Introduction to Computer Graphics

186.832, 2021W, 3.0 ECTS

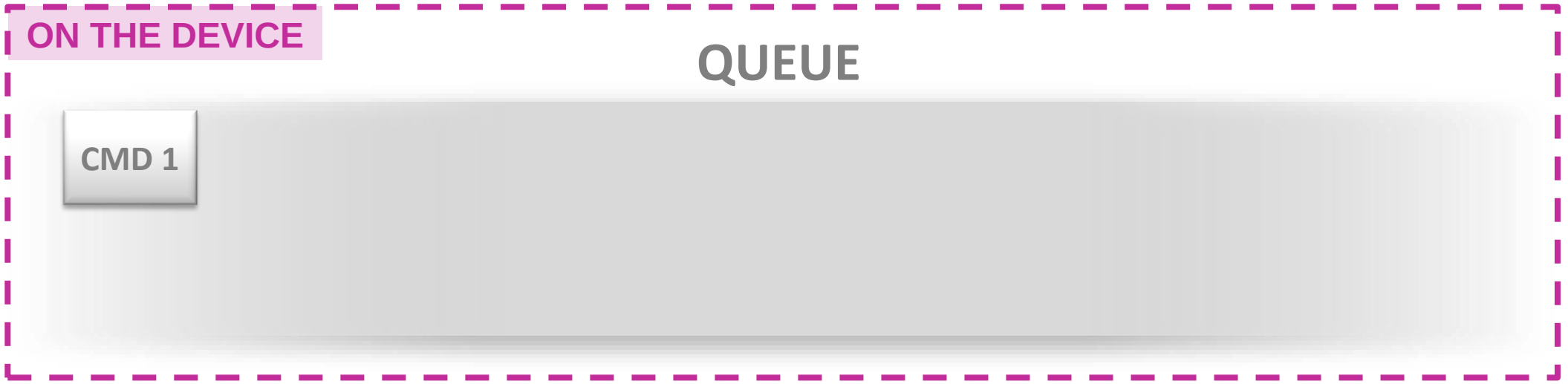
Vulkan Lecture Series, Episode 4:

Commands and Command Buffers

Johannes Unterguggenberger

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

Commands



Command Types

Command Buffer Recording

Command Buffer Lifecycle and Types

Providing Data to Commands

Different Types of Commands

Graphics Pipeline Commands:

vkCmdDraw
vkCmdDrawIndexed
vkCmdDrawIndirect
vkCmdDrawIndirectCount
vkCmdDrawIndexedIndirect
vkCmdDrawIndexedIndirectCount

vkCmdDrawMeshTasksNV
vkCmdDrawMeshTasksIndirectNV
vkCmdDrawMeshTasksIndirectCountNV

vkCmdClearAttachments

Compute Pipeline Commands:

vkCmdDispatch
vkCmdDispatchBase
vkCmdDispatchIndirect

Ray Tracing Pipeline Commands:

vkCmdTraceRaysKHR
vkCmdTraceRaysIndirectKHR

Transfer Commands:

vkCmdCopyBuffer
vkCmdCopyImage
vkCmdCopyBufferToImage
vkCmdCopyImageToBuffer
vkCmdCopyAccelerationStructureKHR
vkCmdCopyAccelerationStructureToMemoryKHR
vkCmdCopyMemoryToAccelerationStructureKHR
vkCmdFillBuffer

vkCmdBlitImage

vkCmdResolveImage

vkCmdClearColorImage
vkCmdClearDepthStencilImage

ACTION-Type

Ray-Tracing Acceleration Structure Build Commands:

vkCmdBuildAccelerationStructuresKHR
vkCmdBuildAccelerationStructuresIndirectKHR

Bind Commands:

vkCmdBindDescriptorSets
vkCmdBindPipeline
vkCmdBindVertexBuffers
vkCmdBindIndexBuffer

Other Commands:

vkCmdPushConstants
vkCmdPushDescriptorSetKHR
vkCmdSetScissor
vkCmdSetViewport
vkCmdSetDepthBias
...

Different Types of Commands

Graphics Pipeline Commands:

vkCmdDraw
vkCmdDrawIndexed
vkCmdDrawIndirect
vkCmdDrawIndirectCount
vkCmdDrawIndexedIndirect
vkCmdDrawIndexedIndirectCount

vkCmdDrawMeshTasksNV
vkCmdDrawMeshTasksIndirectNV
vkCmdDrawMeshTasksIndirectCountNV

vkCmdClearAttachments

Compute Pipeline Commands:

vkCmdDispatch
vkCmdDispatchBase
vkCmdDispatchIndirect

Ray Tracing Pipeline Commands:

vkCmdTraceRaysKHR
vkCmdTraceRaysIndirectKHR

Transfer Commands:

vkCmdCopyBuffer
vkCmdCopyImage
vkCmdCopyBufferToImage
vkCmdCopyImageToBuffer
vkCmdCopyAccelerationStructureKHR
vkCmdCopyAccelerationStructureToMemoryKHR
vkCmdCopyMemoryToAccelerationStructureKHR
vkCmdFillBuffer

vkCmdBlitImage

vkCmdResolveImage

vkCmdClearColorImage
vkCmdClearDepthStencilImage

ACTION-Type

Ray-Tracing Acceleration Structure Build Commands:

vkCmdBuildAccelerationStructuresKHR
vkCmdBuildAccelerationStructuresIndirectKHR

Bind Commands:

vkCmdBindDescriptorSets
vkCmdBindPipeline
vkCmdBindVertexBuffers
vkCmdBindIndexBuffer

Other Commands:

vkCmdPushConstants
vkCmdPushDescriptorSetKHR
vkCmdSetScissor
vkCmdSetViewport
vkCmdSetDepthBias
...

Different Types of Commands

Graphics Pipeline Commands:

vkCmdDraw
vkCmdDrawIndexed
vkCmdDrawIndirect
vkCmdDrawIndirectCount
vkCmdDrawIndexedIndirect
vkCmdDrawIndexedIndirectCount

vkCmdDrawMeshTasksNV
vkCmdDrawMeshTasksIndirectNV
vkCmdDrawMeshTasksIndirectCountNV

vkCmdClearAttachments

Compute Pipeline Commands:

vkCmdDispatch
vkCmdDispatchBase
vkCmdDispatchIndirect

Ray Tracing Pipeline Commands:

vkCmdTraceRaysKHR
vkCmdTraceRaysIndirectKHR

Transfer Commands:

vkCmdCopyBuffer
vkCmdCopyImage
vkCmdCopyBufferToImage
vkCmdCopyImageToBuffer
vkCmdCopyAccelerationStructureKHR
vkCmdCopyAccelerationStructureToMemoryKHR
vkCmdCopyMemoryToAccelerationStructureKHR
vkCmdFillBuffer

vkCmdBlitImage

vkCmdResolveImage

vkCmdClearColorImage
vkCmdClearDepthStencilImage

ACTION-Type

Ray-Tracing Acceleration Structure Build Commands:

vkCmdBuildAccelerationStructuresKHR
vkCmdBuildAccelerationStructuresIndirectKHR

Bind Commands:

vkCmdBindDescriptorSets
vkCmdBindPipeline
vkCmdBindVertexBuffers
vkCmdBindIndexBuffer

Other Commands:

vkCmdPushConstants
vkCmdPushDescriptorSetKHR
vkCmdSetScissor
vkCmdSetViewport
vkCmdSetDepthBias
...

Different Types of Commands

Graphics Pipeline Commands:

vkCmdDraw
vkCmdDrawIndexed
vkCmdDrawIndirect
vkCmdDrawIndirectCount
vkCmdDrawIndexedIndirect
vkCmdDrawIndexedIndirectCount

vkCmdDrawMeshTasksNV
vkCmdDrawMeshTasksIndirectNV
vkCmdDrawMeshTasksIndirectCountNV

vkCmdClearAttachments

Compute Pipeline Commands:

vkCmdDispatch
vkCmdDispatchBase
vkCmdDispatchIndirect

Ray Tracing Pipeline Commands:

vkCmdTraceRaysKHR
vkCmdTraceRaysIndirectKHR

Transfer Commands:

vkCmdCopyBuffer
vkCmdCopyImage
vkCmdCopyBufferToImage
vkCmdCopyImageToBuffer
vkCmdCopyAccelerationStructureKHR
vkCmdCopyAccelerationStructureToMemoryKHR
vkCmdCopyMemoryToAccelerationStructureKHR
vkCmdFillBuffer

vkCmdBlitImage

vkCmdResolveImage

vkCmdClearColorImage
vkCmdClearDepthStencilImage

ACTION-Type

Ray-Tracing Acceleration Structure Build Commands:

vkCmdBuildAccelerationStructuresKHR
vkCmdBuildAccelerationStructuresIndirectKHR

Bind Commands:

vkCmdBindDescriptorSets
vkCmdBindPipeline
vkCmdBindVertexBuffers
vkCmdBindIndexBuffer

Other Commands:

vkCmdPushConstants
vkCmdPushDescriptorSetKHR
vkCmdSetScissor
vkCmdSetViewport
vkCmdSetDepthBias
...

Different Types of Commands

Graphics Pipeline Commands:

vkCmdDraw
vkCmdDrawIndexed
vkCmdDrawIndirect
vkCmdDrawIndirectCount
vkCmdDrawIndexedIndirect
vkCmdDrawIndexedIndirectCount

vkCmdDrawMeshTasksNV
vkCmdDrawMeshTasksIndirectNV
vkCmdDrawMeshTasksIndirectCountNV

vkCmdClearAttachments

Compute Pipeline Commands:

vkCmdDispatch
vkCmdDispatchBase
vkCmdDispatchIndirect

Ray Tracing Pipeline Commands:

vkCmdTraceRaysKHR
vkCmdTraceRaysIndirectKHR

Transfer Commands:

vkCmdCopyBuffer
vkCmdCopyImage
vkCmdCopyBufferToImage
vkCmdCopyImageToBuffer
vkCmdCopyAccelerationStructureKHR
vkCmdCopyAccelerationStructureToMemoryKHR
vkCmdCopyMemoryToAccelerationStructureKHR
vkCmdFillBuffer

vkCmdBlitImage

vkCmdResolveImage

vkCmdClearColorImage
vkCmdClearDepthStencilImage

ACTION-Type

Ray-Tracing Acceleration Structure Build Commands:

vkCmdBuildAccelerationStructuresKHR
vkCmdBuildAccelerationStructuresIndirectKHR

Bind Commands:

vkCmdBindDescriptorSets
vkCmdBindPipeline
vkCmdBindVertexBuffers
vkCmdBindIndexBuffer

Other Commands:

vkCmdPushConstants
vkCmdPushDescriptorSetKHR
vkCmdSetScissor
vkCmdSetViewport
vkCmdSetDepthBias
...

Different Types of Commands

Graphics Pipeline Commands:

vkCmdDraw
vkCmdDrawIndexed
vkCmdDrawIndirect
vkCmdDrawIndirectCount
vkCmdDrawIndexedIndirect
vkCmdDrawIndexedIndirectCount

vkCmdDrawMeshTasksNV
vkCmdDrawMeshTasksIndirectNV
vkCmdDrawMeshTasksIndirectCountNV

vkCmdClearAttachments

Compute Pipeline Commands:

vkCmdDispatch
vkCmdDispatchBase
vkCmdDispatchIndirect

Ray Tracing Pipeline Commands:

vkCmdTraceRaysKHR
vkCmdTraceRaysIndirectKHR

Transfer Commands:

vkCmdCopyBuffer
vkCmdCopyImage
vkCmdCopyBufferToImage
vkCmdCopyImageToBuffer
vkCmdCopyAccelerationStructureKHR
vkCmdCopyAccelerationStructureToMemoryKHR
vkCmdCopyMemoryToAccelerationStructureKHR
vkCmdFillBuffer

vkCmdBlitImage

vkCmdResolveImage

vkCmdClearColorImage
vkCmdClearDepthStencilImage

ACTION-Type

Ray-Tracing Acceleration Structure Build Commands:

vkCmdBuildAccelerationStructuresKHR
vkCmdBuildAccelerationStructuresIndirectKHR

Bind Commands:

vkCmdBindDescriptorSets
vkCmdBindPipeline
vkCmdBindVertexBuffers
vkCmdBindIndexBuffer

Other Commands:

vkCmdPushConstants
vkCmdPushDescriptorSetKHR
vkCmdSetScissor
vkCmdSetViewport
vkCmdSetDepthBias
...

Different Types of Commands

Graphics Pipeline Commands:

vkCmdDraw
vkCmdDrawIndexed
vkCmdDrawIndirect
vkCmdDrawIndirectCount
vkCmdDrawIndexedIndirect
vkCmdDrawIndexedIndirectCount

vkCmdDrawMeshTasksNV
vkCmdDrawMeshTasksIndirectNV
vkCmdDrawMeshTasksIndirectCountNV

vkCmdClearAttachments

Compute Pipeline Commands:

vkCmdDispatch
vkCmdDispatchBase
vkCmdDispatchIndirect

Ray Tracing Pipeline Commands:

vkCmdTraceRaysKHR
vkCmdTraceRaysIndirectKHR

Transfer Commands:

vkCmdCopyBuffer
vkCmdCopyImage
vkCmdCopyBufferToImage
vkCmdCopyImageToBuffer
vkCmdCopyAccelerationStructureKHR
vkCmdCopyAccelerationStructureToMemoryKHR
vkCmdCopyMemoryToAccelerationStructureKHR
vkCmdFillBuffer

vkCmdBlitImage

vkCmdResolveImage

vkCmdClearColorImage
vkCmdClearDepthStencilImage

ACTION-Type

Ray-Tracing Acceleration Structure Build Commands:

vkCmdBuildAccelerationStructuresKHR
vkCmdBuildAccelerationStructuresIndirectKHR

Bind Commands:

vkCmdBindDescriptorSets
vkCmdBindPipeline
vkCmdBindVertexBuffers
vkCmdBindIndexBuffer

Other Commands:

vkCmdPushConstants
vkCmdPushDescriptorSetKHR
vkCmdSetScissor
vkCmdSetViewport
vkCmdSetDepthBias
...

Different Types of Commands

Graphics Pipeline Commands:

vkCmdDraw
vkCmdDrawIndexed
vkCmdDrawIndirect
vkCmdDrawIndirectCount
vkCmdDrawIndexedIndirect
vkCmdDrawIndexedIndirectCount

vkCmdDrawMeshTasksNV
vkCmdDrawMeshTasksIndirectNV
vkCmdDrawMeshTasksIndirectCountNV

vkCmdClearAttachments

Compute Pipeline Commands:

vkCmdDispatch
vkCmdDispatchBase
vkCmdDispatchIndirect

Ray Tracing Pipeline Commands:

vkCmdTraceRaysKHR
vkCmdTraceRaysIndirectKHR

Transfer Commands:

vkCmdCopyBuffer
vkCmdCopyImage
vkCmdCopyBufferToImage
vkCmdCopyImageToBuffer
vkCmdCopyAccelerationStructureKHR
vkCmdCopyAccelerationStructureToMemoryKHR
vkCmdCopyMemoryToAccelerationStructureKHR
vkCmdFillBuffer

vkCmdBlitImage

vkCmdResolveImage

vkCmdClearColorImage
vkCmdClearDepthStencilImage

ACTION-Type

Ray-Tracing Acceleration Structure Build Commands:

vkCmdBuildAccelerationStructuresKHR
vkCmdBuildAccelerationStructuresIndirectKHR

Bind Commands:

vkCmdBindDescriptorSets
vkCmdBindPipeline
vkCmdBindVertexBuffers
vkCmdBindIndexBuffer

Other Commands:

vkCmdPushConstants
vkCmdPushDescriptorSetKHR
vkCmdSetScissor
vkCmdSetViewport
vkCmdSetDepthBias
...

Different Types of Commands

Graphics Pipeline Commands:

vkCmdDraw
vkCmdDrawIndexed
vkCmdDrawIndirect
vkCmdDrawIndirectCount
vkCmdDrawIndexedIndirect
vkCmdDrawIndexedIndirectCount

vkCmdDrawMeshTasksNV
vkCmdDrawMeshTasksIndirectNV
vkCmdDrawMeshTasksIndirectCountNV

vkCmdClearAttachments

Compute Pipeline Commands:

vkCmdDispatch
vkCmdDispatchBase
vkCmdDispatchIndirect

Ray Tracing Pipeline Commands:

vkCmdTraceRaysKHR
vkCmdTraceRaysIndirectKHR

Transfer Commands:

vkCmdCopyBuffer
vkCmdCopyImage
vkCmdCopyBufferToImage
vkCmdCopyImageToBuffer
vkCmdCopyAccelerationStructureKHR
vkCmdCopyAccelerationStructureToMemoryKHR
vkCmdCopyMemoryToAccelerationStructureKHR
vkCmdFillBuffer

vkCmdBlitImage

vkCmdResolveImage

vkCmdClearColorImage
vkCmdClearDepthStencilImage

ACTION-Type

Ray-Tracing Acceleration Structure Build Commands:

vkCmdBuildAccelerationStructuresKHR
vkCmdBuildAccelerationStructuresIndirectKHR

Bind Commands:

vkCmdBindDescriptorSets
vkCmdBindPipeline
vkCmdBindVertexBuffers
vkCmdBindIndexBuffer

Other Commands:

vkCmdPushConstants
vkCmdPushDescriptorSetKHR
vkCmdSetScissor
vkCmdSetViewport
vkCmdSetDepthBias
...

Different Types of Commands

Graphics Pipeline Commands:

vkCmdDraw
vkCmdDrawIndexed
vkCmdDrawIndirect
vkCmdDrawIndirectCount
vkCmdDrawIndexedIndirect
vkCmdDrawIndexedIndirectCount

vkCmdDrawMeshTasksNV
vkCmdDrawMeshTasksIndirectNV
vkCmdDrawMeshTasksIndirectCountNV

vkCmdClearAttachments

Compute Pipeline Commands:

vkCmdDispatch
vkCmdDispatchBase
vkCmdDispatchIndirect

Ray Tracing Pipeline Commands:

vkCmdTraceRaysKHR
vkCmdTraceRaysIndirectKHR

Transfer Commands:

vkCmdCopyBuffer
vkCmdCopyImage
vkCmdCopyBufferToImage
vkCmdCopyImageToBuffer
vkCmdCopyAccelerationStructureKHR
vkCmdCopyAccelerationStructureToMemoryKHR
vkCmdCopyMemoryToAccelerationStructureKHR
vkCmdFillBuffer

vkCmdBlitImage

vkCmdResolveImage

vkCmdClearColorImage
vkCmdClearDepthStencilImage

ACTION-Type

Ray-Tracing Acceleration Structure Build Commands:

vkCmdBuildAccelerationStructuresKHR
vkCmdBuildAccelerationStructuresIndirectKHR

Bind Commands:

vkCmdBindDescriptorSets
vkCmdBindPipeline
vkCmdBindVertexBuffers
vkCmdBindIndexBuffer

Other Commands:

vkCmdPushConstants
vkCmdPushDescriptorSetKHR
vkCmdSetScissor
vkCmdSetViewport
vkCmdSetDepthBias
...

Different Types of Commands

Graphics Pipeline Commands:

vkCmdDraw
vkCmdDrawIndexed
vkCmdDrawIndirect
vkCmdDrawIndirectCount
vkCmdDrawIndexedIndirect
vkCmdDrawIndexedIndirectCount

vkCmdDrawMeshTasksNV
vkCmdDrawMeshTasksIndirectNV
vkCmdDrawMeshTasksIndirectCountNV

vkCmdClearAttachments

Compute Pipeline Commands:

vkCmdDispatch
vkCmdDispatchBase
vkCmdDispatchIndirect

Ray Tracing Pipeline Commands:

vkCmdTraceRaysKHR
vkCmdTraceRaysIndirectKHR

Transfer Commands:

vkCmdCopyBuffer
vkCmdCopyImage
vkCmdCopyBufferToImage
vkCmdCopyImageToBuffer
vkCmdCopyAccelerationStructureKHR
vkCmdCopyAccelerationStructureToMemoryKHR
vkCmdCopyMemoryToAccelerationStructureKHR
vkCmdFillBuffer

vkCmdBlitImage

vkCmdResolveImage

vkCmdClearColorImage
vkCmdClearDepthStencilImage

ACTION-Type

Ray-Tracing Acceleration Structure Build Commands:

vkCmdBuildAccelerationStructuresKHR
vkCmdBuildAccelerationStructuresIndirectKHR

Bind Commands:

vkCmdBindDescriptorSets
vkCmdBindPipeline
vkCmdBindVertexBuffers
vkCmdBindIndexBuffer

Other Commands:

vkCmdPushConstants
vkCmdPushDescriptorSetKHR
vkCmdSetScissor
vkCmdSetViewport
vkCmdSetDepthBias
...

Different Types of Commands

Graphics Pipeline Commands:

vkCmdDraw
vkCmdDrawIndexed
vkCmdDrawIndirect
vkCmdDrawIndirectCount
vkCmdDrawIndexedIndirect
vkCmdDrawIndexedIndirectCount

vkCmdDrawMeshTasksNV
vkCmdDrawMeshTasksIndirectNV
vkCmdDrawMeshTasksIndirectCountNV

vkCmdClearAttachments

Compute Pipeline Commands:

vkCmdDispatch
vkCmdDispatchBase
vkCmdDispatchIndirect

Ray Tracing Pipeline Commands:

vkCmdTraceRaysKHR
vkCmdTraceRaysIndirectKHR

Transfer Commands:

vkCmdCopyBuffer
vkCmdCopyImage
vkCmdCopyBufferToImage
vkCmdCopyImageToBuffer
vkCmdCopyAccelerationStructureKHR
vkCmdCopyAccelerationStructureToMemoryKHR
vkCmdCopyMemoryToAccelerationStructureKHR
vkCmdFillBuffer

vkCmdBlitImage

vkCmdResolveImage

vkCmdClearColorImage
vkCmdClearDepthStencilImage

ACTION-Type

Ray-Tracing Acceleration Structure Build Commands:

vkCmdBuildAccelerationStructuresKHR
vkCmdBuildAccelerationStructuresIndirectKHR

Bind Commands:

vkCmdBindDescriptorSets
vkCmdBindPipeline
vkCmdBindVertexBuffers
vkCmdBindIndexBuffer

Other Commands:

vkCmdPushConstants
vkCmdPushDescriptorSetKHR
vkCmdSetScissor
vkCmdSetViewport
vkCmdSetDepthBias
...

Different Types of Commands

Graphics Pipeline Commands:

vkCmdDraw
vkCmdDrawIndexed
vkCmdDrawIndirect
vkCmdDrawIndirectCount
vkCmdDrawIndexedIndirect
vkCmdDrawIndexedIndirectCount

vkCmdDrawMeshTasksNV
vkCmdDrawMeshTasksIndirectNV
vkCmdDrawMeshTasksIndirectCountNV

vkCmdClearAttachments

Compute Pipeline Commands:

vkCmdDispatch
vkCmdDispatchBase
vkCmdDispatchIndirect

Ray Tracing Pipeline Commands:

vkCmdTraceRaysKHR
vkCmdTraceRaysIndirectKHR

Transfer Commands:

vkCmdCopyBuffer
vkCmdCopyImage
vkCmdCopyBufferToImage
vkCmdCopyImageToBuffer
vkCmdCopyAccelerationStructureKHR
vkCmdCopyAccelerationStructureToMemoryKHR
vkCmdCopyMemoryToAccelerationStructureKHR
vkCmdFillBuffer

vkCmdBlitImage

vkCmdResolveImage

vkCmdClearColorImage
vkCmdClearDepthStencilImage

ACTION-Type

Ray-Tracing Acceleration Structure Build Commands:

vkCmdBuildAccelerationStructuresKHR
vkCmdBuildAccelerationStructuresIndirectKHR

Bind Commands:

vkCmdBindDescriptorSets
vkCmdBindPipeline
vkCmdBindVertexBuffers
vkCmdBindIndexBuffer

Other Commands:

vkCmdPushConstants
vkCmdPushDescriptorSetKHR
vkCmdSetScissor
vkCmdSetViewport
vkCmdSetDepthBias
...

Different Types of Commands

Graphics Pipeline Commands:

vkCmdDraw
vkCmdDrawIndexed
vkCmdDrawIndirect
vkCmdDrawIndirectCount
vkCmdDrawIndexedIndirect
vkCmdDrawIndexedIndirectCount

vkCmdDrawMeshTasksNV
vkCmdDrawMeshTasksIndirectNV
vkCmdDrawMeshTasksIndirectCountNV

vkCmdClearAttachments

Compute Pipeline Commands:

vkCmdDispatch
vkCmdDispatchBase
vkCmdDispatchIndirect

Ray Tracing Pipeline Commands:

vkCmdTraceRaysKHR
vkCmdTraceRaysIndirectKHR

Transfer Commands:

vkCmdCopyBuffer
vkCmdCopyImage
vkCmdCopyBufferToImage
vkCmdCopyImageToBuffer
vkCmdCopyAccelerationStructureKHR
vkCmdCopyAccelerationStructureToMemoryKHR
vkCmdCopyMemoryToAccelerationStructureKHR
vkCmdFillBuffer

vkCmdBlitImage

vkCmdResolveImage

vkCmdClearColorImage
vkCmdClearDepthStencilImage

ACTION-Type

Ray-Tracing Acceleration Structure Build Commands:

vkCmdBuildAccelerationStructuresKHR
vkCmdBuildAccelerationStructuresIndirectKHR

Bind Commands:

vkCmdBindDescriptorSets
vkCmdBindPipeline
vkCmdBindVertexBuffers
vkCmdBindIndexBuffer

Other Commands:

vkCmdPushConstants
vkCmdPushDescriptorSetKHR
vkCmdSetScissor
vkCmdSetViewport
vkCmdSetDepthBias
...

Different Types of Commands

Graphics Pipeline Commands:

vkCmdDraw
vkCmdDrawIndexed
vkCmdDrawIndirect
vkCmdDrawIndirectCount
vkCmdDrawIndexedIndirect
vkCmdDrawIndexedIndirectCount

vkCmdDrawMeshTasksNV
vkCmdDrawMeshTasksIndirectNV
vkCmdDrawMeshTasksIndirectCountNV

vkCmdClearAttachments

Compute Pipeline Commands:

vkCmdDispatch
vkCmdDispatchBase
vkCmdDispatchIndirect

Ray Tracing Pipeline Commands:

vkCmdTraceRaysKHR
vkCmdTraceRaysIndirectKHR

Transfer Commands:

vkCmdCopyBuffer
vkCmdCopyImage
vkCmdCopyBufferToImage
vkCmdCopyImageToBuffer
vkCmdCopyAccelerationStructureKHR
vkCmdCopyAccelerationStructureToMemoryKHR
vkCmdCopyMemoryToAccelerationStructureKHR
vkCmdFillBuffer

vkCmdBlitImage

vkCmdResolveImage

vkCmdClearColorImage
vkCmdClearDepthStencilImage

ACTION-Type

Ray-Tracing Acceleration Structure Build Commands:

vkCmdBuildAccelerationStructuresKHR
vkCmdBuildAccelerationStructuresIndirectKHR

Bind Commands:

vkCmdBindDescriptorSets
vkCmdBindPipeline
vkCmdBindVertexBuffers
vkCmdBindIndexBuffer

Other Commands:

vkCmdPushConstants
vkCmdPushDescriptorSetKHR
vkCmdSetScissor
vkCmdSetViewport
vkCmdSetDepthBias
...

Different Types of Commands

Graphics Pipeline Commands:

vkCmdDraw
vkCmdDrawIndexed
vkCmdDrawIndirect
vkCmdDrawIndirectCount
vkCmdDrawIndexedIndirec
vkCmdDrawIndexedIndirec

vkCmdDrawMeshTasksNV
vkCmdDrawMeshTasksIndir
vkCmdDrawMeshTasksIndir

vkCmdClearAttachments

Compute Pipeline Commands:

vkCmdDispatch
vkCmdDispatchBase
vkCmdDispatchIndirect
vkCmdTraceRaysIndirectKHR

> Unlike other clear commands, vkCmdClearAttachments executes as a drawing command, rather than a transfer command, […]
>
> *The Khronos Group. Vulkan 1.2.200 Specificaton*

Transfer Commands:

vkCmdCopyBuffer
vkCmdCopyImage
vkCmdCopyBufferToImage
vkCmdCopyImageToBuffer
dCopyAccelerationStructureKHR
dCopyAccelerationStructureToMemoryKHR
dCopyMemoryToAccelerationStructureKHR
dFillBuffer

dBlitImage

dResolveImage

vkCmdClearColorImage
vkCmdClearDepthStencilImage

ACTION-Type

Ray-Tracing Acceleration Structure Build Commands:

vkCmdBuildAccelerationStructuresKHR
vkCmdBuildAccelerationStructuresIndirectKHR

Bind Commands:

vkCmdBindDescriptorSets
vkCmdBindPipeline
vkCmdBindVertexBuffers
vkCmdBindIndexBuffer

Other Commands:

vkCmdPushConstants
vkCmdPushDescriptorSetKHR
vkCmdSetScissor
vkCmdSetViewport
vkCmdSetDepthBias
...

Different Types of Commands

Graphics Pipeline Commands:

```
vkCmdDraw
vkCmdDrawIndexed
vkCmdDrawIndirect
vkCmdDrawIndirectCount
vkCmdDrawIndexedIndirect
vkCmdDrawIndexedIndirectCount

vkCmdDrawMeshTasksNV
vkCmdDrawMeshTasksIndirectNV
vkCmdDrawMeshTasksIndirectCountNV

vkCmdClearAttachments
```

Compute Pipeline Commands:

```
vkCmdDispatch
vkCmdDispatchBase
vkCmdDispatchIndirect
```

Ray Tracing Pipeline Commands:

```
vkCmdTraceRaysKHR
vkCmdTraceRaysIndirectKHR
```

Transfer Commands:

```
vkCmdCopyBuffer
vkCmdCopyImage
vkCmdCopyBufferToImage
vkCmdCopyImageToBuffer
vkCmdCopyAccelerationStructureKHR
vkCmdCopyAccelerationStructureToMemoryKHR
vkCmdCopyMemoryToAccelerationStructureKHR
vkCmdFillBuffer

vkCmdBlitImage

vkCmdResolveImage

vkCmdClearColorImage
vkCmdClearDepthStencilImage
```

Ray-Tracing Acceleration Structure Build Commands:

```
vkCmdBuildAccelerationStructuresKHR
vkCmdBuildAccelerationStructuresIndirectKHR
```

STATE-Type

Bind Commands:

```
vkCmdBindDescriptorSets
vkCmdBindPipeline
vkCmdBindVertexBuffers
vkCmdBindIndexBuffer
```

Other Commands:

```
vkCmdPushConstants
vkCmdPushDescriptorSetKHR
vkCmdSetScissor
vkCmdSetViewport
vkCmdSetDepthBias
...
```

Different Types of Commands

Graphics Pipeline Commands:

vkCmdDraw
vkCmdDrawIndexed
vkCmdDrawIndirect
vkCmdDrawIndirectCount
vkCmdDrawIndexedIndirect
vkCmdDrawIndexedIndirectCount

vkCmdDrawMeshTasksNV
vkCmdDrawMeshTasksIndirectNV
vkCmdDrawMeshTasksIndirectCountNV

vkCmdClearAttachments

Compute Pipeline Commands:

vkCmdDispatch
vkCmdDispatchBase
vkCmdDispatchIndirect

Ray Tracing Pipeline Commands:

vkCmdTraceRaysKHR
vkCmdTraceRaysIndirectKHR

Transfer Commands:

vkCmdCopyBuffer
vkCmdCopyImage
vkCmdCopyBufferToImage
vkCmdCopyImageToBuffer
vkCmdCopyAccelerationStructureKHR
vkCmdCopyAccelerationStructureToMemoryKHR
vkCmdCopyMemoryToAccelerationStructureKHR
vkCmdFillBuffer

vkCmdBlitImage

vkCmdResolveImage

vkCmdClearColorImage
vkCmdClearDepthStencilImage

Ray-Tracing Acceleration Structure Build Commands:

vkCmdBuildAccelerationStructuresKHR
vkCmdBuildAccelerationStructuresIndirectKHR

STATE-Type

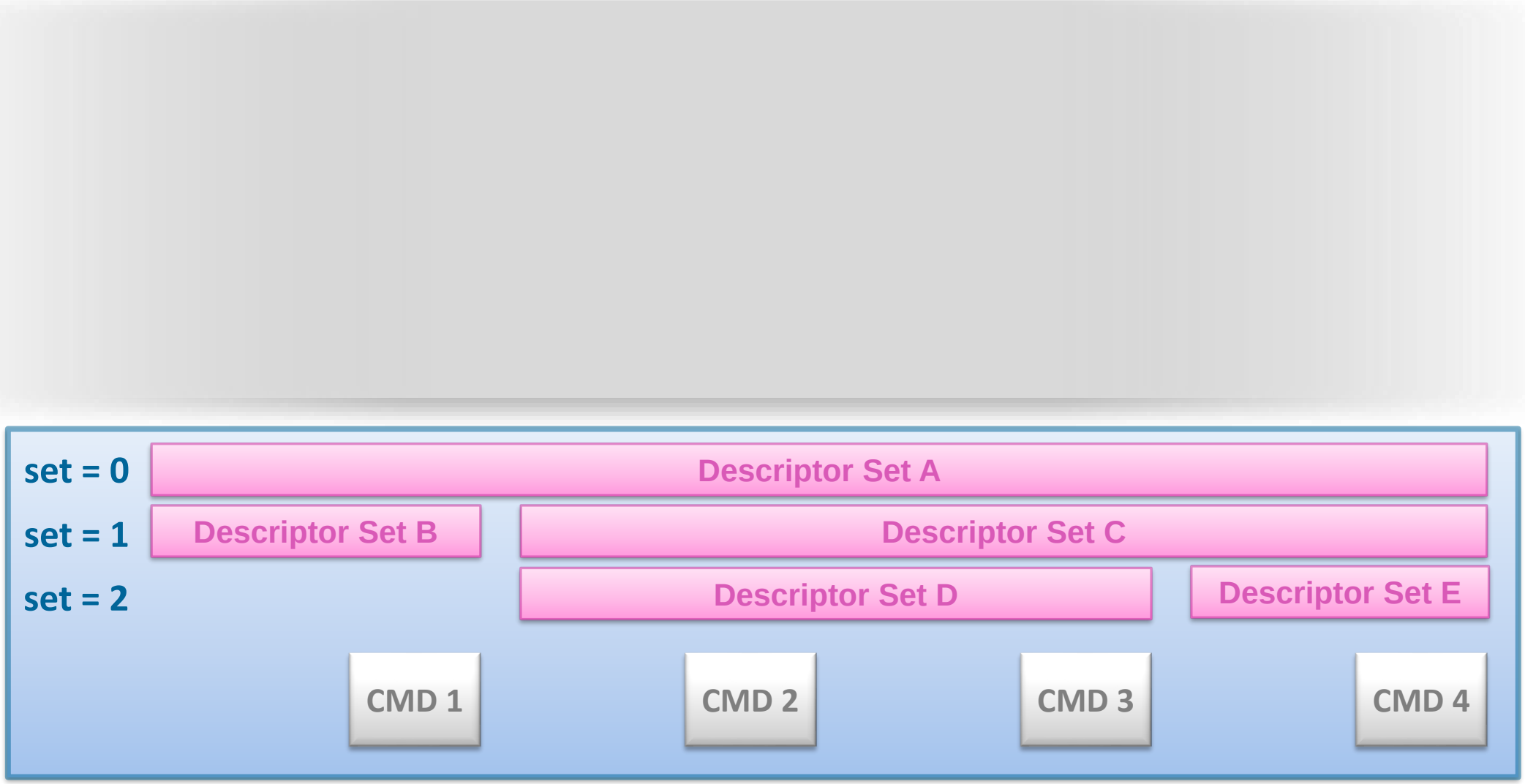
Bind Commands:

vkCmdBindDescriptorSets
vkCmdBindPipeline
vkCmdBindVertexBuffers
vkCmdBindIndexBuffer

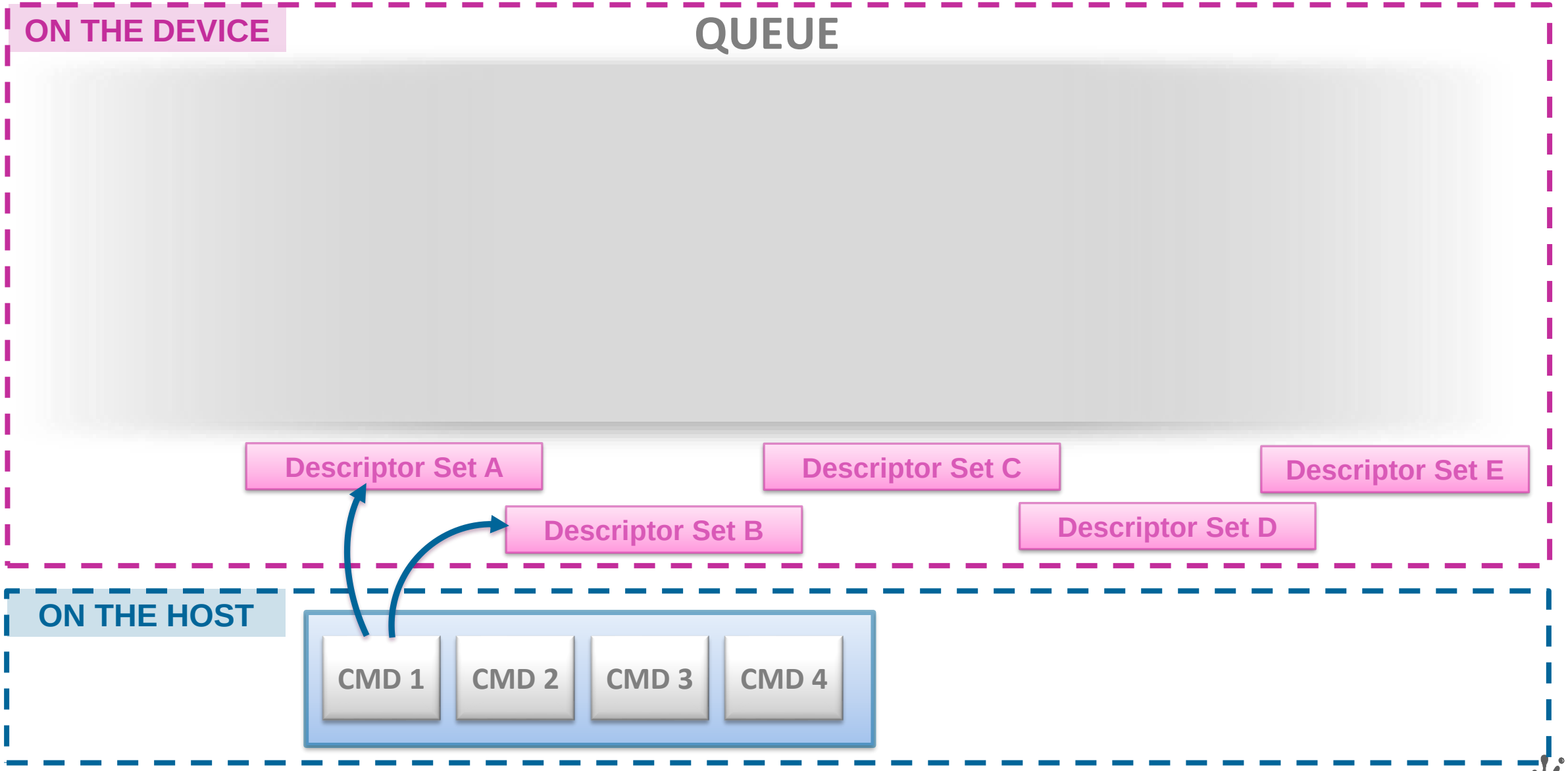
Other Commands:

vkCmdPushConstants
vkCmdPushDescriptorSetKHR
vkCmdSetScissor
vkCmdSetViewport
vkCmdSetDepthBias
...

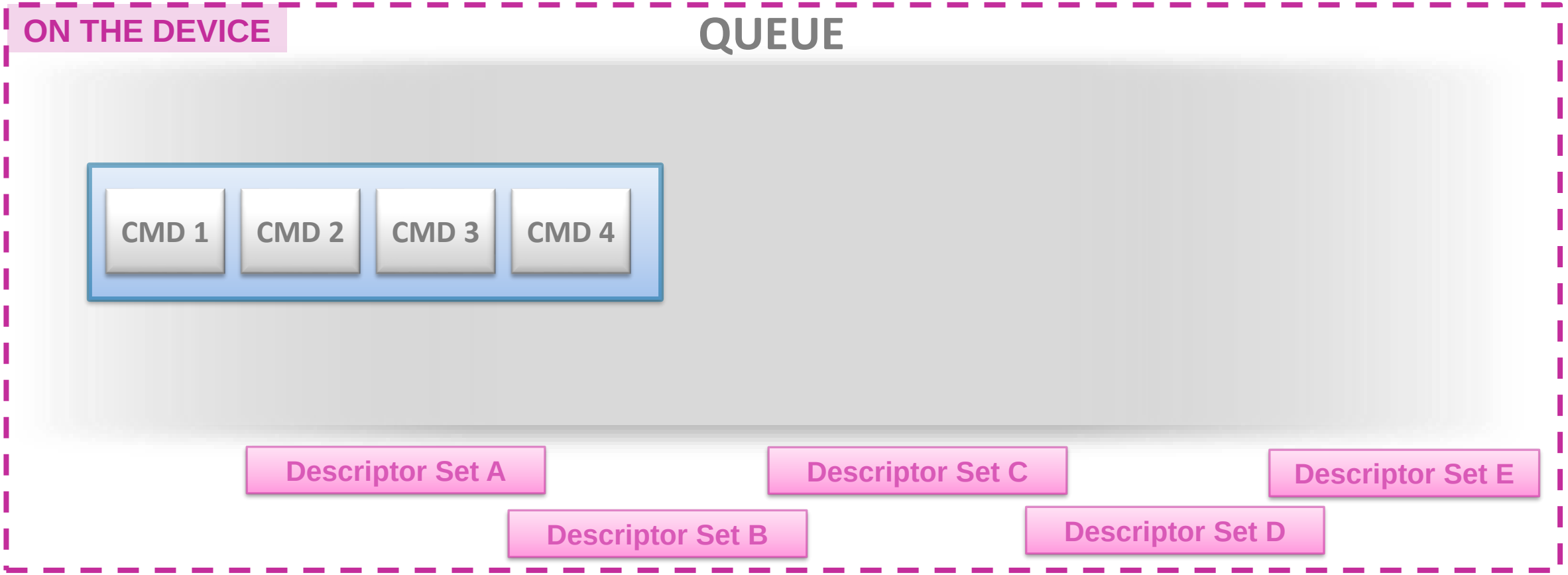
Host -> Device


QUEUE
set = 0
set = 1
set = 2
Descriptor Set A
Descriptor Set B
Descriptor Set C
Descriptor Set D
Descriptor Set E
CMD 1
CMD 2
CMD 3
CMD 4

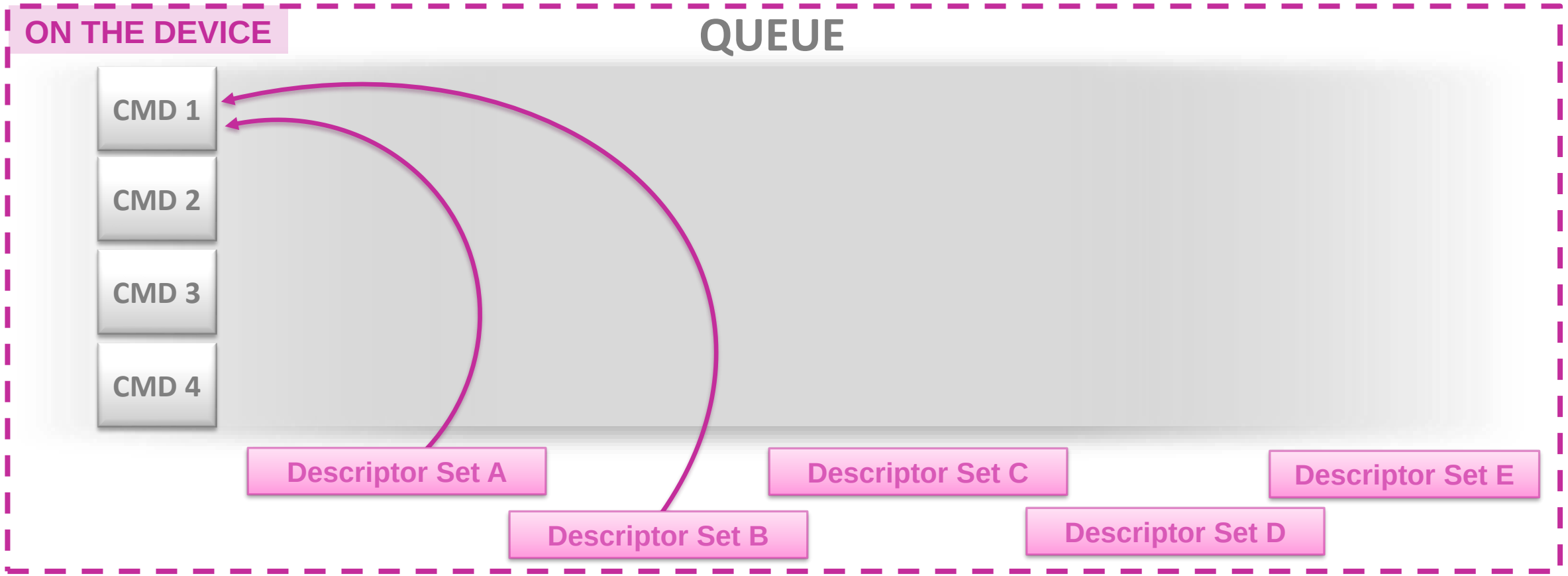
Host -> Device


ON THE DEVICE
QUEUE
Descriptor Set A
Descriptor Set C
Descriptor Set E
Descriptor Set B
Descriptor Set D
ON THE HOST
CMD 1
CMD 2
CMD 3
CMD 4

Host -> Device


ON THE DEVICE
QUEUE
CMD 1
CMD 2
CMD 3
CMD 4
Descriptor Set A
Descriptor Set C
Descriptor Set E
Descriptor Set B
Descriptor Set D

Host -> Device


ON THE DEVICE
QUEUE
CMD 1
CMD 2
CMD 3
CMD 4
Descriptor Set A
Descriptor Set B
Descriptor Set C
Descriptor Set D
Descriptor Set E

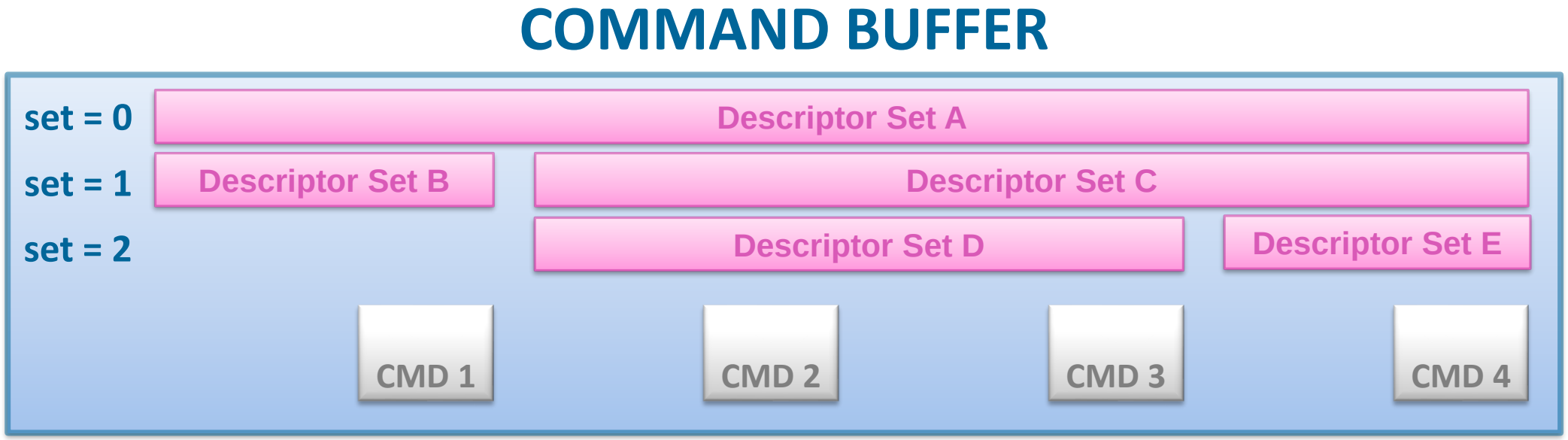
Host -> Device

QUEUE

ON THE HOST

COMMAND BUFFER

Command Buffer Recording



Command Buffer Recording

Graphics Pipeline Commands:

```
vkCmdDraw
vkCmdDrawIndexed
vkCmdDrawIndirect
vkCmdDrawIndirectCount
vkCmdDrawIndexedIndirect
vkCmdDrawIndexedIndirectCount

vkCmdDrawMeshTasksNV
vkCmdDrawMeshTasksIndirectNV
vkCmdDrawMeshTasksIndirectCountNV

vkCmdClearAttachments
```

Compute Pipeline Commands:

```
vkCmdDispatch
vkCmdDispatchBase
vkCmdDispatchIndirect
```

Ray Tracing Pipeline Commands:

```
vkCmdTraceRaysKHR
vkCmdTraceRaysIndirectKHR
```

Transfer Commands:

```
vkCmdCopyBuffer
vkCmdCopyImage
vkCmdCopyBufferToImage
vkCmdCopyImageToBuffer
vkCmdCopyAccelerationStructureKHR
vkCmdCopyAccelerationStructureToMemoryKHR
vkCmdCopyMemoryToAccelerationStructureKHR
vkCmdFillBuffer

vkCmdBlitImage

vkCmdResolveImage

vkCmdClearColorImage
vkCmdClearDepthStencilImage
```

Ray-Tracing Acceleration Structure Build Commands:

```
vkCmdBuildAccelerationStructuresKHR
vkCmdBuildAccelerationStructuresIndirectKHR
```

Bind Commands:

```
vkCmdBindDescriptorSets
vkCmdBindPipeline
vkCmdBindVertexBuffers
vkCmdBindIndexBuffer
```

Other Commands:

```
vkCmdPushConstants
vkCmdPushDescriptorSetKHR
vkCmdSetScissor
vkCmdSetViewport
vkCmdSetDepthBias
...
```

Command Buffer Recording

Graphics Pipeline Commands:

vkCmdDraw
vkCmdDrawIndexed
vkCmdDrawIndirect
vkCmdDrawIndirectCount
vkCmdDrawIndexedIndirect
vkCmdDrawIndexedIndirectCount

vkCmdDrawMeshTasksNV
vkCmdDrawMeshTasksIndirectNV
vkCmdDrawMeshTasksIndirectCountNV

vkCmdClearAttachments

Compute Pipeline Commands:

vkCmdDispatch
vkCmdDispatchBase
vkCmdDispatchIndirect

Ray Tracing Pipeline Commands:

vkCmdTraceRaysKHR
vkCmdTraceRaysIndirectKHR

Transfer Commands:

vkCmdCopyBuffer
vkCmdCopyImage
vkCmdCopyBufferToImage
vkCmdCopyImageToBuffer
vkCmdCopyAccelerationStructureKHR
vkCmdCopyAccelerationStructureToMemoryKHR
vkCmdCopyMemoryToAccelerationStructureKHR
vkCmdFillBuffer

vkCmdBlitImage

vkCmdResolveImage

vkCmdClearColorImage
vkCmdClearDepthStencilImage

Ray-Tracing Acceleration Structure Build Commands:

vkCmdBuildAccelerationStructuresKHR
vkCmdBuildAccelerationStructuresIndirectKHR

Bind Commands:

vkCmdBindDescriptorSets
vkCmdBindPipeline
vkCmdBindVertexBuffers
vkCmdBindIndexBuffer

Other Commands:

vkCmdPushConstants
vkCmdPushDescriptorSetKHR
vkCmdSetScissor
vkCmdSetViewport
vkCmdSetDepthBias
...

Command Buffer Recording

```
VkDevice device = /* ... */; VkQueue queue = /* ... */;

VkCommandPoolCreateInfo poolCreateInfo = { VK_STRUCTURE_TYPE_COMMAND_POOL_CREATE_INFO };
poolCreateInfo.flags = 0;
poolCreateInfo.queueFamilyIndex = 0; // <-- specify queue family index here

VkCommandPool commandPool;
vkCreateCommandPool(device, &poolCreateInfo, nullptr, &commandPool);

VkCommandBufferAllocateInfo allocInfo = { VK_STRUCTURE_TYPE_COMMAND_BUFFER_ALLOCATE_INFO };
allocInfo.commandPool = commandPool;
allocInfo.level = VK_COMMAND_BUFFER_LEVEL_PRIMARY;
allocInfo.commandBufferCount = 1;

VkCommandBuffer commandBuffer;
vkAllocateCommandBuffers(device, &allocInfo, &commandBuffer);

VkCommandBufferBeginInfo beginInfo = { VK_STRUCTURE_TYPE_COMMAND_BUFFER_BEGIN_INFO };
beginInfo.flags = 0;
vkBeginCommandBuffer(commandBuffer, &beginInfo);
// ...
vkCmdBindDescriptorSets(commandBuffer, VK_PIPELINE_BIND_POINT_GRAPHICS, ...);
vkCmdDraw(commandBuffer, ...);
// ...
vkEndCommandBuffer(commandBuffer);
```

Command Buffer Recording

```
VkDevice device = /* ... */; VkQueue queue = /* ... */;

VkCommandPoolCreateInfo poolCreateInfo = { VK_STRUCTURE_TYPE_COMMAND_POOL_CREATE_INFO };
poolCreateInfo.flags = 0;
poolCreateInfo.queueFamilyIndex = 0; // <-- specify queue family index here

VkCommandPool commandPool;
vkCreateCommandPool(device, &poolCreateInfo, nullptr, &commandPool);

VkCommandBufferAllocateInfo allocInfo = { VK_STRUCTURE_TYPE_COMMAND_BUFFER_ALLOCATE_INFO };
allocInfo.commandPool = commandPool;
allocInfo.level = VK_COMMAND_BUFFER_LEVEL_PRIMARY;
allocInfo.commandBufferCount = 1;

VkCommandBuffer commandBuffer;
vkAllocateCommandBuffers(device, &allocInfo, &commandBuffer);

VkCommandBufferBeginInfo beginInfo = { VK_STRUCTURE_TYPE_COMMAND_BUFFER_BEGIN_INFO };
beginInfo.flags = 0;
vkBeginCommandBuffer(commandBuffer, &beginInfo);
// ...
vkCmdBindDescriptorSets(commandBuffer, VK_PIPELINE_BIND_POINT_GRAPHICS, ...);
vkCmdDraw(commandBuffer, ...);
// ...
vkEndCommandBuffer(commandBuffer);
```


Command Buffer Recording

```
VkDevice device = /* ... */; VkQueue queue = /* ... */;

VkCommandPoolCreateInfo poolCreateInfo = { VK_STRUCTURE_TYPE_COMMAND_POOL_CREATE_INFO };
poolCreateInfo.flags = 0;
poolCreateInfo.queueFamilyIndex = 0; // <-- specify queue family index here

VkCommandPool commandPool;
vkCreateCommandPool(device, &poolCreateInfo, nullptr, &commandPool);

VkCommandBufferAllocateInfo allocInfo = { VK_STRUCTURE_TYPE_COMMAND_BUFFER_ALLOCATE_INFO };
allocInfo.commandPool = commandPool;
allocInfo.level = VK_COMMAND_BUFFER_LEVEL_PRIMARY;
allocInfo.commandBufferCount = 1;

VkCommandBuffer commandBuffer;
vkAllocateCommandBuffers(device, &allocInfo, &commandBuffer);

VkCommandBufferBeginInfo beginInfo = { VK_STRUCTURE_TYPE_COMMAND_BUFFER_BEGIN_INFO };
beginInfo.flags = 0;
vkBeginCommandBuffer(commandBuffer, &beginInfo);
// ...
vkCmdBindDescriptorSets(commandBuffer, VK_PIPELINE_BIND_POINT_GRAPHICS, ...);
vkCmdDraw(commandBuffer, ...);
// ...
vkEndCommandBuffer(commandBuffer);
```


Command Buffer Recording

```
VkDevice device = /* ... */; VkQueue queue = /* ... */;

VkCommandPoolCreateInfo poolCreateInfo = { VK_STRUCTURE_TYPE_COMMAND_POOL_CREATE_INFO };
poolCreateInfo.flags = 0;
poolCreateInfo.queueFamilyIndex = 0; // <-- specify queue family index here

VkCommandPool commandPool;
vkCreateCommandPool(device, &poolCreateInfo, nullptr, &commandPool);

VkCommandBufferAllocateInfo allocInfo = { VK_STRUCTURE_TYPE_COMMAND_BUFFER_ALLOCATE_INFO };
allocInfo.commandPool = commandPool;
allocInfo.level = VK_COMMAND_BUFFER_LEVEL_PRIMARY;
allocInfo.commandBufferCount = 1;

VkCommandBuffer commandBuffer;
vkAllocateCommandBuffers(device, &allocInfo, &commandBuffer);

VkCommandBufferBeginInfo beginInfo = { VK_STRUCTURE_TYPE_COMMAND_BUFFER_BEGIN_INFO };
beginInfo.flags = 0;
vkBeginCommandBuffer(commandBuffer, &beginInfo);
// ...
vkCmdBindDescriptorSets(commandBuffer, VK_PIPELINE_BIND_POINT_GRAPHICS, ...);
vkCmdDraw(commandBuffer, ...);
// ...
vkEndCommandBuffer(commandBuffer);
```


Command Buffer Recording

```
VkDevice device = /* ... */; VkQueue queue = /* ... */;

VkCommandPoolCreateInfo poolCreateInfo = { VK_STRUCTURE_TYPE_COMMAND_POOL_CREATE_INFO };
poolCreateInfo.flags = 0;
poolCreateInfo.queueFamilyIndex = 0; // <-- specify queue family index here

VkCommandPool commandPool;
vkCreateCommandPool(device, &poolCreateInfo, nullptr, &commandPool);

VkCommandBufferAllocateInfo allocInfo = { VK_STRUCTURE_TYPE_COMMAND_BUFFER_ALLOCATE_INFO };
allocInfo.commandPool = commandPool;
allocInfo.level = VK_COMMAND_BUFFER_LEVEL_PRIMARY;
allocInfo.commandBufferCount = 1;

VkCommandBuffer commandBuffer;
vkAllocateCommandBuffers(device, &allocInfo, &commandBuffer);

VkCommandBufferBeginInfo beginInfo = { VK_STRUCTURE_TYPE_COMMAND_BUFFER_BEGIN_INFO };
beginInfo.flags = 0;
vkBeginCommandBuffer(commandBuffer, &beginInfo);
// ...
vkCmdBindDescriptorSets(commandBuffer, VK_PIPELINE_BIND_POINT_GRAPHICS, ...);
vkCmdDraw(commandBuffer, ...);
// ...
vkEndCommandBuffer(commandBuffer);
```


Command Buffer Recording

```
VkDevice device = /* ... */; VkQueue queue = /* ... */;

VkCommandPoolCreateInfo poolCreateInfo = { VK_STRUCTURE_TYPE_COMMAND_POOL_CREATE_INFO };
poolCreateInfo.flags = 0;
poolCreateInfo.queueFamilyIndex = 0; // <-- specify queue family index here

VkCommandPool commandPool;
vkCreateCommandPool(device, &poolCreateInfo, nullptr, &commandPool);

VkCommandBufferAllocateInfo allocInfo = { VK_STRUCTURE_TYPE_COMMAND_BUFFER_ALLOCATE_INFO };
allocInfo.commandPool = commandPool;
allocInfo.level = VK_COMMAND_BUFFER_LEVEL_PRIMARY;
allocInfo.commandBufferCount = 1;

VkCommandBuffer commandBuffer;
vkAllocateCommandBuffers(device, &allocInfo, &commandBuffer);

VkCommandBufferBeginInfo beginInfo = { VK_STRUCTURE_TYPE_COMMAND_BUFFER_BEGIN_INFO };
beginInfo.flags = 0;
vkBeginCommandBuffer(commandBuffer, &beginInfo);
// ...
vkCmdBindDescriptorSets(commandBuffer, VK_PIPELINE_BIND_POINT_GRAPHICS, ...);
vkCmdDraw(commandBuffer, ...);
// ...
vkEndCommandBuffer(commandBuffer);
```

Command Buffer Recording

```
VkDevice device = /* ... */; VkQueue queue = /* ... */;

VkCommandPoolCreateInfo poolCreateInfo = { VK_STRUCTURE_TYPE_COMMAND_POOL_CREATE_INFO };
poolCreateInfo.flags = 0;
poolCreateInfo.queueFamilyIndex = 0; // <-- specify queue family index here

VkCommandPool commandPool;
vkCreateCommandPool(device, &poolCreateInfo, nullptr, &commandPool);

VkCommandBufferAllocateInfo allocInfo = { VK_STRUCTURE_TYPE_COMMAND_BUFFER_ALLOCATE_INFO };
allocInfo.commandPool = commandPool;
allocInfo.level = VK_COMMAND_BUFFER_LEVEL_PRIMARY;
allocInfo.commandBufferCount = 1;

VkCommandBuffer commandBuffer;
vkAllocateCommandBuffers(device, &allocInfo, &commandBuffer);

VkCommandBufferBeginInfo beginInfo = { VK_STRUCTURE_TYPE_COMMAND_BUFFER_BEGIN_INFO };
beginInfo.flags = 0;
vkBeginCommandBuffer(commandBuffer, &beginInfo);
// ...
vkCmdBindDescriptorSets(commandBuffer, VK_PIPELINE_BIND_POINT_GRAPHICS, ...);
vkCmdDraw(commandBuffer, ...);
// ...
vkEndCommandBuffer(commandBuffer);
```


Command Buffer Recording

```
VkDevice device = /* ... */; VkQueue queue = /* ... */;

VkCommandPoolCreateInfo poolCreateInfo = { VK_STRUCTURE_TYPE_COMMAND_POOL_CREATE_INFO };
poolCreateInfo.flags = 0;
poolCreateInfo.queueFamilyIndex = 0; // <-- specify queue family index here

VkCommandPool commandPool;
vkCreateCommandPool(device, &poolCreateInfo, nullptr, &commandPool);

VkCommandBufferAllocateInfo allocInfo = { VK_STRUCTURE_TYPE_COMMAND_BUFFER_ALLOCATE_INFO };
allocInfo.commandPool = commandPool;
allocInfo.level = VK_COMMAND_BUFFER_LEVEL_PRIMARY;
allocInfo.commandBufferCount = 1;

VkCommandBuffer commandBuffer;
vkAllocateCommandBuffers(device, &allocInfo, &commandBuffer);

VkCommandBufferBeginInfo beginInfo = { VK_STRUCTURE_TYPE_COMMAND_BUFFER_BEGIN_INFO };
beginInfo.flags = 0;
vkBeginCommandBuffer(commandBuffer, &beginInfo);
// ...
vkCmdBindDescriptorSets(commandBuffer, VK_PIPELINE_BIND_POINT_GRAPHICS, ...);
vkCmdDraw(commandBuffer, ...);
// ...
vkEndCommandBuffer(commandBuffer);
```


Command Buffer Recording

```
VkDevice device = /* ... */; VkQueue queue = /* ... */;

VkCommandPoolCreateInfo poolCreateInfo = { VK_STRUCTURE_TYPE_COMMAND_POOL_CREATE_INFO };
poolCreateInfo.flags = 0;
poolCreateInfo.queueFamilyIndex = 0; // <-- specify queue family index here

VkCommandPool commandPool;
vkCreateCommandPool(device, &poolCreateInfo, nullptr, &commandPool);

VkCommandBufferAllocateInfo allocInfo = { VK_STRUCTURE_TYPE_COMMAND_BUFFER_ALLOCATE_INFO };
allocInfo.commandPool = commandPool;
allocInfo.level = VK_COMMAND_BUFFER_LEVEL_PRIMARY;
allocInfo.commandBufferCount = 1;

VkCommandBuffer commandBuffer;
vkAllocateCommandBuffers(device, &allocInfo, &commandBuffer);

VkCommandBufferBeginInfo beginInfo = { VK_STRUCTURE_TYPE_COMMAND_BUFFER_BEGIN_INFO };
beginInfo.flags = 0;
vkBeginCommandBuffer(commandBuffer, &beginInfo);
// ...
vkCmdBindDescriptorSets(commandBuffer, VK_PIPELINE_BIND_POINT_GRAPHICS, ...);
vkCmdDraw(commandBuffer, ...);
// ...
vkEndCommandBuffer(commandBuffer);
```


Command Buffer Recording

```
VkDevice device = /* ... */; VkQueue queue = /* ... */;

VkCommandPoolCreateInfo poolCreateInfo = { VK_STRUCTURE_TYPE_COMMAND_POOL_CREATE_INFO };
poolCreateInfo.flags = 0;
poolCreateInfo.queueFamilyIndex = 0; // <-- specify queue family index here

VkCommandPool commandPool;
vkCreateCommandPool(device, &poolCreateInfo, nullptr, &commandPool);

VkCommandBufferAllocateInfo allocInfo = { VK_STRUCTURE_TYPE_COMMAND_BUFFER_ALLOCATE_INFO };
allocInfo.commandPool = commandPool;
allocInfo.level = VK_COMMAND_BUFFER_LEVEL_PRIMARY;
allocInfo.commandBufferCount = 1;

VkCommandBuffer commandBuffer;
vkAllocateCommandBuffers(device, &allocInfo, &commandBuffer);

VkCommandBufferBeginInfo beginInfo = { VK_STRUCTURE_TYPE_COMMAND_BUFFER_BEGIN_INFO };
beginInfo.flags = 0;
vkBeginCommandBuffer(commandBuffer, &beginInfo);
// ...
vkCmdBindDescriptorSets(commandBuffer, VK_PIPELINE_BIND_POINT_GRAPHICS, ...);
vkCmdDraw(commandBuffer, ...);
// ...
vkEndCommandBuffer(commandBuffer);
```


Command Buffer Recording

```
VkDevice device = /* ... */; VkQueue queue = /* ... */;

VkCommandPoolCreateInfo poolCreateInfo = { VK_STRUCTURE_TYPE_COMMAND_POOL_CREATE_INFO };
poolCreateInfo.flags = 0;
poolCreateInfo.queueFamilyIndex = 0; // <-- specify queue family index here

VkCommandPool commandPool;
vkCreateCommandPool(device, &poolCreateInfo, nullptr, &commandPool);

VkCommandBufferAllocateInfo allocInfo = { VK_STRUCTURE_TYPE_COMMAND_BUFFER_ALLOCATE_INFO };
allocInfo.commandPool = commandPool;
allocInfo.level = VK_COMMAND_BUFFER_LEVEL_PRIMARY;
allocInfo.commandBufferCount = 1;

VkCommandBuffer commandBuffer;
vkAllocateCommandBuffers(device, &allocInfo, &commandBuffer);

VkCommandBufferBeginInfo beginInfo = { VK_STRUCTURE_TYPE_COMMAND_BUFFER_BEGIN_INFO };
beginInfo.flags = 0;
vkBeginCommandBuffer(commandBuffer, &beginInfo);
// ...
vkCmdBindDescriptorSets(commandBuffer, VK_PIPELINE_BIND_POINT_GRAPHICS, ...);
vkCmdDraw(commandBuffer, ...);
// ...
vkEndCommandBuffer(commandBuffer);
```


Command Buffer Recording

```
VkDevice device = /* ... */; VkQueue queue = /* ... */;

VkCommandPoolCreateInfo poolCreateInfo = { VK_STRUCTURE_TYPE_COMMAND_POOL_CREATE_INFO };
poolCreateInfo.flags = 0;
poolCreateInfo.queueFamilyIndex = 0; // <-- specify queue family index here

VkCommandPool commandPool;
vkCreateCommandPool(device, &poolCreateInfo, nullptr, &commandPool);

VkCommandBufferAllocateInfo allocInfo = { VK_STRUCTURE_TYPE_COMMAND_BUFFER_ALLOCATE_INFO };
allocInfo.commandPool = commandPool;
allocInfo.level = VK_COMMAND_BUFFER_LEVEL_PRIMARY;
allocInfo.commandBufferCount = 1;

VkCommandBuffer commandBuffer;
vkAllocateCommandBuffers(device, &allocInfo, &commandBuffer);

VkCommandBufferBeginInfo beginInfo = { VK_STRUCTURE_TYPE_COMMAND_BUFFER_BEGIN_INFO };
beginInfo.flags = 0;
vkBeginCommandBuffer(commandBuffer, &beginInfo);
// ...
vkCmdBindDescriptorSets(commandBuffer, VK_PIPELINE_BIND_POINT_GRAPHICS, ...);
vkCmdDraw(commandBuffer, ...);
// ...
vkEndCommandBuffer(commandBuffer);
```


Command Buffer Recording

```
VkDevice device = /* ... */; VkQueue queue = /* ... */;

VkCommandPoolCreateInfo poolCreateInfo = { VK_STRUCTURE_TYPE_COMMAND_POOL_CREATE_INFO };
poolCreateInfo.flags = 0;
poolCreateInfo.queueFamilyIndex = 0; // <-- specify queue family index here

VkCommandPool commandPool;
vkCreateCommandPool(device, &poolCreateInfo, nullptr, &commandPool);

VkCommandBufferAllocateInfo allocInfo = { VK_STRUCTURE_TYPE_COMMAND_BUFFER_ALLOCATE_INFO };
allocInfo.commandPool = commandPool;
allocInfo.level = VK_COMMAND_BUFFER_LEVEL_PRIMARY;
allocInfo.commandBufferCount = 1;

VkCommandBuffer commandBuffer;
vkAllocateCommandBuffers(device, &allocInfo, &commandBuffer);

VkCommandBufferBeginInfo beginInfo = { VK_STRUCTURE_TYPE_COMMAND_BUFFER_BEGIN_INFO };
beginInfo.flags = 0;
vkBeginCommandBuffer(commandBuffer, &beginInfo);
// ...
vkCmdBindDescriptorSets(commandBuffer, VK_PIPELINE_BIND_POINT_GRAPHICS, ...);
vkCmdDraw(commandBuffer, ...);
// ...
vkEndCommandBuffer(commandBuffer);
```


Command Buffer Recording

```
VkDevice device = /* ... */; VkQueue queue = /* ... */;

VkCommandPoolCreateInfo poolCreateInfo = { VK_STRUCTURE_TYPE_COMMAND_POOL_CREATE_INFO };
poolCreateInfo.flags = 0;
poolCreateInfo.queueFamilyIndex = 0; // <-- specify queue family index here

VkCommandPool commandPool;
vkCreateCommandPool(device, &poolCreateInfo, nullptr, &commandPool);

VkCommandBufferAllocateInfo allocInfo = { VK_STRUCTURE_TYPE_COMMAND_BUFFER_ALLOCATE_INFO };
allocInfo.commandPool = commandPool;
allocInfo.level = VK_COMMAND_BUFFER_LEVEL_PRIMARY;
allocInfo.commandBufferCount = 1;

VkCommandBuffer commandBuffer;
vkAllocateCommandBuffers(device, &allocInfo, &commandBuffer);

VkCommandBufferBeginInfo beginInfo = { VK_STRUCTURE_TYPE_COMMAND_BUFFER_BEGIN_INFO };
beginInfo.flags = 0;
vkBeginCommandBuffer(commandBuffer, &beginInfo);
// ...
vkCmdBindDescriptorSets(commandBuffer, VK_PIPELINE_BIND_POINT_GRAPHICS, ...);
vkCmdDraw(commandBuffer, ...);
// ...
vkEndCommandBuffer(commandBuffer);
```

Command Buffer Recording

```
VkDevice device = /* ... */; VkQueue queue = /* ... */;

VkCommandPoolCreateInfo poolCreateInfo = { VK_STRUCTURE_TYPE_COMMAND_POOL_CREATE_INFO };
poolCreateInfo.flags = 0;
poolCreateInfo.queueFamilyIndex = 0; // <-- specify queue family index here

VkCommandPool commandPool;
vkCreateCommandPool(device, &poolCreateInfo, nullptr, &commandPool);

VkCommandBufferAllocateInfo allocInfo = { VK_STRUCTURE_TYPE_COMMAND_BUFFER_ALLOCATE_INFO };
allocInfo.commandPool = commandPool;
allocInfo.level = VK_COMMAND_BUFFER_LEVEL_PRIMARY;
allocInfo.commandBufferCount = 1;

VkCommandBuffer commandBuffer;
vkAllocateCommandBuffers(device, &allocInfo, &commandBuffer);

VkCommandBufferBeginInfo beginInfo = { VK_STRUCTURE_TYPE_COMMAND_BUFFER_BEGIN_INFO };
beginInfo.flags = 0;
vkBeginCommandBuffer(commandBuffer, &beginInfo);
// ...
vkCmdBindDescriptorSets(commandBuffer, VK_PIPELINE_BIND_POINT_GRAPHICS, ...);
vkCmdDraw(commandBuffer, ...);
// ...
vkEndCommandBuffer(commandBuffer);
```

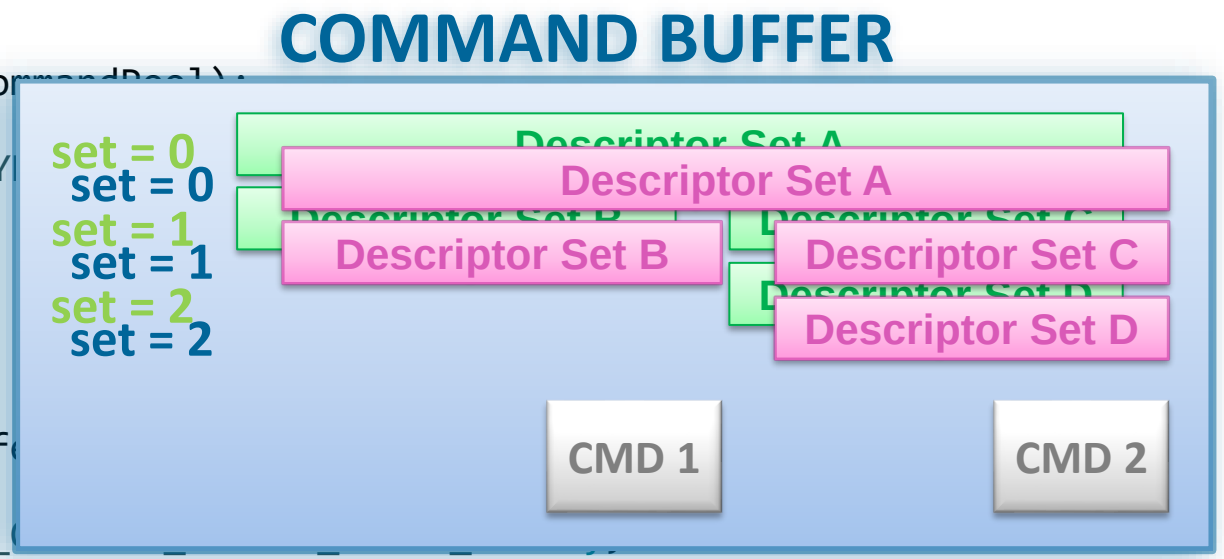

Command Buffer Recording

```
VkDevice device = /* ... */; VkQueue queue = /* ... */;

VkCommandPoolCreateInfo poolCreateInfo = { VK_STRUCTURE_TYPE_COMMAND_POOL_CREATE_INFO };
poolCreateInfo.flags = 0;
poolCreateInfo.queueFamilyIndex = 0; // <-- specify queue family index here

VkCommandPool commandPool;
vkCreateCommandPool(device, &poolCreateInfo, nullptr, &commandPool);

VkCommandBufferAllocateInfo allocInfo = { VK_STRUCTURE_TYPE_COMMAND_BUFFER_ALLOCATE_INFO };
allocInfo.commandPool = commandPool;
allocInfo.level = VK_COMMAND_BUFFER_LEVEL_PRIMARY;
allocInfo.commandBufferCount = 1;

VkCommandBuffer commandBuffer;
vkAllocateCommandBuffers(device, &allocInfo, &commandBuffer);

VkCommandBufferBeginInfo beginInfo = { VK_STRUCTURE_TYPE_COMMAND_BUFFER_BEGIN_INFO };
beginInfo.flags = 0;
vkBeginCommandBuffer(commandBuffer, &beginInfo);
// ...
vkCmdBindDescriptorSets(commandBuffer, VK_PIPELINE_BIND_POINT_GRAPHICS, ...);
vkCmdDraw(commandBuffer, ...);
// ...
vkEndCommandBuffer(commandBuffer);
```

Defines the **state**

Command Buffer Recording

```
VkDevice device = /* ... */; VkQueue queue = /* ... */;

VkCommandPoolCreateInfo poolCreateInfo = { VK_STRUCTURE_TYPE_COMMAND_POOL_CREATE_INFO };
poolCreateInfo.flags = 0;
poolCreateInfo.queueFamilyIndex = 0; // <-- specify queue family index here

VkCommandPool commandPool;
vkCreateCommandPool(device, &poolCreateInfo, nullptr, &co

VkCommandBufferAllocateInfo allocInfo = { VK_STRUCTURE_TY
allocInfo.commandPool = commandPool;
allocInfo.level = VK_COMMAND_BUFFER_LEVEL_PRIMARY;
allocInfo.commandBufferCount = 1;

VkCommandBuffer commandBuffer;
vkAllocateCommandBuffers(device, &allocInfo, &commandBuff

VkCommandBufferBeginInfo beginInfo = { VK_STRUCTURE_TYPE_
beginInfo.flags = 0;
vkBeginCommandBuffer(commandBuffer, &beginInfo);
// ...
vkCmdBindDescriptorSets(commandBuffer, VK_PIPELINE_BIND_POINT_GRAPHICS, ...);
vkCmdDraw(commandBuffer, ...);
// ...
vkEndCommandBuffer(commandBuffer);
```



Defines the **state**
…per **BIND_POINT**!

Command Buffer Recording

```
VkDevice device = /* ... */; VkQueue queue = /* ... */;

VkCommandPoolCreateInfo poolCreateInfo = { VK_STRUCTURE_TYPE_COMMAND_POOL_CREATE_INFO };
poolCreateInfo.flags = 0;
poolCreateInfo.queueFamilyIndex = 0; // <-- specify queue family index here

VkCommandPool commandPool;
vkCreateCommandPool(device, &poolCreateInfo, nullptr, &commandPool);

VkCommandBufferAllocateInfo allocInfo = { VK_STRUCTURE_TYPE_COMMAND_BUFFER_ALLOCATE_INFO };
allocInfo.commandPool = commandPool;
allocInfo.level = VK_COMMAND_BUFFER_LEVEL_PRIMARY;
allocInfo.commandBufferCount = 1;

VkCommandBuffer commandBuffer;
vkAllocateCommandBuffers(device, &allocInfo, &commandBuffer);

VkCommandBufferBeginInfo beginInfo = { VK_STRUCTURE_TYPE_COMMAND_BUFFER_BEGIN_INFO };
beginInfo.flags = 0;
vkBeginCommandBuffer(commandBuffer, &beginInfo);
// ...
vkCmdBindDescriptorSets(commandBuffer, VK_PIPELINE_BIND_POINT_GRAPHICS, ...);
vkCmdDraw(commandBuffer, ...);
// ...
vkEndCommandBuffer(commandBuffer);
```

Defines the **state**
…per **BIND_POINT**!

Records an **action**

Command Buffer Recording

```
VkDevice device = /* ... */; VkQueue queue = /* ... */;

VkCommandPoolCreateInfo poolCreateInfo = { VK_STRUCTURE_TYPE_COMMAND_POOL_CREATE_INFO };
poolCreateInfo.flags = 0;
poolCreateInfo.queueFamilyIndex = 0; // <-- specify queue family index here

VkCommandPool commandPool;
vkCreateCommandPool(device, &poolCreateInfo, nullptr, &commandPool);

VkCommandBufferAllocateInfo allocInfo = { VK_STRUCTURE_TYPE_COMMAND_BUFFER_ALLOCATE_INFO };
allocInfo.commandPool = commandPool;
allocInfo.level = VK_COMMAND_BUFFER_LEVEL_PRIMARY;
allocInfo.commandBufferCount = 1;

VkCommandBuffer commandBuffer;
vkAllocateCommandBuffers(device, &allocInfo, &commandBuffer);

VkCommandBufferBeginInfo beginInfo = { VK_STRUCTURE_TYPE_COMMAND_BUFFER_BEGIN_INFO };
beginInfo.flags = 0;
vkBeginCommandBuffer(commandBuffer, &beginInfo);
// ...
vkCmdBindDescriptorSets(commandBuffer, VK_PIPELINE_BIND_POINT_GRAPHICS, ...);
vkCmdDraw(commandBuffer, ...);
// ...
vkEndCommandBuffer(commandBuffer);
```

Defines the **state**
…per **BIND_POINT**!

Records an **action**
…using the **current state**!

Command Buffer Recording

```
VkDevice device = /* ... */; VkQueue queue = /* ... */;

VkCommandPoolCreateInfo poolCreateInfo = { VK_STRUCTURE_TYPE_COMMAND_POOL_CREATE_INFO };
poolCreateInfo.flags = 0;
poolCreateInfo.queueFamilyIndex = 0; // <-- specify queue family index here

VkCommandPool commandPool;
vkCreateCommandPool(device, &poolCreateInfo, nullptr, &commandPool);

VkCommandBufferAllocateInfo allocInfo = { VK_STRUCTURE_TYPE_COMMAND_BUFFER_ALLOCATE_INFO };
allocInfo.commandPool = commandPool;
allocInfo.level = VK_COMMAND_BUFFER_LEVEL_PRIMARY;
allocInfo.commandBufferCount = 1;

VkCommandBuffer commandBuffer;
vkAllocateCommandBuffers(device, &allocInfo, &commandBuffer);

VkCommandBufferBeginInfo beginInfo = { VK_STRUCTURE_TYPE_COMMAND_BUFFER_BEGIN_INFO };
beginInfo.flags = 0;
vkBeginCommandBuffer(commandBuffer, &beginInfo);
// ...
vkCmdBindDescriptorSets(commandBuffer, VK_PIPELINE_BIND_POINT_GRAPHICS, ...);
vkCmdDraw(commandBuffer, ...);
// ...
vkEndCommandBuffer(commandBuffer);
```

Defines the **state**
…per **BIND_POINT**!

Records an **action**
…using the **current state**!

Command Buffer Recording

```
VkDevice device = /* ... */; VkQueue queue = /* ... */;

VkCommandPoolCreateInfo poolCreateInfo = { VK_STRUCTURE_TYPE_COMMAND_POOL_CREATE_INFO };
poolCreateInfo.flags = 0;
poolCreateInfo.queueFamilyIndex = 0; // <-- specify queue family index here

VkCommandPool commandPool;
vkCreateCommandPool(device, &poolCreateInfo, nullptr, &commandPool);

VkCommandBufferAllocateInfo allocInfo = { VK_STRUCTURE_TYPE_COMMAND_BUFFER_ALLOCATE_INFO };
allocInfo.commandPool = commandPool;
allocInfo.level = VK_COMMAND_BUFFER_LEVEL_PRIMARY;
allocInfo.commandBufferCount = 1;

VkCommandBuffer commandBuffer;
vkAllocateCommandBuffers(device, &allocInfo, &commandBuffer);

VkCommandBufferBeginInfo beginInfo = { VK_STRUCTURE_TYPE_COMMAND_BUFFER_BEGIN_INFO };
beginInfo.flags = 0;
vkBeginCommandBuffer(commandBuffer, &beginInfo);
// ...
vkCmdBindDescriptorSets(commandBuffer, VK_PIPELINE_BIND_POINT_GRAPHICS, ...);
vkCmdDraw(commandBuffer, ...);
// ...
vkEndCommandBuffer(commandBuffer);
```

⚠ REFACTORING IN PROGRESS...

Command Buffer Recording

```
VkCommandPoolCreateInfo poolCreateInfo = { VK_STRUCTURE_TYPE_COMMAND_POOL_CREATE_INFO };
poolCreateInfo.flags = 0;
poolCreateInfo.queueFamilyIndex = 0; // <-- specify queue family index here

VkCommandBufferAllocateInfo allocInfo = { VK_STRUCTURE_TYPE_COMMAND_BUFFER_ALLOCATE_INFO };
allocInfo.commandPool = commandPool;
allocInfo.level = VK_COMMAND_BUFFER_LEVEL_PRIMARY;
allocInfo.commandBufferCount = 1;

VkCommandBuffer commandBuffer;
vkAllocateCommandBuffers(device, &allocInfo, &commandBuffer);

VkCommandBufferBeginInfo beginInfo = { VK_STRUCTURE_TYPE_COMMAND_BUFFER_BEGIN_INFO };
beginInfo.flags = 0;
vkBeginCommandBuffer(commandBuffer, &beginInfo);
// ...
vkCmdBindDescriptorSets(commandBuffer, VK_PIPELINE_BIND_POINT_GRAPHICS, ...);
vkCmdDraw(commandBuffer, ...);
// ...
vkEndCommandBuffer(commandBuffer);
```

⚠ REFACTORING IN PROGRESS...

Command Buffer Recording

```
VkCommandPoolCreateInfo poolCreateInfo = { VK_STRUCTURE_TYPE_COMMAND_POOL_CREATE_INFO };
poolCreateInfo.flags = 0;
poolCreateInfo.queueFamilyIndex = 0; // <-- specify queue family index here

VkCommandBufferAllocateInfo allocInfo = { VK_STRUCTURE_TYPE_COMMAND_BUFFER_ALLOCATE_INFO };
allocInfo.commandPool = commandPool;
allocInfo.level = VK_COMMAND_BUFFER_LEVEL_PRIMARY;
allocInfo.commandBufferCount = 1;

VkCommandBuffer commandBuffer;
vkAllocateCommandBuffers(device, &allocInfo, &commandBuffer);

VkCommandBufferBeginInfo beginInfo = { VK_STRUCTURE_TYPE_COMMAND_BUFFER_BEGIN_INFO };
beginInfo.flags = 0;
vkBeginCommandBuffer(commandBuffer, &beginInfo);
vkCmdBindDescriptorSets(commandBuffer, VK_PIPELINE_BIND_POINT_GRAPHICS, ...);
vkCmdDraw(commandBuffer, ...);
vkEndCommandBuffer(commandBuffer);
```

⚠ REFACTORING IN PROGRESS...

Command Buffer Recording

```
VkCommandPoolCreateInfo poolCreateInfo = { VK_STRUCTURE_TYPE_COMMAND_POOL_CREATE_INFO };
poolCreateInfo.flags = 0; // ...

VkCommandBufferAllocateInfo allocInfo = { VK_STRUCTURE_TYPE_COMMAND_BUFFER_ALLOCATE_INFO };
allocInfo.level = VK_COMMAND_BUFFER_LEVEL_PRIMARY; // ...

VkCommandBuffer commandBuffer;
vkAllocateCommandBuffers(device, &allocInfo, &commandBuffer);

VkCommandBufferBeginInfo beginInfo = { VK_STRUCTURE_TYPE_COMMAND_BUFFER_BEGIN_INFO };
beginInfo.flags = 0;
vkBeginCommandBuffer(commandBuffer, &beginInfo);
vkCmdBindDescriptorSets(commandBuffer, VK_PIPELINE_BIND_POINT_GRAPHICS, ...);
vkCmdDraw(commandBuffer, ...);
vkEndCommandBuffer(commandBuffer);
```

⚠ REFACTORING IN PROGRESS...

Command Buffer Recording

```
VkCommandPoolCreateInfo poolCreateInfo = { VK_STRUCTURE_TYPE_COMMAND_POOL_CREATE_INFO };
poolCreateInfo.flags = 0; // ...

VkCommandBufferAllocateInfo allocInfo = { VK_STRUCTURE_TYPE_COMMAND_BUFFER_ALLOCATE_INFO };
allocInfo.level = VK_COMMAND_BUFFER_LEVEL_PRIMARY; // ...

VkCommandBuffer commandBuffer;
vkAllocateCommandBuffers(device, &allocInfo, &commandBuffer);

VkCommandBufferBeginInfo beginInfo = { VK_STRUCTURE_TYPE_COMMAND_BUFFER_BEGIN_INFO };
beginInfo.flags = 0;
vkBeginCommandBuffer(commandBuffer, &beginInfo);
vkCmdBindDescriptorSets(commandBuffer, VK_PIPELINE_BIND_POINT_GRAPHICS, ...);
vkCmdDraw(commandBuffer, ...);
vkEndCommandBuffer(commandBuffer);

while (true) {
  // ...

  VkSubmitInfo submitInfo = {};
  submitInfo.sType = VK_STRUCTURE_TYPE_SUBMIT_INFO;
  submitInfo.commandBufferCount = 1;
  submitInfo.pCommandBuffers = &commandBuffer;
  vkQueueSubmit(queue, 1, &submitInfo, VK_NULL_HANDLE);
  // ...
}
```

☑ REFACTORING DONE

Command Buffer Recording

```
VkCommandPoolCreateInfo poolCreateInfo = { VK_STRUCTURE_TYPE_COMMAND_POOL_CREATE_INFO };
poolCreateInfo.flags = 0; // ...

VkCommandBufferAllocateInfo allocInfo = { VK_STRUCTURE_TYPE_COMMAND_BUFFER_ALLOCATE_INFO };
allocInfo.level = VK_COMMAND_BUFFER_LEVEL_PRIMARY; // ...

VkCommandBuffer commandBuffer;
vkAllocateCommandBuffers(device, &allocInfo, &commandBuffer);

VkCommandBufferBeginInfo beginInfo = { VK_STRUCTURE_TYPE_COMMAND_BUFFER_BEGIN_INFO };
beginInfo.flags = 0;
vkBeginCommandBuffer(commandBuffer, &beginInfo);
vkCmdBindDescriptorSets(commandBuffer, VK_PIPELINE_BIND_POINT_GRAPHICS, ...);
vkCmdDraw(commandBuffer, ...);
vkEndCommandBuffer(commandBuffer);

while (true) {
  // ...

  VkSubmitInfo submitInfo = {};
  submitInfo.sType = VK_STRUCTURE_TYPE_SUBMIT_INFO;
  submitInfo.commandBufferCount = 1;
  submitInfo.pCommandBuffers = &commandBuffer;
  vkQueueSubmit(queue, 1, &submitInfo, VK_NULL_HANDLE);
  // ...
}
```

Command Buffer Recording

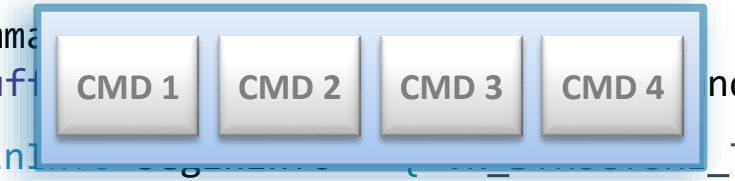
```
VkCommandPoolCreateInfo poolCreateInfo = { VK_STRUCTURE_TYPE_COMMAND_POOL_CREATE_INFO };
poolCreateInfo.flags = 0; // ...

VkCommandBufferAllocateInfo allocInfo = { VK_STRUCTURE_TYPE_COMMAND_BUFFER_ALLOCATE_INFO };
allocInfo.level = VK_COMMAND_BUFFER_LEVEL_PRIMARY; // ...

VkCommandBuffer commandBuffer;
vkAllocateCommandBuffers(device, &allocInfo, &commandBuffer);

VkCommandBufferBeginInfo beginInfo = { VK_STRUCTURE_TYPE_COMMAND_BUFFER_BEGIN_INFO };
beginInfo.flags = 0;
vkBeginCommandBuffer(commandBuffer, &beginInfo);
vkCmdBindDescriptorSets(commandBuffer, VK_PIPELINE_BIND_POINT_GRAPHICS, ...);
vkCmdDraw(commandBuffer, ...);
vkEndCommandBuffer(commandBuffer);

while (true) {
  // ...

  VkSubmitInfo submitInfo = {};
  submitInfo.sType = VK_STRUCTURE_TYPE_SUBMIT_INFO;
  submitInfo.commandBufferCount = 1;
  submitInfo.pCommandBuffers = &commandBuffer;
  vkQueueSubmit(queue, 1, &submitInfo, VK_NULL_HANDLE);
  // ...
}
```

Command Buffer Recording

```
VkCommandPoolCreateInfo poolCreateInfo = { VK_STRUCTURE_TYPE_COMMAND_POOL_CREATE_INFO };
poolCreateInfo.flags = 0; // ...

VkCommandBufferAllocateInfo allocInfo = { VK_STRUCTURE_TYPE_COMMAND_BUFFER_ALLOCATE_INFO };
allocInfo.level = VK_COMMAND_BUFFER_LEVEL_PRIMARY; // ...

VkCommandBuffer commandBuffer;
vkAllocateCommandBuffers(device, &allocInfo, &commandBuffer);

VkCommandBufferBeginInfo beginInfo = { VK_STRUCTURE_TYPE_COMMAND_BUFFER_BEGIN_INFO };
beginInfo.flags = 0;
vkBeginCommandBuffer(commandBuffer, &beginInfo);
vkCmdBindDescriptorSets(commandBuffer, VK_PIPELINE_BIND_POINT_GRAPHICS, ...);
vkCmdDraw(commandBuffer, ...);
vkEndCommandBuffer(commandBuffer);

while (true) {
  // ...

  VkSubmitInfo submitInfo = {};
  submitInfo.sType = VK_STRUCTURE_TYPE_SUBMIT_INFO;
  submitInfo.commandBufferCount = 1;
  submitInfo.pCommandBuffers = &commandBuffer;
  vkQueueSubmit(queue, 1, &submitInfo, VK_NULL_HANDLE);
  // ...
}
```

Command Buffer Recording

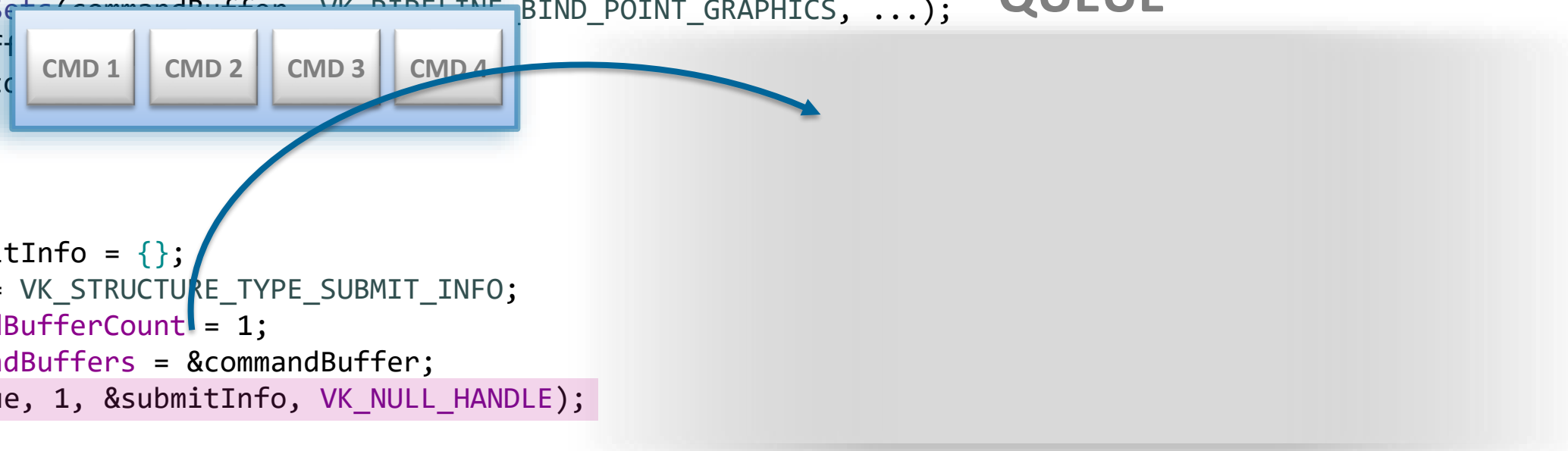
```
VkCommandPoolCreateInfo poolCreateInfo = { VK_STRUCTURE_TYPE_COMMAND_POOL_CREATE_INFO };
poolCreateInfo.flags = 0; // ...

VkCommandBufferAllocateInfo allocInfo = { VK_STRUCTURE_TYPE_COMMAND_BUFFER_ALLOCATE_INFO };
allocInfo.level = VK_COMMAND_BUFFER_LEVEL_PRIMARY; // ...

VkCommandBuffer comma
vkAllocateCommandBuff                                    ndBuffer);

VkCommandBufferBeginI                                    _TYPE_COMMAND_BUFFER_BEGIN_INFO };
beginInfo.flags = 0;
vkBeginCommandBuffer(commandBuffer, &beginInfo);
vkCmdBindDescriptorSets(commandBuffer, VK_PIPELINE_BIND_POINT_GRAPHICS, ...);
vkCmdDraw(commandBuffer, ...);
vkEndCommandBuffer(commandBuffer);

while (true) {
  // ...

  VkSubmitInfo submitInfo = {};
  submitInfo.sType = VK_STRUCTURE_TYPE_SUBMIT_INFO;
  submitInfo.commandBufferCount = 1;
  submitInfo.pCommandBuffers = &commandBuffer;
  vkQueueSubmit(queue, 1, &submitInfo, VK_NULL_HANDLE);
  // ...
}
```



QUEUE

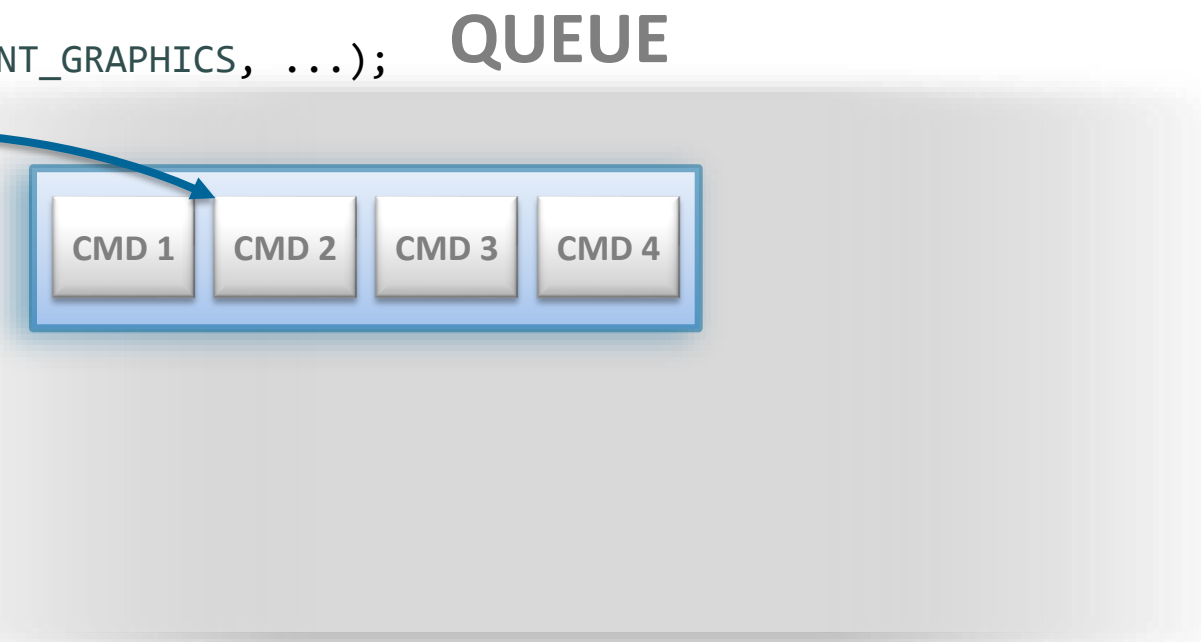
Command Buffer Recording

```
VkCommandPoolCreateInfo poolCreateInfo = { VK_STRUCTURE_TYPE_COMMAND_POOL_CREATE_INFO };
poolCreateInfo.flags = 0; // ...

VkCommandBufferAllocateInfo allocInfo = { VK_STRUCTURE_TYPE_COMMAND_BUFFER_ALLOCATE_INFO };
allocInfo.level = VK_COMMAND_BUFFER_LEVEL_PRIMARY; // ...

VkCommandBuffer commandBuffer;
vkAllocateCommandBuffers(device, &allocInfo, &commandBuffer);

VkCommandBufferBeginInfo beginInfo = { VK_STRUCTURE_TYPE_COMMAND_BUFFER_BEGIN_INFO };
beginInfo.flags = 0;
vkBeginCommandBuffer(commandBuffer, &beginInfo);
vkCmdBindDescriptorSets(commandBuffer, VK_PIPELINE_BIND_POINT_GRAPHICS, ...);
vkCmdDraw(commandBuff
vkEndCommandBuffer(cc

while (true) {
  // ...

  VkSubmitInfo submitInfo = {};
  submitInfo.sType = VK_STRUCTURE_TYPE_SUBMIT_INFO;
  submitInfo.commandBufferCount = 1;
  submitInfo.pCommandBuffers = &commandBuffer;
  vkQueueSubmit(queue, 1, &submitInfo, VK_NULL_HANDLE);
  // ...
}
```



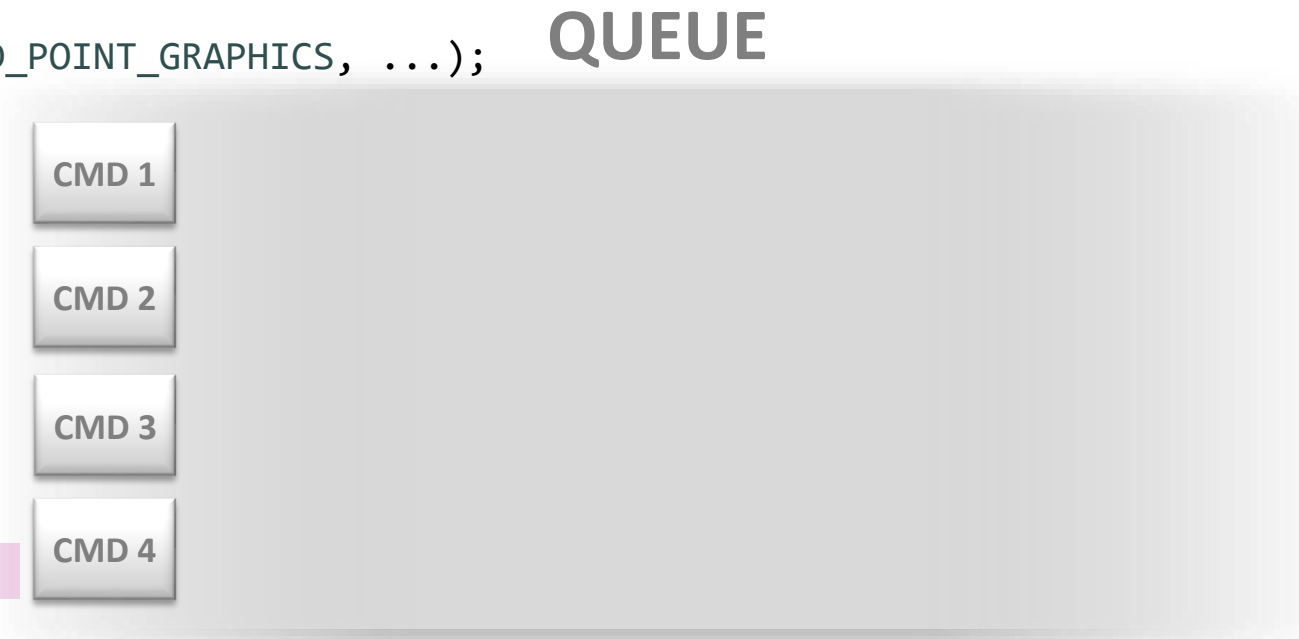
Command Buffer Recording

```
VkCommandPoolCreateInfo poolCreateInfo = { VK_STRUCTURE_TYPE_COMMAND_POOL_CREATE_INFO };
poolCreateInfo.flags = 0; // ...

VkCommandBufferAllocateInfo allocInfo = { VK_STRUCTURE_TYPE_COMMAND_BUFFER_ALLOCATE_INFO };
allocInfo.level = VK_COMMAND_BUFFER_LEVEL_PRIMARY; // ...

VkCommandBuffer commandBuffer;
vkAllocateCommandBuffers(device, &allocInfo, &commandBuffer);

VkCommandBufferBeginInfo beginInfo = { VK_STRUCTURE_TYPE_COMMAND_BUFFER_BEGIN_INFO };
beginInfo.flags = 0;
vkBeginCommandBuffer(commandBuffer, &beginInfo);
vkCmdBindDescriptorSets(commandBuffer, VK_PIPELINE_BIND_POINT_GRAPHICS, ...);
vkCmdDraw(commandBuffer, ...);
vkEndCommandBuffer(commandBuffer);

while (true) {
  // ...

  VkSubmitInfo submitInfo = {};
  submitInfo.sType = VK_STRUCTURE_TYPE_SUBMIT_INFO;
  submitInfo.commandBufferCount = 1;
  submitInfo.pCommandBuffers = &commandBuffer;
  vkQueueSubmit(queue, 1, &submitInfo, VK_NULL_HANDLE);
  // ...
}
```



Command Buffer Recording

```
VkCommandPoolCreateInfo poolCreateInfo = { VK_STRUCTURE_TYPE_COMMAND_POOL_CREATE_INFO };
poolCreateInfo.flags = 0; // ...

VkCommandBufferAllocateInfo allocInfo = { VK_STRUCTURE_TYPE_COMMAND_BUFFER_ALLOCATE_INFO };
allocInfo.level = VK_COMMAND_BUFFER_LEVEL_PRIMARY; // ...

VkCommandBuffer commandBuffer;
vkAllocateCommandBuffers(device, &allocInfo, &commandBuffer);

VkCommandBufferBeginInfo beginInfo = { VK_STRUCTURE_TYPE_COMMAND_BUFFER_BEGIN_INFO };
beginInfo.flags = 0;
vkBeginCommandBuffer(commandBuffer, &beginInfo);
vkCmdBindDescriptorSets(commandBuffer, VK_PIPELINE_BIND_POINT_GRAPHICS, ...);
vkCmdDraw(commandBuffer, ...);
vkEndCommandBuffer(commandBuffer);

while (true) {
  // ...

  VkSubmitInfo submitInfo = {};
  submitInfo.sType = VK_STRUCTURE_TYPE_SUBMIT_INFO;
  submitInfo.commandBufferCount = 1;
  submitInfo.pCommandBuffers = &commandBuffer;
  vkQueueSubmit(queue, 1, &submitInfo, VK_NULL_HANDLE);
  // ...
}
```



Command Buffer Recording

```
VkCommandPoolCreateInfo poolCreateInfo = { VK_STRUCTURE_TYPE_COMMAND_POOL_CREATE_INFO };
poolCreateInfo.flags = 0; // ...

VkCommandBufferAllocateInfo allocInfo = { VK_STRUCTURE_TYPE_COMMAND_BUFFER_ALLOCATE_INFO };
allocInfo.level = VK_COMMAND_BUFFER_LEVEL_PRIMARY; // ...

VkCommandBuffer commandBuffer;
vkAllocateCommandBuffers(device, &allocInfo, &commandBuffer);

VkCommandBufferBeginInfo beginInfo = { VK_STRUCTURE_TYPE_COMMAND_BUFFER_BEGIN_INFO };
beginInfo.flags = 0;
vkBeginCommandBuffer(commandBuffer, &beginInfo);
vkCmdBindDescriptorSets(commandBuffer, VK_PIPELINE_BIND_POINT_GRAPHICS, ...);
vkCmdDraw(commandBuffer, ...);
vkEndCommandBuffer(commandBuffer);

while (true) {
  // ...

  VkSubmitInfo submitInfo = {};
  submitInfo.sType = VK_STRUCTURE_TYPE_SUBMIT_INFO;
  submitInfo.commandBufferCount = 1;
  submitInfo.pCommandBuffers = &commandBuffer;
  vkQueueSubmit(queue, 1, &submitInfo, VK_NULL_HANDLE);
  // ...
}
```

Command Buffer Usage Modes:

- Reuse (submit multiple times)
- Single-use (submit once)
- Reset and re-record

Command Buffer Recording

```
VkCommandPoolCreateInfo poolCreateInfo = { VK_STRUCTURE_TYPE_COMMAND_POOL_CREATE_INFO };
poolCreateInfo.flags = 0; // ...

VkCommandBufferAllocateInfo allocInfo = { VK_STRUCTURE_TYPE_COMMAND_BUFFER_ALLOCATE_INFO };
allocInfo.level = VK_COMMAND_BUFFER_LEVEL_PRIMARY; // ...

VkCommandBuffer commandBuffer;
vkAllocateCommandBuffers(device, &allocInfo, &commandBuffer);

VkCommandBufferBeginInfo beginInfo = { VK_STRUCTURE_TYPE_COMMAND_BUFFER_BEGIN_INFO };
beginInfo.flags = 0;
vkBeginCommandBuffer(commandBuffer, &beginInfo);
vkCmdBindDescriptorSets(commandBuffer, VK_PIPELINE_BIND_POINT_GRAPHICS, ...);
vkCmdDraw(commandBuffer, ...);
vkEndCommandBuffer(commandBuffer);
```

```
while (true) {
  // ...

  VkSubmitInfo submitInfo = {};
  submitInfo.sType = VK_STRUCTURE_TYPE_SUBMIT_INFO;
  submitInfo.commandBufferCount = 1;
  submitInfo.pCommandBuffers = &commandBuffer;
  vkQueueSubmit(queue, 1, &submitInfo, VK_NULL_HANDLE);
  // ...
}
```

Command Buffer Usage Modes:

- **Reuse** (submit multiple times)
- Single-use (submit once)
- Reset and re-record

Command Buffer Recording

```
VkCommandPoolCreateInfo poolCreateInfo = { VK_STRUCTURE_TYPE_COMMAND_POOL_CREATE_INFO };
poolCreateInfo.flags = 0; // ...

VkCommandBufferAllocateInfo allocInfo = { VK_STRUCTURE_TYPE_COMMAND_BUFFER_ALLOCATE_INFO };
allocInfo.level = VK_COMMAND_BUFFER_LEVEL_PRIMARY; // ...

VkCommandBuffer commandBuffer;
vkAllocateCommandBuffers(device, &allocInfo, &commandBuffer);

VkCommandBufferBeginInfo beginInfo = { VK_STRUCTURE_TYPE_COMMAND_BUFFER_BEGIN_INFO };
beginInfo.flags = 0;
vkBeginCommandBuffer(commandBuffer, &beginInfo);
vkCmdBindDescriptorSets(commandBuffer, VK_PIPELINE_BIND_POINT_GRAPHICS, ...);
vkCmdDraw(commandBuffer, ...);
vkEndCommandBuffer(commandBuffer);

while (true) {
  // ...

  VkSubmitInfo submitInfo = {};
  submitInfo.sType = VK_STRUCTURE_TYPE_SUBMIT_INFO;
  submitInfo.commandBufferCount = 1;
  submitInfo.pCommandBuffers = &commandBuffer;
  vkQueueSubmit(queue, 1, &submitInfo, VK_NULL_HANDLE);
  // ...
}
```

Command Buffer Usage Modes:

- **Reuse** (submit multiple times)
- Single-use (submit once)
- Reset and re-record

Command Buffer Recording

```
VkCommandPoolCreateInfo poolCreateInfo = { VK_STRUCTURE_TYPE_COMMAND_POOL_CREATE_INFO };
poolCreateInfo.flags = 0; // ...

VkCommandBufferAllocateInfo allocInfo = { VK_STRUCTURE_TYPE_COMMAND_BUFFER_ALLOCATE_INFO };
allocInfo.level = VK_COMMAND_BUFFER_LEVEL_PRIMARY; // ...

VkCommandBuffer commandBuffer;
vkAllocateCommandBuffers(device, &allocInfo, &commandBuffer);

VkCommandBufferBeginInfo beginInfo = { VK_STRUCTURE_TYPE_COMMAND_BUFFER_BEGIN_INFO };
beginInfo.flags = 0;
vkBeginCommandBuffer(commandBuffer, &beginInfo);
vkCmdBindDescriptorSets(commandBuffer, VK_PIPELINE_BIND_POINT_GRAPHICS, ...);
vkCmdDraw(commandBuffer, ...);
vkEndCommandBuffer(commandBuffer);

while (true) {
  // ...

  VkSubmitInfo submitInfo = {};
  submitInfo.sType = VK_STRUCTURE_TYPE_SUBMIT_INFO;
  submitInfo.commandBufferCount = 1;
  submitInfo.pCommandBuffers = &commandBuffer;
  vkQueueSubmit(queue, 1, &submitInfo, VK_NULL_HANDLE);
  // ...
}
```

Command Buffer Usage Modes:

- Reuse (submit multiple times)
- **Single-use** (submit once)
- Reset and re-record

Command Buffer Recording

```
VkCommandPoolCreateInfo poolCreateInfo = { VK_STRUCTURE_TYPE_COMMAND_POOL_CREATE_INFO };
poolCreateInfo.flags = 0; // ...

VkCommandBufferAllocateInfo allocInfo = { VK_STRUCTURE_TYPE_COMMAND_BUFFER_ALLOCATE_INFO };
allocInfo.level = VK_COMMAND_BUFFER_LEVEL_PRIMARY; // ...

VkCommandBuffer commandBuffer;
vkAllocateCommandBuffers(device, &allocInfo, &commandBuffer);

VkCommandBufferBeginInfo beginInfo = { VK_STRUCTURE_TYPE_COMMAND_BUFFER_BEGIN_INFO };
beginInfo.flags = VK_COMMAND_BUFFER_USAGE_ONE_TIME_SUBMIT_BIT;
vkBeginCommandBuffer(commandBuffer, &beginInfo);
vkCmdBindDescriptorSets(commandBuffer, VK_PIPELINE_BIND_POINT_GRAPHICS, ...);
vkCmdDraw(commandBuffer, ...);
vkEndCommandBuffer(commandBuffer);

while (true) {
  // ...

  VkSubmitInfo submitInfo = {};
  submitInfo.sType = VK_STRUCTURE_TYPE_SUBMIT_INFO;
  submitInfo.commandBufferCount = 1;
  submitInfo.pCommandBuffers = &commandBuffer;
  vkQueueSubmit(queue, 1, &submitInfo, VK_NULL_HANDLE);
  // ...
}
```

⚠ REFACTORING IN PROGRESS...

Command Buffer Usage Modes:

- Reuse (submit multiple times)
- **Single-use** (submit once)
- Reset and re-record

Command Buffer Recording

```
VkCommandPoolCreateInfo poolCreateInfo = { VK_STRUCTURE_TYPE_COMMAND_POOL_CREATE_INFO };
poolCreateInfo.flags = 0; // ...

VkCommandBufferAllocateInfo allocInfo = { VK_STRUCTURE_TYPE_COMMAND_BUFFER_ALLOCATE_INFO };
allocInfo.level = VK_COMMAND_BUFFER_LEVEL_PRIMARY; // ...

VkCommandBuffer commandBuffer;
vkAllocateCommandBuffers(device, &allocInfo, &commandBuffer);

VkCommandBufferBeginInfo beginInfo = { VK_STRUCTURE_TYPE_COMMAND_BUFFER_BEGIN_INFO };
beginInfo.flags = VK_COMMAND_BUFFER_USAGE_ONE_TIME_SUBMIT_BIT;
vkBeginCommandBuffer(commandBuffer, &beginInfo);
vkCmdBindDescriptorSets(commandBuffer, VK_PIPELINE_BIND_POINT_GRAPHICS, ...);
vkCmdDraw(commandBuffer, ...);
vkEndCommandBuffer(commandBuffer);

while (true) {
  // ...

  VkSubmitInfo submitInfo = {};
  submitInfo.sType = VK_STRUCTURE_TYPE_SUBMIT_INFO;
  submitInfo.commandBufferCount = 1;
  submitInfo.pCommandBuffers = &commandBuffer;
  vkQueueSubmit(queue, 1, &submitInfo, VK_NULL_HANDLE);
  // ...
}
```

⚠ REFACTORING IN PROGRESS...

Command Buffer Usage Modes:

- Reuse (submit multiple times)
- **Single-use** (submit once)
- Reset and re-record

Command Buffer Recording

```
VkCommandPoolCreateInfo poolCreateInfo = { VK_STRUCTURE_TYPE_COMMAND_POOL_CREATE_INFO };
poolCreateInfo.flags = VK_COMMAND_POOL_CREATE_TRANSIENT_BIT; // ...

VkCommandBuffer
allocInfo.level

VkCommandBuffer
vkAllocateComma

VkCommandBuffer
beginInfo.flags
vkBeginCommandB
vkCmdBindDescri
vkCmdDraw(comma
vkEndCommandBuf

while (true) {
  // ...

  VkSubmitInfo submitInfo = {};
  submitInfo.sType = VK_STRUCTURE_TYPE_SUBMIT_INFO;
  submitInfo.commandBufferCount = 1;
  submitInfo.pCommandBuffers = &commandBuffer;
  vkQueueSubmit(queue, 1, &submitInfo, VK_NULL_HANDLE);
  // ...
}
```

> VK_COMMAND_POOL_CREATE_TRANSIENT_BIT specifies that command buffers allocated from the pool will be short-lived, meaning that they will be reset or freed in a relatively short timeframe. This flag **may** be used by the implementation to control memory allocation behavior within the pool.
>
> *The Khronos Group. Vulkan 1.2.200 Specificaton*

⚠ REFACTORING IN PROGRESS...

Command Buffer Usage Modes:

- Reuse (submit multiple times)
- **Single-use** (submit once)
- Reset and re-record

Command Buffer Recording

```
VkCommandPoolCreateInfo poolCreateInfo = { VK_STRUCTURE_TYPE_COMMAND_POOL_CREATE_INFO };
poolCreateInfo.flags = VK_COMMAND_POOL_CREATE_TRANSIENT_BIT; // ...

VkCommandBufferAllocateInfo allocInfo = { VK_STRUCTURE_TYPE_COMMAND_BUFFER_ALLOCATE_INFO };
allocInfo.level = VK_COMMAND_BUFFER_LEVEL_PRIMARY; // ...
```

```
VkCommandBuffer commandBuffer;
vkAllocateCommandBuffers(device, &allocInfo, &commandBuffer);

VkCommandBufferBeginInfo beginInfo = { VK_STRUCTURE_TYPE_COMMAND_BUFFER_BEGIN_INFO };
beginInfo.flags = VK_COMMAND_BUFFER_USAGE_ONE_TIME_SUBMIT_BIT;
vkBeginCommandBuffer(commandBuffer, &beginInfo);
vkCmdBindDescriptorSets(commandBuffer, VK_PIPELINE_BIND_POINT_GRAPHICS, ...);
vkCmdDraw(commandBuffer, ...);
vkEndCommandBuffer(commandBuffer);
```

```
while (true) {
  // ...

  VkSubmitInfo submitInfo = {};
  submitInfo.sType = VK_STRUCTURE_TYPE_SUBMIT_INFO;
  submitInfo.commandBufferCount = 1;
  submitInfo.pCommandBuffers = &commandBuffer;
  vkQueueSubmit(queue, 1, &submitInfo, VK_NULL_HANDLE);
  // ...
}
```

⚠ REFACTORING IN PROGRESS...

Command Buffer Usage Modes:

- Reuse (submit multiple times)
- **Single-use** (submit once)
- Reset and re-record

Command Buffer Recording

```
VkCommandPoolCreateInfo poolCreateInfo = { VK_STRUCTURE_TYPE_COMMAND_POOL_CREATE_INFO };
poolCreateInfo.flags = VK_COMMAND_POOL_CREATE_TRANSIENT_BIT; // ...

VkCommandBufferAllocateInfo allocInfo = { VK_STRUCTURE_TYPE_COMMAND_BUFFER_ALLOCATE_INFO };
allocInfo.level = VK_COMMAND_BUFFER_LEVEL_PRIMARY; // ...

while (true) {
  // ...

  VkCommandBuffer commandBuffer;
  vkAllocateCommandBuffers(device, &allocInfo, &commandBuffer);

  VkCommandBufferBeginInfo beginInfo = { VK_STRUCTURE_TYPE_COMMAND_BUFFER_BEGIN_INFO };
  beginInfo.flags = VK_COMMAND_BUFFER_USAGE_ONE_TIME_SUBMIT_BIT;
  vkBeginCommandBuffer(commandBuffer, &beginInfo);
  vkCmdBindDescriptorSets(commandBuffer, VK_PIPELINE_BIND_POINT_GRAPHICS, ...);
  vkCmdDraw(commandBuffer, ...);
  vkEndCommandBuffer(commandBuffer);

  VkSubmitInfo submitInfo = {};
  submitInfo.sType = VK_STRUCTURE_TYPE_SUBMIT_INFO;
  submitInfo.commandBufferCount = 1;
  submitInfo.pCommandBuffers = &commandBuffer;
  vkQueueSubmit(queue, 1, &submitInfo, VK_NULL_HANDLE);
  // ...
}
```

☑ REFACTORING DONE

Command Buffer Usage Modes:

- Reuse (submit multiple times)
- **Single-use** (submit once)
- Reset and re-record

Command Buffer Recording

```
VkCommandPoolCreateInfo poolCreateInfo = { VK_STRUCTURE_TYPE_COMMAND_POOL_CREATE_INFO };
poolCreateInfo.flags = VK_COMMAND_POOL_CREATE_TRANSIENT_BIT; // ...

VkCommandBufferAllocateInfo allocInfo = { VK_STRUCTURE_TYPE_COMMAND_BUFFER_ALLOCATE_INFO };
allocInfo.level = VK_COMMAND_BUFFER_LEVEL_PRIMARY; // ...

while (true) {
  // ...

  VkCommandBuffer commandBuffer;
  vkAllocateCommandBuffers(device, &allocInfo, &commandBuffer);

  VkCommandBufferBeginInfo beginInfo = { VK_STRUCTURE_TYPE_COMMAND_BUFFER_BEGIN_INFO };
  beginInfo.flags = VK_COMMAND_BUFFER_USAGE_ONE_TIME_SUBMIT_BIT;
  vkBeginCommandBuffer(commandBuffer, &beginInfo);
  vkCmdBindDescriptorSets(commandBuffer, VK_PIPELINE_BIND_POINT_GRAPHICS, ...);
  vkCmdDraw(commandBuffer, ...);
  vkEndCommandBuffer(commandBuffer);

  VkSubmitInfo submitInfo = {};
  submitInfo.sType = VK_STRUCTURE_TYPE_SUBMIT_INFO;
  submitInfo.commandBufferCount = 1;
  submitInfo.pCommandBuffers = &commandBuffer;
  vkQueueSubmit(queue, 1, &submitInfo, VK_NULL_HANDLE);
  // ...
}
```

⚠ REFACTORING IN PROGRESS...

Command Buffer Usage Modes:

- Reuse (submit multiple times)
- Single-use (submit once)
- **Reset and re-record**

Command Buffer Recording

```
VkCommandPoolCreateInfo poolCreateInfo = { VK_STRUCTURE_TYPE_COMMAND_POOL_CREATE_INFO };
poolCreateInfo.flags = VK_COMMAND_POOL_CREATE_TRANSIENT_BIT | VK_COMMAND_POOL_CREATE_RESET_COMMAND_BUFFER_BIT;

VkCommandBufferAllocateInfo allocInfo = { VK_STRUCTURE_TYPE_COMMAND_BUFFER_ALLOCATE_INFO };

while (t
  // ...

  VkComm
  vkAllo

  VkComm
  beginI
  vkBegi
  vkCmdBindDescriptorSets(commandBuffer, VK_PIPELINE_BIND_POINT_GRAPHICS, ...);
  vkCmdDraw(commandBuffer, ...);
  vkEndCommandBuffer(commandBuffer);

  VkSubmitInfo submitInfo = {};
  submitInfo.sType = VK_STRUCTURE_TYPE_SUBMIT_INFO;
  submitInfo.commandBufferCount = 1;
  submitInfo.pCommandBuffers = &commandBuffer;
  vkQueueSubmit(queue, 1, &submitInfo, VK_NULL_HANDLE);
  // ...
}
```

> VK_COMMAND_POOL_CREATE_RESET_COMMAND_BUFFER_BIT allows any command buffer allocated from a pool to be individually reset to the initial state; either by calling vkResetCommandBuffer, or via the implicit reset when calling vkBeginCommandBuffer. If this flag is not set on a pool, then vkResetCommandBuffer **must** not be called for any command buffer allocated from that pool.
>
> *The Khronos Group. Vulkan 1.2.200 Specificaton*

Command Buffer Usage Modes:

- Reuse (submit multiple times)
- Single-use (submit once)
- **Reset and re-record**

⚠ REFACTORING IN PROGRESS...

Command Buffer Recording

```
VkCommandPoolCreateInfo poolCreateInfo = { VK_STRUCTURE_TYPE_COMMAND_POOL_CREATE_INFO };
poolCreateInfo.flags = VK_COMMAND_POOL_CREATE_TRANSIENT_BIT | VK_COMMAND_POOL_CREATE_RESET_COMMAND_BUFFER_BIT;

VkCommandBufferAllocateInfo allocInfo = { VK_STRUCTURE_TYPE_COMMAND_BUFFER_ALLOCATE_INFO };
allocInfo.level = VK_COMMAND_BUFFER_LEVEL_PRIMARY; // ...

while (true) {
  // ...

  VkCommandBuffer commandBuffer;
  vkAllocateCommandBuffers(device, &allocInfo, &commandBuffer);

  VkCommandBufferBeginInfo beginInfo = { VK_STRUCTURE_TYPE_COMMAND_BUFFER_BEGIN_INFO };
  beginInfo.flags = VK_COMMAND_BUFFER_USAGE_ONE_TIME_SUBMIT_BIT;
  vkBeginCommandBuffer(commandBuffer, &beginInfo);
  vkCmdBindDescriptorSets(commandBuffer, VK_PIPELINE_BIND_POINT_GRAPHICS, ...);
  vkCmdDraw(commandBuffer, ...);
  vkEndCommandBuffer(commandBuffer);

  VkSubmitInfo submitInfo = {};
  submitInfo.sType = VK_STRUCTURE_TYPE_SUBMIT_INFO;
  submitInfo.commandBufferCount = 1;
  submitInfo.pCommandBuffers = &commandBuffer;
  vkQueueSubmit(queue, 1, &submitInfo, VK_NULL_HANDLE);
  // ...
}
```

⚠ REFACTORING IN PROGRESS...

Command Buffer Usage Modes:

- Reuse (submit multiple times)
- Single-use (submit once)
- **Reset and re-record**

Command Buffer Recording

```
VkCommandPoolCreateInfo poolCreateInfo = { VK_STRUCTURE_TYPE_COMMAND_POOL_CREATE_INFO };
poolCreateInfo.flags = VK_COMMAND_POOL_CREATE_TRANSIENT_BIT | VK_COMMAND_POOL_CREATE_RESET_COMMAND_BUFFER_BIT;

VkCommandBufferAllocateInfo allocInfo = { VK_STRUCTURE_TYPE_COMMAND_BUFFER_ALLOCATE_INFO };
allocInfo.level = VK_COMMAND_BUFFER_LEVEL_PRIMARY; // ...

VkCommandBuffer commandBuffer;
vkAllocateCommandBuffers(device, &allocInfo, &commandBuffer);

while (true) {
  // ...
  VkCommandBufferBeginInfo beginInfo = { VK_STRUCTURE_TYPE_COMMAND_BUFFER_BEGIN_INFO };
  beginInfo.flags = VK_COMMAND_BUFFER_USAGE_ONE_TIME_SUBMIT_BIT;
  vkBeginCommandBuffer(commandBuffer, &beginInfo);
  vkCmdBindDescriptorSets(commandBuffer, VK_PIPELINE_BIND_POINT_GRAPHICS, ...);
  vkCmdDraw(commandBuffer, ...);
  vkEndCommandBuffer(commandBuffer);

  VkSubmitInfo submitInfo = {};
  submitInfo.sType = VK_STRUCTURE_TYPE_SUBMIT_INFO;
  submitInfo.commandBufferCount = 1;
  submitInfo.pCommandBuffers = &commandBuffer;
  vkQueueSubmit(queue, 1, &submitInfo, VK_NULL_HANDLE);
  // ...
}
```

☑ REFACTORING DONE

Command Buffer Usage Modes:

- Reuse (submit multiple times)
- Single-use (submit once)
- **Reset and re-record**

Command Buffer Recording

```
VkCommandPoolCreateInfo poolCreateInfo = { VK_STRUCTURE_TYPE_COMMAND_POOL_CREATE_INFO };
poolCreateInfo.flags = VK_COMMAND_POOL_CREATE_TRANSIENT_BIT | VK_COMMAND_POOL_CREATE_RESET_COMMAND_BUFFER_BIT;

VkCommandBufferAllocateInfo allocInfo = { VK_STRUCTURE_TYPE_COMMAND_BUFFER_ALLOCATE_INFO };

VkCommandBuffer commandBuffer;
vkAllocateCommandBuffers(device, &allocInfo, &commandBuffer);

while (true) {
  // ...
  VkCommandBufferBeginInfo beginInfo = {};
  beginInfo.sType = VK_STRUCTURE_TYPE_COMMAND_BUFFER_BEGIN_INFO;
  vkBeginCommandBuffer(commandBuffer, &beginInfo);
  vkCmdBindDescriptorSets(commandBuffer, VK_PIPELINE_BIND_POINT_GRAPHICS, ...);
  vkCmdDraw(commandBuffer, ...);
  vkEndCommandBuffer(commandBuffer);

  VkSubmitInfo submitInfo = {};
  submitInfo.sType = VK_STRUCTURE_TYPE_SUBMIT_INFO;
  submitInfo.commandBufferCount = 1;
  submitInfo.pCommandBuffers = &commandBuffer;
  vkQueueSubmit(queue, 1, &submitInfo, VK_NULL_HANDLE);
  // ...
}
```

> VK_COMMAND_POOL_CREATE_RESET_COMMAND_BUFFER_BIT allows any command buffer allocated from a pool to be individually reset to the initial state; either by calling vkResetCommandBuffer, or via the implicit reset when calling vkBeginCommandBuffer. If this flag is not set on a pool, then vkResetCommandBuffer **must** not be called for any command buffer allocated from that pool.
>
> *The Khronos Group. Vulkan 1.2.200 Specificaton*

Command Buffer Usage Modes:

- Reuse (submit multiple times)
- Single-use (submit once)
- **Reset and re-record**

Command Buffer Recording

```
VkCommandPoolCreateInfo poolCreateInfo = { VK_STRUCTURE_TYPE_COMMAND_POOL_CREATE_INFO };
poolCreateInfo.flags = VK_COMMAND_POOL_CREATE_TRANSIENT_BIT | VK_COMMAND_POOL_CREATE_RESET_COMMAND_BUFFER_BIT;

VkCommandBufferAllocateInfo allocInfo = { VK_STRUCTURE_TYPE_COMMAND_BUFFER_ALLOCATE_INFO };

VkCommandBuffer commandBuffer;
vkAllocateCommandBuffers(device, &allocInfo, &commandBuffer);

while (true) {
  // ...
  VkCommandBufferBeginInfo beginInfo = {};
  beginInfo.sType = VK_STRUCTURE_TYPE_COMMAND_BUFFER_BEGIN_INFO;
  vkBeginCommandBuffer(commandBuffer, &beginInfo);
  vkCmdBindDescriptorSets(commandBuffer, VK_PIPELINE_BIND_POINT_GRAPHICS, ...);
  vkCmdDraw(commandBuffer, ...);
  vkEndCommandBuffer(commandBuffer);

  VkSubmitInfo submitInfo = {};
  submitInfo.sType = VK_STRUCTURE_TYPE_SUBMIT_INFO;
  submitInfo.commandBufferCount = 1;
  submitInfo.pCommandBuffers = &commandBuffer;
  vkQueueSubmit(queue, 1, &submitInfo, VK_NULL_HANDLE);
  // ...
}
```

> VK_COMMAND_POOL_CREATE_RESET_COMMAND_BUFFER_BIT allows any command buffer allocated from a pool to be individually reset to the initial state; either by calling vkResetCommandBuffer, or via the implicit reset when calling vkBeginCommandBuffer. If this flag is not set on a pool, then vkResetCommandBuffer **must** not be called for any command buffer allocated from that pool.
>
> *The Khronos Group. Vulkan 1.2.200 Specificaton*

Command Buffer Usage Modes:

- Reuse (submit multiple times)
- Single-use (submit once)
- **Reset and re-record**

Command Buffer Recording

```
VkCommandPoolCreateInfo poolCreateInfo = { VK_STRUCTURE_TYPE_COMMAND_POOL_CREATE_INFO };
poolCreateInfo.flags = VK_COMMAND_POOL_CREATE_TRANSIENT_BIT | VK_COMMAND_POOL_CREATE_RESET_COMMAND_BUFFER_BIT;

VkCommandBufferAllocateInfo allocInfo = { VK_STRUCTURE_TYPE_COMMAND_BUFFER_ALLOCATE_INFO };

VkCommandBuffer commandBuffer;
vkAllocateCommandBuffers(device, &allocInfo, &commandBuffer);

while (true) {
  // ...
  VkCommandBufferBeginInfo beginInfo = {};
  beginInfo.sType = VK_STRUCTURE_TYPE_COMMAND_BUFFER_BEGIN_INFO;
  vkBeginCommandBuffer(commandBuffer, &beginInfo);
  vkCmdBindDescriptorSets(commandBuffer, VK_PIPELINE_BIND_POINT_GRAPHICS, ...);
  vkCmdDraw(commandBuffer, ...);
  vkEndCommandBuffer(commandBuffer);

  VkSubmitInfo submitInfo = {};
  submitInfo.sType = VK_STRUCTURE_TYPE_SUBMIT_INFO;
  submitInfo.commandBufferCount = 1;
  submitInfo.pCommandBuffers = &commandBuffer;
  vkQueueSubmit(queue, 1, &submitInfo, VK_NULL_HANDLE);
  // ...
}
```

> VK_COMMAND_POOL_CREATE_RESET_COMMAND_BUFFER_BIT allows any command buffer allocated from a pool to be individually reset to the initial state; either by calling vkResetCommandBuffer, or via the implicit reset when calling **vkBeginCommandBuffer**. If this flag is not set on a pool, then vkResetCommandBuffer **must** not be called for any command buffer allocated from that pool.
>
> *The Khronos Group. Vulkan 1.2.200 Specificaton*

Command Buffer Usage Modes:

- Reuse (submit multiple times)
- Single-use (submit once)
- **Reset and re-record**

Command Buffer Lifecycle

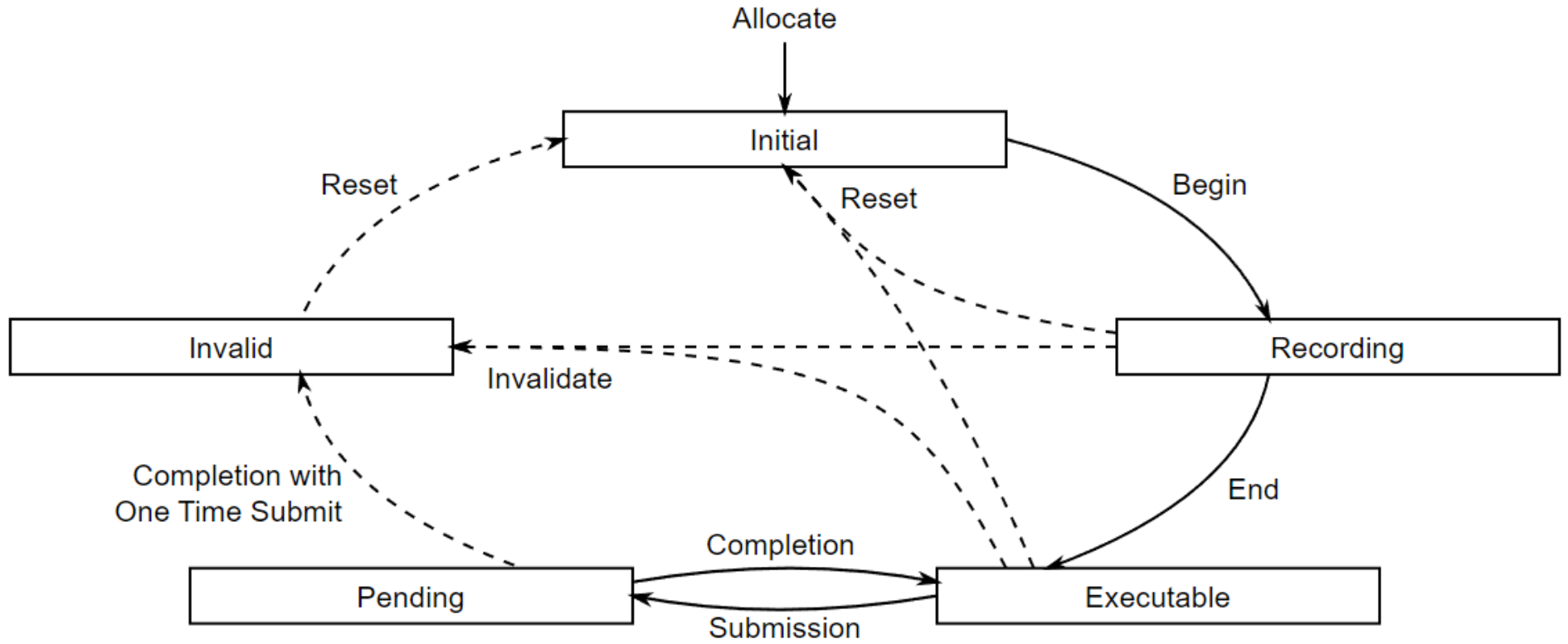


Figure 1. Lifecycle of a command buffer

- See: Chapter Command Buffer Lifecycle in the specification!
- Cleanup: vkFreeCommandBuffers, vkResetCommandPool, vkTrimCommandPool

Command Buffer Lifecycle

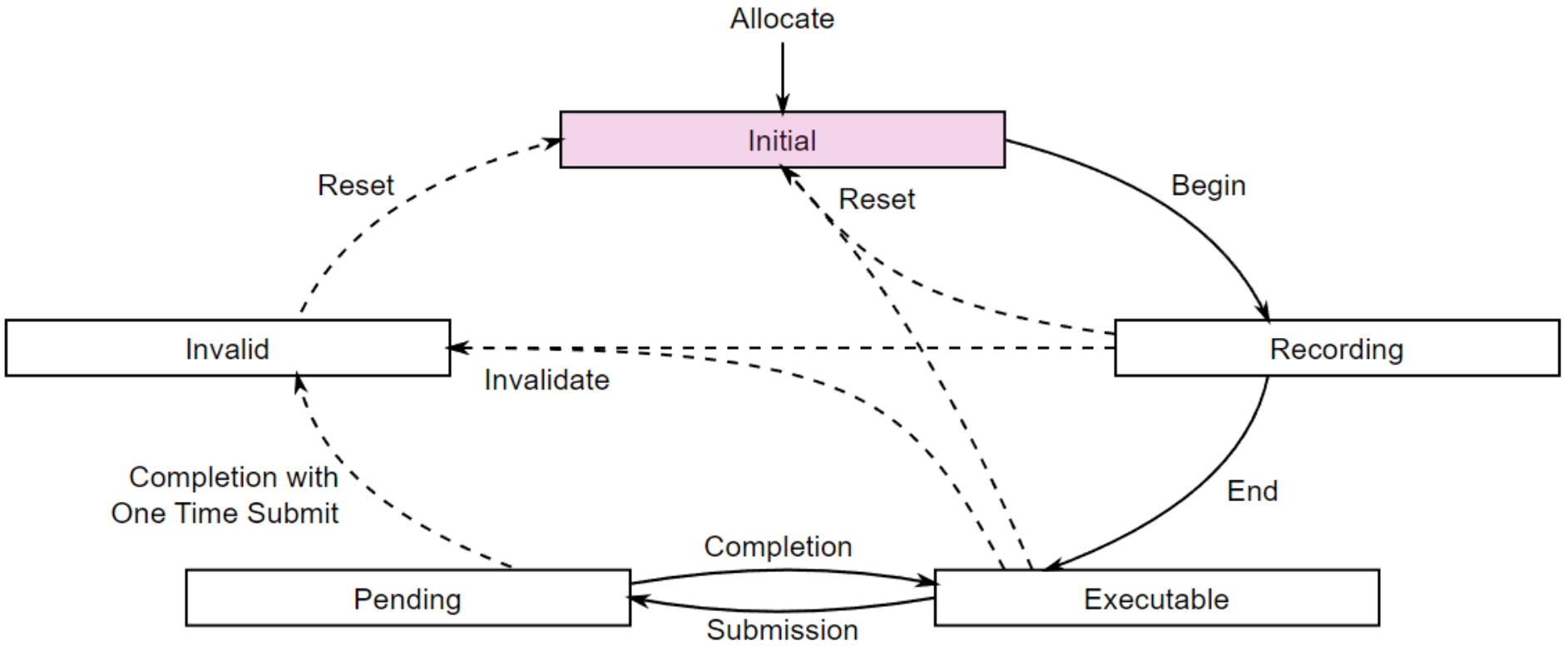


Figure 1. Lifecycle of a command buffer

- See: Chapter Command Buffer Lifecycle in the specification!
- Cleanup: vkFreeCommandBuffers, vkResetCommandPool, vkTrimCommandPool

Command Buffer Lifecycle

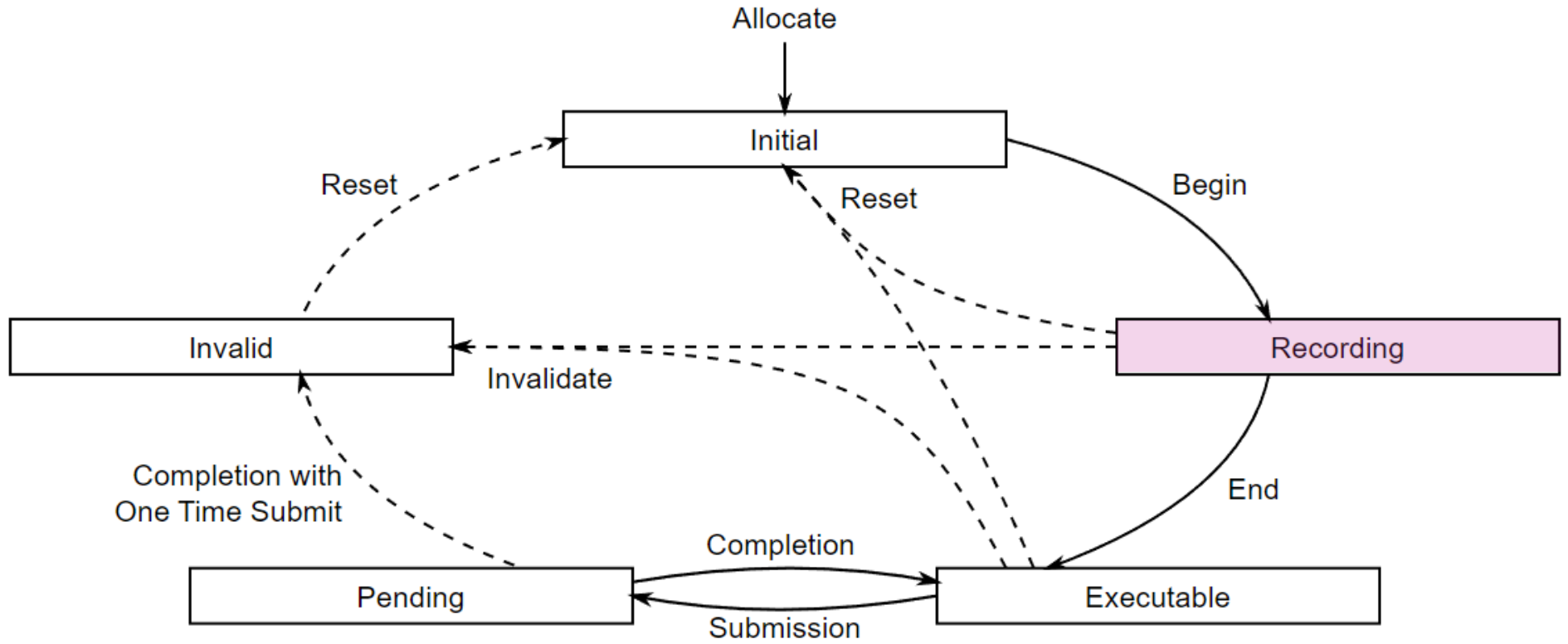


Figure 1. Lifecycle of a command buffer

- See: Chapter Command Buffer Lifecycle in the specification!
- Cleanup: vkFreeCommandBuffers, vkResetCommandPool, vkTrimCommandPool

Command Buffer Lifecycle

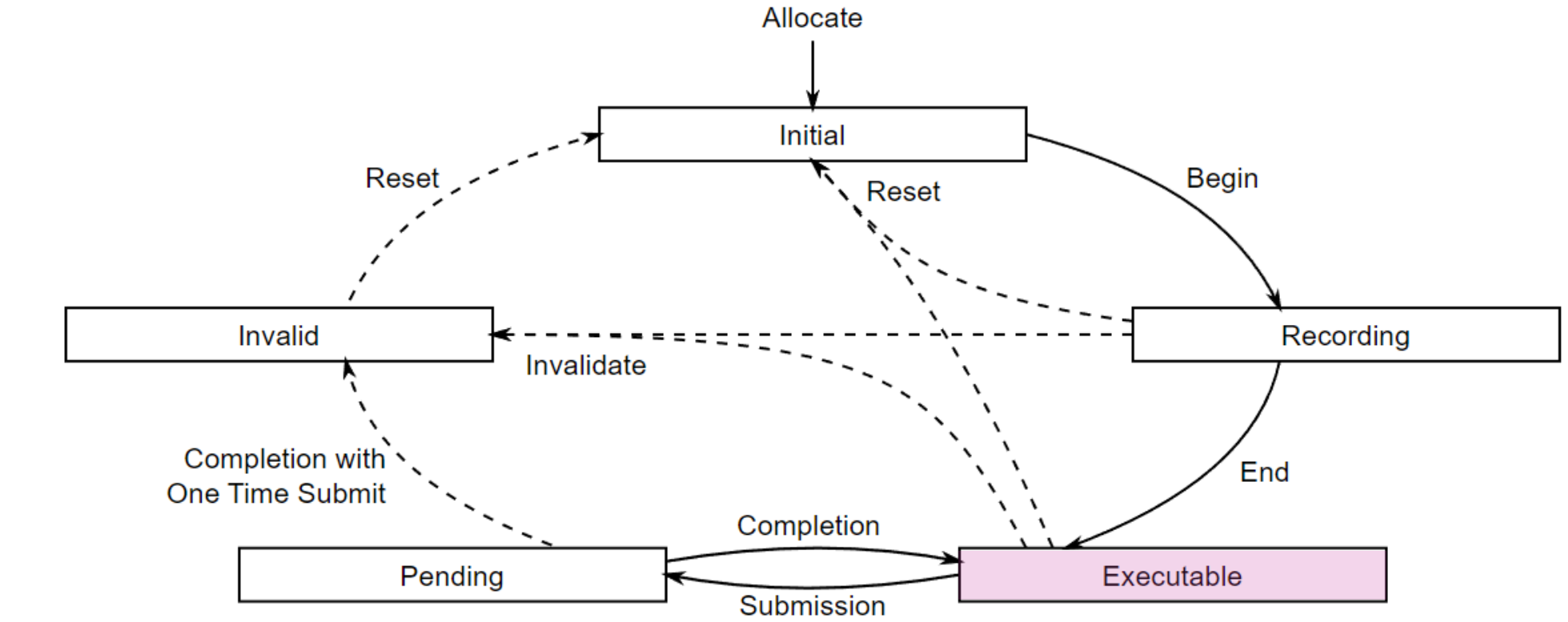


Figure 1. Lifecycle of a command buffer

- See: Chapter Command Buffer Lifecycle in the specification!
- Cleanup: vkFreeCommandBuffers, vkResetCommandPool, vkTrimCommandPool

Command Buffer Lifecycle

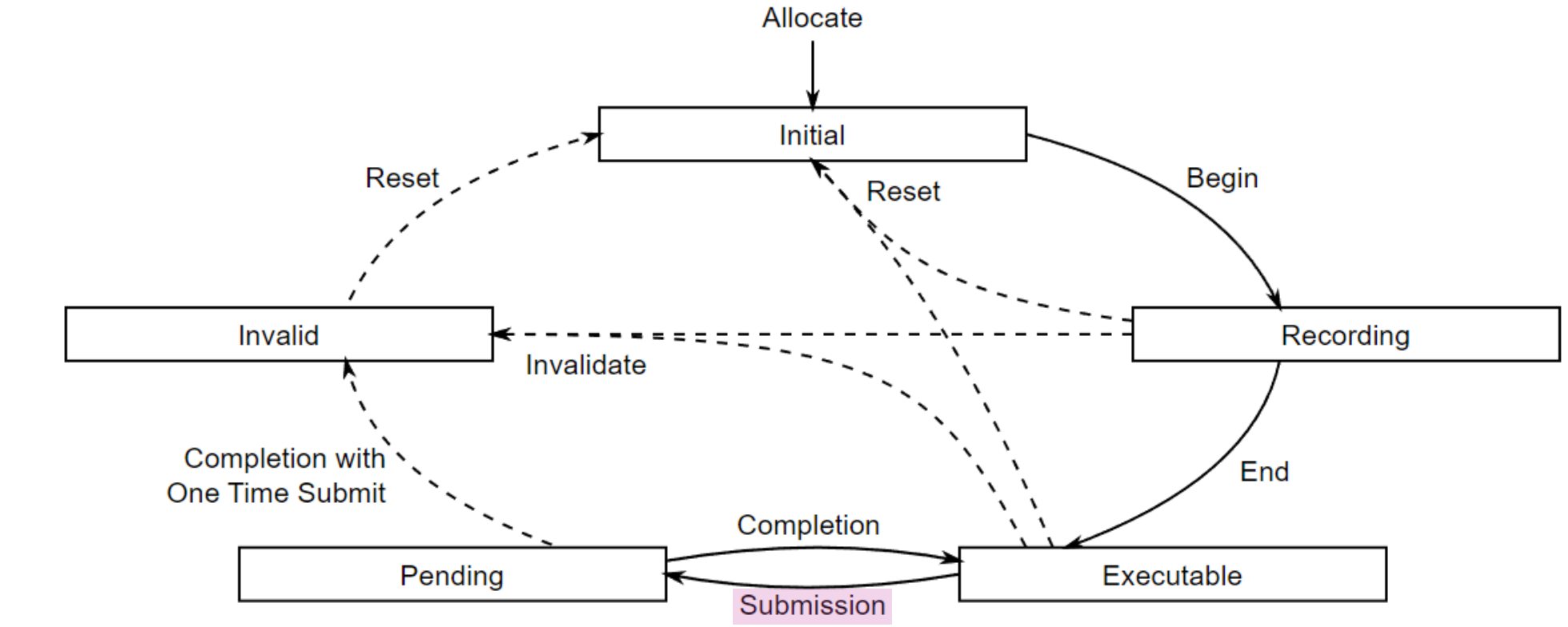


Figure 1. Lifecycle of a command buffer

- See: Chapter Command Buffer Lifecycle in the specification!
- Cleanup: vkFreeCommandBuffers, vkResetCommandPool, vkTrimCommandPool

Command Buffer Lifecycle

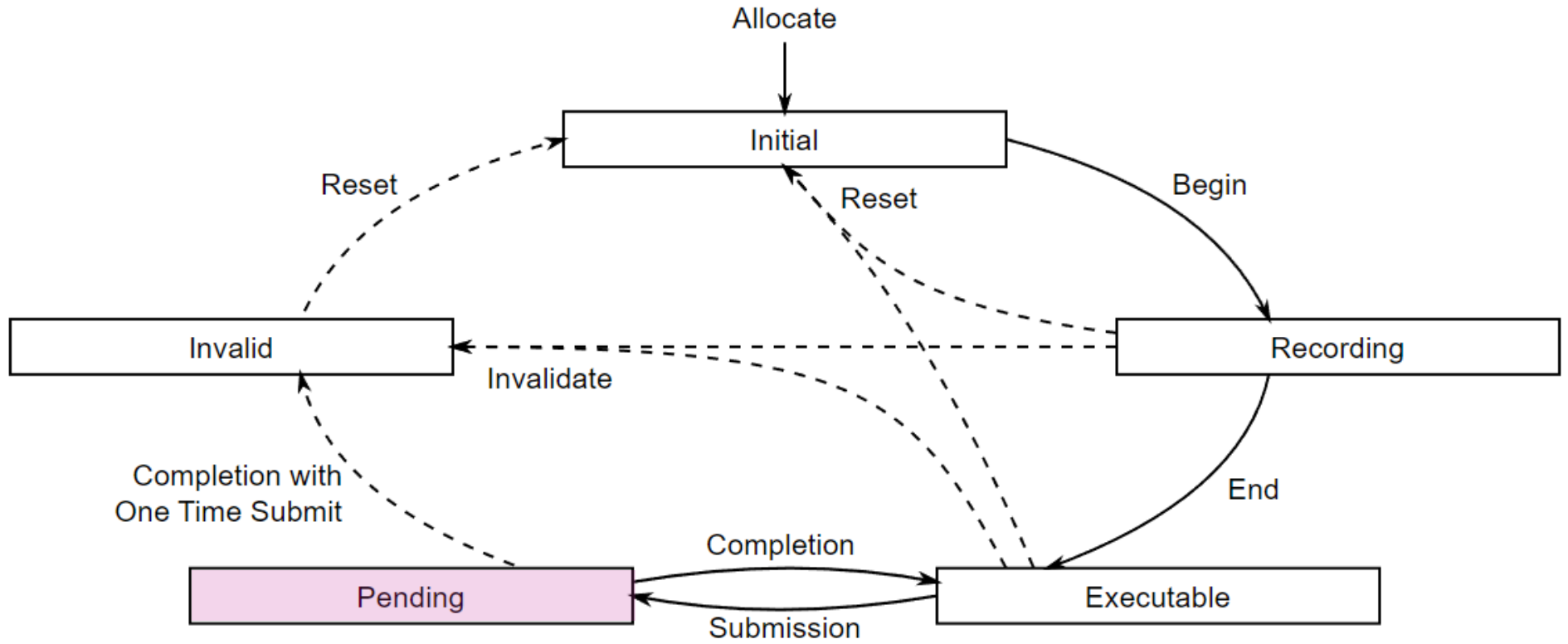


Figure 1. Lifecycle of a command buffer

- See: Chapter Command Buffer Lifecycle in the specification!
- Cleanup: vkFreeCommandBuffers, vkResetCommandPool, vkTrimCommandPool

Command Buffer Lifecycle

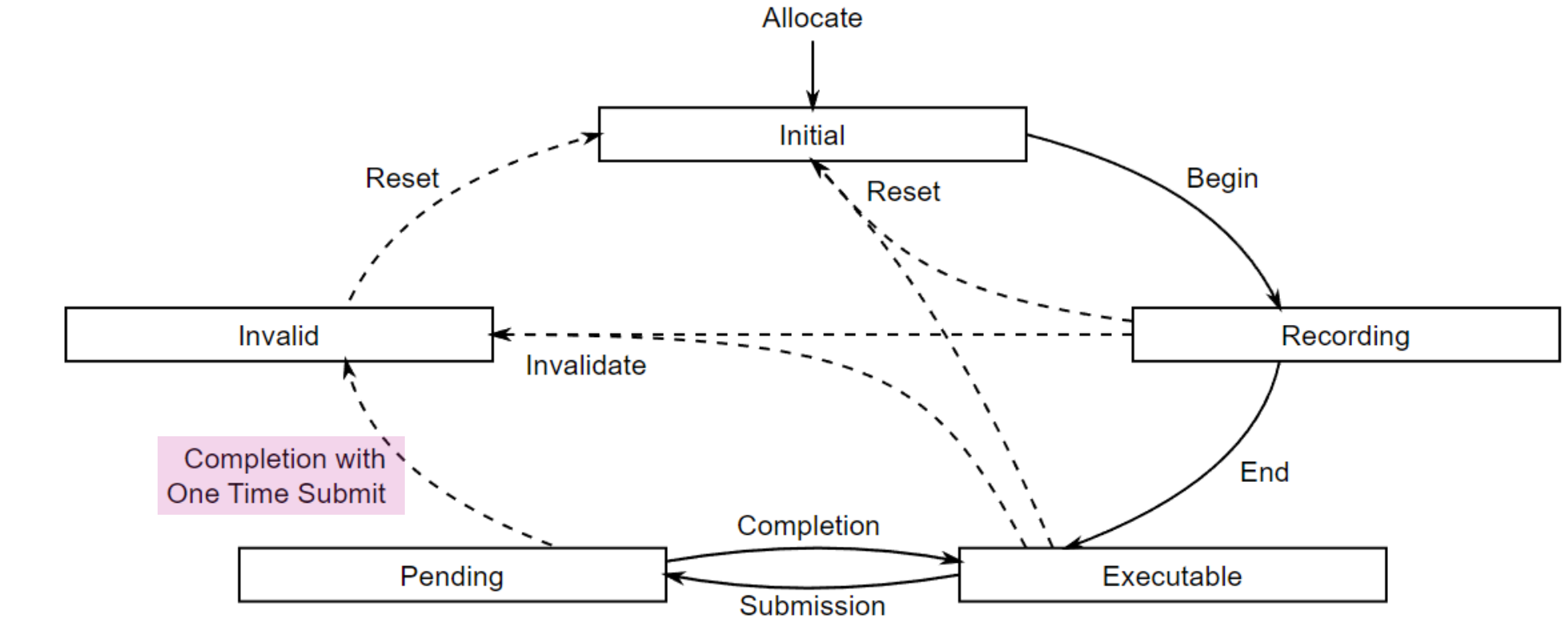


Figure 1. Lifecycle of a command buffer

- See: Chapter Command Buffer Lifecycle in the specification!
- Cleanup: vkFreeCommandBuffers, vkResetCommandPool, vkTrimCommandPool

Command Buffer Lifecycle

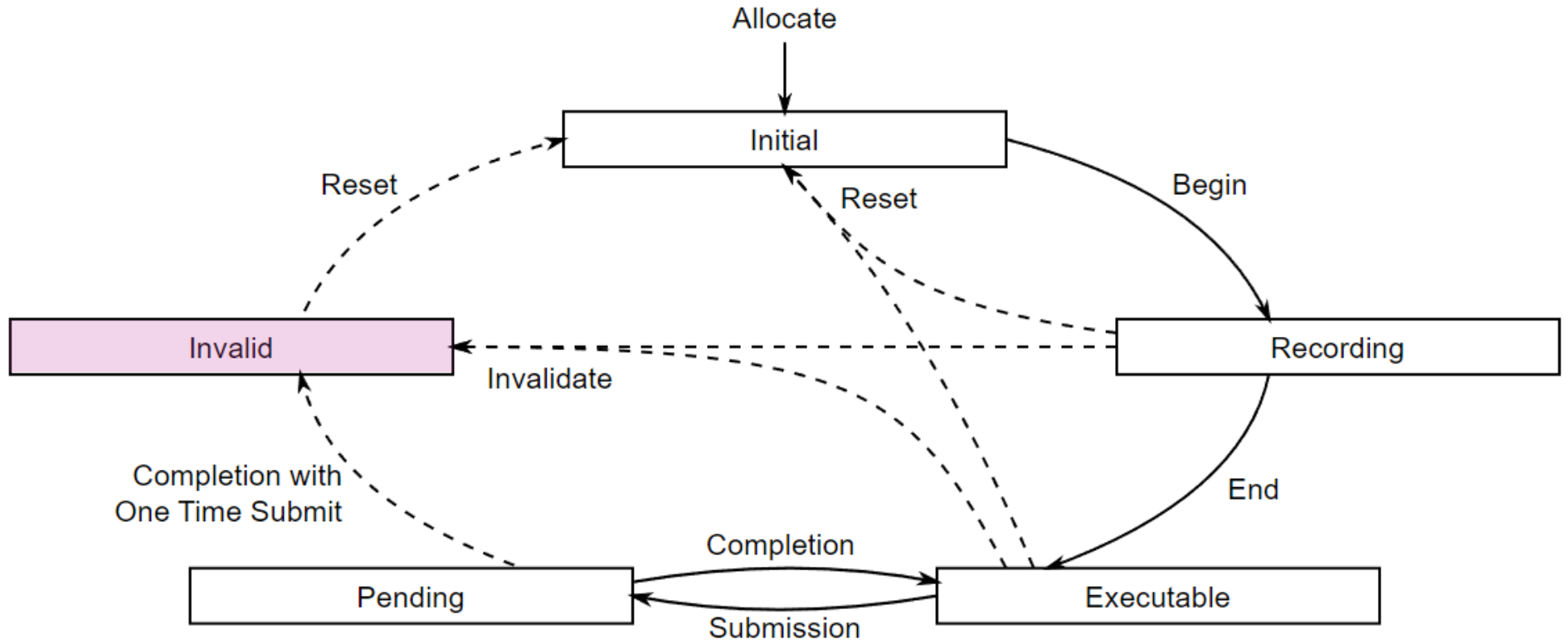


Figure 1. Lifecycle of a command buffer

- See: Chapter Command Buffer Lifecycle in the specification!
- Cleanup: vkFreeCommandBuffers, vkResetCommandPool, vkTrimCommandPool

Command Buffer Lifecycle

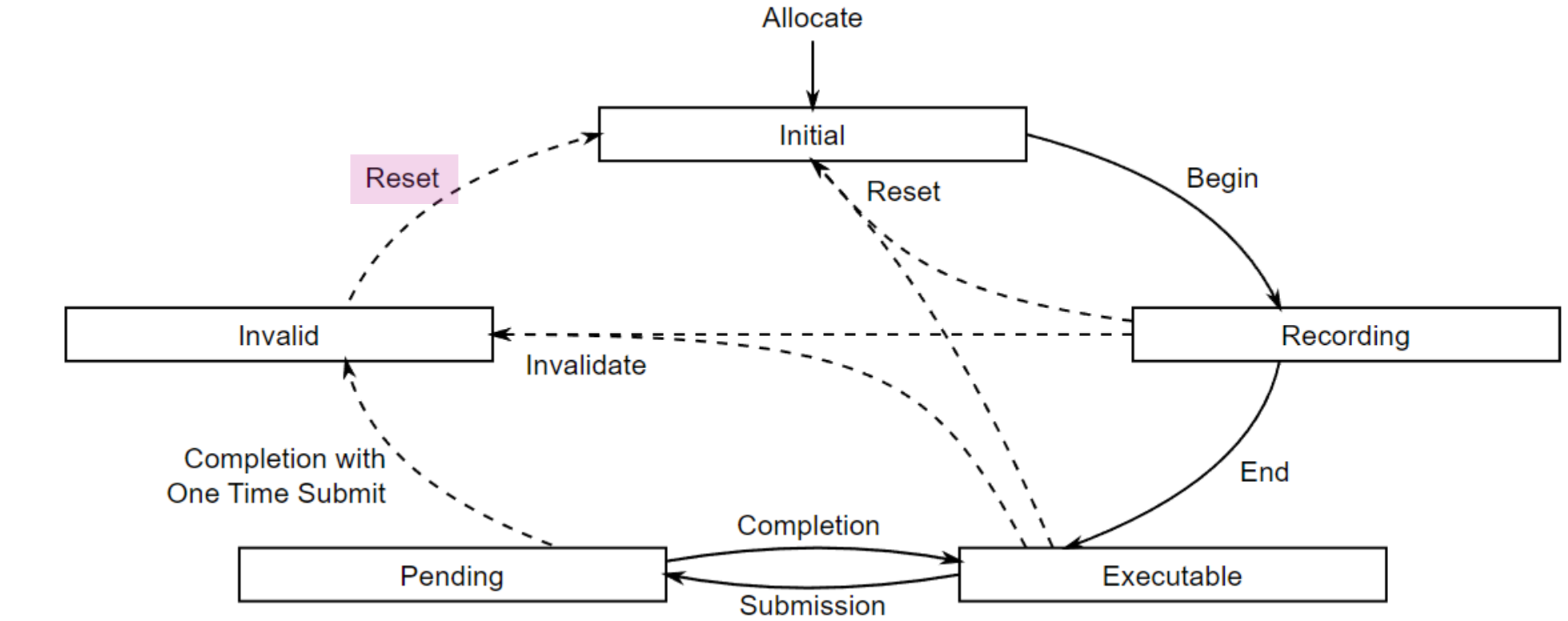


Figure 1. Lifecycle of a command buffer

- See: Chapter Command Buffer Lifecycle in the specification!
- Cleanup: vkFreeCommandBuffers, vkResetCommandPool, vkTrimCommandPool

Command Buffer Lifecycle

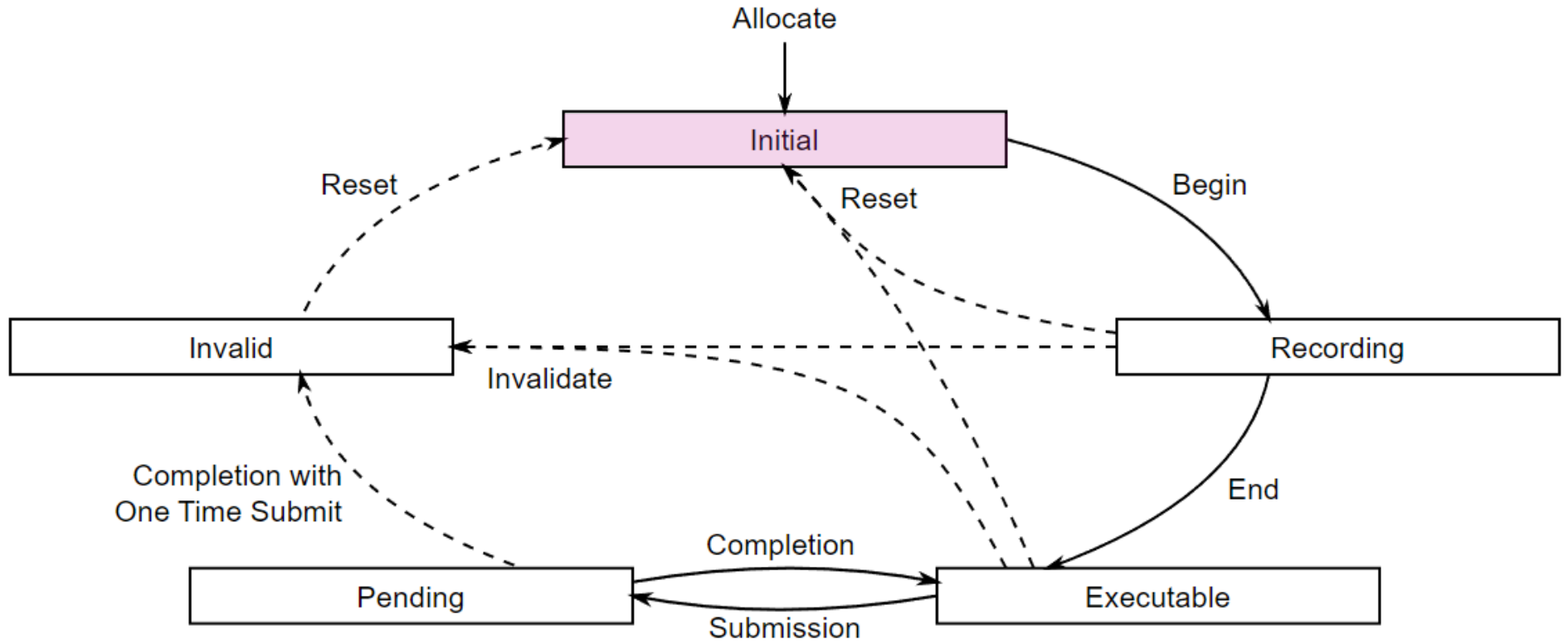


Figure 1. Lifecycle of a command buffer

- See: Chapter Command Buffer Lifecycle in the specification!
- Cleanup: vkFreeCommandBuffers, vkResetCommandPool, vkTrimCommandPool

Command Buffer Lifecycle

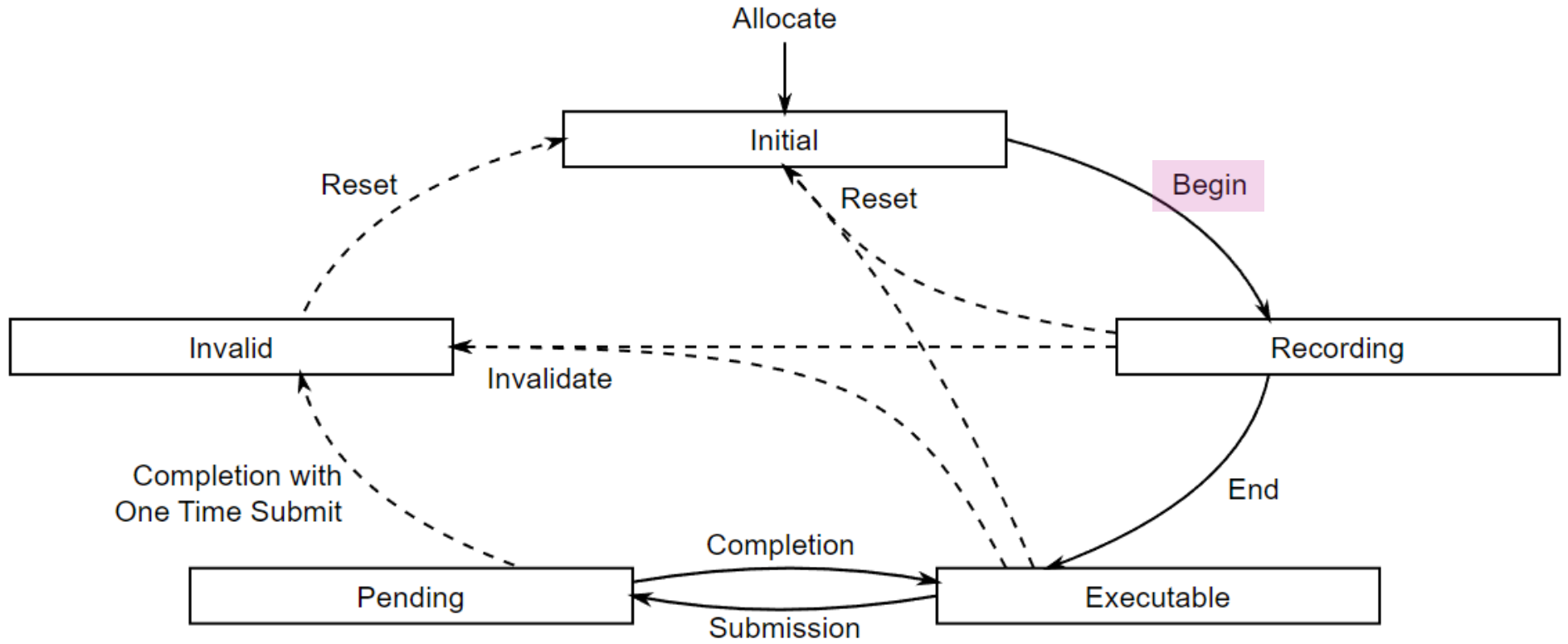
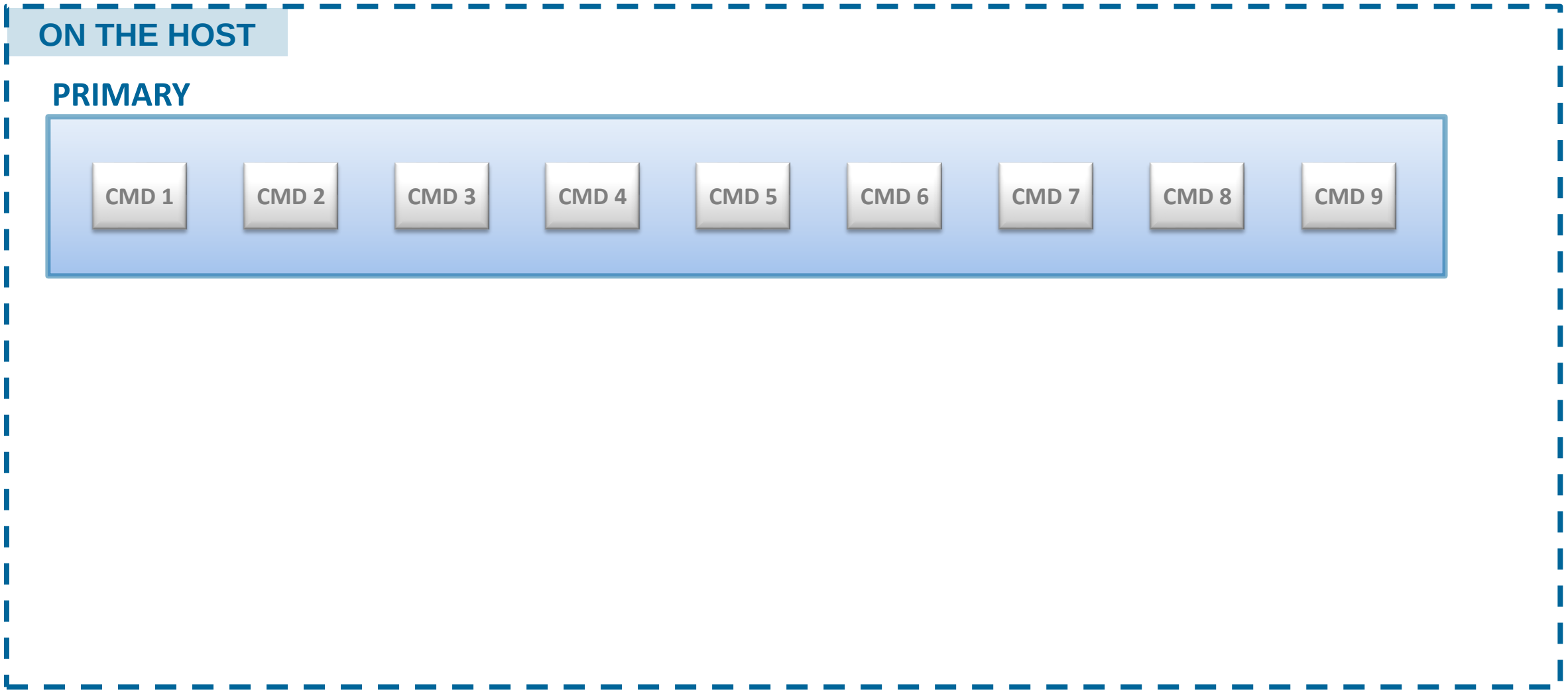


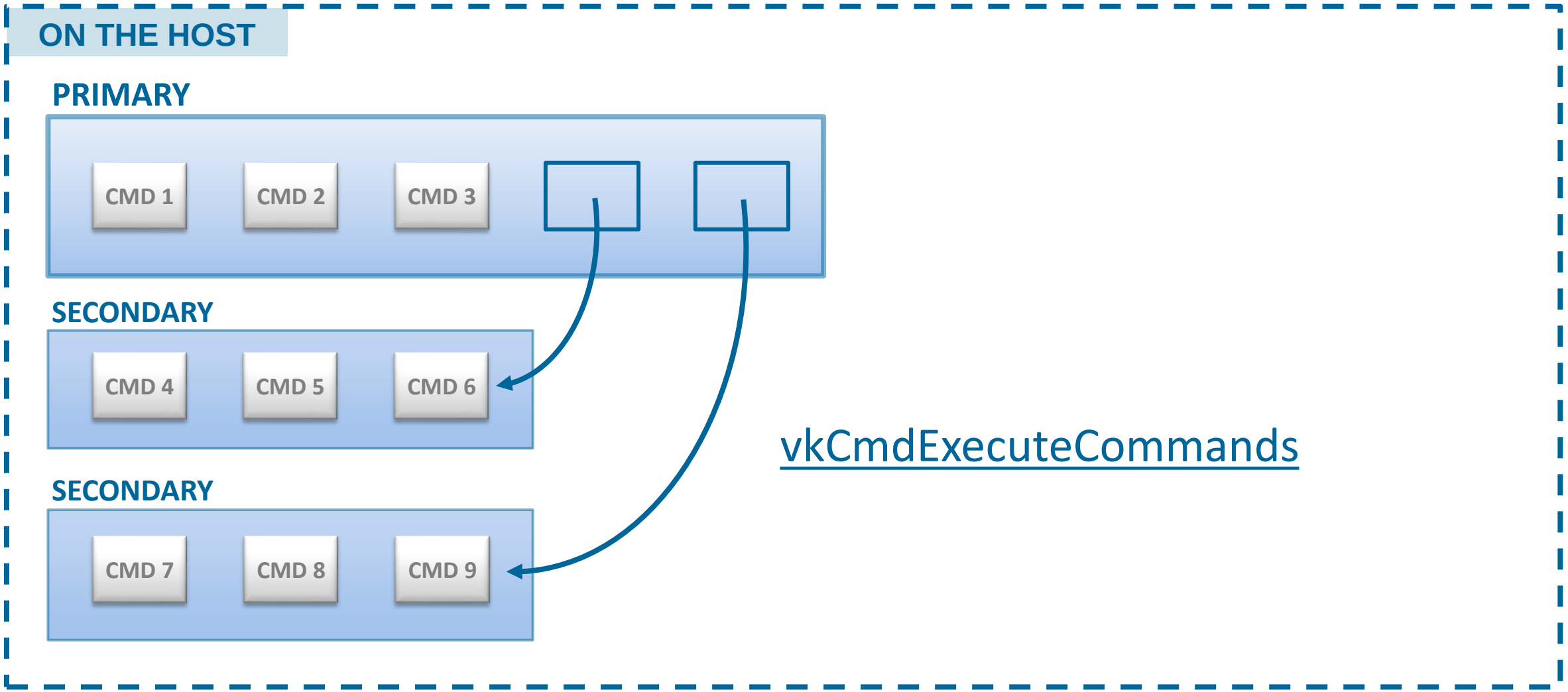
Figure 1. Lifecycle of a command buffer

- See: Chapter Command Buffer Lifecycle in the specification!
- Cleanup: vkFreeCommandBuffers, vkResetCommandPool, vkTrimCommandPool

Primary vs. Secondary Command Buffers



Primary vs. Secondary Command Buffers



Primary vs. Secondary Command Buffers

- In general: **Command buffers can be recorded on different threads**
- In general: **All command buffer state is local**
  See: 6. Command Buffers in the Vulkan specification
- Primary Command Buffers
  - Only these can be submitted to queues
- Secondary Command Buffers
  - Used from primary command buffers via vkCmdExecuteCommands
  - **Can** be advantageous in certain scenarios
  - Probably most useful with render passes
    See: VkCommandBufferInheritanceInfo
    Read: Render passes and secondary command buffers by Samsung Developers

Primary vs. Secondary Command Buffers

- In general: **Command buffers can be recorded on different threads**
- In general: **All command buffer state is local**

  See: 6. Command Buffers in the Vulkan specification

- Pri
  -
  -
  -
  -
- Se
  -
  -
  -

> Each command buffer manages state independently of other command buffers. There is no inheritance of state across primary and secondary command buffers, or between secondary command buffers. When a command buffer begins recording, all state in that command buffer is undefined. When secondary command buffer(s) are recorded to execute on a primary command buffer, the secondary command buffer inherits no state from the primary command buffer, and all state of the primary command buffer is undefined after an execute secondary command buffer command is recorded. There is one exception to this rule - if the primary command buffer is inside a render pass instance, then the render pass and subpass state is not disturbed by executing secondary command buffers.
>
> *The Khronos Group. Vulkan 1.2.200 Specificaton*

Read: Render passes and secondary command buffers by Samsung Developers

Outline

Command Types

Command Buffer Recording

Command Buffer Lifecycle and Types

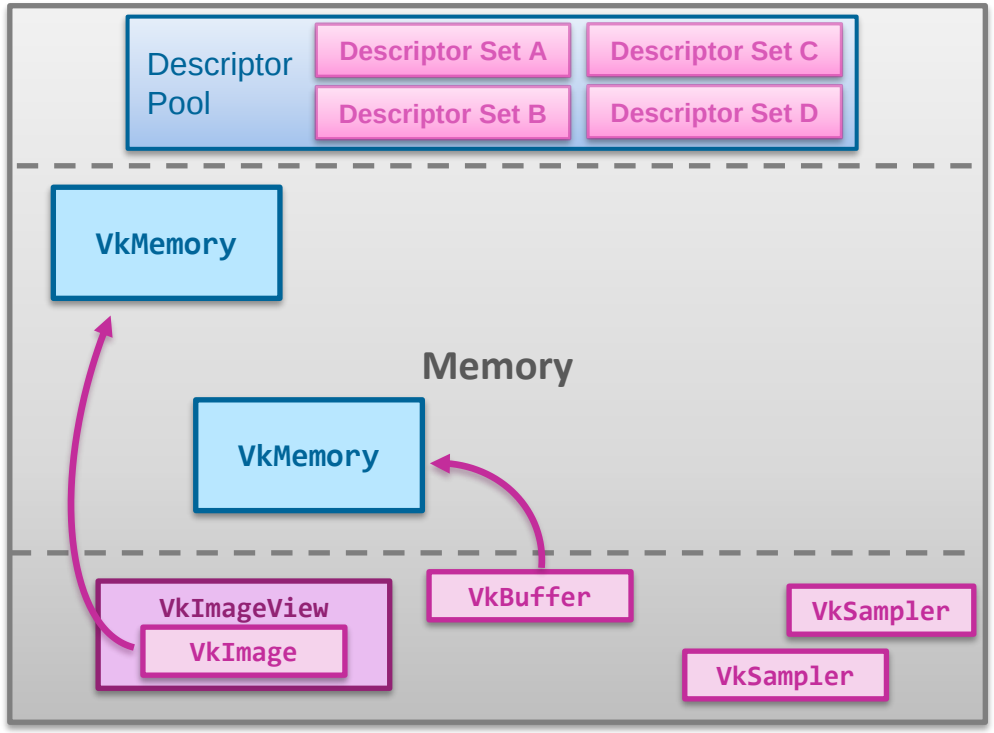
Providing Data to Commands

Data -> Command

- Descriptors
- Push Constants
- Parameters
- Attributes

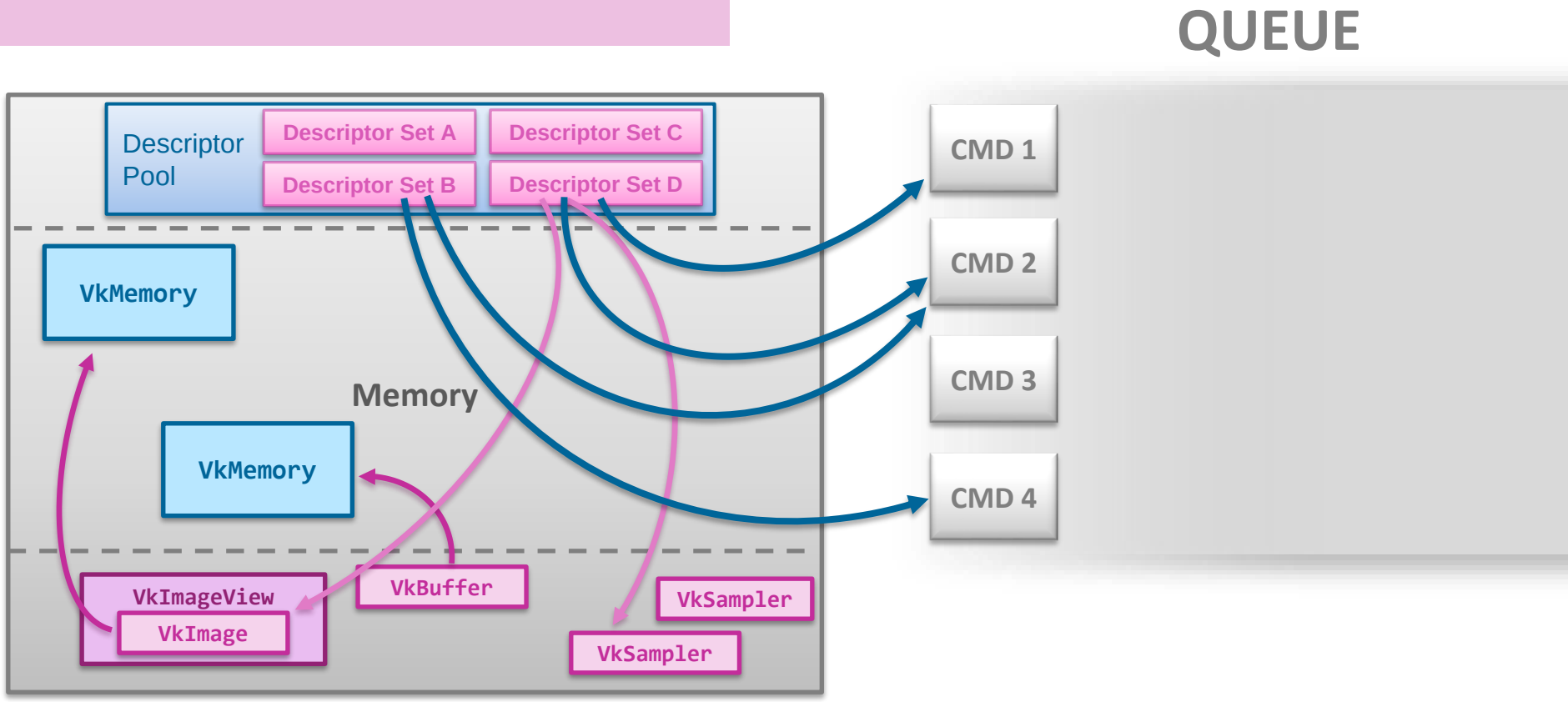
Descriptor Sets -> Command

- Descriptors
  - Establish links to resources via descriptors
  - See Episode 3
- Push Constants
- Parameters
- Attributes



Descriptor Sets -> Command

- Descriptors
  - Establish links to resources via descriptors
  - See Episode 3
- Push Constants
- Parameters
- Attributes

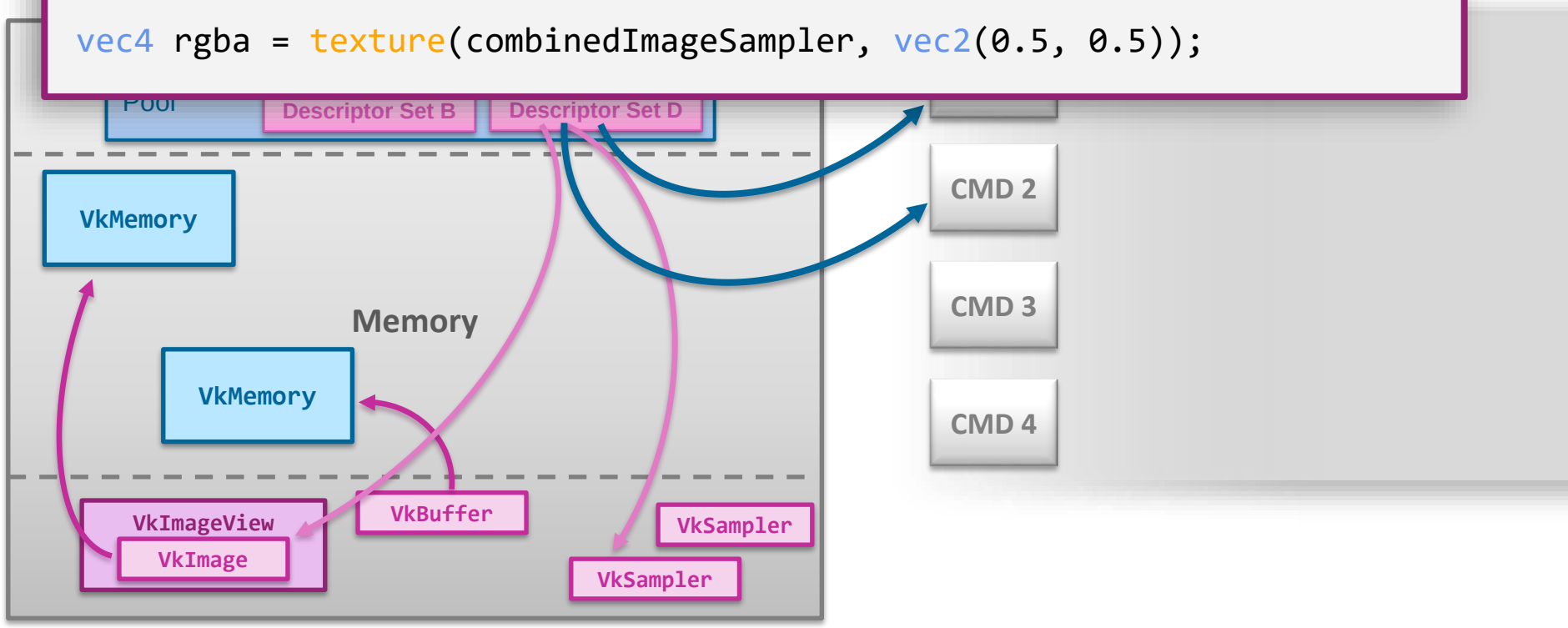


Descriptor Sets -> Command

- Descriptors
  - Establish links to
  - See Episode 3
- Push Constants
- Parameters
- Attributes

GLSL

```
#version 450

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

// ...

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



Push Constants -> Command

- Descriptors
  - Establish links to resources via descriptors
  - See Episode 3
- Push Constants
  - Data bundled with command buffer
  - Very limited in size
- Parameters
- Attributes

Push Constants -> Command

Graphics Pipeline Commands:

vkCmdDraw
vkCmdDrawIndexed
vkCmdDrawIndirect
vkCmdDrawIndirectCount
vkCmdDrawIndexedIndirect
vkCmdDrawIndexedIndirectCount

vkCmdDrawMeshTasksNV
vkCmdDrawMeshTasksIndirectNV
vkCmdDrawMeshTasksIndirectCountNV

vkCmdClearAttachments

Compute Pipeline Commands:

vkCmdDispatch
vkCmdDispatchBase
vkCmdDispatchIndirect

Ray Tracing Pipeline Commands:

vkCmdTraceRaysKHR
vkCmdTraceRaysIndirectKHR

Transfer Commands:

vkCmdCopyBuffer
vkCmdCopyImage
vkCmdCopyBufferToImage
vkCmdCopyImageToBuffer
vkCmdCopyAccelerationStructureKHR
vkCmdCopyAccelerationStructureToMemoryKHR
vkCmdCopyMemoryToAccelerationStructureKHR
vkCmdFillBuffer

vkCmdBlitImage

vkCmdResolveImage

vkCmdClearColorImage
vkCmdClearDepthStencilImage

Ray-Tracing Acceleration Structure Build Commands:

vkCmdBuildAccelerationStructuresKHR
vkCmdBuildAccelerationStructuresIndirectKHR

Bind Commands:

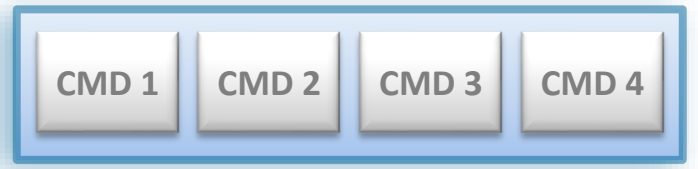
vkCmdBindDescriptorSets
vkCmdBindPipeline
vkCmdBindVertexBuffers
vkCmdBindIndexBuffer

Other Commands:

vkCmdPushConstants
vkCmdPushDescriptorSetKHR
vkCmdSetScissor
vkCmdSetViewport
vkCmdSetDepthBias
...

Push Constants -> Command

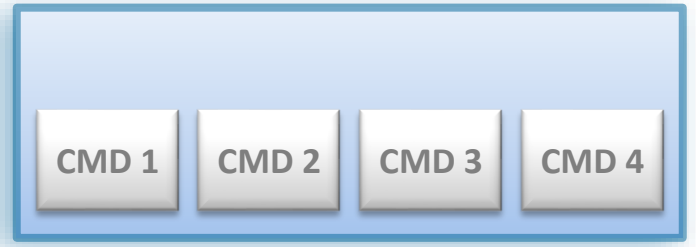
- Descriptors
  - Establish links to resources via descriptors
  - See Episode 3
- Push Constants 128 B 128 B 128 B
  - Data bundled with command buffer
  - Very limited in size
- Parameters
- Attributes



QUEUE

Push Constants -> Command

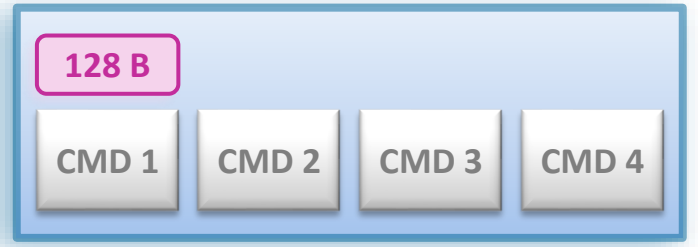
- Descriptors
  - Establish links to resources via descriptors
  - See Episode 3
- Push Constants 128 B 128 B 128 B
  - Data bundled with command buffer
  - Very limited in size
- Parameters
- Attributes



QUEUE

Push Constants -> Command

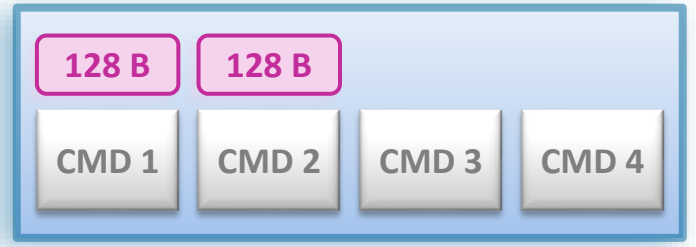
- Descriptors
  - Establish links to resources via descriptors
  - See Episode 3
- Push Constants 128 B 128 B
  - Data bundled with command buffer
  - Very limited in size
- Parameters
- Attributes



QUEUE

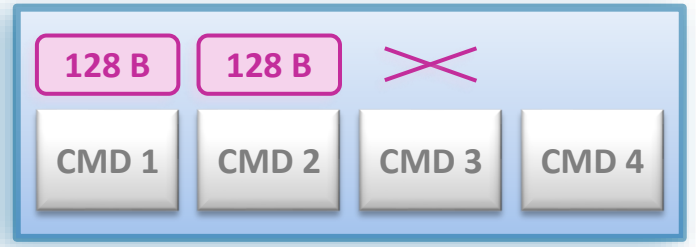
Push Constants -> Command

- Descriptors
  - Establish links to resources via descriptors
  - See Episode 3
- Push Constants 128 B
  - Data bundled with command buffer
  - Very limited in size
- Parameters
- Attributes



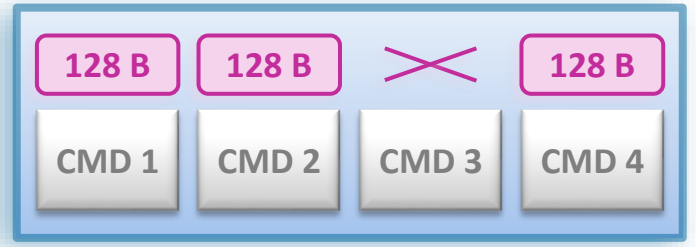
Push Constants -> Command

- Descriptors
  - Establish links to resources via descriptors
  - See Episode 3
- Push Constants 128 B
  - Data bundled with command buffer
  - Very limited in size
- Parameters
- Attributes



Push Constants -> Command

- Descriptors
  - Establish links to resources via descriptors
  - See Episode 3
- Push Constants
  - Data bundled with command buffer
  - Very limited in size
- Parameters
- Attributes

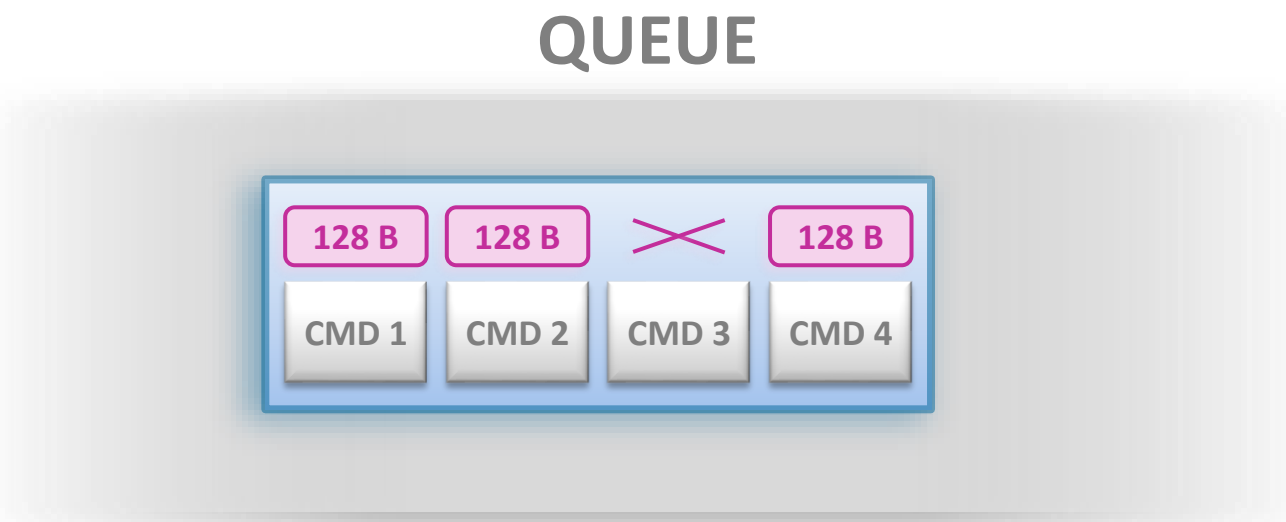


Push Constants -> Command

- Descriptors
  - Establish links to resources via descriptors
  - See Episode 3
- Push Constants
  - Data bundled with command buffer
  - Very limited in size
- Parameters
- Attributes

Relevant API structs/functions:

- VkPipelineLayoutCreateInfo
- VkPushConstantRange
- vkCmdPushConstants



Push Constants -> Command

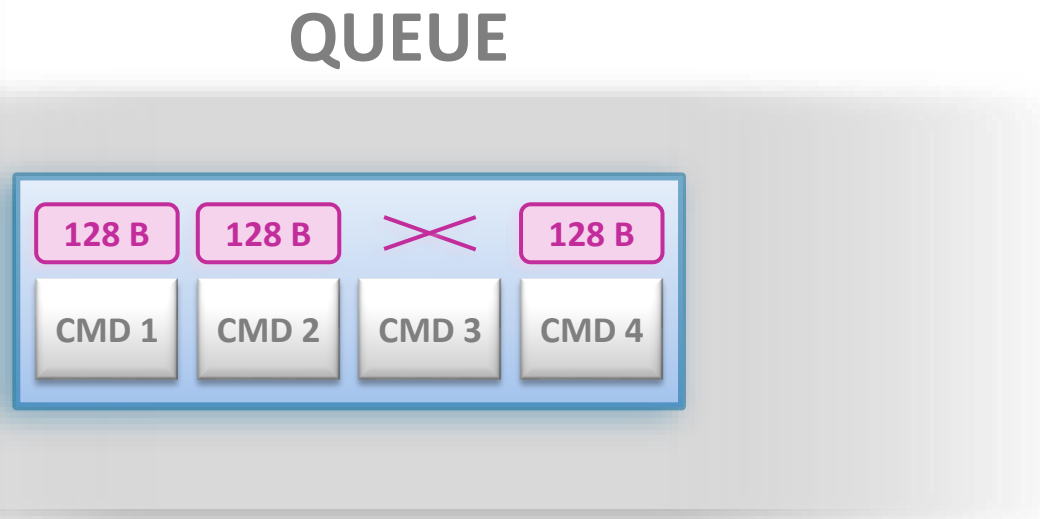
- Descriptors
  - Establish links to resources via descriptors
  - See Episode 3
- Push Constants
  - Data
  - Very
- Param
- Attribu

GLSL

```
#version 450

layout(push_constant) uniform PushConstants {
    vec4  color;
    mat4  matrix;
} pushConstants;

// ...

vec4 rgba = pushConstants.color;
mat4 M    = pushConstants.matrix;
```

Relevant API structs/functions:

- VkPipelineLayoutCreateInfo
- VkPushConstantRange
- vkCmdPushConstants



Parameters -> Command

- Descriptors
  - Establish links to resources via descriptors
  - See Episode 3
- Push Constants
  - Data bundled with command buffer
  - Very limited in size
- Parameters
- Attributes

Parameters -> Command

- Descriptors
  - Establish links to resources via descriptors
  - See Episode 3
- Push Constants
  - Data bundled with command buffer
  - Very limited in size
- Parameters
- Attributes

```
VkCommandBuffer commandBuffer = // ...
VkBuffer sourceBuffer = // ...
VkBuffer destinationBuffer = // ...

VkBufferCopy bufferCopy = {};
bufferCopy.srcOffset = VkDeviceSize{ 0 };
bufferCopy.dstOffset = VkDeviceSize{ 1024 };
bufferCopy.size = VkDeviceSize{ 1048576 };

vkCmdCopyBuffer(
    commandBuffer,
    sourceBuffer, destinationBuffer,
    1, &bufferCopy
);
```

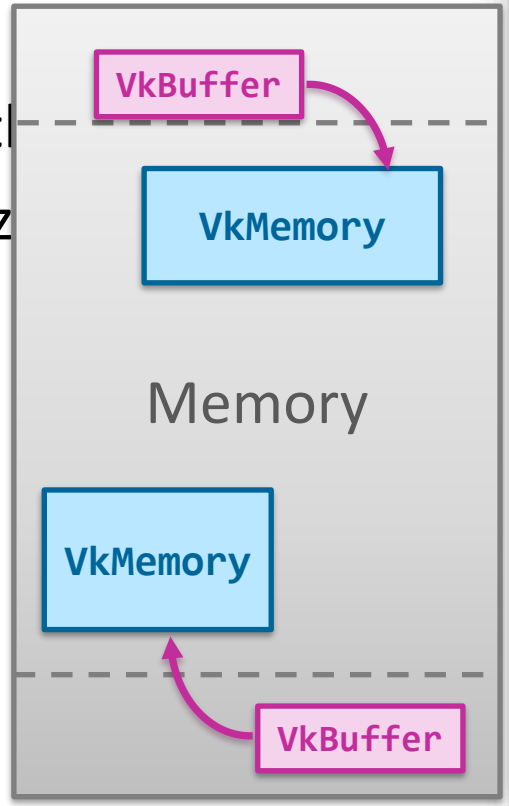
Parameters -> Command

- Descriptors
  - Establish links to resources via descriptors
  - See Episode 3
- Push Constants
  - Data bundled with command buffer
  - Very limited in size
- Parameters
- Attributes

```
VkCommandBuffer commandBuffer = // ...
VkBuffer sourceBuffer = // ...
VkBuffer destinationBuffer = // ...

VkBufferCopy bufferCopy = {};
bufferCopy.srcOffset = VkDeviceSize{ 0 };
bufferCopy.dstOffset = VkDeviceSize{ 1024 };
bufferCopy.size = VkDeviceSize{ 1048576 };

vkCmdCopyBuffer(
    commandBuffer,
    sourceBuffer, destinationBuffer,
    1, &bufferCopy
);
```

Parameters -> Command

- Descriptors
  - Establish links to resources via descriptors
  - See Episode 3
- Push Constants
  - Data bundled wit
  - Very limited in siz
- Parameters
- Attributes




```
VkCommandBuffer commandBuffer = // ...
VkBuffer sourceBuffer = // ...
VkBuffer destinationBuffer = // ...

VkBufferCopy bufferCopy = {};
bufferCopy.srcOffset = VkDeviceSize{ 0 };
bufferCopy.dstOffset = VkDeviceSize{ 1024 };
bufferCopy.size = VkDeviceSize{ 1048576 };

vkCmdCopyBuffer(
    commandBuffer,
    sourceBuffer, destinationBuffer,
    1, &bufferCopy
);
```

Parameters -> Command

- Descriptors
  - Establish links to resources via descriptors
  - See Episode 3
- Push Constants
  - Data bundled with command buffer
  - Very limited in size
- Parameters
- Attributes

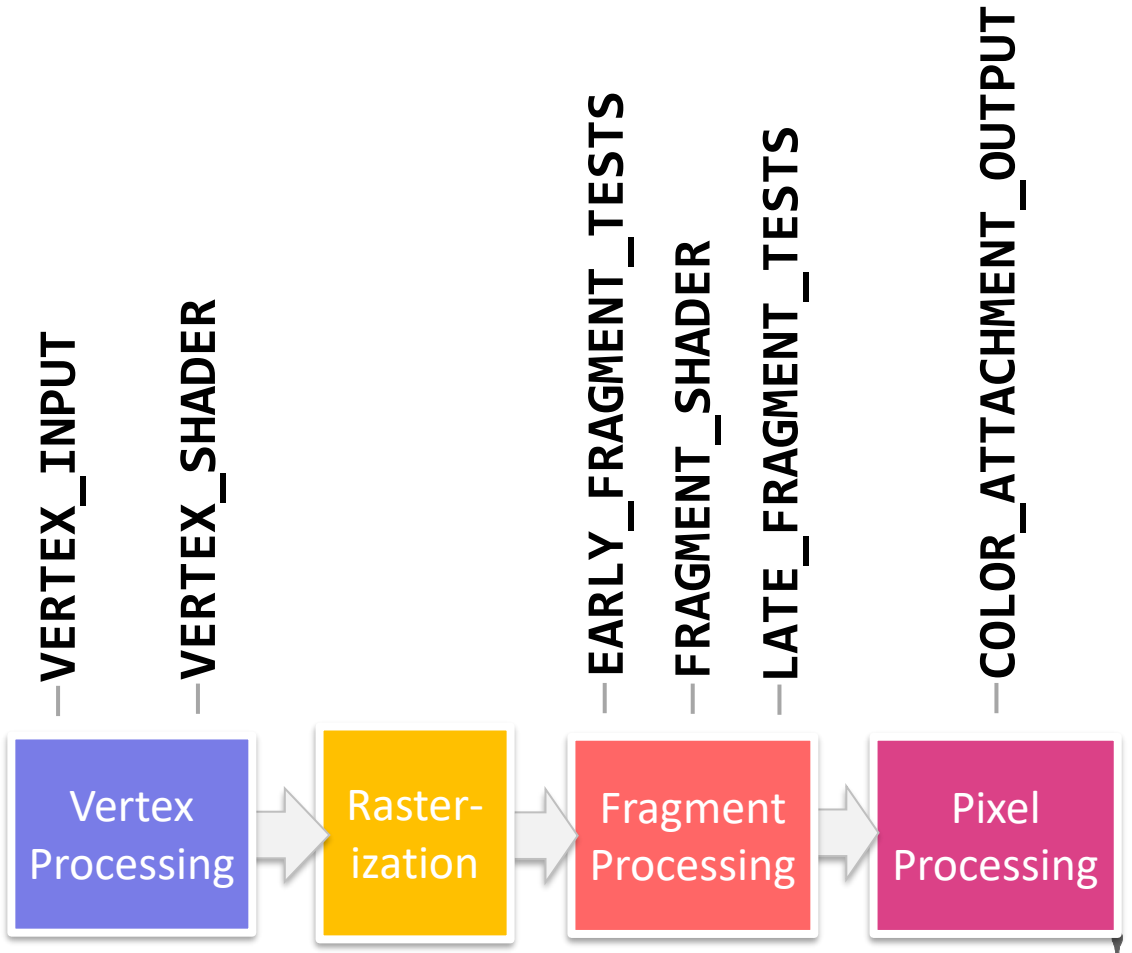
```
VkCommandBuffer commandBuffer = // ...
VkBuffer sourceBuffer = // ...
VkBuffer destinationBuffer = // ...

VkBufferCopy bufferCopy = {};
bufferCopy.srcOffset = VkDeviceSize{ 0 };
bufferCopy.dstOffset = VkDeviceSize{ 1024 };
bufferCopy.size = VkDeviceSize{ 1048576 };

vkCmdCopyBuffer(
    commandBuffer,
    sourceBuffer, destinationBuffer,
    1, &bufferCopy
);
```

Attributes -> Draw Call

- Descriptors
  - Establish links to resources via descriptors
  - See Episode 3
- Push Constants
  - Data bundled with command buffer
  - Very limited in size
- Parameters
- Attributes
  - Graphics pipelines only
  - Streamed to vertex shaders
  - Accessible via input location

Attributes -> Draw Call

- Descriptors
  - Establish links to resources via descriptors
  - See Episode 3
- Push Constants
  - Data bundled with command buffer
  - Very limited in size
- Parameters
- Attributes
  - Graphics pipelines only
  - Streamed to vertex shaders
  - Accessible via input location



Attributes -> Draw Call

- Descriptors
  - Establish links to resources via descriptors
  - See Episode 3
- Push Constants
  - Data bundled with command buffer
  - Very limited in size
- Parameters
- Attributes
  - Graphics pipelines only
  - Streamed to vertex shaders
  - Accessible via input location

GLSL

```
#version 450

layout (location = 0) in vec3 inPosition;

void main()
{
    gl_Position = vec4(inPosition, 1.0);
}
```

Attributes -> Draw Call

- Descriptors
  - Establish links to resources via descriptors
  - See Episode 3
- Push Constants
  - Data bundled with command buffer
  - Very limited in size
- Parameters
- Attributes
  - Graphics pipelines only
  - Streamed to vertex shaders
  - Accessible via input location

GLSL

```
#version 450

layout (location = 0) in vec3 inPosition;
layout (location = 1) in vec3 inNormal;
layout (location = 2) in vec2 inTexCoord;

void main()
{
    gl_Position = vec4(inPosition, 1.0);
}
```

Attributes -> Draw Call

GLSL

```
#version 450

layout (location = 0) in vec3 inPosition;

void main()
{
    gl_Position = vec4(inPosition, 1.0);
}
```


Attributes -> Draw Call

```
VkVertexInputBindingDescription binding0 = {};
binding0.binding = 0;
binding0.stride = sizeof(float) * 3;
binding0.inputRate = VK_VERTEX_INPUT_RATE_VERTEX;

VkVertexInputAttributeDescription attribute0 = {};
attribute0.location = 0;
attribute0.binding = 0;
attribute0.format = VK_FORMAT_R32G32B32_SFLOAT;
attribute0.offset = 0;

VkPipelineVertexInputStateCreateInfo vertexInputState = {};
vertexInputState.sType = VK_STRUCTURE_TYPE_PIPELINE_VERTEX_INPUT_STATE_CREATE_INFO;
vertexInputState.vertexBindingDescriptionCount = 1;
vertexInputState.pVertexBindingDescriptions = &binding0;
vertexInputState.vertexAttributeDescriptionCount = 1;
vertexInputState.pVertexAttributeDescriptions = &attribute0;

VkGraphicsPipelineCreateInfo graphicsPipeCreateInfo = {};
graphicsPipeCreateInfo.sType = VK_STRUCTURE_TYPE_GRAPHICS_PIPELINE_CREATE_INFO;
graphicsPipeCreateInfo.pVertexInputState = &vertexInputState;
// ...
```

GLSL

```
#version 450

layout (location = 0) in vec3 inPosition;

void main()
{
    gl_Position = vec4(inPosition, 1.0);
}
```

Attributes -> Draw Call

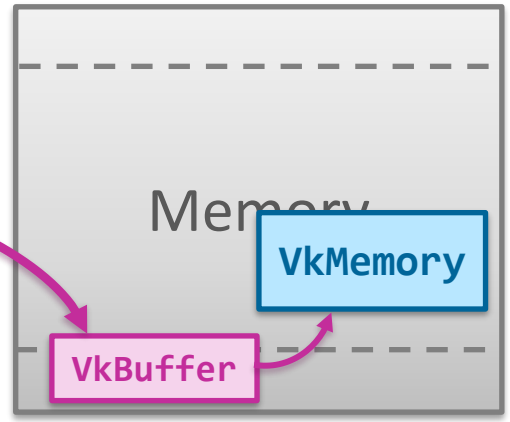
```
VkVertexInputBindingDescription binding0 = {};
binding0.binding = 0;
binding0.stride = sizeof(float) * 3;
binding0.inputRate = VK_VERTEX_INPUT_RATE_VERTEX;

VkVertexInputAttributeDescription attribute0 = {};
attribute0.location = 0;
attribute0.binding = 0;
attribute0.format = VK_FORMAT_R32G32B32_SFLOAT;
attribute0.offset = 0;

VkPipelineVertexInputStateCreateInfo vertexInputState = {};
vertexInputState.sType = VK_STRUCTURE_TYPE_PIPELINE_VERTEX_INPUT_STATE_CREATE_INFO;
vertexInputState.vertexBindingDescriptionCount = 1;
vertexInputState.pVertexBindingDescriptions = &binding0;
vertexInputState.vertexAttributeDescriptionCount = 1;
vertexInputState.pVertexAttributeDescriptions = &attribute0;

VkGraphicsPipelineCreateInfo graphicsPipeCreateInfo = {};
graphicsPipeCreateInfo.sType = VK_STRUCTURE_TYPE_GRAPHICS_PIPELINE_CREATE_INFO;
graphicsPipeCreateInfo.pVertexInputState = &vertexInputState;
// ...
```

GLSL

```
#version 450

layout (location = 0) in vec3 inPosition;

void main()
{
    gl_Position = vec4(inPosition, 1.0);
}
```

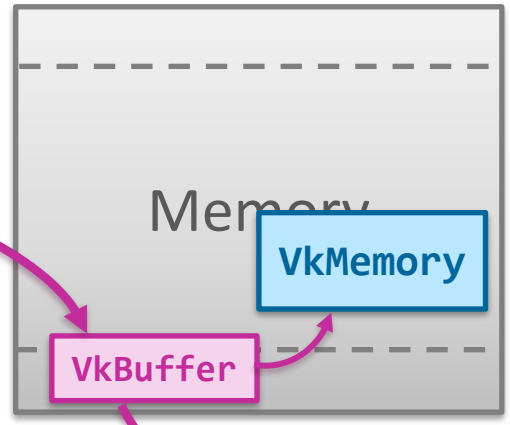
Attributes -> Draw Call

```
VkVertexInputBindingDescription binding0 = {};
binding0.binding = 0;
binding0.stride = sizeof(float) * 3;
binding0.inputRate = VK_VERTEX_INPUT_RATE_VERTEX;

VkVertexInputAttributeDescription attribute0 = {};
attribute0.location = 0;
attribute0.binding = 0;
attribute0.format = VK_FORMAT_R32G32B32_SFLOAT;
attribute0.offset = 0;

VkPipelineVertexInputStateCreateInfo vertexInputState = {};
vertexInputState.sType = VK_STRUCTURE_TYPE_PIPELINE_VERTEX_INPUT_STATE_CREATE_INFO;
vertexInputState.vertexBindingDescriptionCount = 1;
vertexInputState.pVertexBindingDescriptions = &binding0;
vertexInputState.vertexAttributeDescriptionCount = 1;
vertexInputState.pVertexAttributeDescriptions = &attribute0;

VkGraphicsPipelineCreateInfo graphicsPipeCreateInfo = {};
graphicsPipeCreateInfo.sType = VK_STRUCTURE_TYPE_GRAPHICS_PIPELINE_CREATE_INFO;
graphicsPipeCreateInfo.pVertexInputState = &vertexInputState;
// ...
```



Attributes -> Draw Call

```
VkVertexInputBindingDescription binding0 = {};
binding0.binding = 0;
binding0.stride = sizeof(float) * 3;
binding0.inputRate = VK_VERTEX_INPUT_RATE_VERTEX;

VkVertexInputAttributeDescription attribute0 = {};
attribute0.location = 0;
attribute0.binding = 0;
attribute0.format = VK_FORMAT_R32G32B32_SFLOAT;
attribute0.offset = 0;
```

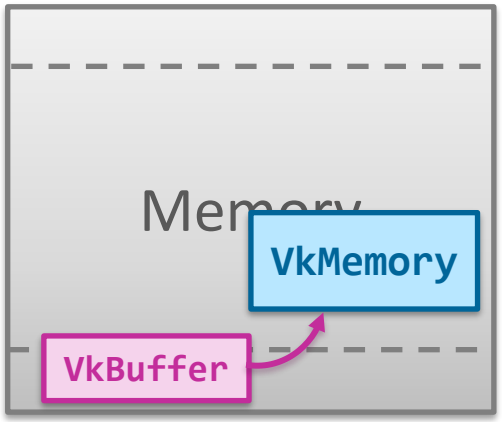



```
VkDeviceSize offset0 = 0;
vkCmdBindVertexBuffers(
  commandBuffer,
  0, 1,
  &buffer0, &offset0
);

vkCmdDraw(...);
```

```
VkPipelineVertexInputStateCreateInfo vertexInputState = {};
vertexInputState.sType = VK_STRUCTURE_TYPE_PIPELINE_VERTEX_INPUT_STATE_CREATE_INFO;
vertexInputState.vertexBindingDescriptionCount = 1;
vertexInputState.pVertexBindingDescriptions = &binding0;
vertexInputState.vertexAttributeDescriptionCount = 1;
vertexInputState.pVertexAttributeDescriptions = &attribute0;

VkGraphicsPipelineCreateInfo graphicsPipeCreateInfo = {};
graphicsPipeCreateInfo.sType = VK_STRUCTURE_TYPE_GRAPHICS_PIPELINE_CREATE_INFO;
graphicsPipeCreateInfo.pVertexInputState = &vertexInputState;
// ...
```

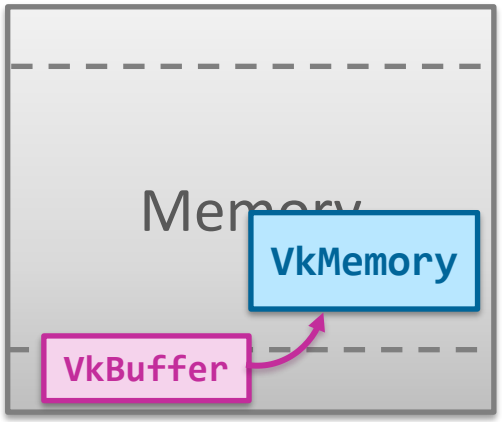
Attributes -> Draw Call

```
VkVertexInputBindingDescription binding0 = {};
binding0.binding = 0;
binding0.stride = sizeof(float) * 3;
binding0.inputRate = VK_VERTEX_INPUT_RATE_VERTEX;

VkVertexInputAttributeDescription attribute0 = {};
attribute0.location = 0;
attribute0.binding = 0;
attribute0.format = VK_FORMAT_R32G32B32_SFLOAT;
attribute0.offset = 0;

VkPipelineVertexInputStateCreateInfo vertexInputState = {};
vertexInputState.sType = VK_STRUCTURE_TYPE_PIPELINE_VERTEX_INPUT_STATE_CREATE_INFO;
vertexInputState.vertexBindingDescriptionCount = 1;
vertexInputState.pVertexBindingDescriptions = &binding0;
vertexInputState.vertexAttributeDescriptionCount = 1;
vertexInputState.pVertexAttributeDescriptions = &attribute0;

VkGraphicsPipelineCreateInfo graphicsPipeCreateInfo = {};
graphicsPipeCreateInfo.sType = VK_STRUCTURE_TYPE_GRAPHICS_PIPELINE_CREATE_INFO;
graphicsPipeCreateInfo.pVertexInputState = &vertexInputState;
// ...
```



Attributes -> Draw Call

```
VkVertexInputBindingDescription binding0 = {};
binding0.binding = 0;
binding0.stride = sizeof(float) * 3;
binding0.inputRate = VK_VERTEX_INPUT_RATE_VERTEX;

VkVertexInputAttributeDescription attribute0 = {};
attribute0.location = 0;
attribute0.binding = 0;
attribute0.format = VK_FORMAT_R32G32B32_SFLOAT;
attribute0.offset = 0;
```

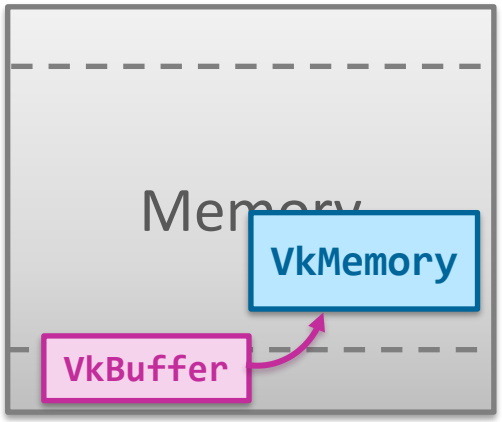



```
struct MyVertexData {
  float position[3];
};
```

```
VkPipelineVertexInputStateCreateInfo vertexInputState = {};
vertexInputState.sType = VK_STRUCTURE_TYPE_PIPELINE_VERTEX_INPUT_STATE_CREATE_INFO;
vertexInputState.vertexBindingDescriptionCount = 1;
vertexInputState.pVertexBindingDescriptions = &binding0;
vertexInputState.vertexAttributeDescriptionCount = 1;
vertexInputState.pVertexAttributeDescriptions = &attribute0;

VkGraphicsPipelineCreateInfo graphicsPipeCreateInfo = {};
graphicsPipeCreateInfo.sType = VK_STRUCTURE_TYPE_GRAPHICS_PIPELINE_CREATE_INFO;
graphicsPipeCreateInfo.pVertexInputState = &vertexInputState;
// ...
```

Attributes -> Draw Call

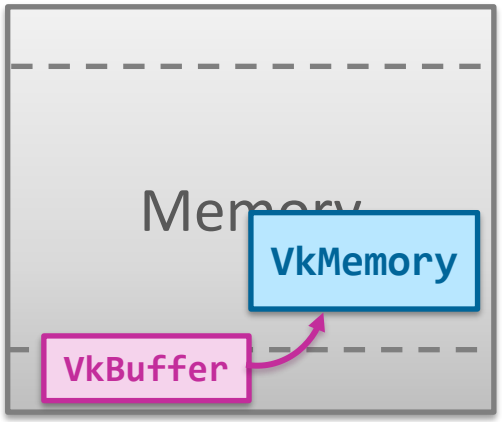
```
VkVertexInputBindingDescription binding0 = {};
binding0.binding = 0;
binding0.stride = sizeof(float) * 3;
binding0.inputRate = VK_VERTEX_INPUT_RATE_VERTEX;

VkVertexInputAttributeDescription attribute0 = {};
attribute0.location = 0;
attribute0.binding = 0;
attribute0.format = VK_FORMAT_R32G32B32_SFLOAT;
attribute0.offset = 0;

VkPipelineVertexInputStateCreateInfo vertexInputState = {};
vertexInputState.sType = VK_STRUCTURE_TYPE_PIPELINE_VERTEX_INPUT_STATE_CREATE_INFO;
vertexInputState.vertexBindingDescriptionCount = 1;
vertexInputState.pVertexBindingDescriptions = &binding0;
vertexInputState.vertexAttributeDescriptionCount = 1;
vertexInputState.pVertexAttributeDescriptions = &attribute0;

VkGraphicsPipelineCreateInfo graphicsPipeCreateInfo = {};
graphicsPipeCreateInfo.sType = VK_STRUCTURE_TYPE_GRAPHICS_PIPELINE_CREATE_INFO;
graphicsPipeCreateInfo.pVertexInputState = &vertexInputState;
// ...
```




```
struct MyVertexData {
  float position[3];
};
```

Attributes -> Draw Call

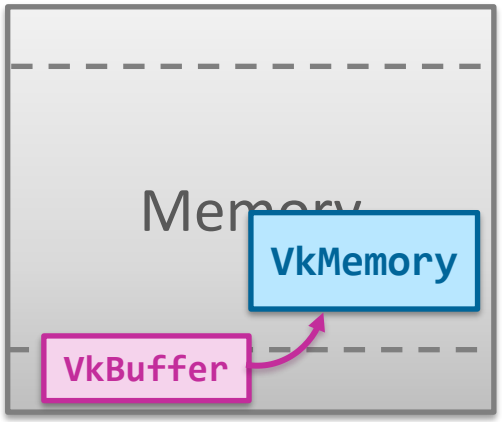
```
VkVertexInputBindingDescription binding0 = {};
binding0.binding = 0;
binding0.stride = sizeof(float) * 3;
binding0.inputRate = VK_VERTEX_INPUT_RATE_VERTEX;

VkVertexInputAttributeDescription attribute0 = {};
attribute0.location = 0;
attribute0.binding = 0;
attribute0.format = VK_FORMAT_R32G32B32_SFLOAT;
attribute0.offset = 0;

VkPipelineVertexInputStateCreateInfo vertexInputState = {};
vertexInputState.sType = VK_STRUCTURE_TYPE_PIPELINE_VERTEX_INPUT_STATE_CREATE_INFO;
vertexInputState.vertexBindingDescriptionCount = 1;
vertexInputState.pVertexBindingDescriptions = &binding0;
vertexInputState.vertexAttributeDescriptionCount = 1;
vertexInputState.pVertexAttributeDescriptions = &attribute0;

VkGraphicsPipelineCreateInfo graphicsPipeCreateInfo = {};
graphicsPipeCreateInfo.sType = VK_STRUCTURE_TYPE_GRAPHICS_PIPELINE_CREATE_INFO;
graphicsPipeCreateInfo.pVertexInputState = &vertexInputState;
// ...
```




```
struct MyVertexData {
  float position[3];
};
```

Attributes -> Draw Call

```
VkVertexInputBindingDescription binding0 = {};
binding0.binding = 0;
binding0.stride = sizeof(float) * 3;
binding0.inputRate = VK_VERTEX_INPUT_RATE_VERTEX;

VkVertexInputAttributeDescription attribute0 = {};
attribute0.location = 0;
attribute0.binding = 0;
attribute0.format = VK_FORMAT_R32G32B32_SFLOAT;
attribute0.offset = 0;
```

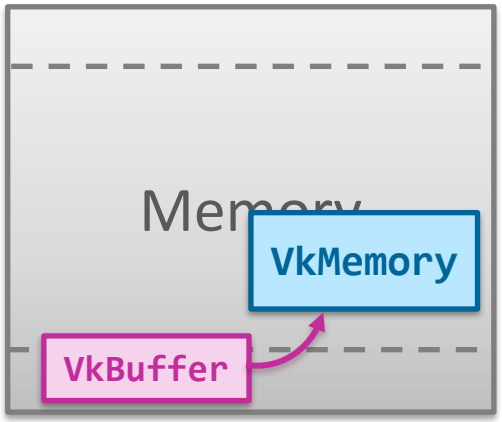



```
struct MyVertexData {
  float position[3];
};
```

```
VkPipelineVertexInputStateCreateInfo vertexInputState = {};
vertexInputState.sType = VK_STRUCTURE_TYPE_PIPELINE_VERTEX_INPUT_STATE_CREATE_INFO;
vertexInputState.vertexBindingDescriptionCount = 1;
vertexInputState.pVertexBindingDescriptions = &binding0;
vertexInputState.vertexAttributeDescriptionCount = 1;
vertexInputState.pVertexAttributeDescriptions = &attribute0;

VkGraphicsPipelineCreateInfo graphicsPipeCreateInfo = {};
graphicsPipeCreateInfo.sType = VK_STRUCTURE_TYPE_GRAPHICS_PIPELINE_CREATE_INFO;
graphicsPipeCreateInfo.pVertexInputState = &vertexInputState;
// ...
```


Attributes -> Draw Call

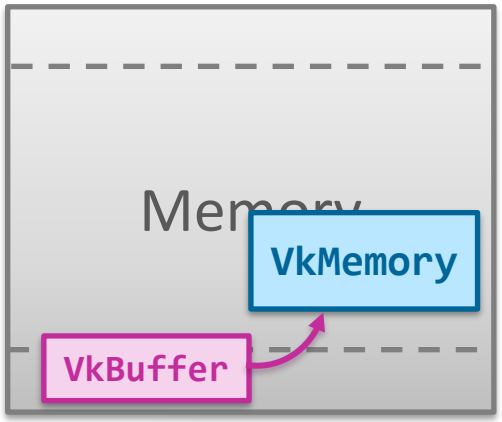
```
VkVertexInputBindingDescription binding0 = {};
binding0.binding = 0;
binding0.stride = sizeof(float) * 3;
binding0.inputRate = VK_VERTEX_INPUT_RATE_VERTEX;

VkVertexInputAttributeDescription attribute0 = {};
attribute0.location = 0;
attribute0.binding = 0;
attribute0.format = VK_FORMAT_R32G32B32_SFLOAT;
attribute0.offset = 0;

VkPipelineVertexInputStateCreateInfo vertexInputState = {};
vertexInputState.sType = VK_STRUCTURE_TYPE_PIPELINE_VERTEX_INPUT_STATE_CREATE_INFO;
vertexInputState.vertexBindingDescriptionCount = 1;
vertexInputState.pVertexBindingDescriptions = &binding0;
vertexInputState.vertexAttributeDescriptionCount = 1;
vertexInputState.pVertexAttributeDescriptions = &attribute0;

VkGraphicsPipelineCreateInfo graphicsPipeCreateInfo = {};
graphicsPipeCreateInfo.sType = VK_STRUCTURE_TYPE_GRAPHICS_PIPELINE_CREATE_INFO;
graphicsPipeCreateInfo.pVertexInputState = &vertexInputState;
// ...
```




```
struct MyVertexData {
  float position[3];
};
```

Attributes -> Draw Call

```
VkVertexInputBindingDescription binding0 = {};
binding0.binding = 0;
binding0.stride = sizeof(float) * 3;
binding0.inputRate = VK_VERTEX_INPUT_RATE_VERTEX;

VkVertexInputAttributeDescription attribute0 = {};
attribute0.location = 0;
attribute0.binding = 0;
attribute0.format = VK_FORMAT_R32G32B32_SFLOAT;
attribute0.offset = 0;

VkPipelineVertexInputStateCreateInfo vertexInputState = {};
vertexInputState.sType = VK_STRUCTURE_TYPE_PIPELINE_VERTEX_I
vertexInputState.vertexBindingDescriptionCount = 1;
vertexInputState.pVertexBindingDescriptions = &binding0;
vertexInputState.vertexAttributeDescriptionCount = 1;
vertexInputState.pVertexAttributeDescriptions = &attribute0;

VkGraphicsPipelineCreateInfo graphicsPipeCreateInfo = {};
graphicsPipeCreateInfo.sType = VK_STRUCTURE_TYPE_GRAPHICS_PIPELINE_CREATE_INFO;
graphicsPipeCreateInfo.pVertexInputState = &vertexInputState;
// ...
```



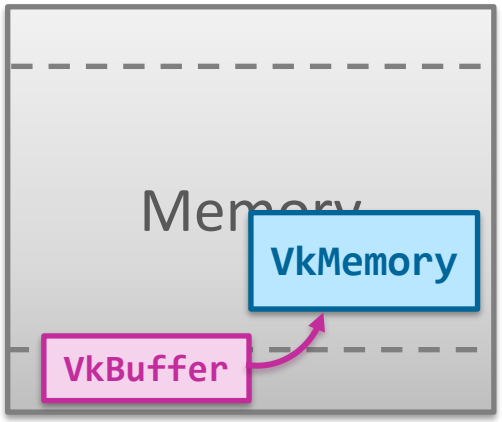

```
struct MyVertexData {
  float position[3];
};
```

GLSL

```
#version 450

layout (location = 0) in vec3 inPosition;

void main()
{
    gl_Position = vec4(inPosition, 1.0);
}
```

Attributes -> Draw Call

```
VkVertexInputBindingDescription binding0 = {};
binding0.binding = 0;
binding0.stride = sizeof(float) * 3;
binding0.inputRate = VK_VERTEX_INPUT_RATE_VERTEX;

VkVertexInputAttributeDescription attribute0 = {};
attribute0.location = 0;
attribute0.binding = 0;
attribute0.format = VK_FORMAT_R32G32B32_SFLOAT;
attribute0.offset = 0;
```

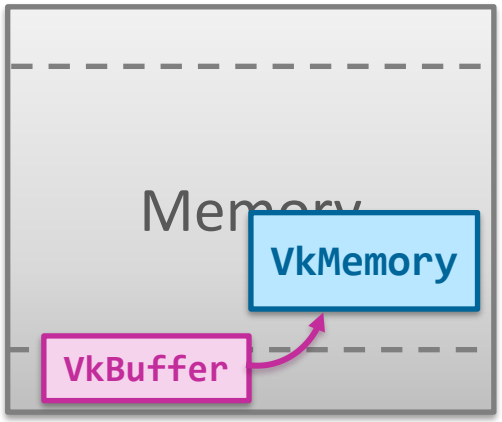



```
struct MyVertexData {
  float position[3];
};
```

```
VkPipelineVertexInputStateCreateInfo vertexInputState = {};
vertexInputState.sType = VK_STRUCTURE_TYPE_PIPELINE_VERTEX_INPUT_STATE_CREATE_INFO;
vertexInputState.vertexBindingDescriptionCount = 1;
vertexInputState.pVertexBindingDescriptions = &binding0;
vertexInputState.vertexAttributeDescriptionCount = 1;
vertexInputState.pVertexAttributeDescriptions = &attribute0;

VkGraphicsPipelineCreateInfo graphicsPipeCreateInfo = {};
graphicsPipeCreateInfo.sType = VK_STRUCTURE_TYPE_GRAPHICS_PIPELINE_CREATE_INFO;
graphicsPipeCreateInfo.pVertexInputState = &vertexInputState;
// ...
```

Attributes -> Draw Call

```
VkVertexInputBindingDescription binding0 = {};
binding0.binding = 0;
binding0.stride = sizeof(float) * 3;
binding0.inputRate = VK_VERTEX_INPUT_RATE_VERTEX;

VkVertexInputAttributeDescription attribute0 = {};
attribute0.location = 0;
attribute0.binding = 0;
attribute0.format = VK_FORMAT_R32G32B32_SFLOAT;
attribute0.offset = 0;
```

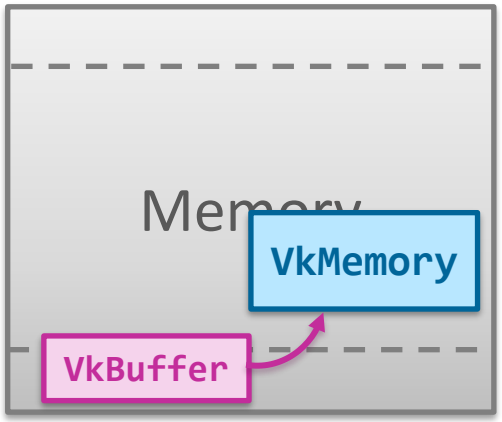



```
struct MyVertexData {
  float position[3];
  float normal[3];
  float texCoord[2];
};
```

⚠ REFACTORING IN PROGRESS...

Attributes -> Draw Call

```
VkVertexInputBindingDescription binding0 = {};
binding0.binding = 0;
binding0.stride = sizeof(float) * 8;
binding0.inputRate = VK_VERTEX_INPUT_RATE_VERTEX;

VkVertexInputAttributeDescription attribute0 = {};
attribute0.location = 0;
attribute0.binding = 0;
attribute0.format = VK_FORMAT_R32G32B32_SFLOAT;
attribute0.offset = 0;
```

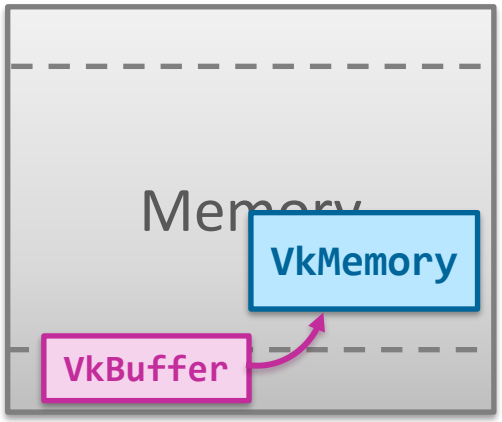



```
struct MyVertexData {
  float position[3];
  float normal[3];
  float texCoord[2];
};
```

⚠ REFACTORING IN PROGRESS...

Attributes -> Draw Call

```
VkVertexInputBindingDescription binding0 = {};
binding0.binding = 0;
binding0.stride = sizeof(float) * 8;
binding0.inputRate = VK_VERTEX_INPUT_RATE_VERTEX;

VkVertexInputAttributeDescription attribute0 = {};
attribute0.location = 0;
attribute0.binding = 0;
attribute0.format = VK_FORMAT_R32G32B32_SFLOAT;
attribute0.offset = 0;

VkVertexInputAttributeDescription attribute1 = {};
attribute1.location = 1;
attribute1.binding = 0;
attribute1.format = VK_FORMAT_R32G32B32_SFLOAT;
attribute1.offset = sizeof(float) * 3;

VkVertexInputAttributeDescription attribute2 = {};
attribute2.location = 2;
attribute2.binding = 0;
attribute2.format = VK_FORMAT_R32G32_SFLOAT;
attribute2.offset = sizeof(float) * 6;
```

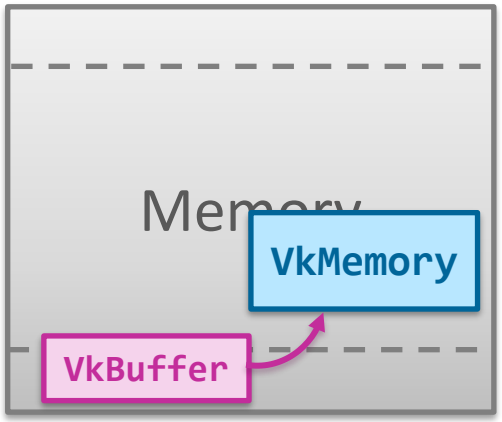



```
struct MyVertexData {
  float position[3];
  float normal[3];
  float texCoord[2];
};
```

☑ REFACTORING DONE

Attributes -> Draw Call

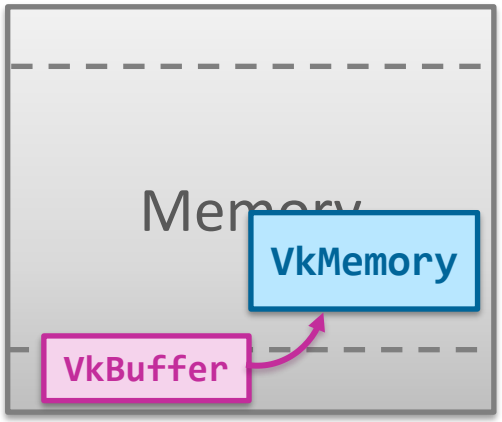
```
VkVertexInputBindingDescription binding0 = {};
binding0.binding = 0;
binding0.stride = sizeof(float) * 8;
binding0.inputRate = VK_VERTEX_INPUT_RATE_VERTEX;

VkVertexInputAttributeDescription attribute0 = {};
attribute0.location = 0;
attribute0.binding = 0;
attribute0.format = VK_FORMAT_R32G32B32_SFLOAT;
attribute0.offset = 0;

VkVertexInputAttributeDescription attribute1 = {};
attribute1.location = 1;
attribute1.binding = 0;
attribute1.format = VK_FORMAT_R32G32B32_SFLOAT;
attribute1.offset = sizeof(float) * 3;

VkVertexInputAttributeDescription attribute2 = {};
attribute2.location = 2;
attribute2.binding = 0;
attribute2.format = VK_FORMAT_R32G32_SFLOAT;
attribute2.offset = sizeof(float) * 6;
```




```
struct MyVertexData {
  float position[3];
  float normal[3];
  float texCoord[2];
};
```

Attributes -> Draw Call

```
VkVertexInputBindingDescription binding0 = {};
binding0.binding = 0;
binding0.stride = sizeof(float) * 8;
binding0.inputRate = VK_VERTEX_INPUT_RATE_VERTEX;

VkVertexInputAttributeDescription attribute0 = {};
attribute0.location = 0;
attribute0.binding = 0;
attribute0.format = VK_FORMAT_R32G32B32_SFLOAT;
attribute0.offset = 0;

VkVertexInputAttributeDescription attribute1 = {};
attribute1.location = 1;
attribute1.binding = 0;
attribute1.format = VK_FORMAT_R32G32B32_SFLOAT;
attribute1.offset = sizeof(float) * 3;

VkVertexInputAttributeDescription attribute2 = {};
attribute2.location = 2;
attribute2.binding = 0;
attribute2.format = VK_FORMAT_R32G32_SFLOAT;
attribute2.offset = sizeof(float) * 6;
```



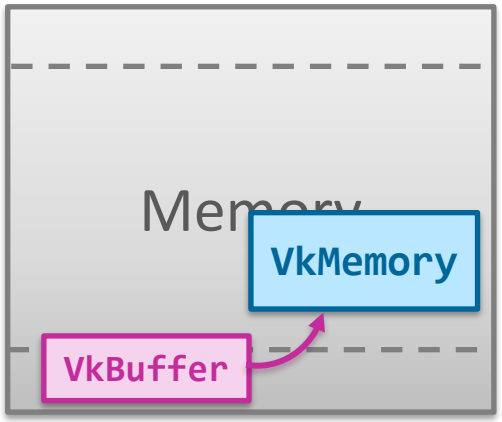

```
struct MyVertexData {
  float position[3];
  float normal[3];
  float texCoord[2];
};
```

GLSL

```
#version 450

layout (location = 0) in vec3 inPosition;
layout (location = 1) in vec3 inNormal;
layout (location = 2) in vec2 inTexCoord;

void main()
{
    gl_Position = vec4(inPosition, 1.0);
}
```

Attributes -> Draw Call

```
VkVertexInputBindingDescription binding0 = {};
binding0.binding = 0;
binding0.stride = sizeof(float) * 8;
binding0.inputRate = VK_VERTEX_INPUT_RATE_VERTEX;

VkVertexInputAttributeDescription attribute0 = {};
attribute0.location = 0;
attribute0.binding = 0;
attribute0.format = VK_FORMAT_R32G32B32_SFLOAT;
attribute0.offset = 0;

VkVertexInputAttributeDescription attribute1 = {};
attribute1.location = 1;
attribute1.binding = 0;
attribute1.format = VK_FORMAT_R32G32B32_SFLOAT;
attribute1.offset = sizeof(float) * 3;

VkVertexInputAttributeDescription attribute2 = {};
attribute2.location = 2;
attribute2.binding = 0;
attribute2.format = VK_FORMAT_R32G32_SFLOAT;
attribute2.offset = sizeof(float) * 6;
```



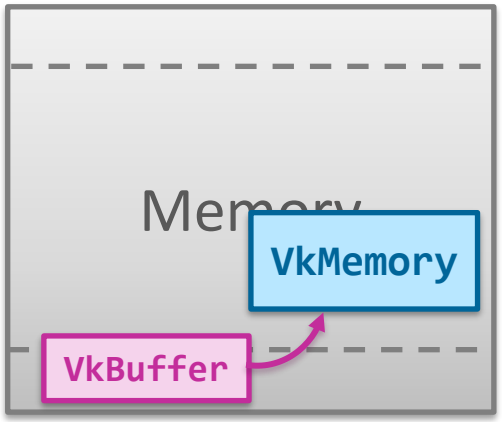

```
struct MyVertexData {
  float position[3];
  float normal[3];
  float texCoord[2];
};
```

GLSL

```
#version 450

layout (location = 0) in vec3 inPosition;
layout (location = 1) in vec3 inNormal;
layout (location = 2) in vec2 inTexCoord;

void main()
{
    gl_Position = vec4(inPosition, 1.0);
}
```

Attributes -> Draw Call

```
VkVertexInputBindingDescription binding0 = {};
binding0.binding = 0;
binding0.stride = sizeof(float) * 8;
binding0.inputRate = VK_VERTEX_INPUT_RATE_VERTEX;

VkVertexInputAttributeDescription attribute0 = {};
attribute0.location = 0;
attribute0.binding = 0;
attribute0.format = VK_FORMAT_R32G32B32_SFLOAT;
attribute0.offset = 0;

VkVertexInputAttributeDescription attribute1 = {};
attribute1.location = 1;
attribute1.binding = 0;
attribute1.format = VK_FORMAT_R32G32B32_SFLOAT;
attribute1.offset = sizeof(float) * 3;

VkVertexInputAttributeDescription attribute2 = {};
attribute2.location = 2;
attribute2.binding = 0;
attribute2.format = VK_FORMAT_R32G32_SFLOAT;
attribute2.offset = sizeof(float) * 6;
```



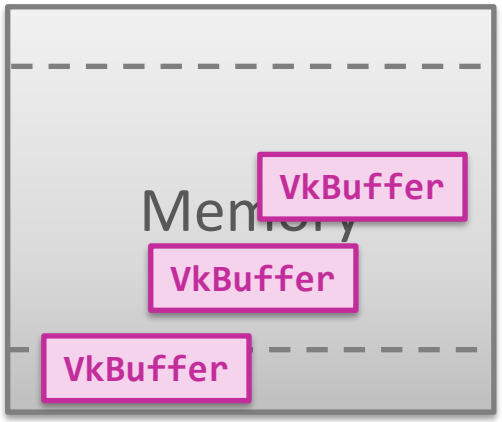

```
struct MyVertexData {
  float position[3];
  float normal[3];
  float texCoord[2];
};
```

GLSL

```
#version 450

layout (location = 0) in vec3 inPosition;
layout (location = 1) in vec3 inNormal;
layout (location = 2) in vec2 inTexCoord;

void main()
{
    gl_Position = vec4(inPosition, 1.0);
}
```

Attributes -> Draw Call

```
VkVertexInputBindingDescription binding0 = {};
binding0.binding = 0;
binding0.stride = sizeof(float) * 8;
binding0.inputRate = VK_VERTEX_INPUT_RATE_VERTEX;

VkVertexInputAttributeDescription attribute0 = {};
attribute0.location = 0;
attribute0.binding = 0;
attribute0.format = VK_FORMAT_R32G32B32_SFLOAT;
attribute0.offset = 0;

VkVertexInputAttributeDescription attribute1 = {};
attribute1.location = 1;
attribute1.binding = 0;
attribute1.format = VK_FORMAT_R32G32B32_SFLOAT;
attribute1.offset = sizeof(float) * 3;

VkVertexInputAttributeDescription attribute2 = {};
attribute2.location = 2;
attribute2.binding = 0;
attribute2.format = VK_FORMAT_R32G32_SFLOAT;
attribute2.offset = sizeof(float) * 6;
```




```
struct MyVertexData {
  float position[3];
};
```

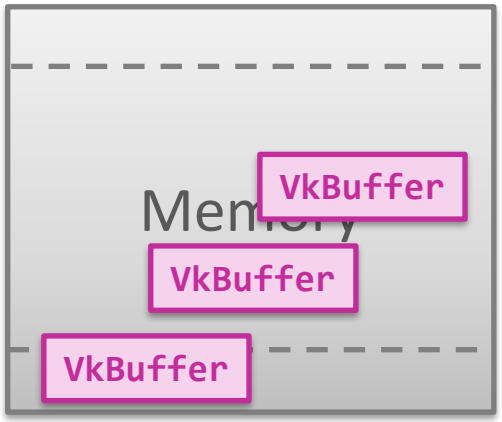
```
struct MyVertexData {
  float normal[3];
};
```

```
struct MyVertexData {
  float texCoord[2];
};
```

⚠ REFACTORING IN PROGRESS...

Attributes -> Draw Call

```
VkVertexInputBindingDescription binding0 = {};
binding0.binding = 0;
binding0.stride = sizeof(float) * 8;
binding0.inputRate = VK_VERTEX_INPUT_RATE_VERTEX;

VkVertexInputAttributeDescription attribute0 = {};
attribute0.location = 0;
attribute0.binding = 0;
attribute0.format = VK_FORMAT_R32G32B32_SFLOAT;
attribute0.offset = 0;

VkVertexInputAttributeDescription attribute1 = {};
attribute1.location = 1;
attribute1.binding = 0;
attribute1.format = VK_FORMAT_R32G32B32_SFLOAT;
attribute1.offset = 0;

VkVertexInputAttributeDescription attribute2 = {};
attribute2.location = 2;
attribute2.binding = 0;
attribute2.format = VK_FORMAT_R32G32_SFLOAT;
attribute2.offset = 0;
```




```
struct MyVertexData {
  float position[3];
};
```

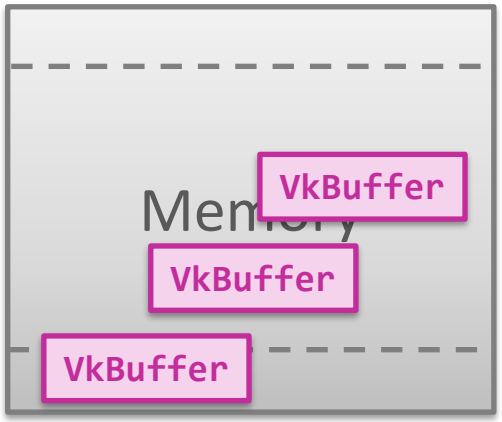
```
struct MyVertexData {
  float normal[3];
};
```

```
struct MyVertexData {
  float texCoord[2];
};
```

⚠ REFACTORING IN PROGRESS...

Attributes -> Draw Call

```
VkVertexInputBindingDescription binding0 = {};
binding0.binding = 0;
binding0.stride = sizeof(float) * 8;
binding0.inputRate = VK_VERTEX_INPUT_RATE_VERTEX;

VkVertexInputAttributeDescription attribute0 = {};
attribute0.location = 0;
attribute0.binding = 0;
attribute0.format = VK_FORMAT_R32G32B32_SFLOAT;
attribute0.offset = 0;

VkVertexInputAttributeDescription attribute1 = {};
attribute1.location = 1;
attribute1.binding = 0;
attribute1.format = VK_FORMAT_R32G32B32_SFLOAT;
attribute1.offset = 0;

VkVertexInputAttributeDescription attribute2 = {};
attribute2.location = 2;
attribute2.binding = 0;
attribute2.format = VK_FORMAT_R32G32_SFLOAT;
attribute2.offset = 0;
```




```
struct MyVertexData {
  float position[3];
};
```

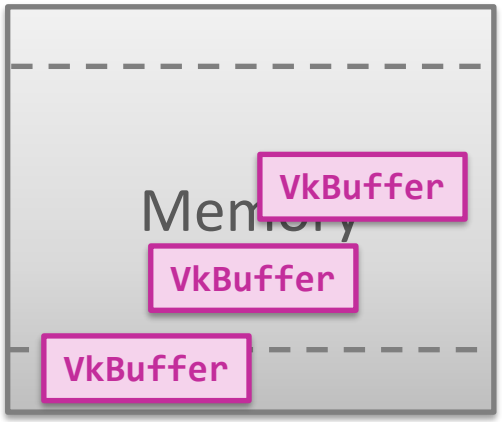
```
struct MyVertexData {
  float normal[3];
};
```

```
struct MyVertexData {
  float texCoord[2];
};
```

⚠ REFACTORING IN PROGRESS...

Attributes -> Draw Call

```
VkVertexInputBindingDescription binding0 = {};
binding0.binding = 0;
binding0.stride = sizeof(float) * 8;
binding0.inputRate = VK_VERTEX_INPUT_RATE_VERTEX;

VkVertexInputAttributeDescription attribute0 = {};
attribute0.location = 0;
attribute0.binding = 0;
attribute0.format = VK_FORMAT_R32G32B32_SFLOAT;
attribute0.offset = 0;

VkVertexInputAttributeDescription attribute1 = {};
attribute1.location = 1;
attribute1.binding = 1;
attribute1.format = VK_FORMAT_R32G32B32_SFLOAT;
attribute1.offset = 0;

VkVertexInputAttributeDescription attribute2 = {};
attribute2.location = 2;
attribute2.binding = 2;
attribute2.format = VK_FORMAT_R32G32_SFLOAT;
attribute2.offset = 0;
```




```
struct MyVertexData {
  float position[3];
};
```

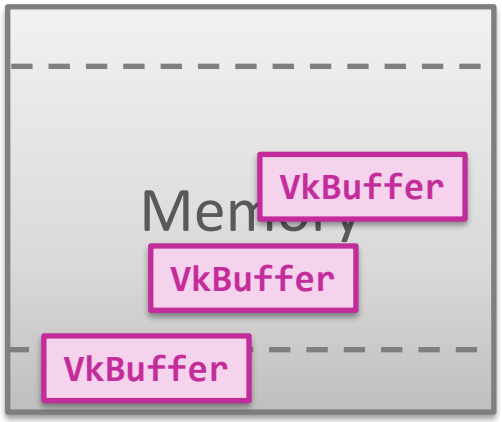
```
struct MyVertexData {
  float normal[3];
};
```

```
struct MyVertexData {
  float texCoord[2];
};
```

⚠ REFACTORING IN PROGRESS...

Attributes -> Draw Call

```
VkVertexInputBindingDescription binding0 = {};
binding0.binding = 0;
binding0.stride = sizeof(float) * 3;
binding0.inputRate = VK_VERTEX_INPUT_RATE_VERTEX;
```




```
struct MyVertexData {
  float position[3];
};
```

```
struct MyVertexData {
  float normal[3];
};
```

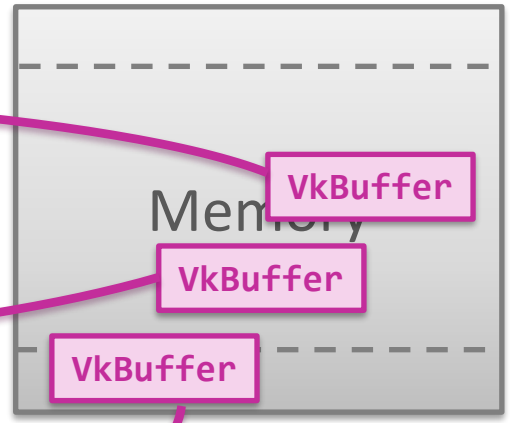
```
struct MyVertexData {
  float texCoord[2];
};
```

```
VkVertexInputAttributeDescription attribute0 = {};
attribute0.binding = 0;

VkVertexInputAttributeDescription attribute1 = {};
attribute1.binding = 1;

VkVertexInputAttributeDescription attribute2 = {};
attribute2.binding = 2;
```

⚠ REFACTORING IN PROGRESS...

Attributes -> Draw Call

```
VkVertexInputBindingDescription binding0 = {};
binding0.binding = 0;
binding0.stride = sizeof(float) * 3;
binding0.inputRate = VK_VERTEX_INPUT_RATE_VERTEX;

VkVertexInputBindingDescription binding0 = {};
binding0.binding = 1;
binding0.stride = sizeof(float) * 3;
binding0.inputRate = VK_VERTEX_INPUT_RATE_VERTEX;

VkVertexInputBindingDescription binding0 = {};
binding0.binding = 2;
binding0.stride = sizeof(float) * 2;
binding0.inputRate = VK_VERTEX_INPUT_RATE_VERTEX;

VkVertexInputAttributeDescription attribute0 = {};
attribute0.binding = 0;

VkVertexInputAttributeDescription attribute1 = {};
attribute1.binding = 1;

VkVertexInputAttributeDescription attribute2 = {};
attribute2.binding = 2;
```




```
struct MyVertexData {
  float position[3];
};
```

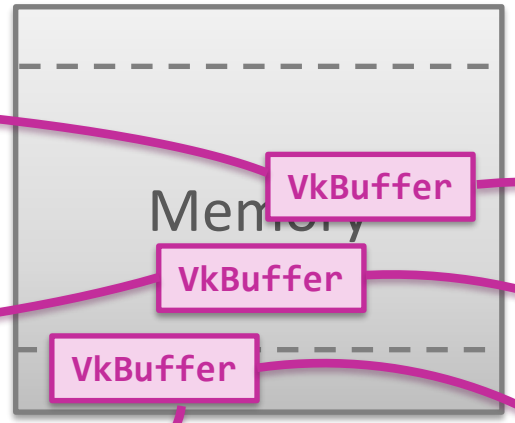
```
struct MyVertexData {
  float normal[3];
};
```

```
struct MyVertexData {
  float texCoord[2];
};
```

☑ REFACTORING DONE

Attributes -> Draw Call

```
VkVertexInputBindingDescription binding0 = {};
binding0.binding = 0;
binding0.stride = sizeof(float) * 3;
binding0.inputRate = VK_VERTEX_INPUT_RATE_VERTEX;

VkVertexInputBindingDescription binding0 = {};
binding0.binding = 1;
binding0.stride = sizeof(float) * 3;
binding0.inputRate = VK_VERTEX_INPUT_RATE_VERTEX;

VkVertexInputBindingDescription binding0 = {};
binding0.binding = 2;
binding0.stride = sizeof(float) * 2;
binding0.inputRate = VK_VERTEX_INPUT_RATE_VERTEX;

VkVertexInputAttributeDescription attribute0 = {};
attribute0.binding = 0;

VkVertexInputAttributeDescription attribute1 = {};
attribute1.binding = 1;

VkVertexInputAttributeDescription attribute2 = {};
attribute2.binding = 2;
```




```
VkBuffer vertexBuffers[3] = {
  buffer0, buffer1, buffer2
};
VkDeviceSize offsets[3] = {
  0, 0, 0
};
vkCmdBindVertexBuffers(
  commandBuffer,
  0, 3,
  vertexBuffers, offsets);
vkCmdDraw(...);
```

Data -> Command

- Descriptors
  - Establish links to resources via descriptors
  - See Episode 3
- Push Constants
  - Data bundled with command buffer
  - Very limited in size
- Parameters
- Attributes
  - Graphics pipelines only
  - Streamed to vertex shaders
  - Accessible via input location

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