ESRF | The European Synchrotron

LIMA2 APIs
Detector and Processing Plugins Development

Samuel Debionne, Alejandro Homs Puron, Laurent Claustre @ BCU / SG / ISDD

https://limagroup.gitlab-pages.esrf.fr/lima2

IMPROVEMENT FROM LIMA1 (SOFTWARE DESIGN POV)

Ease the learning curve

- **Reduce API surface**
- **Modern C++**
- **Recognized Libraries**
- **Generic programming / less boilerplate**
- **Explicit state machine**

Focus on High Performance

- **CPU affinity, memory allocation, placement (specific allocators)**
- **Accelerators support (GPU)**
- **Network (RDMA, io_uring, DPDK)**
- **Data models (dense, sparse, events)**
- **Distributed (MPI)**

CAMERA PLUGIN C++ / PYTHON API FOR CONTROL

```cpp
namespace lima::detectors::simulator
{
    class LIMA_SIMULATOR_EXPORT control
    {
      public:
        using init_params_t = /* unspecified */;
        using acq_params_t = /* unspecified */;
        using det_info_t = hw::info;
        using det_capabilities_t = hw::capabilities;
        using det_status_t = hw::status;

        /// Construct the control and broadcast the camera init parameters
        control(init_params_t const& init_params);

        /// Prepare acquisition
        void prepare_acq(acq_params_t const& acq_params);

        /// Start acquisition
        void start_acq();

        /// Software trigger if the camera supports it
        void soft_trigger();

        /// Stop acquisition
        void stop_acq();

        /// Called when all receivers acquisition has ended
        void close_acq();

        /// Reset camera
        void reset_acq(reset_level_enum level);

        /// Returns the number of frames acquired
        int nb_frames_acquired() const;

        /// Returns the state of the control
        acq_state_enum state() const;

        /// Register a callBack for a change of state event
        void register_on_state_change(std::function<void(acq_state_enum)> cbk);

        /// Returns the detector information, capabilities, status
        det_info_t det_info() const;
        det_capabilities_t det_capabilities() const;
        det_status_t det_status() const;

} // namespace lima ::detectors::simulator
```

initialization

control

state / progress

info / capabilities

CAMERA PLUGIN C++ / PYTHON API FOR ACQUISITION

```cpp
namespace lima::detectors::simulator
{
    class LIMA_SIMULATOR_EXPORT acquisition
    {
      public:
        using init_params_t = /* unspecified */;
        using acq_params_t = /* unspecified */;
        using acq_info_t = /* unspecified */;

        acquisition(std::pmr::polymorphic_allocator<std::byte> alloc);

        /// Prepare acquisition and returns information for the processing
        acq_info_t prepare_acq(acq_params_t const& acq_params);

        /// Start acquisition
        void start_acq();

        /// Stop acquisition
        void stop_acq();

        /// Called when all receivers acquisition has ended
        void close_acq();

        /// Reset acquisition
        void reset_acq();

        /// Returns the number of frames transferred
        int nb_frames_xferred() const;

        /// Returns the state of the receiver
        acq_state_enum state() const;

        /// Register a callack for on change of state event
        void register_on_state_change(std::function<void(acq_state_enum)> cbk);

        /// Register a callack for on start of acquisition event
        void register_on_start_acq(std::function<void()> cbk);

        /// Register a callack for on frame ready event
        void register_on_frame_ready(std::function<void(frame)> cbk);

        /// Register a callack for on end of acquisition event
        void register_on_end_acq(std::function<void(int)> cbk);


} // namespace lima ::detectors::simulator
```

initialization

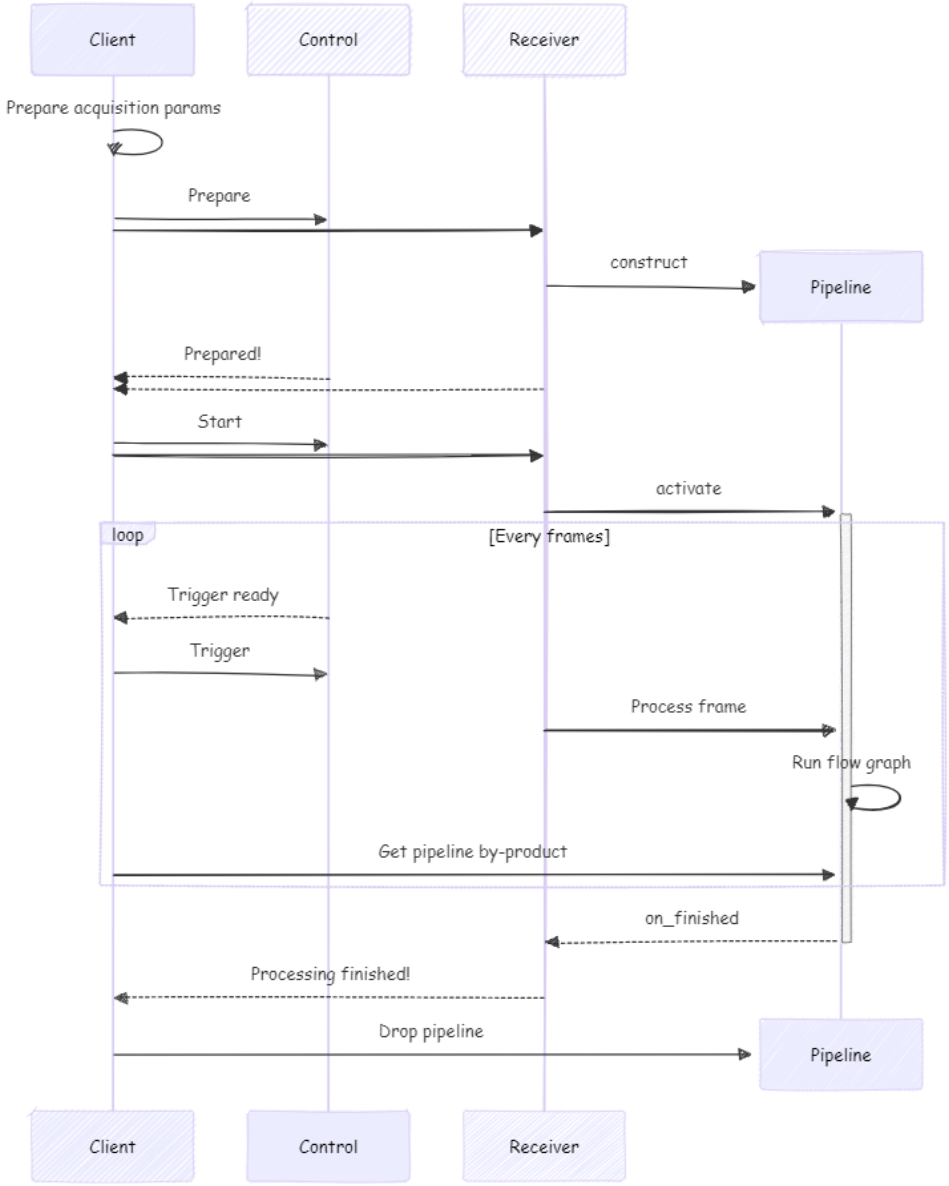
control

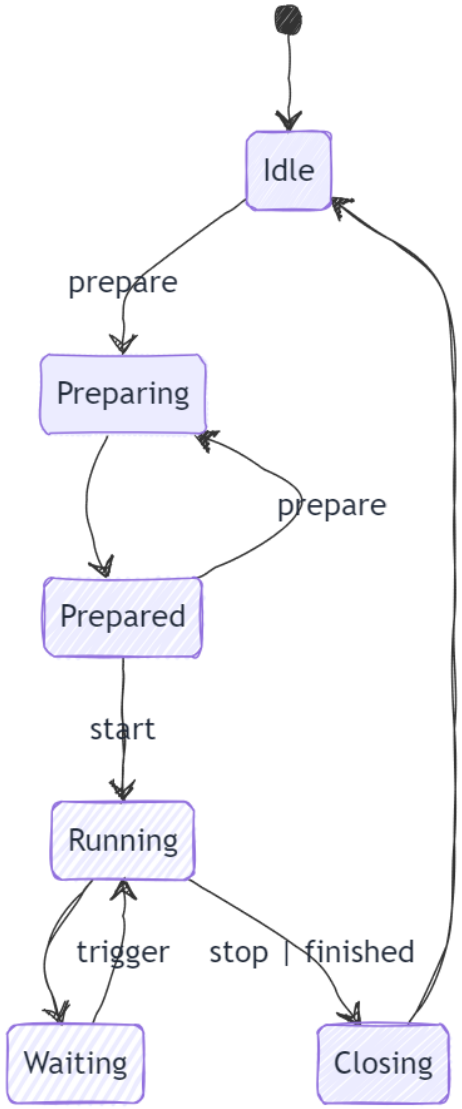
Not all the actions need to be implemented

state / progress

callbacks

SEQUENCE DIAGRAM AND STATE MACHINE





DETECTOR PLUGIN IMPLEMENTATION

- Acquisition thread / pipeline
- Finite State Machine
- Common set of acquisition parameters

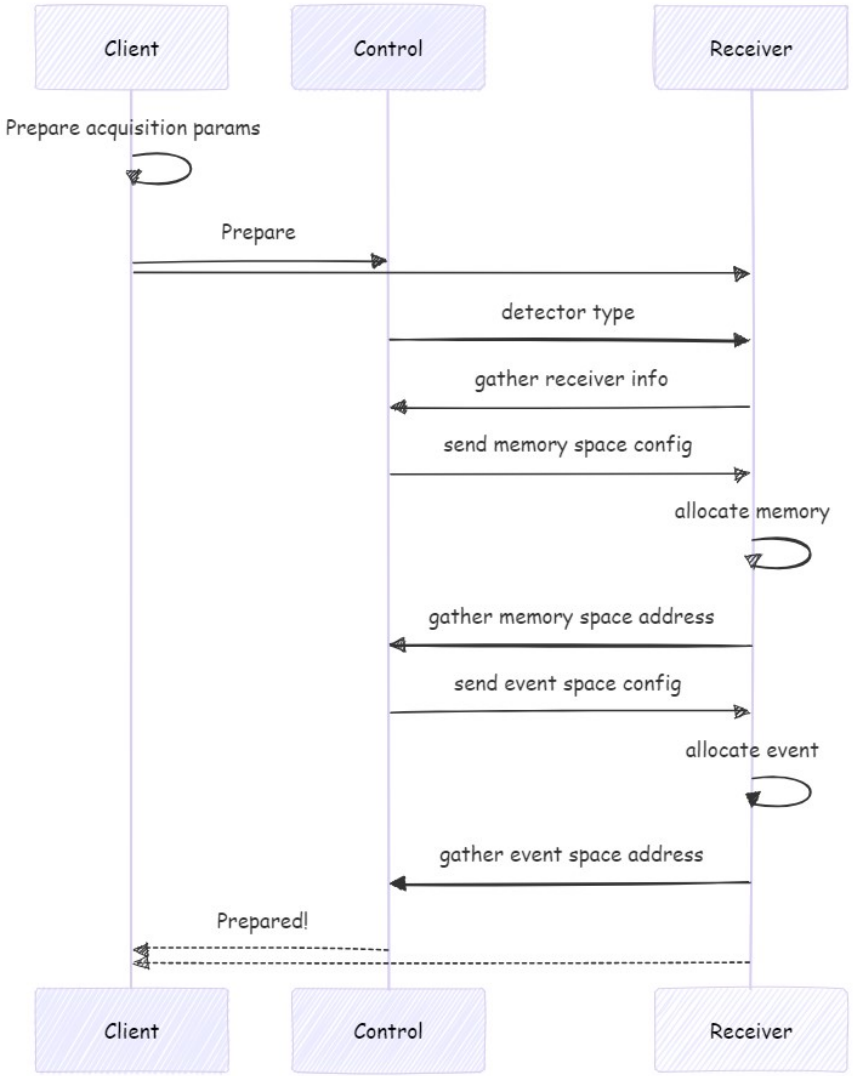
```
// Acquisition part of the detector
class acquisition_impl : public hw::acquisition_init_mpi<config>,
                         public hw::acquisition_thread<acquisition_impl, config>,
                         public hw::acquisition_fsm<acquisition_impl, config>
```

CODE: state machine implementations

```
/// Validate parameters
bool hw_validate_acq_params(acq_params_t const& acq_params) const;

/// Prepare acquisition
void hw_prepare(acq_params_t const& acq_params);

/// Start acquisition
void hw_start();

// Software trigger if the camera supports it
void hw_trigger();

/// Stop acquisition
void hw_stop();

/// Close acquisition
void hw_close();

/// Reset detector
void hw_reset();
```

```
/// Prepare the acquisition (e.g. allocate buffers)
acq_info_t hw_prepare(acq_params_t const& acq_params);

/// Start acquisition loop
void hw_start();

/// Stop acquisition loop
void hw_stop();

/// Close acquisition
void hw_close();

/// Get frame
data_t hw_get_frame() noexcept;
```

SYNCHRONIZATION AND COMMUNICATION BETWEEN COMPONENTS



Control side

```
boost::mpi::broadcast(m_comm, det_type, 0);
```

```
m_comm.recv(rcv.rank, 30, recv_info);
```

```
m_comm.send(rcv.rank, 31, e_buffer);
```

```
m_comm.recv(rcv.rank, 32, addressed_events);
```

```
m_comm.send(rcv.rank, 33, l_buffer);
```

```
m_comm.recv(rcv.rank, 34, addressed_buffers);
```

Requires careful error management to prevent deadlock

PROCESSING API

```
class PIPELINE_LEGACY_EXPORT pipeline
{
  public:
    static constexpr char const* const uid = "classic";

    pipeline(frame_info_t const& frame_info, proc_params_t const& proc_params);

    /// Activate the processing (start poping data from the queue)
    void activate();

    /// Abort the pipeline
    void abort();

    /// Process a frame
    void process(frame const& frm);

    /// Returns the current state of the pipeline
    state_enum state() const;

    /// Register on_finished callback
    void register_on_finished(finished_callback_t on_finished);

    /// Returns the progress counters
    progress_counters_t progress_counters() const;

    /// Accessors for pipeline by-product
    std::vector<roi_counters_result> pop_roi_statistics();
    std::vector<roi_profiles_result> pop_roi_profiles();

    std::optional<frame> get_input_frame(std::size_t frame_idx = -1) const;
    std::optional<frame> get_processed_frame(std::size_t frame_idx = -1) const;

    /// Returns the frame info at various stage of the pipeline
    frame_info_t input_frame_info() const;
    frame_info_t processed_frame_info() const;

    /// Returns the version of the pipeline plugin
    std::string version() const;
}
```

control

state / progress

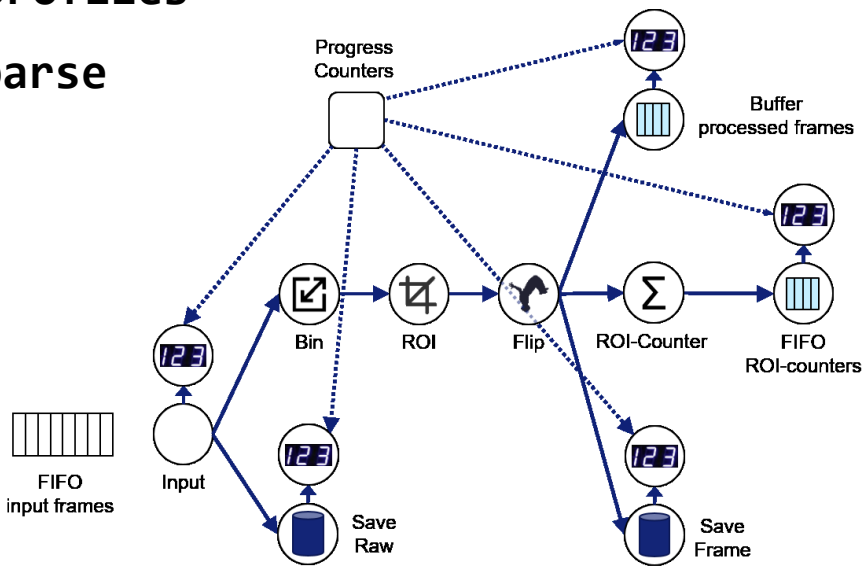
by-product accessors

BUILDING PROCESSING USING ONEAPI TBB FLOW GRAPH

Any technology can be used to implement pipelines.

Collections of nodes that compose a graph:

- **geometry transformation:** `roi, rotation, flip`
- **data reduction:** `binning, accumulation, peak_finder`
- **statistics:** `roi_statistics, roi_profiles`
- **saving:** `saving_dense, saving_sparse`



Collections of graphs:

- **CLASSIC: similar to Lima1**
- **SMX: background extraction, peak finding…**
- **XPCS: saving to sparse data**

BUILDING PROCESSING USING ONEAPI TBB FLOW GRAPH

```
// Source node that pop data from the processing FIFO
tbb::flow::input_node<frame> src(graph, [this, stop = false](tbb::flow_control& fc) mutable { … }

// Accumulation
accumulation_node accumulation(graph, params.accumulation.nb_frames, params.accumulation.pixel_type);

// Crop
crop_node crop(graph, tbb::flow::unlimited, params.geometry.roi);

// Mask
frame mask_frame;
detail::read_h5_dset(params.mask.path, mask_frame, "mask");
mask_node mask(graph, tbb::flow::unlimited, mask_frame);

// ROI statistics node
roi_counters_node roi_counters(graph, tbb::flow::unlimited, params.counters.rect_rois, params.counters.arc_rois);
unbounded_buffer_node<roi_counters_result> roi_counters_buffer(graph, tbb::flow::unlimited, roi_counters_buffer);

// Saving node
io_hdf5_node io_hdf5(graph, params.saving, 0, m_processed_frame_info);

// Connect graph
tbb::flow::make_edge(src, crop);
tbb::flow::make_edge(crop, accumulation);
tbb::flow::make_edge(accumulation, mask);
tbb::flow::make_edge(mask, io_hdf5);
tbb::flow::make_edge(mask, roi_counters);
tbb::flow::make_edge(roi_counters, roi_counters_buffer);

src.activate();
```

ONEAPI TBB FLOW GRAPH AND HETEROGENEOUS COMPUTING

Offload the computation to GPU/FPGA (ancillary thread)

```
// async_node -- Offload to GPU
async_sycl_activity async_gpu;
async_node gpu_node{g, tbb::flow::unlimited, [&async_gpu](gateway_type& gateway) { async_act.submit(gateway); }};
```

TBB Flow Graph as coordination layer

- The glue that connects CPU / GPU (load balancing)
- Simplify integration

ONEAPI TBB FLOW GRAPH AND HETEROGENEOUS COMPUTING

```
class async_sycl_activity
{
    tbb::flow::async_node<float, done_tag>::gateway_type* gateway_ptr;
    std::atomic<bool> submit_flag;
    std::thread service_thread;

  public:
    async_sycl_activity() :
        gateway_ptr(nullptr), submit_flag(false), service_thread([this] {
            while (!submit_flag) {
                std::this_thread::yield();
            }
            const float alpha = 0.5; // coeff for triad calculation

            // By including all the SYCL work in a {} block, all SYCL tasks complete before exiting the block
            {
                // Starting SYCL code
                sycl::range<1> n_items{array_size_sycl};
                sycl::buffer a_buffer(a_array), b_buffer(b_array), c_buffer(c_array);

                sycl::queue q(sycl::default_selector_v, [](sycl::exception_list exs) { ... });
                q.submit([&](sycl::handler& h) {
                     sycl::accessor a_accessor(a_buffer, h, sycl::read_only), b_accessor(…), c_accessor(…);

                     // Run the kernel
                     h.parallel_for(n_items, [=](sycl::id<1> index) {
                         c_accessor[index] = a_accessor[index] + b_accessor[index] * alpha;
                     }); // end of the kernel -- parallel for
                 }).wait();
            }

            gateway_ptr->try_put(done_tag{});
            gateway_ptr->release_wait();
        })
    {
    }
…
```

TANGO INTEGRATION

Detector and Processing encapsulated as Tango classes.

The plugin interface `create_class()` returns a Tango class.

```
using control_t = lima::detectors::simulator::control;

// Explicitely instantiate template for Device
template class lima::tango::control<control_t>;

// Explicitely instantiate template for DeviceClass
template class lima::tango::control_class<control_t>;

// Factory method
static lima::tango::control_class<control_t>* create()
{
    return lima::tango::control_class<control_t>::init("LimaSimulatorControl");
}

BOOST_DLL_ALIAS(create,      // <-- this function is exported with...
                create_class // <-- ...this alias name
)
```

LIMA2 CORE LIBRARY

Vocabulary types: `frame, point, rectangle, arc…`

Introspection: based on Boost.Describe used with Boost.JSON, Boost.MPI serialization, fmtlib, pybind11, iostream

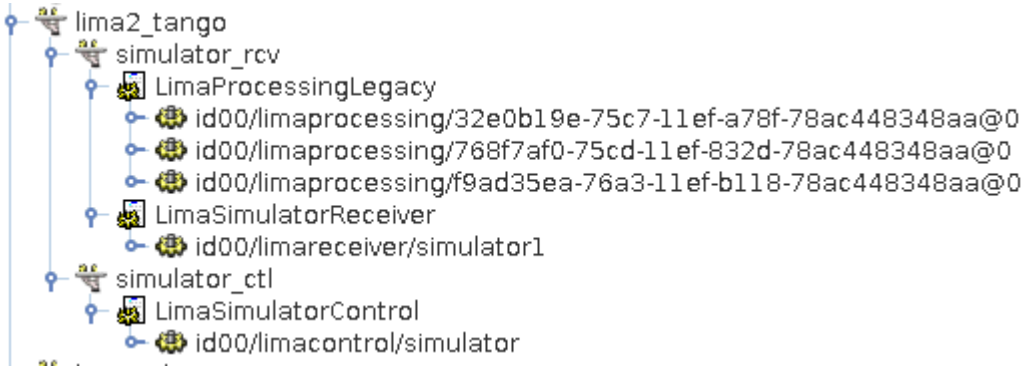
```
/// ZMQ context parameters
struct zmq_params
{
    int nb_io_threads = 1;                 //!< Number of I/O threads
    int thread_sched_policy = -1;          //!< I/O threads scheduler policy
    int thread_priority = -1;              //!< I/O threads priority
    std::vector<int> threads_cpu_affinity; //!< I/O threads CPU affinity
};


BOOST_DESCRIBE_STRUCT(zmq_params, (), (nb_io_threads, thread_sched_policy, thread_priority, threads_cpu_affinity))

/// Hardware ROI enum
enum class roi_enum : int
{
    full,
    roi_4m,
    roi_4m_left,
    roi_4m_right,
};

BOOST_DESCRIBE_ENUM(roi_enum, full, roi_4m, roi_4m_left, roi_4m_right)
```

Exception: based on Boost.Exception

Logging: based on Boost.Log

INSTALLATION AND EXECUTION

```
conda install --channel esrf-bcu --channel conda-forge lima2
```




```
TANGO_HOST="id00:20000" \
mpirun \
-n 1 lima2_tango simulator_ctl --log-level=info : \
-n 1 --map-by numa:PE=0 lima2_tango simulator_rcv -v4 --log-level=info
```

INSTALLATION AND EXECUTION

```
$ lima2_tango --help
Log level set to warning
Usage: lima2_tango <INSTANCE> [options]
Allowed options:
  --help                        Produce help message
  --debug                       Stop the server at the beginning to attach
                                debugger
  --plugin-folder arg (=.)      Plugin folder
  --log-level arg (=warning) Logging level [trace=0, debug, info,
warning,
                                error, fatal=5]
  --log-domain arg (=all)       Logging domain [core, ctl, acq,proc, io...]
```

LIMA2 CLIENT LIBRARY

```
conda install --channel esrf-bcu --channel conda-forge lima2-client
```

```python
import uuid
from lima2.client import Detector, State
from lima2.client.pipelines.legacy import Processing

tango_ctrl_dev = DeviceProxy("id00/limacontrol/ctl")
tango_recv_dev = DeviceProxy("id00/limareceiver/rcv1")
device = Detector(tango_ctrl_dev, tango_recv_dev)

acq_params = Detector.params_default
proc_params = Processing.params_default

def state_cb(state):
    _logger.debug(f"State change to {state=}")

device.register_transition_logger(state_cb)

uuid = uuid1()

device.prepare_acq(uuid, acq_params, proc_params)
device.start_acq()

while device.state == State.RUNNING:
    # Acquire some by-product data
    pass
```

Documentation: https://limagroup.gitlab-pages.esrf.fr/lima2-client/api.html