WHAT IS SYCL?

LEARNING OBJECTIVES

- 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

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!

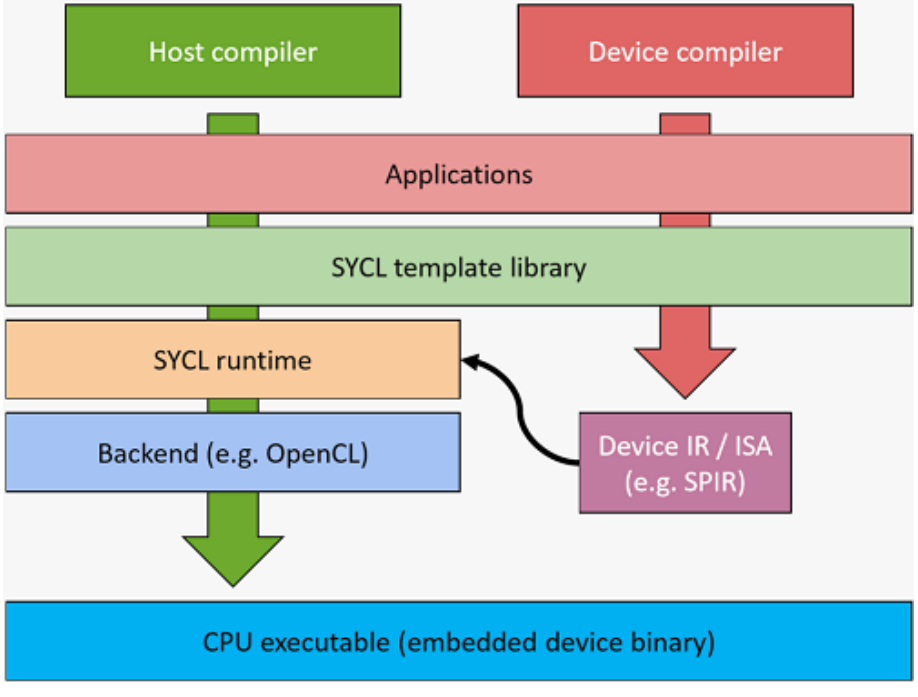
```
#include <CL/sycl.hpp>

#define PRINT_ARRAY(array, n) \
  for (auto i = 0; i < n; ++i) \
    std::cout << array[i] << " "; \
  std::cout << '\n'

using T = float;

int main() {

  constexpr size_t n = 10;

  auto q = sycl::queue{};

  T *array = sycl::malloc_shared<T>(n, q);

  for (auto i = 0; i < n; ++i)
    array[i] = 1;

  std::cout << "Before: \n";
  PRINT_ARRAY(array, n);

  q.parallel_for(n, [=](sycl::id<1> id) { array[id] *= 100; }).wait();

  std::cout << "After: \n";
  PRINT_ARRAY(array, n);

  sycl::free(array, q);
}

```

SYCL IS...

- SYCL extends C++ in two key ways:
  - device discovery (and information)
  - device control (kernels of work, memory)
- SYCL is modern C++
- SYCL is open, multivendor, multiarchitecture

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

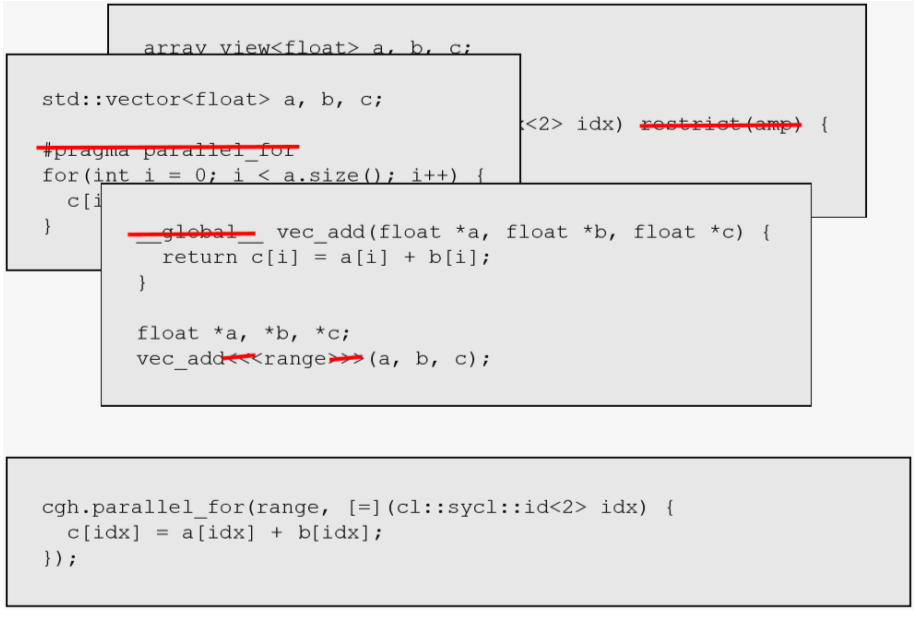
WHAT IS SYCL?

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
  - Kernel function compilation
  - Dependency management and scheduling

WHAT IS SYCL?

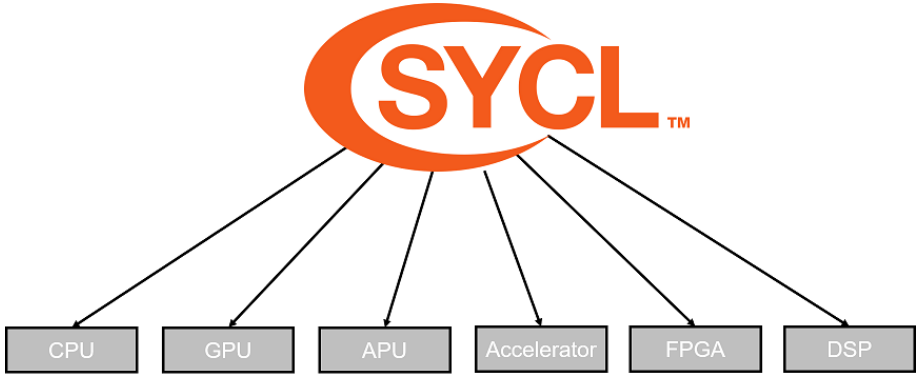
SYCL is a single source, high-level ***standard C++*** programming model, that can target a range of heterogeneous platforms



- SYCL allows you to write 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 attributes

WHAT IS SYCL?

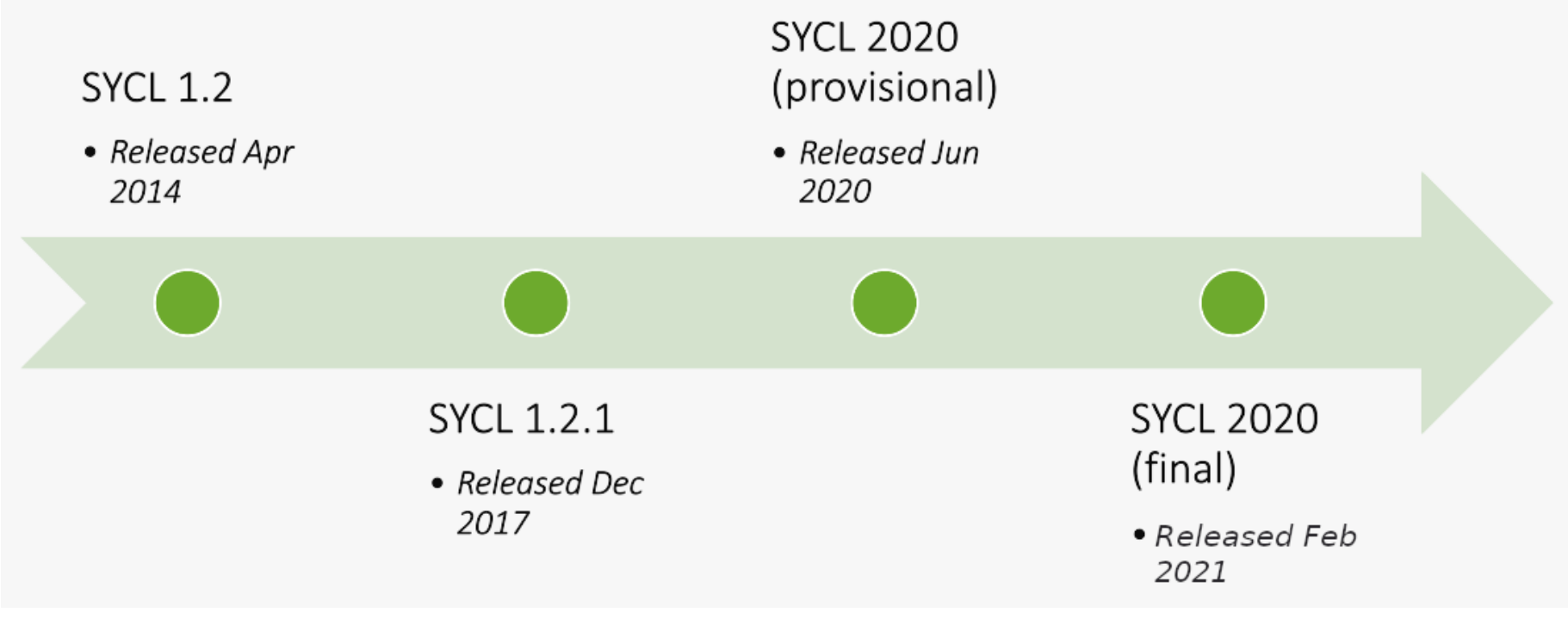
SYCL is a single source, high-level standard C++ programming model, that can **target a range of heterogeneous platforms**



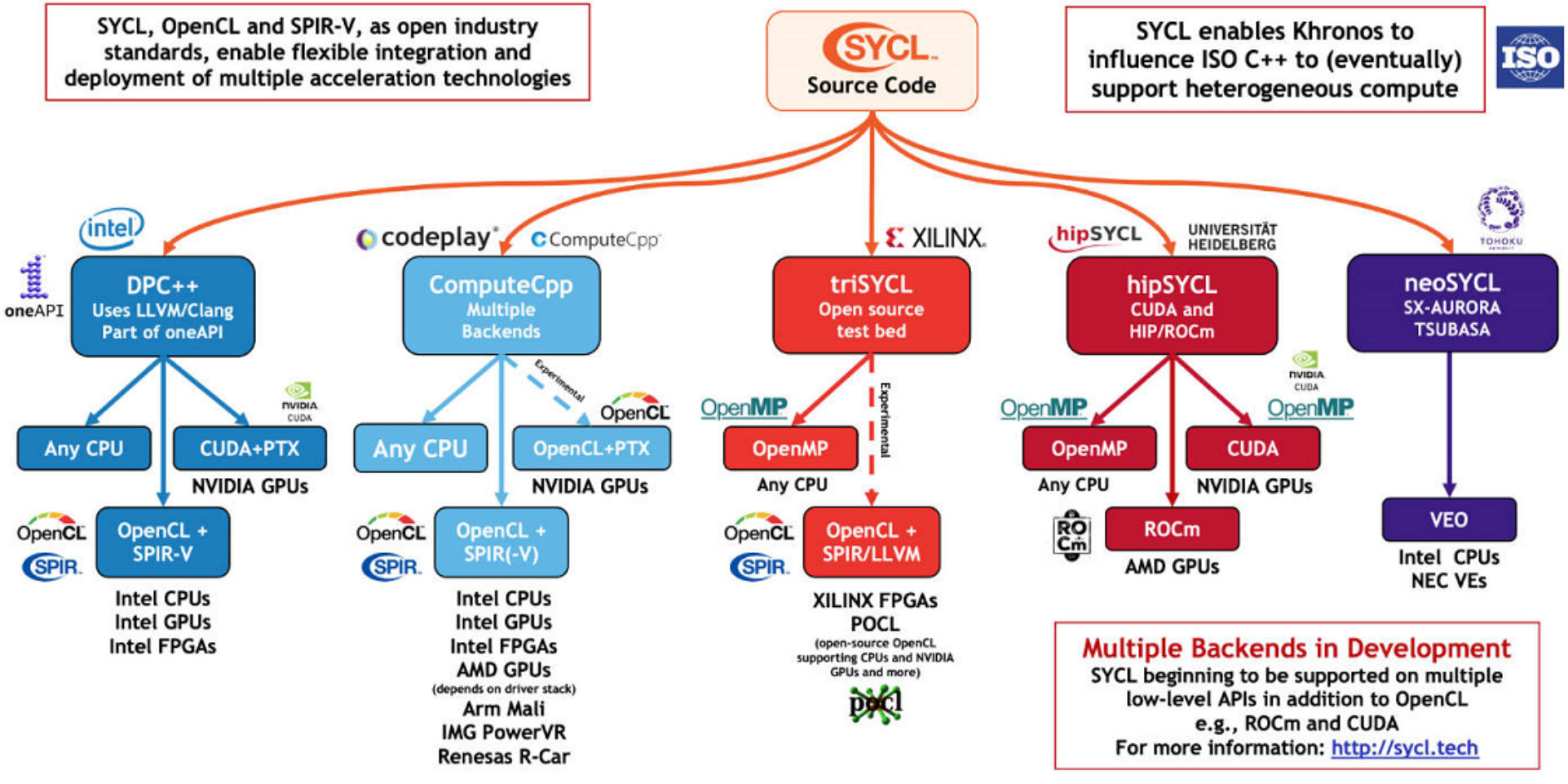
- SYCL can target any device supported by its backend
- SYCL can target a number of different backends

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.

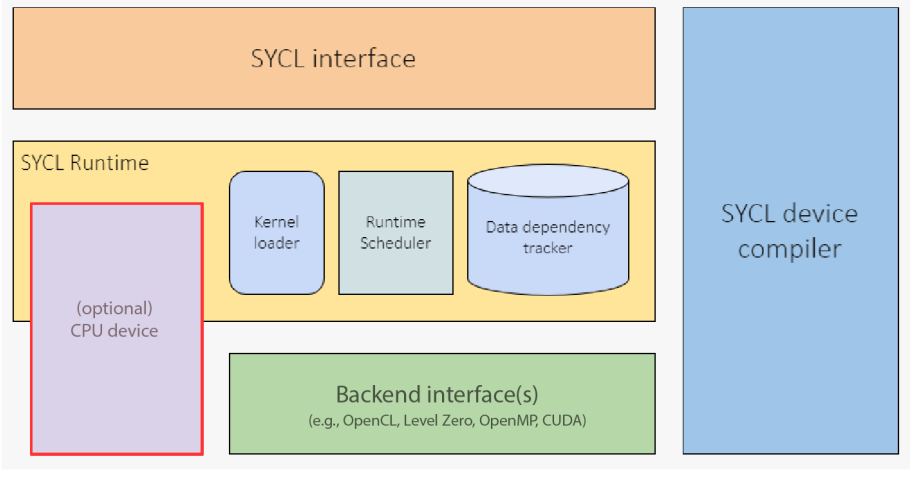
SYCL SPECIFICATION



SYCL IMPLEMENTATIONS



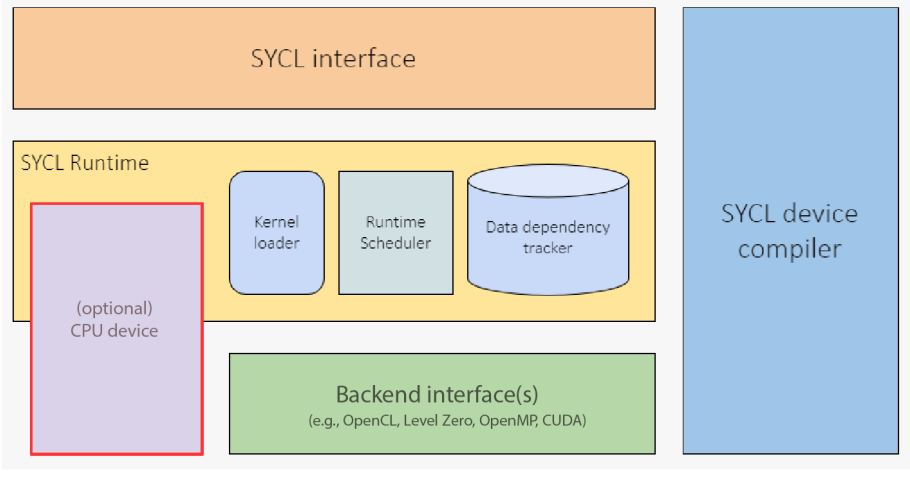
WHAT A SYCL IMPLEMENTATION LOOKS LIKE



- 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

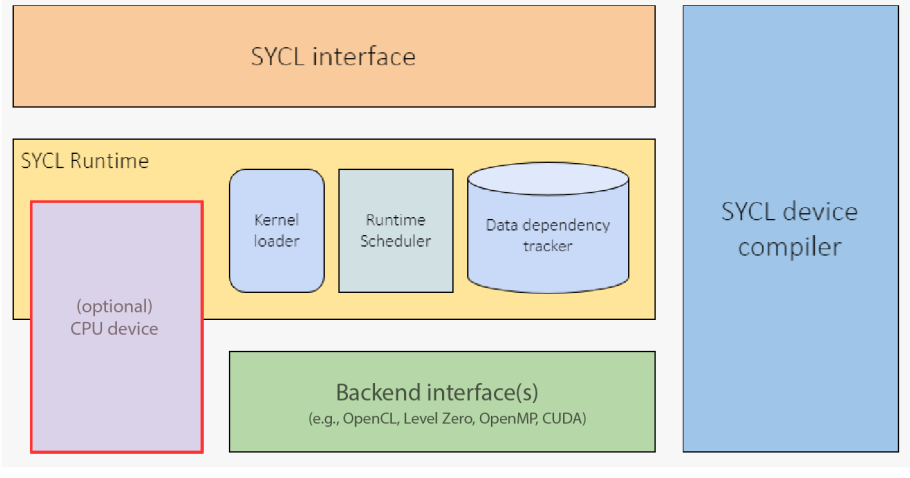
- The host is generally the CPU and is used to dispatch the parallel execution of kernels
- The device is the parallel unit used to execute the kernels, such as a GPU

WHAT A SYCL IMPLEMENTATION LOOKS LIKE



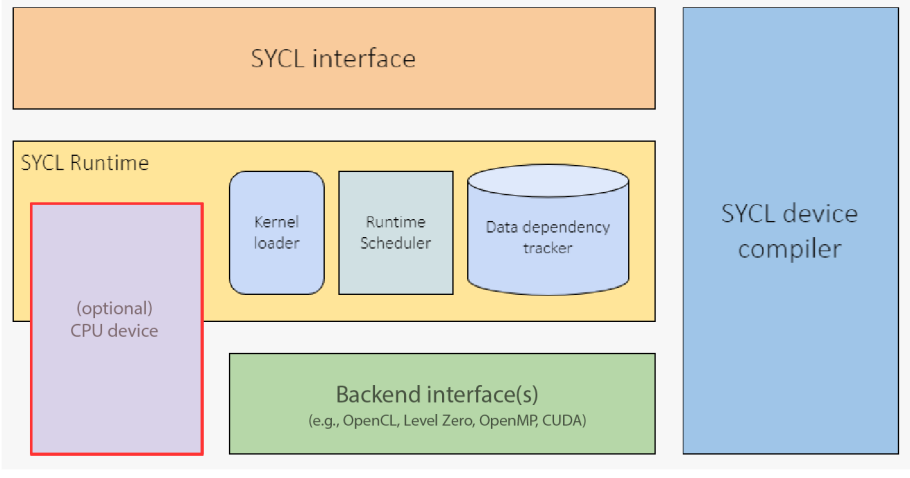
- The SYCL runtime is a library that schedules and executes work
  - It loads kernels, tracks data dependencies and schedules commands

WHAT A SYCL IMPLEMENTATION LOOKS LIKE



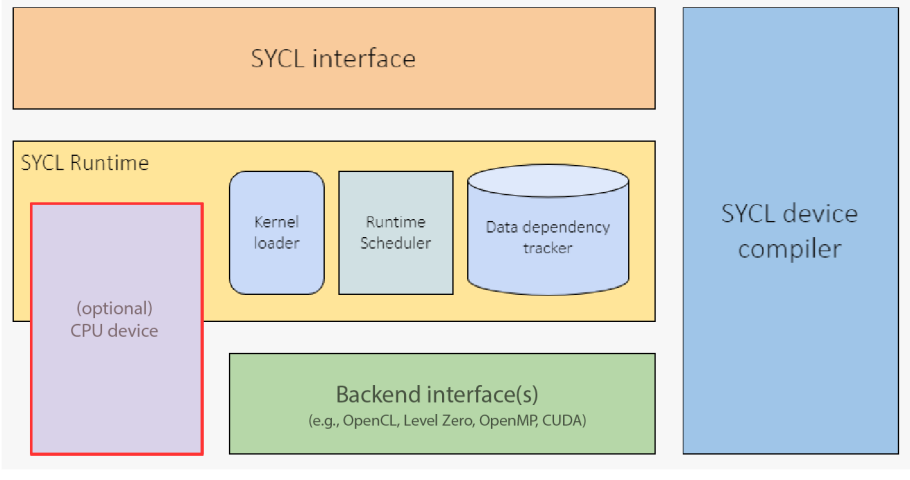
- There is no Host Device in SYCL (as of SYCL 2020)
- SYCL 1.2.1 had a concept of a 'magical' host device - an emulated backend
- SYCL 2020 implementations generally offer a CPU device
- Often, the best debugging on a platform is using a CPU device
- Yet, debugging off the CPU is important to discover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

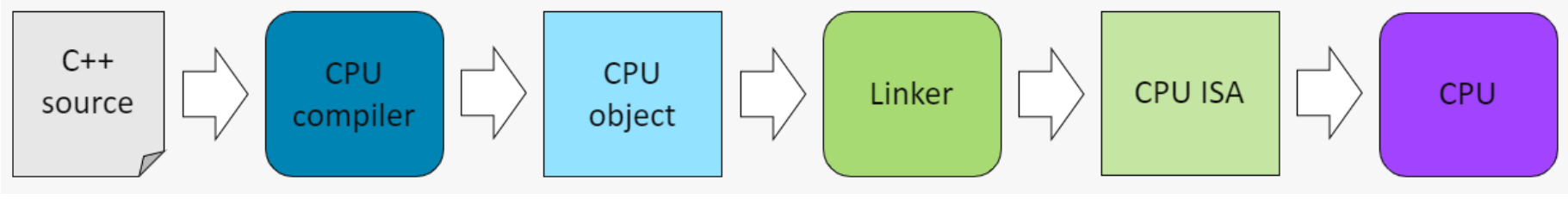
WHAT A SYCL IMPLEMENTATION LOOKS LIKE



- 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

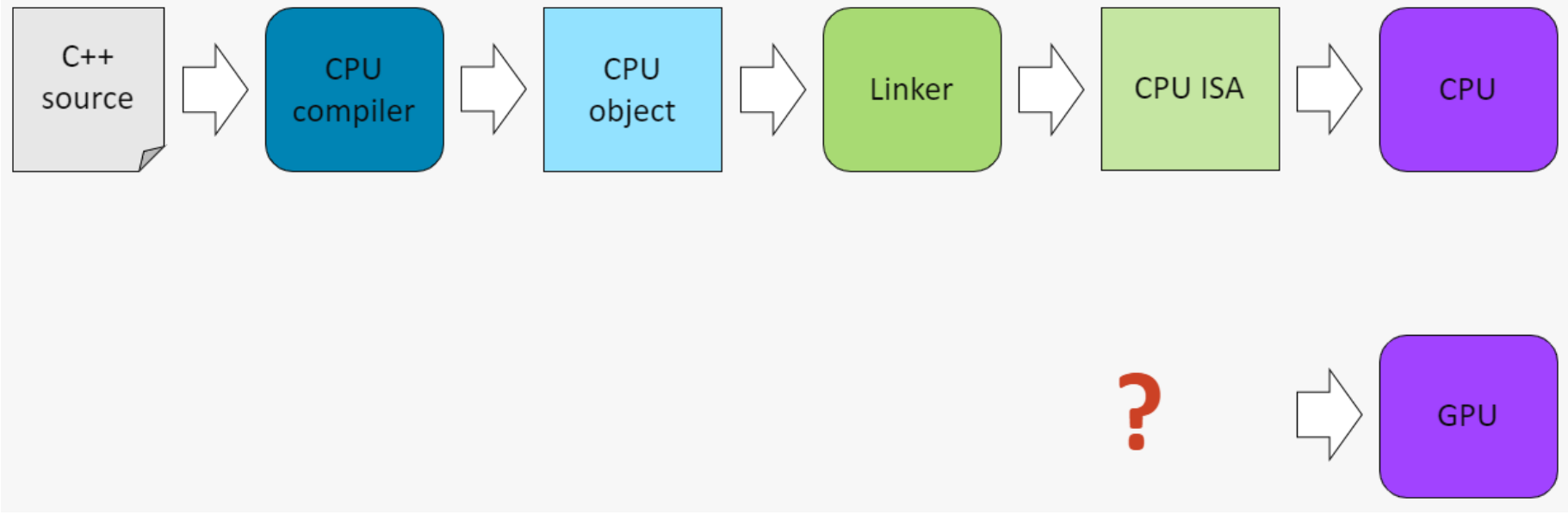
IR = Intermediate Representation **ISA** = Instruction Set Architecture

STD C++ COMPILATION MODEL



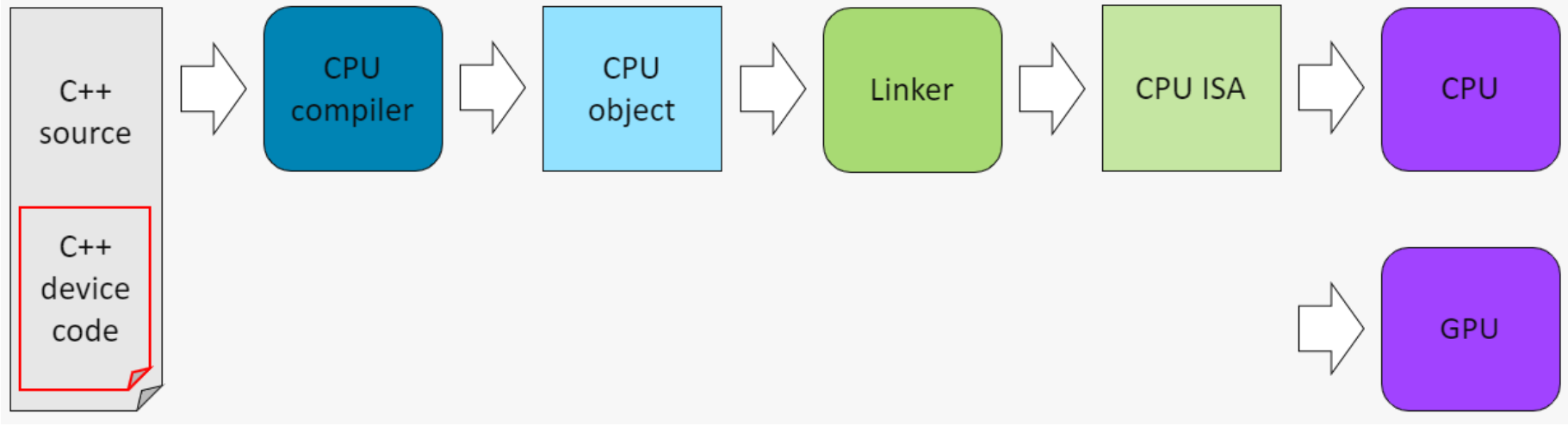
- This is the typical compilation model for a C++ source file.

STD C++ COMPILATION MODEL



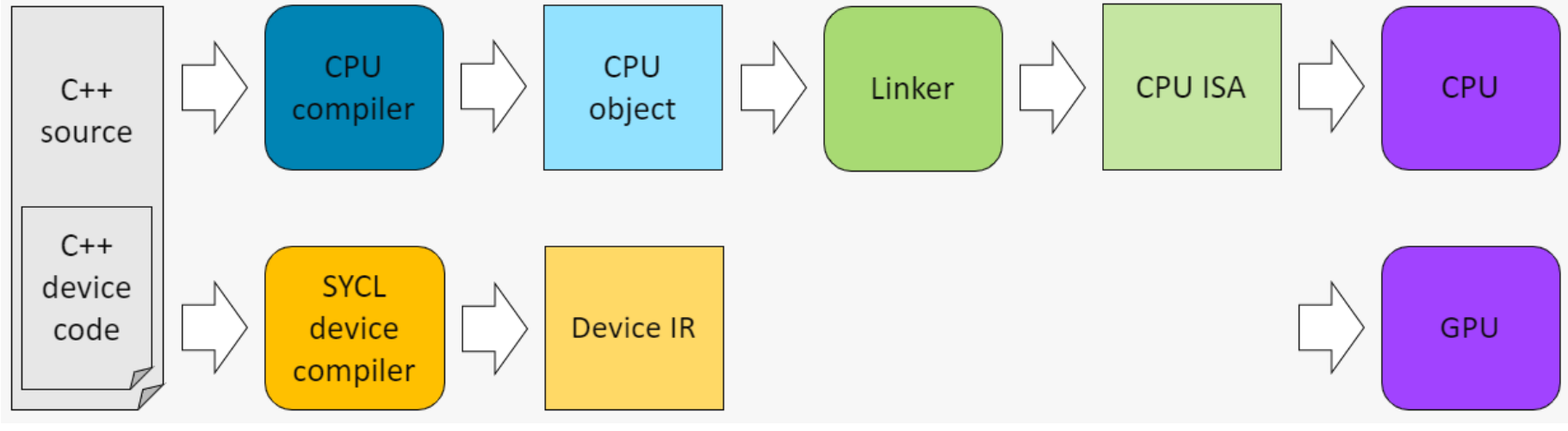
- So how do you compile a source file to also target the GPU?

STD C++ COMPILATION MODEL



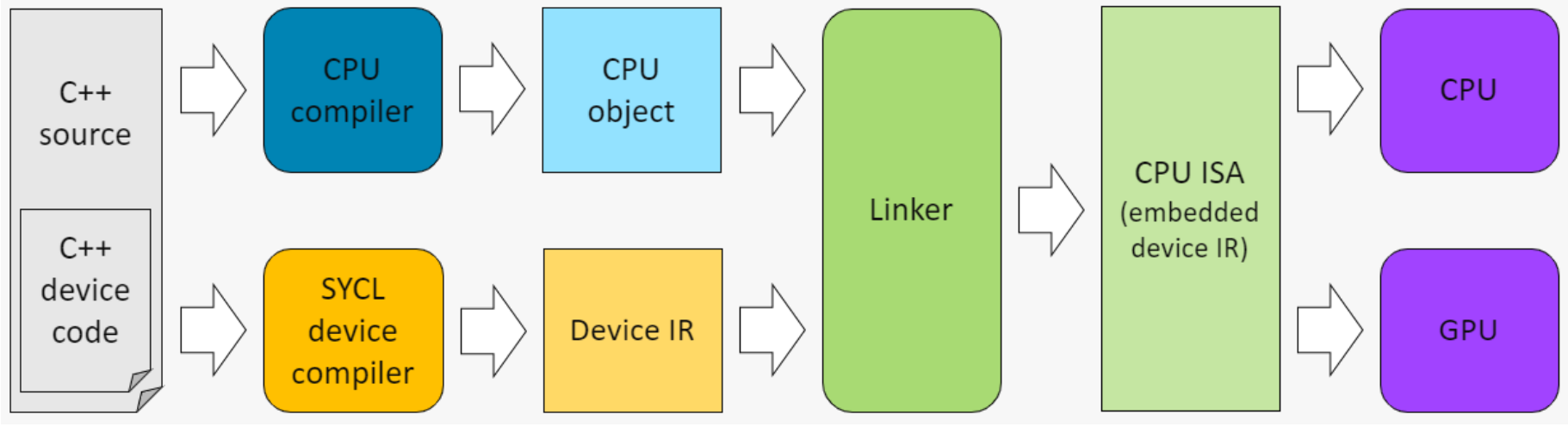
- As SYCL is single source the kernel functions are standard C++ function objects or lambda expressions.
- These are defined by submitting them to specific APIs.

STD C++ COMPILATION MODEL



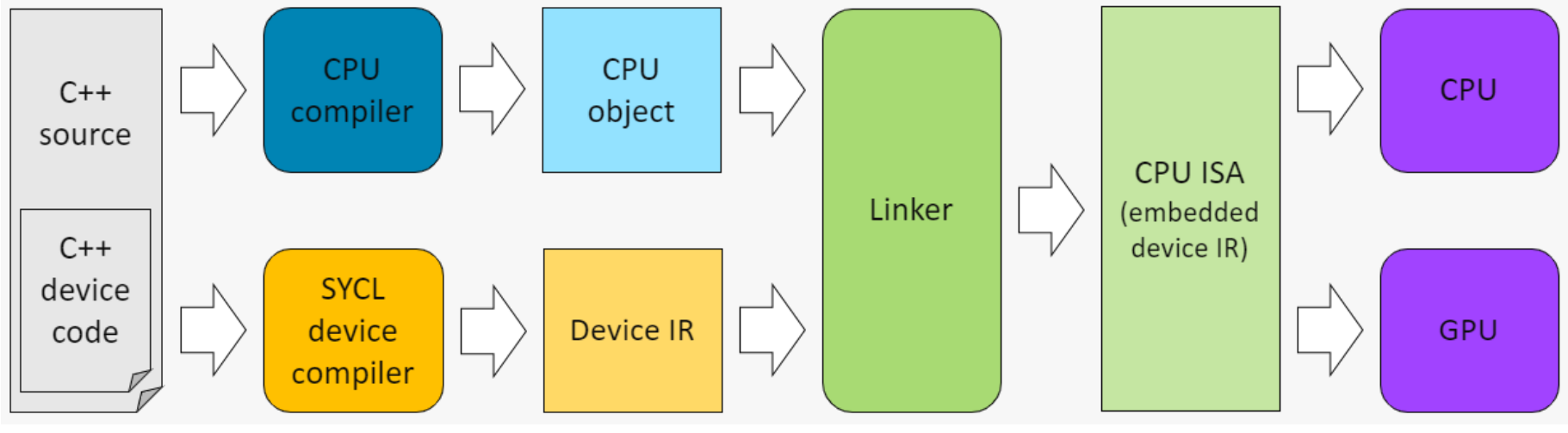
- As well as the standard C++ compiler, the source file is also compiled by a SYCL device compiler.
- This produces a device IR such as SPIR, SPIR-V or PTX or ISA for a specific architecture containing the GPU code.

STD C++ COMPILATION 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++ COMPILATION 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.

DPC++ FOR CUDA

- DPC++ is the oneAPI SYCL compiler.
- Device and host code is written in the same C++ file. (`*.cc`, `*.cpp`, etc.)

CHECK YOUR INSTALLATION

- Check for available devices using `sycl-ls`

```
hdelan@perlmutter:login07:~> sycl-ls
[ext_oneapi_cuda:gpu:0] NVIDIA CUDA BACKEND, A100-PCIE-40GB 0.0 [CUDA 11.5]
[host:host:0] SYCL host platform, SYCL host device 1.2 [1.2]
```

USING THE DPC++ COMPILER

- To compile device code to SPIR-V:
  - `clang++ -fsycl myfile.cpp`.

USING THE DPC++ COMPILER

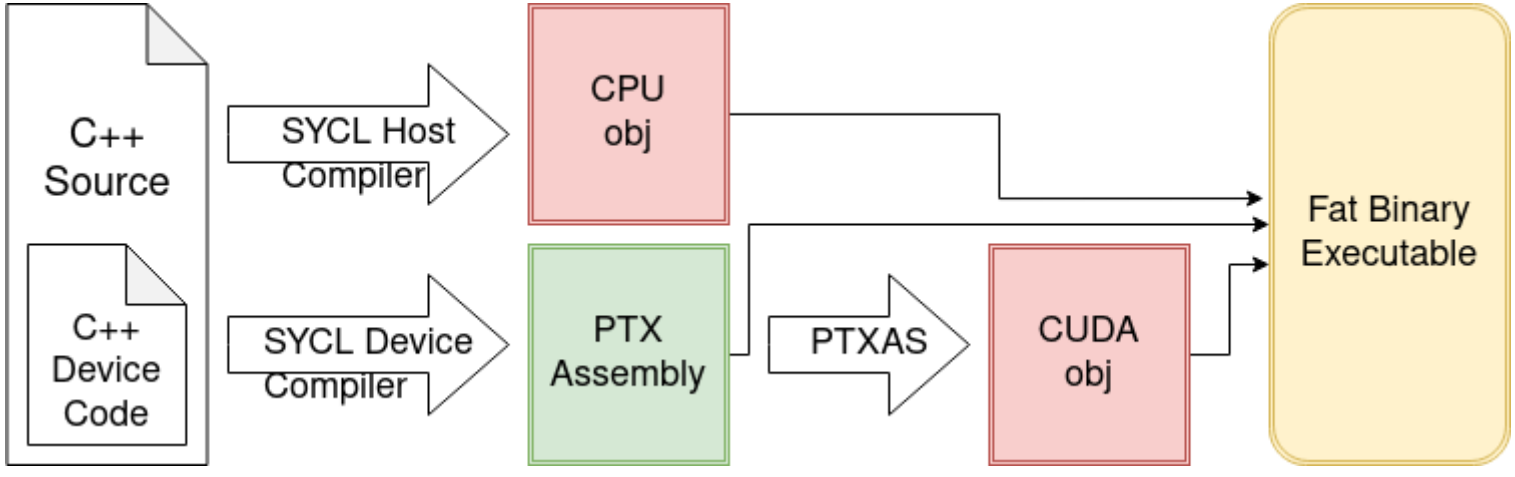
- To compile device code for the CUDA backend:
  - `clang++ -fsycl -fsycl-targets=nvptx64-nvidia-cuda myfile.cpp`.

USING THE DPC++ COMPILER

- CUDA Arch flags can be used.
  - `clang++ -fsycl -fsycl-targets=nvptx64-nvidia-cuda -Xsycl-target-backend --cuda-gpu-arch=sm_80 myfile.cpp`.

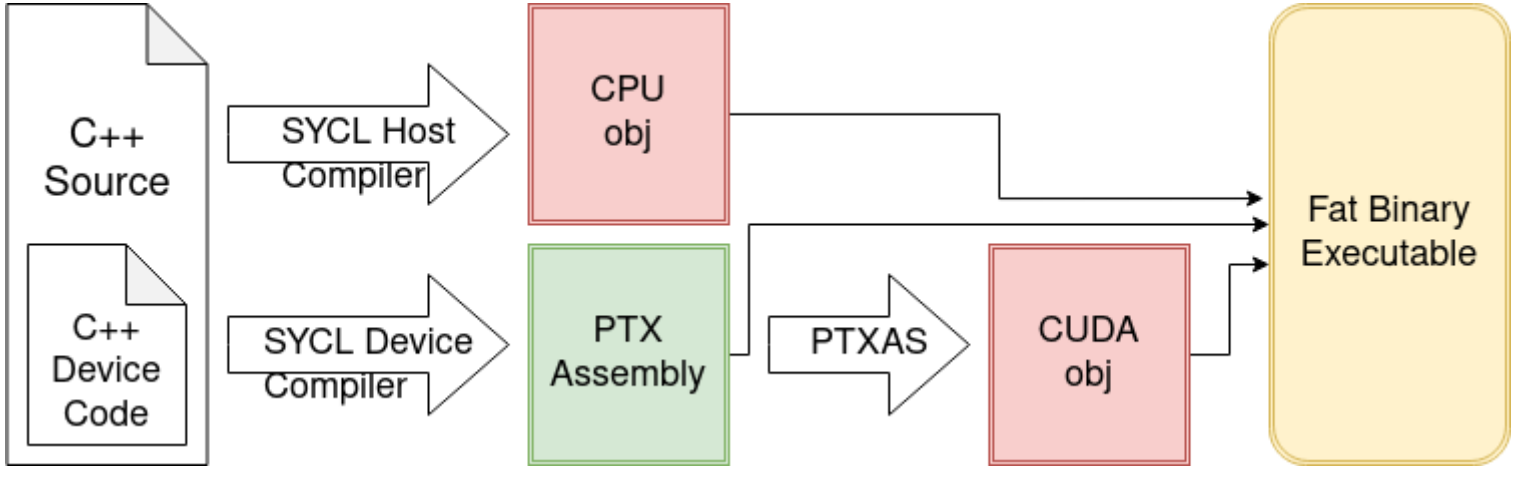
DPC++ FOR CUDA

Under the Hood



DPC++ FOR CUDA

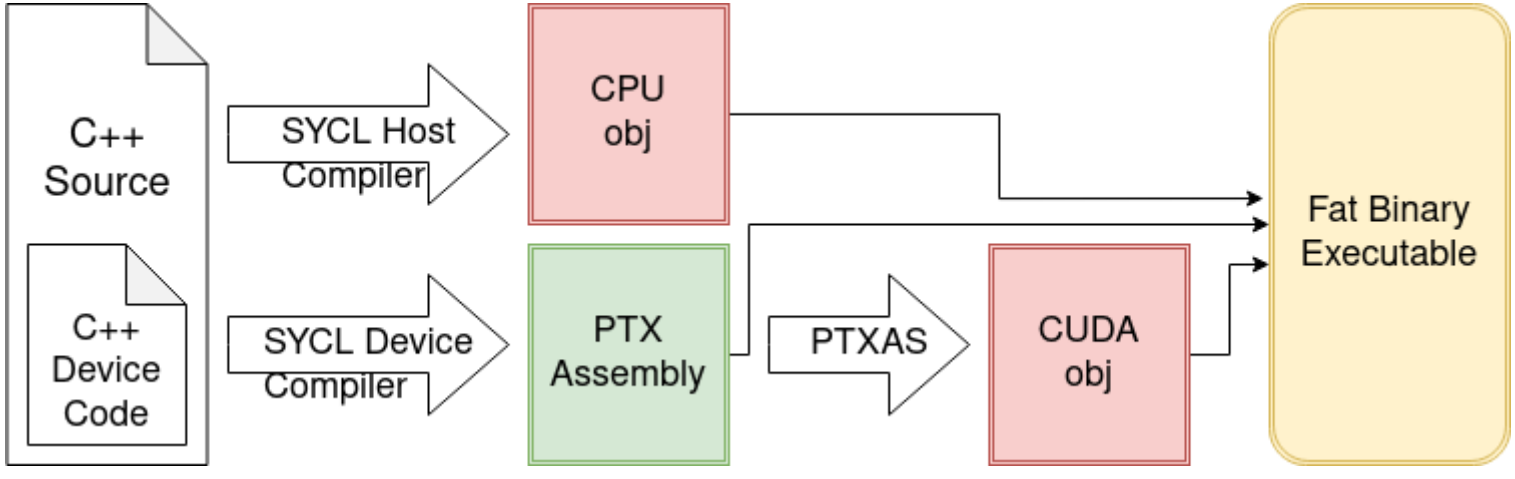
Under the Hood



- DPC++ compilation subprocesses will be printed if the `-###` compiler flag is used.
  - `clang++ -fsycl -fsycl-targets=nvptx64-nvidia-cuda myfile.cpp -###`

DPC++ FOR CUDA

Under the Hood



- Temporary files can be saved by using the `--save-temps` flag. Must be done from within an empty directory.
  - `clang++ -fsycl -fsycl-targets=nvptx64-nvidia-cuda ../myfile.cpp --save-temps`

SPECIFYING THE DEVICE AT RUNTIME

- SYCL_DEVICE_FILTER can be used to select devices at runtime.
  - `SYCL_DEVICE_FILTER=cuda ./a.out`
  - `SYCL_DEVICE_FILTER=host ./a.out`

QUESTIONS

EXERCISE

Code_Exercises/Exercise_1_Hello_SYCL/source.cpp

Configure your environment for using SYCL and compile a source file with the SYCL compiler. Specify the runtime device using `SYCL_DEVICE_FILTER`.

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.