The LLVM Offloading Infrastructure

Joseph Huber *(joseph.huber@amd.com)*

AMD

together we advance_

Introduction — Offloading

- LLVM Utilities to run programs on external accelerators (GPUs)
- Generic and re-usable between languages and vendors
- Offloading infrastructure has many parts
  - Clang Driver
  - Language frontends
  - Language headers
  - Target backends
  - Binary utilities / formats
  - Device runtimes
  - Offloading runtimes
  - MLIR dialects

Introduction — GPU Compute

- You can run DOOM on it

Offloading Compilation

Compilation — Direct Targets

- Accelerators are just **cross-compiling** targets
  - Supported with the **--target=** option
- Clang driver decides the steps necessary to get a **usable executable**
- Everything should work like the targets you know and love
  - LLVM backends and tools
- *If you omit the CPU and emit LTO IR you can get quasi-generic libraries like SPIR-V*

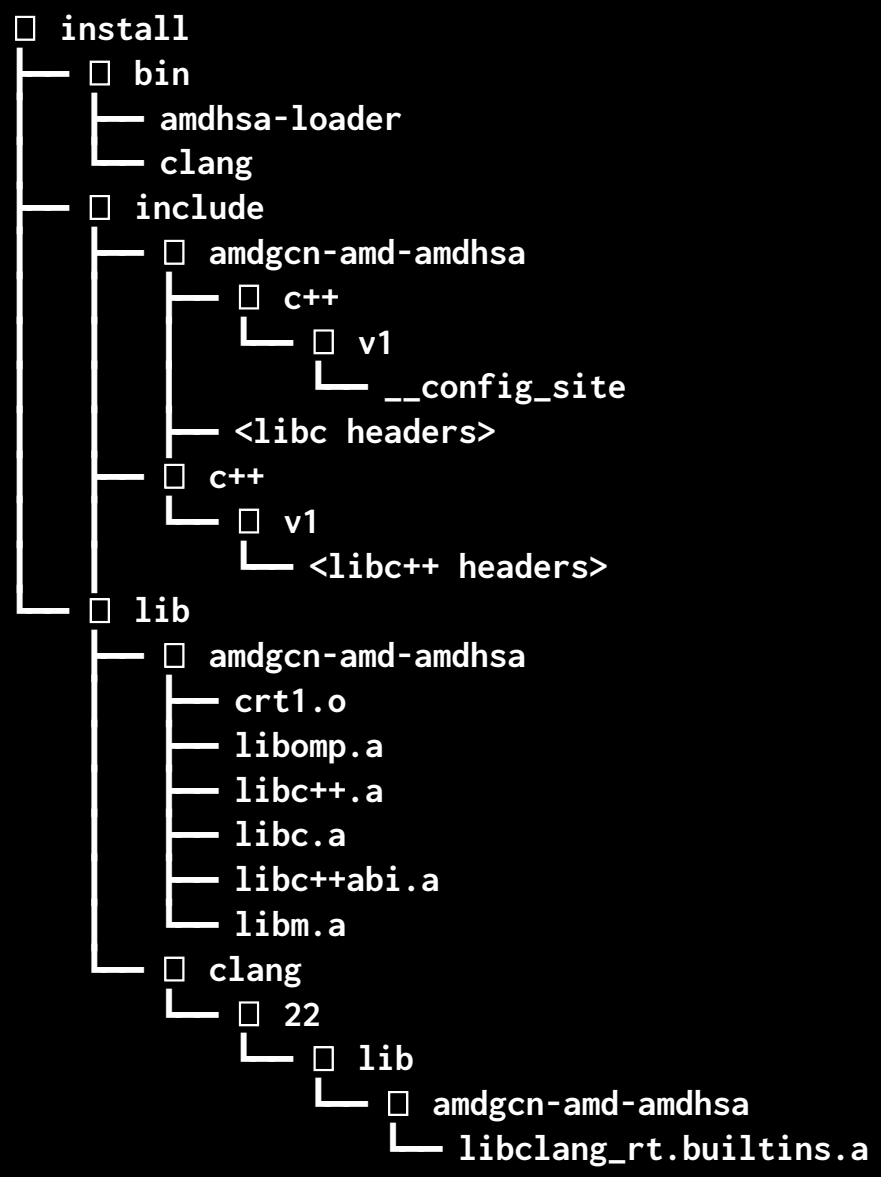
```
#include <gpuintrin.h>
__gpu_kernel void saxpy(int n, float a, float *x, float *y) {
    int i = __gpu_block_id(0) * __gpu_num_blocks(0) + __gpu_thread_id(0);
    if (i < n) y[i] = a * x[i] + y[i];
}
```

```
$ clang kernel.c --target=amdgcn-amd-amdhsa -mcpu=gfx942 -flto
$ clang kernel.c --target=spirv64-- -flto // Work in progress
$ clang kernel.c --target=nvptx64-nvidia-cuda -march=sm_90 -flto
```

Compilation — Runtime Libraries

- Each target gets its own directory
  - `-DLLVM_ENABLE_PER_TARGET_RUNTIME_DIR=ON`
  - `-DLLVM_RUNTIMES_TARGETS=<triples>`
- Included automatically when linking for the language
- Supported GPU runtimes (See my other talks)
  - compiler-rt
  - openmp
  - libc
  - libc++
  - libc++abi
  - flang-rt

```
$ cmake ../llvm -G Ninja                          \
  -C ../offload/cmake/caches/Offload.cmake \
  -DCMAKE_BUILD_TYPE=<Debug|Release>       \
  -DCMAKE_INSTALL_PREFIX=<PATH>            \
$ ninja install
```



Compilation — Offloading Languages

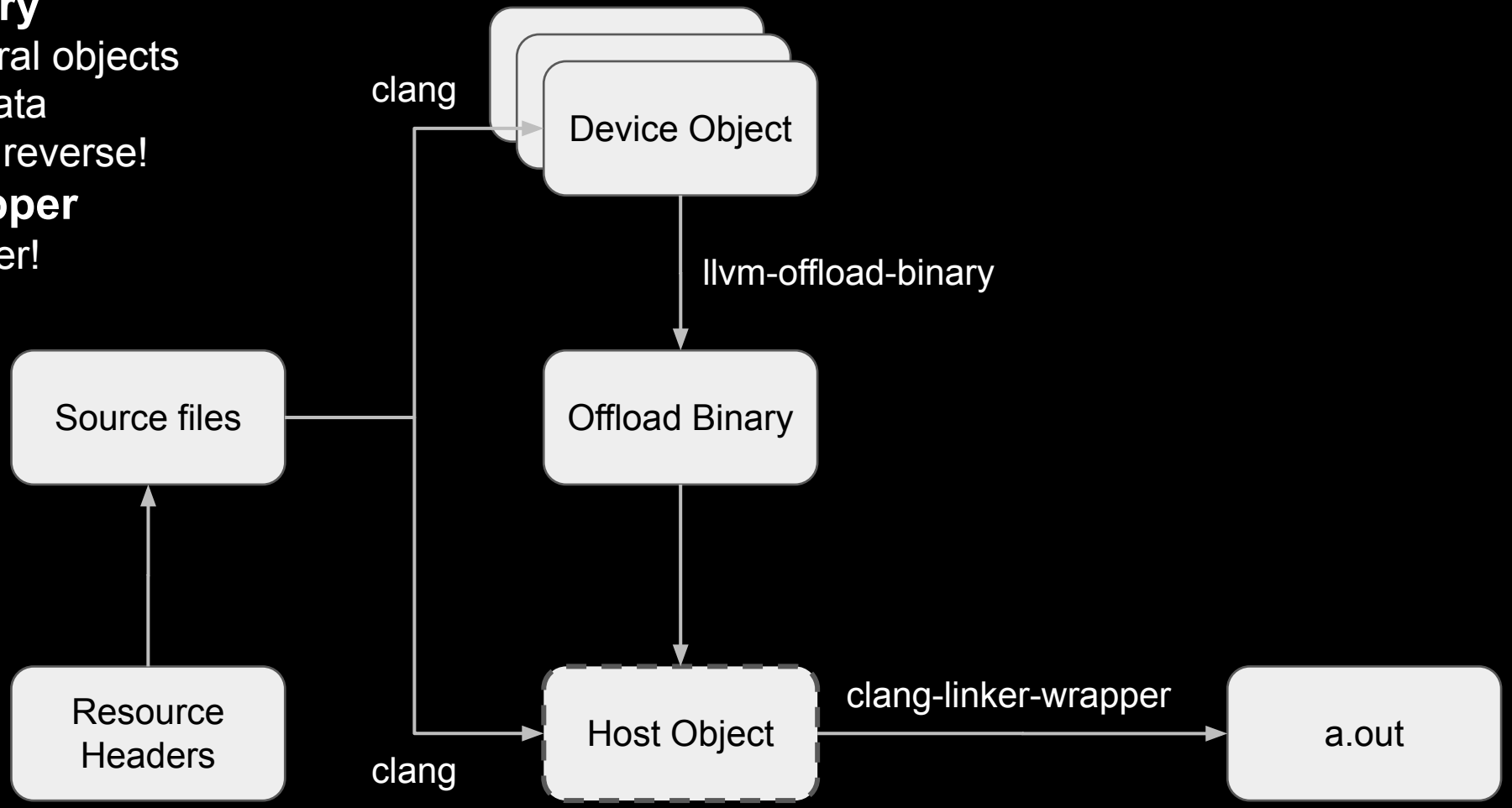
- Ubiquitous for targeting accelerators
  - OpenMP, HIP, SYCL*, CUDA
- Compiler stages managed by the Clang driver toolchain
- Combines **host** and many **device** compilation phases
  - Uses the same core functionality as the standalone case
  - Just some special headers and keywords
- Uses lots of offloading tools under the hood


```
__global__ void saxpy(int n, float a, float *x, float *y) {
    int i = blockIdx.x * blockDim.x + threadIdx.x;
    if (i < n)
        y[i] = a * x[i] + y[i];
}
```

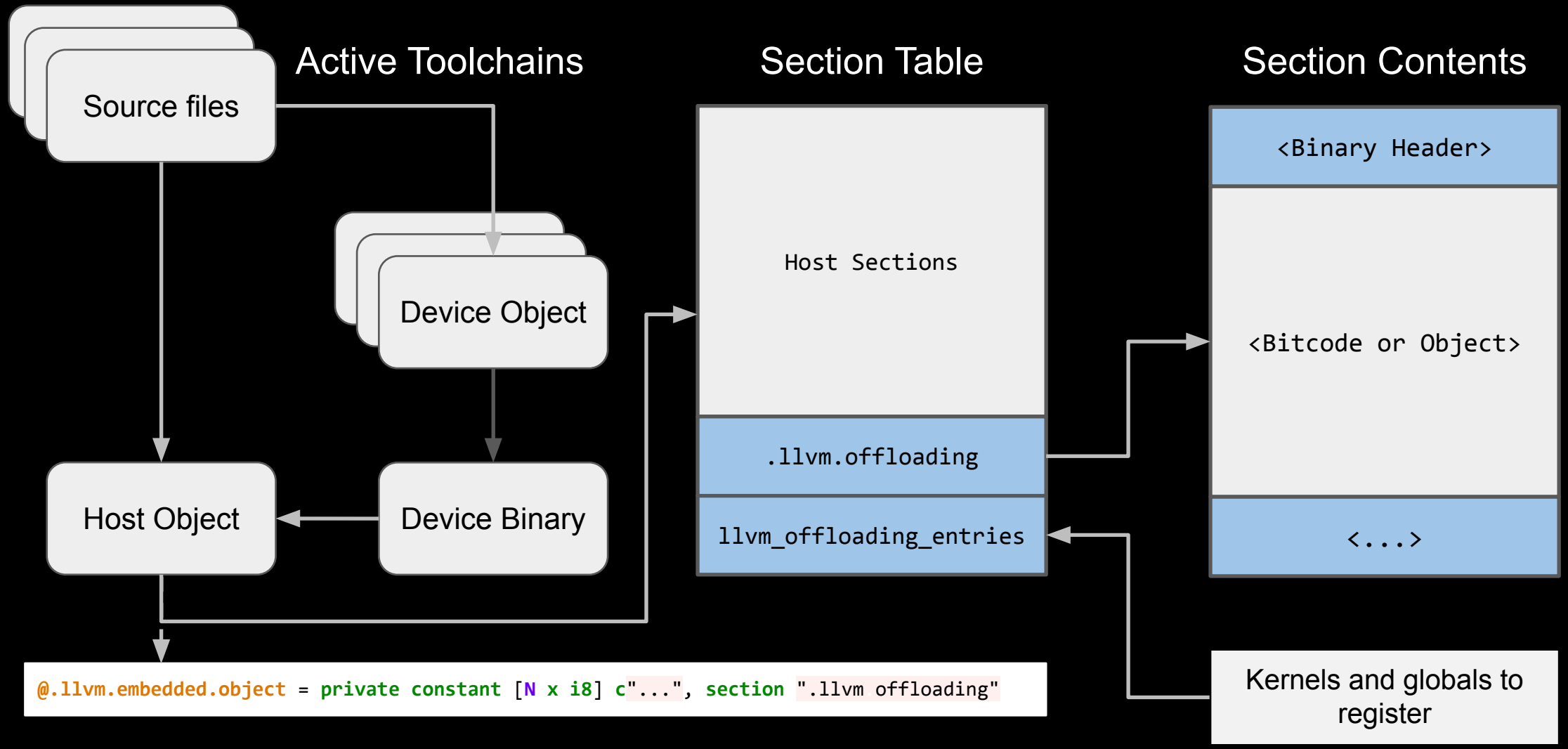
```
$ clang++ -x hip hip.cpp --offload-arch=gfx942 -v
```

Offloading — Compilation Bindings

- **llvm-offload-binary**
  - Combines several objects
  - Includes metadata
  - Can be done in reverse!
- **clang-linker-wrapper**
  - More on this later!



Offloading — Binary Usage



Offloading — Linking

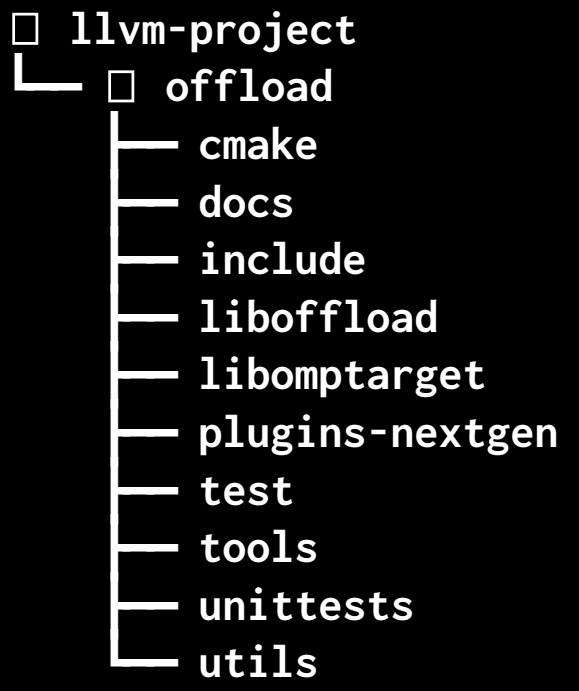
- **clang-linker-wrapper**
  - Performs device linking and runtime call registration in one action
- **llvm-offload-binary**
  - Extracts the files
- **llvm-offload-wrapper**
  - LLVM-IR module that initializes the runtime with the executable
- **llvm-objdump**
  - With **--offloading** will print the contained information

```
$ clang-linker-wrapper host.o --linker-path=/usr/bin/ld.lld
// OR
$ llvm-offload-binary host.o --image=file=dev.o,arch=gfx942
$ clang --target=amdgcn-amd-amdhsa -mcpu=gfx942 dev.o -o image -l<libraries>
$ llvm-offload-wrapper --triple=x86_64-unknown-linux -kind=hip image -o out.bc
$ clang --target=x86_64-unknown-linux out.bc -o reg.o
$ ld.lld host.o reg.o -o a.out
```

Offloading Library / API

Runtime — Project Overview

- The **llvm-project/offload** runtime!
- Generic* interface over vendor GPU runtimes
  - Extensions to keep us from a minimally useful subset
  - Supports AMD, NVIDIA, and Intel* GPUs
    - *Wraps over dynamically opened vendor libraries**
- Originally came from the OpenMP offloading support
  - The libomptarget library
- Made **liboffload** to export it!



OpenMP

Runtime — Liboffload

- Provide a **stable** C API for offloading GPU programs
  - *Aren't there enough of these?*
  - We want ABIs we can control inside LLVM
- **Work-in-progress** still version 0.0 and may change!
- Everything starts with **ol** for **offload library**
  - I vetoed **LLVM offload library**

Runtime — Liboffload Design

- Public interface defined through **tablegen**
  - Automatic documentation
  - Verification and tracing
- Headers installed into **include/offload/OffloadAPI.h***

```
def olInit : Function {
  let desc = "Perform initialization of the Offload library";
  let details = [
    "This must be the first API call made by a user of the Offload library",
    "The underlying platforms are lazily initialized on their first use"
    "Each call will increment an internal reference count that is decremented by `olShutDown`"
  ];
  let params = [];
  let returns = [];
}
```

Runtime — Liboffload Design

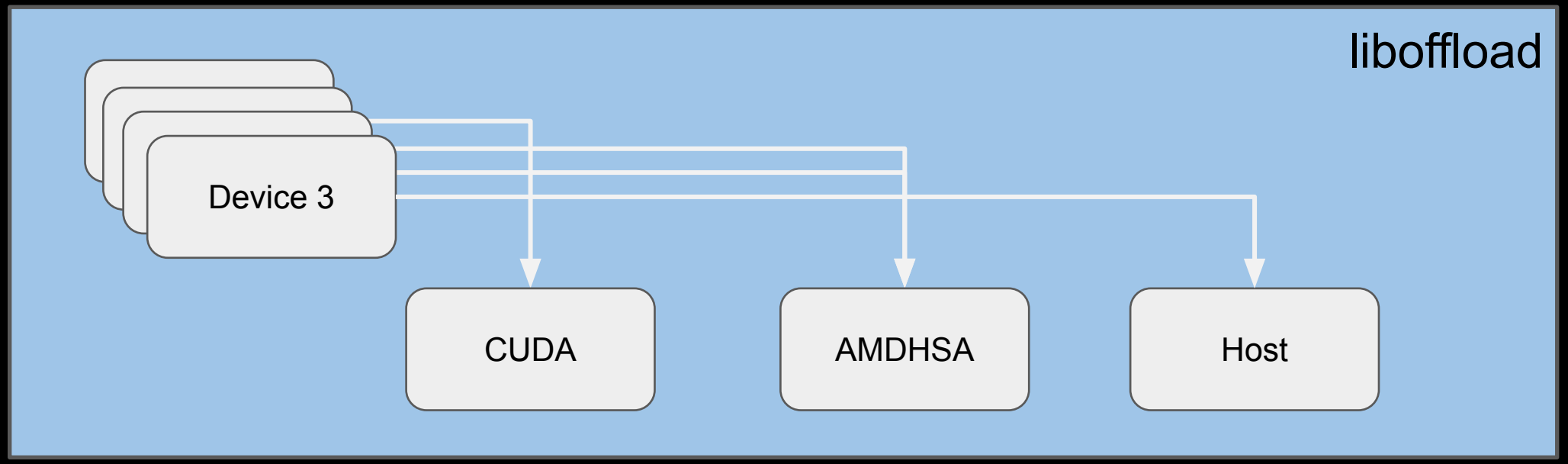
- Generates a public API that calls the internal implementation
- Every API function
  - Returns an error
  - Has a **WithCodeLoc** version for richer error messages

```
OL_APIEXPORT ol_result_t OL_APICALL olInit() {
  if (llvm::offload::isTracingEnabled())
    llvm::errs() << "---> olInit";

  ol_result_t Result = llvmErrorToOffloadError(olInit_impl());

  if (llvm::offload::isTracingEnabled()) {
    llvm::errs() << "()";
    llvm::errs() << "-> " << Result << "\n";
    if (Result && Result->Details)
      llvm::errs() << "     *Error Details* " << Result->Details << " \n";
  }
  return Result;
}
```

```
llvm::Error olInit_impl() { … }
```

Runtime — Liboffload Design

- Objects exposed as **opaque handles**
- Error messages are rich
  - Custom error message
  - Error code
- Exposes **devices** with properties
  - The underlying **platform** is a property of the device



Offloading Example

Example — Compiling an Image

- First we need a GPU program to execute
- Use the direct approach from before
  - Lets use the **libc** library

```c
#include <gpuintrin.h>
#if defined(__AMDGPU__)
const char *platform = "AMDGPU";
#elif defined(__NVPTX__)
const char *platform = "NVPTX";
#endif

__gpu_kernel void kernel(uint32_t x) {
    fprintf(stderr, "Hello from %s thread %d!\n", platform, __gpu_thread_id(0));
}
```

```
$ clang kernel.c --target=amdgcn-amd-amdhsa -mcpu=gfx1030 -lc -flto -o amdgpu
$ clang kernel.c --target=nvptx64-nvidia-cuda -march=sm_89 -lc -flto -o nvptx
```

Example — Initialization

- Load the device image and initialize the runtime

```
#include <offload/OffloadAPI.h>

#define OFFLOAD_ERR(X)                                                  \
  if (ol_result_t Err = X)                                              \
    handleError(Err->Details);

int main(int argc, char **argv) {
  std::vector<char> image = readImage(argv[1]);
  OFFLOAD_ERR(olInit());
  ...
}
```

Example — Device Discovery

- Search for any device that can run our image

```
ol_device_handle_t findDevice(std::vector<char> &binary) {
  ol_device_handle_t Device;
  std::tuple data = std::make_tuple(&Device, &binary);
  OFFLOAD_ERR(olIterateDevices([](ol_device_handle_t Device, void *userData) {
        auto &[output, binary] = *reinterpret_cast<decltype(data)*>(userData);
        bool isValid = false;
        OFFLOAD_ERR(olIsValidBinary(Device, binary->begin(), binary->size(), &isValid));
        if (!isValid) return true; // continue iteration
        *output = Device;
        return false; // found a match
      },
      &data));
  return Device;
```

Example — Program and Queue setup

- Load the image on the discovered device
- Create a queue to push work onto

```
...
ol_device_handle_t device = findDevice(image);

ol_program_handle_t Program;
OFFLOAD_ERR(olCreateProgram(device, image.begin(), image.size(), &Program));

ol_queue_handle_t Queue;
OFFLOAD_ERR(olCreateQueue(device, &Queue));
...
```

Example — Kernel Launch

- Look up the kernel by name
- Push a kernel launch to that symbol

```
  ...
  ol_symbol_handle_t kernel;
  OFFLOAD_ERR(olGetSymbol(Program, "kernel", OL_SYMBOL_KIND_KERNEL, &kernel));

  uint32_t arg = 1;
  ol_kernel_launch_size_args_t launch{1, {1,1,1}, {4,1,1}, 0};
  OFFLOAD_ERR(olLaunchKernel(Queue, device, kernel, &arg, sizeof(arg), &launch));

  OFFLOAD_ERR(olSyncQueue(Queue));
  OFFLOAD_ERR(olDestroyQueue(Queue));
  OFFLOAD_ERR(olDestroyProgram(Program));
  OFFLOAD_ERR(olShutDown());
}
```

Example — Compile and run

- Now you too can run random programs on your GPU!
  - A small version of **llvm-gpu-loader** I use for tests

```
$ clang++ my_offload_program.cpp -I<install>/include -lLLVMOffload -o offload
$ ./offload amdgpu
Hello from AMDGPU thread 0!
Hello from AMDGPU thread 1!
Hello from AMDGPU thread 2!
Hello from AMDGPU thread 3!
$ ./offload nvptx
Hello from NVPTX thread 0!
Hello from NVPTX thread 1!
Hello from NVPTX thread 2!
Hello from NVPTX thread 3!
```

Conclusion — Future Work

- SPIR-V
  - Need to create a SPIR-V version of **<gpuintrin.h>**
  - Needs support in the runtime
- MLIR
  - Should have an **liboffload** target like we have for HIP and CUDA
- API
  - Needs fine-grained initialization options
  - Split **libomptarget** from **offload** entirely
  - Need to finalize the API!
- Reach out if you’re interested in using or contributing!

Disclaimer

The information presented in this document is for informational purposes only and may contain technical inaccuracies, omissions, and typographical errors. The information contained herein is subject to change and may be rendered inaccurate for many reasons, including but not limited to product and roadmap changes, component and motherboard version changes, new model and/or product releases, product differences between differing manufacturers, software changes, BIOS flashes, firmware upgrades, or the like. Any computer system has risks of security vulnerabilities that cannot be completely prevented or mitigated. AMD assumes no obligation to update or otherwise correct or revise this information. However, AMD reserves the right to revise this information and to make changes from time to time to the content hereof without obligation of AMD to notify any person of such revisions or changes.

THIS INFORMATION IS PROVIDED ‘AS IS.” AMD MAKES NO REPRESENTATIONS OR WARRANTIES WITH RESPECT TO THE CONTENTS HEREOF AND ASSUMES NO RESPONSIBILITY FOR ANY INACCURACIES, ERRORS, OR OMISSIONS THAT MAY APPEAR IN THIS INFORMATION. AMD SPECIFICALLY DISCLAIMS ANY IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY, OR FITNESS FOR ANY PARTICULAR PURPOSE. IN NO EVENT WILL AMD BE LIABLE TO ANY PERSON FOR ANY RELIANCE, DIRECT, INDIRECT, SPECIAL, OR OTHER CONSEQUENTIAL DAMAGES ARISING FROM THE USE OF ANY INFORMATION CONTAINED HEREIN, EVEN IF AMD IS EXPRESSLY ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

AMD, the AMD Arrow logo, EPYC, Instinct and combinations thereof are trademarks of Advanced Micro Devices, Inc. PCIe is a registered trademark of PCI-SIG Corporation. Other product names used in this publication are for identification purposes only and may be trademarks of their respective companies.


AMD