OpenCL™

SYCL™

OpenACC

OpenMP®

ROCm ON

NVIDIA® CUDA®

Related Programming Models

07.06.2023

Recap

- CUDA programming model
  - C / C++
  - Fortran
- CUDA API
  - Runtime
  - Driver
- CUDA parallel computing platform
  - Architectures: Tesla, Fermi, Kepler, Maxwell, Pascal, Volta, Truing, Ampere, …
  - PTX ISA

NVIDIA only

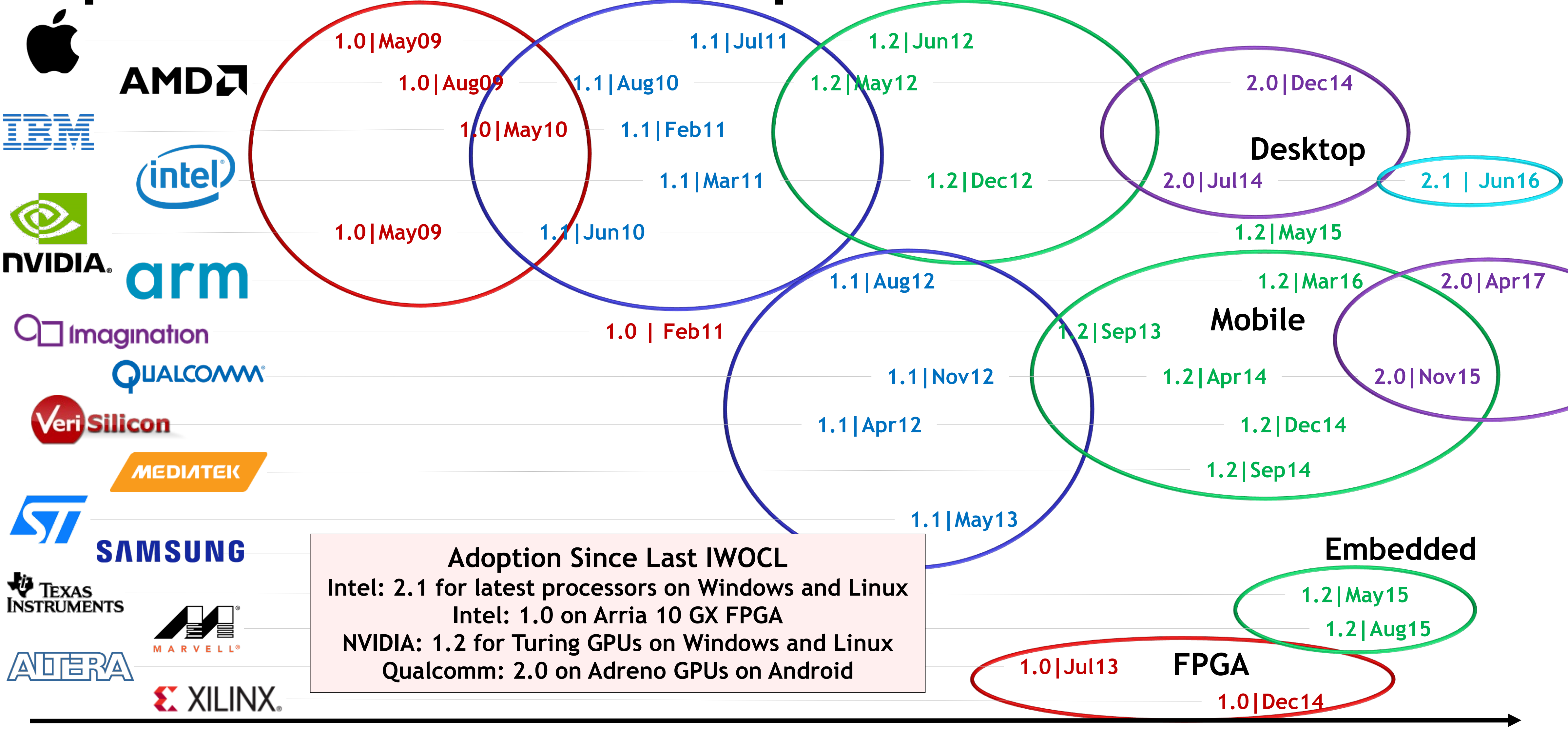
Related Programming Models

- Standards by the Khronos Group
  - OpenCL: cross-platform, parallel programming
  - SYCL: higher-level programming model for OpenCL
  - Vulkan: low-level graphics and compute shader API
- Standard by the HSA foundation
  - HSA (Heterogeneous System Architecture)
- Directive-based standards
  - OpenACC
  - OpenMP 4.0
- Vendor specific
  - C++ AMP (Accelerated Massive Parallelism, Microsoft) – deprecated
  - DPC++ (Data Parallel C++, Intel)
  - ROCm (Radeon Open Compute, AMD)
  - CUDA (Compute Unified Device Architecture, NVIDIA)

OpenCL

- Initiated by Apple (2008)
- Maintained by the Khronos Group
- Goal:
  - Cross-platform: desktop, embedded, HPC
  - Cross-device: CPU, GPU, FPGA, DSP, …
- Problem:
  - Slow adoption of new features
  - Functional, but no performance portability

OpenCL Conformant Implementations



Adoption Since Last IWOCL

Intel: 2.1 for latest processors on Windows and Linux
Intel: 1.0 on Arria 10 GX FPGA
NVIDIA: 1.2 for Turing GPUs on Windows and Linux
Qualcomm: 2.0 on Adreno GPUs on Android

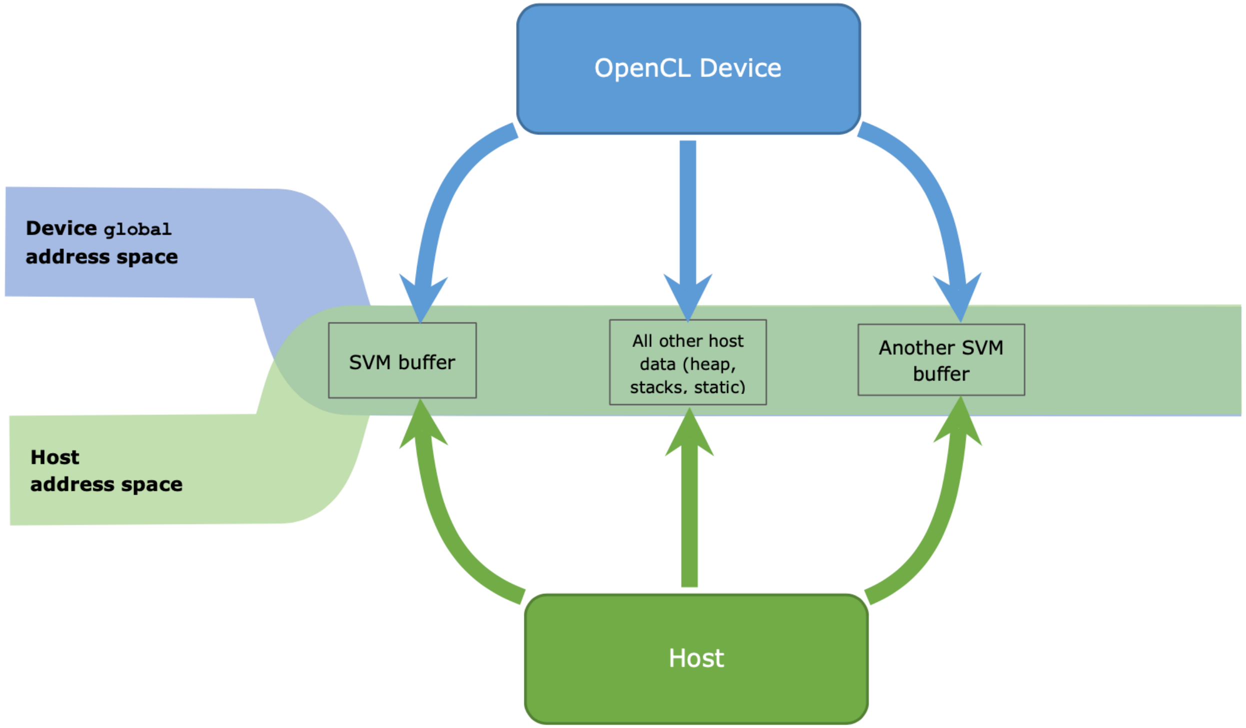
Vendor timelines are first conformant submission for each spec generation

Dec08	Jun10	Nov11	Nov13	Nov15
OpenCL 1.0 Specification	OpenCL 1.1 Specification	OpenCL 1.2 Specification	OpenCL 2.0 Specification	OpenCL 2.1 Specification

OpenCL

History

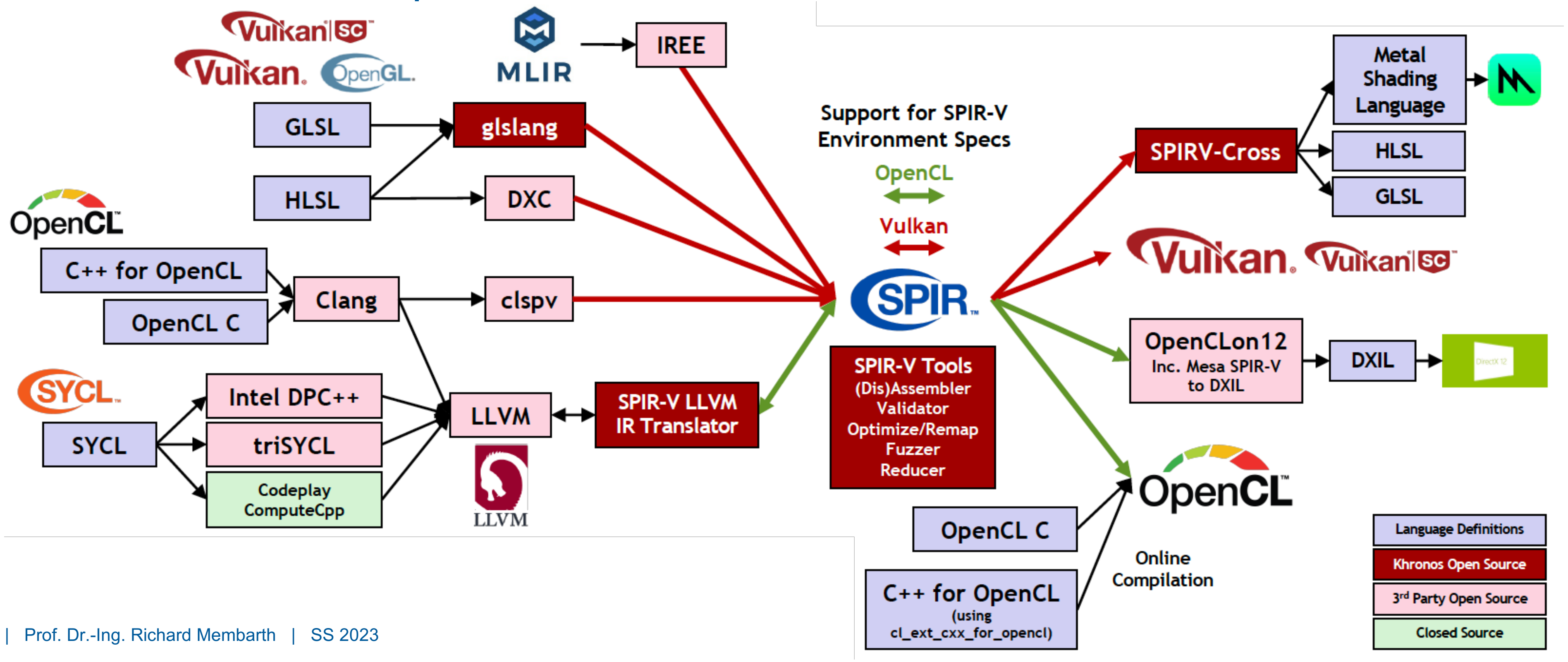
- OpenCL 1.2 (11/2011)
  - C99 based kernel language
  - Most prevalent standard
- OpenCL 2.0 (11/2013)
  - Shared virtual memory (SVM)
    - Coarse grained (required)
    - Fine grained (optional)
    - System (optional)
  - C11 atomics
    - Ordering/consistency: relaxed, acquire, release, acq_rel, seq_cst
    - Memory scope: work_item, work_group, device, all_svm_devices


OpenCL Device
Device global address space
SVM buffer
All other host data (heap, stacks, static)
Another SVM buffer
Host address space
Host

OpenCL

History

- OpenCL 2.1 (11/2015)
  - C++14 based kernel language (optional)
  - SPIR-V intermediate representation



OpenCL

History

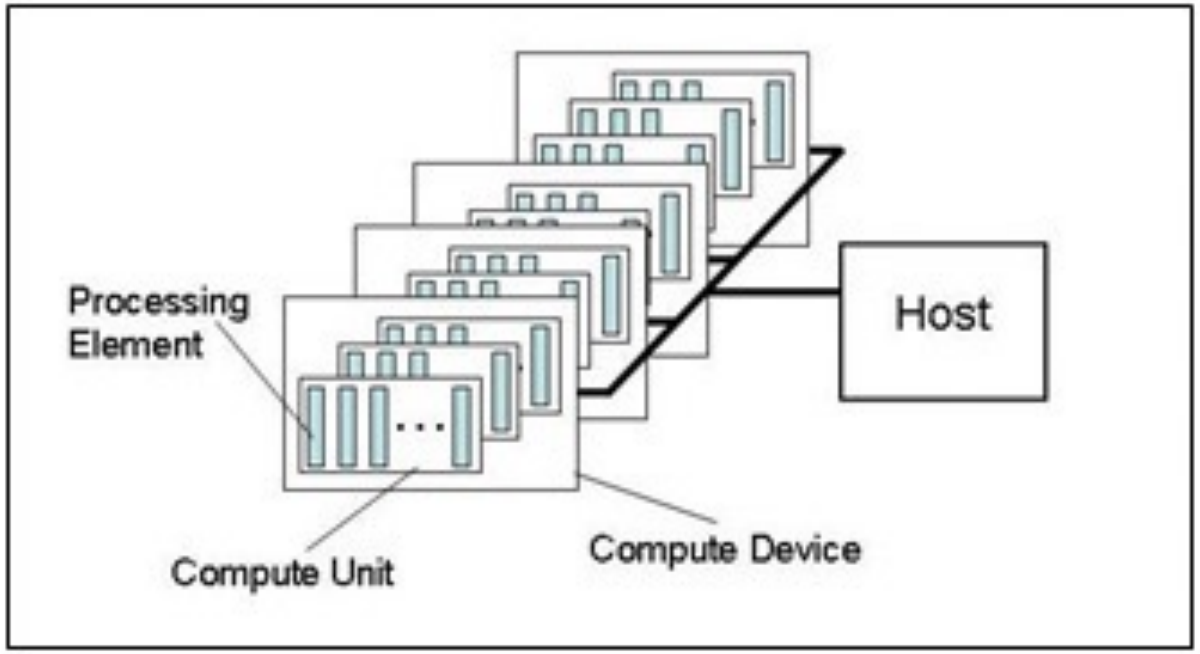
- OpenCL 2.2 (05/2017)
  - C++14 based kernel language (core)
- OpenCL 3.0 (09/2020)
  - OpenCL 1.2 mandatory
  - OpenCL 2.x features as optional modules

- Next: launch OpenCL kernels via Vulkan ?

OpenCL

Platform Model

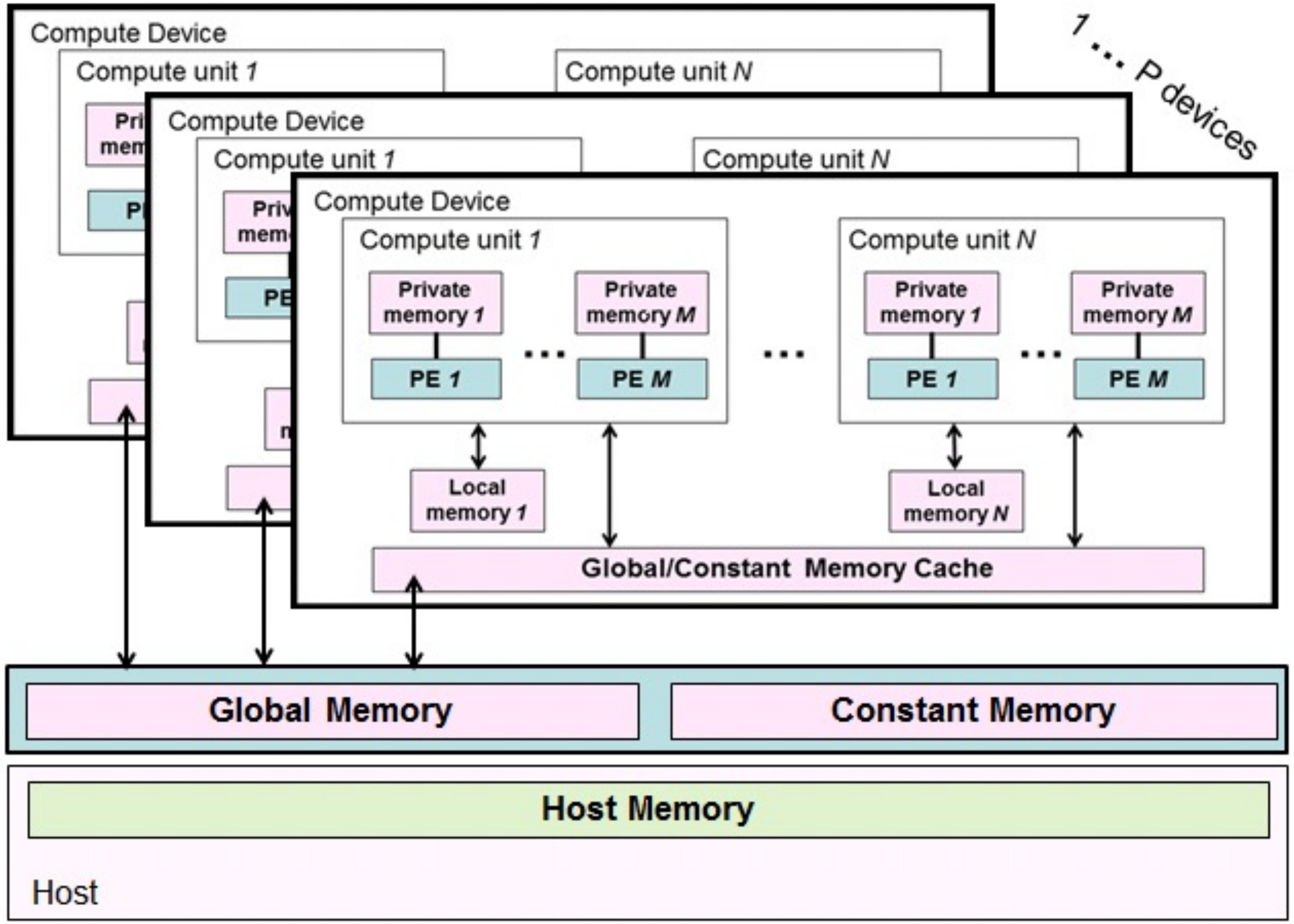
- Single host system
- Multiple compute devices
  - CPU, GPU, FPGA, …
- Compute unit → CUDA multiprocessor
- Processing element → CUDA core



OpenCL

Memory Model

- Similar to CUDA
  - Host memory
  - Global memory
  - Constant memory
  - Local memory
  - Private memory



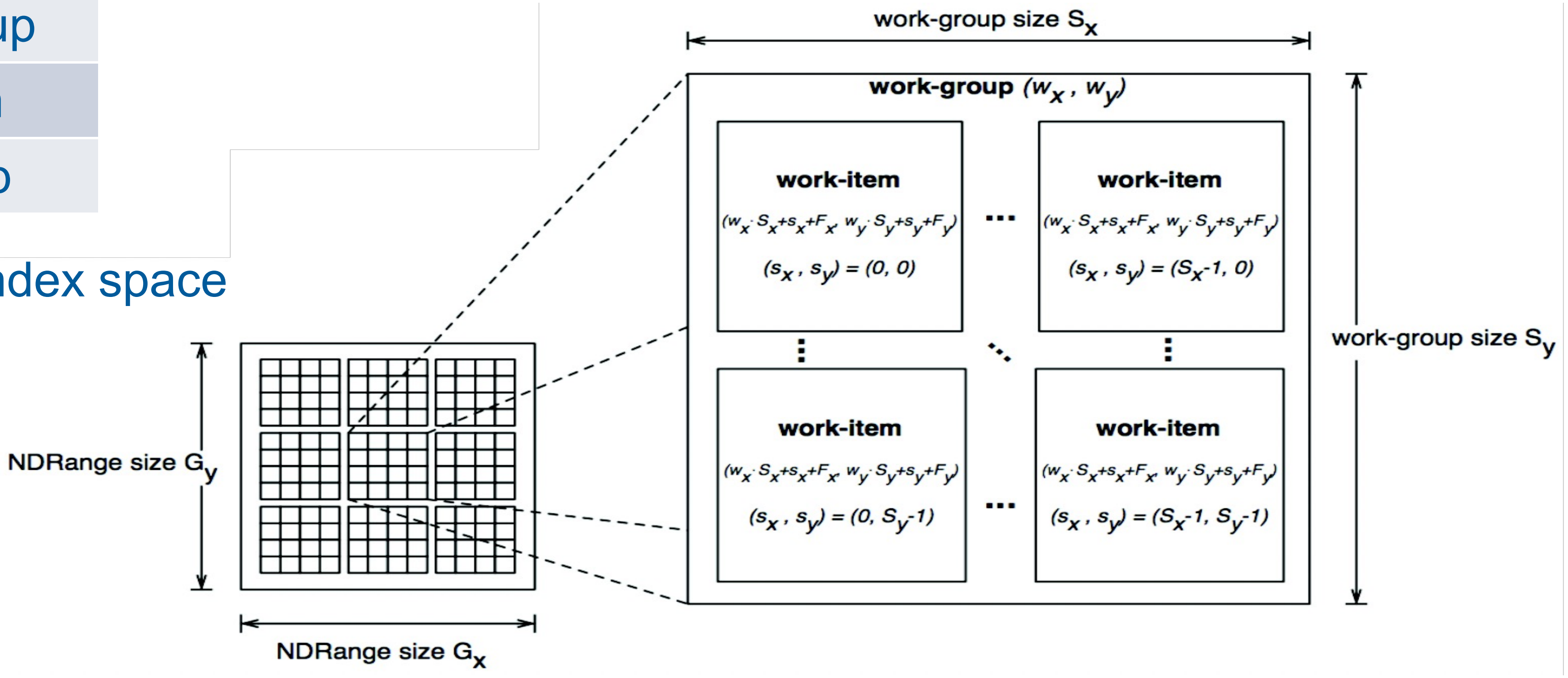
OpenCL

Execution Model

- Index space mapping similar to CUDA

CUDA	OpenCL
grid	NDRange
block	work-group
thread	work-item
warp	sub-group

- Kernel launch specifies index space


work-group size S_x
work-group (w_x, w_y)
work-item
$(w_x \cdot S_x+s_x+F_x, w_y \cdot S_y+s_y+F_y)$
$(s_x, s_y) = (0, 0)$
...
work-item
$(w_x \cdot S_x+s_x+F_x, w_y \cdot S_y+s_y+F_y)$
$(s_x, s_y) = (S_x-1, 0)$
work-group size S_y
work-item
$(w_x \cdot S_x+s_x+F_x, w_y \cdot S_y+s_y+F_y)$
$(s_x, s_y) = (0, S_y-1)$
...
work-item
$(w_x \cdot S_x+s_x+F_x, w_y \cdot S_y+s_y+F_y)$
$(s_x, s_y) = (S_x-1, S_y-1)$
NDRange size G_y
NDRange size G_x

OpenCL

Kernel Language

- Kernel language similar to CUDA

	CUDA	OpenCL
thread index	threadIdx.x	get_local_id(0)
block index	blockIdx.x	get_group_id(0)
block dimension	blockDim.x	get_local_size(0)
grid dimension	gridDim.x	get_global_size(0)
global memory	__global__	__global
group memory	__shared__	__local
constant memory	__constant__	__constant
barrier	__syncthreads	barrier(CLK_LOCAL_MEMFENCE)

OpenCL

macOS Support?

- Generated code often incorrect for complex code
- Deprecated since macOS Mojave (10.14)
- Canceled for macOS Catalina (10.15) via security update 2021-002

- Metal
  - Graphics and compute shader API
  - Replaces OpenGL and OpenCL

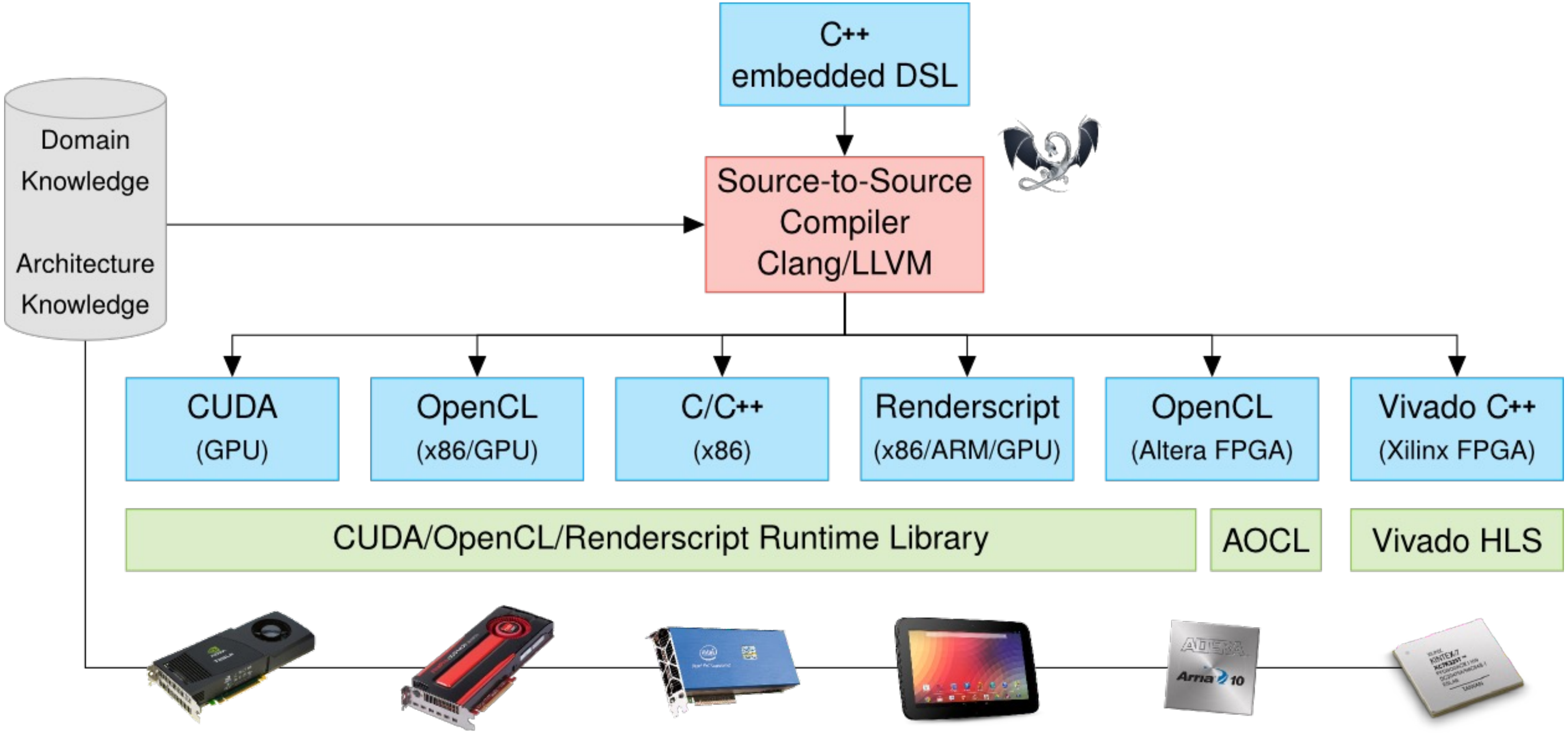
OpenCL

Android Support?

- Not available on Android
- Android provides RenderScript / FilterScript
  - Compute API, C99-derived language
  - Java class reflected by compiler
  - Mainly in maintenance mode since years
  - Introduced with Android 3 (2011)
  - Deprecated in Android 12 (2021)

- Vulkan
  - Supported since Android 7 (2016)
  - Replaces RenderScript / FilterScript

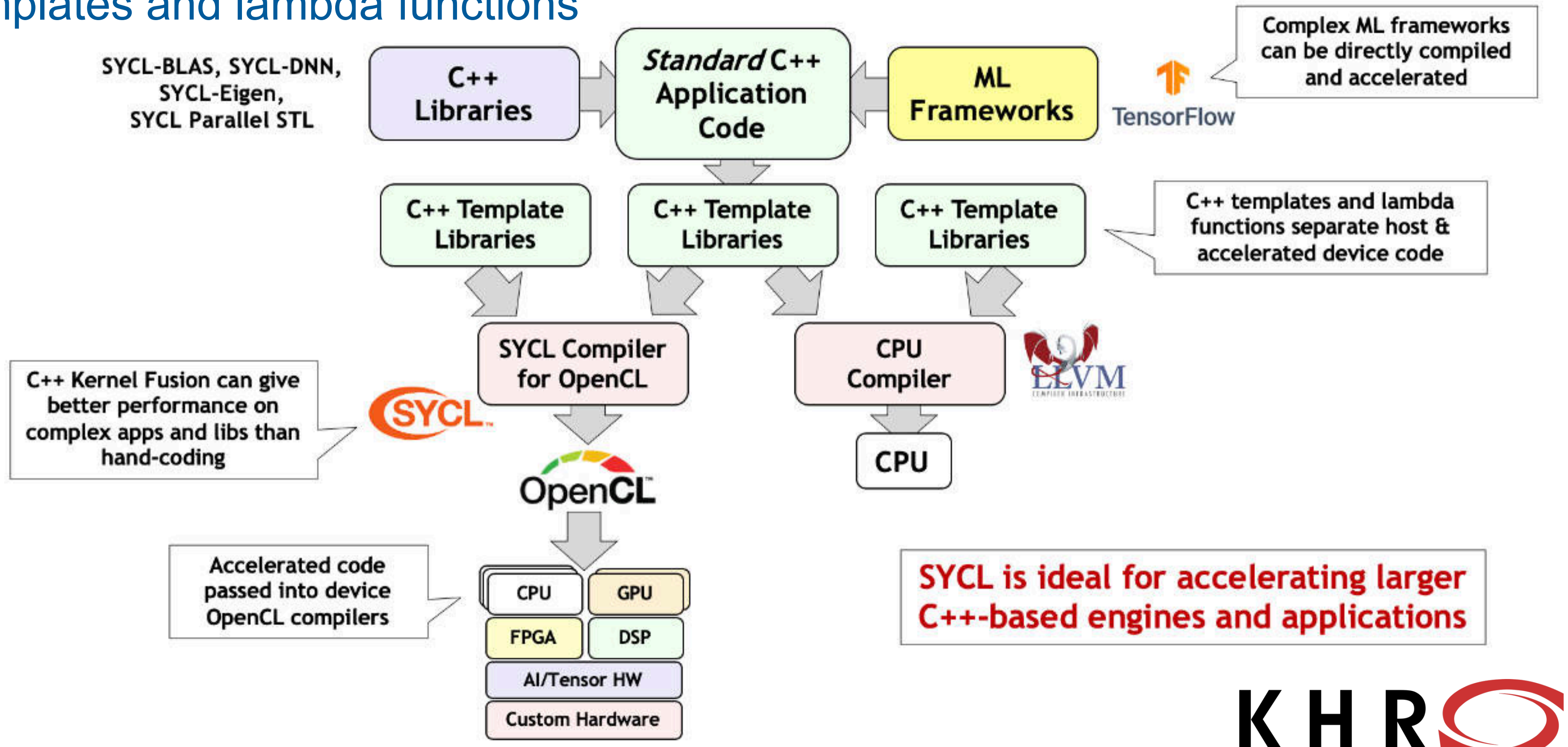
Hipacc



https://hipacc-lang.org

SYCL

- C++ single source heterogeneous programming
  - Templates and lambda functions



SYCL-BLAS, SYCL-DNN, SYCL-Eigen, SYCL Parallel STL

C++ Libraries

Standard C++ Application Code

ML Frameworks

TensorFlow

Complex ML frameworks can be directly compiled and accelerated

C++ Template Libraries

C++ Template Libraries

C++ Template Libraries

C++ templates and lambda functions separate host & accelerated device code

SYCL Compiler for OpenCL

CPU Compiler

LLVM

C++ Kernel Fusion can give better performance on complex apps and libs than hand-coding

SYCL

OpenCL

CPU

Accelerated code passed into device OpenCL compilers

CPU

GPU

FPGA

DSP

AI/Tensor HW

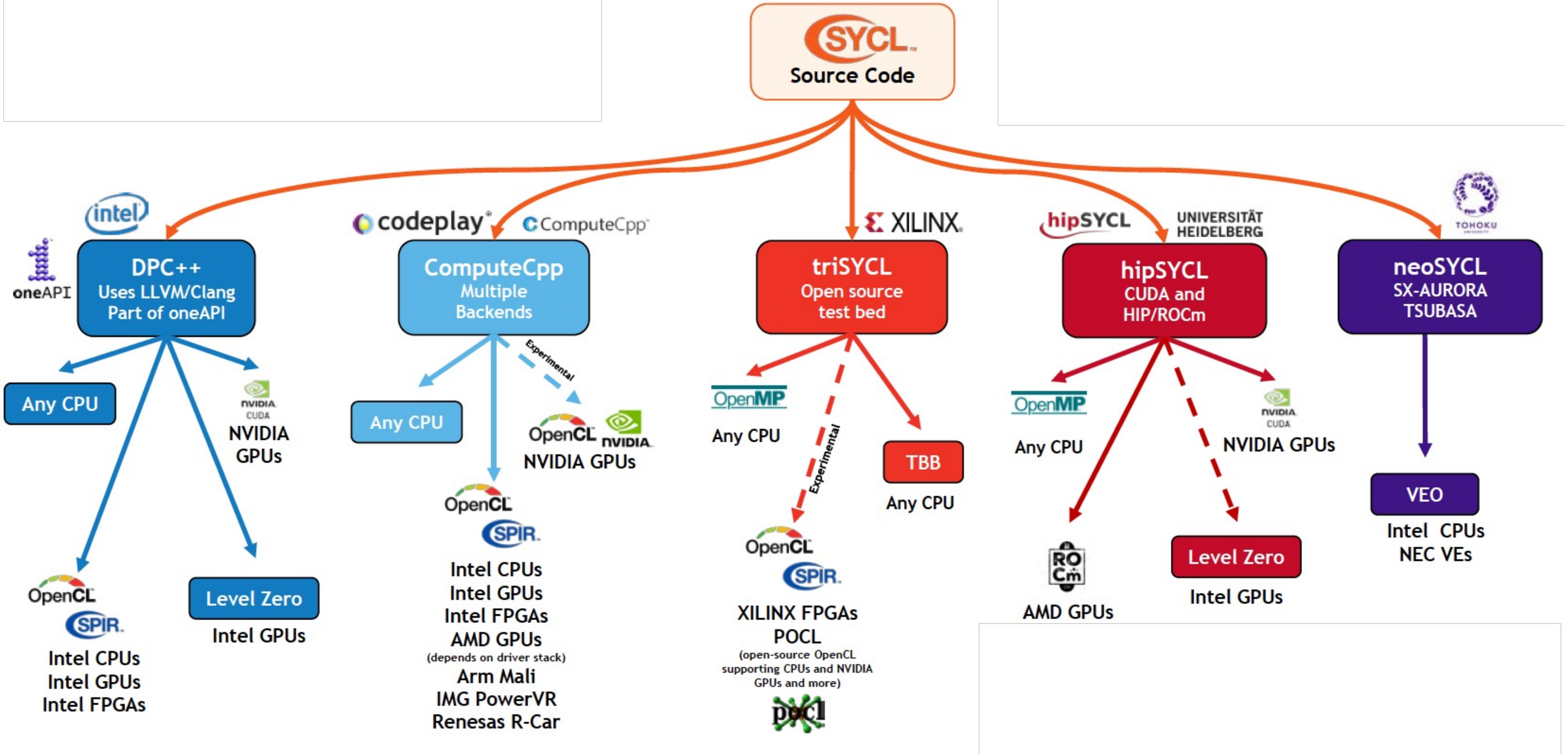
Custom Hardware

SYCL is ideal for accelerating larger C++-based engines and applications

KHRONOS GROUP

SYCL

Backends



SYCL Source Code

DPC++
Uses LLVM/Clang
Part of oneAPI

- Any CPU
- NVIDIA CUDA: NVIDIA GPUs
- OpenCL SPIR: Intel CPUs, Intel GPUs, Intel FPGAs
- Level Zero: Intel GPUs

ComputeCpp
Multiple Backends

- Any CPU
- OpenCL NVIDIA (Experimental): NVIDIA GPUs
- OpenCL SPIR: Intel CPUs, Intel GPUs, Intel FPGAs, AMD GPUs (depends on driver stack), Arm Mali, IMG PowerVR, Renesas R-Car

triSYCL
Open source test bed

- OpenMP: Any CPU
- TBB: Any CPU
- OpenCL SPIR (Experimental): XILINX FPGAs, POCL (open-source OpenCL supporting CPUs and NVIDIA GPUs and more)

hipSYCL
CUDA and HIP/ROCm

- OpenMP: Any CPU
- NVIDIA CUDA: NVIDIA GPUs
- ROCm: AMD GPUs
- Level Zero: Intel GPUs

neoSYCL
SX-AURORA TSUBASA

- VEO: Intel CPUs, NEC VEs

```cpp
#include <SYCL/sycl.hpp>
class vec_add;

int main() {
  auto array_size = 256;
  std::vector<float> A(array_size, 1.0f);
  std::vector<float> B(array_size, 1.0f);
  std::vector<float> C(array_size);
  { // beginning C++ scope - SYCL takes ownership of memory
    auto sycl_queue = cl::sycl::queue;
    auto A_buff = cl::sycl::buffer<float>(A.data(), cl::sycl::range<1>(array_size));
    auto B_buff = cl::sycl::buffer<float>(B.data(), cl::sycl::range<1>(array_size));
    auto C_buff = cl::sycl::buffer<float>(C.data(), cl::sycl::range<1>(array_size));
    auto num_groups = sycl_queue.get_device().get_info<cl::sycl::info::device::max_compute_units>();
    auto work_group_size = sycl_queue.get_device().get_info<cl::sycl::info::device::max_work_group_size>();
    auto total_threads = num_groups * work_group_size;
    sycl_queue.submit([&](cl::sycl::handler &cgh) {
        auto A_acc = A_buff.get_access<cl::sycl::access::mode::read>(cgh);
        auto B_acc = B_buff.get_access<cl::sycl::access::mode::read>(cgh);
        auto C_acc = C_buff.get_access<cl::sycl::access::mode::write>(cgh);
        cgh.parallel_for<class vec_add>(
            cl::sycl::range<1>{total_threads}, [=](cl::sycl::item<1> index) {
                C_acc[index] = A_acc[index] + B_acc[index];
            }
        );
    });
  } // end of C++ scope
}
```

HSA Foundation

- Heterogeneous System Architecture (HSA)
- *Optimized platform architecture for OpenCL*
  - Coherent memory model
  - HSAIL intermediate representation
  - Fully asynchronous dispatch and runtime
- Only publicly available implementation
  - AMD ROCm

Radeon Open Compute (ROCm)

- Open-Source platform for HPC
- Collection of tools, libraries, frameworks
  - HCC: single source C++ accelerator language (deprecated)
  - HIP: runtime / kernel language for AMD/NVIDIA GPUs
    - cudaMalloc → hipMalloc
    - HIPIFY → convert CUDA to HIP
  - OpenCL, OpenMP, …
- Runtime implements HSA standard

Directive-Based Approaches

- Early approaches (~2008)
  - Portland Group: PGI Accelerator
  - CAPS: HMPP Workbench
- Defined new standard (~2011)
  - OpenACC
- OpenMP 4.0 (~2013)
  - Support for offloading

OpenACC

Basics

- Pragma applies to next loop only

```
#pragma acc parallel loop
for (int i=0; i<length; ++i) {
  // sequential
  for (int j=0; j<length; ++j) {
    ...
  }
}
```

```
#pragma acc parallel loop
for (int i=0; i<length; ++i) {
  #pragma acc loop
  for (int j=0; j<length; ++j) {
    ...
  }
}
```

- Region defines scope

```
#pragma acc region
{
  #pragma acc parallel loop
  for (int i=0; i<length; ++i) {
    #pragma acc loop
    for (int j=0; j<length; ++j) {
      ...
    }
  }
}
```

OpenACC

Basics

- Data regions can be specified for region
  - copy, copyin, copyout, create, present

```
#pragma acc data copy(x[0:N])
#pragma acc parallel loop
{
for (int i=0; i<length; ++i) {
  #pragma acc loop
  for (int j=0; j<length; ++j) {
    ...
  }
}
...
}
```

```
#pragma acc parallel loop copy(x[0:N])
for (int i=0; i<length; ++i) {
  #pragma acc loop
  for (int j=0; j<length; ++j) {
    ...
  }
}
```

OpenACC

Vector Addition

- OpenACC provides set of
  - Compiler directives (pragmas)
  - Library routines (runtime)
  - Environment variables

```
void vec_add(float* __restrict__ out, const float* in1, const float* in2, int lenght) {
  #pragma acc parallel loop copyin(in1[0:length], in2[0:length]), copyout(out[0:length])
  for (int i=0; i<length; ++i) {
    out[i] = in1[i] + in2[i];
  }
}
```

Web Standards ?

- WebCL
  - JavaScript bindings to OpenCL
  - Provided by the Khronos Group
  - No native support by browsers
- WebGPU
  - Future web standard and JavaScript API
  - APIs for accelerated graphics and compute
  - Initiated by Apple
  - Support by W3C, Mozilla, Microsoft, Google, …