Vulkanised 2026

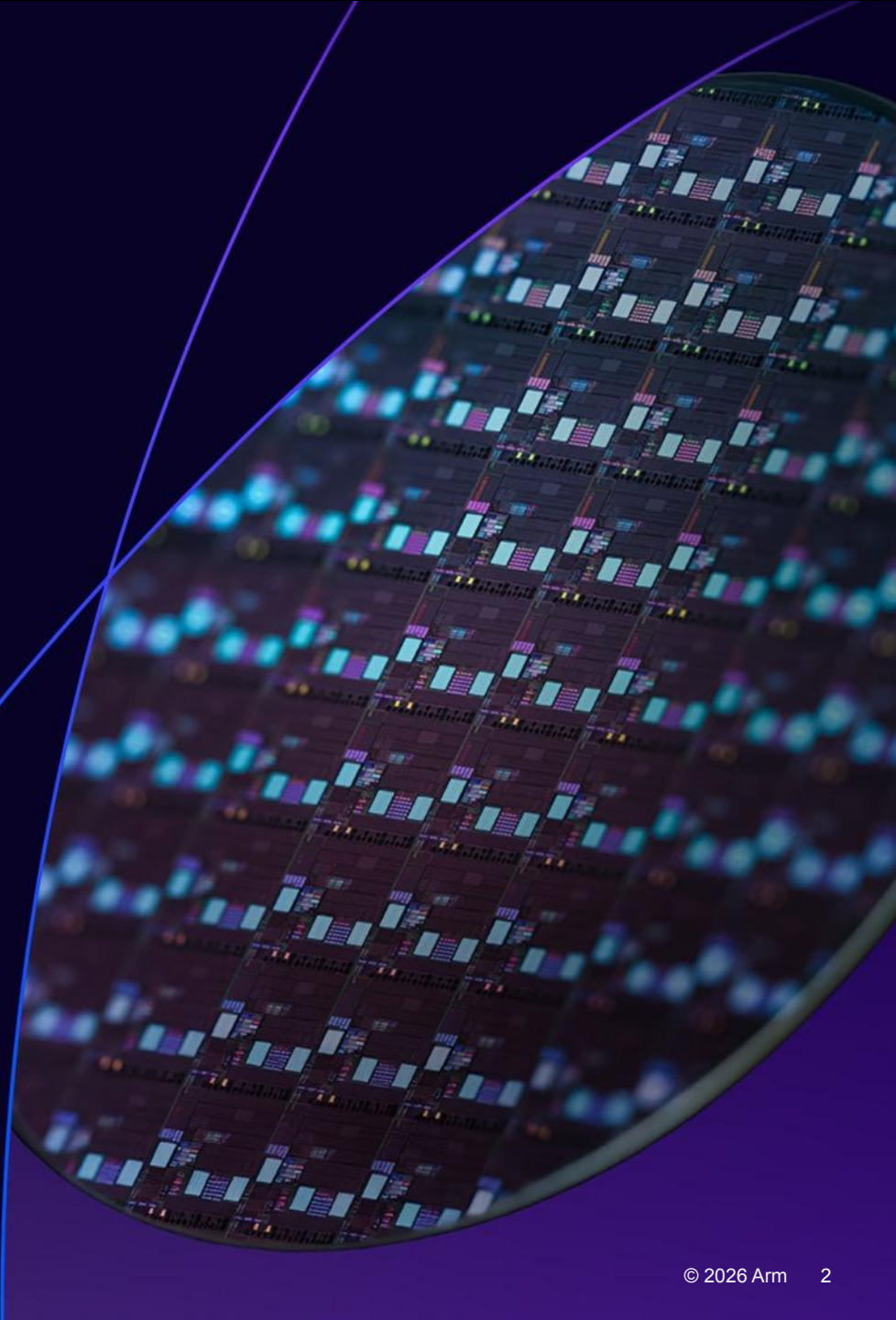
The 8th Vulkan Developer Conference
San Diego, USA | February 9-11, 2026

Implementing Neural Algorithms in Rendering Architectures

Kal Penev, Arm

ML on mobile

- Very different constraints to desktop
- Requires a new paradigm to reach optimal performance
- New way for doing ML inference in Vulkan
  - VK_ARM_tensors
  - VK_ARM_data_graph
  - Neural network is compiled into a single-dispatch VkPipeline rather than multiple shaders
- Higher level of abstraction, allows broader spectrum of ML acceleration hardware

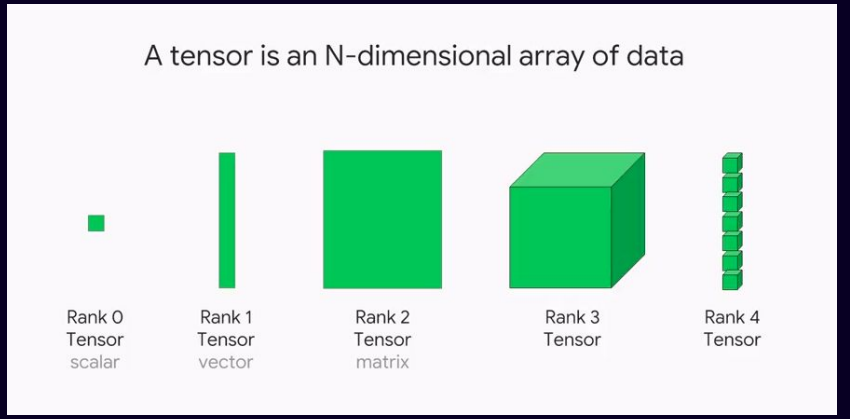


Overview

- VK_ARM_tensors
- VK_ARM_data_graph
- Differences to ML in compute shaders
- Integration of Neural Super Sampling in a Vulkan backend
  - Algorithm overview
  - Tensors
  - Neural Network: From Python to Vulkan
- Synchronization
- Additions to a Rendering Hardware Interface
- Conclusions

Tensor Extension (VK_ARM_tensors)

- Adds '**VkTensor**' as a new first-class resource type, alongside **VkImage** and **VkBuffer**
  - More dimensions!
  - Optimal tiling
  - New API patterns same as VkImage but with tensor keyword



https://data101.org/notes/data-modeling/data-models/

```
// Provided by VK_ARM_tensors
typedef struct VkTensorDescriptionARM {
    VkStructureType          sType;
    const void*              pNext;
    VkTensorTilingARM        tiling;
    VkFormat                 format;
    uint32_t                 dimensionCount;
    const int64_t*           pDimensions;
    const int64_t*           pStrides;
    VkTensorUsageFlagsARM    usage;
} VkTensorDescriptionARM;
```

Linear / Optimal
e.g. VK_FORMAT_R8_SINT
Rank - 3D, 4D, 5D, etc.
Shape, e.g. (1, 1080, 1920, 4)
Packing
How you want to use it

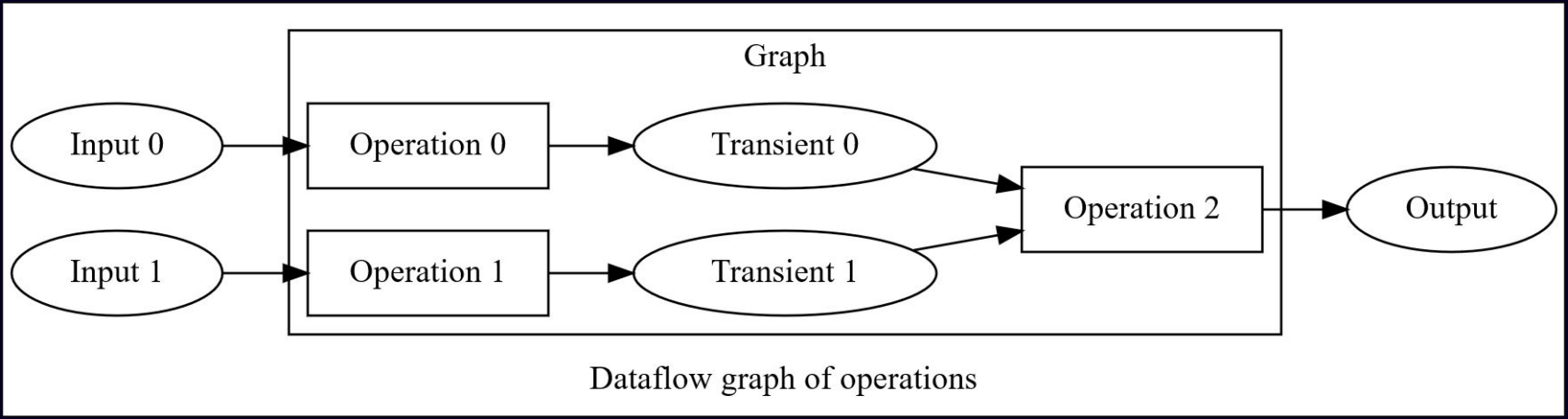
Tensor Extension (GL_ARM_tensors)

- Read/write operations in GLSL shaders:
  - **Tensor** is a new binding type

```
// Rank: 4
// Element type: float16_t
// Shape: {1, 1920, 1080, 8} → tensorSizeARM(...)

layout(set = 0, binding = 0) uniform tensorARM<float16_t, 4> my_tensor;

uint uv_coords[4] = {0, 14, 15, 2};

float16_t data[6];

tensorReadARM(my_tensor, uv_coords, data); // reads 6 elements _only_ along innermost dimension

// data == float16_t[]{tensor[0][14][15][2], tensor[0][14][15][3], ...}

value[3] = value[3] * 2.0hf;

tensorWriteARM(my_tensor, uv_coords, value);
```

Data Graph Extension (VK_ARM_data_graph)

- A data (or dataflow) graph describes the behaviour of a program in terms of how its input data is transformed into output data in a declarative fashion
  - Semantic model of a program rather than a list of arithmetic operations



Dataflow graph of operations

- **Data Graph Pipeline -** a new type of Vulkan pipeline
  - Same usage patterns as **Compute Pipelines** for constructs like **Pipeline Layout**, **Descriptor Sets**, etc.
  - Essentially a loosely-defined “chain of ML shaders”

Data Graph Extension (VK_ARM_data_graph)

- **Data Graph Shader Module** - SPIR-V module with TOSA operations
  - Tensor Operator Set Architecture (**TOSA**) is an open spec IR at the *tensor level of abstraction*
  - Clear definition of numerical behaviour, testable against a reference implementation
  - Neural networks authored in ML frameworks (PyTorch, TensorFlow, etc.)
  - TOSA SPIR-V ops take **whole inputs (**and attributes**)** to produce **whole outputs**

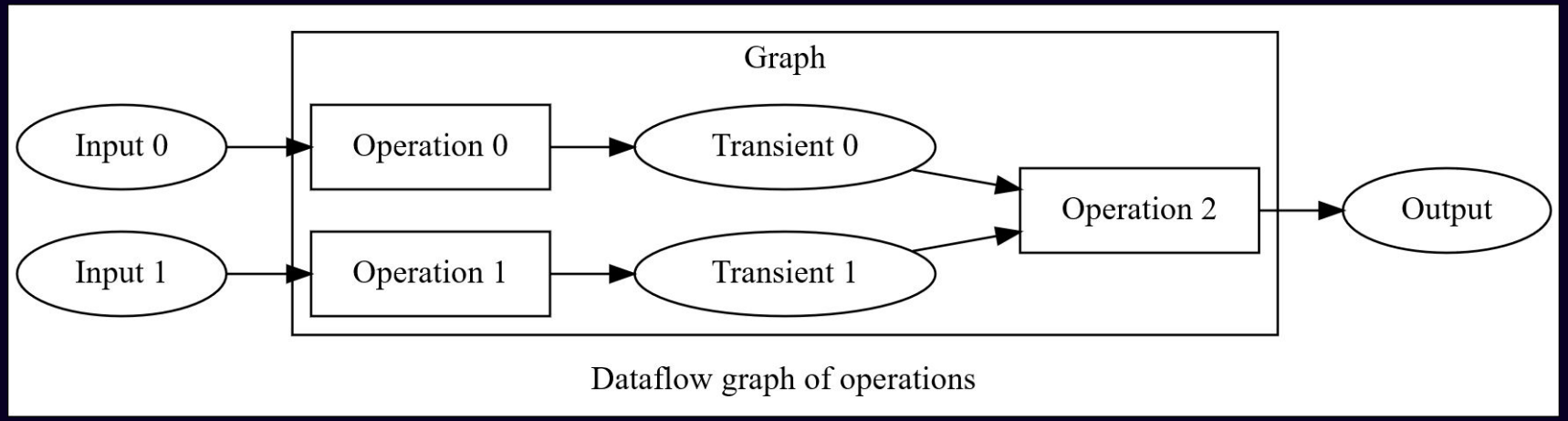
TOSA Operator

Attributes

```
          %in = OpGraphInputARM %float_tensor
        %pool = OpExtInst %float_tensor MAX_POOL2D %avg_pool_kernel
%avg_pool_stride %avg_pool_pad %in
                OpGraphSetOutputARM %pool %uint_0
```

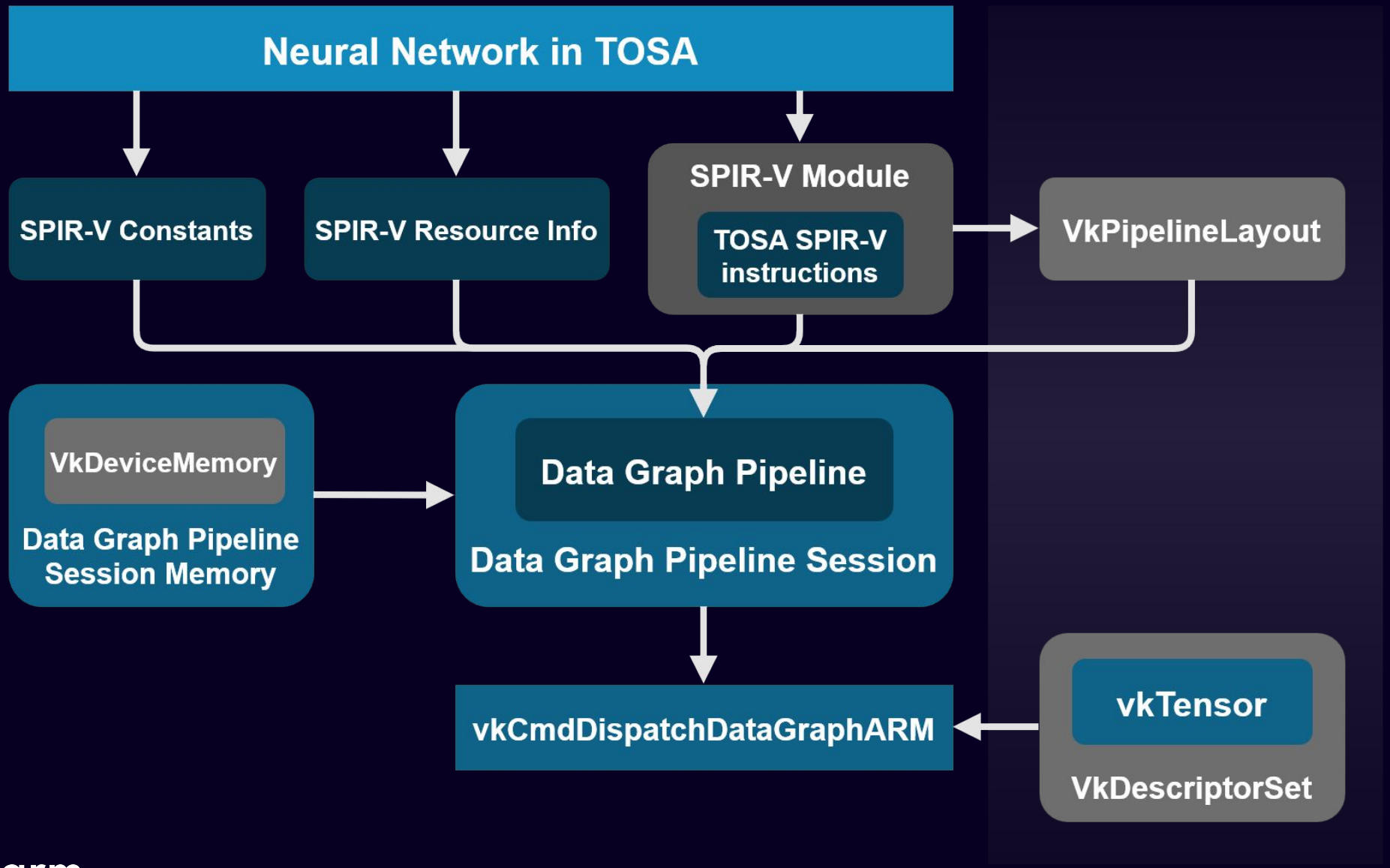

Data Graph Extension (VK_ARM_data_graph)

- **Data Graph Constant** for binary resources that are too large to hardcode in the SPIR-V
  - Referenced in SPIR-V code and loaded by a Vulkan mechanism instead of embedding data in the module
- **Data Graph Resource Info** to describe input and output variables to the graph
  - Required at pipeline creation time for shaping and validation
- **Data Graph Pipeline Session** to encapsulate pipeline
  - Describes memory required for the transient resources, and any other state required to execute the graph

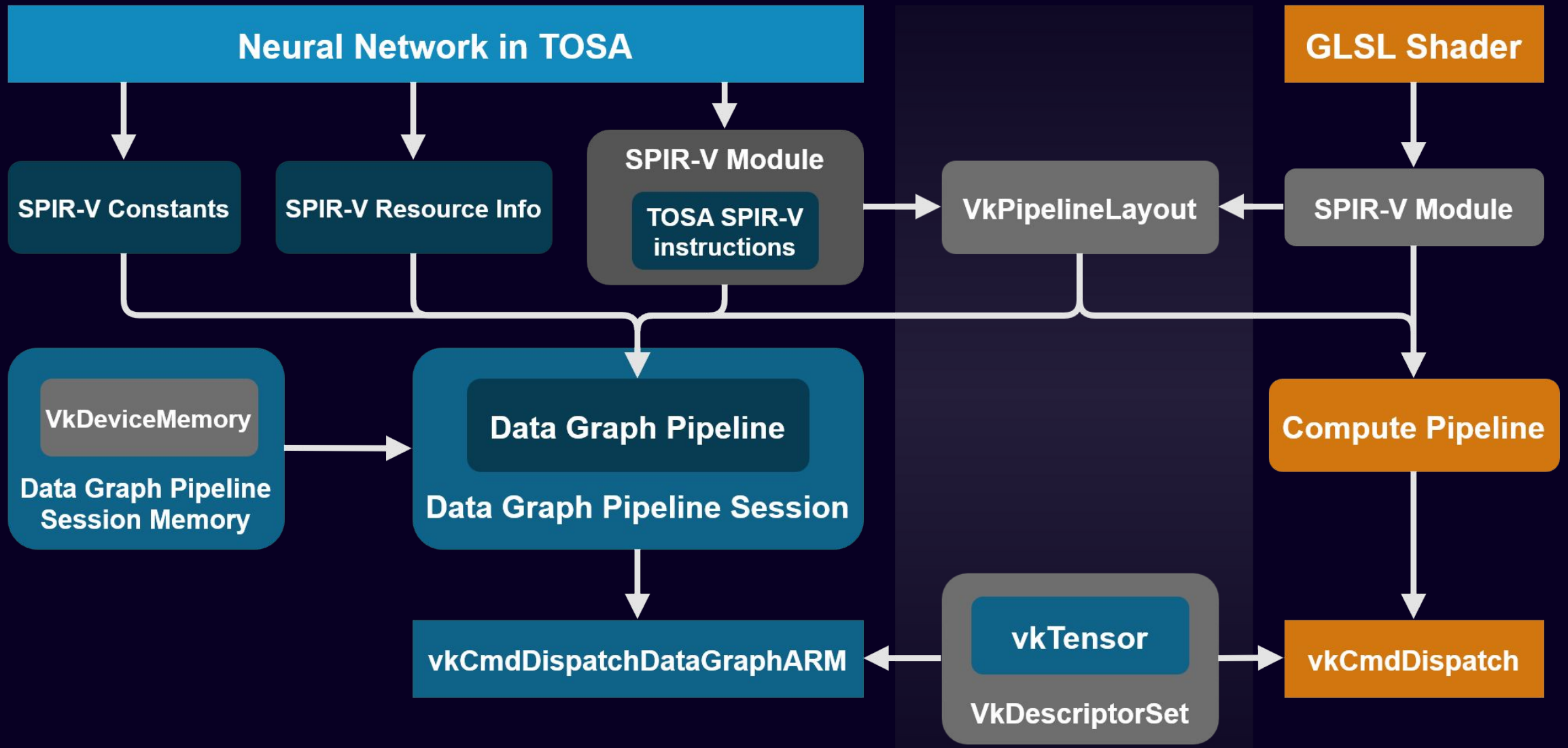


Dataflow graph of operations

Data Graph Extension (VK_ARM_data_graph)



Data Graph Pipeline compared to Compute Pipeline



Data Graph Extension (VK_ARM_data_graph)

- Whole-tensor execution model
  - No SIMT shader code
  - Set of primitive operators - **CONV2D**, **MAX_POOL2D**, **RESCALE**, etc.
  - Dispatch does not specify workgroup count

```
// Provided by VK_ARM_data_graph
void vkCmdDispatchDataGraphARM(
    VkCommandBuffer                         commandBuffer,
    VkDataGraphPipelineSessionARM           session,
    const VkDataGraphPipelineDispatchInfoARM*   pInfo);
```

```
// Provided by VK_ARM_data_graph
typedef struct VkDataGraphPipelineDispatchInfoARM {
    VkStructureType                         sType;
    void*                                   pNext;
    VkDataGraphPipelineDispatchFlagsARM     flags;
} VkDataGraphPipelineDispatchInfoARM;
```

Differences to ML workloads in compute shaders

- Why not just matrix multiplication in compute shaders?
  - No real convergence in ML accelerator architectures across different vendors (especially mobile)
  - Permutations in integrations result in ecosystem fragmentation
  - Whole-tensor abstraction is a solution
  - Data Graphs allow a higher-level API abstraction with less implementation constraints (device-level)

- TOSA and ML ecosystem
  - Allows ML engineers to work in ML frameworks and leverage known tools and workflows
  - Open-source collaboration and code sharing across platforms

- Driver implementation of the TOSA operators
  - Optimized bandwidth, as implementation defines data handling
  - Performance and power gains due to full graph context - operation fusion, reordering, tiled execution, etc.
  - Retrofitting of future hardware to API mechanisms without code changes

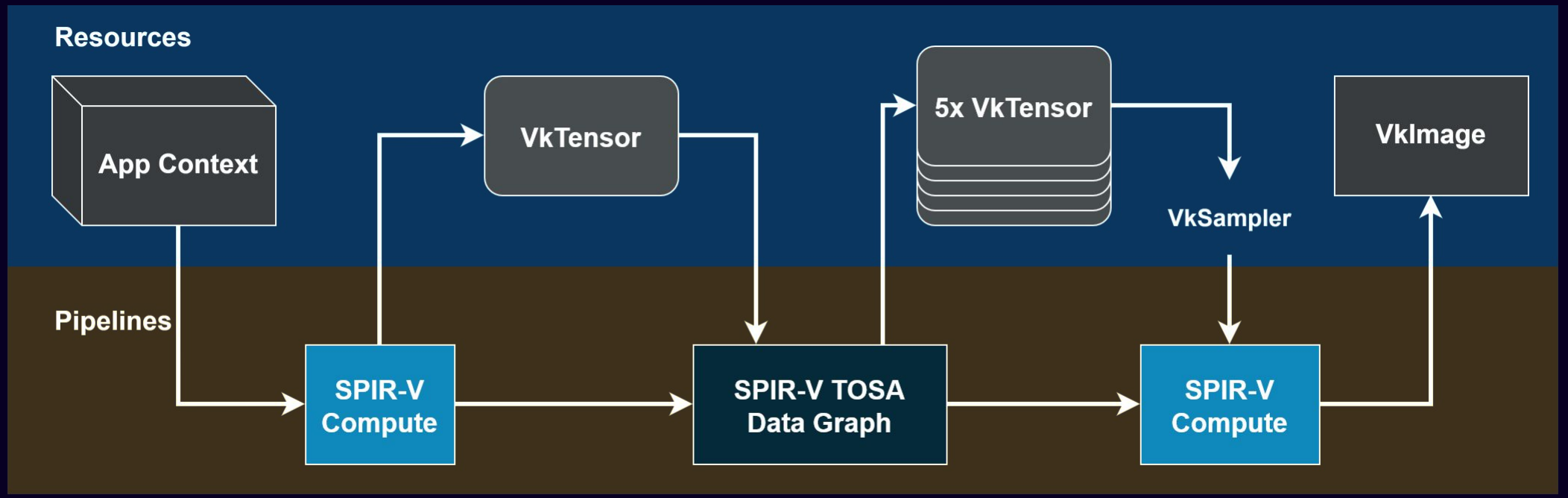
arm

Integration and code

Neural Super Sampling in a Vulkan backend

Neural Super Sampling Algorithm Overview

- Compute sandwich
  - 2 Compute Shaders
  - 1 Convolutional Neural Network



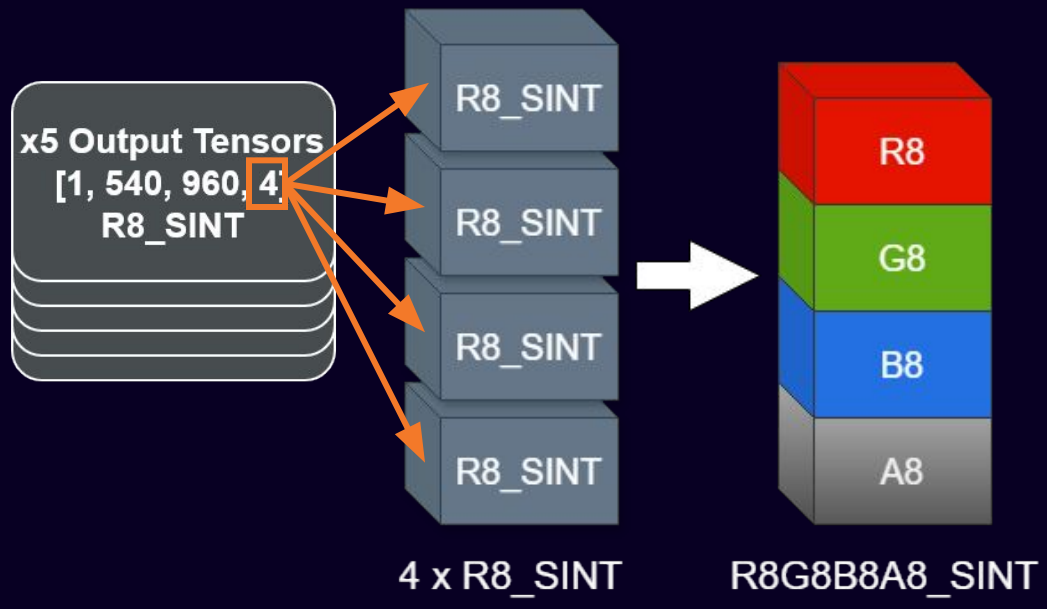
Tensors

```
VkTensorTilingARM tiling = VK_TENSOR_TILING_OPTIMAL_ARM;
VkFormat format = VK_FORMAT_R8_SINT;
std::vector<int64_t> dimensions{1, 540, 960, 4};
VkTensorUsageFlagsARM tensorUsage = VK_TENSOR_USAGE_SHADER_BIT_ARM | VK_TENSOR_USAGE_DATA_GRAPH_BIT_ARM;

VkTensorDescriptionARM tensorDescription{
    ...
    .tiling = tiling,
    .format = format,
    .dimensionCount = dimensions.size(),
    .pDimensions = dimensions.data(),
    .usage = tensorUsage};

// All further operations are 1:1 to the ones for VkImage

// VkTensorCreateInfoARM{...} with the description
// vkCreateTensorARM{...} with the create info

// Get memory requirements via vkGetTensorMemoryRequirementsARM...
// Allocate device memory...
// Bind memory via vkBindTensorMemoryARM...

// Create tensor view via vkCreateTensorViewARM...
```

```
// Provided by VK_ARM_tensors
typedef struct VkTensorDescriptionARM {
    VkStructureType          sType;
    const void*              pNext;
    VkTensorTilingARM        tiling;
    VkFormat                 format;
    uint32_t                 dimensionCount;
    const int64_t*           pDimensions;
    const int64_t*           pStrides;
    VkTensorUsageFlagsARM    usage;
} VkTensorDescriptionARM;
```

Tensor linear filtering

- Tensors encode image information in NSS
  - Shader after DG needs to sample tensor with a linear filter
  - Want to use a VkSampler to use the texture unit
  - … But VkSampler requires a VkImage
- **Solution**: Access tensor memory as a VkImage

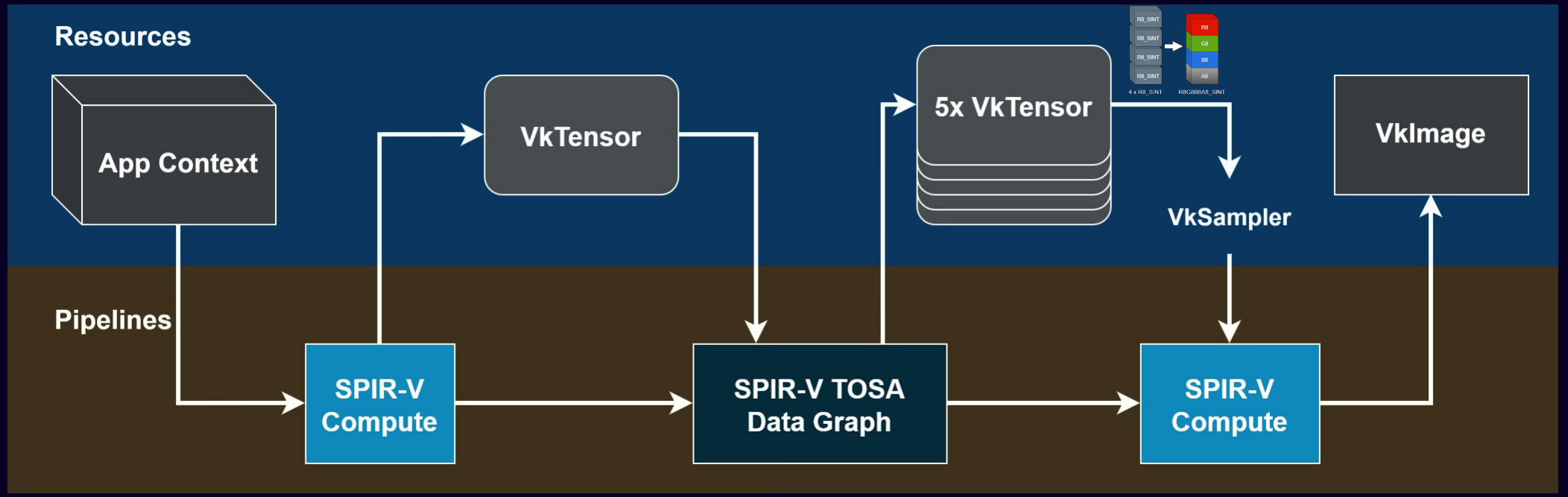


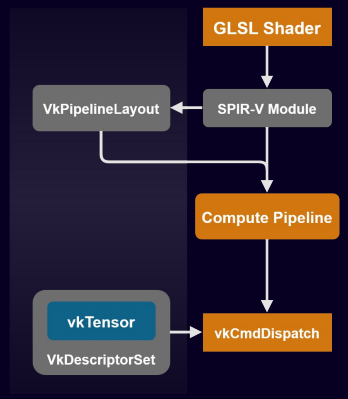
- API mechanisms required for optimal tiling
  - VK_IMAGE_USAGE_TENSOR_ALIASING_BIT_ARM
  - VK_TENSOR_USAGE_IMAGE_ALIASING_BIT_ARM
  - VK_IMAGE_LAYOUT_TENSOR_ALIASING_ARM
- Create image and tensor with the same memory
  - Combine memory requirements!

Format	VK_FORMAT_FEATURE_SAMPLED_IMAGE_FILTER_LINEAR_BIT	VK_FORMAT_FEATURE_SAMPLED_IMAGE_BIT	VK_FORMAT_FEATURE_BLIT_SRC_BIT	↓
VK_FORMAT_R8G8B8A8_SINT	✓	✓		
VK_FORMAT_R8G8B8A8_SNORM	✓	✓	✓	

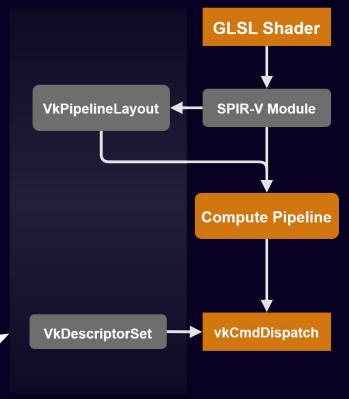
Data reinterpretation necessary

Neural Super Sampling in Vulkan





?



Access Tensor as a VkImage

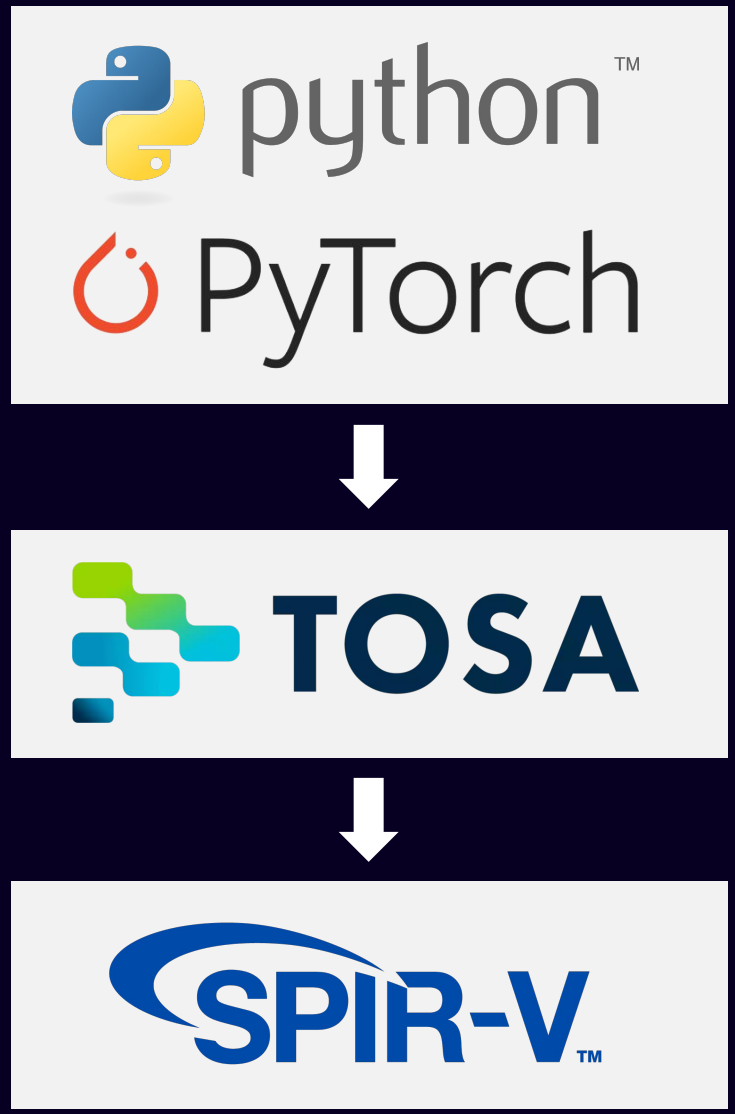
arm

Neural Network: From Python to Vulkan

1) ML Engineering in Python
   - **Output**: A Neural Network Model in a Python framework

2) Convert the Model to TOSA
   - **Output**: TOSA (Tensor Operator Set Architecture) IR
   - Represents the semantics of the neural network in a standard, backend-neutral manner
   - Open-source tooling for conversion from ML frameworks

3) Generate SPIR-V from TOSA
   - **Output**: SPIR-V module with TOSA instructions and data graph metadata
   - We use the vgf lib for this



Neural Network: From Python to Vulkan

- Data Graph Shader Module

```
// Provided by VK_ARM_data_graph
typedef struct VkDataGraphPipelineShaderModuleCreateInfoARM {
    VkStructureType                          sType;
    const void*                              pNext;
    VkShaderModule                           module;
    const char*                              pName;
    const VkSpecializationInfo*              pSpecializationInfo;
    uint32_t                                 constantCount;
    const VkDataGraphPipelineConstantARM*    pConstants;
} VkDataGraphPipelineShaderModuleCreateInfoARM;
```

SPIR-V module

Data Graph Entry Point (as per SPIR-V)

Data Graph Constants referenced in SPIR-V module

- Data Graph Constants

```
// Provided by VK_ARM_data_graph
typedef struct VkDataGraphPipelineConstantARM {
    VkStructureType    sType;
    const void*        pNext;
    uint32_t           id;
    const void*        pConstantData;
} VkDataGraphPipelineConstantARM;
```

A **VkTensorDescriptionArm** to define shape

ID to resolve SPIR-V reference

arm

Neural Network: From Python to Vulkan

- **Data Graph Pipeline Create Info**

```
// Provided by VK_ARM_data_graph
typedef struct VkDataGraphPipelineCreateInfoARM {
    VkStructureType                              sType;
    const void*                                  pNext;
    VkPipelineCreateFlags2KHR                    flags;
    VkPipelineLayout                             layout;
    uint32_t                                     resourceInfoCount;
    const VkDataGraphPipelineResourceInfoARM*    pResourceInfos;
} VkDataGraphPipelineCreateInfoARM;
```

The shader module we just created

Pipeline layout to read/write data with binding/set

Resource infos for input and output tensors

- **Resource info**

```
typedef struct VkDataGraphPipelineResourceInfoARM {
    VkStructureType    sType;
    const void*        pNext;
    uint32_t           descriptorSet;
    uint32_t           binding;
    uint32_t           arrayElement;
} VkDataGraphPipelineResourceInfoARM;
```

A **VkTensorDescriptionArm** to define tensor shape

arm

Neural Network: From Python to Vulkan

- Create **Data Graph Pipeline Session**

```
// Provided by VK_ARM_data_graph
typedef struct VkDataGraphPipelineSessionCreateInfoARM {
    VkStructureType                                 sType;
    const void*                                     pNext;
    VkDataGraphPipelineSessionCreateFlagsARM        flags;
    VkPipeline                                      dataGraphPipeline;
} VkDataGraphPipelineSessionCreateInfoARM;
```

The pipeline resource

- **Bind memory** to it

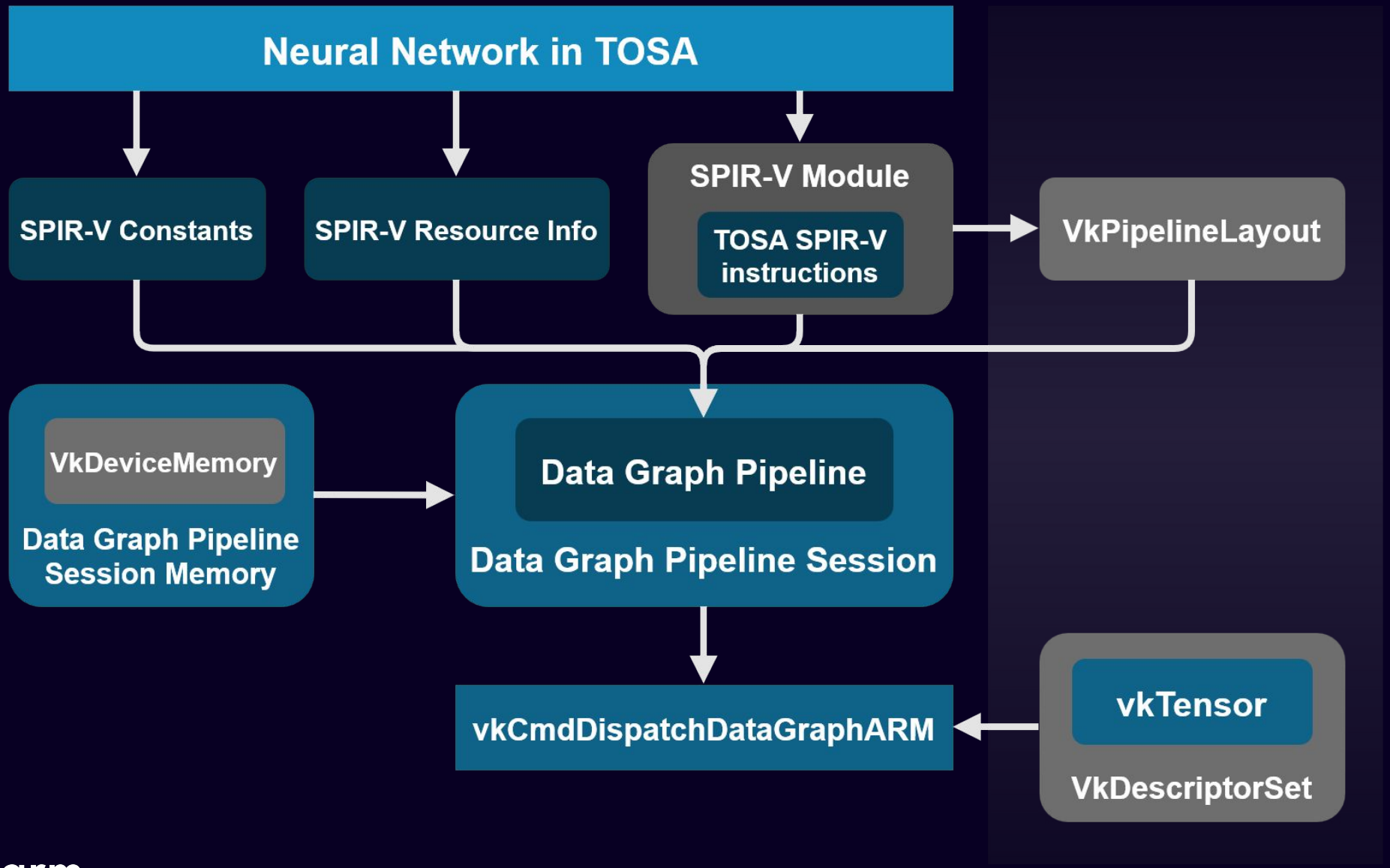
```
typedef struct VkBindDataGraphPipelineSessionMemoryInfoARM {
    VkStructureType                                 sType;
    const void*                                     pNext;
    VkDataGraphPipelineSessionARM                   session;
    VkDataGraphPipelineSessionBindPointARM          bindPoint;
    uint32_t                                        objectIndex;
    VkDeviceMemory                                  memory;
    VkDeviceSize                                    memoryOffset;
} VkBindDataGraphPipelineSessionMemoryInfoARM;
```

What this memory is for, e.g. transient tensors during dispatch

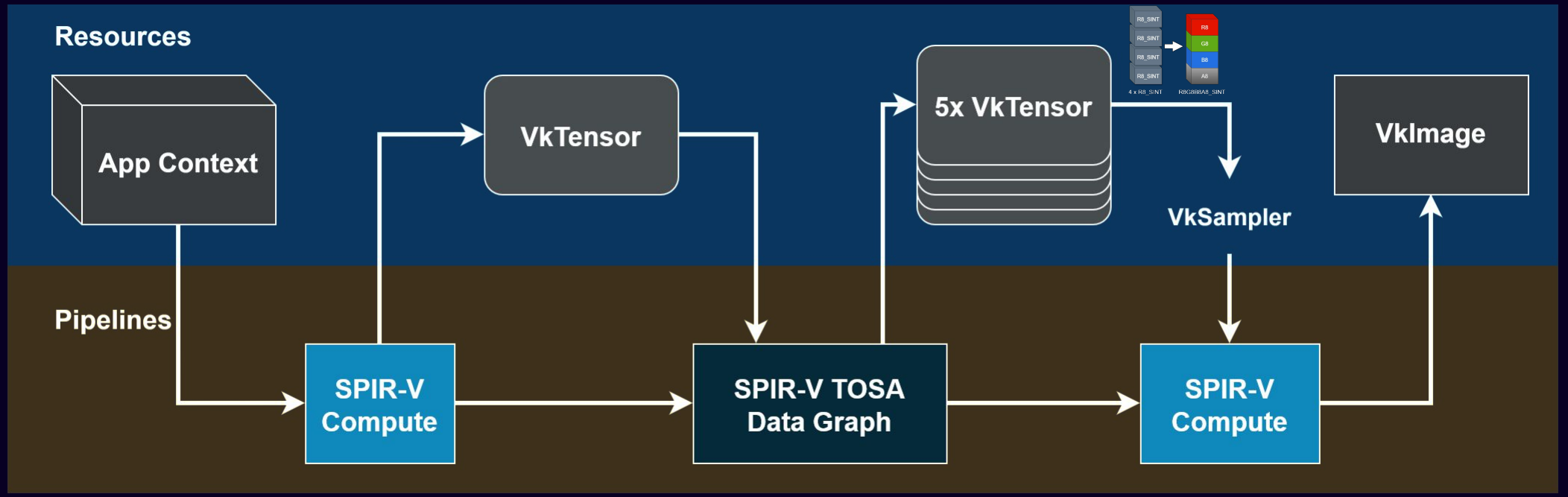
Your usual allocation

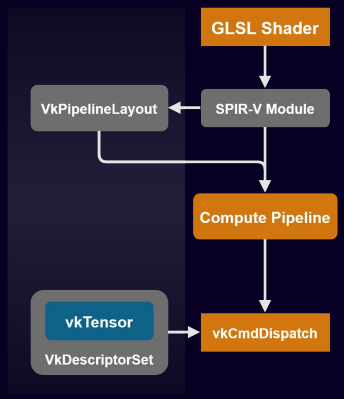
arm

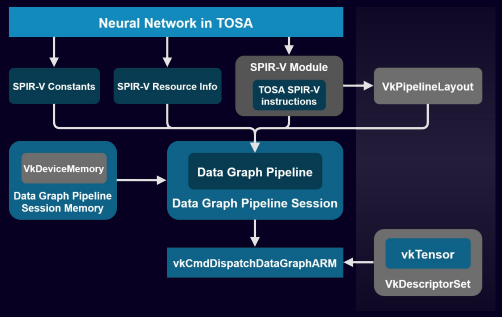
Data Graph Extension (VK_ARM_data_graph)

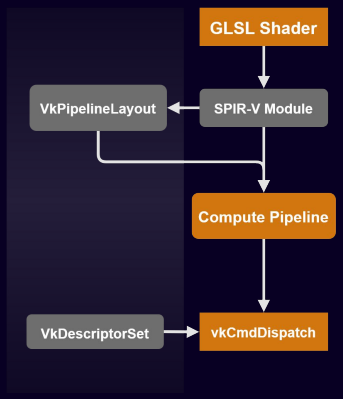


Neural Super Sampling in Vulkan









Synchronization

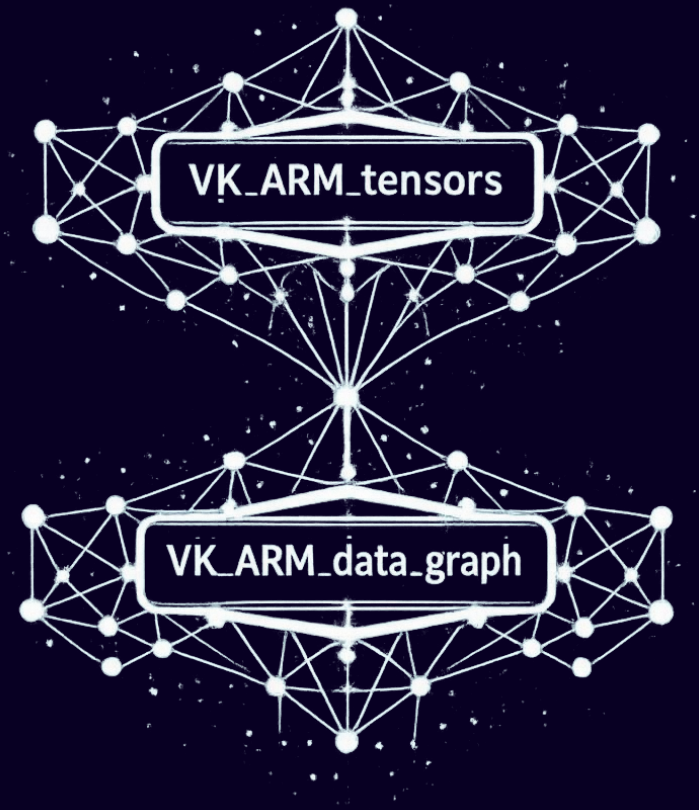
- New stages and access masks:
  - **DATA_GRAPH** stage
  - **DATA_GRAPH_READ** access
  - **DATA_GRAPH_WRITE** access
- API mechanism for tensor barriers same as **VkBufferMemoryBarrier2**

```
VkTensorMemoryBarrierARM tensorBarrier{.sType = ...
    .srcStageMask = VK_PIPELINE_STAGE_2_COMPUTE_SHADER_BIT,
    .srcAccessMask = VK_ACCESS_2_SHADER_WRITE_BIT,

    .dstStageMask = VK_PIPELINE_STAGE_2_DATA_GRAPH_BIT_ARM,
    .dstAccessMask = VK_ACCESS_2_DATA_GRAPH_READ_BIT_ARM,

    .tensor = tensor}; // your VkTensor

VkDependencyInfo dependencyInfo{.sType = ...
    .pNext = &tensorBarrier}; // attached to pNext

// Requires VK_KHR_synchronization2
vkCmdPipelineBarrier2(commandBuffer, &dependencyInfo);
```

- If data graph queue is separate to compute queue, requires command buffer synchronization

Additions to a Rendering Hardware Interface

- New resource type: **Tensor**
- New **image** layout and usage – **Tensor Aliased**
- New pipeline type for compilation: **Data Graph**
  - SPIR-V module, constants and resource infos are internal to creation fn
  - Reflection information from SPIR-V TOSA the same as for shaders
- New execution workload type: **Data Graph**
  - Bind pipeline and descriptors - Data Graph Bind point
  - Dispatched using the *session* resource – needs to persist after creation
- Synchronization
  - New resource barrier, access masks, and stages – **tensors** and **images**
- New command buffer queue type: **Data Graph**



Conclusions

- A new way to do ML inference in Vulkan, built for performance and flexibility
- Familiar patterns
- Bits this talk does not cover:
  - Quantization
  - Generating reflection information
  - Dynamic Graph shaping
- Everything mentioned in this presentation is **fully open source** under a permissive license!

Questions time!

- Get started with:
  - ML Emulation Layer for Vulkan allowing you to try the extensions on desktop
  - RenderDoc for Arm has Data Graph support
    - Compatible with Emulation Layer
  - SDK w/ the extensions
  - Tooling and many tutorials
- QR to access resources page

Links and references

https://developer.arm.com/ml-sdk-for-vulkan

https://registry.khronos.org/vulkan/specs/latest/html/vkspec.html#VK_ARM_data_graph

https://registry.khronos.org/vulkan/specs/latest/html/vkspec.html#VK_ARM_tensors

https://docs.vulkan.org/features/latest/features/proposals/VK_ARM_tensors.html

https://github.com/arm/neural-graphics-sdk-for-game-engines

https://www.mlplatform.org/tosa/

https://docs.pytorch.org/executorch/1.0/tutorial-arm-vgf.html

https://learn.arm.com/learning-paths/mobile-graphics-and-gaming/vulkan-ml-sample/1-introduction/

https://arm.github.io/ai-ml-sdk-for-vulkan/introduction.html

https://arm-software.github.io/Vulkan-Samples/samples/extensions/tensor_and_data_graph/simple_tensor_and_data_graph/README.html#_overview

arm

arm

Tack

ಧನ್ಯವಾದಗಳು

Merci

Danke

Gracias

Grazie

谢谢

ありがとう

Asante

Thank you

감사합니다

धन्यवाद

Kiitos

شكرًا

ধন্যবাদ