




1




Pipeline Barriers: A case of Gate-ing and Wait-ing



**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

PipelineBarriers.pptx mjb – October 4 2018

2


From the Command Buffer Notes:

These are the Commands that can be entered into the Command Buffer, I

```

vkCmdBeginQuery( commandBuffer, flags );
vkCmdBeginRenderPass( commandBuffer, const contents );
vkCmdBindDescriptorSets( commandBuffer, pDynamicOffsets );
vkCmdBindIndexBuffer( commandBuffer, indexType );
vkCmdBindPipeline( commandBuffer, pipeline );
vkCmdBindVertexBuffers( commandBuffer, firstBinding, bindingCount, const pOffsets );
vkCmdBlitImage( commandBuffer, filter );
vkCmdClearAttachments( commandBuffer, attachmentCount, const pRects );
vkCmdClearColorImage( commandBuffer, pRanges );
vkCmdClearDepthStencilImage( commandBuffer, pRanges );
vkCmdCopyBuffer( commandBuffer, pRegions );
vkCmdCopyBufferToImage( commandBuffer, pRegions );
vkCmdCopyImage( commandBuffer, pRegions );
vkCmdCopyImageToBuffer( commandBuffer, pRegions );
vkCmdCopyQueryPoolResults( commandBuffer, flags );
vkCmdDebugMarkerBeginEXT( commandBuffer, pMarkerInfo );
vkCmdDebugMarkerEndEXT( commandBuffer );
vkCmdDebugMarkerInsertEXT( commandBuffer, pMarkerInfo );
vkvCmdDispatch( commandBuffer, groupCountX, groupCountY, groupCountZ );
vkCmdDispatchIndirect( commandBuffer, offset );
vkCmdDraw( commandBuffer, vertexCount, instanceCount, firstVertex, firstInstance );
vkCmdDrawIndexed( commandBuffer, indexCount, instanceCount, firstIndex, int32_t vertexOffset, firstInstance );
vkCmdDrawIndexedIndirect( commandBuffer, stride );
vkCmdDrawIndexedIndirectCountAMD( commandBuffer, stride );
vkCmdDrawIndirect( commandBuffer, stride );
vkCmdDrawIndirectCountAMD( commandBuffer, stride );
vkCmdEndQuery( commandBuffer, query );
vkCmdEndRenderPass( commandBuffer );
vkCmdExecuteCommands( commandBuffer, commandBufferCount, const pCommandBuffers );

```



**Oregon State
University**
Computer Graphics

We don't any one of these commands to have to wait on a previous command unless you say so. In general, we want all of these commands to be able to run **"flat-out"**.

But, if we do that, surely there will be nasty **race conditions!**

mjb – October 4 2018

3


From the Command Buffer Notes:

These are the Commands that can be entered into the Command Buffer, II

```

vkCmdFillBuffer( commandBuffer, dstBuffer, dstOffset, size, data );
vkCmdNextSubpass( commandBuffer, contents );
vkCmdPipelineBarrier( commandBuffer, srcStageMask, dstStageMask, dependencyFlags, memoryBarrierCount, VkMemoryBarrier* pMemoryBarriers,
    bufferMemoryBarrierCount, pBufferMemoryBarriers, imageMemoryBarrierCount, plmageMemoryBarriers );
vkCmdProcessCommandsNVX( commandBuffer, pProcessCommandsInfo );
vkCmdPushConstants( commandBuffer, layout, stageFlags, offset, size, pValues );
vkCmdPushDescriptorSetKHR( commandBuffer, pipelineBindPoint, layout, set, descriptorWriteCount, pDescriptorWrites );
vkCmdPushDescriptorSetWithTemplateKHR( commandBuffer, descriptorUpdateTemplate, layout, set, pData );
vkCmdReserveSpaceForCommandsNVX( commandBuffer, pReserveSpaceInfo );
vkCmdResetEvent( commandBuffer, event, stageMask );
vkCmdResetQueryPool( commandBuffer, queryPool, firstQuery, queryCount );
vkCmdResolveImage( commandBuffer, srcImage, srcImageLayout, dstImage, dstImageLayout, regionCount, pRegions );
vkCmdSetBlendConstants( commandBuffer, blendConstants[4] );
vkCmdSetDepthBias( commandBuffer, depthBiasConstantFactor, depthBiasClamp, depthBiasSlopeFactor );
vkCmdSetDepthBounds( commandBuffer, minDepthBounds, maxDepthBounds );
vkCmdSetDeviceMaskKH( commandBuffer, deviceMask );
vkCmdSetDiscardRectangleEXT( commandBuffer, firstDiscardRectangle, discardRectangleCount, pDiscardRectangles );
vkCmdSetEvent( commandBuffer, event, stageMask );
vkCmdSetLineWidth( commandBuffer, lineWidth );
vkCmdSetScissor( commandBuffer, firstScissor, scissorCount, pScissors );
vkCmdSetStencilCompareMask( commandBuffer, faceMask, compareMask );
vkCmdSetStencilReference( commandBuffer, faceMask, reference );
vkCmdSetStencilWriteMask( commandBuffer, faceMask, writeMask );
vkCmdSetViewport( commandBuffer, firstViewport, viewportCount, pViewports );
vkCmdSetViewportWScalingNV( commandBuffer, firstViewport, viewportCount, pViewportWScalings );
vkCmdUpdateBuffer( commandBuffer, dstBuffer, dstOffset, dataSize, pData );
vkCmdWaitEvents( commandBuffer, eventCount, pEvents, srcStageMask, dstStageMask, memoryBarrierCount, pMemoryBarriers,
    bufferMemoryBarrierCount, pBufferMemoryBarriers, imageMemoryBarrierCount, plmageMemoryBarriers );
vkCmdWriteTimestamp( commandBuffer, pipelineStage, queryPool, query );

```



We don't any one of these commands to have to wait on a previous command unless you say so. In general, we want all of these commands to be able to run **"flat-out"**.

But, if we do that, surely there will be nasty **race conditions!**


mjb - October 4 2018

4

Potential Memory Race Conditions that Pipeline Barriers can Prevent

1. Write-then-Read (WtR) – the memory write in one operation starts overwriting the memory that another operation's read needs to use
2. Read-then-Write (RtW) – the memory read in one operation hasn't yet finished before another operation starts overwriting that memory
3. Write-then-Write (WtW) – two operations start overwriting the same memory and the end result is non-deterministic

Note: there is no problem with Read-then-Read (RtR) as no data has been changed



mjb - October 4 2018

vkCmdPipelineBarrier() Function Call

A **Pipeline Barrier** is a way to establish a memory dependency between commands that were submitted before the barrier and commands that are submitted after the barrier


```

vkCmdPipelineBarrier( commandBuffer,
    srcStageMask,
    dstStageMask,
    VK_DEPENDENCY_BY_REGION_BIT,
    memoryBarrierCount,      pMemoryBarriers,
    bufferMemoryBarrierCount, pBufferMemoryBarriers,
    imageMemoryBarrierCount, pImageMemoryBarriers
);
    
```

srcStageMask, ← Guarantee that this pipeline stage has completely generated one set of data before ...

dstStageMask, ← ... allowing this pipeline stage to consume it


Defines what data we will be blocking/un-blocking on



Oregon State University
Computer Graphics

mjb - October 4 2018

The Scenario



Oregon State University
Computer Graphics

mjb - October 4 2018

The Scenario Rules

1. The cross-streets are named after pipeline stages
2. All traffic lights start out green (“we want all of these commands to be able to run flat-out”)
3. There are special sensors at all intersections that will know when the **first car in the src group** enters that intersection
4. There are connections from those sensors to the traffic lights so that when the **first car in the src group** enters its intersection, the **dst** traffic light will be turned red
5. When the **last car in the src group** completely makes it through its intersection, the **dst** traffic light can be turned back to green
6. The Vulkan command pipeline ordering is this: (1) the **src** cars get released, (2) the pipeline barrier is invoked (which turns some lights red), (3) the **dst** cars get released (which end up being stopped by a red light somewhere), (4) the **src** cars clear their intersection, (5) the **dst** cars get released

mjb – October 4 2018

Pipeline Stage Masks – Where in the Pipeline is this Memory Data being Generated or Consumed?

VK_PIPELINE_STAGE_TOP_OF_PIPE_BIT

VK_PIPELINE_STAGE_DRAW_INDIRECT_BIT

VK_PIPELINE_STAGE_VERTEX_INPUT_BIT

VK_PIPELINE_STAGE_VERTEX_SHADER_BIT

VK_PIPELINE_STAGE_TESSELLATION_CONTROL_SHADER_BIT

VK_PIPELINE_STAGE_TESSELLATION_EVALUATION_SHADER_BIT

VK_PIPELINE_STAGE_GEOMETRY_SHADER_BIT

VK_PIPELINE_STAGE_FRAGMENT_SHADER_BIT

VK_PIPELINE_STAGE_EARLY_FRAGMENT_TESTS_BIT

VK_PIPELINE_STAGE_LATE_FRAGMENT_TESTS_BIT

VK_PIPELINE_STAGE_COLOR_ATTACHMENT_OUTPUT_BIT

VK_PIPELINE_STAGE_COMPUTE_SHADER_BIT

VK_PIPELINE_STAGE_TRANSFER_BIT

VK_PIPELINE_STAGE_BOTTOM_OF_PIPE_BIT

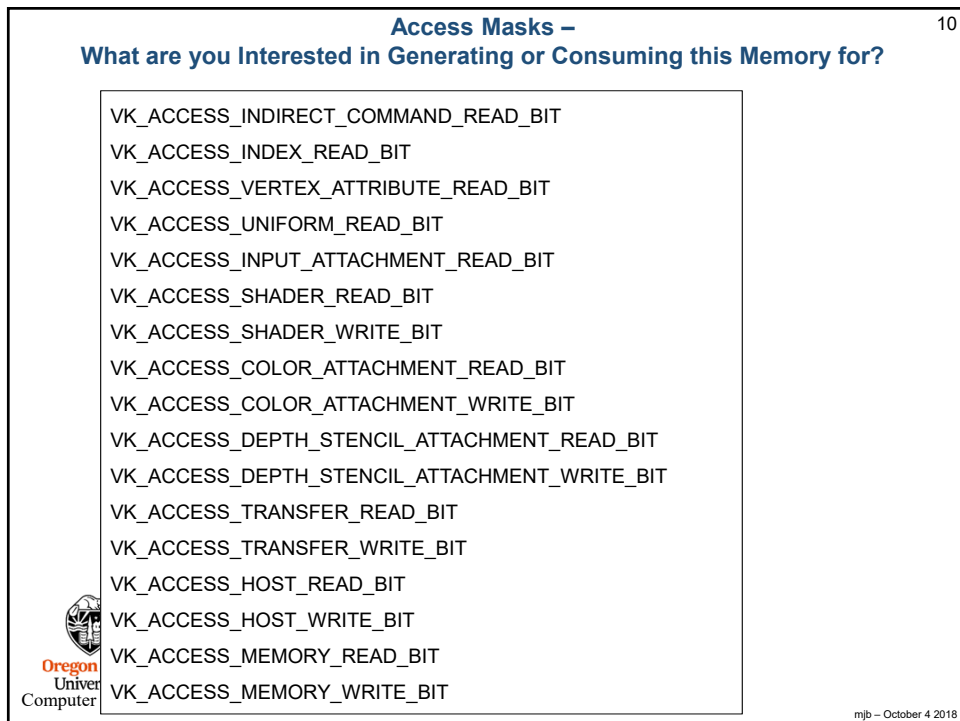
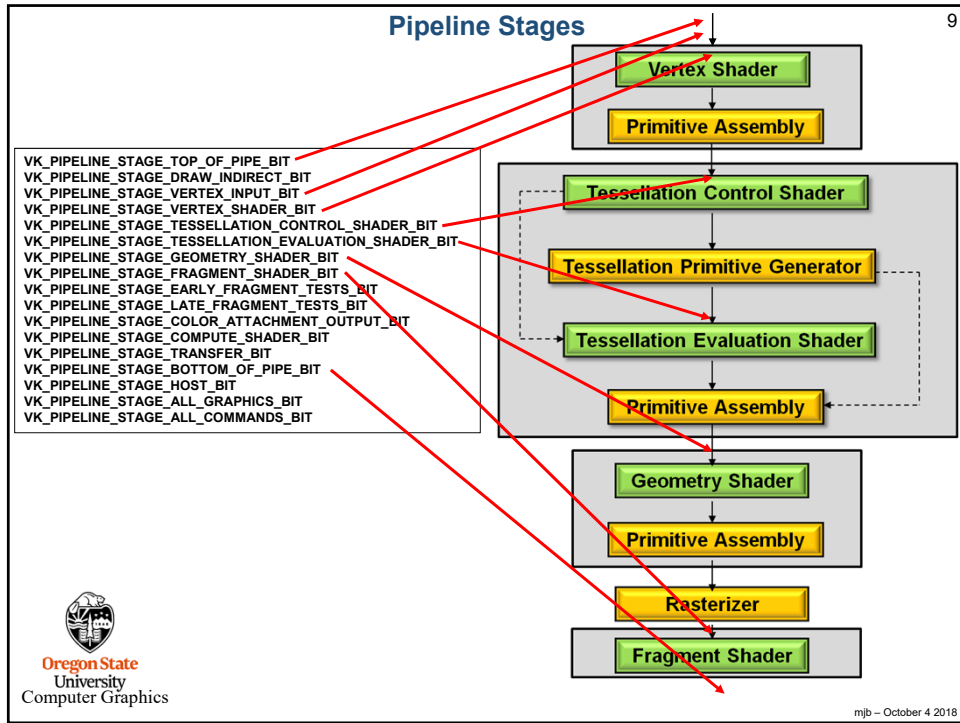
VK_PIPELINE_STAGE_HOST_BIT

VK_PIPELINE_STAGE_ALL_GRAPHICS_BIT

VK_PIPELINE_STAGE_ALL_COMMANDS_BIT

Oregon State University
 Computer Graphics

mjb – October 4 2018



Pipeline Stages and what Access Operations can Happen There

11

	VK_ACCESS_INDIRECT_COMMAND_READ_BIT	VK_ACCESS_INDEX_READ_BIT	VK_ACCESS_VERTEX_ATTRIBUTE_READ_BIT	VK_ACCESS_UNIFORM_READ_BIT	VK_ACCESS_INPUT_ATTACHMENT_READ_BIT	VK_ACCESS_SHADER_READ_BIT	VK_ACCESS_SHADER_WRITE_BIT	VK_ACCESS_COLOR_ATTACHMENT_READ_BIT	VK_ACCESS_COLOR_ATTACHMENT_WRITE_BIT	VK_ACCESS_DEPTH_STENCIL_ATTACHMENT_READ_BIT	VK_ACCESS_DEPTH_STENCIL_ATTACHMENT_WRITE_BIT	VK_ACCESS_TRANSFER_READ_BIT	VK_ACCESS_TRANSFER_WRITE_BIT	VK_ACCESS_HOST_READ_BIT	VK_ACCESS_HOST_WRITE_BIT	VK_ACCESS_MEMORY_READ_BIT	VK_ACCESS_MEMORY_WRITE_BIT
1 VK_PIPELINE_STAGE_TOP_OF_PIPE_BIT																	
2 VK_PIPELINE_STAGE_DRAW_INDIRECT_BIT	•																
3 VK_PIPELINE_STAGE_VERTEX_INPUT_BIT		•	•														
4 VK_PIPELINE_STAGE_VERTEX_SHADER_BIT				•													
5 VK_PIPELINE_STAGE_TESSELLATION_CONTROL_SHADER_BIT				•	•												
6 VK_PIPELINE_STAGE_TESSELLATION_EVALUATION_SHADER_BIT				•	•	•											
7 VK_PIPELINE_STAGE_GEOMETRY_SHADER_BIT				•	•	•											
8 VK_PIPELINE_STAGE_EARLY_FRAGMENT_TESTS_BIT										•	•						
9 VK_PIPELINE_STAGE_FRAGMENT_SHADER_BIT				•	•	•											
10 VK_PIPELINE_STAGE_LATE_FRAGMENT_TESTS_BIT										•	•						
11 VK_PIPELINE_STAGE_COLOR_ATTACHMENT_OUTPUT_BIT								•	•								
12 VK_PIPELINE_STAGE_BOTTOM_OF_PIPE_BIT																	
VK_PIPELINE_STAGE_COMPUTE_SHADER_BIT				•	•	•											
VK_PIPELINE_STAGE_TRANSFER_BIT												•	•				
VK_PIPELINE_STAGE_HOST_BIT														•	•		

Access Operations and what Pipeline Stages they can be used In

12

	1	2	3	4	5	6	7	8	9	10	11	12		
VK_PIPELINE_STAGE_TOP_OF_PIPE_BIT														
VK_PIPELINE_STAGE_DRAW_INDIRECT_BIT	•													
VK_PIPELINE_STAGE_VERTEX_INPUT_BIT		•	•											
VK_PIPELINE_STAGE_VERTEX_SHADER_BIT				•										
VK_PIPELINE_STAGE_TESSELLATION_CONTROL_SHADER_BIT				•	•									
VK_PIPELINE_STAGE_TESSELLATION_EVALUATION_SHADER_BIT				•	•	•								
VK_PIPELINE_STAGE_GEOMETRY_SHADER_BIT				•	•	•								
VK_PIPELINE_STAGE_EARLY_FRAGMENT_TESTS_BIT										•	•			
VK_PIPELINE_STAGE_FRAGMENT_SHADER_BIT				•	•	•								
VK_PIPELINE_STAGE_LATE_FRAGMENT_TESTS_BIT										•	•			
VK_PIPELINE_STAGE_COLOR_ATTACHMENT_OUTPUT_BIT										•	•			
VK_PIPELINE_STAGE_BOTTOM_OF_PIPE_BIT														
VK_PIPELINE_STAGE_COMPUTE_SHADER_BIT				•	•	•								
VK_PIPELINE_STAGE_TRANSFER_BIT												•	•	
VK_PIPELINE_STAGE_HOST_BIT													•	•

Example: Be sure we are done writing an output image before using it for something else

13

Stages

- VK_PIPELINE_STAGE_TOP_OF_PIPE_BIT
- VK_PIPELINE_STAGE_DRAW_INDIRECT_BIT
- VK_PIPELINE_STAGE_VERTEX_INPUT_BIT
- VK_PIPELINE_STAGE_VERTEX_SHADER_BIT
- VK_PIPELINE_STAGE_TESSELLATION_CONTROL_SHADER_BIT
- VK_PIPELINE_STAGE_TESSELLATION_EVALUATION_SHADER_BIT
- VK_PIPELINE_STAGE_GEOMETRY_SHADER_BIT
- VK_PIPELINE_STAGE_FRAGMENT_SHADER_BIT** ← **src**
- VK_PIPELINE_STAGE_EARLY_FRAGMENT_TESTS_BIT
- VK_PIPELINE_STAGE_LATE_FRAGMENT_TESTS_BIT
- VK_PIPELINE_STAGE_COLOR_ATTACHMENT_OUTPUT_BIT
- VK_PIPELINE_STAGE_COMPUTE_SHADER_BIT
- VK_PIPELINE_STAGE_TRANSFER_BIT
- VK_PIPELINE_STAGE_BOTTOM_OF_PIPE_BIT** ← **dst**
- VK_PIPELINE_STAGE_HOST_BIT
- VK_PIPELINE_STAGE_ALL_GRAPHICS_BIT
- VK_PIPELINE_STAGE_ALL_COMMANDS_BIT



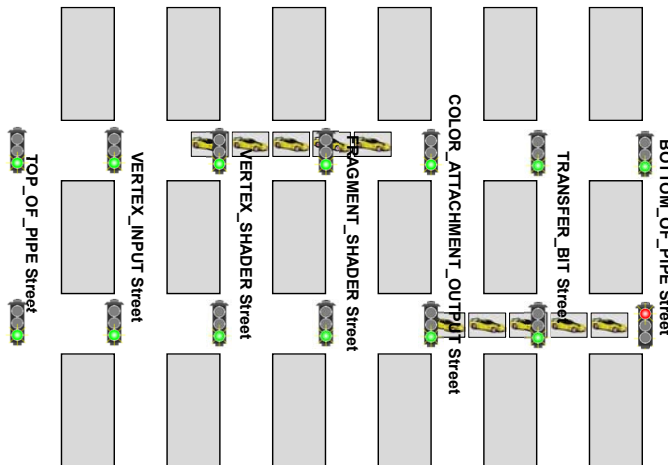
mjb - October 4 2018

The Scenario

14

src cars are generating the image

dst cars are doing something with that image



mjb - October 4 2018

Example: Don't read a buffer back to the host until a shader is done writing it

Stages

- VK_PIPELINE_STAGE_TOP_OF_PIPE_BIT
- VK_PIPELINE_STAGE_DRAW_INDIRECT_BIT
- VK_PIPELINE_STAGE_VERTEX_INPUT_BIT
- VK_PIPELINE_STAGE_VERTEX_SHADER_BIT** ← src
- VK_PIPELINE_STAGE_TESSELLATION_CONTROL_SHADER_BIT
- VK_PIPELINE_STAGE_TESSELLATION_EVALUATION_SHADER_BIT
- VK_PIPELINE_STAGE_GEOMETRY_SHADER_BIT
- VK_PIPELINE_STAGE_FRAGMENT_SHADER_BIT
- VK_PIPELINE_STAGE_EARLY_FRAGMENT_TESTS_BIT
- VK_PIPELINE_STAGE_LATE_FRAGMENT_TESTS_BIT
- VK_PIPELINE_STAGE_COLOR_ATTACHMENT_OUTPUT_BIT
- VK_PIPELINE_STAGE_COMPUTE_SHADER_BIT
- VK_PIPELINE_STAGE_TRANSFER_BIT
- VK_PIPELINE_STAGE_BOTTOM_OF_PIPE_BIT** ← dst
- VK_PIPELINE_STAGE_HOST_BIT
- VK_PIPELINE_STAGE_ALL_GRAPHICS_BIT
- VK_PIPELINE_STAGE_ALL_COMMANDS_BIT

Access types

- VK_ACCESS_INDIRECT_COMMAND_READ_BIT
- VK_ACCESS_INDEX_READ_BIT
- VK_ACCESS_VERTEX_ATTRIBUTE_READ_BIT
- VK_ACCESS_UNIFORM_READ_BIT
- VK_ACCESS_INPUT_ATTACHMENT_READ_BIT
- VK_ACCESS_SHADER_READ_BIT
- VK_ACCESS_SHADER_WRITE_BIT** ← src
- VK_ACCESS_COLOR_ATTACHMENT_READ_BIT
- VK_ACCESS_COLOR_ATTACHMENT_WRITE_BIT
- VK_ACCESS_DEPTH_STENCIL_ATTACHMENT_READ_BIT
- VK_ACCESS_DEPTH_STENCIL_ATTACHMENT_WRITE_BIT
- VK_ACCESS_TRANSFER_READ_BIT
- VK_ACCESS_TRANSFER_WRITE_BIT**
- VK_ACCESS_HOST_READ_BIT
- VK_ACCESS_HOST_WRITE_BIT
- VK_ACCESS_MEMORY_READ_BIT
- VK_ACCESS_MEMORY_WRITE_BIT

dst (no access setting needed)

mjb - October 4 2018

The Scenario

16

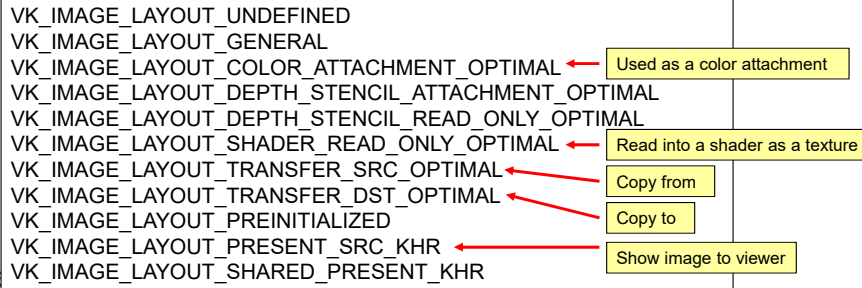
mjb - October 4 2018

VkImageLayout – How an Image gets Laid Out in Memory depends on how it will be Used

17

```

VkImageMemoryBarrier vimb {
    vimb.sType = VK_STRUCTURE_TYPE_IMAGE_MEMORY_BARRIER;
    vimb.pNext = nullptr;
    vimb.srcAccessMask = ??;
    vimb.dstAccessMask = ??;
    vimb.oldLayout = ??;
    vimb.newLayout = ??;
    vimb.srcQueueFamilyIndex = VK_QUEUE_FAMILY_IGNORED;
    vimb.dstQueueFamilyIndex = VK_QUEUE_FAMILY_IGNORED;
    vimb.image = ??;
    vimb.subresourceRange = vlsr;
}
    
```



Here, the use of vkCmdPipelineBarrier() is to simply change the layout of an image