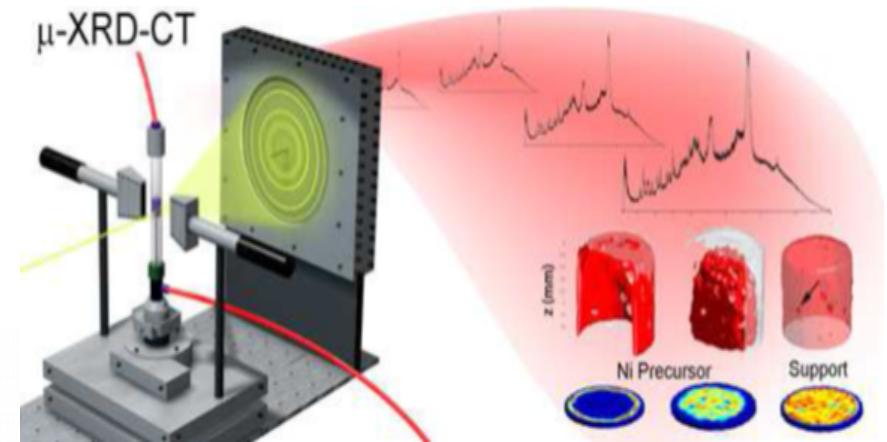




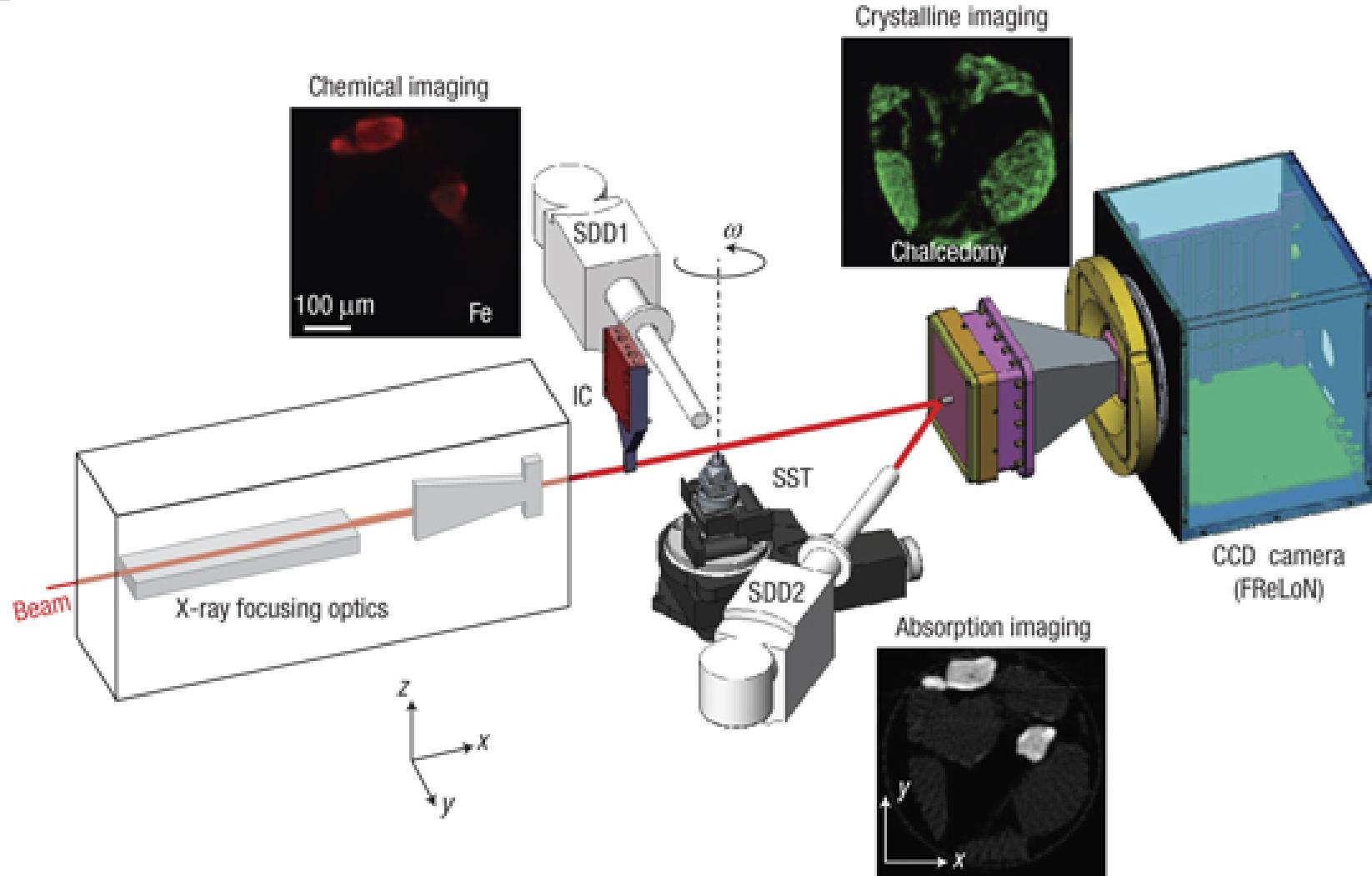
| The European Synchrotron



Imaging with diffraction data ...
on the high-energy beamline for materials engineering

- **Introduction: μ XRD-CT and PDF-CT**
- **Presentation of the ID15 @ESRF**
- **Acquisition scheme**
- **Hardware**
- **Software**
- **Conclusions**
- **Acknowlegments**

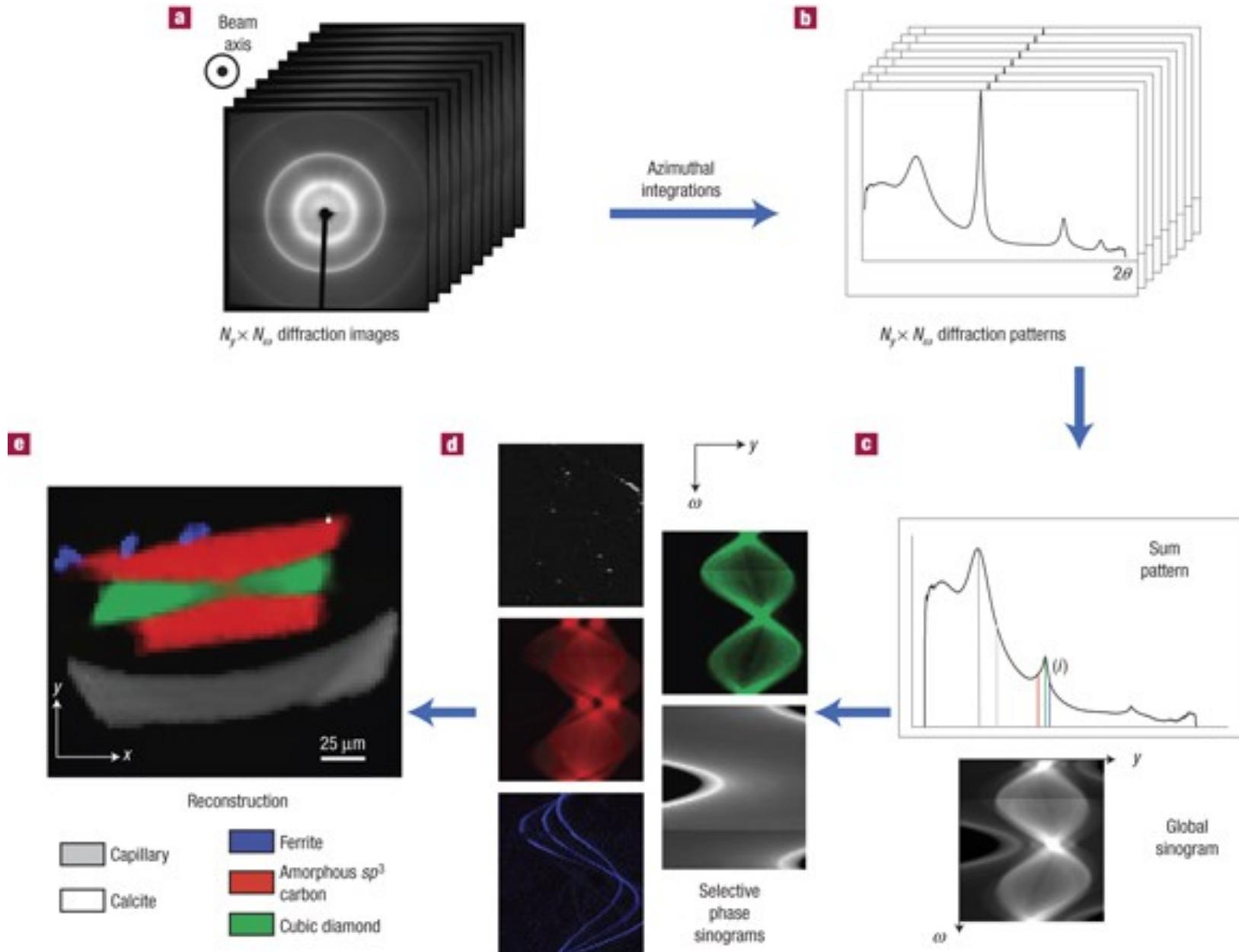
Micro X-Ray diffraction contract tomography



Probing the structure of heterogeneous diluted materials by diffraction tomography

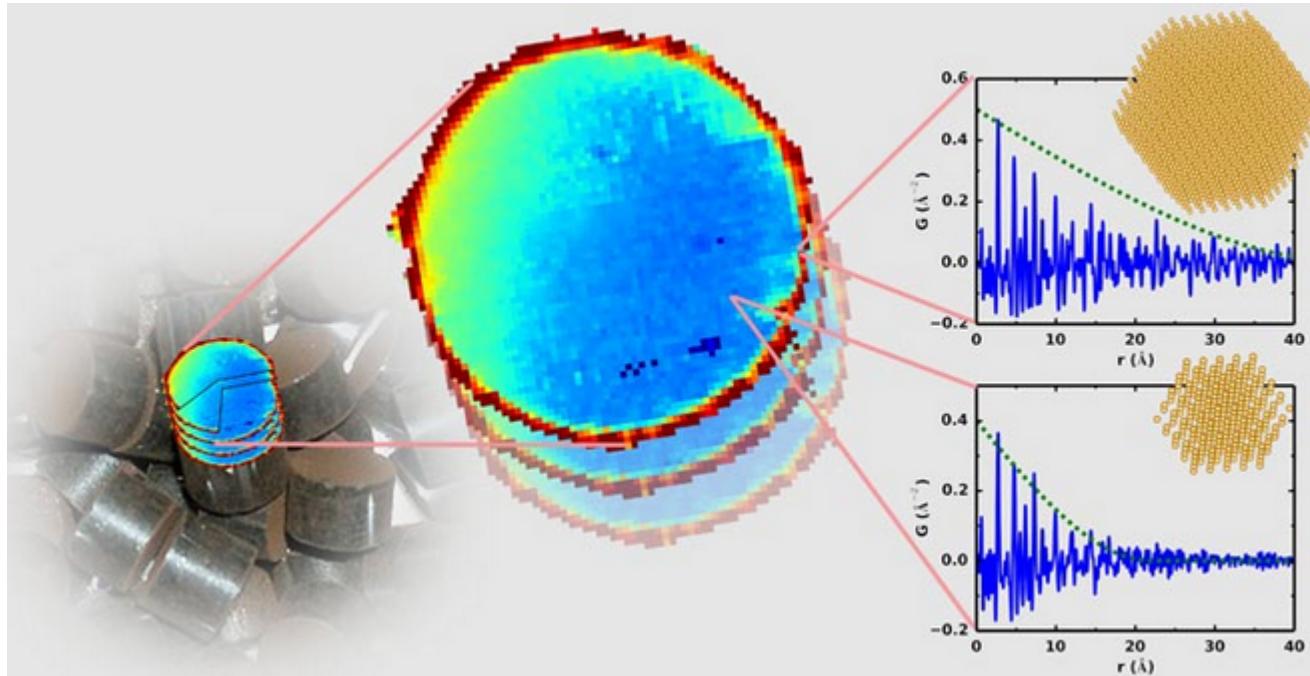
Pierre Bleuet, Eléonore Welcomme, Eric Dooryhée, Jean Susini, Jean-Louis Hodeau & Philippe Walter
Nature Materials 7, 468 - 472 (2008) doi:10.1038/nmat2168

diffraction contrast tomography: data processing



Pairwise Distribution Function – Contrast Tomography

- **Extention of the PDF tools from S. Billinge to amorphous phases**
- **Combined to μ XRD-CT**



Pair distribution function computed tomography

Simon D. M. Jacques, Marco Di Michiel, Simon A. J. Kimber, Xiaohao Yang, Robert J. Cernik, Andrew M. Beale & Simon J. L. Billinge

Nature Communications 4, Article number: 2536 (2013) doi:10.1038/ncomms3536

New ID15a beam-line

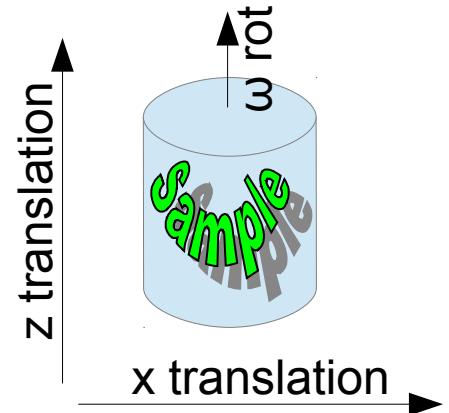
- **High energy materials beamline: 20 - 750 keV**
- **Beam size ~ 1µm**
- **Applications**
 - Solid state chemistry
 - Catalysis
 - In-situ experiments
 - Metallurgy
- **Techniques**
 - energy dispersive diffraction
 - powder diffraction
 - pair-distribution function analysis
 - diffraction contrast tomography

Diffraction contrast tomography

- **Strategy:**

- $6 \rightarrow 600 \omega\text{-steps}$
- $100 \rightarrow 1000 x\text{-steps}$
- $100 \rightarrow 1000 z\text{-steps}$
- Nyquist suggests:
 $\omega\text{-steps} = x\text{-steps} * \pi / 2$

The fastest scan can be ω or x depending on the experiment
the number of z steps is adapted depending on available time



- **The target for data processing is 1000^3 datasets**

Each image is multi-mega-pixel, ~ 10 Mbyte/image

- **The raw dataset represents: 10 Pb (10 000 Tera Bytes !)**

- ESRF has currently only 4PB of storage

Hardware used: Dectris Pilatus 2M CdTe

- **Well known electronics:**
 - 2.4 Mpix images
 - 250 Hz acquisition speed
 - >55% efficiency up to 100 keV
- **Used on many protein diffraction beamlines**
- **Well interfaced in LimA**
 - But never used continuously at full speed ...
... until now
- **Allows large μ XRD-CT scans within a week:**
 - $N_z = 400$, $N_x = 400$, $N_\omega = 600$
 - 100 million files
 - 230 Tera-Bytes of compressed data
 - 800 Giga-Bytes after azimuthal integration :-)



Hardware used: interconnect

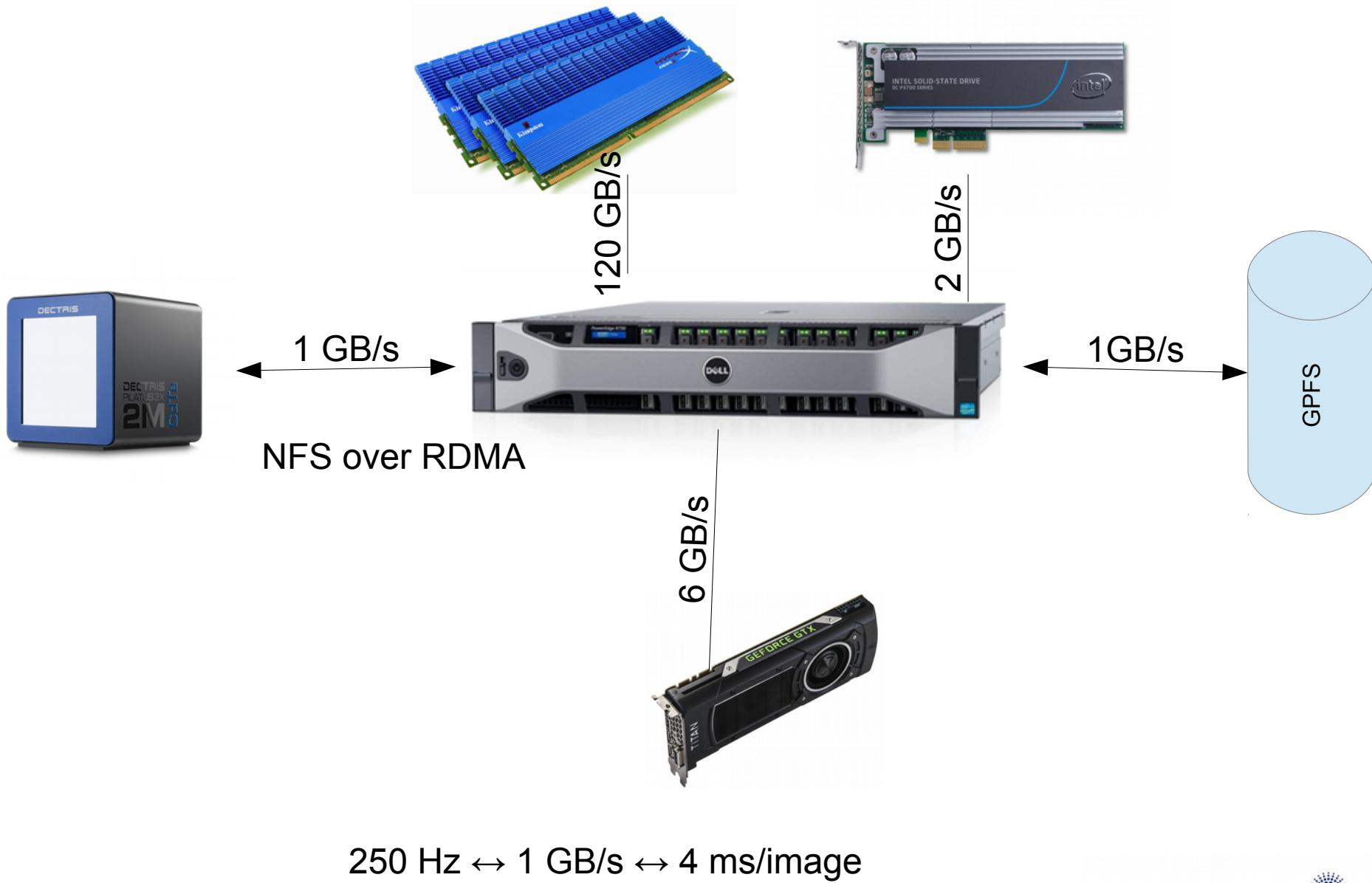
- **Pilatus3 2M detector PC interconnect:**
 - ONE 10GB fiber link out
 - Maximum transfer rate: 1 Gb/s
 - NFS over RDMA is needed
- **Image size and bandwidth needed at 250Hz**
 - 10 Mbyte raw data → **2.5 Gb/s**
 - 2.4 Mbyte CBF data → 0.96 Gb/s
 - LZ4 compression gives variable image size → discarded

CBF compression scheme (4x) is used to start with ...

Data analysis computer

- **Standard dual socket server (Dell R730)**
 - 2 sockets, 3.4 GHz hexacore processors
 - 128GB of memory
- **Network interconnect**
 - 2x 10Gbit ethernet link
 - **To the detector (NFS over RDMA server)**
 - **To the central storage (GPFS client)**
 - 4x 1Gbit link
 - **Commands & normal NFS**
- **Local storage:**
 - Fast SSD interfaced in PCI-e (NVMe), 2TB
 - Disks, 2TB extensible to 24 TB
- **Local GPU computing**
 - Nvidia Titan-X

General layout



- BLISS
- DAHU
- PyFAI
- FABIO
- SILX

Click icon to add picture



The spec replacement project
Started Dec 2015

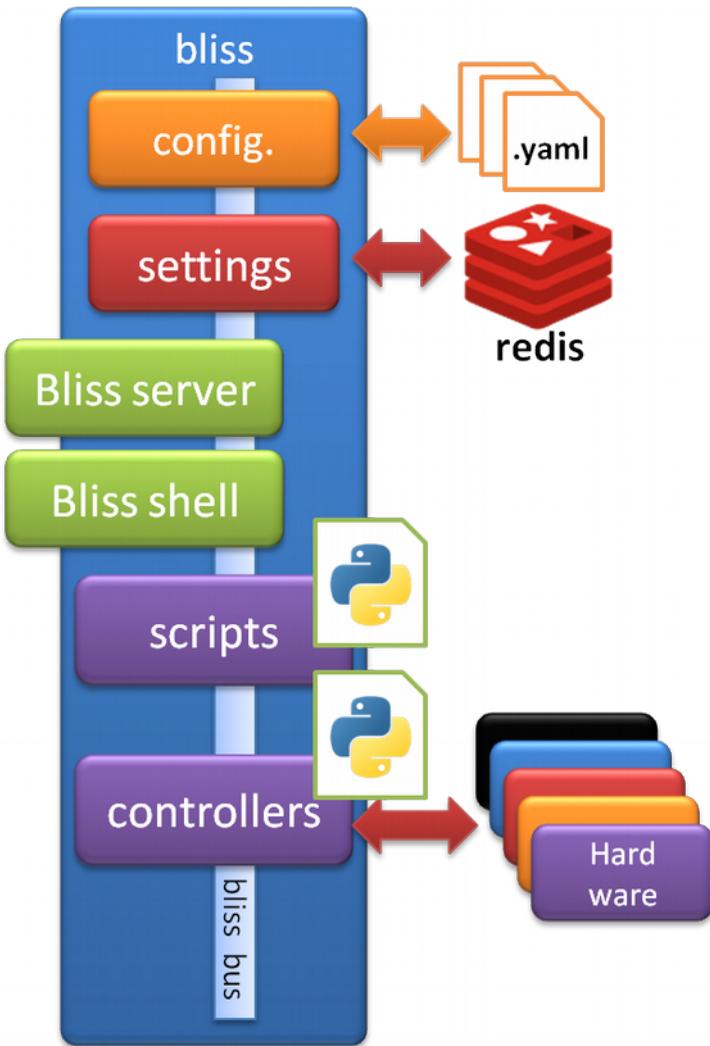
In development

Alpha version running on:

- MASSIF
- ID15 & ID31

Available on <https://gitlab.esrf.fr/bliss/bliss>





- Python control library
- Configuration
- Communication
- Hardware control
 - motion, temperature, ...
- Data recorder
- Scan engine
- Shell + GUI(s)
- Tango Bliss Server

```
coutinho@bcu01ctrl:~
```

Welcome to the bliss shell on ID31

Using sixc session

```
>>> umv(mock_bb, 20)
```

mock_bb
20.000

```
>>> wa()
```

Current Positions (user, dial)

mock_bb	mock_hg	mock_ho	mock_lb	mock_rb	mock_tb	mock_vg
20.00000	0.00000	0.00000	0.00000	0.00000	0.00000	20.00000
20.00000	0.00000	0.00000	0.00000	0.00000	0.00000	20.00000

```
>>> ascan mo
```

```
    mock_bb mock_lb mock_vg move
    mock_hg mock_rb mock_vo
    mock_ho mock_tb monkey
```

[F4] Emacs 31/31 [F3] History [F6] Paste mode [F2] Menu - CPython 2.7.6

```
Welcome to "spec" Release 6.04.01
Copyright (c) 1987-2016 Certified Scientific Software
All rights reserved

(Portions derived from a program developed at Harvard University.)
(Linked with BSD libedit library for command line editing.)

Using "/users/blissadm/spec/debian6/spec.d"
for auxiliary file directory (SPECD).
Faking a state named "esrf".
Using your state file from /dev/esrf.

Getting configuration parameters from "SPECD/spec/config".

=
Type h changes for info on latest changes.
Browse to http://www.certif.com for complete documentation.
=

Reading file "SPECD/site.mac".
Reading file "/users/blissadm/local/spec/macros/ID00setup.mac" (level 2).

Doing SETUP.
Reading file "SPECD/setup".
Reading file "SPECD/spec/setup".
bliss info: 'spec' session configured with 'ID00/bliss/spec'

62.SPEC> b_run wa()
Current Positions (user, dial)



| mock_bb  | mock_hg | mock_ho | mock_lb | mock_rb | mock_tb  | mock_vg  | mock_vo   |
|----------|---------|---------|---------|---------|----------|----------|-----------|
| 35.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | -1.00000 | 34.00000 | -18.00000 |
| 0.00000  | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000  | 34.00000 | -18.00000 |


62.SPEC>
```



ESRF - ID31 configuration application

ID 31 ESRF - ID31 Filter for... Clear filter Add...

Files Objects Instruments

Name: iceid314
Controller: MDS4M
Host: iceid314
Port: 3000

Name	Unit	Steps p/unit	Velocity (units/s ²)	Acceleration (units.s ⁻²)	Backlash (unit)	Low limit (unit)	High limit (unit)
srot	deg	400	5	20	0.1	-1000	1000
sby	mm	200	5	20	0.1	-1000	1000
sbz	mm	200	5	20	0.1	-1000	1000
tx	mm	-100.0	5.0	20.0	0.1	-1000.0	1000.0
mz	mm	200	5	20	0.1	-1000	1000
oy	mm	250.0	1.5	20.0	0.1	-1000.0	1000.0
oz	mm	20000.0	0.1	4.0	0.1	-20.0	20.0
optch	deg	400.0	5.0	20.0	0.1	-1000.0	1000.0
say2	mm	-200	2	8	0.1	-1000	1000
camt	deg	200.0	2.0	20.0	0.1	-1000.0	1000.0

Dahu is a lightweight plugin based framework...

... technically a JSON-RPC server over Tango written in Python

- **The *dahu* server executes jobs**

- Each job lives in its own thread.
- Each job executes one plugin
- The job is responsible for de/serializing JSON string coming from Tango



Plugins are Python classes or functions

- Plugins are dynamically loaded from python modules
- Plugins have a single input and output: *simple* dictionaries.

- **Jobs can be re-processed offline**
- **Lightweight means limited overhead:**

- 1 μ s for a dummy plugin execution
- 150 μ s for a dummy job
- 300 μ s when called from Tango (old figures, sorry)

Available from <https://github.com/kif/UPBL09a>

Simple example of online data analysis using *dahu*

On the server side:

Define the function (in demo.py):

```
import pyFAI, fabio

def integrate_simple(poni, image, curve)
    ai = pyFAI.load(poni)
    img = fabio.open(image).data
    ai.integrate1D(img, 1000, filename=curve)
    return {"out_file":curve}
```

Create the plugin:

```
from dahu.plugin import plugin_from_function

plugin_from_function(integrate_simple)
```

On the client side:

```
import PyTango, json

dahu = PyTango.DeviceProxy("dau/dahu/1")

plugin = 'demo.integrate_simple'

inp = {'poni': 'example.poni',
        'image': 'example.tif',
        'curve': 'integrated.dat'}

jid = dahu.startJob([plugin,
                     json.dumps(inp)])
```

- Reprocess one/multiple job from the command line:

```
dahu-reprocess job_description_1234.json  
runs jobs sequentially on the computer
```



- **Library for accessing diffraction images**
 - Fruit of the Fable collaboration (2007-2009)
 - Supports most detectors format, including CBF and HDF5
 - Now integrated as part of the *silx* project

- **Changed of license:**  → 
 - No more enforcement to publish modifications
 - More open towards industry
 - Compatible with the policy for *silx*



- **Simple profiling:**

```
%timeit data = fabio.open("/nvme/test.cbf").data  
10 ms/image just for reading one image
```

- **Advanced profiling:**

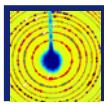
- Total time: 30ms
 - **15 ms for byte-offset decompression**
 - **10 ms for MD5 hash calculation**
 - **1 ms parsing the CIF structure**
 - **1 ms for reading the file**

- **Solutions:**

- Skip the checksum verification
- Use a pool of decompressor thread, optimum: 6 readers

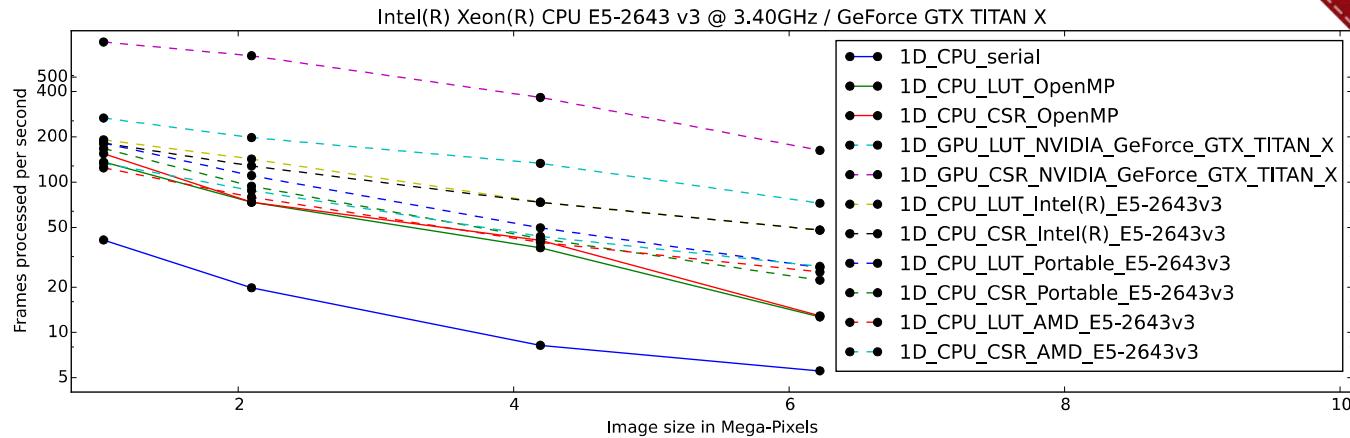
- **Better solutions**

- Implement the byte-offset decompression on GPU → Far from trivial

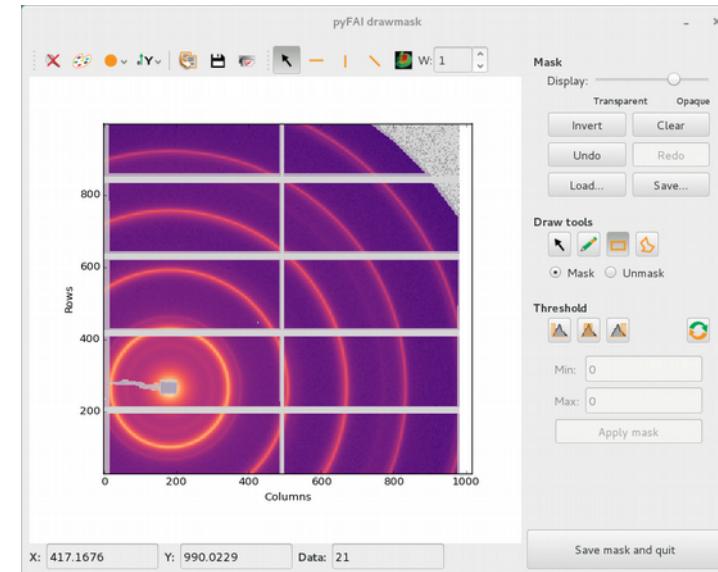


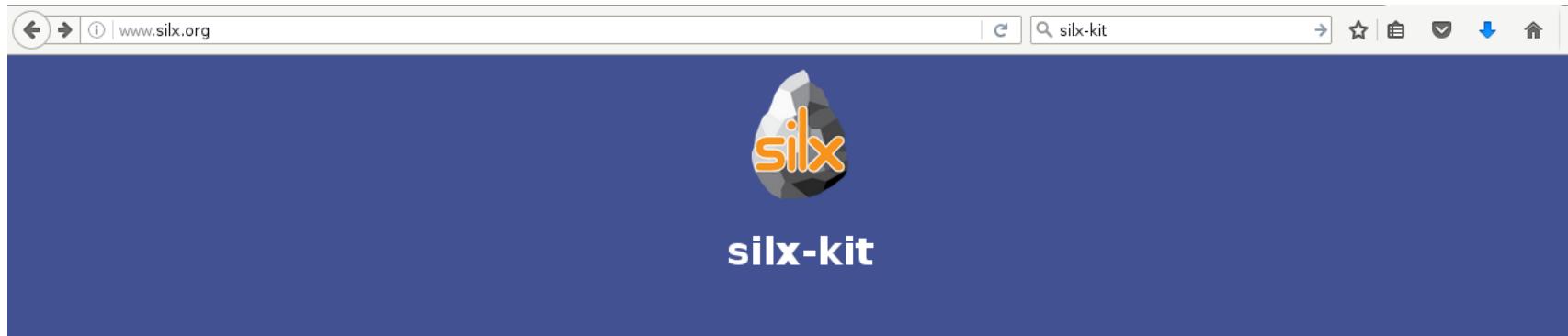
PyFAI: fast azimuthal integration

Fork me on GitHub



- **~2 ms processing per image**
- **Graphical user interface in preparation**
 - Will rely on silx for the GUI
 - Mask tools already using *silx*
- **Version 0.13 planed in the coming weeks**
- **May change license as well**





silx

Scientific Library for
eXperimentalists



Resources

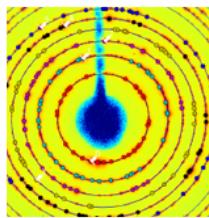
- [silx on GitHub](#)
- [Wheels and source on PyPi](#)
- [Installation instructions](#)

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



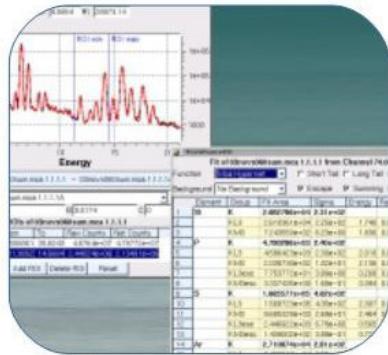
Resources

- [FabIO on GitHub](#)
- [Wheels and source on PyPi](#)
- [Installation instructions](#)

Documentation

- [Latest release](#)
- [Nightly build](#)

Mainly Pierre Knobel ...

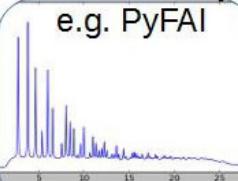


Standard Apps e.g.
PyMCA, PyDIF

... and Valentin Valls

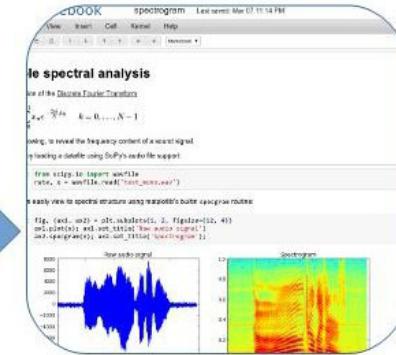
Mainly Jérôme Kieffer

Online data analysis



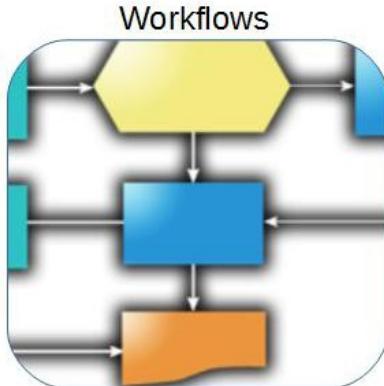
e.g. PyFAI

Mainly Thomas Vincent



Ipython Notebook

General Purpose
Core Toolkit



Mainly Henri Payno

External
Libraries +
Apps

Extensions

Extension library
e.g. PyHST, ...

- **Public project hosted at github**

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

- **Continuous testing**

Linux, Windows and MacOSX

- **Nightly builds**

- Debian packages
- RPM on target

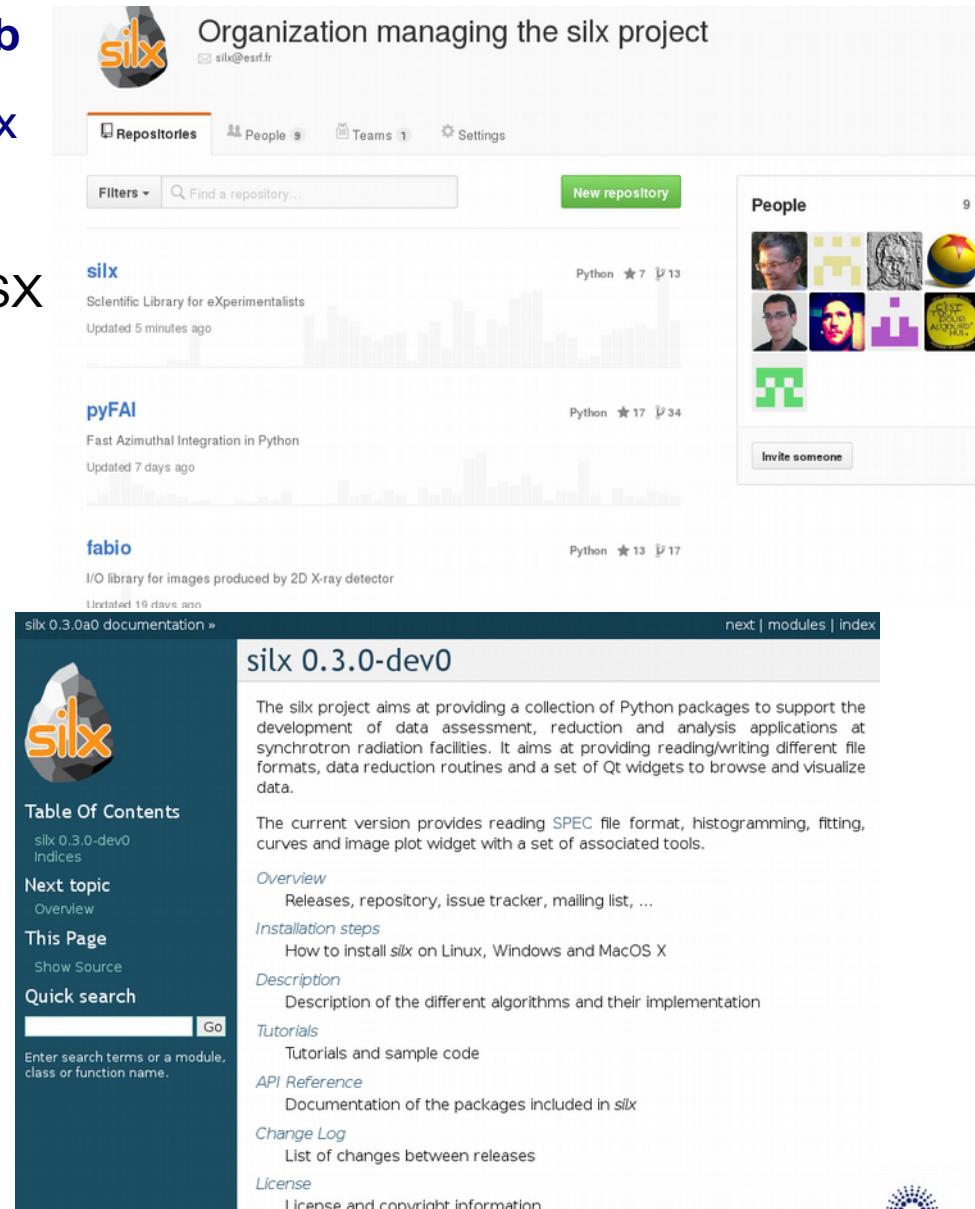
- **Weekly meetings**

- **Quarterly releases**

- **Code camps before release**

- **Continuous documentation**

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



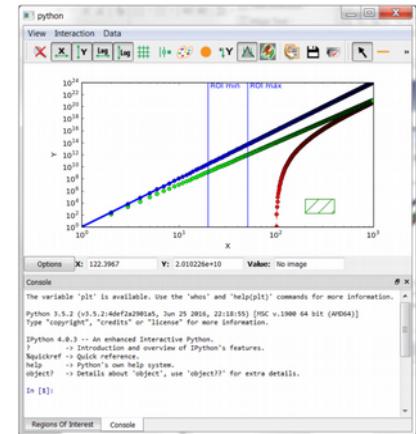
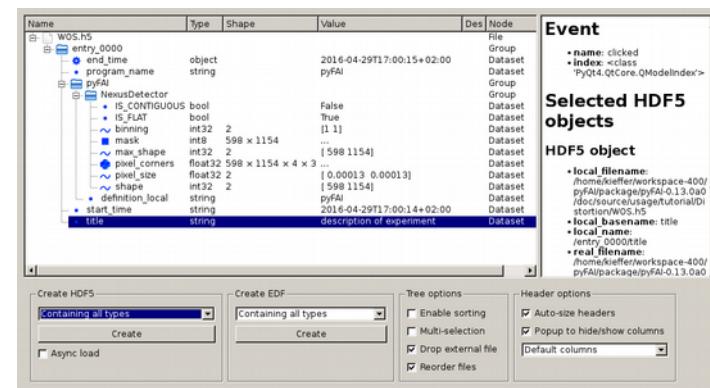
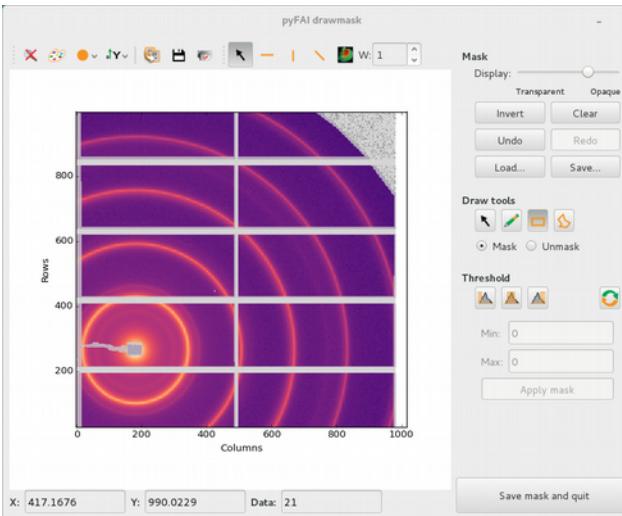
The screenshot shows the GitHub organization page for 'silx' and the documentation for 'silx 0.3.0-dev0'.
Github Organization Page:

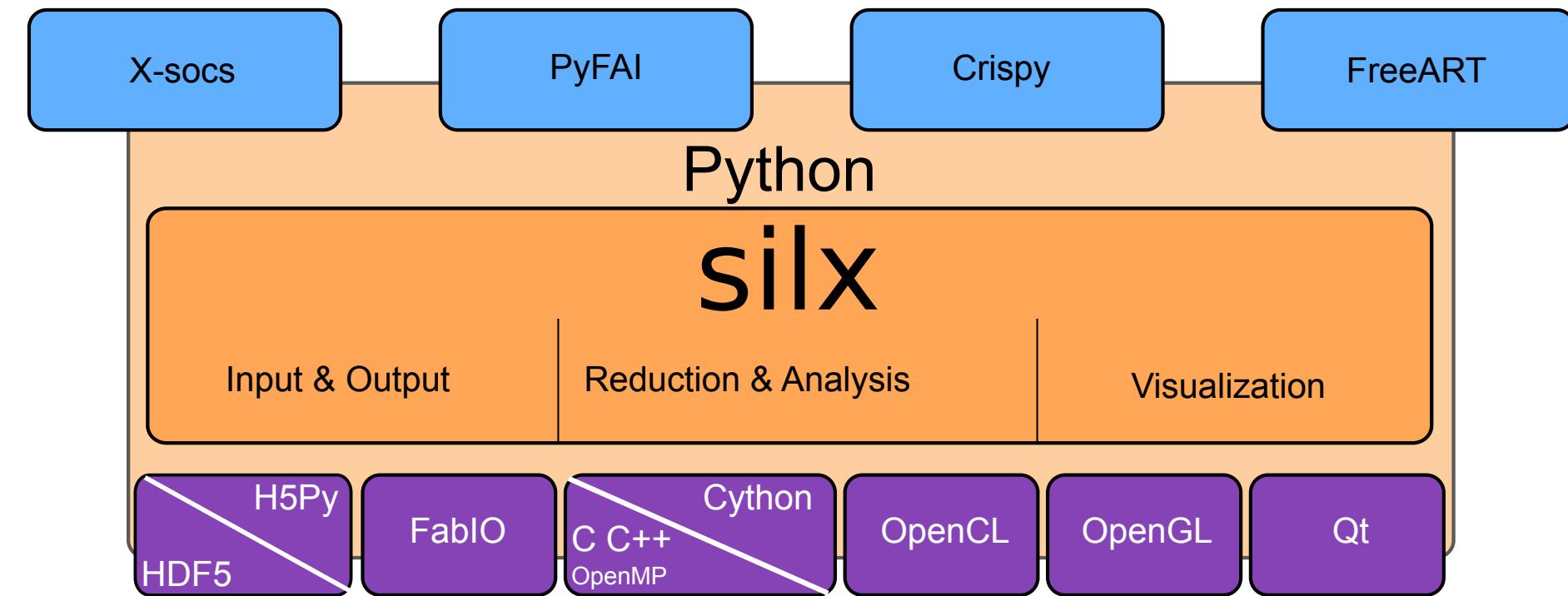
- Repos:** silx, pyFAI, fabio.
- Metrics:** Python, ★ 7, 13, Python, ★ 17, 34, Python, ★ 13, 17.
- Members:** People section with 9 members.
- Actions:** New repository button, Invite someone button.

Documentation Page:

- Header:** silx 0.3.0a0 documentation » next | modules | index
- Content:** silx 0.3.0-dev0
- Description:** The silx project aims at providing a collection of Python packages to support the development of data assessment, reduction and analysis applications at synchrotron radiation facilities. It aims at providing reading/writing different file formats, data reduction routines and a set of Qt widgets to browse and visualize data.
- Current Version:** The current version provides reading SPEC file format, histogramming, fitting, curves and image plot widget with a set of associated tools.
- Links:** Overview, Releases, repository, issue tracker, mailing list, ..., Installation steps, How to install silx on Linux, Windows and Macos X, Description, Description of the different algorithms and their implementation, Tutorials, Tutorials and sample code, API Reference, Documentation of the packages included in silx, Change Log, List of changes between releases, License, License and copyright information.

- **A toolbox for data analysis programs**
 - A uniform way to access data: a la HDF5
 - Common reduction routines (histogram, image alignment, fit)
 - A set of advanced widgets for:
 - **Browse spec, images, HDF5 files**
 - **Plot curves and fit**
 - **Display images, profiles, draw masks**
- **Resources: ~3 FTE over 6 persons**
- **Available everywhere: Linux, MacOSX, Windows**





Conclusion

- One of the most challenging experiment on the engineering point of view
- Very high data-rate, big-data on the radar
- 18 month of work for engineers from:
 - **The ID15-beamline (Marco, Gavin, Thomas)**
 - **The beamline control unit (Seb, Manu, Tiago, Alejandro)**
 - **The computer services (Gabi, Benoit)**
 - **Under the coordination of Jens Meyer**
- The data reduction will be performed on:
 - **CPU for decompressing the images (6 threads)**
 - **GPU for azimuthal integration**
 - **180 Hz achieved in production like mode but:**
 - Multiple bunches of data can be parallelized
 - 300Hz have been achieved on test computer
 - **Data may be exported in HDF5 (or not)**

Acknowledgment

- **ID15:**
 - Marco Di Michiel
 - Gavin Vaughan
- **Data scientists:**
 - Vincent Favre-Nicolin
 - Marius Retegan
- **Computing services:**
 - Benoit Rousselle
 - Gabriele Förstner
- **Beamline control unit**
 - Jens Meyer
 - Sébastien Petitdemange
 - Tiago Coutinho
 - Matias Guijarro
- **Data analysis unit**
 - V. Armando Solé
 - Thomas Vincent
 - Valentin Valls
 - Henri Payno
 - Pierre Knobel
 - Damien Naudet