National Synchrotron Light Source II

bluesky

Brookhaven National Laboratory

U.S. DEPARTMENT OF ENERGY

Bluesky Workshop

Max Rakitin, Dmitri Gavrilov

Data Acquisition and Detectors group, Data Science and Systems Integration Program
National Synchrotron Light Source II
Upton, NY (USA)

EPICS Meeting, ORNL
September 20, 2024

Schedule

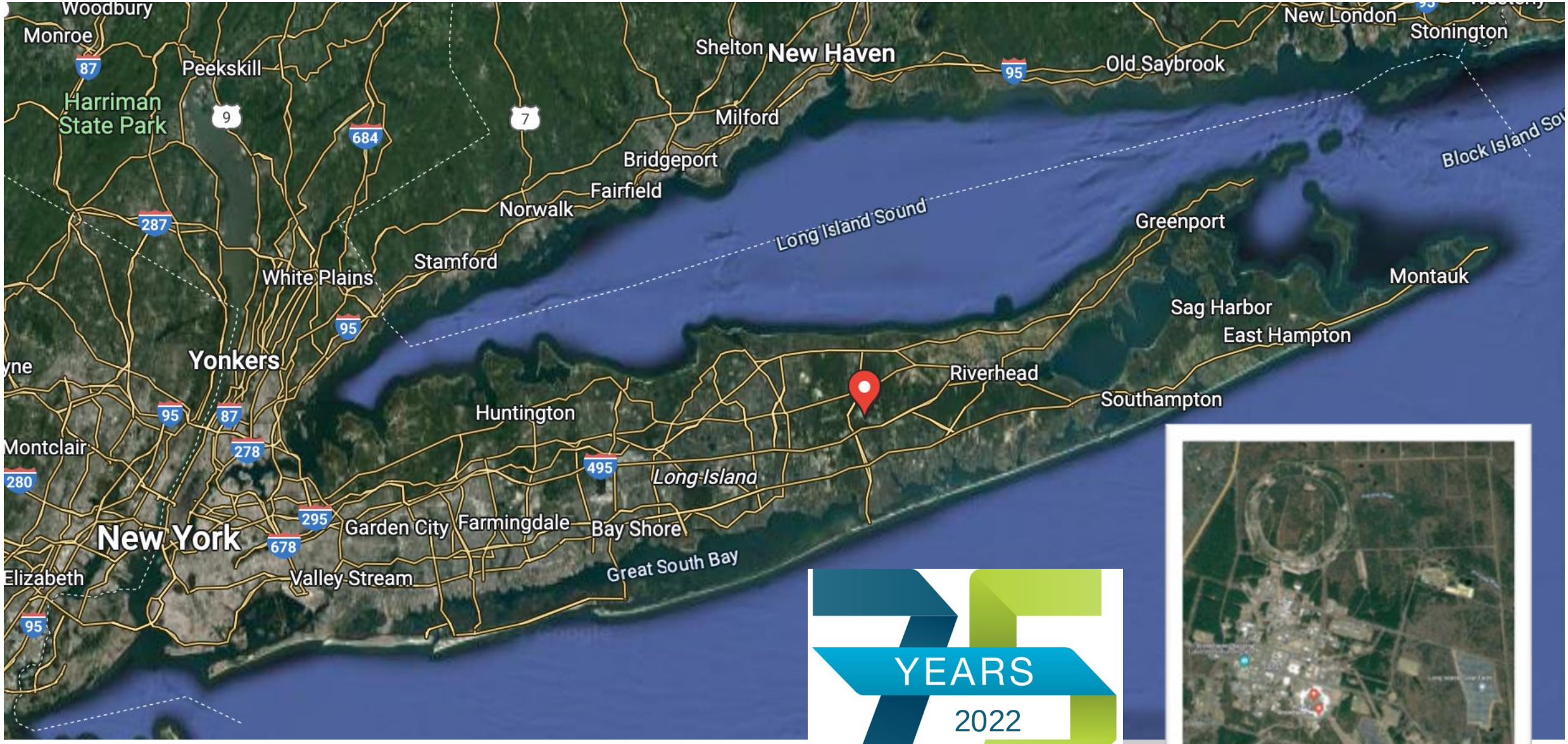
9:00 – 10:20: Bluesky overview and live demo (with epics-containers)

10:20 – 10:50: Break / Q&A

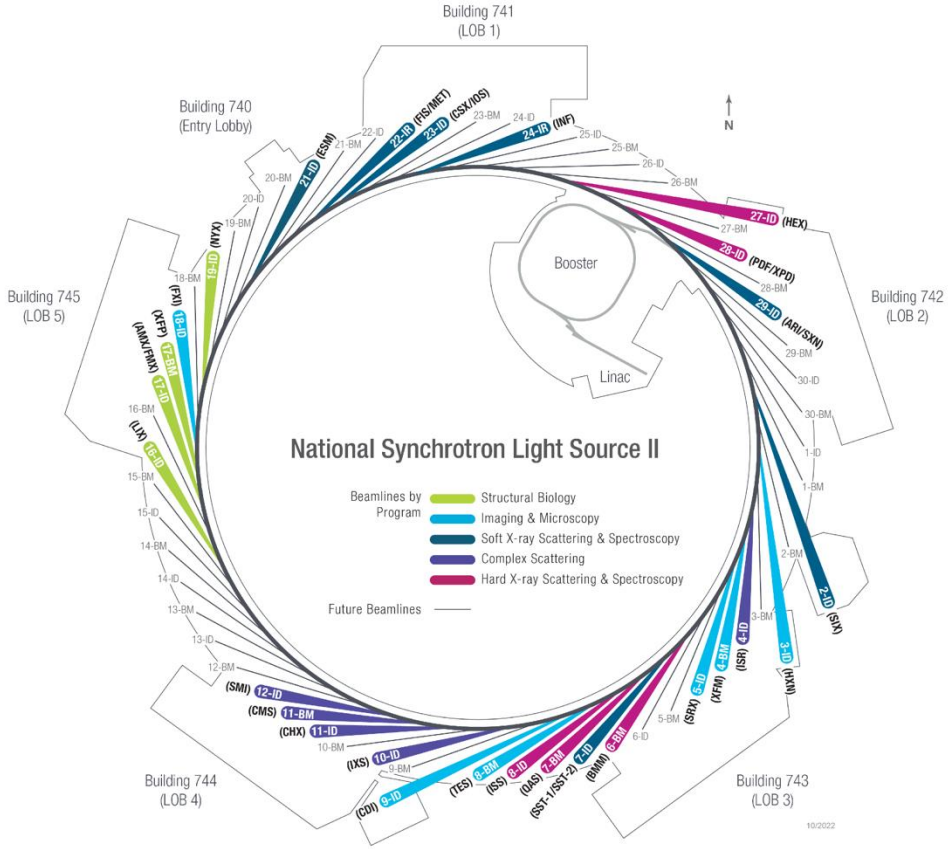
10:50 – 12:10: Bluesky-Queueserver overview and live demo

Bluesky Overview: Outline

- About NSLS-II
- Bluesky Data Acquisition & Experiment Orchestration Framework
- Applications
- Live Demo


Woodbury
Monroe
87
Peekskill
Harriman
State Park
9
684
7
Shelton
New Haven
95
Old Saybrook
New London
Stonington
Milford
Bridgeport
Fairfield
Norwalk
Block Island Sou
Long Island Sound
287
Stamford
Greenport
White Plains
95
Montauk
Sag Harbor
East Hampton
Yonkers
Riverhead
Huntington
Southampton
95
87
Montclair
278
280
495
Long Island
295
Garden City
Farmingdale
Bay Shore
New York
678
Great South Bay
Elizabeth
Valley Stream
95

National Synchrotron Light Source II



Leveraging DS/ML to exploit world leading brightness

29 state-of-the-art beamlines and growing!

NSLS-II: ~350 employees

DSSI: ~65 employees

Bluesky Data Acquisition & Experiment Orchestration Framework

An open-source software project of individually useful components used in experimental orchestration

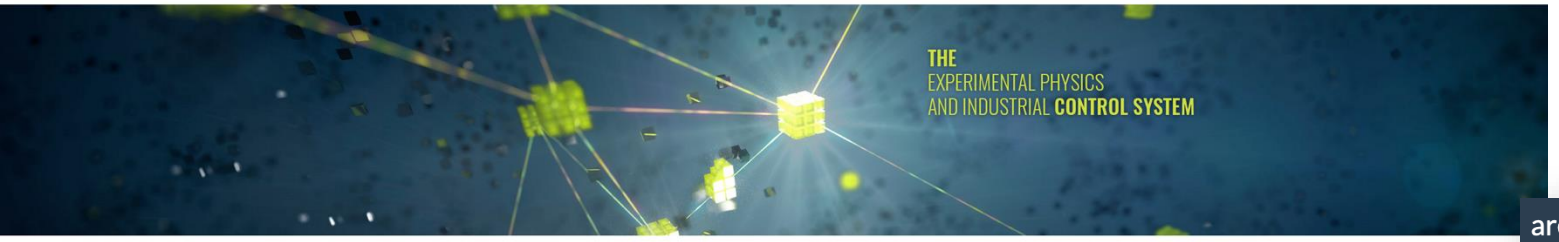
A bit of history…

Software Developed at Brookhaven Lab Could Advance Synchrotron Science Worldwide

October 2, 2017

Thomas Caswell (left) and Dan Allan (right), two of Bluesky's creators.

https://www.bnl.gov/newsroom/news.php?a=212470



https://epics-controls.org
https://docs.epics-controls.org
https://areadetector.github.io

areaDetector 3-12

areaDetector

EPICS Support for Multidimensional Detectors

This page is the home of areaDetector, an application for controlling area (2-D) detectors, including CCDs, pixel array detectors, and online imaging plates.

FREE AND OPEN SOURCE

EPICS is developed as a public open source project. The source code is freely available according to the EPICS Open License.

DEVELOPED COLLABORATIVELY

EPICS was created through collaborative contributions from scientific facilities since a long time. It is the preferred choice for complex, large scale distributed control system applications.

POWERFUL AND RELIABLE

The launch of EPICS 7 marks the biggest change of the EPICS code base for over 10 years. The new, feature-rich pvAccess protocol enables many new applications with unprecedented performance and capacity.
Read more

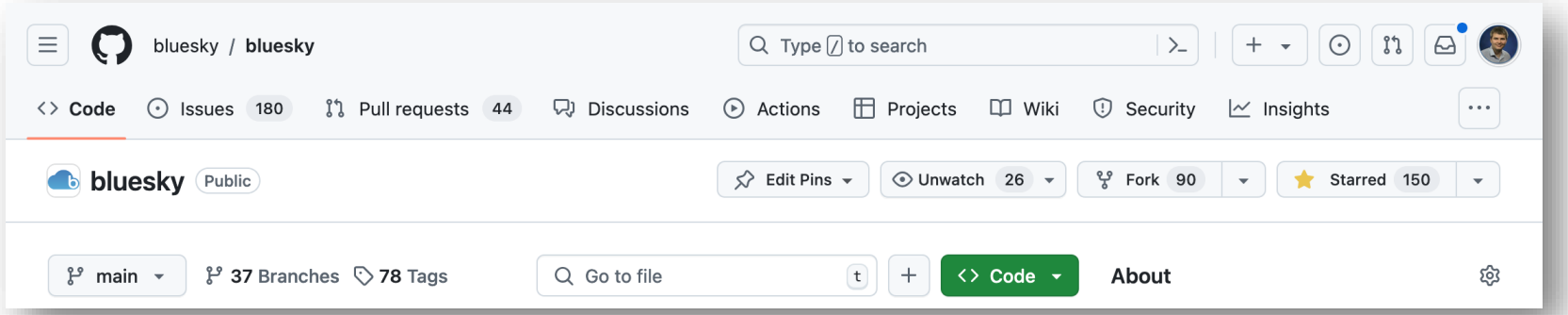
17th Int. Conf. on Acc. and Large Exp. Physics Control Systems ICALEPCS2019, New York, NY, USA JACoW Publishing
ISBN: 978-3-95450-209-7 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2019-MOCPR02

THE EPICS COLLABORATION TURNS 30

L. R. Dalesio, Osprey DCS LLC, Ocean City, USA
A. N. Johnson, Argonne National Laboratory, Lemont, USA
K.-U. Kasemir, Oak Ridge National Laboratory, Oak Ridge, USA

35 in 2024!

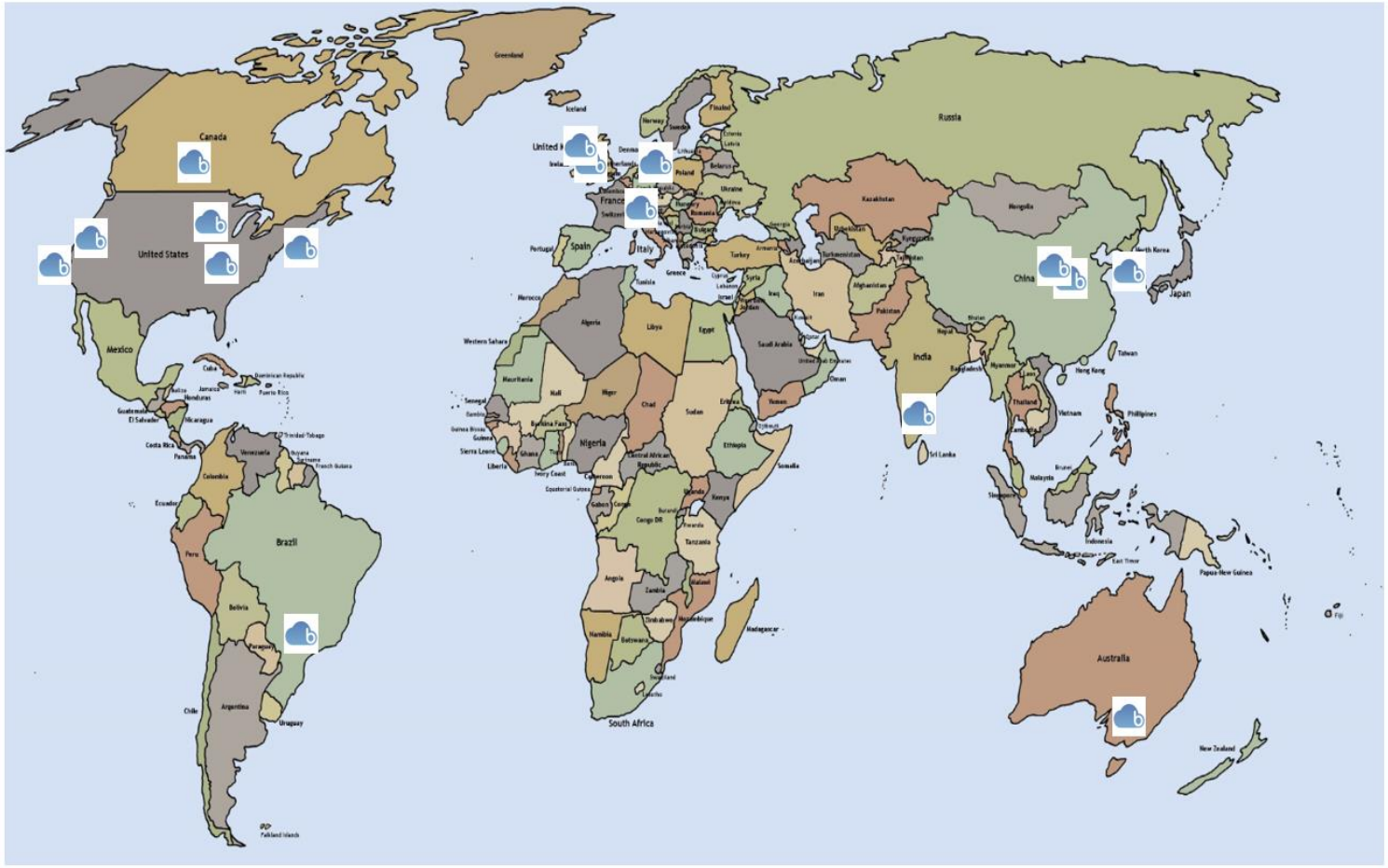
Bluesky Collaboration and Community



- NSLS-II, APS, ALS, LCLS & SSRL (US)
- Diamond Light Source (UK)
- BESSY-II (Germany)
- Canadian Light Source
- PSI (Switzerland)
- Australian Synchrotron
- Pohang Synchrotron (South Korea)
- University of Wisconsin–Madison
- …

Join the community: https://blueskyproject.io/mattermost/

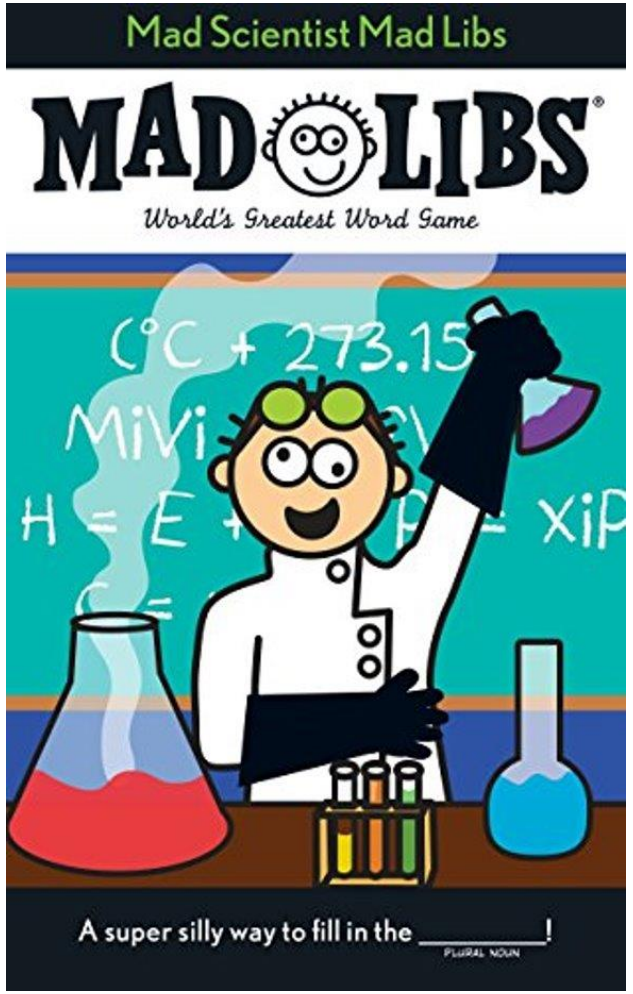
Collaborations: spans 5 continents



Bluesky is “Mad Libs” for science experiments.

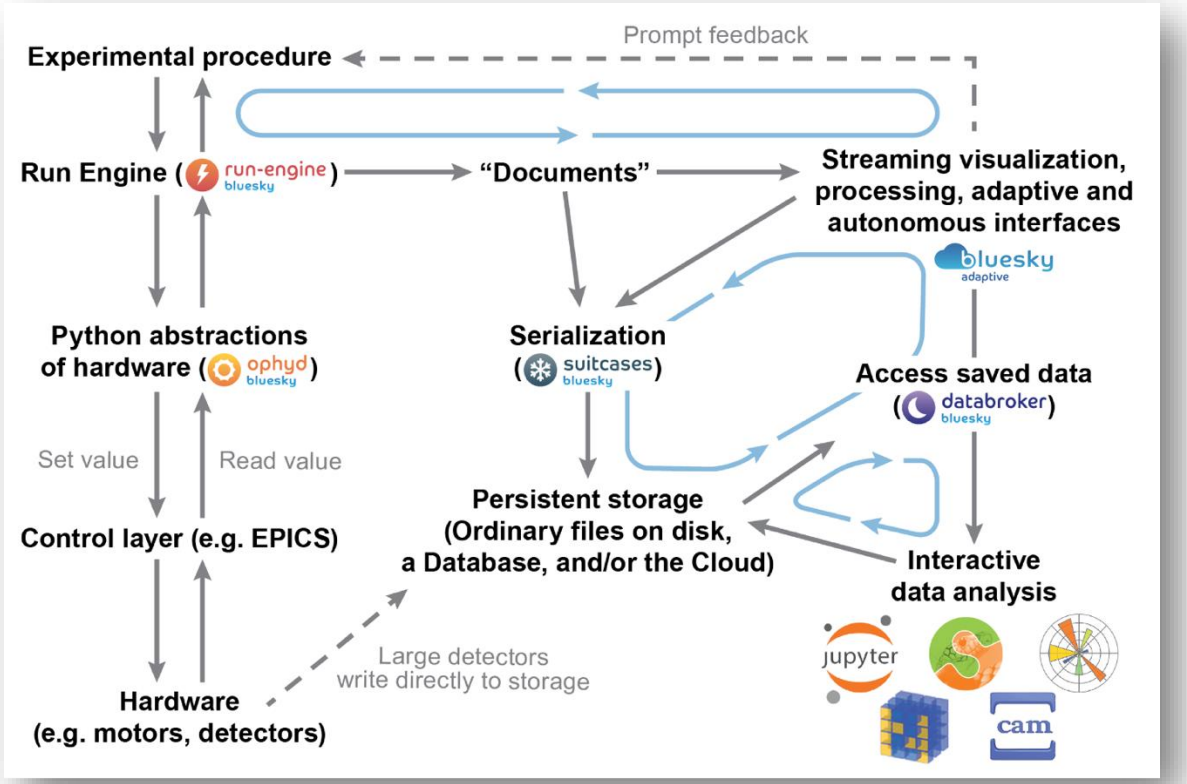
Scan the sample stage in a grid pattern over some range with some resolution while taking measurements with these sensors at each position.

Record that the sample under study was Sample X and that the mood at the beamline that day was hopeful.

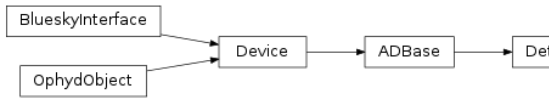


Bluesky Experiment Orchestration Framework

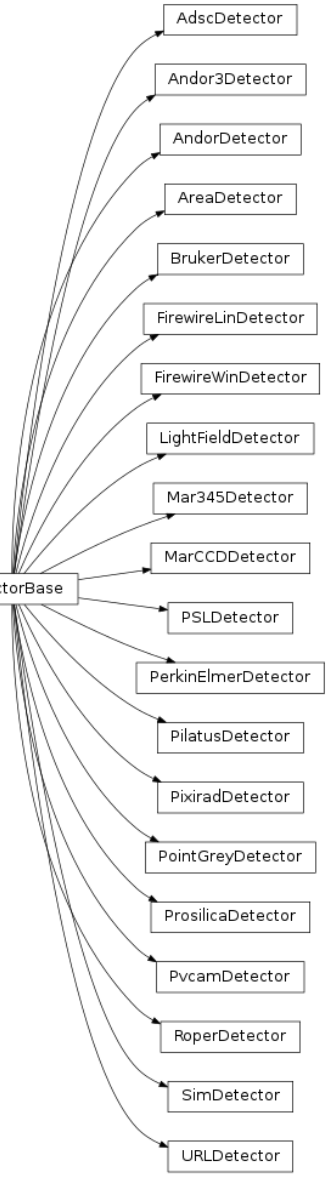
https://blueskyproject.io | DOI: 10.1080/08940886.2019.1608121




```python
from ophyd import Device, EpicsMotor
from ophyd import Component as Cpt

class StageXY(Device):
    x = Cpt(EpicsMotor, ':X')
    y = Cpt(EpicsMotor, ':Y')

stage = StageXY('STAGE_PV', name='stage')
```

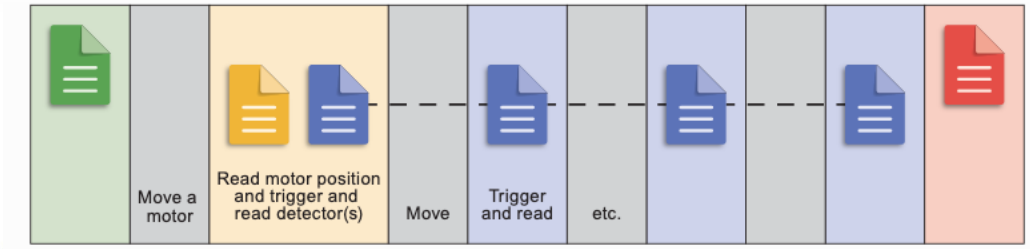


EPICS AreaDetector support




```python
from ophyd import AreaDetector, SingleTrigger

class MyDetector(SingleTrigger, AreaDetector):
    pass

prefix = 'XF:23ID1-ES{Tst-Cam:1}'
det = MyDetector(prefix)
```

"Documents"



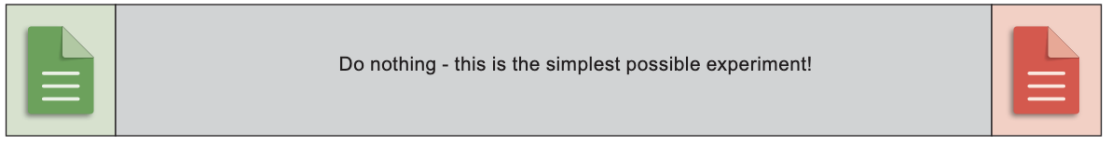

```
RE(bp.scan([det], stage.x, -10, 10, 5))
```

Non-EPICS hardware is supported. Example (laptop camera):
https://github.com/mrakitin/Berlin-2019-tutorials-workshop/blob/master/demo/LaptopCam.ipynb

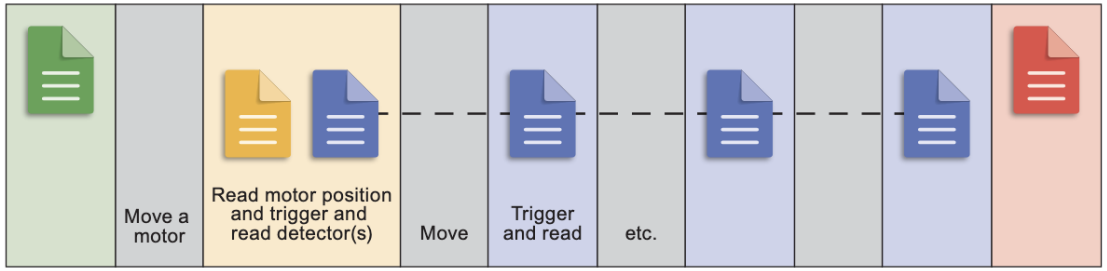
Document Model

NSLS-II: documents are stored in

Example 1: Simplest Possible Run



Example 2: A Simple Scan



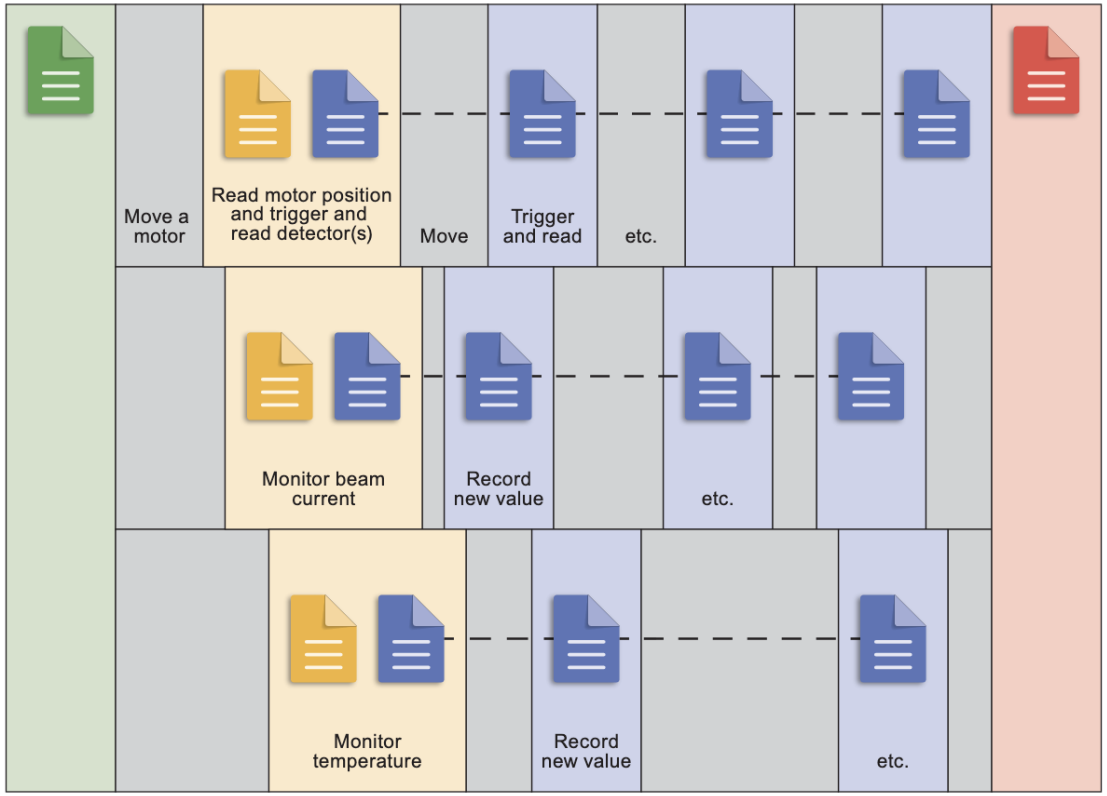
Run Start: Metadata about this run, including everything we know in advance: time, type of experiment, sample info., etc.

Event: Readings and timestamps

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

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

Example 3: Asynchronously Monitor During a Scan



Built-in plans

Python generators

Composable with “yield from …”

`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
`rel_spiral_fermat`	Relative fermat spiral scan
`rel_spiral_square`	Relative square spiral scan, centered around current (x, y) position.
`adaptive_scan`	Scan over one variable with adaptively tuned step size.
`rel_adaptive_scan`	Relative scan over one variable with adaptively tuned step size.
`tune_centroid`	plan: tune a motor to the centroid of signal(motor)
`tweak`	Move and motor and read a detector with an interactive prompt.
`ramp_plan`	Take data while ramping one or more positioners.
`fly`	Perform a fly scan with one or more 'flyers'.

Documents stored in Tiled

Tiled is a **data access** service for data-aware portals and data science tools.

Tiled puts an emphasis on **structures** rather than formats, including:

- N-dimensional strided arrays (i.e. numpy-like arrays)
- Sparse arrays
- Tabular data (e.g. pandas-like “dataframes”)
- Nested, variable-sized data (as implemented by AwkwardArray)
- Hierarchical structures thereof (e.g. xarrays, HDF5-compatible structures like NeXus)

Tiled implements extensible **access control enforcement** based on web security standards.

https://blueskyproject.io/tiled/ https://tiled-demo.blueskyproject.io/

Ophyd-async

Ophyd-async is a Python library for asynchronously interfacing with hardware. Uses both CA and PVA EPICS protocols.

Both **ophyd** and **ophyd-async** are typically used with the Bluesky Run Engine for experiment orchestration and data acquisition.

While EPICS is the most common control system layer that **ophyd-async** can interface with, support for other control systems like Tango will be added in the future. The focus of ophyd-async is:

- Asynchronous signal access, opening the possibility for hardware-triggered scanning (also known as fly-scanning)
- Simpler instantiation of devices (groupings of signals) with less reliance upon complex class hierarchies

Ophyd-async is still under active development, the release of v1.0.0 is planned later this year.

https://blueskyproject.io/ophyd-async/

Ophyd-async example:

```
# Import bluesky and ophyd
import matplotlib.pyplot as plt
from bluesky import RunEngine
from bluesky.callbacks.best_effort import BestEffortCallback
from bluesky.plan_stubs import mov, movr, rd  # noqa
from bluesky.plans import grid_scan  # noqa
from bluesky.utils import ProgressBarManager, register_transform
from ophyd import Component, Device, EpicsSignal, EpicsSignalRO

from ophyd_async.core import DeviceCollector
from ophyd_async.epics import demo

# Create a run engine, with plotting, progressbar and transform
RE = RunEngine({}, call_returns_result=True)
bec = BestEffortCallback()
RE.subscribe(bec)
RE.waiting_hook = ProgressBarManager()
plt.ion()
register_transform("RE", prefix="<")

# Start IOC with demo pvs in subprocess
pv_prefix = demo.start_ioc_subprocess()


# Create ophyd devices
class OldSensor(Device):
    mode = Component(EpicsSignal, "Mode", kind="config")
    value = Component(EpicsSignalRO, "Value", kind="hinted")


det_old = OldSensor(pv_prefix, name="det_old")

# Create ophyd-async devices
with DeviceCollector():
    det = demo.Sensor(pv_prefix)
    det_group = demo.SensorGroup(pv_prefix)
    samp = demo.SampleStage(pv_prefix)
```

Documentation & Tutorials

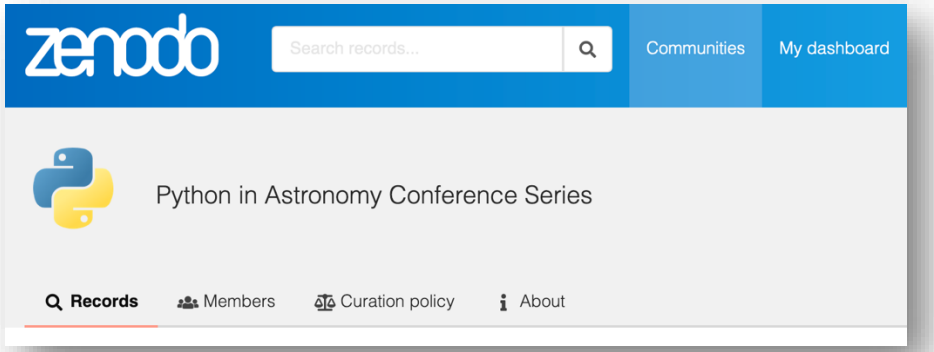
- https://blueskyproject.io/ – general Bluesky project overview.
- https://github.com/bluesky/tutorials – interactive tutorials using Jupyter (press the MyBinder link).
- https://blueskyproject.io/bluesky/main/tutorial.html – non-interactive tutorial on the bluesky library.

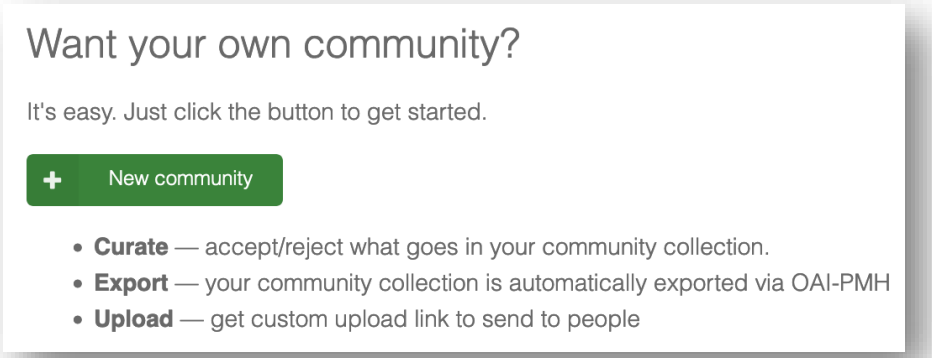
Deployment at NSLS-II

conda-forge for packages (bluesky, ophyd, databroker, etc.)
conda-pack:

- Build once, reuse multiple times!
- Zenodo: zenodo.org – up to 50 GBs per deposition (multiple files)
  - DOI – good for citations
  - https://about.zenodo.org/policies/
  - Used for testing with IPython profiles with beamline startup scripts
  - Integrates with GitHub repos
  - New! Communities, e.g., https://zenodo.org/communities/pyastro

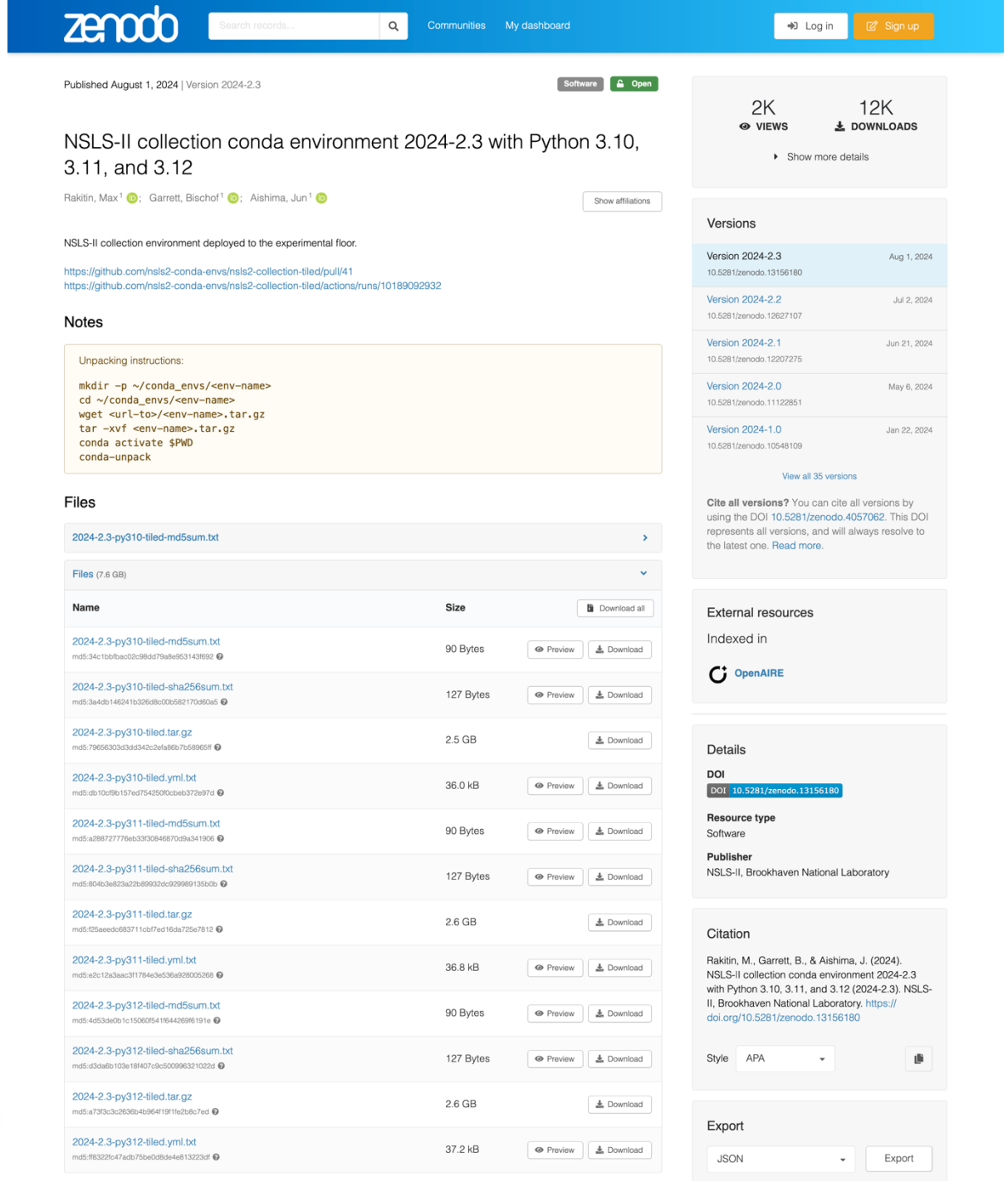
Deployment using Ansible Automation Platform (RedHat ecosystem)
Same environment for data collection and analysis (JupyterHub)





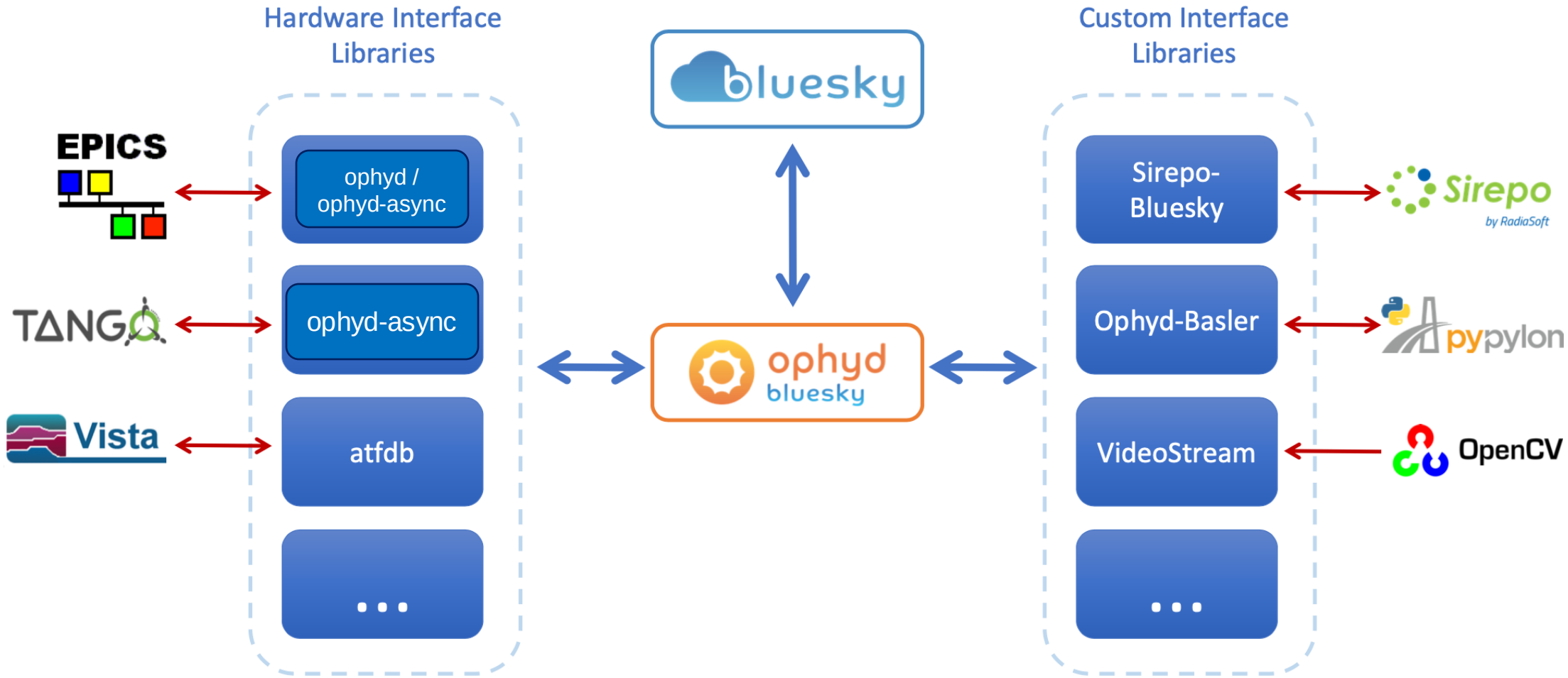
Longevity

- **Versions**: Data files are versioned. Records are not versioned. The uploaded data is archived as a Submission Information Package. Derivatives of data files are generated, but original content is never modified. Records can be retracted from public view; however, the data files and record are preserved.
- **Replicas**: All data files are stored in CERN Data Centres, primarily Geneva, with replicas in Budapest. Data files are kept in multiple replicas in a distributed file system, which is backed up to tape on a nightly basis.
- **Retention period**: Items will be retained for the lifetime of the repository. This is currently the lifetime of the host laboratory CERN, which currently has an experimental programme defined for the next 20 years at least.
- **Functional preservation**: Zenodo makes no promises of usability and understandability of deposited objects over time.
- **File preservation**: Data files and metadata are backed up nightly and replicated into multiple copies in the online system.
- **Fixity and authenticity**: All data files are stored along with a MD5 checksum of the file content. Files are regularly checked against their checksums to assure that file content remains constant.
- **Succession plans**: In case of closure of the repository, best efforts will be made to integrate all content into suitable alternative institutional and/or subject based repositories.

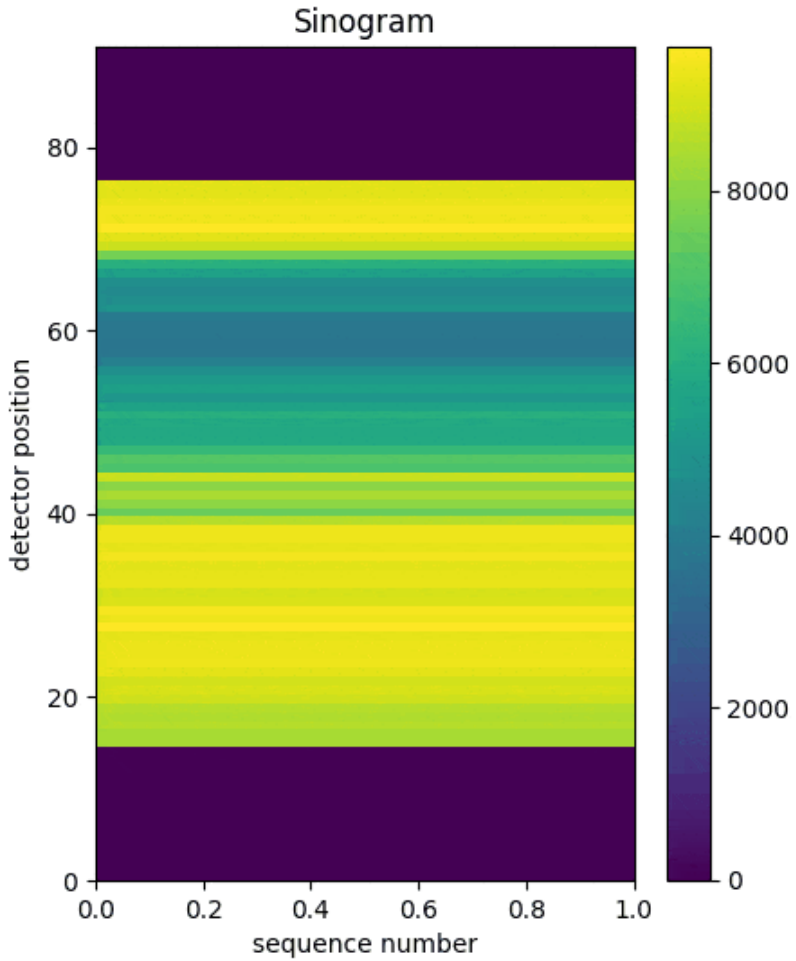

zenodo
Search records...
Communities
My dashboard
Log in
Sign up
Published August 1, 2024 | Version 2024-2.3
Software
Open
NSLS-II collection conda environment 2024-2.3 with Python 3.10, 3.11, and 3.12
Rakitin, Max[1]; Garrett, Bischof[1]; Aishima, Jun[1]
Show affiliations
NSLS-II collection environment deployed to the experimental floor.
https://github.com/nsls2-conda-envs/nsls2-collection-tiled/pull/41
https://github.com/nsls2-conda-envs/nsls2-collection-tiled/actions/runs/10189092932
Notes
Unpacking instructions:
mkdir -p ~/conda_envs/<env-name>
cd ~/conda_envs/<env-name>
wget <url-to>/<env-name>.tar.gz
tar -xvf <env-name>.tar.gz
conda activate $PWD
conda-unpack
Files
2024-2.3-py310-tiled-md5sum.txt
Files (7.6 GB)
Name
Size
Download all
2024-2.3-py310-tiled-md5sum.txt
md5:34c1bbfbac02c98dd79a8e953143f692
90 Bytes
Preview
Download
2024-2.3-py310-tiled-sha256sum.txt
md5:3a4db146241b326d8c00b582170d60a5
127 Bytes
Preview
Download
2024-2.3-py310-tiled.tar.gz
md5:79656303d3dd342c2efa86b7b58965ff
2.5 GB
Download
2024-2.3-py310-tiled.yml.txt
md5:db10cf9b157ed754250f0cbeb372e97d
36.0 kB
Preview
Download
2024-2.3-py311-tiled-md5sum.txt
md5:a288727776eb33f30846870d9a341906
90 Bytes
Preview
Download
2024-2.3-py311-tiled-sha256sum.txt
md5:804b3e823a22b89932dc929989135b0b
127 Bytes
Preview
Download
2024-2.3-py311-tiled.tar.gz
md5:f25aeedc683711cbf7ed16da725e7812
2.6 GB
Download
2024-2.3-py311-tiled.yml.txt
md5:e2c12a3aac3f1784e3e536a928005268
36.8 kB
Preview
Download
2024-2.3-py312-tiled-md5sum.txt
md5:4d53de0b1c15060f541f644269f6191e
90 Bytes
Preview
Download
2024-2.3-py312-tiled-sha256sum.txt
md5:d3da6b103e18f407c9c50099632102 2d
127 Bytes
Preview
Download
2024-2.3-py312-tiled.tar.gz
md5:a73f3c3c2636b4b964f19f1fe2b8c7ed
2.6 GB
Download
2024-2.3-py312-tiled.yml.txt
md5:ff8322fc47adb75be0d8de4e813223df
37.2 kB
Preview
Download
2K
VIEWS
12K
DOWNLOADS
Show more details
Versions
Version 2024-2.3
Aug 1, 2024
10.5281/zenodo.13156180
Version 2024-2.2
Jul 2, 2024
10.5281/zenodo.12627107
Version 2024-2.1
Jun 21, 2024
10.5281/zenodo.12207275
Version 2024-2.0
May 6, 2024
10.5281/zenodo.11122851
Version 2024-1.0
Jan 22, 2024
10.5281/zenodo.10548109
View all 35 versions
Cite all versions? You can cite all versions by using the DOI 10.5281/zenodo.4057062. This DOI represents all versions, and will always resolve to the latest one. Read more.
External resources
Indexed in
OpenAIRE
Details
DOI
DOI 10.5281/zenodo.13156180
Resource type
Software
Publisher
NSLS-II, Brookhaven National Laboratory
Citation
Rakitin, M., Garrett, B., & Aishima, J. (2024). NSLS-II collection conda environment 2024-2.3 with Python 3.10, 3.11, and 3.12 (2024-2.3). NSLS-II, Brookhaven National Laboratory. https://doi.org/10.5281/zenodo.13156180
Style
APA
Export
JSON
Export

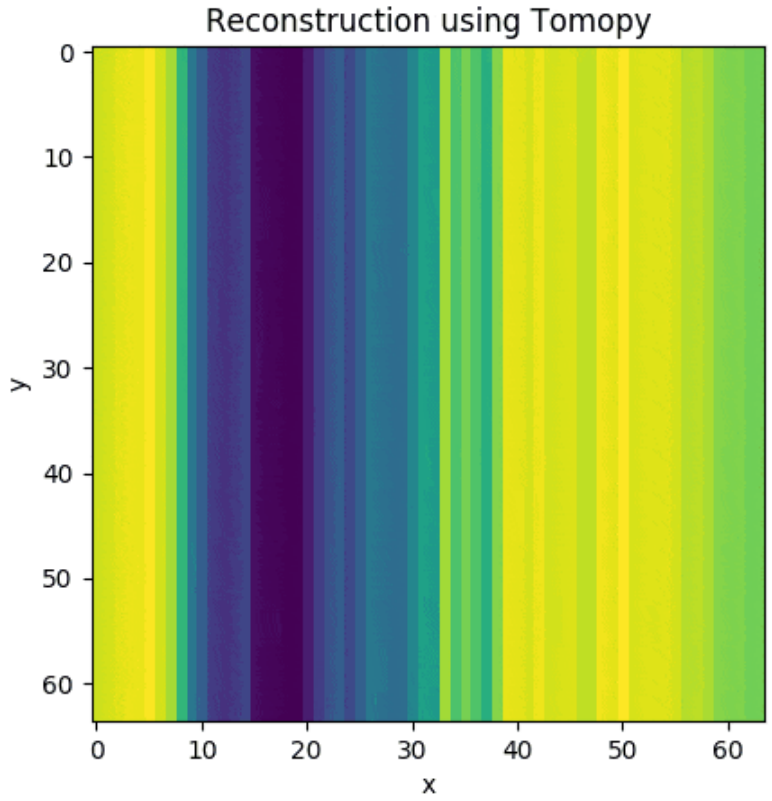
Applications

EPICS and non-EPICS Ophyd Support



Bluesky controls rotating of the image and streams slices tomographically reconstructed using *tomopy* via a callback on every “event” document





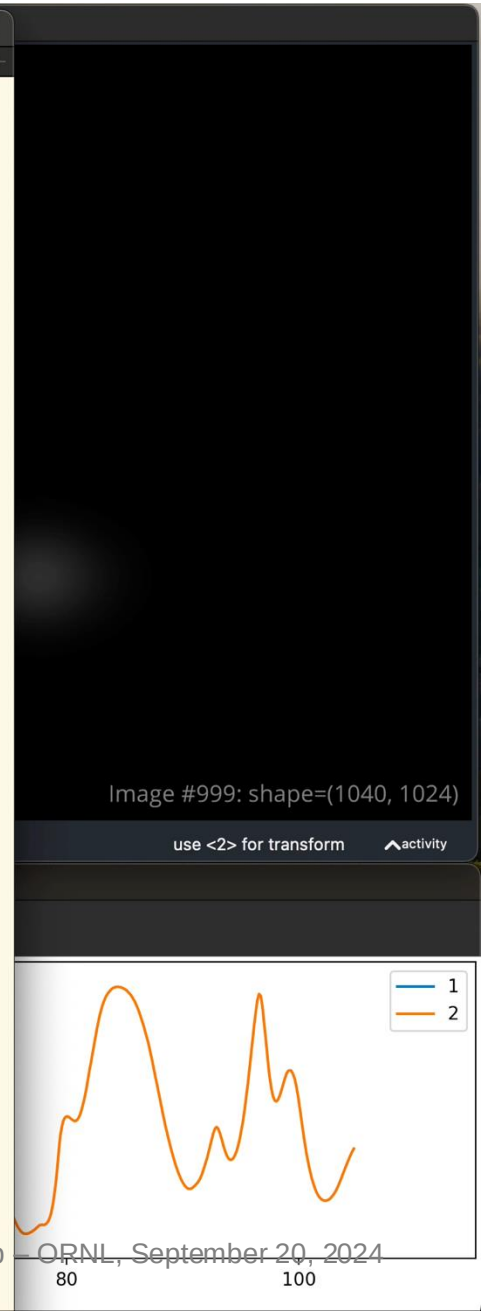
RE-playing a series of images of bouncing Gaussian beam through Pylon API to Napari visualization callback

IPython: src/ophyd-basler

Click to stop screen recording ⌥⌘1

IPython: src/ophyd-basler (python3.11) ⌘1 -zsh ⌘2

```
|       959 | 03:39:49.6 |            2.085 |
|       960 | 03:39:49.7 |            2.168 |
|       961 | 03:39:49.9 |            2.235 |
|       962 | 03:39:50.0 |            2.276 |
|       963 | 03:39:50.3 |            2.279 |
|       964 | 03:39:50.4 |            2.240 |
|       965 | 03:39:50.6 |            2.158 |
|       966 | 03:39:50.7 |            2.040 |
|       967 | 03:39:50.9 |            1.896 |
|       968 | 03:39:51.0 |            1.736 |
|       969 | 03:39:51.2 |            1.572 |
|       970 | 03:39:51.3 |            1.410 |
|       971 | 03:39:51.5 |            1.257 |
|       972 | 03:39:51.6 |            1.116 |
|       973 | 03:39:51.8 |            0.989 |
|       974 | 03:39:51.9 |            0.877 |
|       975 | 03:39:52.0 |            0.780 |
|       976 | 03:39:52.2 |            0.697 |
|       977 | 03:39:52.3 |            0.628 |
|       978 | 03:39:52.5 |            0.573 |
|       979 | 03:39:52.6 |            0.530 |
|       980 | 03:39:52.8 |            0.498 |
|       981 | 03:39:52.9 |            0.477 |
|       982 | 03:39:53.1 |            0.465 |
|       983 | 03:39:53.2 |            0.461 |
|       984 | 03:39:53.4 |            0.464 |
|       985 | 03:39:53.5 |            0.475 |
|       986 | 03:39:53.6 |            0.492 |
|       987 | 03:39:53.8 |            0.516 |
|       988 | 03:39:53.9 |            0.547 |
|       989 | 03:39:54.1 |            0.586 |
|       990 | 03:39:54.2 |            0.631 |
|       991 | 03:39:54.4 |            0.683 |
|       992 | 03:39:54.5 |            0.741 |
|       993 | 03:39:54.6 |            0.801 |
|       994 | 03:39:54.8 |            0.862 |
|       995 | 03:39:54.9 |            0.922 |
|       996 | 03:39:55.1 |            0.982 |
|       997 | 03:39:55.2 |            1.039 |
|       998 | 03:39:55.4 |            1.094 |
|       999 | 03:39:55.5 |            1.143 |
+-----------+------------+------------------+
|   seq_num |       time | basler_cam_mean |
+-----------+------------+------------------+
|      1000 | 03:39:55.7 |            1.186 |
+-----------+------------+------------------+
generator count ['ba30d428'] (scan num: 2)


In [5]: (uid,) = RE(bp.count([emulated_basler_camera], num=1000))
```



Beamline Simulations with Sirepo

https://www.sirepo.com
https://github.com/radiasoft/sirepo
https://doi.org/10.1107/S1600577518010986

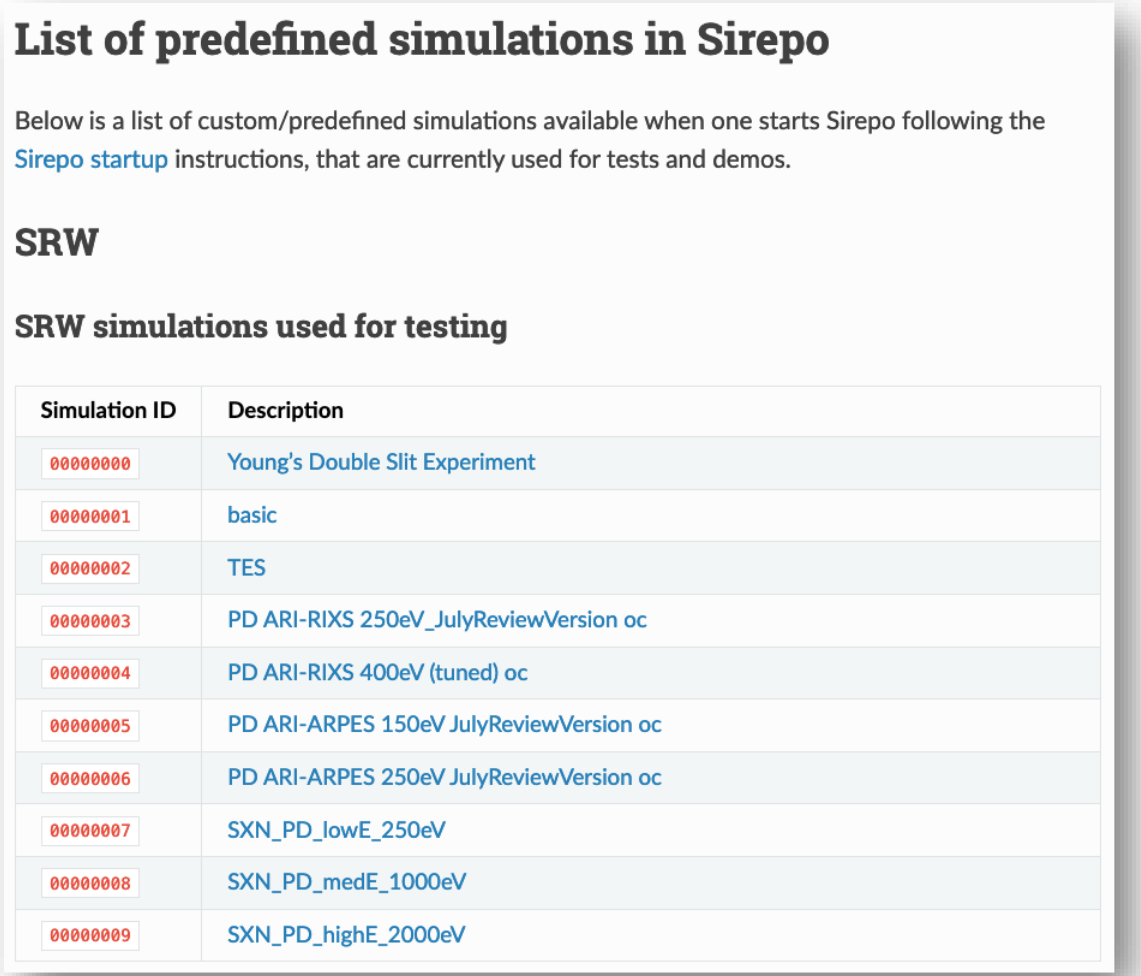
SUPPORTED CODES

- ACTIVAIT
- CONTROLS
- ELEGANT
- FLASH
- GENESIS
- JSPEC
- JUPYTER
- MAD-X
- OPAL
- OPENMC
- RADIA
- SHADOW
- SRW
- WARP PBA
- WARP VND
- ZGOUBI

Sirepo-Bluesky

https://github.com/NSLS-II/sirepo-bluesky
https://nsls-ii.github.io/sirepo-bluesky
https://doi.org/10.1117/12.2678030

- Available on **conda-forge** and **PyPI**
- Support of the **SRW**, **Shadow3**, and **MAD-X** applications in Sirepo
- Simulations are performed on a Sirepo server (a VM, Docker container, or HPC resources)
- Communication is done over HTTP(s) with Sirepo REST API
- Watchpoints or other Sirepo "reports" are wrapped into dedicated "detector" Ophyd objects
- All other optical elements are wrapped into Ophyd's **`Device`**s with **`Signal`**s corresponding to individual parameters in Sirepo
- The exchange format is JSON
- List of predefined simulations in Sirepo: https://nsls-ii.github.io/sirepo-bluesky/simulations.html



List of predefined simulations in Sirepo

Below is a list of custom/predefined simulations available when one starts Sirepo following the Sirepo startup instructions, that are currently used for tests and demos.

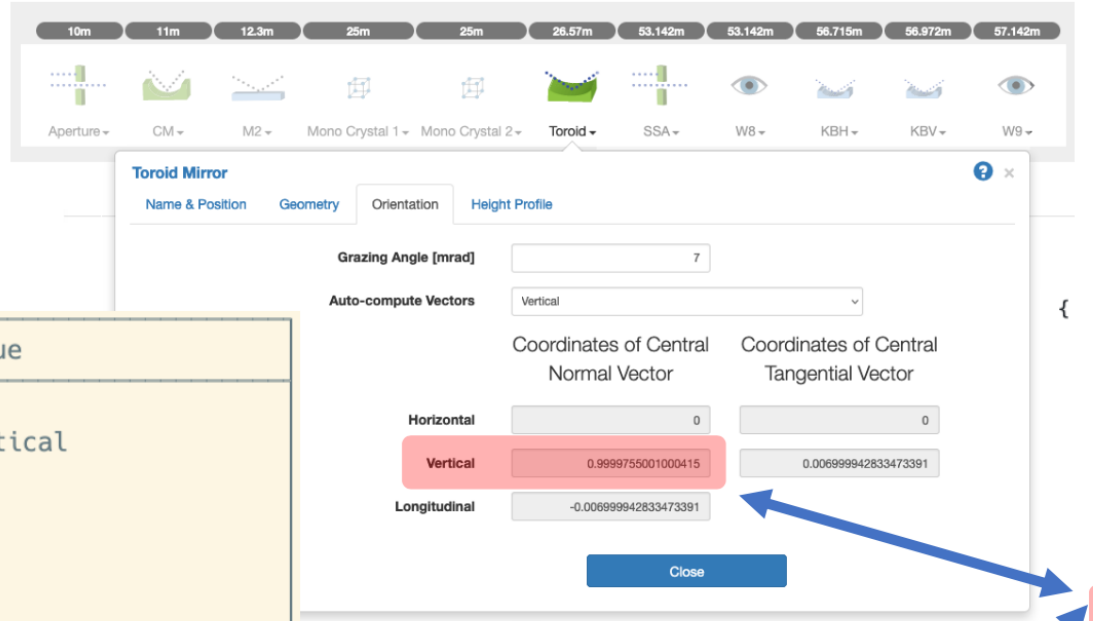
SRW

SRW simulations used for testing

Simulation ID	Description
00000000	Young's Double Slit Experiment
00000001	basic
00000002	TES
00000003	PD ARI-RIXS 250eV_JulyReviewVersion oc
00000004	PD ARI-RIXS 400eV (tuned) oc
00000005	PD ARI-ARPES 150eV JulyReviewVersion oc
00000006	PD ARI-ARPES 250eV JulyReviewVersion oc
00000007	SXN_PD_lowE_250eV
00000008	SXN_PD_medE_1000eV
00000009	SXN_PD_highE_2000eV

Sirepo-Bluesky: linking Sirepo with Ophyd

Sirepo Widget



Ophyd Device Representation

Ophyd object/component	Value
toroid_apertureShape	r
toroid_autocomputeVectors	vertical
toroid_grazingAngle	7
toroid_heightAmplification	1
toroid_heightProfileFile	
toroid_horizontalPosition	0
toroid_id	6
toroid_normalVectorX	0
toroid_normalVectorY	0.9999755001000415
toroid_normalVectorZ	-0.006999942833473391
toroid_orientation	y
toroid_sagittalRadius	0.186
toroid_sagittalSize	0.08
toroid_tangentialRadius	24500
toroid_tangentialSize	0.96
toroid_tangentialVectorX	0
toroid_tangentialVectorY	0.006999942833473391
toroid_title	Toroid
toroid_type	toroidalMirror
toroid_verticalPosition	0
toroid_element_position	26.57

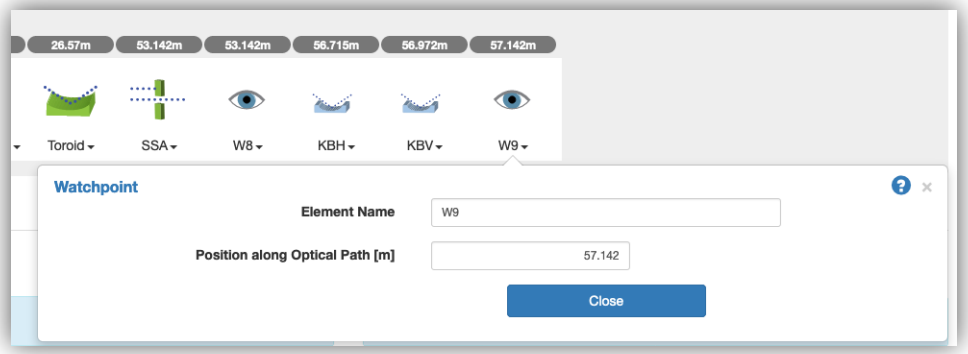
Sirepo JSON

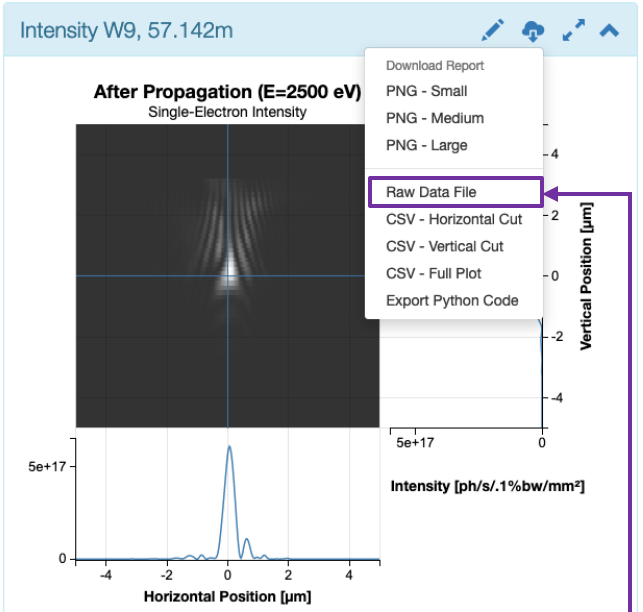

```
{
    "apertureShape": "r",
    "autocomputeVectors": "vertical",
    "grazingAngle": 7,
    "heightAmplification": 1,
    "heightProfileFile": "",
    "horizontalPosition": 0,
    "id": 6,
    "normalVectorX": 0,
    "normalVectorY": 0.9999755001000415,
    "normalVectorZ": -0.006999942833473391,
    "orientation": "y",
    "position": 26.57,
    "sagittalRadius": 0.186,
    "sagittalSize": 0.08,
    "tangentialRadius": 24500,
    "tangentialSize": 0.96,
    "tangentialVectorX": 0,
    "tangentialVectorY": 0.006999942833473391,
    "title": "Toroid",
    "type": "toroidalMirror",
    "verticalPosition": 0
},
```

Sirepo-Bluesky: Watchpoint ‘detector’

```python
class SirepoWatchpoint(DeviceWithJSONData):
    image = Cpt(ExternalFileReference, kind="normal")
    shape = Cpt(Signal)
    mean = Cpt(Signal, kind="hinted")
    photon_energy = Cpt(Signal, kind="normal")
    horizontal_extent = Cpt(Signal)
    vertical_extent = Cpt(Signal)

    def __init__(
        self,
        *args,
        root_dir="/tmp/sirepo-bluesky-data",
        assets_dir=None,
        result_file=None,
        **kwargs,
    ):
```

```python
def trigger(self, *args, **kwargs):
    logger.debug(f"Custom trigger for {self.name}")

    date = datetime.datetime.now()
    self._assets_dir = date.strftime("%Y/%m/%d")
    self._result_file = f"{new_uid()}.dat"

    self._resource_document, self._datum_factory, _ = compose_resource(
        start={"uid": "needed for compose_resource() but will be discarded"},
        spec=self.connection.data["simulationType"],
        root=self._root_dir,
        resource_path=str(Path(self._assets_dir) / Path(self._result_file)),
        resource_kwargs={},
    )
    # now discard the start uid, a real one will be added later
    self._resource_document.pop("run_start")
    self._asset_docs_cache.append(("resource", self._resource_document))

    sim_result_file = str(
        Path(self._resource_document["root"]) / Path(self._resource_document["resource_path"])
    )

    self.connection.data["report"] = f"watchpointReport{self.id._sirepo_dict['id']}"

    _, duration = self.connection.run_simulation()
    self.duration.put(duration)

    datafile = self.connection.get_datafile(file_index=-1)

    with open(sim_result_file, "wb") as f:
        f.write(datafile)
```





start simulation → Start New Simulation

file saving logic

https://github.com/NSLS-II/sirepo-bluesky/blob/main/sirepo_bluesky/sirepo_ophyd.py

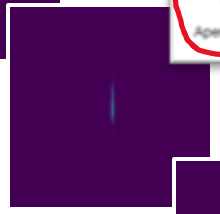
Example of a Bluesky run with Sirepo

```
from sirepo_bluesky import prepare_re_env

%run -i $prepare_re_env.__file__

from sirepo_bluesky.sirepo_bluesky import SirepoBluesky
from sirepo_bluesky.sirepo_ophyd import create_classes

connection = SirepoBluesky("http://localhost:8000")

data, schema = connection.auth("srw", sim_id="00000002")
classes, objects = create_classes(connection=connection)
globals().update(**objects)
```

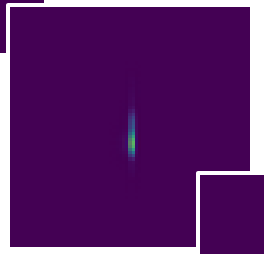
```
aperture.horizontalSize.kind = "hinted"
w9.duration.kind = "hinted"

(uid,) = RE(bp.scan([w9], aperture.horizontalSize, 0, 2, 6))

hdr = db[uid]
tbl = hdr.table(fill=True)
print(tbl)

w9_image = np.array(list(hdr.data("w9_image")))
```

```
Transient Scan ID: 1     Time: 2023-08-17 06:58:23
Persistent Unique Scan ID: '20ea37a4-c24c-4b8c-8c69-e548cc14b806'
New stream: 'primary'
+-----------+------------+-------------------------+-------------+-------------+
|   seq_num |       time | aperture_horizontalSize | w9_duration |     w9_flux |
+-----------+------------+-------------------------+-------------+-------------+
|         1 | 06:58:53.0 |                   0.000 |      29.158 |       0.000 |
|         2 | 06:59:20.5 |                   0.400 |      27.286 | 185263698905|
|         3 | 06:59:48.0 |                   0.800 |      27.299 | 875180761407|
|         4 | 07:00:15.5 |                   1.200 |      27.301 | 283521538357|
|         5 | 07:00:43.0 |                   1.600 |      27.295 | 679062036042|
|         6 | 07:01:10.5 |                   2.000 |      27.303 | 139860155217|
+-----------+------------+-------------------------+-------------+-------------+
generator scan ['20ea37a4'] (scan num: 1)
```

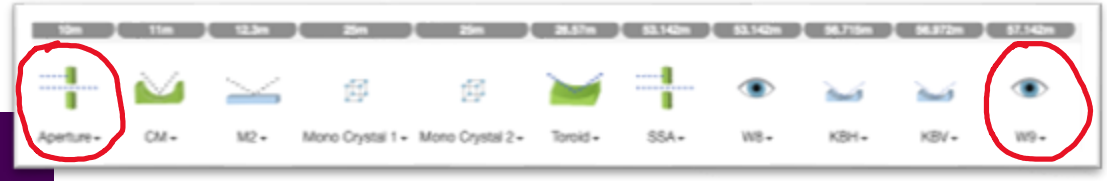
1



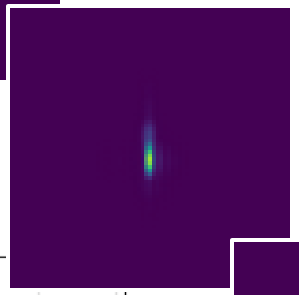
2

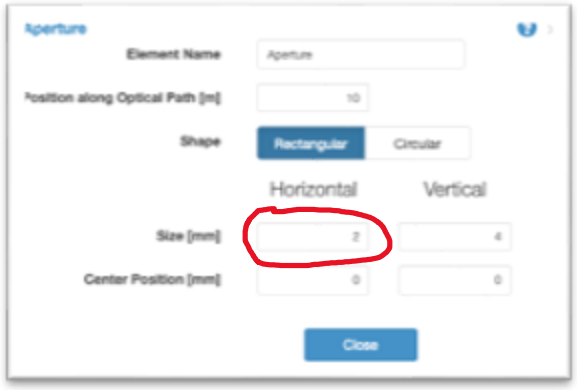


3

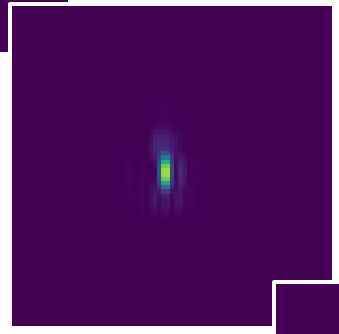


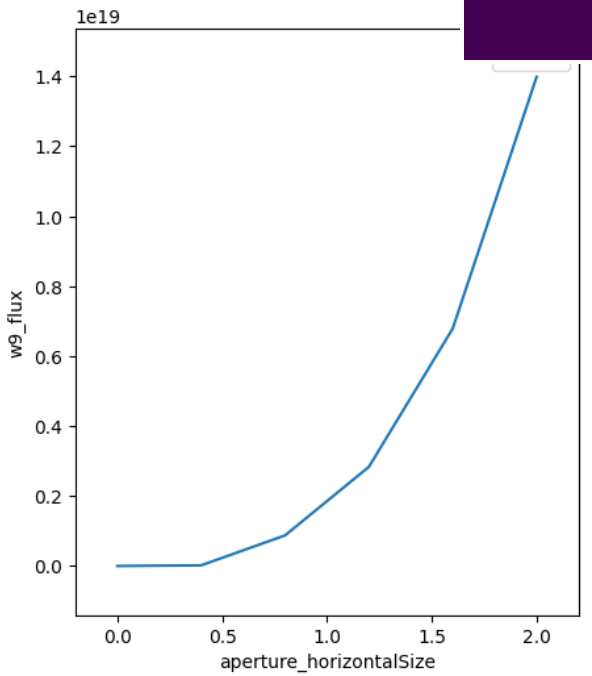
4



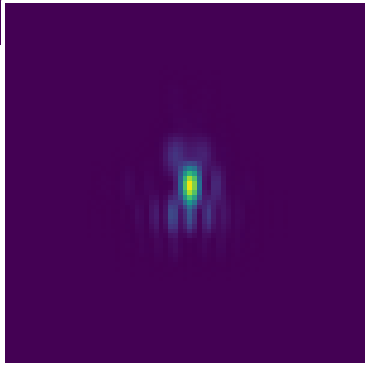


5



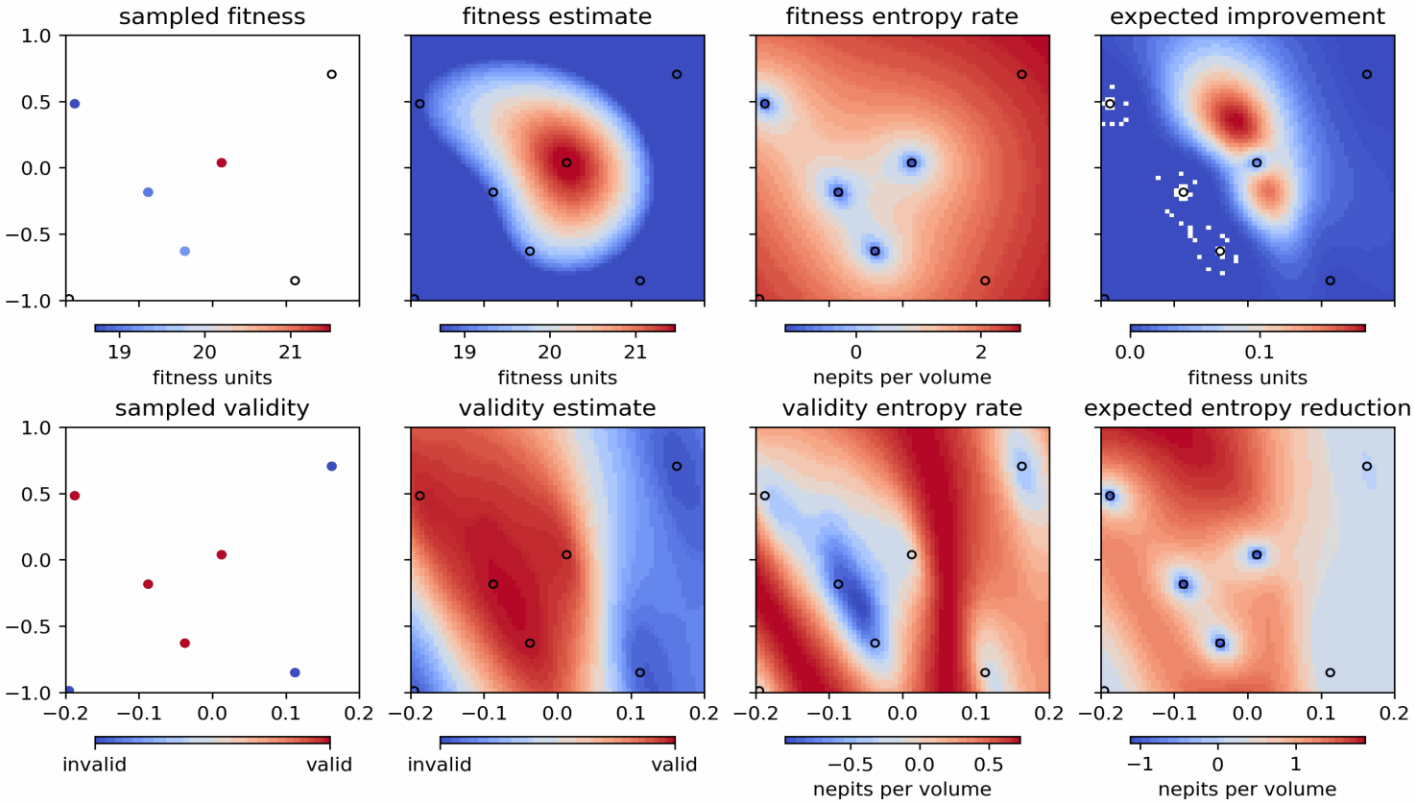


6



Beamline Optimization package: *blop*

https://github.com/NSLS-II/blop
https://nsls-ii.github.io/blop/
https://pypi.org/project/bloptools/



PyTorch

GPyTorch

BoTorch

Interactive Gaussian Processes (GP) exploration

https://distill.pub/2019/visual-exploration-gaussian-processes/

A Visual Exploration of Gaussian Processes

How to turn a collection of small building blocks into a versatile tool for solving regression problems.

Regression is used to find a function **(line)** that represents a set of data points as closely as possible

y = 0

A **Gaussian process** is a probabilistic method that gives a confidence (shaded) for the predicted function

(click to enable)

AUTHORS	AFFILIATIONS	PUBLISHED	DOI
Jochen Görtler	University of Konstanz	April 2, 2019	10.23915/distill.00017
Rebecca Kehlbeck	University of Konstanz		
Oliver Deussen	University of Konstanz		

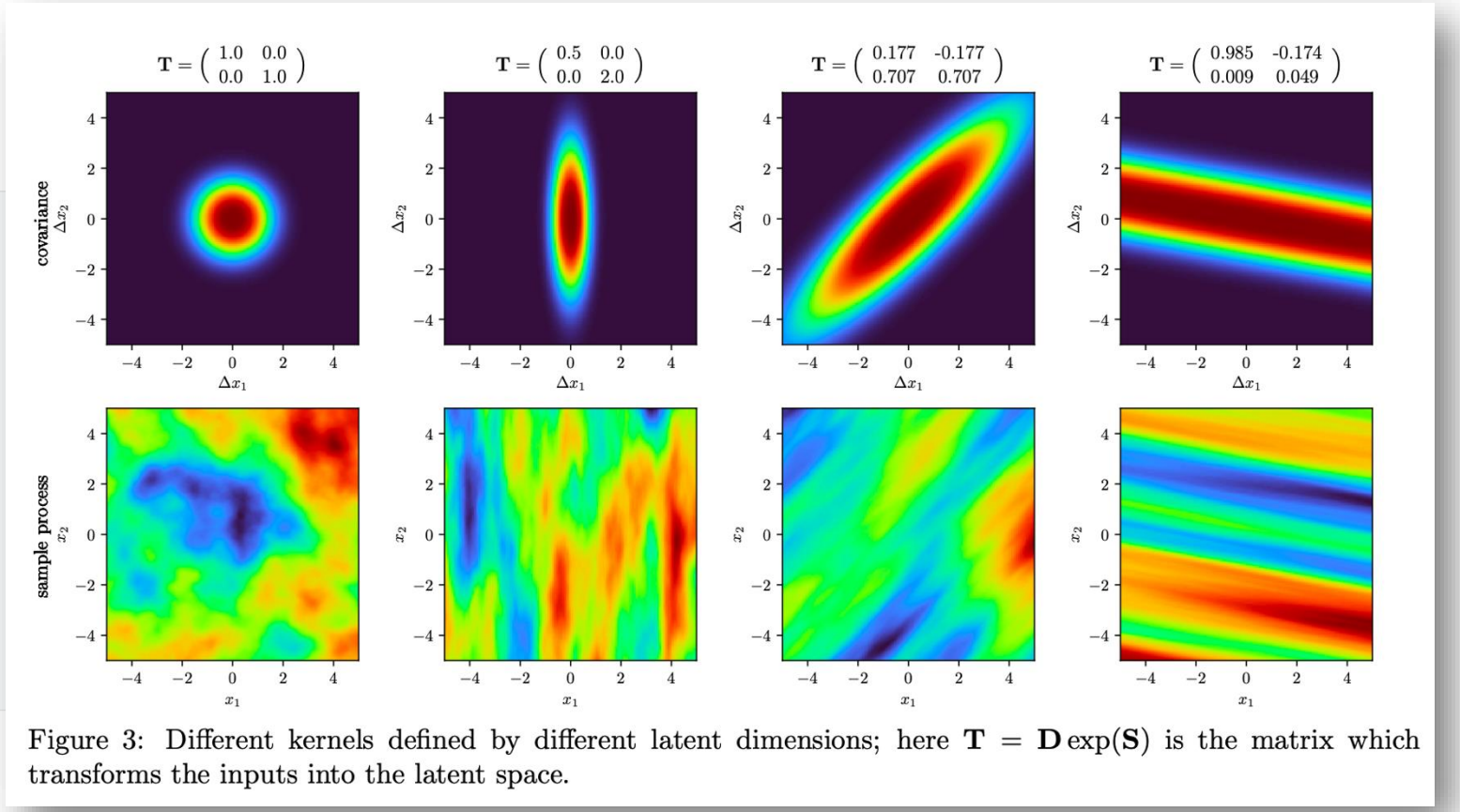


Figure 3: Different kernels defined by different latent dimensions; here $\mathbf{T} = \mathbf{D}\exp(\mathbf{S})$ is the matrix which transforms the inputs into the latent space.

Addressing multi-objective & multi-parameter optimization

Welcome a four-handed scientist!

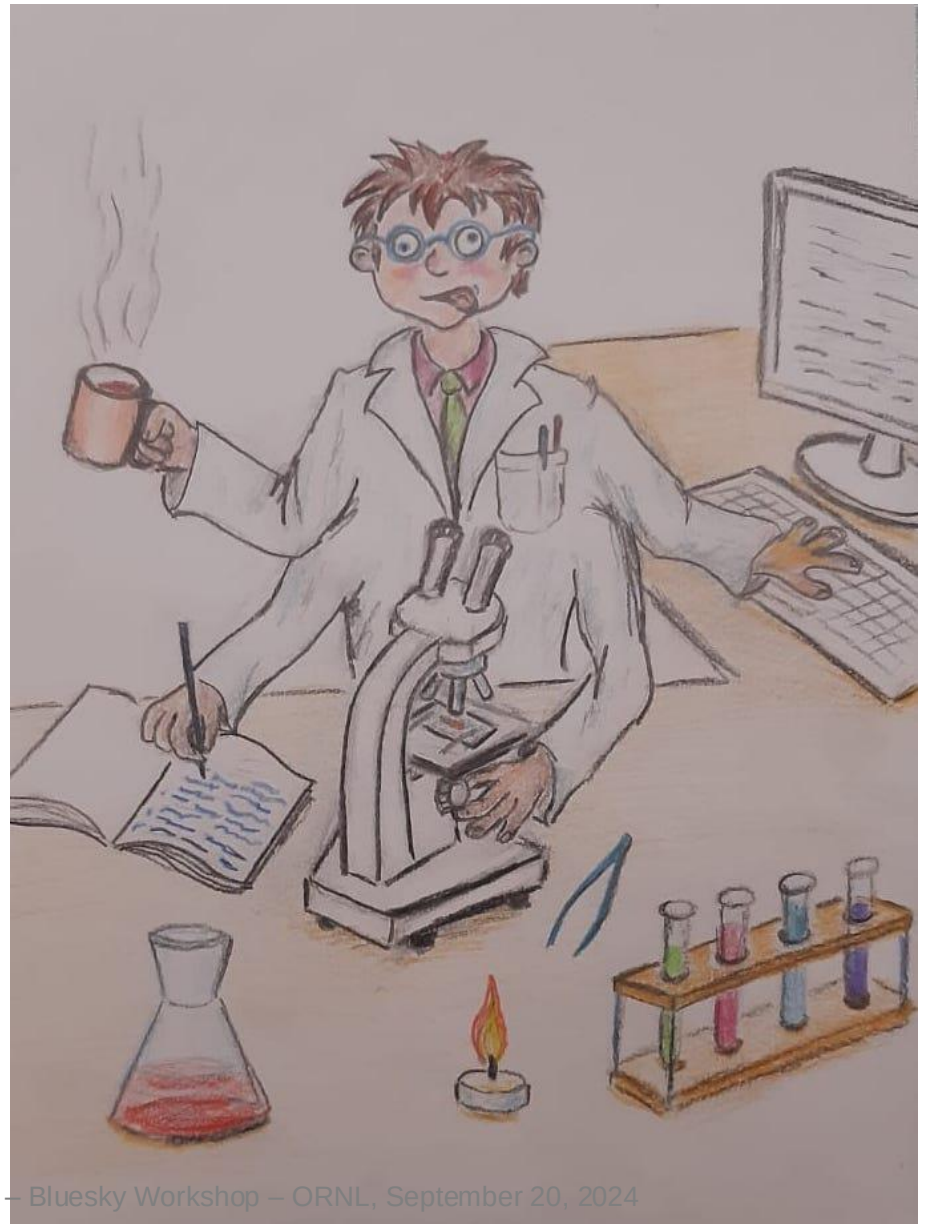
<u>Objectives:</u>

- Measure raw data with hardware
- Record metadata
- Process raw data on computer
- Reward: drink coffee!

<u>Dimensions/parameters:</u>

- 4 hands

Scales up to ~20 dimensions



Credit: Dmitry Rakitin (stable diffusion failed!)

Optimization test functions

Himmelblau's function

5 languages

Article Talk | Read Edit View history Tools

From Wikipedia, the free encyclopedia

In mathematical optimization, **Himmelblau's function** is a multi-modal function, used to test the performance of optimization algorithms. The function is defined by:

$$f(x, y) = (x^2 + y - 11)^2 + (x + y^2 - 7)^2.$$

It has one local maximum at $x = -0.270845$ and $y = -0.923039$ where $f(x, y) = 181.617$, and four identical local minima:

- $f(3.0, 2.0) = 0.0,$
- $f(-2.805118, 3.131312) = 0.0,$
- $f(-3.779310, -3.283186) = 0.0,$
- $f(3.584428, -1.848126) = 0.0.$

The locations of all the minima can be found analytically. However, because they are roots of cubic polynomials, when written in terms of radicals, the expressions are somewhat complicated. [*citation needed*]

The function is named after David Mautner Himmelblau (1924–2011), who introduced it.[1]

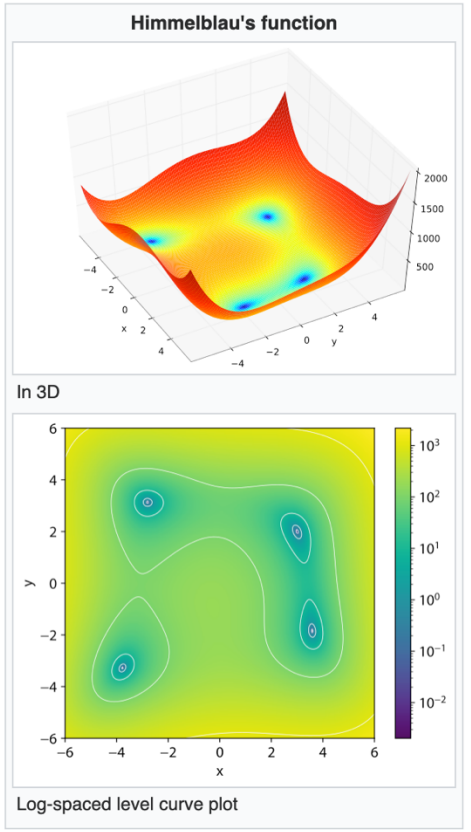
See also [edit]

- Test functions for optimization

References [edit]

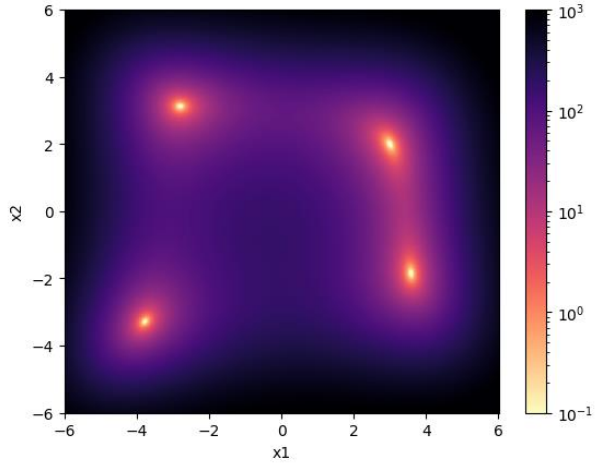
1. ^ Himmelblau, D. (1972). *Applied Nonlinear Programming*. McGraw-Hill. ISBN 0-07-028921-2.

This mathematical analysis–related article is a stub. You can help Wikipedia by expanding it.



https://en.wikipedia.org/wiki/Himmelblau's_function

Optimizing with *blop*

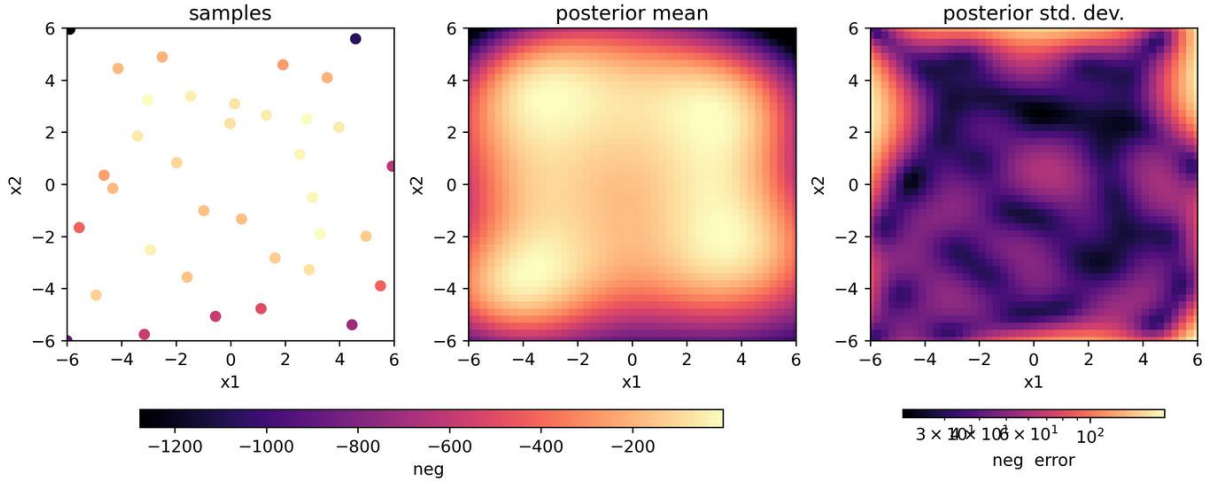


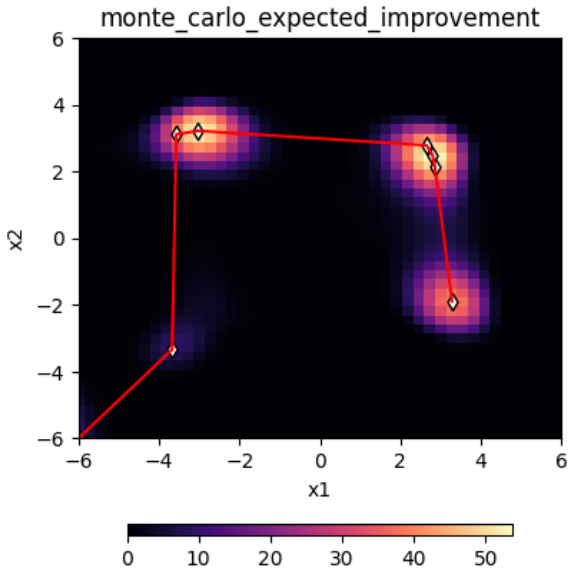

```
agent.all_acq_funcs
```

```
EXPECTED IMPROVEMENT (identifiers: ['ei', 'expected_improvement'])
-> The expected value of max(f(x) - \nu, 0), where \nu is the current maximum.

MONTE CARLO EXPECTED IMPROVEMENT (identifiers: ['qei', 'monte_carlo_expected_improvement'
-> The expected value of max(f(x) - \nu, 0), where \nu is the current maximum.

EXPECTED MEAN (identifiers: ['em', 'expected_mean'])
-> The expected value at each input.

MONTE CARLO EXPECTED MEAN (identifiers: ['qem', 'monte_carlo_e
-> The expected value at each input.

LOWER BOUND MAX VALUE ENTROPY (identifiers: ['lbmve', 'lbmes',
-> Max entropy search, basically

NOISY EXPECTED HYPERVOLUME IMPROVEMENT (identifiers: ['nehvi',
-> It's like a big box. How big is the box?

PROBABILITY OF IMPROVEMENT (identifiers: ['pi', 'probability_o
-> The probability that this input improves on the current max

MONTE CARLO PROBABILITY OF IMPROVEMENT (identifiers: ['qpi', '
-> The probability that this input improves on the current max

RANDOM (identifiers: ['r', 'random'])
-> Uniformly-sampled random points.

QUASI-RANDOM (identifiers: ['qr', 'quasi-random'])
-> Sobol-sampled quasi-random points.

GRID SCAN (identifiers: ['g', 'gs', 'grid'])
-> A grid scan over the parameters.

UPPER CONFIDENCE BOUND (identifiers: ['ucb', 'upper_confidence_
-> The expected value, plus some multiple of the uncertainty (

MONTE CARLO UPPER CONFIDENCE BOUND (identifiers: ['qucb', 'mont
-> The expected value, plus some multiple of the uncertainty (
```

```
agent.plot_objectives()
print(agent.best)
```

```
x1                                          -3.03913
x2                                          3.224808
himmelblau                                  2.26465
time                        2023-11-08 23:56:04.029840231
acq_func                 monte_carlo_expected_improvement
himmelblau_fitness                          -2.26465
Name: 33, dtype: object
```





Google OR-Tools

Optimum Route finding
(N-D traveling salesman)

Click me: https://nsls-ii.github.io/blop/tutorials/himmelblau.html

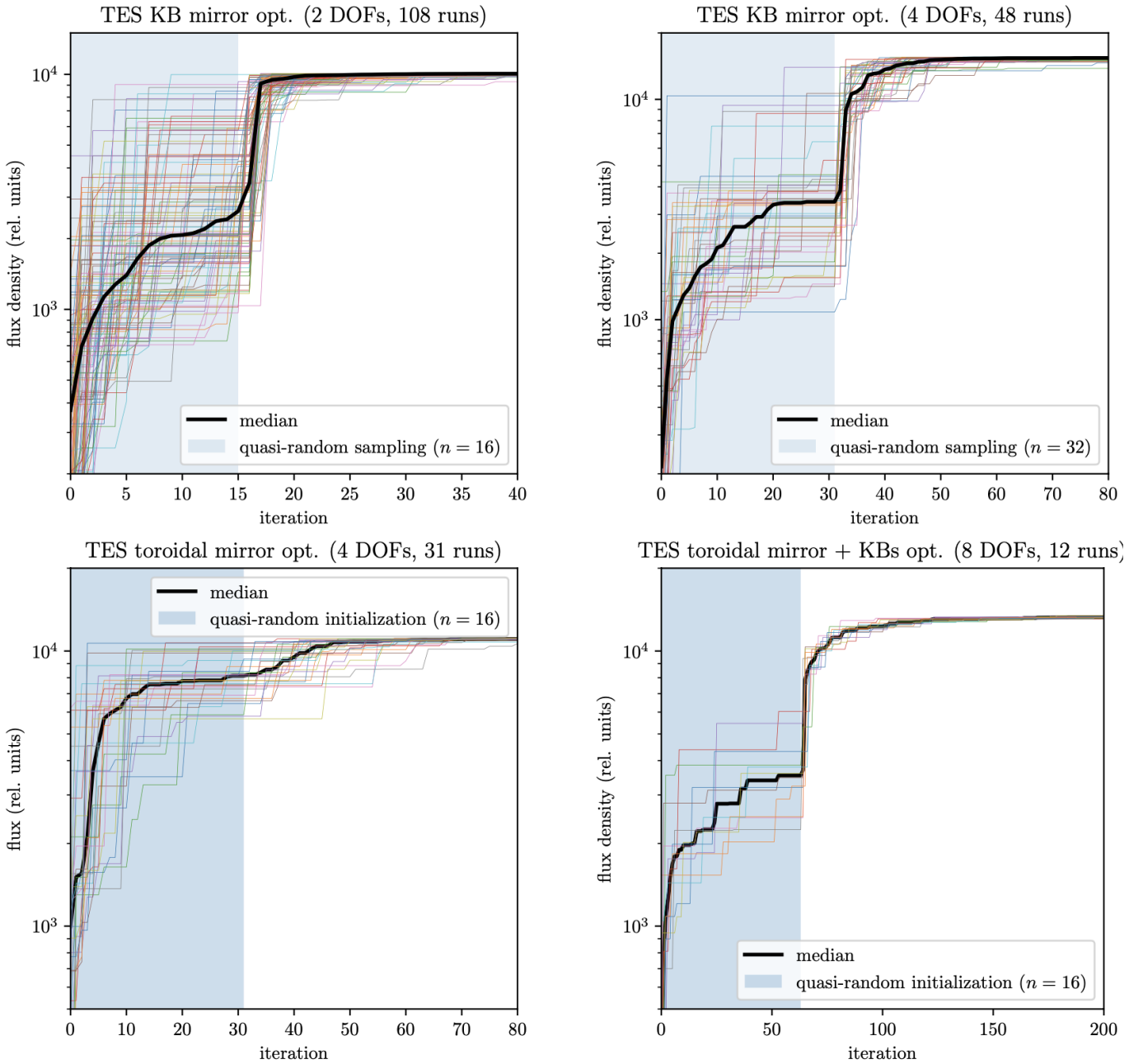


Figure 6: Four optimization problems for the TES beamline, simulating using a SHADOW (ray-tracing) backend. *Top left:* KB mirror rotation (two dimensions) *Top right:* KB mirror rotation + translation (four dimensions) *Bottom left:* pre-SSA toroidal mirror (four dimensions) *Bottom right:* KBs + toroid (eight dimensions)

Presentation + Paper

5 October 2023

Latent Bayesian optimization for the autonomous alignment of synchrotron beamlines

T. W. Morris, Y. Du, M. Fedurin, A. C. Giles, P. Moeller, B. Nash, M. Rakitin, B. Romasky, A. L. Walter, N. Wilson, A. Wojdyla

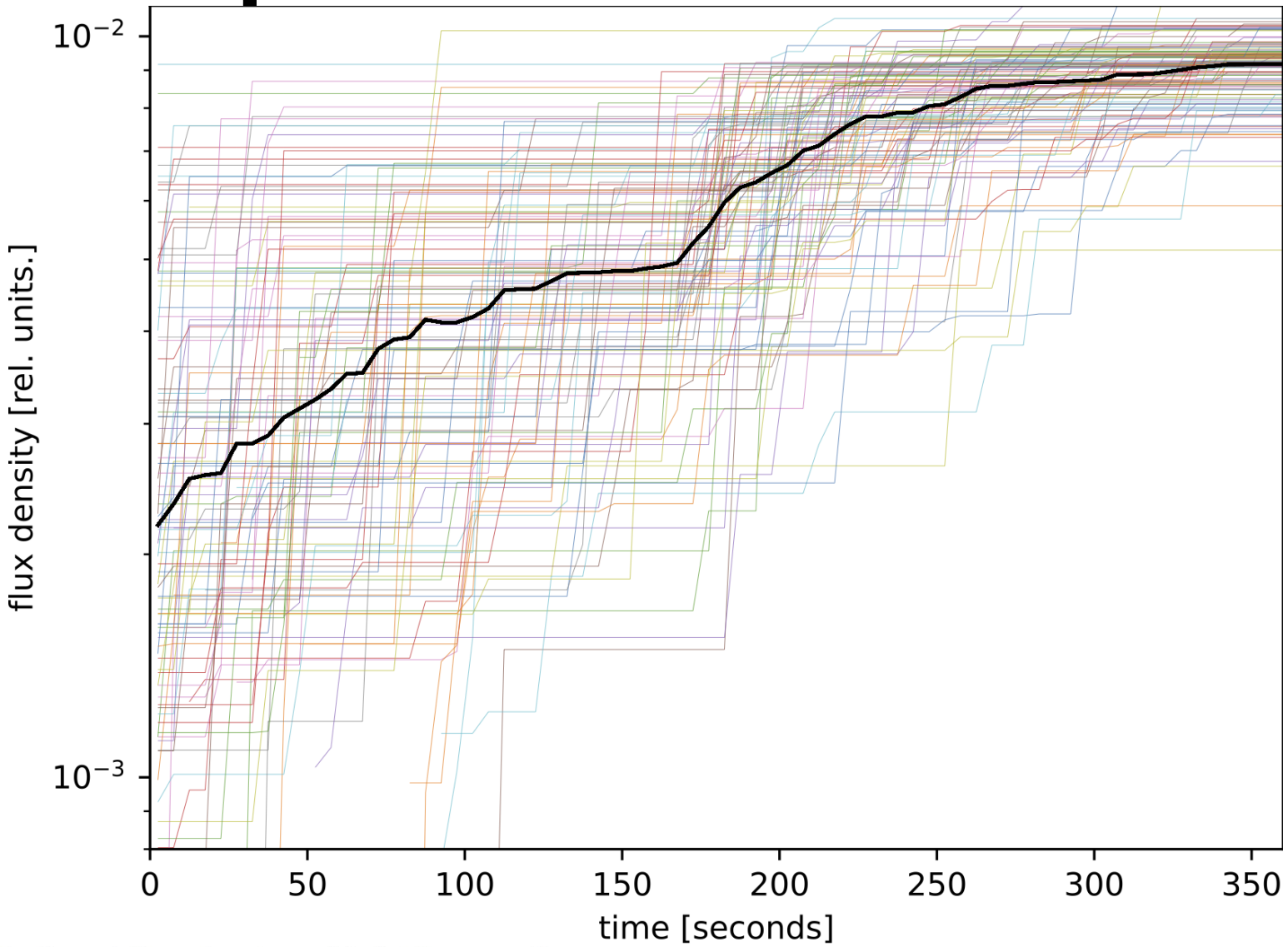
Author Affiliations +

Proceedings Volume 12697, Advances in Computational Methods for X-Ray Optics VI; 126970B (2023)
https://doi.org/10.1117/12.2677895
Event: SPIE Optical Engineering + Applications, 2023, San Diego, California, United States

https://doi.org/10.1117/12.2677895

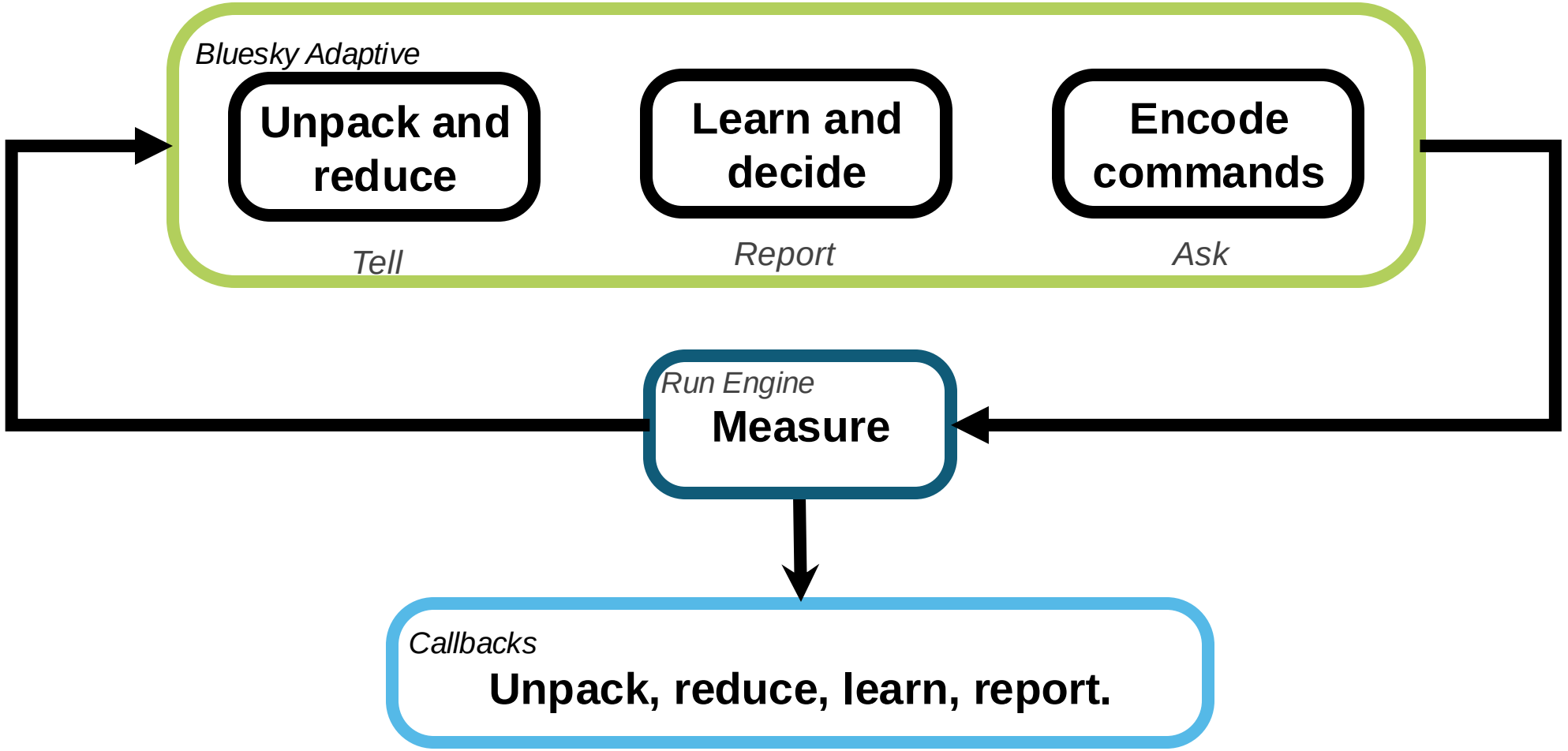
JSR Paper (accepted in Sept. 2024)

6D optimization at NSLS-II TES beamline



Experiment: Sept. 2023
Number of runs: 120

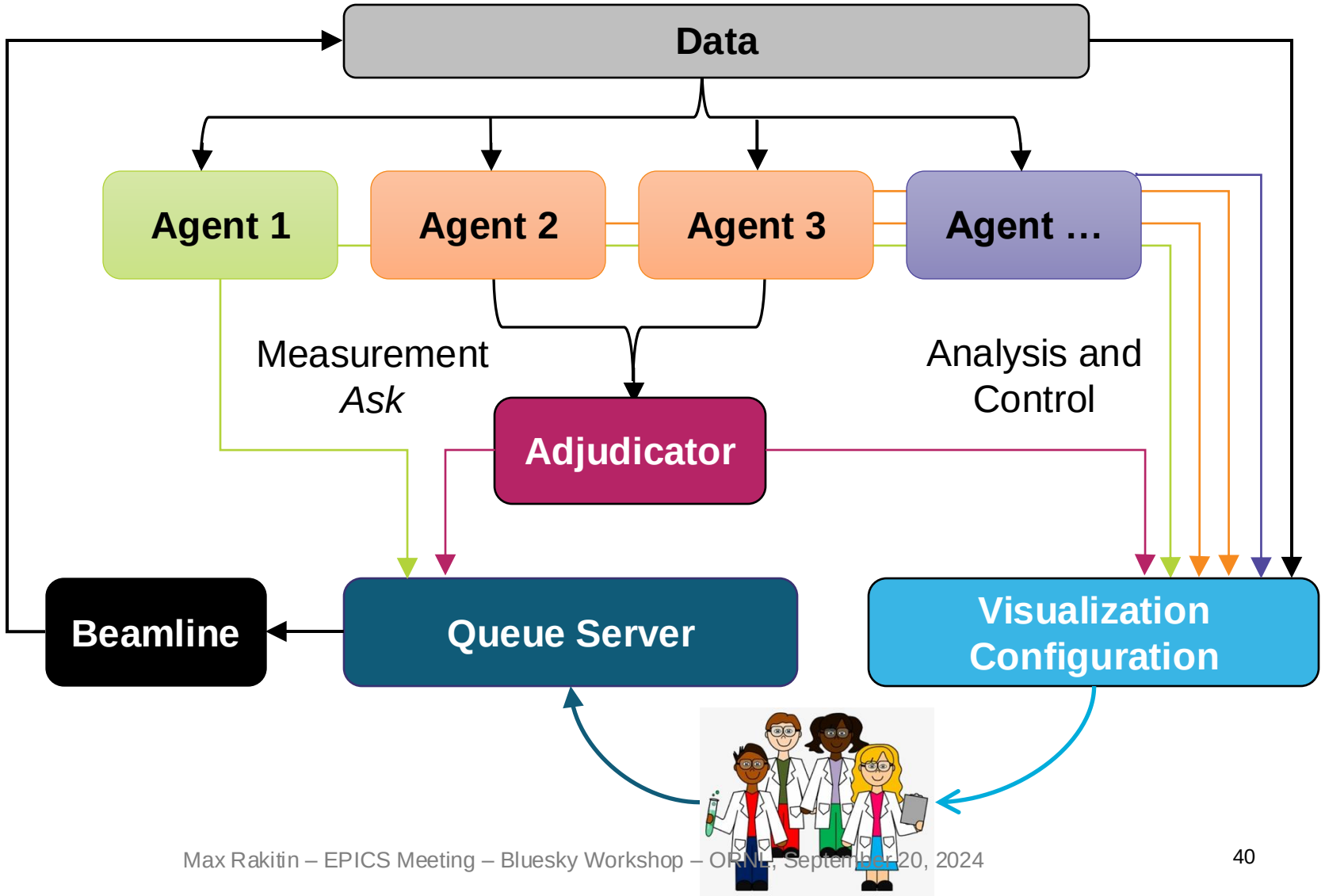
Agents are enabled by Bluesky Adaptive and Run Engine callbacks.



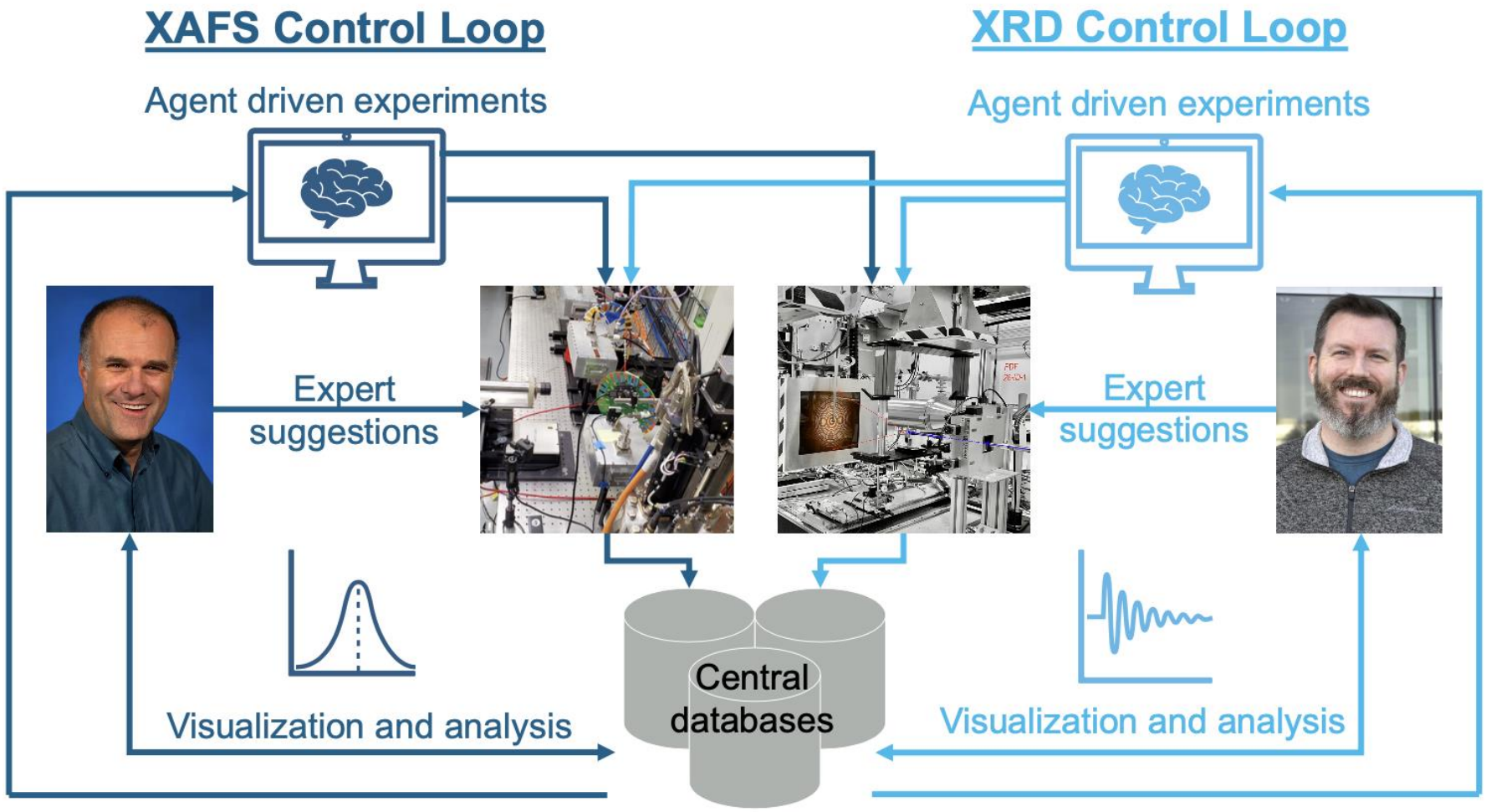
blueskyproject.io/**bluesky-adaptive/**

Empowering the collaboration of many agents and scientists.

- Queue Server acts as control hub
- Users interrogate agents and data
- Adjudicators mediate agent requests



Experimental orchestration of multiple agents and beamlines *in concert*.



NeurIPS (2022). arXiv:2301.09177

Join the team!

Data Acquisition & Detectors Group – Bluesky Developer position:
https://jobs.bnl.gov/job/upton/research-software-engineer/3437/68796372608

or **jobs.bnl.gov** → Search “DSSI” for more openings

Live Demo!

https://github.com/mrakitin/profile_epics_meeting_2024

Thank you!