

# Using Vulkan Validation Effectively

Jeremy Gebben  
LunarG, Inc.



Presented at the February Khronos Vulkanised 2023 Conference



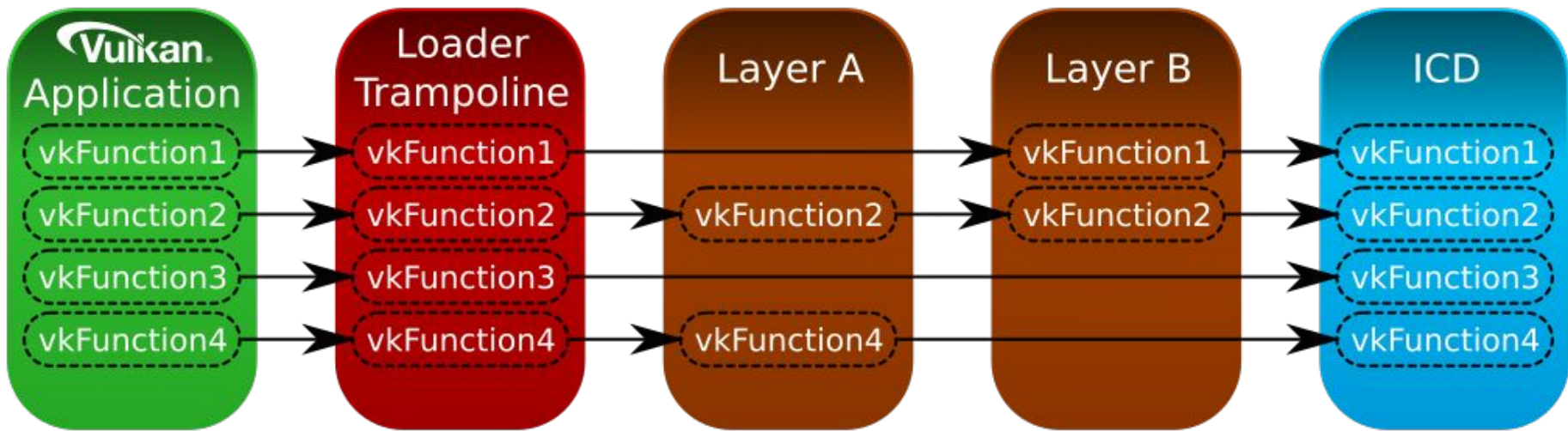
# Agenda

- What the validation does and how it benefits developers
- How to interpret and fix validation errors
- Configuration options to improve productivity
- Using the debug utilities extension
- Current limitations of validation
- Recent and upcoming improvements

# What is the Vulkan Validation Layer?

- A shared library containing almost all error checking for Vulkan
- OpenGL had many error code checks that drivers had to implement
  - Checks always enabled in drivers -> useless CPU overhead
  - Most checking was the similar in all drivers -> duplicated effort
  - Over time, OpenGL drivers added non-standard ways to disable this error checking in production code.
- Vulkan defined the [Loader/Layer Interface](#) to allow:
  - Validation during development only, no CPU overhead in released applications
  - Reuse of common checking code
  - Other types of tooling that wasn't defined during specification development
- *Historical note: At one time there were many separate validation layers, hence the plural name of the [Vulkan-ValidationLayers repository](#).*

# Vulkan Loader / Layer Interface



Attend the Vulkan-Loader presentation later today for more details

# Types of errors

- Usage - developer is using an API incorrectly
  - `memcpy(NULL, src_buffer, 100);`
  - Will almost always crash, because copying into the NULL address is an error
  - Vulkan Validation is supposed to find errors of this type
- Runtime - unsuccessful interaction between application and its environment
  - `ptr = calloc(1ULL << 31, 8);`
  - Allocating 4Gb *might* succeed or fail, it will depend on the current state of the system
  - Validation can help find some, but not all, of these errors (such as exceeding [device limits](#))
  - If an API call returns a `VkResult`, you should check it and handle errors.
- Suboptimal usage of the API
  - `ptr = calloc(0, 8);`
  - “If size is zero, the behavior is implementation defined (null pointer may be returned, or some non-null pointer may be returned that may not be used to access storage)” [cppreference.com](#)”
  - You cannot store anything into a 0 byte buffer, so why try allocate it?
  - Best Practices validation covers checking such as this

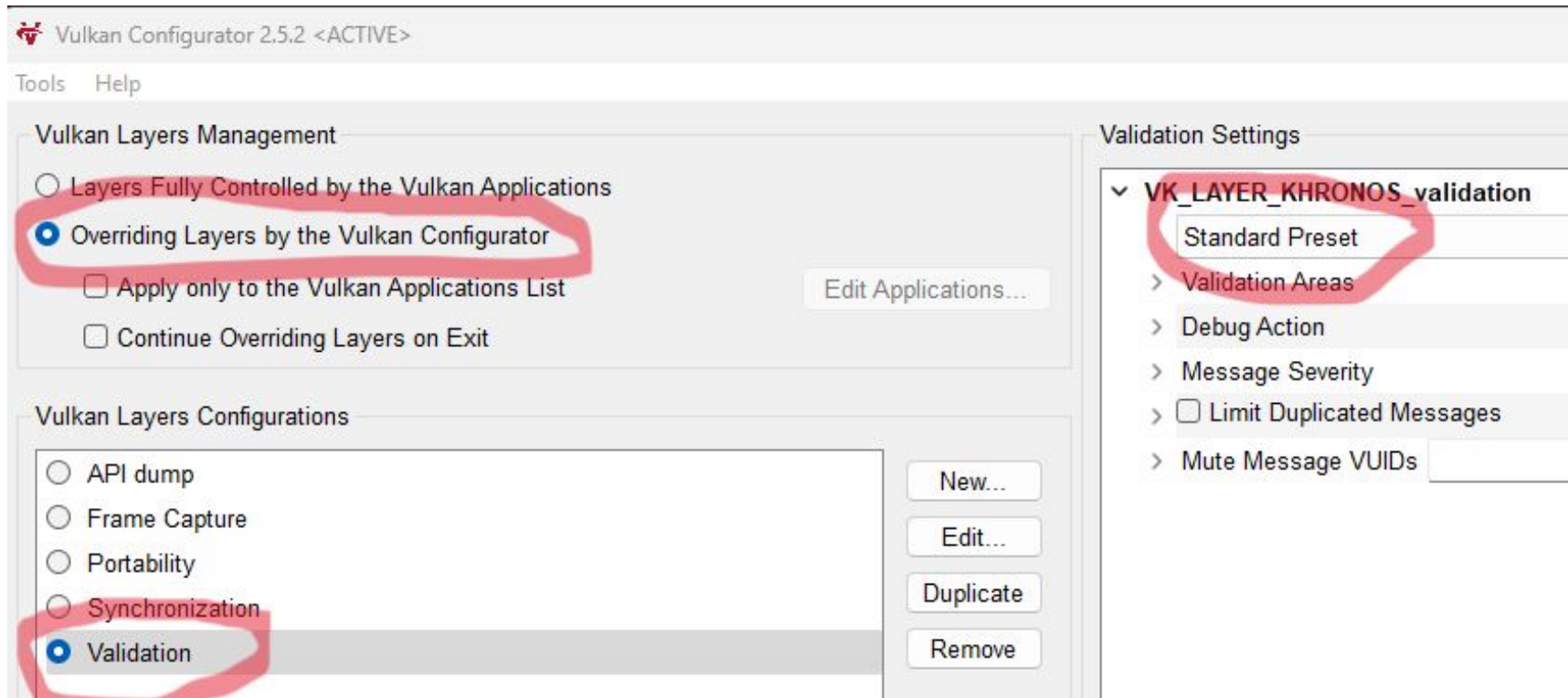


# Validation Quick Start

- Install the Vulkan SDK or OS-provided packages
- Run vkconfig (see next slide)
- From a shell:  

```
export VK_INSTANCE_LAYERS=VK_LAYER_KHRONOS_validation  
./your-application
```
- Note: for non-standard installs you may need to set VK\_LAYER\_PATH
  - It needs to be set to the directory containing VkLayer\_khronos\_validation.json
- You can also enable validation when calling vkCreateInstance()
  - Add the layer name to VkInstanceCreateInfo::ppEnabledLayerNames

# Validation Quick Start (Vulkan Configurator)



# An example error: vkcube -use\_staging

I added an error to a [portion of the vkcube source](#):

```
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://www.khronos.org/registry/vulkan/specs/1.3-extensions/html/vkspec.html#VUID-vkCmdCopyBufferToImage-pRegions-00171)
  Objects: 2
    [0] 0x56313fd28a00, type: 6, name: NULL
    [1] 0xd175b40000000013, type: 9, name: NULL
```

- demo->staging\_texture.tex\_width is 262144 bytes and the staging buffer was created based on that size.

# Validation Output: Valid Usage ID (VUID)

```
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://www.khronos.org/registry/vulkan/specs/1.3-extensions/html/vkspec.html#VUID-vkCmdCopyBufferToImage-pRegions-00171
)
  Objects: 2
    [0] 0x56313fd28a00, type: 6, name: NULL
    [1] 0xd175b40000000013, type: 9, name: NULL
```

- Almost every error in Vulkan has a Valid usage ID: VUID-\*
  - Unique, automatically generated number in the specification text
  - msgNum / MessageID is a hash of the VUID string, used for handling duplicate messages
- Some errors types are not in the specification
  - UNASSIGNED-\*: possible error identified by validation developers, should be moved to spec
  - UNASSIGNED-BestPractices-\*: best practices warnings
  - SYNC-\*: synchronization validation error

# Validation Output: 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://www.khronos.org/registry/vulkan/specs/1.3-extensions/html/vkspec.html#VUID-vkCmdCopyBufferToImage-pRegions-00171>)

**Objects: 2**

**[0] 0x56313fd28a00, type: 6, name: NULL**

**[1] 0xd175b40000000013, type: 9, name: NULL**

# Validation Output: 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://www.khronos.org/registry/vulkan/specs/1.3-extensions/html/vkspec.html#VUID-vkCmdCopyBufferToImage-pRegions-00171)
```

Objects: 2

[0] 0x56313fd28a00, type: 6, name: NULL

[1] 0xd175b40000000013, type: 9, name: NULL

- This takes you back to the section of the spec, for more information

# Fixing errors

- Fix the first error message first
  - Similar to with C/C++ compiler errors, the first error may cause subsequent 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 Vulkan-ValidationLayers source for the VUID string to see how it is validated
- Add object names and command buffer labels with the debug utils extension

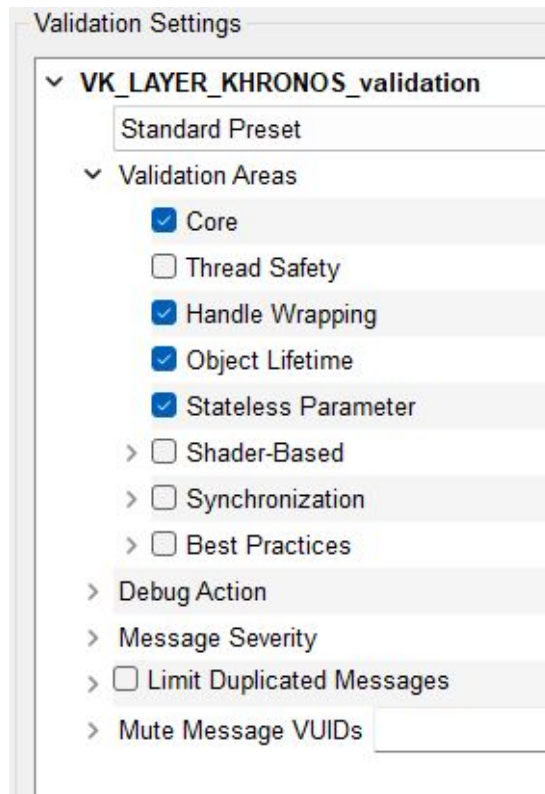
# Configuration options

- Configuring validation is complicated!
  - This section describes some useful settings, not an exhaustive guide
  - See the [documentation](#)
- Options:
  - UI: Vulkan Configurator (vkconfig) - **separate presentation later today!**
  - Config file: [vk\\_layer\\_settings.txt](#)
  - Programatically: [VK\\_EXT\\_layer\\_settings](#)
  - Environment variables (not all options supported)



# Configuration: Validation areas (1)

- Validation is split up into several areas to reduce performance overhead
- Stateless
  - Checks simple VUIDs that don't require expensive state tracking
  - In Vulkan spec: Valid Usage (Implicit) and a few others
- Core
  - Most VUIDs checked here
- Thread Safety
  - Checks [external synchronization requirements](#)
- Handle Wrapping
  - Prevents handle reuse bugs
- Object Lifetime
  - Detects use of destroyed objects



# Configuration: Validation areas (2)

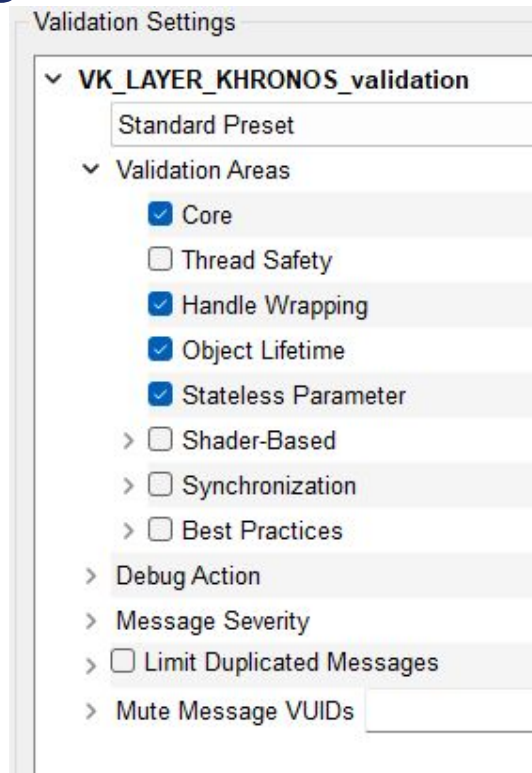
- Shader Based: [GPU-Assisted](#)
  - AKA: GPU-AV
  - Instruments SPIR-V to detect problems in shaders
  - Descriptor indexing
  - Buffer Device Address
  - Not supported on Mac
- Shader Based: [DebugPrintf](#)
  - Adds printf() functionality to shaders
  - Not supported on Mac
- [Synchronization](#)
  - Checks for correct [Execution and Memory Dependencies](#)
  - vkCmdPipelineBarrier(), VkEvents, etc.
- [Best Practices](#)
  - Performance warnings
  - Mixture of common and vendor-specific checks

The screenshot shows the Vulkan Validation Configuration dialog box. The 'Shader-Based' section is expanded and checked. Under 'GPU-Assisted', several options are checked: 'Check descriptor indexing accesses', 'Check Out of Bounds' (which includes 'Generate warning on out of bounds accesses even if buffer robustness is not enabled'), 'Check Draw Indirect Count Buffers and firstInstance values', 'Check Dispatch Indirect group count values', and 'Use VMA linear memory allocations for GPU-AV output buffers'. The 'Debug Printf' section is also expanded and checked, with 'Redirect Printf messages to stdout' checked and 'Printf buffer size (bytes)' set to 1024. The 'Synchronization' and 'Best Practices' sections are currently unchecked.

- ✓ 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 robustness is not enabled
    - ☒ 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)
- ✓ ☐ Synchronization
  - ☐ QueueSubmit Synchronization Validation (ALPHA)
- ✓ ☐ Best Practices
  - ☐ AMD-specific best practices
  - ☐ ARM-specific best practices
  - ☐ NVIDIA-specific best practices

# Configuration: Validation area settings

- **Use vkconfig presets**
  - Commonly used and tested configurations
- In vk\_layer\_settings.txt
  - khronos\_validation.enables
  - khronos\_validation.disables
- Environment variables
  - VK\_LAYER\_ENABLES and VK\_LAYER\_DISABLES
- **Don't enable all areas at once (it will be slow), pick one of**
  - Core
  - Shader-Based
  - Synchronization
  - Best Practices
- Fix errors in each area, then run Core / Standard Preset again

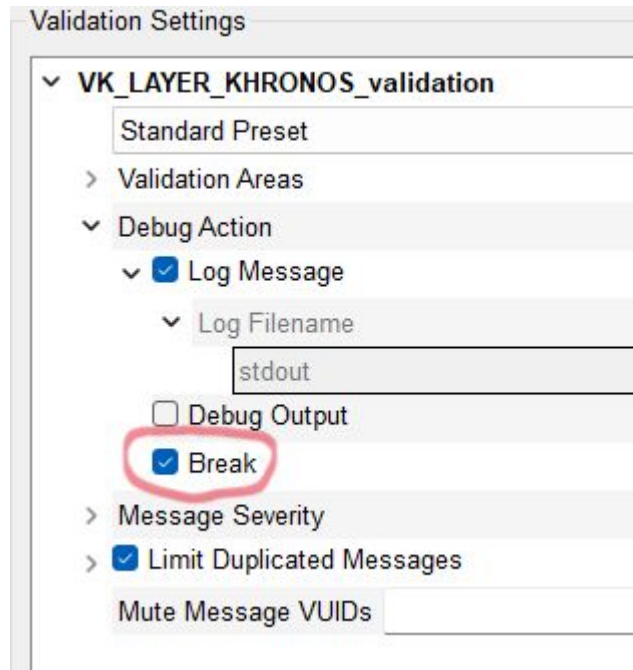


# Configuration: Break on error

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

```
# vk_layer_settings.txt
```

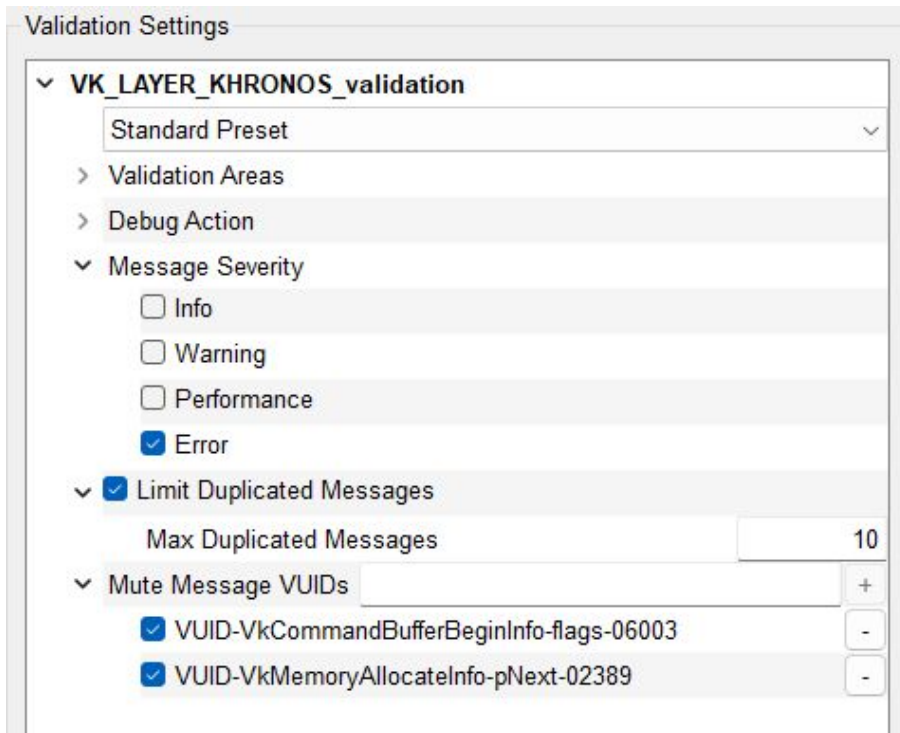
```
khronos_validation.debug_action = VK_DBG_LAYER_ACTION_BREAK
```



# Configuration: Limit repeated messages

- Limit message severity
  - Almost all messages are 'Error'
  - Except Best Practices, which is 'Performance' and 'Warning'
- Limit times a message is repeated
  - Exact VUID string must match to count as a repeat
  - Env var: VK\_LAYER\_DUPLICATE\_MESSAGE\_LIMIT
- Suppress individual error messages entirely
  - Env var: VK\_LAYER\_MESSAGE\_ID\_FILTER

```
# vk_layer_settings.txt
khronos_validation.report_flags = error
khronos_validation.enable_message_limit = true
khronos_validation.duplicate_message_limit = 10
khronos_validation.message_id_filter = <comma
separated list>
```



# Debug Utilities Extension

- Debug utilities extension [VK\\_EXT\\_debug\\_utils](#)
- 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



# Debug Utilities extension: Object naming

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

- 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.

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

Objects - 2

```
Object[0] - VK_OBJECT_TYPE_COMMAND_BUFFER, Handle 0x5566702c9f60, Name "PrepareCB"  
Object[1] - VK_OBJECT_TYPE_BUFFER, Handle 0x9fde6b0000000014, Name "TexBuffer(lunarg.ppm)"
```

# Debug Utilities extension: Command buffer labels

```
typedef struct VkDebugUtilsLabelEXT {  
    VkStructureType    sType;  
    const void*        pNext;  
    const char*        pLabelName;  
    float              color[4];  
} VkDebugUtilsLabelEXT;  
  
void vkCmdBeginDebugUtilsLabelEXT(  
    VkCommandBuffer commandBuffer,  
    const VkDebugUtilsLabelEXT* pLabelInfo);
```

- 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: 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)

# Debug Utils: vkcube code

```
demo_push_cb_label(demo, demo->cmd, NULL, "StagingBufferCopy(%d)", i);
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);
demo_pop_cb_label(demo, demo->cmd); // "StagingBufferCopy"
```

# 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(lunarg.ppm)**, 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(lunarg.ppm)**]) 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://www.khronos.org/registry/vulkan/specs/1.3-extensions/html/vkspec.html#VUID-vkCmdCopyBufferToImage-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(lunarg.ppm)**"

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 }

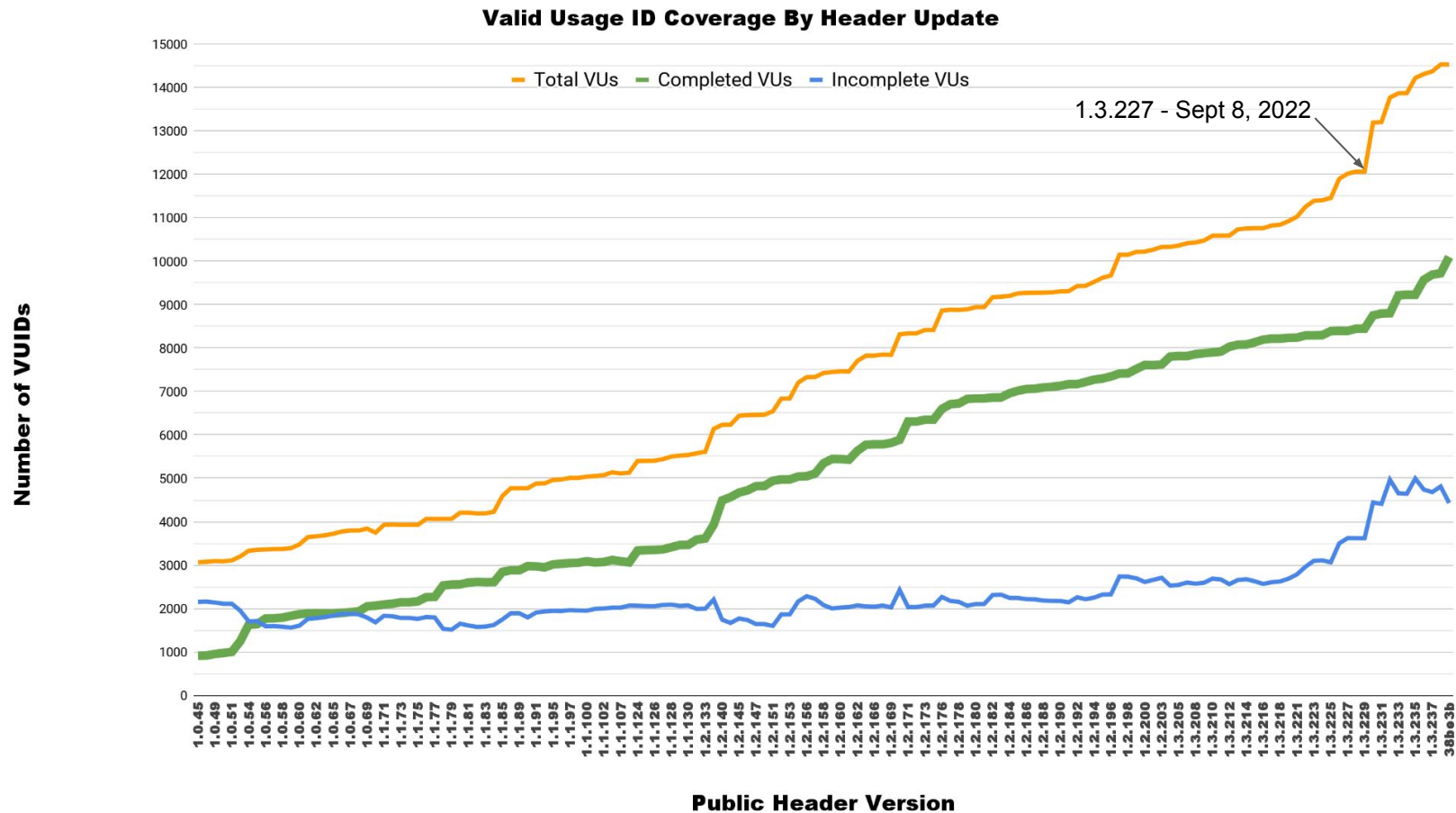
Label[2] - Prepare { 0.000000, 0.000000, 0.000000, 0.000000 }

# Limitations

- Extensions and VUIDs are constantly added
  - Currently there are 14000+ VUIDs!
- Sometimes validating an extension is more difficult than writing or implementing it.
- Vendor extension validation is entirely up to the vendor
- 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: Extension VUID coverage

EXTENSION	CHECKED	TOTAL	COVERAGE
core	1879	2359	79.65%
VK_VERSION_1_3	1716	2332	73.58%
<i>VK_NV_ray_tracing</i>	953	1555	61.29%
VK_VERSION_1_1	960	1292	74.30%
VK_EXT_mesh_shader	295	1190	24.79%
<i>VK_NV_mesh_shader</i>	491	1178	41.68%
VK_KHR_ray_tracing_pipeline	608	1060	57.36%
VK_VERSION_1_2	740	1004	73.71%
VK_KHR_acceleration_structure	516	912	56.58%
VK_KHR_dynamic_rendering	408	601	67.89%
VK_KHR_synchronization2	473	598	79.10%
VK_KHR_surface	418	547	76.42%
VK_KHR_copy_commands2	345	386	89.38%
VK_EXT_transform_feedback	233	319	73.04%
VK_EXT_extended_dynamic_state	166	316	52.53%

# 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 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?

<https://www.lunarg.com/news-insights/white-papers/using-vulkan-validation-effectively-feb2023/>







Help Us Improve the  
Vulkan SDK and Ecosystem

Share Your Feedback

**Take the LunarG annual developer's survey**

- Survey results are tabulated
- Shared with the Vulkan Working Group
- Actions are assigned
- Results are reported

**Survey closes February 27, 2023**



<https://www.surveymonkey.com/r/PVM92RH>