A dive into low level Rendering

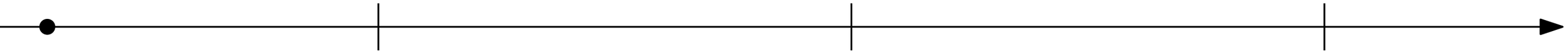
https://gitlab2.informatik.uni-wuerzburg.de/opencolloq/vulkan-hello-triangle

How did we get here?

How did we get here?

Doom (1993)

How did we get here?



Doom (1993)

How did we get here?

Doom (1993)

OpenGL (1992)

How did we get here?

Microsoft®
DirectX®

Doom (1993)
OpenGL (1992)

How did we get here?

OpenGL (1992)

Doom (1993)

DirectX 1 (1995)

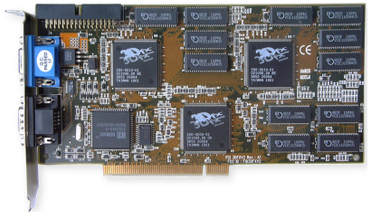
How did we get here?

OpenGL (1992)

Doom (1993)

DirectX 1 (1995)

How did we get here?



3dfx Voodoo (1996)

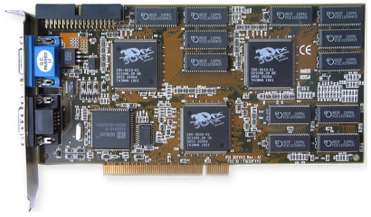
Quake (1996)

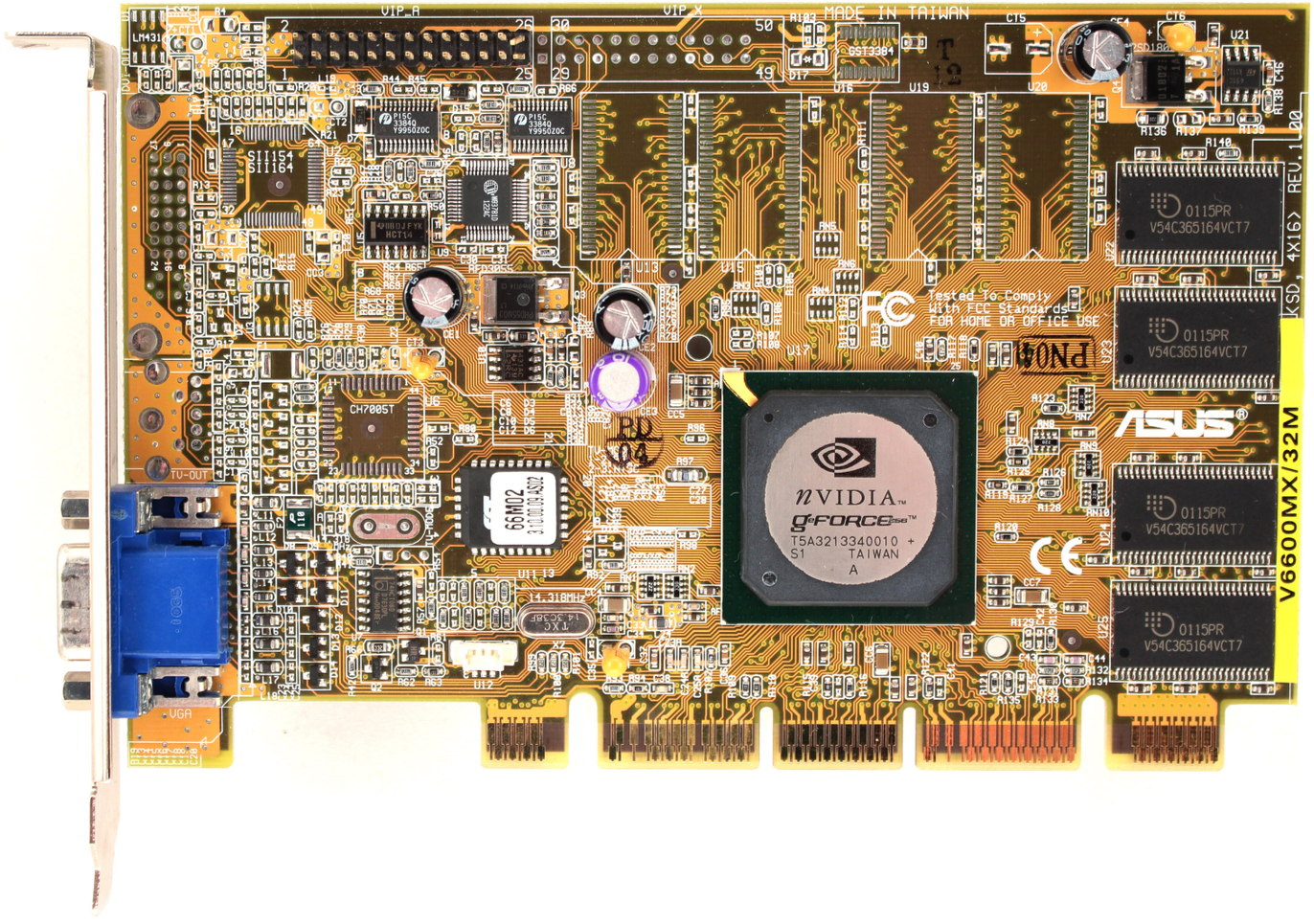
DirectX 1 (1995)

Doom (1993)

OpenGL (1992)

How did we get here?





3dfx Voodoo (19

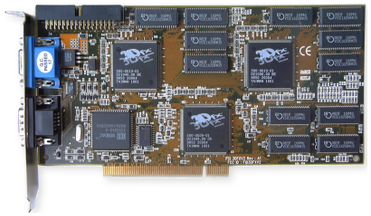
Quake (1996)

DirectX 1 (1995)

Doom (1993)

OpenGL (1992)

How did we get here?



OpenGL (1992)

Doom (1993)

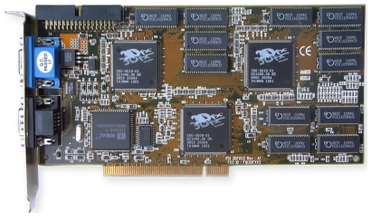
DirectX 1 (1995)

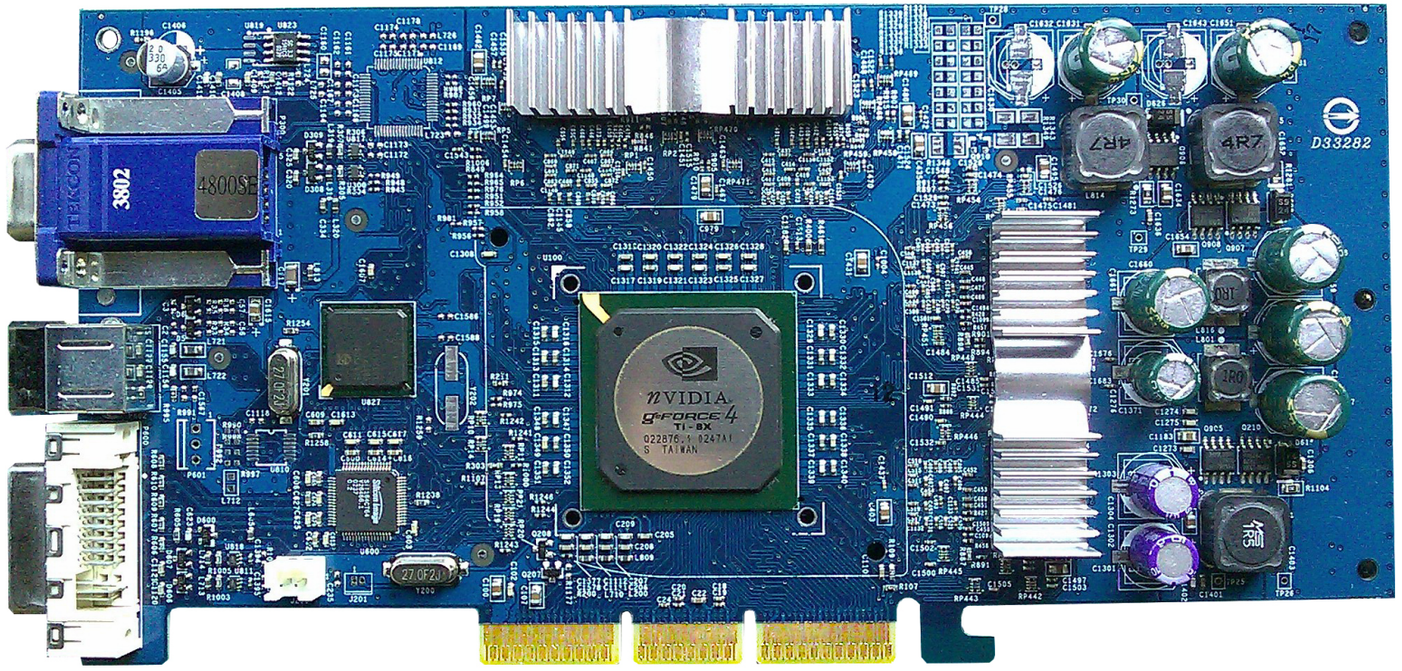
Quake (1996)

3dfx Voodoo (1996)

Nvidia-GeForce-256 (1999)

How did we get here?





OpenGL (1992)

Doom (1993)

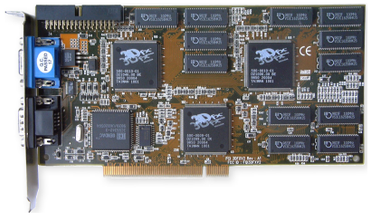
DirectX 1 (1995)

Quake (1996)

3dfx Voodoo (1996)

Nvidi-

How did we get here?



3dfx Voodoo (1996)

Nvidia-GeForce-256 (1999)

GeForce 4 (2002)

OpenGL (1992)

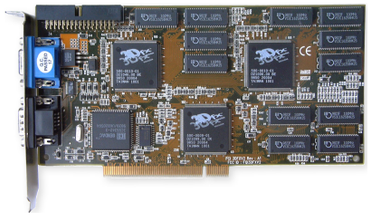
Doom (1993)

DirectX 1 (1995)

Quake (1996)

DirectX 8 (2000)

How did we get here?



OpenGL (1992)

Doom (1993)

DirectX 1 (1995)

Quake (1996)

3dfx Voodoo (1996)

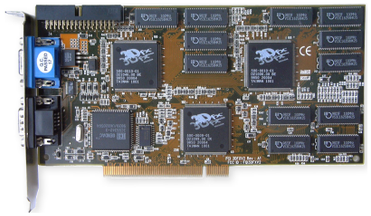
Nvidia-GeForce-256 (1999)

DirectX 8 (2000)

GeForce 4 (2002)

OpenGL 2 (2004)

How did we get here?



OpenGL (1992)

Doom (1993)

DirectX 1 (1995)

Quake (1996)

3dfx Voodoo (1996)

Nvidia-GeForce-256 (1999)

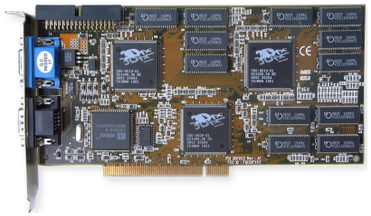
DirectX 8 (2000)

GeForce 4 (2002)

OpenGL 2 (2004)

DirectX 11 (2009)

How did we get here?



OpenGL (1992)
Doom (1993)
DirectX 1 (1995)
Quake (1996)
3dfx Voodoo (1996)
Nvidia-GeForce-256 (1999)
DirectX 8 (2000)
GeForce 4 (2002)
OpenGL 2 (2004)
DirectX 11 (2009)
DirectX 12 (2015)
Vulkan (2016)

Hardware overview

https://gpuopen.com/wp-content/uploads/2021/01/AMD_Graphics_pipeline_GIC2020.pdf

What is Vulkan?

Vulkan is a low-level, low-overhead cross-platform API and open standard for 3D graphics and computing. It was intended to address the shortcomings of OpenGL, and allow developers more control over the GPU. It is designed to support a wide variety of GPUs, CPUs and operating systems, and it is also designed to work with modern multi-core CPUs.

What is Vulkan?

Vulkan is a low-level, low-overhead cross-platform API and open standard for 3D graphics and computing. It was intended to address the shortcomings of OpenGL, and allow developers more control over the GPU. It is designed to support a wide variety of GPUs, CPUs and operating systems, and it is also designed to work with modern multi-core CPUs.

What is Vulkan?

Vulkan is a low-level, low-overhead cross-platform API and open standard for 3D graphics and computing. It was intended to address the shortcomings of OpenGL, and allow developers more control over the GPU. It is designed to support a wide variety of GPUs, CPUs and operating systems, and it is also designed to work with modern multi-core CPUs.

What is Vulkan?

Vulkan is a low-level, low-overhead cross-platform API and open standard for 3D graphics and computing. It was intended to address the shortcomings of OpenGL, and allow developers more control over the GPU. It is designed to support a wide variety of GPUs, CPUs and operating systems, and it is also designed to work with modern multi-core CPUs.

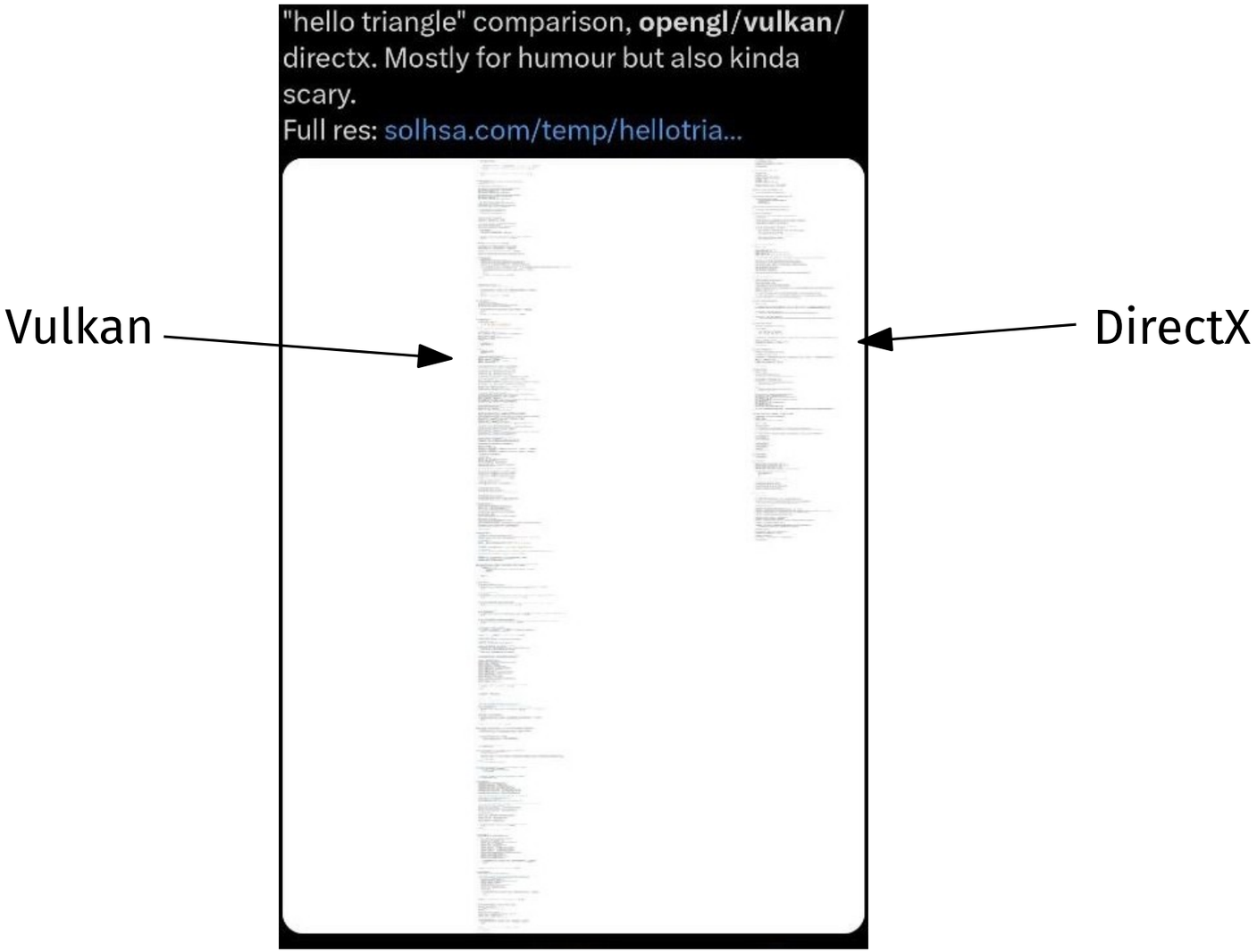
What is Vulkan?

Vulkan is a low-level, low-overhead cross-platform API and open standard for 3D graphics and computing. It was intended to address the shortcomings of OpenGL, and allow developers more control over the GPU. It is designed to support a wide variety of GPUs, CPUs and operating systems, and it is also designed to work with modern multi-core CPUs.

What is Vulkan?

Vulkan is a low-level, low-overhead cross-platform API and open standard for 3D graphics and computing. It was intended to address the shortcomings of OpenGL, and allow developers more control over the GPU. It is designed to support a wide variety of GPUs, CPUs and operating systems, and it is also designed to work with modern multi-core CPUs.

"hello triangle" comparison, **opengl**/**vulkan**/directx. Mostly for humour but also kinda scary.
Full res: solhsa.com/temp/hellotria...


"hello triangle" comparison, **opengl**/**vulkan**/ directx. Mostly for humour but also kinda scary.
Full res: solhsa.com/temp/hellotria...
Vulkan
DirectX

OpenGL 1.x

OpenGL 3+

Vulkan

"Hello Triangle".
- GL1 uses SDL,
- GL3+ and Vulkan use GLFW3.

Admittedly the Vulkan version has more comments and the GL versions could do more error checking..

DX9 **DX11** **DX12**

Hello Triangle[DX]
- DX12 doesn't include shaders (10+25 lines)
- DX9 and 11 could do more error checking..

Extensions

VK_KHR_ray_query

VK_EXT_mesh_shader

VK_EXT_full_screen_exclusive

Vulkan Core

- lightweight
- supported by every device

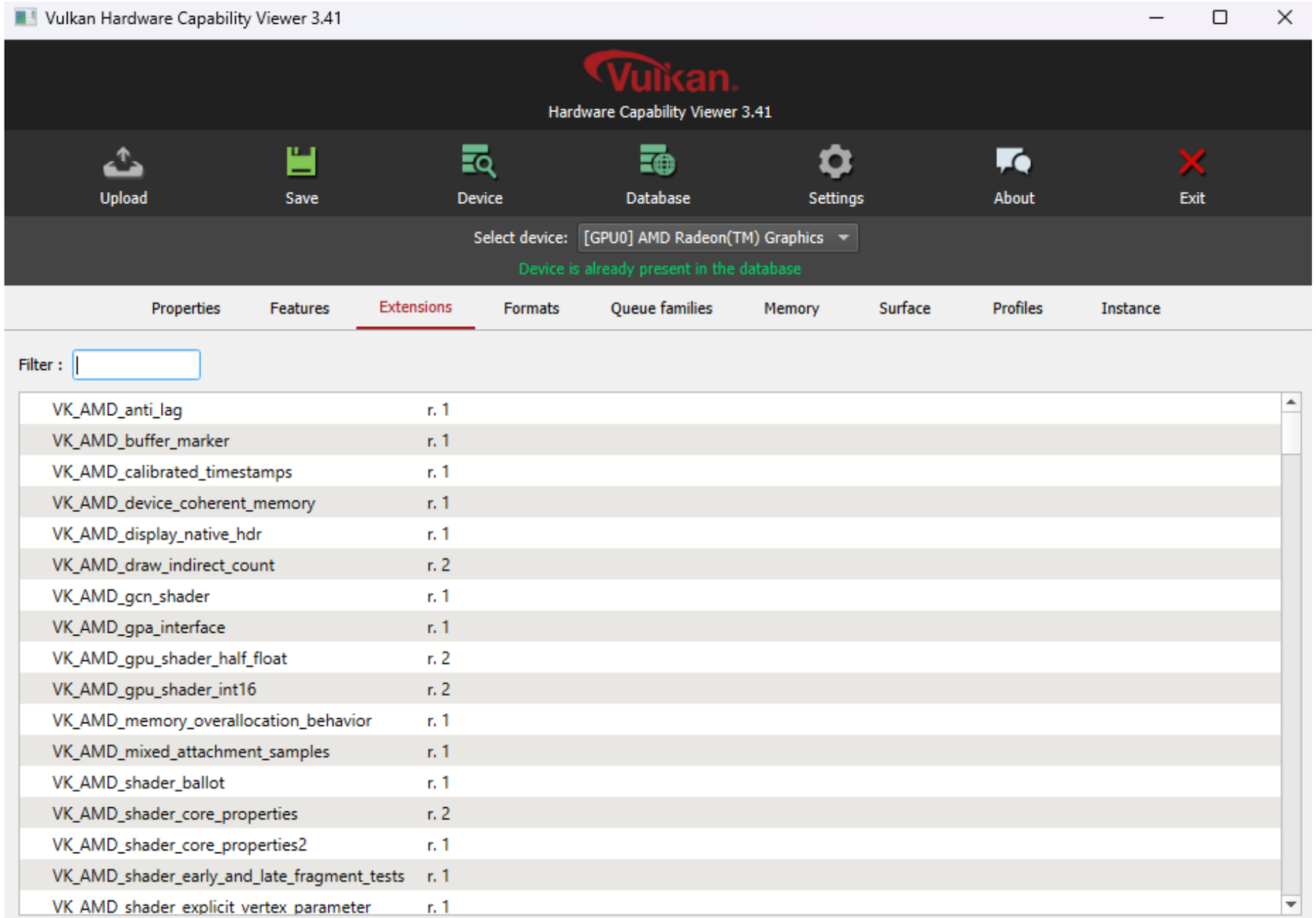
VK_KHR_video_decode_av1

VK_LAYER_KHRONOS_validation

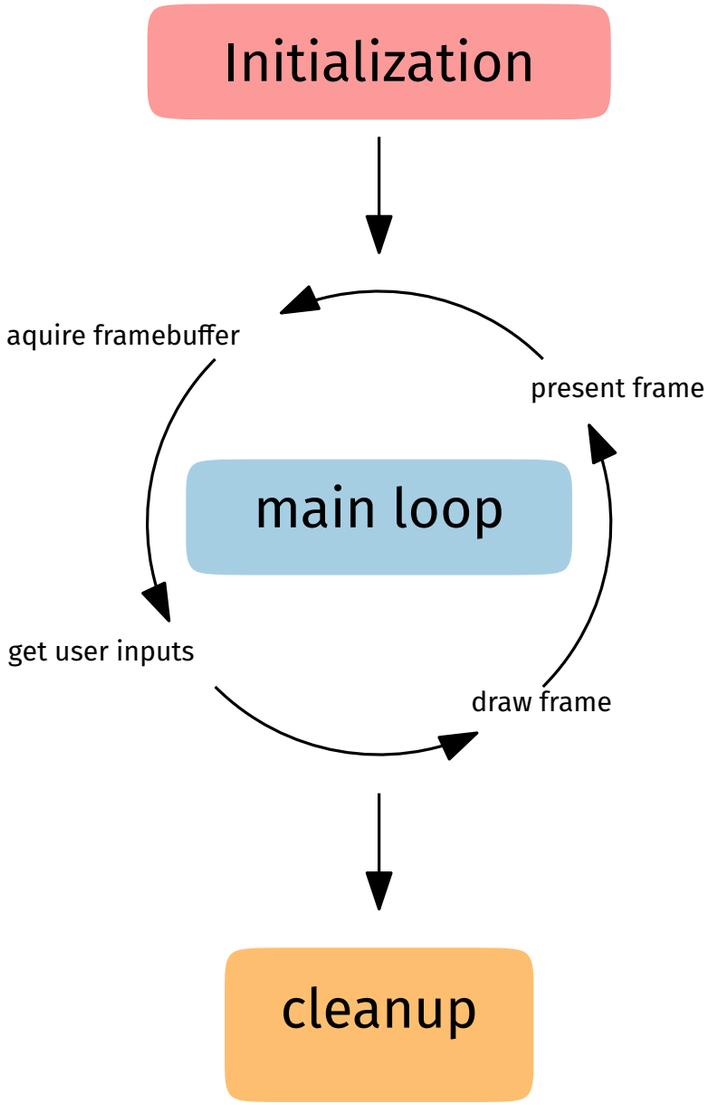
VK_KHR_video_decode_h264

...

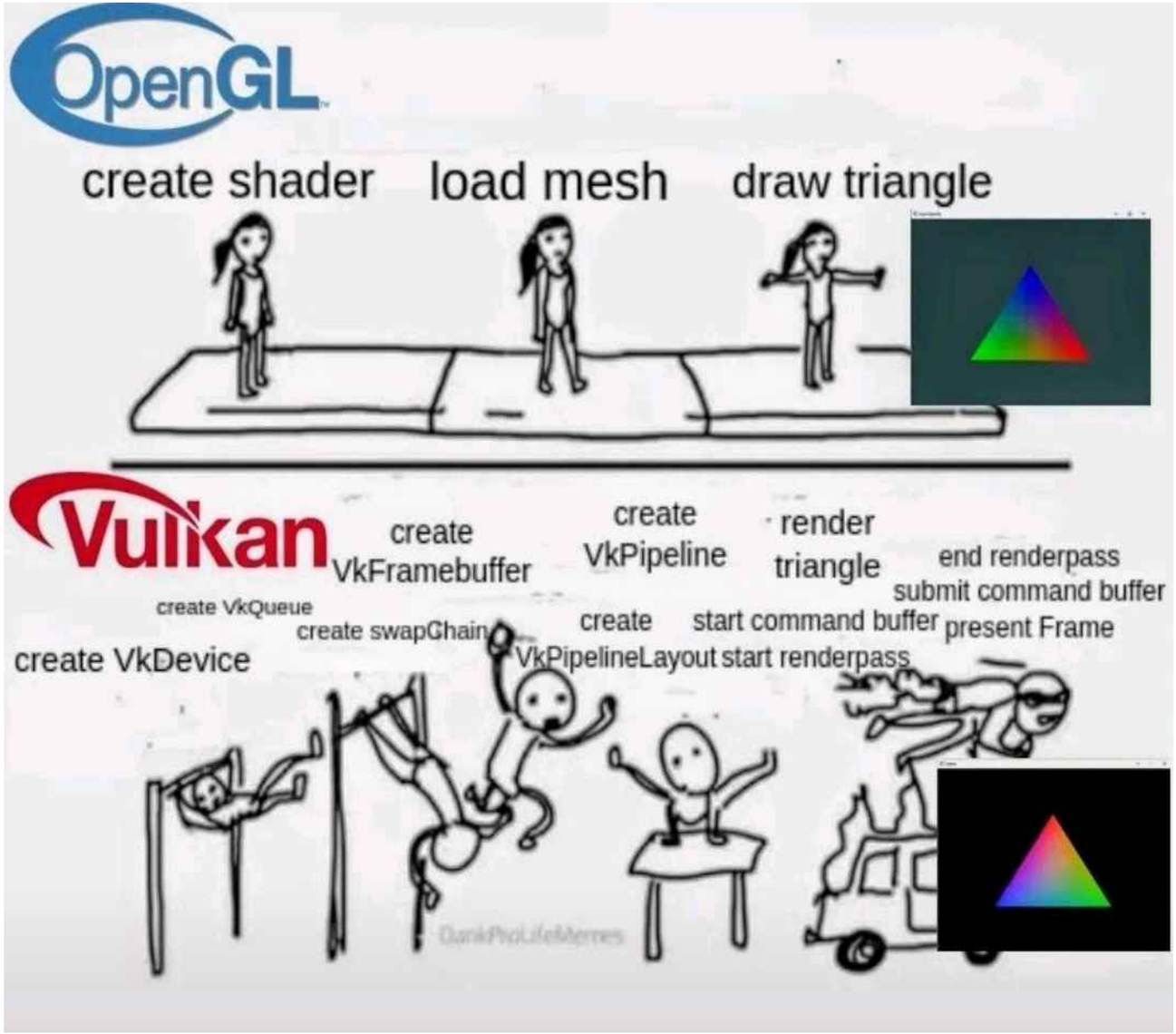
Vulkan Caps Viewer



Lifecycle of a vulkan application

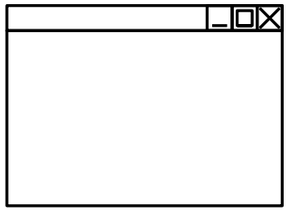


Lifecycle of a vulkan application



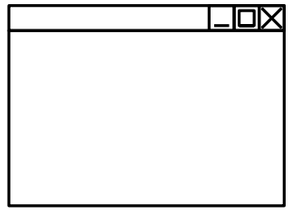
Initialization in Detail

1. Window Creation



Initialization in Detail

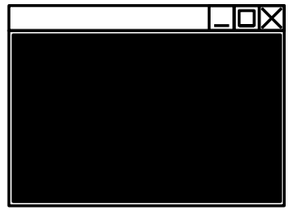
1. Window Creation



2. Vulkan Instance Creation

Initialization in Detail

1. Window Creation

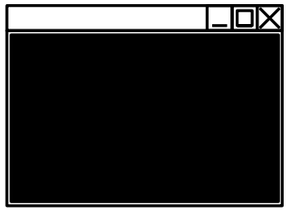


2. Vulkan Instance Creation

3. Surface Creation

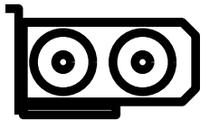
Initialization in Detail

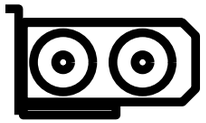
1. Window Creation
3. Surface Creation



2. Vulkan Instance Creation

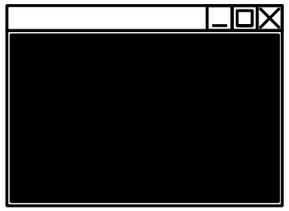
4. Select Physical Device





Initialization in Detail

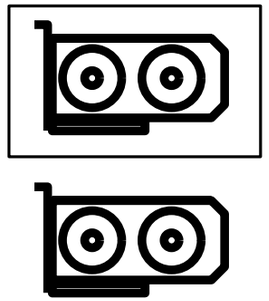
1. Window Creation



3. Surface Creation

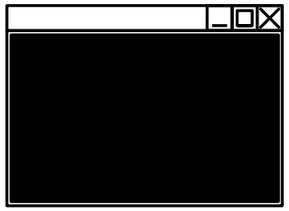
2. Vulkan Instance Creation

4. Select Physical Device



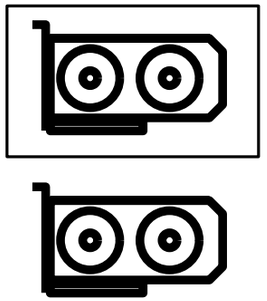
Initialization in Detail

1. Window Creation
3. Surface Creation



2. Vulkan Instance Creation

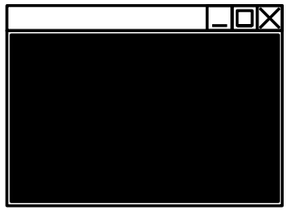
4. Select Physical Device



1. logical device
2. logical device
3. ...

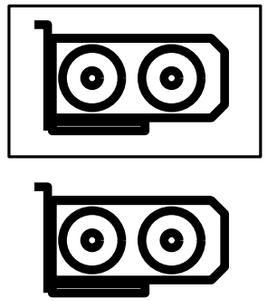
Initialization in Detail

1. Window Creation
3. Surface Creation



2. Vulkan Instance Creation

4. Select Physical Device

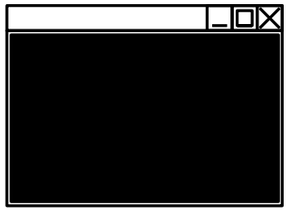


5. Select Logical Device

1. logical device
2. logical device
3. ...

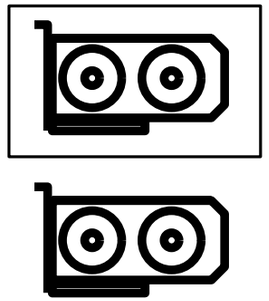
Initialization in Detail

1. Window Creation
3. Surface Creation



2. Vulkan Instance Creation

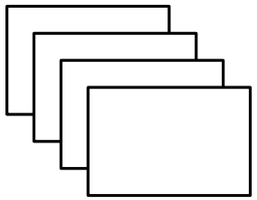
4. Select Physical Device



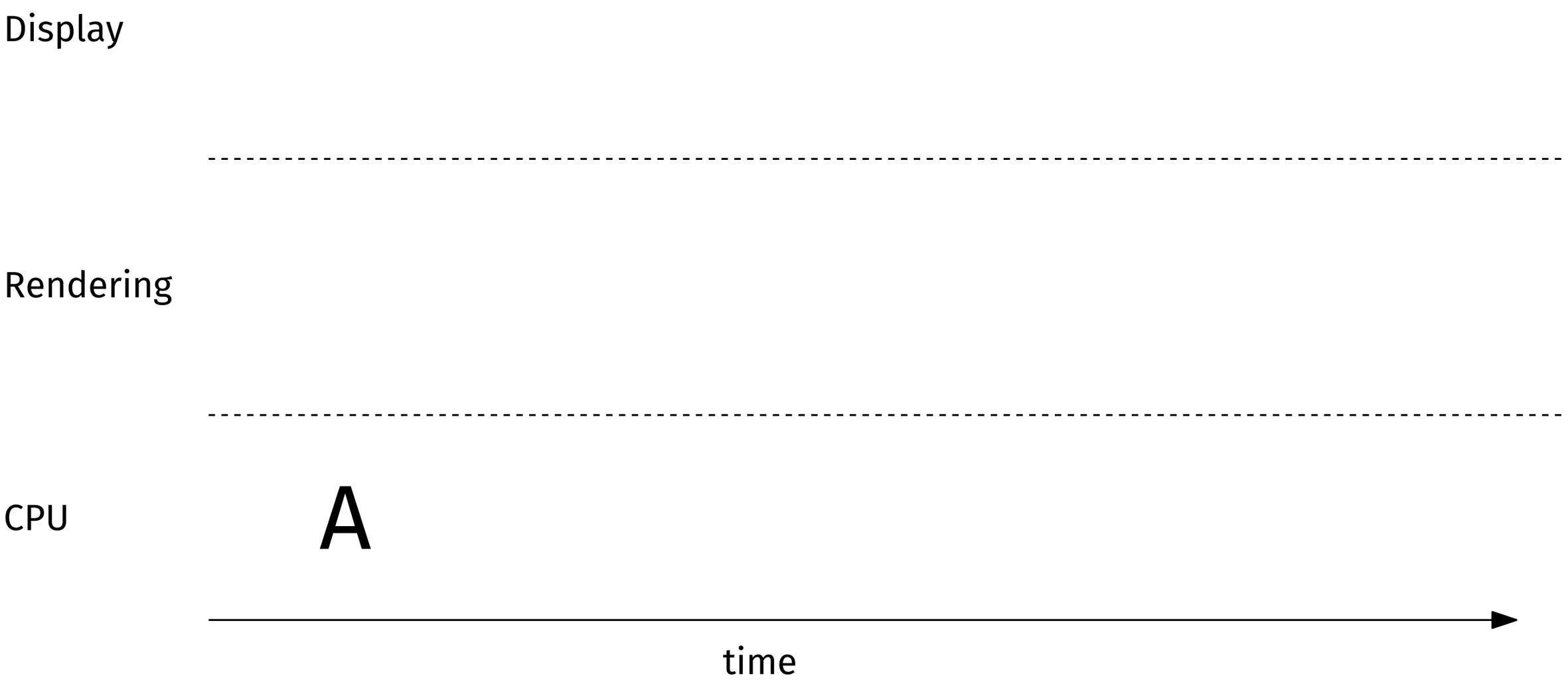
5. Select Logical Device

1. logical device
2. logical device
3. ...

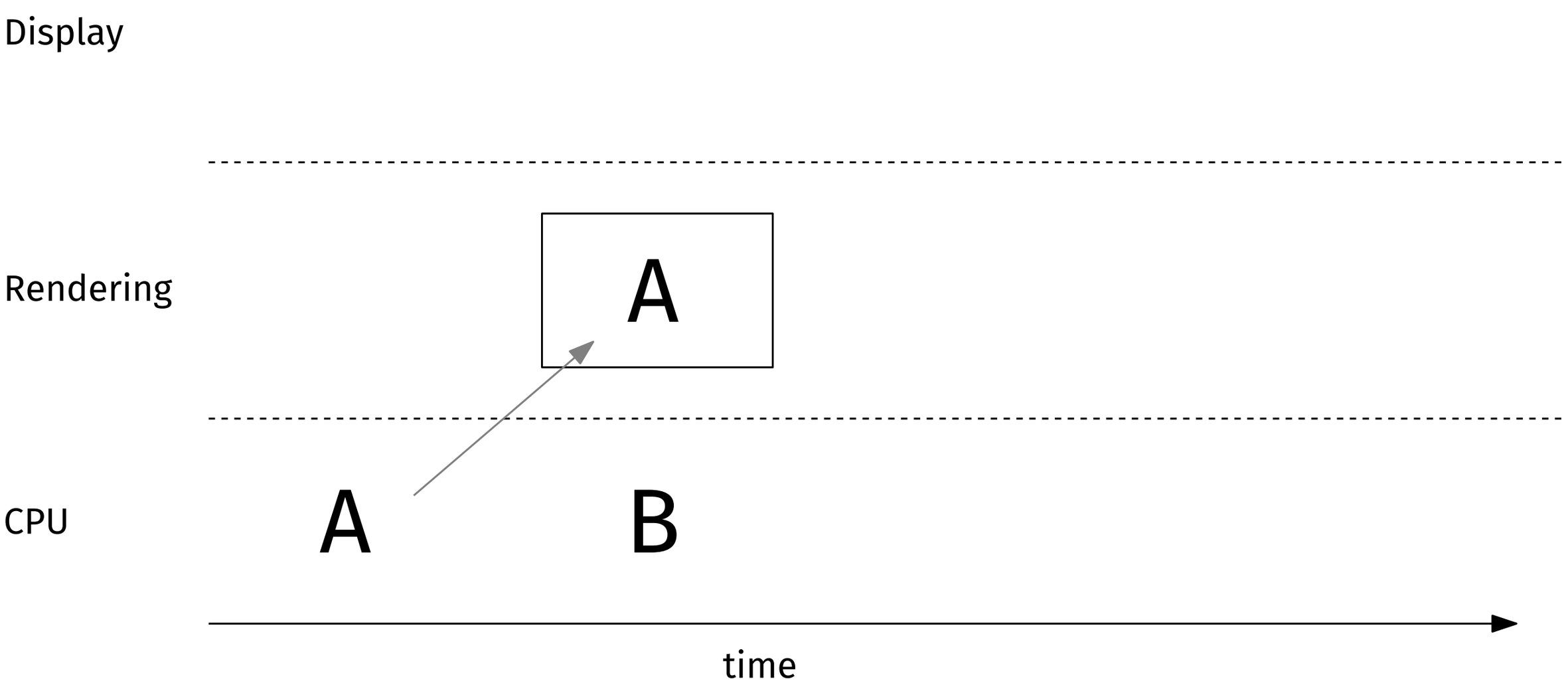
6. SwapChain Creation



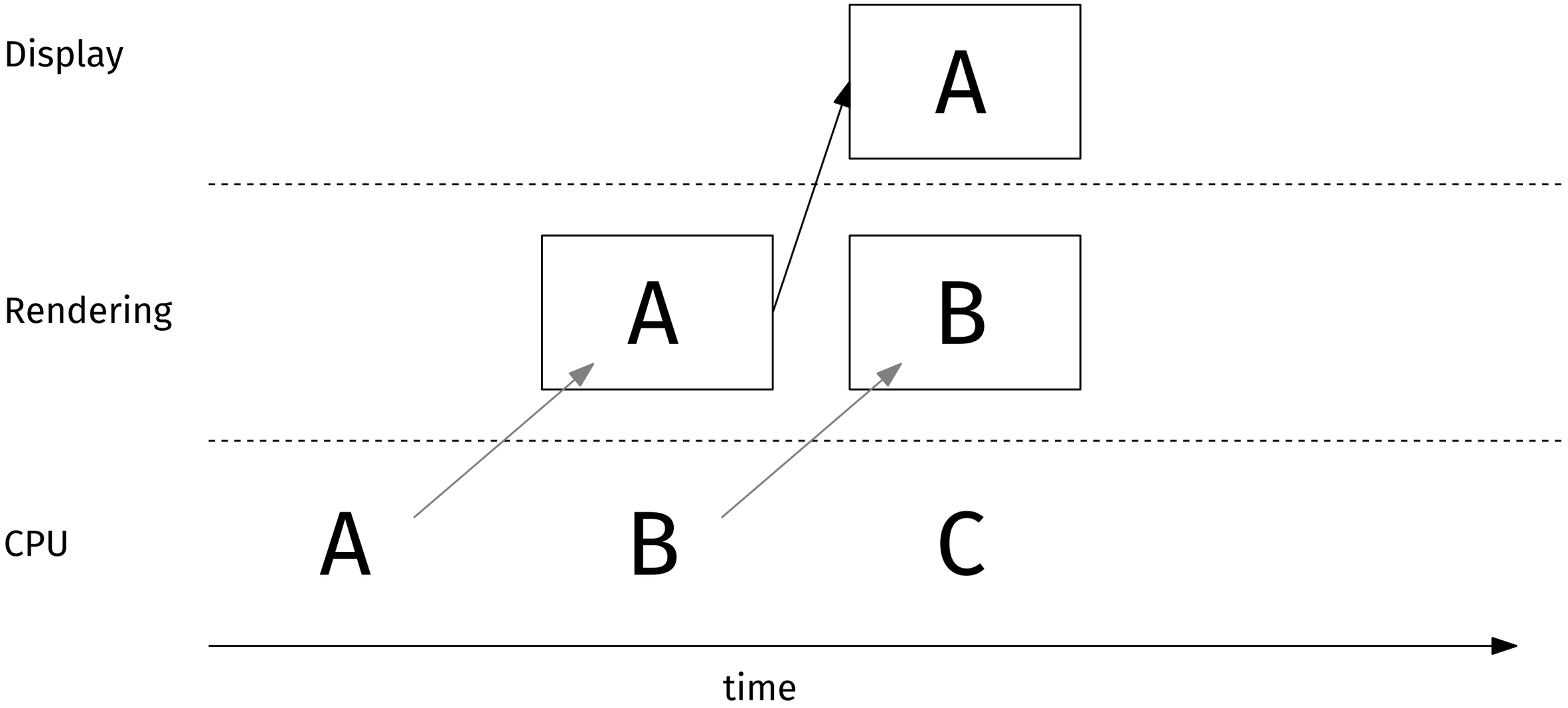
Swapchain - Frames in Flight



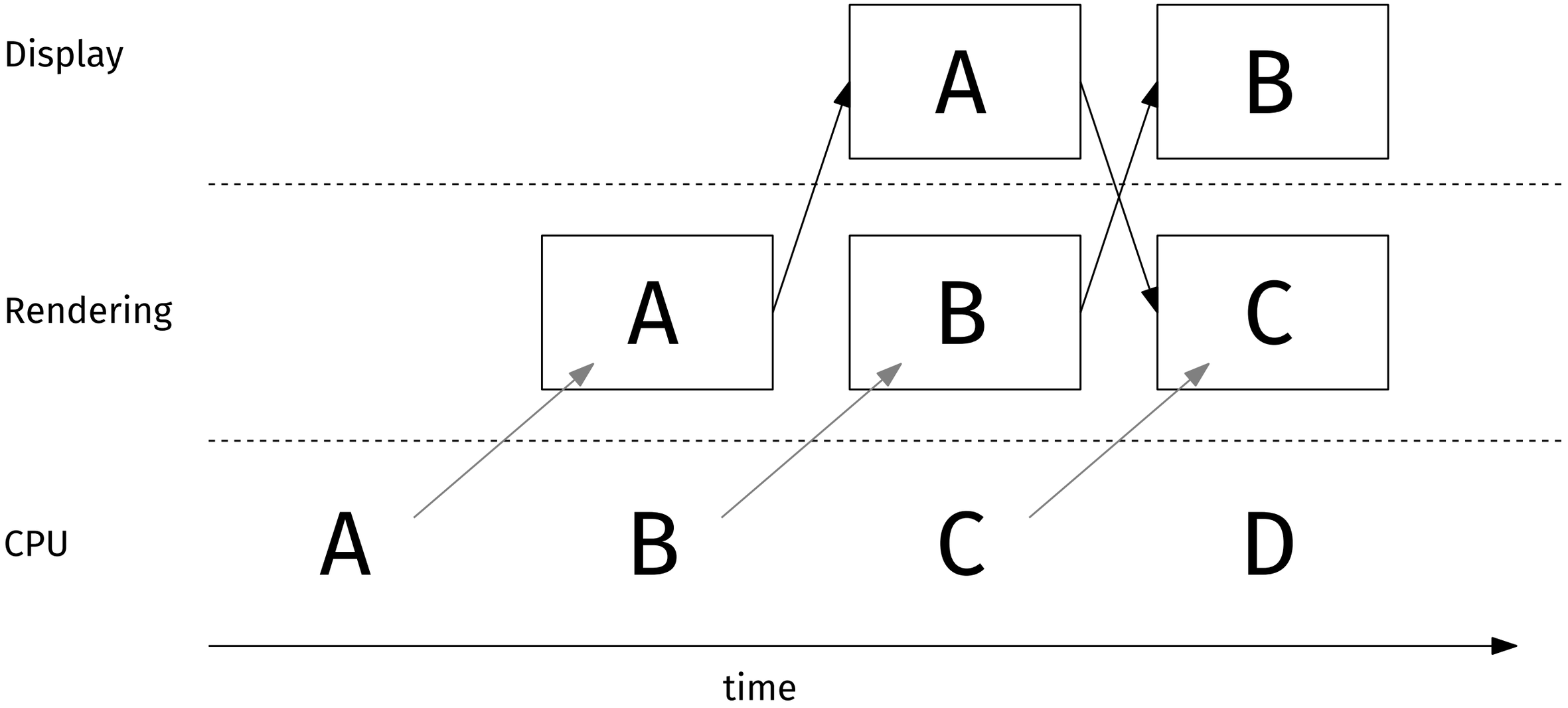
Swapchain - Frames in Flight



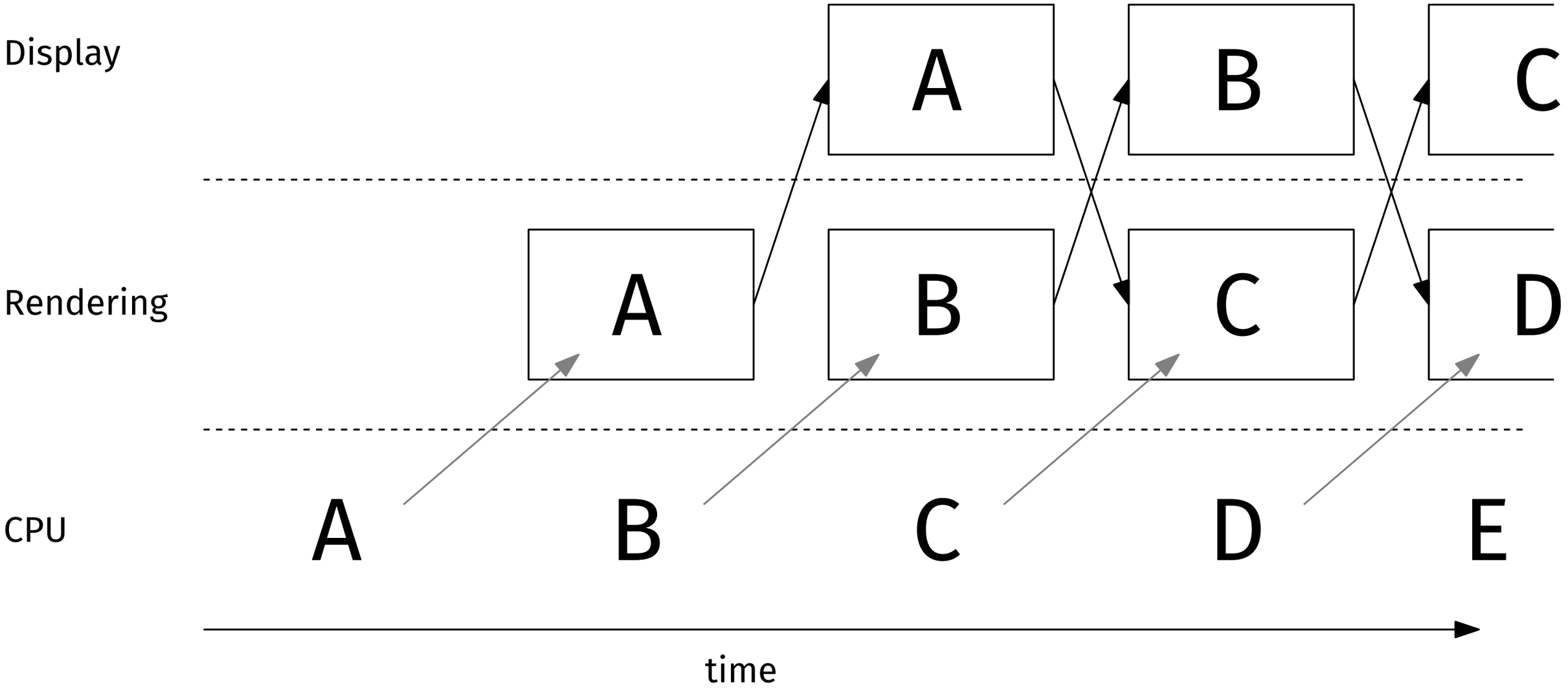
Swapchain - Frames in Flight



Swapchain - Frames in Flight

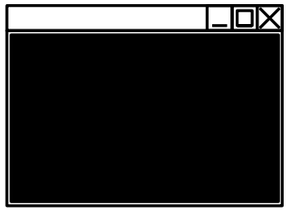


Swapchain - Frames in Flight



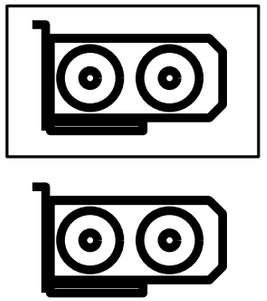
Initialization in Detail

1. Window Creation
3. Surface Creation



2. Vulkan Instance Creation

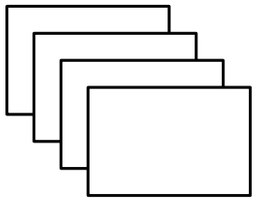
4. Select Physical Device



5. Select Logical Device

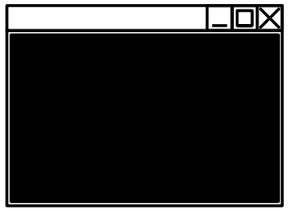
1. logical device
2. logical device
3. ...

6. SwapChain Creation



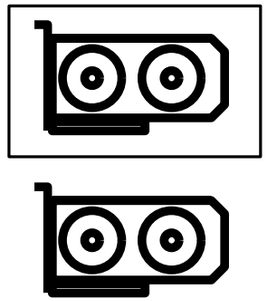
Initialization in Detail

1. Window Creation
3. Surface Creation



2. Vulkan Instance Creation

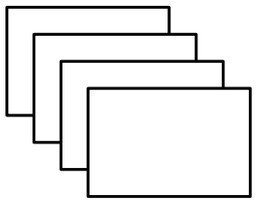
4. Select Physical Device



5. Select Logical Device

1. logical device
2. logical device
3. ...

6. SwapChain Creation

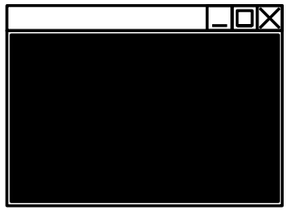


7. Command- & Descriptor Pool

$$\left\{ \begin{array}{l} \text{uniformbuffer} \times 10 \\ \text{imageSampler} \times 20 \end{array} \right\}$$

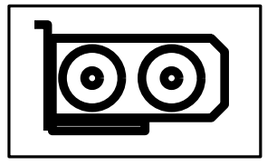
Initialization in Detail

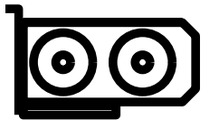
1. Window Creation
3. Surface Creation



2. Vulkan Instance Creation

4. Select Physical Device

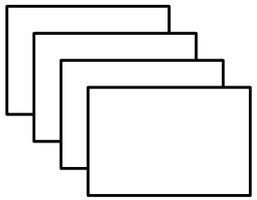




5. Select Logical Device

1. logical device
2. logical device
3. ...

6. SwapChain Creation



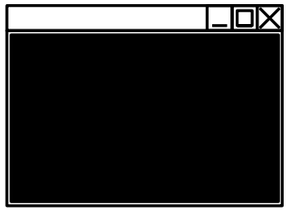
7. Command- & Descriptor Pool

$$\left\{ \begin{array}{l} \text{uniformbuffer} \times 10 \\ \text{imageSampler} \times 20 \end{array} \right\}$$

8. renderpass

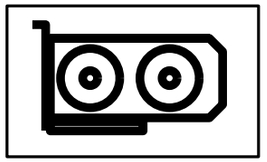
Initialization in Detail

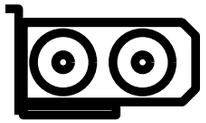
1. Window Creation
3. Surface Creation



2. Vulkan Instance Creation

4. Select Physical Device

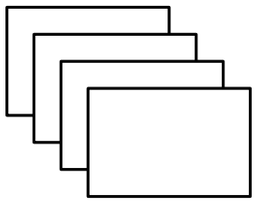




5. Select Logical Device

1. logical device
2. logical device
3. ...

6. SwapChain Creation

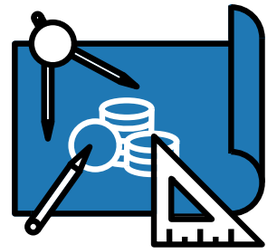


7. Command- & Descriptor Pool

$$\left\{ \begin{array}{l} \text{uniformbuffer} \times 10 \\ \text{imageSampler} \times 20 \end{array} \right\}$$

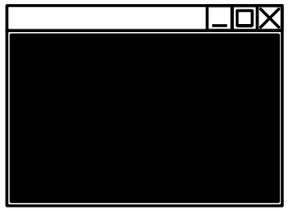
8. renderpass

9. DescriptorSetLayout



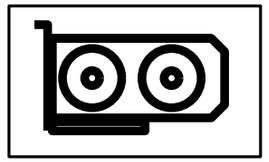
Initialization in Detail

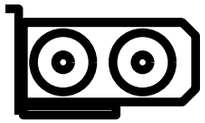
1. Window Creation
3. Surface Creation



2. Vulkan Instance Creation

4. Select Physical Device

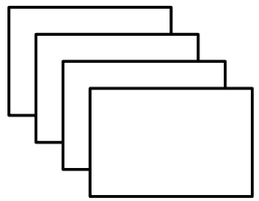




5. Select Logical Device

1. logical device
2. logical device
3. ...

6. SwapChain Creation

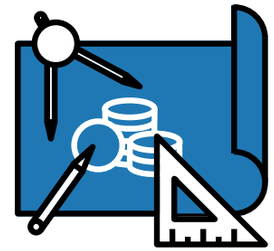


7. Command- & Descriptor Pool

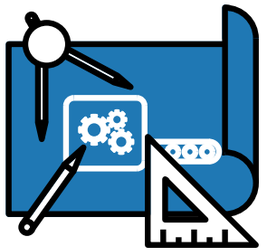
$$\left\{ \begin{array}{l} \text{uniformbuffer} \times 10 \\ \text{imageSampler} \times 20 \end{array} \right\}$$

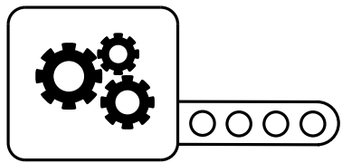
8. renderpass

9. DescriptorSetLayout



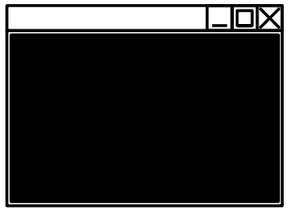
10. PipelineLayout & Pipeline Creation





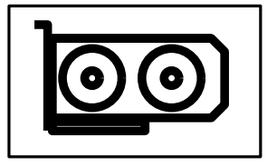
Initialization in Detail

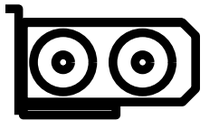
1. Window Creation
3. Surface Creation



2. Vulkan Instance Creation

4. Select Physical Device

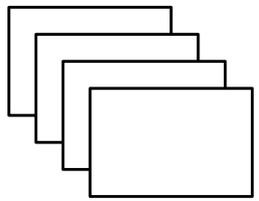




5. Select Logical Device

1. logical device
2. logical device
3. ...

6. SwapChain Creation

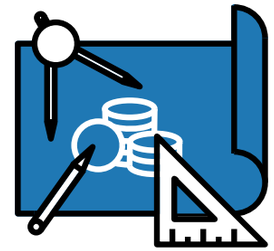


7. Command- & Descriptor Pool

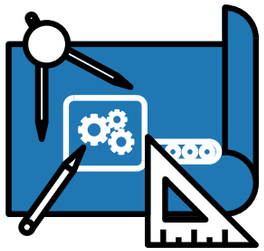
$$\left\{ \begin{array}{l} \text{uniformbuffer} \times 10 \\ \text{imageSampler} \times 20 \end{array} \right\}$$

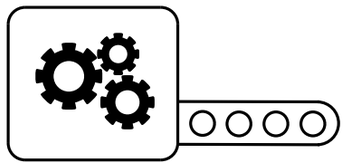
8. renderpass

9. DescriptorSetLayout



10. PipelineLayout & Pipeline Creation

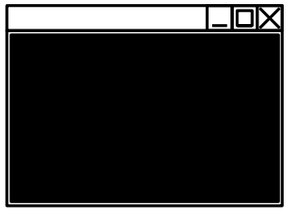




11. Descriptor-Set creation

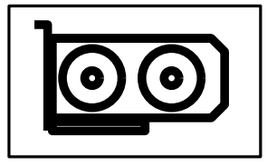
Initialization in Detail

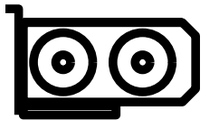
1. Window Creation
3. Surface Creation



2. Vulkan Instance Creation

4. Select Physical Device

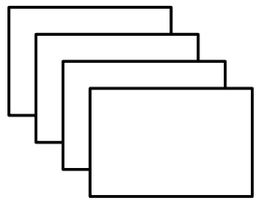




5. Select Logical Device

1. logical device
2. logical device
3. ...

6. SwapChain Creation

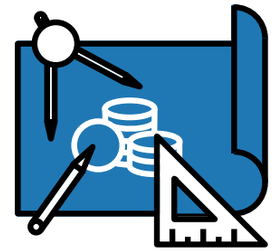


7. Command- & Descriptor Pool

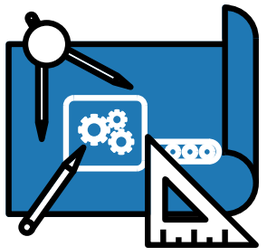
$$\left\{ \begin{array}{l} \text{uniformbuffer} \times 10 \\ \text{imageSampler} \times 20 \end{array} \right\}$$

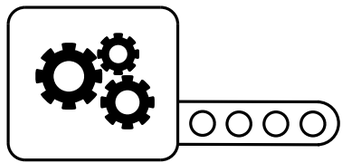
8. renderpass

9. DescriptorSetLayout



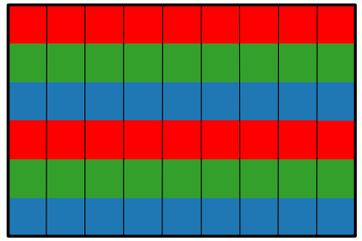
10. PipelineLayout & Pipeline Creation





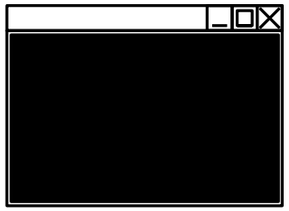
11. Descriptor-Set creation

12. Framebuffer creation



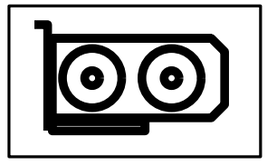
Initialization in Detail

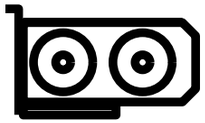
1. Window Creation
3. Surface Creation



2. Vulkan Instance Creation

4. Select Physical Device

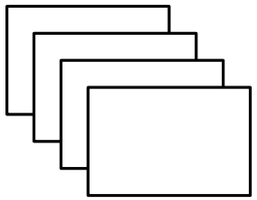




5. Select Logical Device

1. logical device
2. logical device
3. ...

6. SwapChain Creation

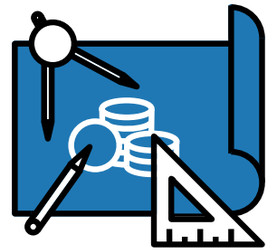


7. Command- & Descriptor Pool

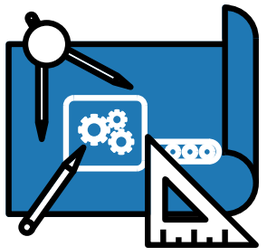
$$\left\{ \begin{array}{l} \text{uniformbuffer} \times 10 \\ \text{imageSampler} \times 20 \end{array} \right\}$$

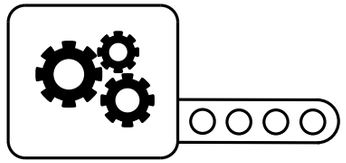
8. renderpass

9. DescriptorSetLayout



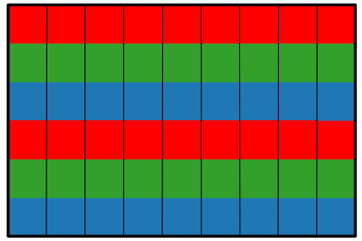
10. PipelineLayout & Pipeline Creation





11. Descriptor-Set creation

12. Framebuffer creation

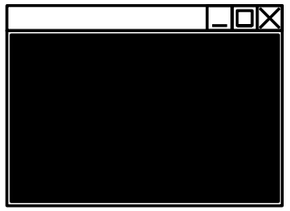


13. Command-buffer creation

```
pool1 = malloc . . .
pool2 = malloc . . .
```

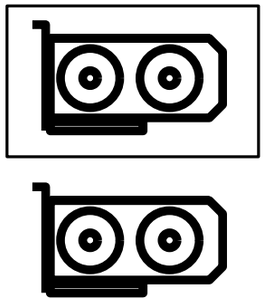
Initialization in Detail

1. Window Creation
3. Surface Creation



2. Vulkan Instance Creation

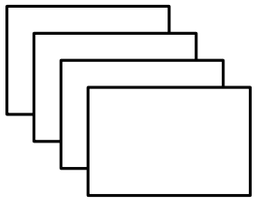
4. Select Physical Device



5. Select Logical Device

1. logical device
2. logical device
3. ...

6. SwapChain Creation

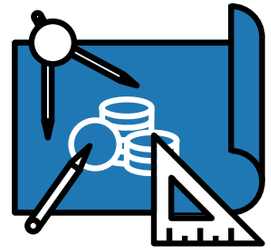


7. Command- & Descriptor Pool

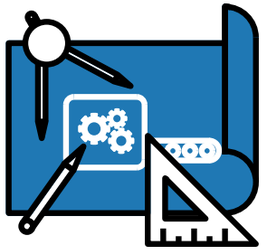
$$\left\{\begin{array}{l}\text{uniformbuffer} \times 10\\ \text{imageSampler} \times 20\end{array}\right\}$$

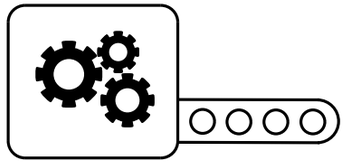
8. renderpass

9. DescriptorSetLayout



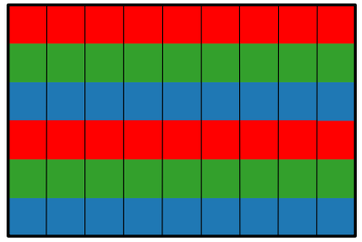
10. PipelineLayout & Pipeline Creation





11. Descriptor-Set creation

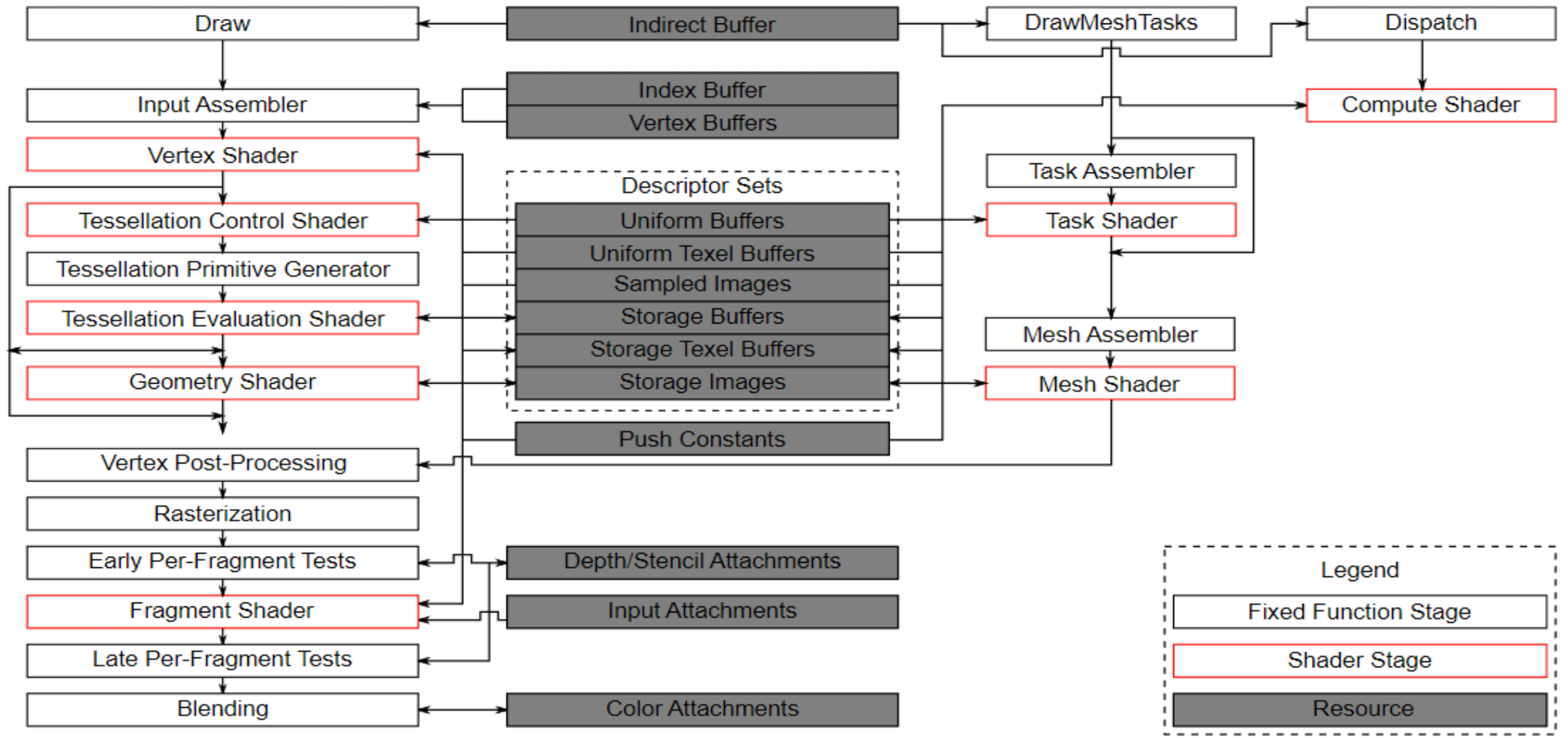
12. Framebuffer creation



13. Command-buffer creation

```
pool1 = malloc . . .
pool2 = malloc . . .
```

14. Sync-Object creation


Draw
Input Assembler
Vertex Shader
Tessellation Control Shader
Tessellation Primitive Generator
Tessellation Evaluation Shader
Geometry Shader
Vertex Post-Processing
Rasterization
Early Per-Fragment Tests
Fragment Shader
Late Per-Fragment Tests
Blending
Indirect Buffer
Index Buffer
Vertex Buffers
Descriptor Sets
Uniform Buffers
Uniform Texel Buffers
Sampled Images
Storage Buffers
Storage Texel Buffers
Storage Images
Push Constants
Depth/Stencil Attachments
Input Attachments
Color Attachments
DrawMeshTasks
Dispatch
Compute Shader
Task Assembler
Task Shader
Mesh Assembler
Mesh Shader
Legend
Fixed Function Stage
Shader Stage
Resource

C-API

```
void createInstance() {
    VkApplicationInfo appInfo{};
    appInfo.sType = VK_STRUCTURE_TYPE_APPLICATION_INFO;
    appInfo.pApplicationName = "Hello Triangle";
    appInfo.applicationVersion = VK_MAKE_VERSION(1, 0, 0);
    appInfo.pEngineName = "No Engine";
    appInfo.engineVersion = VK_MAKE_VERSION(1, 0, 0);
    appInfo.apiVersion = VK_API_VERSION_1_0;

    VkInstanceCreateInfo createInfo{};
    createInfo.sType = VK_STRUCTURE_TYPE_INSTANCE_CREATE_INFO;
    createInfo.pApplicationInfo = &appInfo;

    uint32_t glfwExtensionCount = 0;
    const char** glfwExtensions;
    glfwExtensions = glfwGetRequiredInstanceExtensions(&glfwExtensionCount);

    createInfo.enabledExtensionCount = glfwExtensionCount;
    createInfo.ppEnabledExtensionNames = glfwExtensions;

    createInfo.enabledLayerCount = 0;

    if (vkCreateInstance(&createInfo, nullptr, &instance) != VK_SUCCESS) {
        throw std::runtime_error("failed to create instance!");
    }
}
```

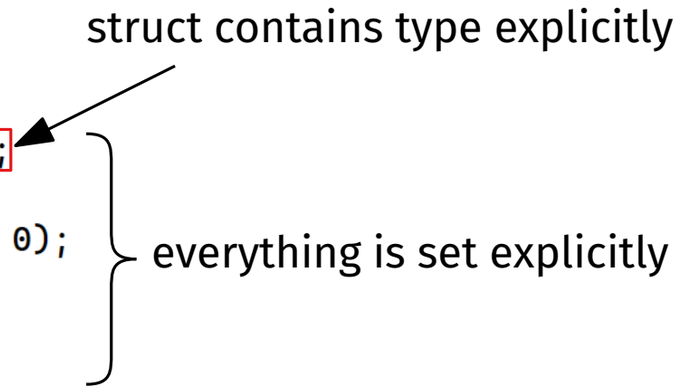
C-API

struct contains type explicitly

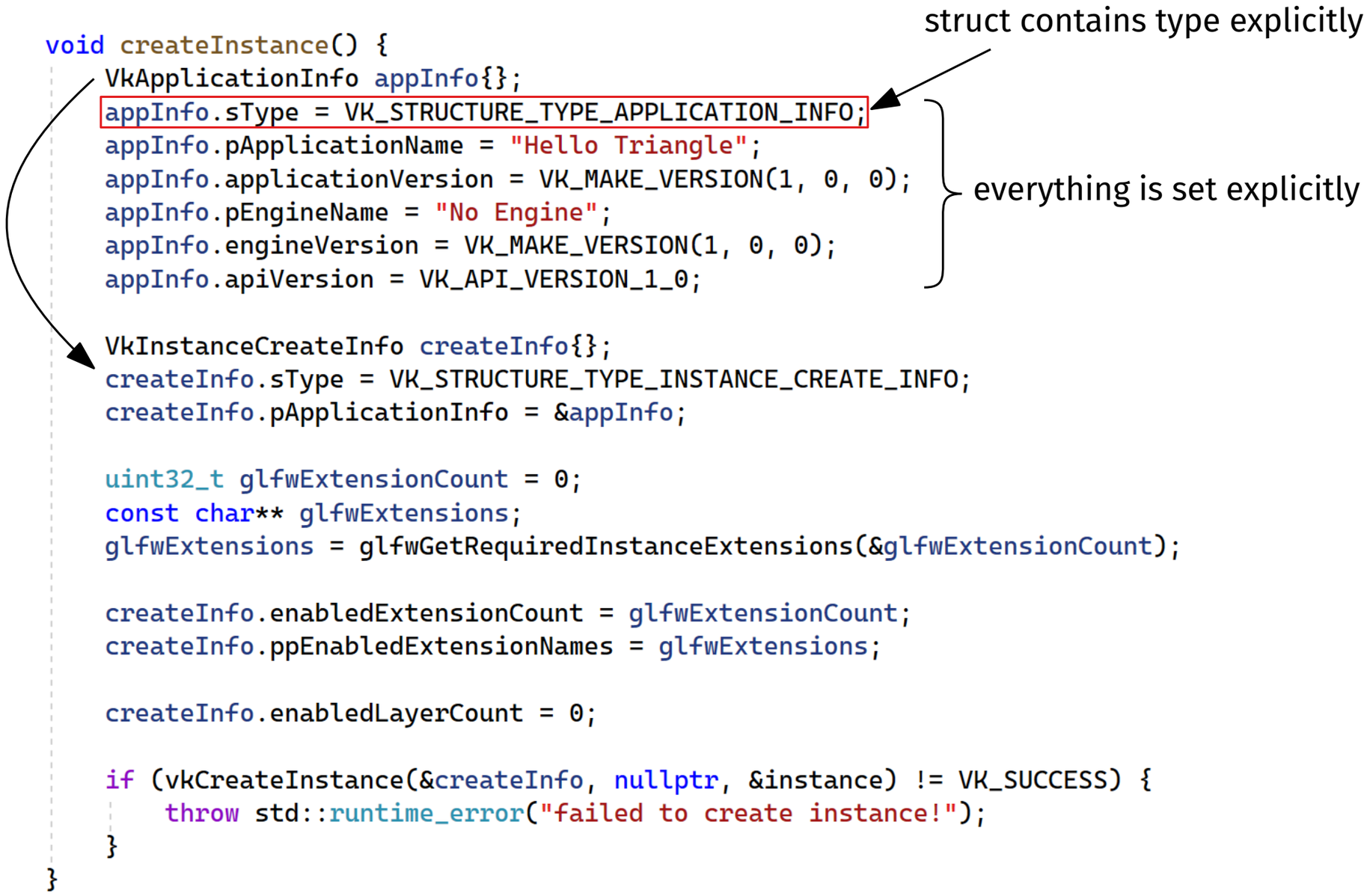
```
void createInstance() {
    VkApplicationInfo appInfo{};
    appInfo.sType = VK_STRUCTURE_TYPE_APPLICATION_INFO;
    appInfo.pApplicationName = "Hello Triangle";
    appInfo.applicationVersion = VK_MAKE_VERSION(1, 0, 0);
    appInfo.pEngineName = "No Engine";
    appInfo.engineVersion = VK_MAKE_VERSION(1, 0, 0);
    appInfo.apiVersion = VK_API_VERSION_1_0;

    VkInstanceCreateInfo createInfo{};
    createInfo.sType = VK_STRUCTURE_TYPE_INSTANCE_CREATE_INFO;
    createInfo.pApplicationInfo = &appInfo;

    uint32_t glfwExtensionCount = 0;
    const char** glfwExtensions;
    glfwExtensions = glfwGetRequiredInstanceExtensions(&glfwExtensionCount);

    createInfo.enabledExtensionCount = glfwExtensionCount;
    createInfo.ppEnabledExtensionNames = glfwExtensions;

    createInfo.enabledLayerCount = 0;

    if (vkCreateInstance(&createInfo, nullptr, &instance) != VK_SUCCESS) {
        throw std::runtime_error("failed to create instance!");
    }
}
```

C-API

```
void createInstance() {
    VkApplicationInfo appInfo{};
    appInfo.sType = VK_STRUCTURE_TYPE_APPLICATION_INFO;
    appInfo.pApplicationName = "Hello Triangle";
    appInfo.applicationVersion = VK_MAKE_VERSION(1, 0, 0);
    appInfo.pEngineName = "No Engine";
    appInfo.engineVersion = VK_MAKE_VERSION(1, 0, 0);
    appInfo.apiVersion = VK_API_VERSION_1_0;

    VkInstanceCreateInfo createInfo{};
    createInfo.sType = VK_STRUCTURE_TYPE_INSTANCE_CREATE_INFO;
    createInfo.pApplicationInfo = &appInfo;

    uint32_t glfwExtensionCount = 0;
    const char** glfwExtensions;
    glfwExtensions = glfwGetRequiredInstanceExtensions(&glfwExtensionCount);

    createInfo.enabledExtensionCount = glfwExtensionCount;
    createInfo.ppEnabledExtensionNames = glfwExtensions;

    createInfo.enabledLayerCount = 0;

    if (vkCreateInstance(&createInfo, nullptr, &instance) != VK_SUCCESS) {
        throw std::runtime_error("failed to create instance!");
    }
}
```



C-API




```
void createInstance() {
    VkApplicationInfo appInfo{};
    appInfo.sType = VK_STRUCTURE_TYPE_APPLICATION_INFO;
    appInfo.pApplicationName = "Hello Triangle";
    appInfo.applicationVersion = VK_MAKE_VERSION(1, 0, 0);
    appInfo.pEngineName = "No Engine";
    appInfo.engineVersion = VK_MAKE_VERSION(1, 0, 0);
    appInfo.apiVersion = VK_API_VERSION_1_0;

    VkInstanceCreateInfo createInfo{};
    createInfo.sType = VK_STRUCTURE_TYPE_INSTANCE_CREATE_INFO;
    createInfo.pApplicationInfo = &appInfo;

    uint32_t glfwExtensionCount = 0;
    const char** glfwExtensions;
    glfwExtensions = glfwGetRequiredInstanceExtensions(&glfwExtensionCount);

    createInfo.enabledExtensionCount = glfwExtensionCount;
    createInfo.ppEnabledExtensionNames = glfwExtensions;

    createInfo.enabledLayerCount = 0;

    if (vkCreateInstance(&createInfo, nullptr, &instance) != VK_SUCCESS) {
        throw std::runtime_error("failed to create instance!");
    }
}
```

C-API

```
void createInstance() {
    VkApplicationInfo appInfo{};
    appInfo.sType = VK_STRUCTURE_TYPE_APPLICATION_INFO;
    appInfo.pApplicationName = "Hello Triangle";
    appInfo.applicationVersion = VK_MAKE_VERSION(1, 0, 0);
    appInfo.pEngineName = "No Engine";
    appInfo.engineVersion = VK_MAKE_VERSION(1, 0, 0);
    appInfo.apiVersion = VK_API_VERSION_1_0;

    VkInstanceCreateInfo createInfo{};
    createInfo.sType = VK_STRUCTURE_TYPE_INSTANCE_CREATE_INFO;
    createInfo.pApplicationInfo = &appInfo;

    uint32_t glfwExtensionCount = 0;
    const char** glfwExtensions;
    glfwExtensions = glfwGetRequiredInstanceExtensions(&glfwExtensionCount);

    createInfo.enabledExtensionCount = glfwExtensionCount;
    createInfo.ppEnabledExtensionNames = glfwExtensions;

    createInfo.enabledLayerCount = 0;

    if (vkCreateInstance(&createInfo, nullptr, &instance) != VK_SUCCESS) {
        throw std::runtime_error("failed to create instance!");
    }
}
```

struct contains type explicitly

everything is set explicitly

C-API

```
void createInstance() {
    VkApplicationInfo appInfo{};
    appInfo.sType = VK_STRUCTURE_TYPE_APPLICATION_INFO;
    appInfo.pApplicationName = "Hello Triangle";
    appInfo.applicationVersion = VK_MAKE_VERSION(1, 0, 0);
    appInfo.pEngineName = "No Engine";
    appInfo.engineVersion = VK_MAKE_VERSION(1, 0, 0);
    appInfo.apiVersion = VK_API_VERSION_1_0;

    VkInstanceCreateInfo createInfo{};
    createInfo.sType = VK_STRUCTURE_TYPE_INSTANCE_CREATE_INFO;
    createInfo.pApplicationInfo = &appInfo;

    uint32_t glfwExtensionCount = 0;
    const char** glfwExtensions;
    glfwExtensions = glfwGetRequiredInstanceExtensions(&glfwExtensionCount);

    createInfo.enabledExtensionCount = glfwExtensionCount;
    createInfo.ppEnabledExtensionNames = glfwExtensions;

    createInfo.enabledLayerCount = 0;

    if (vkCreateInstance(&createInfo, nullptr, &instance) != VK_SUCCESS) {
        throw std::runtime_error("failed to create instance!");
    }
}
```

struct contains type explicitly

everything is set explicitly

handle to vulkan API

C++ header

```
vk::ApplicationInfo applicationInfo("Hello Triangle", 1, "No Engine", 1, VK_MAKE_VERSION(1, 2, 0));

vk::InstanceCreateInfo instanceCreateInfo({}, &applicationInfo,
    enableValidationLayers ? validationLayers.size() : 0,
    enableValidationLayers ? validationLayers.data() : nullptr,
    glfwExtensionCount, glfwExtensions, nullptr);

res = vk::createInstance(&instanceCreateInfo, nullptr, &vkObject.vkInstance);
if (res != vk::Result::eSuccess) {
    throw std::runtime_error(std::format("failed to create vkInstance, error {}", (int)res));
}
```

C++ header

namespace vk

```
vk::ApplicationInfo applicationInfo("Hello Triangle", 1, "No Engine", 1, VK_MAKE_VERSION(1, 2, 0));

vk::InstanceCreateInfo instanceCreateInfo({}, &applicationInfo,
    enableValidationLayers ? validationLayers.size() : 0,
    enableValidationLayers ? validationLayers.data() : nullptr,
    glfwExtensionCount, glfwExtensions, nullptr);

res = vk::createInstance(&instanceCreateInfo, nullptr, &vkObject.vkInstance);
if (res != vk::Result::eSuccess) {
    throw std::runtime_error(std::format("failed to create vkInstance, error {}", (int)res));
}
```

C++ header

namespace vk

no explicit type as parameter

```
vk::ApplicationInfo applicationInfo("Hello Triangle", 1, "No Engine", 1, VK_MAKE_VERSION(1, 2, 0));

vk::InstanceCreateInfo instanceCreateInfo({}, &applicationInfo,
    enableValidationLayers ? validationLayers.size() : 0,
    enableValidationLayers ? validationLayers.data() : nullptr,
    glfwExtensionCount, glfwExtensions, nullptr);

res = vk::createInstance(&instanceCreateInfo, nullptr, &vkObject.vkInstance);
if (res != vk::Result::eSuccess) {
    throw std::runtime_error(std::format("failed to create vkInstance, error {}", (int)res));
}
```

Pick Physical Device

```
uint32_t deviceCount = 0;
vk::Result res = vkObject.vkInstance.enumeratePhysicalDevices(&deviceCount, nullptr);
```

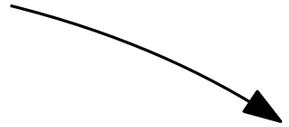
Pick Physical Device

deviceCount reference, nullptr as collection
⇒ stores count in deviceCount

```
uint32_t deviceCount = 0;
vk::Result res = vkObject.vkInstance.enumeratePhysicalDevices(&deviceCount, nullptr);
```

Pick Physical Device

deviceCount reference, nullptr as collection
⇒ stores count in deviceCount



```
uint32_t deviceCount = 0;
vk::Result res = vkObject.vkInstance.enumeratePhysicalDevices(&deviceCount, nullptr);
if (deviceCount == 0) {
    throw std::runtime_error("failed to find GPUs with Vulkan support!");
}
```

Pick Physical Device

deviceCount reference, nullptr as collection
⇒ stores count in deviceCount

```
uint32_t deviceCount = 0;
vk::Result res = vkObject.vkInstance.enumeratePhysicalDevices(&deviceCount, nullptr);
if (deviceCount == 0) {
    throw std::runtime_error("failed to find GPUs with Vulkan support!");
}
std::vector<vk::PhysicalDevice> devices(deviceCount);
res = vkObject.vkInstance.enumeratePhysicalDevices(&deviceCount, devices.data());
```

get deviceCount many devices

Wie kommen die Daten auf die GPU?

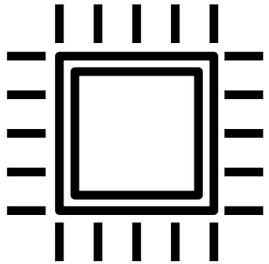
High-Level OpenGL:

```
GLfloat g_vertex_buffer_data[3 * 36] = {
        -1.0f,-1.0f,-1.0f,
        -1.0f,-1.0f, 1.0f,
        -1.0f, 1.0f, 1.0f,
        …
```

```
glGenBuffers(1, &this->vertexbuffer);
glBindBuffer(GL_ARRAY_BUFFER, this->vertexbuffer);
glBufferData(GL_ARRAY_BUFFER, sizeof(this->g_vertex_buffer_data),
    this->g_vertex_buffer_data, GL_STATIC_DRAW);
```

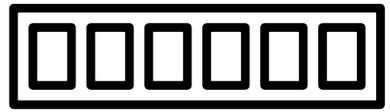
Wie kommen die Daten auf die GPU?

CPU

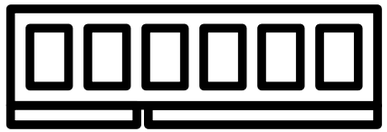


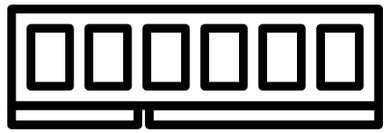
main.cpp

VRAM

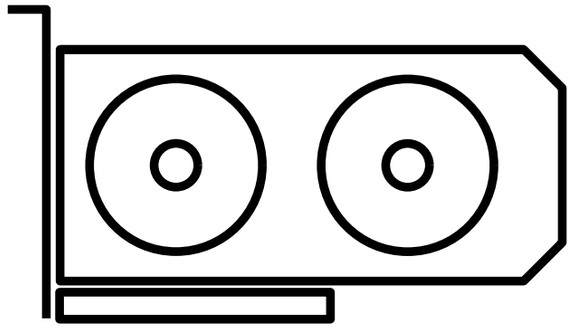


RAM



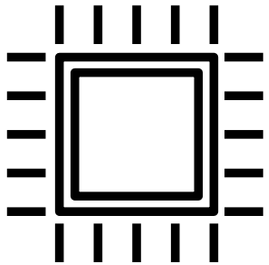


GPU



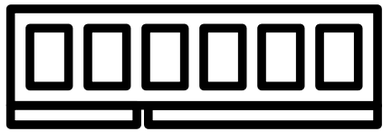
Wie kommen die Daten auf die GPU?

CPU

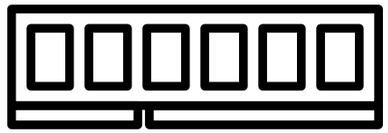


main.cpp

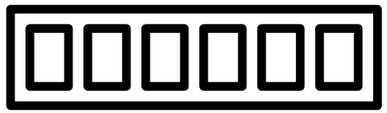
RAM



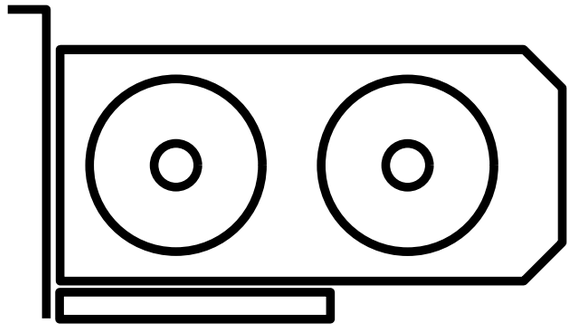
```
const std::vector<Vertex> vertices = {
        {{-0.5f, -0.5f, 0.0f}, ...
};
```



VRAM

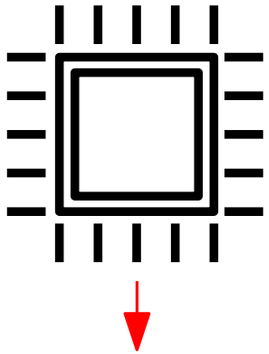


GPU



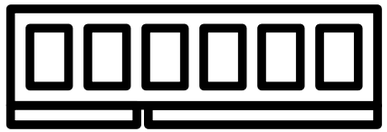
Wie kommen die Daten auf die GPU?

CPU



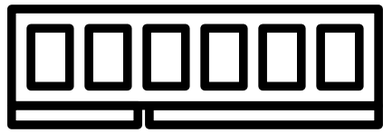
main.cpp

RAM



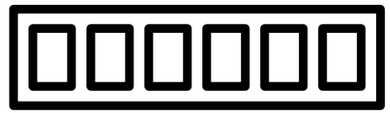
```
const std::vector<Vertex> vertices = {
        {{-0.5f, -0.5f, 0.0f}, ...
};
```

Memory-Mapped

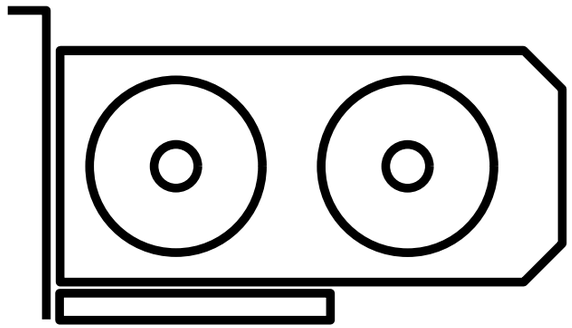


```
vk::Buffer stagingBuffer(size,
TransferSrc, HostVisible | HostCoherent)
```

VRAM

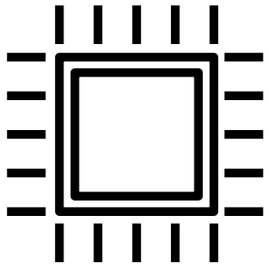


GPU



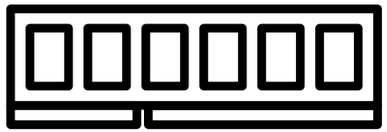
Wie kommen die Daten auf die GPU?

CPU



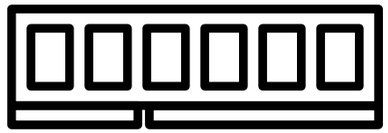
main.cpp

RAM



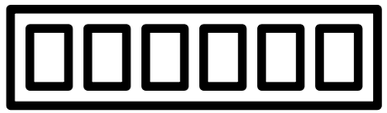
```
const std::vector<Vertex> vertices = {
        {{-0.5f, -0.5f, 0.0f}, ...
};
```

Memory-Mapped

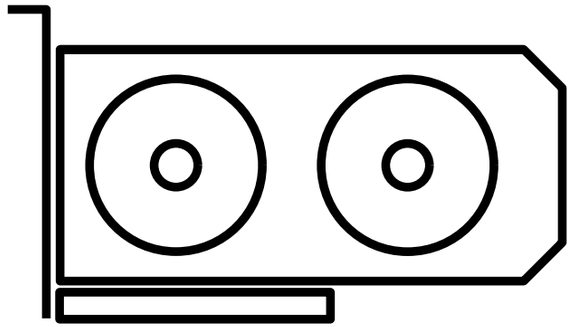


```
vk::Buffer stagingBuffer(size,
TransferSrc, HostVisible | HostCoherent)
```

VRAM

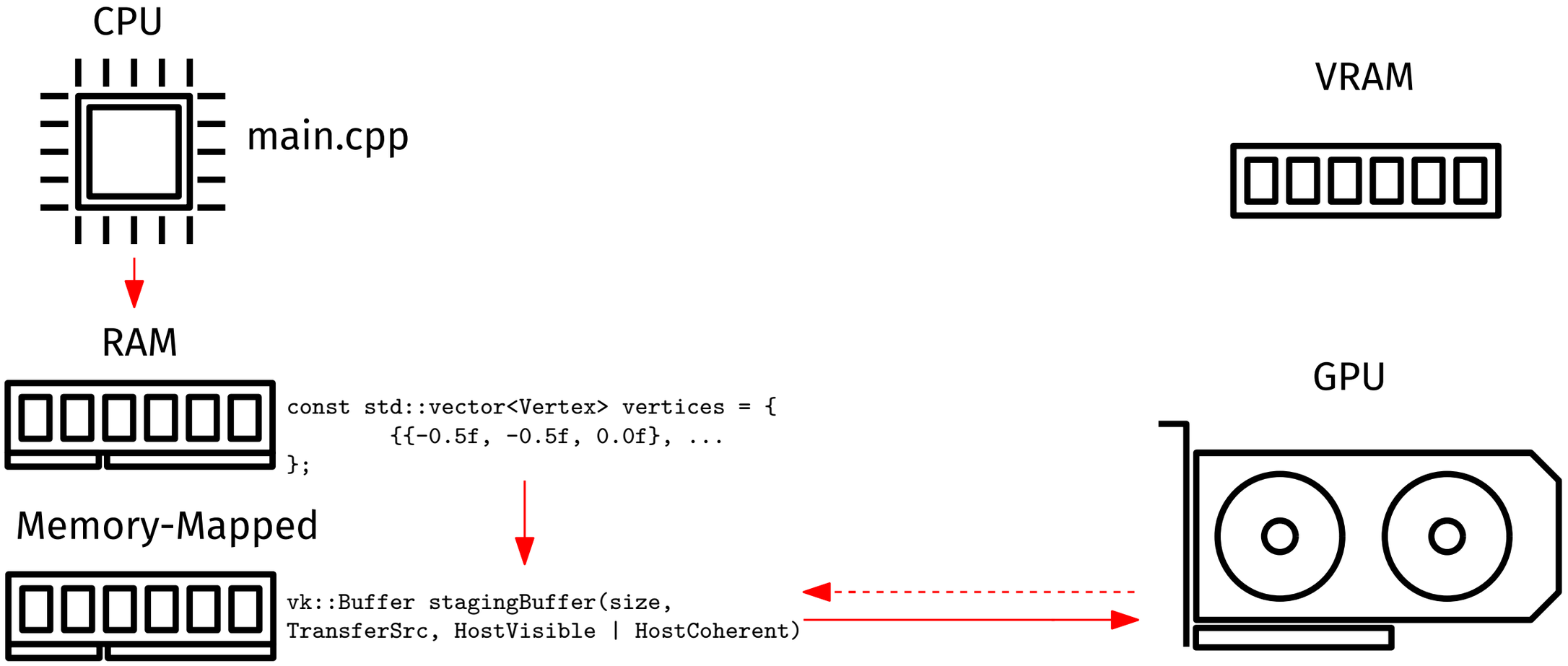


GPU

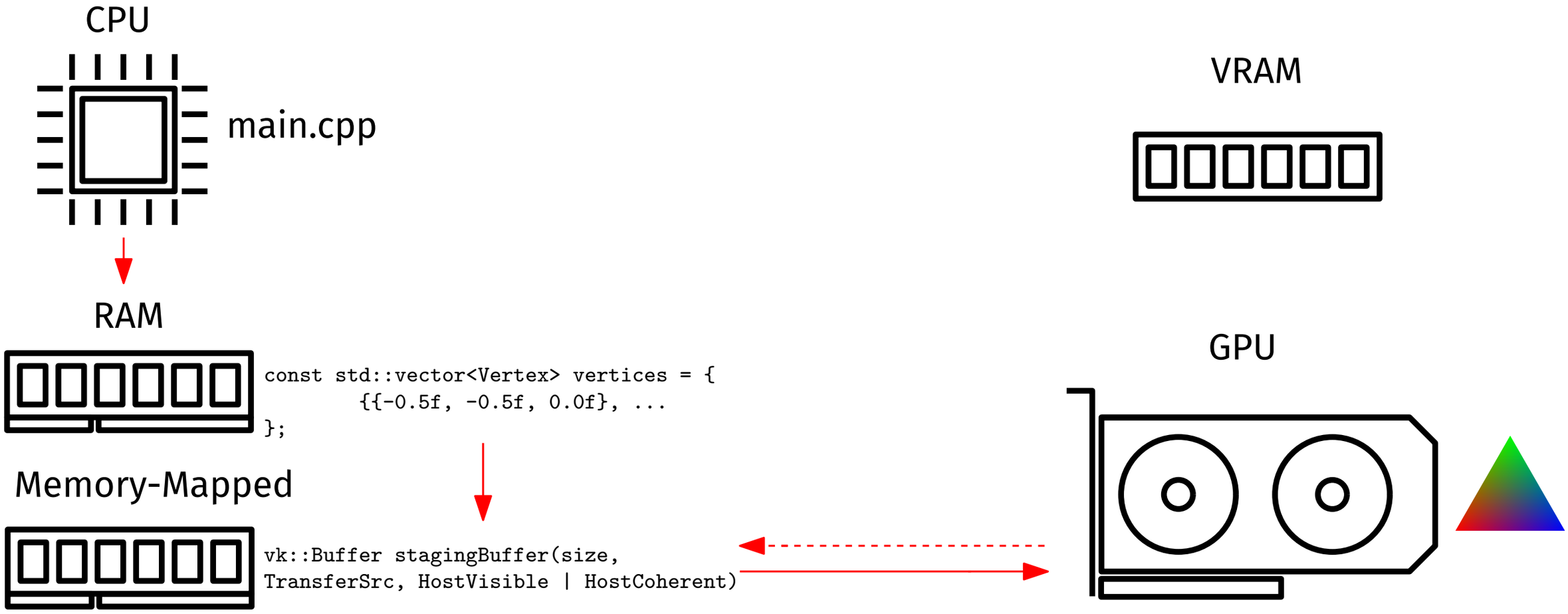


Wie kommen die Daten auf die GPU?

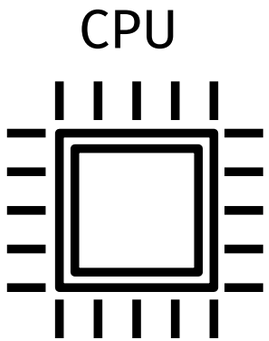
Wie kommen die Daten auf die GPU?



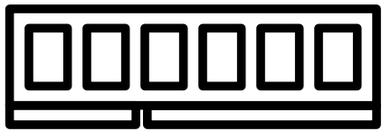
Wie kommen die Daten auf die GPU?



Wie kommen die Daten auf die GPU?

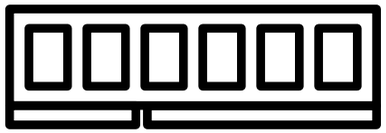


main.cpp



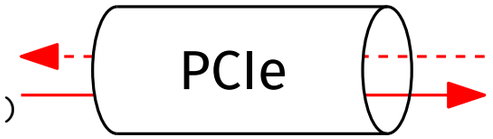

```
const std::vector<Vertex> vertices = {
        {{-0.5f, -0.5f, 0.0f}, ...
};
```

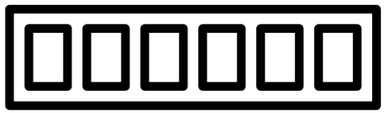
Memory-Mapped

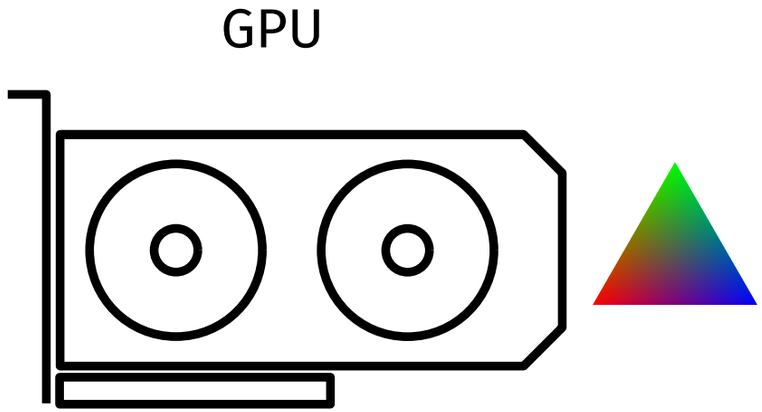


```
vk::Buffer stagingBuffer(size,
TransferSrc, HostVisible | HostCoherent)
```

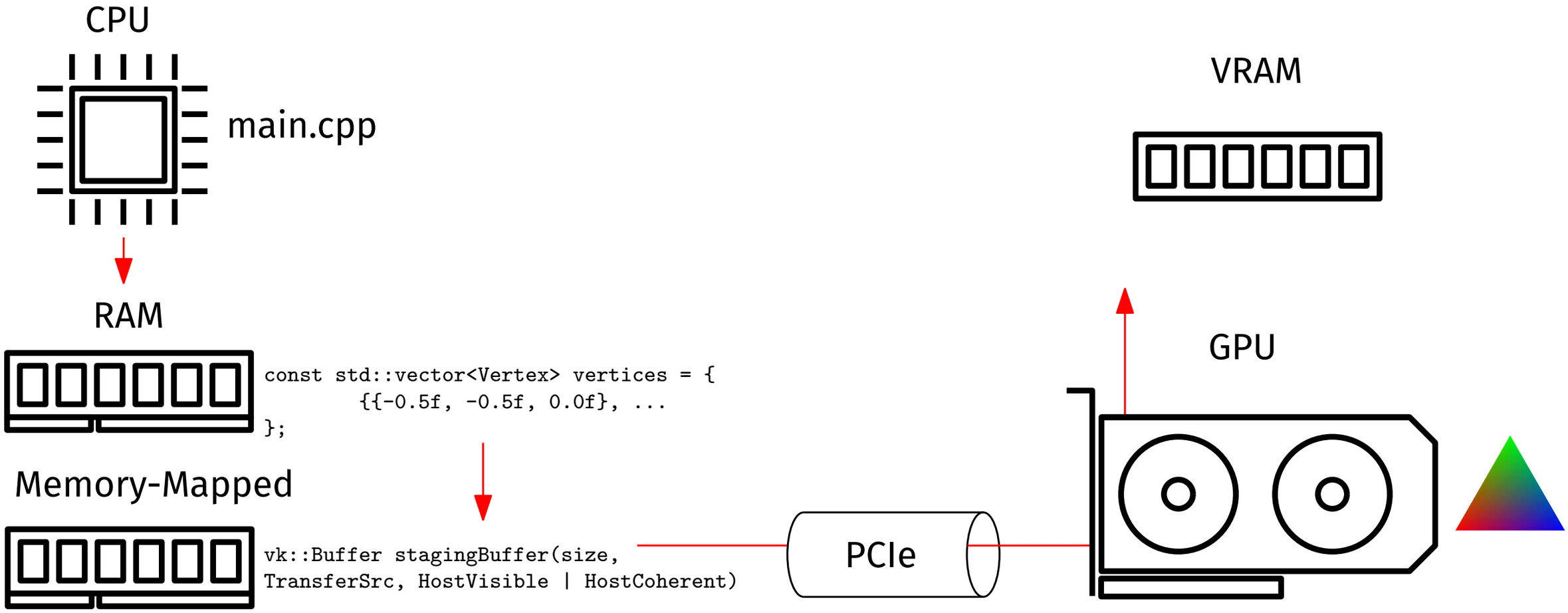
Problem:



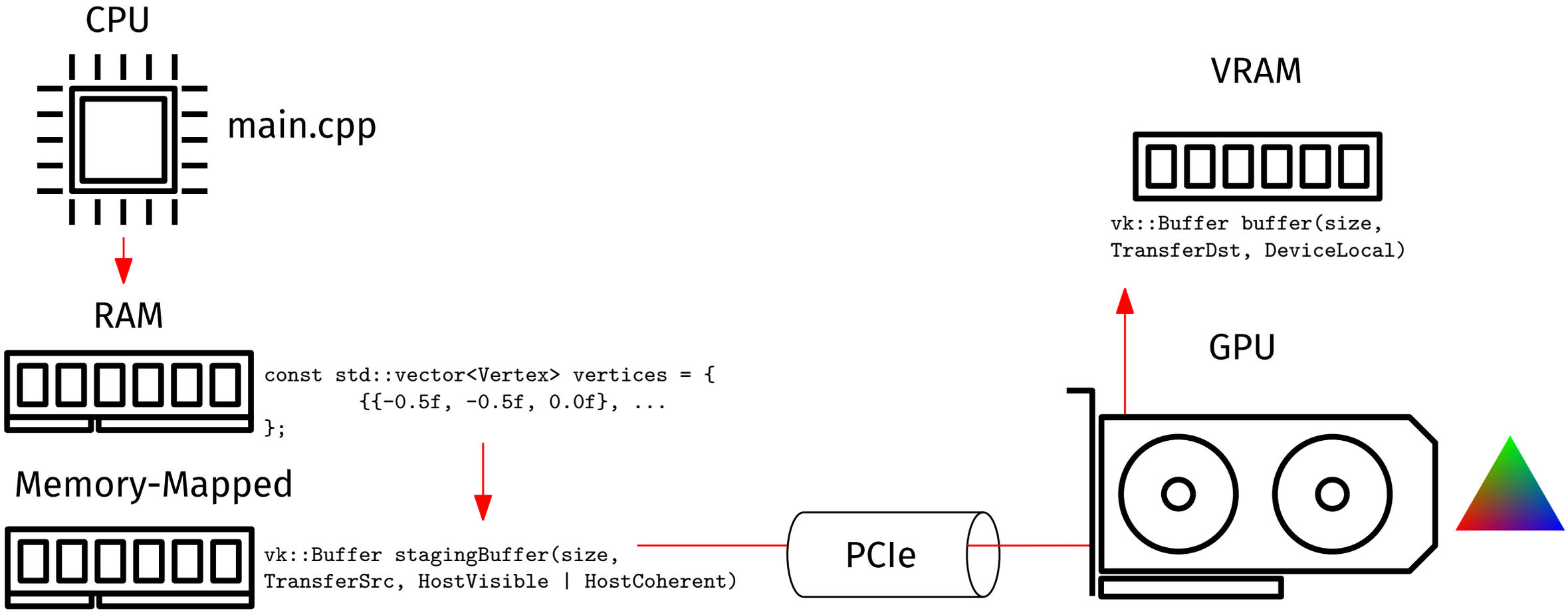




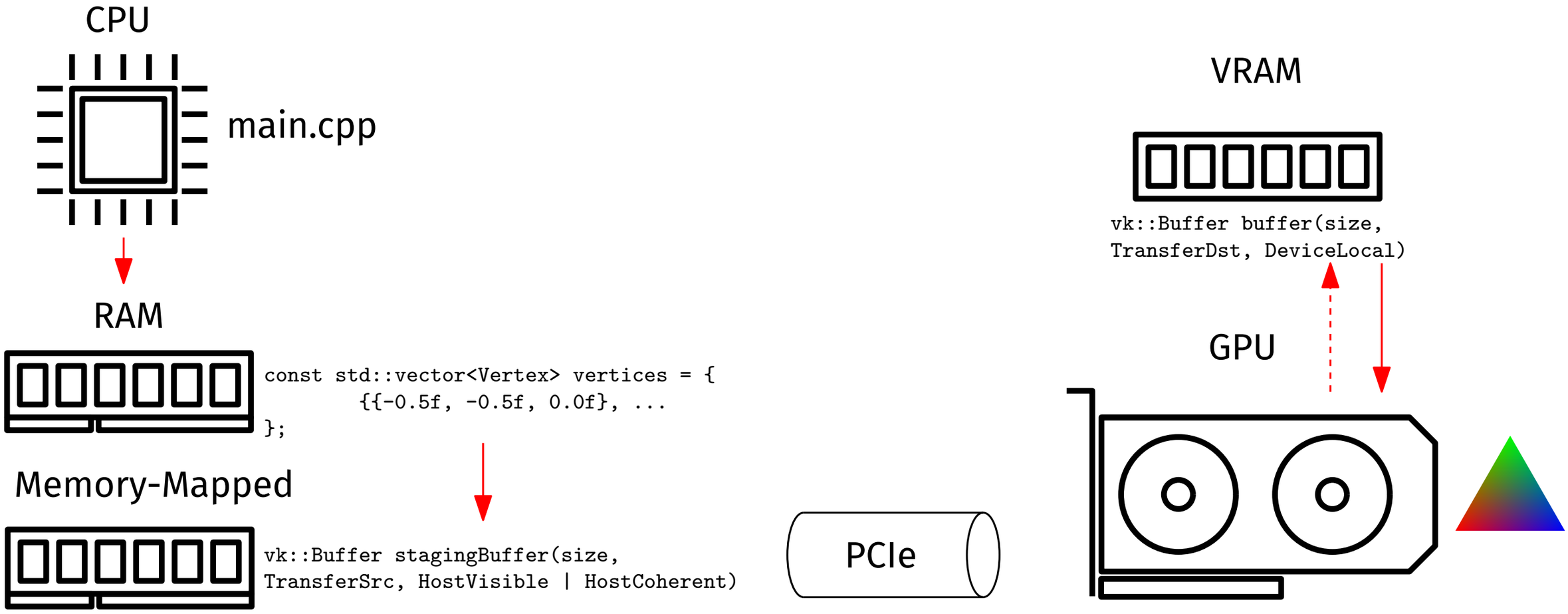
Wie kommen die Daten auf die GPU?



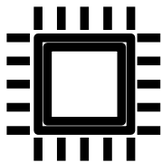
Wie kommen die Daten auf die GPU?

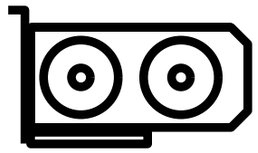


Wie kommen die Daten auf die GPU?



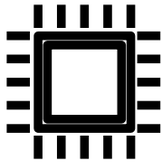
Commandbuffer

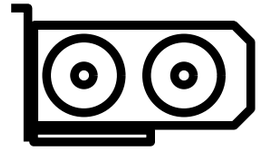
CPU 

GPU 

```
cb.begin(&beginInfo)
```

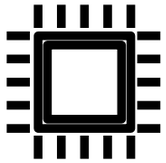
Commandbuffer

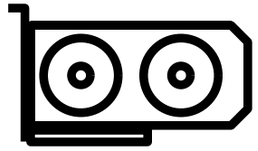
CPU 

GPU 

```
cb.begin(&beginInfo)

cb.bindPipeline(vk::PipelineBindPoint::eGraphics, pipeline);
```

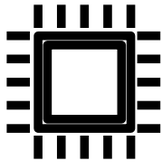
Commandbuffer

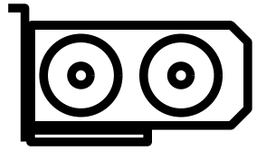
CPU 

GPU 

```
cb.begin(&beginInfo)

cb.bindPipeline(vk::PipelineBindPoint::eGraphics, pipeline);

cb.bindDescriptorSets(vk::PipelineBindPoint::eGraphics,
        layout, 0, 1, graphicsDescriptorSet->GetPtr(), ...
```

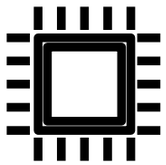
Commandbuffer

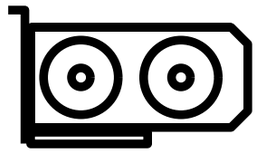
CPU 

GPU 

```
cb.begin(&beginInfo)

cb.bindPipeline(vk::PipelineBindPoint::eGraphics, pipeline);

cb.bindDescriptorSets(vk::PipelineBindPoint::eGraphics,
        layout, 0, 1, graphicsDescriptorSet->GetPtr(), ...

cb.bindVertexBuffers(0, 1, vertexBuffers, offsets);
```

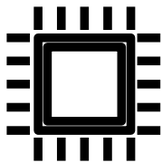
Commandbuffer

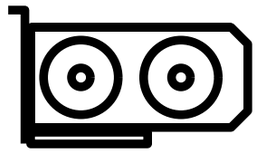
CPU 

GPU 

```
cb.begin(&beginInfo)

cb.bindPipeline(vk::PipelineBindPoint::eGraphics, pipeline);

cb.bindDescriptorSets(vk::PipelineBindPoint::eGraphics,
        layout, 0, 1, graphicsDescriptorSet->GetPtr(), ...

cb.bindVertexBuffers(0, 1, vertexBuffers, offsets);

cb.draw(3, 1, 0, 0);
```

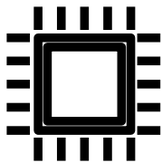
Commandbuffer

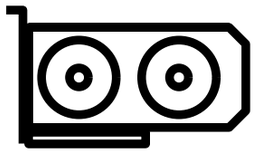
CPU 

GPU 

```
cb.begin(&beginInfo)

cb.bindPipeline(vk::PipelineBindPoint::eGraphics, pipeline);

cb.bindDescriptorSets(vk::PipelineBindPoint::eGraphics,
        layout, 0, 1, graphicsDescriptorSet->GetPtr(), ...

cb.bindVertexBuffers(0, 1, vertexBuffers, offsets);

cb.draw(3, 1, 0, 0);

cb.end();
```

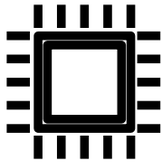
Commandbuffer

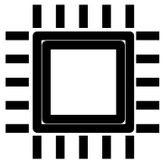
CPU 

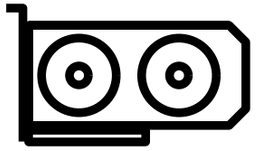
GPU 

```
cb.begin(&beginInfo)

cb.bindPipeline(vk::PipelineBindPoint::eGraphics, pipeline);

cb.bindDescriptorSets(vk::PipelineBindPoint::eGraphics,
        layout, 0, 1, graphicsDescriptorSet->GetPtr(), ...

cb.bindVertexBuffers(0, 1, vertexBuffers, offsets);

cb.draw(3, 1, 0, 0);

cb.end();

vkObject.graphicsQueue.submit(1, &submitInfo,
        this->inFlightFence)
```

Commandbuffer

CPU 

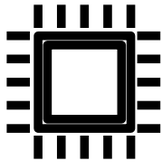


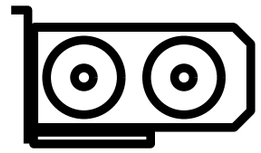
GPU 

```
cb.begin(&beginInfo)

cb.bindPipeline ...

cb.bindDescriptorSets ...    cb.bindDescriptorSets ...


cb.bindVertexBuffers(0, 1,   cb.bindVertexBuffers(0, 1,
        vertexBuffers1, ...          vertexBuffers2, ...
cb.draw(3, 1, 0, 0);         cb.draw(6, 1, 0, 0);

cb.end();

vkObject.graphicsQueue.submit(1, &su
        this->inFlightFence)
```

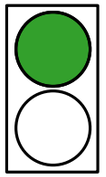
*vereinfacht dargestellt

Synchronization

CPU 

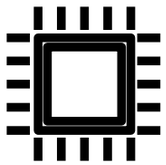
GPU 

S A B

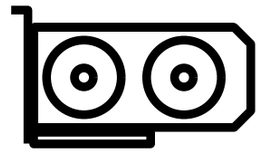


```
vk::CommandBuffer A, B;
vk::Semaphore S

vk::submit(work: A, signal: S, wait: None)

vk::submit(work: B, signal: None, wait: S)
```

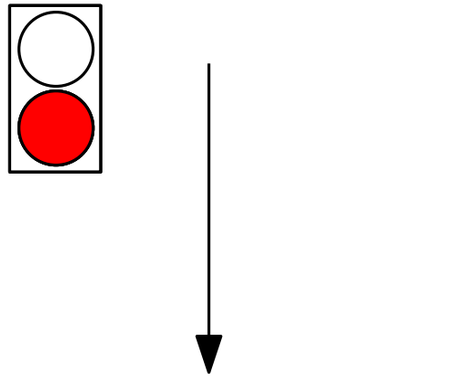
Synchronization

CPU 

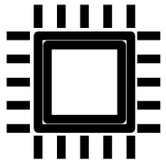
```
vk::CommandBuffer A, B;
vk::Semaphore S

vk::submit(work: A, signal: S, wait: None)

vk::submit(work: B, signal: None, wait: S)
```

GPU 

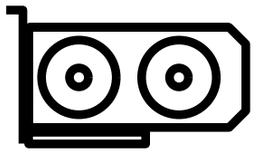
S A B



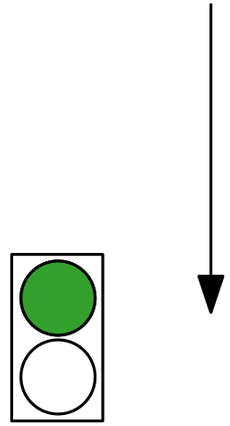
Synchronization

CPU 

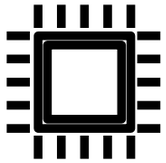
```
vk::CommandBuffer A, B;
vk::Semaphore S

vk::submit(work: A, signal: S, wait: None)

vk::submit(work: B, signal: None, wait: S)
```

GPU 

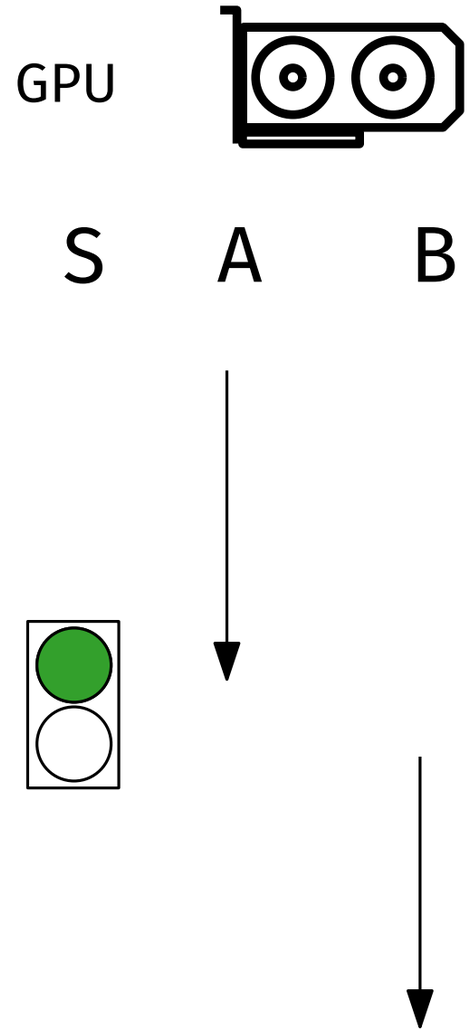
S A B



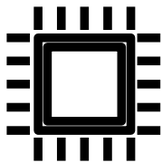
Synchronization

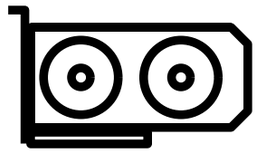
CPU 

```
vk::CommandBuffer A, B;
vk::Semaphore S

vk::submit(work: A, signal: S, wait: None)

vk::submit(work: B, signal: None, wait: S)
```

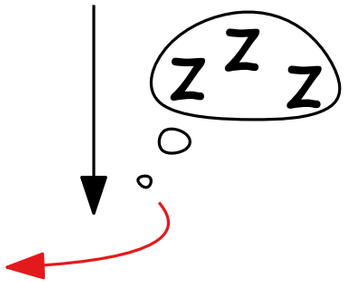


Synchronization 2

CPU 

GPU 

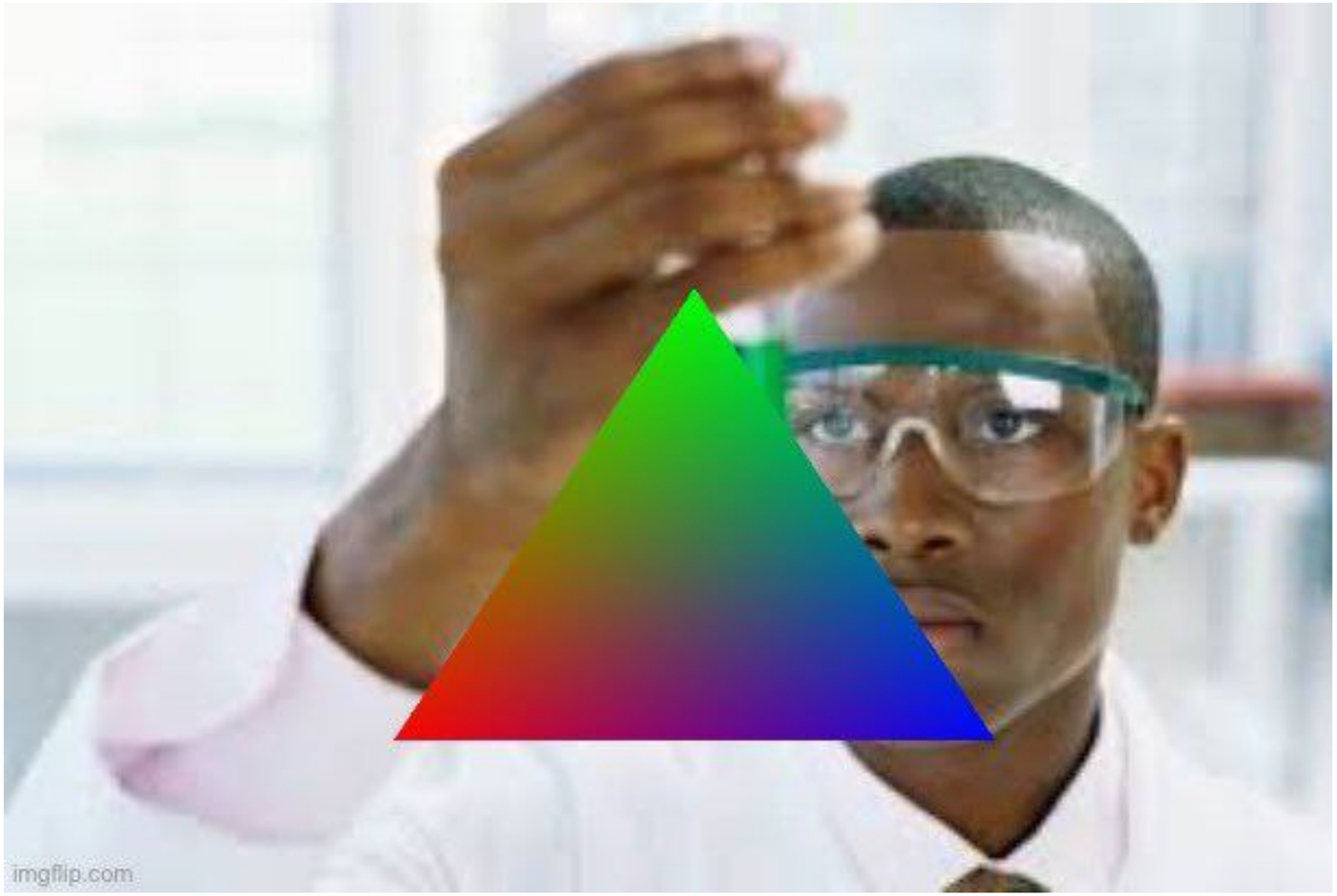
```
vk::CommandBuffer A;
vk::Fence F

vk::submit(work: A, Fence: F)

vk::waitForFence(F)
```



A

```
// save screenshot to disk
```

Finally



Was es sonst noch gibt...

- Textures
- Push-Constants
- Compute-Shader
- ...

Weitere Infos

https://vulkan-tutorial.com/

https://www.vulkan.org/