Image Special Interest Group

Session 2
September 21, 2023

The Image SIG discussion rules

Meeting will be recorded. If you have concerns about this, let us know.

DO NOT share any confidential information or trade secrets with the group

DO keep the discussion at a High Level

- Focus on the specific Agenda topics
- We are asking for feedback on features for the oneIPL specification (e.g., requirements for functionality and performance)
- We are NOT asking for the feedback on any implementation details

Please submit the feedback in writing on GitHub per Contribution Guidelines at spec.oneapi.io. This will allow Intel to further upstream your feedback to other standards bodies, including The Khronos Group SYCL specification.

oneIPL – oneAPI interface for image processing

- Built with SYCL 2020 – based on C++17
- oneIPL provides C++ abstraction over image data, mapping to the most accelerated memory available for format and data types
- oneIPL provides SYCL API for image processing functionality on XPU

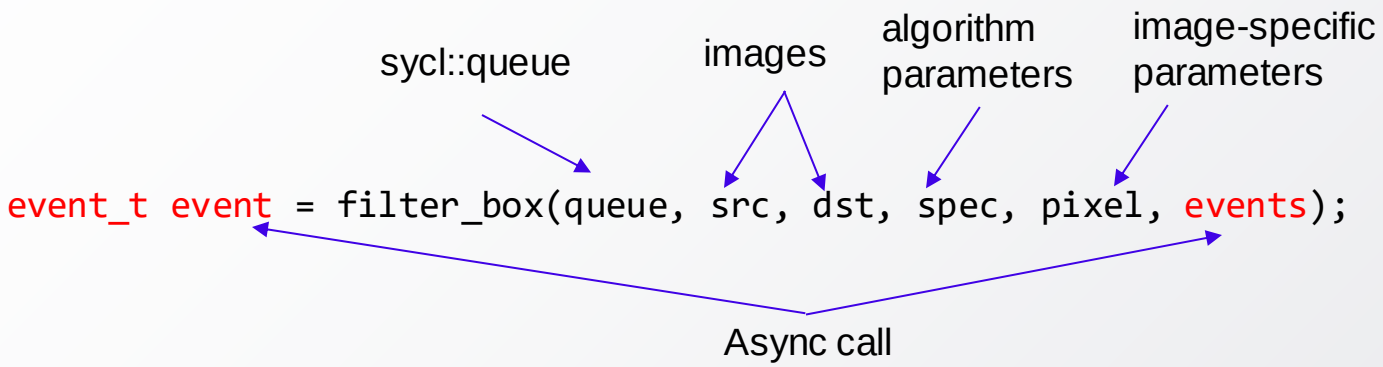
- The oneIPL provisional spec
  - v0.6 has been published
  - v0.7 in progress

oneIPL – oneAPI interface for image processing

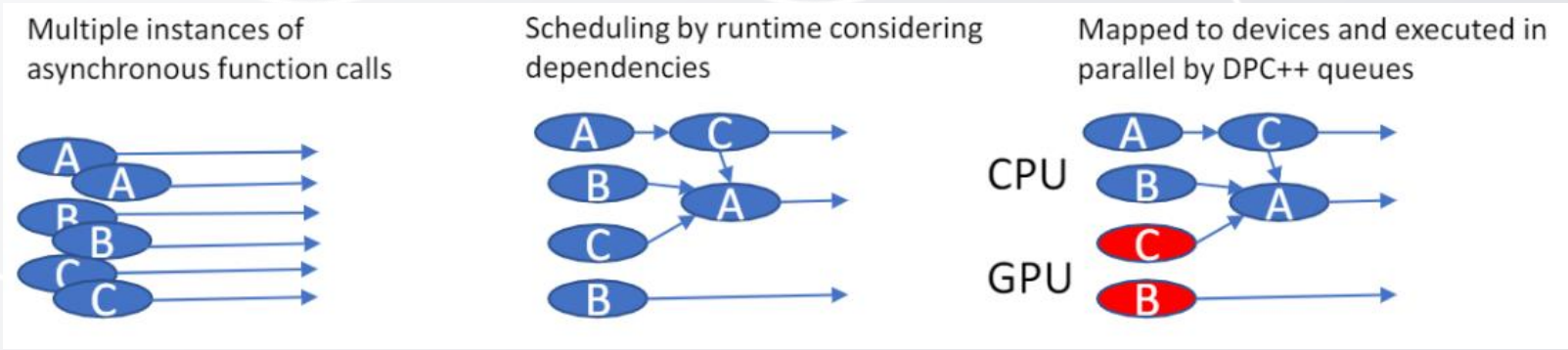
- The src and dst arguments are of class Image type and define either an image or a region of interest (ROI)
- The spec argument defines Image-independent parameters (e.g., filter kernel size or scalar factors)
- Image-dependent arguments follow before dependencies (e.g., border_val defines the border pixel value for constant borders)
- The dependencies argument defines a vector of kernel events to be completed prior to execution

```
template <typename ComputeT = float,
          typename SrcImageT,
          typename DstImageT>
sycl::event filter_box(sycl::queue&                          queue,
                       SrcImageT&                            src,
                       DstImageT&                            dst,
                       const filter_box_spec<ComputeT>&      spec,
                       const typename SrcImageT::pixel_t&    border_val   = {},
                       const std::vector<sycl::event>&       dependencies = {})
```

oneIPL – oneAPI interface for image processing



Uses the sycl::queue to construct processing pipelines that target any xPU devices available in the system. Function calls are asynchronous and are scheduled by device runtime.



oneIPL spec updates plan

Spec v0.6 (2022)

Transformations:
- Resize bilinear
- Resize bicubic
- Resize Lanczos
- Resize supersampling
- Horizontal mirror

Filters:
- Sobel 3x3
- Gaussian

Conversions and other operations:
- gray<->rgb(a)
- I420<->rgb(a)
- nv12<->rgb(a)
- rgbp<->rgb(a)
- rgb<->rgba
- Convert
- Copy
- Normalize

To be Added in Spec v0.7 (2023)

Batch operations:
- Batch mirror
- Batch resize bilinear

Transformations:
- Resize nearest
- Warp perspective nearest
- Transposition

Filters:
- Sobel generic
- Bilateral
- Box
- Convolution
- Separable
- Median

Conversions and other operations
- Color twist
- Swap channels
- Magnitude

To be Added in Spec v0.8 (2024)

Batch operations:
- Batch resize
- Batch warp
- Batch conversions

Transformations:
- Rotate
- Warp affine
- Warp perspective

Filters:
- Erode
- Dilate

Other operations:
- Histogram
- Threshold

How does oneIPL compare to IPP/NPP?

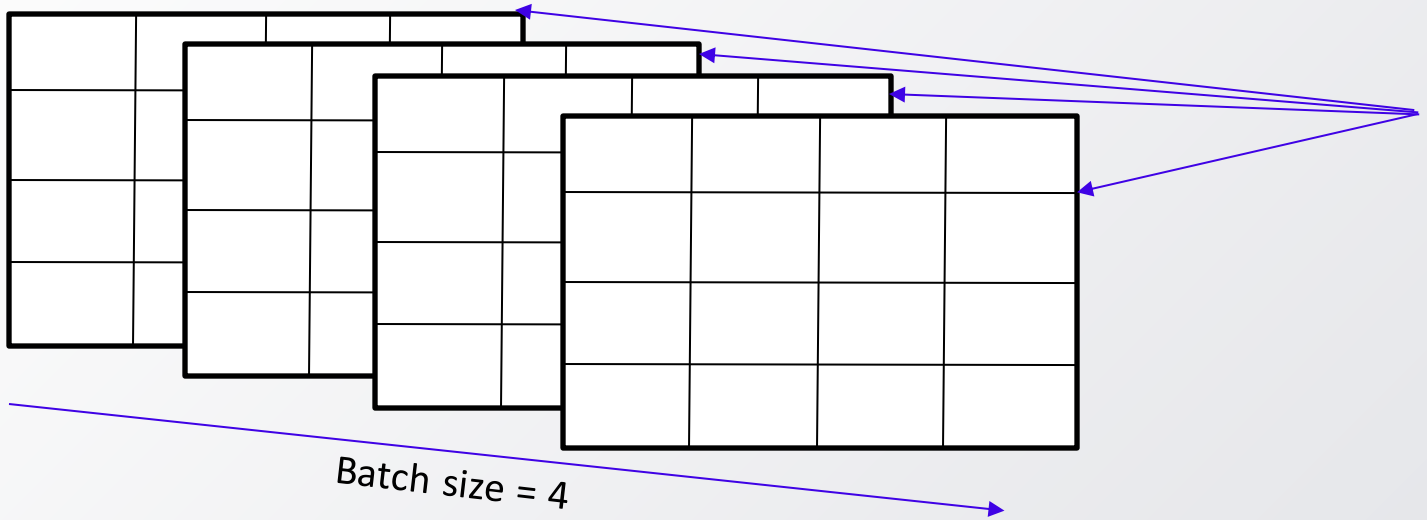
oneIPL is a SYCL/C++ counterpart for the existing libraries with C-APIs:

- Intel IPP is a library of functions for performing x86 accelerated 2D image and signal processing (~30 years old)
- NVIDIA NPP is a library of functions for performing CUDA accelerated 2D image and signal processing (~10 years old)

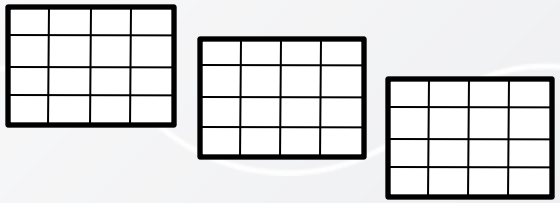
oneIPL covers a subset of IPP/NPP functions for image processing, introducing:

- Data abstraction for 2D images with similarity to SYCL containers
- Async execution on devices based on the SYCL queue
- USM and image memory support, memory management, and allocators
- Error handling based on exceptions

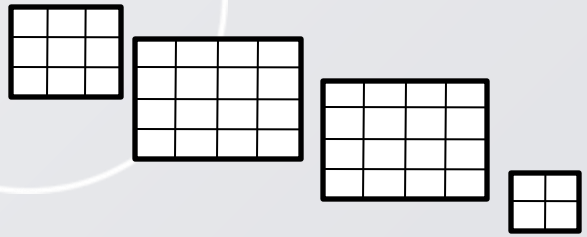
Basic terminology: Batch processing



Uniform batch (all images have same size)



Non-Uniform batch (images have different sizes)



Batch API – Targeted to Machine Learning

Ex1: MLPerf benchmark pipeline (Resnet50)

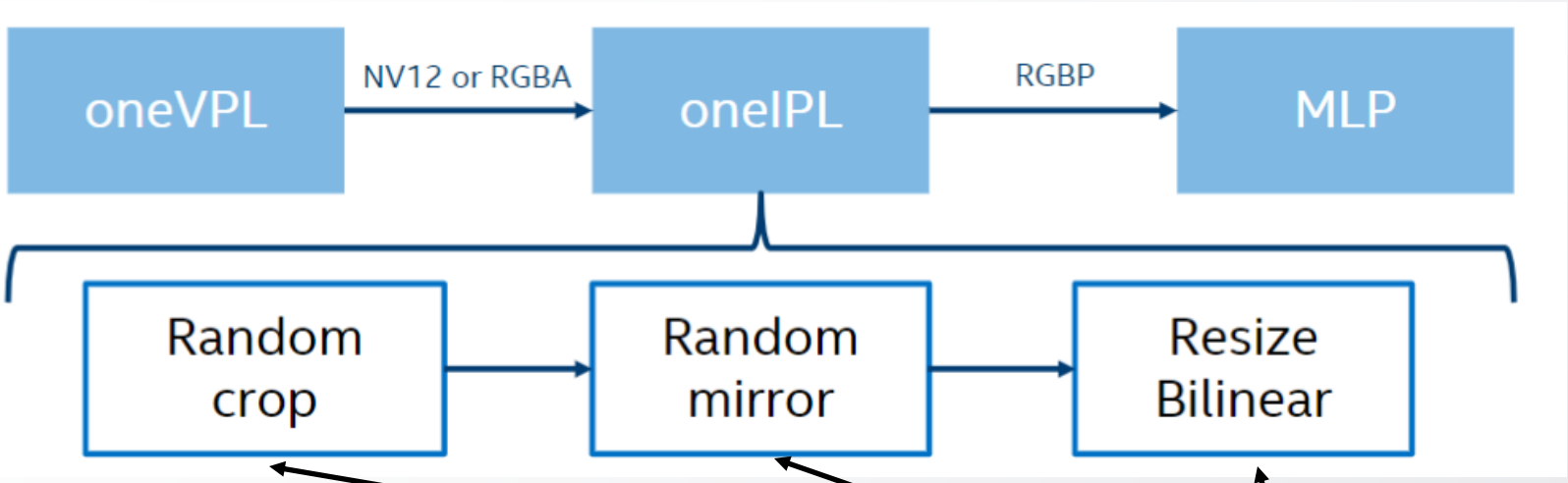
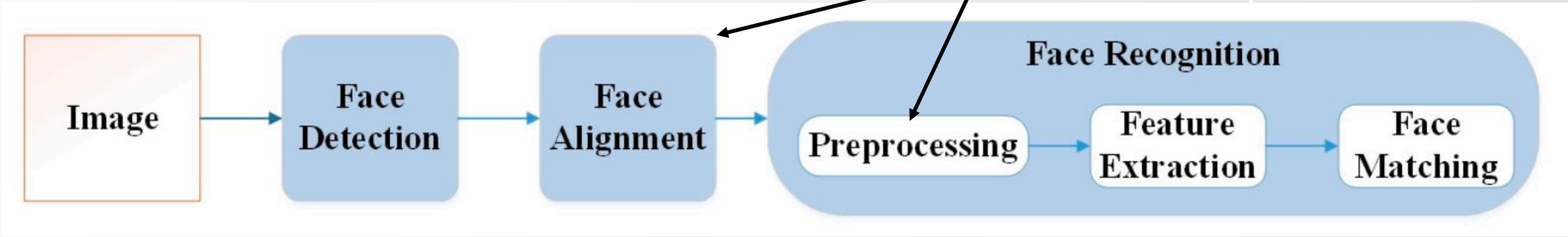


Image processing pipelines for machine learning operate with batches (e.g., multiple images, multiple video frames). Batch APIs provide benefits on GPUs and are also used in oneMKL and NVIDIA NPP.

Ex2: 2-step face recognition



Batch API Plans

Batch API plans based on user requests and analysis of 3rd-party solutions.

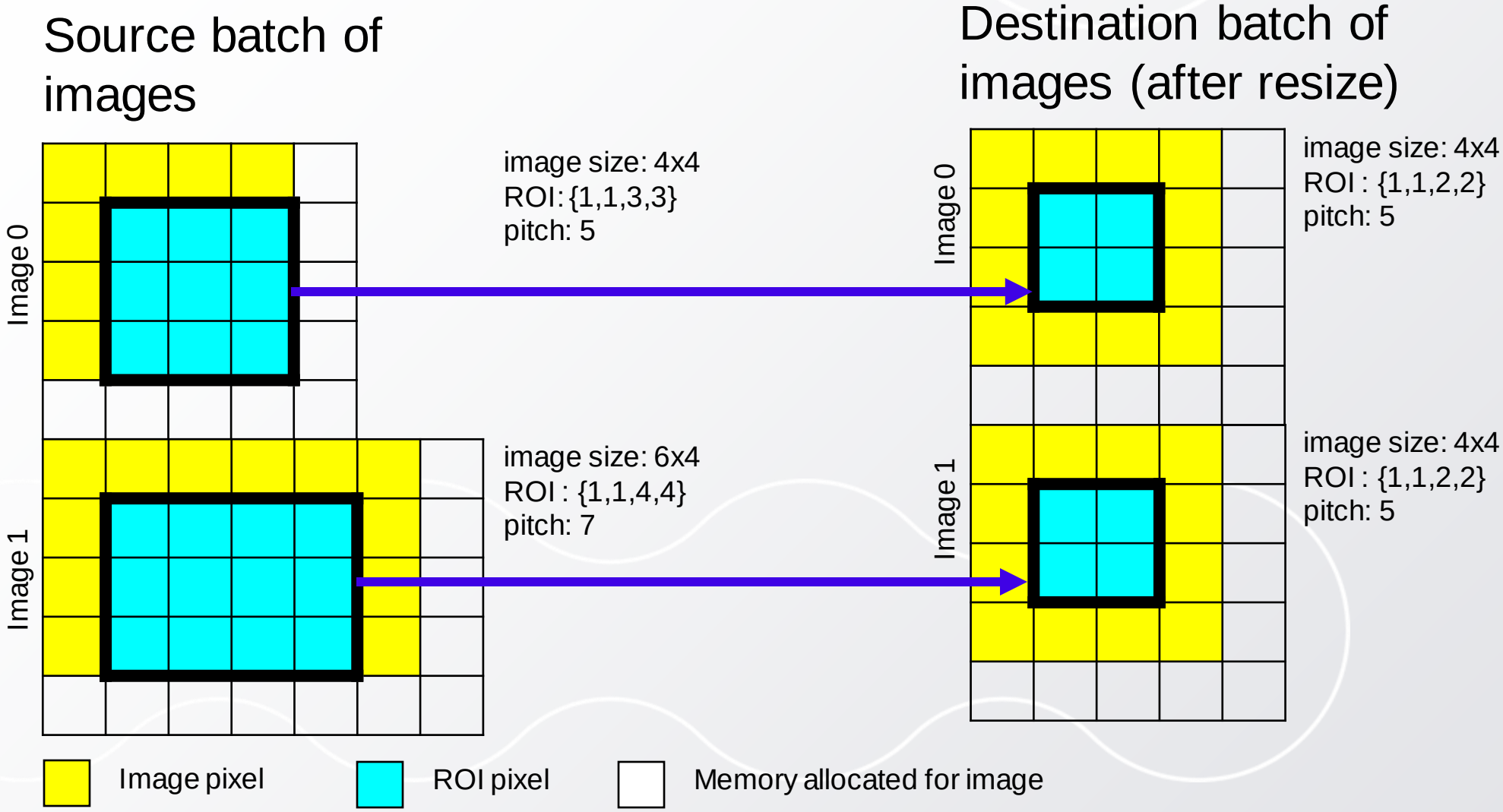
YUV420/422/… to RGB conversions
Resize
Warp
Mirror

} Planned for 2024

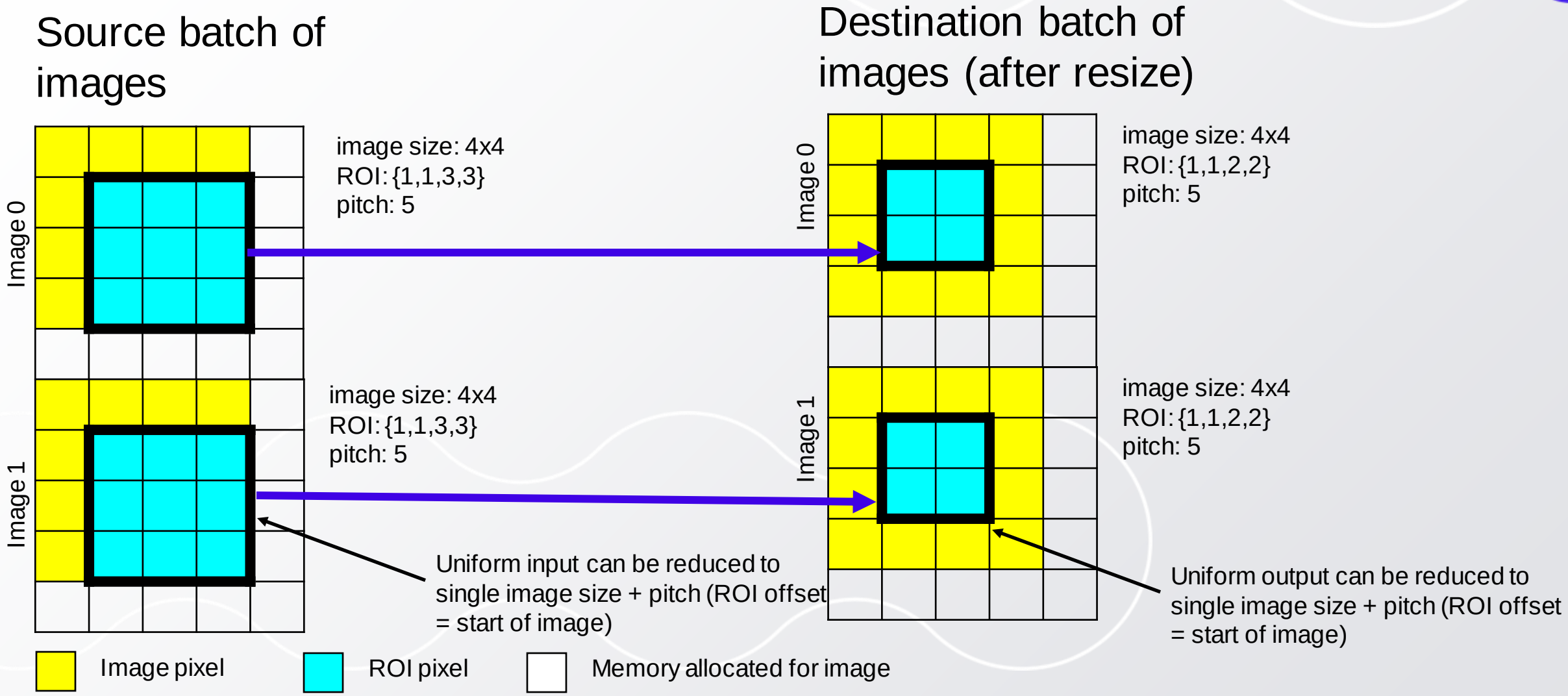
Color Twist,
Label Markers,
Quality Evaluation (PNSR, …)

} By request, not currently planned

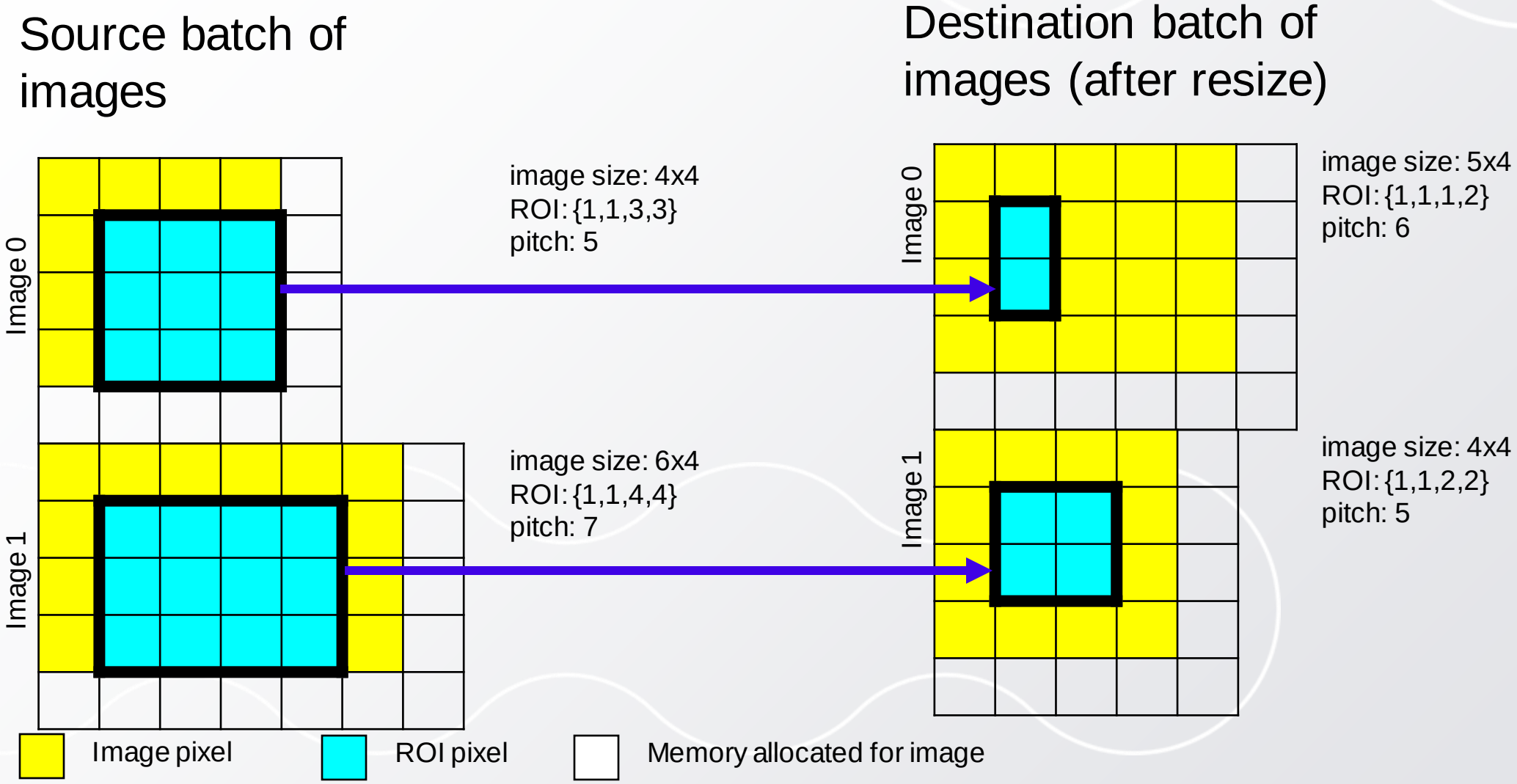
Batch API – non-uniform input, uniform output



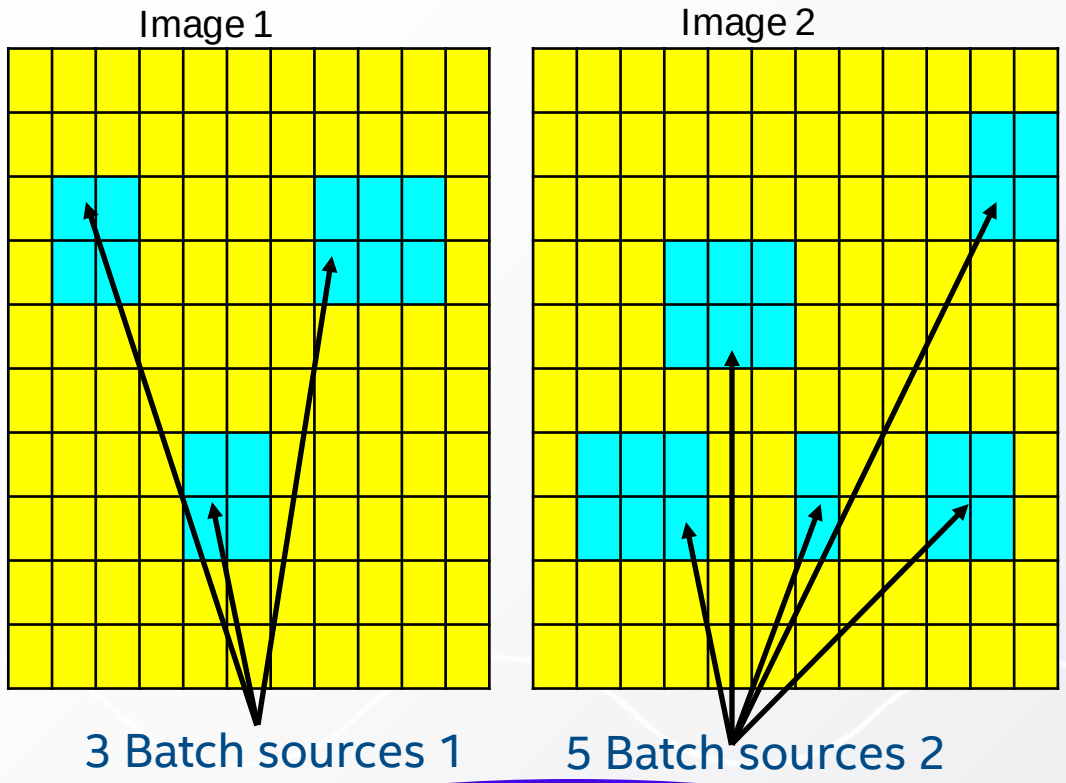
Batch API – uniform input, uniform output



Batch API – non-uniform output (no plans to support)



Batch API – Flexibility for data sources



Input Batch:
{image1 (size1, pitch1), roi11}
{image1 (size1, pitch1), roi12}
{image1 (size2, pitch2), roi13}
{image2 (size2, pitch2), roi21}
{image2 (size2, pitch2), roi22}
{image2 (size2, pitch2), roi23}
{image2 (size2, pitch2), roi24}
{image2 (size2, pitch2), roi25}

Output Batch:
{image1 (size, pitch)}
{image2 (size, pitch)}
{image3 (size, pitch)}
{image4 (size, pitch)}
{image5 (size, pitch)}
{image6 (size, pitch)}
{image7 (size, pitch)}
{image8 (size, pitch)}

Batch API supposes the inputs might be ROIs from a single big image.

The supported use-case is an object recognition pipeline, which provides a batch of ROIs after the detection phase.

Batch-related classes API

```
template <layouts Layout, typename DataT>
struct image_descriptor {
    image_descriptor(DataT* data, const std::size_t pitch, const sycl::range<2>& size, const roi_rect& roi_rect);

    DataT*          data;   ///< pointer to image data (can be pointers to the ROI of the same image)
    std::size_t     pitch;  ///< image pitch in bytes
    sycl::range<2>  size;   ///< 2D size of the image
    roi_rect        roi;    ///< region of interest (ROI)
};
```

```
template <layouts Layout, typename DataT>
class batch {
public:
    using data_t = DataT;
    static constexpr auto layout_v = Layout;
    static constexpr auto channel_count_v = detail::channel_count_v<Layout>;

    batch(image_descriptor<Layout, DataT>* const image_descriptors,
          const std::size_t                      batch_size,
          const sycl::range<2>&                  max_roi_size);

    image_descriptor<Layout, DataT>* get_image_descriptors() noexcept;

    const sycl::range<3>& get_range() const noexcept;
};
```

1) Image descriptor contains image metadata
2) Batch stores pointer to image descriptors for batch images and 3d range. This range has number of images as 0-component and size of the max image ROI in the batch as 1- and 2-components

Batch features in oneIPL spec 0.7


```
template <typename SrcBatchT,
          typename DstBatchT>
sycl::event mirror_batch(sycl::queue&                    queue,
                         SrcBatchT&                      src,
                         DstBatchT&                      dst,
                         const mirror_spec&              spec         = {},
                         const std::vector<sycl::event>& dependencies = {})
```

```
template <typename ComputeT = float,
          typename SrcBatchT,
          typename DstBatchT>
sycl::event resize_bilinear_batch(sycl::queue&                    queue,
                                  SrcBatchT&                      src,
                                  DstBatchT&                      dst,
                                  const resize_bilinear_spec&     spec         = {},
                                  const std::vector<sycl::event>& dependencies = {})
```

oneIPL batch API usage example

Image data pointers (src_ptrs and dst_ptrs) and image descriptors pointers should be accessible from the device.

```
// Allocate shared memory for batch images metadata
auto src_image_descriptors = sycl::malloc_shared<image_descriptor<layouts::channel4, std::uint8_t>>(batch_size, queue);
auto dst_image_descriptors = sycl::malloc_shared<image_descriptor<layouts::channel4, std::uint8_t>>(batch_size, queue);

// Fill batch images metadata
for (std::size_t i{ 0U }; i < batch_size; ++i) {
    src_image_descriptors[i] =
        image_descriptor<layouts::channel4, std::uint8_t>{ src_ptrs[i], src_pitches[i], src_sizes[i], src_roi_rects[i] };
    dst_image_descriptors[i] =
        image_descriptor<layouts::channel4, std::uint8_t>{ dst_ptrs[i], dst_pitches[i], dst_sizes[i], dst_roi_rects[i] };
}

// Create source and destination batches
batch<layouts::channel4, std::uint8_t> src_batch{ src_image_descriptors, batch_size, src_max_roi_size };
batch<layouts::channel4, std::uint8_t> dst_batch{ dst_image_descriptors, batch_size, dst_roi_size };

// Resize batch of images
resize_bilinear_batch(queue, src_batch, dst_batch).wait();

// Free the allocated memory
sycl::free(src_image_descriptors, queue);
sycl::free(dst_image_descriptors, queue);
```

oneIPL batch API usage example

Doing the chain of operations requires explicit synchronization.

```
// Create source and destination batches
batch<layouts::channel4, std::uint8_t> src_batch{ src_image_descriptors, batch_size, src_max_roi_size };
batch<layouts::channel4, std::uint8_t> tmp_batch{ tmp_image_descriptors, batch_size, dst_roi_size };
batch<layouts::channel4, std::uint8_t> dst_batch{ dst_image_descriptors, batch_size, dst_roi_size };


// Resize batch of images
auto resize_event = resize_bilinear_batch(queue, src_batch, dst_batch);
auto mirror_event = mirror_batch(queue, src_batch, dst_batch, {resize_event});

some_user_batch_kernel(queue, src_batch, dst_batch, {mirror_event}).wait();


// Free the allocated memory
sycl::free(src_image_descriptors, queue);
sycl::free(tmp_image_descriptors, queue);
sycl::free(dst_image_descriptors, queue);
```

Thank you for attending!

- If you have content to post to oneAPI.io, please let us know
- Please feel free to extend invitations to others to join the Image SIG or other oneAPI Community Forums (Math, Language, Hardware, AI)

- Subscribe to the mailing list by sending an email to:
  - Image-SIG+subscribe@lists.uxlfoundation.org
- Join oneAPI on LinkedIn:
  - https://www.linkedin.com/groups/14241252/