Breakout Session 1B: Data acquisition and handling

Organized by Taito Osaka (SACLA)

This session aims to share the current capabilities of data acquisition and handling at SACLA. Using **Python-based APIs developed at SACLA (dbpy, stpy, ippy, ecpy etc.)**, **users can design/code advanced data acquisition and handling processes**, which are not able to be accomplished by standard tools officially supported by SACLA. In addition to overview of these tools, the current status and perspectives on data access environment from your institutes will be presented. Then, some good examples that realized efficient experiments by means of those APIs will be introduced by leading users. Finally, we will discuss **how we can maximize scientific outcomes from the view point of data acquisition / handling capabilities**.

Introduction (20-25 min, recorded and to be uploaded online)

"Efficient experiments at SACLA using Python APIs"

T. Osaka (SACLA)

Facility talk (10 min)

"Current status and perspectives on data access environment"

Y. Joti (SACLA)

Talks of leading users (15 min each)

"Efficient pump–probe experiments"

T. Sato (LCLS)

"Efficient nonlinear X-ray optics experiments"

Z. Abhari (U. Wisconsin–Madison)

Discussion (~10:30 am) **Chair: T. Osaka (SACLA)**

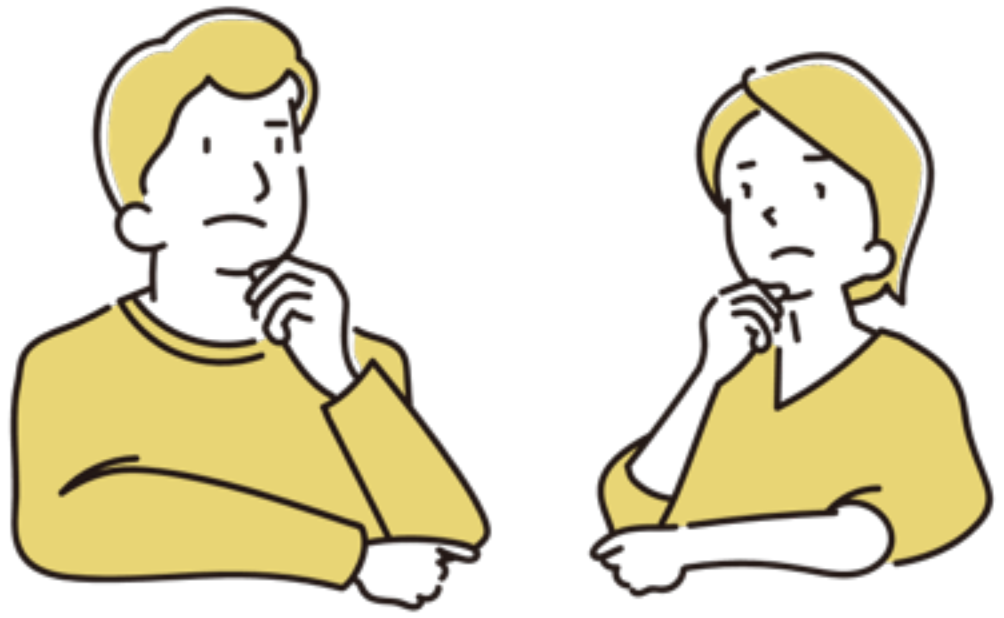
Efficient experiments at SACLA using Python APIs

Taito OSAKA
RIKEN SPring-8 Center

SACLA Users' Meeting 2025
Breakout session 1B: Data acquisition and handling
4th March, 2025

Brief overflow of experiments at SACLA

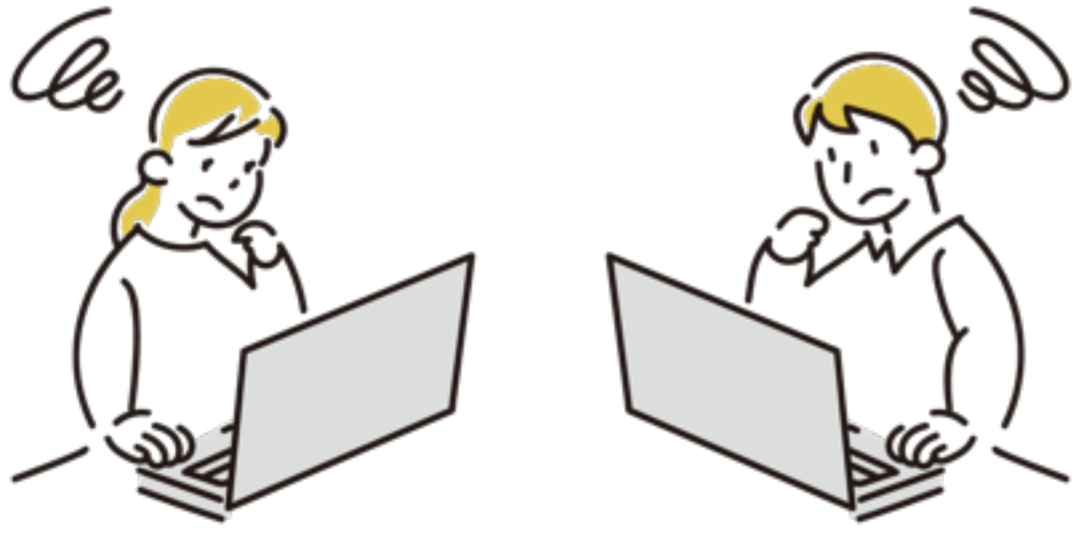
1. Planning methods & procedures



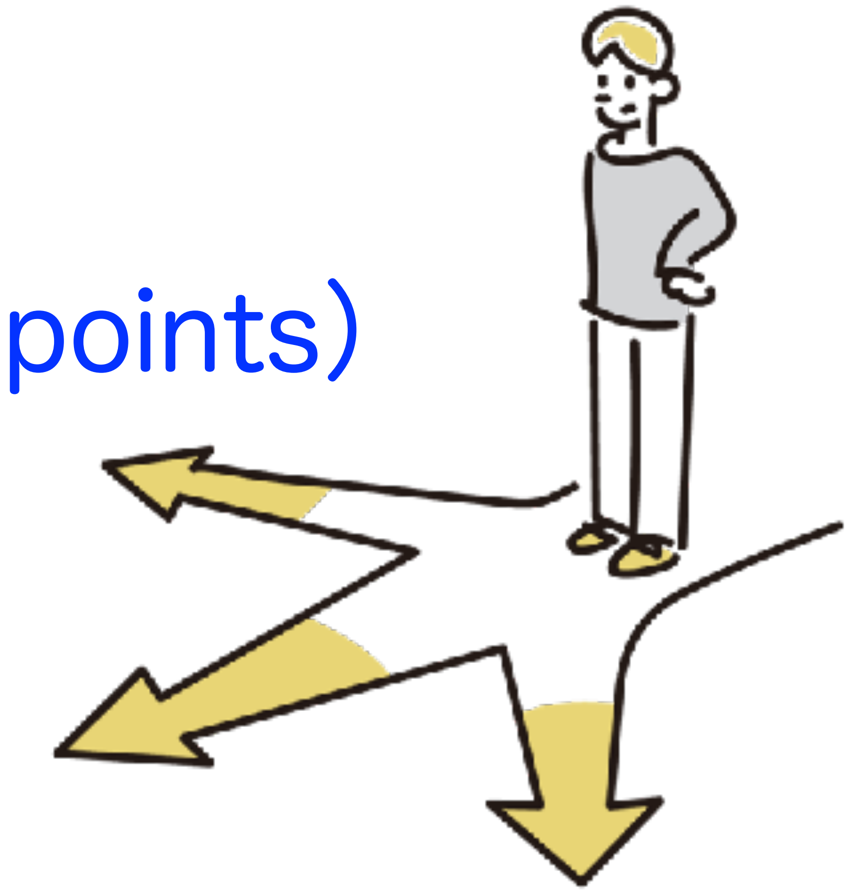
- Prior researches
- Source / BL / detectors
- Remaining time
- Man power

2. Measurements

3. Quick analysis & visualization



- How to analyze / visualize
- Extract important info



Continue
(better statistics, more data points)

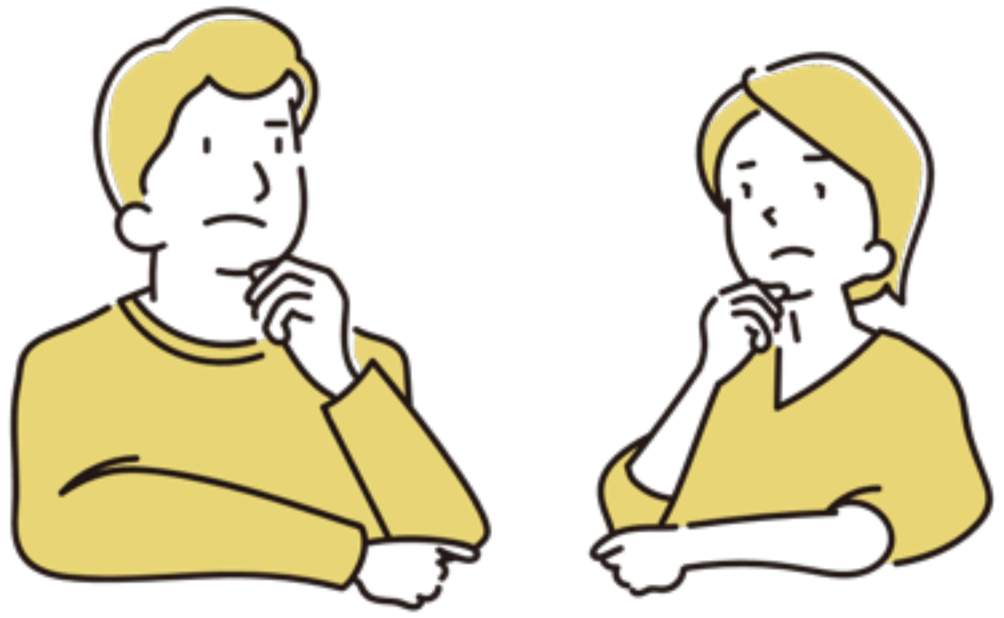
Minor change
(better S/N, new info)

Major change (new info, tests)

Objectives of this talk

②

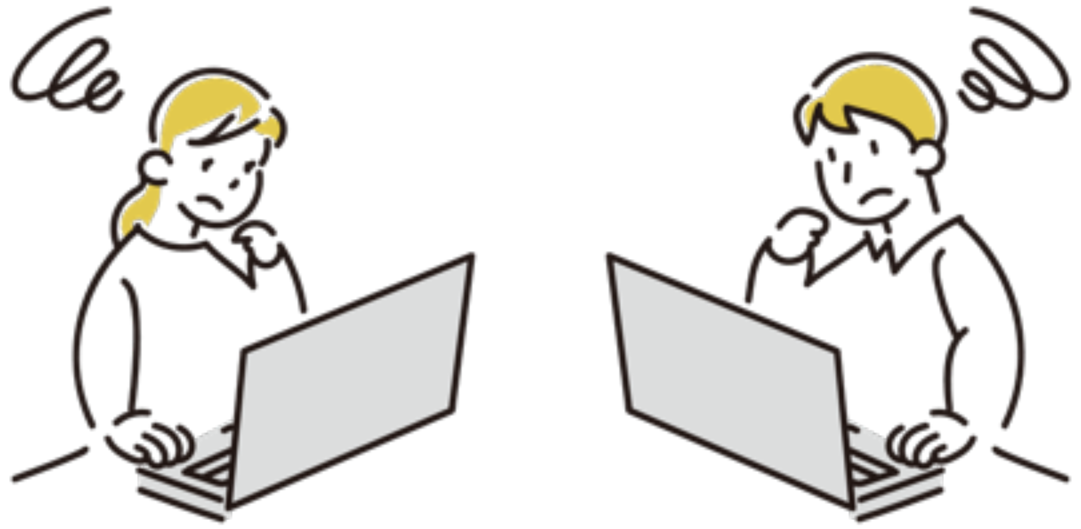
1. Planning methods & procedures



2. Measurements

①

3. Quick analysis & visualization



① Introduce useful tools for data handling / analysis
(dbpy, stpy, ippy)

② Introduce useful tools for data acquisition
(ecpy, ‘semi-’automatic accelerator tuning)

※ For Python users

Useful tools for data handling/analysis

(DataAccessUserAPI_Python: dbpy, stpy)

(ImageProcessingUserAPI_Python: ippy)

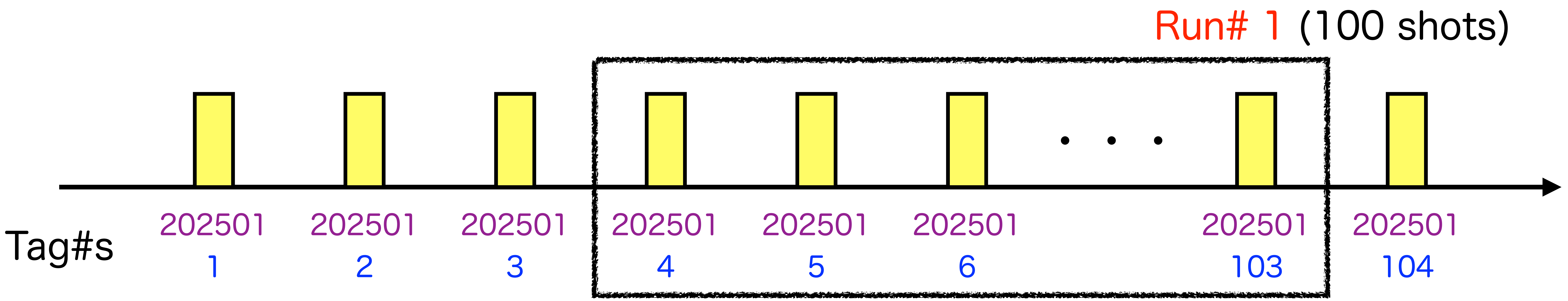
Structure of SACLA data : "Tag" & "Run"

All the XFEL shots are identified by two numbers ("HighTag" & "Tag")

20XX0Y Z

(20XX: FY, 0Y: 01 or 02, Z: 32-bit unsigned integer)

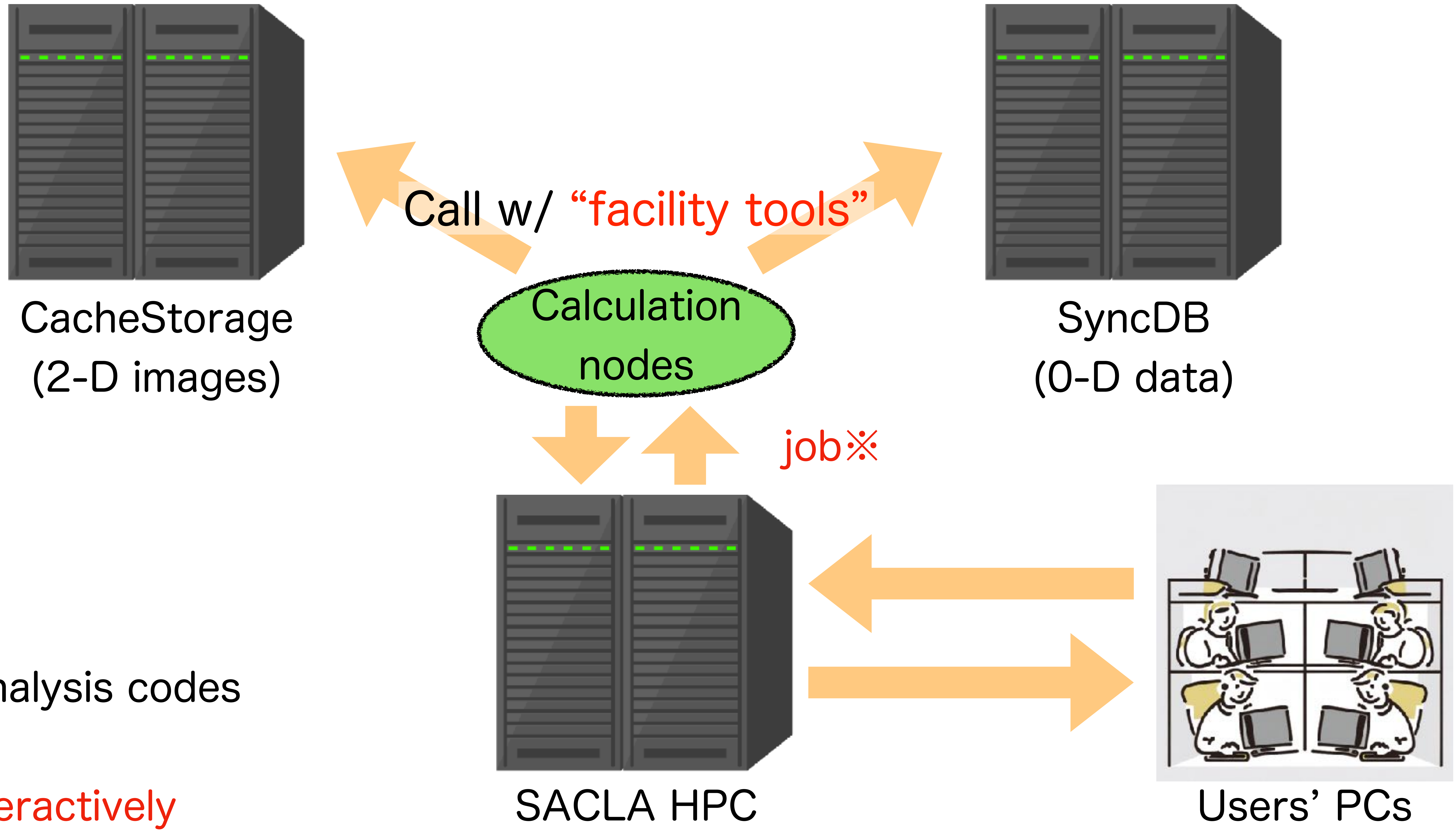
When users store clusters of datasets (mainly for taking 2D images), each cluster is identified by another number, "Run number".



0-D data (PD signals, motor position etc.) : <u>all shots are automatically saved</u> (SyncDB)

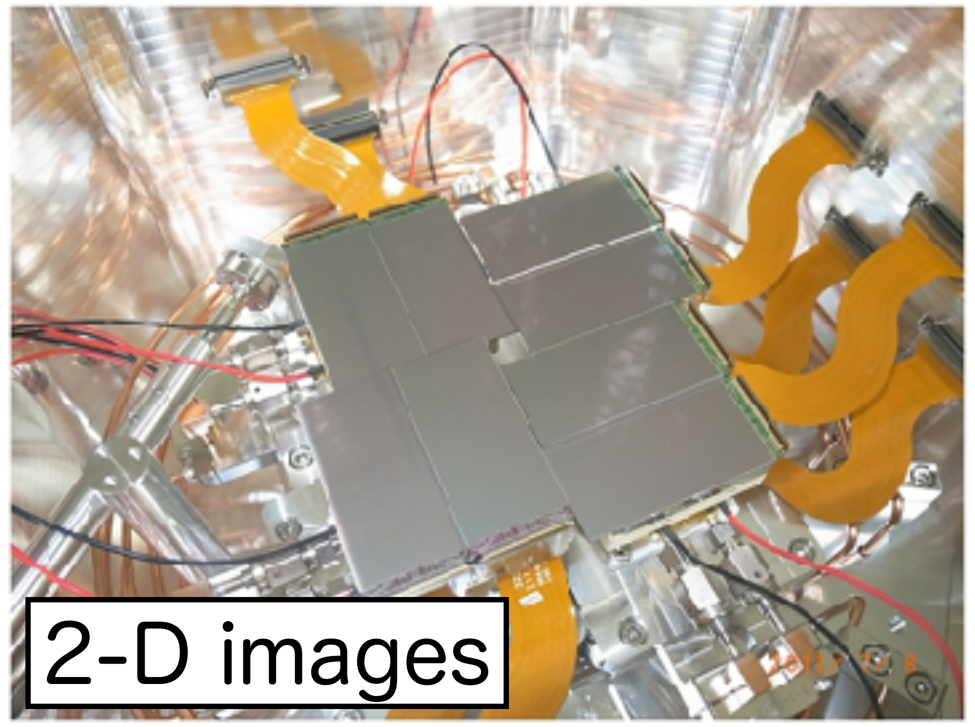
2-D images (MPCCD, Imperx, OPAL etc.) : <u>only shots in each Run are saved</u> (CacheStorage)

Data handling at SACLA



※Execute analysis codes
or
“log-in” interactively

"DataConvert" (standard tool for data handling)

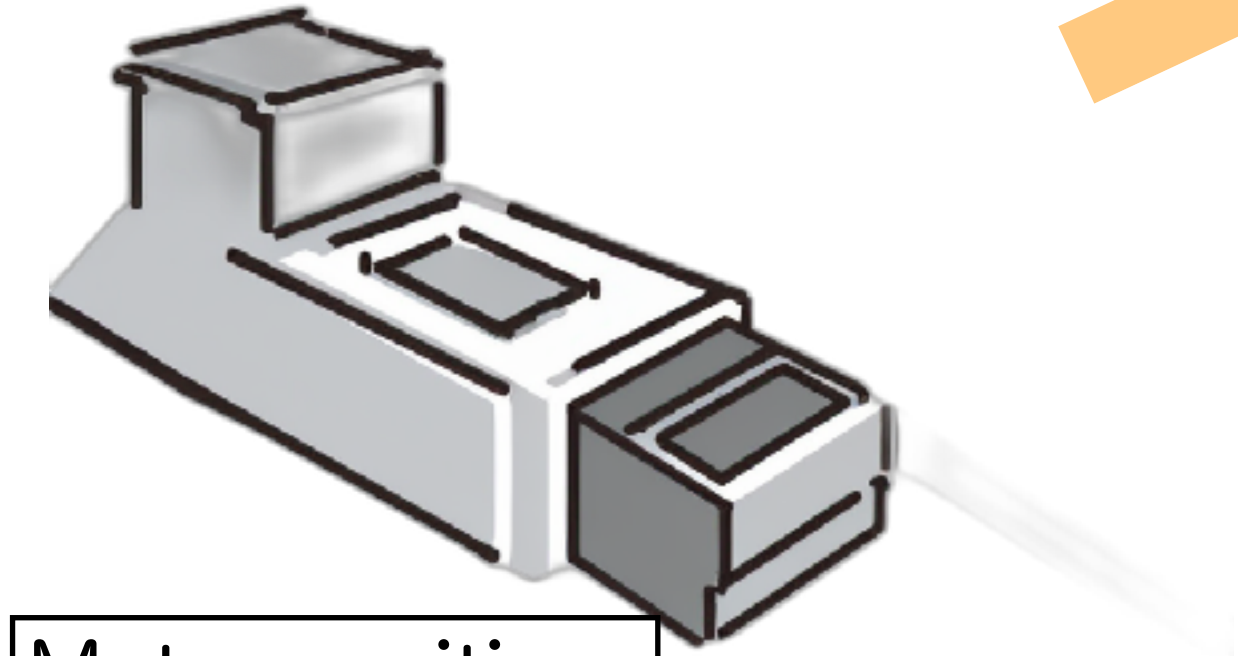


2-D images

Create HDF5 files in which

(pre-processed) 2-D images and 0-D data are contained.

PD signals



Motor positions

Pros:

- ☑ Create accessible 'files' (easy to read & copy to other storages)
- ☑ All the needed data could be contained in a single file
- ☑ Readable by various softwares

Cons:

- Big file size
  (contains many data including unnecessary ones)
- Complicated configuration for selecting data saved in HDF5 files
  (in most cases, users should reset the configuration during or after BT)
- Multiple languages / tools necessary
  for creating files / reading / analyzing / & visualizing them
  (DataConvert is working on Shell, and the other processes need another tool)
- Long pre-processing time for MPCCDs with multiple sensors

Like to complete all the processes with one tool,
while saving time and file size !!

DataAccessUserAPI (dbpy, stpy)

Modules for handling SACLA data via Python

☑ Able to get only specified data as NumPy Array
(0-D data: dbpy, 2-D images: stpy)

☑ Running on a variety of Python versions (2.7, 3.6, 3.7, 3.8 confirmed)

☑ Advanced analysis & visualization possible with established Python modules

☑ Efficient & flexible coding on Jupyter notebook

Who recommended?

- Unclear what info & how analyses are required
  (e.g., need data filtering with some 0-D data but unclear which works well)

- Analyses of a part of MPCCD sensors enough※ (as multi-sensor MPCCDs are used)
  (Assembling multi sensors into a single image takes long time)

- Like to reduce data size
  (only ‘necessary’ data are stored in files)

- Python experts !
  (Only Python is needed)

※Even if you need assembled images,
it is much more efficient to analyze individual sensor images,
and finally assemble them with ‘ippy’

Example (Run info & 0-D data)

Read the newest run number

In [3]:

```
#dbpy.read_runnumber_newest(bl)
#output: newest run number
run_newest = dbpy.read_runnumber_newest(3)
print(run_newest)
```

```
1081050
```

Read the status of the newest Run

In [4]:

```
#dbpy.read_runstatus(bl, run)
#output: -1: not yet exist, 0 = stopped (ready to read), 1: paused, 2: running)
run_newest = dbpy.read_runnumber_newest(3)
run_status = dbpy.read_runstatus(3, run_newest)
print(run_status)
```

Useful for automatic data analysis & visualization

(able to start analyzes soon after the newest Run is completed)

Read Run Info of a specific Run

In [6]:

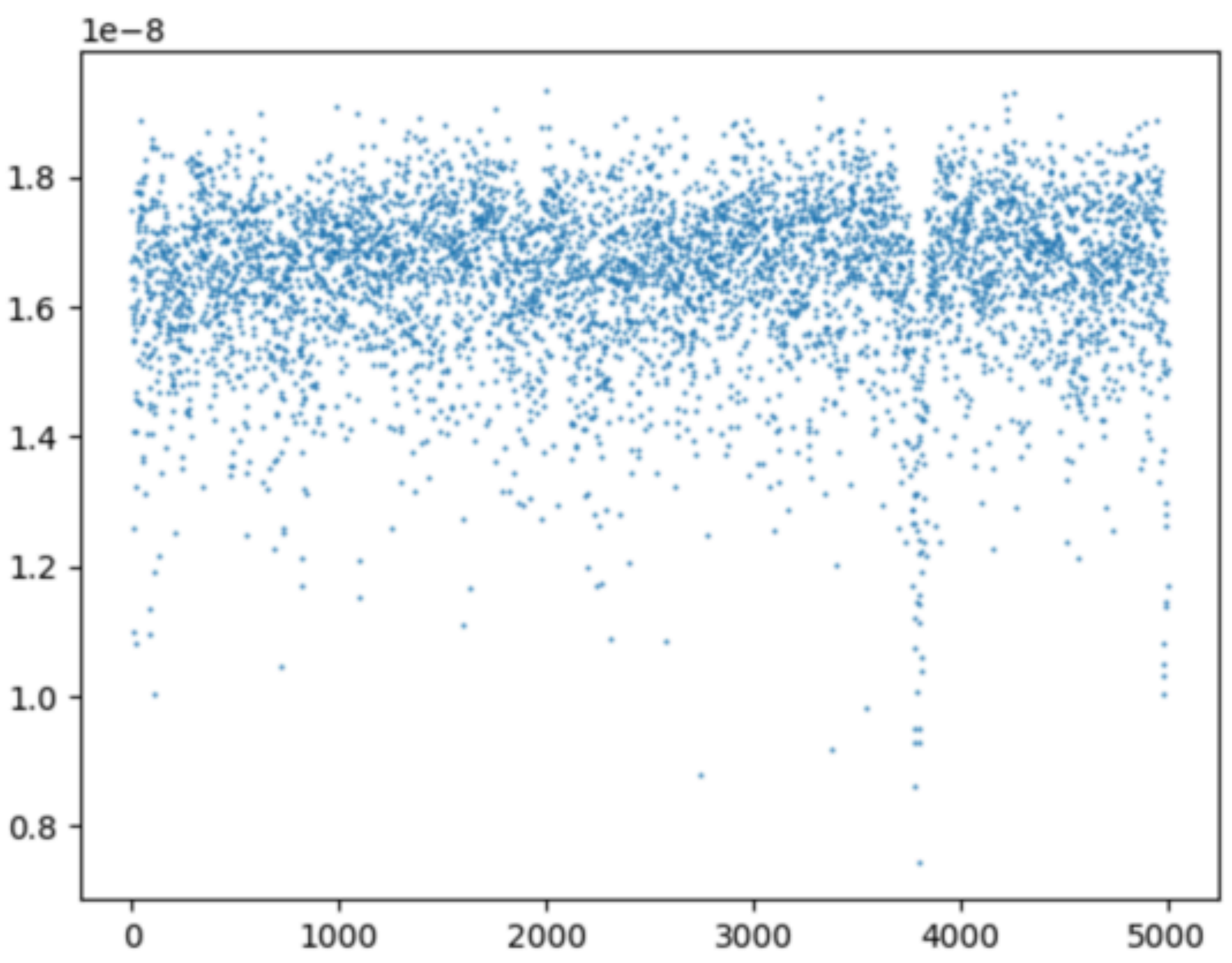
```
#dbpy.read_runinfo(bl, run)
#output: dictionary of run information
run = 651251
run_info = dbpy.read_runinfo(3, run)
print(run_info)
```

```
{'starttime': 1521205345.631467, 'stoptime': 1521205513.202747, 'total_tagnumber': 10000, 'start_tagnumber': 340377562, 'end_tagnumber': 340387562, 'hightagnumber': 201801, 'comment': 'A
CD-2B1-M03-001,stor1_09,stor1_10', 'runtype': '', 'stationnumber': 4, 'runstatus': 0}
```

Read SyncDAQ values

In [7]:

```
equipID = 'xfel_bl_3_tc_bm_1_pd/charge' #BM1 that is shown in the SACLA Operation Status
run = 651251
taglist = dbpy.read_taglist_byrun(3, run)
high_tag = dbpy.read_hightagnumber(3, run)

#dbpy.read_syncdatalist_float(equipID, high_tag, taglist)
#output: tuple of syncDB values
bm1_charge = np.array(dbpy.read_syncdatalist_float(equipID, high_tag, taglist))
plt.plot(bm1_charge, '.', ms=1)
```



- equipID (name of the signal in SyncDB)
- Run number (list of tags can be generated from the Run number)

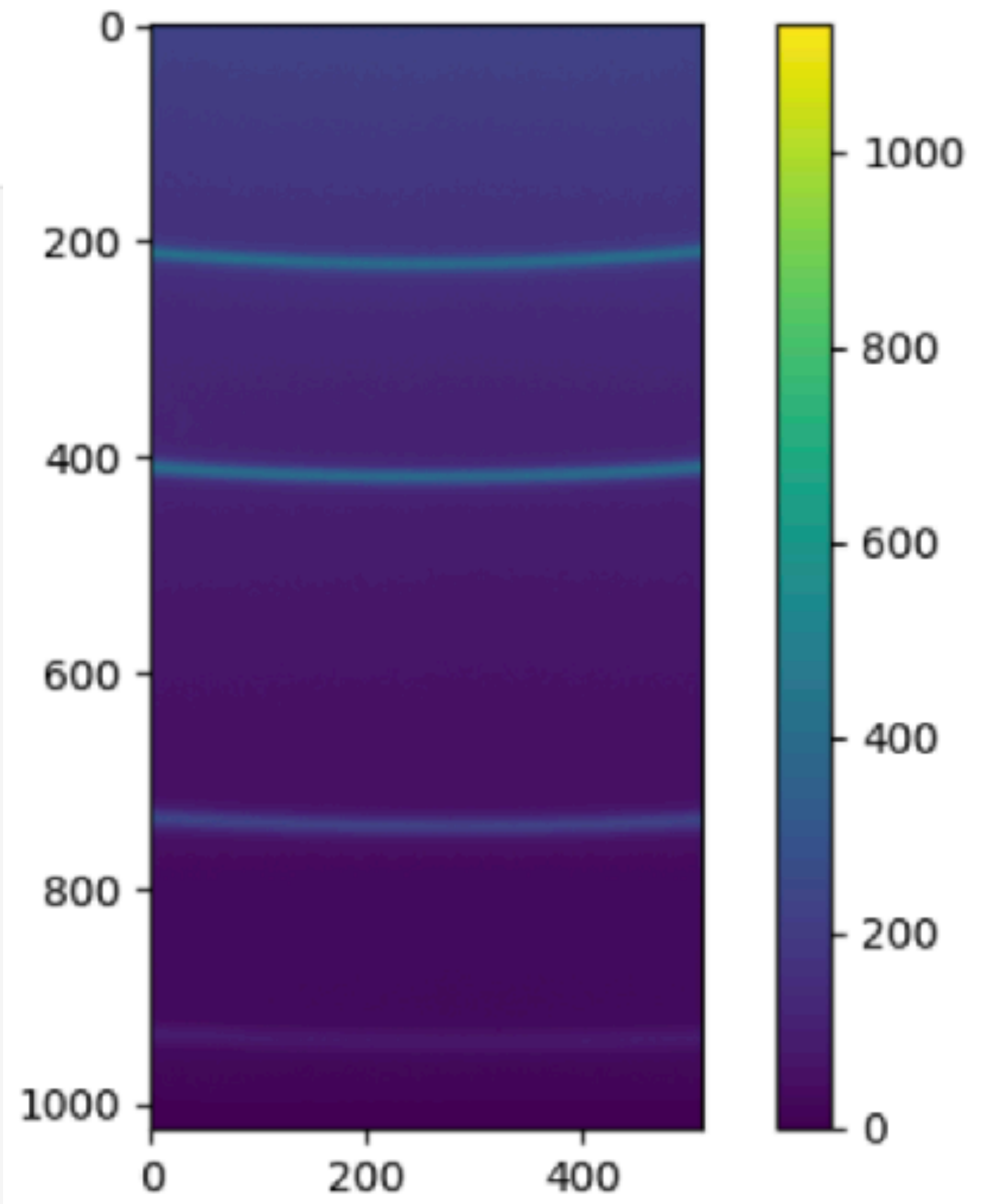
Example (2-D images)

Example of averaging detector images after dark subtraction

In [10]:

```
def read_det_sbt(bl, run, detID, imDark):
    taglist = dbpy.read_taglist_byrun(3, run)
    numIm = len(taglist)
    print('\nRun: {}\nNumber of images: {}\nDetector ID: {}'.format(run, numIm, detID))

    #stpy.StorageReader(detectorID, bl, run_numbers)
    #run_numbers: tuple of run list
    obj = stpy.StorageReader(detID, 3, (run,))
    buff = stpy.StorageBuffer(obj)
    obj.collect(buff, taglist[0])
    im2D = buff.read_det_data(0)

    im2Dall_sbt = np.zeros((numIm, len(im2D[:,0]), len(im2D[0,:])))
    im2Dall_sbt[0] = im2D - imDark

    i = 1
    for tag in taglist[1:]:
        if i % 100 == 0:
            sys.stdout.write('\r%d' % i)
            sys.stdout.flush()
        obj.collect(buff, tag)
        im2Dall_sbt[i] = buff.read_det_data(0) - imDark
        i += 1

    return im2Dall_sbt

detID = 'MPCCD-2B1-M03-001-1'
run = 651236
runDark = 651264

imDark = np.mean(read_det(3, runDark, detID),0)
im2Dall = read_det_sbt(3, run, detID, imDark)
im2Dave = np.mean(im2Dall, 0)

plt.imshow(im2Dave)
plt.colorbar()
```

- detectorID (name of the 2-D detector)
- Run number

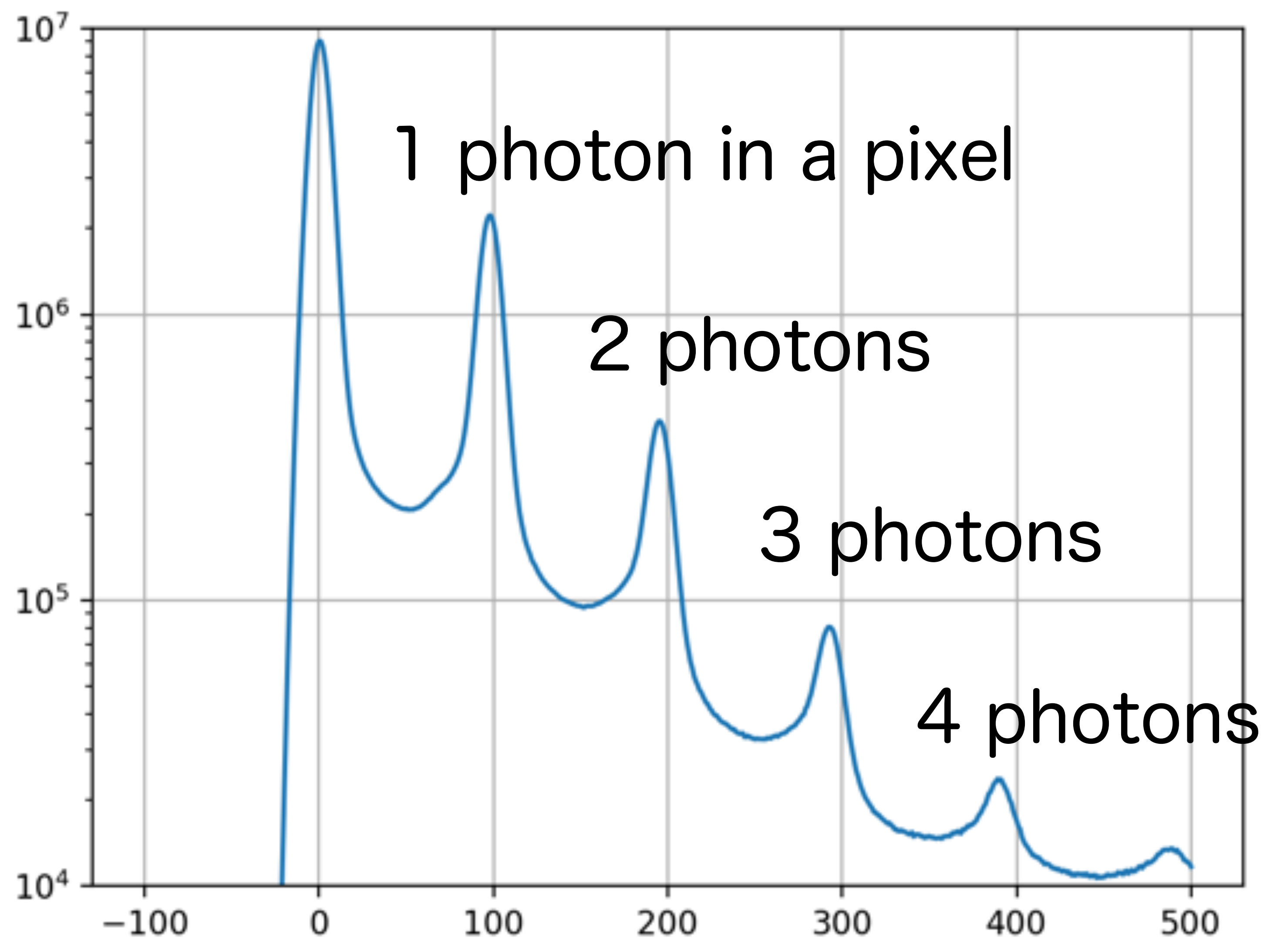


Averaged image of one of MPCCD Dual sensors after dark subtraction

Check single-photon counts on MPCCD

In [15]:

```
detID = 'MPCCD-2N0-M02-001-1'
run = 1360991

im2Dall = read_det_sbt(3, run, detID, imDark)
bins=np.arange(-100,502) - 0.5
hist = np.histogram(im2Dall, bins=bins)[0]

plt.plot(np.arange(-100,501),hist)
plt.grid()
plt.yscale('log')
plt.ylim(1e4,1e7)
```



Histogram of adu values on individual pixels (confirm adu value of a single X-ray photon)

ImageProcessingUserAPI (ippy)

Module for processing MPCCD images via Python

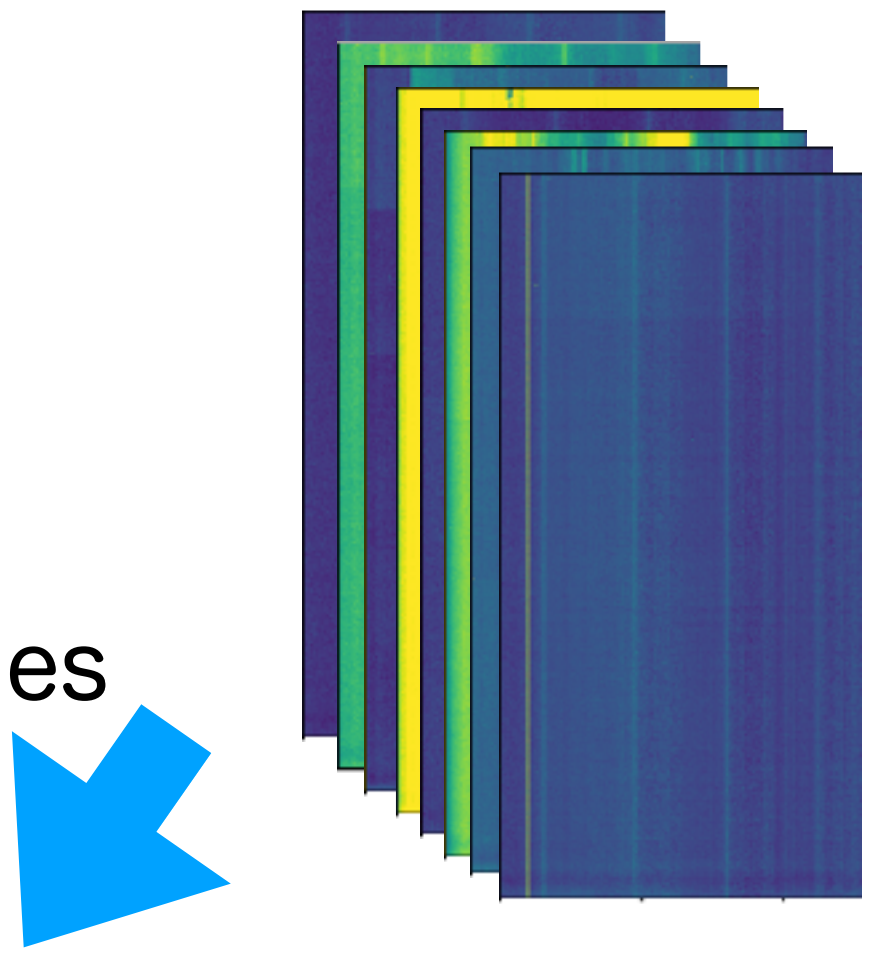
Assemble individual sensor images of MPCCD Octal

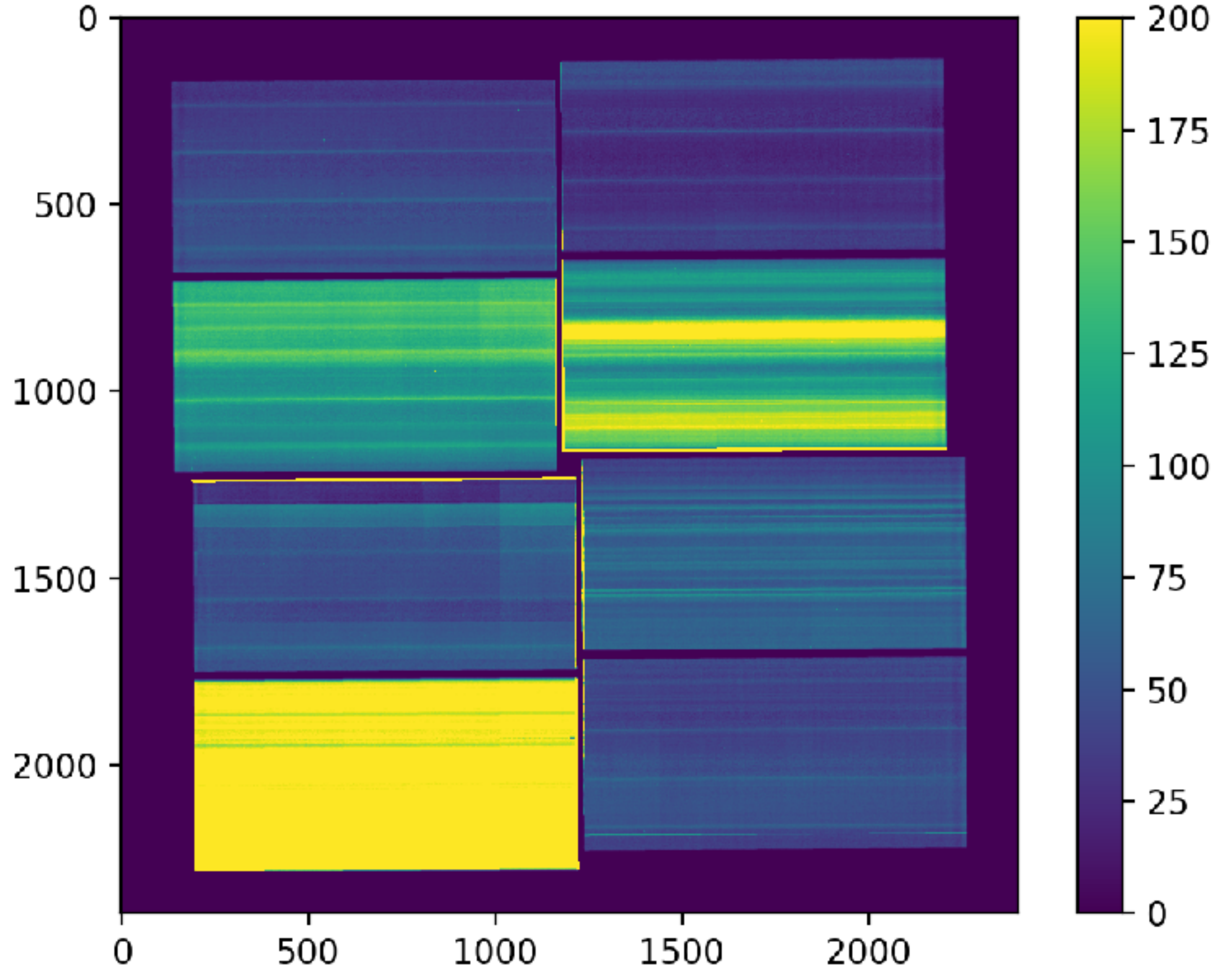
In [18]:

```
run = 220934
taglist = dbpy.read_taglist_byrun(2, run)
srcs = {}
for i in range(8):
    src = {}
    detID = f'MPCCD-8B0-2-008-{i+1}'
    obj = stpy.StorageReader(detID, 2, (run,))
    buff = stpy.StorageBuffer(obj)
    obj.collect(buff, taglist[0])
    src[f'img'] = np.array(buff.read_det_data(0)).copy()
    det_info = buff.read_det_info(0)
    psizex = det_info['mp_pixelsizex']
    psizey = det_info['mp_pixelsizey']
    xsize = det_info['xsize']
    ysize = det_info['ysize']
    posx = det_info['mp_posx']
    posy = det_info['mp_posy']
    rotangle = det_info['mp_rotationangle']

    src['xPosPixel'] = posx/psizex #float
    src['yPosPixel'] = -posy/psizey #the sign of this variable must be -1 x posy
    src['pixelSize'] = round(psizex) #int
    src['dScale'] = 1. #float
    src['rotAngle'] = rotangle #float
    src['mask']= np.ones((ysize,xsize)).astype(np.int16)
    src['succeeded']=True

    if i == 0:
        gain0 = det_info['mp_absgain']
        src['iScale'] = 1.
    else:
        gain = det_info['mp_absgain']
        src['iScale'] = gain / gain0

    srcs[f'det{i+1}'] = src.copy()

srcs_list = []
for i in range(8):
    srcs_list.append(srcs[f'det{i+1}'])

# print(srcs)
asm_img = ippy.reconstruction(srcs_list,2399,2399,50,0)
plt.imshow(asm_img,clim=(0,200))
plt.colorbar()
```

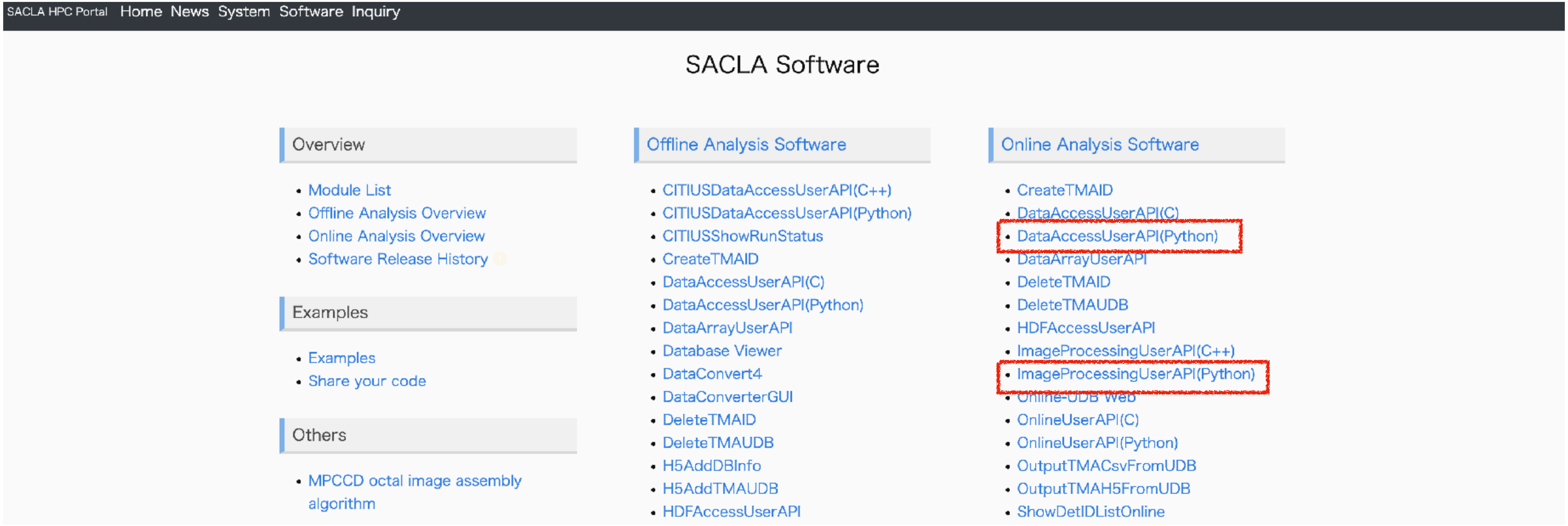
ippy.reconstruction()

Correctly assemble individual sensor images into a single image





References: SACLA HPC Portal



SACLA HPC Portal Home News System Software Inquiry

SACLA Software

Overview

- Module List
- Offline Analysis Overview
- Online Analysis Overview
- Software Release History

Examples

- Examples
- Share your code

Others

- MPCCD octal image assembly algorithm

Offline Analysis Software

- CITIUSDataAccessUserAPI(C++)
- CITIUSDataAccessUserAPI(Python)
- CITIUSShowRunStatus
- CreateTMAID
- DataAccessUserAPI(C)
- DataAccessUserAPI(Python)
- DataArrayUserAPI
- Database Viewer
- DataConvert4
- DataConverterGUI
- DeleteTMAID
- DeleteTMAUDB
- H5AddDBInfo
- H5AddTMAUDB
- HDFAccessUserAPI

Online Analysis Software

- CreateTMAID
- DataAccessUserAPI(C)
- DataAccessUserAPI(Python)
- DataArrayUserAPI
- DeleteTMAID
- DeleteTMAUDB
- HDFAccessUserAPI
- ImageProcessingUserAPI(C++)
- ImageProcessingUserAPI(Python)
- Online-UDB Web
- OnlineUserAPI(C)
- OnlineUserAPI(Python)
- OutputTMACsvFromUDB
- OutputTMAH5FromUDB
- ShowDetIDListOnline

or (if you have an HPC account)

/home/osaka/examples/ExampleForDataAnalysisForPython

(.html or .ipynb)

Useful tool for data acquisition

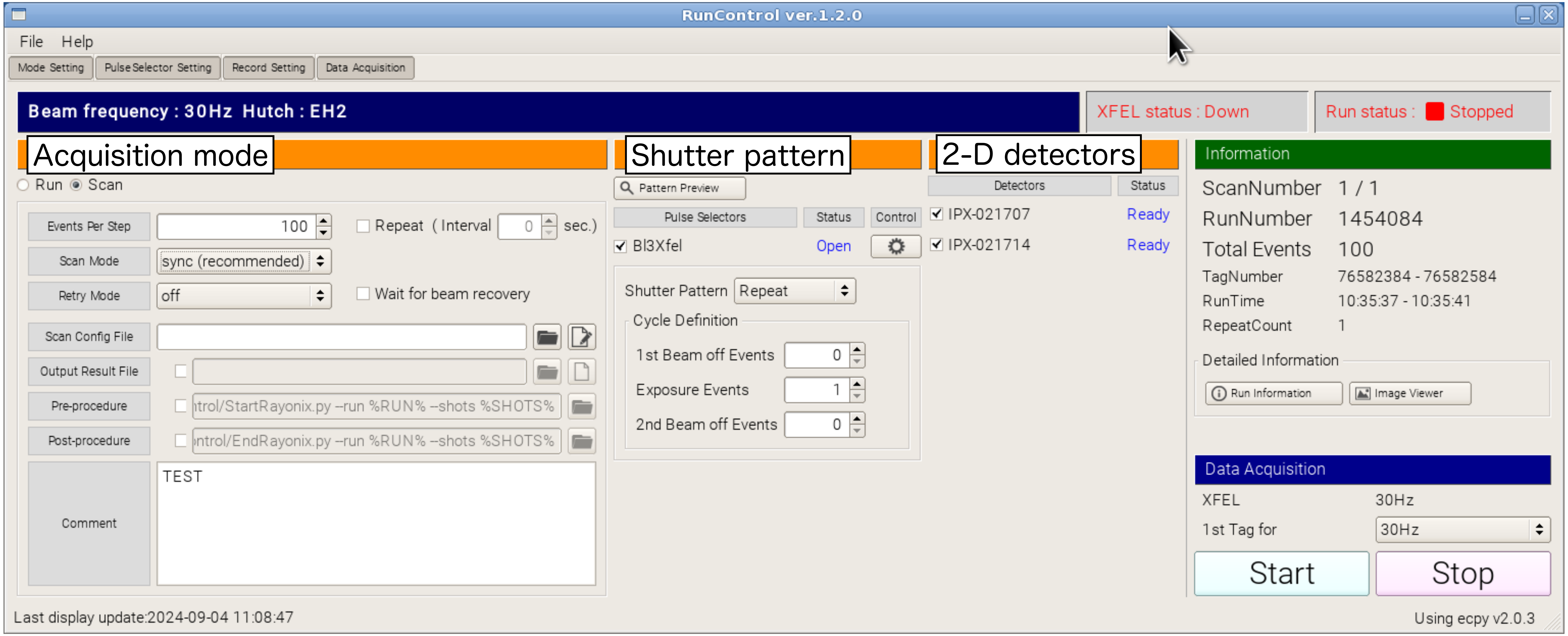
(ExperimentControlAPI：ecpy※)

(‘Semi-’automatic accelerator tuning)

※ ecpy is operational only in OPCONs near hutches for safety

※ecpy is NOT fully opened for users (available after discussion with BL scientists)

RunControlGUI (standard tool for data acquisition)

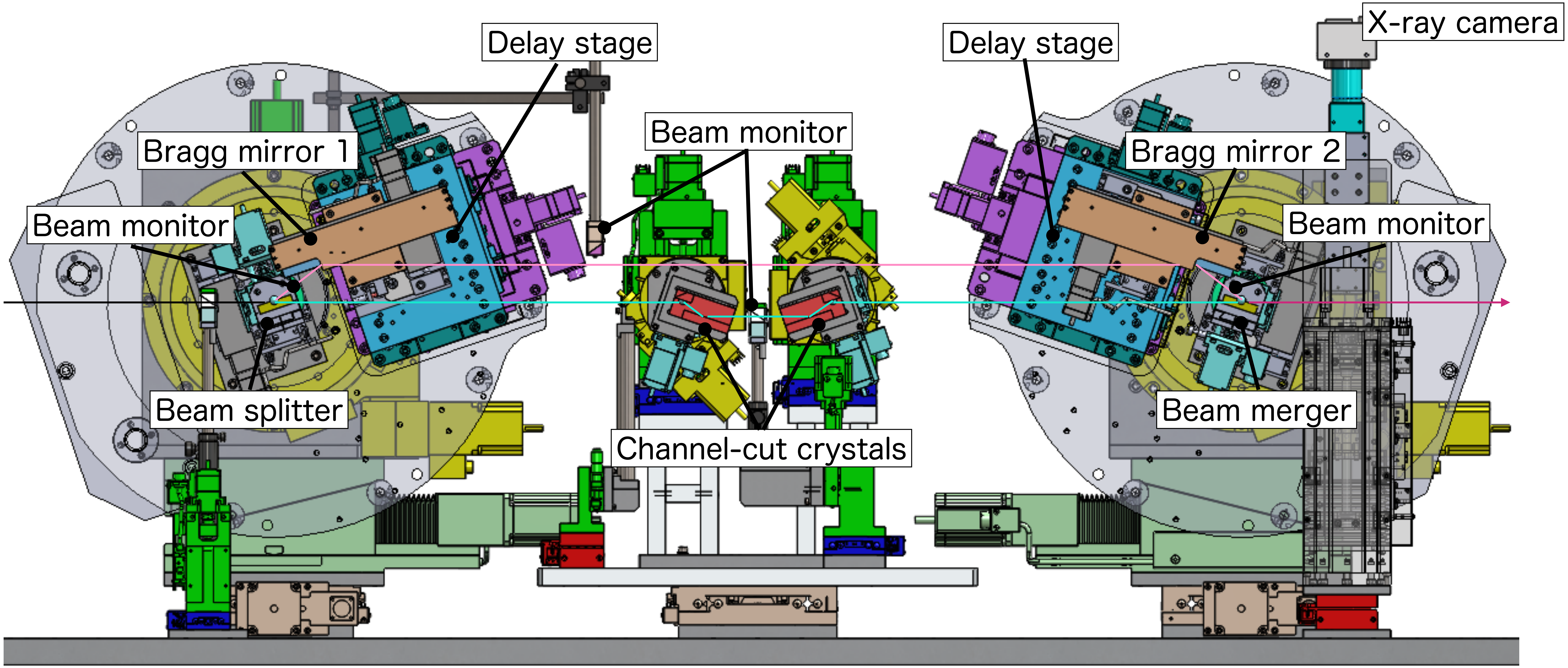


All of what can be done by RunControlGUI (+α) is available with ecpy

(stage control, start & stop Runs, shutter control + setting Amp of PDs etc.)

→ Able to accomplish much more complicated procedures

Example: Control of complex system (Split-Delay Optics)



Generate double FEL pulses with a tunable time delay.

To change a single parameter (time delay, photon energy etc.), multiple stages must be controlled precisely.

Dedicated Python module (sdopy) was coded based on ecpy + dbpy → Easy control by users

Accelerator control

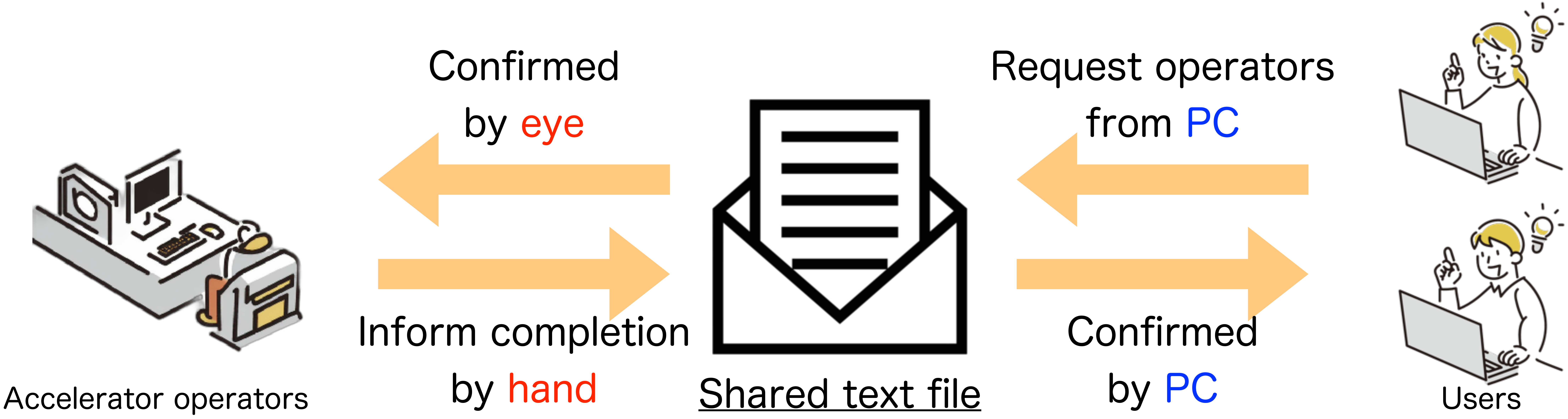
For safety, control of accelerator equipments is inhibited from BLs.

(Only the K value of ID1 is allowed to be controlled)

A useful system has been implemented for ‘semi-’automatic control of accelerator parameters:

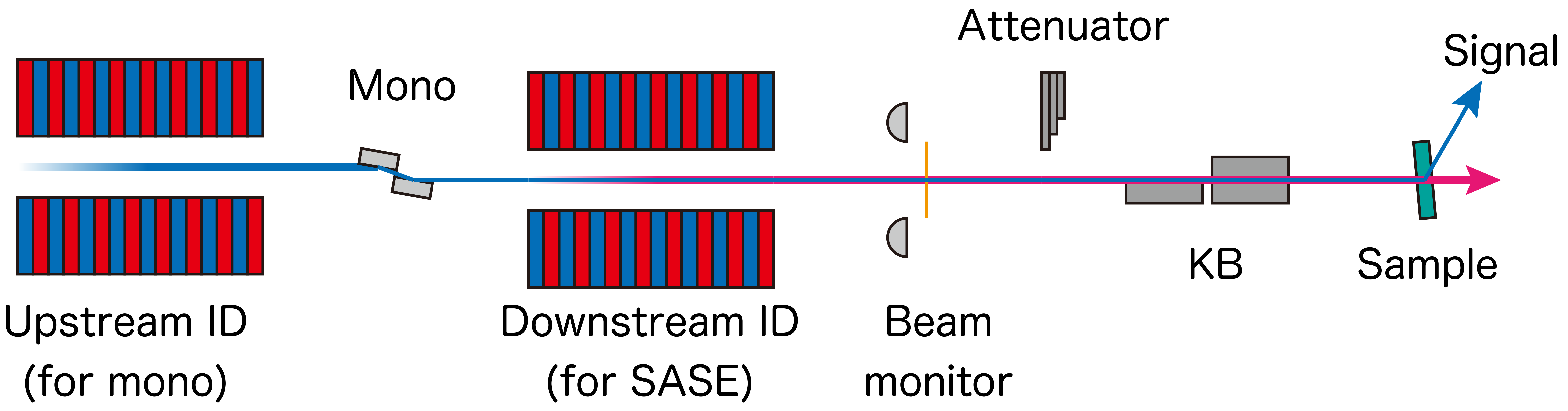
time delay between two-color FEL pulses &

photon energy of self-seeded FELs etc.



Users' example: SASE FEL pump + mono FEL probe exp.

Sample alignment and taking reference (non-pumped) data w/ weak mono beam
→ Take pump-probe data w/ high-intensity FEL pulses



① Move to fresh area in sample
② Stop pump FEL
③ Change gain of beam monitor
④ Insert attenuator
⑤ Rocking curve meas.
⑥ Move to the peak pos.
⑦ Take Run with multiple shots
⑧ Generate pump FEL
⑨ Change gain of beam monitor
⑩ Remove (or change) attenuator
⑪ Take a single-shot Run
⑫ Repeat ①~⑪
⑬ Change time delay
⑭ Repeat ①~⑬

Black : ecpy
Green : Acc. control
Blue : Analysis by dbpy

“Non-official” Python modules (coded by me)

accpy: For ‘semi-’automatic control of accelerator parameters

fspy: For fast delay scans (based on T. Sato’s script)

ccpy: For controlling double channel-cut monochromator

raster: For raster scans of solid samples

Please find more details at

/xdaq/work/share/ecpy_share/

or

contact osaka@spring8.or.jp

In the end,,,

Most of what users want to do is possible via Python APIs at SACLA.
Open OnDemand should facilitate & encourage users to use them.

If you like to use ecpy,
please contact BL scientists as soon as possible.
(all users' requests cannot be covered by facility, due to limited resources,,,)

Users' inputs are always welcome !

Thank you for your attention