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

From the Command Buffer Notes:

These are the Commands that can be entered into the Command Buffer:

- vkCmdBeginQuery(commandBuffer, flags);
- vkCmdBeginRenderPass(commandBuffer, const contents);
- vkCmdBindDescriptorSets(commandBuffer, pDynamicOffsets);
- vkCmdBindIndirectBuffer(commandBuffer, indexType);
- vkCmdBindPipeline(commandBuffer, pipeline);
- vkCmdBindVertextBuffers(commandBuffer, firstBinding, bindingCount, const pOffsets);
- vkCmdBindShaderImage(commandBuffer, texture, count, const pImages);
- vkCmdBindShaderImage(commandBuffer, attachmentCount, const pAttachments);
- 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);
- vkCmdDispatch(commandBuffer, groupCountX, groupCountY, groupCountZ);
- vkCmdDispatchIndirect(commandBuffer, offset);
- vkCmdDraw(commandBuffer, vertexCount, instanceCount, firstVertex, firstInstance);
- vkCmdDrawIndexed(commandBuffer, indexCount, instanceCount, firstIndex, int32_t vertexOffset, firstInstance);
- vkCmdDrawIndirect(commandBuffer, stride);
- vkCmdDrawIndirectCountAMD(commandBuffer, stride);
- vkCmdEndQuery(commandBuffer, query);
- vkCmdEndRenderPass(commandBuffer);
- vkCmdExecuteCommands(commandBuffer, commandBufferCount, const pCommandBuffers);
### From the Command Buffer Notes:

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

<table>
<thead>
<tr>
<th>Command Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>vkCmdFillBuffer</td>
<td>Fill a buffer with data</td>
</tr>
<tr>
<td>vkCmdNextSubpass</td>
<td>Transition to the next subpass</td>
</tr>
<tr>
<td>vkCmdPipelineBarrier</td>
<td>Synchronize the command buffer with memory barriers</td>
</tr>
<tr>
<td>vkCmdProcessCommandNV</td>
<td>Process commands in the command buffer</td>
</tr>
<tr>
<td>vkCmdPushConstants</td>
<td>Set constants for the current pipeline stage</td>
</tr>
<tr>
<td>vkCmdPipelineBarrierKHR</td>
<td>Synchronize the command buffer with memory barriers</td>
</tr>
<tr>
<td>vkCmdUpdateBuffer</td>
<td>Update a buffer with data</td>
</tr>
<tr>
<td>vkCmdWaitEvents</td>
<td>Wait for events to complete</td>
</tr>
<tr>
<td>vkCmdWriteTimestamp</td>
<td>Write a timestamp to a query pool</td>
</tr>
</tbody>
</table>

---

### 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
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
);

The Scenario

TOP_OF_PIPE Street
VERTEX_INPUT Street
VERTEX_SHADER Street
BOTTOM_OF_PIPE Street
TRANSFER_BIT Street
COLOR_ATTACHMENT_OUTPUT Street

```
src cars
```
```
dst cars
```
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 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 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 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)

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
Pipeline 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
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

Access Masks –
What are you Interested in Generating or Consuming 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_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
### Pipeline Stages and what Access Operations can Happen There

<table>
<thead>
<tr>
<th>Pipeline Stage</th>
<th>Access Bit</th>
</tr>
</thead>
<tbody>
<tr>
<td>VK_PIPELINE_STAGE_TOP_OF_PIPE_BIT</td>
<td>VK_ACCESS_INDIRECT_COMMAND_READ_BIT</td>
</tr>
<tr>
<td>VK_PIPELINE_STAGE_DRAW_INDIRECT_BIT</td>
<td>VK_ACCESS_INDEX_READ_BIT</td>
</tr>
<tr>
<td>VK_PIPELINE_STAGE_VERTEX_INPUT_BIT</td>
<td>VK_ACCESS_VERTEX_ATTRIBUTE_READ_BIT</td>
</tr>
<tr>
<td>VK_PIPELINE_STAGE_VERTEX_SHADER_BIT</td>
<td>VK_ACCESS_UNIFORM_READ_BIT</td>
</tr>
<tr>
<td>VK_PIPELINE_STAGE_TESSELLATION_CONTROL_SHADER_BIT</td>
<td>VK_ACCESS_INPUT_ATTACHMENT_READ_BIT</td>
</tr>
<tr>
<td>VK_PIPELINE_STAGE_TESSELLATION_EVALUATION_SHADER_BIT</td>
<td>VK_ACCESS_SHADER_READ_BIT</td>
</tr>
<tr>
<td>VK_PIPELINE_STAGE_GEOMETRY_SHADER_BIT</td>
<td>VK_ACCESS_SHADER_WRITE_BIT</td>
</tr>
<tr>
<td>VK_PIPELINE_STAGE_COLOR_ATTACHMENT_OUTPUT_BIT</td>
<td>VK_ACCESS_COLOR_ATTACHMENT_WRITE_BIT</td>
</tr>
<tr>
<td>VK_PIPELINE_STAGE_BOTTOM_OF_PIPE_BIT</td>
<td>VK_ACCESS_DEPTH_STENCIL_ATTACHMENT_WRITE_BIT</td>
</tr>
</tbody>
</table>

### Access Operations and what Pipeline Stages they can be used In

<table>
<thead>
<tr>
<th>Access Bit</th>
<th>Pipeline Stage</th>
</tr>
</thead>
<tbody>
<tr>
<td>VK_ACCESS_INDIRECT_COMMAND_READ_BIT</td>
<td>VK_PIPELINE_STAGE_TOP_OF_PIPE_BIT</td>
</tr>
<tr>
<td>VK_ACCESS_INDEX_READ_BIT</td>
<td>VK_PIPELINE_STAGE_DRAW_INDIRECT_BIT</td>
</tr>
<tr>
<td>VK_ACCESS_VERTEX_ATTRIBUTE_READ_BIT</td>
<td>VK_PIPELINE_STAGE_VERTEX_INPUT_BIT</td>
</tr>
<tr>
<td>VK_ACCESS_UNIFORM_READ_BIT</td>
<td>VK_PIPELINE_STAGE_VERTEX_SHADER_BIT</td>
</tr>
<tr>
<td>VK_ACCESS_INPUT_ATTACHMENT_READ_BIT</td>
<td>VK_PIPELINE_STAGE_TESSELLATION_CONTROL_SHADER_BIT</td>
</tr>
<tr>
<td>VK_ACCESS_SHADER_READ_BIT</td>
<td>VK_PIPELINE_STAGE_TESSELLATION_EVALUATION_SHADER_BIT</td>
</tr>
<tr>
<td>VK_ACCESS_SHADER_WRITE_BIT</td>
<td>VK_PIPELINE_STAGE_GEOMETRY_SHADER_BIT</td>
</tr>
<tr>
<td>VK_ACCESS_COLOR_ATTACHMENT_READ_BIT</td>
<td>VK_PIPELINE_STAGE_FRAGMENT_COLOR_ATTACHMENT_OUTPUT_BIT</td>
</tr>
<tr>
<td>VK_ACCESS_COLOR_ATTACHMENT_WRITE_BIT</td>
<td>VK_PIPELINE_STAGE_FRAGMENT_DEPTH_ATTACHMENT_WRITE_BIT</td>
</tr>
<tr>
<td>VK_ACCESS_DEPTH_STENCIL_ATTACHMENT_READ_BIT</td>
<td>VK_PIPELINE_STAGE_FRAGMENT_STENCIL_ATTACHMENT_READ_BIT</td>
</tr>
<tr>
<td>VK_ACCESS_DEPTH_STENCIL_ATTACHMENT_WRITE_BIT</td>
<td>VK_PIPELINE_STAGE_FRAGMENT_STENCIL_ATTACHMENT_WRITE_BIT</td>
</tr>
<tr>
<td>VK_ACCESS_TRANSFER_READ_BIT</td>
<td>VK_PIPELINE_STAGE_BOTTOM_OF_PIPE_BIT</td>
</tr>
<tr>
<td>VK_ACCESS_TRANSFER_WRITE_BIT</td>
<td>VK_PIPELINE_STAGE_HOST_BIT</td>
</tr>
</tbody>
</table>
### Example: Be sure we are done writing an output image before using it for something else

<table>
<thead>
<tr>
<th>Stages</th>
<th>Access types</th>
</tr>
</thead>
<tbody>
<tr>
<td>VK_PIPELINE_STAGE_TOP_OF_PIPE_BIT</td>
<td>VK_ACCESS_INDIRECT_COMMAND_READ_BIT</td>
</tr>
<tr>
<td>VK_PIPELINE_STAGE_DRAW_INDIRECT_BIT</td>
<td>VK_ACCESS_INDEX_READ_BIT</td>
</tr>
<tr>
<td>VK_PIPELINE_STAGE_VERTEX_INPUT_BIT</td>
<td>VK_ACCESS_VERTEX_ATTRIBUTE_READ_BIT</td>
</tr>
<tr>
<td>VK_PIPELINE_STAGE_TESSELLATION_CONTROL_SHADER_BIT</td>
<td>VK_ACCESS_UNIFORM_READ_BIT</td>
</tr>
<tr>
<td>VK_PIPELINE_STAGE_TESSELLATION_EVALUATION_SHADER_BIT</td>
<td>VK_ACCESS_INPUT_ATTACHMENT_READ_BIT</td>
</tr>
<tr>
<td>VK_PIPELINE_STAGE_FRAGMENT_SHADER_BIT</td>
<td>VK_ACCESS_SHADER_READ_BIT</td>
</tr>
<tr>
<td>VK_PIPELINE_STAGE_EARLY_FRAGMENT_TESTS_BIT</td>
<td>VK_ACCESS_SHADER_WRITE_BIT</td>
</tr>
<tr>
<td>VK_PIPELINE_STAGE_LATE_FRAGMENT_TESTS_BIT</td>
<td>VK_ACCESS_COLOR_ATTACHMENT_WRITE_BIT</td>
</tr>
<tr>
<td>VK_PIPELINE_STAGE_COLOR_ATTACHMENT_OUTPUT_BIT</td>
<td>VK_ACCESS_DEPTH_STENCIL_ATTACHMENT_READ_BIT</td>
</tr>
<tr>
<td>VK_PIPELINE_STAGE_GEOMETRY_SHADER_BIT</td>
<td>VK_ACCESS_DEPTH_STENCIL_ATTACHMENT_WRITE_BIT</td>
</tr>
<tr>
<td>VK_PIPELINE_STAGE_FRAGMENT_SHADER_BIT</td>
<td>VK_ACCESS_TRANSFER_READ_BIT</td>
</tr>
<tr>
<td>VK_PIPELINE_STAGE_COLOR_ATTACHMENT_OUTPUT_BIT</td>
<td>VK_ACCESS_TRANSFER_WRITE_BIT</td>
</tr>
<tr>
<td>VK_PIPELINE_STAGE_ALL_GRAPHICS_BIT</td>
<td>VK_ACCESS_HOST_READ_BIT</td>
</tr>
<tr>
<td>VK_PIPELINE_STAGE_ALL_COMMANDS_BIT</td>
<td>VK_ACCESS_HOST_WRITE_BIT</td>
</tr>
<tr>
<td>VK_PIPELINE_STAGE_BOTTOM_OF_PIPE_BIT</td>
<td>VK_ACCESS_MEMORY_READ_BIT</td>
</tr>
<tr>
<td>VK_PIPELINE_STAGE_EARLY_FRAGMENT_TESTS_BIT</td>
<td>VK_ACCESS_MEMORY_WRITE_BIT</td>
</tr>
</tbody>
</table>

#### The Scenario

- **src** cars are generating the image
- **dst** cars are doing something with that image
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
- 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

**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_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

The Scenario

- **src** cars
  - TOP_OF_PIPE
  - VERTEX_SHADER
  - FRAGMENT_SHADER
  - TRANSFER_BIT
  - BOTTOM_OF_PIPE

- **dst** cars
  - TOP_OF_PIPE
  - VERTEX_SHADER
  - FRAGMENT_SHADER
  - TRANSFER_BIT
  - BOTTOM_OF_PIPE
 VkImageLayout – How an Image gets Laid Out in Memory depends on how it will be Used

<table>
<thead>
<tr>
<th>VkImageMemoryBarrier</th>
<th>vimb</th>
</tr>
</thead>
<tbody>
<tr>
<td>vimb.sType = VK_STRUCTURE_TYPE_IMAGE_MEMORY_BARRIER;</td>
<td></td>
</tr>
<tr>
<td>vimb.pNext = nullptr;</td>
<td></td>
</tr>
<tr>
<td>vimb.srcAccessMask = ??;</td>
<td></td>
</tr>
<tr>
<td>vimb.dstAccessMask = ??;</td>
<td></td>
</tr>
<tr>
<td>vimb.oldLayout = ??;</td>
<td></td>
</tr>
<tr>
<td>vimb.newLayout = ??;</td>
<td></td>
</tr>
<tr>
<td>vimb.srcQueueFamilyIndex = VK_QUEUE_FAMILY_IGNORED;</td>
<td></td>
</tr>
<tr>
<td>vimb.dstQueueFamilyIndex = VK_QUEUE_FAMILY_IGNORED;</td>
<td></td>
</tr>
<tr>
<td>vimb.image = ??;</td>
<td></td>
</tr>
<tr>
<td>vimb.subresourceRange = visr;</td>
<td></td>
</tr>
</tbody>
</table>

VK_IMAGE_LAYOUT_UNDEFINED
VK_IMAGE_LAYOUT_COLOR_ATTACHMENT_OPTIMAL Used as a color attachment
VK_IMAGE_LAYOUT_DEPTH_STENCIL_ATTACHMENT_OPTIMAL
VK_IMAGE_LAYOUT_DEPTH_STENCIL_READ_ONLY_OPTIMAL Read into a shader as a texture
VK_IMAGE_LAYOUT_SHADER_READ_ONLY_OPTIMAL Copy from
VK_IMAGE_LAYOUT_TRANSFER_SRC_OPTIMAL Copy to
VK_IMAGE_LAYOUT_TRANSFER_DST_OPTIMAL
VK_IMAGE_LAYOUT_PREINITIALIZED
VK_IMAGE_LAYOUT_PRESENT_SRC_KHR Show image to viewer
VK_IMAGE_LAYOUT_SHARED_PRESENT_KHR

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