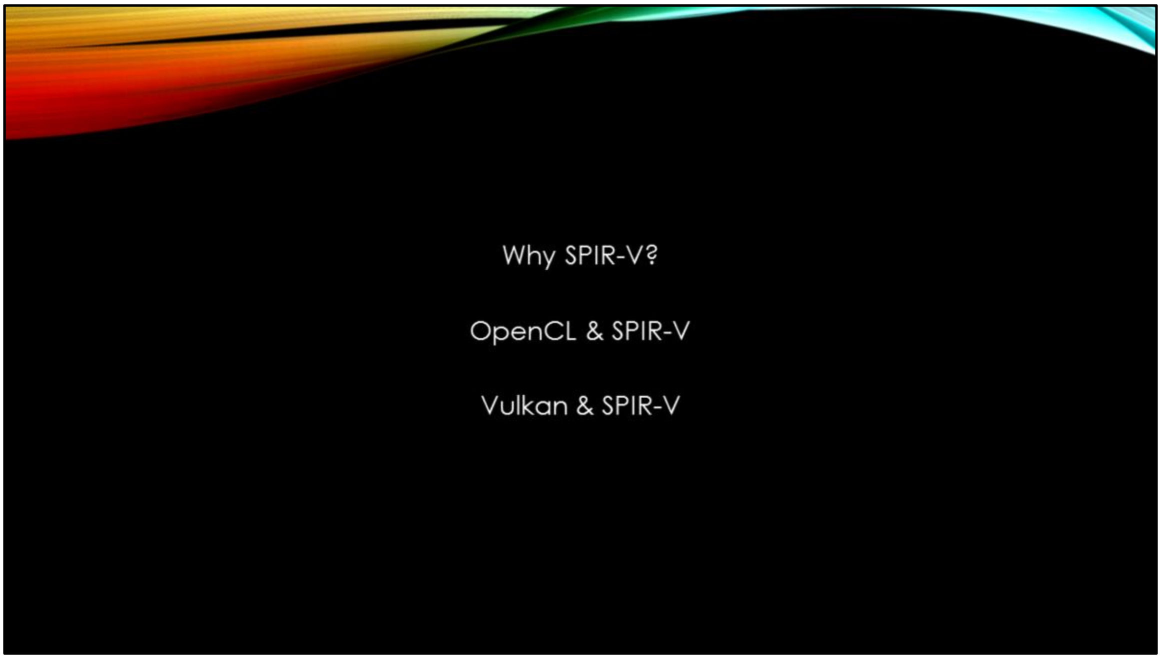
WHY SPIR-V?

Neil Henning

Principal Software Engineer, Vulkan & SPIR-V @ Codeplay

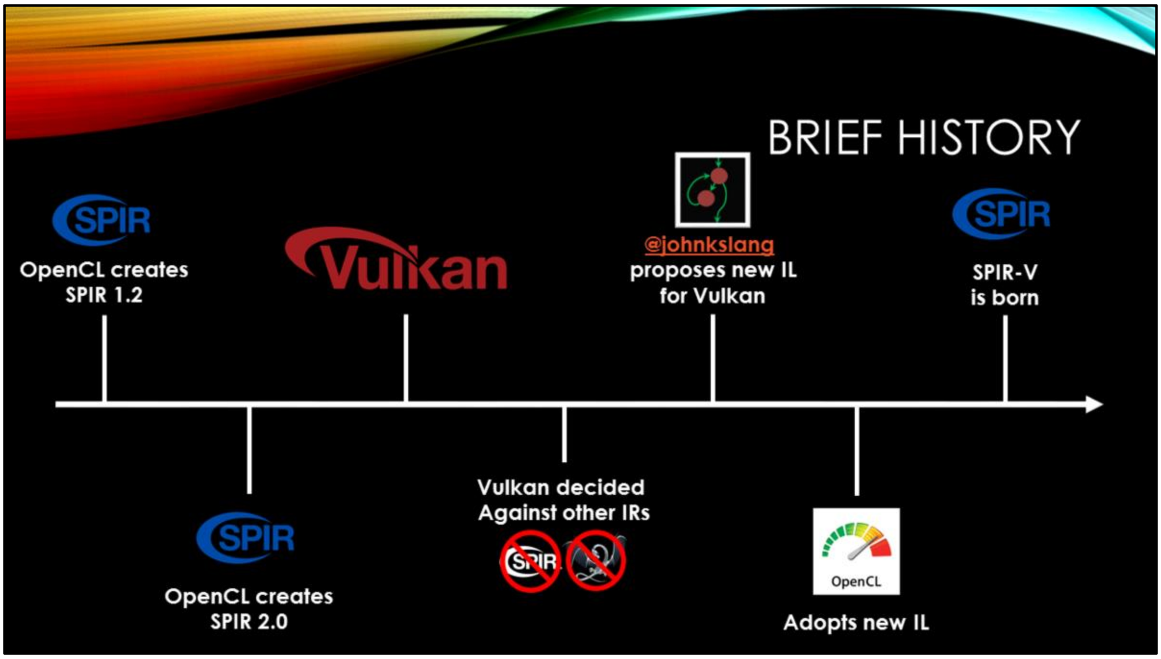


Why SPIR-V?

OpenCL & SPIR-V

Vulkan & SPIR-V

Going to cover;
- Why we have SPIR-V
    - Brief history of SPIR-V
    - Some of the core required features we wanted
- How OpenCL will use SPIR-V
- How Vulkan will use SPIR-V
    - The differences between compute/graphics use
    - Information on one of the new features – Specialization Constants!



BRIEF HISTORY

OpenCL creates SPIR 1.2

OpenCL creates SPIR 2.0

Vulkan

Vulkan decided Against other IRs

@johnkslang proposes new IL for Vulkan

Adopts new IL

SPIR-V is born

SPIR started with SPIR 1.2 (to match OpenCL 1.2) then SPIR 2.0 (to match OpenCL 2.0)

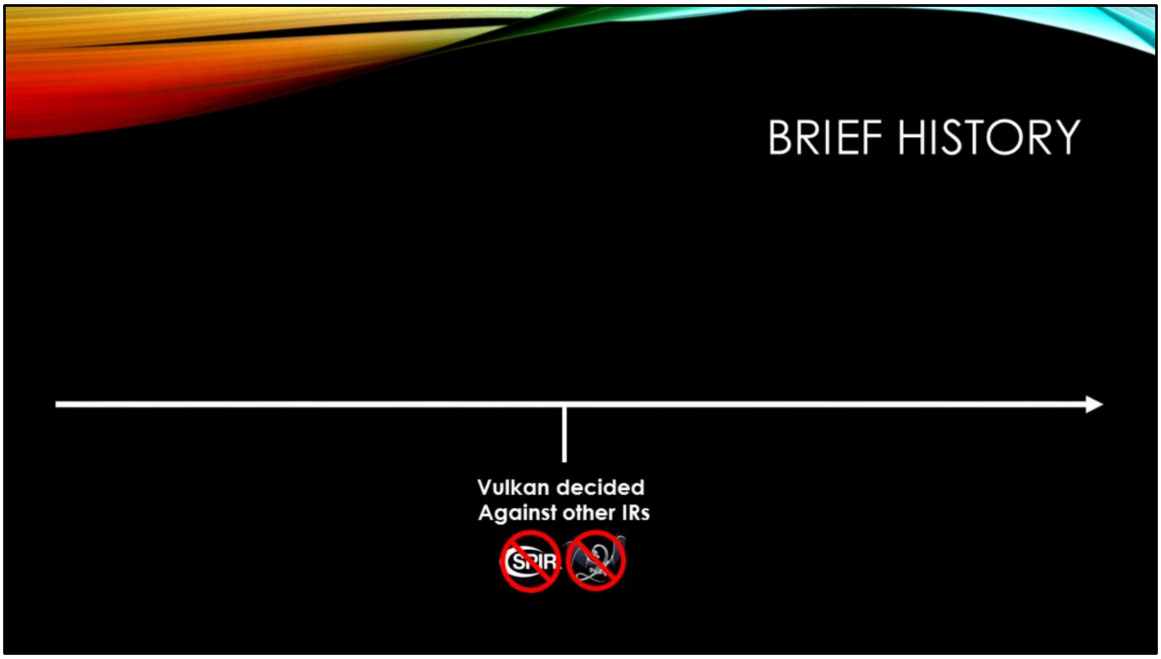
SPIR-V started with Vulkan, we knew we needed a binary shader format, and we were investigating what we could use.

Obvious choice was LLVM IR and extending SPIR 1.2/2.0 – but we ruled them up early on (more info later)

John Kessenich (@johnkslang on twitter) turned up to the group one day with the bones of what would become SPIR-V, huge amount of praise should rightly be sent his way for really jumpstarting the whole endeavour.

Originally SPIR-V was intended for Vulkan only – but OpenCL quickly jumped on board once they realised the benefit SPIR-V gave them too.

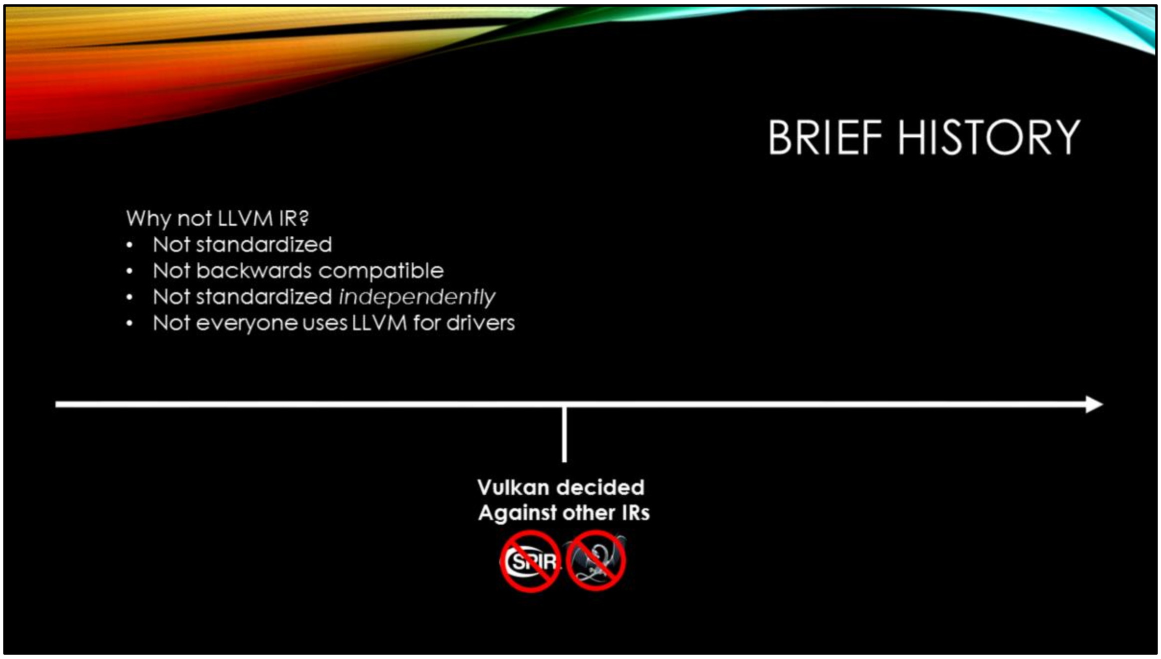
Work work work – crunching to get an initial specification ready for show at GDC’15 – where we presented the specification and SPIR-V was truly born!



BRIEF HISTORY

Vulkan decided
Against other IRs

Lets cover how we ruled out the alternatives first



BRIEF HISTORY

Why not LLVM IR?
- Not standardized
- Not backwards compatible
- Not standardized *independently*
- Not everyone uses LLVM for drivers

Vulkan decided
Against other IRs

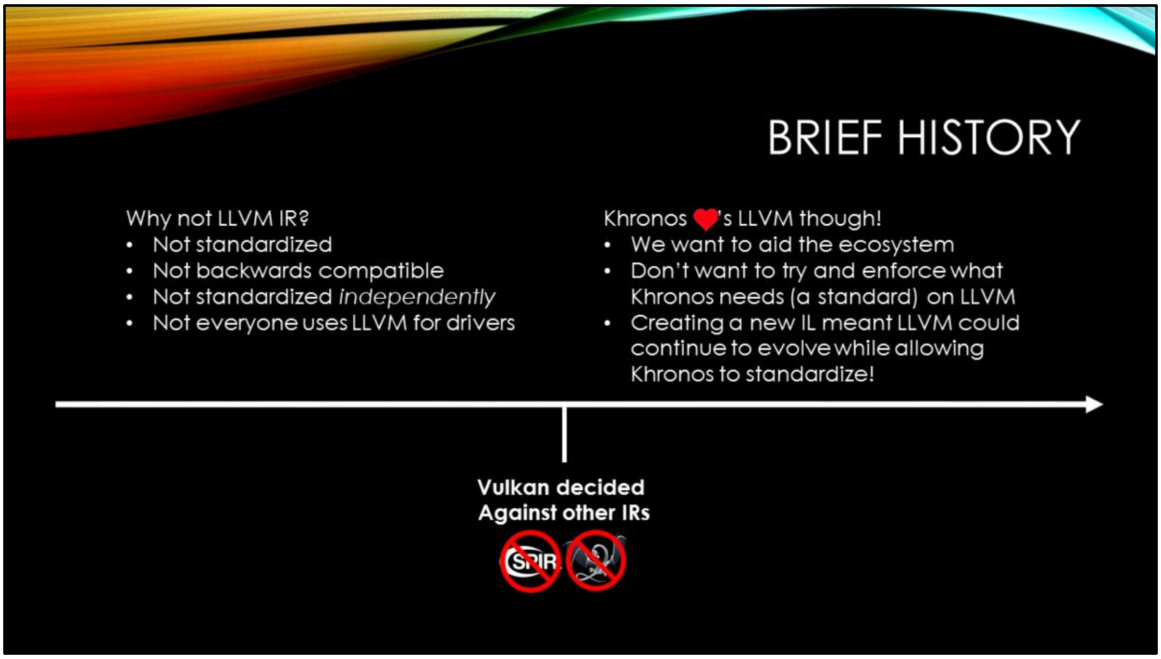
First off – why not LLVM IR?

The heart of it is that it isn't a standard – something that doesn't fit well with Khronos' desire to have cross platform standards for everything to enable easy adoption!

There is no desire to make the IR backwards compatible – EG. if we choose LLVM IR today (version 3.6 for instance) we can't guarantee that the latest versions will be able to produce/consume the same IR.

The IR is not standardized independently – EG. the code **IS** the 'standard'.

Not everyone uses LLVM for drivers. If we choose LLVM IR, we're basically lumping people who don't use LLVM with around 15Mb of extra executable bloat when they don't necessarily require it.



BRIEF HISTORY

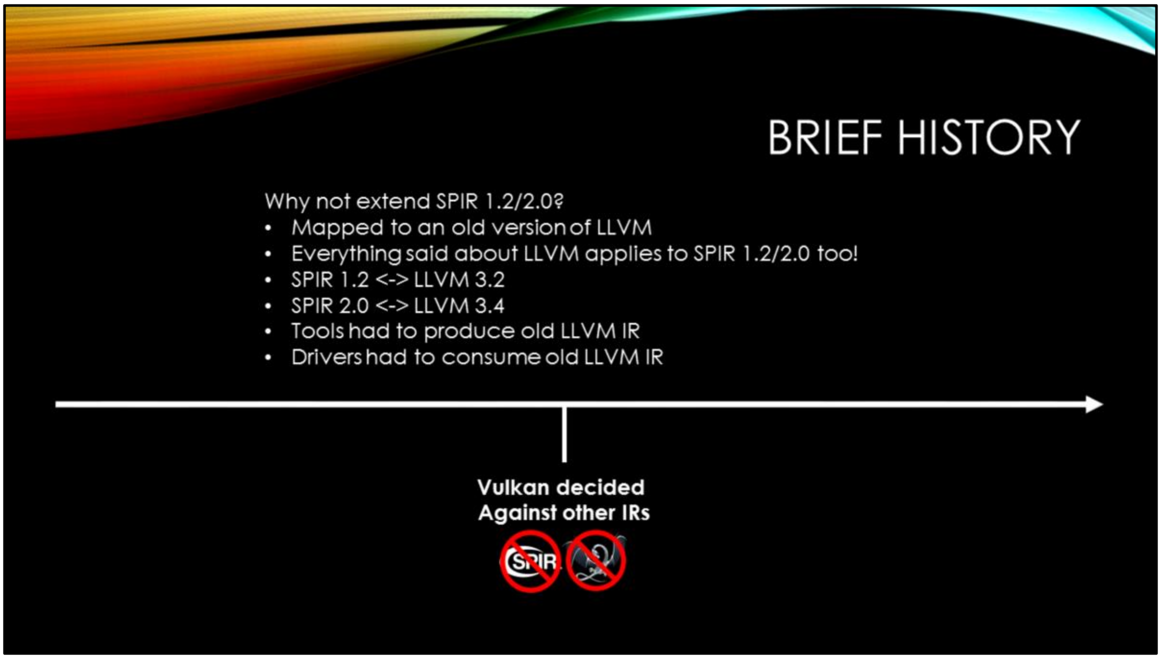
Why not LLVM IR?
- Not standardized
- Not backwards compatible
- Not standardized *independently*
- Not everyone uses LLVM for drivers

Khronos ❤'s LLVM though!
- We want to aid the ecosystem
- Don't want to try and enforce what Khronos needs (a standard) on LLVM
- Creating a new IL meant LLVM could continue to evolve while allowing Khronos to standardize!

Vulkan decided Against other IRs

LLVM has this wonderful philosophy of 'break everything if it is for the better'. If something doesn't work, they change it, end of. This means though that having standards, and backwards compatibility are the first thing to go out the window – not having to support the older 'broken' version is part of the reason why LLVM is such a successful project in my opinion.

We love LLVM at Khronos – that's why it was chosen as the IR for SPIR 1.2/2.0 in the first place! It was only through the process of defining and supporting SPIR 1.2/2.0 that Khronos realised where it just didn't quite fit with what Khronos needed. We didn't want to burden and slow down LLVM by trying to enforce some standard on their IR format, so the logical conclusion was to introduce SPIR-V instead!



BRIEF HISTORY

Why not extend SPIR 1.2/2.0?

- Mapped to an old version of LLVM
- Everything said about LLVM applies to SPIR 1.2/2.0 too!
- SPIR 1.2 <-> LLVM 3.2
- SPIR 2.0 <-> LLVM 3.4
- Tools had to produce old LLVM IR
- Drivers had to consume old LLVM IR

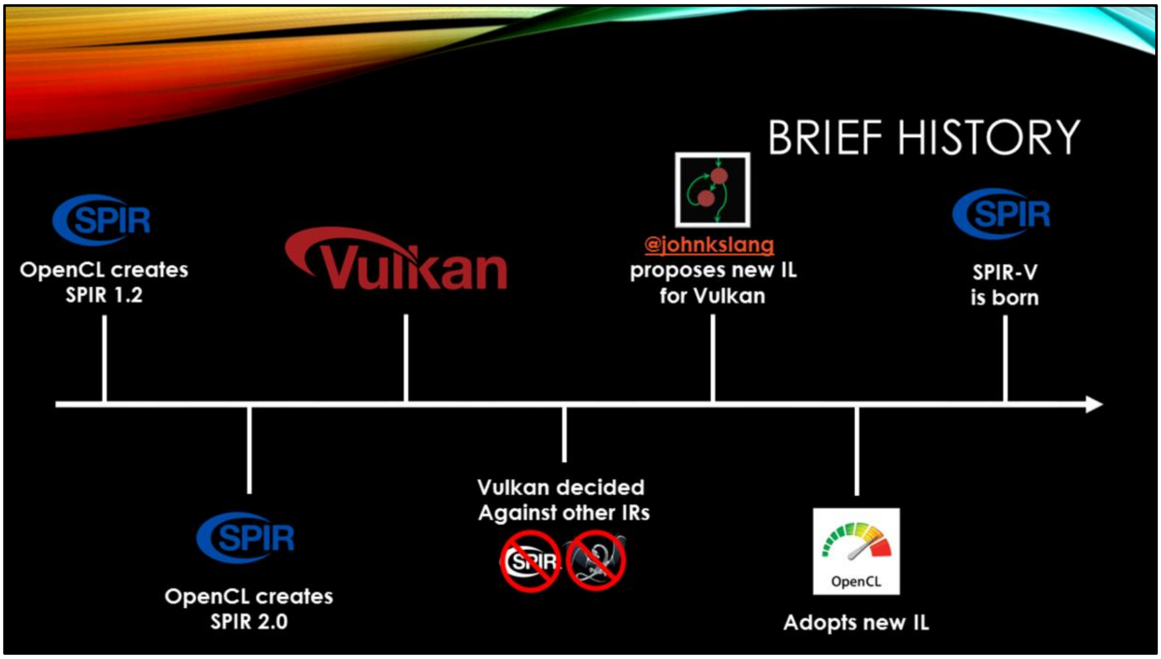
Vulkan decided
Against other IRs

SPIR 1.2/2.0 was just a standard on-top of older versions of LLVM IR.

This meant that you either had to use the older versions of LLVM for your tools (and miss out on all the improvements of tip) or produce older IR from tip LLVM (which I have done for customers in the past and it is more painful than you might imagine).

For consuming LLVM IR – it mostly used to work to consume older IR from tip, but significant changes to the binary metadata format (something which SPIR 1.2/2.0 relied on heavily for things missing in LLVM IR) meant that metadata will be scrubbed, which meant you had to have old bitcode readers too.

All in all it meant for a more troublesome process than it should have been.



BRIEF HISTORY

OpenCL creates SPIR 1.2

OpenCL creates SPIR 2.0

Vulkan

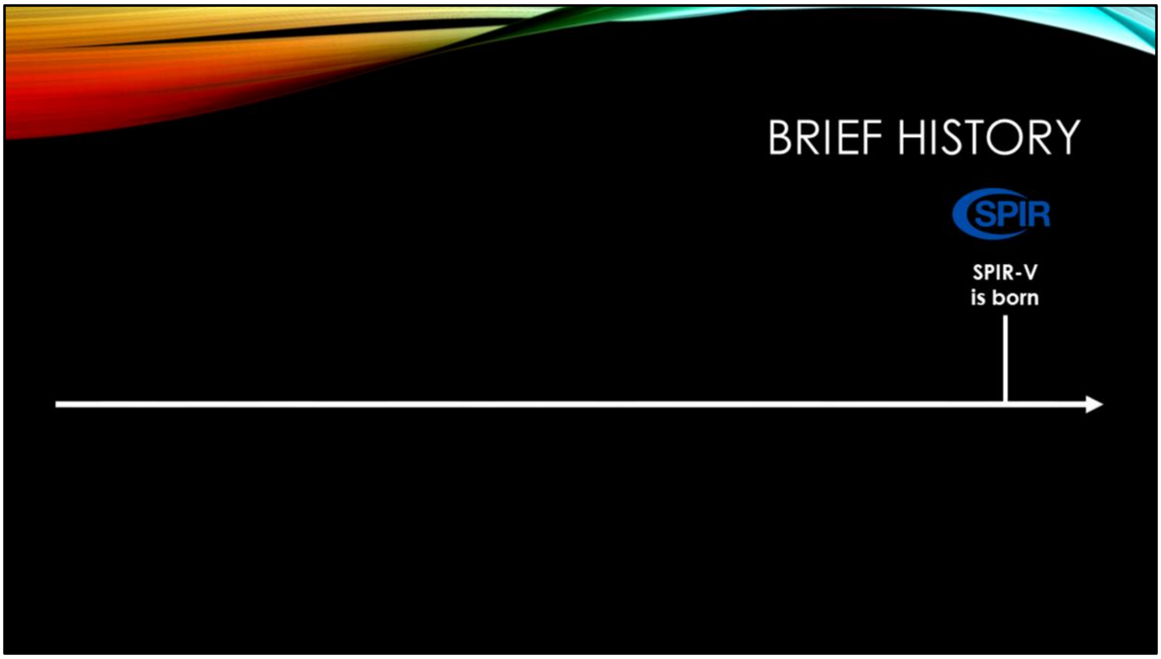
Vulkan decided Against other IRs

@johnkslang proposes new IL for Vulkan

Adopts new IL

SPIR-V is born

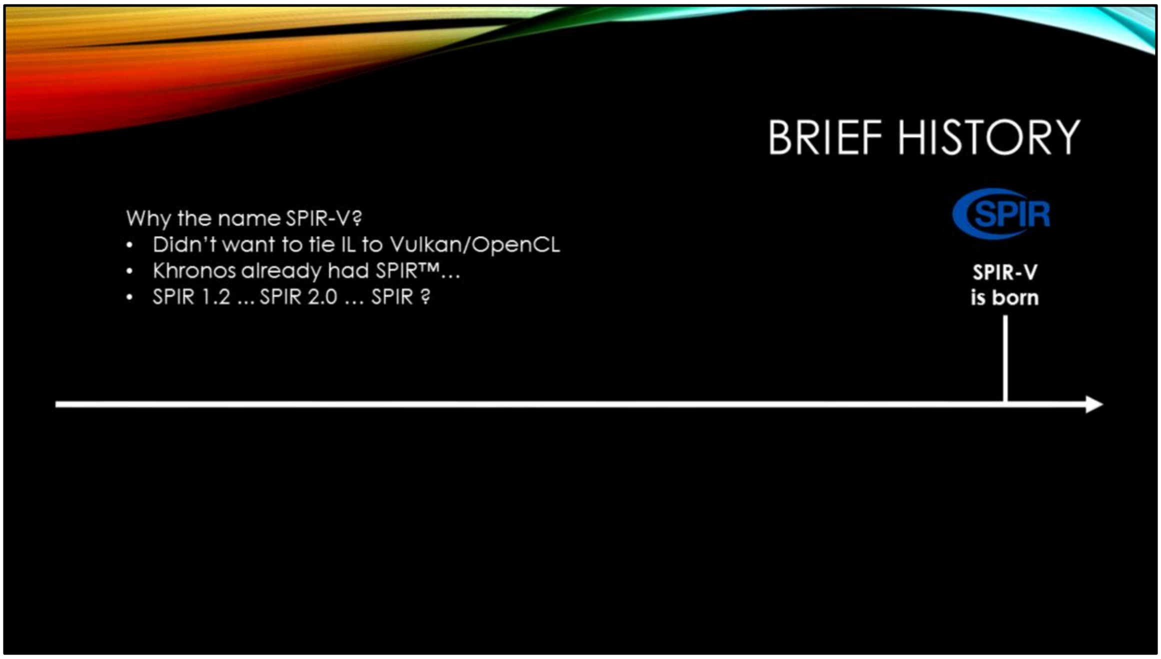
Slide transition…



BRIEF HISTORY

SPIR-V is born

Lets talk about how we came up with the SPIR-V name next



BRIEF HISTORY

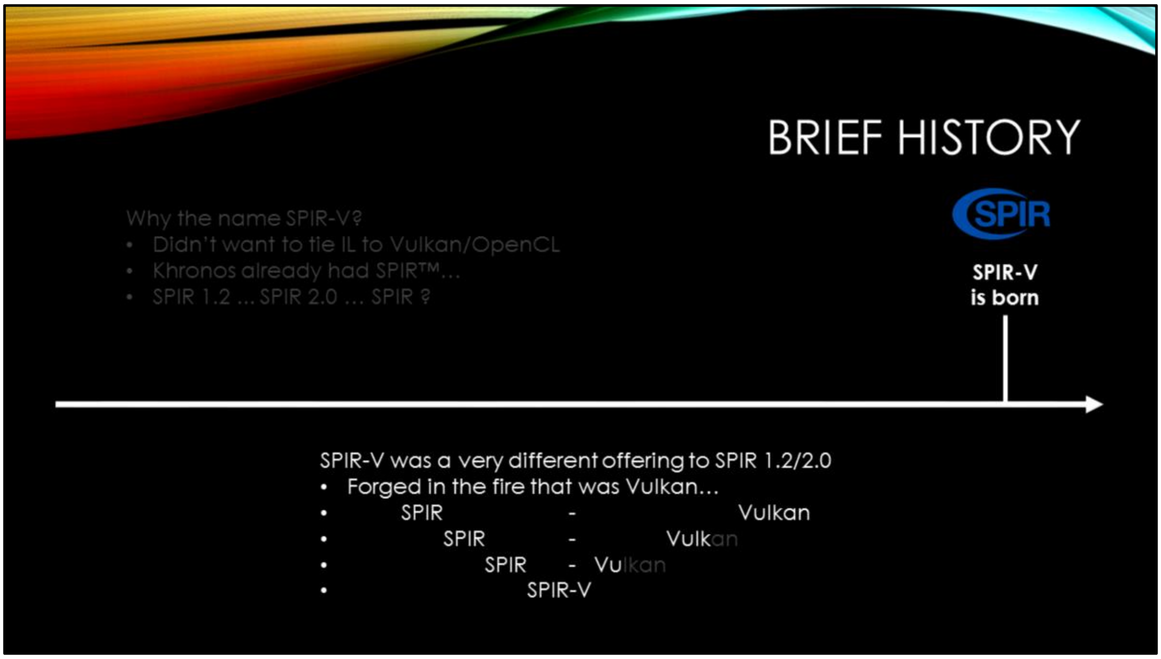
Why the name SPIR-V?
- Didn't want to tie IL to Vulkan/OpenCL
- Khronos already had SPIR™…
- SPIR 1.2 … SPIR 2.0 … SPIR ?

SPIR-V
is born

The name had to be unique, not tied to any other particular API/standard (we want to allow more Khronos APIs and other APIs to use SPIR-V after all), and Khronos already had done all of the legwork to get the SPIR name in the first place.

It was natural for the Khronos groups to simply reuse the existing name.

Now the catch – we knew this was a new offering that, although was under the SPIR Khronos banner, was very different to the previous SPIR offerings.



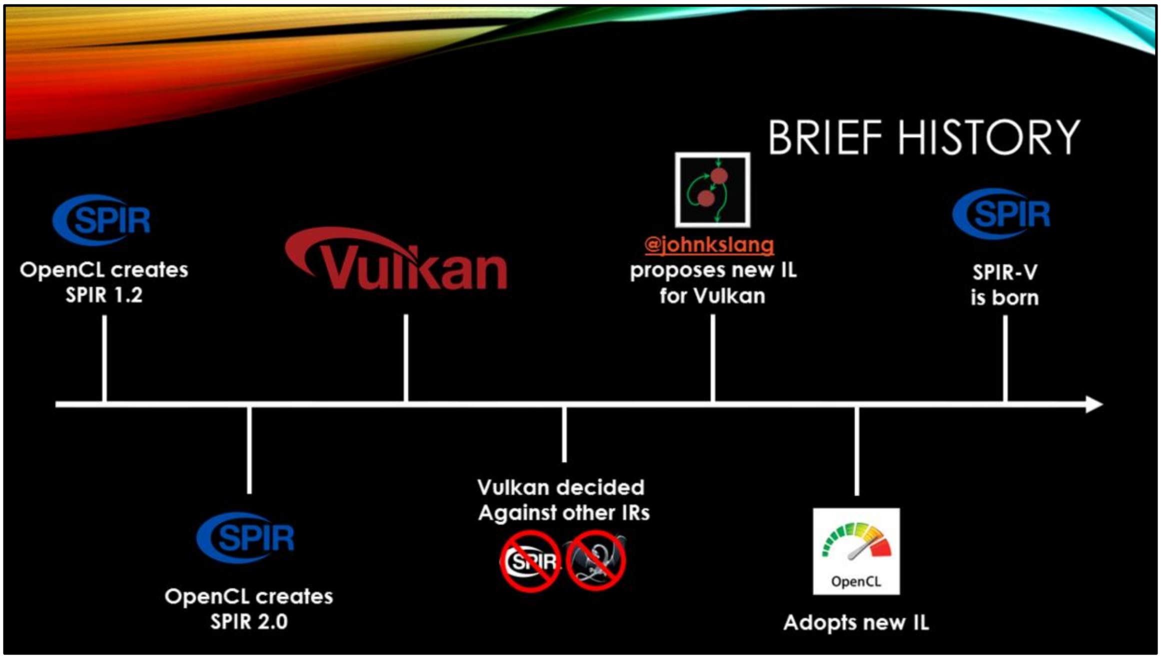
BRIEF HISTORY

Why the name SPIR-V?
- Didn't want to tie IL to Vulkan/OpenCL
- Khronos already had SPIR™…
- SPIR 1.2 … SPIR 2.0 … SPIR ?

SPIR-V is born

SPIR-V was a very different offering to SPIR 1.2/2.0
- Forged in the fire that was Vulkan…
- SPIR - Vulkan
- SPIR - Vulkan
- SPIR - Vulkan
- SPIR-V

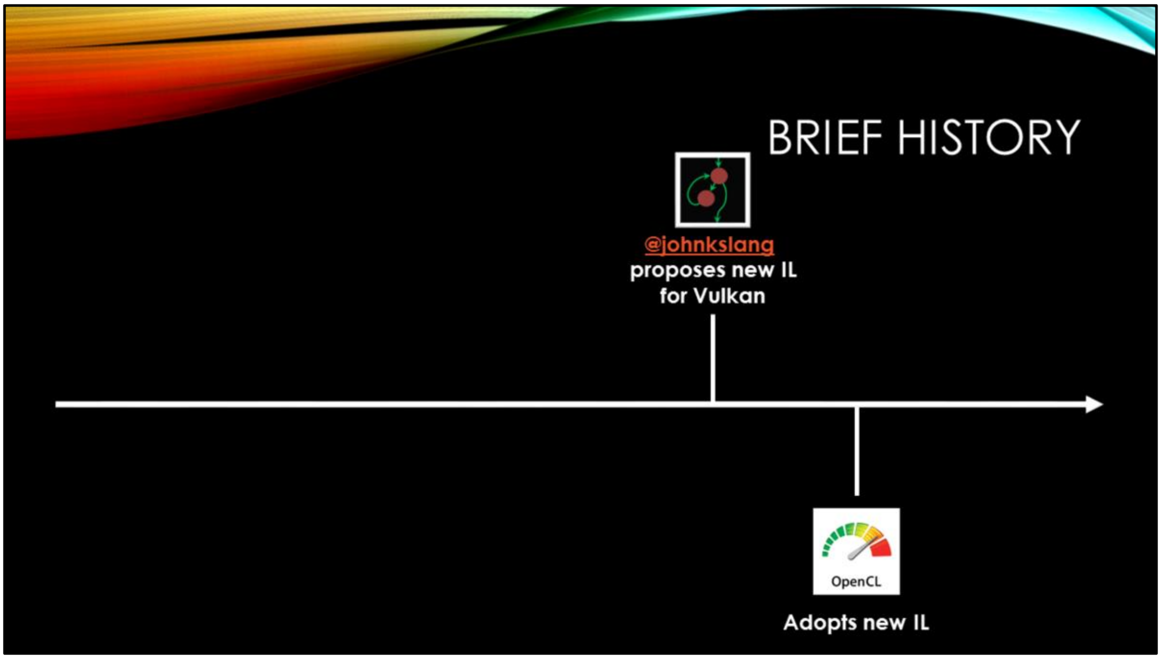
SPIR-V was crafted (much like the One Ring - my precious) in the bowls of a volcano… analogy aside it was Vulkan's inception that bore us this thing we had to name. It made sense then to amalgamate SPIR and the V from Vulkan, and thus we have SPIR-V!



BRIEF HISTORY

- OpenCL creates SPIR 1.2
- OpenCL creates SPIR 2.0
- Vulkan
- Vulkan decided Against other IRs
- @johnkslang proposes new IL for Vulkan
- Adopts new IL
- SPIR-V is born

Slide transition…

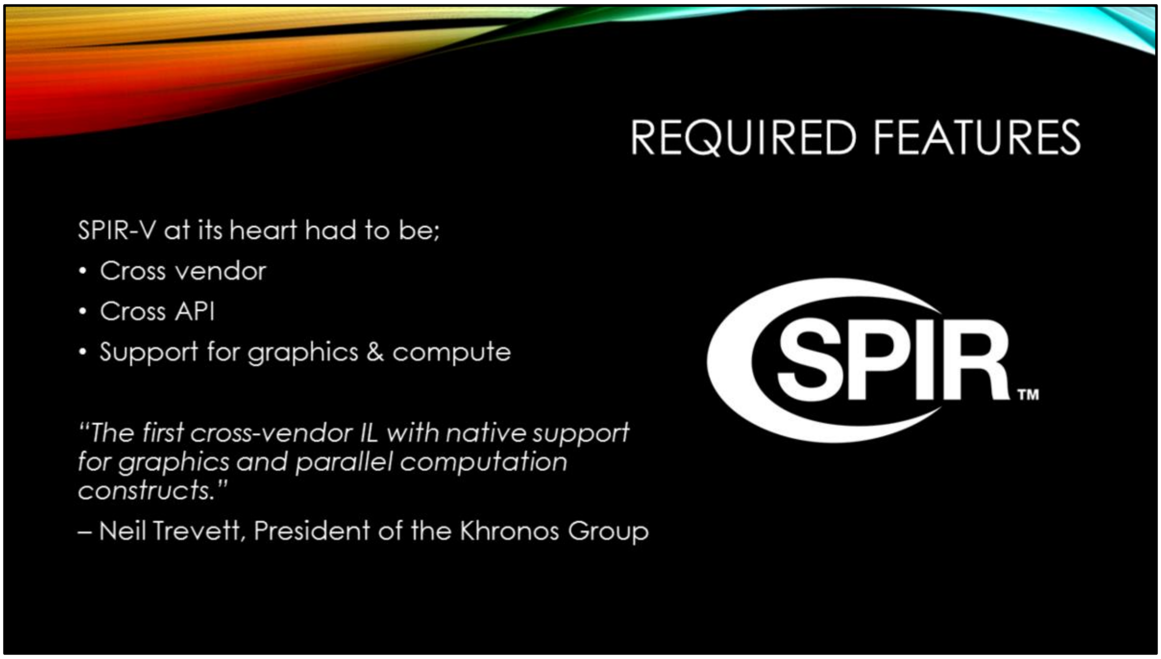


BRIEF HISTORY

@johnkslang proposes new IL for Vulkan

Adopts new IL

Lets talk about the heart of the matter – where we actually defined the IL. I don't have a ton of time here, so I've picked out some of the key features the IL needed to have.



REQUIRED FEATURES

SPIR-V at its heart had to be;

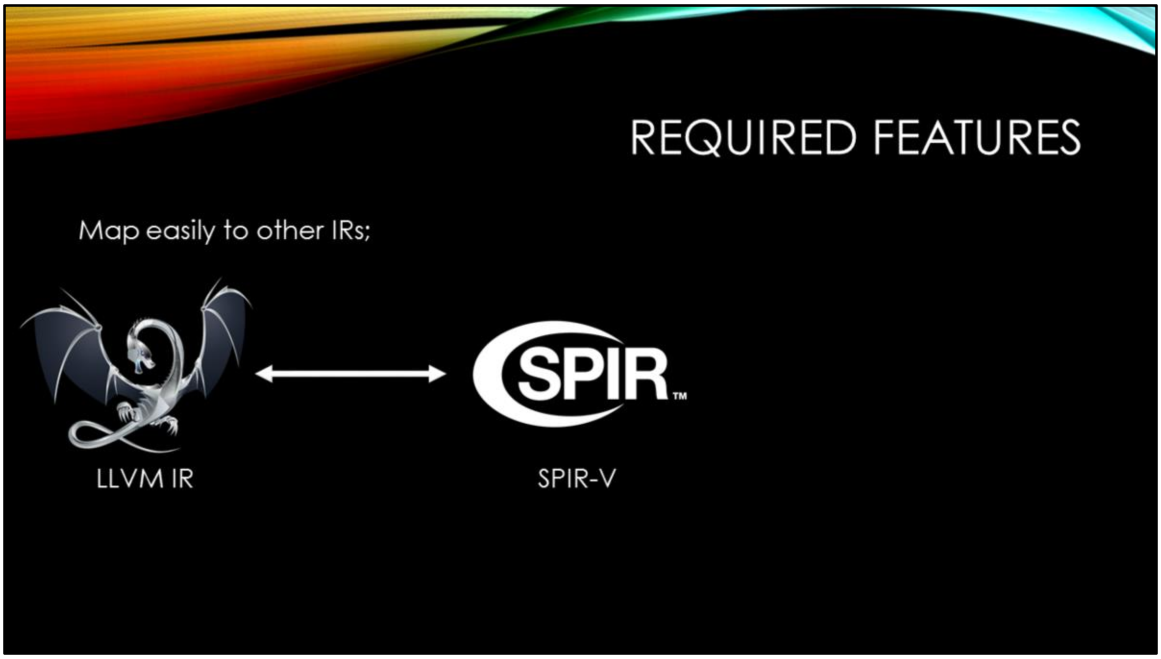
- Cross vendor
- Cross API
- Support for graphics & compute

"The first cross-vendor IL with native support for graphics and parallel computation constructs."

– Neil Trevett, President of the Khronos Group

SPIR-V is the first cross vendor, cross API, intermediate language that supports both graphics and compute.

Think of all the possibilities this allows the community to innovate with!



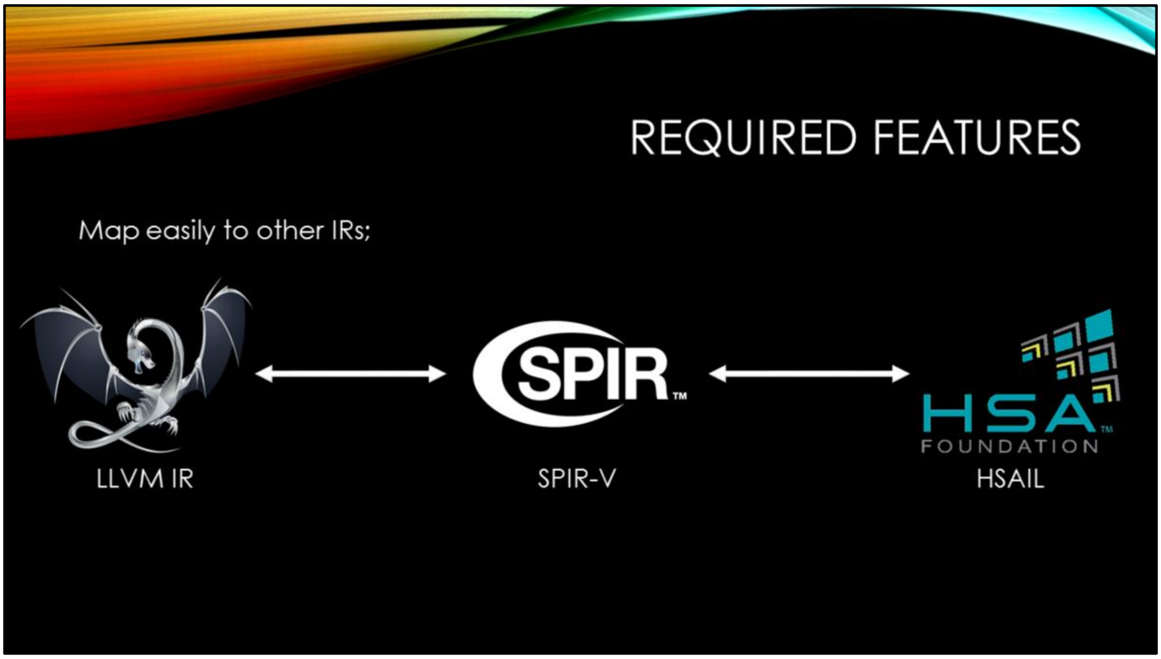
REQUIRED FEATURES

Map easily to other IRs;

LLVM IR

SPIR-V

LLVM LLVM LLVM LLVM LLVM! We didn’t want to use LLVM IR, but we damn well wanted to easily support transformations to and from the LLVM IR, and use all the awesome optimizations that LLVM has for our own benefit!

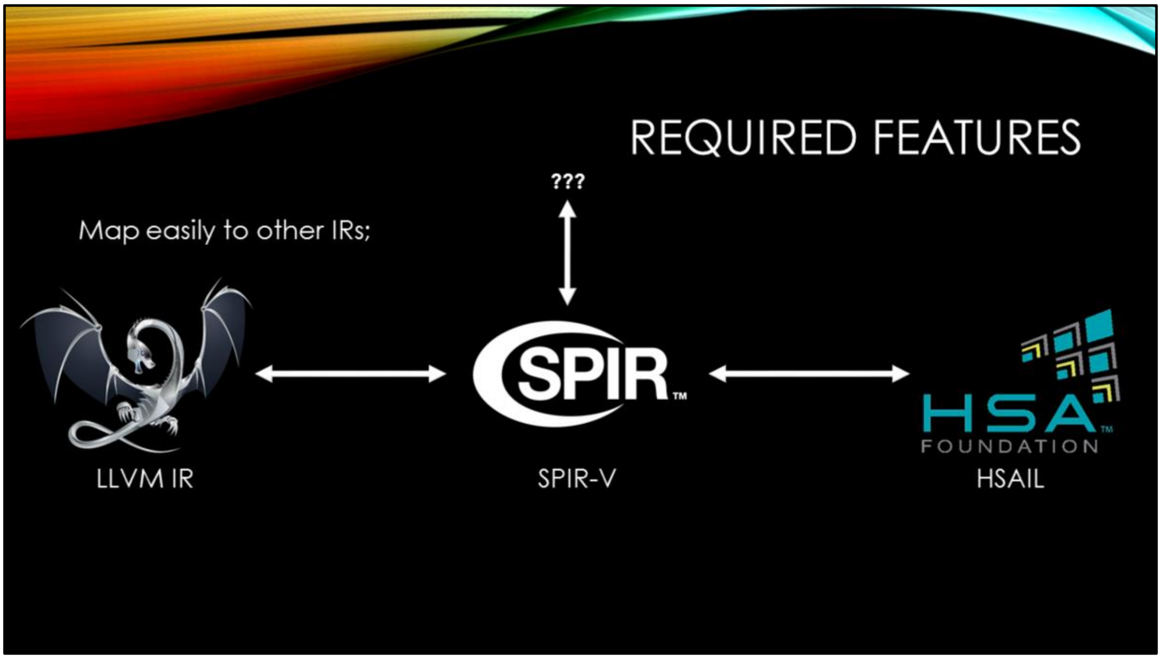


REQUIRED FEATURES

Map easily to other IRs;

LLVM IR ⟷ SPIR-V ⟷ HSAIL

We also wanted to be able to support other intermediates – why not the HSA foundations HSAIL?



REQUIRED FEATURES

Map easily to other IRs;

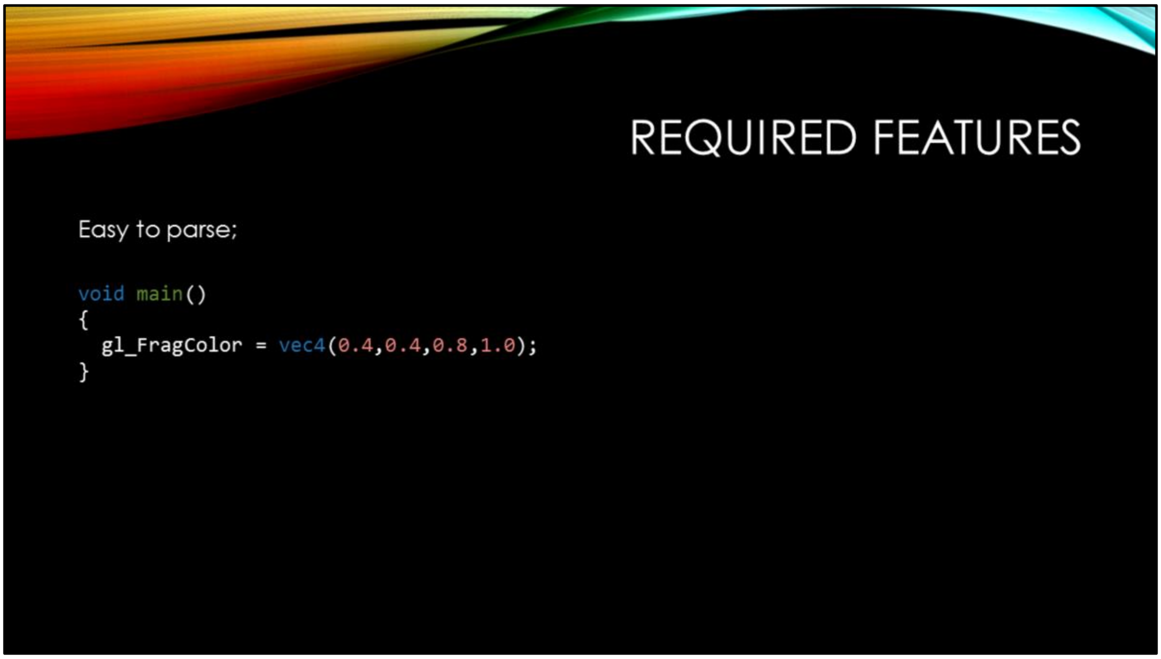
???

LLVM IR

SPIR-V

HSAIL

Or any IL/IR! The door is open to allow SPIR-V to become anything (we’ve tried really hard to wedge this door firmly open too)

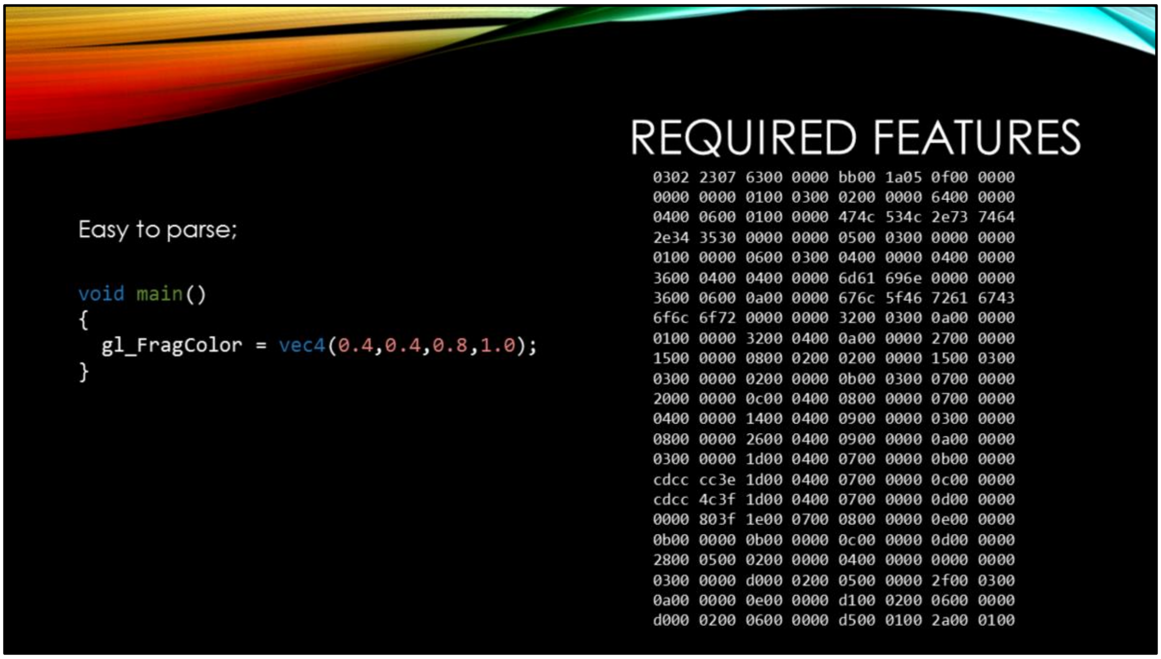


REQUIRED FEATURES

Easy to parse;

```
void main()
{
  gl_FragColor = vec4(0.4,0.4,0.8,1.0);
}
```

Here is a really simple fragment shader.



REQUIRED FEATURES

Easy to parse;

```
void main()
{
  gl_FragColor = vec4(0.4,0.4,0.8,1.0);
}
```

```
0302 2307 6300 0000 bb00 1a05 0f00 0000
0000 0000 0100 0300 0200 0000 6400 0000
0400 0600 0100 0000 474c 534c 2e73 7464
2e34 3530 0000 0000 0500 0300 0000 0000
0100 0000 0600 0300 0400 0000 0400 0000
3600 0400 0400 0000 6d61 696e 0000 0000
3600 0600 0a00 0000 676c 5f46 7261 6743
6f6c 6f72 0000 0000 3200 0300 0a00 0000
0100 0000 3200 0400 0a00 0000 2700 0000
1500 0000 0800 0200 0200 0000 1500 0300
0300 0000 0200 0000 0b00 0300 0700 0000
2000 0000 0c00 0400 0800 0000 0700 0000
0400 0000 1400 0400 0900 0000 0300 0000
0800 0000 2600 0400 0900 0000 0a00 0000
0300 0000 1d00 0400 0700 0000 0b00 0000
cdcc cc3e 1d00 0400 0700 0000 0c00 0000
cdcc 4c3f 1d00 0400 0700 0000 0d00 0000
0000 803f 1e00 0700 0800 0000 0e00 0000
0b00 0000 0b00 0000 0c00 0000 0d00 0000
2800 0500 0200 0000 0400 0000 0000 0000
0300 0000 d000 0200 0500 0000 2f00 0300
0a00 0000 0e00 0000 d100 0200 0600 0000
d000 0200 0600 0000 d500 0100 2a00 0100
```

Here is the SPIR-V generated from the Khronos GLSL -> SPIR-V tool

REQUIRED FEATURES

```
OpSource GLSL 100
OpExtInstImport %1 "GLSL.std.450"
OpMemoryModel Logical GLSL450
OpEntryPoint Fragment $4
OpName $4 "main"
OpName $10 "gl_FragColor"
OpDecorate $10 PrecisionMedium
OpDecorate $10 Built-In
OpTypeVoid %2
OpTypeFunction %3 $2
OpTypeFloat %7 32
OpTypeVector %8 $7 4
OpTypePointer %9 Output $8
OpVariable $9 %10 Output
OpConstant $7 %11 1053609165
OpConstant $7 %12 1061997773
OpConstant $7 %13 1065353216
OpConstantComposite $8 %14 $11 $11 $12 $13
OpFunction $2 %4 NoControl $3
OpLabel %5
OpStore $10 $14
OpBranch $6
OpLabel %6
OpReturn
OpFunctionEnd
```

```
0302 2307 6300 0000 bb00 1a05 0f00 0000
0000 0000 0100 0300 0200 0000 6400 0000
0400 0600 0100 0000 474c 534c 2e73 7464
2e34 3530 0000 0000 0500 0300 0000 0000
0100 0000 0600 0300 0400 0000 0400 0000
3600 0400 0400 0000 6d61 696e 0000 0000
3600 0600 0a00 0000 676c 5f46 7261 6743
6f6c 6f72 0000 0000 3200 0300 0a00 0000
0100 0000 3200 0400 0a00 0000 2700 0000
1500 0000 0800 0200 0200 0000 1500 0300
0300 0000 0200 0000 0b00 0300 0700 0000
2000 0000 0c00 0400 0800 0000 0700 0000
0400 0000 1400 0400 0900 0000 0300 0000
0800 0000 2600 0400 0900 0000 0a00 0000
0300 0000 1d00 0400 0700 0000 0b00 0000
cdcc cc3e 1d00 0400 0700 0000 0c00 0000
cdcc 4c3f 1d00 0400 0700 0000 0d00 0000
0000 803f 1e00 0700 0800 0000 0e00 0000
0b00 0000 0b00 0000 0c00 0000 0d00 0000
2800 0500 0200 0000 0400 0000 0000 0000
0300 0000 d000 0200 0500 0000 2f00 0300
0a00 0000 0e00 0000 d100 0200 0600 0000
d000 0200 0600 0000 d500 0100 2a00 0100
```

Here is the output from Codeplay's SPIR-V disassembler.

The tool took my team about 2 days to have initial support, and from then on we are just adding features (coloured command line output, validation, statistics, queries, etc.)

REQUIRED FEATURES

Unknown ops don't have to kill parsers!

```
0302 2307 6300 0000 bb00 1a05 0f00 0000
0000 0000 0100 0300 0200 0000 6400 0000
0400 0600 0100 0000 474c 534c 2e73 7464
2e34 3530 0000 0000 0500 0300 0000 0000
0100 0000 0600 0300 0400 0000 0400 0000
3600 0400 0400 0000 6d61 696e 0000 0000
3600 0600 0a00 0000 676c 5f46 7261 6743
6f6c 6f72 0000 0000 3200 0300 0a00 0000
0100 0000 3200 0400 0a00 0000 2700 0000
1500 0000 0800 0200 0200 0000 1500 0300
0300 0000 0200 0000 0b00 0300 0700 0000
2000 0000 0c00 0400 0800 0000 0700 0000
0400 0000 1400 0400 0900 0000 0300 0000
0800 0000 2600 0400 0900 0000 0a00 0000
0300 0000 1d00 0400 0700 0000 0b00 0000
cdcc cc3e 1d00 0400 0700 0000 0c00 0000
cdcc 4c3f 1d00 0400 0700 0000 0d00 0000
0000 803f 1e00 0700 0800 0000 0e00 0000
0b00 0000 0b00 0000 0c00 0000 0d00 0000
2800 0500 0200 0000 0400 0000 0000 0000
0300 0000 d000 0200 0500 0000 2f00 0300
0a00 0000 0e00 0000 cdab 0100 d100 0200
0600 0000 d000 0200 0600 0000 d500 0100
2a00 0100
```

Big feature is that if you don't understand or care about an opcode, you can skip it. Each opcode has a word count imbued within, so you know the lengths of words you need to skip even if you don't understand what you are looking at.

REQUIRED FEATURES

Unknown ops don’t have to kill parsers!

```
0302 2307 6300 0000 bb00 1a05 0f00 0000
0000 0000 0100 0300 0200 0000 6400 0000
0400 0600 0100 0000 474c 534c 2e73 7464
2e34 3530 0000 0000 0500 0300 0000 0000
0100 0000 0600 0300 0400 0000 0400 0000
3600 0400 0400 0000 6d61 696e 0000 0000
3600 0600 0a00 0000 676c 5f46 7261 6743
6f6c 6f72 0000 0000 3200 0300 0a00 0000
0100 0000 3200 0400 0a00 0000 2700 0000
1500 0000 0800 0200 0200 0000 1500 0300
0300 0000 0200 0000 0b00 0300 0700 0000
2000 0000 0c00 0400 0800 0000 0700 0000
0400 0000 1400 0400 0900 0000 0300 0000
0800 0000 2600 0400 0900 0000 0a00 0000
0300 0000 1d00 0400 0700 0000 0b00 0000
cdcc cc3e 1d00 0400 0700 0000 0c00 0000
cdcc 4c3f 1d00 0400 0700 0000 0d00 0000
0000 803f 1e00 0700 0800 0000 0e00 0000
0b00 0000 0b00 0000 0c00 0000 0d00 0000
2800 0500 0200 0000 0400 0000 0000 0000
0300 0000 d000 0200 0500 0000 2f00 0300
0a00 0000 0e00 0000 cdab 0100 d100 0200
0600 0000 d000 0200 0600 0000 d500 0100
2a00 0100
```

I’ve injected this dud opcode 0xabcd, word count 1

REQUIRED FEATURES

```
OpSource GLSL 100
OpExtInstImport %1 "GLSL.std.450"
OpMemoryModel Logical GLSL450
OpEntryPoint Fragment $4
OpName $4 "main"
OpName $10 "gl_FragColor"
OpDecorate $10 PrecisionMedium
OpDecorate $10 Built-In
OpTypeVoid %2
OpTypeFunction %3 $2
OpTypeFloat %7 32
OpTypeVector %8 $7 4
OpTypePointer %9 Output $8
OpVariable $9 %10 Output
OpConstant $7 %11 1053609165
OpConstant $7 %12 1061997773
OpConstant $7 %13 1065353216
OpConstantComposite $8 %14 $11 $11 $12 $13
OpFunction $2 %4 NoControl $3
OpLabel %5
OpStore $10 $14
OpUnknown<43981, count 1>
OpBranch $6
OpLabel %6
OpReturn
OpFunctionEnd
```

```
0302 2307 6300 0000 bb00 1a05 0f00 0000
0000 0000 0100 0300 0200 0000 6400 0000
0400 0600 0100 0000 474c 534c 2e73 7464
2e34 3530 0000 0000 0500 0300 0000 0000
0100 0000 0600 0300 0400 0000 0400 0000
3600 0400 0400 0000 6d61 696e 0000 0000
3600 0600 0a00 0000 676c 5f46 7261 6743
6f6c 6f72 0000 0000 3200 0300 0a00 0000
0100 0000 3200 0400 0a00 0000 2700 0000
1500 0000 0800 0200 0200 0000 1500 0300
0300 0000 0200 0000 0b00 0300 0700 0000
2000 0000 0c00 0400 0800 0000 0700 0000
0400 0000 1400 0400 0900 0000 0300 0000
0800 0000 2600 0400 0900 0000 0a00 0000
0300 0000 1d00 0400 0700 0000 0b00 0000
cdcc cc3e 1d00 0400 0700 0000 0c00 0000
cdcc 4c3f 1d00 0400 0700 0000 0d00 0000
0000 803f 1e00 0700 0800 0000 0e00 0000
0b00 0000 0b00 0000 0c00 0000 0d00 0000
2800 0500 0200 0000 0400 0000 0000 0000
0300 0000 d000 0200 0500 0000 2f00 0300
0a00 0000 0e00 0000 cdab 0100 d100 0200
0600 0000 d000 0200 0600 0000 d500 0100
2a00 0100
```

Our parser finds the opcode, hasn't a scooby what it is, so just outputs an OpUnknown. Notice that the opcodes after that unknown opcode are still parse-able because of the word count!

REQUIRED FEATURES

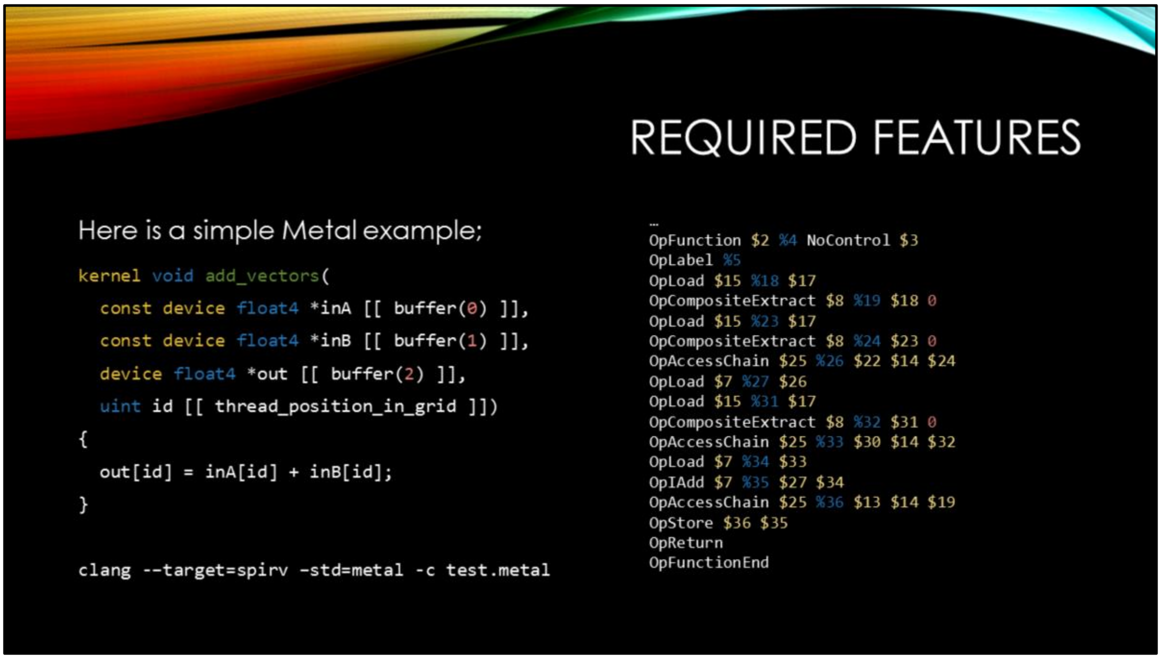
- Support Vulkan & OpenCL
- Support GLSL & OpenCL C/C++
- But not be bound to them
- Other APIs/languages as first class citizens;
  - DirectX?
  - WebGL?
  - OpenGL?
  - OpenGL ES?
  - Metal Shading Language?
  - HLSL?
  - PSSL?
  - C/C++?

We need to support Vulkan and OpenCL (they are using SPIR-V after all).

But we don’t want to be stuck to them! SPIR-V has been designed to let others use it if they wish.

SPIR-V is easily extended, so we hope that others will see the benefit of the technology and come on-board.

We even have a separate group entirely now within Khronos focused solely on SPIR-V, rather than being part of the Vulkan or OpenCL umbrellas.



REQUIRED FEATURES

Here is a simple Metal example;

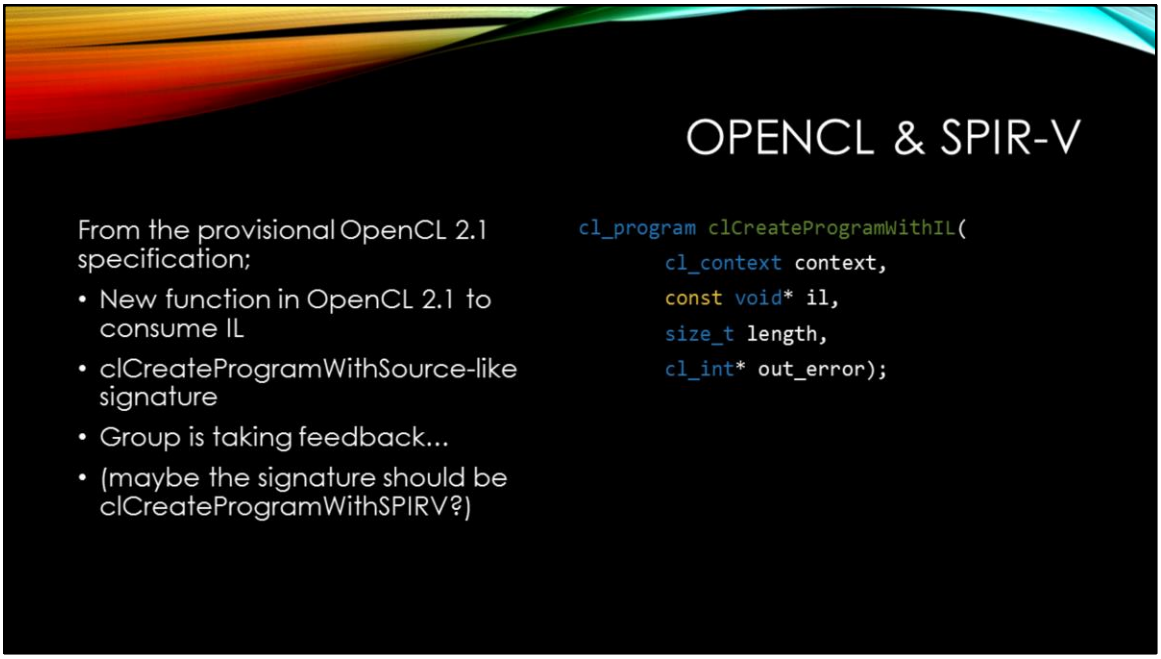
```
kernel void add_vectors(
  const device float4 *inA [[ buffer(0) ]],
  const device float4 *inB [[ buffer(1) ]],
  device float4 *out [[ buffer(2) ]],
  uint id [[ thread_position_in_grid ]])
{
  out[id] = inA[id] + inB[id];
}
```

```
clang --target=spirv -std=metal -c test.metal
```

```
…
OpFunction $2 %4 NoControl $3
OpLabel %5
OpLoad $15 %18 $17
OpCompositeExtract $8 %19 $18 0
OpLoad $15 %23 $17
OpCompositeExtract $8 %24 $23 0
OpAccessChain $25 %26 $22 $14 $24
OpLoad $7 %27 $26
OpLoad $15 %31 $17
OpCompositeExtract $8 %32 $31 0
OpAccessChain $25 %33 $30 $14 $32
OpLoad $7 %34 $33
OpIAdd $7 %35 $27 $34
OpAccessChain $25 %36 $13 $14 $19
OpStore $36 $35
OpReturn
OpFunctionEnd
```

For example, I’ve hacked clang + LLVM to add a SPIR-V target, and a metal C++ standard. I then feed in this simple metal kernel that adds two vectors together, and I get valid SPIR-V on the right!

This initial support took about 3 days to get in (although I can’t say the code is by any stretch pretty or maintainable) but it just shows you how easy it is to support!



OPENCL & SPIR-V

From the provisional OpenCL 2.1 specification;

- New function in OpenCL 2.1 to consume IL
- clCreateProgramWithSource-like signature
- Group is taking feedback…
- (maybe the signature should be clCreateProgramWithSPIRV?)

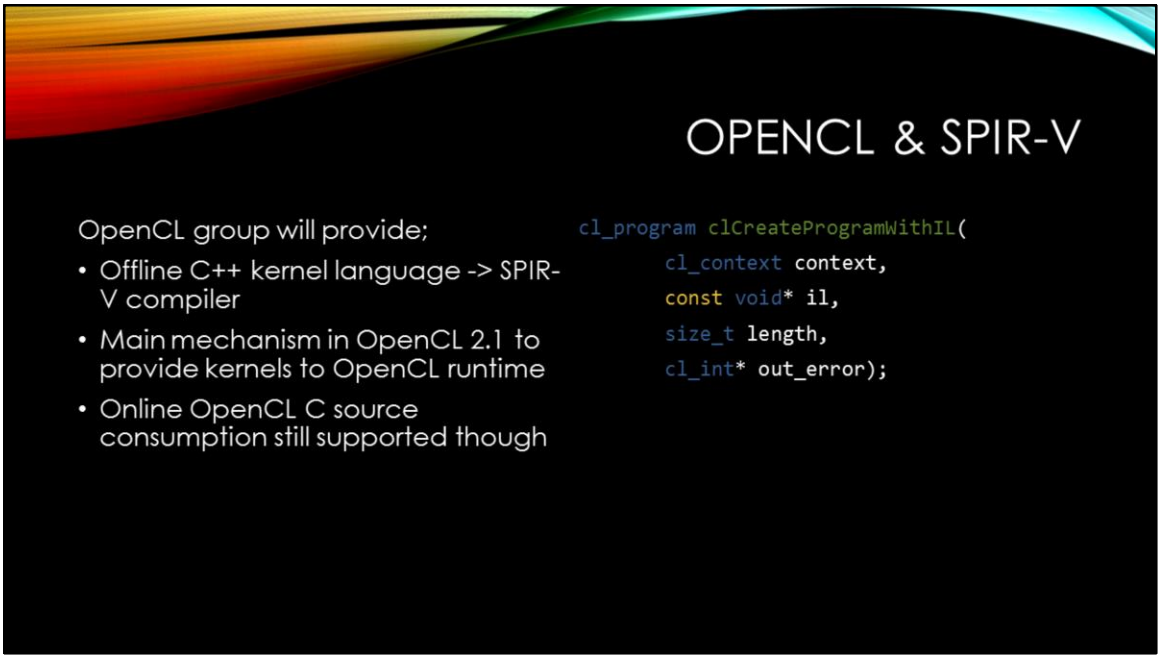
```
cl_program clCreateProgramWithIL(
        cl_context context,
        const void* il,
        size_t length,
        cl_int* out_error);
```

OpenCL 2.1 has CORE support for SPIR-V – that means every implementation of OpenCL 2.1 will support SPIR-V.

There is one entry point for SPIR-V binaries – clCreateProgramWithIL.

This is great news for OpenCL developers (once the drivers hit the market of course).

The group is taking feedback though, this is the time to request things!
- Want vendors on older OpenCL to support a SPIR-V extension? Request it!



OPENCL & SPIR-V

OpenCL group will provide;

- Offline C++ kernel language -> SPIR-V compiler
- Main mechanism in OpenCL 2.1 to provide kernels to OpenCL runtime
- Online OpenCL C source consumption still supported though

```
cl_program clCreateProgramWithIL(
        cl_context context,
        const void* il,
        size_t length,
        cl_int* out_error);
```

The Khronos OpenCL group are going to provide OpenCL C++ -> SPIR-V compiler, which will be the main mechanism for running OpenCL C++ kernels on a compute device.

VULKAN & SPIR-V

Vulkan consumes SPIR-V for graphics & compute with single interface;

- Shader create info is where you give Vulkan your SPIR-V binary
- Other additional information can be provided here too

Vulkan group will provide;

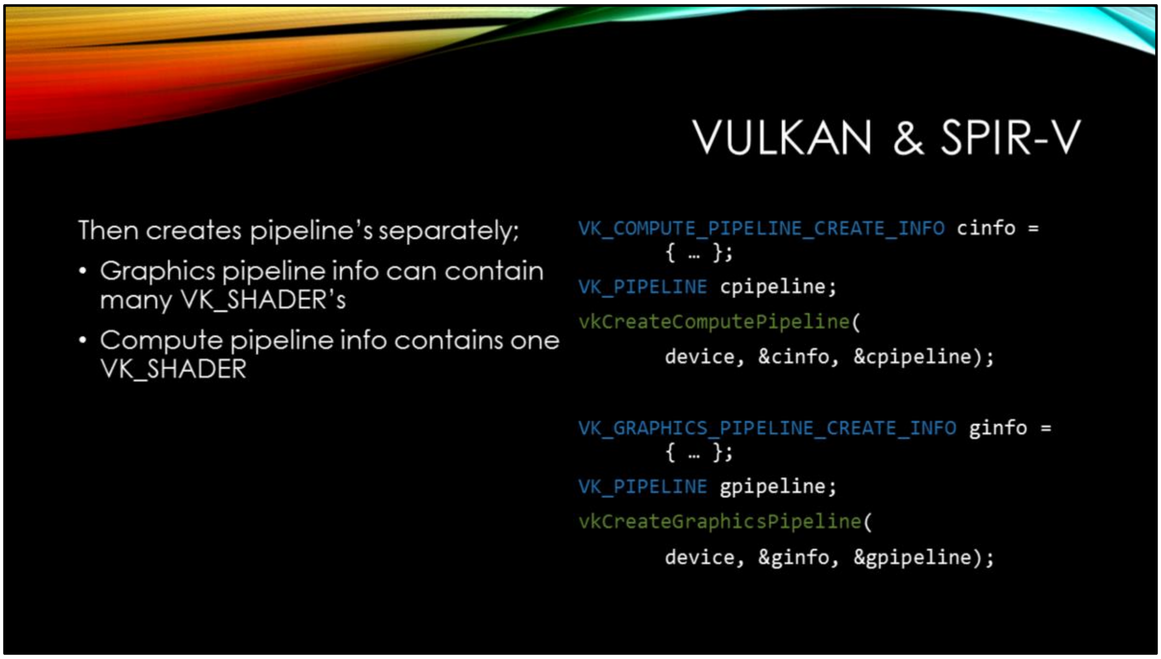
- Offline GLSL -> SPIR-V compiler (available now!)

```
VK_SHADER_CREATE_INFO info = { … };
VK_SHADER shader;
vkCreateShader(device, &info, &shader);
```

I know more about the Vulkan side – it is what I work on after all!

SPIR-V is consumed by Vulkan at the vkCreateShader entry point. This is the same for both compute and graphics shaders.

You can find out more about the GLSL -> SPIR-V support here https://www.khronos.org/opengles/sdk/tools/Reference-Compiler/



VULKAN & SPIR-V

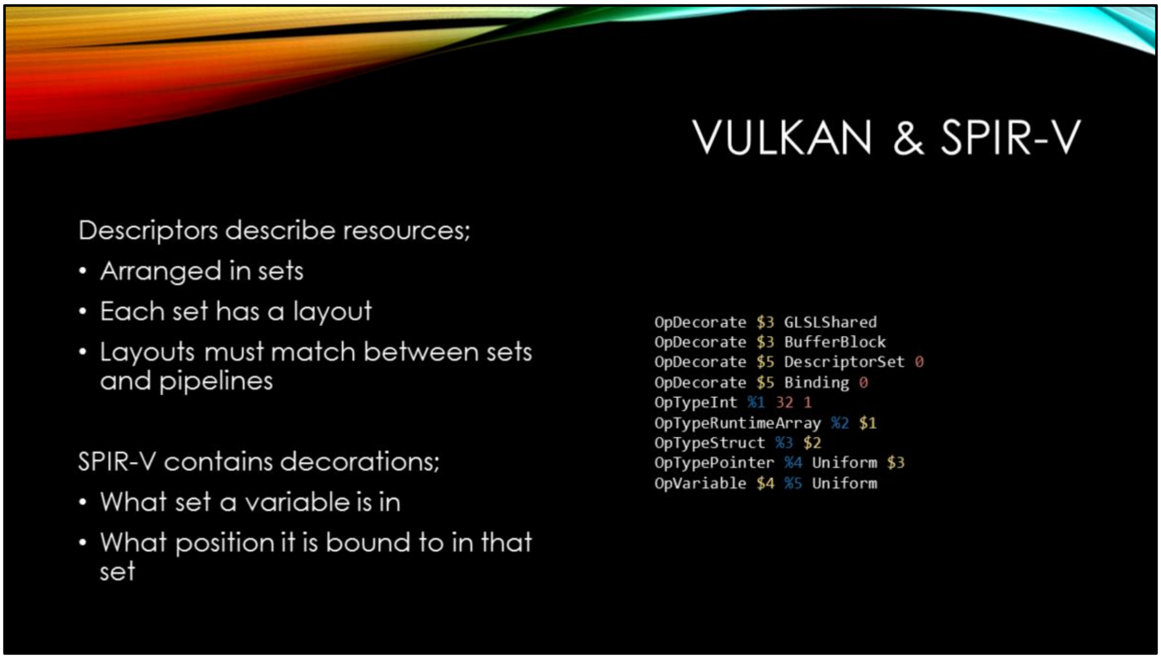
Then creates pipeline's separately;

- Graphics pipeline info can contain many VK_SHADER's
- Compute pipeline info contains one VK_SHADER

```
VK_COMPUTE_PIPELINE_CREATE_INFO cinfo =
        { … };
VK_PIPELINE cpipeline;
vkCreateComputePipeline(
        device, &cinfo, &cpipeline);

VK_GRAPHICS_PIPELINE_CREATE_INFO ginfo =
        { … };
VK_PIPELINE gpipeline;
vkCreateGraphicsPipeline(
        device, &ginfo, &gpipeline);
```

Pipelines are created for graphics and compute separately though. Graphics pipelines can reference multiple shaders (think vertex then fragment shaders in the pipeline) whereas compute pipelines can reference one shader.



VULKAN & SPIR-V

Descriptors describe resources;

- Arranged in sets
- Each set has a layout
- Layouts must match between sets and pipelines

SPIR-V contains decorations;

- What set a variable is in
- What position it is bound to in that set

```
OpDecorate $3 GLSLShared
OpDecorate $3 BufferBlock
OpDecorate $5 DescriptorSet 0
OpDecorate $5 Binding 0
OpTypeInt %1 32 1
OpTypeRuntimeArray %2 $1
OpTypeStruct %3 $2
OpTypePointer %4 Uniform $3
OpVariable $4 %5 Uniform
```

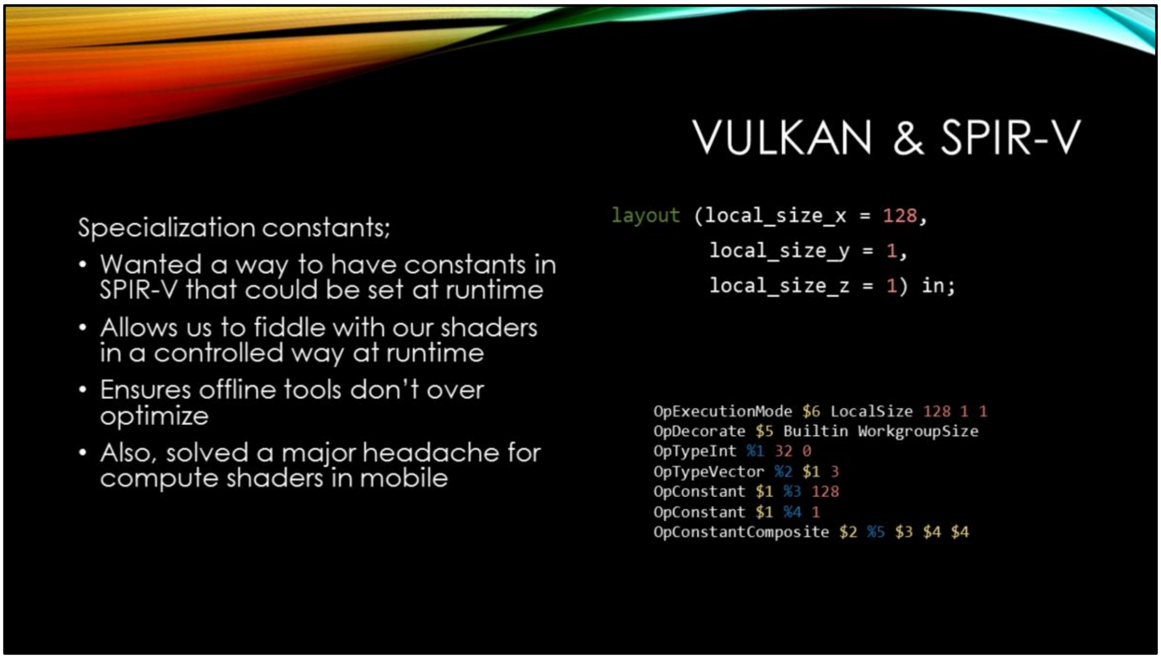
Core part of Vulkan are descriptors and descriptor sets. Graham Sellers already covered these in more detail here https://www.khronos.org/assets/uploads/developers/library/2015-gdc/Khronos-Vulkan-GDC_Mar15.pdf

How SPIR-V uses descriptor sets is via a pair of decorations, DescriptorSet and Binding.

DescriptorSet matches with the descriptor set in the Vulkan API, and Binding is the position within that DescriptorSet that a resource is.

In this example we have a compute shader, and we are bringing in a buffer. This buffer is in the 0th set, and is bound to the 0th position.

Descriptor sets are ***awesome*** because they let you switch out some resources but leave others untouched. Very powerful concept.



VULKAN & SPIR-V

Specialization constants;

- Wanted a way to have constants in SPIR-V that could be set at runtime
- Allows us to fiddle with our shaders in a controlled way at runtime
- Ensures offline tools don't over optimize
- Also, solved a major headache for compute shaders in mobile

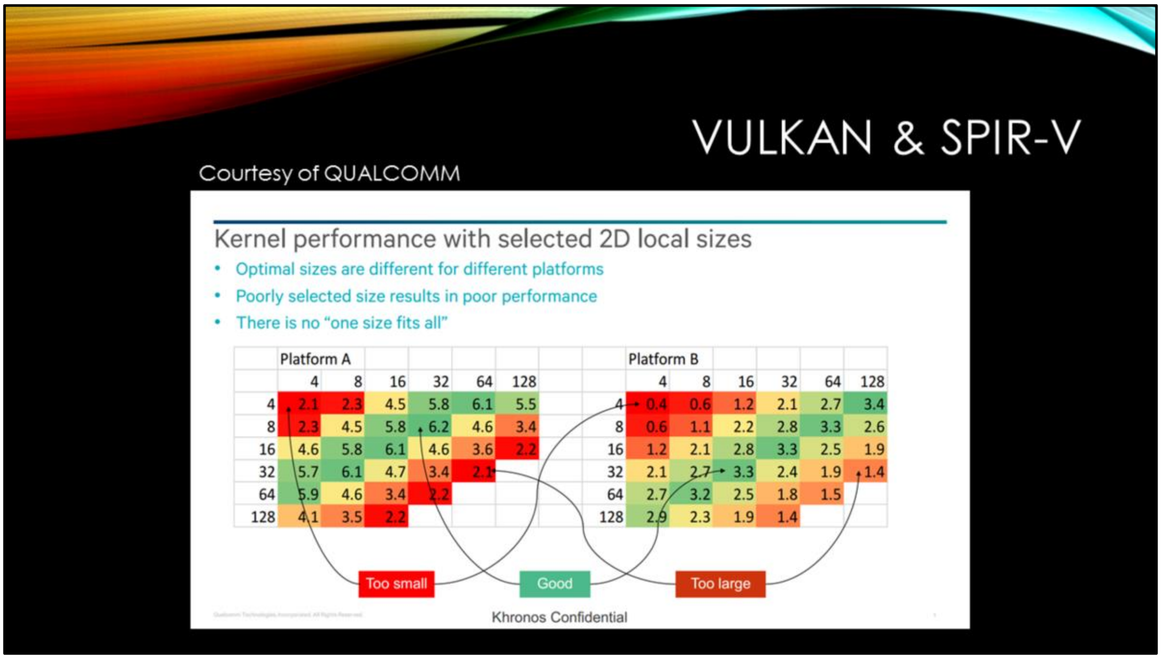
```
layout (local_size_x = 128,
        local_size_y = 1,
        local_size_z = 1) in;
```

```
OpExecutionMode $6 LocalSize 128 1 1
OpDecorate $5 Builtin WorkgroupSize
OpTypeInt %1 32 0
OpTypeVector %2 $1 3
OpConstant $1 %3 128
OpConstant $1 %4 1
OpConstantComposite $2 %5 $3 $4 $4
```

Going to cover a feature another cool feature we added – specialization constants.

Specialization constants are a way to solve *'At runtime, some values might need to be different'*. A good example of this might be in the mobile space, you have a low end GPU that is incapable of some computation. You could use multiple SPIR-V shaders to solve this, or you could use a specialization constant within the SPIR-V that you modify at runtime.

I'm showing here the code for workgroup size setting from a GLSL Compute shader -> SPIR-V. This is how a GLSL compute shader today would look.



VULKAN & SPIR-V

Courtesy of QUALCOMM

Kernel performance with selected 2D local sizes

- Optimal sizes are different for different platforms
- Poorly selected size results in poor performance
- There is no "one size fits all"

Platform A	4	8	16	32	64	128
4	2.1	2.3	4.5	5.8	6.1	5.5
8	2.3	4.5	5.8	6.2	4.6	3.4
16	4.6	5.8	6.1	4.6	3.6	2.2
32	5.7	6.1	4.7	3.4	2.1	
64	5.9	4.6	3.4	2.2		
128	4.1	3.5	2.2			

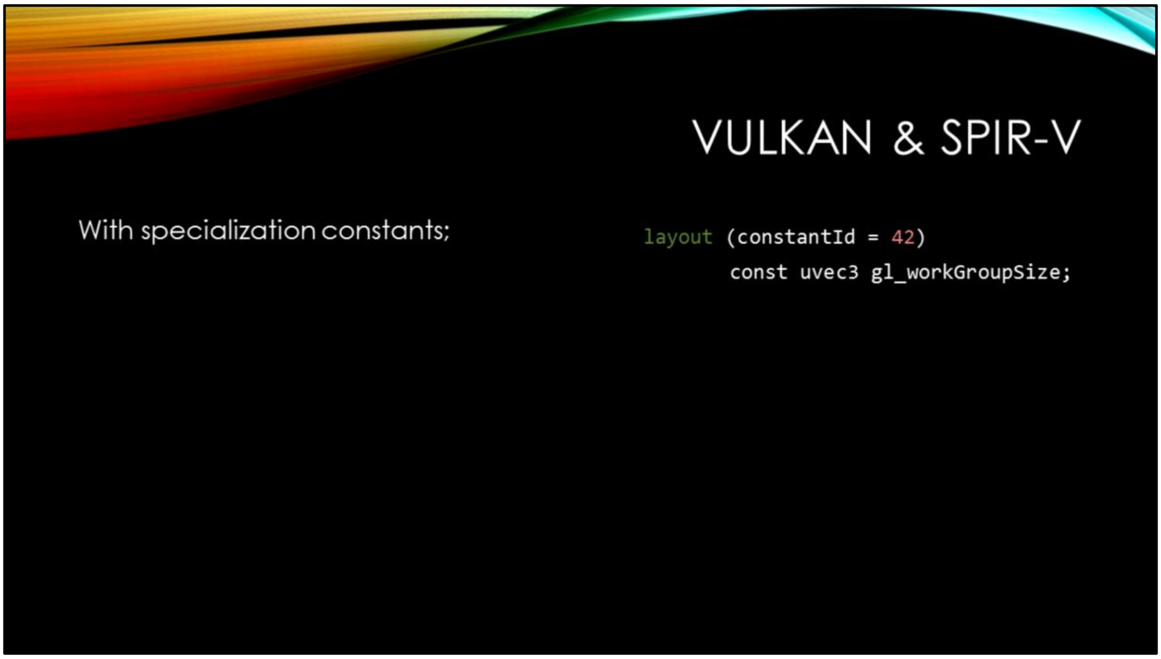
Platform B	4	8	16	32	64	128
4	0.4	0.6	1.2	2.1	2.7	3.4
8	0.6	1.1	2.2	2.8	3.3	2.6
16	1.2	2.1	2.8	3.3	2.5	1.9
32	2.1	2.7	3.3	2.4	1.9	1.4
64	2.7	3.2	2.5	1.8	1.5	
128	2.9	2.3	1.9	1.4		

Khronos Confidential

Thanks to Qualcomm for allowing me to use this slide;

This slide shows performance of a compute shader when varying the work group size in the x/y dimensions on two different mobile GPUs. The crux of the matter is – one work group size set at compile time doesn't cut it for mobile, we needed a way to vary the work group size.

Specialization constants can solve this problem for us too!

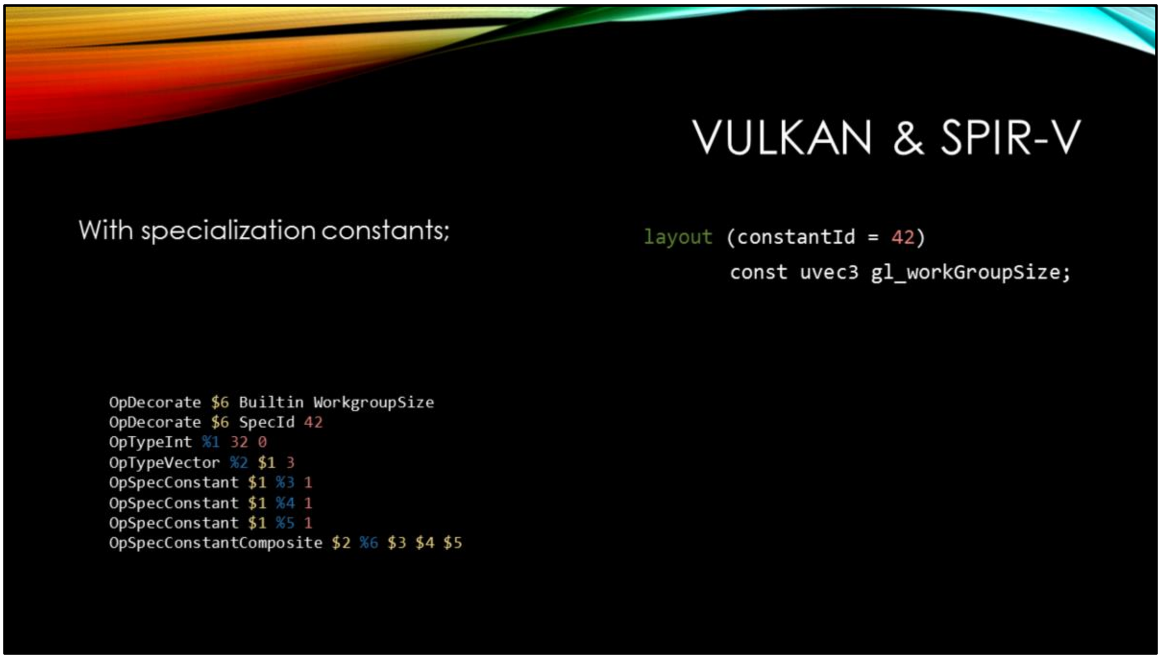


VULKAN & SPIR-V

With specialization constants;

```
layout (constantId = 42)
      const uvec3 gl_workGroupSize;
```

We are using a new mechanism in this GLSL – having a constantId layout specifier.
The value of this constantId can be anything (user chosen) and is used within Vulkan.



VULKAN & SPIR-V

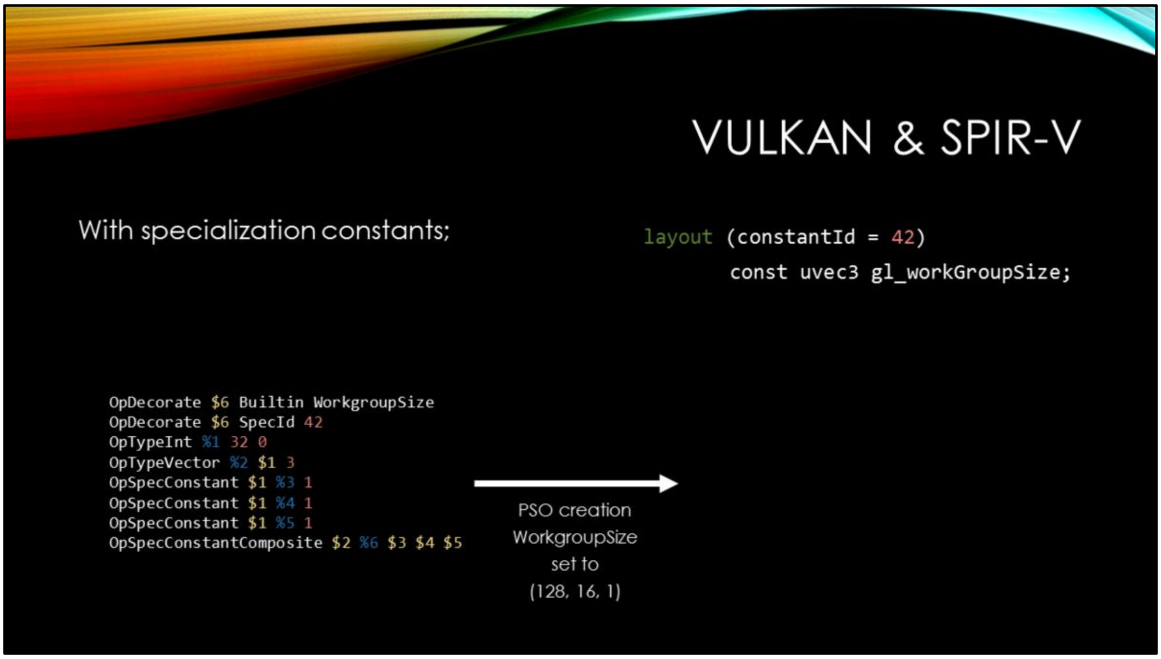
With specialization constants;

```
layout (constantId = 42)
      const uvec3 gl_workGroupSize;
```

```
OpDecorate $6 Builtin WorkgroupSize
OpDecorate $6 SpecId 42
OpTypeInt %1 32 0
OpTypeVector %2 $1 3
OpSpecConstant $1 %3 1
OpSpecConstant $1 %4 1
OpSpecConstant $1 %5 1
OpSpecConstantComposite $2 %6 $3 $4 $5
```

Here is the SPIR-V produced, you can the new decoration SpecId has appeared, and is set to 42. This of course is mapped to the constantId used in the shader.

We also now have OpSpec* constants, all defaulted to 1 (NOTE: All specialization constants ***have*** to have a default value in the SPIR-V, which covers the case that a user did not set them from Vulkan).



VULKAN & SPIR-V

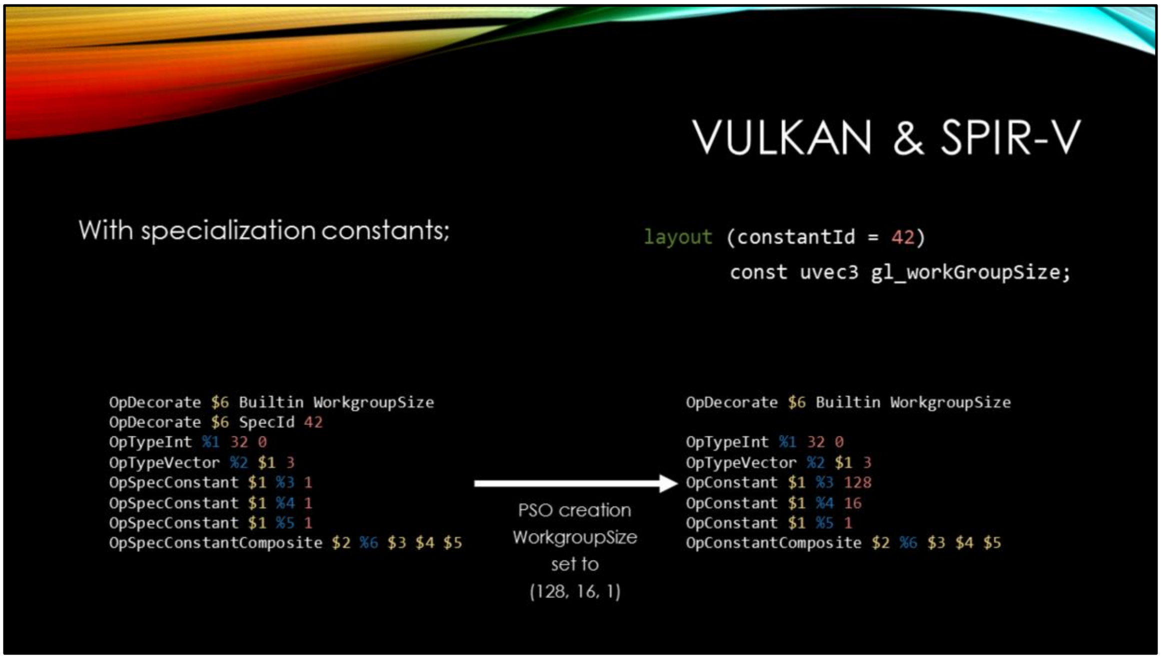
With specialization constants;

```
layout (constantId = 42)
      const uvec3 gl_workGroupSize;
```

```
OpDecorate $6 Builtin WorkgroupSize
OpDecorate $6 SpecId 42
OpTypeInt %1 32 0
OpTypeVector %2 $1 3
OpSpecConstant $1 %3 1
OpSpecConstant $1 %4 1
OpSpecConstant $1 %5 1
OpSpecConstantComposite $2 %6 $3 $4 $5
```

PSO creation
WorkgroupSize
set to
(128, 16, 1)

At pipeline creation time, we’ve set the workgroup size to 128,16,1.



VULKAN & SPIR-V

With specialization constants;

```
layout (constantId = 42)
    const uvec3 gl_workGroupSize;
```

```
OpDecorate $6 Builtin WorkgroupSize
OpDecorate $6 SpecId 42
OpTypeInt %1 32 0
OpTypeVector %2 $1 3
OpSpecConstant $1 %3 1
OpSpecConstant $1 %4 1
OpSpecConstant $1 %5 1
OpSpecConstantComposite $2 %6 $3 $4 $5
```

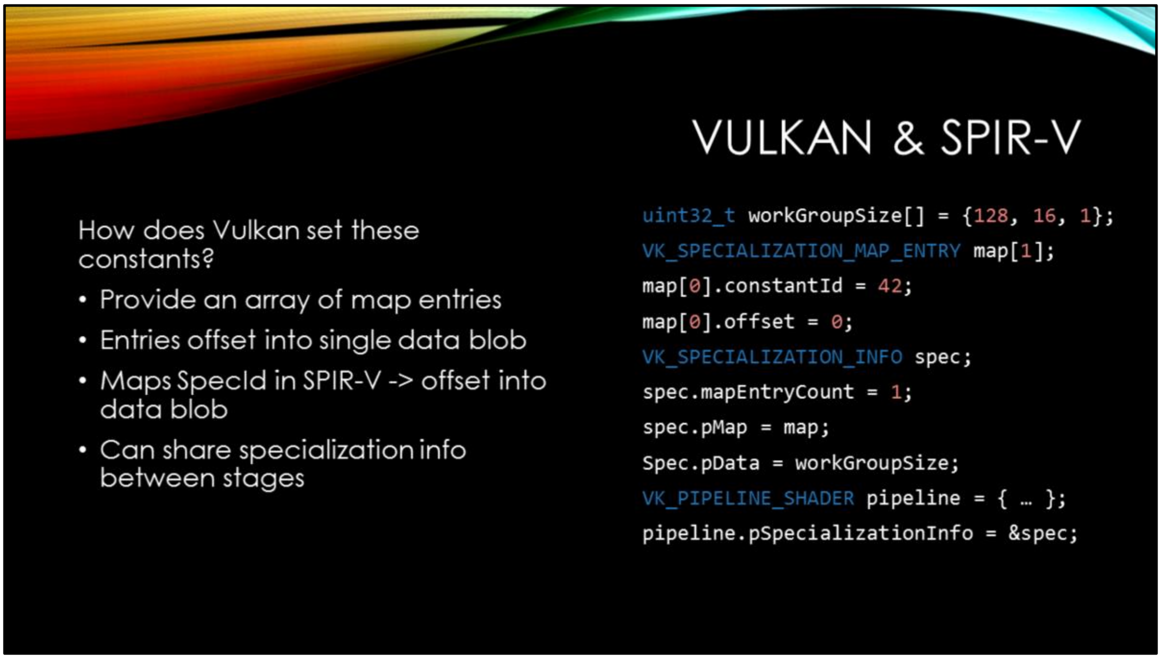
PSO creation
WorkgroupSize
set to
(128, 16, 1)

```
OpDecorate $6 Builtin WorkgroupSize

OpTypeInt %1 32 0
OpTypeVector %2 $1 3
OpConstant $1 %3 128
OpConstant $1 %4 16
OpConstant $1 %5 1
OpConstantComposite $2 %6 $3 $4 $5
```

This is what the SPIR-V would look like after the specialization has taken place. This happens within the Vulkan driver so you wouldn't actually see this in Vulkan.

An interesting side note on this – a user could apply a specialization offline if they wanted by mimicking this approach. OpSpecConstant's -> OpConstant's, OpSpecConstantComposite's -> OpConstantComposite's, and remove the SpecId decoration, done!



VULKAN & SPIR-V

How does Vulkan set these constants?

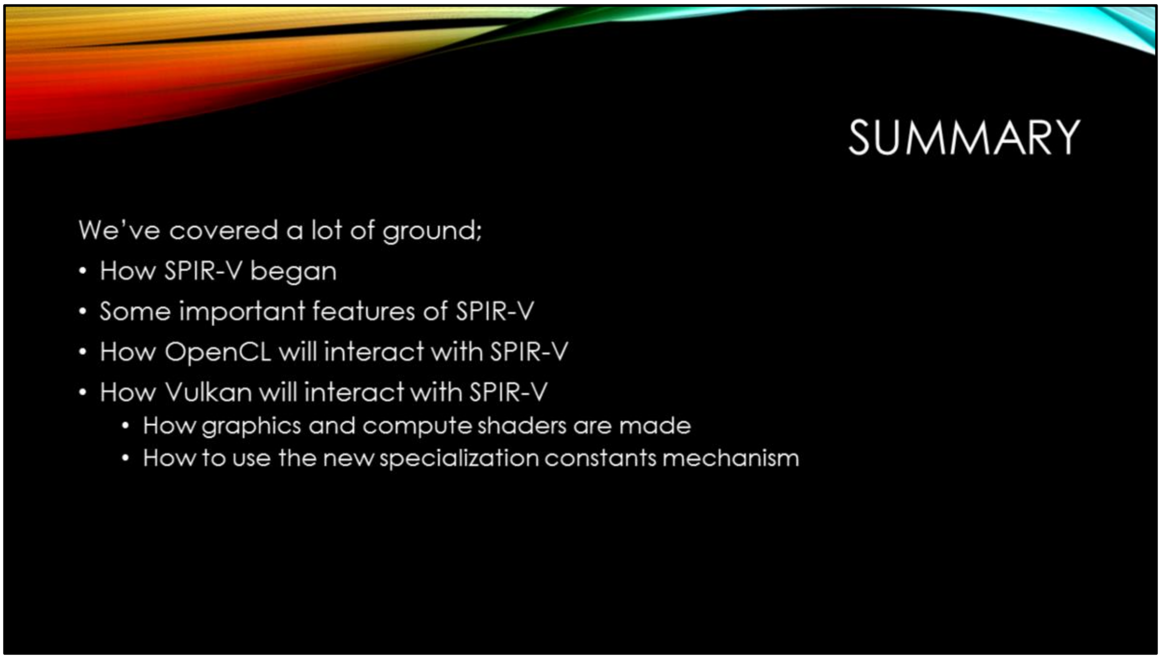
- Provide an array of map entries
- Entries offset into single data blob
- Maps SpecId in SPIR-V -> offset into data blob
- Can share specialization info between stages

```
uint32_t workGroupSize[] = {128, 16, 1};
VK_SPECIALIZATION_MAP_ENTRY map[1];
map[0].constantId = 42;
map[0].offset = 0;
VK_SPECIALIZATION_INFO spec;
spec.mapEntryCount = 1;
spec.pMap = map;
Spec.pData = workGroupSize;
VK_PIPELINE_SHADER pipeline = { … };
pipeline.pSpecializationInfo = &spec;
```

Lastly, how does Vulkan set these values?

We use a single buffer, and an array of mappings into this buffer. These map entries map the SpecId value (42 in our case) to an offset into the single buffer of data.

The specialization info is a pointer in the VK_PIPELINE_SHADER struct, and thus can be shared between multiple stages (EG. you might have some switch in both the vertex and fragment shader you want to fiddle with).



SUMMARY

We've covered a lot of ground;

- How SPIR-V began
- Some important features of SPIR-V
- How OpenCL will interact with SPIR-V
- How Vulkan will interact with SPIR-V
  - How graphics and compute shaders are made
  - How to use the new specialization constants mechanism

Much info was given, I hope enough to satisfy in the short time I had to present.

THANKS!

Find me on Twitter;
@sheredom

Useful links;

- https://www.khronos.org/developers/library/2015-gdc
- http://www.gdcvault.com/play/1022018/
- https://www.khronos.org/registry/spir-v/papers/WhitePaper.pdf

Please do get in touch – SPIR-V is new and there is still plenty of room to mould it further if you think we have missed something.

https://www.khronos.org/bugzilla/ in the SPIR-V product is a great place to tell us where we could be better.