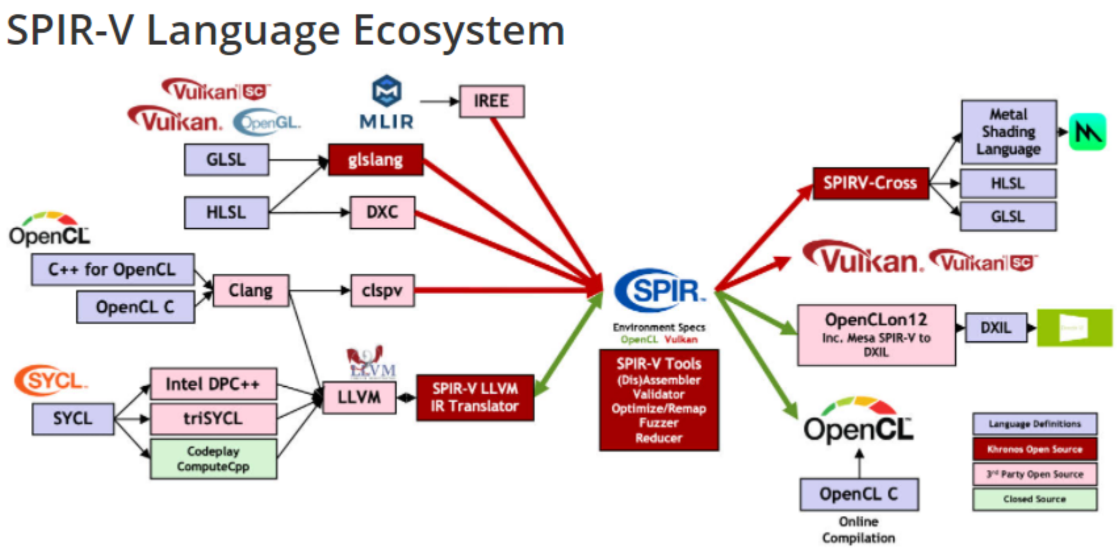
Shader modules

https://vulkan-tutorial.com/Drawing_a_triangle/Graphics_pipeline_basics/Shader_modules

shader code in Vulkan has to be specified in a bytecode format as opposed to human-readable syntax like GLSL and HLSL. This bytecode format is called SPIR-V and is designed to be used with both Vulkan and OpenCL (both Khronos APIs). It is a format that can be used to write graphics and compute shaders, but we will focus on shaders used in Vulkan's graphics pipelines in this tutorial.

如官方介绍:

Shader只是一段**可执行的汇编代码,**无论Warp层用的是哪种格式最终都会被翻译成字节码来执行，SPIR-V：



The SPIR-V ecosystem includes a rich variety of language front-ends, tools and run-times

如：UE4 中的glslang，也就是glslangValidator，UE4.25中增加了Shader Conductor，但毫无疑问，最终还是得转化为SPIR-V

Vulkan在生成ShaderModule的时候会对其进行转化

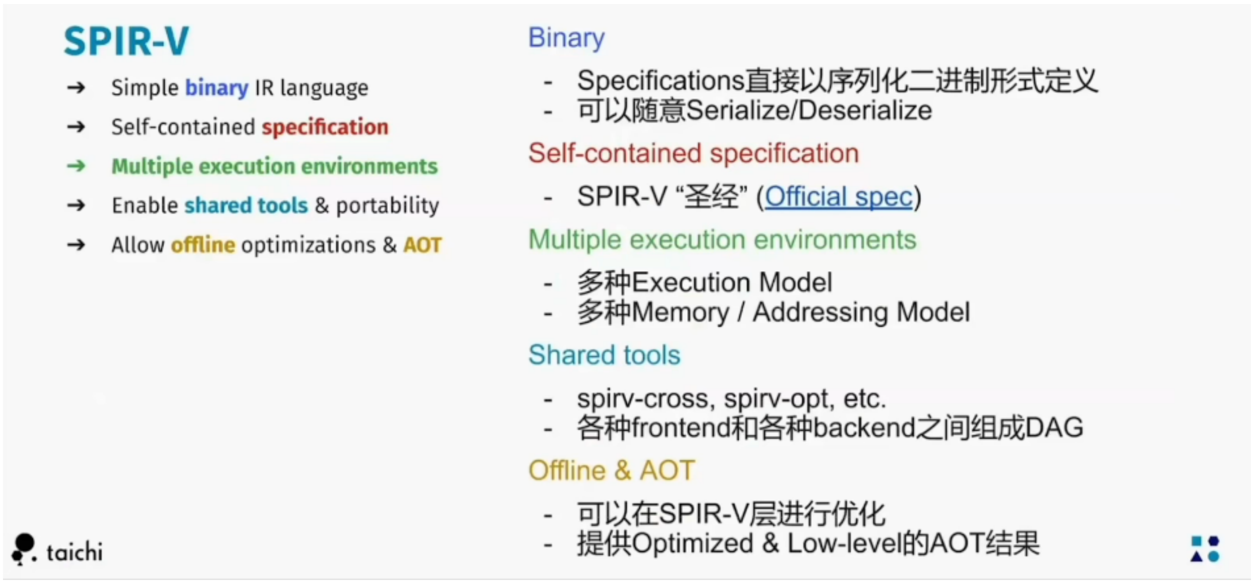
```
Tools::AutoDeleter<VkShaderModule, PFN_vkDestroyShaderModule> Tutorial03::CreateShaderModule(const char* filename) {
        const std::vector<char> code = Tools::GetBinaryFileContents(filename);
        if (code.size() == 0) {
            return Tools::AutoDeleter<VkShaderModule, PFN_vkDestroyShaderModule>();
        }

        VkShaderModuleCreateInfo shader_module_create_info = {
          VK_STRUCTURE_TYPE_SHADER_MODULE_CREATE_INFO,     // VkStructureType                sType
          nullptr,                                         // const void                    *pNext
          0,                                               // VkShaderModuleCreateFlags      flags
          code.size(),                                     // size_t                         codeSize
          reinterpret_cast<const uint32_t*>(code.data())  // const uint32_t                *pCode
        };

        VkShaderModule shader_module;
        if (vkCreateShaderModule(GetDevice(), &shader_module_create_info, nullptr, &shader_module) != VK_SUCCESS) {
            std::cout << "Could not create shader module from a \"" << filename << "\" file!" << std::endl;
            return Tools::AutoDeleter<VkShaderModule, PFN_vkDestroyShaderModule>();
        }

        return Tools::AutoDeleter<VkShaderModule, PFN_vkDestroyShaderModule>(shader_module, vkDestroyShaderModule, GetDevice());
    }
```

对VkShaderModuleCreateInfo的函数解析https://www.khronos.org/registry/vulkan/specs/1.3-extensions/man/html/VkShaderModuleCreateInfo.html

看起来对shader字节的处理应该就在这里了

SPIR-V 的介绍：

https://www.bilibili.com/video/BV1RL4y1g7P1?spm_id_from=333.337.search-card.all.click



SPIR-V

- → Simple **binary** IR language
- → Self-contained **specification**
- → **Multiple execution environments**
- → Enable **shared tools** & portability
- → Allow **offline** optimizations & **AOT**

Binary
- Specifications直接以序列化二进制形式定义
- 可以随意Serialize/Deserialize

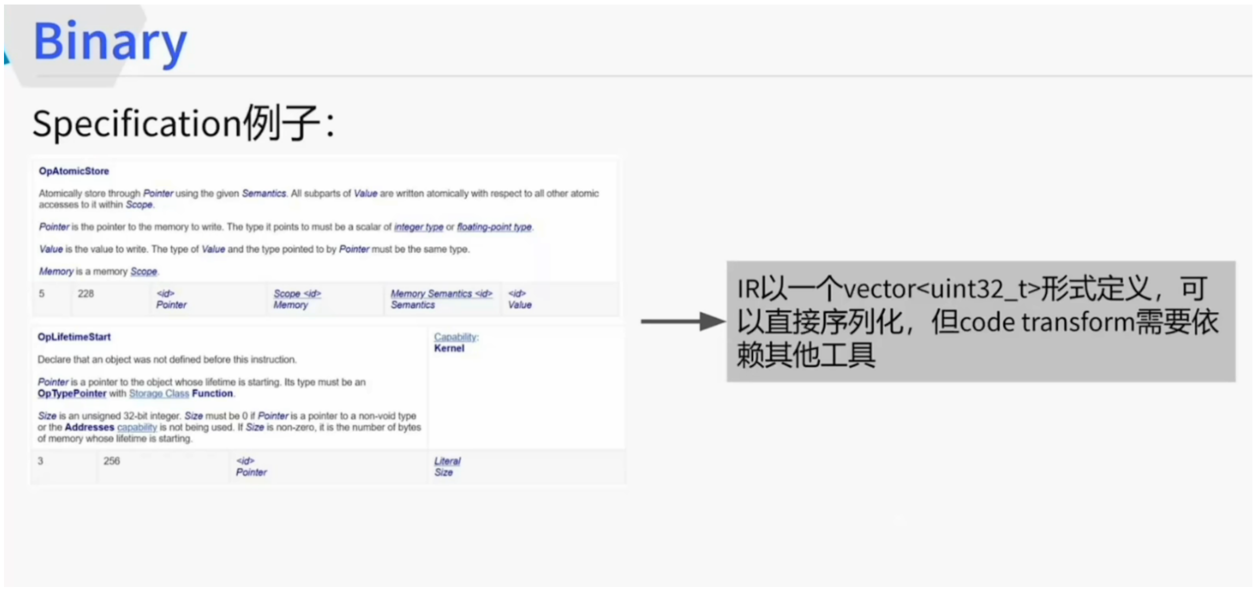
Self-contained specification
- SPIR-V “圣经” (Official spec)

Multiple execution environments
- 多种Execution Model
- 多种Memory / Addressing Model

Shared tools
- spirv-cross, spirv-opt, etc.
- 各种frontend和各种backend之间组成DAG

Offline & AOT
- 可以在SPIR-V层进行优化
- 提供Optimized & Low-level的AOT结果

taichi



Binary

Specification例子：

OpAtomicStore

Atomically store through *Pointer* using the given *Semantics*. All subparts of *Value* are written atomically with respect to all other atomic accesses to it within *Scope*.

Pointer is the pointer to the memory to write. The type it points to must be a scalar of *integer type* or *floating-point type*.

Value is the value to write. The type of *Value* and the type pointed to by *Pointer* must be the same type.

Memory is a memory *Scope*.

5	228	*<id>* *Pointer*	*Scope <id>* *Memory*	*Memory Semantics <id>* *Semantics*	*<id>* *Value*

OpLifetimeStart

Declare that an object was not defined before this instruction.

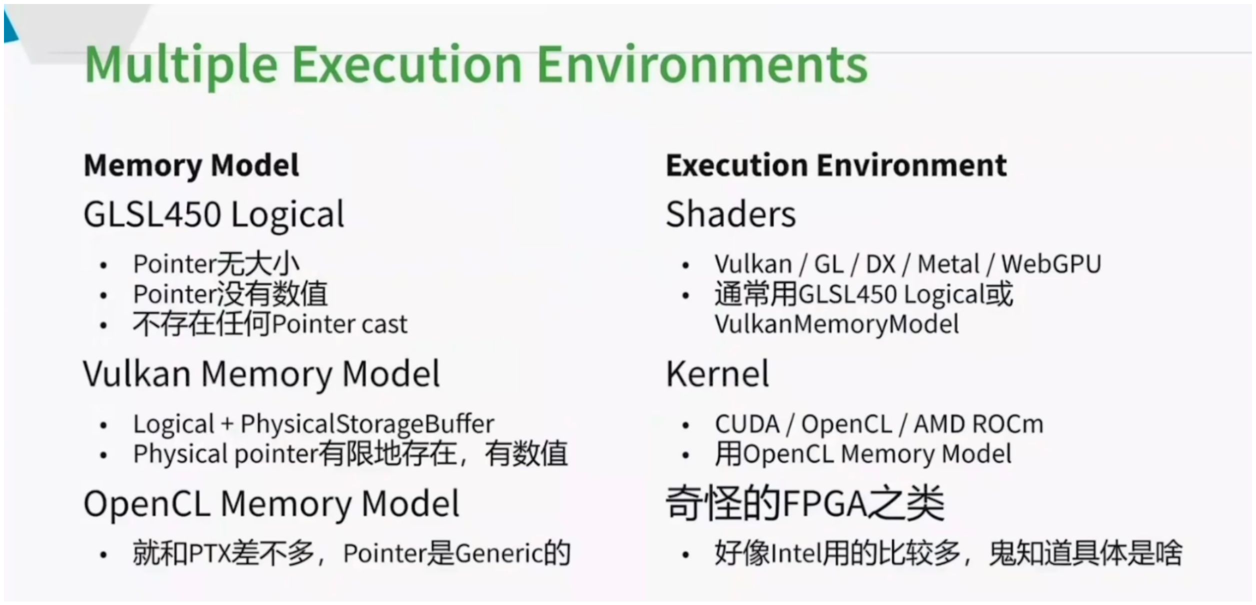
Pointer is a pointer to the object whose lifetime is starting. Its type must be an **OpTypePointer** with Storage Class **Function**.

Size is an unsigned 32-bit integer. *Size* must be 0 if *Pointer* is a pointer to a non-void type or the **Addresses** capability is not being used. If *Size* is non-zero, it is the number of bytes of memory whose lifetime is starting.

Capability:
Kernel

3	256	*<id>* *Pointer*	*Literal* *Size*

IR以一个vector<uint32_t>形式定义，可以直接序列化，但code transform需要依赖其他工具



Multiple Execution Environments

Memory Model

GLSL450 Logical

- Pointer无大小
- Pointer没有数值
- 不存在任何Pointer cast

Vulkan Memory Model

- Logical + PhysicalStorageBuffer
- Physical pointer有限地存在，有数值

OpenCL Memory Model

- 就和PTX差不多，Pointer是Generic的

Execution Environment

Shaders

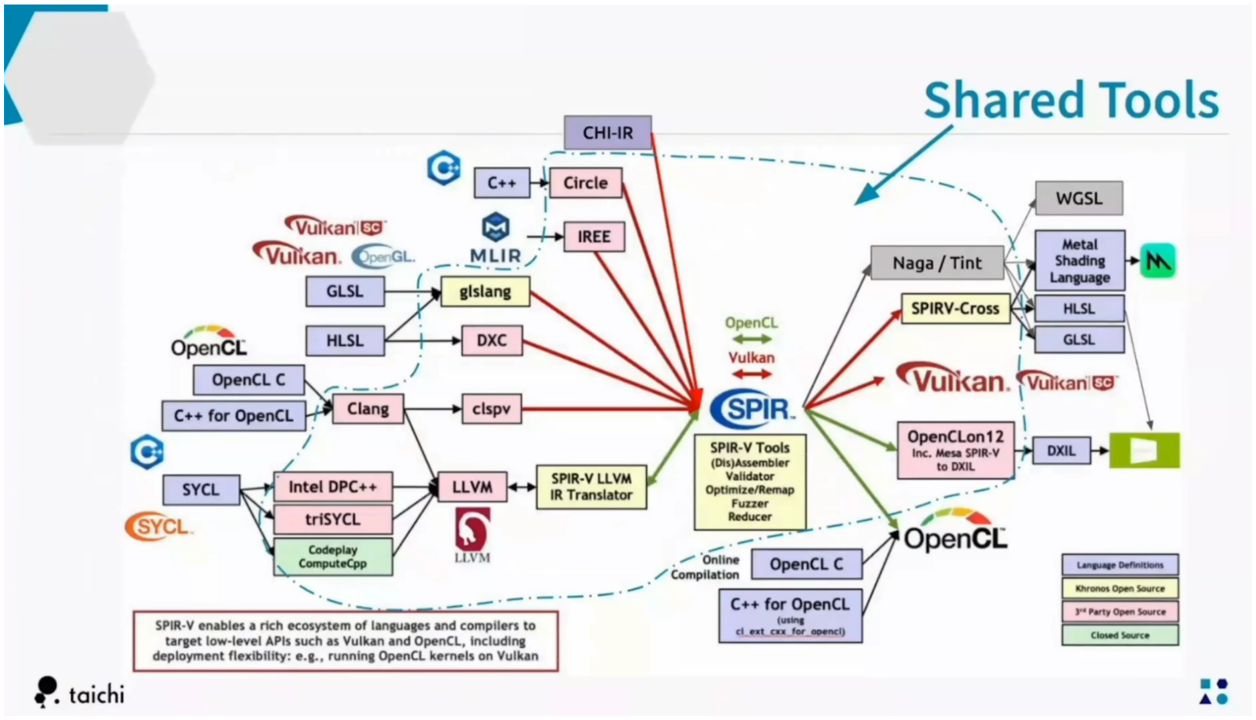
- Vulkan / GL / DX / Metal / WebGPU
- 通常用GLSL450 Logical或VulkanMemoryModel

Kernel

- CUDA / OpenCL / AMD ROCm
- 用OpenCL Memory Model

奇怪的FPGA之类

- 好像Intel用的比较多，鬼知道具体是啥



SPIR-V enables a rich ecosystem of languages and compilers to target low-level APIs such as Vulkan and OpenCL, including deployment flexibility: e.g., running OpenCL kernels on Vulkan