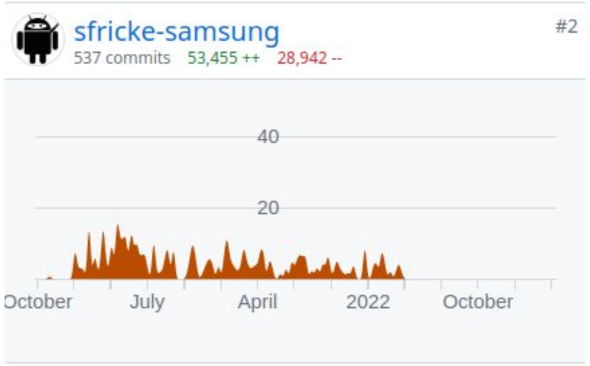
Using Vulkan Validation Effectively

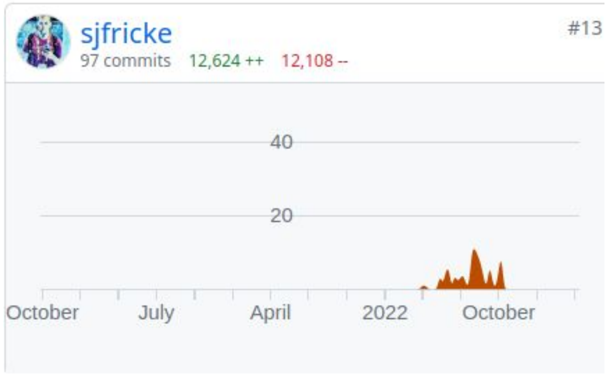
Spencer Fricke
LunarG, Inc.

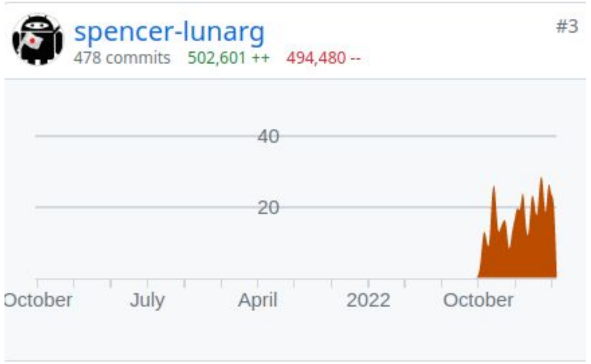
Presented at the May Khronos DevDay in Osaka Japan

Who is Spencer

- Have been working on Validation Layers for about 3 years now
- Currently main task at LunarG

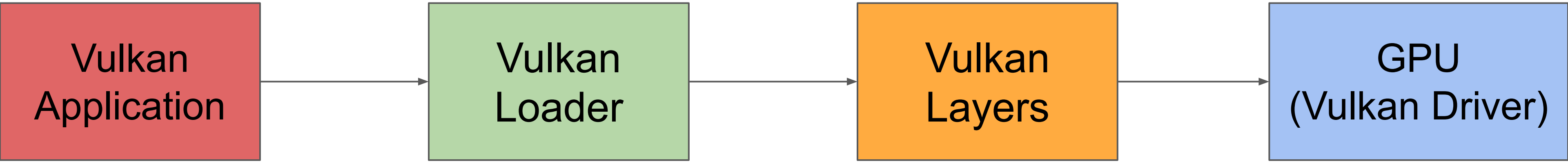






What is a Vulkan Layer

- A shared library
- It is in between the Loader and Driver



What is the Vulkan Validation Layer?

- Does the error checking for Vulkan
- Validation during development only
  - No validation overhead in released applications

Why the Vulkan Validation Layer?

- OpenGL had many error code checks that drivers had to implement
- Checks always enabled in drivers
    - useless CPU overhead
- Most checking was similar in all drivers (duplicated effort)

What ~~is~~ are the Vulkan Validation Layers?

- Only one layer
    - Common mistake
- When first created, were many smaller layers
- Realized there was a lot of duplicate code
- Have settings to toggle objects of the layer now

What is Valid Usage

- Valid Usage = **VU**
    - “set of conditions that must be met in order to achieve well-defined run-time behavior in an application.”
- Rules in the spec that describe what is illegal
- The driver assumes the application provides valid data
- If a VU is broken, it is **undefined behavior**
    - (and everything following it)

Undefined Behavior

- … App might work fine
- … GPU might hang
- … Computer might blow up!
- Anything is possible

Raph Levien's blog

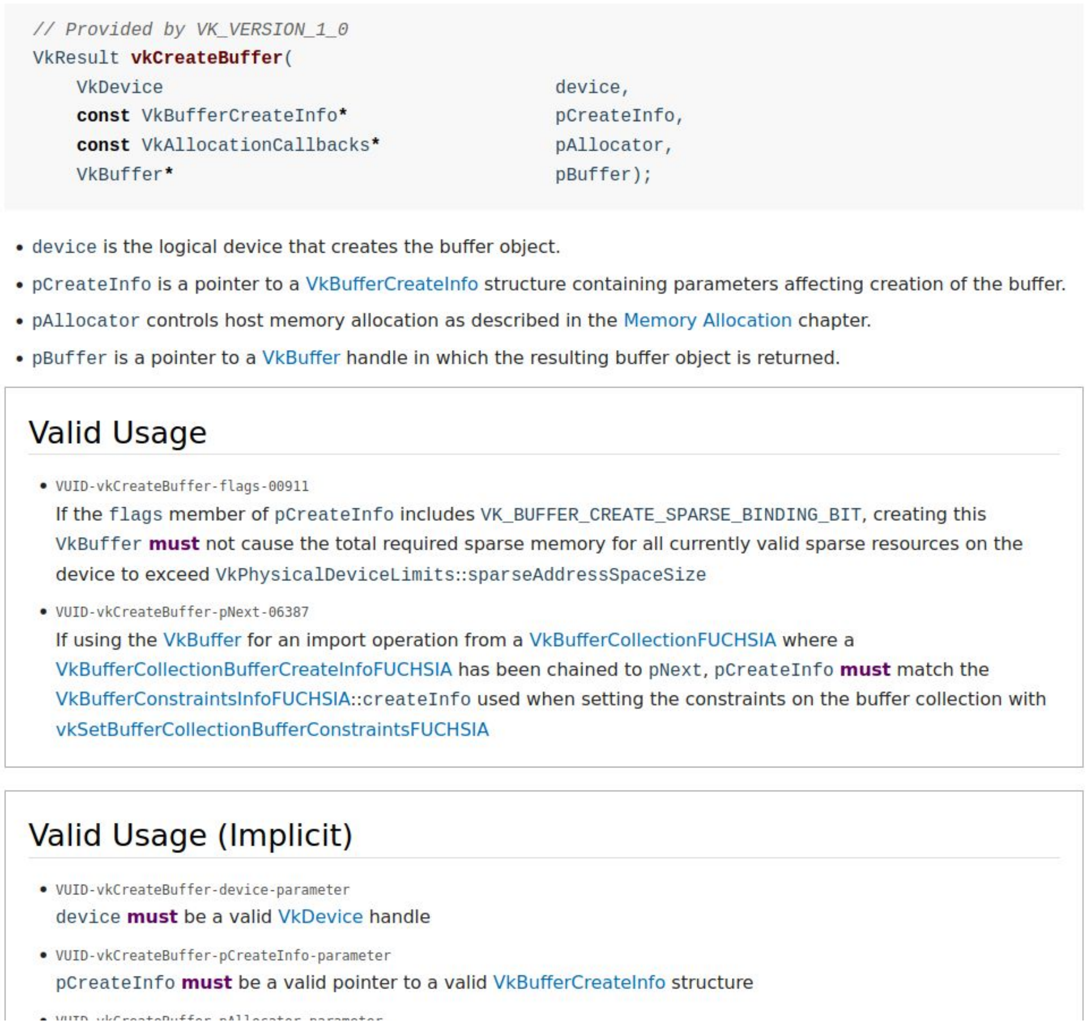
About

With Undefined Behavior, Anything is Possible

Aug 17, 2018

VUID

- **V**alid **U**sage **ID**
- Unique ID to map each error back to the spec
- Automatically generated number when spec is released
- Few UNASSIGNED VUIDs
  - Almost all gone now!



```
// Provided by VK_VERSION_1_0
VkResult vkCreateBuffer(
    VkDevice                                    device,
    const VkBufferCreateInfo*                   pCreateInfo,
    const VkAllocationCallbacks*                pAllocator,
    VkBuffer*                                   pBuffer);
```

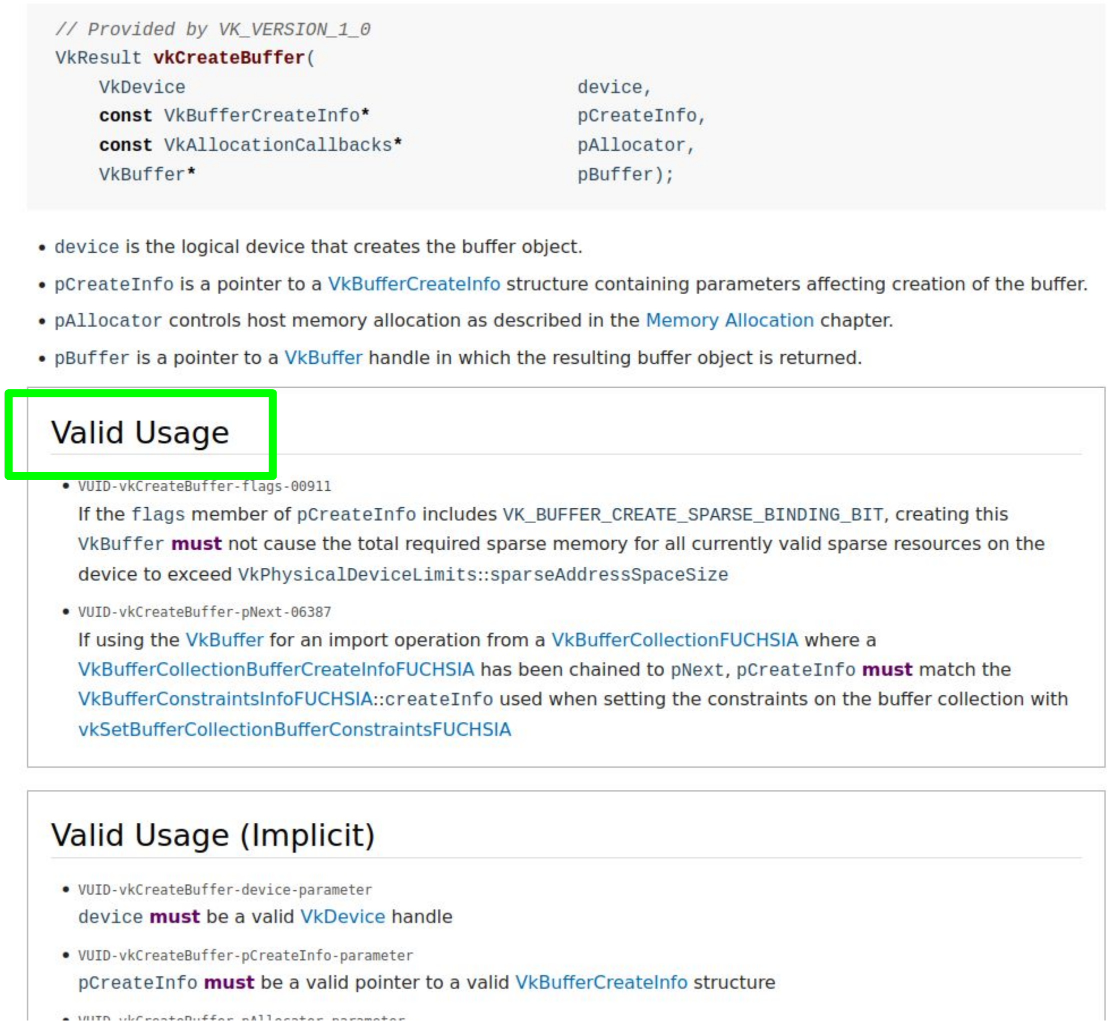
- `device` is the logical device that creates the buffer object.
- `pCreateInfo` is a pointer to a VkBufferCreateInfo structure containing parameters affecting creation of the buffer.
- `pAllocator` controls host memory allocation as described in the Memory Allocation chapter.
- `pBuffer` is a pointer to a VkBuffer handle in which the resulting buffer object is returned.

Valid Usage

- `VUID-vkCreateBuffer-flags-00911`
  If the `flags` member of `pCreateInfo` includes `VK_BUFFER_CREATE_SPARSE_BINDING_BIT`, creating this `VkBuffer` **must** not cause the total required sparse memory for all currently valid sparse resources on the device to exceed `VkPhysicalDeviceLimits::sparseAddressSpaceSize`
- `VUID-vkCreateBuffer-pNext-06387`
  If using the VkBuffer for an import operation from a VkBufferCollectionFUCHSIA where a VkBufferCollectionBufferCreateInfoFUCHSIA has been chained to `pNext`, `pCreateInfo` **must** match the VkBufferConstraintsInfoFUCHSIA::`createInfo` used when setting the constraints on the buffer collection with vkSetBufferCollectionBufferConstraintsFUCHSIA

Valid Usage (Implicit)

- `VUID-vkCreateBuffer-device-parameter`
  `device` **must** be a valid VkDevice handle
- `VUID-vkCreateBuffer-pCreateInfo-parameter`
  `pCreateInfo` **must** be a valid pointer to a valid VkBufferCreateInfo structure
- `VUID-vkCreateBuffer-pAllocator-parameter`



```
// Provided by VK_VERSION_1_0
VkResult vkCreateBuffer(
    VkDevice                                    device,
    const VkBufferCreateInfo*                   pCreateInfo,
    const VkAllocationCallbacks*                pAllocator,
    VkBuffer*                                   pBuffer);
```

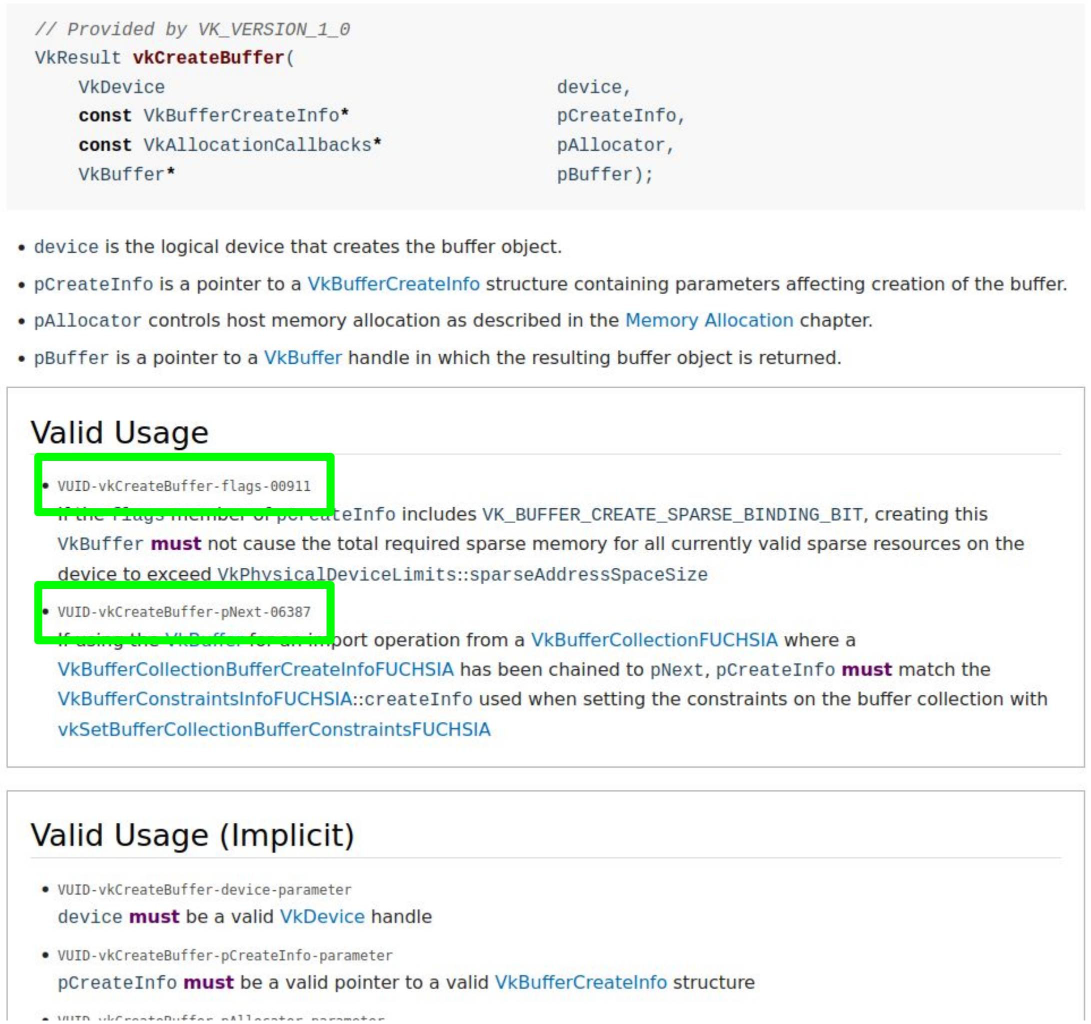
- `device` is the logical device that creates the buffer object.
- `pCreateInfo` is a pointer to a VkBufferCreateInfo structure containing parameters affecting creation of the buffer.
- `pAllocator` controls host memory allocation as described in the Memory Allocation chapter.
- `pBuffer` is a pointer to a VkBuffer handle in which the resulting buffer object is returned.

Valid Usage

- `VUID-vkCreateBuffer-flags-00911`
  If the `flags` member of `pCreateInfo` includes `VK_BUFFER_CREATE_SPARSE_BINDING_BIT`, creating this `VkBuffer` **must** not cause the total required sparse memory for all currently valid sparse resources on the device to exceed `VkPhysicalDeviceLimits::sparseAddressSpaceSize`
- `VUID-vkCreateBuffer-pNext-06387`
  If using the VkBuffer for an import operation from a VkBufferCollectionFUCHSIA where a VkBufferCollectionBufferCreateInfoFUCHSIA has been chained to `pNext`, `pCreateInfo` **must** match the VkBufferConstraintsInfoFUCHSIA::`createInfo` used when setting the constraints on the buffer collection with vkSetBufferCollectionBufferConstraintsFUCHSIA

Valid Usage (Implicit)

- `VUID-vkCreateBuffer-device-parameter`
  `device` **must** be a valid VkDevice handle
- `VUID-vkCreateBuffer-pCreateInfo-parameter`
  `pCreateInfo` **must** be a valid pointer to a valid VkBufferCreateInfo structure
- `VUID-vkCreateBuffer-pAllocator-parameter`



```
// Provided by VK_VERSION_1_0
VkResult vkCreateBuffer(
    VkDevice                                    device,
    const VkBufferCreateInfo*                   pCreateInfo,
    const VkAllocationCallbacks*                pAllocator,
    VkBuffer*                                   pBuffer);
```

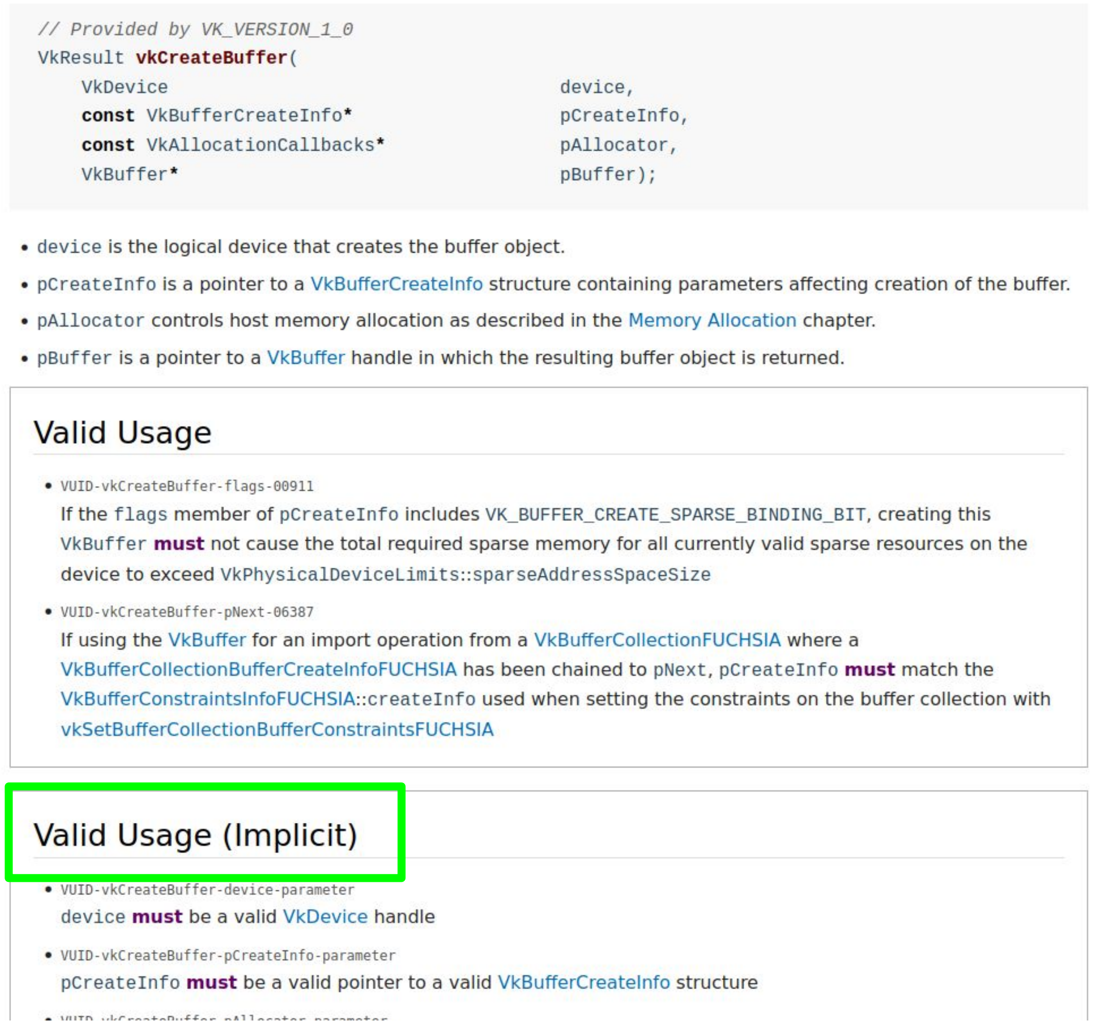
- `device` is the logical device that creates the buffer object.
- `pCreateInfo` is a pointer to a VkBufferCreateInfo structure containing parameters affecting creation of the buffer.
- `pAllocator` controls host memory allocation as described in the Memory Allocation chapter.
- `pBuffer` is a pointer to a VkBuffer handle in which the resulting buffer object is returned.

Valid Usage

- VUID-vkCreateBuffer-flags-00911
  If the `flags` member of `pCreateInfo` includes `VK_BUFFER_CREATE_SPARSE_BINDING_BIT`, creating this `VkBuffer` **must** not cause the total required sparse memory for all currently valid sparse resources on the device to exceed `VkPhysicalDeviceLimits::sparseAddressSpaceSize`
- VUID-vkCreateBuffer-pNext-06387
  If using the VkBuffer for an import operation from a VkBufferCollectionFUCHSIA where a VkBufferCollectionBufferCreateInfoFUCHSIA has been chained to `pNext`, `pCreateInfo` **must** match the VkBufferConstraintsInfoFUCHSIA::`createInfo` used when setting the constraints on the buffer collection with vkSetBufferCollectionBufferConstraintsFUCHSIA

Valid Usage (Implicit)

- VUID-vkCreateBuffer-device-parameter
  `device` **must** be a valid VkDevice handle
- VUID-vkCreateBuffer-pCreateInfo-parameter
  `pCreateInfo` **must** be a valid pointer to a valid VkBufferCreateInfo structure
- VUID-vkCreateBuffer-pAllocator-parameter



```
// Provided by VK_VERSION_1_0
VkResult vkCreateBuffer(
    VkDevice                                    device,
    const VkBufferCreateInfo*                   pCreateInfo,
    const VkAllocationCallbacks*                pAllocator,
    VkBuffer*                                   pBuffer);
```

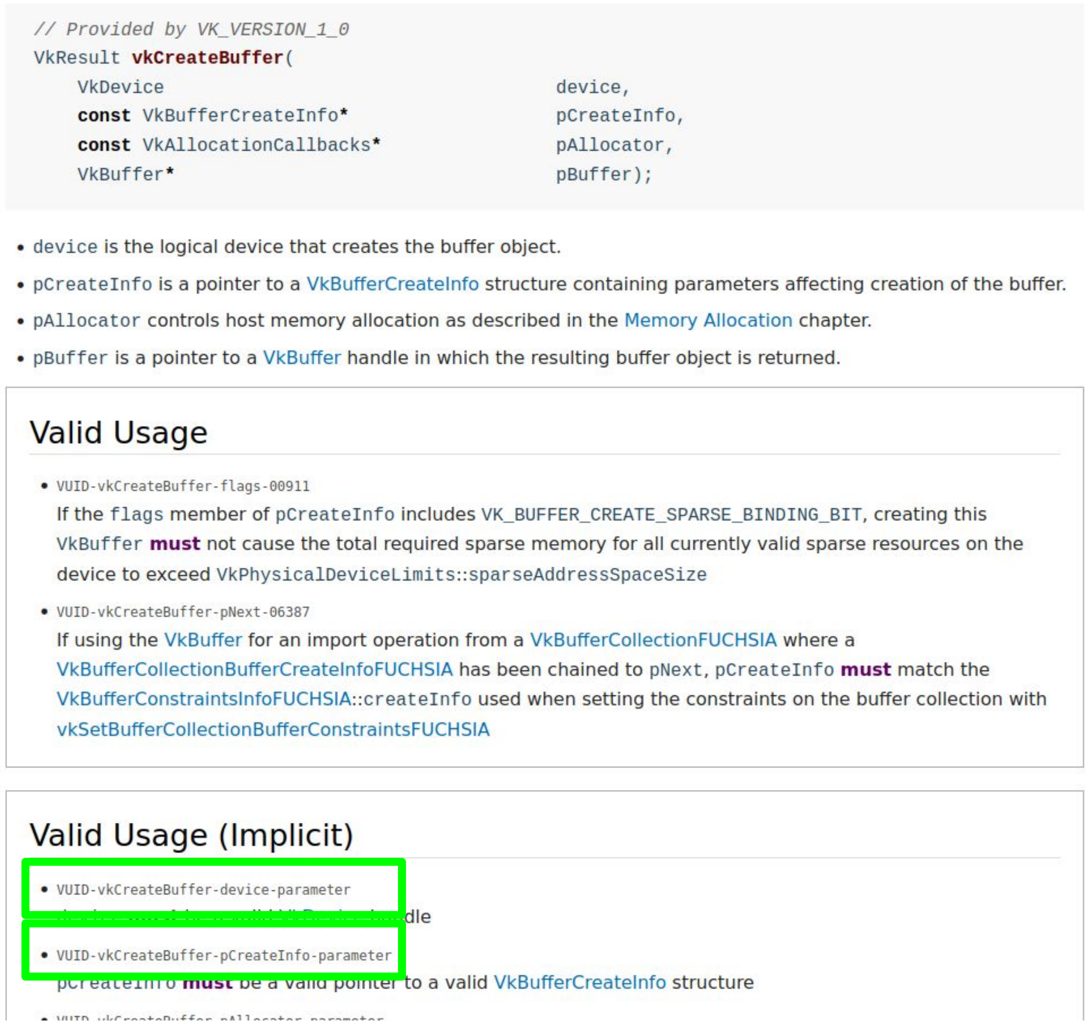
- `device` is the logical device that creates the buffer object.
- `pCreateInfo` is a pointer to a VkBufferCreateInfo structure containing parameters affecting creation of the buffer.
- `pAllocator` controls host memory allocation as described in the Memory Allocation chapter.
- `pBuffer` is a pointer to a VkBuffer handle in which the resulting buffer object is returned.

Valid Usage

- `VUID-vkCreateBuffer-flags-00911`
  If the `flags` member of `pCreateInfo` includes `VK_BUFFER_CREATE_SPARSE_BINDING_BIT`, creating this `VkBuffer` **must** not cause the total required sparse memory for all currently valid sparse resources on the device to exceed `VkPhysicalDeviceLimits::sparseAddressSpaceSize`
- `VUID-vkCreateBuffer-pNext-06387`
  If using the VkBuffer for an import operation from a VkBufferCollectionFUCHSIA where a VkBufferCollectionBufferCreateInfoFUCHSIA has been chained to `pNext`, `pCreateInfo` **must** match the VkBufferConstraintsInfoFUCHSIA::`createInfo` used when setting the constraints on the buffer collection with vkSetBufferCollectionBufferConstraintsFUCHSIA

Valid Usage (Implicit)

- `VUID-vkCreateBuffer-device-parameter`
  `device` **must** be a valid VkDevice handle
- `VUID-vkCreateBuffer-pCreateInfo-parameter`
  `pCreateInfo` **must** be a valid pointer to a valid VkBufferCreateInfo structure
- `VUID-vkCreateBuffer-pAllocator-parameter`



```
// Provided by VK_VERSION_1_0
VkResult vkCreateBuffer(
    VkDevice                                    device,
    const VkBufferCreateInfo*                   pCreateInfo,
    const VkAllocationCallbacks*                pAllocator,
    VkBuffer*                                   pBuffer);
```

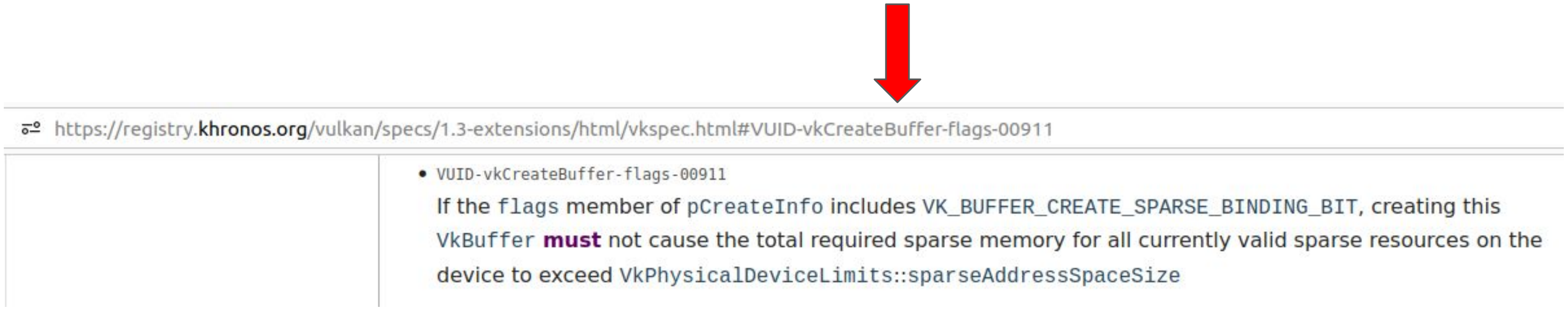
- `device` is the logical device that creates the buffer object.
- `pCreateInfo` is a pointer to a VkBufferCreateInfo structure containing parameters affecting creation of the buffer.
- `pAllocator` controls host memory allocation as described in the Memory Allocation chapter.
- `pBuffer` is a pointer to a VkBuffer handle in which the resulting buffer object is returned.

Valid Usage

- `VUID-vkCreateBuffer-flags-00911`
  If the `flags` member of `pCreateInfo` includes `VK_BUFFER_CREATE_SPARSE_BINDING_BIT`, creating this `VkBuffer` **must** not cause the total required sparse memory for all currently valid sparse resources on the device to exceed `VkPhysicalDeviceLimits::sparseAddressSpaceSize`
- `VUID-vkCreateBuffer-pNext-06387`
  If using the VkBuffer for an import operation from a VkBufferCollectionFUCHSIA where a VkBufferCollectionBufferCreateInfoFUCHSIA has been chained to `pNext`, `pCreateInfo` **must** match the VkBufferConstraintsInfoFUCHSIA::`createInfo` used when setting the constraints on the buffer collection with vkSetBufferCollectionBufferConstraintsFUCHSIA

Valid Usage (Implicit)

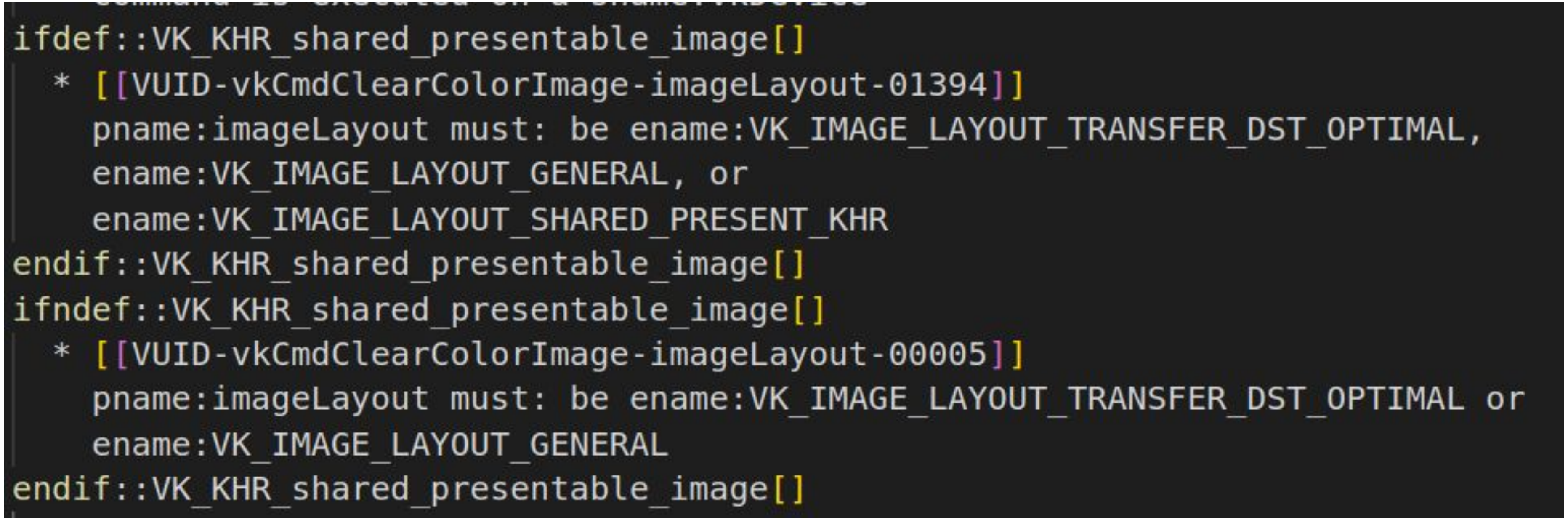
- `VUID-vkCreateBuffer-device-parameter`
  `device` **must** be a valid VkDevice handle
- `VUID-vkCreateBuffer-pCreateInfo-parameter`
  `pCreateInfo` **must** be a valid pointer to a valid VkBufferCreateInfo structure
- `VUID-vkCreateBuffer-pAllocator-parameter`


https://registry.khronos.org/vulkan/specs/1.3-extensions/html/vkspec.html#VUID-vkCreateBuffer-flags-00911

- VUID-vkCreateBuffer-flags-00911

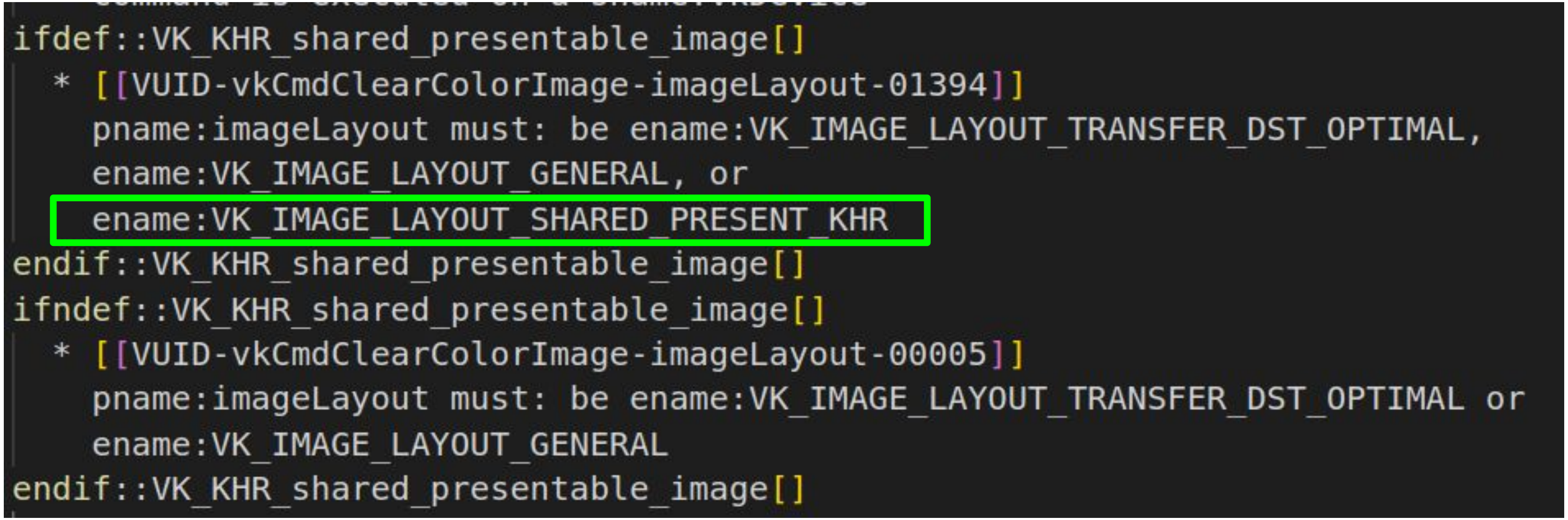
  If the flags member of pCreateInfo includes VK_BUFFER_CREATE_SPARSE_BINDING_BIT, creating this VkBuffer **must** not cause the total required sparse memory for all currently valid sparse resources on the device to exceed VkPhysicalDeviceLimits::sparseAddressSpaceSize

Conditional VUs



```
ifdef::VK_KHR_shared_presentable_image[]
  * [[VUID-vkCmdClearColorImage-imageLayout-01394]]
    pname:imageLayout must: be ename:VK_IMAGE_LAYOUT_TRANSFER_DST_OPTIMAL,
    ename:VK_IMAGE_LAYOUT_GENERAL, or
    ename:VK_IMAGE_LAYOUT_SHARED_PRESENT_KHR
endif::VK_KHR_shared_presentable_image[]
ifndef::VK_KHR_shared_presentable_image[]
  * [[VUID-vkCmdClearColorImage-imageLayout-00005]]
    pname:imageLayout must: be ename:VK_IMAGE_LAYOUT_TRANSFER_DST_OPTIMAL or
    ename:VK_IMAGE_LAYOUT_GENERAL
endif::VK_KHR_shared_presentable_image[]
```

Conditional VUs



```
ifdef::VK_KHR_shared_presentable_image[]
  * [[VUID-vkCmdClearColorImage-imageLayout-01394]]
    pname:imageLayout must: be ename:VK_IMAGE_LAYOUT_TRANSFER_DST_OPTIMAL,
    ename:VK_IMAGE_LAYOUT_GENERAL, or
    ename:VK_IMAGE_LAYOUT_SHARED_PRESENT_KHR
endif::VK_KHR_shared_presentable_image[]
ifndef::VK_KHR_shared_presentable_image[]
  * [[VUID-vkCmdClearColorImage-imageLayout-00005]]
    pname:imageLayout must: be ename:VK_IMAGE_LAYOUT_TRANSFER_DST_OPTIMAL or
    ename:VK_IMAGE_LAYOUT_GENERAL
endif::VK_KHR_shared_presentable_image[]
```

Conditional VUs

https://registry.khronos.org/vulkan/specs/**1.3-extensions**/html/vkspec.html#VUID-vkCmdClearColorImage-imageLayout-**01394**

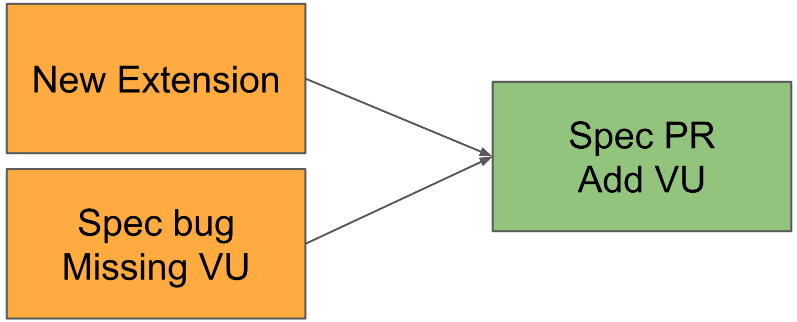
https://registry.khronos.org/vulkan/specs/**1.3**/html/vkspec.html#VUID-vkCmdClearColorImage-imageLayout-**00005**

Life cycle of a VU

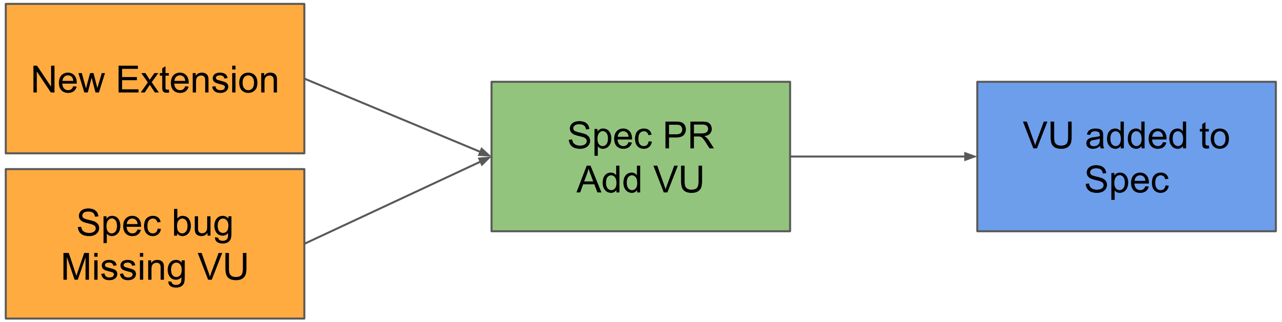
New Extension

Spec bug
Missing VU

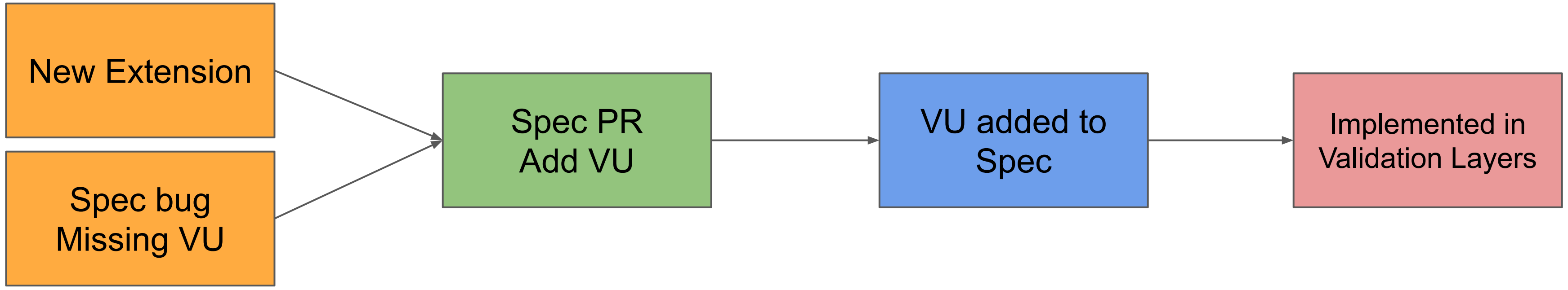
Life cycle of a VU



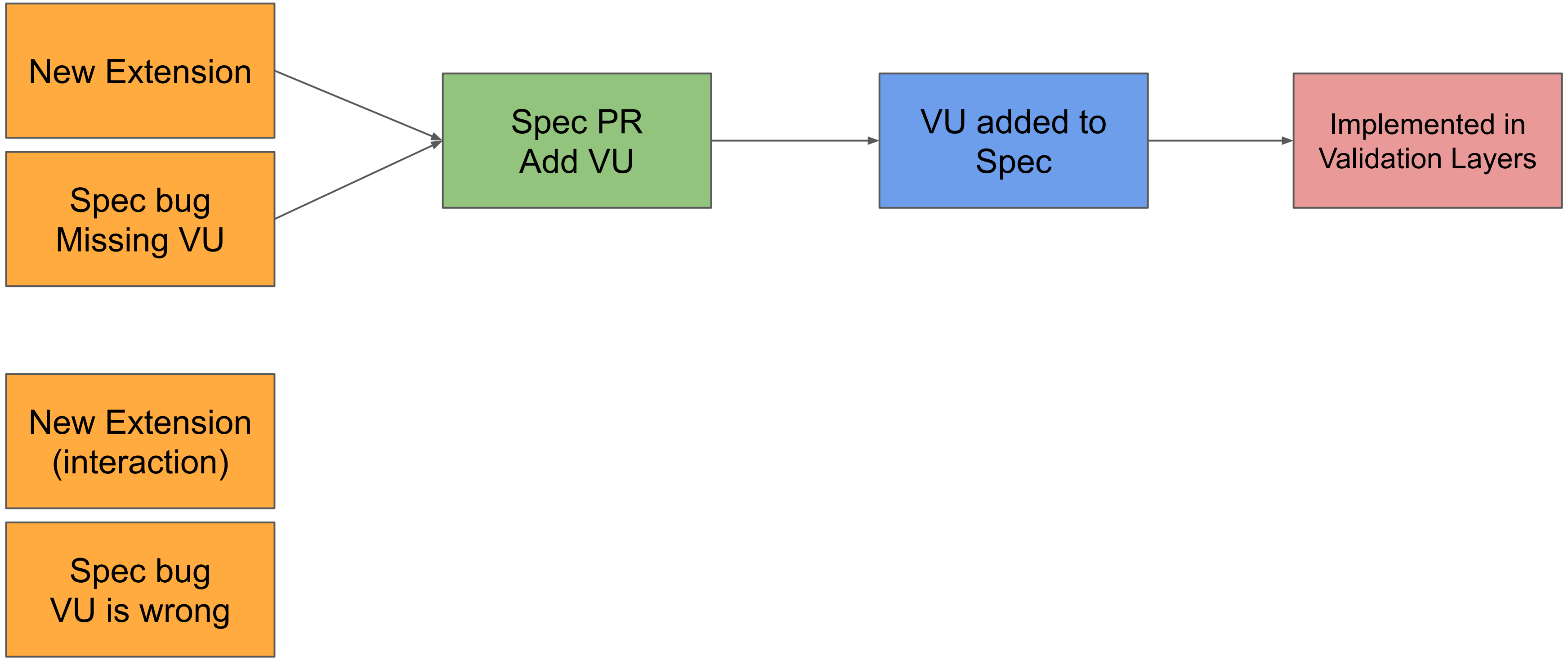
Life cycle of a VU



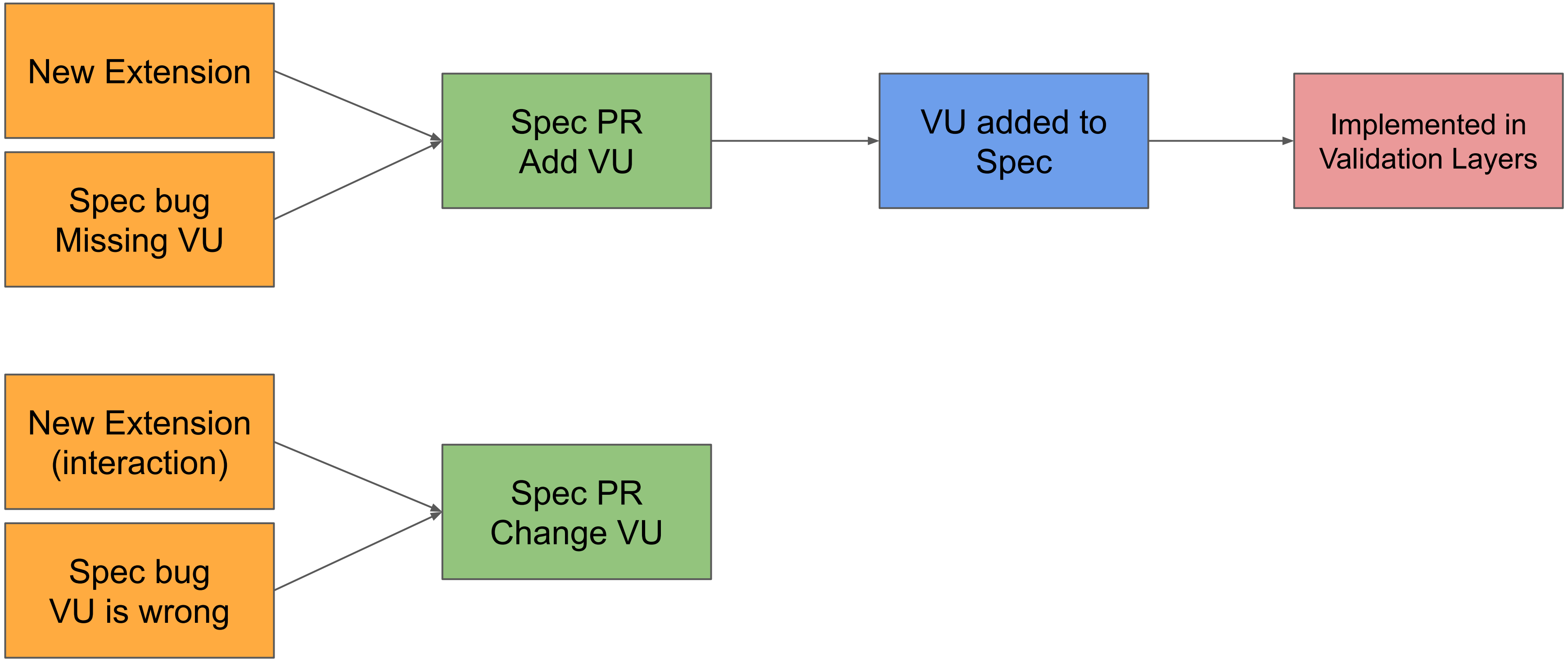
Life cycle of a VU



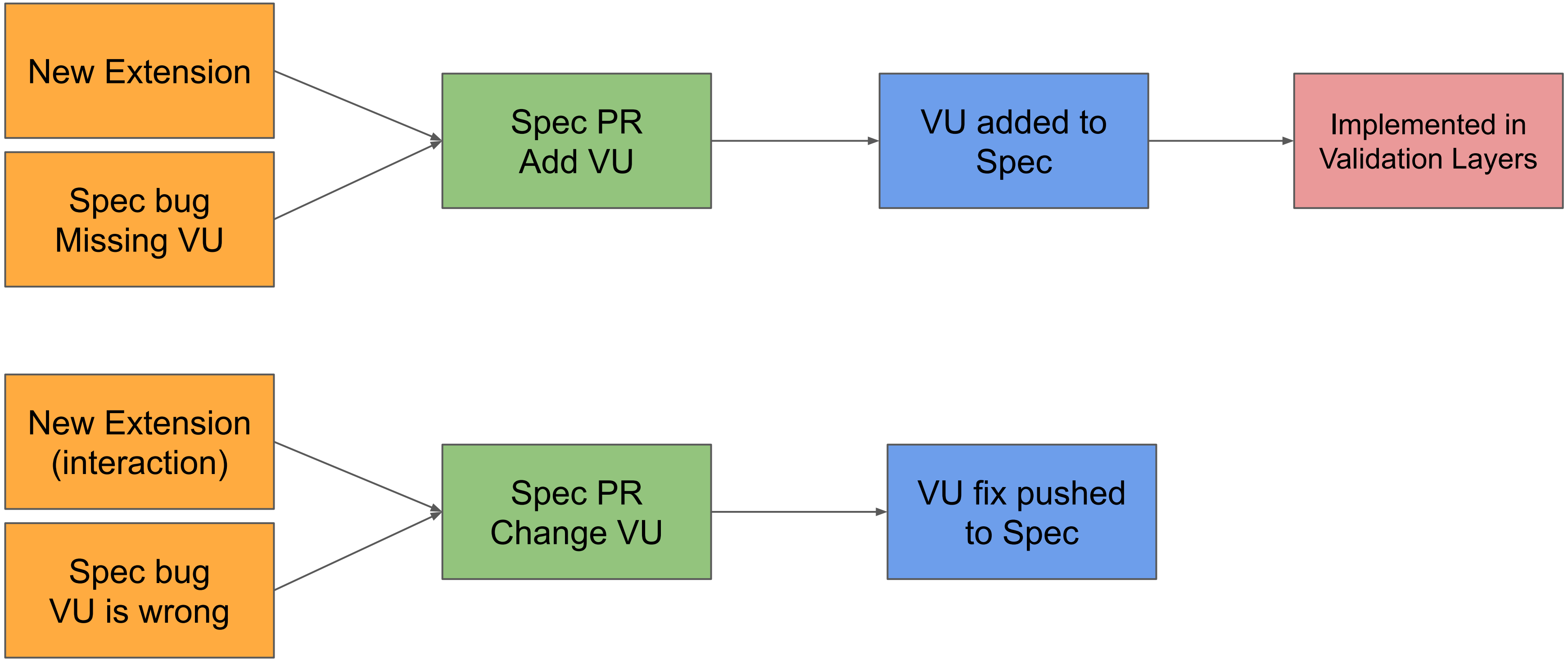
Life cycle of a VU



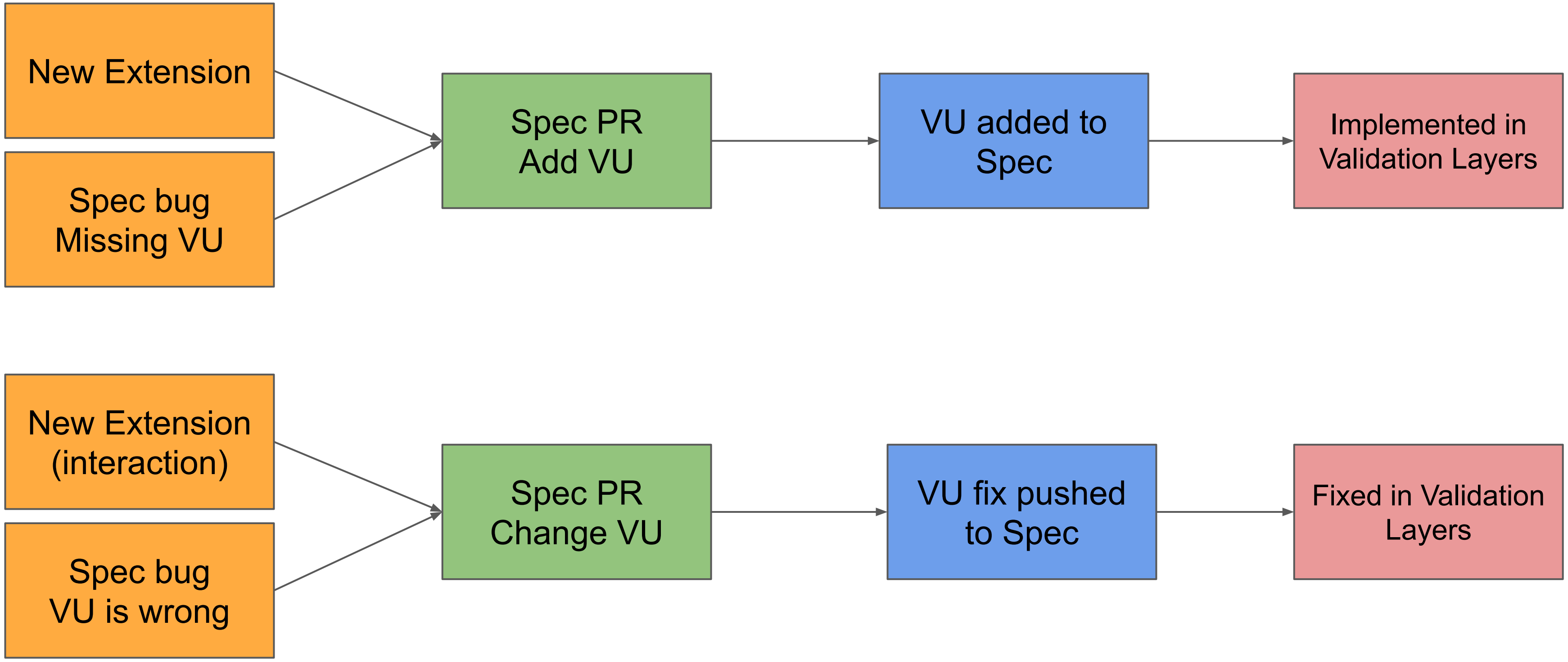
Life cycle of a VU



Life cycle of a VU



Life cycle of a VU



Types of validation - API Usage

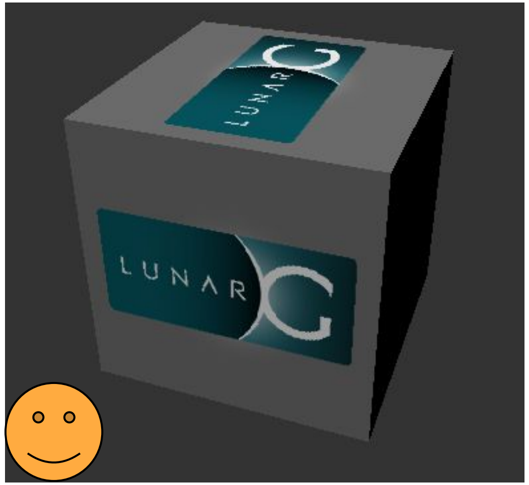
- Developer is using an API incorrectly
    - `vkCreateImage(VK_IMAGE_TYPE_2D, extent.depth = 8);`
- Setting depth, but using a 2D image (not 3D)

Types of validation - Environment

- Unsuccessful interaction between application and its environment
- `VkSubpassDescription::colorAttachmentCount = 5;`
- This *might* succeed or fail, it will depend on the system
  - **maxColorAttachments**
  - Minimum required is only 4

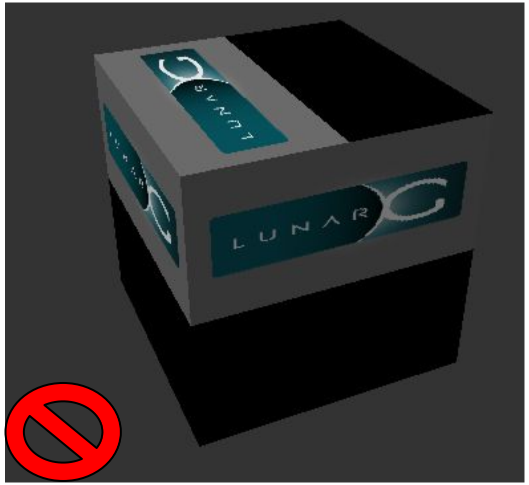
An example error: vkcube

```
VkBufferImageCopy copy_region = {
    .bufferOffset = 0,
    .bufferRowLength = demo->staging_texture.tex_width,
    .bufferImageHeight = demo->staging_texture.tex_height,
    .imageSubresource = {VK_IMAGE_ASPECT_COLOR_BIT, 0, 0, 1},
    .imageOffset = {0, 0, 0},
    .imageExtent = {demo->staging_texture.tex_width, demo->staging_texture.tex_height, 1},
};
vkCmdCopyBufferToImage(demo->cmd, demo->staging_texture.buffer, demo->textures[i].image,
                       VK_IMAGE_LAYOUT_TRANSFER_DST_OPTIMAL, 1, &copy_region)
```



An example error: vkcube

```
VkBufferImageCopy copy_region = {
    .bufferOffset = 0,
    .bufferRowLength = demo->staging_texture.tex_width * 2, // ERROR!
    .bufferImageHeight = demo->staging_texture.tex_height,
    .imageSubresource = {VK_IMAGE_ASPECT_COLOR_BIT, 0, 0, 1},
    .imageOffset = {0, 0, 0},
    .imageExtent = {demo->staging_texture.tex_width, demo->staging_texture.tex_height, 1},
};
vkCmdCopyBufferToImage(demo->cmd, demo->staging_texture.buffer, demo->textures[i].image,
                       VK_IMAGE_LAYOUT_TRANSFER_DST_OPTIMAL, 1, &copy_region)
```



Validation Output: Error Message

```
VUID-vkCmdCopyBufferToImage-pRegions-00171(ERROR / SPEC): msgNum: 1867332608 - Validation Error: [
VUID-vkCmdCopyBufferToImage-pRegions-00171 ] Object 0: handle = 0x56313fd28a00, type =
VK_OBJECT_TYPE_COMMAND_BUFFER; Object 1: handle = 0xd175b40000000013, type = VK_OBJECT_TYPE_BUFFER; |
MessageID = 0x6f4d3c00 | vkCmdCopyBufferToImage: pRegion[0] is trying to copy 523264 bytes plus 0 offset
to/from the VkBuffer (VkBuffer 0xd175b40000000013[]) which exceeds the VkBuffer total size of 262144 bytes.
The Vulkan spec states: srcBuffer must be large enough to contain all buffer locations that are accessed
according to Buffer and Image Addressing, for each element of pRegions
(https://vulkan.lunarg.com/doc/view/1.3.243.0/windows/1.3-extensions/html/vkspec.html#VUID-vkCmdCopyBufferToI
mage-pRegions-00171)
    Objects: 2
        [0] 0x56313fd28a00, type: 6, name: NULL
        [1] 0xd175b40000000013, type: 9, name: NULL
```

Error Message - Basic Info

```
VUID-vkCmdCopyBufferToImage-pRegions-00171(ERROR / SPEC): msgNum: 1867332608 - Validation Error: [
VUID-vkCmdCopyBufferToImage-pRegions-00171 ] Object 0: handle = 0x56313fd28a00, type =
VK_OBJECT_TYPE_COMMAND_BUFFER; Object 1: handle = 0xd175b40000000013, type = VK_OBJECT_TYPE_BUFFER; |
MessageID = 0x6f4d3c00 | vkCmdCopyBufferToImage: pRegion[0] is trying to copy 523264 bytes plus 0 offset
to/from the VkBuffer (VkBuffer 0xd175b40000000013[]) which exceeds the VkBuffer total size of 262144 bytes.
The Vulkan spec states: srcBuffer must be large enough to contain all buffer locations that are accessed
according to Buffer and Image Addressing, for each element of pRegions
(https://vulkan.lunarg.com/doc/view/1.3.243.0/windows/1.3-extensions/html/vkspec.html#VUID-VkCmdCopyBufferToI
mage-pRegions-00171)
    Objects: 2
        [0] 0x56313fd28a00, type: 6, name: NULL
        [1] 0xd175b40000000013, type: 9, name: NULL
```

Error Message - Basic Info

```
VUID-vkCmdCopyBufferToImage-pRegions-00171(ERROR / SPEC): msgNum: 1867332608 - Validation Error: [
VUID-vkCmdCopyBufferToImage-pRegions-00171 ] Object 0: handle = 0x56313fd28a00, type =
VK_OBJECT_TYPE_COMMAND_BUFFER; Object 1: handle = 0xd175b40000000013, type = VK_OBJECT_TYPE_BUFFER; |
MessageID = 0x6f4d3c00 | vkCmdCopyBufferToImage: pRegion[0] is trying to copy 523264 bytes plus 0 offset
to/from the VkBuffer (VkBuffer 0xd175b40000000013[]) which exceeds the VkBuffer total size of 262144 bytes.
The Vulkan spec states: srcBuffer must be large enough to contain all buffer locations that are accessed
according to Buffer and Image Addressing, for each element of pRegions
(https://vulkan.lunarg.com/doc/view/1.3.243.0/windows/1.3-extensions/html/vkspec.html#VUID-VkCmdCopyBufferToI
mage-pRegions-00171)
    Objects: 2
        [0] 0x56313fd28a00, type: 6, name: NULL
        [1] 0xd175b40000000013, type: 9, name: NULL
```

Error Message - Basic Info

```
VUID-vkCmdCopyBufferToImage-pRegions-00171(ERROR / SPEC): msgNum: 1867332608 - Validation Error: [
VUID-vkCmdCopyBufferToImage-pRegions-00171 ] Object 0: handle = 0x56313fd28a00, type =
VK_OBJECT_TYPE_COMMAND_BUFFER; Object 1: handle = 0xd175b40000000013, type = VK_OBJECT_TYPE_BUFFER; |
MessageID = 0x6f4d3c00 | vkCmdCopyBufferToImage: pRegion[0] is trying to copy 523264 bytes plus 0 offset
to/from the VkBuffer (VkBuffer 0xd175b40000000013[]) which exceeds the VkBuffer total size of 262144 bytes.
The Vulkan spec states: srcBuffer must be large enough to contain all buffer locations that are accessed
according to Buffer and Image Addressing, for each element of pRegions
(https://vulkan.lunarg.com/doc/view/1.3.243.0/windows/1.3-extensions/html/vkspec.html#VUID-VkCmdCopyBufferToI
mage-pRegions-00171)
    Objects: 2
        [0] 0x56313fd28a00, type: 6, name: NULL
        [1] 0xd175b40000000013, type: 9, name: NULL
```

Error Message - Basic Info

```
VUID-vkCmdCopyBufferToImage-pRegions-00171(ERROR / SPEC): msgNum: 1867332608 - Validation Error: [
VUID-vkCmdCopyBufferToImage-pRegions-00171 ] Object 0: handle = 0x56313fd28a00, type =
VK_OBJECT_TYPE_COMMAND_BUFFER; Object 1: handle = 0xd175b40000000013, type = VK_OBJECT_TYPE_BUFFER; |
MessageID = 0x6f4d3c00 | vkCmdCopyBufferToImage: pRegion[0] is trying to copy 523264 bytes plus 0 offset
to/from the VkBuffer (VkBuffer 0xd175b40000000013[]) which exceeds the VkBuffer total size of 262144 bytes.
The Vulkan spec states: srcBuffer must be large enough to contain all buffer locations that are accessed
according to Buffer and Image Addressing, for each element of pRegions
(https://vulkan.lunarg.com/doc/view/1.3.243.0/windows/1.3-extensions/html/vkspec.html#VUID-VkCmdCopyBufferToI
mage-pRegions-00171)
    Objects: 2
        [0] 0x56313fd28a00, type: 6, name: NULL
        [1] 0xd175b40000000013, type: 9, name: NULL
```

- msgNum / MessageID is a hash of the VUID string, used for handling duplicate messages

Error Message - Main message

```
VUID-vkCmdCopyBufferToImage-pRegions-00171(ERROR / SPEC): msgNum: 1867332608 - Validation Error: [
VUID-vkCmdCopyBufferToImage-pRegions-00171 ] Object 0: handle = 0x56313fd28a00, type =
VK_OBJECT_TYPE_COMMAND_BUFFER; Object 1: handle = 0xd175b40000000013, type = VK_OBJECT_TYPE_BUFFER; |
MessageID = 0x6f4d3c00 | vkCmdCopyBufferToImage: pRegion[0] is trying to copy 523264 bytes plus 0 offset
to/from the VkBuffer (VkBuffer 0xd175b40000000013[]) which exceeds the VkBuffer total size of 262144 bytes.
The Vulkan spec states: srcBuffer must be large enough to contain all buffer locations that are accessed
according to Buffer and Image Addressing, for each element of pRegions
(https://vulkan.lunarg.com/doc/view/1.3.243.0/windows/1.3-extensions/html/vkspec.html#VUID-vkCmdCopyBufferToI
mage-pRegions-00171)
    Objects: 2
        [0] 0x56313fd28a00, type: 6, name: NULL
        [1] 0xd175b40000000013, type: 9, name: NULL
```

Error Message - Main message

```
VUID-vkCmdCopyBufferToImage-pRegions-00171(ERROR / SPEC): msgNum: 1867332608 - Validation Error: [
VUID-vkCmdCopyBufferToImage-pRegions-00171 ] Object 0: handle = 0x56313fd28a00, type =
VK_OBJECT_TYPE_COMMAND_BUFFER; Object 1: handle = 0xd175b40000000013, type = VK_OBJECT_TYPE_BUFFER; |
MessageID = 0x6f4d3c00 | vkCmdCopyBufferToImage: pRegion[0] is trying to copy 523264 bytes plus 0 offset
to/from the VkBuffer (VkBuffer 0xd175b40000000013[]) which exceeds the VkBuffer total size of 262144 bytes.
The Vulkan spec states: srcBuffer must be large enough to contain all buffer locations that are accessed
according to Buffer and Image Addressing, for each element of pRegions
(https://vulkan.lunarg.com/doc/view/1.3.243.0/windows/1.3-extensions/html/vkspec.html#VUID-vkCmdCopyBufferToI
mage-pRegions-00171)
    Objects: 2
        [0] 0x56313fd28a00, type: 6, name: NULL
        [1] 0xd175b40000000013, type: 9, name: NULL
```

Error Message - Main message

```
VUID-vkCmdCopyBufferToImage-pRegions-00171(ERROR / SPEC): msgNum: 1867332608 - Validation Error: [
VUID-vkCmdCopyBufferToImage-pRegions-00171 ] Object 0: handle = 0x56313fd28a00, type =
VK_OBJECT_TYPE_COMMAND_BUFFER; Object 1: handle = 0xd175b40000000013, type = VK_OBJECT_TYPE_BUFFER; |
MessageID = 0x6f4d3c00 | vkCmdCopyBufferToImage: pRegion[0] is trying to copy 523264 bytes plus 0 offset
to/from the VkBuffer (VkBuffer 0xd175b40000000013[]) which exceeds the VkBuffer total size of 262144 bytes.
The Vulkan spec states: srcBuffer must be large enough to contain all buffer locations that are accessed
according to Buffer and Image Addressing, for each element of pRegions
(https://vulkan.lunarg.com/doc/view/1.3.243.0/windows/1.3-extensions/html/vkspec.html#VUID-vkCmdCopyBufferToI
mage-pRegions-00171)
    Objects: 2
        [0] 0x56313fd28a00, type: 6, name: NULL
        [1] 0xd175b40000000013, type: 9, name: NULL
```

Error Message - Main message

```
VUID-vkCmdCopyBufferToImage-pRegions-00171(ERROR / SPEC): msgNum: 1867332608 - Validation Error: [
VUID-vkCmdCopyBufferToImage-pRegions-00171 ] Object 0: handle = 0x56313fd28a00, type =
VK_OBJECT_TYPE_COMMAND_BUFFER; Object 1: handle = 0xd175b40000000013, type = VK_OBJECT_TYPE_BUFFER; |
MessageID = 0x6f4d3c00 | vkCmdCopyBufferToImage: pRegion[0] is trying to copy 523264 bytes plus 0 offset
to/from the VkBuffer (VkBuffer 0xd175b40000000013[]) which exceeds the VkBuffer total size of 262144 bytes.
The Vulkan spec states: srcBuffer must be large enough to contain all buffer locations that are accessed
according to Buffer and Image Addressing, for each element of pRegions
(https://vulkan.lunarg.com/doc/view/1.3.243.0/windows/1.3-extensions/html/vkspec.html#VUID-vkCmdCopyBufferToI
mage-pRegions-00171)
    Objects: 2
        [0] 0x56313fd28a00, type: 6, name: NULL
        [1] 0xd175b40000000013, type: 9, name: NULL
```

Advise - Fixing errors

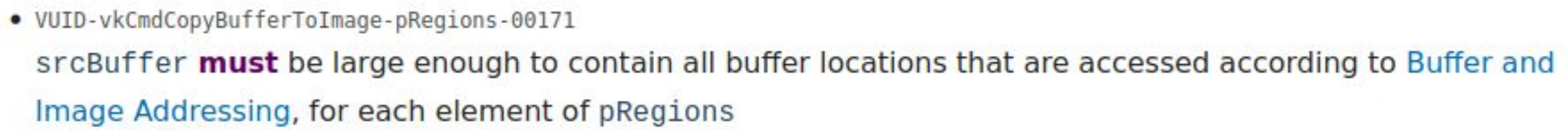
- Fix the first error message first
  - Similar to with C/C++ compiler errors, the first error may cause subsequent errors

Error Message - Spec Reference

```
VUID-vkCmdCopyBufferToImage-pRegions-00171(ERROR / SPEC): msgNum: 1867332608 - Validation Error: [
VUID-vkCmdCopyBufferToImage-pRegions-00171 ] Object 0: handle = 0x56313fd28a00, type =
VK_OBJECT_TYPE_COMMAND_BUFFER; Object 1: handle = 0xd175b40000000013, type = VK_OBJECT_TYPE_BUFFER; |
MessageID = 0x6f4d3c00 | vkCmdCopyBufferToImage: pRegion[0] is trying to copy  523264 bytes plus 0 offset
to/from the VkBuffer (VkBuffer 0xd175b40000000013[]) which exceeds the VkBuffer total size of 262144 bytes.
The Vulkan spec states: srcBuffer must be large enough to contain all buffer locations that are accessed
according to Buffer and Image Addressing, for each element of pRegions
(https://vulkan.lunarg.com/doc/view/1.3.243.0/windows/1.3-extensions/html/vkspec.html#VUID-vkCmdCopyBufferToI
mage-pRegions-00171)
    Objects: 2
        [0] 0x56313fd28a00, type: 6, name: NULL
        [1] 0xd175b40000000013, type: 9, name: NULL
```

Error Message - Spec Reference

```
VUID-vkCmdCopyBufferToImage-pRegions-00171(ERROR / SPEC): msgNum: 1867332608 - Validation Error: [
VUID-vkCmdCopyBufferToImage-pRegions-00171 ] Object 0: handle = 0x56313fd28a00, type =
VK_OBJECT_TYPE_COMMAND_BUFFER; Object 1: handle = 0xd175b40000000013, type = VK_OBJECT_TYPE_BUFFER; |
MessageID = 0x6f4d3c00 | vkCmdCopyBufferToImage: pRegion[0] is trying to copy  523264 bytes plus 0 offset
to/from the VkBuffer (VkBuffer 0xd175b40000000013[]) which exceeds the VkBuffer total size of 262144 bytes.
The Vulkan spec states: srcBuffer must be large enough to contain all buffer locations that are accessed
according to Buffer and Image Addressing, for each element of pRegions
(https://vulkan.lunarg.com/doc/view/1.3.243.0/windows/1.3-extensions/html/vkspec.html#VUID-vkCmdCopyBufferToI
mage-pRegions-00171)
    Objects: 2
        [0] 0x56313fd28a00, type: 6, name: NULL
        [1] 0xd175b40000000013, type: 9, name: NULL
```



- VUID-vkCmdCopyBufferToImage-pRegions-00171

  srcBuffer **must** be large enough to contain all buffer locations that are accessed according to Buffer and Image Addressing, for each element of pRegions

Error Message - Object Handles

```
VUID-vkCmdCopyBufferToImage-pRegions-00171(ERROR / SPEC): msgNum: 1867332608 - Validation Error: [
VUID-vkCmdCopyBufferToImage-pRegions-00171 ] Object 0: handle = 0x56313fd28a00, type =
VK_OBJECT_TYPE_COMMAND_BUFFER; Object 1: handle = 0xd175b40000000013, type = VK_OBJECT_TYPE_BUFFER; |
MessageID = 0x6f4d3c00 | vkCmdCopyBufferToImage: pRegion[0] is trying to copy 523264 bytes plus 0 offset
to/from the VkBuffer (VkBuffer 0xd175b40000000013[]) which exceeds the VkBuffer total size of 262144 bytes.
The Vulkan spec states: srcBuffer must be large enough to contain all buffer locations that are accessed
according to Buffer and Image Addressing, for each element of pRegions
(https://vulkan.lunarg.com/doc/view/1.3.243.0/windows/1.3-extensions/html/vkspec.html#VUID-vkCmdCopyBufferToI
mage-pRegions-00171)
    Objects: 2
        [0] 0x56313fd28a00, type: 6, name: NULL
        [1] 0xd175b40000000013, type: 9, name: NULL
```

Error Message - Object Handles

```
VUID-vkCmdCopyBufferToImage-pRegions-00171(ERROR / SPEC): msgNum: 1867332608 - Validation Error: [
VUID-vkCmdCopyBufferToImage-pRegions-00171 ] Object 0: handle = 0x56313fd28a00, type =
VK_OBJECT_TYPE_COMMAND_BUFFER; Object 1: handle = 0xd175b40000000013, type = VK_OBJECT_TYPE_BUFFER; |
MessageID = 0x6f4d3c00 | vkCmdCopyBufferToImage: pRegion[0] is trying to copy 523264 bytes plus 0 offset
to/from the VkBuffer (VkBuffer 0xd175b40000000013[]) which exceeds the VkBuffer total size of 262144 bytes.
The Vulkan spec states: srcBuffer must be large enough to contain all buffer locations that are accessed
according to Buffer and Image Addressing, for each element of pRegions
(https://vulkan.lunarg.com/doc/view/1.3.243.0/windows/1.3-extensions/html/vkspec.html#VUID-vkCmdCopyBufferToI
mage-pRegions-00171)
    Objects: 2
        [0] 0x56313fd28a00, type: 6, name: NULL
        [1] 0xd175b40000000013, type: 9, name: NULL
```

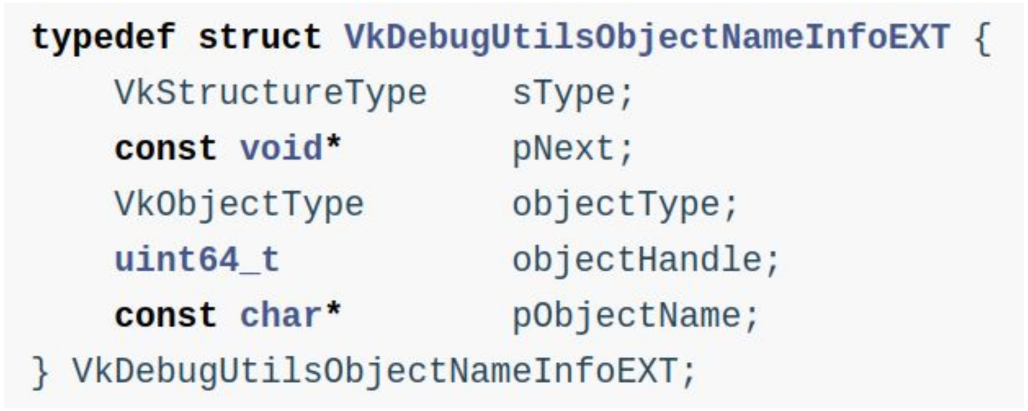
- List the Objects that were part of the error
  - Helps to know which VkCommandBuffer and VkBuffer this error is about
  - Can use **VK_EXT_debug_utils** to give these objects name

Debug Utilities Extension

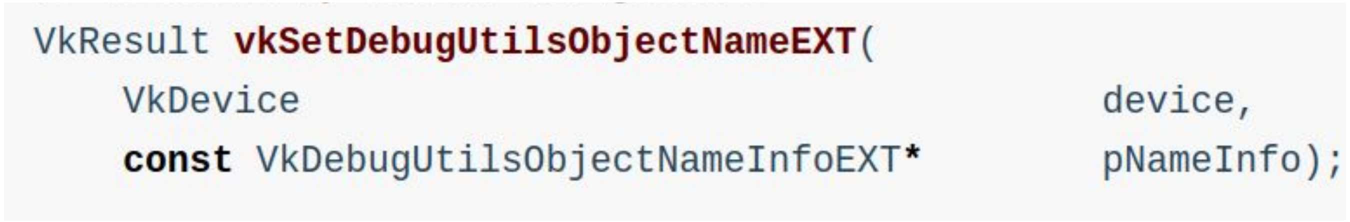
- VK_EXT_debug_utils
  - Replaced original VK_EXT_debug_report/VK_EXT_debug_marker
- Implemented by Vulkan-ValidationLayers
- Provides the ability to attach user-defined names to
  - Vulkan Objects
  - Sequences of commands recorded in Command Buffers
  - Queue submissions
- Names show up in validation error messages and are also used by other tools such as RenderDoc
- Allows applications to register their own validation error handling callback

```
typedef struct VkDebugUtilsObjectNameInfoEXT {
    VkStructureType    sType;
    const void*        pNext;
    VkObjectType       objectType;
    uint64_t           objectHandle;
    const char*        pObjectName;
} VkDebugUtilsObjectNameInfoEXT;
```

```
VkResult vkSetDebugUtilsObjectNameEXT(
    VkDevice                                    device,
    const VkDebugUtilsObjectNameInfoEXT*        pNameInfo);
```



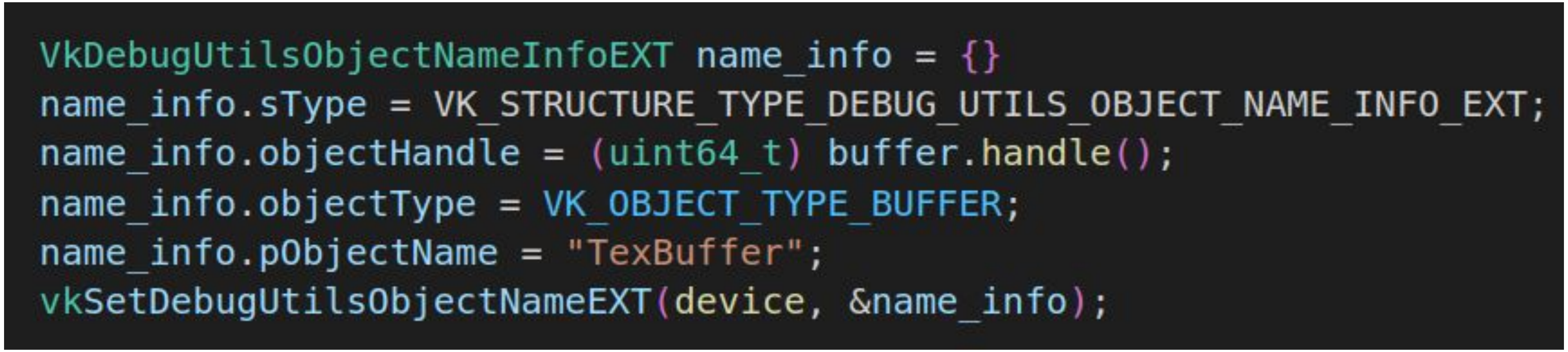
```
typedef struct VkDebugUtilsObjectNameInfoEXT {
    VkStructureType    sType;
    const void*        pNext;
    VkObjectType       objectType;
    uint64_t           objectHandle;
    const char*        pObjectName;
} VkDebugUtilsObjectNameInfoEXT;
```



```
VkResult vkSetDebugUtilsObjectNameEXT(
    VkDevice                                    device,
    const VkDebugUtilsObjectNameInfoEXT*        pNameInfo);
```

- Allows a name to be attached to any Vulkan object
- Can help you identify what part of your code is causing an error.
- Contents of `pObjectName` is copied to internal storage.

Debug Utilities Extension: Object naming



```
VkDebugUtilsObjectNameInfoEXT name_info = {}
name_info.sType = VK_STRUCTURE_TYPE_DEBUG_UTILS_OBJECT_NAME_INFO_EXT;
name_info.objectHandle = (uint64_t) buffer.handle();
name_info.objectType = VK_OBJECT_TYPE_BUFFER;
name_info.pObjectName = "TexBuffer";
vkSetDebugUtilsObjectNameEXT(device, &name_info);
```

```
Objects - 2
     Object[0] - VK_OBJECT_TYPE_COMMAND_BUFFER, Handle 0x5566702c9f60, Name "PrepareCB"
     Object[1] - VK_OBJECT_TYPE_BUFFER, Handle 0x9fde6b0000000014, Name "TexBuffer"
```

```
typedef struct VkDebugUtilsLabelEXT {
    VkStructureType    sType;
    const void*        pNext;
    const char*        pLabelName;
    float              color[4];
} VkDebugUtilsLabelEXT;
```

```
void vkCmdBeginDebugUtilsLabelEXT(
    VkCommandBuffer                             commandBuffer,
    const VkDebugUtilsLabelEXT*                 pLabelInfo);
```

Debug Utilities extension: Command buffer labels

- Allows a name to be attached to a sequence of commands in a command buffer
- Stack-like, multiple labels can be present at once
    - `vkCmdBeginDebugUtilsLabelEXT()` pushes
    - `vkCmdEndDebugUtilsLabelEXT()` pops
- The `color` field is used by tools like RenderDoc
- See also `vkQueueBeginDebugUtilsLabelEXT()`
- Not printed by default error handler!

```
Command Buffer Labels - 3
     Label[0] - StagingBufferCopy(0) { 0.000000, 0.000000, 0.000000, 0.000000}
     Label[1] - StagingTexture(0) { 0.000000, 0.000000, 0.000000, 0.000000}
     Label[2] - Prepare { 0.000000, 0.000000, 0.000000, 0.000000}
```

Debug Utilities extension: vkcube error callback

```
ERROR : VALIDATION - Message Id Number: 1867332608 | Message Id Name:
VUID-vkCmdCopyBufferToImage-pRegions-00171
      Validation Error: [ VUID-vkCmdCopyBufferToImage-pRegions-00171 ] Object 0: handle = 0x562780095ca0,
name = PrepareCB, type = VK_OBJECT_TYPE_COMMAND_BUFFER; Object 1: handle = 0x9fde6b0000000014, name =
TexBuffer type = VK_OBJECT_TYPE_BUFFER; | MessageID = 0x6f4d3c00 | vkCmdCopyBufferToImage: pRegion[0] is
trying to copy  523264 bytes plus 0 offset to/from the VkBuffer (VkBuffer 0x9fde6b0000000014[TexBuffer])
which exceeds the VkBuffer total size of 262144 bytes. The Vulkan spec states: srcBuffer must be large enough
to contain all buffer locations that are accessed according to Buffer and Image Addressing, for each element
of pRegions
(https://vulkan.lunarg.com/doc/view/1.3.243.0/windows/1.3-extensions/html/vkspec.html#VUID-vkCmdCopyBufferToI
mage-pRegions-00171)

      Objects - 2
            Object[0] - VK_OBJECT_TYPE_COMMAND_BUFFER, Handle 0x562780095ca0, Name "PrepareCB"
            Object[1] - VK_OBJECT_TYPE_BUFFER, Handle 0x9fde6b0000000014, Name "TexBuffer"

      Command Buffer Labels - 3
            Label[0] - StagingBufferCopy(0) { 0.000000, 0.000000, 0.000000, 0.000000}
            Label[1] - StagingTexture(0) { 0.000000, 0.000000, 0.000000, 0.000000}
            Label[2] - Prepare { 0.000000, 0.000000, 0.000000, 0.000000}
```

LUNARG

Debug Utilities extension: Custom message callback

- Set up by calling `vkCreateDebugUtilsMessengerEXT()`
  - Your callback receives a complex struct for each error
  - Same mechanism used for default error logging
- Make your own message format
- Add messages to application logging stream
- Send messages to somewhere other than the console
- Trigger failures in your unit test framework
- Filter out unwanted messages (NOT recommended, built-in filtering is faster)

LUNARG

Validation Quick Start - Get the binary

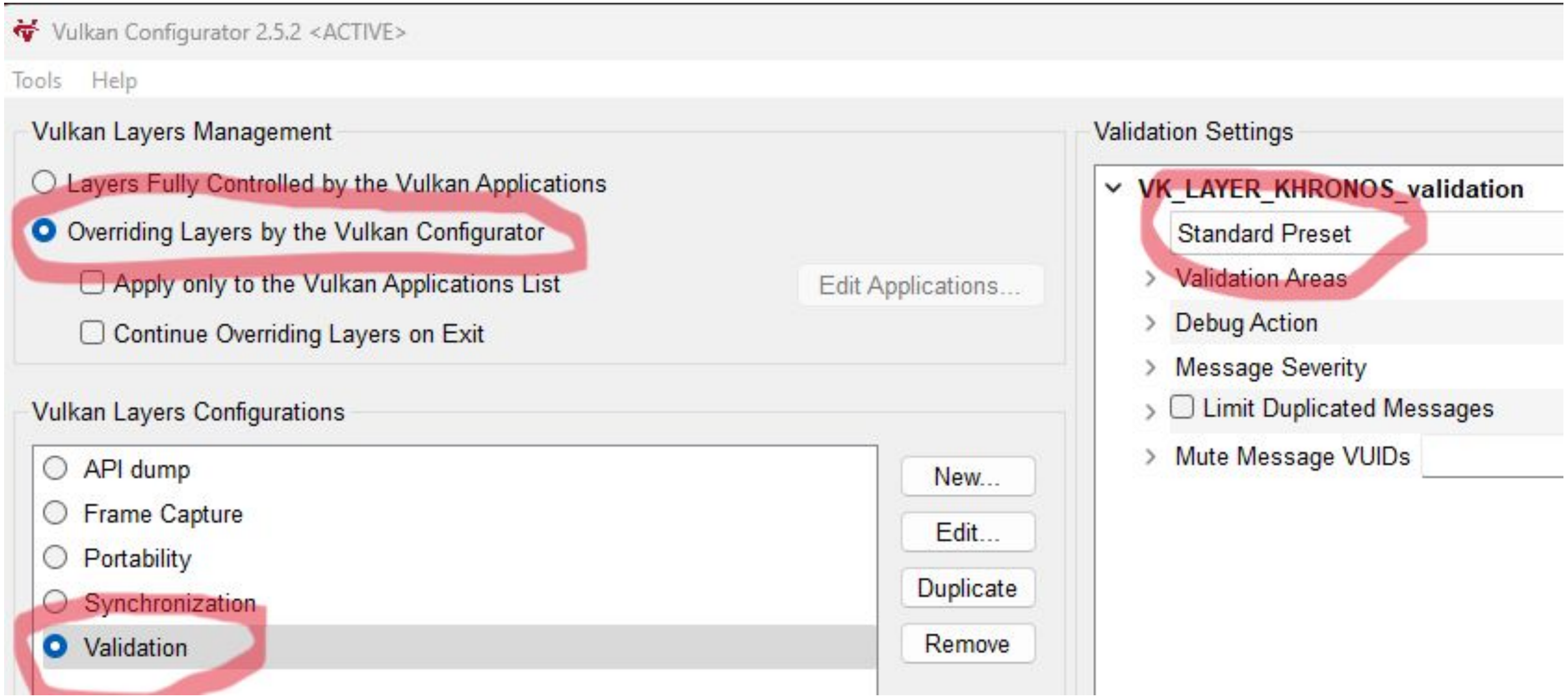
- Install the Vulkan SDK or OS-provided packages
  - Well tested version
- Build from source
  - Great for tracking down a bug
  - Get latest changes
  - Hopefully not hard to build

Validation Quick Start - Enable

- Validation Layers are used like any other Vulkan Layer

- Run **vkconfig** (Simplest)
- At **vkCreateInstance()** time
  - Add the layer name to `VkInstanceCreateInfo::ppEnabledLayerNames`
- From the terminal (Power user)
  - `export VK_INSTANCE_LAYERS=VK_LAYER_KHRONOS_validation ./your-application`

Vulkan Configurator



Vulkan Configurator 2.5.2 <ACTIVE>

Tools Help

Vulkan Layers Management

( ) Layers Fully Controlled by the Vulkan Applications

(●) Overriding Layers by the Vulkan Configurator

☐ Apply only to the Vulkan Applications List

☐ Continue Overriding Layers on Exit

Vulkan Layers Configurations

( ) API dump

( ) Frame Capture

( ) Portability

( ) Synchronization

(●) Validation

Validation Settings

VK_LAYER_KHRONOS_validation

Standard Preset

Validation Areas

Debug Action

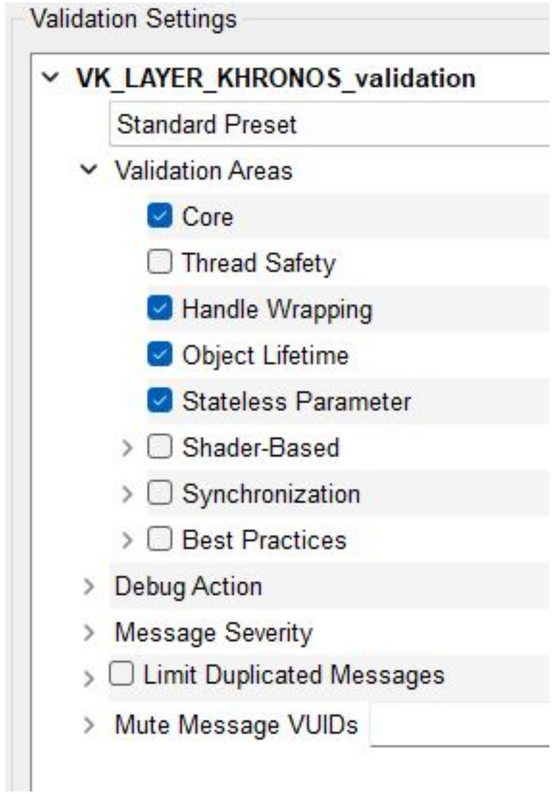
Message Severity

☐ Limit Duplicated Messages

Mute Message VUIDs

Configuration

- Validation is split up into several areas to reduce performance overhead
- Don’t enable all areas at once (it will be slow!)
- Fix errors in each area,
    - then run Core / Standard Preset again

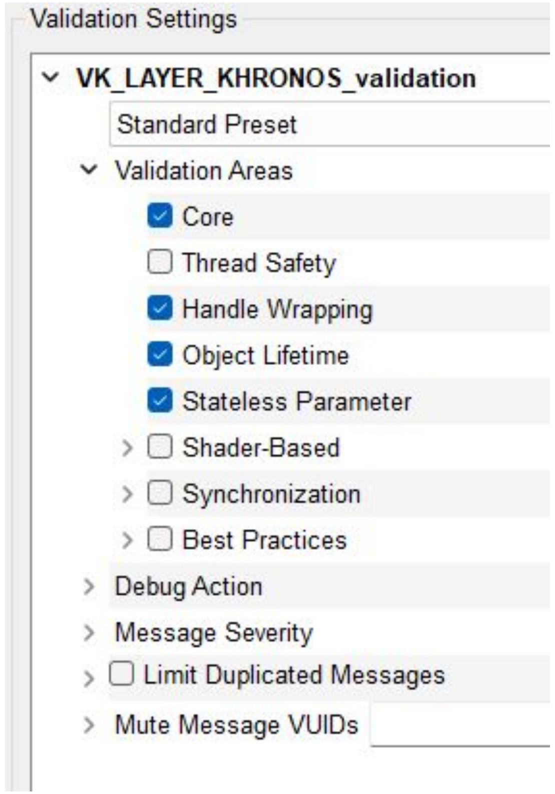


Validation Settings

- VK_LAYER_KHRONOS_validation
    - Standard Preset
    - Validation Areas
        - [✓] Core
        - [ ] Thread Safety
        - [✓] Handle Wrapping
        - [✓] Object Lifetime
        - [✓] Stateless Parameter
        - [ ] Shader-Based
        - [ ] Synchronization
        - [ ] Best Practices
    - Debug Action
    - Message Severity
    - [ ] Limit Duplicated Messages
    - Mute Message VUIDs

Configuration - How to set

- Use vkconfig presets
  - Commonly used and tested configurations
- Can use vk_layer_settings.txt
  - Khronos_validation.enables
  - khronos_validation.disables
- Environment variables
  - VK_LAYER_ENABLES
  - VK_LAYER_DISABLES
- VK_EXT_validation_features
  - Set at VkDevice creation time
- https://vulkan.lunarg.com/doc/sdk/latest/windows/khronos_validation_layer.html

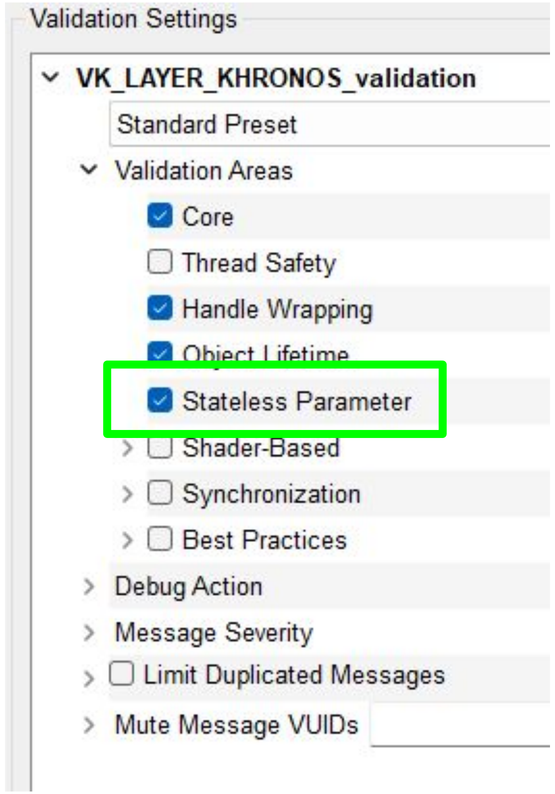


Validation Settings

- VK_LAYER_KHRONOS_validation
  - Standard Preset
  - Validation Areas
    - ✓ Core
    - ☐ Thread Safety
    - ✓ Handle Wrapping
    - ✓ Object Lifetime
    - ✓ Stateless Parameter
    - ☐ Shader-Based
    - ☐ Synchronization
    - ☐ Best Practices
  - Debug Action
  - Message Severity
  - ☐ Limit Duplicated Messages
  - Mute Message VUIDs

Configuration: Stateless

- Checks simple VUIDs
- Lots of generated checks
- doesn't require expensive state tracking


Validation Settings

VK_LAYER_KHRONOS_validation

Standard Preset

Validation Areas

- [x] Core
- [ ] Thread Safety
- [x] Handle Wrapping
- [x] Object Lifetime
- [x] Stateless Parameter
- [ ] Shader-Based
- [ ] Synchronization
- [ ] Best Practices

Debug Action

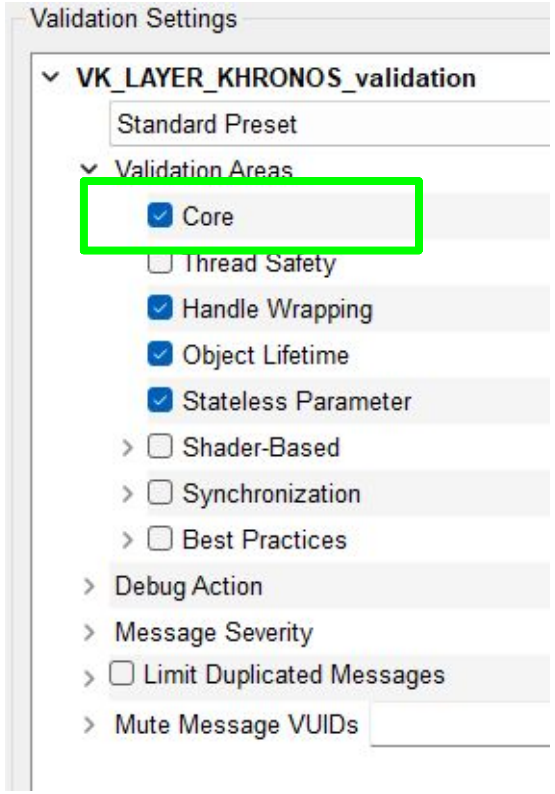
Message Severity

- [ ] Limit Duplicated Messages

Mute Message VUIDs

Configuration: Core

- Most VUIDs checked here


Validation Settings

VK_LAYER_KHRONOS_validation

Standard Preset

Validation Areas

- [✓] Core
- [ ] Thread Safety
- [✓] Handle Wrapping
- [✓] Object Lifetime
- [✓] Stateless Parameter
- [ ] Shader-Based
- [ ] Synchronization
- [ ] Best Practices

Debug Action

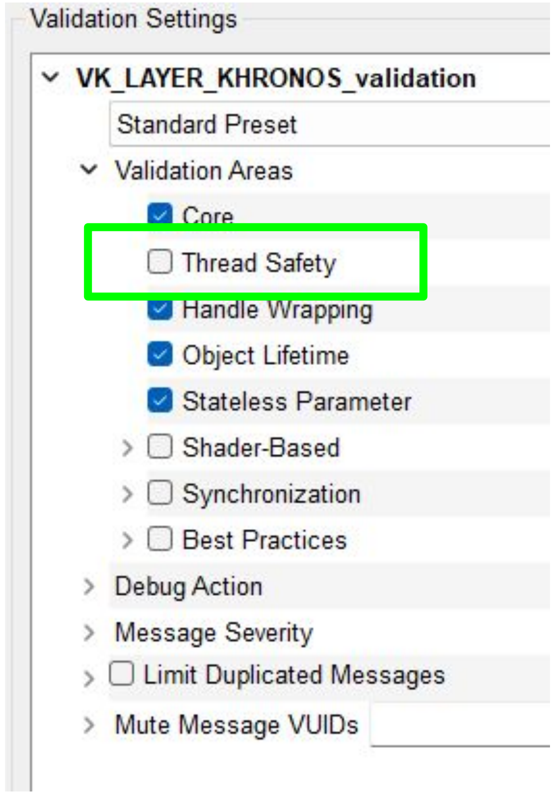
Message Severity

[ ] Limit Duplicated Messages

Mute Message VUIDs

Configuration: Thread Safety

- Checks external synchronization requirements

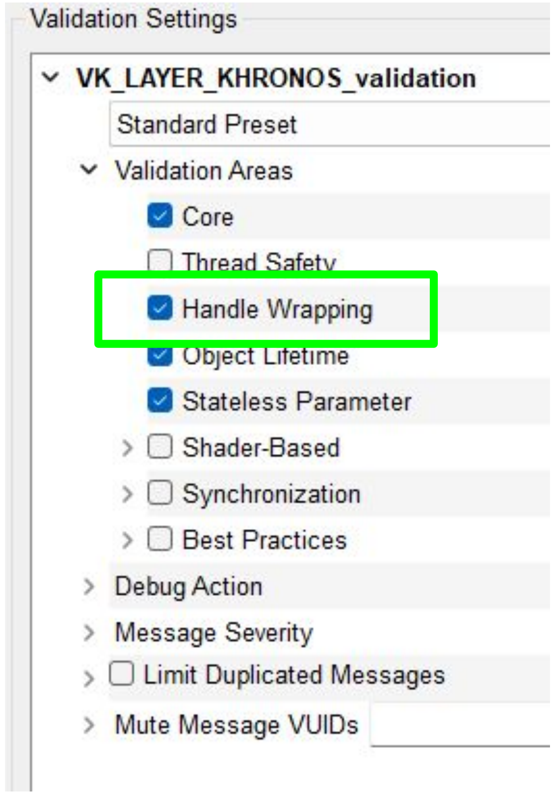


Validation Settings

- VK_LAYER_KHRONOS_validation
  - Standard Preset
  - Validation Areas
    - [✓] Core
    - [ ] Thread Safety
    - [✓] Handle Wrapping
    - [✓] Object Lifetime
    - [✓] Stateless Parameter
    - [ ] Shader-Based
    - [ ] Synchronization
    - [ ] Best Practices
  - Debug Action
  - Message Severity
  - [ ] Limit Duplicated Messages
  - Mute Message VUIDs

Configuration: Handle Wrapping

- Prevents handle reuse bugs

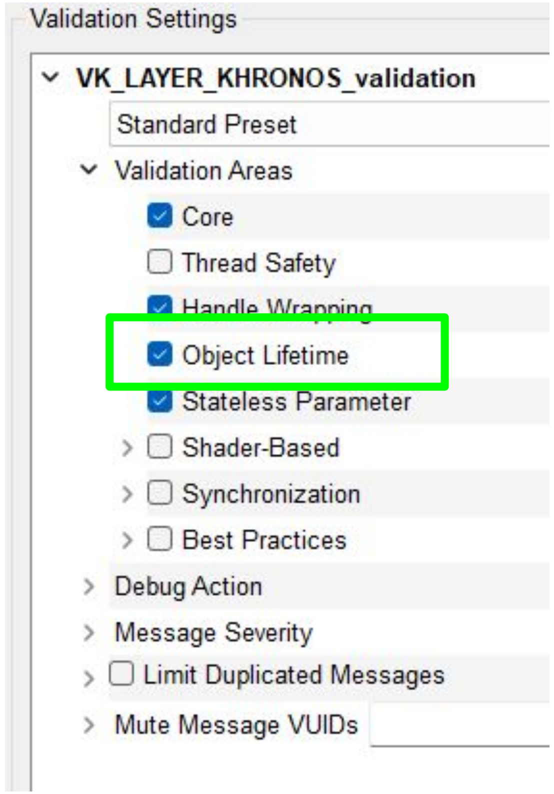


Validation Settings

- VK_LAYER_KHRONOS_validation
  - Standard Preset
  - Validation Areas
    - [✓] Core
    - [ ] Thread Safety
    - [✓] Handle Wrapping
    - [✓] Object Lifetime
    - [✓] Stateless Parameter
    - [ ] Shader-Based
    - [ ] Synchronization
    - [ ] Best Practices
  - Debug Action
  - Message Severity
  - [ ] Limit Duplicated Messages
  - Mute Message VUIDs

Configuration: Object Lifetime

- Detects use of destroyed objects

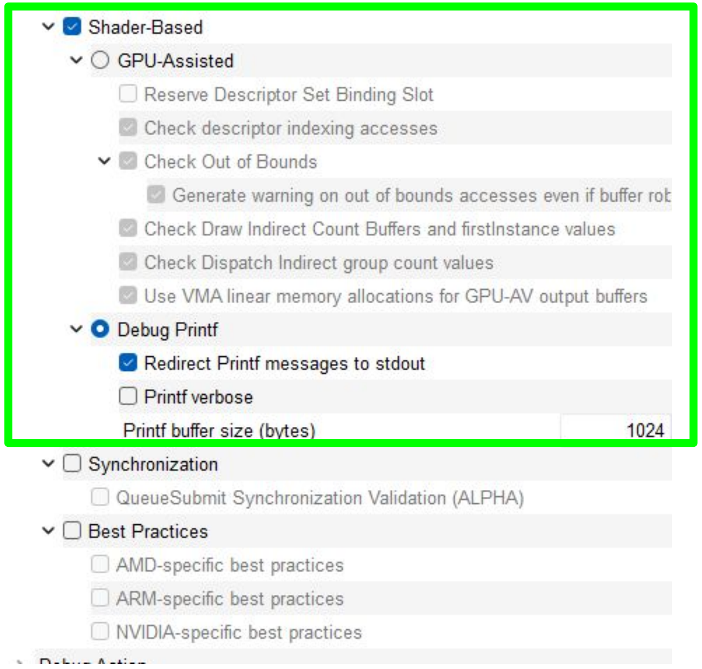


Validation Settings

- VK_LAYER_KHRONOS_validation
  - Standard Preset
  - Validation Areas
    - [✓] Core
    - [ ] Thread Safety
    - [✓] Handle Wrapping
    - [✓] Object Lifetime
    - [✓] Stateless Parameter
    - [ ] Shader-Based
    - [ ] Synchronization
    - [ ] Best Practices
  - Debug Action
  - Message Severity
  - [ ] Limit Duplicated Messages
  - Mute Message VUIDs

Configuration: Shader Based

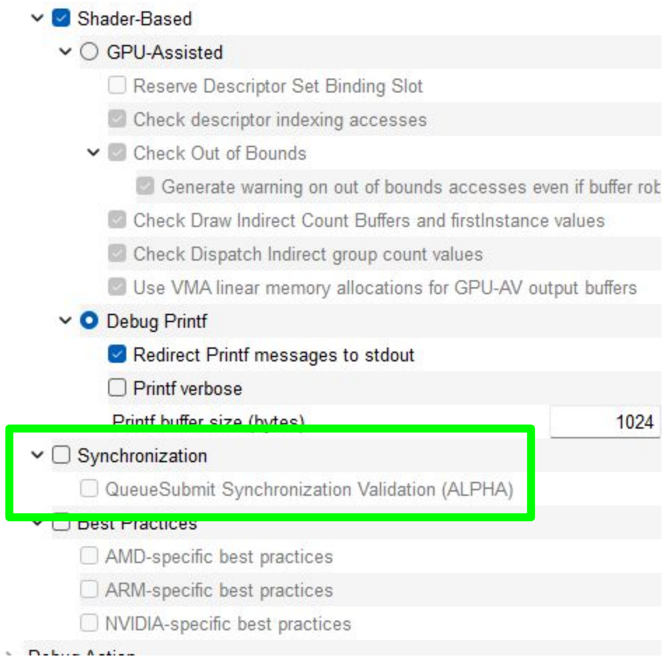
- GPU-Assisted
  - AKA: GPU-AV
  - Instruments SPIR-V to detect problems in shaders
  - Descriptor indexing
  - Buffer Device Address
  - Not supported on Mac
- DebugPrintf
  - Adds printf() functionality to shaders
  - Not supported on Mac



- [✓] Shader-Based
  - ○ GPU-Assisted
    - ☐ Reserve Descriptor Set Binding Slot
    - ✓ Check descriptor indexing accesses
    - ✓ Check Out of Bounds
      - ✓ Generate warning on out of bounds accesses even if buffer rob
    - ✓ Check Draw Indirect Count Buffers and firstInstance values
    - ✓ Check Dispatch Indirect group count values
    - ✓ Use VMA linear memory allocations for GPU-AV output buffers
  - ● Debug Printf
    - [✓] Redirect Printf messages to stdout
    - ☐ Printf verbose
    - Printf buffer size (bytes) 1024
- ☐ Synchronization
  - ☐ QueueSubmit Synchronization Validation (ALPHA)
- ☐ Best Practices
  - ☐ AMD-specific best practices
  - ☐ ARM-specific best practices
  - ☐ NVIDIA-specific best practices

Configuration: Synchronization

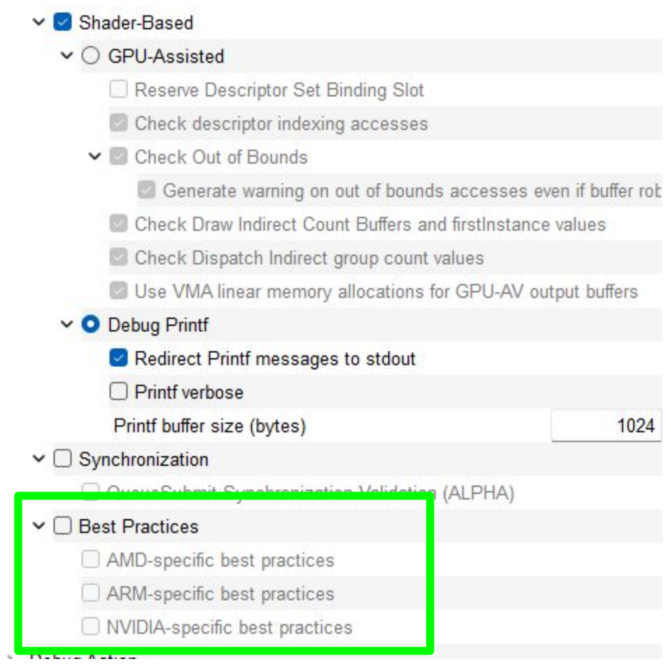
- Checks for correct Execution and Memory Dependencies
- vkCmdPipelineBarrier(), VkEvents, etc.



- ☑ Shader-Based
  - ○ GPU-Assisted
    - ☐ Reserve Descriptor Set Binding Slot
    - ☑ Check descriptor indexing accesses
    - ☑ Check Out of Bounds
      - ☑ Generate warning on out of bounds accesses even if buffer rob
    - ☑ Check Draw Indirect Count Buffers and firstInstance values
    - ☑ Check Dispatch Indirect group count values
    - ☑ Use VMA linear memory allocations for GPU-AV output buffers
  - ● Debug Printf
    - ☑ Redirect Printf messages to stdout
    - ☐ Printf verbose
    - Printf buffer size (bytes) 1024
- ☐ Synchronization
  - ☐ QueueSubmit Synchronization Validation (ALPHA)
- ☐ Best Practices
  - ☐ AMD-specific best practices
  - ☐ ARM-specific best practices
  - ☐ NVIDIA-specific best practices

Configuration: Best Practice

- Performance warnings
- Mixture of common and vendor-specific checks

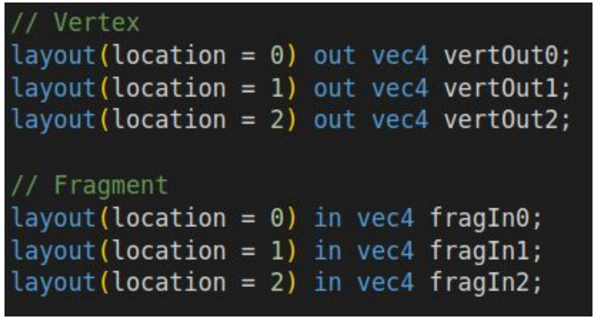


- [✓] Shader-Based
  - ○ GPU-Assisted
    - ☐ Reserve Descriptor Set Binding Slot
    - [✓] Check descriptor indexing accesses
    - [✓] Check Out of Bounds
      - [✓] Generate warning on out of bounds accesses even if buffer rob
    - [✓] Check Draw Indirect Count Buffers and firstInstance values
    - [✓] Check Dispatch Indirect group count values
    - [✓] Use VMA linear memory allocations for GPU-AV output buffers
  - ● Debug Printf
    - [✓] Redirect Printf messages to stdout
    - ☐ Printf verbose
    - Printf buffer size (bytes) 1024
- ☐ Synchronization
  - ☐ QueueSubmit Synchronization Validation (ALPHA)
- ☐ Best Practices
  - ☐ AMD-specific best practices
  - ☐ ARM-specific best practices
  - ☐ NVIDIA-specific best practices

Undefined Value

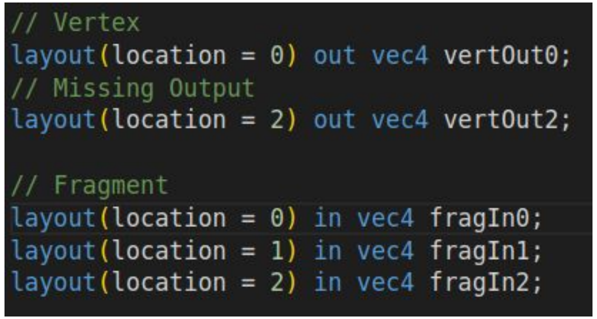
- Undefined **Value** != Undefined **Behavior**
- The app will never crash
- Your data might be garbage
- Great use of Best Practices layers

Undefined Behavior vs Best Practice



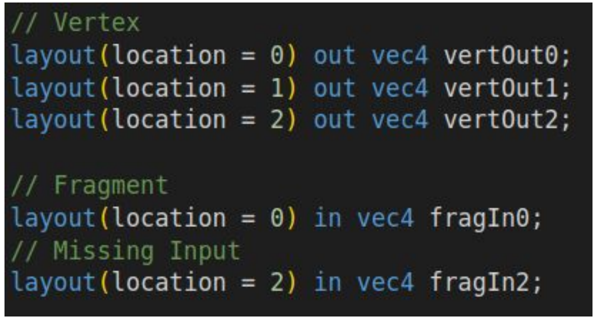
```
// Vertex
layout(location = 0) out vec4 vertOut0;
layout(location = 1) out vec4 vertOut1;
layout(location = 2) out vec4 vertOut2;

// Fragment
layout(location = 0) in vec4 fragIn0;
layout(location = 1) in vec4 fragIn1;
layout(location = 2) in vec4 fragIn2;
```



```
// Vertex
layout(location = 0) out vec4 vertOut0;
// Missing Output
layout(location = 2) out vec4 vertOut2;

// Fragment
layout(location = 0) in vec4 fragIn0;
layout(location = 1) in vec4 fragIn1;
layout(location = 2) in vec4 fragIn2;
```



```
// Vertex
layout(location = 0) out vec4 vertOut0;
layout(location = 1) out vec4 vertOut1;
layout(location = 2) out vec4 vertOut2;

// Fragment
layout(location = 0) in vec4 fragIn0;
// Missing Input
layout(location = 2) in vec4 fragIn2;
```

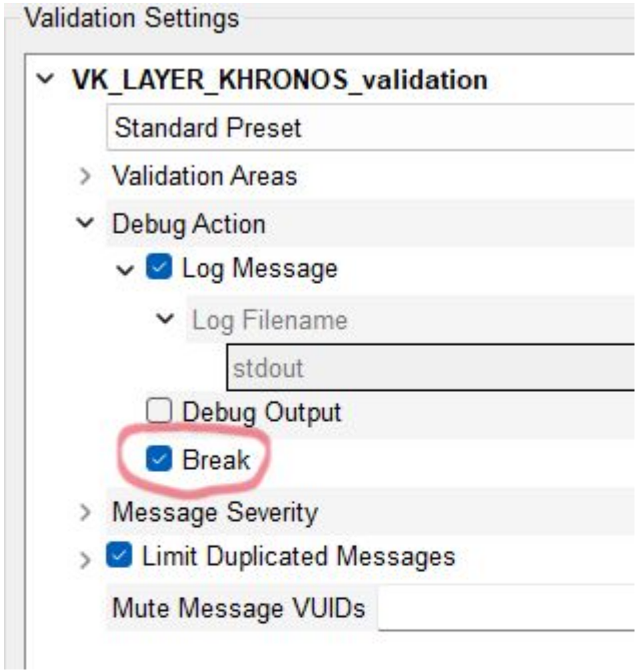
Normal

Error

Valid
But is this what you wanted?

Configuration: Break on error

- Will stop program when an error is detected
  - Calls `DebugBreak();` or `raise(SIGTRAP);`

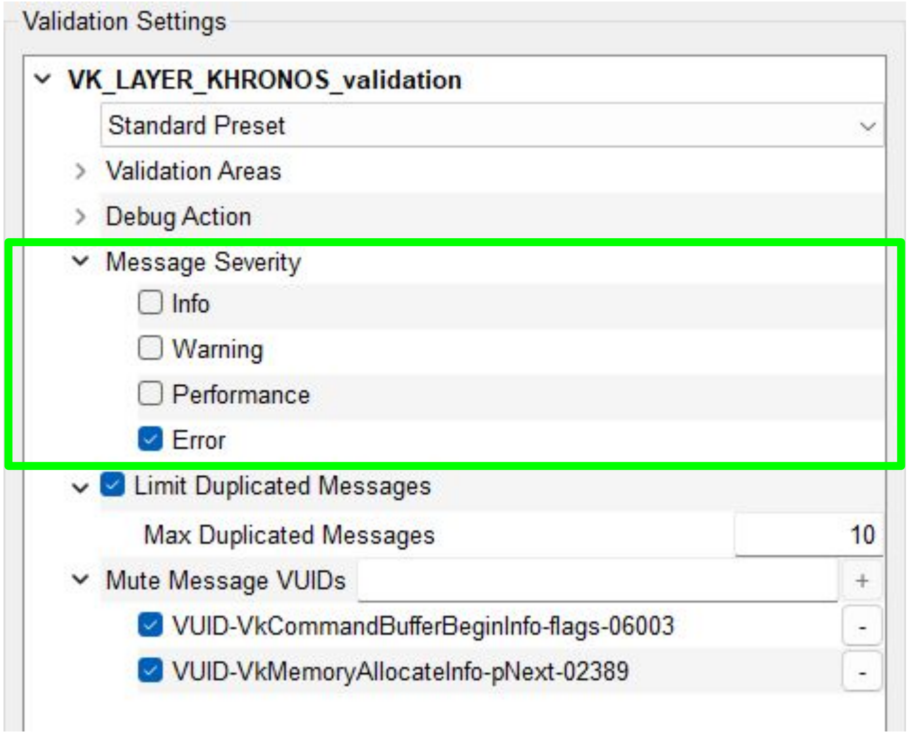


Validation Settings

- VK_LAYER_KHRONOS_validation
  - Standard Preset
  - Validation Areas
  - Debug Action
    - [✓] Log Message
      - Log Filename
        - stdout
    - [ ] Debug Output
    - [✓] Break
  - Message Severity
  - [✓] Limit Duplicated Messages
  - Mute Message VUIDs

Configuration: Limit message severity

- Almost all messages are ‘Error’
- Except Best Practices, which is ‘Performance’ and ‘Warning’

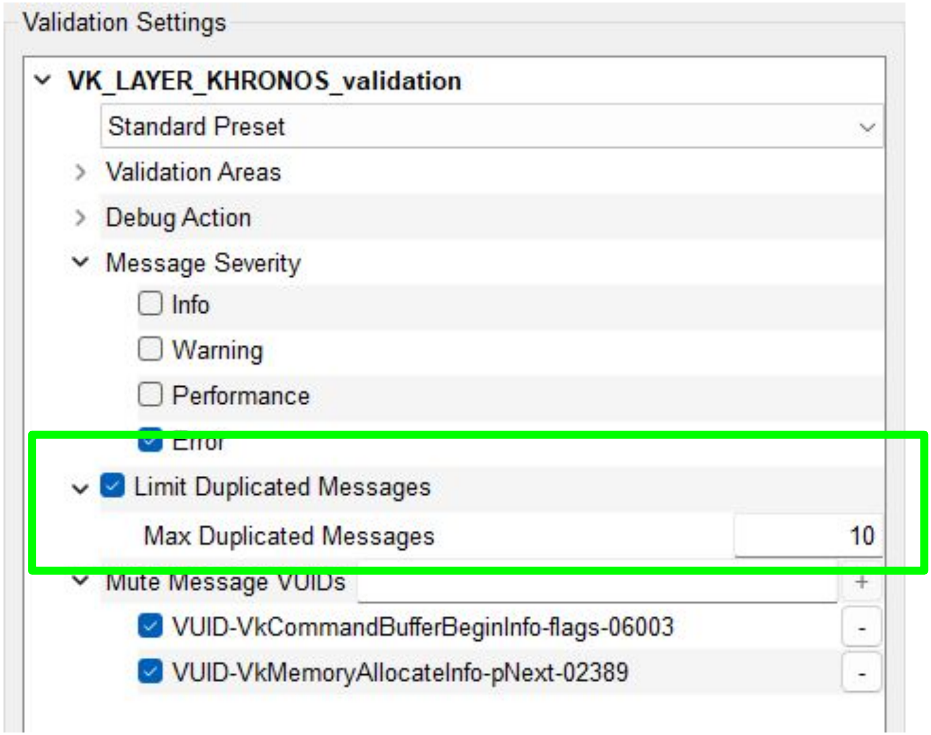


Validation Settings

- VK_LAYER_KHRONOS_validation
  - Standard Preset
  - Validation Areas
  - Debug Action
  - Message Severity
    - [ ] Info
    - [ ] Warning
    - [ ] Performance
    - [x] Error
  - [x] Limit Duplicated Messages
    - Max Duplicated Messages: 10
  - Mute Message VUIDs
    - [x] VUID-VkCommandBufferBeginInfo-flags-06003
    - [x] VUID-VkMemoryAllocateInfo-pNext-02389

Configuration: Limit repeated messages

- Limit times a message is repeated
  - Exact VUID string must match to count as a repeat



Validation Settings

VK_LAYER_KHRONOS_validation

Standard Preset

Validation Areas

Debug Action

Message Severity

- [ ] Info
- [ ] Warning
- [ ] Performance
- [x] Error

[x] Limit Duplicated Messages

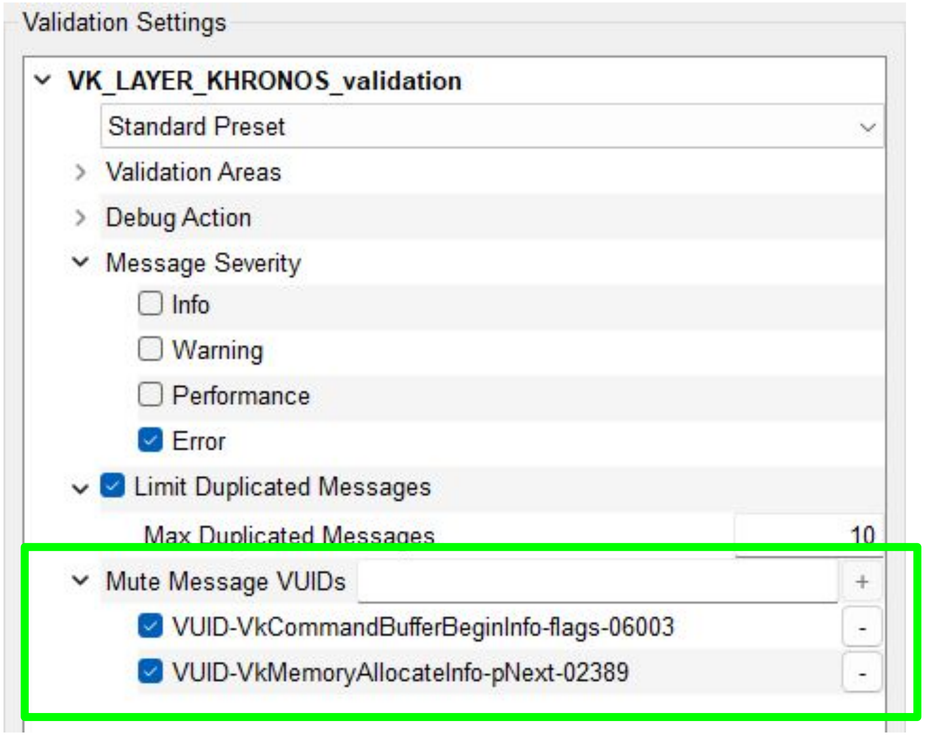
Max Duplicated Messages 10

Mute Message VUIDs

- [x] VUID-VkCommandBufferBeginInfo-flags-06003
- [x] VUID-VkMemoryAllocateInfo-pNext-02389

Configuration: Mute message

- Sometimes undefined behaviour works
- Sometimes the Validation Layers have bugs
- Sometimes the Vulkan Spec have bugs

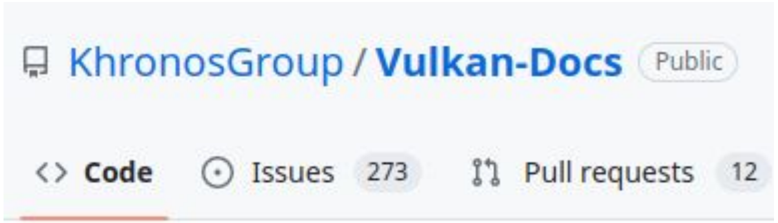


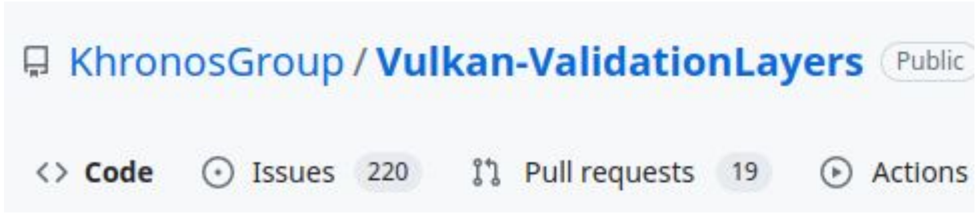
Validation Settings

VK_LAYER_KHRONOS_validation
- Standard Preset
- Validation Areas
- Debug Action
- Message Severity
  - [ ] Info
  - [ ] Warning
  - [ ] Performance
  - [✓] Error
- [✓] Limit Duplicated Messages
  - Max Duplicated Messages: 10
- Mute Message VUIDs
  - [✓] VUID-VkCommandBufferBeginInfo-flags-06003
  - [✓] VUID-VkMemoryAllocateInfo-pNext-02389

Spec bug vs Validation bug

- If not sure which to choose, feel free to put in Validation repo
    - We can always move it
    - Also check Khronos Slack, Discord, etc - the problem might be something simple on your end


KhronosGroup / **Vulkan-Docs** Public

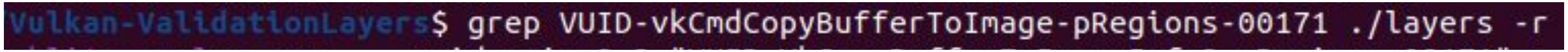

KhronosGroup / **Vulkan-ValidationLayers** Public

Advise - Read the spec

- “Read the spec early and often”
- Has all the answers
- Knowing how to look at the subset you care about is a skill

Advise - Fixing errors

- Run in a debugger and use the Break Debug Action
  - Almost all error checking occurs immediately in each Vulkan API call
  - Stack trace will take you to the part of your code causing the error
- Search in the source for the VUID string to see how it is validated

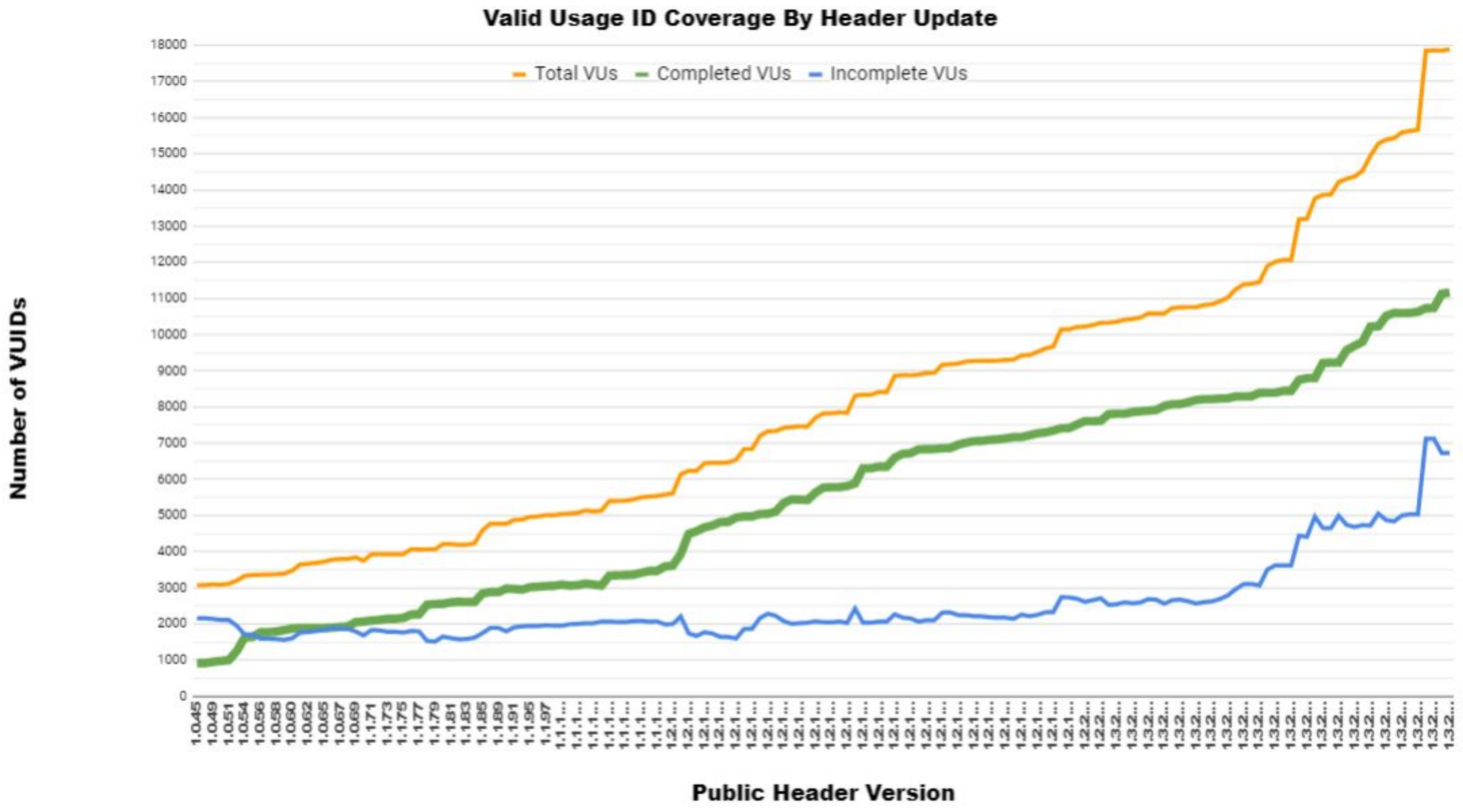


```
Vulkan-ValidationLayers$ grep VUID-vkCmdCopyBufferToImage-pRegions-00171 ./layers -r
```

Limitations

- Extensions and VUIDs are constantly added
  - Currently there are 18000 VUIDs!
  - 3000 at 1.0 launch
- Sometimes validating an extension is more difficult than writing or implementing it.
- Triage
  - Try to ensure new KHR or EXT extensions are fully validated
  - Respond to ‘Incomplete’ Issues to implement VUIDs that are needed by the community
  - Please submit an Issue on github if we’re missing something you need!

Limitations: Not all VUIDs checked



Limitations: Some VUIDs hard to check

- VK_DESCRIPTOR_BINDING_PARTIALLY_BOUND_BIT_EXT (aka ‘bindless’)
  - Only descriptors ‘dynamically used’ by a shader must be valid
  - Bindless descriptor sets may contain 1 million+ descriptors
  - But each shader invocation will only use a few of them
  - Descriptor index is calculated in the shader
    - CPU side code doesn’t know which descriptors to validate.
- Validating all descriptors results in large CPU overhead
- Many false positives due to validating unused descriptors
- Need to use GPU-AV to improve validation

Recent Improvements (last 12 months)

- Validation for new extensions
    - Video extensions, VK_EXT_mesh_shader, VK_KHR_descriptor_buffer, VK_KHR_dynamic_rendering, VK_EXT_pipeline_library, and more
    - Big THANK YOU to those who wrote validation for these extensions
- Synchronization validation Phase II
    - Multi-CommandBuffer and multi-Queue checking
- Increased SPIR-V runtime validation
- Improved performance for multithreaded applications
- GPU-AV performance improvements
- Adding UNASSIGNED validation errors to the spec (ongoing)
- Upgrade from C++11 to C++17

Upcoming Improvements

- Better error messages
- Better descriptor indexing checking using GPU-AV
  - Improve performance
  - Close gaps in error checking
- Better handling of timeline semaphores and ‘execution-time’ VUIDs
- Shader validation improvements
- Again, please submit an Issue on github if we’re missing something you need!
  - We also accept Pull Requests :)

Questions?