Algorithms for Real-Time Rendering

186.192, 2022S, 3.0ECTS

Vulkan Lecture Series, Episode 7:

Synchronization

Johannes Unterguggenberger

Institute of Visual Computing & Human-Centered Technology

TU Wien, Austria

Recap: Commands

Graphics Pipeline Commands:

vkCmdDraw
vkCmdDrawIndexed
vkCmdDrawIndirect

vkCmdDrawMeshTasksNV

vkCmdClearAttachments

Compute Pipeline Commands:

vkCmdDispatch
vkCmdDispatchIndirect

Ray Tracing Pipeline Commands:

vkCmdTraceRaysKHR
vkCmdTraceRaysIndirectKHR

Transfer Commands:

vkCmdCopyBuffer
vkCmdCopyImage
vkCmdCopyBufferToImage
vkCmdCopyImageToBuffer
vkCmdCopyAccelerationStructureKHR

vkCmdBlitImage

vkCmdResolveImage

vkCmdClearColorImage

Ray-Tracing Acceleration Structure Build Commands:

vkCmdBuildAccelerationStructuresKHR
vkCmdBuildAccelerationStructuresIndirectKHR

Synchronization Commands:

vkCmdPipelineBarrier[2]
vkCmdSetEvent[2]
vkCmdEndRenderPass[2]
vkCmdBeginRenderPass[2]
vkCmdEndRenderPass[2]

Bind Commands:

vkCmdBindDescriptorSets
vkCmdBindPipeline
vkCmdBindVertexBuffers
vkCmdBindIndexBuffer

Other Commands:

vkCmdPushConstants
vkCmdPushDescriptorSetKHR
vkCmdSetScissor
vkCmdSetViewport
vkCmdSetDepthBias
...

Recap: Commands

Graphics Pipeline Commands:

vkCmdDraw
vkCmdDrawIndexed
vkCmdDrawIndirect

vkCmdDrawMeshTasksNV

vkCmdClearAttachments

Compute Pipeline Commands:

vkCmdDispatch
vkCmdDispatchIndirect

Ray Tracing Pipeline Commands:

vkCmdTraceRaysKHR
vkCmdTraceRaysIndirectKHR

Transfer Commands:

vkCmdCopyBuffer
vkCmdCopyImage
vkCmdCopyBufferToImage
vkCmdCopyImageToBuffer
vkCmdCopyAccelerationStructureKHR

vkCmdBlitImage

vkCmdResolveImage

vkCmdClearColorImage

Ray-Tracing Acceleration Structure Build Commands:

vkCmdBuildAccelerationStructuresKHR
vkCmdBuildAccelerationStructuresIndirectKHR

Synchronization Commands:

vkCmdPipelineBarrier[2]
vkCmdSetEvent[2]
vkCmdEndRenderPass[2]
vkCmdBeginRenderPass[2]
vkCmdEndRenderPass[2]

STATE-Type

Bind Commands:

vkCmdBindDescriptorSets
vkCmdBindPipeline
vkCmdBindVertexBuffers
vkCmdBindIndexBuffer

Other Commands:

vkCmdPushConstants
vkCmdPushDescriptorSetKHR
vkCmdSetScissor
vkCmdSetViewport
vkCmdSetDepthBias
...

Recap: Commands

Graphics Pipeline Commands:

vkCmdDraw
vkCmdDrawIndexed
vkCmdDrawIndirect

vkCmdDrawMeshTasksNV

vkCmdClearAttachments

Ray-Tracing Acceleration Structure Build Commands:

vkCmdBuildAccelerationStructuresKHR
vkCmdBuildAccelerationStructuresIndirectKHR

Compute Pipeline Commands:

vkCmdDispatch
vkCmdDispatchIndirect

Ray Tracing Pipeline Commands:

vkCmdTraceRaysKHR
vkCmdTraceRaysIndirectKHR

Transfer Commands:

vkCmdCopyBuffer
vkCmdCopyImage
vkCmdCopyBufferToImage
vkCmdCopyImageToBuffer
vkCmdCopyAccelerationStructureKHR

vkCmdBlitImage

vkCmdResolveImage

vkCmdClearColorImage

ACTION-Type

Synchronization Commands:

vkCmdPipelineBarrier[2]
vkCmdSetEvent[2]
vkCmdEndRenderPass[2]
vkCmdBeginRenderPass[2]
vkCmdEndRenderPass[2]

Bind Commands:

vkCmdBindDescriptorSets
vkCmdBindPipeline
vkCmdBindVertexBuffers
vkCmdBindIndexBuffer

Other Commands:

vkCmdPushConstants
vkCmdPushDescriptorSetKHR
vkCmdSetScissor
vkCmdSetViewport
vkCmdSetDepthBias
...

Commands

Graphics Pipeline Commands:

vkCmdDraw
vkCmdDrawIndexed
vkCmdDrawIndirect

vkCmdDrawMeshTasksNV

vkCmdClearAttachments

Compute Pipeline Commands:

vkCmdDispatch
vkCmdDispatchIndirect

Ray Tracing Pipeline Commands:

vkCmdTraceRaysKHR
vkCmdTraceRaysIndirectKHR

Transfer Commands:

vkCmdCopyBuffer
vkCmdCopyImage
vkCmdCopyBufferToImage
vkCmdCopyImageToBuffer
vkCmdCopyAccelerationStructureKHR

vkCmdBlitImage

vkCmdResolveImage

vkCmdClearColorImage

Ray-Tracing Acceleration Structure Build Commands:

vkCmdBuildAccelerationStructuresKHR
vkCmdBuildAccelerationStructuresIndirectKHR

SYNC-Type

Synchronization Commands:

vkCmdPipelineBarrier[2]
vkCmdSetEvents[2]
vkCmdWaitEvents[2]
vkCmdBeginRenderPass[2]
vkCmdEndRenderPass[2]

Bind Commands:

vkCmdBindDescriptorSets
vkCmdBindPipeline
vkCmdBindVertexBuffers
vkCmdBindIndexBuffer

Other Commands:

vkCmdPushConstants
vkCmdPushDescriptorSetKHR
vkCmdSetScissor
vkCmdSetViewport
vkCmdSetDepthBias
...

Recap: Commands

Graphics Pipeline Commands:

vkCmdDraw
vkCmdDrawIndexed
vkCmdDrawIndirect

vkCmdDrawMeshTasksNV

vkCmdClearAttachments

Compute Pipeline Commands:

vkCmdDispatch
vkCmdDispatchIndirect

Ray Tracing Pipeline Commands:

vkCmdTraceRaysKHR
vkCmdTraceRaysIndirectKHR

Transfer Commands:

vkCmdCopyBuffer
vkCmdCopyImage
vkCmdCopyBufferToImage
vkCmdCopyImageToBuffer
vkCmdCopyAccelerationStructureKHR

vkCmdBlitImage

vkCmdResolveImage

vkCmdClearColorImage

Ray-Tracing Acceleration Structure Build Commands:

vkCmdBuildAccelerationStructuresKHR
vkCmdBuildAccelerationStructuresIndirectKHR

ACTION-Type

Synchronization Commands:

vkCmdPipelineBarrier[2]
vkCmdSetEvent[2]
vkCmdEndRenderPass[2]
vkCmdBeginRenderPass[2]
vkCmdEndRenderPass[2]

Bind Commands:

vkCmdBindDescriptorSets
vkCmdBindPipeline
vkCmdBindVertexBuffers
vkCmdBindIndexBuffer

Other Commands:

vkCmdPushConstants
vkCmdPushDescriptorSetKHR
vkCmdSetScissor
vkCmdSetViewport
vkCmdSetDepthBias
...

Recap: Pipeline Stages

The Stages of…

Graphics Pipelines:
CONDITIONAL_RENDERING(EXT)
DRAW_INDIRECT
INDEX_INPUT(KHR)
VERTEX_ATTRIBUTE_INPUT(KHR)
VERTEX_SHADER
TESSELLATION_CONTROL_SHADER
TESSELLATION_EVALUATION_SHADER
GEOMETRY_SHADER
TRANSFORM_FEEDBACK(EXT)
FRAGMENT_SHADING_RATE(KHR)
EARLY_FRAGMENT_TESTS
FRAGMENT_SHADER
LATE_FRAGMENT_TESTS
COLOR_ATTACHMENT_OUTPUT

Compute Pipelines:
CONDITIONAL_RENDERING(EXT)
DRAW_INDIRECT
COMPUTE_SHADER

Ray Tracing Pipelines:
DRAW_INDIRECT
RAY_TRACING_SHADER(KHR)

Ray-Tracing Acceleration Structure Build Commands:
ACCELERATION_STRUCTURE_BUILD(KHR)

Copy Commands:
COPY

Blit Commands:
BLIT

Resolve Commands:
RESOLVE

Clear Commands:
CLEAR

None/Aggregate Stages:
NONE
ALL_GRAPHICS
ALL_TRANSFER
ALL_COMMANDS

Full List of Pipeline Stages:
VkPipelineStageFlagBits2
The Khronos Group. Vulkan 1.3.212 Specificaton

Pipeline Stages

The Stages of…

Graphics Pipelines:

CONDITIONAL_RENDERING(EXT)
DRAW_INDIRECT
INDEX_INPUT(KHR)
VERTEX_ATTRIBUTE_INPUT(KHR)
VERTEX_SHADER
TESSELLATION_CONTROL_SHADER
TESSELLATION_EVALUATION_SHADER
GEOMETRY_SHADER
TRANSFORM_FEEDBACK(EXT)
FRAGMENT_SHADING_RATE(KHR)
EARLY_FRAGMENT_TESTS
FRAGMENT_SHADER
LATE_FRAGMENT_TESTS
COLOR_ATTACHMENT_OUTPUT

Compute Pipelines:

CONDITIONAL_RENDERING(EXT)
DRAW_INDIRECT
COMPUTE_SHADER

Ray Tracing Pipelines:

DRAW_INDIRECT
RAY_TRACING_SHADER(KHR)

Ray-Tracing Acceleration Structure Build Commands:

ACCELERATION_STRUCTURE_BUILD(KHR)

Copy Commands:

COPY

Blit Commands:

BLIT

Resolve Commands:

RESOLVE

Clear Commands:

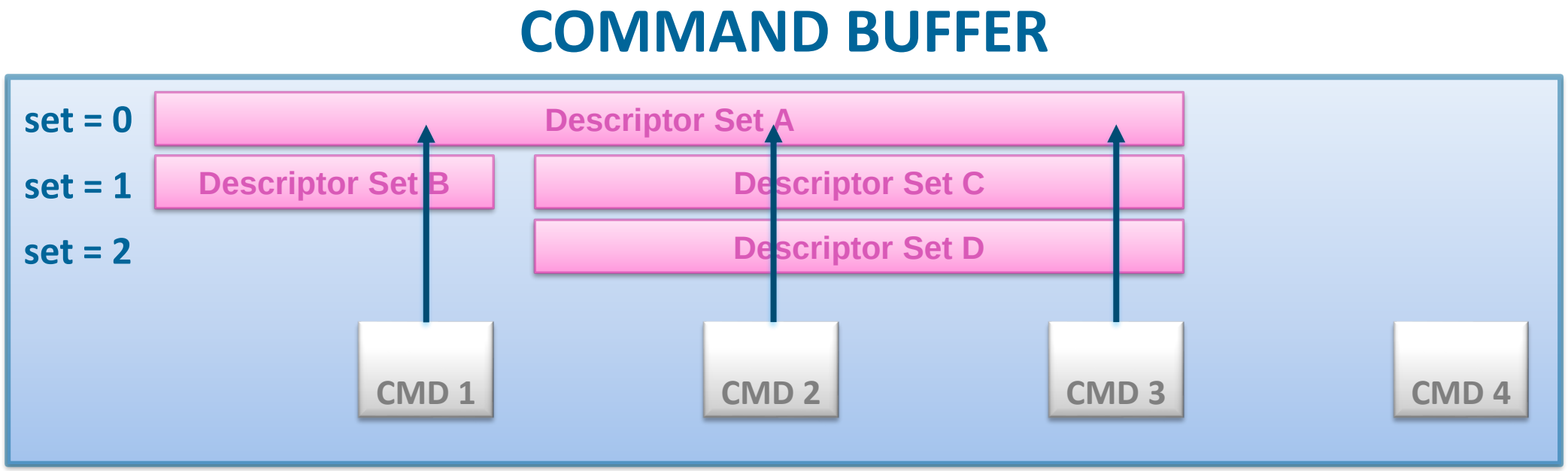
CLEAR

None/Aggregate Stages:

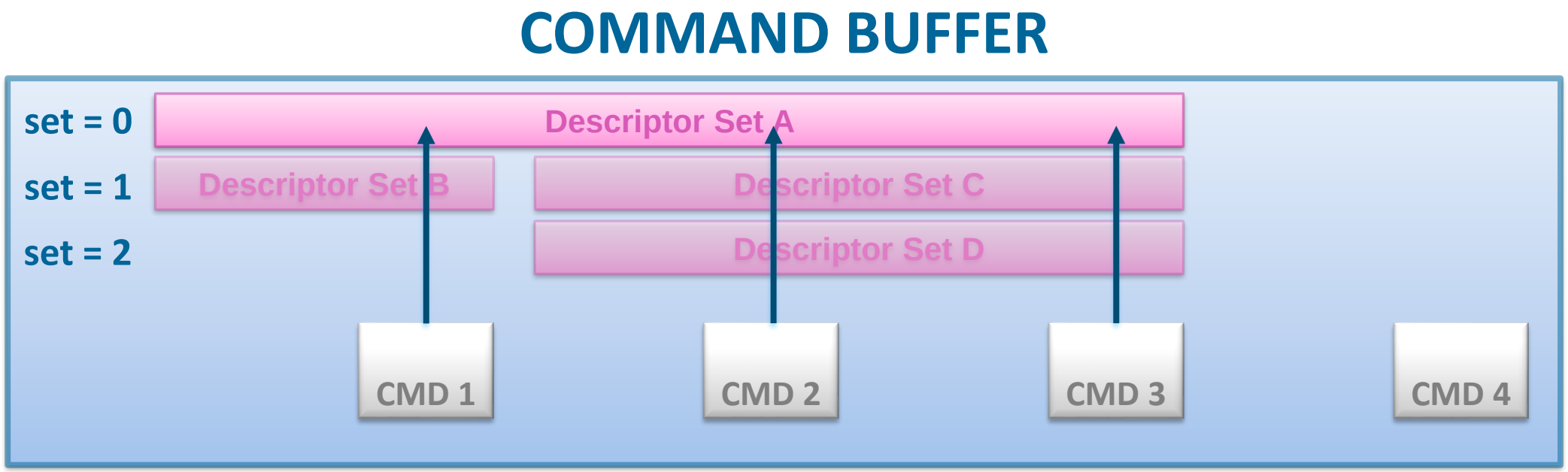
NONE
ALL_GRAPHICS
ALL_TRANSFER
ALL_COMMANDS

Full List of Pipeline Stages:
VkPipelineStageFlagBits2
The Khronos Group. Vulkan 1.3.212 Specificaton

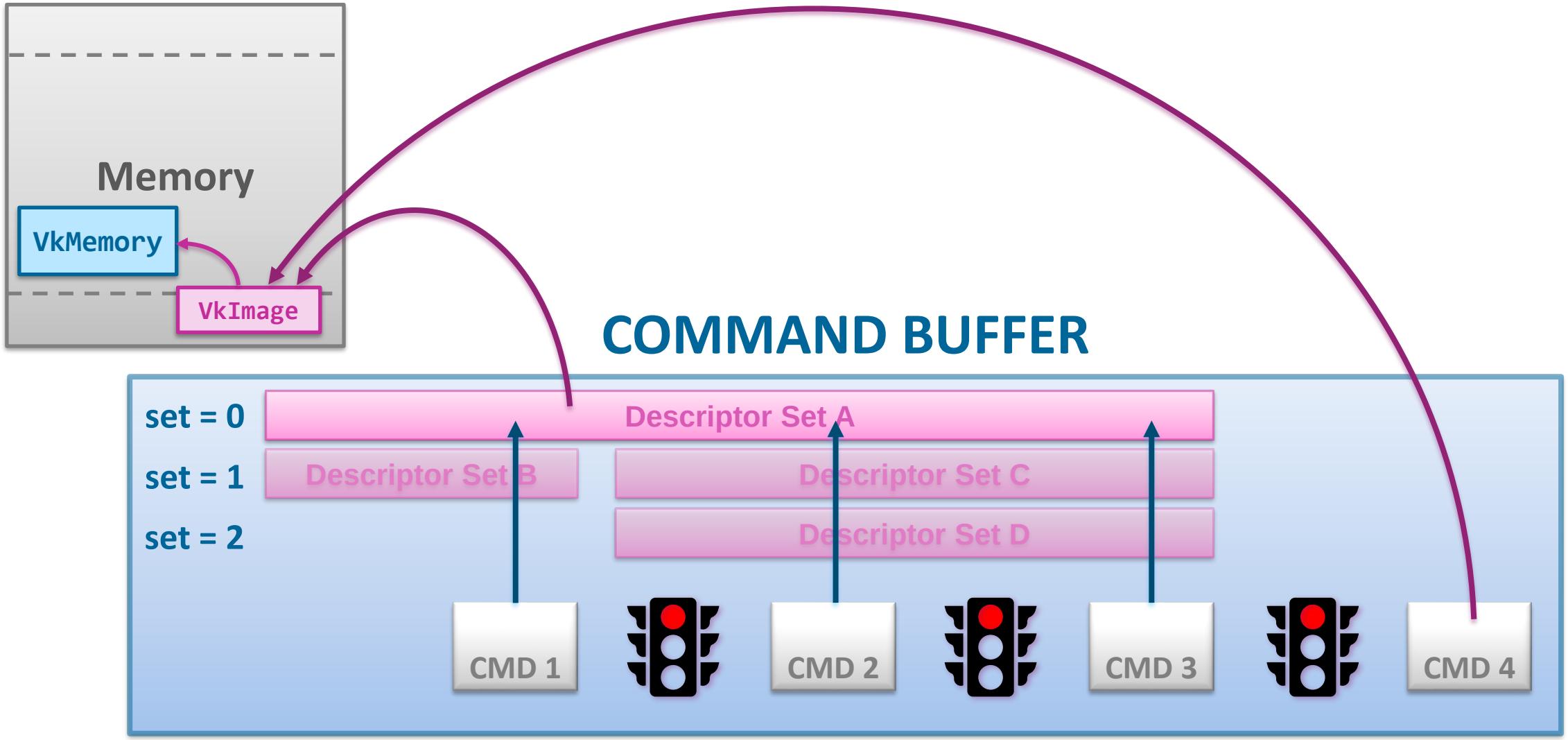
Command Recording and Queue Submission



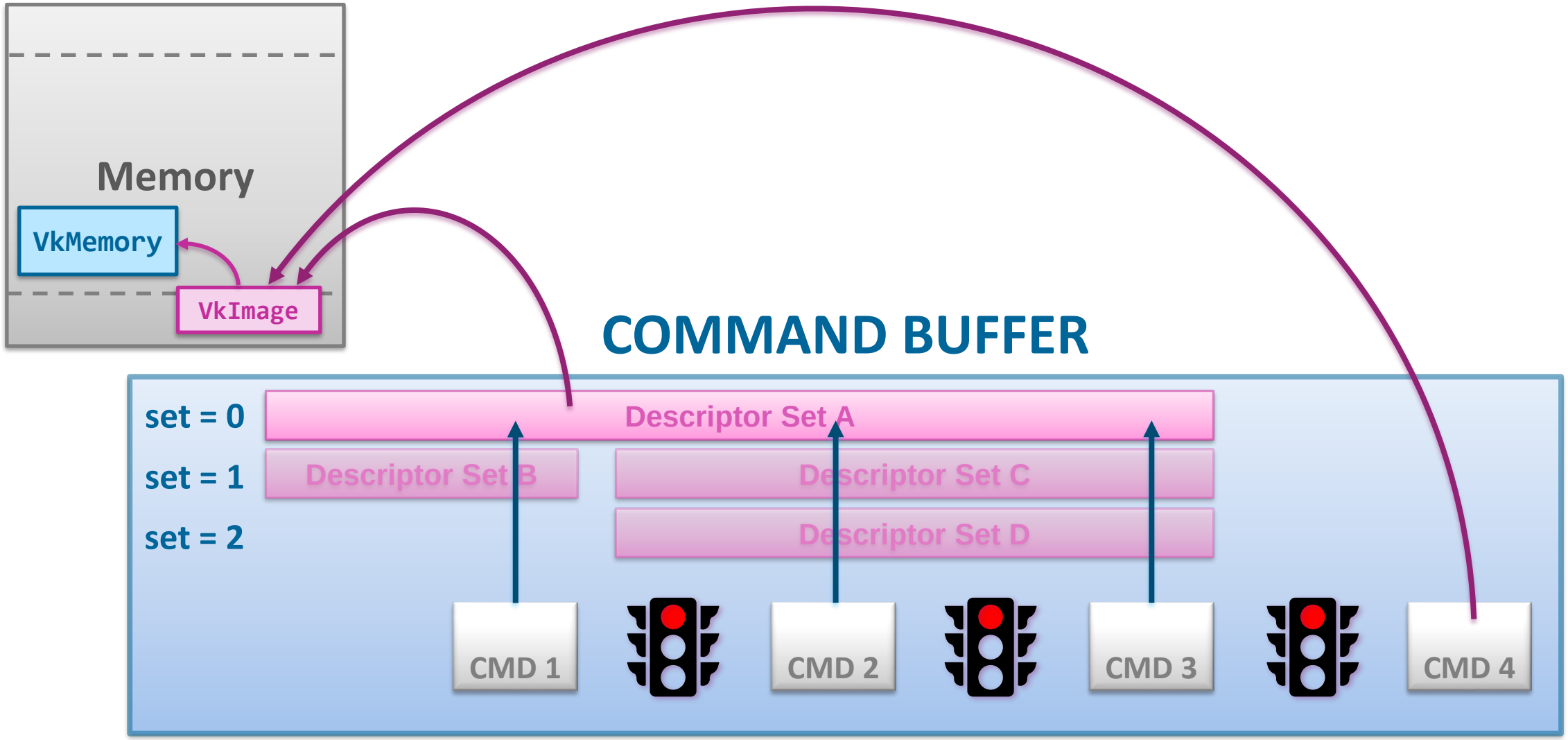
Command Recording and Queue Submission



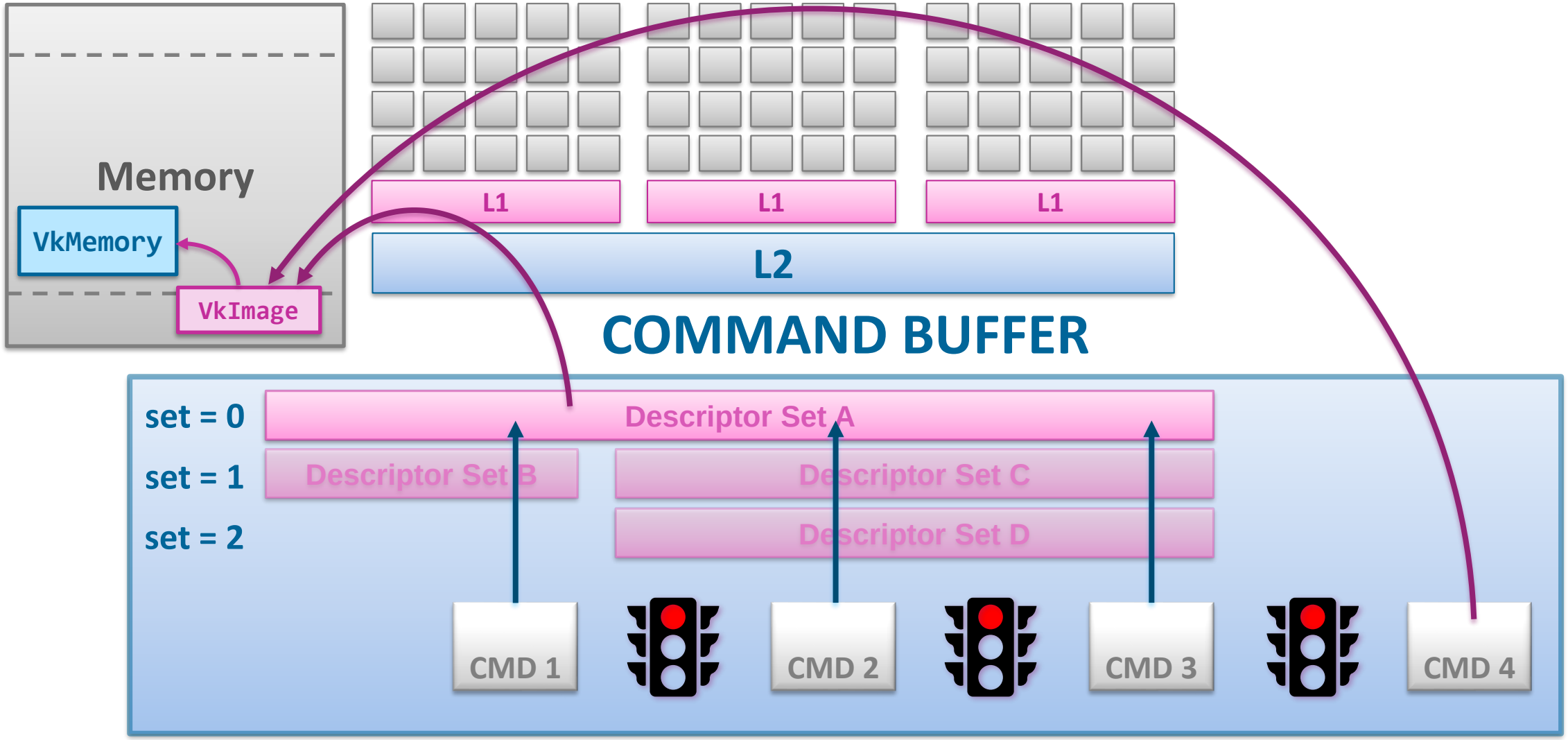
Command Recording and Queue Submission



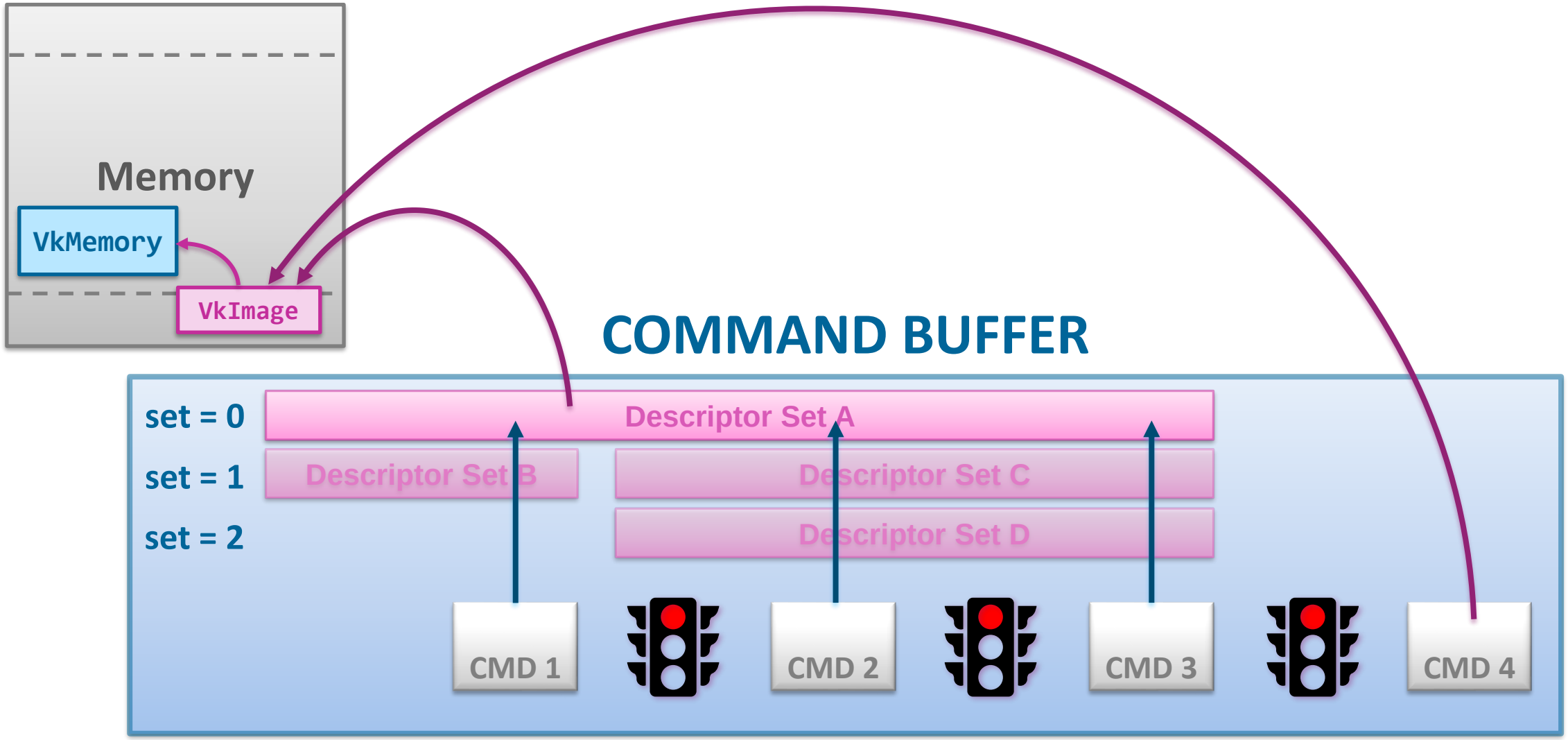
Command Recording and Queue Submission



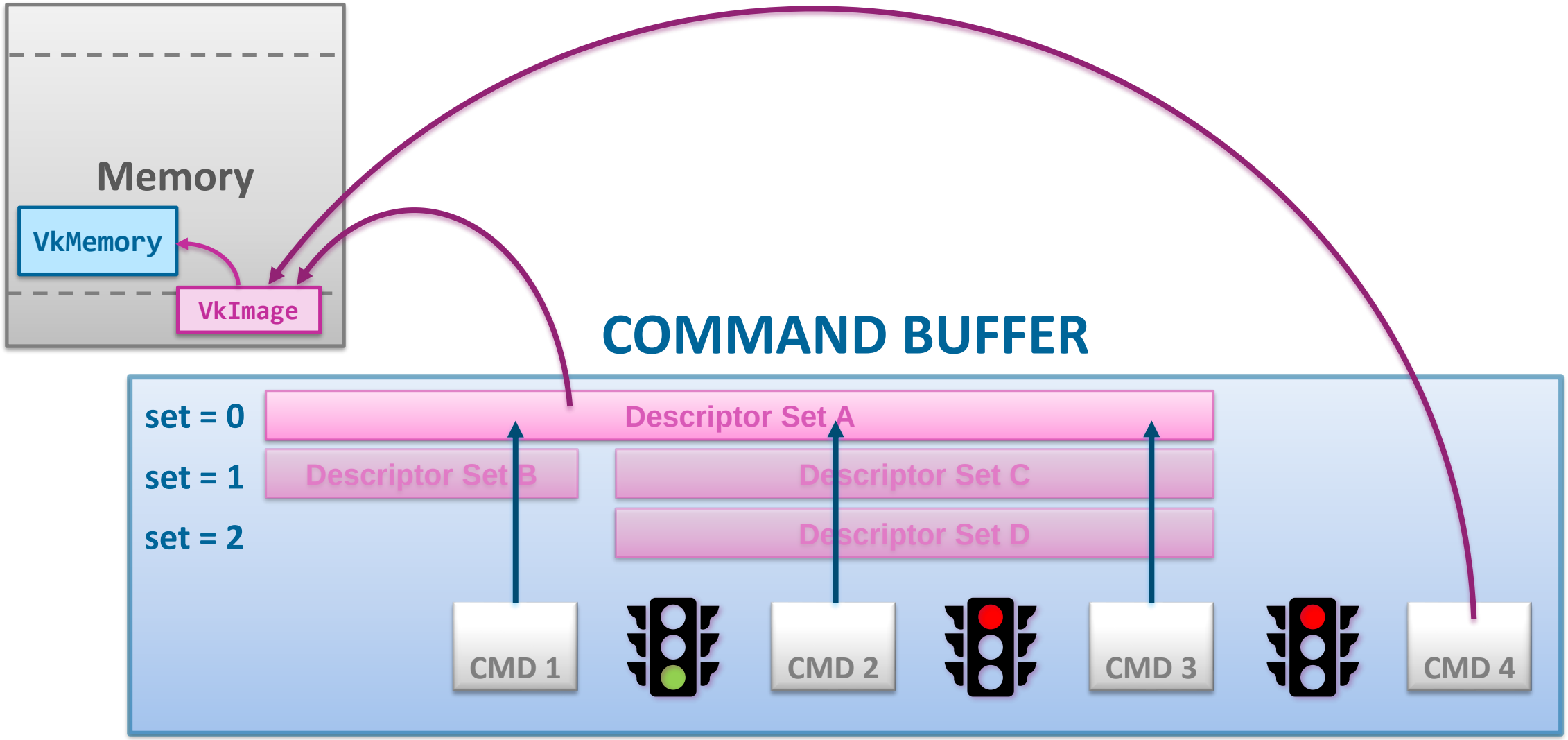
Command Recording and Queue Submission



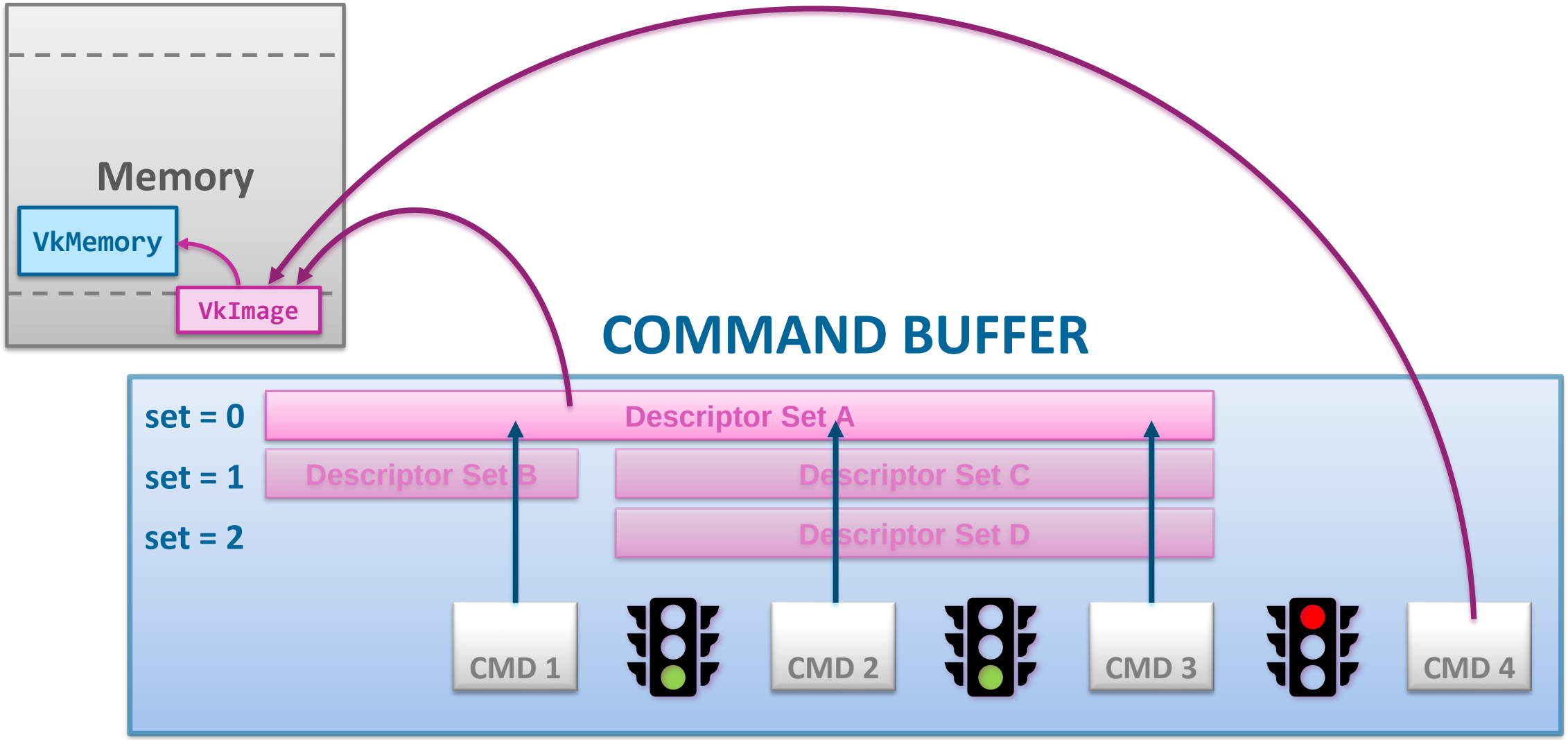
Command Recording and Queue Submission



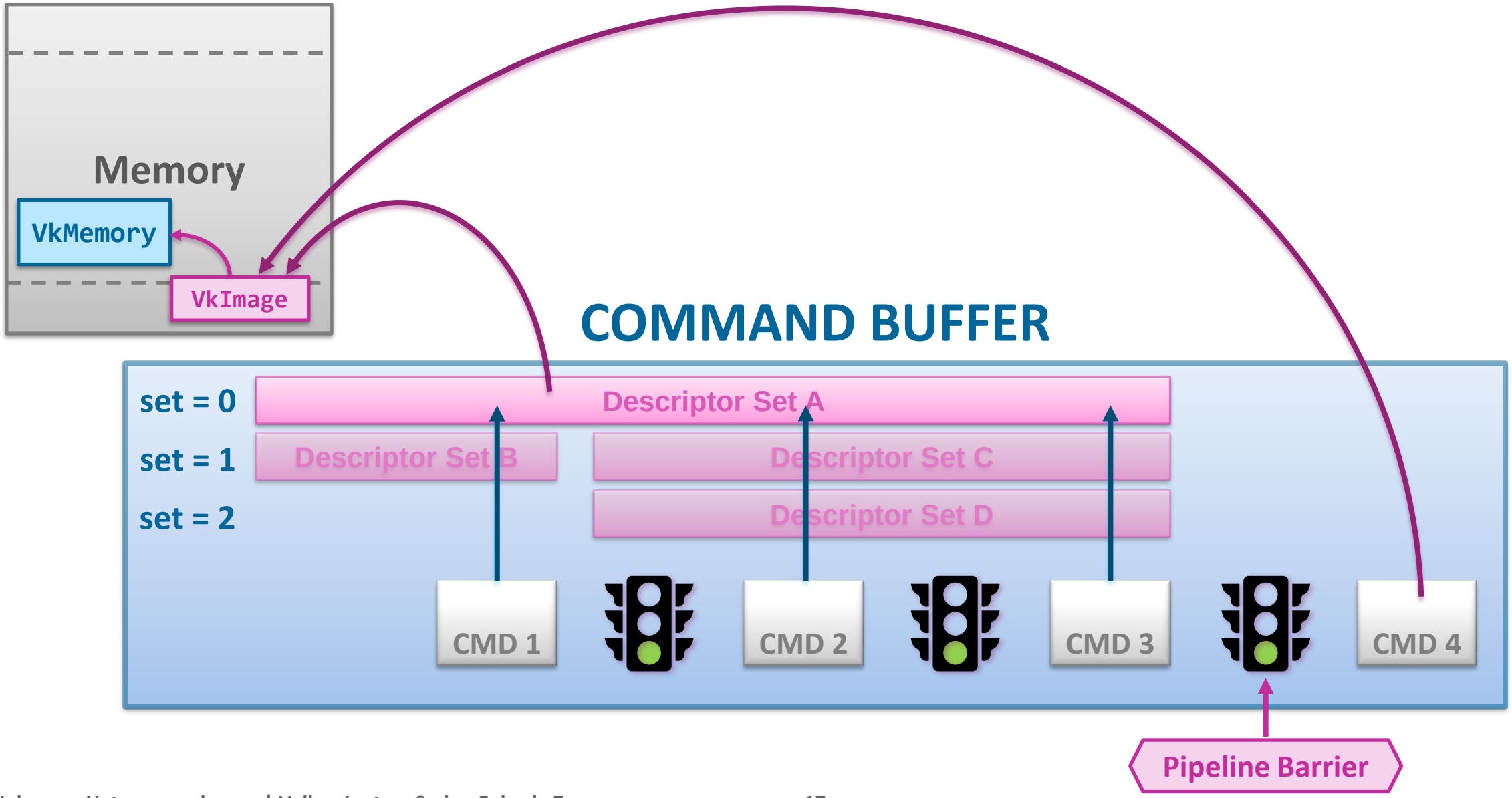
Command Recording and Queue Submission



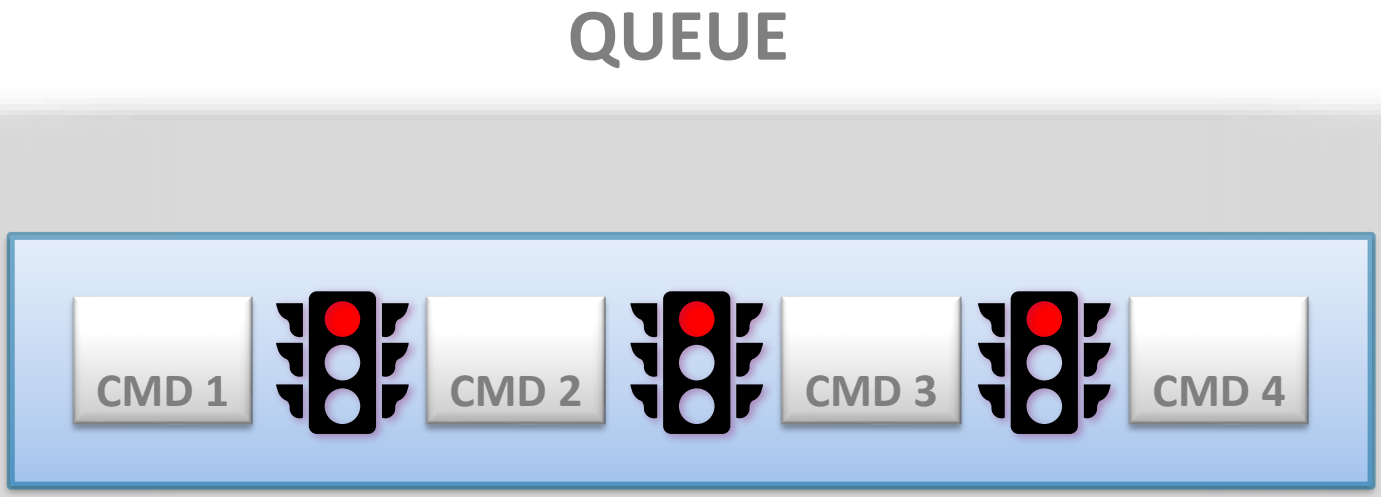
Command Recording and Queue Submission



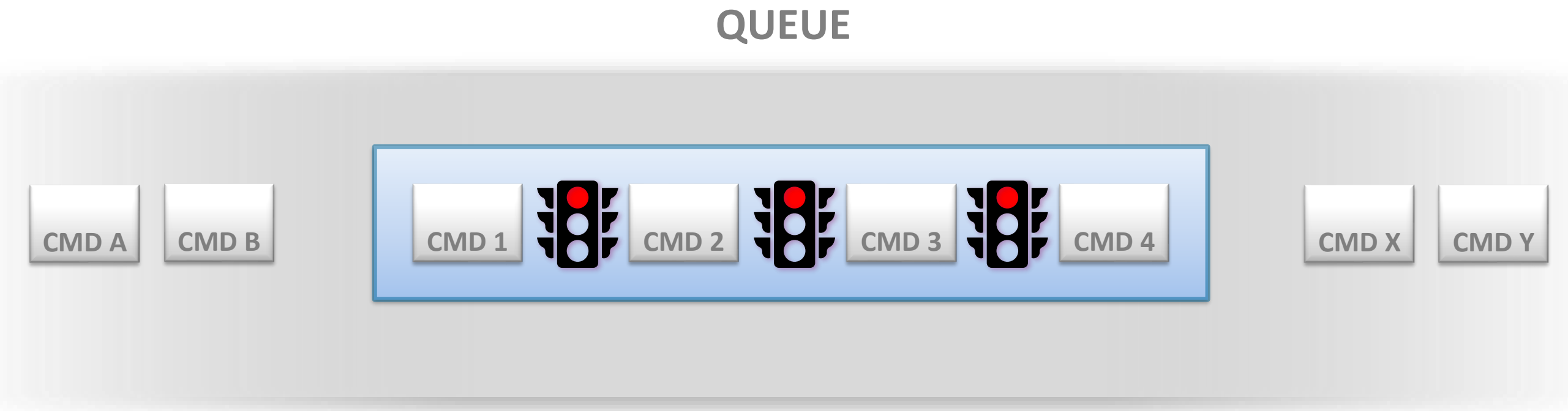
Command Recording and Queue Submission



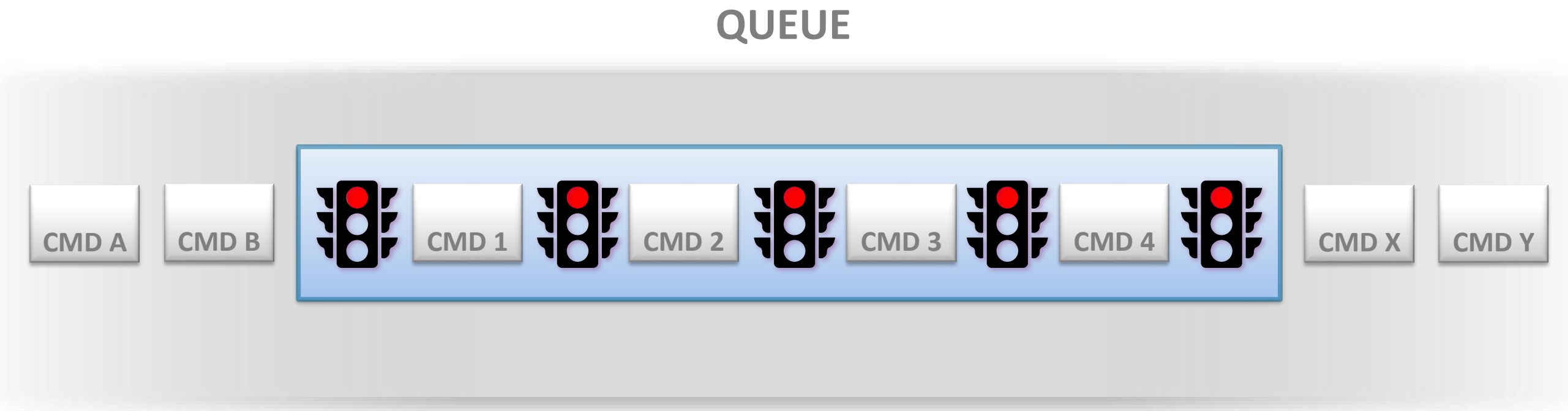
Command Recording and Queue Submission



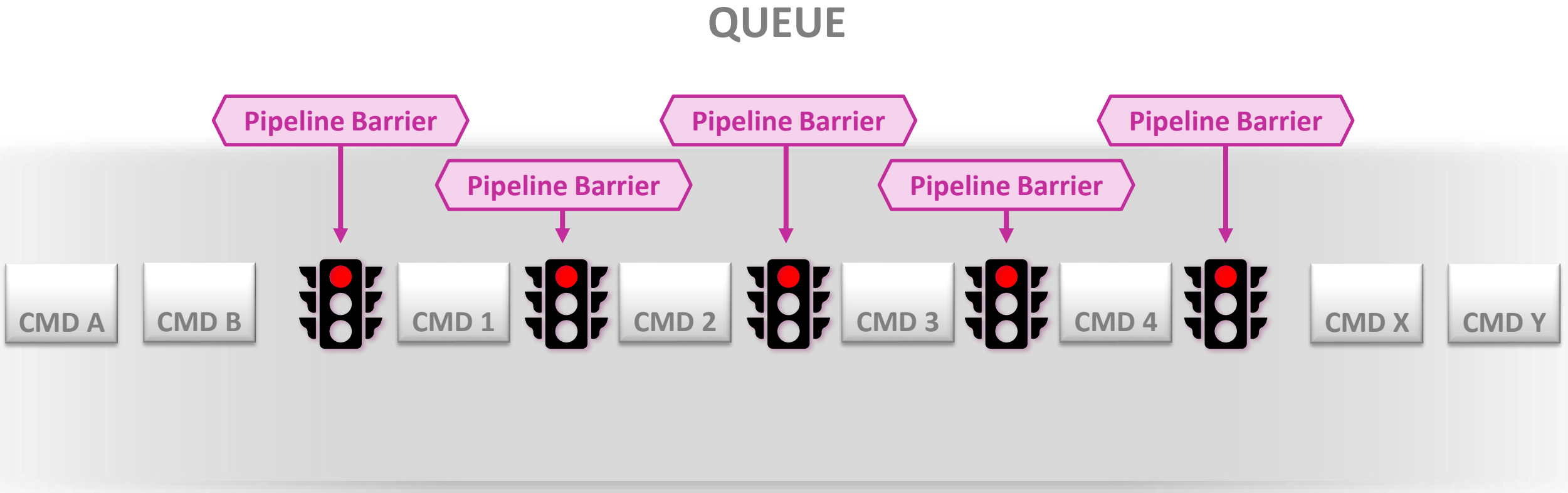
Command Recording and Queue Submission



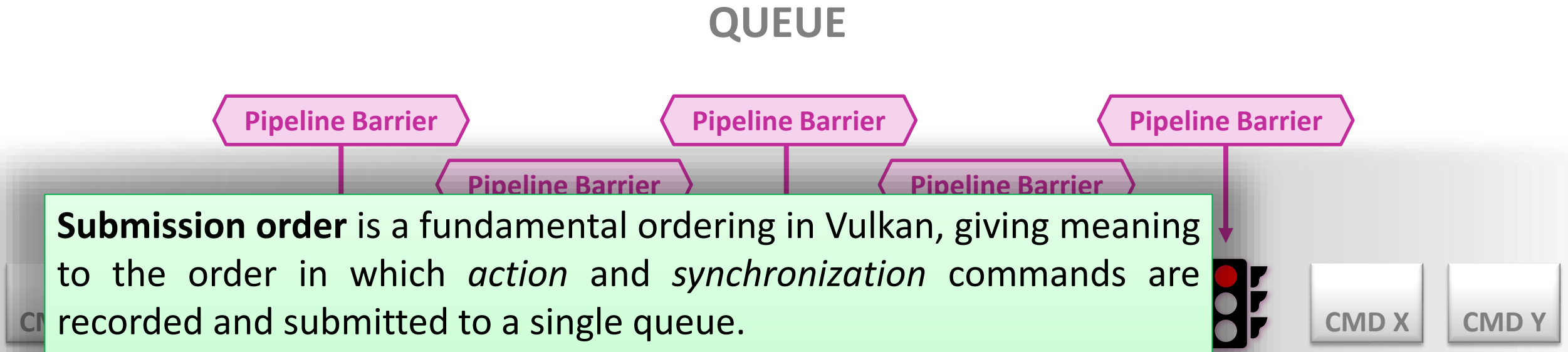
Command Recording and Queue Submission



Command Recording and Queue Submission



Command Recording and Queue Submission



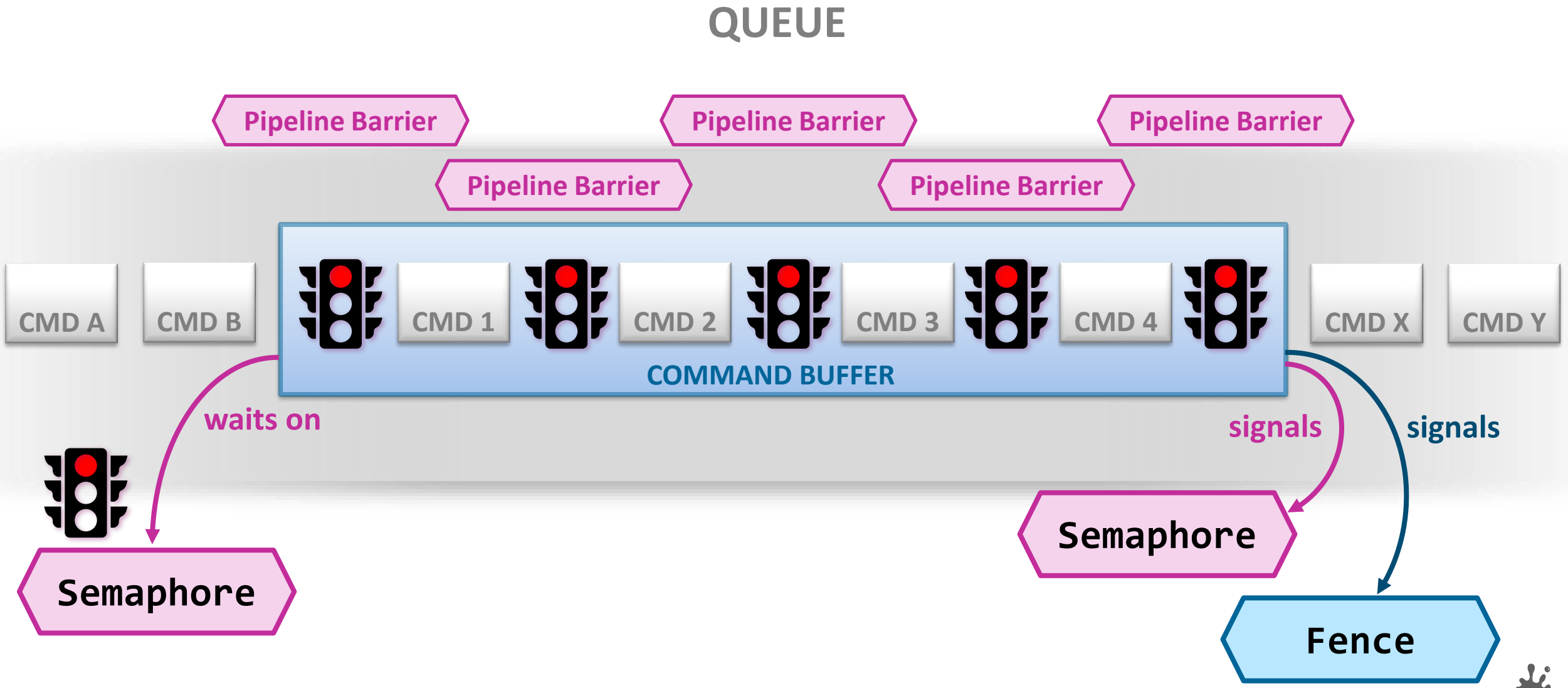
Submission order is a fundamental ordering in Vulkan, giving meaning to the order in which *action* and *synchronization* commands are recorded and submitted to a single queue.

Explicit and implicit ordering guarantees between commands in Vulkan all work on the premise that this ordering is meaningful.

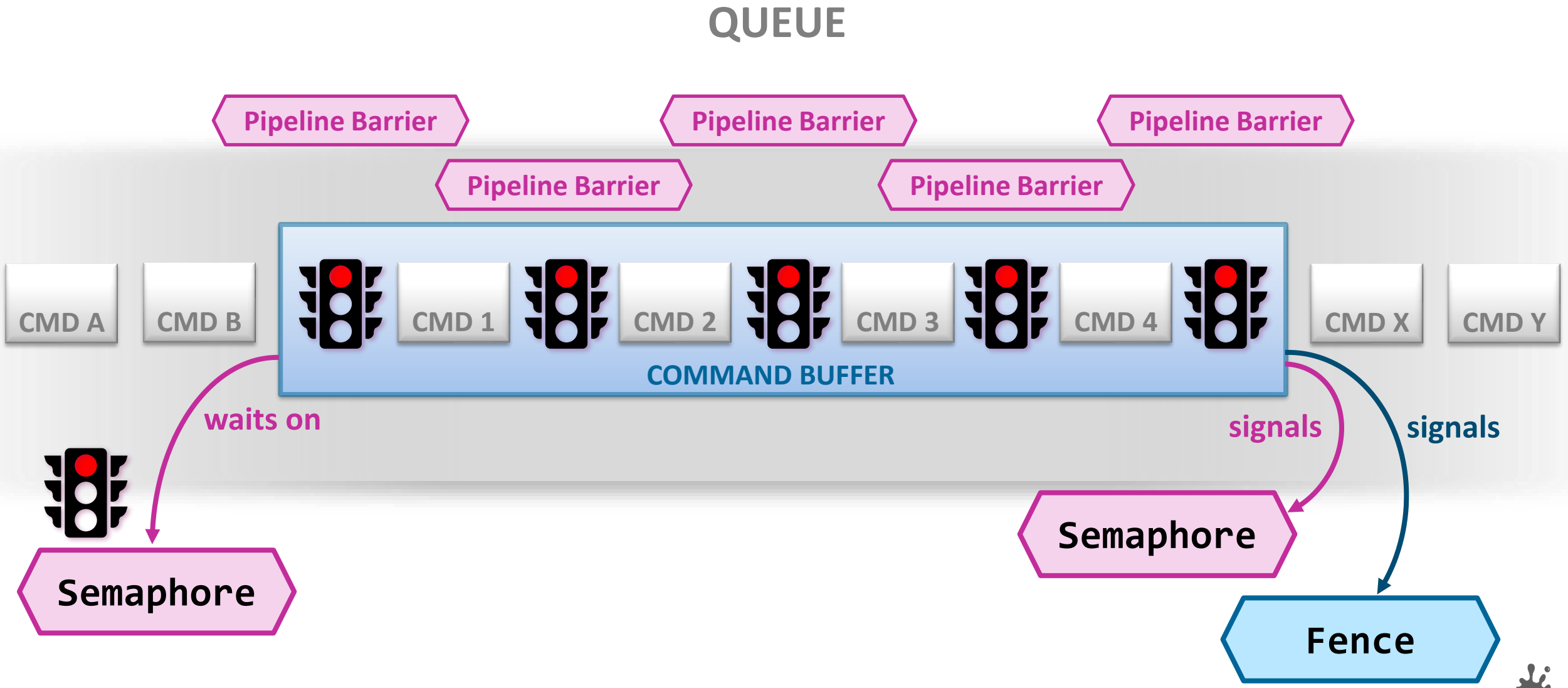
This order does not itself define any execution or memory dependencies; synchronization commands and other orderings within the API use this ordering to define their scopes.

The Khronos Group. Vulkan 1.3.209 Specificaton

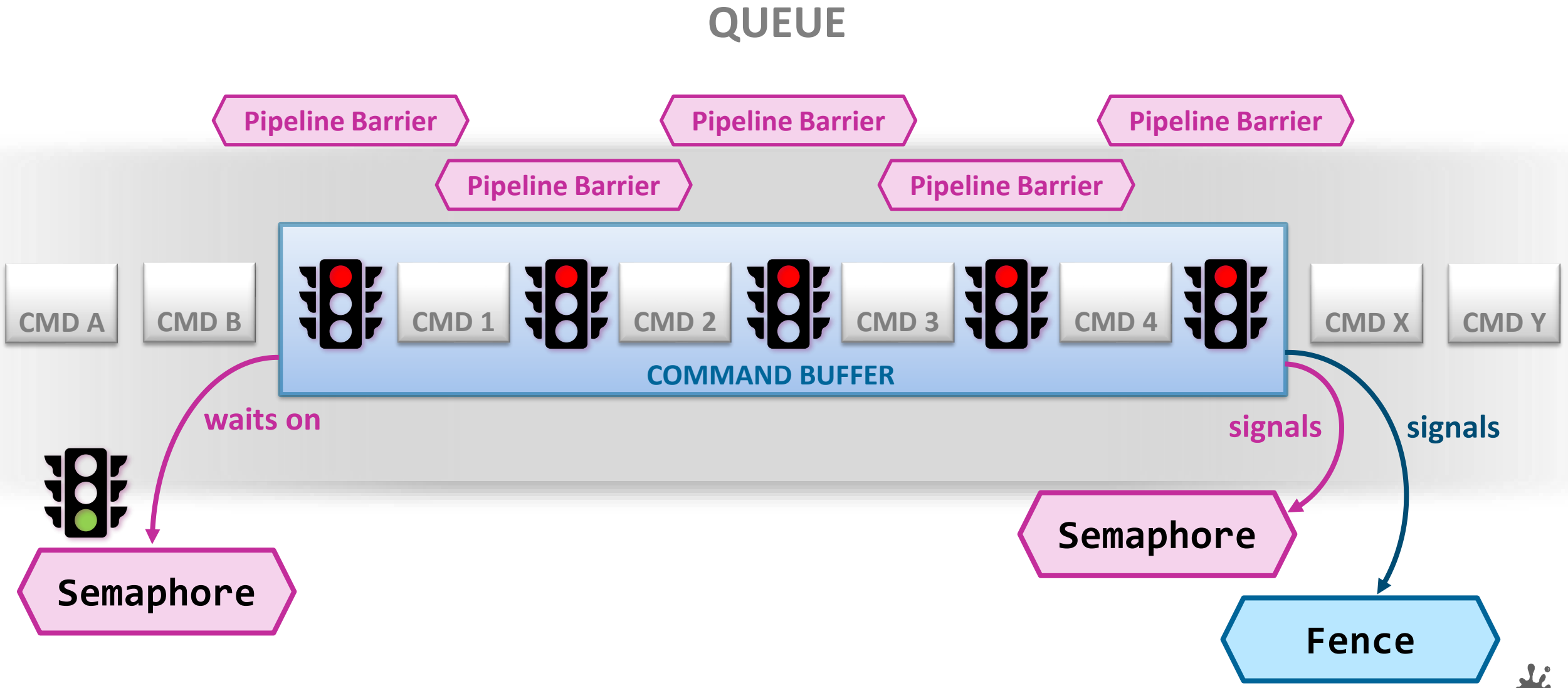
Command Recording and Queue Submission



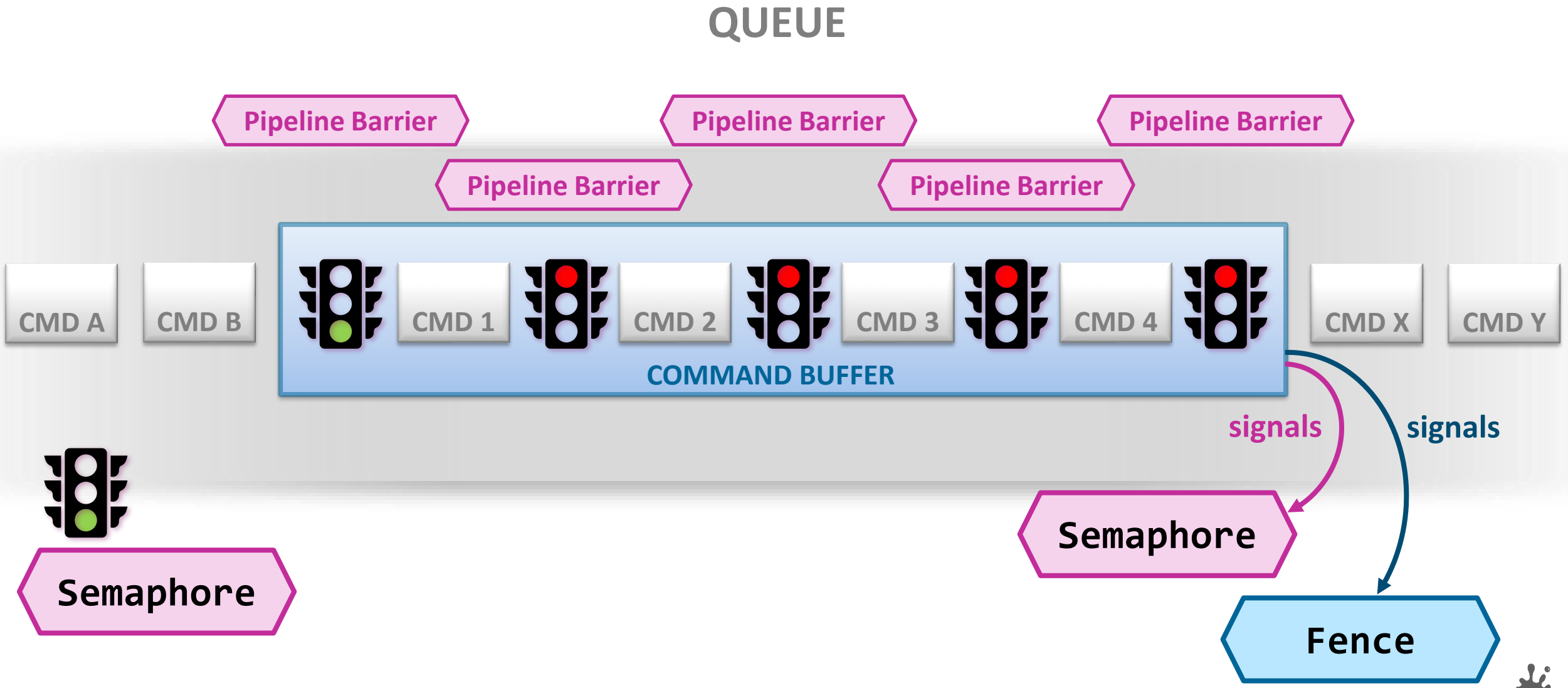
Command Recording and Queue Submission



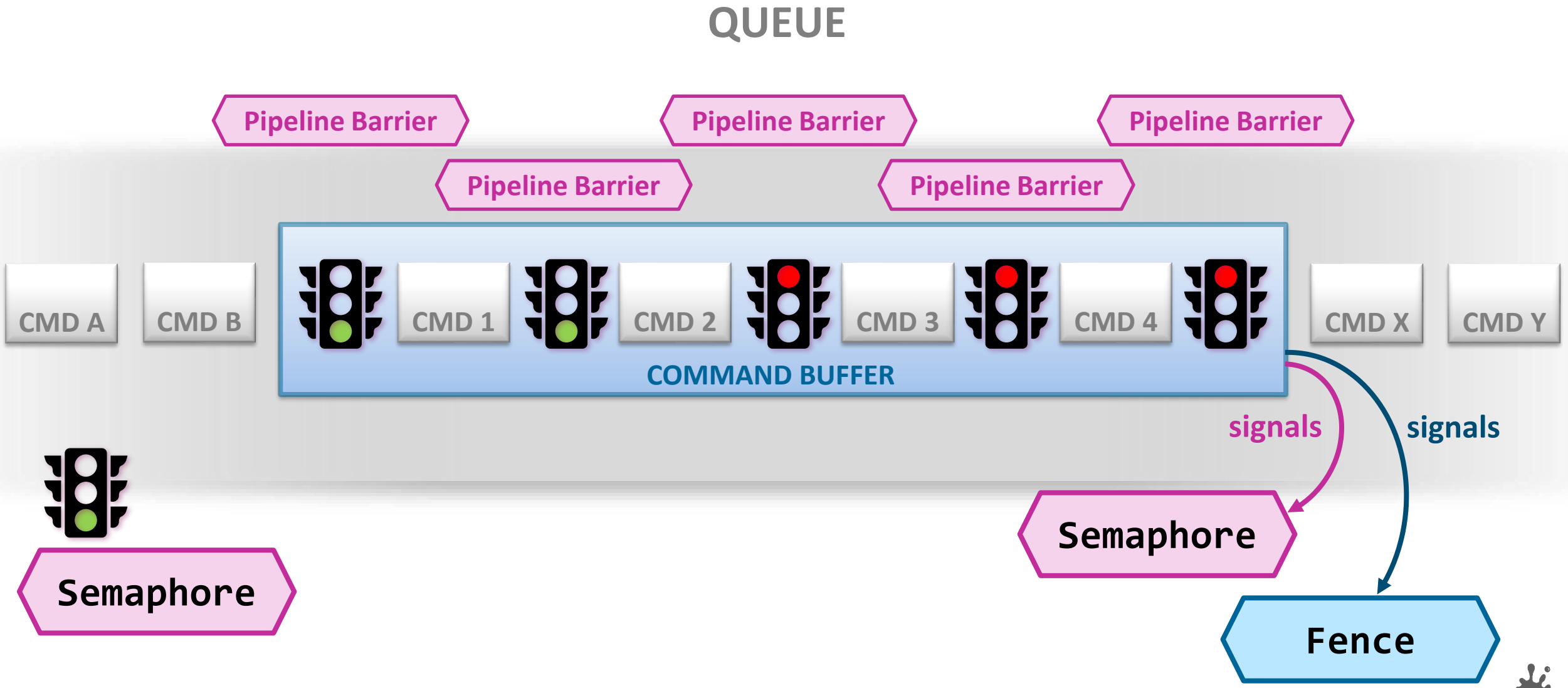
Command Recording and Queue Submission



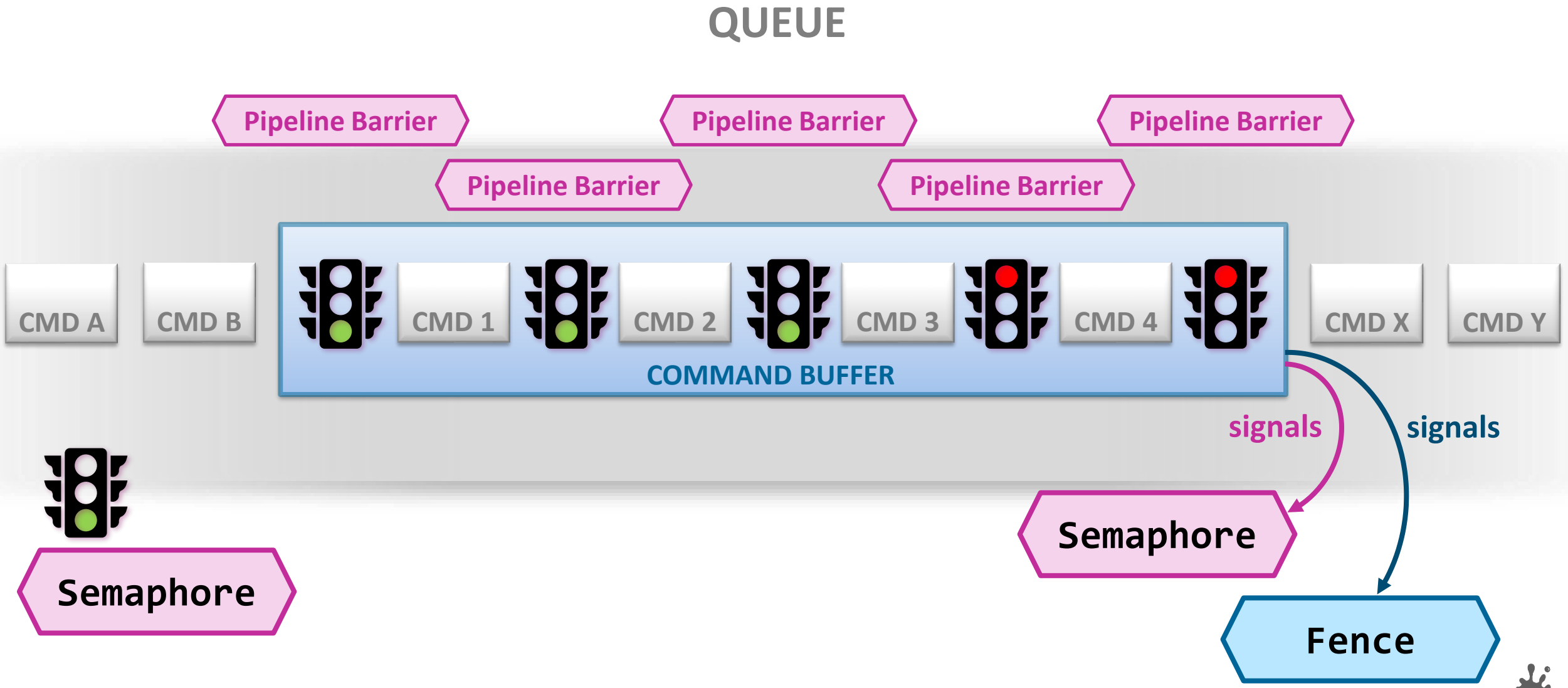
Command Recording and Queue Submission



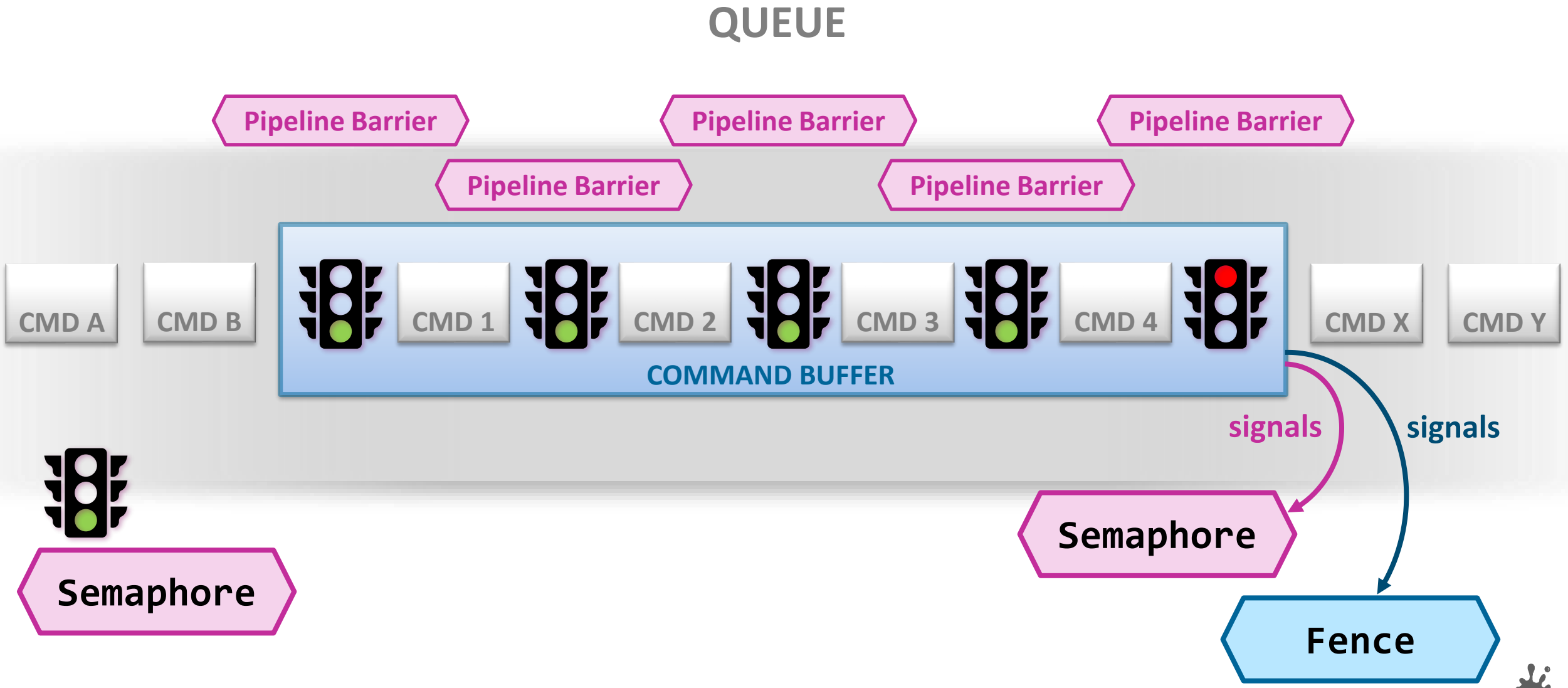
Command Recording and Queue Submission



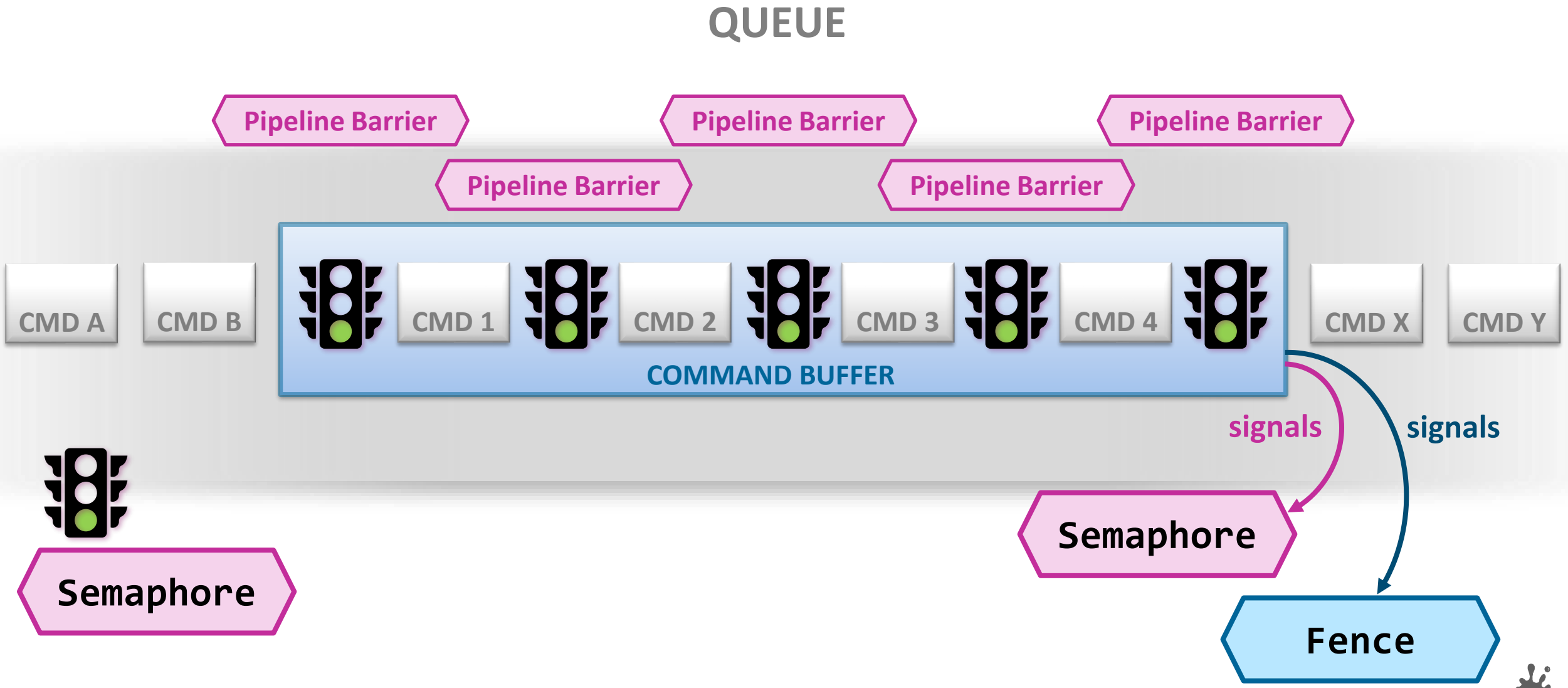
Command Recording and Queue Submission



Command Recording and Queue Submission



Command Recording and Queue Submission



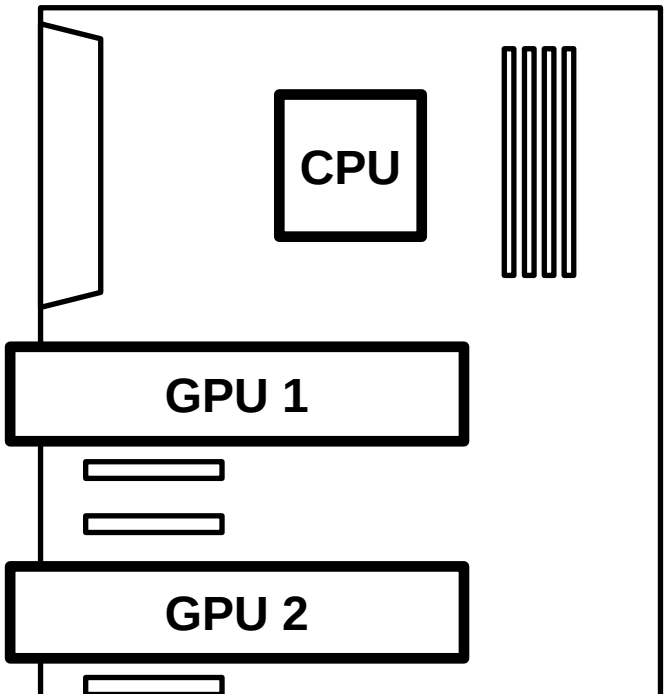
Command Recording and Queue Submission

Semaphore

Fence

QUEUE

COMMAND BUFFER



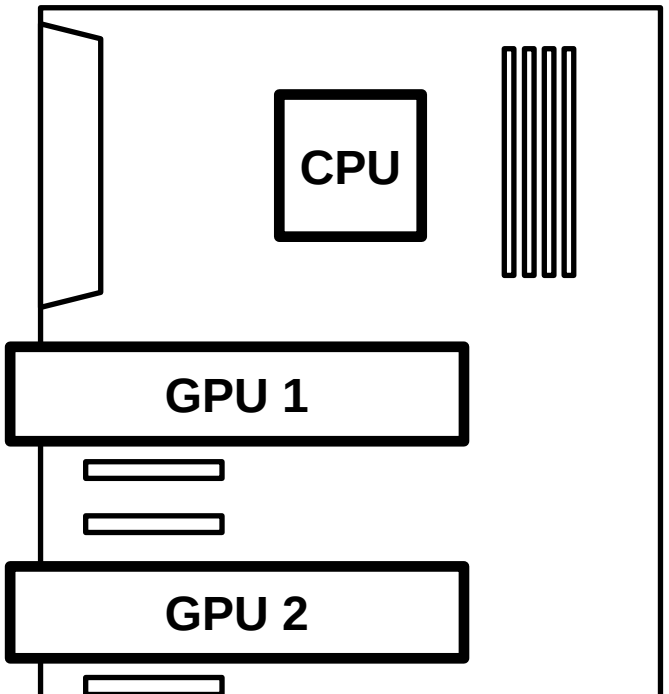
Command Recording and Queue Submission

Semaphore

Fence

QUEUE

COMMAND BUFFER 1 | COMMAND BUFFER 2



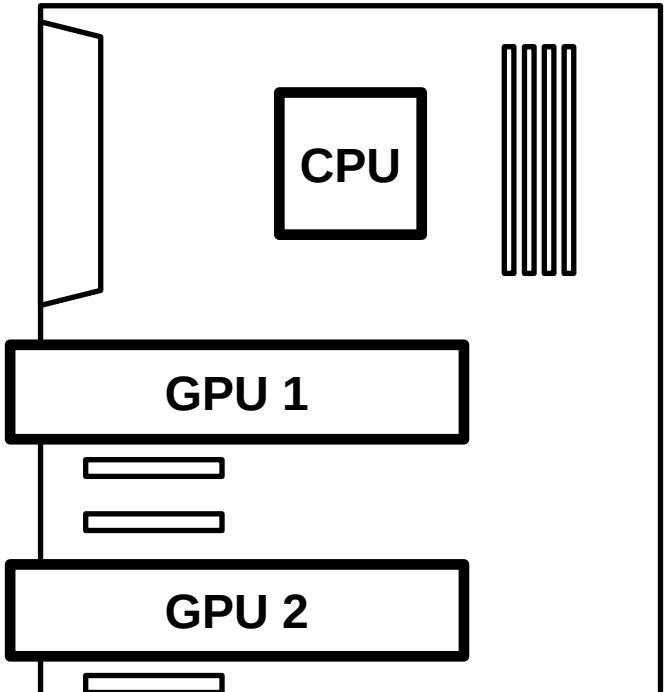
Command Recording and Queue Submission

Semaphore

Fence

QUEUE

BATCH



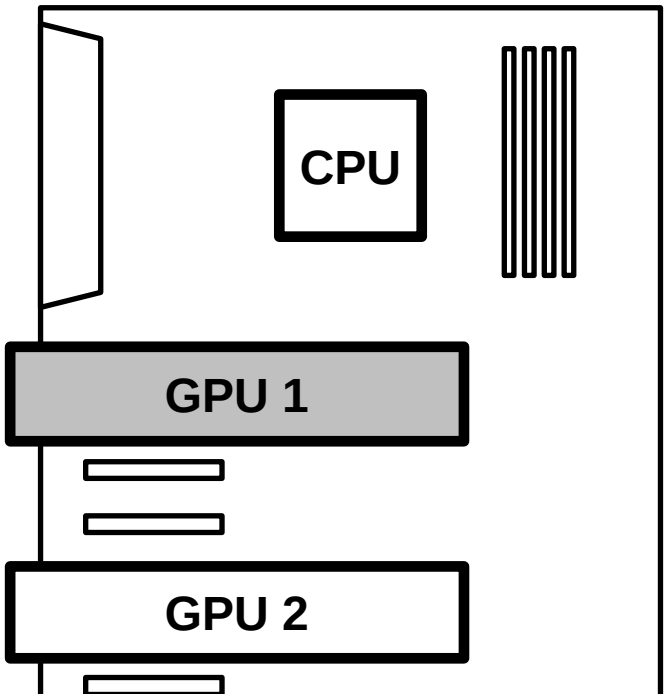
Command Recording and Queue Submission

Semaphore

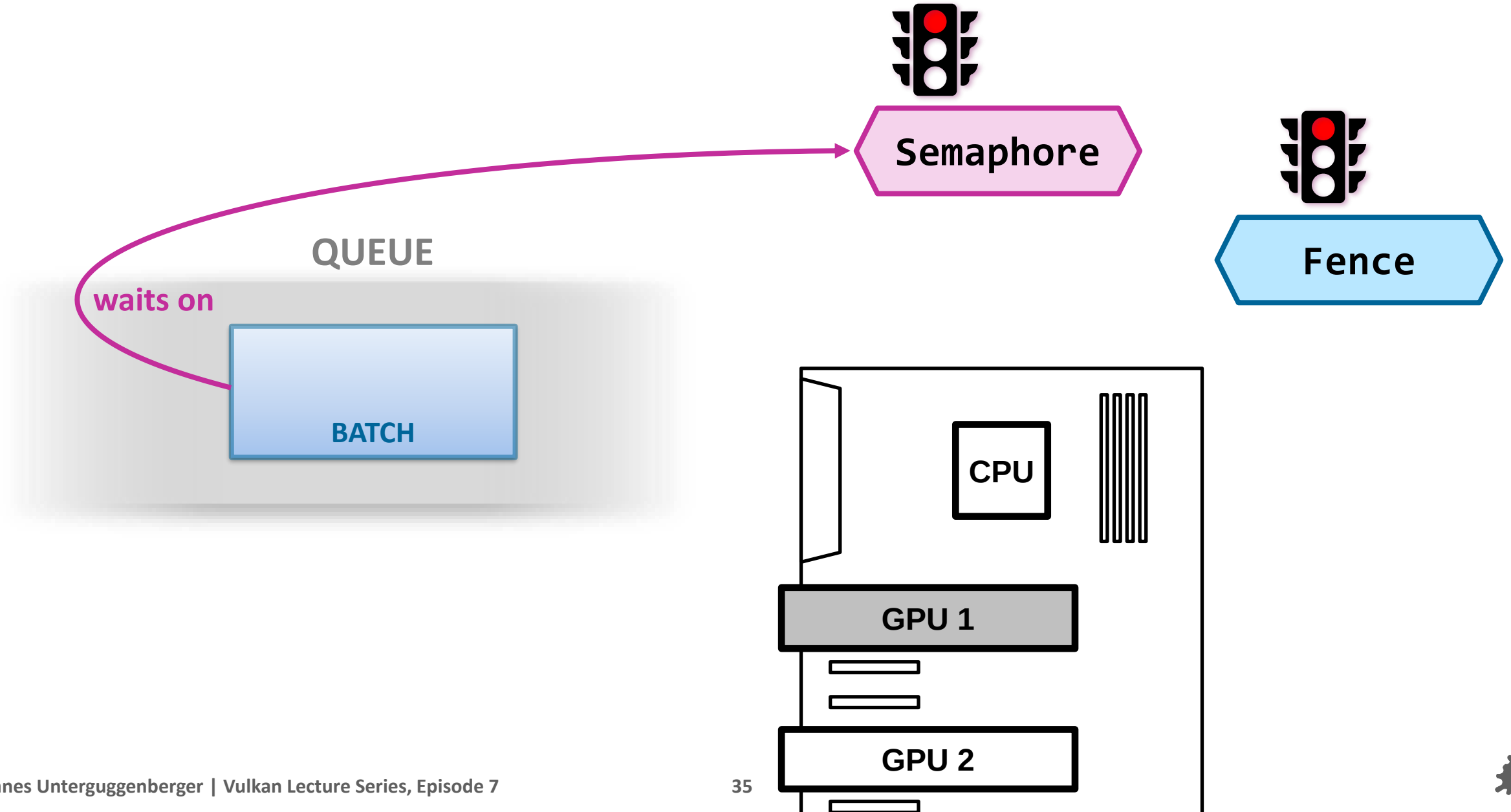
Fence

QUEUE

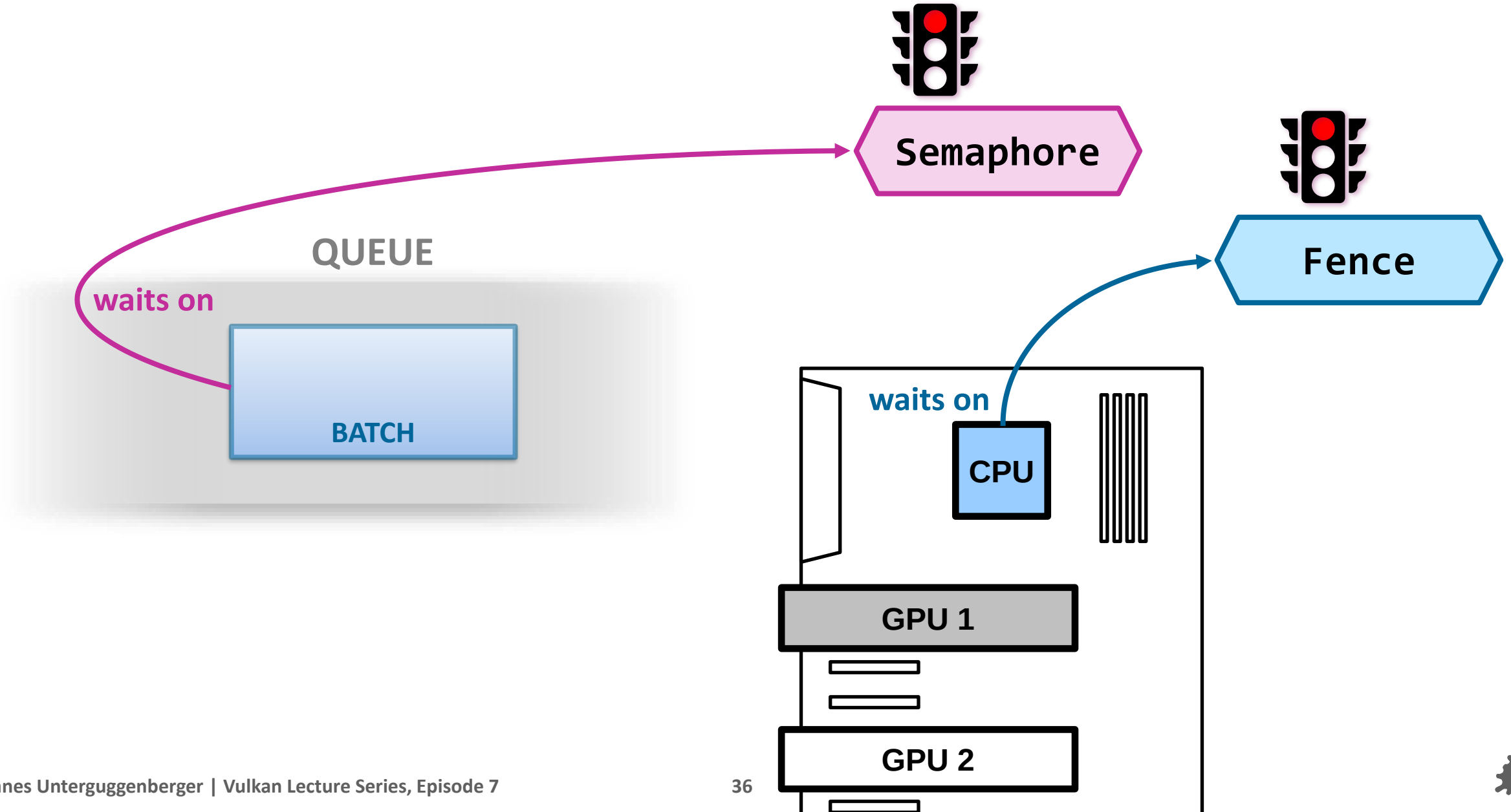
BATCH



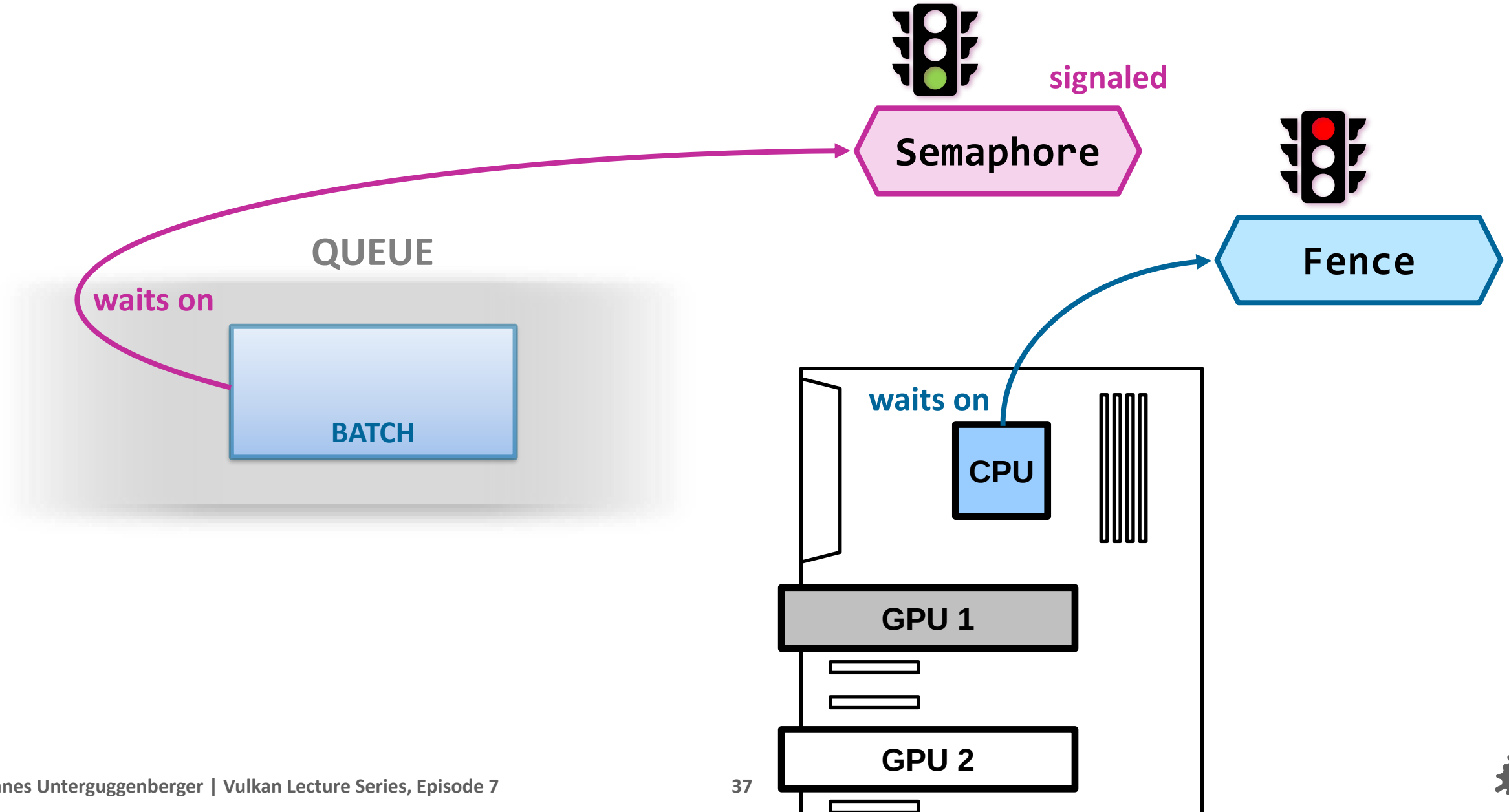
Command Recording and Queue Submission



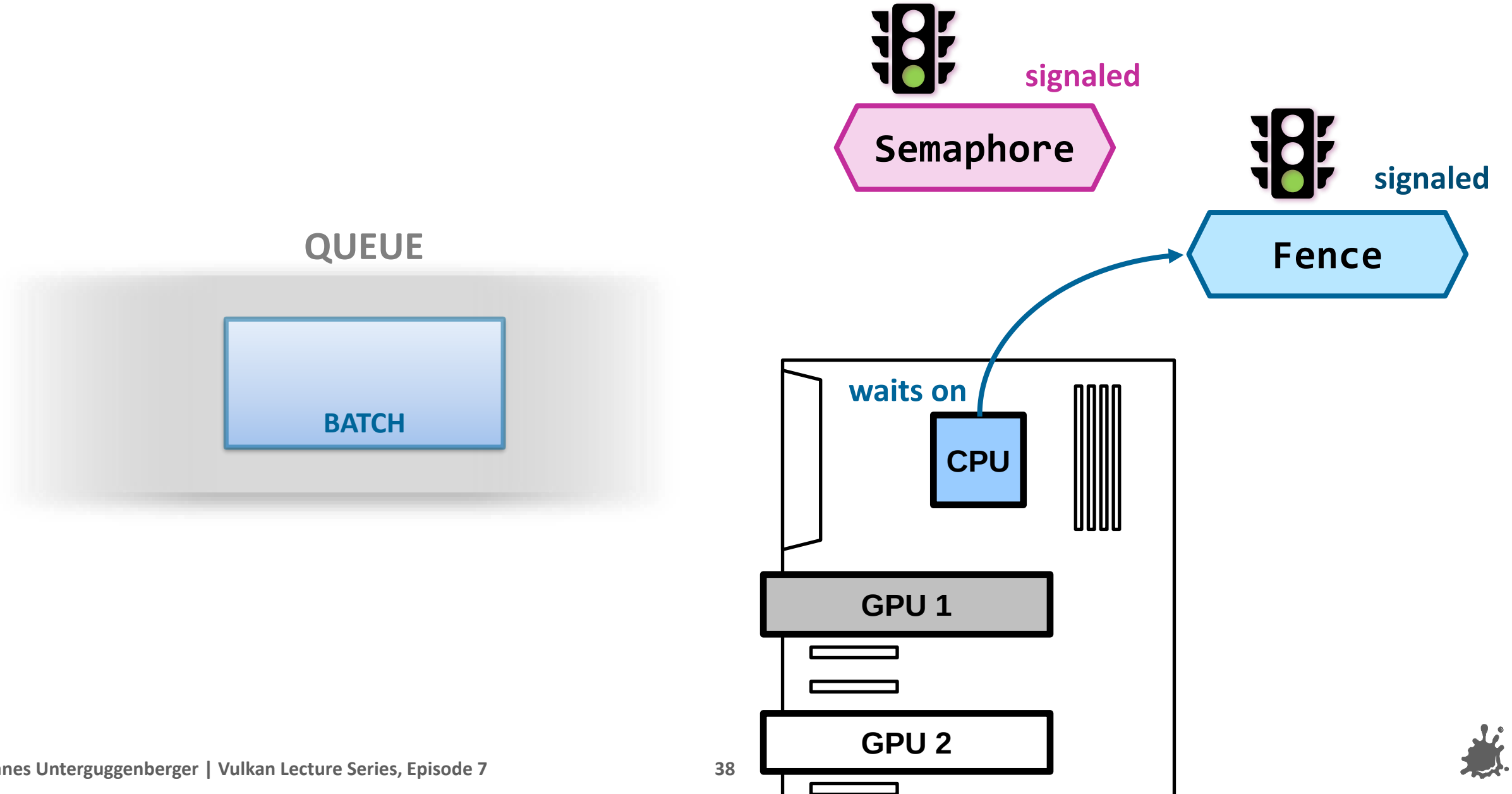
Command Recording and Queue Submission



Command Recording and Queue Submission



Command Recording and Queue Submission



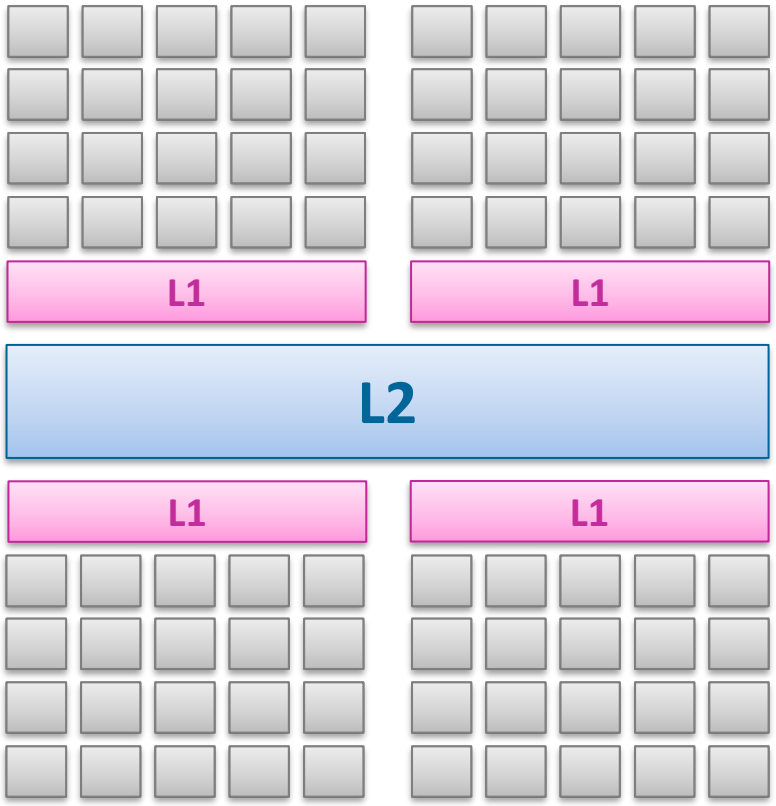
Overview of Synchronization Methods

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

Overview of Synchronization Methods

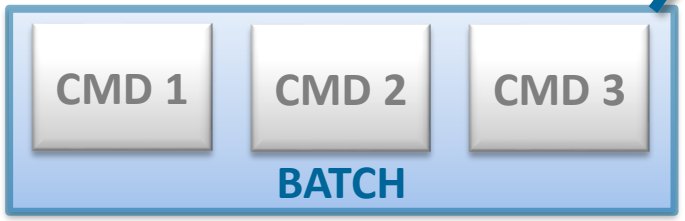
- Wait Idle Operations
- Fences
- Semaphores
  - Binary Semaphores
  - Timeline Semaphores
- Pipeline Barriers
  - Execution Barriers
  - Memory Barriers
- Render Pass Subpass Dependencies
- Events

Wait Idle Operations

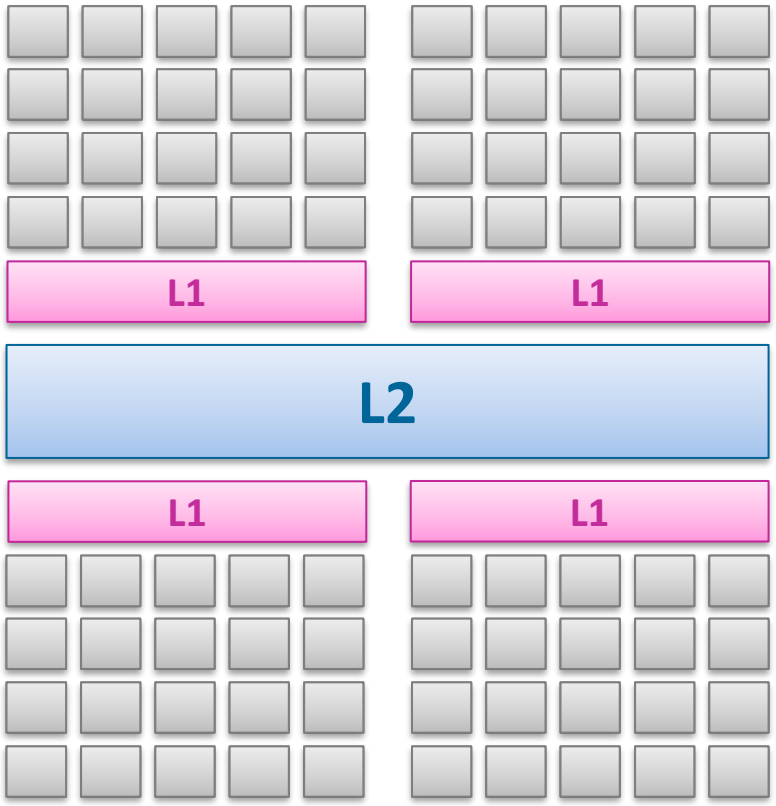


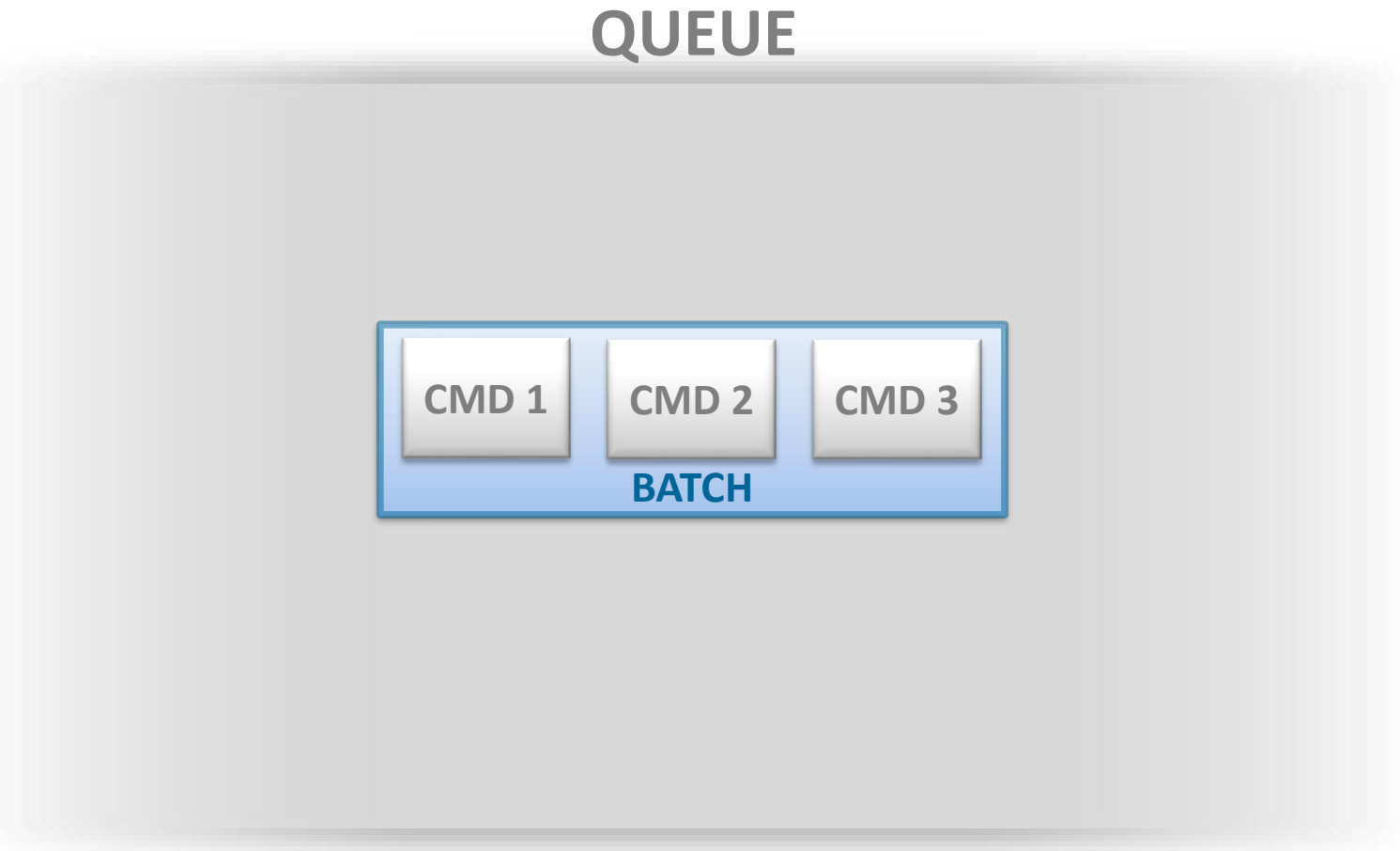
QUEUE

vkQueueSubmit:

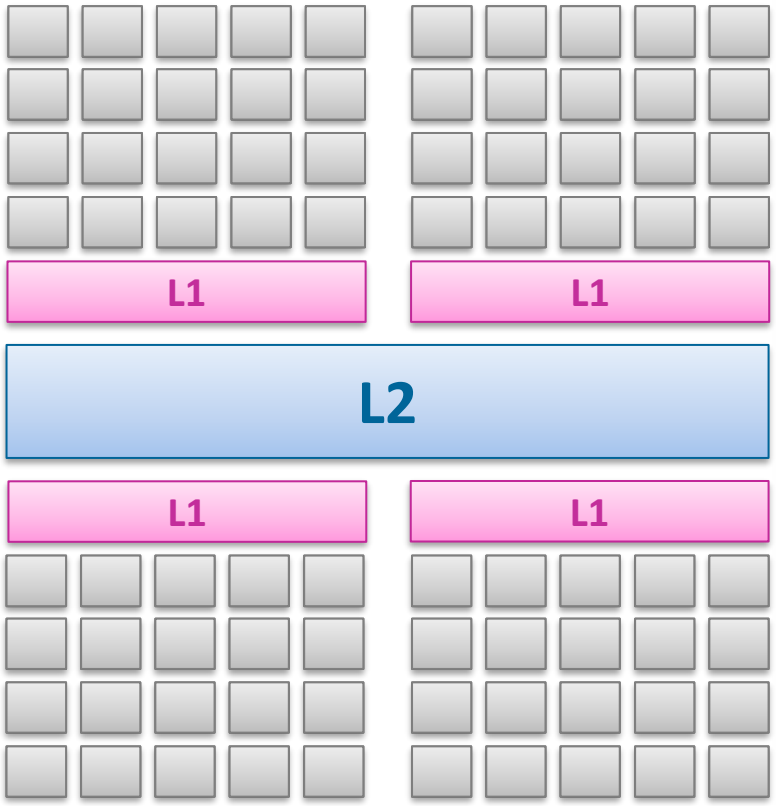


Wait Idle Operations





Wait Idle Operations



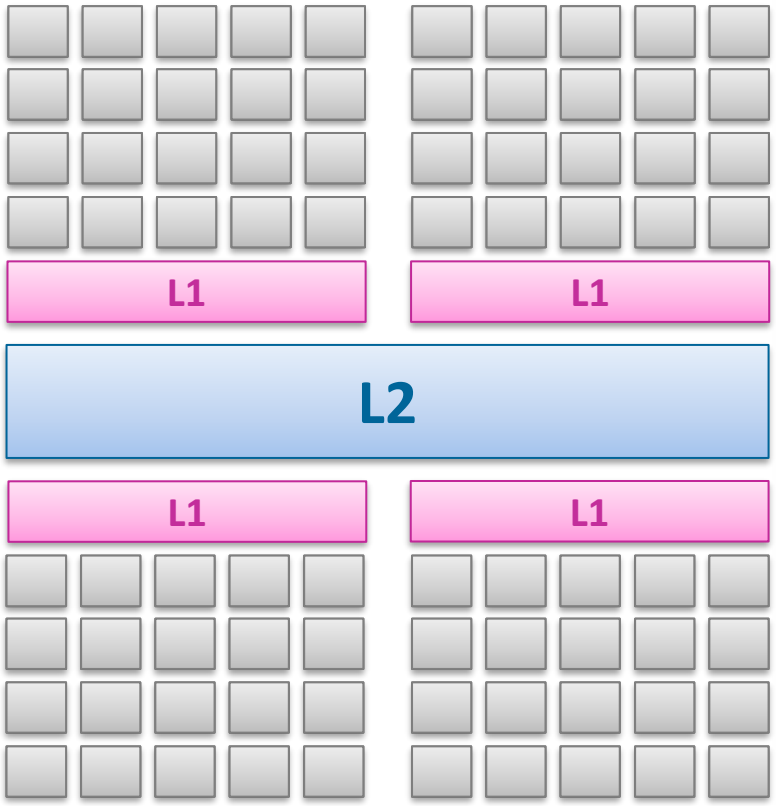
QUEUE

CMD 1

CMD 2

CMD 3

Wait Idle Operations



QUEUE

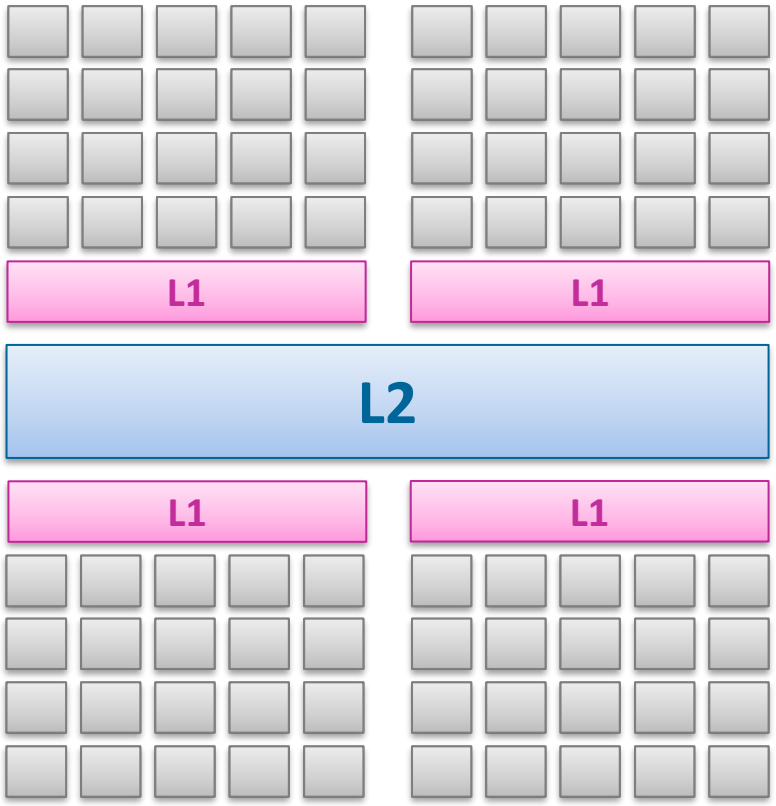
CMD 1 CMD 2 CMD 3

BATCH

Wait Idle

waits on

Wait Idle Operations



QUEUE

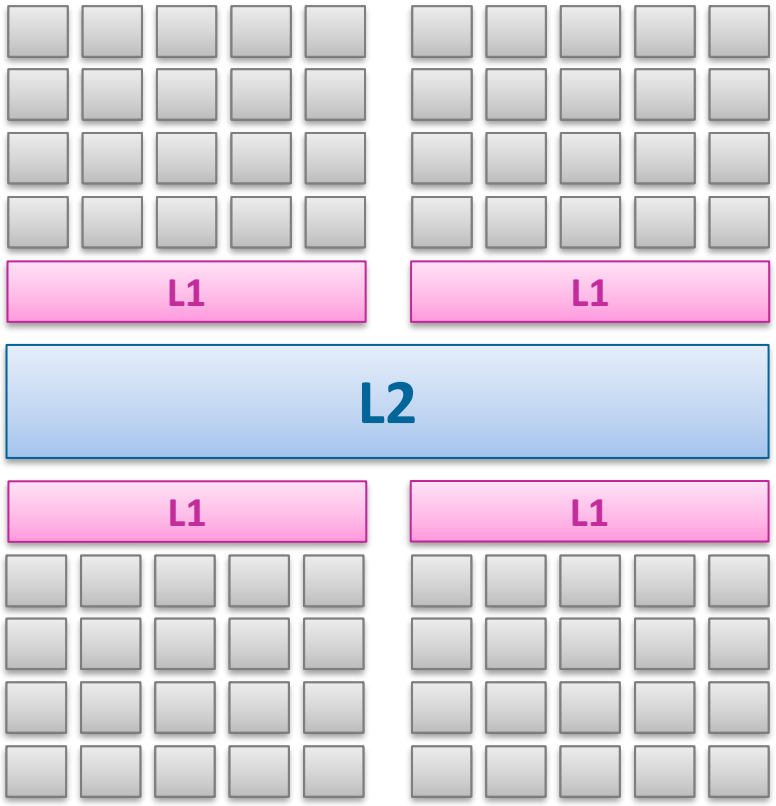
CMD 1 CMD 2 CMD 3

BATCH

waits on

Wait Idle

Wait Idle Operations



QUEUE

CMD 1

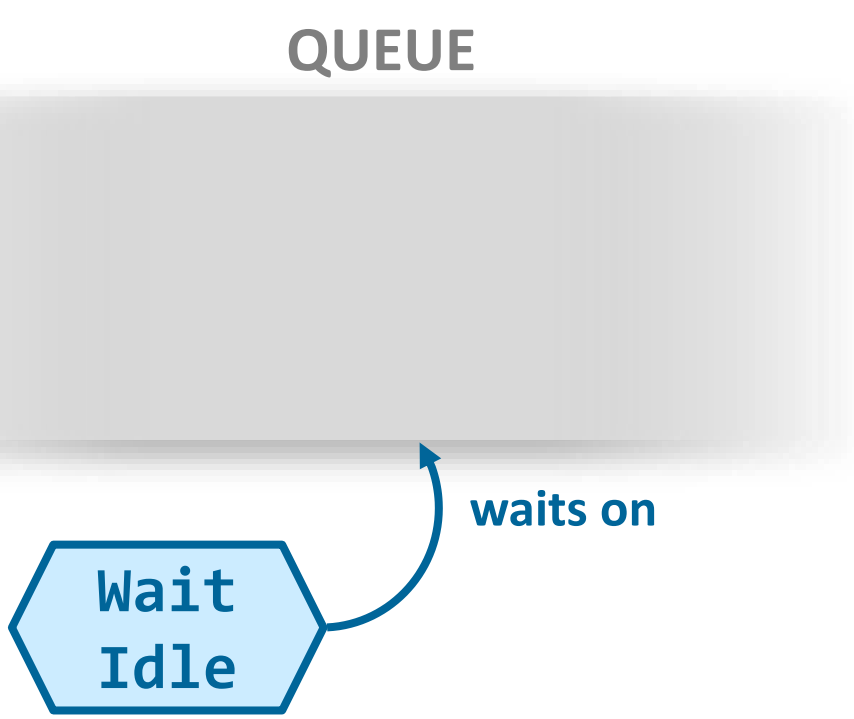
CMD 2

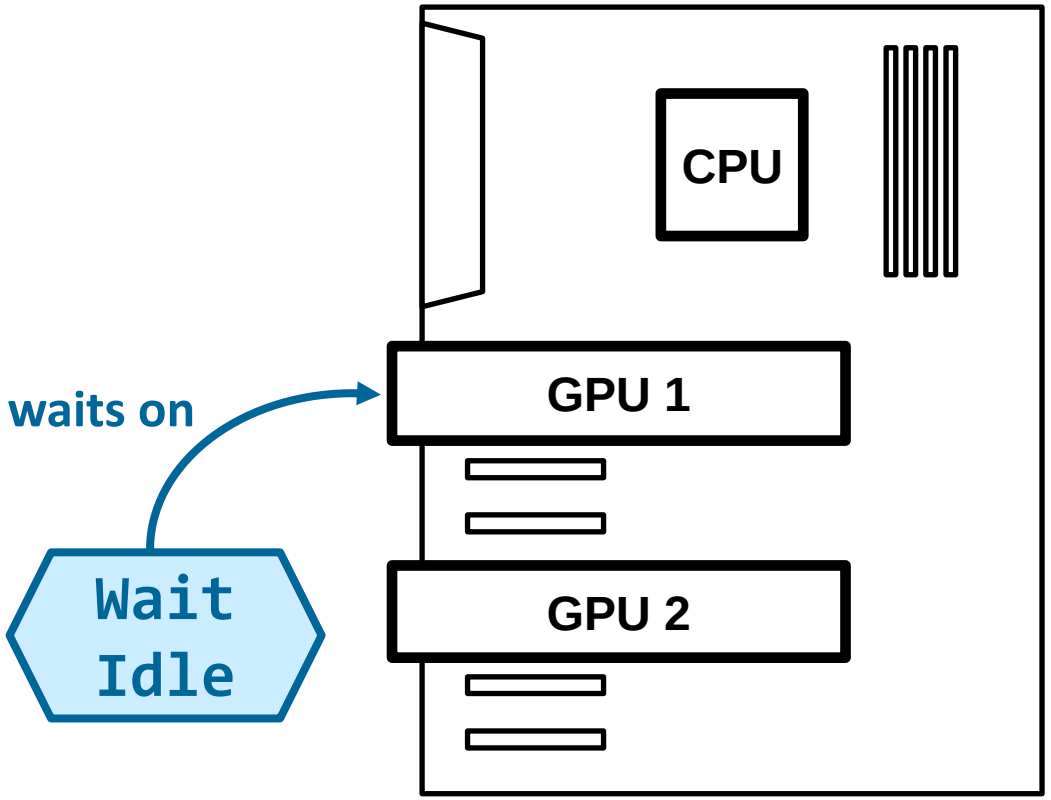
CMD 3

waits on

Wait Idle

Wait Idle Operations






```
VkQueue queue;
vkQueueWaitIdle(queue);
```

```
VkDevice device;
vkDeviceWaitIdle(device);
```

Overview of Synchronization Methods

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

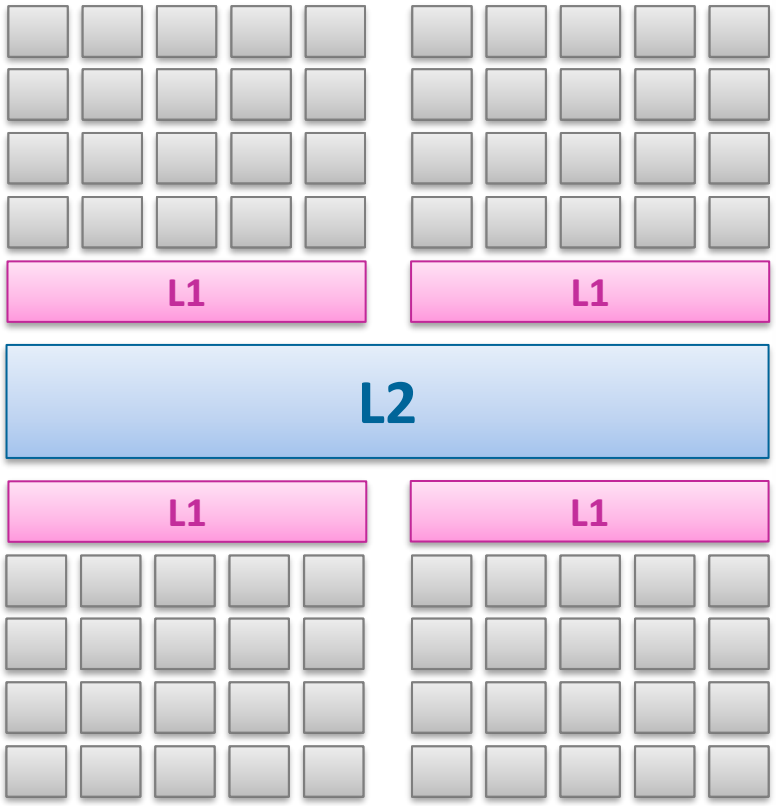
Overview of Synchronization Methods

- Wait Idle Operations (device -> host sync)
- Fences
- Semaphores
  - Binary Semaphores
  - Timeline Semaphores
- Pipeline Barriers
  - Execution Barriers
  - Memory Barriers
- Render Pass Subpass Dependencies
- Events

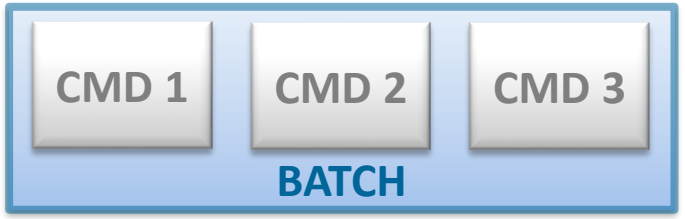
Overview of Synchronization Methods

- Wait Idle Operations (device -> host sync)
- Fences
- Semaphores
  - Binary Semaphores
  - Timeline Semaphores
- Pipeline Barriers
  - Execution Barriers
  - Memory Barriers
- Render Pass Subpass Dependencies
- Events

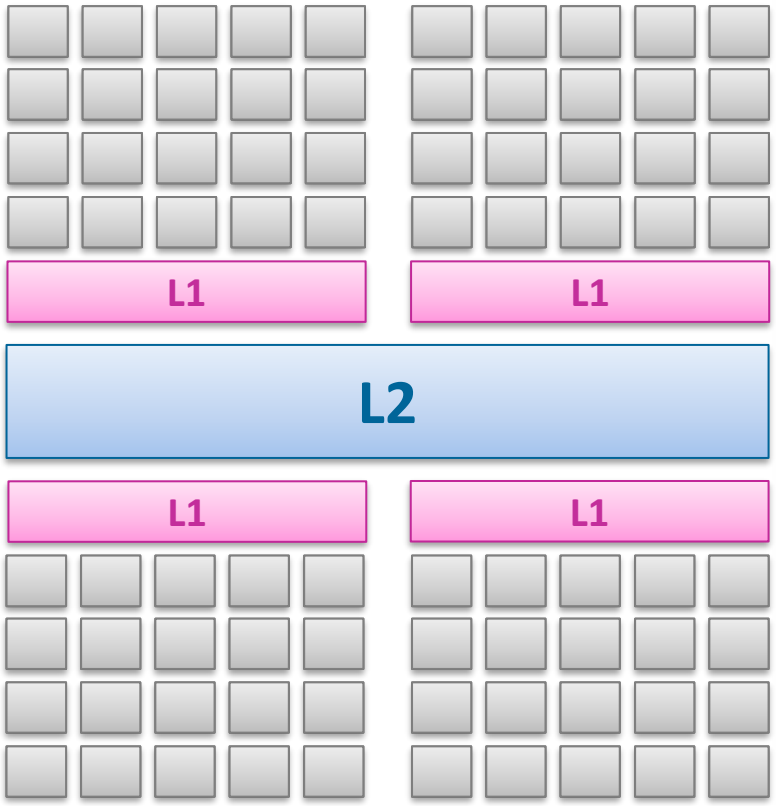
Fences

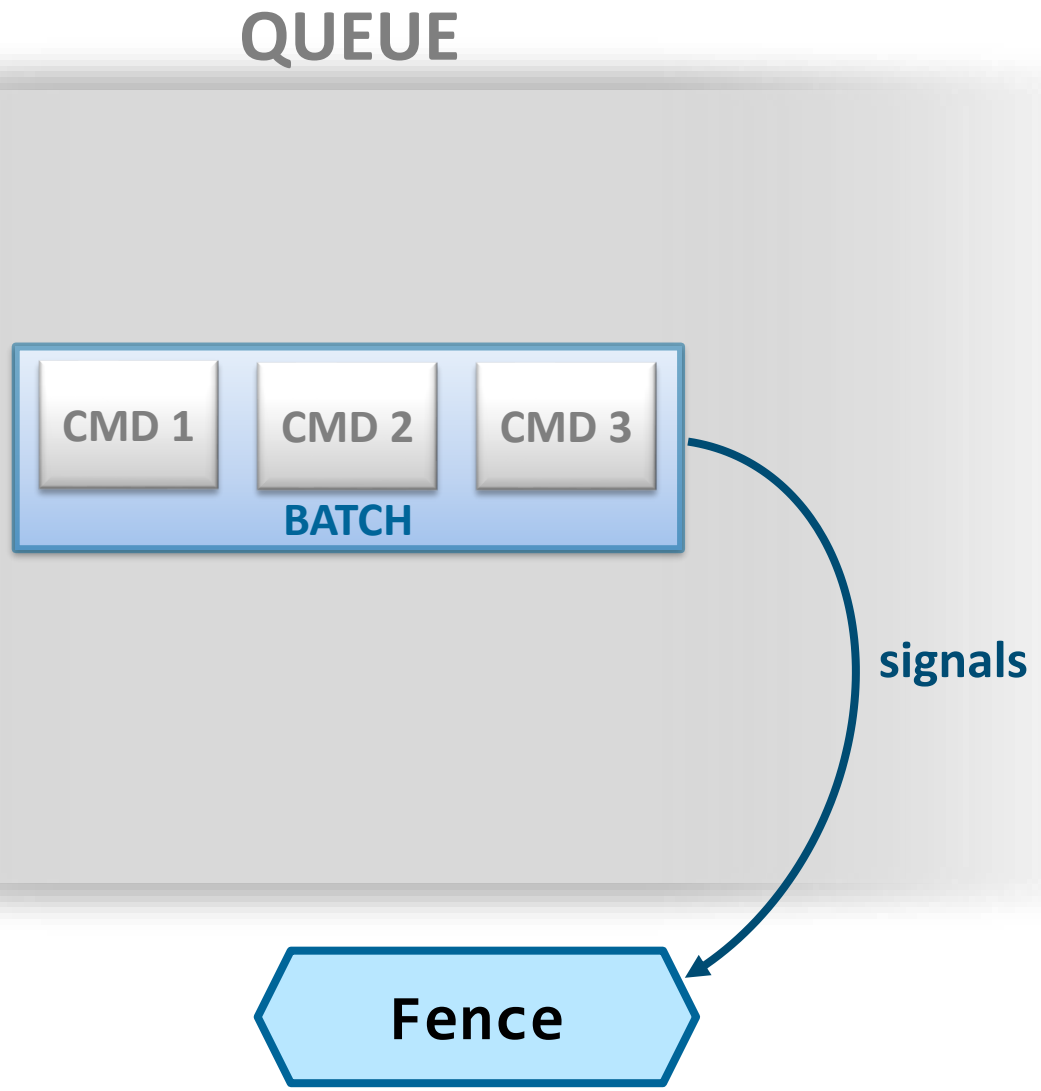


QUEUE

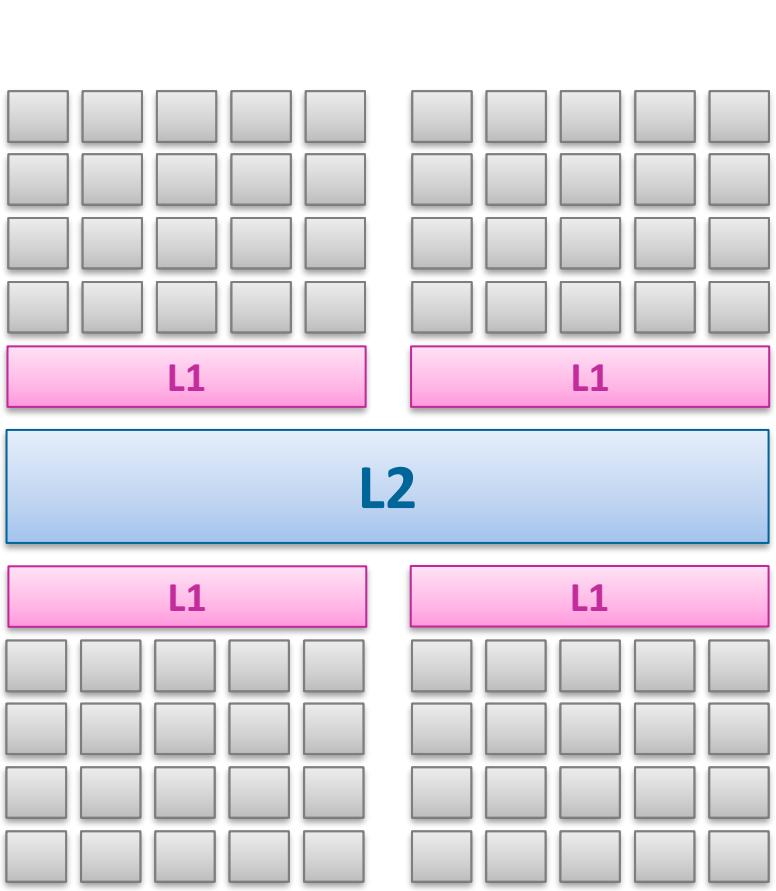


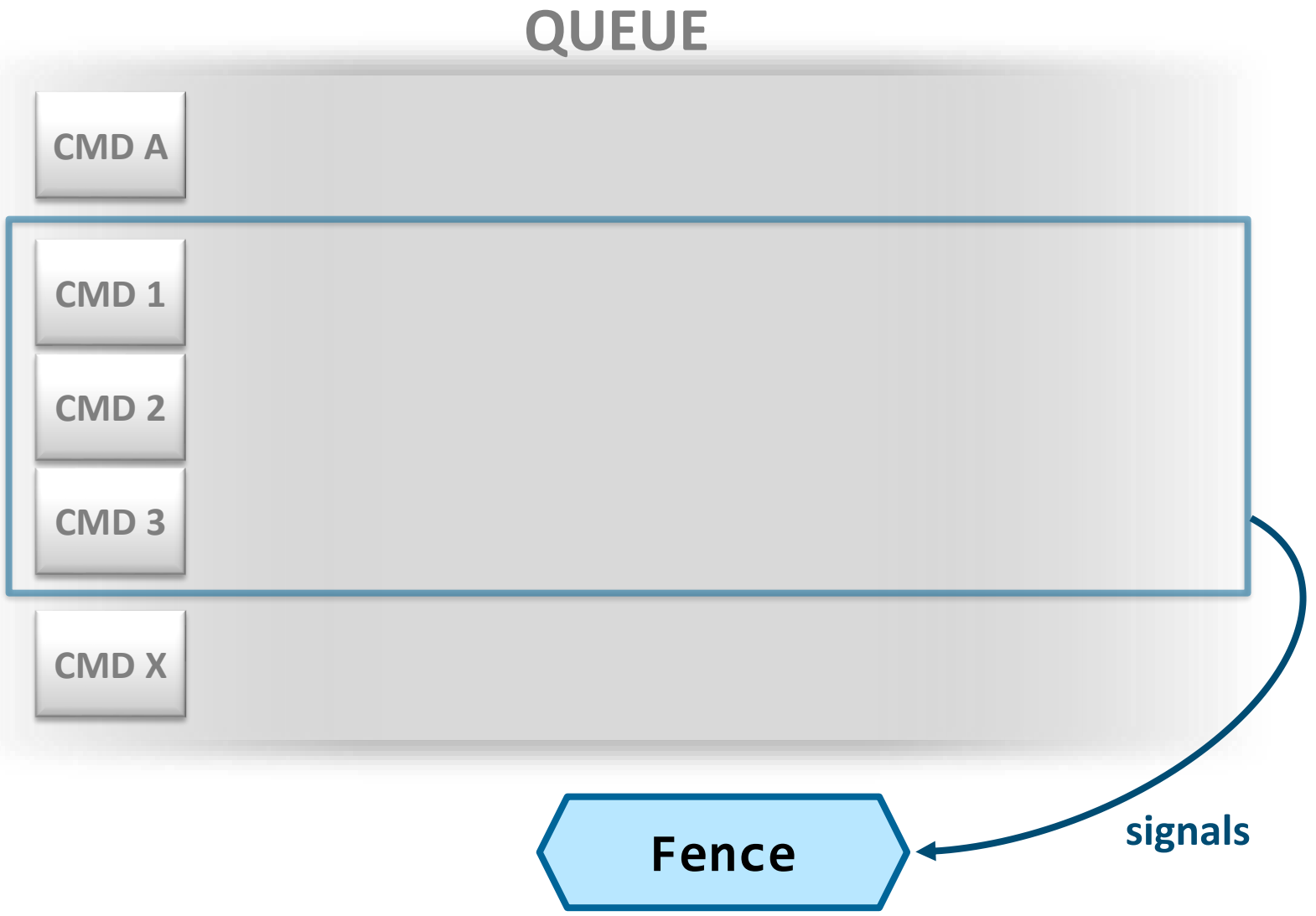
Fences



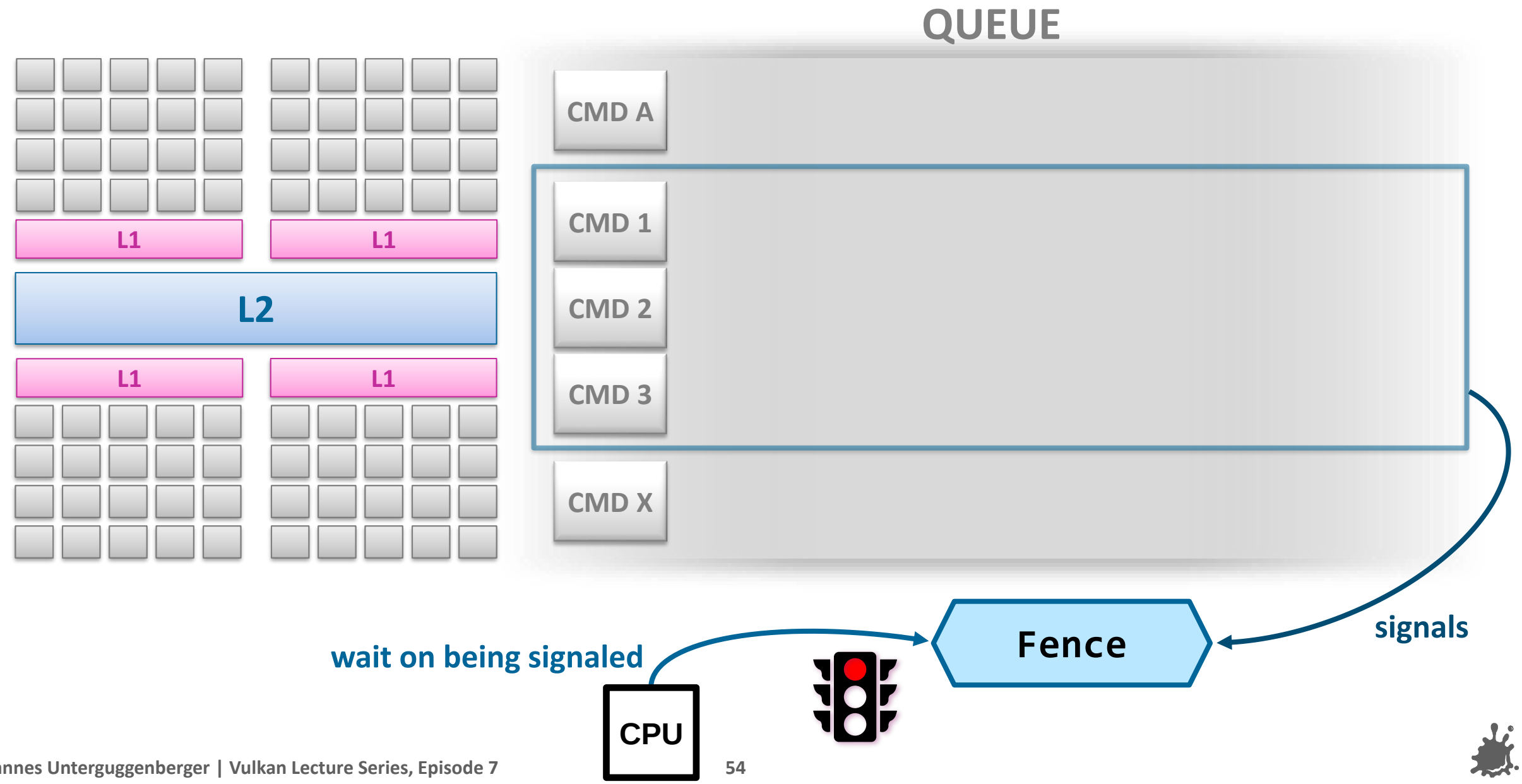


Fences

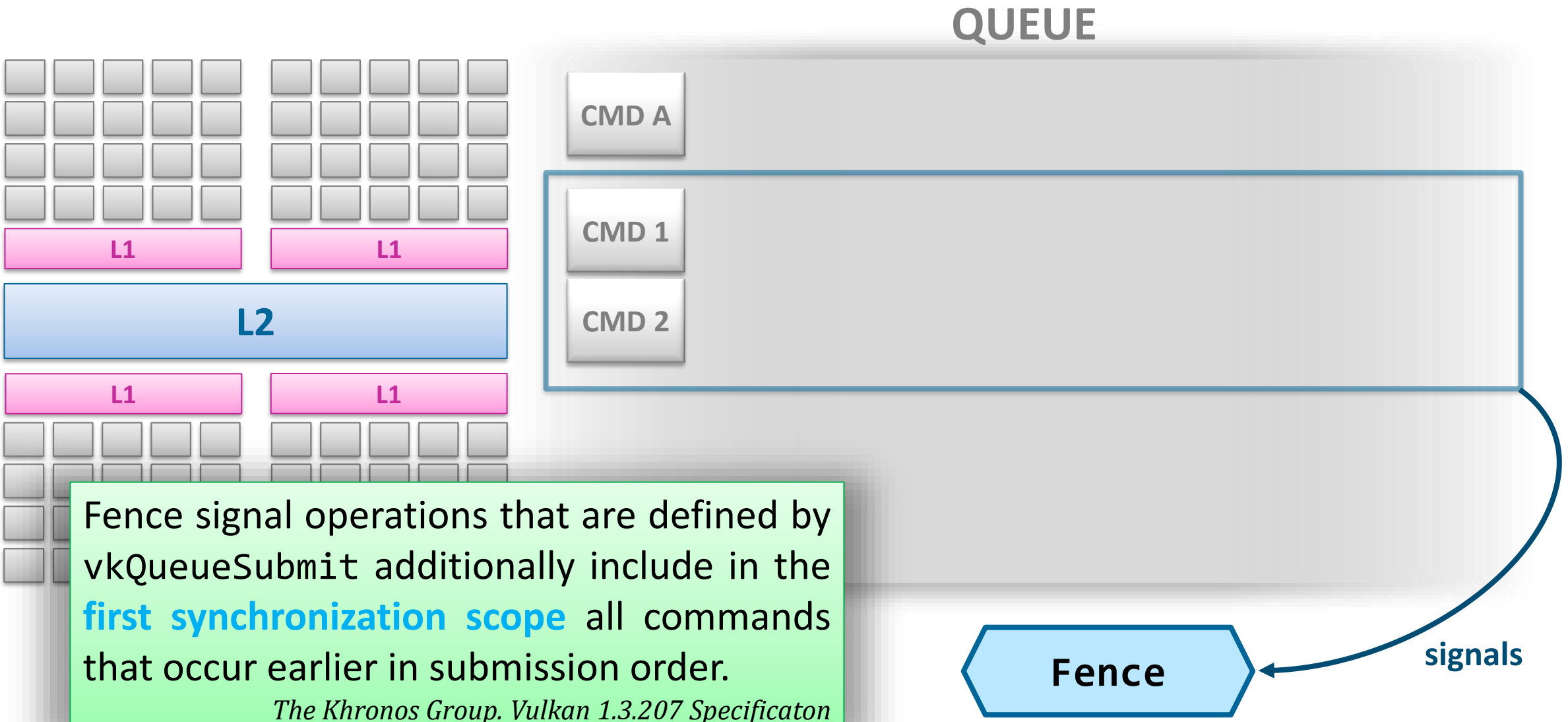




Fences


QUEUE
CMD A
CMD 1
CMD 2
CMD 3
CMD X
L1
L1
L2
L1
L1
signals
Fence
wait on being signaled
CPU

Fences



Fence signal operations that are defined by `vkQueueSubmit` additionally include in the **first synchronization scope** all commands that occur earlier in submission order.

The Khronos Group. Vulkan 1.3.207 Specificaton

Fences

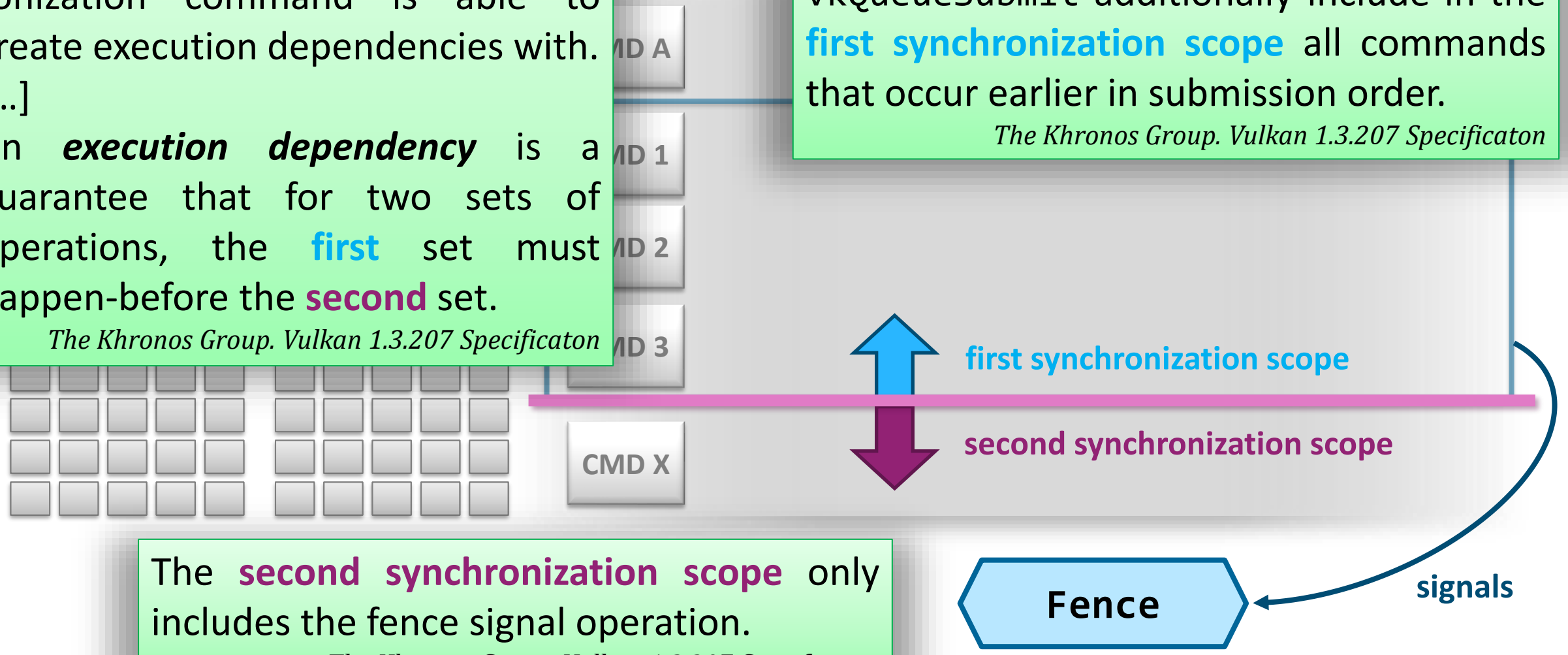
The ***synchronization scopes*** define which other operations a synch-ronization command is able to create execution dependencies with. […]

An ***execution dependency*** is a guarantee that for two sets of operations, the **first** set must happen-before the **second** set.

The Khronos Group. Vulkan 1.3.207 Specificaton

Fence signal operations that are defined by `vkQueueSubmit` additionally include in the **first synchronization scope** all commands that occur earlier in submission order.

The Khronos Group. Vulkan 1.3.207 Specificaton



The **second synchronization scope** only includes the fence signal operation.

The Khronos Group. Vulkan 1.3.207 Specificaton

Overview of Synchronization Methods

- Wait Idle Operations (device -> host sync)
- Fences
- Semaphores
  - Binary Semaphores
  - Timeline Semaphores
- Pipeline Barriers
  - Execution Barriers
  - Memory Barriers
- Render Pass Subpass Dependencies
- Events

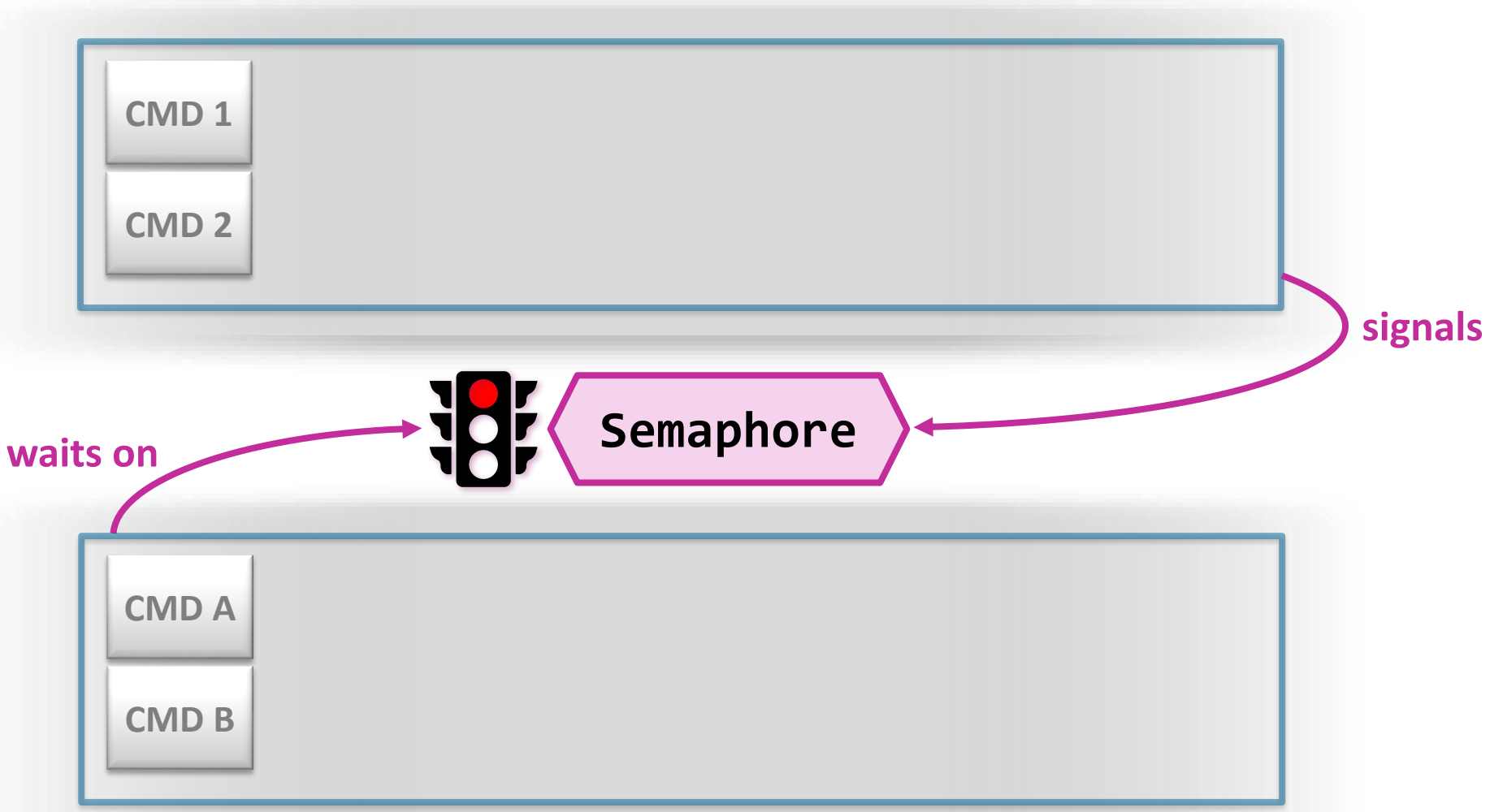
Overview of Synchronization Methods

- Wait Idle Operations (device -> host sync)
- Fences (device -> host sync)
- Semaphores
  - Binary Semaphores
  - Timeline Semaphores
- Pipeline Barriers
  - Execution Barriers
  - Memory Barriers
- Render Pass Subpass Dependencies
- Events

Overview of Synchronization Methods

- Wait Idle Operations (device -> host sync)
- Fences (device -> host sync)
- Semaphores
  - Binary Semaphores
  - Timeline Semaphores
- Pipeline Barriers
  - Execution Barriers
  - Memory Barriers
- Render Pass Subpass Dependencies
- Events

Semaphores


QUEUE 1:
CMD 1
CMD 2
signals
Semaphore
waits on
QUEUE 2:
CMD A
CMD B

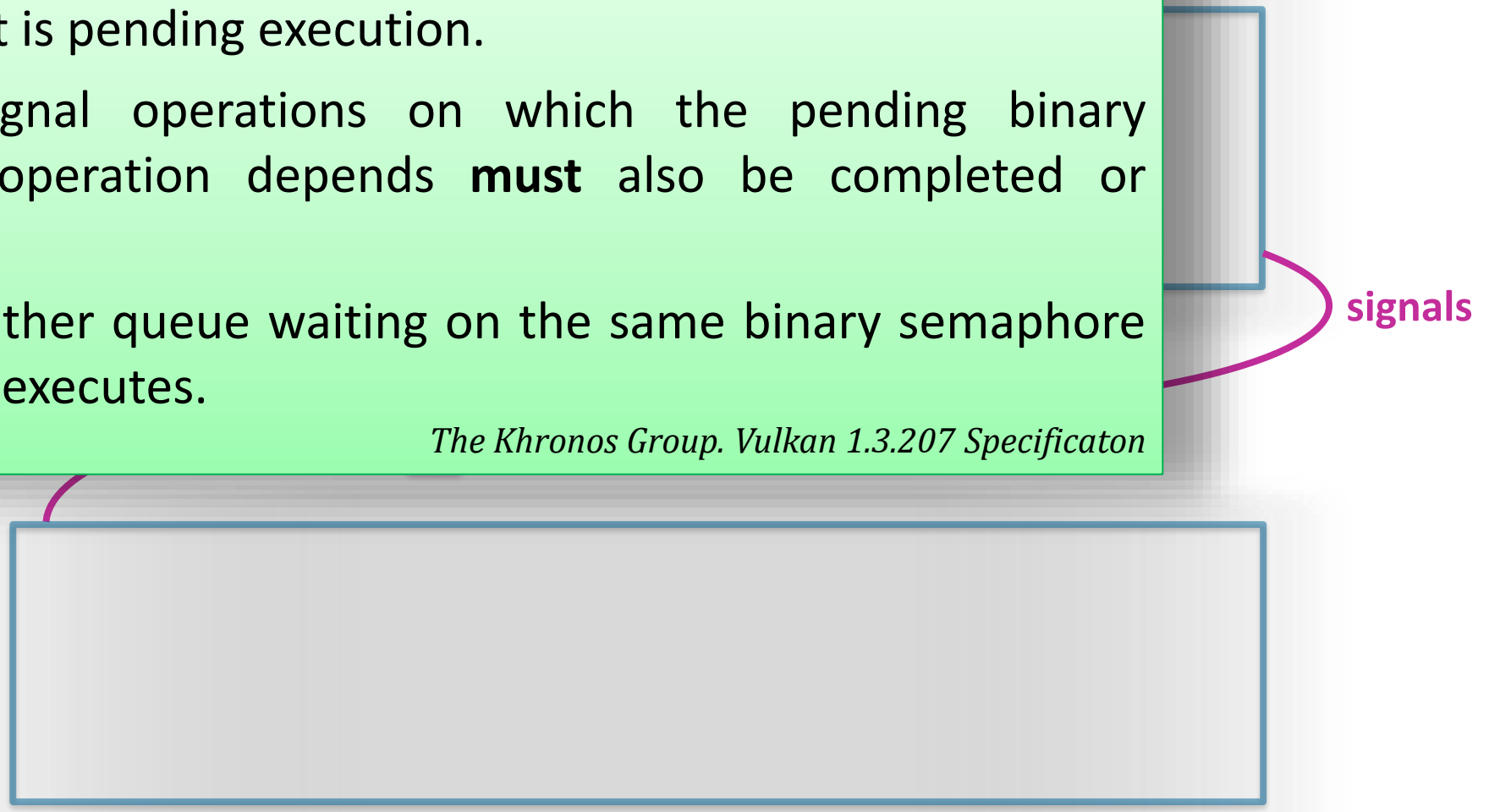
Semaphores

A binary semaphore **must** be signaled, or have an associated semaphore signal operation that is pending execution.

Any semaphore signal operations on which the pending binary semaphore signal operation depends **must** also be completed or pending execution.

There **must** be no other queue waiting on the same binary semaphore when the operation executes.

The Khronos Group. Vulkan 1.3.207 Specificaton



Overview of Synchronization Methods

- Wait Idle Operations (device -> host sync)
- Fences (device -> host sync)
- Semaphores
  - Binary Semaphores
  - Timeline Semaphores
- Pipeline Barriers
  - Execution Barriers
  - Memory Barriers
- Render Pass Subpass Dependencies
- Events

Overview of Synchronization Methods

- Wait Idle Operations (device -> host sync)
- Fences (device -> host sync)
- Semaphores (queue sync)
  - Binary Semaphores
  - Timeline Semaphores
- Pipeline Barriers
  - Execution Barriers
  - Memory Barriers
- Render Pass Subpass Dependencies
- Events

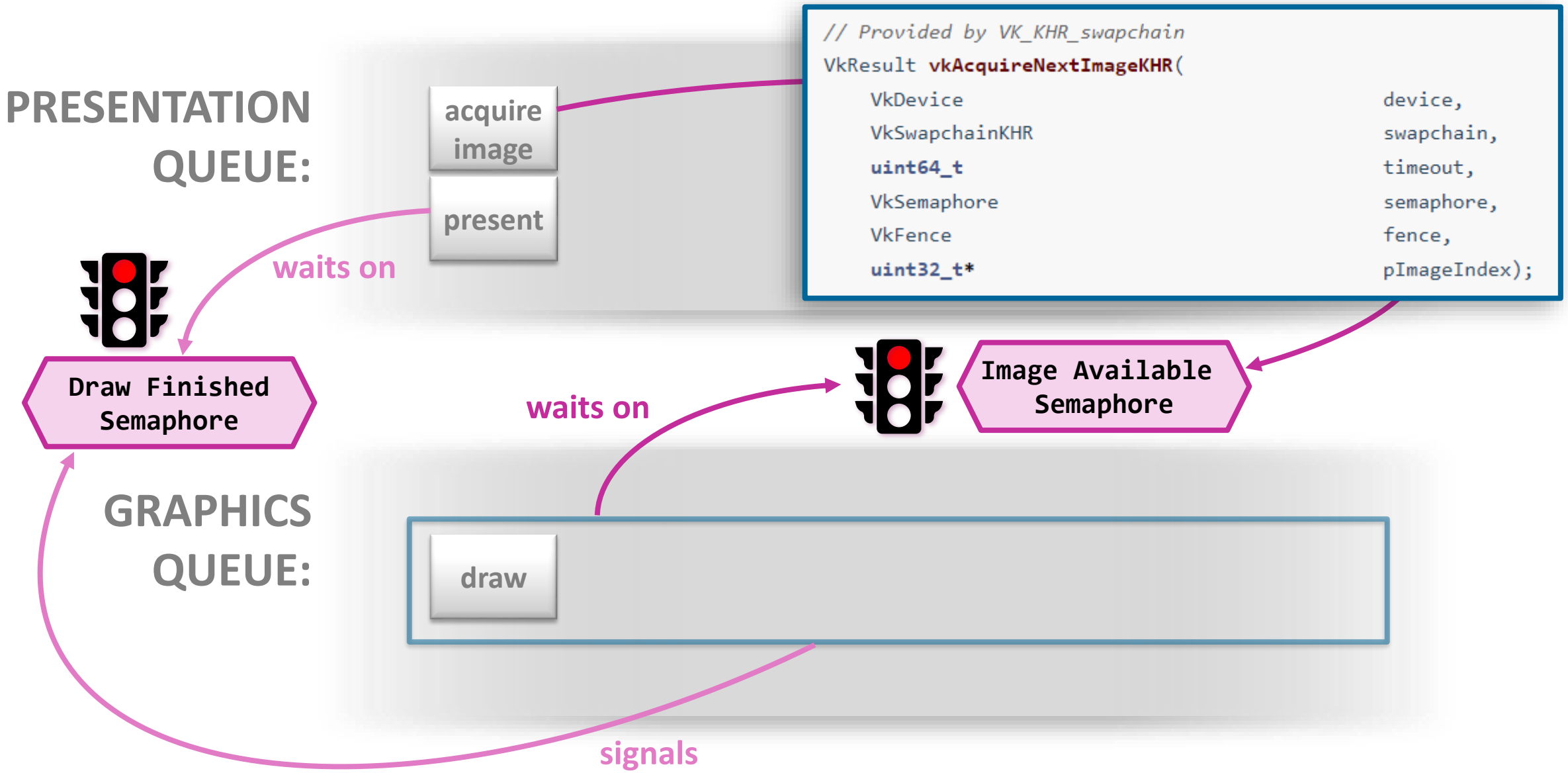
Overview of Synchronization Methods

- Wait Idle Operations (device -> host sync)
- Fences (device -> host sync)
- Semaphores (queue sync)
  - Binary Semaphores
  - Timeline Semaphores
- Pipeline Barriers
  - Execution Barriers
  - Memory Barriers
- Render Pass Subpass Dependencies
- Events

Overview of Synchronization Methods

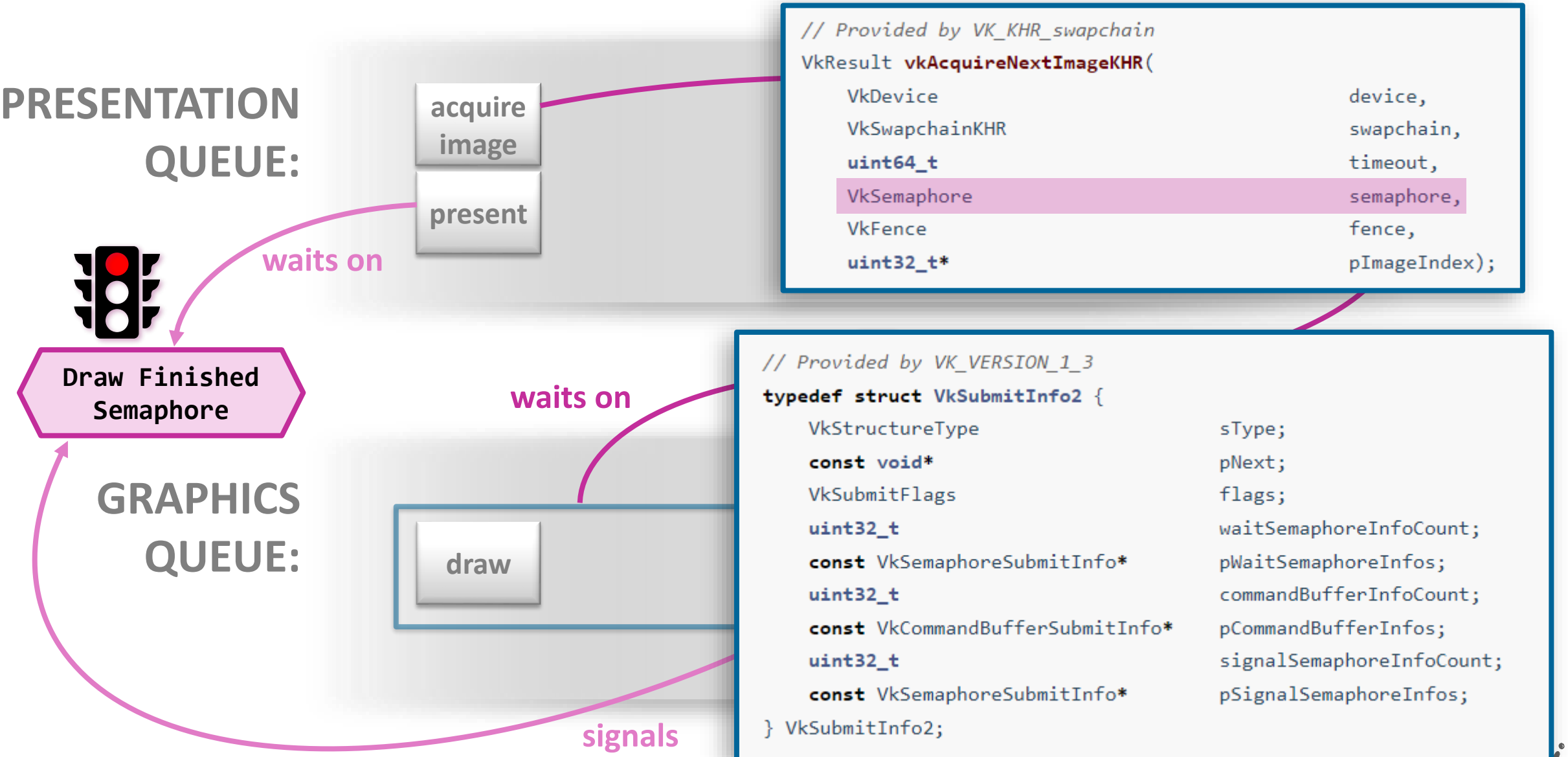
- Wait Idle Operations (device -> host sync)
- Fences (device -> host sync)
- Semaphores (queue sync)
  - Binary Semaphores (queue -> queue sync, device only)
  - Timeline Semaphores
- Pipeline Barriers
  - Execution Barriers
  - Memory Barriers
- Render Pass Subpass Dependencies
- Events

Binary Semaphores




```
// Provided by VK_KHR_swapchain
VkResult vkAcquireNextImageKHR(
    VkDevice                                    device,
    VkSwapchainKHR                              swapchain,
    uint64_t                                    timeout,
    VkSemaphore                                 semaphore,
    VkFence                                     fence,
    uint32_t*                                   pImageIndex);
```

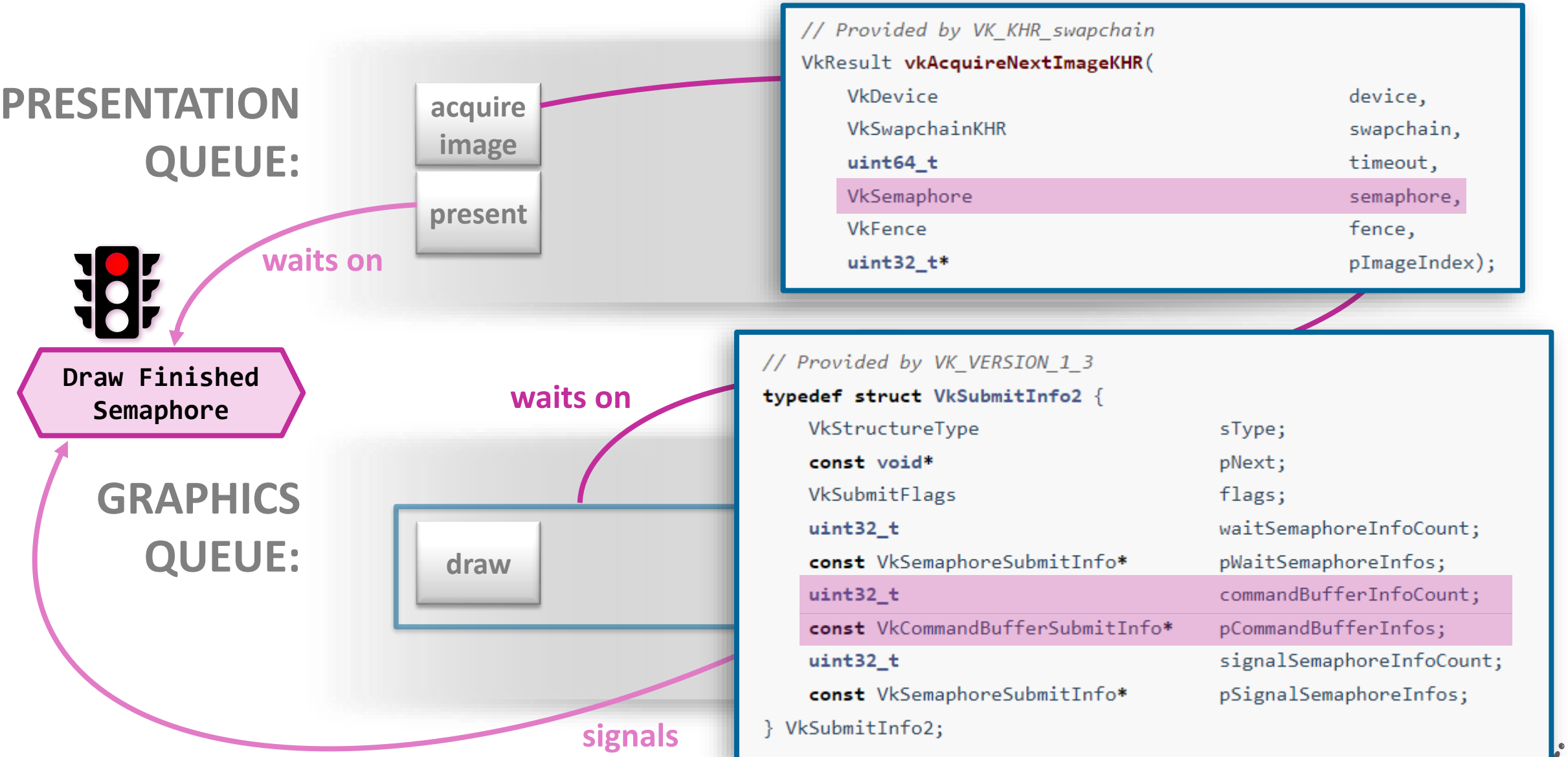
Binary Semaphores




```
// Provided by VK_KHR_swapchain
VkResult vkAcquireNextImageKHR(
    VkDevice                                    device,
    VkSwapchainKHR                              swapchain,
    uint64_t                                    timeout,
    VkSemaphore                                 semaphore,
    VkFence                                     fence,
    uint32_t*                                   pImageIndex);
```

```
// Provided by VK_VERSION_1_3
typedef struct VkSubmitInfo2 {
    VkStructureType                             sType;
    const void*                                 pNext;
    VkSubmitFlags                               flags;
    uint32_t                                    waitSemaphoreInfoCount;
    const VkSemaphoreSubmitInfo*                pWaitSemaphoreInfos;
    uint32_t                                    commandBufferInfoCount;
    const VkCommandBufferSubmitInfo*            pCommandBufferInfos;
    uint32_t                                    signalSemaphoreInfoCount;
    const VkSemaphoreSubmitInfo*                pSignalSemaphoreInfos;
} VkSubmitInfo2;
```

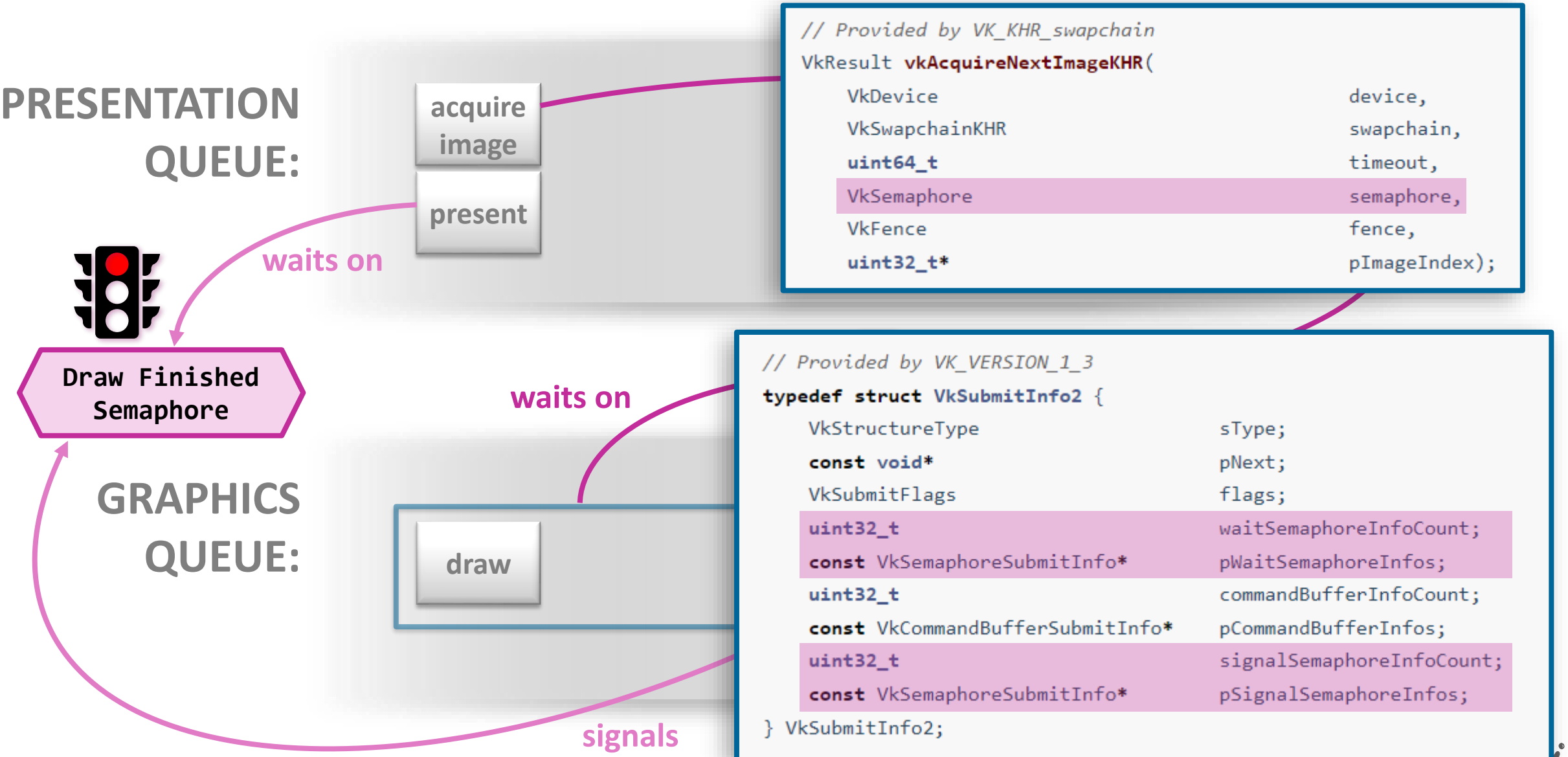
Binary Semaphores




```
// Provided by VK_KHR_swapchain
VkResult vkAcquireNextImageKHR(
    VkDevice                                device,
    VkSwapchainKHR                          swapchain,
    uint64_t                                timeout,
    VkSemaphore                             semaphore,
    VkFence                                 fence,
    uint32_t*                               pImageIndex);
```

```
// Provided by VK_VERSION_1_3
typedef struct VkSubmitInfo2 {
    VkStructureType                         sType;
    const void*                             pNext;
    VkSubmitFlags                           flags;
    uint32_t                                waitSemaphoreInfoCount;
    const VkSemaphoreSubmitInfo*            pWaitSemaphoreInfos;
    uint32_t                                commandBufferInfoCount;
    const VkCommandBufferSubmitInfo*        pCommandBufferInfos;
    uint32_t                                signalSemaphoreInfoCount;
    const VkSemaphoreSubmitInfo*            pSignalSemaphoreInfos;
} VkSubmitInfo2;
```

Binary Semaphores




```
// Provided by VK_KHR_swapchain
VkResult vkAcquireNextImageKHR(
    VkDevice                                    device,
    VkSwapchainKHR                              swapchain,
    uint64_t                                    timeout,
    VkSemaphore                                 semaphore,
    VkFence                                     fence,
    uint32_t*                                   pImageIndex);
```

```
// Provided by VK_VERSION_1_3
typedef struct VkSubmitInfo2 {
    VkStructureType                             sType;
    const void*                                 pNext;
    VkSubmitFlags                               flags;
    uint32_t                                    waitSemaphoreInfoCount;
    const VkSemaphoreSubmitInfo*                pWaitSemaphoreInfos;
    uint32_t                                    commandBufferInfoCount;
    const VkCommandBufferSubmitInfo*            pCommandBufferInfos;
    uint32_t                                    signalSemaphoreInfoCount;
    const VkSemaphoreSubmitInfo*                pSignalSemaphoreInfos;
} VkSubmitInfo2;
```

Binary Semaphores

PRESENTATION QUEUE:

acquire image

present

waits on

```
// Provided by VK_KHR_swapchain
VkResult vkAcquireNextImageKHR(
    VkDevice                    device,
    VkSwapchainKHR              swapchain,
    uint64_t                    timeout,
    VkSemaphore                 semaphore,
    VkFence                     fence,
    uint32_t*                   pImageIndex);
```

```
// Provided by VK_KHR_swapchain
typedef struct VkPresentInfoKHR {
    VkStructureType          sType;
    const void*              pNext;
    uint32_t                 waitSemaphoreCount;
    const VkSemaphore*       pWaitSemaphores;
    uint32_t                 swapchainCount;
    const VkSwapchainKHR*    pSwapchains;
    const uint32_t*          pImageIndices;
    VkResult*                pResults;
} VkPresentInfoKHR;
```

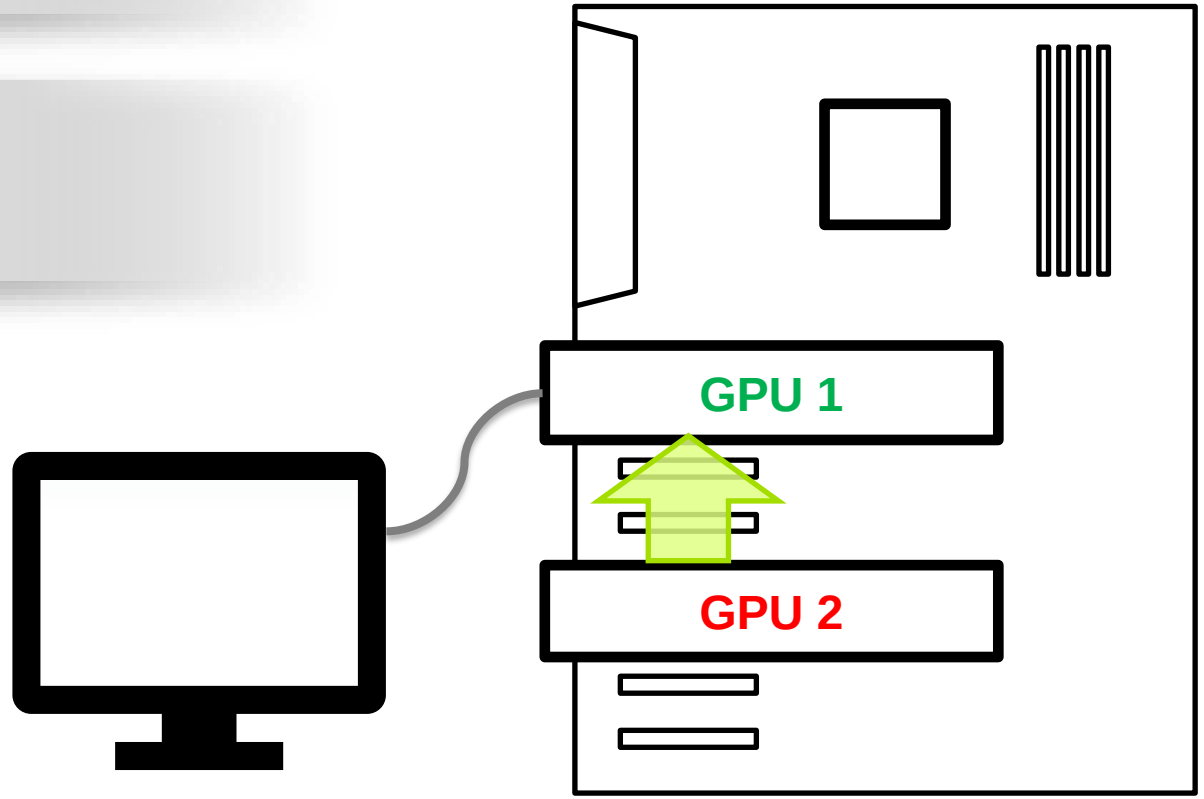
```
// Provided by VK_VERSION_1_3
typedef struct VkSubmitInfo2 {
    VkStructureType                  sType;
    const void*                      pNext;
    VkSubmitFlags                    flags;
    uint32_t                         waitSemaphoreInfoCount;
    const VkSemaphoreSubmitInfo*     pWaitSemaphoreInfos;
    uint32_t                         commandBufferInfoCount;
    const VkCommandBufferSubmitInfo* pCommandBufferInfos;
    uint32_t                         signalSemaphoreInfoCount;
    const VkSemaphoreSubmitInfo*     pSignalSemaphoreInfos;
} VkSubmitInfo2;
```

PRESENTATION QUEUE:

present

GRAPHICS QUEUE:

draw



PRESENTATION QUEUE:

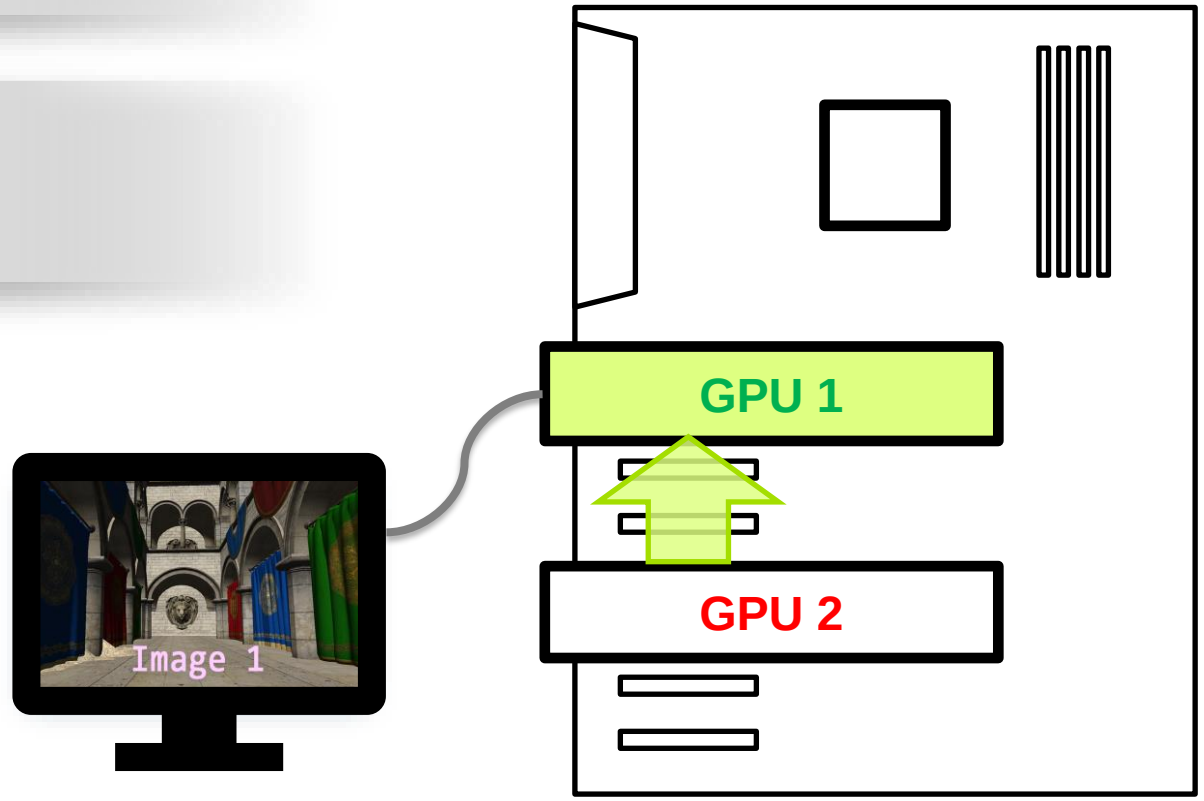
present

All elements of the `pWaitSemaphores` member of `pPresentInfo` **must** be created with a `VkSemaphoreType` of `VK_SEMAPHORE_TYPE_BINARY`.

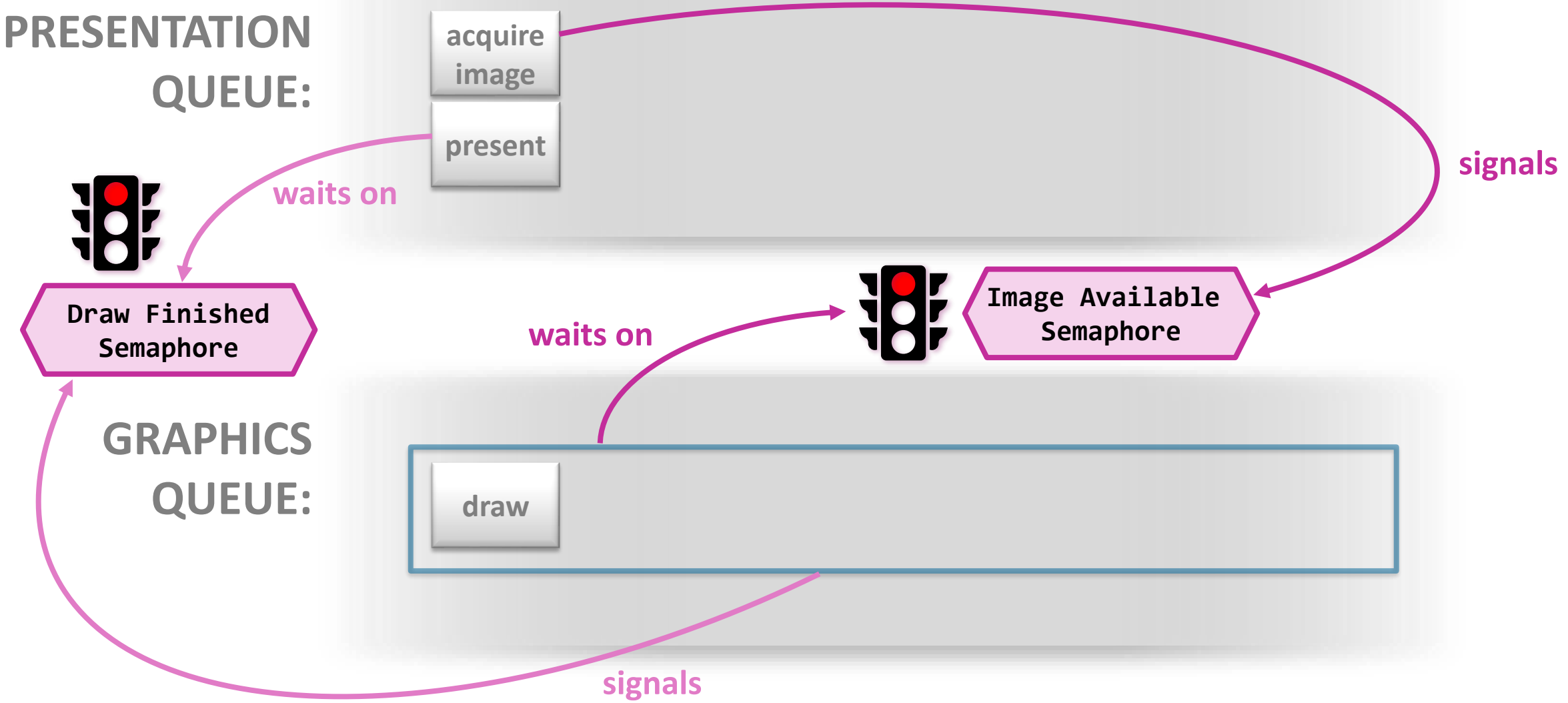
The Khronos Group. Vulkan 1.3.207 Specificaton

All elements of the `pWaitSemaphores` member of `pPresentInfo` **must** be semaphores that are signaled, or have semaphore signal operations previously submitted for execution.

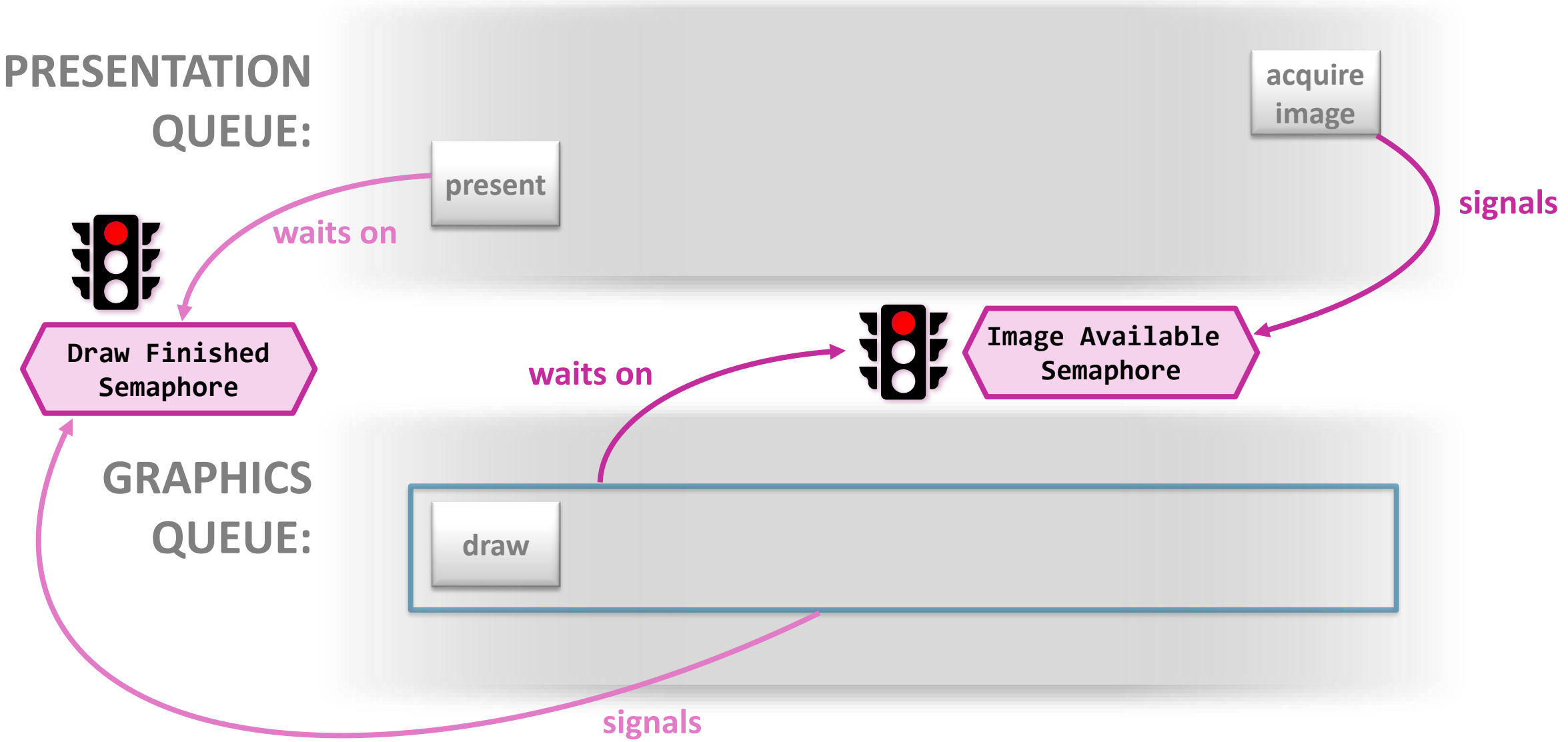
The Khronos Group. Vulkan 1.3.207 Specificaton



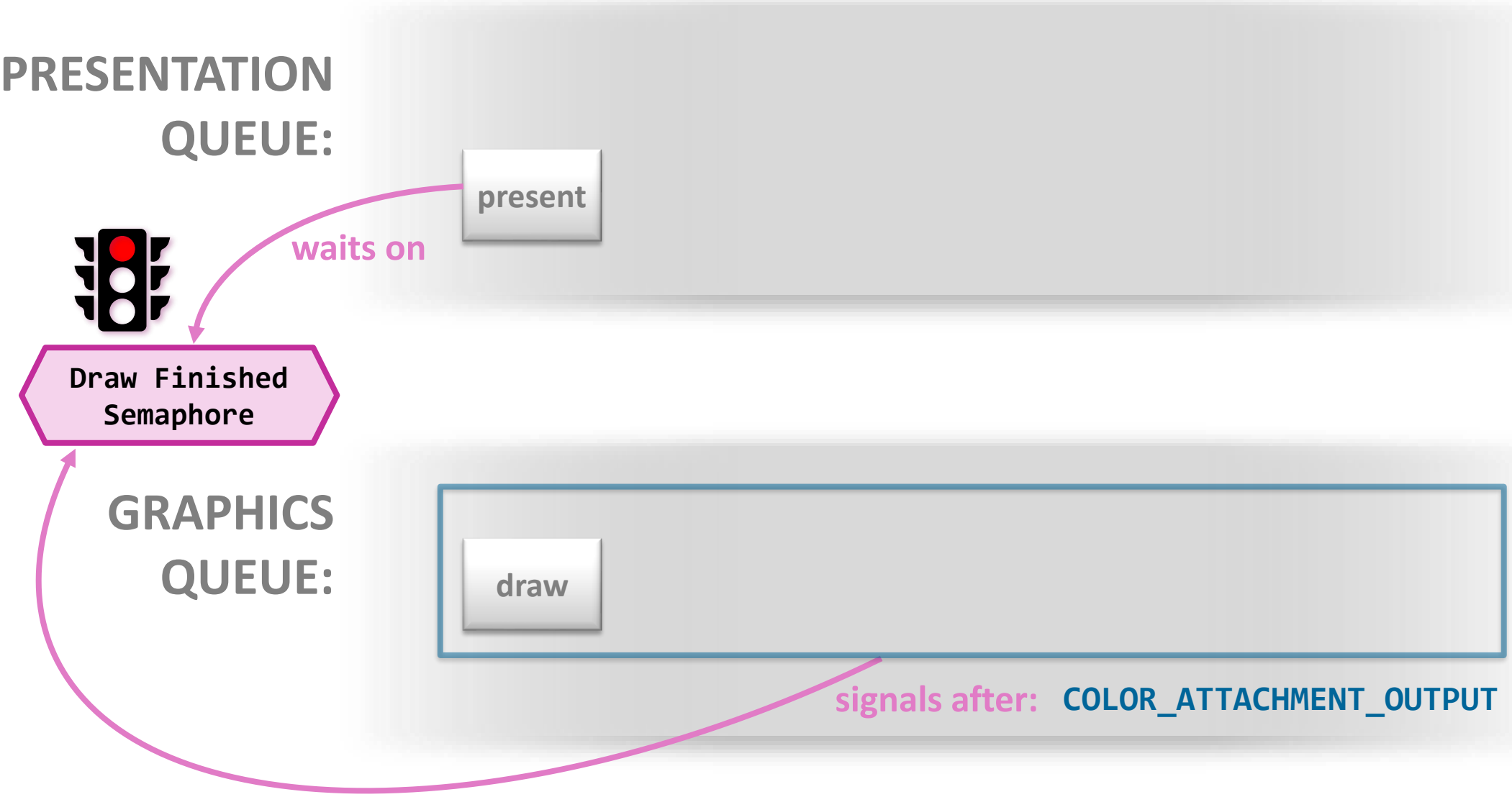
Binary Semaphores



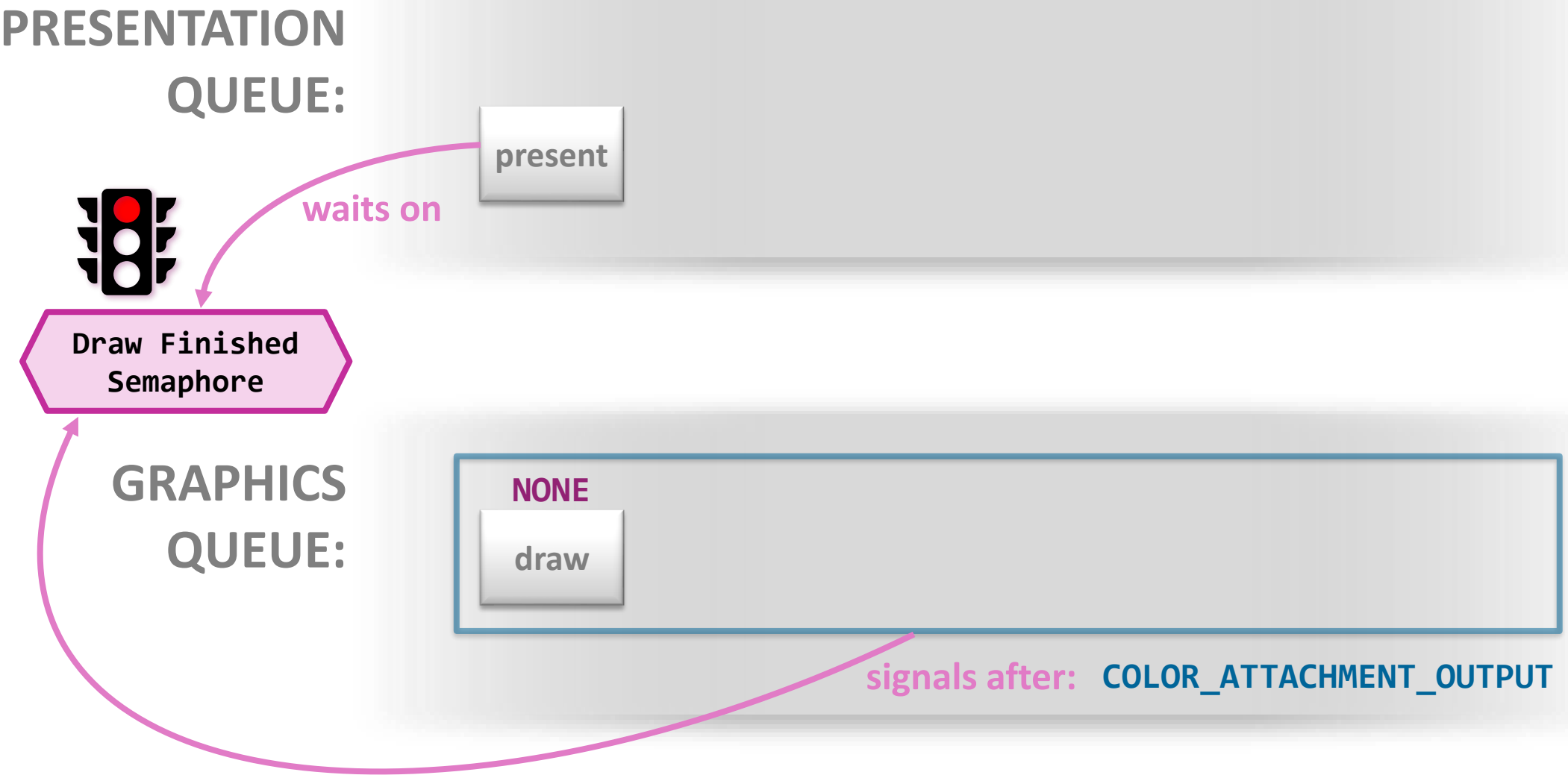
Binary Semaphores



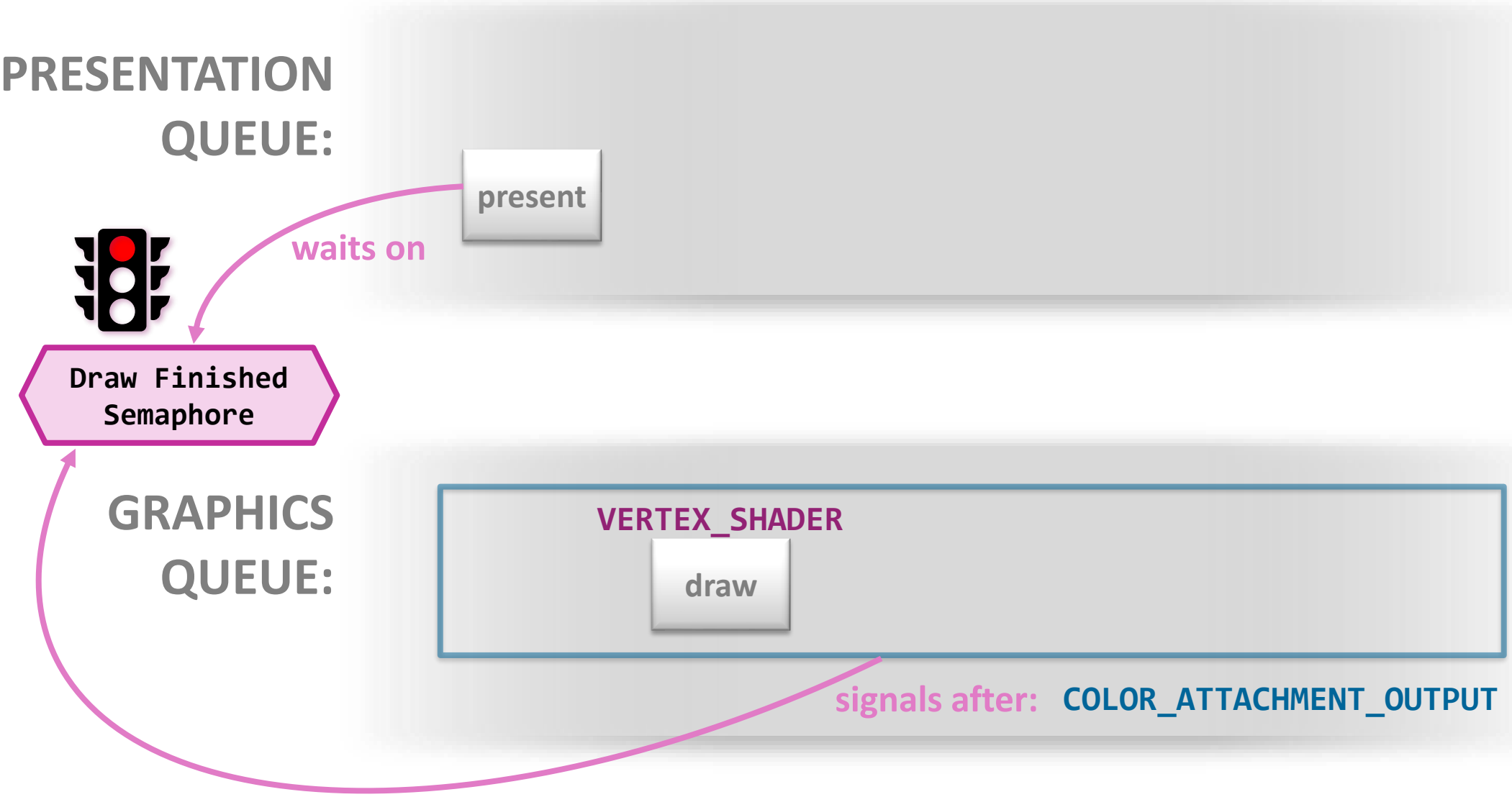
Binary Semaphores



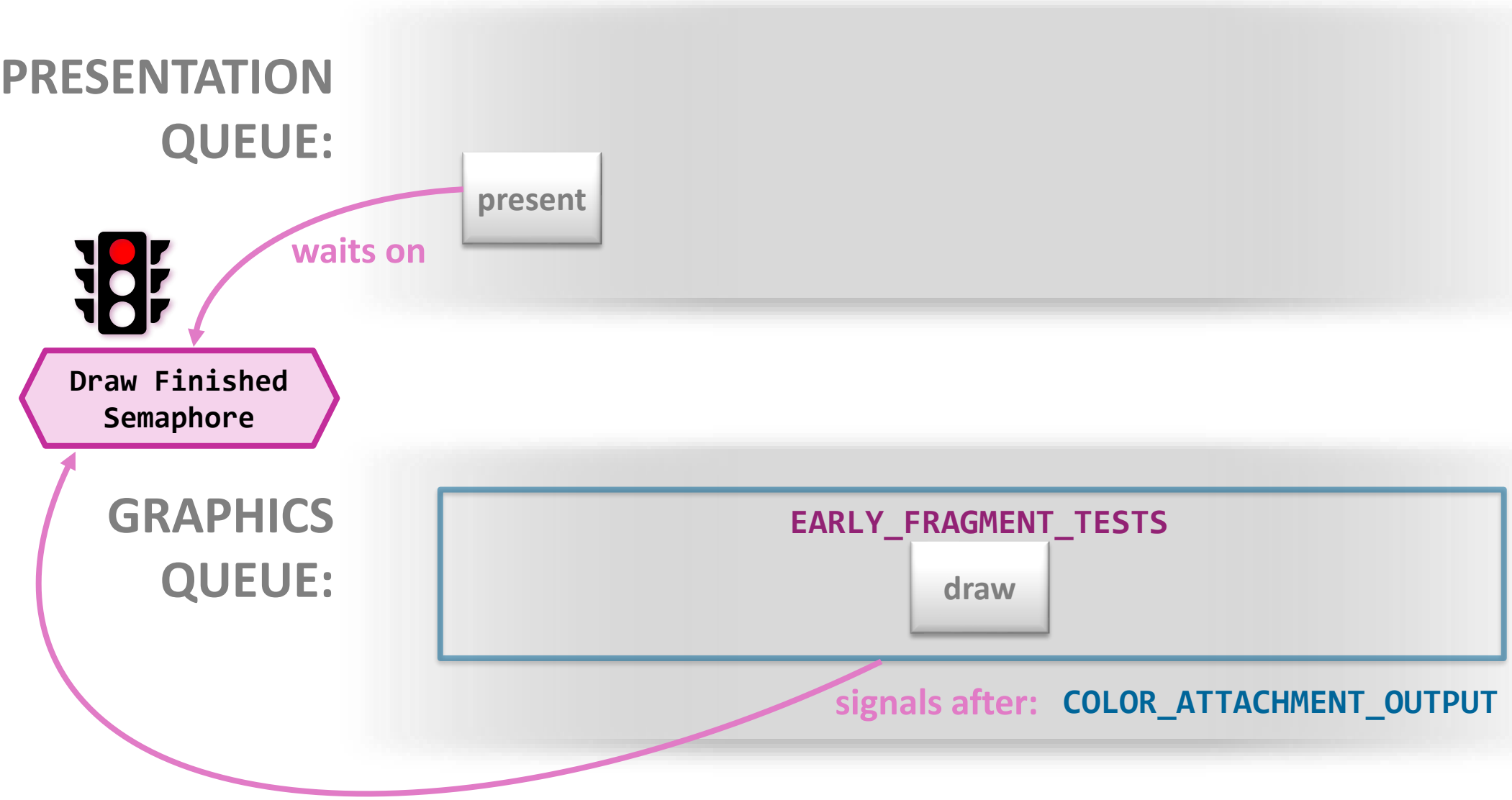
Binary Semaphores



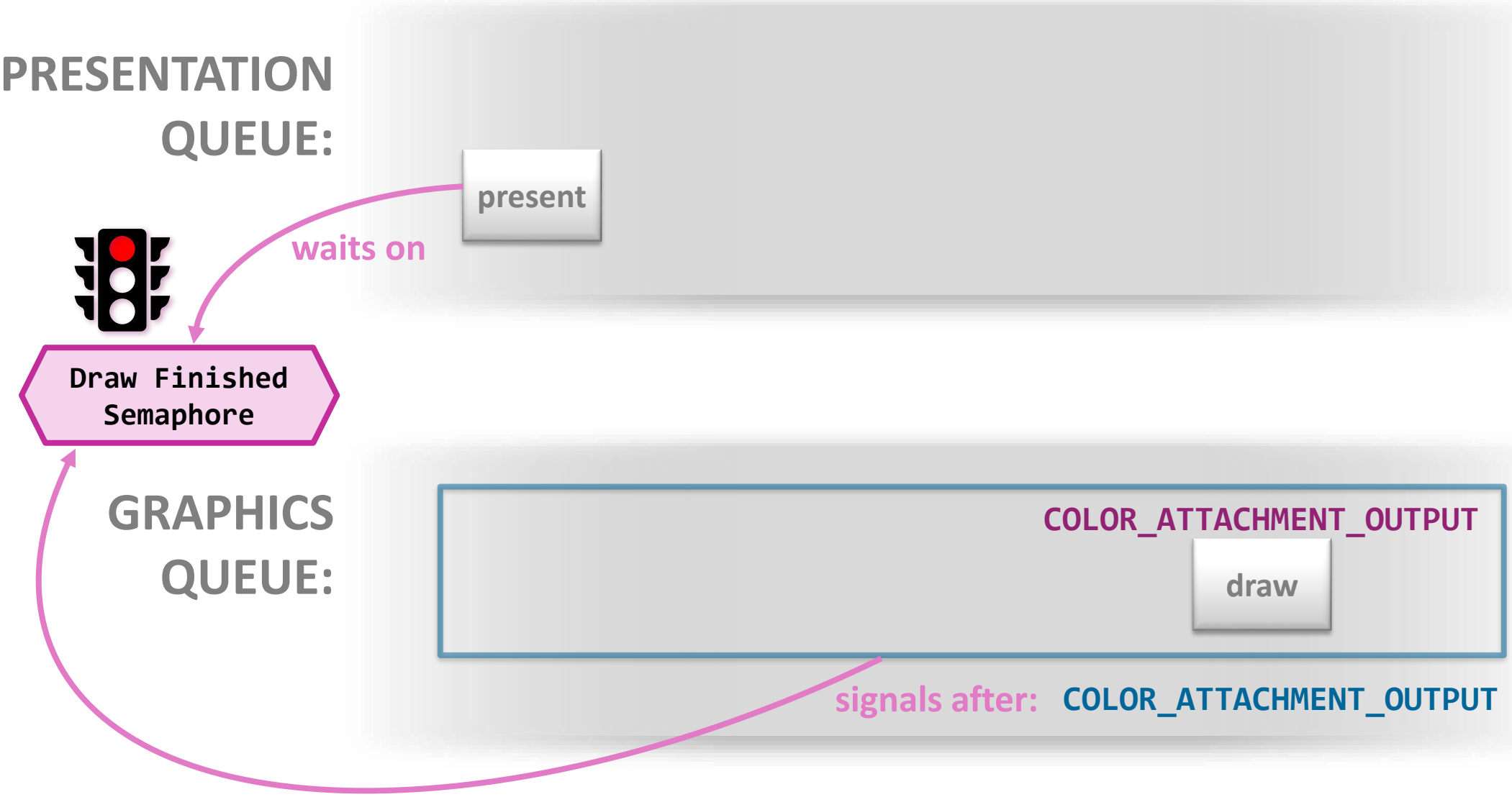
Binary Semaphores



Binary Semaphores



Binary Semaphores



Binary Semaphores

PRESENTATION QUEUE:

present

waits on

Draw Finished Semaphore

GRAPHICS QUEUE:

draw

signals after: COLOR_ATTACHMENT_OUTPUT

Overview of Synchronization Methods

- Wait Idle Operations (device -> host sync)
- Fences (device -> host sync)
- Semaphores (queue sync)
  - Binary Semaphores (queue -> queue sync, device only)
  - Timeline Semaphores
- Pipeline Barriers
  - Execution Barriers
  - Memory Barriers
- Render Pass Subpass Dependencies
- Events

Overview of Synchronization Methods

- Wait Idle Operations (device -> host sync)
- Fences (device -> host sync)
- Semaphores (queue sync)
  - Binary Semaphores (queue -> queue sync, device only)
  - Timeline Semaphores
- Pipeline Barriers
  - Execution Barriers
  - Memory Barriers
- Render Pass Subpass Dependencies
- Events

Timeline Semaphores

integer payload

[…] identifying a point in a timeline […]

The Khronos Group. Vulkan 1.3.213 Specificaton

[…] strictly increasing 64-bit unsigned integer payload […]

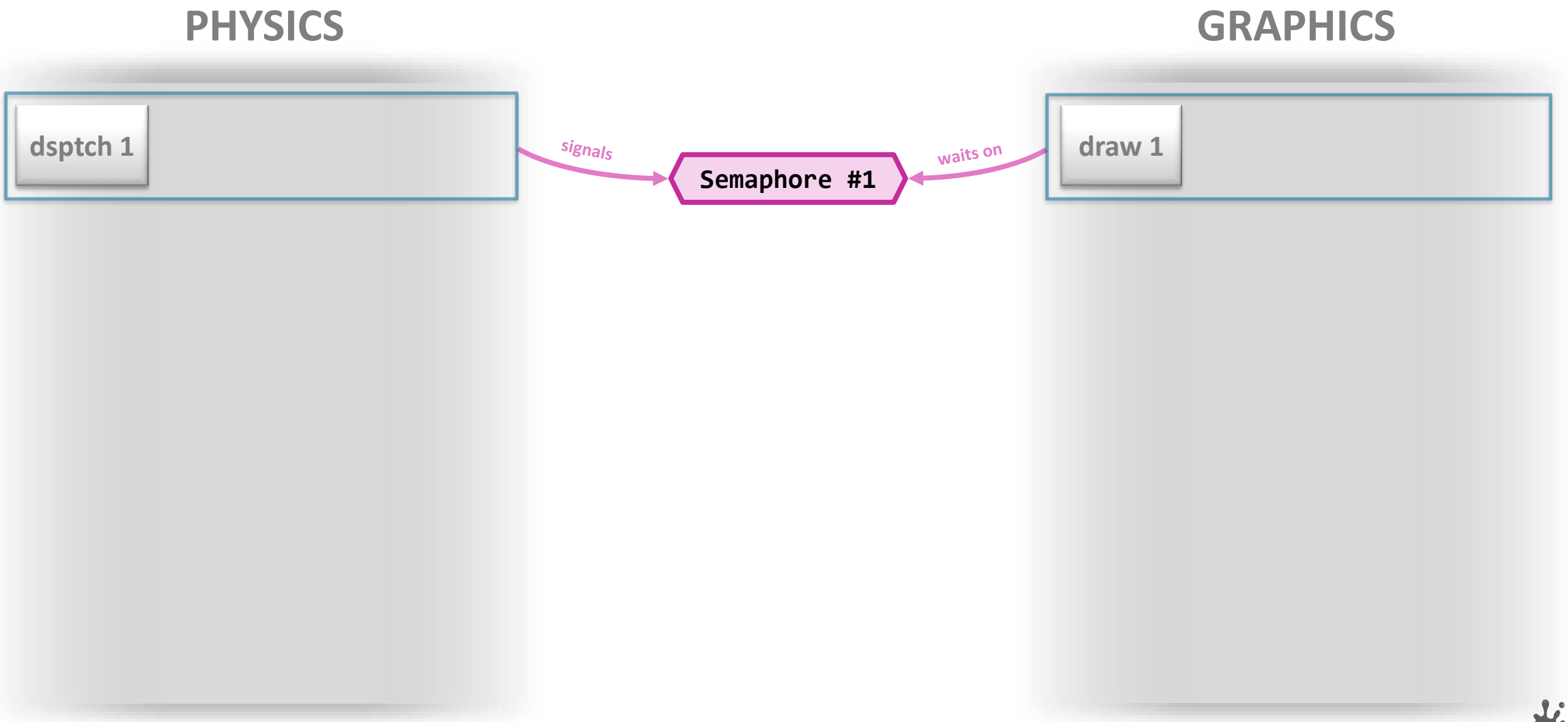
The Khronos Group. Vulkan 1.3.213 Specificaton

Such timeline semaphores support the following operations:

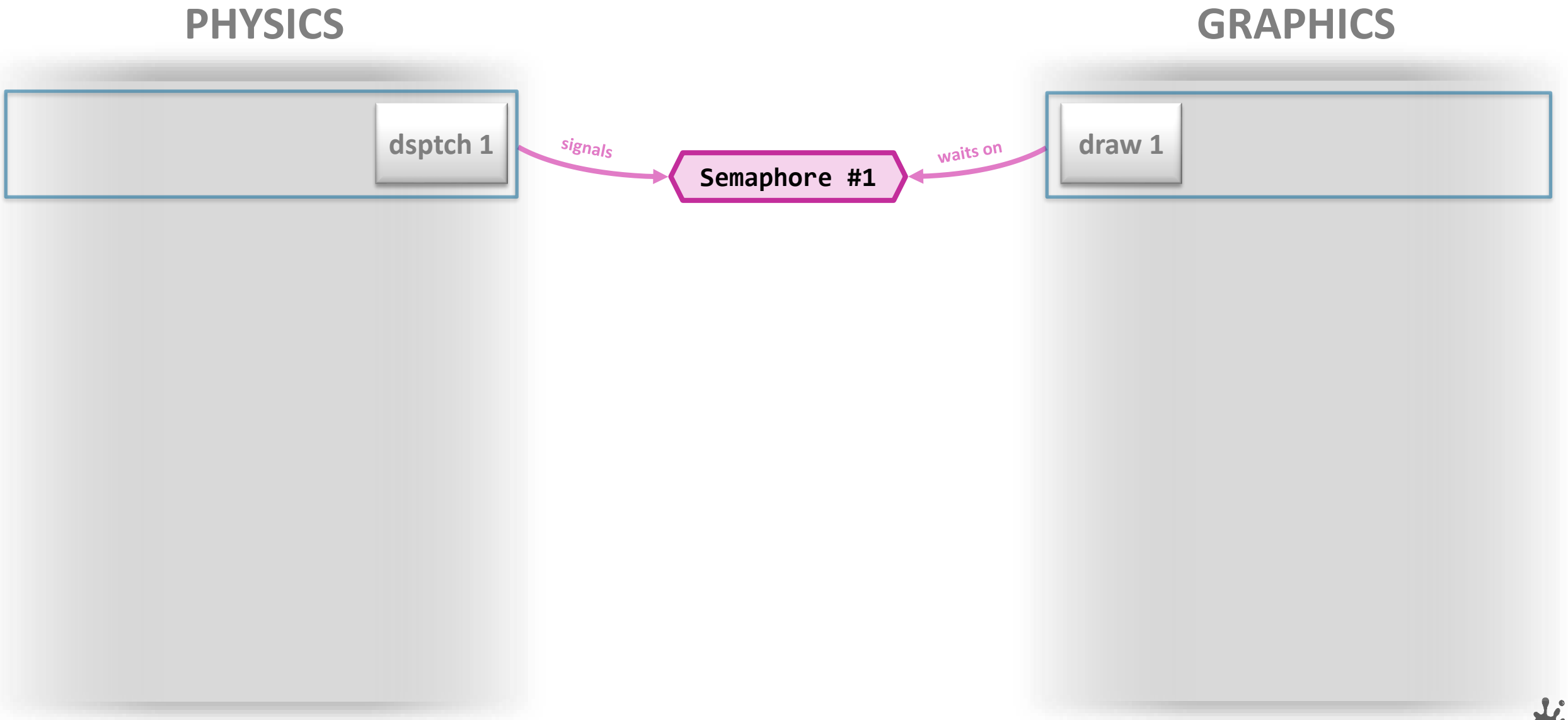
- Host query - A host operation that allows querying the payload of the timeline semaphore.
- Host wait - A host operation that allows a blocking wait for a timeline semaphore to reach a specified value.
- Host signal - A host operation that allows advancing the timeline semaphore to a specified value.
- Device wait - A device operation that allows waiting for a timeline semaphore to reach a specified value.
- Device signal - A device operation that allows advancing the timeline semaphore to a specified value.

The Khronos Group. Vulkan 1.3.213 Specificaton

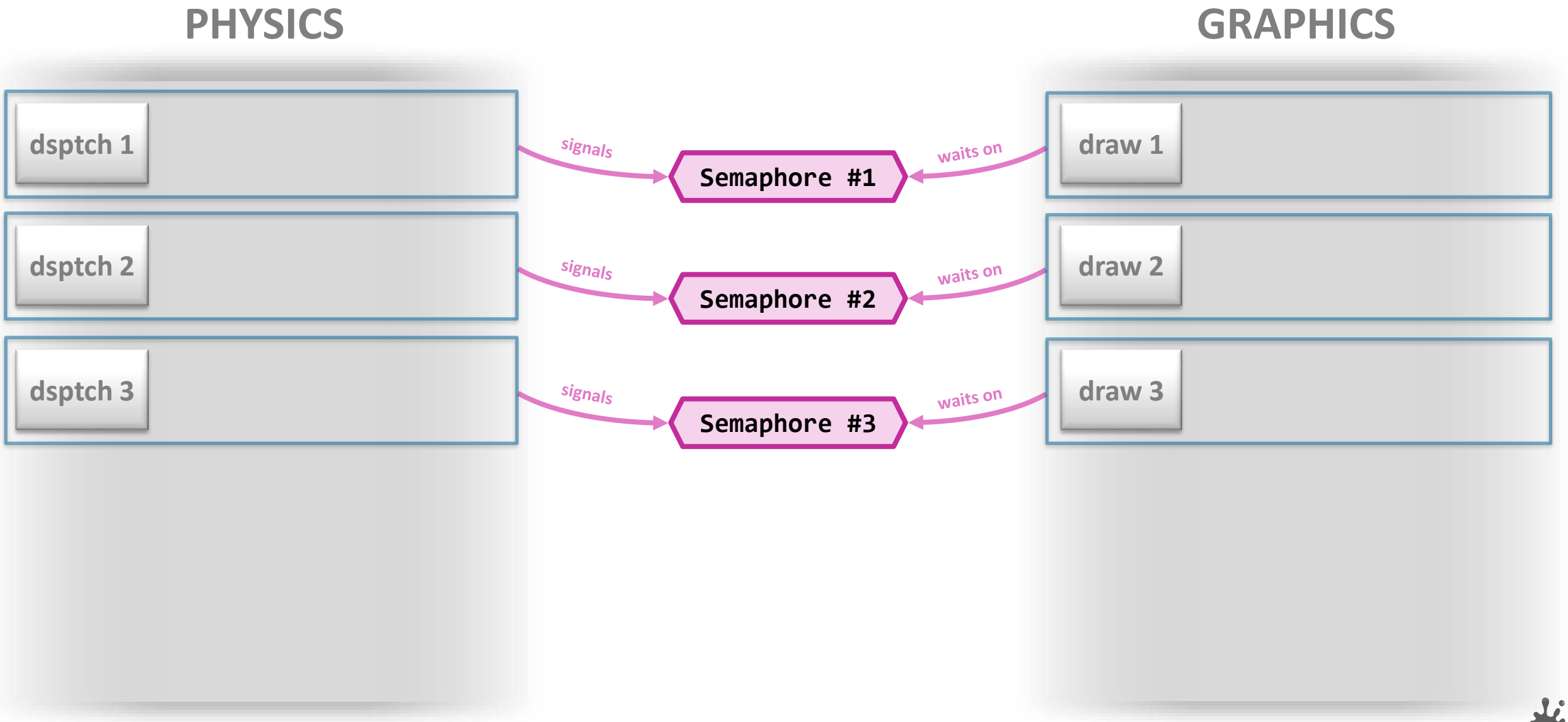
Troubles with Binary Semaphores



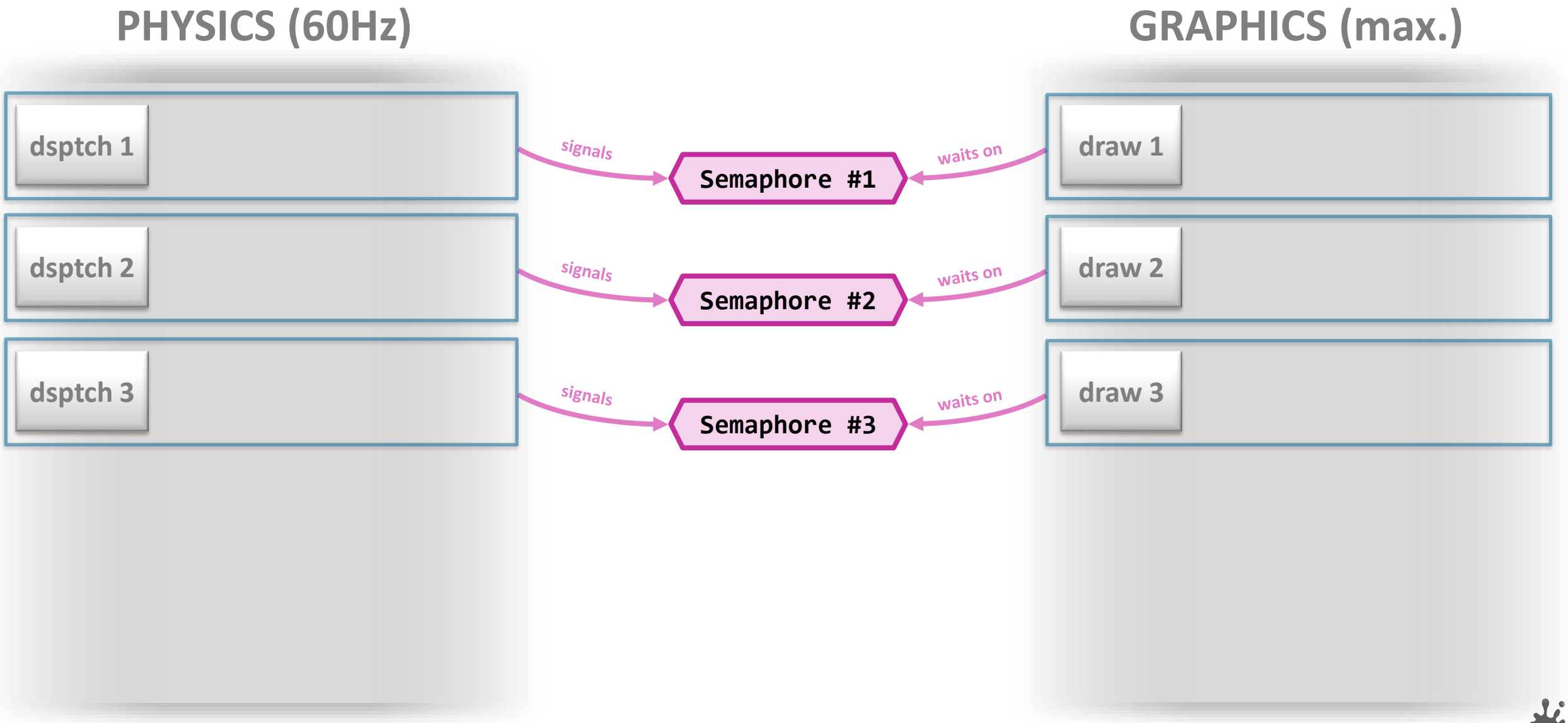
Troubles with Binary Semaphores



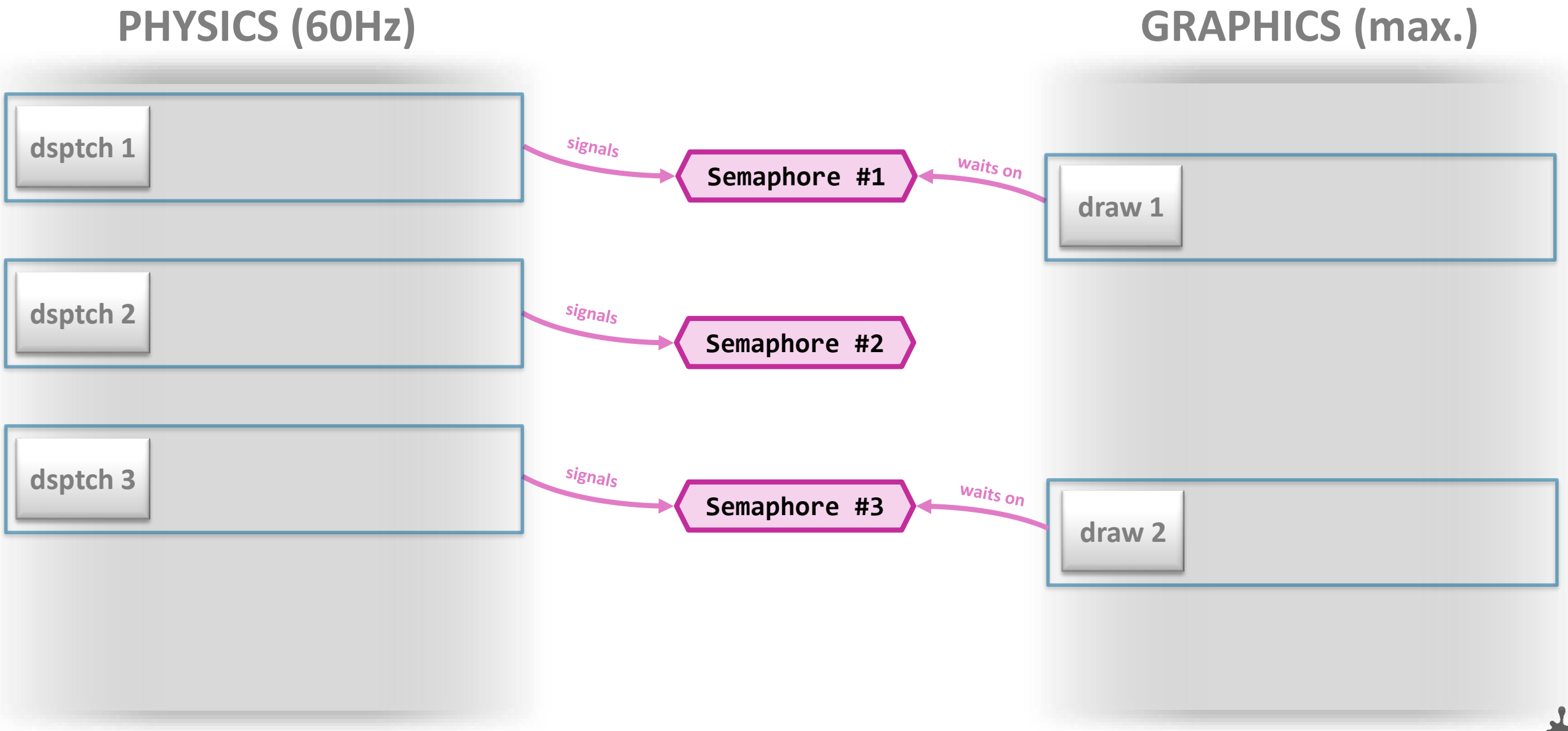
Troubles with Binary Semaphores



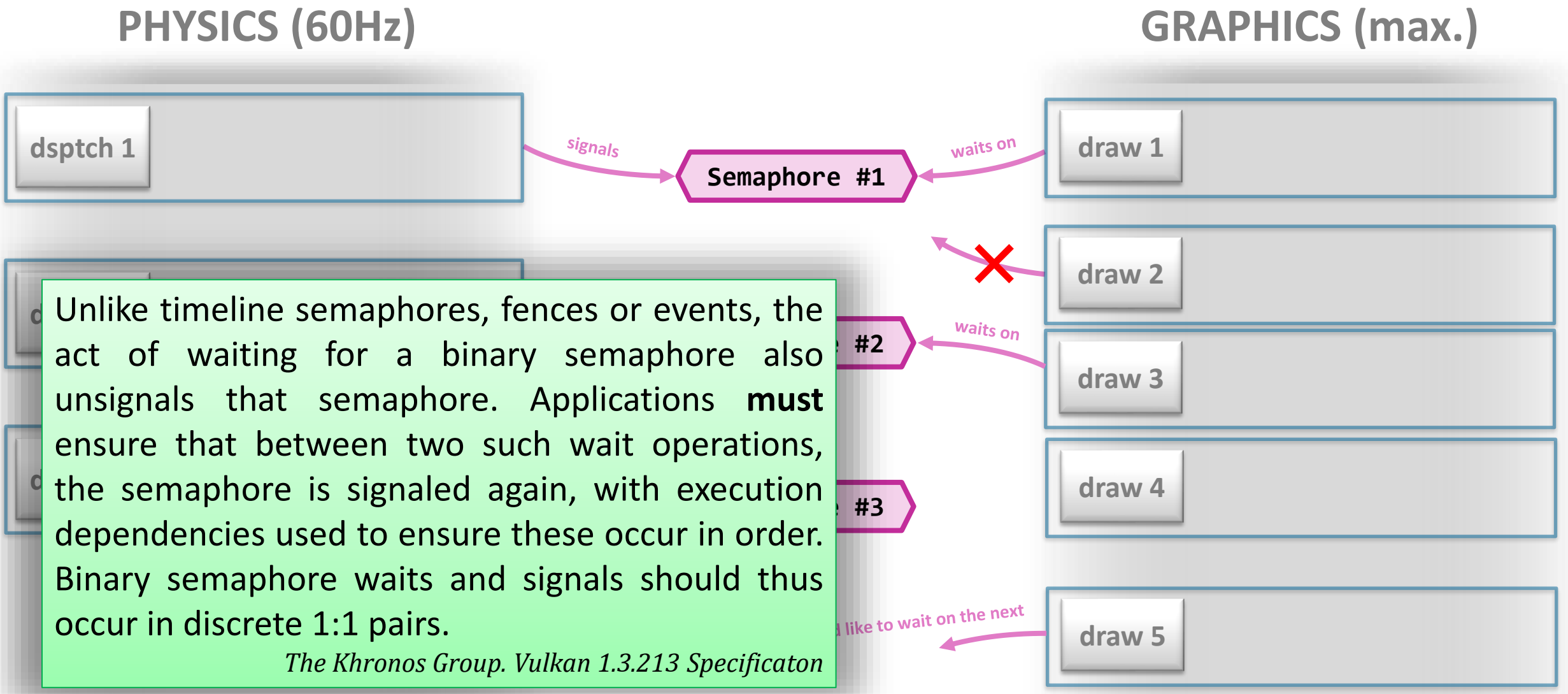
Troubles with Binary Semaphores



Troubles with Binary Semaphores



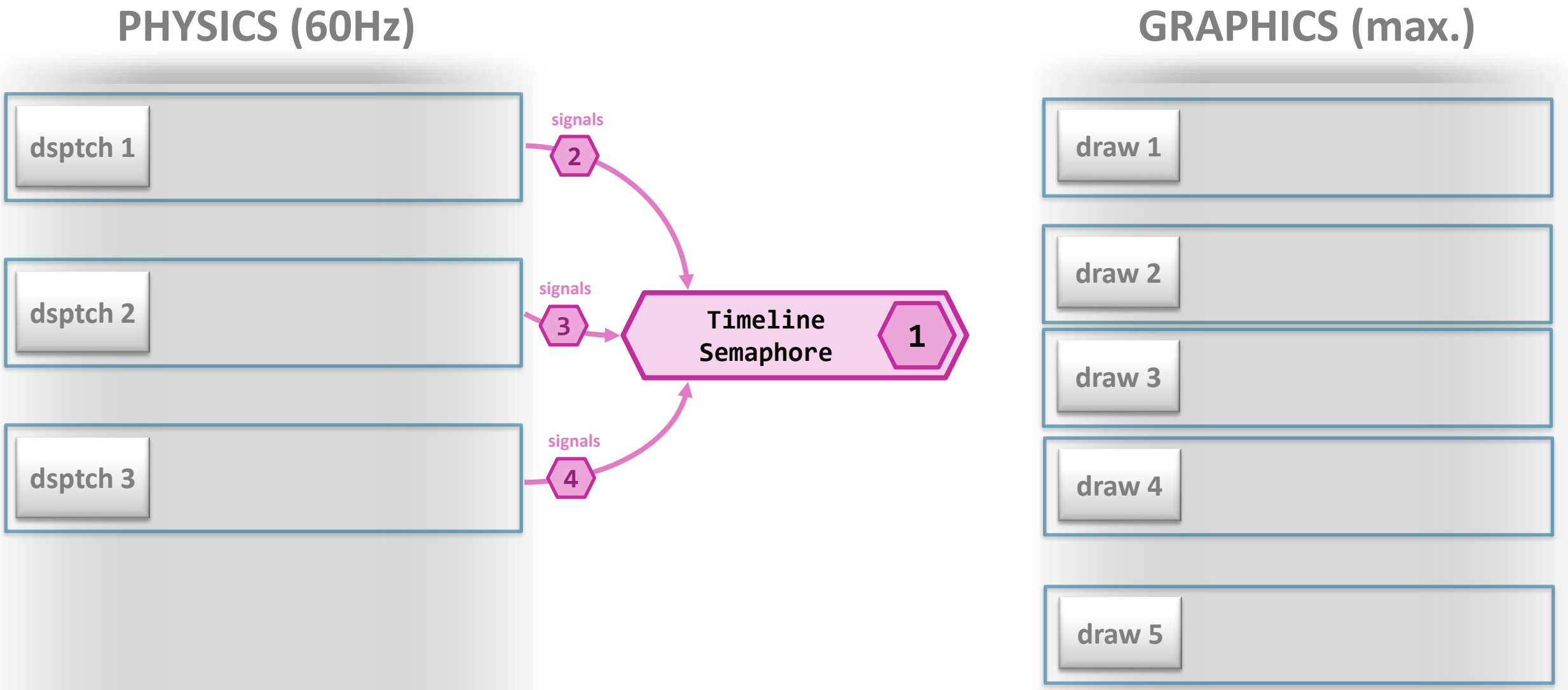
Troubles with Binary Semaphores



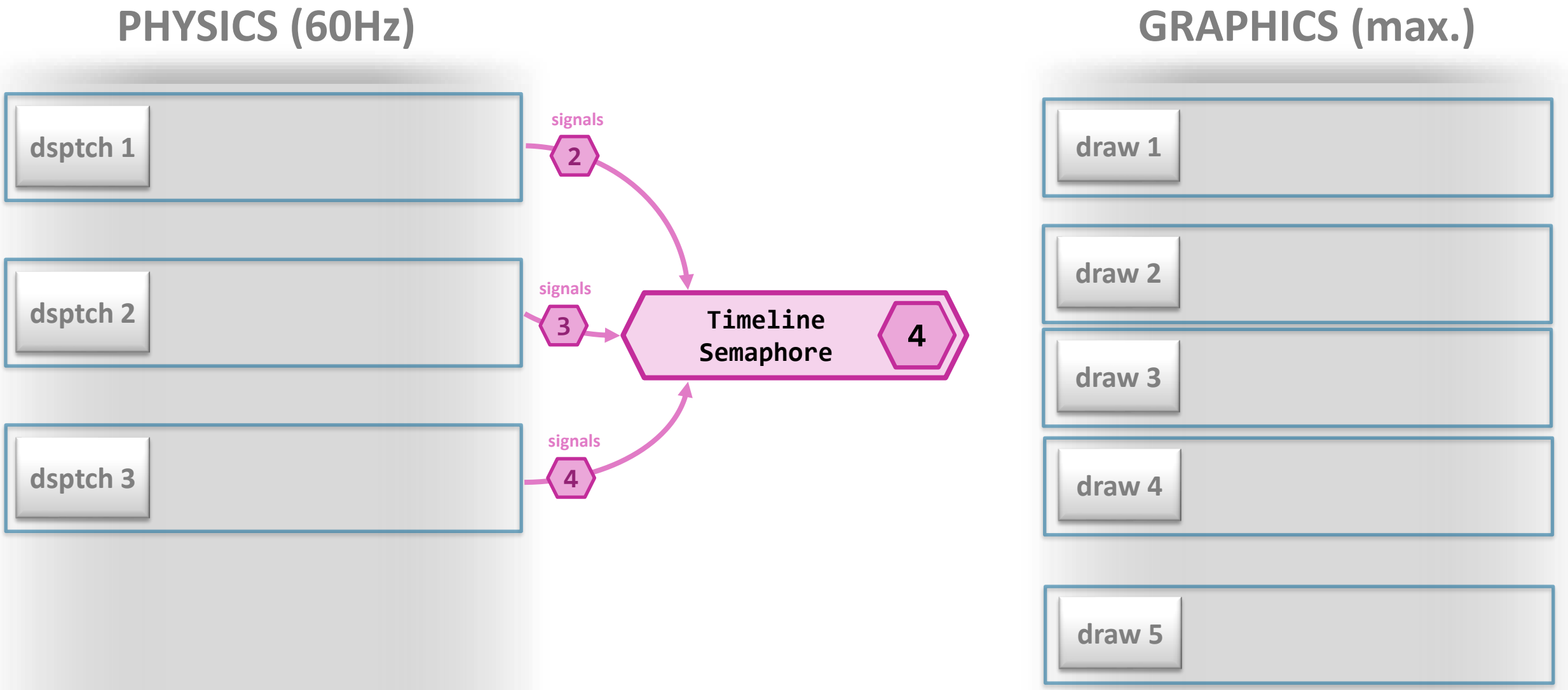
Unlike timeline semaphores, fences or events, the act of waiting for a binary semaphore also unsignals that semaphore. Applications **must** ensure that between two such wait operations, the semaphore is signaled again, with execution dependencies used to ensure these occur in order. Binary semaphore waits and signals should thus occur in discrete 1:1 pairs.

The Khronos Group. Vulkan 1.3.213 Specificaton

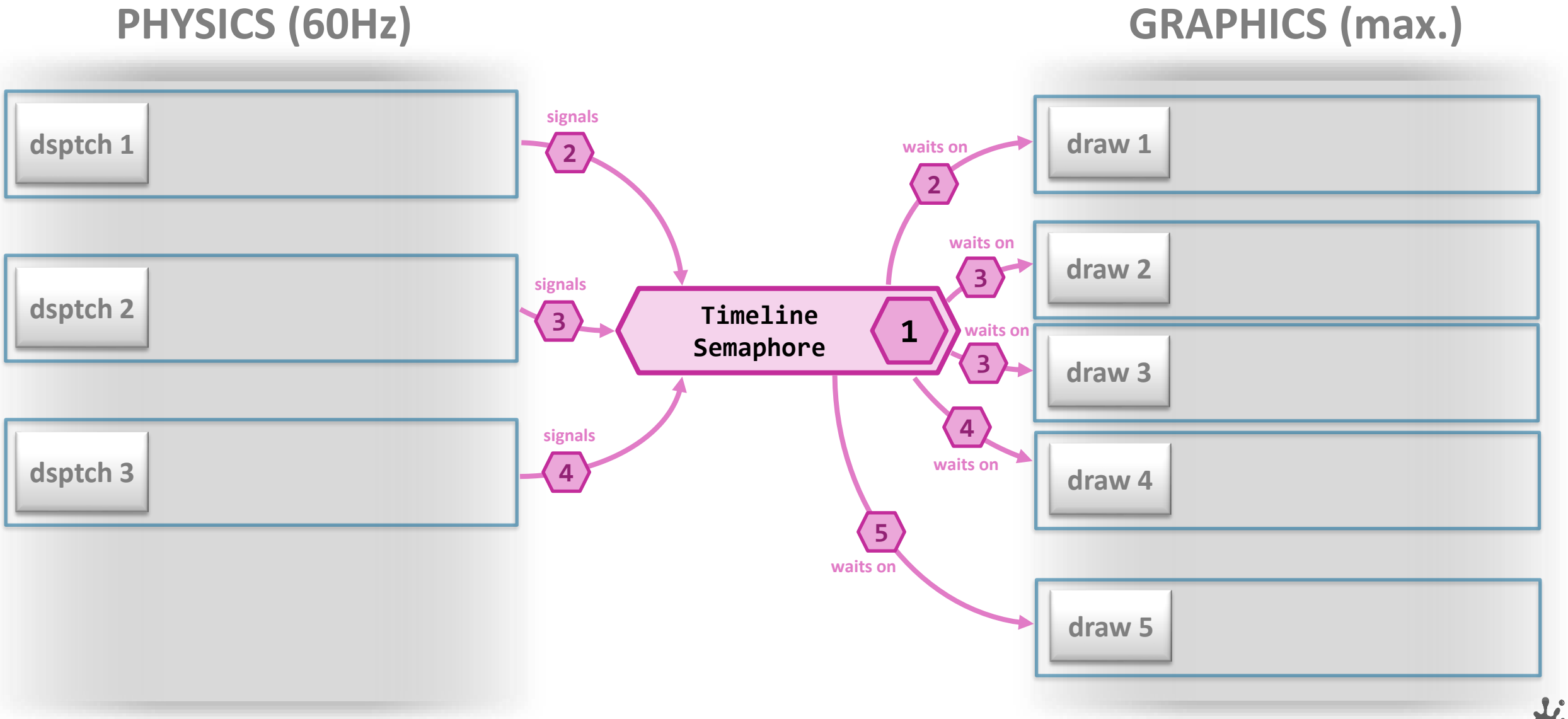
Timeline Semaphores to the Rescue



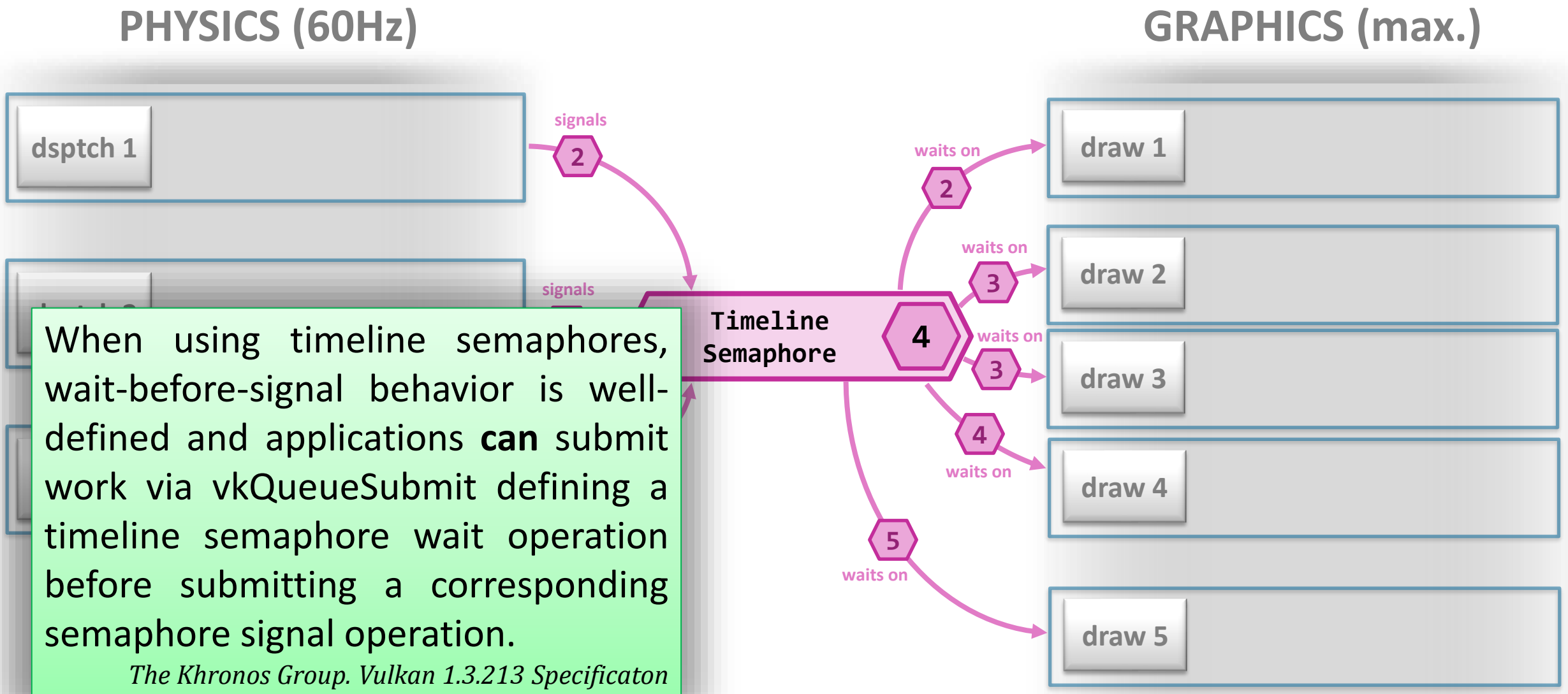
Timeline Semaphores to the Rescue



Timeline Semaphores to the Rescue



Timeline Semaphores to the Rescue



When using timeline semaphores, wait-before-signal behavior is well-defined and applications **can** submit work via vkQueueSubmit defining a timeline semaphore wait operation before submitting a corresponding semaphore signal operation.

The Khronos Group. Vulkan 1.3.213 Specificaton

Timeline Semaphores

- Same type as binary semaphores: `VkSemaphore`
  - But created with `VK_SEMAPHORE_TYPE_TIMELINE`
  - (in contrast to `VK_SEMAPHORE_TYPE_BINARY` for binary semaphores)
- Payload = strictly increasing 64-bit unsigned integer
- Host query: `vkGetSemaphoreCounterValue`
- Host wait: `vkWaitSemaphores`
- Host signal: `vkSignalSemaphore`

Overview of Synchronization Methods

- Wait Idle Operations (device -> host sync)
- Fences (device -> host sync)
- Semaphores (queue sync)
  - Binary Semaphores (queue -> queue sync, device only)
  - Timeline Semaphores
- Pipeline Barriers
  - Execution Barriers
  - Memory Barriers
- Render Pass Subpass Dependencies
- Events

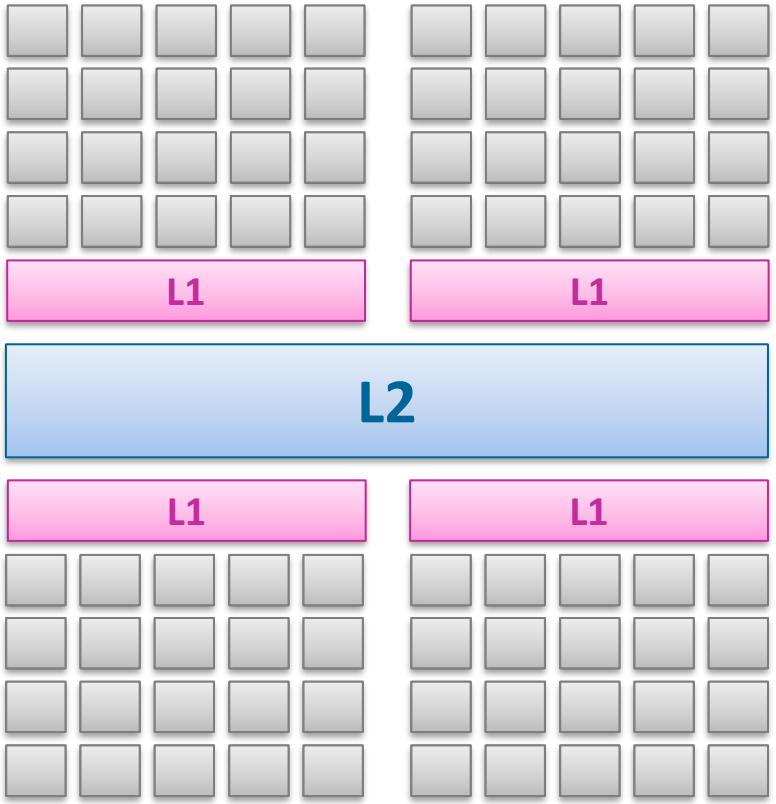
Overview of Synchronization Methods

- Wait Idle Operations (device -> host sync)
- Fences (device -> host sync)
- Semaphores (queue sync)
  - Binary Semaphores (queue -> queue sync, device only)
  - Timeline Semaphores (queue -> queue sync, host <-> device sync)
- Pipeline Barriers
  - Execution Barriers
  - Memory Barriers
- Render Pass Subpass Dependencies
- Events

Overview of Synchronization Methods

- Wait Idle Operations (device -> host sync)
- Fences (device -> host sync)
- Semaphores (queue sync)
  - Binary Semaphores (queue -> queue sync, device only)
  - Timeline Semaphores (queue -> queue sync, host <-> device sync)
- Pipeline Barriers
  - Execution Barriers
  - Memory Barriers
- Render Pass Subpass Dependencies
- Events

Pipeline **Execution** Barriers



QUEUE

CMD 1

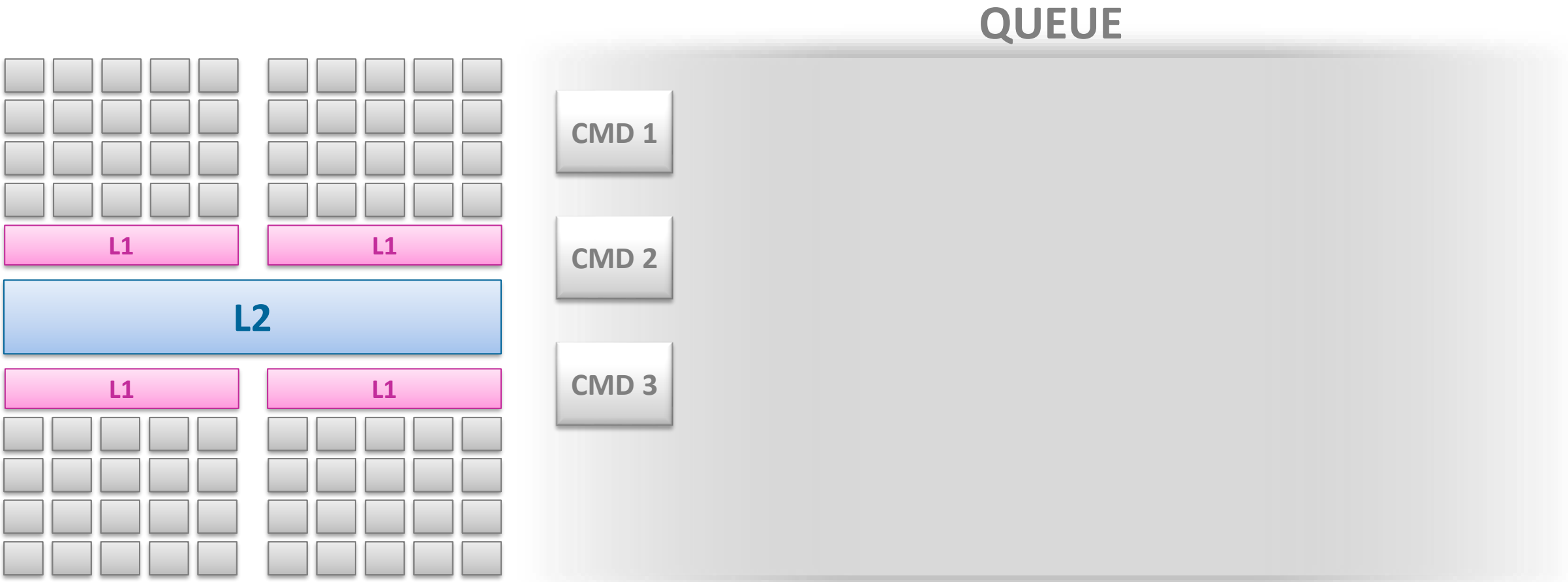
CMD 2

CMD 3

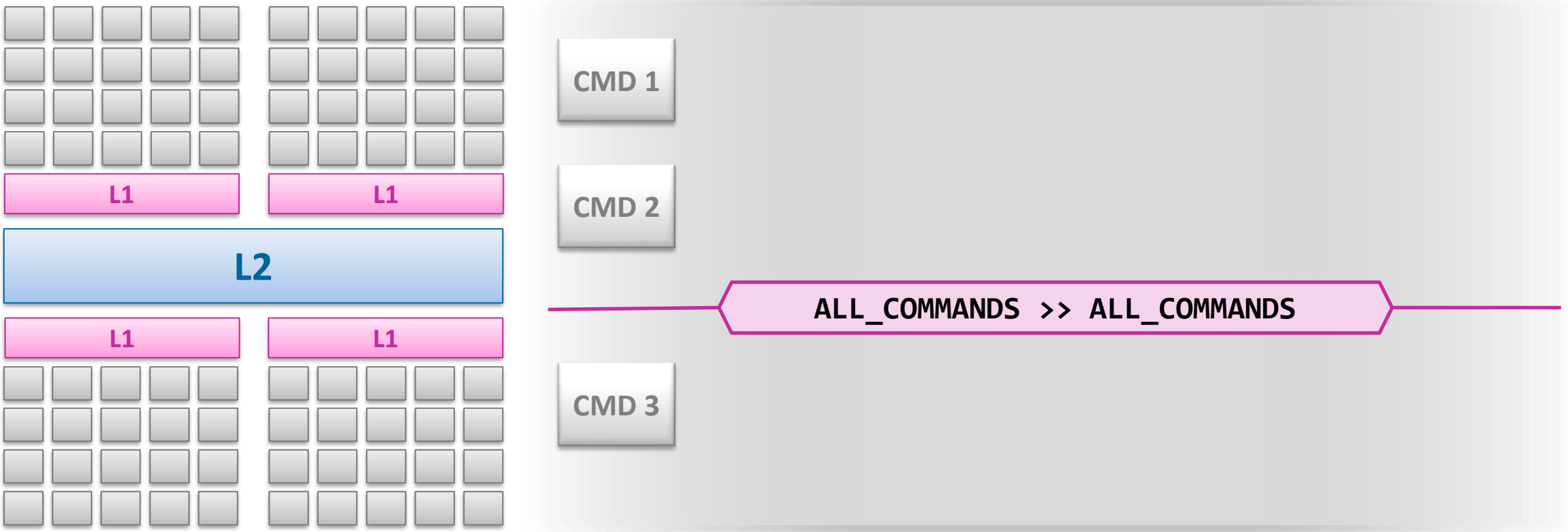
Command buffer submissions to a single queue respect submission order and other implicit ordering guarantees, but otherwise **may** overlap or execute out of order.

The Khronos Group. Vulkan 1.3.213 Specificaton

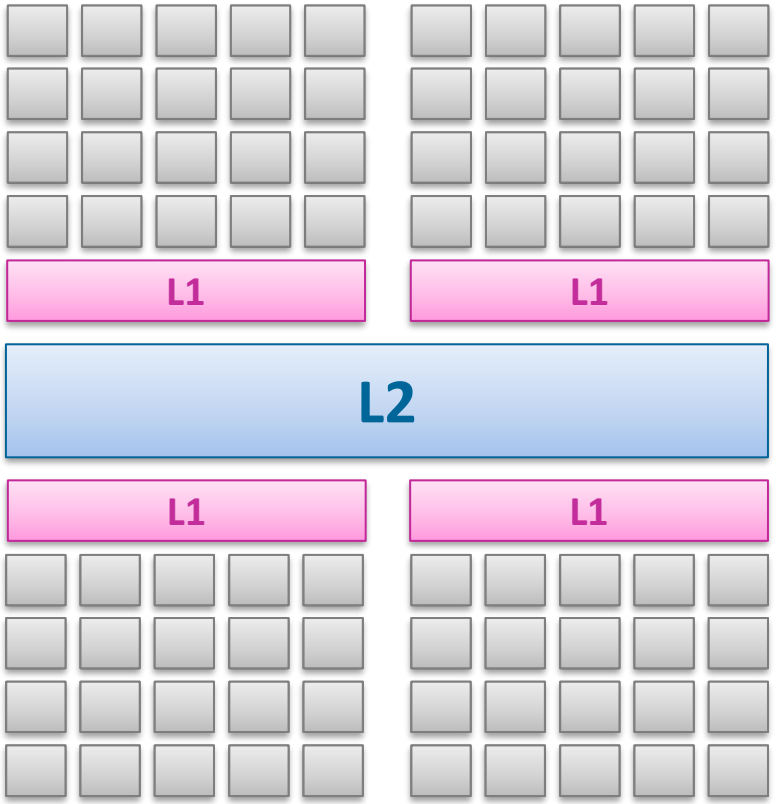
Pipeline **Execution** Barriers



Pipeline **Execution** Barriers



Pipeline **Execution** Barriers



QUEUE

CMD 1

CMD 2

CMD 3

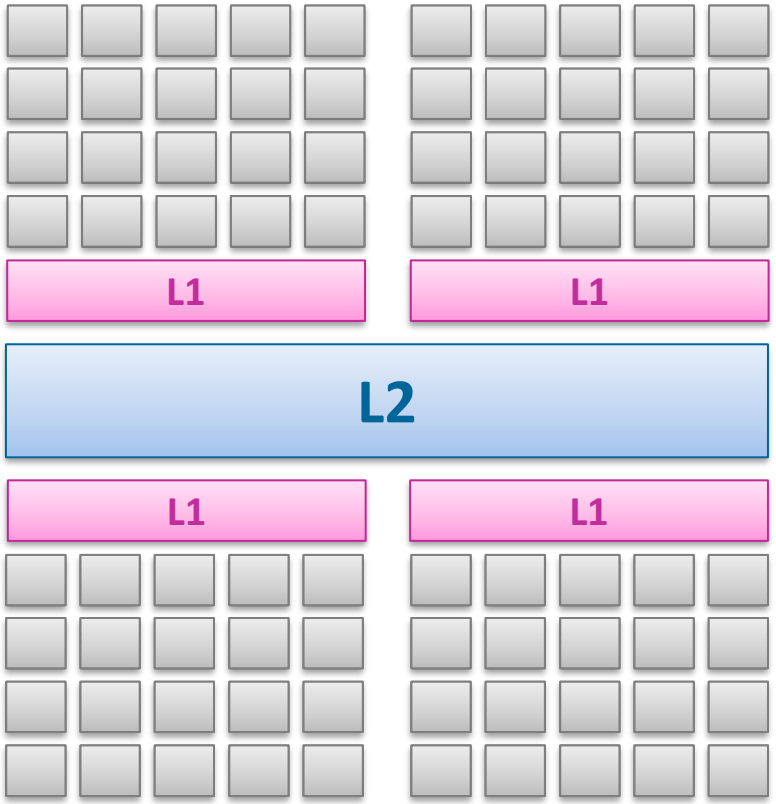
```
srcStageMask =
  VK_PIPELINE_STAGE_2_ALL_COMMANDS_BIT;
dstStageMask =
  VK_PIPELINE_STAGE_2_ALL_COMMANDS_BIT;
```

ALL_COMMANDS >> ALL_COMMANDS

When `vkCmdPipelineBarrier2` is submitted to a queue, it defines memory dependencies between commands that were submitted before it, and those submitted after it.

The Khronos Group. Vulkan 1.3.207 Specificaton

Pipeline **Execution** Barriers



QUEUE

CMD 1
GRAPHICS

COLOR_ATTACHMENT_OUTPUT

CMD 2
GRAPHICS

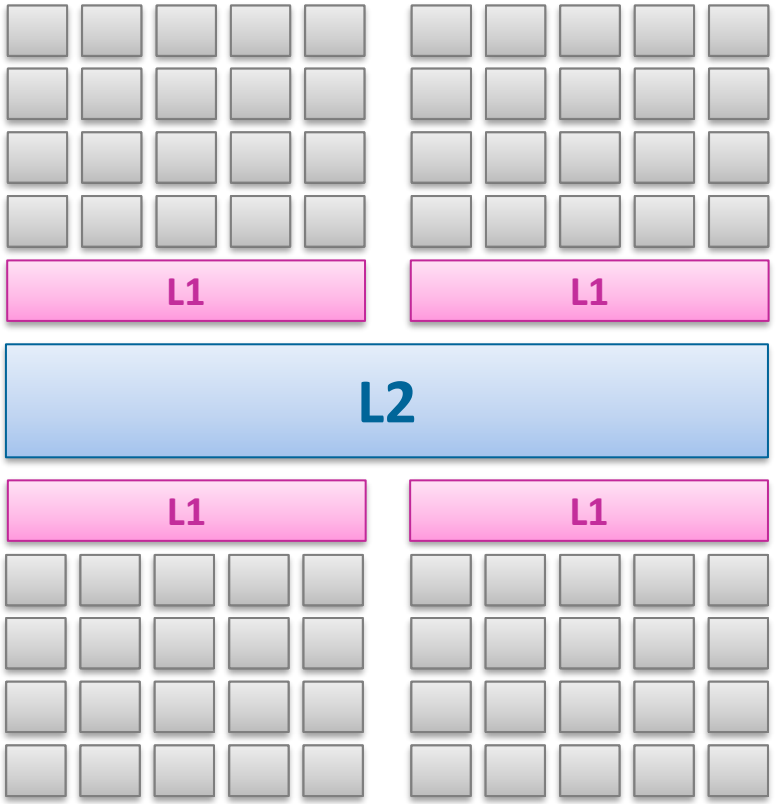
COLOR_ATTACHMENT_OUTPUT

COLOR_ATTACHMENT_OUTPUT >> FRAGMENT_SHADER

CMD 3
GRAPHICS

FRAGMENT_SHADER

Pipeline **Execution** Barriers



QUEUE

CMD 1
GRAPHICS

COLOR_ATTACHMENT_OUTPUT

CMD 2
GRAPHICS

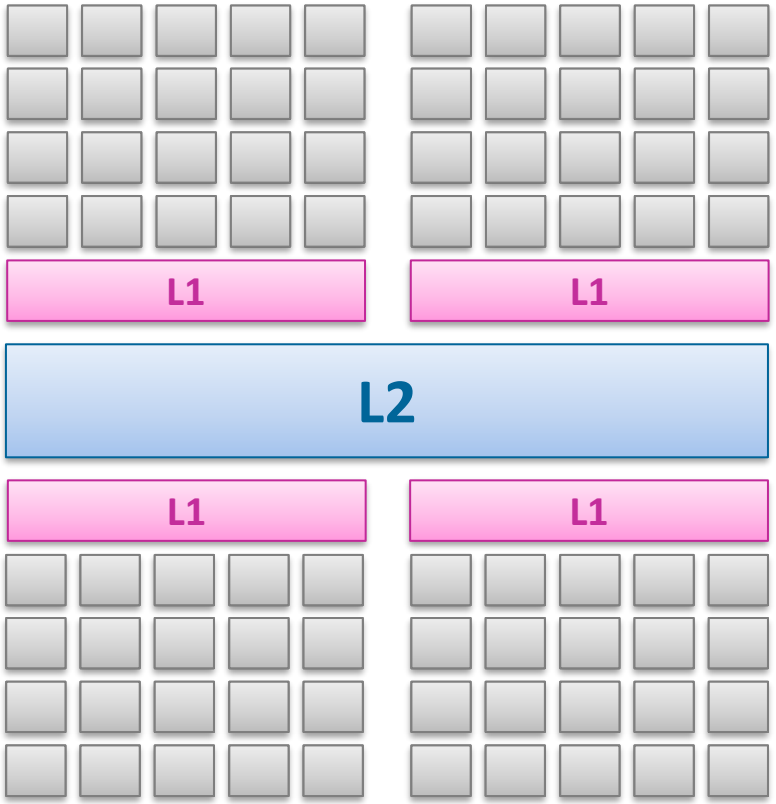
EARLY_FRAGMENT_TESTS

```
COLOR_ATTACHMENT_OUTPUT | EARLY_FRAGMENT_TESTS
>> FRAGMENT_SHADER
```

CMD 3
GRAPHICS

FRAGMENT_SHADER

Pipeline **Execution** Barriers



CMD 1
GRAPHICS

CMD 2
GRAPHICS

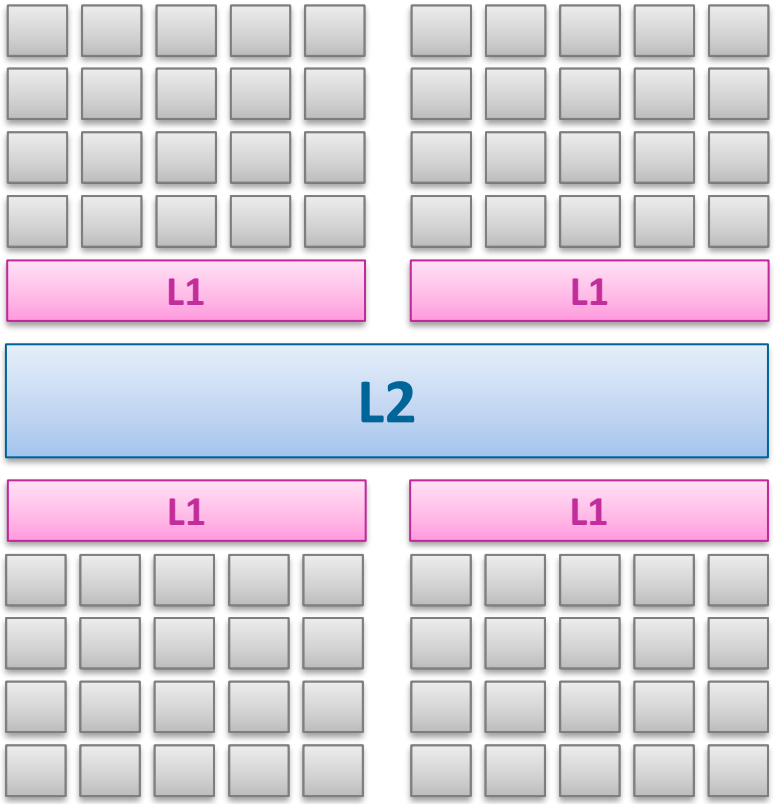
CMD 3
GRAPHICS

Including a particular pipeline stage in the first synchronization scope of a command implicitly includes logically earlier pipeline stages in the synchronization scope. Similarly, the second synchronization scope includes logically later pipeline stages.

The Khronos Group. Vulkan 1.3.207 Specificaton

```
COLOR_ATTACHMENT_OUTPUT | EARLY_FRAGMENT_TESTS
>> FRAGMENT_SHADER
```

Pipeline **Execution** Barriers



CMD 1
GRAPHICS

CMD 2
GRAPHICS

CMD 3
GRAPHICS

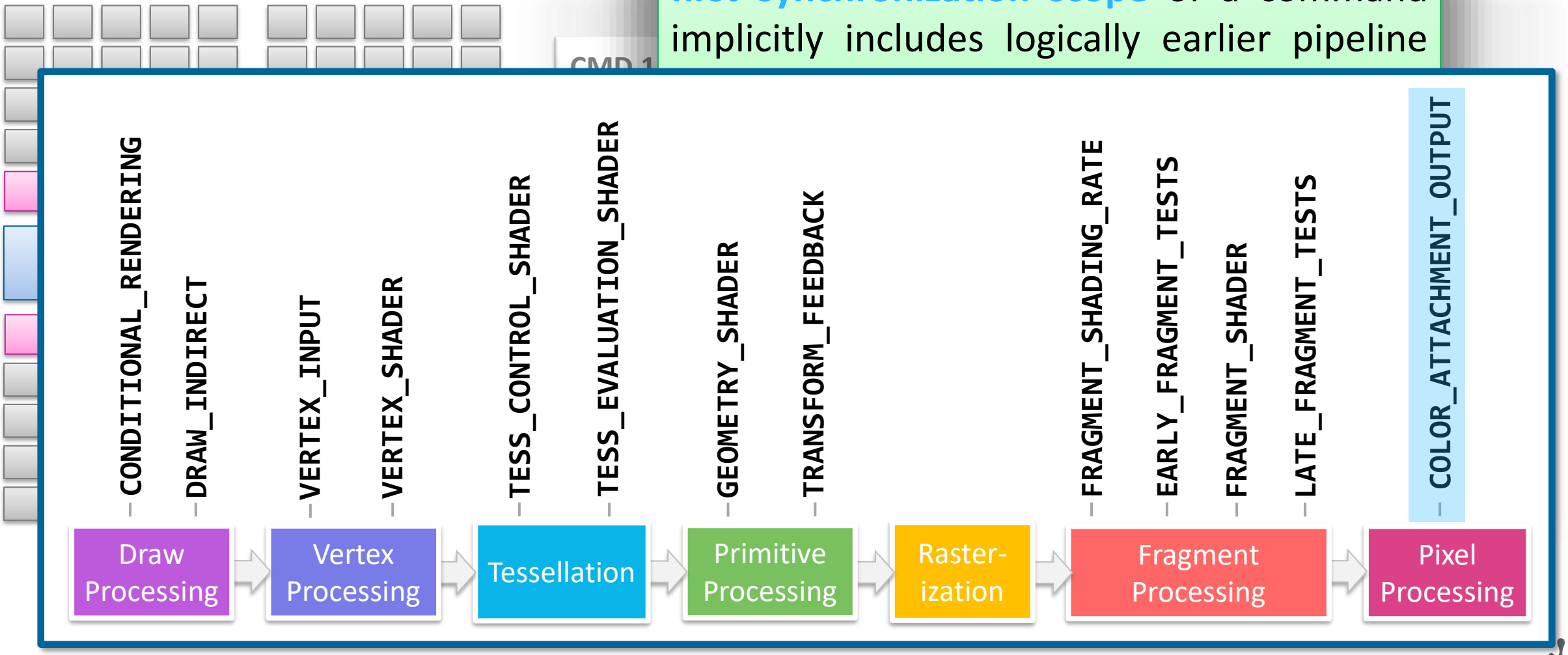
Including a particular pipeline stage in the **first synchronization scope** of a command implicitly includes logically earlier pipeline stages in the synchronization scope. Similarly, the **second synchronization scope** includes logically later pipeline stages.

The Khronos Group. Vulkan 1.3.207 Specificaton

```
COLOR_ATTACHMENT_OUTPUT | EARLY_FRAGMENT_TESTS
>> FRAGMENT_SHADER
```

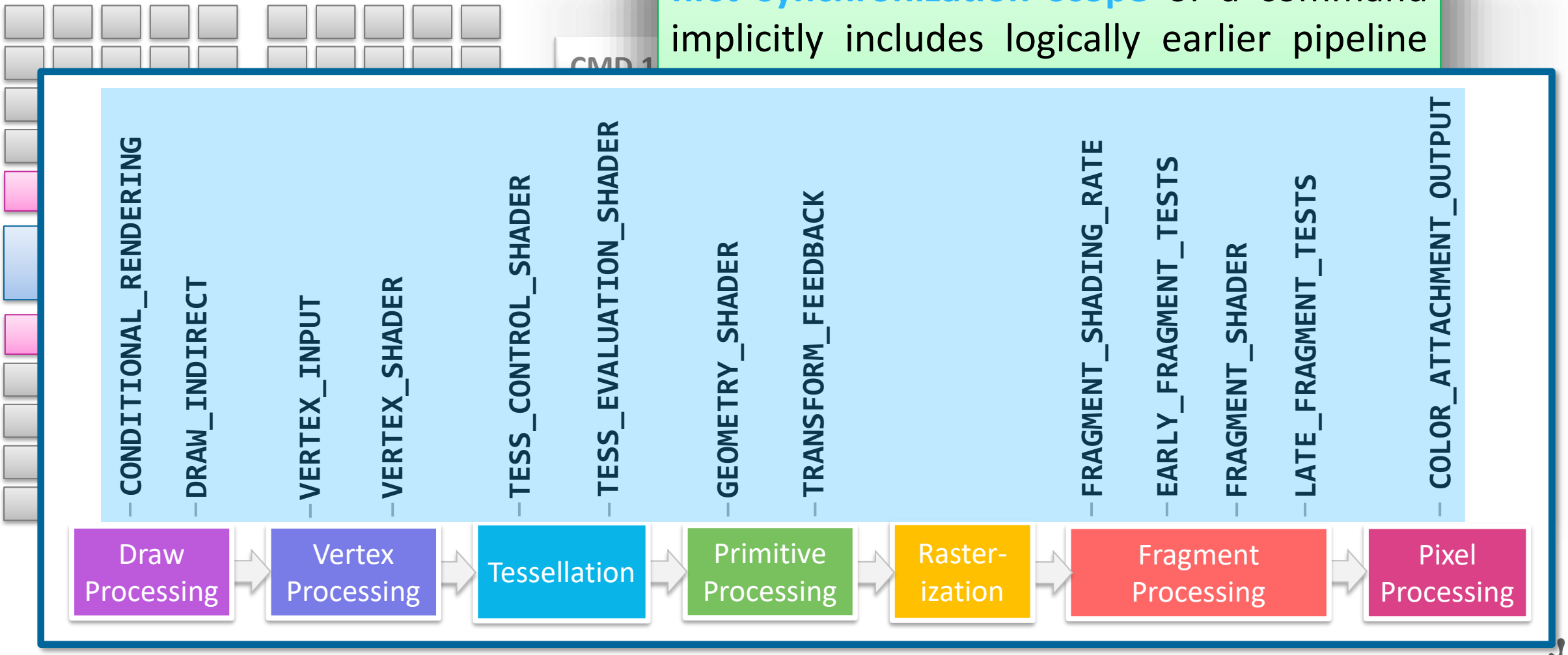
Pipeline **Execution** Barriers

Including a particular pipeline stage in the **first synchronization scope** of a command implicitly includes logically earlier pipeline

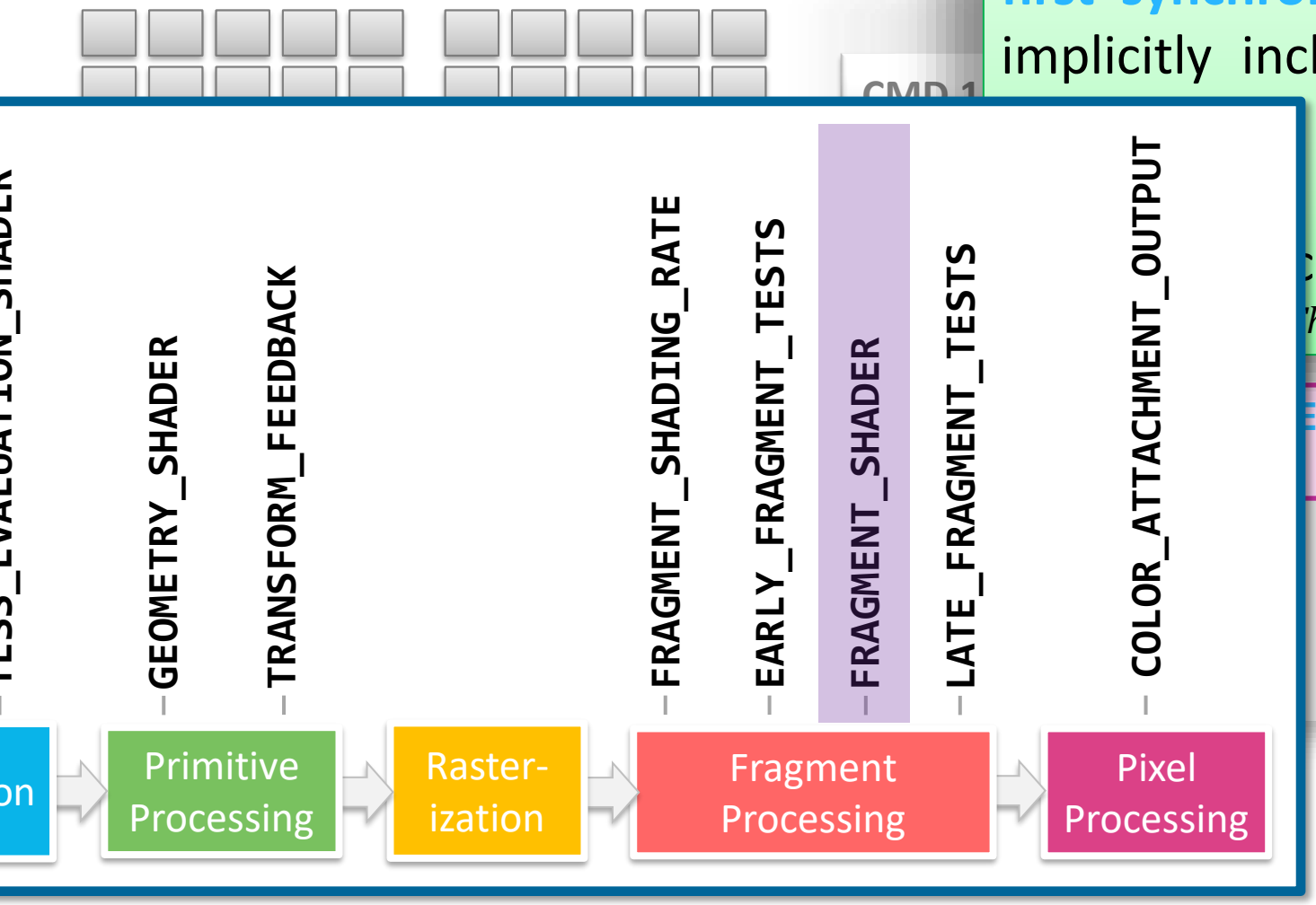


Pipeline **Execution** Barriers

Including a particular pipeline stage in the **first synchronization scope** of a command implicitly includes logically earlier pipeline



Pipeline **Execution** Barriers



Including a particular pipeline stage in the **first synchronization scope** of a command implicitly includes logically earlier pipeline
the synchronization scope.
second synchronization scope
cally later pipeline stages.

The Khronos Group. Vulkan 1.3.207 Specificaton

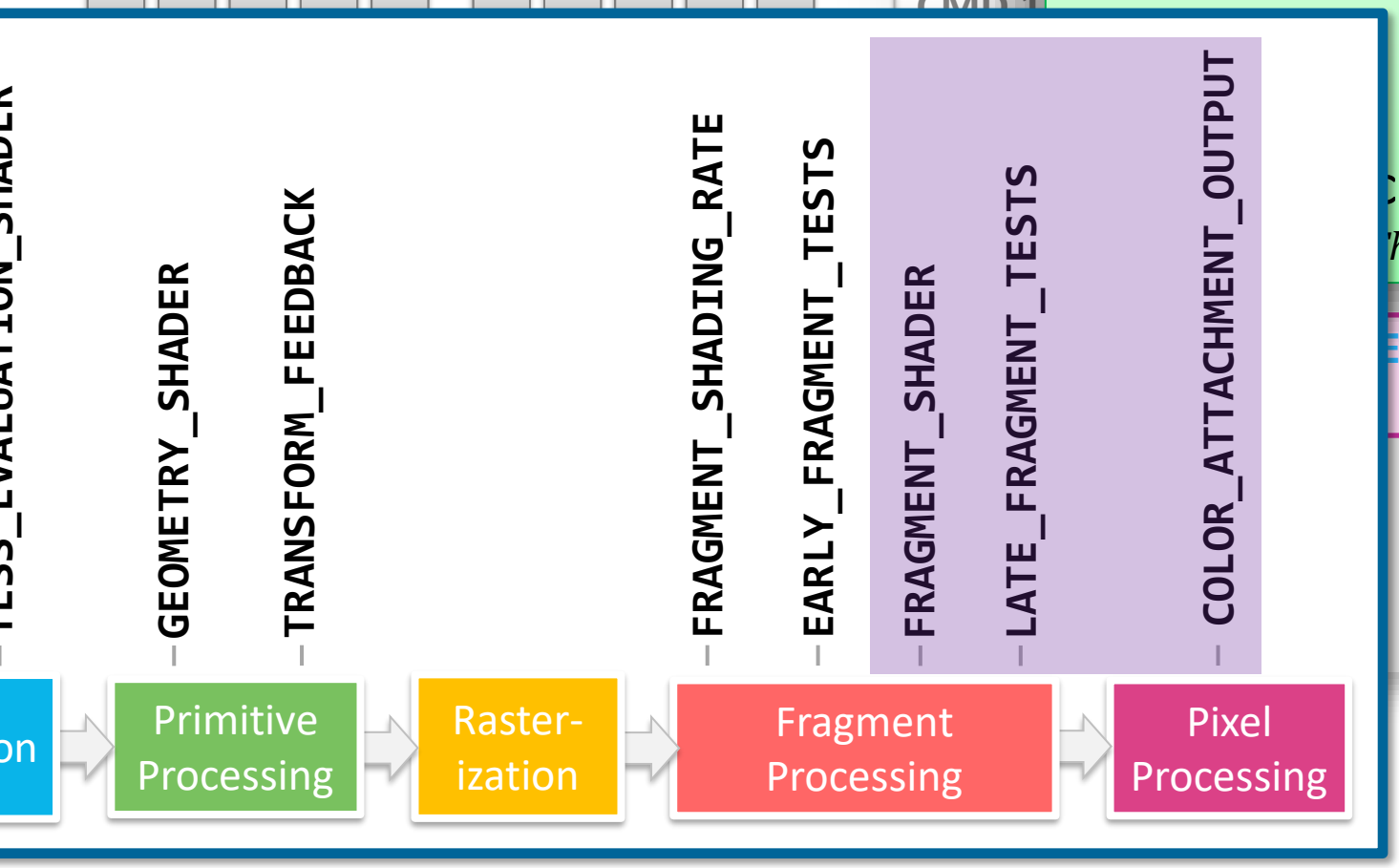
```
ENT_OUTPUT | EARLY_FRAGMENT_TESTS
  >> FRAGMENT_SHADER
```

Pipeline **Execution** Barriers

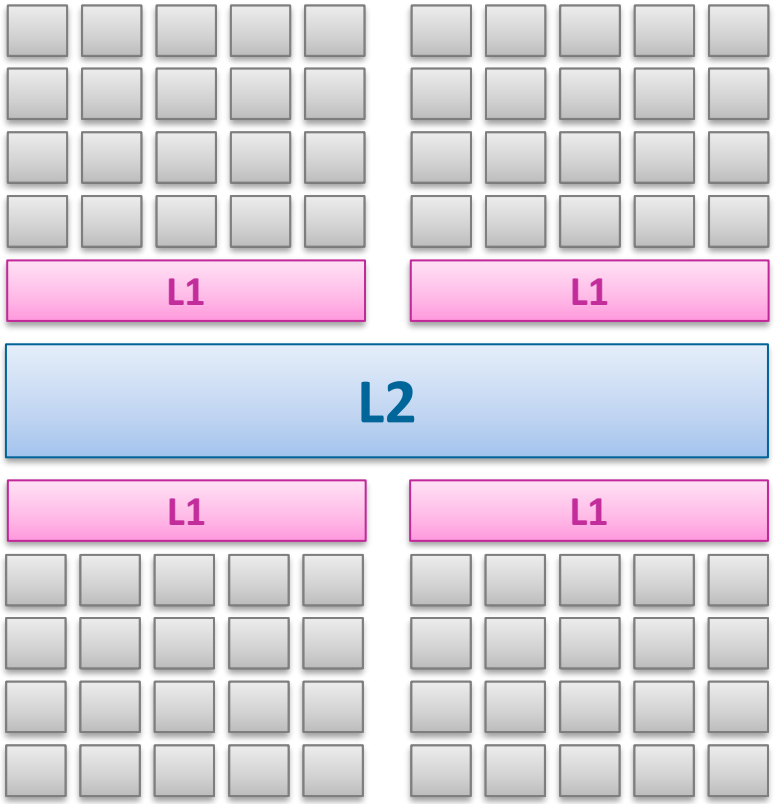
Including a particular pipeline stage in the **first synchronization scope** of a command implicitly includes logically earlier pipeline
the synchronization scope.
second synchronization scope
cally later pipeline stages.

The Khronos Group. Vulkan 1.3.207 Specificaton

```
ENT_OUTPUT | EARLY_FRAGMENT_TESTS
  >> FRAGMENT_SHADER
```



Pipeline **Execution** Barriers



CMD 1
GRAPHICS

CMD 2
GRAPHICS

CMD 3
GRAPHICS

Including a particular pipeline stage in the **first synchronization scope** of a command implicitly includes logically earlier pipeline stages in the synchronization scope. Similarly, the **second synchronization scope** includes logically later pipeline stages.

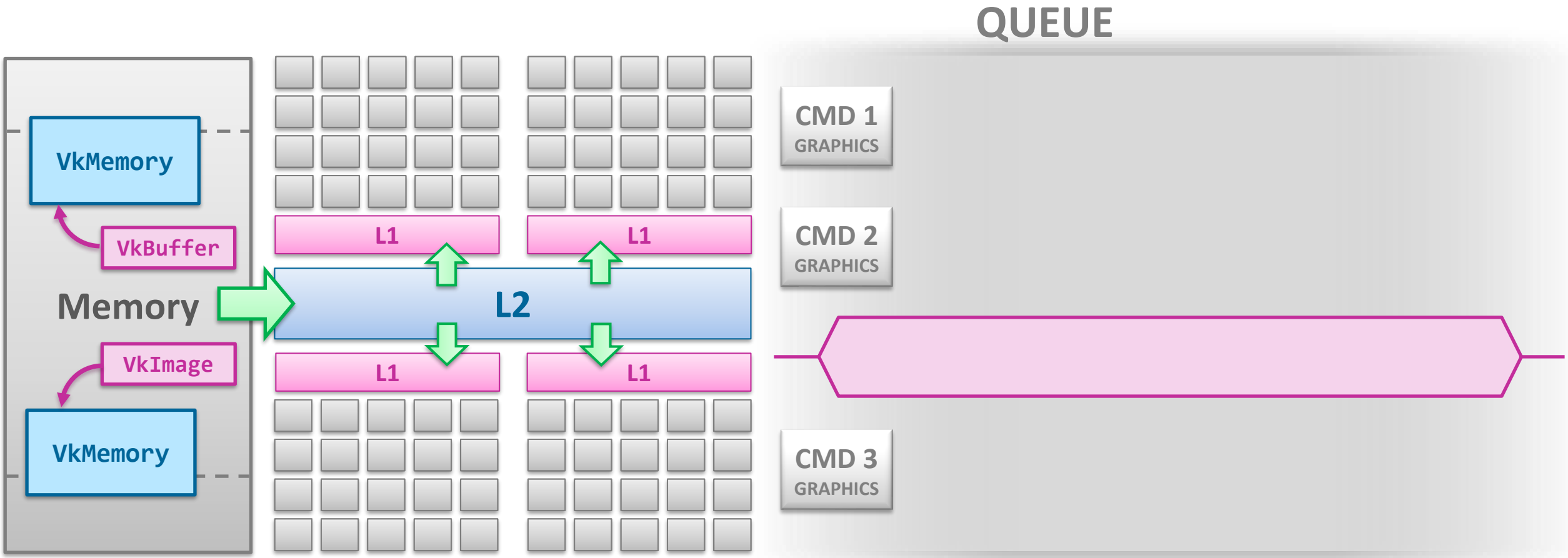
The Khronos Group. Vulkan 1.3.207 Specificaton

`COLOR_ATTACHMENT_OUTPUT | EARLY_FRAGMENT_TESTS >> FRAGMENT_SHADER`

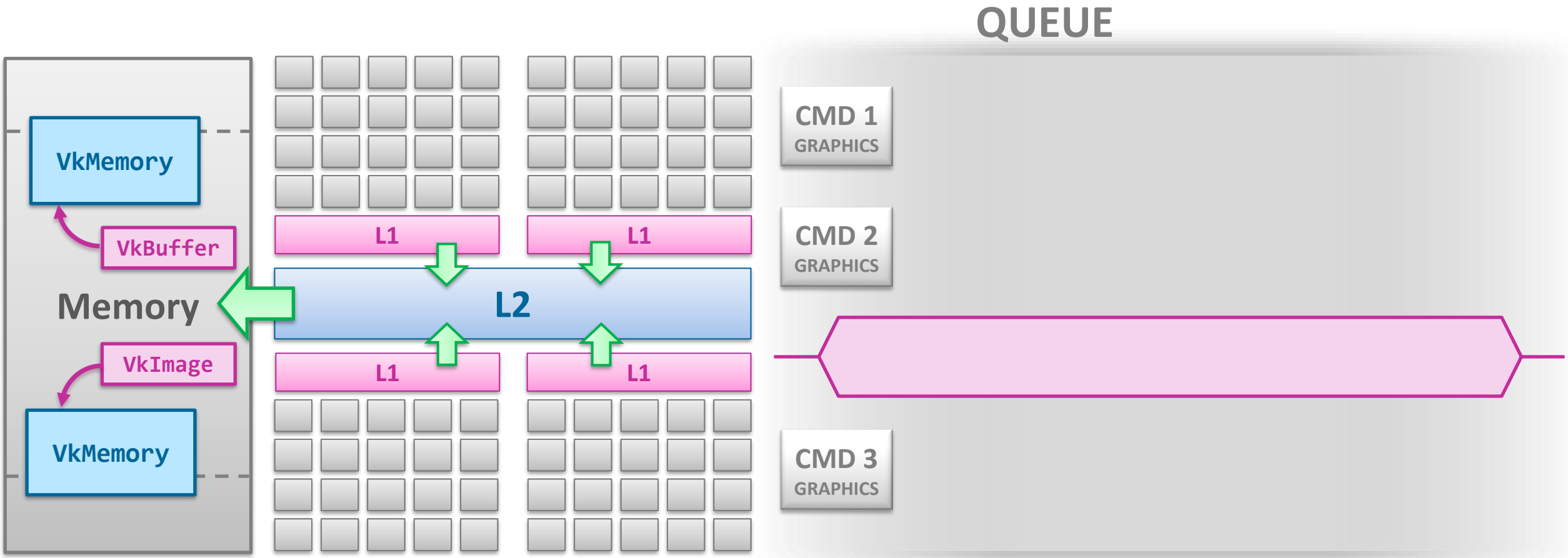
However, note that access scopes are not affected in this way - only the **precise stages specified** are considered **part of each access scope**.

The Khronos Group. Vulkan 1.3.207 Specificaton

Pipeline **Execution** Barriers



Pipeline **Execution** Barriers



Overview of Synchronization Methods

- Wait Idle Operations (device -> host sync)
- Fences (device -> host sync)
- Semaphores (queue sync)
  - Binary Semaphores (queue -> queue sync, device only)
  - Timeline Semaphores (queue -> queue sync, host <-> device sync)
- Pipeline Barriers
  - Execution Barriers
  - Memory Barriers
- Render Pass Subpass Dependencies
- Events

Overview of Synchronization Methods

- Wait Idle Operations (device -> host sync)
- Fences (device -> host sync)
- Semaphores (queue sync)
  - Binary Semaphores (queue -> queue sync, device only)
  - Timeline Semaphores (queue -> queue sync, host <-> device sync)
- Pipeline Barriers (command -> command sync, intra-queue)
  - Execution Barriers
  - Memory Barriers
- Render Pass Subpass Dependencies
- Events

Overview of Synchronization Methods

- Wait Idle Operations (device -> host sync)
- Fences (device -> host sync)
- Semaphores (queue sync)
  - Binary Semaphores (queue -> queue sync, device only)
  - Timeline Semaphores (queue -> queue sync, host <-> device sync)
- Pipeline Barriers (command -> command sync, intra-queue)
  - Execution Barriers
  - Memory Barriers
- Render Pass Subpass Dependencies
- Events

Overview of Synchronization Methods

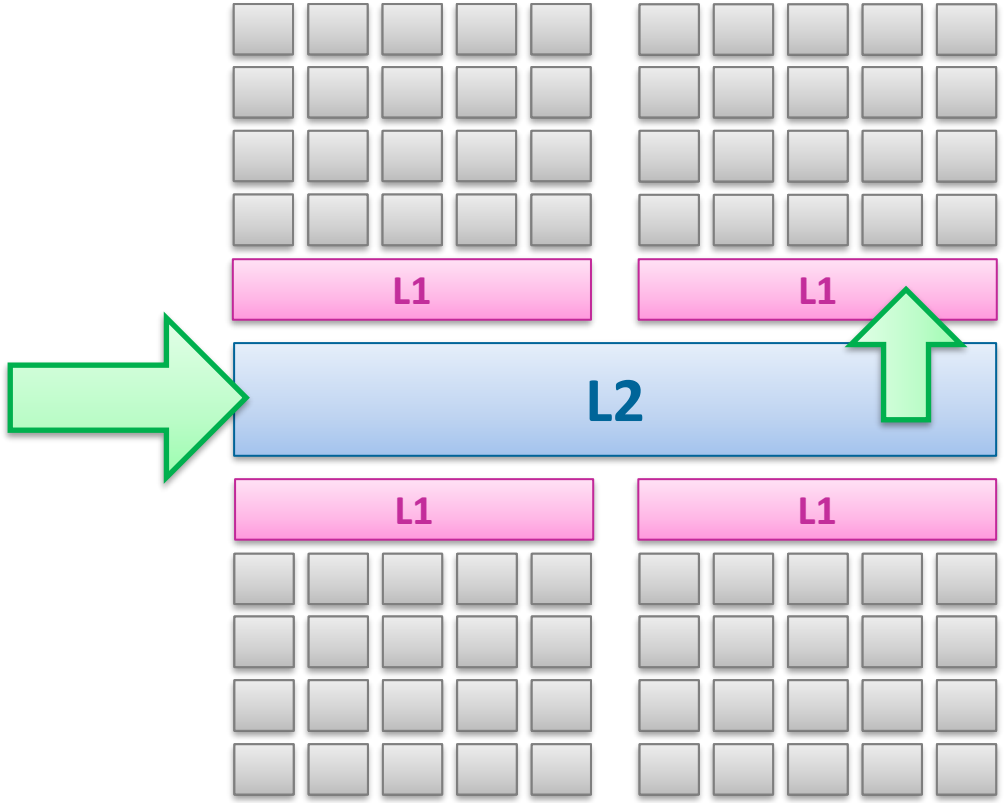
- Wait Idle Operations (device -> host sync)
- Fences (device -> host sync)
- Semaphores (queue sync)
  - Binary Semaphores (queue -> queue sync, device only)
  - Timeline Semaphores (queue -> queue sync, host <-> device sync)
- Pipeline Barriers (command -> command sync, intra-queue)
  - Execution Barriers (execution-only dependency, memory disregarded)
  - Memory Barriers
- Render Pass Subpass Dependencies
- Events

Overview of Synchronization Methods

- Wait Idle Operations (device -> host sync)
- Fences (device -> host sync)
- Semaphores (queue sync)
  - Binary Semaphores (queue -> queue sync, device only)
  - Timeline Semaphores (queue -> queue sync, host <-> device sync)
- Pipeline Barriers (command -> command sync, intra-queue)
  - Execution Barriers (execution-only dependency, memory disregarded)
  - Memory Barriers
- Render Pass Subpass Dependencies
- Events

Memory Availability and Visibility

e.g.:
Stage: VK_PIPELINE_STAGE_2_**FRAGMENT_SHADER**_BIT
Access Mask: VK_ACCESS_2_**SHADER_READ**_BIT



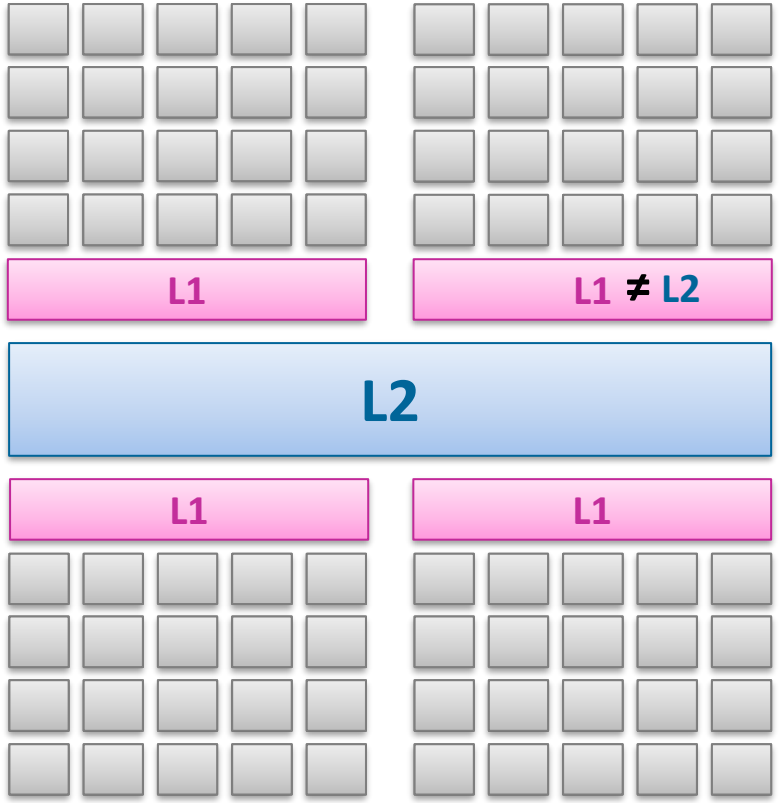
"**visibility**" always refers to a combination of **pipeline stage + access mask** which memory is *visible to*

Memory is "**visible**"

Memory has become "**available**"

Memory Availability and Visibility

e.g.:
Stage: VK_PIPELINE_STAGE_2_**COLOR_ATTACHMENT_OUTPUT**_BIT
Access Mask: VK_ACCESS_2_**COLOR_ATTACHMENT_WRITE**_BIT



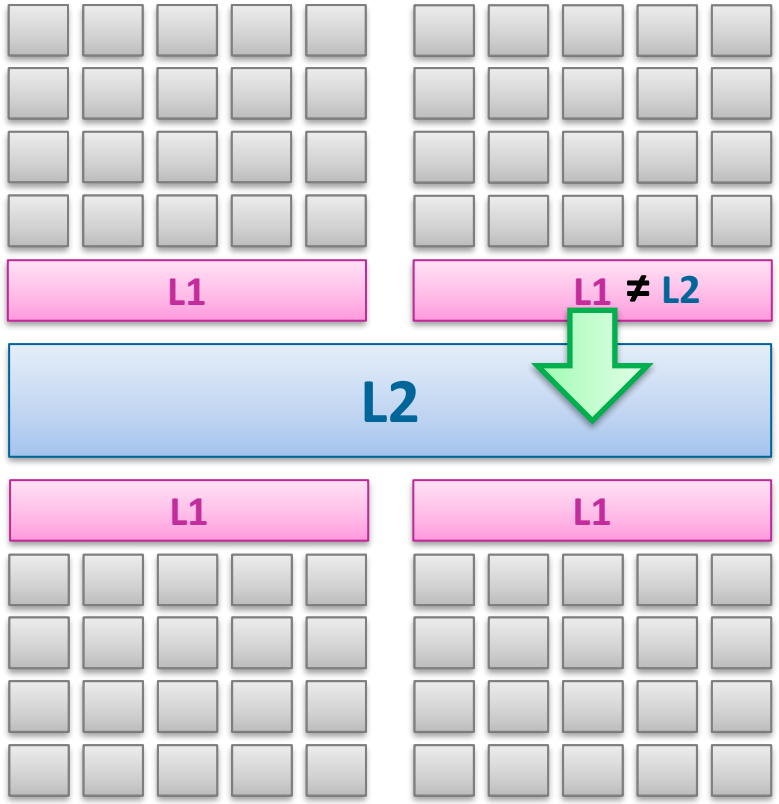
"**visibility**" always refers to a combination of **pipeline stage + access mask** which memory is *visible to*

Memory is "**visible**"

Memory is no longer **available**

Can be made available again using a **memory barrier**:

Memory Availability and Visibility



"**visibility**" always refers to a combination of **pipeline stage + access mask** which memory is *visible to*

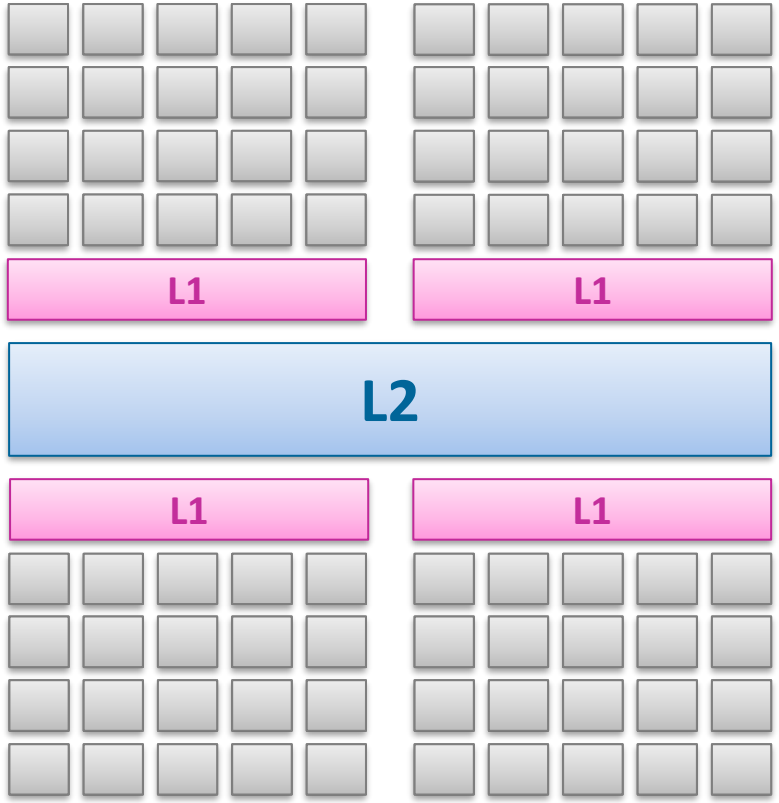
Memory is "**visible**"

Memory is no longer **available**

Can be made available again using a **memory barrier**:

```
VkMemoryBarrier2 memoryBarrier;
memoryBarrier.srcStageMask =
  VK_PIPELINE_STAGE_2_COLOR_ATTACHMENT_OUTPUT_BIT;
memoryBarrier.srcAccessMask =
  VK_ACCESS_2_COLOR_ATTACHMENT_WRITE_BIT;
```

Memory Availability and Visibility



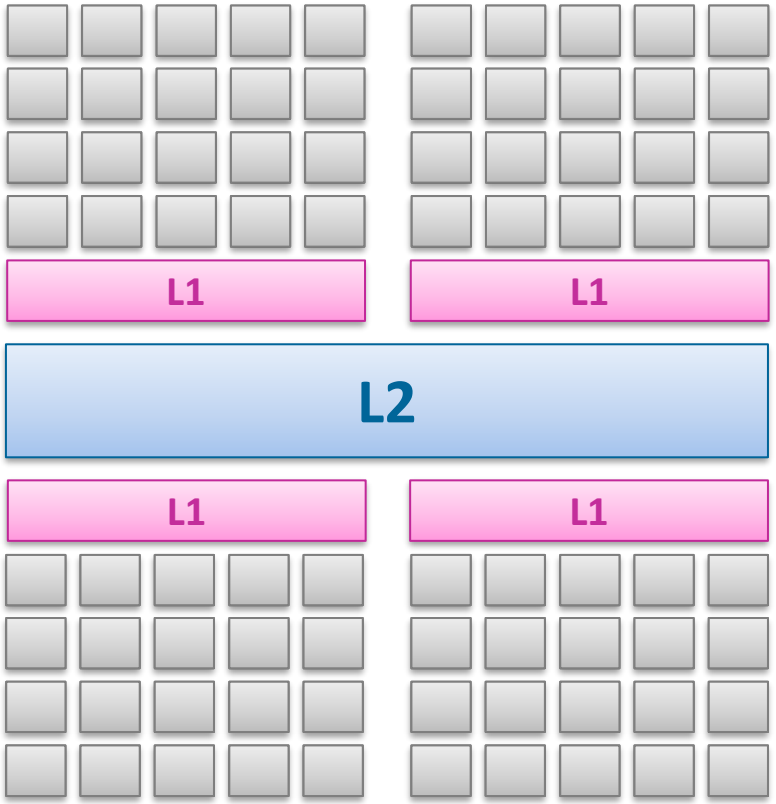
"**visibility**" always refers to a combination of **pipeline stage + access mask** which memory is *visible to*

Memory is "**visible**"

Memory has become **available** again

"**availability**" is established *from* a combination of source **pipeline stage** + source **access mask**

Pipeline **Execution** Barriers



QUEUE

CMD 1
GRAPHICS

COLOR_ATTACHMENT_OUTPUT

CMD 2
GRAPHICS

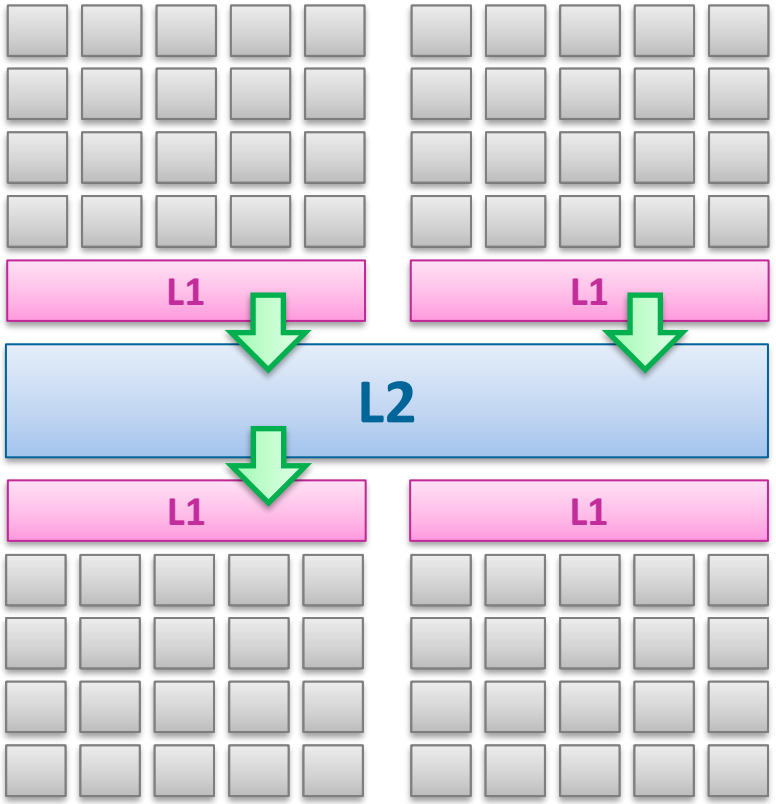
COLOR_ATTACHMENT_OUTPUT

COLOR_ATTACHMENT_OUTPUT >> FRAGMENT_SHADER

CMD 3
GRAPHICS

FRAGMENT_SHADER

Pipeline **Memory** Barriers



QUEUE

CMD 1
GRAPHICS

COLOR_ATTACHMENT_OUTPUT

CMD 2
GRAPHICS

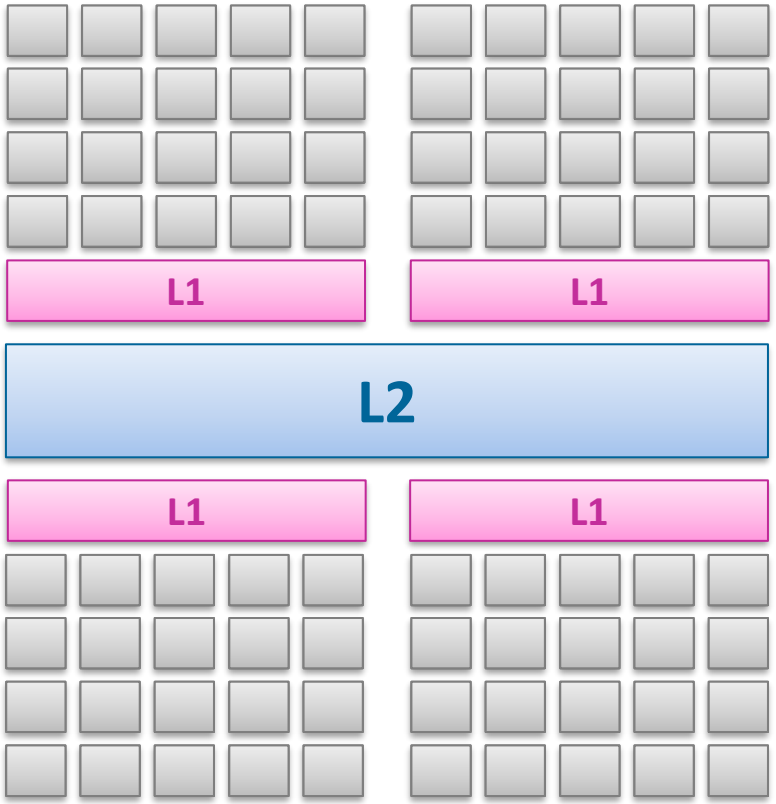
COLOR_ATTACHMENT_OUTPUT

COLOR_ATTACHMENT_OUTPUT >> FRAGMENT_SHADER
COLOR_ATTACHMENT_WRITE >> SHADER_READ

CMD 3
GRAPHICS

FRAGMENT_SHADER

Pipeline **Memory** Barriers



QUEUE

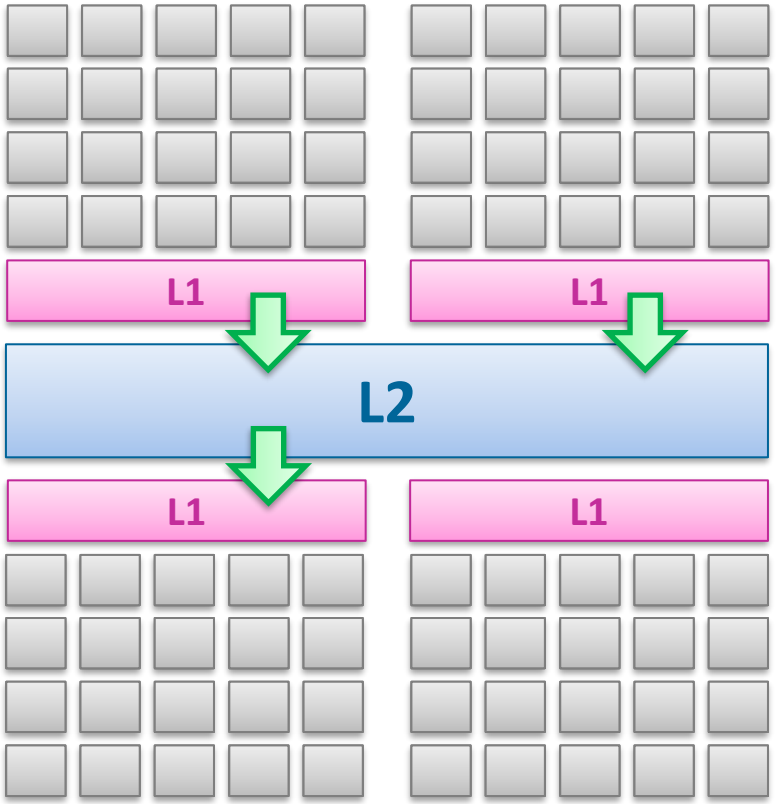
CMD 1
GRAPHICS

CMD 2
GRAPHICS

CMD 3
GRAPHICS

```
COLOR_ATTACHMENT_OUTPUT >> FRAGMENT_SHADER
COLOR_ATTACHMENT_WRITE  >>     SHADER_READ
```

Pipeline **Memory** Barriers



QUEUE

CMD 1
GRAPHICS

COLOR_ATTACHMENT_OUTPUT

CMD 2
COMPUTE

COMPUTE_SHADER

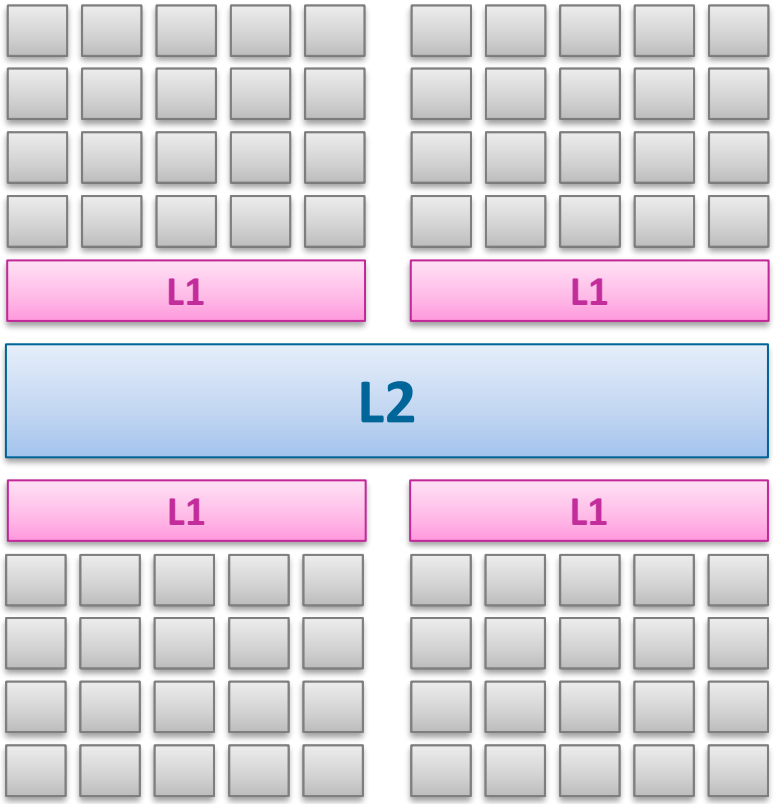
COLOR_ATTACHMENT_OUTPUT >> FRAGMENT_SHADER
COLOR_ATTACHMENT_WRITE >> SHADER_READ

COMPUTE_SHADER >> FRAGMENT_SHADER
SHADER_WRITE >> SHADER_READ

CMD 3
GRAPHICS

FRAGMENT_SHADER

Pipeline **Memory** Barriers



QUEUE

CMD 1
GRAPHICS

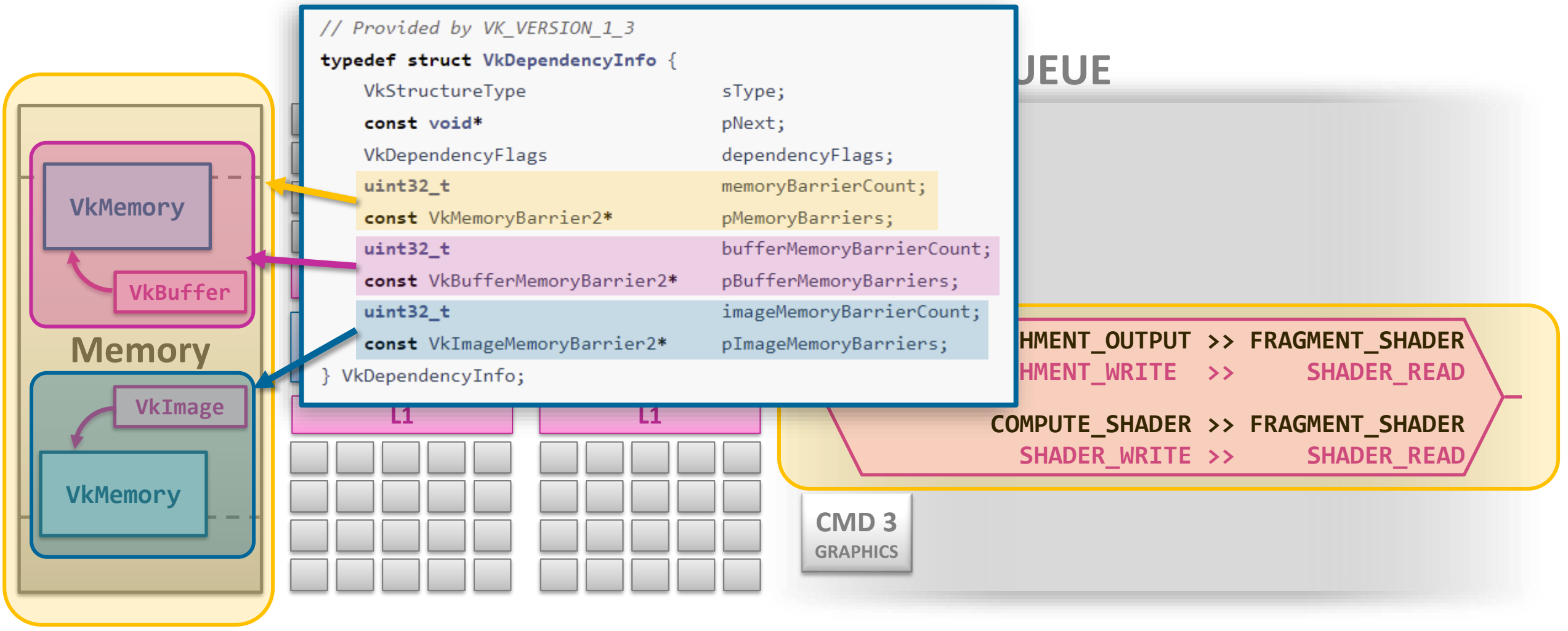
CMD 2
COMPUTE

```
COLOR_ATTACHMENT_OUTPUT >> FRAGMENT_SHADER
COLOR_ATTACHMENT_WRITE  >>      SHADER_READ

COMPUTE_SHADER >> FRAGMENT_SHADER
SHADER_WRITE   >>      SHADER_READ
```

CMD 3
GRAPHICS

Pipeline **Memory** Barriers



Overview of Synchronization Methods

- Wait Idle Operations (device -> host sync)
- Fences (device -> host sync)
- Semaphores (queue sync)
  - Binary Semaphores (queue -> queue sync, device only)
  - Timeline Semaphores (queue -> queue sync, host <-> device sync)
- Pipeline Barriers (command -> command sync, intra-queue)
  - Execution Barriers (execution-only dependency, memory disregarded)
  - Memory Barriers
- Render Pass Subpass Dependencies
- Events

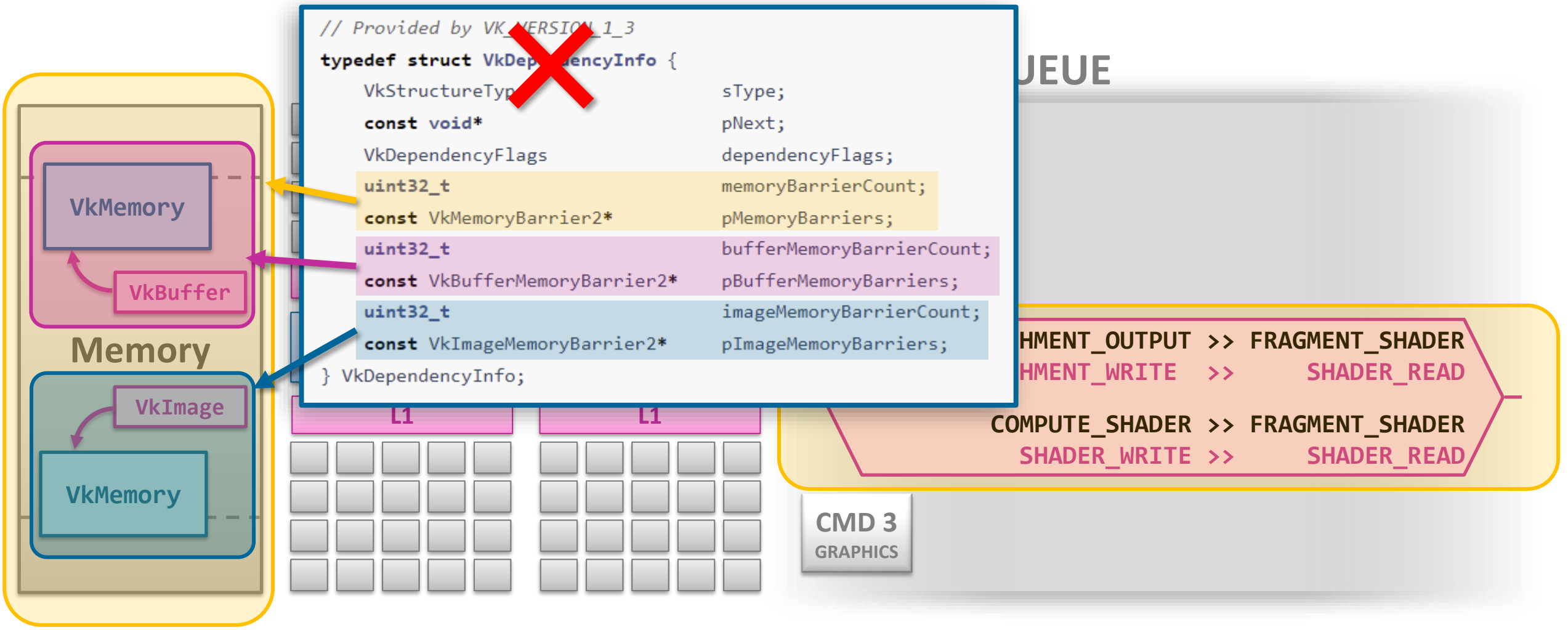
Overview of Synchronization Methods

- Wait Idle Operations (device -> host sync)
- Fences (device -> host sync)
- Semaphores (queue sync)
  - Binary Semaphores (queue -> queue sync, device only)
  - Timeline Semaphores (queue -> queue sync, host <-> device sync)
- Pipeline Barriers (command -> command sync, intra-queue)
  - Execution Barriers (execution-only dependency, memory disregarded)
  - Memory Barriers (execution and memory dependencies)
- Render Pass Subpass Dependencies
- Events

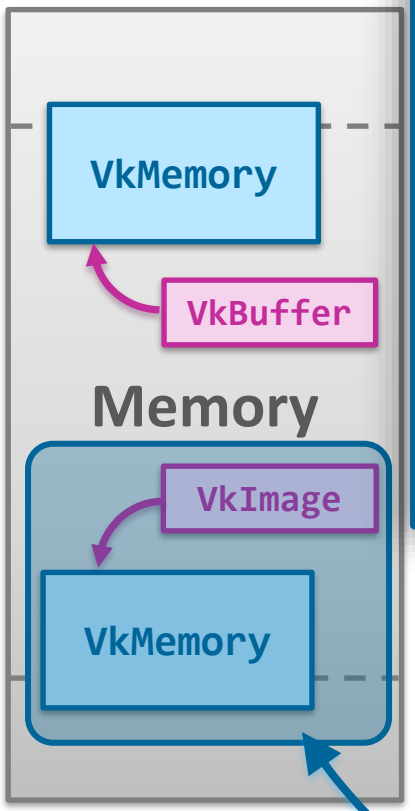
Overview of Synchronization Methods

- Wait Idle Operations (device -> host sync)
- Fences (device -> host sync)
- Semaphores (queue sync)
  - Binary Semaphores (queue -> queue sync, device only)
  - Timeline Semaphores (queue -> queue sync, host <-> device sync)
- Pipeline Barriers (command -> command sync, intra-queue)
  - Execution Barriers (execution-only dependency, memory disregarded)
  - Memory Barriers (execution and memory dependencies)
- Render Pass Subpass Dependencies
- Events

Limiting the Synchronization Scope...



Render Pass Subpass Dependencies




```
// Provided by VK_VERSION_1_2
typedef struct VkSubpassDependency2 {
    VkStructureType         sType;
    const void*             pNext;
    uint32_t                srcSubpass;
    uint32_t                dstSubpass;
    VkPipelineStageFlags    srcStageMask;
    VkPipelineStageFlags    dstStageMask;
    VkAccessFlags           srcAccessMask;
    VkAccessFlags           dstAccessMask;
    VkDependencyFlags       dependencyFlags;
    int32_t                 viewOffset;
} VkSubpassDependency2;
```

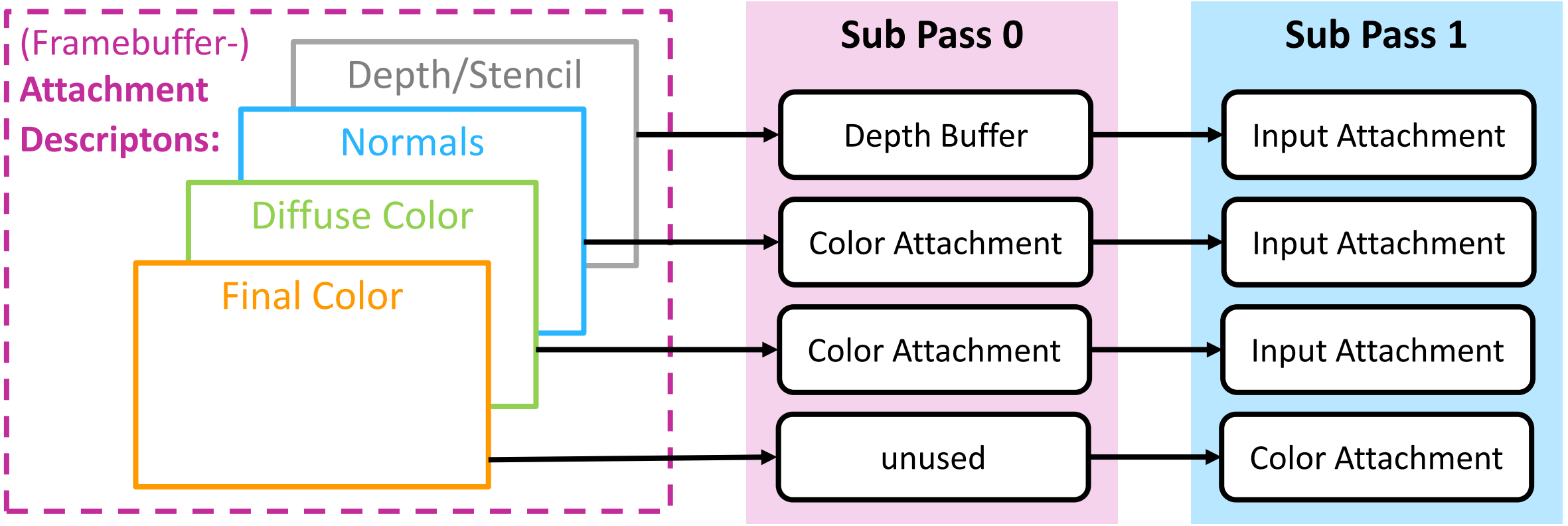
```
// Provided by VK_VERSION_1_2
void vkCmdBeginRenderPass2(
    VkCommandBuffer                 commandBuffer,
    const VkRenderPassBeginInfo*    pRenderPassBegin,
    const VkSubpassBeginInfo*       pSubpassBeginInfo);
```

```
// Provided by VK_VERSION_1_0
typedef struct VkRenderPassBeginInfo {
    VkStructureType         sType;
    const void*             pNext;
    VkRenderPass            renderPass;
    VkFramebuffer           framebuffer;
    VkRect2D                renderArea;
    uint32_t                clearValueCount;
    const VkClearValue*     pClearValues;
} VkRenderPassBeginInfo;
```

```
// Provided by VK_VERSION_1_0
typedef struct VkFramebufferCreateInfo {
    VkStructureType             sType;
    const void*                 pNext;
    VkFramebufferCreateFlags    flags;
    VkRenderPass                renderPass;
    uint32_t                    attachmentCount;
    const VkImageView*          pAttachments;
    uint32_t                    width;
    uint32_t                    height;
    uint32_t                    layers;
} VkFramebufferCreateInfo;
```

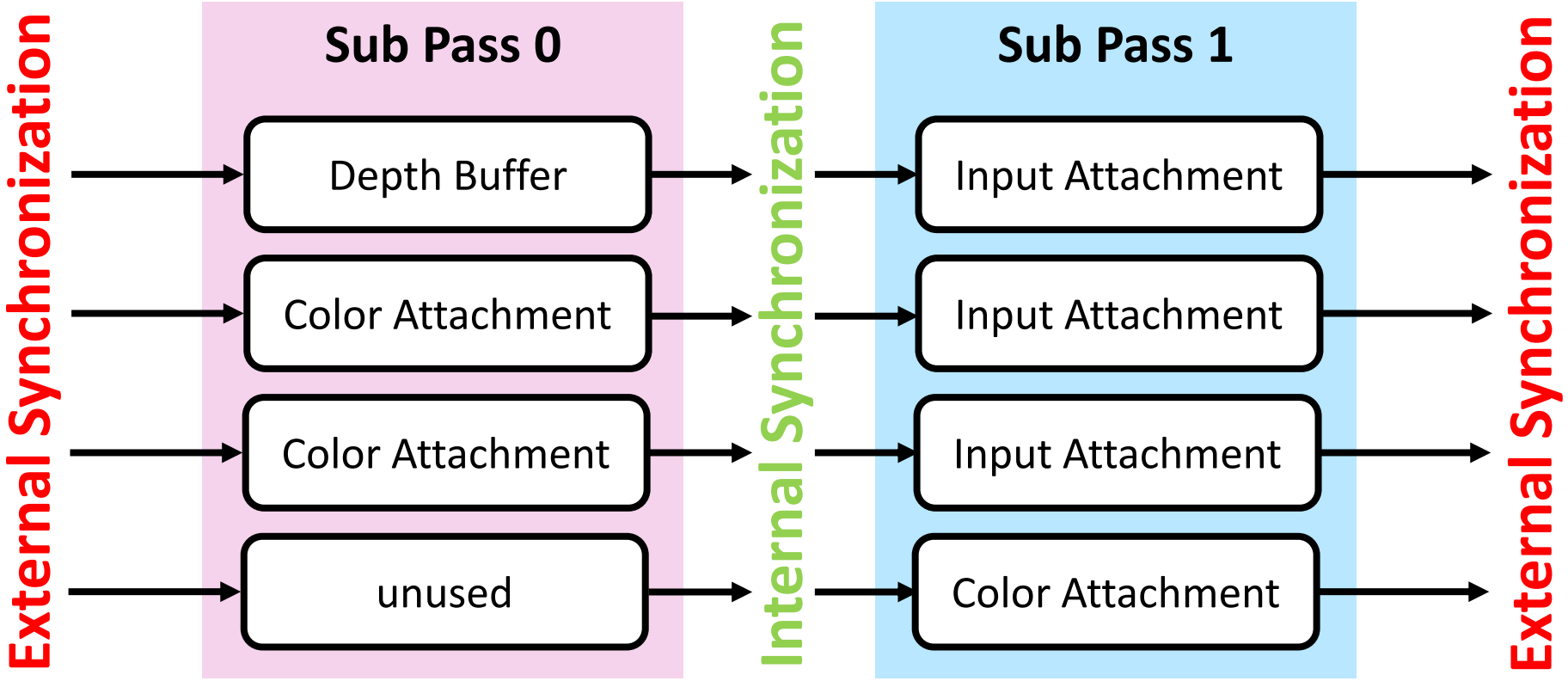
Render Pass

- Only for Graphics Pipelines
- Describes how framebuffer attachments are used
- Describes synchronization between multiple “sub passes”



Sub-pass Dependencies

- **External** and **internal** synchronization, i.e. synchronize with other commands, or synchronize with graphics stages and attachments
- **Stage flags + access flags** (same principle as with pipeline barriers)



Overview of Synchronization Methods

- Wait Idle Operations (device -> host sync)
- Fences (device -> host sync)
- Semaphores (queue sync)
  - Binary Semaphores (queue -> queue sync, device only)
  - Timeline Semaphores (queue -> queue sync, host <-> device sync)
- Pipeline Barriers (command -> command sync, intra-queue)
  - Execution Barriers (execution-only dependency, memory disregarded)
  - Memory Barriers (execution and memory dependencies)
- Render Pass Subpass Dependencies
- Events

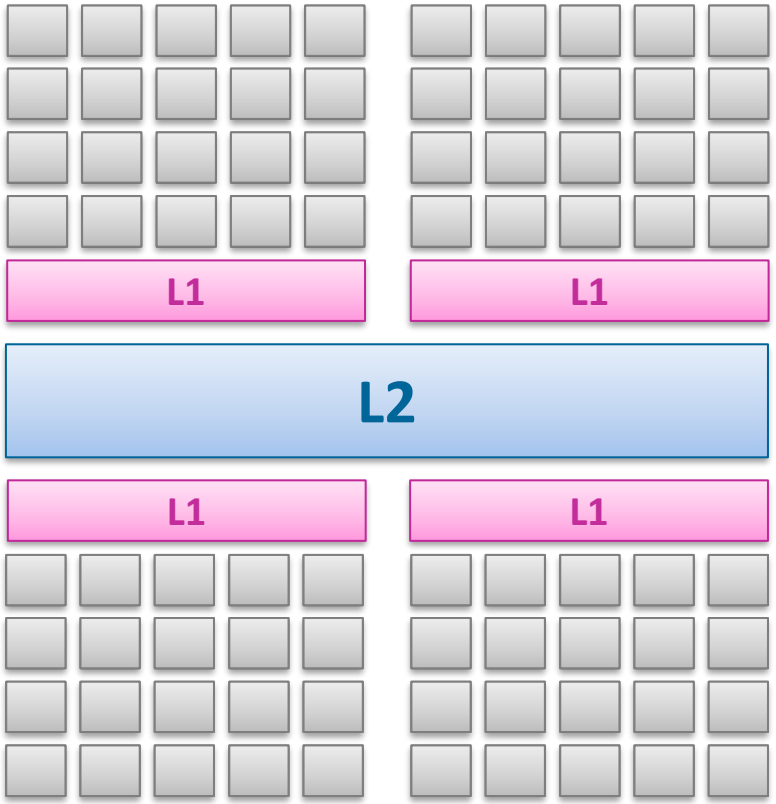
Overview of Synchronization Methods

- Wait Idle Operations (device -> host sync)
- Fences (device -> host sync)
- Semaphores (queue sync)
  - Binary Semaphores (queue -> queue sync, device only)
  - Timeline Semaphores (queue -> queue sync, host <-> device sync)
- Pipeline Barriers (command -> command sync, intra-queue)
  - Execution Barriers (execution-only dependency, memory disregarded)
  - Memory Barriers (execution and memory dependencies)
- Render Pass Subpass Dependencies (subpass memory dependencies)
- Events

Overview of Synchronization Methods

- Wait Idle Operations (device -> host sync)
- Fences (device -> host sync)
- Semaphores (queue sync)
  - Binary Semaphores (queue -> queue sync, device only)
  - Timeline Semaphores (queue -> queue sync, host <-> device sync)
- Pipeline Barriers (command -> command sync, intra-queue)
  - Execution Barriers (execution-only dependency, memory disregarded)
  - Memory Barriers (execution and memory dependencies)
- Render Pass Subpass Dependencies (subpass memory dependencies)
- Events

Events



QUEUE

CMD 1
GRAPHICS

COLOR_ATTACHMENT_OUTPUT

vkCmdSetEvent(X, COLOR_ATTACHMENT_OUTPUT)

CMD 2
GRAPHICS

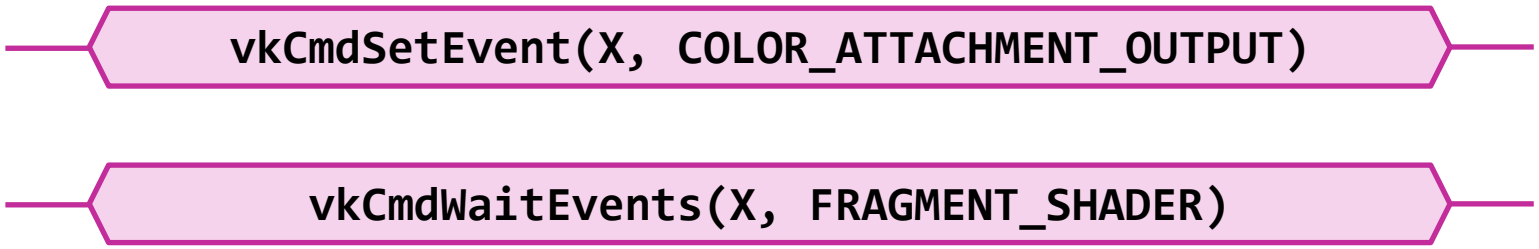
CMD 3
COMPUTE

vkCmdWaitEvents(X, FRAGMENT_SHADER)

CMD 4
GRAPHICS

FRAGMENT_SHADER

Events



- To signal an event from a device: `vkCmdSetEvent2`
- To wait for one or more events to enter the signaled state on a device: `vkCmdWaitEvents2`
- To unsignal the event from a device: `vkCmdResetEvent2`

- To set the state of an event to signaled from the host: `vkSetEvent`
- To query the state of an event from the host: `vkGetEventStatus`
- To set the state of an event to unsignaled from the host: `vkResetEvent`

Overview of Synchronization Methods

- Wait Idle Operations (device -> host sync)
- Fences (device -> host sync)
- Semaphores (queue sync)
  - Binary Semaphores (queue -> queue sync, device only)
  - Timeline Semaphores (queue -> queue sync, host <-> device sync)
- Pipeline Barriers (command -> command sync, intra-queue)
  - Execution Barriers (execution-only dependency, memory disregarded)
  - Memory Barriers (execution and memory dependencies)
- Render Pass Subpass Dependencies (subpass memory dependencies)
- Events

Overview of Synchronization Methods

- Wait Idle Operations (device -> host sync)
- Fences (device -> host sync)
- Semaphores (queue sync)
  - Binary Semaphores (queue -> queue sync, device only)
  - Timeline Semaphores (queue -> queue sync, host <-> device sync)
- Pipeline Barriers (command -> command sync, intra-queue)
  - Execution Barriers (execution-only dependency, memory disregarded)
  - Memory Barriers (execution and memory dependencies)
- Render Pass Subpass Dependencies (subpass memory dependencies)
- Events (“split barriers”, host -> device sync)

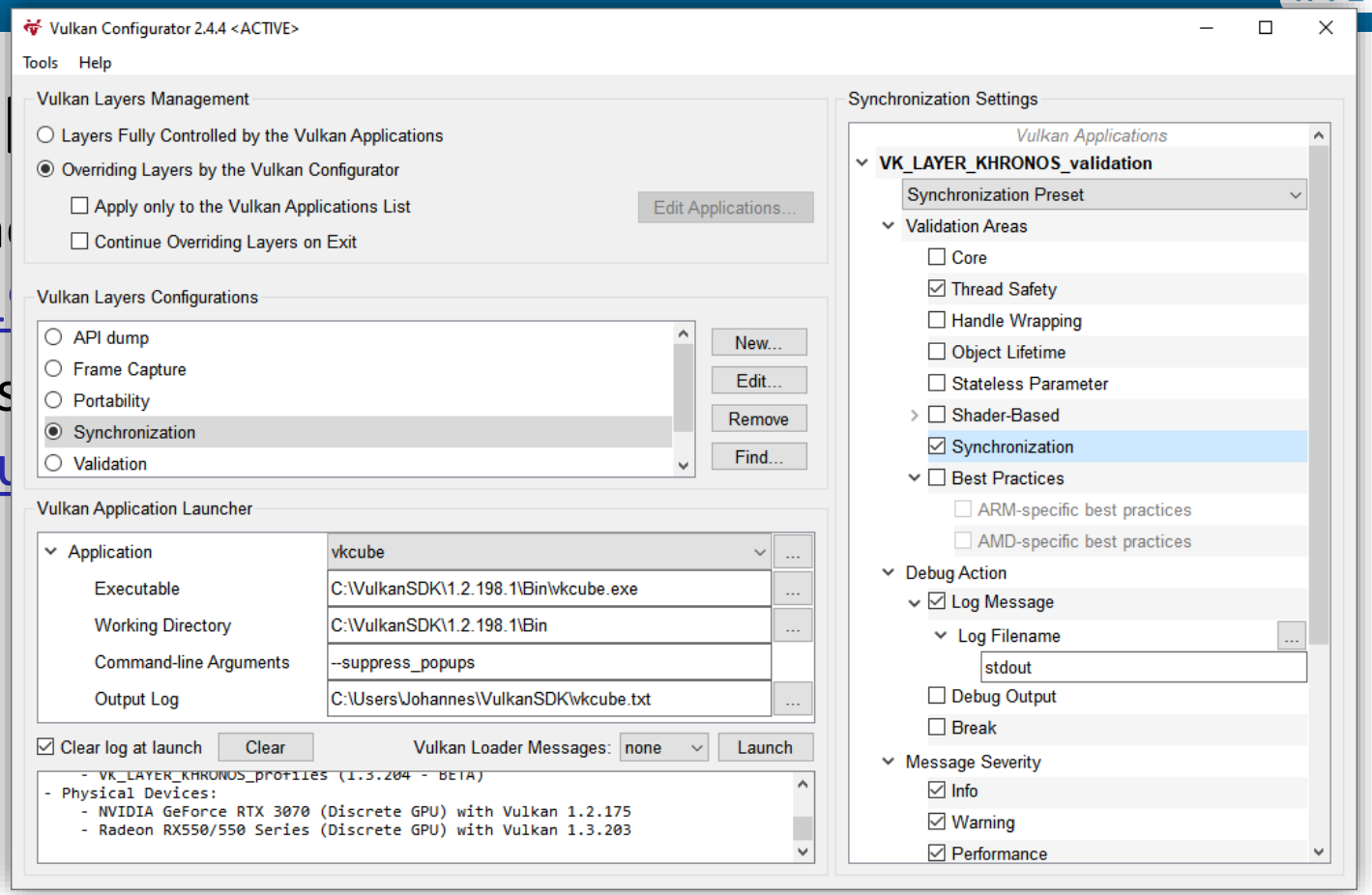
Useful Stuff

- Great resources about synchronization
  - Hans-Kristian Arntzen. Yet another blog explaining Vulkan synchronization. https://themaister.net/blog/2019/08
  - The Khronos Group. Vulkan-Docs. Synchronization Examples https://github.com/KhronosGroup/Vulkan-Docs/wiki/Synchronization-Examples
  - Jason Ekstrand. Vulkan Timeline Semaphores. https://youtu.be/SpE--Rf516Y

- Synchronization validation
  - `VK_LAYER_KHRONOS_validation`
  - The Khronos Group. Vulkan-ValidationLayers. Synchronization Validation https://github.com/KhronosGroup/Vulkan-ValidationLayers/blob/master/docs/synchronization_usage.md

Useful Stuff

- Great resources about synch
  - Hans-Kristian Arntzen. Yet anoth
    https://themaister.net/blog/201
  - The Khronos Group. Vulkan-Docs
    https://github.com/KhronosGrou
  - Jason Ekstrand. Vulkan Timeline
    https://youtu.be/SpE--Rf516Y



- Synchronization validation
  - `VK_LAYER_KHRONOS_validation`
  - The Khronos Group. Vulkan-ValidationLayers. Synchronization Validation
    https://github.com/KhronosGroup/Vulkan-ValidationLayers/blob/master/docs/synchronization_usage.md

Algorithms for Real-Time Rendering

186.192, 2022S, 3.0ECTS

Thank you for your attention!

Johannes Unterguggenberger

Institute of Visual Computing & Human-Centered Technology

TU Wien, Austria