bluesky

The Bluesky Project Contributors

Bluesky originated at National Synchrontron 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

MOTIVATION

Build software that enables collaboration and specialization

- Each beamline is one of a kind by design, so no software can be a complete solution for everyone.
- Aim to enable beamlines to share yet also support their unique needs.
- Use software design patterns that encourage building on a shared core.

Bluesky is designed in service to data analysis

When analyzing data we want....

- To easily **find** the data we're looking for.
- **Access** to that data, not particularly caring where it's stored or in what file format.
- **Well-structured data marked up with relevant context**, to support easy and sometimes automated batch analysis.
- Seamless **integration** with popular data analysis tools.

Bluesky may be used from IPython or from graphical user interfaces

- At first we targeted command-line usage in **IPython**, thinking of users coming from SPEC.
- Staff at our facility and at other facilities have built their own graphical applications that interface to parts of Bluesky. (See following slides.)
- We are developing a **toolbox of reusable components for building graphical interfaces** for data acquisition, search, access, and visualization, supporting:
  - Desktop applications
  - Jupyter
  - Web applications

Bluesky is a bridge to the open-source ecosystem

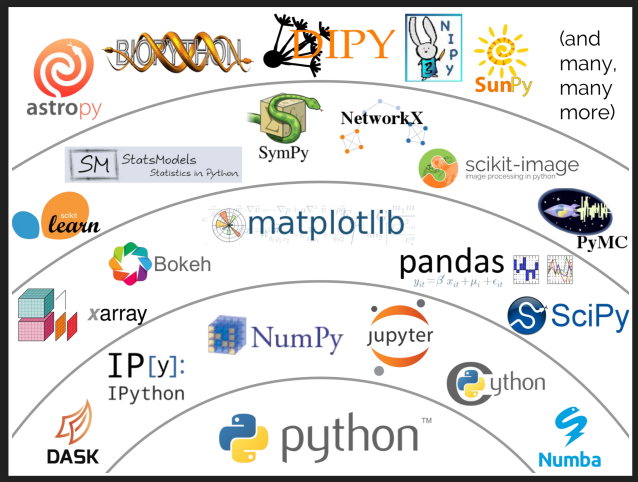


Figure Credit: "The Unexpected Effectiveness of Python in Science", PyCon 2017

Bluesky is written in Python, which is very popular

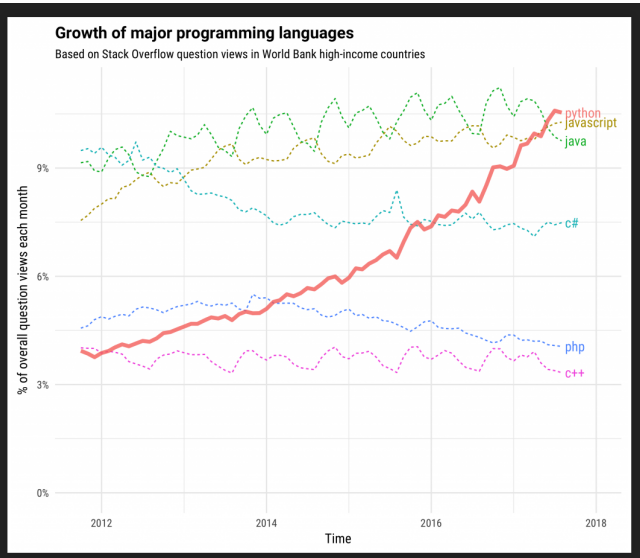


Figure Credit: Stack Overflow Blog https://stackoverflow.blog/2017/09/06/incredible-growth-python/

Bluesky is designed for the long term

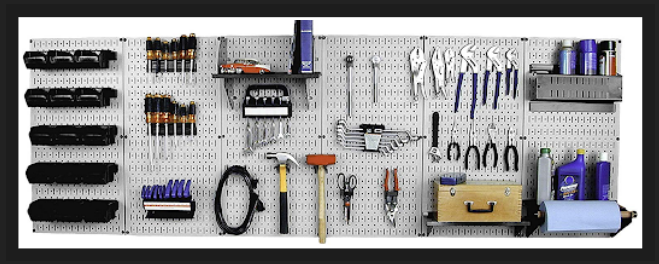
- Make it easy to keep **file-reading and -writing code separate from scientific code**.
- Support **streaming** (live-updating) visualization and processing, and **adaptive experiment steering**.
- Integrate well with web technologies and be **cloud**-friendly.
- But also meet users where they are!

Bluesky has individually useful core components

- **Bluesky Run Engine**: experiment orchestration and data acquisition
  - Emits data and metadata in a streaming fashion as it's available during acquisition
  - Centrally manages any interruptions (pause/resume), failure modes, etc.
- **Bluesky Plans**: experiment sequences (e.g. scans)
  - Designed with adaptive sequences in mind from the start
  - Leaves room for hardware-triggered scanning — interesting new work happening here

- **Ophyd**: interface to hardware
- **Suitcases**
  - streaming export to popular formats (e.g. CSV, TIFF, …. NeXus support in progress)
  - or streaming save lossless storage (e.g. MongoDB, msgpack — plus external large arrays)
- **Tiled and Databroker**: search and access saved data

Other facilities have adopted Bluesky piecemeal, adapting, extending, or replacing components to meet their requirements.



List of facilities known to use Bluesky (1 of 2)

- NSLS-II (used at 26/28 beamlines)
- LCLS (widespread use) and SSRL (one or two instruments)
- APS (scaling up from a couple beamlines to dozens)
- Australian Synchrotron (several beamlines)
- ALS (at least one beamline, also scaling up)
- Diamond (evaluating, but has made significant development investments)

List of facilities known to use Bluesky (2 of 2)

- Canadian Light Source (at least one)
- PSI (evaluating, not yet committed to adoption)
- Pohang Light Source II
- Various academic labs
- BESSY II

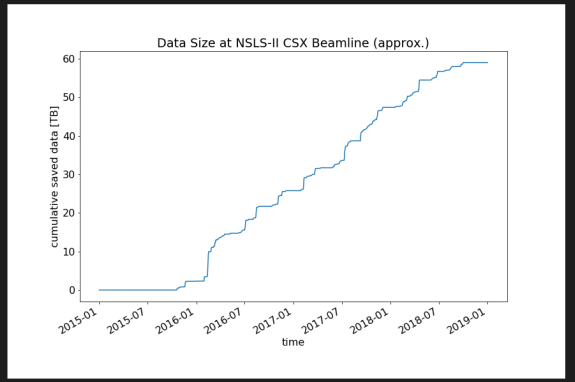
USER FACILITIES HAVE A DATA PROBLEM

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?

HPC is becoming more accessible.

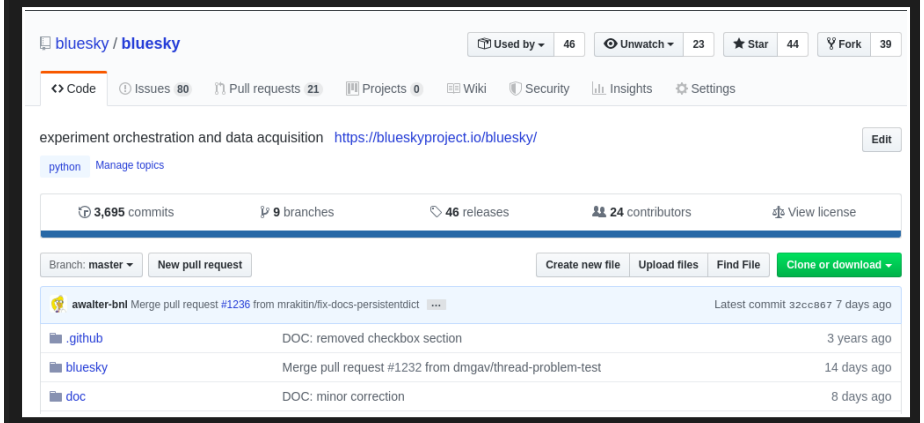
One inviting example: jupyter.nersc.gov

- 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.

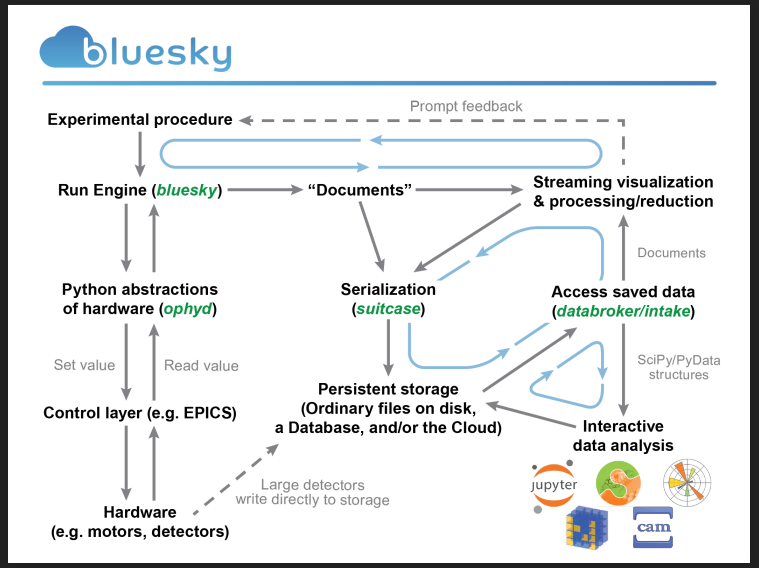
Bluesky is designed for Distributed Collaboration

- Facilities and instruments within a facility can share **common components** and benefit from a share knowledge base and a shared code base
- While also having **room to innovate and specialize** to suit their own priorities and timelines.
- The scientific Python community is an example of how this can work well.
- Use the parts of Bluesky that work for you *a la carte*, **extend** them, or **replace** them.

Bluesky is designed for Distributed Collaboration (cont.)

- This is not an all-or-nothing framework that you have to buy into; it's a **mini-ecosystem of co-developed but individually useful tools** that you can build on.
- It's all in Python. Some beamline staff and partner users have built on it.

BLUESKY ARCHITECTURE



Layered design of Python libraries that are:

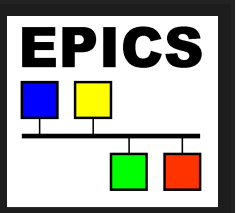
- co-developed and compatible...
- ...but individually usable and useful
- with well-defined programmatic interfaces

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

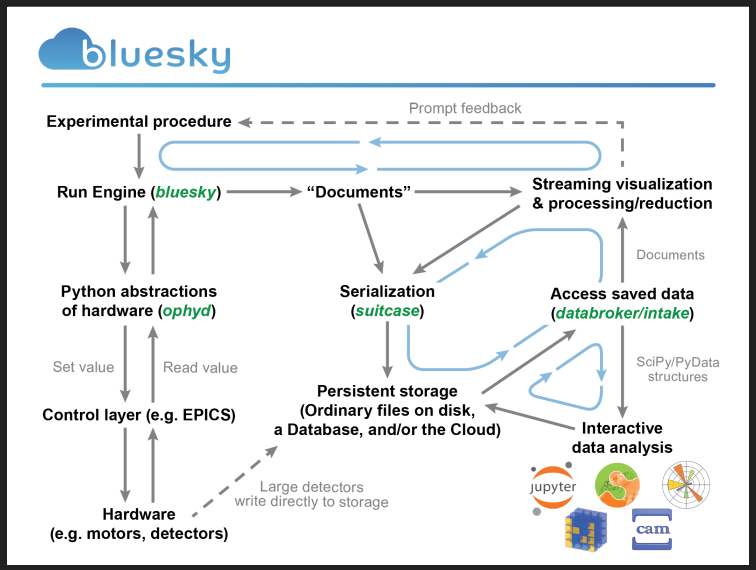
Device Drivers and Underlying Control Layer(s)

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



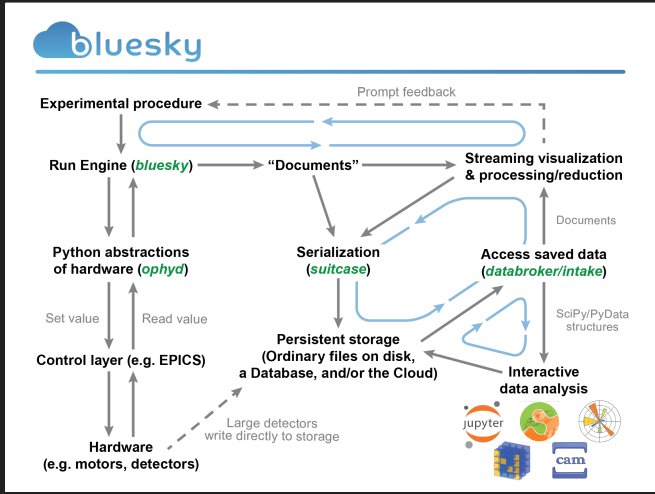
Ophyd abstracts over the specific control layer.



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

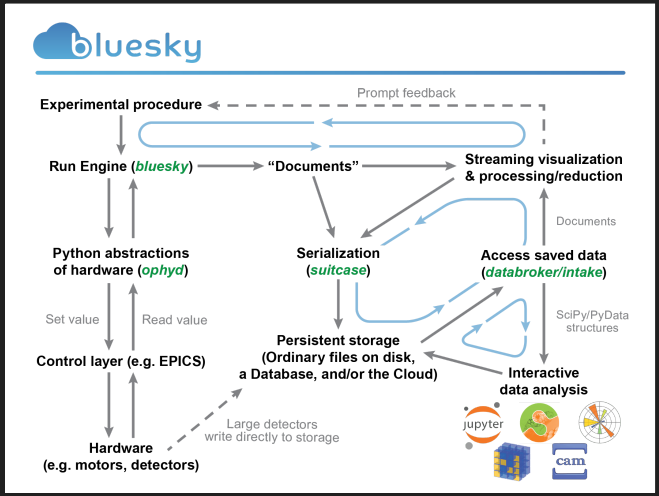
...and streaming-friendly viz...

...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.").

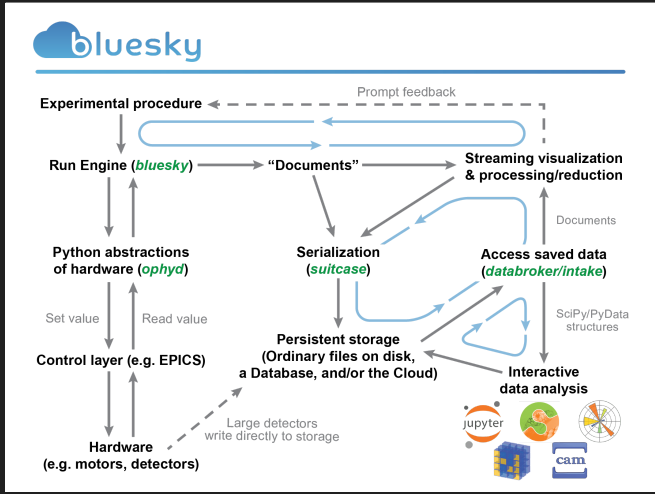
Suitcase encodes documents for storage or export.



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():
        ...
```

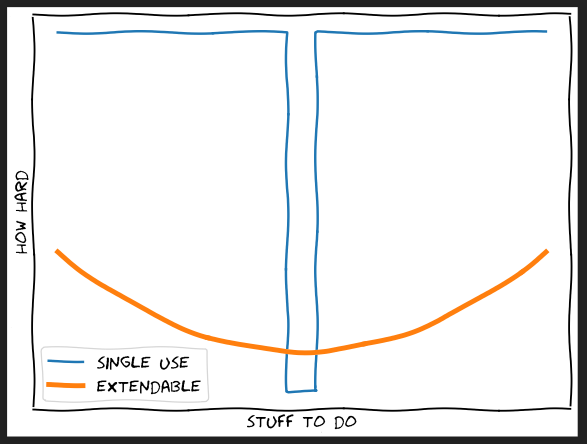
Interface Example: numpy array protocol

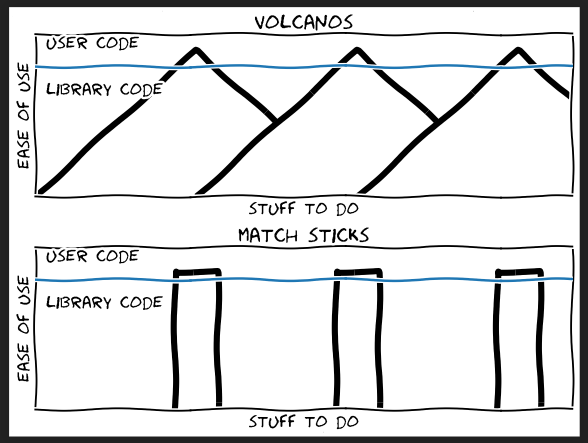
```
import pandas
import numpy

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





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.

Build a lasting collaboration

- Governance model (in process)
- **Maintainers**: per repo, make day-to-day decisions and set processes as appropriate to the repo
- **Technical Steering Committee**: arbitrate when maintainers cannot reach rough consensus
- **Project Advisory Board**: management-level stakeholders, oversee big-picture priorities
- Currently in process of assembling these groups

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).

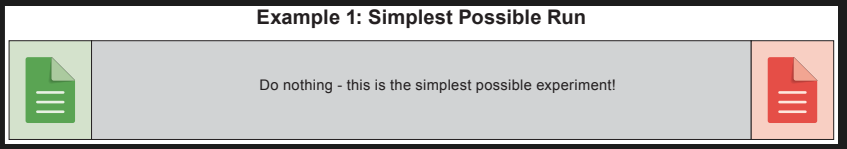
Run Start: Metadata about this run, including everything we know in advance: time, type of experiment, sample info., etc.

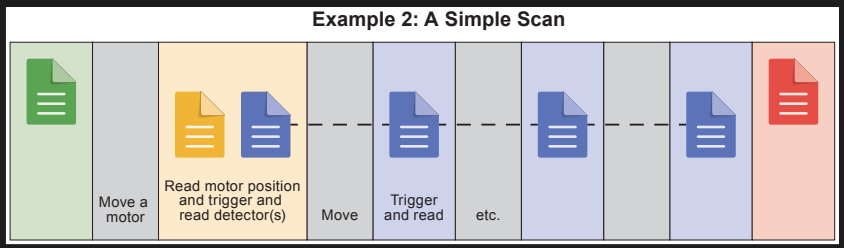
Run Stop: Additional metadata known at the end: what time it completed and its exit status (success, aborted, failed)

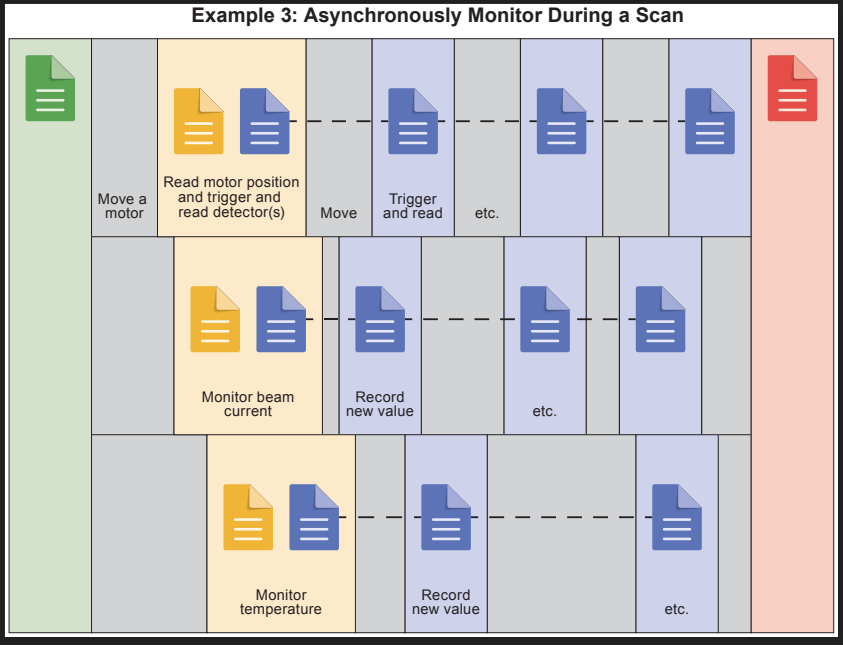
Event: Readings and timestamps

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

Example 1: Simplest Possible Run







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).

FLY SCANS

- For high performance fly scanning, coordination is needed "below" Bluesky in hardware.
- Bluesky simply provide a way to:
  - Configure
  - Start ("Kickoff")
  - Incrementally collect data ("Collect")
  - Initiate or await completion ("Complete")

- The status quo in Bluesky is very coarse.
- Highly flexible (good place to start...)
- But each fly scan application is built from scratch (leads to duplicated efforts)

This is an area of very active development is Bluesky.

Coordinated efforts underway at:

- Diamond Light Source
- Australian Synchrotron
- NSLS0II

- Diamond has invested a decade of research into fly-scanning in previous Python projects.
- Prototype from Diamond applying this expertise in a Bluesky-compatible way: Bluesky
- Work in progress to integrate this with Bluesky itself: bluesky PR#1502

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: ≫ 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

"QUEUE SERVER": AN EDITABLE CONTROL QUEUE

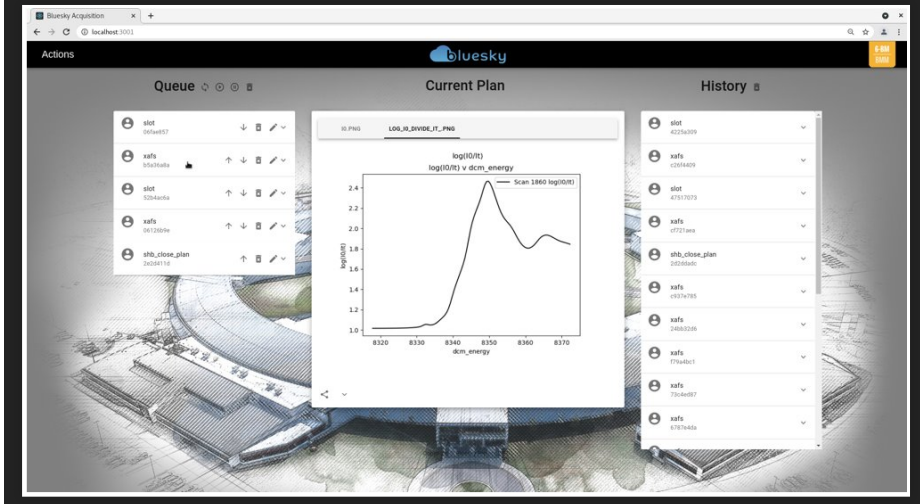
Bluesky Queue Server

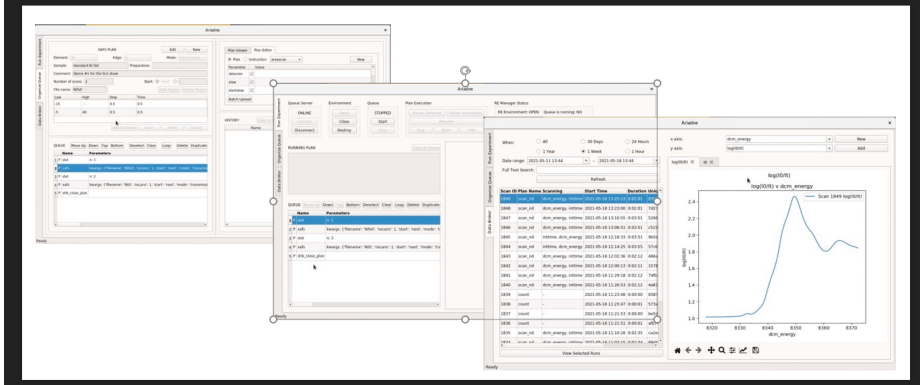
- Support remote and multi-tenant data acquisition
- Documentation: bluesky-queueserver
- Has been used for user experiments
- Still under rapid development

Bluesky's first target was users coming from SPEC

New Capability: Editable Control Queue

- Provides an editable queue of Bluesky plans to run
- All the same Bluesky plans (experiment procedures) work
- All the same Ophyd devices work
- Can be easily populated from a user's Excel spreadsheet
- You can safely mutate—rearrange and edit items—during acquisition
- Well suited to graphical interfaces





Separation between user app and queue server

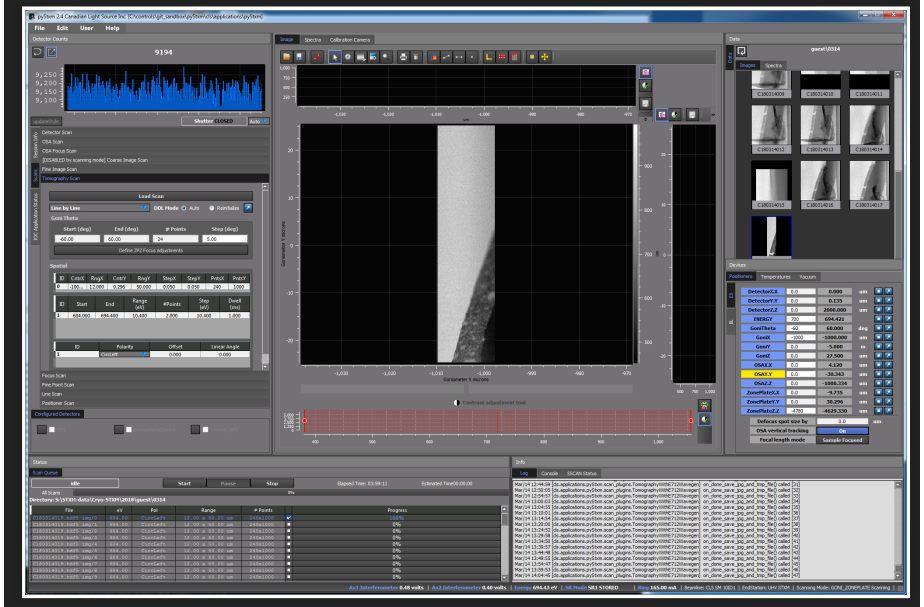
- **If app is closed or crashes, acquisition continues**. Just restart app to reconnect.
- The app can provide access controls
  - Guiderails (avoid too many options)
  - Security
- App can run on different machine from queue
- Many client programs can be used simultaneously to monitor or control the queue (web, desktop GUI, commandline)

SURVEY OF COMMUNITY UI DEVELOPMENTS

Various institutions are building graphical user interfaces on Bluesky.

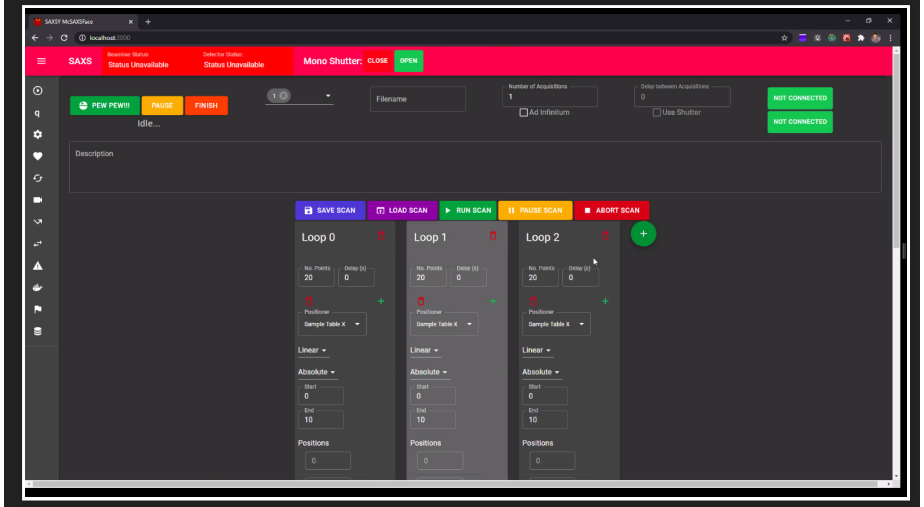
pyStxm at Canadian Light Source (Russ Berg)

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



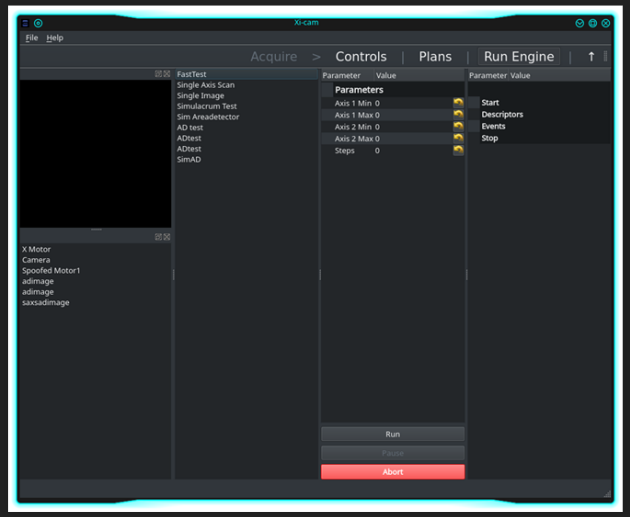
GUI for SAXS at Australian Synchrotron (Stephen Mudie)

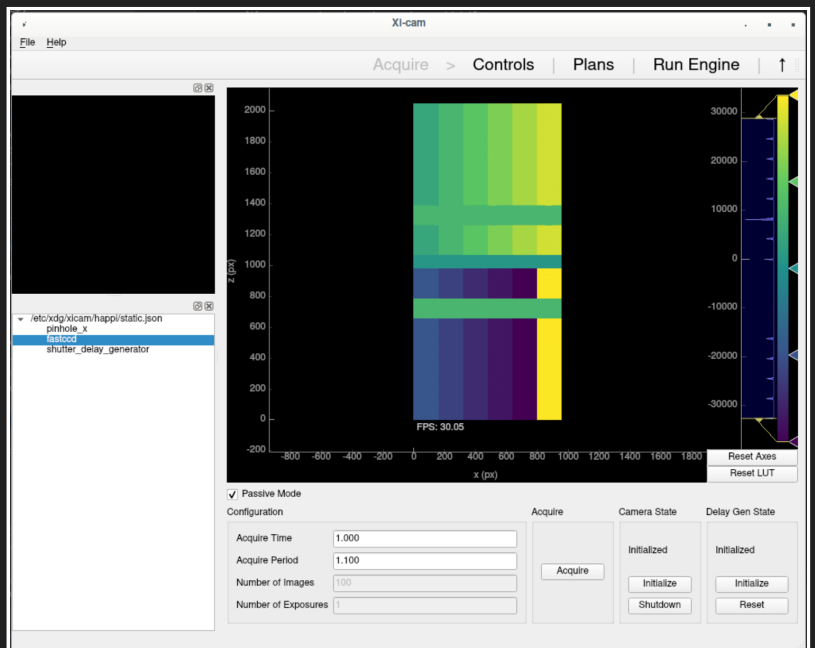
- Web-based (React)
- Code partially available at AustralianSynchrotron/saxs_beamline_library_react
- Plans to be fully open in a week or so



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

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





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

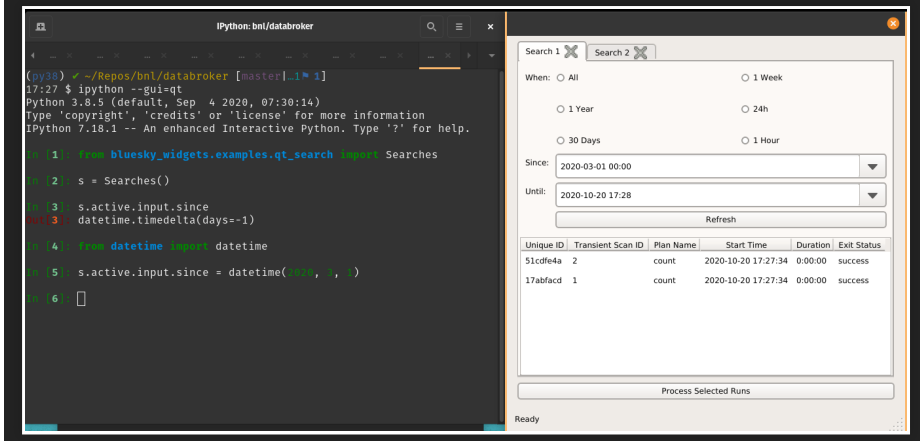
A new project aimed at sharing GUI components built on Bluesky interfaces

bluesky/bluesky-widgets

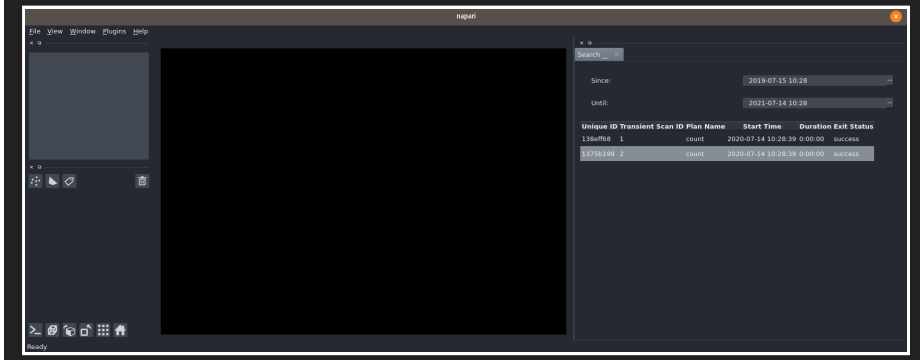
Goals

- 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.

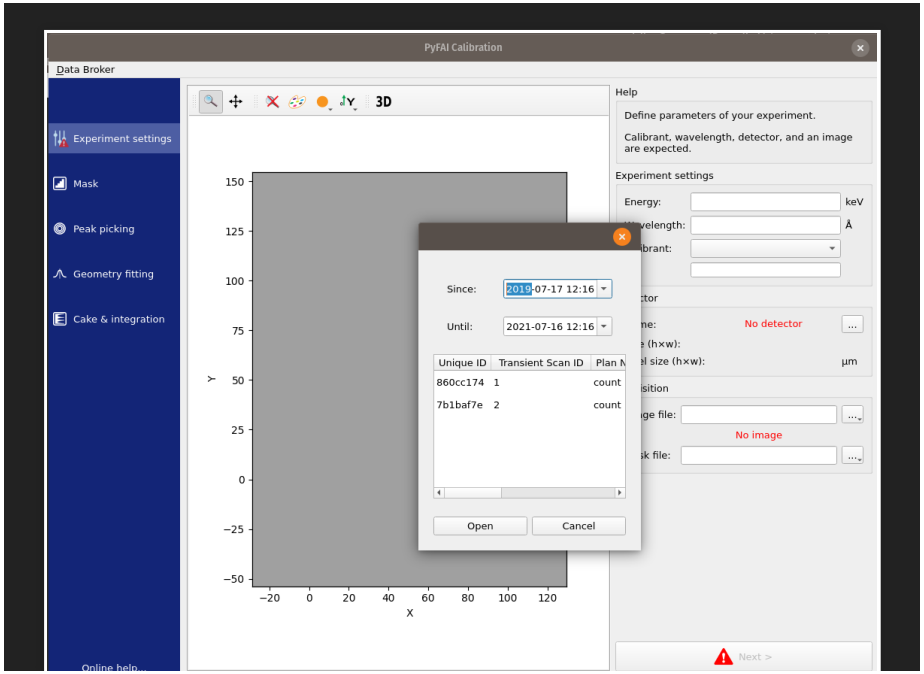
Examples of integrating Data Broker search into existing software...



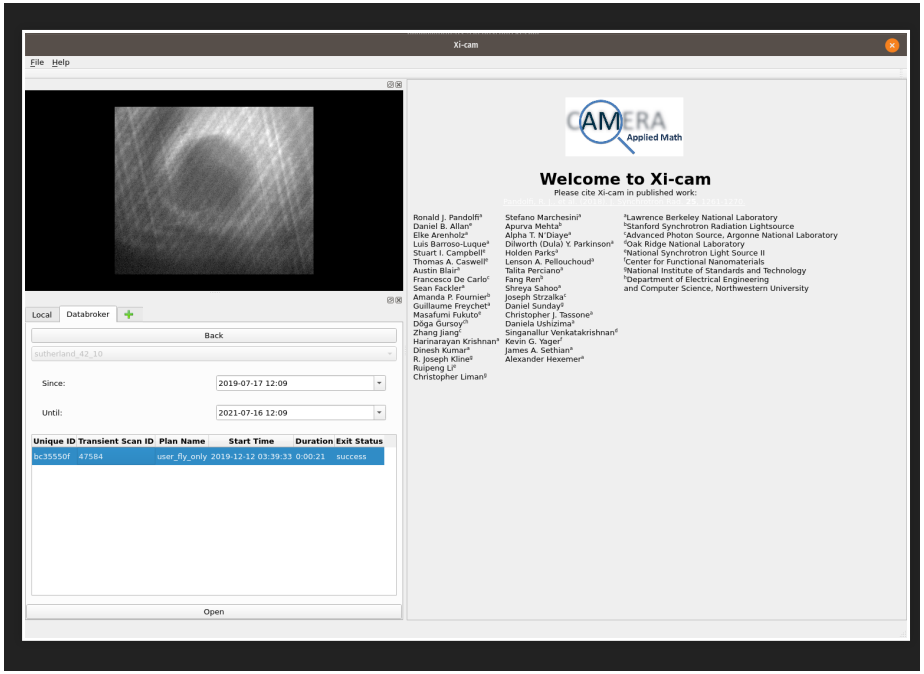
Model can be manipulated from IPython terminal



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



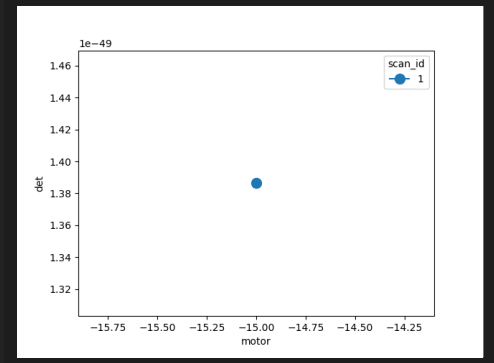
Search Data Broker from PyFAI (powder diffraction software)



Search Data Broker from Xi-CAM

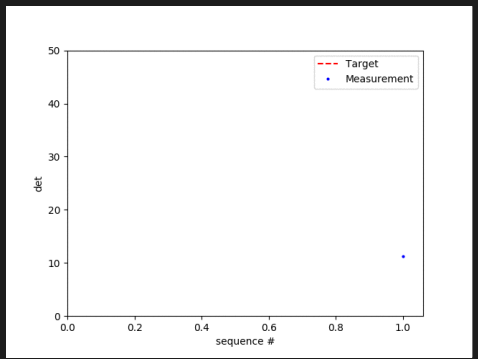
PAYOFF:
EASY AND ROBUST INTEGRATION WITH EXISTING SOFTWARE

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

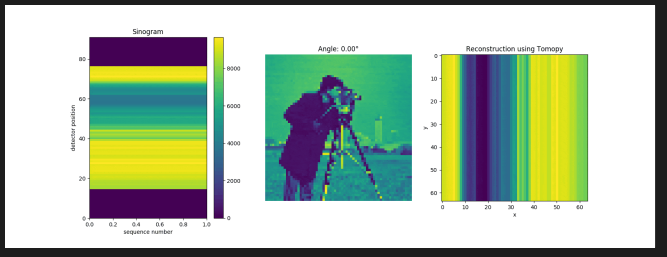


The system is designed to make fast feedback easy to write.

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

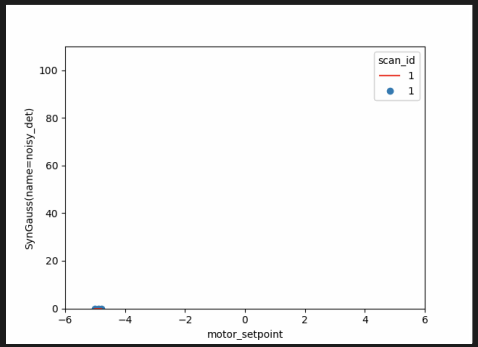


A stream of images from a linear detector is reconstructed into a volume using *tomopy* (APS).

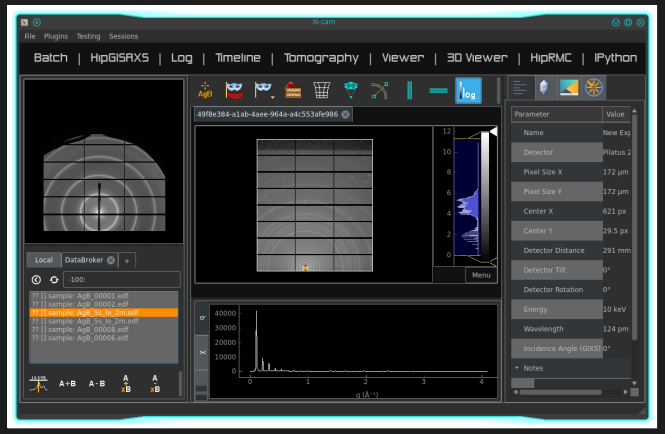


It took one TomoPy developer and one Bluesky developer less than 20 minutes to write this.

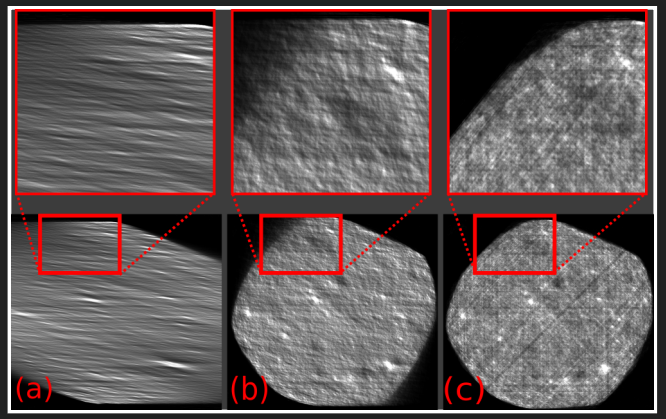
A Gaussian is fit to a stream of measured data using the Python library *lmfit* (from U. Chicago / APS).



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