



SPIR-V Requirements for DPC++ / SYCL

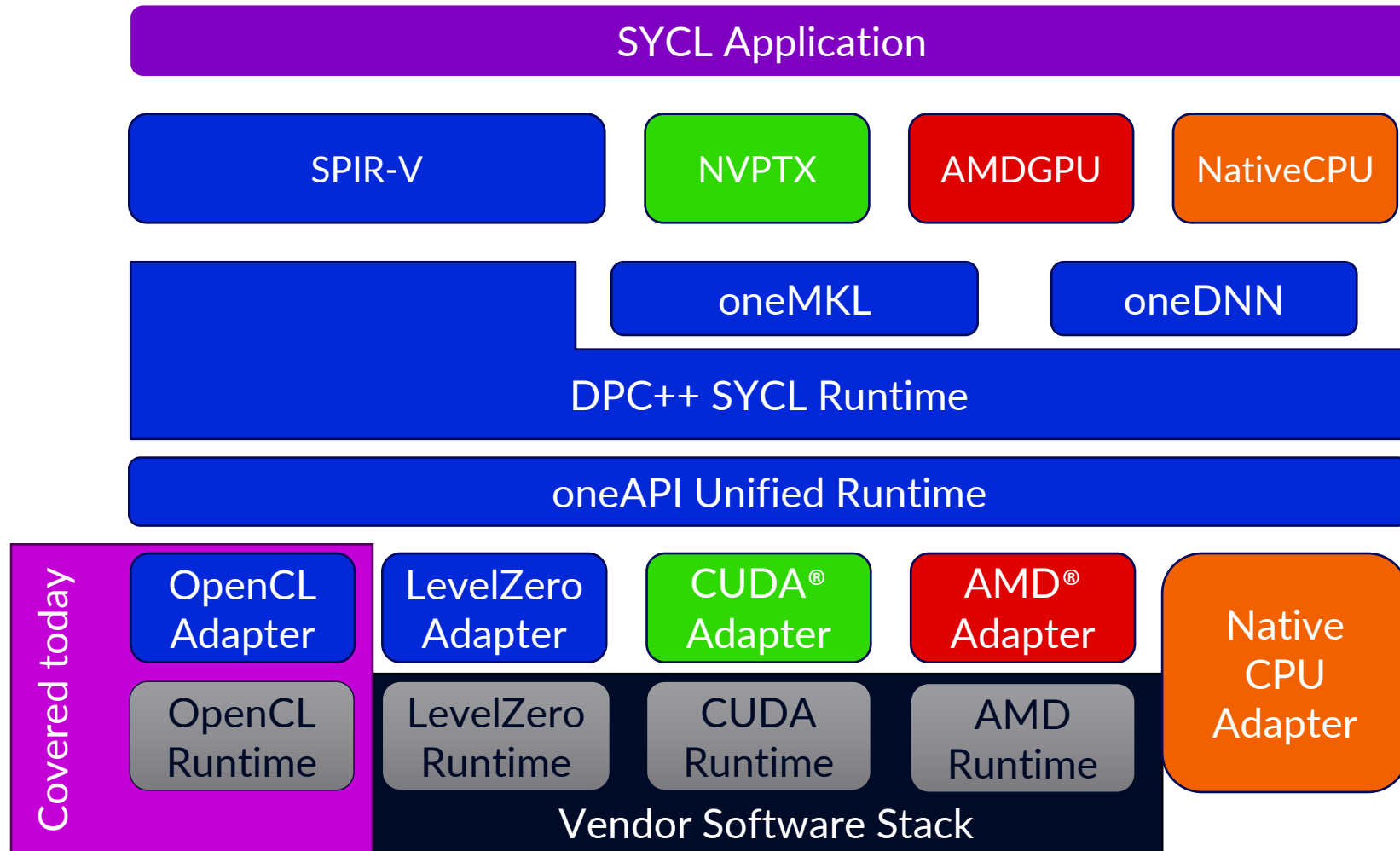
Victor Lomüller – Principal Software Engineer

2024-11-05

Agenda

- Overview of the SYCL SW stack
- What is SPIR-V and why is important
- What does an OpenCL + SPIR-V platform need to support to enable SYCL

Overview of the oneAPI Software Stack for SYCL




SYCL vector addition example

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

```
// A, B and C are USM pointers
```

```
void vecAdd(const int *A, const int *B, int *C, size_t size) {  
    sycl::queue deviceQueue;  
    deviceQueue.submit([&](sycl::handler &cgh) {  
        cgh.parallel_for(sycl::range<1>{size}, [=](sycl::id<1> pos) {  
            C[pos.get(0)] = A[pos.get(0)] + B[pos.get(0)];  
        });  
    });  
    deviceQueue.wait();  
}
```

Only this part is extracted
and compiled to SPIR-V

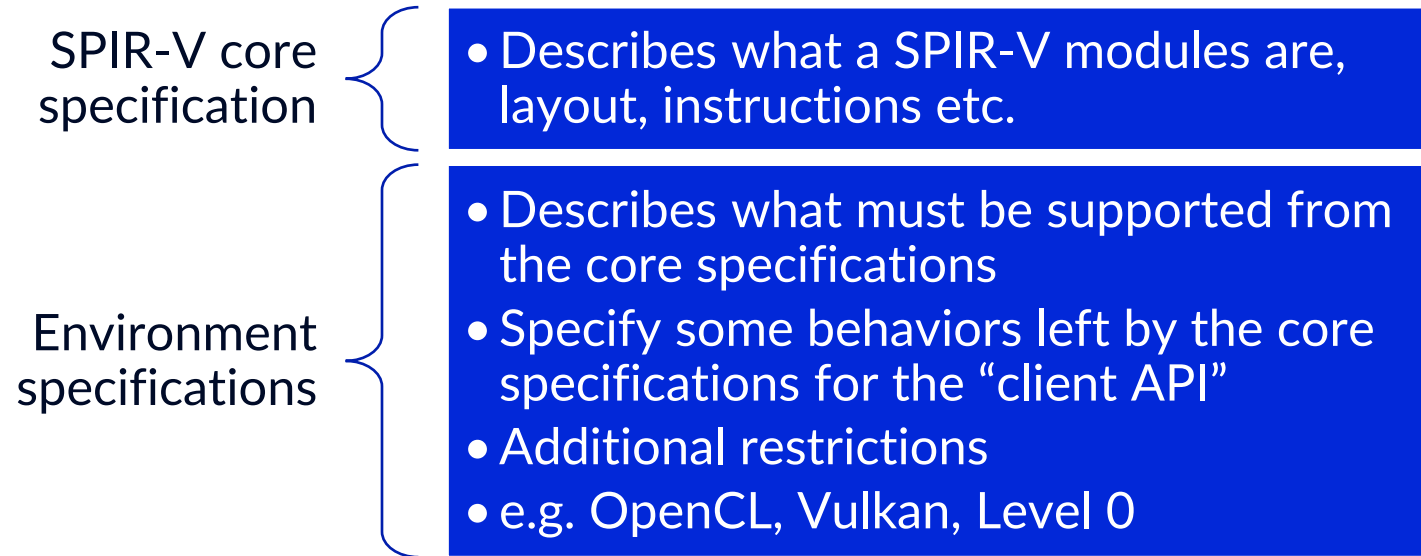


What is SPIR-V

SPIR-V Overview



- SPIR-V is a cross API intermediate representation for compute kernel / shaders
- Brings a stable exchange format to represent high level languages (OpenCL C, SYCL, GLSL, HLSL etc.) and execute them on a given driver stack





SPIR-V in a nutshell

- A SPIR-V module (file) is a collection of kernel / shaders
 - Stream of 32 bits words
 - Start with a small header and a list of instructions
- Each instruction starts with an opcode and a word count packed into a single word (16 bits each)
- Content is similar to LLVM in spirit
 - SSA form IR
 - Parametrized data types (int, float), aggregates and special types (like images)
 - Function, variables, entry point, FP control, builtins etc.
- Embeds an extension mechanism



SPIR-V extension mechanism

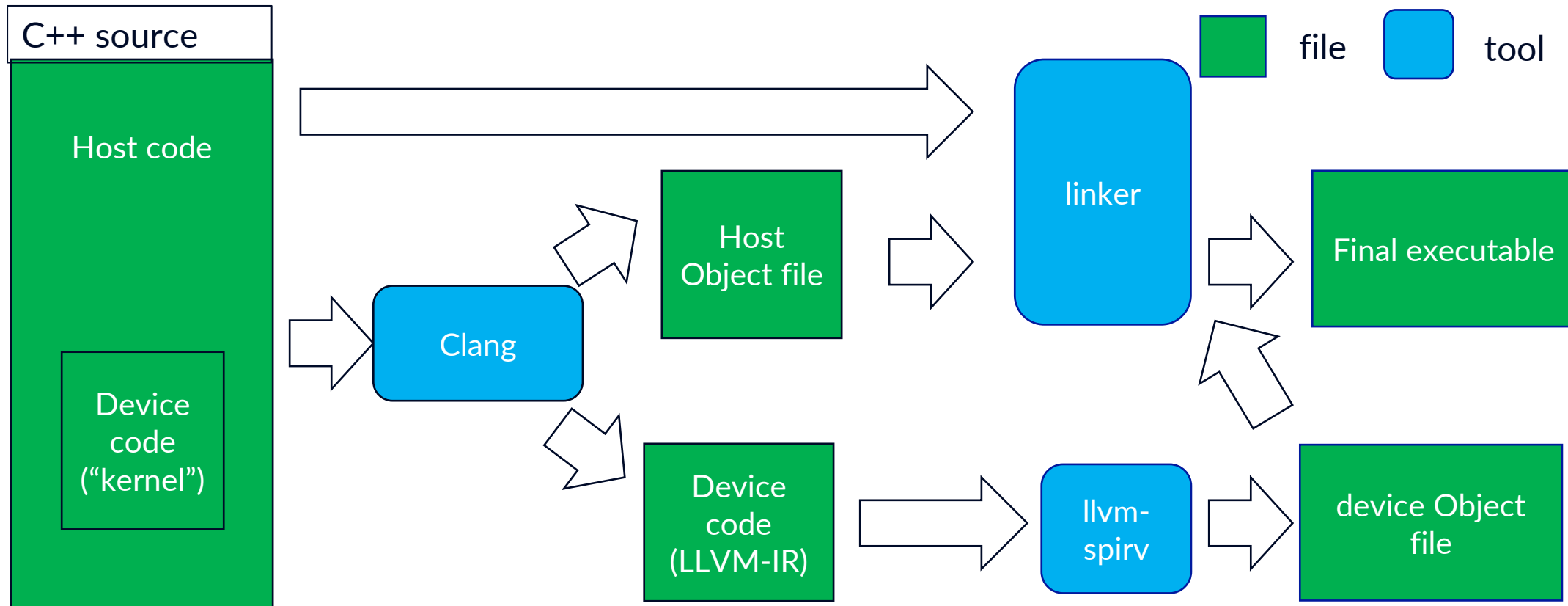
- SPIR-V has several ways to opt-in to different features
- Mechanism is designed to allow for early rejection of modules a driver cannot support
- OpCapability
 - Capability is an enum used by SPIR-V to guard the use of features in the module
 - At the start of a SPIR-V module, the application must declare all the capabilities used inside the module
 - OpenCL environment specifies a list of capabilities that must be supported by a conformant implementation, any extension used will be listed additionally here
- OpExtension
 - This make the module opt-in to an extension
 - Extension can modify and change many aspect of the core SPIR-V spec, but usually they add new capabilities and then new types and operations tied to these new capabilities
- OpExtInstImport
 - This allows to import a set of extra instructions (extended instruction set) behind a “namespace”
 - Set defined in a separate document (like an extension)
 - Operation in an extended instruction set are used with a special instruction: OpExtInst
 - Useful for non semantic instructions, which guaranties there is no semantic impact on the module and so can be safely removed / ignored



SPIR-V <-> LLVM Translation

- 2 tools can be used to translate LLVM to/from SPIR-V
 - SPIRV-LLVM-Translator (<https://github.com/KhronosGroup/SPIRV-LLVM-Translator>)
 - Currently the default tool for DPC++
 - Set of LLVM passes producing or consuming SPIR-V
 - LLVM SPIR-V backend
 - Soon no longer experimental
 - ARM and Intel are the main current contributors today
- Support up to SPIR-V 1.6 and many extensions
 - Both tools will try to produce a SPIR-V 1.0 module and gradually increase the required version based on used features up to a requested maximum
 - Similar for extensions, the tools starts with no extension declared in the module and adds allowed extensions on a per-need basis (e.g. a feature offered by the extension is required)
 - If a feature is both available in core SPIR-V and in some extension, the core version will be favored over the extension
 - This means the tools favors increasing the SPIR-V version rather than adding extension

Simplified SPIR-V flow in DPC++



SPIR-V example (trimmed)

```
; SPIR-V
; Version: 1.4
; Generator: Khronos LLVM/SPIR-V Translator; 14
; Bound: 28
; Schema: 0
OpCapability Addresses
OpCapability Linkage
OpCapability Kernel
OpCapability Int64
OpCapability ExpectAssumeKHR
OpExtension "SPV_KHR_expect_assume"
%1 = OpExtInstImport "OpenCL.std"
OpMemoryModel Physical64 OpenCL
OpEntryPoint Kernel %10

"_ZTSZZ6vecAddPKiS0_PimENKU1RN4sycl3_V17handlerEE_cl
ES5_EU1NS3_2idILI1EEEE_"
    %__spirv_BuiltInGlobalInvocationId
[...]
OpDecorate %add_i NoSignedWrap
[...]
```

```
%10 = OpFunction %void None %9
%_arg_C = OpFunctionParameter %_ptr_CrossWorkgroup_uint
%_arg_A = OpFunctionParameter %_ptr_CrossWorkgroup_uint
%_arg_B = OpFunctionParameter %_ptr_CrossWorkgroup_uint
%entry = OpLabel
%17 = OpInBoundsPtrAccessChain %_ptr_CrossWorkgroup_ulong
    %__spirv_BuiltInGlobalInvocationId %uint_0 %uint_0
%18 = OpLoad %ulong %17 Aligned 32
%cmp_i_i = OpULessThan %bool %18 %ulong_2147483648
OpAssumeTrueKHR %cmp_i_i
%arrayidx_i = OpInBoundsPtrAccessChain
    %_ptr_CrossWorkgroup_uint %_arg_A %18
%23 = OpLoad %uint %arrayidx_i Aligned 4
%arrayidx3_i = OpInBoundsPtrAccessChain
    %_ptr_CrossWorkgroup_uint %_arg_B %18
%25 = OpLoad %uint %arrayidx3_i Aligned 4
%add_i = OpIAdd %uint %23 %25
%arrayidx5_i = OpInBoundsPtrAccessChain
    %_ptr_CrossWorkgroup_uint %_arg_C %18
OpStore %arrayidx5_i %add_i Aligned 4
OpReturn
OpFunctionEnd
```

Running DPC++/SYCL with OpenCL

SPIR-V extensions in DPC++

- DPC++ SYCL implementation uses a variety of SPIR-V extensions to map SYCL features to hardware
 - Many of these are KHR or EXT SPIR-V extensions
 - Currently working on standardize those that are not
- Some SYCL extensions or compiler options enable Intel SPIR-V vendor extensions
 - Intel Compiler team is happy to work with other members of the community that may be interested on these
- All extensions mentioned in the presentation are usable by the SPIRV-LLVM translator by default
 - To stop their use, either
 - change associated flags (if applicable) or
 - Use `-Xspirv-translator -spirv-ext=-<ext name>`

Categories of SPIR-V extensions

SYCL core required extensions

- SPIR-V extensions required to implement SYCL 2020 core features
- They are not vendor specific except for one

DPC++ required extensions

- SPIR-V extensions required for any SYCL program
- They are not necessarily target vendor specific

Extensions required for performance

- Default or optional flags from DPC++ may enable these extensions
- They can be disabled manually but likely to hurt performances

Common features that require an extension

- Real SYCL applications use non-core features
- Some of these features require SPIR-V vendor extensions

OpenCL requirements

- SYCL being based on C++ has a “pay only for what you use” approach
 - While we list all capabilities needed to get conformance, only the capabilities used by a program will be used required in the module
- SYCL 2020 requires
 - OpenCL 2.0 or
 - OpenCL 3.0 with `__opencl_c_generic_address_space`, `__opencl_c_program_scope_global_variables` and `__opencl_c_int64` optional features
- OpenCL required capabilities are described in the OpenCL environment specs
 - https://registry.khronos.org/OpenCL/specs/3.0-unified/html/OpenCL_Env.html#required-capabilities

SPIR-V core requirements

- SPIR-V version
 - SPIR-V 1.0 which will be bump to
 - 1.1 if subgroup attributes are needed
 - 1.3 if using some group algorithm (any_of_group, all_of_group ...)
 - 1.4 if no sign/unsigned wrap is needed
- DPC++ defaults to 1.4 which avoids to use some extensions

Required extensions for SYCL/DPC++ (if feature used)

- SYCL `atomic_ref`
 - SPV_EXT_shader_atomic_float_add, SPV_EXT_shader_atomic_float_min_max and SPV_EXT_shader_atomic_float16_add
 - SYCL's `atomic_ref` defines arithmetic operations for 16, 32 and 64 bits float and int
 - SPIR-V 1.0 support these operations for int but 16, 32 and 64 float operations require the SPIR-V extensions
 - Unlike the name suggests, these are not bound to shaders
- Note: workaround is possible using compare exchange operation but not implemented in DPC++

Required extensions for SYCL/DPC++ (if feature used)

- SYCL group algorithms
 - SPV_KHR_uniform_group_instructions
 - Group algorithms for multiplication, bitwise and logical and/or/xor
 - SPV_INTEL_subgroups
 - For shuffle operations
- Handling of SYCL_EXTERNAL
 - SPV_KHR_linkonce_odr
 - SYCL Macro defines the ability to define functions over several translation units, but SPIR-V lacks the C++ One Definition Rule understanding / linkage capability

De-facto required extensions for DPC++

- The following is a series of enabled extensions that are not strictly necessary but likely to be used
 - However, they may be necessary for good performance !

De-facto required extensions for DPC++

- `SPV_KHR_expect_assume`
 - Used in some places to give hints to the compiler
 - E.g. `sycl::id` fits in a 32 bits integer
- `SPV_KHR_non_semantic_info`
 - Used to encode debug info and LLVM metadata not representable in SPIR-V (opt-in)
- `SPV_INTEL_long_constant_composite`
 - Allow for the initialization of arrays and structs with more than 65535 element

Optimization triggered extensions

- SPV_KHR_no_integer_wrap_decoration
 - Allows the compiler to assume no integer overflow
 - Note: the SPIRV-LLVM-Translator will favor emitting a SPIR-V 1.4 module rather than emitting the extension
- SPV_INTEL_unstructured_loop_controls
 - Allow to apply loop controls on non “structured” loops
 - In practice, this mainly means supporting do / while loops and rotated loops (for / while loops turned into do while loops)

Optimization triggered extensions

- C++ Compiler flag controlled
 - `SPV_INTEL_fp_fast_math_mode`
 - Some of the fast math flags will be added by `-mreassociate` or `-ffp-contract=fast`
 - `SPV_INTEL_optnone` (now published as `SPV_EXT_optnone`)
 - Will be used with `O0`
 - can be prevented `-disable-O0-optnone`

SYCL extensions that require SPIR-V extensions

- Joint matrix
 - SYCL extension `sycl_ext_oneapi_matrix`
 - SPIR-V extensions
 - `SPV_INTEL_joint_matrix`
 - `SPV_KHR_cooperative_matrix`
- Float controls
 - SYCL extension `sycl_ext_intel_fp_control`
 - SPIR-V extensions
 - `SPV_KHR_float_controls` (or SPIR-V 1.4)
 - `SPV_INTEL_float_controls2`
- Cache control
 - SYCL extension `sycl_ext_intel_cache_controls`
 - SPIR-V extension `SPV_INTEL_cache_controls`
- SYCL Bindless Textures
 - SYCL extension `sycl_ext_oneapi_bindless_images`
 - SPIR-V extension `SPV_INTEL_bindless_images`
- Kernel properties
 - SYCL extension `sycl_ext_oneapi_kernel_properties`
 - SPIR-V extension `SPV_INTEL_kernel_attributes`
- Virtual Functions
 - SYCL extension `sycl_ext_oneapi_virtual_functions`
 - SPIR-V extension `SPV_INTEL_function_pointers`
- BFloat16
 - SYCL extension `sycl_ext_oneapi_bfloat16`
 - SPIR-V extension `SPV_INTEL_bfloat16_conversion`

OCK supported extensions

- oneAPI Construction Kit has support for the following
 - SPV_KHR_no_integer_wrap_decoration
 - SPV_INTEL_kernel_attributes
 - SPV_EXT_shader_atomic_float_add
 - SPV_EXT_shader_atomic_float_min_max
 - SPV_KHR_expect_assume
 - SPV_KHR_linkonce_odr
 - SPV_KHR_uniform_group_instructions
 - SPV_INTEL_arbitrary_precision_integers
 - SPV_INTEL_optnone
 - SPV_INTEL_memory_access_aliasing
 - SPV_INTEL_subgroups
- Cover the core SYCL functionalities and a bit more
- Some extensions are missing, this is often due to the fact they are still experimental

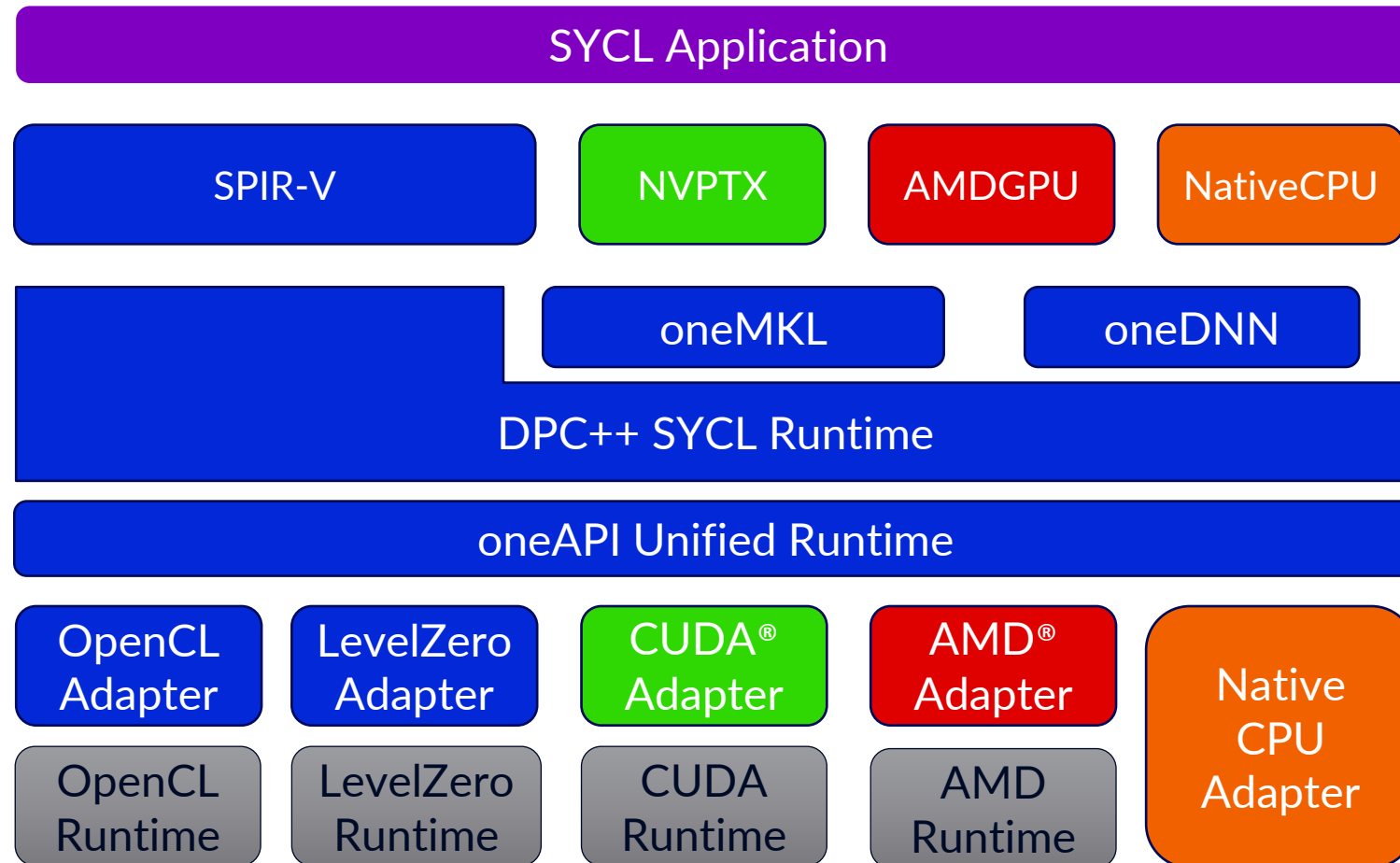
Call for action

- Review the spreadsheet and provide feedback on the extensions
 - https://docs.google.com/spreadsheets/d/1pgPno--m-MiQljN1kKkgzi1KEr4o_0cF/edit?usp=drive_link&ouid=112032962187651132508&rt=pof=true&sd=true
- Are any extensions not supported in your platform?
- Are there extensions you may need for platform?
- Are there any Intel vendor extensions you would like to see as EXT or KHR for your platform ?
- Join the Slack for additional discussions

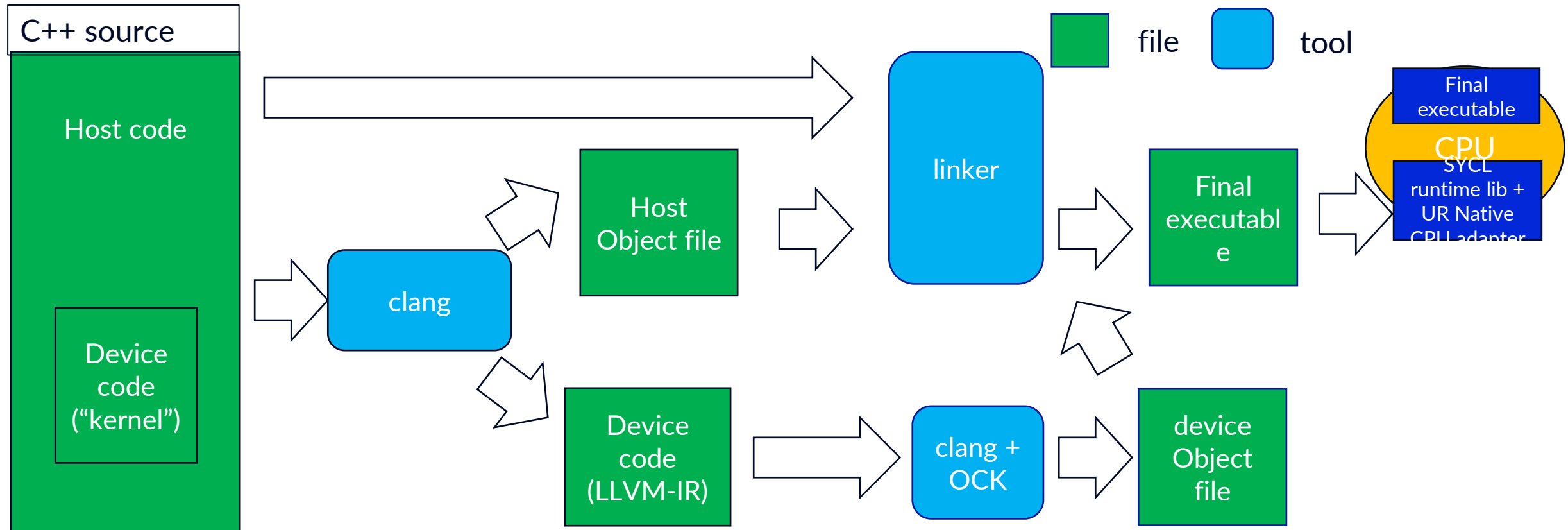
Native CPU

Alternative path to SPIR-V

oneAPI Software Stack for SYCL



Native CPU flow



Native CPU adaptive Parameters

- Pluggable Runtime Scheduler
 - TBB based
 - Default
 - Custom scheduler (OpenMP based?)
- Pluggable Vectorisation
 - Whole function vectorisation based on OCK
 - Intel vectorizer
 - LLVM loop vectorizer
- More information about NativeCPU can be found here:
 - <https://github.com/intel/llvm/blob/sycl/sycl/doc/design/SYCLNativeCPU.md>