GFXReconstruct -

Tools to Capture and Replay Graphics API Calls

Brad Grantham
LunarG, Inc.

Presented at the Khronos Vulkanised 2023 Conference

LUNARG

GFXReconstruct - Agenda

- Overview
- Use Cases
- Capture, replay, and other tools
- Architecture - file format, repository structure, classes
- Live demo

LUNARG

GFXReconstruct - Overview

Source Available : https://github.com/LunarG/GFXReconstruct

- Captures commands to a file (aka “a capture”)
- Replays captures
- Handful of additional tools
- C++ libraries, layers, and apps; some Python wrappers
- Linux, Android, Windows
- API-agnostic; Vulkan and Direct3D 12 available in the Github repository

GFXReconstruct - Use Cases

Save an app's Vulkan commands and replay them consistently

- Driver regression testing
- Architecture simulation
- Silicon bringup
- Bug reporting

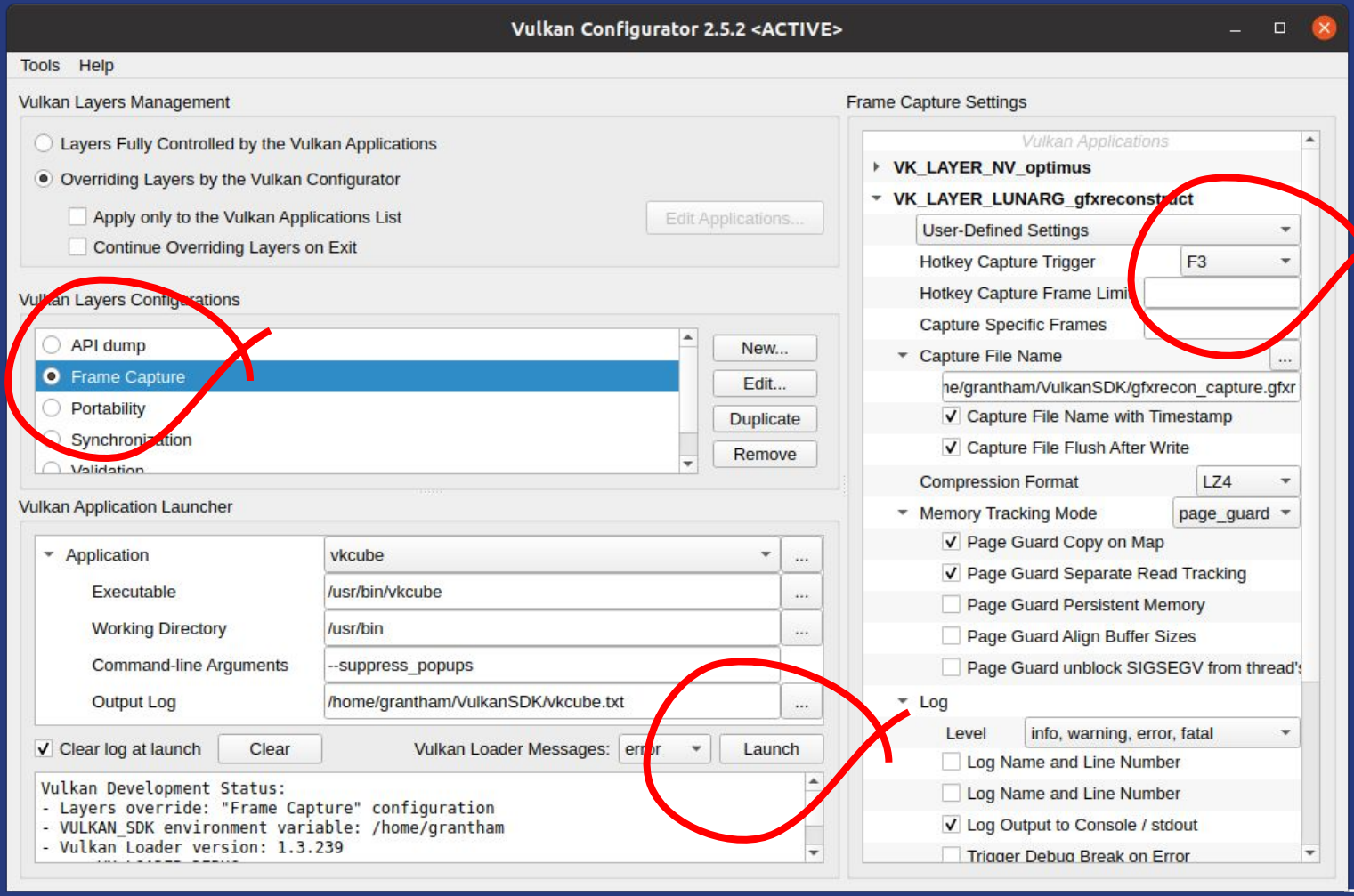
Currently in use by several GPU, chipset, platform vendors

LUNARG

GFXReconstruct - Capturing An Application

Vulkan API layer “`libVkLayer_gfxreconstruct.so`” or `.dll`

- Use Vulkan Configurator (VkConfig)
  - (or “`gfxrecon.py capture-vulkan`”)
  - (or `VK_INSTANCE_LAYERS=VK_LAYER_LUNARG_gfxreconstruct`)
- Hooks all core 1.3 Vulkan function calls and many extensions
- Captures most function inputs and outputs
  - (almost - e.g. outputs not stored on error return)
- Writes a blocks per call to capture file
  - Compressed by default


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
Edit Applications...
Continue Overriding Layers on Exit
Vulkan Layers Configurations
API dump
Frame Capture
Portability
Synchronization
Validation
New...
Edit...
Duplicate
Remove
Vulkan Application Launcher
Application
vkcube
Executable
/usr/bin/vkcube
Working Directory
/usr/bin
Command-line Arguments
--suppress_popups
Output Log
/home/grantham/VulkanSDK/vkcube.txt
Clear log at launch
Clear
Vulkan Loader Messages:
error
Launch
Vulkan Development Status:
- Layers override: "Frame Capture" configuration
- VULKAN_SDK environment variable: /home/grantham
- Vulkan Loader version: 1.3.239
Frame Capture Settings
Vulkan Applications
VK_LAYER_NV_optimus
VK_LAYER_LUNARG_gfxreconstruct
User-Defined Settings
Hotkey Capture Trigger
F3
Hotkey Capture Frame Limi
Capture Specific Frames
Capture File Name
...
ne/grantham/VulkanSDK/gfxrecon_capture.gfxr
Capture File Name with Timestamp
Capture File Flush After Write
Compression Format
LZ4
Memory Tracking Mode
page_guard
Page Guard Copy on Map
Page Guard Separate Read Tracking
Page Guard Persistent Memory
Page Guard Align Buffer Sizes
Page Guard unblock SIGSEGV from thread's
Log
Level
info, warning, error, fatal
Log Name and Line Number
Log Name and Line Number
Log Output to Console / stdout
Trigger Debug Break on Error

```
Selected GPU 0: NVIDIA GeForce RTX 2070 with Max-Q Design, type: DiscreteGpu

[gfxrecon] INFO - Found layer settings file: /home/grantham/.local/share/vulkan/settings.d/
vk_layer_settings.txt
[gfxrecon] INFO - Successfully loaded settings from file
[gfxrecon] INFO - Initializing GFXReconstruct capture layer
[gfxrecon] INFO -   GFXReconstruct Version 0.9.16 (focal)
[gfxrecon] INFO -   Vulkan Header Version 1.3.239
[gfxrecon] INFO - Recording graphics API capture to /home/grantham/VulkanSDK/
gfxrecon_capture_trim_trigger_20230208T005340.gfxr
[gfxrecon] INFO - Finished recording graphics API capture

Process terminated
```


LUNARG

GFXReconstruct - Capturing An Application

(continued)

- Memory tracking mode - `GFXRECON_MEMORY_TRACKING_MODE` or `--memory-tracking-mode`

`pageguard` (default)	Attempt to detect writes to mapped buffers, write only changed pages
`assisted`	If application uses `vkFlushMappedMemoryRanges`
`unassisted`	Can just write all buffers all the time before Submit

LUNARG

GFXReconstruct - Capturing An Application

Trimming

- Can capture a range of frames or (on desktop) using a hotkey
- Can capture multiple ranges
  - E.g. `GFXRECON_CAPTURE_FRAMES=1,2,10-20`
- All graphics state up to the range is tracked
  - Stored in the capture file as state setup
- But conservative
  - Writes capture as it goes
  - Can’t know what future frames will reference

LUNARG

GFXReconstruct - `gfxrecon.py info`

Display useful information about a capture

- Compression
- Frames
- App info
- Device info

```
$ gfxrecon.py info ~/gfxrecon_capture_20220412T075011.gfxr
File info:
        Compression format: LZ4
        Total frames: 50

Application info:
        Application name: vkcube
        Application version: 0
        Engine name: vkcube
        Engine version: 0
        Target API version: 4198400 (1.1.0)

Physical device info:
        Device name: AMD Radeon RX 6700 XT
        Device ID: 0x73df
        Vendor ID: 0x1002
        Driver version: 8388821 (0x8000d5)
        API version: 4206795 (1.3.203)

Device memory allocation info:
[...]
```


LUNARG

GFXReconstruct - `gfxrecon.py replay`

Plays Vulkan function call stream as close to the original as possible

The *primary* purpose of GFXReconstruct is verbatim playback for regression and application inspection.

- Select one of multiple GPUs: “`--gpu`”
- Choose one of multiple captured surfaces to replay: “--surface-index”
  - Helpful on Android
- Save presented images: “--screenshots”, “--screenshot-all”

GFXReconstruct - `gfxrecon.py replay`

But it *is* possible to replay on other drivers, GPUs, vendors, to varying degrees.

- Memory alignment, types not guaranteed
  - Can translate using “`-m`”: “`remap`”, “`realign`”, “`rebind`”
  - Chooses subclasses of `VulkanResourceAllocator`

- Can mask off extensions: “`--remove-unsupported`”

- Will warn if function has different return codes in replay
  - Can also skip `Allocate` calls that failed in capture with “`--sfa`”

GFXReconstruct - `gfxrecon.py replay`

(continued)

- Swapchain acquisition order not guaranteed even on same GPU
  - Default option “Virtual Swapchain” renders to texture, then blits to swapchain image by replay index (can be disabled)

- Can attempt replay even on different platform with “--wsi”

GFXReconstruct - `gfxrecon.py optimize`

Removes unused resources, precomputes some values

Trimmed captures contain all objects created before the start of the trim range.

For Vulkan captures, `optimize`:

- Scans for unused resources
- Creates a new capture without unused resources
- Improves replay performance
- Reduces file size

GFXReconstruct - gfxrecon.py optimize

```
$ gfxrecon.py optimize trim.gfxr trim.opt.gfxr
Scanning F:/SaschaWillems-Vulkan-Samples/bin/trim.gfxr for unreferenced
resources.
[...]
Resource filtering complete.
        Original file size: 9588217 bytes
        Optimized file size: 6873678 bytes


$ gfxrecon.py replay trim.gfxr
[...] Replay FPS: 1514.922818 fps [...]


$ gfxrecon.py replay trim.opt.gfxr
[...] Replay FPS: 1794.189268 fps [...]
```

28% reduction

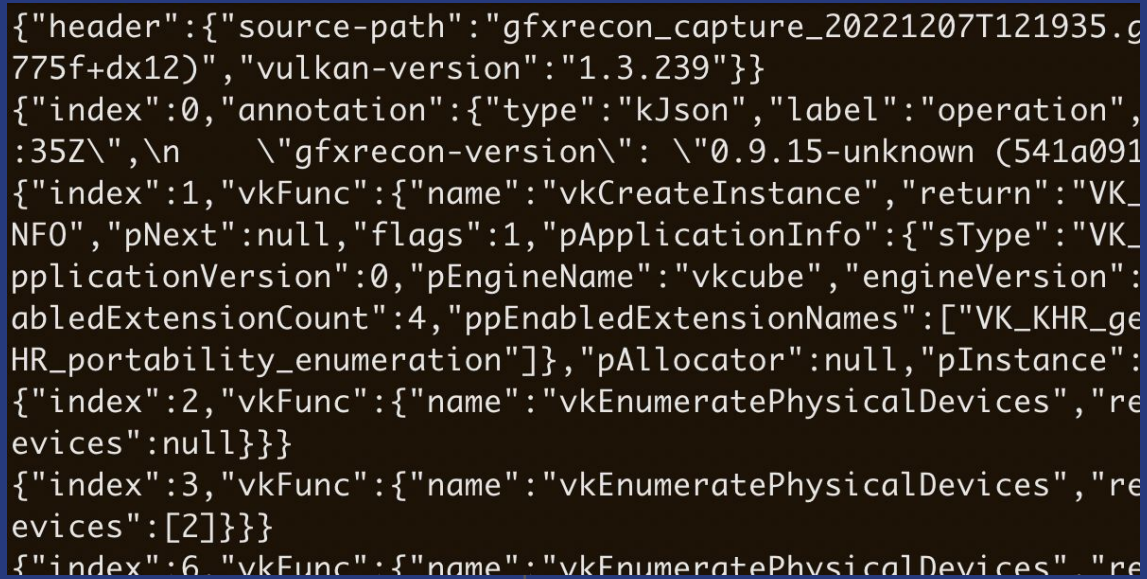
18% improvement

LUNARG

GFXReconstruct - `gfxrecon.py convert`

Output JSONLines representing a capture file

Compact, parseable, but not really human readable…



LUNARG

GFXReconstruct - `gfxrecon.py convert`

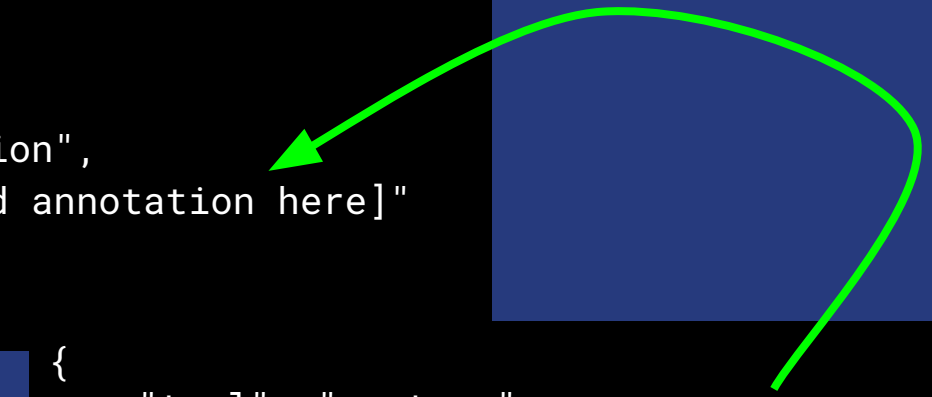
To make more readable, pipe through `jq`

Results are one object output per input JSONLines line, e.g.:

```
{
  "header": {
    "source-path": "gfxrecon_capture_20221207T121935.gfxr",
    "json-version": "0.8.0",
    "gfxrecon-version": "0.9.18-dev (dev:b28775f+dx12)",
    "vulkan-version": "1.3.239"
  }
}
```

GFXReconstruct - `gfxrecon.py convert`

```
{
  "index": 0,
  "annotation": {
    "type": "kJson",
    "label": "operation",
    "data": "[encoded annotation here]"
  }
}
```

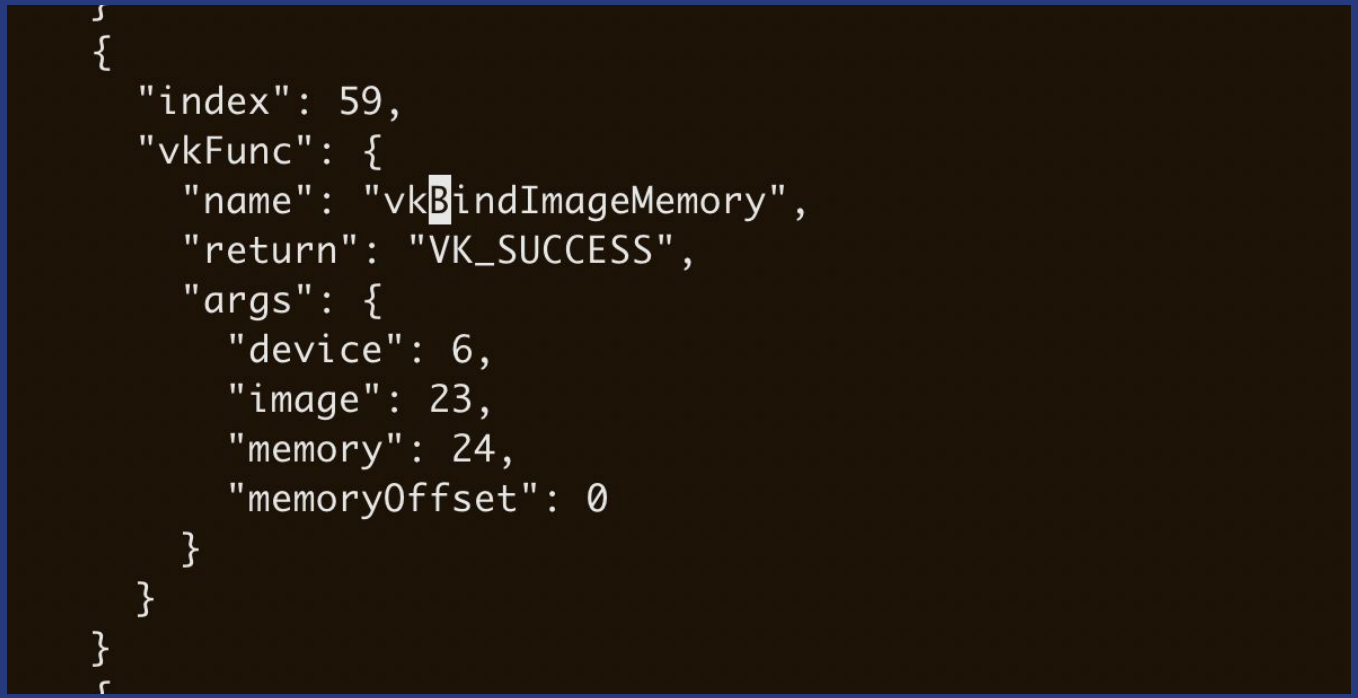


```
{
    "tool": "capture",
    "timestamp": "2022-11-07T20:19:35Z",
    "gfxrecon-version": "0.9.15-unknown (541a091*)",
    "vulkan-version": "1.3.231"
}
```

GFXReconstruct - `gfxrecon.py convert`

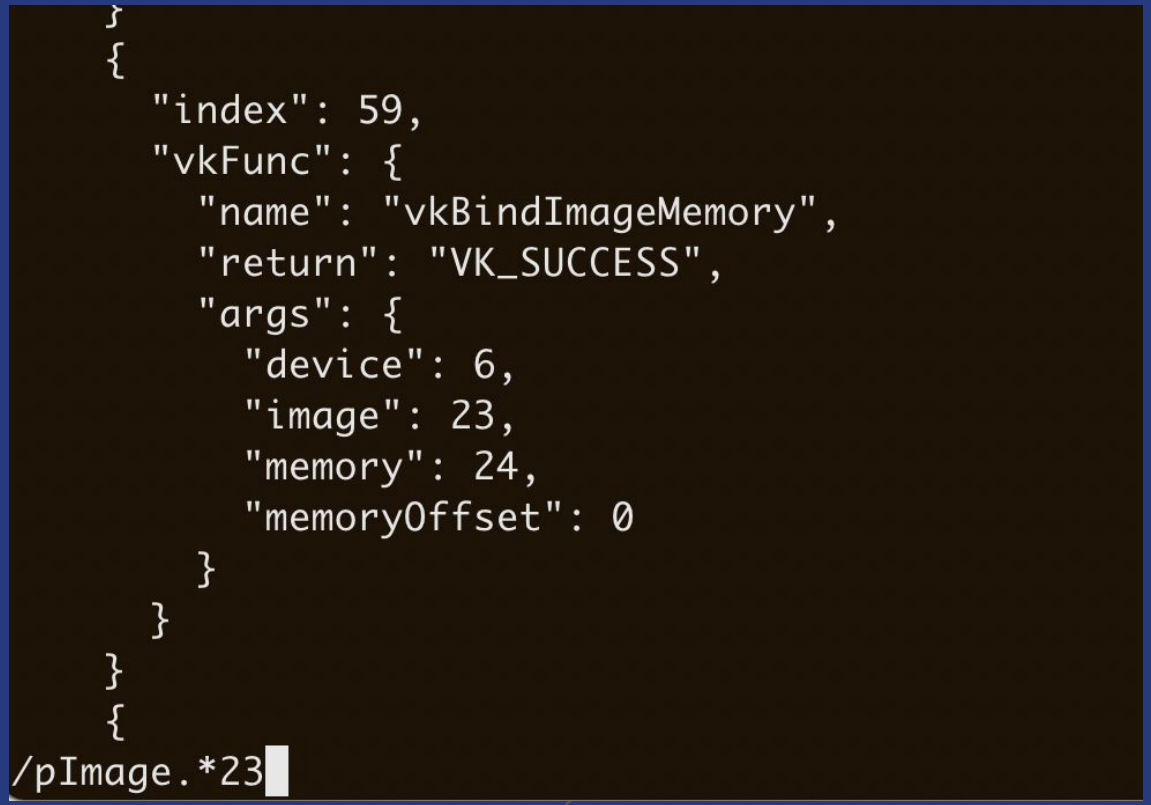
```
{
  "index": 1,
  "vkFunc": {
    "name": "vkCreateInstance",
    "return": "VK_SUCCESS",
    "args": {
      "pCreateInfo": {
        "sType": "VK_STRUCTURE_TYPE_INSTANCE_CREATE_INFO",
        "pNext": null,
        "flags": 1,
        "pApplicationInfo": {
          "sType": "VK_STRUCTURE_TYPE_APPLICATION_INFO",
          "pNext": null,
          "pApplicationName": "vkcube",
[...]
```

GFXReconstruct - `gfxrecon.py convert`



LUNARG

GFXReconstruct - `gfxrecon.py convert`



LUNARG

GFXReconstruct - `gfxrecon.py convert`


```
{
  "index": 56,
  "vkFunc": {
    "name": "vkCreateImage",
    "return": "VK_SUCCESS",
    "args": {
      "device": 6,
      "pCreateInfo": {
        "sType": "VK_STRUCTURE_TYPE_IMAGE_CREATE_INFO",
        "pNext": null,
        "flags": 0,
        "imageType": "VK_IMAGE_TYPE_2D",
        "format": "VK_FORMAT_D16_UNORM",
        "extent": {
          "width": 500,
          "height": 500,
          "depth": 1
        },
          ...
      },
      "pAllocator": null,
      "pImage": 23
    }
  }
}
```

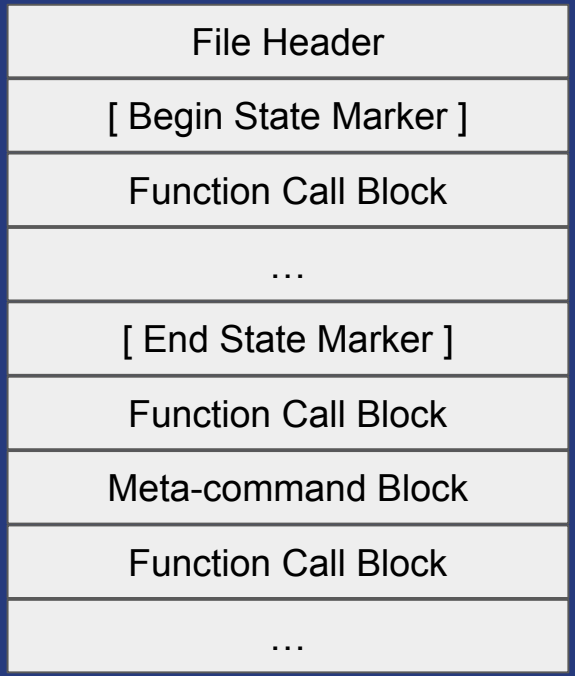

LUNARG

GFXReconstruct -- Other Tools in the Package

- `compress` - Change compression format or decompress
- `extract` - Extract shader binaries for inspection or replacement

GFXReconstruct File Format

- File header followed by a series of blocks
- Meta-command blocks, e.g.
  - Blocks to mark beginning and end of state setup
  - Set up swapchain index for trimmed range
- Each Vulkan function call generates a block
  - API call ID
    - e.g. `ApiCall_vkCreateInstance` = 0x1001
  - Input, output parameters



BUT, the preferred way to process a capture is **to subclass `Consumer`!**

GFXReconstruct Architecture

Components

- `VulkanCaptureManager` - deal with API specifics, trimming, misc
- `Encoder` - serialize API call info and parameters
- `FileProcessor` - read blocks from a file, decompress and call `Decoders`
- `Decoder` - deserialize API call info, call `Consumers`
- `Consumer` - take API call info, do something with it
  - E.g. `VulkanReplayConsumer`
  - E.g. `VulkanStatsConsumer`

GFXReconstruct

Source code directory structure

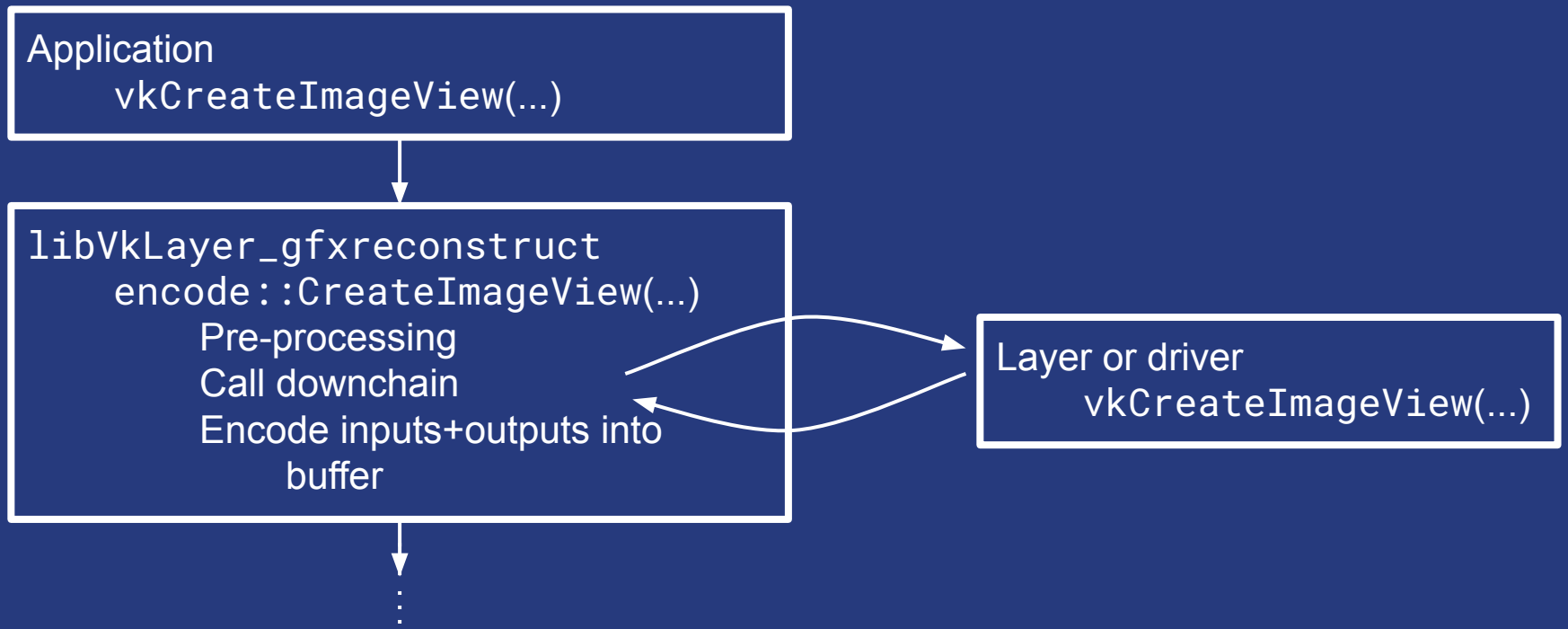
- `framework/`
  - `generated/` - generators & generated code *is checked in*
  - `encode/` - capture manager, handwritten capture, state tracking
  - `decode/` - file processing, decoding, replay, and other consumers
  - `format/` - file format metacommand structs, API call IDs
  - `util/` - etc

GFXReconstruct Architecture

Source code directory structure - cont.

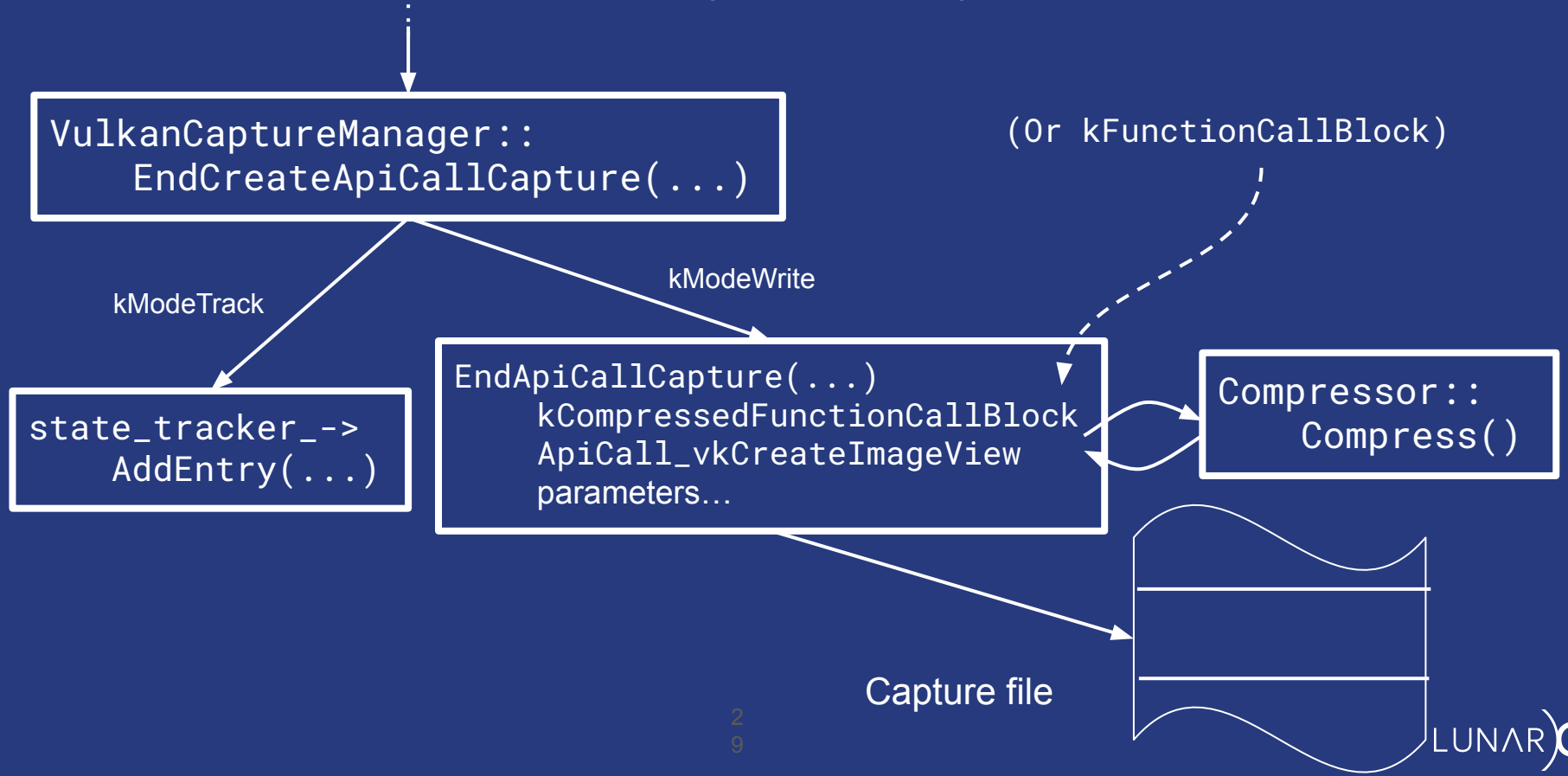
- `tools/` - settings, tool `main()`s, etc
- `layer/` - Vulkan API layer dispatch table, boilerplate, manifest
- `scripts/` - `gfxrecon.py, build.py`

GFXReconstruct - Capture

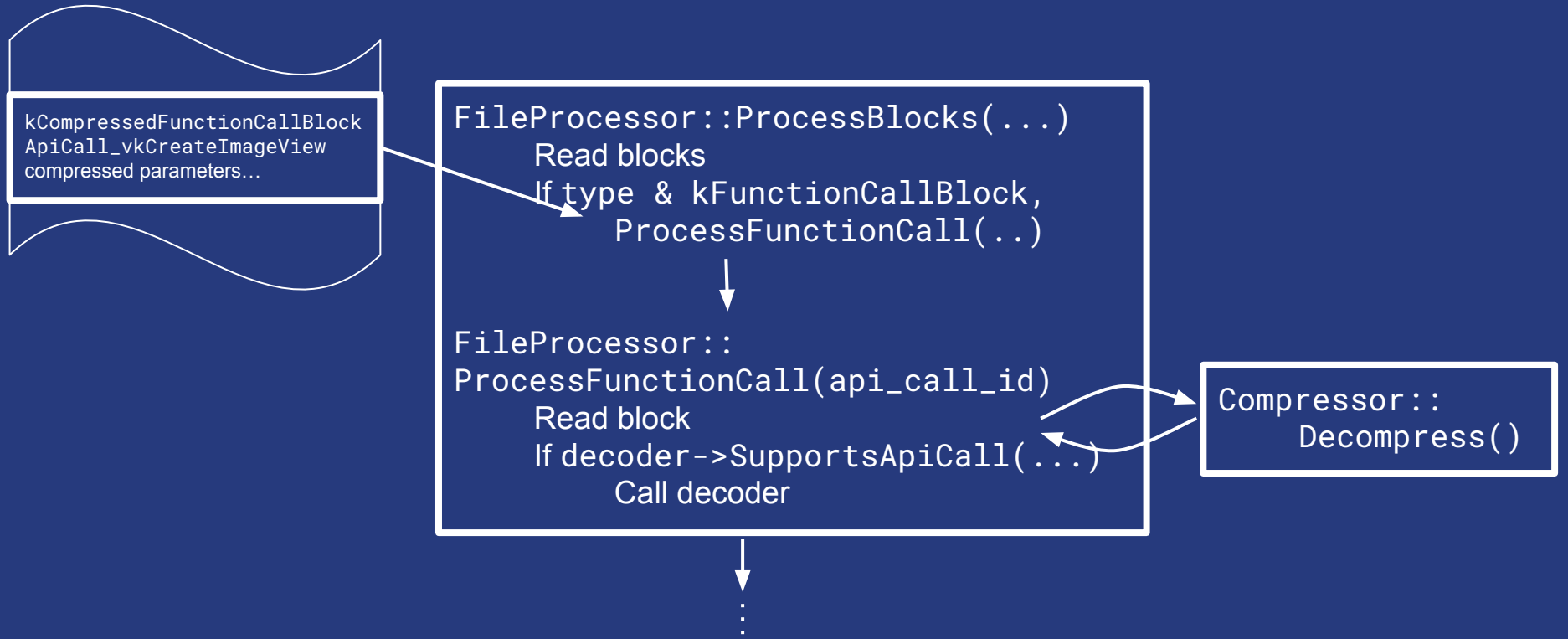


LUNARG

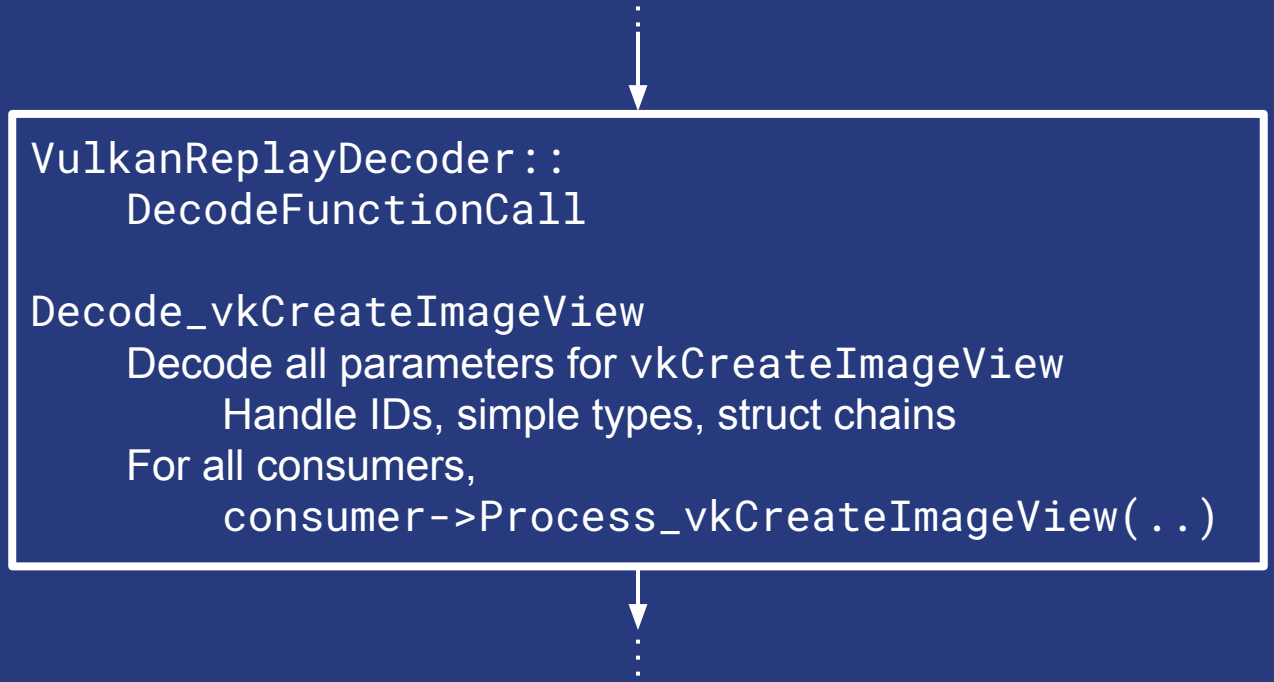
GFXReconstruct - Capture (continued)



GFXReconstruct - Replay

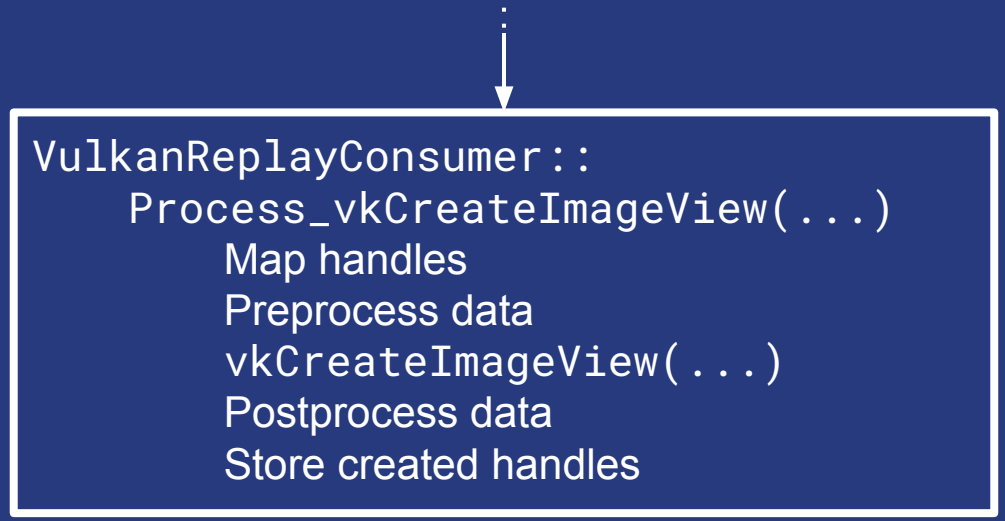


GFXReconstruct - Replay (continued)



LUNARG

GFXReconstruct - Replay (continued)



LUNARG

GFXReconstruct Future Work

Some high-priority items:

- ~~Vulkan video~~ - merged Feb 7!
- vkd3d-proton capture and replay
- Sparse residency / sparse textures

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

LUNARG

Thanks!

https://github.com/LunarG/GFXReconstruct

https://lunarg.com

brad@lunarg.com

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

This Presentation

Ecosystem Survey

LUNARG


LUNARG®

GFXReconstruct

Demos!