KHRONOS™ GROUP

Making SPIR-V Modules

John Kessenich

October 2016

Google

Talk Overview

1. Making a SPIR-V Module

What's in a module

Tools to make modules

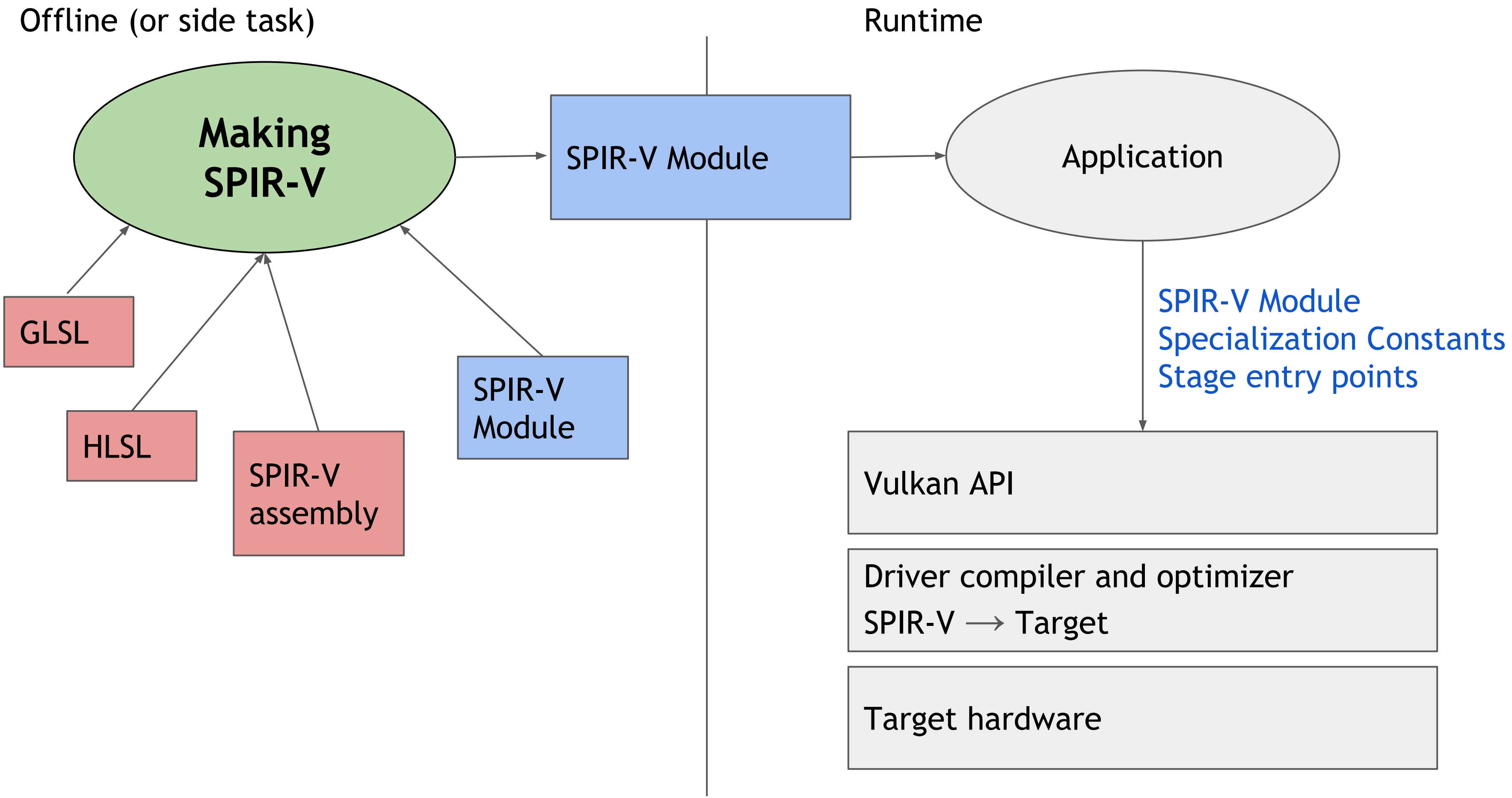
Tools to manipulate modules

2. Managing Size of a Large Collection of Modules

Using SPIR-V features: Specialization Constants

Using normalization and compression

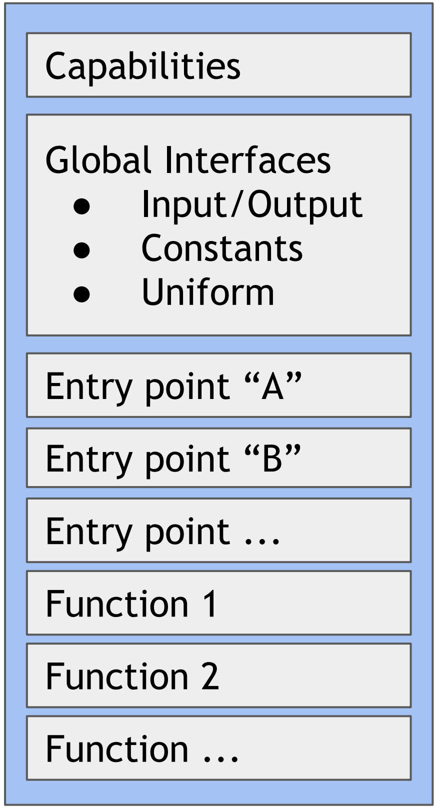
1. Making a SPIR-V Module


Offline (or side task)
Runtime
Making SPIR-V
SPIR-V Module
Application
GLSL
HLSL
SPIR-V assembly
SPIR-V Module
SPIR-V Module
Specialization Constants
Stage entry points
Vulkan API
Driver compiler and optimizer
SPIR-V ⟶ Target
Target hardware

A SPIR-V Module

Binary

Application specifies {stage, entry point} pairs to subset module



Capabilities

Global Interfaces

- Input/Output
- Constants
- Uniform

Entry point “A”

Entry point “B”

Entry point ...

Function 1

Function 2

Function ...

Target must support all declared capabilities, sharerd by all entry points

Includes specialization constants

Multiple entry points share interfaces and functions

one .spv file == one SPIR-V module

Projects on GitHub
Each contains multiple tools

Will discuss today:

- **Glslang:** https://github.com/KhronosGroup/glslang
- **SPIR-V Tools:** https://github.com/KhronosGroup/SPIRV-Tools
- **Shaderc:** https://github.com/google/shaderc

Also see

- **SPIR-V Cross:** https://github.com/KhronosGroup/SPIRV-Cross
  - SPIR-V reflection and translation to higher-level languages
- **SMOL-V:** https://github.com/aras-p/smol-v
  - Compression

Glslang

- **Khronos reference GLSL validator**
  - Command-line options derived from being a validator
- **Kept widely portable by community, across many platforms**
- **GLSL/ESSL → SPIR-V compiler**
  - `glslangValidator -V -o module.spv shader.frag`
  - Need to use Vulkan features (GL_KHR_vulkan_glsl)
    - No loose uniforms; need to use blocks
      - `uniform blockName { <uniform members> };`
    - All **uniform**/**buffer** blocks, samplers, etc. needing bindings and sets
      - `layout(binding = 0, set = 1) <resource declaration>`
    - All **in**/**out** variables need locations
      - `layout(location = 5) <variable declaration>`
    - Precision qualifiers work with desktop shaders

Glslang (continued)

- **HLSL → SPIR-V compiler (new from Google, Valve, LunarG)**
  - Active project, working for many large shaders now
    - All SteamVR HLSL shaders are working without modification
    - Additional features in progress for other shaders
  - Many command-line options to manage mapping of I/O to Vulkan

```
-D          input is HLSL
-e          specify entry-point name
-S <stage>  uses explicit stage specified
--shift-sampler-binding [stage] num     set base binding number for samplers
--shift-texture-binding [stage] num     set base binding number for textures
--shift-UBO-binding [stage] num         set base binding number for UBOs
--auto-map-bindings                     automatically bind uniform variables
--flatten-uniform-arrays                flatten uniform texture & sampler arrays
--no-storage-format                     use Unknown image format
```

- **Reflection of HLL**
- **Remapper: discuss in size section**

Glslang as a Library

- **Glslang is mostly libraries**
  - Can be linked into by other tools
  - glslangValidator is just an example provided wrapper, easy to add others
  - See glslang/StandAlone/StandAlone.cpp

```
glslang::InitializeProcess();

glslang::TShader shader;
shader.setStrings(...file content...);
shader.parse(...);

glslang::TProgram program;
program.addShader(shader);
program.link();

glslang::FinalizeProcess();
```

SPIR-V Tools

spirv-dis: Disassembler

- Binary SPIR-V file → human readable assembly form
- Based on a reusable SPIR-V binary-parser API (use it if you make tool!)

spirv-as: Assembler

- Human-readable assembly form → binary form

⇒ round-trip editing works

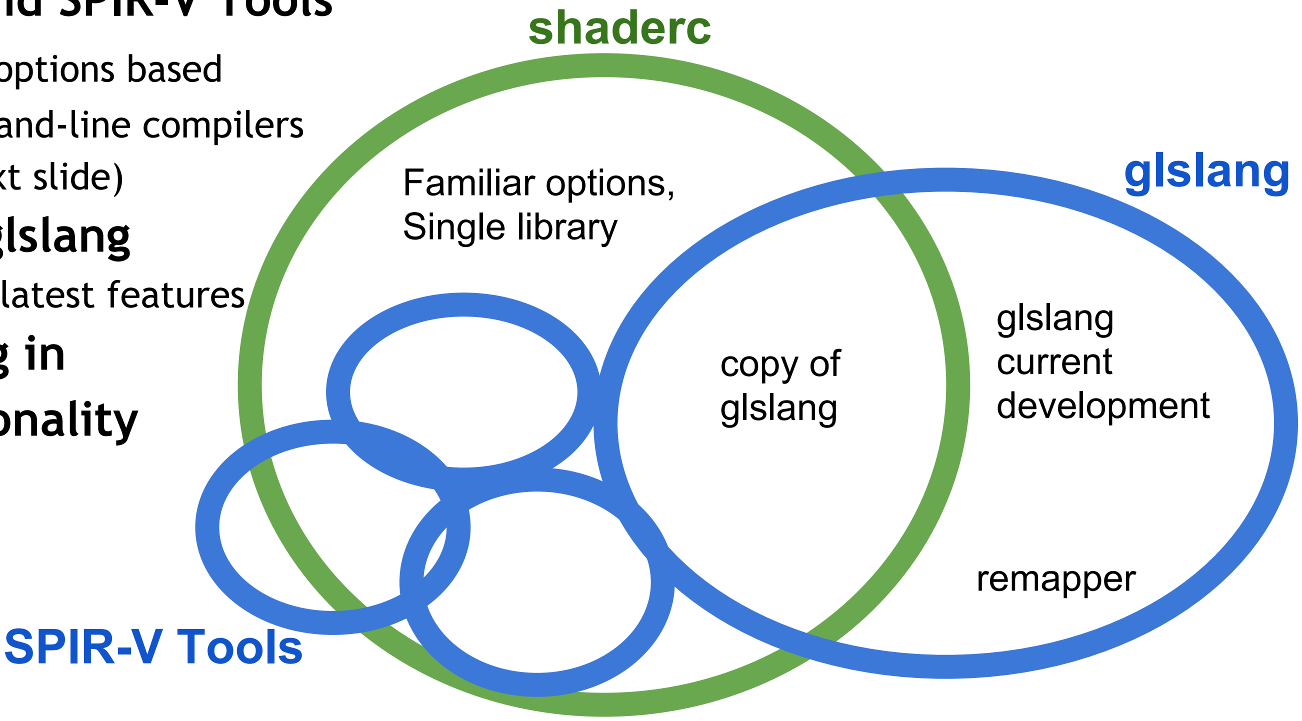
spirv-val: Validator

- Individual module
- Within-module validation (there is different SPIR-V validation done in Vulkan API validation layers

spirv-opt: Optimizer, work in progress

Shaderc

- **Wraps glslang and SPIR-V Tools**
  - Command-line options based on other command-line compilers
  - **glslc** … (see next slide)
- **Uses a copy of glslang**
  - Can be missing latest features
- **Ahead of glslang in #include functionality**



Shaderc (continued)

```
# Preprocess
glslc -E shader.glsl
# Compile
glslc -c shader.vert
# Disassemble
glslc -S shader.vert
# Optimize
glslc -c -Os shader.vert

# Specify shader stage
glslc -c -fshader-stage=vertex shader.glsl
# Specify language version and profile
glslc -c -std=310es shader.vert
# Specify target environment
glslc -c --target-env=vulkan shader.vert
```

```
# Specify output file name
glslc -c shader.vert -o shader.spv
# Specify output format
# E.g., output SPIR-V binary code as
# a C-style initializer list
glslc -c -mfmt=c shader.vert

# Define
glslc -E -DVALUE=42 shader.glsl
# Include
glslc -E -I../include shader.glsl
# Generate dependencies for builds
glslc -MD -c shader.glsl

# Warnings and errors
glslc -c -Werror shader.vert
```

Shaderc as a Library

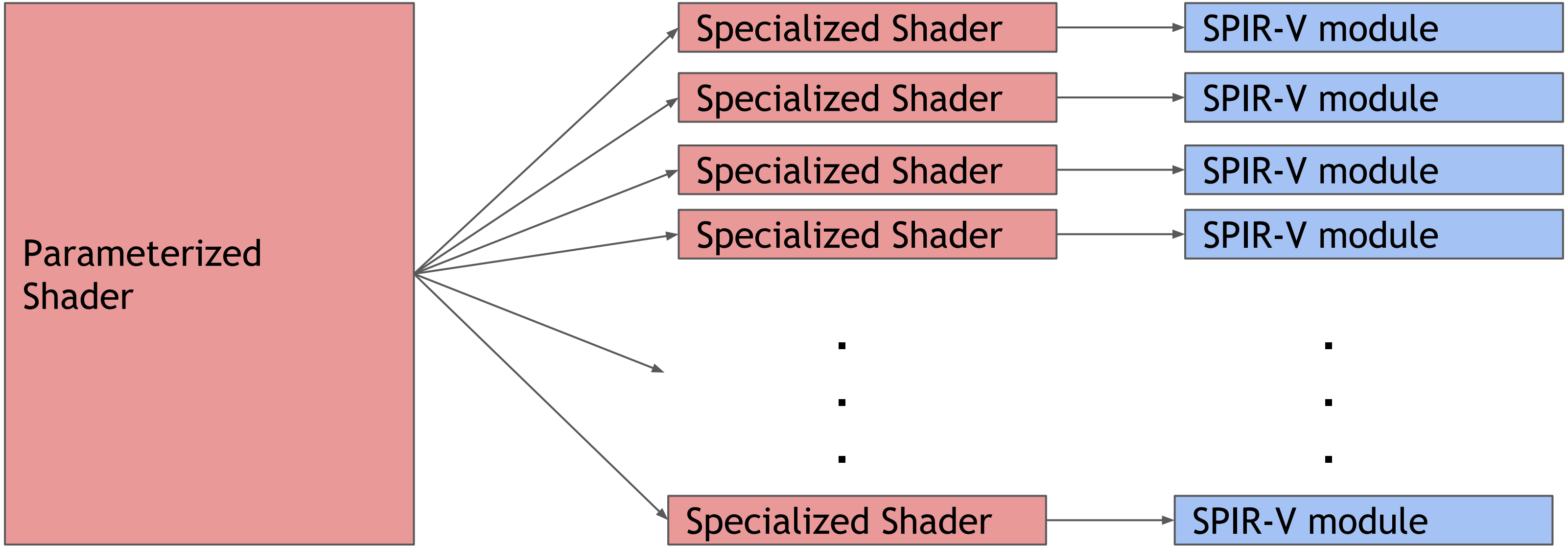
```
#include <shaderc/shaderc.hpp>

shaderc::Compiler compiler;
shaderc::CompileOptions options;

const std::string source =
    "#version 310 es\nvoid main() { int v = VALUE; }\n";

options.AddMacroDefinition("VALUE", "42");
options.SetOptimizationLevel(shaderc_optimization_level_size);

auto pp = compiler.PreprocessGlsl(source.c_str(), source.size(),
    source, shaderc_glsl_vertex_shader, "shader.glsl", options);
std::cout << "Preprocessed shader:\n"
          << std::string(pp.cbegin(), pp.cend()) << std::endl;
// Preprocessed shader:
// #version 310 es
// void main(){ int v = 42; }

auto compiling = compiler.CompileGlslToSpvAssembly(
    source, shaderc_glsl_vertex_shader, "shader.glsl", options);
std::string assembly(compiling.cbegin(), compiling.cend());
std::cout << "Compiled SPIR-V assembly:\n" << assembly << std::endl;
// Compiled SPIR-V assembly:
// ...
//           OpCapability Shader
//    %1 = OpExtInstImport "GLSL.std.450"
//           OpMemoryModel Logical GLSL450
//           OpEntryPoint Vertex %4 "main"
// %void = OpTypeVoid
// ...

auto assembling = compiler.AssembleToSpv(assembly.c_str(), assembly.size());
std::vector<uint32_t> spirv(assembling.cbegin(), assembling.cend());
std::cout << "Final binary # words: " << spirv.size() << std::endl;
// Final binary # words: 54

const std::string bad_source =
    "#version 310 es\n void main() { the_ultimate_shader }";
auto error = compiler.CompileGlslToSpv(
    source, shaderc_glsl_vertex_shader, "shader.glsl", options);
std::cerr << error.GetErrorMessage() << std::endl;
// shader.glsl:2: error: 'the_ultimate_shader' : undeclared identifier
// shader.glsl:2: error: '' :  syntax error
```

2. Managing Size of a Large Collection of Modules

Size

- **SPIR-V is verbose, designed**
  - For ease of processing by SPIR-V tools and drivers
  - To be explicit, not inference based, e.g., type redundancy
  - a portable non-lossy standard
- **Individual shaders are small enough**
  - Glslang output includes **OpName**, redundant load/store
  - **spirv-remap --dce all --opt all --strip all -o out_dir -i module.spv**
  - **spirv-opt ...**
  - **glslc -Os ...**

Issue: Making a large number of variations of similar shaders



Offline

Managing Large Collections

Two completely different approaches:

1. **Recommended: Less SPIR-V, using SPIR-V features**
   - SPIR-V was designed to represent variations efficiently
   - Requires using the features provided
2. **Legacy: Compression**
   - Works best on collections of related modules
   - Involves normalization followed by compression

Less SPIR-V, Using SPIR-V Features

- **Specialization constants**
  - Fewer SPIR-V modules
  - Defer some options/sizes to runtime
- **Lots of entry points in a single SPIR-V module**
  - Share functions
  - Share uniform interface
  - Same capabilities
  - Tools still immature

Will focus on Specialization Constants here...

Specialization Constants

1. Declare specialization constants in GLSL or HLSL:

   ```
   layout(constant_id = 13) const int size = 9;
   ```

   - Provide default values (9 above)
   - Generates SPIR-V with specialization constant ids (13 above)
2. Distribute SPIR-V with specialization constants
3. Specialize at runtime
   - Set constants when creating pipeline
   - Those not set use their default
4. Driver compiler will optimize knowing the specialized constant value

Vulkan 1.0.31

9.7. Specialization Constants

```
VkPipelineShaderStageCreateInfo.pSpecializationInfo

typedef struct VkSpecializationInfo {
    uint32_t                           mapEntryCount;
    const VkSpecializationMapEntry*    pMapEntries;
    size_t                             dataSize;
    const void*                        pData;
} VkSpecializationInfo;
```

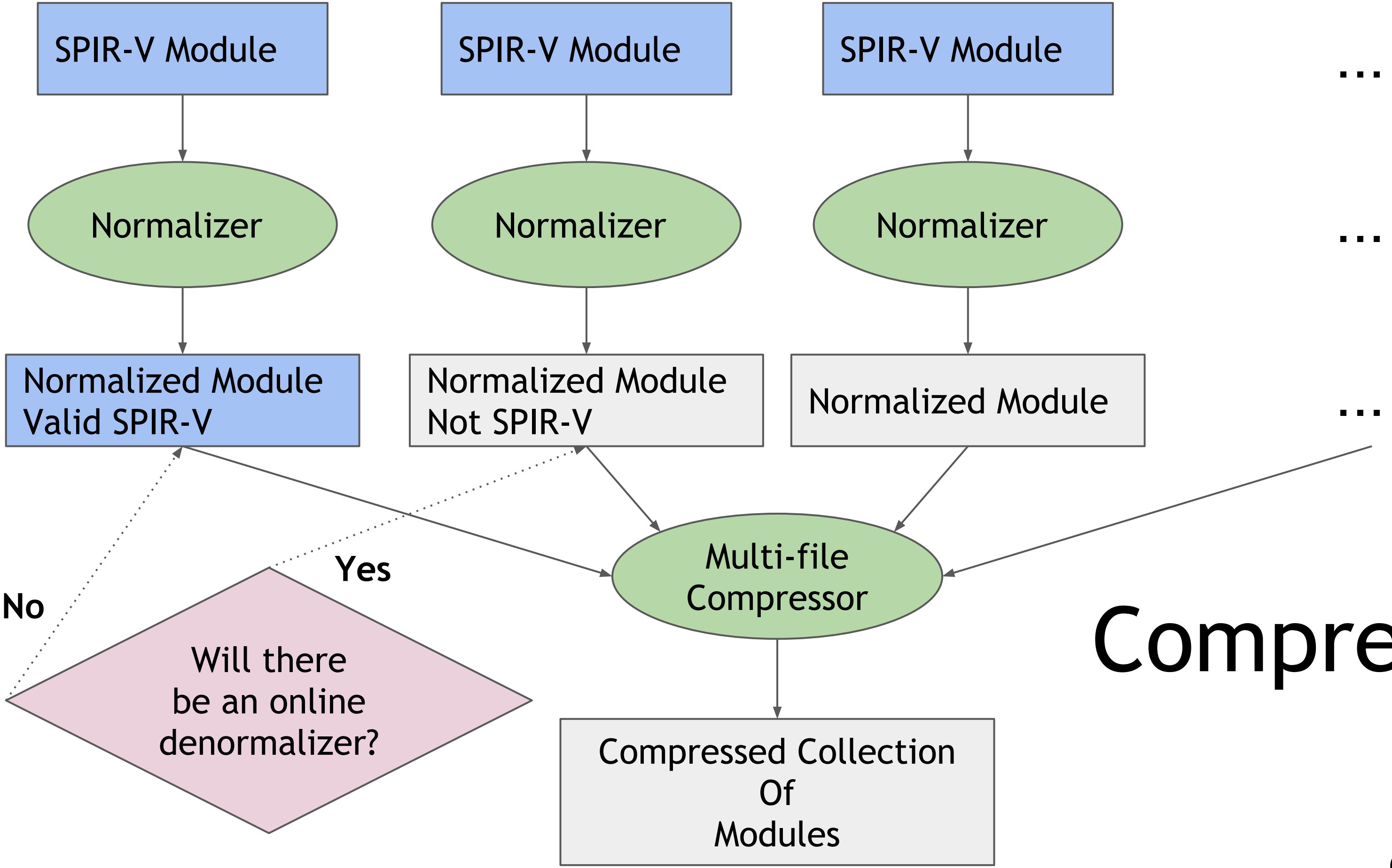
Change size 9 → 8

```
struct SpecializationData {
    int32_t data0;
    ...
};

const VkSpecializationMapEntry entries[] =
{
    {
        13,                                   // constant_id
        offsetof(SpecializationData, data0),  // offset
        sizeof(SpecializationData::data0)     // size
    },
    ...
};

SpecializationData data;
data.data0 = 8;                         // new value for the constant
...

const VkSpecializationInfo info =
{
    <number of constants being set>,    // mapEntryCount
    entries,                            // pMapEntries
    sizeof(data),                       // dataSize
    &data,                              // pData
};
```

Compression

- **Much better across multiple modules**
- **Much better if normalized first**
    - Can normalize to SPIR-V or something else
    - If something else, need to denormalize before decompression

Compression



SPIR-V Module

SPIR-V Module

SPIR-V Module

...

Normalizer

Normalizer

Normalizer

...

Normalized Module
Valid SPIR-V

Normalized Module
Not SPIR-V

Normalized Module

...

Multi-file
Compressor

Yes

No

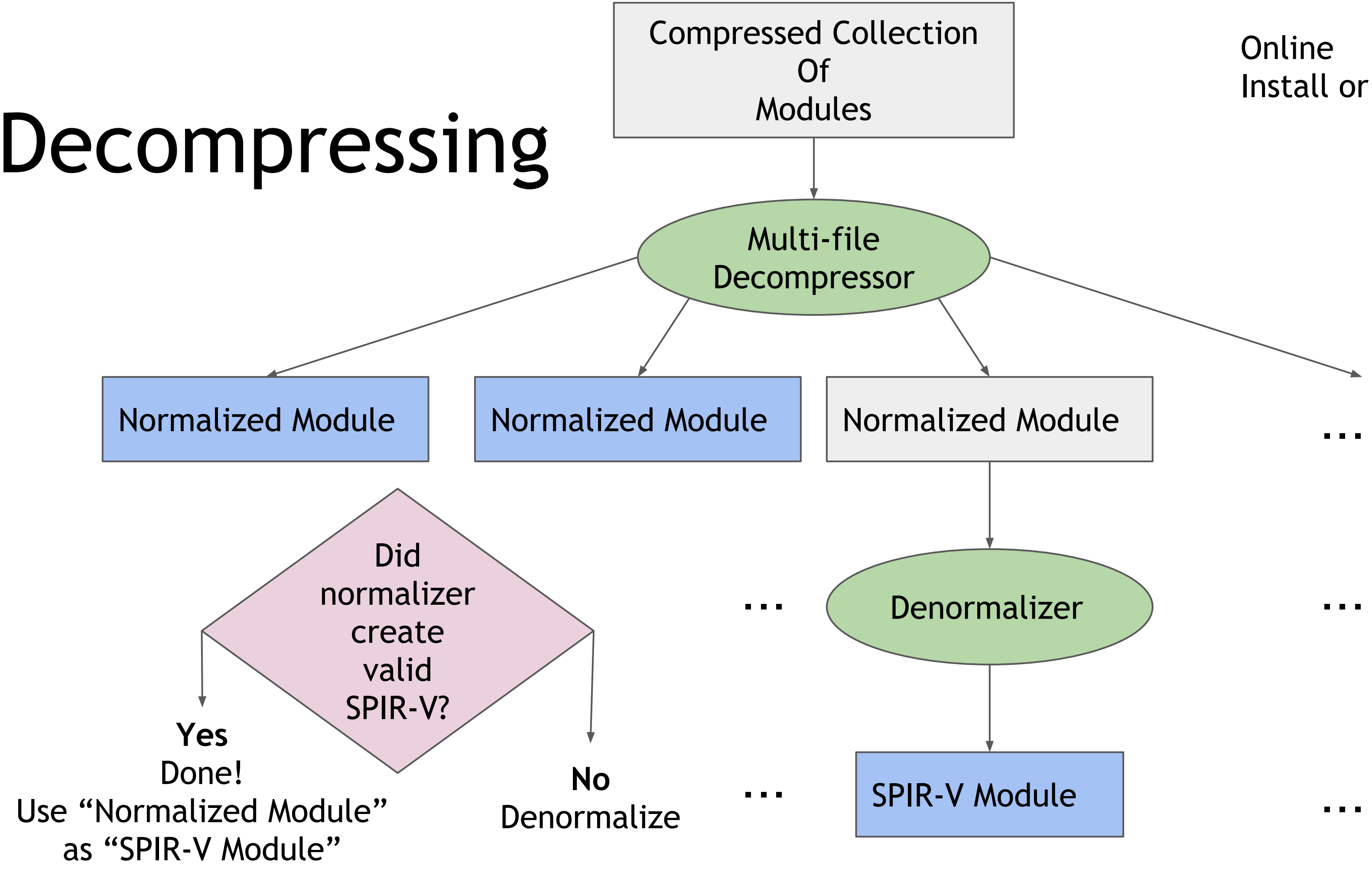
Will there
be an online
denormalizer?

Compressed Collection
Of
Modules

Offline

Decompressing

Online
Install or Runtime



Normalize and Compress with Glslang’s Remapper

1. **Normalize step (offline):**

   ```
   spirv-remap --map all -o out_dir -i mod1.spv mod2.spv …
   ```

2. **Compress**

   ```
   tar -cf - out_dir | lzma -z > compressed.lzma
   ```

3. **Distribute**
4. **Decompress**

   ```
   lzma -d < compressed.lzma | tar -xvf -
   ```

5. **No denormalization needed**

Also see SMOL-V: https://github.com/aras-p/smol-v

- Smaller than SPIR-V normalizer
- Needs denormalizer

Future work

- **More offline optimizations**
- **Merge multiple modules into a single module**
  - Same capabilities
  - Share same utility functions

End