WHAT IS SYCL?

LEARNING OBJECTIVES

- Heterogeneous parallel programming
- Learn about the SYCL specification and its implementations
- Learn about the components of a SYCL implementation
- Learn about how a SYCL source file is compiled
- Learn where to find useful resources for SYCL

SUPERCOMPUTING LANDSCAPE AT EXASCALE

- Need for high levels of performance driving use of accelerators
- Many, but not all, large supercomputers using GPUs:
  - LUMI at EuroHPC JU: AMD Trento CPU and AMD MI250X GPUs (4 per node)
  - Perlmutter at NERSC: AMD EPYC Milan CPUs and NVIDIA A100 GPUs
  - Frontier at ORNL: AMD EPYC custom CPUs and Radeon Instinct GPUs (4 per node)
  - Aurora at ALCF: Intel Xeon Sapphire Rapids CPUs and Xe Ponte Vecchio GPUs (6 per node)
  - El Capitan at LLNL: AMD EPYC Genoa CPUs and Radeon Instinct GPUs (4 per node)

Multiple vendor solutions to get to Exascale

PERFORMANCE PORTABLE HETEROGENEOUS PROGRAMMING

- Scientific applications need to be performant across a range of processors
- Need to write applications in (heterogeneous) parallel programming model
    - Open Standards: SYCL, OpenMP, ...
    - DSLs and abstractions: Kokkos, Raja, ...
    - Language parallelism: ISO C++, Fortran, ...

WHAT IS SYCL?

SYCL is a single source, high-level, standard C++ programming model, that can target a range of heterogeneous platforms

WHAT IS SYCL?

A first example of SYCL code. Elements will be explained in coming sections!

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

int main(int argc, char *argv[]) {
  std::vector<float> dA{2.3}, dB{3.2}, dO{7.9};

  try {
    auto asyncHandler = [&](sycl::exception_list eL) {
      for (auto &e : eL)
        std::rethrow_exception(e);
    };
    sycl::queue gpuQueue{sycl::default_selector{}, asyncHandler};

    sycl::buffer bufA{dA.data(), sycl::range{dA.size()}};
    sycl::buffer bufB{dB.data(), sycl::range{dB.size()}};
    sycl::buffer bufO{dO.data(), sycl::range{dO.size()}};

    gpuQueue.submit([&](sycl::handler &cgh) {
      sycl::accessor inA(bufA, cgh, sycl::read_only);
      sycl::accessor inB(bufB, cgh, sycl::read_only);
      sycl::accessor out(bufO, cgh, sycl::write_only);

      cgh.parallel_for(sycl::range{dA.size()},
                       [=](sycl::id<1> i) { out[i] = inA[i] + inB[i]; });
    });

    gpuQueue.wait_and_throw();

  } catch (sycl::exception &e) {
                    YCL exception */

}
```

Managing the data

Work unit

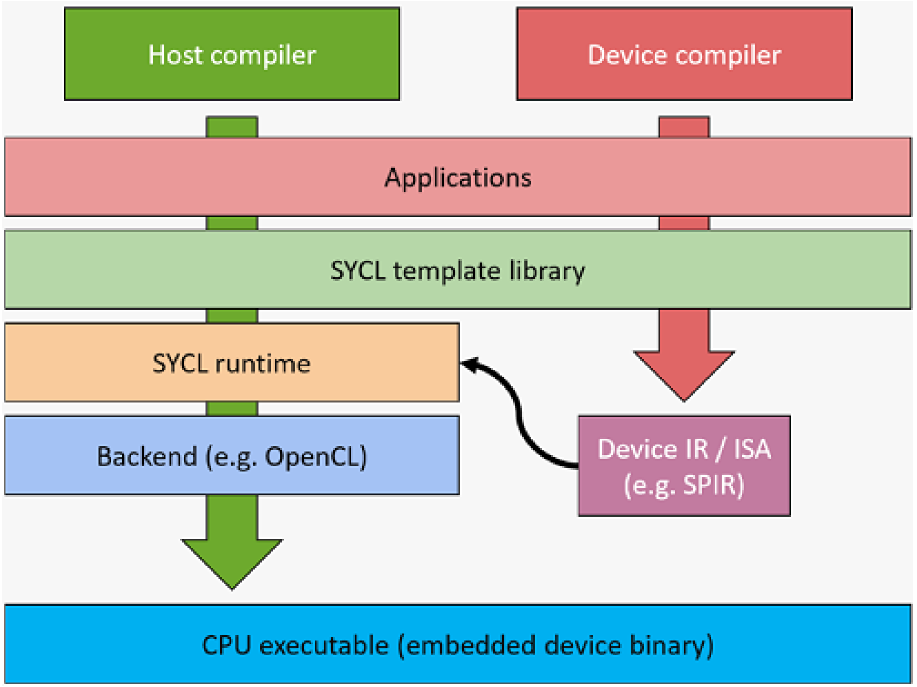
Device code

SYCL IS...

- SYCL extends C++ in two key ways:
  - heterogeneous memory
  - heterogeneous parallel compute
- SYCL is modern C++, with APIs for
  - device discovery (and information)
  - device control (kernels of work, memory)
- SYCL doesn't add extensions to the core language
- SYCL is an open standard
  - multivendor and multiarchitecture support

WHAT IS SYCL?

SYCL is a **single source**, high-level, standard C++ programming model, that can target a range of heterogeneous platforms



- SYCL allows you to write both host CPU and device code in the same C++ source file
- This requires two compilation passes; one for the host code and one for the device code

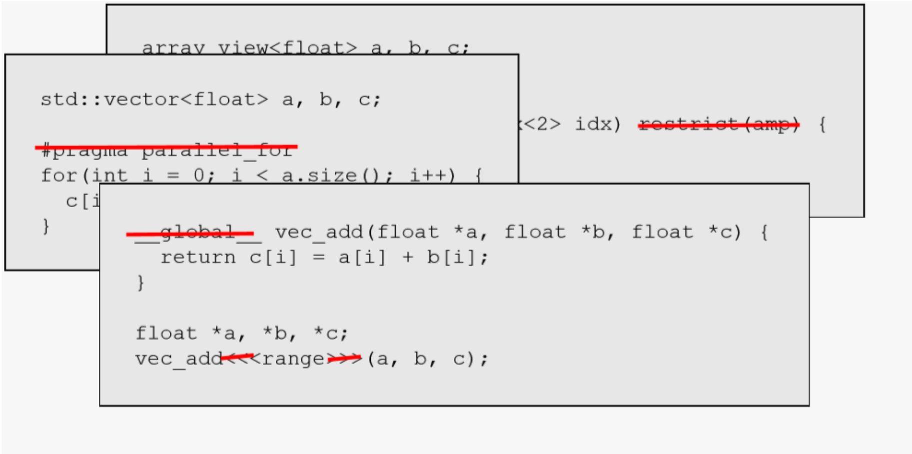
WHAT IS SYCL?

SYCL is a single source, **high-level**, standard C++ programming model, that can target a range of heterogeneous platforms

- SYCL provides high-level abstractions over common boilerplate code
  - Platform/device selection
  - Buffer creation and data movement
  - Kernel function compilation
  - Dependency management and scheduling
- High-level abstractions are good for productivity
  - SYCL has layers of abstractions when control is needed

WHAT IS SYCL?

SYCL is a single source, high-level **standard C++** programming model, that can target a range of heterogeneous platforms




```
cgh.parallel_for(range, [=](cl::sycl::id<2> idx) {
  c[idx] = a[idx] + b[idx];
});
```

- SYCL allows you to write standard C++
  - SYCL 2020 is based on C++17
- Unlike the other implementations shown on the left there are:
  - No language extensions
  - No pragmas
  - No mandatory attributes

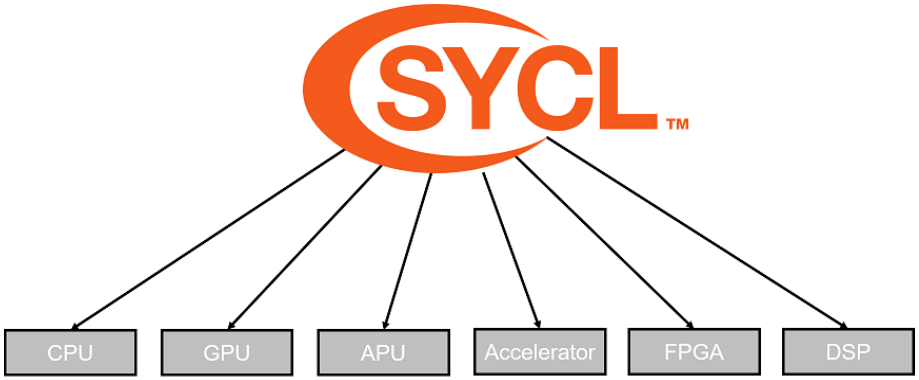
SYCL AND ISO C++

- ISO C++ has some notion of concurrency via threads and futures
- and data parallelism via algorithm and numeric libraries
- Assumes single execution space and single memory
- No control of where to run (yet)
- No asynchrony of algorithms (yet)

SYCL is aligning with and helping shape the future for heterogeneous compute in C++

WHAT IS SYCL?

SYCL is a single source, high-level standard C++ programming model, that can **target a range of heterogeneous platforms**

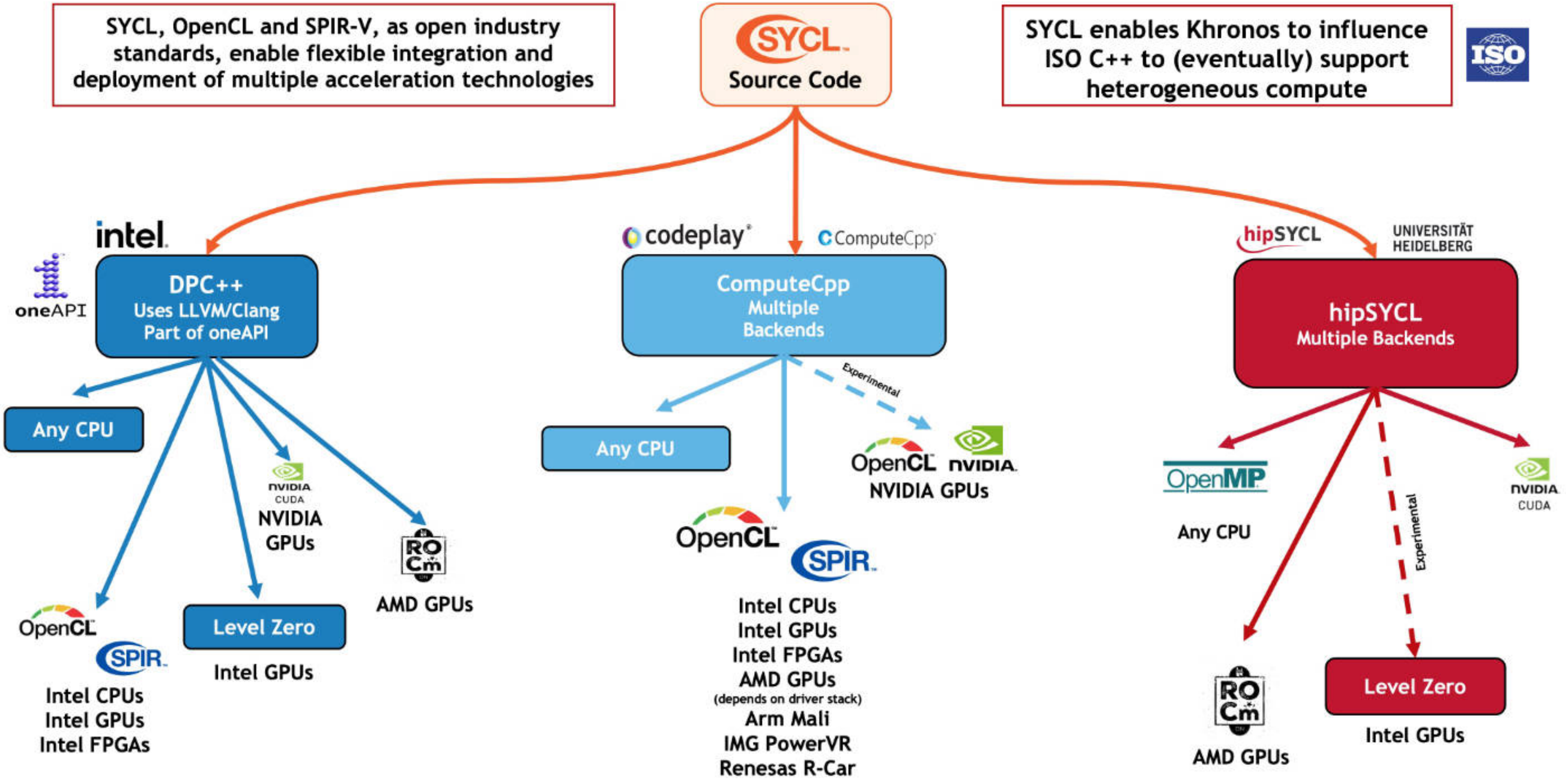


- SYCL can target any device supported by its backend
- SYCL can target a number of different backends

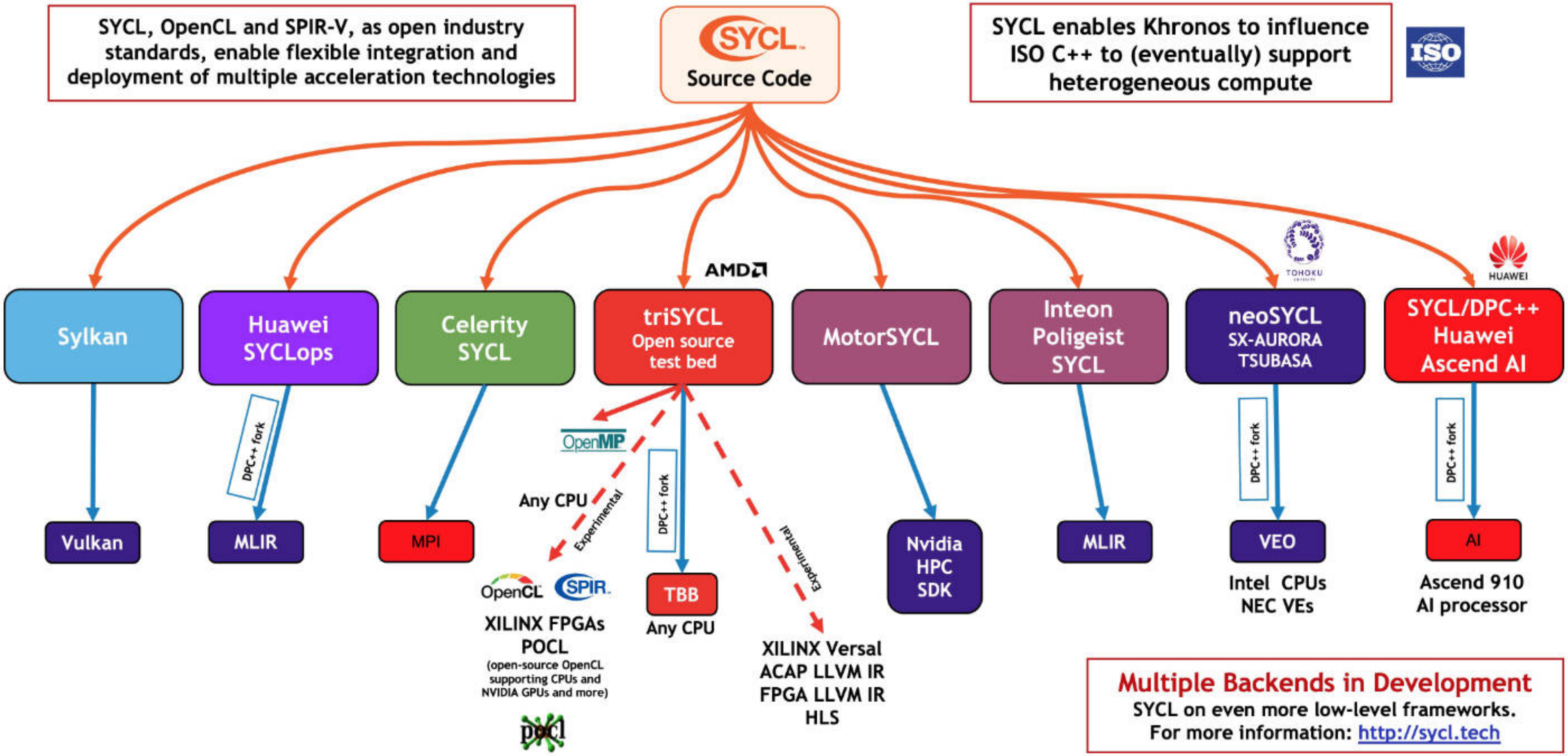
SYCL has been designed to be implemented on top of a variety of backends. Current implementations support backends such as OpenCL, CUDA, HIP, OpenMP and others.

SYCL SPECIFICATION

SYCL IMPLEMENTATIONS



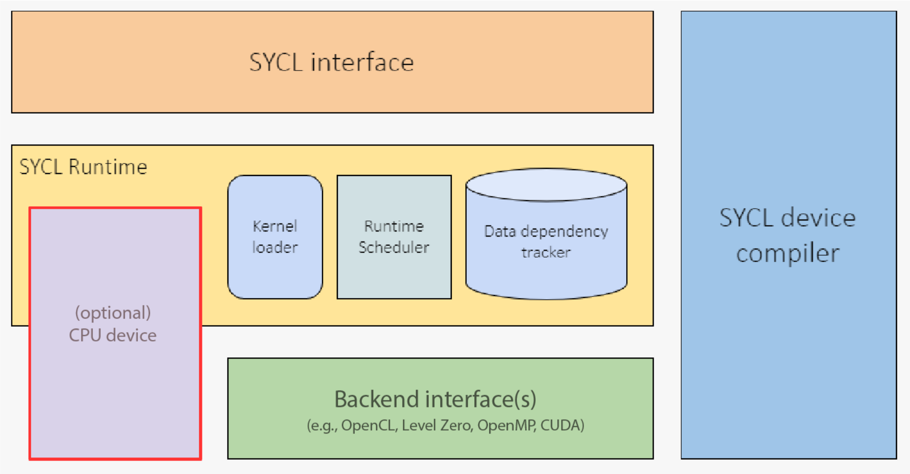
SYCL IMPLEMENTATIONS



IMPLEMENTATIONS OF A STANDARD

- SYCL is a *standard*
- Document defines behaviour of API:
    - Platform, device model
    - Memory and execution model
    - What the APIs are and what they do
- Implementations (like DPC++, hipSYCL, etc) *implement* the standard
    - Once conformant, guarenteed all APIs are supported by the implementation

WHAT A SYCL IMPLEMENTATION LOOKS LIKE



- The SYCL interface is a C++ template library that developers can use to access the features of SYCL
- The same interface is used for both the host and device code

- The host is generally the CPU and is used to dispatch the parallel execution of kernels
- The device is the parallel unit used to execute the kernels, such as a GPU

WHERE TO GET STARTED WITH SYCL

- Visit **https://sycl.tech** to find out about all the SYCL book, implementations, tutorials, news, and videos
- Visit **https://www.khronos.org/sycl/** to find the latest SYCL specifications
- Checkout the documentation provided with one of the SYCL implementations.

QUESTIONS

EXERCISE

Code_Exercises/Exercise_01_Compiling_with_SYCL/source.cpp

Configure your environment for using SYCL and compile a source file with the SYCL compiler.

Task: Include the SYCL header and successfully build and run a binary.

INTEL DEVCLOUD

1. Read the README for the SYCL Academy isc23 branch
2. Register for the Intel DevCloud
3. Follow instructions to use the Jupyter Lab console

https://devcloud.intel.com/oneapi/get_started/