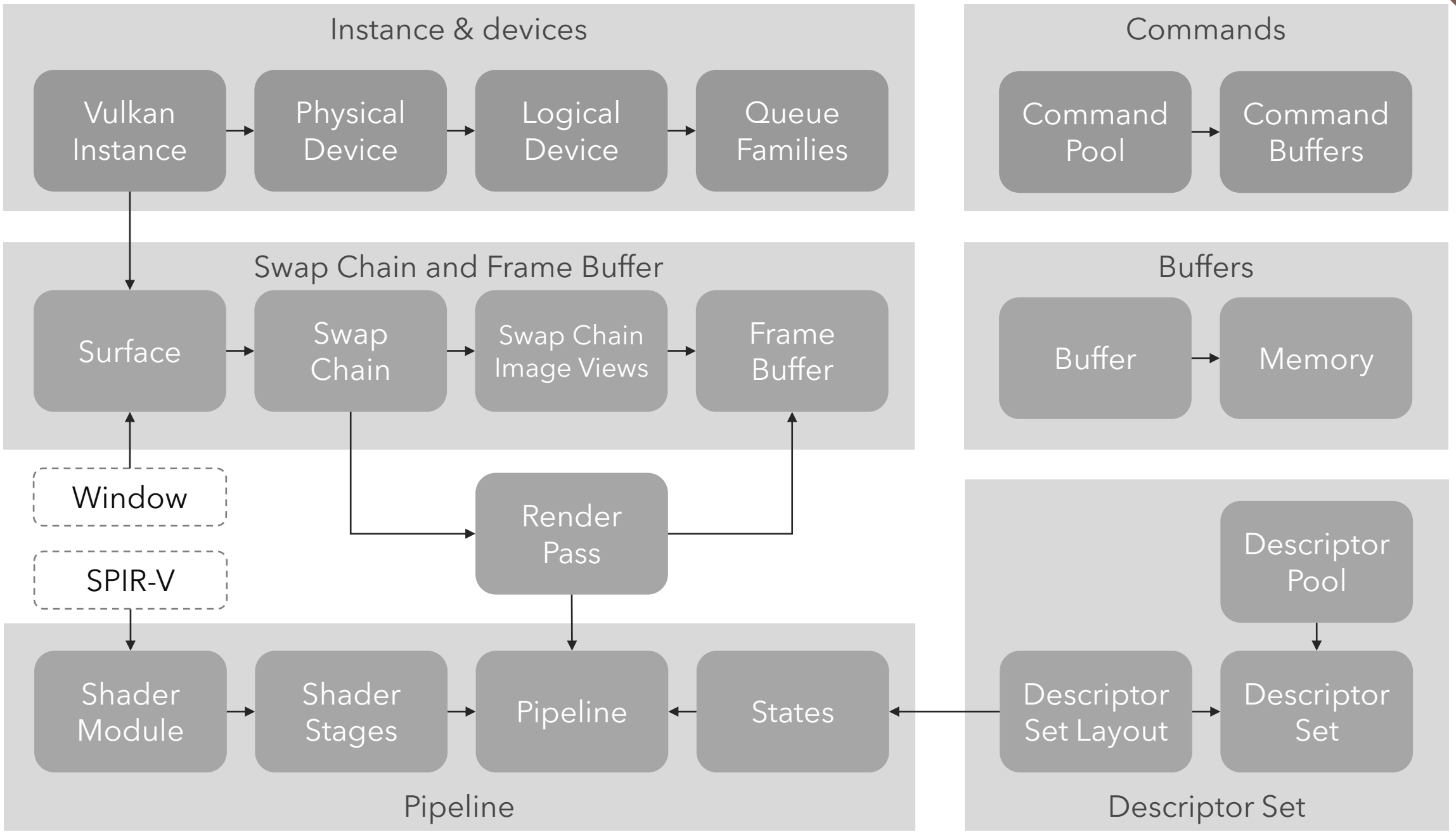


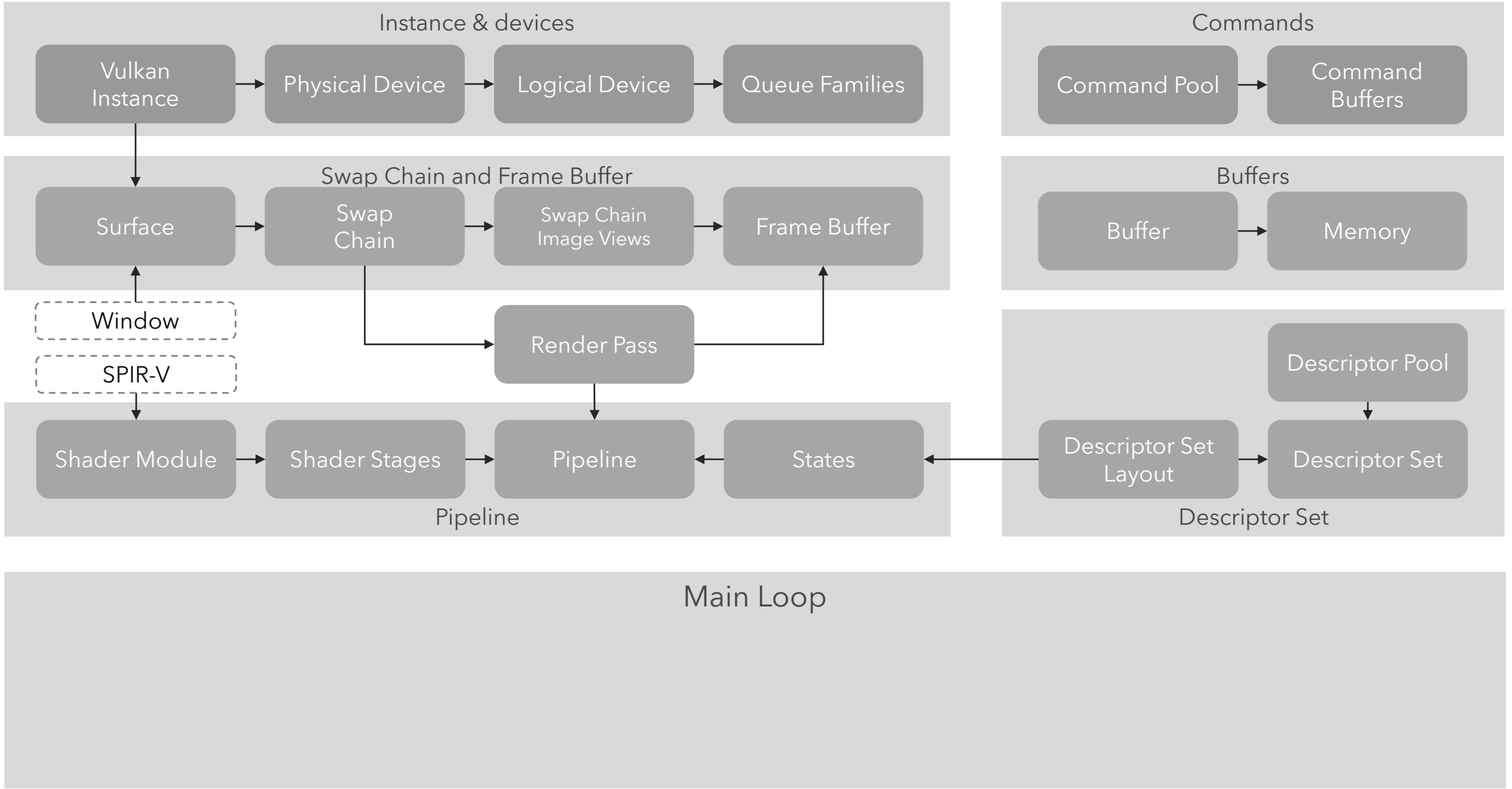

Instance & devices
Vulkan Instance
Physical Device
Logical Device
Queue Families
Commands
Swap Chain and Frame Buffer
Surface
Swap Chain
Swap Chain Image Views
Frame Buffer
Buffers
Buffer
Memory
Window
SPIR-V
Render Pass
Descriptor Pool
Shader Module
Shader Stages
Pipeline
States
Descriptor Set Layout
Descriptor Set
Pipeline
Descriptor Set

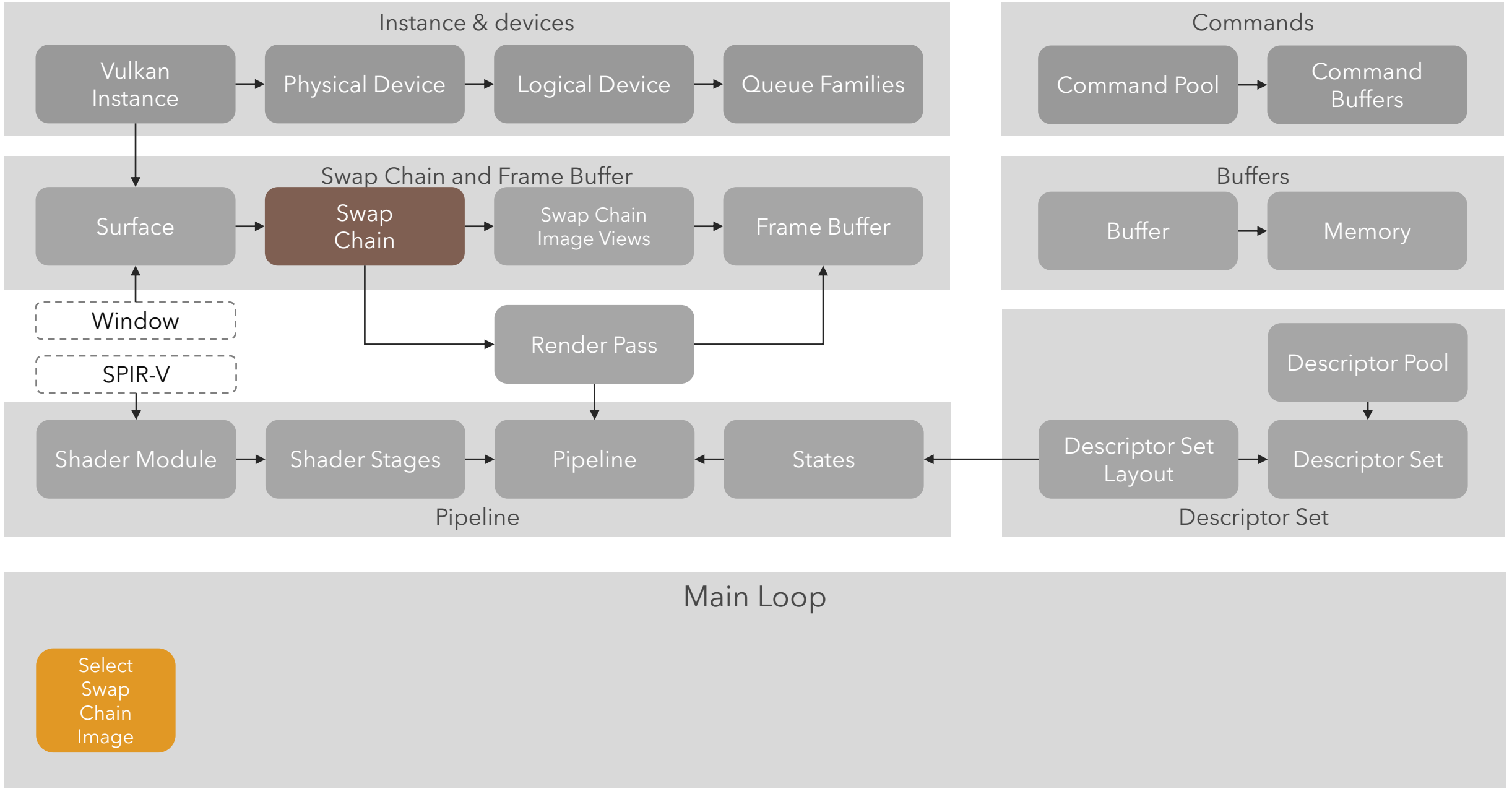

Instance & devices
Vulkan Instance
Physical Device
Logical Device
Queue Families
Commands
Command Pool
Command Buffers
Swap Chain and Frame Buffer
Surface
Swap Chain
Swap Chain Image Views
Frame Buffer
Buffers
Buffer
Memory
Window
SPIR-V
Render Pass
Descriptor Pool
Shader Module
Shader Stages
Pipeline
States
Descriptor Set Layout
Descriptor Set
Pipeline
Descriptor Set

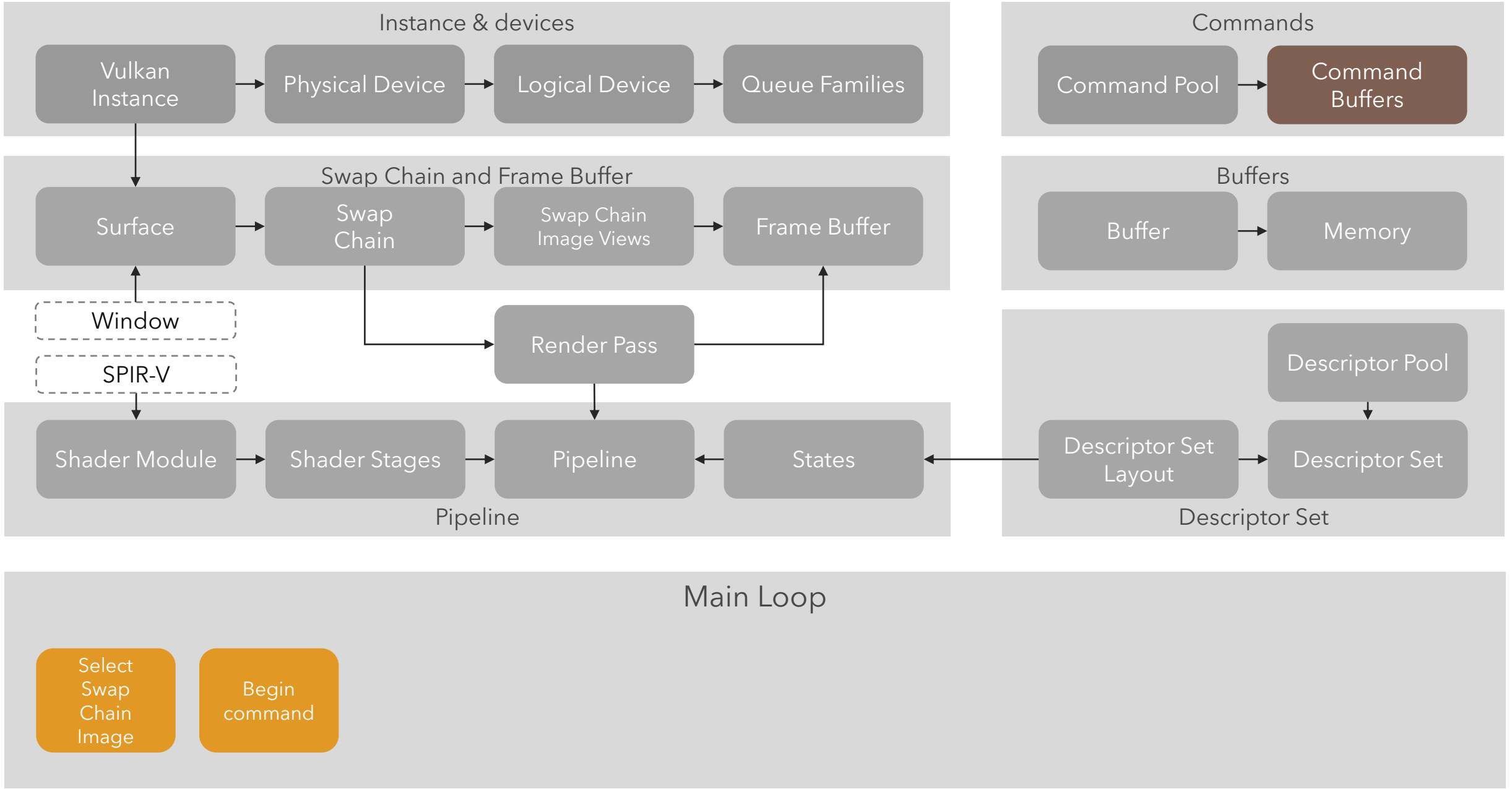

Instance & devices
Vulkan Instance
Physical Device
Specify queue family
Commands
Command Pool
Command Buffers
Swap Chain and Frame Buffer
Surface
Swap Chain
Swap Chain Image Views
Frame Buffer
Buffers
Buffer
Memory
Window
SPIR-V
Render Pass
Descriptor Pool
Shader Module
Shader Stages
Pipeline
States
Descriptor Set Layout
Descriptor Set
Pipeline
Descriptor Set

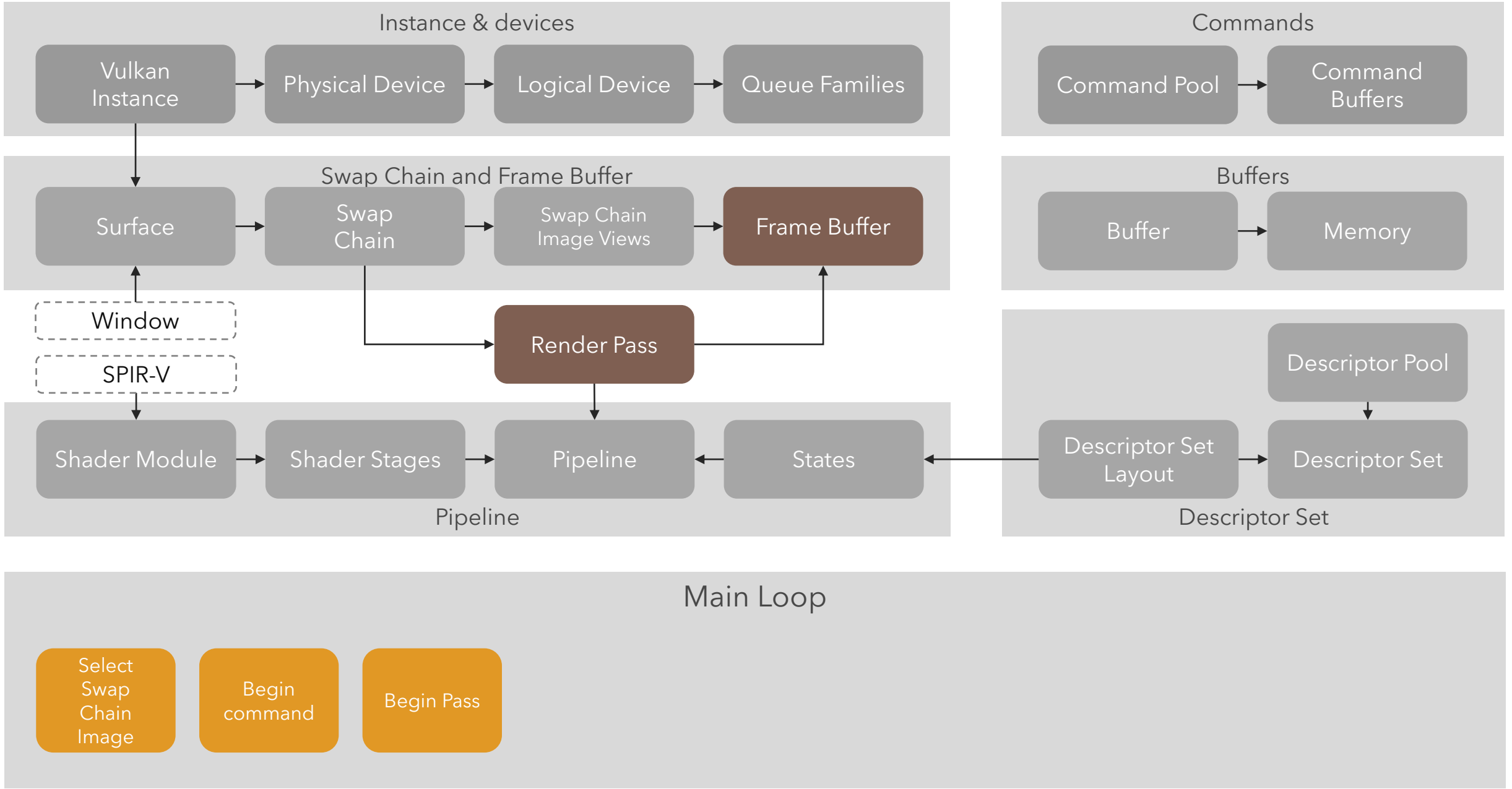

Instance & devices
Vulkan Instance
Physical Device
Specify command pool
Count
Commands
Command Pool
Command Buffers
Swap Chain and Frame Buffer
Surface
Swap Chain
Swap Chain Image Views
Frame Buffer
Buffers
Buffer
Memory
Window
SPIR-V
Render Pass
Descriptor Pool
Shader Module
Shader Stages
Pipeline
States
Descriptor Set Layout
Descriptor Set
Pipeline
Descriptor Set

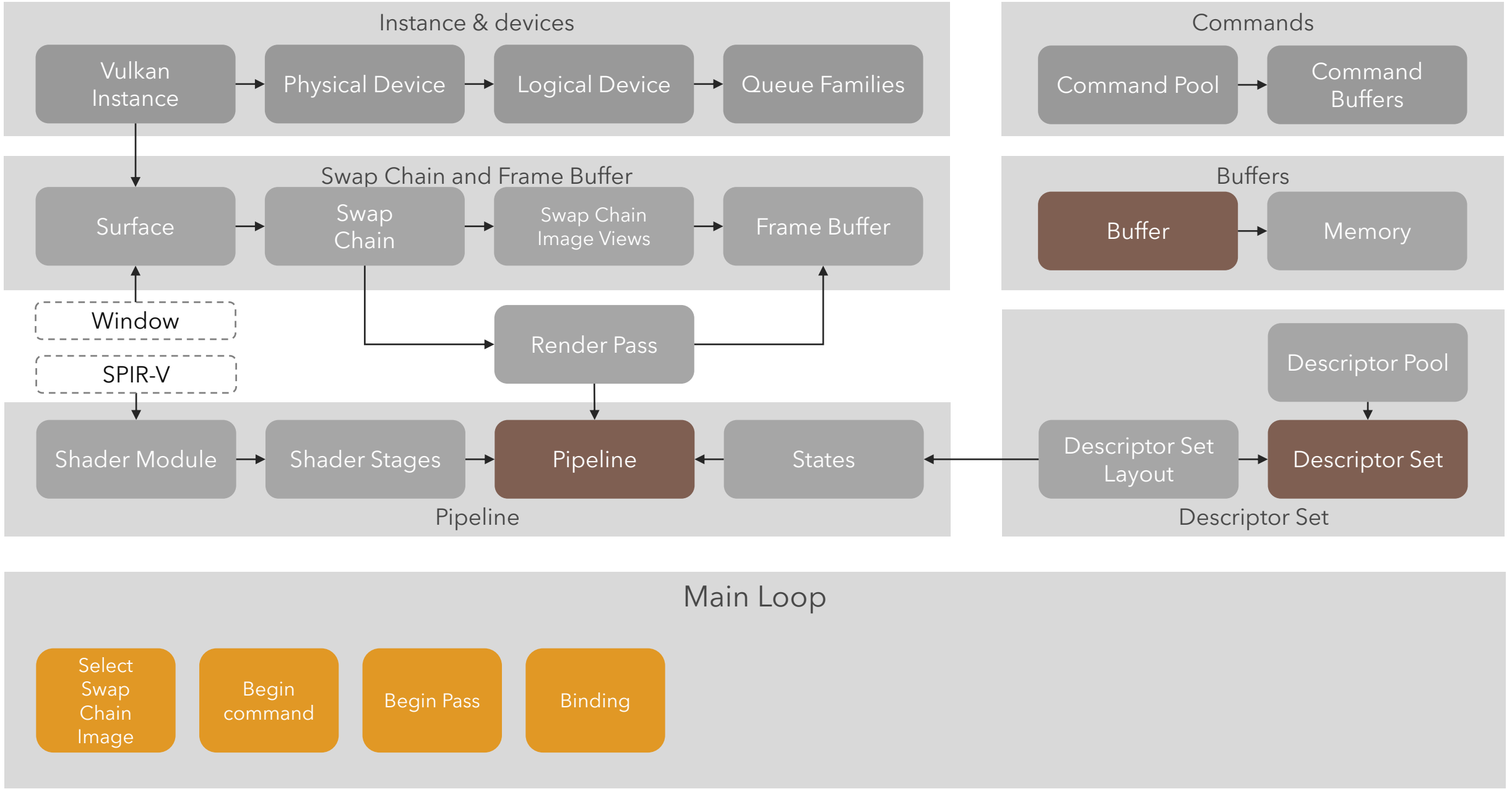

Instance & devices
Vulkan Instance
Physical Device
Logical Device
Queue Families
Commands
Command Pool
Command Buffers
Swap Chain and Frame Buffer
Surface
Swap Chain
Swap Chain Image Views
Frame Buffer
Buffers
Buffer
Memory
Window
SPIR-V
Render Pass
Descriptor Pool
Shader Module
Shader Stages
Pipeline
States
Descriptor Set Layout
Descriptor Set
Pipeline
Descriptor Set

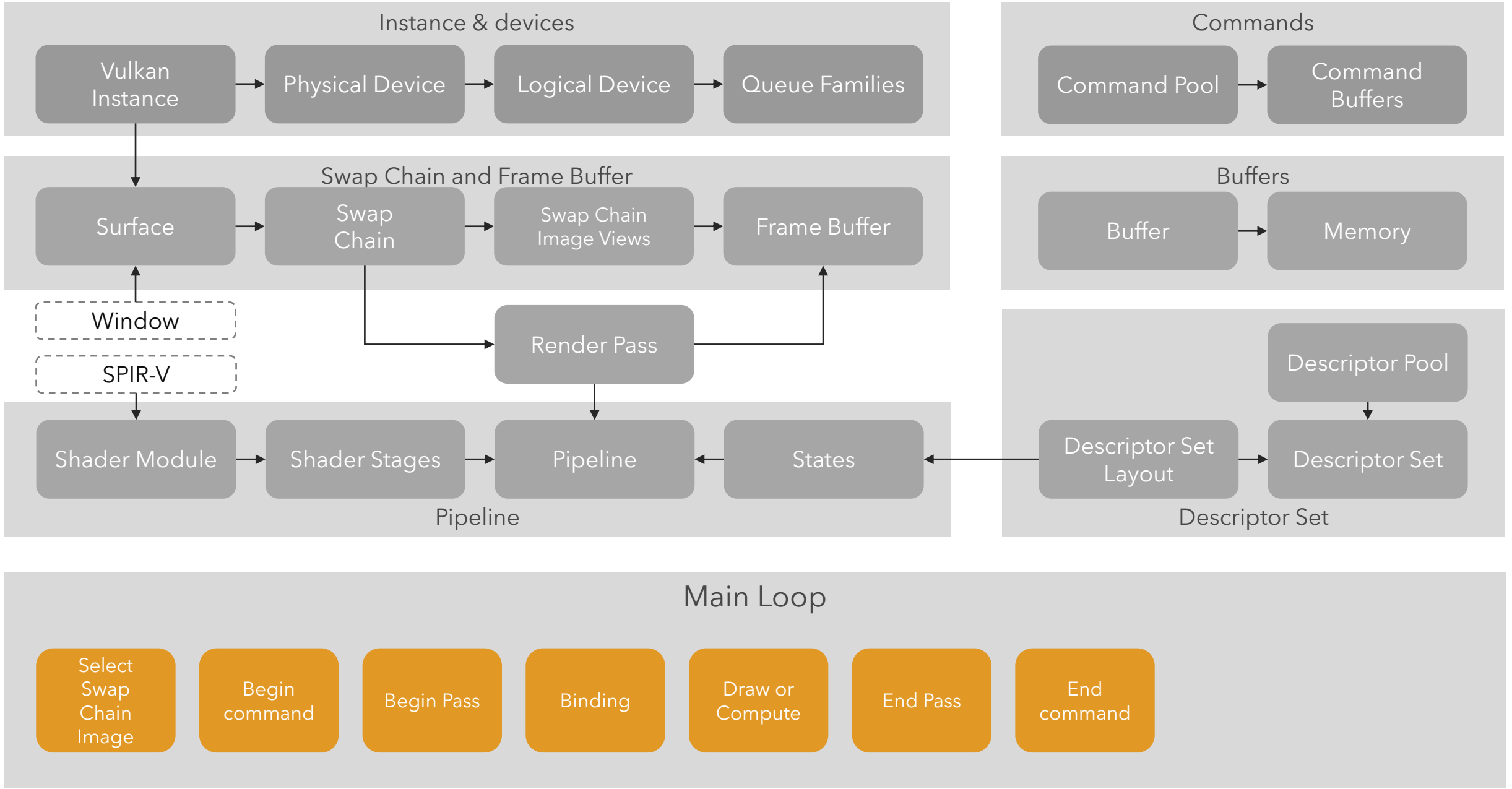

Instance & devices
Vulkan Instance
Physical Device
Logical Device
Queue Families
Commands
Command Pool
Command Buffers
Swap Chain and Frame Buffer
Surface
Swap Chain
Swap Chain Image Views
Frame Buffer
Buffers
Buffer
Memory
Window
SPIR-V
Render Pass
Descriptor Pool
Shader Module
Shader Stages
Pipeline
States
Descriptor Set Layout
Descriptor Set
Pipeline
Descriptor Set
Main Loop

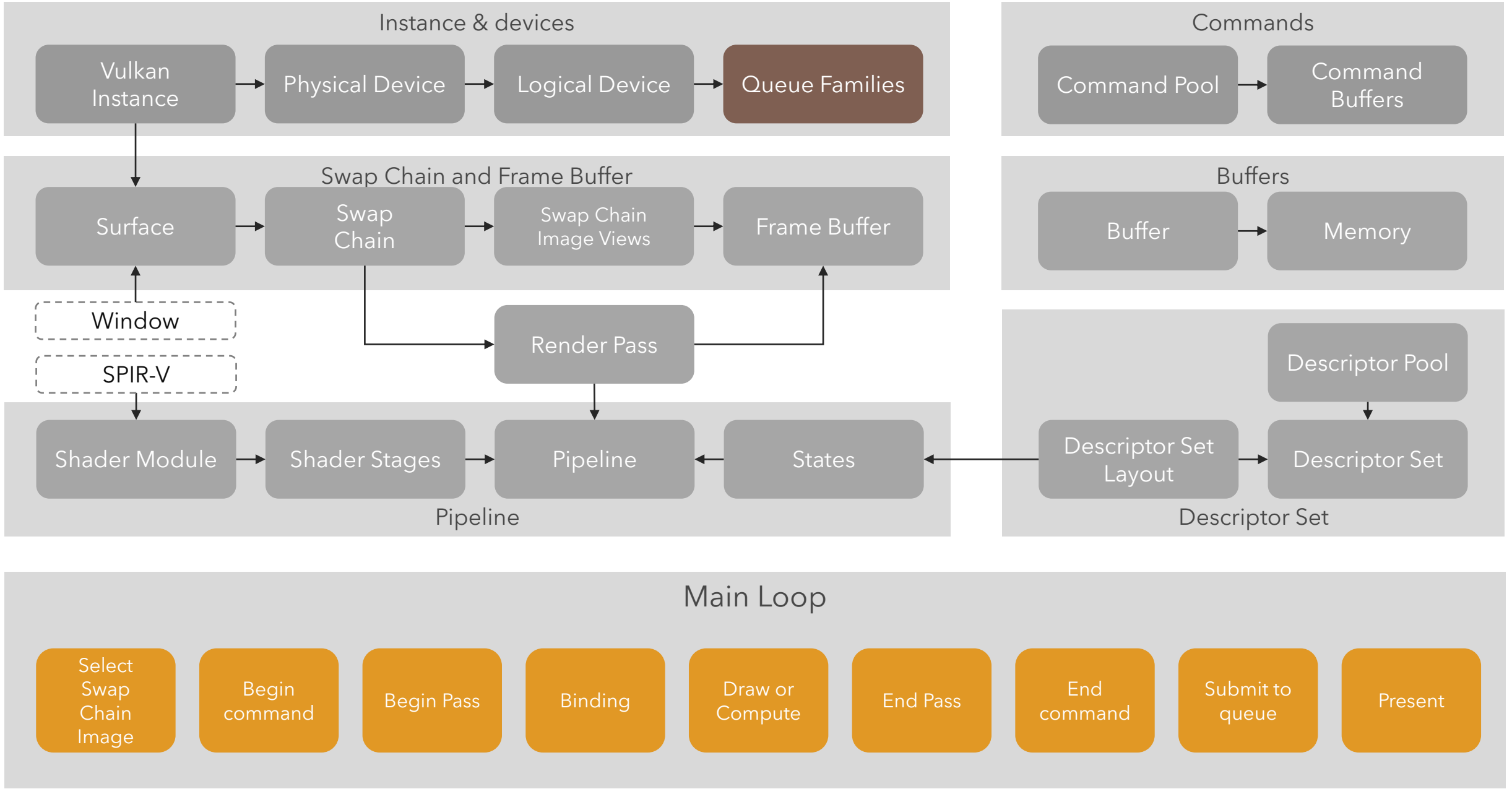

Instance & devices
Vulkan Instance
Physical Device
Logical Device
Queue Families
Commands
Command Pool
Command Buffers
Swap Chain and Frame Buffer
Surface
Swap Chain
Swap Chain Image Views
Frame Buffer
Buffers
Buffer
Memory
Window
SPIR-V
Render Pass
Descriptor Pool
Shader Module
Shader Stages
Pipeline
States
Descriptor Set Layout
Descriptor Set
Pipeline
Descriptor Set
Main Loop
Select Swap Chain Image

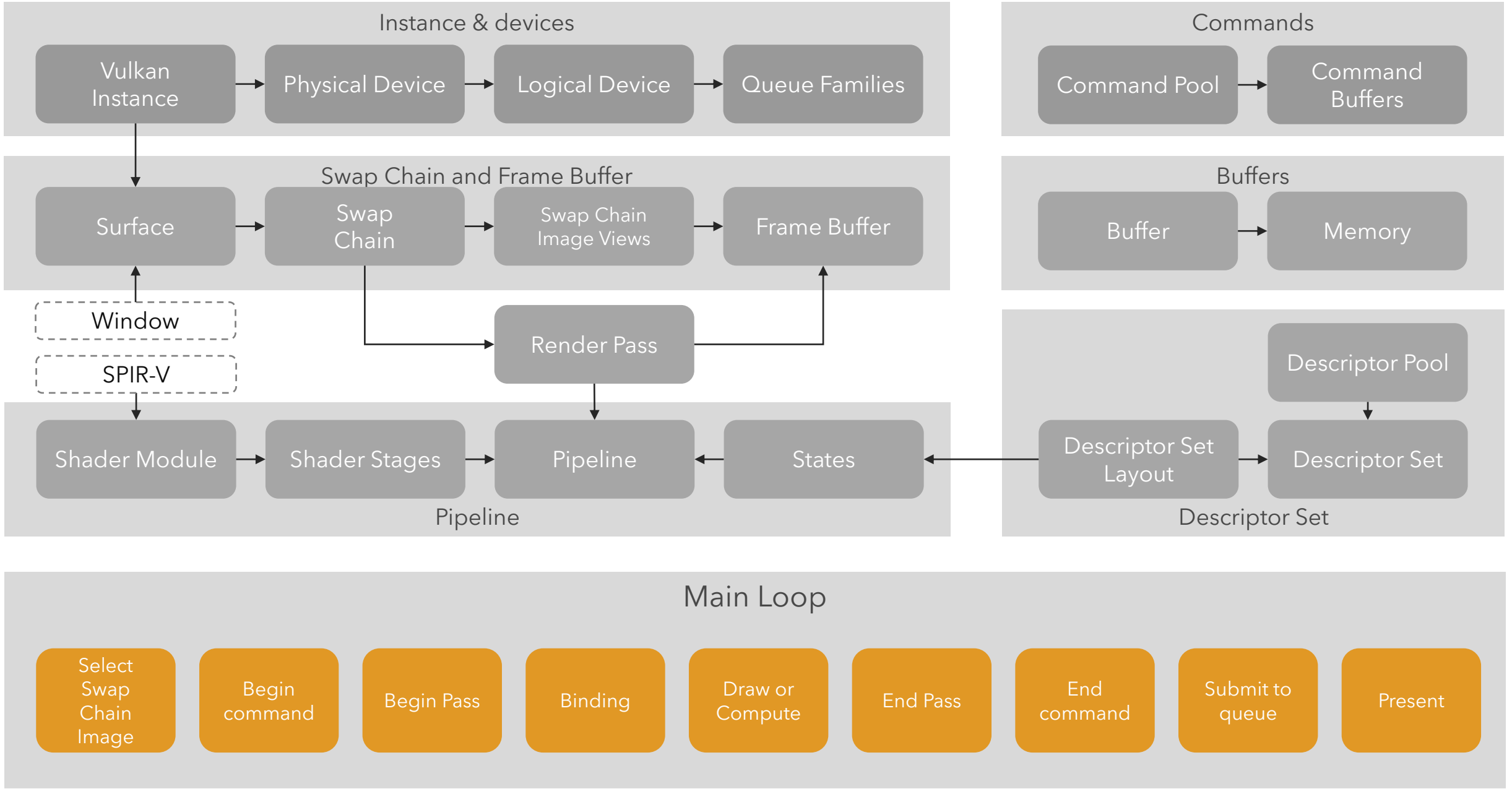

Instance & devices
Vulkan Instance
Physical Device
Logical Device
Queue Families
Commands
Command Pool
Command Buffers
Swap Chain and Frame Buffer
Surface
Swap Chain
Swap Chain Image Views
Frame Buffer
Buffers
Buffer
Memory
Window
SPIR-V
Render Pass
Descriptor Pool
Shader Module
Shader Stages
Pipeline
States
Descriptor Set Layout
Descriptor Set
Pipeline
Descriptor Set
Main Loop
Select Swap Chain Image
Begin command

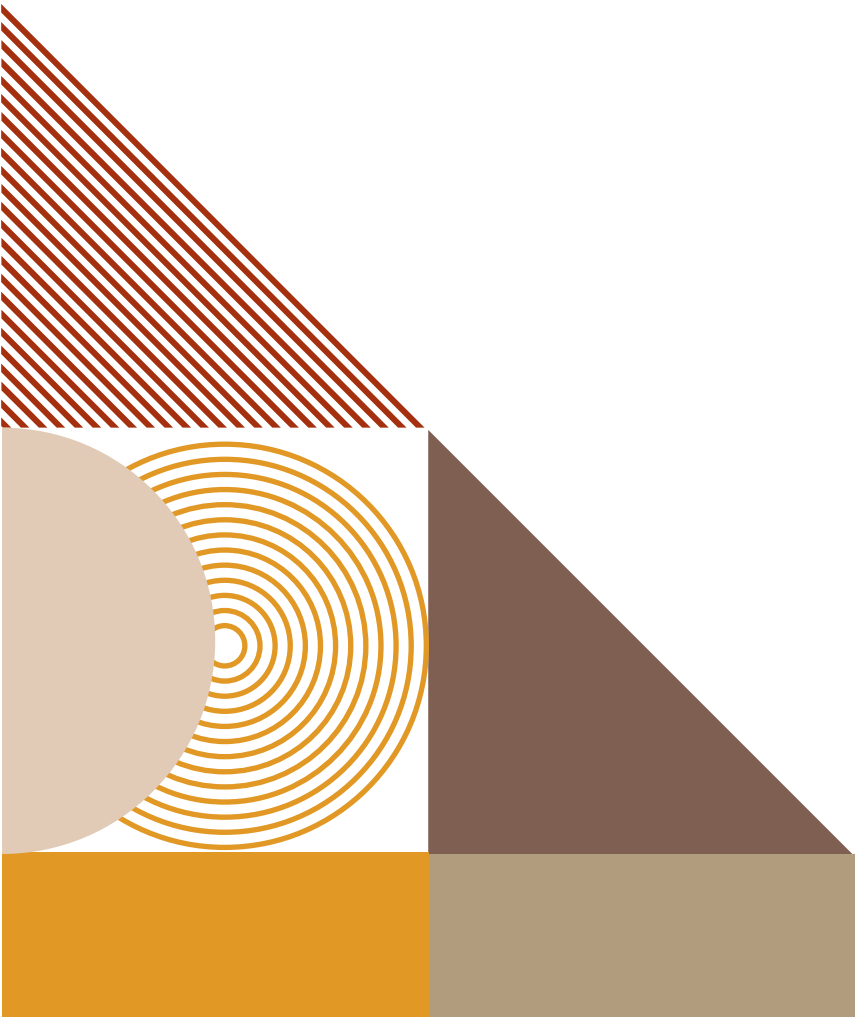

Instance & devices
Vulkan Instance
Physical Device
Logical Device
Queue Families
Commands
Command Pool
Command Buffers
Swap Chain and Frame Buffer
Surface
Swap Chain
Swap Chain Image Views
Frame Buffer
Buffers
Buffer
Memory
Window
SPIR-V
Render Pass
Descriptor Pool
Shader Module
Shader Stages
Pipeline
States
Descriptor Set Layout
Descriptor Set
Pipeline
Descriptor Set
Main Loop
Select Swap Chain Image
Begin command
Begin Pass


Instance & devices
Vulkan Instance
Physical Device
Logical Device
Queue Families
Commands
Command Pool
Command Buffers
Swap Chain and Frame Buffer
Surface
Swap Chain
Swap Chain Image Views
Frame Buffer
Buffers
Buffer
Memory
Window
SPIR-V
Render Pass
Descriptor Pool
Shader Module
Shader Stages
Pipeline
States
Descriptor Set Layout
Descriptor Set
Pipeline
Descriptor Set
Main Loop
Select Swap Chain Image
Begin command
Begin Pass
Binding


Instance & devices
Vulkan Instance
Physical Device
Logical Device
Queue Families
Commands
Command Pool
Command Buffers
Swap Chain and Frame Buffer
Surface
Swap Chain
Swap Chain Image Views
Frame Buffer
Buffers
Buffer
Memory
Window
SPIR-V
Render Pass
Descriptor Pool
Shader Module
Shader Stages
Pipeline
States
Descriptor Set Layout
Descriptor Set
Pipeline
Descriptor Set
Main Loop
Select Swap Chain Image
Begin command
Begin Pass
Binding
Draw or Compute
End Pass
End command


Instance & devices
Vulkan Instance
Physical Device
Logical Device
Queue Families
Commands
Command Pool
Command Buffers
Swap Chain and Frame Buffer
Surface
Swap Chain
Swap Chain Image Views
Frame Buffer
Buffers
Buffer
Memory
Window
SPIR-V
Render Pass
Descriptor Pool
Shader Module
Shader Stages
Pipeline
States
Descriptor Set Layout
Descriptor Set
Pipeline
Descriptor Set
Main Loop
Select Swap Chain Image
Begin command
Begin Pass
Binding
Draw or Compute
End Pass
End command
Submit to queue
Present

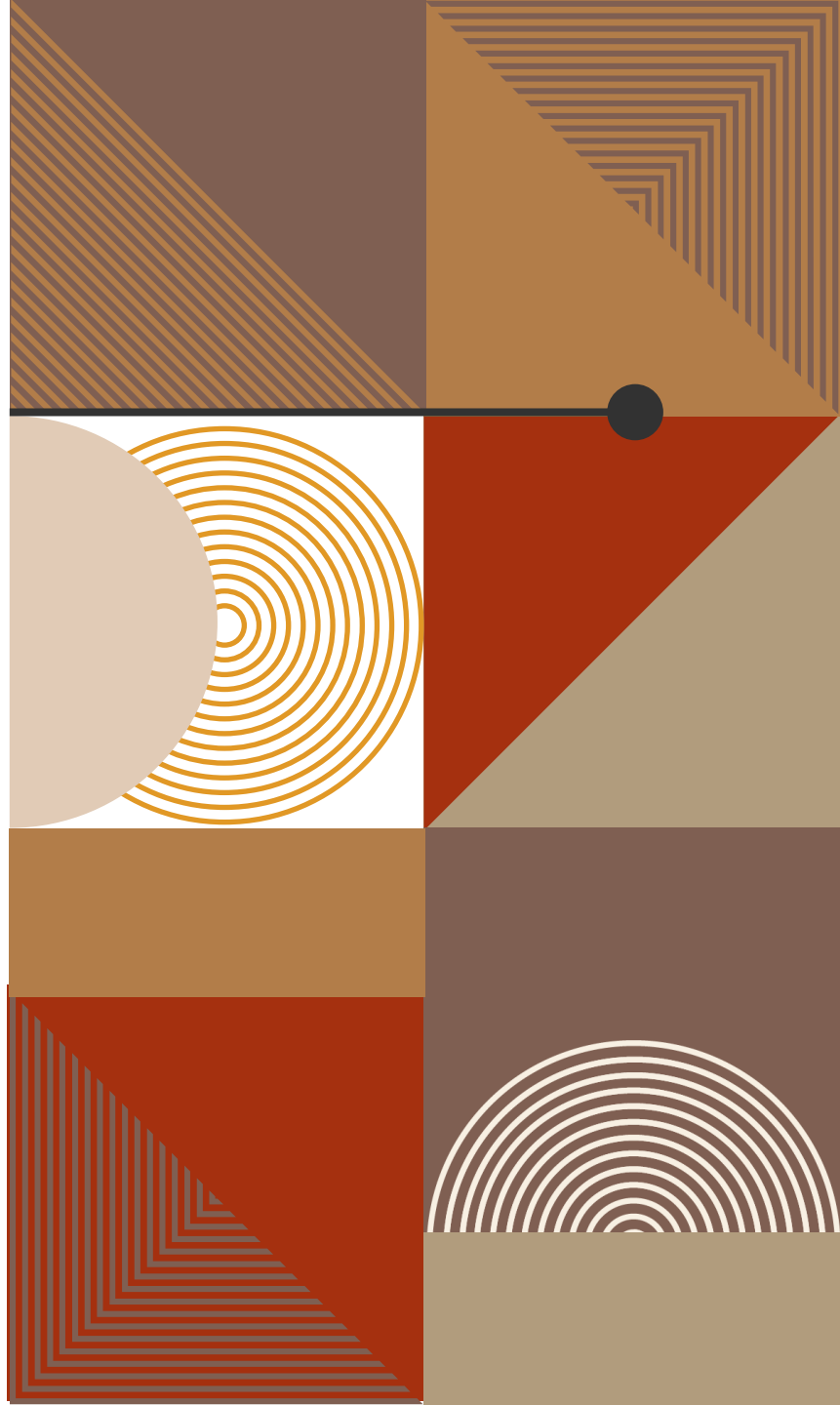

Instance & devices
Vulkan Instance
Physical Device
Logical Device
Queue Families
Commands
Command Pool
Command Buffers
Swap Chain and Frame Buffer
Surface
Swap Chain
Swap Chain Image Views
Frame Buffer
Buffers
Buffer
Memory
Window
SPIR-V
Render Pass
Descriptor Pool
Shader Module
Shader Stages
Pipeline
States
Descriptor Set Layout
Descriptor Set
Pipeline
Descriptor Set
Main Loop
Select Swap Chain Image
Begin command
Begin Pass
Binding
Draw or Compute
End Pass
End command
Submit to queue
Present



GLSL OVERVIEW

SPIR-V AND GLSL

- GLSL is a shading language made originally for OpenGL
- Syntax similar to C++
- Can be compiled to SPIR-V and fed into Vulkan



INTRODUCTION

GLSL is a very simple language

- Has scalar types: `float, int, bool`
- Has vector types:

  `vec2, vec3, vec4, ivec2, ivec3, ivec4`
- Has matrix types: `mat2, mat3, mat4`
- Texture sampling: `sampler1D, sampler2D, sampler3D`
- Constructors are easy

```
Vec3 positions = vec3(1, 0.4, 5.2);
```

COMPONENTS AND SWIZZLING

- Can access components of vector/matrix types:

```
position[0]
position.xyzw, position.rgba, position.strq
```

- Swizzling:

```
positions.x, positions.yz, positions.xzxy
```

OPERATIONS AND BUILT-INS

- Operators:
  - Usual arithmetic: `+, -, *, /`
- Lots of useful functions:
  - `mix, norm, dot, clamp, max, min, sqrt, abs, pow, length, reflect, sin`
- Variables:
  - [required] `gl_Position`: output the position in the vertex shader

GLSL: QUALIFIERS

- Qualifiers: How to send data to the shader program
- Layout: from the specified set layout or pipeline binding descriptions

```
layout (location = 0) in vec3 postion;
layout(binding = 0) uniform UniformBufferObject {
    mat4 model;
    mat4 view;
    mat4 proj;
} ubo;
```

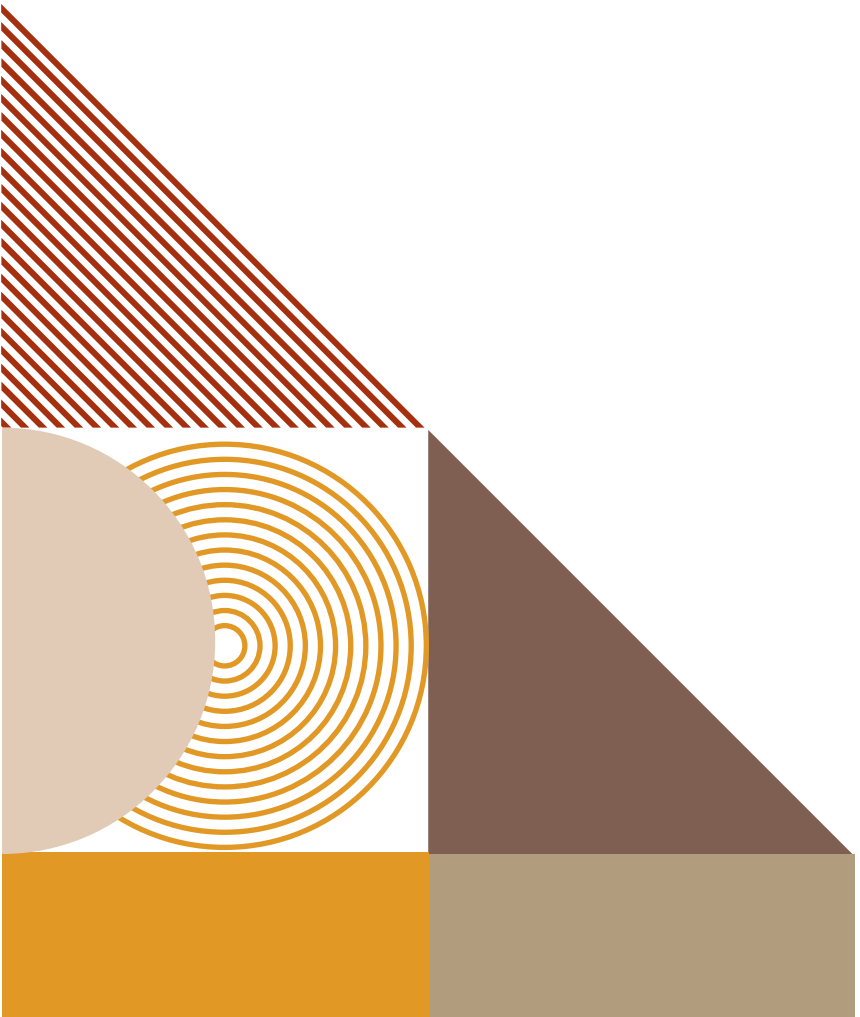
- `In, out`: copy variables into and out of the shader
- Used to communicate between shading stages

GLSL: LAYOUTS

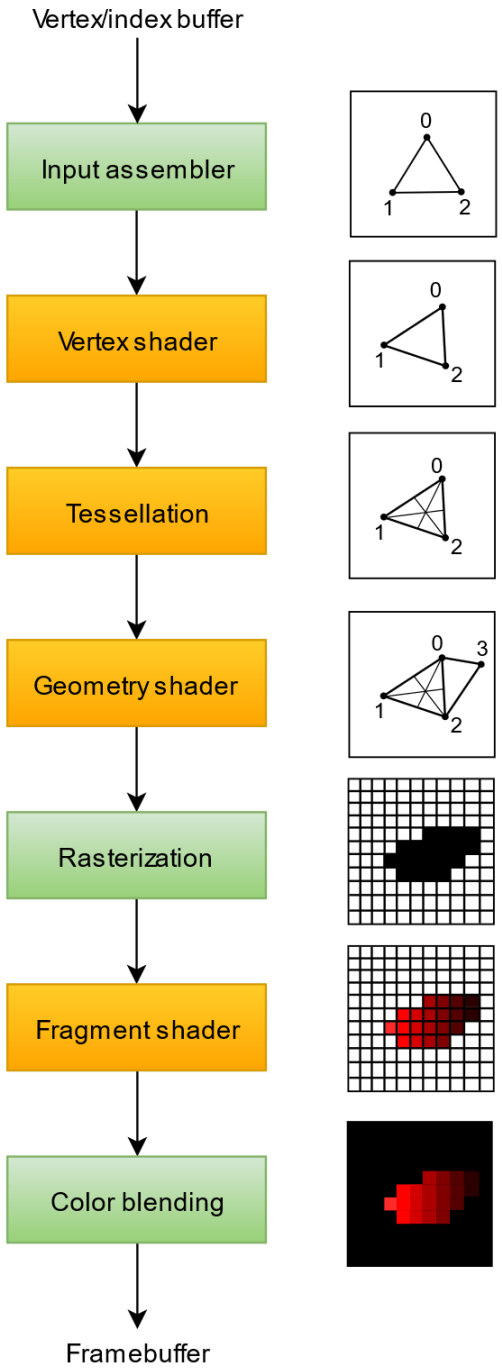
```
// texture
layout(binding = 0) uniform sampler2D mySampler;

// storage buffers
layout(std140, binding = 1) readonly buffer ParticleSSBOIn {
   Particle myParticles[];
};
```

GLSL: LAYOUTS

```
// push constants
layout(push_constant) uniform myPushConstants
{
  vec4 variable1;
  float variable2;
};

// shared memory
shared int[32] shared_ints;
```



SHADING STAGES



THE GRAPHICS PIPELINE

- Programmable and fixed functions state
- Fixed functions are explicitly specified while creating the pipeline
- The programmable stages:
  - Vertex Shader
  - Tessellation Shader
  - Geometry Shader
  - Fragment Shader

VERTEX SHADER

- Receives a single vertex
- Transform from object space to screen space
- Output: transformed position, other vertex attributes

VERTEX SHADER

```
#version 450

// per vertex input
layout(location = 0) in vec4 inPosition;

// outputs to the geometry shader
layout(location = 0) out vec4 gsPosition;

void main() {
    gsPosition = inPosition;
}
```

GEOMETRY SHADER

- Input: a single primitive
- Output: zero or more primitives

GEOMETRY SHADER

```
#version 450
layout(points) in;
layout(triangle_strip, max_vertices = 64) out;

layout(location = 0) in vec4 gsInPosition[];
layout(location = 0) out vec3 fsColor;

void main() {
// ...
gl_Position = // ...;
    fsColor = // ...;
    EmitVertex();
EndPrimitive();
}
```

FRAGMENT SHADER

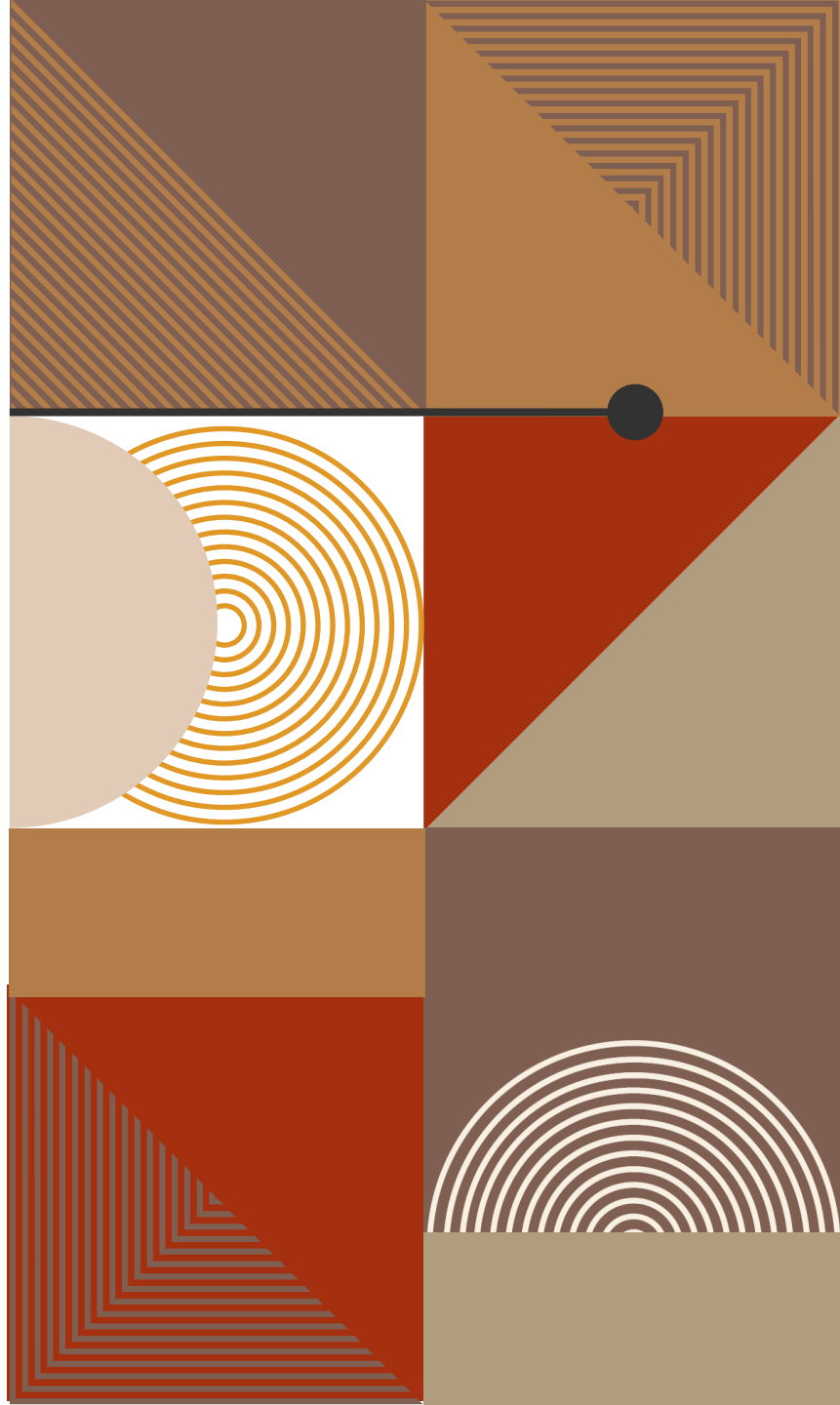
- The stage after a primitive is rasterized
- Contains interpolated per-vertex attributes
- Output: color and depth

FRAGMENT SHADER

```
#version 450
// input from previous shading stage
layout(location = 0) in vec4 fsColor;

// output color
layout(location = 0) out vec4 outColor;

void main() {
    outColor = fsColor;
}
```

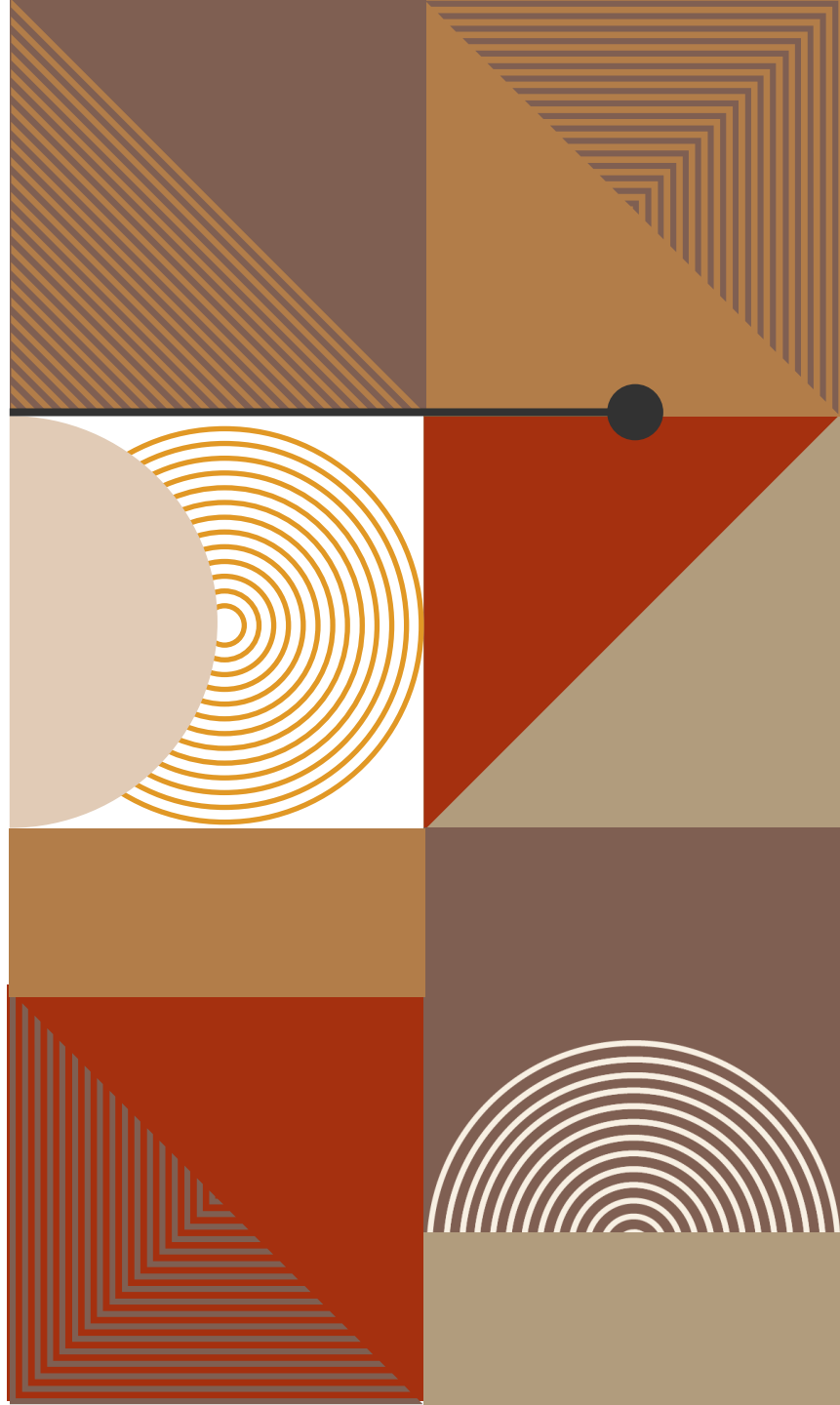
THE COMPUTE PIPELINE

- Besides the graphics pipeline, it is possible to do arbitrary computation on the GPU using the compute shader
- Must explicitly define the number of threads to execute and the workgroup size
- Unlike the graphics shaders, the compute shader does not have well-defined input/output values

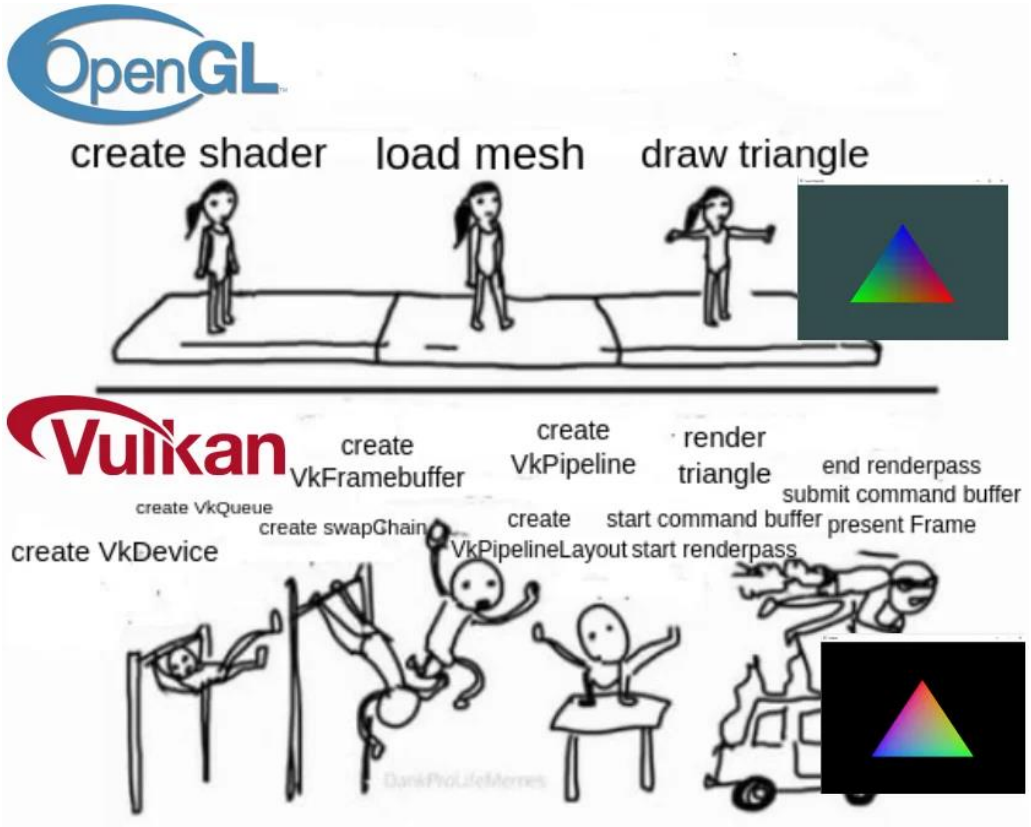


RESOURCES

- vulkan-tutorial.com
- Vulkan-Guide by Khronos
- Vulkan Specification
- Vulkan Documentation



THANK YOU


OpenGL
create shader
load mesh
draw triangle
Vulkan
create VkDevice
create VkQueue
create swapGhain
create VkFramebuffer
create VkPipelineLayout
create VkPipeline
start renderpass
render triangle
start command buffer
end renderpass
submit command buffer
present Frame