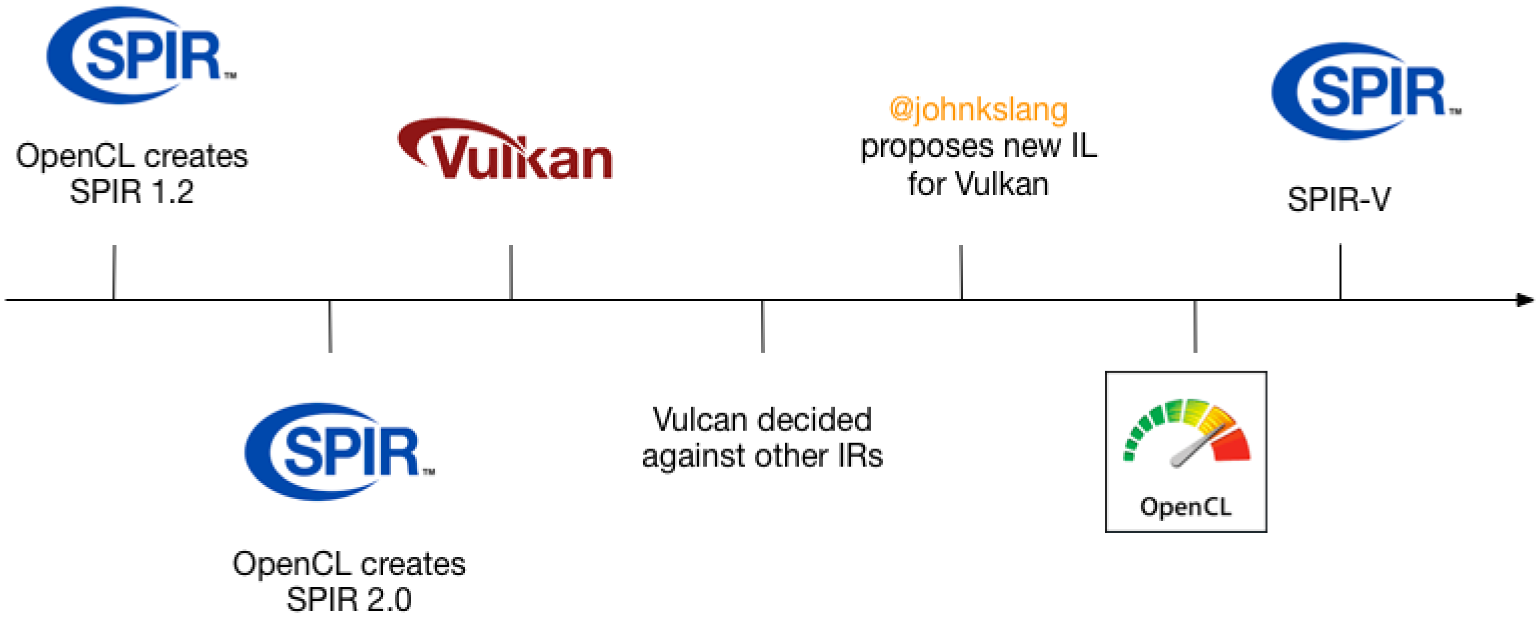
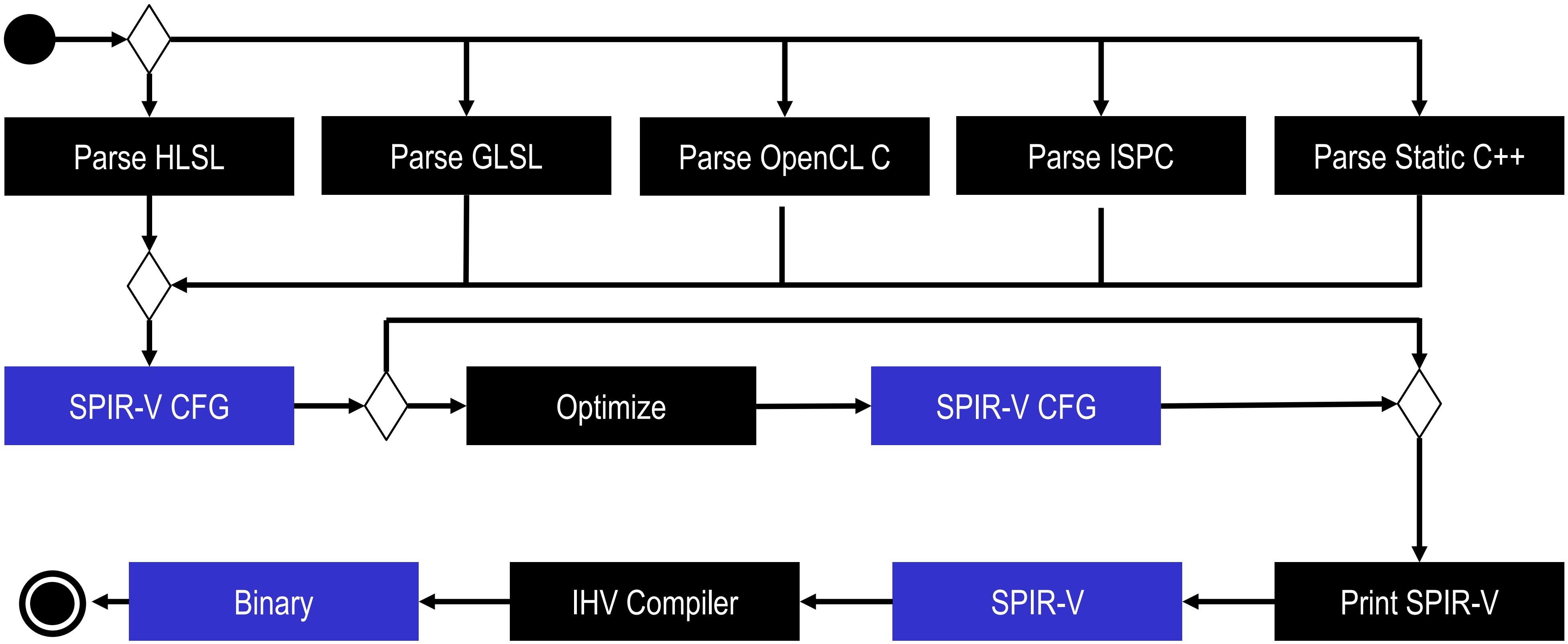
Introduction to SPIR-V Shaders

Neil Hickey
Compiler Engineer, ARM

SPIR History



SPIR-V Purpose



Developer Ecosystem

- Multiple Developer Advantages:
  - Same front-end compiler for multiple platforms
  - Reduces runtime kernel compilation time
  - Don’t have to ship shader/kernel source code
  - Drivers are simpler and more reliable

Standard
Portable
Intermediate
Representation

Vulkan and OpenCL

	SPIR 1.2	SPIR 2.0	SPIR-V 1.0
LLVM Interaction	Uses LLVM 3.2	Uses LLVM 3.4	100% Khronos defined Round-trip lossless conversion
Compute Constructs	Metadata/Intrinsics	Metadata/Intrinsics	Native
Graphics Constructs	No	No	Native
Supported Language Feature Sets	OpenCL C 1.2	OpenCL C 1.2 OpenCL C 2.0	OpenCL C 1.2 – 2.0 OpenCL C++ and GLSL
OpenCL Ingestion	OpenCL C 1.2 Extension	OpenCL C 2.0 Extension	OpenCL 2.1 Core OpenCL 1.2 / 2.0 Extensions
Vulkan Ingestion	-	-	Vulkan 1.0 Core

KHRONOS GROUP

Compiler flow

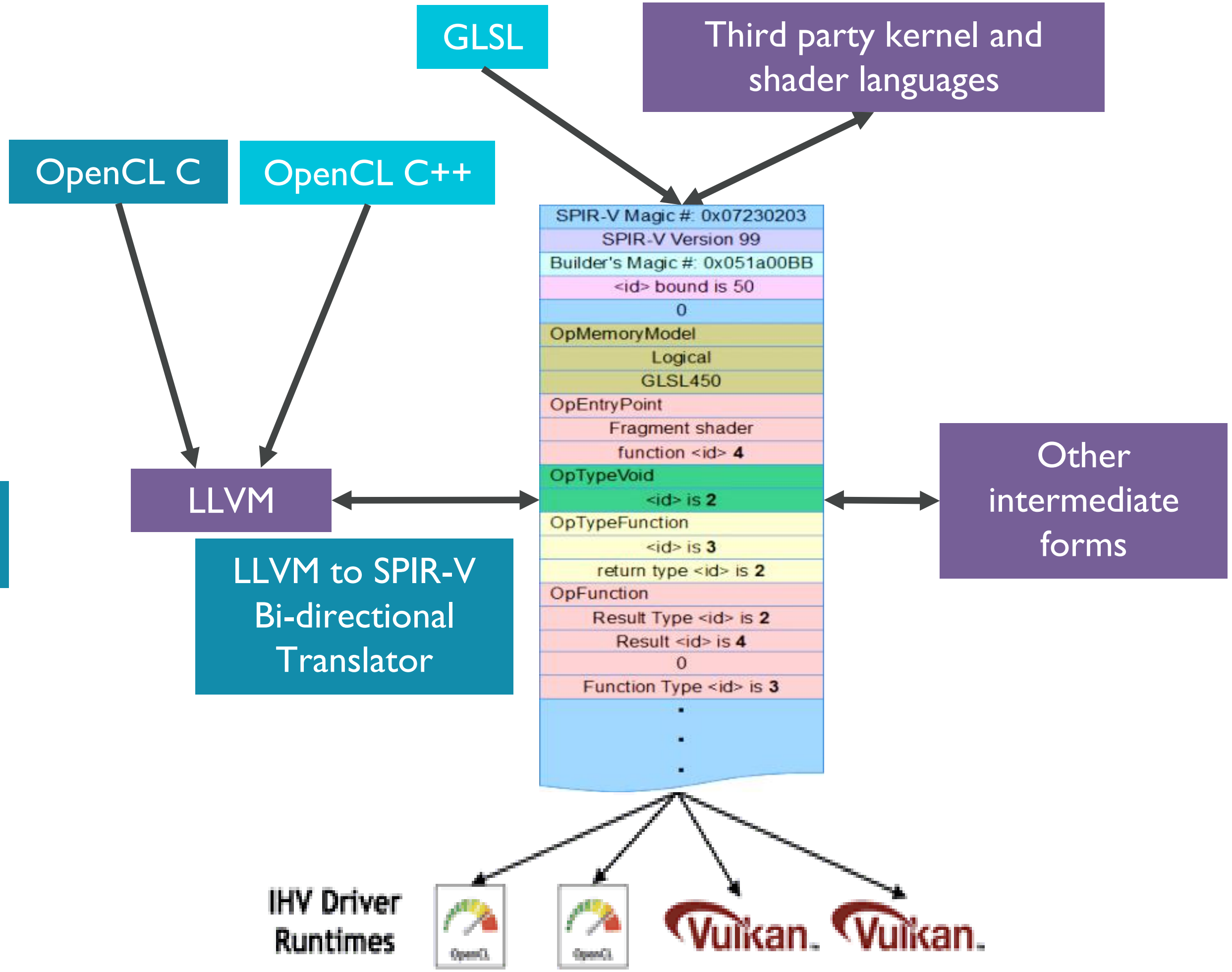
Khronos has open sourced these tools and translators

Khronos plans to open source these tools soon

SPIR-V Tools

SPIR-V Validator

SPIR-V (Dis)Assembler



SPIR-V

- 32-bit word stream
- Extensible and easily parsed
- Retains data object and control flow information for effective code generation and translation

SPIR-V Capabilities

- OpenCL and Vulkan
- Capabilities define feature sets
- Separate capabilities for Vulkan shaders and OpenCL kernels
- Validation layer checks correct capabilities requested

```
OpCapability Addresses
OpCapability Linkage
OpCapability Kernel
OpCapability Vector16
OpCapability Int16
```


SPIR-V Extensions

- OpExtension
- New functionality
- New instructions
- New semantics

OpExtInstImport
“OpenCL.std”

Vulkan shaders vs. GL shaders

- Program GLSL/ESSL shaders in high level language
- Ship high level source with application
- Graphics drivers compile at runtime
- Each driver needs a full compilation tool chain

- Shaders in binary format
- Compile offline
- Ship intermediate language with application
- Graphics drivers “just” lower from IL
- Higher level compilation can be shared among vendors (provided by Khronos)

Vulkan shaders vs. GL shaders

```
#version 310 es
precision mediump float;
uniform sampler2D s;
in vec2 texcoord;
out vec4 color;

void main()
{
        color = texture(s, texcoord);
}
```

```
; SPIR-V
; Version: 1.0
; Generator: Khronos Glslang Reference Front End; 1
; Bound: 20
; Schema: 0
          OpCapability Shader
     %1 = OpExtInstImport "GLSL.std.450"
          OpMemoryModel Logical GLSL450
          OpEntryPoint Fragment %4 "main" %9 %17
          OpExecutionMode %4 OriginUpperLeft
          OpSource ESSL 310
          OpName %4 "main"
          OpName %9 "color"
          OpName %13 "s"
          OpName %17 "texcoord"
          OpDecorate %9 RelaxedPrecision
          OpDecorate %13 RelaxedPrecision
          OpDecorate %13 DescriptorSet 0
          OpDecorate %14 RelaxedPrecision
          OpDecorate %17 RelaxedPrecision
          OpDecorate %18 RelaxedPrecision
          OpDecorate %19 RelaxedPrecision
     %2 = OpTypeVoid
     %3 = OpTypeFunction %2
```

```
%6 = OpTypeFloat 32
     %7 = OpTypeVector %6 4
     %8 = OpTypePointer Output %7
     %9 = OpVariable %8 Output
    %10 = OpTypeImage %6 2D 0 0 0 1 Unknown
    %11 = OpTypeSampledImage %10
    %12 = OpTypePointer UniformConstant %11
    %13 = OpVariable %12 UniformConstant
    %15 = OpTypeVector %6 2
    %16 = OpTypePointer Input %15
    %17 = OpVariable %16 Input
     %4 = OpFunction %2 None %3
     %5 = OpLabel
    %14 = OpLoad %11 %13
    %18 = OpLoad %15 %17
    %19 = OpImageSampleImplicitLod %7 %14 %18
          OpStore %9 %19
          OpReturn
          OpFunctionEnd
```

Khronos SPIR-V Tools

- Reference frontend (glslang)

```
glslangValidator –V –o shader.spv shader.frag
```

- SPIR-V disassembler (spirv-dis)

```
spirv-dis -o shader.spvasm shader.spv
```

- SPIR-V assembler (spirv-as)

```
spirv-as –o shader.spv shader.spvasm
```

- SPIR-V reflection (spirv-cross)

```
spirv-cross shader.spv
```

Vulkan shaders in a high level language

- GL_KHR_vulkan_glsl
- Exposes SPIR-V features
- Similar to GLSL with some changes
- Extends #version 140 and higher on desktop and #version 310 es for mobile content

Vulkan_glsl removed features

- Default uniforms
- Atomic-counter bindings
- Subroutines
- Packed block layouts

Vulkan_glsl new features

- Push constants
- Separate textures and samplers
- Descriptor sets
- Specialization constants
- Subpass inputs

Push Constants

- **Push constants replace non-opaque uniforms**
  - Think of them as small, fast-access uniform buffer memory
- **Update in Vulkan with vkCmdPushConstants**

```
// New
layout(push_constant, std430) uniform PushConstants {
    mat4 MVP;
    vec4 MaterialData;
} RegisterMapped;

// Old, no longer supported in Vulkan GLSL
uniform mat4 MVP;
uniform vec4 MaterialData;

// Opaque uniform, still supported
uniform sampler2D sTexture;1
```

Separate textures and samplers

- sampler contains just filtering information
- texture contains just image information
- combined in code at the point of texture lookup

```
uniform sampler s;
uniform texture2D t;
in vec2 texcoord;
...
void main()
{
        fragColor = texture(sampler2D(t,s), texcoord);
}
```

Descriptor sets

- Bound objects can optionally define a descriptor set
- Allows bound objects to be updated in one block
- Allows objects in other descriptor sets to remain the same
- Enabled with the set = ... syntax in the layout specifier

```
layout(set = 0, binding = 0) uniform sampler s;
layout(set = 1, binding = 0) uniform texture2D t;
```

Specialization constants

- Allows for special constants to be created whose value is overridable at pipeline creation time.
- Can be used in expressions
- Can be combined with other constants to form new specialization constants
- Declared using layout(constant_id=...)
- Can have a default value if not overridden at runtime

```
layout(constant_id = 1) const int arraySize = 12;

vec4 data[arraySize];
```

Specialization constants(2)

- gl_WorkGroupSize can be specialized with values for the x,y and z component.

```
layout(local_size_x_id = 2, local_size_z_id = 3) in;
```

- These specialization constants can be set at pipeline creation time by using vkSpecializationMapInfo

```
const VkSpecializationMapEntry entries[] =
{
{  1,                    // constantID
   0*sizeof(uint32_t),   // offset
   sizeof(uint32_t)      // size
},
};
```


Specialization constants(3)

```
const uint32_t data[] = { 16};
const VkSpecializationInfo info =
{
        1,        // mapEntryCount
   entries,  // pMapEntries
   1*sizeof(uint32_t),  // dataSize
        data,    // pData
};
```

Subpass Inputs

- Vulkan supports subpasses within render passes
- Standardized GL_EXT_shader_pixel_local_storage!

```
// GLSL
#extension GL_EXT_shader_pixel_local_storage : require
__pixel_local_inEXT GBuffer {
    layout(rgba8) vec4 albedo;
    layout(rgba8) vec4 normal;
    ...
} pls;

// Vulkan
layout(input_attachment_index = 0) uniform subpassInput albedo;
layout(input_attachment_index = 1) uniform subpassInput normal;
...
```


Acknowledgements

- Hans-Kristian Arntzen - ARM
- Benedict Gaster - University of the West of England
- Neil Henning - Codeplay