SPIR-V

BUILDING THE NEXT GENERATION OF GRAPHICS AND COMPUTE LANGUAGES

Benedict R. Gaster / @cuberoo_

UWE BRISTOL
University of the West of England

bettertogether

THE NEXT 700 PROGRAMMING LANGUAGES

"A family of unimplemented computing languages is described that is intended to span differences of application area by a unified framework." P. J. Landin, 1966

THE NEXT 700 SHADER LANGUAGES

WHY USE IT?

NEW SHADING LANGUAGES

- Modern C++ shader language, alternative to GLSL
- Graphics languages are (at core) functional languages
  - A lot has happened in functional language design
  - Modern functional lanugage foxsr graphics

DOMAIN SPECIFIC LANGUAGES (DSL)

- Image processing is not 3D graphics or GPGPU
  - So why use OpenCL C/C++ or GLSL?
  - Halide, from MIT, excellent example of DSL for image processing

WHATSIT 2D - A GAME ENGINE FOR UWE STUDENTS

- 2nd year undergraduate course on programming
- Whatsit 2D is a simple game engine
  - Used through out the course
- Written in Scala, abstracting LWJGL 3.x
  - OpenGL 3.2 (Core profile only)
  - OpenCL 1.2
  - OpenAL

DIFFERENT LANGUAGE FOR CORE FEATURES!

- Using OpenGL, OpenGL, and OpenAL directly means
  - 3 different APIs and 3 different ways of thinking

SINGLE BASE LANGUAGE

- Don't want to teach OpenCL and OpenGL
  - Simplify OpenCL to parallel_for
  - Simplify OpenGL to shaders, sprite sheets, world maps, cameras, and so on

SOLUTION

- Define 3 (simplified) internal DSLs within Scala
  - Graphics, automatically generate SPIR-V (today GLSL)
  - Compute, automatically generate SPIR-V (today OpenCL C)
  - Audio, automatically generate ??? (not started)

THE TRIVIAL VERTEX SHADER: GLSL

```
uniform mat4 projView;

layout(location = 0) in vec2 position;
layout(location = 1) in vec4 color;

varying vec4 vColor;

void main() {
  vColor = color;
  gl_Position = projView * vec4(position, 0.0, 1.0);
}
```

THE TRIVIAL VERTEX SHADER: WHATSIT2D

```
val vShader(args: VertexIn[Mat4f :: Vec2f :: Vec4f :: HNil]) :
  VertexOut[Vec4f :: Vec4f with AsBuiltin("position") :: HNil] = {
    val (projView, position, color) = args split
    color :: (projView :*: position)
}
```

SPIR-V

FUNCTIONAL REQUIREMENTS

Enable compiler ecosystem for portable parallel programs

FUNCTIONAL REQUIREMENTS

Retain details of high-level semantics

FUNCTIONAL REQUIREMENTS

Portable binary representation of graphic shader programs and compute kernels

FUNCTIONAL REQUIREMENTS

Developed and maintained by Khronos

FUNCTIONAL REQUIREMENTS

Target for OpenCL C/C++, GLSL, and other compute kernel languages

SPIR-V: A SOLUTION TO THE REQUIREMENTS

INTERMEDIATE LANGUAGE

An intermediate language for input to Khronos graphics and compute APIs

A STANDARD

Fully specified, Khronos defined standard

CONVERTABLE

Easily convertable to/from LLVM IR

SUPPORTED IN LATEST KHRONOS APIS

Is a core feature in Vulkan!

SUPPORTED IN LATEST KHRONOS APIS

Is a core optional feature in OpenCL 2.1!

SUPPORTED IN OPENCL 1.2 AND 2.0

As a (new) optional extension!

OPENCL C AND C++ SUPPORT

- OpenCL C 1.2 kernel language support
- OpenCL C 2.0 kernel language support
- OpenCL C++ 2.1 kernel language support

SPIR-V: SOME DETAILS

OPENCL C AND C++ SUPPORT

- Scoped based memory model
- Ndrange, work-group, and sub-group execution model
- Address spaces, including Generic
- Device side kernel enqueue

OPENCL C AND C++ SUPPORT

- C11 and C++11 style atomic operations:
  - Extended to support scopes
  - Extended to support address spaces
- and much more...

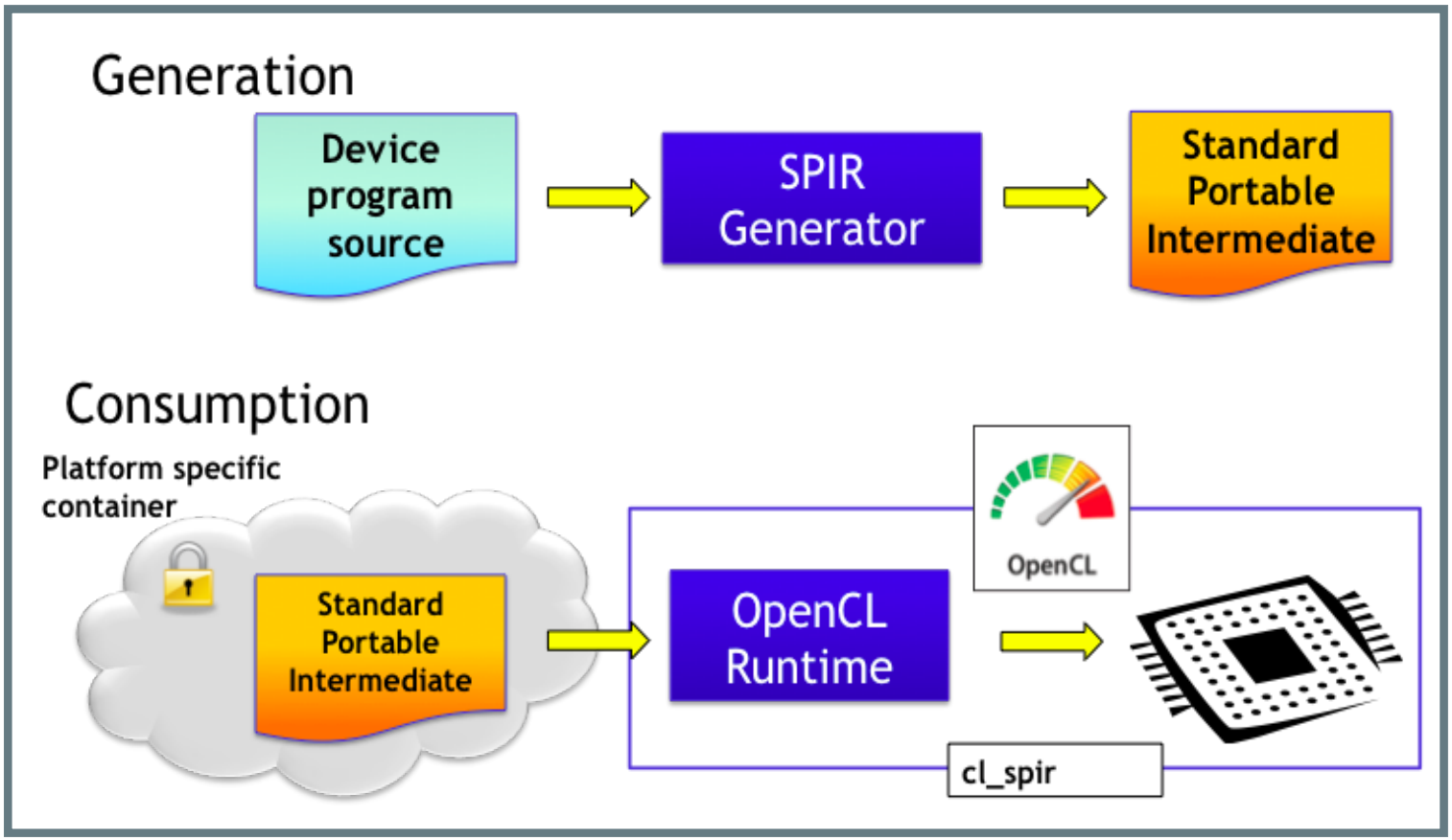
VULKAN GLSL

- Under development (i.e. not yet public)
- Graphics specific instructions and resources

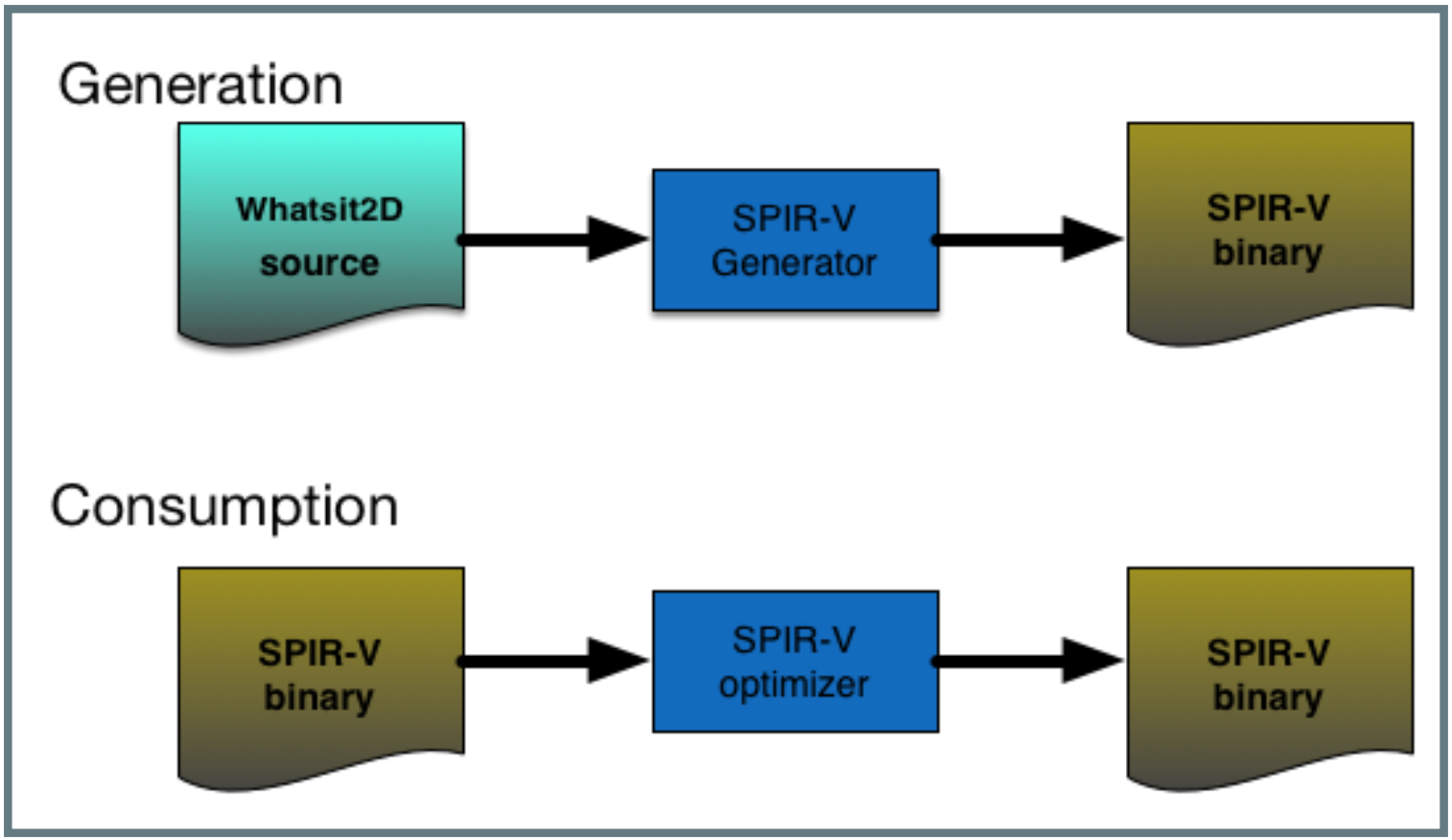
TOOL FLOW

- Compiler front-end (e.g. OpenCL C++ or GLSL) generates SPIR-V
- SPIR-V is consumed by:
  - 3rd party tool that tranforms the input and generates SPIR-V
  - 3rd party tool (e.g. GPU driver) that generates machine specific binary

EXAMPLE OPENCL SPIR-V TOOL FLOW



EXAMPLE WHATSIT2D SPIR-V TOOL FLOW



REPRESENTATION

- A Binary Intermediate Language
  - A linear stream of words (32-bits)
- Functions inside a module represented as a Control-Flow-Graph (CFG):
  - Static Single Assignment (SSA) form
  - Structured control
  - Logicial and physical addressing models (i.e. GLSL vs OpenCL C style addressing)

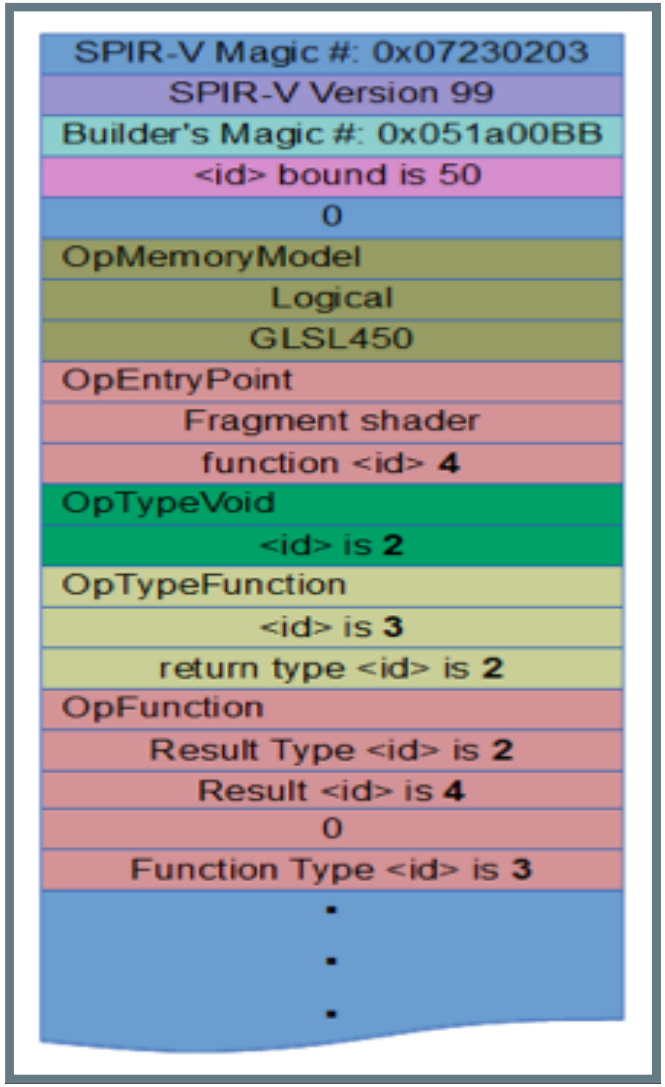
REPRESENTATION

- Standard load/Store architecture
- Data objects are represented logically, with hierarchical type information
  - e.g. No flattening of aggregates or assignment to physical registers
- Extendable
- Debug information

BINARY FORM

- Stream of 32-bit unsigned integers
- Not a file format, per say, however
  - works well to start file with magic number

EXAMPLE LAYOUT

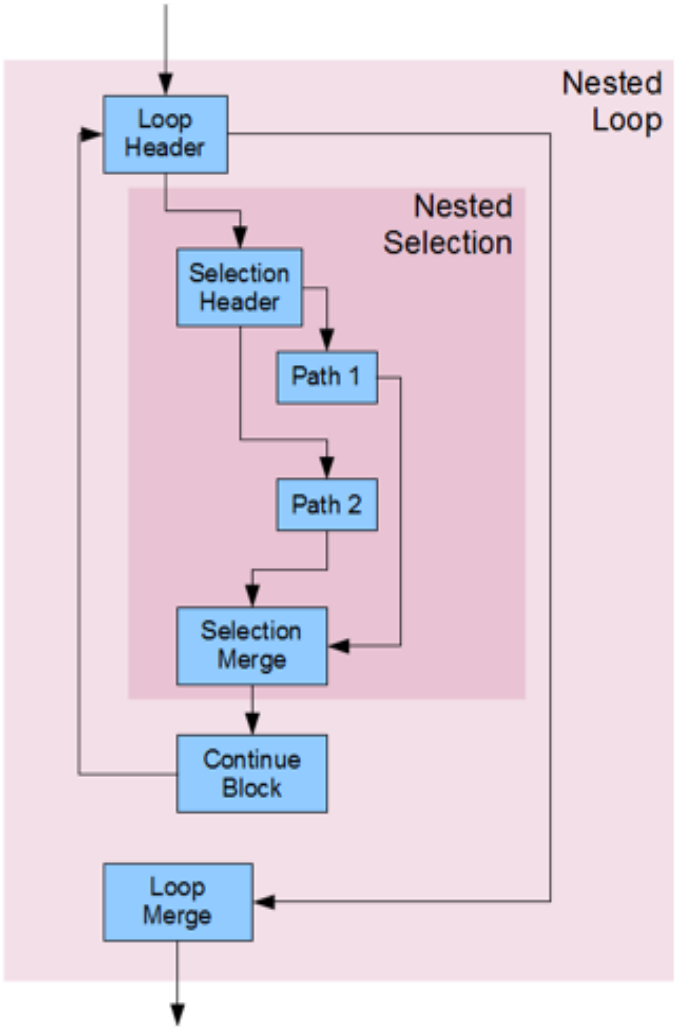


STRUCTURED CONTROL FLOW

- GLSL supports only structured control flow
- Some (read most) DSLs might only support structured control flow
- Some compilers can benifit from this knowledge
  - No need to perform expensive unstructured to structured control-flow transformation, necessary for some GPUs
  - Easier for some optimizations
  - Would be interesting to know how many modern drivers/compilers actually benifit from this!

STRUCTURED CONTROL FLOW

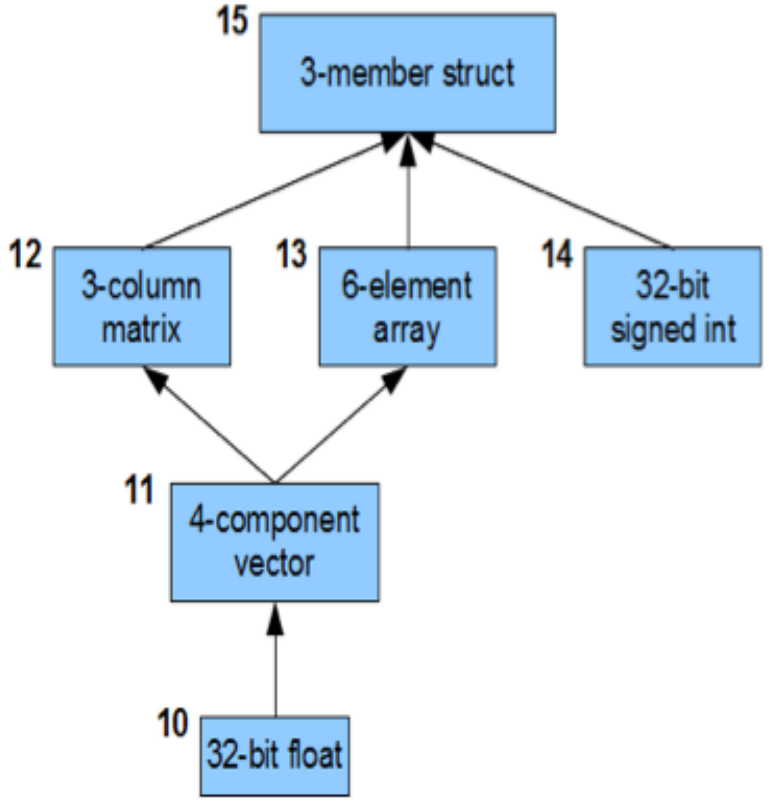
```
for (...) {


    if (...)


        ...
    else

        ...


    ...


}
```




```
11: Label
    ...
    LoopMerge 12 NoControl
    BranchConditional 18 19 12
19:   Label
22:   ...
      SelectionMerge 24 NoControl
      BranchConditional 22 23 28
23:     Label
        ...
        Branch 24
28:     Label
        ...
        Branch 24

24:   Label
      ...


      Branch 11


12: Label
```

HIERARCHICAL TYPES, CONSTANTS, AND OBJECTS

```
struct {
    mat3x4;
    vec4[6];
    int;
};
```




```
10: OpTypeFloat 32
11: OpTypeVector 10 4
12: OpTypeMatrix 11 3
13: OpTypeArray 11 6
14: OpTypeInt 32 1
15: OpTypeStruct 12 13 14
```

CONCLUSION

- An IL for ALL Khronos APIs
- Simply and extendable
- Core in Valcan and OpenCL 2.1
- Support for OpenCL C 1.2 and 2.0 and C++ 2.1

CONCLUSION

- Opens path to more than just GLSL
- A new shading language called GKRK
  - aka What's That Elephant Doing In The Room?