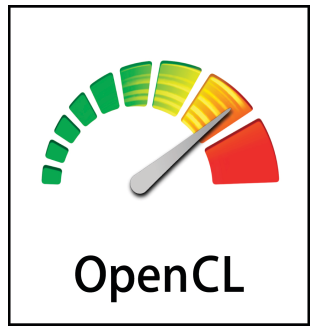


Vulkan, SPIR-V and OpenCL 2.1

GTC 2015

Neil Trevett | Khronos President

NVIDIA Vice President Mobile Ecosystem

Khronos Connects Software to Silicon

Open Consortium creating ROYALTY-FREE, OPEN STANDARD APIs for hardware acceleration

Defining the roadmap for low-level silicon interfaces needed on every platform

Graphics, compute, rich media, vision, sensor and camera processing

Rigorous specifications AND conformance tests for cross-vendor portability

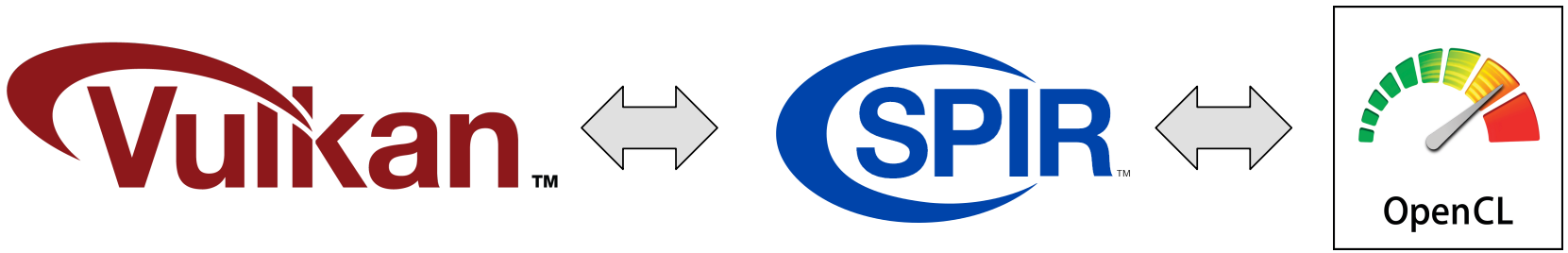
Acceleration APIs
BY the Industry
FOR the Industry

Well over a *BILLION* people use Khronos APIs *Every Day…*

http://accelerateyourworld.org/

Significant Khronos API Ecosystem Advances

- **Vulkan API – next generation graphics API**
  - Low overhead, high-efficiency graphics and compute on GPUs
  - Formerly discussed as Next Generation OpenGL Initiative
  - Reveal and demos at GDC – no formal specification yet
- **OpenCL 2.1 provisional specification released**
  - C++ Kernel language
  - Provisional specification released at GDC March 2015
- **SPIR-V – first intermediate language for graphics and parallel computation**
  - Will be used by both Vulkan AND OpenCL 2.1 core specifications
  - Provisional specification released at GDC March 2015

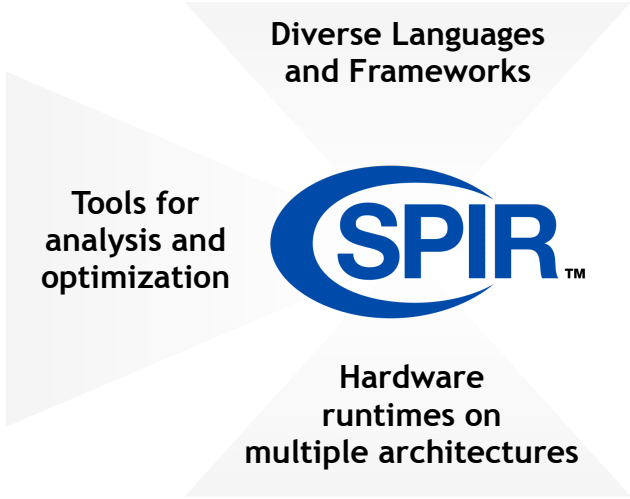


KHRONOS™ GROUP

SPIR-V Transforms the Language Ecosystem

- **Cross vendor intermediate representation**
  - Language front-ends can easily access multiple hardware run-times
  - Acceleration hardware can leverage multiple language front-ends
  - Encourages tools for program analysis and optimization in SPIR form
- **SPIR-V - first multi-API, intermediate language for parallel compute and graphics**
  - Native representation for Vulkan shader and OpenCL kernel source languages

SPIR-V is a significant convergence point in the language ecosystem for graphics and parallel computation



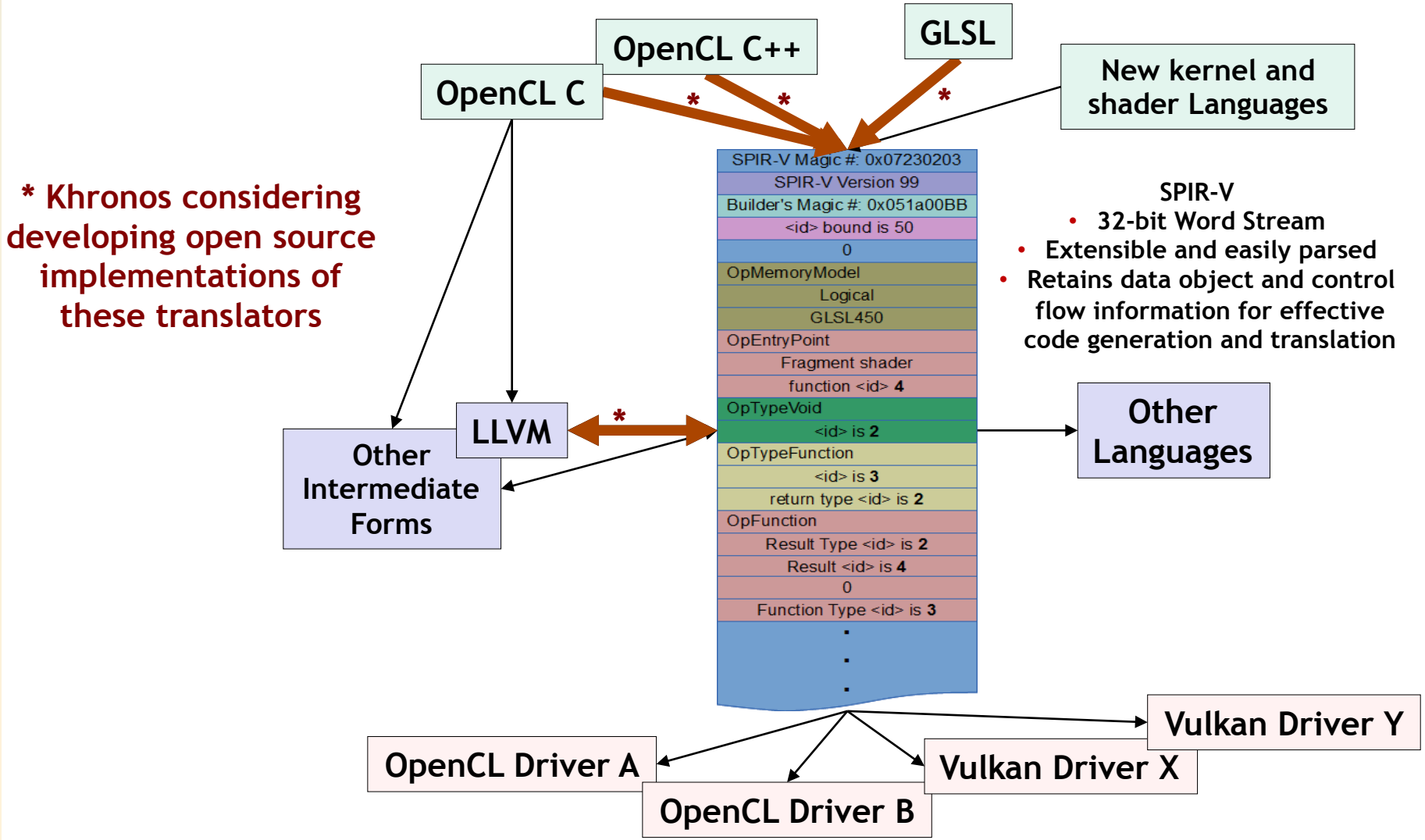
Standard
Portable
Intermediate
Representation

Evolution of SPIR Family

- **SPIR-V is first fully specified Khronos-defined SPIR standard**
  - Does not use LLVM to isolate from LLVM roadmap changes
  - Includes full flow control, graphics and parallel constructs beyond LLVM
  - Khronos will open source SPIR-V <-> LLVM conversion tools

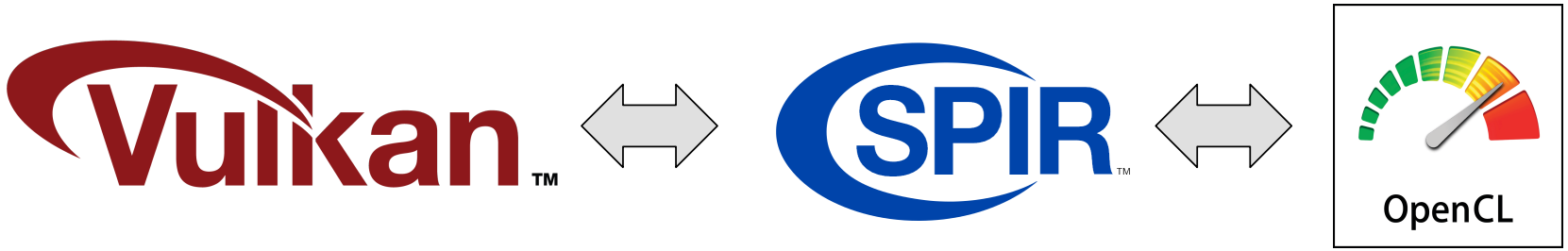
SPIR™	SPIR 1.2	SPIR 2.0	SPIR-V
LLVM Interaction	Uses LLVM 3.2	Uses LLVM 3.4	100% Khronos defined Round-trip lossless conversion
Compute Constructs	Metadata/Intrinsics	Metadata/Intrinsics	Native
Graphics Constructs	No	No	Native
Supported Language Feature Set	OpenCL C 1.2	OpenCL C 1.2 OpenCL C 2.0	OpenCL C 1.2 / 2.0 OpenCL C++ GLSL
OpenCL Ingestion	OpenCL 1.2 Extension	OpenCL 2.0 Extension	OpenCL 2.1 Core
Vulkan Ingestion	-	-	Vulkan Core

SPIR-V at the Center of Language Ecosystem



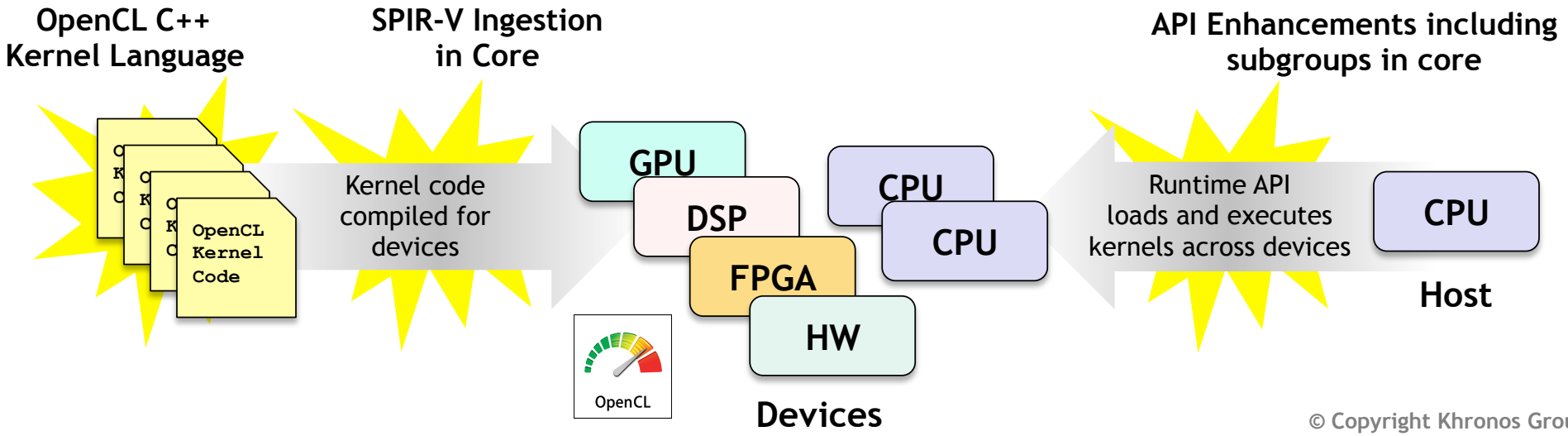
SPIR-V Advantages for Developers

- **Developers can use same front-end compiler across multiple platforms**
  - Eliminating major source of cross-vendor portability
- **Reduces runtime shader/kernel compilation time**
  - Driver only has to process SPIR-V not full source language
- **Don't have to ship shader/kernel source code**
  - Provides a measure of IP protection
- **Drivers are simpler and more reliable**
  - No need to include front-end compilers
- **SPIR-V Whitepaper**
  - https://www.khronos.org/registry/spir-v/papers/WhitePaper.pdf



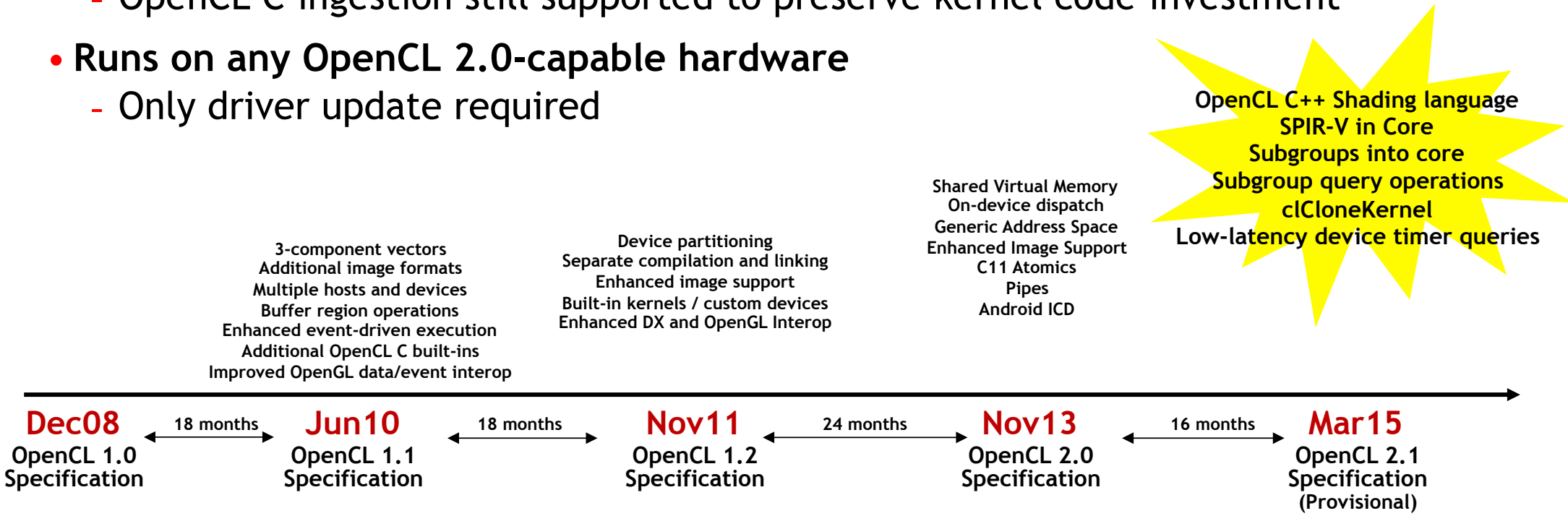
OpenCL - Portable Heterogeneous Computing

- **Heterogeneous parallel programming of diverse compute resources**
  - Targeting supercomputers -> embedded systems -> mobile devices
- **One code tree can be executed on CPUs, GPUs, DSPs, FPGA and hardware**
  - Dynamically interrogate system load and balance work across available processors
- **OpenCL = Two APIs and Kernel language**
  - C Platform Layer API to query, select and initialize compute devices
  - C Runtime API to build and execute kernels across multiple devices

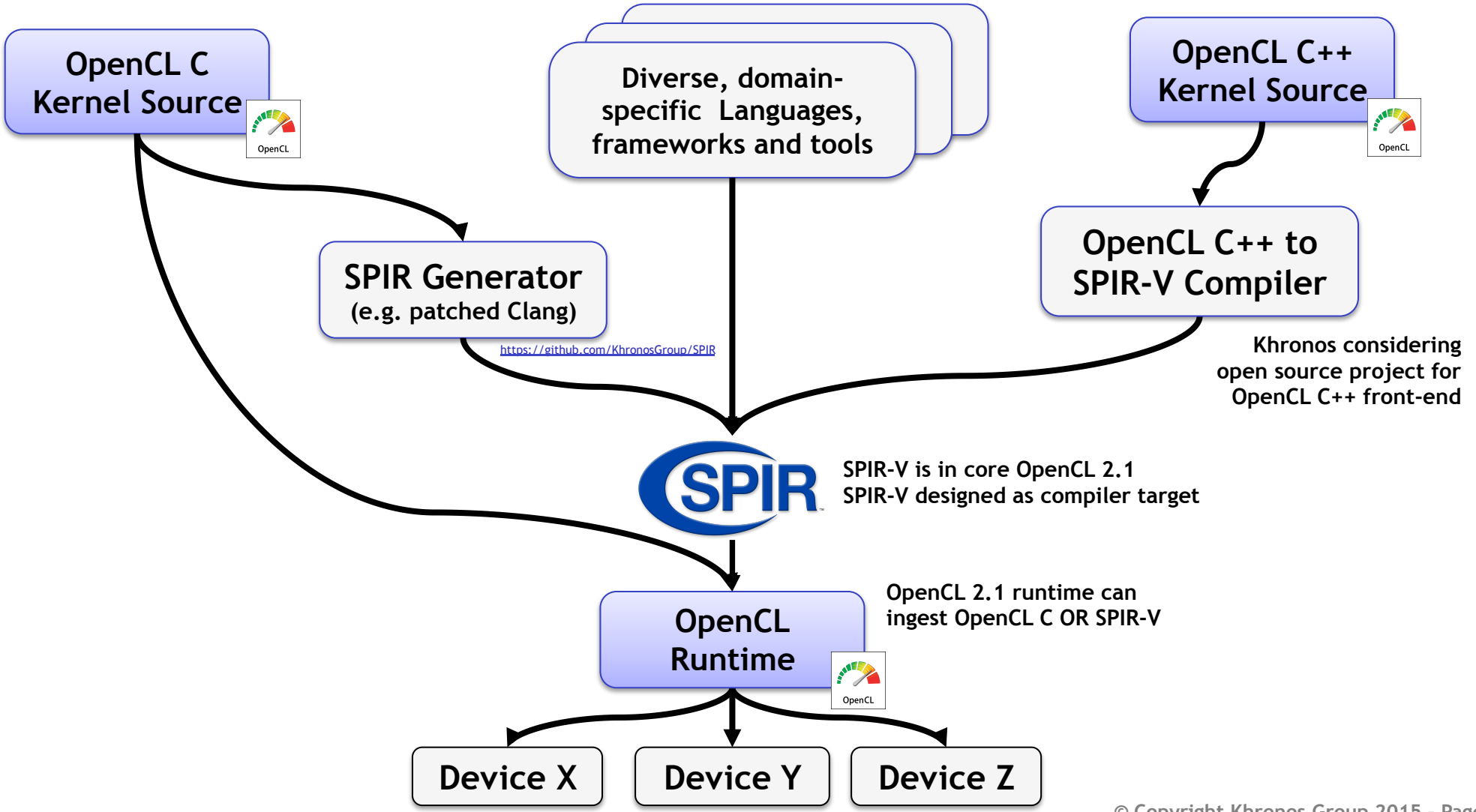


OpenCL 2.1 Provisional Released!

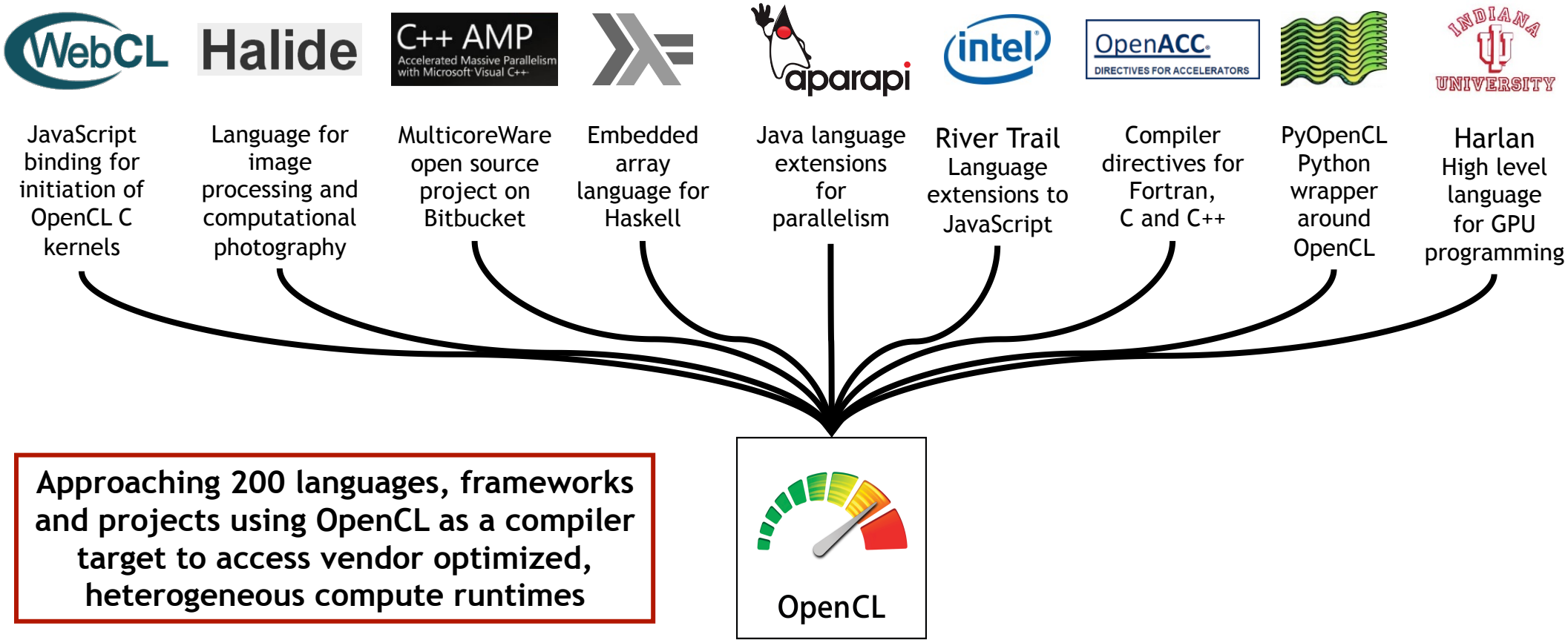
- **New OpenCL C++ kernel language based on a subset of C++14**
  - Significantly enhanced programmer productivity and code performance
- **Support for the new Khronos SPIR-V intermediate language in core**
  - SPIR-V used to ingest from C++ front-end – no C++ compiler in driver
  - OpenCL C ingestion still supported to preserve kernel code investment
- **Runs on any OpenCL 2.0-capable hardware**
  - Only driver update required



New OpenCL 2.1 Compiler Ecosystem



OpenCL as Parallel Language Backend



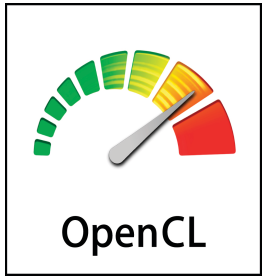
Approaching 200 languages, frameworks and projects using OpenCL as a compiler target to access vendor optimized, heterogeneous compute runtimes

KHRONOS GROUP

OpenCL C++

- **The OpenCL C++ kernel language is a static subset of C++14**
  - Frees developers from low-level coding details without sacrificing performance
- **C++14 features removed from OpenCL C++ for parallel programming**
  - Exceptions, Allocate/Release memory, Virtual functions and abstract classes Function pointers, Recursion and goto
- **Classes, lambda functions, templates, operator overloading etc..**
  - Fast and elegant sharable code - reusable device libraries and containers
  - Templates enable meta-programming for highly adaptive software
  - Lambdas used to implement nested/dynamic parallelism
- **C++11-based standard library optimized for data-parallel programming**
  - Atomics, meta-programming & type traits, math functions…
  - Plus new library features: Work-item & Work-group functions, Dynamic parallelism, Image & Pipe functions…

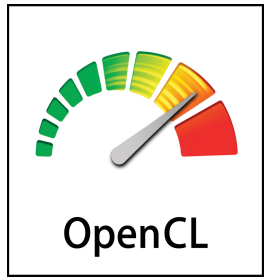
Highly adaptive parallel software that delivers tuned performance across diverse platforms



KHRONOS GROUP

OpenCL 2.1 API Enhancements

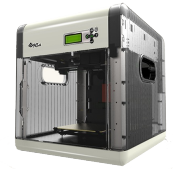
- **Subgroup functionality moved into core with additional subgroup query operations**
  - Expose hardware threads/warps/wavefronts and their cross-lane operations
  - Host queries for forward progress extension, and workgroup->subgroup mapping
- **clCloneKernel enables copying of kernel objects and state**
  - Safe implementation of copy constructors in wrapper classes
  - Used to pass kernel to second host thread, or for C++ wrappers for kernel objects
- **Low-latency device timer queries**
  - Support alignment of profiling data between device and host code
- **clCreateProgramWithIL**
  - Enables ingestion of SPIR-V code by the runtime
- **Priority and throttle hint extensions for queues**
  - Specify execution priority on a per-queue basis
- **Zero-size enqueue**
  - Zero-sized dispatches are valid from the host



The Need for Vulkan

Ground-up design of a modern open standard API for driving high-efficiency graphics and compute on GPUs used across diverse devices

Vulkan™



In the twenty two years since OpenGL was invented - the architecture of GPUs and platforms has changed radically

GPUs being used for graphics, compute and vision processing on a rapidly *increasing* diversity of platforms - *increasing* the need for cross-platform standards

Vulkan Explicit GPU Control

Complex drivers lead to driver overhead and cross vendor unpredictability

Error management is always active

Driver processes full shading language source

Separate APIs for desktop and mobile markets

Application	Application responsible for memory allocation and thread management to generate command buffers
Traditional graphics drivers include significant context, memory and error management	Direct GPU Control
GPU	GPU

Simpler drivers for low-overhead efficiency and cross vendor consistency

Layered architecture so validation and debug layers can be unloaded when not needed

Run-time only has to ingest SPIR-V intermediate language

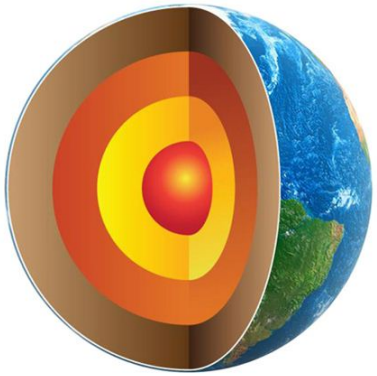
Unified API for mobile, desktop, console and embedded platforms

Vulkan delivers the maximized performance and cross platform portability needed by sophisticated engines, middleware and apps

KHRONOS GROUP

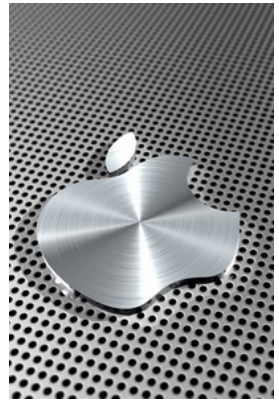
Cross Platform Challenge

- An explicit API that is also cross-platform needs careful design



One family of GPUs

One OS



One GPU on one OS

Vulkan™

All Modern Platforms and GPUs

A challenge that needs...

Participation of key players

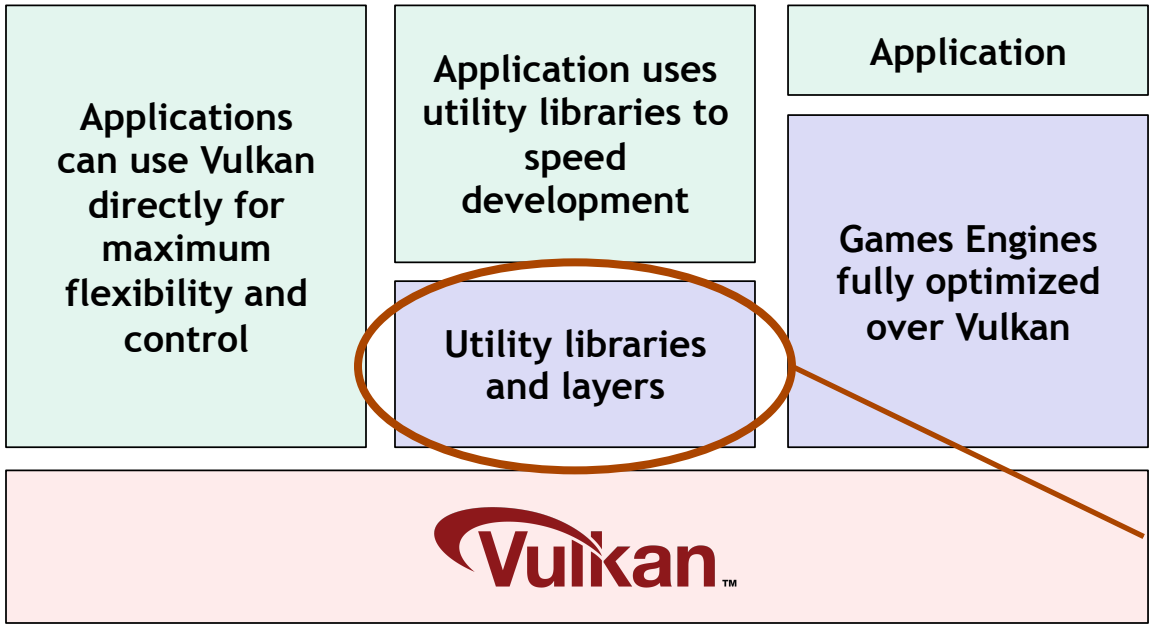
Proven IP Framework

Battle-tested cooperative model

The *drive* to not let the 3D industry fragment

KHRONOS GROUP

Vulkan Layered Ecosystem



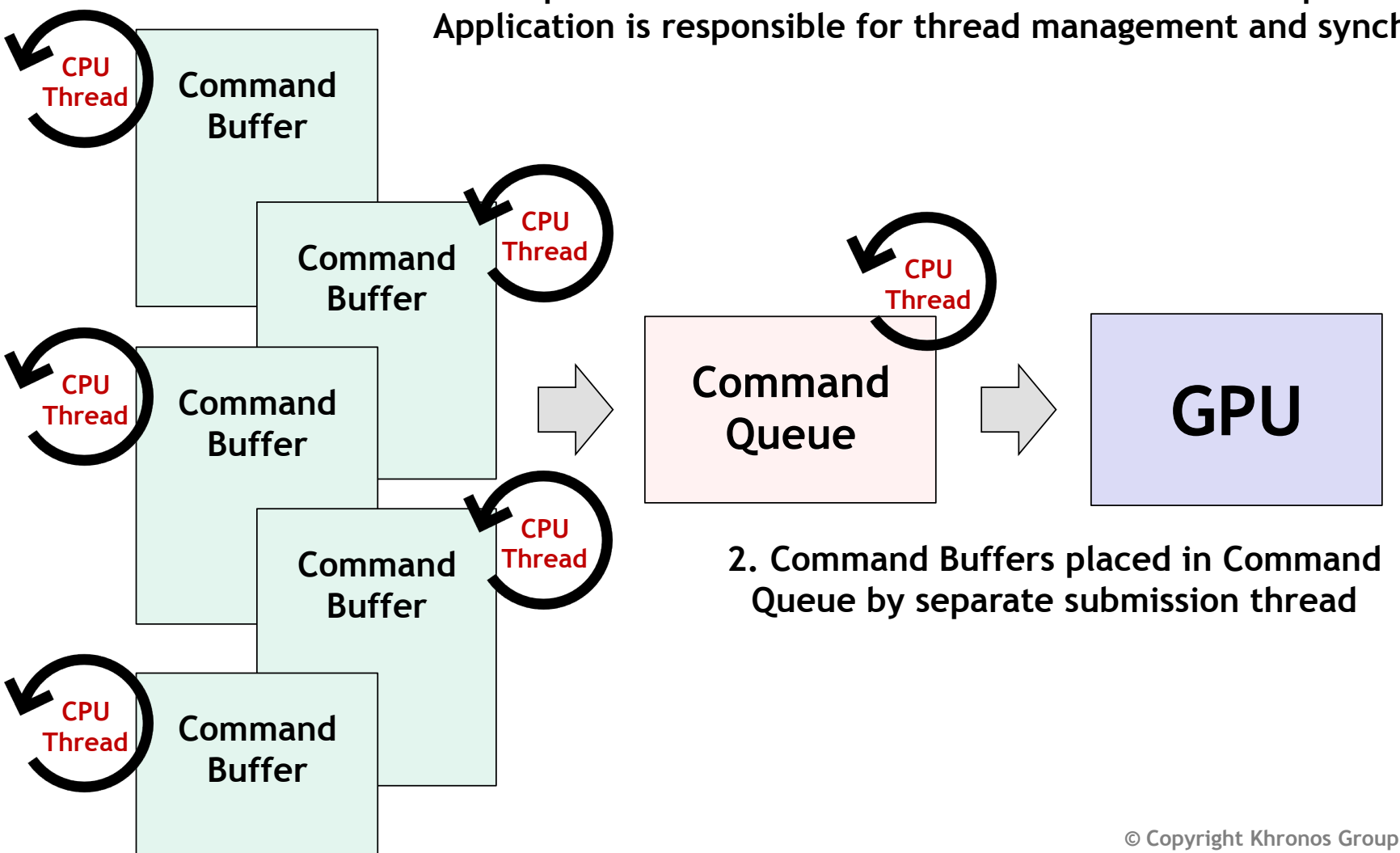
Developers can choose at which level to use the Vulkan Ecosystem

Rich Area for Innovation

- Many utilities and layers will be in open source
- Layers to ease transition from OpenGL
- Domain specific flexibility

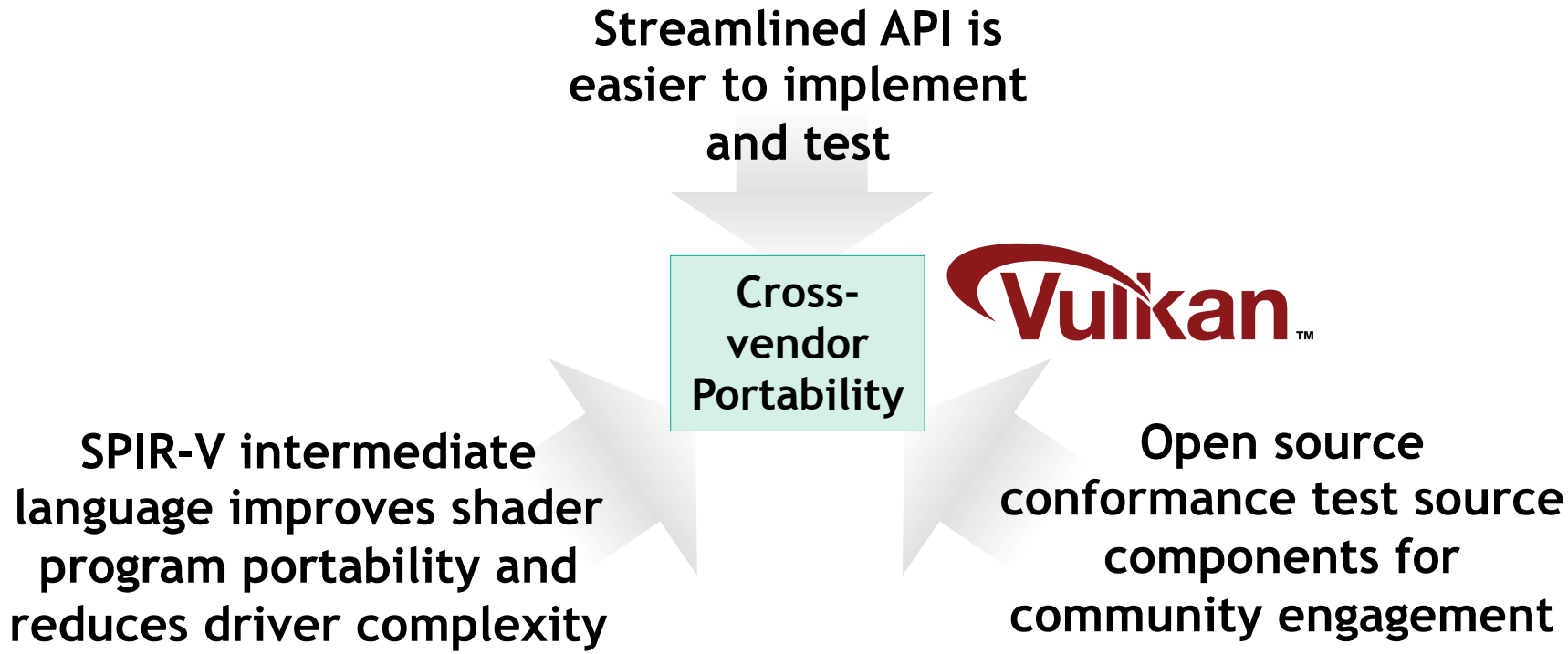
Vulkan Multi-threading Efficiency

1. Multiple threads can construct Command Buffers in parallel
Application is responsible for thread management and synch

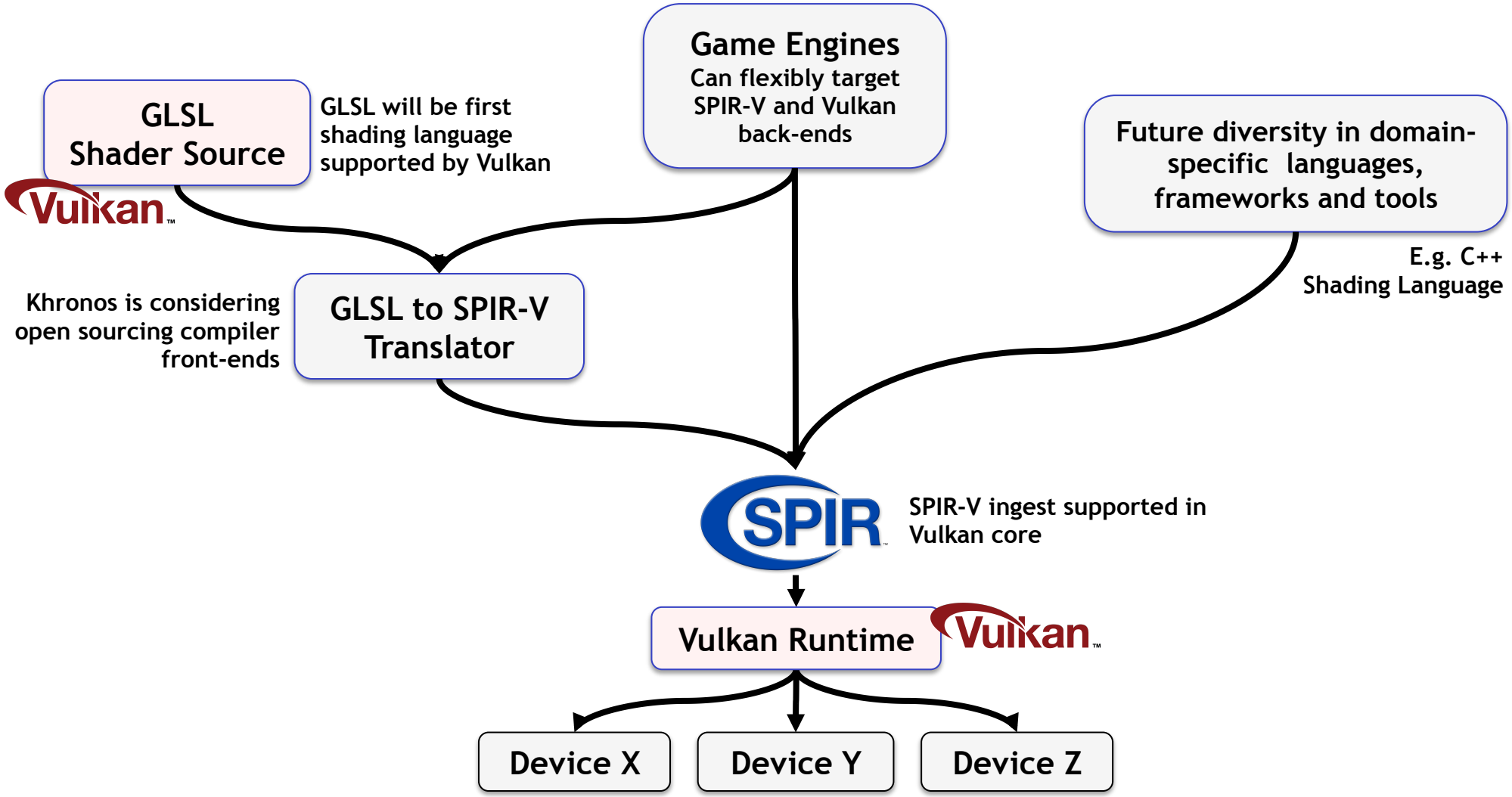


2. Command Buffers placed in Command Queue by separate submission thread

Vulkan - Enhancing Driver Reliability

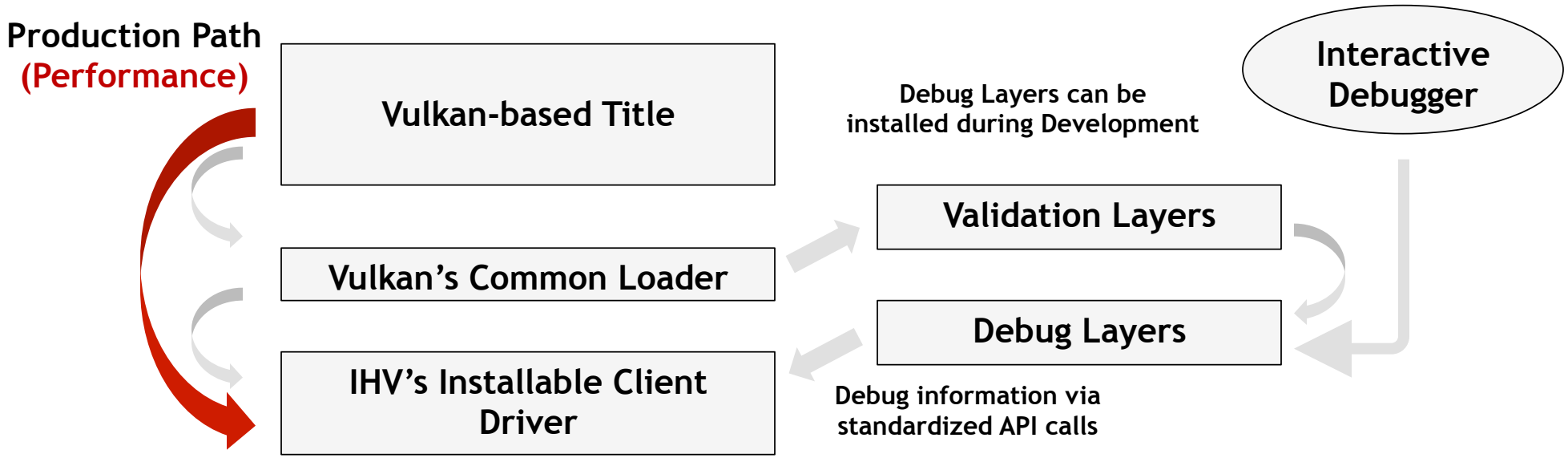


Vulkan Language Ecosystem



Vulkan Tools Architecture

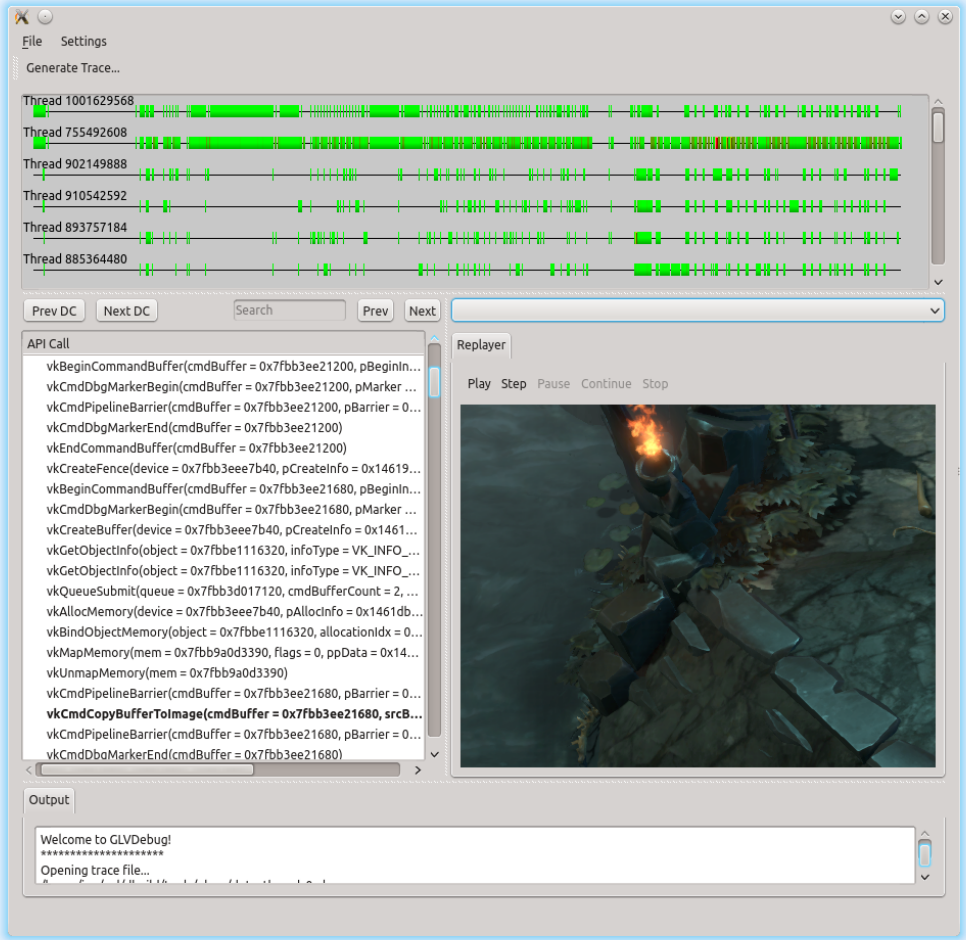
- **Layered design for cross-vendor tools innovation and flexibility**
  - IHVs plug into a common, extensible architecture for code validation, debugging and profiling during development without impacting production performance
- **Common Loader used to enable use of tools layers during debug**
  - Cross-vendor API calls provide debug data



Vulkan Tools Ecosystem

- 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



Ground-up Explicit API Redesign

OpenGL®	Vulkan™
Originally architected for graphics workstations with direct renderers and split memory	Matches architecture of modern platforms including mobile platforms with unified memory, tiled rendering
Driver does lots of work: state validation, dependency tracking, error checking. Limits and randomizes performance	Explicit API – the application has direct, predictable control over the operation of the GPU
Threading model doesn't enable generation of graphics commands in parallel to command execution	Multi-core friendly with multiple command buffers that can be created in parallel
Syntax evolved over twenty years – complex API choices can obscure optimal performance path	Removing legacy requirements simplifies API design, reduces specification size and enables clear usage guidance
Shader language compiler built into driver. Only GLSL supported. Have to ship shader source	SPIR-V as compiler target simplifies driver and enables front-end language flexibility and reliability
Despite conformance testing developers must often handle implementation variability between vendors	Simpler API, common language front-ends, more rigorous testing increase cross vendor functional/performance portability

KHRONOS™ GROUP

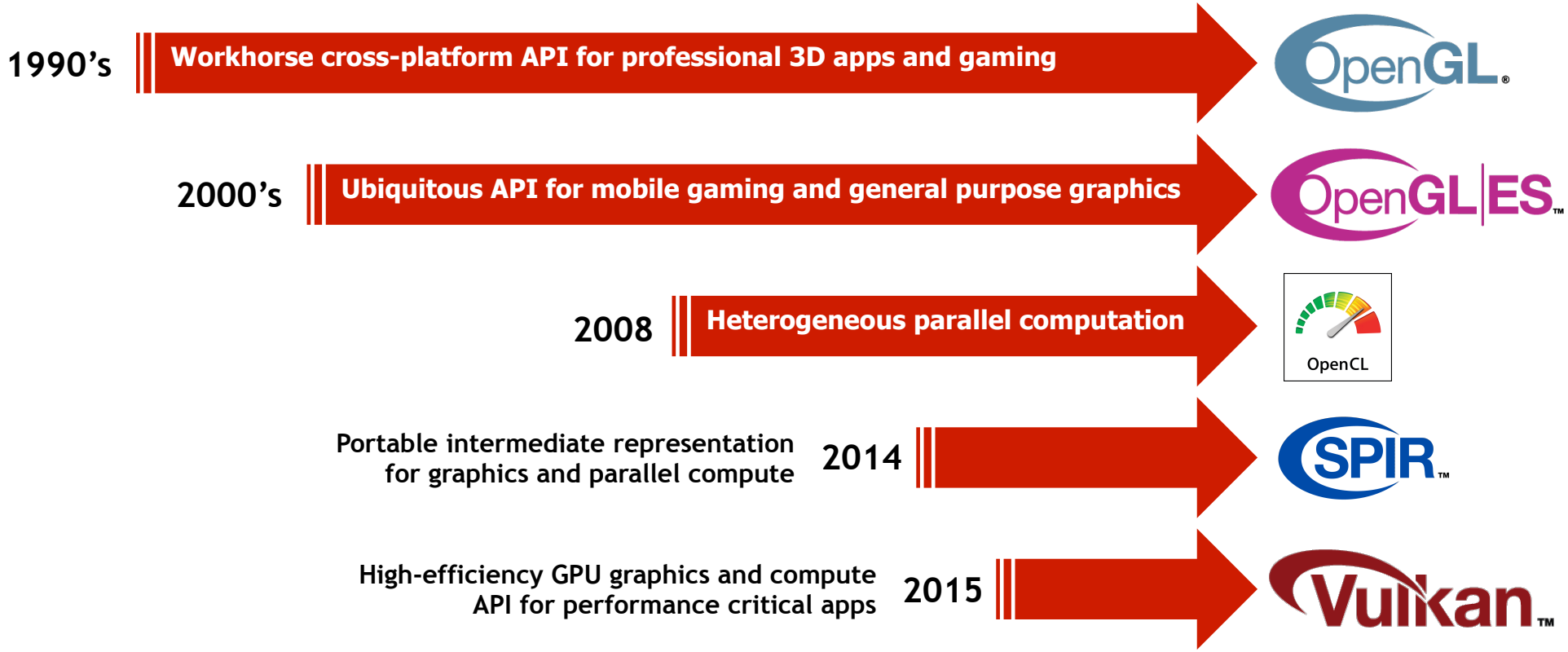
Vulkan Status

- **Rapid progress since project start in June 2014**
  - Significant proposals and IP contributions received from members
- **Participants come from all segments of the graphics industry**
  - Including an unprecedented level of participation from game engine ISVs
- **Initial specs and implementations expected this year**
  - Will work on any GPU hardware that supports OpenGL ES 3.1 and up
  - Can ship on any OS – including previous versions of Windows

Working Group Participants

Khronos Open Standards for Graphics and Compute

A comprehensive family of APIs to address the full spectrum of developer requirements



All APIs will be evolved and maintained to meet industry needs
Rich mix of open technologies for future innovation

KHRONOS GROUP

Call to Action

- More detailed information
  - https://www.khronos.org/spir
  - https://www.khronos.org/opencl/
  - https://www.khronos.org/vulkan
- Khronos seeking feedback on Vulkan, SPIR and OpenCL 2.1 Forums
  - https://www.khronos.org/spir_v_feedback_forum
  - https://www.khronos.org/opencl/opencl_feedback_forum
  - https://www.khronos.org/vulkan/vulkan_feedback_forum
- Any company or organization is welcome to join Khronos for a voice and a vote in any of these standards
  - www.khronos.org

KHRONOS™ GROUP

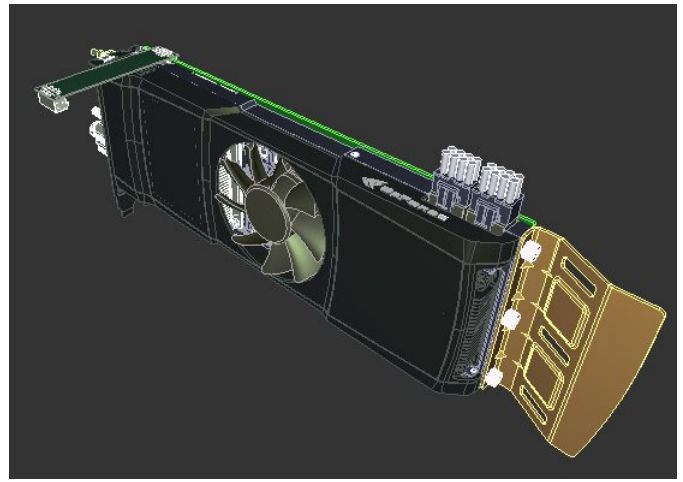
KHRONOS™ GROUP

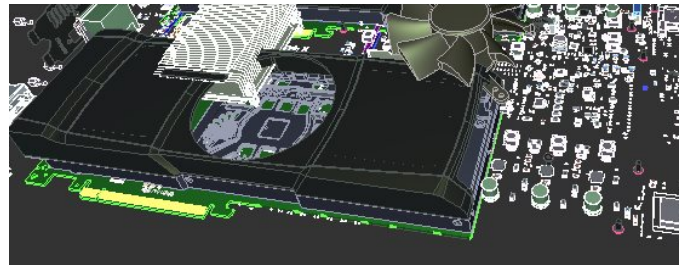
Vulkan Drawing

- Command Buffers greatly reduce CPU time when being re-used
- Replace thousands of traditional OpenGL calls with just one
- Keeps flexibility to still manipulate data afterwards (vertices, transforms, materials…)

```
glBindBufferBase (UBO, 0, uboView);
foreach (obj in scene) {
      glBindVertexBuffer (0, obj.geometry->vbo, 0, vtxSize);
      glBindBuffer (ELEMENT, obj.geometry->ibo);
      glBindBufferRange (UBO, 1, uboMatrices, obj.mtxOffset, mtxSize);
      glUseProgram ( progSolid );
      foreach ( batch in obj.batches) {
         glBindBufferRange (UBO, 2, uboMaterial, batch.mtlOffset, mtlSize);
         glMultiDrawElements (TRIANGLES ...);
      }
      glUseProgram (progEdges );
      ...
}
```

Thousands of state changes and 10s of thousands drawcalls





```
vkQueueSubmit (..., 1 , &cmdbuffer, ...);
```