# Vulkan Syncronization Notes

[https://github.com/KhronosGroup/Vulkan-Docs/wiki/Synchronization-Examples]

## General

A ‘vkCmdPipelineBarrier’ covers all resources globally.

```
const VkMemoryBarrier mem_barrier = {
    .sType         = VK_STRUCTURE_TYPE_MEMORY_BARRIER,
    .pNext         = NULL,
    .srcAccessMask = VkAccessFlagBits,
    .dstAccessMask = VkAccessFlagBits,
};

const VkImageMemoryBarrier img_barrier = {
    .sType               = VK_STRUCTURE_TYPE_IMAGE_MEMORY_BARRIER,
    .pNext               = NULL,
    .srcAccessMask       = VkAccessFlagBits,
    .dstAccessMask       = VkAccessFlagBits,
    .oldLayout           = VkImageLayout,
    .newLayout           = VkImageLayout,
    .srcQueueFamilyIndex = (uint32_t)VK_QUEUE_FAMILY_IGNORED or a queue index,
    .dstQueueFamilyIndex = (uint32_t)VK_QUEUE_FAMILY_IGNORED or a queue index,
    .image               = VkImage,
    .subresourceRange = {
        .aspectMask     = VkImageAspectFlagBits,
        .baseMipLevel   = uint32_t,
        .levelCount     = uint32_t,
        .baseArrayLayer = uint32_t,
        .layerCount     = uint32_t,
    }
};

const VkSubpassDependency sub_pass_dep = {
    .srcSubpass      = uint32_t,
    .dstSubpass      = uint32_t,
    .srcStageMask    = VkPipelineStageFlagBits,
    .dstStageMask    = VkPipelineStageFlagBits,
    .srcAccessMask   = VkAccessFlagBits,
    .dstAccessMask   = VkAccessFlagBits,
    .dependencyFlags = VK_DEPENDENCY_BY_REGION_BIT,
};

'VK_DEPENDENCY_BY_REGION_BIT' means that you only read / write pixels you own in a single sa
```

```
This would not work if you had a blur shader that read outside of that area.

const VkBufferMemoryBarrier buf_mem_barrier = {
    .sType               = VK_STRUCTURE_TYPE_BUFFER_MEMORY_BARRIER,
    .pNext               = NULL,
    .srcAccessMask       = VkAccessFlagBits,
    .dstAccessMask       = VkAccessFlagBits,
    .srcQueueFamilyIndex = uint32_t,
    .dstQueueFamilyIndex = uint32_t,
    .buffer              = VkBuffer,
    .offset              = VkDeviceSize,
    .size                = VkDeviceSize,
};
```

## Compute -> Compute

### Write -> Read

Using a "vkCmdPipelineBarrier" inbetween two 'vkCmdDispatch' protect against read after write.

'VK_PIPELINE_STAGE_COMPUTE_SHADER_BIT' for both src and dst stage. 'VK_ACCESS_SHADER_WRITE_BIT' for srcAccessMask 'VK_ACCESS_SHADER_READ_BIT' for dstAccessMask

### Read -> Write

A simpler execution dependency solves the write after read issue. 'vkCmd-PipelineBarrier' with no memory barrier.

## Compute -> Graphics

### Write (C) -> Read (G)

Same as Compute -> Compute for this situation.

### Write (C) -> Read (G) -> Read (C)

Nearly the same but adds more flags thus batching rather than using two barriers. Or the 'dstStageMask' and 'dstAccessMask' so that it includes both of the following ops.

### Image based writes and reads

Use 'VkImageMemoryBarrier' with a correct layout transition. This is only if the Draw wants to view it as an image other wise mem_bariier is fine.

## Graphics -> Compute

Use a ‘VkImageMemoryBarrier’ with the correct stages and access masks. Super simple.

## Graphics -> Graphics

Use a ‘VkSubpassDependency’ if you can. “VkAttachmentDescription” does implicit layout transitions.

Otherwise use a ‘VkImageMemoryBarrier’.

### WAR Hazard

You would normally only need an execution dep but since you need a layout transition then you need an ‘VkImageMemoryBarrier’ with an empty ‘srcAccess-Mask’.

### WAW Hazard (RP -> RP reusing the same depth buffer)

Always needs a memory dep. Needs use of ‘VK_SUBPASS_EXTERNAL’ because of the automatic transition. Use pipeline stages with the gramment tests as those use the depth buffer. .dependencyFlags should be 0.

## Memory Transfer

### Staging buffer -> Device Local Memory

Start with a ‘vkCmdCopyBuffer’. Then:

If the queue is unified a normal ‘vkCmdPipelineBarrier’ is fine.

Otherwise: A ‘VkBufferMemoryBarrier’ is needed. End and submit to the queue. (Must have a semaphore the gfx queue waints on) Begin commands Another ‘VkBufferMemoryBarrier’ with the cirrect dst Access. End cm and submit to gfx queue.

> Images have the same story except they can use ‘VkImageMemory-Barrier’ as they also have the family queue indices field. (unified uses: VK_QUEUE_FAMILY_IGNORED)