

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 LANDCAPE 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 application 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!

```
1 #include <CL/sycl.hpp>
2
3 int main(int argc, char *argv[]) {
4     std::vector<float> dA{2.3}, dB{3.2}, d0{7.9};
5
6     try {
7         auto asyncHandler = [&](sycl::exception_list eL) {
8             for (auto &e : eL)
9                 std::rethrow_exception(e);
10        };
11        sycl::queue gpuQueue{sycl::default_selector{}, asyncHandler};
12
13        sycl::buffer bufA{dA.data(), sycl::range{dA.size()}};
14        sycl::buffer bufB{dB.data(), sycl::range{dB.size()}};
15        sycl::buffer buf0{d0.data(), sycl::range{d0.size()}};
16
17        gpuQueue.submit([&](sycl::handler &cgh) {
18            sycl::accessor inA(bufA, cgh, sycl::read_only);
19            sycl::accessor inB(bufB, cgh, sycl::read_only);
20            sycl::accessor out(buf0, cgh, sycl::write_only);
21
22            cgh.parallel_for(sycl::range{dA.size()},
23                             [=](sycl::id<1> i) { out[i] = inA[i] + inB[i]; });
24        });
25
26        gpuQueue.wait_and_throw();
27
28    } catch (sycl::exception &e) {
29        /* SYCL exception */
30    }
31 }
```

Managing the data

Work unit

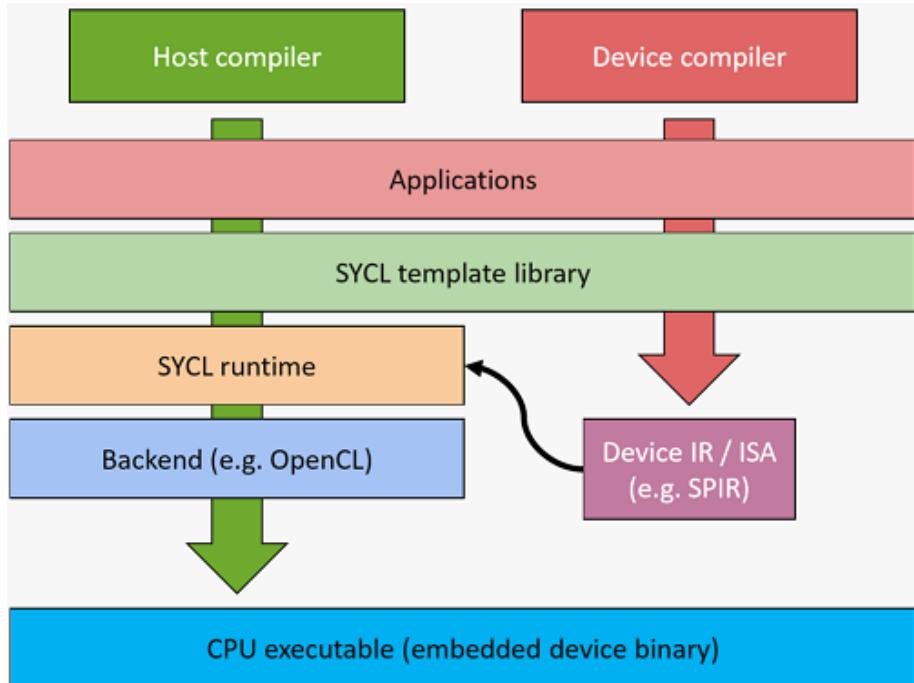
Device code

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

```
array view<float> a, b, c;

std::vector<float> a, b, c;
<2> idx) restrict(amp) {
    for(int i = 0; i < a.size(); i++) {
        c[i] = a[i] + b[i];
    }
}

float *a, *b, *c;
vec_add<<range>>(a, b, c);
```

```
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

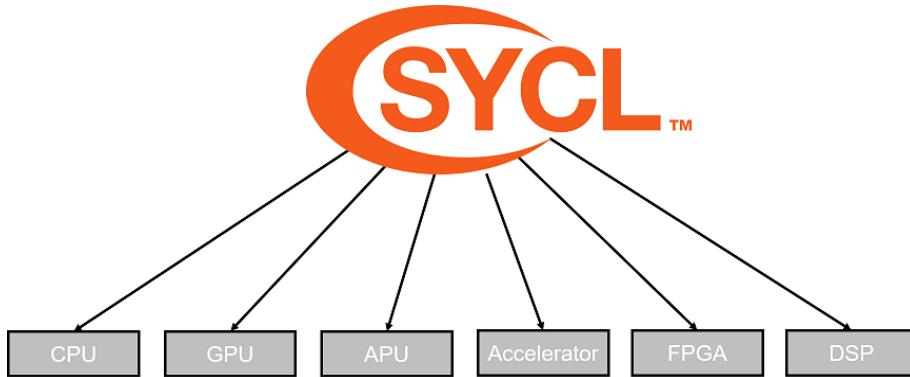
SYCL AND ISO C++

- 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 1.2

- *Released Apr 2014*

SYCL 2020
(provisional)

- *Released Jun 2020*

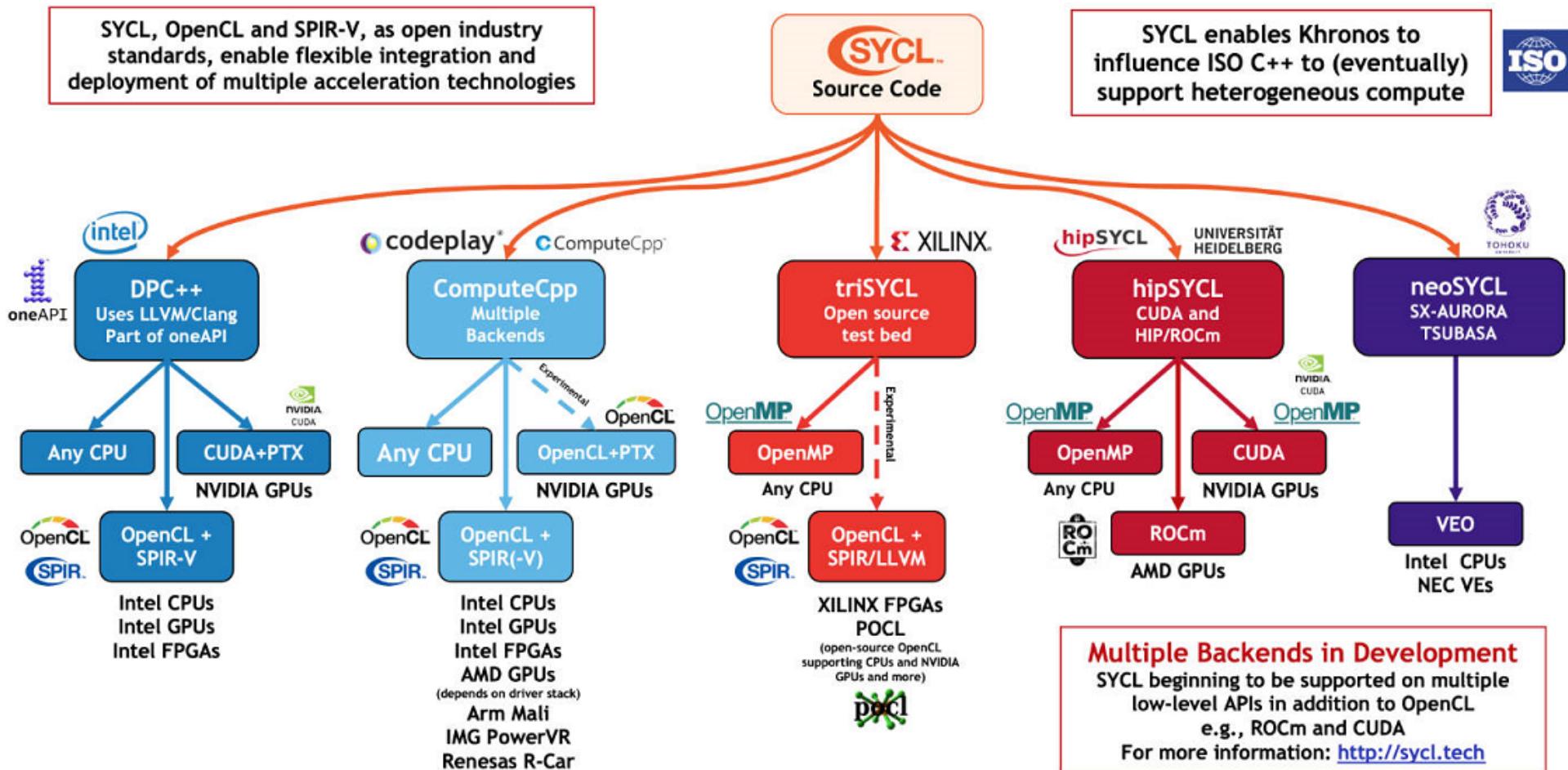
SYCL 1.2.1

- *Released Dec 2017*

SYCL 2020
(final)

- *Released Feb 2021*

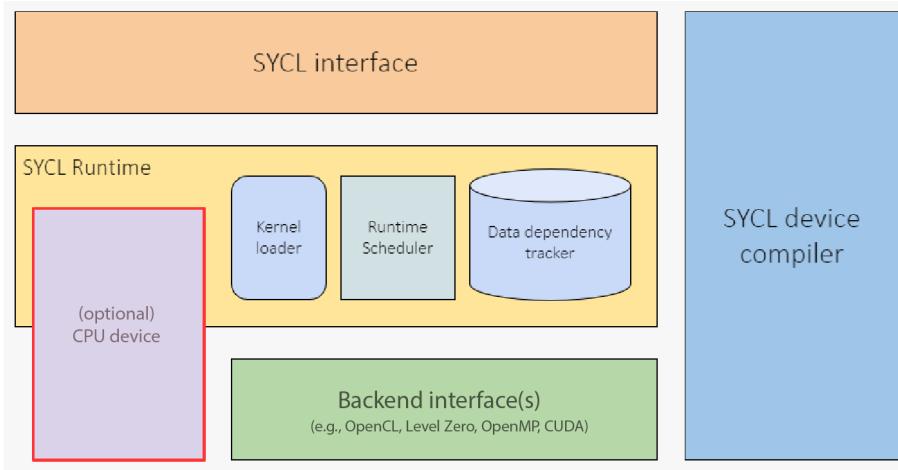
SYCL IMPLEMENTATIONS



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, guaranteed all APIs are supported by the implementation

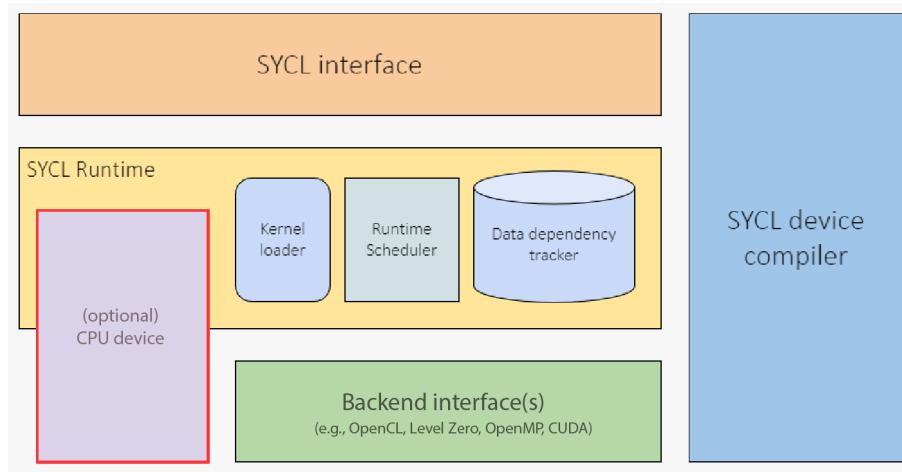
WHAT A SYCL IMPLEMENTATION LOOKS LIKE



- 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

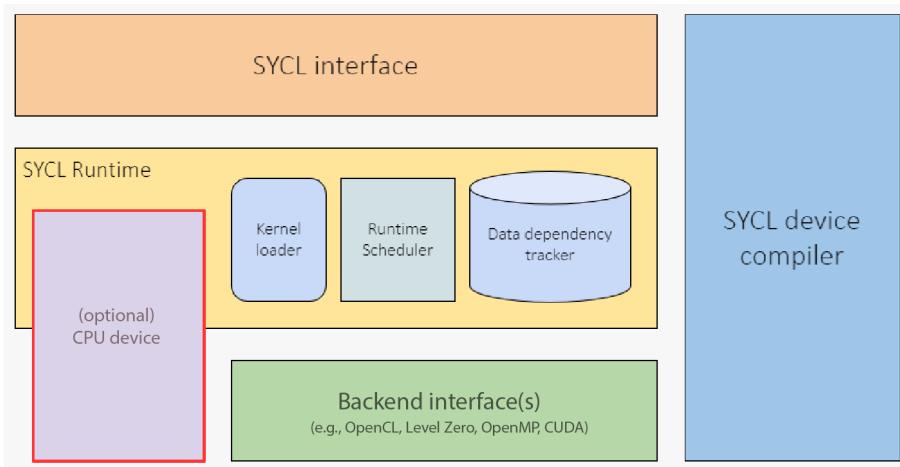
- The host is generally the CPU and is used to dispatch the parallel execution of kernels
- The device is the parallel unit used to execute the kernels, such as a GPU

WHAT A SYCL IMPLEMENTATION LOOKS LIKE



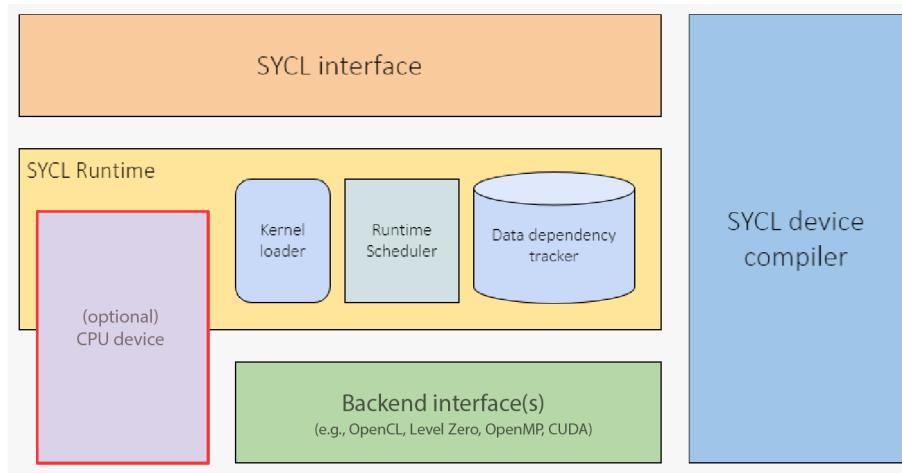
- The SYCL runtime is a library that schedules and executes work
 - It loads kernels, tracks data dependencies and schedules commands

WHAT A SYCL IMPLEMENTATION LOOKS LIKE



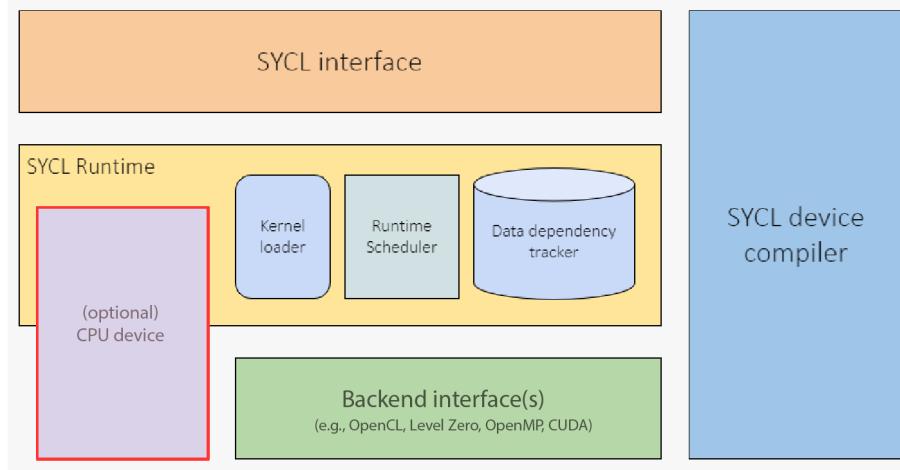
- There is no Host Device in SYCL (as of SYCL 2020)
- SYCL 1.2.1 had a concept of a 'magical' host device - an emulated backend
- SYCL 2020 implementations generally offer a CPU device
- Often, the best debugging on a platform is using a CPU device
- Yet, debugging off the CPU is important to discover offloading issues

WHAT A SYCL IMPLEMENTATION LOOKS LIKE



- The back-end interface is where the SYCL runtime calls down into a back-end in order to execute on a particular device
- Many implementations provide OpenCL backends, but some provide additional or different backends.

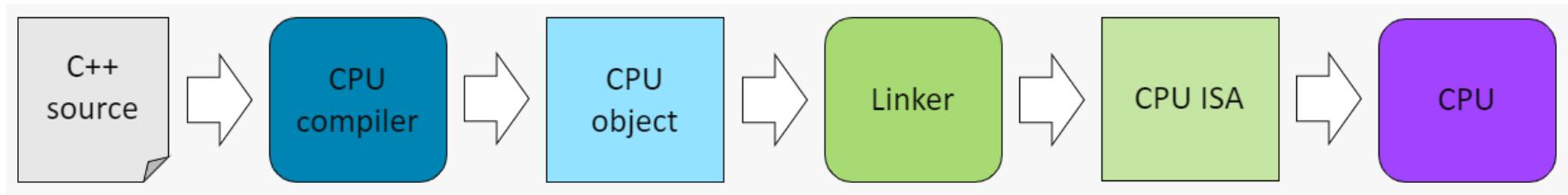
WHAT A SYCL IMPLEMENTATION LOOKS LIKE



- The SYCL device compiler is a C++ compiler which can identify SYCL kernels and compile them down to an IR or ISA
 - This can be SPIR, SPIR-V, GCN, PTX or any proprietary vendor ISA
- Some SYCL implementations are library only in which case they do not require a device compiler

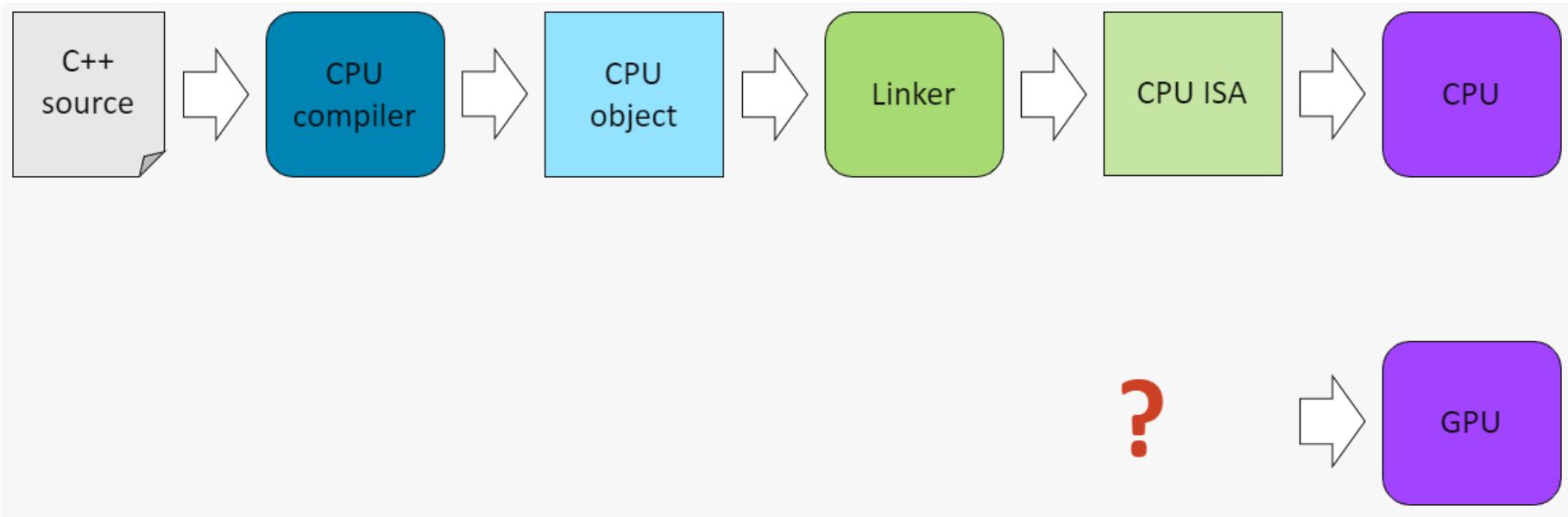
IR = Intermediate Representation **ISA** = Instruction Set Architecture

STD C++ COMPIRATION MODEL



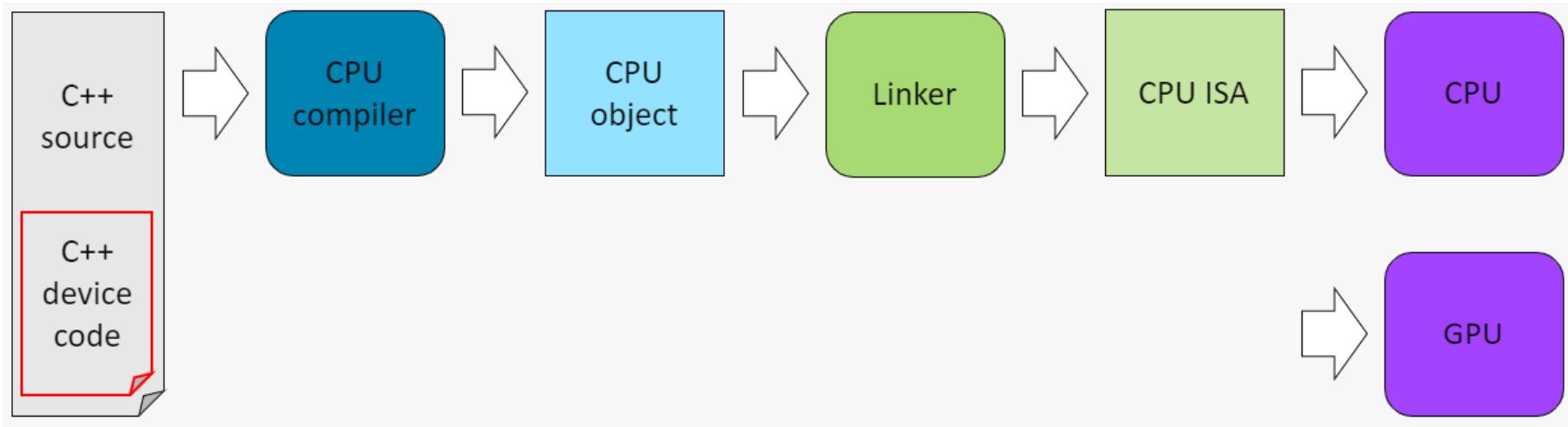
- This is the typical compilation model for a C++ source file.

STD C++ COMPIRATION MODEL



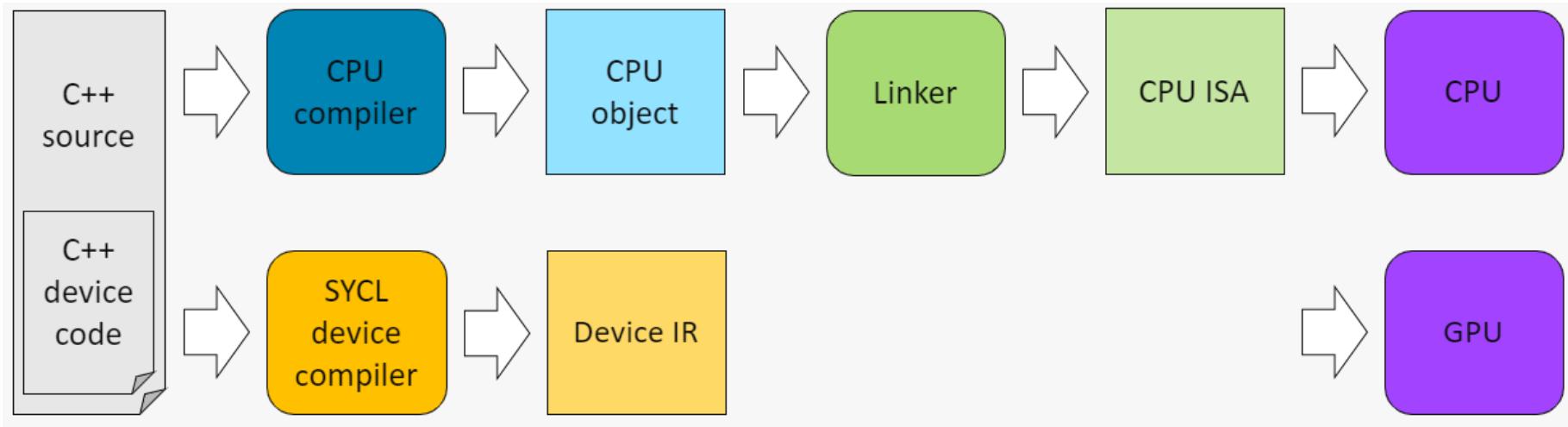
- So how do you compile a source file to also target the GPU?

STD C++ COMPIRATION MODEL



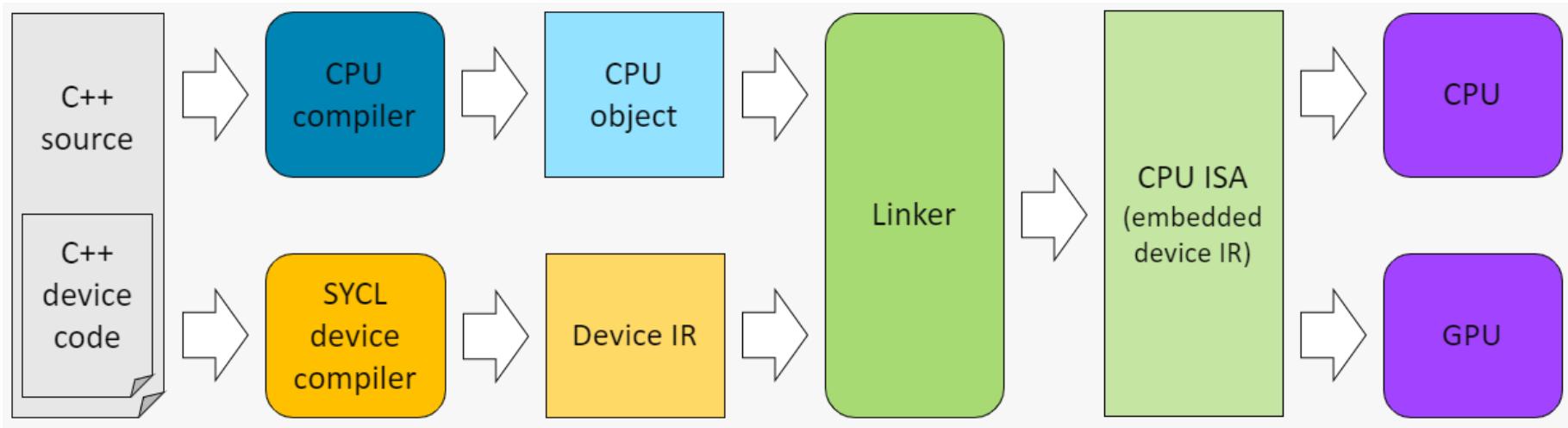
- As SYCL is single source the kernel functions are standard C++ function objects or lambda expressions.
- These are defined by submitting them to specific APIs.

STD C++ COMPIRATION MODEL



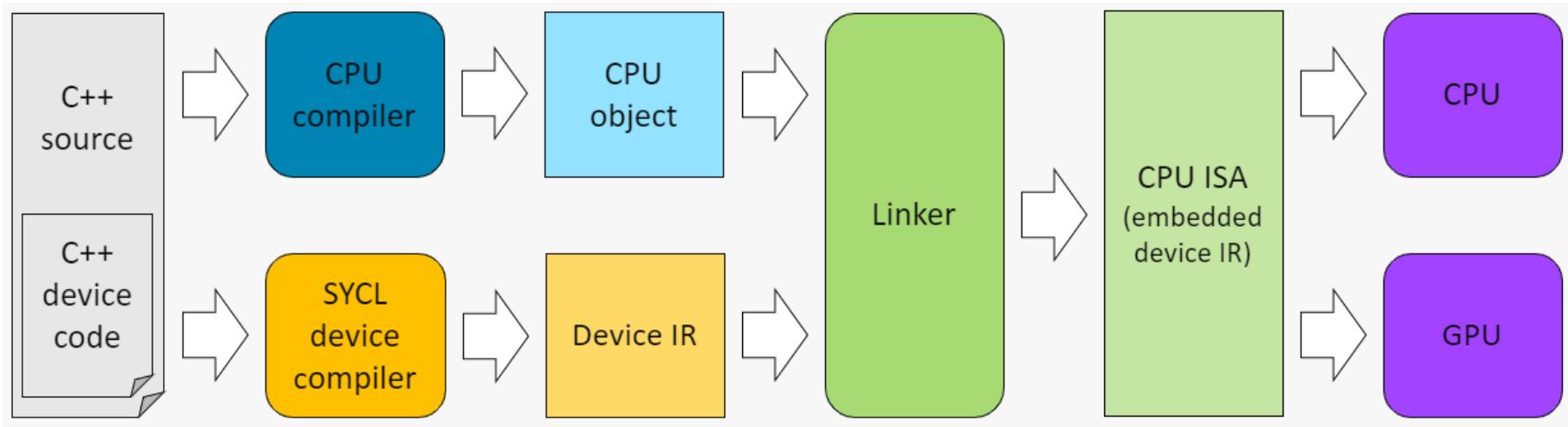
- As well as the standard C++ compiler, the source file is also compiled by a SYCL device compiler.
- This produces a device IR such as SPIR, SPIR-V or PTX or ISA for a specific architecture containing the GPU code.

STD C++ COMPIRATION MODEL



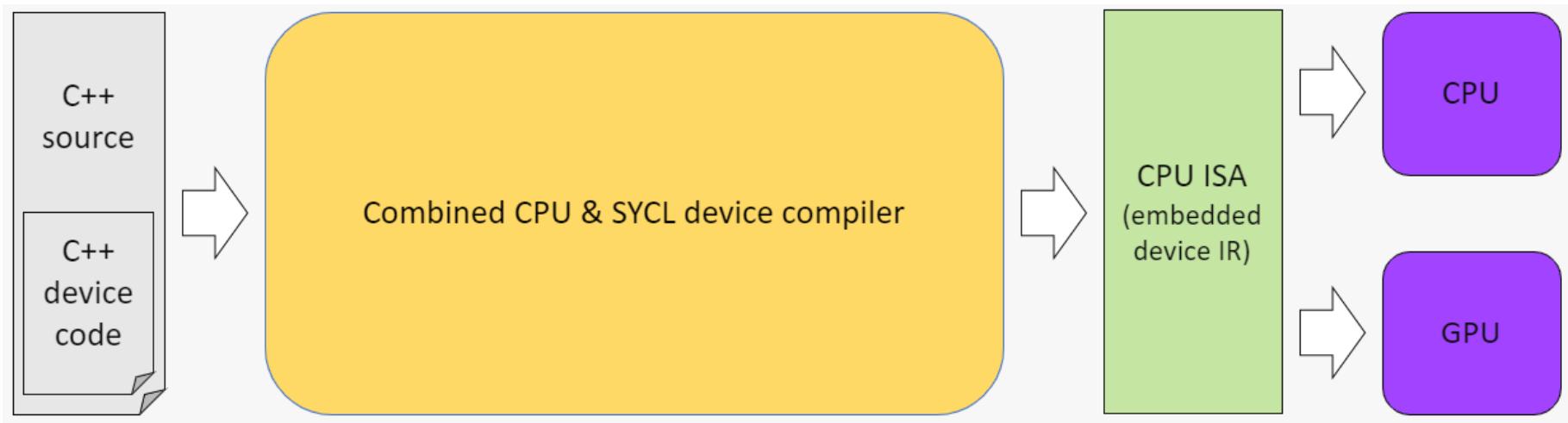
- The CPU object is then linked with the device IR or ISA to form a single executable with both the CPU and GPU code.

STD C++ COMPIRATION MODEL



- This is the multi-compiler compilation model.
- This allows the host compiler (MSVC, clang, icx, gcc) to be independent of the SYCL device compiler.

STD C++ COMPIRATION MODEL



- SYCL also supports a single-compiler compilation model.
- Where both the host compiler and SYCL device compiler are invoked from the same driver.

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_1_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 DEV CLOUD

1. Register for the Intel DevCloud
2. Follow instructions to set up SSH

<https://devcloud.intel.com/oneapi/documentation/>

DEVCLOUD DEMO

```
$ ssh devcloud
$ git clone https://github.com/codeplaysoftware/syclacademy -b isc22 --recursive
$ cd syclacademy
$ mkdir build
$ dpcpp -fsycl -o sycl-ex-1 -I../External/Catch2/single_include ../Code_Exercises/Exercise_01_Compiling
$ qsub -I -l nodes=1:gpu:ppn=2 -d .
$ ./sycl-ex-1
```