OpenCL 1.0 to 2.2 : a quick survey

Massimo Coppola
09/05/2018

Source material taken from Khronos group
https://www.khronos.org/
Original presentations were held at several events during 2013—2017

2013

UNIVERSITÀ DI PISA

ISTITUTO DI SCIENZA E TECNOLOGIE DELL'INFORMAZIONE "A. FAEDO"

HPC laboratory

OpenCL Introduction

Neil Trevett
Vice President NVIDIA, President Khronos
OpenCL Working Group Chair

Give us YOUR Feedback!

- **Full OpenCL 2.0 Documentation available**
  - Final Specification
  - Header files
  - Reference Card
  - Online Reference pages
- **OpenCL Registry contains all specifications**
  - www.khronos.org/registry/cl/
- **Open Resources Area**
  - Community submitted resources
  - http://www.khronos.org/opencl/resources
- **Public Forum and Bugzilla is open for comments**
  - All feedback welcome!

Resources

Commercial and Open Source Implementations

Frameworks & Libraries

Tutorials, Technical Whitepapers and How to Guides

OpenCL Presentations in This Session

- **OpenCL 2.0 Overview**
  - Allen Hux, Intel
- **Accelerated Science - use of OpenCL in Land Down Under**
  - Tomasz Bednarz, CSIRO
  - Sydney Khronos Chapter Leader

KHRONOS GROUP

OpenCL 2.0 Overview

Allen Hux
Intel Corporation

© Copyright Khronos Group 2013 - Page 1

Goals

- Enable New Programming Patterns
- Performance Improvements
- Well-defined Execution & Memory Model
- Improve CL / GL sharing

KHRONOS GROUP

Shared Virtual Memory

- **In OpenCL 1.2 buffer objects can only be passed as kernel arguments**
- **Buffer object described as pointer to type in kernel**
- **Restrictions**
  - Pass a pointer + offset as argument value
  - Store pointers in buffer object(s)
- **Why?**
  - Host and OpenCL device may not share the same virtual address space
  - No guarantee that the same virtual address will be used for a kernel argument across multiple enqueues

KHRONOS GROUP

Shared Virtual Memory

- clSVMAlloc - allocates a shared virtual memory buffer
  - Specify size in bytes
  - Specify usage information
  - Optional alignment value
- SVM pointer can be shared by the host and OpenCL device
- Examples

```
clSVMAlloc(ctx, CL_MEM_READ_WRITE, 1024 * 1024, 0)

clSVMAlloc(ctx, CL_MEM_READ_ONLY, 1024 * 1024, sizeof(cl_float4))
```

- Free SVM buffers
  - clEnqueueSVMFree, clSVMFree

KHRONOS GROUP

Shared Virtual Memory

- clSetKernelArgSVMPointer
  - SVM pointers as kernel arguments
  - A SVM pointer
  - A SVM pointer + offset

```
kernel void
vec_add(float *src, float *dst)
{
    size_t id = get_global_id(0);
    dst[id] += src[id];
}
```

```
// allocating SVM pointers
cl_float *src = (cl_float *)clSVMAlloc(ctx, CL_MEM_READ_ONLY, size, 0);
cl_float *dst = (cl_float *)clSVMAlloc(ctx, CL_MEM_READ_WRITE, size, 0);

// Passing SVM pointers as arguments
clSetKernelArgSVMPointer(vec_add_kernel, 0, src);
clSetKernelArgSVMPointer(vec_add_kernel, 1, dst);

// Passing SVM pointer + offset as arguments
clSetKernelArgSVMPointer(vec_add_kernel, 0, src + offset);
clSetKernelArgSVMPointer(vec_add_kernel, 1, dst + offset);
```

KHRONOS GROUP

Shared Virtual Memory

- clSetKernelExecInfo
  - Passing SVM pointers in other SVM pointers or buffer objects

```
// allocating SVM pointers
my_info_t *pA = (my_info_t *)clSVMAlloc(ctx,
          CL_MEM_READ_ONLY, sizeof(my_info_t), 0);
pA->pB = (cl_float *)clSVMAlloc(ctx,
          CL_MEM_READ_WRITE, size, 0);

// Passing SVM pointers
clSetKernelArgSVMPointer(my_kernel, 0, pA);

clSetKernelExecInfo(my_kernel,
                    CL_KERNEL_EXEC_INFO_SVM_PTRS,
                    1 * sizeof(void *), &pA->pB);
```

```
typedef struct {
  …
  float *pB;
  …
} my_info_t;

kernel void
my_kernel(global my_info_t *pA, …)
{
   …
   do_stuff(pA->pB, …);
   …
}
```

KHRONOS GROUP

© Copyright Khronos Group 2013 - Page 6

Shared Virtual Memory

- **Three types of sharing**
  - Coarse-grained buffer sharing
  - Fine-grained buffer sharing
  - System sharing

KHRONOS GROUP

Shared Virtual Memory - Coarse & Fine Grained

- **SVM buffers allocated using clSVMAlloc**
- **Coarse grained sharing**
  - Memory consistency only guaranteed at synchronization points
  - Host still needs to use synchronization APIs to update data
    - clEnqueueSVMMap / clEnqueueSVMUnmap or event callbacks
    - Memory consistency is at a buffer level
  - Allows sharing of pointers between host and OpenCL device
- **Fine grained sharing**
  - No synchronization needed between host and OpenCL device
    - Host and device can update data in buffer concurrently
    - Memory consistency using C11 atomics and synchronization operations
  - Optional Feature

KHRONOS GROUP

© Copyright Khronos Group 2013

Shared Virtual Memory - System Sharing

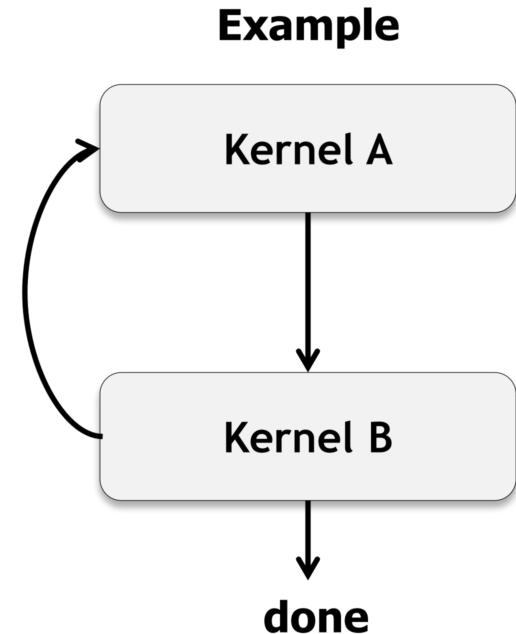
- **Can directly use any pointer allocated on the host**
  - No OpenCL APIs needed to allocate SVM buffers
- **Both host and OpenCL device can update data using C11 atomics and synchronization functions**
- **Optional Feature**

KHRONOS GROUP

© Copyright Khronos Group 2013 -

Nested Parallelism

- In OpenCL 1.2 only the host can enqueue kernels
- Iterative algorithm example
  - kernel A queues kernel B
  - kernel B decides to queue kernel A again
- Requires host - device interaction and for the host to wait for kernels to finish execution
  - Can use callbacks to avoid waiting for kernels to finish but still overhead
- A very simple but extremely common nested parallelism example



KHRONOS GROUP

Nested Parallelism

- **Allow a device to queue kernels to itself**
  - Allow a work-item(s) to queue kernels
- **Use similar approach to how host queues commands**
  - Queues and Events
  - Functions that queue kernels and other commands
  - Event and Profiling functions

KHRONOS GROUP

© Copyright Khronos Group 2013

Nested Parallelism

- Use clang Blocks to describe kernel to queue

```
kernel void my_func(global int *a, global int *b)
{
    …
    void (^my_block_A)(void) =
            ^{
                size_t id = get_global_id(0);
                b[id] += a[id];
             };

   enqueue_kernel(get_default_queue(),
                  CLK_ENQUEUE_FLAGS_WAIT_KERNEL,
                  ndrange_1D(…),
                  my_block_A);
}
```

KHRONOS GROUP

© Copyright Khronos Group 2013 - Page 12

Nested Parallelism

```
int enqueue_kernel(queue_t queue,
                   kernel_enqueue_flags_t flags,
                   const ndrange_t ndrange,
                   void (^block)())

int enqueue_kernel(queue_t queue,
                   kernel_enqueue_flags_t flags,
                   const ndrange_t ndrange,
                   uint num_events_in_wait_list,
                   const clk_event_t *event_wait_list,
                   clk_event_t *event_ret,
                   void (^block)())
```

KHRONOS GROUP

Nested Parallelism

- Queuing kernels with pointers to local address space as arguments

```
int enqueue_kernel(queue_t queue,
                   kernel_enqueue_flags_t flags,
                   const ndrange_t ndrange,
                   void (^block)(local void *, …), uint size0, …)

int enqueue_kernel(queue_t queue,
                   kernel_enqueue_flags_t flags,
                   const ndrange_t ndrange,
                   uint num_events_in_wait_list,
                   const clk_event_t *event_wait_list,
                   clk_event_t *event_ret,
                   void (^block)(local void *, …), uint size0, …)
```

Nested Parallelism

- Example showing queuing kernels with local address space arguments

```
void my_func_local_arg (global int *a, local int *lptr, …) { … }

kernel void my_func(global int *a, …)
{
    …
    uint local_mem_size = compute_local_mem_size(…);

    enqueue_kernel(get_default_queue(),
                   CLK_ENQUEUE_FLAGS_WAIT_KERNEL,
                   ndrange_1D(…),
                   ^(local int *p){my_func_local_arg(a, p, …);},
                   local_mem_size);
}
```

KHRONOS GROUP

Nested Parallelism

- **Specify when a child kernel can begin execution (pick one)**
  - Don't wait on parent
  - Wait for kernel to finish execution
  - Wait for work-group to finish execution

- **A kernel's execution status is complete**
  - when it has finished execution
  - *and* all its child kernels have finished execution

KHRONOS GROUP

© Copyright Khronos Group 2013 - Page 16

Nested Parallelism

- **Other Commands**
  - Queue a marker
- **Query Functions**
  - Get workgroup size for a block
- **Event Functions**
  - Retain & Release events
  - Create user event
  - Set user event status
  - Capture event profiling info
- **Helper Functions**
  - Get default queue
  - Return a 1D, 2D or 3D ND-range descriptor

KHRONOS GROUP

Generic Address Space

- In OpenCL 1.2, function arguments that are a pointer to a type must declare the address space of the memory region pointed to
- Many examples where developers want to use the same code but with pointers to different address spaces

```
void
my_func (local int *ptr, …)
{
    …
    foo(ptr, …);
    …
}
```

```
void
my_func (global int *ptr, …)
{
    …
    foo(ptr, …);
    …
}
```

- Above example is not supported in OpenCL 1.2
- Results in developers having to duplicate code

Generic Address Space

- OpenCL 2.0 no longer requires an address space qualifier for arguments to a function that are a pointer to a type
  - Except for kernel functions
- Generic address space assumed if no address space is specified
- Makes it really easy to write functions without having to worry about which address space arguments point to

```
void
my_func (int *ptr, …)
{
    …
}

kernel void
foo(global int *g_ptr, local int *l_ptr, …)
{
    …
    my_func(g_ptr, …);
    my_func(l_ptr, …);
}
```

KHRONOS GROUP

Generic Address Space – Casting Rules

- Implicit casts allowed from named to generic address space
- Explicit casts allowed from generic to named address space
- Cannot cast between constant and generic address spaces

```
kernel void foo()
{
    int *ptr;
    local int *lptr;
    global int *gptr;
    local int val = 55;

    ptr = gptr; // legal
    lptr = ptr; // illegal
    lptr = gptr; // illegal
    ptr = &val; // legal
    lptr = (local int *)ptr; // legal
}
```

© Copyright Khronos Group 2013

Generic Address Space - Built-in Functions

- **global gentype* to_global(const gentype*)**
  local gentype* to_local(const gentype *)
  private gentype* to_private(const gentype *)
  - Returns NULL if cannot cast
- **cl_mem_fence_flags get_fence(const void *ptr)**
  - Returns the memory fence flag value
  - Needed by work_group_barrier and mem_fence functions

KHRONOS GROUP

C11 Atomics

- **Implements a subset of the C11 atomic and synchronization operations**
  - Enable assignments in one work-item to be visible to others
- **Atomic operations**
  - loads & stores
  - exchange, compare & exchange
  - fetch and modify (add, sub, or, xor, and, min, max)
  - test and set, clear
- **Fence operation**
- **Atomic and Fence operations take**
  - Memory order
  - Memory scope
- **Operations are supported for global and local memory**

KHRONOS GROUP

© Copyright Khronos Group 2013 - Page 22

C11 Atomics

- **memory_order_relaxed**
  - Atomic operations with this memory order are not synchronization operations
  - Only guarantee atomicity
- **memory_order_acquire, memory_order_release, memory_order_acq_rel**
  - Atomic store in work-item A for variable M is tagged with memory_order_release
  - Atomic load in work-item B for same variable M is tagged with memory_order_acquire
  - Once the atomic load is completed work-item B is guaranteed to see everything work-item A wrote to memory before atomic store
  - Synchronization is only guaranteed between work-items releasing and acquiring the same atomic variable
- **memory_order_seq_cst**
  - Same as memory_order_acq_rel, and
  - A single total order exists in which all work-items observe all modifications

KHRONOS GROUP

C11 Atomics

- **Memory scope - specifies scope of memory ordering constraints**
  - Work-items in a work-group
  - Work-items of a kernel executing on a device
  - Work-items of a kernel & host threads executing across devices and host
    - For shared virtual memory

KHRONOS GROUP

C11 Atomics

- Supported Atomic Types
  - atomic_int, atomic_uint
  - atomic_long, atomic_ulong
  - atomic_float
  - atomic_double
  - atomic_intptr_t, atomic_uintptr_t, atomic_ptrdiff_t
  - atomic_size_t
  - atomic_flag
- Atomic types have the same size & representation as the non-atomic types except for atomic_flag
- Atomic functions must be lock-free

KHRONOS GROUP

Images

- **2D image from buffer**
  - GPUs have dedicated and fast hardware for texture addressing & filtering
  - Accessing a buffer as a 2D image allows us to use this hardware
  - Both buffer and 2D image use the same data storage
- **Reading & writing to an image in a kernel**
  - Declare images with the read_write qualifier
  - Use barrier between writes and reads by work-items to the image
    - work_group_barrier(CLK_IMAGE_MEM_FENCE)
  - Only sampler-less reads are supported

KHRONOS GROUP

© Copyright Khronos Group 2013

Images

- Writes to 3D images is now a core feature
- New image formats
  - sRGB
  - Depth
- Extended list of required image formats
- Improvements to CL / GL sharing
  - Multi-sampled GL textures
  - Mip-mapped GL textures

KHRONOS GROUP

© Copyright Khronos Group 2013

Pipes

- Memory objects that store data organized as a FIFO
- Kernels can read from or write to a pipe object
- Host can only create pipe objects

KHRONOS GROUP

© Copyright Khronos Group 2013 -

Pipes

- **Why introduce a pipe object?**
  - Allow vendors to implement dedicated hardware to support pipes
  - Read from and write to a pipe without requiring atomic operations to global memory
  - Enable producer – consumer relationships between kernels

KHRONOS GROUP

© Copyright Khronos Group 2013 -

Pipes - Read & Write Functions

- **Work-item read pipe functions**
  - Read a packet from a pipe
  - Read with reservation
    - Reserve n packets for reading
    - Read individual packets (identified by reservation ID and packet index)
    - Confirm that the reserved packets have been read
- **Work-item write pipe functions**
  - Write a packet to a pipe
  - Write with reservation
- **Work-group pipe functions**
  - Reserve and commit packets for reading / writing

KHRONOS GROUP

Other 2.0 Features

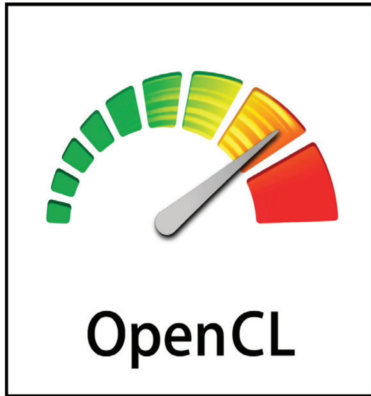
- Program scope variables
- Flexible work-groups
- New work-item functions
  - get_global_linear_id, get_local_linear_id
- Work-group functions
  - broadcast, reduction, vote (any & all), prefix sum
- Sub-groups
- Sharing with EGL images and events

KHRONOS GROUP

2015-2016

OpenCL 2.1 and SPIR-V 1.0 Launch

November 2015



OpenCL

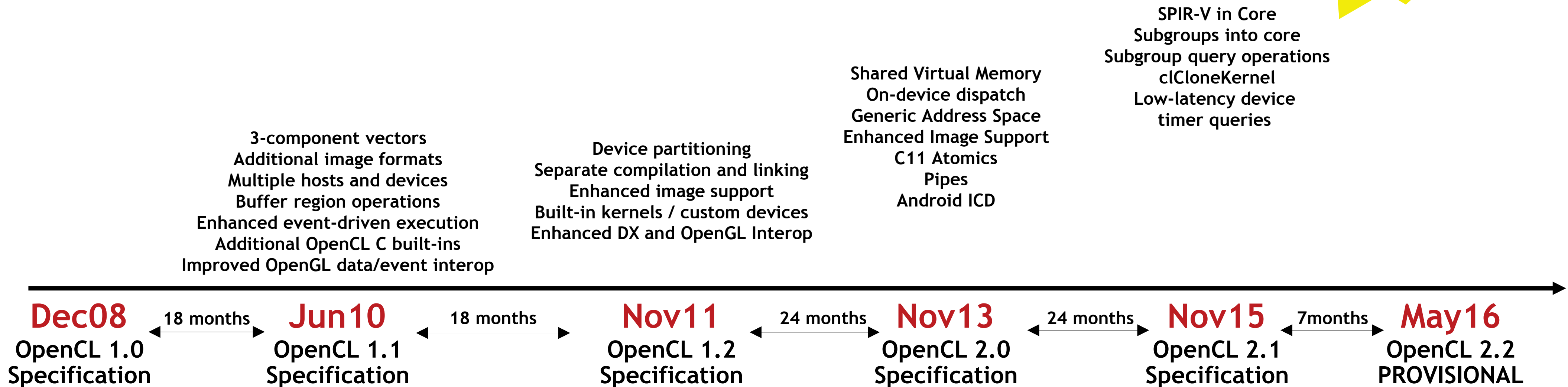
A State of the Union

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

Vienna, April 2016

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

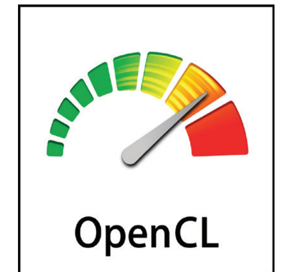


KHRONOS GROUP

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

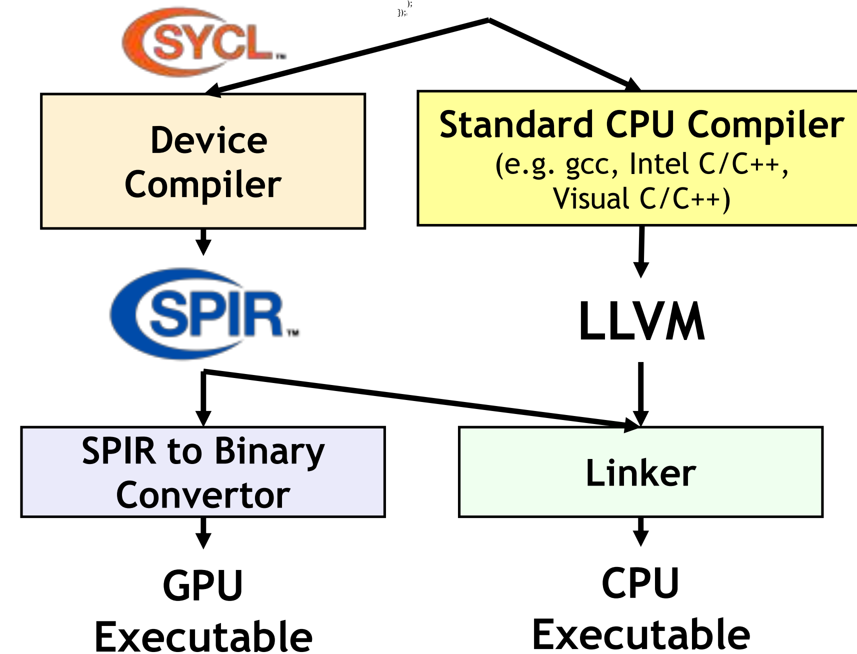


KHRONOS GROUP

© Copyright Khronos Group 2016

SYCL - Single Source Heterogeneous C++

- Pronounced 'sickle'
  - To go with 'spear' (SPIR)
- C++11 code for multiple OpenCL devices
  - Construct complex reusable algorithm templates using OpenCL for acceleration
- C++ templates contain host & device code
  - e.g. parallel_sort<MyType> (myData);
- Cross-toolchain as well as cross-platform
  - No language extensions - so standard C++ compilers can process SYCL source
- Device compilers enable SYCL on devices
  - Can have multiple device compilers linking into final executable



KHRONOS GROUP

SYCL Status and Benefits

- **SYCL 1.2 Final spec released**
  - At IWOCL in May 2014
- **Multiple implementations**
  - Including open source triSYCL from AMD
  - https://github.com/amd/triSYCL
- **Developers can move quickly into writing SYCL code**
  - Provides methods for dealing with targets that do not have OpenCL(yet!)
- **A fallback CPU implementation is debuggable!**
  - Using normal C++ debuggers
  - Profiling tools also work on CPU device
- **Huge bonus for productivity and adoption**
  - Cost of entry to use SYCL very low

SYCL is a practical, open, royalty-free standard to deliver high performance software on today's highly-parallel systems

KHRONOS GROUP

The Choice of SYCL 2.2 or OpenCL C++

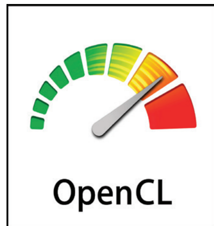
Developer Choice

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



Single-source C++

Programmer Familiarity

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

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

KHRONOS™ GROUP

SPIR-V Transforms the Language Ecosystem

- **First multi-API, intermediate language for parallel compute and graphics**
  - Native representation for Vulkan shader and OpenCL kernel source languages
  - https://www.khronos.org/registry/spir-v/papers/WhitePaper.pdf
- **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

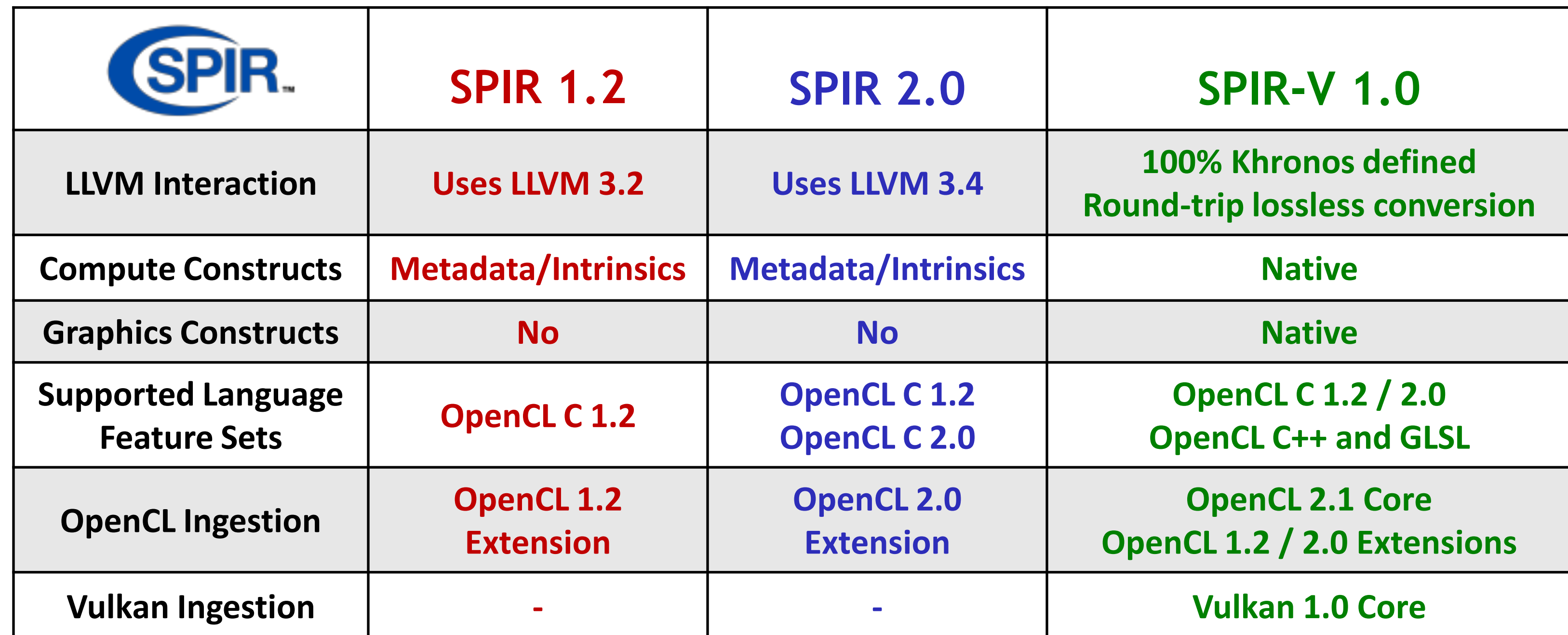
Multiple Developer Advantages
Same front-end compiler for multiple platforms
Reduces runtime kernel compilation time
Don't have to ship shader/kernel source code
Drivers are simpler and more reliable

Standard
Portable
Intermediate
Representation

KHRONOS GROUP

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 has open sourced SPIR-V <-> LLVM conversion tools to enable construction of flexible toolchains that use both intermediate languages



SPIR	SPIR 1.2	SPIR 2.0	SPIR-V 1.0
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 Sets	OpenCL C 1.2	OpenCL C 1.2 OpenCL C 2.0	OpenCL C 1.2 / 2.0 OpenCL C++ and GLSL
OpenCL Ingestion	OpenCL 1.2 Extension	OpenCL 2.0 Extension	OpenCL 2.1 Core OpenCL 1.2 / 2.0 Extensions
Vulkan Ingestion	-	-	Vulkan 1.0 Core

KHRONOS GROUP

SPIR-V Ecosystem

Khronos has open sourced these tools and translators

Khronos plans to open source these tools soon

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

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

GLSL

Third party kernel and shader Languages

OpenCL C

OpenCL C++

LLVM to SPIR-V Bi-directional Translator

LLVM

Other Intermediate Forms

SPIR-V Magic #: 0x07230203
SPIR-V Version 99
Builder's Magic #: 0x051a00BB
<id> bound is 50
0
OpMemoryModel
Logical
GLSL450
OpEntryPoint
Fragment shader
function <id> 4
OpTypeVoid
<id> is 2
OpTypeFunction
<id> is 3
return type <id> is 2
OpFunction
Result Type <id> is 2
Result <id> is 4
0
Function Type <id> is 3
SPIR

IHV Driver Runtimes

OpenCL OpenCL Vulkan Vulkan

KHRONOS GROUP

SPIR-V Open Source Community Activity

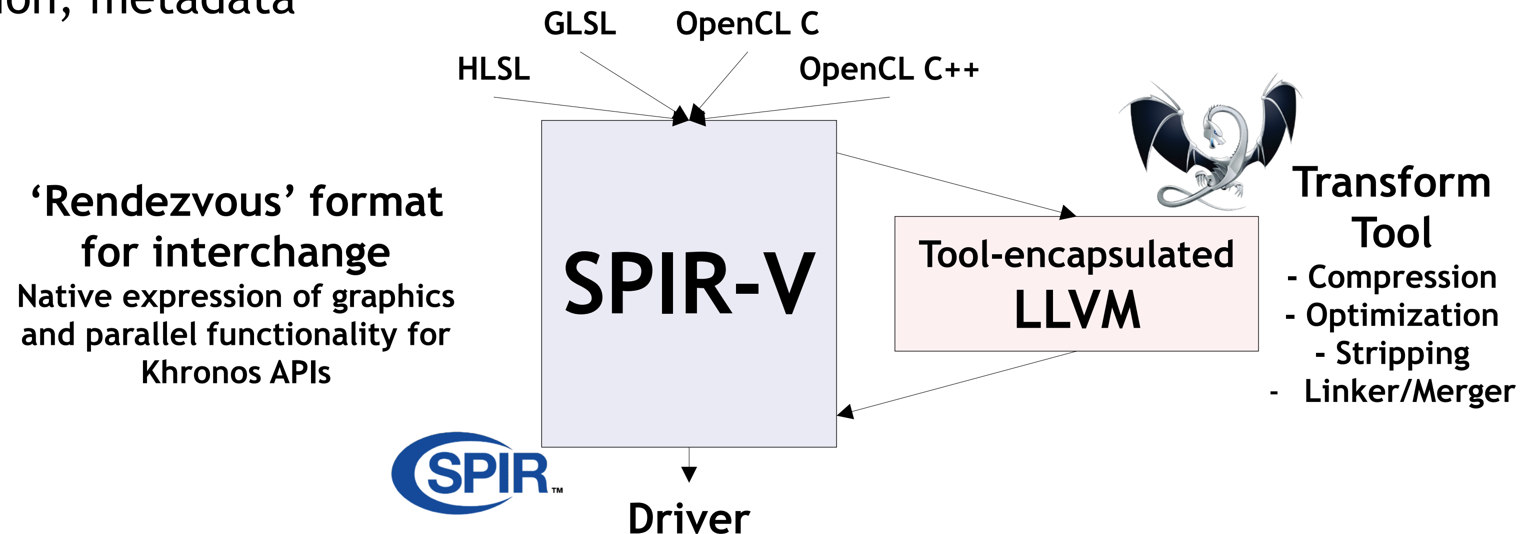
- **Python byte code to SPIR-V Convertor**
  - Write shaders or kernels in Python, Encode and decode SPIR-V in Python
  - Dis(Assembler) with high level human readable assembler syntax
- **.NET IL to SPIR-V Convertor**
  - Write and debug shaders or kernels using C# , SPIR-V interpreter
- **Shade SPIR-V virtual machine**
  - Test and debug SPIR-V binaries for binary correctness in human readable format
- **Otherside SPIR-V virtual machine**
  - Academic software rasterizer project to produce C code from SPIR-V
- **Rust (Dis)Assembler**
  - Encode and decode SPIR-V binaries in Rust
- **Go (Dis)Assembler**
  - Encode and decode SPIR-V in Go, SPIR-V represented in Go data structures
- **Haskell EDSL**
  - SPIR-V like language embedded in Haskell with significantly relaxed layout constraints
- **Lisp SPIR-V Specification**
  - Lisp readable SPIR-V specification
- **JSON SPIR-V specification**
  - Conversion of HTML SPIR-V specification to JSON format
- **This is just the start….**

KHRONOS GROUP

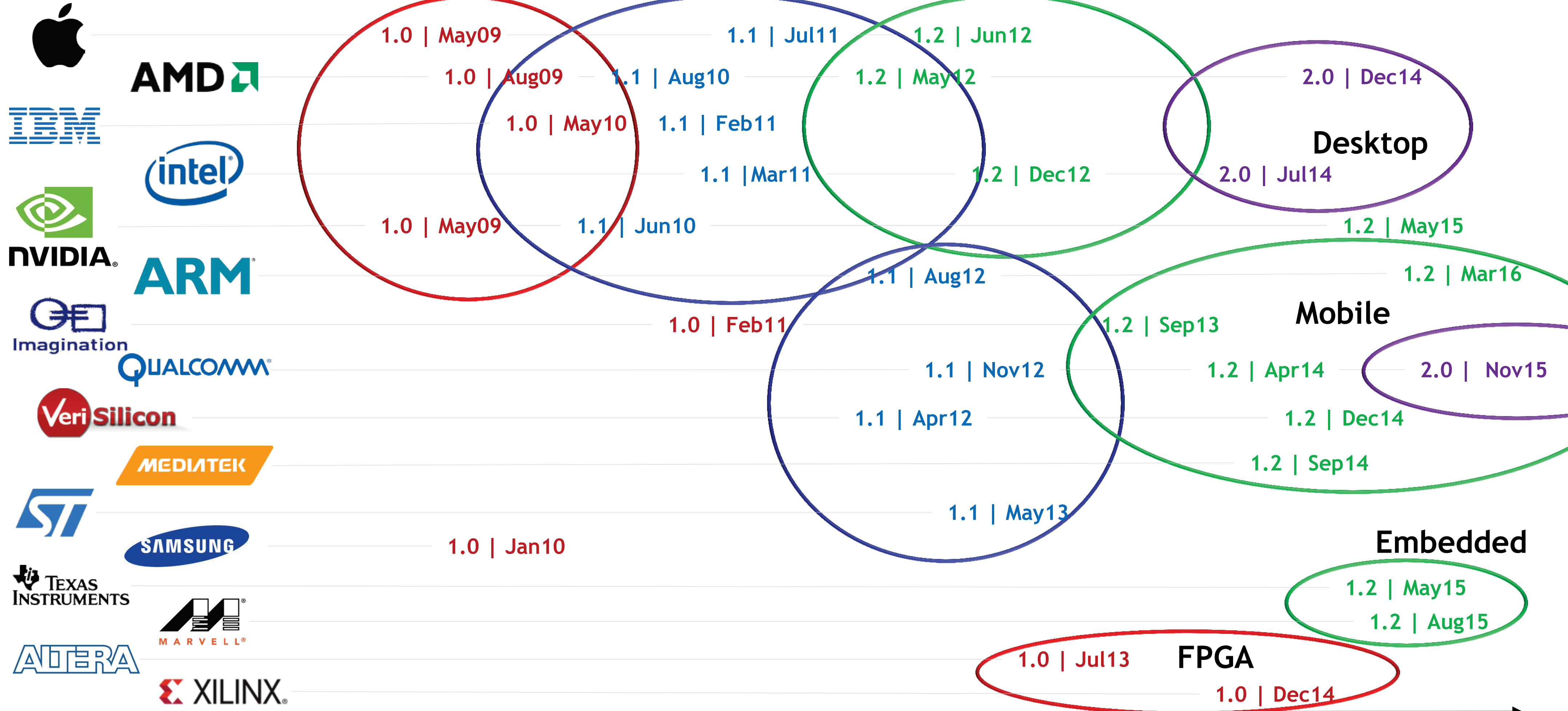
© Copyright Khronos Group 2015 - Page 13

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



OpenCL Implementations



© Copyright Khronos Group 2016 - Page 10

OpenCL at a Crossroads

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

CUDA, NVIDIA Shipping 1.2, Lack of libraries

Desktop

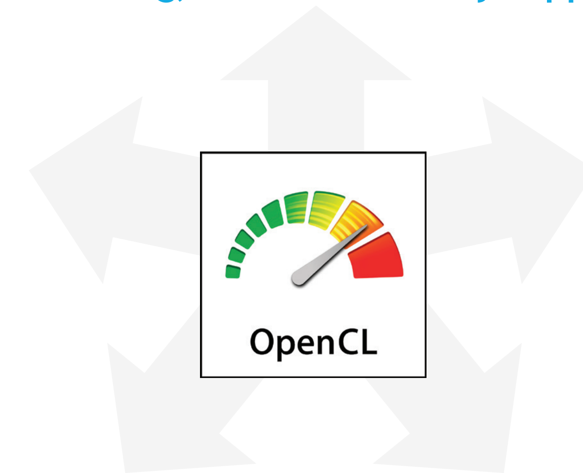
Use cases: Video and Image Processing, Gaming Compute

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

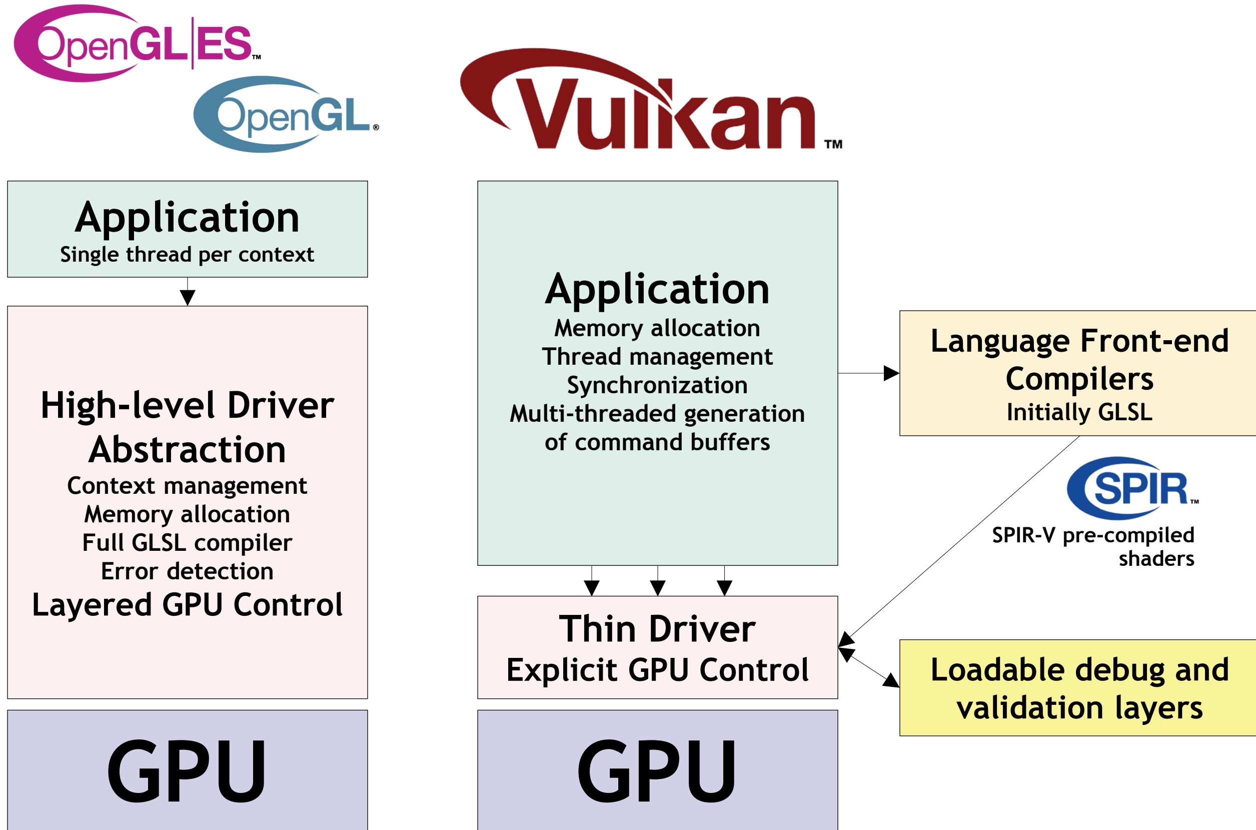
The Universal Struggle for Open Standards

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

KHRONOS GROUP

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

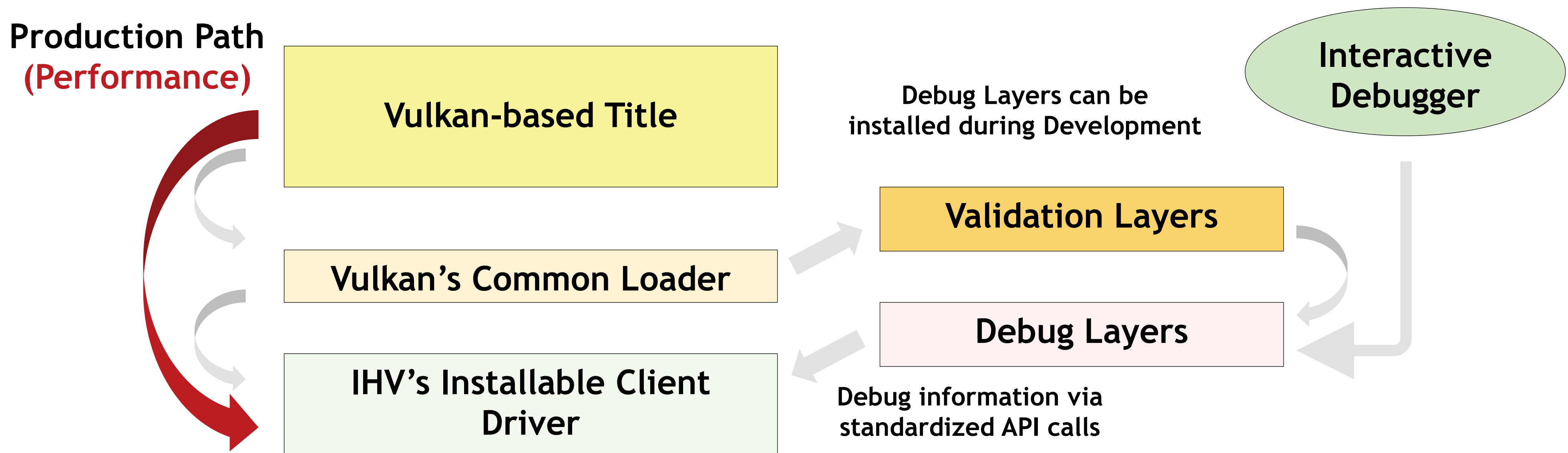
Loadable Layers
No error handling overhead in production code

KHRONOS GROUP

© Copyright Khronos Group 2016 -

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



KHRONOS GROUP

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 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

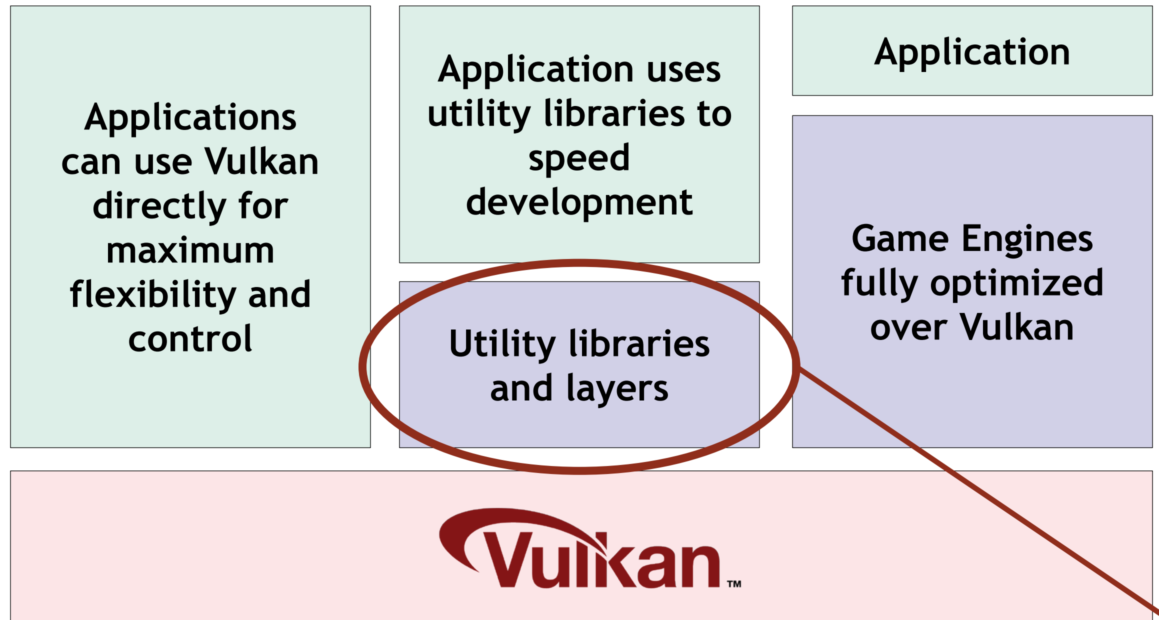
A high-energy working group effort

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

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

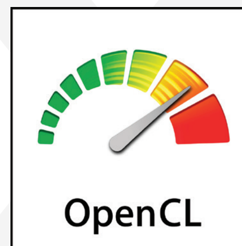
KHRONOS GROUP

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

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

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



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++

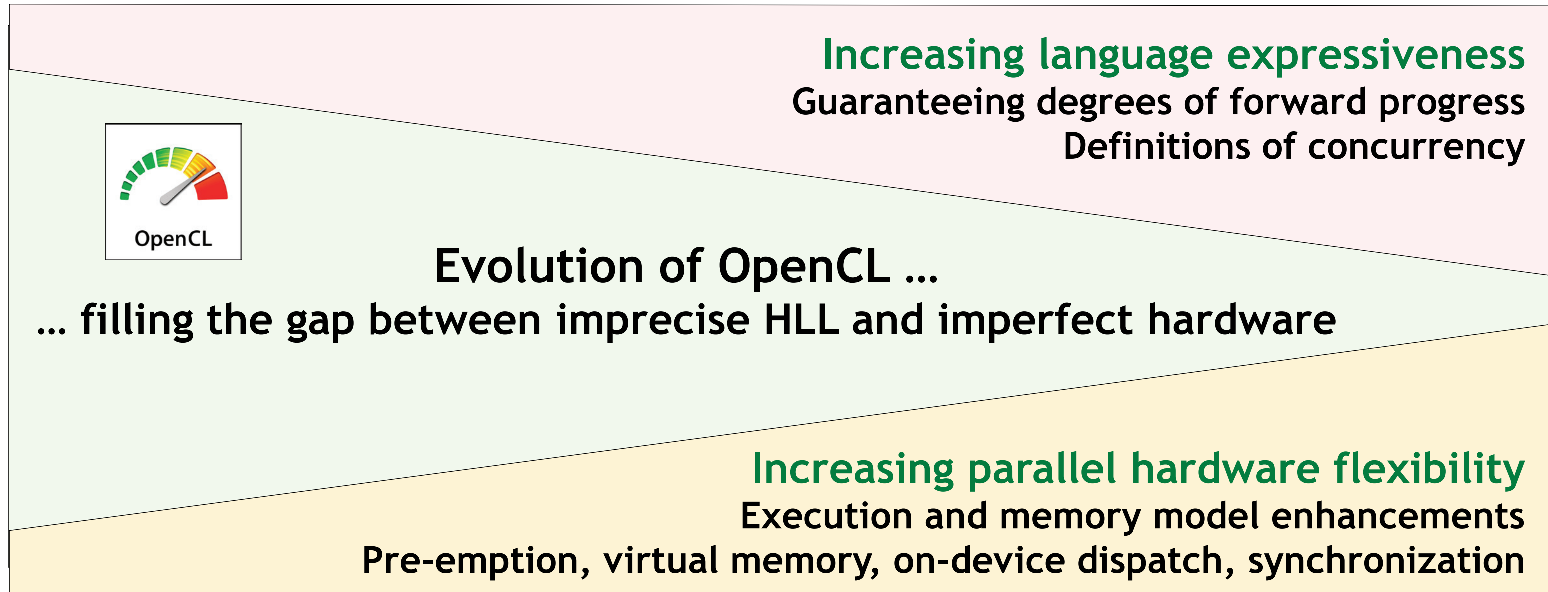
Embedded
Use cases: Signal and Pixel Processing
Roadmap: arbitrary precision for power efficiency, hard real-time scheduling, asynch DMA

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!

KHRONOS GROUP

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?

KHRONOS GROUP

© Copyright Khronos Group 2016

2017

ISTITUTO DI SCIENZA E TECNOLOGIE
DELL'INFORMAZIONE "A. FAEDO"

HPC laboratory