Goal

Able to run/understand some simple (yet not simple) psana-python examples. For example, our 46-line example will:

- **Access LCLS data**
- **Can be run online/offline**
- **“Randomly accesses” events**
- **Has real-time plots**
- **Will run in parallel, in principle on thousands of cores**
- **You can run it on your laptop (including parallelism)**

Introduction to Psana

SLAC

Psana: Photon Systems ANAlysis (comparable to CASS)

- based on C++ and python
- development started in 2011 (two years after start of LCLS!) and continues…
- algorithms in reusable "modules" that can be chained together in a serial fashion, but emphasizing putting logic (like filters, ROIs, other algorithms) in python.
- support for detetector calibrations using standard tools
- the same software functions offline and online (with real-time plot display)
- analyzes data online/offline parallelizing over events (up to thousands of cores). **Many experiments have analyzed 120Hz online in real-time**.
- LCLS specific: not specific for crystallography.

Recently recommending **psana-python** to LCLS users:

- easy, free, powerful state-of-the-art language, with reusable algorithm libraries (e.g. scipy)
- use parallelization to overcome CPU limitations (many LCLS analyses are I/O bound anyhow)
- some cool self-documenting features (e.g. tab-completion in `ipython`, python help strings)

Introduction

SLAC

Psana used by:

- Cctbx
- Many LCLS experiments in all hutches
- Produces the standard “translated” LCLS hdf5 files

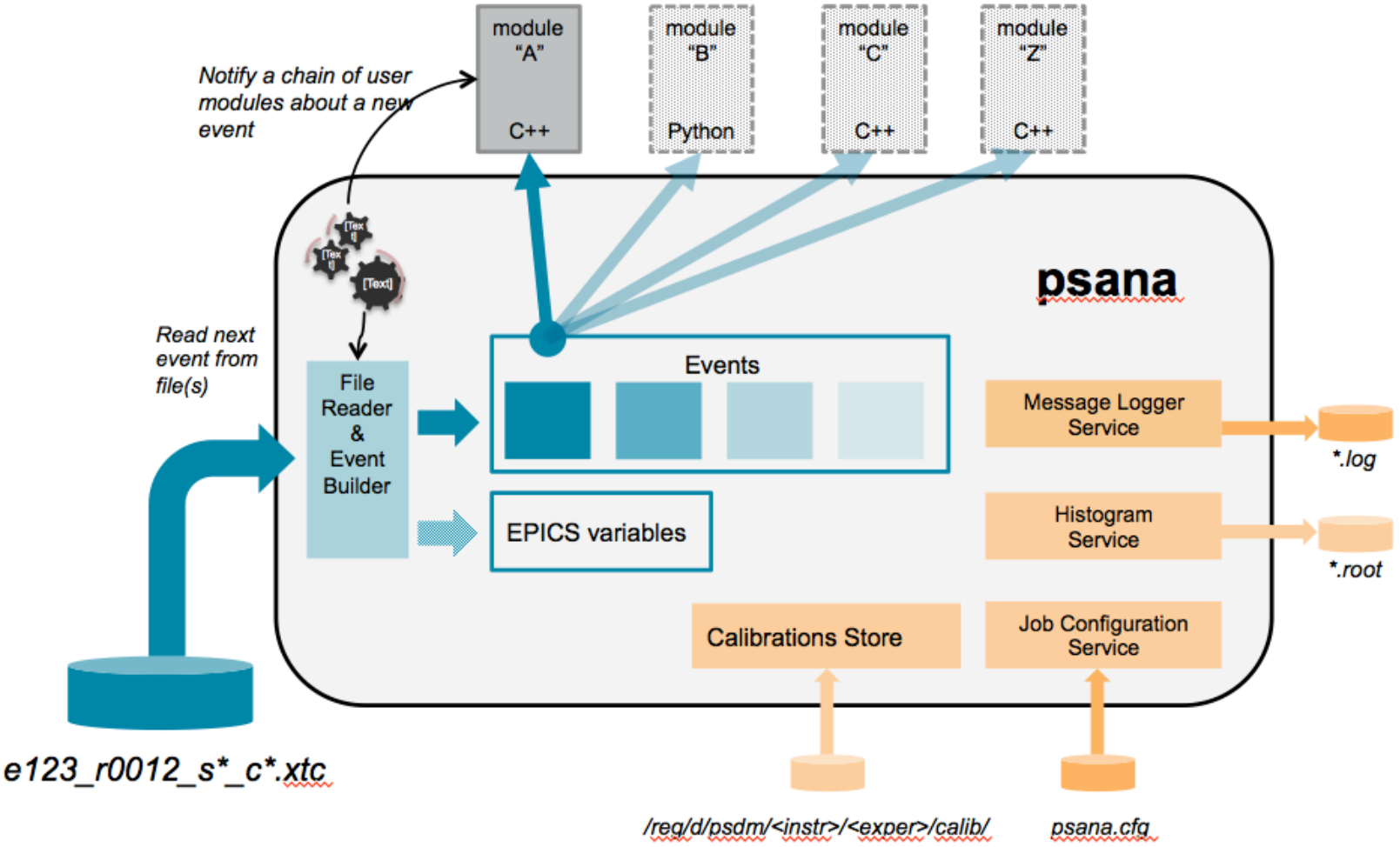
Workshop is about BioXFEL software, but you may also want to “play” with LCLS data using psana (e.g. how is the beam energy changing? what is the range for this diode?)

A First 17-Line Psana-Python Script

- Go to confluence.slac.stanford.edu
- Click on “LCLS Data Analysis” on the right-hand side
- Click on “psana: Python Interface” (fifth from top on left-hand side)

With a few small changes this script will work online/offline/laptop, or run on 1000 cores parallelizing over events, or access “calibrated” images.

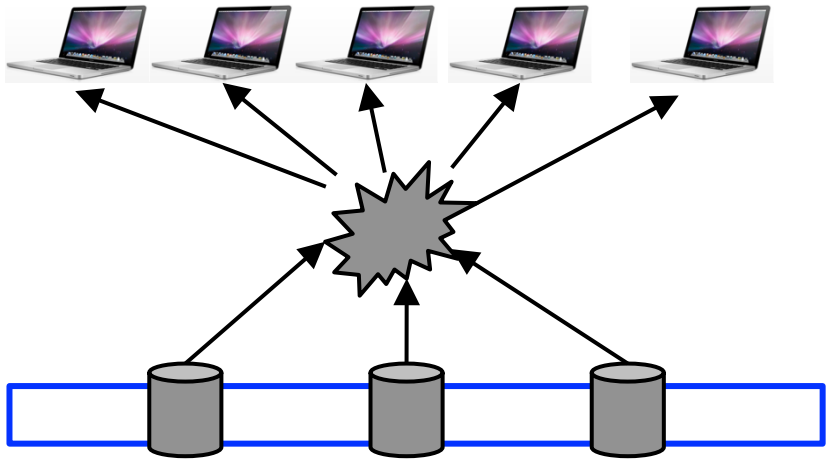
Psana Event Processing



New Features (Last 10 Months)

MPI Parallelization (!!)

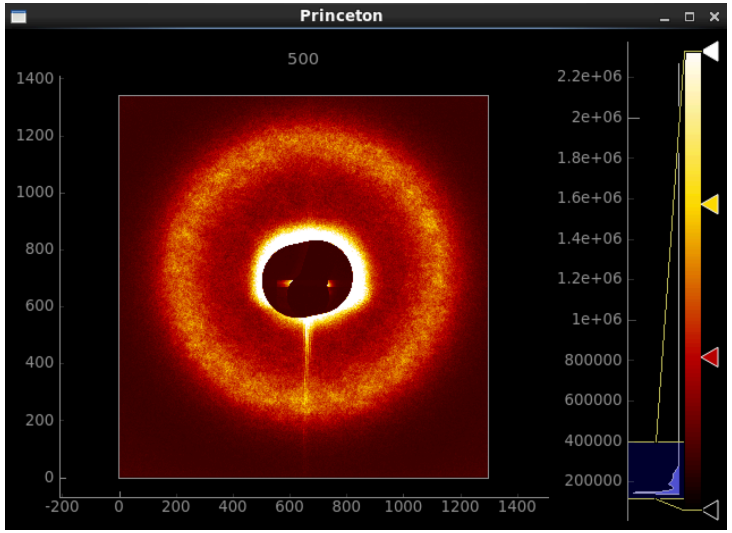
- MPI is the most commonly used multi-node parallelization for science
- Can help with CPU/memory/input-output bottlenecks (up to parallel-filesystem performance)
- Batch–system support
- psana provides parallelization-over-events (and others) using MPI
- Each node access a different set of events in the file using random-access



One file, distributed across parallel file system

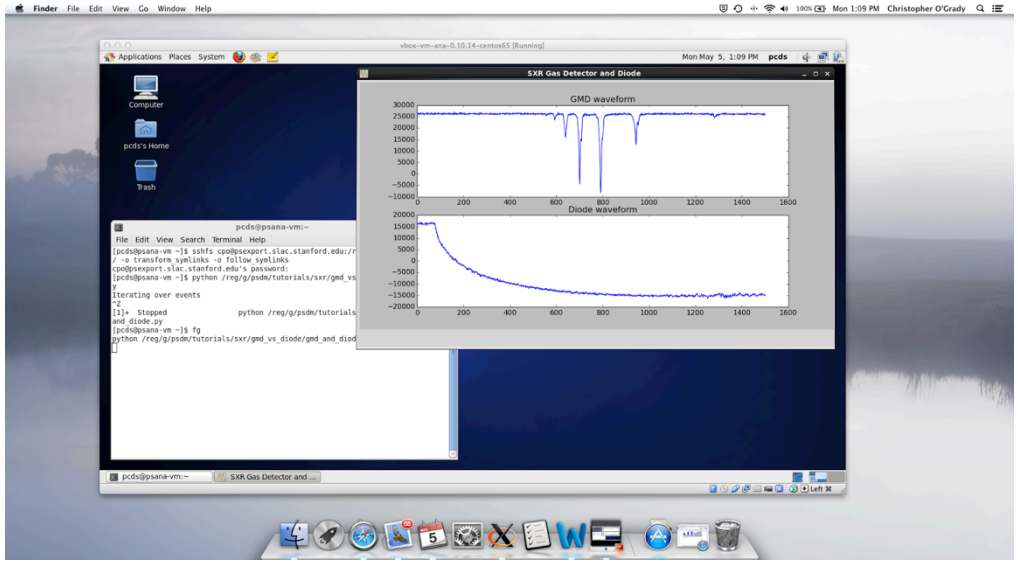
Online Analysis

- The same offline MPI parallelization works online as well, listening to the DAQ network-multicast data
- A 46-line python script is enough code to sum an image at 120Hz in real-time (>1GB/s) running on multiple machines with MPI
- User-definable plots show the results in real-time (complements “AMI”)
- Many experiments that need custom monitoring have used this
- Same calibration/geometry used for online/offline running



Remote Analysis and Visualization

- Many users analyze data “in France” and remote visualization/analysis can be cumbersome. Two recently-available improvements:
  - run full psana on your own Mac/Windows laptop on reduced-size data using free “virtual box” software. (1GB, takes about 1 hour to install)
  - login to SLAC using free *NX technology* to improve remote X11 performance



Random Access to XTC Events

SLAC

Can save "EventId" of interesting events (e.g. online), then after run ends can jump just to those interesting events:

```
import psana
ds = psana.DataSource('exp=cxib7913:run=34:idx')

for run in ds.runs():
    times = run.times()
    for i in range(3,-1,-1):
        evt=run.event(times[i])
```

Demo of 46-line python script:

- MPI Parallelization
- online/offline analysis
- Real-time plotting
- Running on a laptop
- XTC-file “random access”

How to Avoid Reading Documentation

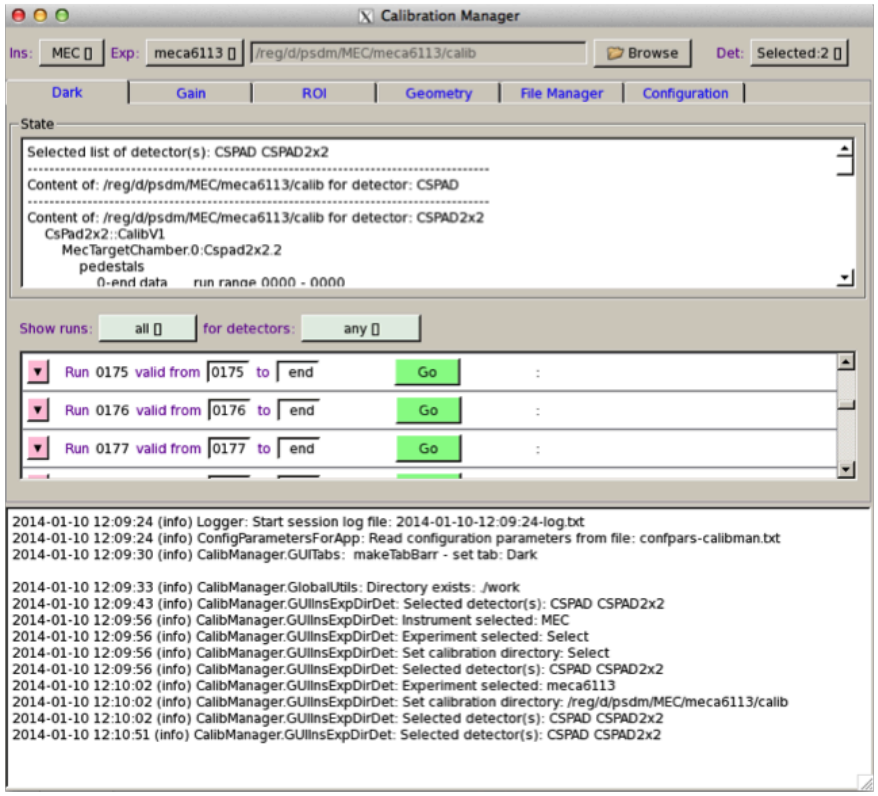
SLAC

What to do if you only know your experiment name and a run number

“ipython” tab-completion demo

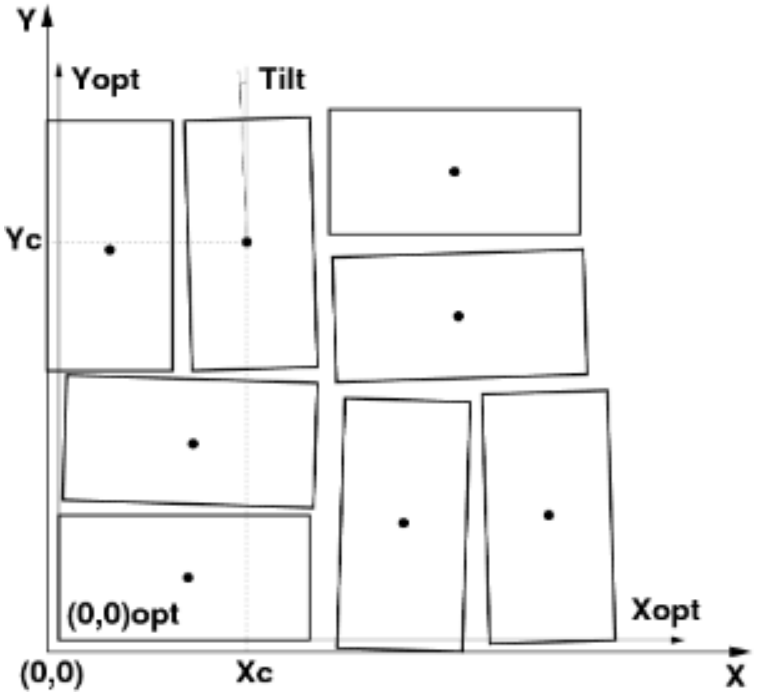
Calibration Support

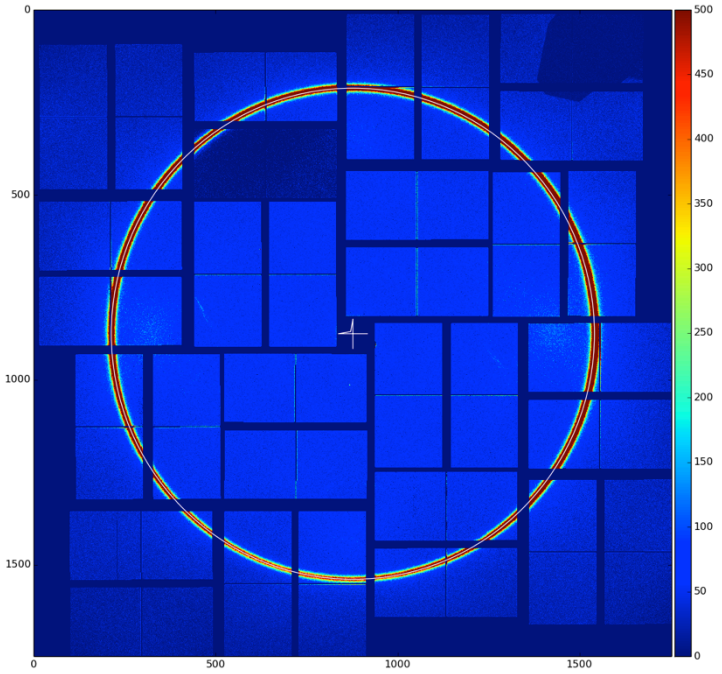
- Psana does: geometry, bad-pixel, common-mode, dark-subtraction, many others.
- Creating and managing run-dependent calibration/geometry constants for many detectors is a challenging problem
- We have both a GUI and a command-line interface for managing these constants
- Every hutch uses this, and the same constants work both offline/online
- New generalized detector handling allows any imaging detector to be “calibratable/viewable” using the same software



Detector Geometry

- Geometry is another difficult problem that would ideally be solved once, and not many times (cheetah, psana, cctbx, crystfel).
- Dream: solve this problem once, not multiple times.
- We would like to automate this process more

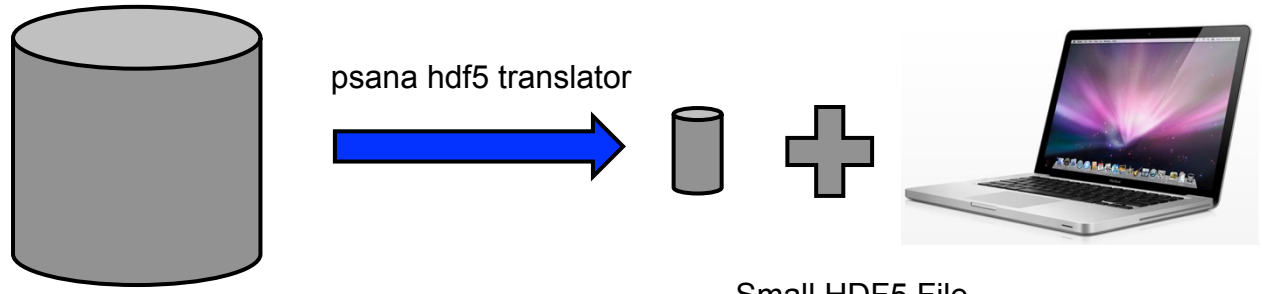




HDF5 Conversion and Custom Data Management

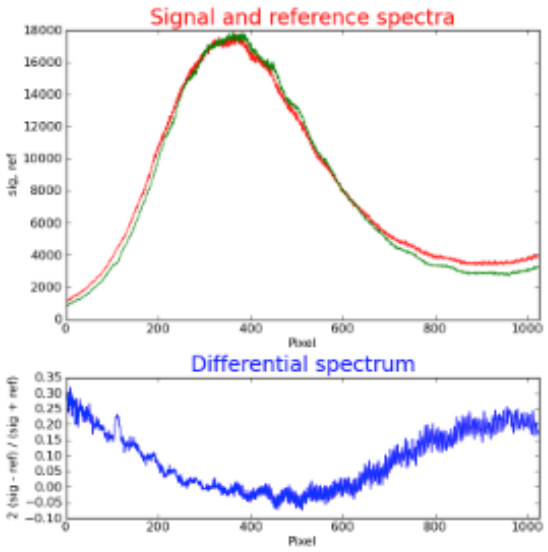
SLAC

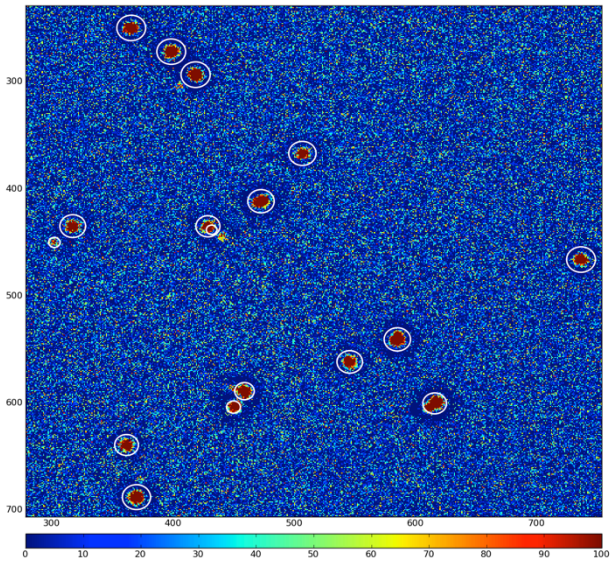
- LCLS users require a portable/standard data format (HDF5)
- psana provides the conversion from LCLS format ("XTC") to HDF5 on-demand
- As a bonus, psana allows users to choose which data they want to convert (including their own processed data)
- They can then move this "standard reduced-size" data to their laptop for further psana analysis (or other tool)



Algorithms

- A lot already available with existing python modules (e.g. scipy)
- Psana has a central “repository” for reusable algorithms (calibration, radial integration, cluster finding)
- many written in C++, now starting repository of python algorithms
- High priority: we are adding more LCLS-specific algorithms





Accessing Data from “Outside” the Hutch

SLAC

- More and more: important devices are not included in the DAQ data. (devices used by multiple hutches are not included!).
- With psana we can now add the “outside hutch” data with the DAQ data.

Summary

SLAC

Major new psana features:

- Same code online/offline
- Easy python scripts
- MPI Parallelization online/offline (several expts have analyzed all 120Hz online)
- Real-time online plotting
- Better detector calibration support
- Random-access to events
- Access to non-DAQ data

Growth areas:

- Algorithms (although scipy provides many)
- Detector Geometry: ideally more reuse (Cheetah, CASS, Psana, cctbx, Crystfel) and more automation. Would be good if we could avoid duplicate effort when new detector (e.g. EPIX) is deployed.
- Easier data access (simpler names, more uniform accessor methods)