Vulkanised 2023

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

An Introduction to Vulkan

Johannes Unterguggenberger

TU Wien, Huawei

Schedule

PART 1:

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

PART 2:

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

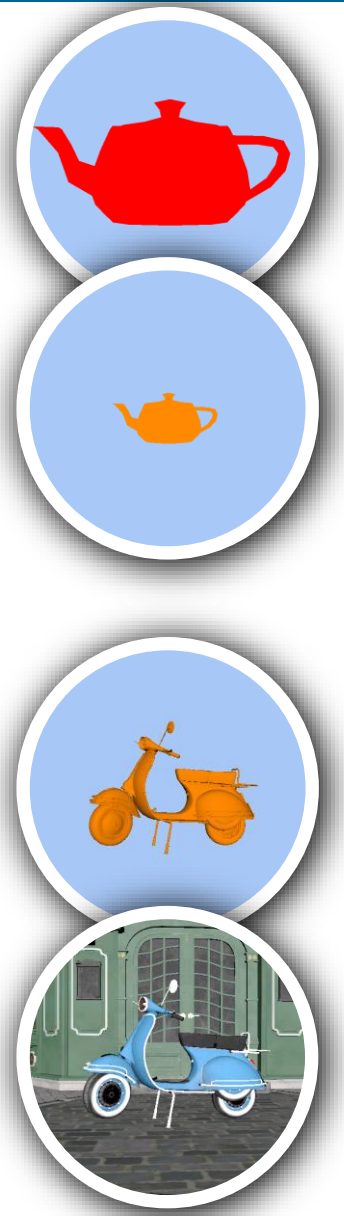
Lunch Break 13:00 – 14:00

PART 3:

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

PART 4:

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



Vulkanised 2023

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

PART 4

- Command Buffer Allocation
- Memory
- Image Layout Transitions
- Synchronization

Platinum Sponsors: AMD arm Google LUNARG KHRONOS GROUP SAMSUNG

Vulkanised 2023

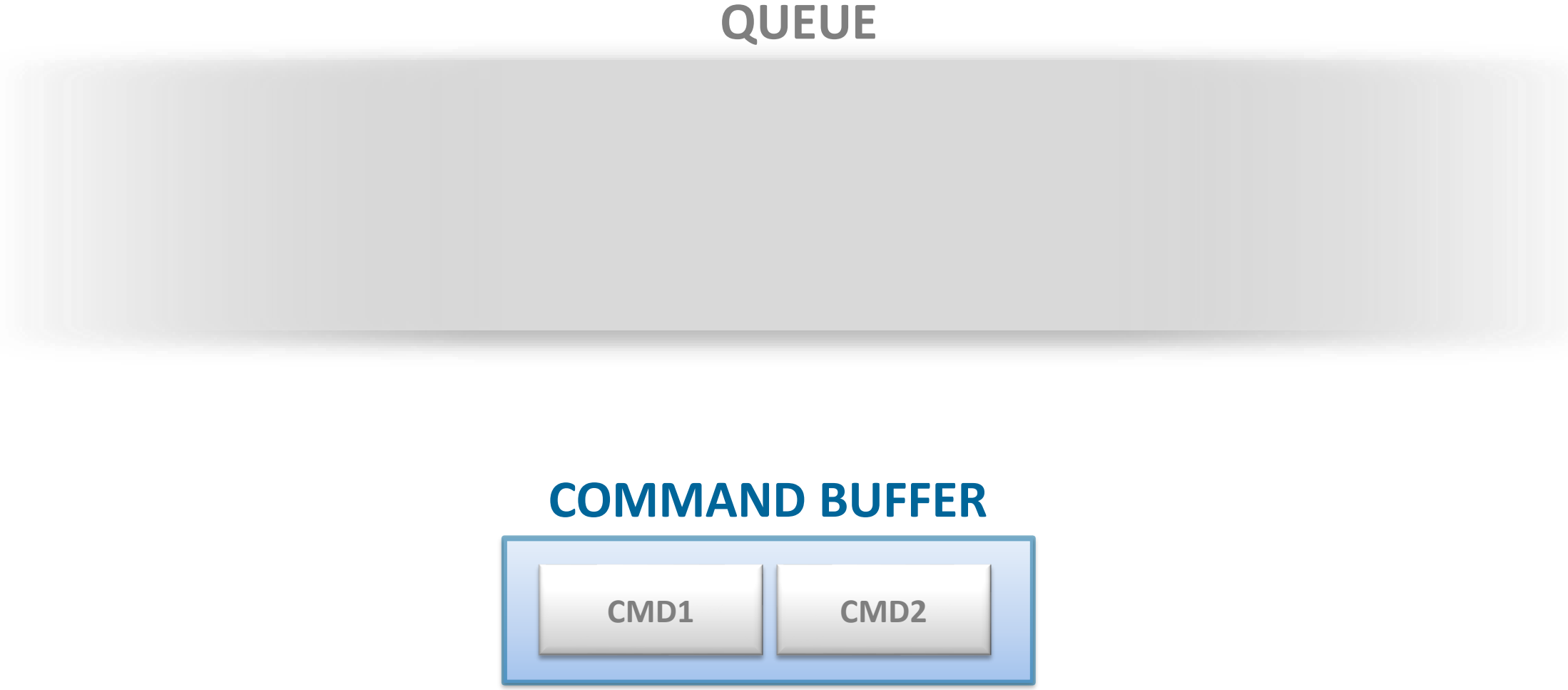
The 5th Vulkan Developer Conference
Munich, Germany / February 7–9

PART 4

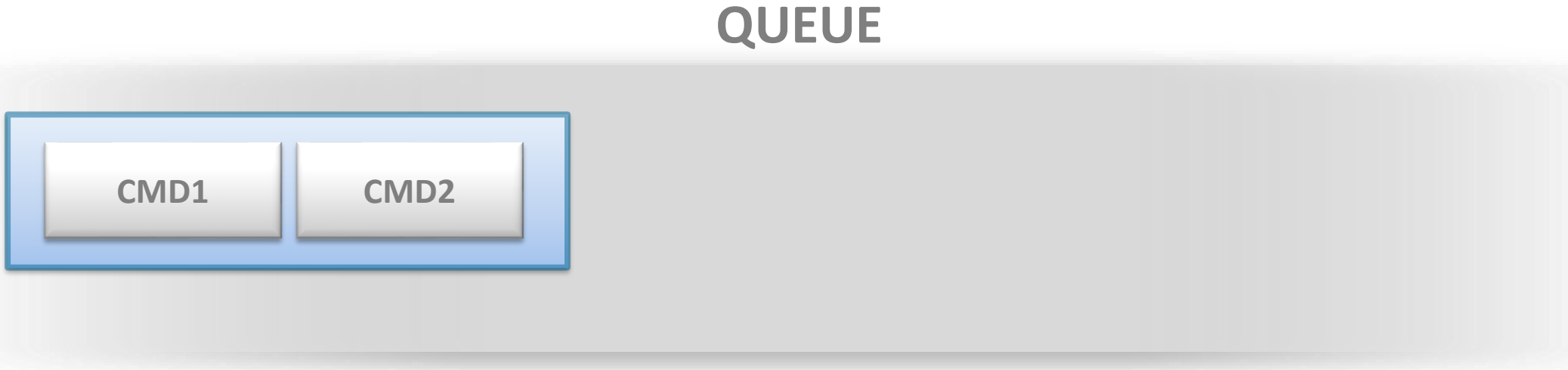
- **Command Buffer Allocation**
- Memory
- Image Layout Transitions
- Synchronization

Platinum Sponsors: AMD arm Google LUNARG KHRONOS GROUP SAMSUNG

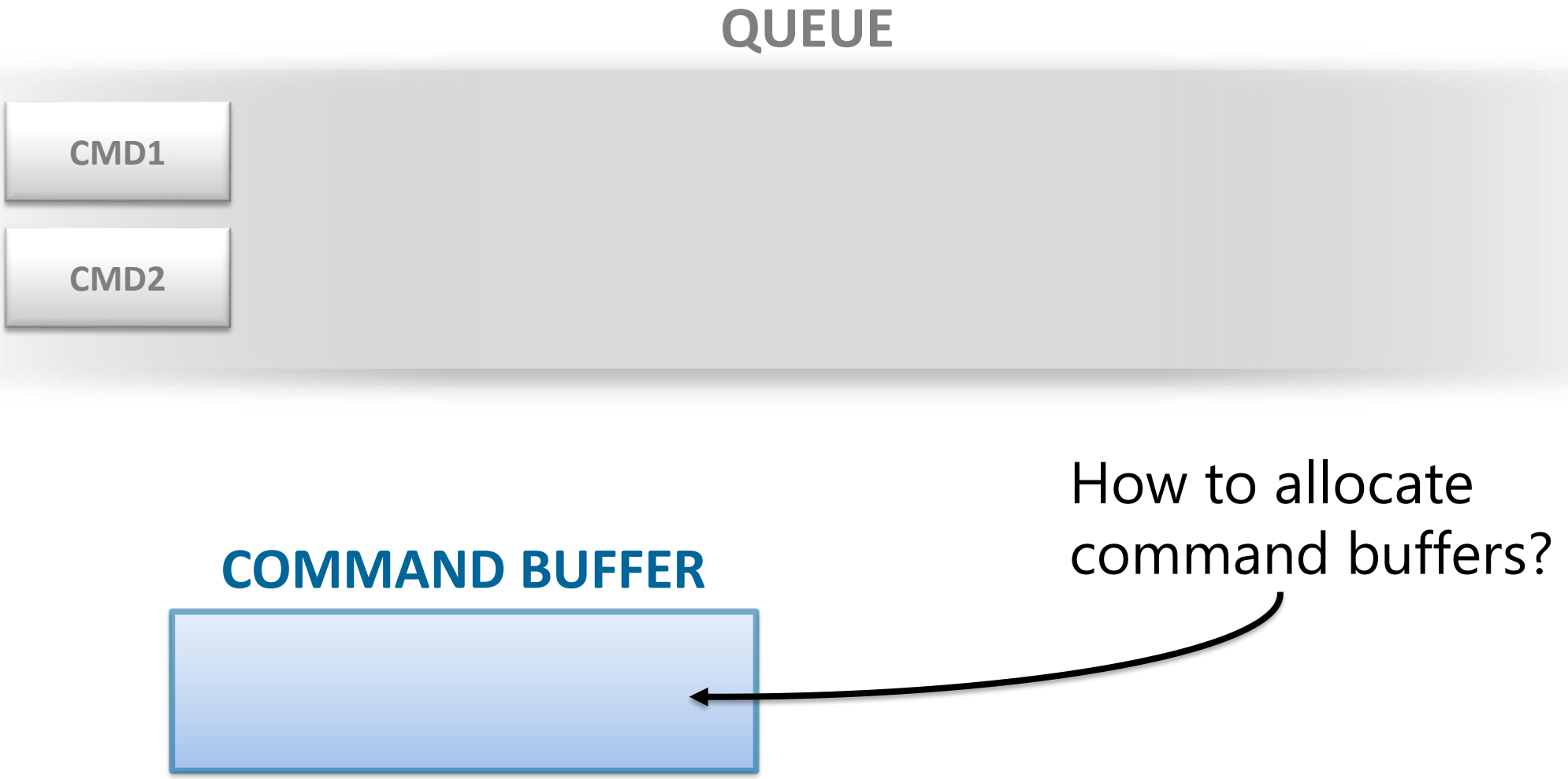
Recap: Command Buffer Recording



Recap: Command Buffer Recording



Recap: Command Buffer Recording



Command Buffer Allocation

```
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 Allocation

```
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 Allocation

```
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 Allocation

```
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 Allocation

```
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 Allocation

```
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 Allocation

```
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 Allocation

```
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 Allocation

```
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 Allocation

```
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 Allocation

```
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);
```

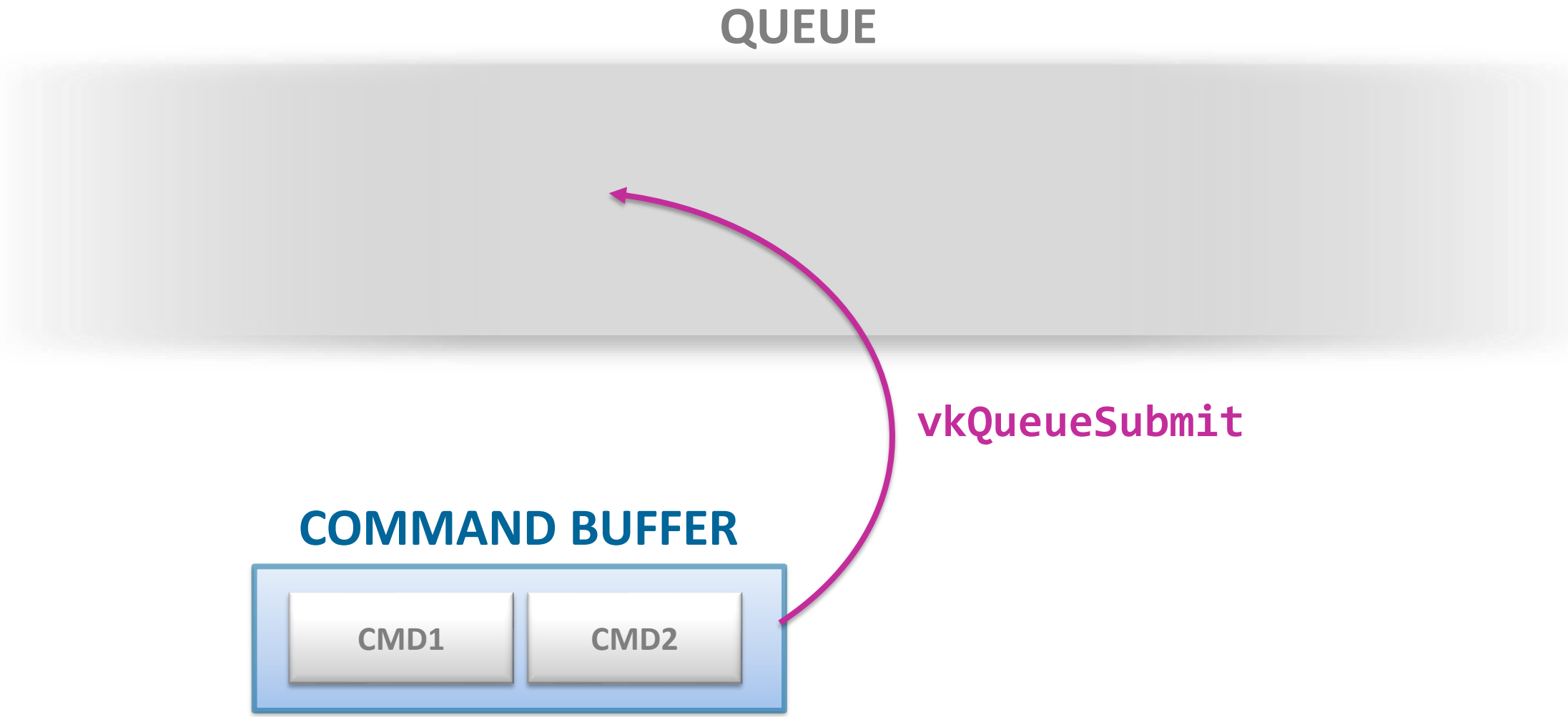

Command Buffer Allocation

```
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);
```

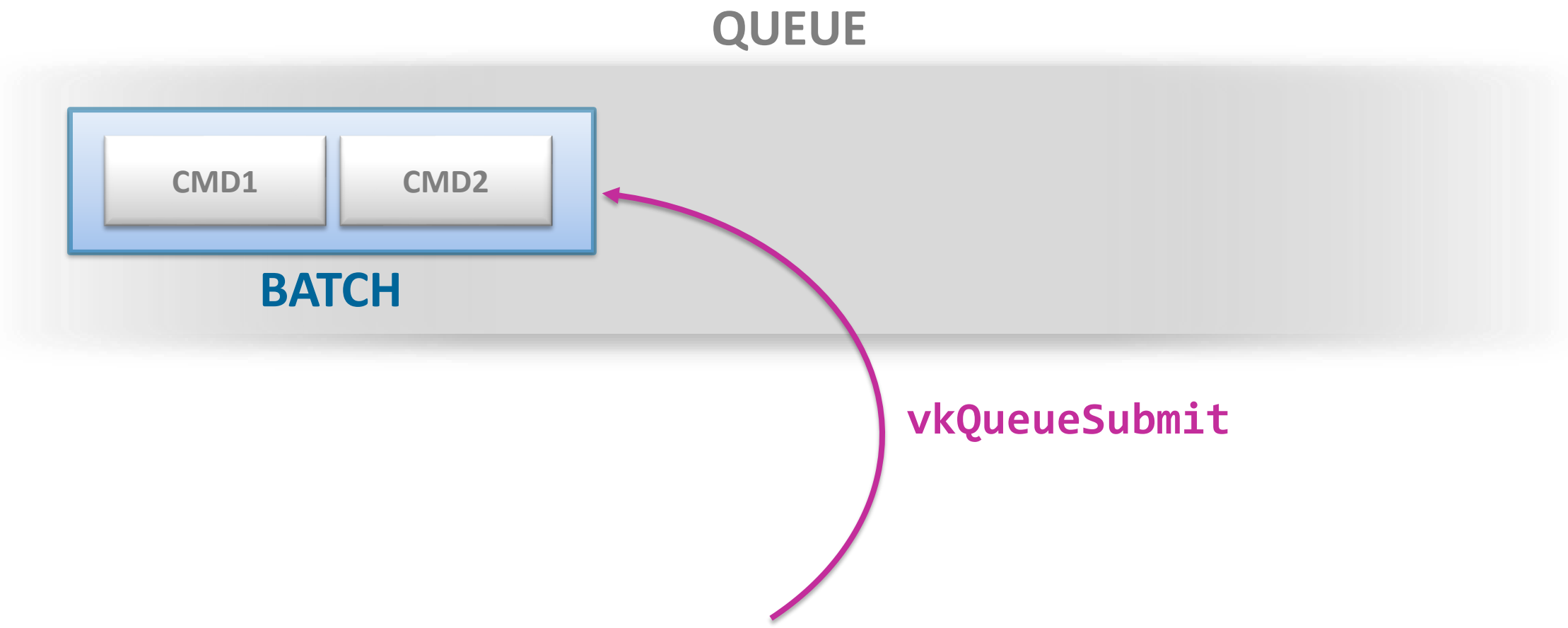

Command Buffer Allocation

```
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);
```

Queue Submission



Queue Submission



Vulkanised 2023

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

PART 4

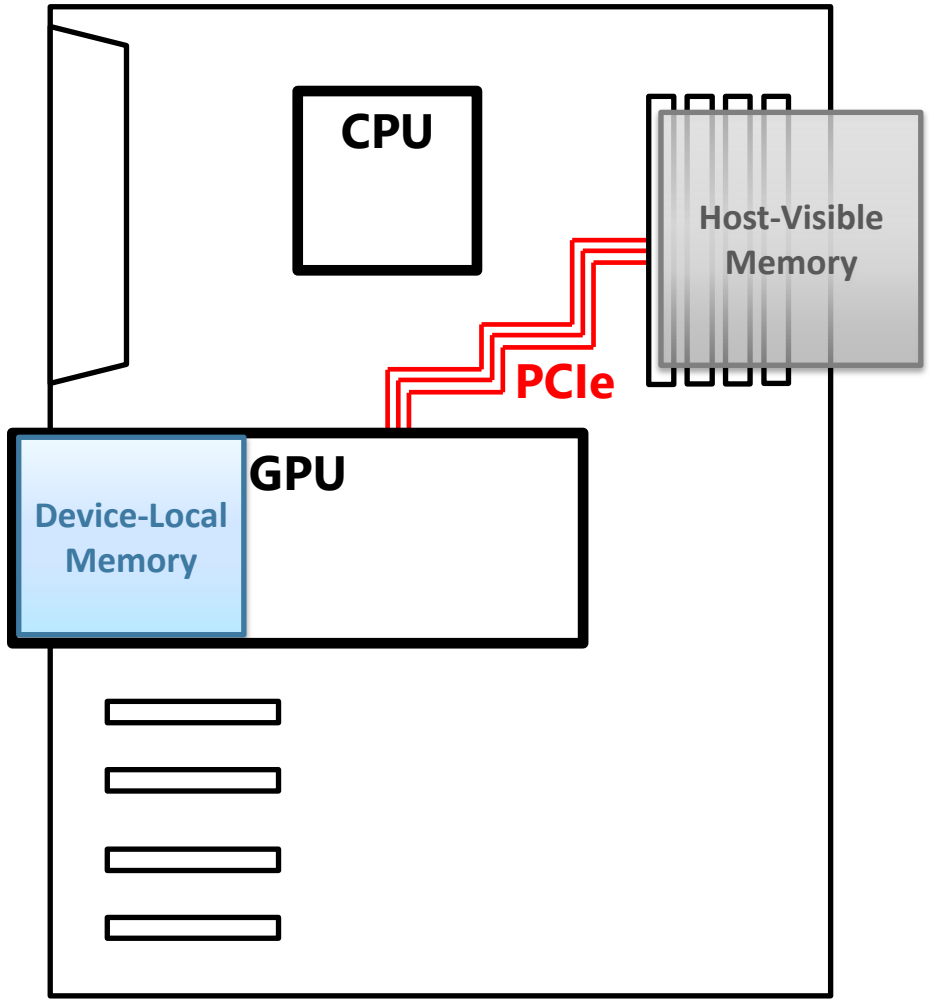
- **Command Buffer Allocation**
- Memory
- Image Layout Transitions
- Synchronization

Platinum Sponsors: AMD arm Google LUNARG KHRONOS GROUP SAMSUNG

PART 4

- Command Buffer Allocation
- **Memory**
- Image Layout Transitions
- Synchronization

Memory

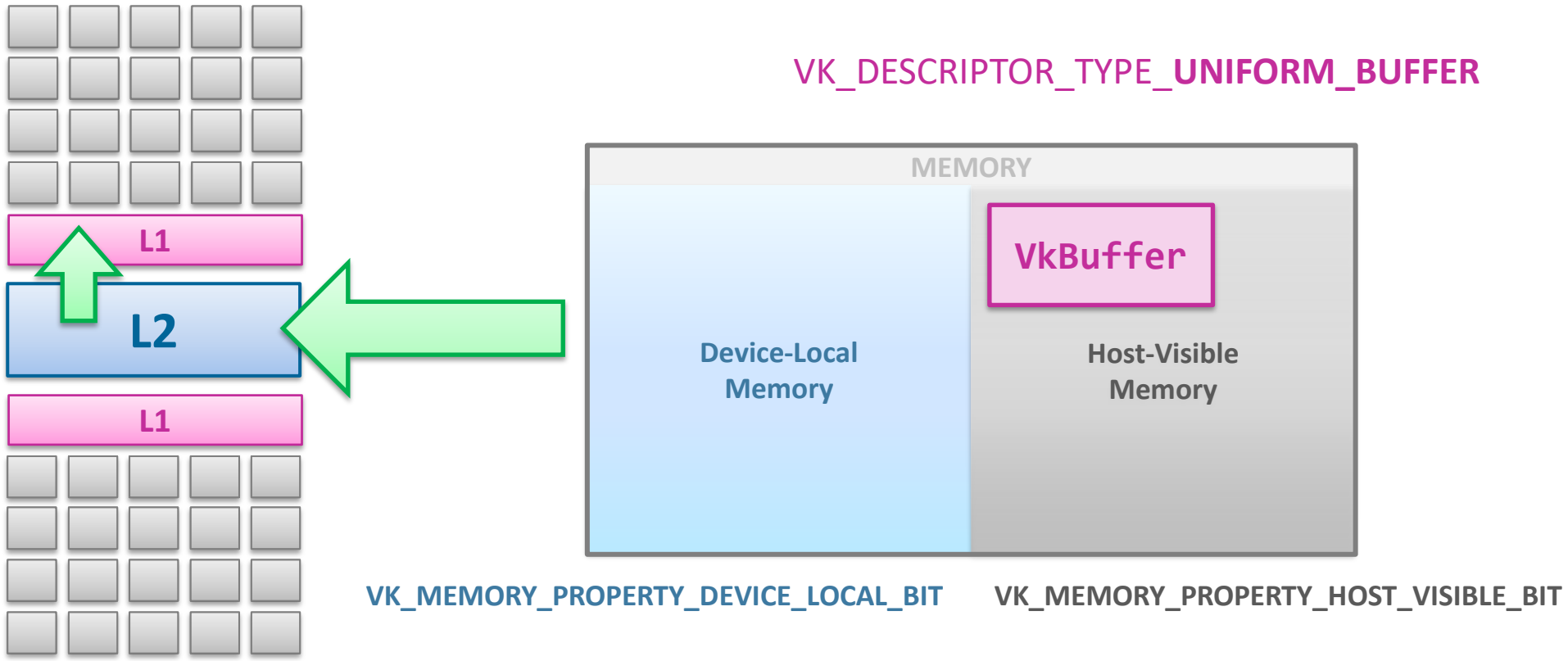


VK_MEMORY_PROPERTY_DEVICE_LOCAL_BIT

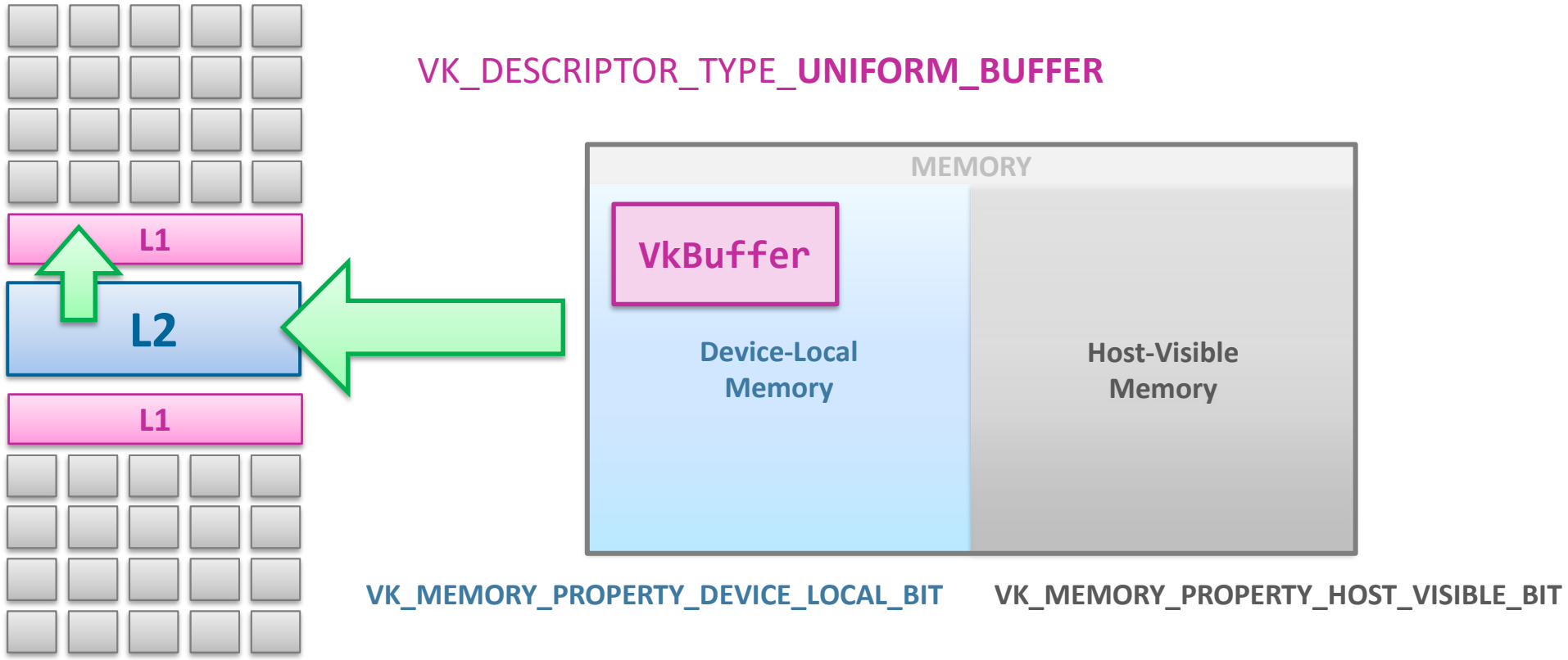
VK_MEMORY_PROPERTY_HOST_VISIBLE_BIT

VK_MEMORY_PROPERTY_HOST_COHERENT_BIT
(no memory flush needed)

Memory

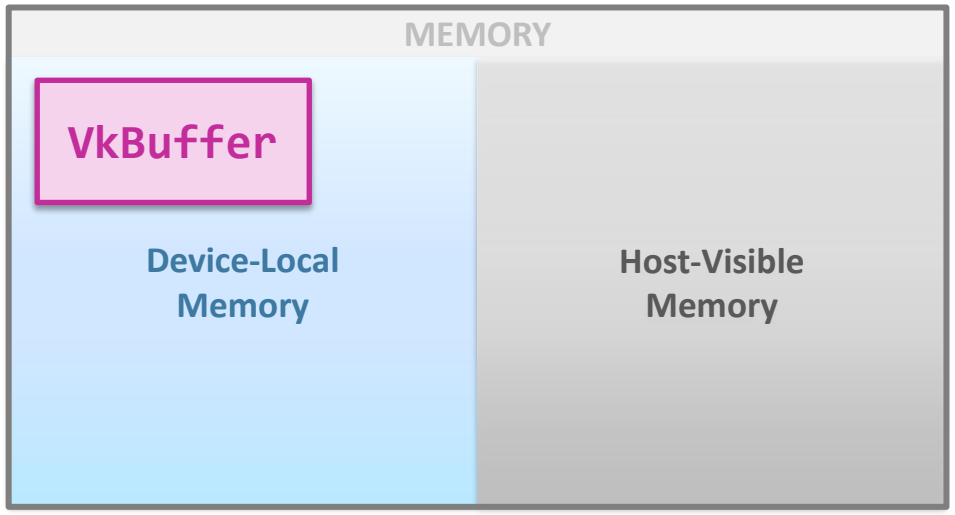

VK_DESCRIPTOR_TYPE_UNIFORM_BUFFER
L1
L2
L1
MEMORY
VkBuffer
Device-Local Memory
Host-Visible Memory
VK_MEMORY_PROPERTY_DEVICE_LOCAL_BIT
VK_MEMORY_PROPERTY_HOST_VISIBLE_BIT

Memory


VK_DESCRIPTOR_TYPE_UNIFORM_BUFFER
L1
L2
L1
MEMORY
VkBuffer
Device-Local Memory
Host-Visible Memory
VK_MEMORY_PROPERTY_DEVICE_LOCAL_BIT
VK_MEMORY_PROPERTY_HOST_VISIBLE_BIT

Memory

VK_DESCRIPTOR_TYPE_**UNIFORM_BUFFER**



← Good for data that is updated frequently from the host

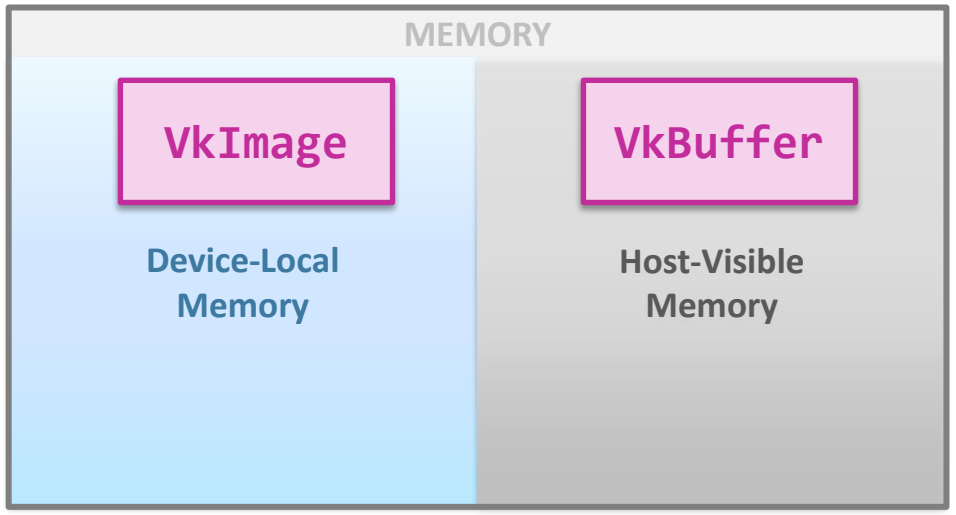
VK_MEMORY_PROPERTY_DEVICE_LOCAL_BIT VK_MEMORY_PROPERTY_HOST_VISIBLE_BIT

Memory

VK_DESCRIPTOR_TYPE_**SAMPLED_IMAGE**
VK_DESCRIPTOR_TYPE_**STORAGE_IMAGE**

VK_DESCRIPTOR_TYPE_**UNIFORM_BUFFER**

Good for data which should be read fast, or which can be up-dated on the device →



← Good for data that is updated frequently from the host

VK_MEMORY_PROPERTY_DEVICE_LOCAL_BIT

VK_MEMORY_PROPERTY_HOST_VISIBLE_BIT

PART 4

- Command Buffer Allocation
- Memory
- **Image Layout Transitions**
- Synchronization

Image Layout Transitions

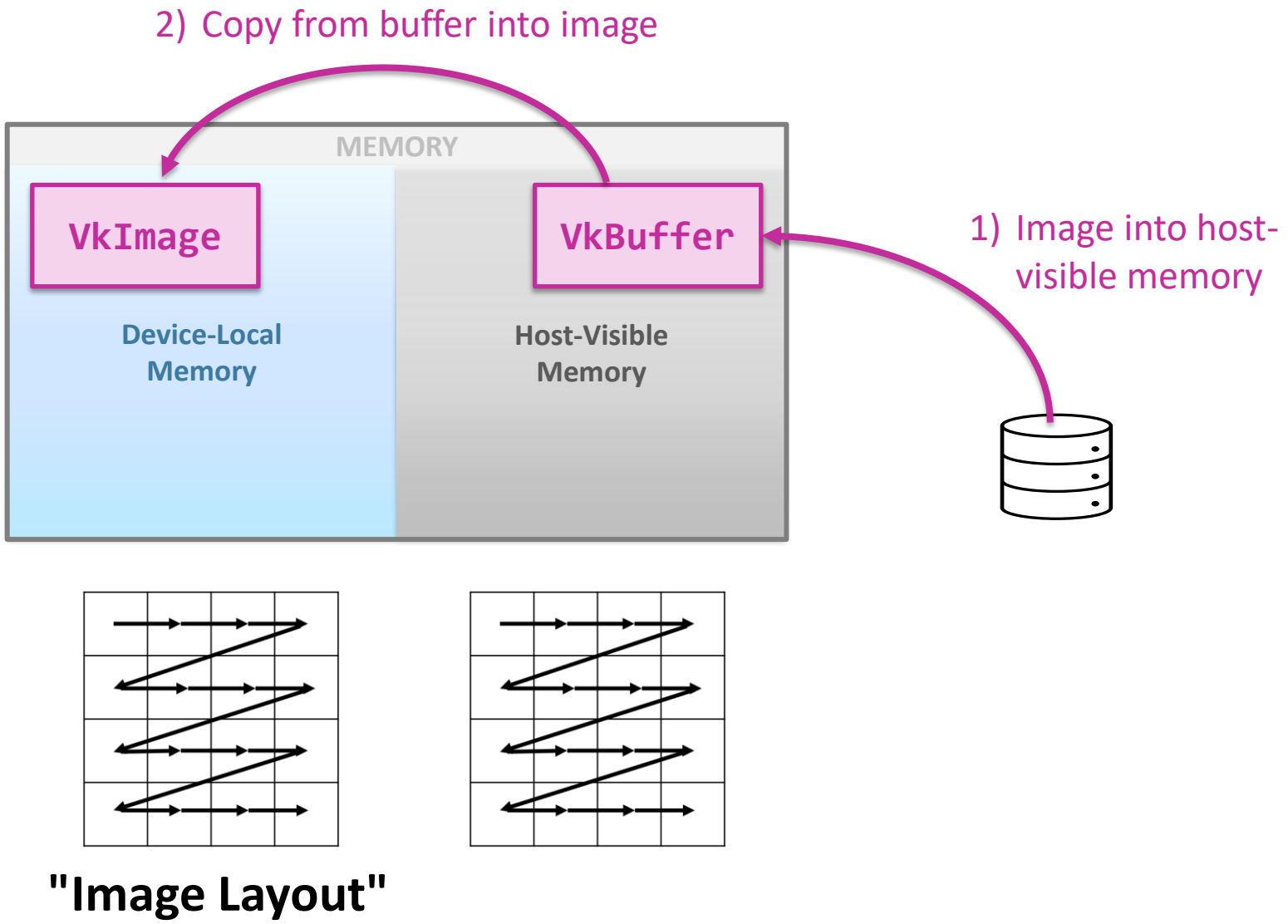
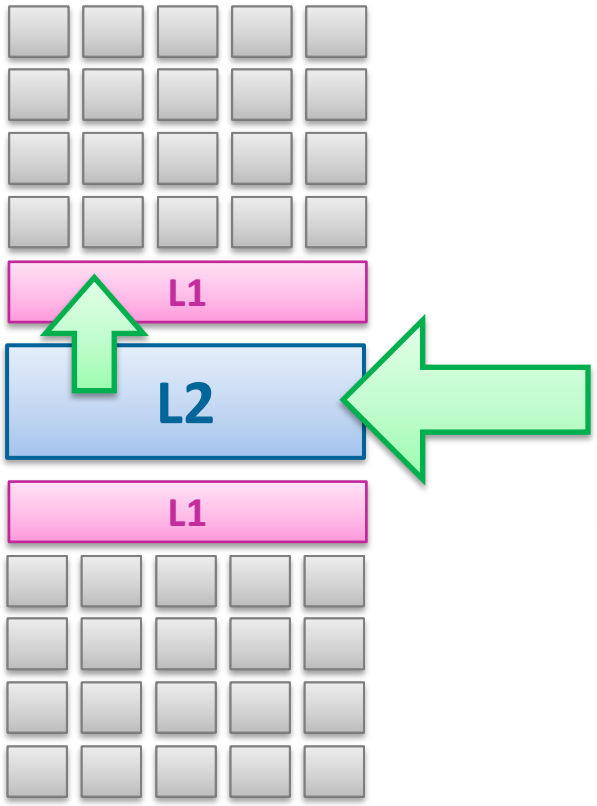
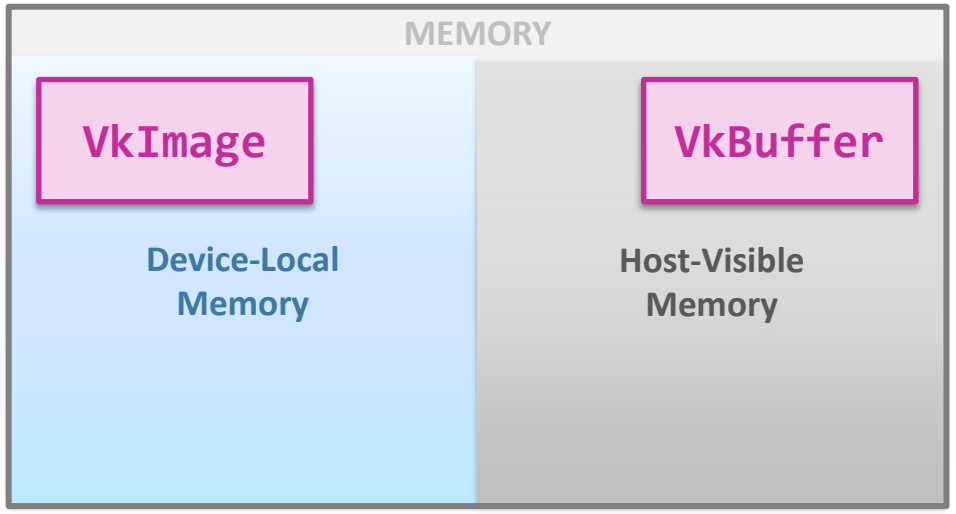
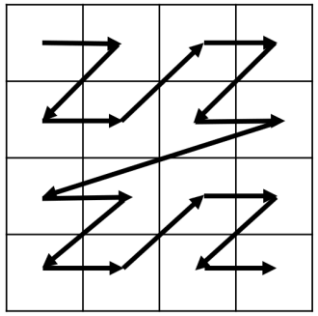
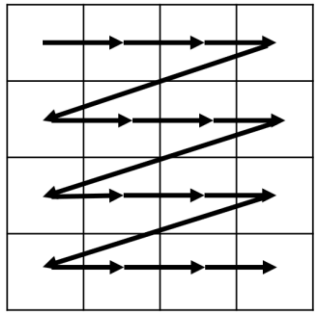


Image Layout Transitions









"Image Layout"

Note:
Doesn't have to be canging the storage order!
Can also mean some kind of (vendor-specific) compression.

Image Layout Transitions

- You **don’t** specify the exact memory layout
- But the usage scenario
  - Concrete memory layout can be vendor-specific
  - Concrete transition is implemented in drivers/in hardware
  - You specify the **usage scenario**
  - Vendor does the right thing

Image Layout Transitions

- Image usage descriptions via layouts: `VkImageLayout`

```
typedef enum VkImageLayout {
    VK_IMAGE_LAYOUT_UNDEFINED = 0,
    VK_IMAGE_LAYOUT_GENERAL = 1,
    VK_IMAGE_LAYOUT_COLOR_ATTACHMENT_OPTIMAL = 2,
    VK_IMAGE_LAYOUT_DEPTH_STENCIL_ATTACHMENT_OPTIMAL = 3,
    VK_IMAGE_LAYOUT_DEPTH_STENCIL_READ_ONLY_OPTIMAL = 4,
    VK_IMAGE_LAYOUT_SHADER_READ_ONLY_OPTIMAL = 5,
    VK_IMAGE_LAYOUT_TRANSFER_SRC_OPTIMAL = 6,
    VK_IMAGE_LAYOUT_TRANSFER_DST_OPTIMAL = 7,
    ...
} VkImageLayout;
```


Vulkanised 2023

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

PART 4

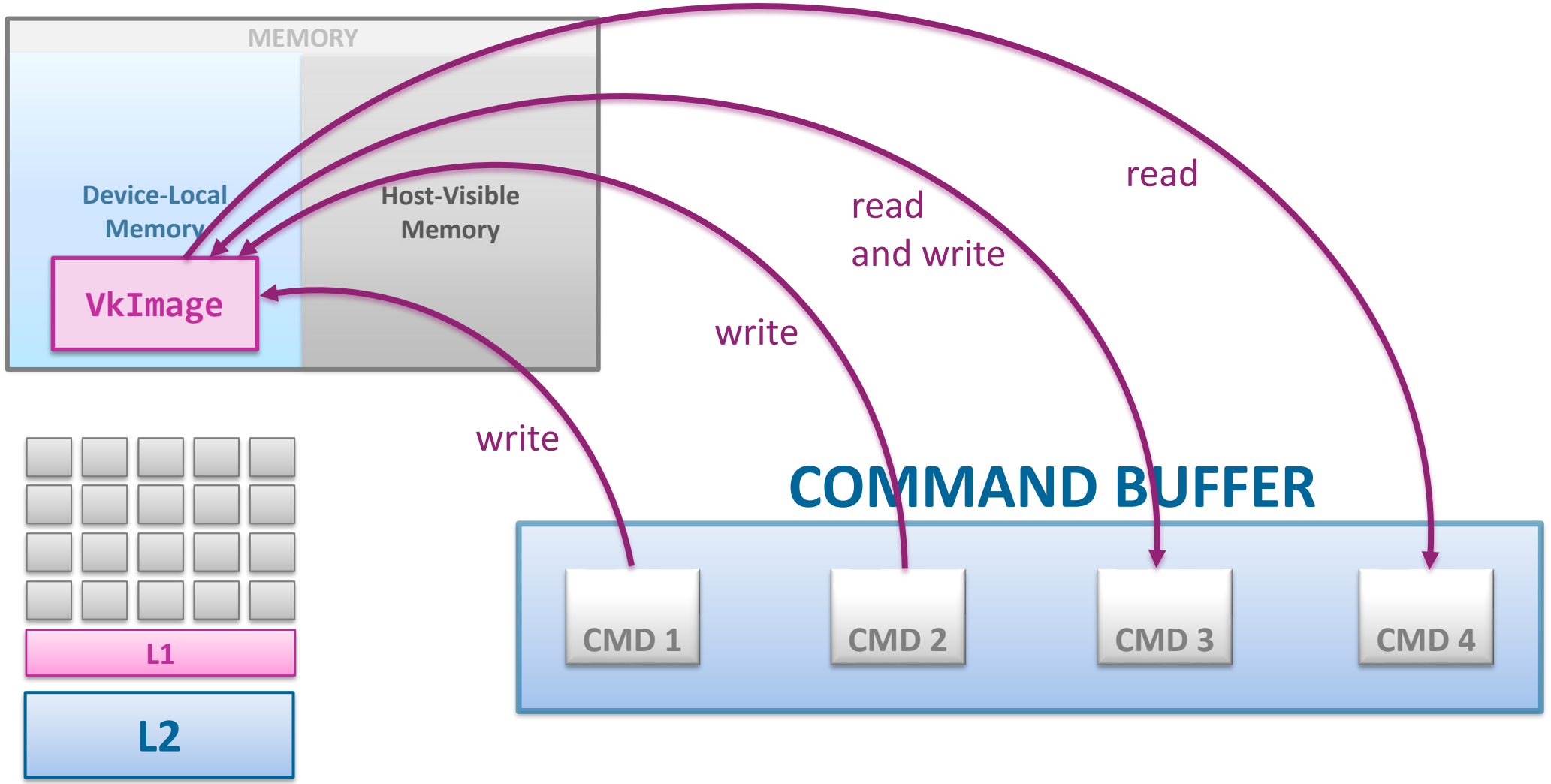
- Command Buffer Allocation
- Memory
- **Image Layout Transitions**
- Synchronization

Platinum Sponsors: AMD arm Google LUNARG KHRONOS GROUP SAMSUNG

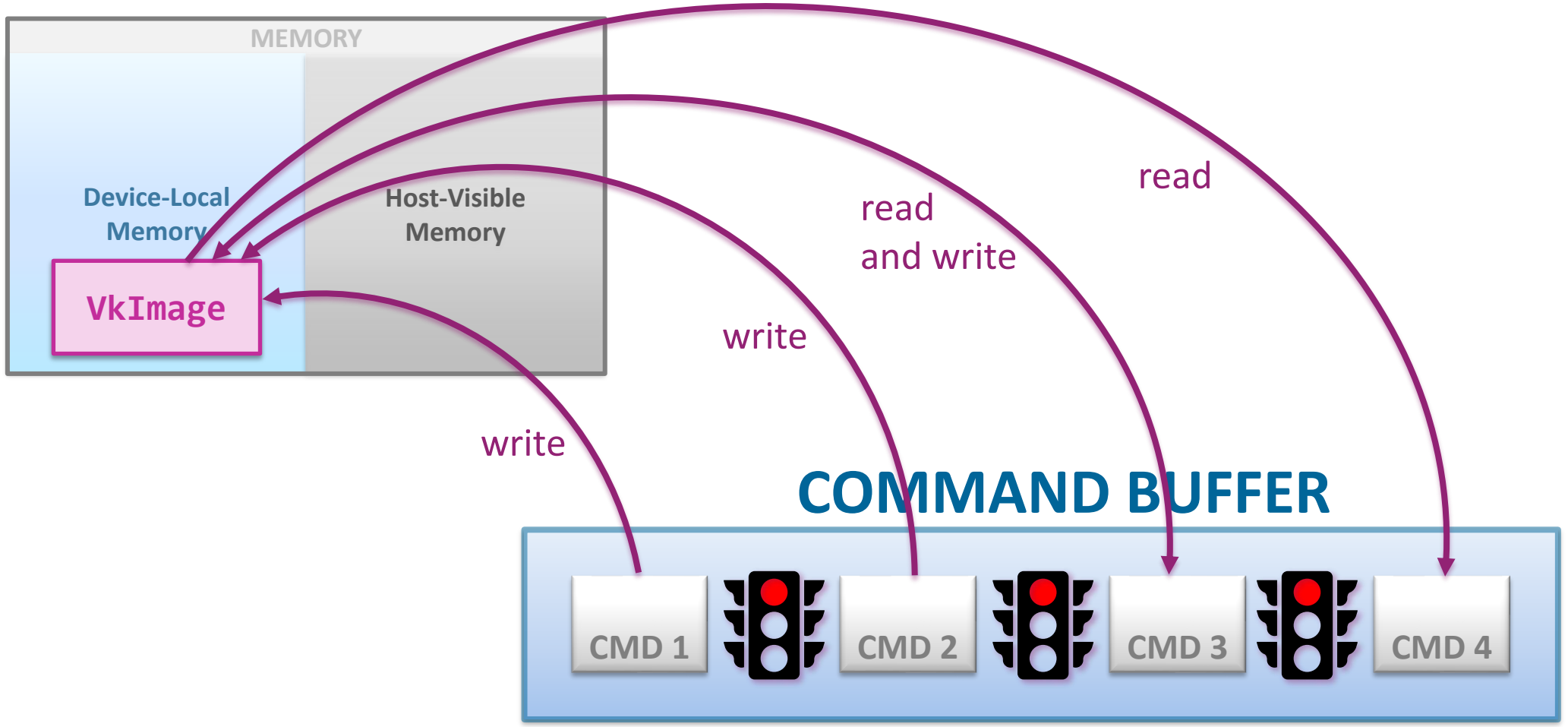
PART 4

- Command Buffer Allocation
- Memory
- Image Layout Transitions
- **Synchronization**

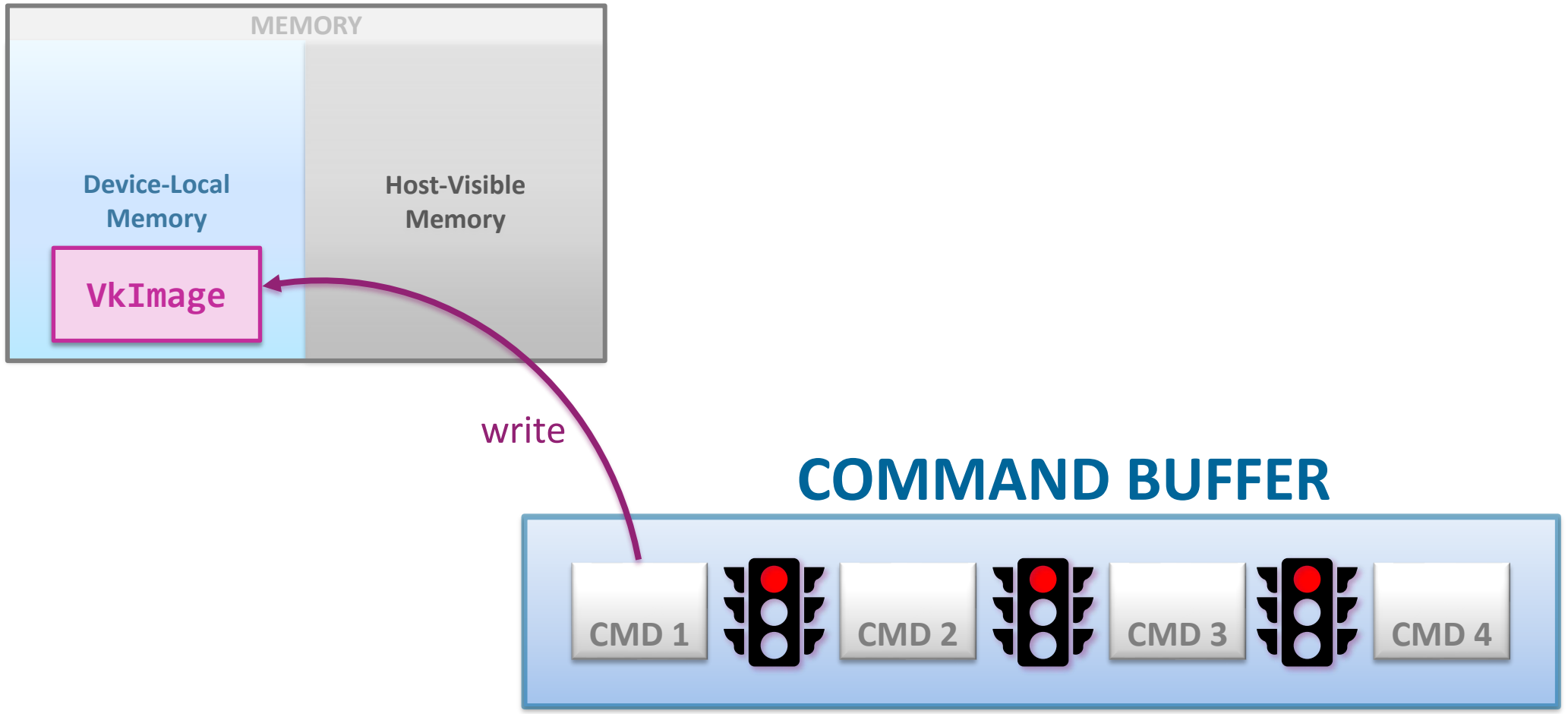
Synchronization



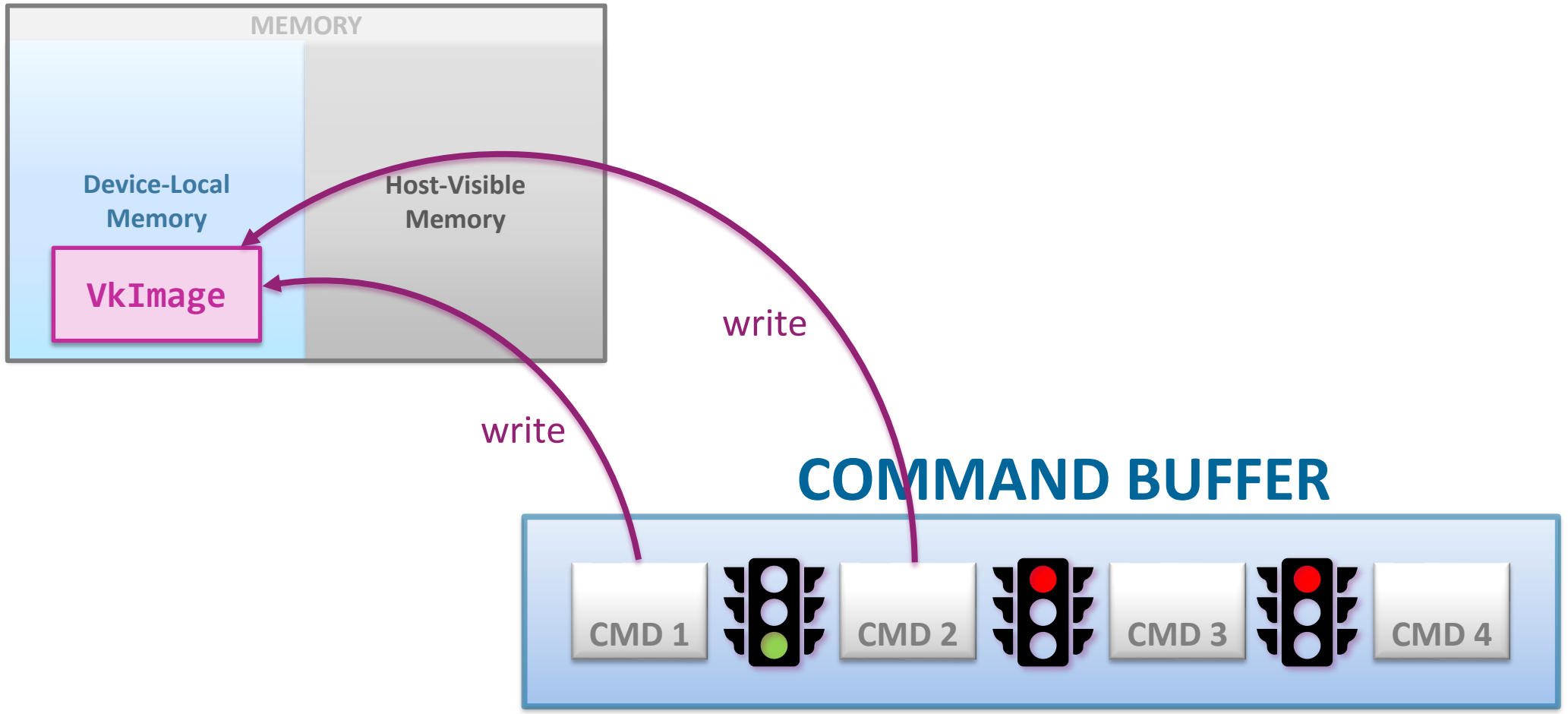
Synchronization



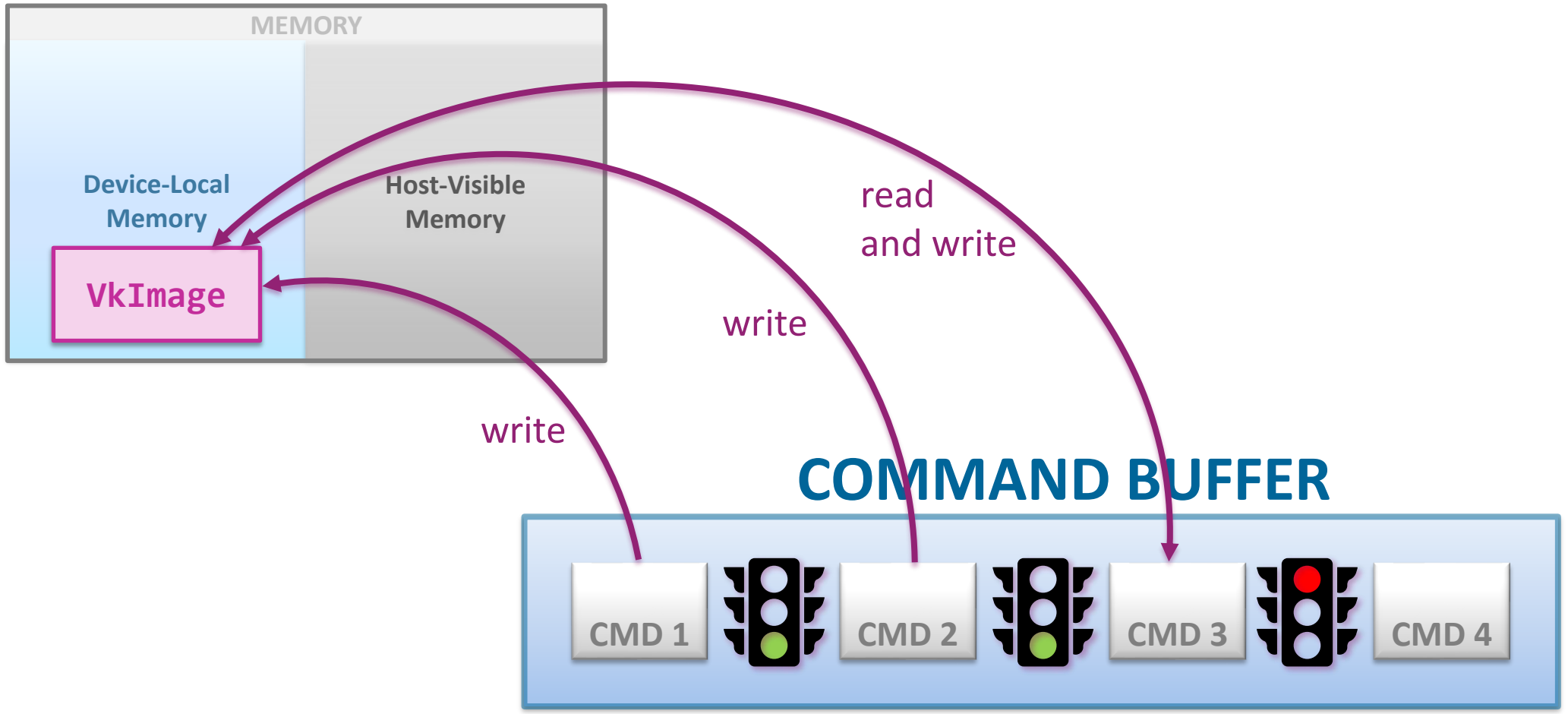
Synchronization



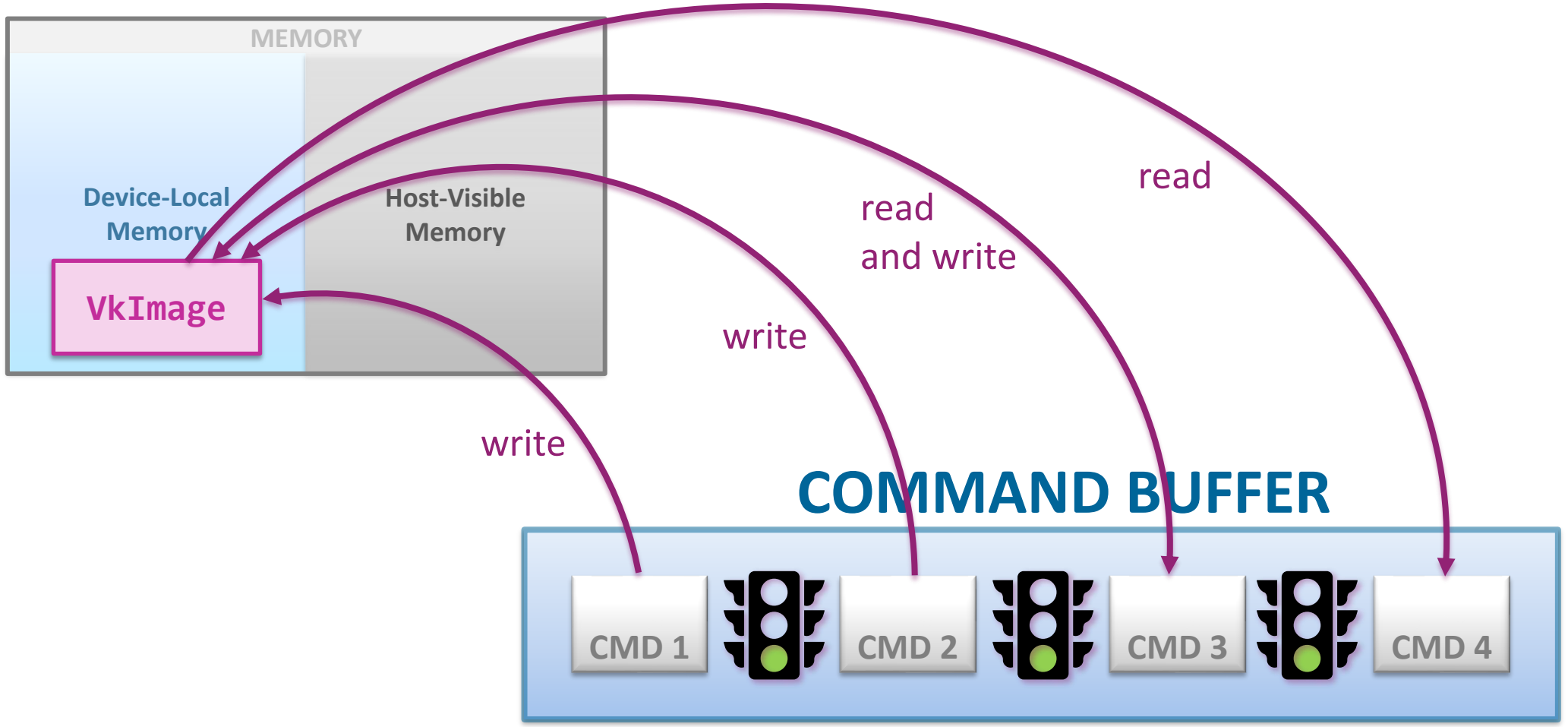
Synchronization



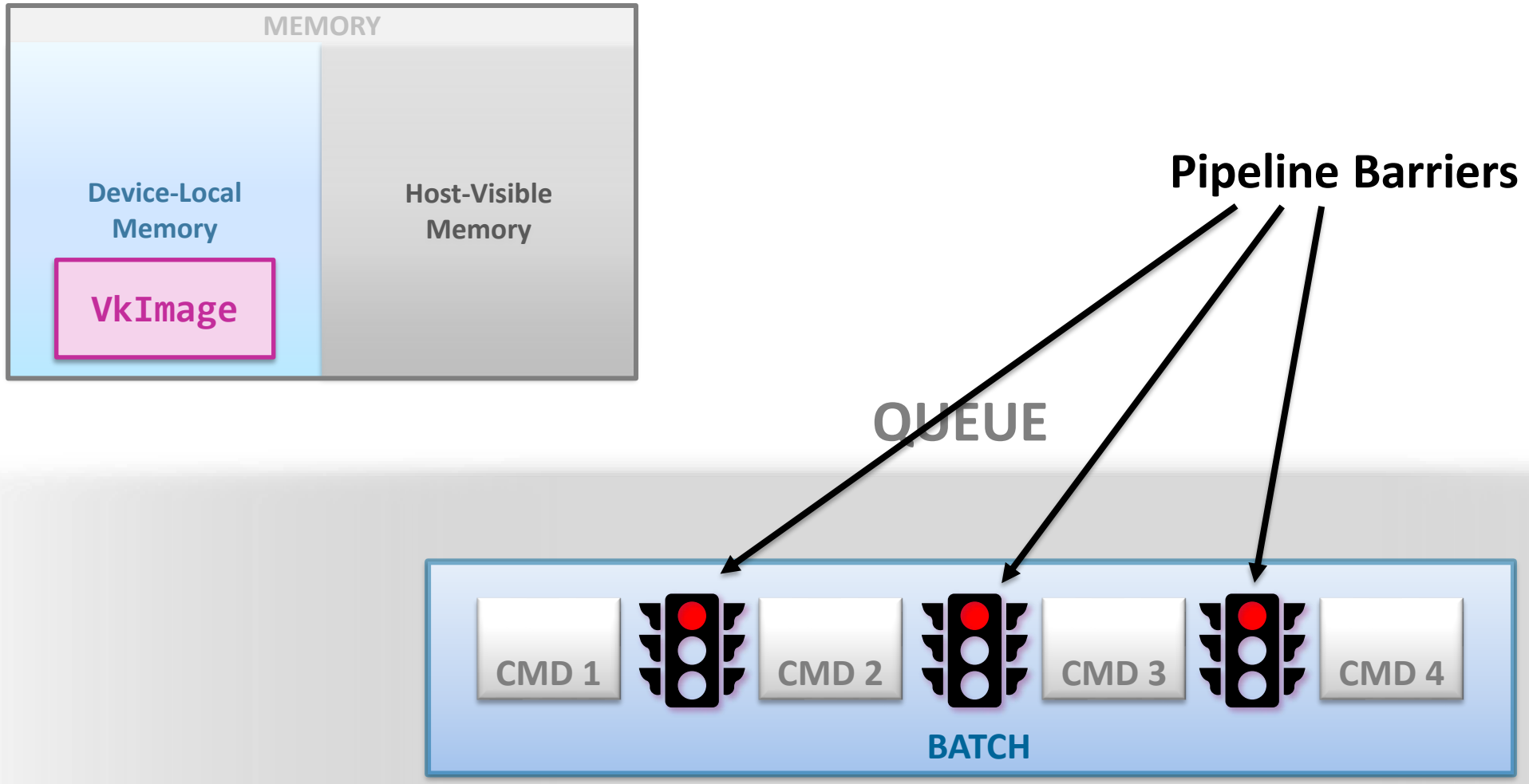
Synchronization



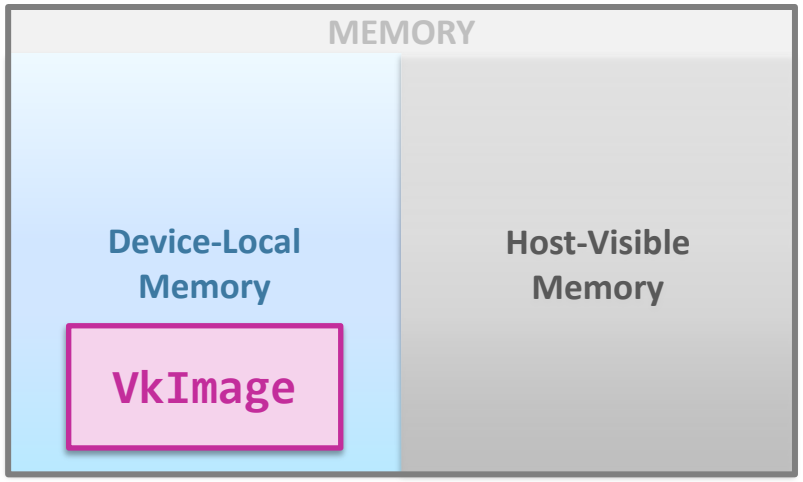
Synchronization



Synchronization

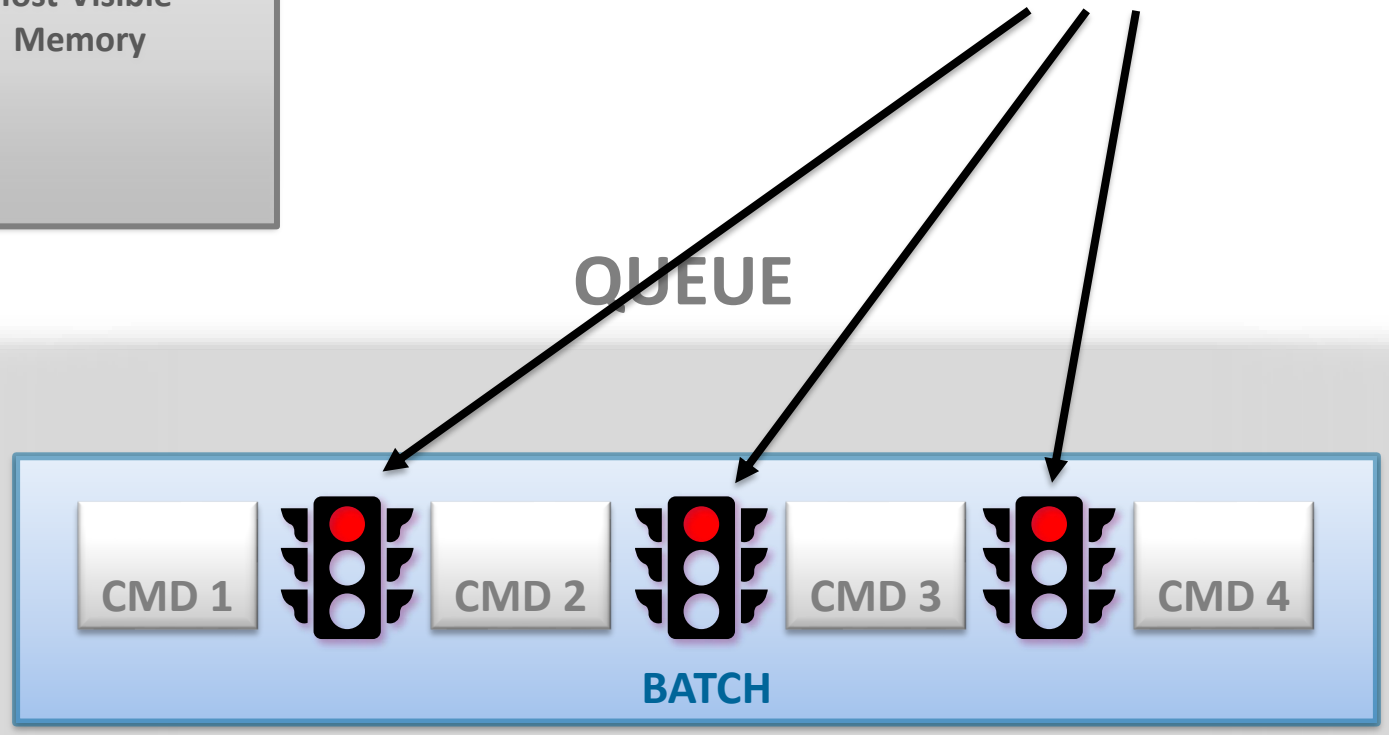


Synchronization

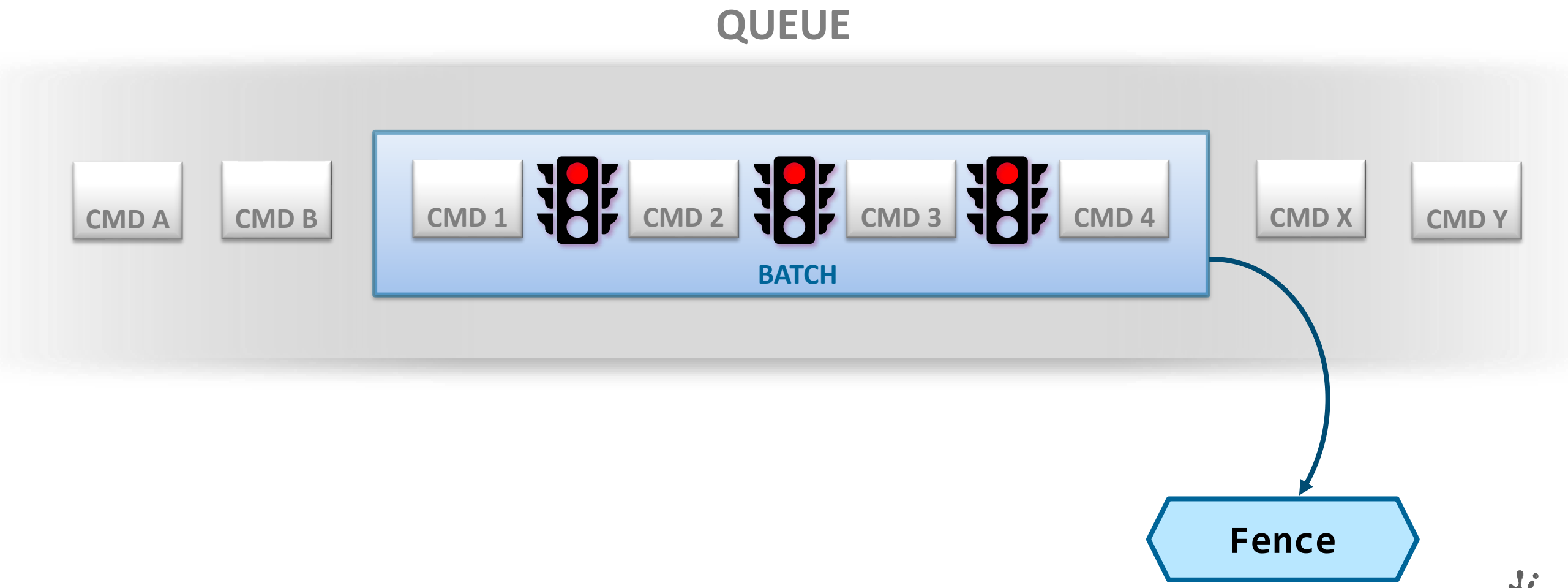


Memory Barriers

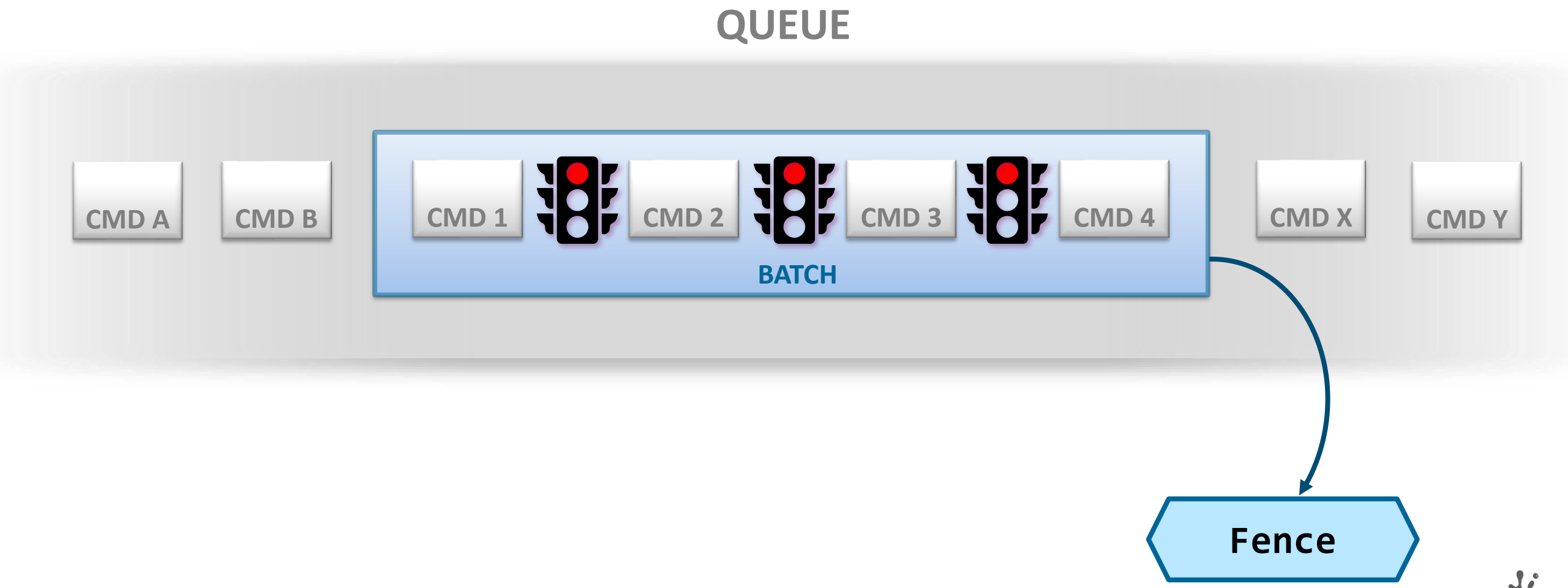
= Pipeline Barrier + Memory Access



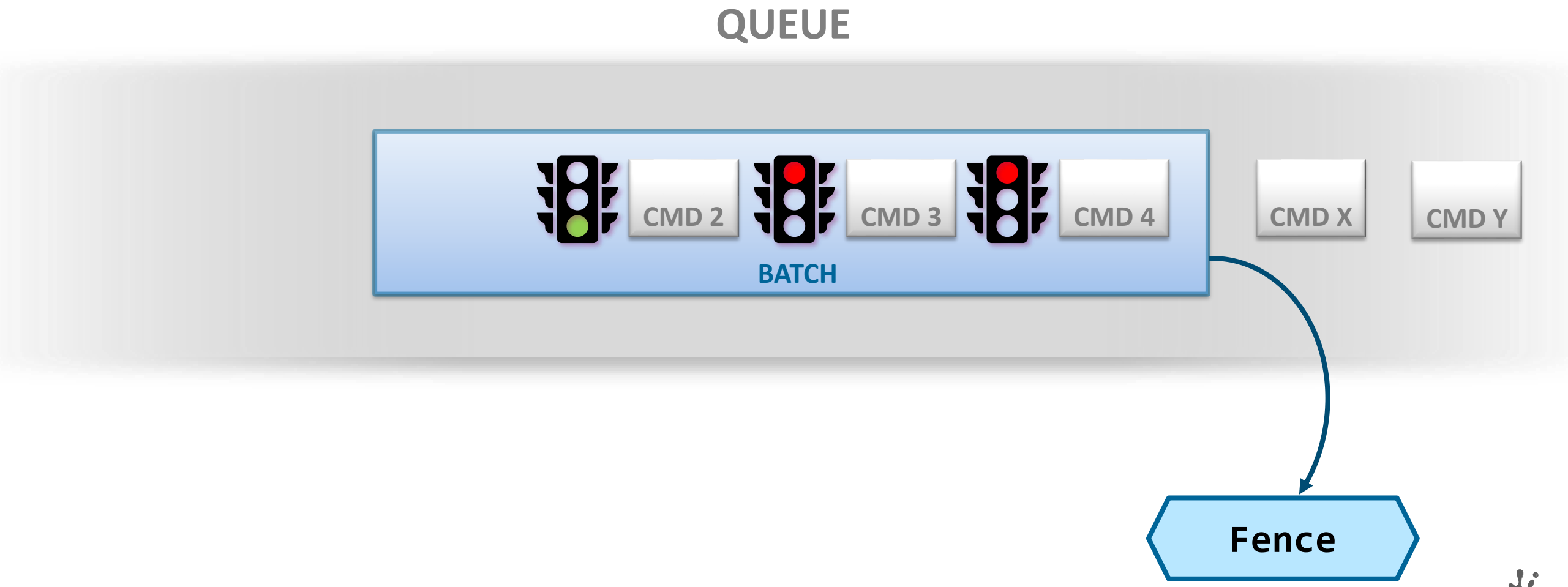
Synchronization



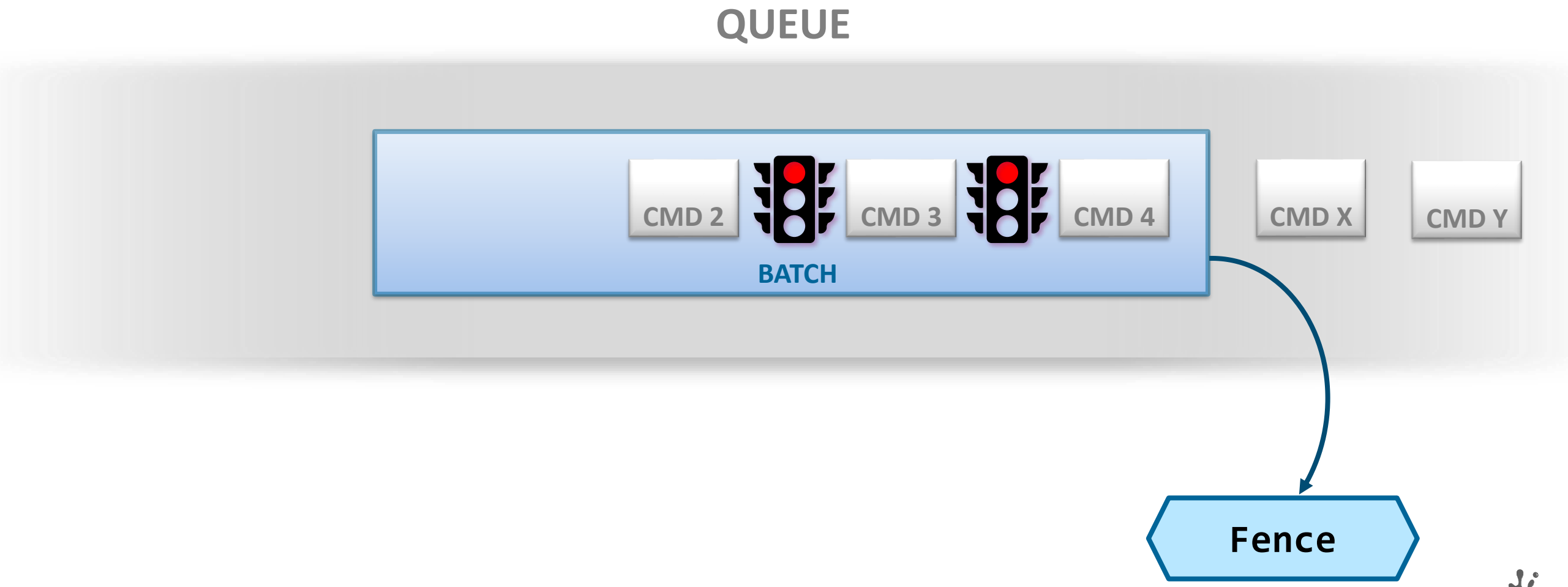
Synchronization



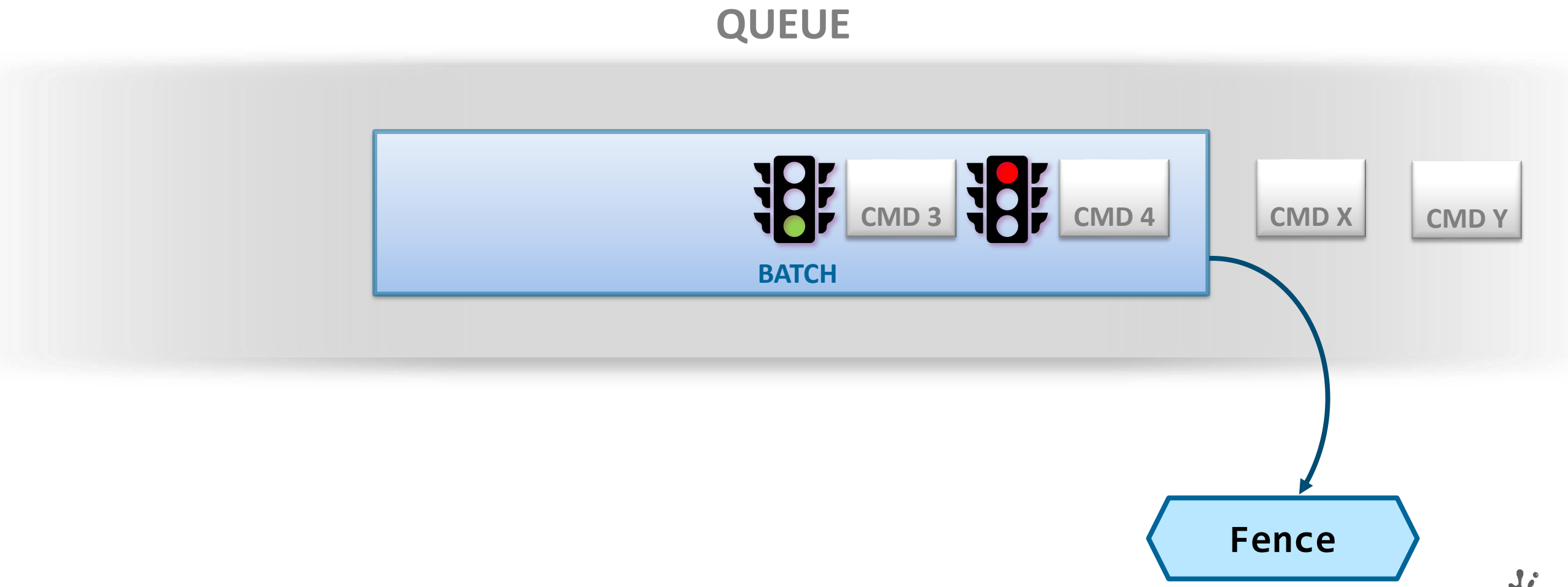
Synchronization



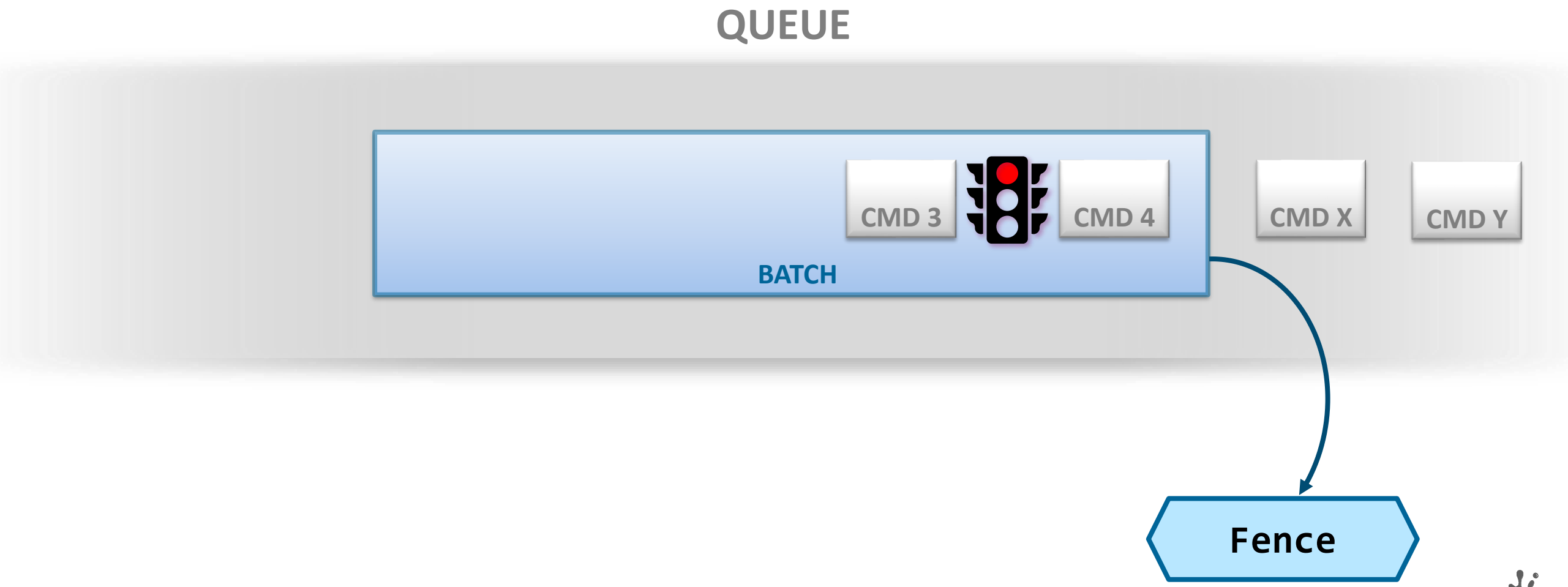
Synchronization



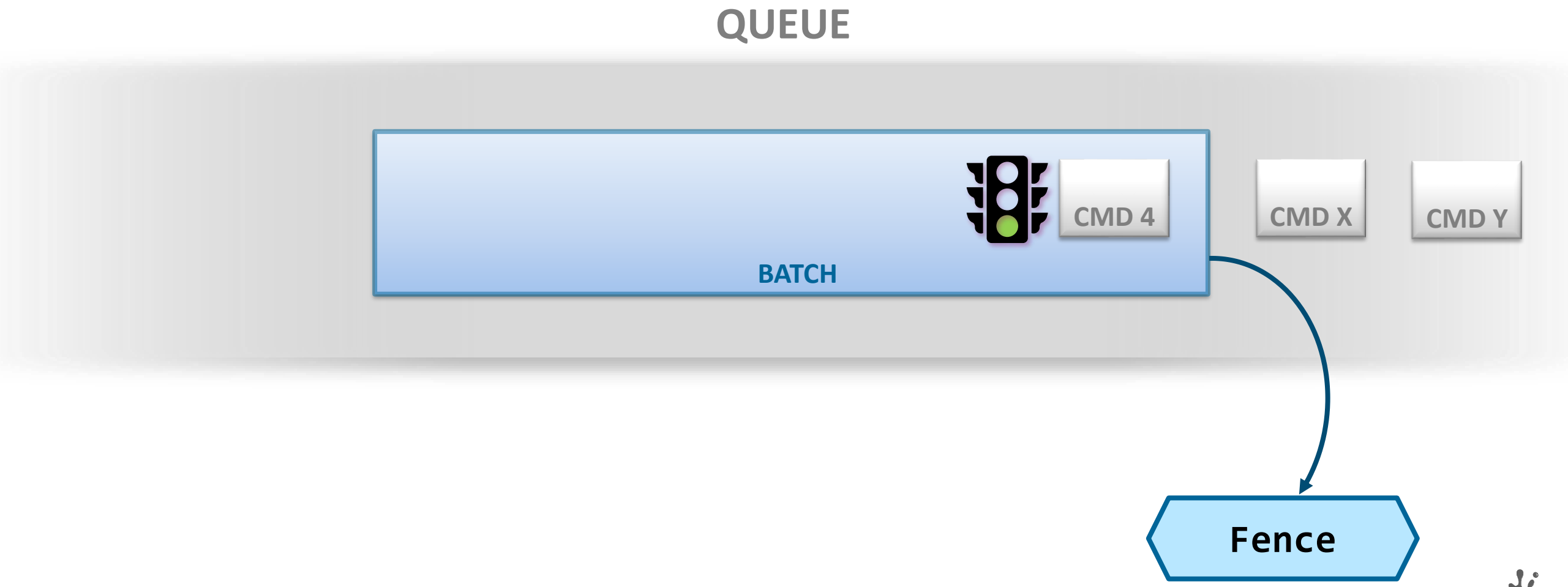
Synchronization



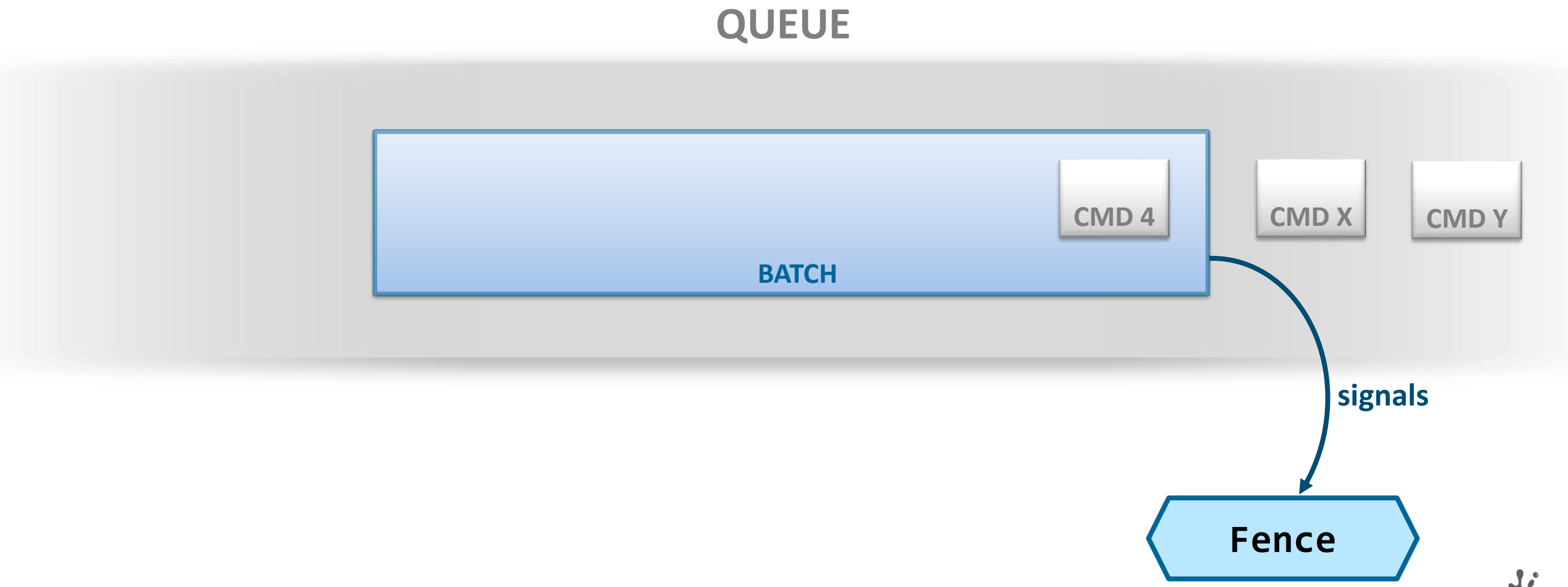
Synchronization



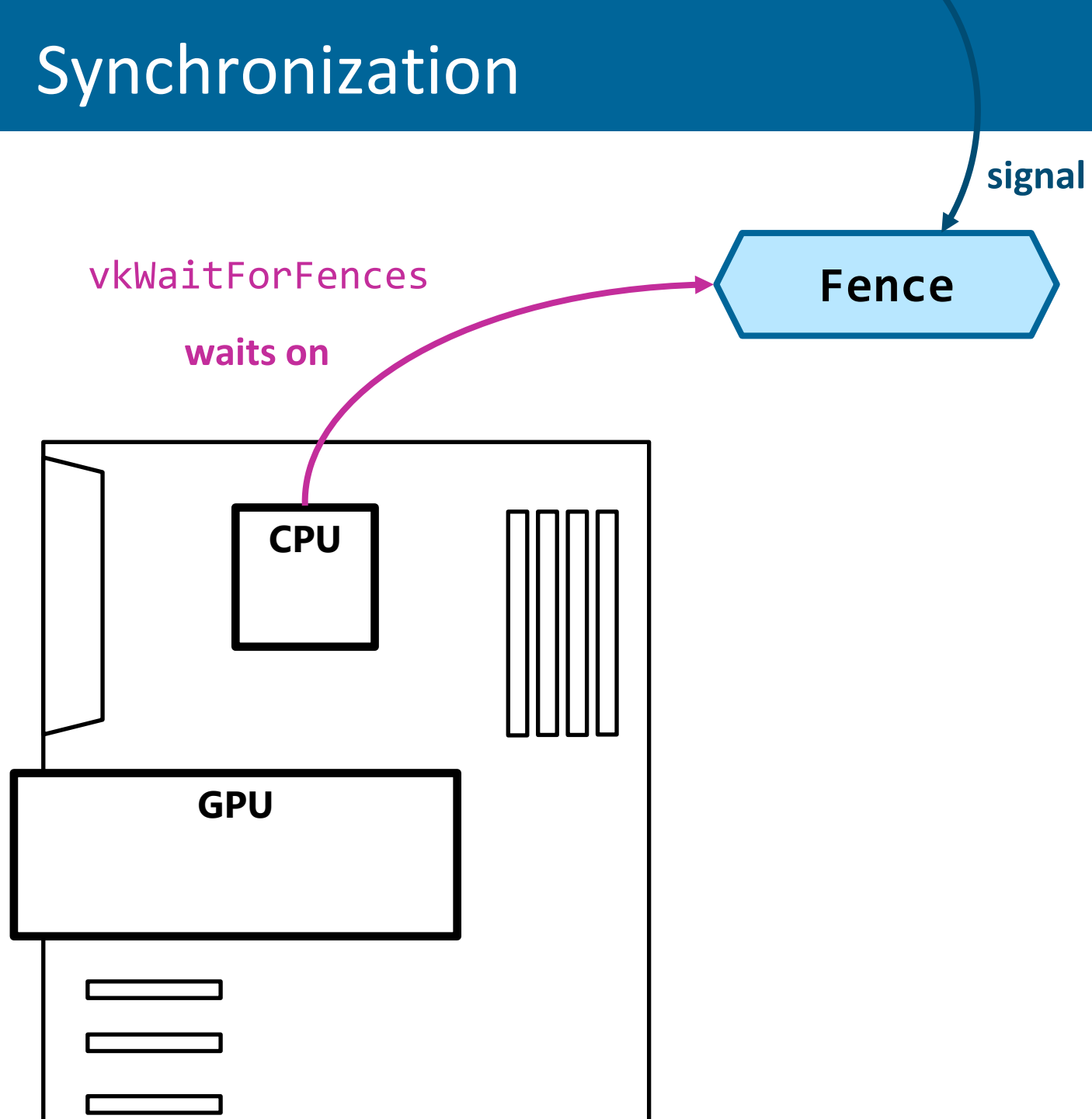
Synchronization



Synchronization



Synchronization



Overview of Synchronization Methods

- Wait Idle Operations
- Fences
- Semaphores
  - Binary Semaphores
  - Timeline Semaphores
- Pipeline Barriers
  - Execution Barriers
  - Memory Barriers
- Render Pass Subpass Dependencies
- Events

Image Memory Barriers + Image Layout Transitions

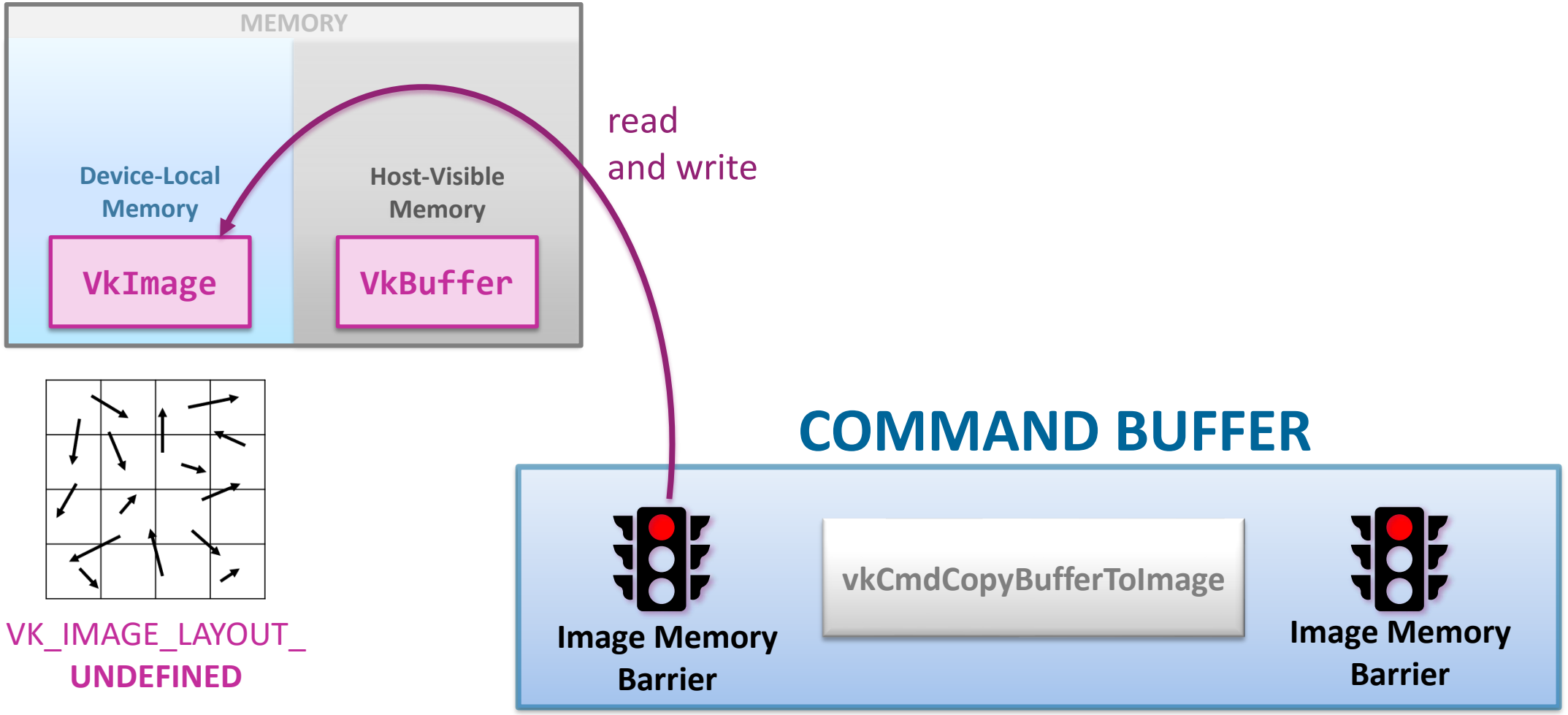
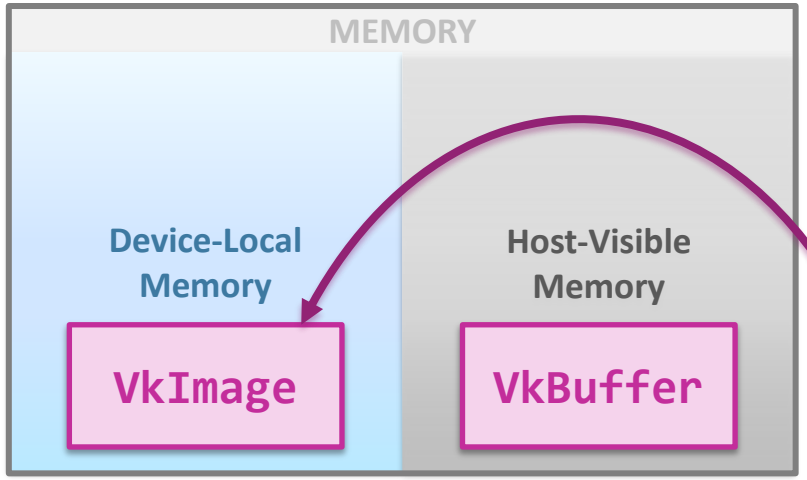
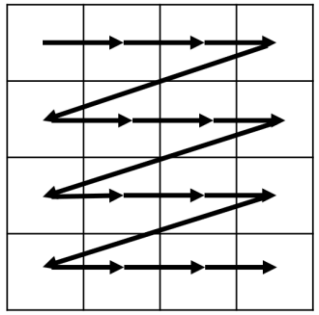


Image Memory Barriers + Image Layout Transitions



read
and write



VK_IMAGE_LAYOUT_
TRANSFER_DST_OPTIMAL

COMMAND BUFFER

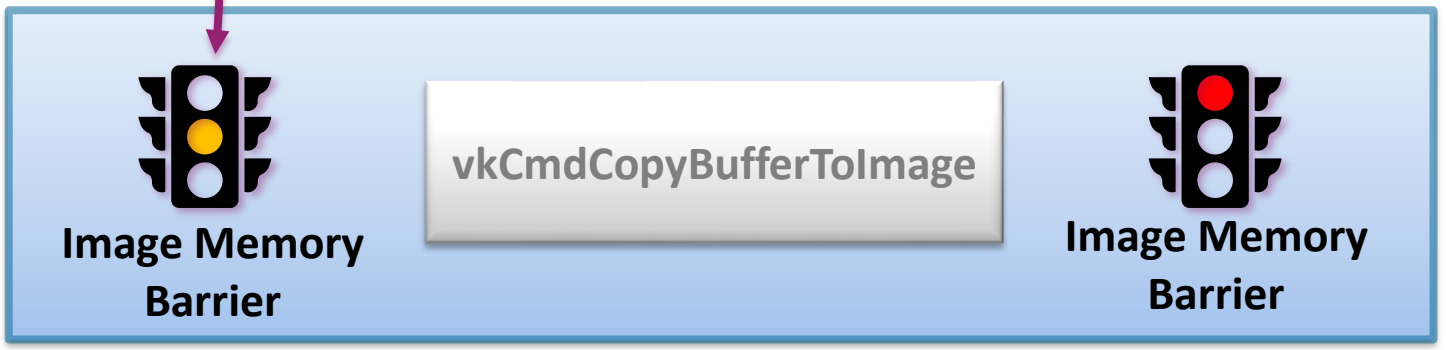


Image Memory Barriers + Image Layout Transitions

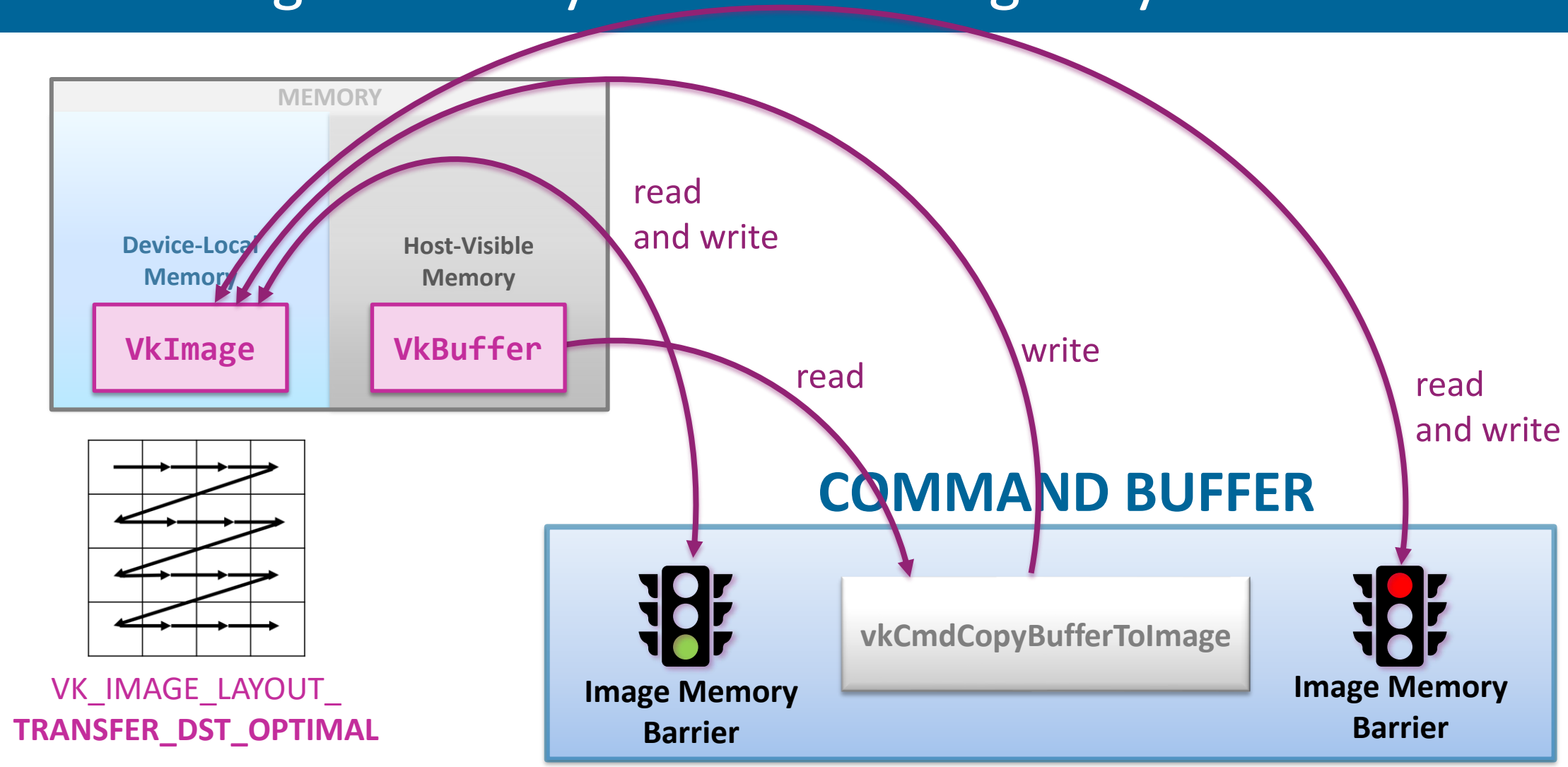


Image Memory Barriers + Image Layout Transitions

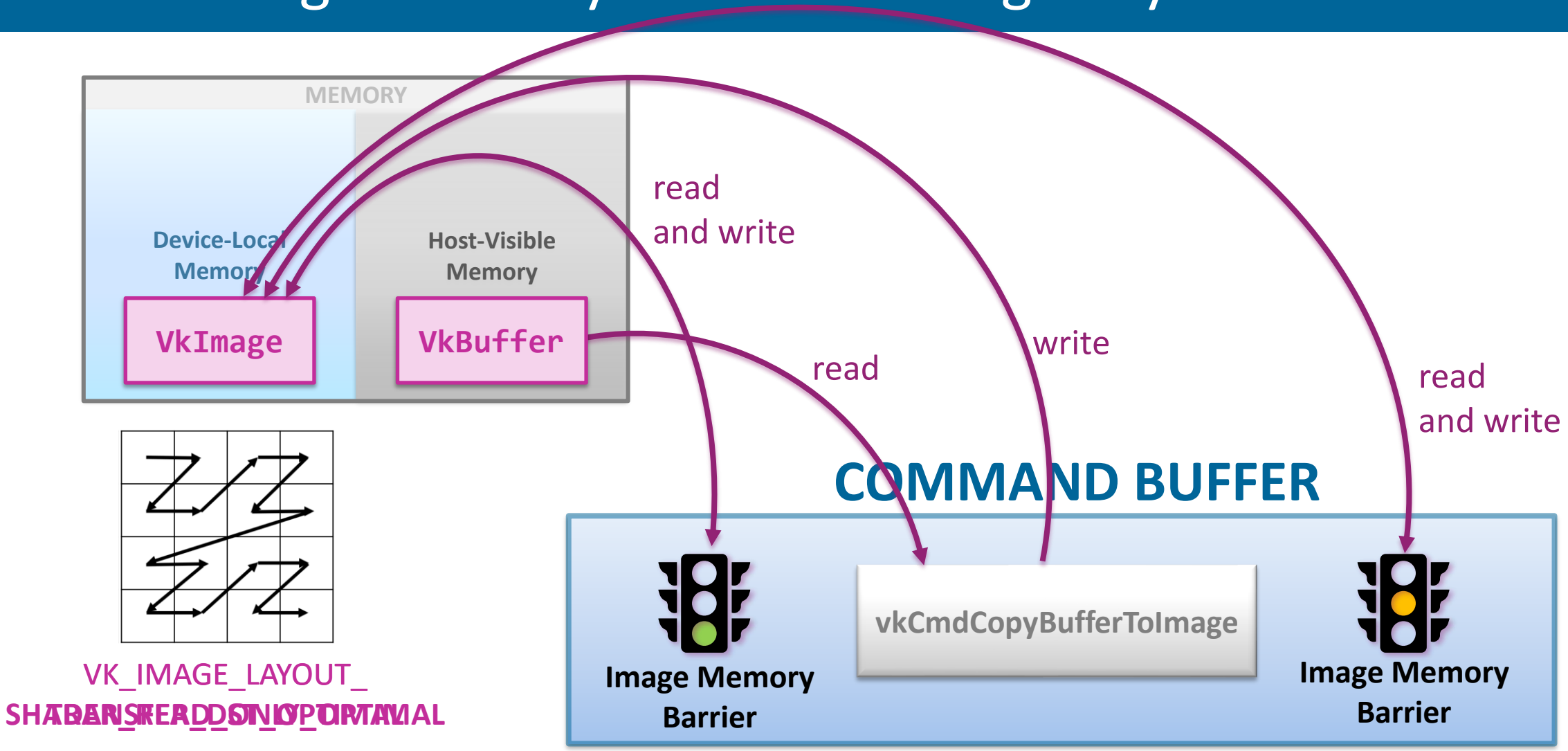
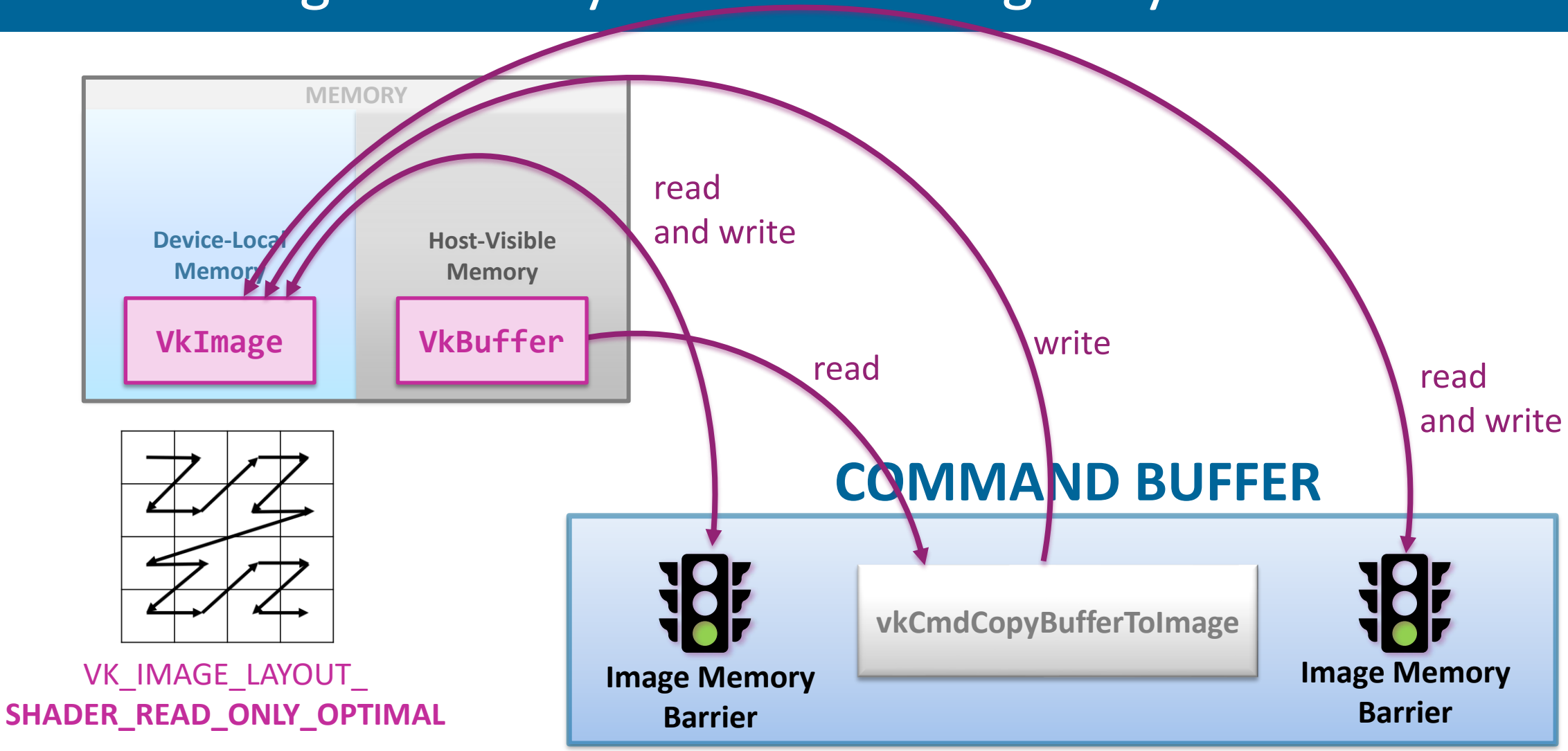


Image Memory Barriers + Image Layout Transitions



Vulkanised 2023

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

PART 4

- Command Buffer Allocation
- Memory
- Image Layout Transitions
- **Synchronization**

Platinum Sponsors: AMD arm Google LUNARG KHRONOS GROUP SAMSUNG

Vulkanised 2023

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

PART 4

- Command Buffer Allocation
- Memory
- Image Layout Transitions
- Synchronization

Platinum Sponsors: AMD arm Google LUNARG KHRONOS GROUP SAMSUNG

Schedule

PART 1:

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

PART 2:

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

Lunch Break 13:00 – 14:00

PART 3:

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

PART 4:

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