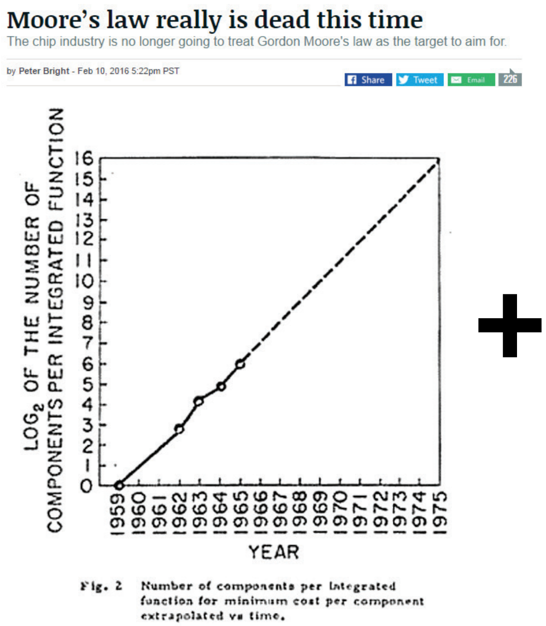
OpenCL
A State of the Union

Neil Trevett | Khronos President
NVIDIA Vice President Developer Ecosystem
OpenCL Working Group Chair
ntrevett@nvidia.com | @neilt3d

Vienna, April 2016

Need for Heterogeneous Parallelism



+

Flynn's Taxonomy (1966)

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

=

Languages / Directives

Libraries

Explicit Kernels

Threads and Messages

"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 Ecosystem

Hardware Implementers
Desktop/Mobile/Embedded/FPGA

Apple, AMD, IBM, intel, NVIDIA, ARM, Imagination, QUALCOMM, VeriSilicon, MEDIATEK, ST, SAMSUNG, ALTERA, XILINX, Texas Instruments, MARVELL

OpenCL 2.2 - Top to Bottom C++

SYCL™

Single Source C++ Programming

OpenCL

Core API and Language Specs

SPIR™

Portable Kernel Intermediate Language

Working Group Members
Apps/Tools/Tests/Courseware

codeplay, Adobe, mobica, HUAWEI, NXP, SONY, University of BRISTOL, University of Windsor, MULTICORE WARE, vmware, Los Alamos NATIONAL LABORATORY EST. 1943

OpenCL 2.2

- Provisional - seeking industry feedback before finalization at SIGGRAPH or SC16
- OpenCL C++ kernel language into core
- SPIR-V 1.1 adds OpenCL C++ support
- SYCL 2.2 fully leverages OpenCL 2.2 from a single source file
- Runs on any OpenCL 2.0-capable hardware

Dec08	18 months	Jun10	18 months	Nov11	24 months	Nov13	24 months	Nov15	7months	May16
OpenCL 1.0 Specification		OpenCL 1.1 Specification		OpenCL 1.2 Specification		OpenCL 2.0 Specification		OpenCL 2.1 Specification		OpenCL 2.2 PROVISIONAL
		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		Device partitioning Separate compilation and linking Enhanced image support Built-in kernels / custom devices Enhanced DX and OpenGL Interop		Shared Virtual Memory On-device dispatch Generic Address Space Enhanced Image Support C11 Atomics Pipes Android ICD		SPIR-V in Core Subgroups into core Subgroup query operations clCloneKernel Low-latency device timer queries		OpenCL C++ Kernel Language SPIR-V 1.1 with C++ support SYCL 2.2 for single source C++

OpenCL C++ Kernel Language

- **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
- **Enhanced support for authoring libraries**
  - Increased safety, reduced undefined behavior while accessing atomics, iterators, images, samplers, pipes, device queue built-in types and address spaces

Safer, more adaptable, more reusable parallel software

OpenCL

KHRONOS™ GROUP

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

OpenCL

Single-source C++

Programmer Familiarity

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

SYCL™

SYCL is an important initiative to represent the OpenCL perspective as the industry as a whole figures out parallel programming from C++

KHRONOS™ GROUP

More OpenCL 2.2 - with help from SPIR-V 1.1

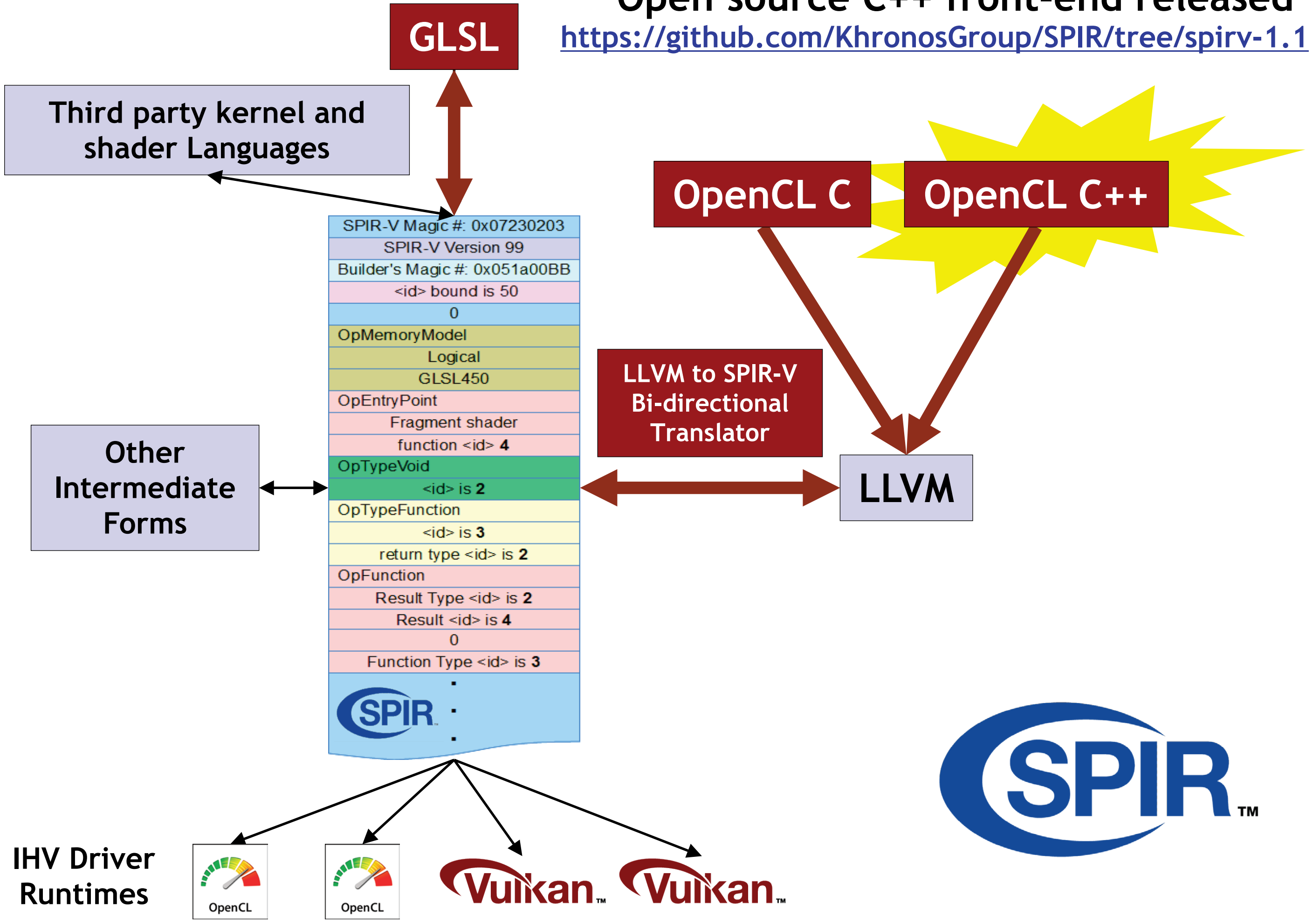
- **SPIR-V 1.1 adds full support for OpenCL C++**
  - Initializer/finalizer function execution modes to support constructors/destructors
  - Enhances the expressiveness of kernel programs by supporting named barriers, subgroup execution, and program scope pipes
- **SPIR-V specialization constants - previously available in Vulkan shaders**
  - SPIR-V module can express a family of parameterized OpenCL kernel programs
  - Embedded compile-time settings can be specialized at runtime
  - Eliminates the need to ship or recompile multiple variants of a kernel
- **Pipe storage device-side type - useful for FPGA implementations**
  - Makes connectivity size and type known at compile time
  - Enables efficient device-scope communication between kernels
- **Enhanced optimization of generated code**
  - Query non-trivial constructors/destructors of program scope global objects
  - User callbacks can be set at program release time

SPIR-V Ecosystem

Khronos has open sourced these tools and translators

Khronos plans to open source these tools soon

Open source C++ front-end released
https://github.com/KhronosGroup/SPIR/tree/spirv-1.1



SPIR-V Tools

SPIR-V Validator

SPIR-V (Dis)Assembler

SPIR-V

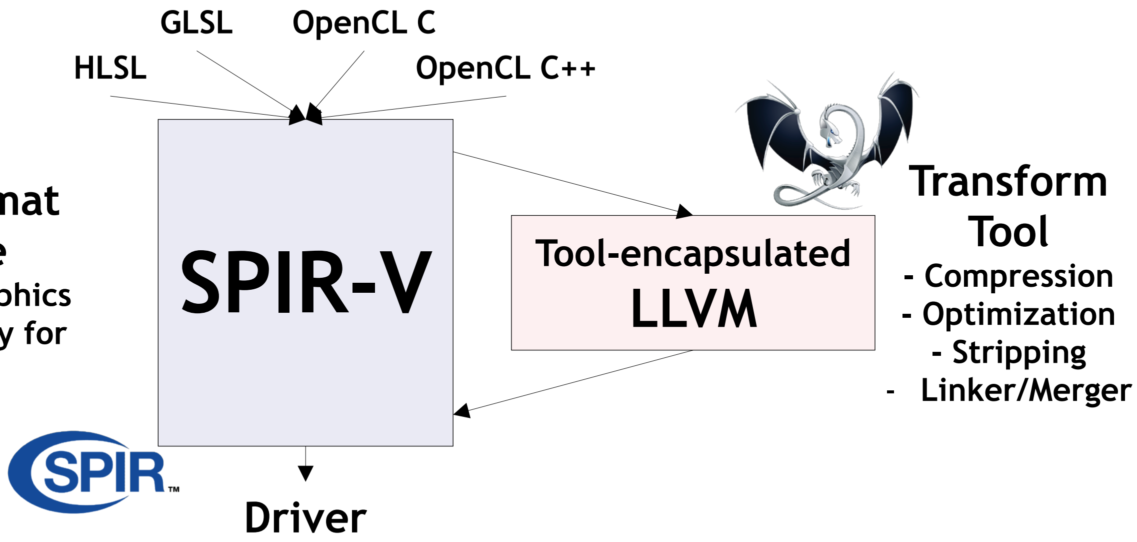
- Khronos defined and controlled cross-API intermediate language
  - Native support for graphics and parallel constructs
  - 32-bit Word Stream
- Extensible and easily parsed
- Retains data object and control flow information for effective code generation and translation

Support for Both SPIR-V and LLVM

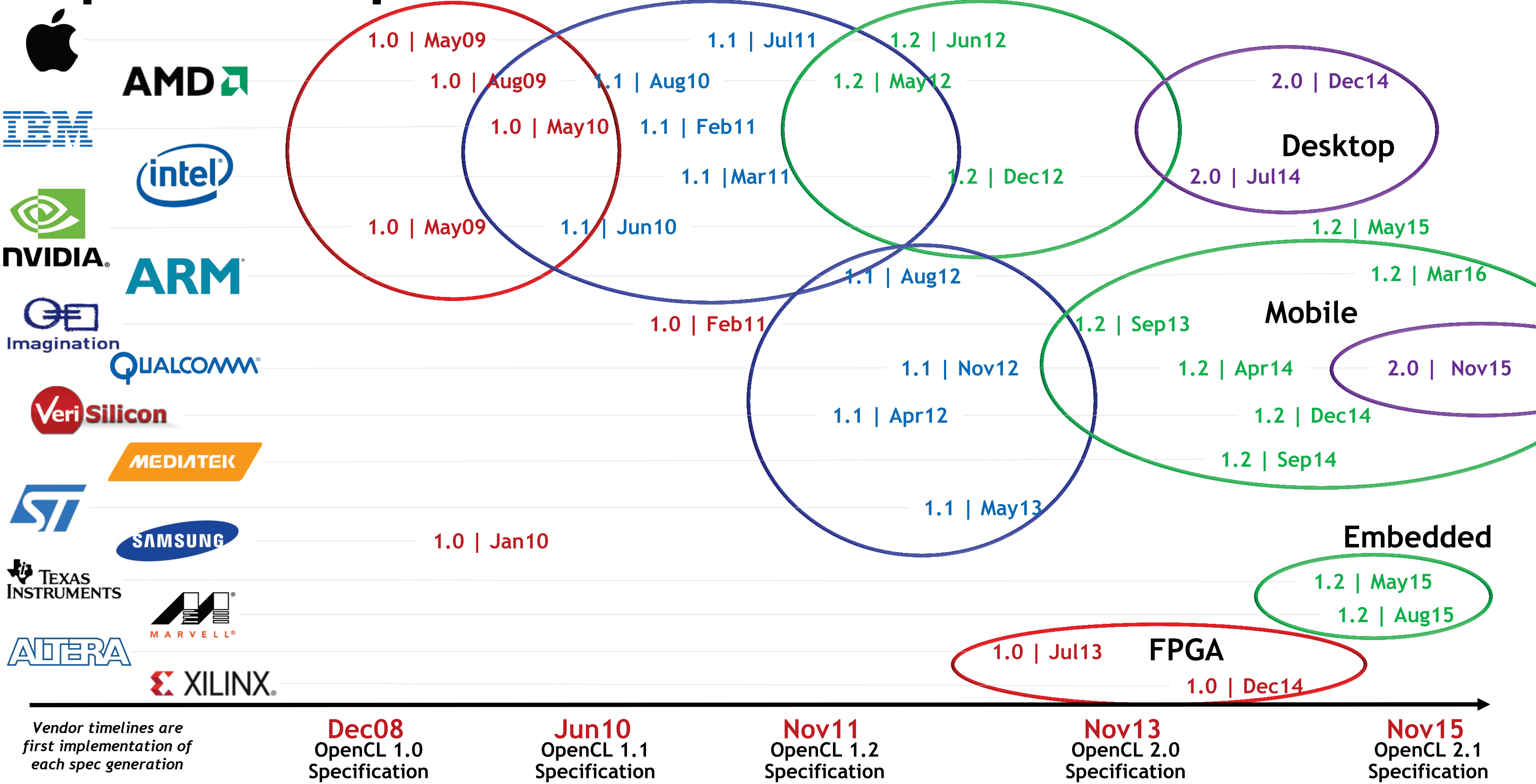
- **LLVM is an SDK, not a formally defined standard**
  - Khronos moved away from trying to use LLVM IR as a standard
  - Issues with versioning, metadata, etc.
- **But LLVM is a treasure chest of useful transforms**
  - SPIR-V tools can encapsulation and use LLVM to do useful SPIR-V transforms
- **SPIR-V 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

'Rendezvous' format for interchange

Native expression of graphics and parallel functionality for Khronos APIs



OpenCL Implementations



OpenCL at a Crossroads

CUDA, NVIDIA Shipping 1.2,
Lack of libraries

Lack of Tools
'Too complex to program'
Performance portability is hard

HPC, SciViz, Datacenter

Use case: Numerical Simulation, Virtualization

Roadmap: enhanced streaming processing, enhanced library support

Desktop

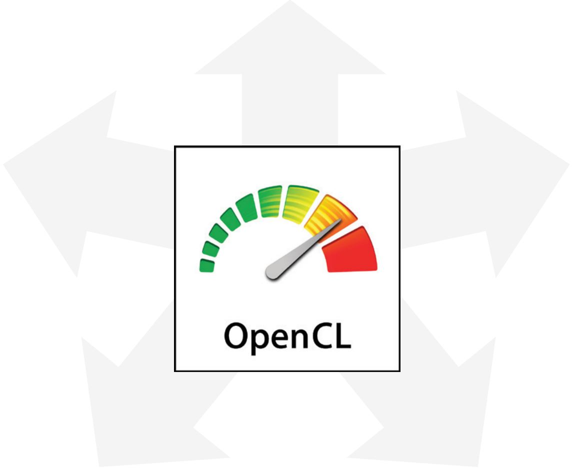
Use cases: Video and Image Processing, Gaming Compute

Roadmap: Vulkan interop, arbitrary precision for increased performance, pre-emption, Collective Programming and improved execution model

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++



CUDA, NVIDIA Shipping 1.2 Apple Metal

Mobile

Use case: Photo and Vision Processing

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

Embedded

Use cases: Signal and Pixel Processing

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

RenderScript confusion on Android, Apple Metal

* Roadmap topics in discussion

The Universal Struggle for Open Standards

Platforms
Idealized Universe =
Total content lock.
All commercially significant apps run on your platform and nowhere else

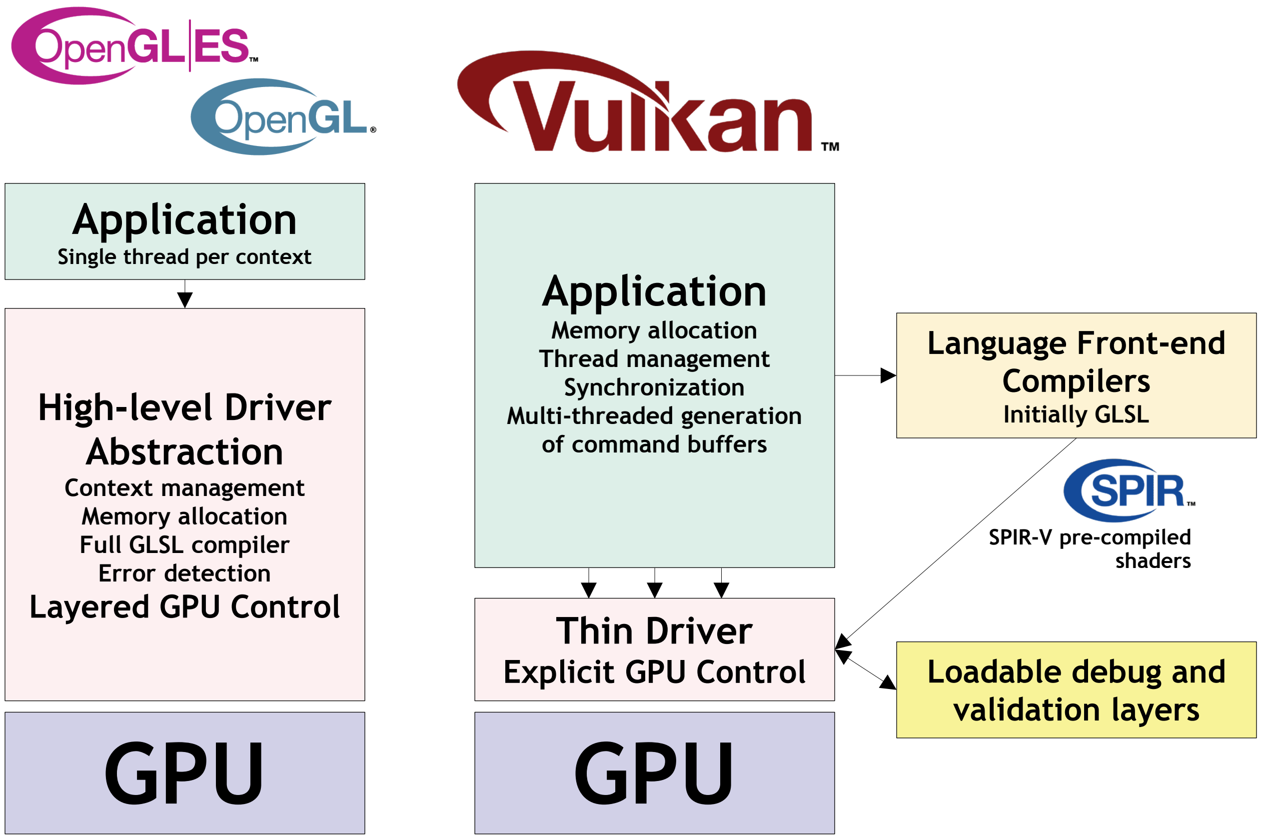
Independent Hardware and Software Vendors
Idealized Universe =
zero cost to monetize apps and processors across all platforms

Proprietary Solution Providers
Idealized Universe =
single viable solution.
All platforms and applications use your solution and nothing else

Effective Open Standard Strategies

1. Create joint investment in a solution that is too expensive for any one company to develop themselves
2. Create enough momentum that companies gain more content than they lose by supporting an open standard

Vulkan Explicit GPU Control



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

Vulkan Benefits

Resource management in app code:
Less hitches and surprises

Simpler drivers:
Improved efficiency/performance
Reduced CPU bottlenecks
Lower latency
Increased portability

Command Buffers:
Command creation can be multi-threaded
Multiple CPU cores increase performance

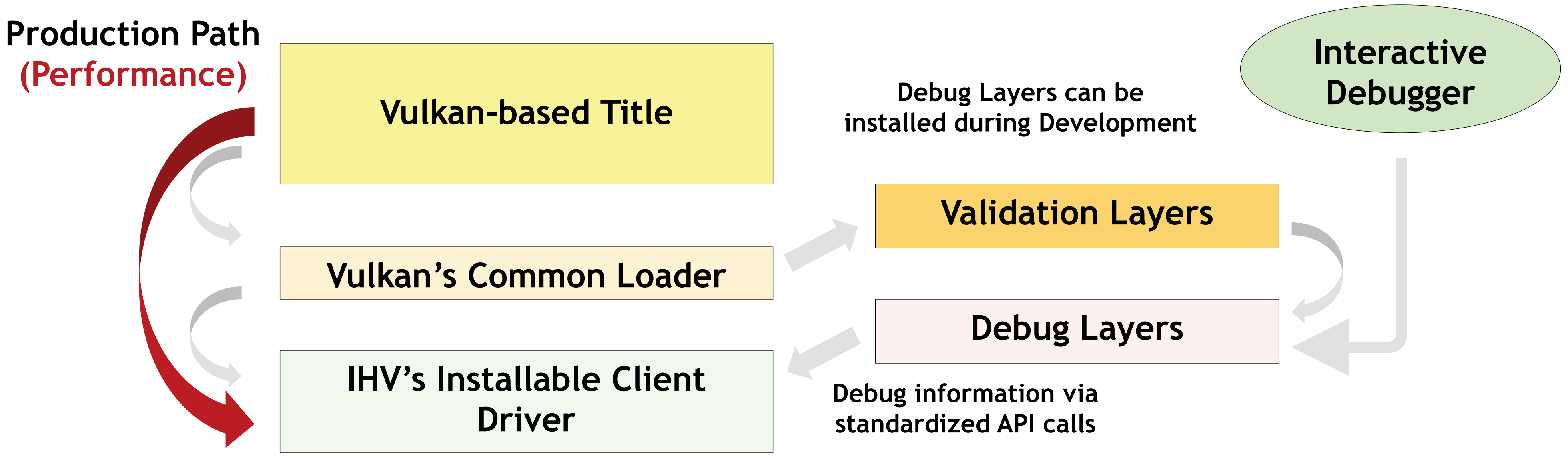
Graphics, compute and DMA queues:
Work dispatch flexibility

SPIR-V Pre-compiled Shaders:
No front-end compiler in driver
Future shading language flexibility

Loadable Layers
No error handling overhead in production code

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

Vulkan Genesis

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

Including an unprecedented level of participation from game engine developers

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

18 months

A high-energy working group effort

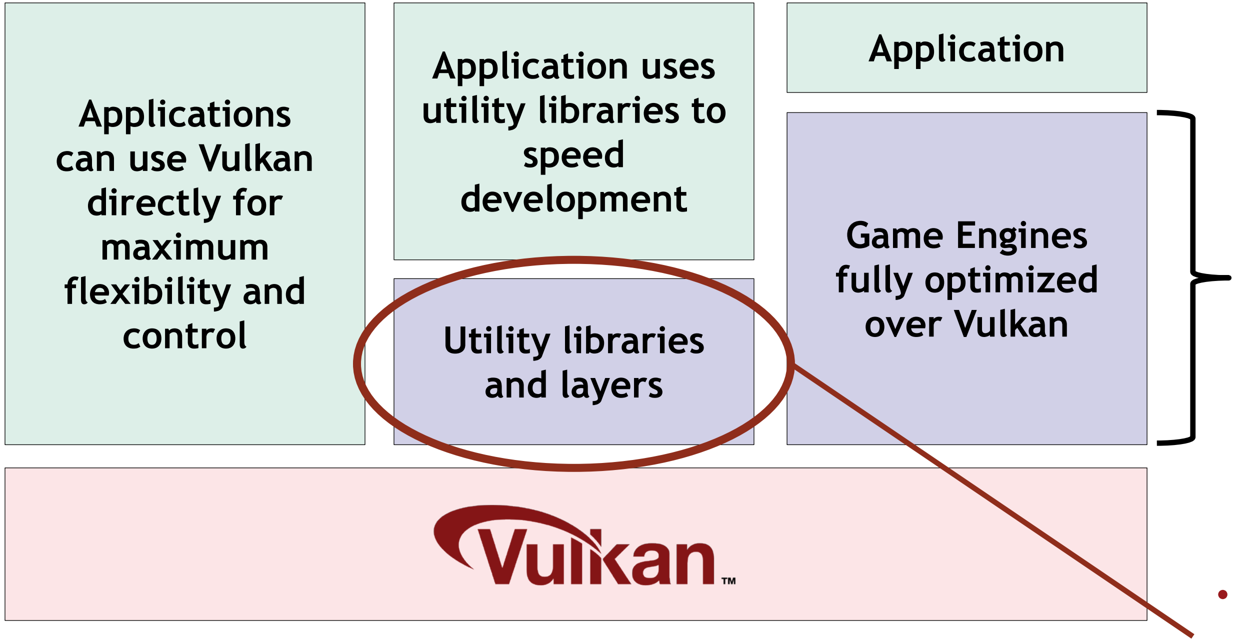
Vulkan

Khronos' first API 'hard launch' 16Feb16

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

Vulkan Working Group Participants

The Secret to Performance Portability



Applications using game engines will automatically benefit from Vulkan's enhanced performance

OXIDE · LUCASFILM Ltd · BLIZZARD ENTERTAINMENT · EA · unity · EPIC GAMES · VALVE

Rich Area for Innovation

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

Similar ecosystem dynamic as WebGL

A widely pervasive, powerful, flexible foundation layer enables diverse middleware tools and libraries

KHRONOS GROUP

Add Compute to Vulkan? In Discussion…

Desktop
Use cases: Video and Image Processing, Gaming Compute
Roadmap: Vulkan interop, arbitrary precision for increased performance, pre-emption, collective programming and improved execution model

HPC, SciViz, Datacenter
Use case: Numerical Simulation, 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++

Vulkan Compute?
Gaming Compute, Pixel Processing, Inference
Fine grain graphics and compute (no interop needed)
SPIR-V for shading language flexibility - C/C++
Low-latency, fine grain run-time
Google Android adoption
Competes well with Metal (=C++/OpenCL 1.2)
Roadmap: arbitrary precision, SVM, dynamic parallelism, pre-emption and QoS scheduling

OpenCL

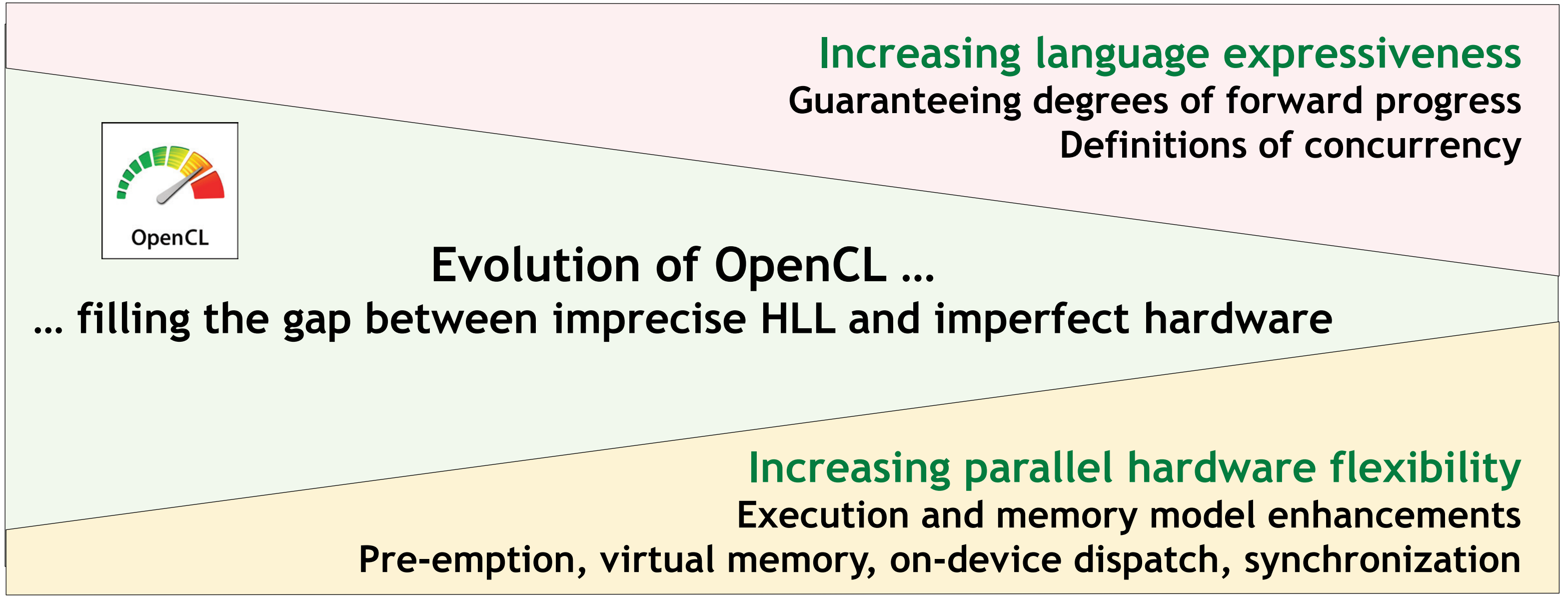
Embedded
Use cases: Signal and Pixel Processing
Roadmap: arbitrary precision for power efficiency, hard real-time scheduling, asynch DMA

Mobile
Use case: Photo and Vision Processing
Roadmap: arbitrary precision for inference engine and pixel processing efficiency, pre-emption and QoS scheduling for power efficiency

Vulkan Lessons
1. Engine developer insights were essential during design
2. Engine prototyping during design was essential during design
3. Open sourcing tests, tools, specs drives deeper community engagement
4. Explicit API - supports strong middleware ecosystem

BUT its 'just' a GPU API - still need OpenCL!

Possible OpenCL Evolution



Should OpenCL evolve to focus on the things that ONLY OpenCL can do…

1. Enable low-level, explicit access to heterogeneous hardware - needed by languages and libraries
2. Provide efficient runtime coordination of tasks, resources, scheduling on target hardware
3. Leverage, synergize and co-exist with Vulkan compute - and learn from Vulkan …
4. Define feature sets so target hardware does not have to implement inappropriate functionality
5. Adopt layered tools architecture to drive tools momentum and decrease run-time overhead
6. Leave usability, portability and performance portability to higher levels in the ecosystem

Or what do YOU think?

Get Involved!

- **OpenCL is driving to new level of community engagement**
  - Learning from the Vulkan experience
  - We need to know what you need from OpenCL
  - IWOCL is the perfect opportunity to find out!
- **Any company or organization is welcome to join Khronos**
  - For a voice and a vote in any of these standards
  - www.khronos.org
- **If joining is not possible - ask about the OpenCL Advisory Panel**
  - Free of charge – enable design reviews and contributions
- **Neil Trevett**
  - ntrevett@nvidia.com
  - @neilt3d