Introduction to Computer Graphics

186.832, 2021W, 3.0 ECTS

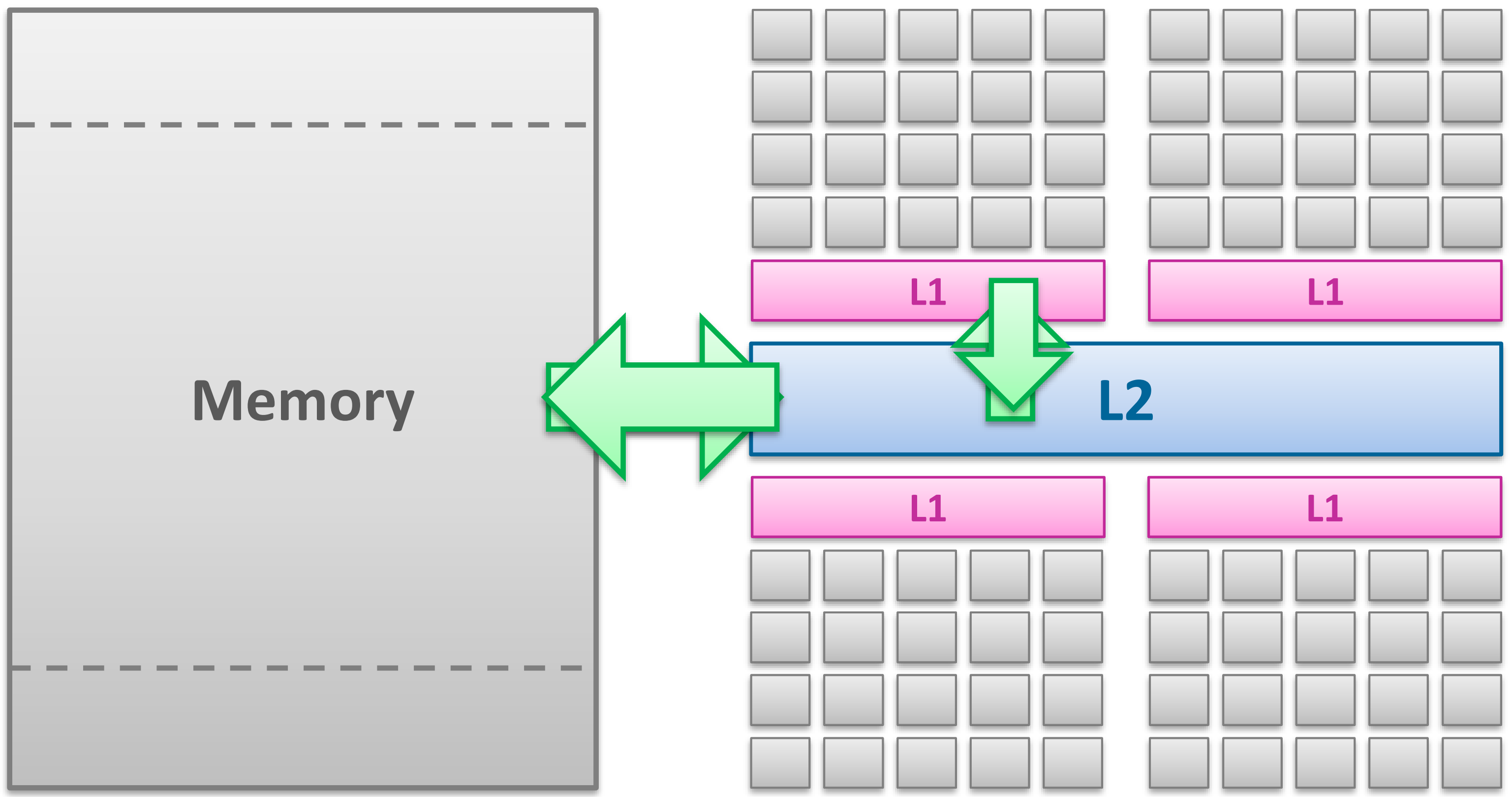
Vulkan Lecture Series, Episode 3:

Resources & Descriptors

Johannes Unterguggenberger

Institute of Visual Computing & Human-Centered Technology
TU Wien, Austria

GPU Activity Depiction



Resources & Descriptors

- ~Four fundamentally different types of **resources**
  - Buffers
  - Images
  - Samplers
  - Acceleration
- **Descriptors** des
  - + usage type
  - + offsets, sometimes
  - + some meta data, sometimes
  - + combinations of resources, sometimes

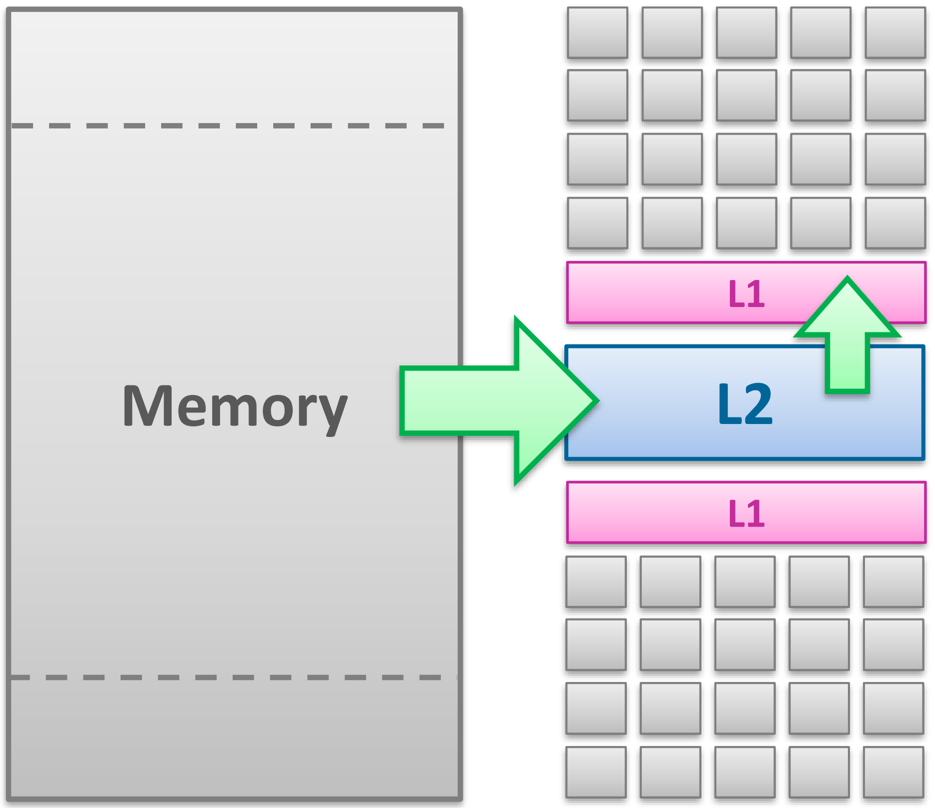
```
Vulkan supports two primary resource types:
buffers and images. Resources are views of memory
with associated formatting and dimensionality.
Buffers are essentially unformatted arrays of
bytes whereas images contain format information,
can be multidimensional and may have associated
metadata.
```

The Khronos Group. Vulkan 1.2.196 Specificaton

Buffers

- Different usage types of **buffers**
  - As **uniform buffer**
  - As storage buffer
  - As texel buffer
    - Uniform texel buffer
    - Storage texel buffer
  - As dynamic buffer
    - Dynamic uniform buffer
    - Dynamic storage buffer
  - (Inline uniform block)

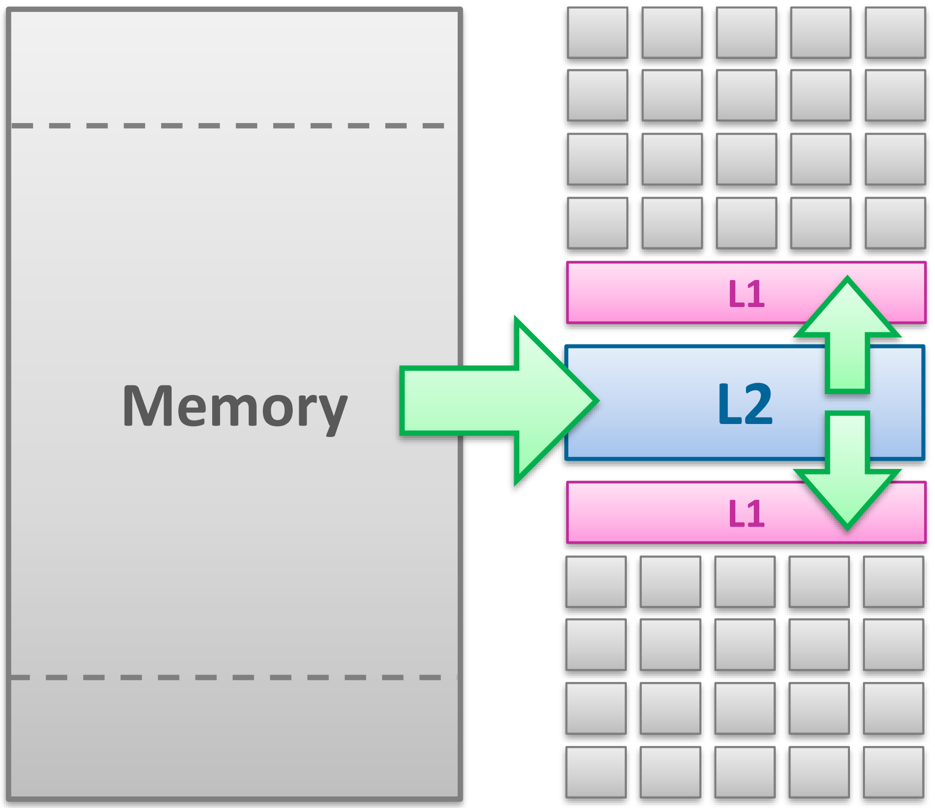
VK_DESCRIPTOR_TYPE_UNIFORM_BUFFER



Buffers

- Different usage types of **buffers**
  - As **uniform buffer**
  - As storage buffer
  - As texel buffer
    - Uniform texel buffer
    - Storage texel buffer
  - As dynamic buffer
    - Dynamic uniform buffer
    - Dynamic storage buffer
  - (Inline uniform block)

VK_DESCRIPTOR_TYPE_UNIFORM_BUFFER

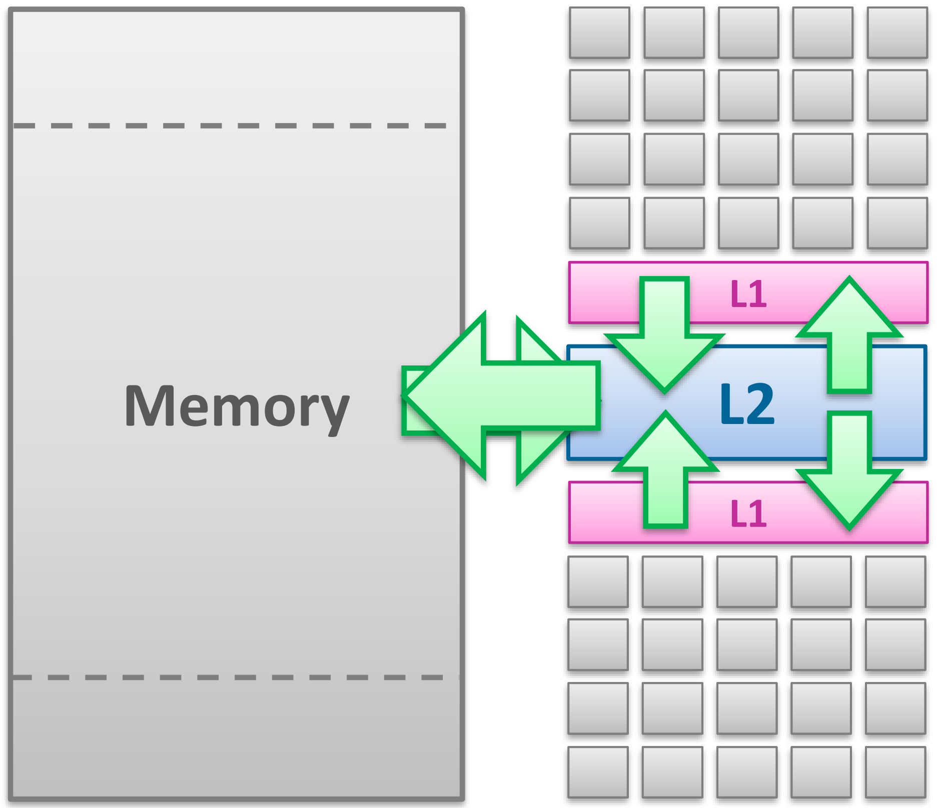


read-only buffers
"read" == "load"

Buffers

- Different usage types of **buffers**
  - As uniform buffer
  - As **storage buffer**
  - As texel buffer
    - Uniform texel buffer
    - Storage texel buffer
  - As dynamic buffer
    - Dynamic uniform buffer
    - Dynamic storage buffer
  - (Inline uniform block)

VK_DESCRIPTOR_TYPE_STORAGE_BUFFER

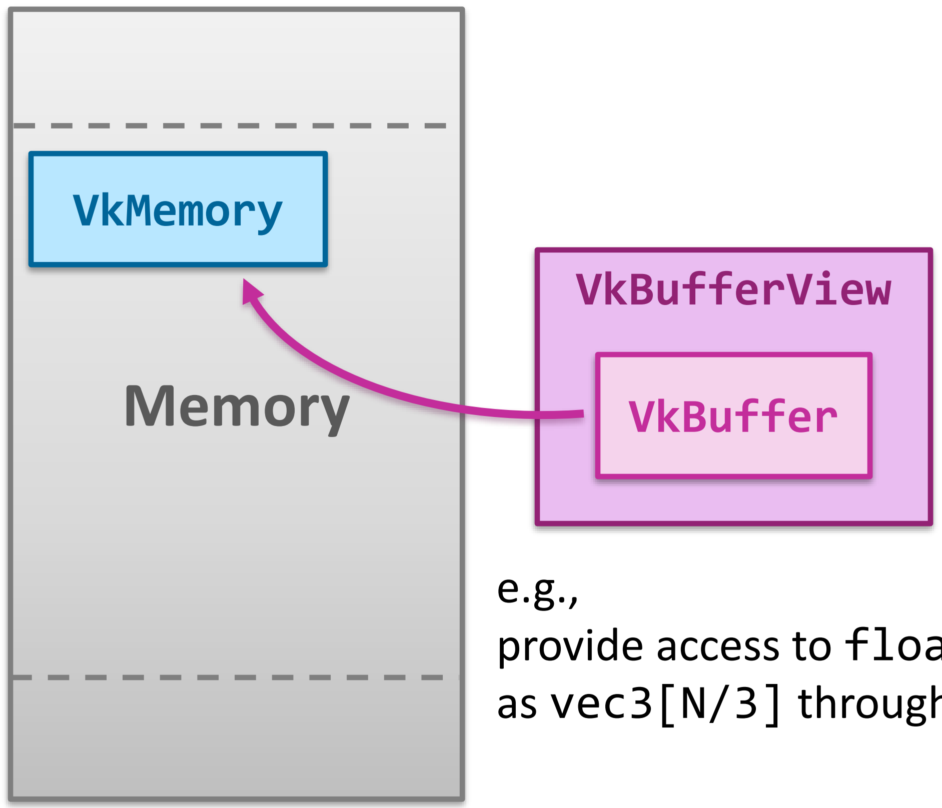


load and store buffers
(also support atomic operations)

Buffers

- Different usage types of **buffers**
  - As uniform buffer
  - As storage buffer
  - As **texel buffer**
    - Uniform texel buffer
    - Storage texel buffer
  - As dynamic buffer
    - Dynamic uniform buffer
    - Dynamic storage buffer
  - (Inline uniform block)

VK_DESCRIPTOR_TYPE_UNIFORM_TEXEL_BUFFER
VK_DESCRIPTOR_TYPE_STORAGE_TEXEL_BUFFER



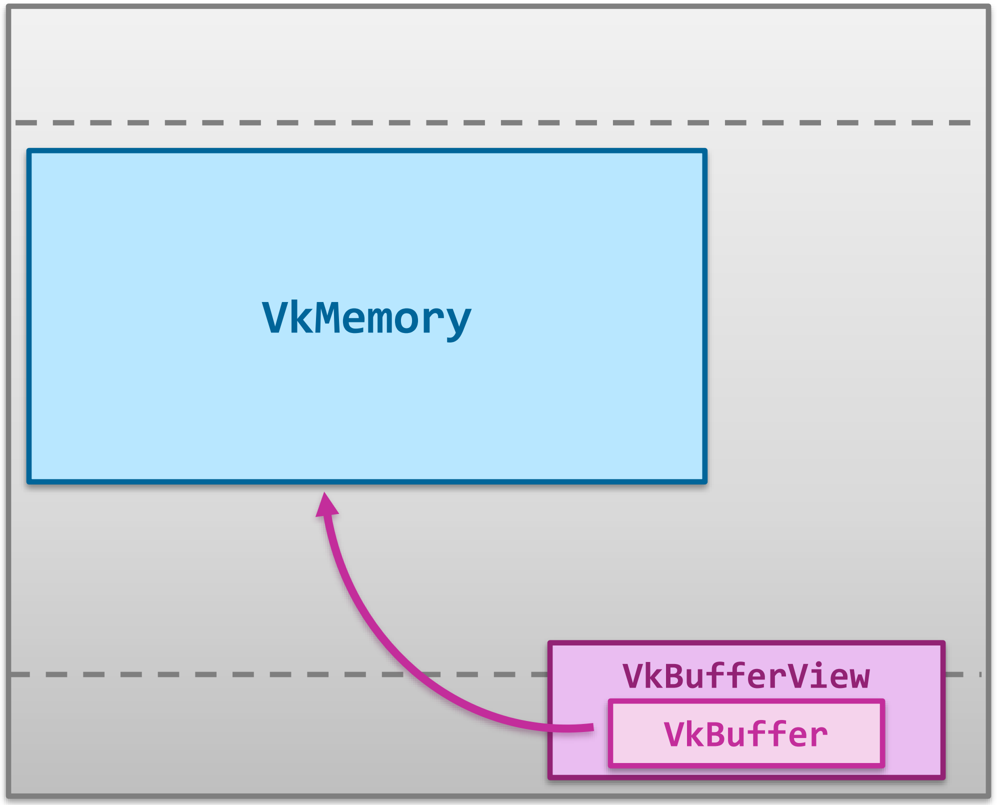
e.g.,
provide access to `float[N]`
as `vec3[N/3]` through view

formatted load (and store, and atomic) operations on buffers

Buffers

- Different usage types of **buffers**
  - As uniform buffer
  - As storage buffer
  - As **texel buffer**
    - Uniform texel buffer
    - Storage texel buffer
  - As dynamic buffer
    - Dynamic uniform buffer
    - Dynamic storage buffer
  - (Inline uniform block)

VK_DESCRIPTOR_TYPE_UNIFORM_TEXEL_BUFFER
VK_DESCRIPTOR_TYPE_STORAGE_TEXEL_BUFFER

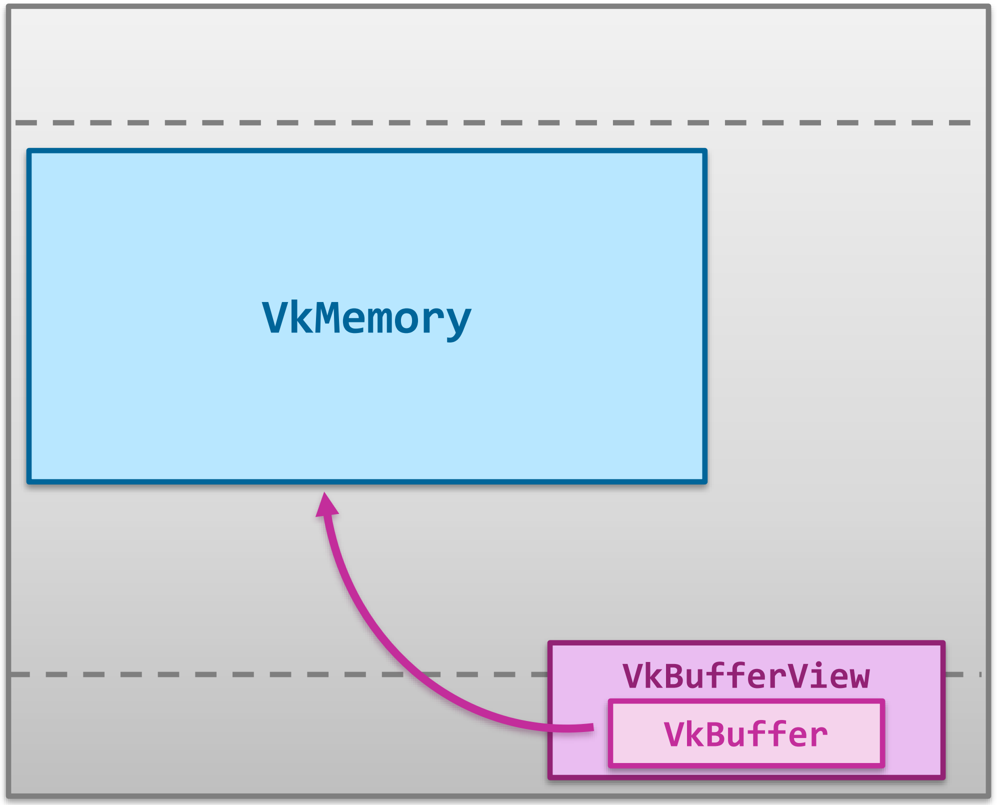


formatted load (and store, and atomic) operations on buffers

Example: Uniform Texel Buffer

```
VkDevice device = // ...

VkBufferCreateInfo bufferInfo = {};
bufferInfo.sType = VK_STRUCTURE_TYPE_BUFFER_CREATE_INFO;
bufferInfo.size = 1024;
bufferInfo.usage = VK_BUFFER_USAGE_UNIFORM_BUFFER_BIT
           | VK_BUFFER_USAGE_UNIFORM_TEXEL_BUFFER_BIT;
bufferInfo.sharingMode = VK_SHARING_MODE_EXCLUSIVE;
VkBuffer buffer;
vkCreateBuffer(device, &bufferInfo, nullptr, buffer);

VkMemoryRequirements req;
vkGetBufferMemoryRequirements(device, buffer, &req);

VkMemoryAllocateInfo memInfo = {};
memInfo.sType = VK_STRUCTURE_TYPE_MEMORY_ALLOCATE_INFO;
memInfo.allocationSize = requirements.size;
memInfo.memoryTypeIndex = // TODO: Find using req
VkDeviceMemory memory;
vkAllocateMemory(device, &memInfo, nullptr, &memory);

vkBindBufferMemory(device, buffer, memory, 0);
```

VK_DESCRIPTOR_TYPE_UNIFORM_TEXEL_BUFFER

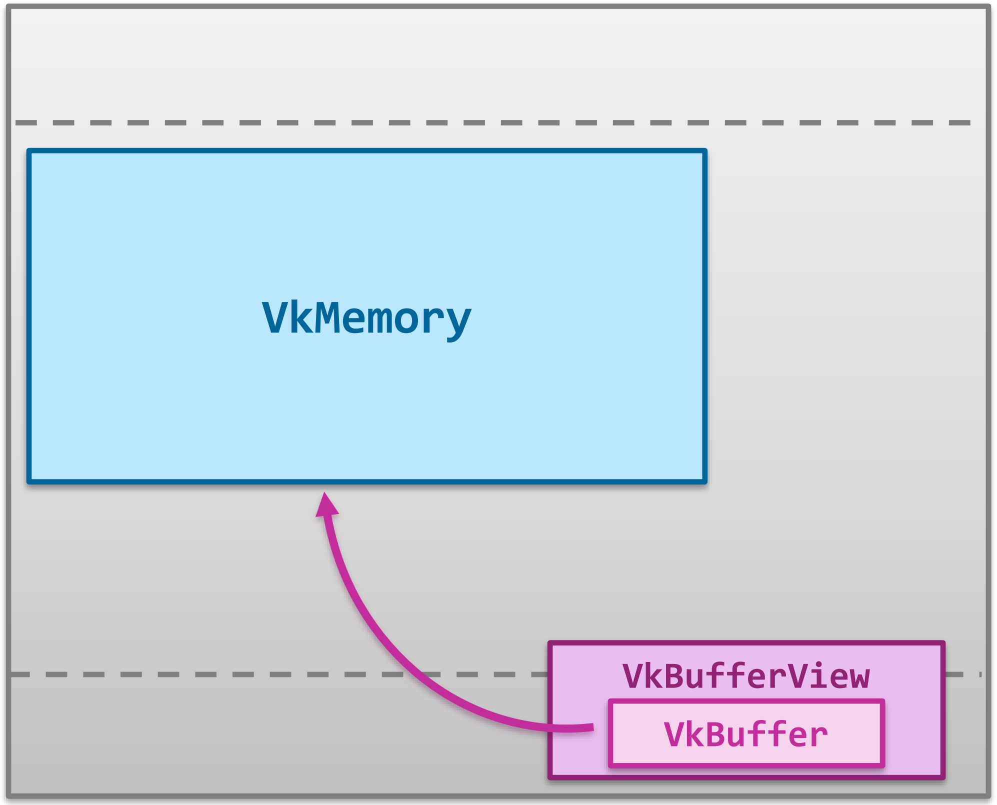


formatted load operations on uniform buffers

Example: Uniform Texel Buffer

```
VkDevice device = // ...

VkBufferCreateInfo bufferInfo = {};
bufferInfo.sType = VK_STRUCTURE_TYPE_BUFFER_CREATE_INFO;
bufferInfo.size = 1024;
bufferInfo.usage = VK_BUFFER_USAGE_UNIFORM_BUFFER_BIT
            | VK_BUFFER_USAGE_UNIFORM_TEXEL_BUFFER_BIT;
bufferInfo.sharingMode = VK_SHARING_MODE_EXCLUSIVE;
VkBuffer buffer;
vkCreateBuffer(device, &bufferInfo, nullptr, buffer);

VkMemoryRequirements req;
vkGetBufferMemoryRequirements(device, buffer, &req);

VkMemoryAllocateInfo memInfo = {};
memInfo.sType = VK_STRUCTURE_TYPE_MEMORY_ALLOCATE_INFO;
memInfo.allocationSize = requirements.size;
memInfo.memoryTypeIndex = // TODO: Find using req
VkDeviceMemory memory;
vkAllocateMemory(device, &memInfo, nullptr, &memory);

vkBindBufferMemory(device, buffer, memory, 0);
```

VK_DESCRIPTOR_TYPE_UNIFORM_TEXEL_BUFFER

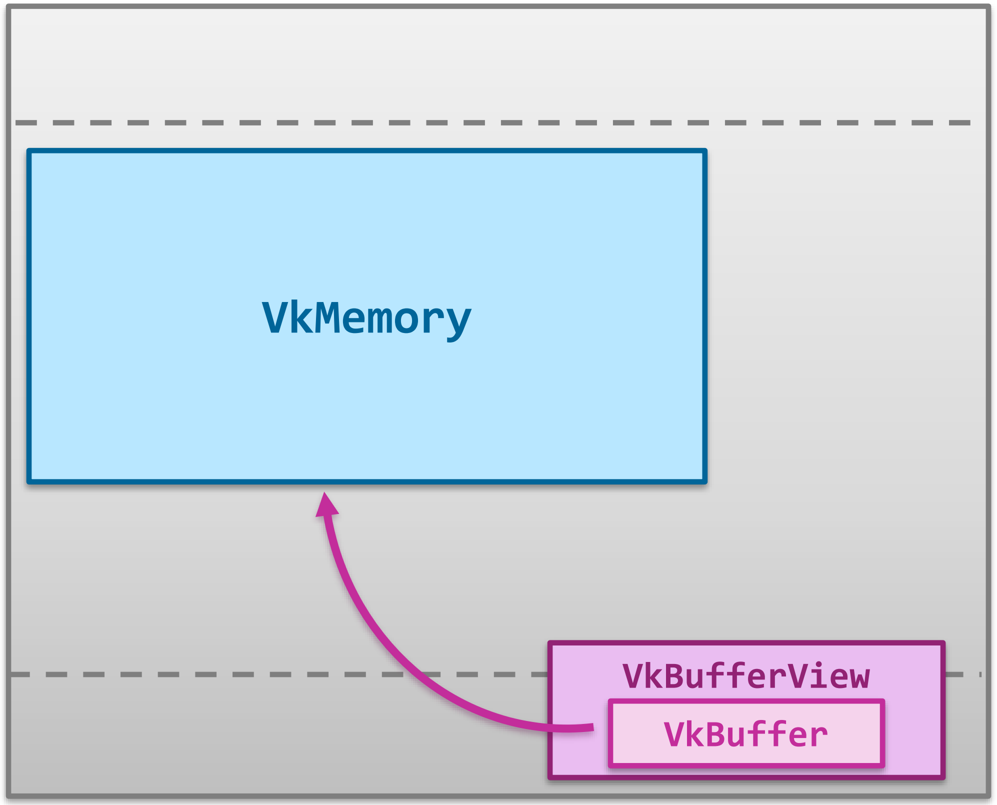


formatted load operations on uniform buffers

Example: Uniform Texel Buffer

```
VkDevice device = // ...

VkBufferCreateInfo bufferInfo = {};
bufferInfo.sType = VK_STRUCTURE_TYPE_BUFFER_CREATE_INFO;
bufferInfo.size = 1024;
bufferInfo.usage = VK_BUFFER_USAGE_UNIFORM_BUFFER_BIT
           | VK_BUFFER_USAGE_UNIFORM_TEXEL_BUFFER_BIT;
bufferInfo.sharingMode = VK_SHARING_MODE_EXCLUSIVE;
VkBuffer buffer;
vkCreateBuffer(device, &bufferInfo, nullptr, buffer);

VkMemoryRequirements req;
vkGetBufferMemoryRequirements(device, buffer, &req);

VkMemoryAllocateInfo memInfo = {};
memInfo.sType = VK_STRUCTURE_TYPE_MEMORY_ALLOCATE_INFO;
memInfo.allocationSize = requirements.size;
memInfo.memoryTypeIndex = // TODO: Find using req
VkDeviceMemory memory;
vkAllocateMemory(device, &memInfo, nullptr, &memory);

vkBindBufferMemory(device, buffer, memory, 0);
```

VK_DESCRIPTOR_TYPE_UNIFORM_TEXEL_BUFFER

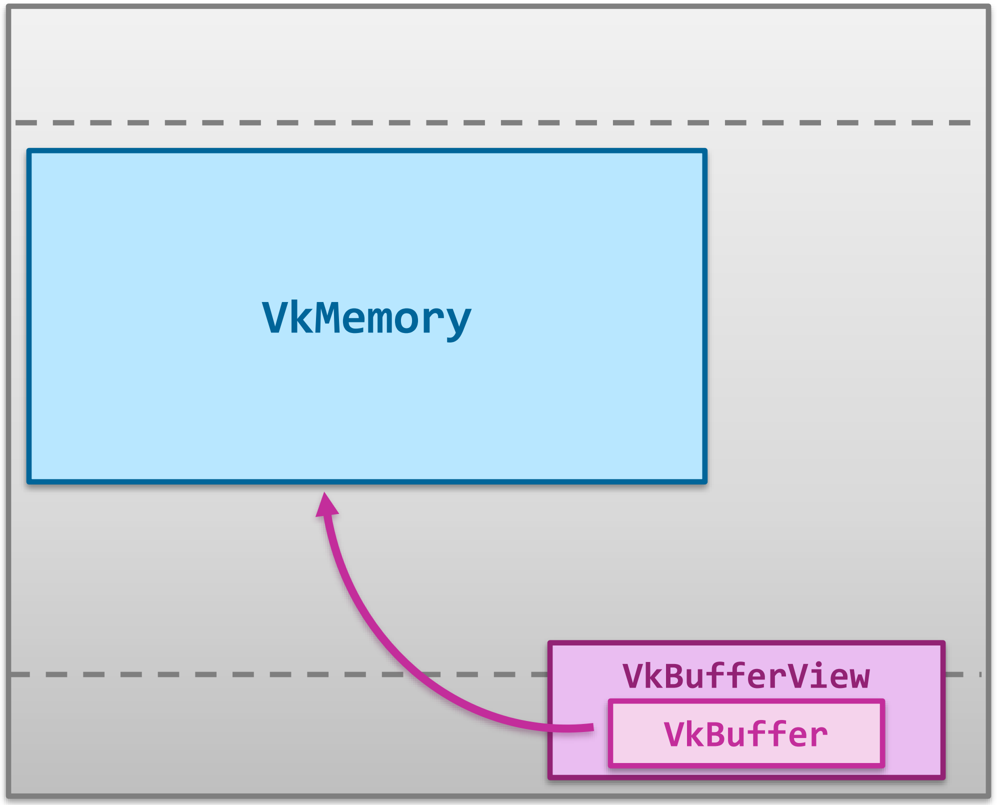


formatted load operations on uniform buffers

Example: Uniform Texel Buffer

```
VkDevice device = // ...

VkBufferCreateInfo bufferInfo = {};
bufferInfo.sType = VK_STRUCTURE_TYPE_BUFFER_CREATE_INFO;
bufferInfo.size = 1024;
bufferInfo.usage = VK_BUFFER_USAGE_UNIFORM_BUFFER_BIT
            | VK_BUFFER_USAGE_UNIFORM_TEXEL_BUFFER_BIT;
bufferInfo.sharingMode = VK_SHARING_MODE_EXCLUSIVE;
VkBuffer buffer;
vkCreateBuffer(device, &bufferInfo, nullptr, buffer);

VkMemoryRequirements req;
vkGetBufferMemoryRequirements(device, buffer, &req);

VkMemoryAllocateInfo memInfo = {};
memInfo.sType = VK_STRUCTURE_TYPE_MEMORY_ALLOCATE_INFO;
memInfo.allocationSize = requirements.size;
memInfo.memoryTypeIndex = // TODO: Find using req
VkDeviceMemory memory;
vkAllocateMemory(device, &memInfo, nullptr, &memory);

vkBindBufferMemory(device, buffer, memory, 0);
```

VK_DESCRIPTOR_TYPE_UNIFORM_TEXEL_BUFFER

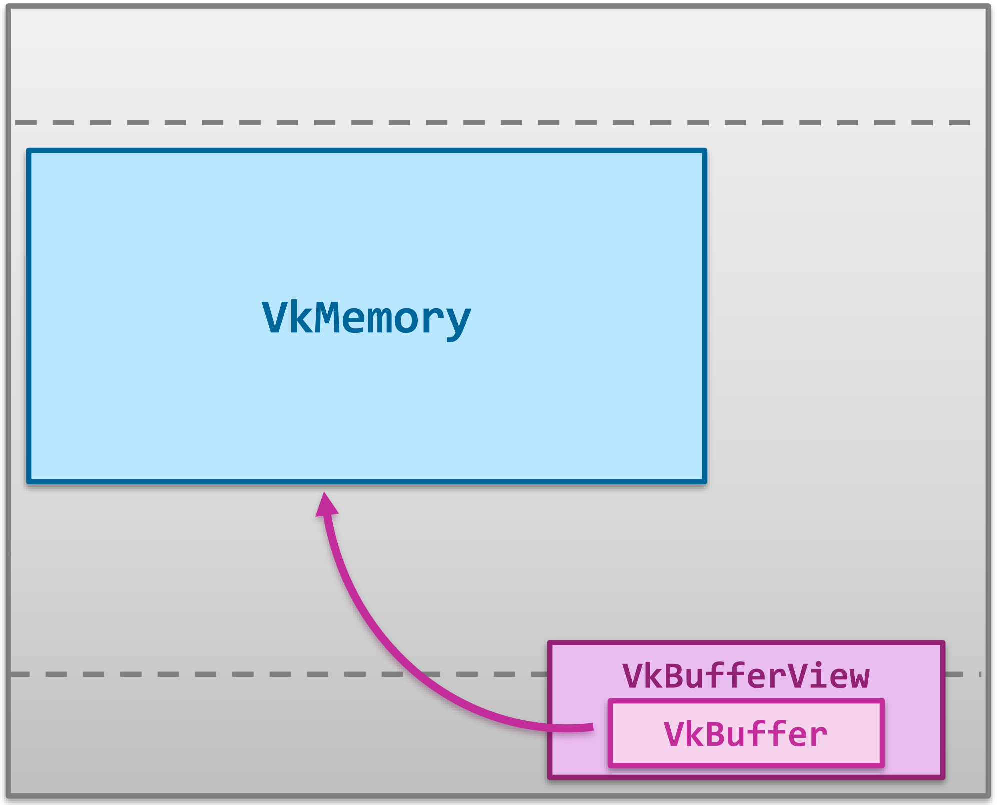


formatted load operations on uniform buffers

Example: Uniform Texel Buffer

```
VkDevice device = // ...

VkBufferCreateInfo bufferInfo = {};
bufferInfo.sType = VK_STRUCTURE_TYPE_BUFFER_CREATE_INFO;
bufferInfo.size = 1024;
bufferInfo.usage = VK_BUFFER_USAGE_UNIFORM_BUFFER_BIT
             | VK_BUFFER_USAGE_UNIFORM_TEXEL_BUFFER_BIT;
bufferInfo.sharingMode = VK_SHARING_MODE_EXCLUSIVE;
VkBuffer buffer;
vkCreateBuffer(device, &bufferInfo, nullptr, buffer);

VkMemoryRequirements req;
vkGetBufferMemoryRequirements(device, buffer, &req);

VkMemoryAllocateInfo memInfo = {};
memInfo.sType = VK_STRUCTURE_TYPE_MEMORY_ALLOCATE_INFO;
memInfo.allocationSize = requirements.size;
memInfo.memoryTypeIndex = // TODO: Find using req
VkDeviceMemory memory;
vkAllocateMemory(device, &memInfo, nullptr, &memory);

vkBindBufferMemory(device, buffer, memory, 0);
```

VK_DESCRIPTOR_TYPE_UNIFORM_TEXEL_BUFFER



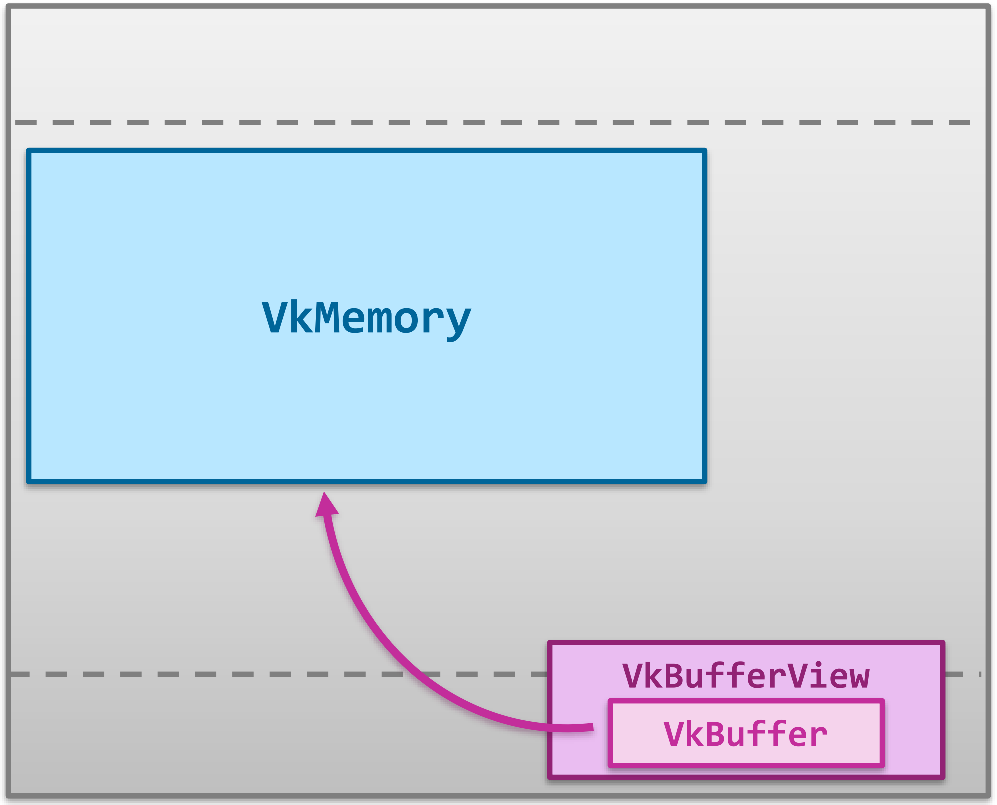
formatted load operations on uniform buffers

Example: Uniform Texel Buffer

```
VkDevice device = // ...
```

VK_DESCRIPTOR_TYPE_UNIFORM_TEXEL_BUFFER

```
VkBufferViewCreateInfo viewInfo = {};
viewInfo.sType = VK_STRUCTURE_TYPE_BUFFER_VIEW_CREATE_INFO;
viewInfo.buffer = buffer;
viewInfo.format = VK_FORMAT_R32G32B32_SFLOAT;
viewInfo.offset = 0;
viewInfo.range = VK_WHOLE_SIZE;

VkBufferView bufferView;
vkCreateBufferView(device, &viewInfo, nullptr, &bufferView);
```

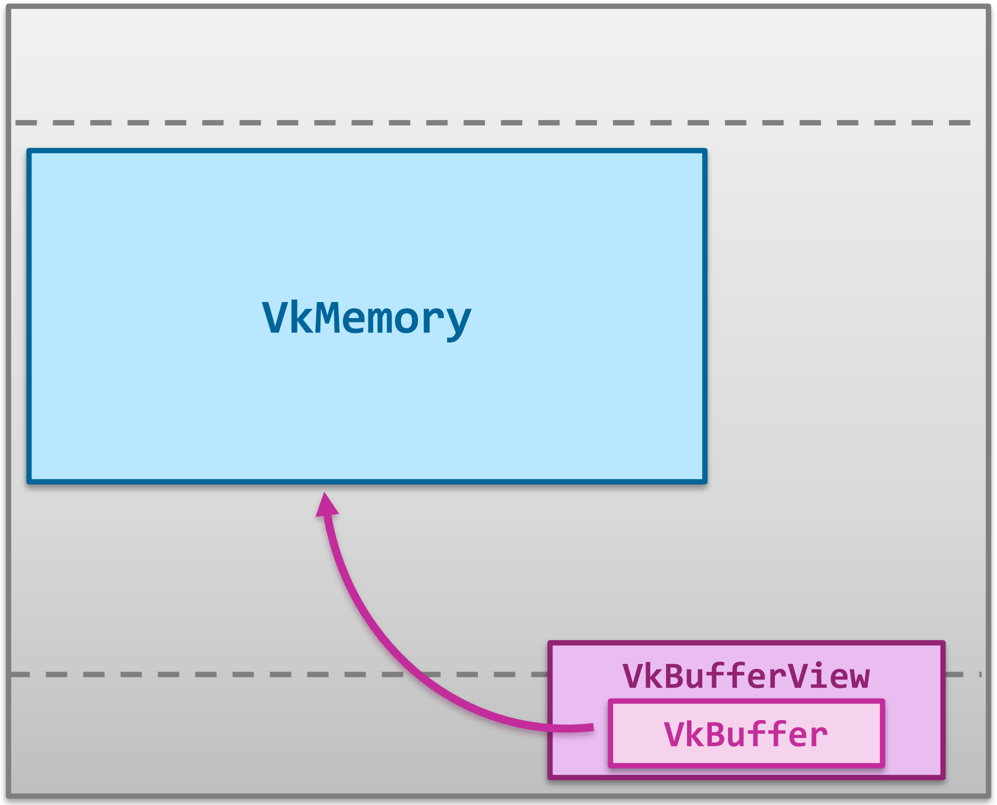


formatted load operations on uniform buffers

Example: Uniform ~~Texel~~ Buffer

```
VkDevice device = // ...

VkBufferCreateInfo bufferInfo = {};
bufferInfo.sType = VK_STRUCTURE_TYPE_BUFFER_CREATE_INFO;
bufferInfo.size = 1024;
bufferInfo.usage = VK_BUFFER_USAGE_UNIFORM_BUFFER_BIT
             | VK_BUFFER_USAGE_UNIFORM_TEXEL_BUFFER_BIT;
bufferInfo.sharingMode = VK_SHARING_MODE_EXCLUSIVE;
VkBuffer buffer;
vkCreateBuffer(device, &bufferInfo, nullptr, buffer);

VkMemoryRequirements req;
vkGetBufferMemoryRequirements(device, buffer, &req);

VkMemoryAllocateInfo memInfo = {};
memInfo.sType = VK_STRUCTURE_TYPE_MEMORY_ALLOCATE_INFO;
memInfo.allocationSize = requirements.size;
memInfo.memoryTypeIndex = // TODO: Find using req
VkDeviceMemory memory;
vkAllocateMemory(device, &memInfo, nullptr, &memory);

vkBindBufferMemory(device, buffer, memory, 0);
```

VK_DESCRIPTOR_TYPE_UNIFORM_TEXEL_BUFFER

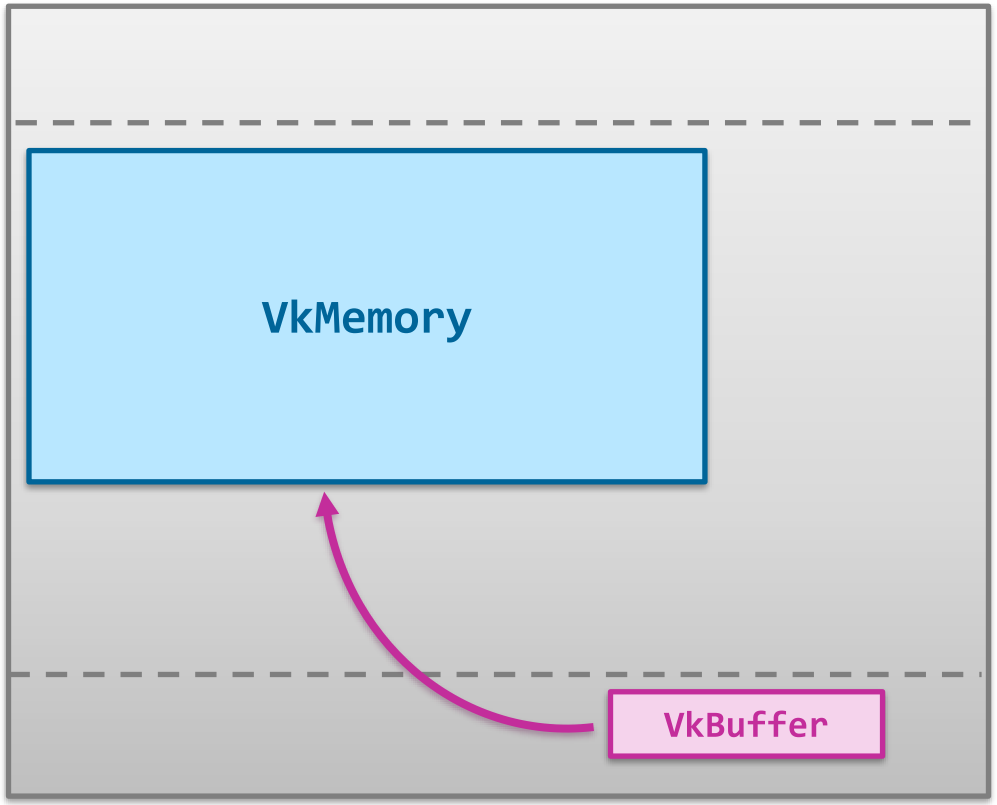


formatted load operations on uniform buffers

Example: Uniform ~~Texel~~ Buffer

```
VkDevice device = // ...

VkBufferCreateInfo bufferInfo = {};
bufferInfo.sType = VK_STRUCTURE_TYPE_BUFFER_CREATE_INFO;
bufferInfo.size = 1024;
bufferInfo.usage = VK_BUFFER_USAGE_UNIFORM_BUFFER_BIT
           | VK_BUFFER_USAGE_UNIFORM_TEXEL_BUFFER_BIT;
bufferInfo.sharingMode = VK_SHARING_MODE_EXCLUSIVE;
VkBuffer buffer;
vkCreateBuffer(device, &bufferInfo, nullptr, buffer);

VkMemoryRequirements req;
vkGetBufferMemoryRequirements(device, buffer, &req);

VkMemoryAllocateInfo memInfo = {};
memInfo.sType = VK_STRUCTURE_TYPE_MEMORY_ALLOCATE_INFO;
memInfo.allocationSize = requirements.size;
memInfo.memoryTypeIndex = // TODO: Find using req
VkDeviceMemory memory;
vkAllocateMemory(device, &memInfo, nullptr, &memory);

vkBindBufferMemory(device, buffer, memory, 0);
```

VK_DESCRIPTOR_TYPE_UNIFORM_TEXEL_BUFFER

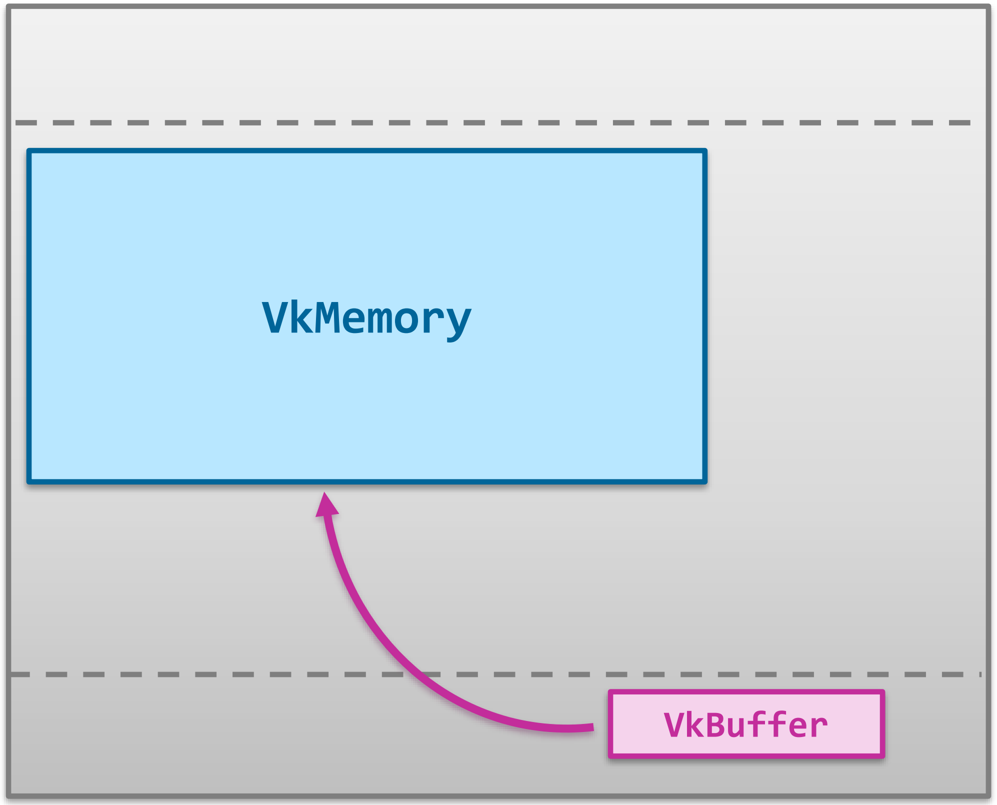


load operations on uniform buffers

Example: Uniform ~~Texel~~ Buffer

```
VkDevice device = // ...

VkBufferCreateInfo bufferInfo = {};
bufferInfo.sType = VK_STRUCTURE_TYPE_BUFFER_CREATE_INFO;
bufferInfo.size = 1024;
bufferInfo.usage = VK_BUFFER_USAGE_UNIFORM_BUFFER_BIT
          | VK_BUFFER_USAGE_UNIFORM_TEXEL_BUFFER_BIT;
bufferInfo.sharingMode = VK_SHARING_MODE_EXCLUSIVE;
VkBuffer buffer;
vkCreateBuffer(device, &bufferInfo, nullptr, buffer);

VkMemoryRequirements req;
vkGetBufferMemoryRequirements(device, buffer, &req);

VkMemoryAllocateInfo memInfo = {};
memInfo.sType = VK_STRUCTURE_TYPE_MEMORY_ALLOCATE_INFO;
memInfo.allocationSize = requirements.size;
memInfo.memoryTypeIndex = // TODO: Find using req
VkDeviceMemory memory;
vkAllocateMemory(device, &memInfo, nullptr, &memory);

vkBindBufferMemory(device, buffer, memory, 0);
```

VK_DESCRIPTOR_TYPE_UNIFORM_BUFFER

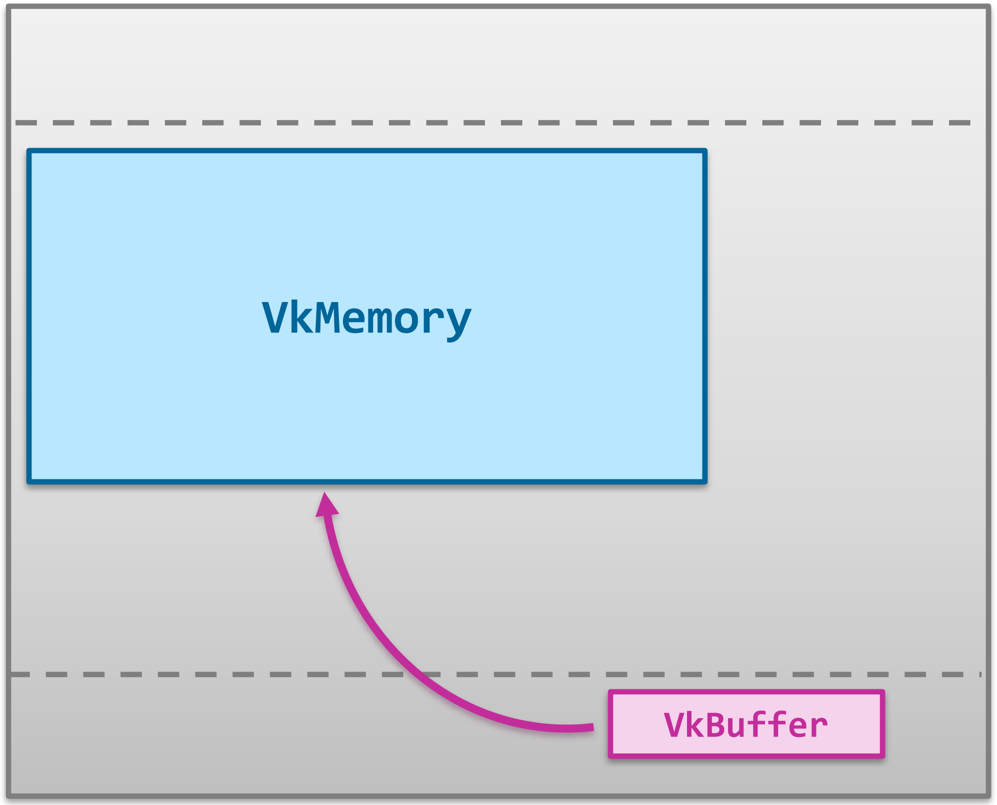


load operations on uniform buffers

Example: Uniform ~~Texel~~ Buffer

```
VkDevice device = // ...

VkBufferCreateInfo bufferInfo = {};
bufferInfo.sType = VK_STRUCTURE_TYPE_BUFFER_CREATE_INFO;
bufferInfo.size = 1024;
bufferInfo.usage = VK_BUFFER_USAGE_UNIFORM_BUFFER_BIT;

bufferInfo.sharingMode = VK_SHARING_MODE_EXCLUSIVE;
VkBuffer buffer;
vkCreateBuffer(device, &bufferInfo, nullptr, buffer);

VkMemoryRequirements req;
vkGetBufferMemoryRequirements(device, buffer, &req);

VkMemoryAllocateInfo memInfo = {};
memInfo.sType = VK_STRUCTURE_TYPE_MEMORY_ALLOCATE_INFO;
memInfo.allocationSize = requirements.size;
memInfo.memoryTypeIndex = // TODO: Find using req
VkDeviceMemory memory;
vkAllocateMemory(device, &memInfo, nullptr, &memory);

vkBindBufferMemory(device, buffer, memory, 0);
```

VK_DESCRIPTOR_TYPE_UNIFORM_BUFFER

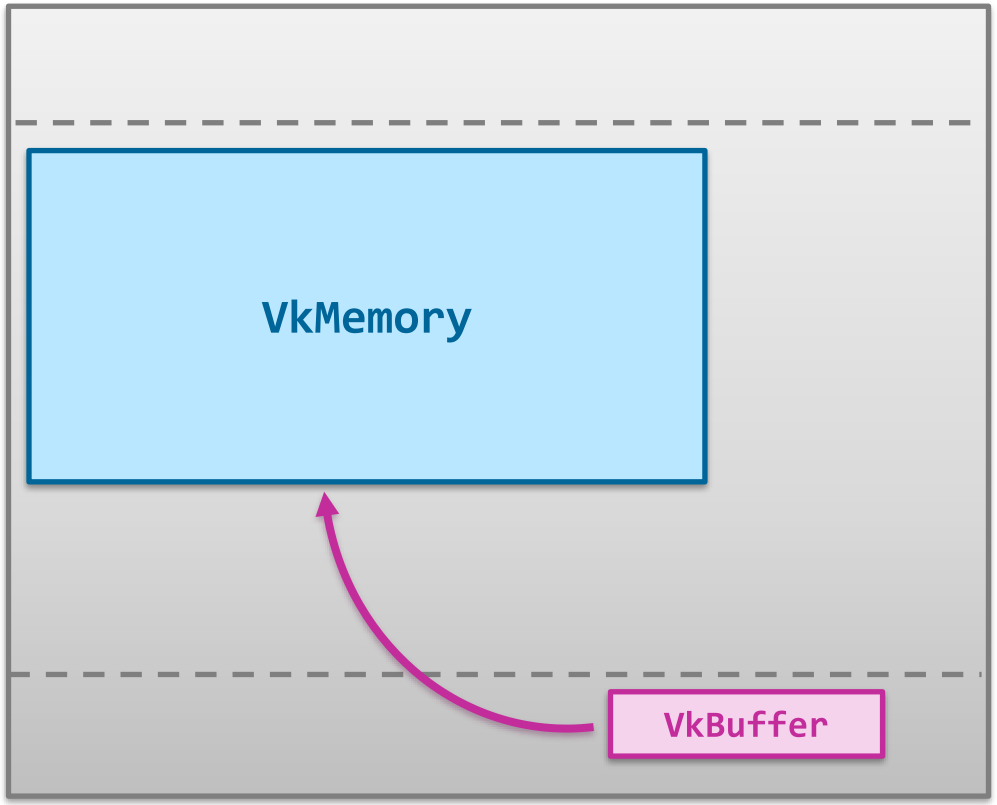


load operations on uniform buffers

Buffers

- Different usage types of **buffers**
  - As uniform buffer
  - As storage buffer
  - As texel buffer
    - Uniform texel buffer
    - Storage texel buffer
  - As **dynamic buffer**
    - Dynamic uniform buffer
    - Dynamic storage buffer
  - (Inline uniform block)

VK_DESCRIPTOR_TYPE_UNIFORM_BUFFER_DYNAMIC
VK_DESCRIPTOR_TYPE_STORAGE_BUFFER_DYNAMIC

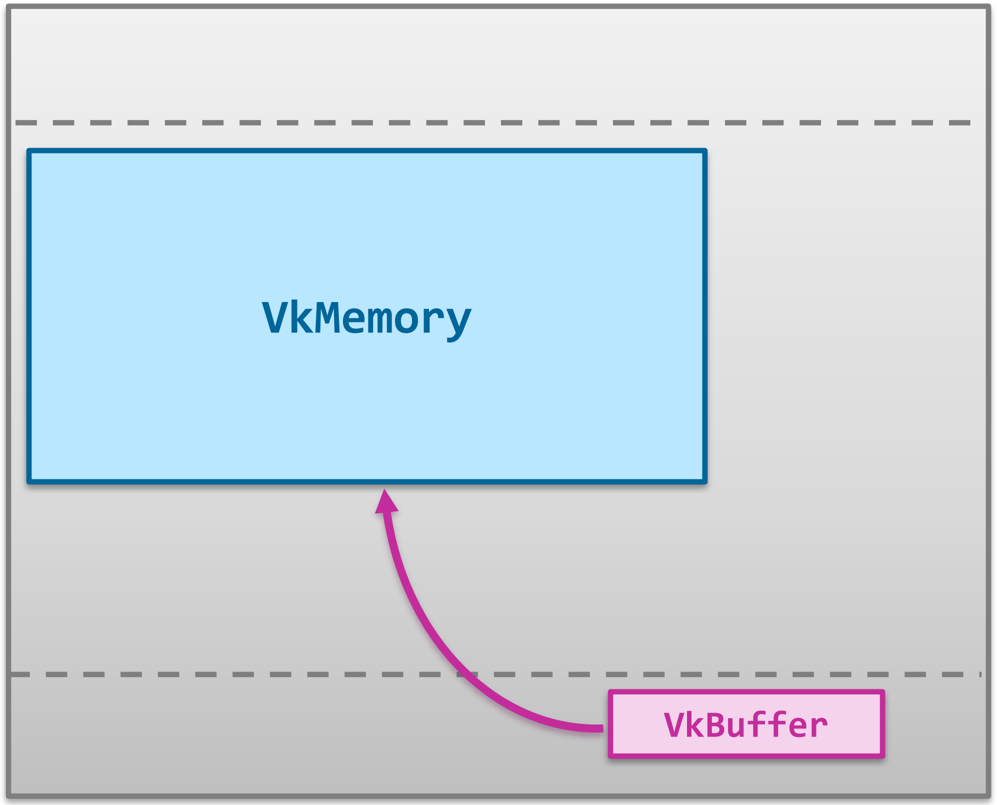


+ additional offset into VkMemory, changeable at run-time with little overhead

Buffers

- Different usage types of **buffers**
  - As uniform buffer
  - As storage buffer
  - As texel buffer
    - Uniform texel buffer
    - Storage texel buffer
  - As **dynamic buffer**
    - Dynamic uniform buffer
    - Dynamic storage buffer
  - (Inline uniform block)

VK_DESCRIPTOR_TYPE_UNIFORM_BUFFER_DYNAMIC
VK_DESCRIPTOR_TYPE_STORAGE_BUFFER_DYNAMIC

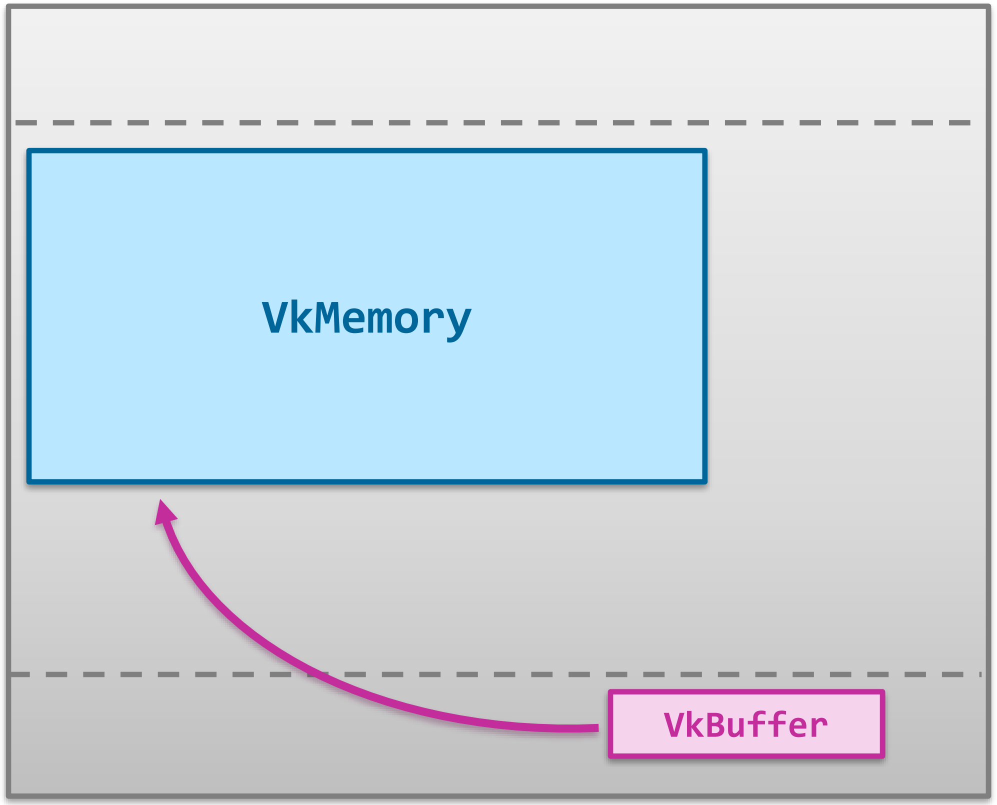


+ additional offset into VkMemory, changeable at run-time with little overhead

Buffers

- Different usage types of **buffers**
  - As uniform buffer
  - As storage buffer
  - As texel buffer
    - Uniform texel buffer
    - Storage texel buffer
  - As **dynamic buffer**
    - Dynamic uniform buffer
    - Dynamic storage buffer
  - (Inline uniform block)

VK_DESCRIPTOR_TYPE_UNIFORM_BUFFER_DYNAMIC
VK_DESCRIPTOR_TYPE_STORAGE_BUFFER_DYNAMIC

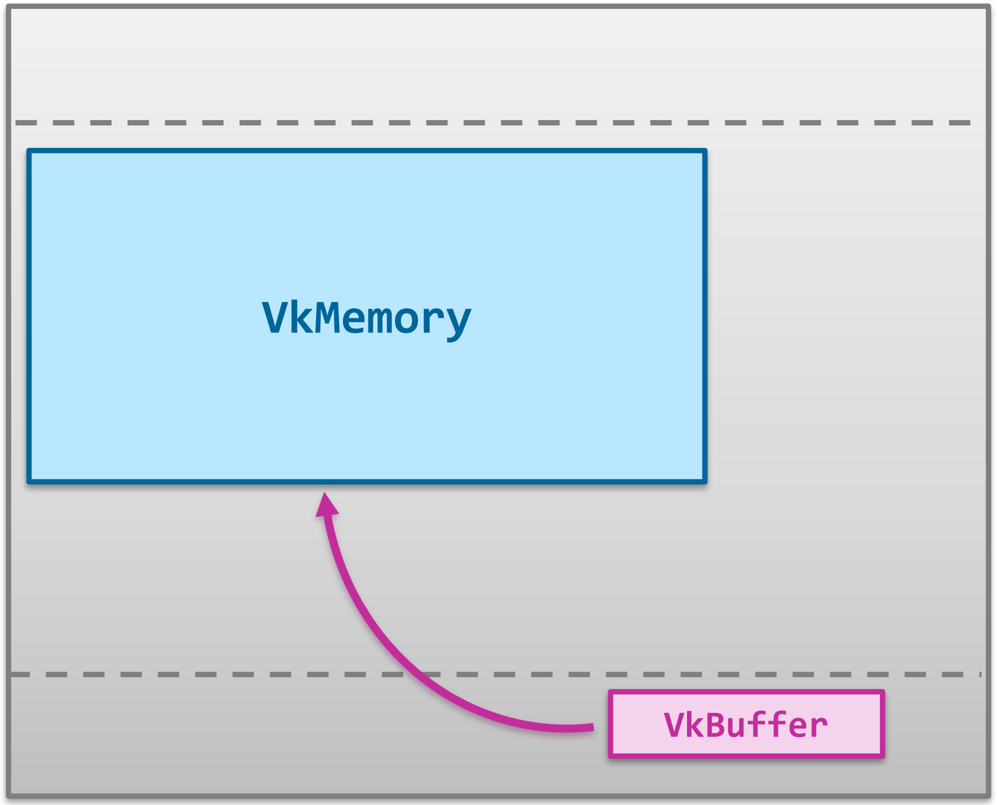


+ additional offset into `VkMemory`, changeable at run-time with little overhead

Buffers

- Different usage types of **buffers**
  - As uniform buffer
  - As storage buffer
  - As texel buffer
    - Uniform texel buffer
    - Storage texel buffer
  - As **dynamic buffer**
    - Dynamic uniform buffer
    - Dynamic storage buffer
  - (Inline uniform block)

VK_DESCRIPTOR_TYPE_UNIFORM_BUFFER_DYNAMIC
VK_DESCRIPTOR_TYPE_STORAGE_BUFFER_DYNAMIC

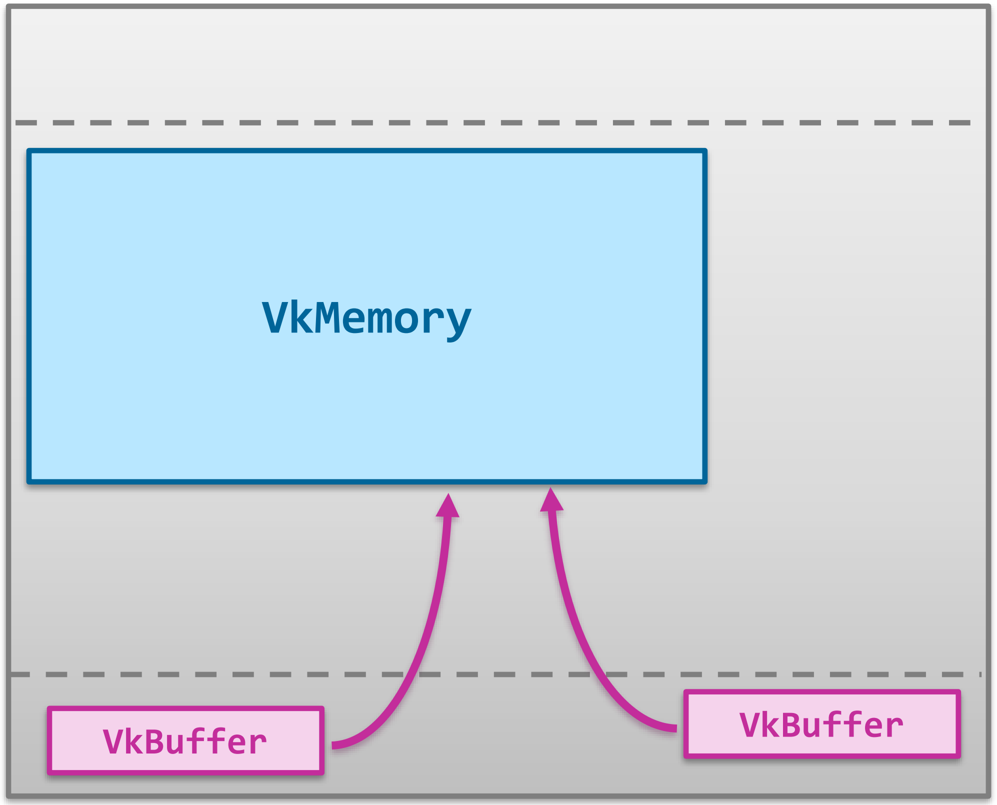


+ additional offset into VkMemory, changeable at run-time with little overhead

Buffers

- Different usage types of **buffers**
  - As uniform buffer
  - As storage buffer
  - As texel buffer
    - Uniform texel buffer
    - Storage texel buffer
  - As **dynamic buffer**
    - Dynamic uniform buffer
    - Dynamic storage buffer
  - (Inline uniform block)

VK_DESCRIPTOR_TYPE_UNIFORM_BUFFER_DYNAMIC
VK_DESCRIPTOR_TYPE_STORAGE_BUFFER_DYNAMIC

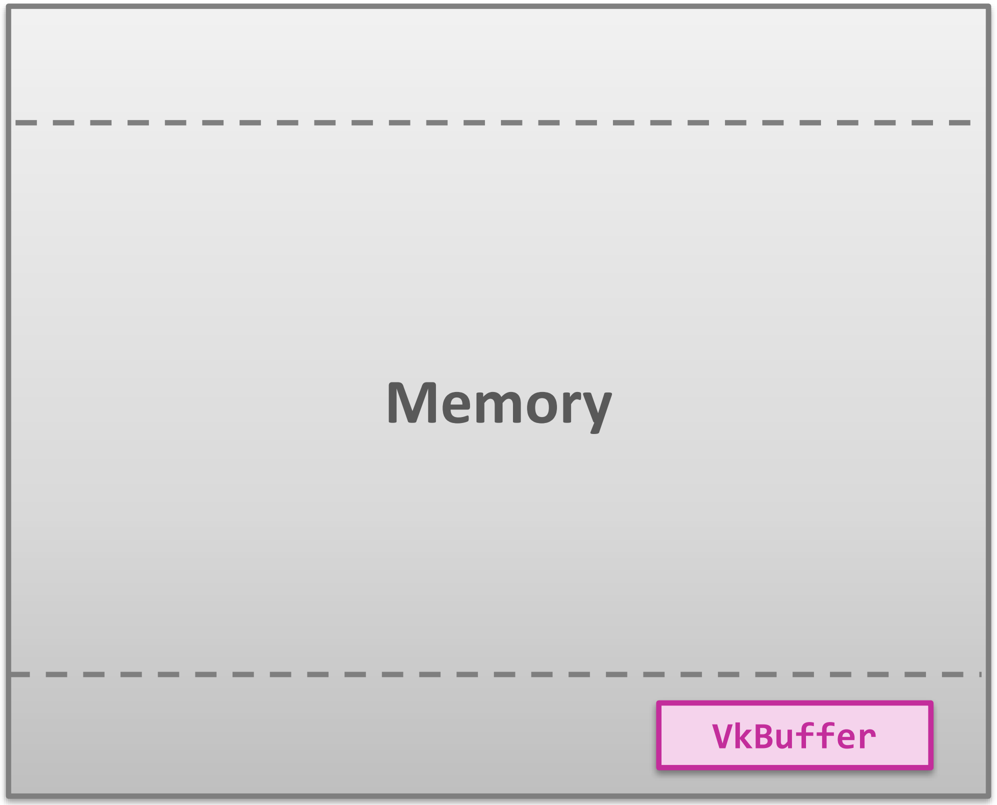


+ additional offset into VkMemory, changeable at run-time with little overhead

Buffers

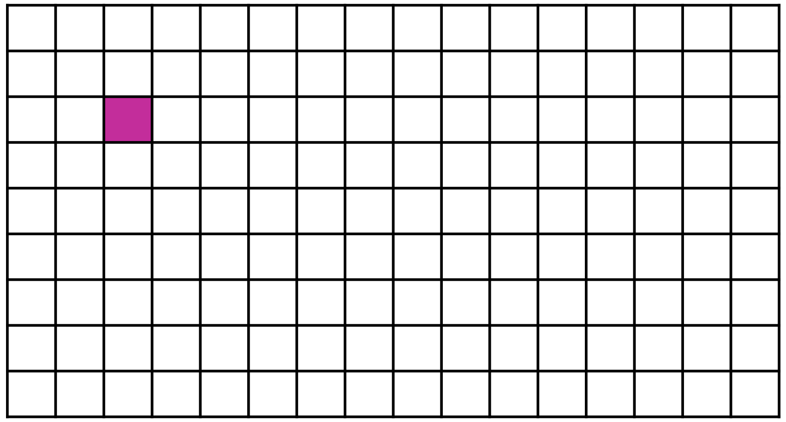
- Different usage types of **buffers**
  - As uniform buffer
  - As storage buffer
  - As texel buffer
    - Uniform texel buffer
    - Storage texel buffer
  - As dynamic buffer
    - Dynamic uniform buffer
    - Dynamic storage buffer
  - (**Inline uniform block**)

VK_DESCRIPTOR_TYPE_INLINE_UNIFORM_BLOCK_EXT



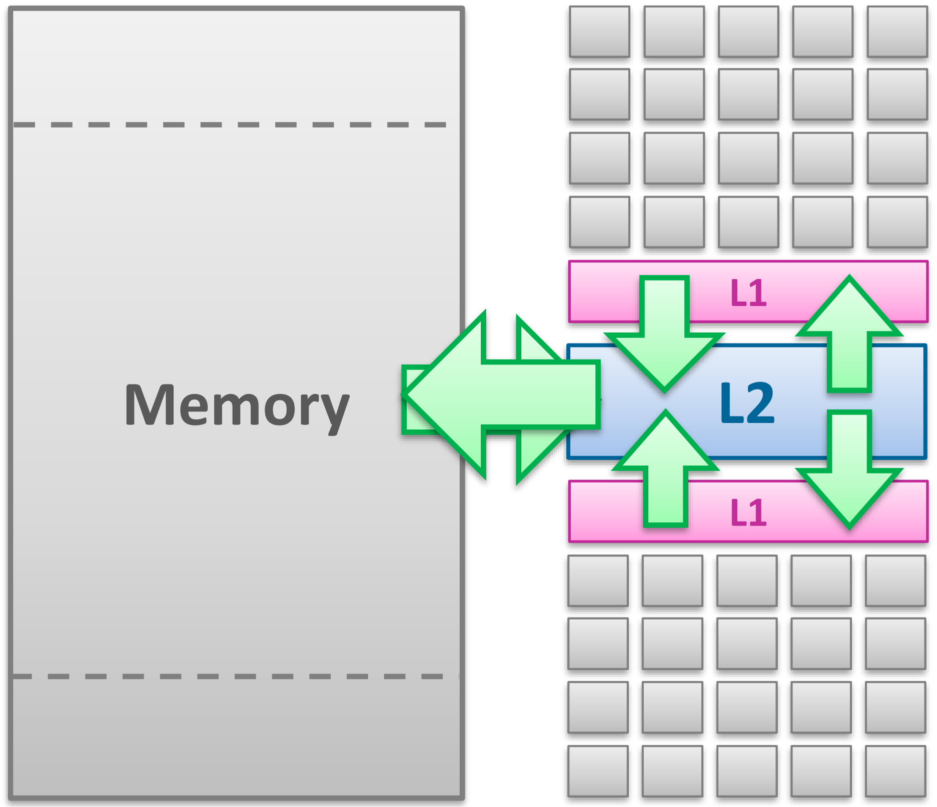
Images

- Different usage types of **images**
  - As **storage image**
  - As sampled image
  - As input attachment



Extents: 16 x 9 pixel
Marked pixel at coordinates: (2, 2)

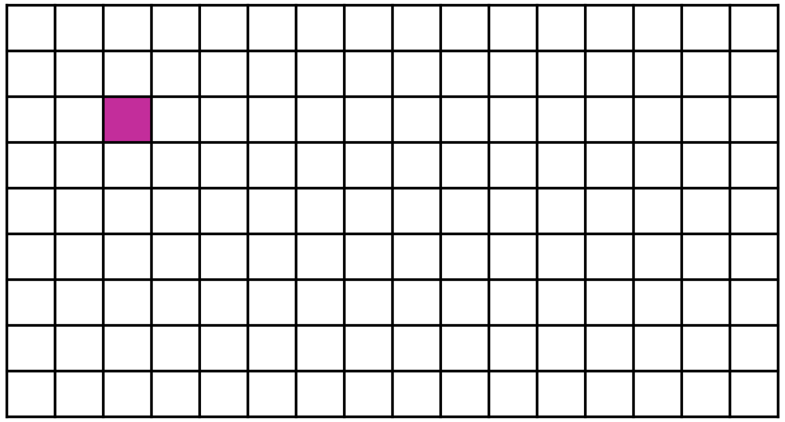
VK_DESCRIPTOR_TYPE_STORAGE_IMAGE



load and store image
(also support atomic operations)

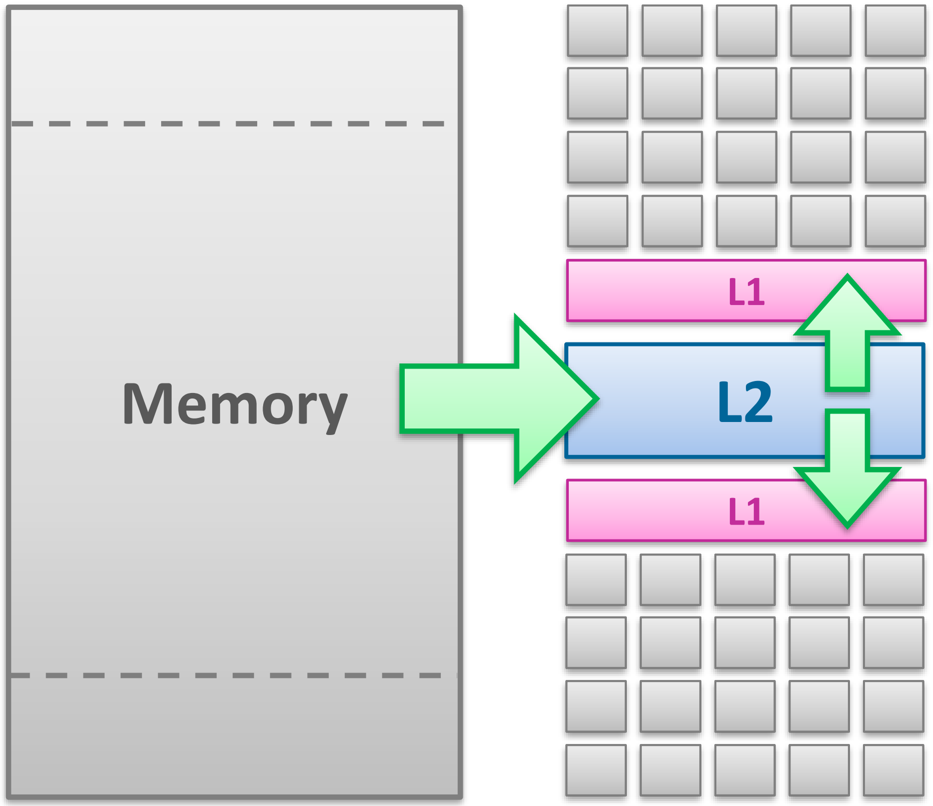
Images

- Different usage types of **images**
  - As storage image
  - As **sampled image**
  - As input attachment



Extents: 16 x 9 pixel
Normalized range: [0, 1] for x and y
Marked pixel's *center* at coordinates: (0.15625, 0.27777)

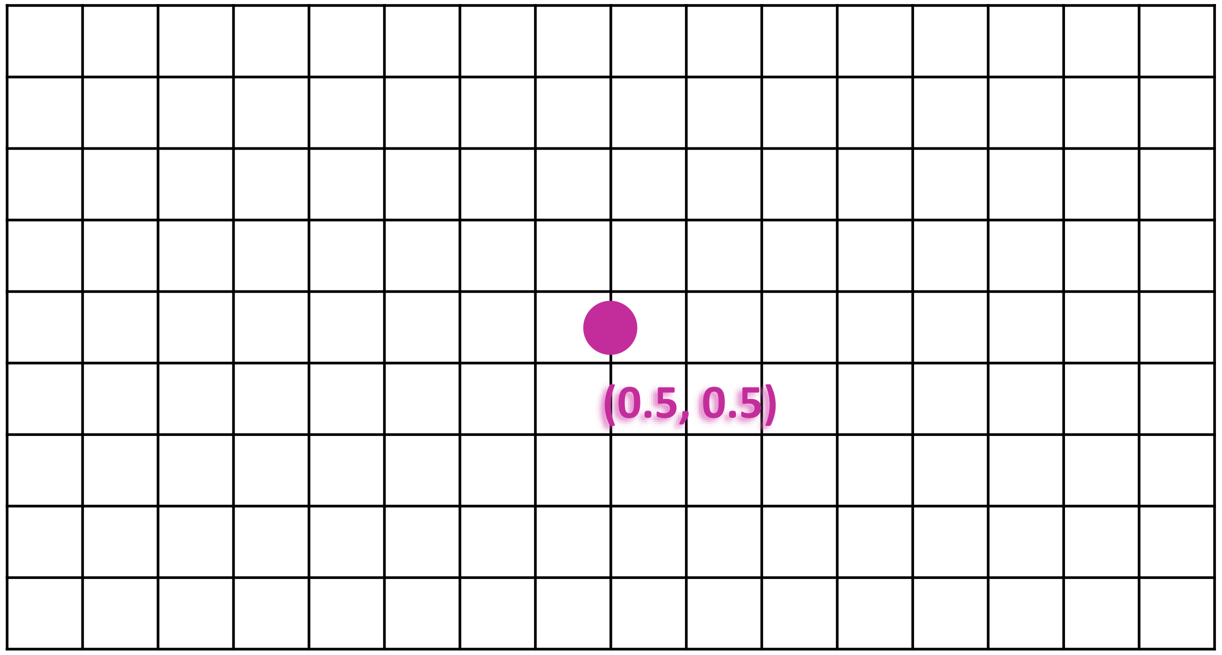
VK_DESCRIPTOR_TYPE_SAMPLED_IMAGE



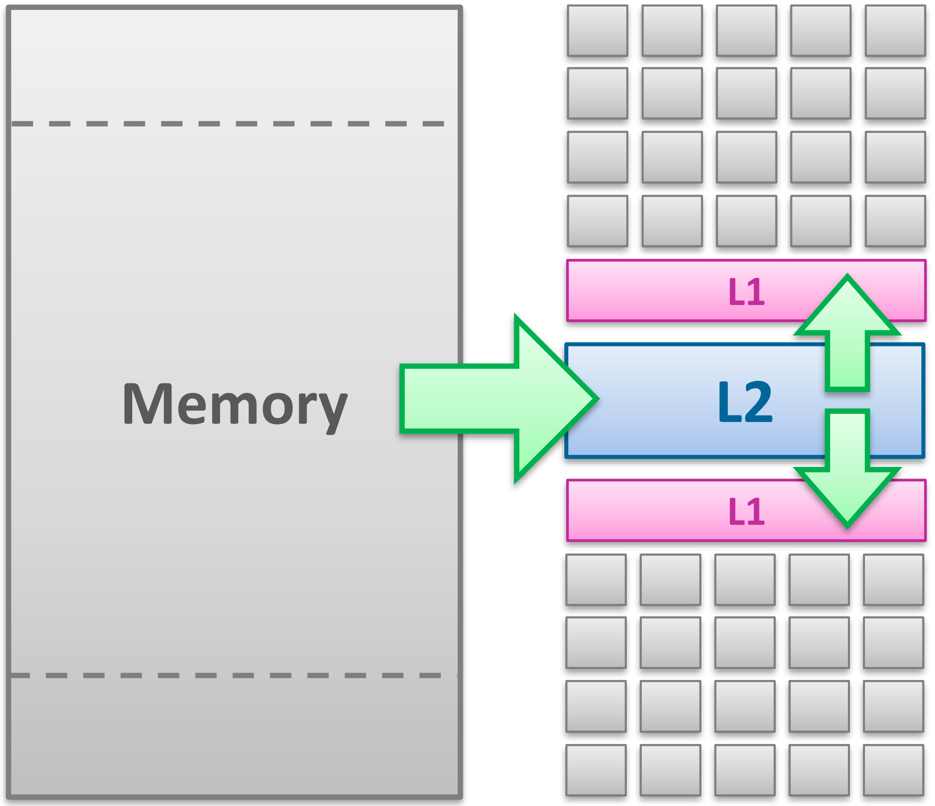
sampled load operations from image

Images

- Different usage types of **images**
  - As storage image
  - As **sampled image**
  - As input attachment



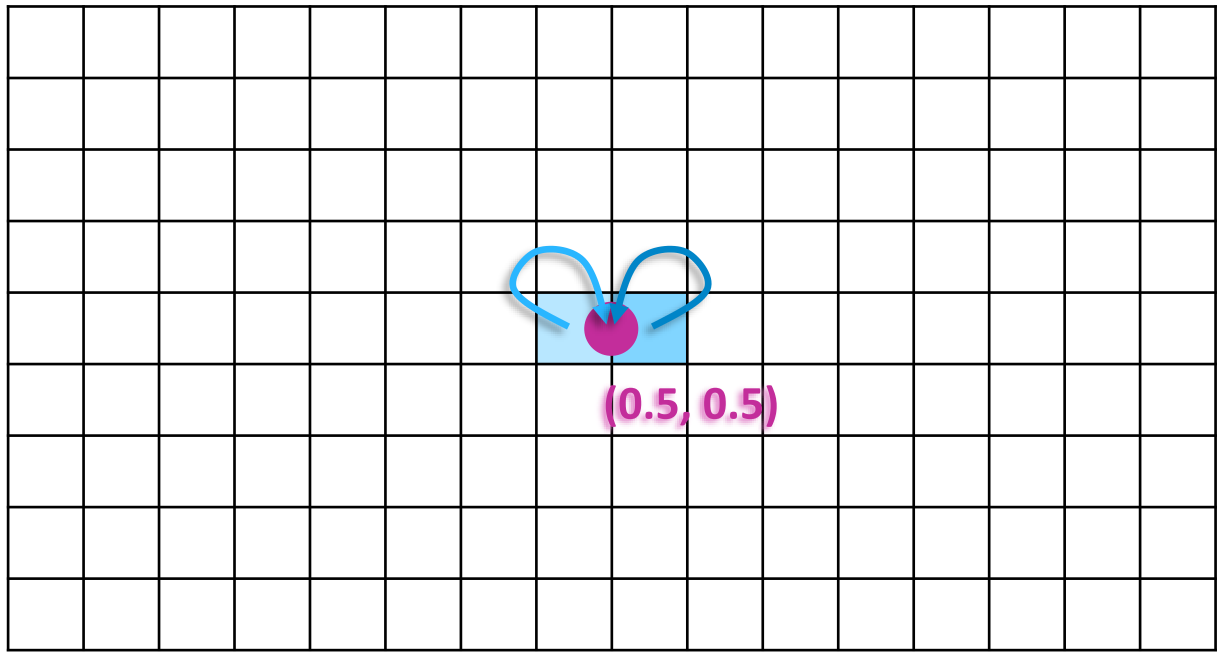
VK_DESCRIPTOR_TYPE_SAMPLED_IMAGE



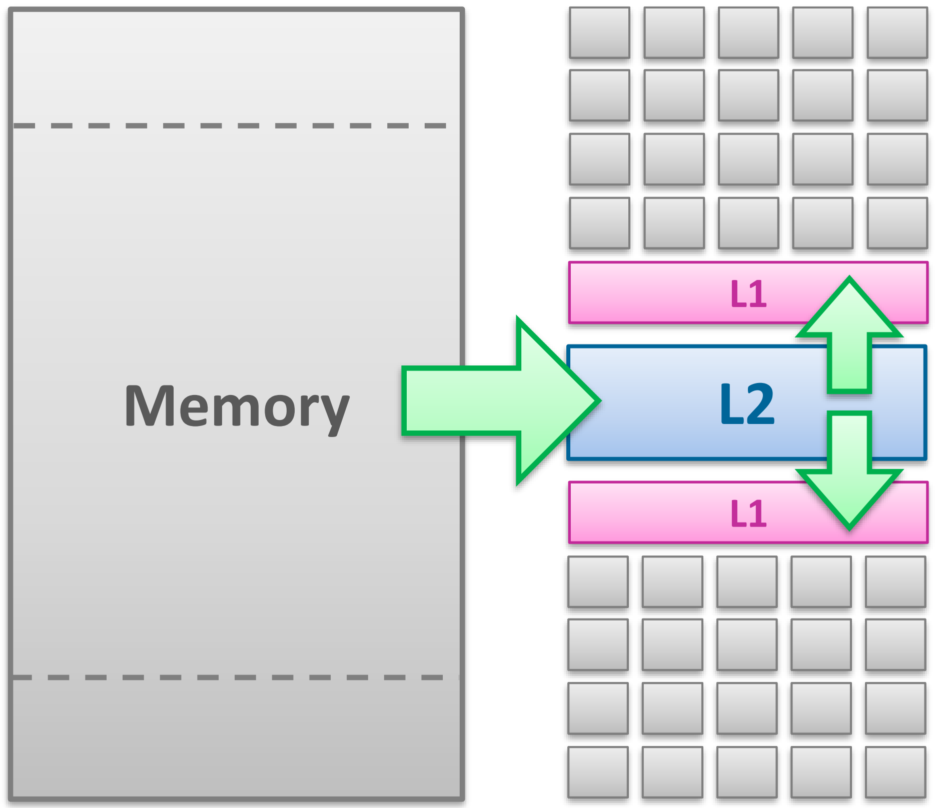
sampled load operations from image

Images

- Different usage types of **images**
  - As storage image
  - As **sampled image**
  - As input attachment



VK_DESCRIPTOR_TYPE_SAMPLED_IMAGE



sampled load operations from image

Images

- Different usage types of **images**

  VK_DESCRIPTOR_TYPE_INPUT_ATTACHMENT

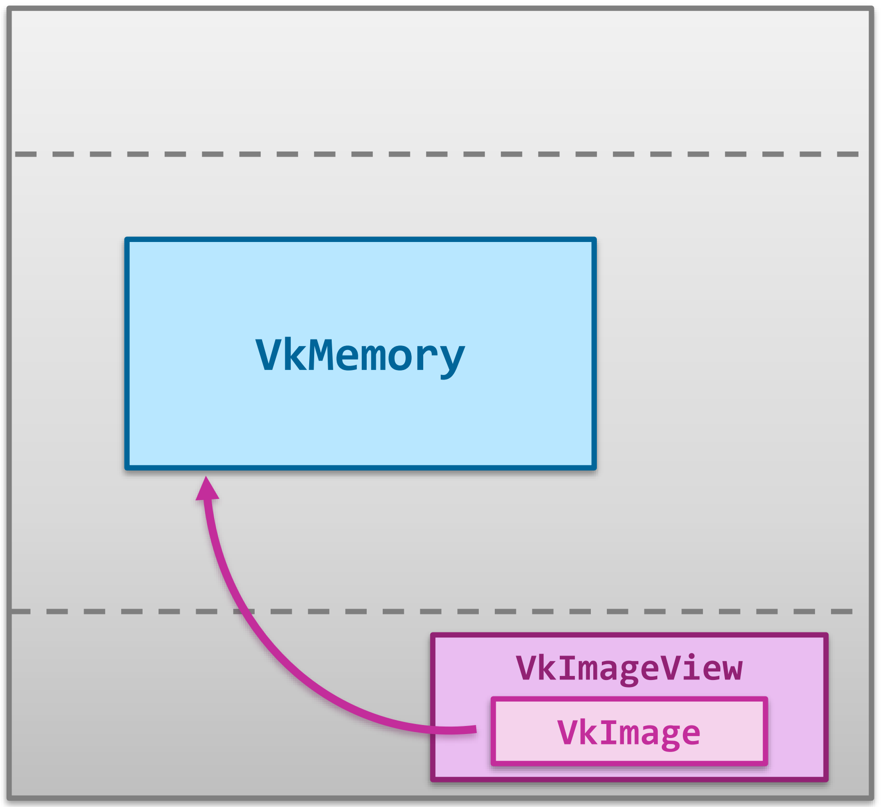
  - As storage image
  - As sampled image
  - As **input attachment**
    - Load-only
    - Within renderpass
    - Framebuffer-local, meaning:

      Access to one single coordinate only,

      No access to other coordinates in that image

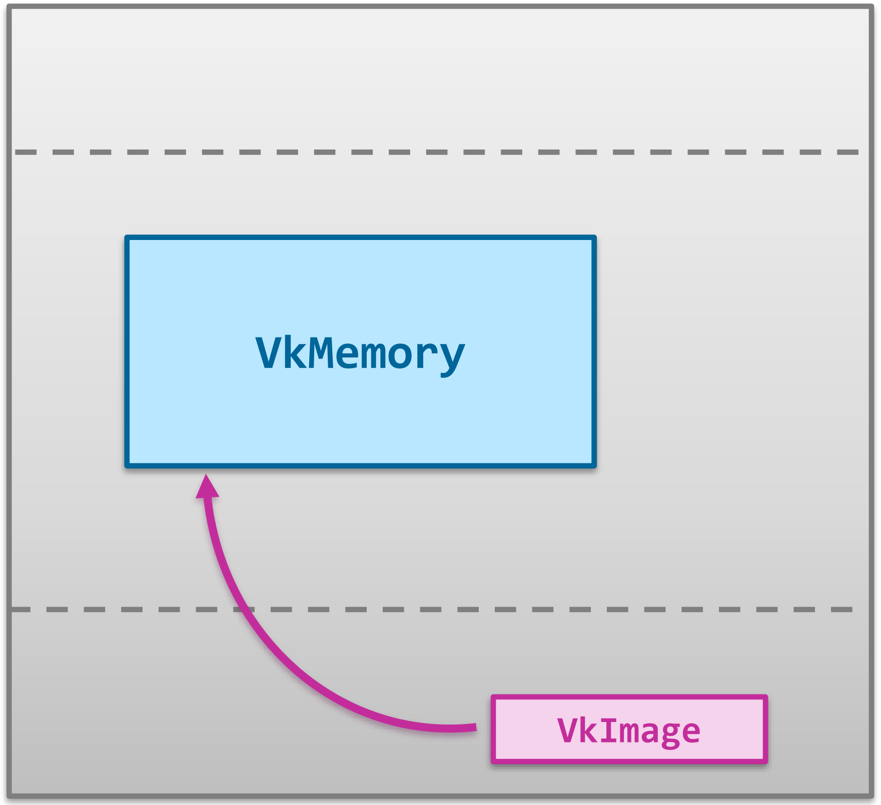
Images

- Different usage types of **images**
  - As storage image
  - As sampled image
  - As input attachment



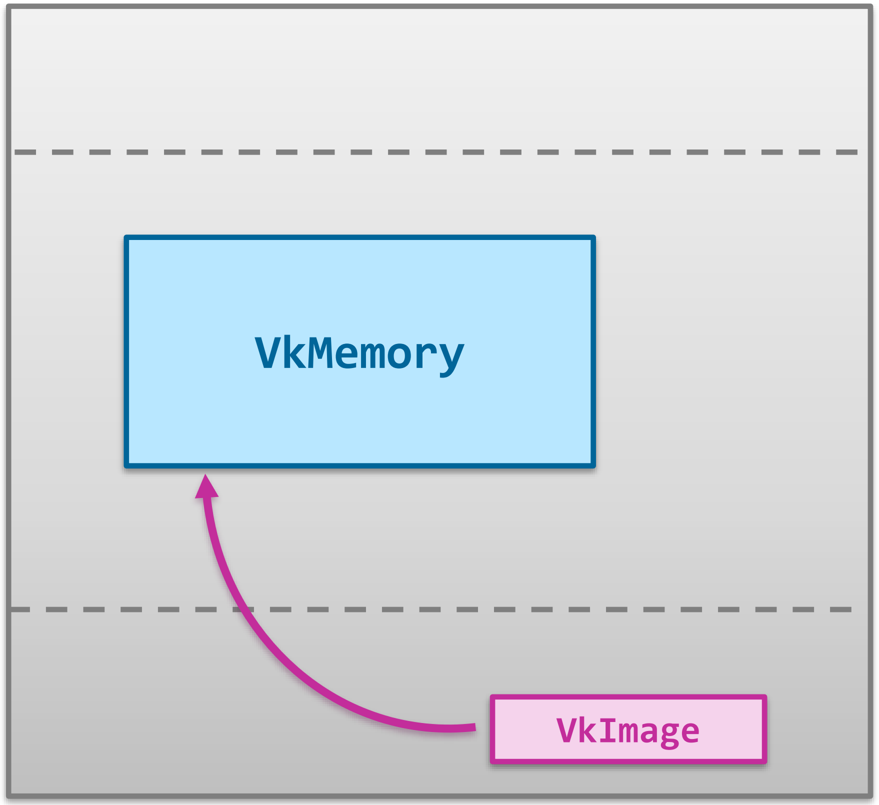
Images

```
VkDevice device = // ...

VkImageCreateInfo imageInfo = {};
imageInfo.sType = VK_STRUCTURE_TYPE_IMAGE_CREATE_INFO;
// Set image info data...

VkImage image;
vkCreateImage(device, &imageInfo, nullptr, &image);

VkMemoryRequirements req;
vkGetImageMemoryRequirements(device, image, &req);

VkMemoryAllocateInfo memInfo = {};
memInfo.sType = VK_STRUCTURE_TYPE_MEMORY_ALLOCATE_INFO;
memInfo.allocationSize = req.size;
memInfo.memoryTypeIndex = // TODO: Find using req

VkDeviceMemory memory;
vkAllocateMemory(device, &allocInfo, nullptr, &memory);

vkBindImageMemory(device, image, memory, 0);
```



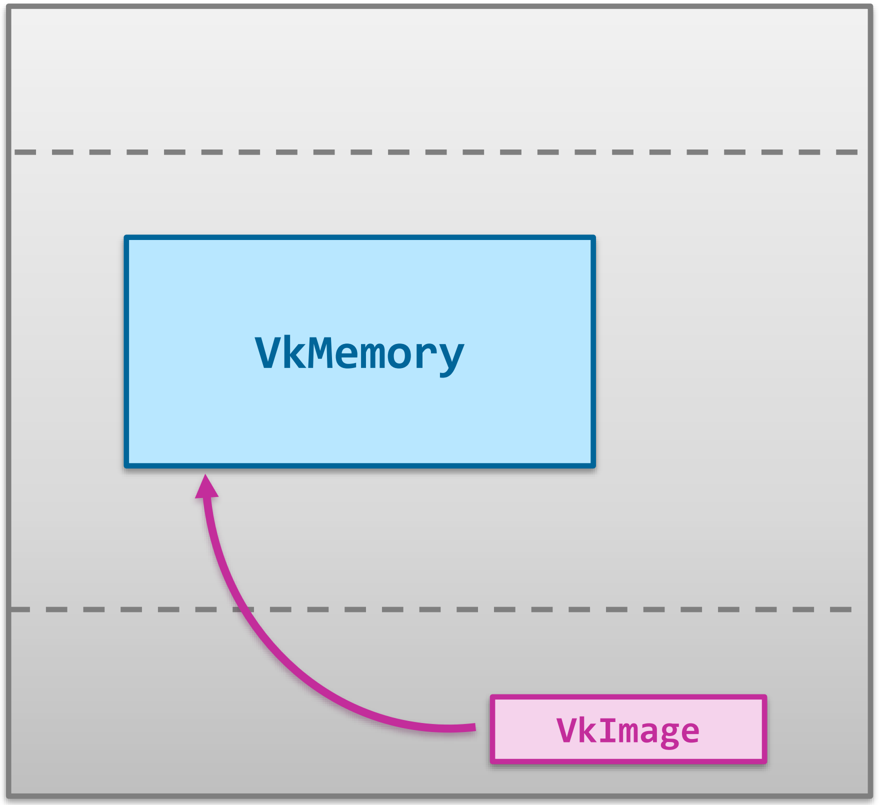
Images

```
VkDevice device = // ...

VkImageCreateInfo imageInfo = {};
imageInfo.sType = VK_STRUCTURE_TYPE_IMAGE_CREATE_INFO;
// Set image info data...

VkImage image;
vkCreateImage(device, &imageInfo, nullptr, &image);

VkMemoryRequirements req;
vkGetImageMemoryRequirements(device, image, &req);

VkMemoryAllocateInfo memInfo = {};
memInfo.sType = VK_STRUCTURE_TYPE_MEMORY_ALLOCATE_INFO;
memInfo.allocationSize = req.size;
memInfo.memoryTypeIndex = // TODO: Find using req

VkDeviceMemory memory;
vkAllocateMemory(device, &allocInfo, nullptr, &memory);

vkBindImageMemory(device, image, memory, 0);
```



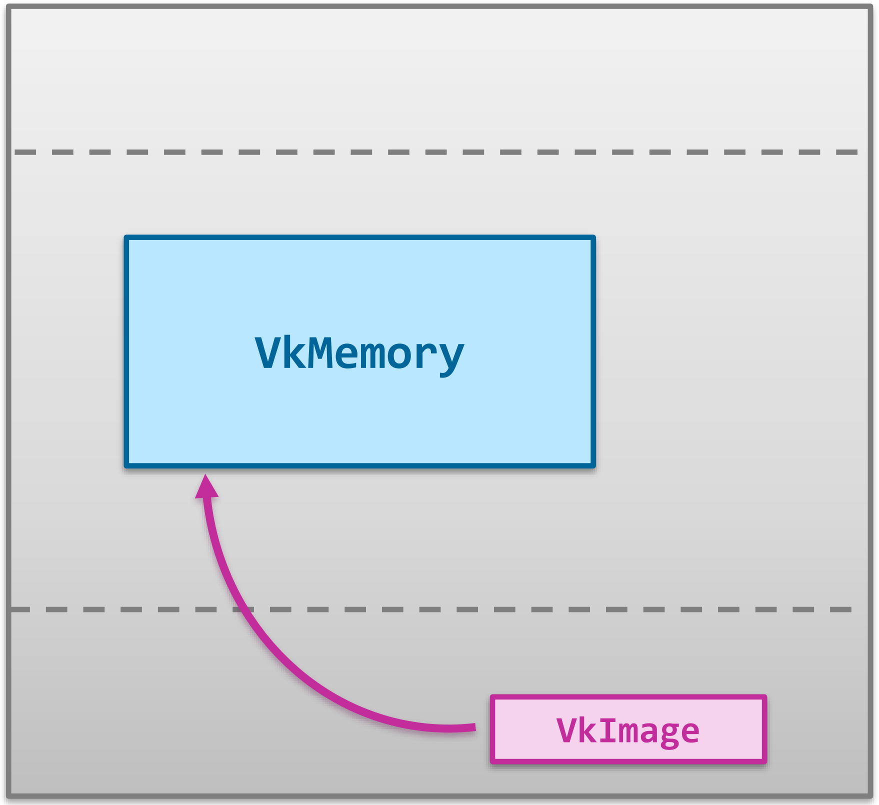
Images

```
VkDevice device = // ...

VkImageCreateInfo imageInfo = {};
imageInfo.sType = VK_STRUCTURE_TYPE_IMAGE_CREATE_INFO;
// Set image info data...

VkImage image;
vkCreateImage(device, &imageInfo, nullptr, &image);

VkMemoryRequirements req;
vkGetImageMemoryRequirements(device, image, &req);

VkMemoryAllocateInfo memInfo = {};
memInfo.sType = VK_STRUCTURE_TYPE_MEMORY_ALLOCATE_INFO;
memInfo.allocationSize = req.size;
memInfo.memoryTypeIndex = // TODO: Find using req

VkDeviceMemory memory;
vkAllocateMemory(device, &allocInfo, nullptr, &memory);

vkBindImageMemory(device, image, memory, 0);
```



Images

```
VkDevice device = // ...

VkImageCreateInfo imageInfo = {};
imageInfo.sType = VK_STRUCTURE_TYPE_IMAGE_CREATE_INFO;
// Set image info data...

VkImage image;
vkCreateImage(device, &imageInfo, nullptr, &image);

VkMemoryRequirements req;
vkGetImageMemoryRequirements(device, image, &req);

VkMemoryAllocateInfo memInfo = {};
memInfo.sType = VK_STRUCTURE_TYPE_MEMORY_ALLOCATE_INFO;
memInfo.allocationSize = req.size;
memInfo.memoryTypeIndex = // TODO: Find using req

VkDeviceMemory memory;
vkAllocateMemory(device, &allocInfo, nullptr, &memory);

vkBindImageMemory(device, image, memory, 0);
```



Images

```
VkDevice device = // ...

VkImageCreateInfo imageInfo = {};
imageInfo.sType = VK_STRUCTURE_TYPE_IMAGE_CREATE_INFO;
// Set image info data...

VkImage image;
vkCreateImage(device, &imageInfo, nullptr, &image);

VkMemoryRequirements req;
vkGetImageMemoryRequirements(device, image, &req);

VkMemoryAllocateInfo memInfo = {};
memInfo.sType = VK_STRUCTURE_TYPE_MEMORY_ALLOCATE_INFO;
memInfo.allocationSize = req.size;
memInfo.memoryTypeIndex = // TODO: Find using req

VkDeviceMemory memory;
vkAllocateMemory(device, &allocInfo, nullptr, &memory);

vkBindImageMemory(device, image, memory, 0);
```

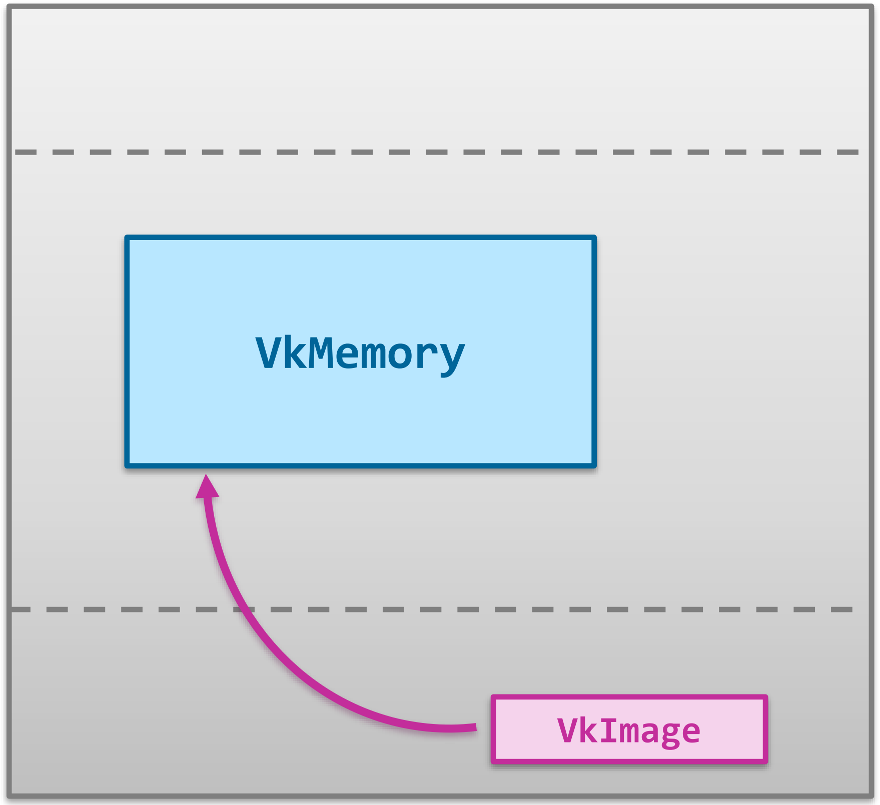


Image Views

```
VkImageViewCreateInfo viewInfo = {};
viewInfo.sType = VK_STRUCTURE_TYPE_IMAGE_VIEW_CREATE_INFO;
viewInfo.image = image;
viewInfo.viewType = VK_IMAGE_VIEW_TYPE_2D;
viewInfo.format = VK_FORMAT_R8G8B8A8_SNORM;
// ^ image must have been created with
//   VK_IMAGE_CREATE_MUTABLE_FORMAT_BIT if a different
//   format than the image's format is specified here.

viewInfo.components.r = VK_COMPONENT_SWIZZLE_B;
viewInfo.components.g = VK_COMPONENT_SWIZZLE_G;
viewInfo.components.b = VK_COMPONENT_SWIZZLE_R;
viewInfo.components.a = VK_COMPONENT_SWIZZLE_A;

viewInfo.subresourceRange.aspectMask = VK_IMAGE_ASPECT_COLOR_BIT;
viewInfo.subresourceRange.baseMipLevel = 0;
viewInfo.subresourceRange.levelCount = 1;
viewInfo.subresourceRange.baseArrayLayer = 0;
viewInfo.subresourceRange.layerCount = 1;

VkImageView imageView;
vkCreateImageView(device, &viewInfo, nullptr, &imageView);
```

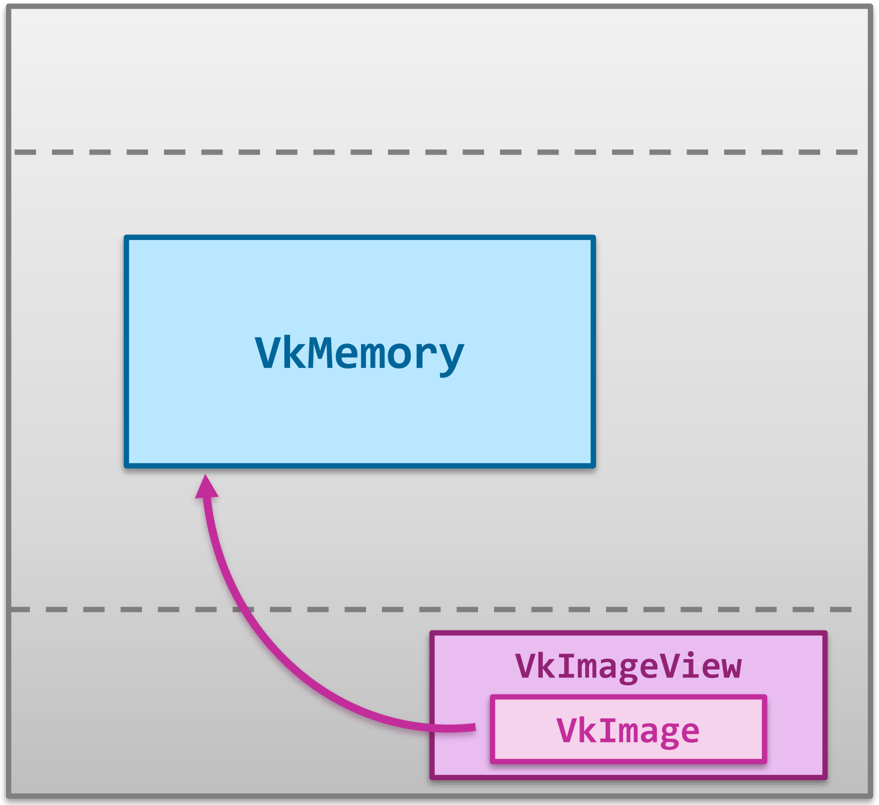


Image Views

```
VkImageViewCreateInfo viewInfo = {};
viewInfo.sType = VK_STRUCTURE_TYPE_IMAGE_VIEW_CREATE_INFO;
viewInfo.image = image;
viewInfo.viewType = VK_IMAGE_VIEW_TYPE_2D;
viewInfo.format = VK_FORMAT_R8G8B8A8_SNORM;
// ^ image must have been created with
//   VK_IMAGE_CREATE_MUTABLE_FORMAT_BIT if a different
//   format than the image's format is specified here.

viewInfo.components.r = VK_COMPONENT_SWIZZLE_B;
viewInfo.components.g = VK_COMPONENT_SWIZZLE_G;
viewInfo.components.b = VK_COMPONENT_SWIZZLE_R;
viewInfo.components.a = VK_COMPONENT_SWIZZLE_A;

viewInfo.subresourceRange.aspectMask = VK_IMAGE_ASPECT_COLOR_BIT;
viewInfo.subresourceRange.baseMipLevel = 0;
viewInfo.subresourceRange.levelCount = 1;
viewInfo.subresourceRange.baseArrayLayer = 0;
viewInfo.subresourceRange.layerCount = 1;

VkImageView imageView;
vkCreateImageView(device, &viewInfo, nullptr, &imageView);
```

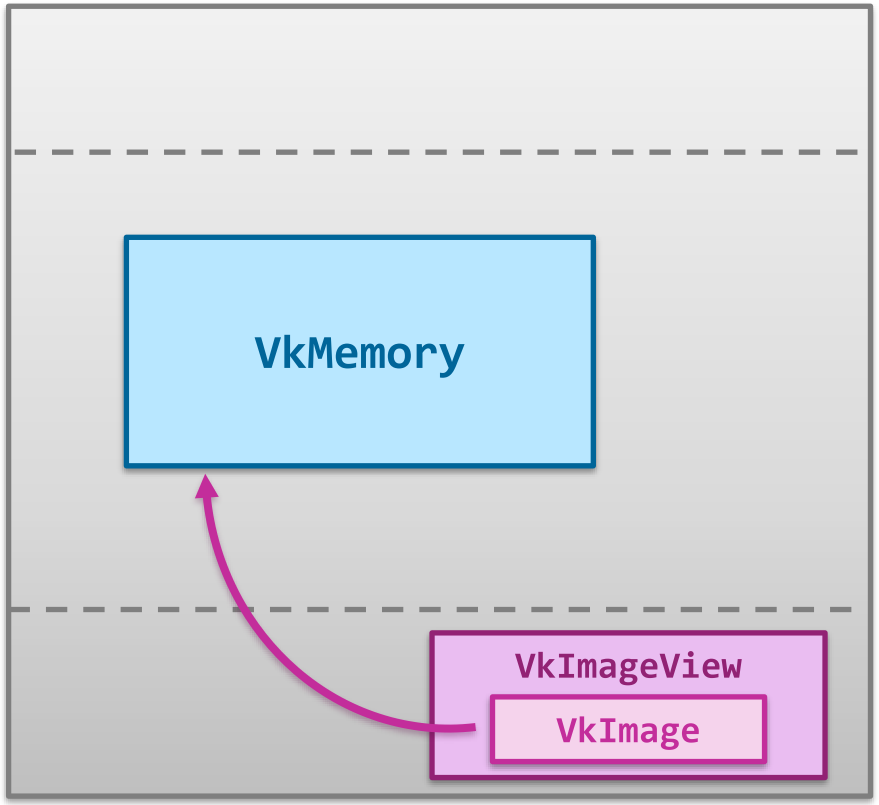


Image Views

```
VkImageViewCreateInfo viewI
viewInfo.sType = VK_STRUCTU
viewInfo.image = image;
viewInfo.viewType = VK_IMAG
viewInfo.format = VK_FORMAT
// ^ image must have been c
//   VK_IMAGE_CREATE_MUTABL
//   format than the image'
```

> This remapping **must** be the identity swizzle for storage image descriptors, input attachment descriptors, framebuffer attachments, and any VkImageView used with a combined image sampler that enables sampler Y'C_BC_R conversion.
>
> *The Khronos Group. Vulkan 1.2.196 Specificaton*

```
viewInfo.components.r = VK_COMPONENT_SWIZZLE_B;
viewInfo.components.g = VK_COMPONENT_SWIZZLE_G;
viewInfo.components.b = VK_COMPONENT_SWIZZLE_R;
viewInfo.components.a = VK_COMPONENT_SWIZZLE_A;

viewInfo.subresourceRange.aspectMask = VK_IMAGE_ASPECT_COLOR_BIT;
viewInfo.subresourceRange.baseMipLevel = 0;
viewInfo.subresourceRange.levelCount = 1;
viewInfo.subresourceRange.baseArrayLayer = 0;
viewInfo.subresourceRange.layerCount = 1;

VkImageView imageView;
vkCreateImageView(device, &viewInfo, nullptr, &imageView);
```

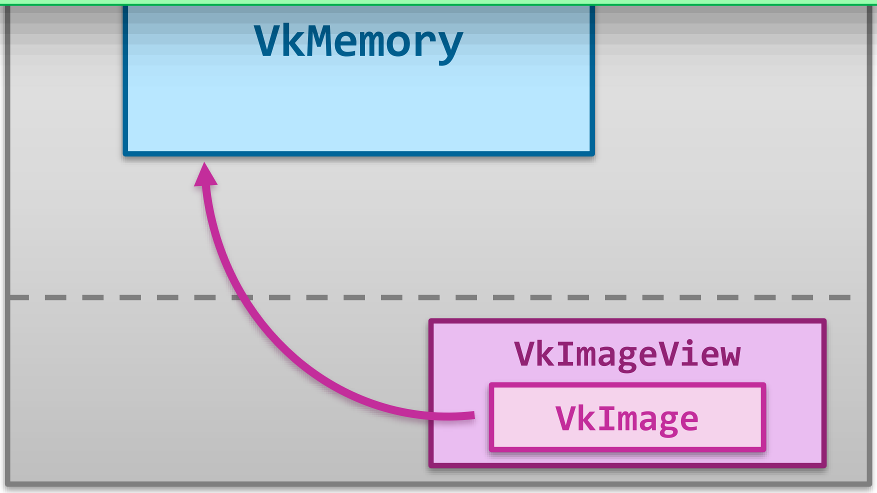


Image Views

```
VkImageViewCreateInfo viewInfo = {};
viewInfo.sType = VK_STRUCTURE_TYPE_IMAGE_VIEW_CREATE_INFO;
viewInfo.image = image;
viewInfo.viewType = VK_IMAGE_VIEW_TYPE_2D;
viewInfo.format = VK_FORMAT_R8G8B8A8_SNORM;
// ^ image must have been created with
//   VK_IMAGE_CREATE_MUTABLE_FORMAT_BIT if a different
//   format than the image's format is specified here.

viewInfo.components.r = VK_COMPONENT_SWIZZLE_B;
viewInfo.components.g = VK_COMPONENT_SWIZZLE_G;
viewInfo.components.b = VK_COMPONENT_SWIZZLE_R;
viewInfo.components.a = VK_COMPONENT_SWIZZLE_A;

viewInfo.subresourceRange.aspectMask = VK_IMAGE_ASPECT_COLOR_BIT;
viewInfo.subresourceRange.baseMipLevel = 0;
viewInfo.subresourceRange.levelCount = 1;
viewInfo.subresourceRange.baseArrayLayer = 0;
viewInfo.subresourceRange.layerCount = 1;

VkImageView imageView;
vkCreateImageView(device, &viewInfo, nullptr, &imageView);
```

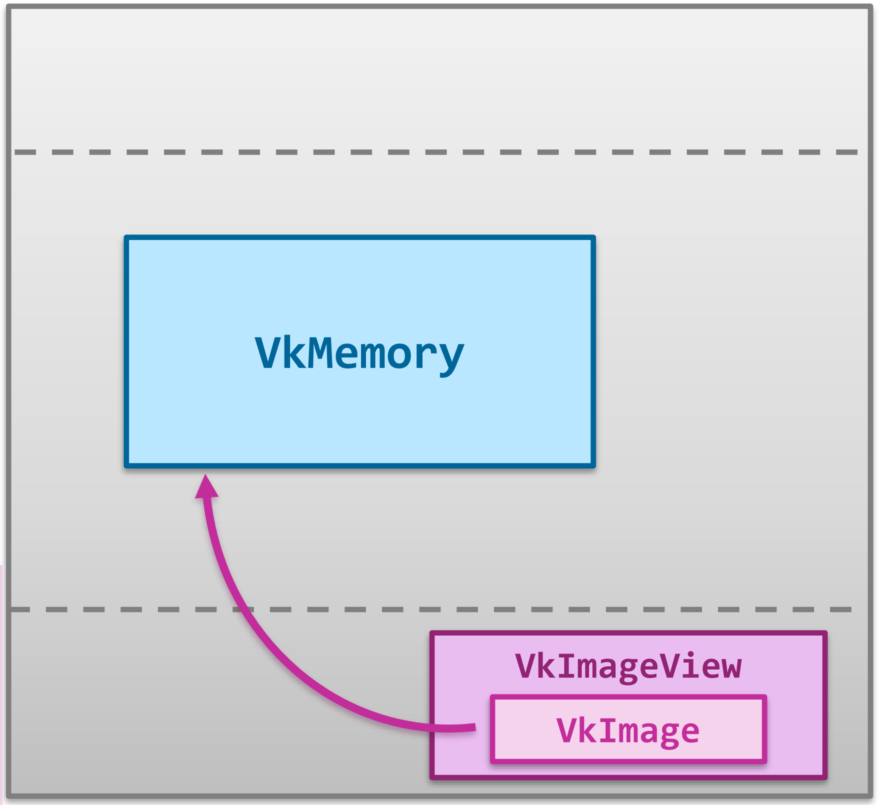
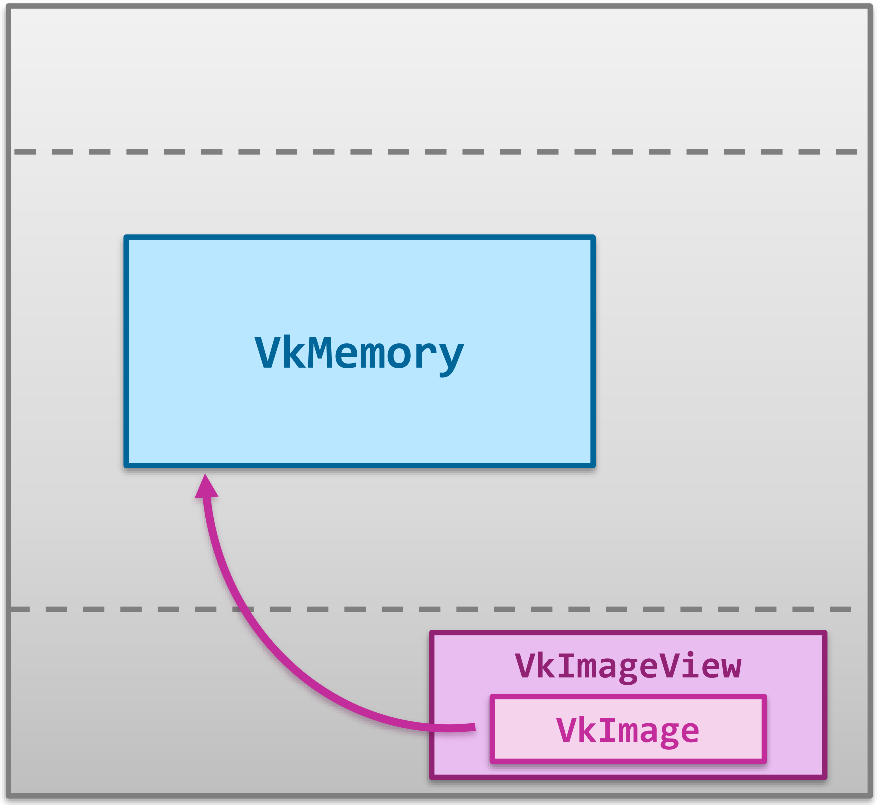


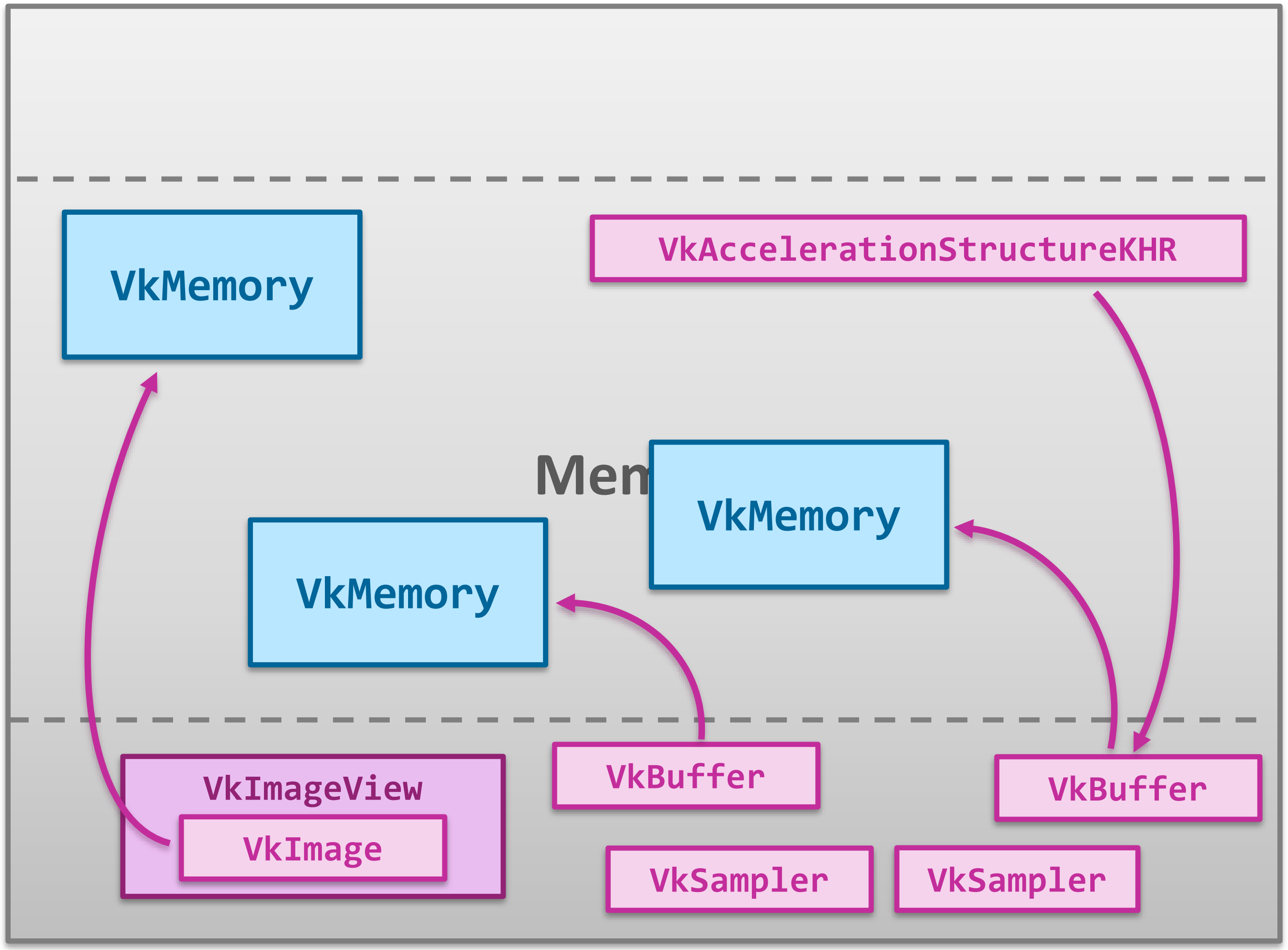
Image Views

```
VkImageViewCreateInfo viewInfo = {};
viewInfo.sType = VK_STRUCTURE_TYPE_IMAGE_VIEW_CREATE_INFO;
viewInfo.image = image;
viewInfo.viewType = VK_IMAGE_VIEW_TYPE_2D;
viewInfo.format = VK_FORMAT_R8G8B8A8_SNORM;
// ^ image must have been created with
//   VK_IMAGE_CREATE_MUTABLE_FORMAT_BIT if a different
//   format than the image's format is specified here.

viewInfo.components.r = VK_COMPONENT_SWIZZLE_B;
viewInfo.components.g = VK_COMPONENT_SWIZZLE_G;
viewInfo.components.b = VK_COMPONENT_SWIZZLE_R;
viewInfo.components.a = VK_COMPONENT_SWIZZLE_A;

viewInfo.subresourceRange.aspectMask = VK_IMAGE_ASPECT_COLOR_BIT;
viewInfo.subresourceRange.baseMipLevel = 0;
viewInfo.subresourceRange.levelCount = 1;
viewInfo.subresourceRange.baseArrayLayer = 0;
viewInfo.subresourceRange.layerCount = 1;

VkImageView imageView;
vkCreateImageView(device, &viewInfo, nullptr, &imageView);
```



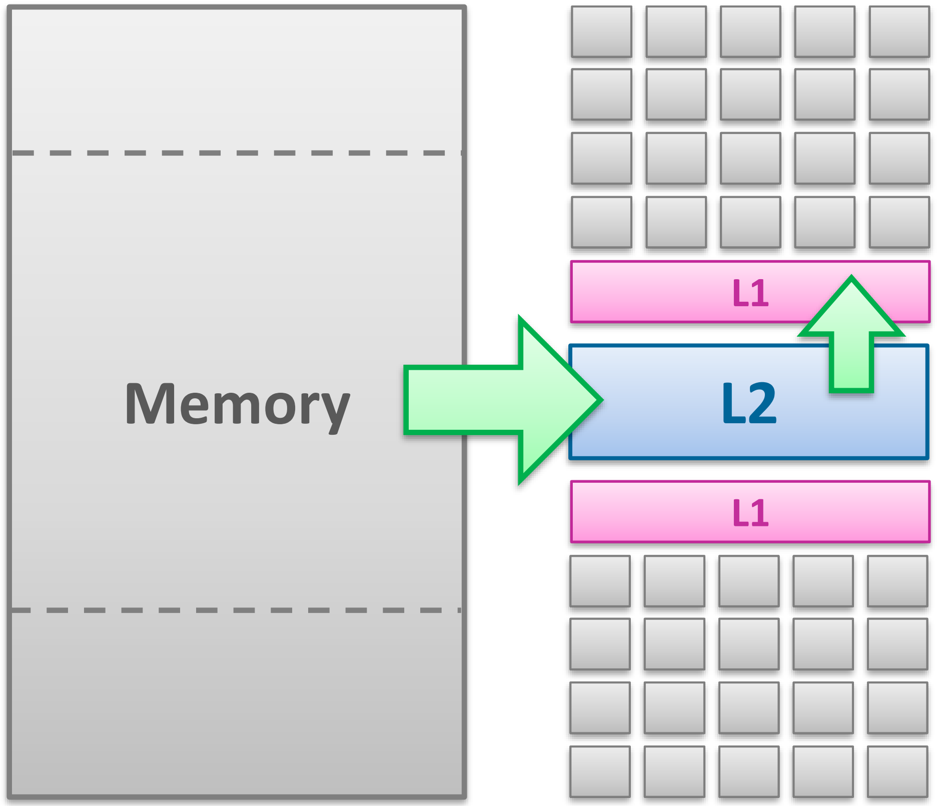
Resources & Descriptors

- ~Four fundamentally different types of **resources**
  - Buffers
  - Images
  - Samplers
  - Acceleration structures



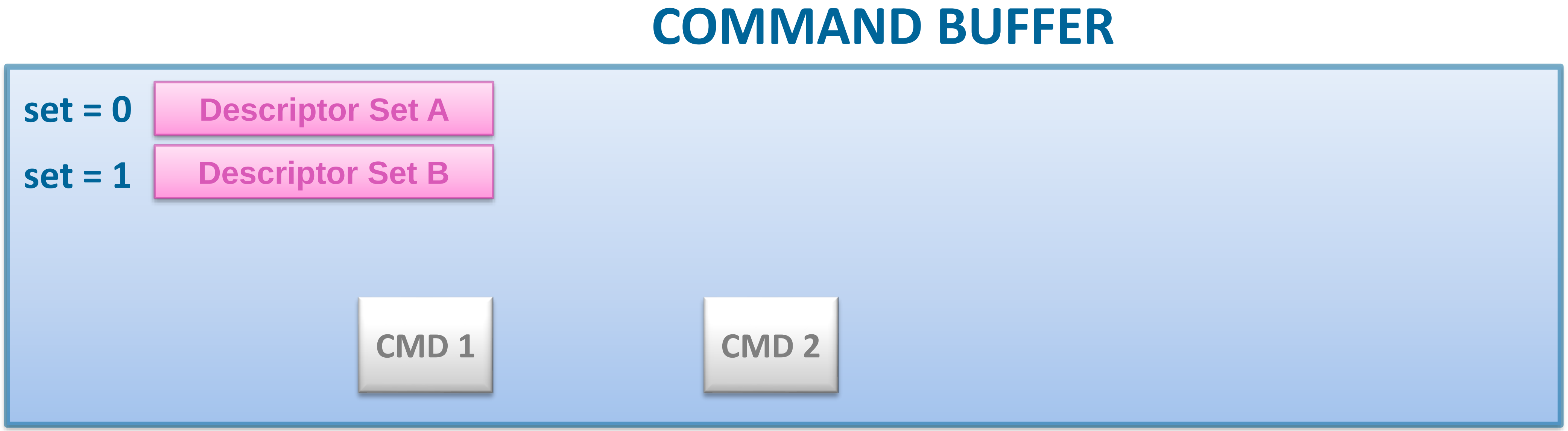
Resources & Descriptors

- ~Four fundamentally different types of **resources**
  - Buffers
  - Images
  - Samplers
  - Acceleration structures
- **Descriptors** describe where to find a resource
  - + usage type of a resource
  - + some meta data, sometimes
  - + combinations of resources, sometimes



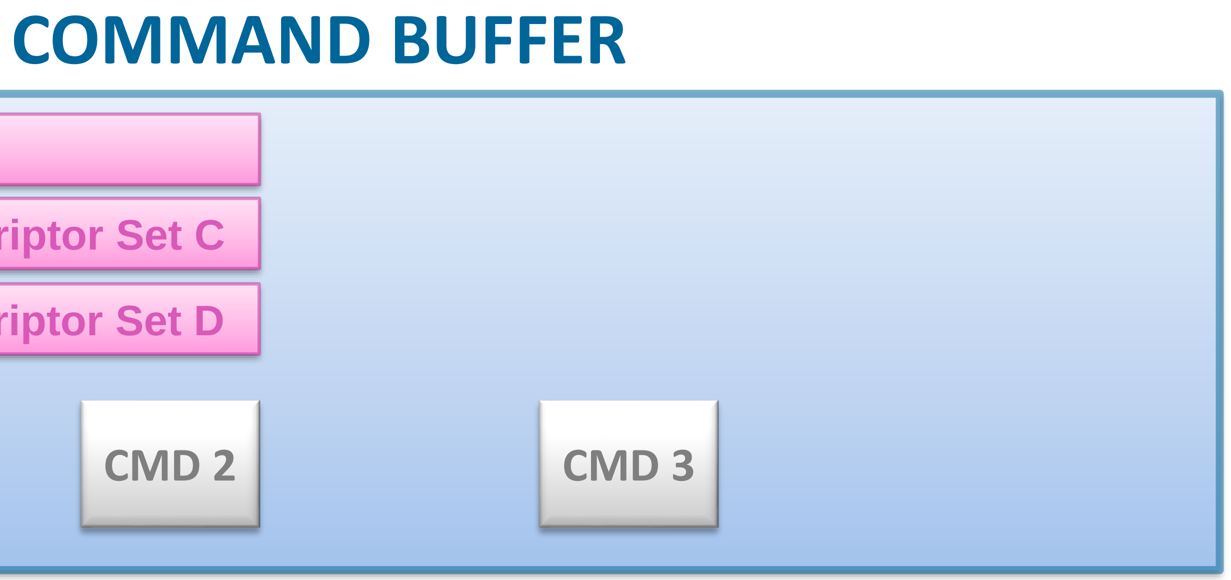
Descriptors and Descriptor Sets

- One descriptor describes one resource
- Descriptors are always organized in descriptor sets
  - One or multiple descriptors contained
  - Combine descriptors which are used in conjunction!
- Multiple sets can be used



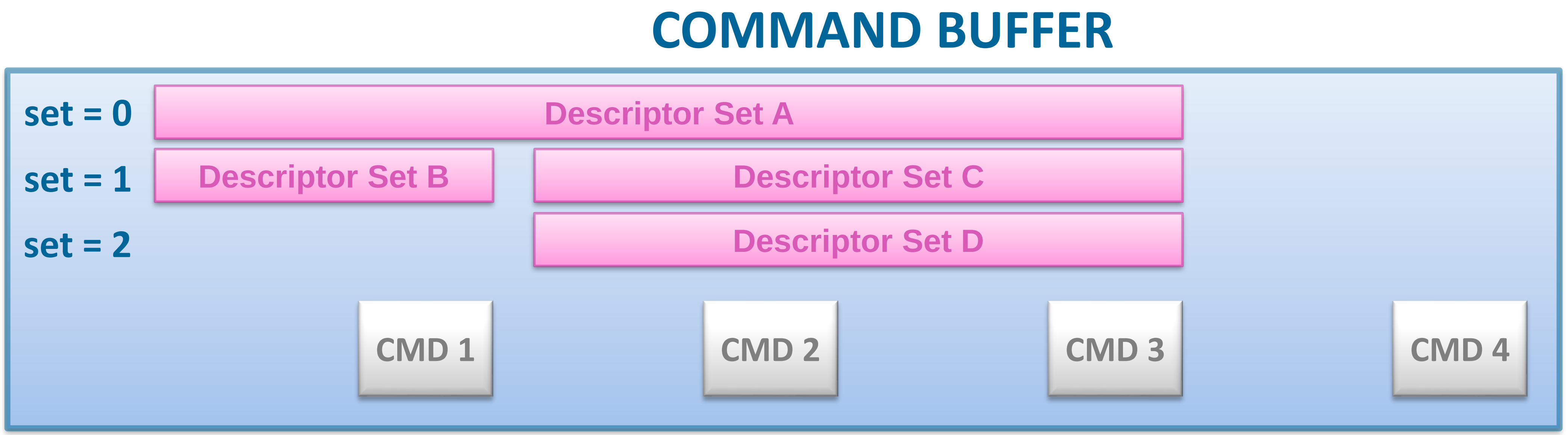
Descriptors and Descriptor Sets

- One descriptor describes one resource
- Descriptors are always organized in descriptor sets
  - One or multiple descriptors contained
  - Combine descriptors which are used in conjunction!
- Multiple sets can be used



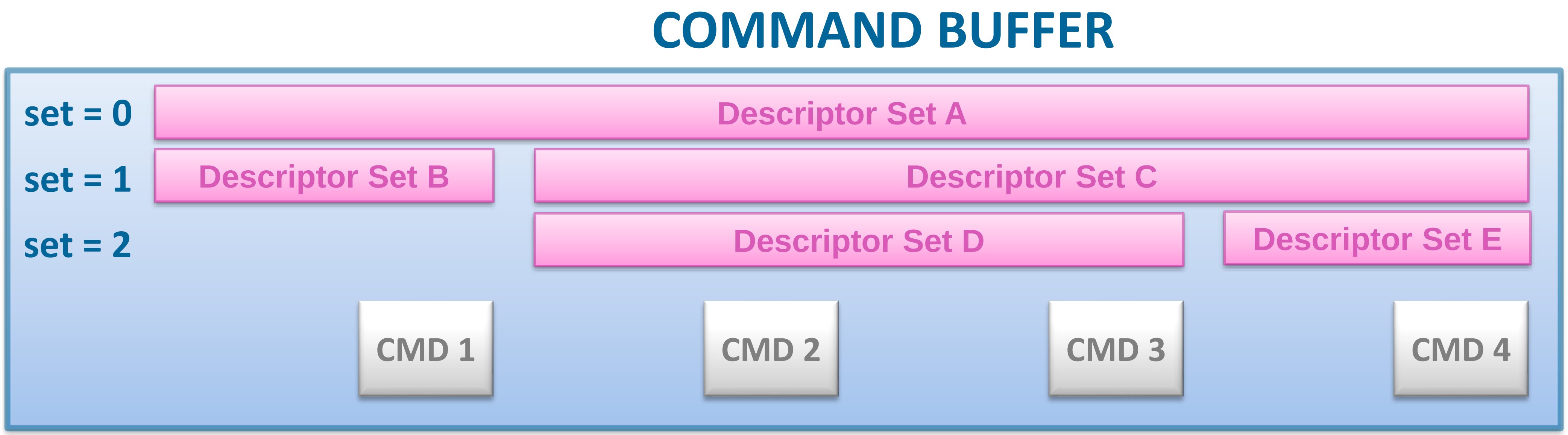
Descriptors and Descriptor Sets

- One descriptor describes one resource
- Descriptors are always organized in descriptor sets
  - One or multiple descriptors contained
  - Combine descriptors which are used in conjunction!
- Multiple sets can be used



Descriptors and Descriptor Sets

- One descriptor describes one resource
- Descriptors are always organized in descriptor sets
  - One or multiple descriptors contained
  - Combine descriptors which are used in conjunction!
- Multiple sets can be used



Descriptors and Descriptor Sets

Descriptor Set A

1x	VK_DESCRIPTOR_TYPE_STORAGE_IMAGE
50x	VK_DESCRIPTOR_TYPE_SAMPLED_IMAGE
2x	VK_DESCRIPTOR_TYPE_SAMPLER

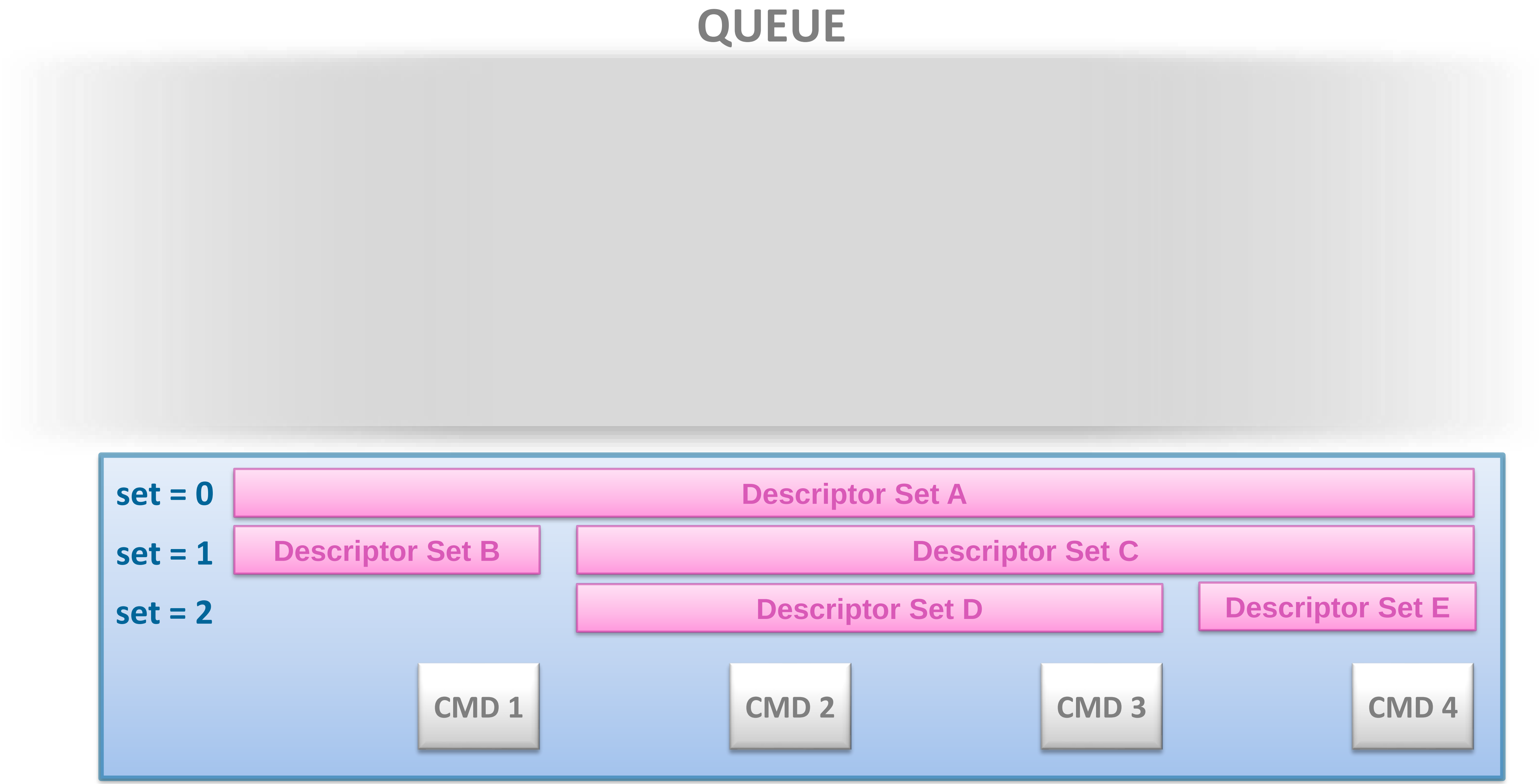
Descriptor Set C

1x	VK_DESCRIPTOR_TYPE_UNIFORM_BUFFER
3x	VK_DESCRIPTOR_TYPE_STORAGE_BUFFER
1x	VK_DESCRIPTOR_TYPE_ACCELERATION_STRUCTURE_KHR

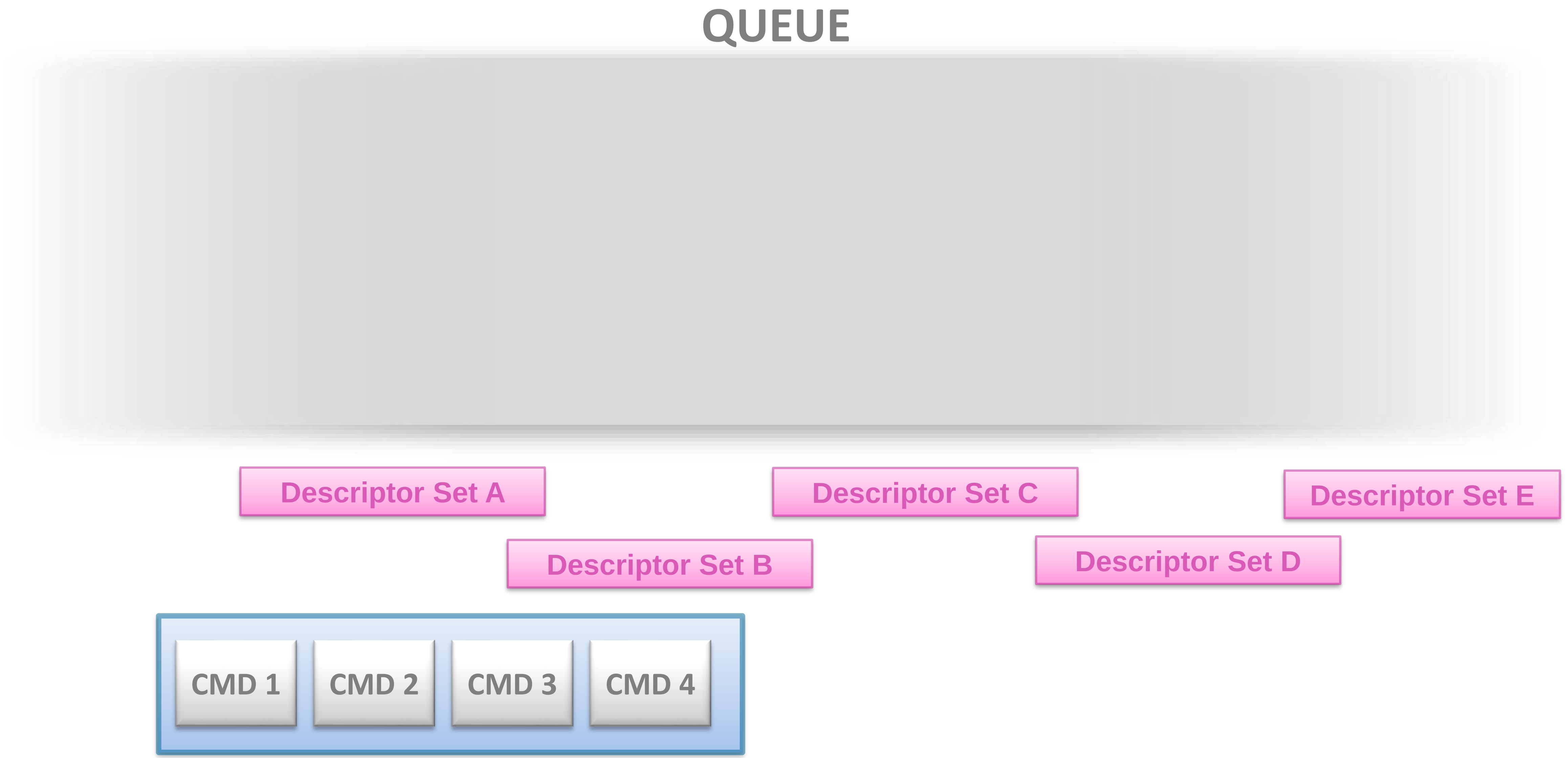
Descriptor Set B

1x	VK_DESCRIPTOR_TYPE_UNIFORM_BUFFER
3x	VK_DESCRIPTOR_TYPE_UNIFORM_BUFFER

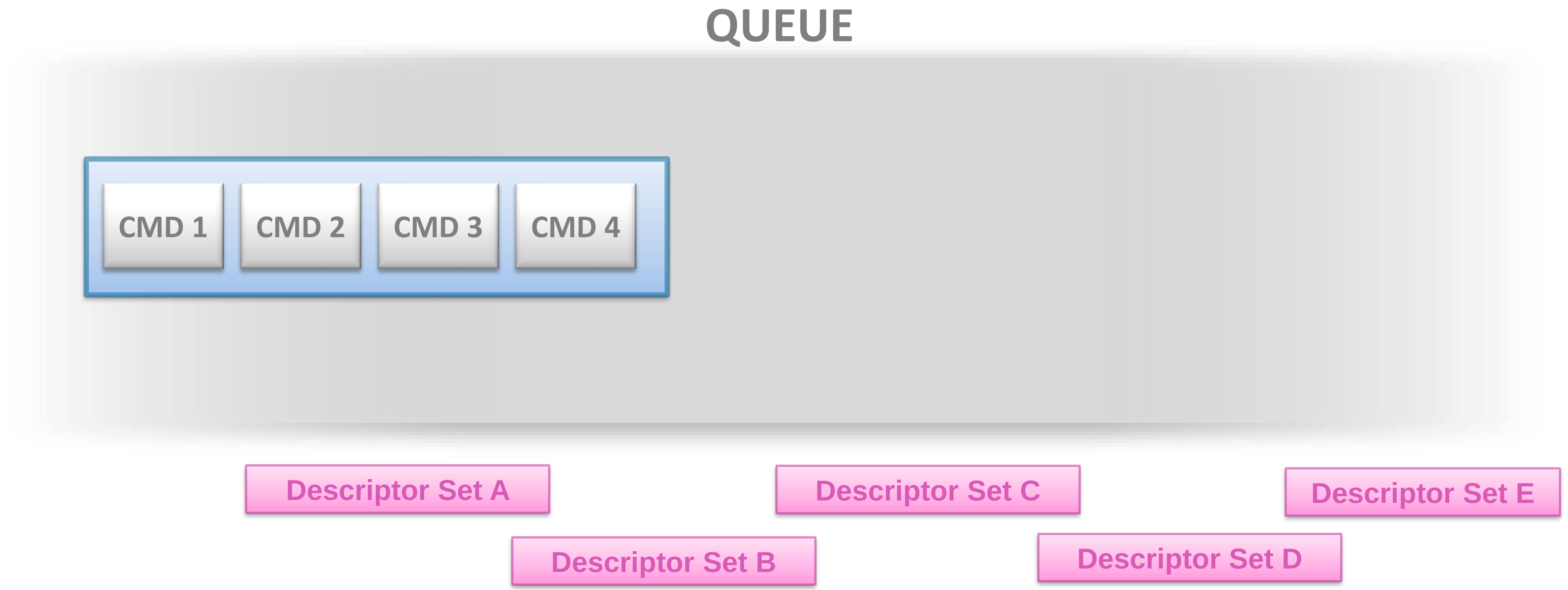
Descriptors and Descriptor Sets



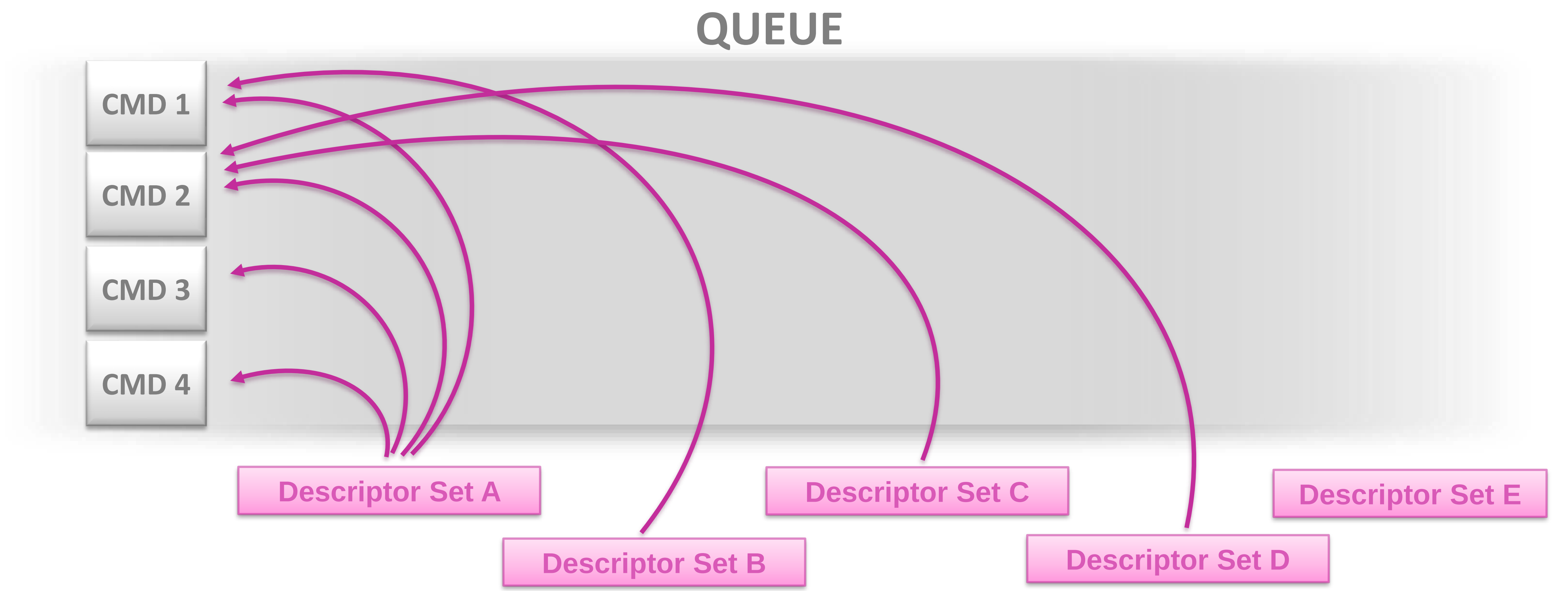
Descriptors and Descriptor Sets



Descriptors and Descriptor Sets

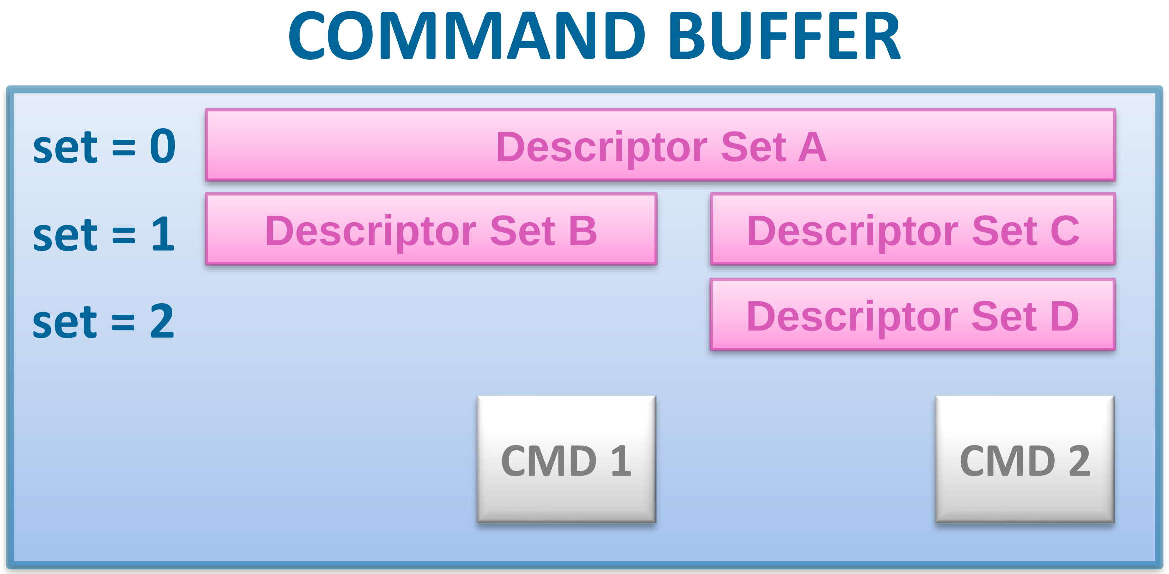


Descriptors and Descriptor Sets



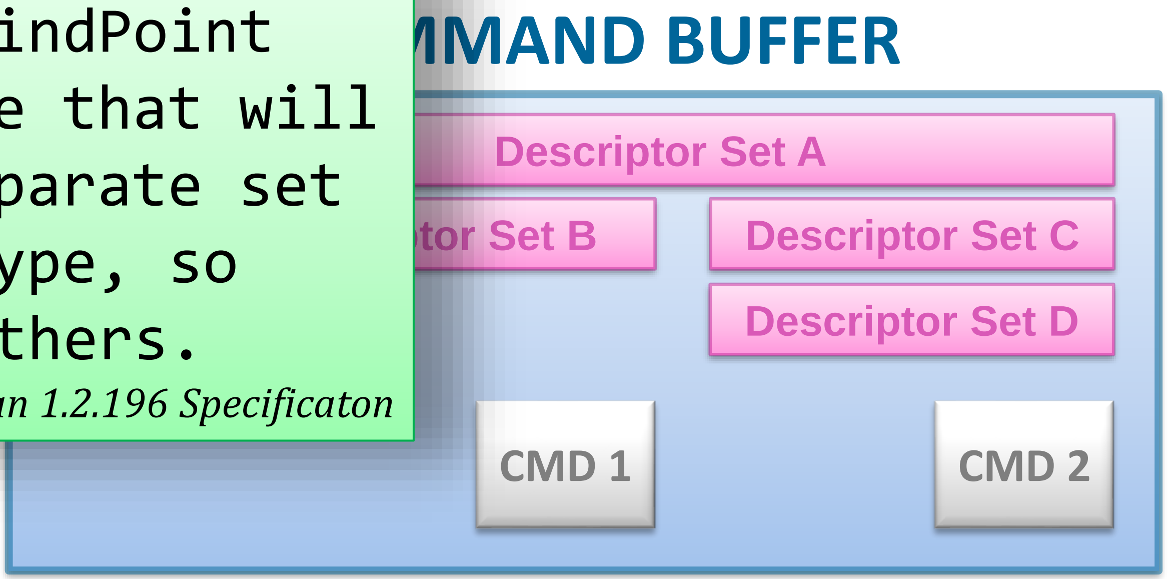
Descriptors and Descriptor Sets

- One descriptor describes one resource
- Descriptors are always organized in descriptor sets
  - One or multiple descriptors contained
  - Combine descriptors which are used in conjunction!
- Multiple sets can be used
- **Important:**
  - Bound descriptor state tracked at command-buffer level!
  - Not at queue-level, not globally, not otherwise



Descriptors and Descriptor Sets

- Attention: Different bind points! (see VkPipelineBindPoint)
  - VK_PIPELINE_BIND_POINT_GRAPHICS
  - VK_PIPELINE_BIND_POINT_COMPUTE
  - VK_PIPELINE_BIND_POINT_RAY_TRACING_KHR

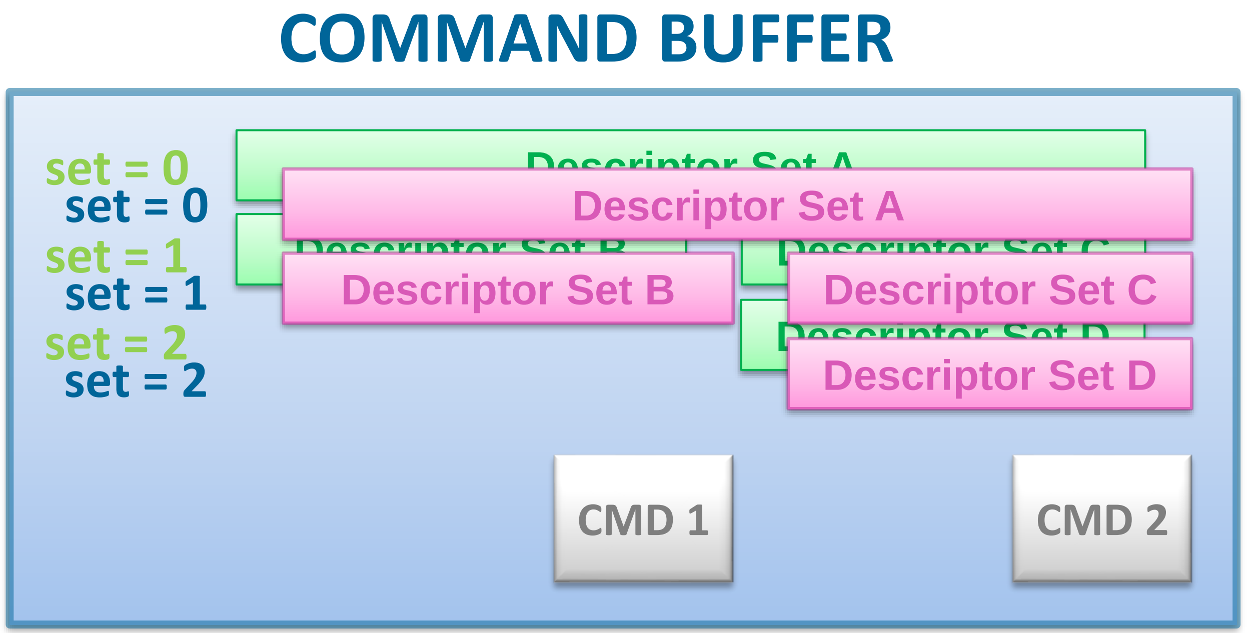


```
pipelineBindPoint is a VkPipelineBindPoint
indicating the type of the pipeline that will
use the descriptors. There is a separate set
of bind points for each pipeline type, so
binding one does not disturb the others.
```

The Khronos Group. Vulkan 1.2.196 Specificaton

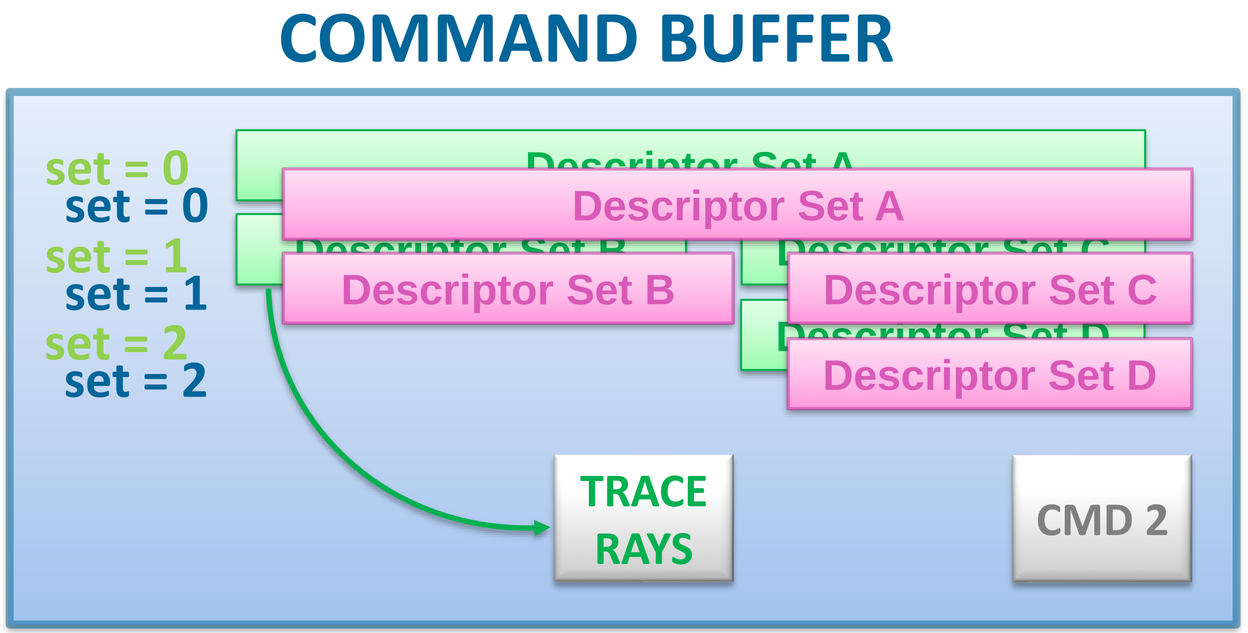
Descriptors and Descriptor Sets

- Attention: Different bind points! (see VkPipelineBindPoint)
  - VK_PIPELINE_BIND_POINT_GRAPHICS
  - VK_PIPELINE_BIND_POINT_COMPUTE
  - VK_PIPELINE_BIND_POINT_RAY_TRACING_KHR



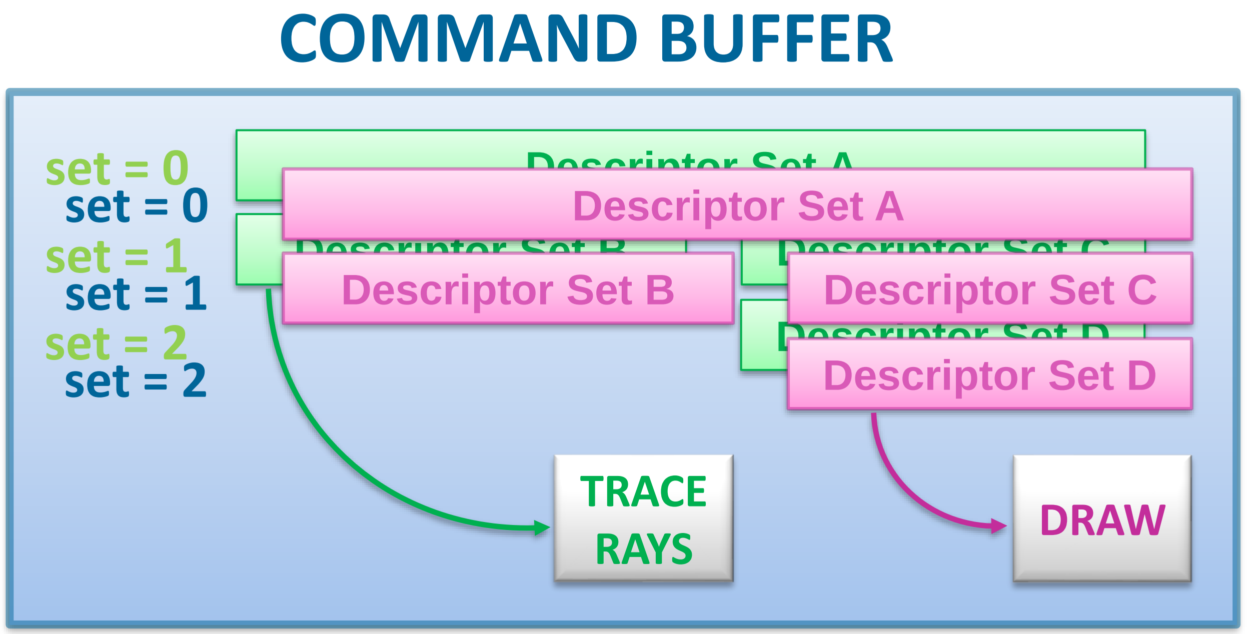
Descriptors and Descriptor Sets

- Attention: Different bind points! (see VkPipelineBindPoint)
  - VK_PIPELINE_BIND_POINT_GRAPHICS
  - VK_PIPELINE_BIND_POINT_COMPUTE
  - VK_PIPELINE_BIND_POINT_RAY_TRACING_KHR



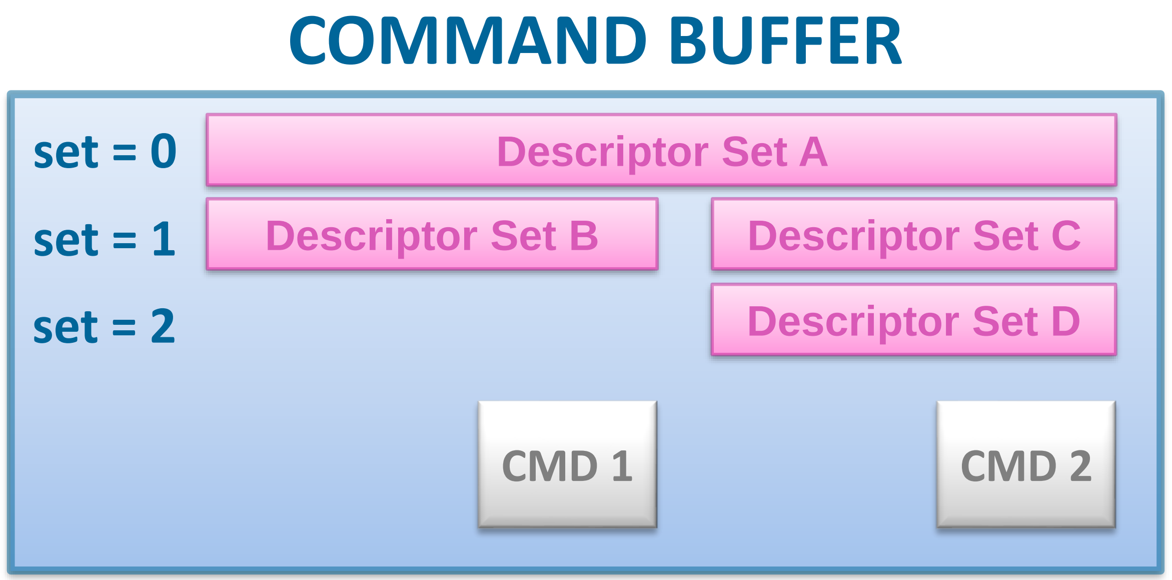
Descriptors and Descriptor Sets

- Attention: Different bind points! (see VkPipelineBindPoint)
  - VK_PIPELINE_BIND_POINT_GRAPHICS
  - VK_PIPELINE_BIND_POINT_COMPUTE
  - VK_PIPELINE_BIND_POINT_RAY_TRACING_KHR

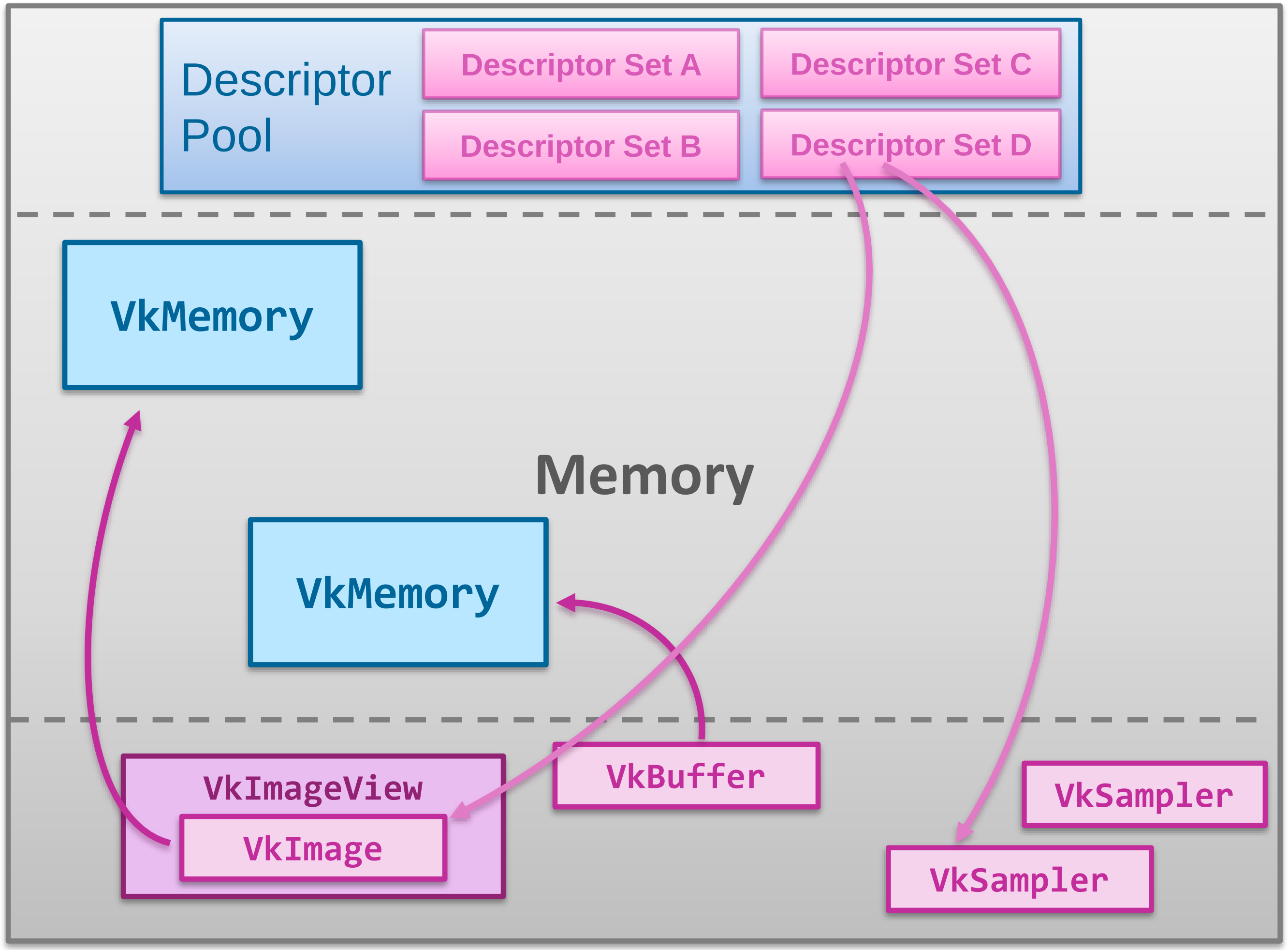


Descriptors and Descriptor Sets

- Attention: Different bind points! (see VkPipelineBindPoint)
  - VK_PIPELINE_BIND_POINT_GRAPHICS
  - VK_PIPELINE_BIND_POINT_COMPUTE
  - VK_PIPELINE_BIND_POINT_RAY_TRACING_KHR



Descriptor Pool, Allocation, and Update/Write



Descriptor Pool, Allocation, and Update/Write

- How to **allocate** descriptor sets:
  - 1. Create a pool of sufficient size (use multiple `VkDescriptorPoolSize`)
    - Use `vkCreateDescriptorPool` to actually create the pool on the GPU
  - 2. Create a `VkDescriptorSetLayout` **for each** descriptor set
    - Specify the resource bindings *within* the descriptor set using `VkDescriptorSetLayoutBinding` elements per resource
  - 3. Allocate a new set from the pool using `vkAllocateDescriptorSets`
    - The reference to the `VkDescriptorPool` is specified in the associated `VkDescriptorSetAllocateInfo` config struct.
- Bind all relevant `VkDescriptorSet` handles (from step 3.) for draw/compute/ray tracing via `vkCmdBindDescriptorSets`

Descriptor Pool, Allocation, and Update/Write

- How to get **usable descriptors onto the GPU**:
  - Allocating descriptor sets is *not* sufficient
  - They must still be filled with data (referencing the actual resources!)
    => **For each** resource *within* a descriptor set:
    - Create a `VkWriteDescriptorSet`
    - Specify the binding IDs within the descriptor set
    - Add the handles to buffers, buffer views, images, samplers, acceleration structures (via `pNext`)
    - If a binding refers to an array of descriptors, pass the count and multiple handles
  - Write to the GPU using `vkUpdateDescriptorSets`

Descriptor Types

- Different descriptor types (see VkDescriptorType)
  - VK_DESCRIPTOR_TYPE_SAMPLER (set only the VkSampler member of VkDescriptorImageInfo)
  - VK_DESCRIPTOR_TYPE_COMBINED_IMAGE_SAMPLER (set all members of VkDescriptorImageInfo)
  - VK_DESCRIPTOR_TYPE_SAMPLED_IMAGE (set VkImageView and VkImageLayout)
  - VK_DESCRIPTOR_TYPE_STORAGE_IMAGE (set VkImageView and VkImageLayout)
    (^ passing one or multiple VkDescriptorImageInfo handles to VkWriteDescriptorSet::pImageInfo ^)
  - VK_DESCRIPTOR_TYPE_UNIFORM_TEXEL_BUFFER (set VkWriteDescriptorSet::pTexelBufferView …)
  - VK_DESCRIPTOR_TYPE_STORAGE_TEXEL_BUFFER (… passing one multiple VkBufferView handles)
  - VK_DESCRIPTOR_TYPE_UNIFORM_BUFFER (set members of VkDescriptorBufferInfo accordingly, …)
  - VK_DESCRIPTOR_TYPE_STORAGE_BUFFER (...one or multiple via VkWriteDescriptorSet::pBufferInfo)
  - VK_DESCRIPTOR_TYPE_UNIFORM_BUFFER_DYNAMIC (^ Same as with uniform/storage buffers ^)
  - VK_DESCRIPTOR_TYPE_STORAGE_BUFFER_DYNAMIC (^ Same ^)
  - VK_DESCRIPTOR_TYPE_INPUT_ATTACHMENT (^ Same as with image-type descriptors at the top ^)
  - VK_DESCRIPTOR_TYPE_INLINE_UNIFORM_BLOCK_EXT (see dstArrayElement and descriptorCount)
  - VK_DESCRIPTOR_TYPE_ACCELERATION_STRUCTURE_KHR (VkWriteDescriptorSet::pNext chain contains pointer to VkWriteDescriptorSetAccelerationStructureKHR)

Descriptor Types and Usage in GLSL

- Different descriptor types (see VkDescriptorType)
  - VK_DESCRIPTOR_TYPE_SAMPLER (set only the VkSampler member of VkDescriptorImageInfo)
  - VK_DESCRIPTOR_TYPE_COMBINED_IMAGE_SAMPLER (set all members of VkDescriptorImageInfo)
  - VK_DESCRIPTOR_TYPE_SAMPLED_IMAGE (set VkImageView and VkImageLayout)
  - VK_DESCRIPTOR_TYPE_STORAGE_IMAGE (set VkImageView and VkImageLayout)
    (^ passing one or multiple VkDescriptorImageInfo handles to VkWriteDescriptorSet::pImageInfo ^)
  - VK_DESCRIPTOR_TYPE_UNI …)
  - VK_DESCRIPTOR_TYPE_STO
  - VK_DESCRIPTOR_TYPE_UNI .)
  - VK_DESCRIPTOR_TYPE_STO nfo)
  - VK_DESCRIPTOR_TYPE_UNI
  - VK_DESCRIPTOR_TYPE_STO
  - VK_DESCRIPTOR_TYPE_INPU
  - VK_DESCRIPTOR_TYPE_INLI)
  - VK_DESCRIPTOR_TYPE_ACCELERATION_STRUCTURE_KHR (VkWriteDescriptorSet::pNext chain contains pointer to VkWriteDescriptorSetAccelerationStructureKHR)

GLSL

```glsl
#version 450

layout (set = 0, binding = 0) uniform sampler s;

layout (set = 0, binding = 1) uniform texture2D sampledImage;

// ...

vec4 rgba = texture(sampler2D(sampledImage, s), vec2(0.5, 0.5));
```

Descriptor Types and Usage in GLSL

- Different descriptor types (see VkDescriptorType)
  - VK_DESCRIPTOR_TYPE_SAMPLER (set only the VkSampler member of VkDescriptorImageInfo)
  - VK_DESCRIPTOR_TYPE_COMBINED_IMAGE_SAMPLER (set all members of VkDescriptorImageInfo)
  - VK_DESCRIPTOR_TYPE_SAMPLED_IMAGE (set VkImageView and VkImageLayout)
  - VK_DESCRIPTOR_TYPE_STORAGE_IMAGE (set VkImageView and VkImageLayout)
    (^ passing one or multiple VkDescriptorImageInfo handles to VkWriteDescriptorSet::pImageInfo ^)
  - VK_DESCRIPTOR_TYPE_UN
  - VK_DESCRIPTOR_TYPE_STO
  - VK_DESCRIPTOR_TYPE_UN
  - VK_DESCRIPTOR_TYPE_STO
  - VK_DESCRIPTOR_TYPE_UN
  - VK_DESCRIPTOR_TYPE_STO
  - VK_DESCRIPTOR_TYPE_INPUT_ATTACHMENT (^ same as with image type descriptors at the top ^)
  - VK_DESCRIPTOR_TYPE_INLINE_UNIFORM_BLOCK_EXT (see dstArrayElement and descriptorCount)
  - VK_DESCRIPTOR_TYPE_ACCELERATION_STRUCTURE_KHR (VkWriteDescriptorSet::pNext chain contains pointer to VkWriteDescriptorSetAccelerationStructureKHR)

GLSL

```
#version 450

layout (set = 1, binding = 0) uniform sampler2D combinedImageSampler;

// ...

vec4 rgba = texture(combinedImageSampler, vec2(0.5, 0.5));
```

Descriptor Types and Usage in GLSL

- Different descriptor types (see VkDescriptorType)
  - VK_DESCRIPTOR_TYPE_SAMPLER (set only the VkSampler member of VkDescriptorImageInfo)
  - VK_DESCRIPTOR_TYPE_COMBINED_IMAGE_SAMPLER (set all members of VkDescriptorImageInfo)
  - VK_DESCRIPTOR_TYPE_SAMPLED_IMAGE (set VkImageView and VkImageLayout)
  - VK_DESCRIPTOR_TYPE_STORAGE_IMAGE (set VkImageView and VkImageLayout)
    (^ passing one or multiple VkDescriptorImageInfo handles to VkWriteDescriptorSet::pImageInfo ^)
  - VK_DESCRIPTOR_TYPE_UN
  - VK_DESCRIPTOR_TYPE_STO
  - VK_DESCRIPTOR_TYPE_UN
  - VK_DESCRIPTOR_TYPE_STO
  - VK_DESCRIPTOR_TYPE_UN
  - VK_DESCRIPTOR_TYPE_STO
  - VK_DESCRIPTOR_TYPE_INP
  - VK_DESCRIPTOR_TYPE_INLINE_UNIFORM_BLOCK_EXT (see dstArrayElement and descriptorCount)
  - VK_DESCRIPTOR_TYPE_ACCELERATION_STRUCTURE_KHR (VkWriteDescriptorSet::pNext chain contains pointer to VkWriteDescriptorSetAccelerationStructureKHR)

GLSL

```glsl
#version 450

layout (set = 2, binding = 0, rgba8) uniform image2D storageImage;

// ...

vec4 rgba = imageLoad(storageImage, ivec2(2, 2));

imageStore(storageImage, ivec2(2, 2), vec4(0.299, 0.587, 0.114, 1.0));
```

Descriptor Types and Usage in GLSL

- Different descriptor types (see VkDescriptorType)
  - VK_DESCRIPTOR_TYPE_SAMPLER (set only the VkSampler member of VkDescriptorImageInfo)
  - VK_DESCRIPTOR_TYPE_COMBINED_IMAGE_SAMPLER (set all members of VkDescriptorImageInfo)
  - VK_DESCRIPTOR_TYPE_SAMPLED_IMAGE (set VkImageView and VkImageLayout)
  - VK_DESCRIPTOR_TYPE_STORAGE_IMAGE (set VkImageView and VkImageLayout)
    (^ passing one or multiple VkDescriptorImageInfo handles to VkWriteDescriptorSet::pImageInfo ^)
  - VK_DESCRIPTOR_TYPE_UNIFORM_TEXEL_BUFFER (set VkWriteDescriptorSet::pTexelBufferView …)
  - VK_DESCRIPTOR_TYPE_
  - VK_DESCRIPTOR_TYPE_
  - VK_DESCRIPTOR_TYPE_)
  - VK_DESCRIPTOR_TYPE_
  - VK_DESCRIPTOR_TYPE_
  - VK_DESCRIPTOR_TYPE_
  - VK_DESCRIPTOR_TYPE_
  - VK_DESCRIPTOR_TYPE_ACCELERATION_STRUCTURE_KHR (VkWriteDescriptorSet::pNext chain
    contains pointer to VkWriteDescriptorSetAccelerationStructureKHR)

GLSL

```glsl
#version 450

layout (set = 3, binding = 0) uniform samplerBuffer uniformTexelBuffer;

// ...

int index = 0;
vec4 formattedValue = texelFetch(uniformTexelBuffer, index);
```

Descriptor Types and Usage in GLSL

- Different descriptor types (see VkDescriptorType)
  - VK_DESCRIPTOR_TYPE_SAMPLER (set only the VkSampler member of VkDescriptorImageInfo)
  - VK_DESCRIPTOR_TYPE_COMBINED_IMAGE_SAMPLER (set all members of VkDescriptorImageInfo)
  - VK_DESCRIPTOR_TYPE_SAMPLED_IMAGE (set VkImageView and VkImageLayout)
  - VK_DESCRIPTOR_TYPE_STORAGE_IMAGE (set VkImageView and VkImageLayout)
    (^ passing one or multiple VkDescriptorImageInfo handles to VkWriteDescriptorSet::pImageInfo ^)
  - VK_DESCRIPTOR_TYPE_UNIFORM_TEXEL_BUFFER (set VkWriteDescriptorSet::pTexelBufferView …)
  - VK_DESCRIPTOR_TYPE_STORAGE_TEXEL_BUFFER (… passing one multiple VkBufferView handles)
  - VK_DESCRIPTO
  - VK_DESCRIPTO o)
  - VK_DESCRIPTO
  - VK_DESCRIPTO
  - VK_DESCRIPTO
  - VK_DESCRIPTO
  - VK_DESCRIPTO

GLSL

```glsl
#version 450

layout (set = 4, binding = 0, rgba32f) uniform imageBuffer storageTexelBuffer;

// ...

int index = 0;
vec4 formattedValue = imageLoad(storageTexelBuffer, index);
imageStore(storageTexelBuffer, index, vec4(1.0, 2.0, 3.0, 4.0));
```

GLSL

```glsl
#version 450

layout (set = 5, binding = 0) uniform UniformBuffer
{
    mat4 projection;
    mat4 view;
    mat4 model;
} uniformBuffer;

// ...

mat4 M = uniformBuffer.projection * uniformBuffer.view * uniformBuffer.model;
```

- Different descrip
  - VK_DESCRIPTO
  - VK_DESCRIPTO
  - VK_DESCRIPTO
  - VK_DESCRIPTO
    (^ passing one ^)
  - VK_DESCRIPTO ..)
  - VK_DESCRIPTO
  - VK_DESCRIPTOR_TYPE_UNIFORM_BUFFER (set members of VkDescriptorBufferInfo accordingly, …)
  - VK_DESCRIPTOR_TYPE_STORAGE_BUFFER (…one or multiple via VkWriteDescriptorSet::pBufferInfo)
  - VK_DESCRIPTOR_TYPE_UNIFORM_BUFFER_DYNAMIC (^ Same as with uniform/storage buffers ^)
  - VK_DESCRIPTOR_TYPE_STORAGE_BUFFER_DYNAMIC (^ Same ^)
  - VK_DESCRIPTOR_TYPE_INPUT_ATTACHMENT (^ Same as with image-type descriptors at the top ^)
  - VK_DESCRIPTOR_TYPE_INLINE_UNIFORM_BLOCK_EXT (see dstArrayElement and descriptorCount)
  - VK_DESCRIPTOR_TYPE_ACCELERATION_STRUCTURE_KHR (VkWriteDescriptorSet::pNext chain contains pointer to VkWriteDescriptorSetAccelerationStructureKHR)

Descriptor Types and Usage in GLSL

- Different descriptor types (see VkDescriptor
  - VK_DESCRIPTOR_TYPE_SAMPLER (set only the V
  - VK_DESCRIPTOR_TYPE_COMBINED_IMAGE_SA
  - VK_DESCRIPTOR_TYPE_SAMPLED_IMAGE (set V
  - VK_DESCRIPTOR_TYPE_STORAGE_IMAGE (set V
    (^ passing one or multiple VkDescriptorImage
  - VK_DESCRIPTOR_TYPE_UNIFORM_TEXEL_BUFF
  - VK_DESCRIPTOR_TYPE_STORAGE_TEXEL_BUFF
  - VK_DESCRIPTOR_TYPE_UNIFORM_BUFFER (set
  - VK_DESCRIPTOR_TYPE_STORAGE_BUFFER (...or
  - VK_DESCRIPTOR_TYPE_UNIFORM_BUFFER_DY
  - VK_DESCRIPTOR_TYPE_STORAGE_BUFFER_DYNAMIC (^ Same ^)
  - VK_DESCRIPTOR_TYPE_INPUT_ATTACHMENT (^ Same as with image-type descriptors at the top ^)
  - VK_DESCRIPTOR_TYPE_INLINE_UNIFORM_BLOCK_EXT (see dstArrayElement and descriptorCount)
  - VK_DESCRIPTOR_TYPE_ACCELERATION_STRUCTURE_KHR (VkWriteDescriptorSet::pNext chain contains pointer to VkWriteDescriptorSetAccelerationStructureKHR)

GLSL

```glsl
#version 450

struct Particle {
    vec4 position;
    vec4 velocity;
};

layout (set = 5, binding = 1) buffer StorageBuffer {
    Particle particles[];
} storageBuffer;

// ...

int i = 0;
vec3 p = storageBuffer.particles[i].position.xyz;
```

Descriptor Types and Usage in GLSL

- Different descriptor types (see VkDescri
  - VK_DESCRIPTOR_TYPE_SAMPLER (set only
  - VK_DESCRIPTOR_TYPE_COMBINED_IMAGE
  - VK_DESCRIPTOR_TYPE_SAMPLED_IMAGE (
  - VK_DESCRIPTOR_TYPE_STORAGE_IMAGE (
    (^ passing one or multiple VkDescriptorIr
  - VK_DESCRIPTOR_TYPE_UNIFORM_TEXEL_I
  - VK_DESCRIPTOR_TYPE_STORAGE_TEXEL_E
  - VK_DESCRIPTOR_TYPE_UNIFORM_BUFFER
  - VK_DESCRIPTOR_TYPE_STORAGE_BUFFER
  - VK_DESCRIPTOR_TYPE_UNIFORM_BUFFER
  - VK_DESCRIPTOR_TYPE_STORAGE_BUFFER
  - VK_DESCRIPTOR_TYPE_INPUT_ATTACHME
  - VK_DESCRIPTOR_TYPE_INLINE_UNIFORM_
  - VK_DESCRIPTOR_TYPE_ACCELERATION_ST
    contains poi

GLSL

```
#version 450

layout (set = 5, binding = 0) uniform UniformBuffer {
    float deltaTime;
} uniformBuffer;

struct Particle {
    vec4 position;
    vec4 velocity;
};

layout (set = 5, binding = 1) buffer StorageBuffer {
    Particle particles[];
} storageBuffer;

// ...

int i = 0;
vec3 p = storageBuffer.particles[i].position.xyz;
vec3 v = storageBuffer.particles[i].velocity.xyz;
float dt = uniformBuffer.deltaTime;
storageBuffer.particles[i].position.xyz = p + p * v * dt;
```

Descriptor Types and Usage in GLSL

- Different descriptor types (see VkDescriptorType)
  - VK_DESCRIPTOR_TYPE_SAMPLER (set only the VkSampler member of VkDescriptorImageInfo)
  - VK_DESCRIPTOR_TYPE_COMBINED_IMAGE_SAMPLER (set all members of VkDescriptorImageInfo)
  - VK_DESCRIPTOR_TYPE_SAMPLED_IMAGE (set VkImageView and VkImageLayout)
  - VK_DESCRIPTOR_TYPE_STORAGE_IMAGE (set VkImageView and VkImageLayout) (^ pas
  - VK_D
  - VK_D
  - VK_D
  - VK_D
  - VK_D
  - VK_DESCRIPTOR_TYPE_STORAGE_BUFFER_DYNAMIC (^ Same ^)
  - VK_DESCRIPTOR_TYPE_INPUT_ATTACHMENT (^ Same as with image-type descriptors at the top ^)
  - VK_DESCRIPTOR_TYPE_INLINE_UNIFORM_BLOCK_EXT (see dstArrayElement and descriptorCount)
  - VK_DESCRIPTOR_TYPE_ACCELERATION_STRUCTURE_KHR (VkWriteDescriptorSet::pNext chain contains pointer to VkWriteDescriptorSetAccelerationStructureKHR)

GLSL

```glsl
#version 450

layout (input_attachment_index = 0, set = 6, binding = 0) uniform subpassInput depthImage;

// ...

float depthAtCurrentFramebufferLocation = subpassLoad(depthImage).r;
```

Descriptor Types and Usage in GLSL

GLSL

```
#version 450
#extension GL_EXT_ray_tracing : require
#extension GL_EXT_ray_query : require

layout(set = 7, binding = 0) uniform accelerationStructureEXT topLevelAS;

// ...

vec3 orig = vec3(0.0, 0.0, 0.0);
vec3 dir  = vec3(0.0, 0.0, 1.0);
float tMin = 0.01;
float tMax = 1000.0;

traceRayEXT(topLevelAS, gl_RayFlagsNoneEXT, 0xFF, 0, 0, 0, origin, tMin, dir, tMax, 0);

// ...

rayQueryEXT rayQuery;
rayQueryInitializeEXT(rayQuery, topLevelAS, gl_RayFlagsNoneEXT, 0xFF, origin, tMin, dir, tMax);
```

- VK_DESCRIPTOR_TYPE_ACCELERATION_STRUCTURE_KHR (`VkWriteDescriptorSet::pNext` chain contains pointer to `VkWriteDescriptorSetAccelerationStructureKHR`)

Introduction to Computer Graphics

186.832, 2021W, 3.0 ECTS

Thank you for your attention!

Johannes Unterguggenberger

Institute of Visual Computing & Human-Centered Technology

TU Wien, Austria