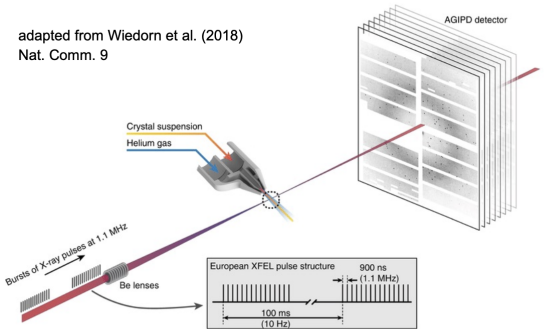
Serial Femtosecond Crystallography at European XFEL

Introduction to the analysis of Serial Femtosecond X-ray Crystallography (SFX) data at European XFEL with the EXtra-Xwiz tool.

This tutorial is based on and can be used in conjunction with EXtra-Xwiz paper published in Crystals vol.13 2023 and available at: https://www.mdpi.com/2073-4352/13/11/1533

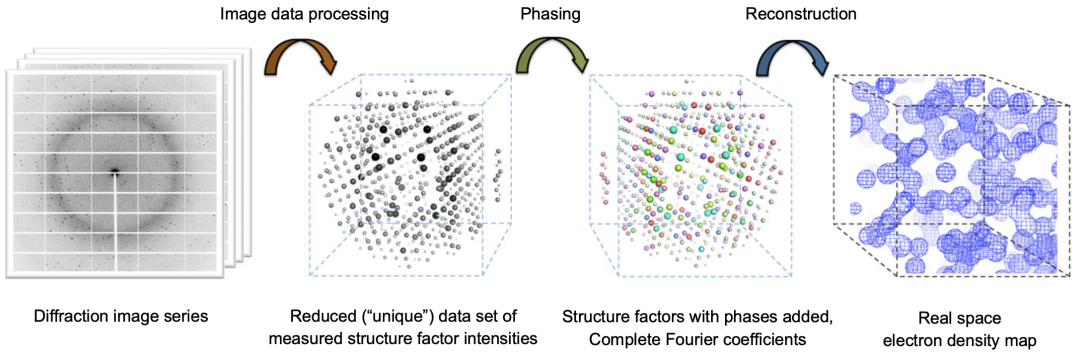
A typical Serial Femtosecond Crystallography experiment at EuXFEL (SPB/SFX, FXE)



- Diffraction before destruction
- Each image stems from a differently oriented crystal, indexing cannot make use of multi-frame information
- Requirement of high crystal isomorphism, each image must represent (closely) the same crystal “template”
- Still images, no oscillation range, integration must use a 3D model for 2D pixel data
- Many empty frames due to missing shots

In a typical SFX experiment at the SPB/SFX (and also the FXE) instrument, microcrystalline sample is delivered by a liquid jet system - either acqueous solution delivered into a vacuum chamber with jet speeds up to 120 m/s, or (at SPB) more viscous solutions delivered to an interaction region at atmospheric pressure. The jet stream is aligned to the X-ray beam such that sample is hit by FEL X-ray pulses perpendicularly at a rate of typically 1.1 MHz (900 ns intervals of femtosecond pulses, up to 352 pulses in a train, 10 trains per second), and with the detector at the same repetition rate for recordings, i.e. taking a total of 3520 image frames per second. The crystal hit rate largely depends on the jet speed, and lies in the range of 1% to 10% for most experiments. It is much higher for the slower viscous jets.

The overall process of crystal structure determination



The first part of data analysis by SFX beamtime users ends with a unique set of crystallographic structure factors.

- The software used to achieve this is in most cases CrystFEL.
- The DA group provides support in terms of the pipeline tool ("workflow manager") EXtra-Xwiz which employs CrystFEL

CrystFEL link and reference:

- https://www.desy.de/~twhite/crystfel/

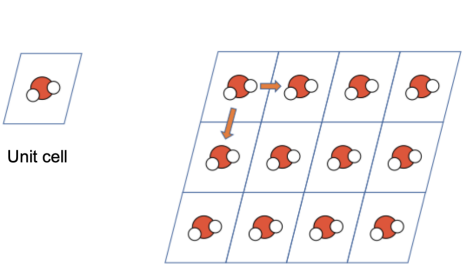
The goal of the experiment is to reconstruct the electron density within the crystal unit cell (resp. asymmetric unit) so that a structural model of the protein can be built into it.

Today we will demonstrate a use case based on data collected from the protein Hen Egg-White Lysozyme (HEWL):

Unit cell and indexing

The features of solid matter in crystalline state determine the diffraction pattern geometry.

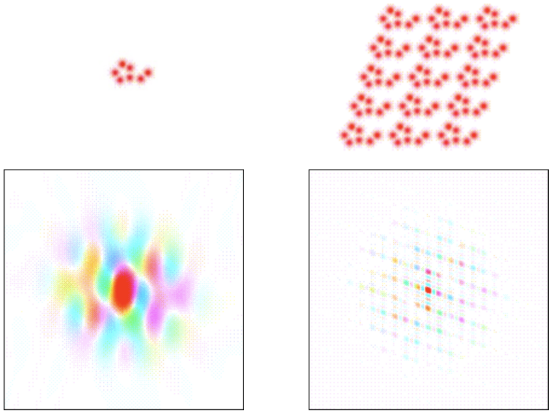
- Most important feature of a crystal is its internal arrangement: the crystal lattice, build of unit cells.



A crystal is a 3D translation lattice

Effect of the lattice:

- Amplification of scattering signal
- Discretization of the diffraction pattern



Kevin Cowtan, "Book of Fourier Transforms", © University of York
http://www.ysbl.york.ac.uk/~cowtan/fourier/gallery.html

As described by the Bragg model, and quantified by the Bragg equation, diffraction peaks are the result of constructive interference of scattered photons given a 3D grating

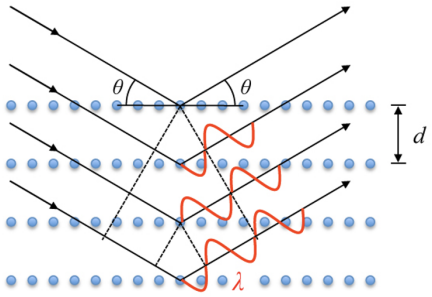
- Unit cell constants correspond to translation vector lengths and angles in the lattice
- Bragg peaks, also called "reflections", can be assigned Miller indices
- Miller indices are deduced from the inter-peak distances, being Fourier space equivalents of translation vector combinations at integer-multiple samples of the lattice

Description of a crystal unit cell as read by CrystFEL:

In [1]:
```
with open('cell/hewl.cell', 'r') as f:
    for line in f:
        print(line[:-1])
```

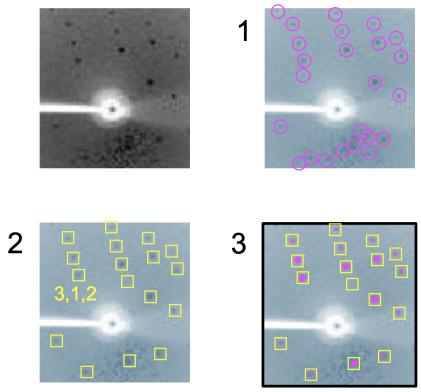
```
CrystFEL unit cell file version 1.0

lattice_type = tetragonal
centering = P
unique_axis = c
a = 79.1 A
b = 79.1 A
c = 37.9 A
al = 90.00 deg
be = 90.00 deg
ga = 90.00 deg
```

Bragg's law: $2d \sin(\theta) = n\lambda$



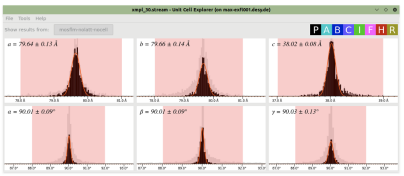
The steps to reduced experimental data

All of the following steps are taken care of by one program of the CrystFEL package: indexamajig



1 Peak finding
- ▶ threshold important to identify diffraction events

2 Indexing
- ▶ in principle, only frames with events need to be indexed
- ▶ in practice, indexing success helps picking good frames
- ▶ keeping the consistency of unit cell constants in mind



3 Integration of pixel intensities
- ▶ based on still images

After pixel intensity integration, multiple observations (Bragg peaks) belonging to the same or symmetry-equivalent structure factor have to be averaged after some scaling (correction of systematic effects)

- this is taken care of by the CrystFEL program "partialator"
- the resulting set of structure factors consists of Miller indices H, K, L, the intensity and its uncertainty

Format of a HKL -type structure factor file as produced by CrystFEL:

In [2]:
```
with open('xwiz/example_results/lyso_sample.hkl', 'r') as f:
    for line in f:
        print(line[:-1])
```

```
   h    k    l          I    phase   sigma(I)   nmeas
   0    0    2     263.84        -     150.05       7
   0    0    3    -136.06        -     554.21       6
   0    0    4    1833.76        -     390.34       6
   1    5    8     378.27        -     193.14      21
   1    5    9    3801.79        -     729.02      20
   1    5   10    1561.88        -     493.09      17
   2   29    2    1173.55        -     528.36      21
   2   29    3    -126.11        -     495.38       6
   2   29    4     616.39        -     381.23      11
   2   29    5    2460.75        -     596.84      10
```

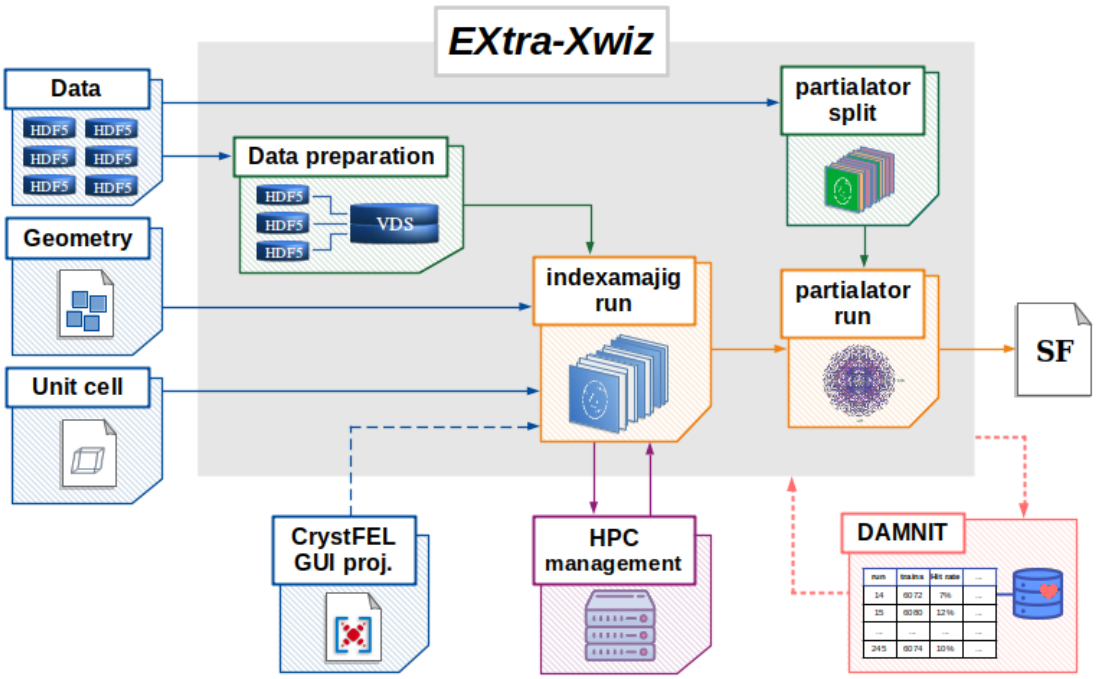
The SFX workflow run with EXtra-Xwiz

CrystFEL comes with a GUI, but for batch processing with known parameters, its tools like `indexamajig` can be run from the command line in a terminal.

Aspects like:

- preparation of data from `proc` runs into virtual datasets
- chaining of different programs into a processing pipeline
- management of SLURM arrays in batch jobs
- data frames splitting into separate datasets in case of pump-probe experiment

are facilitated by the the `EXtra-xwiz` tool:



EXtra-xwiz allows to run analysis by setting peak finding, indexing and reflections scaling and merging parameters in a single configuration file. It handles distribution of computations over nodes on the SLURM cluster for faster computations and allows to handle automatically some of the processing steps specific for EuXFEL, such as generating virtual dataset files or splitting data frames into saparate datasets in the experiments with sample illumination by a pump laser.

Let's take a look on the example of xwiz configuration file, for this run in the terminal:

```
cd ~/xwiz/example_files/
vim xwiz_conf.toml
```

(to exit `vim` simply press `Esc` and type `:q`)

In [3]:
```
xwiz_conf_file_t01 = "xwiz/example_files/xwiz_conf.toml"
with open(xwiz_conf_file_t01, 'r') as fin:
    xwiz_config = fin.readlines()
for line in xwiz_config:
    print(line, end='')
```

```
[data]
proposal = 700000
runs = [30]
#frames_range = {start = 0, end = 100, step = 1}

# ! This file is used only for tutorial demonstration on VISA:
frames_list_file = "../example_files/indexed_p700000_r0030_cut.lst"

[crystfel]
# Available versions on Maxwell: '0.8.0', '0.9.1', '0.10.2', 'maxwell_dev
'
# Available version on VISA: '0.10.2_visa'
version = '0.10.2'

[geom]
file_path = "../../geom/agipd_p700000_r0030.geom"

[slurm]
# Available partitions: 'local', 'all', 'upex', 'exfel'
partition = "local"
# In case you have slurm nodes reservation
reservation = "none"
duration_all = "1:00:00"
n_nodes_all = 20

[indexamajig_run]
resolution = 1.6
peak_method = "peakfinder8"
peak_threshold = 800
peak_snr = 5
peak_min_px = 1
peak_max_px = 2
peaks_hdf5_path = "entry_1/result_1"
index_method = "mosflm"
n_cores = 1
local_bg_radius = 3
integration_radii = "2,3,5"
max_res = 1600
min_peaks = 10
extra_options = "--no-non-hits-in-stream"

[partialator_split]
execute = false
# Available modes: "on_off", "on_off_numbered", "by_pulse_id", "by_train_
id"
mode = "by_pulse_id"

# Required only for "on_off" or "on_off_numbered" modes:
xray_signal = ["SPB_LAS_SYS/ADC/UTC1-1:channel_0.output", "data.rawData"]
laser_signal = ["SPB_LAS_SYS/ADC/UTC1-1:channel_1.output", "data.rawDat
a"]
plot_signal = true

# Required only for "by_pulse_id" or "by_train_id" mode:
[partialator_split.manual_datasets]
  my_on = {start=0, end=-1, step=12}
  my_off = [{start=4, step=12}, {start=8, step=12}]

[unit_cell]
file_path = "../../cell/hewl.cell"
run_refine = false

[merging]
```

```
point_group = "422"
scaling_model = "unity"
scaling_iterations = 1
max_adu = 100000
```

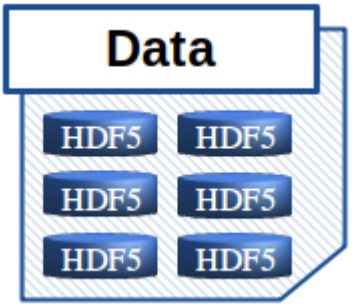
This configuration file is split into sections:

- `[data]` - contains information related to the input detector data, such as proposal and run number.
- `[crystfel]` - allows to specify different versions of CrystFEL suite. Out visa image contain only the newest to date version `0.10.2` which we specify as `0.10.2_visa` .
- `[geom]` - path to the same AGIPD geometry file we have used before.
- `[slurm]` - configuration of SLURM cluster computations distribution. We don't have access to the cluster and therefore will use `partition = 'local'` .
- `[indexamajig_run]` - parameters for peak search and indexing diffraction patterns.
- `[partialator_split]` - configures splitting of the reflections data into custom subsets before merging. It is commonly used in the analysis of data from time-resolved experiments.
- `[unit_cell]` - path to the file with unit cell parameters.
- `[merging]` - scaling and marging parameters for CrystFEL's `partialator` tool.

Before we continue - load some libraries:

In [4]:
```
from subprocess import check_output
import os.path as osp
from pathlib import Path

import numpy as np
import h5py
import matplotlib.pyplot as plt

from extra_geom import AGIPD_1MGeometry
```

Input data



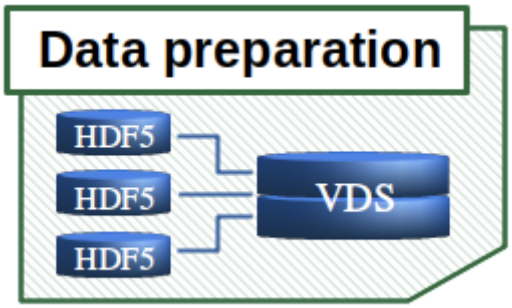
The data to be processed by the pipeline can be specified with just a proposal number and a list of runs in the `[data]` section of the configuration file:

In [5]:
```
for line in xwiz_config[:7]:
    print(line, end='')
```

```
[data]
proposal = 700000
runs = [30]
#frames_range = {start = 0, end = 100, step = 1}

# ! This file is used only for tutorial demonstration on VISA:
frames_list_file = "../example_files/indexed_p700000_r0030_cut.lst"
```

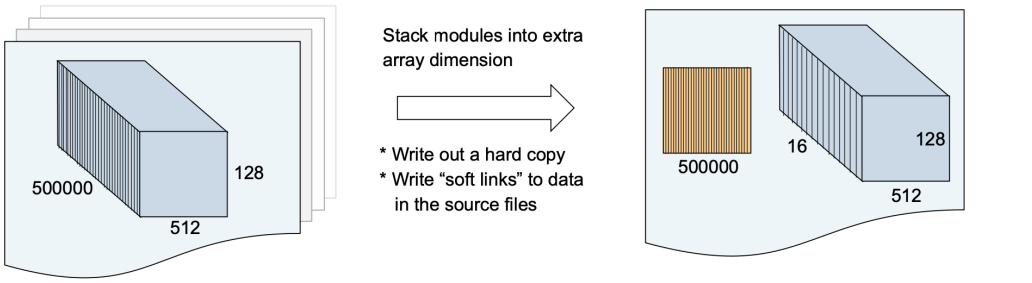
Here we specify a `frames_list_file` with the preselection of frames which are easy to index. This file is prepared only for the purpose of pipeline demonstration and is never used in the actual analysis.

Stacking detector data: virtual data set



CrystFEL wants detector data - at least all pixel intensities of one image frame - in one file, and one dataset object within the file.

- both stacking and "slabbing" of module arrays into one image array is possible
- HDF5 allows for virtualization, i.e. linking actual data



(N ×) 16 HDF5 files, one (set) per module, in a run folder: several TB in total

Single HDF5 file
- real data set / copy (not feasible)
- virtual data set (~ 50 MB)

An HDF5 file in the CXI format with virtual datasets will be generated by the pipeline. It can also be prepared with a script in our VISA image:

```
cd data
./make_vds_proc.sh
```

`make_vds_proc.sh` script executes `extra-data-make-virtual-cxi` , a command line interface of EXtra-data library for generating VDS files.

Generated VDS file can be inspected with:

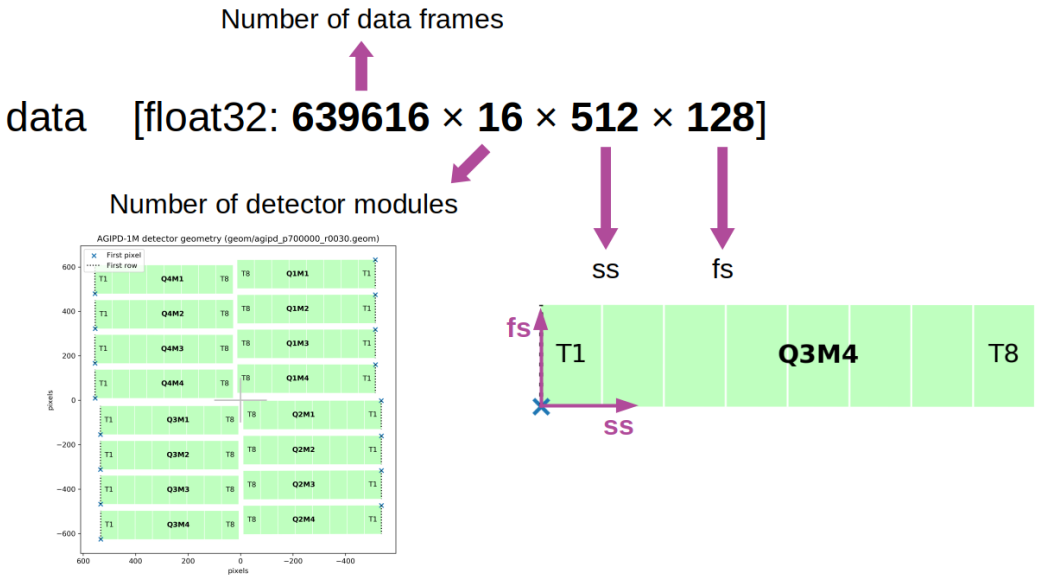
```
h5glance p700000_r0030_proc.cxi
```

or:

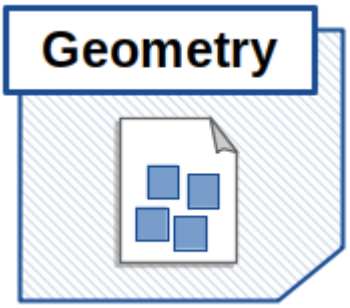
In [6]:
```
p700000_r0030_vds_file = "data/p700000_r0030_proc.cxi"
if not osp.exists(p700000_r0030_vds_file):
    vds_gen_result = check_output(f"./make_vds_proc.sh", cwd="data")

p700000_r0030_vds_structure = check_output(f"h5glance {p700000_r0030_vds_
print(p700000_r0030_vds_structure)
```

```
data/p700000_r0030_proc.cxi
├cxi_version    [int64: 1]
└entry_1
  ├cellId       [uint16: 639616 × 16] virtual (1 attributes)
  ├data_1       -> /entry_1/instrument_1/detector_1
  ├experiment_identifier        [UTF-8 string: 639616]
  ├instrument_1
  │ └detector_1
  │   ├data     [float32: 639616 × 16 × 512 × 128] virtual (1 attributes)
  │   ├experiment_identifier    -> /entry_1/experiment_identifier
  │   ├gain     [uint8: 639616 × 16 × 512 × 128] virtual (1 attributes)
  │   ├mask     [uint32: 639616 × 16 × 512 × 128] virtual (1 attributes)
  │   └module_identifier        [int64: 16]
  ├pulseId      [uint64: 639616]
  └trainId      [uint64: 639616]
```

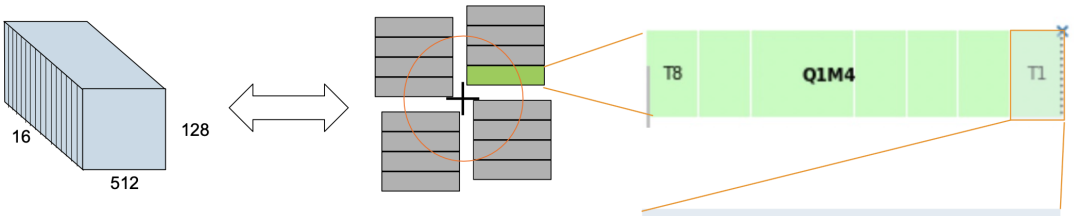
Here `data`, `mask` and `gain` arrays have dimensions:



Detector geometry description



We need to know the detector topology and exact position of pixels (given e.g. quadrant motor positions) for a physically accurate assembly of an image. Spatial position of each detector module in laboratory frame along with other relevant information such as sample-to-detector distance, X-ray beam energy or bad pixel mask is usually stored in a dedicated detector geometry file.



- CrystFEL geometry file
- Separate description of each detector tile
  - Fast-scan and slow-scan pixel axis
  - absolute corner pixel position with respect to the detector center (origin)
  - Translation vector in the 2D plane to be applied to index increments

```
p4a1/min_fs = 0
p4a1/min_ss = 64
p4a1/max_fs = 127
p4a1/max_ss = 127
p4a1/fs = -0.001899x -0.999999y
p4a1/ss = +0.999999x -0.001899y
p4a1/corner_x = -482.901
p4a1/corner_y = -4.48727
```

Detailed description of the CrystFEL geometry file can be found in the dedicated manual:

https://www.desy.de/~twhite/crystfel/manual-crystfel_geometry.html

It is worth mentioning that precise tuning of the detector geometry is a topic on its own and is outside of the scope of this tutorial.

Very useful tools to fine-tune position and orientation of detector modules in space are *geoptimiser*:

https://www.desy.de/~twhite/crystfel/manual-geoptimiser.html

and *align_detector*:

https://gitlab.desy.de/thomas.white/crystfel/-/blob/master/doc/man/align_detector.1.md

Path to the actual detector geometry file in CrystFEL format should be specified in the `[geom]` section of the EXtra-Xwiz configuration file:

In [7]:
```
for line in xwiz_config[13:15]:
    print(line, end='')
```

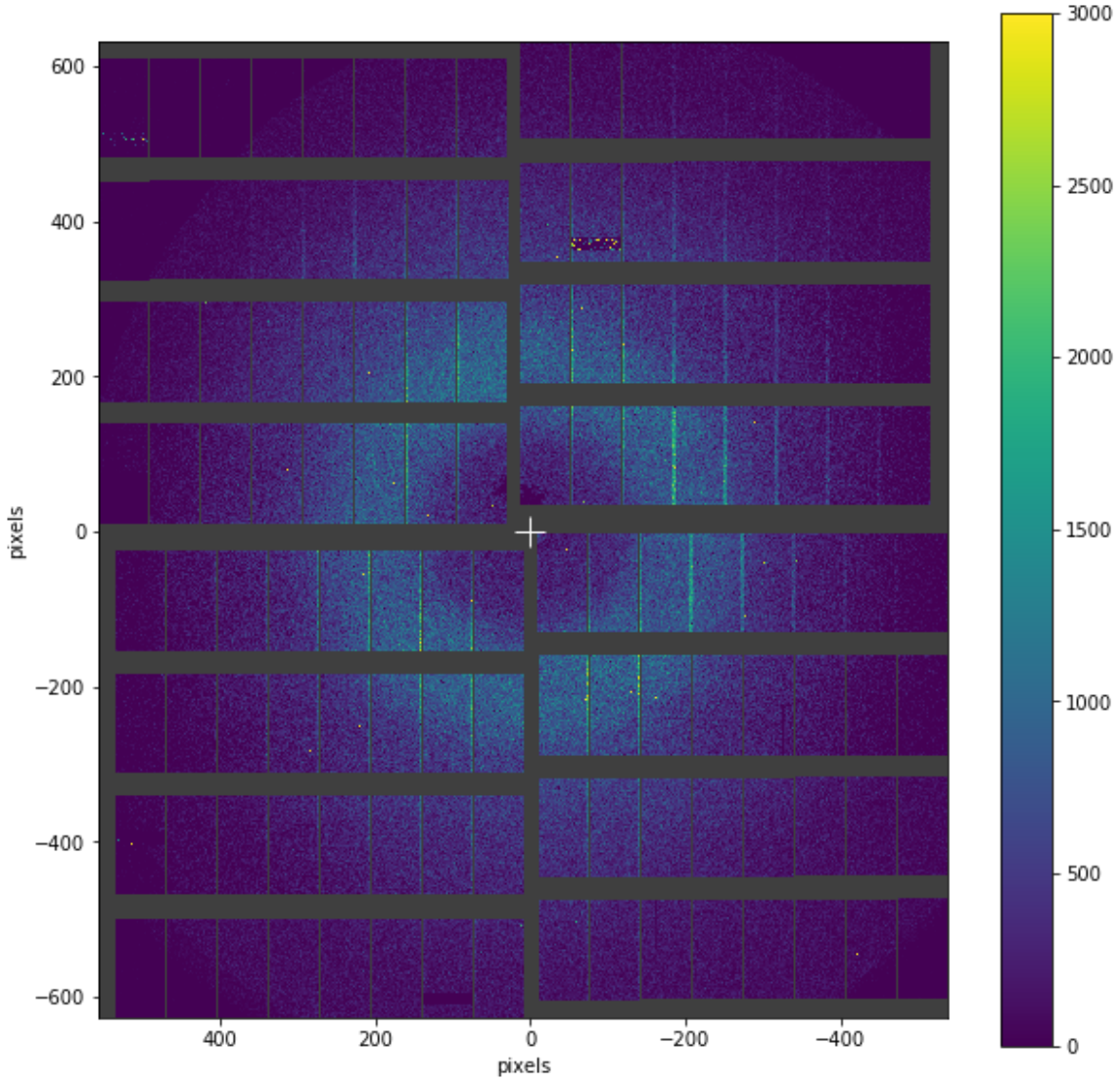
```
[geom]
file_path = "../../geom/agipd_p700000_r0030.geom"
```

Geometry file used in the tutorial can be found at `geom/agipd_p700000_r0030.geom` .

The same geometry file can be used with the EXtra-geom library to inspect the data in physical layout:

In [8]:
```
agipd_geom_file = "geom/agipd_p700000_r0030.geom"
agipd_geom = AGIPD_1MGeometry.from_crystfel_geom(agipd_geom_file)

with h5py.File(p700000_r0030_vds_file, 'r') as vds_file:
    agipd_data_arr = vds_file['/entry_1/data_1/data'][95]

agipd_geom.plot_data(agipd_data_arr, vmin=0, vmax=3000)
```

Out[8]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x2b6644664b00>
```

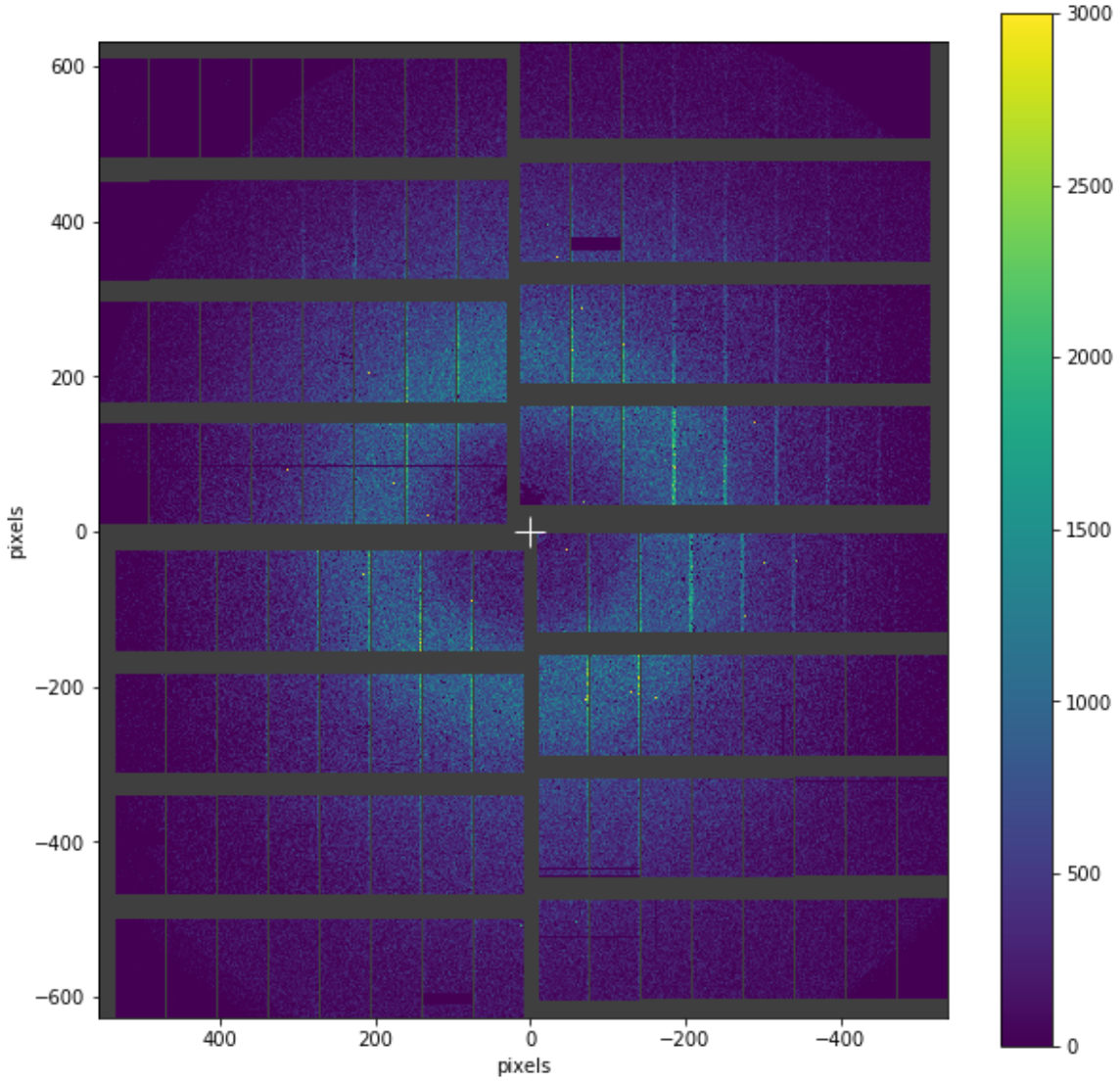


Mask bad detector pixels

By exploring the detector image above one might notice that it contains not only water scattering and lysozyme break peaks, but also some bright misbehaving pixels. Most of these bad pixels can be covered with a mask generated by the offline calibration pipeline, which is stored under `/INSTRUMENT/SPB_DET_AGIPD1M-1/DET/4CH0:xtdf /image/mask` in the `proc` data or `/entry_1/instrument_1 /detector_1/mask` in the VDS file.

In [9]:
```
with h5py.File(p700000_r0030_vds_file, 'r') as vds_file:
    agipd_mask_arr = vds_file['/entry_1/data_1/mask'][95]

agipd_plot_arr = agipd_data_arr * (agipd_mask_arr == 0).astype(float)

agipd_geom.plot_data(agipd_plot_arr, vmin=0, vmax=3000)
```

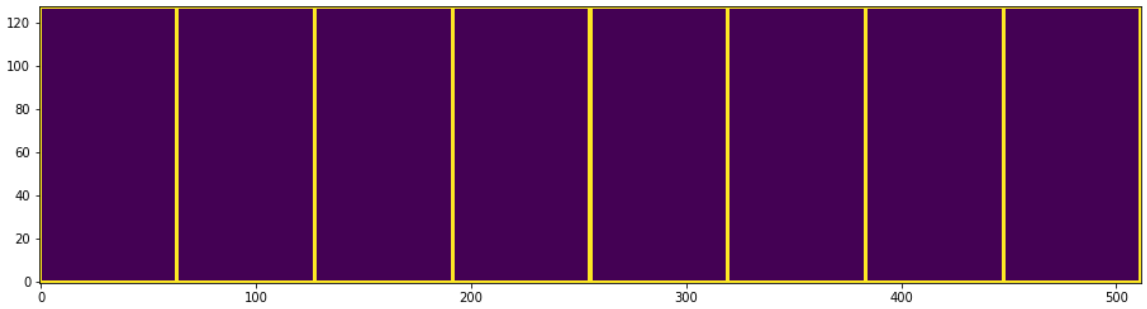
Out[9]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x2b66447921d0>
```



Sometimes a mask from the offline calibration is not enough and an additional, external, mask may be required. For example brighter pixel columns originate from the asics edges which have twice larger pixel size and as a result measure artificialy larger intensity. A static mask to exclude these pixels from the analysis can be prepared manually with:

In [10]:
```
agipd_manual_mask = np.zeros(agipd_data_arr.shape, dtype=np.uint32)

# loop over 8 asics in the panel:
for i_asic in range(8):
    asic_ss = i_asic * 64
    agipd_manual_mask[:, asic_ss, :] = 1
    agipd_manual_mask[:, asic_ss + 63, :] = 1
agipd_manual_mask[:, :, 0] = 1
agipd_manual_mask[:, :, 127] = 1

plt.figure(figsize=(16, 4))
plt.imshow(np.swapaxes(agipd_manual_mask[0], 0, 1), origin='lower', inter
```

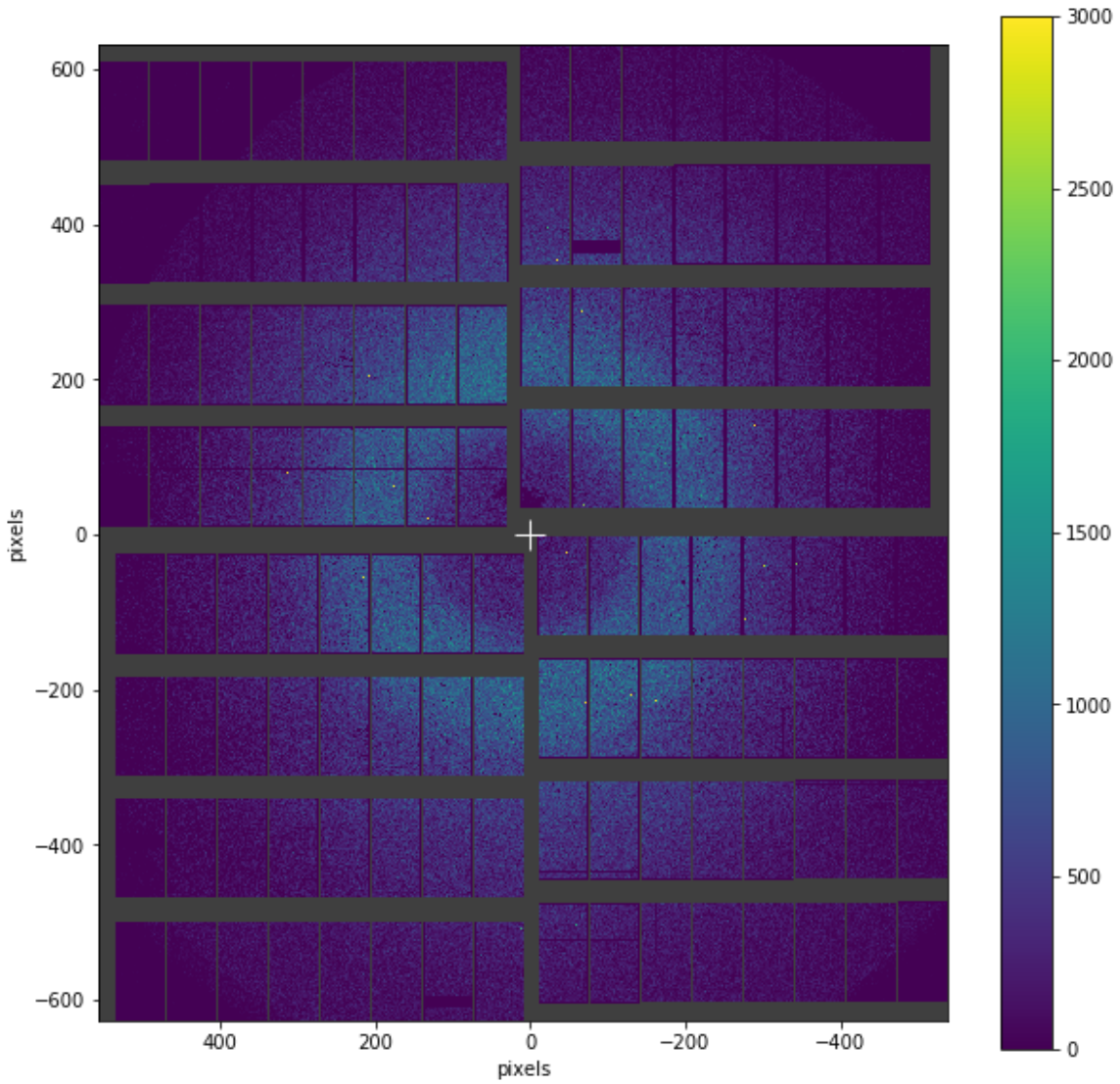
Out[10]:
```
<matplotlib.image.AxesImage at 0x2b66d3f602e8>
```



We have displayed the mask just for one module for better visibility. Now plot the data applying additional mask:

In [11]:
```
agipd_plot_arr = agipd_data_arr * (agipd_mask_arr == 0).astype(float) * (
agipd_geom.plot_data(agipd_plot_arr, vmin=0, vmax=3000)
```

Out[11]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x2b66d3f82e10>
```



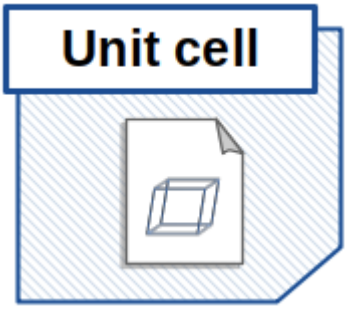
Let's store this static mask into an hd5 file:

In [12]:
```
Path("mask").mkdir(exist_ok=True)

static_mask_file = "mask/mask_manual_v01.h5"
with h5py.File(static_mask_file, 'w') as fout:
    fout.create_dataset('/entry_1/data_1/mask', data=agipd_manual_mask)
```

It can be specified in line 9 of our geometry file 'geom/agipd_p700000_r0030.geom' as:

```
mask1_file = <path_to_your_folder>/mask/mask_manual_v01.h5
mask1_data = /entry_1/data_1/mask
mask1_goodbits = 0
mask1_badbits = 1
```

Unit cell file



As already mentioned above - cell file contains description of the unit cell:

In [13]:
```
with open('cell/hewl.cell', 'r') as f:
    for line in f:
        print(line[:-1])
```

```
CrystFEL unit cell file version 1.0

lattice_type = tetragonal
centering = P
unique_axis = c
a = 79.1 A
b = 79.1 A
c = 37.9 A
al = 90.00 deg
be = 90.00 deg
ga = 90.00 deg
```

It should be provided to the pipeline under `[unit_cell]` section:

In [14]:
```
for line in xwiz_config[55:58]:
    print(line, end='')
```

```
[unit_cell]
file_path = "../../cell/hewl.cell"
run_refine = false
```

CrystFEL graphical user interface

As already mentioned, CrystFEL is a suite of programs for processing Serial Femtosecond Crystallography diffraction data. It comprises programs for indexing and integrating diffraction patterns, scaling and merging intensities, calculating figures of merit, and many more.

To start a CrystFEL GUI application execute in the terminal:

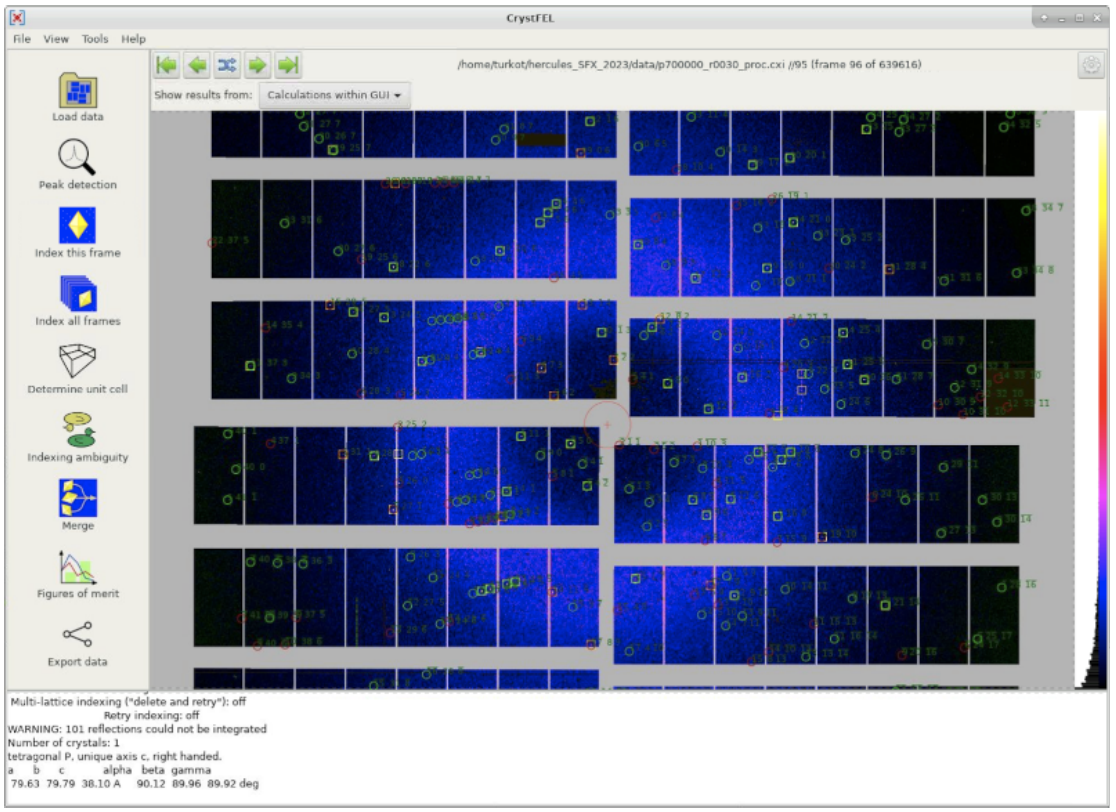
```
crystfel
```

Within GUI:

- load VDS data file from `~/data/p700000_r0030_proc.cxi`
- select geometry from `~/geom/agipd_p700000_r0030.geom`
- in menu `Tools -> Jump to frame` choose frame number `//95`
- scroll and vary the scale on the right fo better data visibility
- in `Peak detection` tab:
  - select 'peakfinder8'
  - change minimum number of pixels to 1
- in `Index this frame` tab:
  - select cell file from `~/cell/hewl.cell`
  - chose only `XGANDALF` as indexing method
  - deselect all additional options

This should provide some guidance on how to start using the GUI - feel free to modify any parameters or select a different frame. A list of interesting frames can be found at:

```
~/xwiz/example_files/indexed_p700000_r0030.lst
```



Bragg peaks detection and indexing

EXtra-Xwiz supports a few different versions of the CrystFEL suite which can be selected in the `[crystfel]` configuration block:

In [15]:
```
for line in xwiz_config[8:12]:
    print(line, end='')
```

```
[crystfel]
# Available versions on Maxwell: '0.8.0', '0.9.1', '0.10.2', 'maxwell_dev
'
# Available version on VISA: '0.10.2_visa'
version = '0.10.2'
```

CrystFEL version `'0.10.2_visa'` corresponds to a specific installation on VISA, on Maxwell it should be modified to `'0.10.2'`.

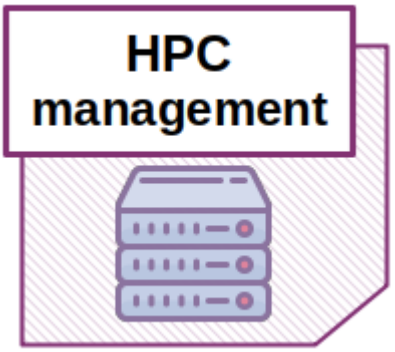
The parameters for Bragg peak identification and indexing using the indexamajig program have to be specified in the `[indexamajig_run]` configuration section:

In [16]:
```
for line in xwiz_config[24:39]:
    print(line, end='')
```

```
[indexamajig_run]
resolution = 1.6
peak_method = "peakfinder8"
peak_threshold = 800
peak_snr = 5
peak_min_px = 1
peak_max_px = 2
peaks_hdf5_path = "entry_1/result_1"
index_method = "mosflm"
n_cores = 1
local_bg_radius = 3
integration_radii = "2,3,5"
max_res = 1600
min_peaks = 10
extra_options = "--no-non-hits-in-stream"
```

In this example `n_cores` is set to `1` only due to the limitations of the VISA virtual machine. When performing analysis on the Maxwell cluster at European XFEL this parameter is usually set to `-1`, which corresponds to using all available cores of the cluster node.

Parallel computing on HPC cluster



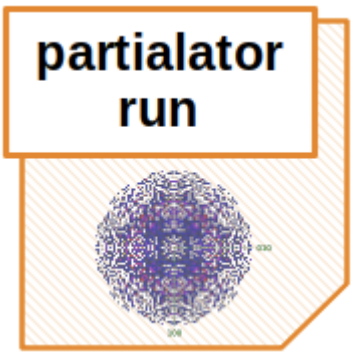
Data processing with indexamajig is the most time-consuming step of the whole pipeline, but the computations are usually performed in parallel on multiple nodes of the Maxwell cluster. Cluster partition, the number of nodes to use in parallel and maximum expected duration of the individual jobs, should be specified under the `[slurm]` section of the configuration file:

In [17]:
```
for line in xwiz_config[16:23]:
    print(line, end='')
```

```
[slurm]
# Available partitions: 'local', 'all', 'upex', 'exfel'
partition = "local"
# In case you have slurm nodes reservation
reservation = "none"
duration_all = "1:00:00"
n_nodes_all = 20
```

VISA instance is not connected to the Maxwell HPC cluster, therefore `partition` is set to `"local"` and analysis will be performed on the current node.

Merging and post-refining reflections



The reflection intensities obtained from the Bragg peaks indexing are merged and postrefined with the partialator tool, and the required parameters have to be specified under the `[merging]` block of the EXtra-Xwiz configuration:

In [18]:
```
for line in xwiz_config[59:64]:
    print(line, end='')
```

```
[merging]
point_group = "422"
scaling_model = "unity"
scaling_iterations = 1
max_adu = 100000
```

Point groups corresponding to the symmetry groups of the crystallized samples can be identified with the table the in CrystFEL documentation: https://www.desy.de/~twhite/crystfel/twin-calculator.pdf

Running EXtra-Xwiz

To start processing with the xwiz pipeline:

```
cd ~/xwiz/
mkdir t_01
cp example_files/xwiz_conf.toml t_01/
cd t_01/

xwiz-workflow -a -d
```

During processing pipeline will provide an operation log similar to:

```
-----   TASK: prepare distributed computing   -----

Reading frames list from: ../example_files
/indexed_p700000_r0030_cut.lst
Total number of frames to process: 100
Split into: 100

-----   TASK: run CrystFEL (I)   -----

Geometry file is format-compatible to corresponding data
 [cell-file read - o.k.]
 Waiting for the local process 11014
 |############-----------------------------------| 24.0%, ◆
24, Indexing rate: 100.0%
```

Pipeline output

EXtra-Xwiz will generate a `p700000_r0030.stream` with CrystFEL output, a `partialator` folder with `.hkl` unique structure factors files and figures-of-merit tables, as well as a `p700000_r0030.summary` file.

Structure factors file with the `.hkl` extension is the main produced output. In can be used with the phasing and reconstruction software to obtain protein's electron density map and identify it's structure:

Summary file contains overview of the processing results, for example:

```
vim ~/xwiz/example_results/p700000_r0030.summary
```

In [19]:
```
xwiz_summary_file = "xwiz/example_results/p700000_r0030.summary"
with open(xwiz_summary_file, 'r') as fin:
    summary_lines = fin.readlines()
    print("...")
    for line in summary_lines[61:]:
        print(line, end='')
```

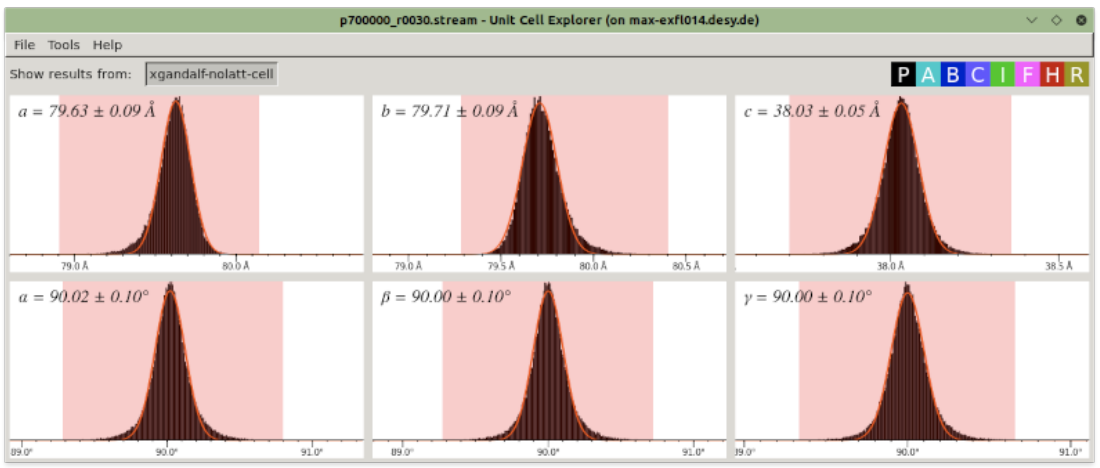
```
...
Overall frame rates:
              all_data
N_frames         21516
N_hits           21516
N_indexed        21516
hit_rate       100.000%
index_rate     100.000%

Crystallographic FOMs:
                            all_data
                          overall   outer shell
Completeness                100.0         100.0
Signal-over-noise           3.518          0.83
CC_1/2                     0.8272       0.01392
CC*                        0.9515        0.1657
R_split                     35.23         100.0
```

In our example figures of merit are a bit worse since we are processing only 100 crystals.

CrystFEL **output stream file** contains a lot of useful information and can be used, for example, to plot the distribution of cell parameters. For this run in the terminal within the visa image:

```
cell_explorer ~/xwiz/t_01/p700000_r0030.stream
```

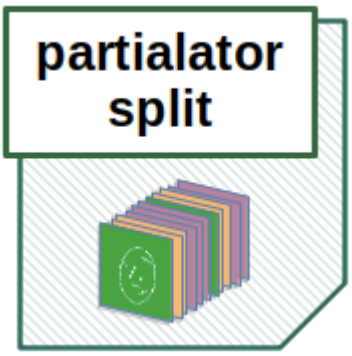
Distributions of cell parameters from our test run `t_01` will be pretty low in statistics, but if we would have processed the whole run they should look like:



To fit the six unit cell parameters:

- Adjust the scale for each parameter with the mouse scrolling while the pointer is over the distribution figure
- Drag the mouse with shift left-button to mark an interval around the peak, for all 6 parameters
- Select `Tools -> Fit cell` from the menu

Split frames for time-resolved experiments



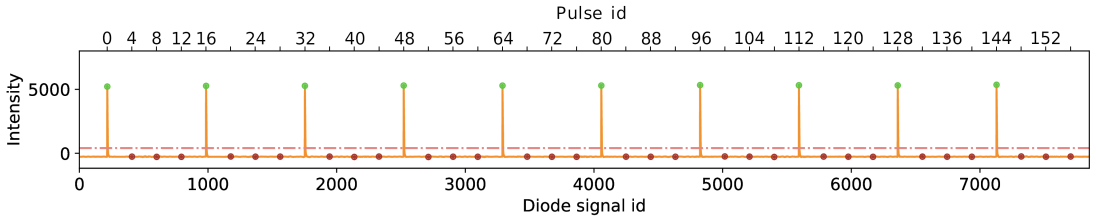
In time-resolved SFX experiments the sample is usually excited by optical or infrared pump laser at some temporal distance before being probed by the XFEL beam. The interpretation of such data usually relies on the difference between the illuminated ("pump on") and non-illuminated ("pump off") subsets of data. It is crucial to ensure that all of the other experimental conditions, e.g., the X-ray beam fluence, as well as data processing, are kept as close as possible between the sets. Therefore "pump on" and "pump off" frames of data are usually collected within the same train of X-ray pulses in an interleaved fashion.

Reflections data from "pump on" and "pump off" frames should be scaled and postrefined together, and split only before merging. This can be achieved with the *partialator* tool from *CrystFEL* by providing it with a file containing “data set identifier” (e.g., the pump status) for each data frame, for example:

In [20]:
```
frams_pls_file = "xwiz/example_results/frame_datasets.plst"
with open(frams_pls_file, 'r') as fin:
    frame_datasets = fin.readlines()
    for line in frame_datasets[:9]:
        print(line, end='')
```

```
p700000_r0030_vds.h5 //0 my_on
p700000_r0030_vds.h5 //1 my_off
p700000_r0030_vds.h5 //2 my_off
p700000_r0030_vds.h5 //3 my_on
p700000_r0030_vds.h5 //4 my_off
p700000_r0030_vds.h5 //5 my_off
p700000_r0030_vds.h5 //6 my_on
p700000_r0030_vds.h5 //7 my_off
p700000_r0030_vds.h5 //8 my_off
```

EXtra-Xwiz can produce such files either using the frame pattern information specified by a user or by automatically utilizing information from a diode, which records the signal from the pump laser. Example of the pum laser diode signal matched to the X-ray pulses:



Parameters for splitting time-resolved data should be specified “[partialator_split]” section of the EXtra-Xwiz configuration file. For example to utilize the signal from the pump diode:

```
[partialator_split]
execute = true
# Available modes: "on_off", "on_off_numbered", "by_pulse_id",
"by_train_id"
mode = "on_off_numbered"

# Required only for "on_off" or "on_off_numbered" modes:
xray_signal = ["SPB_LAS_SYS/ADC/UTC1-1:channel_0.output",
"data.rawData"]
laser_signal = ["SPB_LAS_SYS/ADC/UTC1-1:channel_1.output",
"data.rawData"]
```

User defined frames pattern can be specified in respect to either train or pulse ids.

In our example data pulse ids are multiples of 4:

In [21]:
```
with h5py.File(p700000_r0030_vds_file, 'r') as vds_file:
    train_arr = vds_file['/entry_1/trainId'][:1000]
    pulse_arr = vds_file['/entry_1/pulseId'][:1000]
display(pulse_arr[:12])
```

```
array([ 0,  4,  8, 12, 16, 20, 24, 28, 32, 36, 40, 44], dtype=uint64)
```

Therefore a user-defined pump laser pattern "on off off" can be specified with:

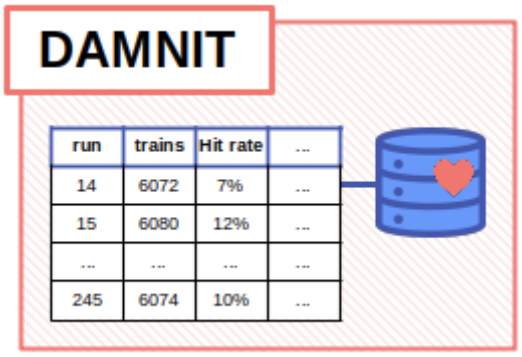
```
[partialator_split]
execute = true
mode = "by_pulse_id"

[partialator_split.manual_datasets]
  my_on = {start=0, end=-1, step=12}
  my_off = [{start=4, step=12}, {start=8, step=12}]
```

This would produce the 'xwiz/example_files/frame_datasets.plst' file mentioned before.

Merged reflection hkl files, as well as FOMs, are produced for each subset, and *EXtra-Xwiz* summarizes statics and FOMs per data set in the summary file, e.g.:

```
Overall frame rates:
              all_data       my_on      my_off
N_frames        639616      214871      424745
N_hits          244764       81785      162979
N_indexed        46898       15649       31249
hit_rate       38.267%     38.062%     38.371%
index_rate      7.332%      7.283%      7.357%

Crystallographic FOMs:
                            all_data                     my_off
my_on
                         overall   outer shell       overall
outer shell       overall   outer shell
Completeness               100.0         100.0         100.0
100.0         100.0         100.0
Signal-over-noise          4.223          0.98         3.693
0.84         3.005          0.64
CC_1/2                    0.8935      0.007131        0.8575
0.02009        0.7162      0.005451
CC*                       0.9715         0.119        0.9609
0.1985        0.9136        0.1041
R_split                    27.59         81.86         32.38
92.56         43.92         114.3
```

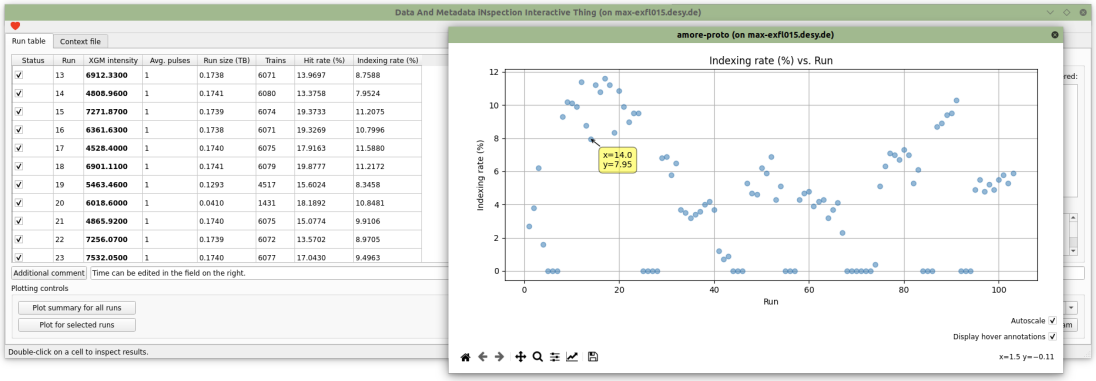
Integration with DAMNIT



On request we provide integration of EXtra-Xwiz with the DAMNIT tool:

https://rtd.xfel.eu/docs/damnit/en/latest/

Such integration allows to automatically trigger pipeline execution as soon as experimental data become available. Selected statistics and values of figures of merit are stored into a convenient table, for example:


Data And Metadata iNspection Interactive Thing (on max-exfl015.desy.de)
Run table
Context file
Status Run XGM intensity Avg. pulses Run size (TB) Trains Hit rate (%) Indexing rate (%)
13 6912.3300 1 0.1738 6071 13.9697 8.7588
14 4808.9600 1 0.1741 6080 13.3758 7.9524
15 7271.8700 1 0.1739 6074 19.3733 11.2075
16 6361.6300 1 0.1738 6071 19.3269 10.7996
17 4528.4000 1 0.1740 6075 17.9163 11.5880
18 6901.1100 1 0.1741 6079 19.8777 11.2172
19 5463.4600 1 0.1293 4517 15.6024 8.3458
20 6018.6000 1 0.0410 1431 18.1892 10.8481
21 4865.9200 1 0.1740 6075 15.0774 9.9106
22 7256.0700 1 0.1739 6072 13.5702 8.9705
23 7532.0500 1 0.1740 6077 17.0430 9.4963
Additional comment
Time can be edited in the field on the right.
Plotting controls
Plot summary for all runs
Plot for selected runs
Double-click on a cell to inspect results.
amore-proto (on max-exfl015.desy.de)
Indexing rate (%) vs. Run
Indexing rate (%)
Run
x=14.0
y=7.95
Autoscale
Display hover annotations
x=1.5 y=−0.11

In [ ]: