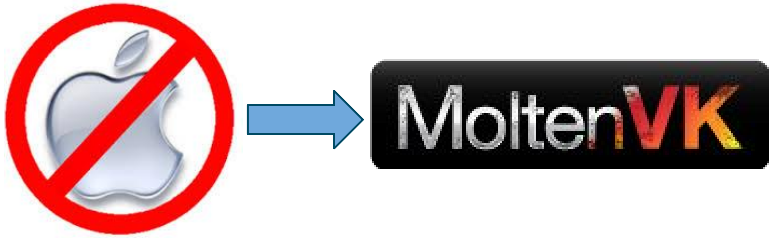
Getting Started with Vulkan

Loader, Layers, and Other Resources

Before You Begin

Supported Operating Systems:

- Windows 7+ (32-bit or 64-bit)
- Linux
  - Ubuntu 14.04 or newer
  - Fedora Core 23 or newer
- Android Nougat or newer



C/C++ Compiler (minimum):

- Visual Studio 2013
- GCC 4.8.1
- Clang 3.3

Tools:

- Python 3
- CMake 3.0+
- Git

Graphics Hardware

Desktop Devices (minimum):

AMD:	Radeon HD 77xx	[Windows]
	Radeon R9	[Linux]
Intel:	Skylake	[Windows]
	Ivy Bridge	[Linux]
Nvidia:	GeForce 600 series	

Android Nougat (or newer) Devices:

ARM:	Mali T760
Imagination:	PowerVR Series 6
Nvidia:	Tegra K1
Qualcomm:	Adreno 500

NOTE: This is an approximate list. Contact your HW provider for up-to-date support info.

Vulkan SDKs Contain Useful Content

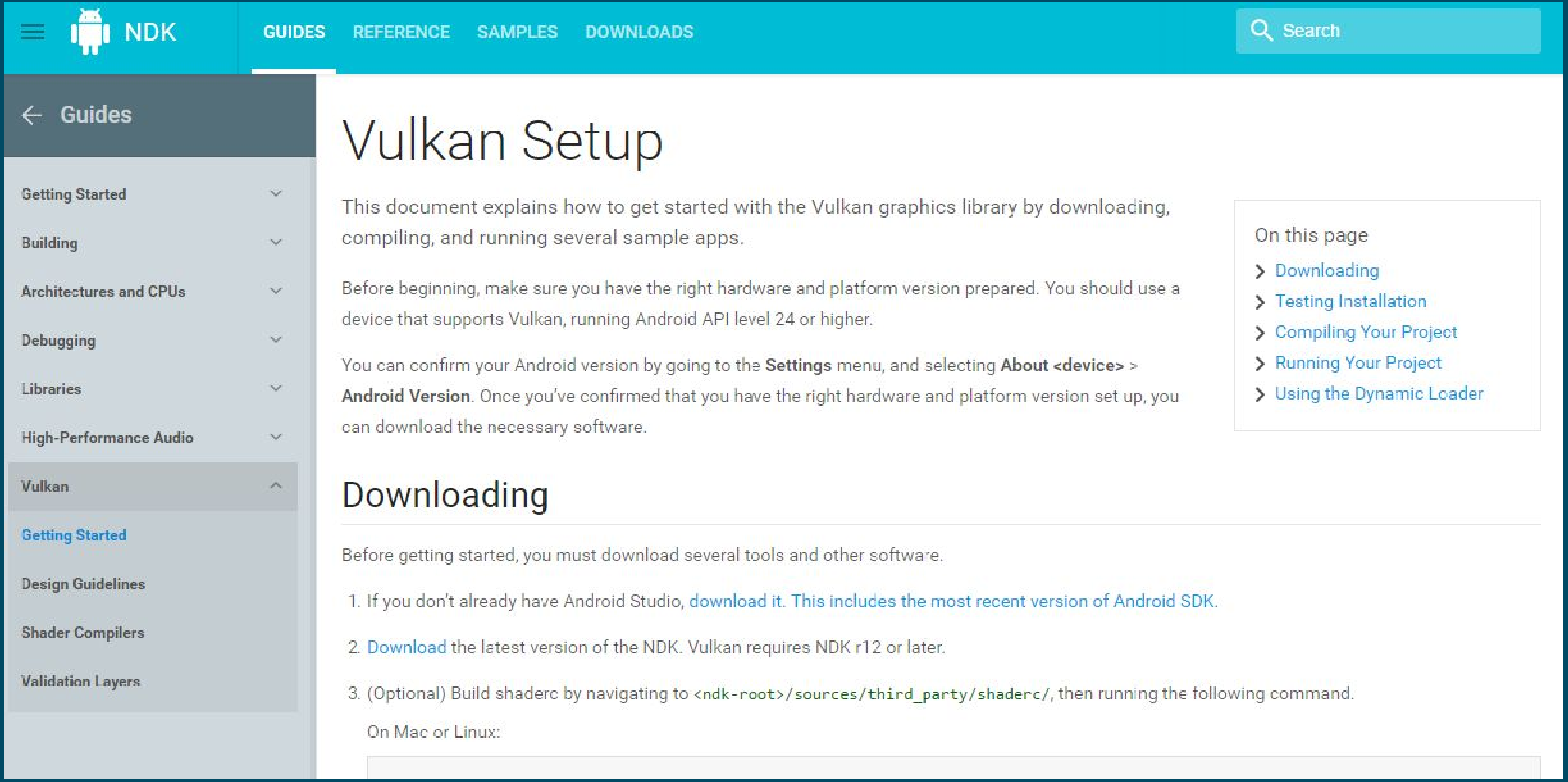
- Latest Documentation
- Validation layers
- Samples
- Other Useful Tools

Android Vulkan SDK

https://developer.android.com/ndk/guides/graphics/getting-started.html

Targets:
- Android Nougat Devices

Requires:
- Android Studio 2.1+



Vulkan Setup

This document explains how to get started with the Vulkan graphics library by downloading, compiling, and running several sample apps.

Before beginning, make sure you have the right hardware and platform version prepared. You should use a device that supports Vulkan, running Android API level 24 or higher.

You can confirm your Android version by going to the **Settings** menu, and selecting **About <device>** > **Android Version**. Once you've confirmed that you have the right hardware and platform version set up, you can download the necessary software.

Downloading

Before getting started, you must download several tools and other software.

1. If you don't already have Android Studio, download it. This includes the most recent version of Android SDK.
2. Download the latest version of the NDK. Vulkan requires NDK r12 or later.
3. (Optional) Build shaderc by navigating to `<ndk-root>/sources/third_party/shaderc/`, then running the following command.

   On Mac or Linux:

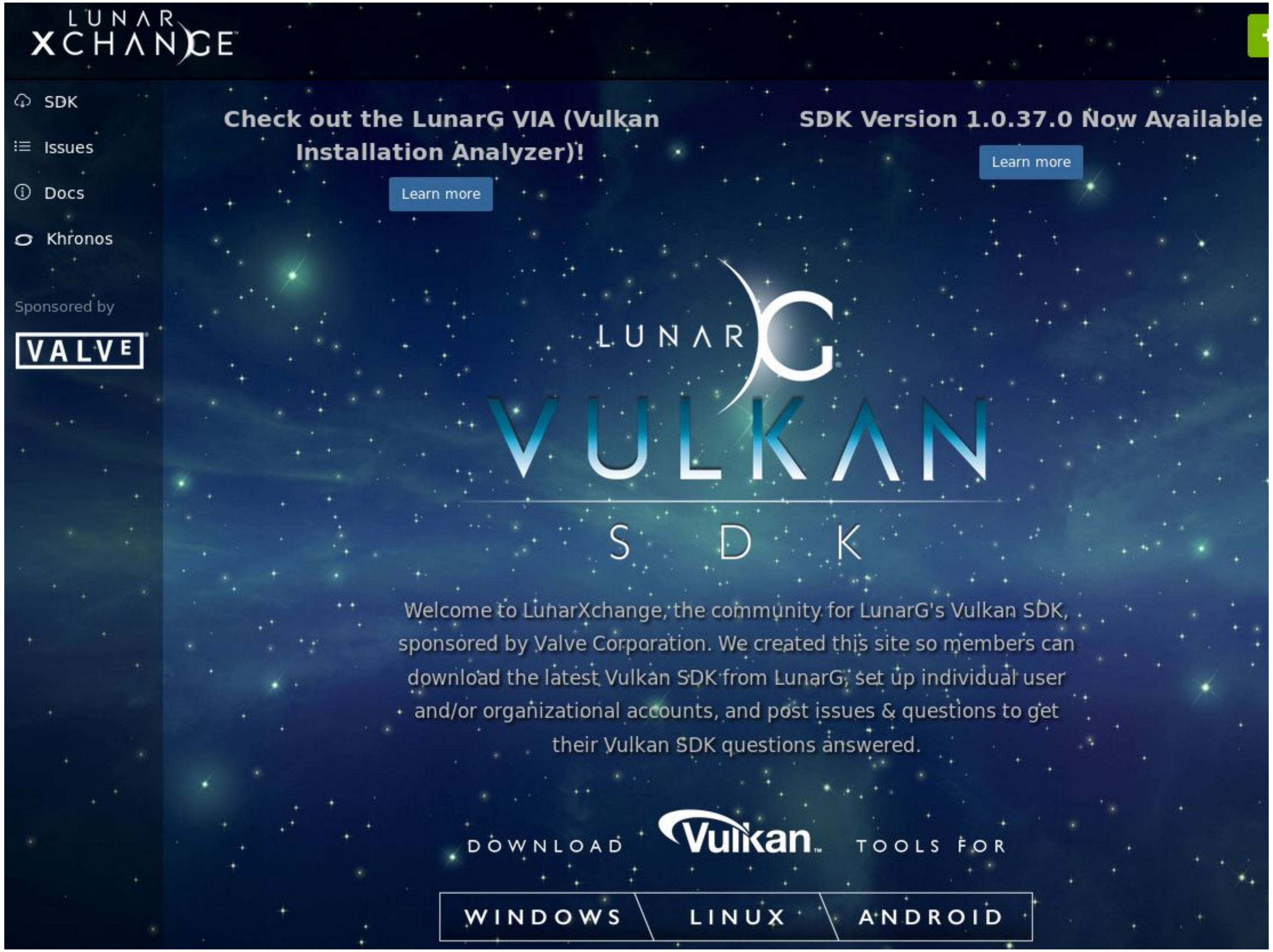
Desktop Vulkan SDK

https://vulkan.lunarg.com/

Targets:
- Windows
- Linux

Also Includes:
- Latest Vulkan Loader/Run-time
- Additional Layers
  - Screenshot
  - Trace/replay
- Tutorial

Released every 4 - 6 weeks



Check out the LunarG VIA (Vulkan Installation Analyzer)!

SDK Version 1.0.37.0 Now Available

Welcome to LunarXchange, the community for LunarG's Vulkan SDK, sponsored by Valve Corporation. We created this site so members can download the latest Vulkan SDK from LunarG, set up individual user and/or organizational accounts, and post issues & questions to get their Vulkan SDK questions answered.

Vulkan Installation Analyzer (VIA)

Validates your desktop setup

- Determines available Vulkan drivers, runtime, and layers
- Captures system state into HTML

Run it:

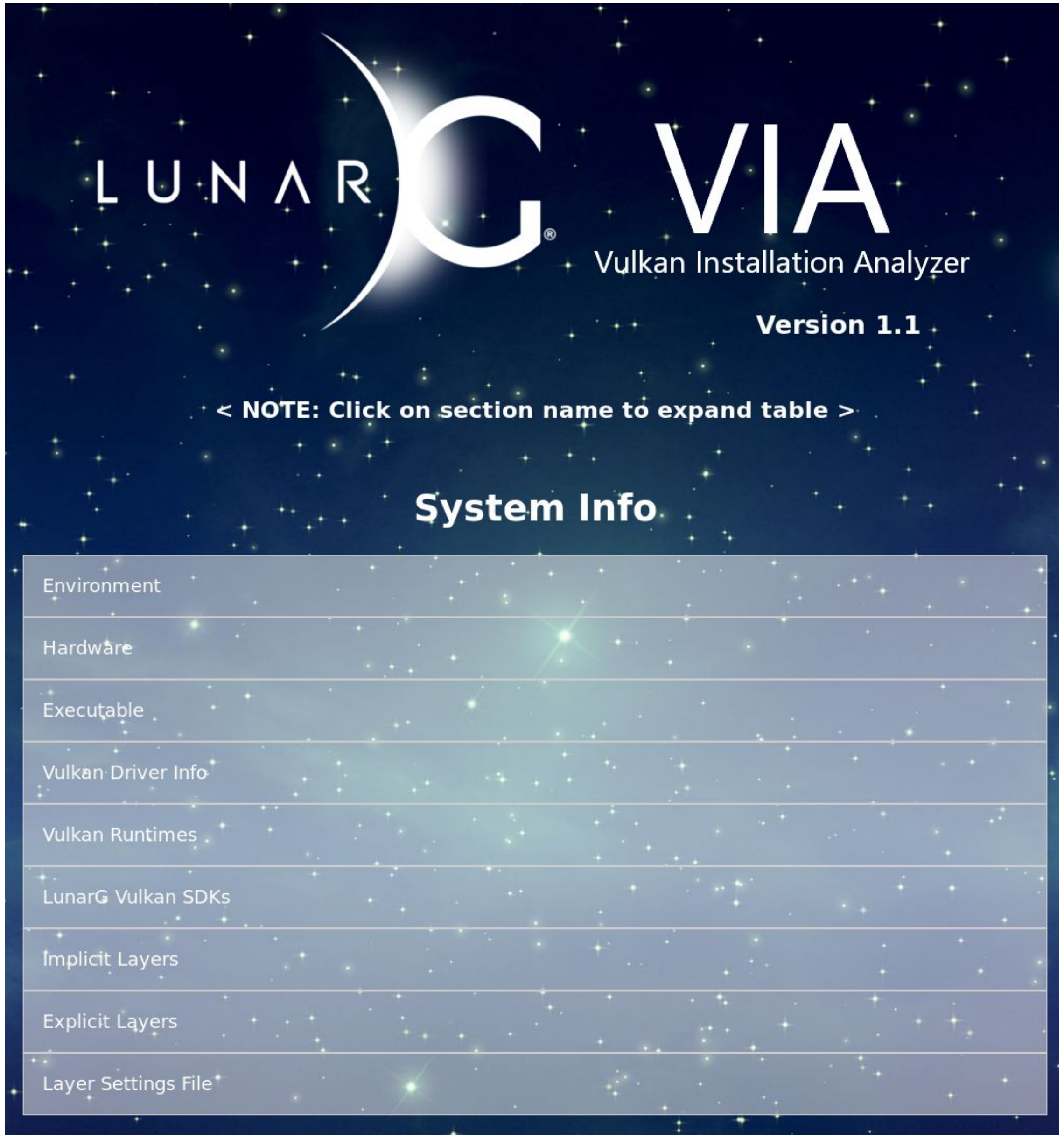
- After installing SDK or new drivers
- When you file a bug

VIA Results

Overall Result spit out to Command-line (detailed in Readme file):

```
  SUCCESS: Validation completed properly
  ERROR: Failed to find Vulkan Driver JSON
  ERROR: Failed to find Vulkan Driver Lib
  ERROR: Vulkan failed to find a compatible driver
  ...
```

HTML contains details (in collapsible sections)



LUNARG VIA

Vulkan Installation Analyzer

Version 1.1

< NOTE: Click on section name to expand table >

System Info

Environment

Hardware

Executable

Vulkan Driver Info

Vulkan Runtimes

LunarG Vulkan SDKs

Implicit Layers

Explicit Layers

Layer Settings File

Vulkan Loader

The gateway to Vulkan on a user's system

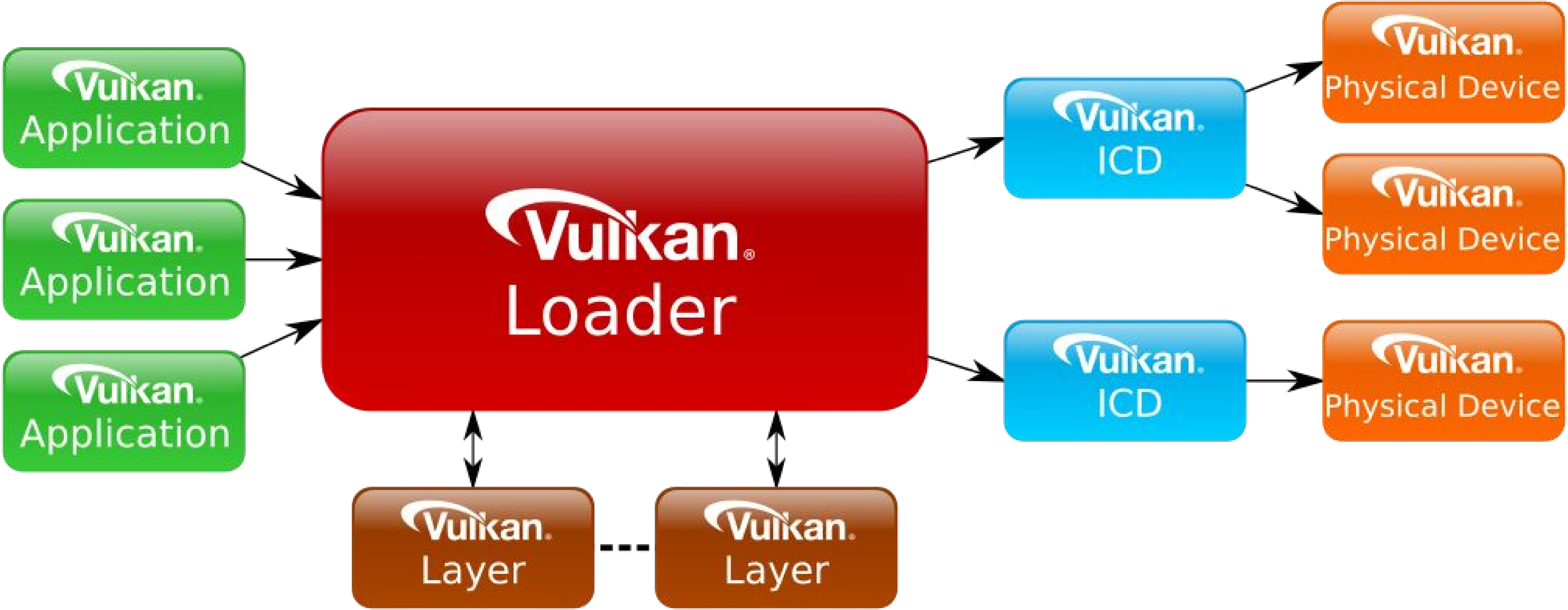
Similar to OpenGL loader typically provided by OS Vendors, but:

- Owned by Khronos
- Updated regularly (4-6 weeks)
- Desktop Loader is Open Source on GitHub
  - Largely developed by LunarG (funded by Valve)
  - Community support/bug-fixes greatly appreciated and accepted
    - NOTE: CLA required for any contributions

How does it get installed on a system?

- Drivers
- Applications
- SDKs

Vulkan Loader (High Level Interfaces)



LUNARG

Okay, Really Vulkan Loaders (Plural)

Intent is only one loader to rule them all

Two different loaders:

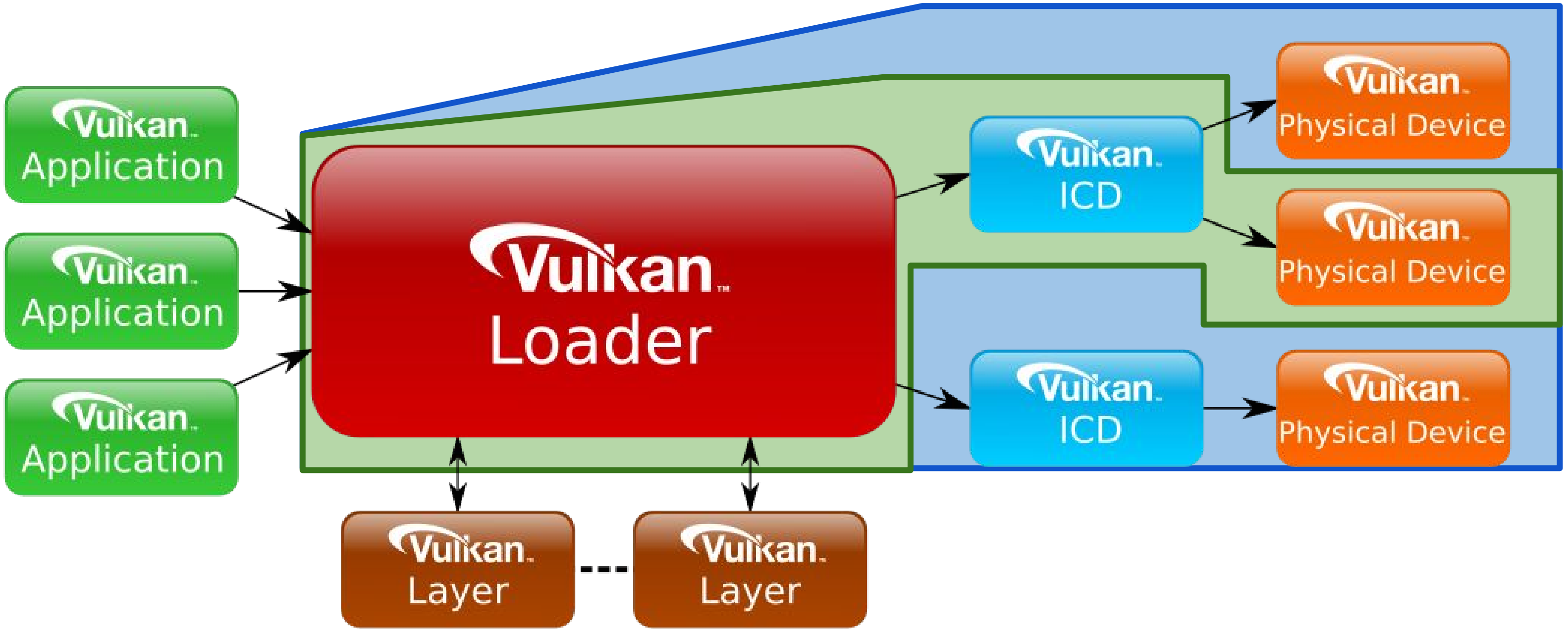
- Desktop Loader
  - Same source used for Linux/Windows
  - Open Source (in Github)
- Android Loader
  - Nougat+ devices
  - Closed Source

But one loader interface design (in GitHub and LunarG Vulkan SDK)
[Link provided at end]

Object Groups

Instance: High-level construct (similar to GL Context)
Works with all ICDs and Physical Devices
Includes: VkInstance and VkPhysicalDevice

Device: Logical accessor to a particular Physical Device (through a particular ICD)
Includes: VkDevice, VkQueue, VkCmdBuffer and other objects derived from these

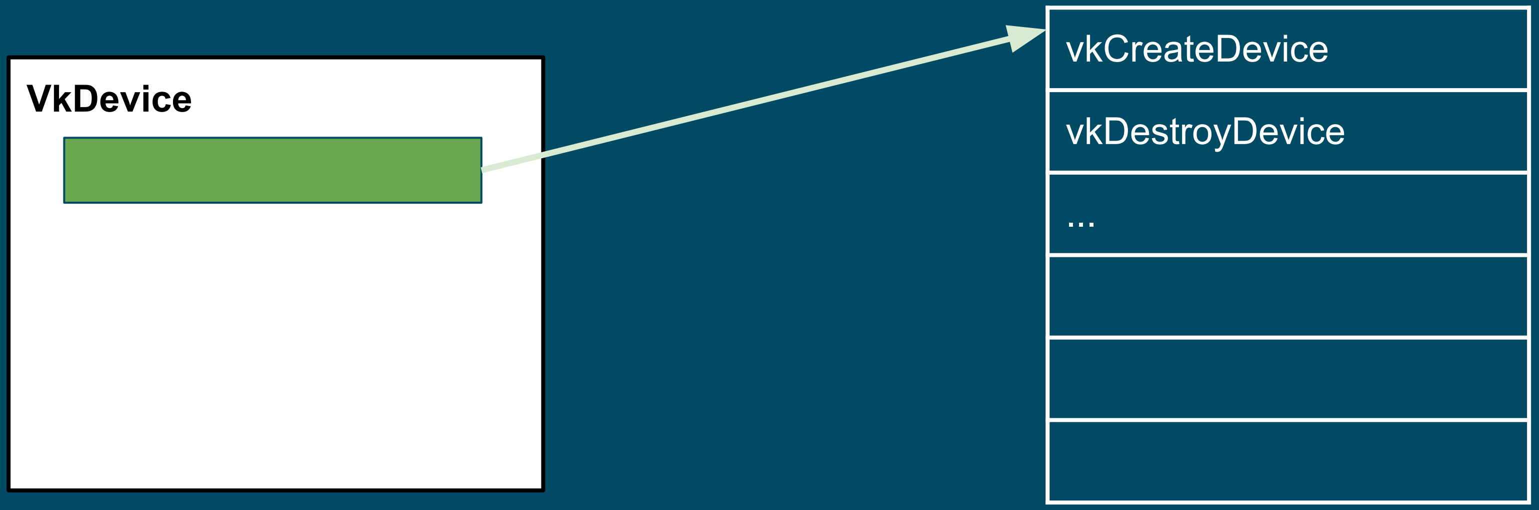


Dispatchable Objects

- Most commands take an opaque dispatchable object as first parameter

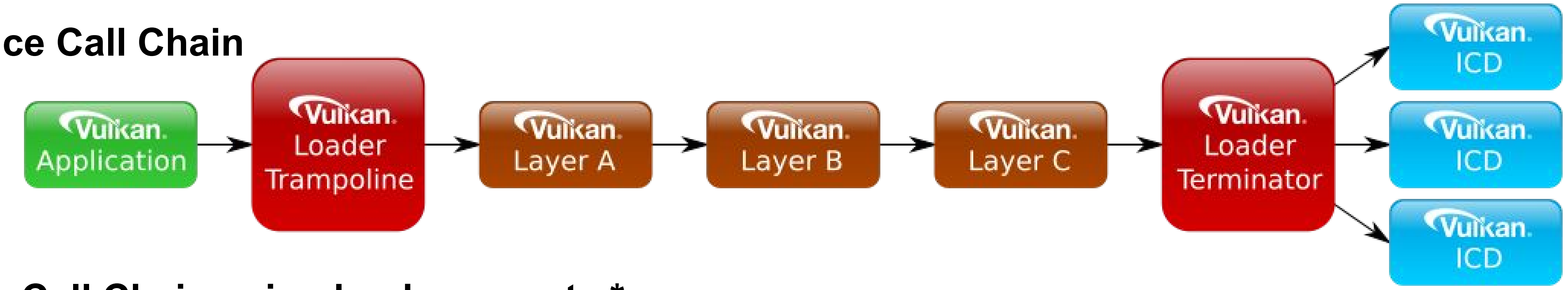
```
VkResult vkGetEventStatus(VkDevice device, VkEvent event);
```

- First field in each dispatchable object is a dispatch table pointer
  - Used by loader trampoline code
  - ICDs must reserve first element of created objects for a pointer the loader will fill in



Call Chains

Instance Call Chain



Application

Loader Trampoline

Layer A

Layer B

Layer C

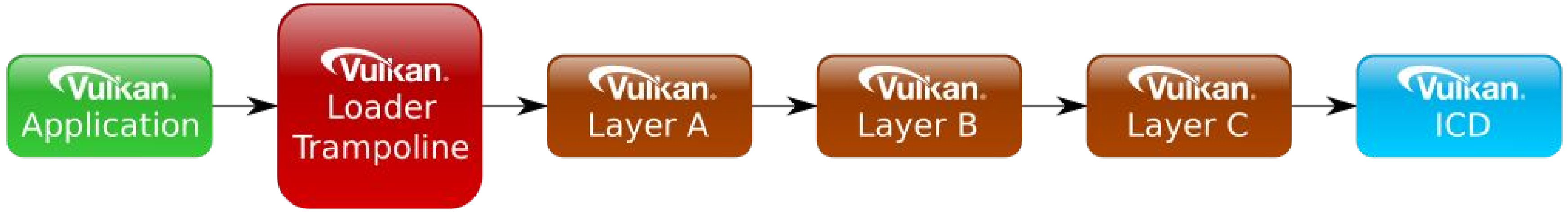
Loader Terminator

ICD

ICD

ICD

Device Call Chain using loader exports *



Application

Loader Trampoline

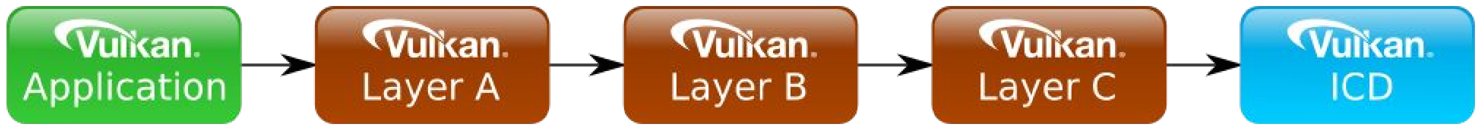
Layer A

Layer B

Layer C

ICD

Device Call Chain using vkGetDeviceProcAddr *



Application

Layer A

Layer B

Layer C

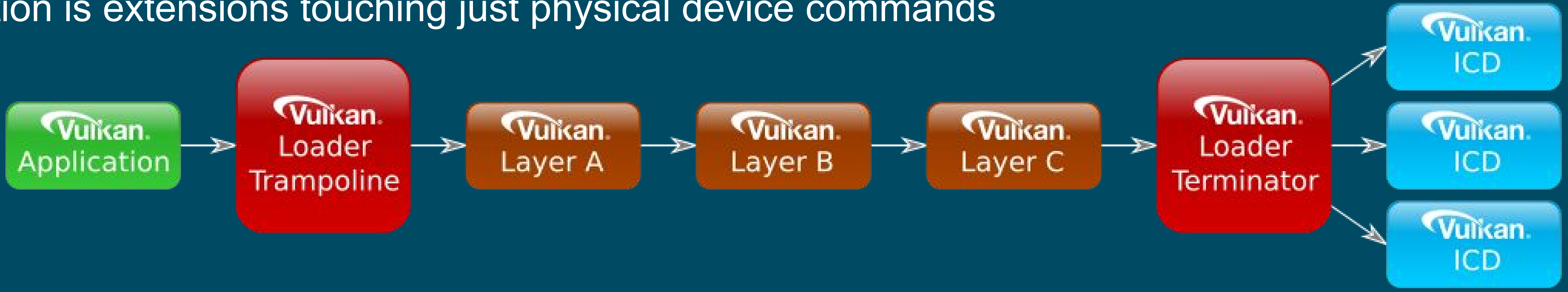
ICD

* Some special cases still require a specific device call chain to include a trampoline/terminator

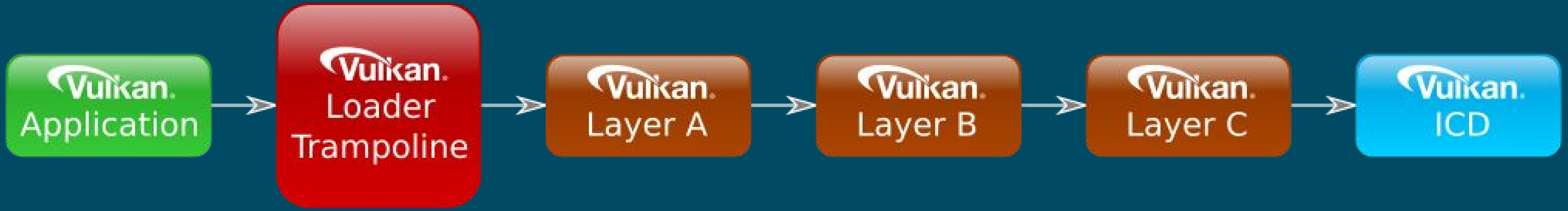
Vulkan Loader - Extensions

Instance Extensions **must be known by the loader!**

- Exception is extensions touching just physical device commands



The loader doesn't need to know about Device Extensions



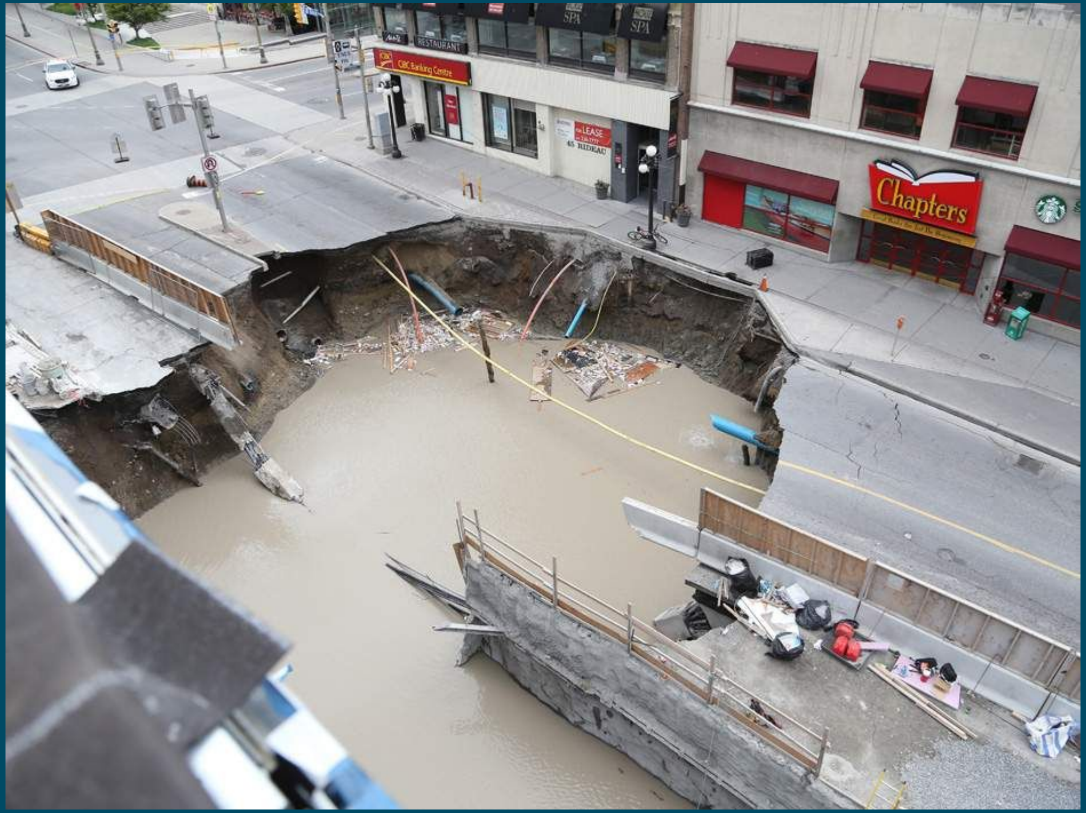
Vulkan Desktop Loader Debug Environment Variable

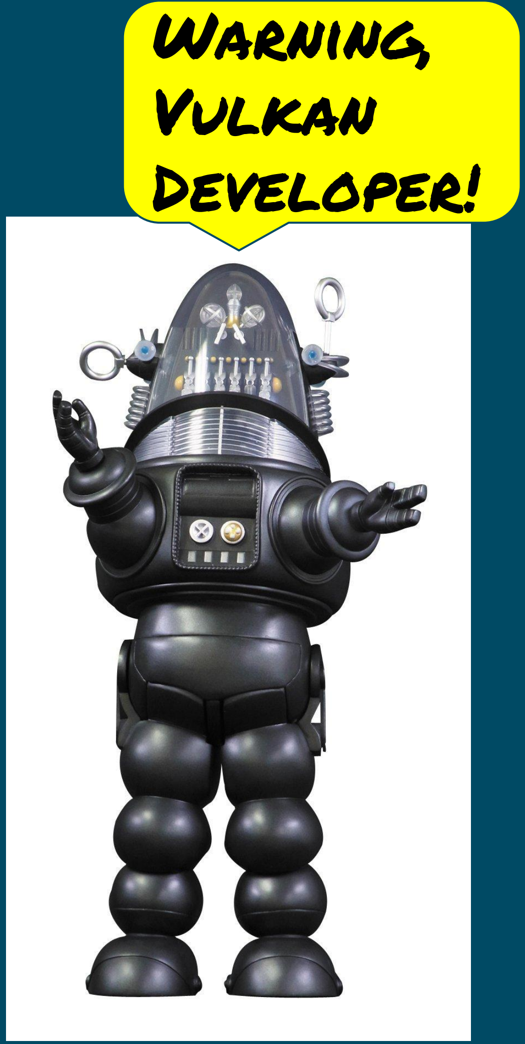
Enable Loader debug messages:

VK_LOADER_DEBUG warn, error, info, perf, debug, all

Vulkan Loader Warning

- Loader will crash if you use it improperly
- Designed for performance and functionality
- Like C, enough rope...





Vulkan ICDs [Desktop]

- Looks for Manifest files
  - Formatted in JSON
  - Contain basic information about ICD (name, API version, library)
  - Windows
    - Registry: HKLM/Software/Khronos/Vulkan/Drivers
  - Linux
    - Standard folders (under vulka/icd.d/):
      - /usr/local/etc/vulkan/icd.d
      - /usr/local/share/vulkan/icd.d
      - ...

- Loader investigates these during vkEnumerateInstanceExtensions and vkCreateInstance

Vulkan Loader ICD Debug Environment Variables

Force a particular Driver path:

VK_DRIVERS_PATH Delimited list of paths to location of driver JSON files

Force a particular ICD:

VK_ICD_FILENAMES Delimited list of specific driver JSON files (by full driver name)

Vulkan Layers

- Optional components, enabled by request
  - App passes layer names to vkCreateInstance via VkInstanceCreateInfo member ppEnabledLayerNames
  - Desktop environment var: VK_INSTANCE_LAYERS

- Can add, remove, or augment Vulkan behavior

○ Validation	VK_LAYER_LUNARG_standard_validation
○ Track debug data	VK_LAYER_RENDERDOC_Capture
○ Render FPS	VK_LAYER_LUNARG_monitor
○ Log content	VK_LAYER_LUNARG_api_dump
○ Grab screenshots	VK_LAYER_LUNARG_screenshot
○ Write your own!	

Reasons for Using Validation

Determine application correctness

Catch portability issues

- Produces validation errors, but still works for you
- May not work for others

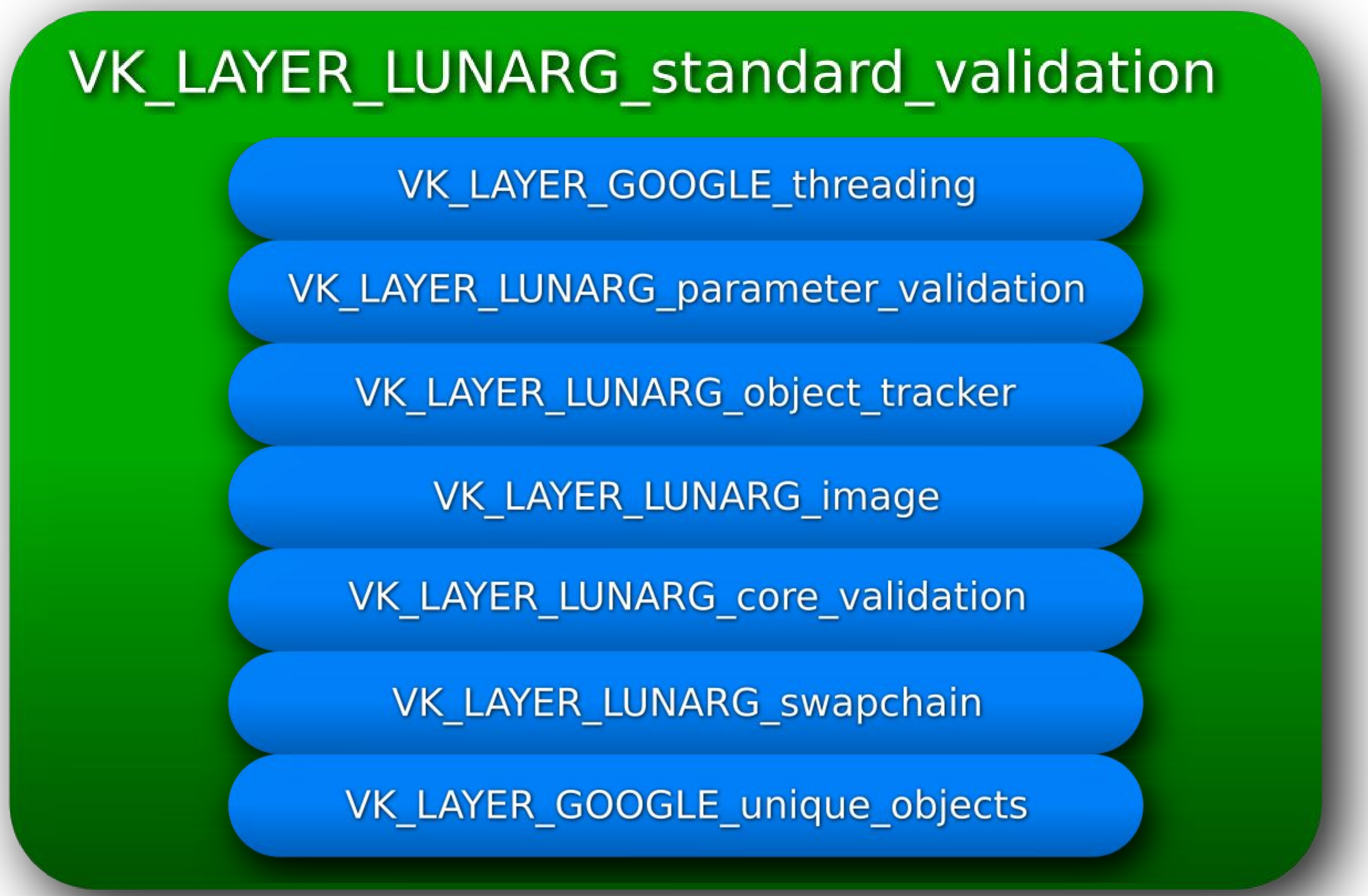
Evaluate Vulkan usage efficiency

- More focus on this in the future

You want to be like: VALVE, id, EPIC GAMES, oculus, OXIDE, unity

What’s in Standard Validation?

A “Meta-Layer” grouping other layers in proper order



VK_LAYER_LUNARG_standard_validation

VK_LAYER_GOOGLE_threading

VK_LAYER_LUNARG_parameter_validation

VK_LAYER_LUNARG_object_tracker

VK_LAYER_LUNARG_image

VK_LAYER_LUNARG_core_validation

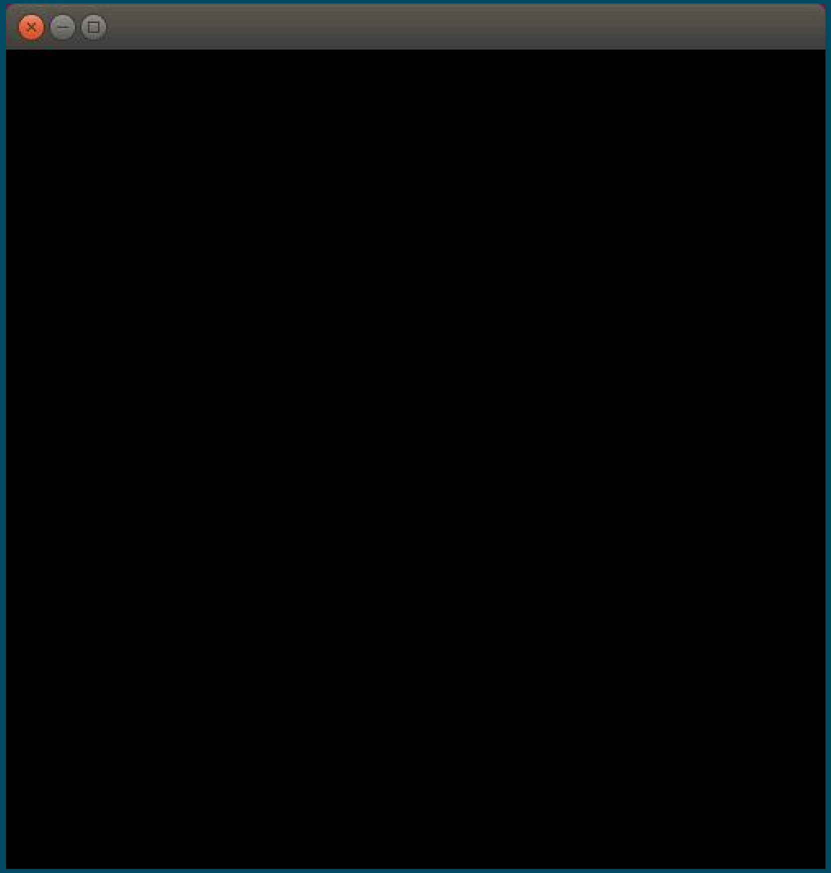
VK_LAYER_LUNARG_swapchain

VK_LAYER_GOOGLE_unique_objects

Only on Desktop, Sorry Android Developers

Using Standard validation

I wrote my app, it runs (without returning a bad VkResult) but...



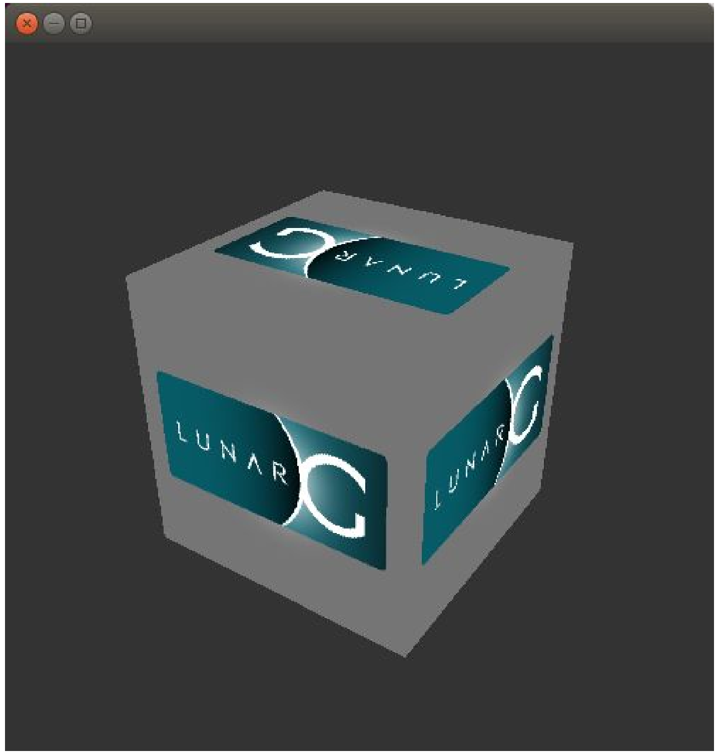
Initial response: “Man, Vulkan Sucks!”

Validation Output

Turn on validation and you see:

```
ERROR: [DS] Code 31 : You must call vkEndCommandBuffer() on
CB 0x97b8e0 before this call to vkQueueSubmit()!
```

Easy fix, and then:



Always Grab the Latest

Download the latest SDKs (for Desktop or Android)

Continually improving:

- Validation coverage
- Support for new Extensions
- Bug fixes
- Performance tweaks
- Warning/Error message clarifications
  - Listing Spec sections

Useful, But Don’t Always Enable

Validation layers causes perf impact

Performance hit depends on application complexity

Smoke (in LVL demos) on Intel Linux Mesa:

Normal:	160+ fps
With Validation:	6+ fps (roughly 4% of initial perf)

If higher perf needed

- Don’t use “standard_validation”
- Manually enable some validation layers

Needed during development, but not final product

Vulkan Layer Dispatching

- Must have own dispatch table (to call next in chain)
  - Don’t use table in object

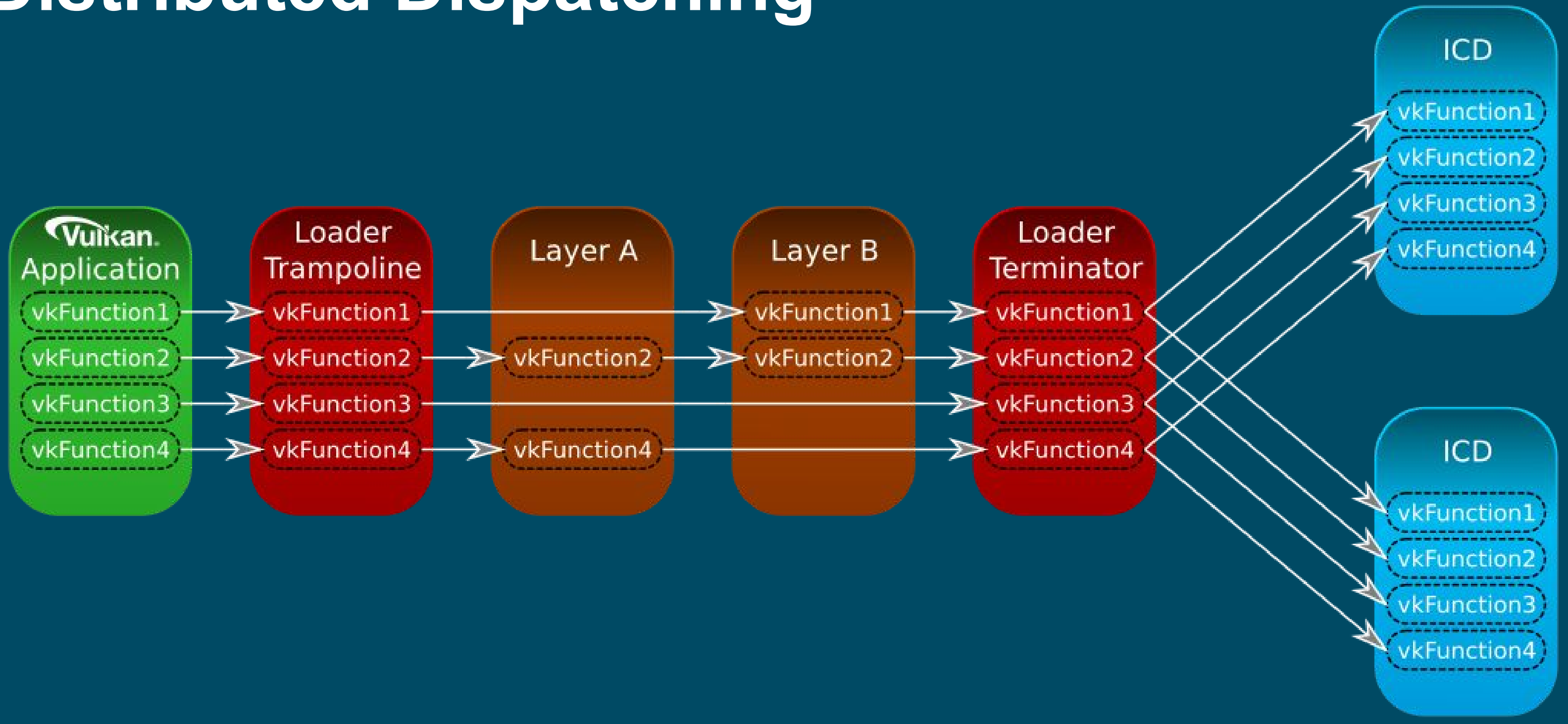
- Assistance available:
  - vk_layer.h defines instance and device dispatch table structures and utility funcs
  - Some extensions present, but layers may need to define their own extension function pointer storage

Vulkan Layer Distributed Dispatching

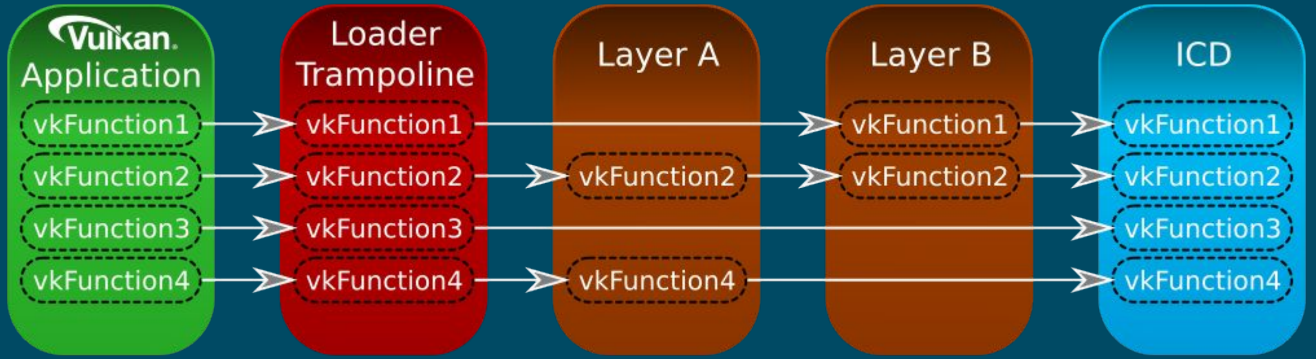
- Layers do NOT have to intercept all calls
  - Must intercept
    - vkCreateInstance
    - vkEnumerateInstanceLayerProperties
    - vkGetInstanceProcAddr
  - If implementing device commands, must also intercept
    - vkCreateDevice
    - vkGetDeviceProcAddr
- Layers should pass the info along, except
  - vkNegotiateLoaderLayerInterfaceVersion
  - Others may choose to not pass down

Vulkan Layer Distributed Dispatching

Instance Chain



Device Chain



LUNARG

Vulkan Layer Definitions (Desktop)

- Stored in JSON file
  - Windows : Define in Registry
    - HKLM/Software/Khronos
  - Linux: Found in Standard paths
    - /usr/local/etc/vulkan
    - /usr/local/share/vulkan
    - /etc/vulkan
    - /usr/share/vulkan
    - $HOME/.local/share/vulkan
- Queried by loader without loading library for security reasons

Desktop Layer Debug Environment Vars

- Force on a Layer from outside the application:

  VK_INSTANCE_LAYERS		Delimited list of layer names to enable

- Force/Override the Layer path:

  VK_LAYER_PATH			Delimited list of paths to search for layer JSON files

Desktop Layer Loading

Implicit

- Can be always enabled
- Disable with Environment Variable (Defined in JSON)
- Example: VK_LAYER_NV_Optimus

Explicit

- Must be enabled by app or environment

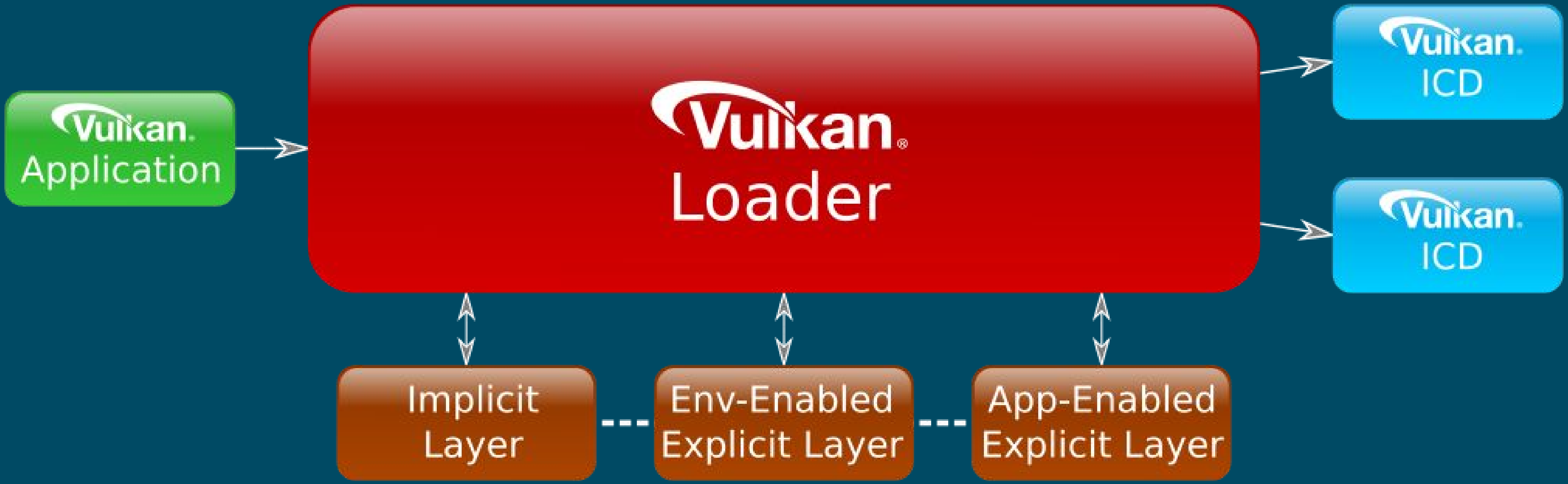
Different registry/folder locations:

- Windows Registry:
  - ImplicitLayers
  - ExplicitLayers

- Linux folders:
  - implicit_layer.d
  - explicit_layer.d

Only
“Explicit Layers” on
Android

Overall Desktop Layer Order



Vulkan Layer Wrapping

- "Wrapping"
  - Creating your own object that contains a dispatchable object
  - Return your object pointer back up call chain
  - When called, "unwraps" object on way back down call chain
- Possibilities
  - If you can avoid wrapping:
    - Use hash table (or something similar) to reference your data based on dispatchable object value
  - If you have to wrap:
    - **Must** "unwrap" your object in any extension command that uses that object for everything to work properly
    - **Suggest** you maintain a "whitelist" of supported extensions and warn on something new
    - Layer **must** wrap with struct containing dispatch table
      - Initialize with SetInstanceLoaderData or SetDeviceLoaderData

RenderDoc

Graphical Debugger with Vulkan support

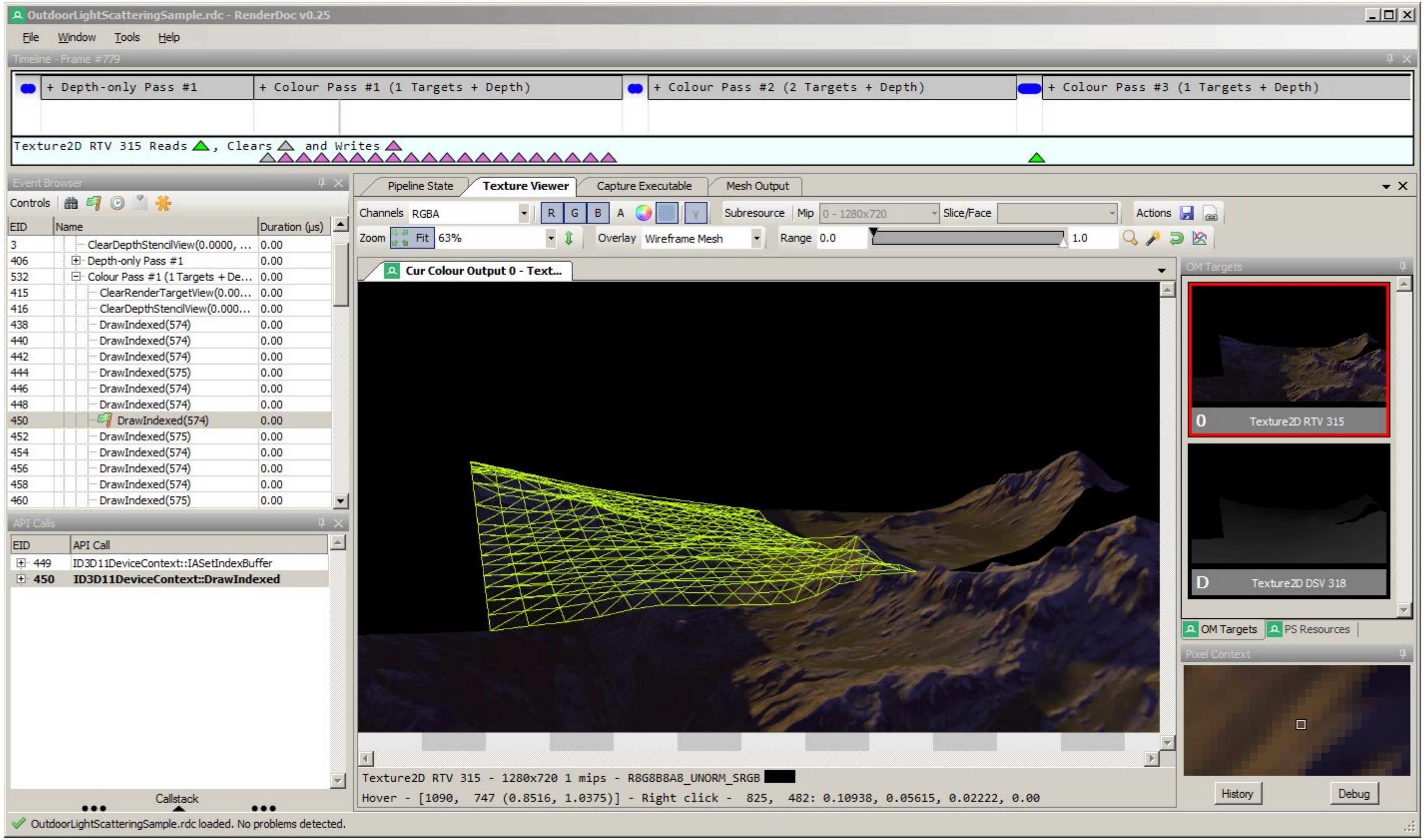
Currently only on Windows

Record and then investigate

Where?

- Installed with LunarG’s Vulkan SDK
- Source available in Github

RenderDoc



+ Depth-only Pass #1 | + Colour Pass #1 (1 Targets + Depth) | + Colour Pass #2 (2 Targets + Depth) | + Colour Pass #3 (1 Targets + Depth)

Texture2D RTV 315 Reads △, Clears △ and Writes △

EID	Name	Duration (µs)
3	ClearDepthStencilView(0.0000, ...	0.00
406	⊞ Depth-only Pass #1	0.00
532	⊟ Colour Pass #1 (1 Targets + De...	0.00
415	ClearRenderTargetView(0.00...	0.00
416	ClearDepthStencilView(0.000...	0.00
438	DrawIndexed(574)	0.00
440	DrawIndexed(574)	0.00
442	DrawIndexed(574)	0.00
444	DrawIndexed(575)	0.00
446	DrawIndexed(574)	0.00
448	DrawIndexed(574)	0.00
450	DrawIndexed(574)	0.00
452	DrawIndexed(575)	0.00
454	DrawIndexed(574)	0.00
456	DrawIndexed(574)	0.00
458	DrawIndexed(574)	0.00
460	DrawIndexed(575)	0.00

EID	API Call
⊞ 449	ID3D11DeviceContext::IASetIndexBuffer
⊞ **450**	**ID3D11DeviceContext::DrawIndexed**

Channels RGBA

Subresource Mip 0 - 1280x720 Slice/Face

Zoom Fit 63%

Overlay Wireframe Mesh

Range 0.0 1.0

0 Texture2D RTV 315

D Texture2D DSV 318

```
Texture2D RTV 315 - 1280x720 1 mips - R8G8B8A8_UNORM_SRGB
Hover - [1090,  747 (0.8516, 1.0375)] - Right click -  825,  482: 0.10938, 0.05615, 0.02222, 0.00
```

✓ OutdoorLightScatteringSample.rdc loaded. No problems detected.

Beyond RenderDoc

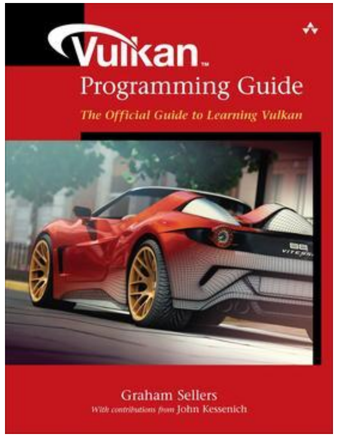
RenderDoc is a great place to start, but missing GPU internal data

- No kernel-level thread timing
- No GPU context submission information
- Missing throughput information

For that, use your HW vendor’s tools (Vulkan Support may vary):

- AMD PerfStudio
- Intel GPA
- Nvidia Nsight
- ARM Streamline Performance Analyzer
- Imagination PowerVR Tools
- Qualcomm Adreno Profiler
- ...

Other Resources


Vulkan
Programming Guide
The Official Guide to Learning Vulkan
Graham Sellers
With contributions from John Kessenich

Vulkan Book:

Vulkan Programming Guide is out!

Vulkan Tutorial:

LunarG SDK : https://vulkan.lunarg.com/doc/sdk/latest/windows/tutorial/html/index.html

Fancier Examples:

Sascha Willems : https://github.com/SaschaWillems/Vulkan

Many others available (listed in Khronos' Vulkan Resource Page)

Links

Khronos Vulkan Resources:
https://github.com/KhronosGroup/Khronosdotorg/blob/master/api/vulkan/resources.md

Vulkan SDKs:

- LunarG : https://vulkan.lunarg.com/
- Android : https://developer.android.com/ndk/guides/graphics/index.html

LoaderAndValidationLayers GitHub (Khronos): Loader, Validation Layers, Docs
https://github.com/KhronosGroup/Vulkan-LoaderAndValidationLayers

LoaderAndLayerIf Document: <GitHub>/blob/master/loader/LoaderAndLayerInterface.md

VulkanTools GitHub (LunarG): VIA, VkTrace, ApiDump, Screenshot layer
https://github.com/LunarG/VulkanTools

MoltenVK : https://moltengl.com/moltenvk/

RenderDoc : https://github.com/baldurk/renderdoc