Lima1 Plugin Development Demystified

Laurent Claustre, Samuel Debionne, Alejandro Homs Puron
Sebastien Petitdemange, Emmanuel Papillon, Vicente Rey Bakaikoa
@ BCU / SG / ISDD

OUTLINE

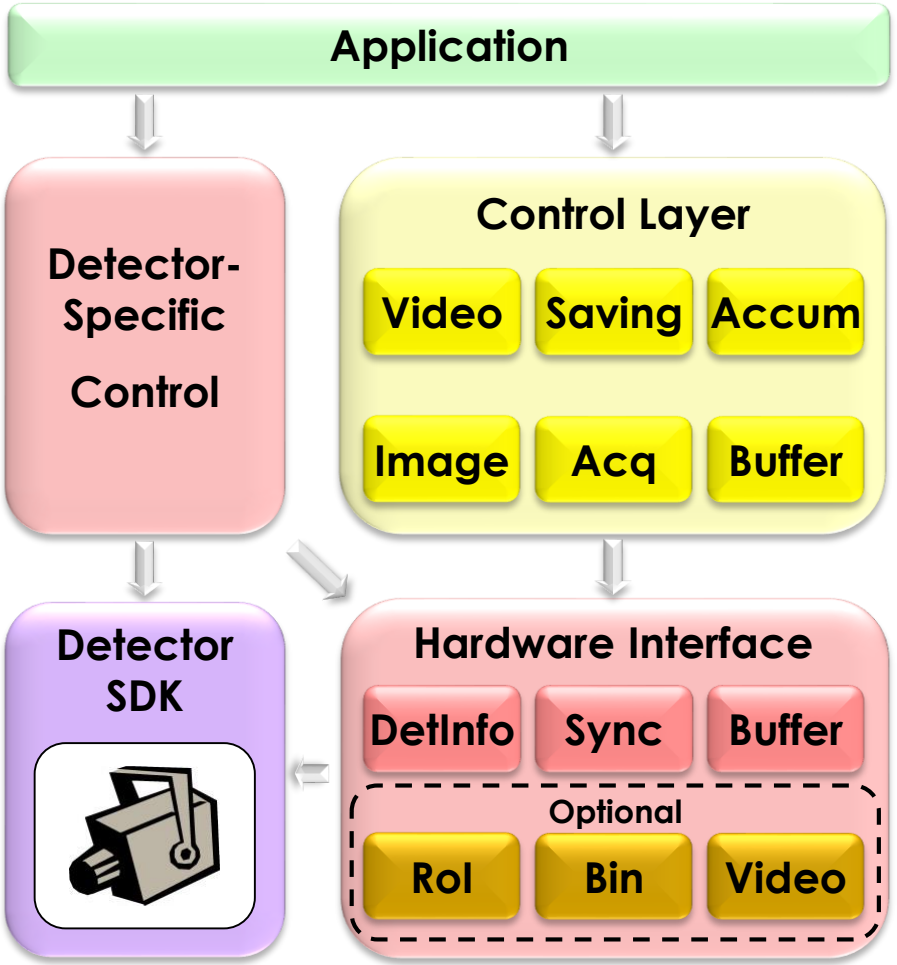
Introduction:

- LImA1 structure

Camera plugin:

- Hardware interface and capabilities
- SDK models
- Principles
- Basic example

LIBRARY STRUCTURE LAYOUT



LIMA CORE RESPONSIBILITIES

The Core will:

- Initialise the hardware (HW) interface
- Prepare the acquisition (ACQ):
  - Setting parameters in the HW capabilities
  - Prepare the HW interface
- Start the ACQ
- Poll the detector status
- Stop the ACQ if stopped by the user (external) or fatal event (internal)
  - Stop is not systematically called at the end of ACQ

CAMERA PLUGIN RESPONSIBILITIES

The camera plugin will:

- Expose its capabilities:
  - Statically: HW capability list
  - Dynamically:
    - Maximum image size
    - Valid timings
    - Effective binning / roi
- Notify on every new frame event:
  - It will be injected in the processing chain
  - Sequential frame number: no gap is allowed
- Reconstruct the frame, if needed
- Report any abnormal event like overrun, communication failure, etc

THE HARDWARE INTERFACE: MODULAR DESIGN

- Basic ACQ control:
  - Prepare, start, stop, status.
- List of functional HW capabilities, implemented by “control objects”
- Mandatory:
  - Detector information
  - Buffer management & newFrameReady event
  - Synchronization: NbFrames, ExpTime, ExtTriggerMode, etc.
- Optional:
  - Image transformation: Binning, ROI, Flip
  - Shutter control
  - Video control
  - Saving through SDK
  - Frame reconstruction
  - Configuration saving / restore
  - Event control: exceptions

HW CAP: DETECTOR INFORMATION

- Image sizes:
  - Detector sensor size
  - Maximum image size in current detector profile
  - Notify on maximum image size change with a CB
- Image type: bits-per-pixel
- Pixel size
- Detector type and model
- User-provided identifiers:
  - Detector name
  - Instrument name

HW CAP: BUFFER PRINCIPLES

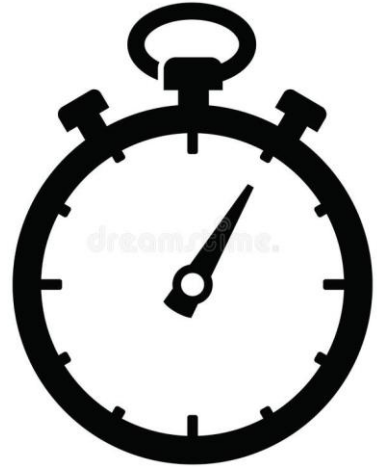
- Image buffer allocation can take time
  - Prepared in advance
- Small (stripe) ROIs can be taken very fast
  - Concatenation speeds-up readout
  - Multiple frames per buffer
- SDK can allocate low-level buffers
  - Default *SoftBufferCtrlObj* available
- LImA core will request low-level buffer allocation
  - Based on a *reqMemSizePercent* (default 70% of computer RAM)
- Triggers the *newFrameReady* events
  - Normally associated to frame grabbers
- Ring buffer policy

HW CAP: BUFFER IMPLEMENTATION

- Image frame dimension: width, size & depth
  - Determined by hardware Bin & ROI
- Concatenated frames per buffer (default: 1)
  - Used in stripe-concatenation mode
- Maximum number of buffers
  - Can be limited by the computer memory or SDK
- Buffer allocation and release
  - Keep track of buffer use in processing chain
  - Can slowdown new frame injection and/or detect overrun
- Image frame management
  - Memory pointer
  - Frame timestamp (since start of ACQ)
  - *newFrameReady* CB

HW CAP: SYNCHRONISATION

- Trigger mode – see next slide
  - Allow checking supported trigger modes
- Exposure time and latency (pause) time
  - Define the frame period in internal trigger
- Valid timing ranges
  - Minimum and maximum exposure and latency times
  - Notify when the valid ranges change with a CB
- Auto-exposure mode
  - Can be available in video cameras
- Number of HW frames in acquisition
  - 0 means endless acquisition



HW CAP: SYNCHRONISATION

Trigger modes explained

- *IntTrig*
  - All frames are triggered by internal detector timing after calling *startAcq*
- *IntTrigMult*
  - One frame is triggered at each call to *startAcq* (**multiple** triggers needed)
- *ExtTrigSingle*
  - All frames are triggered by internal detector timing after a **single** external pulse
- *ExtTrigMult*
  - One frame is triggered by an external pulse (**multiple** pluses needed)

- *ExtGate*
  - One frame is triggered by an external gate, defining the exposure
- *ExtStartStop*
  - Each frame is triggered by a first external pulse, a second pulse stops exposure
- *ExtTrigReadout*
  - Each external pulse stops the exposure of one frame and starts the next one

HW INTERFACE

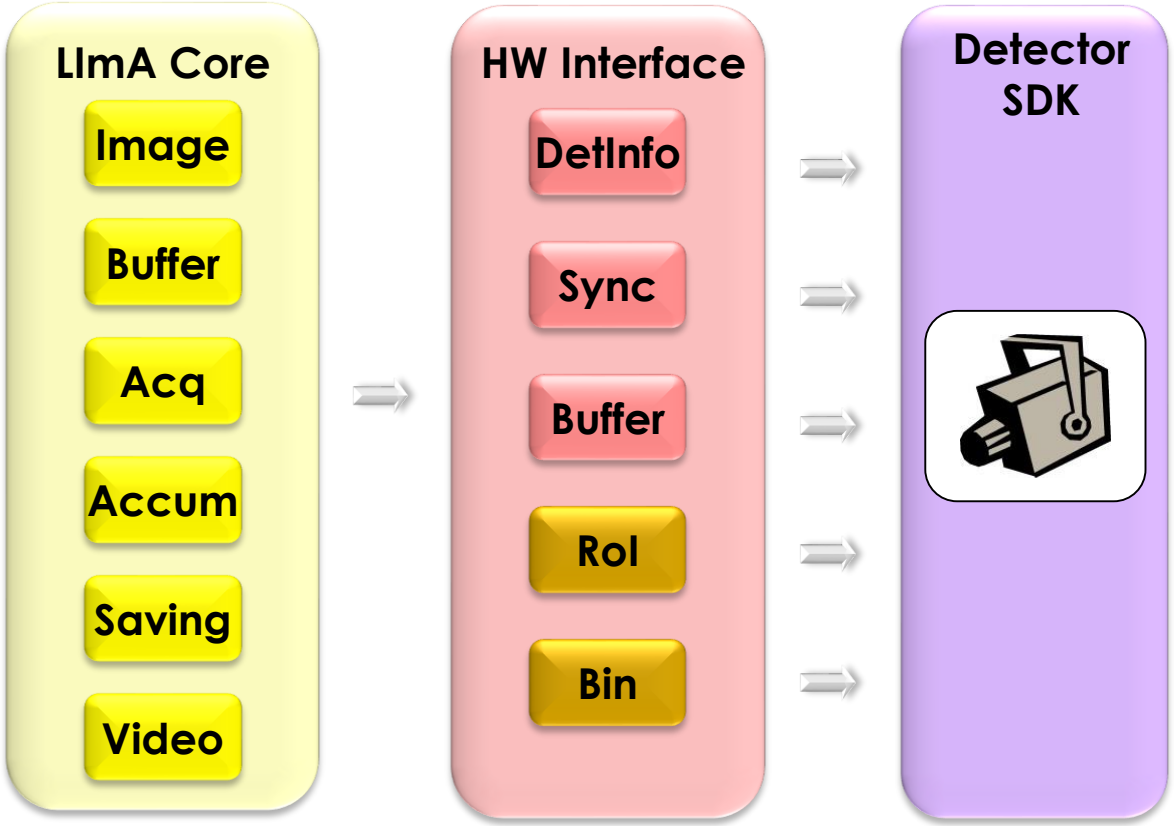
- HW capability list
  - And the corresponding control objects
- Reset – initialization to default values
- Prepare ACQ
  - Called after all HW capabilities have been configured
  - Final preparation before ACQ start
- Start ACQ
  - Can be called multiple times in *IntTrigMult*
- Stop ACQ
  - Might not be called if ACQ ends normally
  - Mandatory in endless ACQs
- Detector head and frame grabber ACQ status
  - Basic status provides predefined combinations
- Number of HW acquired frames

HW INTERFACE: STATUS

Basic status

- *Ready*
  - Detector can be triggered by SW or HW
- *Config*
  - Initialisation is not complete and additional configuration is need
- *Exposure*
  - Detector is running on exposure
- *Readout*
  - Detector is reading the frame – might not be visible in fast ACQs
- *Latency*
  - Detector is waiting between frames – might not be visible in fast ACQs
- *Fault*
  - Something failed

SDK MODELS: EVENT DRIVEN



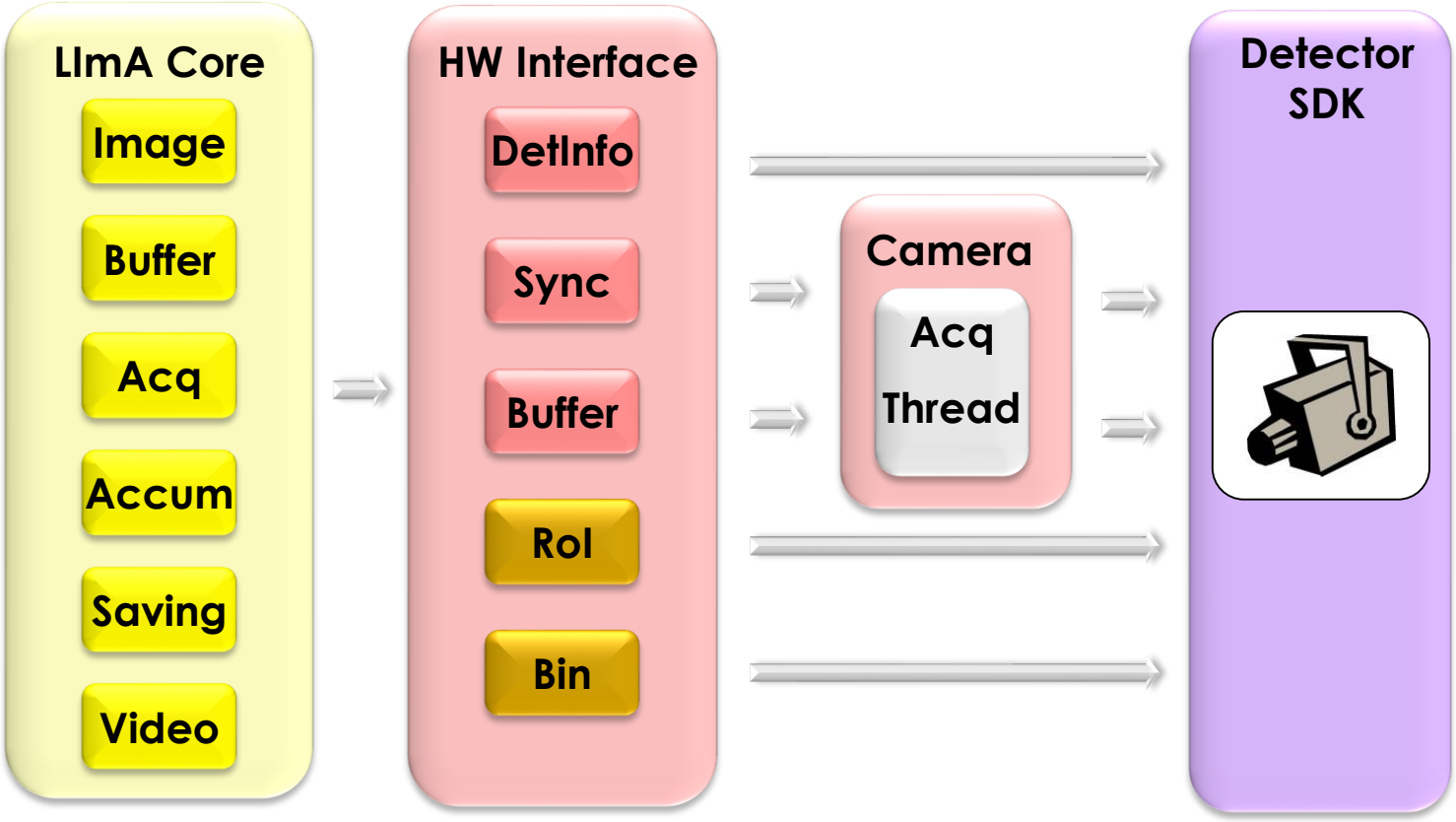
SDK features

- Events (callbacks)
- Status

HW plugin

- Virtually one-to-one function correspondence

SDK MODELS: BLOCKING CALLS



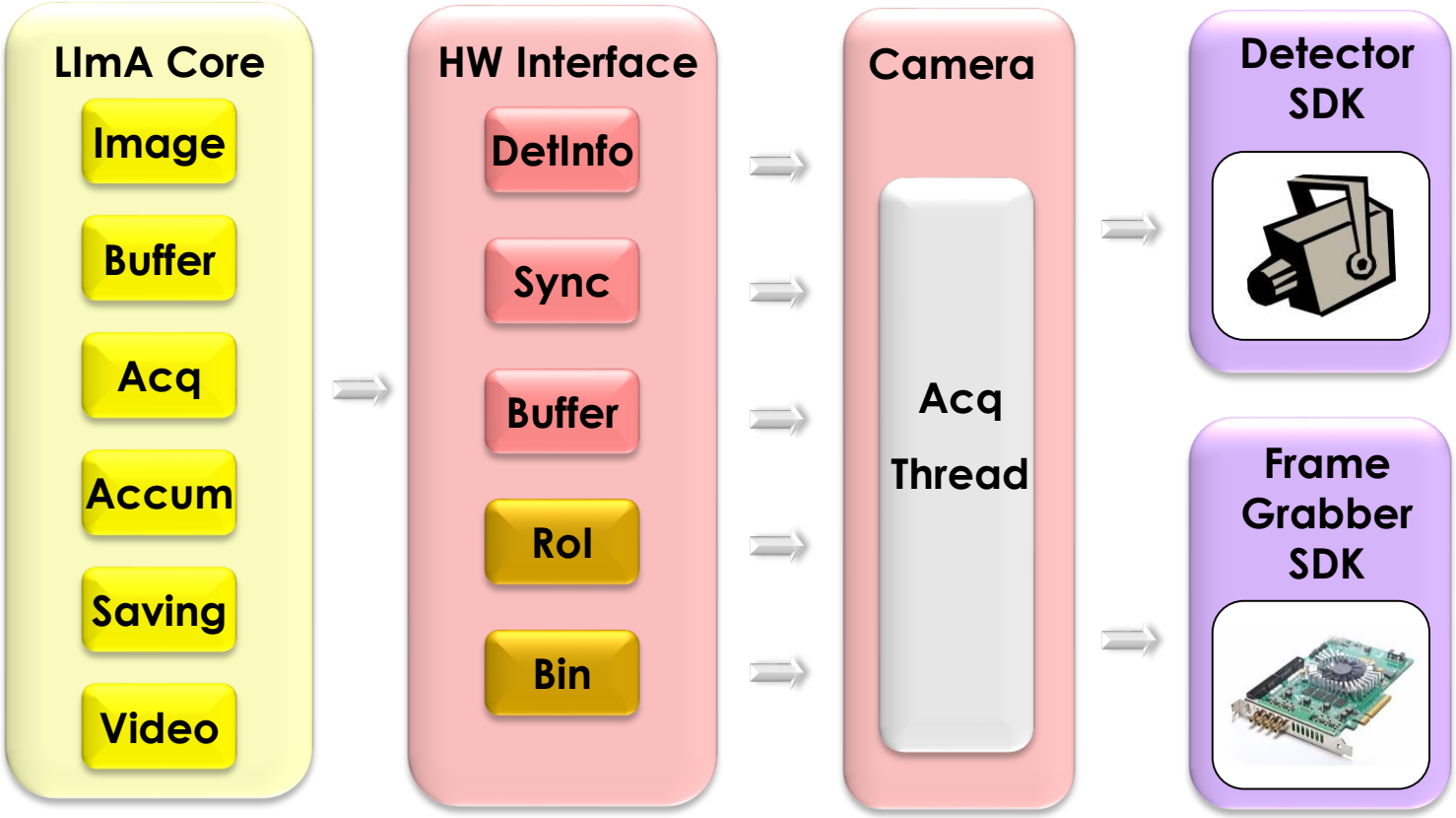
SDK features

- Blocking calls to *GetFrame()*

HW plugin

- Implement *AcqThread* calling *GetFrame()* and *newFrameReady()*

SDK MODELS: INDEPENDENT SDKS



SDK features

- Independent control of detector and frame grabber

HW plugin

- More complex *Camera and AcqThread* loop

HW PLUGIN DESIGN

- There is not a single model to write HW plugins
  - They are as diverse as SDK implementations are
- The HW interface is split into functional blocks
- Granularity was chosen to fit:
  - The LImA core needs
  - Common detectors and SDKs
- A plugin is implemented by inheriting from C++ abstract classes:
  - *HwInterface*
  - All the corresponding *HwXxxCtrlObj*
- A HW plugin design can be very simple yet different from other plugin models
  - It just needs to satisfy the LImA core contract requirements

HELPER TOOLS - PRE C++-11

General purpose

- *Debug & Exceptions*
- *DirectoryEvent*: follow file creation events in directories
- *RegEx*: regular expressions with some Python extensions
- *Timer*: high-precision software timer
- *VideoImage*: conversion between common video formats and LImA data images

HELPER TOOLS - PRE C++-11

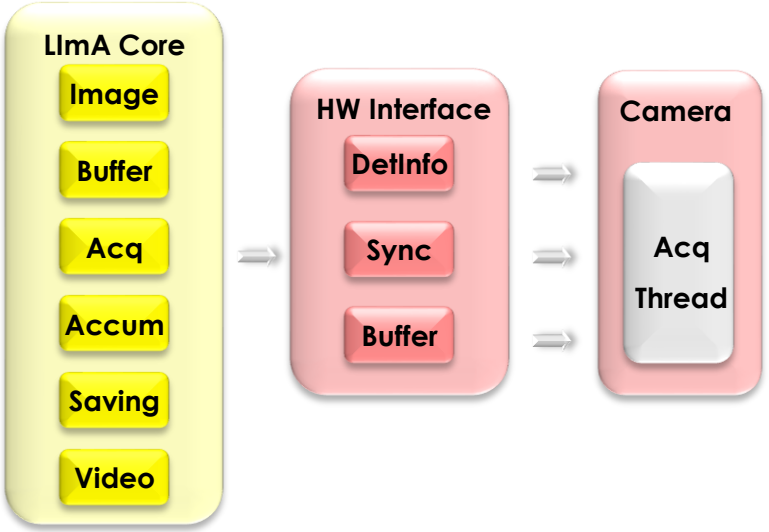
Threads and synchronisation

- *Mutex & Cond*
  - *AutoMutex*: uses RAII principle to unlock the mutex at the end of its scope
  - *AutoMutexUnlock*: temporarily unlocks an *AutoMutex* inside a sub-block (RAII)
- *ReadWriteLock*: allows multiple readers but a single, exclusive writer
- *Thread*: must be explicitly started after construction
  - *CmdThread*: a command-controlled *Thread* that exports a status
  - *ExceptionCleanUp*: an object ensuring that the *AcqThread* main loop exits properly, like stopping the hardware and setting a *Fault* status.

HW PLUGIN DESIGN: EXAMPLE

- Simple camera plugin:
  - Generating 512x512 8-bit images
  - With a gaussian spot in the center
  - Variable peak amplitude: linear increase, cycle every 16 frames
  - Only internal trigger is supported
  - Simulate the requested timing

Let's be crazy!



HW PLUGIN DESIGN: EXAMPLE

Template camera uses *cookiecutter*

```
(base) ~/devel$ mamba create -n cookiecutter cookiecutter

(base) ~/devel$ git clone https://gitlab.esrf.fr/limagroup/Lima-camera-template

(base) ~/devel$ (conda activate cookiecutter &&
                 cookiecutter Lima-camera-template)

project_name [MyCamera]: MyExample

folder_name [myexample]:

namespace_name [myexample]:

include_folder [myexample]:

macro_prefix [LIMA_MYEXAMPLE]:

lowercase_projectname [myexample]:

uppercase_projectname [MYEXAMPLE]:

build_for_windows [True]: False

build_for_linux [True]:
```

HW PLUGIN DESIGN: EXAMPLE

```
(base) ~/devel$ ls -R myexample

myexample:
cmake  CMakeLists.txt  conda  COPYING  doc  include  python
README.md  sip  src  tango  test

myexample/include/myexample:
Camera.h  DetInfoCtrlObj.h  Interface.h  ShutterCtrlObj.h
SyncCtrlObj.h

myexample/src:
MyExampleCamera.cpp          MyExampleInterface.cpp
MyExampleSyncCtrlObj.cpp     MyExampleDetInfoCtrlObj.cpp
MyExampleShutterCtrlObj.cpp

myexample/sip:
MyExampleCamera.sip  MyExampleInterface.sip

myexample/python:
__init__.py  MyExample.py

myexample/tango:
CMakeLists.txt  MyExample.py
```

HW PLUGIN EXAMPLE: CAMERA

```
#include <lima/Debug.h>

namespace lima
{
namespace myexample
{

  class MYEXAMPLE_EXPORT Camera
  {
    DEB_CLASS_NAMESPC(DebModCamera, "Camera", "MyExample");
  public:
    Camera();
    ~Camera();
  };

} // namespace myexample
} // namespace lima
```

Completely empty *Camera* class

Let’s transform it into a detector SDK …

HW PLUGIN EXAMPLE: CAMERA

```
public:
  enum Status { Init, Ready, Exposure, Readout };

  Camera();
  ~Camera();

  SoftBufferCtrlObj& getBufferCtrlObj();

  FrameDim getFrameDim();

  void setExpTime(double  exp_time);
  void getExpTime(double& exp_time);

  void setLatTime(double  lat_time);
  void getLatTime(double& lat_time);

  void setNbFrames(int  nb_frames);
  void getNbFrames(int& nb_frames);

  void prepareAcq();
  void startAcq();
  void stopAcq();

  Status getStatus();
  int getNbAcqFrames();
```

Detector status

Camera holds *SoftBufferCtrlObj* for easier buffer & frames management

Report frame size and pixel depth

Basic ACQ config. parameters

ACQ control

HW PLUGIN EXAMPLE: CAMERA

```
private:
  class AcqThread;
  void acqThreadFunction();

  void fillImage(int frame_nb);

  SoftBufferCtrlObj   m_buffer;

  TrigMode            m_trig_mode{IntTrig};
  int                 m_nb_frames{1};
  double              m_exp_time{1};
  double              m_lat_time{10e-9};
  int                 m_cycle_frames{16};

  Cond                m_cond;
  bool                m_stopped{false};
  Status              m_status{Ready};
  int                 m_acquired{0};
  AutoPtr<AcqThread>  m_acq_thread;
```

The *SoftBufferCtrlObj*

Basic ACQ parameters

ACQ thread: status & sync

AutoPtr is similar to *std::shared_ptr*

HW PLUGIN EXAMPLE: CAMERA

```cpp
SoftBufferCtrlObj& Camera::getBufferCtrlObj()
{
  return m_buffer;
}

FrameDim Camera::getFrameDim()
{
  DEB_MEMBER_FUNCT();
  FrameDim fdim(512, 512, Bpp8);
  DEB_RETURN() << DEB_VAR1(fdim);
  return fdim;
}

void Camera::setExpTime(double exp_time)
{
  DEB_MEMBER_FUNCT();
  DEB_PARAM() << DEB_VAR1(exp_time);
  m_exp_time = exp_time;
}

void Camera::getExpTime(double& exp_time)
{
  DEB_MEMBER_FUNCT();
  exp_time = m_exp_time;
  DEB_RETURN() << DEB_VAR1(exp_time);
}
```

Debug tracing helper

Many parameters are returned by reference

Idem with *LatTime* and *NbFrames*

HW PLUGIN EXAMPLE: CAMERA

```
void Camera::prepareAcq()
{
  DEB_MEMBER_FUNCT();
  AutoMutex l(m_cond.mutex());
  if (m_acq_thread->hasStarted())
    m_acq_thread->join();
  m_status = Init;
  m_stopped = false;
}

void Camera::startAcq()
{
  DEB_MEMBER_FUNCT();
  AutoMutex l(m_cond.mutex());
  if (m_status != Init)
    THROW_HW_ERROR(Error) << "Camera already started";
  m_acq_thread->start();
  while (m_status == Init)
    m_cond.wait();
}

void Camera::stopAcq()
{
  DEB_MEMBER_FUNCT();
  AutoMutex l(m_cond.mutex());
  m_stopped = true;
}
```

AutoMutex locks the *Mutex* on construction and un-locks it on destruction (RAII). Equivalent to *std::unique_lock*

Throw *lima::Exception* on error

Wait for *AcqThread* to be active before continuing

Cond::wait must be called with *Mutex* in locked state

HW PLUGIN EXAMPLE: CAMERA

```
Camera::Status Camera::getStatus()
{
  DEB_MEMBER_FUNCT();
  AutoMutex l(m_cond.mutex());
  DEB_RETURN() << DEB_VAR1(m_status);
  return m_status;
}

int Camera::getNbAcqFrames()
{
  DEB_MEMBER_FUNCT();
  AutoMutex l(m_cond.mutex());
  DEB_RETURN() << DEB_VAR1(m_acquired);
  return m_acquired;
}
```

Access to variables exchanged with *AcqThread* should be protected by *Mutex*

HW PLUGIN EXAMPLE: CAMERA

```
//---------------------------
//- AcqThread
//---------------------------

class Camera::AcqThread : public Thread
{
public:
  AcqThread(Camera& cam) : m_cam(cam) {}

private:
  void threadFunction() override { m_cam.acqThreadFunction(); }

  Camera& m_cam;
};

//---------------------------
//- Ctor
//---------------------------
Camera::Camera()
  : m_acq_thread(new AcqThread(*this))
{
  DEB_CONSTRUCTOR();
}
```

Very simple *AcqThread* class, real code implemented in *Camera*::*acqThreadFunction*

AcqThread is created in *Camera* constructor, *AutoPtr* will delete it on destruction

HW PLUGIN EXAMPLE: CAMERA

```
void Camera::acqThreadFunction()
{
  DEB_MEMBER_FUNCT();

  AutoMutex l(m_cond.mutex());

  TrigMode trig_mode = m_trig_mode;
  int acq_nb_frames = m_nb_frames;
  double acq_exp_time = m_exp_time;
  double acq_lat_time = m_lat_time;

  m_status = Ready;
  m_cond.signal();
```

Most of *AcqThread* code works with *Mutex* locked to protect status & synch. variables

Make a local copy of parameters and notify calling thread that we are already operational

HW PLUGIN EXAMPLE: CAMERA

```
auto end = [&]() {
  bool endless = (acq_nb_frames == 0);
  return m_stopped || (!endless && (m_acquired == acq_nb_frames));
};

StdBufferCbMgr& buffer_cb_mgr = m_buffer.getBuffer();
buffer_cb_mgr.setStartTimestamp(Timestamp::now());

for (m_acquired = 0; !end(); ++m_acquired) {
  DEB_TRACE() << "Starting frame " << m_acquired << " exposure ...";
  m_status = Exposure;
  {
    AutoMutexUnlock u(l);
    Sleep(acq_exp_time);
    fillImage(m_acquired);
  }
  if (m_stopped)
    break;
  m_status = Readout;
  {
    AutoMutexUnlock u(l);
    Sleep(acq_lat_time);
  }
  DEB_TRACE() << "Frame " << m_acquired << " ready!";
  HwFrameInfo finfo;
  finfo.acq_frame_nb = m_acquired;
  {
    AutoMutexUnlock u(l);
    buffer_cb_mgr.newFrameReady(finfo);
  }
}
```

Endless ACQs must be explicitly stopped

Buffer start timestamp must be set, frame timestamp is relative

AutoMutexUnlock temporarily releases the *Mutex* during lengthy/blocking operations

Fill the buffer with new image

Shared variables are always protected by mutex

Inject new frame into Lima processing chain

HW PLUGIN EXAMPLE: CAMERA

```
  m_status = Ready;
  DEB_TRACE() << "Acquisition finished: frames=" << m_acquired;
}
```

The *AcqThread* ends after last frame or *stopAcq.* It must be joined on next *prepareAcq*

HW PLUGIN EXAMPLE: CAMERA

```
void Camera::fillImage(int frame_nb)
{
  DEB_MEMBER_FUNCT();
  DEB_PARAM() << DEB_VAR1(frame_nb);

  const int max_val = 255;
  const Size size = getFrameDim().getSize();
  const int width = size.getWidth();
  const int height = size.getHeight();
  const Point center(width / 2, height / 2);
  const int fwhm = std::min(width, height) / 2;
  const double sigma_fwhm = 0.424661; // 1/(2*sqrt(2*ln(2)))
  const double sigma2 = std::pow(sigma_fwhm * fwhm, 2);

  StdBufferCbMgr& buffer_cb_mgr = m_buffer.getBuffer();
  char *p = static_cast<char *>(buffer_cb_mgr.getFrameBufferPtr(frame_nb));
  double a = double(max_val) / m_cycle_frames * (frame_nb % m_cycle_frames + 1);
  for (int y = 0; y < height; ++y) {
    for (int x = 0; x < width; ++x) {
      Point c = Point(x, y) - center;
      double r2 = std::pow(c.x, 2) + std::pow(c.y, 2);
      double v = a * std::exp(-r2 / sigma2);
      *p++ = std::round(v);
    }
  }
}
```

Spot centered
In image

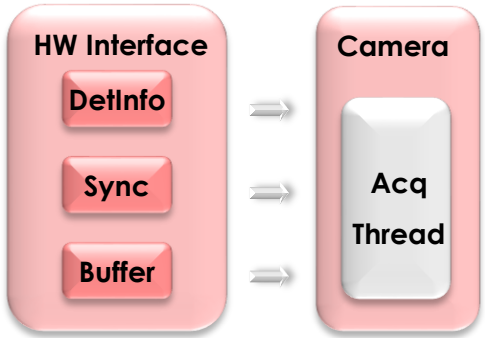
Get pointer to
Frame buffer

Cycle on peak
intensity

HW PLUGIN EXAMPLE: HW INTERFACE

Camera code is ready with the desired functionality

Let's link the HW interface and control objects to it!



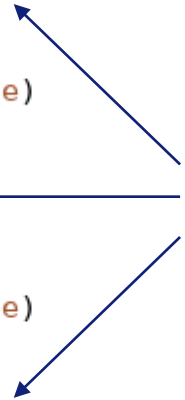
HW PLUGIN EXAMPLE: DETINFO

```
/// Control object providing detector info interface
class MYEXAMPLE_EXPORT DetInfoCtrlObj : public HwDetInfoCtrlObj {
  DEB_CLASS_NAMESPC(DebModCamera, "DetInfoCtrlObj", "MyExample");

public:
  DetInfoCtrlObj(Camera &cam) : m_cam(cam) {}

  virtual void getMaxImageSize(Size &max_image_size);
  virtual void getDetectorImageSize(Size &det_image_size);

  virtual void getDefImageType(ImageType &def_image_type);
  virtual void getCurrImageType(ImageType &curr_image_type);
  virtual void setCurrImageType(ImageType curr_image_type);

  virtual void getPixelSize(double &x_size, double &y_size);
  virtual void getDetectorType(std::string &det_type);
  virtual void getDetectorModel(std::string &det_model);

  virtual void registerMaxImageSizeCallback(HwMaxImageSizeCallback &cb);
  virtual void unregisterMaxImageSizeCallback(HwMaxImageSizeCallback &cb);

private:
  Camera &m_cam;
};
```

Mandatory methods already declared

HW PLUGIN EXAMPLE: DETINFO

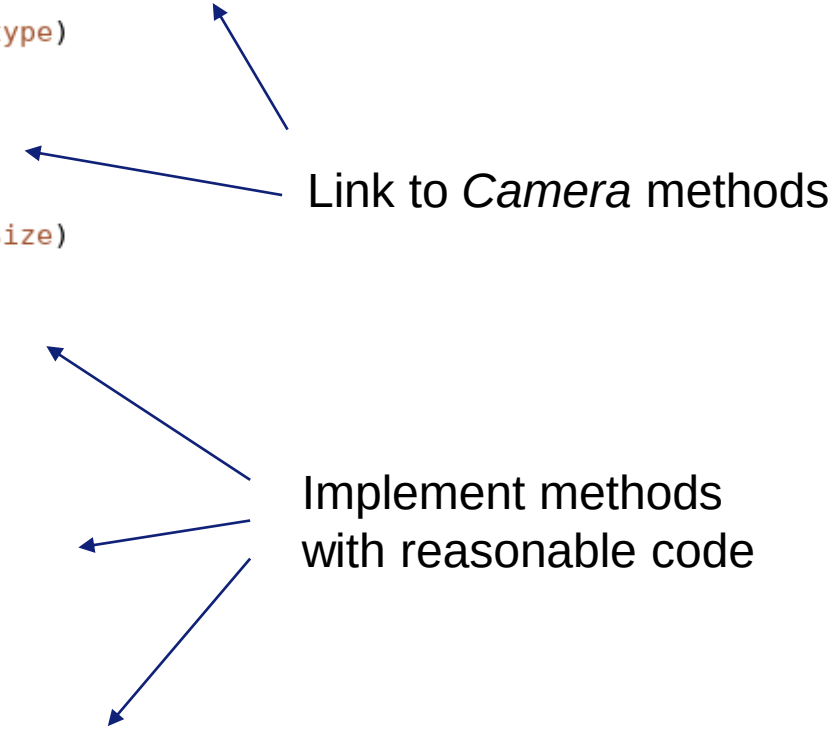
```
void DetInfoCtrlObj::getMaxImageSize(Size &max_image_size)
{
  DEB_MEMBER_FUNCT();
  max_image_size = m_cam.getFrameDim().getSize();
}

void DetInfoCtrlObj::getDetectorImageSize(Size &det_image_size)
{
  DEB_MEMBER_FUNCT();
  getMaxImageSize(det_image_size);
}

void DetInfoCtrlObj::getDefImageType(ImageType &def_image_type)
{
  DEB_MEMBER_FUNCT();
  def_image_type = m_cam.getFrameDim().getImageType();
}
```



Link to *Camera* methods

HW PLUGIN EXAMPLE: DETINFO

```
void DetInfoCtrlObj::setCurrImageType(ImageType curr_image_type)
{
  DEB_MEMBER_FUNCT();
  if (curr_image_type != m_cam.getFrameDim().getImageType())
    THROW_HW_ERROR(NotSupported) << "Invalid image type: " << curr_image_type;
}

void DetInfoCtrlObj::getCurrImageType(ImageType &curr_image_type)
{
  DEB_MEMBER_FUNCT();
  getDefImageType(curr_image_type);
}

void DetInfoCtrlObj::getPixelSize(double &x_size, double &y_size)
{
  DEB_MEMBER_FUNCT();
  x_size = y_size = 100e-6;
}

void DetInfoCtrlObj::getDetectorType(std::string &det_type)
{
  DEB_MEMBER_FUNCT();
  det_type = "MyExample";
}

void DetInfoCtrlObj::getDetectorModel(std::string &det_model)
{
  DEB_MEMBER_FUNCT();
  det_model = "BasicOne";
}
```



HW PLUGIN EXAMPLE: SYNC

```
/// Control object providing camera synchronization interface
class MYEXAMPLE_EXPORT SyncCtrlObj : public HwSyncCtrlObj {
  DEB_CLASS_NAMESPC(DebModCamera, "ShutterCtrlObj", "MyExample");

public:
  SyncCtrlObj(Camera &simu);
  virtual ~SyncCtrlObj();

  virtual bool checkTrigMode(TrigMode trig_mode);
  virtual void setTrigMode(TrigMode trig_mode);
  virtual void getTrigMode(TrigMode &trig_mode);

  virtual void setExpTime(double exp_time);
  virtual void getExpTime(double &exp_time);

  virtual void setLatTime(double lat_time);
  virtual void getLatTime(double &lat_time);

  virtual void setNbHwFrames(int nb_frames);
  virtual void getNbHwFrames(int &nb_frames);

  virtual void getValidRanges(ValidRangesType &valid_ranges);

private:
  Camera &m_cam;
};
```

Mandatory methods already declared

HW PLUGIN EXAMPLE: SYNC

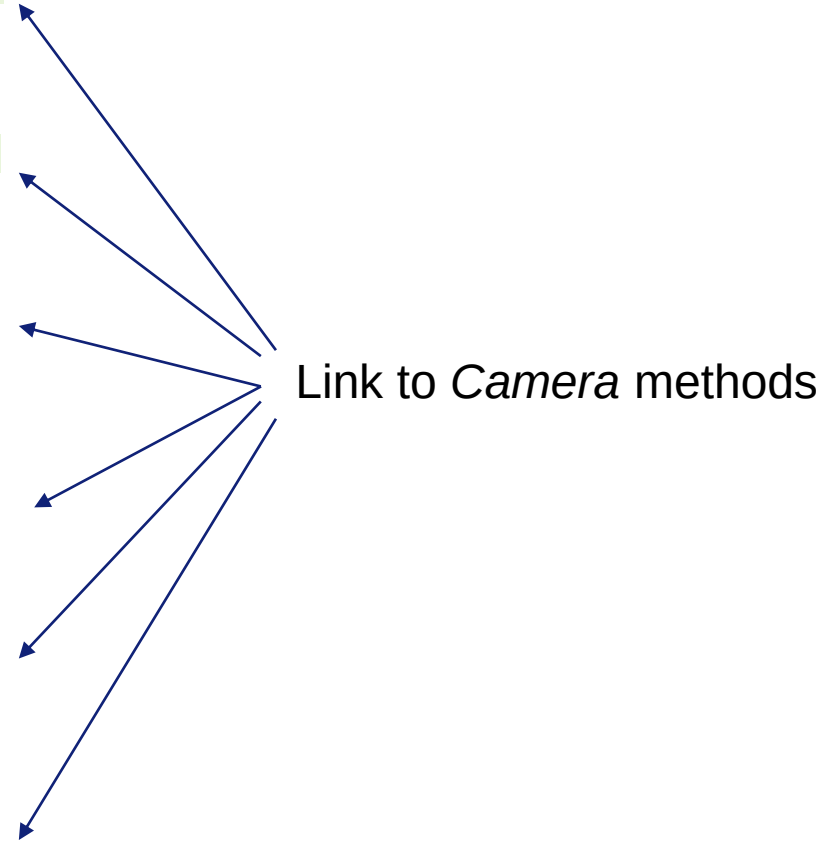
```
bool SyncCtrlObj::checkTrigMode(TrigMode trig_mode)
{
  switch (trig_mode) {
  case IntTrig:
    return true;
  case IntTrigMult:
  case ExtTrigSingle:
  case ExtTrigMult:
    return true;
  default:
    return false;
  }
}

void SyncCtrlObj::setTrigMode(TrigMode trig_mode)
{
  if (!checkTrigMode(trig_mode)) throw LIMA_HW_EXC(InvalidValue, "Invalid (exter
nal) trigger");
}

void SyncCtrlObj::getTrigMode(TrigMode &trig_mode)
{
  trig_mode = IntTrig;
}
```

Only support *IntTrig*

Default template code also support more modes

HW PLUGIN EXAMPLE: SYNC

```
void SyncCtrlObj::setExpTime(double exp_time)
{
  m_cam.setExpTime(exp_time);
}

void SyncCtrlObj::getExpTime(double &exp_time)
{
  m_cam.getExpTime(exp_time);
}

void SyncCtrlObj::setLatTime(double lat_time)
{
  m_cam.setLatTime(lat_time);
}

void SyncCtrlObj::getLatTime(double &lat_time)
{
  m_cam.getLatTime(lat_time);
}

void SyncCtrlObj::setNbHwFrames(int nb_frames)
{
  m_cam.setNbFrames(nb_frames);
}

void SyncCtrlObj::getNbHwFrames(int &nb_frames)
{
  m_cam.getNbFrames(nb_frames);
}
```



HW PLUGIN EXAMPLE: SYNC

```
void SyncCtrlObj::getValidRanges(ValidRangesType &valid_ranges)
{
  double min_time           = 10e-9;
  double max_time           = 1e6;
  valid_ranges.min_exp_time = min_time;
  valid_ranges.max_exp_time = max_time;
  valid_ranges.min_lat_time = min_time;
  valid_ranges.max_lat_time = max_time;
}
```

We can leave the default template values

HW PLUGIN DESIGN: INTERFACE

```
class MYEXAMPLE_EXPORT Interface : public HwInterface
{
  DEB_CLASS_NAMESPC(DebModCamera, "MyExampleInterface", "MyExample");

public:
  Interface(Camera&);
  virtual ~Interface();
  //- From HwInterface
  virtual void        getCapList(CapList&) const;
  virtual void        reset(ResetLevel reset_level);
  virtual void        prepareAcq();
  virtual void        startAcq();
  virtual void        stopAcq();
  virtual void        getStatus(StatusType& status);
  virtual int         getNbHwAcquiredFrames();

private:
  Camera&             m_cam;
  DetInfoCtrlObj      m_det_info;
  SyncCtrlObj         m_sync;
  CapList             m_cap_list;
};
```

Mandatory methods already declared

DetInfo/SyncCtrlObj & HW capability list

HW PLUGIN DESIGN: INTERFACE

```
Interface::Interface(Camera& cam) :
  m_cam(cam)
  m_cam(cam), m_det_info(cam), m_sync(cam)
{
  DEB_CONSTRUCTOR();

  HwDetInfoCtrlObj *det_info = &m_det_info;
  m_cap_list.push_back(HwCap(det_info));

  HwSyncCtrlObj *sync = &m_sync;
  m_cap_list.push_back(HwCap(sync));

  HwBufferCtrlObj *buffer = &cam.getBufferCtrlObj();
  m_cap_list.push_back(HwCap(buffer));
}

Interface::~Interface()
{
  DEB_DESTRUCTOR();
}

void Interface::getCapList(CapList &cap_list) const
{
  cap_list = m_cap_list;
}
```

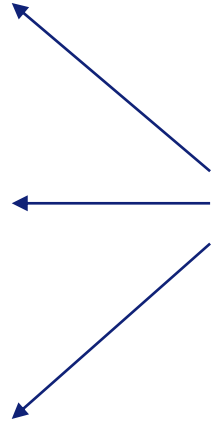
Fill HW capability list with mandatory control objects

Get *HwBufferCtrlObj* from *Camera*

Export HW capability list

HW PLUGIN DESIGN: INTERFACE

```
void Interface::prepareAcq()
{
  DEB_MEMBER_FUNCT();
  m_cam.prepareAcq();
}

void Interface::startAcq()
{
  DEB_MEMBER_FUNCT();
  m_cam.startAcq();
}

void Interface::stopAcq()
{
  DEB_MEMBER_FUNCT();
  m_cam.stopAcq();
}
```



Link to *Camera* methods

HW PLUGIN DESIGN: INTERFACE

```
void Interface::getStatus(StatusType& status)
{
  DEB_MEMBER_FUNCT();
  switch (m_cam.getStatus()) {
  case Camera::Init:
  case Camera::Ready:
    status.set(StatusType::Basic::Ready);
    break;
  case Camera::Exposure:
    status.set(StatusType::Basic::Exposure);
    break;
  case Camera::Readout:
    status.set(StatusType::Basic::Readout);
    break;
  default:
    THROW_HW_ERROR(Error) << "Invalid status";
  }
}

int Interface::getNbHwAcquiredFrames()
{
  DEB_MEMBER_FUNCT();

  // TODO: get the number of acquired frames
  int acq_frames = 0;

  return acq_frames;
  return m_cam.getNbAcqFrames();
}
```

Convert from *Camera* status to Lima status

Again, link to *Camera*

HW PLUGIN DESIGN: EXAMPLE

Create lima environment …

```
(base) ~/devel$ (mamba create -n lima
                                lima-camera-simulator-tango
                                cmake gxx_linux-64 ninja pytest)
```

… compile …

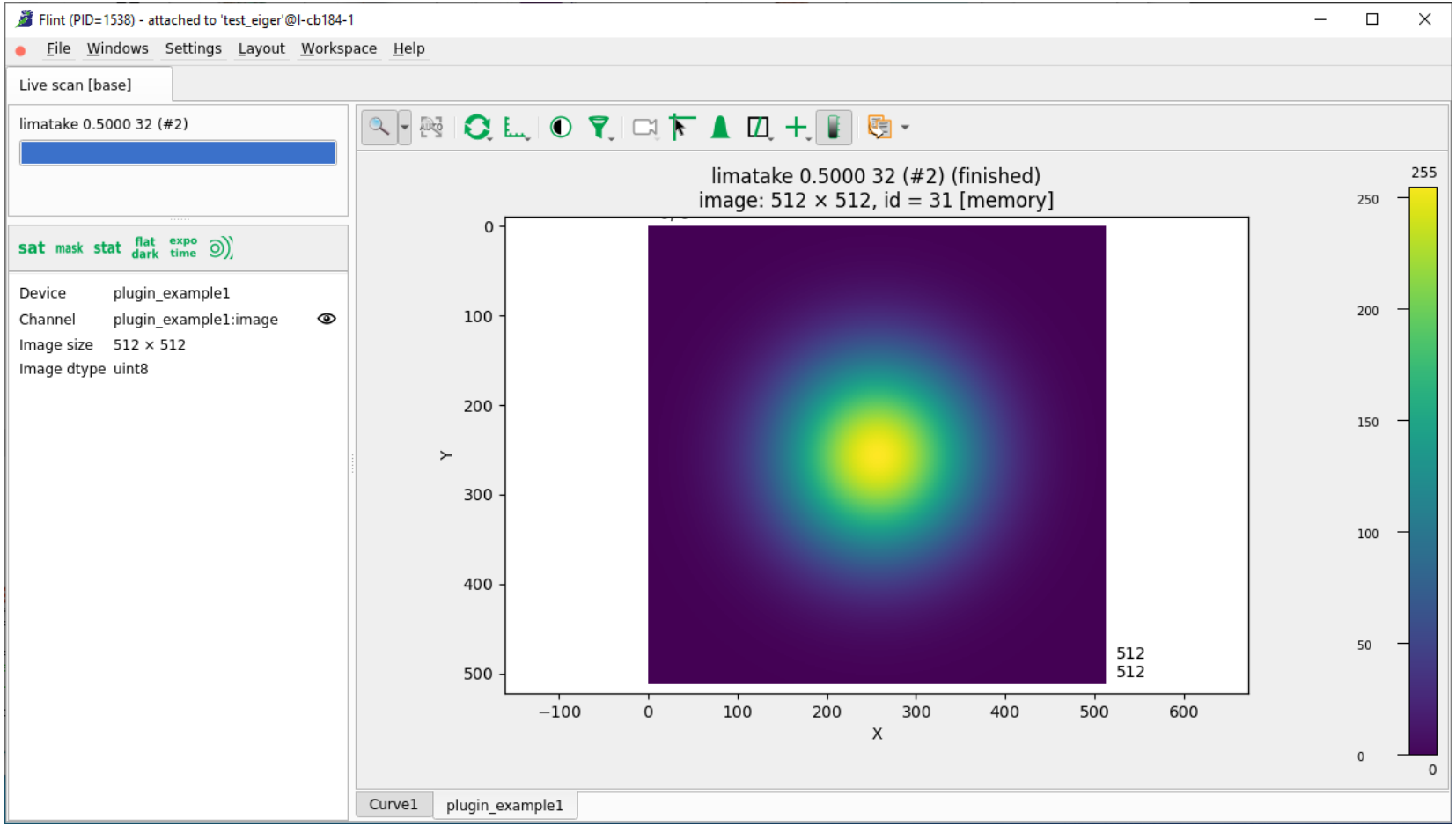
```
(lima) ~/devel/myexample$ (PREFIX=${CONDA_PREFIX}
                           bash -e conda/camera/build.sh)
```

… remove *IntTrigMult* & *ExtTrigSingle* tests in test_apy.py …

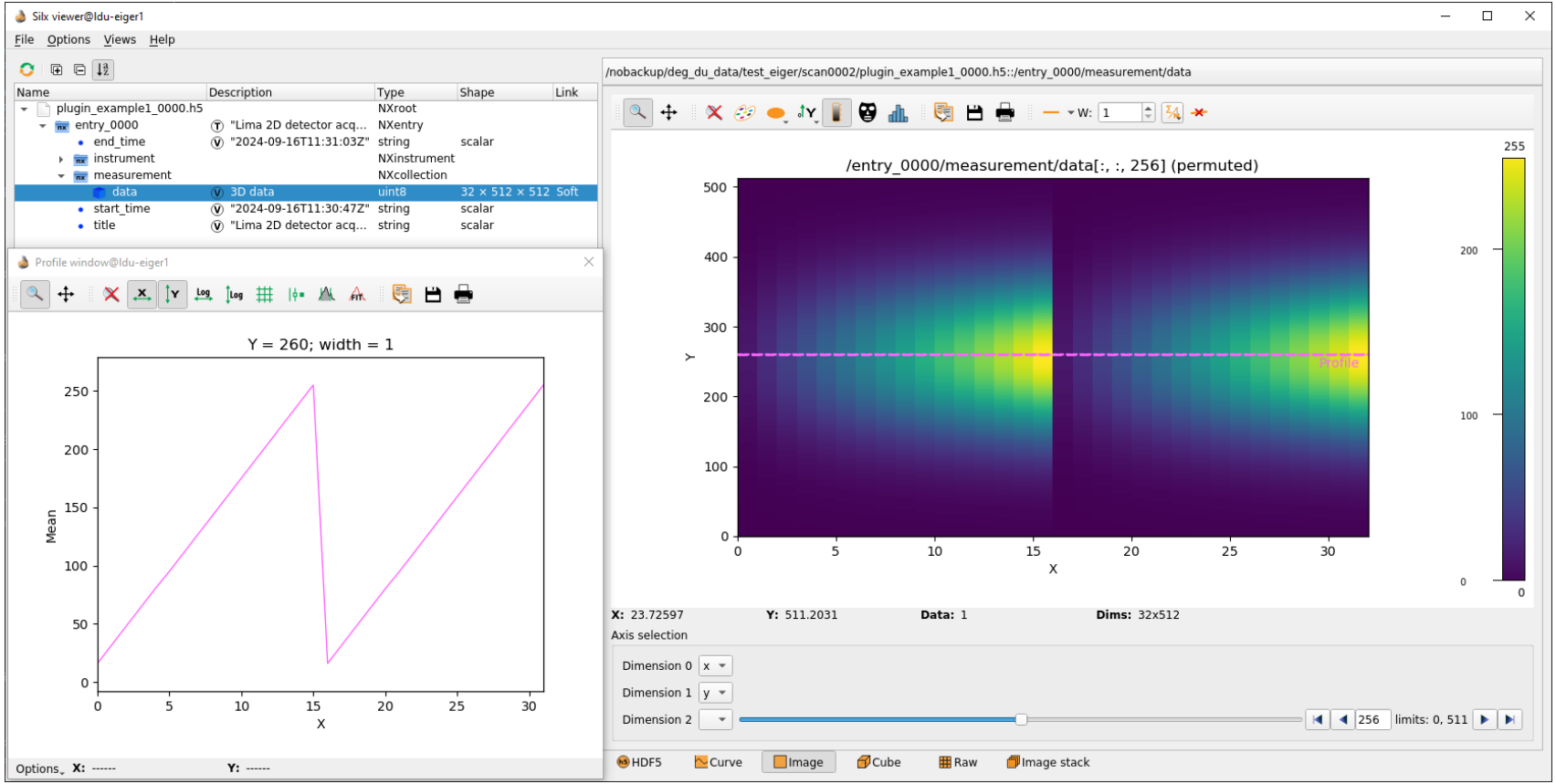
… and test …

```
(lima) ~/devel/myexample$ pytest test/test_api.py
```

HW PLUGIN DESIGN: GENERATED DATA



HW PLUGIN DESIGN: GENERATED DATA



THE END

Thank you very much!!