



# OpenCL 2.1 and SPIR-V 1.0 Launch

November 2015

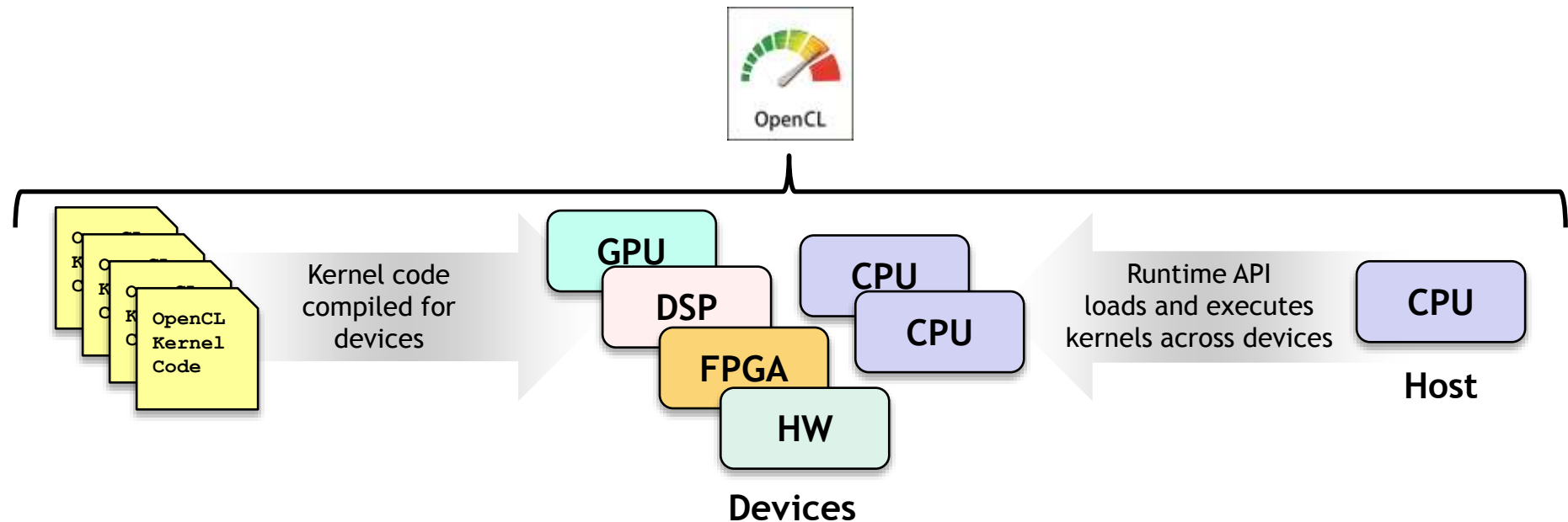
# OpenCL 2.1 and SPIR-V 1.0 Launch

- OpenCL 2.1 and SPIR-V 1.0 publicly launched
  - Supercomputing 2015, Austin, Monday 16th November
- OpenCL 2.1 brings SPIR-V 1.0 ingestion into core
  - [www.khronos.org/opencl/](http://www.khronos.org/opencl/)
- SPIR-V 1.0 intermediate language with native support for parallel compute
  - [www.khronos.org/spir/](http://www.khronos.org/spir/)
- SPIR-V 1.0 enables front-end language and framework innovation
  - Khronos releasing SPIR-V open source tools
  - SPIR-V will also be used in upcoming Vulkan graphics API



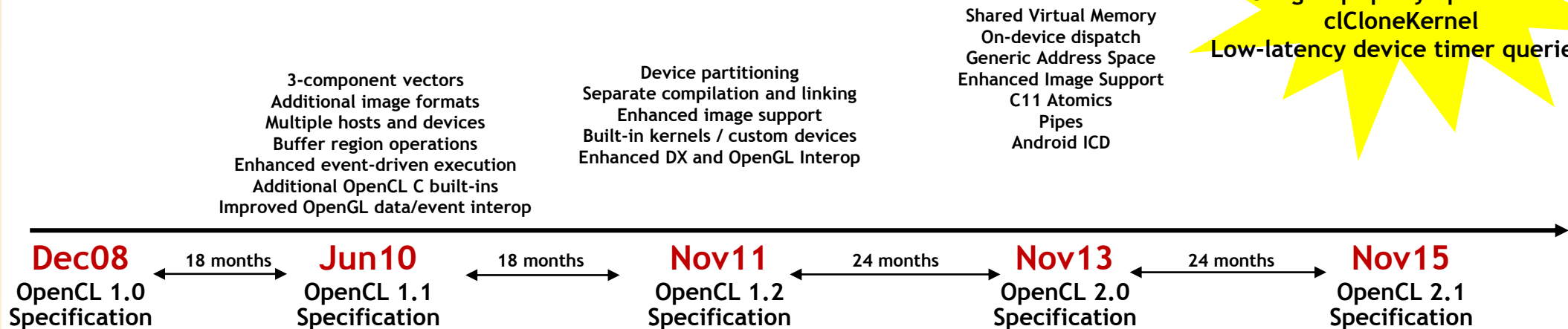
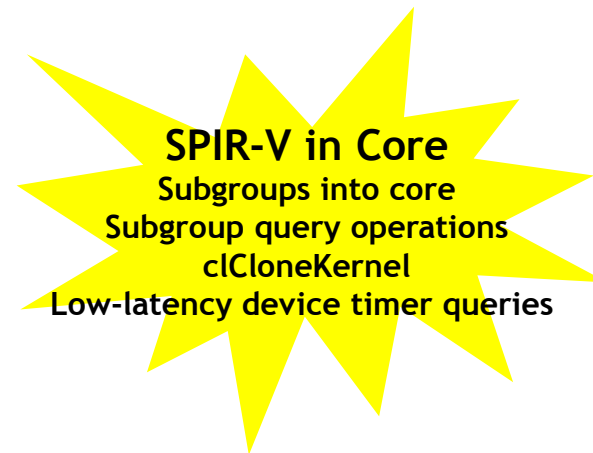
# OpenCL - Portable Heterogeneous Computing

- **OpenCL = Two APIs and Two Kernel languages**
  - C Platform Layer API to query, select and initialize compute devices
  - OpenCL C and (soon) OpenCL C++ kernel languages to write parallel code
  - C Runtime API to build and execute kernels across multiple devices
- **One code tree can be executed on CPUs, GPUs, DSPs, FPGA and hardware**
  - Dynamically balance work across available processors



# OpenCL 2.1 Released - November 2015

- Support for the SPIR-V 1.0 intermediate language in core
  - E.g. SPIR-V used to ingest from diverse language front-ends
  - OpenCL C ingestion still supported to preserve kernel code investment
- OpenCL API updates
  - E.g. subgroups and subgroup queries in core
- Runs on any OpenCL 2.0-capable hardware
  - Only driver update required



# OpenCL 2.1 API Enhancements

- **clCreateProgramWithIL**
  - SPIR-V support built-in to the runtime
- **Subgroup query operations**
  - Subgroups expose hardware threading in the core feature set
- **clCloneKernel enables copying of kernel objects and state**
  - Safe implementation of copy constructors in wrapper classes
- **Low-latency device timer queries**
  - Support alignment of profiling between device and host code
- **Priority and throttle hint extensions for queues**
  - Specify execution priority on a per-queue basis
- **Zero-size enqueue**
  - Zero-sized dispatches are valid from the host



# OpenCL C++ - Finalization Imminent

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

**Highly adaptive parallel software that delivers tuned performance across diverse platforms**



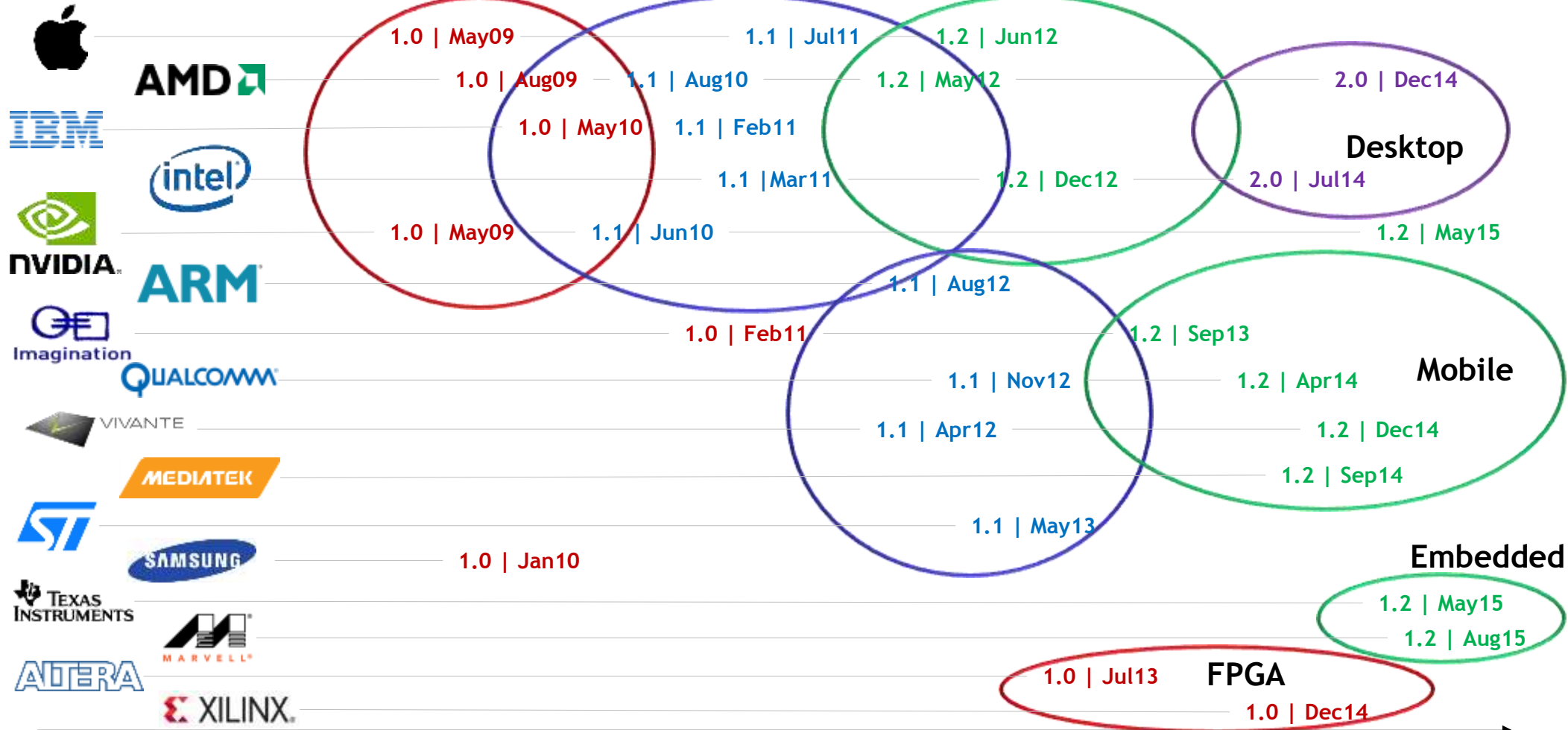
# OpenCL Accelerated Apps

- Desktop Apps
  - Imaging, video, vision
  - Design simulation
  - Neural net training and acceleration
- Embedded
  - Camera and photography pipelines
  - Embedded neural net acceleration
- Over 2000 open source projects use OpenCL
  - Sourceforge, Github, Google Code etc.
  - OpenCL implementations - Beignet, pocl
  - Imaging, video, vision, compression, crypto
- Benchmarks
  - PCMark 8 - video chat and edit
  - Basemark CL, CompuBench Desktop

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



# OpenCL Implementations

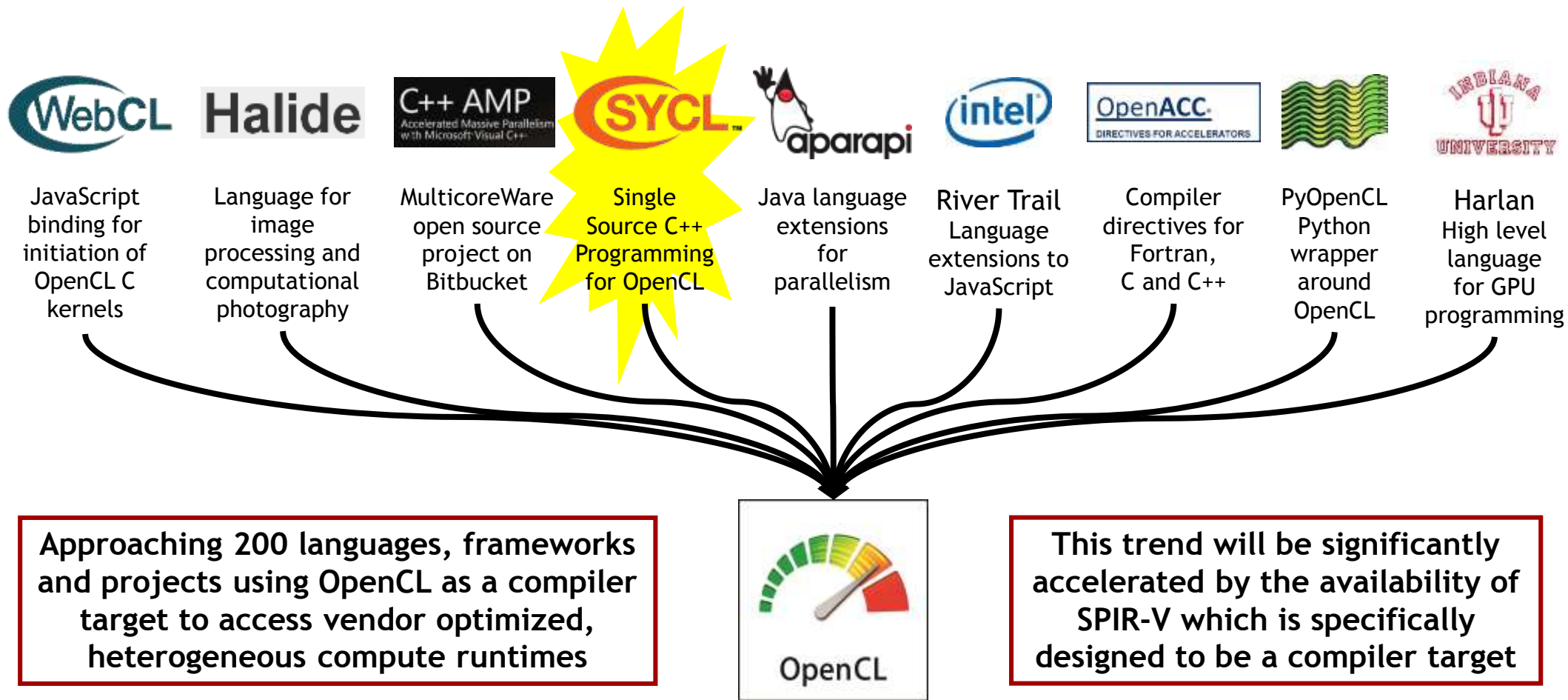


Vendor timelines are first implementation of each spec generation

**Dec08** OpenCL 1.0 Specification  
**Jun10** OpenCL 1.1 Specification  
**Nov11** OpenCL 1.2 Specification  
**Nov13** OpenCL 2.0 Specification  
**Mar15** OpenCL 2.1 Specification



# OpenCL as Parallel Language Backend



# 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



# 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 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

# Driving the SPIR-V Open Source Ecosystem

Khronos has open sourced these tools and translators

Khronos plans to open source these tools soon

OpenCL C    OpenCL C++

GLSL

Third party kernel and shader Languages

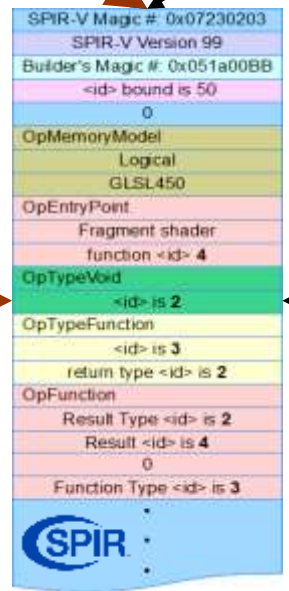
SPIR-V Tools

SPIR-V Validator

SPIR-V (Dis)Assembler

LLVM

LLVM to SPIR-V Bi-directional Translator



Other Intermediate Forms

- SPIR-V**
- 32-bit Word Stream
  - Extensible and easily parsed
  - Retains data object and control flow information for effective code generation and translation

IHV Driver Runtimes



# 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...**



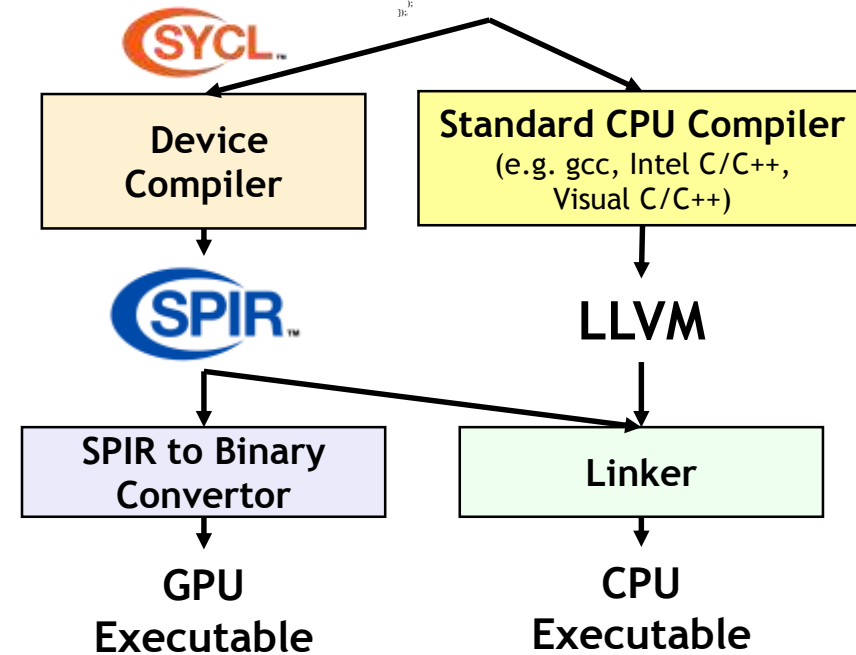
# SYCL - Single Source Heterogeneous C++

- 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

```
#include <CL/sycl.hpp>
```

```
int main ()  
{  
  // Device buffers  
  // Device buffers  
  buffer<float, 1> buf_array_a, range<1>(count);  
  buffer<float, 1> buf_array_b, range<1>(count);  
  buffer<float, 1> buf_array_c, range<1>(count);  
  buffer<float, 1> buf_array_d, range<1>(count);  
  queue myQueue;  
  myQueue.submit(B)(handler& cgh)  
  {  
    // Data accessors  
    auto a = buf_a.get_access<access::read>();  
    auto b = buf_b.get_access<access::read>();  
    auto c = buf_c.get_access<access::read>();  
    auto d = buf_d.get_access<access::write>();  
    // Kernel  
    cgh.parallel_for<class three_way_add<count, 1>()>(i)  
    {  
      d[i] = a[i] + b[i] + c[i];  
    }  
  }  
};
```

Single Standard C++  
Source File



# 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*

# OpenCL Ecosystem

## Implementers

Desktop/Mobile/Embedded/FPGA



Single Source C++ Programming



Core API and Language Specs



Portable Kernel Intermediate Language

## Working Group Members

Apps/Tools/Tests/Courseware

