Introducing the Vulkan WSI Layer

Rosen Zhelev
2nd Oct 2019

Introduction

- Who am I?
  Software Engineer at Arm. Working on the Mali GPU driver for the past 2 years.
- Overview
  - Introduce how windowing system integration works within the Vulkan API.
  - Describe approach to move windowing system code outside of a Vulkan driver and into a standalone Vulkan Layer.
  - Present the current status of the project and discuss the reasons behind initiating this.
    Vulkan WSI Layer project is hosted at https://gitlab.freedesktop.org/mesa/vulkan-wsi-layer
- The idea for this project has been floating for some time. It was described by Jason Ekstrand in his blog post Linux Window System Integration as a Layer (almost)

Windowing system integration for 3D Graphics drivers

- 3D Graphics APIs have a concept of a windowing system that manages and displays the rendered output to the user.
- The graphics driver needs to communicate with the windowing system through specific APIs and negotiate image formats, memory allocation and synchronization.
- For most operating systems, a single windowing system exists making it easier to support. Not so with Linux, where windowing systems have seen some extensive development in the past years – X11, Wayland, Direct to display (DRM), Mir
- It is more desirable to have this integration code aligned with the windowing system rather than the GPU focused graphics driver in order to better respond to changes.

Vulkan WSI Extensions

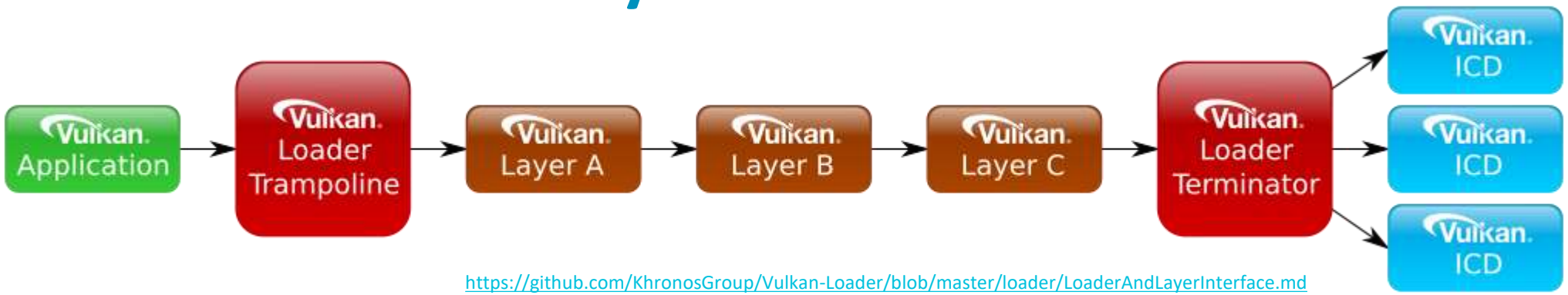
- The core Vulkan API does not provide support for windowing systems. Instead windowing system integration (WSI) is entirely handled through optional extensions.
- Instance extensions - *VK_KHR_surface* introduces a *VkSurfaceKHR* object. This represents a native window.
  - VK_KHR_wayland_surface
  - VK_KHR_xcb_surface
  - VK_KHR_xlib_surface
  - VK_KHR_display
- Device extension *VK_KHR_swapchain* – Allows creation of a *VkSwapchainKHR* connected to a surface and represents a queue of buffers that can be presented to the windowing system.

arm

Vulkan WSI implementations

- GPU drivers often make use of a defined interface to between WSI and the core rendering to achieve a clean separation between them.
- Windowing System code can be mostly standalone from the rest of the driver.
- Mesa – Accomplishes separation through something that mostly looks like a Vulkan extension. Makes significant use of DRM modifiers to describe image layout and format.
- Android – Implements most of the WSI extensions in its Vulkan Loader and just requires drivers to support the private *VK_ANDROID_native_buffer* extension.

Vulkan Loader and Layers

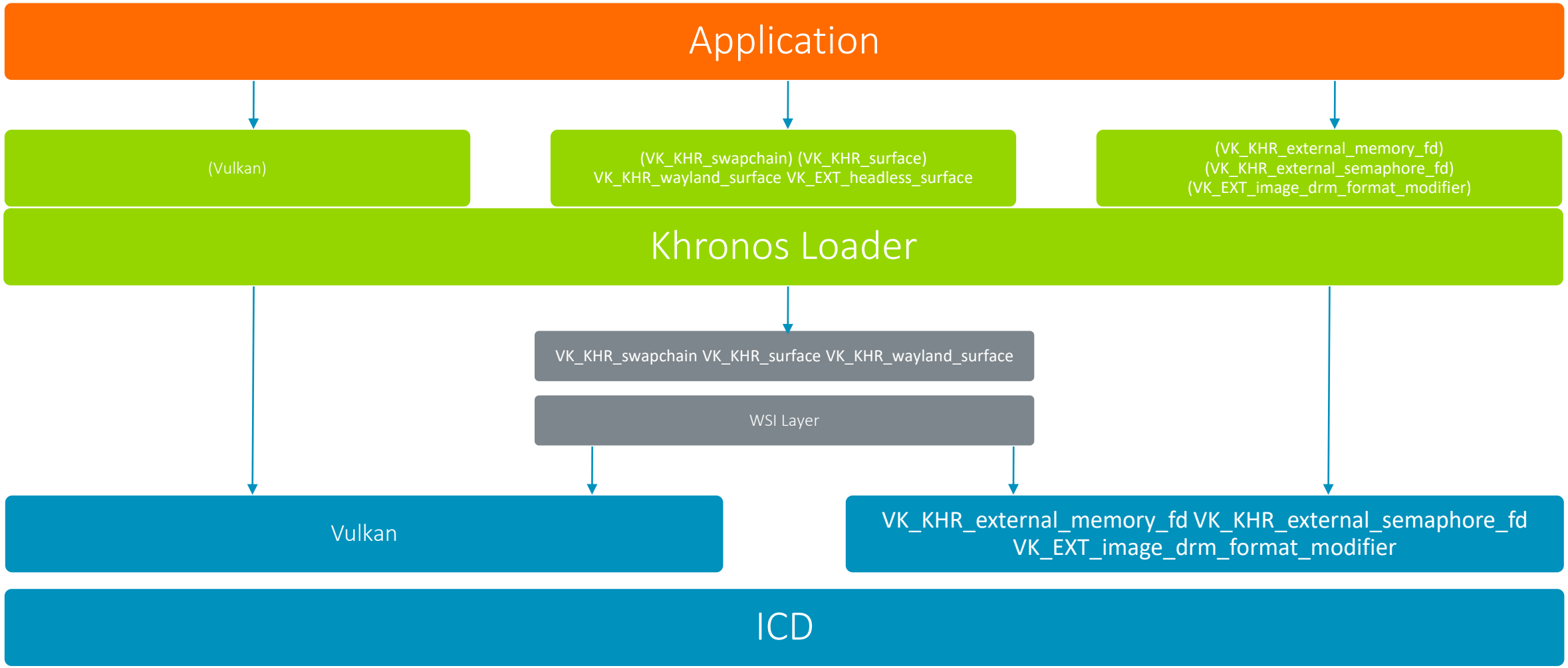


https://github.com/KhronosGroup/Vulkan-Loader/blob/master/loader/LoaderAndLayerInterface.md

- The Vulkan API does not mandate but allows for a system loader.
- Most Linux systems use the reference Khronos Loader developed by LunarG.
- The Loader allows for multiple Vulkan drivers to co-exist and enables the use of Vulkan Layers.
- Layers are an API defined way to intercept the API calls and provide additional functionality.
- Vulkan drivers are referred to as an installable client drivers (ICD) and may expose one or more GPUs as a *VkPhysicalDevice*

arm

WSI Layer Diagram



(APIs in brackets are exposed but rely on lower layers for implementation)

Getting existing WSI implementation in a Layer

- Intercept of *VK_KHR_surface* and *VK_KHR_swapchain* entrypoints.
- The Layer can be implicit always loaded by the loader, so from an Application point of view it is the same as with WSI extensions being implemented by the ICD.
- The common functionality is implemented through a shared interface by all windowing system specific implementations. Specific implementation is chosen based on the type encoded in the generic *VkSurfaceKHR*.
- Replaces the private image creation within a driver with the use of *VK_EXT_image_drm_format_modifier*
- Most of the difference between ICD implemented WSI Extension and a Vulkan Layer is in the added global bookkeeping to support multiple ICDs.

Layer Image Memory Allocation and Synchronization

- Two ways to allocate memory: Exporting DMA-BUF from the ICD itself or using a system memory allocator and importing the DMA-BUF into the Vulkan driver.
- Preference for exporting the memory from the GPU when it shares the same DRM driver with display but using a system memory allocator could be used for more or multiple vendors.

- Synchronization can be achieved by user space CPU threads and core Vulkan primitives - *VkFence* and *VkSemaphore*
- Alternatively Explicit Synchronization can be implemented by importing external fences (e.g. Sync FD *with VK_KHR_external_fence_fd* and *VK_KHR_external_semaphore_fd*)
- We may need to specify new Vulkan extensions in order to support implicit synchronisation within the Layer

arm

Current status of the WSI Layer project

- Works as an implicit layer integrating with the Vulkan Loader and providing some basic bookkeeping per instance and device.
- Implements the generic Surface and Swapchain functionality providing the abstraction which needs to be implemented for windowing system specific support.
- Initially only implements *VK_EXT_headless_surface*
- Currently a Work in Progress Wayland support exists as a Merge Request for the project.
- Wayland support adds external memory allocator interface and a simple implementation using the ION allocator.

arm

Challenges

- Introduces some overhead for the WSI implementation.
- For most windowing systems, the implementation would rely on extension that are not widely supported – *VK_EXT_image_drm_format_modifier*
- Current implementation needs some issues addressed to properly implement the Vulkan specification and not leak unexpected calls to the ICD, e.g. Layout transition changes.
- May require a system memory allocator, which is not standard on Linux.
- May need to work more widely in defining new extensions if the currently published ones are not enough to support the interface between the WSI Layer and the ICD.

Benefits

- From a software engineering perspective separating windowing system code makes good sense and fits well as part of the Loader and Layers in Vulkan.
- Same code can be shared by all Vulkan drivers, which should benefit the ecosystem with more uniform support for windowing systems and features. This also encourages Vulkan implementors to support required extensions for import/export, which have other uses outside of WSI.
- Enables more developers to contribute and may make it easier for windowing system developers work on without knowing a full 3D graphics driver stack.
- If distributed separately from the Vulkan GPU driver it may allow easier updates for windowing system fixes.

Thank You
Danke
Merci
谢谢
ありがとう
Gracias
Kiitos
감사합니다
धन्यवाद
תודה

arm