Vulkanised 2023

The 5th Vulkan Developer Conference
Munich, Germany / February 7–9

An Introduction to Vulkan

Johannes Unterguggenberger

TU Wien, Huawei

Vulkanised 2023

The 5th Vulkan Developer Conference
Munich, Germany / February 7–9

Elias Kristmann
TU Wien

Andreas Wiesinger
TU Wien

Viktoria Pogrzebacz
TU Wien

Annalena Ulschmid
TU Wien

Lukas Lipp
TU Wien

Johannes Unterguggenberger
TU Wien, Huawei

Platinum Sponsors: AMD, arm, Google, LunarG, Khronos Group, Samsung

Schedule

PART 1:

Setup	Lecture	Coding Session
10 min	**20** min	**90** min
Starts at 09:00	Starts at 09:10	Starts at 09:30

PART 2:

Lecture	Coffee Break	Coding Session
15 min	**25** min	**80** min
Starts at 11:00	Starts at 11:15	Starts at 11:40

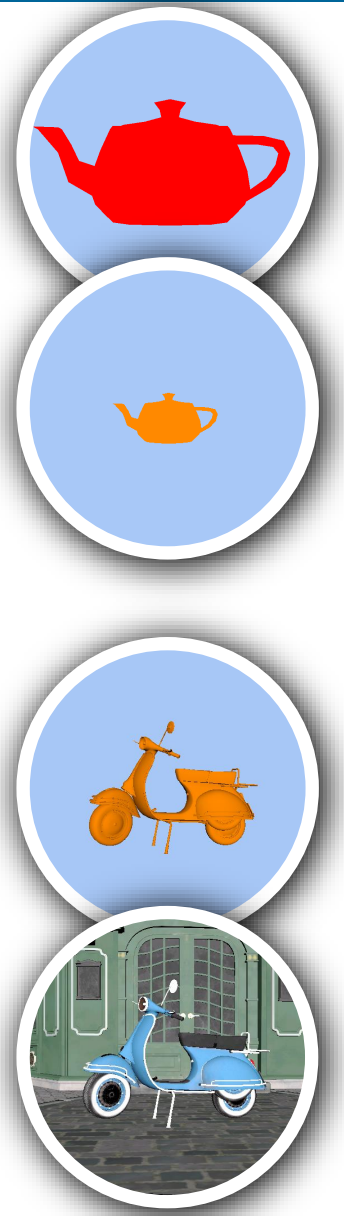
Lunch Break 13:00 – 14:00

PART 3:

Lecture	Coding Session	Coffee Break
15 min	**65** min	**30** min
Starts at 14:00	Starts at 14:15	Starts at 15:20

PART 4:

Lecture	Coding Session	Closing
20 min	**70** min	**10** min
Starts at 15:50	Starts at 16:10	Starts at 17:20



Schedule

PART 1:

Setup	Lecture	Coding Session
10 min	**20** min	**90** min
Starts at 09:00	Starts at 09:10	Starts at 09:30

PART 2:

Lecture	Coffee Break	Coding Session
15 min	**25** min	**80** min
Starts at 11:00	Starts at 11:15	Starts at 11:40

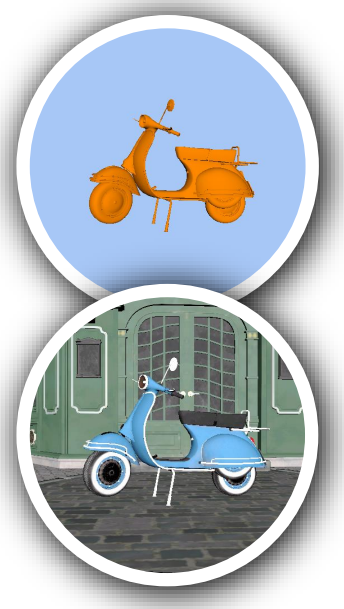
Lunch Break 13:00 – 14:00

PART 3:

Lecture	Coding Session	Coffee Break
15 min	**65** min	**30** min
Starts at 14:00	Starts at 14:15	Starts at 15:20

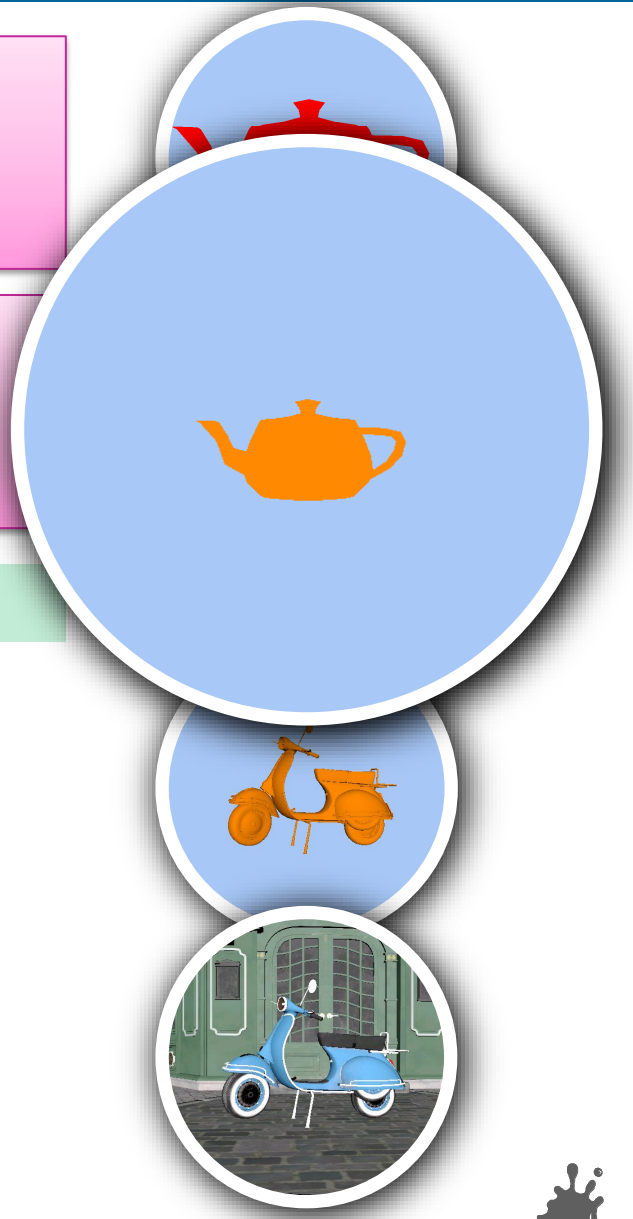
PART 4:

Lecture	Coding Session	Closing
20 min	**70** min	**10** min
Starts at 15:50	Starts at 16:10	Starts at 17:20



Schedule

PART 1:	Setup **10** min Starts at 09:00	Lecture **20** min Starts at 09:10	Coding Session **90** min Starts at 09:30
PART 2:	Lecture **15** min Starts at 11:00	Coffee Break **25** min Starts at 11:15	Coding Session **80** min Starts at 11:40
	Lunch Break 13:00 – 14:00		
PART 3:	Lecture **15** min Starts at 14:00	Coding Session **65** min Starts at 14:15	Coffee Break **30** min Starts at 15:20
PART 4:	Lecture **20** min Starts at 15:50	Coding Session **70** min Starts at 16:10	Closing **10** min Starts at 17:20



Schedule

PART 1:

Setup	Lecture	Coding Session
10 min	**20** min	**90** min
Starts at 09:00	Starts at 09:10	Starts at 09:30

PART 2:

Lecture	Coffee Break	Coding Session
15 min	**25** min	**80** min
Starts at 11:00	Starts at 11:15	Starts at 11:40

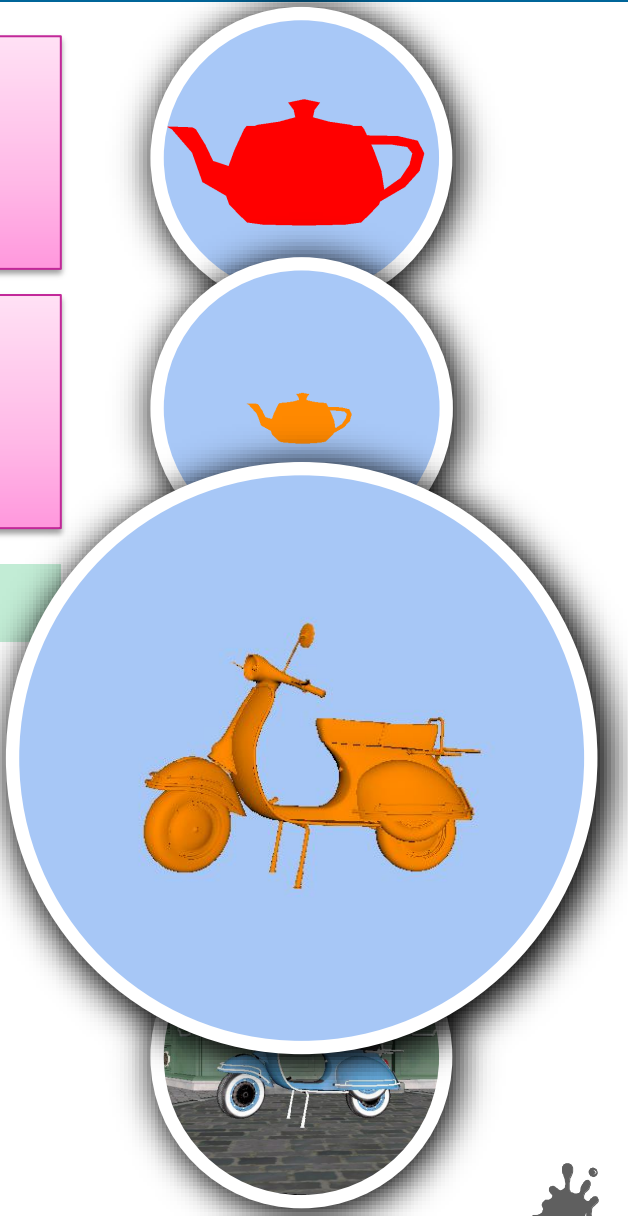
Lunch Break 13:00 – 14:00

PART 3:

Lecture	Coding Session	Coffee Break
15 min	**65** min	**30** min
Starts at 14:00	Starts at 14:15	Starts at 15:20

PART 4:

Lecture	Coding Session	Closing
20 min	**70** min	**10** min
Starts at 15:50	Starts at 16:10	Starts at 17:20



Schedule

PART 1:

Setup	Lecture	Coding Session
10 min	**20** min	**90** min
Starts at 09:00	Starts at 09:10	Starts at 09:30

PART 2:

Lecture	Coffee Break	Coding Session
15 min	**25** min	**80** min
Starts at 11:00	Starts at 11:15	Starts at 11:40

Lunch Break 13:00 – 14:00

PART 3:

Lecture	Coding Session	Coffee Break
15 min	**65** min	**30** min
Starts at 14:00	Starts at 14:15	Starts at 15:20

PART 4:

Lecture	Coding Session	Closing
20 min	**70** min	**10** min
Starts at 15:50	Starts at 16:10	Starts at 17:20

Schedule

PART 1:

Setup	Lecture	Coding Session
10 min	**20** min	**90** min
Starts at 09:00	Starts at 09:10	Starts at 09:30

PART 2:

Lecture	Coffee Break	Coding Session
15 min	**25** min	**80** min
Starts at 11:00	Starts at 11:15	Starts at 11:40

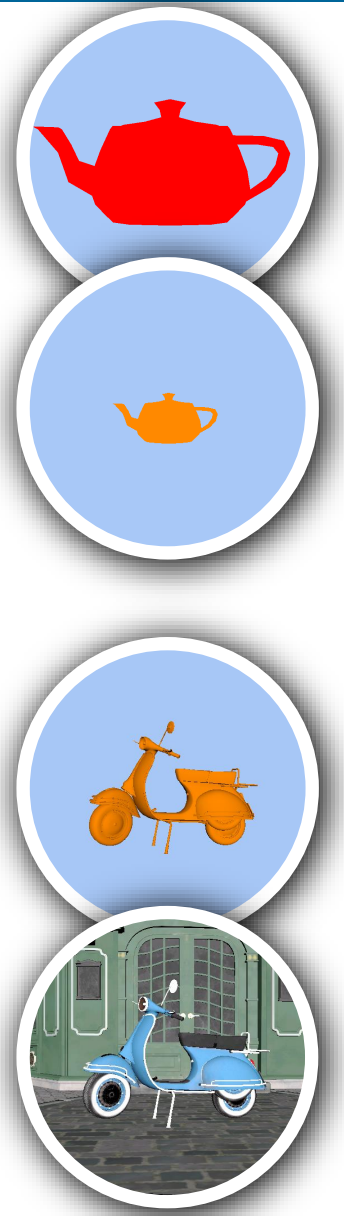
Lunch Break 13:00 – 14:00

PART 3:

Lecture	Coding Session	Coffee Break
15 min	**65** min	**30** min
Starts at 14:00	Starts at 14:15	Starts at 15:20

PART 4:

Lecture	Coding Session	Closing
20 min	**70** min	**10** min
Starts at 15:50	Starts at 16:10	Starts at 17:20



Coding Sessions

During each coding session…

Coding Session
90 min
Starts at
09:30

…you have two options:

1) Code yourself!
(recommended!)

+ assistance
from our team

2) Watch live coding

Code: **obz-chhh-npo**

Setup Instructions

STEP 1

Use our VulkanLaunchpadStarter template

to create a **private**(!) project on **your** GitHub account

STEP 2

Join our "**Vulkanised**" Discord server:

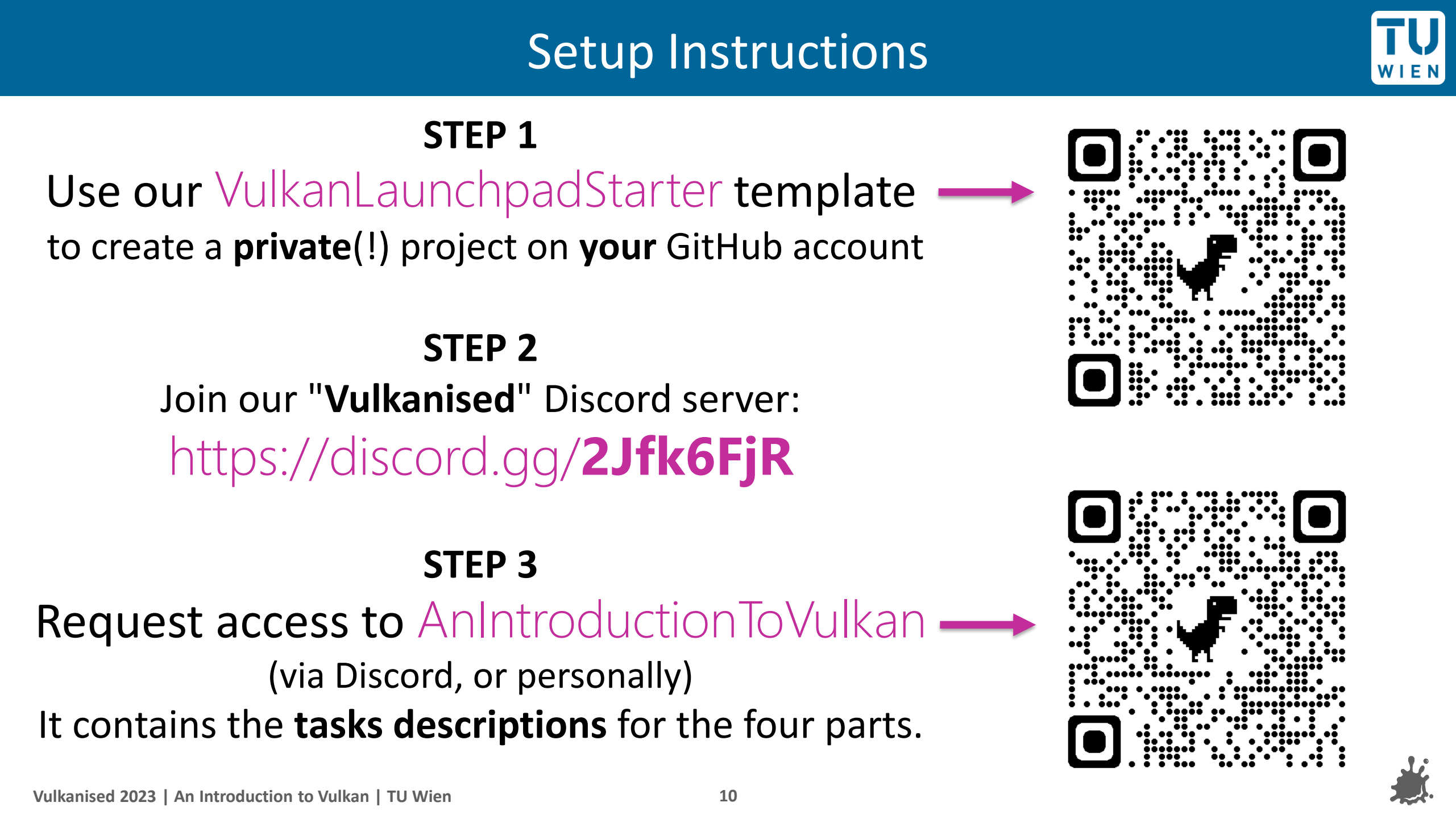
https://discord.gg/**2Jfk6FjR**

STEP 3

Request access to AnIntroductionToVulkan

(via Discord, or personally)

It contains the **tasks descriptions** for the four parts.



Vulkanised 2023

The 5th Vulkan Developer Conference
Munich, Germany / February 7–9

PART 1

- Fundamental Vulkan Handles
- Window System Integration
- The Swap Chain

Platinum Sponsors: AMD arm Google LUNARG KHRONOS GROUP SAMSUNG

Overview

We need to create/get hold of a couple of handles:		
Instance	Vulkan on your system	`VkInstance`
Window Surface	A window of your OS	`VkSurfaceKHR`
Physical Device	A hardware device (GPU)	`VkPhysicalDevice`
Queue	Received commands to be executed on a physical device	`VkQueue`
Logical Device	Main interface to a physical device (active configuration)	`VkDevice`
Swap Chain	Sends images to a monitor, Provides images to render into	`VkSwapchainKHR`

Instance Creation

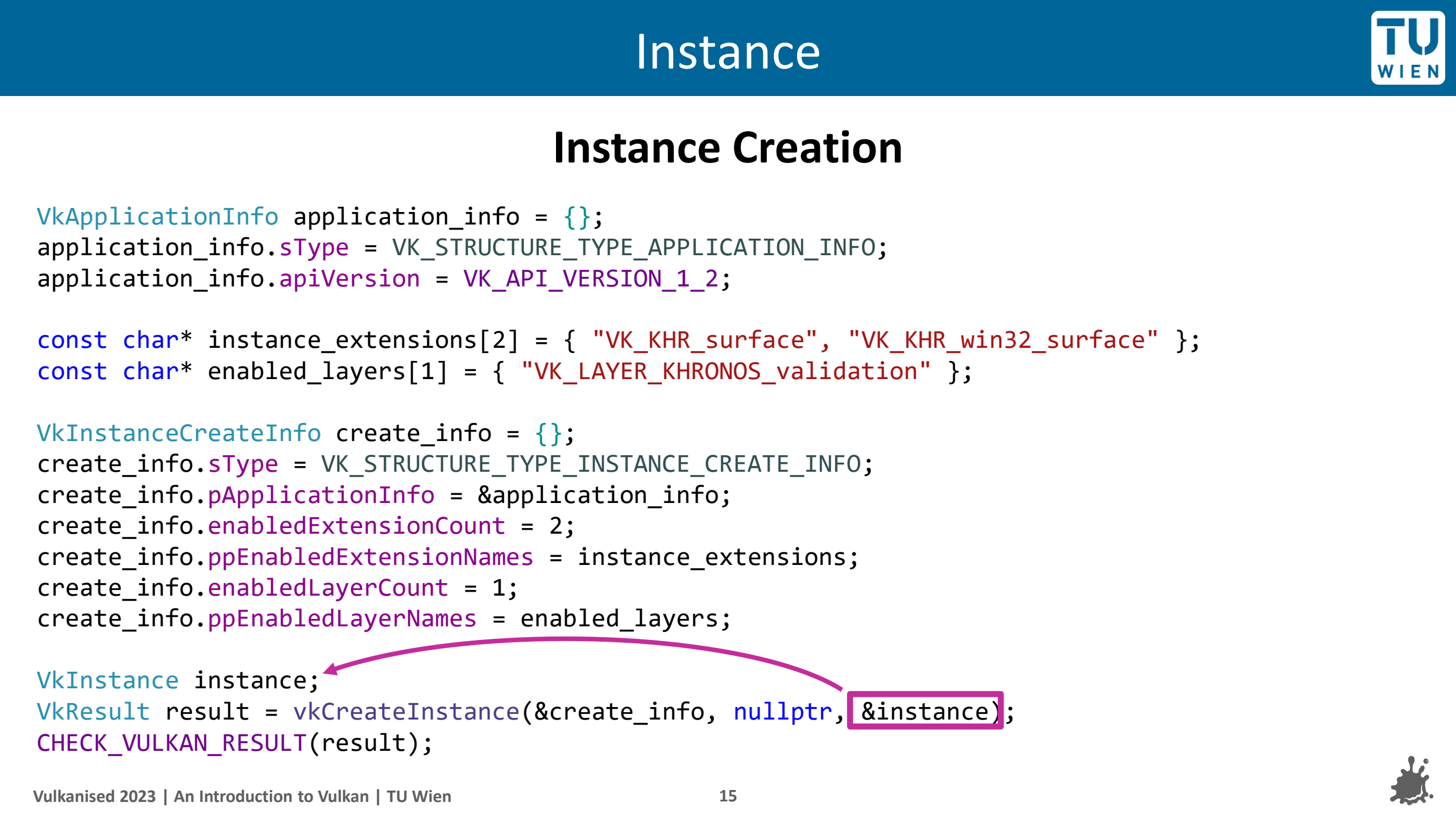
```
VkApplicationInfo application_info = {};
application_info.sType = VK_STRUCTURE_TYPE_APPLICATION_INFO;
application_info.apiVersion = VK_API_VERSION_1_2;

const char* instance_extensions[2] = { "VK_KHR_surface", "VK_KHR_win32_surface" };
const char* enabled_layers[1] = { "VK_LAYER_KHRONOS_validation" };

VkInstanceCreateInfo create_info = {};
create_info.sType = VK_STRUCTURE_TYPE_INSTANCE_CREATE_INFO;
create_info.pApplicationInfo = &application_info;
create_info.enabledExtensionCount = 2;
create_info.ppEnabledExtensionNames = instance_extensions;
create_info.enabledLayerCount = 1;
create_info.ppEnabledLayerNames = enabled_layers;

VkInstance instance;
VkResult result = vkCreateInstance(&create_info, nullptr, &instance);
CHECK_VULKAN_RESULT(result);
```


Instance Creation

```
VkApplicationInfo application_info = {};
application_info.sType = VK_STRUCTURE_TYPE_APPLICATION_INFO;
application_info.apiVersion = VK_API_VERSION_1_2;

const char* instance_extensions[2] = { "VK_KHR_surface", "VK_KHR_win32_surface" };
const char* enabled_layers[1] = { "VK_LAYER_KHRONOS_validation" };

VkInstanceCreateInfo create_info = {};
create_info.sType = VK_STRUCTURE_TYPE_INSTANCE_CREATE_INFO;
create_info.pApplicationInfo = &application_info;
create_info.enabledExtensionCount = 2;
create_info.ppEnabledExtensionNames = instance_extensions;
create_info.enabledLayerCount = 1;
create_info.ppEnabledLayerNames = enabled_layers;

VkInstance instance;
VkResult result = vkCreateInstance(&create_info, nullptr, &instance);
CHECK_VULKAN_RESULT(result);
```

Instance Creation

```
VkApplicationInfo application_info = {};
application_info.sType = VK_STRUCTURE_TYPE_APPLICATION_INFO;
application_info.apiVersion = VK_API_VERSION_1_2;

const char* instance_extensions[2] = { "VK_KHR_surface", "VK_KHR_win32_surface" };
const char* enabled_layers[1] = { "VK_LAYER_KHRONOS_validation" };

VkInstanceCreateInfo create_info = {};
create_info.sType = VK_STRUCTURE_TYPE_INSTANCE_CREATE_INFO;
create_info.pApplicationInfo = &application_info;
create_info.enabledExtensionCount = 2;
create_info.ppEnabledExtensionNames = instance_extensions;
create_info.enabledLayerCount = 1;
create_info.ppEnabledLayerNames = enabled_layers;

VkInstance instance;
VkResult result = vkCreateInstance(&create_info, nullptr, &instance);
CHECK_VULKAN_RESULT(result);
```



Instance Creation

```
VkApplicationInfo application_info = {};
application_info.sType = VK_STRUCTURE_TYPE_APPLICATION_INFO;
application_info.apiVersion = VK_API_VERSION_1_2;

const char* instance_extensions[2] = { "VK_KHR_surface", "VK_KHR_win32_surface" };
const char* enabled_layers[1] = { "VK_LAYER_KHRONOS_validation" };

VkInstanceCreateInfo create_info = {};
create_info.sType = VK_STRUCTURE_TYPE_INSTANCE_CREATE_INFO;
create_info.pApplicationInfo = &application_info;
create_info.enabledExtensionCount = 2;
create_info.ppEnabledExtensionNames = instance_extensions;
create_info.enabledLayerCount = 1;
create_info.ppEnabledLayerNames = enabled_layers;

VkInstance instance;
VkResult result = vkCreateInstance(&create_info, nullptr, &instance);
CHECK_VULKAN_RESULT(result);
```


Instance Creation

```
VkApplicationInfo application_info = {};
application_info.sType = VK_STRUCTURE_TYPE_APPLICATION_INFO;
application_info.apiVersion = VK_API_VERSION_1_2;

const char* instance_extensions[2] = { "VK_KHR_surface", "VK_KHR_win32_surface" };
const char* enabled_layers[1] = { "VK_LAYER_KHRONOS_validation" };

VkInstanceCreateInfo create_info = {};
create_info.sType = VK_STRUCTURE_TYPE_INSTANCE_CREATE_INFO;
create_info.pApplicationInfo = &application_info;
create_info.enabledExtensionCount = 2;
create_info.ppEnabledExtensionNames = instance_extensions;
create_info.enabledLayerCount = 1;
create_info.ppEnabledLayerNames = enabled_layers;

VkInstance instance;
VkResult result = vkCreateInstance(&create_info, nullptr, &instance);
CHECK_VULKAN_RESULT(result);
```


Instance Creation

```
VkApplicationInfo application_info = {};
application_info.sType = VK_STRUCTURE_TYPE_APPLICATION_INFO;
application_info.apiVersion = VK_API_VERSION_1_2;

const char* instance_extensions[2] = { "VK_KHR_surface", "VK_KHR_win32_surface" };
const char* enabled_layers[1] = { "VK_LAYER_KHRONOS_validation" };

VkInstanceCreateInfo create_info = {};
create_info.sType = VK_STRUCTURE_TYPE_INSTANCE_CREATE_INFO;
create_info.pApplicationInfo = &application_info;
create_info.enabledExtensionCount = 2;
create_info.ppEnabledExtensionNames = instance_extensions;
create_info.enabledLayerCount = 1;
create_info.ppEnabledLayerNames = enabled_layers;

VkInstance instance;
VkResult result = vkCreateInstance(&create_info, nullptr, &instance);
CHECK_VULKAN_RESULT(result);
```


Instance Creation

```
VkApplicationInfo application_info = {};
application_info.sType = VK_STRUCTURE_TYPE_APPLICATION_INFO;
application_info.apiVersion = VK_API_VERSION_1_2;

const char* instance_extensions[2] = { "VK_KHR_surface", "VK_KHR_win32_surface" };
const char* enabled_layers[1] = { "VK_LAYER_KHRONOS_validation" };

VkInstanceCreateInfo create_info = {};
create_info.sType = VK_STRUCTURE_TYPE_INSTANCE_CREATE_INFO;
create_info.pApplicationInfo = &application_info;
create_info.enabledExtensionCount = 2;
create_info.ppEnabledExtensionNames = instance_extensions;
create_info.enabledLayerCount = 1;
create_info.ppEnabledLayerNames = enabled_layers;

VkInstance instance;
VkResult result = vkCreateInstance(&create_info, nullptr, &instance);
CHECK_VULKAN_RESULT(result);
```

Instance Creation

```
VkApplicationInfo application_info = {};
application_info.sType = VK_STRUCTURE_TYPE_APPLICATION_INFO;
application_info.apiVersion = VK_API_VERSION_1_2;

const char* instance_extensions[2] = { "VK_KHR_surface", "VK_KHR_win32_surface" };
const char* enabled_layers[1] = { "VK_LAYER_KHRONOS_validation" };

VkInstanceCreateInfo create_info = {};
create_info.sType = VK_STRUCTURE_TYPE_INSTANCE_CREATE_INFO;
create_info.pApplicationInfo = &application_info;
create_info.enabledExtensionCount = 2;
create_info.ppEnabledExtensionNames = instance_extensions;
create_info.enabledLayerCount = 1;
create_info.ppEnabledLayerNames = enabled_layers;

VkInstance instance;
VkResult result = vkCreateInstance(&create_info, nullptr, &instance);
CHECK_VULKAN_RESULT(result);
```


Instance Creation

```
VkApplicationInfo application_info = {};
application_info.sType = VK_STRUCTURE_TYPE_APPLICATION_INFO;
application_info.apiVersion = VK_API_VERSION_1_2;

const char* instance_extensions[2] = { "VK_KHR_surface", "VK_KHR_win32_surface" };
const char* enabled_layers[1] = { "VK_LAYER_KHRONOS_validation" };

VkInstanceCreateInfo create_info = {};
create_info.sType = VK_STRUCTURE_TYPE_INSTANCE_CREATE_INFO;
create_info.pApplicationInfo = &application_info;
create_info.enabledExtensionCount = 2;
create_info.ppEnabledExtensionNames = instance_extensions;
create_info.enabledLayerCount = 1;
create_info.ppEnabledLayerNames = enabled_layers;

VkInstance instance;
VkResult result = vkCreateInstance(&create_info, nullptr, &instance);
CHECK_VULKAN_RESULT(result);
```


Instance Creation

```
VkApplicationInfo application_info = {};
application_info.sType = VK_STRUCTURE_TYPE_APPLICATION_INFO;
application_info.apiVersion = VK_API_VERSION_1_2;

const char* instance_extensions[2] = { "VK_KHR_surface", "VK_KHR_win32_surface" };
const char* enabled_layers[1] = { "VK_LAYER_KHRONOS_validation" };

VkInstanceCreateInfo create_info = {};
create_info.sType = VK_STRUCTURE_TYPE_INSTANCE_CREATE_INFO;
create_info.pApplicationInfo = &application_info;
create_info.enabledExtensionCount = 2;
create_info.ppEnabledExtensionNames = instance_extensions;
create_info.enabledLayerCount = 1;
create_info.ppEnabledLayerNames = enabled_layers;

VkInstance instance;
VkResult result = vkCreateInstance(&create_info, nullptr, &instance);
CHECK_VULKAN_RESULT(result);
```


Instance Creation

```
VkApplicationInfo application_info = {};
application_info.sType = VK_STRUCTURE_TYPE_APPLICATION_INFO;
application_info.apiVersion = VK_API_VERSION_1_2;

const char* instance_extensions[2] = { "VK_KHR_surface", "VK_KHR_win32_surface" };
const char* enabled_layers[1] = { "VK_LAYER_KHRONOS_validation" };

VkInstanceCreateInfo create_info = {};
create_info.sType = VK_STRUCTURE_TYPE_INSTANCE_CREATE_INFO;
create_info.pApplicationInfo = &application_info;
create_info.enabledExtensionCount = 2;
create_info.ppEnabledExtensionNames = instance_extensions;
create_info.enabledLayerCount = 1;
create_info.ppEnabledLayerNames = enabled_layers;

VkInstance instance;
VkResult result = vkCreateInstance(&create_info, nullptr, &instance);
CHECK_VULKAN_RESULT(result);
```


Overview

We need to create/get hold of a couple of handles:		
Instance	Vulkan on your system	`VkInstance`
Window Surface	A window of your OS	`VkSurfaceKHR`
Physical Device	A hardware device (GPU)	`VkPhysicalDevice`
Queue	Received commands to be executed on a physical device	`VkQueue`
Logical Device	Main interface to a physical device (active configuration)	`VkDevice`
Swap Chain	Sends images to a monitor, Provides images to render into	`VkSwapchainKHR`

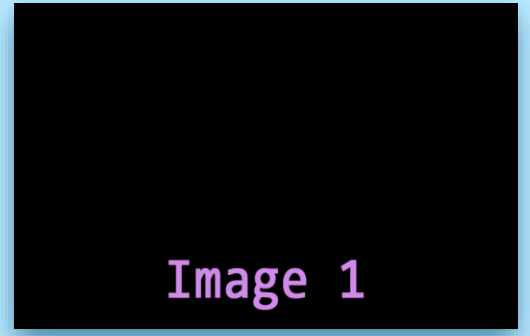
Overview

We need to create/get hold of a couple of handles:		
Instance	Vulkan on your system	`VkInstance`
Window Surface	A window of your OS	`VkSurfaceKHR`
Physical Device	A hardware device (GPU)	`VkPhysicalDevice`
Queue	Received commands to be executed on a physical device	`VkQueue`
Logical Device	Main interface to a physical device (active configuration)	`VkDevice`
Swap Chain	Sends images to a monitor, Provides images to render into	`VkSwapchainKHR`

Window Surface

Vulkan Application

```
while (true) {

  draw();

}
```



Operating System

```
present();
```

Window Surface

Vulkan Application

```
while (true) {

  draw();

}
```

Operating System

```
present();
```

Window Surface

Vulkan Application

```
while (true) {

  draw();

}
```

Operating System

```
present();
```

Window Surface

Vulkan Application

```
while (true) {

  draw();

}
```

Operating System

```
present();
```

VkSurfaceKHR

Overview

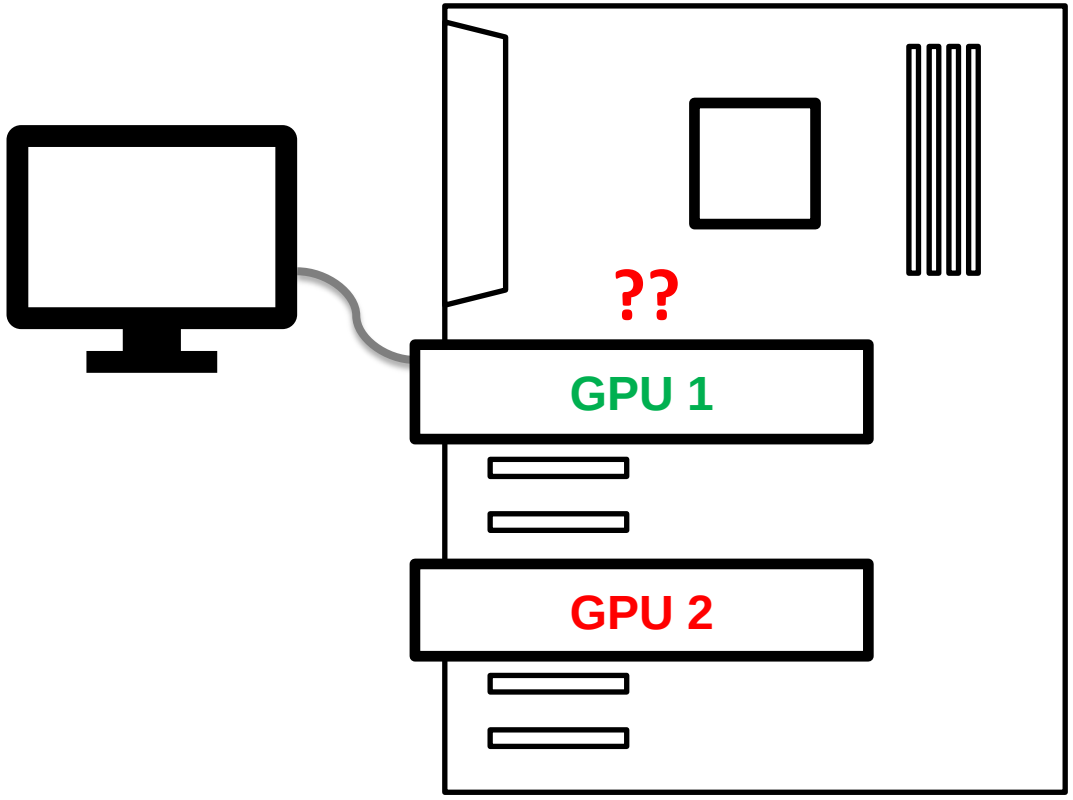
We need to create/get hold of a couple of handles:		
Instance	Vulkan on your system	`VkInstance`
Window Surface	A window of your OS	`VkSurfaceKHR`
Physical Device	A hardware device (GPU)	`VkPhysicalDevice`
Queue	Received commands to be executed on a physical device	`VkQueue`
Logical Device	Main interface to a physical device (active configuration)	`VkDevice`
Swap Chain	Sends images to a monitor, Provides images to render into	`VkSwapchainKHR`

Overview

We need to create/get hold of a couple of handles:		
Instance	Vulkan on your system	`VkInstance`
Window Surface	A window of your OS	`VkSurfaceKHR`
Physical Device	A hardware device (GPU)	`VkPhysicalDevice`
Queue	Received commands to be executed on a physical device	`VkQueue`
Logical Device	Main interface to a physical device (active configuration)	`VkDevice`
Swap Chain	Sends images to a monitor, Provides images to render into	`VkSwapchainKHR`

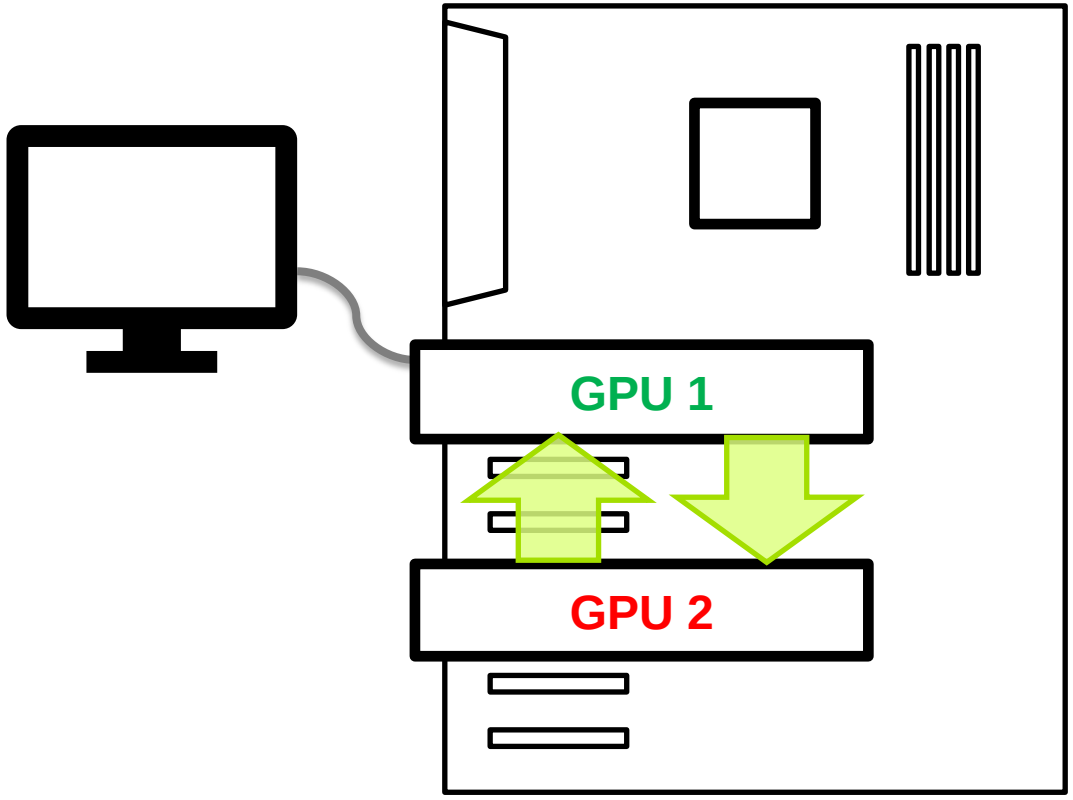
Physical Device Selection

If you have two GPUs, which GPU does the rendering?



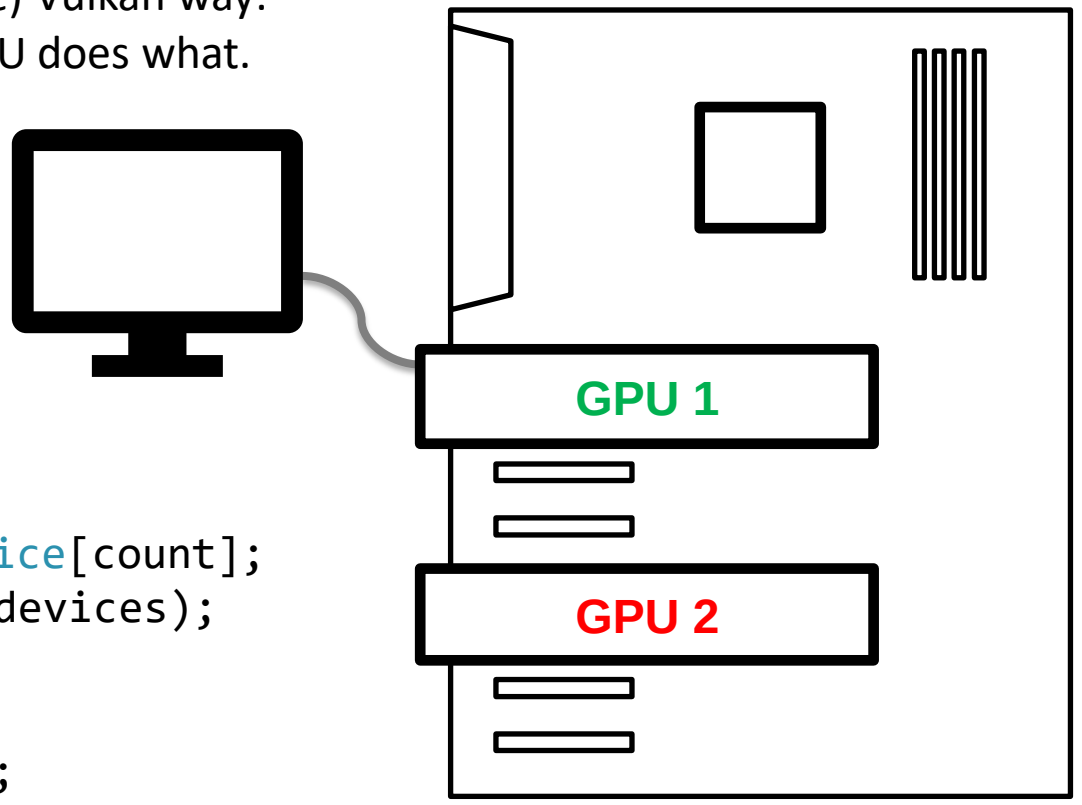
Physical Device Selection

The (expressive and verbose) Vulkan way:
Explicitly express which GPU does what.



Physical Device Selection

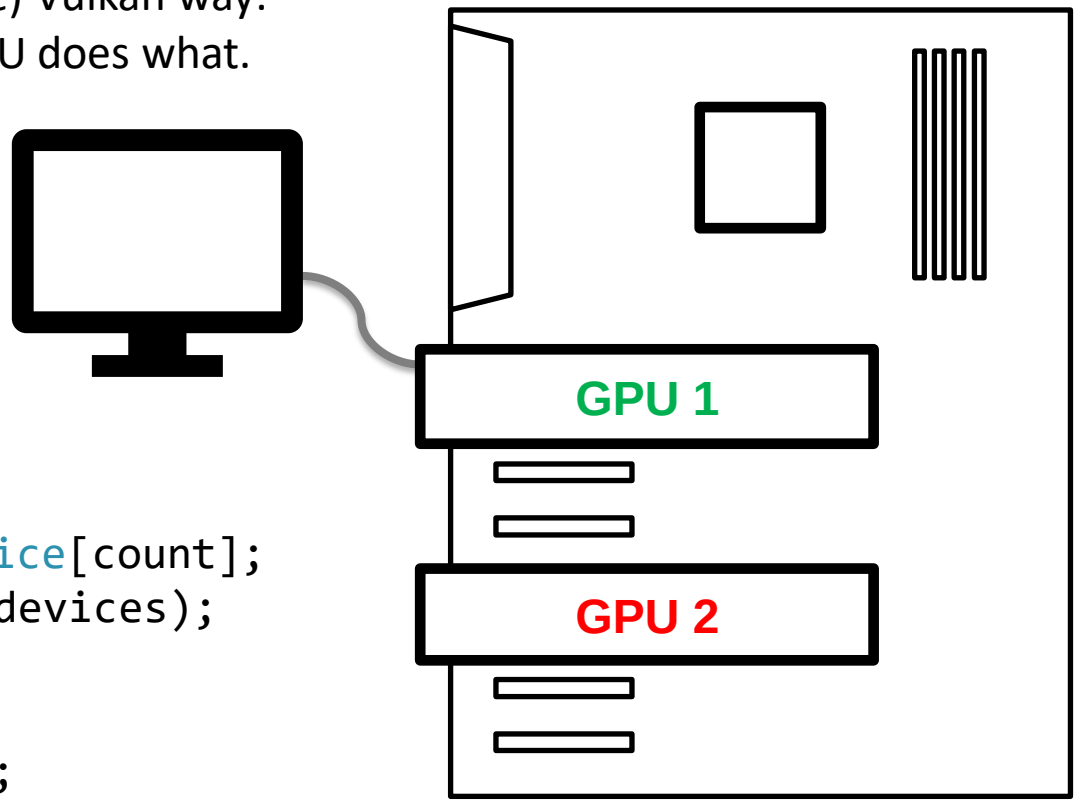
The (expressive and verbose) Vulkan way:
Explicitly express which GPU does what.




```
// Query the number of physical devices:
uint32_t count;
vkEnumeratePhysicalDevices(instance, &count, nullptr);
assert(count > 0);

// Get all physical device handles:
VkPhysicalDevice* physical_devices = new VkPhysicalDevice[count];
vkEnumeratePhysicalDevices(instance, &count, physical_devices);

// Select a physical device:
VkPhysicalDevice physical_device = physical_devices[0];

vkEnumerateDeviceExtensionProperties(physical_device, ...);
vkGetPhysicalDeviceProperties(physical_device, ...);
```

Physical Device Selection

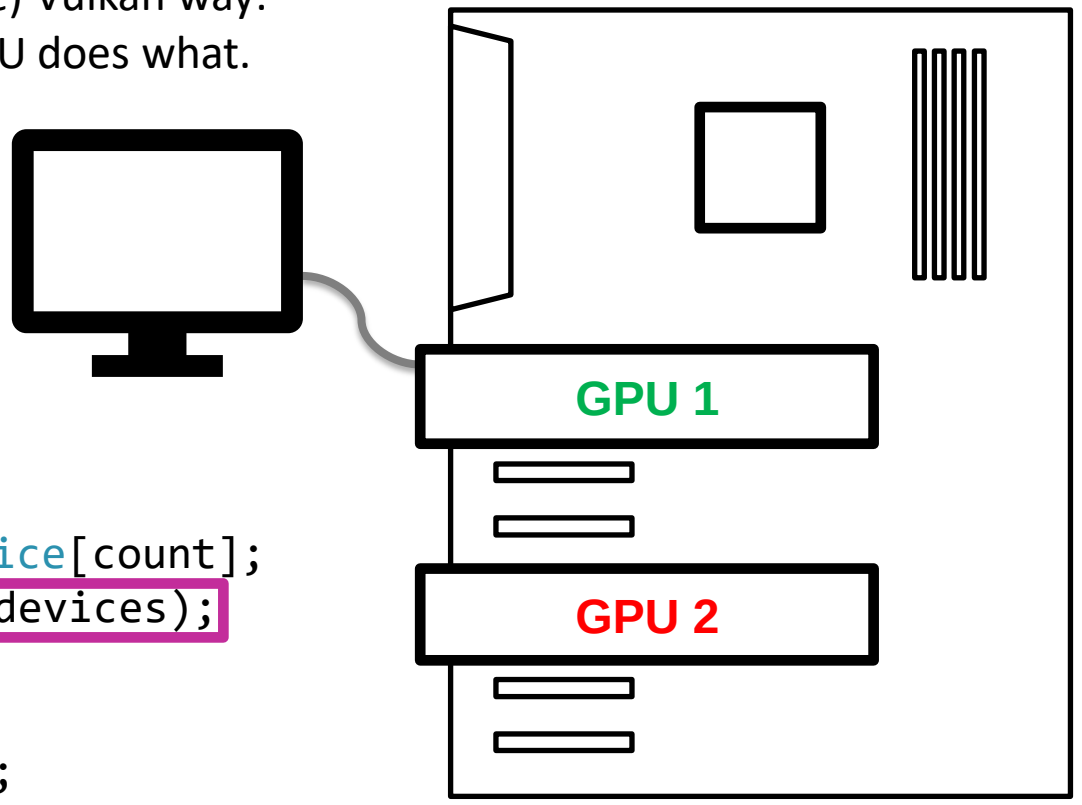
The (expressive and verbose) Vulkan way:
Explicitly express which GPU does what.




```
// Query the number of physical devices:
uint32_t count;
vkEnumeratePhysicalDevices(instance, &count, nullptr);
assert(count > 0);

// Get all physical device handles:
VkPhysicalDevice* physical_devices = new VkPhysicalDevice[count];
vkEnumeratePhysicalDevices(instance, &count, physical_devices);

// Select a physical device:
VkPhysicalDevice physical_device = physical_devices[0];

vkEnumerateDeviceExtensionProperties(physical_device, ...);
vkGetPhysicalDeviceProperties(physical_device, ...);
```

Physical Device Selection

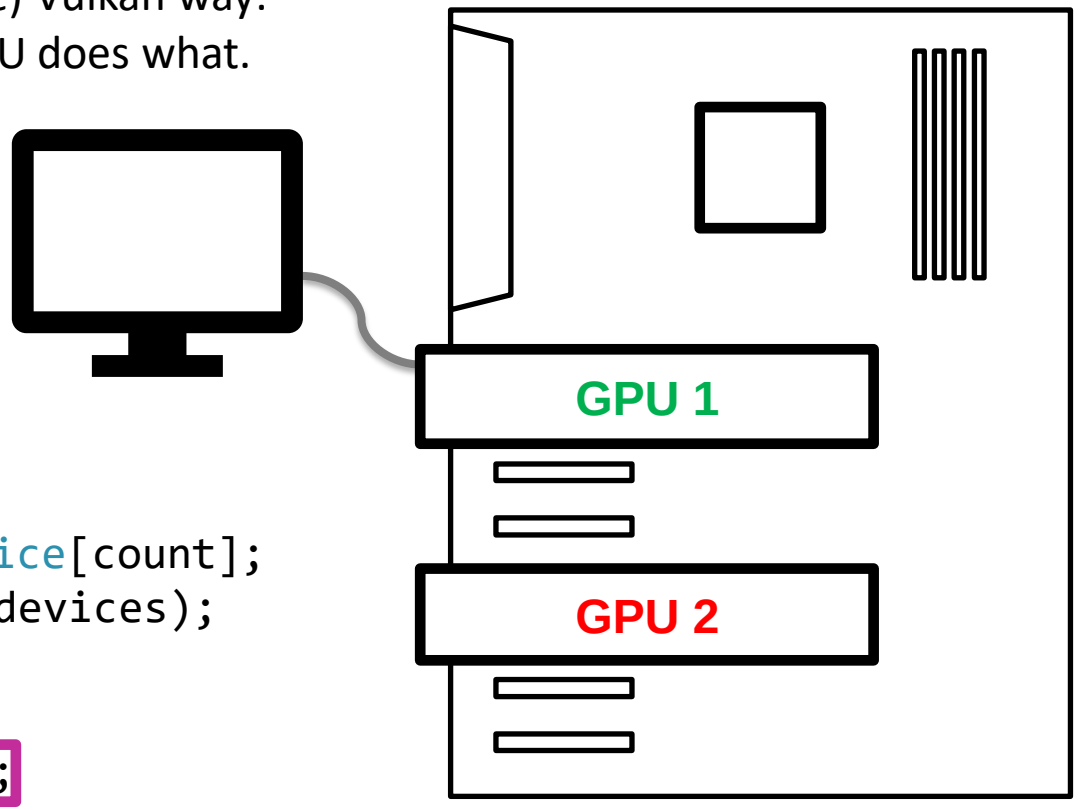
The (expressive and verbose) Vulkan way:
Explicitly express which GPU does what.




```
// Query the number of physical devices:
uint32_t count;
vkEnumeratePhysicalDevices(instance, &count, nullptr);
assert(count > 0);

// Get all physical device handles:
VkPhysicalDevice* physical_devices = new VkPhysicalDevice[count];
vkEnumeratePhysicalDevices(instance, &count, physical_devices);

// Select a physical device:
VkPhysicalDevice physical_device = physical_devices[0];

vkEnumerateDeviceExtensionProperties(physical_device, ...);
vkGetPhysicalDeviceProperties(physical_device, ...);
```

Physical Device Selection

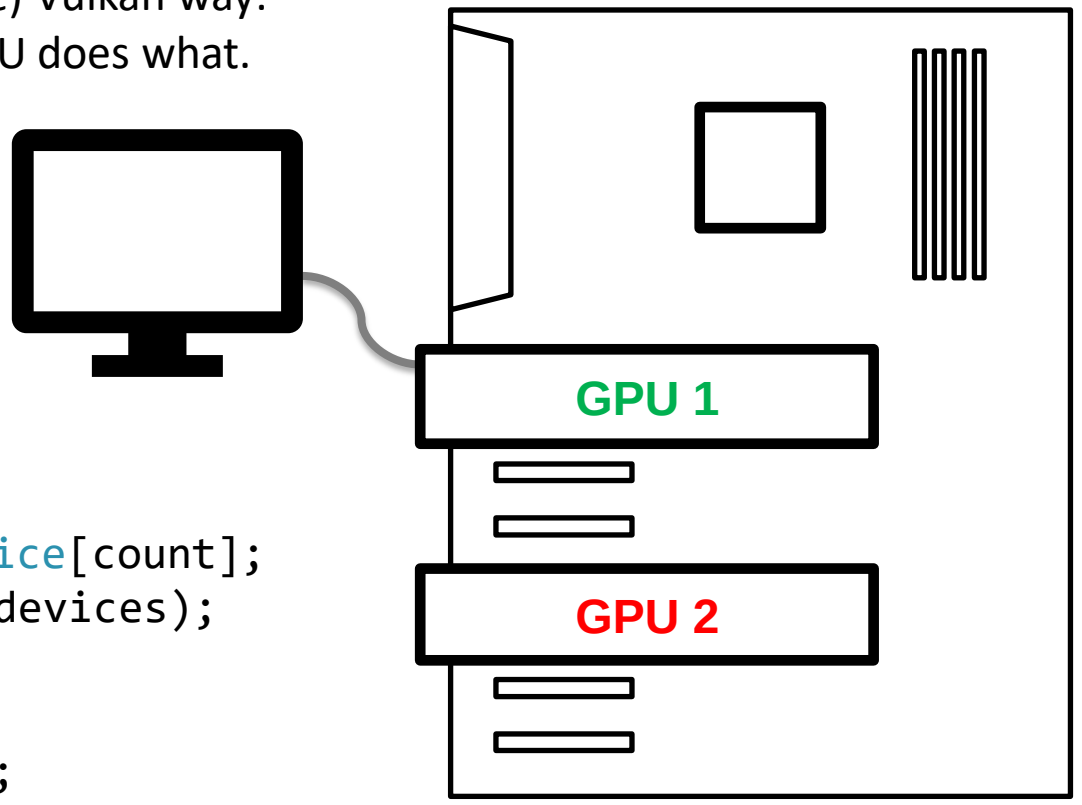
The (expressive and verbose) Vulkan way:
Explicitly express which GPU does what.




```
// Query the number of physical devices:
uint32_t count;
vkEnumeratePhysicalDevices(instance, &count, nullptr);
assert(count > 0);

// Get all physical device handles:
VkPhysicalDevice* physical_devices = new VkPhysicalDevice[count];
vkEnumeratePhysicalDevices(instance, &count, physical_devices);

// Select a physical device:
VkPhysicalDevice physical_device = physical_devices[0];

vkEnumerateDeviceExtensionProperties(physical_device, ...);
vkGetPhysicalDeviceProperties(physical_device, ...);
```

Physical Device Selection

The (expressive and verbose) Vulkan way:
Explicitly express which GPU does what.




```
// Query the number of physical devices:
uint32_t count;
vkEnumeratePhysicalDevices(instance, &count, nullptr);
assert(count > 0);

// Get all physical device handles:
VkPhysicalDevice* physical_devices = new VkPhysicalDevice[count];
vkEnumeratePhysicalDevices(instance, &count, physical_devices);

// Select a physical device:
VkPhysicalDevice physical_device = physical_devices[0];

vkEnumerateDeviceExtensionProperties(physical_device, ...);
vkGetPhysicalDeviceProperties(physical_device, ...);
```

Overview

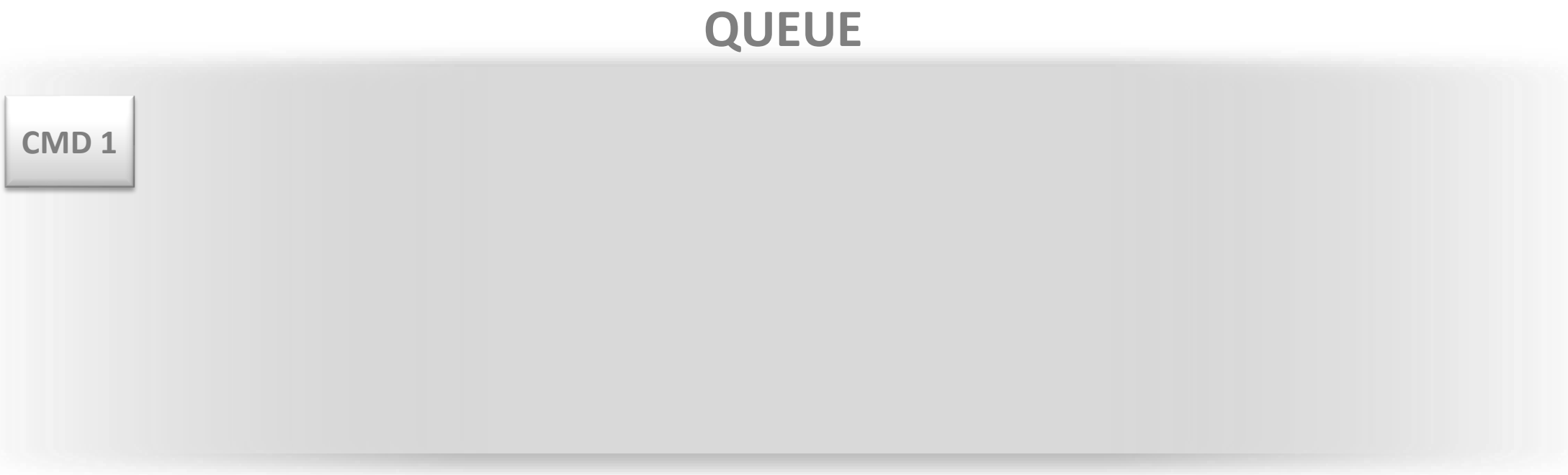
We need to create/get hold of a couple of handles:		
Instance	Vulkan on your system	`VkInstance`
Window Surface	A window of your OS	`VkSurfaceKHR`
Physical Device	A hardware device (GPU)	`VkPhysicalDevice`
Queue	Received commands to be executed on a physical device	`VkQueue`
Logical Device	Main interface to a physical device (active configuration)	`VkDevice`
Swap Chain	Sends images to a monitor, Provides images to render into	`VkSwapchainKHR`

Overview

We need to create/get hold of a couple of handles:		
Instance	Vulkan on your system	`VkInstance`
Window Surface	A window of your OS	`VkSurfaceKHR`
Physical Device	A hardware device (GPU)	`VkPhysicalDevice`
Queue	Received commands to be executed on a physical device	`VkQueue`
Logical Device	Main interface to a physical device (active configuration)	`VkDevice`
Swap Chain	Sends images to a monitor, Provides images to render into	`VkSwapchainKHR`

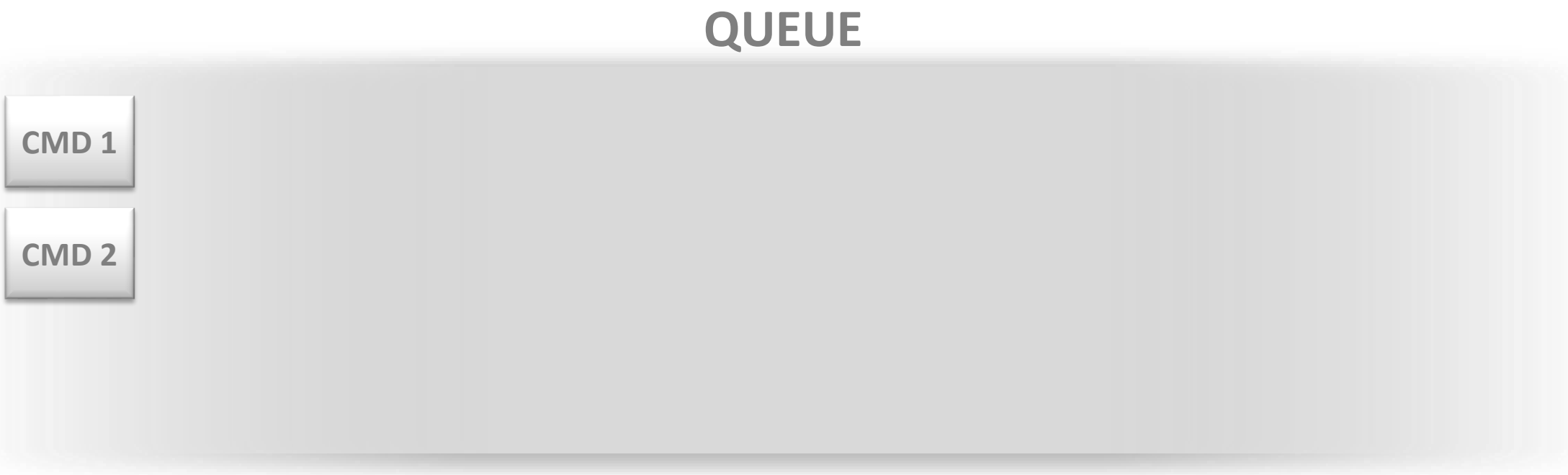
Queues

- A queue receives commands which are to be processed by the physical device.
- Commands (more precisely: command buffers) are queued for processing.
- Commands start being processed in submission order; can complete out of order



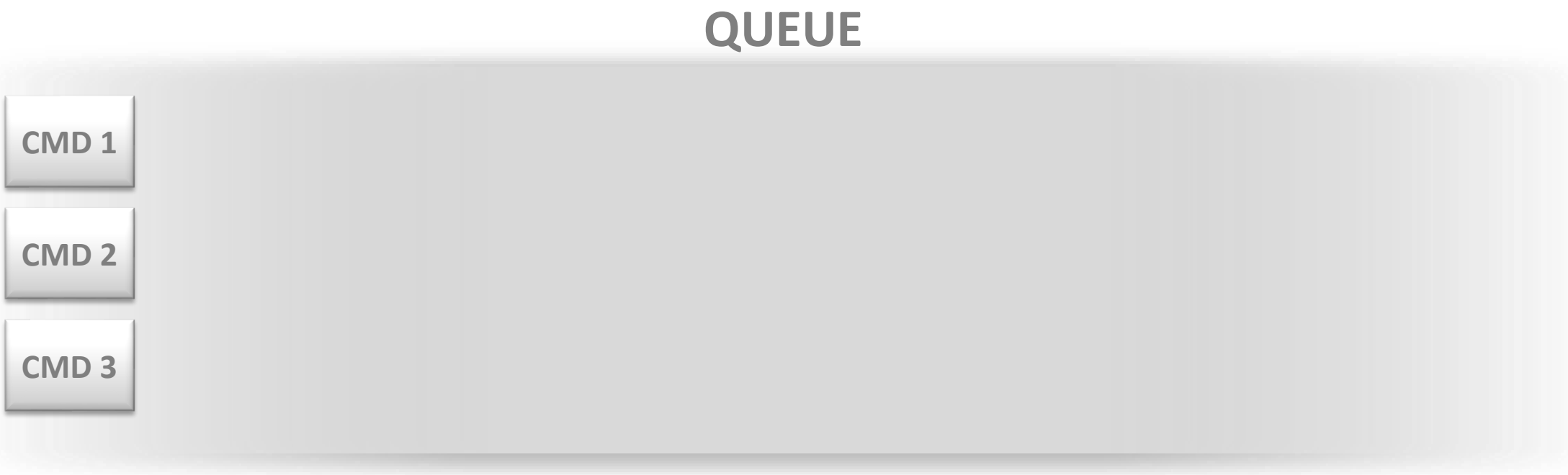
Queues

- A queue receives commands which are to be processed by the physical device.
- Commands (more precisely: command buffers) are queued for processing.
- Commands start being processed in submission order; can complete out of order



Queues

- A queue receives commands which are to be processed by the physical device.
- Commands (more precisely: command buffers) are queued for processing.
- Commands start being processed in submission order; can complete out of order



Logical Device Creation

```
float priority = 1.0f;
VkDeviceQueueCreateInfo queue_create_info  = {};
queue_create_info.sType                    = VK_STRUCTURE_TYPE_DEVICE_QUEUE_CREATE_INFO;
queue_create_info.queueFamilyIndex         = 0;
queue_create_info.queueCount               = 1;
queue_create_info.pQueuePriorities         = &priority;

const char* enabled_extensions[1]          = { "VK_KHR_swapchain" };

VkDeviceCreateInfo create_info             = {};
create_info.sType                          = VK_STRUCTURE_TYPE_DEVICE_CREATE_INFO;
create_info.queueCreateInfoCount           = 1;
create_info.pQueueCreateInfos              = &device_queue_create_info;
create_info.enabledExtensionCount          = 1;
create_info.ppEnabledExtensionNames        = enabled_extensions;

VkDevice device;
VkResult result = vkCreateDevice(physical_device, &create_info, nullptr, &device);
CHECK_VULKAN_RESULT(result);
```


Logical Device Creation

```
float priority = 1.0f;
VkDeviceQueueCreateInfo queue_create_info  = {};
queue_create_info.sType                    = VK_STRUCTURE_TYPE_DEVICE_QUEUE_CREATE_INFO;
queue_create_info.queueFamilyIndex         = 0;
queue_create_info.queueCount               = 1;
queue_create_info.pQueuePriorities         = &priority;

const char* enabled_extensions[1]          = { "VK_KHR_swapchain" };

VkDeviceCreateInfo create_info             = {};
create_info.sType                          = VK_STRUCTURE_TYPE_DEVICE_CREATE_INFO;
create_info.queueCreateInfoCount           = 1;
create_info.pQueueCreateInfos              = &device_queue_create_info;
create_info.enabledExtensionCount          = 1;
create_info.ppEnabledExtensionNames        = enabled_extensions;

VkDevice device;
VkResult result = vkCreateDevice(physical_device, &create_info, nullptr, &device);
CHECK_VULKAN_RESULT(result);
```

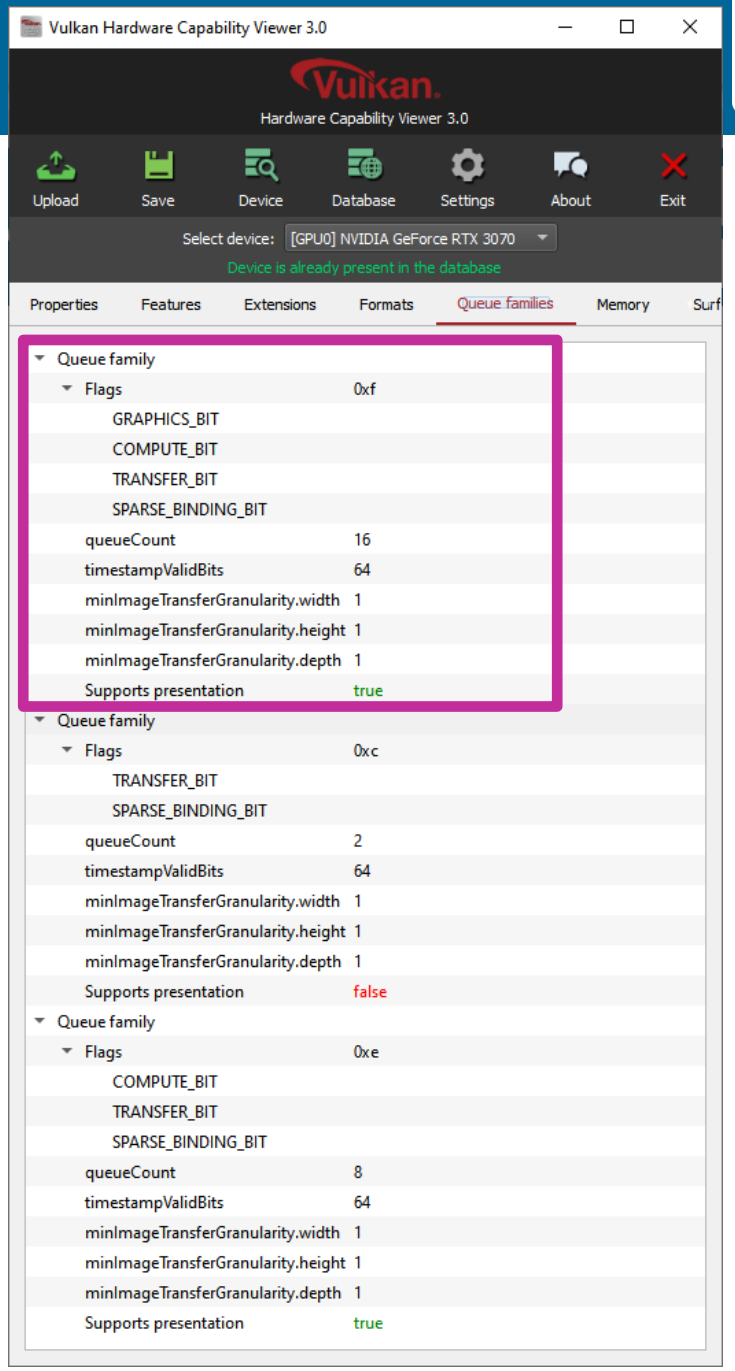

Queues

- A **queue** always belongs to a **queue family**.
- Queue families
  - A physical device **can** support different queue families … or only one.
  - Different queue families have different properties.
  - Multiple queues of the same queue family **can** be created and used.
- Why use multiple queues?
  - Increase concurrency
  - (Potentially) increase performance with specialized queues:
    - e.g., a “transfer queue”
    - e.g., an “async compute queue”

Queues

Queues

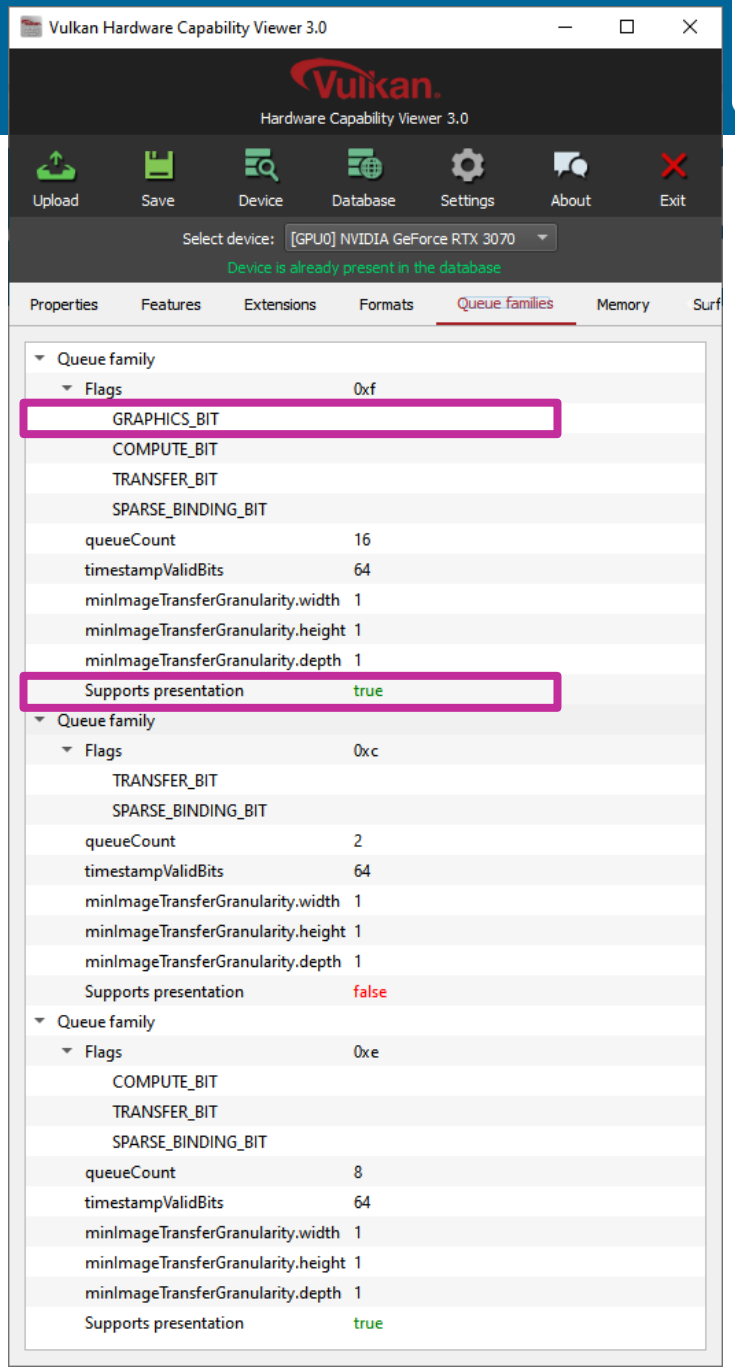
- A **queue** always belongs to a **queue family**.
- Queue families
  - A physical device **can** support different queue families … or only one.
  - Different queue families have different properties.
  - Multiple queues of the same queue family **can** be created and used.
- Why use multiple queues?
  - Increase concurrency
  - (Potentially) increase performance with specialized queues:
    - e.g., a “transfer queue”
    - e.g., an “async compute queue”



Vulkan Hardware Capability Viewer 3.0
© 2016-2020 by Sascha Willems

Queues

- A **queue** always belongs to a **queue family**.
- Queue families
  - A physical device **can** support different queue families … or only one.
  - Different queue families have different properties.
  - Multiple queues of the same queue family **can** be created and used.
- Why use multiple queues?
  - Increase concurrency
  - (Potentially) increase performance with specialized queues:
    - e.g., a “transfer queue”
    - e.g., an “async compute queue”



Vulkan Hardware Capability Viewer 3.0
© 2016-2020 by Sascha Willems

Overview

We need to create/get hold of a couple of handles:		
Instance	Vulkan on your system	`VkInstance`
Window Surface	A window of your OS	`VkSurfaceKHR`
Physical Device	A hardware device (GPU)	`VkPhysicalDevice`
Queue	Received commands to be executed on a physical device	`VkQueue`
Logical Device	Main interface to a physical device (active configuration)	`VkDevice`
Swap Chain	Sends images to a monitor, Provides images to render into	`VkSwapchainKHR`

Overview

We need to create/get hold of a couple of handles:		
Instance	Vulkan on your system	`VkInstance`
Window Surface	A window of your OS	`VkSurfaceKHR`
Physical Device	A hardware device (GPU)	`VkPhysicalDevice`
Queue	Received commands to be executed on a physical device	`VkQueue`
Logical Device	Main interface to a physical device (active configuration)	`VkDevice`
Swap Chain	Sends images to a monitor, Provides images to render into	`VkSwapchainKHR`

Logical Device Creation

```
float priority = 1.0f;
VkDeviceQueueCreateInfo queue_create_info  = {};
queue_create_info.sType                    = VK_STRUCTURE_TYPE_DEVICE_QUEUE_CREATE_INFO;
queue_create_info.queueFamilyIndex         = 0;
queue_create_info.queueCount               = 1;
queue_create_info.pQueuePriorities         = &priority;

const char* enabled_extensions[1]          = { "VK_KHR_swapchain" };

VkDeviceCreateInfo create_info             = {};
create_info.sType                          = VK_STRUCTURE_TYPE_DEVICE_CREATE_INFO;
create_info.queueCreateInfoCount           = 1;
create_info.pQueueCreateInfos              = &device_queue_create_info;
create_info.enabledExtensionCount          = 1;
create_info.ppEnabledExtensionNames        = enabled_extensions;

VkDevice device;
VkResult result = vkCreateDevice(physical_device, &create_info, nullptr, &device);
CHECK_VULKAN_RESULT(result);
```


Logical Device Creation

```
float priority = 1.0f;
VkDeviceQueueCreateInfo queue_create_info  = {};
queue_create_info.sType                    = VK_STRUCTURE_TYPE_DEVICE_QUEUE_CREATE_INFO;
queue_create_info.queueFamilyIndex         = 0;
queue_create_info.queueCount               = 1;
queue_create_info.pQueuePriorities         = &priority;

const char* enabled_extensions[1]          = { "VK_KHR_swapchain" };

VkDeviceCreateInfo create_info             = {};
create_info.sType                          = VK_STRUCTURE_TYPE_DEVICE_CREATE_INFO;
create_info.queueCreateInfoCount           = 1;
create_info.pQueueCreateInfos              = &device_queue_create_info;
create_info.enabledExtensionCount          = 1;
create_info.ppEnabledExtensionNames        = enabled_extensions;

VkDevice device;
VkResult result = vkCreateDevice(physical_device, &create_info, nullptr, &device);
CHECK_VULKAN_RESULT(result);
```


Logical Device Creation

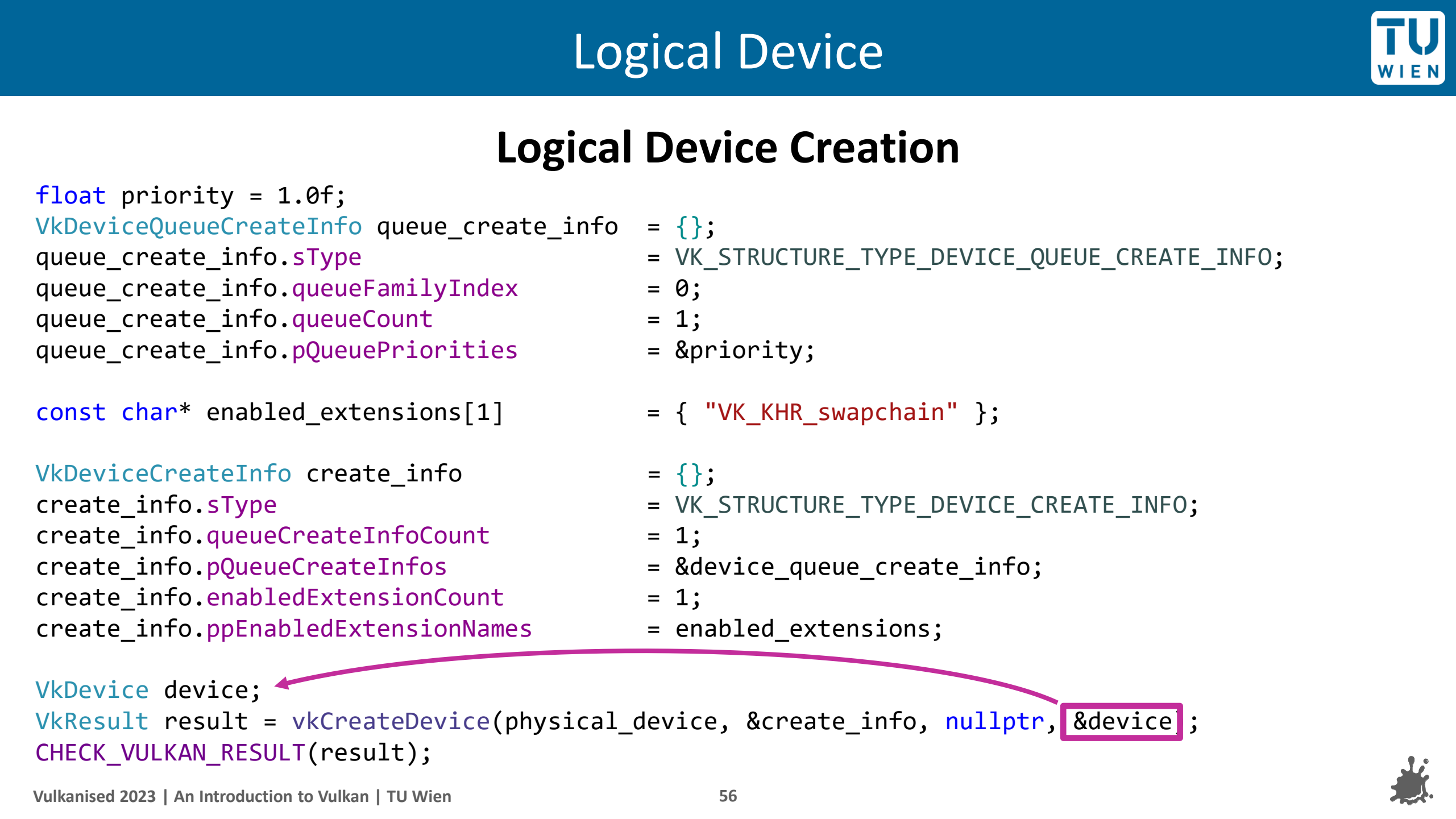
```
float priority = 1.0f;
VkDeviceQueueCreateInfo queue_create_info  = {};
queue_create_info.sType                    = VK_STRUCTURE_TYPE_DEVICE_QUEUE_CREATE_INFO;
queue_create_info.queueFamilyIndex         = 0;
queue_create_info.queueCount               = 1;
queue_create_info.pQueuePriorities         = &priority;

const char* enabled_extensions[1]          = { "VK_KHR_swapchain" };

VkDeviceCreateInfo create_info             = {};
create_info.sType                          = VK_STRUCTURE_TYPE_DEVICE_CREATE_INFO;
create_info.queueCreateInfoCount           = 1;
create_info.pQueueCreateInfos              = &device_queue_create_info;
create_info.enabledExtensionCount          = 1;
create_info.ppEnabledExtensionNames        = enabled_extensions;

VkDevice device;
VkResult result = vkCreateDevice(physical_device, &create_info, nullptr, &device);
CHECK_VULKAN_RESULT(result);
```


Logical Device Creation

```
float priority = 1.0f;
VkDeviceQueueCreateInfo queue_create_info  = {};
queue_create_info.sType                    = VK_STRUCTURE_TYPE_DEVICE_QUEUE_CREATE_INFO;
queue_create_info.queueFamilyIndex         = 0;
queue_create_info.queueCount               = 1;
queue_create_info.pQueuePriorities         = &priority;

const char* enabled_extensions[1]          = { "VK_KHR_swapchain" };

VkDeviceCreateInfo create_info             = {};
create_info.sType                          = VK_STRUCTURE_TYPE_DEVICE_CREATE_INFO;
create_info.queueCreateInfoCount           = 1;
create_info.pQueueCreateInfos              = &device_queue_create_info;
create_info.enabledExtensionCount          = 1;
create_info.ppEnabledExtensionNames        = enabled_extensions;

VkDevice device;
VkResult result = vkCreateDevice(physical_device, &create_info, nullptr, &device);
CHECK_VULKAN_RESULT(result);
```


Logical Device Creation

```
float priority = 1.0f;
VkDeviceQueueCreateInfo queue_create_info  = {};
queue_create_info.sType                    = VK_STRUCTURE_TYPE_DEVICE_QUEUE_CREATE_INFO;
queue_create_info.queueFamilyIndex         = 0;
queue_create_info.queueCount               = 1;
queue_create_info.pQueuePriorities         = &priority;

const char* enabled_extensions[1]          = { "VK_KHR_swapchain" };

VkDeviceCreateInfo create_info             = {};
create_info.sType                          = VK_STRUCTURE_TYPE_DEVICE_CREATE_INFO;
create_info.queueCreateInfoCount           = 1;
create_info.pQueueCreateInfos              = &device_queue_create_info;
create_info.enabledExtensionCount          = 1;
create_info.ppEnabledExtensionNames        = enabled_extensions;

VkDevice device;
VkResult result = vkCreateDevice(physical_device, &create_info, nullptr, &device);
CHECK_VULKAN_RESULT(result);
```



Overview

We need to create/get hold of a couple of handles:		
Instance	Vulkan on your system	`VkInstance`
Window Surface	A window of your OS	`VkSurfaceKHR`
Physical Device	A hardware device (GPU)	`VkPhysicalDevice`
Queue	Received commands to be executed on a physical device	`VkQueue`
Logical Device	Main interface to a physical device (active configuration)	`VkDevice`
Swap Chain	Sends images to a monitor, Provides images to render into	`VkSwapchainKHR`

Overview

We need to create/get hold of a couple of handles:		
Instance	Vulkan on your system	`VkInstance`
Window Surface	A window of your OS	`VkSurfaceKHR`
Physical Device	A hardware device (GPU)	`VkPhysicalDevice`
Queue	Received commands to be executed on a physical device	`VkQueue`
Logical Device	Main interface to a physical device (active configuration)	`VkDevice`
Swap Chain	Sends images to a monitor, Provides images to render into	`VkSwapchainKHR`

The Swap Chain

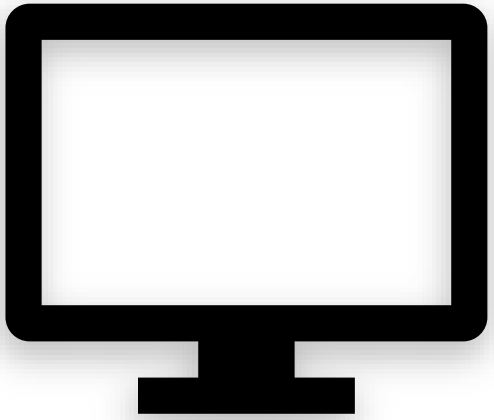
Application/Render Loop

```
while (true) {

  acquireNextImage();



  draw();



  present();

}
```

Swap Chain

available images:

presented image:



The Swap Chain

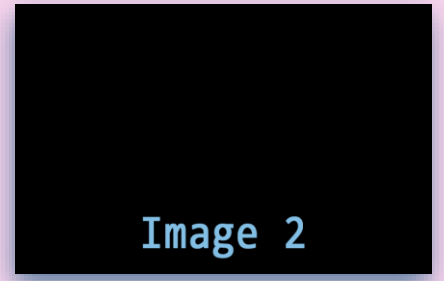
Application/Render Loop

```
while (true) {

  acquireNextImage();

  draw();

  present();

}
```

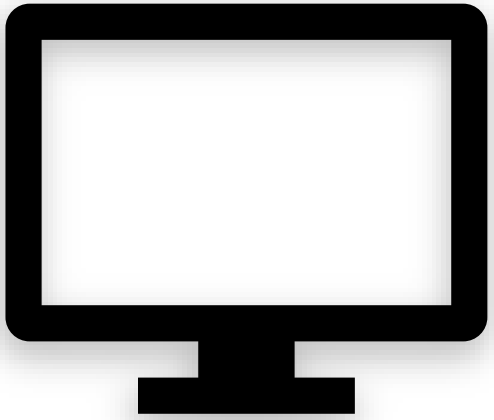
Current **Backbuffer**

Swap Chain

available images:



presented image:



The Swap Chain

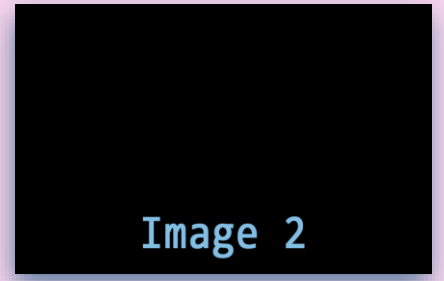
Application/Render Loop

```
while (true) {

  acquireNextImage();

  draw();

  present();

}
```

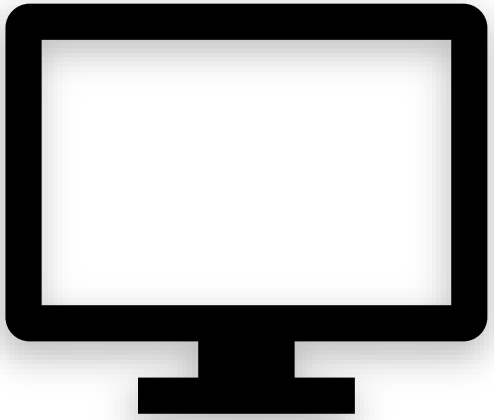
Current **Backbuffer**

Swap Chain

available images:



presented image:



The Swap Chain

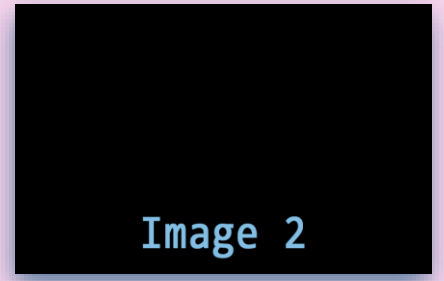
Application/Render Loop

```
while (true) {

  acquireNextImage();



  draw();



  present();

}
```

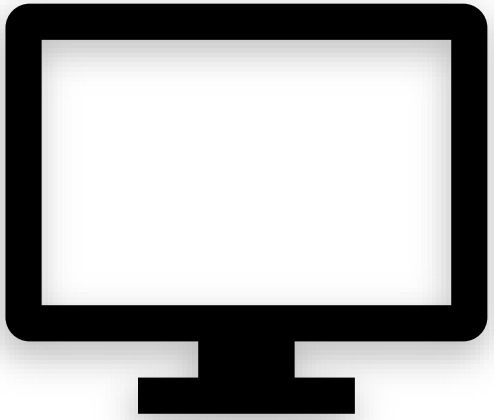
Current **Backbuffer**

Swap Chain

available images:



presented image:



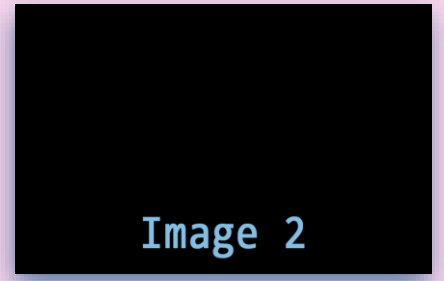
The Swap Chain

Application/Render Loop

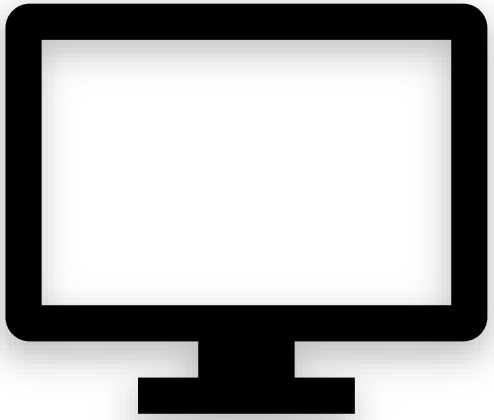
```
while (true) {

  acquireNextImage();



  draw();



  present();
}
```

Swap Chain

available images:



presented image:



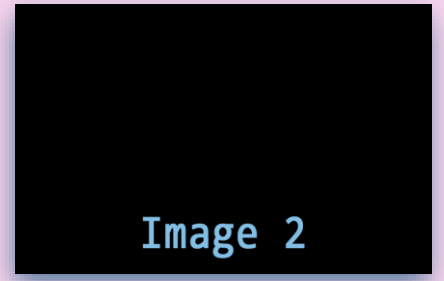
The Swap Chain

Application/Render Loop

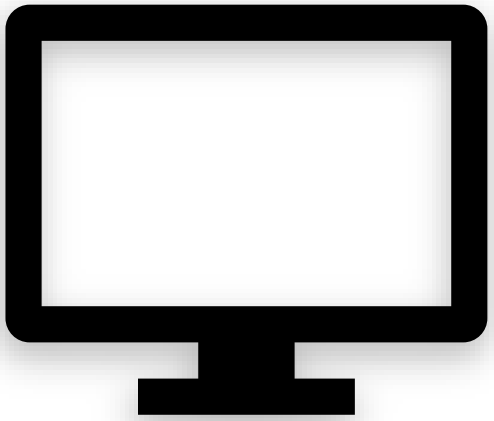
```
while (true) {

  acquireNextImage();



  draw();



  present();

}
```

Swap Chain

available images:



presented image:



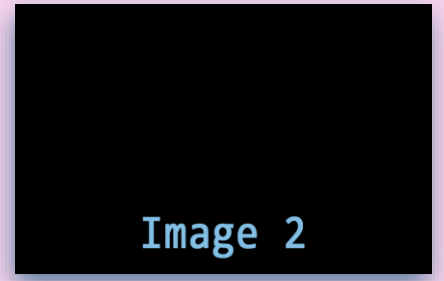
The Swap Chain

Application/Render Loop

```
while (true) {

  acquireNextImage();



  draw();



  present();

}
```

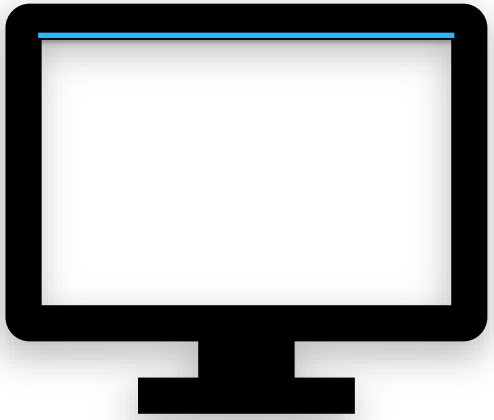
Swap Chain

available images:



presented image:

Current **Frontbuffer**



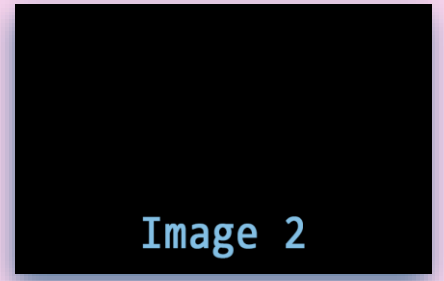
The Swap Chain

Application/Render Loop

```
while (true) {

  acquireNextImage();



  draw();



  present();

}
```

Swap Chain

available images:



presented image:

Current **Frontbuffer**

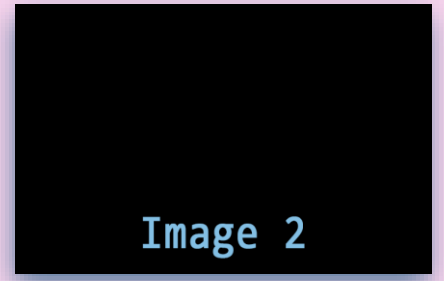
The Swap Chain

Application/Render Loop

```
while (true) {

  acquireNextImage();



  draw();



  present();

}
```

Swap Chain

available images:



presented image:

Current **Frontbuffer**

The Swap Chain

Application/Render Loop

```
while (true) {

  acquireNextImage();

  draw();

  present();

}
```

Image 2

Current **Backbuffer**

Swap Chain

available images:

presented image:

Current **Frontbuffer**

The Swap Chain

Application/Render Loop

```
while (true) {

  acquireNextImage();



  draw();

  present();

}
```

Current **Backbuffer**

Swap Chain

available images:

presented image:

The Swap Chain

Application/Render Loop

```
while (true) {

  acquireNextImage();



  draw();



  present();
}
```

Swap Chain

available images:

presented image:

The Swap Chain

Application/Render Loop

```
while (true) {

  acquireNextImage();

  draw();

  present();

}
```

Swap Chain

available images:

presented image:

Overview

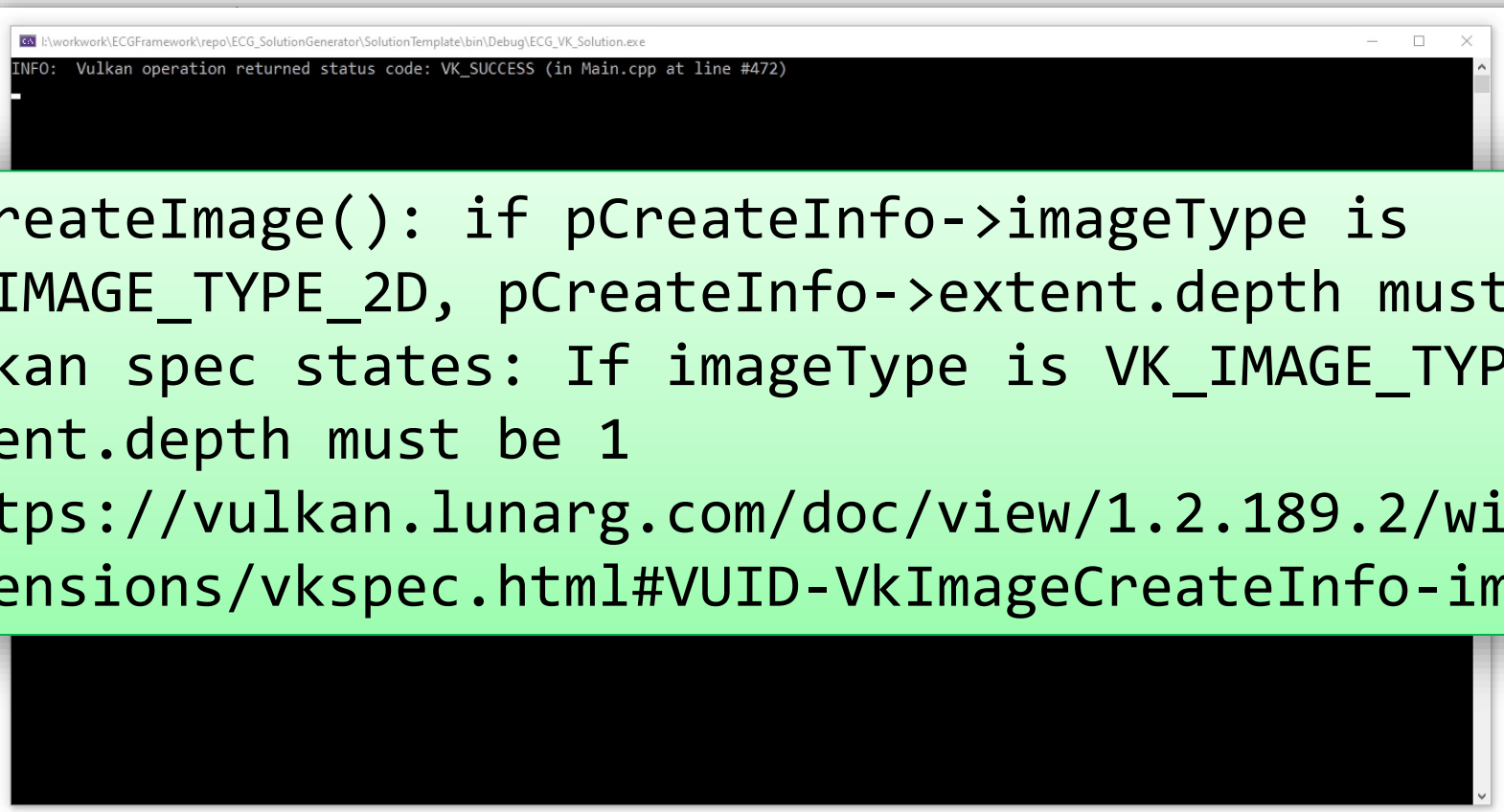
We need to create/get hold of a couple of handles:		
Instance	Vulkan on your system	`VkInstance`
Window Surface	A window of your OS	`VkSurfaceKHR`
Physical Device	A hardware device (GPU)	`VkPhysicalDevice`
Queue	Received commands to be executed on a physical device	`VkQueue`
Logical Device	Main interface to a physical device (active configuration)	`VkDevice`
Swap Chain	Sends images to a monitor, Provides images to render into	`VkSwapchainKHR`

Overview

We need to create/get hold of a couple of handles:		
Instance	Vulkan on your system	`VkInstance`
Window Surface	A window of your OS	`VkSurfaceKHR`
Physical Device	A hardware device (GPU)	`VkPhysicalDevice`
Queue	Received commands to be executed on a physical device	`VkQueue`
Logical Device	Main interface to a physical device (active configuration)	`VkDevice`
Swap Chain	Sends images to a monitor, Provides images to render into	`VkSwapchainKHR`

Validation OFF

```
// Crea
VkImag
create
c
c
c
c
c
c
c

V
VkResu
CHECK_
```

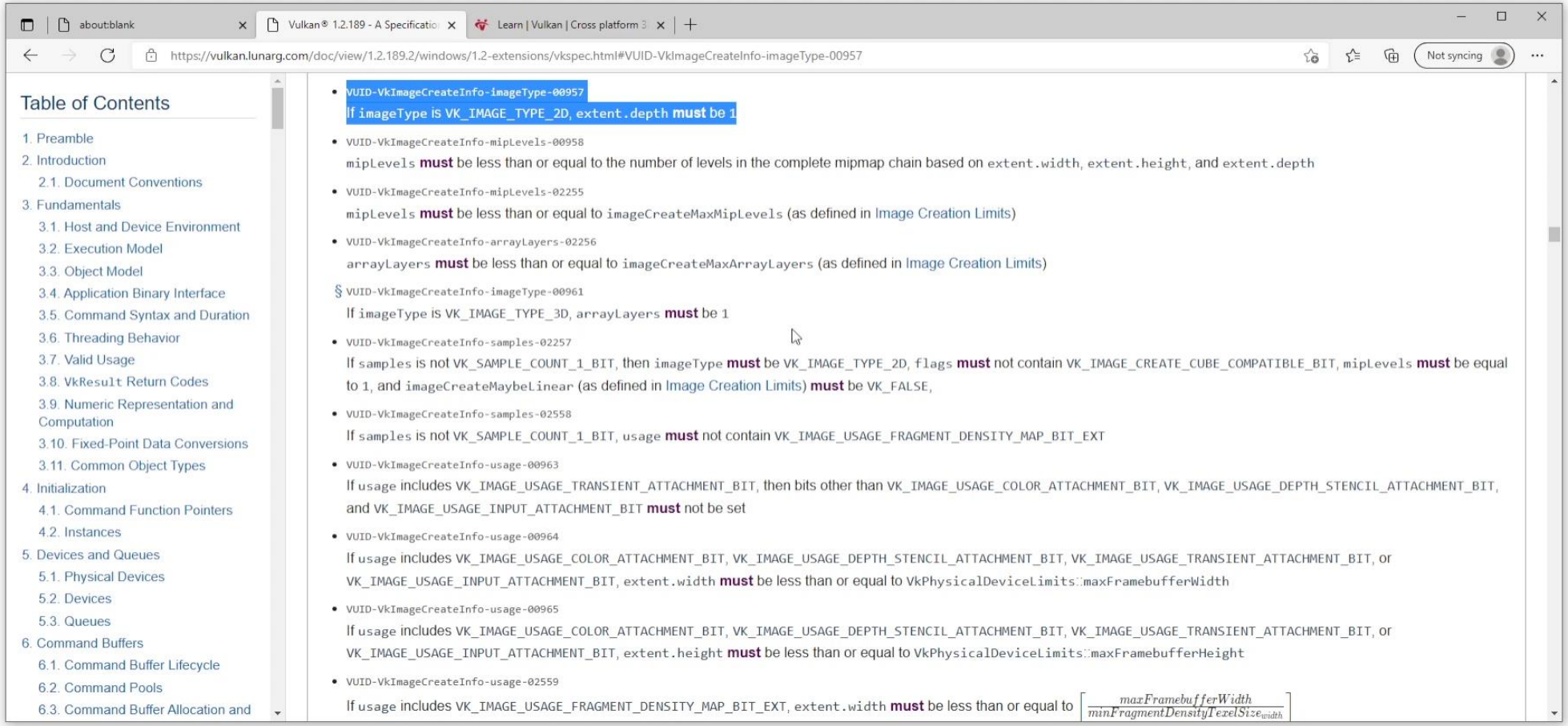


vkCreateImage(): if pCreateInfo->imageType is VK_IMAGE_TYPE_2D, pCreateInfo->extent.depth must be 1. The Vulkan spec states: If imageType is VK_IMAGE_TYPE_2D, extent.depth must be 1 (https://vulkan.lunarg.com/doc/view/1.2.189.2/windows/1.2-extensions/vkspec.html#VUID-VkImageCreateInfo-imageType-00957)

The Vulkan Specification

is your best friend!

```
// Cre
VkImag
create
create
create
create
create
create
create
create
create

VkImag
VkResu
CHECK_
```



The Vulkan Specification

is very explicit.

```
// Create a new image:
VkImageCreateInfo create_info = {};
create_info.sType             = VK_STRUCTURE_TYPE_IMAGE_CREATE_INFO;
create_info.imageType         = VK_IMAGE_TYPE_2D;
create_info.format            = VK_FORMAT_R8G8B8A8_UNORM;
create_info.extent.width      = 512;
create_info.extent.height     = 512;
create_info.extent.depth      =   1;
create_info.arrayLayers       =   1;
create_info.mipLevels         =   1;
create_info.samples           = VK_SAMPLE_COUNT_1_BIT;

VkImage image;
VkResult result = vkCreateImage(device, &create_info, nullptr, &image);
CHECK_VULKAN_RESULT(result);
```

Vulkanised 2023

The 5th Vulkan Developer Conference
Munich, Germany / February 7–9

PART 1

- Fundamental Vulkan Handles
- Window System Integration
- The Swap Chain

GOOD LUCK!

Schedule

PART 1:

Setup	Lecture	Coding Session
10 min	**20** min	**90** min
Starts at 09:00	Starts at 09:10	Starts at 09:30

PART 2:

Lecture	Coffee Break	Coding Session
15 min	**25** min	**80** min
Starts at 11:00	Starts at 11:15	Starts at 11:40

Lunch Break 13:00 – 14:00

PART 3:

Lecture	Coding Session	Coffee Break
15 min	**65** min	**30** min
Starts at 14:00	Starts at 14:15	Starts at 15:20

PART 4:

Lecture	Coding Session	Closing
20 min	**70** min	**10** min
Starts at 15:50	Starts at 16:10	Starts at 17:20