Vulkanised 2026

The 8th Vulkan Developer Conference
San Diego, USA | February 9-11, 2026

Vulkan: Forging Ahead

Ralph Potter, Vulkan WG Chair

Focusing on Usability

Vulkan was released nearly a decade ago, we've learned some things

Experience has allowed us to design more ergonomic APIs (dynamic rendering)
Hardware constraints have shifted over that time (increased dynamic state)
We have received a decades worth of developer feedback

Vulkan was designed as an explicit API with no compromise

Pre-existing hardware design meant a complex API
Now asking hardware to evolve to fix rough edges
Designing new features to be a “Joy to Use”
See dynamic rendering, unified image layouts, shader objects

Navigating Vulkan is becoming challenging

The API surface has become large, with multiple ways to solve single tasks
How can we help developers better navigate the API?

We are working to tackle complexity, usability and effectively communicate how to use the API today

What are we doing?

Reducing Complexity

Today

- Tooling and SDK
- Descriptor Heaps

Short Term

- Cross-vendor extensions
- Revising legacy features
- Focus on “joy to use”
- Maintenance extensions

Long Term

- Hardware Requirements in Roadmap
- Raising Core Baselines

Improving Accessibility

Today

- Previewing API simplification (Pier’s talk)

Short Term

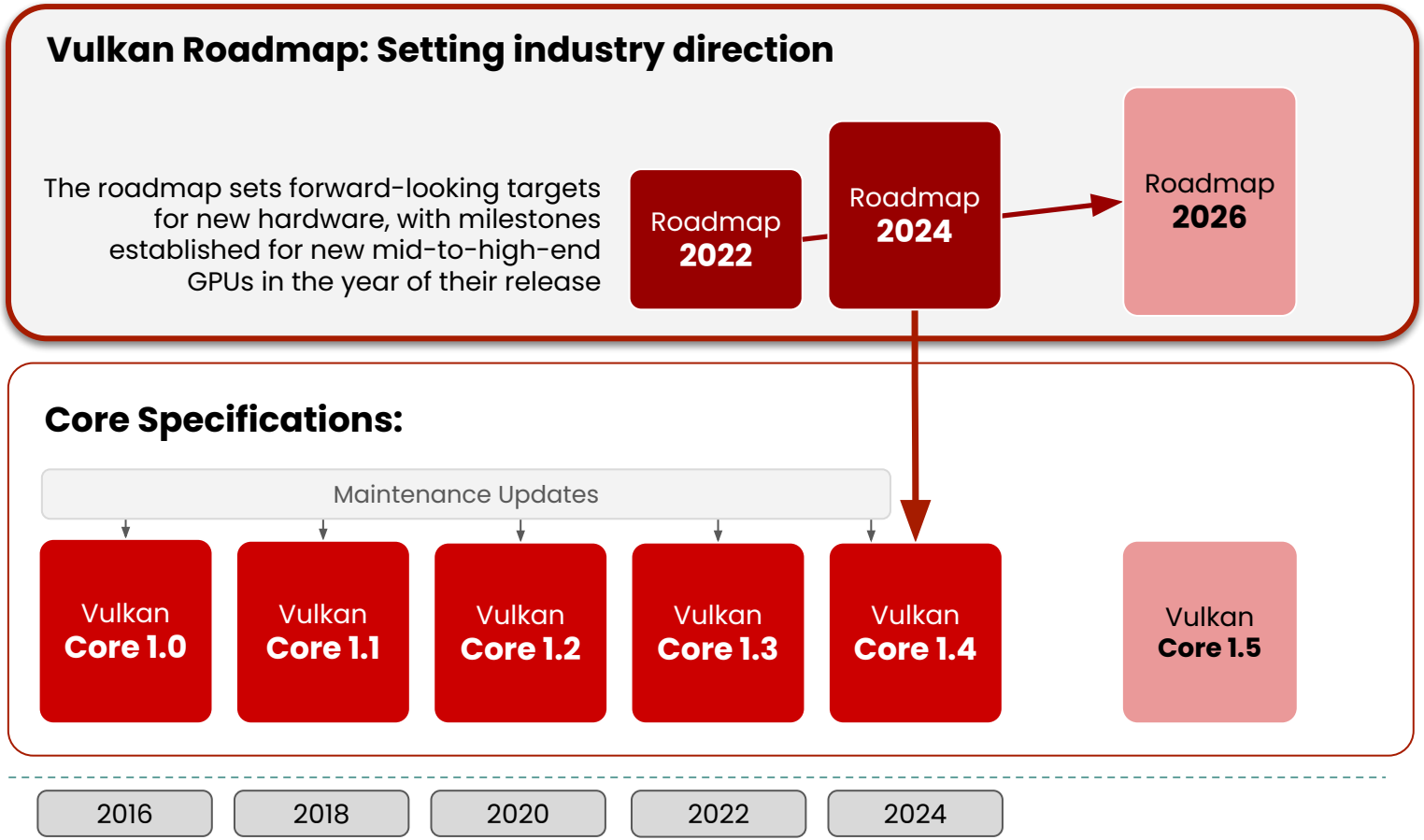
- Samples
- Tutorials and Guide Docs
- Shading Language Symposium
- Vulkanised
- Documentation Improvements
- Formal Deprecation Model

Long Term

- Vulkan Educator’s Forum
- Focus on “whole cloth” replacement

Tackling Fragmentation

Roadmap 2026



Roadmap 2026

- **Vulkan 1.4, Roadmap 2024 plus…**
- Variable rate shading
- Shader clock queries
- Cooperative matrix
- Host image copies
- Compute shader derivatives
- Various swapchain improvements
- Higher descriptor and shader interface limits
- … and more (20 extensions, 4 required features, 21 limit increases)

https://docs.vulkan.org/spec/latest/appendices/roadmap.html#roadmap-2026

Let’s Talk About ~~Deprecation~~ Legacy APIs

Let's talk about ~~Deprecation~~ Legacy APIs

There are NO plans to remove functionality from Vulkan

Applications need to support a broad range of hardware with long lifetimes
Maintaining multiple duplicate code paths incurs significant costs to ISVs
Strong desire for backward compatibility, so older apps keep running

There may be market specific reasons to use legacy functionality

Superseding functionality will take time to achieve widespread adoption in some markets
Embedded and mass-market mobile have a very long tail
We want to highlight where developers should focus their efforts

We CAN make it easier to focus on the functionality you want

The specification already includes ad-hoc references to deprecated/superseded functionality
We are formalizing that, and beginning to reflect that in the specification and tooling

Let’s talk about Legacy APIs

Our API XML Schema now tags legacy functionality

Enables us to build and maintain tooling/documentation describing legacy functionality
This tagging reflects through to the specification

To create a render pass, call:

WARNING

This functionality is superseded by Vulkan Version 1.4. See Legacy Functionality for more information.

```
// Provided by VK_VERSION_1_2
VkResult vkCreateRenderPass2(
    VkDevice                                    device,
    const VkRenderPassCreateInfo2*              pCreateInfo,
    const VkAllocationCallbacks*                pAllocator,
    VkRenderPass*                               pRenderPass);
```

```
// Provided by VK_KHR_create_renderpass2
// Equivalent to vkCreateRenderPass2
VkResult vkCreateRenderPass2KHR(
    VkDevice                                    device,
    const VkRenderPassCreateInfo2*              pCreateInfo,
    const VkAllocationCallbacks*                pAllocator,
    VkRenderPass*                               pRenderPass);
```

Let’s talk about Legacy APIs

Explanatory Notes

We provide a dedicated appendix providing further detail and recommendations

Render Pass Objects: Superseded via dynamic rendering

VK_KHR_dynamic_rendering and Vulkan 1.3 added a new way to specify render passes without needing to create VkFramebuffer and VkRenderPass objects. However, subpass functionality had no equivalent, meaning dynamic rendering was only suitable as a substitute for content not using subpasses.

VK_KHR_dynamic_rendering_local_read and Vulkan 1.4 later allowed the expression of most subpass functionality in core or extensions. Any subpass functionality which was not replicated is still expressible but requires applications to split work over multiple dynamic render pass instances. Functionality not covered with local reads would result in most or all vendors splitting the subpass internally.

Let’s talk about Legacy APIs

Validation Layer Support

Validation layers now have optional support for reporting the use of deprecated/superseded entrypoints in validation layers

Vulkan Object Scripts for CodeGen Support

https://github.com/KhronosGroup/vulkan-object

There’s more to come…

Continuing to expand the list of legacy functionality

WG discussing opt-in compiler warnings or producing a header that contains only current recommended functionality

Introducing Descriptor Heaps

Descriptor Heaps

- **A full revision of the descriptor API**
  - Descriptor sets/layouts are no longer needed
  - Descriptor buffers are deprecated
  - Mapping APIs provide flexible compatibility for all existing APIs

- **Focus on usability**
  - Brand new entry points, limiting pNext chains
  - Straightforward interfaces to new functionality
  - Surface area of new API is tiny (memcpy does most of the work!)

- **Bindless as standard**
  - No concessions for fixed binding HW
  - Non-uniform indexing required, and assumed by default (in SPIR-V)

Descriptors as Data

- **Descriptors are the HW description of a resource**
  - Textures and samplers
  - Storage images
  - Texel buffers
  - UBOs and SSBOs
  - Acceleration structures
- **Descriptors are small blobs of data that tell the GPU how to access a resource**
  - Buffers are typically base address + size
  - Images are typically base address + layout information (MIPs, tiling etc.)
  - Descriptor implementations are HW-specific and vary in size
- **What if we just let you manage these blobs of data yourself?**
  - i.e. memcpy them around, store them in GPU memory, access them directly in shaders

https://www.gfxstrand.net/faith/blog/2022/08/descriptors-are-hard/

Obtaining Descriptors

- Allocate your resources as normal, then call:

```
VkResult vkWriteSamplerDescriptorsEXT(
    VkDevice                                device,
    uint32_t                                samplerCount,
    const VkSamplerCreateInfo*              pSamplers,
    const VkHostAddressRangeEXT*            pDescriptors);

VkResult vkWriteResourceDescriptorsEXT(
    VkDevice                                device,
    uint32_t                                resourceCount,
    const VkResourceDescriptorInfoEXT*      pResources,
    const VkHostAddressRangeEXT*            pDescriptors);
```

- Supports batched retrieval, unlike EXT_descriptor_buffer

Allocating Heap Storage

- **Implementations need some storage reserved for internal usage**
  - Query `VkPhysicalDeviceDescriptorHeapPropertiesEXT` for required sizes
  - Implementations need some space for their own use, and you need to accommodate that
- **Allocate a buffer with** `VK_BUFFER_USAGE_SHADER_DEVICE_ADDRESS_BIT | VK_BUFFER_USAGE_DESCRIPTOR_HEAP_BIT_EXT` with space for your descriptors plus reserved space
- **Retrieve the buffer device address via** vkGetBufferDeviceAddress
- Copy your descriptors into your heap storage

Binding Heaps

```c
typedef struct VkBindHeapInfoEXT {
    VkStructureType                 sType;
    const void*                     pNext;
    VkDeviceAddressRangeEXT         heapRange;
    VkDeviceSize                    reservedRangeOffset;
    VkDeviceSize                    reservedRangeSize;
} VkBindHeapInfoEXT;

void vkCmdBindSamplerHeapEXT(
    VkCommandBuffer                 commandBuffer,
    const VkBindHeapInfoEXT*        pBindInfo);

void vkCmdBindResourceHeapEXT(
    VkCommandBuffer                 commandBuffer,
    const VkBindHeapInfoEXT*        pBindInfo);
```

How do I draw with these things?

- **Two approaches: binding API vs. untyped pointers**

- **Binding API:**
  - Use your existing location/binding based shaders unmodified
  - Add mappings from shader location/bindings to heap locations at vkCreate*Pipeline
  - Use VkShaderDescriptorSetAndBindingMappingInfoEXT to express this mapping

- **Modify your shaders for direct access into the heaps**

How do I draw with these things?

- **Two approaches: binding API vs. untyped pointers**

- **Binding API:**
  - Use your existing location/binding based shaders unmodified
  - Add mappings from shader location/bindings to heap locations at vkCreate*Pipeline
  - Use VkShaderDescriptorSetAndBindingMappingInfoEXT to express this mapping

- **Modify your shaders for direct access into the heaps**

https://docs.vulkan.org/guide/latest/descriptor_heap.html

How do I draw with these things?

```
// Typed - Standard way to provide the set/binding location
layout(set = 0, binding = 0) buffer SSBO {
    vec4 payload;
} s_buffers[];

layout(set = 0, binding = 1) buffer UBO {
    vec4 payload;
} u_buffers[];

// -----

// Untyped - Both are bound to the heap
layout(descriptor_heap) buffer SSBO {
    vec4 payload;
} s_buffers[];

layout(descriptor_heap) uniform UBO {
    vec4 payload;
} u_buffers[];
```

VK_KHR_unified_image_layouts

So Long, Image Layouts: Simplifying Synchronization

Synchronization is a major developer pain point

Tracking image layouts and managing layout transitions introduces application complexity

Today many Image Layout Transitions resolve to No-Ops

Image layouts typically control image (de)compression

Hardware has become more capable since Vulkan 1.0

Use VK_IMAGE_LAYOUT_GENERAL (almost) everywhere

Continue to use VK_IMAGE_LAYOUT_UNDEFINED for invalidation

We still need layout transitions for external sharing (including queue present)

https://www.khronos.org/blog/so-long-image-layouts-simplifying-vulkan-synchronisation

Ray Tracing Shader Execution Reordering

- New multi-vendor extension:
  VK_EXT_ray_tracing_invocation_reorder

- Enables Vulkan implementations to repack RT shader invocations to reduce divergence

https://www.khronos.org/blog/boosting-ray-tracing-performance-with-shader-execution-reordering-introducing-vk-ext-ray-tracing-invocation-reorder

EXT_present_timing

- **State of the Art Frame Pacing for Vulkan**
  - Culmination of 6 years of work
- **Vulkan devices now have the ability to:**
  - Receive timing feedback about previous presentation requests
  - Explicitly specify a target presentation time for each request
- **Supported on Windows (NVIDIA), Linux (Mesa/NVIDA) today. Coming to Android 17**

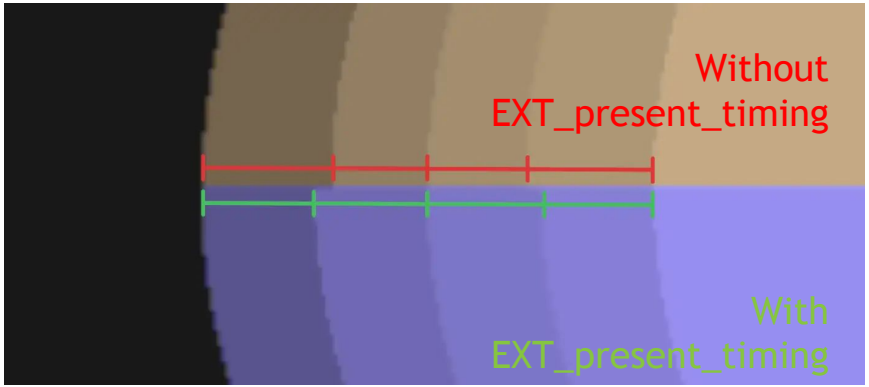


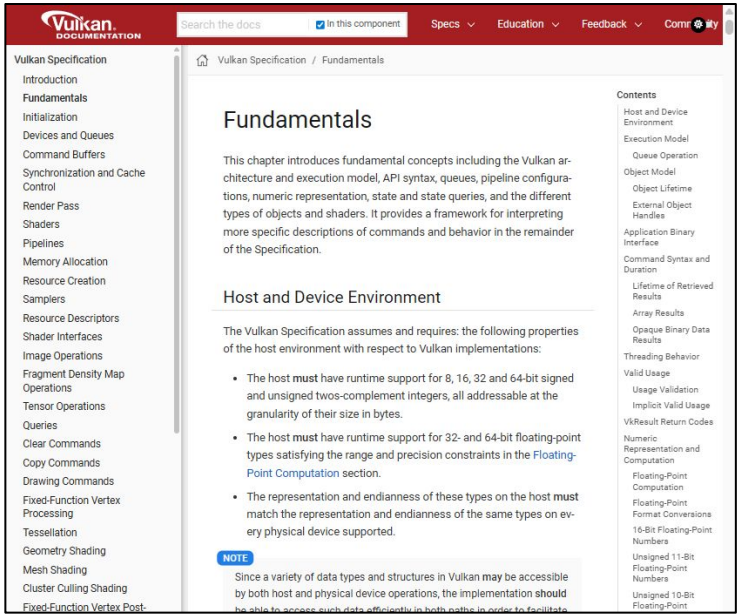
Fig. 1: Blended frames of an animated circle moving linearly from left to right.

https://www.khronos.org/blog/vk-ext-present-timing-the-journey-to-state-of-the-art-frame-pacing-in-vulkan

Documentation and Samples

Unified Documentation

- https://docs.vulkan.org/
- **Unifies documentation into a single, searchable location:**
  - Vulkan specification
  - Vulkan Guide
  - API proposal docs
  - Vulkan Samples documentation and tutorials
  - Khronos Vulkan Tutorial (derived from vulkan-tutorial.com - with many thanks!)
  - Shader language documentation - GLSL, HLSL
- **Launched Q4 2024 with ongoing improvements since then**
  - SPIR-V specs coming soon!



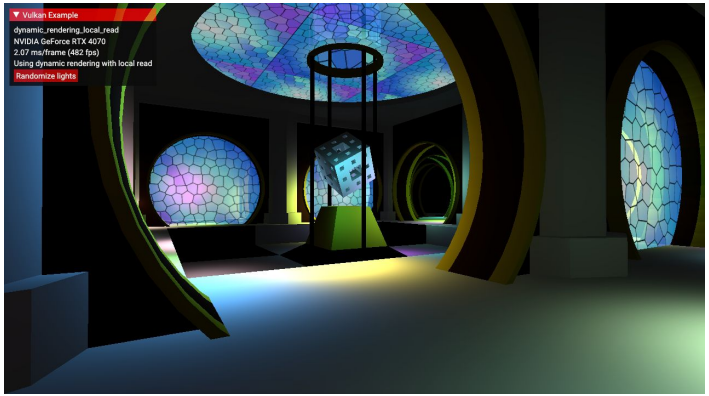
Vulkan Guide

- **New Chapters:**
  - Layers
  - Descriptor Heaps and Buffers
  - Deprecation chapter
  - IDE Integration
  - Windowing/Audio/Input
  - Vulkan Profiles
  - Vulkan versions & portins
  - Debug utils
  - Storage images and texel buffers
  - Primitive Topology chapter
  - Descriptor Array
  - Image Copy
  - Swapchain Semaphores

Vulkan Samples

- **Modernizing the Core:** The Vulkan team recently released several updates to the Vulkan Tutorial focused on streamlining to reflect the latest API specifications and development practices aimed at making it easier to write clean and efficient code (includes C++20 modules, Vulkan 1.4 alignment, Dynamic Rendering, modernized Synchronization, and more)

- **Integration with Modern Toolchains:** The Vulkan Samples now work seamlessly with popular CMake-based build systems and integrate tightly with the Vulkan SDK

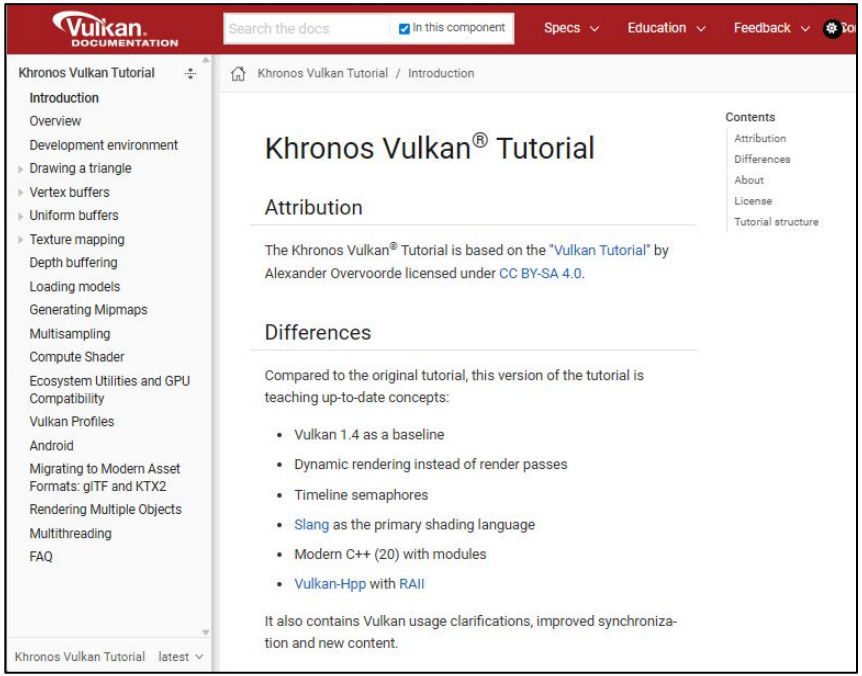


PLACEHOLDER

- **Samples/Tutorials**
  - Updated vulkan_tutorial: Vulkan 1.4 baseline, Dynamic Rendering, Timeline Semaphores, Slang, Modern C++ and more
  - Complex Sample/Tutorial: Building a Simple Game Engine
  - Support for Apple compatibility using KosmicKrisp
  - Slang and HLSL shaders added for many samples
  - New Vulkan-HPP versions of several samples
  - Recent extension samples:
    - VK_EXT_raytracing_invocation_reorder
    - VK_EXT_fragment_density_map
    - VK_EXT_dynamic_rendering_local_read
    - VK_ARM_tensores and VK_ARM_data_graph sample
    - Hopefully before Vulkanised
      - VK_EXT_device_fault
      - VK_KHR_pipeline_binaries
      - Complex sample: Octomap Rendering
      - Several others TBD

Vulkan Tutorial

- **New Vulkan Tutorial Chapters for Real-World Applications:**
  - Shader authoring with Slang
  - Modern asset formats
  - Ecosystem utilities
  - Vulkan Profiles
  - Android support
  - …and more



Thanks to Alexander Overvoorde for the original Vulkan Tutorial!

Building a Simple Game Engine

- **Tutorial complete? Now what?**
  - Engine architecture and software design patterns
  - Scene management using hierarchical object systems
  - Camera systems and flexible control schemes
  - Efficient resource and GPU memory management
  - Entity Component System (ECS) design and integration
  - Render system abstraction and extensibility
  - Input handling and robust game loop design with accurate timing

https://docs.vulkan.org/tutorial/latest/Building_a_Simple_Engine/introduction.html

New Vulkan Educators Forum

- We want to help Vulkan educators integrate the latest API improvements into your curriculum and educational materials
- The Khronos Vulkan team has initiated a Discord channel to provide a forum for Vulkan educators to share their experience teaching Vulkan. The intent is to facilitate collaboration across the Khronos Vulkan team and interested educational institutions
- If you are involved in teaching Vulkan (online or in-person) and would like to participate in this invite-only forum, please contact Vulkan Marketing and Developer Relations (marketing@khronosgroup.org)

Resources

- Vulkan: https://www.vulkan.org/
  - Unified Documentation: https://docs.vulkan.org/
  - Spec GitHub Repo: https://github.com/KhronosGroup/Vulkan-Docs/
  - Discord Link for community discussion: https://discord.com/invite/vulkan

Thank You!

Slide Deck Template

CC-BY
January 2024

Guidelines, Tips and Tricks

- **Layouts**
  - Every slide should use one of the eight available layouts (see layout button)
  - Click Reset early and often to make sure you are using the layout!
  - Don’t delete slides in this template until you have used all the layouts you need
    - PowerPoint RANDOMLY deletes unused layouts (use two indents sparingly)
- **Text**
  - Use Trebuchet font for ALL text
  - Don’t insert empty lines within layout text boxes
- **Graphics**
  - Do not use shading or shadows on graphics
  - Try to connect your lines to boxes to make editing easier
- **Animations and Transitions**
  - Don’t forget to check them before presenting! Don’t use transitions on Zoom
- **Don’t create boring slides with just text (like this one!)**
  - Use more pictures and less text to get your message across

Use this blank layout when your slide content is self explanatory
and you don't need a title

This is the default standalone text box style

Add a background and/or outline using 'Format Shape'
Automatically fit the text to the outline box using Autofit
If a box with no text mysteriously won't change size - turn off Autofit!

Alternative to Right-aligned Bullets

Brief overview of Khronos compute acceleration standards

And why they might be of interest to the RISC-V Community

Deeper dive into OpenCL

Including roadmap developments

Discussion on how Khronos and RISC-V could collaborate

Khronos is open to any organization – please get directly involved if you wish!

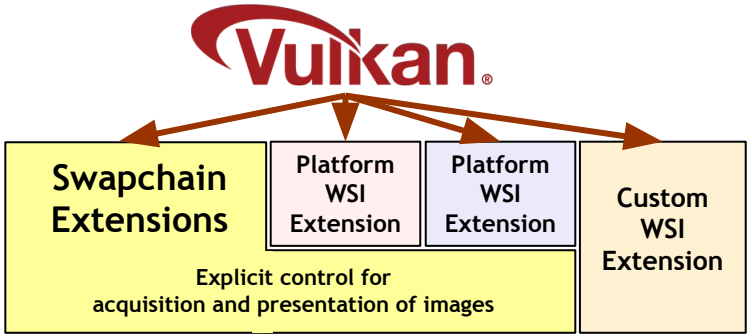
We welcome feedback and cooperation between organizations

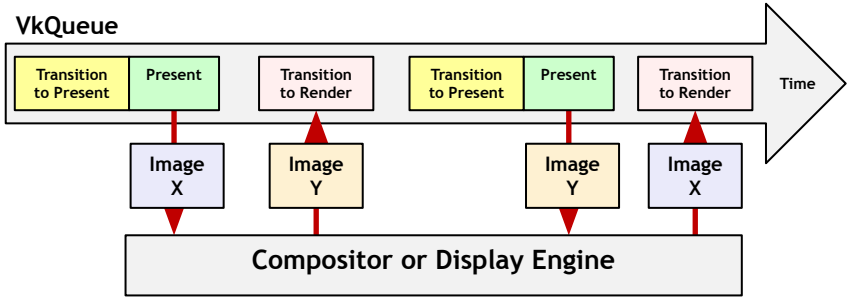
These slides will be available online

www.khronos.org

Smaller Font Bullets Make Space for Graphics

- **Explicit control for acquisition and presentation of images**
  - Designed to fit the Vulkan API and today's compositing window systems
  - Cleanly separates device creation from window system
- **Platform provides an array of persistent presentable images = Vulkan Swapchain**
  - Device exposes which queues support presentation
  - Application explicitly controls which image to render and present
- **Standardized extensions - unified API for multiple window systems**
  - Works across Android, Mir, Windows (Vista and up), Wayland and X (with DRI3)
  - Platforms can extend functionality, define custom WSI stack, or have no display at all





Medium Bullets

- **Broad commercial uptake of OpenCL**
  - Imaging, video, vision, simulation
  - Adobe, Apple, SONY, Corel, ArcSoft
  - Dassault, Houdini, Mathematica, MAYA…
- **"OpenCL" on Sourceforge, Github, Google Code, Bitbucket finds over 2,000 projects**
  - OpenCL implementations - Beignet, pocl
  - VLC, X264, FFMPEG, Handbrake
  - GIMP, ImageMagick, IrfanView
  - Hadoop, Memcached
  - WinZip, Crypto++ Etc. Etc.
- **Desktop benchmarks use OpenCL**
  - PCMark 8 – video chat and edit
  - Basemark CL, CompuBench Desktop

https://www.khronos.org/opencl/resources/opencl-applications-using-opencl

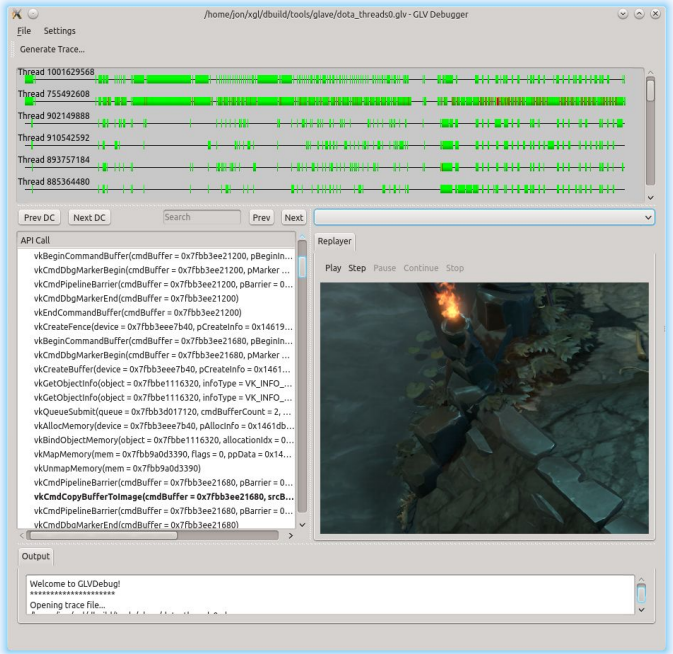
Short Bullets

- To accompany larger graphics

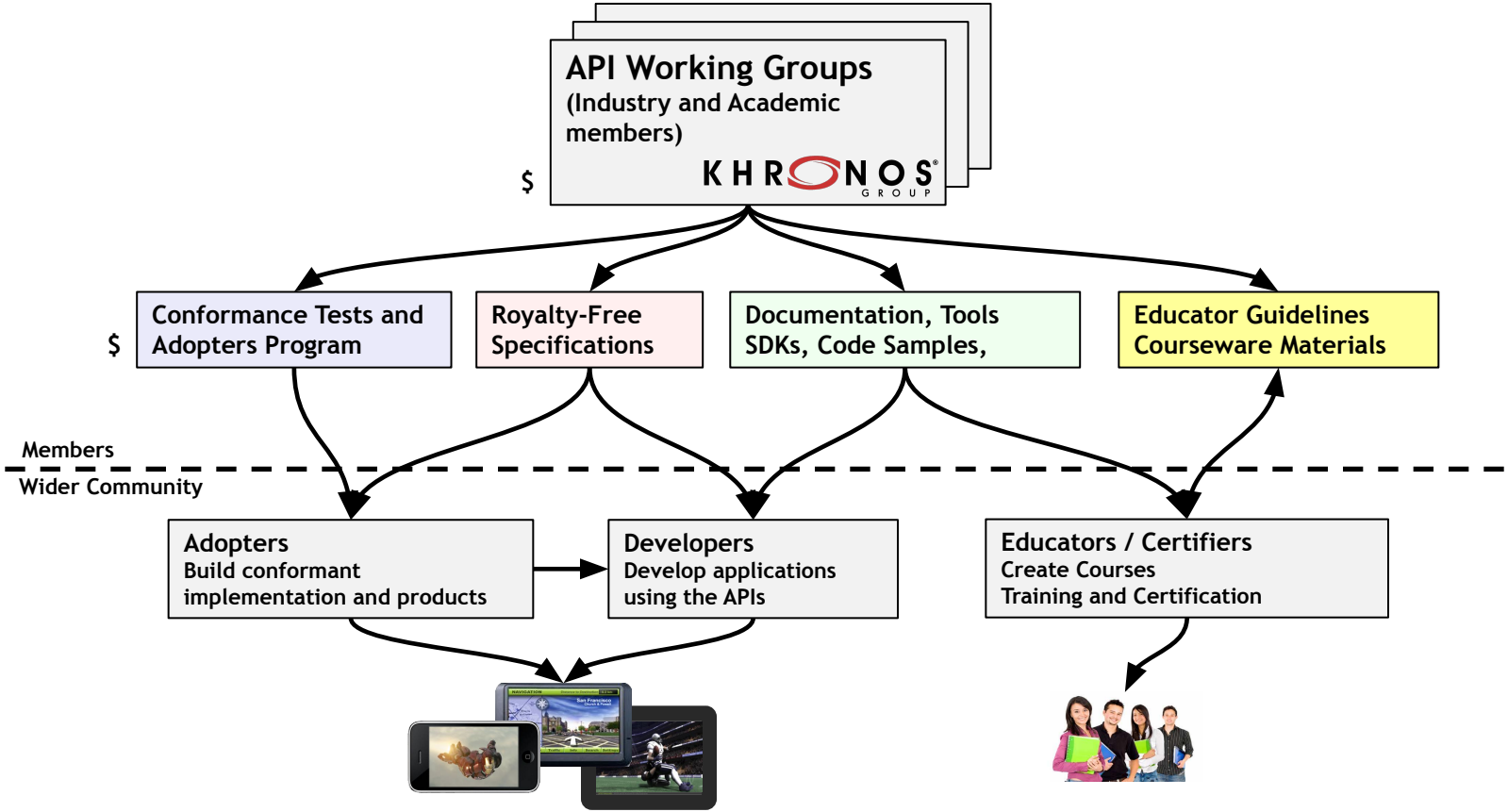
Dual Column Bullets

- Extensible modular architecture encourages many fine-grained layers - new layers can be added easily
- Khronos encouraging open community of tools e.g. shader debugging
- Valve, LunarG, Codeplay and others are already driving the development of open source Vulkan tools
- Customized interactive debugging and validation layers will be available together with first drivers

- Prototype Vulkan Debugger from Valve and LunarG
- LunarG.com/Vulkan



Typical Diagram Using Title Only Layout



KHRONOS GROUP

KHRONOS GROUP

CONNECTING SOFTWARE TO SILICON

KHRONOS GROUP