Ophyd v2

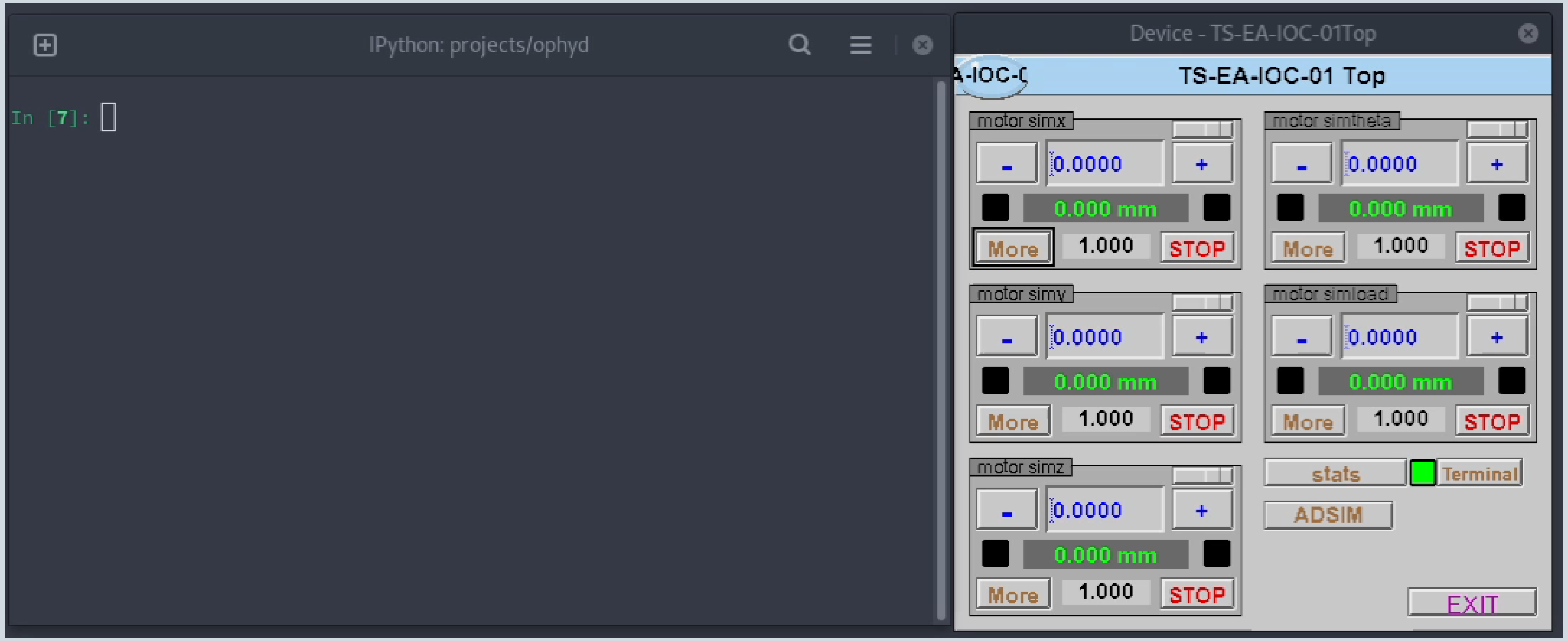
CALLUM FORRESTER, TOM COBB, DAN ALLEN, TOM CASWELL

Contents

Ophyd

Module in the Bluesky architecture

Hardware abstraction layer focused on data acquisition



```
In [7]: 
```

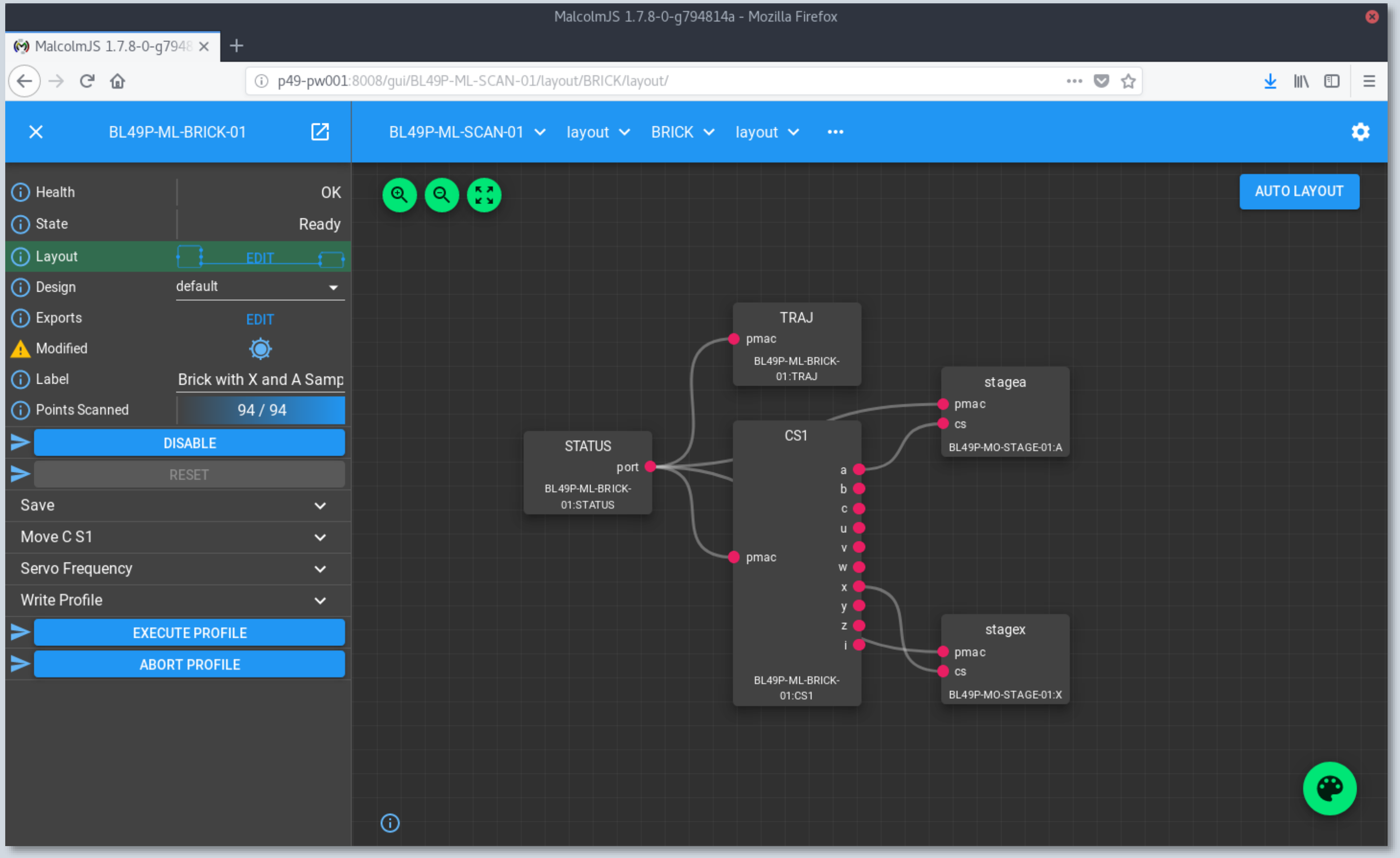
Malcolm

Fly scanning service

Provides map of hardware

Hierarchy of “devices”

Configuration UI



Health	OK
State	Ready
Layout	EDIT
Design	default
Exports	EDIT
Modified	
Label	Brick with X and A Samp
Points Scanned	94 / 94

Ophyd v2

Drop-in replacement for Ophyd as-is

Collaboration between

- NSLS-II
- Diamond Light Source

Incorporates lessons learnt from Ophyd and Malcolm

Key Design Concerns

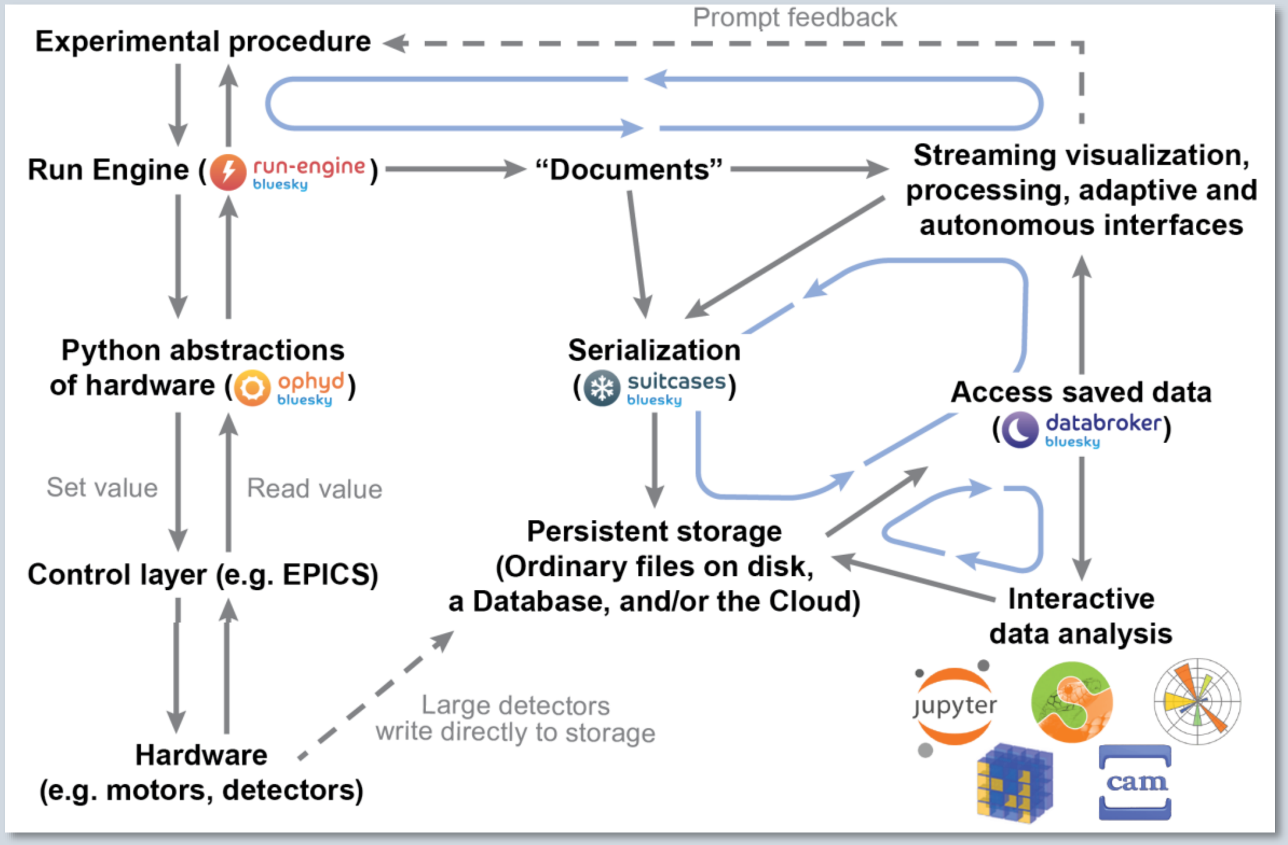
- Modularity
- Clear Migration Path
- Separation of logic and I/O
- Easy to implement hardware triggered scanning solutions

The Bluesky Collaboration

Analysis-driven Data Acquisition Libraries

Started at NSLS-II

Technical Steering Committee

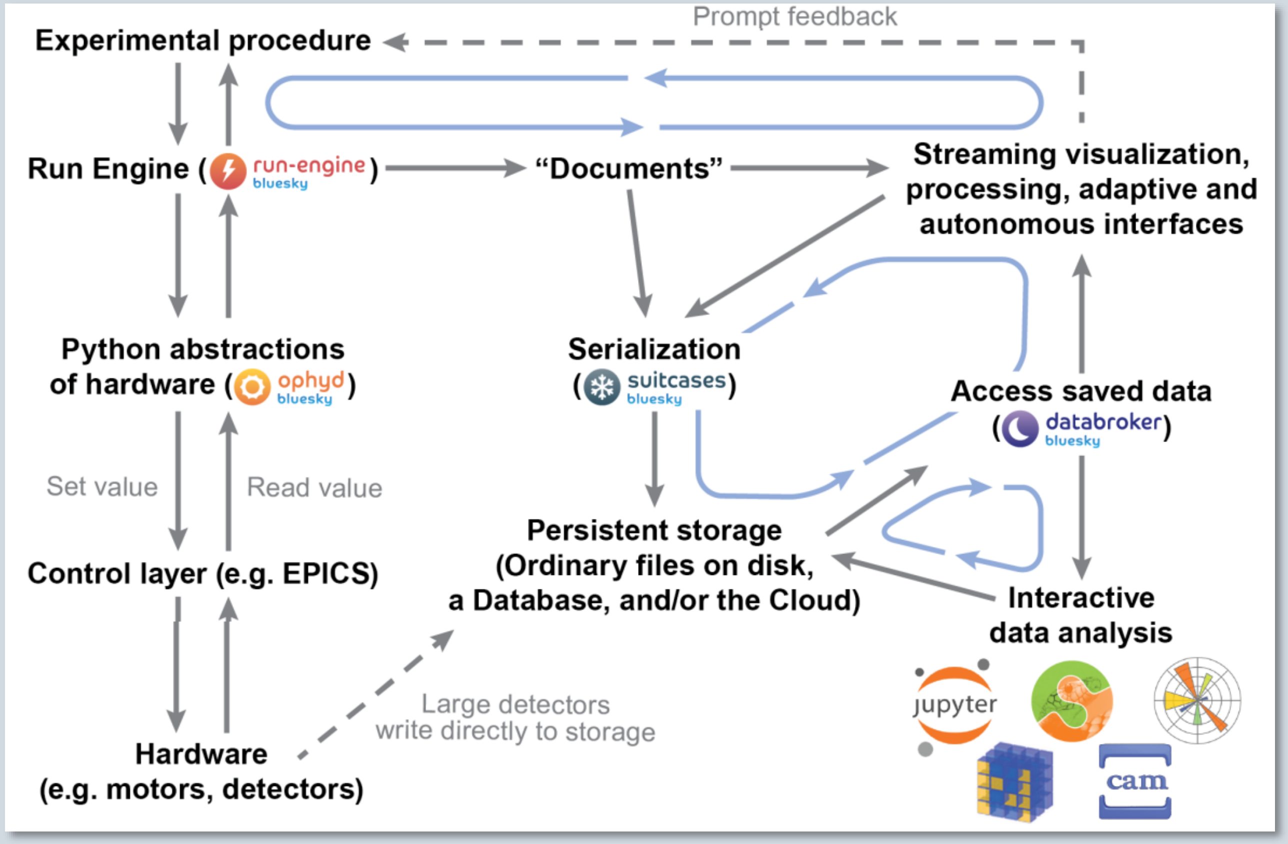


Prompt feedback
Experimental procedure
Run Engine (run-engine bluesky)
"Documents"
Streaming visualization, processing, adaptive and autonomous interfaces
Python abstractions of hardware (ophyd bluesky)
Serialization (suitcases bluesky)
Access saved data (databroker bluesky)
Set value
Read value
Control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

The Bluesky Collaboration

Now in use at multiple facilities

- NSLS-II (at least some use at all 29 beamlines, primary data acquisition at most beamlines)
- LCLS (widespread use) and SSRL (one or two instruments)
- APS (scaling up from a couple beamlines to dozens)
- ALS (at least one beamline, also scaling up)
- Diamond (piloting, has made significant development investments, see Dom's poster)
- Australian Synchrotron (several beamlines)
- Canadian Light Source (at least one)
- PSI (evaluating, not yet committed to adoption)
- Pohang Light Source II
- BESSY II
- Various academic labs


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

✗ Problems and ✓ Solutions

HOW WE GOT HERE

✗ Problem: Scattered Scan Logic at Diamond

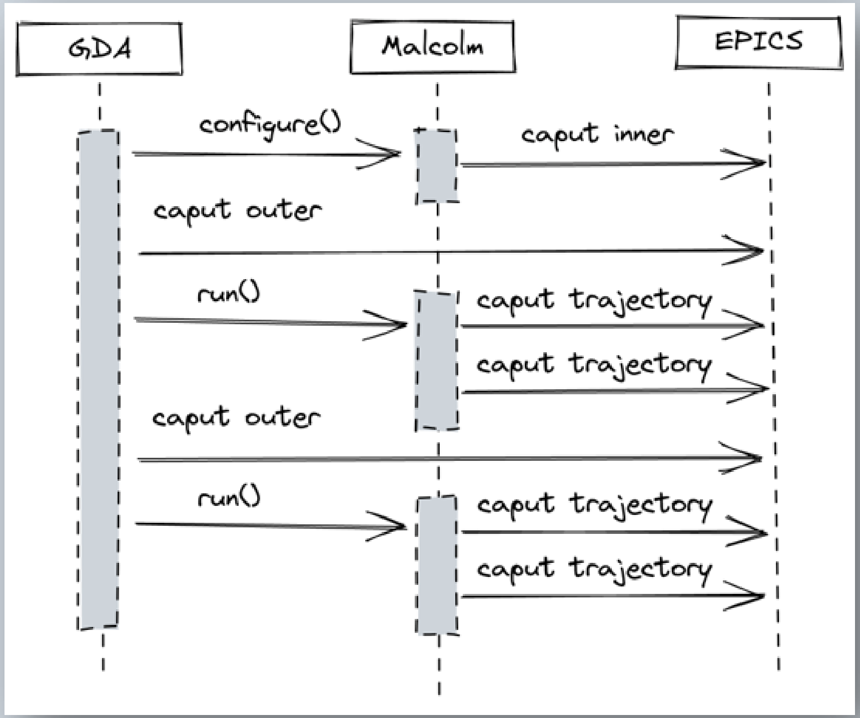
Fly scans in Malcolm (Python)

Outer step scans in GDA (Java)

Duplication

Shared state over network interface

Too complicated



✓ Solution: Bluesky as Unifying Framework

Lightweight clean architecture

Python3 based

Event model for data

Ecosystem of complementary modules

Proven at NSLS-II

Adopted by several other facilities

Our prototypes were leading to a similar architecture

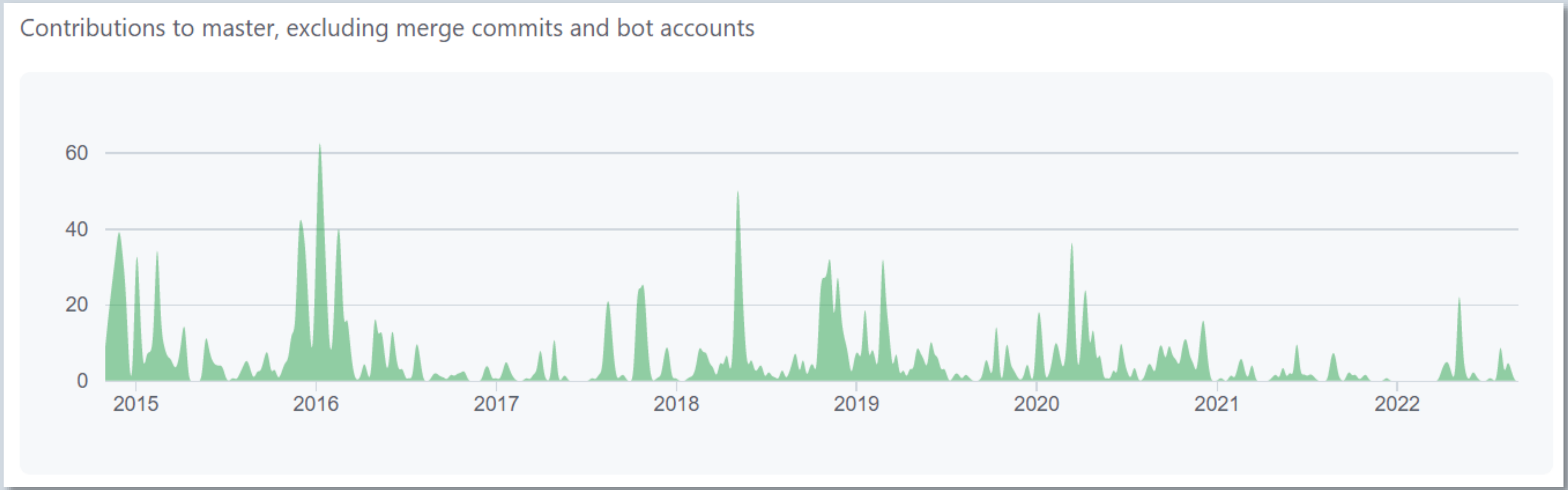
✗ Problem: Ophyd Codebase

Nearly 10000 lines of code

Many lessons learnt

Hard to maintain backward compatibility

Must maintain legacy code



✓ Solution: Brand new Package Following Bluesky Protocols

Rids us of legacy and allows us to make new technology/design choices

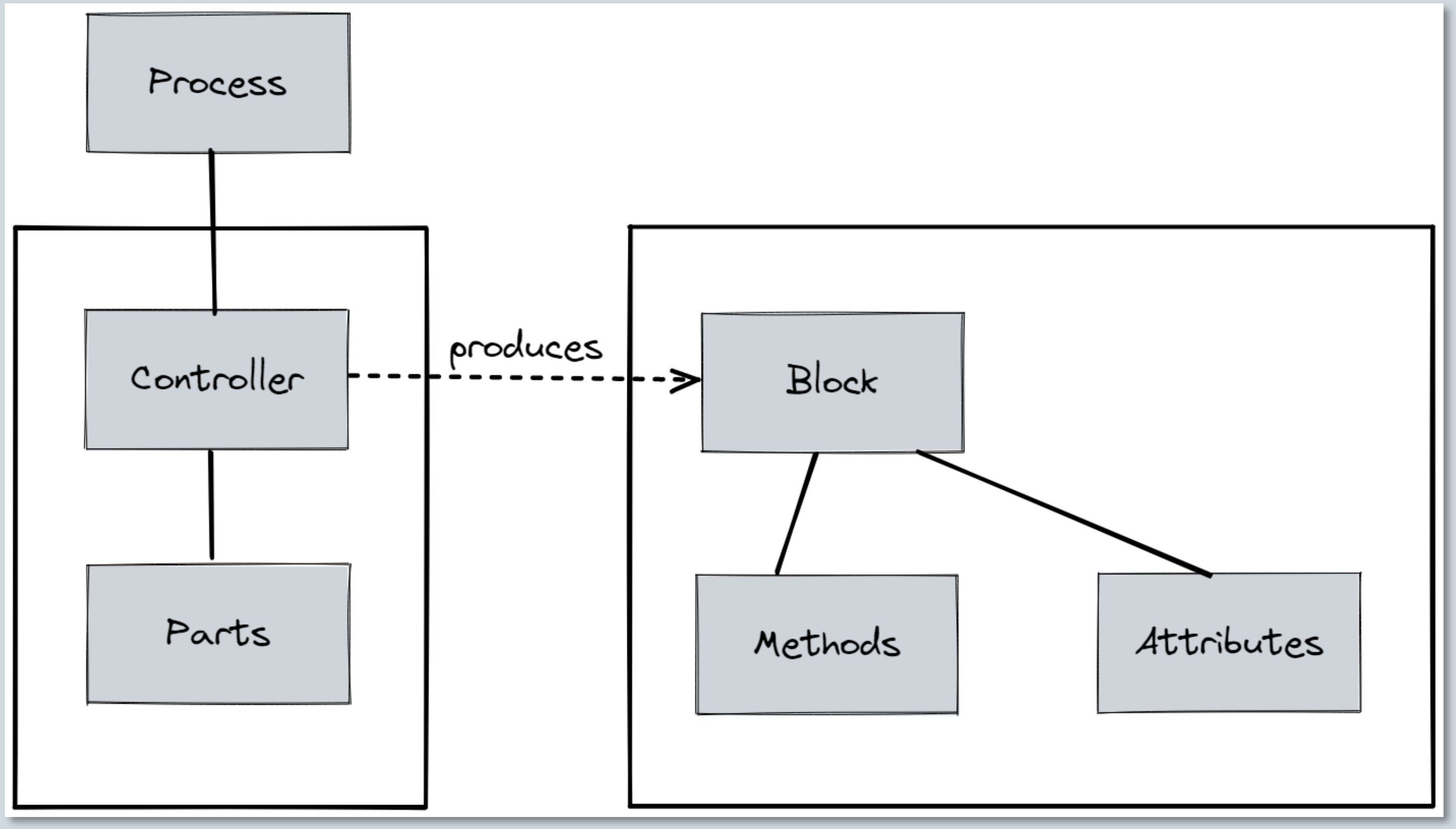
Backward compatibility is provided by the ability to run Ophyd v1 and Ophyd v2 devices alongside one another

Agile migration path is also provided as this allows porting devices as-needed

Written in modern Python 3/asyncio with appropriate QA controls

- Type hints
- CI for formatting and mypy

✗ Problem: Malcolm Device Tree is Too Abstract



✓ Solution: Bluesky Protocols

Readable, Movable, Flyable…

Compatible with Run Engine, so can write plans

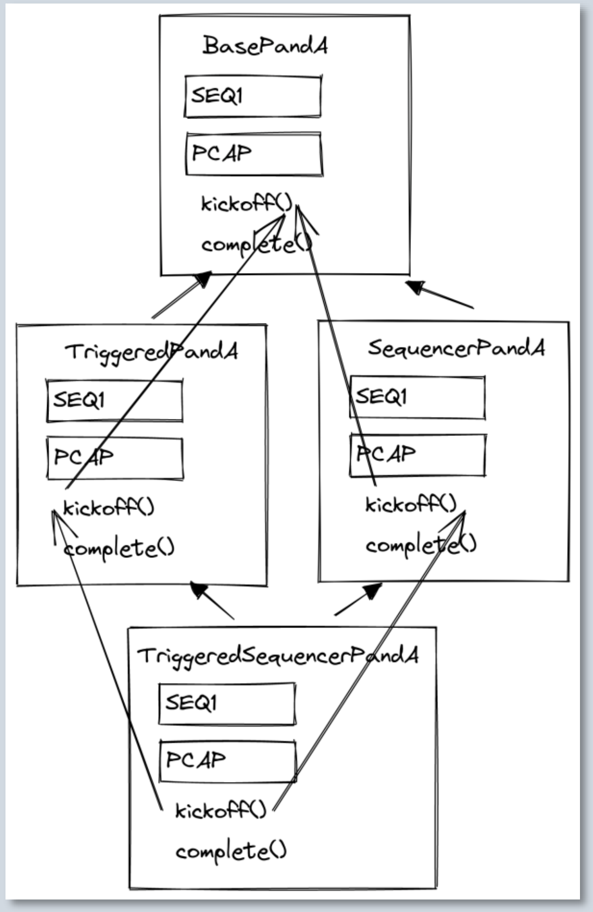
Easy to understand but flexible

✗ Problem: I/O Code Coupled to Logic in Ophyd

Readability/maintainability

Hard to deal with multiple configurations for devices, commonly use multiple inheritance

Like Malcolm, leaves us with a tree of devices with scattered logic which is hard to understand

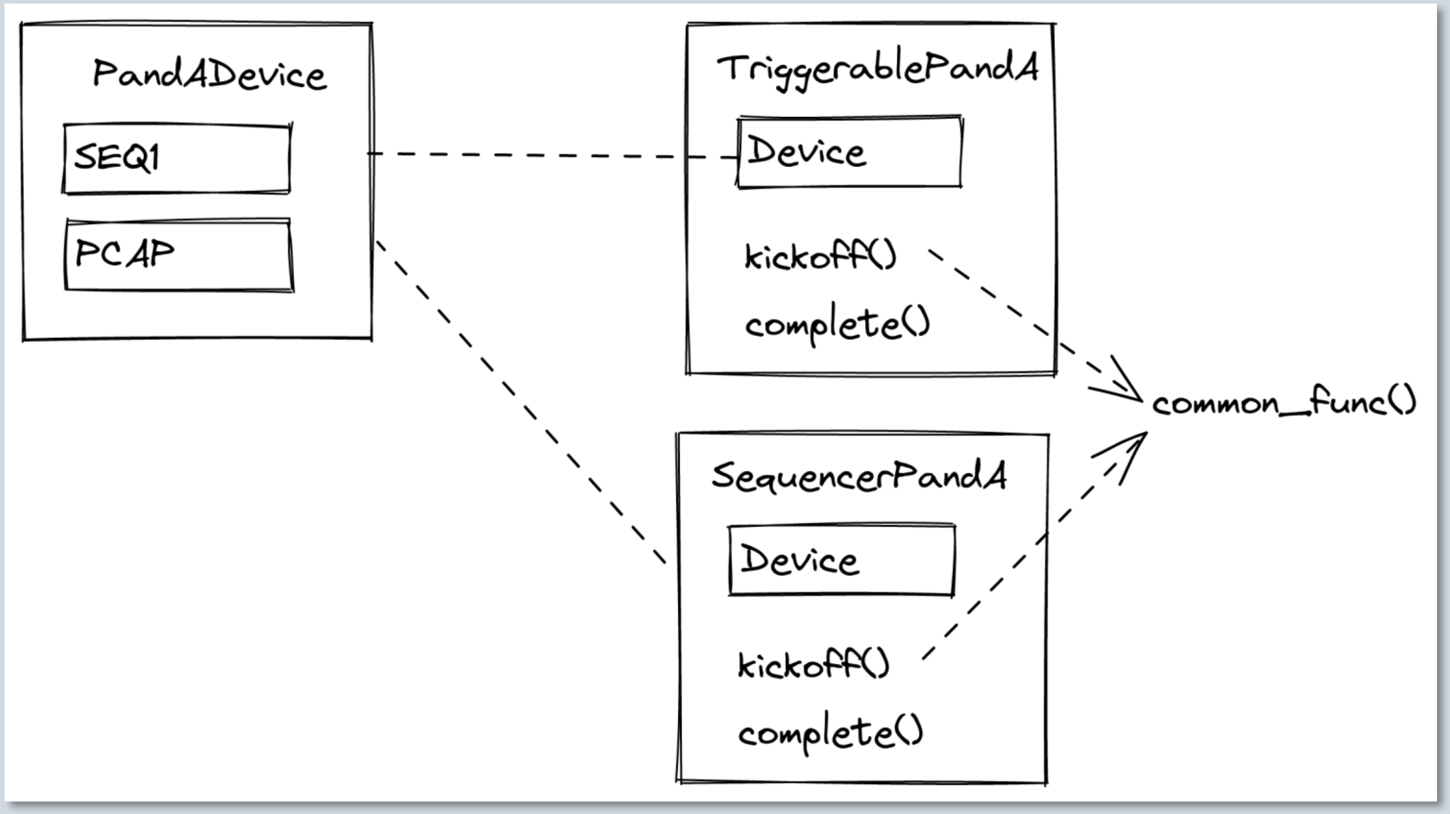


✓ Solution: Separate Objects to Handle I/O

Comms objects
- Collections of signals
- Async

Composition over inheritance
- Better way to handle multiple device configurations

Asyncio with aioca for EPICS comms



✗ Problem: PVs and Pre/Suffixes are Hard Coded

```
class CamBase(ADBase):
    ...
    # Shared among all cams and plugins
    array_counter = ADCpt(SignalWithRBV, "ArrayCounter")
    array_rate = ADCpt(EpicsSignalRO, "ArrayRate_RBV")
    asyn_io = ADCpt(EpicsSignal, "AsynIO")
    nd_attributes_file = ADCpt(EpicsSignal, "NDAttributesFile", string=True)
    pool_alloc_buffers = ADCpt(EpicsSignalRO, "PoolAllocBuffers")
    pool_free_buffers = ADCpt(EpicsSignalRO, "PoolFreeBuffers")
    pool_max_buffers = ADCpt(EpicsSignalRO, "PoolMaxBuffers")
    pool_max_mem = ADCpt(EpicsSignalRO, "PoolMaxMem")
    pool_used_buffers = ADCpt(EpicsSignalRO, "PoolUsedBuffers")
    pool_used_mem = ADCpt(EpicsSignalRO, "PoolUsedMem")
    port_name = ADCpt(EpicsSignalRO, "PortName_RBV", string=True)
```

✓ Solution: Separate Definitions and Option for PVI

New component: SignalCollector, to provide signal definitions

Multiple implementations:

- Hard-coded
- Naming convention
- PVI
- Databases

Current Status

Early alpha

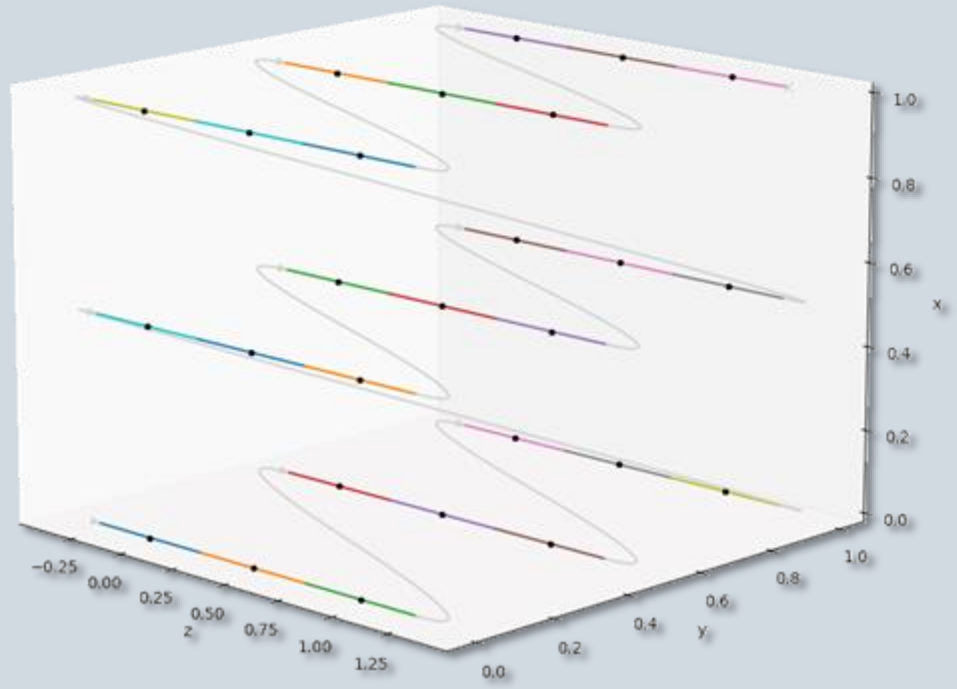
Scan modelling library: https://github.com/dls-controls/scanspec

Currently being merged into Ophyd repository, ophyd.v2 package

Issue currently open to update build system: https://github.com/bluesky/ophyd/issues/1059

Device repositories:

- https://github.com/bluesky/ophyd-tango-devices
- https://github.com/bluesky/ophyd-epics-devices



Summary

Ophyd v2 developed to replace Ophyd and Malcolm

Incorporates lessons learned from Ophyd v2 (NSLS-II) and fly scanning functionality from Malcolm (Diamond)

Can run Ophyd v1 and Ophyd v2 devices concurrently

Clear migration path

Jointly developed by the two facilities, bi-weekly call

Also regular representation from:

- PSI
- Kyle Sunden & Blaise Thomson, UW Madison

https://github.com/bluesky/ophyd/issues/1059