ESRF | The European Synchrotron

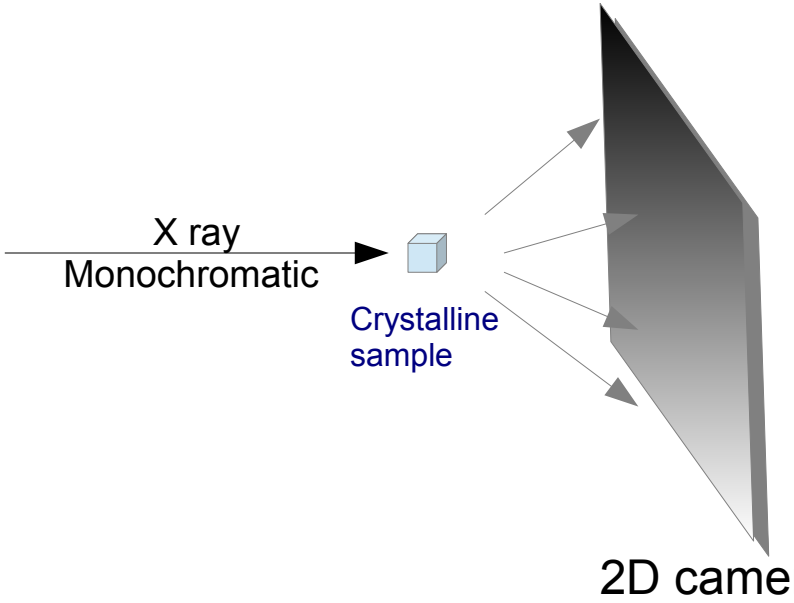
PyFAI: Data reduction tools for scattering experiments

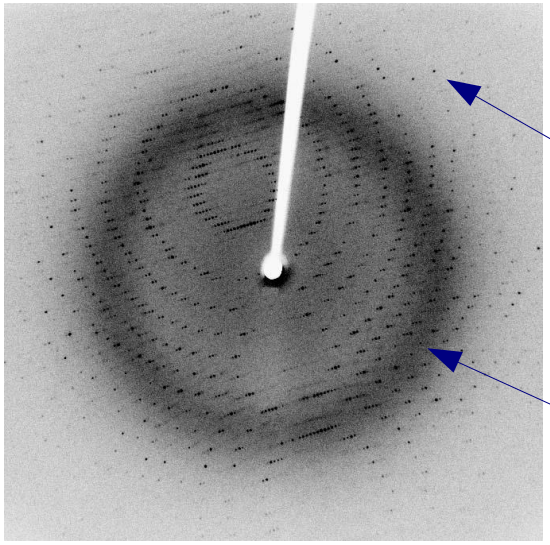
Jérôme Kieffer
Data analysis, ESRF

Layout

- **Power diffraction and scattering of X-Rays**
- **What is azimuthal integration**
- **The need for faster data processing …**
- **… without compromising quality**
- **PyFAI:**
  - Ecosystem and user community
  - Understanding few concepts
    - **Integration**
    - **Calibration**
  - Latest developments: 3D view of the experimental setup
- **Collaboration around silx-kit**

X-ray scattering experiments





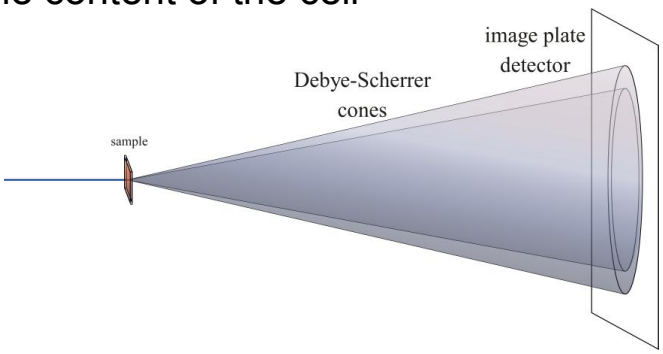
Source: Wikipedia
CC-BY-SA: Jeff Dahl

Bragg spots: diffraction from single crystal

Ice ring: diffraction from powder

- **Light is reflected as on mirror:**
  - No energy change
  - Direction of diffracted beam depend on the crystalline cell and its orientation
  - Intensity of the diffracted beam depend on the the content of the cell

    → Nobel price of Bragg (1915) $n\lambda = 2d\sin\theta\,,$
- **Multiple small crystals (powder)**
  - Isotropic cones giving conics (mainly ellipses) when intersected with the detector



Powder diffraction and small angle scattering

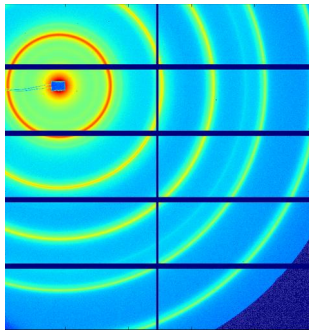
Application of powder diffraction:

- Phase identification (mapping)
- Crystallinity
- Lattice parameters
- Thermal expansion
- Phase transition
- Crystal structure
- Strain and crystallite size

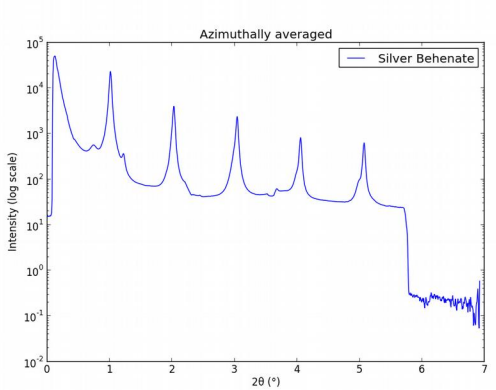
Application of small angle scattering

- Micro/nano-scale structure
- Particle shape
- Protein domains
- Protein folding
- Colloids

- **Both rely on the same transformation: 2D image → azimuthal average**



Azimuthal integration



Many different tools exists … beside pyFAI

- **FIT2D**
  - Freeware, reference code written in Fortran by A. Hammersley, ESRF
- **XRDUA**
  - Focuses of diffraction mapping, based on IDL, GPL, by W de Nolf
- **Dawn**
  - Data analysis framework developed @ DLS in Java.
- **DataSqueeze**
  - Freeware written by Paul Heiney @ University of Pennsylvania
- **Foxtrot**
  - Developed in Java @ synchrotron Soleil with Xenocs
- **Maud**
  - Freeware, developed in Java @ University Trento
- **GSAS-II**
  - Freeware tool written in Python @ APS … License issue
- **Scikit-beam**
  - Very similar to pyFAI in licenses and philosophy. Developed @NSLS-II

Python implementation of azimuthal integration

Common initialization step:

In [1]:
```
import numpy
npt = 1024
y,x  = numpy.ogrid[-512:512,-512:512]
radius = (x*x+y*y)**0.5
rmax = radius.max()+0.1
data = numpy.random.random((1024,1024))
```

Naive approach integration using corona extraction with masks:

In [2]:
```
%%time
res_loop = numpy.zeros(npt)
for i in range(npt):
    rinf = rmax * i / npt
    rsup = rinf + rmax / npt
    mask = numpy.logical_and((rinf <= radius),(radius < rsup))
    res_loop[i] = data[mask].mean()
```

```
CPU times: user 1.04 s, sys: 0 ns, total: 1.04 s
Wall time: 1.04 s
```

Vectorized version using histograms:

In [3]:
```
%%time
count_of_pixels = numpy.histogram(radius, npt, range=[0,rmax] )[0]
sum_of_intensities = numpy.histogram(radius, npt, weights=data, range=[0,rmax])[0]
res_vec = sum_of_intensities / count_of_pixels
```

```
CPU times: user 19.5 ms, sys: 1.44 ms, total: 20.9 ms
Wall time: 19.4 ms
```

In [4]:
```
# Speed-up: 50x, validation:
numpy.allclose(res_loop, res_vec)
```

Out[4]: True

Need for speed ...

- **New EBS source … 50x brighter**
  - Upgrade currently ongoing
- **New generation of detectors …which are always faster**
  - Eiger detector (3000Hz)
  - Jungfrau detector (2000Hz)
- **The gap between what is available and what scientists expect grows:**
  - Most other codes use the same algorithm … and reach the same speed
    - **Fit2D written in Fortran**
    - **SPD written in C**
    - **Foxtrot written in Java**
  - Change the algorithm, not the programming language !
    - **Histograms cannot be parallelized ! (easily)**
    - **CSR dot product is many-core friendly**

Look-up table integration using only *numpy & scipy.sparse*

Using a Sparse matrix multiplication

Those multiplication can take advantage of parallel hardware unlike histogram which require costly *atomic* operations. This trick is called "scatter to gather" transformation in parallel programming.

In [5]:
```
%%time
from scipy.sparse import csc_matrix
positions = numpy.histogram(radius, npt, range=[0,rmax] )[1]
row = numpy.digitize(radius.ravel(), positions) - 1
size = row.size
col = numpy.arange(size)
dat = numpy.ones(size, dtype=float)
csr = csc_matrix((dat, (row, col)), shape = (npt, data.size))
print(csr.shape)
```

```
(1024, 1048576)
CPU times: user 60.5 ms, sys: 6.21 ms, total: 66.7 ms
Wall time: 69.7 ms
```

In [6]:
```
%%time
count_csr = csr.dot(numpy.ones(data.size))
sum_csr = csr.dot(data.ravel())
res_csr = sum_csr / count_csr
```

```
CPU times: user 3.11 ms, sys: 3.1 ms, total: 6.21 ms
Wall time: 4.69 ms
```

In [7]:
```
# Speed-up: 5x vs histograms, validation:
numpy.allclose(res_csr, res_vec)
```

Out[7]: True

Sources of this demo available on:
https://gist.github.com/kif/ab37c61351d8238f90245b0afb56192e

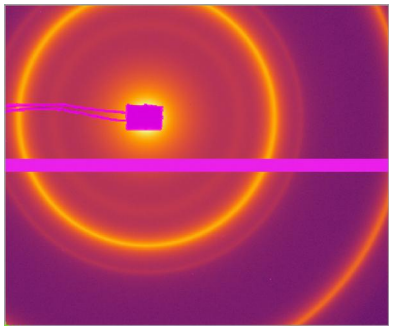
Advantages of histograms vs sparse matrix multiplication

	Histograms	Sparse matrix multiplication
Pro	• **Easier to understand** • **Low memory consumption** • **Fast initialization**	• **Faster, even on one core** • **Many-core friendly** – OpenMP and OpenCL
Con	• **Pretty slow** • **Hardly parallelizable**	• **Slower initialization** • **The sparse matrix can be large**
Rule of thumb:	less than 5 images	more than 5 images

High frequency noise issue

In 2D patterns

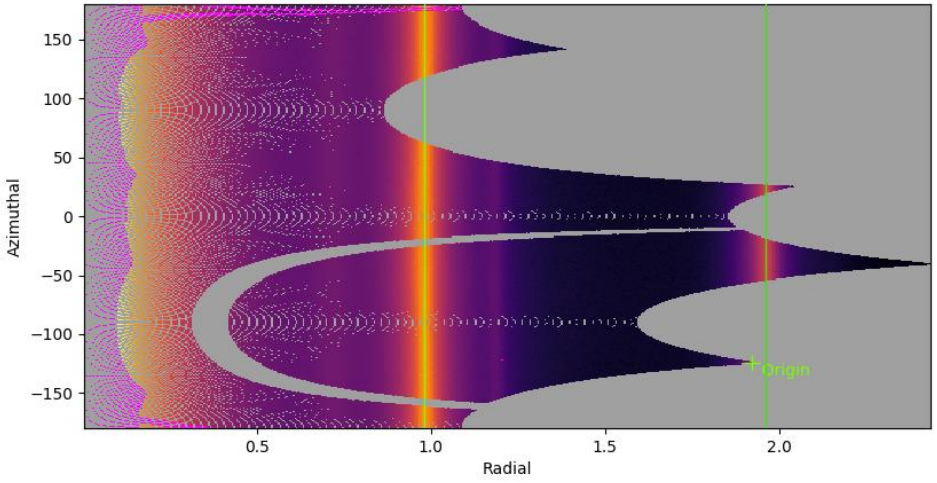
Example with SAXS data integrated in 2D



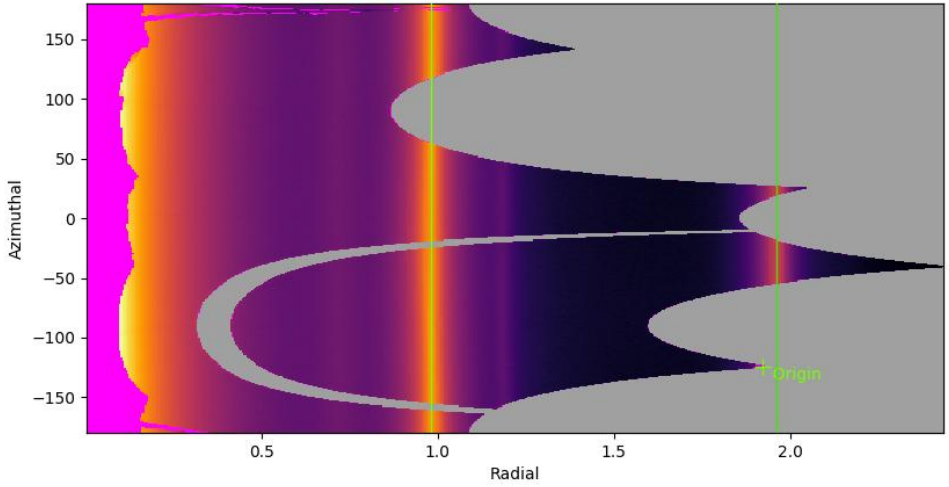
Pilatus 200k:
~500x400 pixels

2D averaging over 512x360 bins

Without pixel splitting

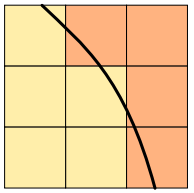


With pixel splitting

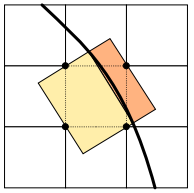


Pixel splitting schemes available in pyFAI

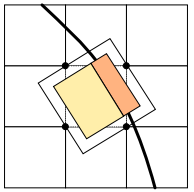
- **No pixel splitting: default histograms**
  - Each pixel contributes to a single bin of the result
  - No bin correlation but more noisy
  - The pixel has no surface: sharpest peaks



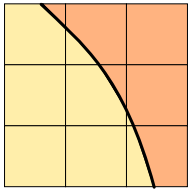
- **Bounding-box pixel splitting**
  - The smoothest integrated curve
  - Blurs a bit the signal



- **Pseudo pixel splitting**
  - Scale down the bounding box to the pixel area, before splitting.
  - Good cost/precision compromise, similar to FIT2D



- **Full pixel splitting**
  - Split each pixel as a polygon on the output bins.
  - Costly high-precision choice



Impact of pixel splitting on integration speed

- **Histogram based algorithms:**
  - Each pixel is split over the bins it covers.
  - The corner coordinates have to be calculated (4x slower initialization)
  - The slow down is function of the oversampling factor, for every image

- **Sparse matrix multiplication based algorithms**
  - The sparse matrix contains already the pixel splitting scheme
  - Longer initialization time related to the oversampling factor
  - There are *NO* performance penalty on the integration itself

All those consideration are independent of the programming language

Nevertheless, Python which is interpreted is expected to be 1000x slower than:
- compiled code like C, C++, Fortran, ...
- JIT compiled code like Java, Julia or numba

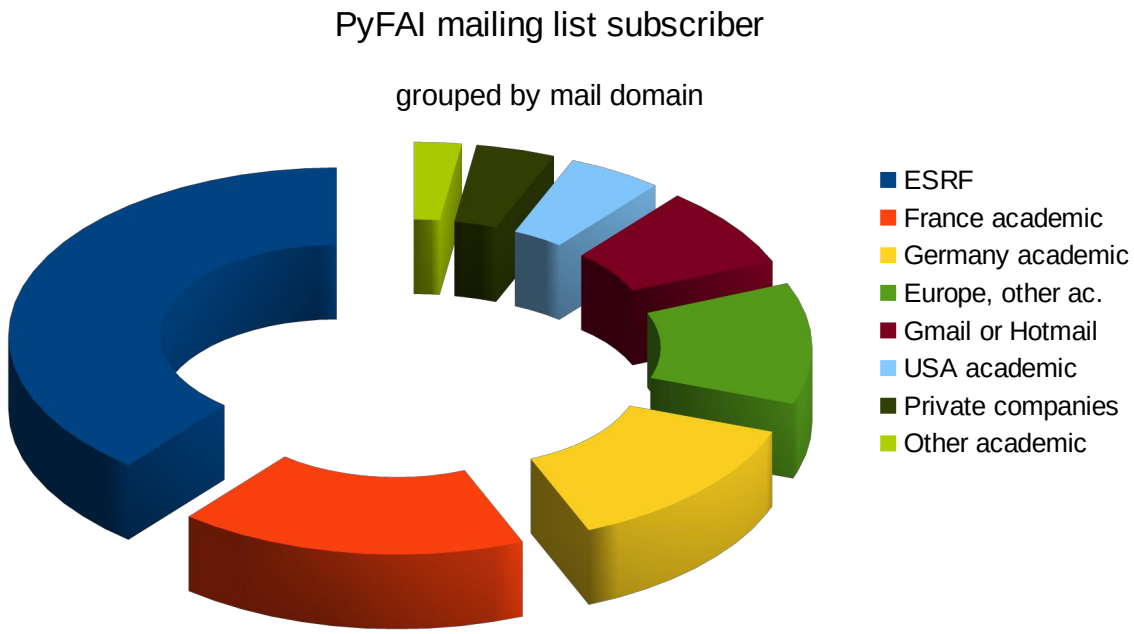
So why using pyFAI ?

PyFAI is yet another azimuthal integration tool

- **Written in Python (compatible with ~~2.7, 3.4,~~ 3.5, 3.6 & 3.7)**
  - Free, fast and portable
  - MIT licensed: fully compatible with the SciPy stack and with business
  - Part of the *silx* collaboration on data analysis initiated by ESRF
  - Graphical user interface using Qt5
- **Open to collaboration**
  - About 20 contributors,
    - **half from ESRF**
    - **few contributions from industry**
- **Avoid compromises**
  - No difficulty is hidden:
    - **science does not suffer approximations**

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**
  - Archived on http://www.silx.org/lurker/list/pyfai.en.html
  - 132 people subscribed to the list (Jan 2019; 112 in 2018)
  - limited activity (~1 thread/month)

http://www.silx.org/doc/pyFAI/dev/project.html#getting-help

Reasons to chose pyFAI

- **Faster than other**
  - First tool using sparse matrix multiplication to perform integration
  - All computation intensive kernel are ported to C, C++ or OpenCL
  - PyFAI is the only azimuthal integration tool benefiting from GPU
- **More versatile (hackable) than other**
  - Many integration space already exists ...
    - **you can add your own easily**
  - It's geometry is so generic it matches all configuration
    - **SAXS, WAXS …**
  - Most detectors are already defined
    - **Each detector can be adapted, and saved in a Nexus file**
  - It has a nice user interface thanks to Valentin !
- **Part of the *silx* collaboration**
  - Bus-count slightly larger than one !

Layers in pyFAI

- **Applications level:**
  - GUI applications: pyFAI-calib2, 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 with OpenMP or OpenCL)

Ideally used from jupyter

Flexibility

Ease of use

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

  It is up to you !

Examples of other application relying on pyFAI

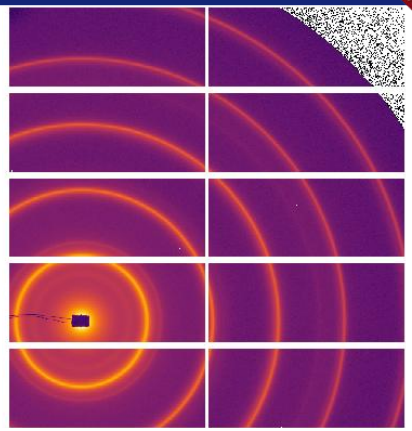
- **NanoPeakCell: Serial crystallography pre-processing**
  - Nicolas Coquelle, IBS Grenoble
- **PySaxs: data analysis for SAXS experimental station**
  - Olivier Tache, CEA Saclay
- **Dpdak: online data analysis for Saxs data**
  - Gunthard Benecke, Petra III
- **Dioptas: offline data analysis for high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

http://www.silx.org/doc/dev/pyFAI/ecosystem.html

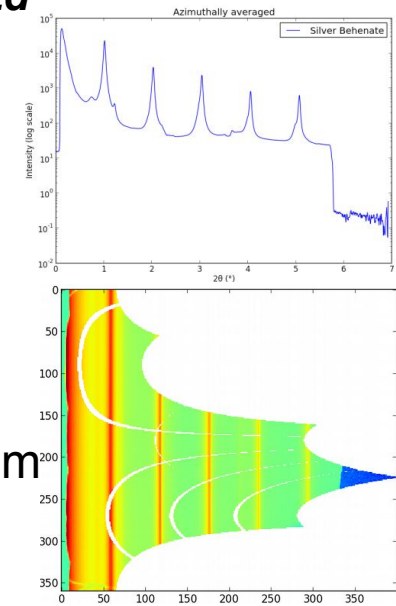
Concepts in PyFAI

- **Image**

  2D array of pixels as *numpy* array
  read using *silx, fabio, h5py,* ...



- **Azimuthal integrator: core object**
  - **powder diagram using *integrate1d***
  - **“cake” image, azimuthally regrouped using *integrate2d***



- **Detector**

  - **Calculates the pixel position (center and corners)**
  - **Calculate or store the mask**

    → **saved as a HDF5 file**

- **Geometry**

  Position of the detector from the sample & incoming beam

  → **saved as *PONI*-file**

http://www.silx.org/doc/pyFAI/dev/geometry.html#detector-position

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 and rotate around the sample**

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) **Histogram bin position with associated intensities**

6) **Histogram bin position with solid angle (and other normalization)**

7) **Return bin position and the ratio of sum of intensities / sum of Ω**

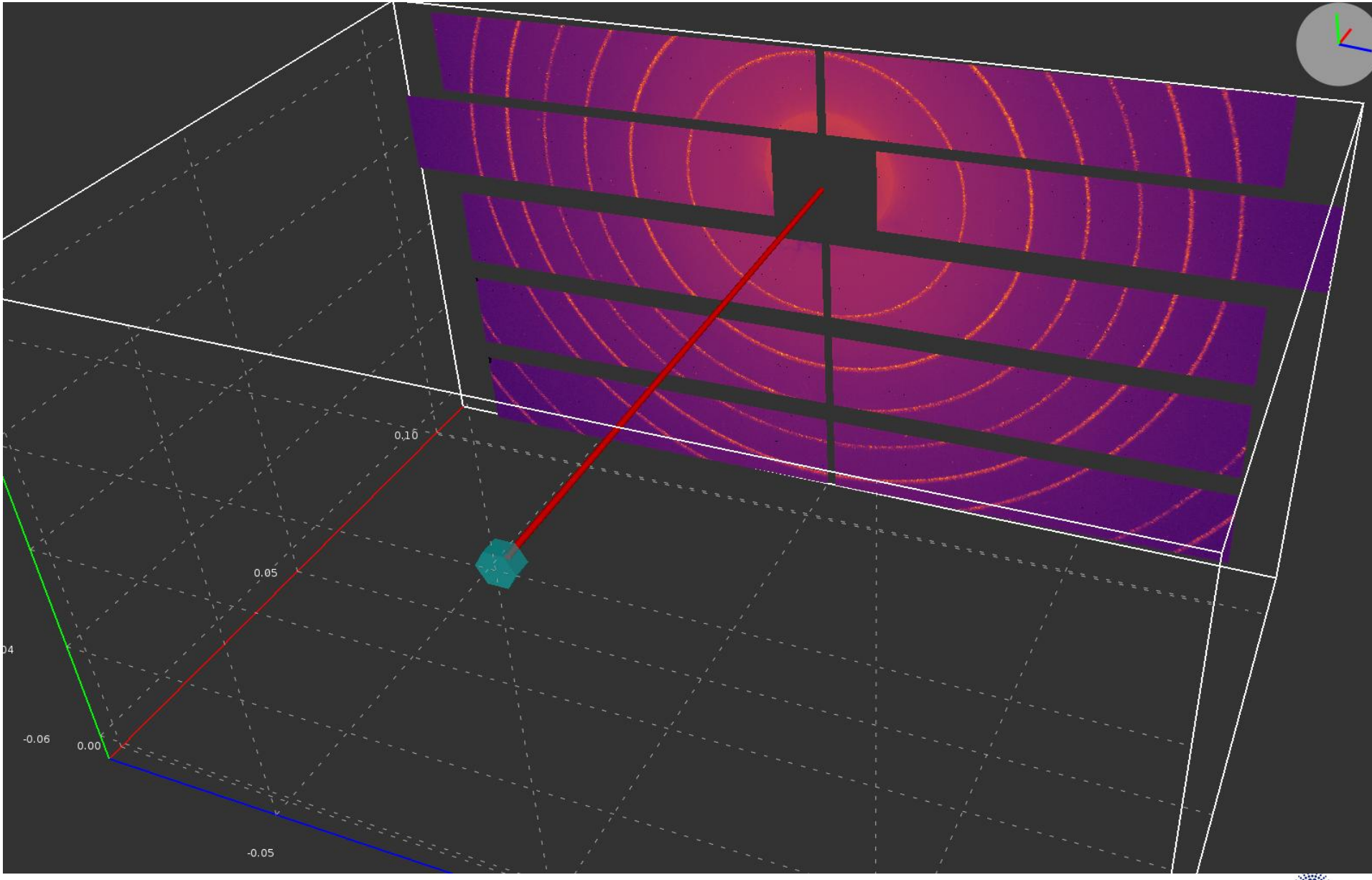
Calibration in pyFAI

- **The determination of the geometry is also known as calibration**
  - The prerequisite is:
    - **detector geometry and mask,**
    - **calibrant (LaB_6, CeO_2, AgBh, …)**
    - **wavelength or energy used**
  - Only the position of the detector and the rotation needs to be refined:
    - **3 translations: dist, $poni_1$ and $poni_2$**
    - **3 rotations: rot_1, rot_2, rot_3**
- **It is divided into 4 major steps:**
  1) Extraction of groups of peaks
  2) Identification of peaks and groups of peaks belonging to same ring
  3) Least-squares refinement of the geometry parameters on peak position
  4) Validation by an human being of the geometry
- **PyFAI assumes this setup does not change during the experiment**
- **Tutorial:**

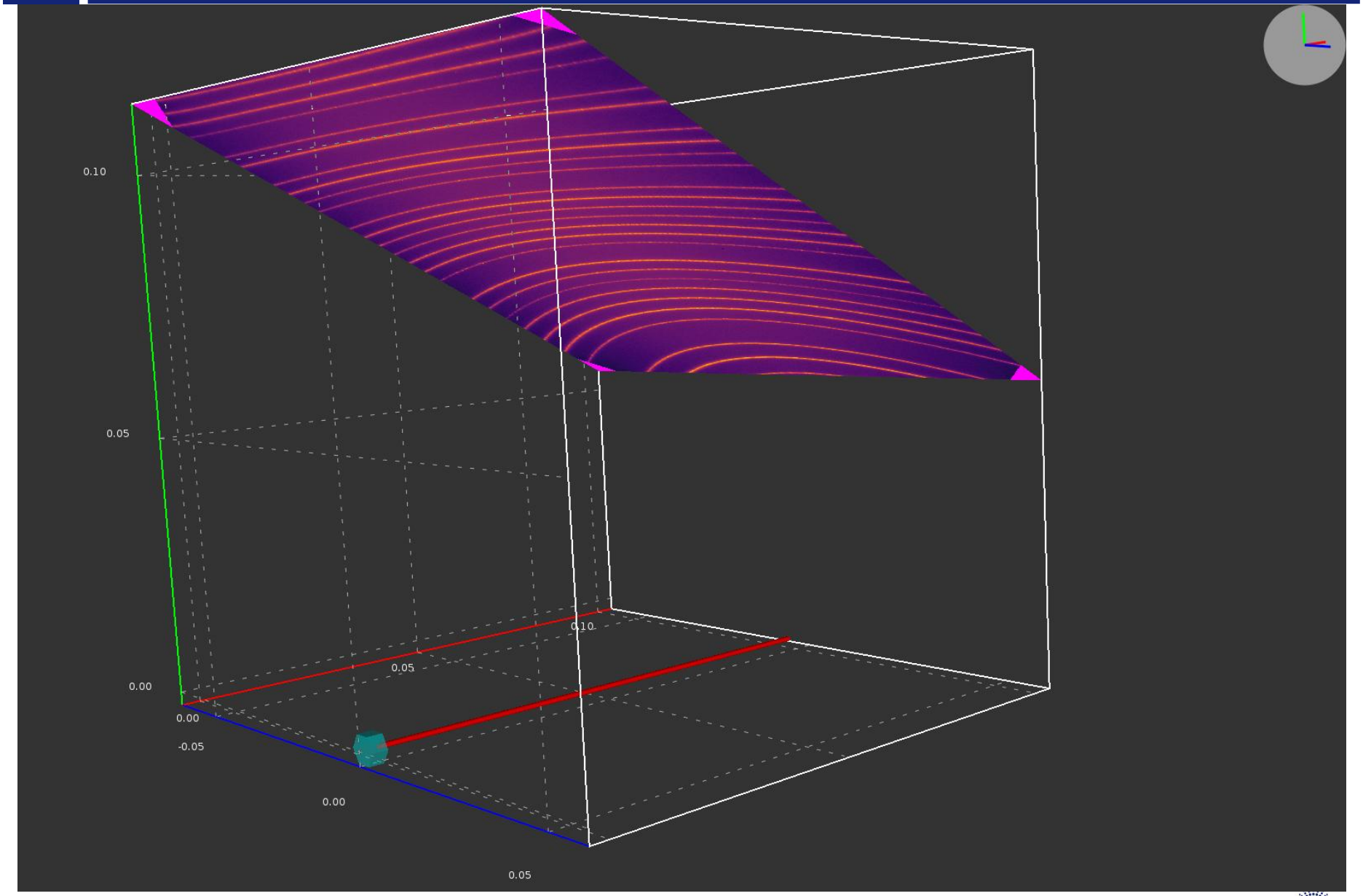
  http://www.silx.org/doc/pyFAI/dev/usage/cookbook/calib-gui/index.html

Latest *cool* feature:

3D view of the diffraction setup



3D view of the diffraction setup



Collaboration around *silx*

Shared development around:

- User interface (Valentin Valls)
- GPU computing (Pierre Paleo)
- Scientific data analysis

silx-kit: join efforts, share the maintenance

Fork me on GitHub

silx

Scientific Library for eXperimentalists

Resources

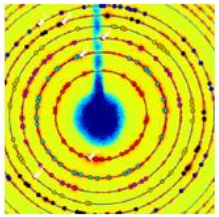
- silx on GitHub
- Wheels and source on PyPi
- Installation instructions

Documentation

- Latest release
- Nightly build
- v0.3.0
- v0.2.0
- v0.1.0

pyFAI

Fast Azimuthal Integration in Python



Resources

- pyFAI on GitHub
- Wheels and source on PyPi
- Installation instructions

Documentation

- Latest release
- Nightly build

FabIO

I/O library for images produced by 2D X-ray detector

Resources

- FabIO on GitHub
- Wheels and source on PyPi
- Installation instructions

Documentation

- Latest release
- Nightly build

Management of the *silx-kit* project

- **Public project hosted at github**

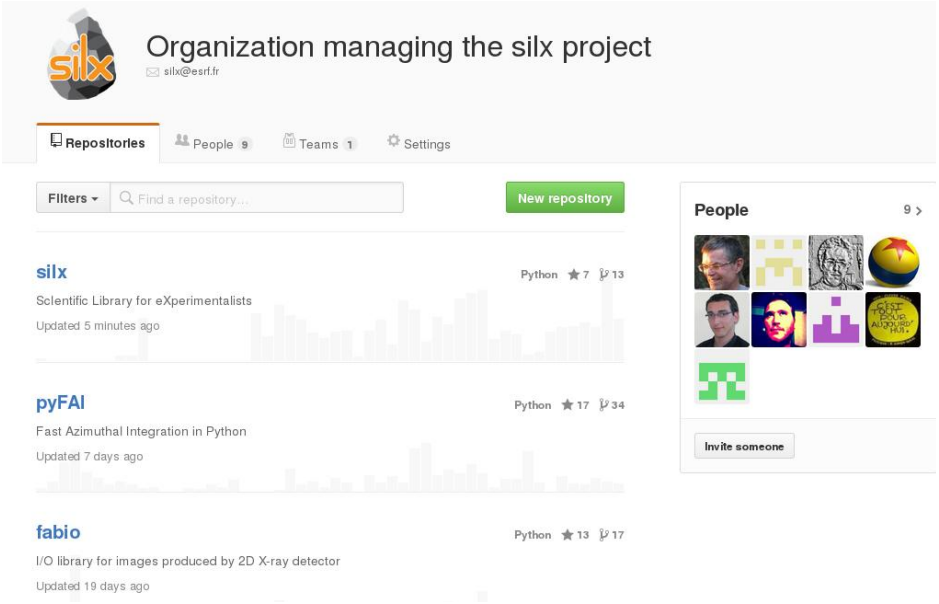
  https://github.com/silx-kit/silx

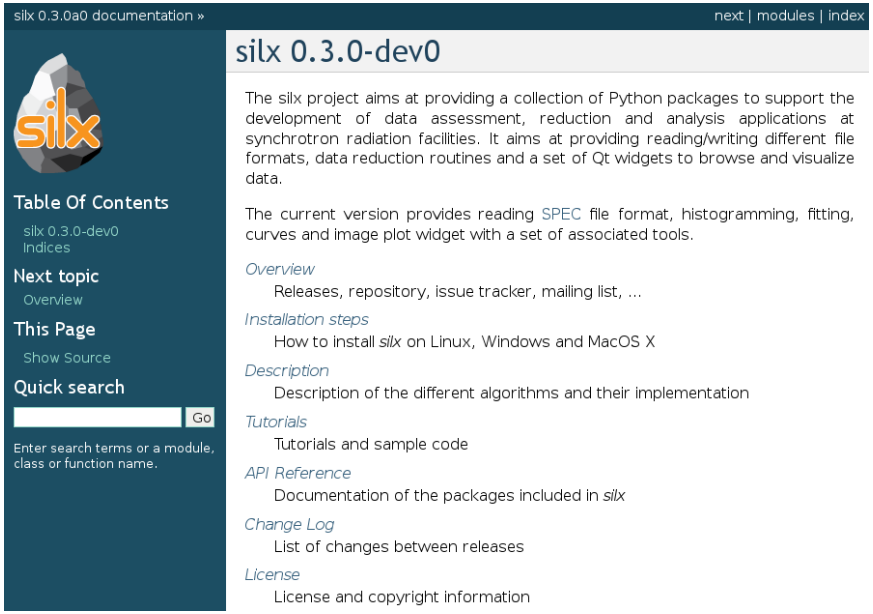
- **Continuous testing**

  Linux, Windows and macOS

- **Nightly builds**
  - Debian packages
- **Weekly meetings**
- **Quarterly releases**
- **Code camps before release**
- **Continuous documentation**

  http://www.silx.org/doc/silx/





silx-kit project and the *silx* library

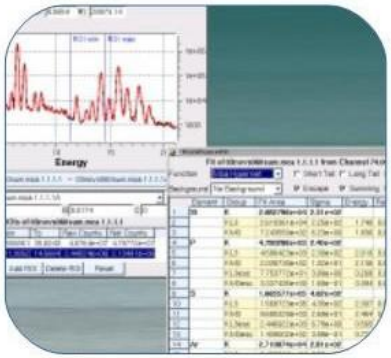
Mainly Pierre Knobel …

Mainly Jérôme Kieffer

Mainly Thomas Vincent

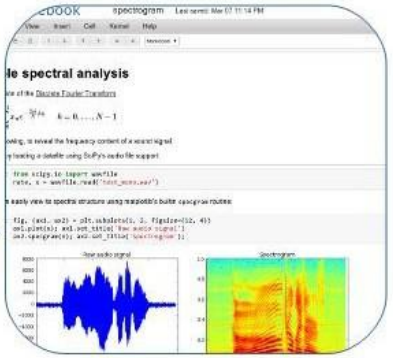
Online data analysis

e.g. PyFAI



Standard Apps e.g.
PyMCA, PyDIF

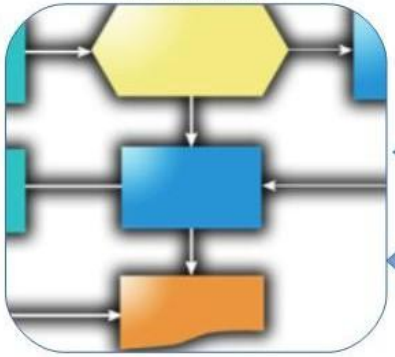
… and Valentin Valls



Ipython Notebook

General Purpose
Core Toolkit

Workflows



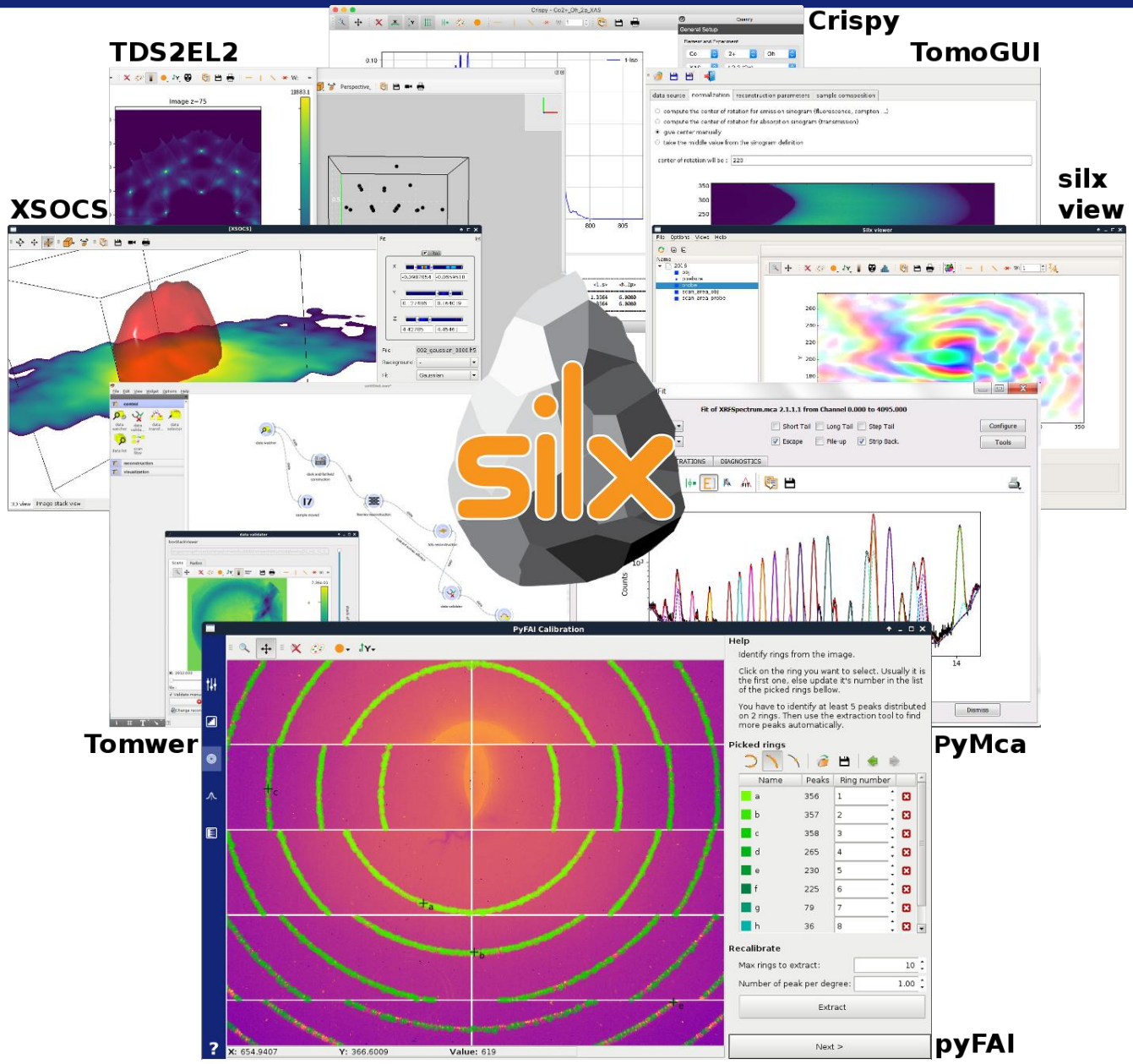
Extensions

Extension library
e.g. PyHST, ...

External
Libraries +
Apps

Mainly Henri Payno

Outcome of the *silx* toolkit after 3 years:



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

    Silx collaboration (GUI + OpenCL)

PyFAI: Present & Future ...

- **Recently done:**
  - Detector distortion, correction, NeXus representation (ID15, BM02)
  - Multi-detector integrators & goniometer calibration (BM02, BM20, Soleil)
  - log(q) & other user defined output spaces (ID02)
  - GUI for calibration and batch processing (ID15, ID16, ID21, ID22)
- **On the radar**
  - Corrected variance propagation (EMBL Hamburg)
  - Integration into JupyterLab: Panosc & Calypso+ EU projects
  - Version 1.0
- **You have ideas ? We are open for collaboration !**

Acknowledgments

- **Software group & DAU:**
  - Fabienne Mengoni
  - Andy Götz
  - Claudio Ferrero †
- **ESRF Beamlines:**

  BM01, BM02, ID02, ID11, ID13, ID15, ID21, ID22, ID23, BM26, BM29, ID29, ID30, ID31 ...
- **Trainees:**
  - Aurore Deschildre
  - Frederic Sulzmann
  - Guillaume Bonamis
- **Other synchrotron/labs**
  - Soleil: Fred Picca, Diffabs & Cristal beamlines
  - APS: Clemens Prescher
  - NSLS-II: skbeam team
- **LinkSCEEM-2 grant**
  - Dimitris Karkoulis
  - Giannis Ashiotis
  - Zubair Nawaz
- ***Silx* project:**

Valentin Valls, Pierre Knobel

Henri Payno, Pierre Paleo

Thomas Vincent, V. Armando Solé

Questions ?