14th International Workshop on OpenCL and SYCL

IWOCL 2026

SYCL Everywhere:
An Open Source Ecosystem for Parallel Computing

Ben Ashbaugh, Intel Corporation

With contributions from many…

May 6-8, 2026 | Heilbronn, Germany | iwocl.org

KHRONOS GROUP

“Why do we think about GPU compute, and not just compute?”

SYCL

“Why do we think about ~~GPU~~ programming

~~compute~~, and not just ~~compute~~ programming?”

Claims and Motivations:

SYCL is still a niche technology:

Need to download special compilers

Need to download special runtimes

Niche-ness is hurting adoption

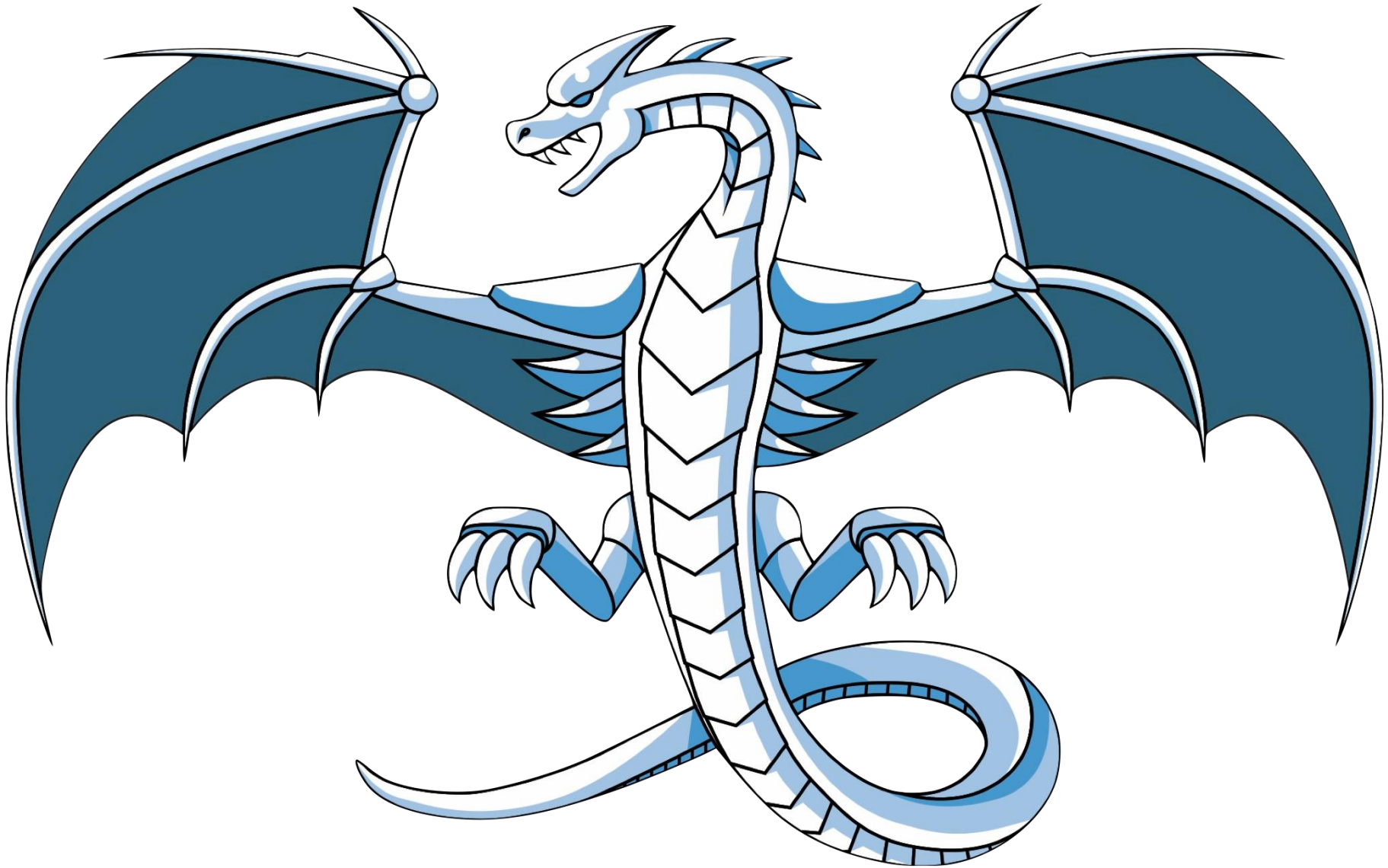
Open source!

SYCL everywhere is a desirable goal

SYCL everywhere is an attainable goal!

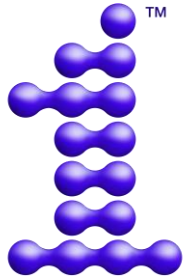
Caveats and Disclaimers

- More SYCL implementations is “a good thing”
  - This will not describe the only way to implement SYCL!
- Focus will be on Khronos technologies
  - This is what I know most about ☺
- These activities are happening now
  - However, it is still early, things can change
  - More participation and voices are welcome!



Clang and LLVM

- Clang and LLVM is a natural place to add SYCL support
- Has a vibrant community and broad availability
- Both mainstream SYCL compilers are Clang and LLVM based



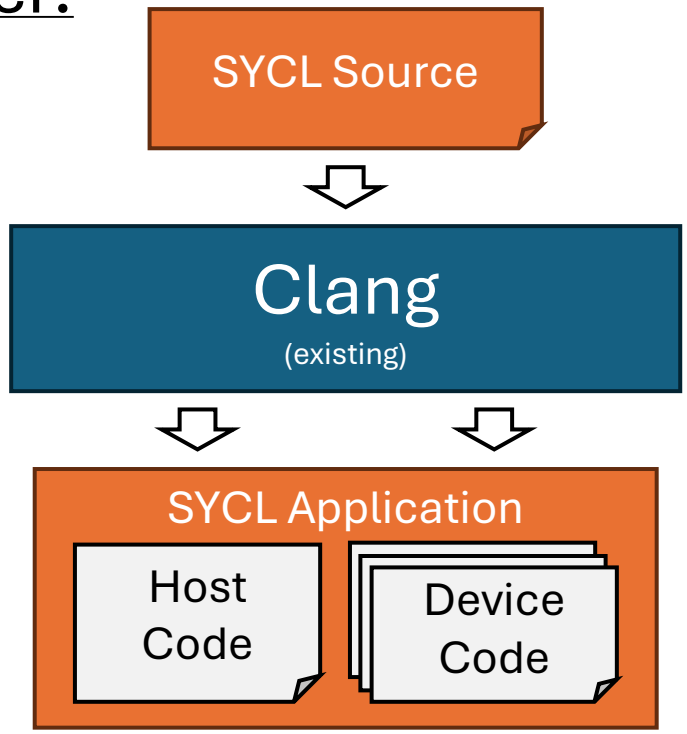
oneAPI

AdaptiveCpp

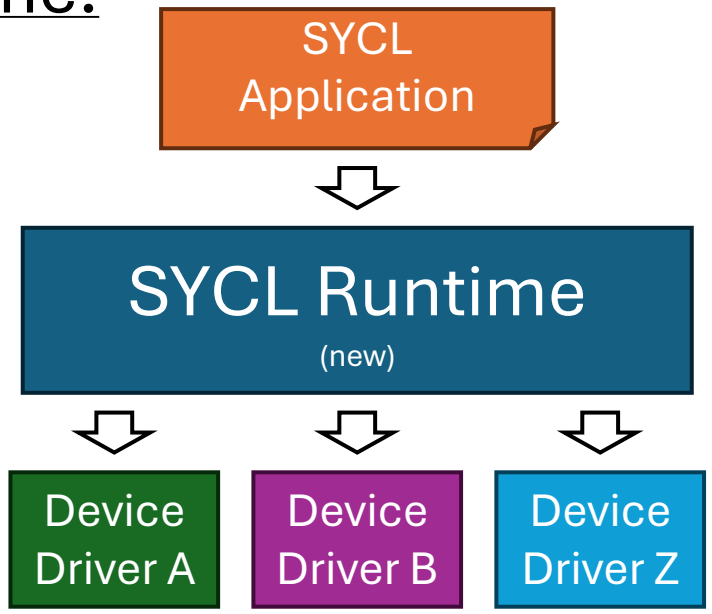
SYCL Clang and LLVM Goals

- SYCL compiler and runtime support in the upstream LLVM repo
  - No forks needed!

Compiler:

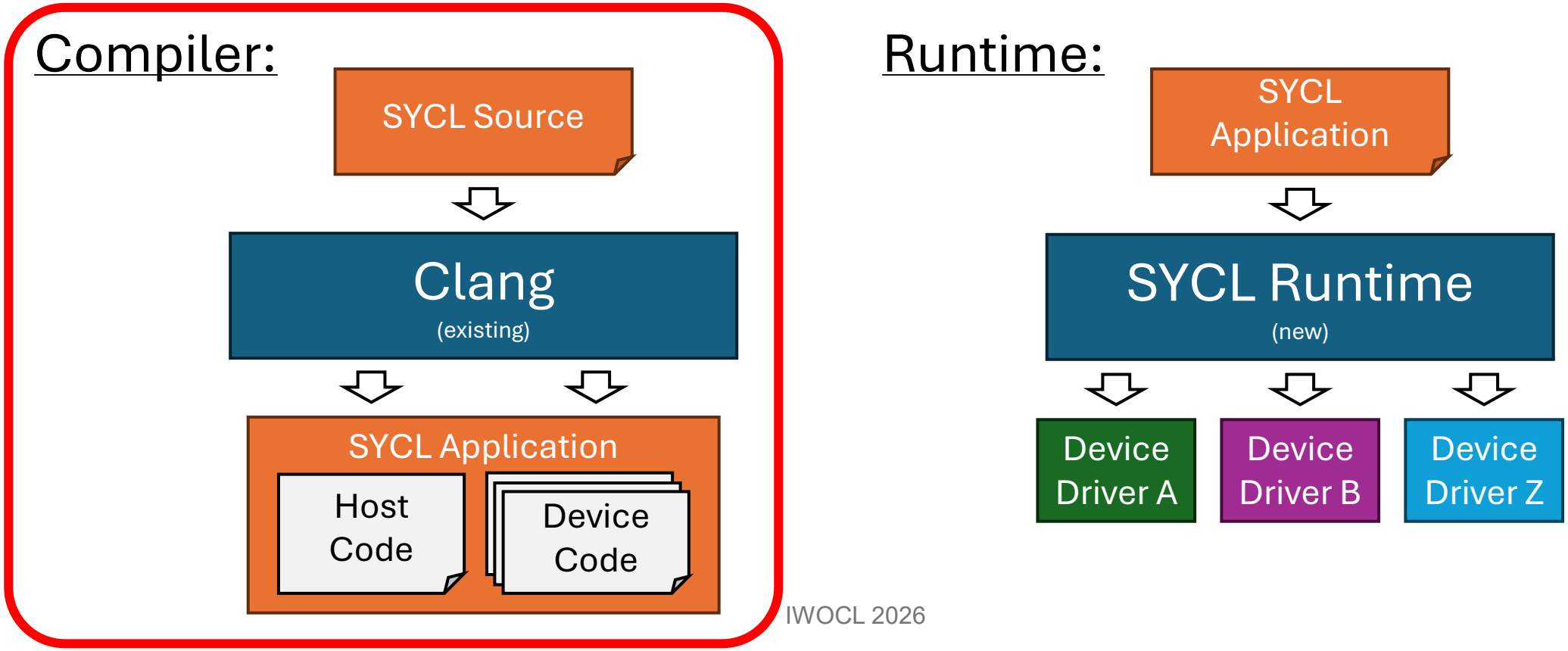


Runtime:



SYCL Clang and LLVM Goals

- Let's start by looking at the SYCL compiler…



Clang Details

- Lots of important details are being designed and implemented…
  - Command line options
  - Identifying and separating host and device code
  - Packaging host code and device code into an executable
  - … and more!

- Focus of this talk is SPIR-V
  - Specifically, the “SPIR-V LLVM backend”

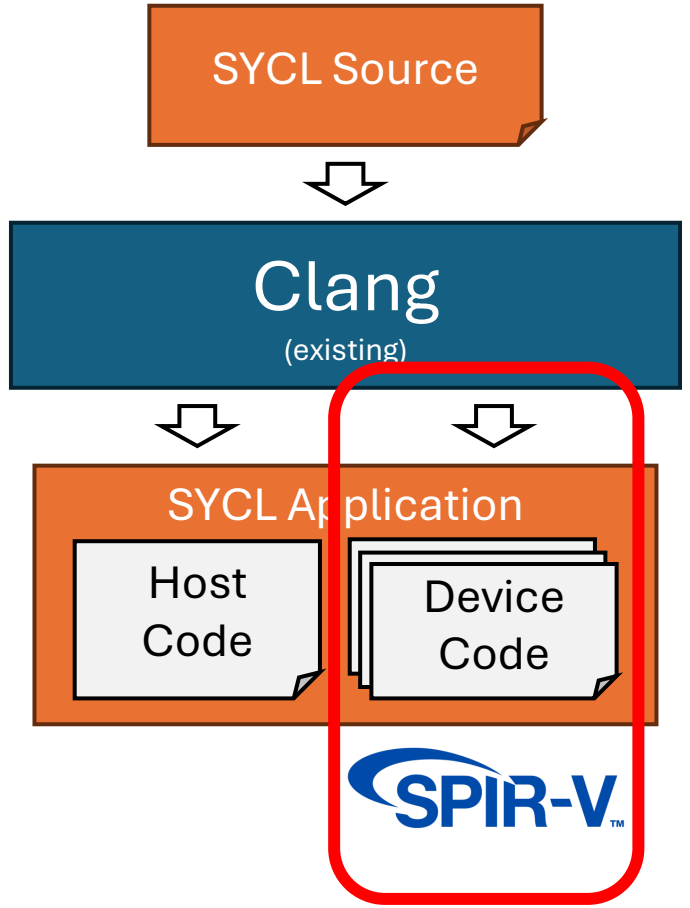
SPIR-V LLVM Backend: What is it?

- An upstream mechanism to generate **SPIR-V** from **LLVM IR**
  - Supported as an official LLVM target since January 2025
- **LLVM IR** is the IR used internally by Clang and LLVM
  - Useful and expressive, but not a stable distribution format
- **SPIR-V** is the **S**tandard **P**ortable **I**ntermediate **R**epresentation
  - A Khronos standard IR to represent device code
  - Stable, accepted by device compilers from many vendors

SPIR-V LLVM Backend

- Provides a mechanism to generate standard device code

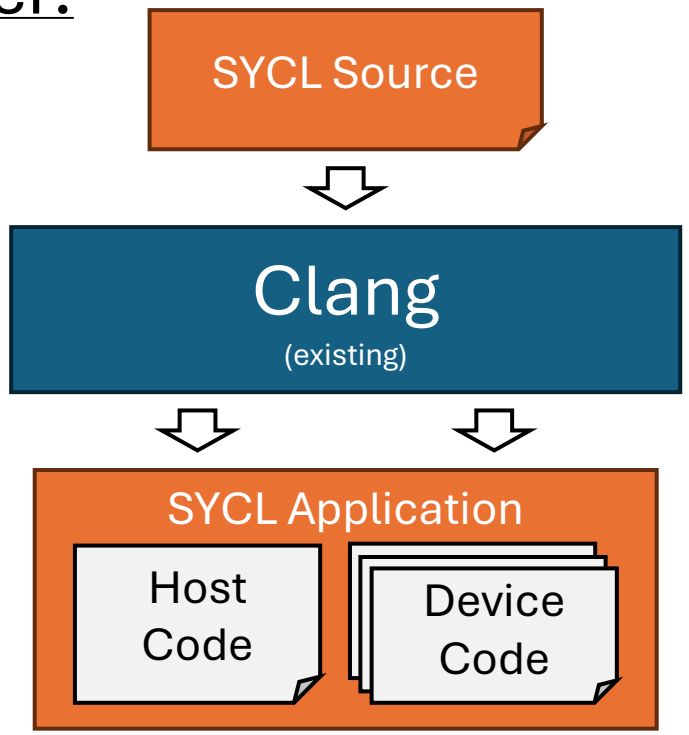
- Notes:
  - **A** mechanism, not the only mechanism! Some devices may support other device code formats
  - The SPIR-V LLVM backend is useful for more than just SYCL!



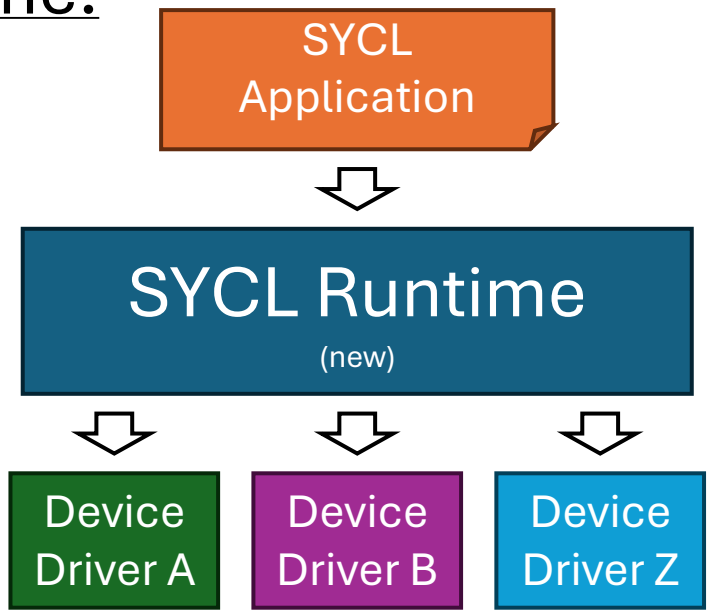
SYCL Clang and LLVM Goals

- Let's shift focus to the SYCL runtime…

Compiler:



Runtime:



Runtime Details

- LLVM has an offload runtime!
  - Used for OpenMP offload - **libomptarget**
  - Supports devices from multiple vendors
  - Can this be repurposed to execute SYCL applications?

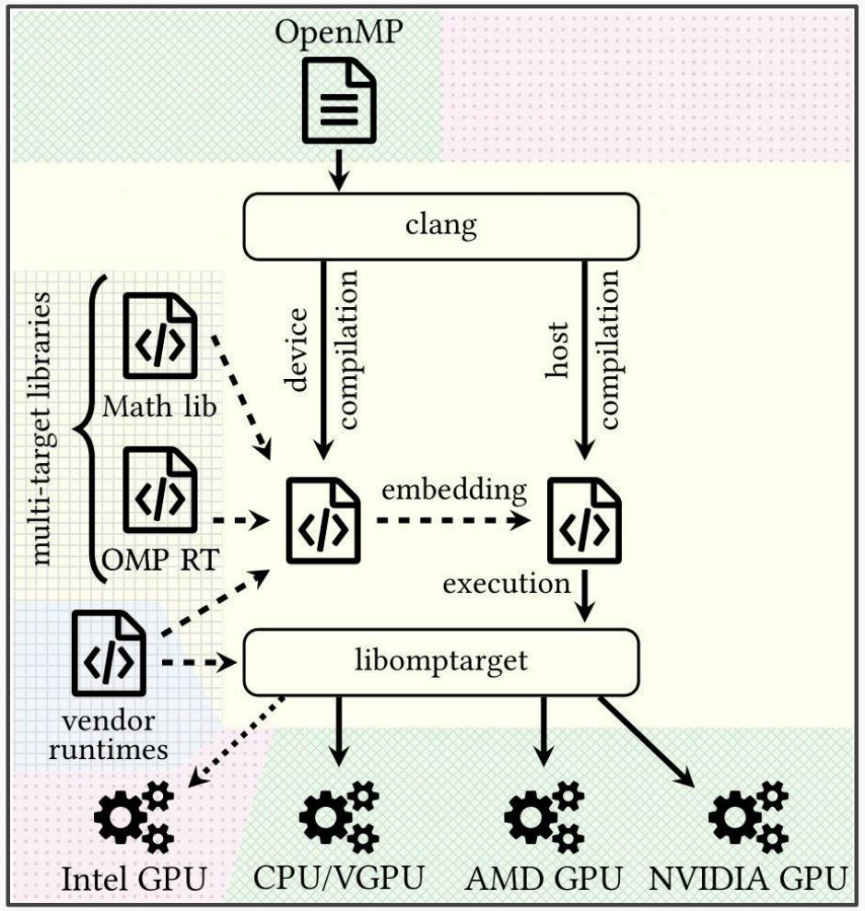
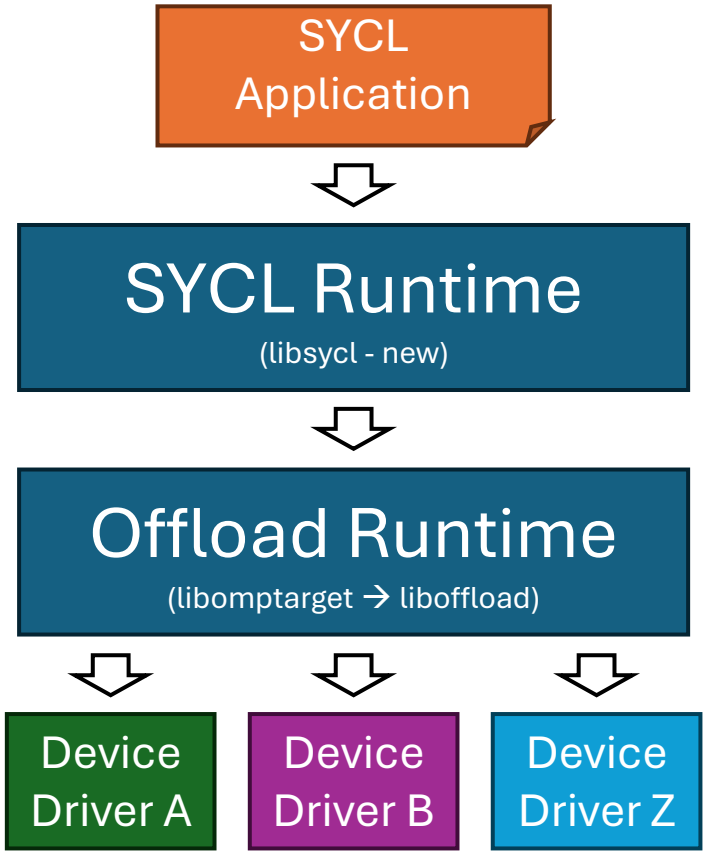


Diagram source: https://llvm.org/devmtg/2023-05/slides/QuickTalks-May10/04 - EuroLLVM 23 - OpenMP Kernel Language.pdf

Runtime Details

- Short answer: Yes!

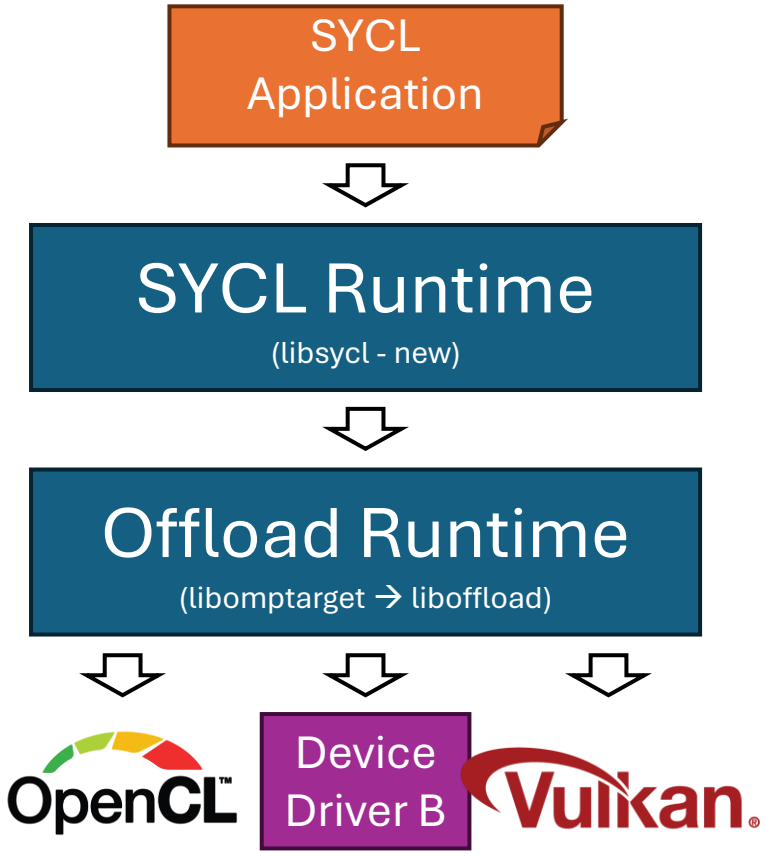
- **libomptarget** becomes **liboffload**
  - A shared offload runtime for OpenMP, SYCL, and more
  - Continues to support devices from multiple vendors
  - Continues to support multiple device types (GPUs, CPUs, FPGAs, and more)
  - Via multiple driver APIs



Runtime Details: Khronos APIs

- There are several “obvious” Khronos APIs that liboffload could target
  - OpenCL – likely future target
  - Vulkan – possible future target?

- Note:
  - Both OpenCL and Vulkan need extensions for full SYCL support
  - Both APIs consume SPIR-V! (kind of)



Current Status

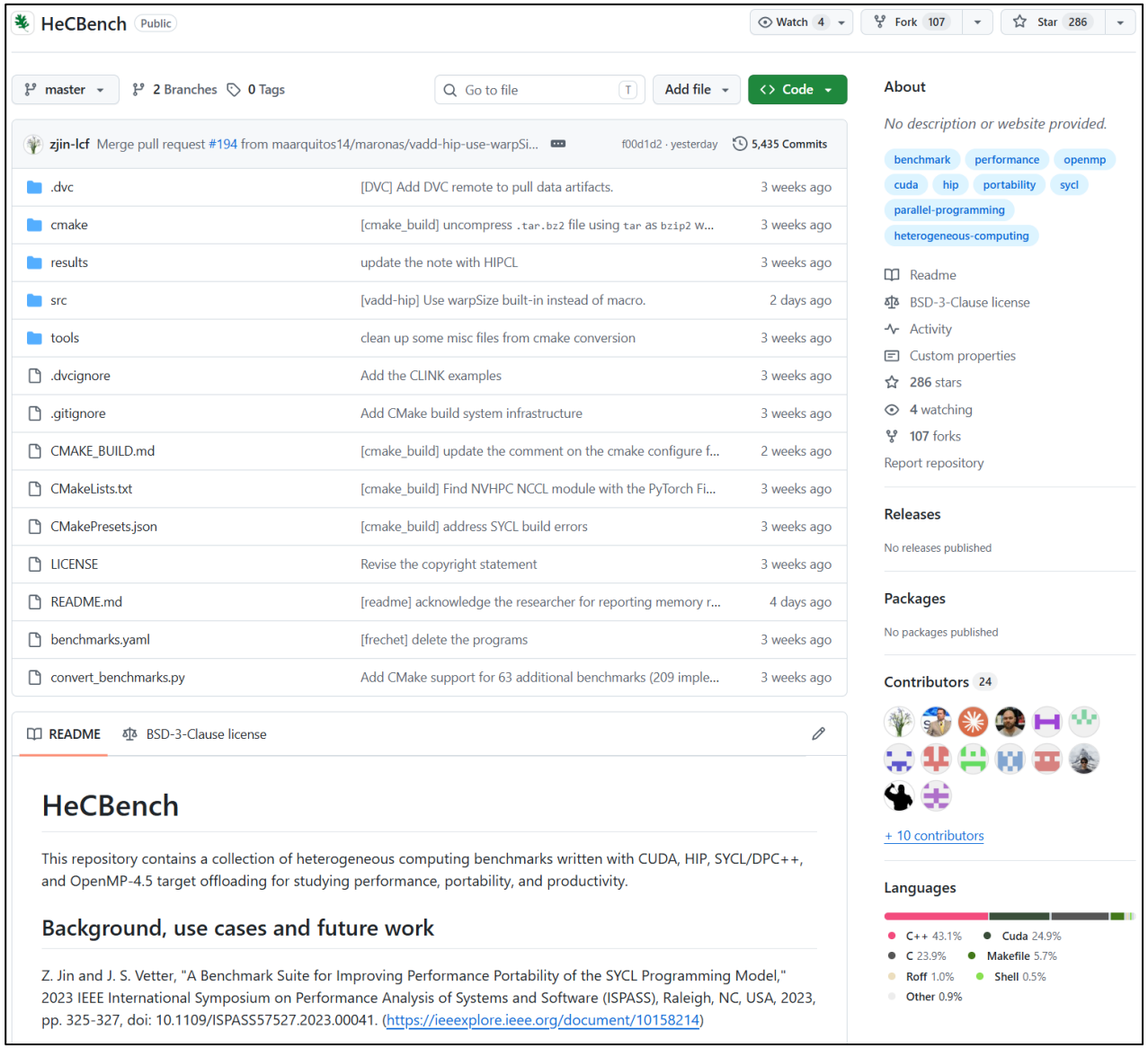
Current Status (as of April 2026):

- Have community agreement, development has started
- Enabling SYCL “hello, world”
  - Compiling to SPIR-V
  - Running on an Intel GPU
  - On both Windows and Linux
- Not *quite* working yet…
  - But making lots of progress!
  - Expecting to work “soon”

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

int main() {
  //  Create a queue to enqueue work for the default device
  sycl::queue myQueue;

  // Allocate memory for the device associated with the queue
  int* data = sycl::malloc_shared<int>(1024, myQueue);

  myQueue.parallel_for(1024, [=](sycl::id<1> idx) {
    // Initialize each element with an index starting at zero
    data[idx] = idx;
  });

  // Explicitly wait for kernel execution to complete
  myQueue.wait();

  // Print results
  for (int i = 0; i < 1024; i++)
    std::cout << "data[" << i << "] = " << data[i] << std::endl;

  // Free memory
  sycl::free(data, myQueue);
  return 0;
}
```

What's next?

- Add more SYCL features!
  - Prioritizing features used by real-world applications (HeCBench)
- Evaluate performance
  - Especially through liboffload
- Eventually, possibly:
  - SYCL Conformance?



One more thing…

Open Source Drivers

- Can’t have a fully open source solution without open source drivers…
- To the vendors that have open sourced their drivers – thank you!
- To the developers working on Mesa and PoCL open source implementations – thank you!

- Even without full open source solutions, we can still have a *mostly* open source solution built upon standards (which is pretty good!)

Wrap Up

Summary and Calls to Action

- SYCL is coming to Clang and LLVM!
  - Not a science project, work is happening right now
- For device vendors:
  - Contribute to the SPIR-V LLVM backend and liboffload
  - Enabling SYCL on your device may be easier than you think!
- For the Khronos SPIR-V and OpenCL (and Vulkan?) working groups:
  - Prioritize features used by the SPIR-V LLVM backend and liboffload
  - High ROI development – will be useful for more than just SYCL
- For application developers and SYCL fans:
  - Review, test, and advocate for SYCL in the LLVM community!

Thank you!