LLVM Developer Meeting ’24

Enhance SYCL offloading support to use the new offloading model

Speaker: Ravi Narayanaswamy

Contributors: Arvind Sudarsanam, Maksim Sabianin, Nick Sarnie, Alexey Sachkov, Mike Toguchi

intel.

Agenda

- Overview of SYCL offloading
  - Highlight some of its important features
- Proposed design to use the new offloading model
- Deviations from existing community flow
  - Motivation and proposed changes
- Work done so far
- What next?

SYCL Offloading

- SYCL is designed for data parallel programming and heterogenous computing, and provides a consistent programming language (C++) and APIs across CPU, GPU, FPGA, and AI accelerators.
- Compiler enables multiple toolchains (one for host and one each for the targets provided)

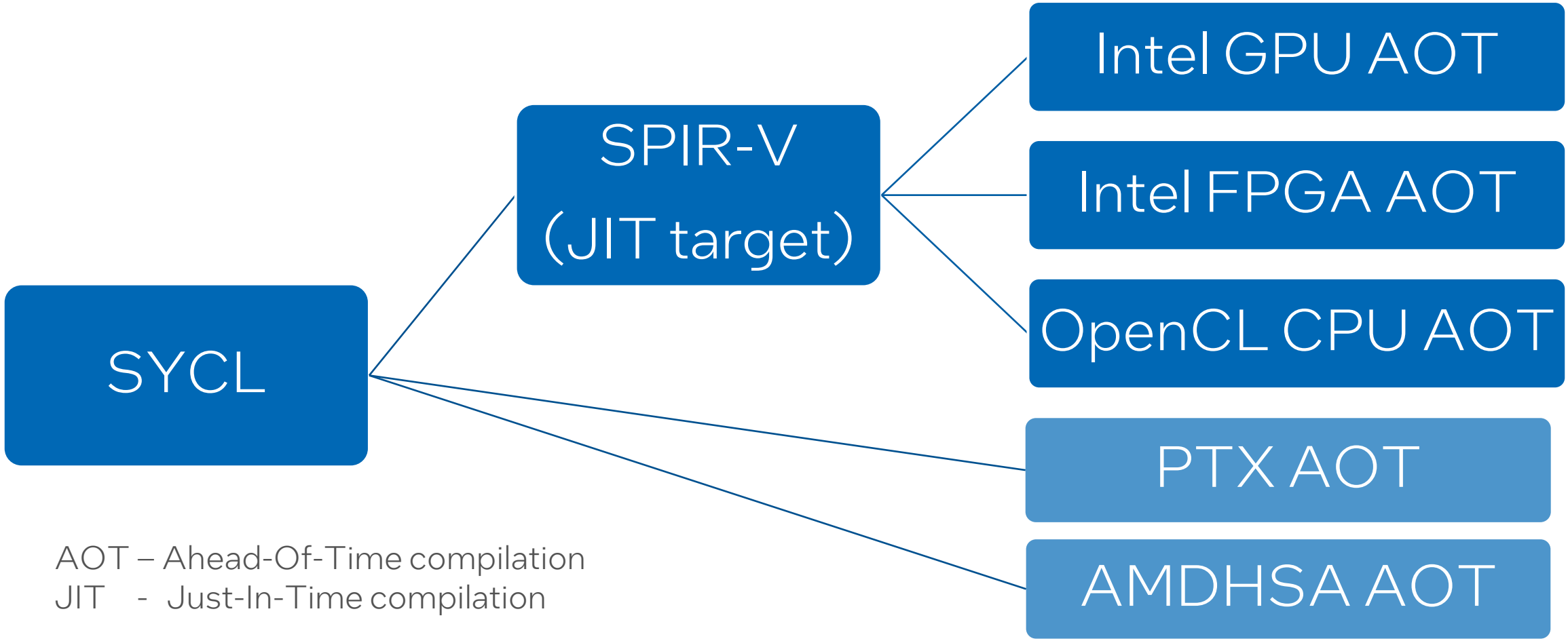
```
clang++ -fsycl –fsycl-targets=intel-gpu-YYY, intel-gpu-XXX test.cpp
```

```
sycl::ext::oneapi::experimental::device_global<int> dev_var;

void func(sycl::queue q) {
 int val = 42;
 q.copy(&val, dev_var).wait();
 // The 'dev_var' parameter is by reference
 q.submit([&](sycl::handler &cgh) {
  cgh.single_task([=] {
   for (int i = 0; i < 10; ++i)
     dev_var += i;
  });
 });
}
```

SYCL kernel

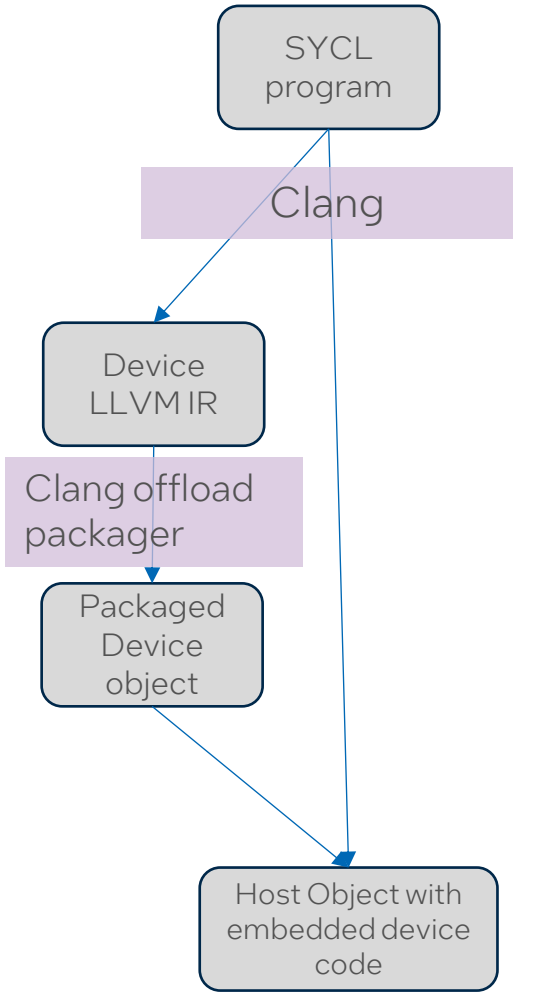
SYCL offloading and supported AOT/JIT targets



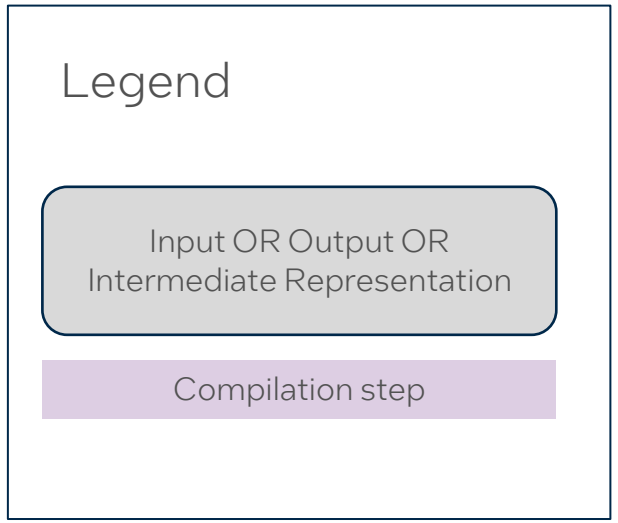
AOT – Ahead-Of-Time compilation
JIT - Just-In-Time compilation

Overview of SYCL compiler - Using new offloading model

COMPILATION STAGE

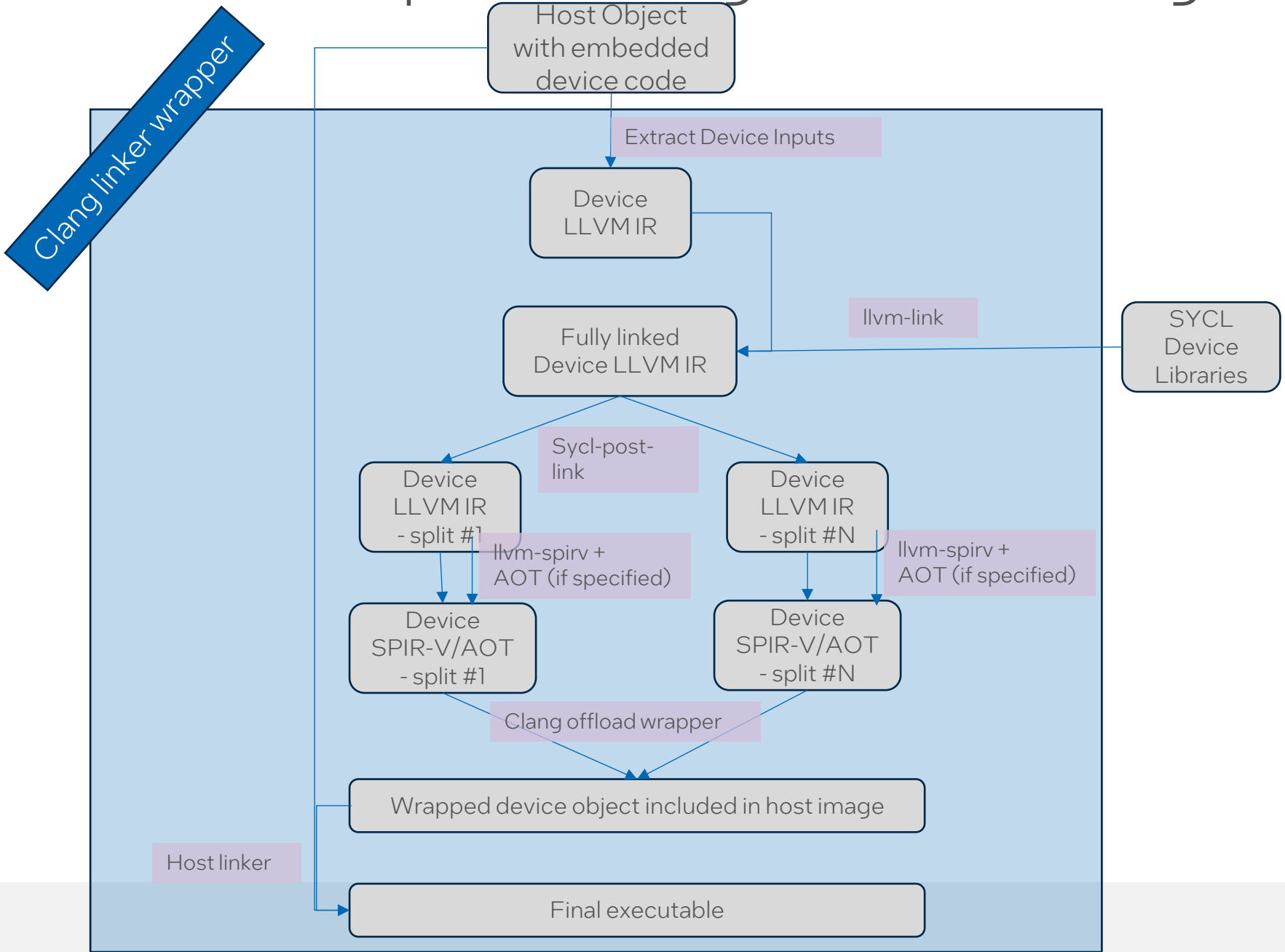


Uses -fembed-offload-object



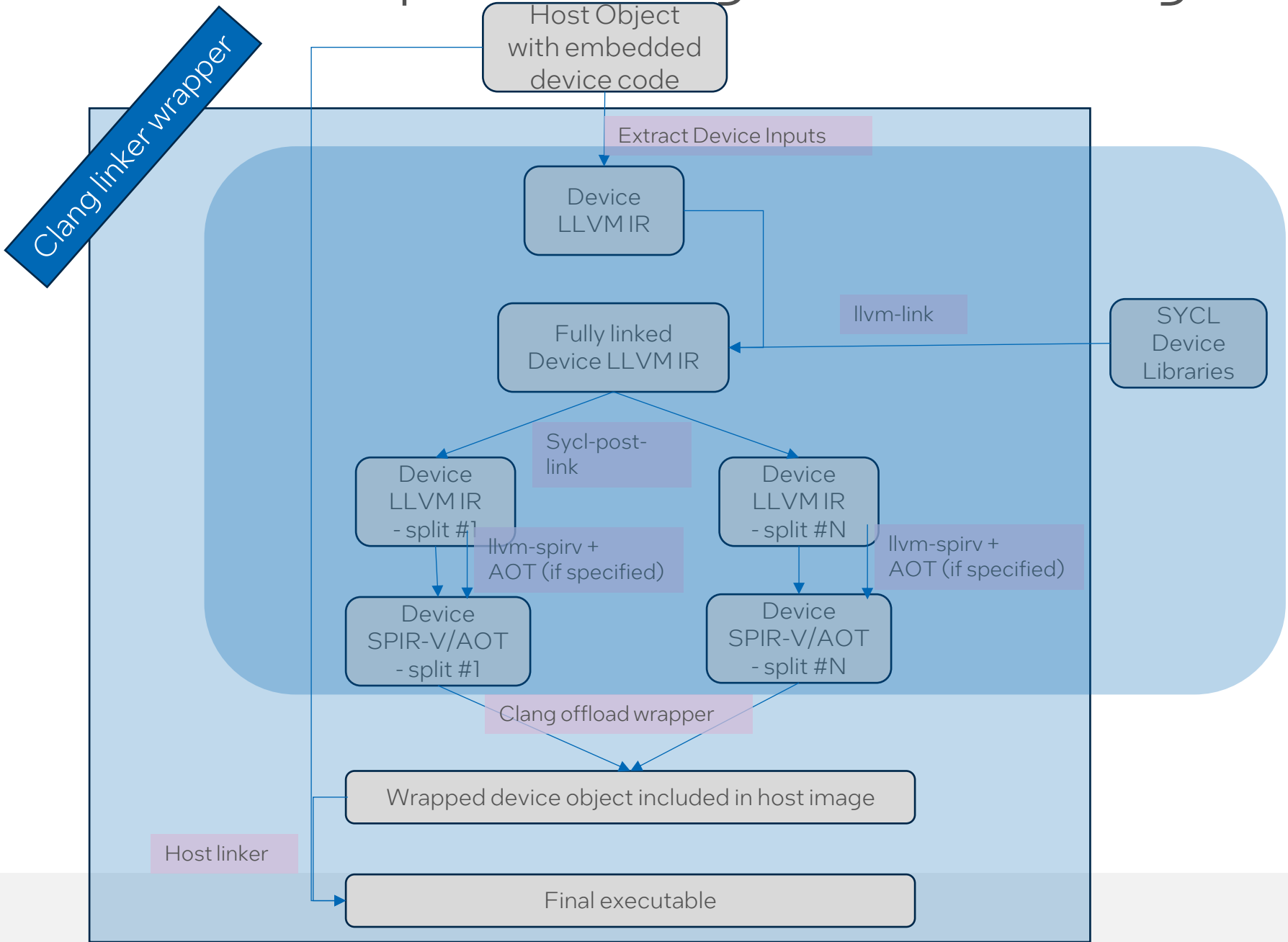
Overview of SYCL compiler - Using new offloading model

LINKING STAGE



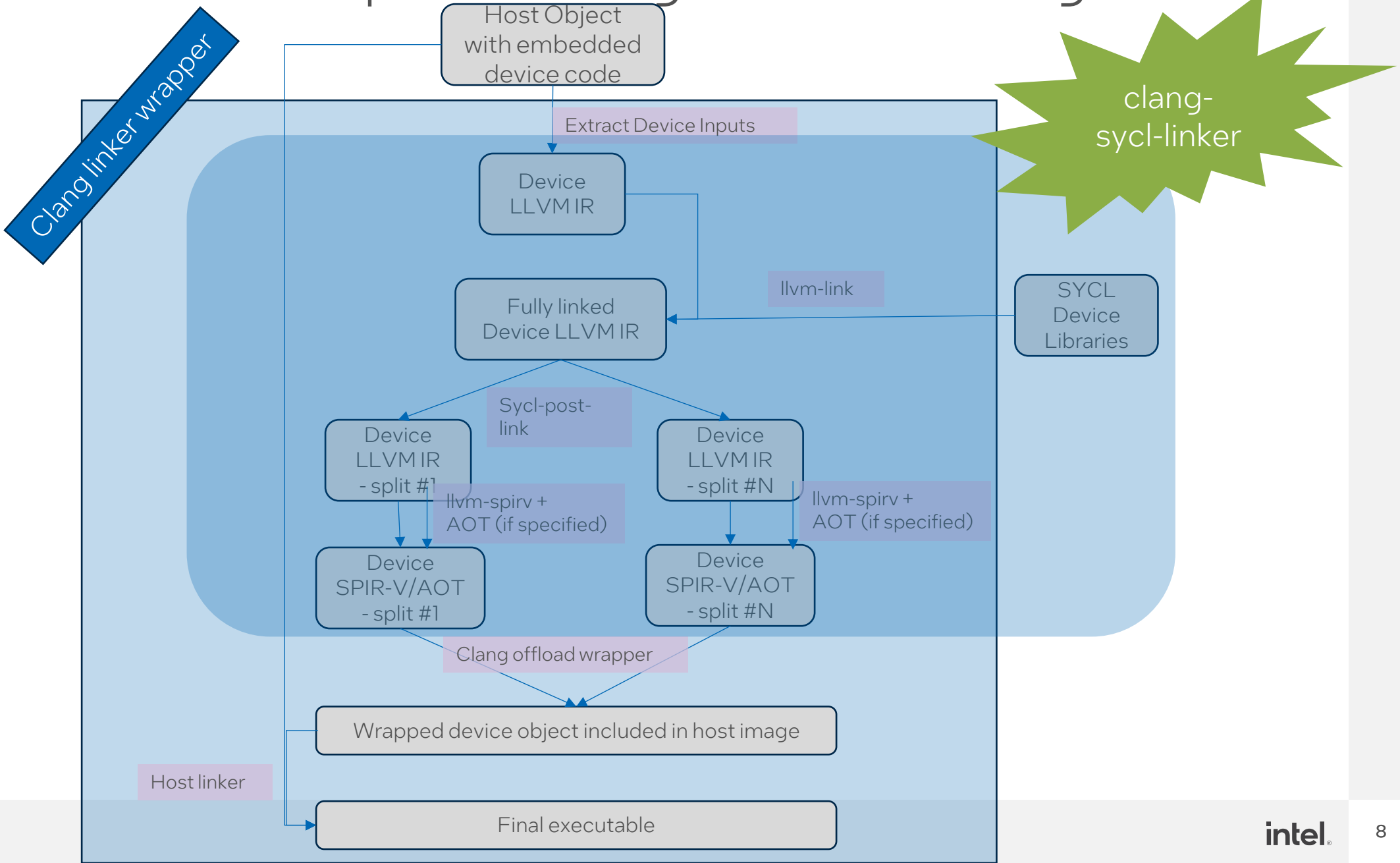
Overview of SYCL compiler - Using new offloading model

LINKING STAGE

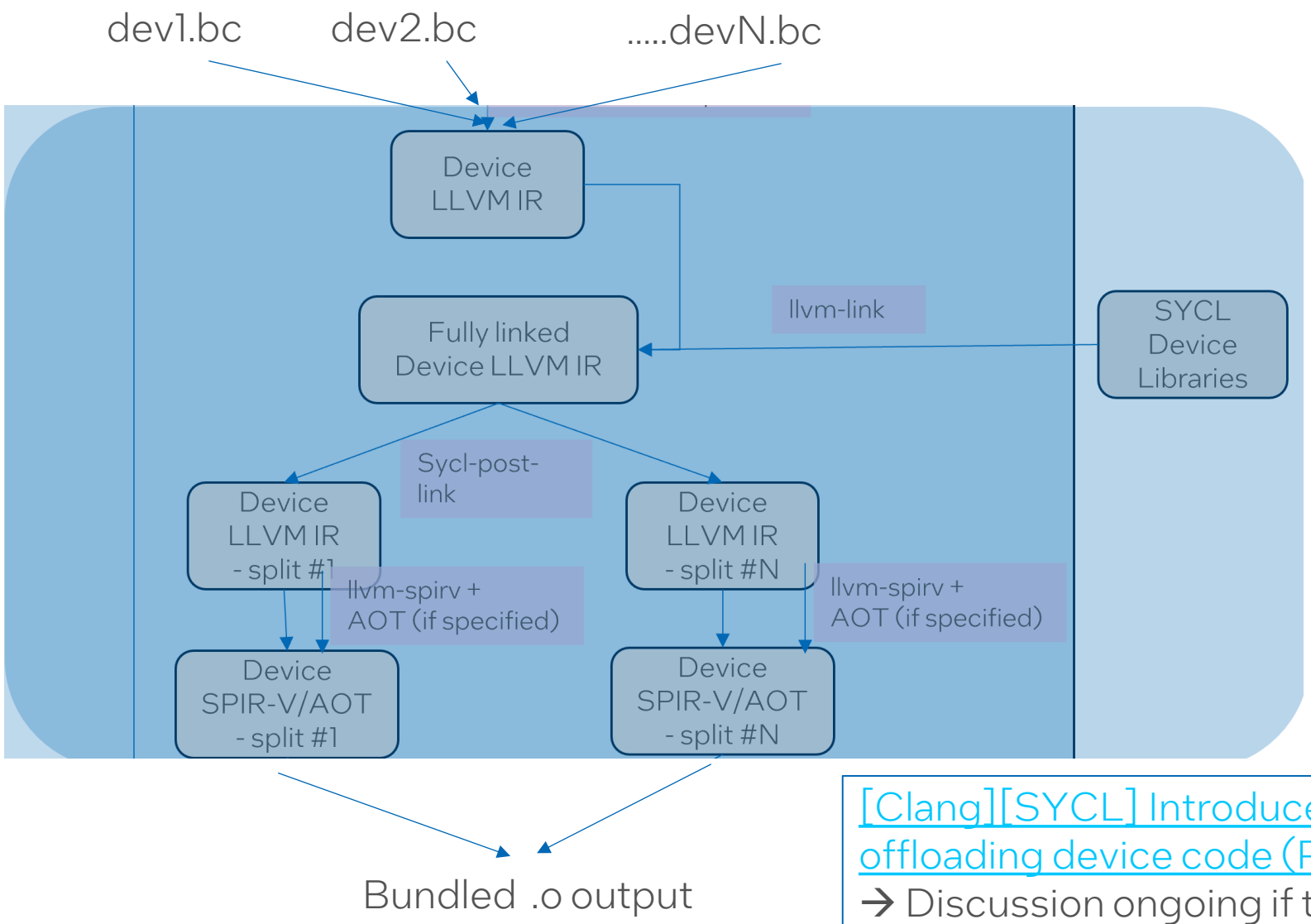


Overview of SYCL compiler - Using new offloading model

LINKING STAGE



Overview of SYCL compiler - Using new offloading model

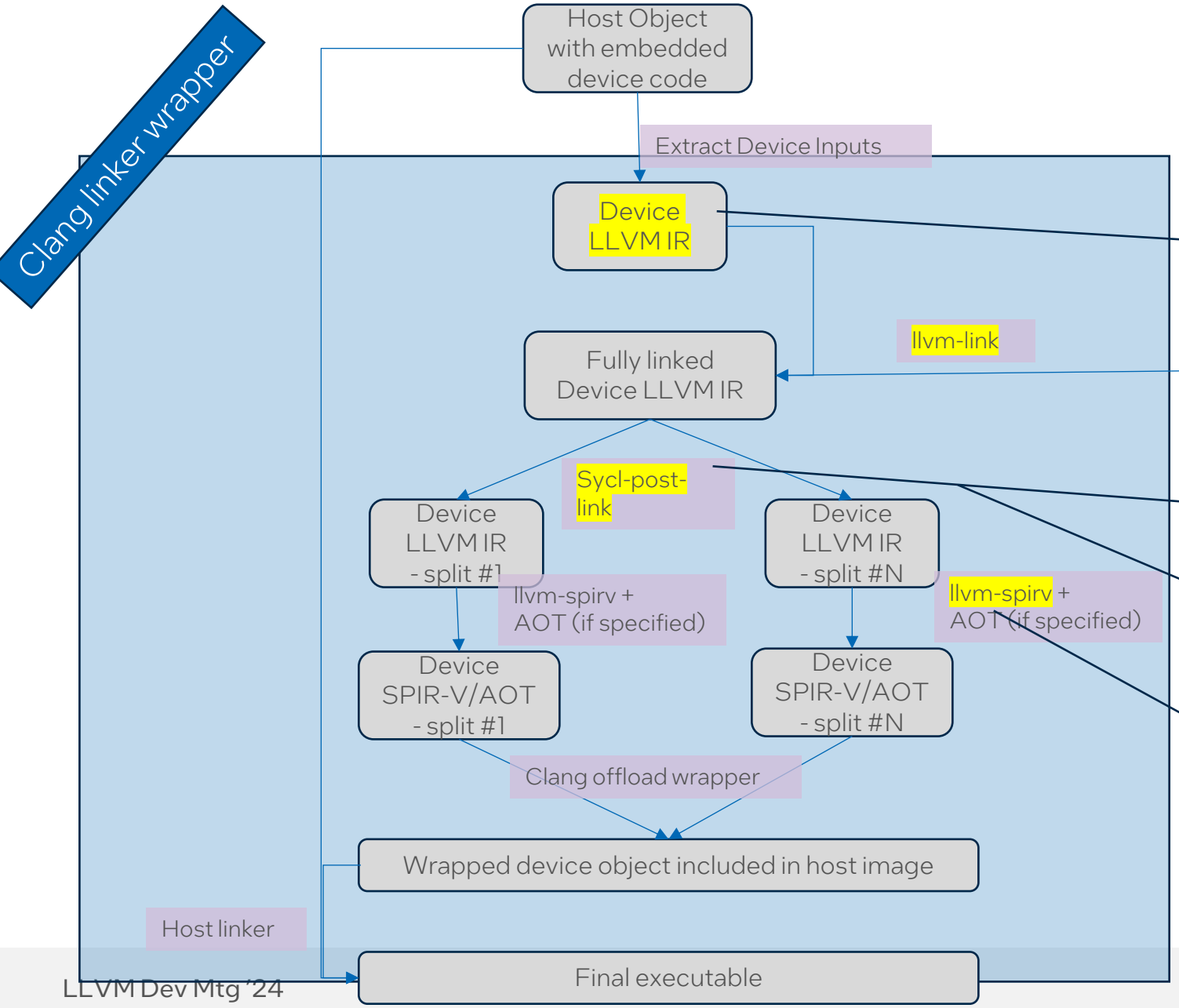



```
clang++ dev1.bc dev2.bc
–target=spirv
--sycl-link
-Xlinker <SYCL Link Options>
-o out.o
```

This invokes the linking job of SPIR-V device toolchain. '—sycl-link' directs the job to invoke a new tool called '**clang-sycl-linker**' which performs device code linking.

[Clang][SYCL] Introduce clang-sycl-linker to link SYCL offloading device code (Part 1 of many)
→ Discussion ongoing if this should be a llvm tool instead.

Deviations from existing OpenMP offloading flow



Use of llvm-link instead of LTO for bitcode linking

Linking device libraries during link-time

Device image requires extra information to be passed to runtime

Device code gets split into multiple chunks

External tool used for backend code gen

Variation #1: Device code linking

Device code linking is performed at LLVM IR level as there is no 'mature' SPIR-V IR linker available.

Current status: llvm-link is used for device code linking.

Final goal: Once SPIR-V backend is available, linking can be performed using LTO (full or thin).

Variation #2: Linking of SYCL device libraries

Several SYCL device compilation use cases require SYCL device libraries to be linked into the device IR.

Current status: llvm-link (with –only-needed option) is used for linking device libraries.

Final goal: To be incorporated into the LTO pipeline once SPIR-V backend is available.

Variation #3: Transmission of user specified data from SYCL compilation phase to SYCL runtime

```
struct OffloadingImage {
    // LLVM BC, PTX, Object, SPIR-V etc.
    ImageKind TheImageKind;
    // OpenMP, CUDA, SYCL, etc.
    OffloadKind TheOffloadKind;
    uint32_t Flags;
    // Used to store metadata supposedly required
    by runtime (triple, arch)
    MapVector<StringRef, StringRef> StringData;
    // actual target code
    std::unique_ptr<MemoryBuffer> Image;
}
```

For some use cases, it is expected that some of the compilation flags provided by the user need to be propagated to the SYCL runtime

For instance, if a program is compiled using –O0, the flag should be propagated to the SYCL runtime.

```
StringData["compiler options"] = "-O0";
StringData["linker options"] = "-O0";
```

Goal: Use StringData map to store this information.

Variation #3: Transmission of SYCL specific data from SYCL compilation phase to SYCL runtime

Device image properties
Each device image is accompanied by a 'property set' listing device requirements

Example: Optional kernel features (aspect::fp16; aspect::fp64)

```
queue q;
q.single_task([=]() {
    // kernel uses aspect::fp64
    double pi = 3.14;
});
// single_task is expected to throw feature_not_supported exception
during runtime if it is run on a device that does not support fp64.
```

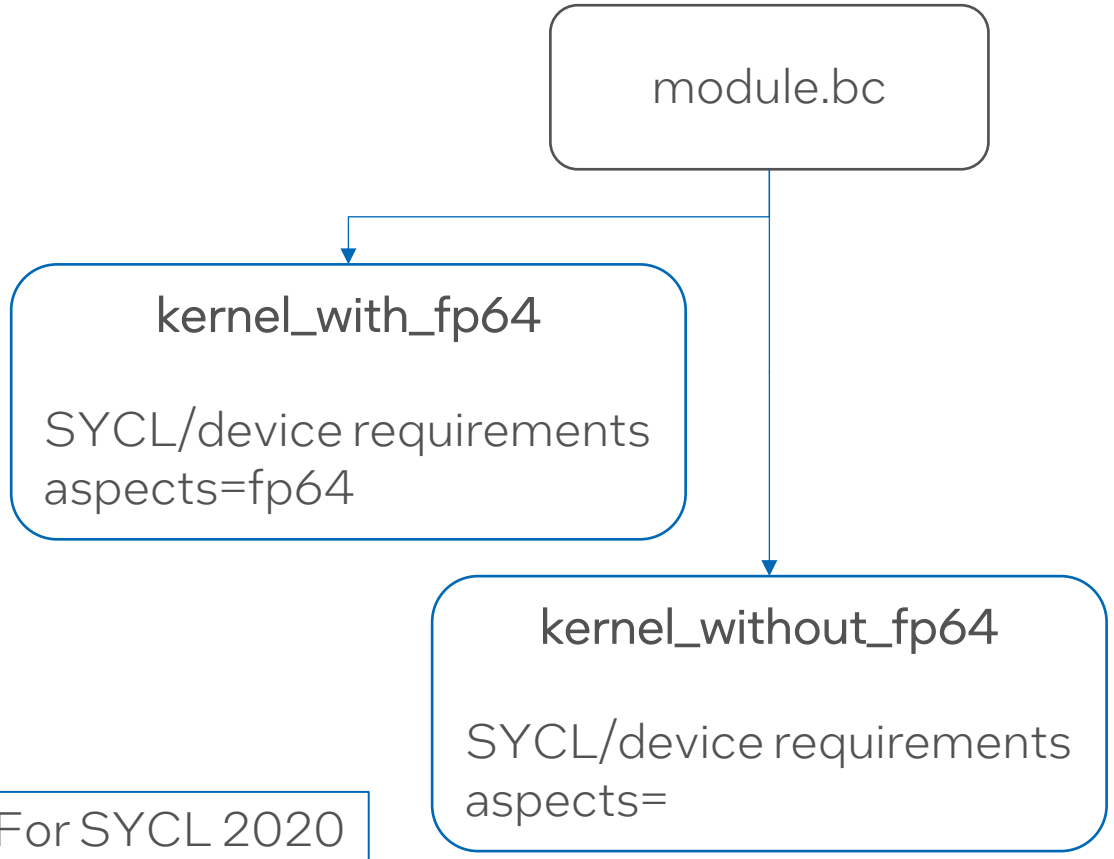
```
struct __device_image_property {
    char *Name;
    void *Value;
    // Type is uint32 or byte array
    uint32_t Type; uint64_t ValueSize;
}
```

Goal: Use StringData map to store this information.

Variation #4: Device code splitting

```
queue q;

if (q.get_device().has(aspect::fp64))
  q.single_task<kernel_with_fp64>([=]() {
    // kernel uses fp64
    double pi = 3.14;
  });
else
  q.single_task<kernel_without_fp64>([=]()
    // kernel *does not* use fp64
    float pi = 3.14f;
  });
```



Motivation #1: Per used optional feature; For SYCL 2020 conformance
Motivation #2: Per kernel; To reduce JIT overhead

Variation #5: LLVM to SPIR-V IR translation

Backend compilation flow for Intel targets require the code to be available in SPIR-V format.

Current status: llvm-spirv (an external tool) is used for LLVM to SPIR-V translation

Final goal: Once SPIR-V backend is made available for use in compilation flows, translation can be performed using the backend passes.

Work done so far

- Top level RFC submitted during end of 2023
  - [RFC] Add Full Support for the SYCL Programming Model - Link
  - [RFC] Offloading design for SYCL offload kind and SPIR targets - Link
- Initial analysis presented during the EuroLLVM 2024 conference (Thanks Alexey Sachkov)
  - https://www.youtube.com/watch?v=uhNHlytKX4c
- Initial sets of changes are currently being made upstream (Two PRs under review)
  - [SYCL][LLVM] Adding property set I/O library for SYCL
  - [Clang][SYCL] Introduce clang-sycl-linker to link SYCL offloading device code (Part 1 of many)
  - Special thanks to Joseph Huber for kind guidance on enhancing SYCL offloading flow to use the new offload model
  - Special thanks to Matt Arsenault, Chris B, and Tom Honermann for great feedback thus far.

Next Steps

- More PRs to complete the support to add SYCL offloading flow to the new offloading model.
  1. SYCL finalization steps that will run after llvm-link will be added to the linking flow inside clang-sycl-linker. One of the finalization steps is device code splitting.
  2. Changes to clang-linker-wrapper to invoke the clang-sycl-linker (via the clang driver call) for SYCL offloading case.
  3. Add SYCL offload wrapping logic to clang-linker-wrapper
  4. AOT compilation support for Intel, AMD and NVidia GPUs

- Once SPIR-V backend is available, update the implementation to use LTO.

intel®