The Bluesky Project Contributors

National Synchrotron Light Source II

("Giant X-ray beam")

NSLS-II is a "User Facility"

- Scientific Staff spend 20% time on experiments, the rest on user support
- Users mail samples or visit for ~1–10 days
- 28 active instruments ("beamlines"), 60-70 planned

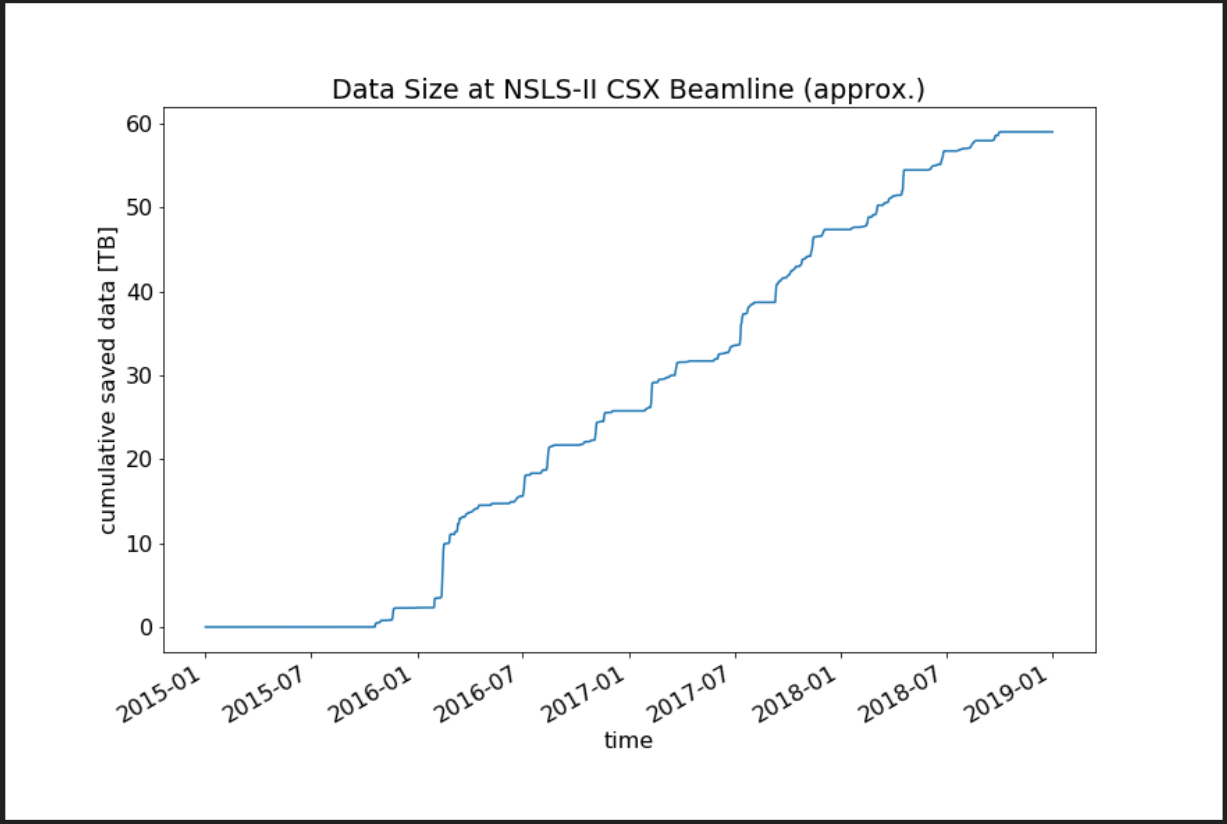
USER FACILITIES HAVE A DATA PROBLEM

We can learn a lot from particle physics, astronomy, and climate science....but we have some unique problems too.

What changed to make data problems harder?

- Sources got brighter; detectors got larger and faster: greater data *velocity* and *volume*.
- This exposes the *variety* problem we have at user facilities:
  - Large and changing collection of instruments
  - Wide span of data rates, structures, and access patterns
  - Mix of well-established data processing procedures and original, improvised techniques
- Multi-modal analysis makes this an N^2, ... problem.

"Big data is whatever is larger than your field is used to."



A spot check for data volume at an NSLS-II Project Beamline so far...

What changed to make data problems easier?

Free, open-source scientific software exploded.

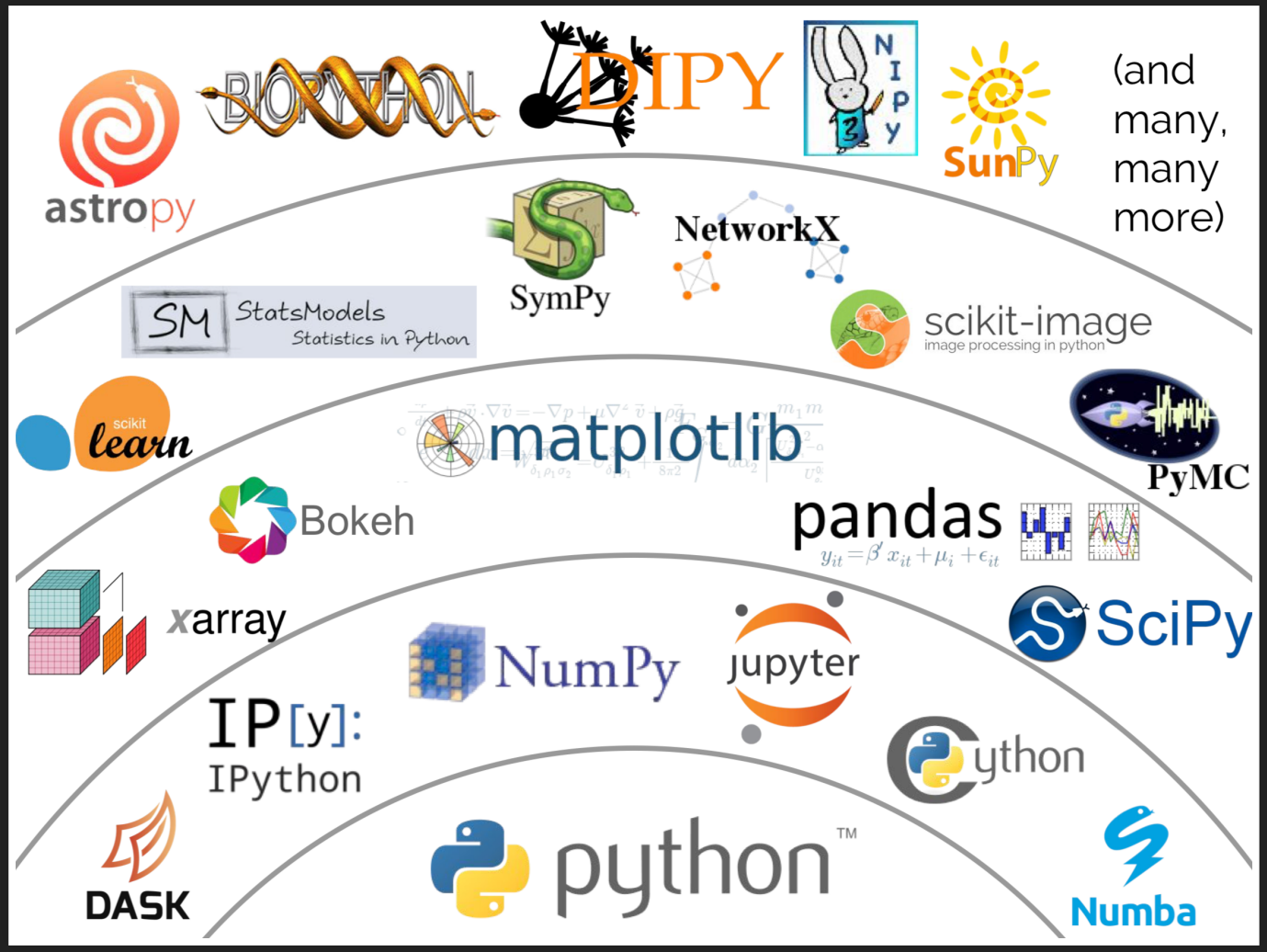


Figure Credit: "The Unexpected Effectiveness of Python in Science", PyCon 2017

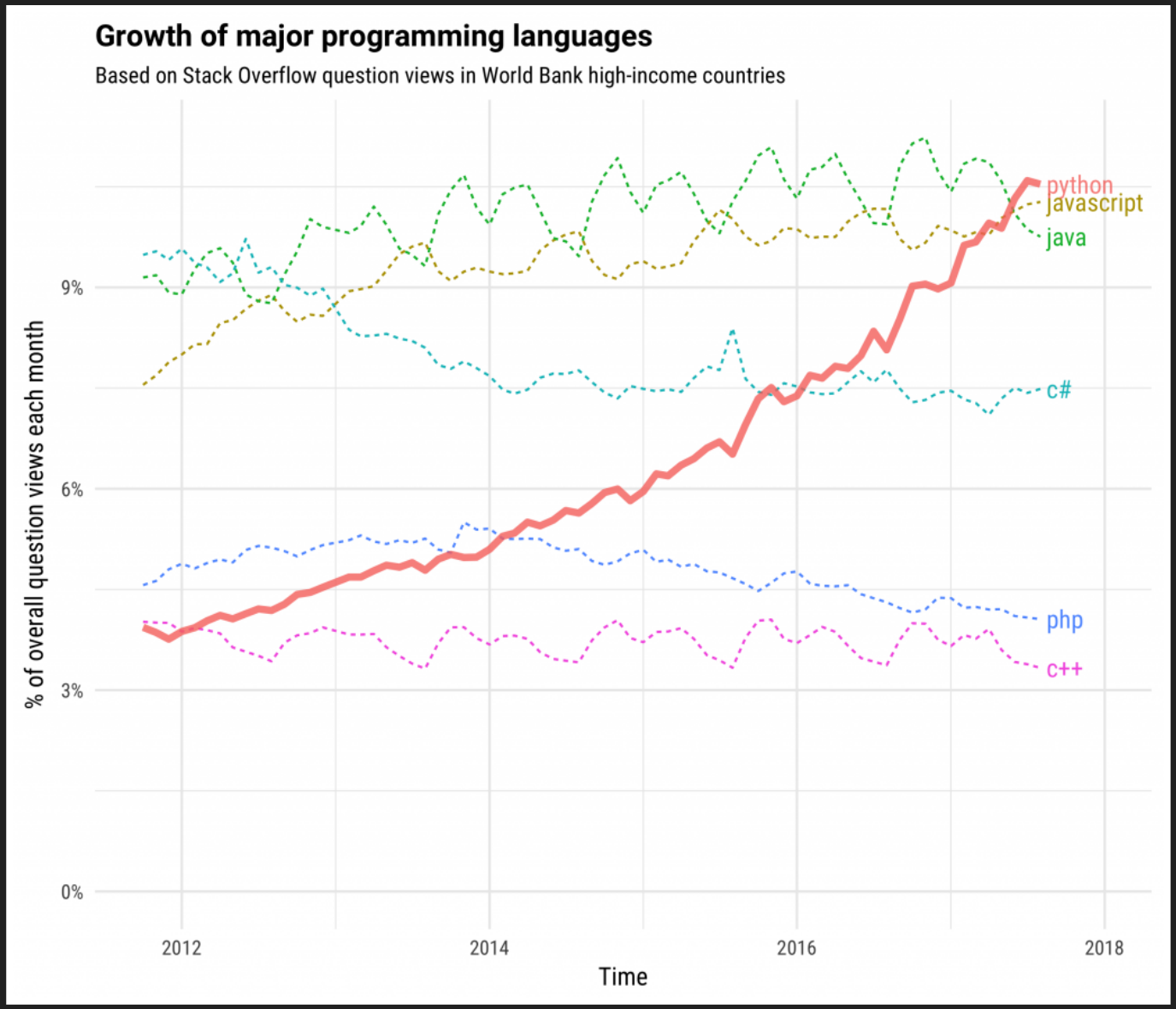


Figure Credit: Stack Overflow Blog https://stackoverflow.blog/2017/09/06/incredible-growth-python/

HPC is becoming more accessible.

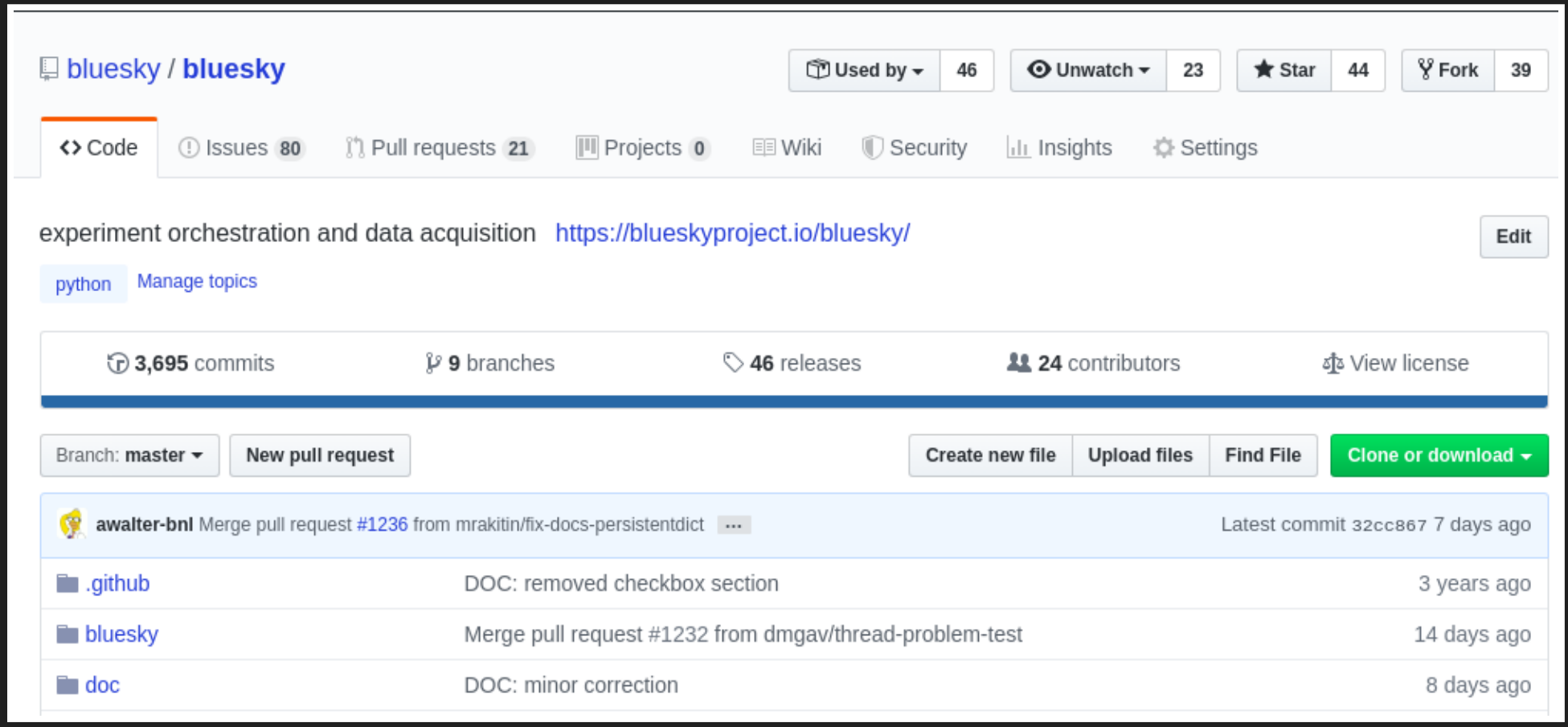
One inviting example: jupyter.nersc.gov

- Jupyter as a familiar, user-friendly portal
- Dask for familiar numpy/pandas idioms distributed over many nodes

Also: Commodity cloud-based tools

Lately it's become more practical to work openly and collaboratively....

- across instruments within a facility
- between facilities
- with outside communities with similar data problems (e.g. climate science)



...which is not a *new* idea, but ease-of-use matters.

99-80
September 8, 1999

For more information, contact:
Diane Greenberg, (631)344-2347, greenb@bnl.gov,
or Mona S. Rowe, (631) 344-5056, mrowe@bnl.gov

Brookhaven Lab to Hold "Open Source/Open Science" Conference, Oct. 2

UPTON, NY - The U.S. Department of Energy's Brookhaven National Laboratory will host a conference titled "Open Source/Open Science" on Saturday, October 2, from 8 a.m. to 5 p.m. in the Laboratory's Berkner Hall. The public is invited to the conference, but pre-registration is required. The fee for the conference is $25, which includes lunch.

Open Source software is free software, typically released over the Internet so that a broad audience can evaluate and test it. The conference will bring together scientists and developers to exchange ideas about the use of Open Source software in scientific research at Brookhaven, as well as at other laboratories and universities. Talks by invited speakers, posters, demonstrations and vendor exhibits will be featured at the conference. Also offered will be tours of some of Brookhaven Lab's facilities in which Open Source software is used.

Among the talks and speakers featured at the conference will be:

- "What is Open Source?" Bruce Perens, the principal author of the Open Source definition
- "The Open Science Project," by Dan Gezelter from openscience.org
- "Open Source Computing for BNL's Relativistic Heavy Ion Collider," Tom Throwe, Brookhaven Lab
- "Open Visualization Data Explorer," Bill Horn, IBM
- "Open GL and GLX: High Performance 3D Graphics for Linux," Jon Leech, SGI
- "ACEDB: An Open Source Object-Oriented Database," Lincoln Stein, Cold Spring Harbor Laboratory
- "Open Source in Medical Imaging," Bill Rooney, Brookhaven Lab
- "Using Beowulf for Macromolecular Crystallography at the National Synchrotron Light Source," Malcolm Capel, Brookhaven Lab

Also, a panel of experts will discuss the issues that must be overcome by federal facilities in order to use and contribute to Open Source technologies.

To register for the conference, go to the Web site http://openscience.bnl.gov, and follow the registration instructions. The registration deadline is September 26. For more information, call (516)344-3582 or (516)344-5682.

Brookhaven National Laboratory is located on William Floyd Parkway (County Road 46), one-and-a-half miles north of Exit 68 on the Long Island Expressway.

The U.S. Department of Energy's Brookhaven National Laboratory creates and operates major facilities available to university, industrial and government personnel for basic and applied research in the physical, biomedical and environmental sciences, and in selected energy technologies. The Laboratory is operated by Brookhaven Science Associates, a not-for-profit research management company, under contract with the U.S. Department of Energy.

* * *

STATUS QUO:
DATA AND METADATA ARE SCATTERED

- Some critical context is only in people's heads
- Many file formats (tif, cbf, Nexus, other HDF5, proprietary, ...)
- `meta_data_in_37K_fname_005_NaCl_cal.t`
- "Magic numbers" buried in analysis tools
- Notes in paper notebooks

What's the problem?

- Not machine-readable or searchable
- Relationship between any two pieces of data unclear
- Inhibits multi-modal work
- Inhibits code reuse
- Not streaming friendly

What do we need to systematically track?

Experimental Data

Analysis needs more than "primary" data stream:

- Timestamps
- Secondary measurements
- "Fixed" experimental values
- Calibration / beam-line configuration data
- Hardware settings
- Hardware diagnostics
- Physical details of the hardware

Sample Data

- What is the sample?
- What is the contrast mechanism?
- Why are we looking at it?
- How was it prepared?

Bureaucratic & Management Information

- Where is the data and how to get it?
- Who took the data?
- Who owns or can access the data?
- How long will we keep the data?

DESIGN GOALS

both technical and sociological

for an end-to-end data acquisition and analysis solution that leverages data science libraries

Technical Goals

Technical Goals

- Generic across science domains

Technical Goals

- Generic across science domains
- Lightweight

Technical Goals

- Generic across science domains
- Lightweight
- Put metadata in a predictable place

Technical Goals

- Generic across science domains
- Lightweight
- Put metadata in a predictable place
- Handle asynchronous data streams

Technical Goals

- Generic across science domains
- Lightweight
- Put metadata in a predictable place
- Handle asynchronous data streams
- Support multi-modal: simultaneous, cross-beamline, cross-facility

Technical Goals

- Generic across science domains
- Lightweight
- Put metadata in a predictable place
- Handle asynchronous data streams
- Support multi-modal: simultaneous, cross-beamline, cross-facility
- Support streaming

Technical Goals

- Generic across science domains
- Lightweight
- Put metadata in a predictable place
- Handle asynchronous data streams
- Support multi-modal: simultaneous, cross-beamline, cross-facility
- Support streaming
- Cloud friendly

Technical Goals

- Generic across science domains
- Lightweight
- Put metadata in a predictable place
- Handle asynchronous data streams
- Support multi-modal: simultaneous, cross-beamline, cross-facility
- Support streaming
- Cloud friendly
- Integrate with third-party (meta)data sources

Sociological Goals

Sociological Goals

- Overcome "not-invented-here"-ism.

Sociological Goals

- Overcome "not-invented-here"-ism.
- Make *co-developed but separately useful* components with well-defined boundaries which can be adopted piecemeal by other facilities.

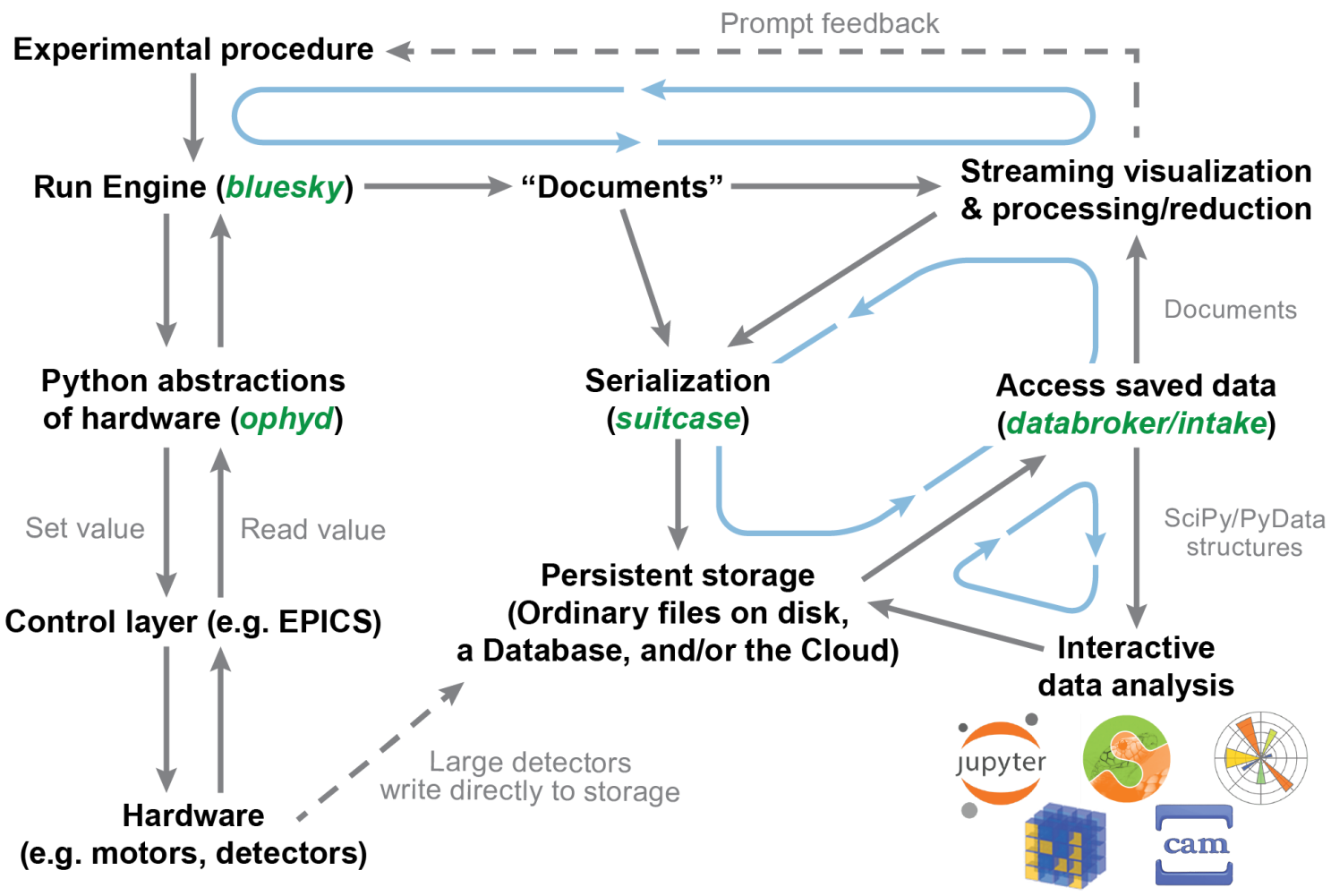
Sociological Goals

- Overcome "not-invented-here"-ism.
- Make *co-developed but separately useful* components with well-defined boundaries which can be adopted piecemeal by other facilities.
- Drawing inspiration from the numpy project, embrace *protocols* and *interfaces* for interoperability.

It's working!

BLUESKY ARCHITECTURE

bluesky


Prompt feedback
Experimental procedure
Run Engine (bluesky)
"Documents"
Streaming visualization & processing/reduction
Documents
Python abstractions of hardware (ophyd)
Serialization (suitcase)
Access saved data (databroker/intake)
Set value
Read value
SciPy/PyData structures
Control layer (e.g. EPICS)
Persistent storage (Ordinary files on disk, a Database, and/or the Cloud)
Interactive data analysis
Large detectors write directly to storage
Hardware (e.g. motors, detectors)
jupyter
cam

Layered design of Python libraries that are:

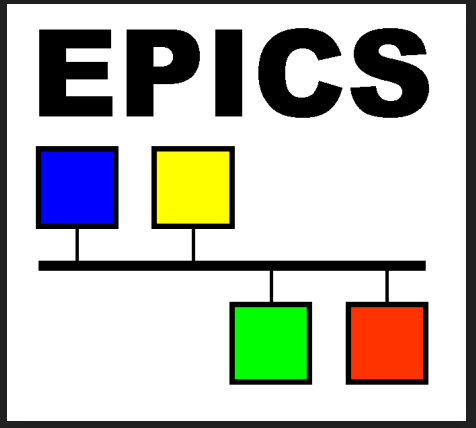
- co-developed and compatible...
- ...but individually usable and useful
- with well-defined programmatic interfaces

Looking at each component, from the bottom up....

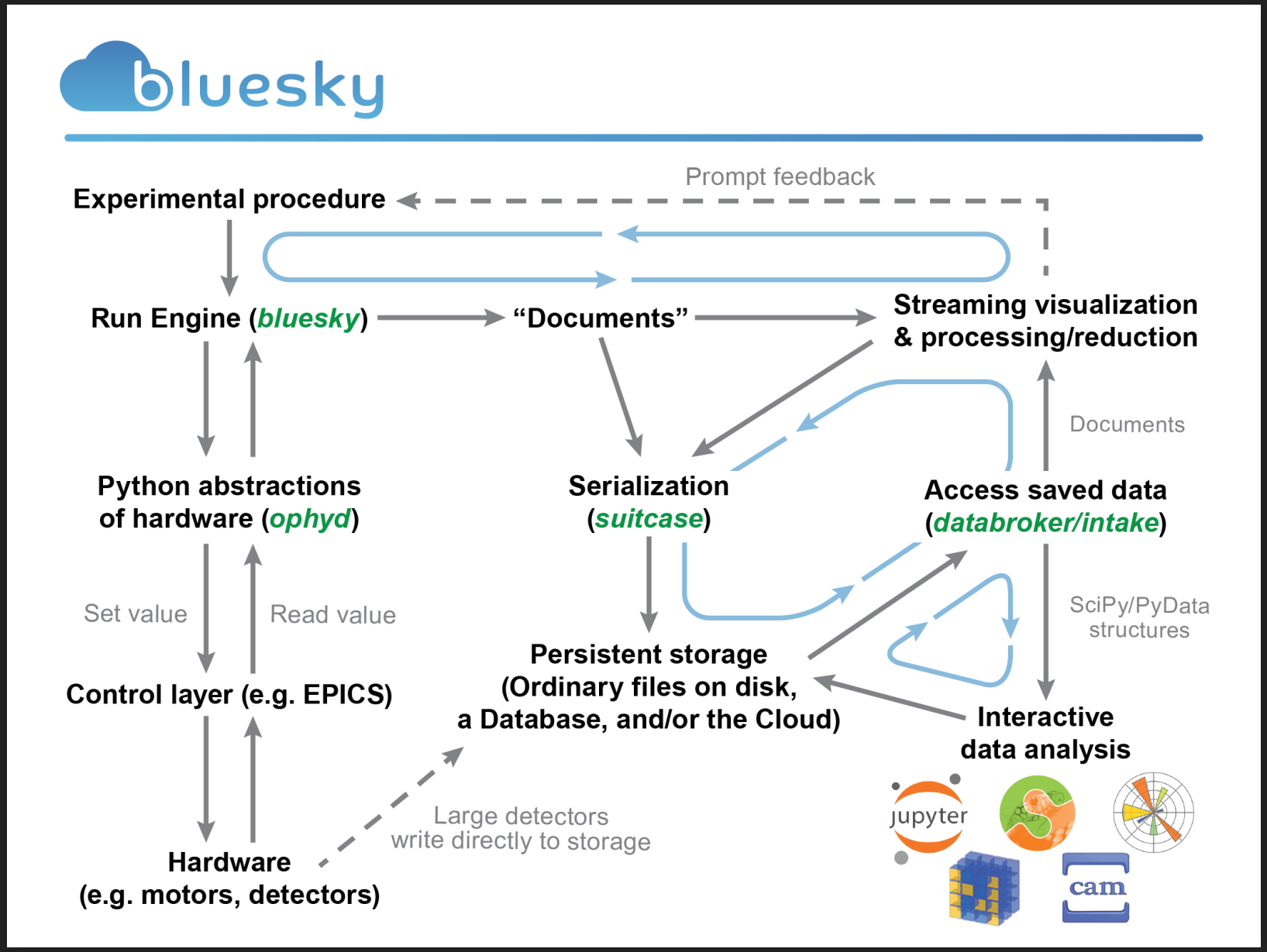
Device Drivers and Underlying Control Layer(s)

You might have a pile of hardware that communicates over one or more of:

- Experimental Physics and Industrial Control System (EPICS)
- LabView
- Some other standard
- Some vendor-specific, one-off serial or socket protocol



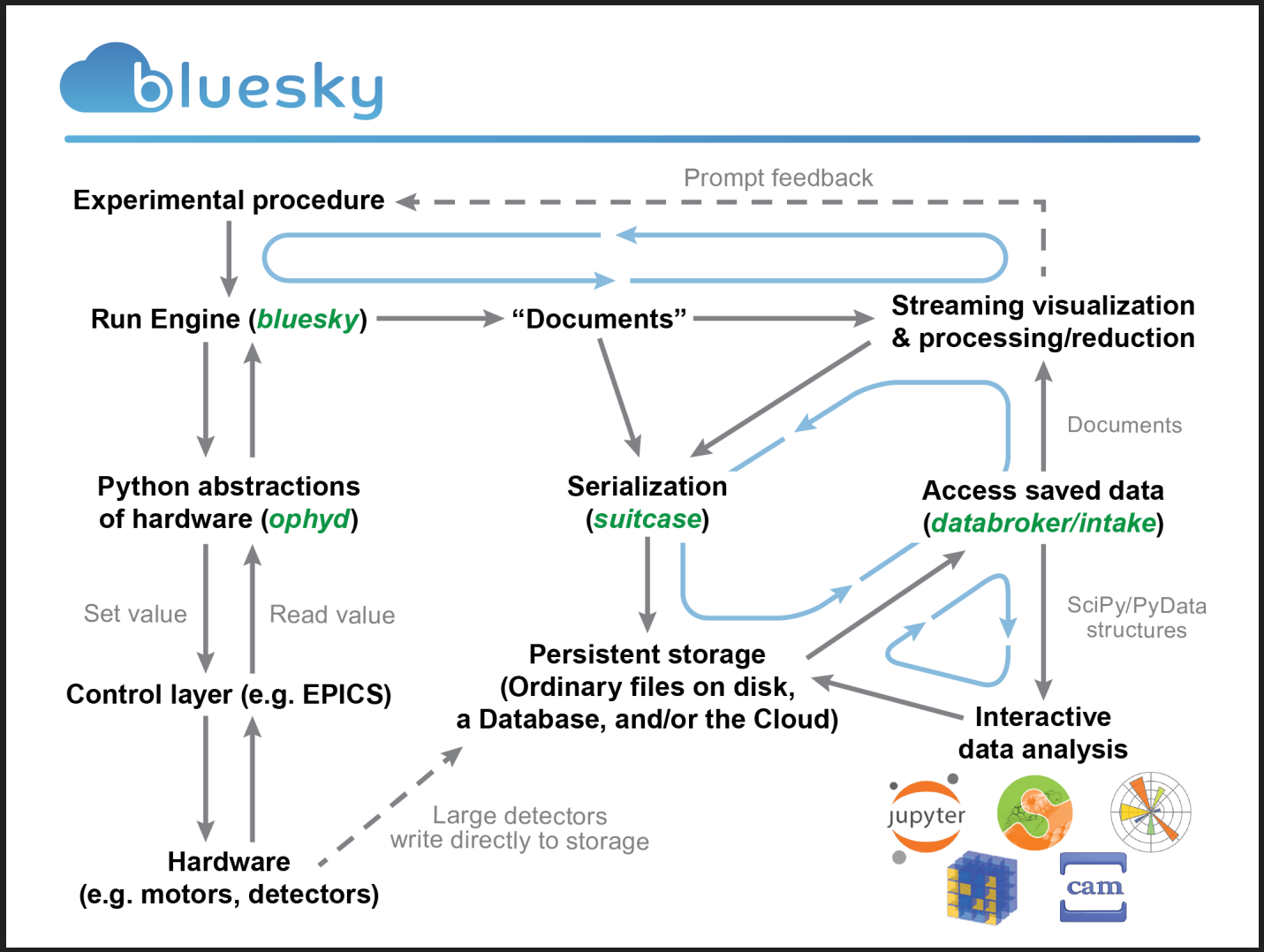
Ophyd abstracts over the specific control layer.



Ophyd: a hardware abstraction layer

- Put the control layer behind a **high-level interface** with methods like `trigger()`, `read()`, and `set(...)`.
- **Group** individual signals into logical "Devices" to be configured and used as one unit.
- Assign signals and devices **human-friendly names** that propagate into metadata.
- **Categorize** signals by "kind" (primary reading, configuration, engineering/debugging).

Bluesky abstracts over hardware.



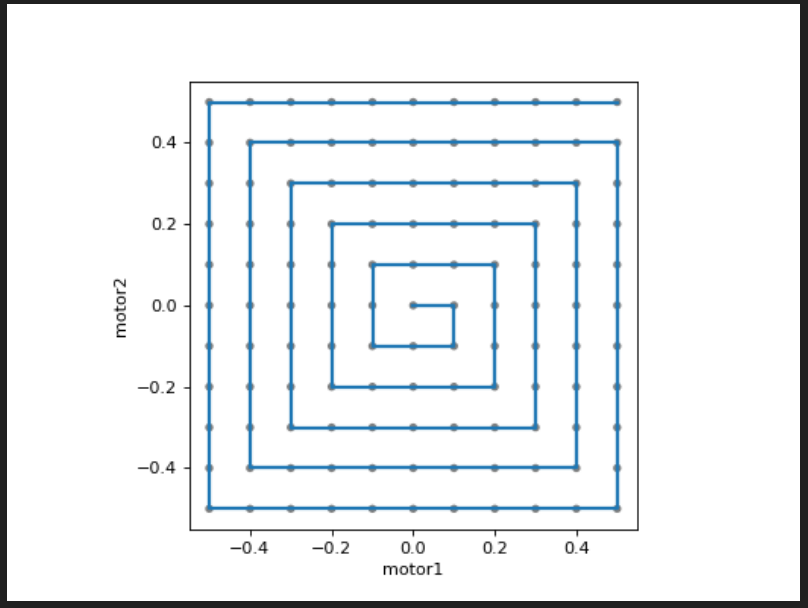
Bluesky: an experiment specification and orchestration engine

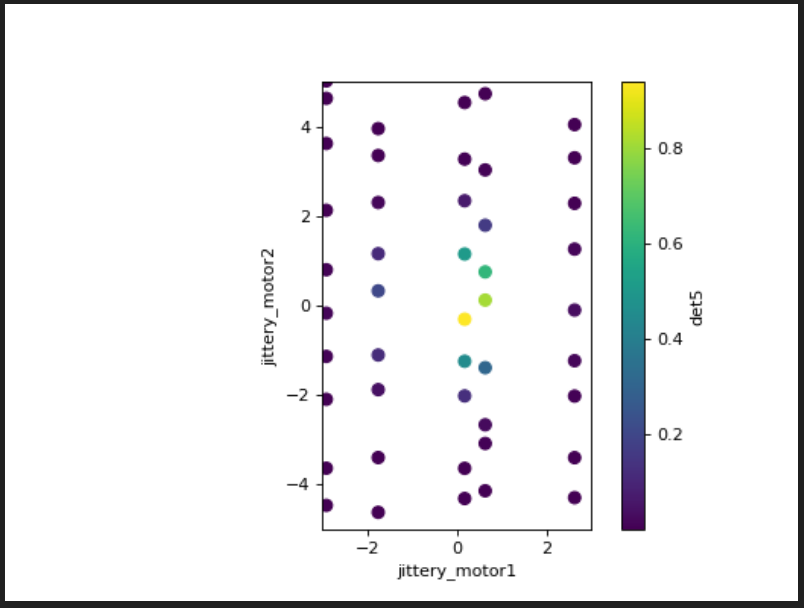
- Specify the logic of an experiment in a hardware-abstracted way. Bluesky says a detector *should* be triggered; ophyd sorts out *how*.
- First-class support for **adaptive feedback** between analysis and acquisition.
- Data is emitted in a **streaming** fashion in standard Python data structures.
- Pause/resume, robust error handling, and rich metadata capture are built in.

Mix and match (or create your own) plans...

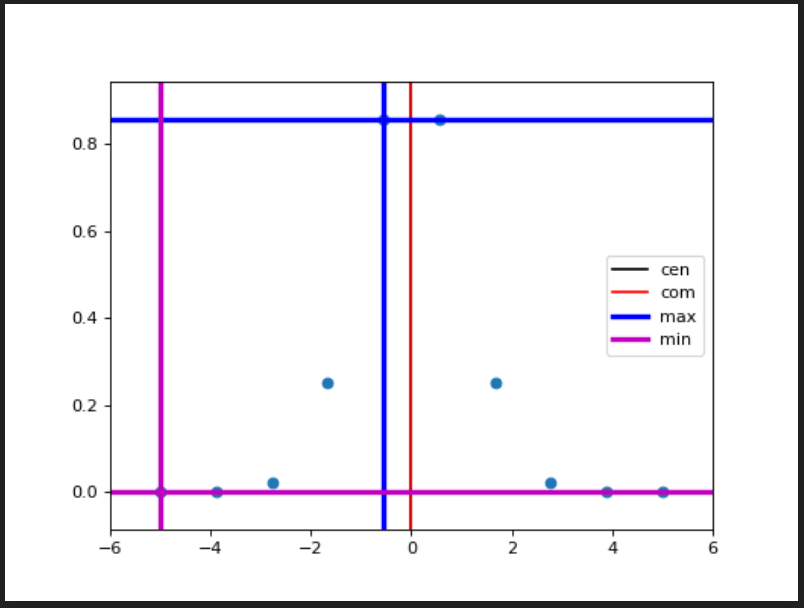
`count`	Take one or more readings from detectors.
`scan`	Scan over one multi-motor trajectory.
`rel_scan`	Scan over one multi-motor trajectory relative to current position.
`list_scan`	Scan over one or more variables in steps simultaneously (inner product).
`rel_list_scan`	Scan over one variable in steps relative to current position.
`list_grid_scan`	Scan over a mesh; each motor is on an independent trajectory.
`rel_list_grid_scan`	Scan over a mesh; each motor is on an independent trajectory.
`log_scan`	Scan over one variable in log-spaced steps.
`rel_log_scan`	Scan over one variable in log-spaced steps relative to current position.
`grid_scan`	Scan over a mesh; each motor is on an independent trajectory.
`rel_grid_scan`	Scan over a mesh relative to current position.
`scan_nd`	Scan over an arbitrary N-dimensional trajectory.
`spiral`	Spiral scan, centered around (x_start, y_start)
`spiral_fermat`	Absolute fermat spiral scan, centered around (x_start, y_start)
`spiral_square`	Absolute square spiral scan, centered around (x_center, y_center)
`rel_spiral`	Relative spiral scan

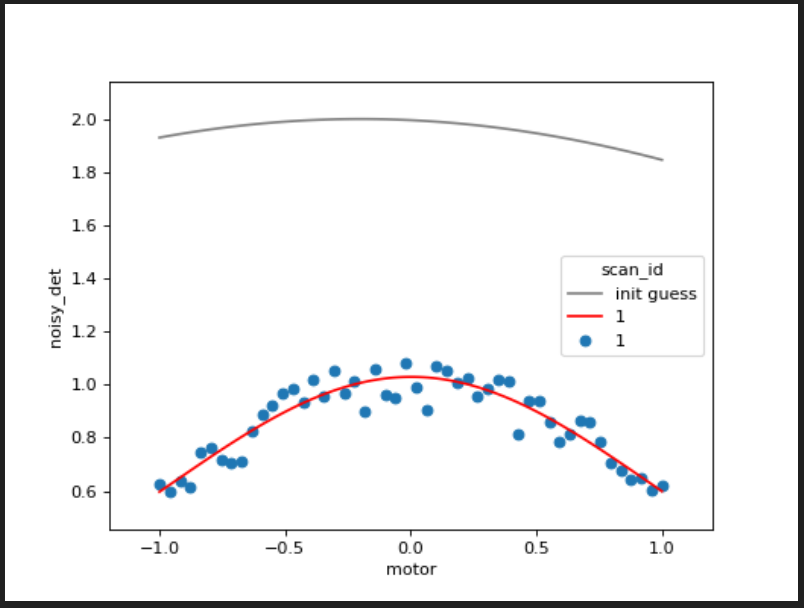
...and streaming-friendly viz...





...and streaming-friendly analysis

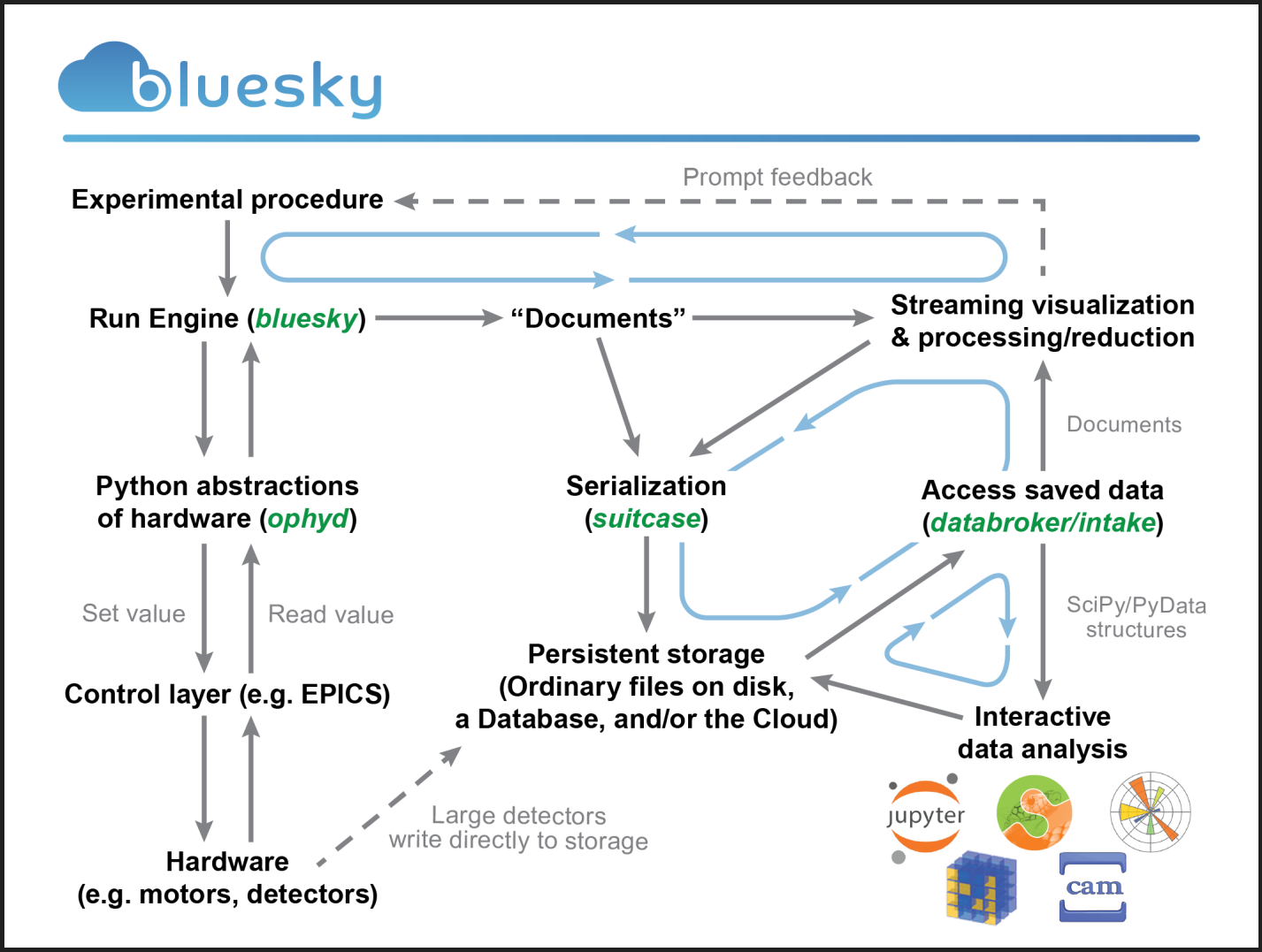




High Throughput

- Bluesky can process 30k messages/second ("message" = `trigger`, `read`, `save`, ...).
- Typically, the vast majority of its time is spent waiting for hardware to move or acquire.
- To go faster than that, use `kickoff` ("Go!") — `complete` ("Call me when you're done.") — `collect` ("Read out data asynchronously.").

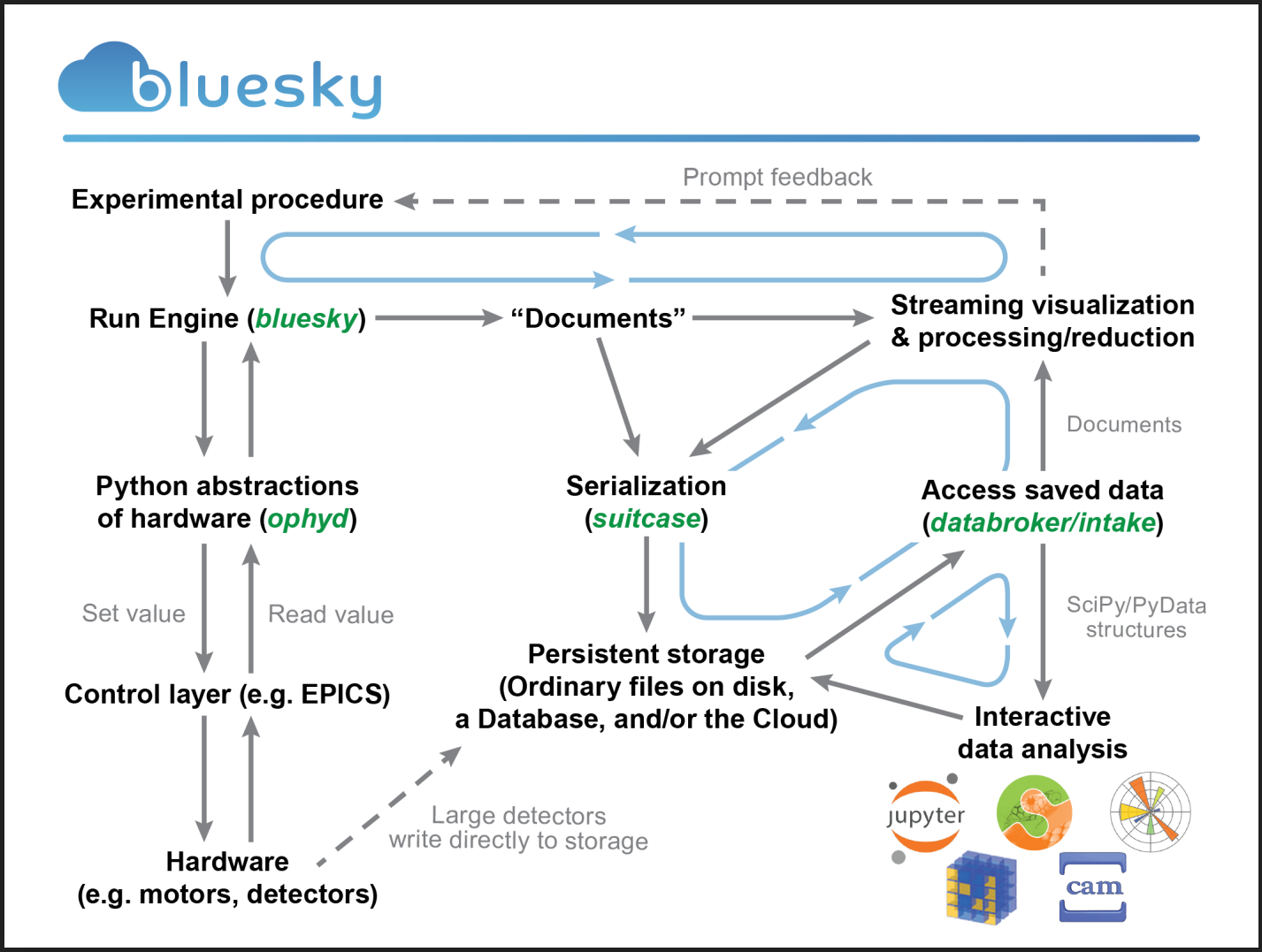
Suitcase encodes documents for storage or export.



Suitcase: store in any database or file format

- Lossless storage: MongoDB, msgpack, JSONL
- Lossy export: TIFF, CSV, specfile, ...
- Documentation on how to write a suitcase for your own format
- Use any transport you like. Write to disk (ordinary files), memory buffer, network socket, ...

DataBroker provides search, access to stored data.



DataBroker takes the hassle out of data access.

- An API on top of a database and/or file.
- Search user-provided and automatically-captured metadata.
- Exactly the same layout originally emitted by Bluesky, so consumer code does not distinguish between "online" and saved data

Keep I/O Separate from Science Logic!

Interfaces, not File Formats

- The system is **unopinionated about data formats**.
- Can change storage with no change to consumer code.
- Any file I/O happens transparently: the **user never sees files**, just gets data in memory (e.g. a numpy array, a mapping with labeled metadata).
- Your detector writes in a special format? Register a custom reader.

EMBRACE INTERFACES

The most important aspect of the Bluesky architecture are the well-defined protocols and interfaces.

Interfaces enable:

- Interoperable tools without explicit coordination
- Unforeseen applications

Interface Example: Iteration in Python

```
for x in range(10):
    ...

class MyObject:
    def __iter__(self):
        ...

for x in MyObject():
        ...
```

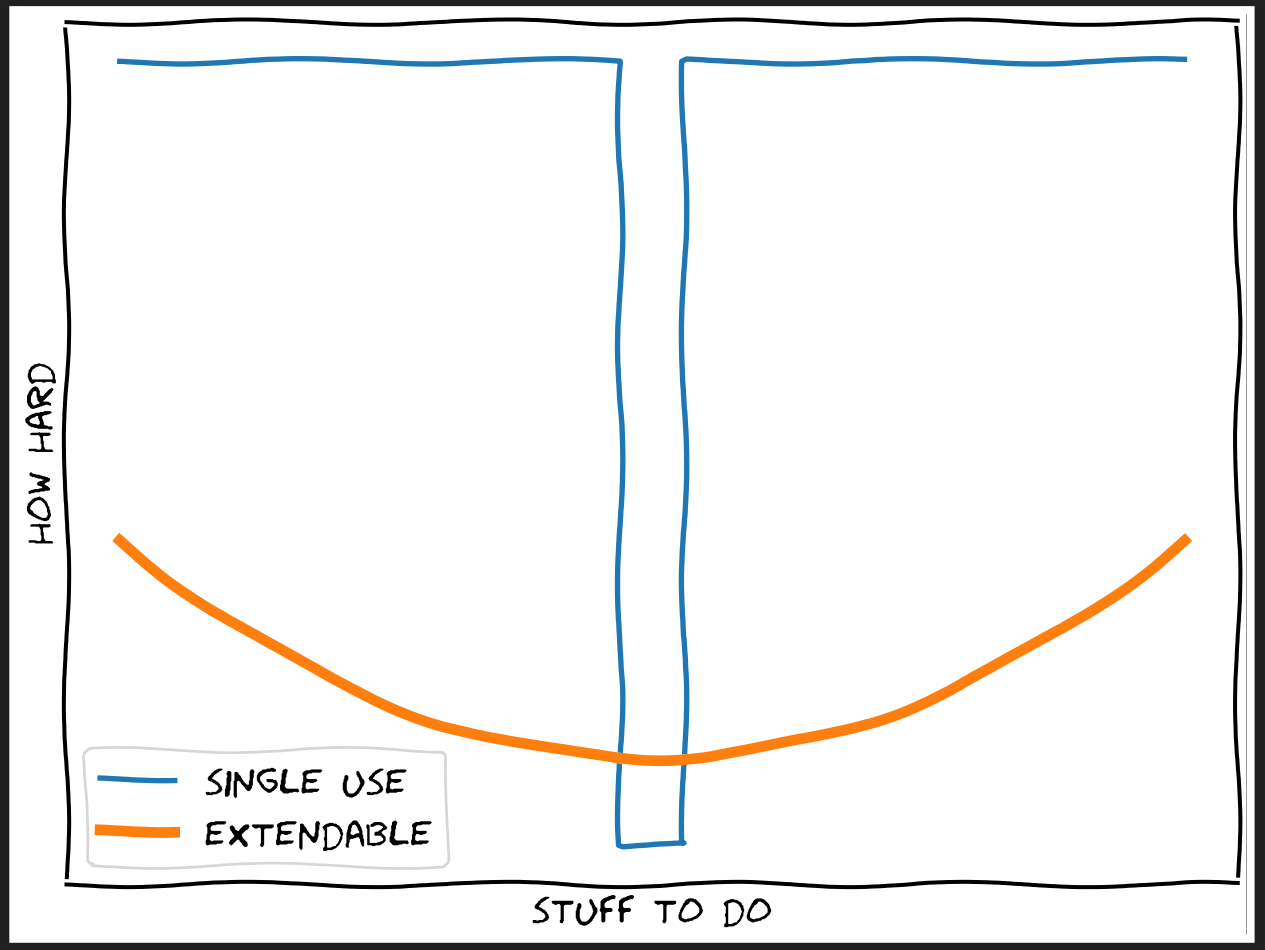
Interface Example: numpy array protocol

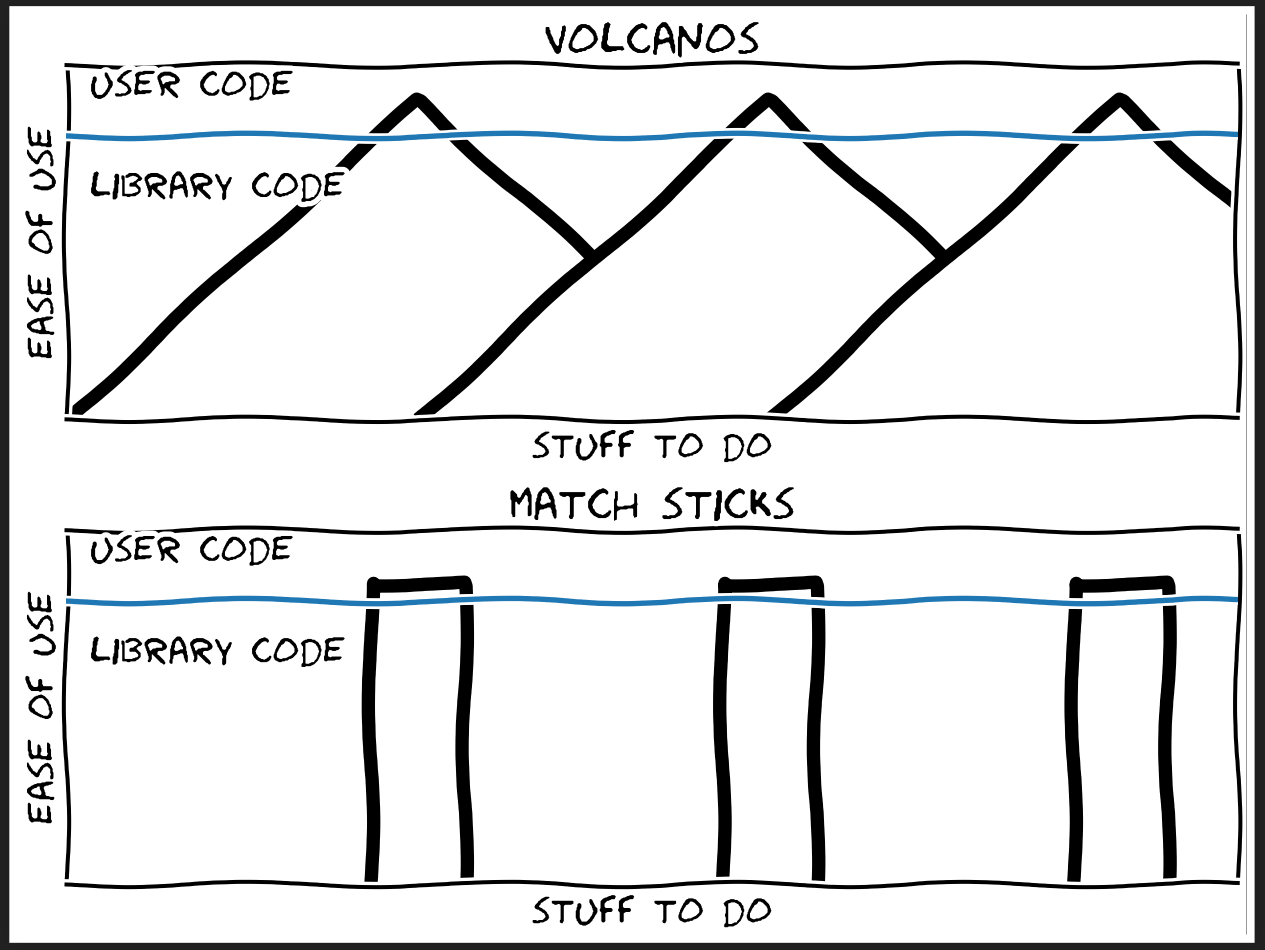
```
import pandas
import numpy

df = pandas.DataFrame({'intensity': [1,1,2,3]})
numpy.sum(df)
```

Interfaces in Bluesky

- Event Model — connects data producers to consumers
- Message protocol — connects experiment sequencing with inspection and execution
- Ophyd hardware abstraction — connects what you want to do to how to do it

EMBRACE LAYERED EXTENDABLE CODE


HOW HARD
STUFF TO DO
SINGLE USE
EXTENDABLE


VOLCANOS
USER CODE
LIBRARY CODE
EASE OF USE
STUFF TO DO
MATCH STICKS
USER CODE
LIBRARY CODE
EASE OF USE
STUFF TO DO

EMBRACE COMMUNITY OPEN-SOURCE PROCESSES

Work openly

- Use version control.
- Make new work public from the start.
- Put ideas and roadmaps on GitHub issues where others can search, read, comment.

Automated tests are essential

They enable people to try new ideas with confidence.

- Ensure that we don't accidentally break our ability to recreate important results.
- Ensure that *my* "improvement" won't accidentally break *your* research code by protecting it with tests that verify key results.
- Continuous Integration services ensure the tests always get run on every proposed change.

Good, current documentation is essential.

It convinces people that it might be easier to learn *your* thing than to write their own.

- Complete installation instructions
- Fully worked examples
- Tools for simulating data or public links to example data sets

EVENT MODEL

Minimalist and Extensible

- Every document has a unique ID and a timestamp.
- Specific domains, facilities, collaborations, research groups can overlay schemas implementing their own standards (e.g. SciData, PIF).

Run Start: Metadata about this run, including everything we know in advance: time, type of experiment, sample info., etc.

Run Stop: Additional metadata known at the end: what time it completed and its exit status (success, aborted, failed)

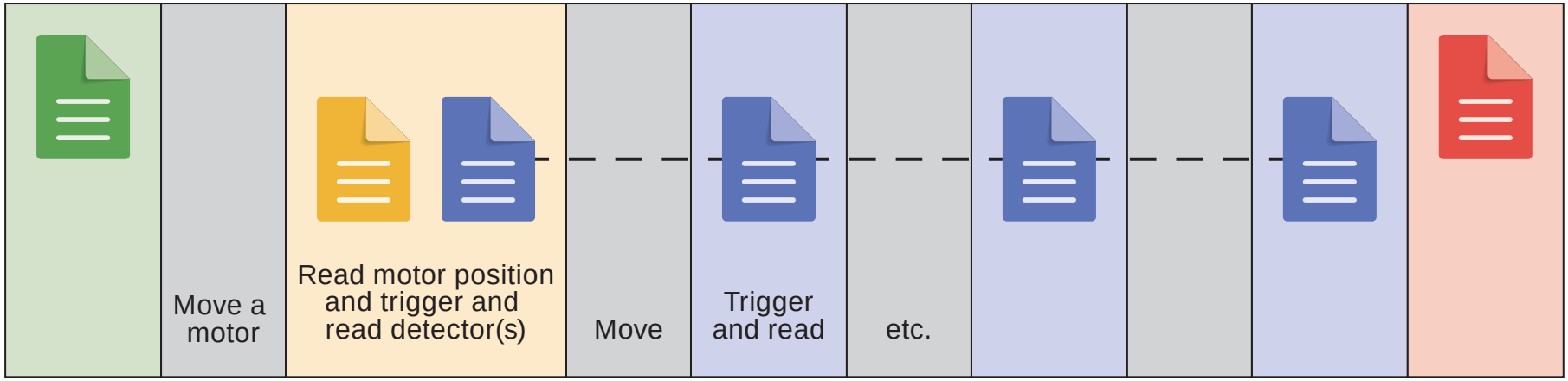
Event: Readings and timestamps

Event Descriptor: Metadata about the readings in the event (units, precision, etc.) and the relevant hardware

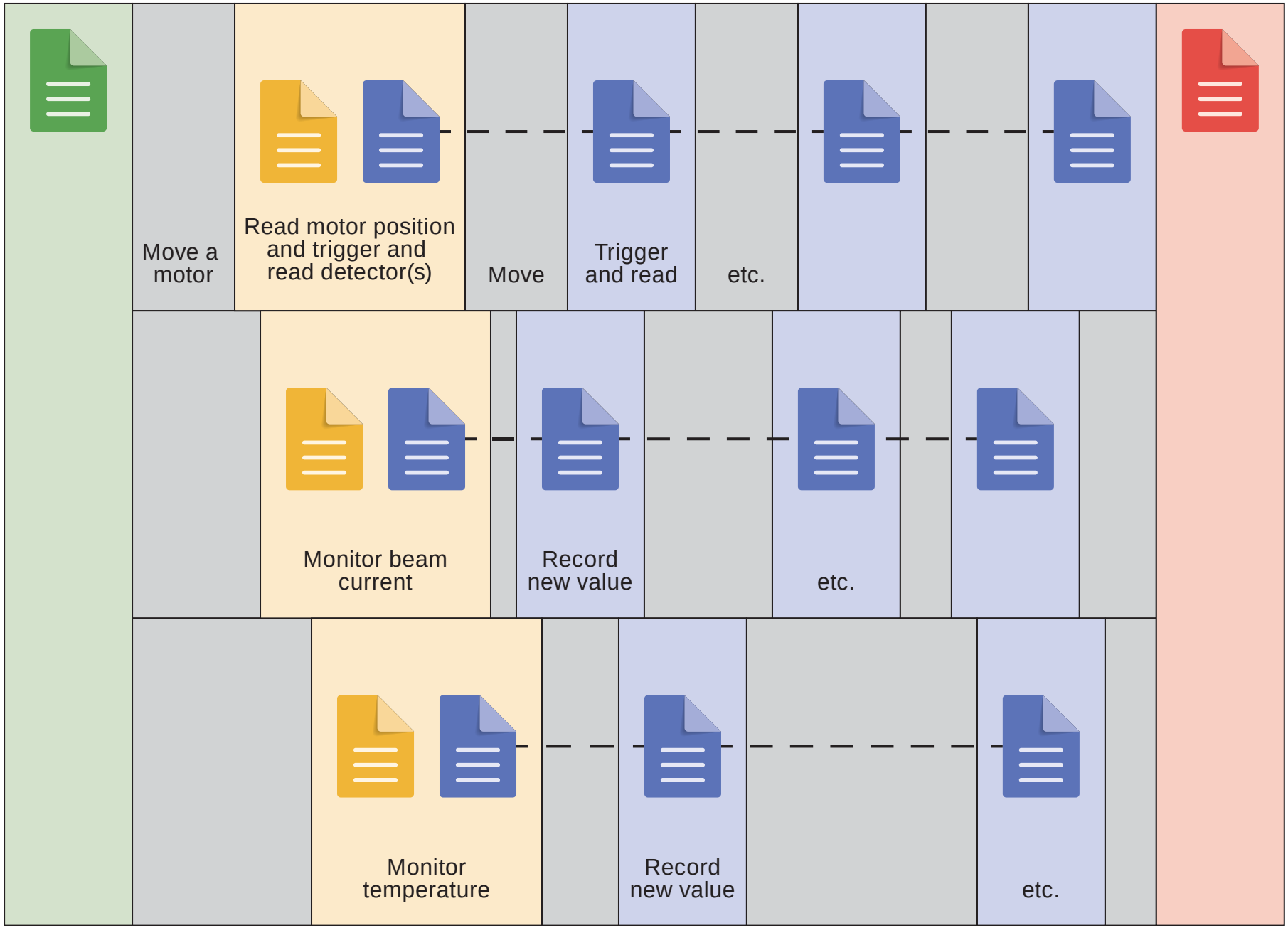
Example 1: Simplest Possible Run

Do nothing - this is the simplest possible experiment!

Example 2: A Simple Scan



Example 3: Asynchronously Monitor During a Scan



Bluesky emits documents, streamed or in batches

- Bluesky is responsible for organizing metadata and readings from hardware into valid documents.
- Sometimes the readings come one at a time and Events are emitted steadily during an experiment.
- In special applications (commonly, *fly scans*) the readings come from the hardware in bulk and Events are emitted in batch(es).

ADAPTIVE EXPERIMENTS

Feedback Paths

- prompt / real-time analysis to steer experiment
- "human-in-the-loop"
- "computer-in-the-loop"
- data quality checks

Scales of Adaptive-ness

1. below `bluesky` & `ophyd`
2. in `bluesky` plans, but without generating `event`
3. providing feedback on a per-`event` basis
4. providing feedback on a per-run / `start` basis
5. providing feedback across many runs
6. asynchronous and decoupled feedback

below `bluesky` & `ophyd`

- timescale: $\gg$ 10Hz
- very limited time budget for analysis
- very limited access to data
- tightly coupled to hardware (PID loop, FGPA)
- expensive to develop

in `bluesky` plans, but without generating `event`

- timescale: 1-10Hz
- limited time budget for analysis
- limited access to data
- logic implemented in Python in acquisition process
- coupled to hardware
- can be used for filtering

providing feedback on a per-`event` basis

- timescale: 1-5s
- modest time budget for analysis
- access to "single point" of data (& cache)
- run in or out of acquisition process

providing feedback on a per-run / `start` basis

- timescale: 5-60s
- modest time budget for analysis
- access to "full scan" data (& cache)
- run in or out of acquisition process

providing feedback across many runs

- timescale: ∞
- arbitarily compute budget
- access to all historical data
- muliti-modal

asynchronous and decoupled feedback

- Is the beam up?
- Is the shutter open?
- Is the sample still in the beam?
- Do we have enough data on this sample?
- Is the sample toast?

Docs with theory and examples:

bluesky/bluesky-adaptive

WHAT IS NEEDED FOR REMOTE DATA ACQUISITION?

Components of Remote Acquisition

- Monitoring the instrument's status
- Invoking acquisition to collect data
- Reviewing / processing / interact with just-collected data

Beamline Status

- Is the RunEngine running or idle?
- Is the shutter open?
- Is the ring up?
- Where are some key motors?

This is best handled by something that will re-publish from the control system (e.g. EPICS) using web-based OPIs, industry standard monitoring tools, or simple web pages.

Invoke Acquisition

Each level of access builds on the previous one:

1. Physical, manual access (e.g. with a wrench)
2. Direct access (read/write to PVs)
3. Device-level access (Ophyd, Tango)
4. Procedure-level access (bluesky plans)

The procedure level is the one we target for remote access.

Review Data

- Live-updating visualizations of streaming data
- Browsable catalog of saved data
- Export / download
- Interactive exploration via Jupyter

This is addressed by web portals which likely need to be facility- or instrument-specific (but we can share components).

BLUESKY RUNENGINE AS A SERVICE

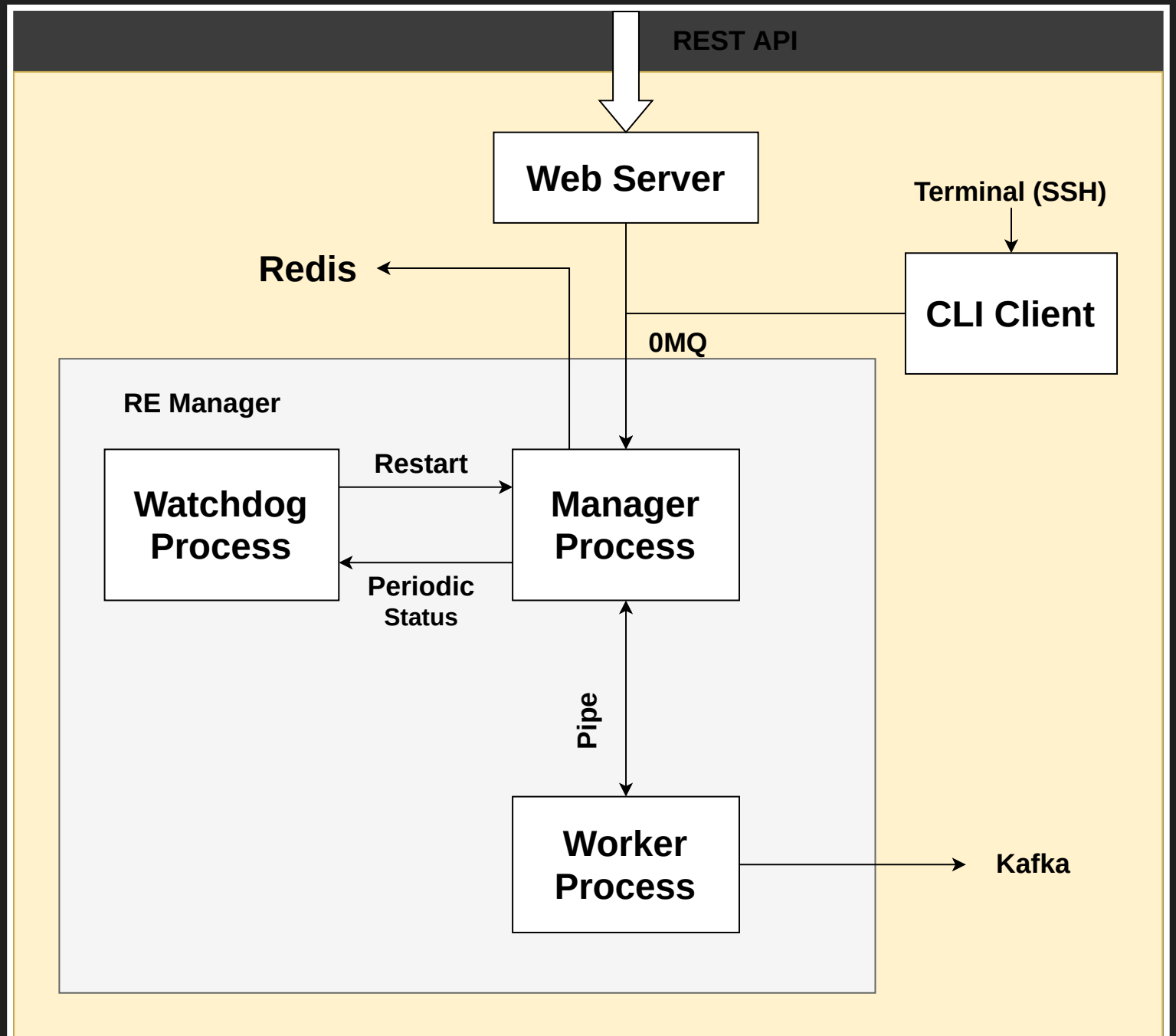
- Support remote and multi-tenant data acquisition
- Documentation: bluesky-queueserver
- Highly resourced by NSLS-II Remote Experiments Task Force
- Under rapid development (~2 large PRs / week)

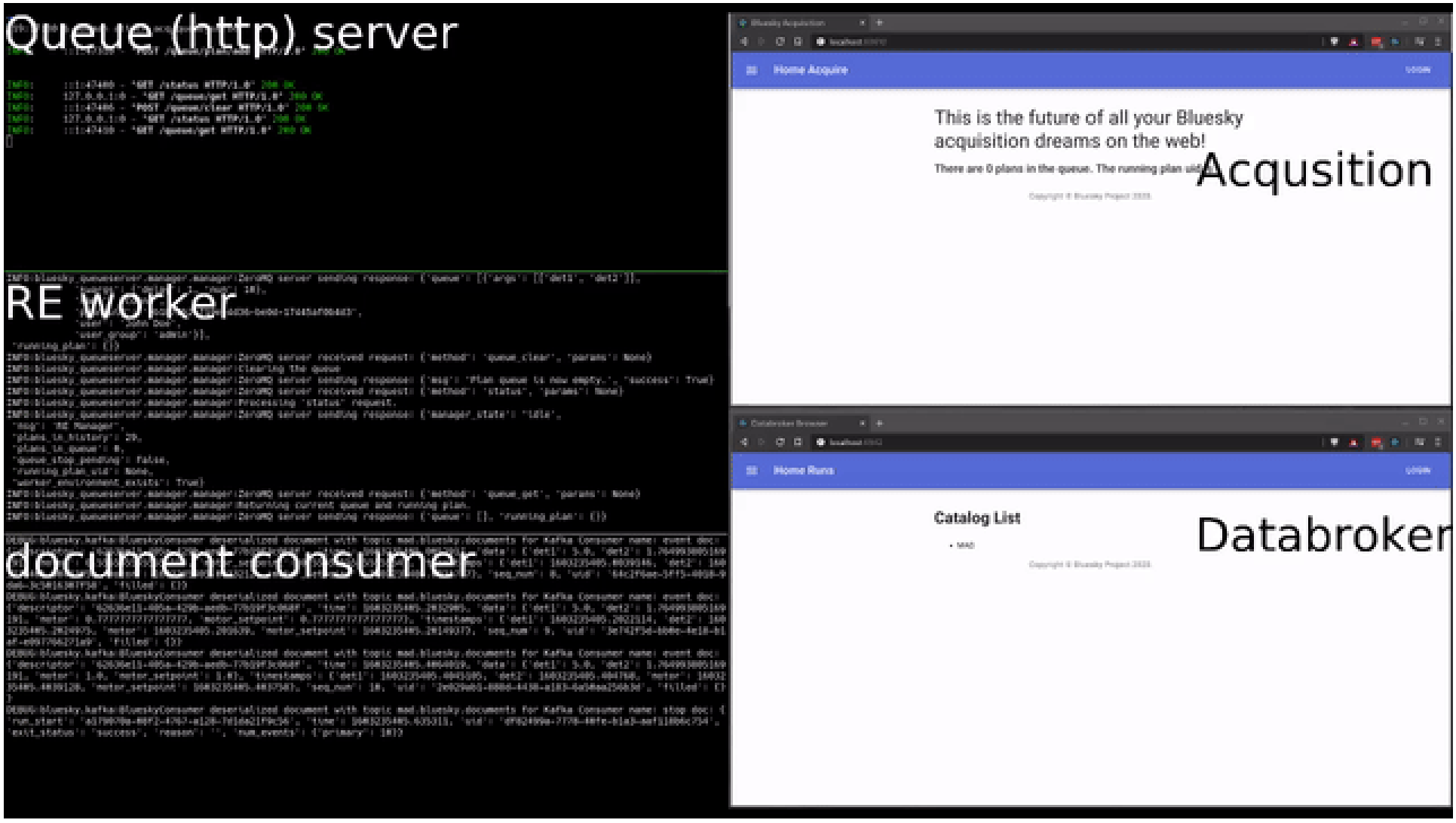
Traditional in-person, REPL-based control assumes:

- RunEngine, ophyd objects, CA connections all live in one process that user runs at local workstation
- User has exclusive control, enforced by physical presence

To enable remote operation we need to:

- Enforce exclusive control
- Run procedures without full access to the beamline
- Provide per-user profiles of what devices and procedures a user can access


REST API
Web Server
Terminal (SSH)
Redis
CLI Client
0MQ
RE Manager
Watchdog Process
Restart
Manager Process
Periodic Status
Pipe
Worker Process
Kafka

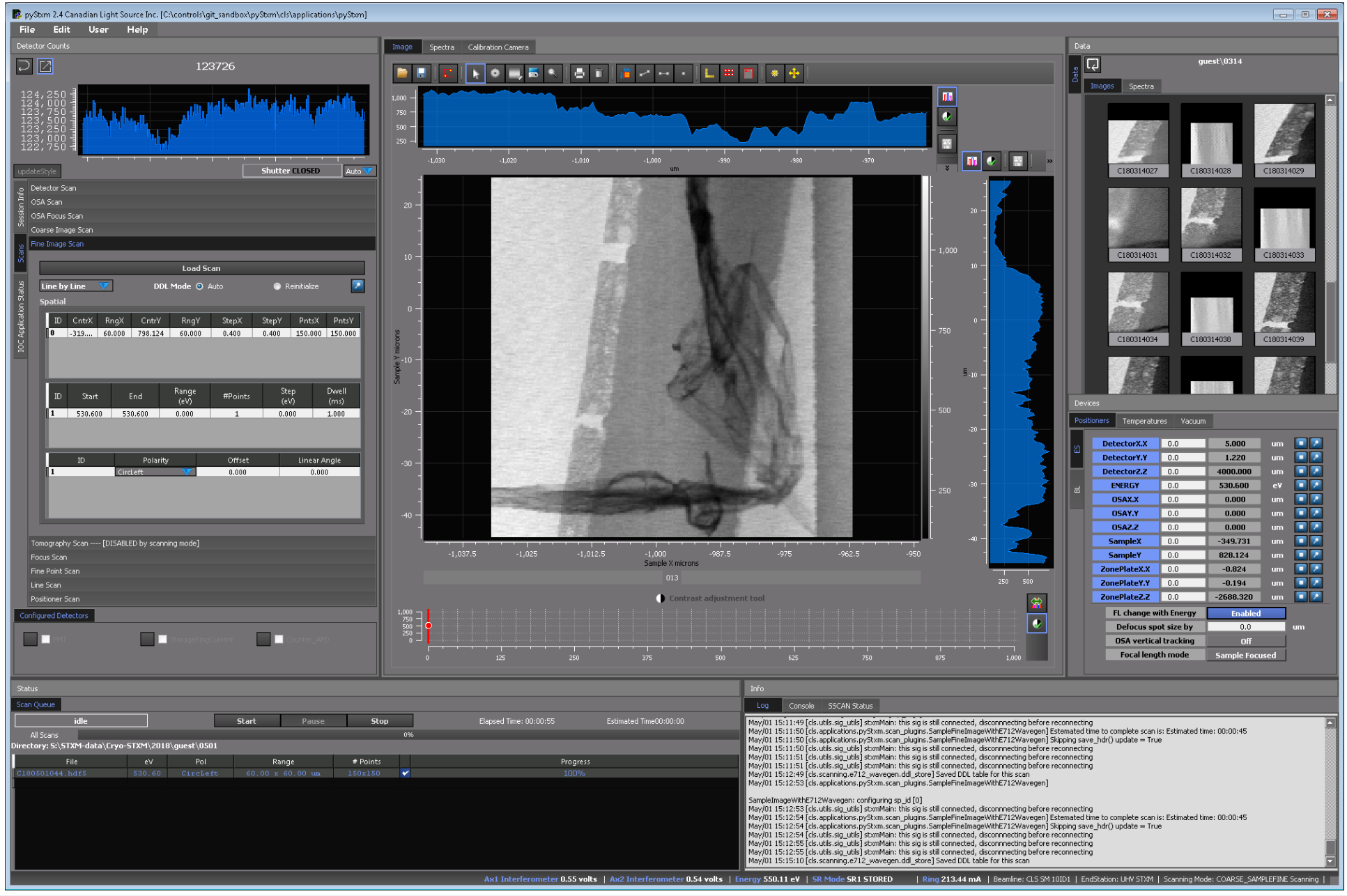

Queue (http) server
RE worker
document consumer
Acqusition
Databroker
Home Acquire
This is the future of all your Bluesky acquisition dreams on the web!
Home Runs
Catalog List

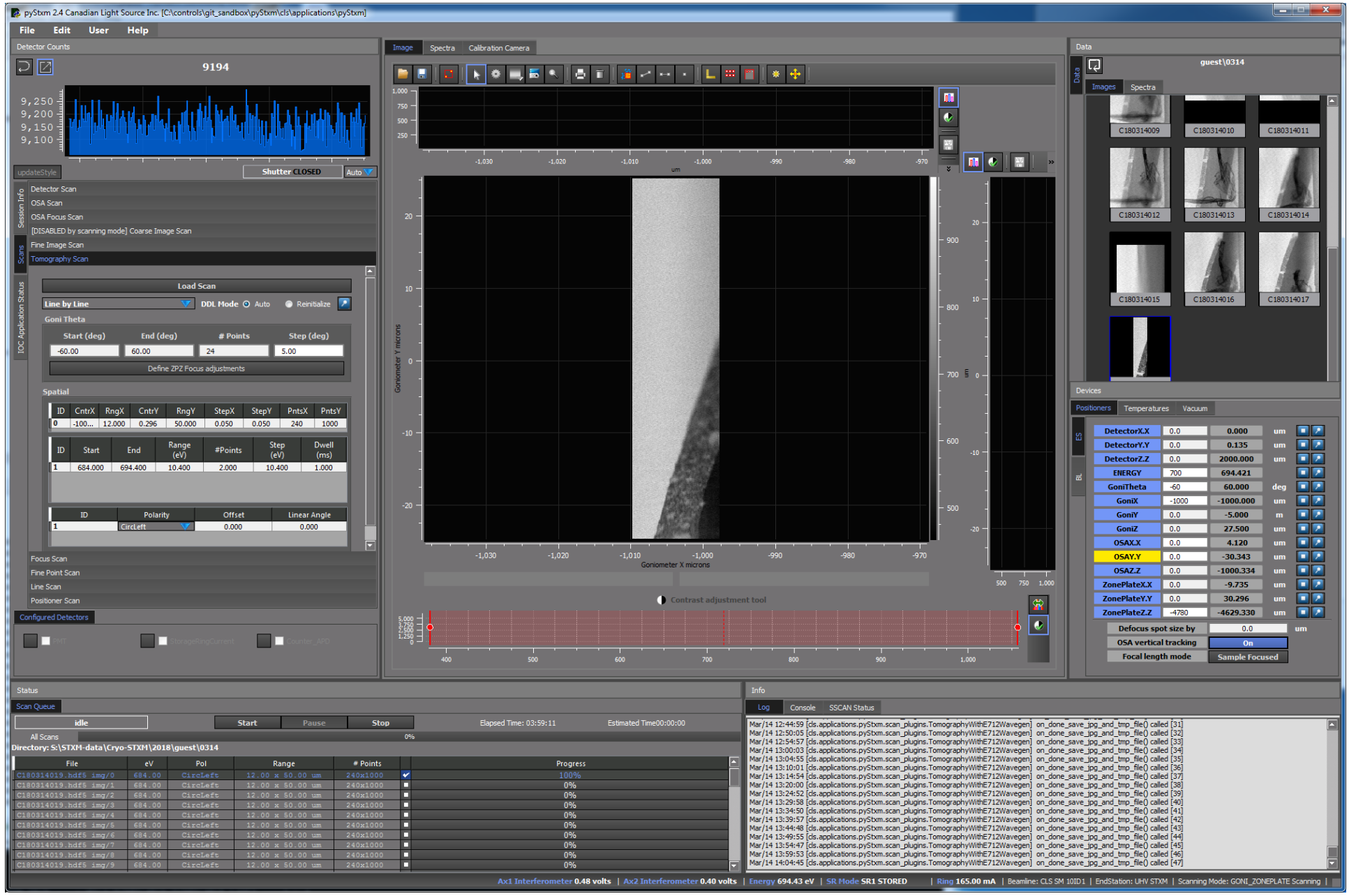
SURVEY OF COMMUNITY UI DEVELOPMENTS

Various institutions are building graphical user interfaces on Bluesky.

pyStxm at Canadian Light Source (Russ Berg)

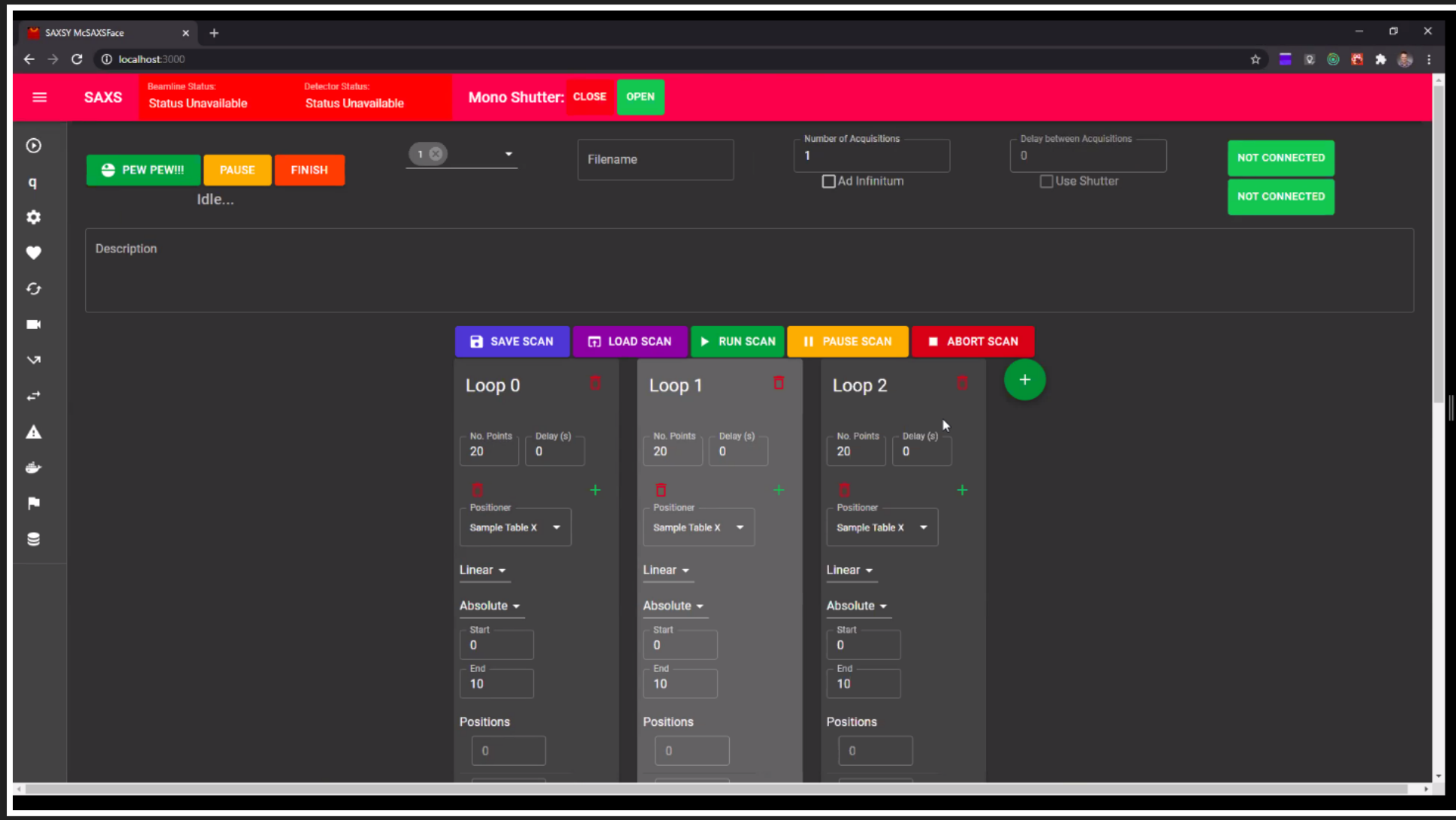
- Desktop-based (Qt)
- Formerly had custom scanning engine
- Refactored to use Bluesky
- RussBerg/pyStxm3


pySTXM 2.4 Canadian Light Source Inc. [C:\controls\git_sandbox\pySTXM\cls\applications\pySTXM]
File Edit User Help
Detector Counts
123726
Shutter CLOSED
Auto
Detector Scan
OSA Scan
OSA Focus Scan
Coarse Image Scan
Fine Image Scan
Load Scan
Line by Line
DDL Mode Auto Reinitialize
Spatial
Tomography Scan ---- [DISABLED by scanning mode]
Focus Scan
Fine Point Scan
Line Scan
Positioner Scan
Configured Detectors
Image Spectra Calibration Camera
Sample X microns
Sample Y microns
Contrast adjustment tool
Data
guest\0314
Images Spectra
Devices
Positioners Temperatures Vacuum
DetectorX.X 5.000 um
DetectorY.Y 1.220 um
DetectorZ.Z 4000.000 um
ENERGY 530.600 eV
OSAX.X 0.000 um
OSAY.Y 0.000 um
OSAZ.Z 0.000 um
SampleX -349.731 um
SampleY 828.124 um
ZonePlateX.X -0.824 um
ZonePlateY.Y -0.194 um
ZonePlateZ.Z -2688.320 um
FL change with Energy Enabled
Defocus spot size by 0.0 um
OSA vertical tracking Off
Focal length mode Sample Focused
Status
Scan Queue
idle
Start Pause Stop
Elapsed Time: 00:00:55
Estimated Time00:00:00
Directory: S:\STXM-data\Cryo-STXM\2018\guest\0501
Info
Log Console SSCAN Status
Ax1 Interferometer 0.55 volts | Ax2 Interferometer 0.54 volts | Energy 550.11 eV | SR Mode SR1 STORED | Ring 213.44 mA | Beamline: CLS SM 10ID1 | EndStation: UHV STXM | Scanning Mode: COARSE_SAMPLEFINE Scanning


pySTXM 2.4 Canadian Light Source Inc. [C:\controls\git_sandbox\pySTXM\cls\applications\pySTXM]
File Edit User Help
Detector Counts
9194
9,250
9,200
9,150
9,100
updateStyle
Shutter CLOSED
Auto
Session Info
Scans
IOC Application Status
Detector Scan
OSA Scan
OSA Focus Scan
[DISABLED by scanning mode] Coarse Image Scan
Fine Image Scan
Tomography Scan
Load Scan
Line by Line
DDL Mode Auto Reinitialize
Goni Theta
Start (deg) End (deg) # Points Step (deg)
-60.00 60.00 24 5.00
Define ZPZ Focus adjustments
Spatial
ID CntrX RngX CntrY RngY StepX StepY PntsX PntsY
0 -100... 12.000 0.296 50.000 0.050 0.050 240 1000
ID Start End Range (eV) #Points Step (eV) Dwell (ms)
1 684.000 694.400 10.400 2.000 10.400 1.000
ID Polarity Offset Linear Angle
1 CircLeft 0.000 0.000
Focus Scan
Fine Point Scan
Line Scan
Positioner Scan
Configured Detectors
PMT
StorageRingCurrent
Counter_APD
Image Spectra Calibration Camera
Goniometer Y microns
Goniometer X microns
um
Contrast adjustment tool
Data
guest\0314
Images Spectra
C180314009 C180314010 C180314011
C180314012 C180314013 C180314014
C180314015 C180314016 C180314017
Devices
Positioners Temperatures Vacuum
DetectorX.X 0.0 0.000 um
DetectorY.Y 0.0 0.135 um
DetectorZ.Z 0.0 2000.000 um
ENERGY 700 694.421
GoniTheta -60 60.000 deg
GoniX -1000 -1000.000 um
GoniY 0.0 -5.000 m
GoniZ 0.0 27.500 um
OSAX.X 0.0 4.120 um
OSAY.Y 0.0 -30.343 um
OSAZ.Z 0.0 -1000.334 um
ZonePlateX.X 0.0 -9.735 um
ZonePlateY.Y 0.0 30.296 um
ZonePlateZ.Z -4780 -4629.330 um
Defocus spot size by 0.0 um
OSA vertical tracking On
Focal length mode Sample Focused
Status
Scan Queue
idle
Start Pause Stop
Elapsed Time: 03:59:11
Estimated Time00:00:00
All Scans 0%
Directory: S:\STXM-data\Cryo-STXM\2018\guest\0314
File eV Pol Range # Points Progress
C180314019.hdf5 img/0 684.00 CircLeft 12.00 x 50.00 um 240x1000 100%
C180314019.hdf5 img/1 684.00 CircLeft 12.00 x 50.00 um 240x1000 0%
C180314019.hdf5 img/2 684.00 CircLeft 12.00 x 50.00 um 240x1000 0%
C180314019.hdf5 img/3 684.00 CircLeft 12.00 x 50.00 um 240x1000 0%
C180314019.hdf5 img/4 684.00 CircLeft 12.00 x 50.00 um 240x1000 0%
C180314019.hdf5 img/5 684.00 CircLeft 12.00 x 50.00 um 240x1000 0%
C180314019.hdf5 img/6 684.00 CircLeft 12.00 x 50.00 um 240x1000 0%
C180314019.hdf5 img/7 684.00 CircLeft 12.00 x 50.00 um 240x1000 0%
C180314019.hdf5 img/8 684.00 CircLeft 12.00 x 50.00 um 240x1000 0%
C180314019.hdf5 img/9 684.00 CircLeft 12.00 x 50.00 um 240x1000 0%
Info
Log Console SSCAN Status
Mar/14 12:44:59 [cls.applications.pySTXM.scan_plugins.TomographyWithE712Wavegen] on_done_save_jpg_and_tmp_file() called [31]
Mar/14 12:50:05 [cls.applications.pySTXM.scan_plugins.TomographyWithE712Wavegen] on_done_save_jpg_and_tmp_file() called [32]
Mar/14 12:54:57 [cls.applications.pySTXM.scan_plugins.TomographyWithE712Wavegen] on_done_save_jpg_and_tmp_file() called [33]
Mar/14 13:00:03 [cls.applications.pySTXM.scan_plugins.TomographyWithE712Wavegen] on_done_save_jpg_and_tmp_file() called [34]
Mar/14 13:04:55 [cls.applications.pySTXM.scan_plugins.TomographyWithE712Wavegen] on_done_save_jpg_and_tmp_file() called [35]
Mar/14 13:10:01 [cls.applications.pySTXM.scan_plugins.TomographyWithE712Wavegen] on_done_save_jpg_and_tmp_file() called [36]
Mar/14 13:14:54 [cls.applications.pySTXM.scan_plugins.TomographyWithE712Wavegen] on_done_save_jpg_and_tmp_file() called [37]
Mar/14 13:20:00 [cls.applications.pySTXM.scan_plugins.TomographyWithE712Wavegen] on_done_save_jpg_and_tmp_file() called [38]
Mar/14 13:24:52 [cls.applications.pySTXM.scan_plugins.TomographyWithE712Wavegen] on_done_save_jpg_and_tmp_file() called [39]
Mar/14 13:29:58 [cls.applications.pySTXM.scan_plugins.TomographyWithE712Wavegen] on_done_save_jpg_and_tmp_file() called [40]
Mar/14 13:34:50 [cls.applications.pySTXM.scan_plugins.TomographyWithE712Wavegen] on_done_save_jpg_and_tmp_file() called [41]
Mar/14 13:39:57 [cls.applications.pySTXM.scan_plugins.TomographyWithE712Wavegen] on_done_save_jpg_and_tmp_file() called [42]
Mar/14 13:44:48 [cls.applications.pySTXM.scan_plugins.TomographyWithE712Wavegen] on_done_save_jpg_and_tmp_file() called [43]
Mar/14 13:49:55 [cls.applications.pySTXM.scan_plugins.TomographyWithE712Wavegen] on_done_save_jpg_and_tmp_file() called [44]
Mar/14 13:54:47 [cls.applications.pySTXM.scan_plugins.TomographyWithE712Wavegen] on_done_save_jpg_and_tmp_file() called [45]
Mar/14 13:59:53 [cls.applications.pySTXM.scan_plugins.TomographyWithE712Wavegen] on_done_save_jpg_and_tmp_file() called [46]
Mar/14 14:04:45 [cls.applications.pySTXM.scan_plugins.TomographyWithE712Wavegen] on_done_save_jpg_and_tmp_file() called [47]
Ax1 Interferometer 0.48 volts | Ax2 Interferometer 0.40 volts | Energy 694.43 eV | SR Mode SR1 STORED | Ring 165.00 mA | Beamline: CLS SM 10ID1 | EndStation: UHV STXM | Scanning Mode: GONI_ZONEPLATE Scanning

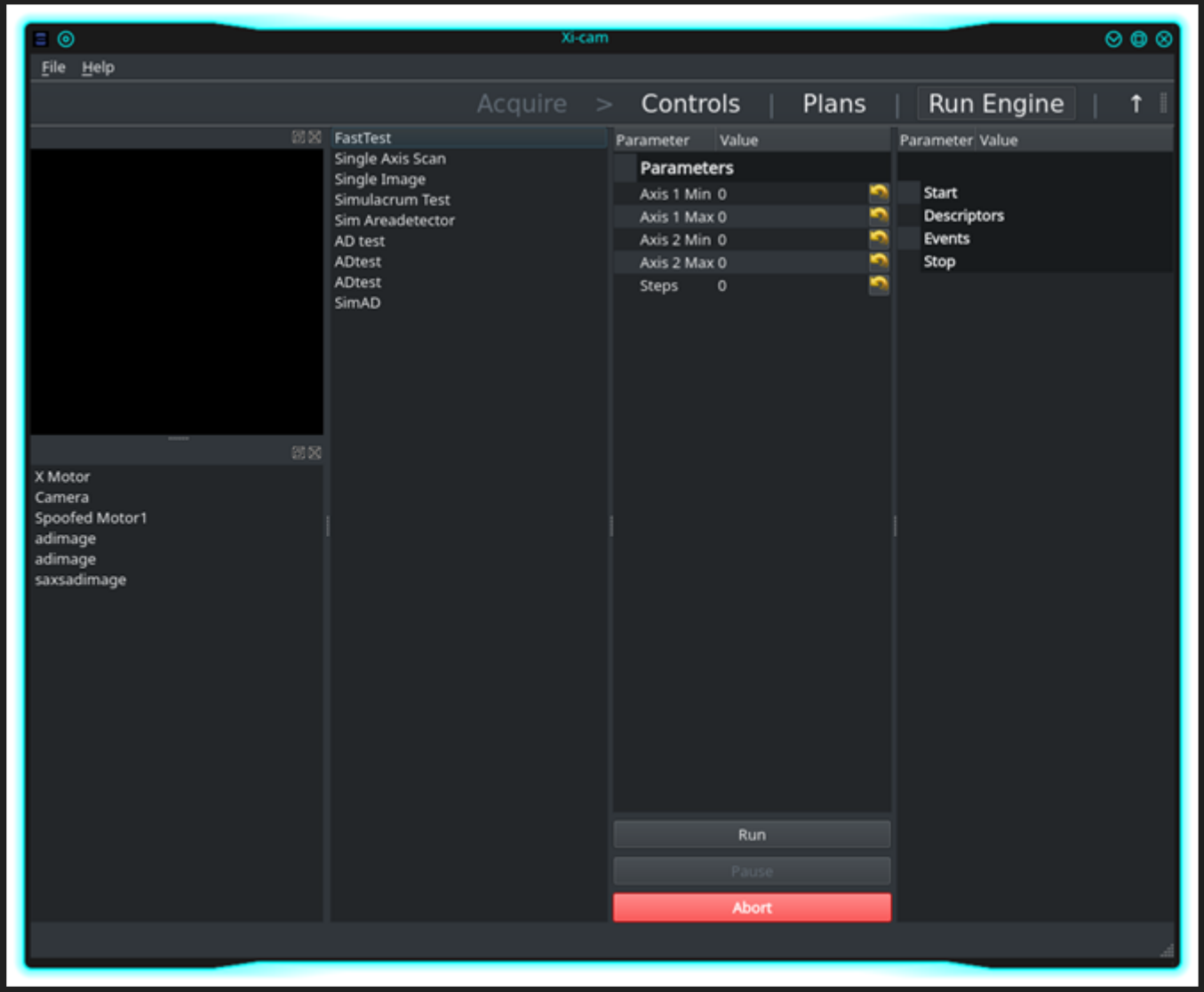
GUI for SAXS at Australian Synchrotron (Stephen Mudie)

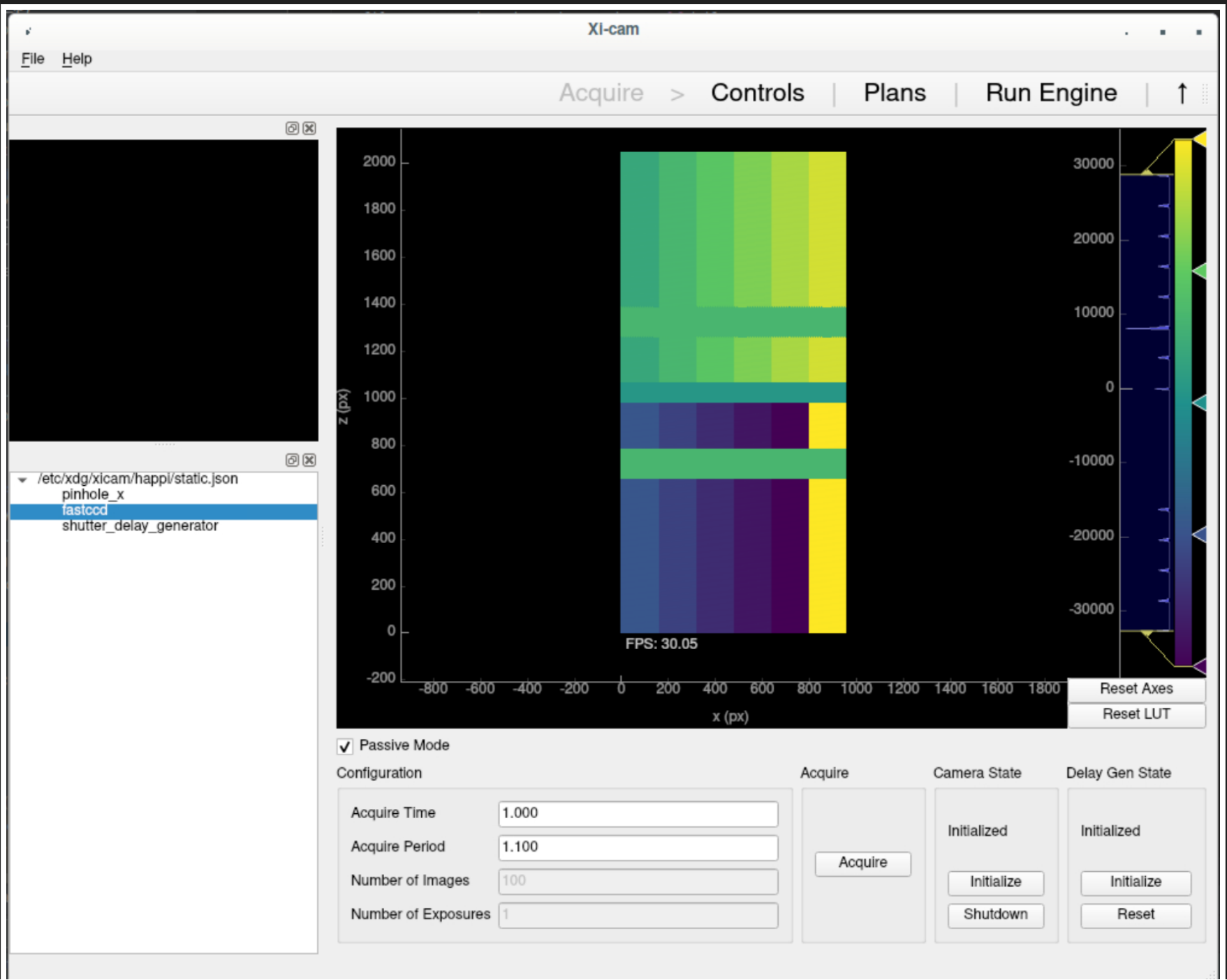
- Web-based (React)
- Code partially available at AustralianSynchrotron/saxs_beamline_library_react
- Plans to be fully open in a week or so


SAXSY McSAXSFace
localhost:3000
SAXS
Beamline Status:
Status Unavailable
Detector Status:
Status Unavailable
Mono Shutter:
CLOSE
OPEN
PEW PEW!!!
PAUSE
FINISH
Idle...
1
Filename
Number of Acquisitions
1
Ad Infinitum
Delay between Acquisitions
0
Use Shutter
NOT CONNECTED
NOT CONNECTED
Description
SAVE SCAN
LOAD SCAN
RUN SCAN
PAUSE SCAN
ABORT SCAN
Loop 0
Loop 1
Loop 2
No. Points
20
Delay (s)
0
Positioner
Sample Table X
Linear
Absolute
Start
0
End
10
Positions
0

GUI for COSMIC at Advanced Light Source (Xi-CAM Team)

- Desktop-based
- Plugin to the Xi-CAM framework (Qt)
- Xi-CAM/Xi-cam.Acquire


Xi-cam
File
Help
Acquire
>
Controls
Plans
Run Engine
FastTest
Single Axis Scan
Single Image
Simulacrum Test
Sim Areadetector
AD test
ADtest
ADtest
SimAD
Parameter
Value
Parameters
Axis 1 Min 0
Axis 1 Max 0
Axis 2 Min 0
Axis 2 Max 0
Steps 0
Parameter
Value
Start
Descriptors
Events
Stop
X Motor
Camera
Spoofed Motor1
adimage
adimage
saxsadimage
Run
Pause
Abort


XI-cam
File
Help
Acquire > Controls | Plans | Run Engine
/etc/xdg/xicam/happi/static.json
pinhole_x
fastccd
shutter_delay_generator
FPS: 30.05
z (px)
x (px)
Reset Axes
Reset LUT
Passive Mode
Configuration
Acquire Time
1.000
Acquire Period
1.100
Number of Images
100
Number of Exposures
1
Acquire
Acquire
Camera State
Initialized
Initialize
Shutdown
Delay Gen State
Initialized
Initialize
Reset

Finally, various one-off solutions developed by beamline and/or Controls staff at NSLS-II

- "X-Live" (NSLS-II ISS & QAS)
- "xpdacq" (NSLS-II XPD & PDF)
- "XFP High-Throughput Multi-Sample Holder" (NSLS-II XFP)
- "BS-Studio" (NSLS-II ESM)

We intend to guide a systematic refactor of these onto components from bluesky-queueserver and bluesky-widgets.

BLUESKY WIDGETS

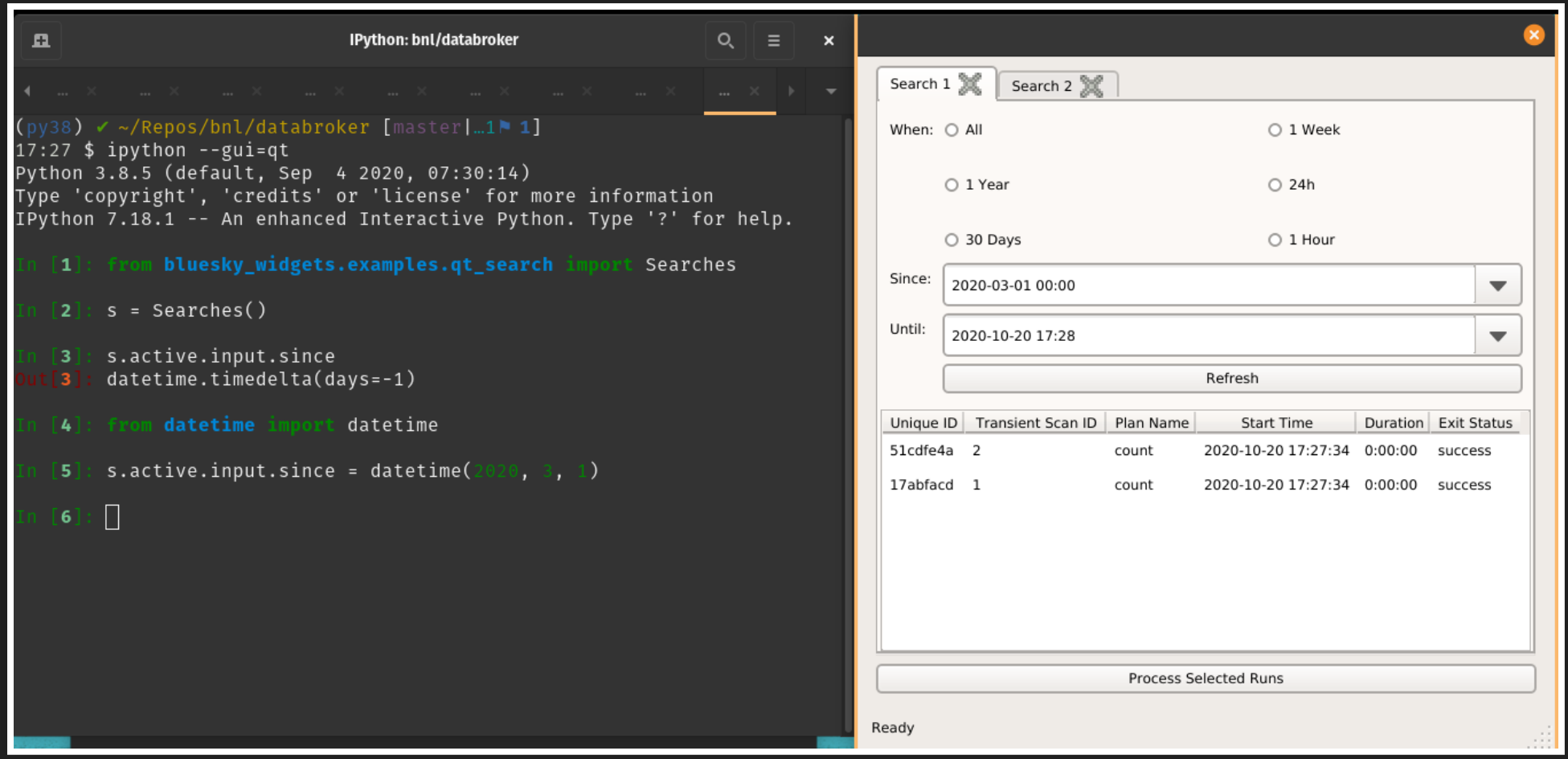
A new project aimed at sharing GUI components built on Bluesky interfaces

bluesky/bluesky-widgets

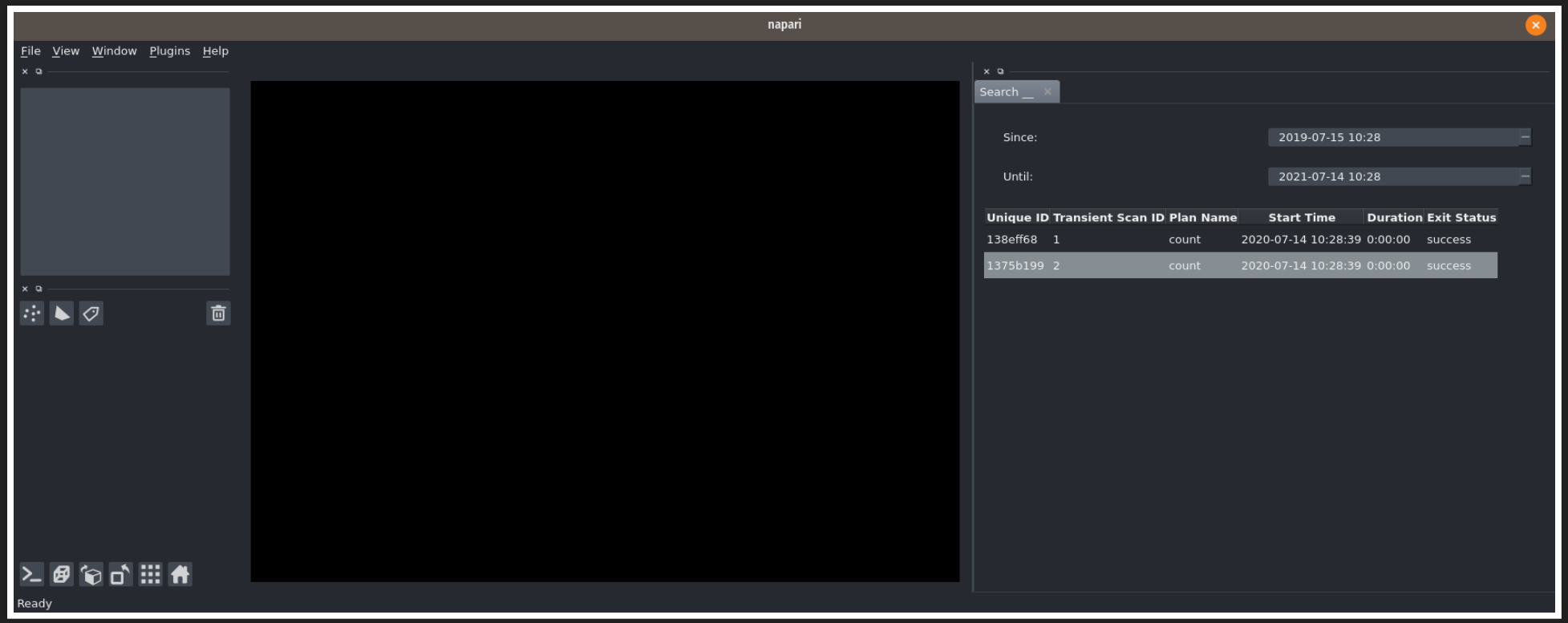
Goals

- A component library, not an extensible application
- Ships runnable examples, but instruments should build their own
- Integrate with existing applications (napari, PyFAI, Xi-CAM, ...)
- All actions can be performed from a terminal or run headless
- Model is framework-agnostic. Front-ends will include Qt, Jupyter.

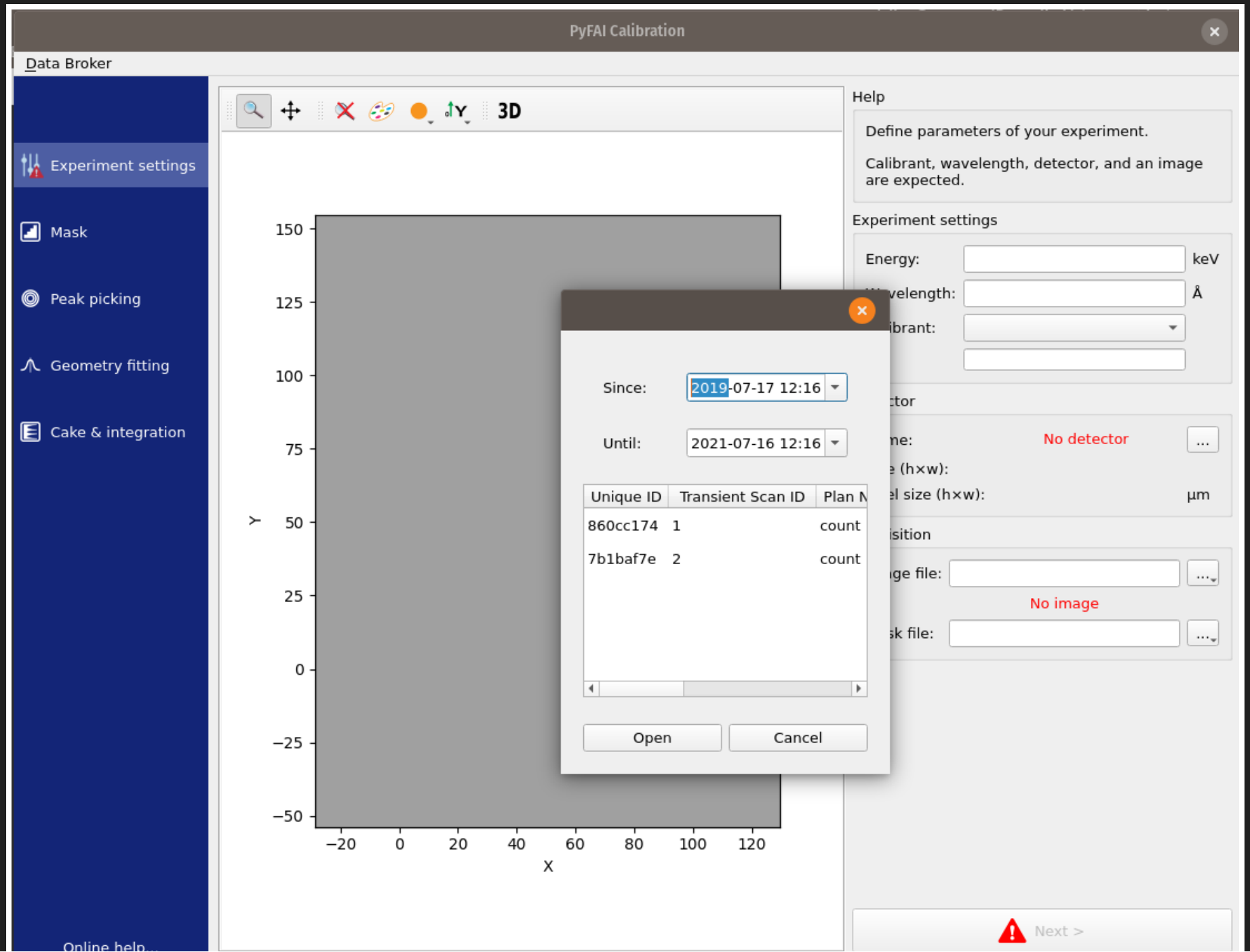
Examples of integrating Data Broker search into existing software...

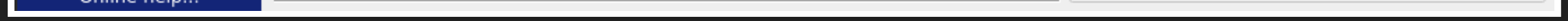


Model can be manipulated from IPython terminal



Search Data Broker from napari (N-dimensional image viewer)

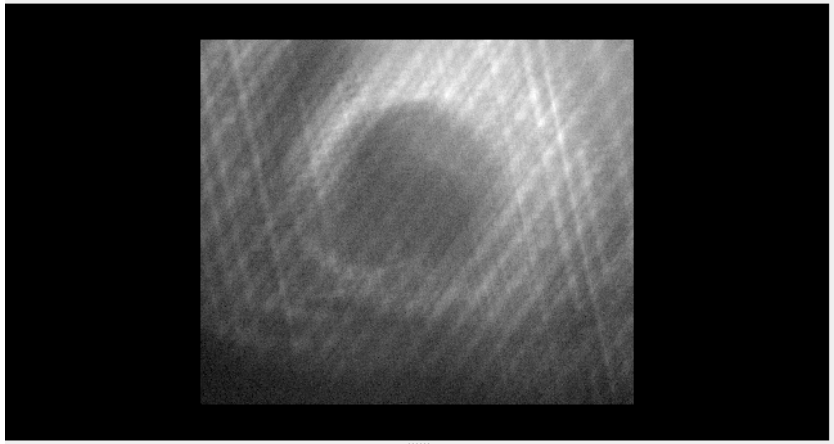

PyFAI Calibration
Data Broker
Experiment settings
Mask
Peak picking
Geometry fitting
Cake & integration
Online help...
3D
150
125
100
75
50
25
0
−25
−50
Y
−20
0
20
40
60
80
100
120
X
Since:
2019-07-17 12:16
Until:
2021-07-16 12:16
Unique ID
Transient Scan ID
Plan N
860cc174 1 count
7b1baf7e 2 count
Open
Cancel
Help
Define parameters of your experiment.
Calibrant, wavelength, detector, and an image are expected.
Experiment settings
Energy:
keV
Å
No detector
No image
Next >



Search Data Broker from PyFAI (powder diffraction software)

Xi-cam

File Help



Local | Databroker

Back

sutherland_42_10

Since: 2019-07-17 12:09

Until: 2021-07-16 12:09

Unique ID	Transient Scan ID	Plan Name	Start Time	Duration	Exit Status
bc35550f	47584	user_fly_only	2019-12-12 03:39:33	0:00:21	success

Open

CAMERA Applied Math

Welcome to Xi-cam

Please cite Xi-cam in published work:

Pandolfi, R. J., et al. (2018). J. Synchrotron Rad. **25**, 1261-1270.

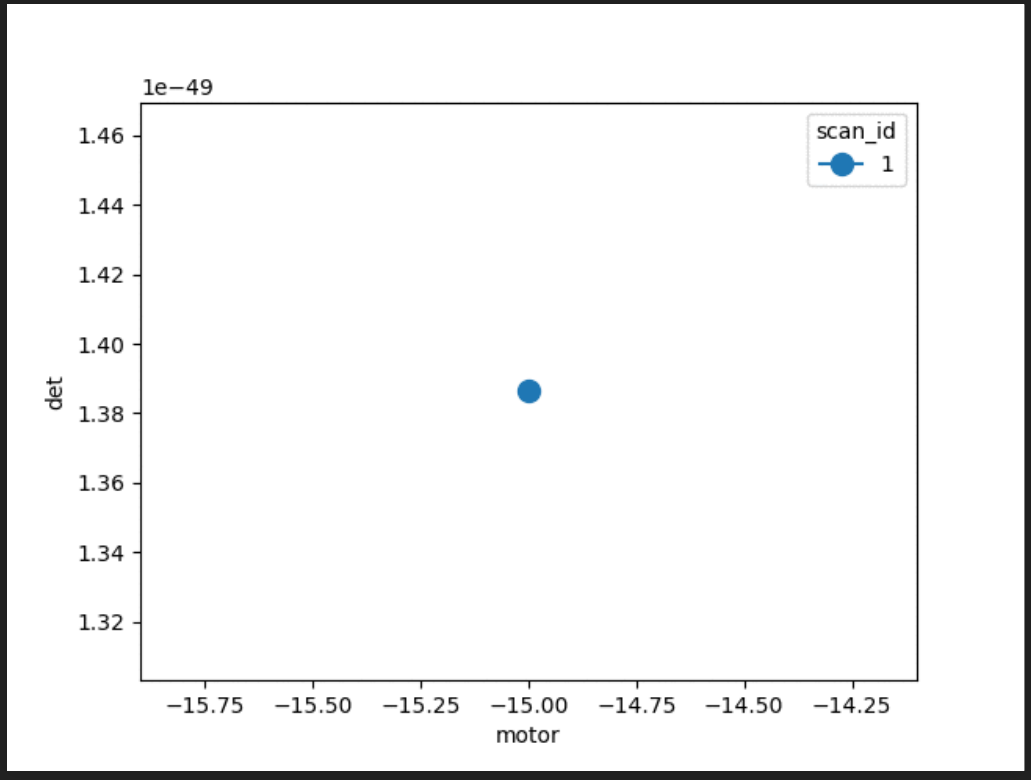
Ronald J. Pandolfi[a]
Daniel B. Allan[e]
Elke Arenholz[a]
Luis Barroso-Luque[a]
Stuart I. Campbell[e]
Thomas A. Caswell[e]
Austin Blair[a]
Francesco De Carlo[c]
Sean Fackler[a]
Amanda P. Fournier[b]
Guillaume Freychet[a]
Masafumi Fukuto[e]
Dŏga Gürsoy[ch]
Zhang Jiang[c]
Harinarayan Krishnan[a]
Dinesh Kumar[a]
R. Joseph Kline[g]
Ruipeng Li[e]
Christopher Liman[g]
Stefano Marchesini[a]
Apurva Mehta[b]
Alpha T. N'Diaye[a]
Dilworth (Dula) Y. Parkinson[a]
Holden Parks[a]
Lenson A. Pellouchoud[a]
Talita Perciano[a]
Fang Ren[b]
Shreya Sahoo[a]
Joseph Strzalka[c]
Daniel Sunday[g]
Christopher J. Tassone[a]
Daniela Ushizima[a]
Singanallur Venkatakrishnan[d]
Kevin G. Yager[f]
James A. Sethian[a]
Alexander Hexemer[a]

[a]Lawrence Berkeley National Laboratory
[b]Stanford Synchrotron Radiation Lightsource
[c]Advanced Photon Source, Argonne National Laboratory
[d]Oak Ridge National Laboratory
[e]National Synchrotron Light Source II
[f]Center for Functional Nanomaterials
[g]National Institute of Standards and Technology
[h]Department of Electrical Engineering and Computer Science, Northwestern University

Search Data Broker from Xi-CAM

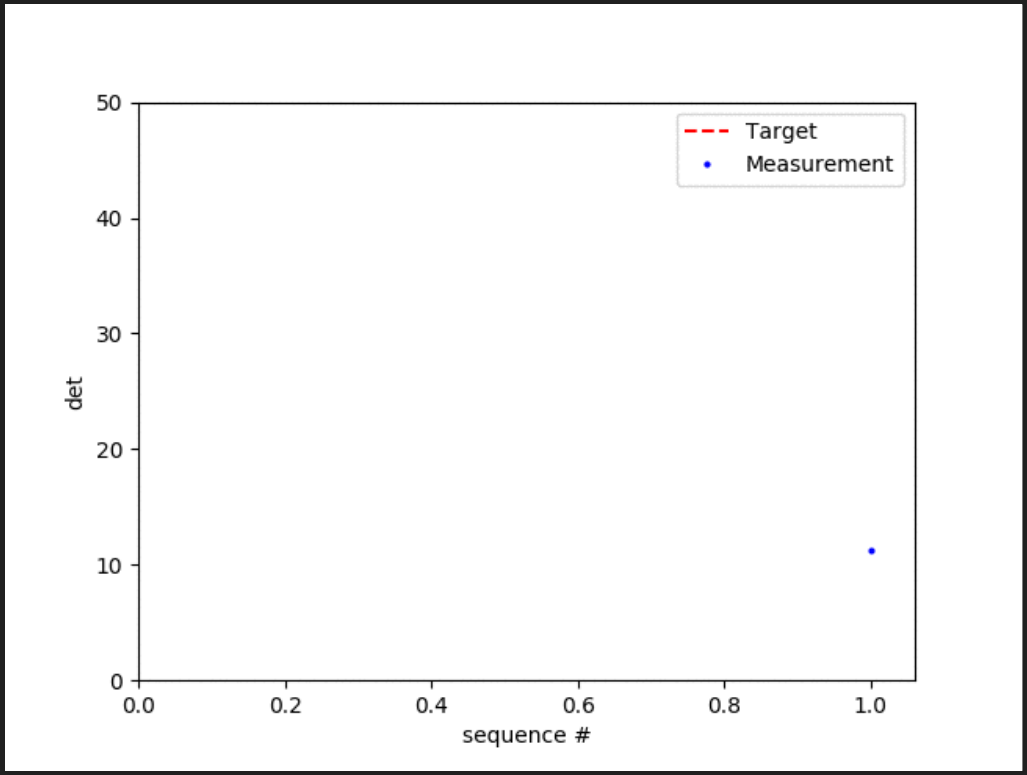
PAYOFF:
EASY AND ROBUST INTEGRATION WITH EXISTING SOFTWARE

Proof of concept:
In this scan, each step is determined adaptively in response to local slope.

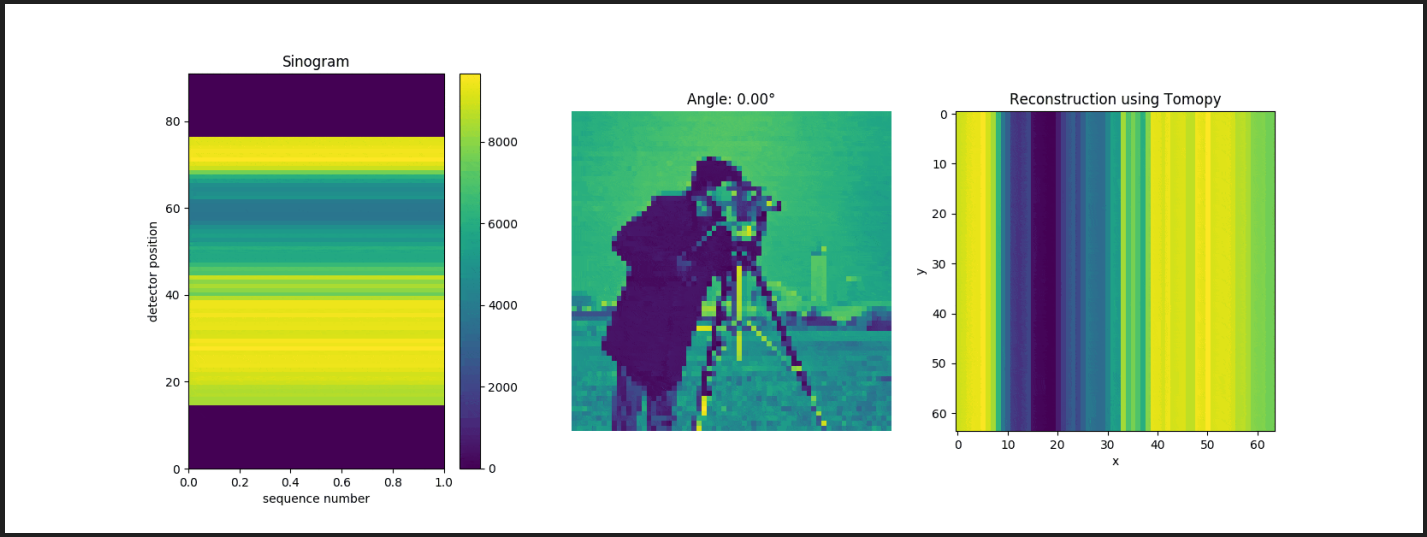


The system is designed to make fast feedback easy to write.

LCLS's Skywalker project builds on this to automatically deliver the photon beam to a number of experimental hutches at LCLS.

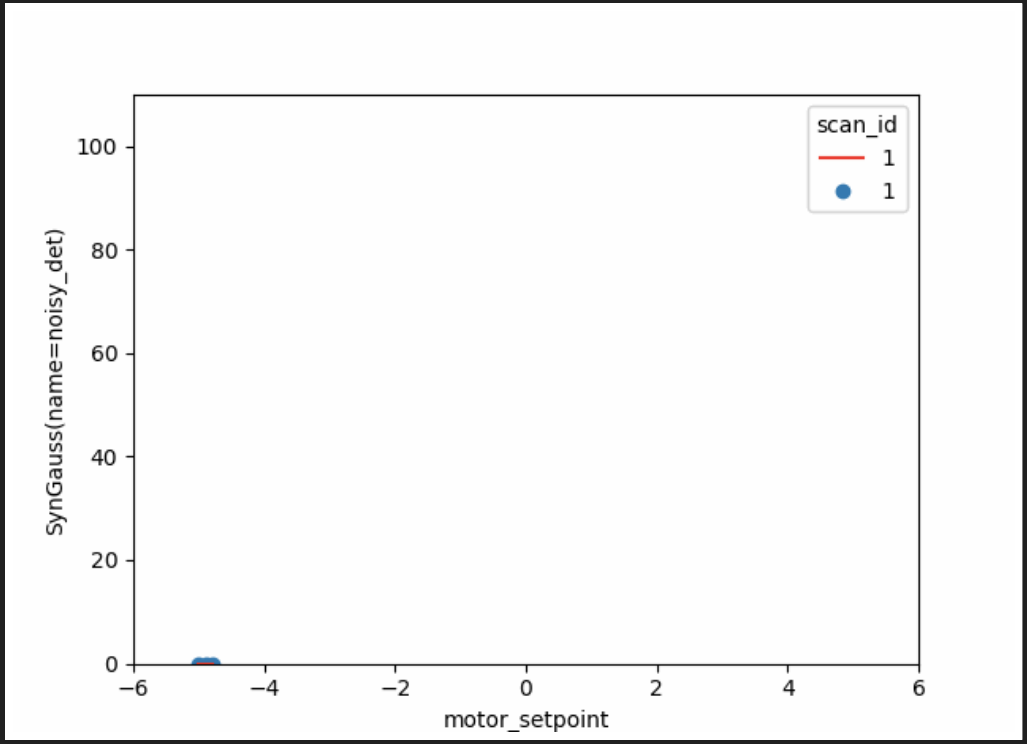


A stream of images from a linear detector is reconstructed into a volume using *tomopy* (APS).

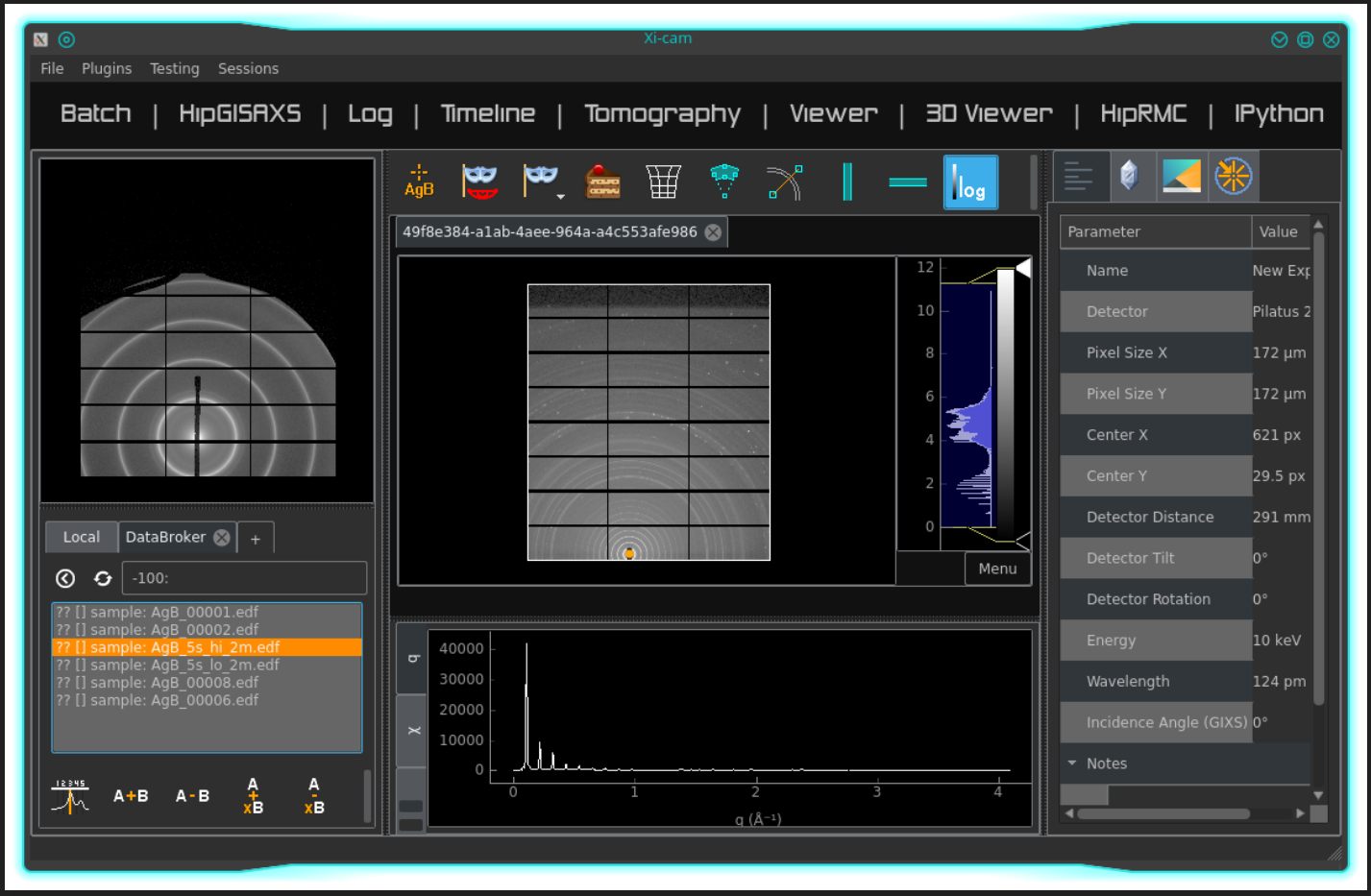


It took one TomoPy developer and one Bluesky developer less than 20 minutes to write this.

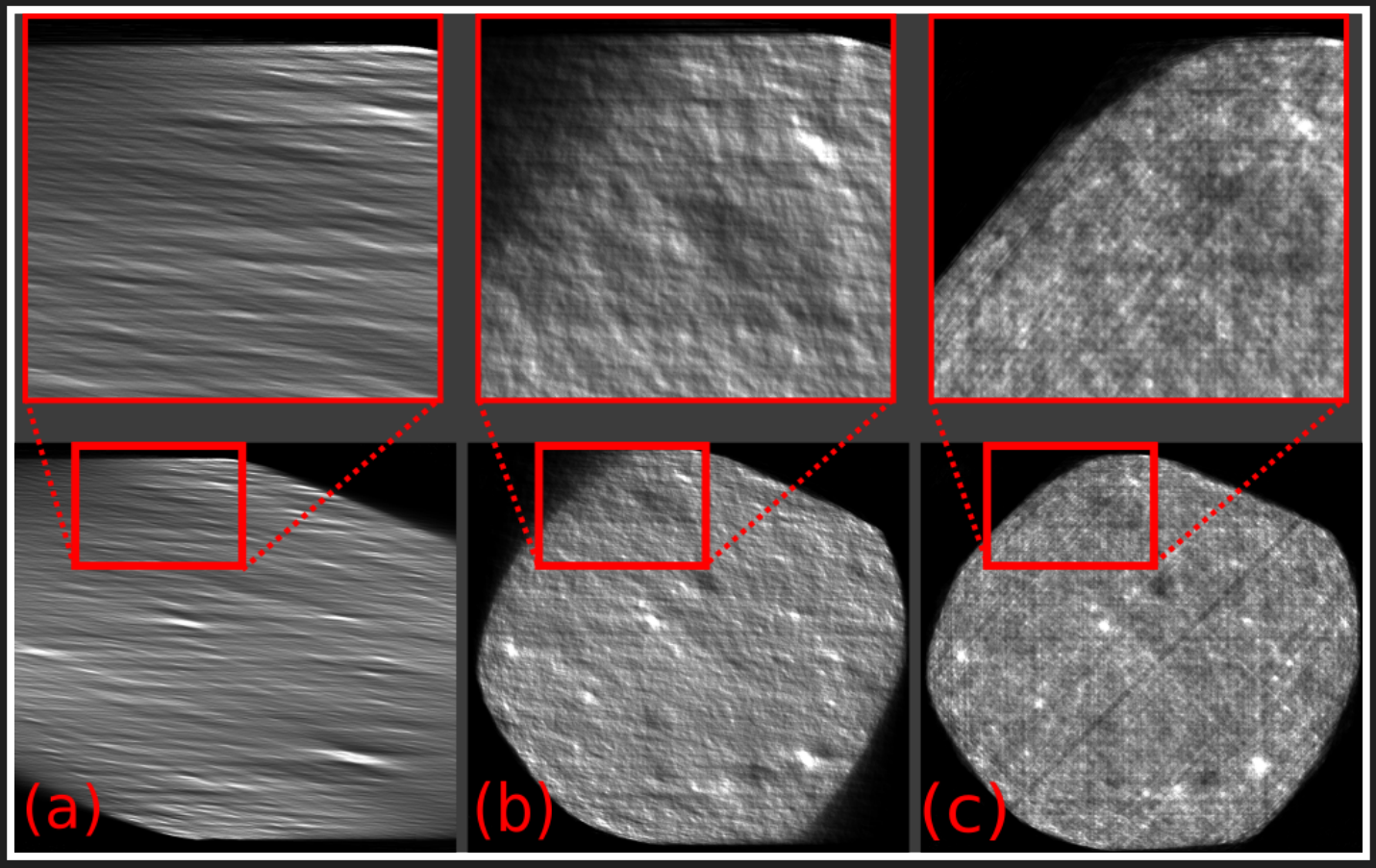
A Gaussian is fit to a stream of measured data using the Python library *lmfit* (from U. Chicago / APS).



The Xi-cam 2 GUI / plugin framework from CAMERA has adopted Bluesky's Event Model for its internal data structures.



Real-time Data Analysis at APS



Data is streamed from APS to Argonne Leadership Compute Facility. Results are immediately visualized at APS.

LINKS

- Home Page and Documentation: blueskyproject.io
- Code and Arguments about Code: github.com/bluesky
- Live, Public Demo Deployment (using Jupyter): try.nsls2.bnl.gov
- These Slides: blueskyproject.io/bluesky-slides