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

- vkCmdSetDepthBias( commandBuffer, depthBiasConstantFactor, depthBiasClamp, depthBiasSlopeFactor );
- vkCmdSetBlendConstants( commandBuffer, blendConstants[4] );
- vkCmdResetEvent( commandBuffer, event, stageMask );
- vkCmdPushDescriptorSetWithTemplateKHR( commandBuffer, descriptorUpdateTemplate, layout, set, pData );
- vkCmdWriteTimestamp( commandBuffer, pipelineStage, queryPool, query );
- vkCmdWaitEvents( commandBuffer, eventCount, pEvents, srcStageMask, dstStageMask, memoryBarrierCount, pMemoryBarriers, VK_DEPENDENCY_BY_REGION_BIT, dstStageMask );
- vkCmdSetViewportWScalingNV( commandBuffer, firstViewport, viewportCount, pViewportWScalings );
- vkCmdSetViewport( commandBuffer, firstViewport, viewportCount, pViewports );
- vkCmdSetStencilWriteMask( commandBuffer, faceMask, writeMask );
- vkCmdSetStencilReference( commandBuffer, faceMask, reference );
- vkCmdSetStencilCompareMask( commandBuffer, faceMask, compareMask );
- vkCmdSetScissor( commandBuffer, firstScissor, scissorCount, pScissors );
- vkCmdSetDiscardRectangleEXT( commandBuffer, firstDiscardRectangle, discardRectangleCount, pDiscardRectangles );
- vkCmdSetDeviceMaskKHX( commandBuffer, deviceMask );
- vkCmdSetDepthBounds( commandBuffer, minDepthBounds, maxDepthBounds );
- vkCmdPushConstants( commandBuffer, layout, stageFlags, offset, size, pValues );
- vkCmdProcessCommandsNVX( commandBuffer, pProcessCommandsInfo );
- vkCmdNextSubpass( commandBuffer, contents );
- vkCmdFillBuffer( commandBuffer, dstBuffer, dstOffset, size, data );
- These are the Commands that can be entered into the Command Buffer, II

- vkCmdPipelineBarrier( commandBuffer, srcStageMask, dstStageMask, memoryBarrierCount, pBufferMemoryBarriers, imageMemoryBarrierCount, pImageMemoryBarriers );
- vkCmdCopyQueryPoolResults( commandBuffer, flags );
- vkCmdCopyImageToBuffer( commandBuffer, pRegions );
- vkCmdCopyImage( commandBuffer, pRegions );
- vkCmdCopyBufferToImage( commandBuffer, pRegions );
- vkCmdCopyBuffer( commandBuffer, pRegions );
- vkCmdClearColorImage( commandBuffer, pRanges );
- vkCmdClearAttachments( commandBuffer, attachmentCount, const pRects );
- vkCmdBlitImage( commandBuffer, filter );
- vkCmdBindVertexBuffers( commandBuffer, firstBinding, bindingCount, const pOffsets );
- vkCmdBindPipeline( commandBuffer, pipeline );
- vkCmdBindIndexBuffer( commandBuffer, indexType );
- vkCmdBindDescriptorSets( commandBuffer, pDynamicOffsets );
- vkCmdBeginRenderPass( commandBuffer, const contents );
- vkCmdBeginQuery( commandBuffer, flags );

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
The Scenario

1. The cross-streets are named after pipeline stages
2. All traffic lights start out green
3. There are special sensors at all intersections that will know when any car in the src group is in that intersection
4. There are connections from those sensors to the traffic lights so that when any car in the src group is in the intersection, the proper dst traffic light will be turned red
5. When the last car in the src group completely makes it through its intersection, the proper dst traffic light is 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 red light), (3) the dst cars stop at the red light, (4) the src intersection clears, (5) all lights are now green, (6) the dst cars continue.

Pipeline Stages and what Access Operations are Allowed

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

Pipeline Stage Masks – Access Operations and what Pipeline Stages they can be used In

Access Masks – What are you Interested in Generating or Consuming this Memory for?

Access Operations and what Pipeline Stages they can be used In
Example: Be sure we are done writing an output image before using it for something else.

```muşahhur
vkImageMemoryBarrier
```

Example: Don’t read a buffer back to the host until a shader is doing work with that image.

```muşahhur
vkImageMemoryBarrier
```

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