In [2]:
```
import warnings

from IPython.display import HTML
import matplotlib
from matplotlib import pyplot as plt
import numpy as np
from matplotlib.cm import viridis
import matplotlib
viridis.set_bad('0.25',1)
matplotlib.rc('image', cmap=viridis, origin='lower')


warnings.filterwarnings("ignore")
%matplotlib inline
```

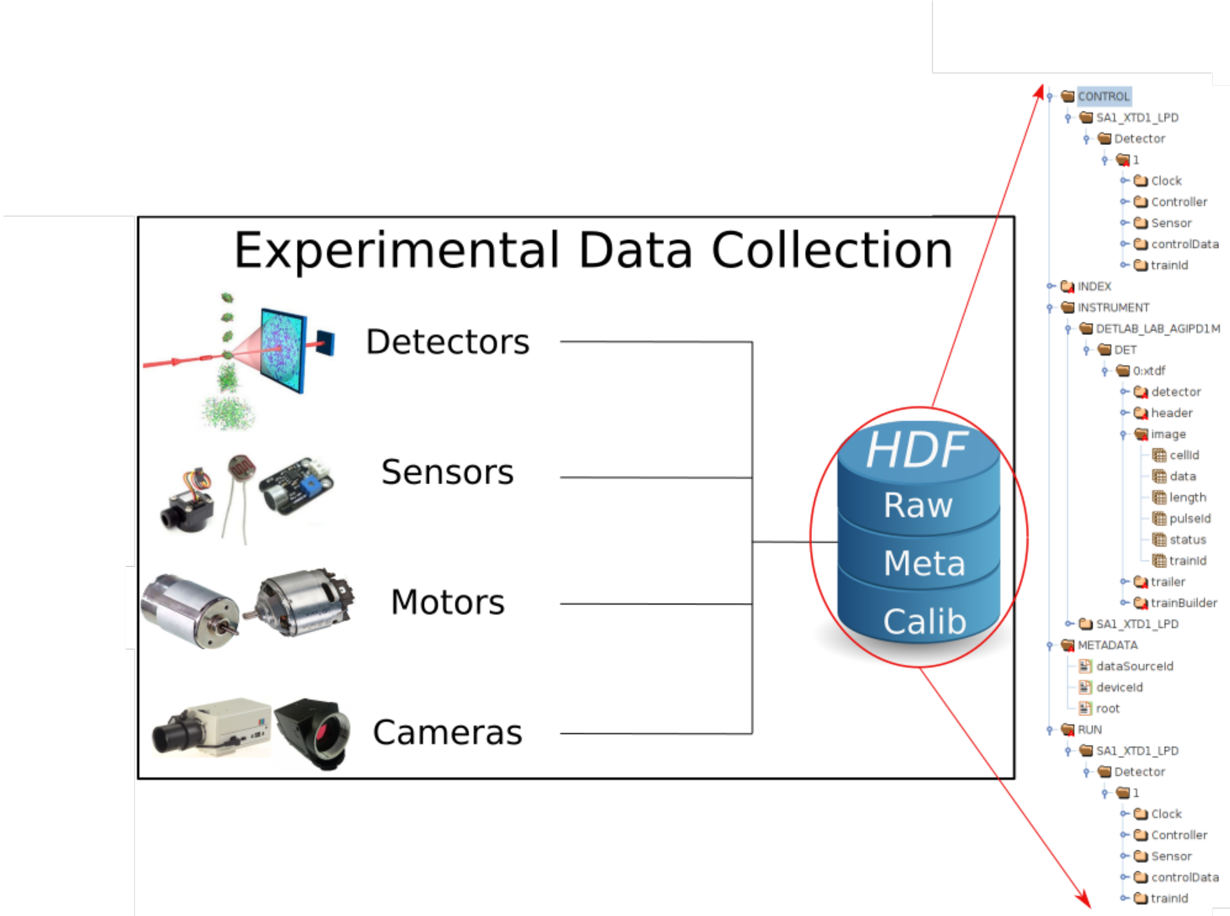
OFFLINE DATA ANALYSIS

THE KARABO-DATA ECOSYSTEM

Martin Bergemann
Control and Analysis Software (CAS)

2019-01-24

European XFEL


Experimental Data Collection
Detectors
Sensors
Motors
Cameras
HDF
Raw
Meta
Calib
CONTROL
SA1_XTD1_LPD
Detector
1
Clock
Controller
Sensor
controlData
trainId
INDEX
INSTRUMENT
DETLAB_LAB_AGIPD1M
DET
0:xtdf
detector
header
image
cellId
data
length
pulseId
status
trainId
trailer
trainBuilder
SA1_XTD1_LPD
METADATA
dataSourceId
deviceId
root
RUN
SA1_XTD1_LPD
Detector
1
Clock
Controller
Sensor
controlData
trainId

```
ls /gpfs/exfel/exp/XMPL/201750/p700000/proc/r0273/
CORR-R0273-AGIPD00-S00000.h5  CORR-R0273-AGIPD05-S00002.h5  CORR-R0273-AGIPD10-S00004.h5
CORR-R0273-AGIPD00-S00001.h5  CORR-R0273-AGIPD05-S00003.h5  CORR-R0273-AGIPD10-S00005.h5
CORR-R0273-AGIPD00-S00002.h5  CORR-R0273-AGIPD05-S00004.h5  CORR-R0273-AGIPD11-S00000.h5
CORR-R0273-AGIPD00-S00003.h5  CORR-R0273-AGIPD05-S00005.h5  CORR-R0273-AGIPD11-S00001.h5
CORR-R0273-AGIPD00-S00004.h5  CORR-R0273-AGIPD06-S00000.h5  CORR-R0273-AGIPD11-S00002.h5
CORR-R0273-AGIPD00-S00005.h5  CORR-R0273-AGIPD06-S00001.h5  CORR-R0273-AGIPD11-S00003.h5
CORR-R0273-AGIPD01-S00000.h5  CORR-R0273-AGIPD06-S00002.h5  CORR-R0273-AGIPD11-S00004.h5
CORR-R0273-AGIPD01-S00001.h5  CORR-R0273-AGIPD06-S00003.h5  CORR-R0273-AGIPD11-S00005.h5
CORR-R0273-AGIPD01-S00002.h5  CORR-R0273-AGIPD06-S00004.h5  CORR-R0273-AGIPD12-S00000.h5
CORR-R0273-AGIPD01-S00003.h5  CORR-R0273-AGIPD06-S00005.h5  CORR-R0273-AGIPD12-S00001.h5
CORR-R0273-AGIPD01-S00004.h5  CORR-R0273-AGIPD07-S00000.h5  CORR-R0273-AGIPD12-S00002.h5
CORR-R0273-AGIPD01-S00005.h5  CORR-R0273-AGIPD07-S00001.h5  CORR-R0273-AGIPD12-S00003.h5
CORR-R0273-AGIPD02-S00000.h5  CORR-R0273-AGIPD07-S00002.h5  CORR-R0273-AGIPD12-S00004.h5
CORR-R0273-AGIPD02-S00001.h5  CORR-R0273-AGIPD07-S00003.h5  CORR-R0273-AGIPD12-S00005.h5
CORR-R0273-AGIPD02-S00002.h5  CORR-R0273-AGIPD07-S00004.h5  CORR-R0273-AGIPD13-S00000.h5
CORR-R0273-AGIPD02-S00003.h5  CORR-R0273-AGIPD07-S00005.h5  CORR-R0273-AGIPD13-S00001.h5
CORR-R0273-AGIPD02-S00004.h5  CORR-R0273-AGIPD08-S00000.h5  CORR-R0273-AGIPD13-S00002.h5
CORR-R0273-AGIPD02-S00005.h5  CORR-R0273-AGIPD08-S00001.h5  CORR-R0273-AGIPD13-S00003.h5
CORR-R0273-AGIPD03-S00000.h5  CORR-R0273-AGIPD08-S00002.h5  CORR-R0273-AGIPD13-S00004.h5
CORR-R0273-AGIPD03-S00001.h5  CORR-R0273-AGIPD08-S00003.h5  CORR-R0273-AGIPD13-S00005.h5
CORR-R0273-AGIPD03-S00002.h5  CORR-R0273-AGIPD08-S00004.h5  CORR-R0273-AGIPD14-S00000.h5
CORR-R0273-AGIPD03-S00003.h5  CORR-R0273-AGIPD08-S00005.h5  CORR-R0273-AGIPD14-S00001.h5
CORR-R0273-AGIPD03-S00004.h5  CORR-R0273-AGIPD09-S00000.h5  CORR-R0273-AGIPD14-S00002.h5
CORR-R0273-AGIPD03-S00005.h5  CORR-R0273-AGIPD09-S00001.h5  CORR-R0273-AGIPD14-S00003.h5
CORR-R0273-AGIPD04-S00000.h5  CORR-R0273-AGIPD09-S00002.h5  CORR-R0273-AGIPD14-S00004.h5
CORR-R0273-AGIPD04-S00001.h5  CORR-R0273-AGIPD09-S00003.h5  CORR-R0273-AGIPD14-S00005.h5
CORR-R0273-AGIPD04-S00002.h5  CORR-R0273-AGIPD09-S00004.h5  CORR-R0273-AGIPD15-S00000.h5
CORR-R0273-AGIPD04-S00003.h5  CORR-R0273-AGIPD09-S00005.h5  CORR-R0273-AGIPD15-S00001.h5
CORR-R0273-AGIPD04-S00004.h5  CORR-R0273-AGIPD10-S00000.h5  CORR-R0273-AGIPD15-S00002.h5
CORR-R0273-AGIPD04-S00005.h5  CORR-R0273-AGIPD10-S00001.h5  CORR-R0273-AGIPD15-S00003.h5
CORR-R0273-AGIPD05-S00000.h5  CORR-R0273-AGIPD10-S00002.h5  CORR-R0273-AGIPD15-S00004.h5
CORR-R0273-AGIPD05-S00001.h5  CORR-R0273-AGIPD10-S00003.h5  CORR-R0273-AGIPD15-S00005.h5
```

A Closer Look at Detector Data:

Data from Modular Detectors is stored across multiple files

In [4]:
```
!lsxfel /gpfs/exfel/exp/XMPL/201750/p700000/raw/r0273
```

```
r0273 : Run directory

# of trains:    156
Duration:       0:00:15.500000
First train ID: 198425241
Last train ID:  198425396

16 detector modules (SPB_DET_AGIPD1M-1)
  e.g. module SPB_DET_AGIPD1M-1 0 : 512 x 128 pixels
  176 frames per train, 27456 total frames

2 instrument sources (excluding detectors):
  - SA1_XTD2_XGM/XGM/DOOCS:output
  - SPB_XTD9_XGM/XGM/DOOCS:output

13 control sources:
  - ACC_SYS_DOOCS/CTRL/BEAMCONDITIONS
  - SA1_XTD2_XGM/XGM/DOOCS
  - SPB_IRU_AGIPD1M/PSC/HV
  - SPB_IRU_AGIPD1M/TSENS/H1_T_EXTHOUS
  - SPB_IRU_AGIPD1M/TSENS/H2_T_EXTHOUS
  - SPB_IRU_AGIPD1M/TSENS/Q1_T_BLOCK
  - SPB_IRU_AGIPD1M/TSENS/Q2_T_BLOCK
  - SPB_IRU_AGIPD1M/TSENS/Q3_T_BLOCK
  - SPB_IRU_AGIPD1M/TSENS/Q4_T_BLOCK
  - SPB_IRU_AGIPD1M1/CTRL/MC1
  - SPB_IRU_AGIPD1M1/CTRL/MC2
  - SPB_IRU_VAC/GAUGE/GAUGE_FR_6
  - SPB_XTD9_XGM/XGM/DOOCS
```

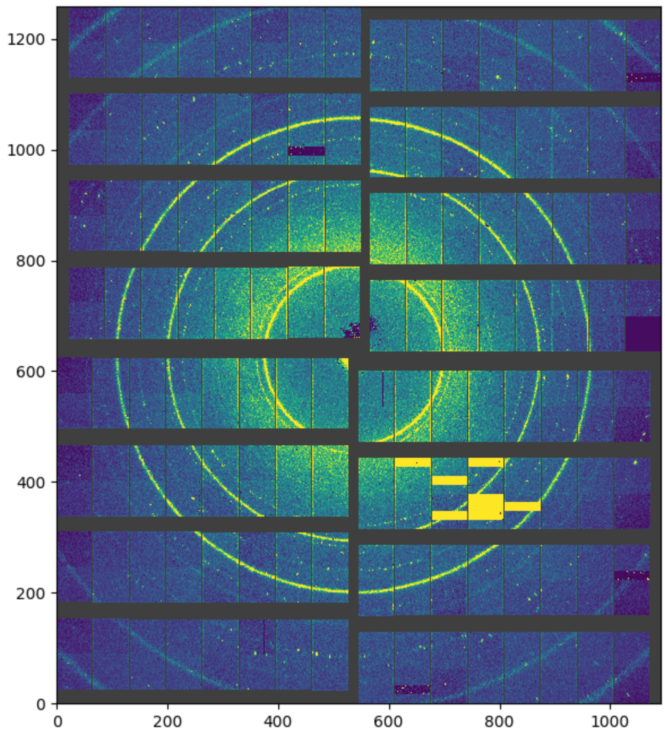
Using standard hdf5 tools can be tricky and tedious

karabo-data

- Python Library that supports Analysis of EuXFEL Data.
- It is Open Source and available on the maxwell cluster @ desy.

Aim of this presentation: Give you a glimpse of what is possible and provide a ground base for your own data analysis at EuXFEL

Scenario I: Plot some detector data



An image from a single module of the AGIP-D Detector at SPB

- How could this data be retrieved and visualized?

Reading a run directory:

One of the big advantages of karabo-data is that whole runs can be read with only *one* command:

In [3]:
```
import karabo_data as kd
run_dir = '/gpfs/exfel/exp/XMPL/201750/p700000/proc/r0273'
run_data = kd.RunDirectory(run_dir)
```

Data can be accessed by:

- selecting trains by id's → `.train_from_id`
- selecting trains by indexes → `.train_from_index`
- iteration (looping) over trains → `.trains`

Let's select data based on indexes:

In [4]:
```
train_id, train_data = run_data.train_from_index(10)
train_id, type(train_data)
```

Out[4]:
```
(198425251, dict)
```

The data is stored in a so called *dictionary*. Hence the data can be accessed by giving keys:

In [5]:
```
sorted(train_data.keys())
```

Out[5]:
```
['SPB_DET_AGIPD1M-1/DET/0CH0:xtdf',
 'SPB_DET_AGIPD1M-1/DET/10CH0:xtdf',
 'SPB_DET_AGIPD1M-1/DET/11CH0:xtdf',
 'SPB_DET_AGIPD1M-1/DET/12CH0:xtdf',
 'SPB_DET_AGIPD1M-1/DET/13CH0:xtdf',
 'SPB_DET_AGIPD1M-1/DET/14CH0:xtdf',
 'SPB_DET_AGIPD1M-1/DET/15CH0:xtdf',
 'SPB_DET_AGIPD1M-1/DET/1CH0:xtdf',
 'SPB_DET_AGIPD1M-1/DET/2CH0:xtdf',
 'SPB_DET_AGIPD1M-1/DET/3CH0:xtdf',
 'SPB_DET_AGIPD1M-1/DET/4CH0:xtdf',
 'SPB_DET_AGIPD1M-1/DET/5CH0:xtdf',
 'SPB_DET_AGIPD1M-1/DET/6CH0:xtdf',
 'SPB_DET_AGIPD1M-1/DET/7CH0:xtdf',
 'SPB_DET_AGIPD1M-1/DET/8CH0:xtdf',
 'SPB_DET_AGIPD1M-1/DET/9CH0:xtdf']
```

The properties of the above shown devices are can be accessed by selecting one of the names (keys):

In [6]:
```
module_data = train_data['SPB_DET_AGIPD1M-1/DET/0CH0:xtdf']
sorted(module_data.keys())
```

Out[6]:
```
['detector.data',
 'detector.trainId',
 'header.dataId',
 'header.linkId',
 'header.magicNumberBegin',
 'header.majorTrainFormatVersion',
 'header.minorTrainFormatVersion',
 'header.pulseCount',
 'header.reserved',
 'header.trainId',
 'image.cellId',
 'image.data',
 'image.gain',
 'image.length',
 'image.mask',
 'image.pulseId',
 'image.status',
 'image.trainId',
 'metadata',
 'trailer.checksum',
 'trailer.magicNumberEnd',
 'trailer.status',
 'trailer.trainId']
```

The actual detector data is stored under the property value *image.data*

In [7]:
```
img_ary = module_data['image.data']
img_ary
```

```
Out[7]: array([[[ 19.061886  ,  27.452316  ,  37.597824  , ...,   5.4725885 ,
                  -7.8149734 , -25.363352  ],
                [ 33.70245   ,  22.299713  ,   0.93073153, ...,  -4.7377877 ,
                 -27.944302  ,  -5.8846965 ],
                [  2.7929337 ,  31.081982  ,  23.773945  , ...,  -3.6882186 ,
                   1.8594275 ,  -9.089156  ],
                ...,
                [ -4.8543234 ,  -8.909287  ,  -8.339569  , ...,  24.457874  ,
                  43.014267  ,  37.71819   ],
                [  6.3957534 ,  -1.3840029 ,   7.470259  , ...,  24.279312  ,
                  39.219994  ,  47.97113   ],
                [ -5.3256435 ,  -8.172988  ,  18.213696  , ...,  37.48085   ,
                  45.80402   ,  87.84654   ]],

               [[185.36618   , 284.957     , 235.30476   , ..., 183.23979   ,
                 160.8983    , 121.14058   ],
                [274.3449    , 209.62767   , 173.04436   , ..., 141.82707   ,
                  63.43843   , 169.06215   ],
                [188.98767   , 250.20027   , 212.46991   , ...,  52.332073  ,
                 122.65631   , 133.87366   ],
                ...,
                [228.12357   , 198.33406   , 149.30414   , ..., 161.04172   ,
                 184.60376   , 191.79622   ],
                [295.8046    , 251.95976   ,  82.540054  , ..., 358.52158   ,
                 212.06604   , 428.72366   ],
                [327.44974   , 183.78809   , 348.145     , ..., 239.44443   ,
                 223.869     , -13.2062025 ]],

               [[350.2579    , 363.42407   , 374.40512   , ..., 254.54288   ,
                 383.2801    , 405.62738   ],
                [190.35628   , 185.2005    , 260.0179    , ..., 225.84837   ,
                 248.45126   , 335.21387   ],
                [181.68295   , 142.093     , 167.58374   , ..., 325.80933   ,
                 369.17328   , 389.54196   ],
                ...,
                [369.41043   , 337.76416   , 527.0745    , ..., 369.6595    ,
                 283.05963   , 485.79013   ],
                [555.3881    , 355.8697    , 286.35693   , ..., 543.9558    ,
                 389.25992   , 431.60147   ],
                [495.20718   , 418.86093   , 355.20557   , ..., 459.5037    ,
                 441.35126   , 443.27362   ]],

               ...,

               [[ 12.515985  ,  32.807766  ,  17.720337  , ..., 116.88453   ,
                 120.97933   , 109.11631   ],
                [  5.1309557 ,  23.454872  ,  10.613807  , ...,  87.08459   ,
                  87.28112   ,  99.29501   ],
                [  6.67852   ,   8.346293  ,  11.348902  , ...,  91.26027   ,
                  81.58455   , 105.83609   ],
                ...,
                [138.54237   , 125.0114    ,  97.77815   , ..., 146.52628   ,
                 149.10728   , 167.16751   ],
                [129.00229   ,  91.16923   ,  92.66123   , ..., 161.34824   ,
                 153.23645   , 151.73201   ],
                [129.10573   , 111.126595  , 107.48337   , ..., 151.1517    ,
                 161.1087    , 317.03162   ]],

               [[ 17.032906  ,  23.841585  ,  12.79358   , ..., 121.621025  ,
                 120.05122   , 112.59217   ],
                [ -1.0304956 ,   9.197788  ,   2.0586886 , ...,  82.65873   ,
                 101.2338   , 108.61823   ],
                [ -9.271917  ,   1.0464283 ,  13.519192  , ...,  92.814224  ,
                 112.34066   , 119.469475  ],
                ...,
                [131.70792   , 132.71964   , 119.65845   , ..., 153.29634   ,
                 166.48473   , 167.00223   ],
                [120.10571   , 113.27077   , 128.64536   , ..., 188.1266    ,
                 174.23666   , 175.4878    ],
                [123.17761   , 128.3278    , 136.94447   , ..., 167.48372   ,
```

The shape of the returned array is number of pulses x Y x X:

In [8]:
```
img_ary.shape
```

Out[8]:
```
(176, 512, 128)
```

To achieve the above shown plot three essential steps are necessary:

- stacking the detector data into one big Nd array
- loading the geometry description
- applying the loaded geometry to the data

Let's stack the data detector modules in the module_data dictionary first. The `stack_detector_data` method will create one array from all selected detector modules:

In [13]:
```
train_img = kd.stack_detector_data(train_data, 'image.data', only='SPB_D
ET_AGIPD1M-1/DET')
train_img.shape
```

Out[13]:
```
(176, 16, 512, 128)
```

Now load the geometry information. This information is stored in a so called *geometry file* that describes the layout of the detector. Here we use the *.geom* format which is used by standard crystallography tools like CrystFEL. Karabo-data is able to handle this format:

In [14]:
```
from karabo_data.geometry2 import AGIPD_1MGeometry
geom = AGIPD_1MGeometry.from_crystfel_geom('xfel.geom')
```

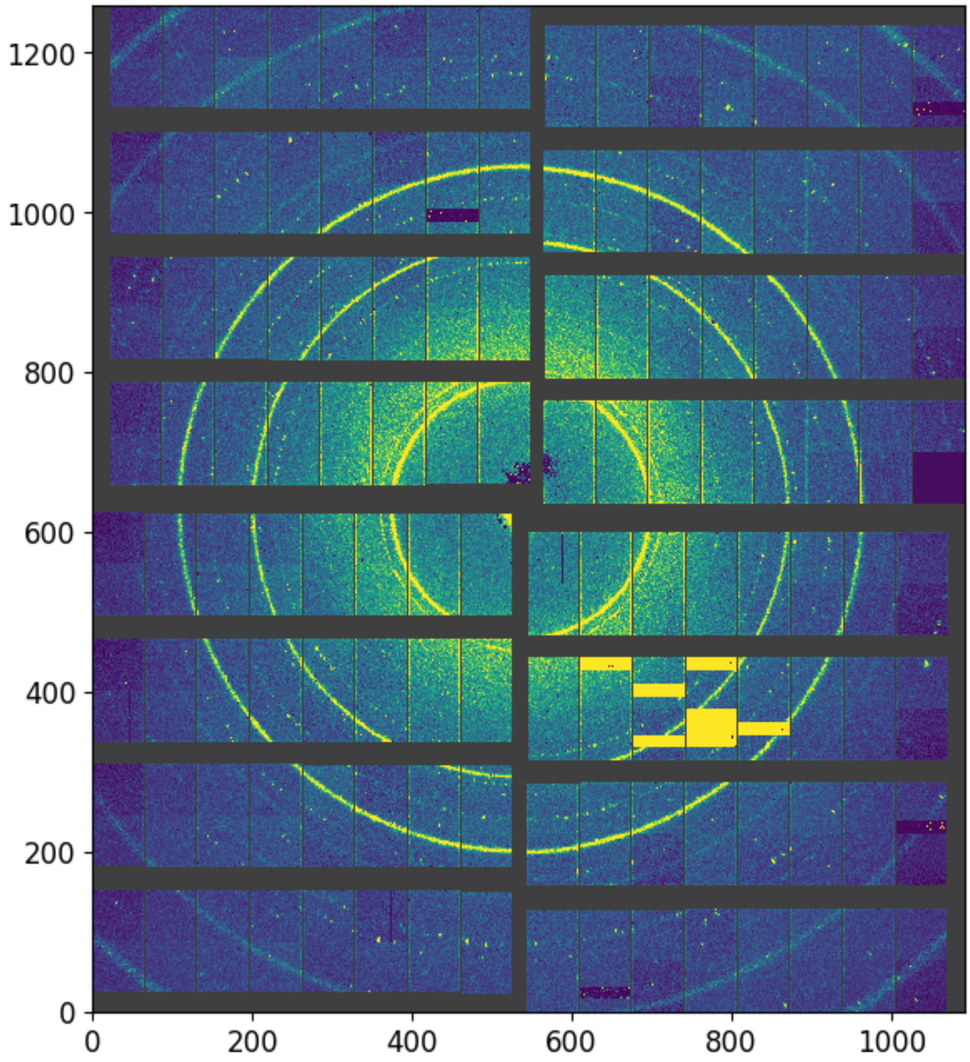
And let's apply the geometry and plot the data (11th pulse)

In [15]:
```
img, center = geom.position_all_modules(train_img)
img.shape
```

Out[15]:
```
(176, 1259, 1092)
```

In [16]:
```
from matplotlib import pyplot as plt
plt.imshow(np.clip(img[10], -50, 1500))
```

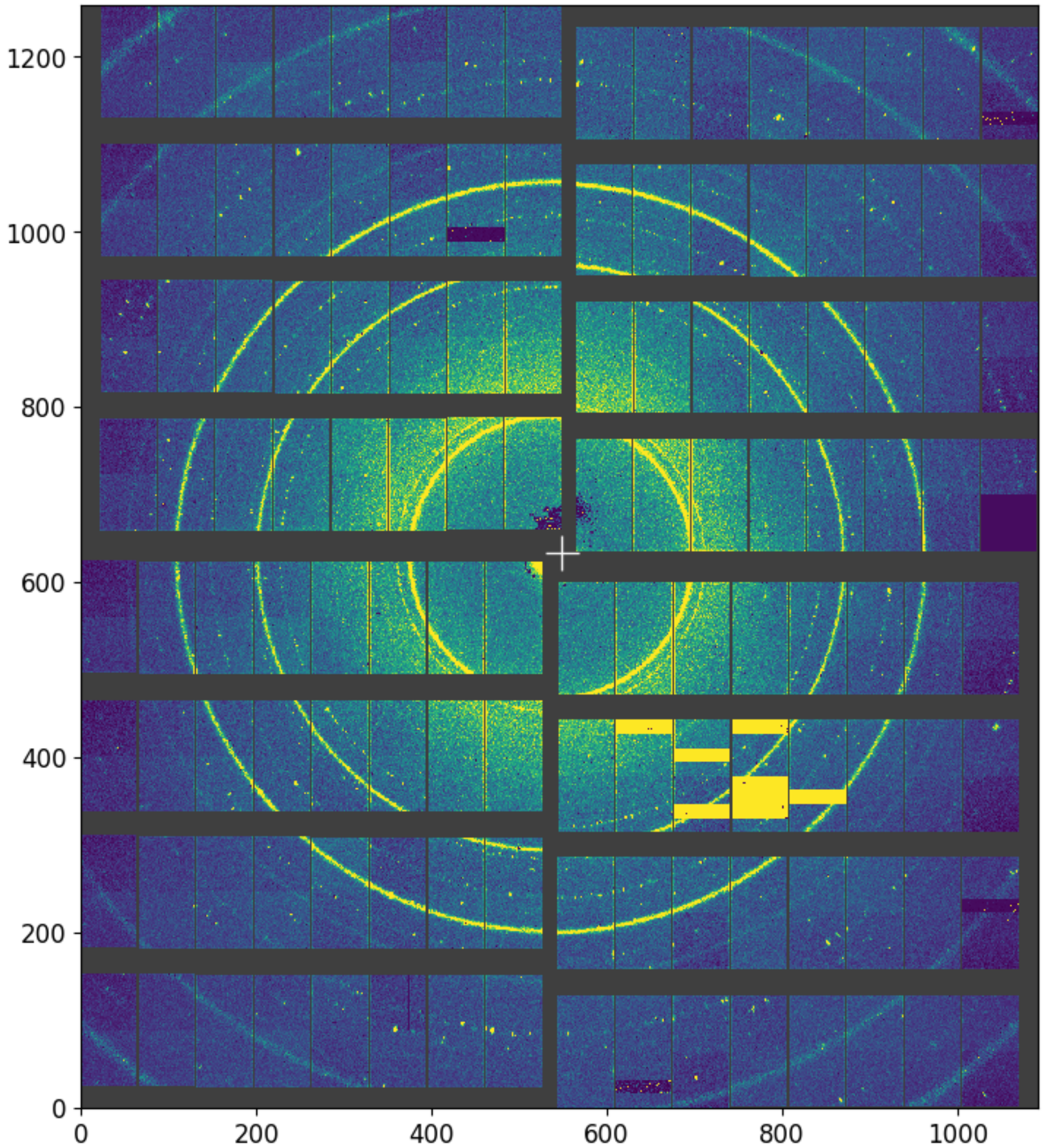
Out[16]: <matplotlib.image.AxesImage at 0x2ad29c614a20>



Or even more easy you could *just* apply the `plot_data_fast` method:

In [17]:
```
geom.plot_data_fast(np.clip(train_img[10], -50, 1500))
```

Out[17]:



Getting run information:

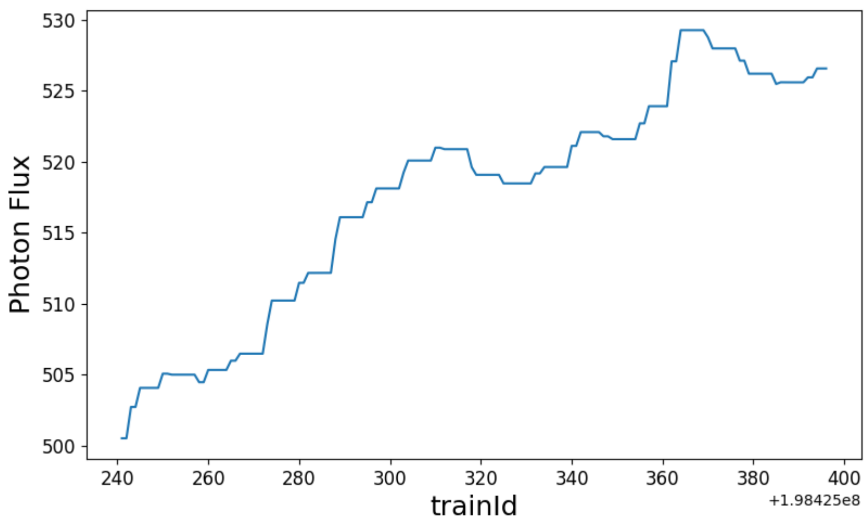
`RunDirectory` has a `info` method that displays a useful run experiment overview:

In [18]:
```
run_folder = '/gpfs/exfel/exp/XMPL/201750/p700000/raw/r0273'
run_dir = kd.RunDirectory(run_folder)
run_dir.info()
```

```
# of trains:    156
Duration:       0:00:15.500000
First train ID: 198425241
Last train ID:  198425396

16 detector modules (SPB_DET_AGIPD1M-1)
  e.g. module SPB_DET_AGIPD1M-1 0 : 512 x 128 pixels
  176 frames per train, 27456 total frames

2 instrument sources (excluding detectors):
  - SA1_XTD2_XGM/XGM/DOOCS:output
  - SPB_XTD9_XGM/XGM/DOOCS:output

13 control sources:
  - ACC_SYS_DOOCS/CTRL/BEAMCONDITIONS
  - SA1_XTD2_XGM/XGM/DOOCS
  - SPB_IRU_AGIPD1M/PSC/HV
  - SPB_IRU_AGIPD1M/TSENS/H1_T_EXTHOUS
  - SPB_IRU_AGIPD1M/TSENS/H2_T_EXTHOUS
  - SPB_IRU_AGIPD1M/TSENS/Q1_T_BLOCK
  - SPB_IRU_AGIPD1M/TSENS/Q2_T_BLOCK
  - SPB_IRU_AGIPD1M/TSENS/Q3_T_BLOCK
  - SPB_IRU_AGIPD1M/TSENS/Q4_T_BLOCK
  - SPB_IRU_AGIPD1M1/CTRL/MC1
  - SPB_IRU_AGIPD1M1/CTRL/MC2
  - SPB_IRU_VAC/GAUGE/GAUGE_FR_6
  - SPB_XTD9_XGM/XGM/DOOCS
```

Selecting data based by trains is simple with karabo-data but what if data should be selected across trains?

Scenario II: Extracting Data across trains with one value per train



Photon flux *time-series* (by trainID)

- How can 1D data be extracted and plotted?

The *get_series* method can extract a series across trainID's for a given device and property:

The photon flux values are stored in the `pulseEnergy.photonFlux.value` property of device `SA1_XTD2_XGM/XGM/DOOCS` and has exactly one entry per train:

In [19]:
```
ph_flux = run_dir.get_series('SA1_XTD2_XGM/XGM/DOOCS', 'pulseEnergy.phot
onFlux.value')
type(ph_flux)
```

Out[19]:
```
pandas.core.series.Series
```

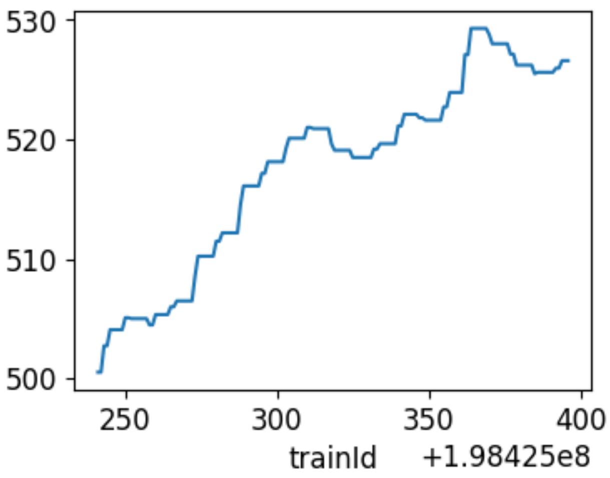
In [20]:
```
ph_flux.head(5)
```

Out[20]:
```
trainId
198425241    500.519470
198425242    500.519470
198425243    502.727203
198425244    502.727203
198425245    504.070953
Name: SA1_XTD2_XGM/XGM/DOOCS/pulseEnergy.photonFlux, dtype: float32
```

Pandas is a very useful data analysis library. More information is available under [https://pandas.pydata.org (https://pandas.pydata.org)](https://pandas.pydata.org).

In [21]:
```
ph_flux.plot(figsize=(4,3))
```

Out[21]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x2ad29c6cf4e0>
```



What if you wanted to get more than *one* device and/or property?

The `get_dataframe` method combines different sources into one single data object (also **pandas**):

In [22]:
```
fluxes_pos = run_dir.get_dataframe(fields=[("*/XGM/DOOCS", "*.i[xy]Pos")
])
type(fluxes_pos)
```

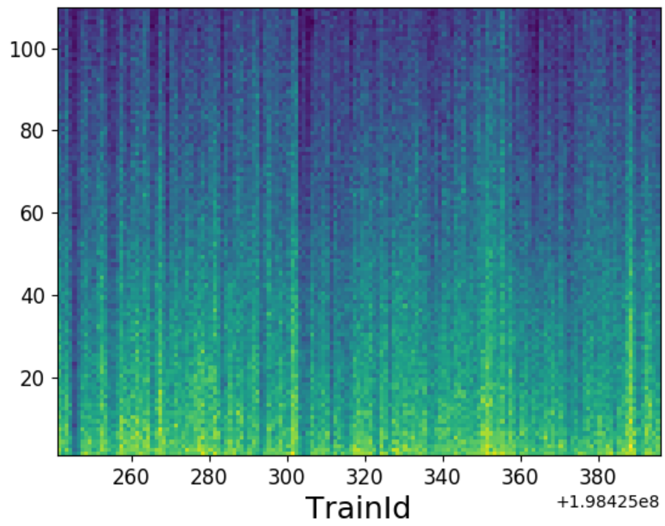
Out[22]:
```
pandas.core.frame.DataFrame
```

In [23]:
```
fluxes_pos.head(5)
```

Out[23]:

trainId	SPB_XTD9_XGM/XGM/DOOCS/beamPosition.iyPos	SPB_XTD9_XGM/XGM/DOOCS/beamPosition.ixPos	SA1_XTD2_XGM/XGM/DOOCS/beamPosition.iyPos	SA1_XTD2_XGM/XGM/DOOCS/beamPosition.ixPos
198425241	-3.121433	5.512009	0.315761	1.293711
198425242	-3.121433	5.512009	0.315761	1.293711
198425243	-3.121433	5.512009	0.315761	1.293711
198425244	-3.090523	5.528512	0.341187	1.336566
198425245	-3.090523	5.528512	0.341187	1.336566

Scenario III: Getting data with multiple values per train



X-ray gas intensity data is pulse resolved

- How can this 2D data be extracted and plotted?

the `get_array` method returns a data array that contains more than one value per train.

X-ray gas intensity data is pulse resolved and serves as an example:

the data can be accessed, similarly to `get_dataframe` by giving *device* name and *property*:

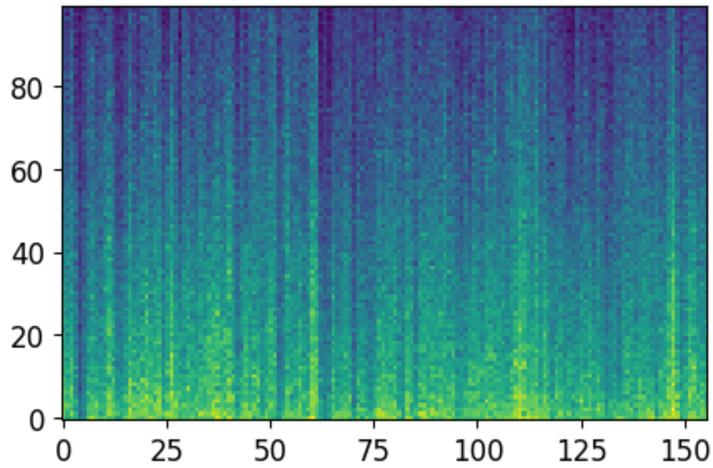
In [24]:
```
xgm_intensity = run_dir.get_array('SA1_XTD2_XGM/XGM/DOOCS:output', 'data
.intensityTD')
xgm_intensity
```

Out[24]:
```
<xarray.DataArray (trainId: 156, dim_0: 1000)>
array([[ 957.0532 , 1026.0005 ,  949.8755 , ...,    0.     ,    0.     ,
           0.     ],
       [ 763.8806 ,  794.2738 ,  868.2455 , ...,    0.     ,    0.     ,
           0.     ],
       [ 859.37   ,  995.1641 ,  838.5669 , ...,    0.     ,    0.     ,
           0.     ],
       ...,
       [ 945.2731 ,  812.4336 ,  839.45654, ...,    0.     ,    0.     ,
           0.     ],
       [ 903.26855,  940.15125,  953.9436 , ...,    0.     ,    0.     ,
           0.     ],
       [ 944.08386,  949.549  ,  861.7509 , ...,    0.     ,    0.     ,
           0.     ]], dtype=float32)
Coordinates:
  * trainId  (trainId) uint64 198425241 198425242 ... 198425395 198425396
Dimensions without coordinates: dim_0
```

In [25]:
```
#Reset the default image size
matplotlib.rc('image', cmap='viridis', origin='lower')
matplotlib.rc('figure', figsize=(5,5))
```

In [26]:
```
plt.imshow(xgm_intensity[:,:100].T)
```

Out[26]:
```
<matplotlib.image.AxesImage at 0x2ad29c988eb8>
```



a labeled array (**xarray**) is returned. More information on labeled arrays can be found on http://xarray.pydata.org (http://xarray.pydata.org)

Scenario IV: In the unlikely event if something goes wrong

A common source of errors is an invalid structure of the created data files. Hence a useful starting point to debug any errors is to check for valid file structures using the `karabo-data-validate` command

```
$: karabo-data-validate /gpfs/exfel/exp/XMPL/201750/p700000/raw/r0273
Checking run directory: /gpfs/exfel/exp/XMPL/201750/p700000/raw/r0273
No problems found
```

The command checks if:

- All .h5 files in a run can be opened, and the run contains at least one usable file.
- The list of train IDs in a file has no zeros except for padding at the end.
- Each train ID in a file is greater than the one before it.
- The indexes do not point to data beyond the end of a dataset.
- The indexes point to the start of the dataset, and then to successive chunks for successive trains, without gaps or overlaps between them.

How to install/get karabo-data?

Karabo-data is available on GitHub and there are multiple ways to install it:

- it is automatically available when you enter maxwell via jupyther-hub → https://max-jhub.desy.de (https://max-jhub.desy.de)
- it is installed in maxwell's anaconda environment → `module load anaconda/3`
- it can be install using *pip* → `pip install (--user) karabo_data`
- the latest version could be downloaded from GitHub → `git clone https://github.com/European-XFEL/karabo_data.git`

A much more detailed documentation is available on *readthedocs* :

https://karabo-data.readthedocs.io/en/latest/ (https://karabo-data.readthedocs.io/en/latest/)

In [ ]: