



Synchronization



Oregon State
University

Mike Bailey

mjb@cs.oregonstate.edu



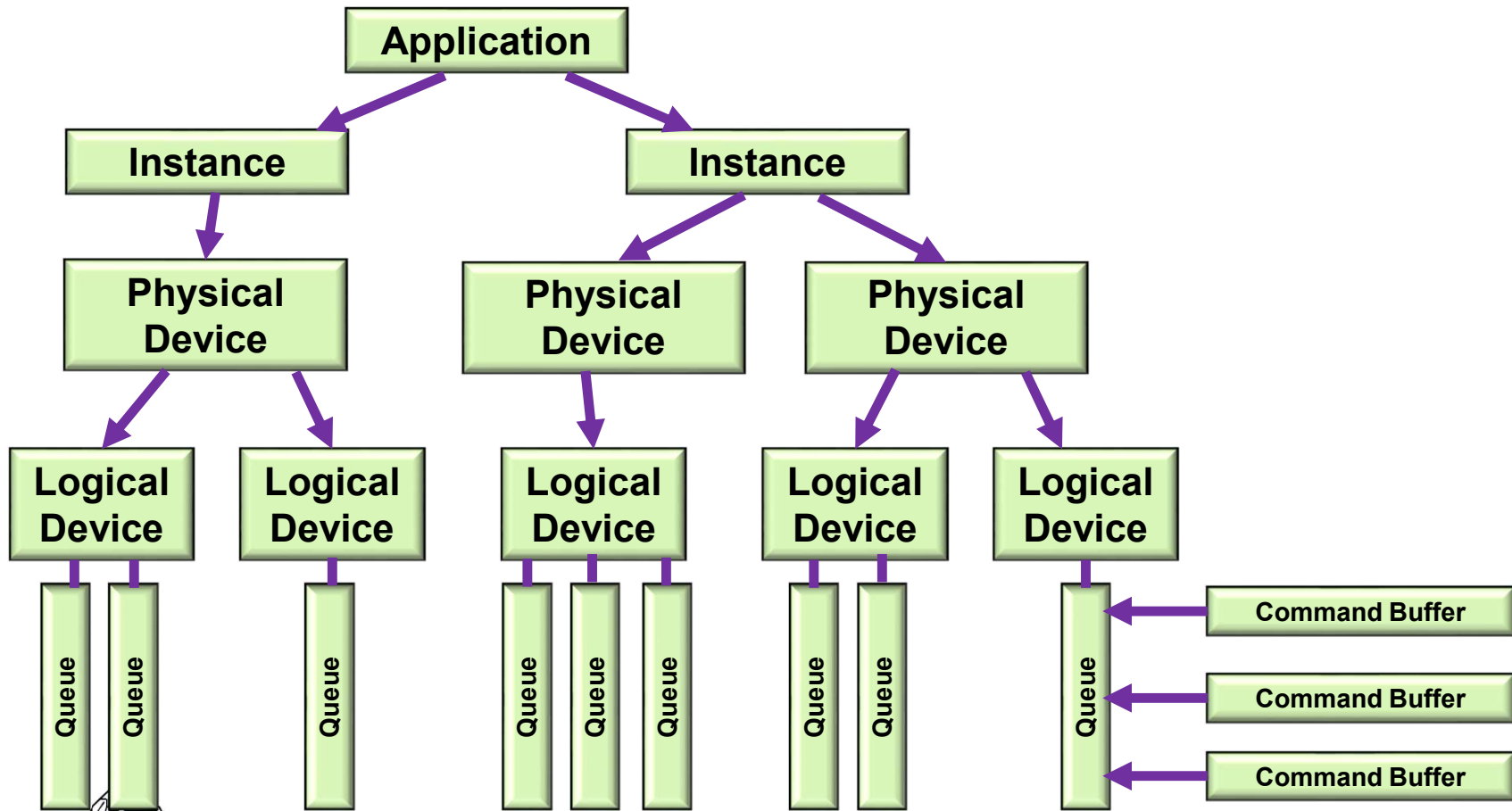
This work is licensed under a [Creative Commons Attribution-NonCommercial-NoDerivatives 4.0 International License](https://creativecommons.org/licenses/by-nc-nd/4.0/)



Oregon State
University
Computer Graphics

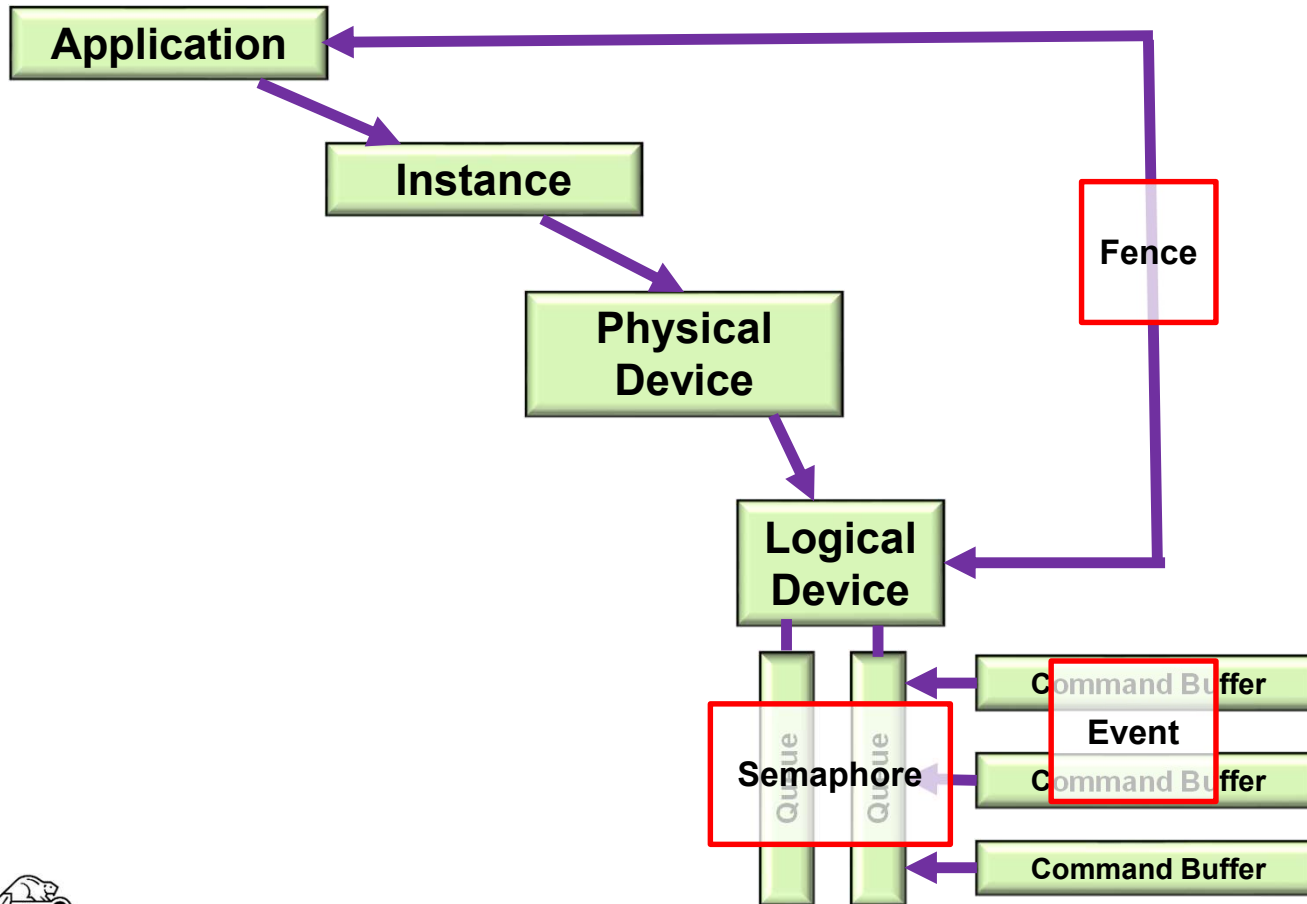
Remember the Overall Block Diagram?

2



Where Synchronization Fits in the Overall Block Diagram

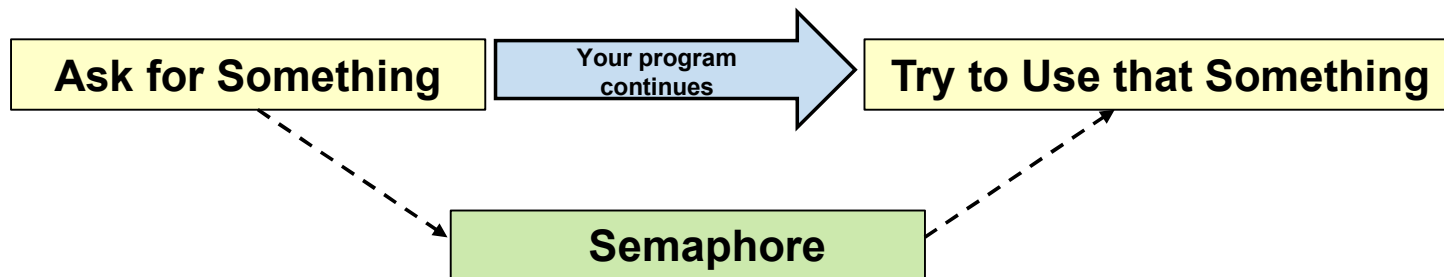
3



Semaphores

4

- Indicates that a batch of commands has been processed from a queue. Basically announces “I am finished!”.
- You create one and give it to a Vulkan function which sets it. Later on, you tell another Vulkan function to wait for this semaphore to be signaled.
- You don’t end up setting, resetting, or checking the semaphore yourself.
- Semaphores must be initialized (“created”) before they can be used.



Creating a Semaphore

5

```
VkSemaphoreCreateInfo    vsci;  
    vsci.sType = VK_STRUCTURE_TYPE_SEMAPHORE_CREATE_INFO;  
    vsci.pNext = nullptr;  
    vsci.flags = 0;;  
  
VkSemaphore              semaphore;  
result = vkCreateSemaphore( LogicalDevice, IN &vsci, PALLOCATOR, OUT &semaphore );
```

This doesn't actually do anything with the semaphore – it just sets it up



Semaphores Example during the Render Loop

6

```
VkSemaphore imageReadySemaphore;
```

```
VkSemaphoreCreateInfo
```

```

    vsci;
    vsci.sType = VK_STRUCTURE_TYPE_SEMAPHORE_CREATE_INFO;
    vsci.pNext = nullptr;
    vsci.flags = 0;

```

```
result = vkCreateSemaphore( LogicalDevice, IN &vsci, PALLOCATOR, OUT &imageReadySemaphore );
```

```
uint32_t nextImageIndex;
```

```
vkAcquireNextImageKHR( LogicalDevice, IN SwapChain, IN UINT64_MAX,
    IN imageReadySemaphore, IN VK_NULL_HANDLE, OUT &nextImageIndex );
```

Set the semaphore

```
...
```

```
VkPipelineStageFlags waitAtBottomOfPipe = VK_PIPELINE_STAGE_BOTTOM_OF_PIPE_BIT;
```

```
VkSubmitInfo
```

```

    vsi;
    vsi.sType = VK_STRUCTURE_TYPE_SUBMIT_INFO;
    vsi.pNext = nullptr;
    vsi.waitSemaphoreCount = 1;
    vsi.pWaitSemaphores = &imageReadySemaphore;
    vsi.pWaitDstStageMask = &waitAtBottomOfPipe;
    vsi.commandBufferCount = 1;
    vsi.pCommandBuffers = &CommandBuffers[nextImageIndex];
    vsi.signalSemaphoreCount = 0;
    vsi.pSignalSemaphores = (VkSemaphore) nullptr;

```

Wait on the semaphore

```
result = vkQueueSubmit( presentQueue, 1, IN &vsi, IN renderFence );
```

You do this to wait for an image to be ready to be rendered into

Fences

7

- Used to synchronize CPU-GPU tasks.
- Used when the host needs to wait for the device to complete something big.
- Announces that queue-submitted work is finished.
- You can un-signal, signal, test or block-while-waiting.



Fences

8

```

#define VK_FENCE_CREATE_UNSIGNALED_BIT    0

VkFenceCreateInfo
    vfc_i;
    vfc_i.sType = VK_STRUCTURE_TYPE_FENCE_CREATE_INFO;
    vfc_i.pNext = nullptr;
    vfc_i.flags = VK_FENCE_CREATE_UNSIGNALED_BIT; // = 0
                // VK_FENCE_CREATE_SIGNALED_BIT is only other option

VkFence    fence;
result = vkCreateFence( LogicalDevice, IN &vfc_i, PALLOCATOR, OUT &fence );

    , , ,

// returns to the host right away:
result = vkGetFenceStatus( LogicalDevice, IN fence );
    // result = VK_SUCCESS means it has signaled
    // result = VK_NOT_READY means it has not signaled

// blocks the host from executing:
result = vkWaitForFences( LogicalDevice, 1, IN &fence, waitForAll, timeout );
    // waitForAll = VK_TRUE:  wait for all fences in the list
    // waitForAll = VK_FALSE: wait for any one fence in the list
    // timeout is a uint64_t timeout in nanoseconds (could be 0, which means to return immediately)
    // timeout can be up to UINT64_MAX = 0xfffffffffffff (= 580+ years)
    // result = VK_SUCCESS means it returned because a fence (or all fences) signaled
    // result = VK_TIMEOUT means it returned because the timeout was exceeded

```

Set the fence

Wait on the fence(s)

Fence Example

9

```

VkFence          renderFence;
vkCreateFence( LogicalDevice, &vpci, PALLOCATOR, OUT &renderFence );

VkPipelineStageFlags waitAtBottom = VK_PIPELINE_STAGE_BOTTOM_OF_PIPE_BIT;

VkQueue presentQueue;
vkGetDeviceQueue( LogicalDevice, FindQueueFamilyThatDoesGraphics( ), 0, OUT &presentQueue );

VkSubmitInfo
    vsi;
    vsi.sType = VK_STRUCTURE_TYPE_SUBMIT_INFO;
    vsi.pNext = nullptr;
    vsi.waitSemaphoreCount = 1;
    vsi.pWaitSemaphores = &imageReadySemaphore;
    vsi.pWaitDstStageMask = &waitAtBottom;
    vsi.commandBufferCount = 1;
    vsi.pCommandBuffers = &CommandBuffers[nextImageIndex];
    vsi.signalSemaphoreCount = 0;
    vsi.pSignalSemaphores = (VkSemaphore) nullptr;

result = vkQueueSubmit( presentQueue, 1, IN vsi, IN renderFence );
...

result = vkWaitForFences( LogicalDevice, 1, IN &renderFence, VK_TRUE, UINT64_MAX );
...

result = vkQueuePresentKHR( presentQueue, IN &vpi );           // don't present the image until done rendering

```

Events

10

- Events provide even finer-grained synchronization.
- Events are a primitive that can be signaled by the host or the device.
- Can even signal at one place in the pipeline and wait for it at another place in the pipeline.
- Signaling in the pipeline means “signal me as the last piece of this draw command passes that point in the pipeline”.
- You can signal, un-signal, or test from a vk function or from a vkCmd function.
- Can wait from a vkCmd function.



Oregon State
University
Computer Graphics

mjb – February 26, 2023

Controlling Events from the Host

11

```

VkEventCreateInfo
    veci.sType = VK_STRUCTURE_TYPE_EVENT_CREATE_INFO;
    veci.pNext = nullptr;
    veci.flags = 0;

VkEvent      event;
result = vkCreateEvent( LogicalDevice, IN &veci, PALLOCATOR, OUT &event );

result = vkSetEvent( LogicalDevice, IN event );

result = vkResetEvent( LogicalDevice, IN event );

result = vkGetEventStatus( LogicalDevice, IN event );
    // result = VK_EVENT_SET: signaled
    // result = VK_EVENT_RESET: not signaled
  
```

Note: the host cannot *block* waiting for an event, but it can test for it



Controlling Events from the Device

12

```
result = vkCmdSetEvent(  CommandBuffer, IN event, pipelineStageBits );
```

```
result = vkCmdResetEvent( CommandBuffer, IN event, pipelineStageBits );
```

```
result = vkCmdWaitEvents( CommandBuffer, 1, &event,
```

```
srcPipelineStageBits, dstPipelineStageBits,
```

```
memoryBarrierCount, pMemoryBarriers,  
bufferMemoryBarrierCount, pBufferMemoryBarriers,  
imageMemoryBarrierCount, pImageMemoryBarriers
```

```
);
```

Could be an
array of events

Where signaled, where wait
for the signal

Memory barriers get executed
after events have been signaled

Note: the device cannot *test* for an event, but it can block

