From the Command Buffer Notes:

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

- `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);`
- `Pipeline Barriers` (with various functions listed)
- `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);`
- `vkCmdDispatch(commandBuffer, groupCountX, groupCountY, groupCountZ);`
- `vkCmdDrawIndexed(commandBuffer, indexCount, instanceCount, firstIndex, int32_t vertexOffset, firstInstance);`
- `vkCmdDrawIndexedIndirectCountAMD(commandBuffer, stride);`
- `vkCmdDrawIndirect(commandBuffer, stride);`
- `vkCmdDrawIndirectCountAMD(commandBuffer, stride);`
- `vkCmdEndQuery(commandBuffer, query);`
- `vkCmdEndRenderPass(commandBuffer);`
- `vkCmdExecuteCommands(commandBuffer, commandBufferCount, const pCommandBuffers);`
- `vkCmdNextSubpass(commandBuffer, contents);`
- `vkCmdPipelineBarrier(commandBuffer, srcStageMask, dstStageMask, dependencyFlags, memoryBarrierCount, VkMemoryBarrier* pMemoryBarriers, bufferMemoryBarrierCount, pBufferMemoryBarriers, imageMemoryBarrierCount, pImageMemoryBarriers);`
- `vkCmdProcessCommandsNVX(commandBuffer, pProcessCommandsInfo);`
- `vkCmdPushDescriptorSetKHR(commandBuffer, pipelineBindPoint, layout, set, descriptorWriteCount, pDescriptorWrites);`
- `vkCmdPushDescriptorSetWithTemplateKHR(commandBuffer, descriptorUpdateTemplate, layout, set, pData);`
- `vkCmdReserveSpaceForCommandsNVX(commandBuffer, pReserveSpaceInfo);`
- `vkCmdResetDeviceMaskKHX(commandBuffer, deviceMask);`
- `vkCmdSetDeviceMaskKHX(commandBuffer, deviceMask);`
- `vkCmdSetDiscardRectangleEXT(commandBuffer, firstDiscardRectangle, discardRectangleCount, pDiscardRectangles);`
- `vkCmdSetEvent(commandBuffer, event, stageMask);`
- `vkCmdSetLineWidth(commandBuffer, lineWidth);`
- `vkCmdSetScissor(commandBuffer, firstScissor, scissorCount, pScissors);`
- `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, pImageMemoryBarriers);`
- `vkCmdWriteTimestamp(commandBuffer, pipelineStage, queryPool, query);`

We want all of these commands to be able to run "flat-out", but, if we do that, surely there will be race conditions!

The Scenario

1. All traffic lights start green ("we want all of these commands to be able to run flat-out")
2. There are special sensors at all intersections that get triggered when the first car in a group enters that intersection
3. There are special connections from those sensors to the traffic lights so that when a sensor gets triggered, an arbitrary traffic light can turn red
4. When an entire group of cars completely makes its way through an intersection, that sensor can use that same connection to turn the light back to green
5. The src cars get released first, immediately followed by the trigger/connection rules, immediately followed by the dst cars
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()` Function Call

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

Pipeline Stage Flags – Where in the Pipeline is this Memory being Accessed?

- 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

Pipeline Stages

- Vertex Shader
- Primitive Assembly
- Tessellation Control Shader
- Tessellation Primitive Generator
- Tessellation Evaluation Shader
- Primitive Assembly
- Geometry Shader
- Primitive Assembly
- Rasterizer
- Fragment Shader

Memory Barrier

```
VkMemoryBarrier vmb;
vmb.sType = VK_STRUCTURE_TYPE_MEMORY_BARRIER;
vmb.pNext = nullptr;
vmb.srcAccessMask = ??;
vmb.dstAccessMask = ??;
```
### Access Flags – What Operation are you Interested in this Memory for?

- 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_ACCESSCOLOR_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

### Pipeline Stage Flags and Access Flags Together

- 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

### Example: Don’t overwrite a texture image until we are done using it

```plaintext
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
```

### The Scenario

- **src cars**
- **dst cars**

**src**
```
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
```

**dst**
```
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
```
Example: Don’t read a buffer back to the host until a shader is done writing.

Image Memory Barrier:

```c
VkImageSubresourceRange visr;
visr.aspectMask = VK_IMAGE_ASPECT_COLOR_BIT;
visr.baseMipLevel = 0;
visr.levelCount = 1;
visr.baseArrayLayer = 0;
visr.layerCount = 1;
```

Buffer Memory Barrier:

```c
VkBufferMemoryBarrier vbmb;
vbmb.sType = VK_STRUCTURE_TYPE_BUFFER_MEMORY_BARRIER;
vbmb.pNext = nullptr;
vbmb.srcAccessMask = ??;
vbmb.dstAccessMask = ??;
vbmb.srcQueueFamilyIndex = VK_QUEUE_FAMILY_IGNORED;
vbmb.dstQueueFamilyIndex = VK_QUEUE_FAMILY_IGNORED;
vbmb.buffer = ??;
vbmb.offset = 0;
vbmb.size = VK_WHOLE_SIZE;
```
### VkImageLayout – How an Image gets Laid Out in Memory depends on How it will be Used

<table>
<thead>
<tr>
<th>VkImageLayout</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>VK_IMAGE_LAYOUT_UNDEFINED</td>
<td>Used as a color attachment</td>
</tr>
<tr>
<td>VK_IMAGE_LAYOUT_GENERAL</td>
<td>Read into a shader as a texture</td>
</tr>
<tr>
<td>VK_IMAGE_LAYOUT_COLOR_ATTACHMENT_OPTIMAL</td>
<td>Copy from</td>
</tr>
<tr>
<td>VK_IMAGE_LAYOUT_DEPTH_STENCIL_ATTACHMENT_OPTIMAL</td>
<td>Copy to</td>
</tr>
<tr>
<td>VK_IMAGE_LAYOUT_DEPTH_STENCIL_READ_ONLY_OPTIMAL</td>
<td>Show image to viewer</td>
</tr>
<tr>
<td>VK_IMAGE_LAYOUT_SHADER_READ_ONLY_OPTIMAL</td>
<td></td>
</tr>
<tr>
<td>VK_IMAGE_LAYOUT_TRANSFER_SRC_OPTIMAL</td>
<td></td>
</tr>
<tr>
<td>VK_IMAGE_LAYOUT_TRANSFER_DST_OPTIMAL</td>
<td></td>
</tr>
<tr>
<td>VK_IMAGE_LAYOUT_PREINITIALIZED</td>
<td></td>
</tr>
<tr>
<td>VK_IMAGE_LAYOUT_PRESENT_SRC_KHR</td>
<td></td>
</tr>
<tr>
<td>VK_IMAGE_LAYOUT_SHARED_PRESENT_KHR</td>
<td></td>
</tr>
</tbody>
</table>

One use of `vkCmdPipelineBarrier()` is to simply change the layout of an image.