Epics Collaboration Meeting

September 20th, 2016

LCLS Python

Command-line EPICS Interface

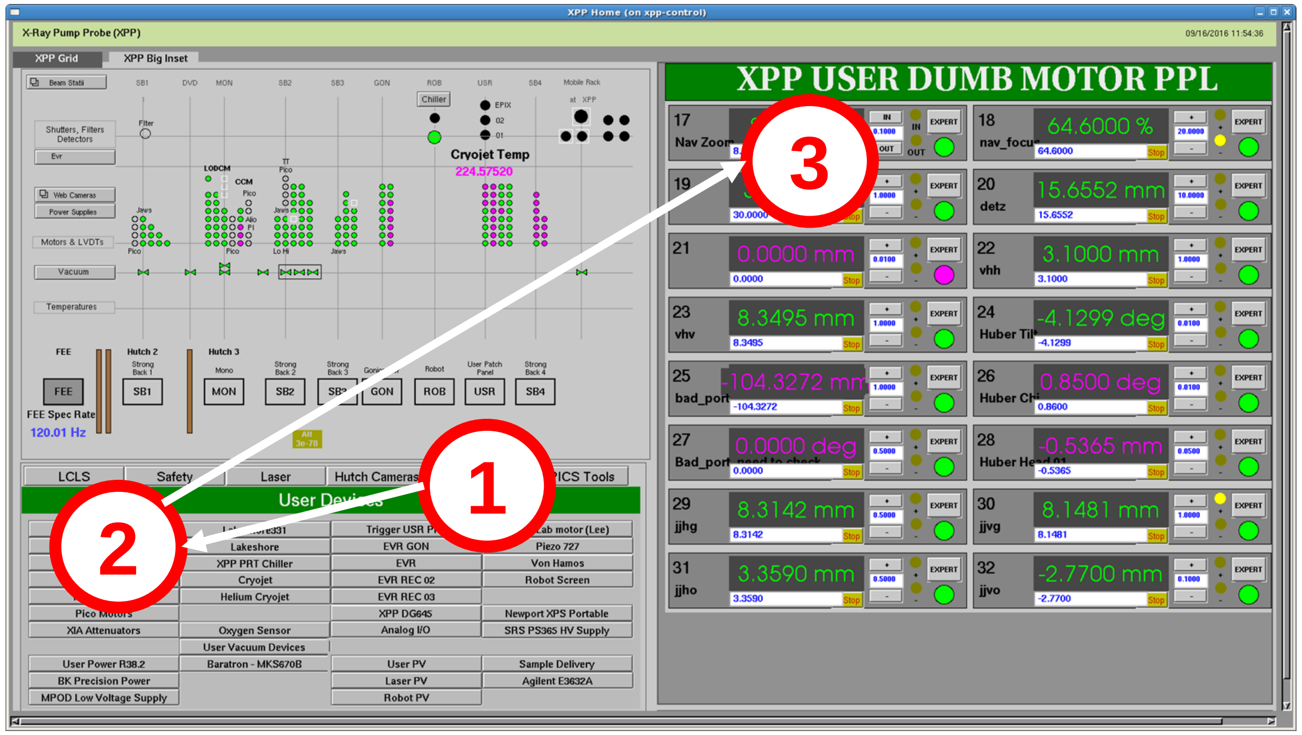
Zachary Lentz
zlentz@slac.stanford.edu
LCLS Controls & Data Systems Division

U.S. DEPARTMENT OF ENERGY
Office of Science

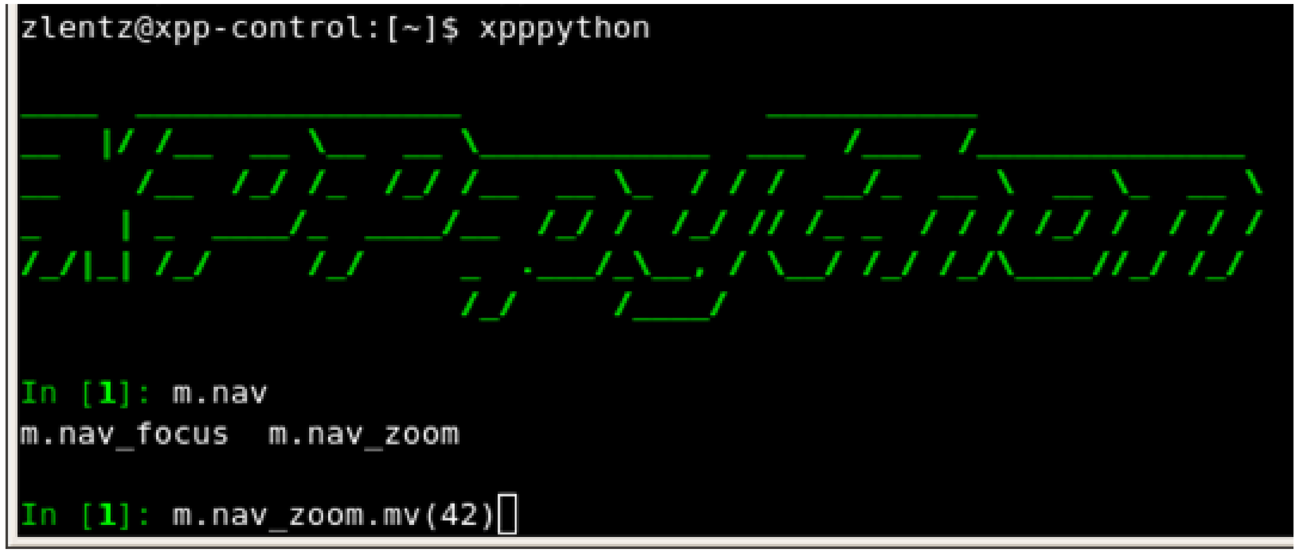
SLAC NATIONAL ACCELERATOR LABORATORY

Motivation

- **GUIs are Inefficient**
  - Clicking and mouse movement are slow
- **Fundamental EDM Limitations**
  - Cannot effectively save/reuse command sequences
  - No DAQ support (unless we bog down DAQ servers by having them host PVS)
- **Ease of access/extension**
  - Scientist familiarity with Spec
  - Growing popularity of Python
  - Code flexibility



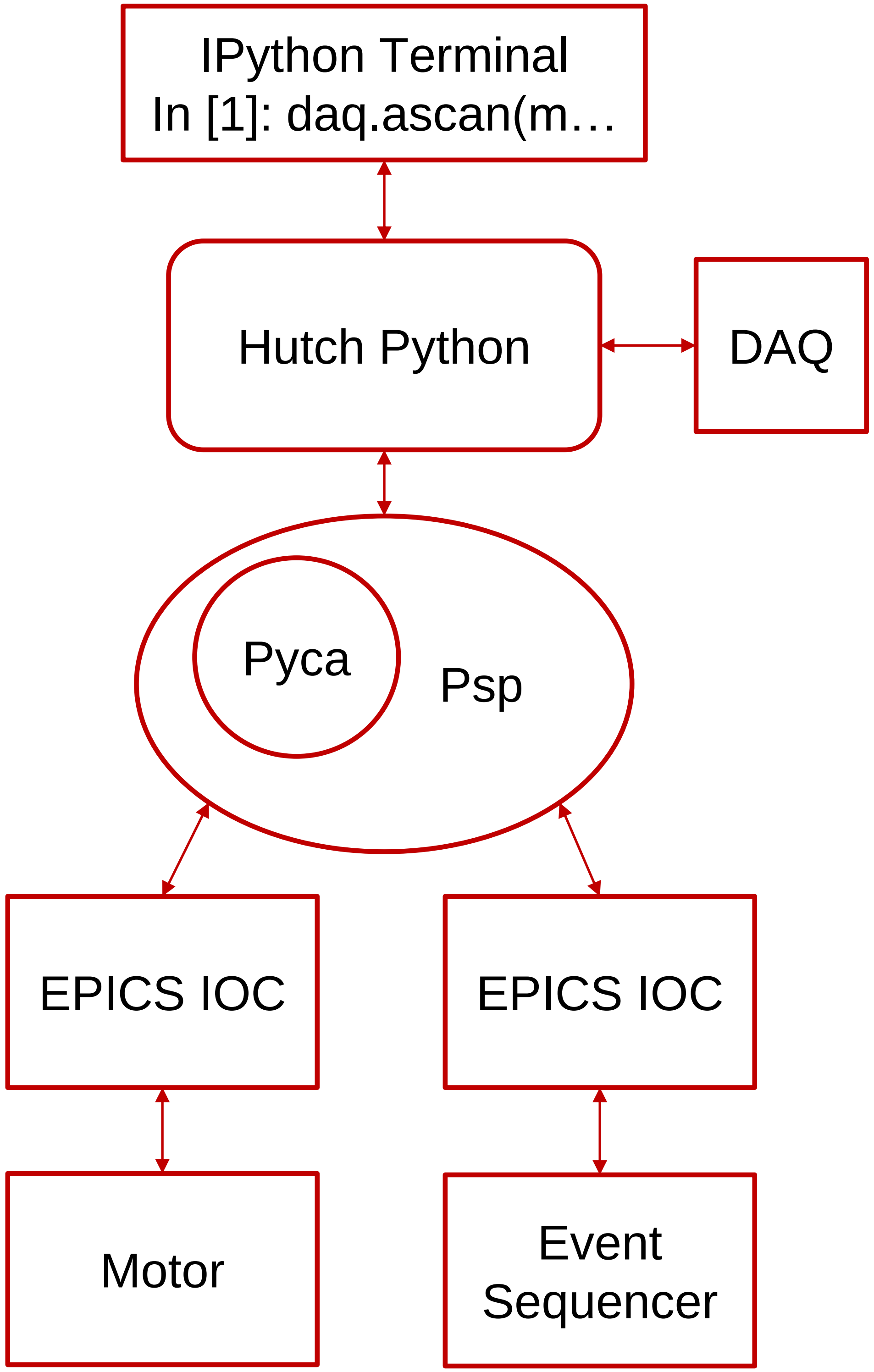
17 Nav Zoom
18 nav_focus 64.6000 % 64.6000
19 30.0000
20 detz 15.6552 mm 15.6552
21 0.0000 mm 0.0000
22 vhh 3.1000 mm 3.1000
23 vhv 8.3495 mm 8.3495
24 Huber Tilt -4.1299 deg -4.1299
25 bad_port -104.3272 mm -104.3272
26 Huber Chi 0.8500 deg 0.8600
27 Bad_port need to check 0.0000 deg 0.0000
28 Huber Head 01 -0.5365 mm -0.5365
29 jjhg 8.3142 mm 8.3142
30 jjvg 8.1481 mm 8.1481
31 jjho 3.3590 mm 3.3590
32 jjvo -2.7700 mm -2.7700



```
zlentz@xpp-control:[~]$ xpppython

In [1]: m.nav
m.nav_focus   m.nav_zoom

In [1]: m.nav_zoom.mv(42)
```

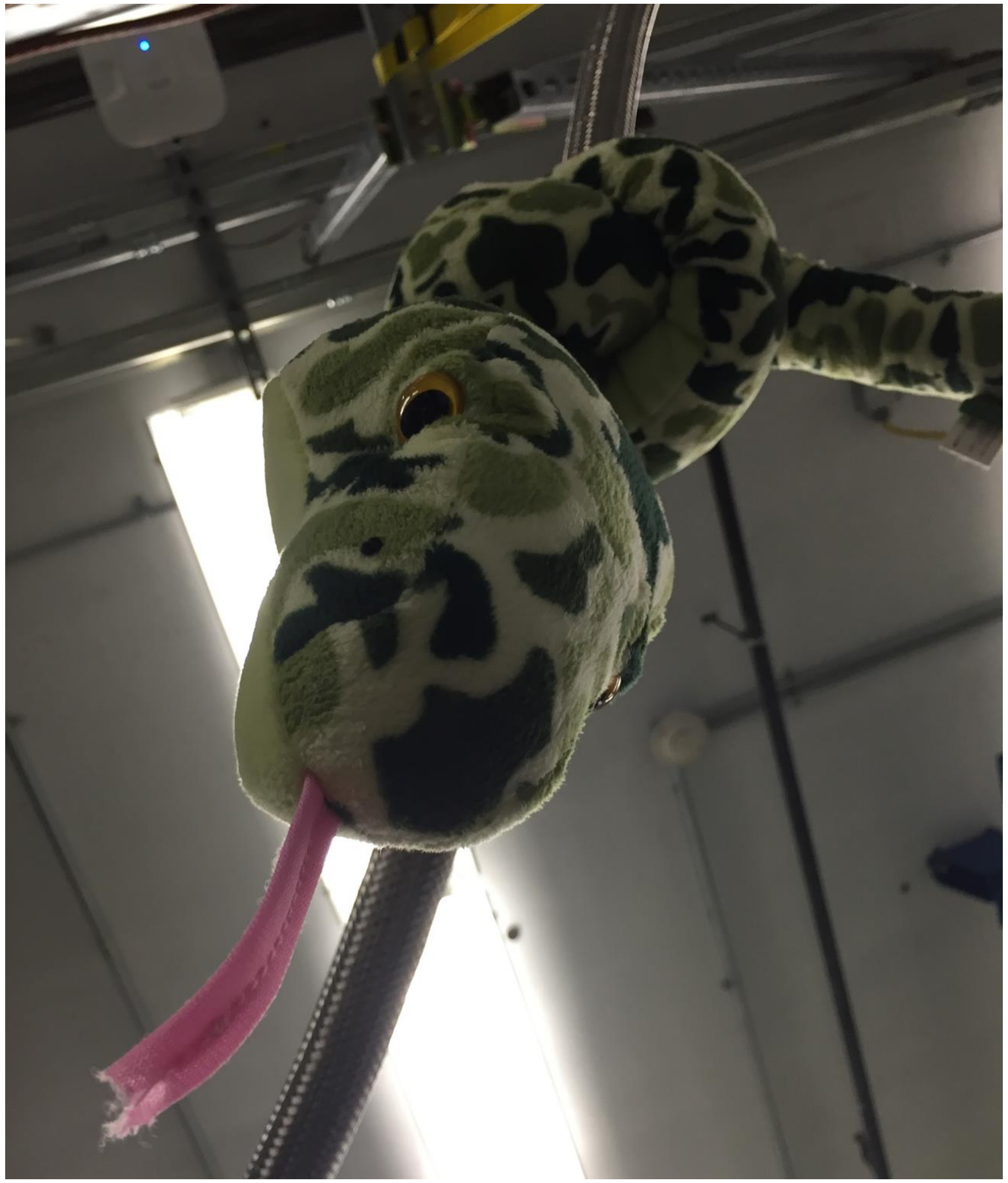
Overview

- **Python Beamline Environment**
  - Controls beamline devices through EPICS records using Pyca/Psp
  - Controls DAQ devices through Python interfaces written in C++
  - Has tons of built-in macros to ease experimental setup, do calculations, etc
- **IPython Command Line**
  - Tab completion
  - View docstrings
  - View source code
  - Manage history



Development History

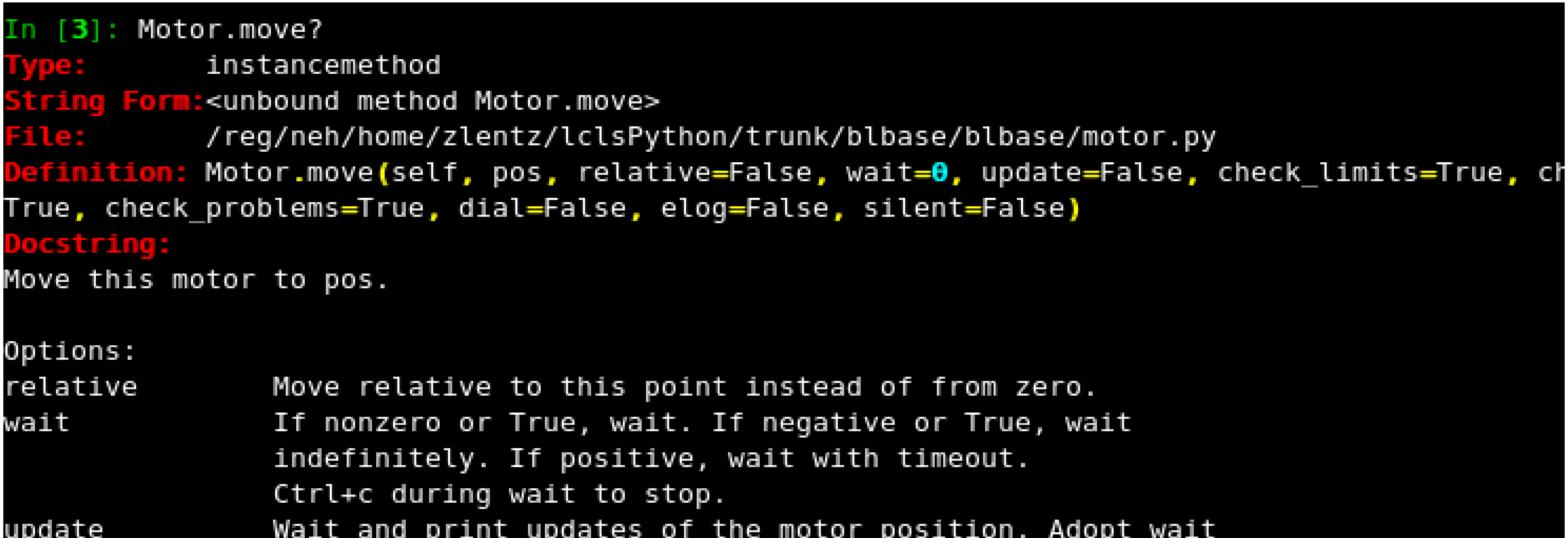
- Scientists decide they want something like Spec instead of a GUI
- Pyca is born, a lightweight EPICs interface in pure C++
- Scientists write a Pyca wrapper and build xpppython, xcspython, etc.
- Code grows organically and rapidly
- Psp is created as a PyEpics-like wrapper for Pyca
- Controls developers regroup to polish the Python code
- Combine code into cohesive, shared packages.



The XPP Python in its natural habitat, the XPP instrument at LCLS

Using the IPython Terminal

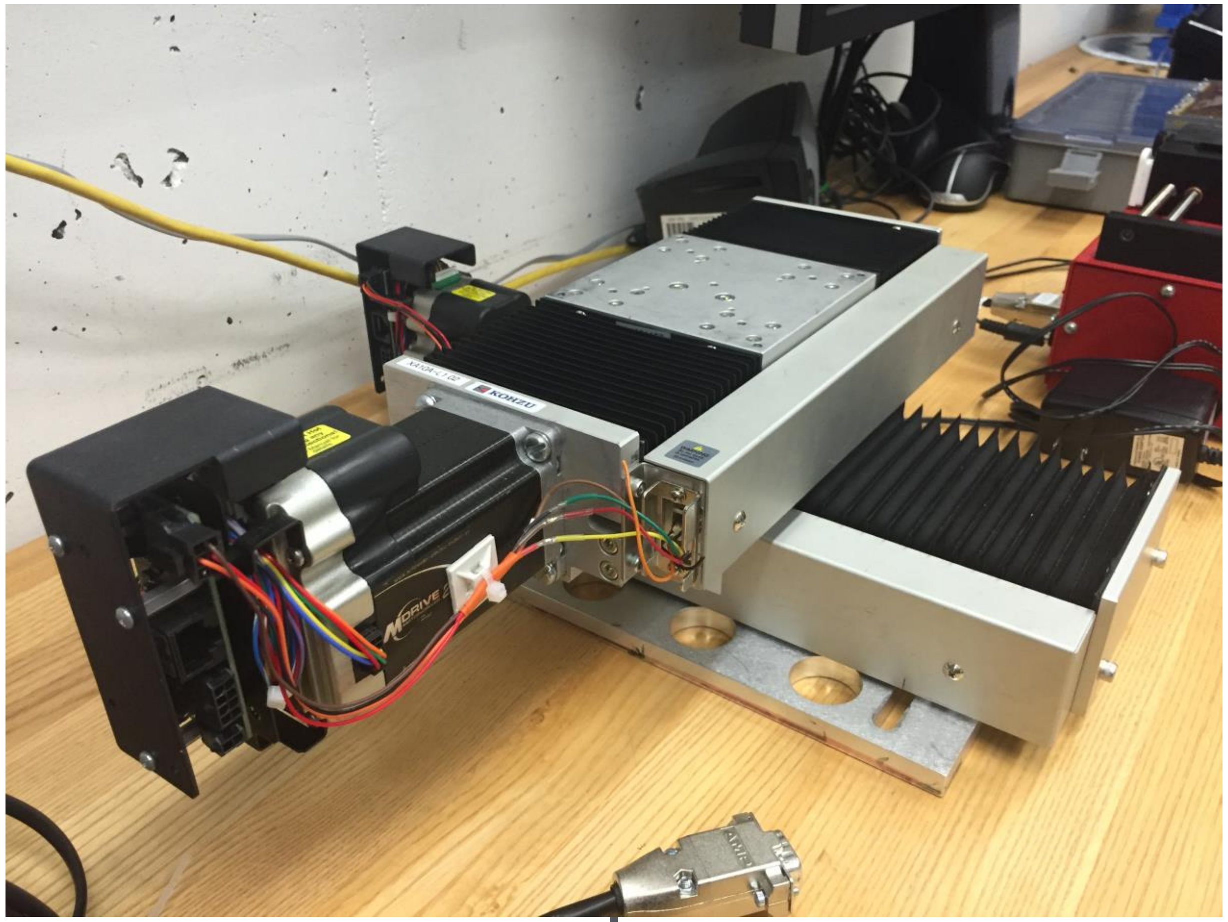
- Do not need to remember exact commands or interface details
    - Tab completion: press tab to see available options
    - History between sessions: press up for similar commands
    - View docstrings: use the ? operator to view docstrings
    - View source code: use the ?? operator to view source
- Can write short Python code in-line for convenience

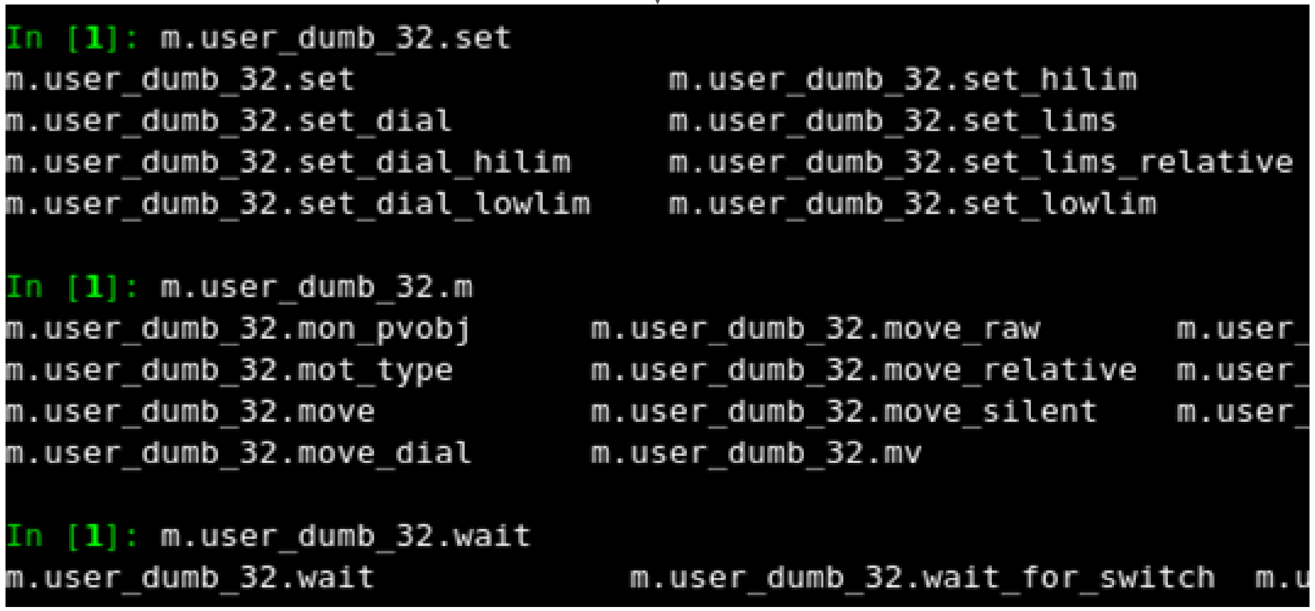


```
In [3]: Motor.move?
Type:        instancemethod
String Form:<unbound method Motor.move>
File:        /reg/neh/home/zlentz/lclsPython/trunk/blbase/blbase/motor.py
Definition: Motor.move(self, pos, relative=False, wait=0, update=False, check_limits=True, ch
True, check_problems=True, dial=False, elog=False, silent=False)
Docstring:
Move this motor to pos.

Options:
relative        Move relative to this point instead of from zero.
wait            If nonzero or True, wait. If negative or True, wait
                indefinitely. If positive, wait with timeout.
                Ctrl+c during wait to stop.
update          Wait and print updates of the motor position. Adopt wait
```

Device Class Abstractions

- **Each physical device has an associated Python class**
  - Motor, Valve, Attenuator
- **Classes are interfaces to the IOC with user-friendly names**
  - sample_z.move(42)
- **We also define macros for common device tasks**
  - BeLens can calculate which lens to use
  - Goniometer calculates center of rotation
  - Can open EDM screen

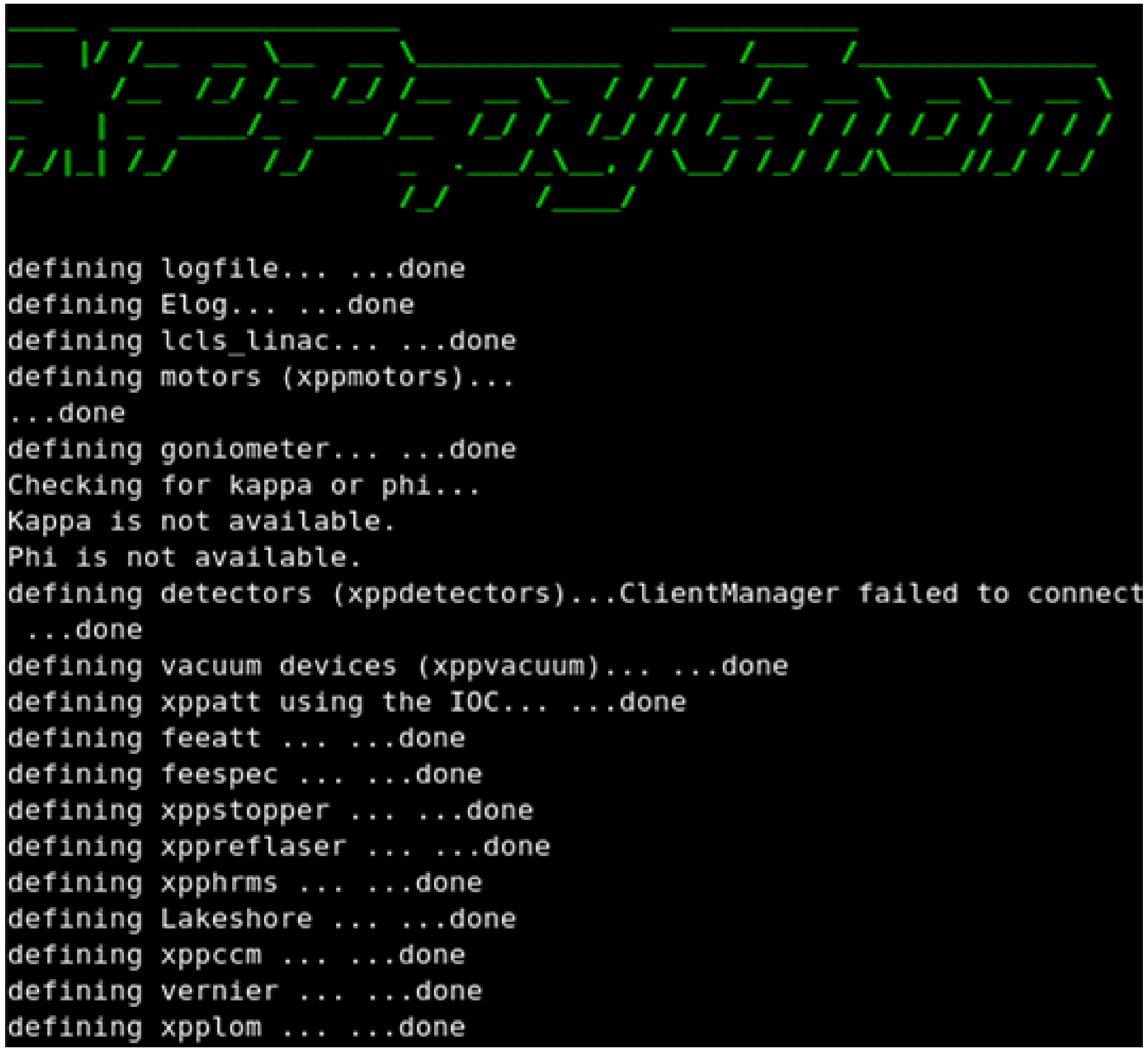




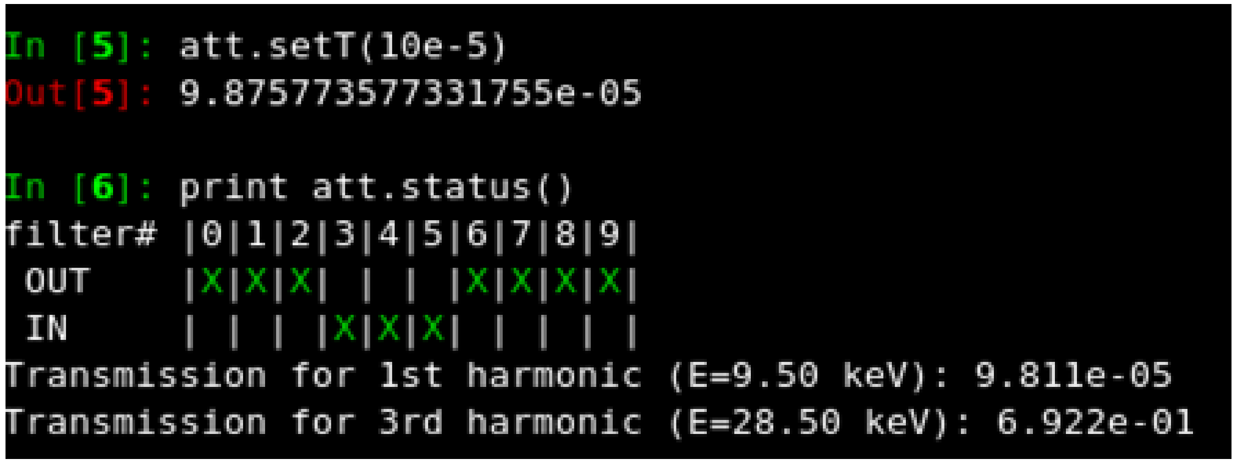
```
In [1]: m.user_dumb_32.set
m.user_dumb_32.set                 m.user_dumb_32.set_hilim
m.user_dumb_32.set_dial            m.user_dumb_32.set_lims
m.user_dumb_32.set_dial_hilim      m.user_dumb_32.set_lims_relative
m.user_dumb_32.set_dial_lowlim     m.user_dumb_32.set_lowlim

In [1]: m.user_dumb_32.m
m.user_dumb_32.mon_pvobj       m.user_dumb_32.move_raw        m.user_
m.user_dumb_32.mot_type        m.user_dumb_32.move_relative   m.user_
m.user_dumb_32.move            m.user_dumb_32.move_silent     m.user_
m.user_dumb_32.move_dial       m.user_dumb_32.mv

In [1]: m.user_dumb_32.wait
m.user_dumb_32.wait                  m.user_dumb_32.wait_for_switch  m.u
```

Beamline Python Environment

- All Beamline and Experiment-specific device objects are loaded into an interactive IPython terminal
- Add in useful utilities and calculations not related to individual devices
- Group objects by type for manageability and tab-completion efficiency
- From one place, an operator can do basically anything without navigating a GUI



```
defining logfile... ...done
defining Elog... ...done
defining lcls_linac... ...done
defining motors (xppmotors)...
...done
defining goniometer... ...done
Checking for kappa or phi...
Kappa is not available.
Phi is not available.
defining detectors (xppdetectors)...ClientManager failed to connect
 ...done
defining vacuum devices (xppvacuum)... ...done
defining xppatt using the IOC... ...done
defining feeatt ... ...done
defining feespec ... ...done
defining xppstopper ... ...done
defining xppreflaser ... ...done
defining xpphrms ... ...done
defining Lakeshore ... ...done
defining xppccm ... ...done
defining vernier ... ...done
defining xpplom ... ...done
```



```
In [5]: att.setT(10e-5)
Out[5]: 9.875773577331755e-05

In [6]: print att.status()
filter# |0|1|2|3|4|5|6|7|8|9|
 OUT    |X|X|X| | | |X|X|X|X|
 IN     | | | |X|X|X| | | | |
Transmission for 1st harmonic (E=9.50 keV): 9.811e-05
Transmission for 3rd harmonic (E=28.50 keV): 6.922e-01
```

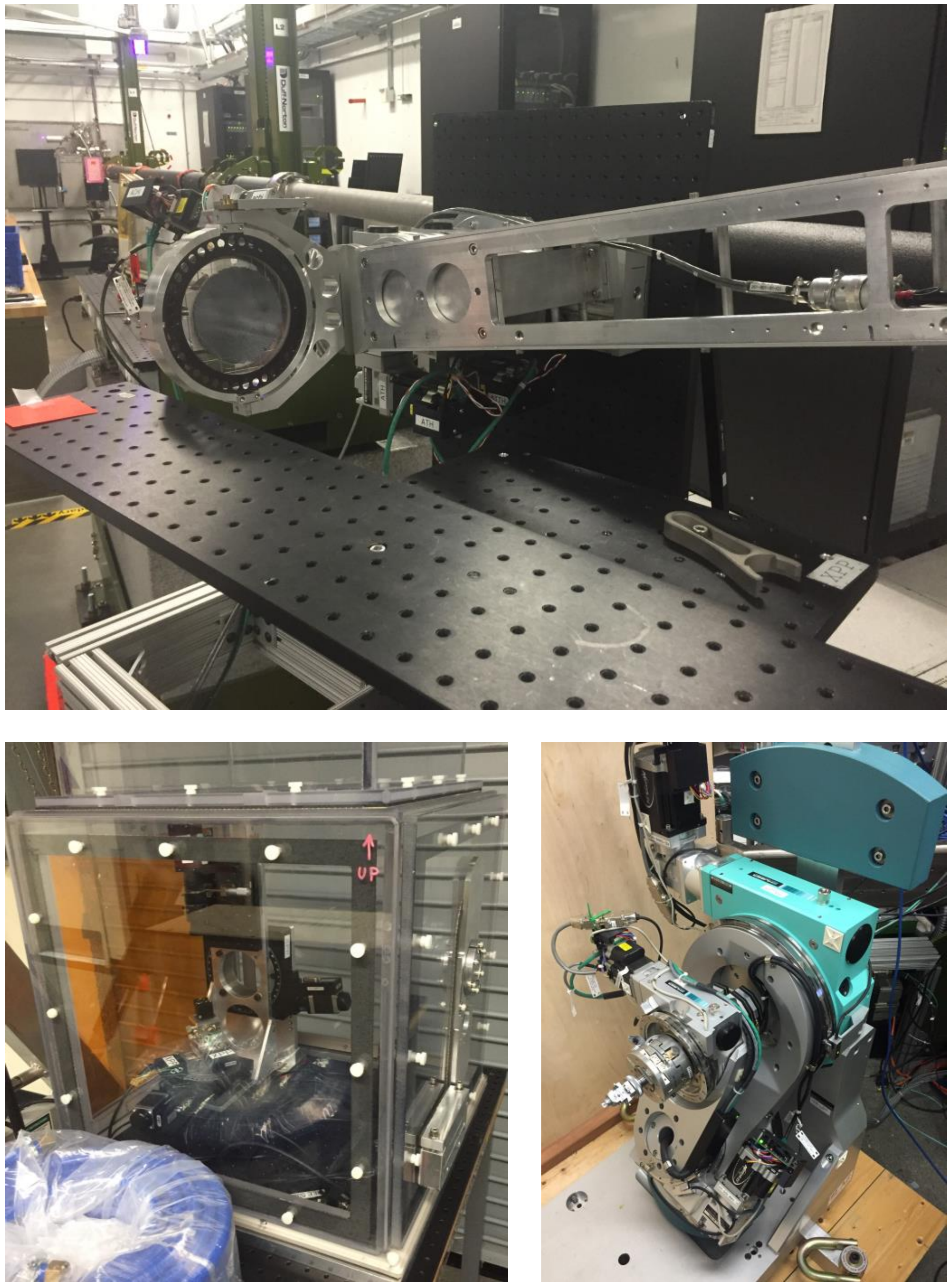
Under the Hood: Pyca and Psp

- **Pyca: SLAC-made Python interface for EPICS**
  - Fast, lightweight, efficient. Direct interface to EPICS calls.
- **Advantages over PyEpics:**
  - Faster PV instantiation, less overhead
  - Much better scaling when we connect to 1000+ PVs
  - Can write application-specific C++ code to process data before sending it to Python. This lets us efficiently process and use image data without anticipating all use cases at the IOC level.
- **Disadvantages:**
  - Obtuse interface, too low-level for easy use in Python
- **Psp: Human interface for Pyca**
  - Wrapper to address concerns over user-friendliness
  - Used in each device class to connect to the IOC

Advantages of Python

- Python code is easy to write, giving us a large pool of ideas to work with
- Allows us to work with the users and quickly create experiment-specific controls scripts
  - Coordinated motion
  - Custom scans
- Tons of useful, open source Python packages can be incorporated (and we can write our own!)

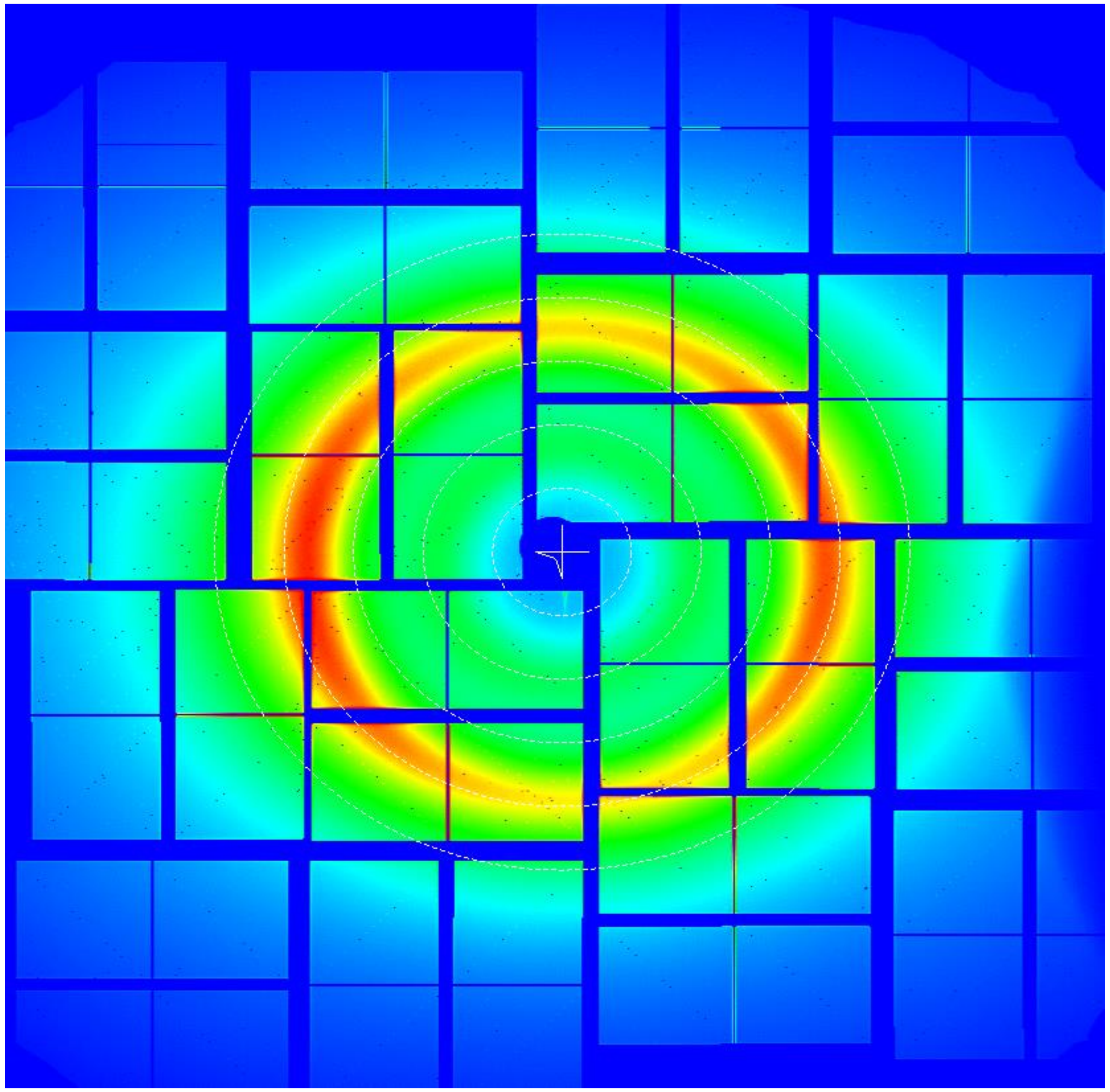
Various open source Python modules



Python allows us adapt to a variety of experimental setups

DAQ and Scans

- The LCLS DAQ can be controlled through Python
- This allows us to efficiently do data scans in one line:
    - daq.ascan(motor, args…)
- The code will move a motor (or group of motors) through various positions, collecting data only at each point.
- Experiments sometimes become “for i in range(100), scan”, allowing us to focus on data analysis.
- We can also do “fly-scans”, provided we have fast readback.



X-Ray Detector Image

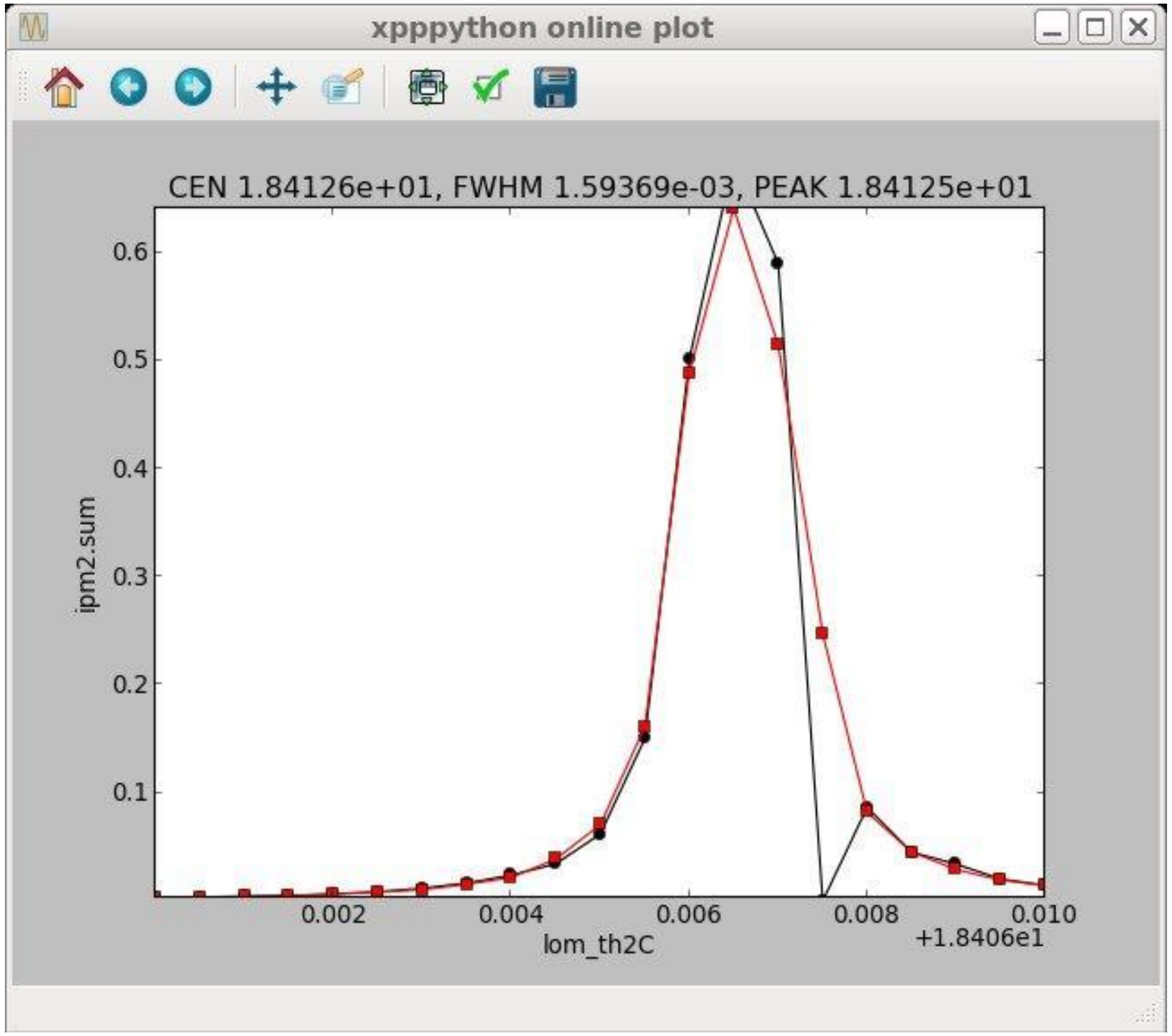
Disadvantages of Python

- Performance of Python code is not on par with compiled languages
    - We need to make sure we do data processing in the C++ layer
- Many people can can write in Python, but the code isn't always so great
    - We need to have good quality-control to keep the codebase managable

Getty image of an African rock python

Development Directions

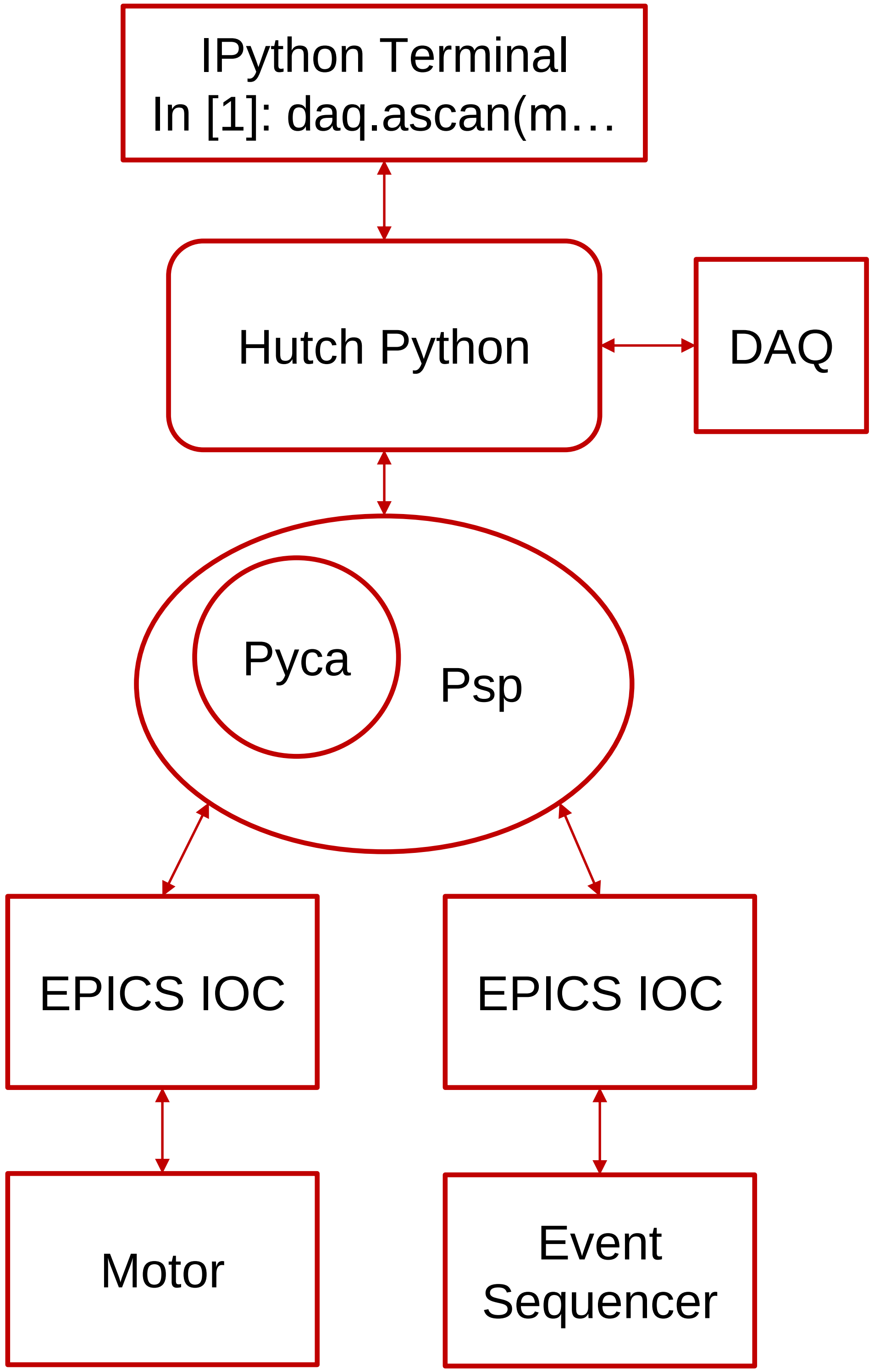
- **Improve DAQ classes for reliability and scan flexibility**
  - Some of these weren't implemented so well
- **Merge environments with the analysis group**
  - Code is currently not compatible
- **Realtime analysis and diagnostics**
  - Adjust scan points as we go based on analysis



Typical scan that could be made smoother, faster, or easier with real time analysis

Summary

- IPython interface allows operators to quickly and intuitively access all available resources
- Python is flexible, which is key at LCLS
- The system lets us coordinate our DAQ and our EPICS devices in a simple, reconfigurable way



IPython Terminal
In [1]: daq.ascan(m…

Hutch Python

DAQ

Pyca

Psp

EPICS IOC

EPICS IOC

Motor

Event Sequencer

Acknowledgements

I’d like to acknowledge the major contributors to the Pyca library and beamline python modules

Amedeo Perazzo

Marco Cammarata

David Fritz

Henrik Lemke

Zhou Xing

Daniel Flath

Bruce Hill

Matt Weaver

Daniel Damiani

Mike Browne

Silke Nelson

Theodore Rendahl

Dehong Zhang

Jing Yin