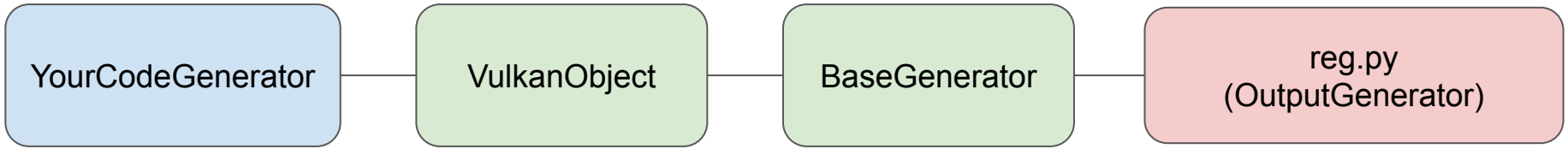

Vulkan Code Generation
Spencer Fricke
LunarG, Inc.
1
LUNARG

Lesson Learned (from years of doing it the old ways)

- Code generation is important!
- Python code is maintained by C++ programmers
  - No coding standards enforced
- **vk.xml** is a single dimension view of the API
  - Need to think of it as a relational database
  - Spec has a reg.py script/workflow to help
  - Easy to get wrong / not know about every edge case
- Most people just want to access data
  - Don’t want to understand random python frameworks
  - Don’t want to become vk.xml spec experts

New workflow



Lives in project repo
(ex. Validation Layers)

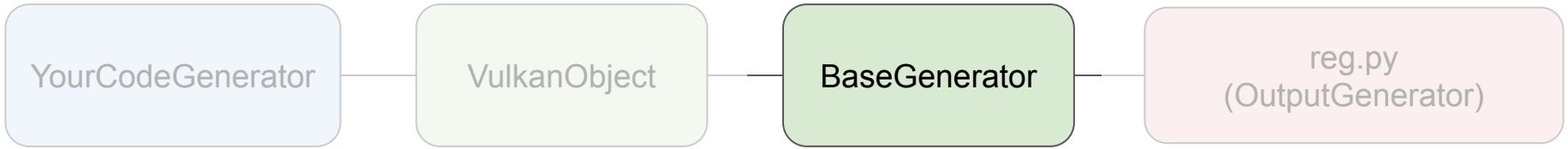
Newly added to the Vulkan Headers in 1.4.309

Original helper script in
the Vulkan Headers

New workflow

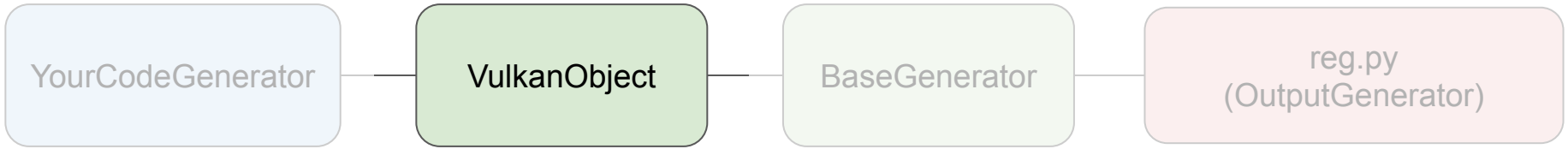
- Parses XML and builds relationships
  - Ex. Which extensions for which objects
  - Has callbacks (ex. genCmd, beginFeature, etc)
- Used by other users
  - Ex. Spec generation

New workflow



- Understands the reg.py framework “passes”
- Job is to map it to VulkanObject
- Only few people need to understand this code
- Handles when XML schema changes occur

New workflow

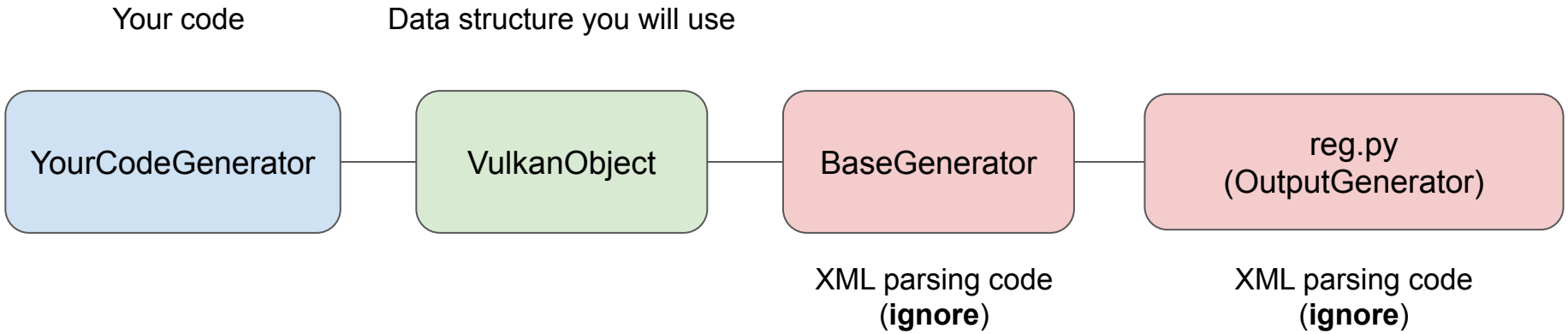


- Python version of a C-Header
- Strongly typed representation of objects
  - using Python Dataclasses
- Easy to see what can be grabbed

New workflow

- inherit a single **generate()** function
- Only focus is to output C++
- Shouldn’t need to build own data structures
- Has own mutable VulkanObject instance if wants to make changes

Put simply


Your code
Data structure you will use
YourCodeGenerator
VulkanObject
BaseGenerator
reg.py
(OutputGenerator)
XML parsing code
(**ignore**)
XML parsing code
(**ignore**)

Example Task

- Create generation code that
  - Gets all Vulkan functions
  - That have parameter of of type **uint32_t**
  - Was added in **Vulkan 1.2**

```python
@dataclass
class VulkanObject():
    headerVersion: int = 0 # value of VK_HEADER_VERSION

    extensions: Dict[str, Extension] = field(default_factory=dict, init=False)
    versions:   Dict[str, Version]   = field(default_factory=dict, init=False)

    handles:  Dict[str, Handle]      = field(default_factory=dict, init=False)
    commands: Dict[str, Command]     = field(default_factory=dict, init=False)
    structs:  Dict[str, Struct]      = field(default_factory=dict, init=False)
    enums:    Dict[str, Enum]        = field(default_factory=dict, init=False)
    bitmasks: Dict[str, Bitmask]     = field(default_factory=dict, init=False)
    formats:  Dict[str, Format]      = field(default_factory=dict, init=False)

    syncStage:    List[SyncStage]    = field(default_factory=list, init=False)
    syncAccess:   List[SyncAccess]   = field(default_factory=list, init=False)
    syncPipeline: List[SyncPipeline] = field(default_factory=list, init=False)

    spirv: List[Spirv]               = field(default_factory=list, init=False)

    # ex) [ xlib : VK_USE_PLATFORM_XLIB_KHR ]
    platforms: Dict[str, str]        = field(default_factory=dict, init=False)
    # List of all vendor Sufix names (KHR, EXT, etc. )
    vendorTags: List[str]            = field(default_factory=list, init=False)
    # ex) [ Queues.COMPUTE : VK_QUEUE_COMPUTE_BIT ]
    queueBits: Dict[IntFlag, str]    = field(default_factory=dict, init=False)
```

```python
@dataclass
class VulkanObject():
    headerVersion: int = 0 # value of VK_HEADER_VERSION

    extensions: Dict[str, Extension] = field(default_factory=dict, init=False)
    versions:   Dict[str, Version]   = field(default_factory=dict, init=False)

    handles:    Dict[str, Handle]    = field(default_factory=dict, init=False)
    commands: Dict[str, Command]     = field(default_factory=dict, init=False)
    structs:    Dict[str, Struct]    = field(default_factory=dict, init=False)
    enums:    Dict[str, Enum]        = field(default_factory=dict, init=False)
    bitmasks: Dict[str, Bitmask]     = field(default_factory=dict, init=False)
    formats:  Dict[str, Format]      = field(default_factory=dict, init=False)

    syncStage:    List[SyncStage]    = field(default_factory=list, init=False)
    syncAccess:   List[SyncAccess]   = field(default_factory=list, init=False)
    syncPipeline: List[SyncPipeline] = field(default_factory=list, init=False)

    spirv: List[Spirv]               = field(default_factory=list, init=False)

    # ex) [ xlib : VK_USE_PLATFORM_XLIB_KHR ]
    platforms: Dict[str, str]        = field(default_factory=dict, init=False)
    # List of all vendor Sufix names (KHR, EXT, etc. )
    vendorTags: List[str]            = field(default_factory=list, init=False)
    # ex) [ Queues.COMPUTE : VK_QUEUE_COMPUTE_BIT ]
    queueBits: Dict[IntFlag, str]    = field(default_factory=dict, init=False)
```

Easy!

```
class ExampleOutputGenerator(BaseGenerator):
    def __init__(self):
        BaseGenerator.__init__(self)

    def generate(self):
        for command in [x for x in self.vk.commands.values() if x.version and x.version.name == 'VK_VERSION_1_2']:
            for param in [x for x in command.params if x.type == 'uint32_t']:
                print(f'{command.name} | {param.cDeclaration}')
```

- Used to literally be at least 100 lines of code to do this before

```
class ExampleOutputGenerator(BaseGenerator):
    def __init__(self):
        BaseGenerator.__init__(self)

    def generate(self):
        for command in [x for x in self.vk.commands.values() if x.version and x.version.name == 'VK_VERSION_1_2']:
            for param in [x for x in command.params if x.type == 'uint32_t']:
                print(f'{command.name} | {param.cDeclaration}')
```

- Simply inherit new **BaseGenerator**

```
class ExampleOutputGenerator(BaseGenerator):
    def __init__(self):
        BaseGenerator.__init__(self)

    def generate(self):
        for command in [x for x in self.vk.commands.values() if x.version and x.version.name == 'VK_VERSION_1_2']:
            for param in [x for x in command.params if x.type == 'uint32_t']:
                print(f'{command.name} | {param.cDeclaration}')
```

- Entrypoint / “main” function
- Callback from **BaseGenerator**
- After **VulkanObject** has been created

```
class ExampleOutputGenerator(BaseGenerator):
    def __init__(self):
        BaseGenerator.__init__(self)

    def generate(self):
        for command in [x for x in self.vk.commands.values() if x.version and x.version.name == 'VK_VERSION_1_2']:
            for param in [x for x in command.params if x.type == 'uint32_t']:
                print(f'{command.name} | {param.cDeclaration}')
```

- Get an instance of **VulkanObject** with **self.vk**

```
class ExampleOutputGenerator(BaseGenerator):
    def __init__(self):
        BaseGenerator.__init__(self)

    def generate(self):
        for command in [x for x in self.vk.commands.values() if x.version and x.version.name == 'VK_VERSION_1_2']:
            for param in [x for x in command.params if x.type == 'uint32_t']:
                print(f'{command.name} | {param.cDeclaration}')
```

- Gets all functions

```
class ExampleOutputGenerator(BaseGenerator):
    def __init__(self):
        BaseGenerator.__init__(self)

    def generate(self):
        for command in [x for x in self.vk.commands.values(  if x.version and x.version.name == 'VK_VERSION_1_2']
            for param in [x for x in command.params if x.ty e == 'uint32_t']:
                print(f'{command.name} | {param.cDeclaration}')
```

- Gets all functions
- Quickly sort if added in Vulkan 1.2

```
class ExampleOutputGenerator(BaseGenerator):
    def __init__(self):
        BaseGenerator.__init__(self)

    def generate(self):
        for command in [x for x in self.vk.commands.values() if x.version and x.version.name == 'VK_VERSION_1_2']:
            for param in [x for x in command.params if x.type == 'uint32_t']:
                print(f'{command.name} | {param.cDeclaration}')
```

- Gets all functions
- Quickly sort if added in Vulkan 1.2
- Get all parameters of the function

```
class ExampleOutputGenerator(BaseGenerator):
    def __init__(self):
        BaseGenerator.__init__(self)

    def generate(self):
        for command in [x for x in self.vk.commands.values() if x.version and x.version.name == 'VK_VERSION_1_2']:
            for param in [x for x in command.params if x.type == 'uint32_t']:
                print(f'{command.name} | {param.cDeclaration}')
```

- Gets all functions
- Quickly sort if added in Vulkan 1.2
- Get all parameters of the function
- Sort by type of the parameter

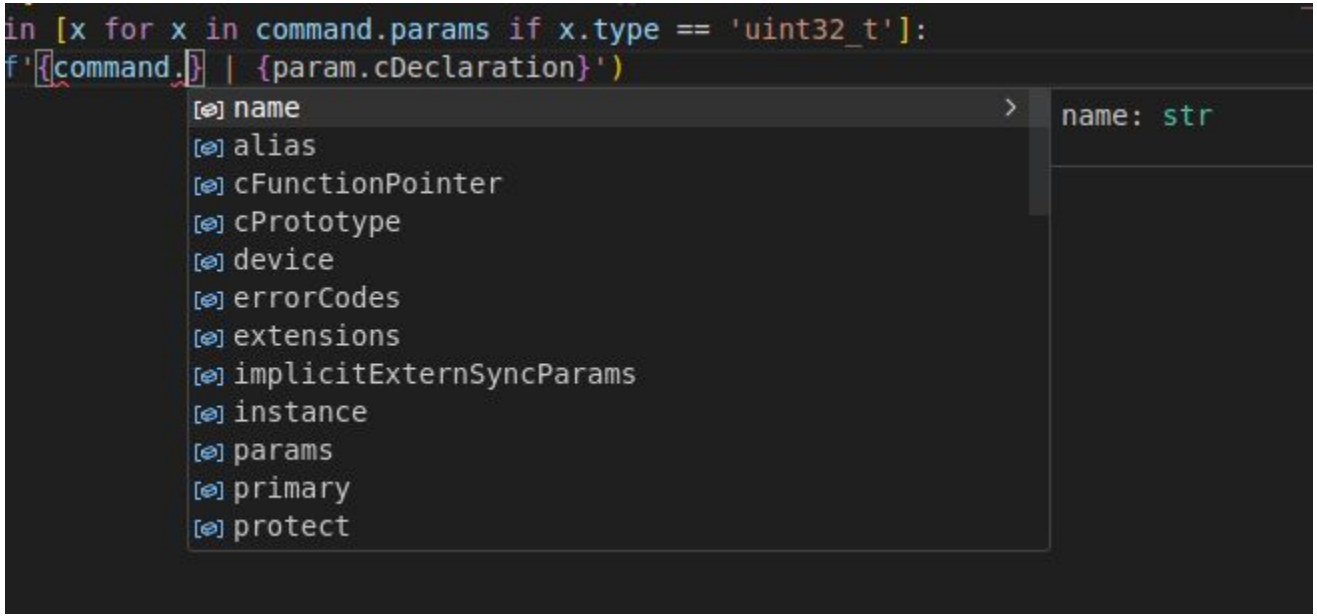
```python
class ExampleOutputGenerator(BaseGenerator):
    def __init__(self):
        BaseGenerator.__init__(self)

    def generate(self):
        for command in [x for x in self.vk.commands.values() if x.version and x.version.name == 'VK_VERSION_1_2']:
            for param in [x for x in command.params if x.type == 'uint32_t']:
                print(f'{command.name} | {param.cDeclaration}')
```

- Real output

```
vkCmdDrawIndirectCount |     uint32_t maxDrawCount
vkCmdDrawIndirectCount |     uint32_t stride
vkCmdDrawIndexedIndirectCount |     uint32_t maxDrawCount
vkCmdDrawIndexedIndirectCount |     uint32_t stride
vkResetQueryPool |     uint32_t firstQuery
vkResetQueryPool |     uint32_t queryCount
```

We now have intellisense**!!!**



Can know if things will be “null” or not

```python
@dataclass
class Extension:
    """<extension>"""
    name: str # ex) VK_KHR_SURFACE
    nameString: str # marco with string, ex) VK_KHR_SURFACE_EXTENSION_NAME

    # Only one will be True, the other is False
    instance: bool
    device: bool

    depends: (str | None)
    vendorTag: (str | None)   # ex) EXT, KHR, etc
    platform: (str | None)    # ex) android
    protect: (str | None)     # ex) VK_USE_PLATFORM_ANDROID_KHR
    provisional: bool
    promotedTo: (str | None) # ex) VK_VERSION_1_1
    deprecatedBy: (str | None)
    obsoletedBy: (str | None)
    specialUse: list[str]
```

Other things to note

- Requires Python 3.10
- The goal is people writing code gen only care about **vulkan_object.py**
  - Only a few people need to understand the vk.xml and how it evolves
- Very fast to generate **VulkanObject**
- Optional way to serialized it with Pickle it if generating many files