Est-ce le moment du Gallium de Vulkan?

Faith Ekstrand

XDC 2026

COLLABORA

Oui

Questions?

Is it time for Vulkan Gallium?

Faith Ekstrand

XDC 2026

COLLABORA

Let’s take a look at the current runtime…

Who implements entrypoints?

We all do!

- Drivers
- The runtime
- WSI
- Driver layers
- vkFoo2() wrappers

How are they implemented?

Often in terms of other entrypoints!

What inheritance model do we use?

All of them!

- Struct inheritance
  - nvk_foo starts with a vk_foo
- Vfunc tables
  - Dispatch tables, vfunc tables for pipelines etc.
- Containment
  - vk_command_buffer contains a vk_dynmic_graphics_state
- And everybody calls into everybody else!

COLLABORA

We heard Vulkan was a good abstraction...

Okay, real talk…

- This has all worked out far better than it had any right to
- It’s only worked because Vulkan is actually a pretty good level of abstraction for a lot of things
- It’s not really working anymore

COLLABORA

Case study #1:

VkFormatFeatureFlags2

Case study: `VkFormatFeatureFlags2`

- Introduced in Vulkan 1.3

Case study: `VkFormatFeatureFlags2`

- Introduced in Vulkan 1.3
- We have a helper to convert to v1
  - `vk_format_features2_to_features()`

Case study: `VkFormatFeatureFlags2`

- Introduced in Vulkan 1.3
- We have a helper to convert to v1
- Drivers have to call the helper everywhere

Case study: `VkFormatFeatureFlags2`

- Introduced in Vulkan 1.3
- We have a helper to convert to v1
- Drivers have to call the helper everywhere
- There are two DRM format modifier properties structs
  - Since it's an array, it couldn't be extended
  - Drivers have to manually handle both

Case study: VkFormatFeatureFlags2

- Introduced in Vulkan 1.3
- We have a helper to convert to v1
- Drivers have to call the helper everywhere
- There are two DRM format modifier properties structs
- We really can’t do much better
  - We currently have no way to wrap queries

Case study: `VkFormatFeatureFlags2`

- Introduced in Vulkan 1.3
- We have a helper to convert to v1
- Drivers have to call the helper everywhere
- There are two DRM format modifier properties structs
- We really can’t do much better
- We’re almost out of bits and Khronos is talking about a 3…

COLLABORA

Case study #2:

Subpass merging

Case study: Subpass merging

- Looks simple on the surface
  - Some heuristic decides when to merge
  - Take a union of the render targets
  - Replace input attachment reads framebuffer fetch
  - Update input and output attachment locations at subpass boundaries

Case study: Subpass merging

- Looks simple on the surface
- Have to communicate all that to the compiler
  - We have a bunch of helpers to get `vk_render_pass` versions of `pNext` structs
  - They're getting increasingly janky

Case study: Subpass merging

- Looks simple on the surface
- Have to communicate all that to the compiler
- Render target re-ordering doesn't re-order blend state
  - VK_KHR_dynamic_rendering allows changing attachment locations
  - But this only updates the mapping from shader to attachment
  - Blend is still in `VkRenderingInfo::pColorAttachments` order

Case study: Subpass merging

- Looks simple on the surface
- Have to communicate all that to the compiler
- Render target re-ordering doesn't re-order blend state
- So we add more dynamic state
  - There's now a 2nd way to re-order attachments that does re-order blend
  - The interaction between the two is poorly defined

Case study: Subpass merging

- Looks simple on the surface
- Have to communicate all that to the compiler
- Render target re-ordering doesn’t re-order blend state
- So we add more dynamic state
- We really need a better way to intercept state

COLLABORA

Case study #3:

Vulkan meta

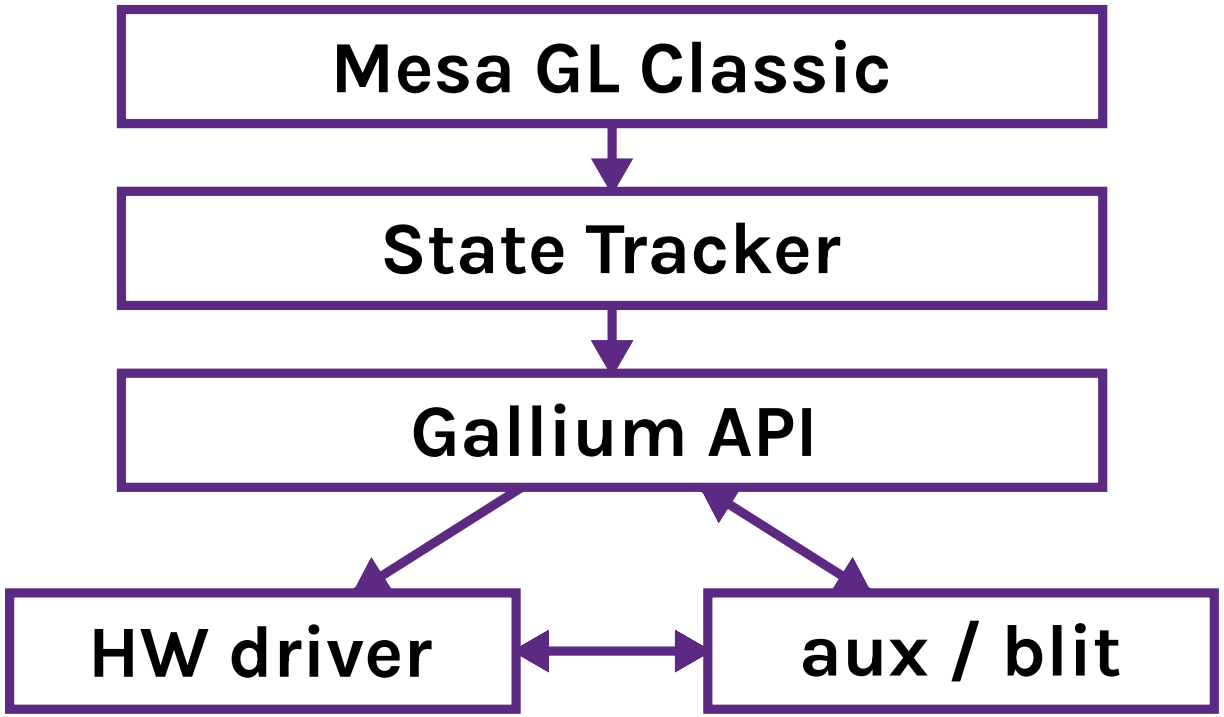
Do I even need to explain this one?

COLLABORA

The problem:

The Vulkan runtime sits inside drivers, not on top of them.

The Gallium model



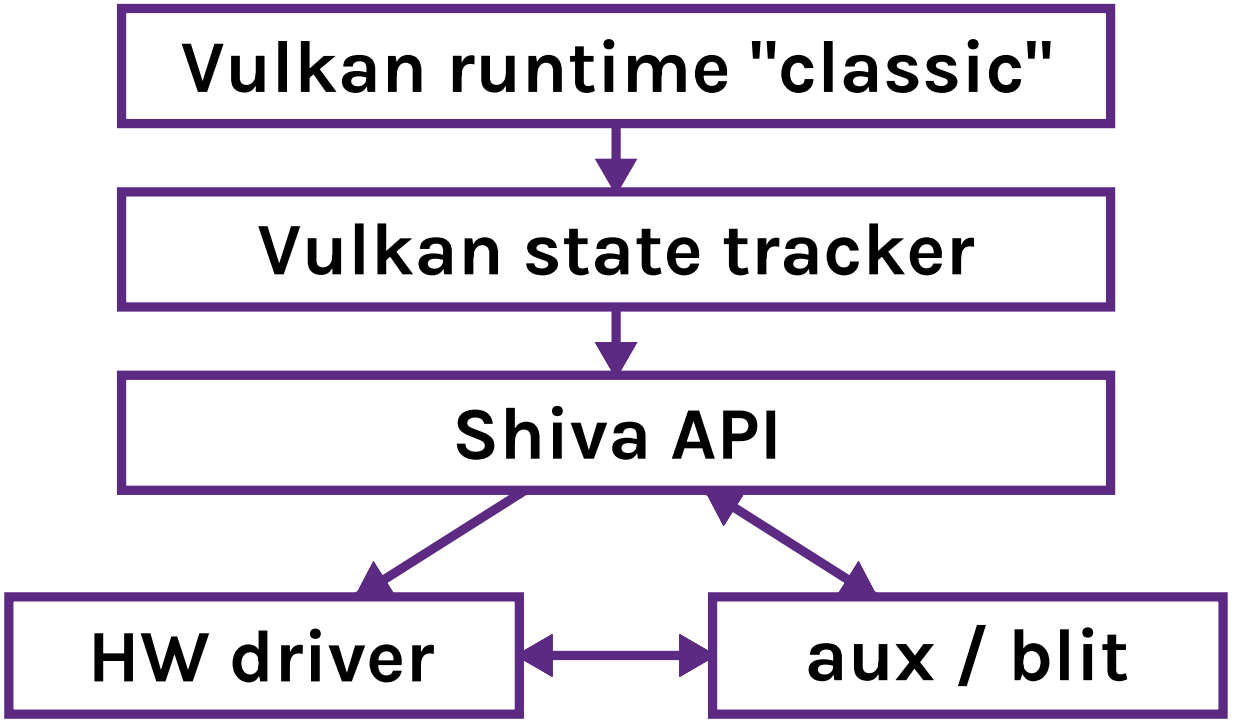
The Gallium model

- Gallium sits fully between the driver and the API
  - Drivers implement the Gallium API, not OpenGL
- Gives us more ability to sanitize the GL crazy
- Gives us a much better API for meta ops
  - `u_blitter` implements gallium in terms of gallium, not GL on GL
- Gives us a mutable API we can change as needed
  - Vulkan and OpenGL are ABI-stable so none of the crazy ever goes away

COLLABORA

Enter, Shiva...

The Shiva model



Why “Shiva”?

- Shiva is the ice goddess in the Final Fantasy mythos
- Vulcan is the god of fire
- It works…

Why Shiva?

- We need to stop implementing Vulkan directly
  - Vulkan was fun and easy to implement in 2016
  - Vulkan in 2026 is full of backwards compatibility cruft
  - It’s only going to get worse as Vulkan continues to evolve

Why Shiva?

- We need to stop implementing Vulkan directly
- We need an API we can modify and extend
  - We have a bunch of pseudo-extensions for various things
  - Sometimes they get standardized because Zink or VKD3D want them
  - Ultimately, pseudo-extensions make Vulkan more complicated

Why Shiva?

- We need to stop implementing Vulkan directly
- We need an API we can modify and extend
- We need to be able to intercept all state
  - We can sanitize dynamic state, but only if we can intercept it
  - Render passes, pipelines etc. do a sort of state intercept but it's janky
  - 90% of the current runtime is optional, including all state interception

Why Shiva?

- We need to stop implementing Vulkan directly
- We need an API we can modify and extend
- We need to be able to intercept all state
- We need an API that can safely call itself
  - Vulkan meta is a lot less dicy than OpenGL meta was
  - But it's still not great
  - State save/restore is per-driver and super janky

Why Shiva?

- We need to stop implementing Vulkan directly
- We need an API we can modify and extend
- We need to be able to intercept all state
- We need an API that can safely call itself
- We need an API that is designed for Vulkan
  - Shiva won't be a copy+paste of gallium

Why Shiva?

- We need to stop implementing Vulkan directly
- We need an API we can modify and extend
- We need to be able to intercept all state
- We need an API that can safely call itself
- We need an API that is designed for Vulkan
- We want an API that's safer for Rust and C++ drivers

COLLABORA

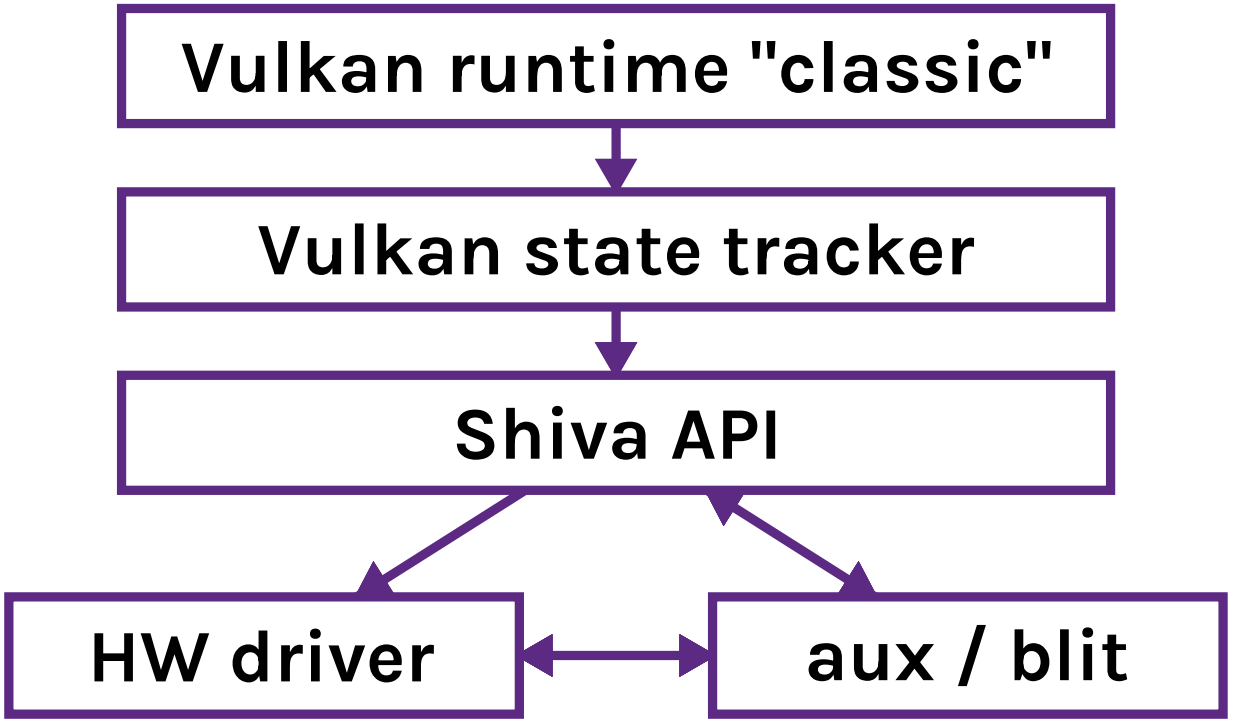
What will Shiva look like?

COLLABORA

Disclaimer:

None if this is code yet so it’s all subject to change at any moment. Don’t come complaining to me if the final thing doesn’t 100% match my XDC slides.

The Shiva design



The Shiva design

- The “classic” Vulkan runtime in Mesa will still exist
- The Shiva Vulkan state tracker will be a “classic” driver
- The state tracker will call into the Shiva API
- Drivers will implement the Shiva API
- Abstractions will migrate into Shiva as it makes sense
  - If Shiva is the only user of something, we pull it in
  - Some things will be worth duplicating because Shiva can do it better

The Shiva API design

- Sits between the Vulkan state tracker and Shiva drivers
- Modern API with as little cruft as possible
  - Shader objects, dynamic rendering, etc.
- No `pNext` chains or flags2!
  - Everything passed to drivers will be flat structs with defaults for unused features
- Clearly specified object lifetimes and ownership
- Meta-op friendly, I hope

Rust back-ends for Shiva

- Supporting Rust back-ends is a explicit design goal
- The Shiva API will auto-bind to rust
  - Vfunc tables in the C side, traits on the Rust side
- The Shiva API will use Rust's lifetime rules
- The Shiva API will use Rust concepts like slices
  - We may also provide a C++ header that gives `std::span`
- Reference counted will have Rust smart pointer types

Open Questions

- How do we define the Shiva API?
  - XML? JSON? A C header? Rust?
- How do we expose flags which may be > 64 bits?
- Should we expose one Shiva instance for all drivers?
- How should descriptors work?
- How do we want to handle dynamic state?
  - Do we want to copy gallium CSOs somehow?

COLLABORA

We'll figure all this out as we go

Shiva project plan / timeline

- I (Faith) plan to start work on Shiva right after XDC
  - There's a Khronos meeting next week
  - And a few bits of Kraid to wrap up...
  - But I should be free by the end of October

Shiva project plan / timeline

- I (Faith) plan to start work on Shiva right after XDC
- It will probably be a year before others can join
  - There are a lot of details to work out
  - The whole thing will probably get rewritten multiple times
  - The new API needs an editorial voice, not design-by-committee

Shiva project plan / timeline

- I (Faith) plan to start work on Shiva right after XDC
- It will probably be a year before others can join
- Initial prototyping will be on Nouveau and Panfrost
  - Those are the two hardwares I know best
  - The Nouveau back-end will be written in Rust
  - The Panfrost back-end will be written in C

Shiva project plan / timeline

- I (Faith) plan to start work on Shiva right after XDC
- It will probably be a year before others can join
- Initial prototyping will be on Nouveau and Panfrost
- Best guess is 2 years before Shiva be ready for users
  - This is just for Nouveau and maybe Panfrost
  - We need to prove it out before we get too excited about porting

Shiva project plan / timeline

- I (Faith) plan to start work on Shiva right after XDC
- It will probably be a year before others can join
- Initial prototyping will be on Nouveau and Panfrost
- Best guess is 2 years before Shiva be ready for users
- Migrating other drivers will happen slowly
  - It took 10 years to move everyone to gallium.
  - I don’t expect the Shiva migration to go much faster

Shiva project plan / timeline

- I (Faith) plan to start work on Shiva right after XDC
- It will probably be a year before others can join
- Initial prototyping will be on Nouveau and Panfrost
- Best guess is 2 years before Shiva be ready for users
- Migrating other drivers will happen slowly
- This is all a best guess!

COLLABORA

Wish me luck!

Thank you!

We are hiring

col.la/careers