National Synchrotron Light Source II

Tiled: A Service for Structured Data Access

Daniel Allan and Bluesky Project Contributors
Data Science and Systems Integration Program
National Synchrotron Light Source II, Brookhaven National Laboratory

U.S. DEPARTMENT OF ENERGY | Office of Science
BROOKHAVEN NATIONAL LABORATORY

Acknowledgements

- Bluesky development and maintenance is supported by contributors (too many to name) at:
  - NSLS-II
  - LCLS
  - APS
  - ALS
  - SSRL
  - Australian Synchrotron
  - Diamond Light Source
  - Various individual contributors
- Recent work on *data access* for data science is supported by
  - NSLS-II (**Thomas Caswell, Marcus Hanwell, Garrett Bischof, Stuart Campbell**)
  - DOE Basic Energy Sciences Grant *Integrated Platform for Multimodal Data Capture, Exploration and Discovery Driven by AI Tools* (P.I. **Eli Stavitski**)
  - ALS (**Dylan McReynolds**)

Bluesky is a multi-facility collaboration

- Light sources
  - **NSLS-II** (26 out of 28 beamlines)
  - The other four DOE Light Sources: **APS, LCLS, SSRL, ALS** all formally plan adoption, are at various stages of rolling it out
  - International: **Australian Synchrotron**, **Canadian Light Source**, **BESSY-II** (Berlin), **Pohang Light Source** (Korea); currently being closely evaluated by **Diamond Light Source** (UK)
- Some University research groups
  - **Electrical Engineering**: Lucas J. Koerner group at University of St. Thomas
  - **Chemistry**: Blaise Thompson and Kyle Sunden, Wright group at University of Wisconsin-Madison
- Other various folks who turn up in our support channels and GitHub

Obviously,
the *social* problem is much harder than the *technical* problem.

Collaborate and specialize. Share and differentiate.

Bluesky: Some Assembly Required...

...But it's easy to quickly build complex custom applications

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.
- **Use** the parts of Bluesky that work for you *a la carte*, **extend** them, or **replace** them.
- 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.
- The scientific Python community is an example of how this can work well.

One current focus:
Improving *data access* for data science
leveraging web standards and scientific computing standards

https://blueskyproject.io/tiled

Lineage

- Two of us created PIMS (Python Image Sequence) as grad students (2013)
- Databroker (BNL, 2015)
- Intake (Anaconda, 2017)
  - Contributed ~40 Pull Requests, focused on scaling and performance
  - Refactored Databroker on it
  - Very interested in the "intake-server"
- Attempt to productionalize intake-server led to Tiled
  - The requirements of a performant HTTP service are in tension with Intake's other commitments.
  - Now refactoring Databroker off of Intake and onto Tiled

Support search with extensible queries

```
catalog.search(
    RawMongo({
        “motors”: {“$in”: [“x”, “y”]},  # scanning either x or y
        “temperature” {“$lt”: 300},  # temperature less than 300
        “sample.element”: “Ni”,
    })
).search(
    TimeRange(since="2020-01", until="2020-03")
)
```

Our users are largely still stuck under the tyranny of manually parsing `meta_data_in_37K_fname_005_NaCl_cal.tif` !

Standardize on *structures* not *formats*

- There will never be One Format to Rule Them All.
- There is not even always one optimal format for a given use case!
- But many things understand these structures:
  - **Strided arrays, dense and sparse** (e.g. numpy arrays)
  - **Ragged arrays** (Awkward Array)
  - **Tables** (e.g. pandas DataFrames, Apache Arrow tables)
  - **Hierarchical structures** of these (e.g. HDF5, NetCDF, NeXus, zarr, directories)
- Not tied to any particular format or programming language

Separate how the data is *stored* from how it is *accessed*

- Provide data over a (HTTP) **service**
- Freedom to write and store data however it makes sense for given hardware resources
- And change our mind later **without breaking the scientific analysis code**
- Let analysis code request it in **any appropriate format**, such as...

Tabular data as...

- Apache Arrow
  (pandas' blessed wire format)
  directly into **xarray**
  or **pandas** for data science

```
In [8]: c[22521]['primary']['data'][['dcm_energy', 'I0']]
Out[8]:
<xarray.Dataset>
Dimensions:     (time: 411)
Dimensions without coordinates: time
Data variables:
    dcm_energy  (time) float64 5.789e+03 5.799e+03 ... 6.726e+03 6.731e+03
    I0          (time) float64 83.65 83.7 83.7 83.63 ... 15.74 15.59 15.44 15.27

In [9]: c[22521]['primary']['data'][['dcm_energy', 'I0']].to_dataframe()
Out[9]:
      dcm_energy         I0
time
0     5789.197840  83.653961
1     5799.206692  83.704274
2     5809.221688  83.702556
3     5819.213169  83.626278
4     5829.210295  83.563962
...           ...        ...
406   6710.166344  15.909260
407   6715.417728  15.744356
408   6720.672014  15.593643
409   6725.959728  15.441166
410   6731.286749  15.271944

[411 rows x 2 columns]
```

- **CSV** or **Excel** (e.g. from commandline or web browser)

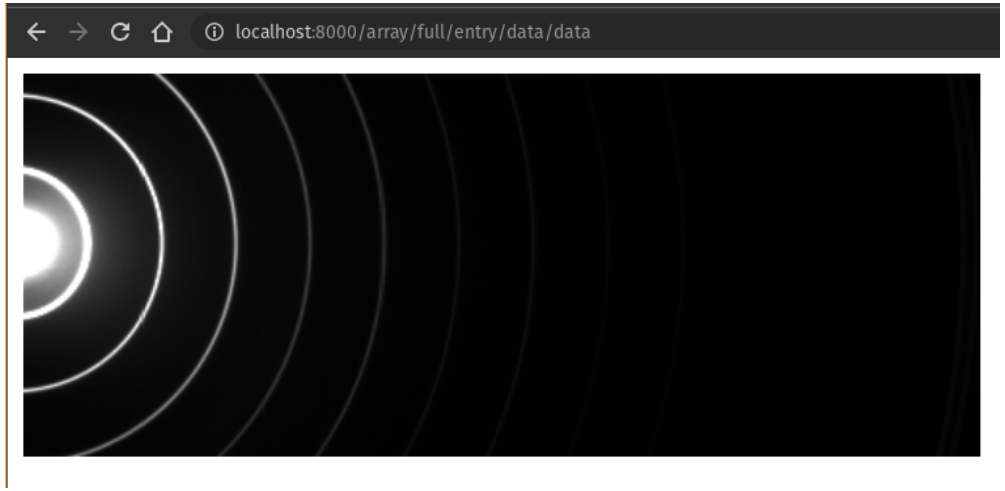
```
10:11 $ curl -H Accept:application/json "http://localhost:8000/data
time,dcm_energy,I0
1583209091.0413885,5789.197840063711,83.65396070480354
1583209097.041978,5799.206691542305,83.70427360534669
1583209098.670894,5809.221687763265,83.70255594253535
1583209103.687407,5819.2131686283365,83.6262778282166
1583209105.3131258,5829.210294797604,83.56396198272704
1583209106.9227643,5839.208674383871,83.52745981216428
1583209108.5199769,5849.208115721978,83.40440158843998
1583209110.1262672,5859.206999444012,83.20500516891484
```

localhost:8000/dataset/full/f8c83910-4ad

	dcm_energy	I0
time		
1.583209e+09	5789.197840	83.653961
1.583209e+09	5799.206692	83.704274
1.583209e+09	5809.221688	83.702556
1.583209e+09	5819.213169	83.626278
1.583209e+09	5829.210295	83.563962
1.583209e+09	5839.208674	83.527460
1.583209e+09	5849.208116	83.404402
1.583209e+09	5859.206999	83.205005
1.583209e+09	5869.206547	81.534167

Array data as...

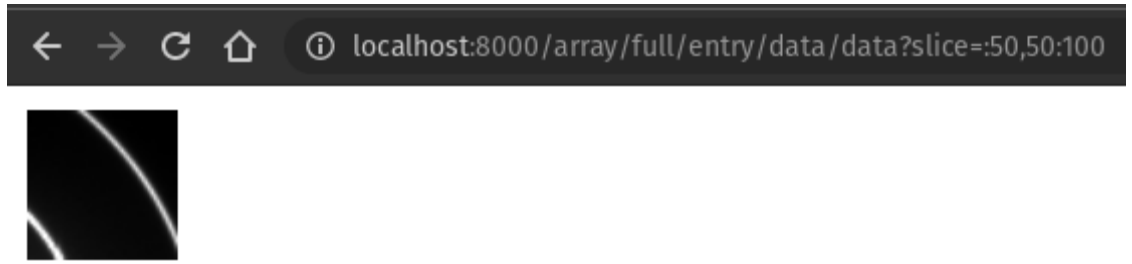
- Direct C-ordered buffer for **numpy**
- 2- or 3-dimensional image arrays at TIFF for **ImageJ**
- 2-dimensional image arrays as PNG for a **web applications**
- CSV (inefficient but universal)




```
In [19]: c['entry']['data']['data'][:]
Out[19]:
array([[473, 398, 432, ..., 156, 119, 114],
       [442, 423, 427, ..., 124, 112, 110],
       [447, 426, 419, ..., 146, 125,  96],
       ...,
       [285, 290, 303, ...,  87,  87,  92],
       [330, 306, 300, ...,  85,  94,  93],
       [398, 331, 349, ...,  91,  96, 105]], dtype=int32)
```

Read and transport only the data you need...

- A **slice** of an array
- A **subset** of a table's columns, and **partitions** of its rows

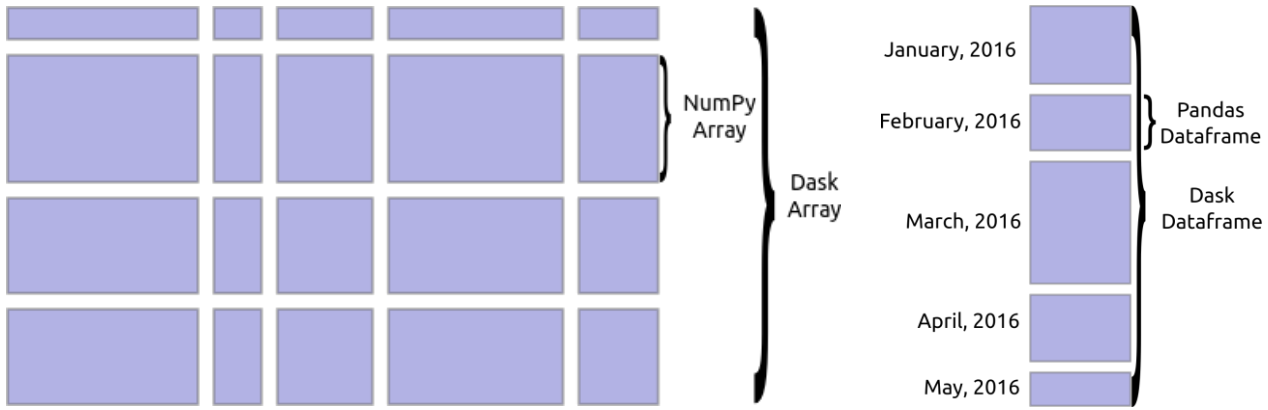


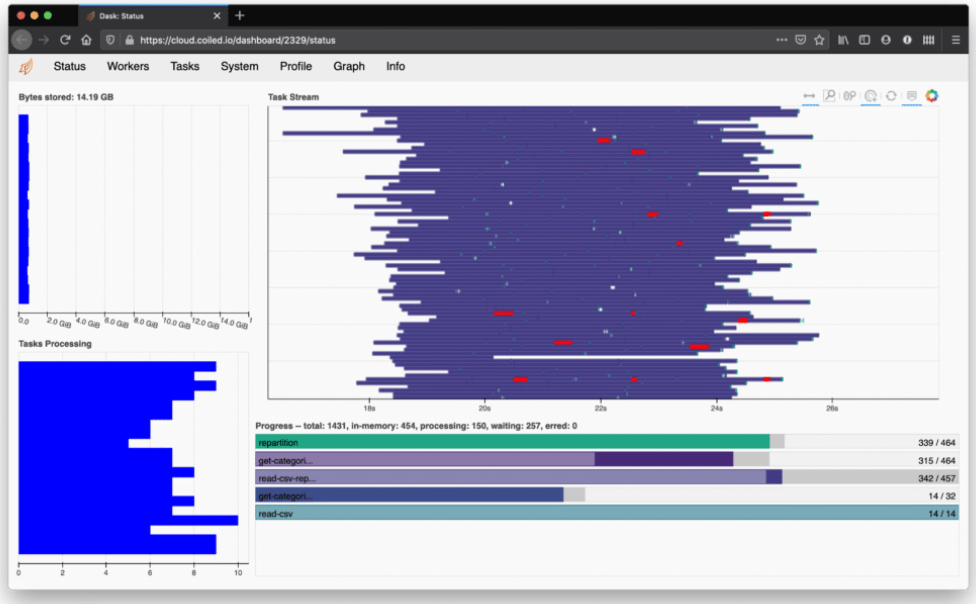

```
In [17]: c['entry']['data']['data'][:3, 5:10]
Out[17]:
array([[416, 411, 415, 440, 437],
       [427, 416, 414, 450, 412],
       [441, 396, 421, 441, 451]], dtype=int32)
```

Demonstrated using a Python client and in a normal web browser

Integrate naturally with popular data science tools

Chunks / partitions of arrays / tables **in parallel on demand** into dask.array / dask.dataframe and from there into, say, **scikit-learn**





The standard, useful things you can do with a *service*

- Modern web security standards
- Pluggable authentication
  - B.Y.O user management system (local system users, Globus, ORCID, etc.)
  - Just like JupyterHub
- Granular access control enforcement
- Service-side caching of "hot" datasets and resources
- Local client-side caching (just like a web browser's cache)
- Offline use (i.e. "airplane mode")

Status and Roadmap

- Fairly solid:
  - **Tree** structures of Nodes
  - **Metadata** (like HDF5 attrs) on each Node
  - **Array**, **dataframe, xarray** structures (Variable, DataArray, Dataset)
  - Integration with **dask** for deferred and parallel chunkwise access from Python
  - Validated server- and client-side **configuration**
  - **CLIs** for easy server startup and configuration inspection
  - Serialization in **many formats** --- user extensible
  - **Documentation!**

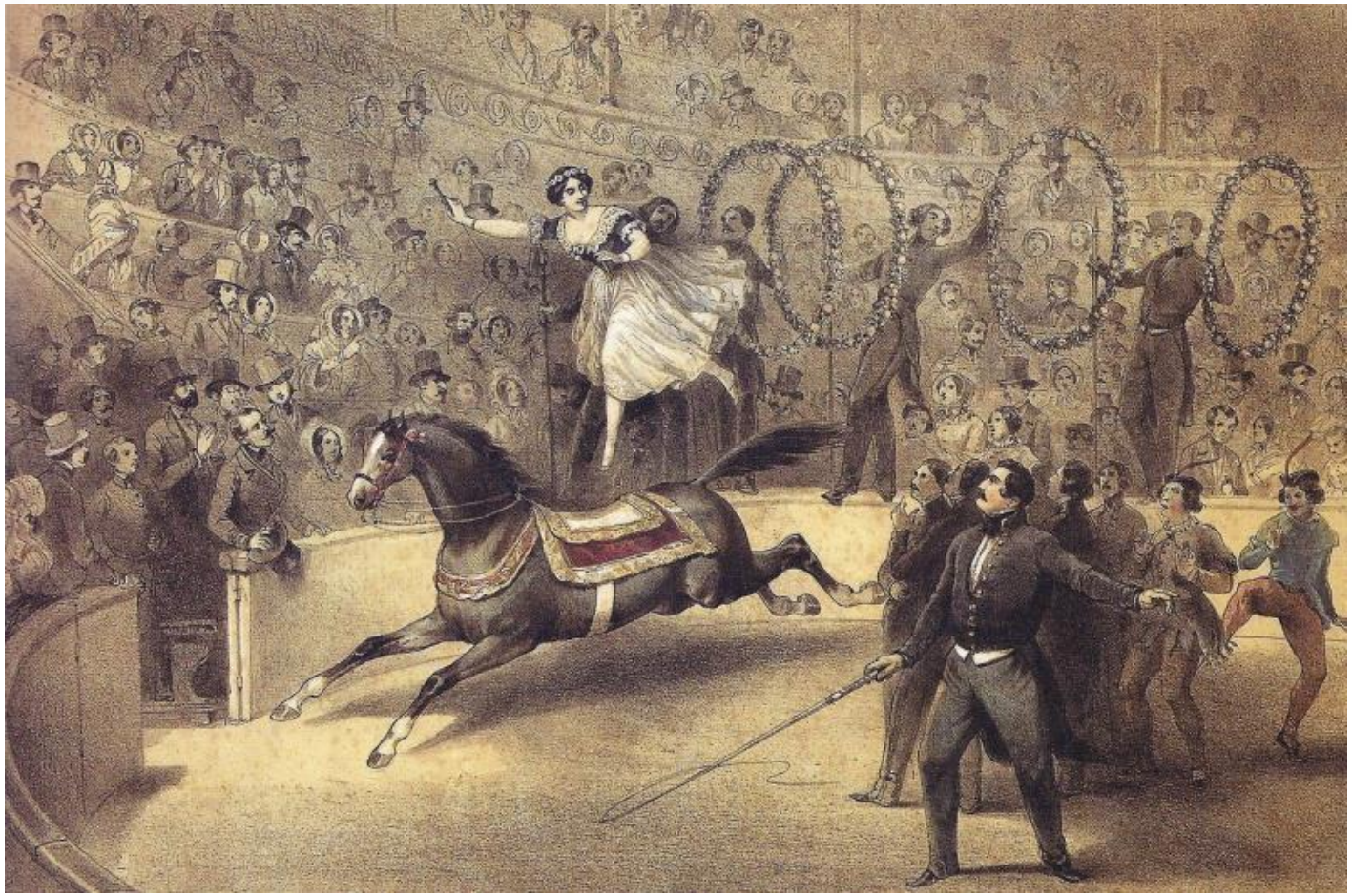
Status and Roadmap

- Prototyped, working:
  - (Optional) Simple shared-secret authentication for single-user usage
  - (Optional) **OAuth2** Authentication and Resource servers with **JWT** authentication tokens and "refresh" tokens ("**sliding sessions**")
  - **Horizontal scaling** with standard ASGI tools (uvicorn, gunicorn)
  - **Client-side caching** using HTTP standards (ETag, 304)
  - Dispatch to specialized clients based on **semantics** of structure (i.e. "not just *any* dataframe but a dataframe with columns that fit this specification known to the service and the client")

Status and Roadmap

- Dreamed of:
  - **Ragged** Arrays, integration with Awkward Array
  - **Sparse** Arrays
  - Image **pyramids** (downsampled views)
  - Fast paths to **batch requests for large nested structures**
  - **GraphQL** endpoint for more complex queries
  - Integration with high-performance storage format **Caterva**
  - A **Zarr** compatibility layer (i.e. /zarr/...)
  - Support **writing** via client. (Currently, data *can* be inserted, removed, or updated, but not those updates cannot be initiated via HTTP.)

Demo Time!



Discussion Points

- Are we missing any opportunities in the way that we are framing this?
- Are there other related work that we should know about?
- Will Awkward Array's structures fit neatly alongside the structures that Tiled currently supports? Or will they prompt a deeper rethinking?
- Are there high-performance wire formats we should care about beyond Arrow and Parquet?