Changes

v1.0.1 to v1.1

High Level Objectives

- **Improving ease-of-use of application interface**
  - Easier and less error prone methods to read and write data to and from OpenVX objects
  - More convenient and flexible API for extending OpenVX with user kernels
  - Many other improvements and clarifications to infrastructure functions and vision kernels.

- **Computational Photography Use Cases**
  - Adding Laplacian pyramids processing kernels
  - Replicate node for simplifying graph construction with pyramids

- **Adding control and functional flexibility**
  - Adding mechanism for assigning a node execution to a specific “target” within a heterogeneous device.
  - New median, erosion, and dilation image filters, with programmable structuring element sizes and patterns.

Object Data Access (1/3) : Version 1.0

- **Access limited in time**
  - vxAccessImagePatch: get access (Read, Write, Read & Write)
  - vxCommitImagePatch: release the access
- **Two modes**
  - MAP: OpenVX controls *address* and *memory layout*

```
void * ptr = NULL;
vx_imagepatch_addressing_t addr;
vx_rectangle_t rect = { 0u, 0u, width, height };
vxAccessImagePatch( img, &rect, plane, &addr, &ptr, VX_READ_AND_WRITE );
// Access data in ptr
vxCommitImagePatch( img, &rect, plane, &addr, ptr );
```

  - COPY: The application controls *address* and *memory layout*

```
void * ptr = &my_array[0];
vx_imagepatch_addressing_t addr = { /* to fill */ };
vx_rectangle_t rect = { 0u, 0u, width, height };
vxAccessImagePatch( img, &rect, plane, &addr, &ptr, VX_READ_AND_WRITE );
// Access data in my_array
vxCommitImagePatch( img, &rect, plane, &addr, ptr );
```

Object Data Access (2/3) : Version 1.1

- **Remove Access/Commit functions; added separate functions for each mode**
  - **Map:** OpenVX controls *address* and *memory layout*
    - vxMapImagePatch: get access (Read, Write, Read & Write)
    - vxUnmapImagePatch: release the access

```
void * ptr = NULL;
vx_imagepatch_addressing_t addr;
vx_rectangle_t rect = { 0u, 0u, width, height };
vxMapImagePatch( img, &rect, plane, &map_id, &addr, &ptr,
                 VX_READ_AND_WRITE, VX_MEMORY_TYPE_HOST, 0);
// Access data in ptr
vxUnmapImagePatch( img, map_id);
```

  - **Copy**: The application controls *address* and *memory layout*
    - vxCopyImagePatch: Transfer data between OpenVX memory and application memory

```
void * ptr = &my_array[0];
vx_imagepatch_addressing_t addr = { /* to fill */ };
vx_rectangle_t rect = { 0u, 0u, width, height };
vxCopyImagePatch( img, &rect, plane, &addr, &ptr,
                  VX_WRITE_ONLY, VX_MEMORY_TYPE_HOST);
```

Object Data Access (3/3) : Scope of Change

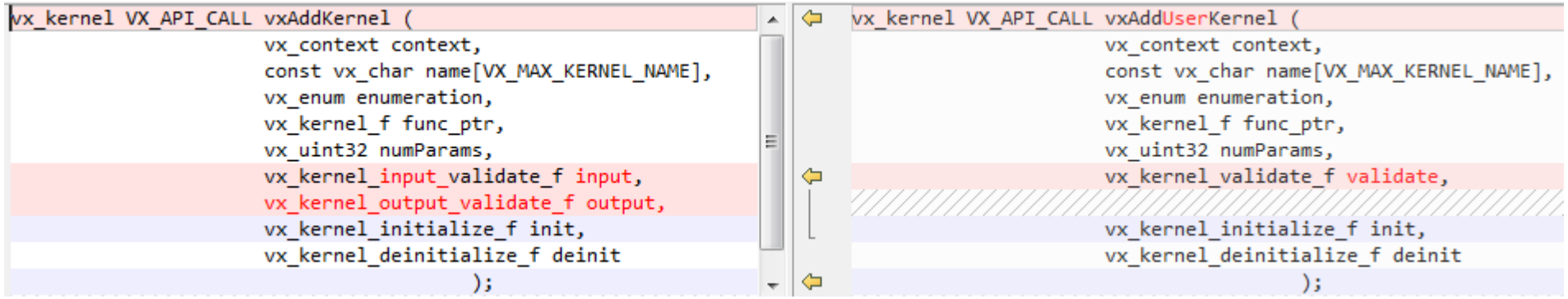
- **Scope of change**
  - Perhaps most obvious change in application interface for existing code (vx_compatability.h)
  - All data objects except vx_pyramid, vx_remap, and vx_threshold are affected by at least one of the 2 changes below:

- **Access/Commit Functions -> Map/Unmap/Copy**
  - vx_array, vx_distribution, vx_image, vx_lut

- **Read/Write Functions -> Copy**
  - vx_convolution, vx_matrix, vx_scalar

User Kernel Simplification

- **vxAddKernel has been replaced with vxAddUserKernel**



```
vx_kernel VX_API_CALL vxAddKernel (
                    vx_context context,
                    const vx_char name[VX_MAX_KERNEL_NAME],
                    vx_enum enumeration,
                    vx_kernel_f func_ptr,
                    vx_uint32 numParams,
                    vx_kernel_input_validate_f input,
                    vx_kernel_output_validate_f output,
                    vx_kernel_initialize_f init,
                    vx_kernel_deinitialize_f deinit
                                        );
```

```
vx_kernel VX_API_CALL vxAddUserKernel (
                    vx_context context,
                    const vx_char name[VX_MAX_KERNEL_NAME],
                    vx_enum enumeration,
                    vx_kernel_f func_ptr,
                    vx_uint32 numParams,
                    vx_kernel_validate_f validate,
                    vx_kernel_initialize_f init,
                    vx_kernel_deinitialize_f deinit
                                        );
```

- **Combined input and output validation callbacks into single callback function**
  - Added parameters[] array to this new callback

```
typedef vx_status( vx_kernel_validate_f )(vx_node node, const vx_reference
        parameters[ ], vx_uint32 num, vx_meta_format metas[])
```

- **Results: Significantly reduces validation code and execution time**
  - Remove need to get/query/release each vx_parameter internally
  - Removed redundancies in code between input/output validators for independent parameter checking

```
static vx_status VX_CALLBACK vxAbsDiffInputValidator(vx_node node, vx_uint32 index) {
    vx_status status = VX_ERROR_INVALID_PARAMETERS;
    if (index == 0 ) {
        vx_image input = 0;
        vx_parameter param = vxGetParameterByIndex(node, index);
        vxQueryParameter(param, VX_PARAMETER_ATTRIBUTE_REF, &input, sizeof(input));
        if (input) {
            vx_df_image format = 0;
            vxQueryImage(input, VX_IMAGE_ATTRIBUTE_FORMAT, &format, sizeof(format));
            if (format == VX_DF_IMAGE_U8 || format == VX_DF_IMAGE_S16 )
                status = VX_SUCCESS;
            vxReleaseImage(&input);
        }
        vxReleaseParameter(&param);
    }
    else if (index == 1) {
        vx_image images[2];
        vx_parameter param[2] = {
            vxGetParameterByIndex(node, 0),
            vxGetParameterByIndex(node, 1),
        };
        vxQueryParameter(param[0], VX_PARAMETER_ATTRIBUTE_REF, &images[0], sizeof(images[0]));
        vxQueryParameter(param[1], VX_PARAMETER_ATTRIBUTE_REF, &images[1], sizeof(images[1]));
        if (images[0] && images[1]) {
            vx_uint32 width[2], height[2];
            vx_df_image format[2];
            vxQueryImage(images[0], VX_IMAGE_ATTRIBUTE_WIDTH, &width[0], sizeof(width[0]));
            vxQueryImage(images[1], VX_IMAGE_ATTRIBUTE_WIDTH, &width[1], sizeof(width[1]));
            vxQueryImage(images[0], VX_IMAGE_ATTRIBUTE_HEIGHT, &height[0], sizeof(height[0]));
            vxQueryImage(images[1], VX_IMAGE_ATTRIBUTE_HEIGHT, &height[1], sizeof(height[1]));
            vxQueryImage(images[0], VX_IMAGE_ATTRIBUTE_FORMAT, &format[0], sizeof(format[0]));
            vxQueryImage(images[1], VX_IMAGE_ATTRIBUTE_FORMAT, &format[1], sizeof(format[1]));
            if (width[0] == width[1] && height[0] == height[1] && format[0] == format[1])  {
                status = VX_SUCCESS;
            }
            vxReleaseImage(&images[0]);
            vxReleaseImage(&images[1]);
        }
        vxReleaseParameter(&param[0]);
        vxReleaseParameter(&param[1]);
    }
    return status;
}
```

Useful code

Use of vx_parameter

Redundency due to independent checking

Useless code due to independent checking

```
static vx_status VX_CALLBACK vxAbsDiffOutputValidator(vx_node node, vx_uint32 index, vx_meta_format_t *ptr)
{
    vx_status status = VX_ERROR_INVALID_PARAMETERS;
    if (index == 2) {
        vx_parameter param[2] = {
            vxGetParameterByIndex(node, 0),
            vxGetParameterByIndex(node, 1),
        };
        if (param[0] && param[1]) {
            vx_image images[2];
            vxQueryParameter(param[0], VX_PARAMETER_ATTRIBUTE_REF, &images[0], sizeof(images[0]));
            vxQueryParameter(param[1], VX_PARAMETER_ATTRIBUTE_REF, &images[1], sizeof(images[1]));
            if (images[0] && images[1])
            {
                vx_uint32 width[2], height[2];
                vx_df_image format = 0;
                vxQueryImage(images[0], VX_IMAGE_ATTRIBUTE_FORMAT, &format, sizeof(format));
                vxQueryImage(images[0], VX_IMAGE_ATTRIBUTE_WIDTH, &width[0], sizeof(width[0]));
                vxQueryImage(images[1], VX_IMAGE_ATTRIBUTE_WIDTH, &width[1], sizeof(width[1]));
                vxQueryImage(images[0], VX_IMAGE_ATTRIBUTE_HEIGHT, &height[0], sizeof(height[0]));
                vxQueryImage(images[1], VX_IMAGE_ATTRIBUTE_HEIGHT, &height[1], sizeof(height[1]));
                if (width[0] == width[1] && height[0] == height[1] &&
                    (format == VX_DF_IMAGE_U8 || format == VX_DF_IMAGE_S16))
                {
                    ptr->type = VX_TYPE_IMAGE;
                    ptr->dim.image.format = format;
                    ptr->dim.image.width = width[0];
                    ptr->dim.image.height = height[1];
                    status = VX_SUCCESS;
                }
                vxReleaseImage(&images[0]);
                vxReleaseImage(&images[1]);
            }
            vxReleaseParameter(&param[0]);
            vxReleaseParameter(&param[1]);
        }
    }
    return status;
}
```

Useful code

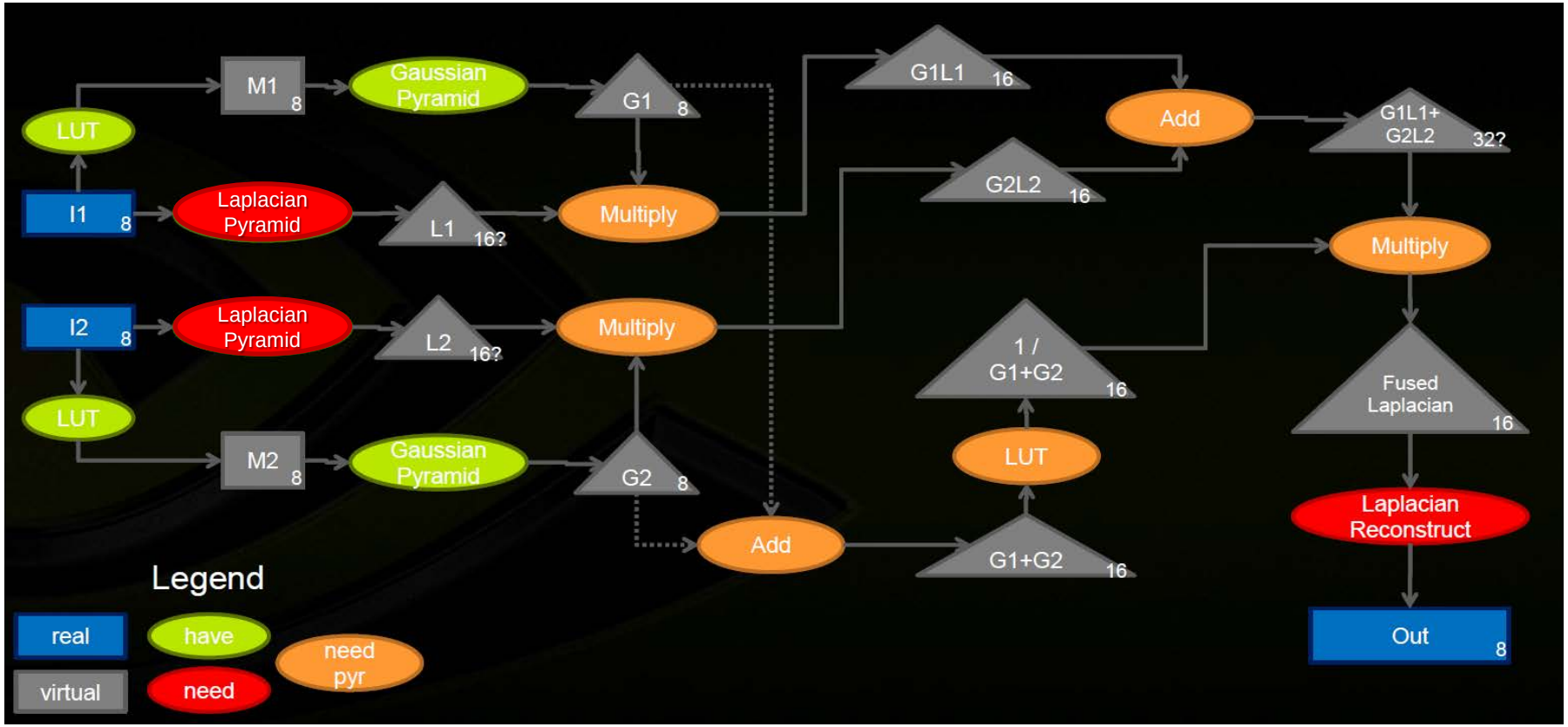
Use of vx_parameter

Redundency due to independent checking

Useless code due to independent checking

Computational Photography Use Case (1/3)

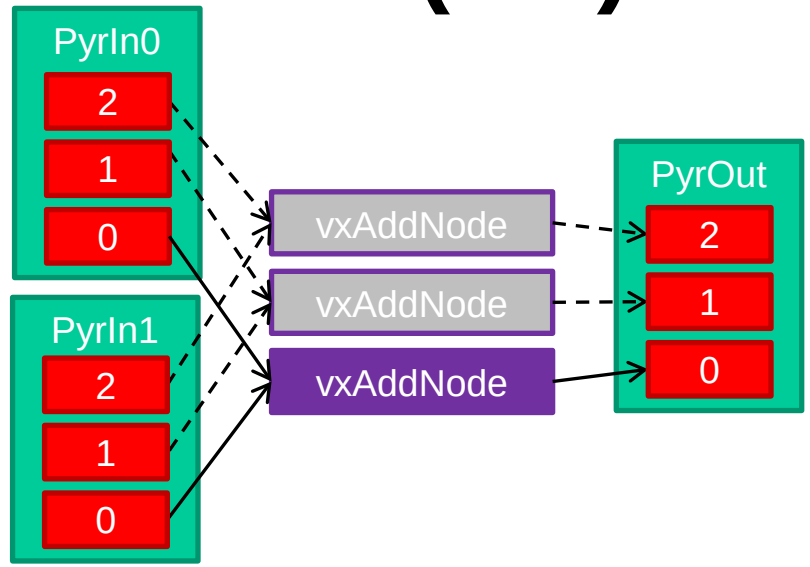
- **Exposure Fusion Graph (example of fusing 2 images, but can be scaled to N)**



- **Need to add Laplacian Pyramid and Laplacian Reconstruct**
- **Nice to have pyramid processing versions of Multiply/Add/LUT**

Computational Photography Use Case (2/3)

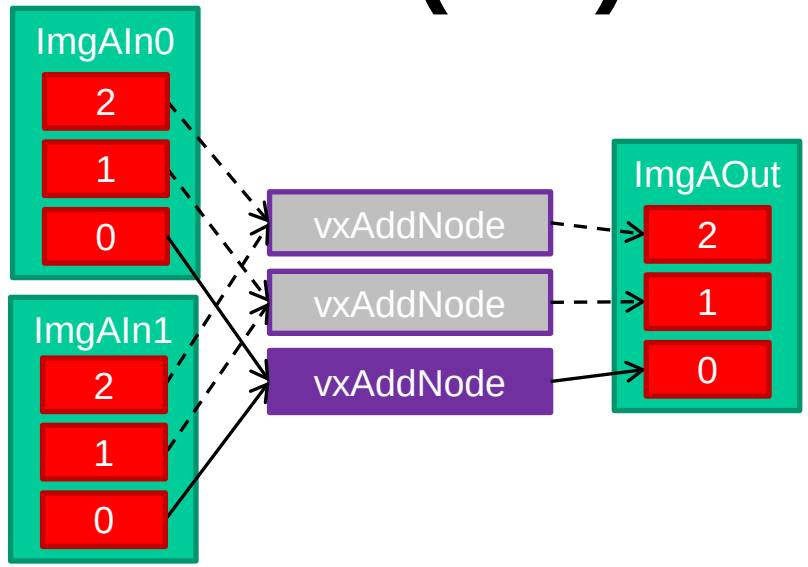
- **Add vxReplicateNode**
  - Conceptually equivalent to replicating a node functionality across all images within connected pyramids.
  - Works with ANY node.
  - Replicate flags indicate which parameters should be replicated.




```
vx_pyramid pyrIn0 = vxCreatePyramid( context, 3, VX_SCALE_PYRAMID_HALF, w, h, VX_DF_IMAGE_U8);
vx_pyramid pyrIn1 = vxCreatePyramid( context, 3, VX_SCALE_PYRAMID_HALF, w, h, VX_DF_IMAGE_U8);
vx_pyramid pyrOut = vxCreatePyramid( context, 3, VX_SCALE_PYRAMID_HALF, w, h, VX_DF_IMAGE_U8);

vx_image firstIn0 = vxGetPyramidLevel( pyrIn0, 0);
vx_image firstIn1 = vxGetPyramidLevel( pyrIn1, 0);
vx_image firstOut = vxGetPyramidLevel( pyrOut, 0);

vx_node nAdd = vxAddNode(graph, firstIn0, firstIn1, X_CONVERT_POLICY_SATURATE, firstOut);

vx_bool replicate[] = {vx_true_e, vx_true_e, vx_false_e, vx_true_e};

vxReplicateNode(graph, nAdd, replicate, 4);
```

Computational Photography Use Case (3/3)

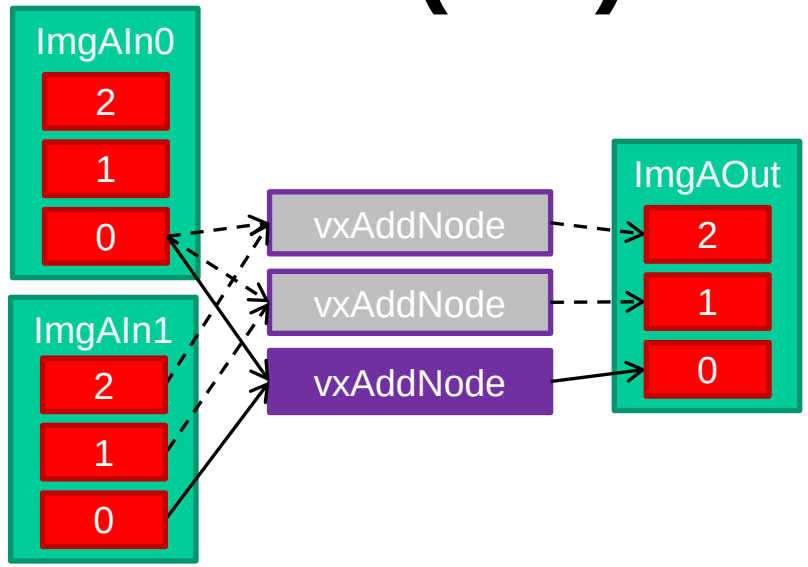
- **Support functionality for vxReplicateNode**
  - New attributes:
    - -VX_NODE_PARAMETERS
    - -VX_NODE_IS_REPLICATED
    - -VX_NODE_REPLICATE_FLAGS
- **Add vx_object_array**
  - Similar to pyramid, except can be arrays of any object type (except delay and object arrays)




```
vx_image exemplar = vxCreateImage( context, w, h, VX_DF_IMAGE_U8);
vx_object_array imgAIn0 = vxCreateObjectArray( context, (vx_reference)exemplar, 3);
vx_object_array imgAIn1 = vxCreateObjectArray( context, (vx_reference)exemplar, 3);
vx_object_array imgAOut = vxCreateObjectArray( context, (vx_reference)exemplar, 3);

vx_image firstIn0 = vxGetObjectArrayItem(imgAIn0, 0);
vx_image firstIn1 = vxGetObjectArrayItem(imgAIn1, 0);
vx_image firstOut = vxGetObjectArrayItem(imgAOut, 0);

vx_node nAdd = vxAddNode(graph, firstIn0, firstIn1, X_CONVERT_POLICY_SATURATE, firstOut);

vx_bool replicate[] = {vx_true_e, vx_true_e, vx_false_e, vx_true_e};

vxReplicateNode(graph, nAdd, replicate, 4);
```

Computational Photography Use Case (3/3)

- **Support functionality for vxReplicateNode**
  - New attributes:
    - -VX_NODE_PARAMETERS
    - -VX_NODE_IS_REPLICATED
    - -VX_NODE_REPLICATE_FLAGS
- **Add vx_object_array**
  - Similar to pyramid, except can be arrays of any object type (except delay and object arrays)




```
vx_image exemplar = vxCreateImage( context, w, h, VX_DF_IMAGE_U8);
vx_object_array imgAIn0 = vxCreateObjectArray( context, (vx_reference)exemplar, 3);
vx_object_array imgAIn1 = vxCreateObjectArray( context, (vx_reference)exemplar, 3);
vx_object_array imgAOut = vxCreateObjectArray( context, (vx_reference)exemplar, 3);

vx_image firstIn0 = vxGetObjectArrayItem(imgAIn0, 0);
vx_image firstIn1 = vxGetObjectArrayItem(imgAIn1, 0);
vx_image firstOut = vxGetObjectArrayItem(imgAOut, 0);

vx_node nAdd = vxAddNode(graph, firstIn0, firstIn1, X_CONVERT_POLICY_SATURATE, firstOut);

vx_bool replicate[] = {vx_false_e, vx_true_e, vx_false_e, vx_true_e};

vxReplicateNode(graph, nAdd, replicate, 4);
```

Kernels

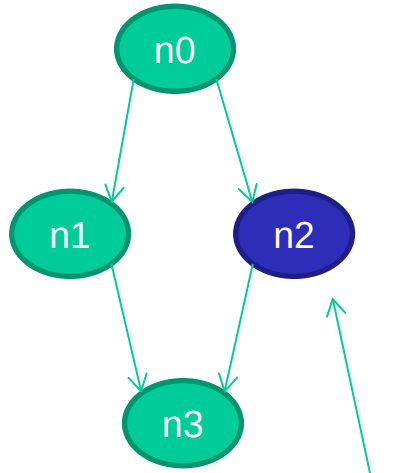
- **New**
  - vxLaplacianPyramidNode
  - vxLaplacianReconstructNode
  - vxNonLinearFilterNode
    - Median, Erosion, Dilation with programmable filter sizes and patterns
- **Updated**
  - vxTableLookupNode
    - Add S16 support for input/lut/output
  - vxThresholdNode
    - Uses programmable VX_THRESHOLD_TRUE_VALUE and VX_THRESHOLD_FALSE_VALUE for output instead of fixed 0 and 255
  - vxCannyEdgeDetectorNode
    - Clarification that hyst type can be U8 or S16, and ignores TRUE and FALSE value of the hyst type when writing the output.

Targets

- **Add vxSetNodeTarget**
  - Specifies which target to run a particular node

- **Add target specification interface**
  - VX_TARGET_ANY
  - VX_TARGET_STRING
  - VX_TARGET_VENDOR_BEGIN

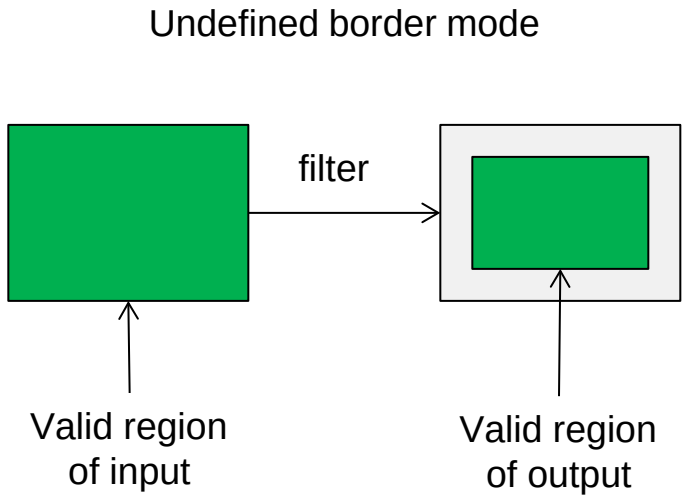
- **Add *vxSetImmediateModeTarget***



vxSetNodeTarget(n2, VX_TARGET_STRING, “HWA”);

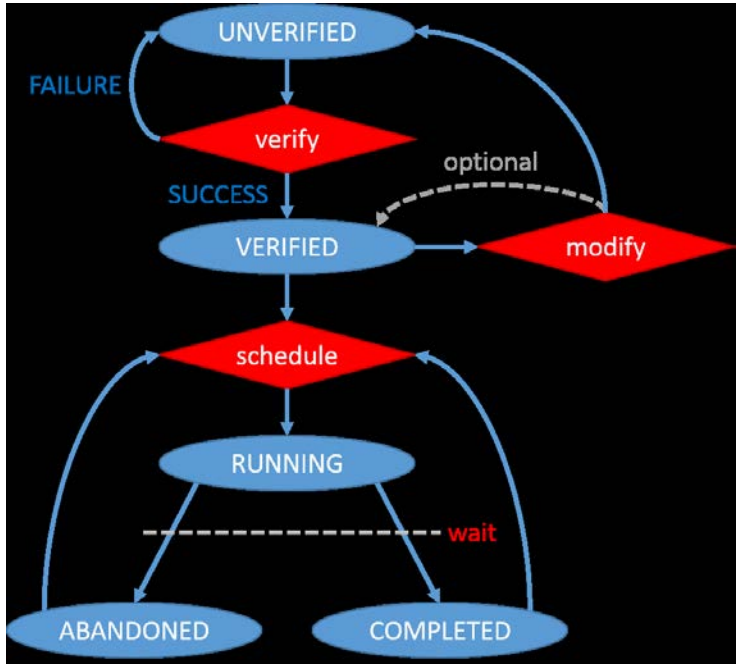
Valid Image Region

- **Added detailed behavioral description and semantic for valid image concept for image processing functions**
- **Removed vx_delta_rectangles_t**
- **Add vx_border_policy_e**



Graph State

- **Add VX_GRAPH_STATE attribute**
  - Returns type **vx_graph_state_e**
  - User can query the state of the graph, as defined by state machine.



Hints/Directives/Enums

- **Hints**
  - Added Performance hints (add to vx_hint_e enumerations)
    - VX_HINT_PERFORMANCE_DEFAULT
    - VX_HINT_PERFORMANCE_LOW_POWER
    - VX_HINT_PERFORMANCE_HIGH_SPEED
  - Removed Serialize hint
    - VX_HINT_SERIALIZE
- **Directives**
  - Added enable/disable performance reporting
    - VX_DIRECTIVE_DISABLE_PERFORMANCE
    - VX_DIRECTIVE_ENABLE_PERFORMANCE
- **Enums**
  - Change many of the enum names to help reduce their length for MISRAC conformance (namely removing ATTRIBUTE_ portion of names)
    - First 31 characters of identifiers to be unique

Summary of Changes (1/3)

- **Kernels:**
  - New
    - Non-linear Filter
    - Laplacian Image Pyramid
    - Reconstruction from a Laplacian Image Pyramid
  - Updated
    - Table Lookup
      - Add S16 support
    - Threshold
      - Uses programmable VX_THRESHOLD_TRUE_VALUE and VX_THRESHOLD_FALSE_VALUE for output instead of fixed 0 and 255
    - Canny Edge Detector
      - Clarification that hyst type can be U8 or S16, and ignores TRUE and FALSE value of the hyst type when writing the output.
- **Objects:**
  - New
    - Object Array: vx_object_array
  - Updated
    - Reference Object: vx_reference:
      - Add *vxReleaseReference*
      - Add *vxRetainReference*
      - Add *vxSetReferenceName*
- **Objects:**
  - Updated
    - Context: vx_context
      - Add *vxSetImmediateModeTarget*
      - **vx_import_type_e** -> **vx_memory_type_e**
    - Graph: vx_graph
      - Add VX_GRAPH_STATE attribute (returns type **vx_graph_state_e**) so user can query the state of the graph, as defined by state machine.
      - Add *vxRegisterAutoAging* (registers a delay object for auto aging)
    - Node: vx_node
      - Add *vxSetNodeTarget*
      - Add *vxReplicateNode*
    - Array: vx_array
      - *vxAccessArrayRange/vxCommitArrayRange -> vxMapArrayRange/vxUnmapArrayRange/vxCopyArrayRange*
    - Convolution: vx_convolution
      - *vxReadConvolutionCoefficients/vxWriteConvolutionCoefficients -> vxCopyConvolutionCoefficients*
    - Distribution: vx_distribution
      - *vxAccessDistribution/vxCommitDistribution -> vxMapDistribution/vxUnmapDistribution/vxCopyDistribution*

Summary of Changes (2/3)

- **Objects:**
  - Updated
    - Image: vx_image
      - *vxAccessImagePatch/ vxCommitImagePatch -> vxMapImagePatch/vxUnmapImagePatch/vxCopyImagePatch*
      - Add *vxSwapImageHandle*
      - Add union vx_pixel_value_t
      - Add **vx_map_flag_e**
    - LUT: vx_lut
      - *vxAccessLUT/ vxCommitLUT -> vxMapImageLUT/vxUnmapLUT/vxCopyLUT*
      - Add int16 support
      - Add VX_LUT_OFFSET attribute to support negative range in INT16 type
    - Matrix: vx_matrix
      - *vxReadMatrix/vxWriteMatrix -> vxCopyMatrix*
      - Add *vxCreateMatrixFromPattern*
    - Scalar: vx_scalar
      - *vxReadScalarValue/vxWriteScalarValue -> vxCopyScalarValue*
    - Threshold: vx_threshold
      - Data type no longer limited to U8
  - No change
    - Pyramid: vx_pyramid
    - Remap: vx_remap

- **User Kernel Support:**
  - Functions
    - Rename vxAddKernel -> vxAddUserKernel
    - Add *vxAllocateUserKernelId*
    - Add *vxAllocateUserKernelLibraryId*
    - Add *vxUnloadKernels*
    - Add *vxSetMetaFormatFromReference*
  - Callbacks
    - Add *vx_unpublish_kernels_f* callback
    - Add *vx_kernel_image_valid_rectangle_f* callback
    - Combine *vx_kernel_input_validate_f* and *vx_kernel_output_validate_f -> vx_kernel_validate_f* callback
  - Enums
    - vx_meta_format_attribute_e -> vx_meta_valid_rect_attribute_e

Summary of Changes (3/3)

- **Framework Concepts**
  - Valid image region
    - Removed vx_delta_rectangles_t
    - Added detailed behavioral description for valid image concept for image processing functions.
    - Add vx_border_policy_e
  - Targets
    - Added target specification interface
      - VX_TARGET_ANY
      - VX_TARGET_STRING
      - VX_TARGET_VENDOR_BEGIN
  - Hints
    - Added Performance hints (add to vx_hint_e enumerations)
      - VX_HINT_PERFORMANCE_DEFAULT
      - VX_HINT_PERFORMANCE_LOW_POWER
      - VX_HINT_PERFORMANCE_HIGH_SPEED
    - Removed Serialize hint
      - VX_HINT_SERIALIZE
  - Directives
    - Added enable/disable performance reporting
      - VX_DIRECTIVE_DISABLE_PERFORMANCE
      - VX_DIRECTIVE_ENABLE_PERFORMANCE

- **Miscellaneous**
  - Bug fixes and clarifications throughout
  - Change many of the enum names to help reduce length of them for MISRAC conformance (namely removing ATTRIBUTE_ portion of names).