EUROPEAN
SPALLATION
SOURCE

Modern Metadata Modelling

Tobias Richter
European Spallation Source, Lund, Sweden

Workshop Scientific Data Management for Photon and Neutron Facilities
HZB, Berlin, March 2018

Modern Metadata Modelling?

- What is modern?

  NeXus is certainly old – over 20 years.

- What is metadata?

  There is no such thing as metadata.

  Data processing or analysis is increasingly reliant on information other than “counts”.

  Open data requires a full self consistent description of experiments.

– Perfect is the enemy of good –
Voltaire

NeXus International Advisory Committee NIAC

EUROPEAN SPALLATION SOURCE

- *Mark Basham, Diamond Light Source, UK*
- *Herbert Bernstein, CIF (non-facility member)*
- *Aaron Brewster, Lawrence Berkeley Laboratory, USA*
- *Stuart Campbell, Brookhaven National Laboratory, USA*
- *Bjorn Clausen, Los Alamos National Laboratory, USA*
- *Stephen Cottrell, Rutherford Appleton Laboratory, UK (Muon Representative)*
- *Ricardo Ferraz-Leal, SNS and HFIR at ORNL, USA*
- *Jens-Uwe Hoffmann, Helmholtz Zentrum Berlin, Germany*
- *Pete Jemian, Advanced Photon Source, USA (Documentation Release Manager)*
- *Mark Könnecke, Paul Scherrer Institute, Switzerland (Executive Secretary)*
- *Raymond Osborn, Argonne National Laboratory, USA (non-facility member)*
- *Tobias Richter, European Spallation Source, Sweden (Chair)*
- *Armando Sole, European Synchrotron Radiation Facility, France*
- *Jiro Suzuki, KEK, Japan*
- *Benjamin Watts, Swiss Light Source, Switzerland*
- *Eugen Wintersberger, DESY, Germany (Technical Manager)*

Frequent guest:

+ *Andreas Förster, Dectris, Switzerland*

NeXus – nexusformat.org

NeXus aims to provide a format that can hold (all optional):

- raw experimental data
(with all information required for processing)
- all data needed for diagnostics
- metadata
- processed data

NeXus uses HDF5 as efficient hierarchical container format for storage.

For all techniques at

- neutron
- muon
- X-ray
- soft X-ray
- VUV
- EM?

research facilities.

Aims to replace any formats that require implicit knowledge about the experiment.

HDF5 Dataset

Metadata

Dataspace

Rank	Dimensions
3	Dim_1 = 4
	Dim_2 = 5
	Dim_3 = 7

Datatype

IEEE 32-bit float

Storage info

Chunked

Compressed

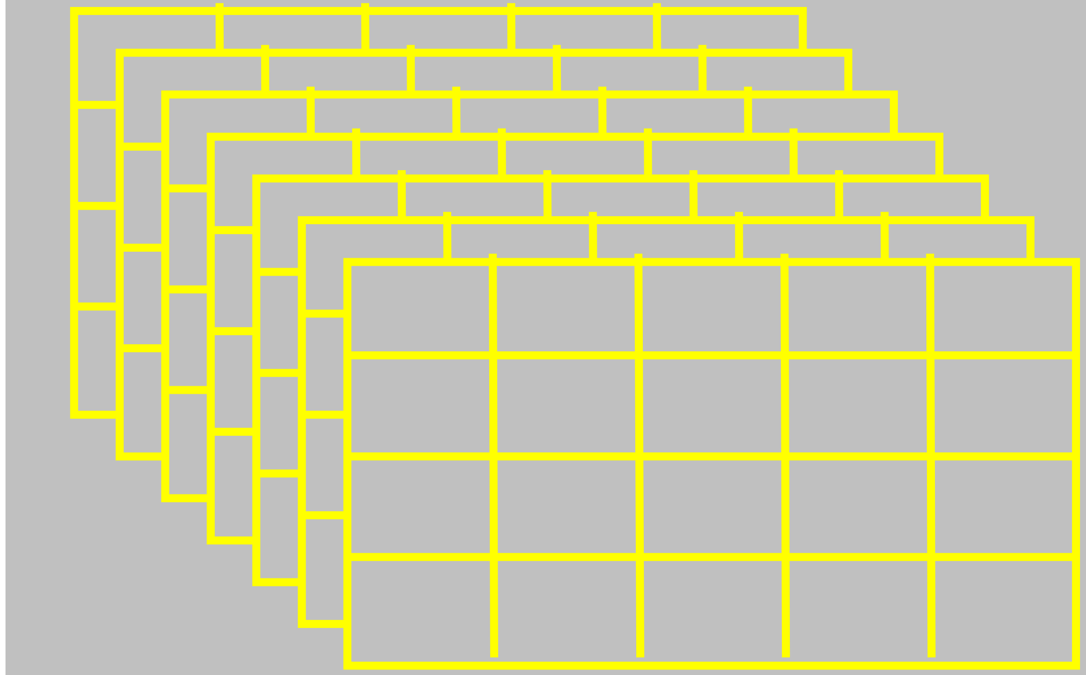
Attributes

Time = 32.4

Pressure = 987

Temp = 56

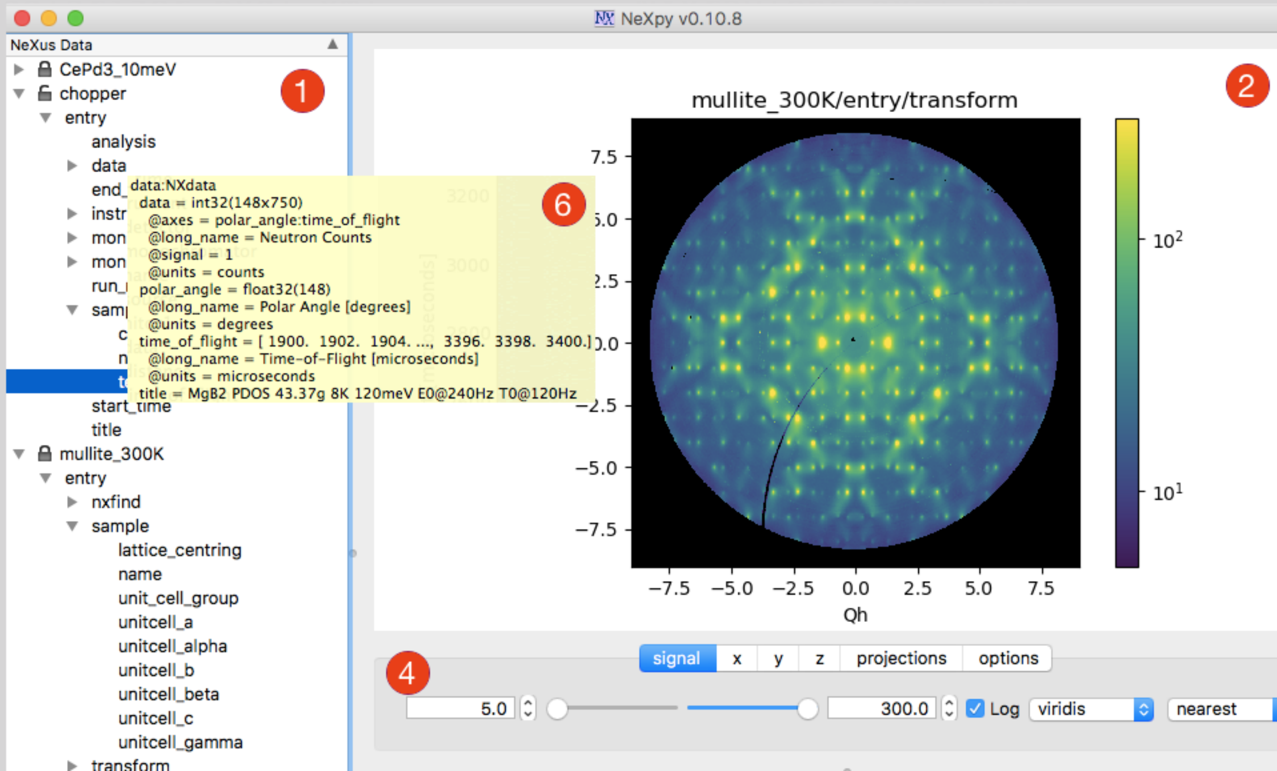
Dataset data



NeXus Base Classes

Contain dictionaries for parameters common for particular types of equipment or sample, user, etc.

```
base_classes$ ls
NXaperture.nxdl.xml             NXdetector_module.nxdl.xml      NXlog.nxdl.xml                  NXsample.nxdl.xml
NXattenuator.nxdl.xml           NXdisk_chopper.nxdl.xml         NXmirror.nxdl.xml               NXsensor.nxdl.xml
NXbeam.nxdl.xml                 NXentry.nxdl.xml                NXmoderator.nxdl.xml            NXshape.nxdl.xml
NXbeam_stop.nxdl.xml            NXenvironment.nxdl.xml          NXmonitor.nxdl.xml              NXslit.nxdl.xml
NXbending_magnet.nxdl.xml       NXevent_data.nxdl.xml           NXmonochromator.nxdl.xml        NXsource.nxdl.xml
NXcapillary.nxdl.xml            NXfermi_chopper.nxdl.xml        NXnote.nxdl.xml                 NXsubentry.nxdl.xml
NXcharacterization.nxdl.xml     NXfilter.nxdl.xml               NXobject.nxdl.xml               NXtransformations.nxdl.xml
NXcite.nxdl.xml                 NXflipper.nxdl.xml              NXorientation.nxdl.xml          NXtranslation.nxdl.xml
NXcollection.nxdl.xml           NXfresnel_zone_plate.nxdl.xml   NXparameters.nxdl.xml           NXuser.nxdl.xml
NXcollimator.nxdl.xml           NXgeometry.nxdl.xml             NXpinhole.nxdl.xml              NXvelocity_selector.nxdl.xml
NXcrystal.nxdl.xml              NXgrating.nxdl.xml              NX
NXdata.nxdl.xml                 NXguide.nxdl.xml                NX
NXdetector.nxdl.xml             NXinsertion_device.nxdl.xml     NX
NXdetector_group.nxdl.xml       NXinstrument.nxdl.xml           NX
```

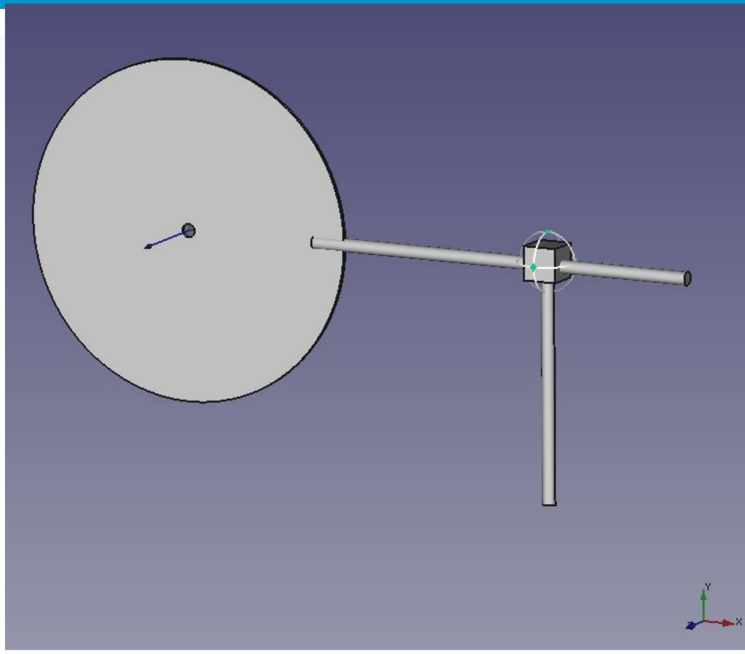


```
data:NXdata
  data = int32(148x750)
    @axes = polar_angle:time_of_flight
    @long_name = Neutron Counts
    @signal = 1
    @units = counts
  polar_angle = float32(148)
    @long_name = Polar Angle [degrees]
    @units = degrees
  time_of_flight = [ 1900.  1902.  1904. ...,  3396.  3398.  3400.]
    @long_name = Time-of-Flight [microseconds]
    @units = microseconds
  title = MgB2 PDOS 43.37g 8K 120meV E0@240Hz T0@120Hz
```

With those you can build up a hierarchy describing a fairly complete description of an experiment.

Agreement on those classes and further definitions across facilities benefits both users and software developers.

Example: NXpinhole



3.3.1.38. NXpinhole

Status:

base class, extends *NXobject*, version 1.0

Description:

Template of a simple pinhole. For more complex geometries NXaperture should be used.

Symbols:

No symbol table

Groups cited:

none

Structure:

depends_on: *NX_CHAR*

Points to the path of the last element in the geometry chain that places this object in space. When followed through that chain is supposed to end at an element depending on "." i.e. the origin of the coordinate system. If desired the location of the slit can also be described relative to an NXbeam, which will allow a simple description of a non-centred pinhole.

diameter: *NX_NUMBER* {units=*NX_LENGTH*}

Size of the circular hole defining the transmitted beam size.

NXDL Source:

https://github.com/nexusformat/definitions/blob/master/base_classes/NXpinhole.nxdl.xml

Application Definitions

- Guarantee the presence of base classes and fields expected for one specific type of experiment or measurement.
- Defined in XML, for static validation and documentation.
- Community adoption is usually slow - NXmx is a success story
- Traditional levels of NeXus adoption:

    1. HDF5 container
    2. base classes used (all content optional)
    3. application definition followed

Application Definitions

- NXarchive
- NXarpes
- NXcanSAS
- NXdirecttof
- NXfluo
- NXindirecttof
- NXiqproc
- NXlauetof
- NXmonopd
- NXmx
- NXrefscan
- NXreftof
- NXsas
- NXsastof
- NXscan
- NXspe
- NXsqom
- NXstxm
- NXtas
- NXtofnpd
- NXtofraw
- NXtofsingle
- NXtomo
- NXtomophase
- NXtomoproc
- NXxas
- NXxasproc
- NXxbase
- NXxeuler
- NXxkappa
- NXxlaue
- NXxlaueplate
- NXxnb
- NXxrot

Those ensure presence of relevant information for a specific technique.

Defined via an XML schema that allows formal validation of files.

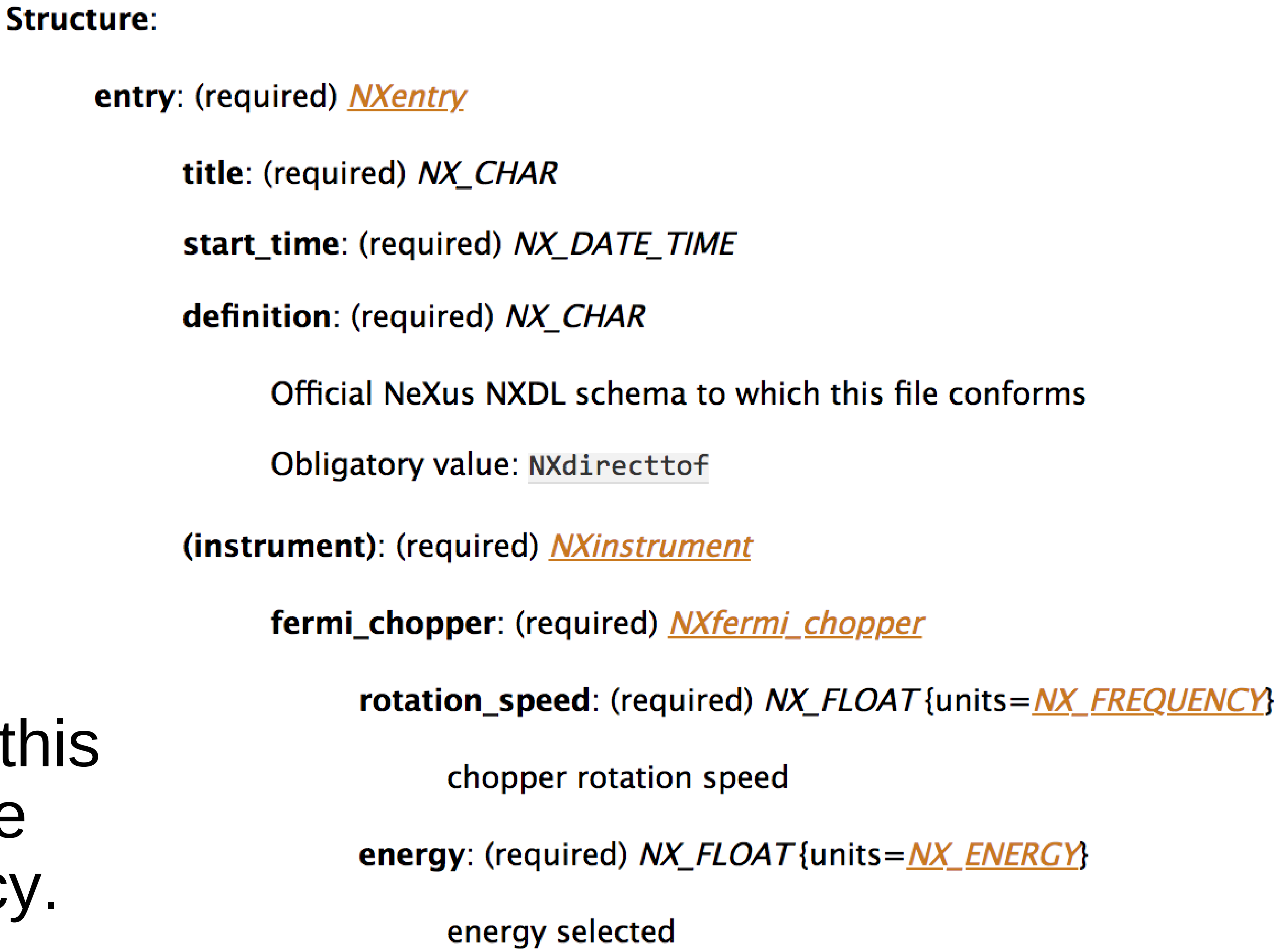
Example: NXdirecttof

For time of flight spectrometers.

What is defined?

What isn't?

Slightly strange, this example could be considered legacy.



Structure:

- **entry**: (required) *NXentry*
  - **title**: (required) *NX_CHAR*
  - **start_time**: (required) *NX_DATE_TIME*
  - **definition**: (required) *NX_CHAR*
    - Official NeXus NXDL schema to which this file conforms
    - Obligatory value: `NXdirecttof`
  - **(instrument)**: (required) *NXinstrument*
    - **fermi_chopper**: (required) *NXfermi_chopper*
      - **rotation_speed**: (required) *NX_FLOAT* {units=*NX_FREQUENCY*}
        - chopper rotation speed
      - **energy**: (required) *NX_FLOAT* {units=*NX_ENERGY*}
        - energy selected

Metadata for Scans

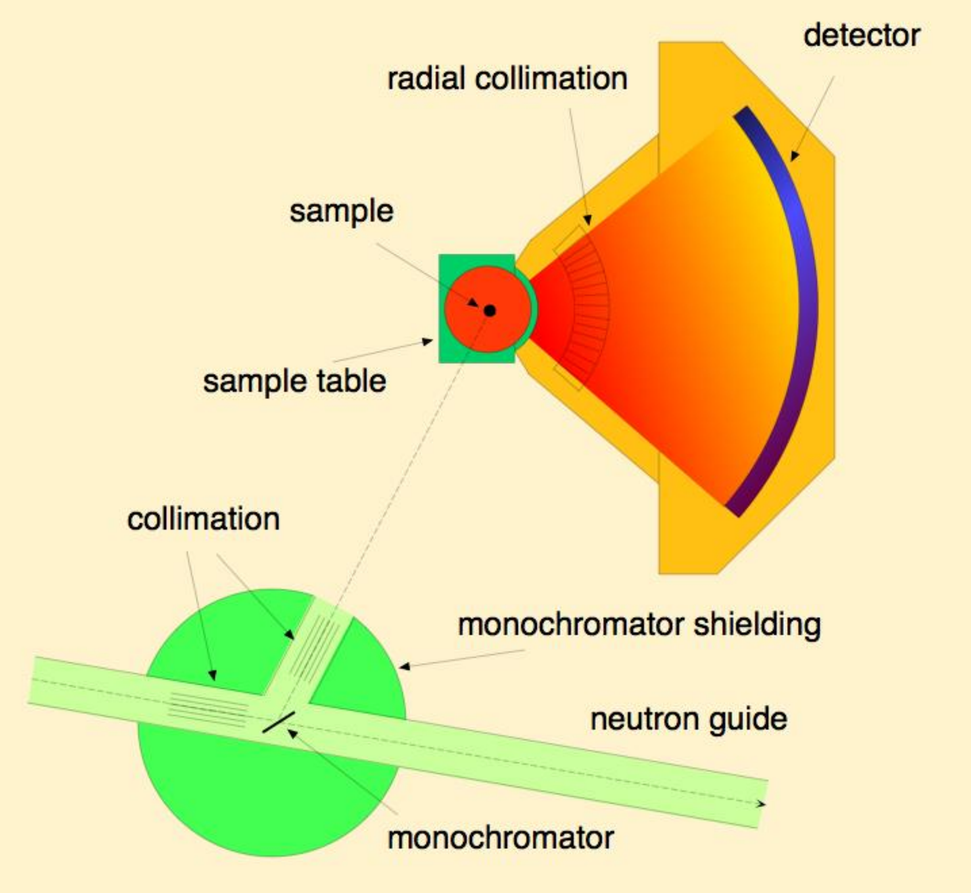
From simple to complex or
from old school to modern.

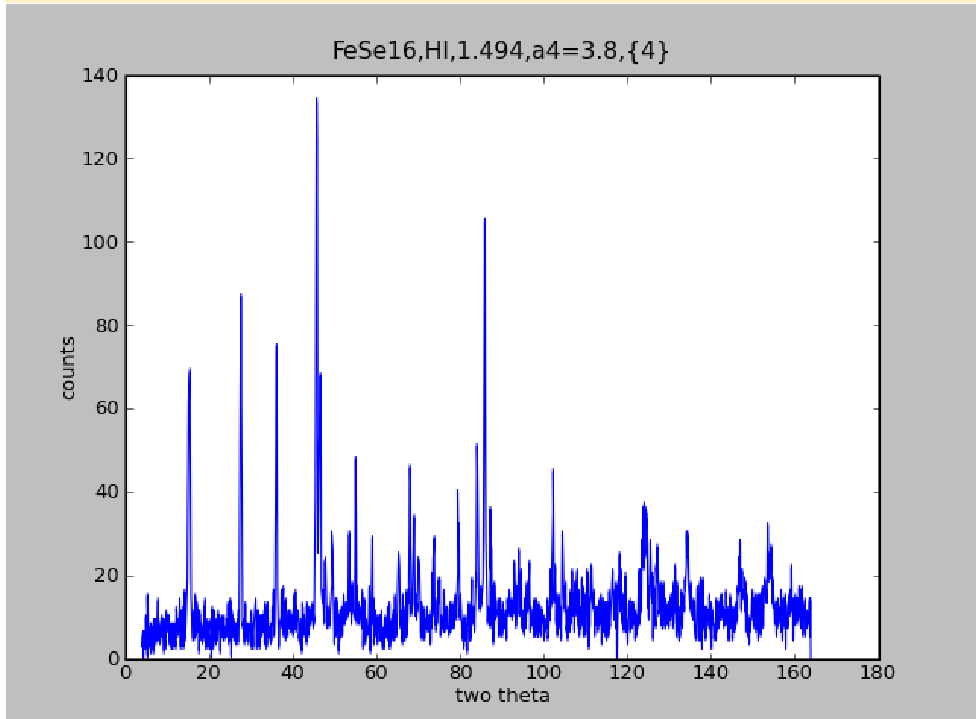
- Static Exposure
- NeXus Scan Rules
- Multiple Dimensions
- Time Stamp Everything

Static Exposure

“sit and count” – old school

entry:NXentry
  NXdata
    data[1024]
    two_theta[1024]
  NXinstrument
   NXdetector
     data[1024]
  NXsample





This is what standard application definitions specify.

Pros	Cons
Easy	Very limiting

“Scan Rules”

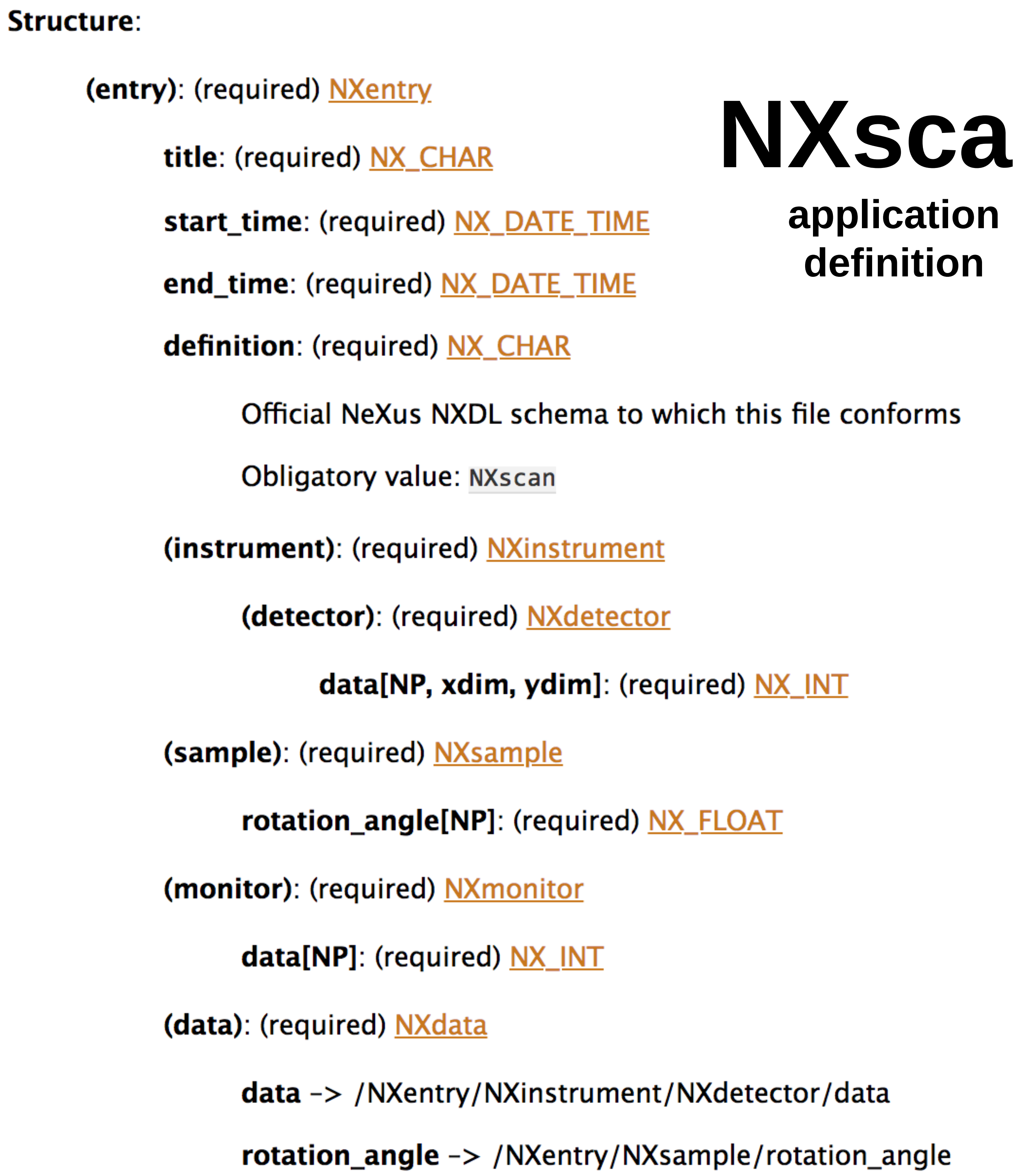
Just add one (or more) parameter.

```
entry:NXentry
  NXdata
     data[101,1024]
     two_theta[1024]
     rotation_angle[101]
```

Pros	Cons
Looks simple	Can start to get confusing, does not cover every case

NXscan

application definition



Structure:

- **(entry)**: (required) NXentry
  - **title**: (required) NX_CHAR
  - **start_time**: (required) NX_DATE_TIME
  - **end_time**: (required) NX_DATE_TIME
  - **definition**: (required) NX_CHAR

    Official NeXus NXDL schema to which this file conforms

    Obligatory value: `NXscan`
  - **(instrument)**: (required) NXinstrument
    - **(detector)**: (required) NXdetector
      - **data[NP, xdim, ydim]**: (required) NX_INT
  - **(sample)**: (required) NXsample
    - **rotation_angle[NP]**: (required) NX_FLOAT
  - **(monitor)**: (required) NXmonitor
    - **data[NP]**: (required) NX_INT
  - **(data)**: (required) NXdata
    - **data** -> /NXentry/NXinstrument/NXdetector/data
    - **rotation_angle** -> /NXentry/NXsample/rotation_angle

“Scan Rules” cont’d

Unroll all scan parameters to a 1D table (recommendation from 2010).

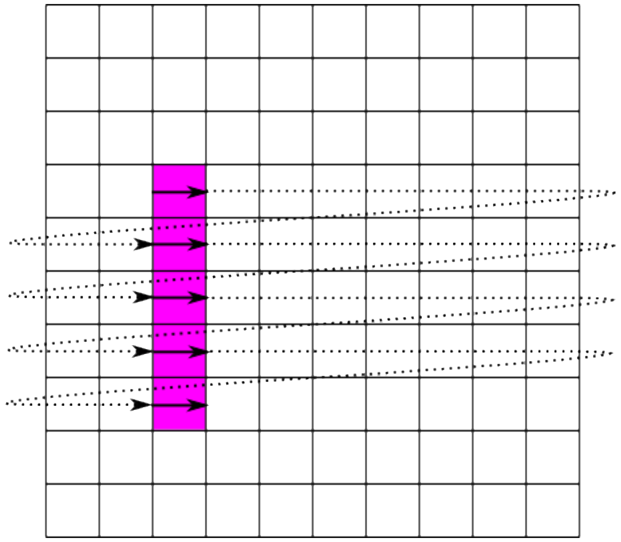
Works for 2D spatial scans or other complex examples:

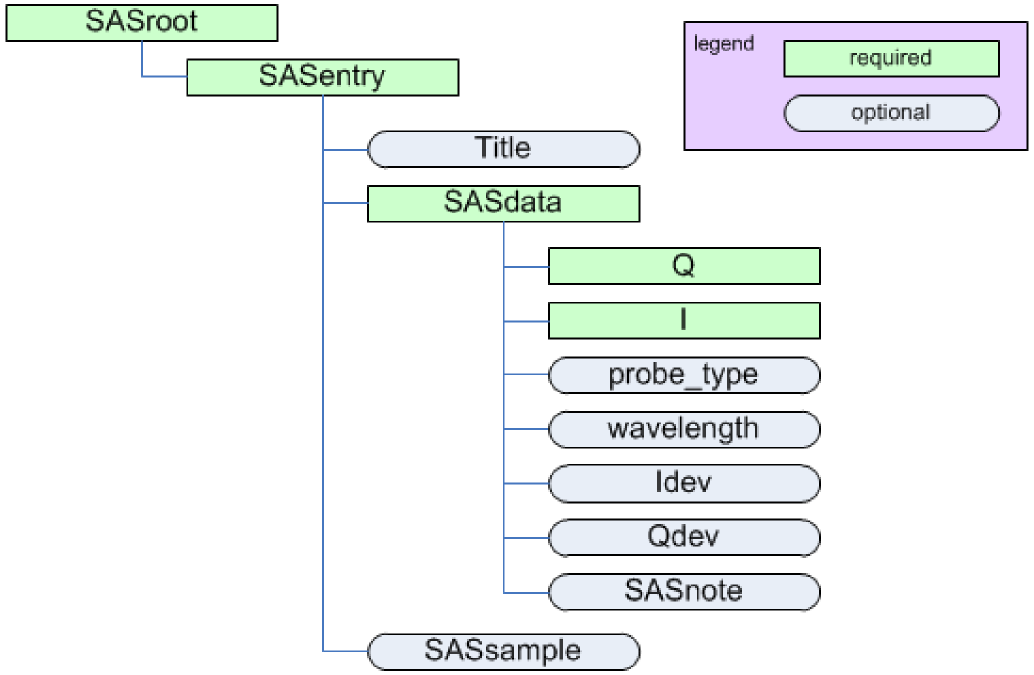
```
entry:NXentry
 NXdata
   data[21*101,1024]
   time_of_flight[1024]
   xscan[21*101]
   yscan[21*101]
```

Pros	Cons
Generic solution	Confusing, requires post processing for plotting

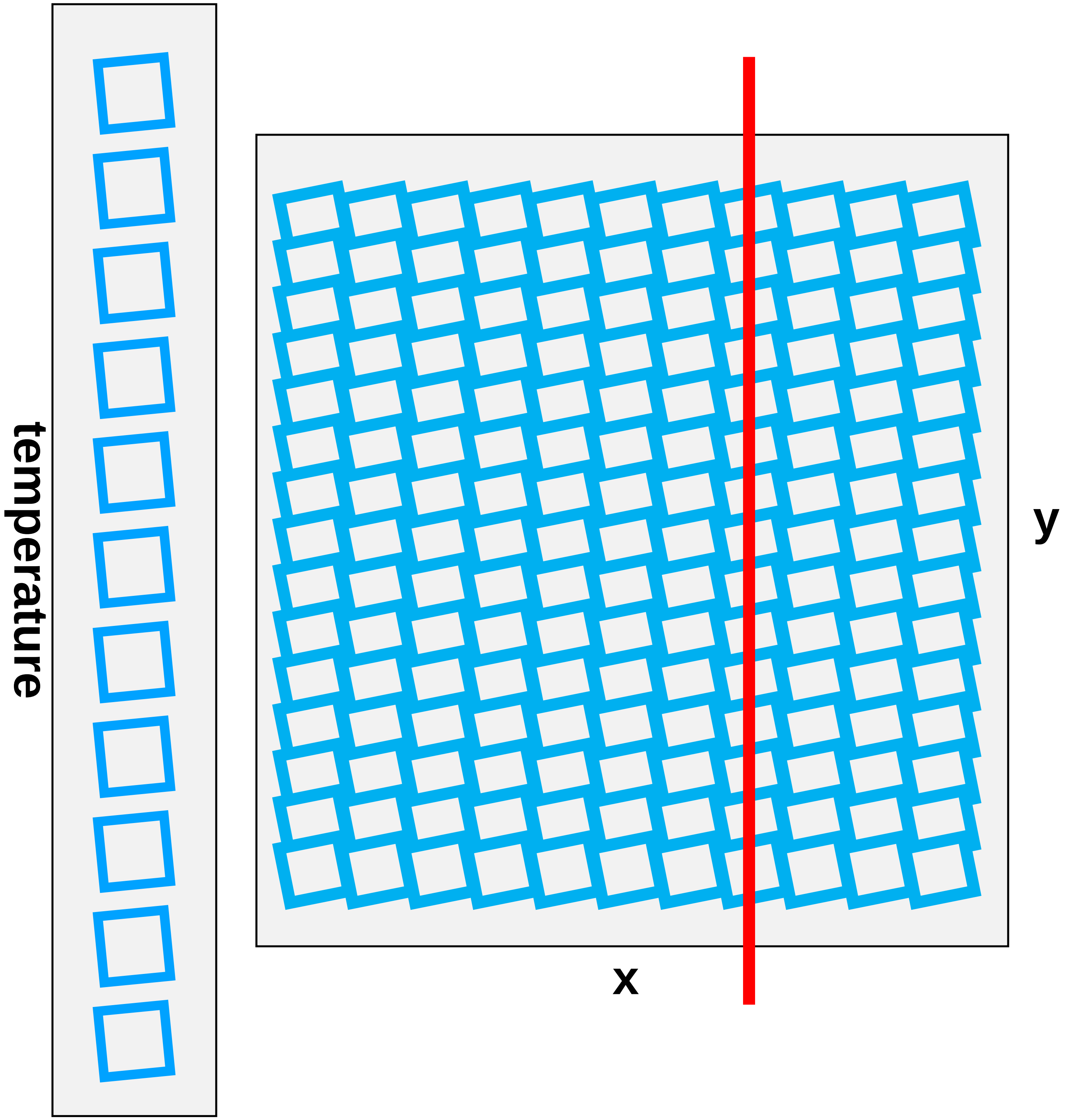
Aside: canSAS / NXcanSAS

- The collective action for nomadic small angle scatterers (canSAS) working group defined a standard for multi dimensional reduced (processed) data in 2012.
- That that allows for slicing of intensities along individual parameter dimensions.





Multi dimensional data

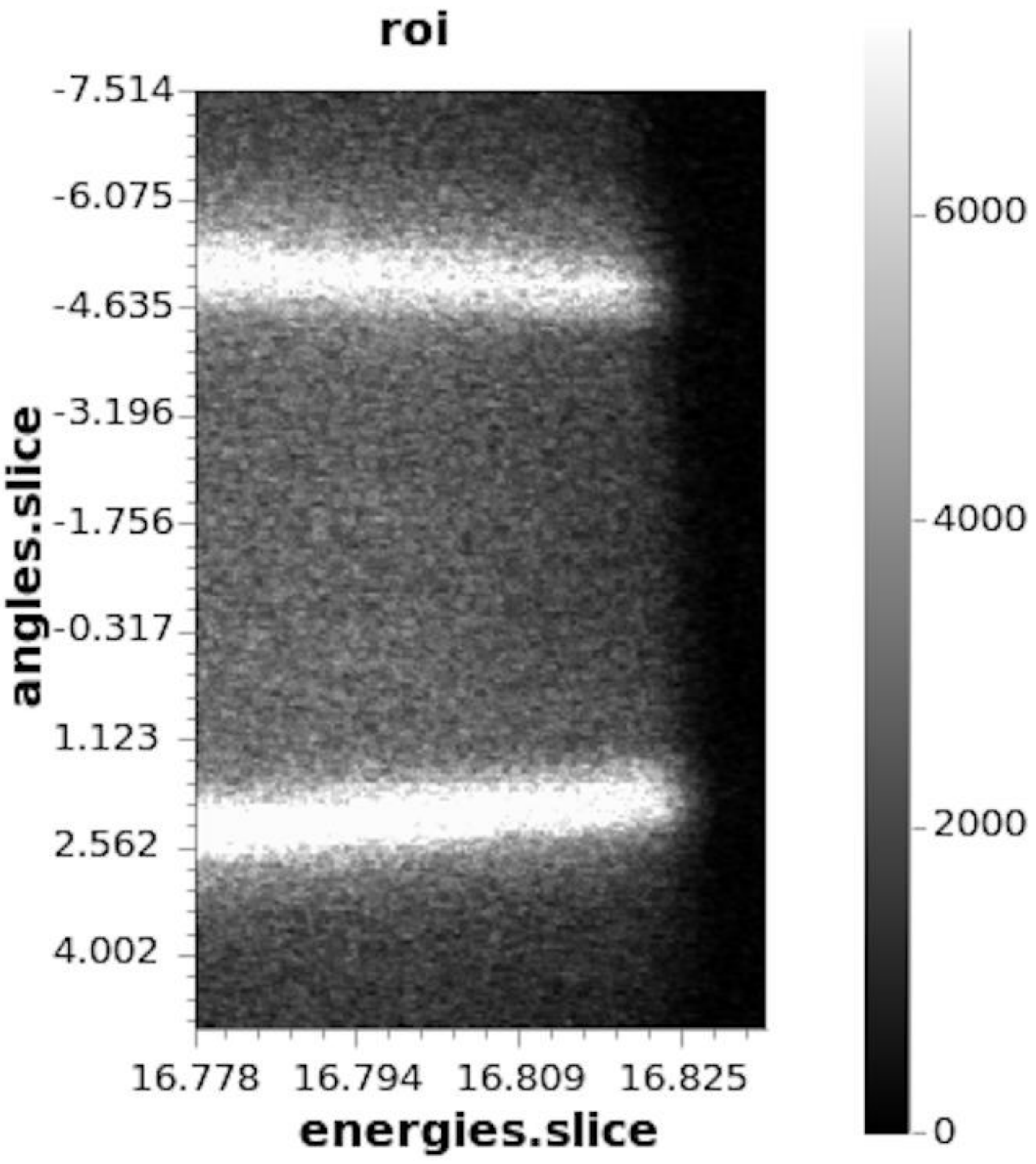


```
entry:NXentry
  NXdata
    data[11,1024,1024,512]
    x_scan[1024]
    y_scan[1024]
    energy[1024,1024,512]
    temperature[11]
```

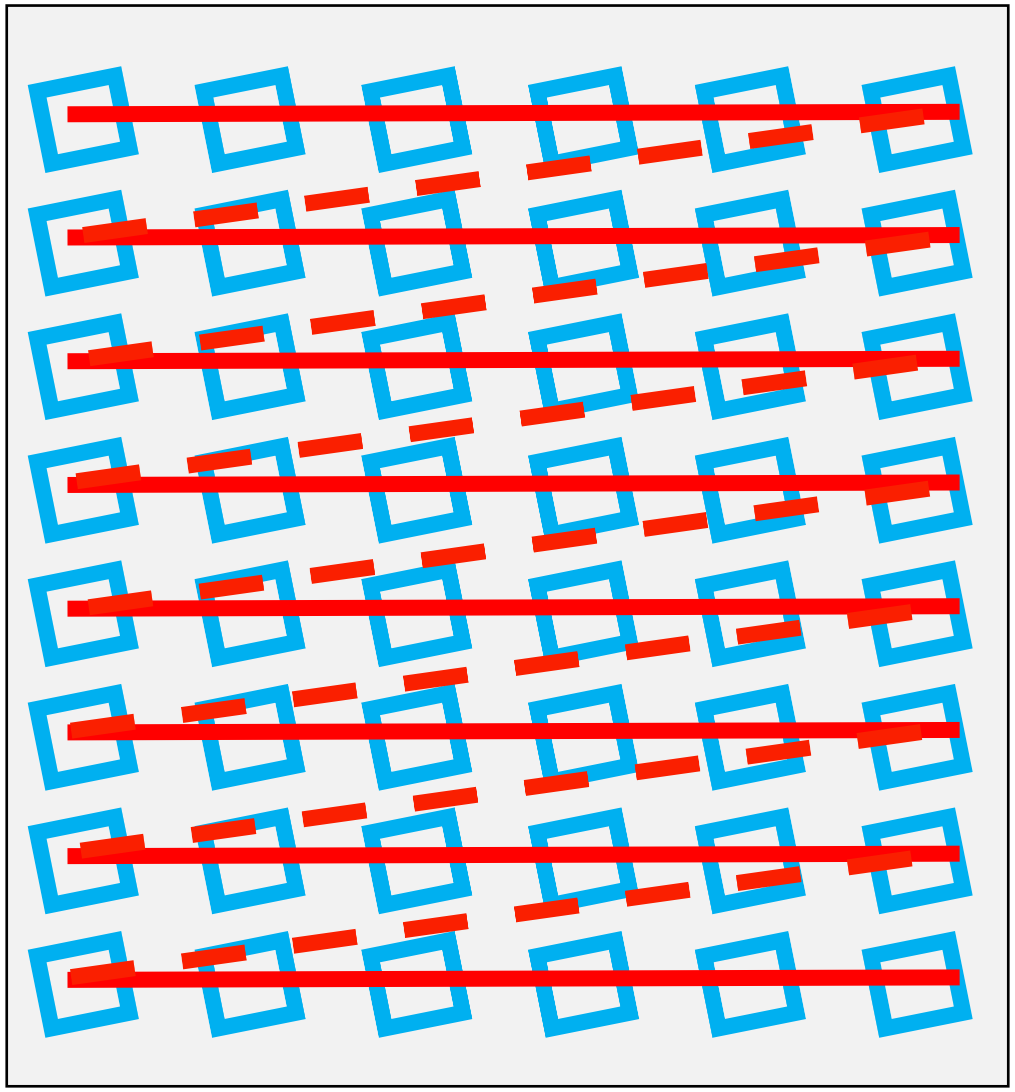
Pros	Cons
Easy slicing and plotting, captures scan intend	Limited to rectangular geometries

Plotting via Slicing - NXdata

- Multi-dimensional data stored in native HDF5 arrays allow slicing and hence simple visualization without custom tools
- Multiple, alternative axes can be specified in many dimensions
  (HDF dimension scales are not as powerful)

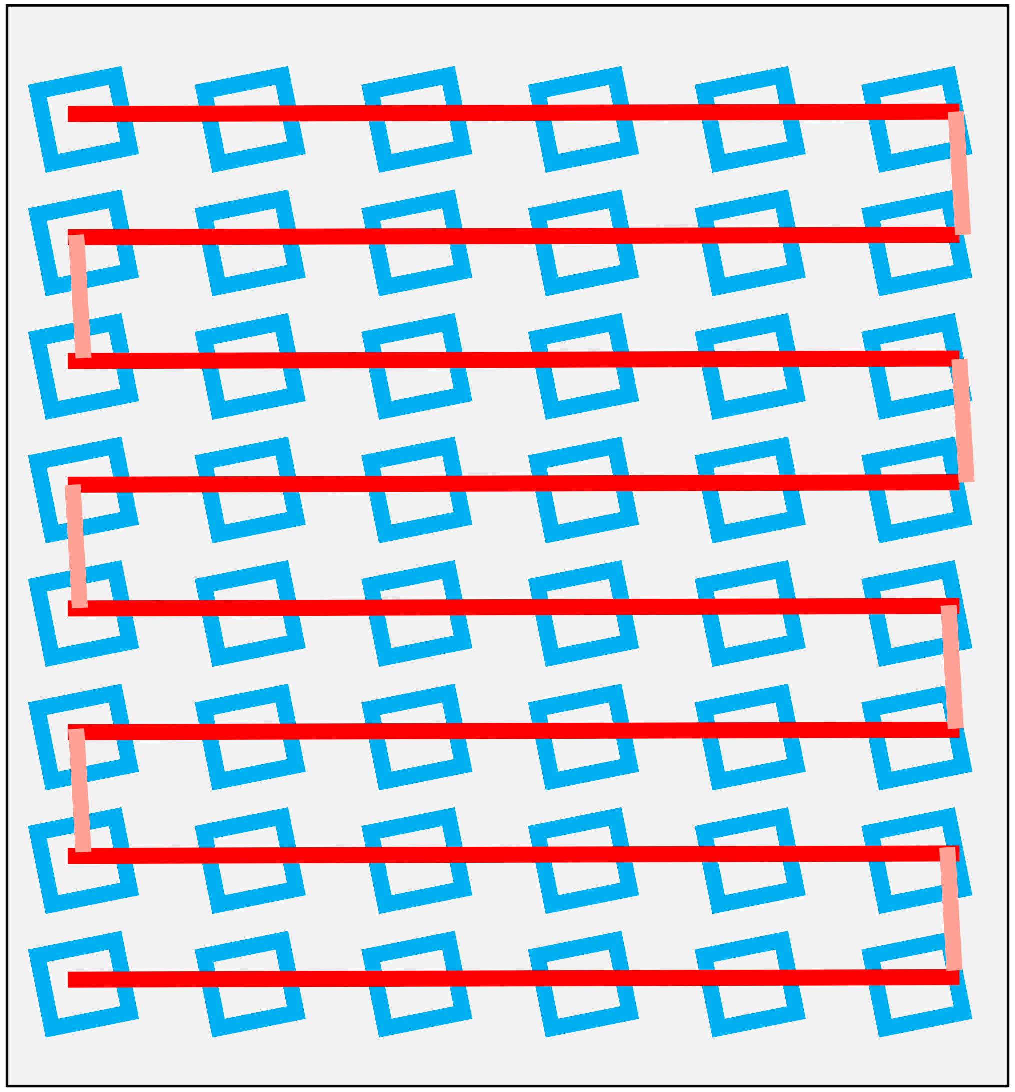


Efficient multi dimensional DAQ



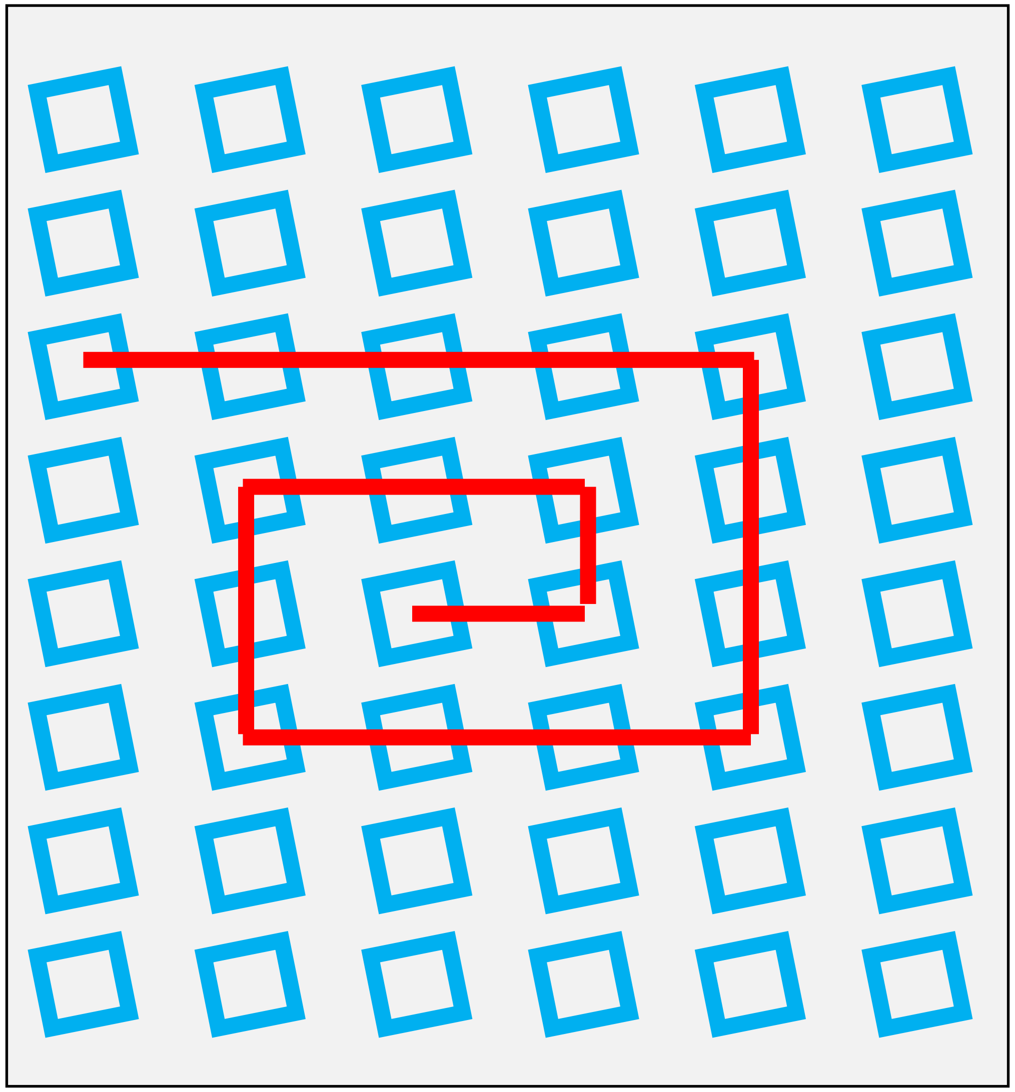
- To optimise data recording and reduce dead time it often makes sense to fill the HDF5 arrays not in their natural order.
- Requires complex synchronisation of control and data acquisition and can lead to problems with aborted scans.
- For fast fly- rather than step-scans, special hardware may be required for triggering exposures and/or reading encoders.

Efficient multi dimensional DAQ



- To optimise data recording and reduce dead time it often makes sense to fill the HDF5 arrays not in their natural order.
- Requires complex synchronisation of control and data acquisition and can lead to problems with aborted scans.
- For fast fly- rather than step-scans, special hardware may be required for triggering exposures and/or reading encoders.

Efficient multi dimensional DAQ

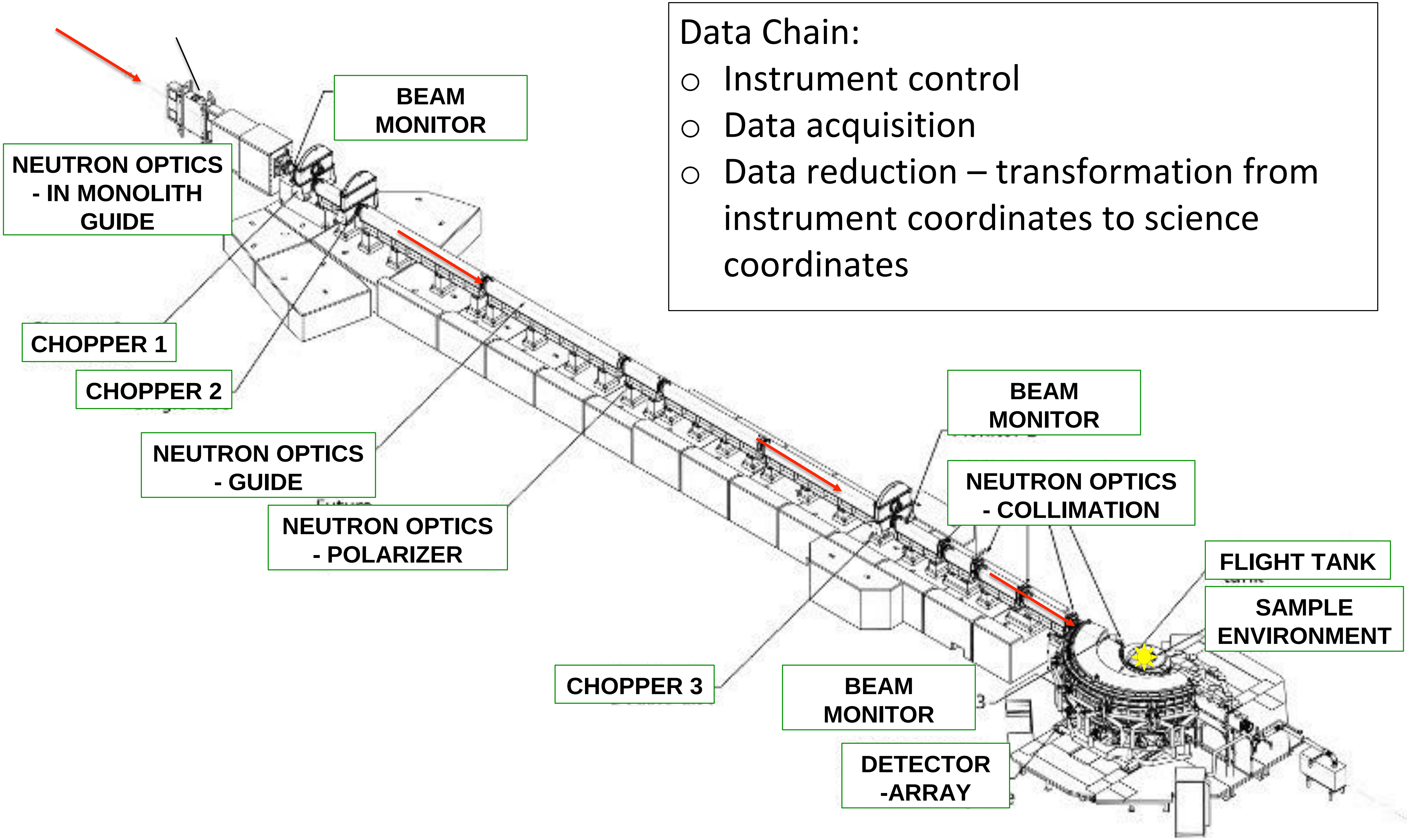


- To optimise data recording and reduce dead time it often makes sense to fill the HDF5 arrays not in their natural order.
- Requires complex synchronisation of control and data acquisition and can lead to problems with aborted scans.
- For fast fly- rather than step-scans, special hardware may be required for triggering exposures and/or reading encoders.

Time of Flight Neutron Instruments

Data Chain:

- ○ Instrument control
- ○ Data acquisition
- ○ Data reduction – transformation from instrument coordinates to science coordinates



Asynchronous recording: NXlog

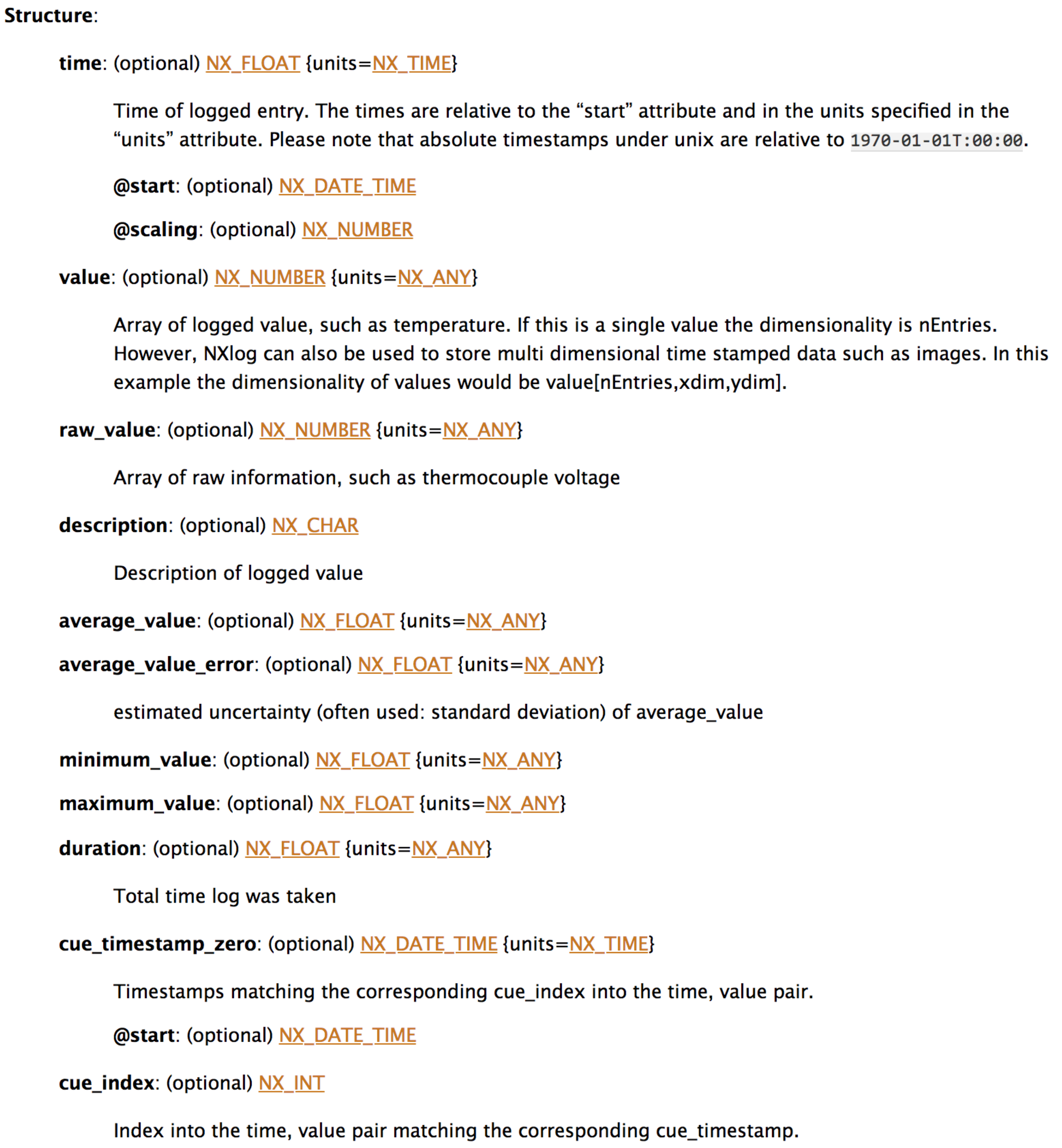
Group for asynchronous time stamped data that replaces a dataset in a base class or application definition.

Fits well with neutron event recording.

Brings us closer to a full description of the instrument.

- entry:NXentry
  - NXdata
    - data:NXlog
    - polarisation:NXlog
    - temperature:NXlog
    - rotation_angle:NXlog

Pros	Cons
very flexible, efficient storage, requires little to no DAQ support	No default plot, requires post processing



Structure:

time: (optional) NX_FLOAT {units=NX_TIME}

Time of logged entry. The times are relative to the “start” attribute and in the units specified in the “units” attribute. Please note that absolute timestamps under unix are relative to `1970-01-01T:00:00`.

@start: (optional) NX_DATE_TIME

@scaling: (optional) NX_NUMBER

value: (optional) NX_NUMBER {units=NX_ANY}

Array of logged value, such as temperature. If this is a single value the dimensionality is nEntries. However, NXlog can also be used to store multi dimensional time stamped data such as images. In this example the dimensionality of values would be value[nEntries,xdim,ydim].

raw_value: (optional) NX_NUMBER {units=NX_ANY}

Array of raw information, such as thermocouple voltage

description: (optional) NX_CHAR

Description of logged value

average_value: (optional) NX_FLOAT {units=NX_ANY}

average_value_error: (optional) NX_FLOAT {units=NX_ANY}

estimated uncertainty (often used: standard deviation) of average_value

minimum_value: (optional) NX_FLOAT {units=NX_ANY}

maximum_value: (optional) NX_FLOAT {units=NX_ANY}

duration: (optional) NX_FLOAT {units=NX_ANY}

Total time log was taken

cue_timestamp_zero: (optional) NX_DATE_TIME {units=NX_TIME}

Timestamps matching the corresponding cue_index into the time, value pair.

@start: (optional) NX_DATE_TIME

cue_index: (optional) NX_INT

Index into the time, value pair matching the corresponding cue_timestamp.

Asynchronous recording: NXlog

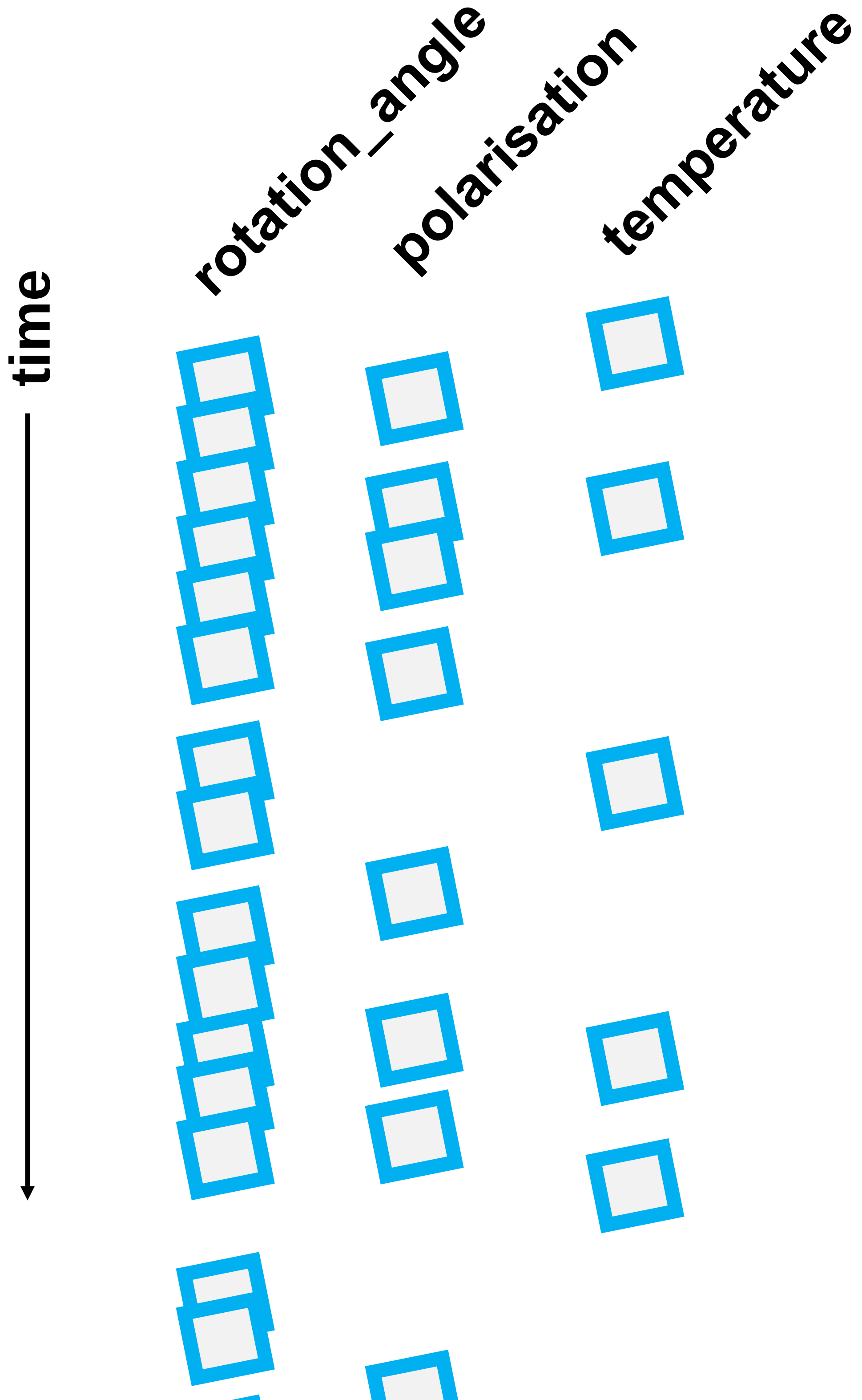
Group for asynchronous time stamped data that replaces a dataset in a base class or application definition.

Fits well with neutron event recording.

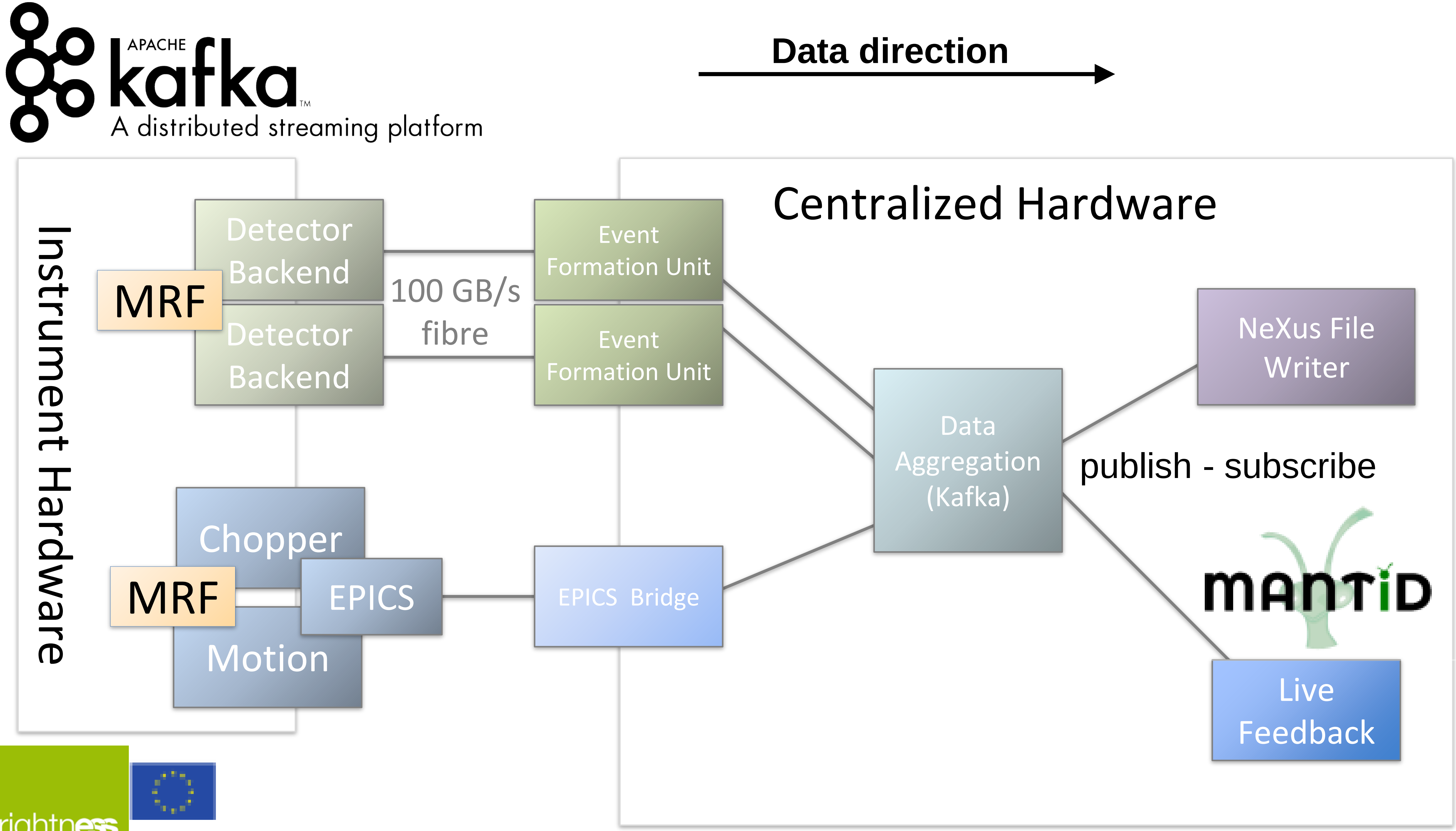
Brings us closer to a full description of the instrument.

```
entry:NXentry
  NXdata
     data:NXlog
     polarisation:NXlog
     temperature:NXlog
     rotation_angle:NXlog
```

Pros	Cons
very flexible, efficient storage, requires little to no DAQ support	No default plot, requires post processing

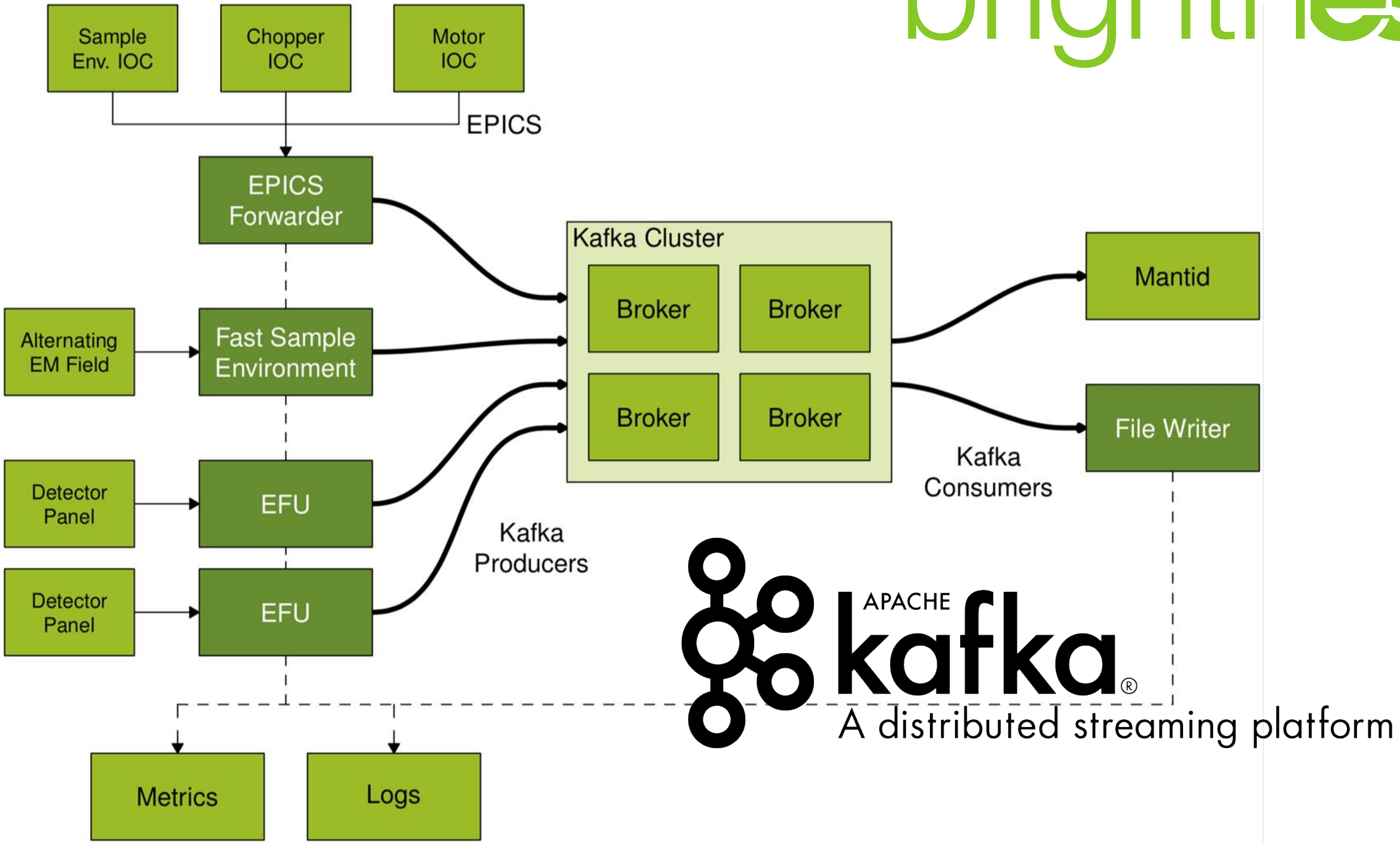


ESS Data Pipeline



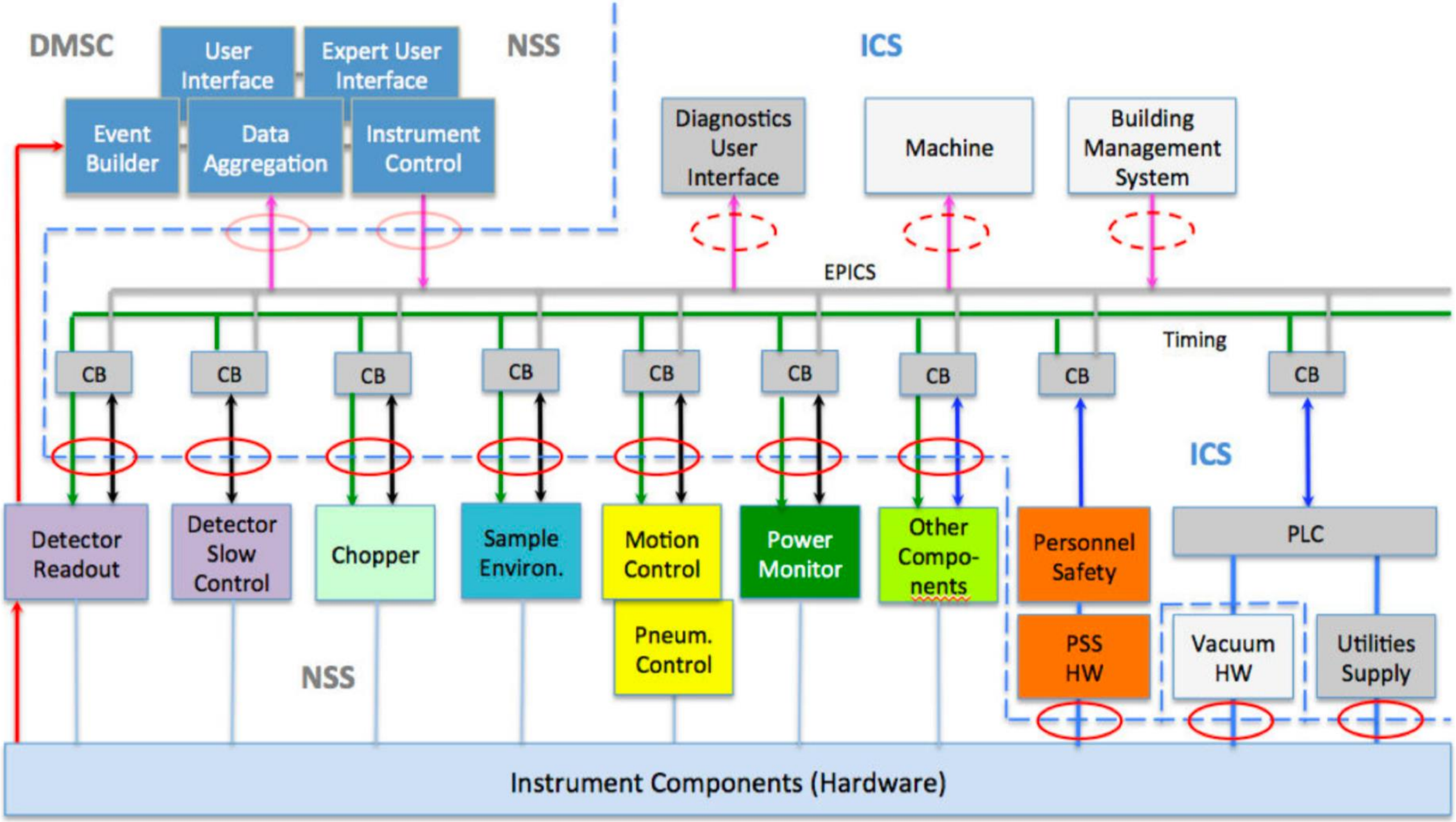
BrightnESS is funded by the European Union's Horizon 2020 research and innovation programme under grant agreement No. 676548

ESS DAQ with Kafka



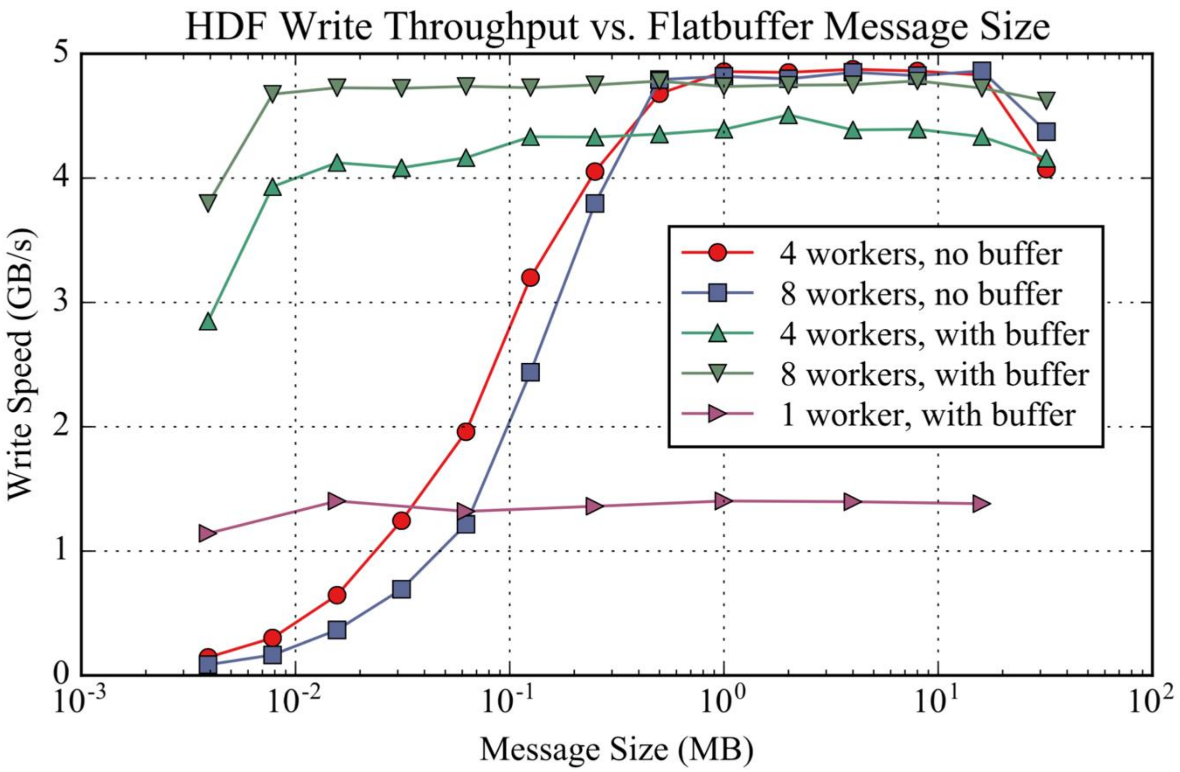
time-stamped, event based data acquisition

ESS Controls Architecture



Kafka-to-NeXus FileWriter

- receives messages with serialised using Google Flatbuffers
- writes asynchronous messages to NeXus, structure defined in a JSON document
- configuration and control via dedicated Kafka topic



PAUL SCHERRER INSTITUT PSI

brightness

How do you support this in software?

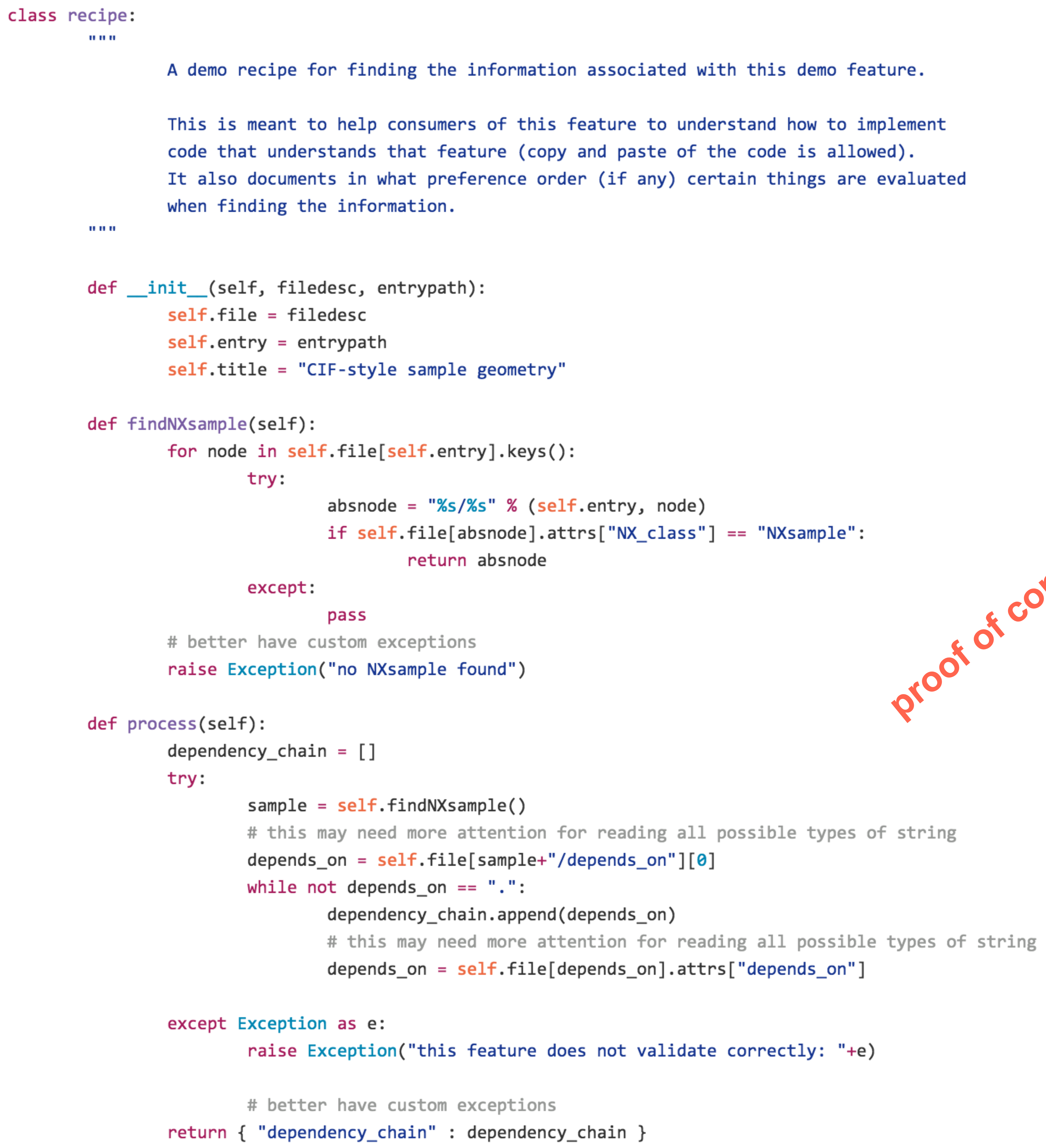
- Every scheme puts a burden on developers of NeXus and consuming software.
- Everything needs to be documented and tested.
- The question is not:
  Does the file or software comply yes or no?
- We need to know:
  - Does my file correctly capture the state of the instrument at any time?
  - Does my software extract the information correctly?
- Not all code will read all files.

Modular File Content – “Features”

Goal:
Finer granularity control of how information is kept in the file.

- Use readable Python code both to support the documentation and as reference implementation.
- Works like a unit test for data file and processing code.
- Another example:
  Code could should how to extract the incident wavelength spectrum on the sample which could be encoded as:
  - as parameter of the source
  - as parameter of the monochromator (if one exists)
  - as property of incident beam on sample

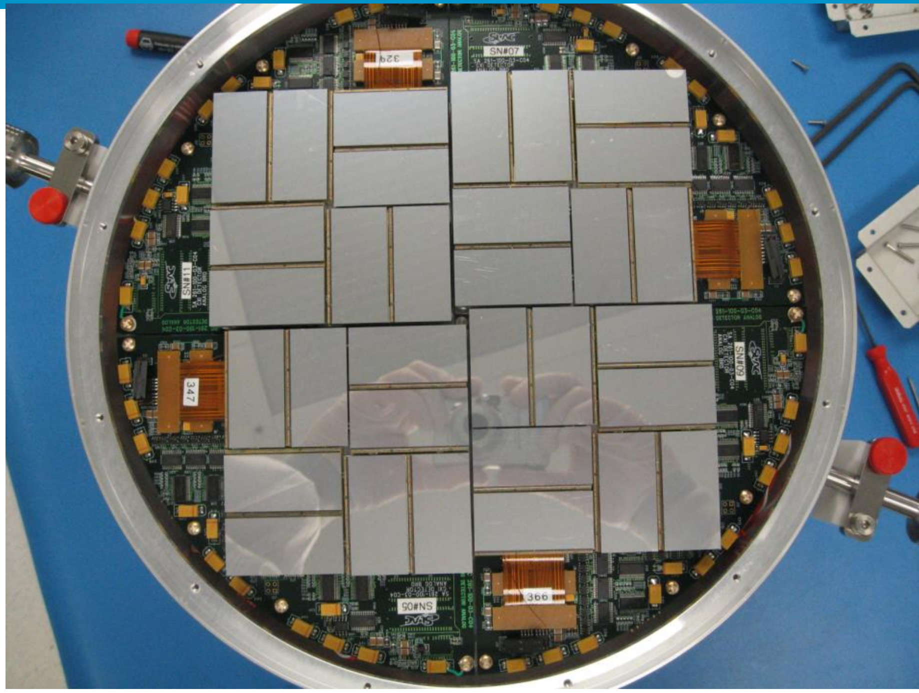
Example Recipe



```
class recipe:
        """
                A demo recipe for finding the information associated with this demo feature.

                This is meant to help consumers of this feature to understand how to implement
                code that understands that feature (copy and paste of the code is allowed).
                It also documents in what preference order (if any) certain things are evaluated
                when finding the information.
        """

        def __init__(self, filedesc, entrypath):
                self.file = filedesc
                self.entry = entrypath
                self.title = "CIF-style sample geometry"

        def findNXsample(self):
                for node in self.file[self.entry].keys():
                        try:
                                absnode = "%s/%s" % (self.entry, node)
                                if self.file[absnode].attrs["NX_class"] == "NXsample":
                                        return absnode
                        except:
                                pass
                # better have custom exceptions
                raise Exception("no NXsample found")

        def process(self):
                dependency_chain = []
                try:
                        sample = self.findNXsample()
                        # this may need more attention for reading all possible types of string
                        depends_on = self.file[sample+"/depends_on"][0]
                        while not depends_on == ".":
                                dependency_chain.append(depends_on)
                                # this may need more attention for reading all possible types of string
                                depends_on = self.file[depends_on].attrs["depends_on"]

                except Exception as e:
                        raise Exception("this feature does not validate correctly: "+e)

                        # better have custom exceptions
                return { "dependency_chain" : dependency_chain }
```

proof of concept

Upcoming



- *Shapes!*
  OFF geometry supported for modelling and analysis

- *Nxtransformations!*
  NXmx heavily relies on it – otherwise routine use is still rare

- *Detector Modules!*
  Complicated compound geometries like CSPAD can be defined and refined

- *NXpdb!*
  Inclusion of PDB dictionaries

RCSB PDB
PROTEIN DATA BANK

Outlook

- Modularisation and Versioning Lifecycle of NeXus will be a hot topic for a good while. This is how we keep NeXus modern and suitable for future science.
- Personal Opinion: Application Definitions for raw data might not become much more relevant.
- Defining contents of non-Raw Files, i.e. for processed data, will become more important in many disciplines.
- Better coupling of NeXus to data catalogues would enable more intelligent searching.

Thank you