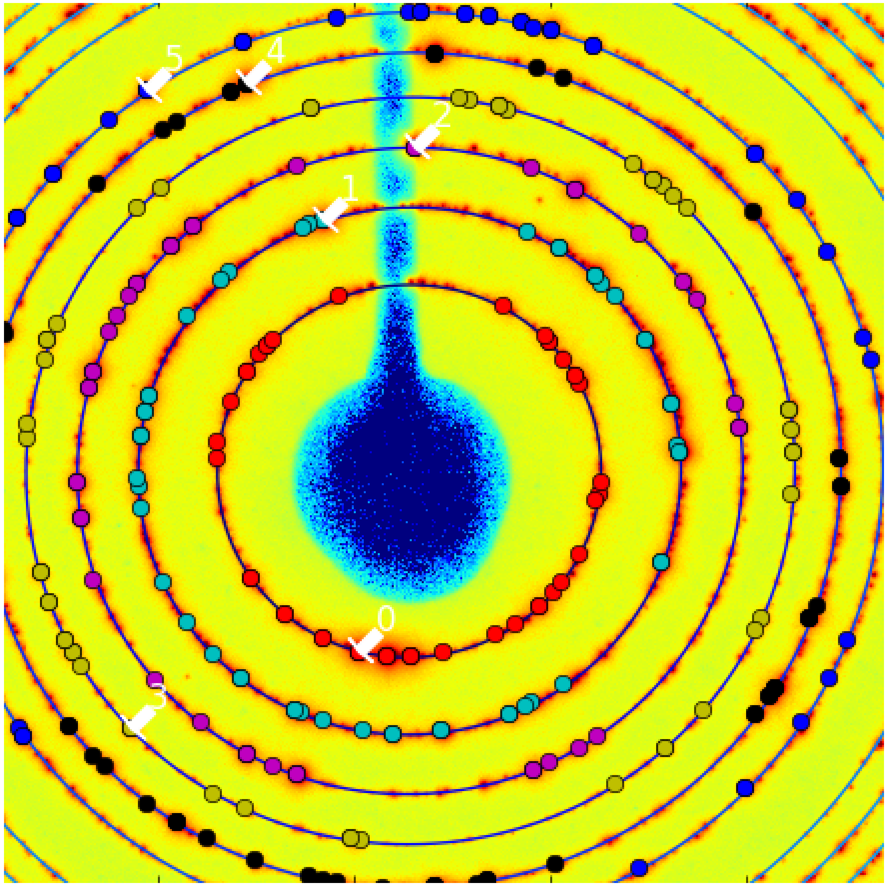
ESRF | The European Synchrotron

Fast azimuthal integration … in Python

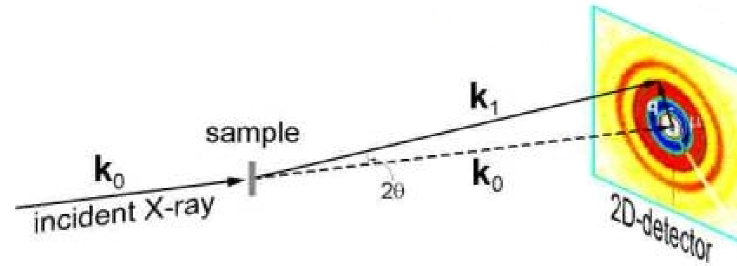


While speed is only needed at large facilities …
… proper calculation is needed for any scientific application

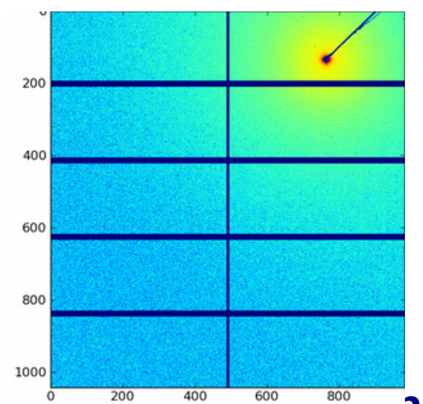
Introduction to PyFAI

Introduction to Azimuthal integration

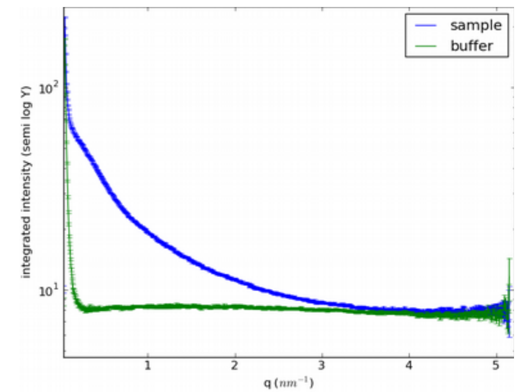
- **Allows the use of area detectors for**
  - Small angle scattering
  - Powder diffraction, PDF, ...



- **Better harvesting of X-ray photons (large solid angle)**



Azimuthal integration



- **The devil is hidden in the details (of implementation)**

- **PyFAI is:**
  - Open source
  - Open to contribution
  - Open to discussion
  - Free
  - Fast

But many other tools exists:
- FIT2D
- DataSqueeze
- XRDUA
- Foxtrot
- Maud
- GSAS-II

Concepts in PyFAI

http://pyfai.readthedocs.io/en/latest/pyFAI.html#experiment-description

- **Image**

  2D array of pixels, often read using the FabIO library.

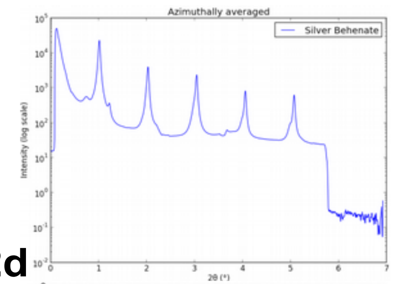
- **Stack of images**

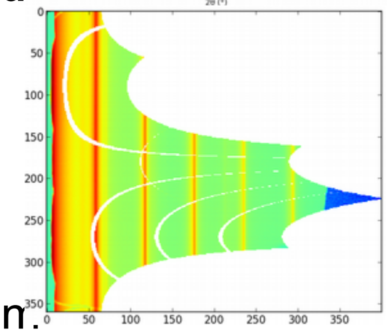
  3D volume composed of a list of images. Read using HDF5

- **Azimuthal integrator**

  Core pyFAI object which can transform an image into:

  - **powder diagram using integrate1d**
  - **“cake” image, azimuthally regrouped using integrate2d**





- **Detector**

  Calculates the pixel position and mask, flat, ...

- **Geometry**

  Position of the detector from the sample & incoming beam.

- **PONI-file**

  Small text file with the detector description and the geometry.
  Loaded by the azimuthal integrator

PyFAI is a library on which applications are built on

Library	≠	**Graphical application**
– Re-usable code		– Easier to use
– Needs the definition of an API		– Looks better
– Faster to develop		– Only one application
– Easier to test and maintain		– Code not re-usable

- **PyFAI is itself relying on the Scientific Python stack:**
  - Numpy
  - Scipy
  - Matplotlib
  - H5Py
  - Cython
  - FabIO

+PyQt, for the graphical part
+ silx (soon)

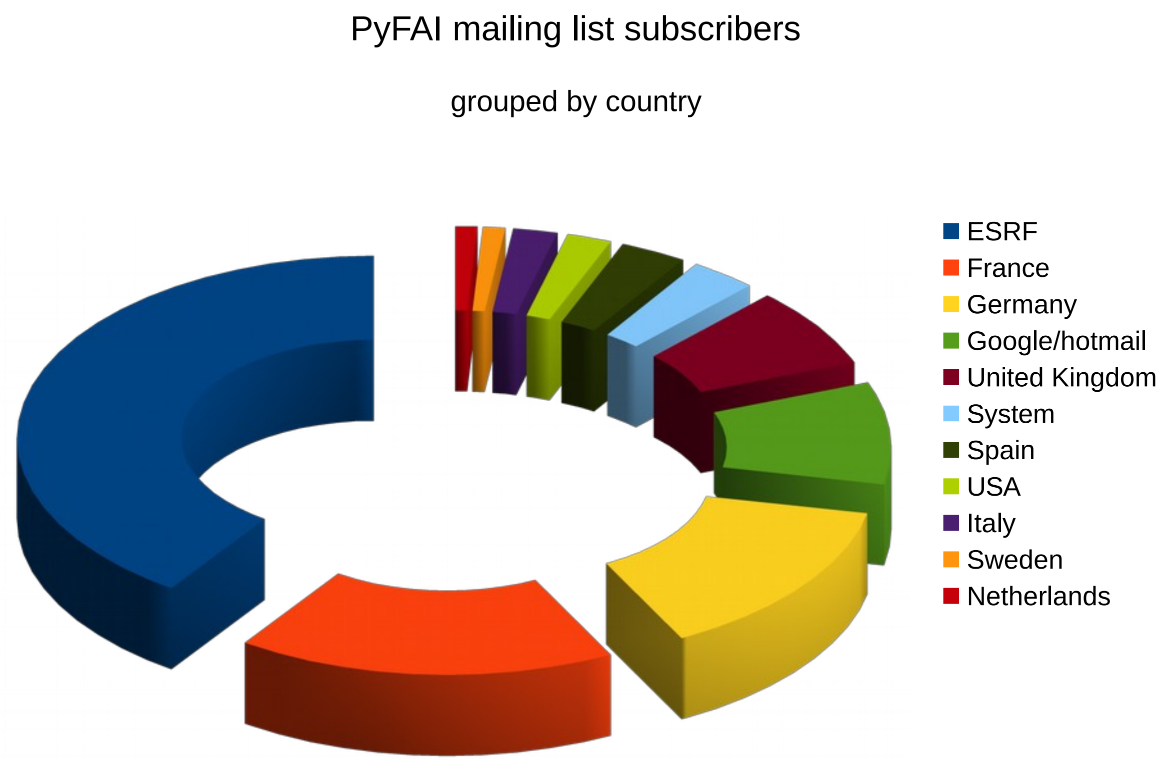
Examples of application relying on pyFAI

- **NanoPeakCell: Serial crystallography pre-processing**
  - Nicolas Coquelle, IBS Grenoble
- **PySaxs: data analysis for SAXS experimental station**
  - Olivier Tache, CEA Saclay
- **Dpdak: online data analysis for Saxs data**
  - Gunthard Benecke, Petra III
- **Dioptas: offline data analysis for high pressure diffraction**
  - Clemens Percher, APS → Germany
- **Bubble: online data analysis for Saxs/Waxs data**
  - Vadim Diadkin, Dubble & SNBL CRG beamlines, now ID11
- **Project for materials and strain analysis**
  - Jozef Keckes, Loeben university, Austria
- **xPDFsuite**
  - Prof. Simon Billinge, U. of Columbia

http://pyfai.readthedocs.io/en/latest/ecosystem.html

User community of pyFAI

- **PyFAI is used in most European and American synchrotons/FELs**



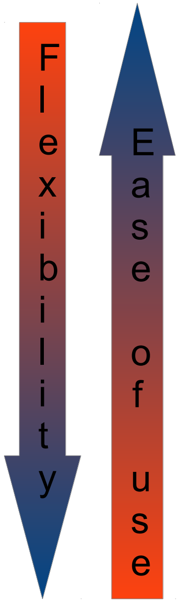
- **User support is provided via the mailing list: pyFAI@esrf.fr**
  - Direct contact with authors is discouraged

    https://pythonhosted.org/pyFAI/project.html#getting-help

Layers in pyFAI

- **Applications level:**
  - GUI applications: pyFAI-calib, pyFAI-integrate, diff_map
  - Scriptable applications:pyFAI-average, pyFAI-saxs, pyFAI-waxs, diff_tomo, …

- **Python interface:**
  - Top level: azimuthal integrator
  - Mid level: calibrant, detector, geometry, calibration
  - Low level: rebinning/histogramming engines (Cython or OpenCL)



- **Question: how to define the right balance ?**

  It is up to you !

Description of a few application in pyFAI:

- **Preprocessing**
- **Mask drawing tool**
- **Calibration**
- **Integration**
- **Diffraction mapping**
- **…**

Image pre-processing: *pyFAI-average*

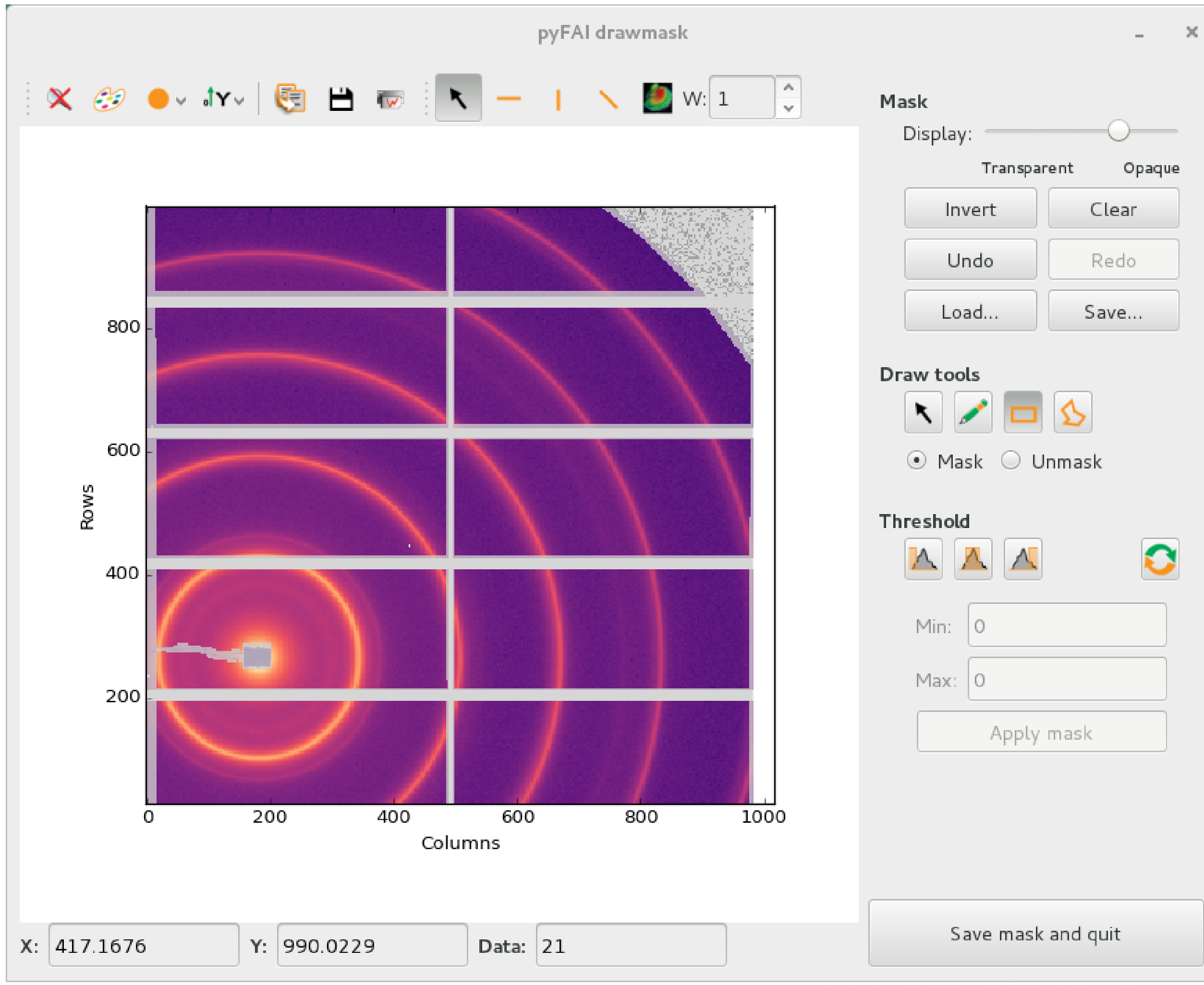
- **A tool for filtering a stack of images :**
  - Used to merge multiple input images (can be a multiframe nexus)
  - Merging methods available:

    min, max, mean, std, median, sum, quantiles, cutoff
  - Correct for dark-current & flat-field
  - Normalize for a monitor value (from headers)
  - Exports in multiple formats (see FabIO)
    - **Can be used to convert image format (NeXus → TIF)**

http://www.silx.org/doc/pyFAI/man/pyFAI-average.html

Mask drawing tool: pyFAI-drawmask

- **First application relying on *silx* (still compatible with PyMca)**



Contribution from Valentin Valls

Calibration: pyFAI-calib

- **The determination of the geometry is also known as calibration**
  - The prerequisite is:
    - **detector geometry and mask,**
    - **calibrant (LaB6, CeO2, AgBh, …)**
    - **wavelength or energy used**
  - Only the position of the detector and the rotation needs to be refined:
    - **3 translations: dist, poni1 and poni2**
    - **3 rotations: rot1, rot2, rot3**
- **PyFAI assumes this setup does not change during the experiment**
- **It is divided into 4 major steps:**
  - Extraction of groups of peaks
  - Identification of peaks and groups of peaks belonging to same ring
  - Least-squares refinement of the geometry parameters on peak position
  - Validation by an human being of the geometry

http://pyfai.readthedocs.io/en/latest/usage/cookbook/calibrate.html

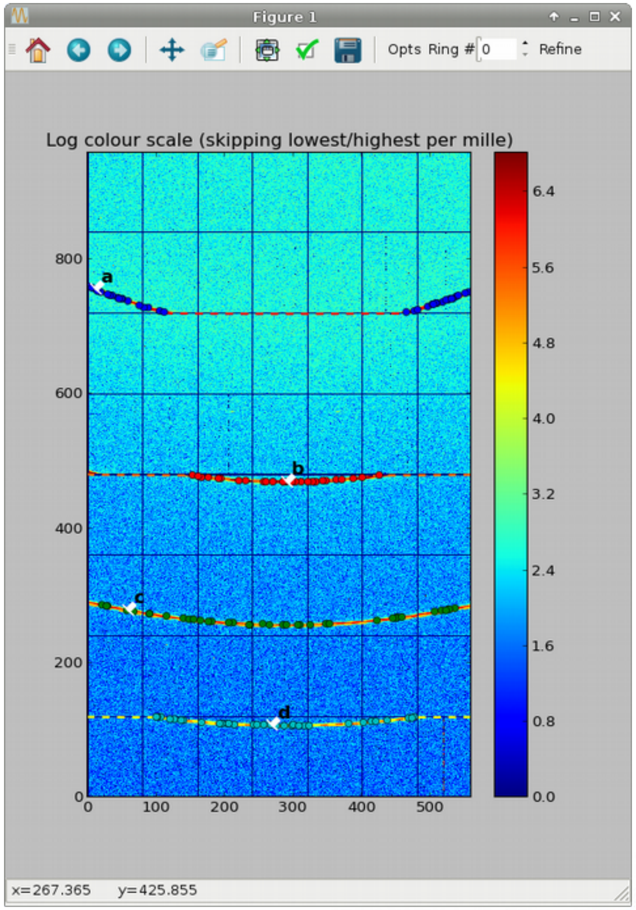
Detectors

- **Detector are 2D array of pixel, they contain:**
  - pixel size
  - mask
  - A way to calculate where a pixel is located in space (3D)
- **PyFAI provides 120 (56 unique) detectors pre-defined**
  - Dectris, ImXpad, Rayonix, Dexela, Perkin-Elmer, …
- **Detectors can easily be specialized:**
  - With their specific masks
  - With their specific pixel positions
  - Then saved to a NeXus file
- **Detector can be contiguous or not ...**
- **Detectors can be flat or not ...**

Example of non-contiguous detectors:

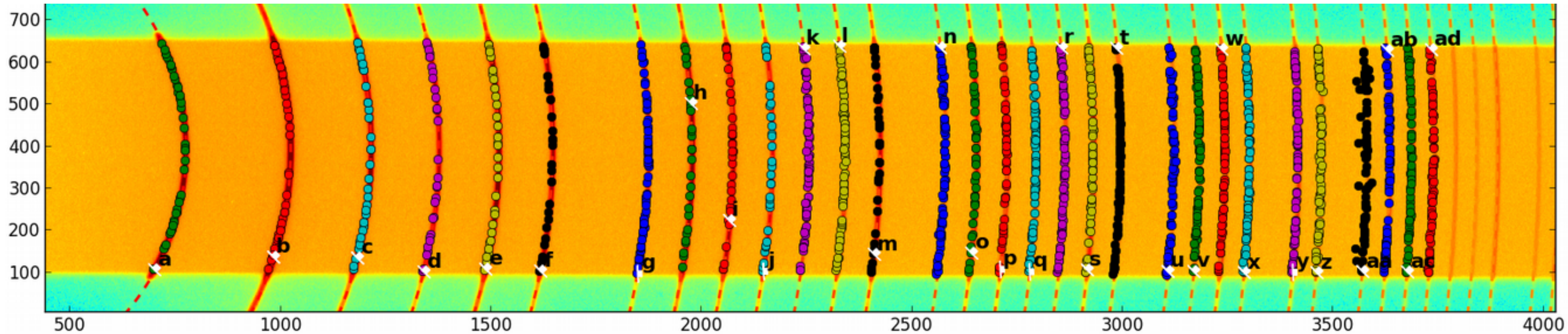
- **Xpad are module based pixel-detectors**
  - The S540 is 8 strips of 7 modules each
  - Gaps between modules within a strip are small (few pixels)
  - Gaps between strips are large (hundreds of pixels)

- **Can be challenging to calibrate !**
  - Calibrant: LaB6 at 18.57keV

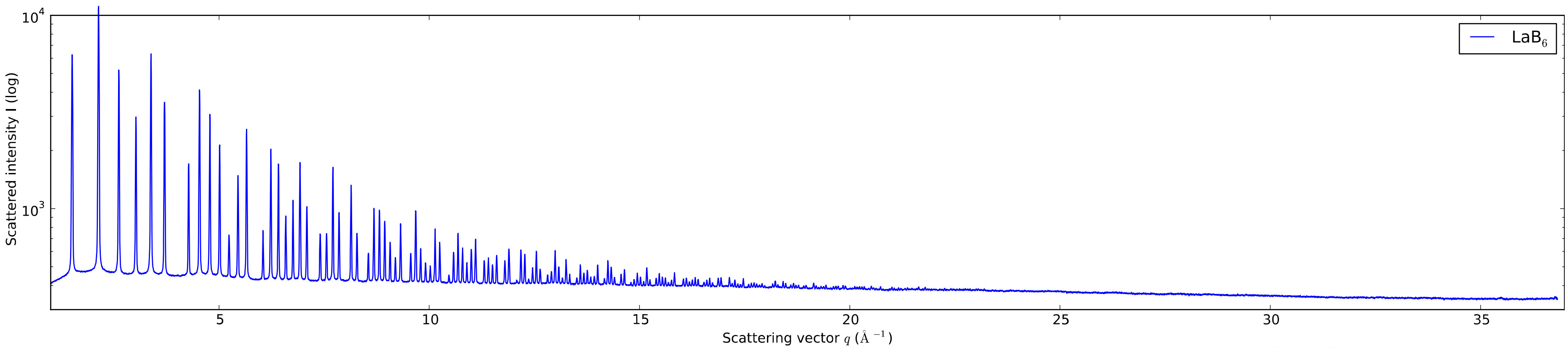


Example of non-planar detector: cylindrical

- **Every pixel has its own geometry**
- **Hemi-cylindrical detector based on a bent imaging plate:**
  - Calibration of such detector is naturally possible with pyFAI



Courtesy of U. Aarhus



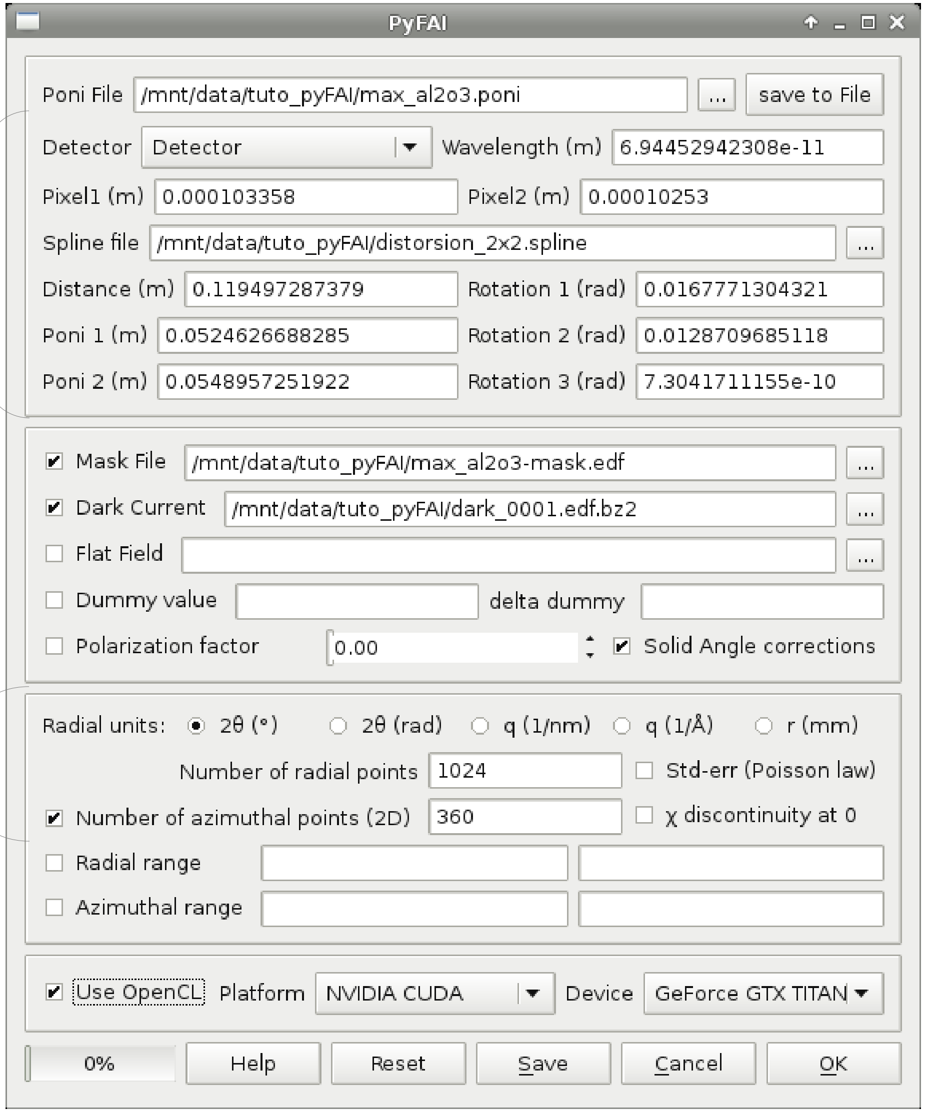
Calibrants: provide aperture of Debye-Scherrer cones

- **PyFAI ships 15 reference samples (decreasing 2θ of first ring) + variants:**
  - Au: Gold
  - ZnO: Blende
  - CeO2: Ceria
  - Si: Silicon
  - NaCl: Salt
  - alpha_Al2O3: Corundum
  - Cristobaltite and Quartz (SiO2)
  - Cr2O3 and CrOx : Chromium oxide (the later being the undefined oxide used on MX beamlines)
  - LaB6: Lantanide hexaboride
  - PBBA: Para Bromo Benzoic Acid
  - C14H30O: tetradecanol
  - AgBh: Silver Behenate
- **But you can provide your d-spacing file if you prefer:**
  - Ascii text files with d-spacing written in Angstrom (like FIT2D)
  - Use the American Minaralogist database:
    - **http://rruff.geo.arizona.edu/AMS/amcsd.php**

Azimuthal integration tool: pyFAI-integrate

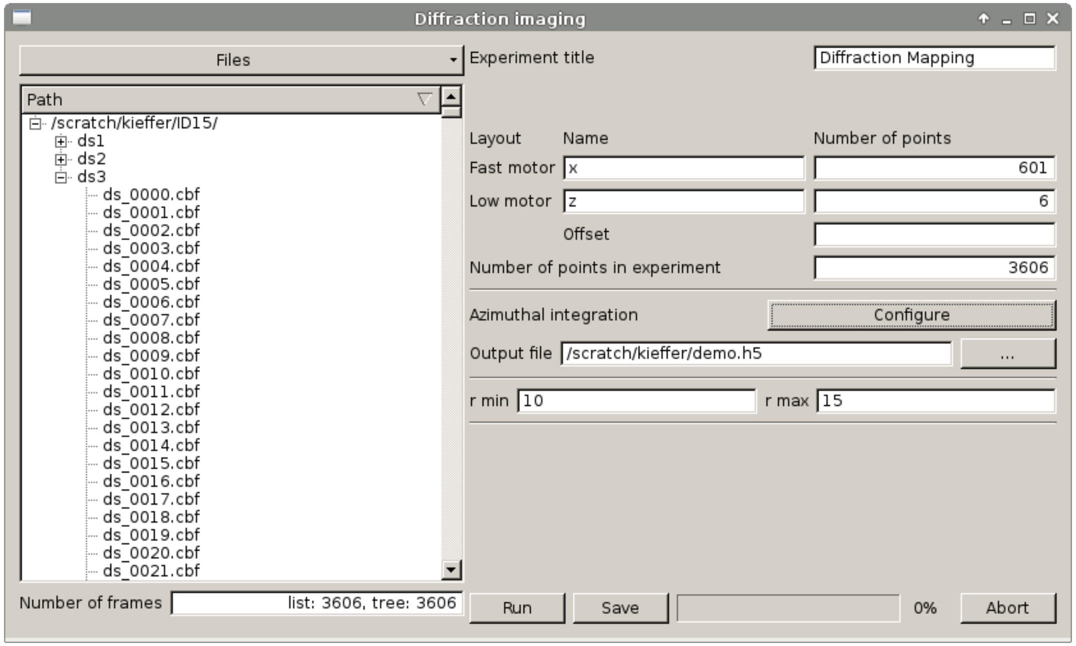
From PONI file

Define the output space



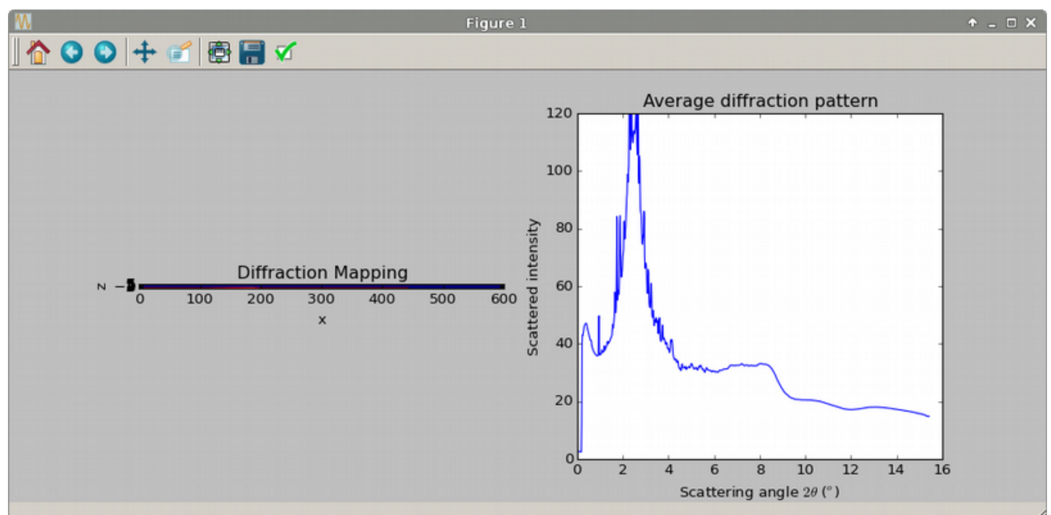
Can now be used in command line mode without Qt

Diffraction imaging offline tool: diff-map



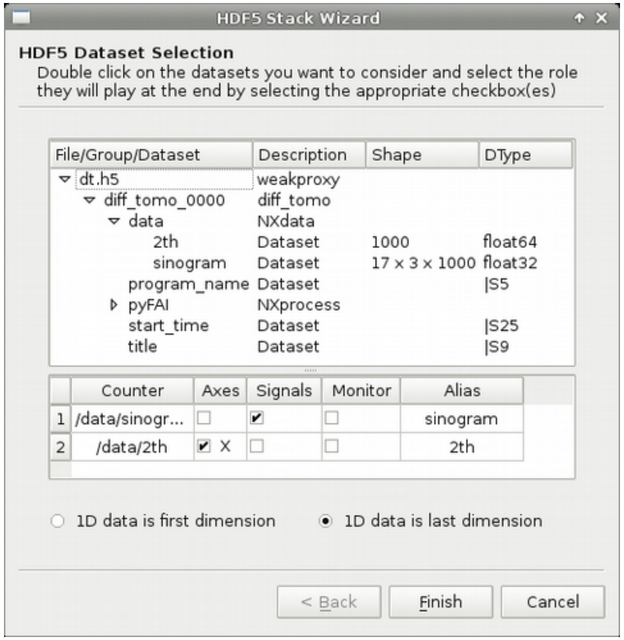
Created as part of the IR-drx2015 project

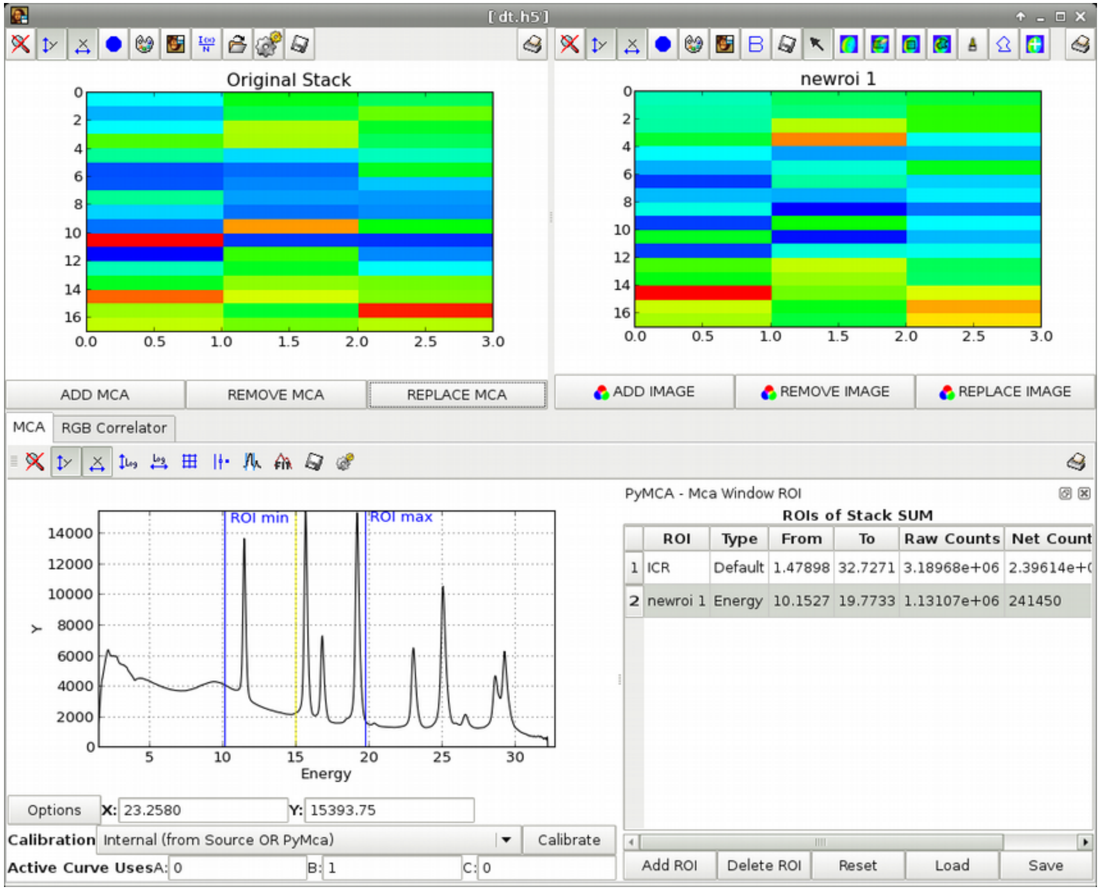
Produces NeXus files



Diffraction imaging HDF5 Visualization

- **Visualize and analyze 3D stack using pymcaroitool**





Subsequent analysis are based on PCA and other multivariate analysis …

Why a library rather than an application ?

- **An application for diffraction purposes already exists:**
  - And it has been around for 20 years: FIT2D
- **But this application was not flexible enough !**
  - To be integrated into a beamline acquisition scheme
  - To test new ideas (easily)

    → This is why pyFAI was started in 2011
- **A library is easier to:**
  - Test: thanks to a testing framework
  - Develop: no need to master GUI programming
  - Maintain over the years (>10y life-cycle)
- **A library does not prevent GUIs, ...**
  but ensures a clear separation of logic and processing
- **Many tools can be easily developed and put in a toolkit**
  - Following the UNIX philosophy: many tools, one for each task.

Description of the Python API

- **Top level API:**
  - AzimuthalIntegrator
    - **Method for azimuthal averaging: integrate1d**
    - **Method for azimuthal regrouping: integrate2d**
  - Distortion
    - **Correct and uncorrect methods**
- **Mid level API:**
  - Geometry: Parent class of AzimuthalIntegrator
  - Detector: Calculate the pixel position & masks
  - Calibrant: provide 2θ as function of the wavelength
- **Low level API: different rebinning engines**
  - OCL_LUT_Integrator, OCL_CSR_Integrator, ...
  - SplitBBoxLUT, splitBBoxCSR, ...

What happens during an integration

1) **Get the pixel coordinates from the detector, in meter.**

   There are 3 coordinates par pixel corner, and usually 4 corners per pixel.

   1Mpix image → 48 Mbyte !

2) **Offset the detector's origin to the PONI**

3) **Calculate the radial (2θ) and azimuthal (χ) positions of each corner**

4) **Assign each pixel to one or multiple bins.**

   If a look-up table is used, just store the fraction of the pixel.

   Then for each bin sum the content of all contributing pixels.

5) **Return bin position and associated intensities**

Azimuthal Integrator

Performs the azimuthal regrouping in 1&2D. Inherits Geometry, composes Detector, Integrators

- **Creation: import a PONI-file:**

  ai=pyFAI.load(ponifile)

- **Important methods (note many deprecated methods):**
  - Integrate1d; integrate2d; separate
- **Common arguments:**
  - *data (ndarray) – 2D array from the Detector/CCD camera*
  - *npt / (int) – number of points in the output pattern # npt_rad, npt_azim*
  - *filename (str) – output filename in 2/3 column ascii format*
  - *correctSolidAngle (bool) – correct for solid angle of each pixel if True*
  - *variance (ndarray) – array containing the variance of the data. If not available, no error propagation is done*
  - *error_model (str) – When the variance is unknown, an error model can be given: "poisson" (variance = I), "azimuthal" (variance = (I-<I>)^2)*
  - *radial_range ((float, float), optional) – The lower and upper range of the radial unit. If not provided, range is simply (data.min(), data.max()). Values outside the range are ignored.*
  - *azimuth_range ((float, float), optional) – The lower and upper range of the azimuthal angle in degree. If not provided, range is simply (data.min(), data.max()). Values outside the range are ignored.*
  - *mask (ndarray) – array (same size as image) with 1 for masked pixels, and 0 for valid pixels*
  - *dummy (float) – value for dead/masked pixels*
  - *delta_dummy (float) – precision for dummy value*
  - *polarization_factor (float) – polarization factor between -1 (vertical) and +1 (horizontal). 0 for circular polarization or random, None for no correction*
  - *dark (ndarray) – dark noise image*
  - *flat (ndarray) – flat field image*
  - *method (str) – can be "numpy", "cython", "BBox" or "splitpixel", "lut", "csr", "nosplit_csr", "full_csr", "lut_ocl" and "csr_ocl" if you want to go on GPU. To Specify the device: "csr_ocl_1,2"*
  - *unit (pyFAI.units.Enum) – Output units, can be "q_nm^-1", "q_A^-1", "2th_deg", "2th_rad", "r_mm" for now*
  - *safe (bool) – Do some extra checks to ensure LUT/CSR is still valid. False is faster.*
  - *normalization_factor (float) – Value of a normalization monitor*
- **Returns:**
  - Integrate result: looks like a tuple with intensity and bin-center coordinates

http://pyfai.readthedocs.io/en/latest/api/pyFAI.html#pyFAI.azimuthalIntegrator.AzimuthalIntegrator

Geometry

In charge of calculating the 2th/q/r/chi position for a point in space, handles array caching and locking. Contains the detector (composition)

- **Usage:**
  - Not directly: Usually via *ai* objects (inherited by AzimuthalIntegrator)
- **Important methods:**
  - calcfrom1d(tth,I): back-project powder pattern in a 2D image
  - get/set|PyFAI/SPD/Fit2D: exchange geometries with other programs
  - load(ponifile): instanciate geometry/aifrom a poni-file
  - reset(): empty all caches
- **Warning:**
  - may be re-implemented one day with pluggable geometry-engines to have them interchangeable

http://pyfai.readthedocs.io/en/latest/api/pyFAI.html#pyFAI.geometry.Geometry

Detector

Detector is a base-class defining any kind of 2D-detectors. There are about 56 specialized detectors: Pilatus, Xpad, Rayonix …

- **Usage: there is a factory to instantiate a detector from its name:**

  det = pyFAI.detector_factory("pilatus1M")

- **Important methods:**
  - get_mask(): calculate and cache the mask for this detector
  - save(nexusfile): save the detector configuration into HDF5
  - get_pixel_corners(): in cartesian position -> 4D array (Ny,Nx,Nc,3)

http://pyfai.readthedocs.io/en/latest/api/pyFAI.html#module-pyFAI.detectors

Geometry Refinement

Given a set of points (x,y) and associated ring number, refines the parameter of the PONI-file. Inherits from AzimuthalIntegrator. Contains a calibrant

- **Usage:**
  - Used by calibration
- **Important methods:**
  - Simplex, Refine1, Refine2: wraps scipy.optimize.fmin function
- **Warning:**
  - should not inherit from AzimuthalIntegrator but compose Geometry

http://pyfai.readthedocs.io/en/latest/api/pyFAI.html#pyFAI.geometryRefinement.GeometryRefinement

Calibrant

A calibrant is a reference compound where the d-spacings (interplanar distances) are known. The Calibrant class loads them from a file and contains the wavelength.

- **Usage:**
  - LaB6 = pyFAI.calibrant.ALL_CALIBRANT(“LaB6”)
  - Pt = pyFAI.calibrant.Calibrant(dspacing=[2.265,1.962,1.387,1.183,1.133])
- **Important method**
  - set_wavelength(1e-10): write once !!!!
  - get_2th(): get the position in 2theta of the reflection
  - fake_calibration_image(ai): simulate a calibration image given the geometry and the detector in ai

http://pyfai.readthedocs.io/en/latest/api/pyFAI.html#pyFAI.calibrant.Calibrant

Calibration

Command line interface for calibration

- **Usage:**
  - Used from pyFAI-calib script.
- **Alternative:**
  - There is a procedural interface to Calibration:

    ai = pyFAI.calibration.calib(img, calibrant, detector)
  - Can be used, for example, in ipython or NexPy

http://pyfai.readthedocs.io/en/latest/api/pyFAI.html#calibration-module

Distortion

Use the rebinning engines to perform distortion correction of detectors

- **Usage:**

  dis = pyFAI.distortion.Distortion(detector)

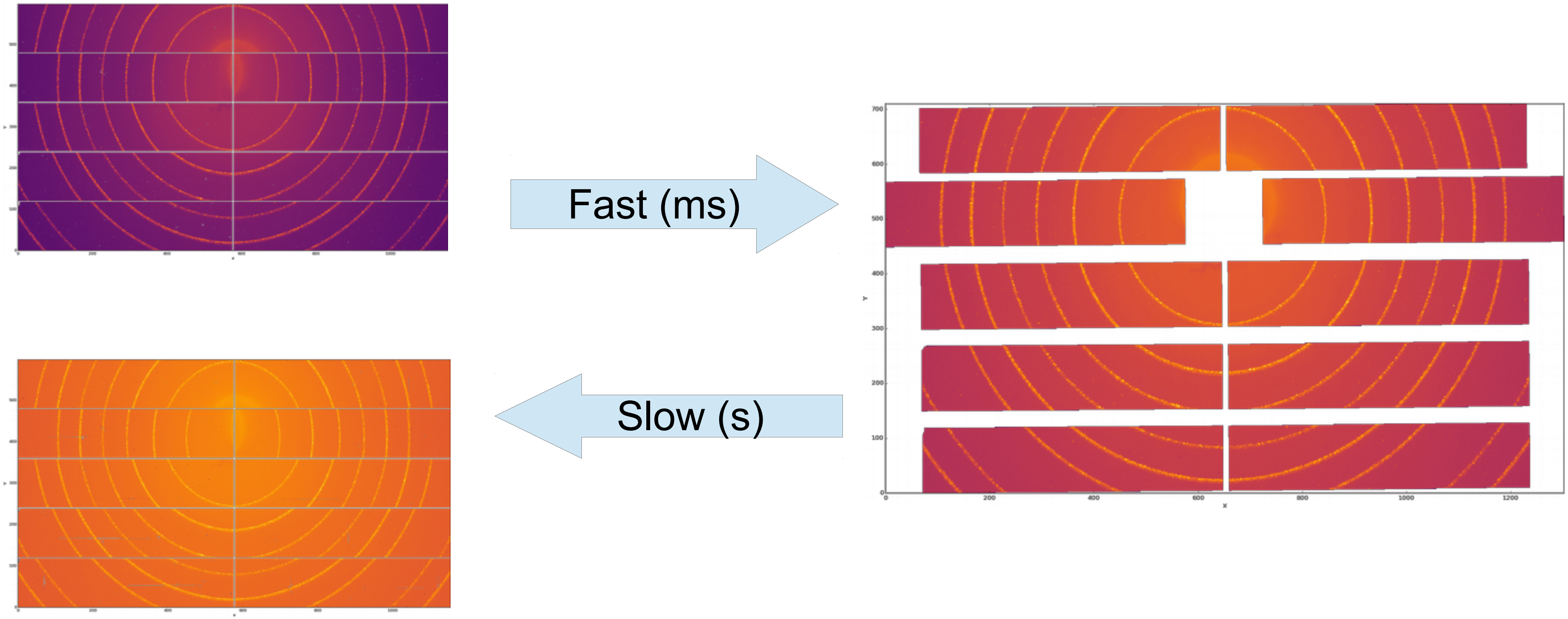
- **Important method:**
  - correct(img): re-distribute intensity on a regular grid.
  - uncorrect(img): reverse a correction, for masks in Fit2D

- **Nota:**

  Because of the great regularity of this rebinning, LUT is faster than CSR

http://pyfai.readthedocs.io/en/latest/api/pyFAI.html#pyFAI.distortion.Distortion

Distortion correction, just an example



WOS detector, courtesy of D2AM CRG beamline

Worker

Set of classes to perform azimuthal integration, distortion correction or normalization, repetitively on a set of files.

- **Usage:**

  w = pyFAI.worker.Worker(ai)

  w = pyFAI.worker.DistortionWorker(detector)

  w = pyFAI.worker.PixelwiseWorker(dark, flat, mask)

- **Important method:**
  - w.process(img)

http://pyfai.readthedocs.io/en/latest/api/pyFAI.html#pyFAI.worker

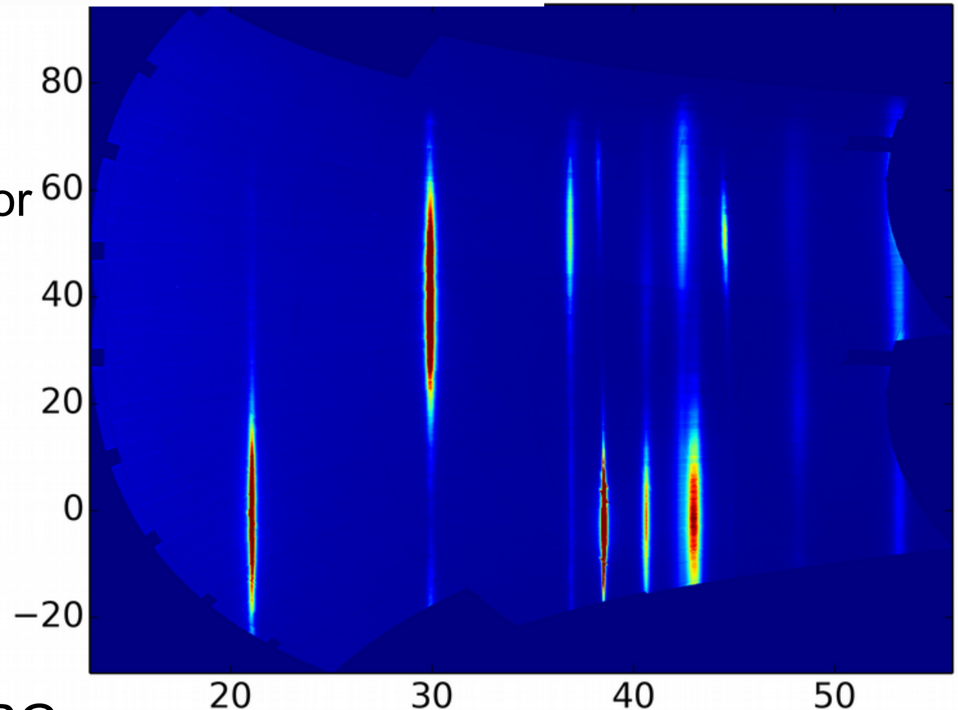
Multi-geometry integrator

- **Assemble multiple images taken at various position into a single pattern**

Documented on: http://pyfai.readthedocs.org/en/latest/usage/tutorial/multi-geometry.html

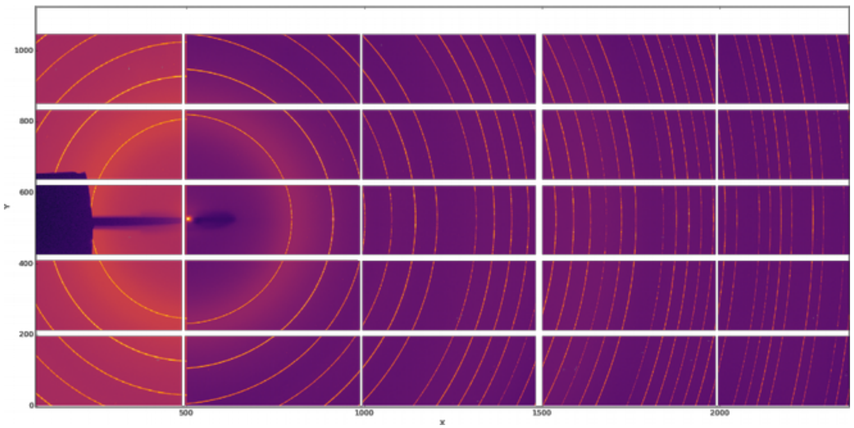
```python
import glob
import fabio
from pyFAI.multi_geometry import MultiGeometry

img_files = glob.glob("*.cbf")
img_data = [fabio.open(i).data for i in img_files]
ais = [i[:-4]+".poni" for i in img_files]
mg = MultiGeometry(ais, unit="q_A^-1", radial_range=(0, 50), wavelength=1e-10)
q, I = mg.integrate1d(img_data, 10000)
```

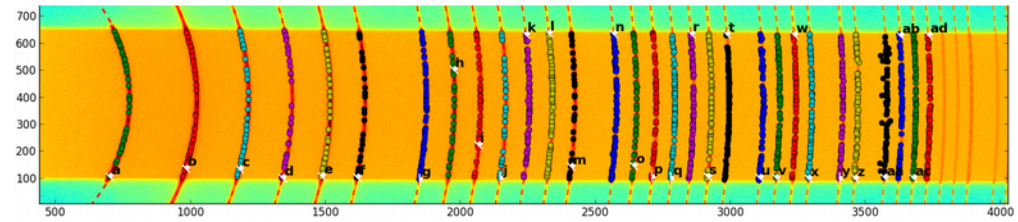
Takes care of solid-angle normalization between detector



Courtesy of D2AM CRG

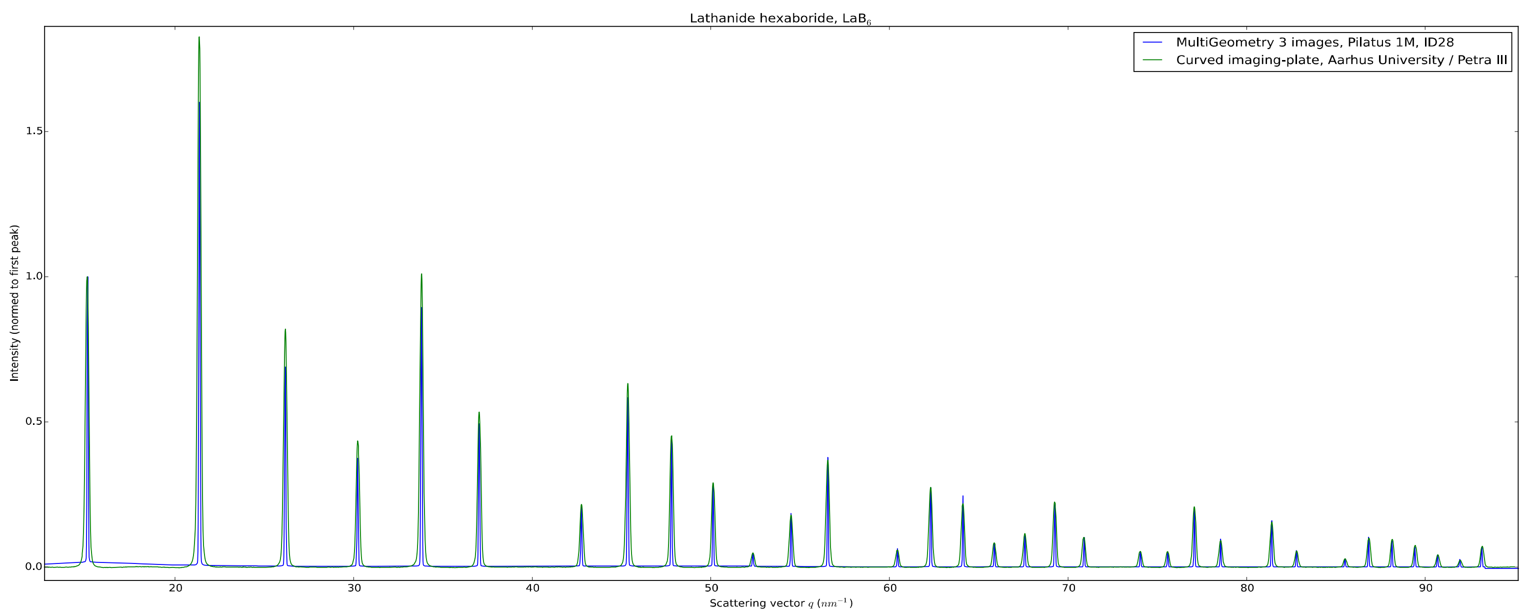
Multi-geometry vs larger detector





↑1 image taken with a curved imaging plate (detector built at Aarhus/Denmark)

↑ 3 images taken with a Pilatus_1M on a rotating arm (ID28) offset by 0°/17°/45°



Past & Future:

What are the projects ?

PyFAI: Past ...

- **Looking back:**
  - 2011: Basic idea: geometry, refinement, histograms
  - 2012: Dimitris Karkoulis: histogramming in OpenCL,
    Pixel splitting
  - 2013: Zubair Nawaz: spline calculation in OpenCL,
    Look-up table
  - 2014: Aurore Deschildre: blob pixel detection
    Giannis Ashiotis: CSR sparse matrix multiplication
  - 2015: Frederic Sulzman pixel-detector description
  - 2016 - 2019: Valentin Valls: graphical interface for pyFAI

PyFAI: Present & Future ...

- **Recently done:**
  - OpenCL port of “separate”, request from by ID13
  - Detector distortion, correction, NeXus representation (ID15, ID02, BM02)
  - Diffraction imaging (collaboration with Soleil & CRGs)
  - Multi-detector integrators
  - log(q) or other user defined output spaces (ID02)
- **On the radar**
  - CLI interface
    - **Single application → ease distribution for windows & MacOSX**
    - **watershed segmentation (not yet production ready)**
    - **image reconstruction of gaps**
  - Graphical interface for calibration
  - Clean-up/merge LUT and CSR cython code base
  - Variance propagation with pixel splitting
- **You have ideas ? We are open to collaboration !**

Thank you for your attention

PyFAI is fast