vktrace/vkreplay

apitrace for Vulkan

vktrace and vkreplay - Basic Roles

- vktrace
  - records Vulkan commands by writing to a binary vktrace file
  - standalone mode
  - server mode
- vkreplay
  - basically a Vulkan application
  - reads binary trace file and plays back recorded Vulkan commands
  - creates necessary windows, etc

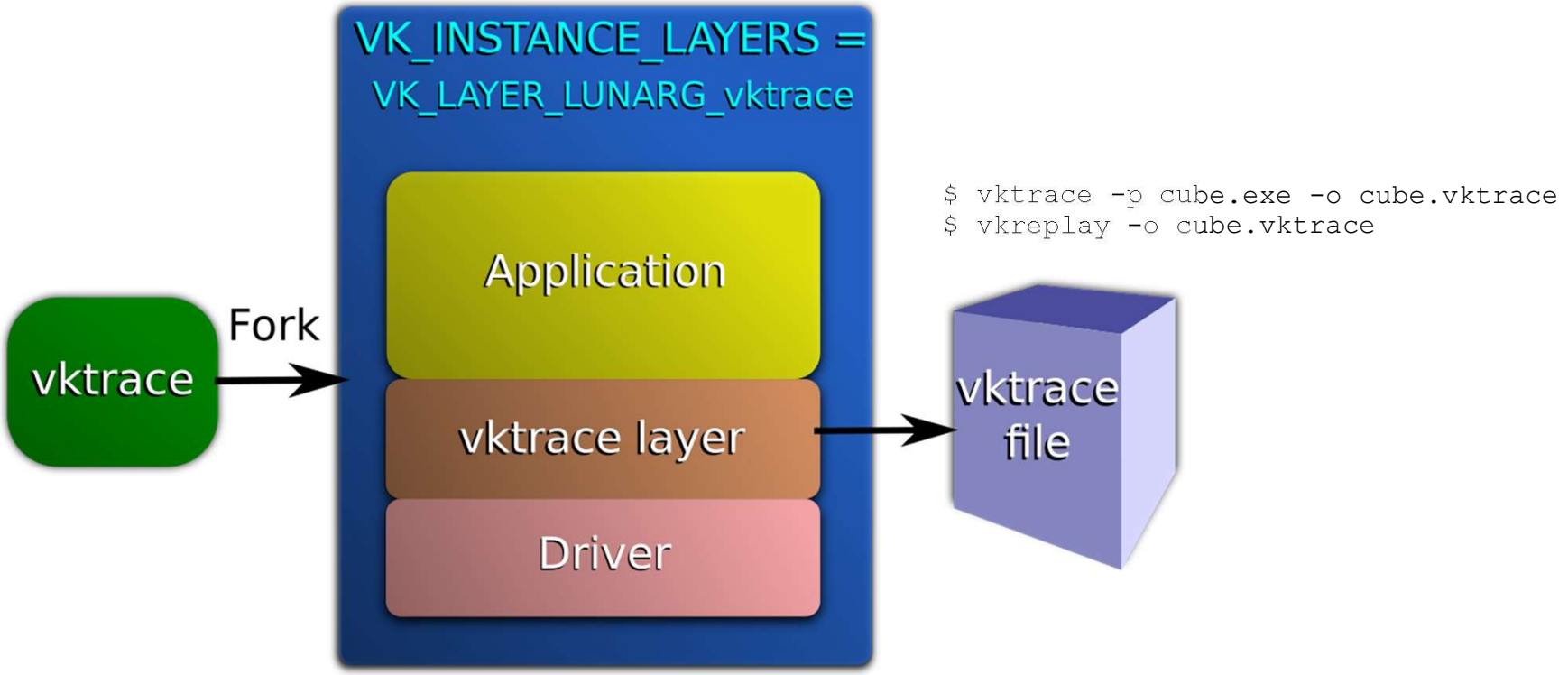
Tools can be found in the VulkanTools repo: https://github.com/LunarG/VulkanTools

Uses for vktrace and vkreplay

- Record hard-to-obtain application sequences
  - Playback for debugging drivers
  - Playback for testing drivers
- Create “library” of traces for driver testing
- Capture an application sequence to send to a driver developer as part of a bug report

Note that since Vulkan is an “explicit” API with a low level of abstraction, portability of vktrace trace files across different drivers and hardware is limited. More on this later.

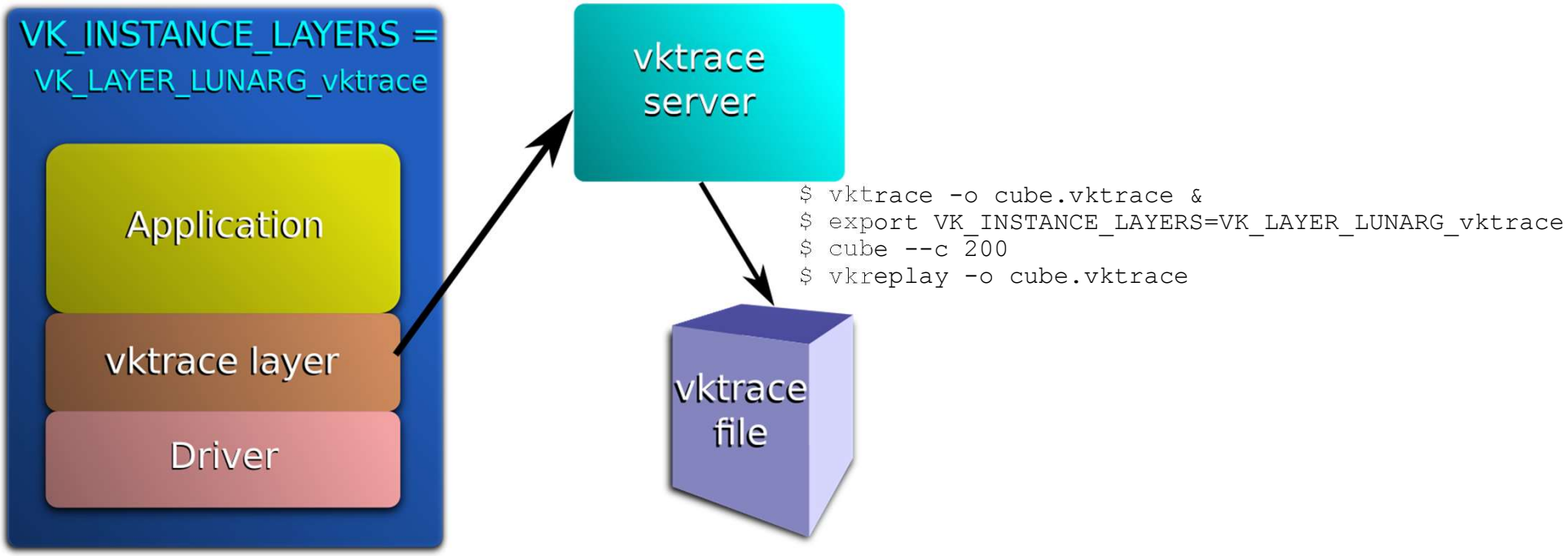
vktrace Architecture - Standalone Mode



- vktrace forks, enabling vktrace layer and starting application as a child process.

LUNARG

vktrace Architecture - Server Mode



- Application with vktrace layer collects Vulkan commands and sends to server

LUNARG

vktrace - Command Syntax

```
$ vktrace --help
vktrace available options:
    -p, --Program <string>              The program to trace.
    -a, --Arguments <string>            Command line arguments to pass to trace program.
    -w, --WorkingDir <string>           The program's working directory.
    -o, --OutputTrace <string>          Path to the generated output trace file.
    -s, --ScreenShot <string>           Comma separated list of frames to take a snapshot of.
    -ptm, --PrintTraceMessages <BOOL>   Print trace messages to vktrace console.
    -P, --PMB <BOOL>                    Optimize tracing of persistently mapped buffers, default is TRUE.
    -v, --Verbosity <string>            Verbosity mode. Modes are "quiet", "errors", "warnings", "full".
```

LUNARG

vkreplay - Command Syntax

```
$ vkreplay  --help
vkreplay available options:
    -o, --Open <string>               The trace file to open and replay.
    -t, --Tracefile <string>          The trace file to open and replay. (Deprecated)
    -l, --NumLoops <uint>             The number of times to replay the trace file or loop range.
    -lsf, --LoopStartFrame <int>      The start frame number of the loop range.
    -lef, --LoopEndFrame <int>        The end frame number of the loop range.
    -s, --Screenshot <string>         Comma separated list of frames to take a snapshot of.
    -v, --Verbosity <string>          Verbosity mode. Modes are "quiet", "errors", "warnings", "full".
```

LUNARG

Trace Server Works Across Systems

Trace server and traced program can be on different systems:

```
user@sys1 $ vktrace -o cube.vktrace &


user@sys2 $ export VK_INSTANCE_LAYERS=VK_LAYER_LUNARG_vktrace
user@sys2 $ export VKTRACE_LIB_IPADDR=sys1
user@sys2 $ cube --c 200
```

LUNARG

Replay Looping

```
$ vkreplay -t cube.vktrace -l 100
```

Replays cube.vktrace 100 times.

Make sure that api state at end of the trace file is compatible with the state expected at the start of the trace file. This usually means the application under trace should probably shut down gracefully while it is being traced.

Replay Looping - Frame Subsets

```
$ vkreplay -t cube.vktrace -l 100 -lsf 50 -lef 60
```

Replays frames 50 to 60, repeating 100 times

Make sure that api state after drawing frame 60 is compatible with the state expected at the start of frame 50. Otherwise, bad things may happen.

Android Support

- Generally uses the Linux code path, supports android-23 and beyond.
- vktrace executable on the host talks to vktrace_layer on the device over ADB.
- vkreplay APK consumes the trace on the device.
- Currently requires the following permissions to record your APK:

  `android.permission.READ_EXTERNAL_STORAGE`

  `android.permission.WRITE_EXTERNAL_STORAGE`

More info in the VulkanTools repo: https://github.com/LunarG/VulkanTools

BUILDVT.md contains details for Android.

See working, maintained examples at:

- macOS/Linux: build and test
- Windows: build and test

Persistently Mapped Buffers

Potential Tracing Challenge:
Capturing host memory writes to mapped device memory.

- Vulkan application uses vkAllocateMemory/vkMapMemory to map device memory.
- Application modifies the mapped device memory
- vktrace layer copies mapped memory to trace file during:
  - vkFlushMappedMemoryRanges
  - vkUnmapMemory
- If the application modifies mapped memory without calling one of the above API entry points, the memory changes won't be recorded to the trace file for use by subsequent API entry points
- vktrace addresses this...

LUNARG

Persistently Mapped Buffers

- PMB implementation asynchronously detects changes to mapped memory and records those changes to trace file during the next call to:
  - vkQueueSubmit
  - vkFlushMappedMemoryRanges
  - vkUnmapMemory
- PMB implementation only writes a page to the trace file if it has been modified since the last time it was written to the trace file

Persistently Mapped Buffers - Linux Solution

Linux vktrace layer

- Reads /proc/self/pagemap to determine which pages in Vulkan mapped memory have changed
- Adds those changed pages to the trace file
- Writes to /proc/self/clear_refs to clear the change status of /proc/self/pagemap

Persistently Mapped Buffers - Windows Solution

Windows vktrace layer

- Calls VirtualProtect with PAGE_GUARD to set a page guard on all Vulkan mapped memory
- Modification to Vulkan mapped memory generates an exception
- Exception handler will mark those pages as changed and make sure those pages are added to the trace file

Persistently Mapped Buffers - Android Solution

Android vktrace layer

- In progress, waiting for dust to settle on desktop implementation.
- Will mirror Linux path, but without using /proc/self/pagemap or /proc/self/clear_refs

Persistently Mapped Buffers - Cautions

Note:

Lots of writes to persistently mapped buffers can result in a very large trace file.

If changing the application is possible:

- Avoid re-writing mapped buffers when all you care about is the last write. On some platforms, vktrace will capture and record all the writes, when only the last one matters.
- Minimize the amount of mapped memory and time it is mapped.

Cross Platform Trace/Replay

Challenges

- Vulkan API is low-level so hardware details make it into the trace file
- Trace files contain choices made by application based on the hardware of the trace machine

Scenarios

- Trace and Playback environments identical
  - Should always work
- Same GPU and driver, different OS
  - Should work via WSI remapping
  - Example: XCB WSI extension calls in a trace file get mapped to Windows WSI extensions
- Same OS and GPU, different driver version
  - Can fail, depending on nature of driver changes
- Different GPU
  - The simplest applications may work

LUNARG

Coming Soon

- Screenshot
  - Specify:
    - first frame to screenshot
    - last frame to screenshot
    - interval to use between screenshots
- Trimming
  - Trim trace file to include or exclude certain frames.
  - Difficult problem because API state must be preserved
- vktraceviewer
  - View api calls in trace file
  - Will be added to LunarG's SDK in an upcoming release
- Android
  - Use sockets for file I/O to avoid need for extra permissions

LUNARG