Tutorial Practice Session

Step 3: User Kernels and Nodes

Kernel and Node Terminology

- **KERNEL**

Implementation code of a CV primitive, generic with regard to data objects

  - C++ world : Class

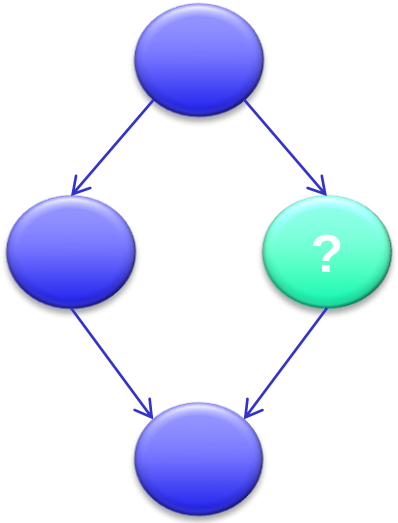
- **NODE**

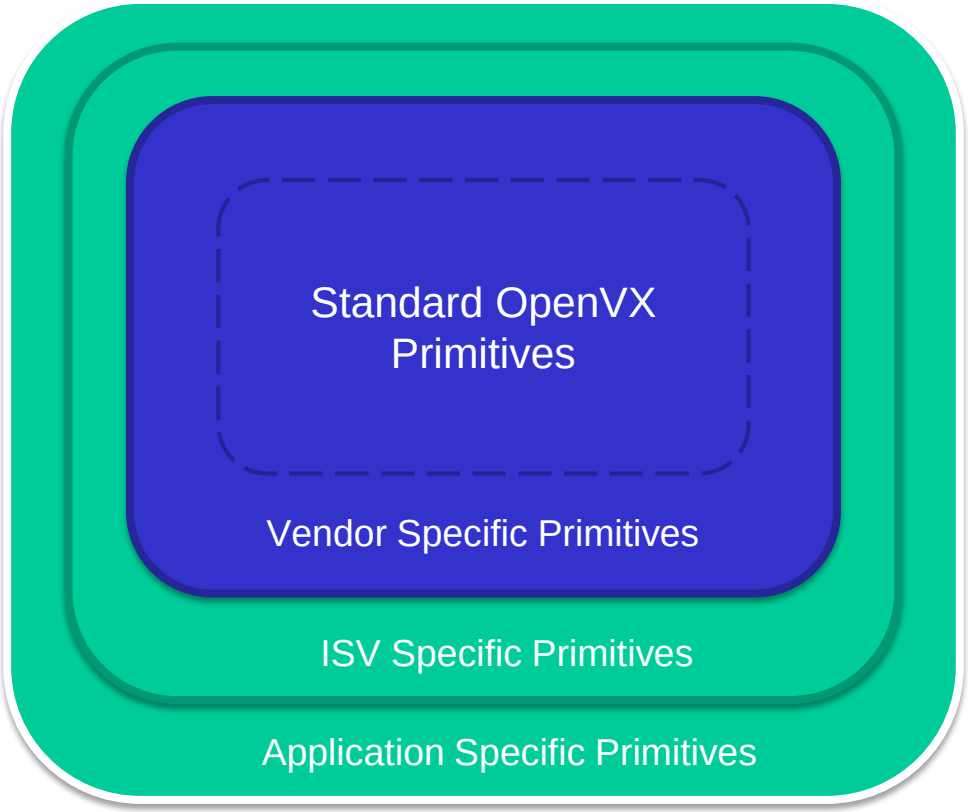
Instance of a kernel in a graph with well defined data objects parameters

  - C++ world : Object

User Kernels & Nodes, What for ?

- Extend the set of CV primitives
- Avoid breaking the execution flow and benefit from some graph optims





User Node Creation

- **Node creation**
  - Instantiate a kernel
  - Set parameter objects

- **Prerequisite : Kernel creation**
  - Kernel callback functions
  - Kernel registration in the context

Exercise Goal : Generic NxN Median Filter

- **NxN median filter user-node :**

```
vx_node userMedianBlurNode( vx_graph graph,
                            vx_image input,
                            vx_image output,
                            vx_int32 ksize );
```

- **User node as a wrapper to OpenCV medianBlur :**

```
cv::Mat mat_input ( height, width, CV_8U, ptr_input,  addr_input .stride_y );
cv::Mat mat_output( height, width, CV_8U, ptr_output, addr_output.stride_y );

cv::medianBlur( mat_input, mat_output, ksize );
```

Node Creation From a Kernel

Create a User Node

- Get a kernel reference for its ID or Name

```
vx_kernel kernel = vxGetKernelByEnum(context, USER_KERNEL_AND3);
vx_kernel kernel = vxGetKernelByName(context, “com.mycompany.and3” );
```

- Create a generic node

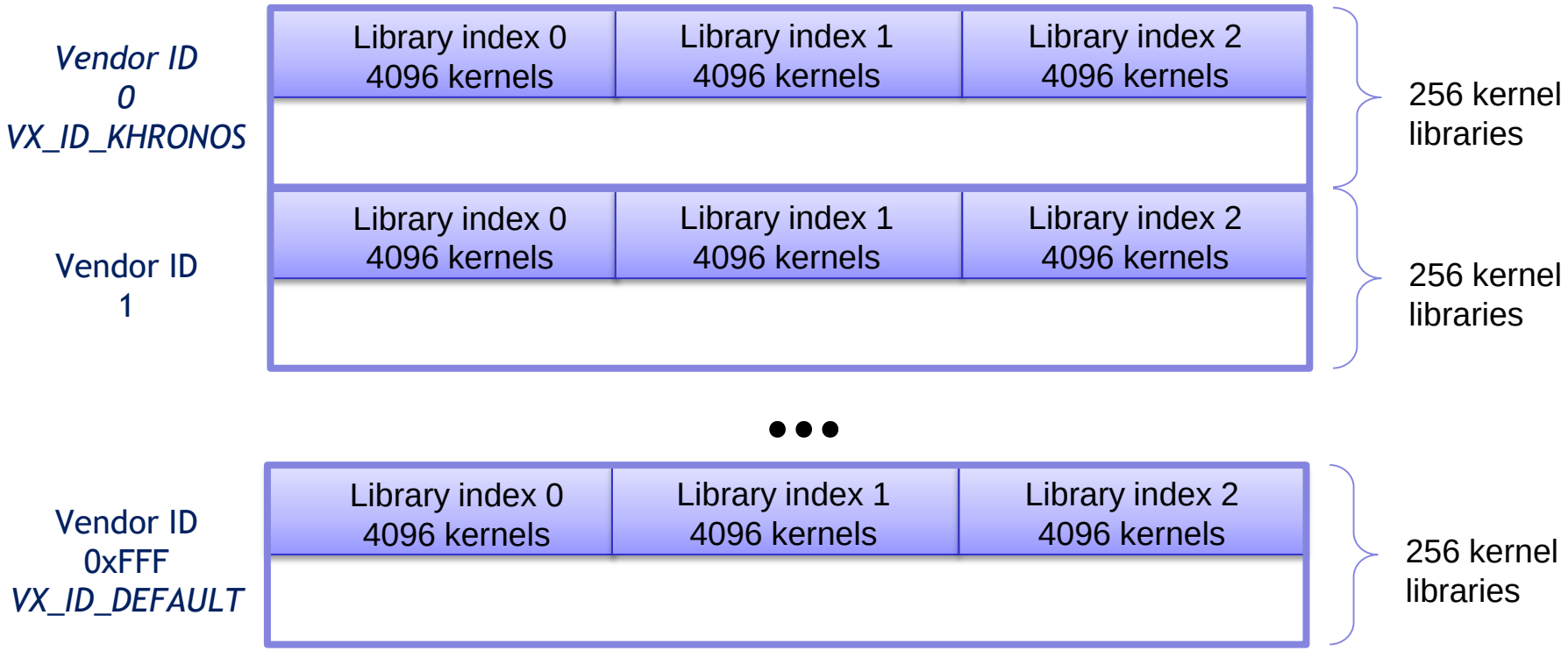
```
vx_node node = vxCreateGenericNode(graph, kernel);
```

- Set node parameters (must be a reference)

```
vxSetParameterByIndex(node, 0, (vx_reference)in0);
vxSetParameterByIndex(node, 1, (vx_reference)in1);
vxSetParameterByIndex(node, 2, (vx_reference)scalar);
vxSetParameterByIndex(node, 3, (vx_reference)out);
```

Kernel Unique ID

```
KERNEL_ID = VX_KERNEL_BASE( VENDOR_ID, LIBRARY_INDEX ) + INDEX_IN_LIB
```



Vendor IDs in vx_vendors.h

```
enum vx_vendor_id_e {
    VX_ID_KHRONOS    = 0x000, /*!< \brief The Khronos Group */
    VX_ID_TI         = 0x001, /*!< \brief Texas Instruments, Inc. */
    VX_ID_QUALCOMM   = 0x002, /*!< \brief Qualcomm, Inc. */
    VX_ID_NVIDIA     = 0x003, /*!< \brief NVIDIA Corporation */
    VX_ID_ARM        = 0x004, /*!< \brief ARM Ltd. */
    VX_ID_BDTI       = 0x005, /*!< \brief Berkley Design Technology, Inc. */
    VX_ID_RENESAS    = 0x006, /*!< \brief Renesas Electronics */

    // …

    VX_ID_DEFAULT    = 0xFFF
}
```

OpenVX 1.1 Evolution

- **OpenVX 1.0.1 : Fully manual ID allocation**

```
USER_KERNEL_AND3 = VX_KERNEL_BASE( VX_ID_DEFAULT, USER_LIBRARY_ARITH ) +  0x4
```

- **OpenVX 1.1 : Automatic ID allocation in the new VX_ID_USER vendor domain.**
  For companies that don't have a Khronos ID

```
vx_status vxAllocateUserKernelId( vx_context context, vx_enum *pKernelEnumId )
vx_status vxAllocateUserKernelLibraryId ( vx_context context, vx_enum  *pLibraryId )
```

Exercise: switch to the VM

Kernel Callbacks

User Kernels Callbacks

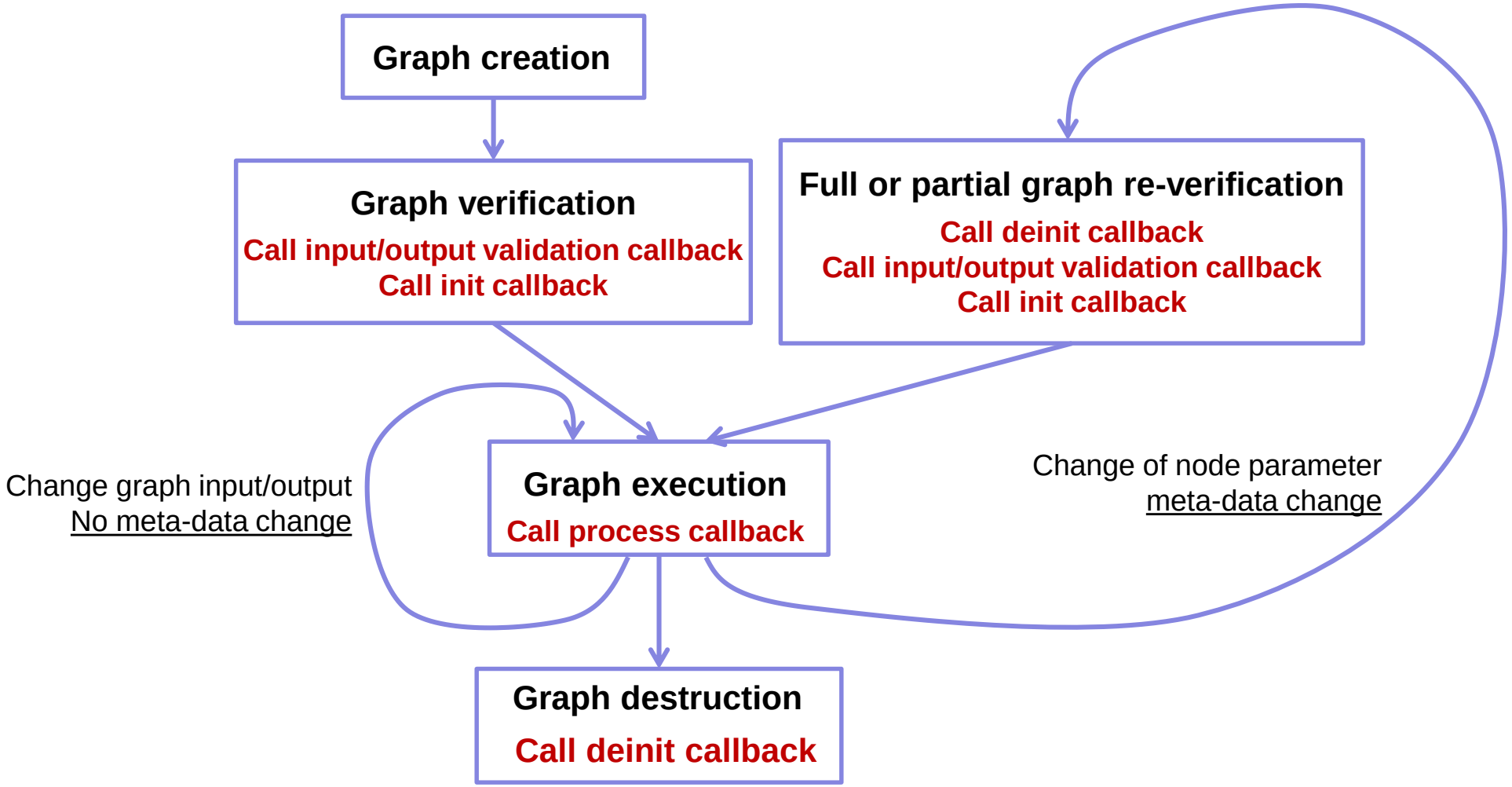
OpenVX 1.0.1

1. **Input validation function**
   Called at graph (re)verification
2. **Output validation function**
   Called at graph (re)verification
3. **Initialization function (optional)**
   Called at graph (re)verification
4. **Deinitializer function (optional)**
   Called at graph re-verification and destruction
5. **Process function (Host)**
   Called at node execution

OpenVX 1.1

1. **(unique) Param validation function**
   Called at graph (re)verification
   -> Code simplification + performance improvement
2. **Initialization function (optional)**
   Called at graph (re)verification
3. **Deinitializer function (optional)**
   Called at graph re-verification and destruction
4. **Process function (Host)**
   Called at node execution

Callback Usage: Examples



Kernel Input Validation Function

- **Prototype**

```
typedef vx_status (vx_kernel_input_validate_f)( vx_node node,
                                                vx_uint32 index );
```

- **The framework checks the object type (e.g., it is an image reference)**

- **What needs to be done in the callback**
  - Get the reference to the parameter object

```
vx_parameter param = vxGetParameterByIndex( node, index );
vx_reference input = NULL;
vxQueryParameter(param, VX_PARAMETER_ATTRIBUTE_REF, &input, sizeof(input));
```

  - Check meta-data restriction (e.g., image format)
  - Check consistency with other parameters (e.g., same image formats & dimensions)

Kernel Output Validation Function

- **Prototype**

```
typedef vx_status ( vx_kernel_output_validate_f)(vx_node node,
                    vx_uint32 index, vx_meta_format meta );
```

- **The framework checks the object type (e.g., it is an image reference)**

- **What needs to be done in the callback**

  Set the meta_format object with expected meta-data for the output (usually determined from inputs)

```
vxSetMetaFormatAttribute( meta, VX_IMAGE_ATTRIBUTE_WIDTH, &width, sizeof(width) );
vxSetMetaFormatAttribute( meta, VX_IMAGE_ATTRIBUTE_HEIGHT, &height, sizeof(height) );
vx_df_image format = VX_DF_IMAGE_U8;
vxSetMetaFormatAttribute( meta, VX_IMAGE_ATTRIBUTE_FORMAT, &format, sizeof(format) );
```

- **The framework will do the actual meta-data check**

Kernel Process Function

- Prototype

```
typedef vx_status (vx_kernel_f)( vx_node node,
                    const vx_reference * params, vx_uint32 n );
```

- The framework ensures parameters were validated

- What needs to be done in the callback :
  Actual processing : generate output data from input data

Kernel Init/Deinit Functions

1. Initializer (Optional)

```
typedef vx_status (vx_kernel_initialize_f)(vx_node node,
                                           const vx_reference * params, vx_uint32 num);
```

- Can request scratch memory to be used by the process callback (e.g., temporary buffer)

```
vx_size size = ...;
vxSetNodeAttribute(node, VX_NODE_ATTRIBUTE_LOCAL_DATA_SIZE, &size, sizeof(size));
```

- Can allocate and initialize some memory to be used by the process callback (e.g., lookup table)

```
MyData * ptr = new MyData;
vxSetNodeAttribute(node, VX_NODE_ATTRIBUTE_LOCAL_DATA_PTR, &ptr, sizeof(ptr));
```

2. Deinitializer (Optional)

```
typedef vx_status (vx_kernel_deinitialize_f)(vx_node node,
                                             const vx_reference * params, vx_uint32 num);
```

- De-allocate memory allocated at initialization

```
vxQueryNode(node, VX_NODE_ATTRIBUTE_LOCAL_DATA_PTR, &ptr, sizeof(ptr));
delete ptr;
```

Exercise: switch to the VM

User Kernel Registration

- Kernel object creation

```
vx_kernel kernel = vxAddKernel(context,
                               "user.kernel.and3", USER_KERNEL_AND3,
                               And3Process,
                               4, // NumParams
                               And3InputValidator,
                               And3OutputValidator,
                               NULL, NULL // Init/Deinit
                               );
```

- Describe kernel parameters

```
vxAddParameterToKernel(kernel, 0, VX_INPUT, VX_TYPE_IMAGE,
                       VX_PARAMETER_STATE_REQUIRED);
…
vxAddParameterToKernel(kernel, 3, VX_OUTPUT, VX_TYPE_IMAGE,
                       VX_PARAMETER_STATE_REQUIRED);
```

- Finalize the kernel creation

```
vxFinalizeKernel(kernel);
```

User Kernel Registration

- OpenVX 1.0.1

```
// Create a kernel object
vx_kernel kernel = vxAddKernel(context,
                               "user.kernel.and3", USER_KERNEL_AND3,
                               And3Process,
                               4, // NumParams
                               And3InputValidator,
                               And3OutputValidator,
                               NULL, NULL // Init/Deinit
                               );
```

- OpenVX 1.1

```
// Create a kernel object
vx_kernel kernel = vxAddUserKernel(context,
                               "user.kernel.and3", USER_KERNEL_AND3,
                               And3Process,
                               4, // NumParams
                               And3ParameterValidator,
                               NULL, NULL // Init/Deinit
                               );
```

Complete the Exercise