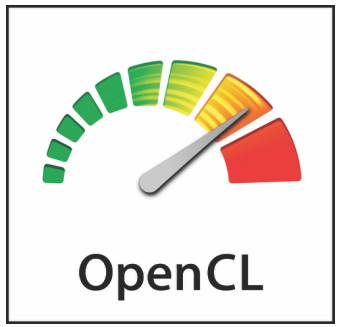


KHRONOS™ GROUP

Embedded Outreach

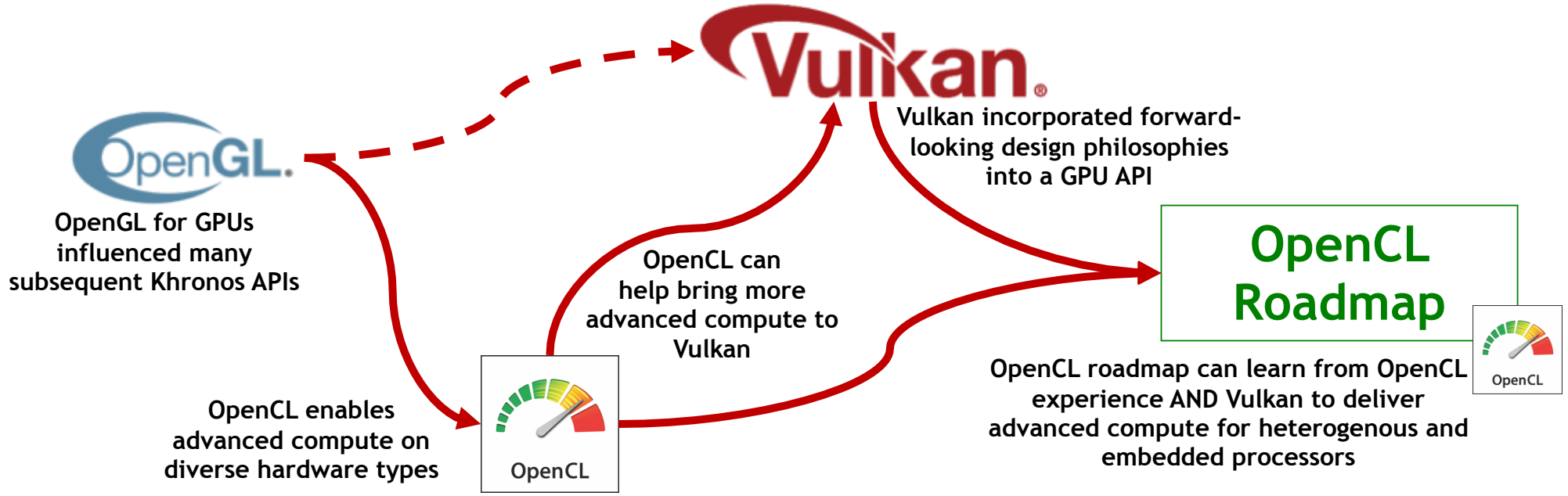
Focusing OpenCL for the Embedded Industry

January 2018

Neil Trevett | Khronos President
NVIDIA | VP Developer Ecosystem
ntrevett@nvidia.com | @neilt3d
www.khronos.org

Agenda for this Session

- Introduction to Khronos and the need for open standards
- Update on OpenCL (and SPIR-V) – the standards for heterogenous computing
- Update on Vulkan - a new explicit-style API
- Lessons from Vulkan and OpenCL that can guide the OpenCL roadmap
- Directions for OpenCL roadmap – and increasing relevance to the embedded industry



What is An Open Standard?

An INTEROPERABILITY STANDARD
enables two interoperating entities to
COMMUNICATE

Software <-> Hardware
Sensor <-> Processor
Device <-> Host

Bad Standards

- Overprescribes implementation details
- Forces everyone to implement a lowest common denominator
- Stifles innovation

-> Commoditization

Good Standards

- Prescribes only interoperability
- Enables implementation diversity
- Encourages innovation

-> Differentiation

A truly OPEN standard

Is not controlled by a single company - but by the whole industry
Is freely available to use by any company without royalty payments

Why Do We Need Open Standards?

An advanced product such as mobile uses *100s* of standards

Wireless: LTE, WiFi, Bluetooth

Physical: HDMI, USB, CSI

Software APIs: HTML5, OpenGL ES

Standards drive mobile market growth by expanding device capabilities

Standard Defining Organizations

Wi Fi ALLIANCE

3GPP A GLOBAL INITIATIVE

mipi mobile industry processor interface

SUPERSPEED USB CERTIFIED

W3C

KHRONOS GROUP

Standards Grow Markets…

By reducing consumer confusion and increasing usability

… Reduce Costs …

By sharing development between many companies and driving volume

… Accelerate Time to Market

With well-proven testing and interoperability

Khronos Mission

E.g. OpenGL ES provides 3D on almost *every* smartphone - over 4 Billion units shipped

PROMOTER MEMBERS

KHRONOS GROUP

Over 100 members worldwide
Any company is welcome to join

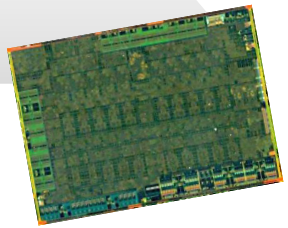
AMD, Apple, arm, EPIC GAMES, Google, HUAWEI, Imagination, intel, NOKIA, QUALCOMM, SAMSUNG, SONY, VALVE, NVIDIA, VeriSilicon

3D Incorporated, Adobe, AIMOTIVE, Amazon.com, antilatency, AXELL CORPORATION, AXIS COMMUNICATIONS, BASEMARK, BINOMIAL, BLIZZARD ENTERTAINMENT, BROADCOM, BRENWILL, cadence, CEVA, CAICT, codeplay, Continental, Coordinate, COREAVI, DASSAULT SYSTEMES, DELL, CATAPULT Digital, DMP, ETRI, Fraunhofer, FUTUREMARK CORPORATION, Gaijin, HARMAN, HITACHI Inspire the Next, htc, hugherk, HYPEREAL, igalia, IKEA Communications AB, Imperial College London, ITRI, KDAB, KNU, LG, Linaro, logitech, Los Alamos NATIONAL LABORATORY, LUNARG, matrox, MAXON, MEDIATEK, Mentor Graphics, Microsoft, MIT Lincoln Laboratory, MITSUBISHI ELECTRIC, mozilla, mobica, NEC, NIHON UNIVERSITY, Nintendo, NXP, oculus, ON Semiconductor, OSU Oregon State, owlii, OXIDE, Panasonic, Pico, PIXAR, PLUTO VR, Qt The Qt Company, RAZER, RENESAS, Rockwell Collins, 서울대학교 SEOUL NATIONAL UNIVERSITY, sensics, Silicon Studio, SIRU INNOVATIONS, socionext, SPREADTRUM, STREAM High Performance Computing, SYNOPSYS, TAKUMI, TAMPERE UNIVERSITY OF TECHNOLOGY, TU WIEN TECHNISCHE UNIVERSITÄT WIEN, Tencent, Texas Instruments, thinci, ThinkSilicon, tobii, TOSHIBA, umbra, Unity, UCL, University of BRISTOL, UNIVERSITY OF TORONTO, University of Windsor, UX3D, 兆芯, Visteon, vmware, vrg, XILINX, zSpace

Software

OpenGL ES, Vulkan, glTF, SPIR, OpenGL, OpenCL

Silicon



Khronos is an International Industry Consortium of over 100 companies creating royalty-free, open standard APIs to enable software to access hardware acceleration for 3D graphics, Virtual and Augmented Reality, Parallel Computing, Neural Networks and Vision Processing

Why Mobile and Embedded Need API Standards

Without API Standards

Platform Fragmentation

Everything runs on CPU

With API Standards

Application Portability

Silicon Acceleration

Standard Acceleration APIs provide PERFORMANCE, POWER AND PORTABILITY

Khronos Open APIs for Today's Industry

Download 3D augmentation object and scene data

glTF™

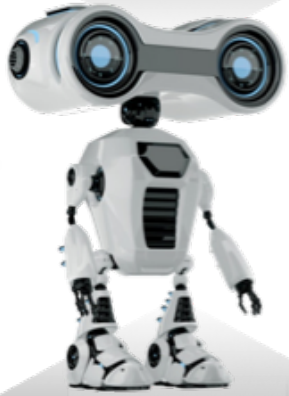
SYCL™

OpenVX™

NNEF™

OpenCL

Vision and sensor processing - including neural networks



Generate Low Latency 3D Content and Simulations

Vulkan®

OpenGL®

OpenGL ES

WebGL™

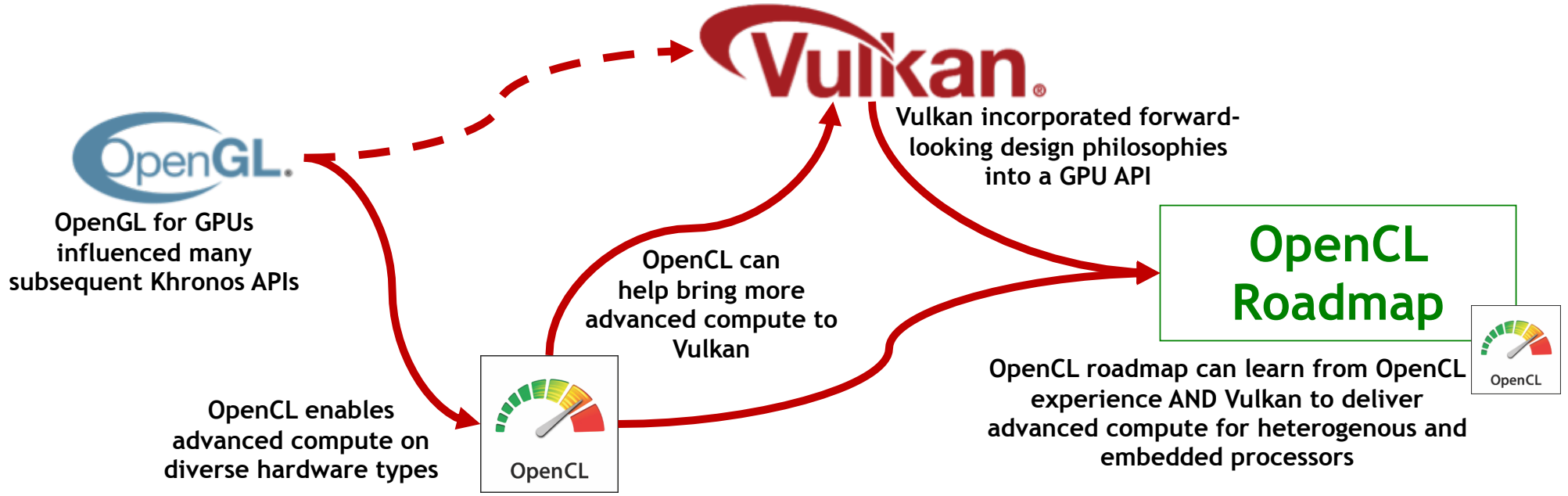
OpenXR™

Interact with sensor, haptic and VR/AR display devices

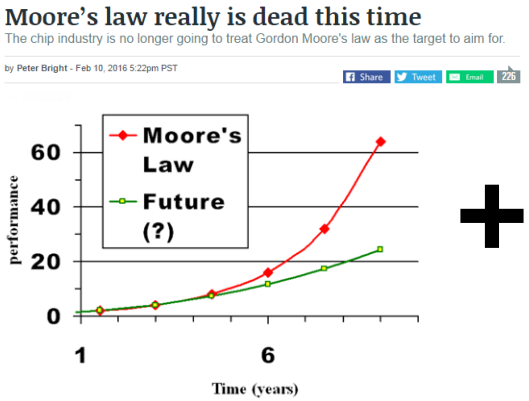
KHRONOS GROUP

Agenda for this Session

- Introduction to Khronos and the need for open standards
- Update on OpenCL (and SPIR-V) – the standards for heterogenous computing
- Update on Vulkan - a new explicit-style API
- Lessons from Vulkan and OpenCL that can guide the OpenCL roadmap
- Directions for OpenCL roadmap – and increasing relevance to the embedded industry



Need for Heterogeneous Parallelism



+

Flynn's Taxonomy (1966)

SISD	SIMD
Single Instruction stream Single Data stream	Single Instruction stream Multiple Data stream
MISD	MIMD
Multiple Instruction stream Single Data stream	Multiple Instruction stream Multiple Data stream

=

Languages / Directives

OpenACC
THE C++ PROGRAMMING LANGUAGE
Charm++
OpenHMPP
OpenMP
C++ AMP
Intel® Cilk™ Plus
UPC
SYCL™

Libraries

ViennaCL
Thrust
global
boost C++ LIBRARIES
Microsoft Parallel Patterns Library
intel TBB
HPX V0.9.6 High Performance ParalleX

Explicit Kernels

NVIDIA CUDA
OpenCL
DirectX 12
Vulkan
SPIR

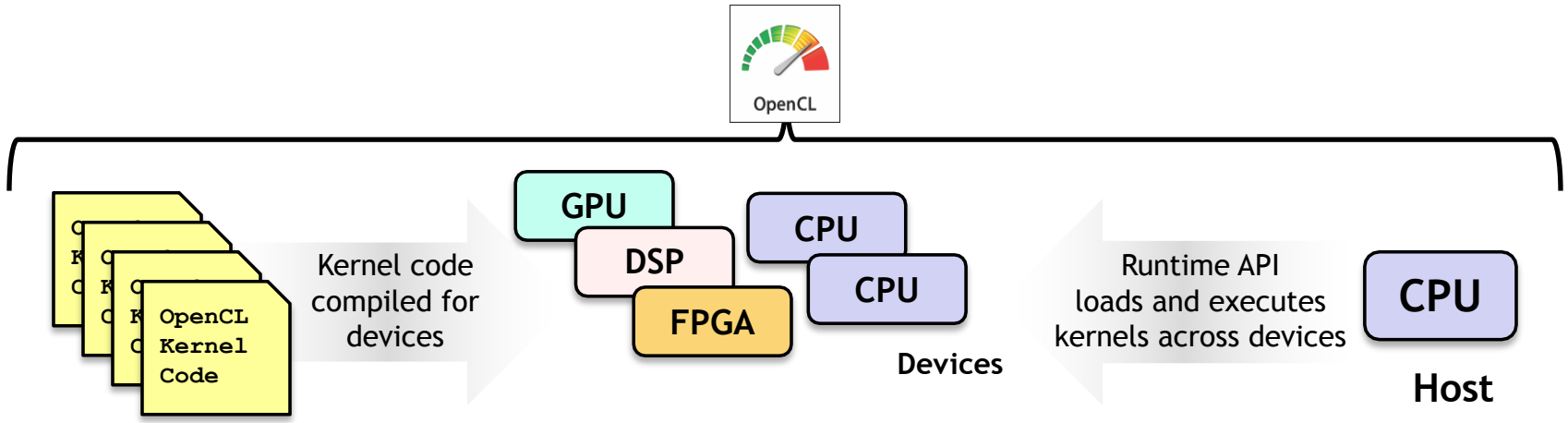
Threads and Messages

pthreads
MPI
OPEN MPI

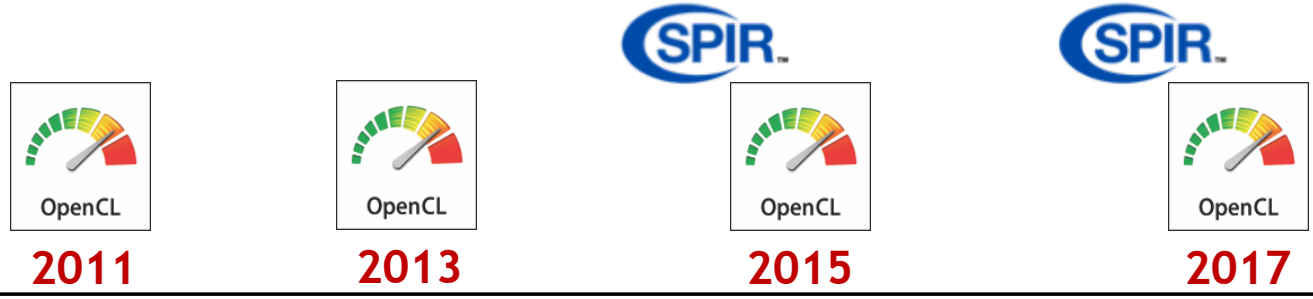
"The purpose of abstraction is not to be vague, but to create a new semantic level in which one can be absolutely precise" - Edsger Dijkstra

OpenCL - Low-level Parallel Programing

- **Low-level programming of heterogeneous parallel compute resources**
  - One code tree can be executed on CPUs, GPUs, DSPs and FPGA …
- **OpenCL C or C++ language to write kernel programs to execute on any compute device**
  - Platform Layer API - to query, select and initialize compute devices
  - Runtime API - to build and execute kernels programs on multiple devices
- **The programmer gets to control:**
  - What programs execute on what device
  - Where data is stored in various speed and size memories in the system
  - When programs are run, and what operations are dependent on earlier operations



OpenCL 2.2 Released in May 2017



2011

OpenCL 1.2

Becomes industry baseline for heterogeneous parallel computing

2013

OpenCL 2.0

Enables new class of hardware
SVM
Generic Addresses
On-device dispatch

2015

OpenCL 2.1
SPIR-V 1.0

SPIR-V in Core
Kernel Language
Flexibility

2017

OpenCL 2.2
SPIR-V 1.2

OpenCL C++ Kernel Language
Static subset of C++14
Templates and Lambdas

SPIR-V 1.2
OpenCL C++ support

Pipes
Efficient device-scope communication between kernels

Code Generation Optimizations
- Specialization constants at SPIR-V compilation time
- Constructors and destructors of program scope global objects
- User callbacks can be set at program release time


OpenCL 2.2 Reference Card Page 1

OpenCL (Open Computing Language) is a multi-vendor open standard for general-purpose parallel programming of heterogeneous systems that include CPUs, GPUs, and other processors. OpenCL provides a uniform programming environment for software developers to write efficient, portable code for high-performance compute servers, desktop computer systems, and handheld devices.

Specification documents and online reference are available at www.khronos.org/opencl.

KHRONOS GROUP

[n.n.n] and purple text: sections and text in the OpenCL API 2.2 Spec.
[n.n.n] and green text: sections and text in the OpenCL C++ 2.2 Spec.
[n.n.n] and brown text: sections and text in the OpenCL C 2.0 Spec.
[n.n.n] and blue text: sections and text in the OpenCL Extension 2.2 Spec.

OpenCL API Reference

Section and table references are to the OpenCL API 2.2 specification.

The OpenCL Platform Layer

The OpenCL platform layer implements platform-specific features that allow applications to query OpenCL devices, device configuration information, and to create OpenCL contexts using one or more devices. Items in blue apply only when the appropriate extension is enabled (see Extensions on page 21 of this reference guide).

Querying platform info & devices [4.1-2] [B.16.9]

cl_int clGetPlatformIDs (cl_uint num_entries, cl_platform_id *platforms, cl_uint *num_platforms)

cl_int clGetPlatformInfo (cl_platform_id platform, cl_platform_info param_name, size_t param_value_size, void *param_value, size_t *param_value_size_ret)

param_name: CL_PLATFORM_{PROFILE, VERSION}, CL_PLATFORM{_NUMERIC_VERSION, EXTENSIONS}, CL_PLATFORM_HOST_TIMER_RESOLUTION, CL_PLATFORM{ICD_SUFFIX_KHR} [Table 4.1]

cl_int clGetDeviceIDs (cl_platform_id platform, cl_device_type device_type, cl_uint num_entries, cl_device_id *devices, cl_uint *num_devices)

device_type [Table 4.2]: CL_DEVICE_TYPE_{ACCELERATOR, ALL, CPU, CUSTOM, DEFAULT, GPU}

cl_int clGetDeviceInfo (cl_device_id device, cl_device_info param_name, size_t param_value_size, void *param_value, size_t *param_value_size_ret)

param_name: [Table 4.3]: CL_DEVICE_{ADDRESS_BITS, CL_DEVICE_AVAILABLE, BUILT_IN_KERNELS, COMPILER_AVAILABLE, DOUBLE_FP_CONFIG, ENDIAN_LITTLE, ERROR_CORRECTION_SUPPORT, EXECUTION_CAPABILITIES, GLOBAL_MEM_CACHE_{SIZE, TYPE}, GLOBAL_MEM_CACHELINE_SIZE, SIZE}, GLOBAL_VARIABLE_PREFERRED_TOTAL_SIZE, HALF_FP_CONFIG, HOST_UNIFIED_MEMORY, IMAGE_SUPPORT, IMAGE2D_MAX_{WIDTH, HEIGHT}, IMAGE3D_MAX_{WIDTH, HEIGHT, DEPTH}, IMAGE_BASE_ADDRESS_ALIGNMENT, IMAGE_PITCH_ALIGNMENT, LINKER_AVAILABLE, LOCAL_MEM_{TYPE, SIZE}, MAX_{CLOCK_FREQUENCY, PIPE_ARGS, COMPUTE_UNITS, SAMPLERS}, MAX_CONSTANT_{ARGS, BUFFER_SIZE}, MAX_MEM_ALLOC_SIZE, MAX_NUM_SUB_GROUPS, MAX_ON_DEVICE_{QUEUES, EVENTS}, MAX_{READ, WRITE}_IMAGE_ARGS, MAX_READ_WRITE_IMAGE_ARGS, MAX_SUB_GROUPS, MAX_WORK_GROUP_SIZE, MAX_WORK_ITEM_{DIMENSIONS, SIZES}, MEM_BASE_ADDR_ALIGN, NAME, NATIVE_VECTOR_WIDTH_{CHAR, INT, DOUBLE, HALF, LONG, SHORT, FLOAT}, CL_DEVICE_NATIVE_VECTOR_WIDTH_FLOAT, CL_DEVICE_DEVICE_ENQUEUE_CAPABILITIES, CL_DEVICE_PARTITION_{AFFINITY_DOMAIN, MAX_SUB_DEVICES}, CL_DEVICE_PARTITION_{PROPERTIES, TYPES}, CL_DEVICE_PIPE_MAX_{ACTIVE_RESERVATIONS, PACKET_SIZE}, CL_DEVICE_PLATFORM, PRINTF_BUFFER_SIZE}, CL_DEVICE_PREFERRED_{V_ATOMIC_ALIGNMENT, VECTOR_WIDTH_{CHAR, INT, DOUBLE, HALF, LONG, SHORT, FLOAT}}, CL_DEVICE_PREFERRED_INTEROP_USER_SYNC, CL_DEVICE_PROFILE, CL_DEVICE_PROFILING_TIMER_RESOLUTION, CL_DEVICE_SPIR_VERSIONS, CL_DEVICE_SUB_GROUP_INDEPENDENT_FORWARD_PROGRESS, CL_DEVICE_QUEUE_ON_{DEVICE_HOST}_PROPERTIES, CL_DEVICE_QUEUE_ON_DEVICE_MAX_SIZE, CL_DEVICE_QUEUE_ON_DEVICE_PREFERRED_SIZE, CL_DEVICE_REFERENCE_COUNT_VENDOR_{ID}, CL_DEVICE_SVM_CAPABILITIES, CL_DEVICE_THREAD_LIMIT_{AMD, NV}, CL_DEVICE_{TYPE, VENDOR}_ID, CL_DEVICE_VERSION, CL_DEVICE_OPENCL_C_VERSION, CL_DEVICE_MAX_NAMED_BARRIER_COUNT_KHR

cl_int clGetDeviceAndHostTimer (cl_device_id device, cl_ulong *device_timestamp, cl_ulong *host_timestamp)

cl_int clGetHostTimer (cl_device_id device, cl_ulong *host_timestamp)

Partitioning a device [4.3]

cl_int clCreateSubDevices (cl_device_id in_device, const cl_device_partition_property *properties, cl_uint num_devices, cl_device_id *out_devices, cl_uint *num_devices_ret)

properties: [Table 4.4] CL_DEVICE_PARTITION_EQUALLY, CL_DEVICE_PARTITION_BY_{COUNTS, AFFINITY_DOMAIN}

The OpenCL Runtime

API calls that manage OpenCL objects such as command-queues, memory objects, program objects, kernel objects for __kernel functions in a program and calls that allow you to enqueue commands to a command-queue such as executing a kernel, reading, or writing a memory object.

Command queues [5.1]

cl_command_queue clCreateCommandQueueWithProperties (cl_context context, cl_device_id device, const cl_command_queue_properties *properties, cl_int *errcode_ret)

*properties: Points to a zero-terminated list of properties and their values: [Table 5.1] CL_QUEUE_SIZE, CL_QUEUE_PROPERTIES (bitfield which may be set to an OR of CL_QUEUE_ * where * may be: OUT_OF_ORDER_EXEC_MODE_ENABLE, PROFILING_ENABLE, ON_DEVICE, ON_DEVICE_DEFAULT, THROTTLED_{AMD, KHR}, PRIORITY_{AMD, KHR} (requires cl_khr_priority_hints extension)), CL_QUEUE_PRIORITY_KHR (bitfield which may be one of CL_QUEUE_PRIORITY_HIGH_KHR, CL_QUEUE_PRIORITY_MED_KHR, CL_QUEUE_PRIORITY_LOW_KHR (requires the cl_khr_priority_hints extension))

cl_int clRetainDevice (cl_device_id device)
cl_int clReleaseDevice (cl_device_id device)

Contexts [4.4]

cl_context clCreateContext (const cl_context_properties *properties, cl_uint num_devices, const cl_device_id *devices, void (CL_CALLBACK *pfn_notify) (const char *errinfo, const void *private_info, size_t cb, void *user_data), void *user_data, cl_int *errcode_ret)

properties: [Table 4.5] NULL or CL_CONTEXT_PLATFORM, CL_CONTEXT_INTEROP_USER_SYNC, CL_CONTEXT_{D3D10, D3D11}_DEVICE_KHR, CL_CONTEXT_ADAPTER_{D3D9, D3D9EX, DXVA}_KHR, CL_CONTEXT_ADAPTER_D3D9{,EX,DXVA}_DEVICE_KHR, CL_CONTEXT_{D3D11}_DEVICE_KHR, CL_CONTEXT_VA_API_DISPLAY_INTEL, CL_{EGL_CONTEXT, EGL_DISPLAY, GLX_DISPLAY, CGL_SHAREGROUP, WGL_HDC}_KHR

cl_context clCreateContextFromType (const cl_context_properties *properties, cl_device_type device_type, void (CL_CALLBACK *pfn_notify) (const char *errinfo, const void *private_info, size_t cb, void *user_data), void *user_data, cl_int *errcode_ret)

properties: See clCreateContext
device_type: See clGetDeviceIDs

cl_int clRetainContext (cl_context context)
cl_int clReleaseContext (cl_context context)
cl_int clGetContextInfo (cl_context context, cl_context_info param_name, size_t param_value_size, void *param_value, size_t *param_value_size_ret)

param_name: CL_CONTEXT_X where X may be REFERENCE_COUNT, DEVICES, NUM_DEVICES, PROPERTIES, D3D10_PREFER_SHARED_RESOURCES_KHR, D3D11_PREFER_SHARED_RESOURCES_KHR [Table 4.6]

cl_int clTerminateContextKHR (cl_context context)

Get CL extension function pointers [B.3]

void* clGetExtensionFunctionAddressForPlatform (cl_platform_id platform, const char *funcname)

cl_int clSetDefaultDeviceCommandQueue (cl_context context, cl_device_id device, cl_command_queue command_queue)

cl_int clRetainCommandQueue (cl_command_queue command_queue)

cl_int clReleaseCommandQueue (cl_command_queue command_queue)

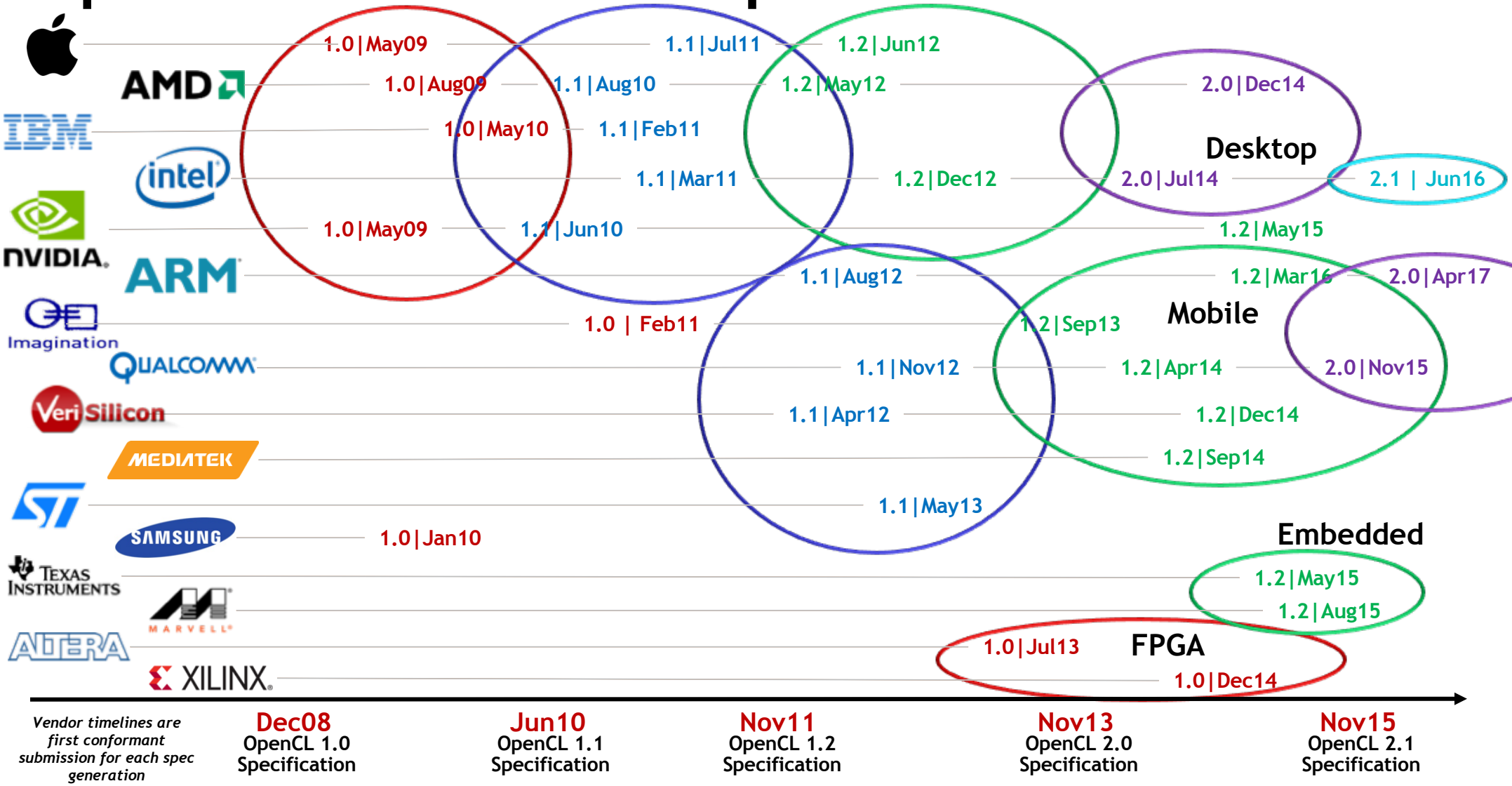
cl_int clGetCommandQueueInfo (cl_command_queue command_queue, cl_command_queue_info param_name, size_t param_value_size, void *param_value, size_t *param_value_size_ret)

param_name: [Table 5.2] CL_QUEUE_{CONTEXT, DEVICE, DEVICE_DEFAULT, CL_QUEUE_SIZE, REFERENCE_COUNT, CL_QUEUE_PROPERTIES}

©2017 Khronos Group - Rev. 0517 www.khronos.org/opencl

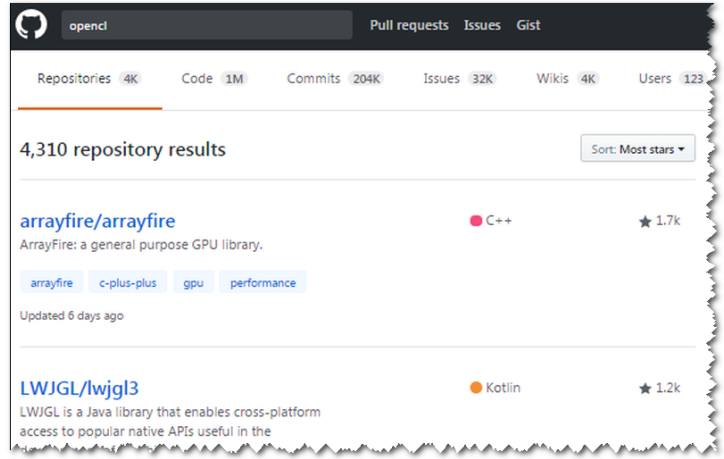
https://www.khronos.org/opencl/

OpenCL Conformant Implementations

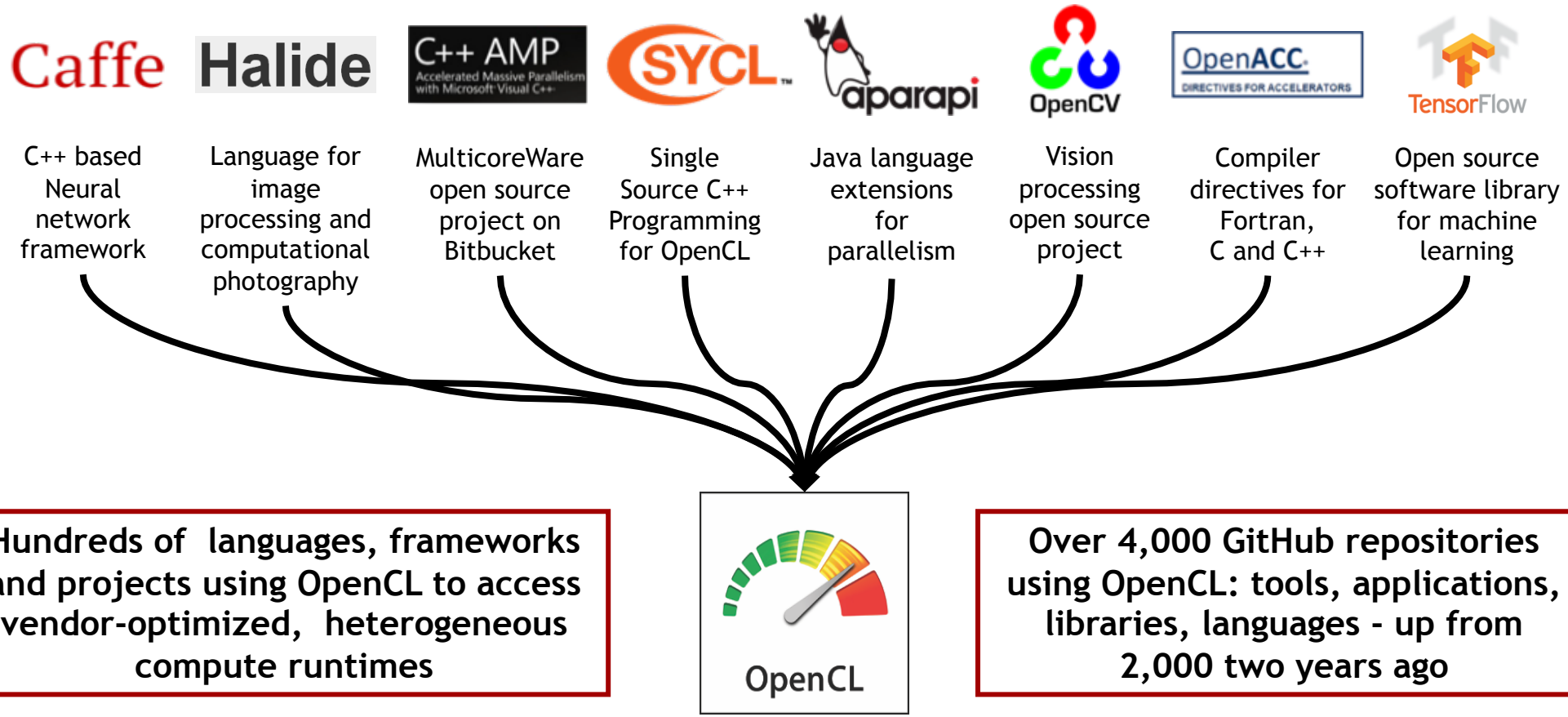


OpenCL Adoption

- **100s of applications using OpenCL acceleration**
  - Rendering, visualization, video editing, simulation, image processing
- **Over 4,000 GitHub repositories using OpenCL**
  - Tools, applications, libraries, languages
  - Up form 2,000 two years ago
- **Multiple silicon and open source implementations**
  - Increasingly used for embedded vision and neural network inferencing
- **Khronos Resource Hub**
  https://www.khronos.org/opencl/resources/opencl-applications-using-opencl



OpenCL as Language/Library Backend

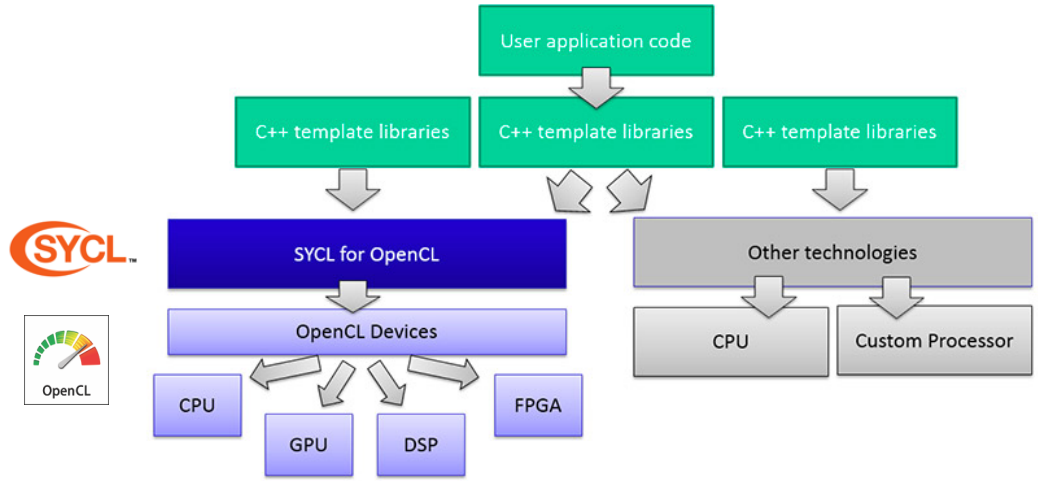


Hundreds of languages, frameworks and projects using OpenCL to access vendor-optimized, heterogeneous compute runtimes

Over 4,000 GitHub repositories using OpenCL: tools, applications, libraries, languages - up from 2,000 two years ago

SYCL Ecosystem

- **Single-source heterogeneous programming using STANDARD C++**
  - Use C++ templates and lambda functions for host & device code
  - Layered over OpenCL
- **Fast and powerful path for bring C++ apps and libraries to OpenCL**
  - C++ Kernel Fusion - better performance on complex software than hand-coding
  - Halide, Eigen, Boost.Compute, SYCLBLAS, SYCL Eigen, SYCL TensorFlow, SYCL GTX
  - triSYCL, ComputeCpp, VisionCpp, ComputeCpp SDK …
- **More information at http://sycl.tech**



The Choice of SYCL 2.2 or OpenCL C++

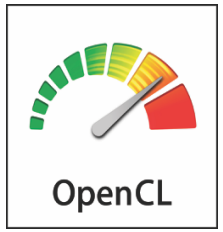
Developer Choice

The development of the two specifications are aligned so code can be easily shared between the two approaches

C++ Kernel Language

Low Level Control

‘GPGPU’-style separation of device-side kernel source code and host code



Single-source C++

Programmer Familiarity

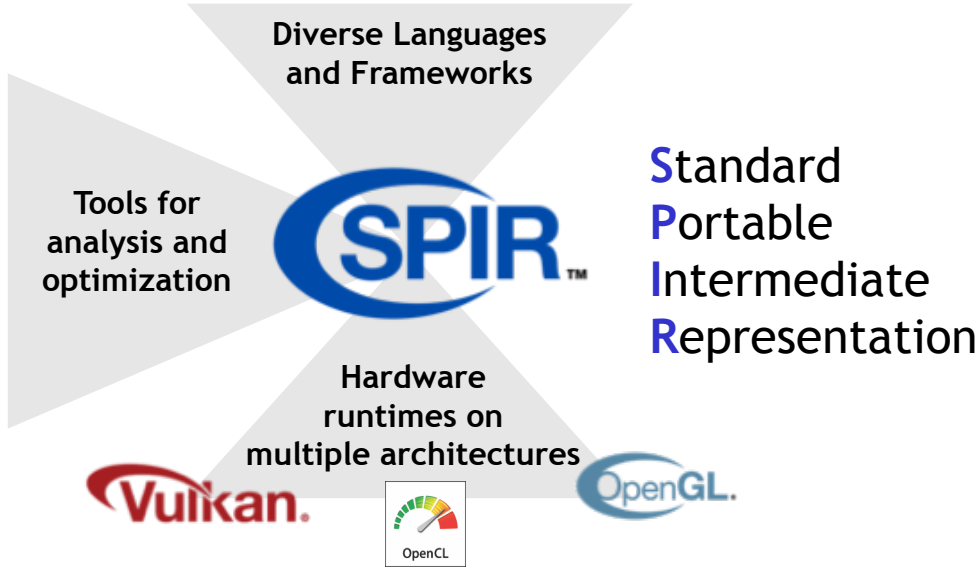
Approach also taken by C++ AMP, OpenMP and the C++ 17 Parallel STL

SPIR-V Transforms the Language Ecosystem

- **First multi-API, intermediate language for parallel compute and graphics**
  - Natively represents structures in shader and kernel languages
  - https://www.khronos.org/registry/spir-v/papers/WhitePaper.pdf
- **Compiler IR for OpenCL, Vulkan and OpenGL**
  - Easy to parse - just a stream of words
  - Easy to transform - designed to be easy to convert to and from LLVM IR
  - Easy to manipulate and optimize - single-Static Assignment form

Multiple Developer Advantages

Use same front-end compiler for all platforms
Ship SPIR-V – not shader source code
Simpler and more reliable drivers
Reduces runtime kernel compilation time



Standard
Portable
Intermediate
Representation

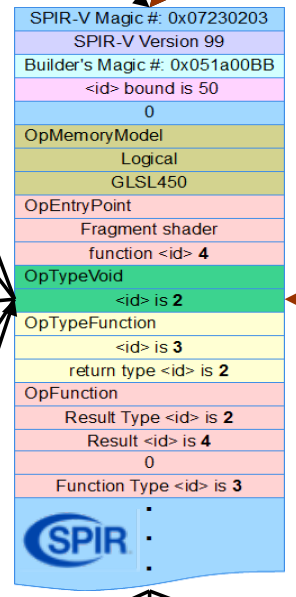
SPIR-V Ecosystem

Khronos open source tools and translators

https://github.com/KhronosGroup/SPIRV-Tools

SPIR-V
- Khronos defined cross-API IR
- Native graphics and parallel compute
- Easily parsed/extended 32-bit stream
- Data object/control flow retained for effective code generation/translation

GLSL
HLSL
Third party kernel and shader languages
glslang
MSL
HLSL
GLSL
SPIRV-Cross
SPIR-V (Dis)Assembler
SPIR-V Validator
SPIRV-opt | SPIRV-remap
Additional Intermediate Forms



OpenCL C Front-end
OpenCL C++ Front-end
LLVM
LLVM to SPIR-V Bi-directional Translator

Khronos liaising with Clang/LLVM Community
E.g. discussing SPIR-V as supported Clang target

NEW!

SPIR-V Optimizations
- Inlining (exhaustive)
- Store/Load Elimination
- Dead Code Elimination
- Dead Branch Elimination
- Common Uniform Elimination

Coming
- Loop Unrolling and Constant Folding
- Common Subexpression Elimination

IHV Driver Runtimes

OpenCL
Vulkan
OpenGL

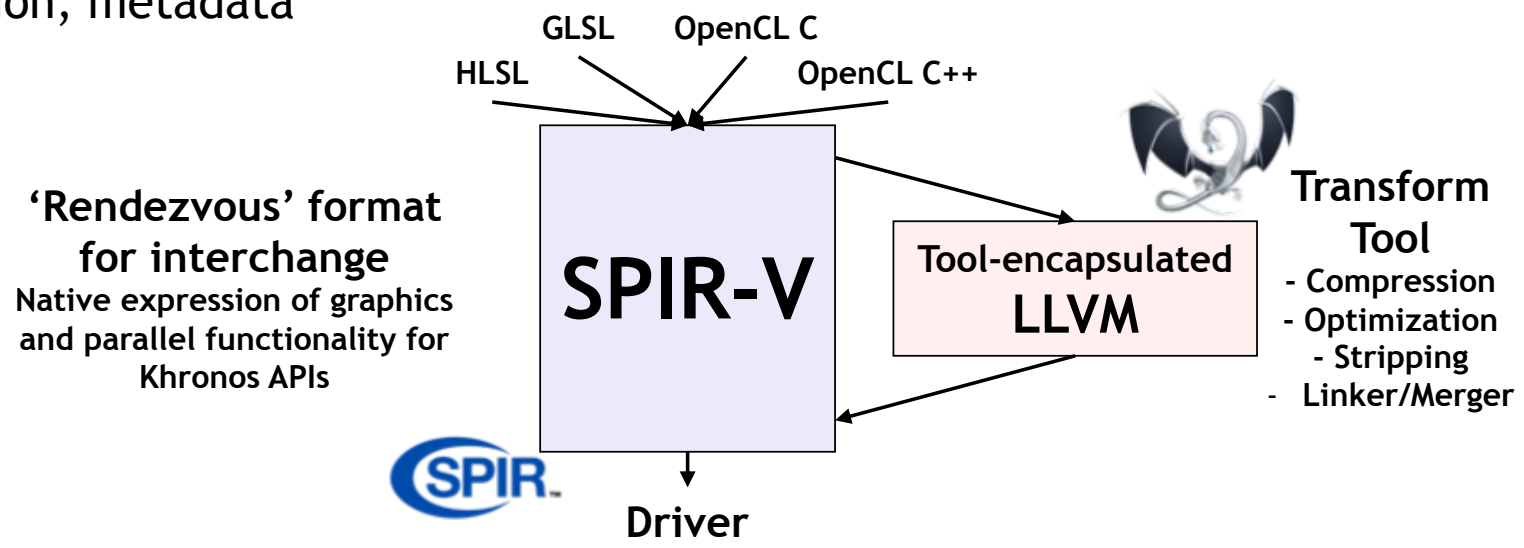
Evolution of SPIR Family

- **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 open sourcing SPIR-V <-> LLVM conversion tools

SPIR	SPIR 1.2	SPIR 2.0	SPIR-V 1.X
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.X OpenCL C++ GLSL
OpenCL Ingestion	OpenCL 1.2 Extension	OpenCL 2.0 Extension	OpenCL 2.1/2.2 Core
Graphics API Ingestion	-	-	Vulkan and OpenGL 4.6 Core

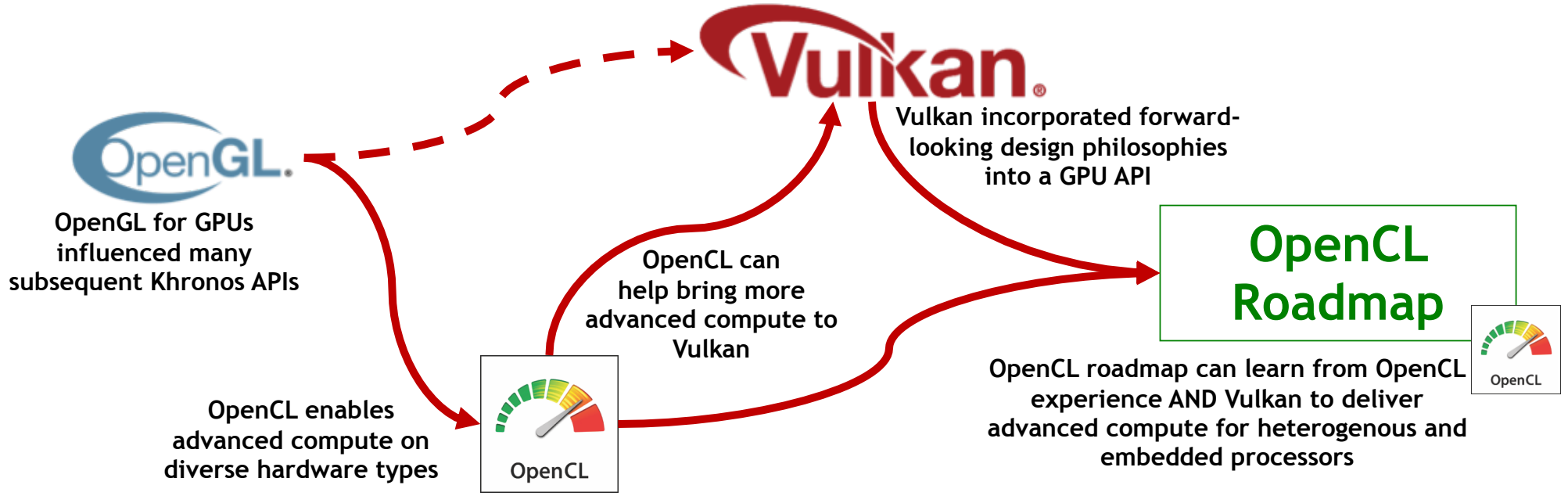
Support for Both SPIR-V and LLVM

- **LLVM is an SDK, not a formally defined standard**
  - Khronos moved away from trying to use LLVM IR as a standard
  - Issues with versioning, metadata, etc.
- **But LLVM is a treasure chest of useful transforms**
  - SPIR-V tools can encapsulation and use LLVM to do useful SPIR-V transforms
- **SPIR-V tools can all use different rules - and there will be lots of these**
  - May be lossy and only support SPIR-V subset
  - Internal form is not standardized
  - May hide LLVM version, metadata



Agenda for this Session

- Introduction to Khronos and the need for open standards
- Update on OpenCL (and SPIR-V) – the standards for heterogenous computing
- Update on Vulkan – a new explicit-style API
- Lessons from Vulkan and OpenCL that can guide the OpenCL roadmap
- Directions for OpenCL roadmap – and increasing relevance to the embedded industry



The Key Principle of Vulkan: *Explicit Control*

- **Application tells the driver what it is going to do**
  - *In enough detail* that driver doesn't have to guess
  - *When* the driver needs to know it

You have to supply a LOT of information!

And, you have to supply it early

- **In return, driver promises to do**
  - *What* the application asks for
  - *When* it asks for it
  - *Very quickly*

Efficient, predictable performance

- ***No driver magic – no surprises***

Vulkan Explicit GPU Control

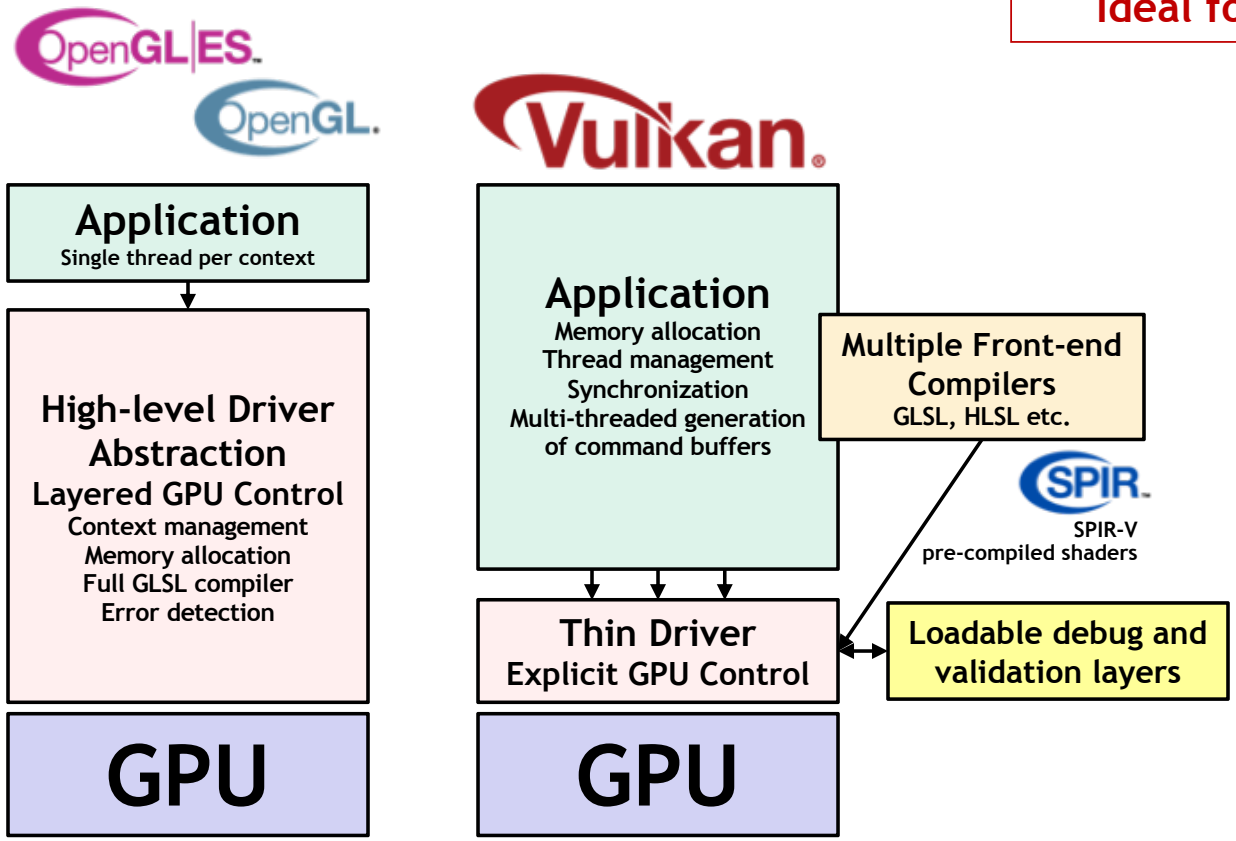
Vulkan = high performance and low latency 3D and GPU Compute. Ideal for VR/AR applications

Complex drivers cause overhead and inconsistent behavior across vendors

Always active error handling

Full GLSL preprocessor and compiler in driver

OpenGL vs. OpenGL ES



Vulkan 1.0 provides access to
OpenGL ES 3.1 / OpenGL 4.X-class GPU functionality
but with increased performance and flexibility

Resource management offloaded to app: low-overhead, low-latency driver

Consistent behavior: no 'fighting with driver heuristics'

Validation and debug layers loaded only when needed

SPIR-V intermediate language: shading language flexibility

Multi-threaded command creation. Multiple graphics, command and DMA queues

Unified API across all platforms with feature set flexibility

The Need for Cross-Platform 3D Standards

Vulkan Provides

Clean, modern architecture | Low overhead, explicit GPU access
Portable across desktop and mobile | Multi-thread / multi-core friendly
Primary 3D API on Android Platform
Efficient, low-latency, predictable performance
Access to multiple platforms

Pervasive Vulkan

All Major GPU Companies shipping Vulkan Drivers for Desktop and Mobile Platforms

AMD arm Imagination intel NVIDIA QUALCOMM VeriSilicon

http://vulkan.gpuinfo.org/

Desktop, Mobile, Embedded and Console Platforms Supporting Vulkan

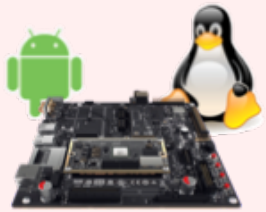
Including phones and tablets from Google, Huawei, Samsung, Sony, Xiaomi - both premium and mid-range devices

Desktop

Android 7.0+

Nintendo Switch


Android TV


Embedded Linux

VR Platforms


SteamVR


GearVR


Oculus Rift

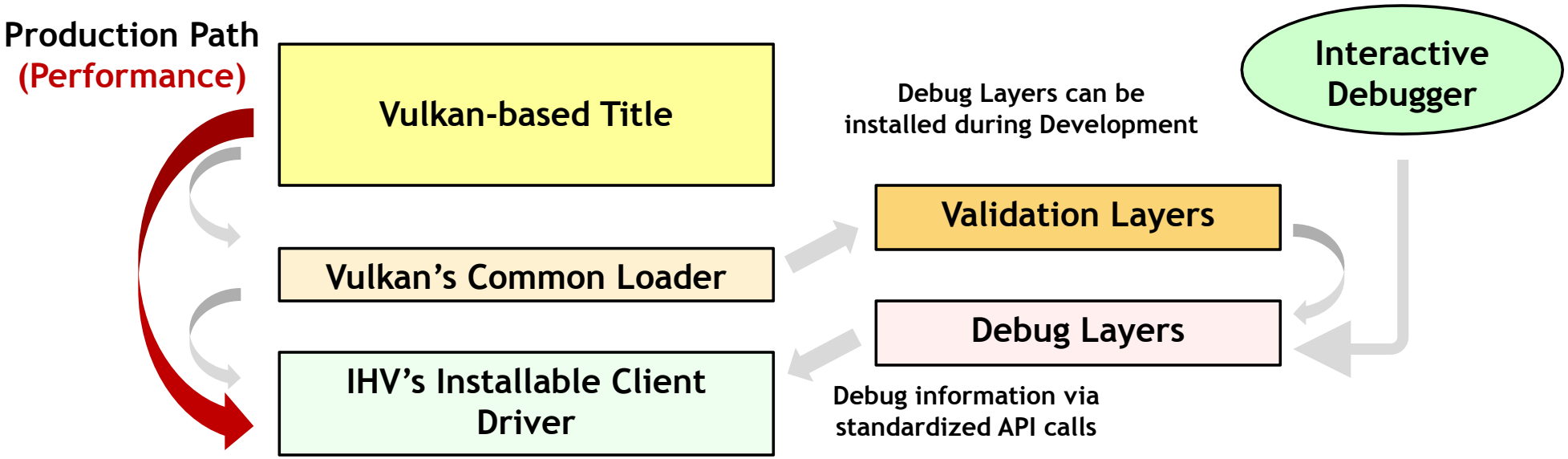

Google Daydream

Game Engines

VALVE id CRYENGINE unity EPIC GAMES CROTEAM XENKO

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
- **Khronos Open Source Loader enables use of tools layers during debug**
  - Finds and loads drivers, dispatches API calls to correct driver and layers



Vulkan Feature Sets

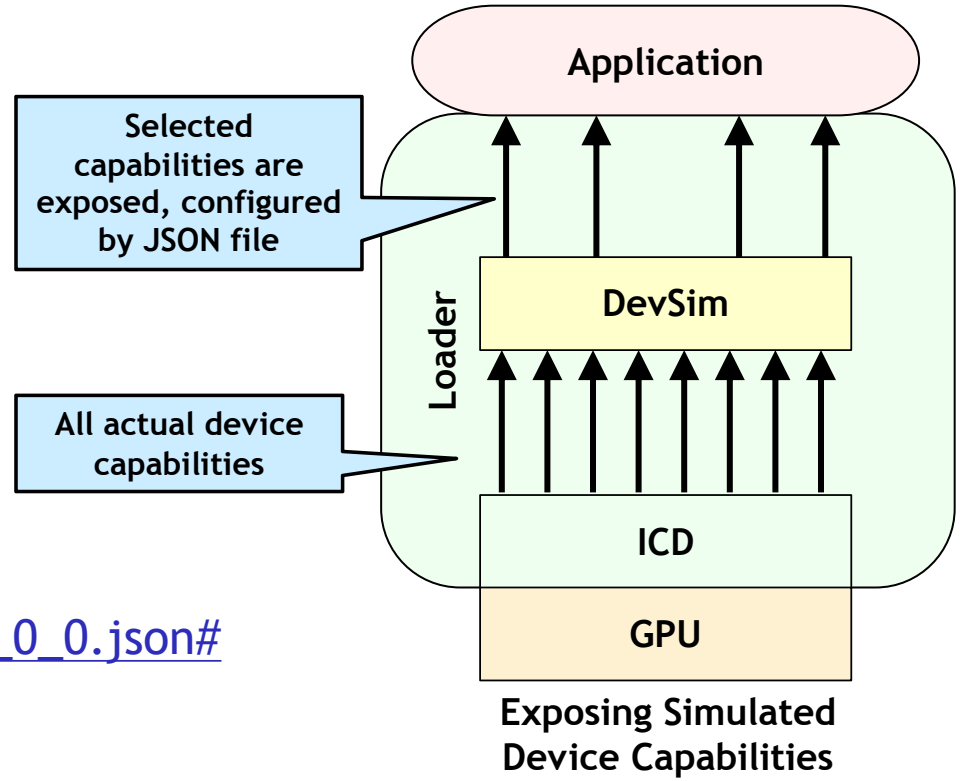
- **Vulkan supports hardware with a wide range of hardware capabilities**
  - Mobile OpenGL ES 3.1 up to desktop OpenGL 4.5 and beyond
- **One unified API framework for desktop, mobile, console, and embedded**
  - No "Vulkan ES" or "Vulkan Desktop"
- **Vulkan precisely defines a set of "fine-grained features"**
  - Features are specifically enabled at device creation time (similar to extensions)
- **Platform owners define a Feature Set for their platform**
  - Vulkan provides the mechanism but does not mandate policy
  - Khronos will define Feature Sets for platforms where owner is not engaged
- **Khronos will define feature sets for Windows and Linux**
  - After initial developer feedback

KHRONOS GROUP

Vulkan®

VK_LAYER_LUNARG_device_simulation

- **Simulate capabilities of mobile and embedded devices**
  - Test application without requiring physical device
  - Exercise fall-back code paths, when a capability isn't available
  - Find unintentional assumptions (triggers validation errors)
- **Modifies results from Vulkan queries**
  - Device configuration defined by JSON file
  - Integrated with Sascha Willems database
- **Simulation, NOT Emulation**
  - Doesn't add more capabilities not already present in actual device
- **Source available now**
  - https://github.com/LunarG/VulkanTools
  - Please submit issues
- **Verify configuration files are correct**
  - https://schema.khronos.org/vulkan/devsim_1_0_0.json#



Exposing Simulated Device Capabilities

Vulkan Genesis

Khronos members from all segments of the graphics industry agree the need for new generation cross-platform GPU API

Significant proposals, IP contributions and engineering effort from many working group members

Khronos' first API 'hard launch' February 2016

Unprecedented level of participation from game engine developers

18 months

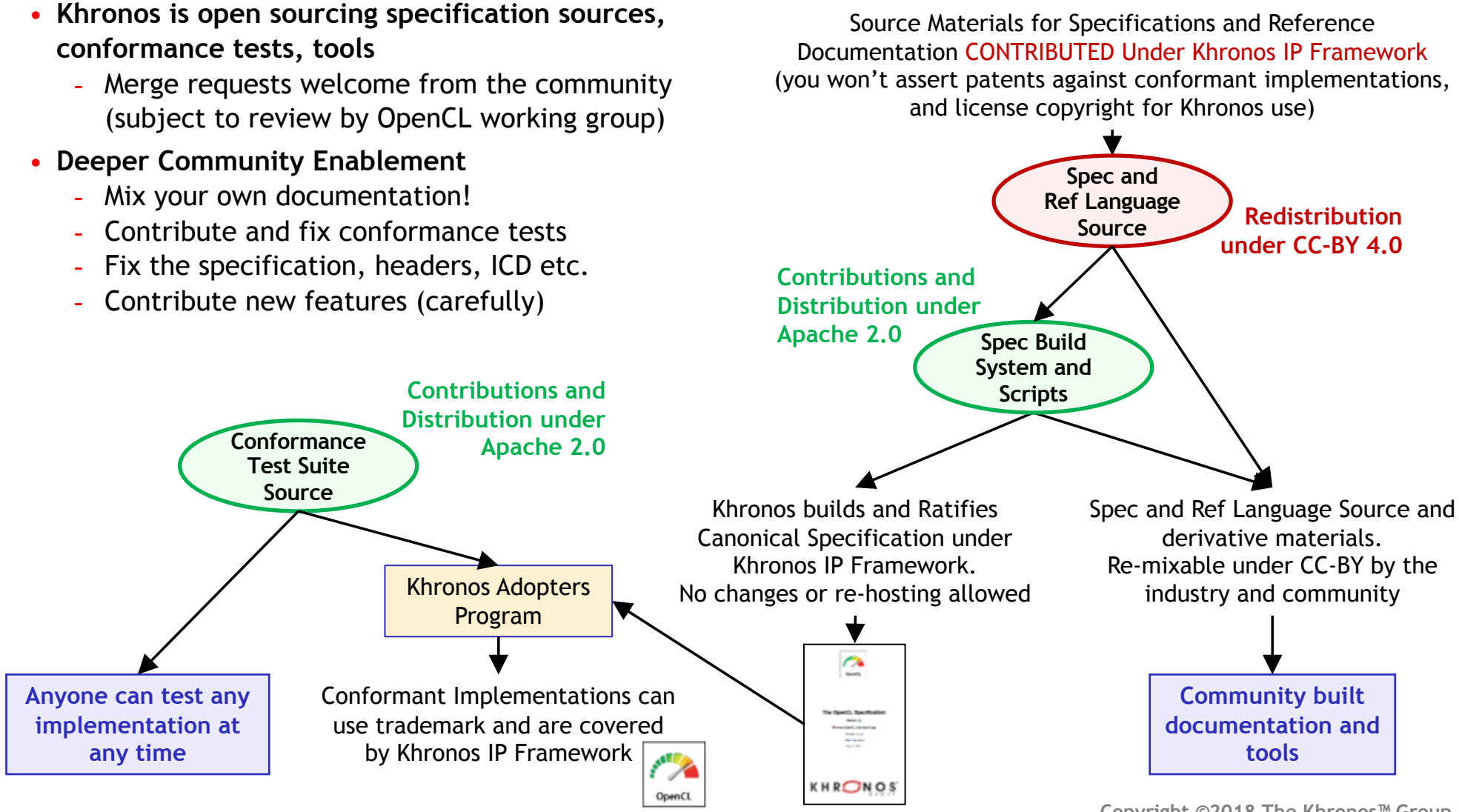
A high-energy working group effort

Specifications, Conformance Tests, SDKs, Reference Materials, Compiler front-ends, Samples - all open source…
Multiple Conformant Drivers on multiple OS

Vulkan Working Group Participants

New Open Source Engagement Model

- **Khronos is open sourcing specification sources, conformance tests, tools**
  - Merge requests welcome from the community (subject to review by OpenCL working group)
- **Deeper Community Enablement**
  - Mix your own documentation!
  - Contribute and fix conformance tests
  - Fix the specification, headers, ICD etc.
  - Contribute new features (carefully)



Vulkan Evolution

Strengthened Ecosystem and SDK

Enhanced developer and debugging tools
Regression testing for SDK stability
Enhanced Conformance Testing (API now has 198K test cases - up from 107K last year)
Compiler robustness - including HLSL support

Feb16
Vulkan 1.0

Explicit Access to GPU Acceleration

Vulkan Extensions

Maintenance updates plus additional functionality

Explicit Building Blocks for VR
Explicit Building Blocks for Homogeneous Multi-GPU
Enhanced Windows System Integration
Increased Shader Language Flexibility
Enhanced Cross-Process and Cross-API Sharing

Widened Platform Support

Through Vulkan Portability Initiative
Including Vulkan on macOS and iOS

Vulkan Roadmap

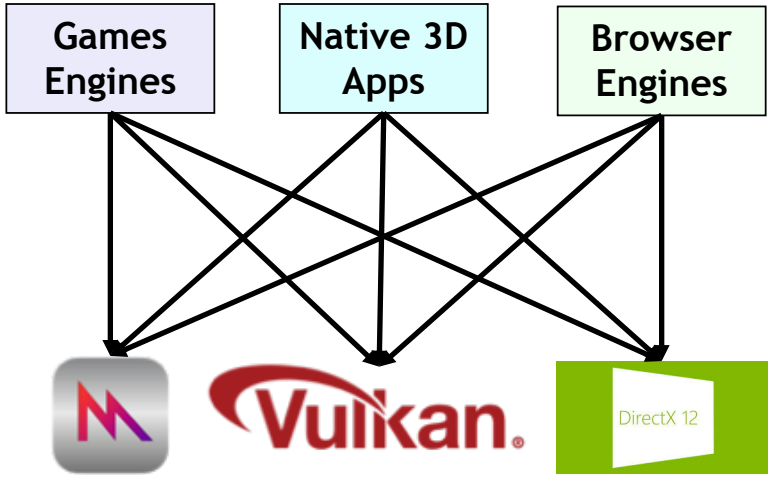
Regular Vulkan Core Releases

Integrates proven KHR extensions

Roadmap Discussions

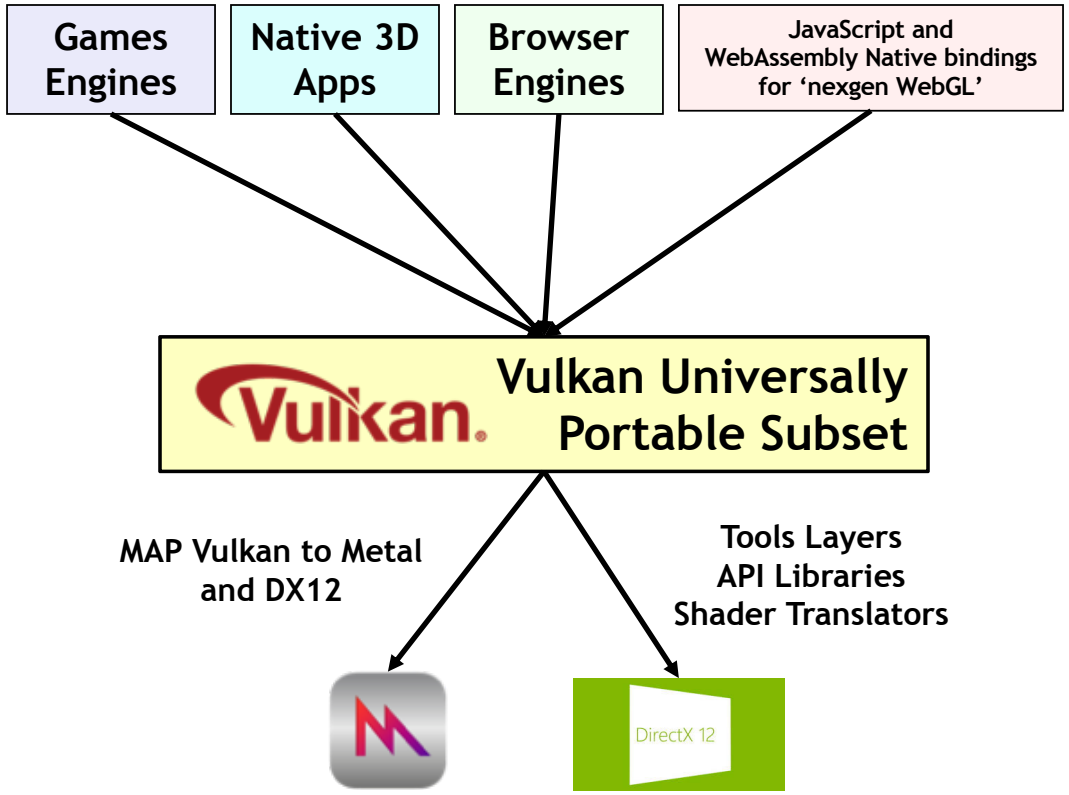
Pushes forward the envelope of GPU Acceleration:
Enhanced Compute and Language Flexibility
Optimized Vision and Inferencing Acceleration
Ray Tracing

Market Demand for Universal 3D Portability

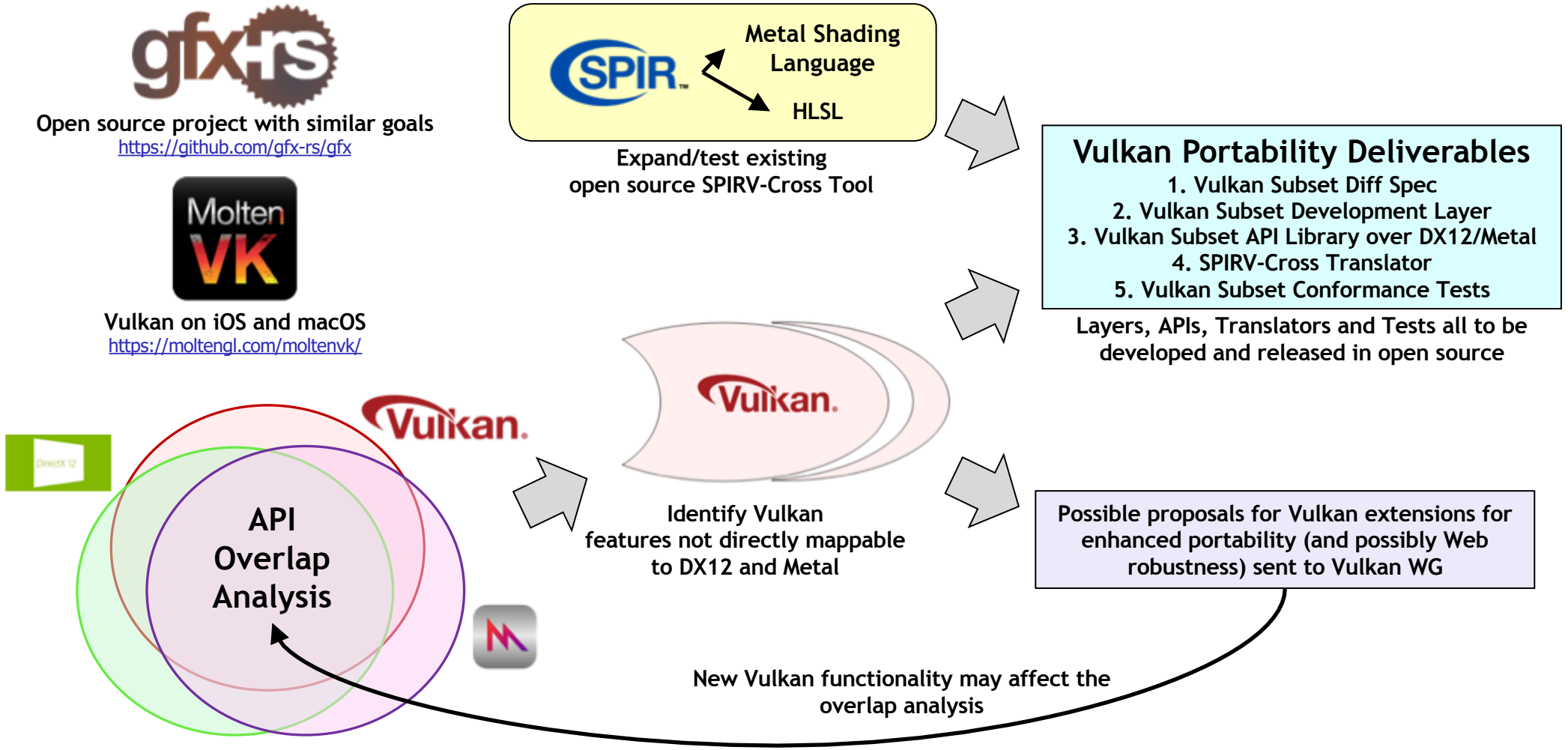


Community Outreach at GDC 2017
Create a hybrid Portability API?

Feedback - AVOID CREATING A FOURTH API!!!
Would need new specification, CTS, Documentation.
Additional developer learning curve.
A whole new specification to name, brand, promote.
Would INCREASE industry fragmentation

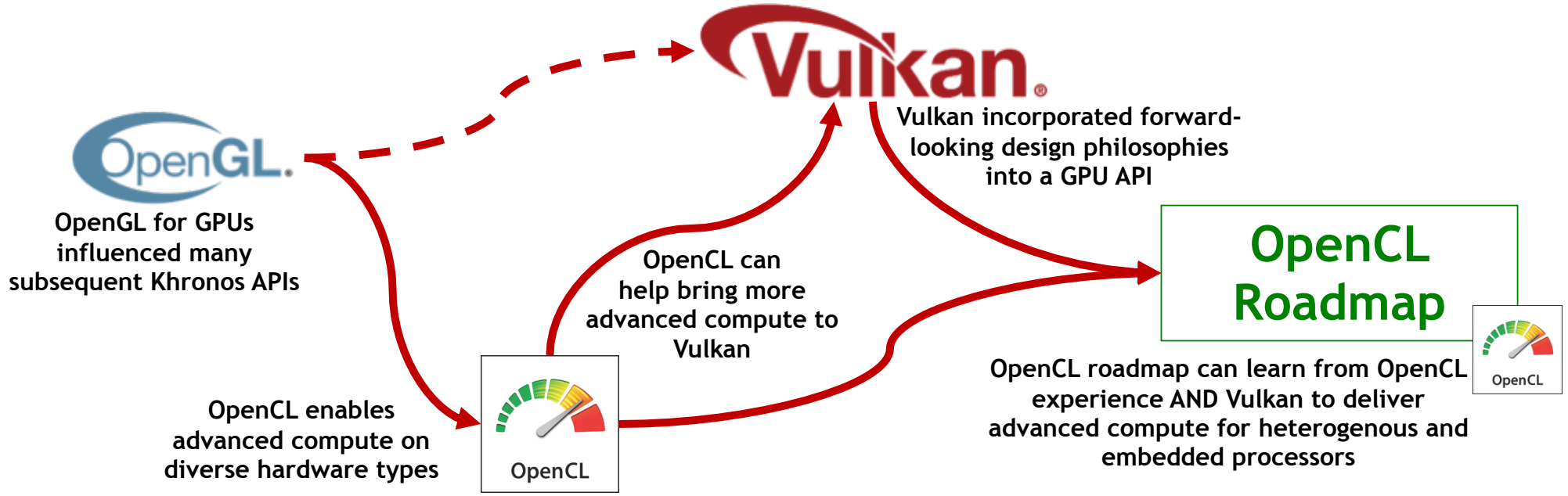


Vulkan Universal Portability



Agenda for this Session

- Introduction to Khronos and the need for open standards
- Update on OpenCL (and SPIR-V) – the standards for heterogenous computing
- Update on Vulkan - a new explicit-style API
- Lessons from Vulkan and OpenCL that can guide the OpenCL roadmap
- Directions for OpenCL roadmap – and increasing relevance to the embedded industry



Key Vulkan Lessons

- **Engine developer insights were essential during design**
  - Engine prototyping during design was essential during design
- **Open sourcing tests, tools, specs drives deeper community engagement**
  - Support strong middleware ecosystem
- **Explicit API**
  - Provide straightforward hardware access
  - Aim for as much functionality portability as possible

Rely on middleware engines and libraries for
1. Ease of Use
2. Performance Portability

OpenCL - 1000s Man Years Effort

Single Source C++ Programming
Full support for features in C++14-based Kernel Language

API and Language Specs
Brings C++14-based Kernel Language into core specification

Portable Kernel Intermediate Language
Support for C++14-based kernel language e.g. constructors/destructors

OpenCL C++ Kernel Language
Static subset of C++14
Templates and Lambdas
SPIR-V 1.2 with C++ support
SYCL 2.2 single source C++
Pipes
Efficient device-scope communication between kernels
Multiple Code Generation Optimizations

Dec08 — OpenCL 1.0 Specification
- 3-component vectors
- Additional image formats
- Multiple hosts and devices
- Buffer region operations
- Enhanced event-driven execution
- Additional OpenCL C built-ins
- Improved OpenGL data/event interop

18 months

Jun10 — OpenCL 1.1 Specification
- Device partitioning
- Separate compilation and linking
- Enhanced image support
- Built-in kernels / custom devices
- Enhanced DX and OpenGL Interop

18 months

Nov11 — OpenCL 1.2 Specification
- Shared Virtual Memory
- On-device dispatch
- Generic Address Space
- Enhanced Image Support
- C11 Atomics
- Pipes
- Android ICD

24 months

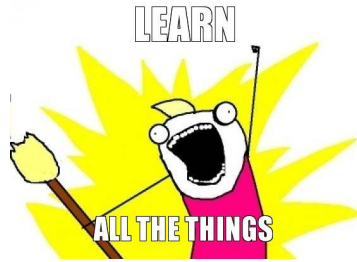
Nov13 — OpenCL 2.0 Specification
- SPIR-V in Core
- Subgroups into core
- Subgroup query operations
- clCloneKernel
- Low-latency device timer queries

24 months

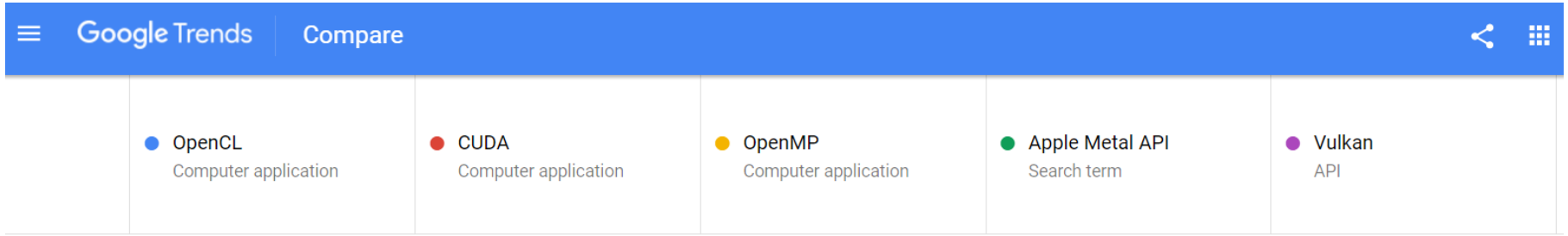
Nov15 — OpenCL 2.1 Specification

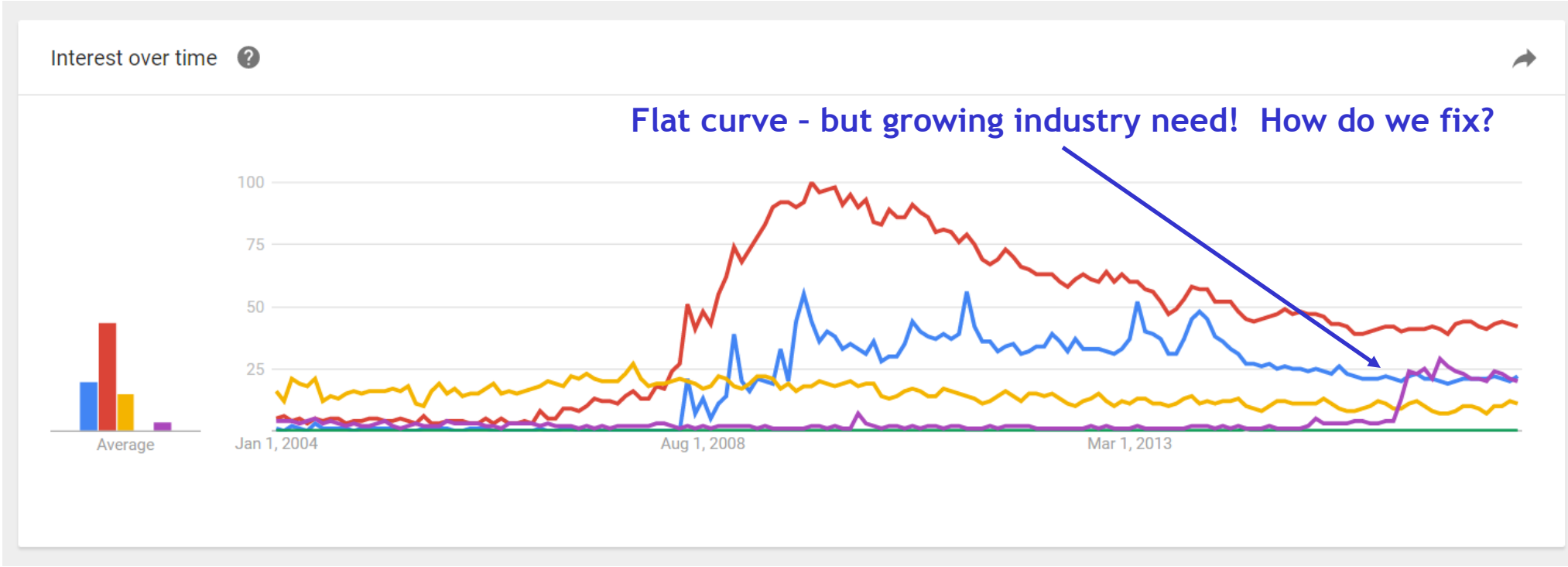
18 months

May17 — OpenCL 2.2 Specification



Google Trends





Key Lessons

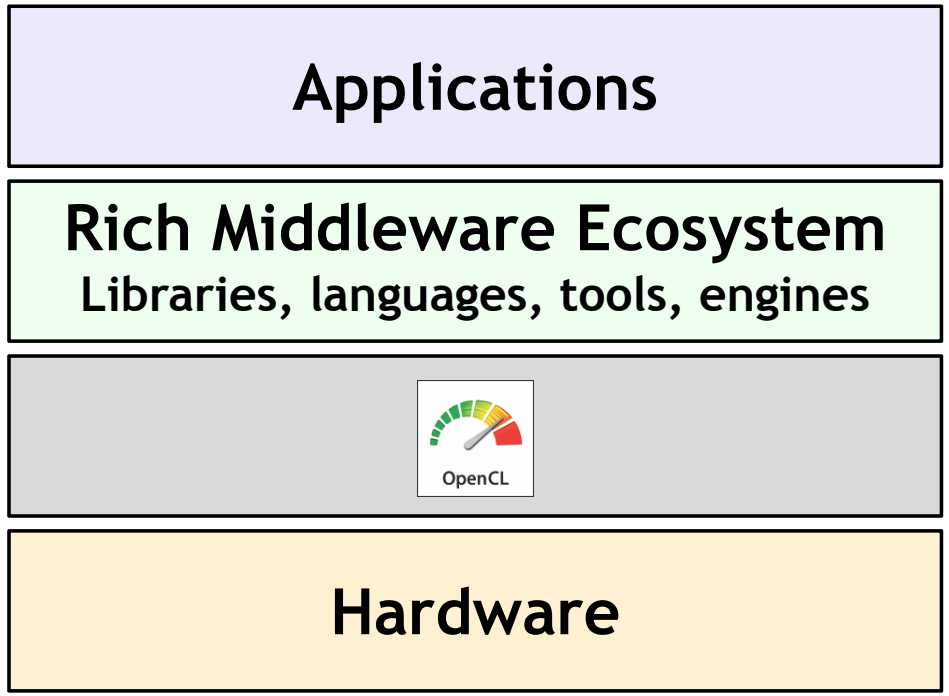
Lessons	How We Learned Them	How We Do Better!
Language flexibility is good! Enable language innovation!	OpenCL WG spent too long designing OpenCL C and C++	Ingest SPIR-V! and open source front-ends
Optimized tools and libraries are key	Lack of highly-optimized libraries for key use cases compared to proprietary solutions	Engage with library ecosystem and ensure the standard enables effective library deployment
Platform adoption is key influencer on overall ecosystem momentum	Apple are focused on Metal RenderScript confusion on Android NVIDIA not pushing to full 2.0	Enable implementations to ship that are relevant to a platform= and its customers
Software developer insights and prototyping are essential during standards design	OpenCL roadmap has been to feature oriented - not fully utilized by developers	Encourage ISVs to join Khronos as members or on OpenCL Advisory Panels

Embrace the Layered Ecosystem

OpenCL mixes providing low-level hardware access with 'ease-of-use'

Didn't make it clear that low-level performance portability is impossible

Did not focus on rapidly porting efficient libraries



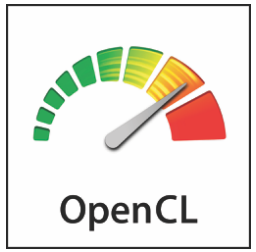
Middleware just needs direct access to hardware. Driver should 'get out of the way'

Middleware can provide ease of use

Middleware has the system/domain context to try to provide performance portability

Run-time abstraction to compute hardware IS needed:
- Software vendors can't afford to port to every type/generation hardware
- Hardware vendors want to keep innovating under an abstraction

Market Segments Need Deployment Flexibility



OpenCL has been over-monolithic

E.g. DSP inferencing should not be forced to ship IEEE FP32

Solution: feature sets – enabling toggling capabilities within a coherent framework without losing conformance

Desktop (physical and cloud)

Use cases: Video Image Processing, Gaming Compute, Rendering, Neural Network Training and Inferencing

Roadmap: Vulkan interop, dialable precision, pre-emption, collective programming and improved execution model, dynamic parallelism, pre-emption

---

Mobile

Use case: Photo and Vision Processing, Neural Network Inferencing

Roadmap: SVM, dialable precision for inference engine and pixel processing efficiency, pre-emption and QoS scheduling for power efficiency

---

HPC

Use case: Numerical Simulation, Neural Network Training, Virtualization

Roadmap: enhanced streaming processing, enhanced library support

---

FPGAs

Use cases: Network and Stream Processing

Roadmap: enhanced execution model, self-synchronized and self-scheduled graphs, fine-grained synchronization between kernels, DSL in C++

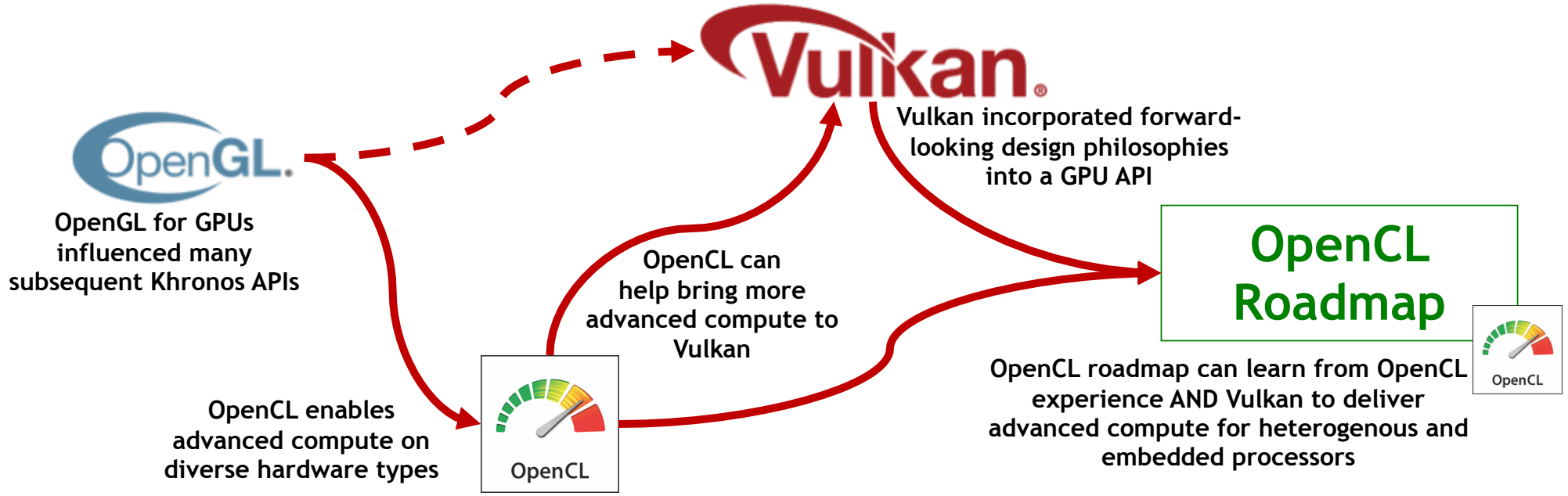
Embedded

Use cases: Vison, Signal and Pixel Processing, Neural Network and Inferencing

Roadmap: arbitrary precision for power efficiency, hard real-time scheduling, asynch DMA

Agenda for this Session

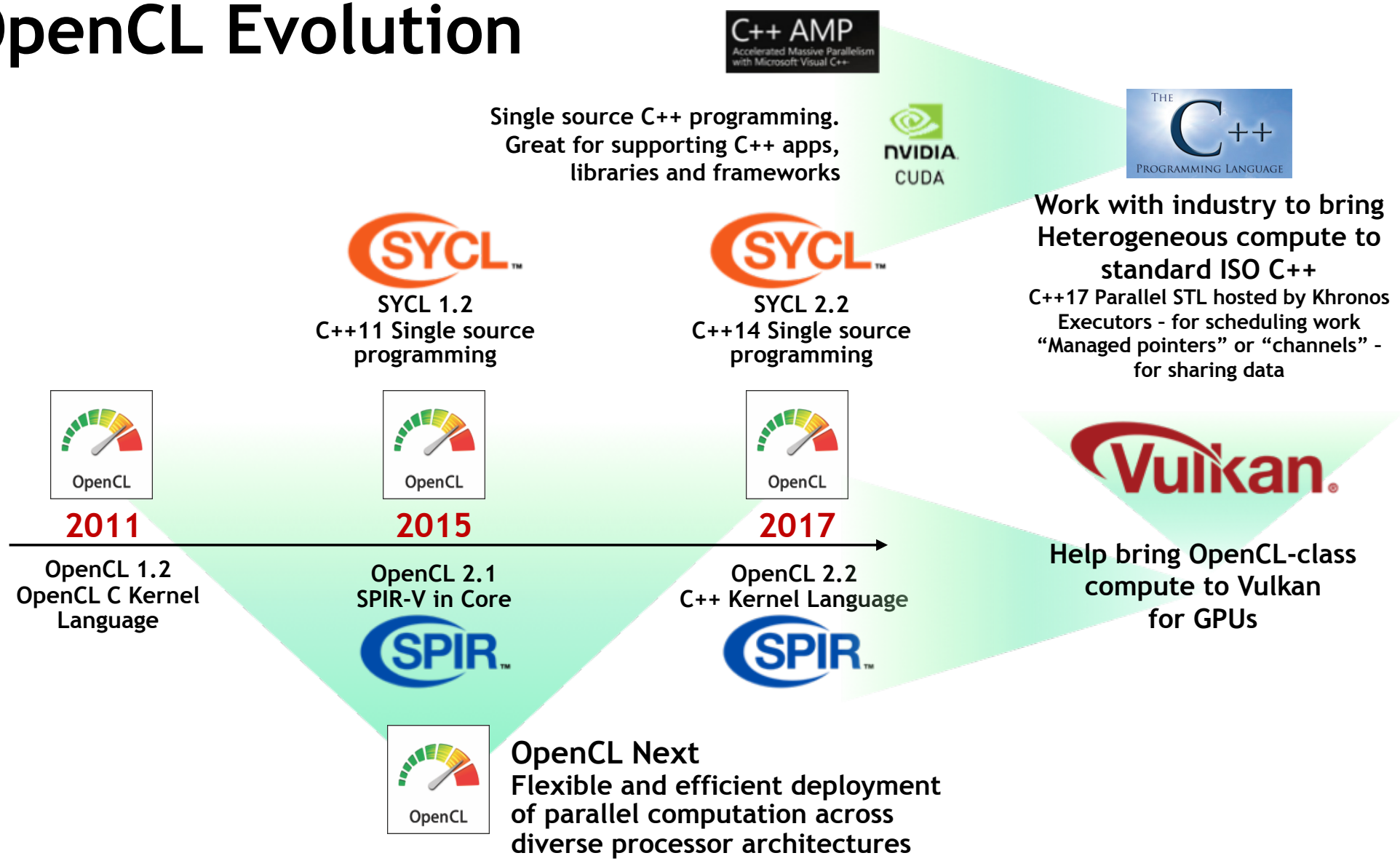
- Introduction to Khronos and the need for open standards
- Update on OpenCL (and SPIR-V) – the standards for heterogenous computing
- Update on Vulkan - a new explicit-style API
- Lessons from Vulkan and OpenCL that can guide the OpenCL roadmap
- Directions for OpenCL roadmap – and increasing relevance to the embedded industry



Requirements for OpenCL Roadmap

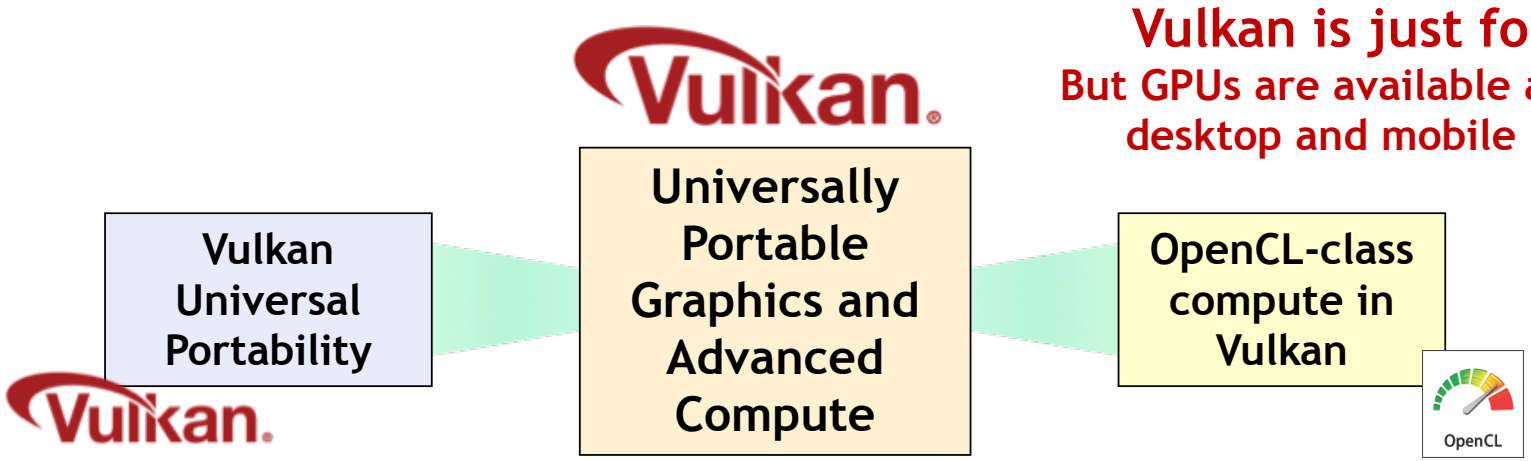
	OpenCL	Vulkan
Low-level explicit API as Foundation of multi-layer compute ecosystem	✓	✓
Flexible features selection for deployment flexibility	✓	✓
SPIR-V Ingestion for Language flexibility	✓	✓
Widely Adopted No market barriers to deployment	✓	✓
Installable tools architecture for development and deployment flexibility	✓	✓
Low-latency, multi-threaded dispatch for fine-grained, high-performance	✓	✓
Enables at least OpenCL 2.X-class compute capabilities	✓	✗
Support for diverse processor types	✓	✗

OpenCL Evolution



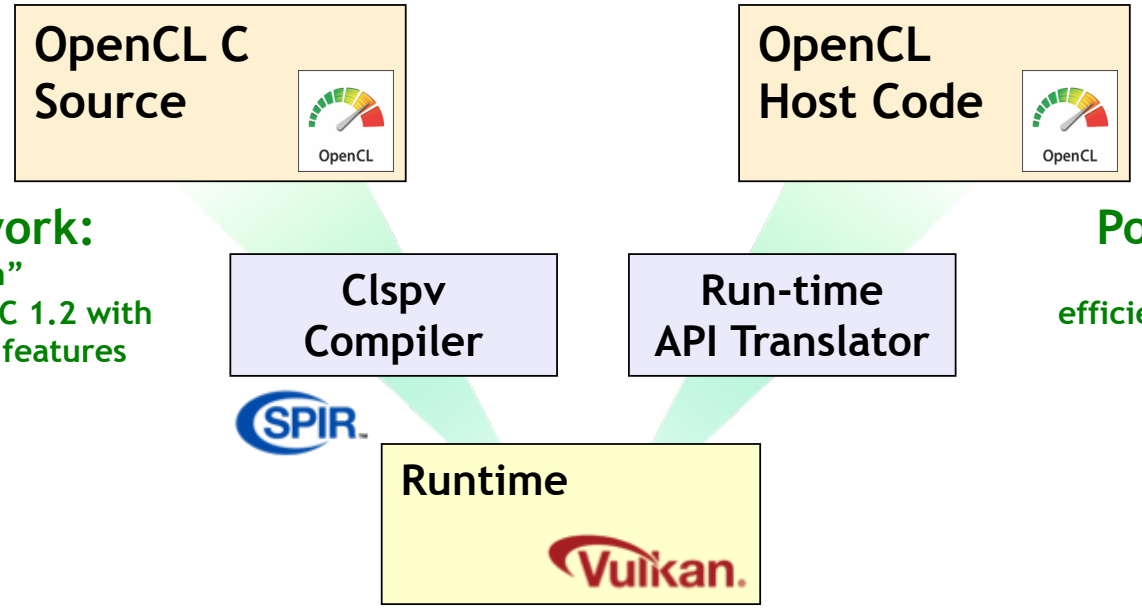
Bringing Advanced Compute to Vulkan

- **Already developed Vulkan/SPIR-V functionality to support OpenCL C compute operations**
  - 16-bit storage and use of fp16 and int8 in shaders
  - Variable Pointers and Subgroups
- **In future will target additional advanced use cases**
  - Improve Compute for Graphics (advocated in a number of HPG/SIGGRAPH 2016/17 talks, including Andrew Lauritzen's talk @ Open Problems in Real-Time Rendering, SIGGRAPH'17)
  - Improve support for processing images acquired from camera sensors
  - Enhanced support for performant and power-efficient machine learning



Clspv OpenCL C to Vulkan Compiler

- **Experimental collaboration between Google, Codeplay, and Adobe**
  - Successfully tested on over 200K lines of Adobe OpenCL C production code
  - Released in open source https://github.com/google/clspv
  - Tracks top-of-tree LLVM and clang, not a fork
- **Compiles OpenCL C's programming model to Vulkan's SPIR-V execution environment**
  - Proof-of-concept that OpenCL compute can be brought seamlessly to Vulkan



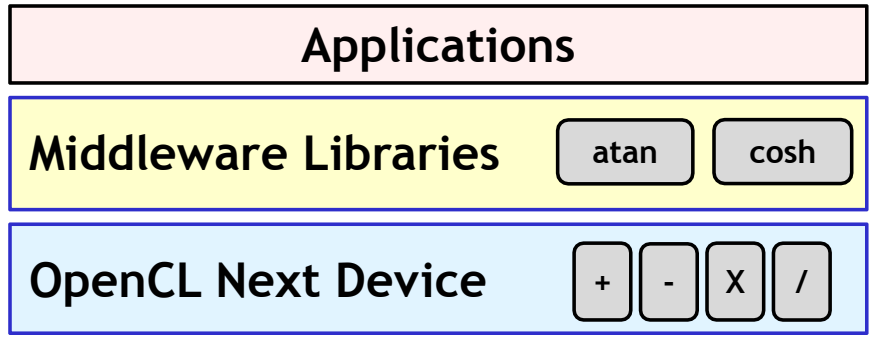
Possible future work:
"OpenCL C Vulkan Edition"
Proper subset of OpenCL C 1.2 with support for select Vulkan features

Possible future work:
Subset of OpenCL API that efficiently maps to Vulkan through a run-time shim

All tools, compilers and conformance tests in open source

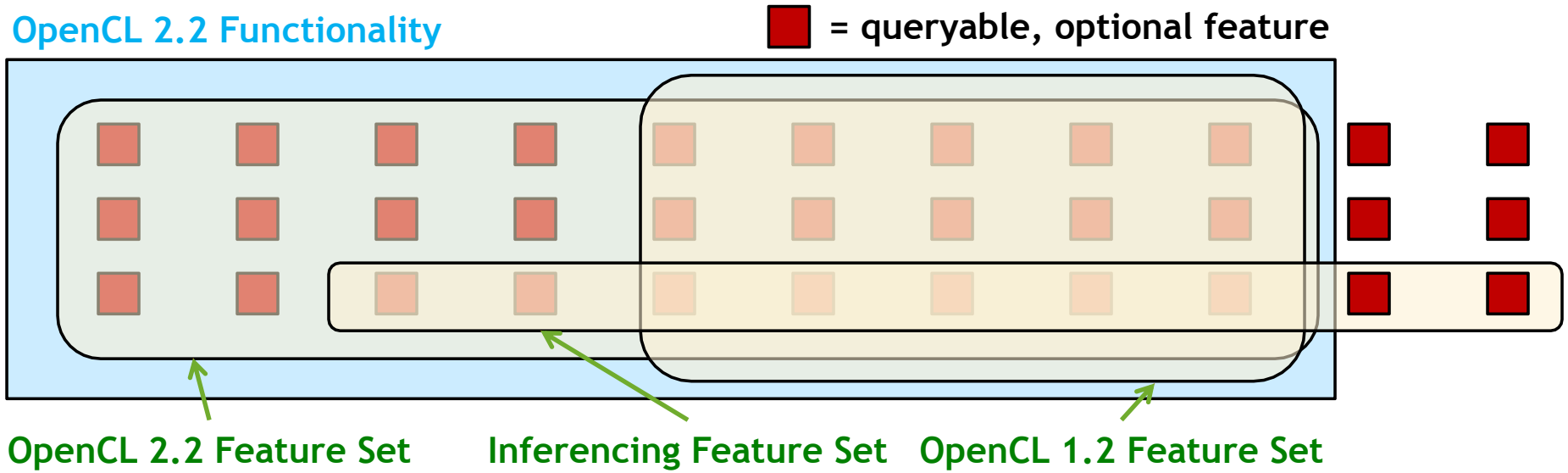
OpenCL Next

- **Enable diverse, heterogenous computing architectures**
  - Not just GPUs
- **Enable vendors to ship targeted functionality for their customers/markets**
  - While still being formally conformant
- **Empower middleware + libraries to emulate features**
  - Reduces implementation cost
  - Enables portability without vendors implementing all features
- **More features in OpenCL become optional – e.g. floating point**
  - Enable smaller devices to support recent OpenCL versions
  - Dependency tree between features
- **Enhanced query mechanisms**
  - Application queries which OpenCL features are supported by a device



OpenCL Next Feature Sets

- **Optional Feature Sets to reduce market fragmentation**
  - Opportunity for industry to flexibly agree on commonly supported functionality
  - Features optional in both language and API
- **Vendor can support ANY combination of features**
  - Not necessary to support any Feature Set – as long as supported features are conformant
  - If features include 100% of a Feature Set - can claim conformance to that Feature Set
  - Supporting popular Feature Sets may help drive sales
  - An Implementation may support multiple Feature Sets



OpenCL Next and DSPs

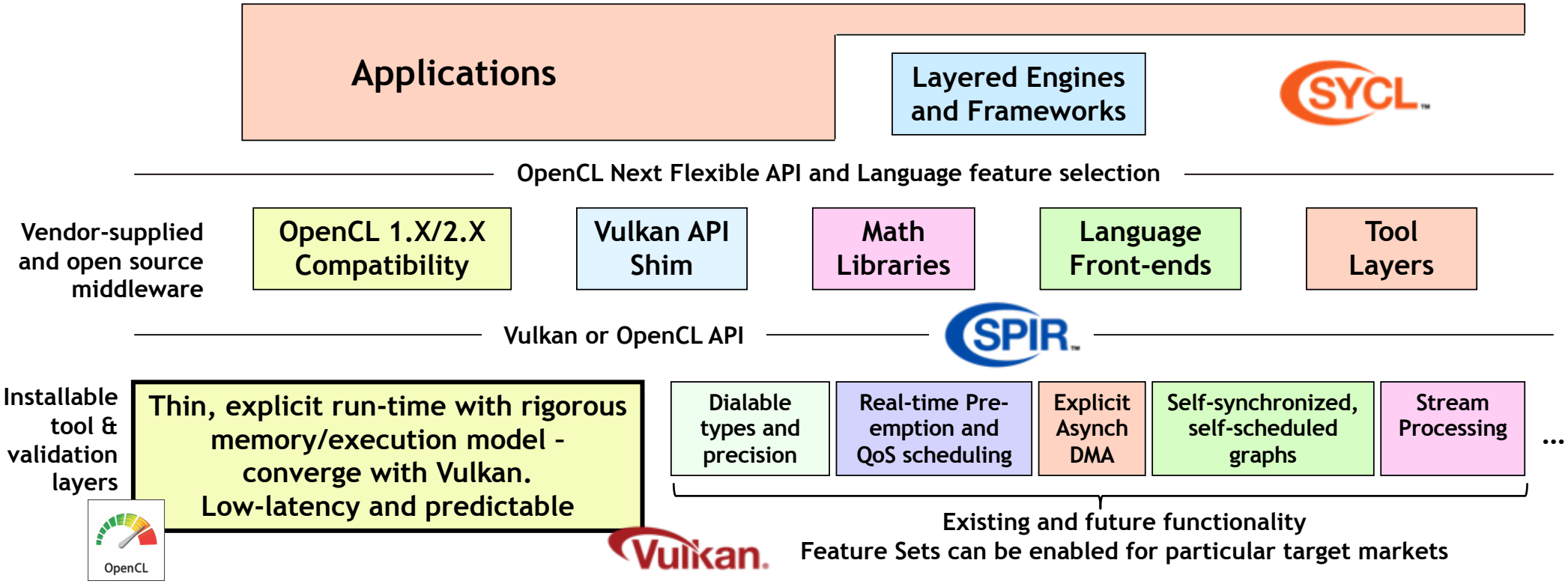
- **Flexible Feature Sets enable precise market targeting**
  - Embedded imaging, vision and inferencing etc.
- **What functionality should be optional?**
  - Flexible reduced precision
  - Conformance without IEEE 32 Floating Point
- **What new features for DSP and embedded devices?**
  - E.g. Explicit synch DMA
  - What else?

OpenCL can be a low-cost, flexible run-time framework to enable cost-effective code generation that is portable code across a wide range of heterogenous devices

	DSP
8-bit int	✓
16-bit int	✓
32-bit int	✓
64-bit int	✗
16-bit float	✗
32-bit float	✗
64-bit float	✗

Longer Term Roadmap Convergence

- **Use OpenCL Next flexibility to enable OpenCL API/language over diverse run-times**
  - OpenCL, Vulkan or third-party
- **Layered ecosystem for backwards-compatibility and market flexibility**
  - Feature sets for target market agility



Comments and Questions?

- We value your honest feedback!
- Looking for input/feedback, especially from DSP vendors!