codeplay®

SYCL - a modern C++ programming model for accelerators

Lukas Sommer

CODAI Workshop – 21st September 2023

codeplay

Enabling AI & HPC To Be Open, Safe & Accessible To All

Established 2002 in **Edinburgh, Scotland**.

Grown successfully to around 100 employees.

In 2022, we became a **wholly owned subsidiary** of Intel.

oneAPI

Committed to expanding the **open ecosystem** for heterogeneous computing.

Through our involvement in oneAPI and SYCL governance, we help to **maintain and develop** open standards.

Developing at the forefront of **cutting-edge research**.

Currently involved in two research projects - **SYCLOPS** and **AERO**, both funded by the Horizon Europe Project.

Agenda

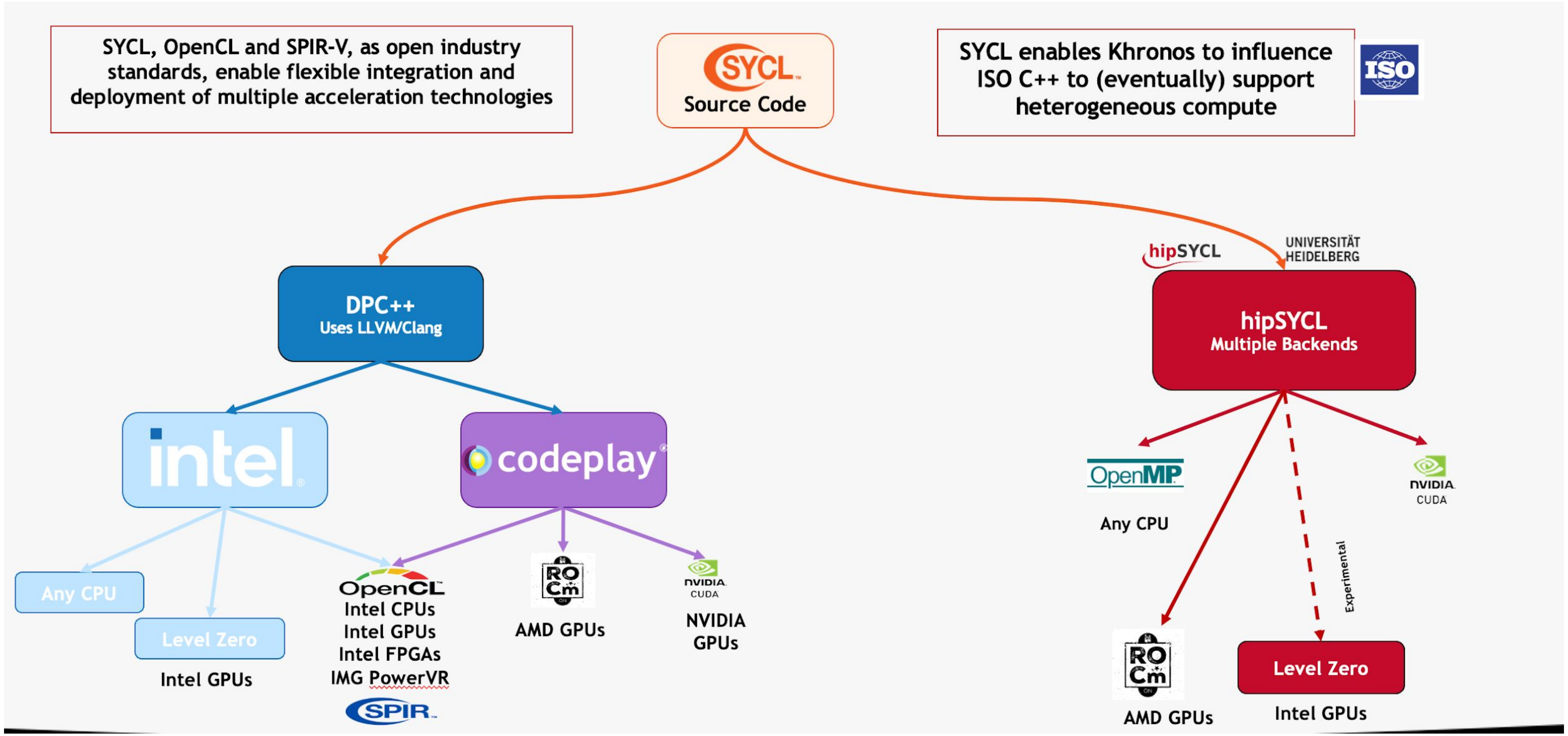
- What is SYCL?
- How can (edge) accelerators benefit from SYCL?
- How can we create better compilers for SYCL?

What is SYCL?

What is SYCL?

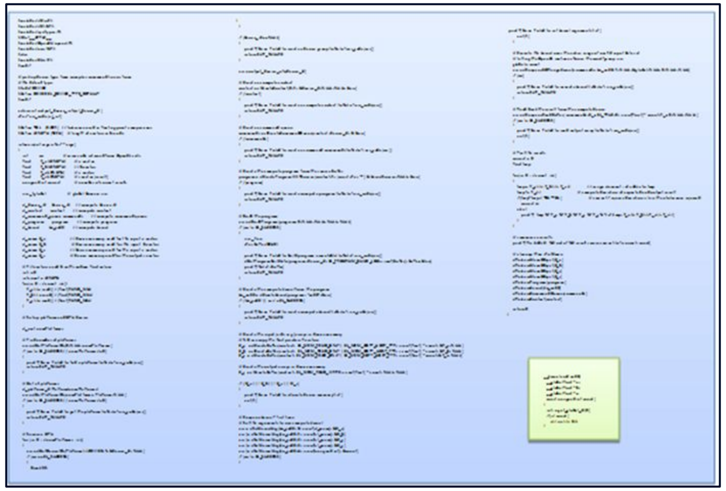
- An **open standard** heterogenous programming API introduced by Khronos
- Provides **single-source** programming model for accelerator processors
- Using ISO **standard C++** code

SYCL Implementations



Why SYCL?

- SYCL provides high-level abstractions over common boiler-plate code
  - Platform/device selection
  - Buffer creation and data movement
  - Kernel function compilation
  - Dependency management and scheduling



Typical OpenCL hello world application

Typical SYCL hello world application

Vector Add in SYCL

```
#include <sycl/sycl.hpp>
using namespace sycl;

int main() {

  std::vector<float> a{…}, b{…}, c{…};

  queue q;














}
```

Create device work queue

Vector Add in SYCL

```
int main() {
  std::vector<float> a{…}, b{…}, c{…};
  queue q;
  {
    buffer<float> bufA{a}, bufB{b}, bufC{c};









  }
}
```

Create data buffers

Vector Add in SYCL

```
int main() {
  std::vector<float> a{…}, b{…}, c{…};
  queue q;
  {
    buffer<float> bufA{a}, bufB{b}, bufC{c};
    q.submit([&](handler &cgh) {
      accessor accA{bufA, cgh, read_only};
      accessor accB{bufB, cgh, read_only};
      accessor out{bufC, cgh, write_only, no_init};

    });
  }
}
```

Specify access and requirements

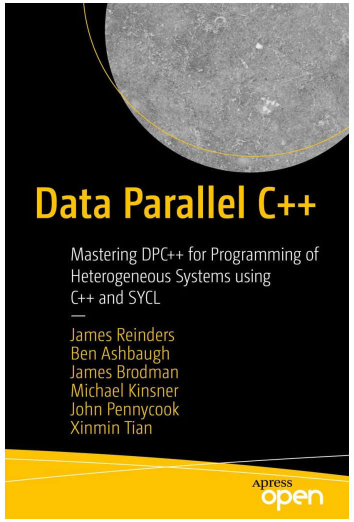
Vector Add in SYCL

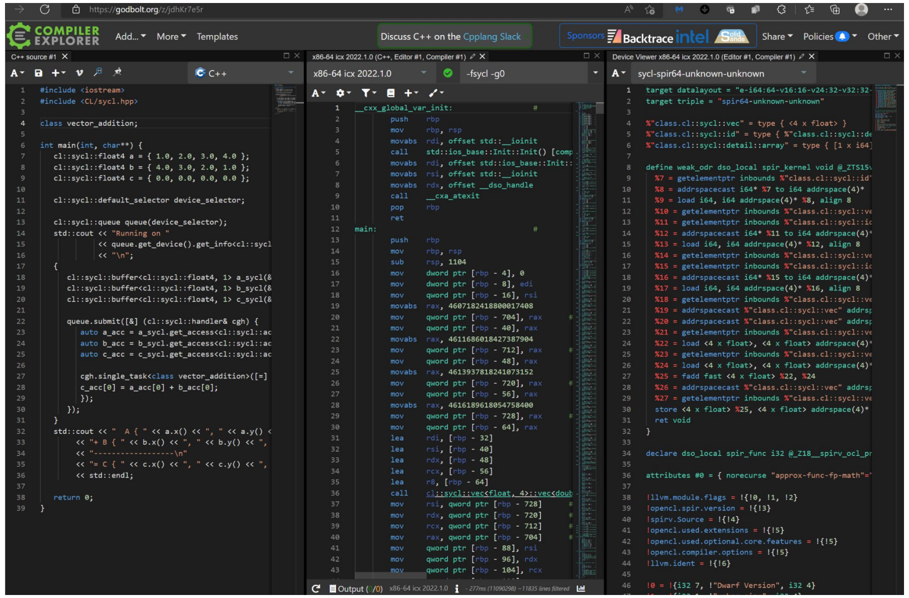
```
int main() {
  std::vector<float> a{…}, b{…}, c{…};
  queue q;
  {
    buffer<float> bufA{a}, bufB{b}, bufC{c};
    q.submit([&](handler &cgh) {
      accessor accA{bufA, cgh, read_only};
      accessor accB{bufB, cgh, read_only};
      accessor out{bufC, cgh, write_only, no_init};
      cgh.parallel_for<class add>(a.size(),
          [=](id<1> i) { out[i] = accA[i] + accB[i]; });
    });
  }
}
```

Submit data-parallel device kernel

More about SYCL

- Khronos-backed website for all things SYCL: https://sycl.tech/
- Free ebook to learn SYCL: https://tinyurl.com/sycl-book
- Try out on compiler explorer: https://godbolt.org/z/jdhKr7e5r



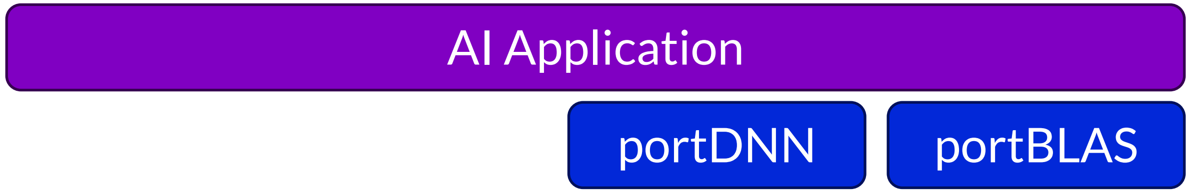


How to run AI with SYCL

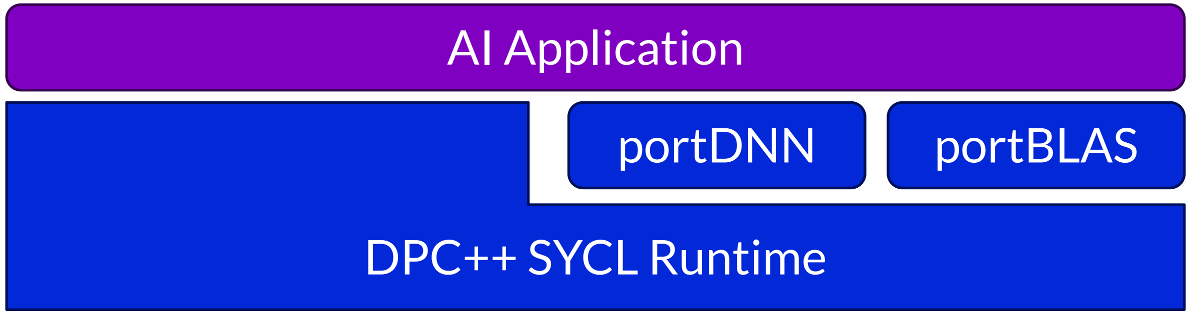
The oneAPI Software Stack for SYCL

AI Application

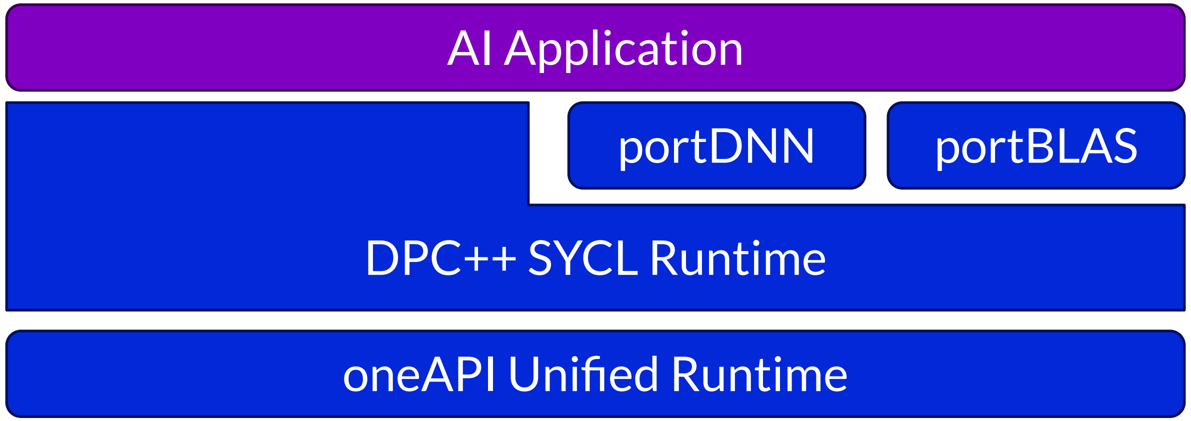
The oneAPI Software Stack for SYCL



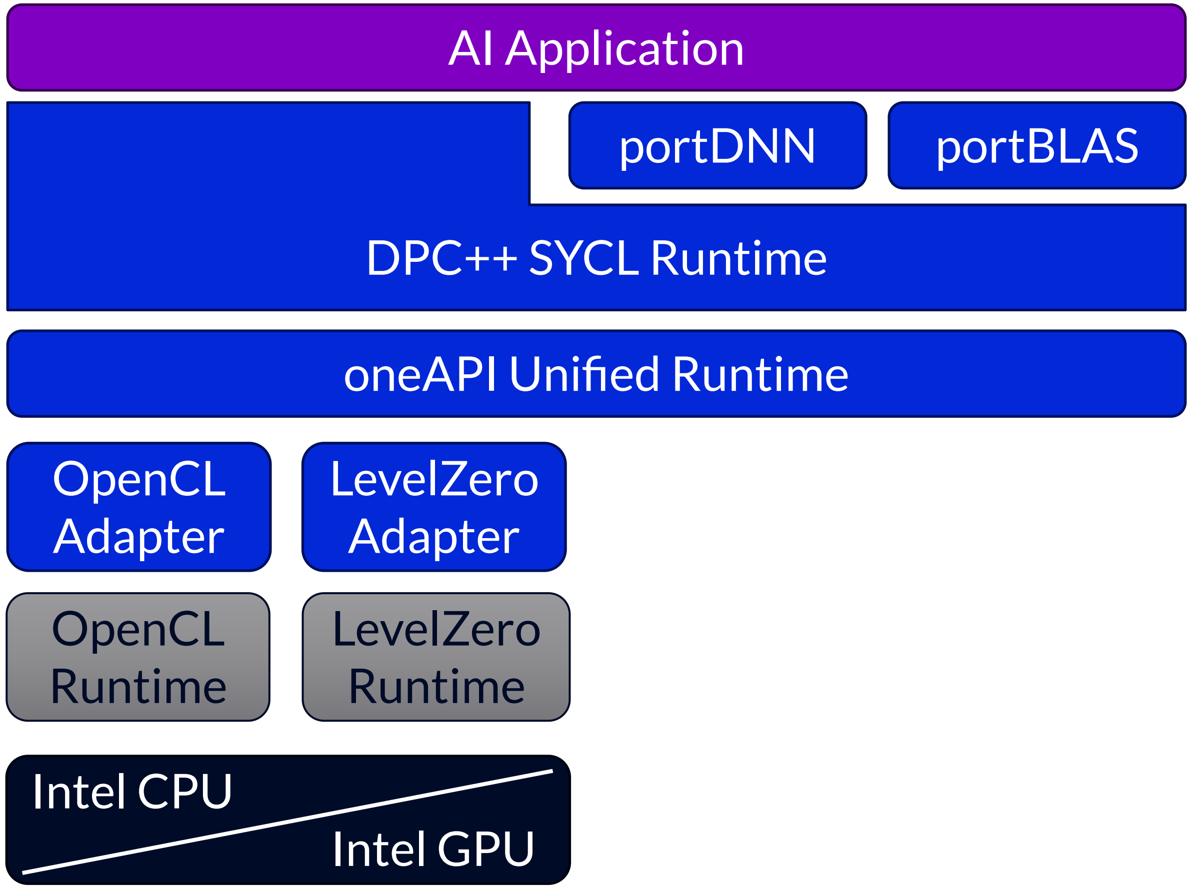
The oneAPI Software Stack for SYCL



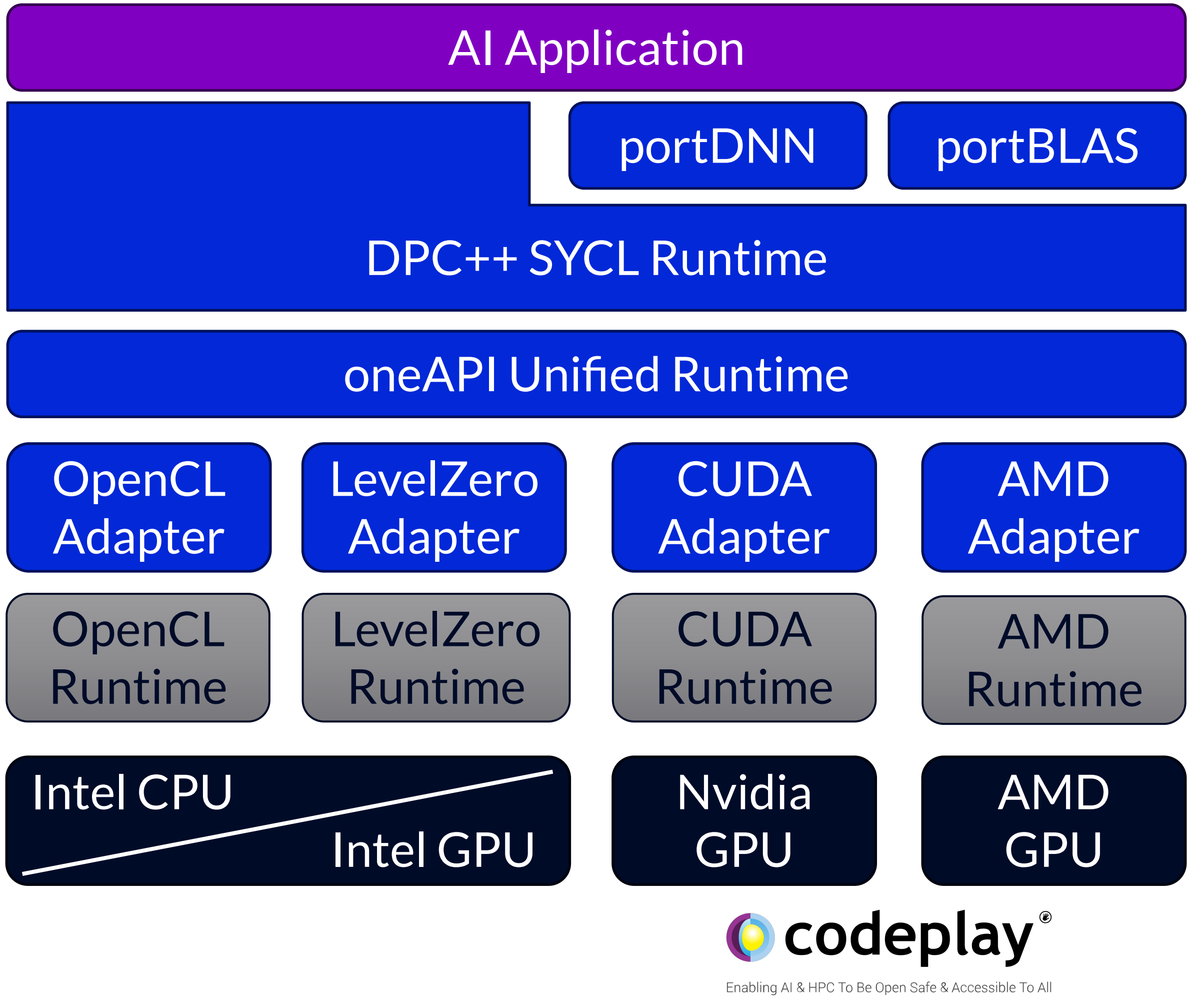
The oneAPI Software Stack for SYCL



The oneAPI Software Stack for SYCL

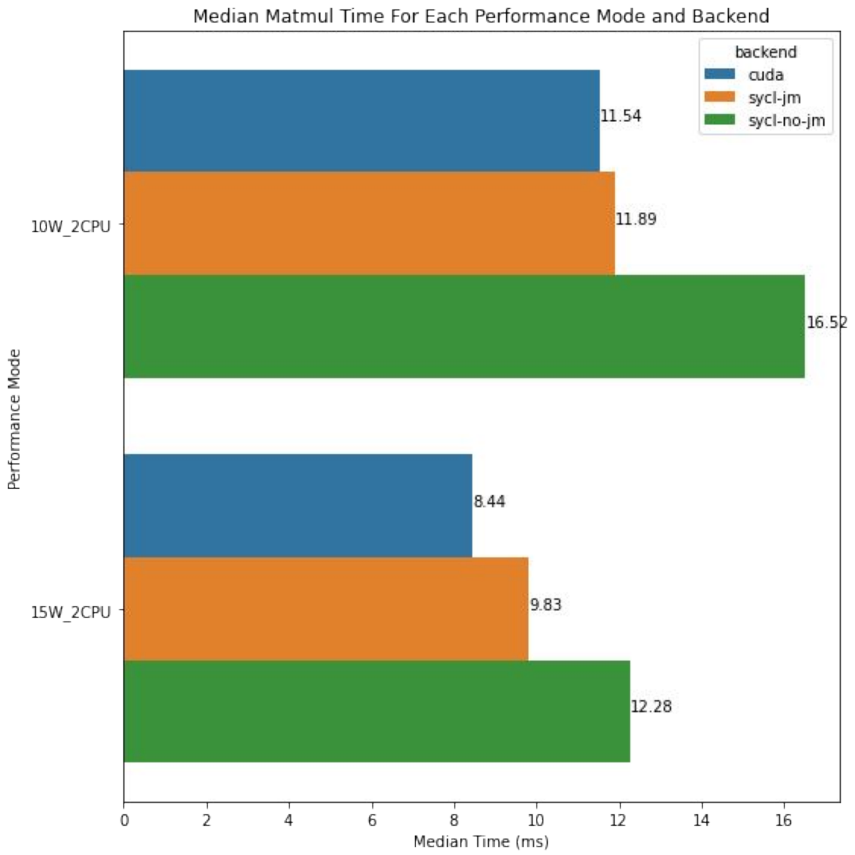


The oneAPI Software Stack for SYCL



SYCL on CUDA edge devices

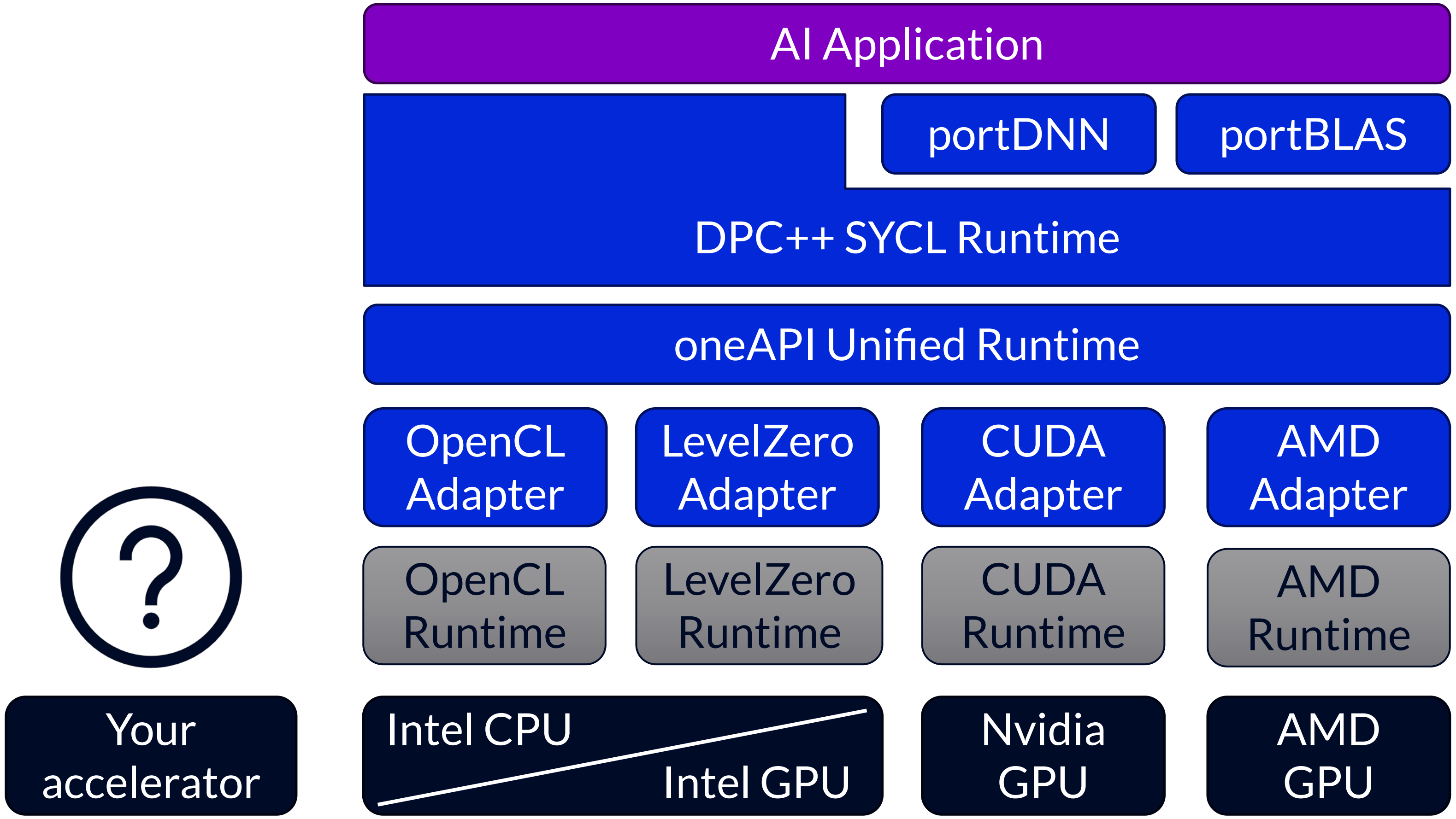
- Application: PointNet NN model to classify point cloud in scene understanding
- SYCL operator implementation through ONNX runtime
- Use oneAPI CUDA plugins to run on Nvidia Jetson NX edge GPU
- Performance:
  - 97% of CUDA performance in 10W mode
  - 86% of CUDA performance in 15W mode



Dylan Angus, Svetlozar Georgiev, Hector Arroyo Gonzalez, James Riordan, Paul Keir, and Mehdi Goli. 2023. Porting SYCL accelerated neural network frameworks to edge devices. In Proceedings of the 2023 International Workshop on OpenCL (IWOCL '23). https://doi.org/10.1145/3585341.3585346

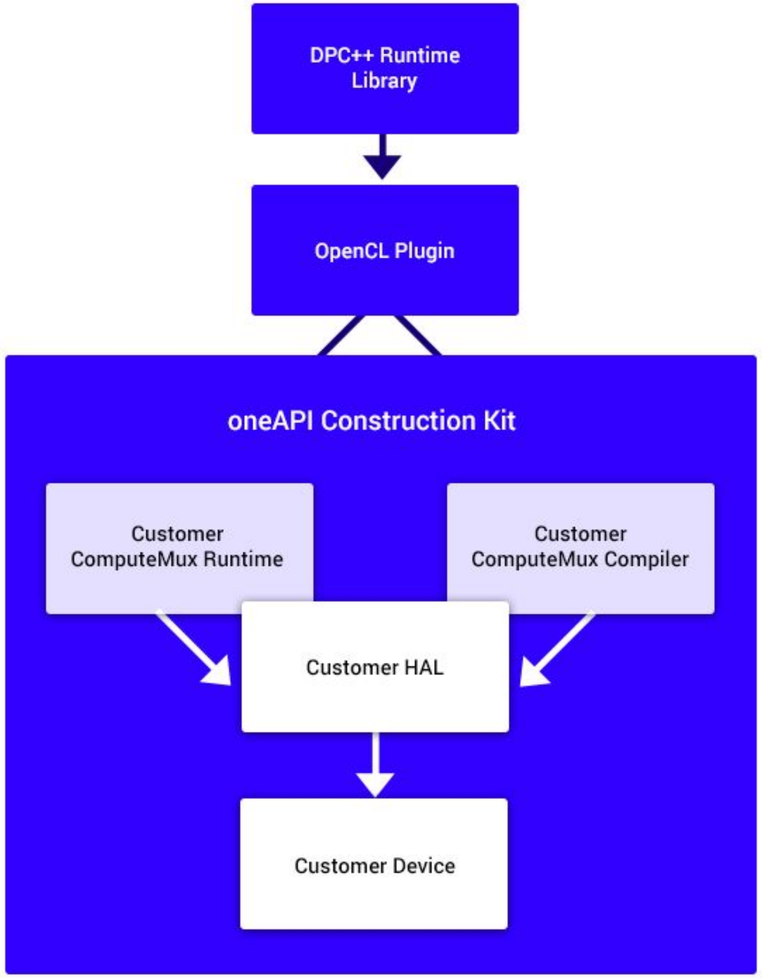
Performance varies by use, configuration and other factors. See the paper for workloads and configurations. Results may vary

The oneAPI Software Stack for SYCL

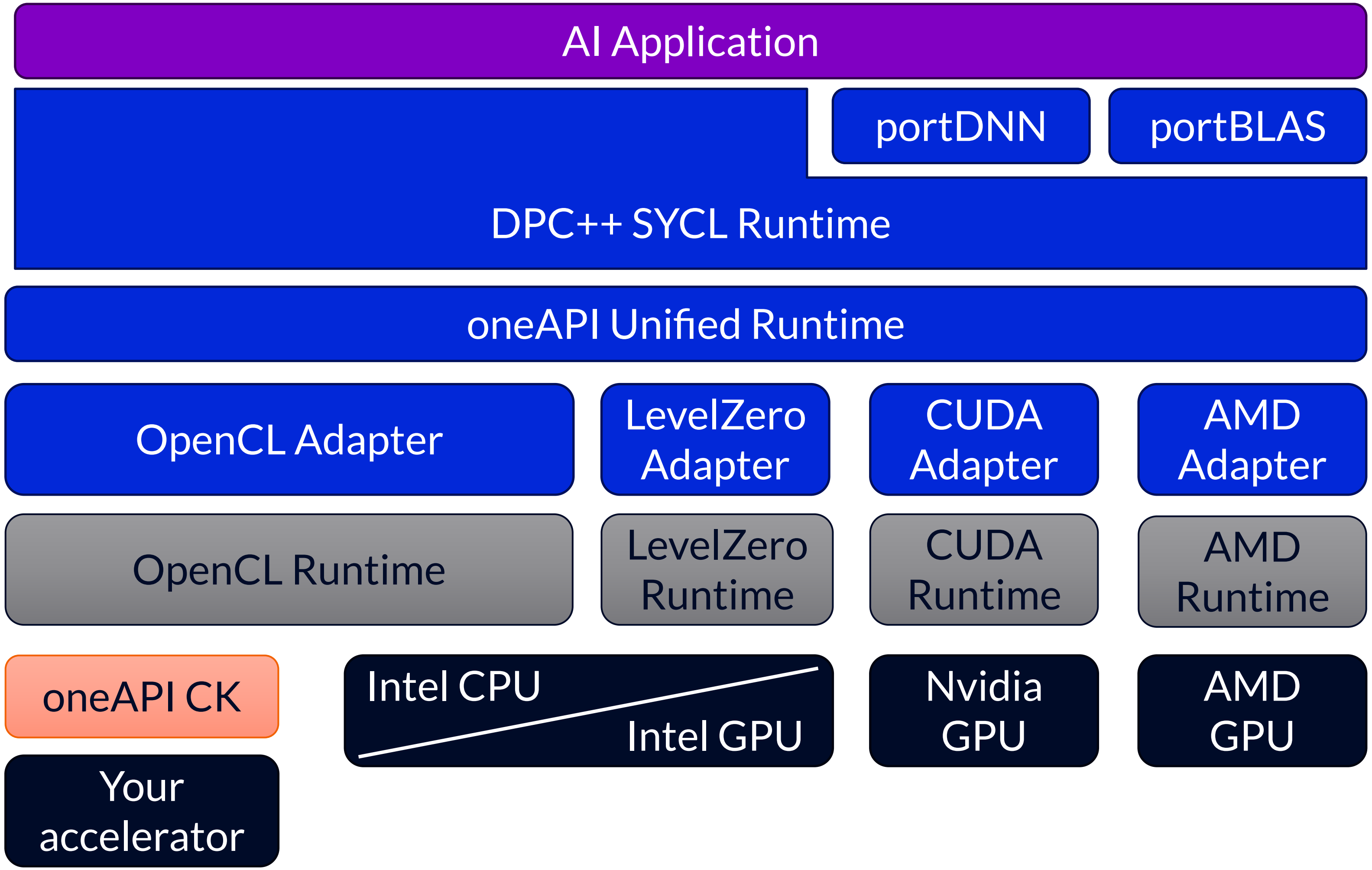


oneAPI Construction Kit

- oneAPI Construction Kit enables integration of custom accelerators into the oneAPI software stack
- Only need to provide
  - Runtime component
    - e.g., data movement between host and accelerator
  - Device binary compiler
    - Sufficient to compile from SPIR-V to accelerator binary
    - Prior compilation from SYCL to SPIR-V handled by DPC++ compiler
- oneAPI Construction Kit is open-source!



The oneAPI Software Stack for SYCL



New Working Group within KHRONOS GROUP
Created March ‘23

Why?

- Safety-critical industries (automotive, avionics, medical, etc.) increasingly require *acceleration* of software, due to
  - Rising popularity of **AI** algorithms
  - Proliferation of **heterogeneous** computing
  - Increasing demand for **performance**

What?

- Based on SYCL 2020
- Modifications to ease safety-certification
  - Of the implementation of the standard
  - Of the SYCL application

Simplified
Runtime can be more easily certified

Robust
Comprehensive error handling
Removal of ambiguity
Clarification of undefined behaviour

SYCL SC™

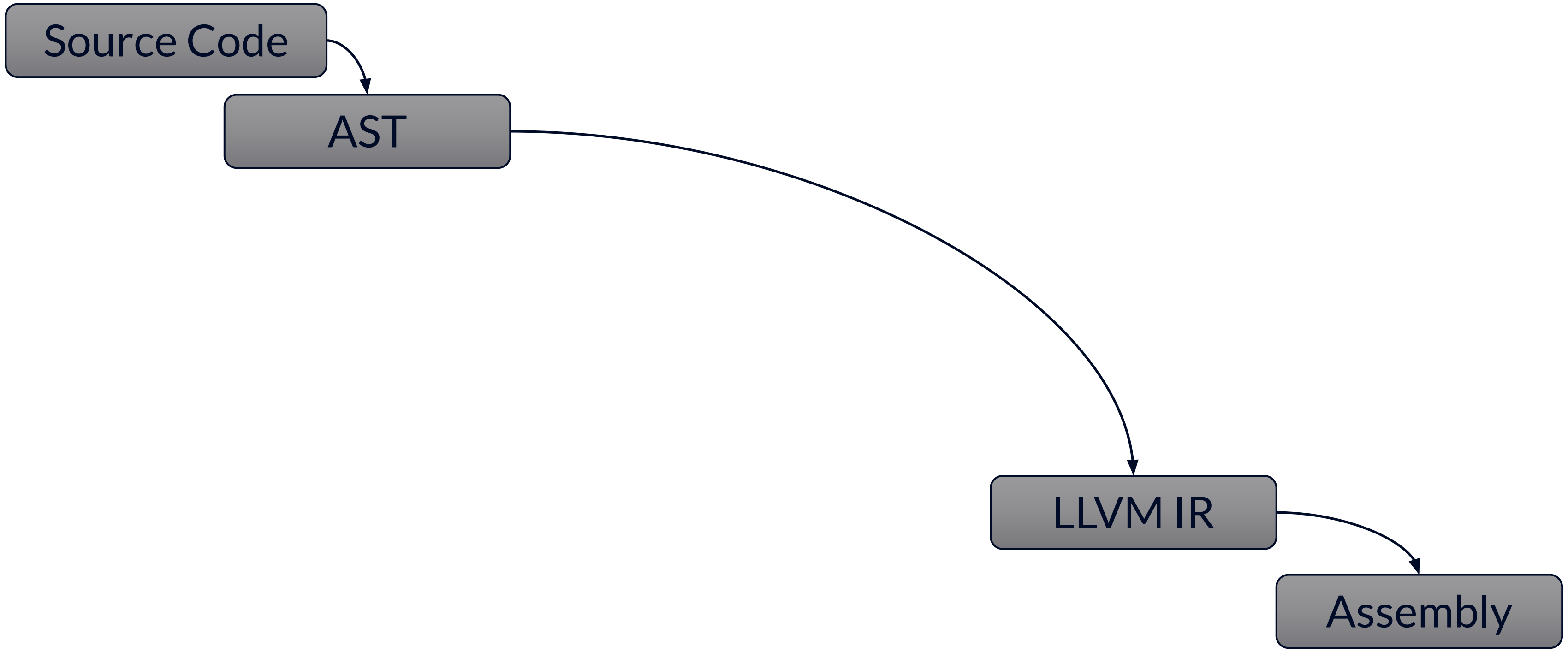
Deterministic
Predictable execution time
Predictable results

Industry safety-critical standards include
RTCA DO-178C (avionics) | ISO 26262 (automotive)
IEC 61508 (industrial) | IEC 62304 (medical)

Interested?
Visit https://www.khronos.org/syclsc
Contact sycl_sc-chair@lists.khronos.org
Join the Working Group

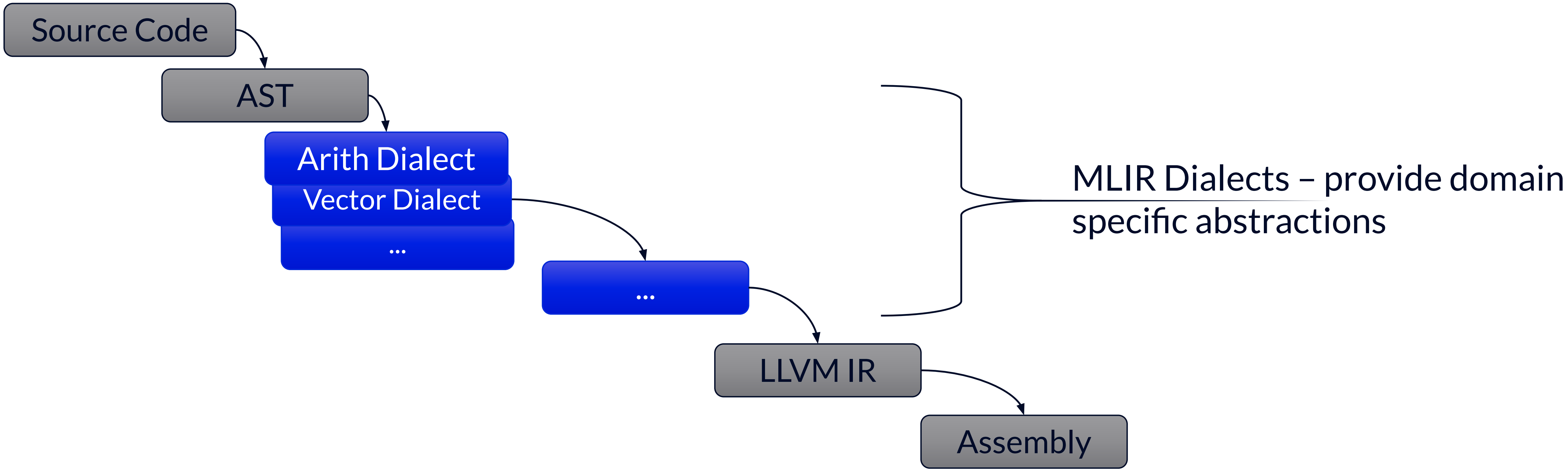
How can we improve SYCL compilation

Motivation



- Huge translation step to LLVM IR loses much high-level semantics
- Many languages have already introduced intermediate step

MLIR to the Rescue



- Gradual lowering in more, but smaller steps
- Transformations leverage domain-specific semantics of dialects – the right abstraction level
- High-level semantics is preserved and available for analysis & transformation

SYCL-MLIR Project Overview

- Aim: Better optimisations for SYCL compilers
  - Better optimisation for device code
  - Optimisation across the border between host and device code
- LLVM IR is just not enough
  - Too low-level for some advanced optimizations
  - Currently no way of representing host and device code in one module
- MLIR is better suited
  - Benefit from higher-level abstractions and gradual lowering
  - Ability to nest device code inside host

➡ Build an MLIR-based SYCL compiler

SYCL-MLIR Approach

- Define an MLIR dialect for SYCL
- Capture semantics and key abstractions
  - Data access
  - Parallel semantics
- Represent host and device code in the same MLIR module
  - Analyse host code to get context for device optimizations

Optimisation example

```
for(size_t k=0; k<M; ++k)
  R[item] += <expr(k)>;
```

➡

```
DATA_TYPE R_reduction = R[item];
for(size_t k=0; k<M; ++k)
  R_reduction += <expr(k)>;
R[item] = R_reduction;
```

- Goal: replace uses of `R[item]` by a reduction variable

Example: LLVM sees function calls

```
for(size_t k=0; k<M; ++k)
  R[item] += <expr(k)>;
```

→

```
DATA_TYPE R_reduction = R[item];
for(size_t k=0; k<M; ++k)
  R_reduction += <expr(k)>;
R[item] = R_reduction;
```

```
  call spir_func void
@_ZN4sycl3_V12idILi1EEC2ILi1ELb1EEERNSt9enable_ifIXeqT_Li1EEKNS0_4itemILi1EXT0_EEEE4typeE(%"class.sycl::_V1::id"
addrspace(4)* noundef align 8 dereferenceable_or_null(8) %agg.tmp3.ascast, %"class.sycl::_V1::item" addrspace(4)*
noundef align 8 dereferenceable(24) %item.ascast) #11
  %agg.tmp3.ascast.ascast = addrspacecast %"class.sycl::_V1::id" addrspace(4)* %agg.tmp3.ascast to
%"class.sycl::_V1::id"*
  %call4 = call spir_func noundef align 4 dereferenceable(4) i32 addrspace(4)*
@_ZNK4sycl3_V18accessorIiLi1ELNS0_6access4modeE1026ELNS2_6targetE2014ELNS2_11placeholderE0ENS0_3ext6oneapi22accessor_
property_listIJEEEEixILi1EvEERiNS0_2idILi1EEE(%"class.sycl::_V1::accessor" addrspace(4)* noundef align 8
dereferenceable_or_null(32) %R2, %"class.sycl::_V1::id"* noundef byval(%"class.sycl::_V1::id") align 8
%agg.tmp3.ascast.ascast) #11
```

Example: SYCL-MLIR encodes semantic

```
for(size_t k=0; k<M; ++k)
  R[item] += <expr(k)>;
```

→

```
DATA_TYPE R_reduction = R[item];
for(size_t k=0; k<M; ++k)
  R_reduction += <expr(k)>;
R[item] = R_reduction;
```

```
sycl.constructor @id(%id, %item)
%R_item_ptr = sycl.accessor.subscript %accessor[%id]
%R_item = affine.load %R_item_ptr[0] : memref<?xf32, 4>
%0 = arith.addf %R_item, <expr(k)> : f32
affine.store %0, %R_item_ptr[0] : memref<?xf32, 4>
```

Conclusion

Conclusion

- SYCL enables productive heterogeneous programming
  - Leverage modern C++ to reduce boilerplate
  - Open standard, portable across vendors and architectures
- Open nature of oneAPI ecosystem allows integration of custom accelerators
  - Open-source oneAPI Construction Kit as starting point
  - Benefit from the rich ecosystem
- Better representations will enable better compiler optimizations
  - MLIR capture parallel semantics and makes it available to transformations

oneAPI Construction Kit

Scan QR code or visit **developer.codeplay.com** codeplay®

Disclaimers

A wee bit of legal

Performance varies by use, configuration and other factors.

Performance results are based on testing as of dates shown in configurations and may not reflect all publicly available updates. See backup for configuration details.

No product or component can be absolutely secure.

Your costs and results may vary.

Intel technologies may require enabled hardware, software or service activation.