Vulkan®

DEVELOPER DAY

VLF/Assistant Layer

Mark Lobodzinski, LunarG

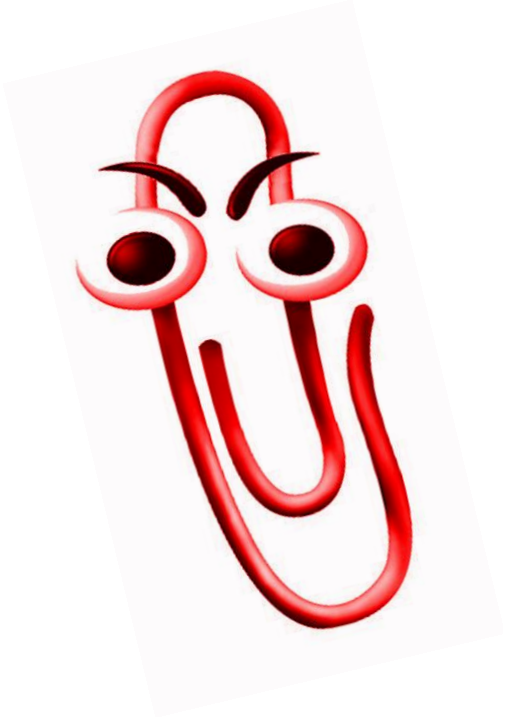
KHRONOS™ GROUP

UBISOFT

MONTRÉAL
APRIL 2018

We'll spend 20-30 minutes on:

- Quick intro to the Vulkan Assistant Layer
- Overview of the Vulkan Layer Factory
- Demonstrations



KHRONOS GROUP

Assistant Layer

The `VK_LAYER_LUNARG_assistant_layer` is a Vulkan 'best-practices' layer

This layer will help application developers to find:

The Assistant Layer should be run periodically along with normal validation checks so that issues may be addressed in early stages of development

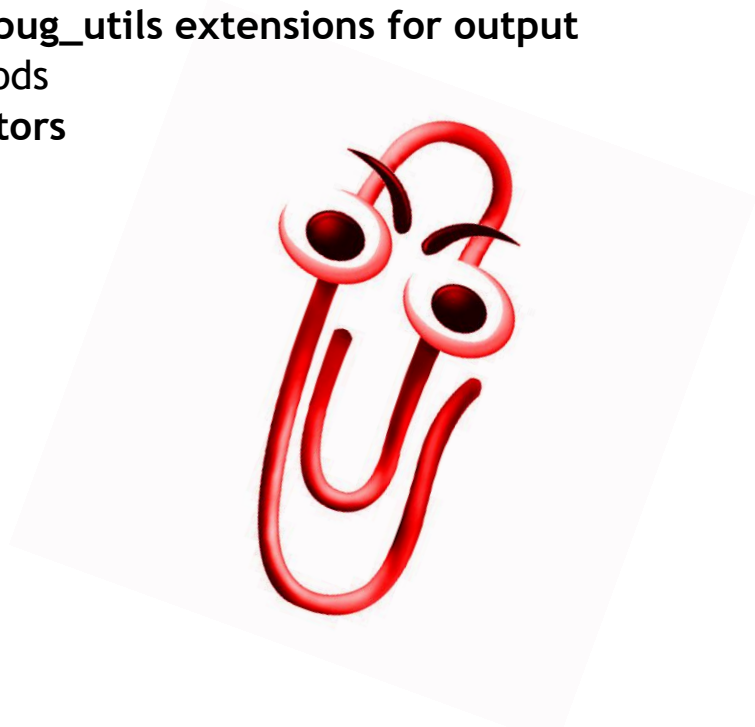
Assistant Layer Coverage

- Checks tracked in Khronos Github V-LVL Issue #1612
- Not shown here are portability checks suggested in Ecosystem Issue #11
- Warnings not *explicitly* declared in Vulkan spec will be migrated to the Assistant Layer
- Please suggest or add new checks in this issue!

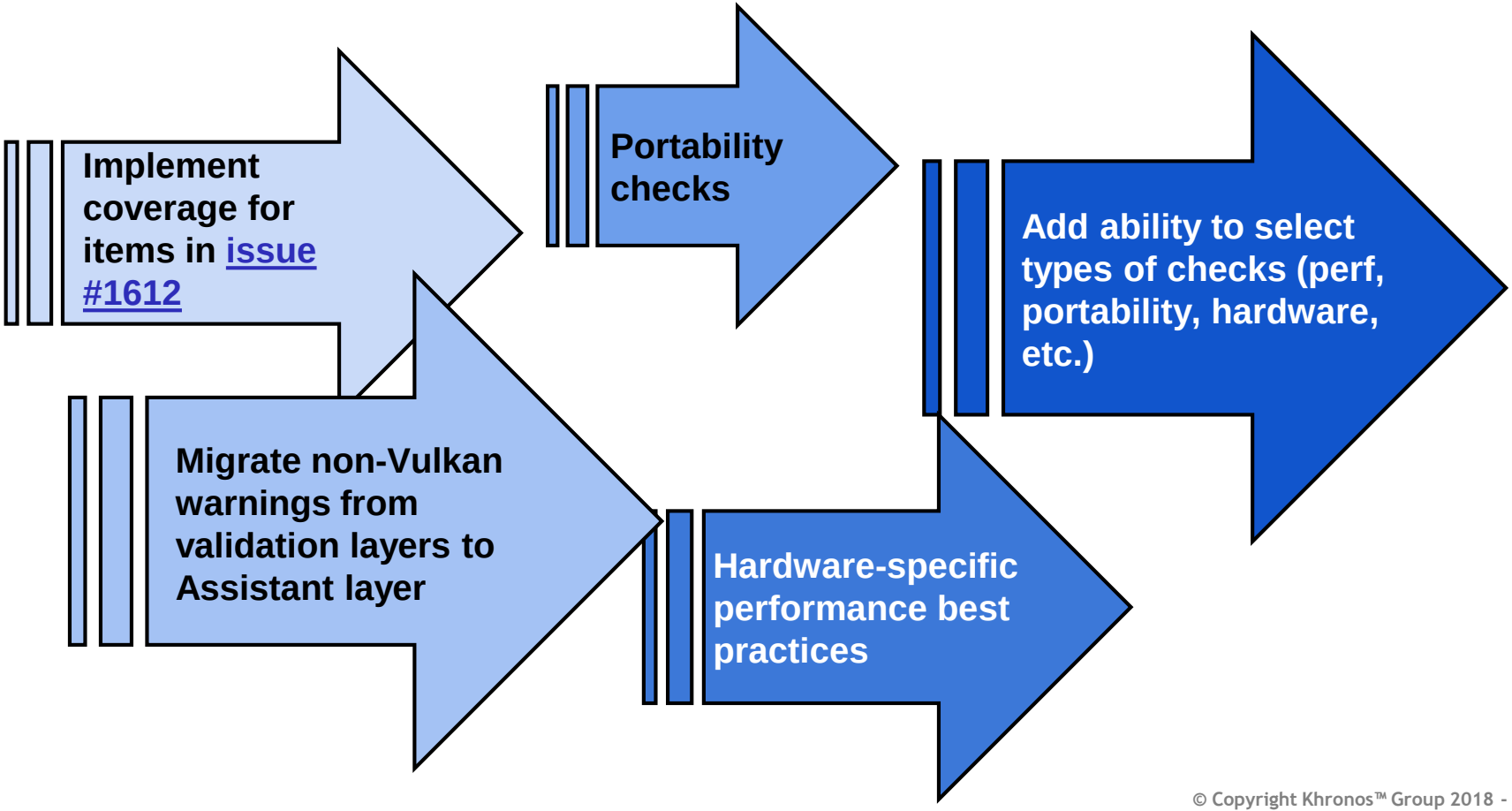
- [ ] #2535 Validate getter's structure count
- [ ] #2486 Potentially false validation error if image size > maxResourceSize
- [ ] Add warnings for non-success driver return values
- [ ] Warn for skipping Get calls -- GetBufferMemoryRequirements, GetImageMemoryRequirements, etc.
- [x] Move validation WARNINGs not explicitly defined by the specification to the assistant layer
- [x] #2248 validation: Add a warning for LOAD_OP_LOAD + LAYOUT_UNDEFINED (note that this check will be present in the core_validation layer).
- [ ] #2137 Add warning on mismatch between `sharingMode` and `queueFamilyIndexCount`
- [ ] #2089 No error when recreating swapchain while rendering to an existing swapchain image
- [x] #2034 Need validation layer to add redundancy check for vulkan states.
- [x] #1912 Add warning when a parameter of vkCmdDispatch is 0
- [x] #1807 Give error when requesting device extensions on instance creation and vice versa
- [ ] #1696 Performance hint: Warn if application does not use pipeline cache
- [x] #1628 VK_LAYER_LUNARG_core_validation queue submissions list grows forever and causes crash
- [ ] #1401 Performance warning: VK_PIPELINE_STAGE_ALL_COMMANDS_BIT or VK_PIPELINE_STAGE_ALL_GRAPHICS_BIT
- [x] #1253 Performance warning: Pipeline barriers used for layout transitions instead of Render Passes
- [ ] #1155 Performance warning: High number of memory objects
- [x] #1407 Performance warning: Detect over-synchronization
- [ ] #727 Performance warning: when instanceCount = 0 (or drawCount = 0)

Assistant Layer

- **Built with Vulkan Layer Factory**
  - Checks implemented as individual interceptors
  - Trivial to extend with new checks (pull-requests are welcome!)
- **Uses VK_EXT_debug_report and VK_EXT_debug_utils extensions for output**
  - In addition to several other output methods
- **Easy to customize with proprietary interceptors**
  - Enforce your own best practices



Assistant Layer Roadmap

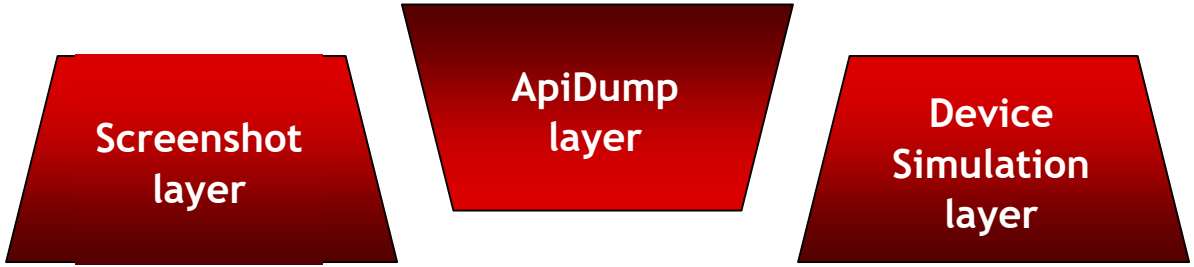


Assistant Layer

You can find the LunarG VulkanTools Repository at

https://github.com/LunarG/VulkanTools

There are other useful tools as well --



All this stuff is *ALREADY* included in the Windows and Linux SDKs

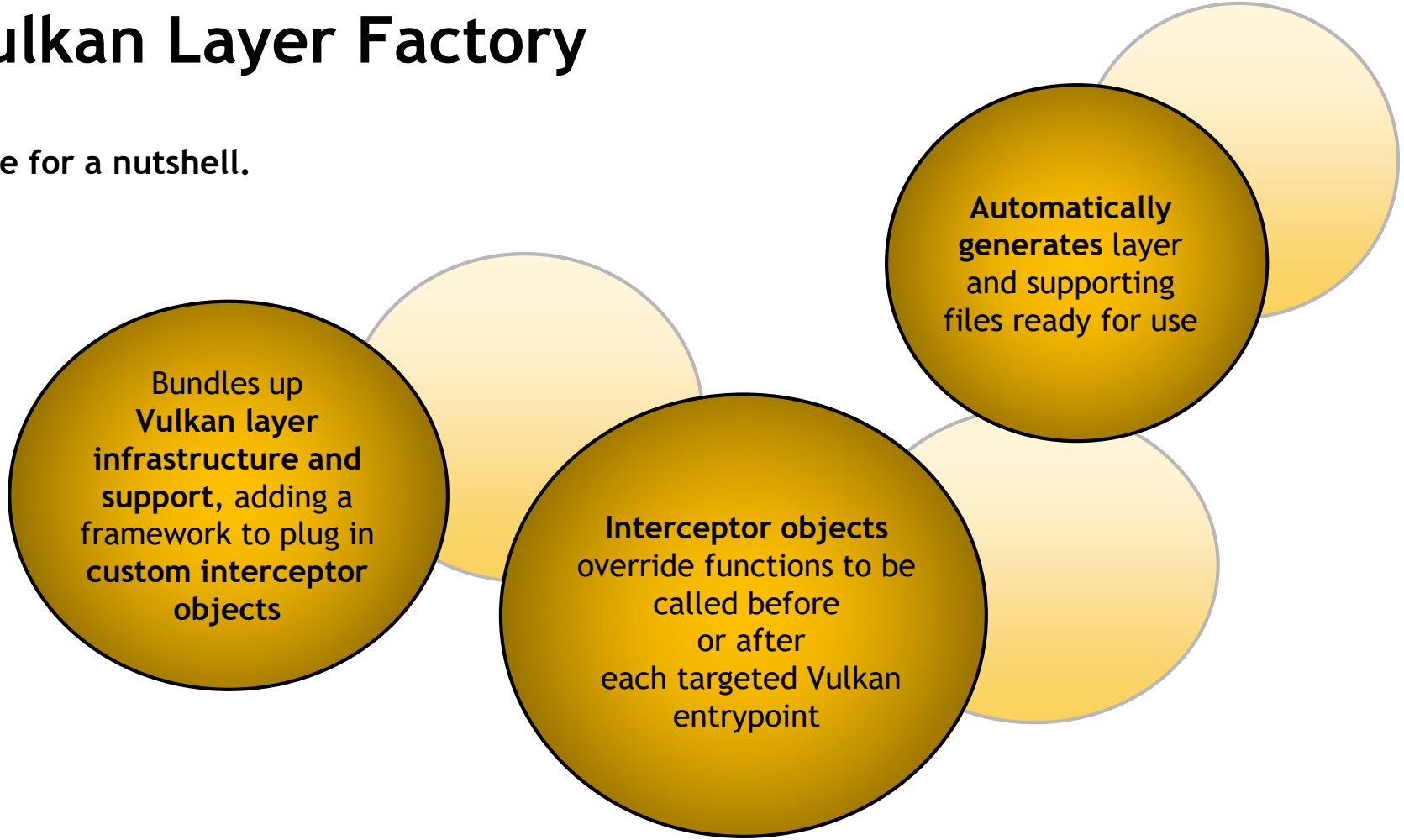
This layer has been created with the *Vulkan Layer Factory*, and third-party contributions are welcome!

Vulkan Layer Factory

- Is a framework based on canonical Vulkan layer model that helps in creating Vulkan Layers
- Hides the majority of the loader-layer interface, layer boilerplate, setup and initialization, and complexities of layer development
- Helps to fulfill original promise of the Vulkan layer concept, making layer creation straightforward and practical enough even for one-off debugging or investigation purposes
- It really does

Vulkan Layer Factory

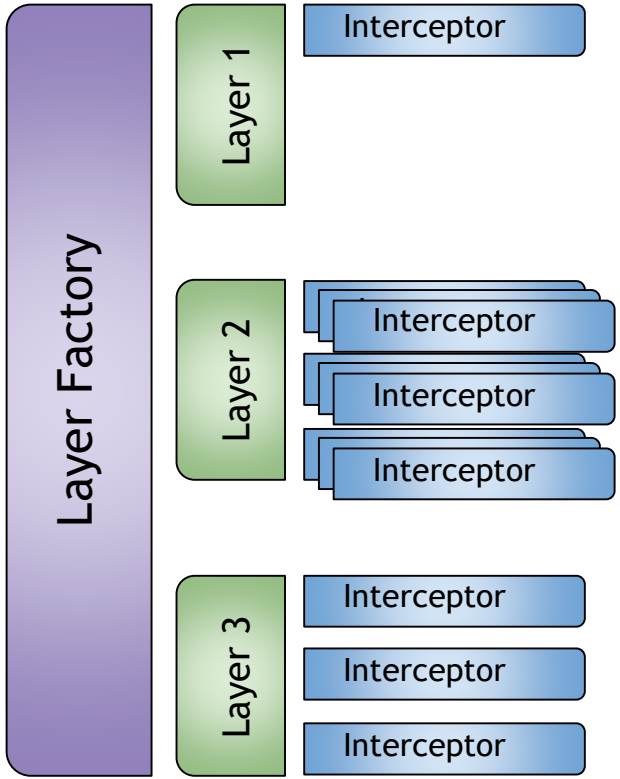
Time for a nutshell.



Vulkan Layer Factory

Framework Structure:

- One factory instance can host many layers and interceptors
- Each Factory Layer is comprised of one or more *interceptor* objects
- Interceptor objects override functions to be called before (PreCallApiName) or after (PostCallApiName) each Vulkan entrypoint of interest
- Each interceptor is independent of all others within a Factory Layer, and their call order is not guaranteed



Vulkan Layer Factory

Layers have historically been underutilized due to complexity and a high barrier to entry for implementation, due in-part to:

- CMake/Build configurations/Multi-platform support
- Dispatch tables & GetProcAddress Handlers
- Loader/Layer interface functions (does anybody even know what these are?)
- Windows module definition files
- Layer .json configuration files
- Layer Properties functions (you probably hadn't even heard of these)
- Extension handling
- Debug Report/Debug Utils extensions for output
- Linux display server support
- Layer implementation environment

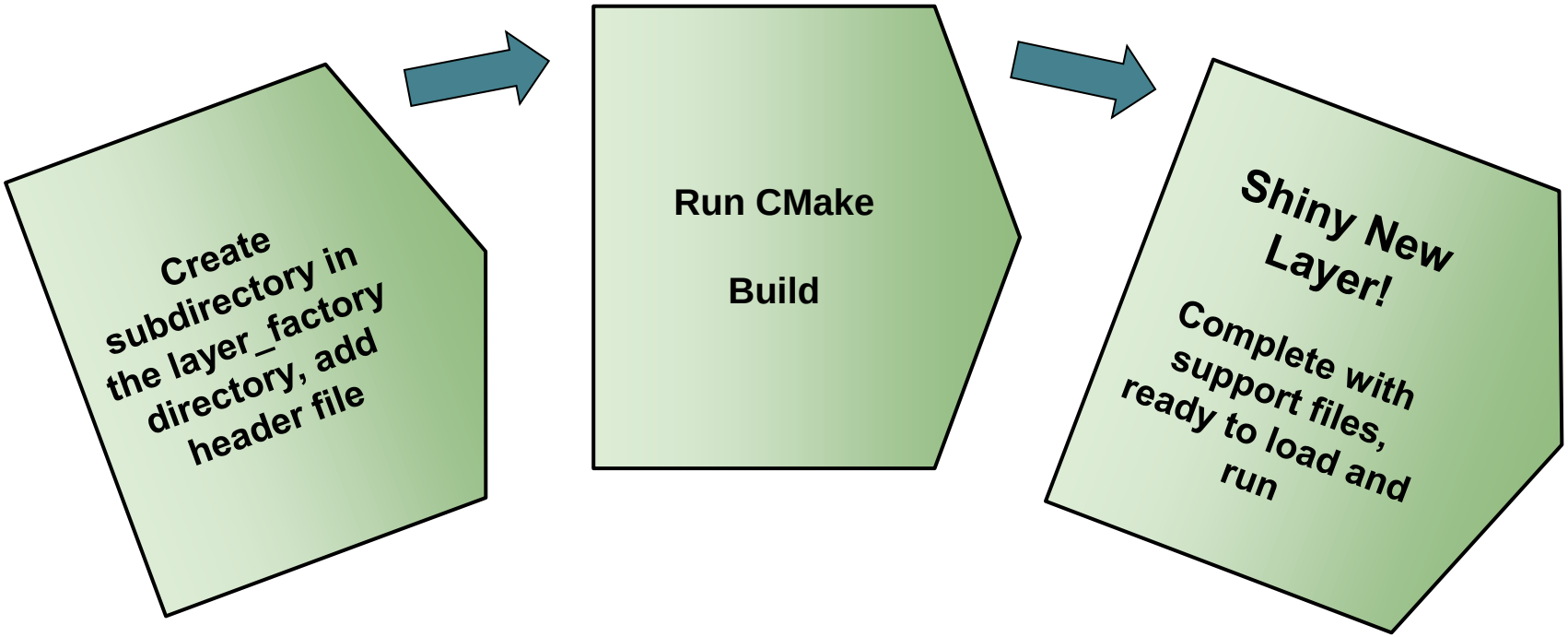
VLF Features!

Not only is the grunt-work greatly reduced, but helpful features were added

- **Pre- and Post-API-call intercept points**
  - Before- and after- down-chain call intercept points for each API
- **More Output Options**
  - Standard layer-provided **`log_msg()`** calls
  - Simplified output helpers for simpler access to VK_EXT_debug_report output, i.e., **`Error()`**, **`Information()`**, etc.
  - **`printf()`** for standard-out or **`OutputDebugString()`** for Windows
- **Debug Helpers**
  - **BreakPoint()** helper generates break in Windows or Linux debugger
- **Global Intercept Helpers**
  - Override **`PreCallApiFunction()`** and/or **`PostCallApiFunction()`** and they will be called for EVERY API call

We'll now take a look at what it takes to create a layer

Make a Vulkan Layer using the Factory

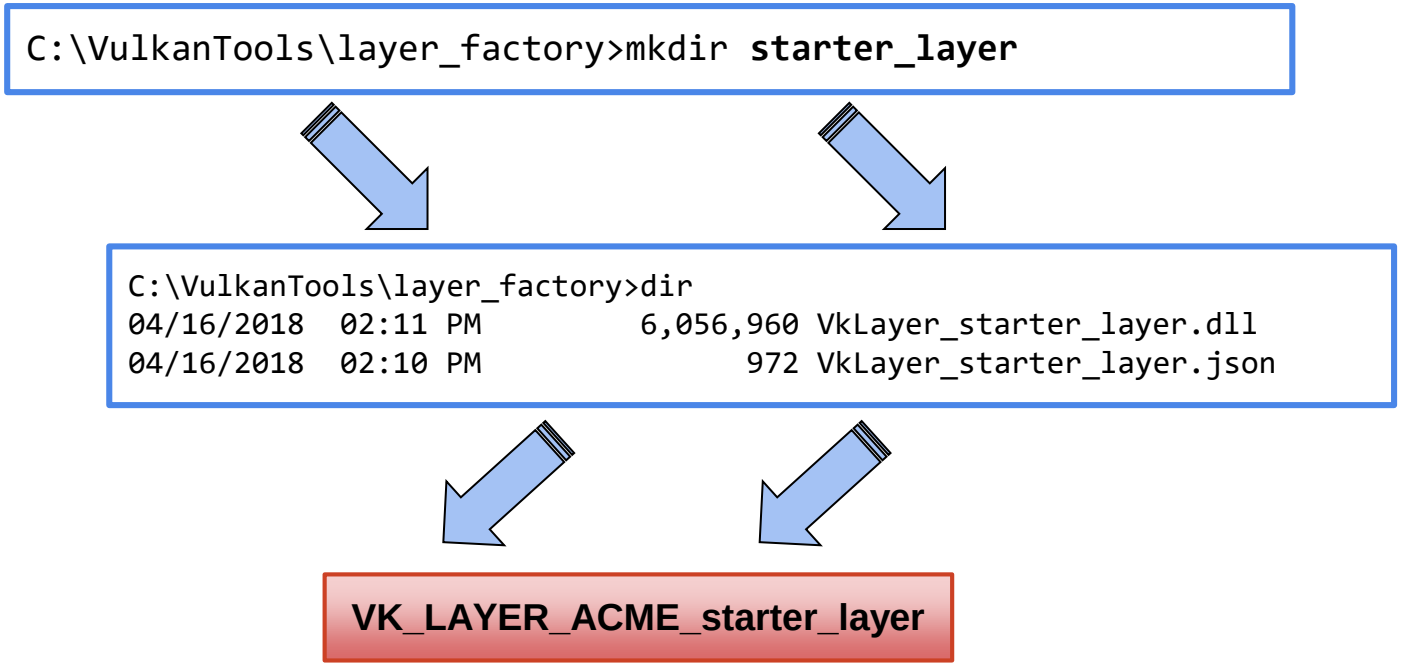


These steps, with a bit more detail, follow

Create a Factory Layer -- Step One

Create a subdirectory in the layer_factory directory

- **Select a name that will serve as the resultant layer**



Create a Factory Layer -- Step Two

Add your state tracker file(s) to the subdirectory

This can be a single header file, or multiple header and source files

memory_allocation_stats.h

```
#pragma once
#include <sstream>
#include <unordered_map>
static uint32_t display_rate = 60;

class MemAllocLevel : public layer_factory {
  public:
    // Constructor for interceptor
    MemAllocLevel() : layer_factory(this), number_mem_objects_(0), total_memory_(0), present_count_(0){};
    // Intercept the memory allocation calls and increment the counter
    VkResult PostCallAllocateMemory(VkDevice device, […more params…], VkDeviceMemory *pMemory) {
       return VK_SUCCESS;
    }
    // Intercept the free memory calls and update totals
    void PreCallFreeMemory(VkDevice device, […more params…], const VkAllocationCallbacks *pAllocator) {
    }
    VkResult PreCallQueuePresentKHR(VkQueue queue, const VkPresentInfoKHR *pPresentInfo) {
         return VK_SUCCESS;
    }
  private:
    uint32_t number_mem_objects_;
    VkDeviceSize total_memory_;
    uint32_t present_count_;
    std::unordered_map<VkDeviceMemory, VkDeviceSize> mem_size_map_;
};

MemAllocLevel memory_allocation_stats;
```

Create a Factory Layer -- Step Three

Create (or copy) an '`interceptor_objects.h`' header file into your new directory

- This should include the header file for *each* of the included interceptors

```
C:\VulkanTools\layer_factory>type interceptor_objects.h
#include "memory_allocation_stats.h"
```

- Here's `interceptor_objects.h` for the Assistant Layer:

```
#include "mem_objects.h"
#include "zero_counts.h"
#include "stage_all_warn.h"
#include "load_op_load_undefined.h"
#include "pipeline_cache_warning.h"
#include "extension_type_warning.h"
#include "exclusive_qfi.h"
```

Create a Factory Layer -- ~~Step Four~~ Last Step!

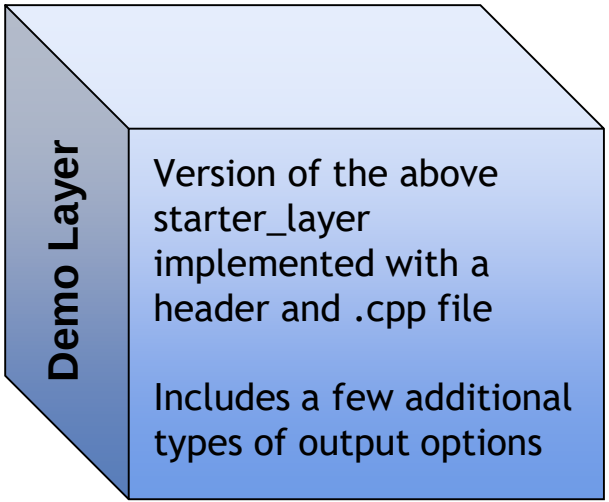
Run CMake and build

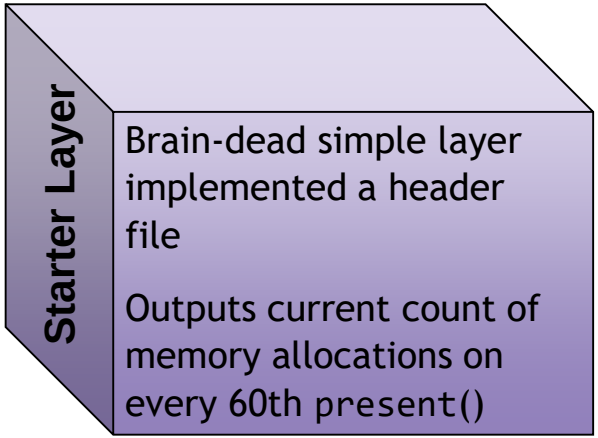
- CMake discovers Factory Layer subdirectories and interceptors each time it is run*
- A .json layer config file and .def file (for Windows) will also be created for your layer

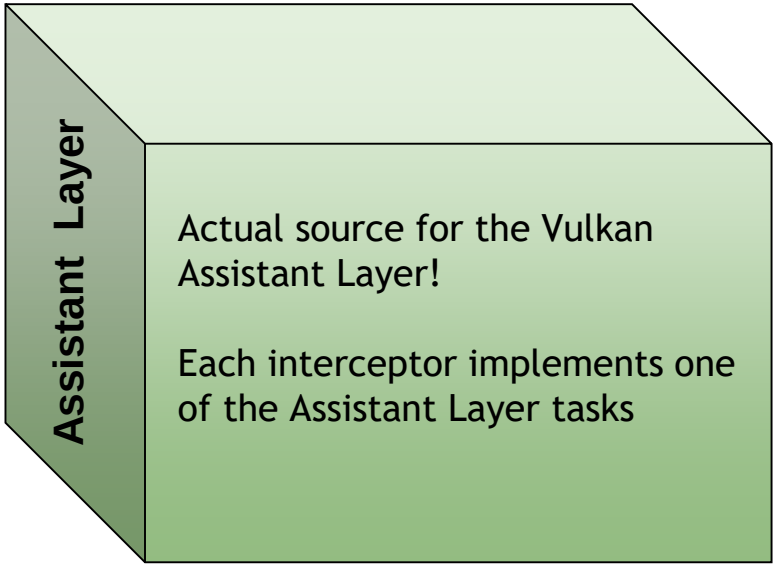
Layer.

*Adding or removing a layer_factory subdirectory/interceptor requires re-running CMake!

Layer Factory Samples







VLF Future Direction

- Move to stand-alone repository
- Structure to more easily incorporate VLF output into existing projects
- Use `add_interceptor` script to create API intercept stubs and source files
- Add Vulkan object templates for accessing per-object data
- Increase flexibility in handling of return values/modified parameters
- We are hungry for suggestions -- please share!

DEMONSTRATIONS

Resources

Github VulkanTools Repo

- https://github.com/LunarG/VulkanTools
- Ask questions, create issues or enhancement requests
- Submit Pull Requests!

Email:

Mark Lobodzinski mark@lunarg.com

Mike Schuchardt mikes@lunarg.com