Online Data Analysis at the European XFEL

Thomas Michelat
Control and Analysis Software Group

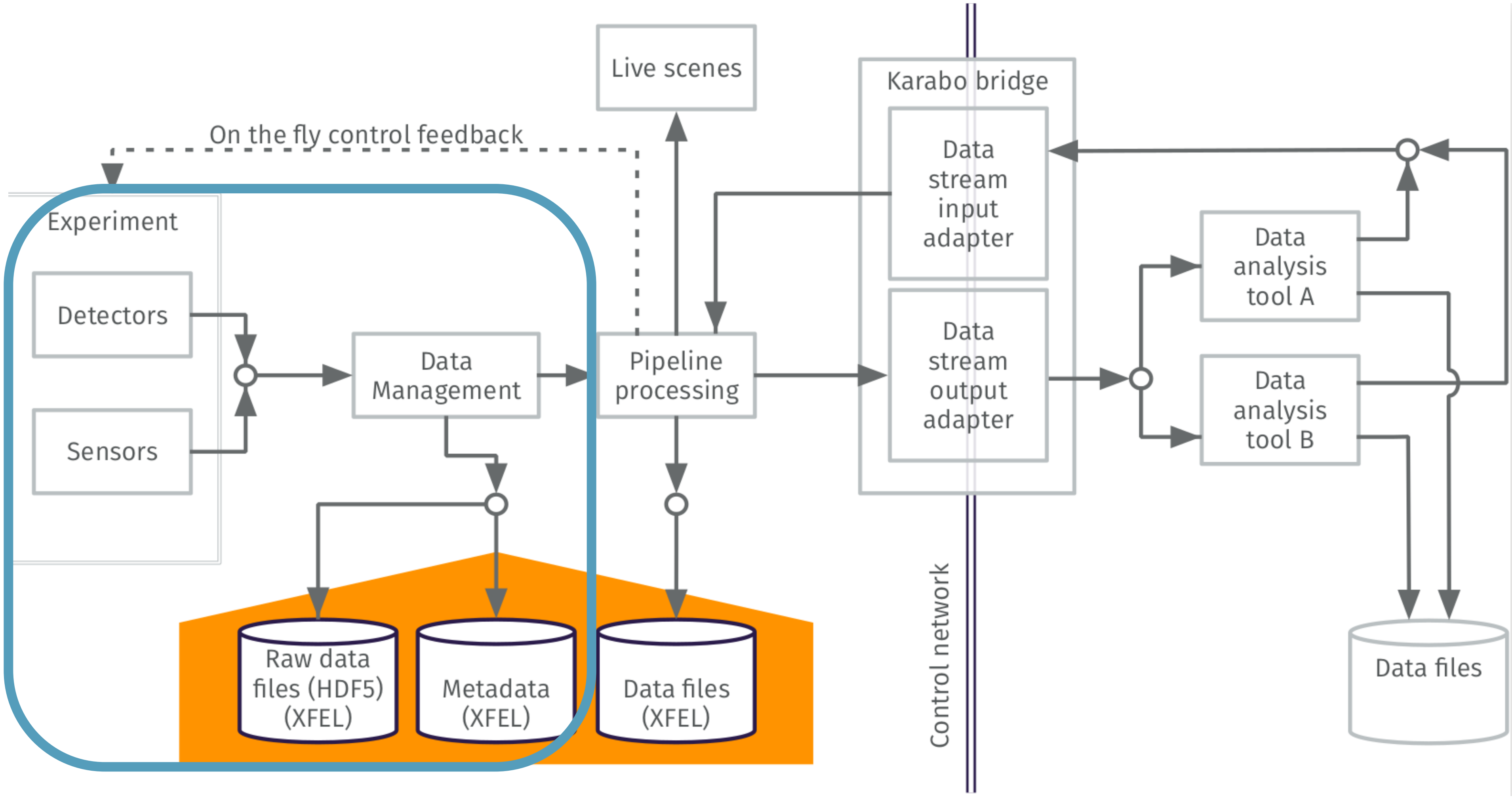
Hamburg, 24 January 2019

Outline

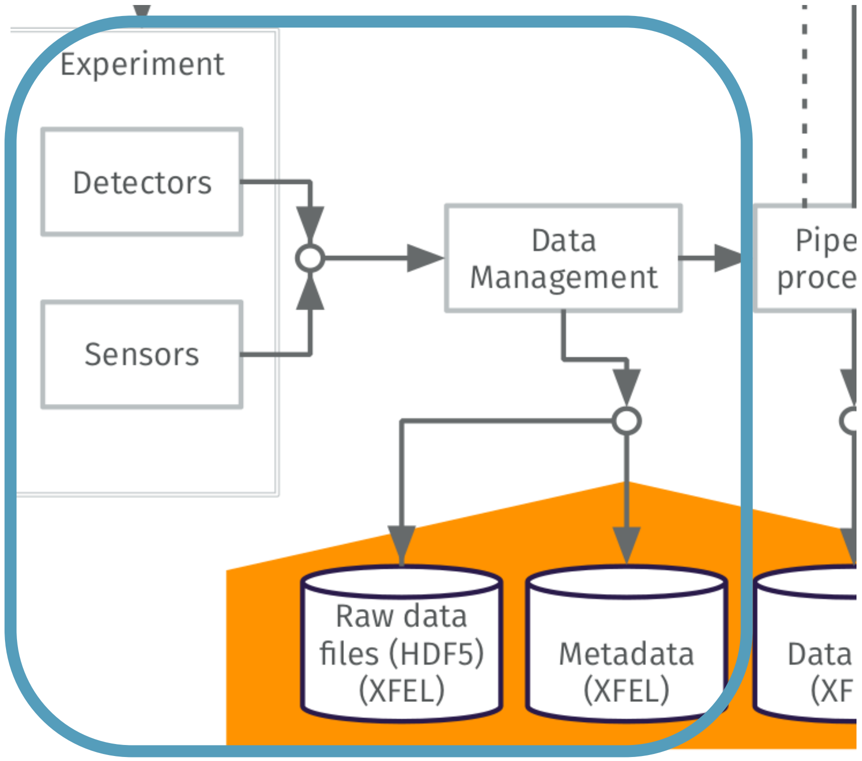
- Online analysis with Karabo
- Karabo Bridge
- Summary

Online analysis with Karabo

Data Acquisition



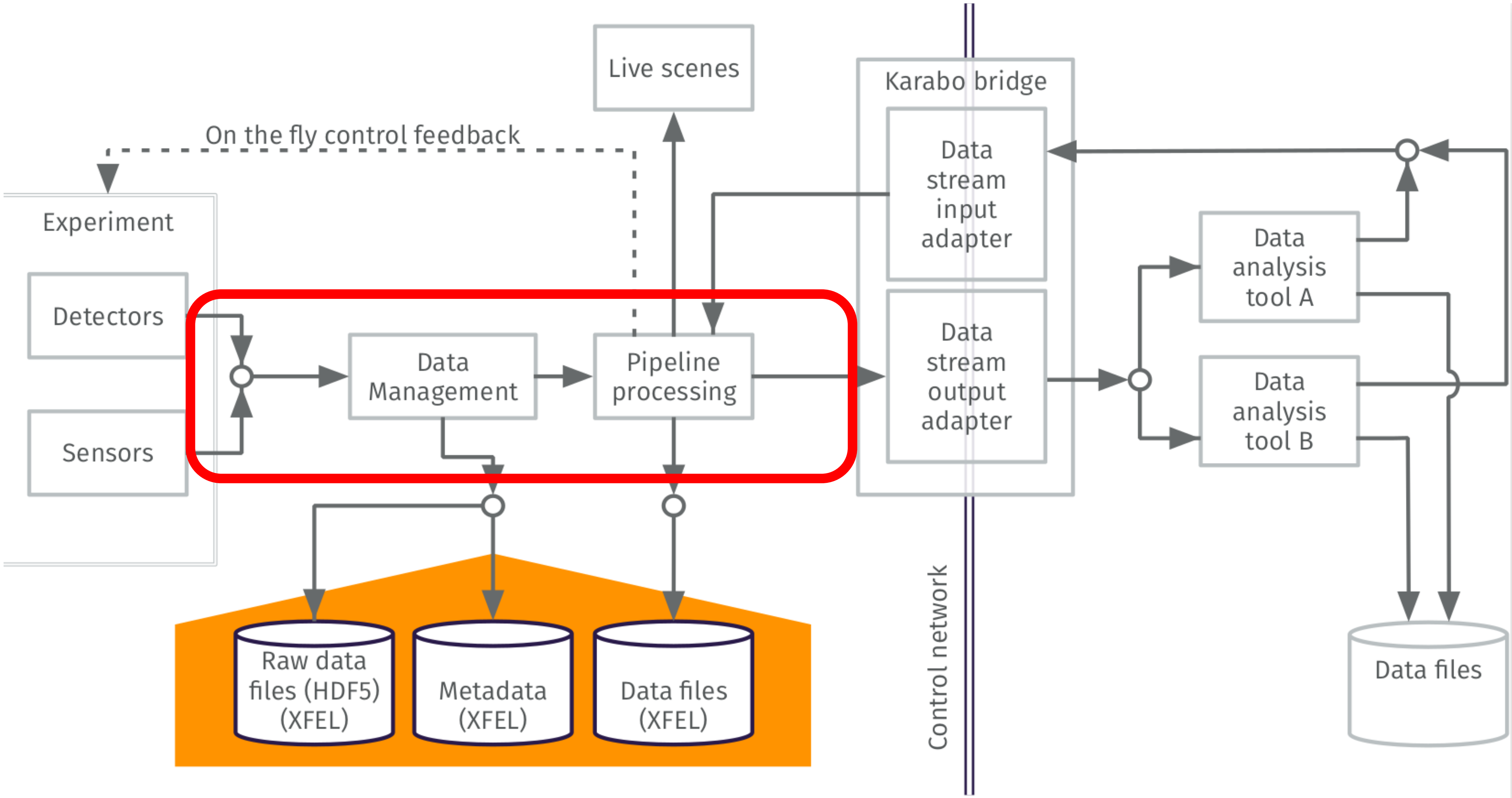
Data Acquisition



- Various data sources
  - Detectors
  - Cameras
  - Sensors
  - Actuators
  - Computing
  - …

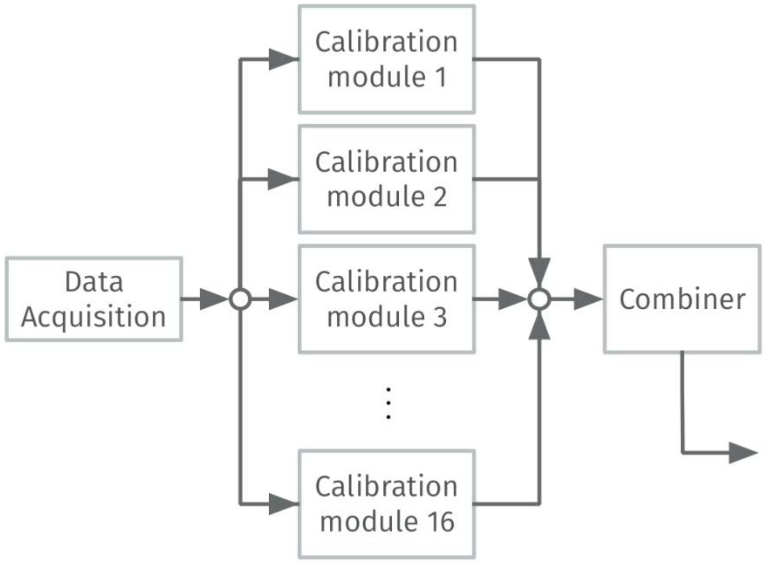
- Interesting sources are gathered in the DAQ system
  - Synchronized by train ID
  - Stored to file (HDF5)
  - Streamed over TCP

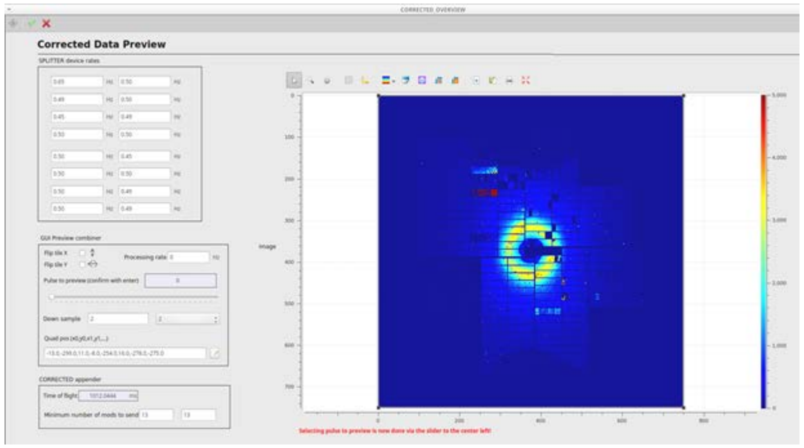
Karabo Data Pipeline



Karabo Data Pipeline

- Integration with the **Karabo** control system
  - Data token pass through pipeline
  - Processing units called **Devices**
  - Devices can be **distributed** over hardware
  - Can be **controlled** and **monitored** through **GUI**
  - Results can be used in other devices

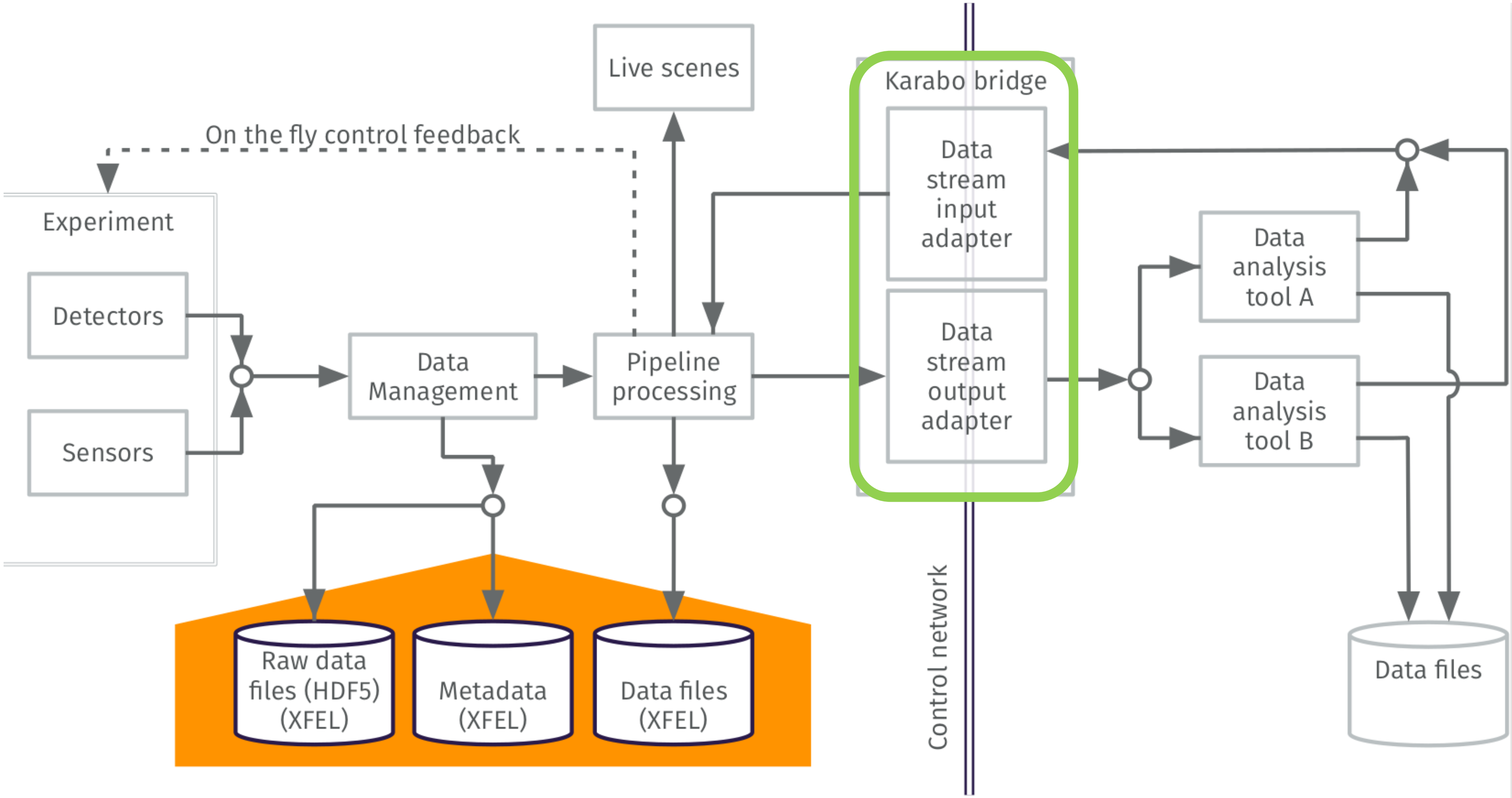




- Pipelines use cases
  - Detector calibration (AGIPD, LPD, …)
  - Image processor
    - ► Beam position tracking
    - ► Correlation with other devices (e.g. XGM)
  - XAS processor
  - Digitizer Processor
    - ► Peak finding

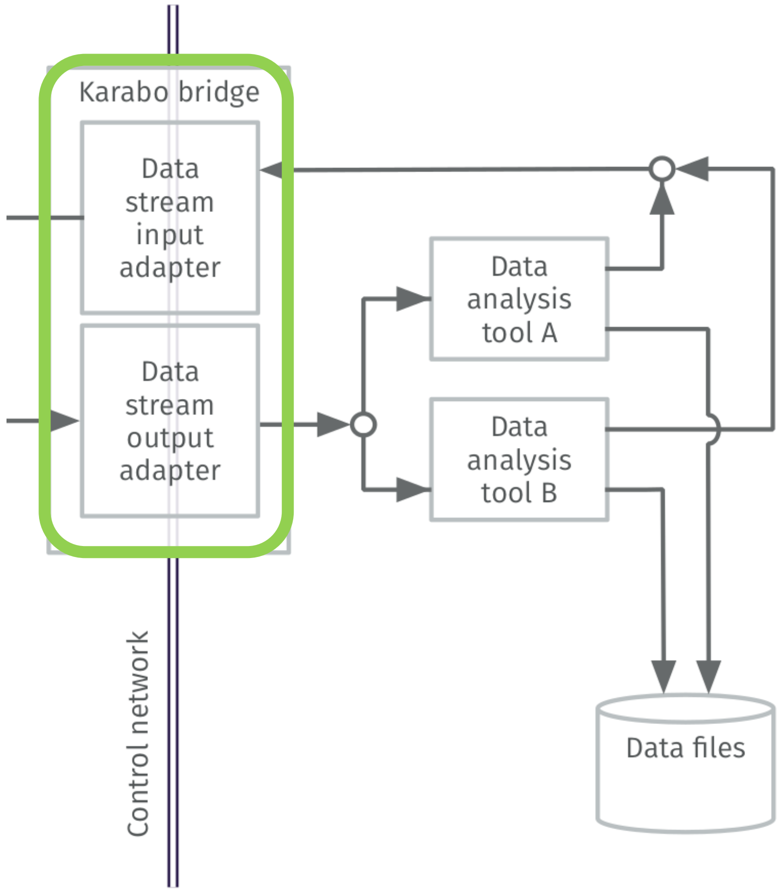
Karabo Bridge

Karabo Bridge



Export Data Pipeline – Karabo Bridge

- We provide an interface to listen to Karabo pipelines
  - **Integrate** existing (complex) user provided tools
  - Quick (dirty) **specific** scripts to use during an experiment
- Karabo Bridge requirements
  - Loosely coupled **Interface** between Karabo and external programs
  - Export data in a **generic** container
  - Using **straightforward** network interface
  - Low latency
- Development in collaboration with CFEL Chapman Group (S. Aplin, A. Barty, M. Kuhn, V. Mariani)



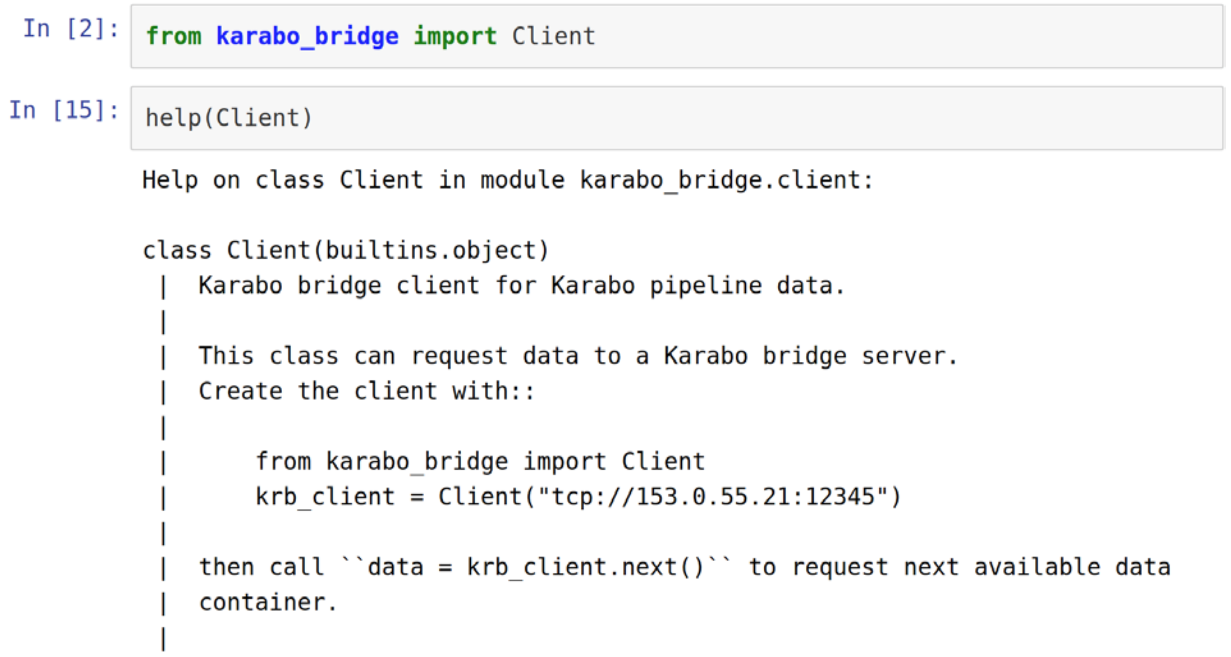
DEMO

Karabo Bridge Client

- Install the client

```
$ pip install karabo-bridge
```

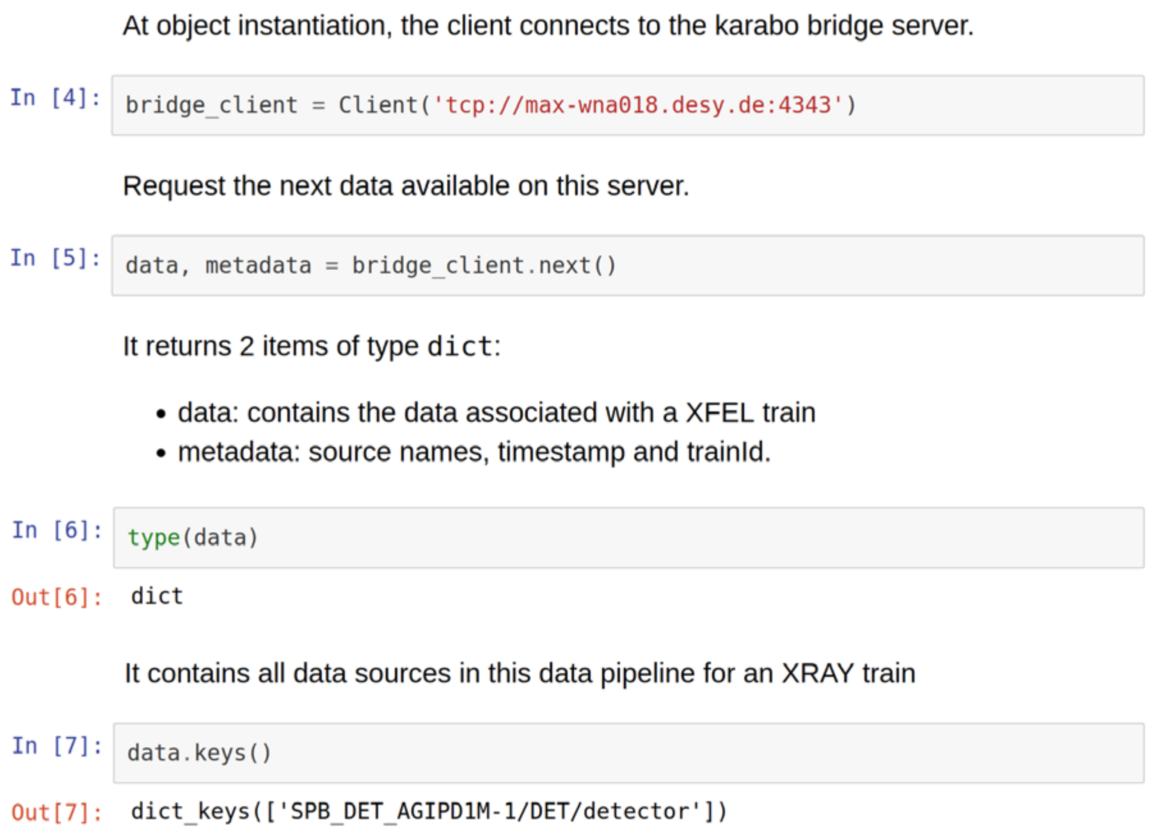
- How to use it



```
In [2]:  from karabo_bridge import Client

In [15]: help(Client)

Help on class Client in module karabo_bridge.client:

class Client(builtins.object)
 |  Karabo bridge client for Karabo pipeline data.
 |
 |  This class can request data to a Karabo bridge server.
 |  Create the client with::
 |
 |      from karabo_bridge import Client
 |      krb_client = Client("tcp://153.0.55.21:12345")
 |
 |  then call ``data = krb_client.next()`` to request next available data
 |  container.
 |
```

Karabo Bridge Client

- Connection to a server
- Request data
- Data is contained in a dictionary
- One entry per data source in the train



At object instantiation, the client connects to the karabo bridge server.

```
In [4]: bridge_client = Client('tcp://max-wna018.desy.de:4343')
```

Request the next data available on this server.

```
In [5]: data, metadata = bridge_client.next()
```

It returns 2 items of type `dict`:

- data: contains the data associated with a XFEL train
- metadata: source names, timestamp and trainId.

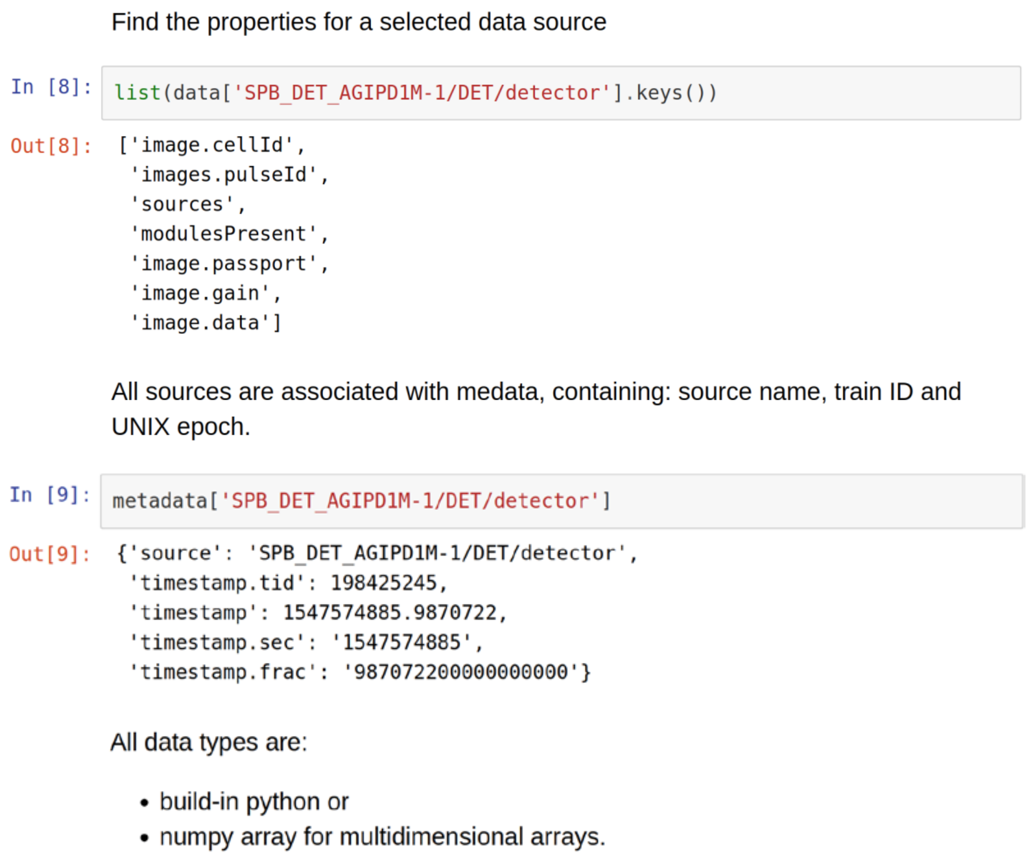
```
In [6]: type(data)

Out[6]: dict
```

It contains all data sources in this data pipeline for an XRAY train

```
In [7]: data.keys()

Out[7]: dict_keys(['SPB_DET_AGIPD1M-1/DET/detector'])
```

Karabo Bridge Client

- Data sources are dictionary
  - contains device parameters
- Associated metadata
- All data are python built-in types
- Big array are Numpy array



Find the properties for a selected data source

```
In [8]: list(data['SPB_DET_AGIPD1M-1/DET/detector'].keys())

Out[8]: ['image.cellId',
         'images.pulseId',
         'sources',
         'modulesPresent',
         'image.passport',
         'image.gain',
         'image.data']
```

All sources are associated with medata, containing: source name, train ID and UNIX epoch.

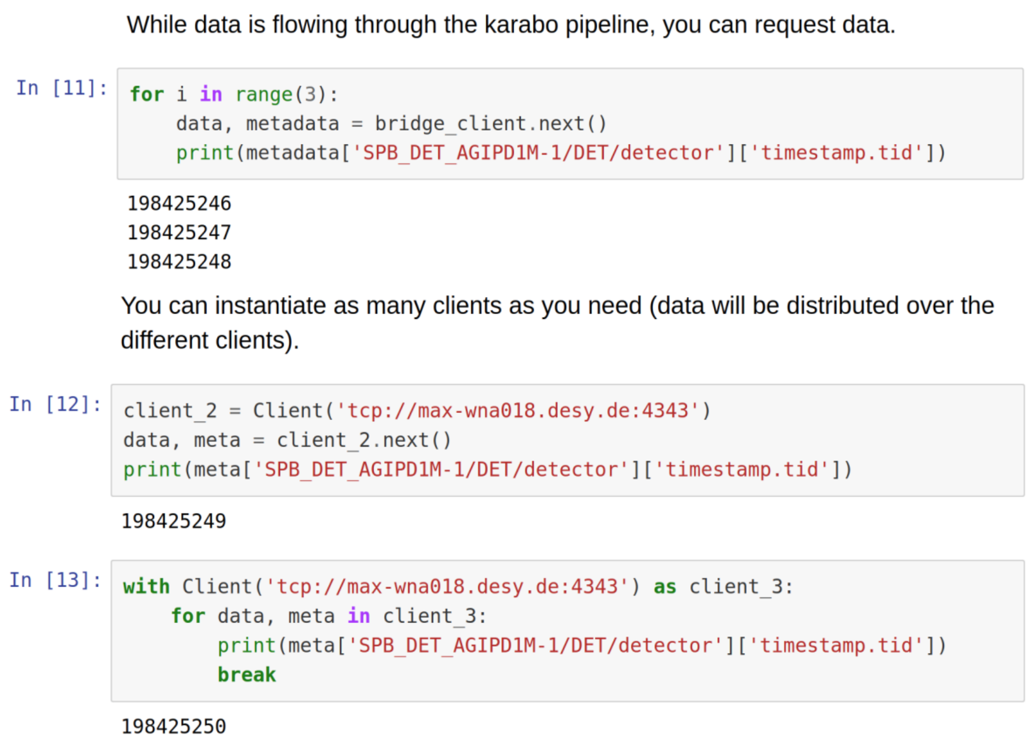
```
In [9]: metadata['SPB_DET_AGIPD1M-1/DET/detector']

Out[9]: {'source': 'SPB_DET_AGIPD1M-1/DET/detector',
         'timestamp.tid': 198425245,
         'timestamp': 1547574885.9870722,
         'timestamp.sec': '1547574885',
         'timestamp.frac': '987072200000000000'}
```

All data types are:

- build-in python or
- numpy array for multidimensional arrays.

Karabo Bridge Client

- Requesting data will return the latest available train in the Pipeline
- Data is dispatched among all clients



While data is flowing through the karabo pipeline, you can request data.

```
In [11]: for i in range(3):
             data, metadata = bridge_client.next()
             print(metadata['SPB_DET_AGIPD1M-1/DET/detector']['timestamp.tid'])
```

```
198425246
198425247
198425248
```

You can instantiate as many clients as you need (data will be distributed over the different clients).

```
In [12]: client_2 = Client('tcp://max-wna018.desy.de:4343')
         data, meta = client_2.next()
         print(meta['SPB_DET_AGIPD1M-1/DET/detector']['timestamp.tid'])
```

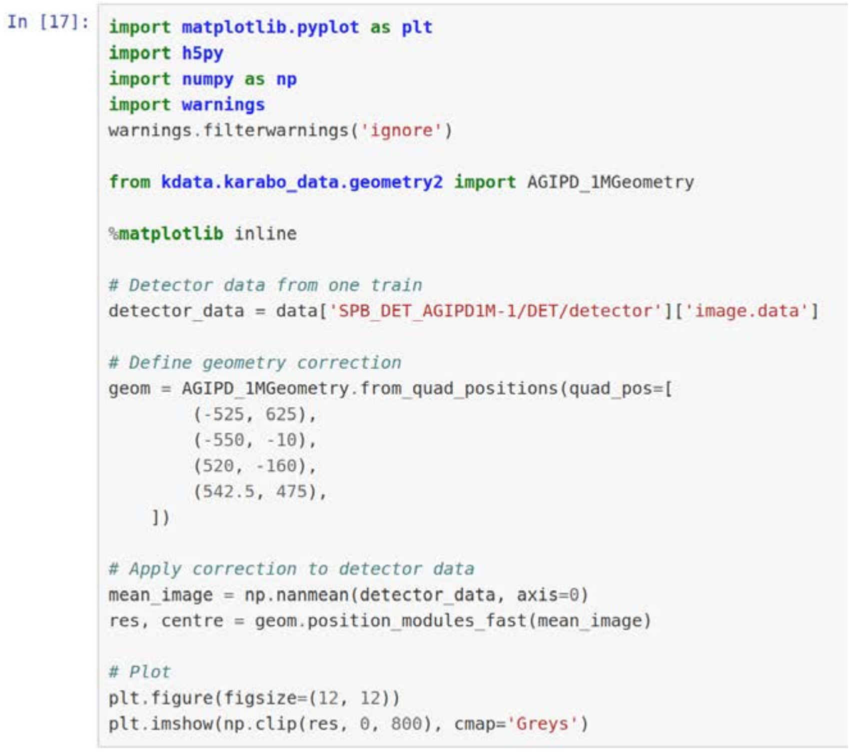
```
198425249
```

```
In [13]: with Client('tcp://max-wna018.desy.de:4343') as client_3:
             for data, meta in client_3:
                 print(meta['SPB_DET_AGIPD1M-1/DET/detector']['timestamp.tid'])
                 break
```

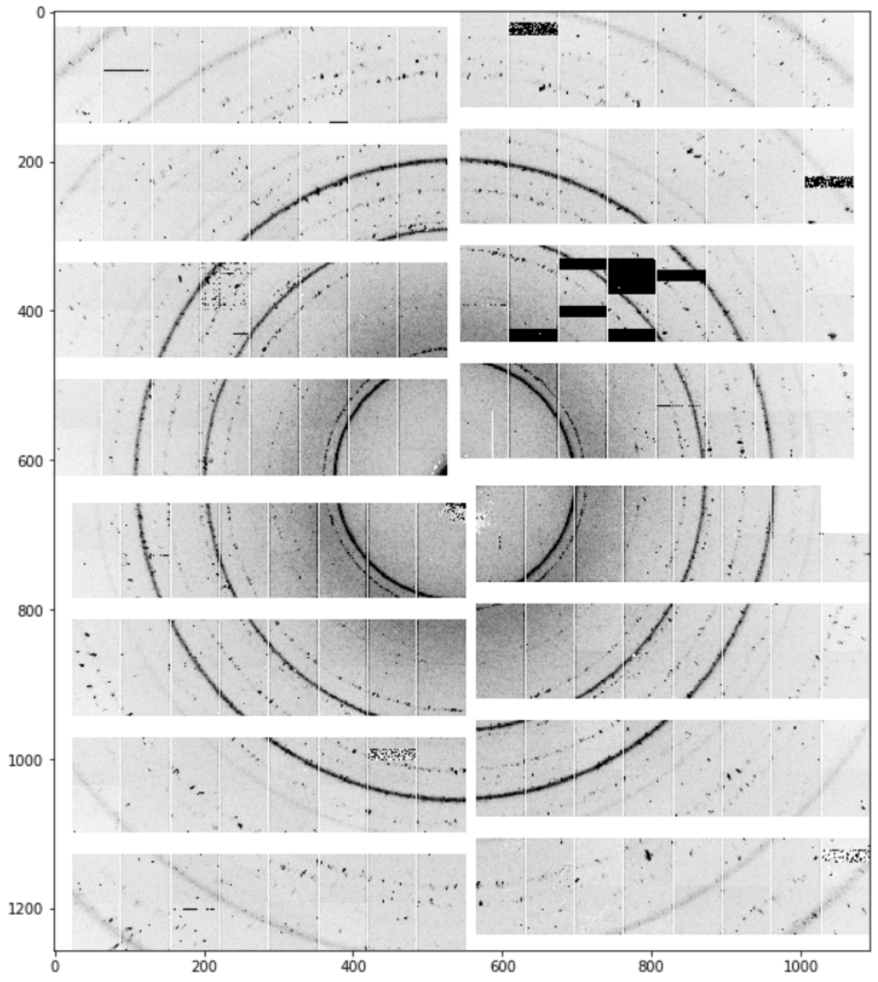
```
198425250
```

Karabo Bridge Client

- Apply detector geometry with karabo_data



```
In [17]: import matplotlib.pyplot as plt
         import h5py
         import numpy as np
         import warnings
         warnings.filterwarnings('ignore')

         from kdata.karabo_data.geometry2 import AGIPD_1MGeometry

         %matplotlib inline

         # Detector data from one train
         detector_data = data['SPB_DET_AGIPD1M-1/DET/detector']['image.data']

         # Define geometry correction
         geom = AGIPD_1MGeometry.from_quad_positions(quad_pos=[
                 (-525, 625),
                 (-550, -10),
                 (520, -160),
                 (542.5, 475),
             ])

         # Apply correction to detector data
         mean_image = np.nanmean(detector_data, axis=0)
         res, centre = geom.position_modules_fast(mean_image)

         # Plot
         plt.figure(figsize=(12, 12))
         plt.imshow(np.clip(res, 0, 800), cmap='Greys')
```



Karabo Bridge Client – Command line tools

- Useful CLI tools
  - Server simulation
    - Simulate a karabo bridge server with simulated data

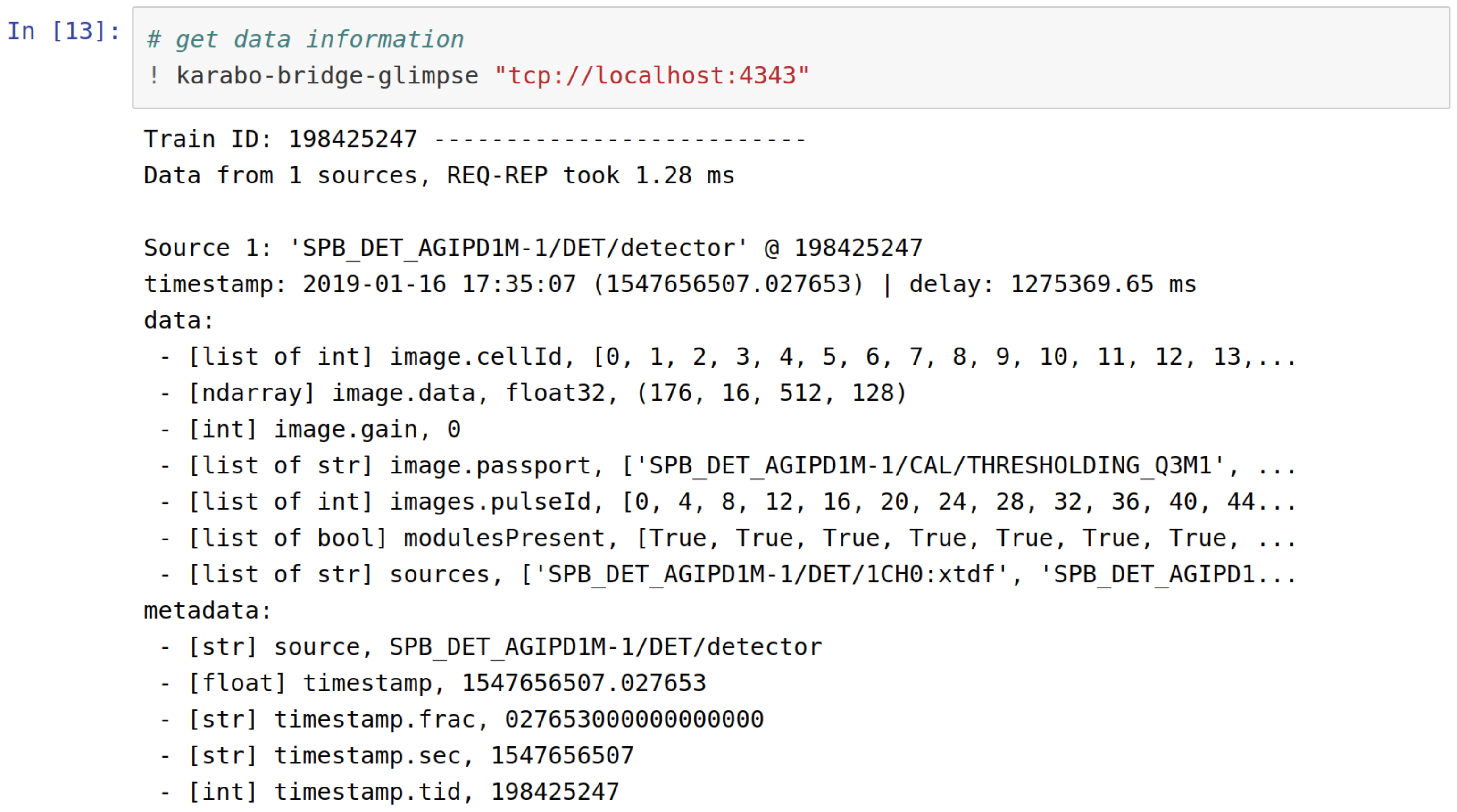
      ```
      $ karabo-bridge-server-sim
      ```
  - File streaming
    - Simulate a karabo bridge server from data reccorded at the EuXFEL

      ```
      $ karabo-bridge-serve-files
      ```
  - Data exploration
    - get information on the data sent by the Karabo Bridge

      ```
      $ karabo-bridge-glimpse
      ```
  - Data monitor
    - monitor the data sent by the Karabo Bridge

      ```
      $ karabo-bridge-monitor
      ```

Karabo Bridge Client – Command line tools



```
In [13]: # get data information
         ! karabo-bridge-glimpse "tcp://localhost:4343"
```

```
Train ID: 198425247 --------------------------
Data from 1 sources, REQ-REP took 1.28 ms

Source 1: 'SPB_DET_AGIPD1M-1/DET/detector' @ 198425247
timestamp: 2019-01-16 17:35:07 (1547656507.027653) | delay: 1275369.65 ms
data:
 - [list of int] image.cellId, [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13,...
 - [ndarray] image.data, float32, (176, 16, 512, 128)
 - [int] image.gain, 0
 - [list of str] image.passport, ['SPB_DET_AGIPD1M-1/CAL/THRESHOLDING_Q3M1', ...
 - [list of int] images.pulseId, [0, 4, 8, 12, 16, 20, 24, 28, 32, 36, 40, 44...
 - [list of bool] modulesPresent, [True, True, True, True, True, True, True, ...
 - [list of str] sources, ['SPB_DET_AGIPD1M-1/DET/1CH0:xtdf', 'SPB_DET_AGIPD1...
metadata:
 - [str] source, SPB_DET_AGIPD1M-1/DET/detector
 - [float] timestamp, 1547656507.027653
 - [str] timestamp.frac, 027653000000000000
 - [str] timestamp.sec, 1547656507
 - [int] timestamp.tid, 198425247
```

Karabo Bridge Client – C++

Installation

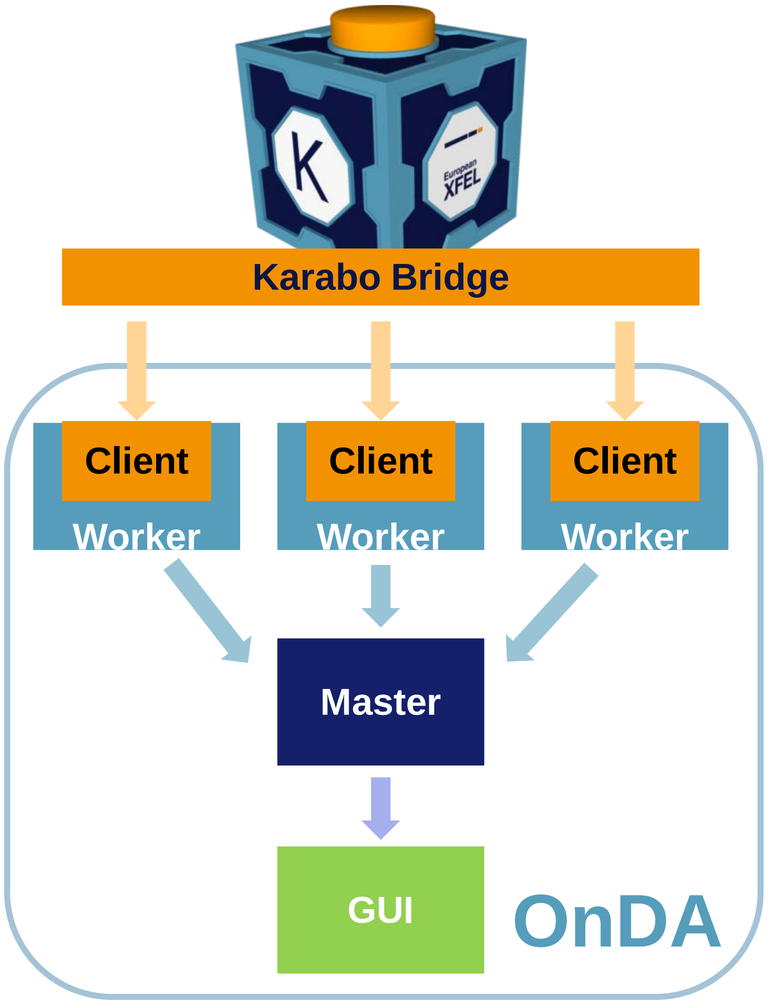
```
$ git clone https://github.com/European-XFEL/karabo-bridge-cpp.git
$ cd karabo-bridge-cpp
$ ./autogen.sh install /YOUR/TARGET/FOLDER
```

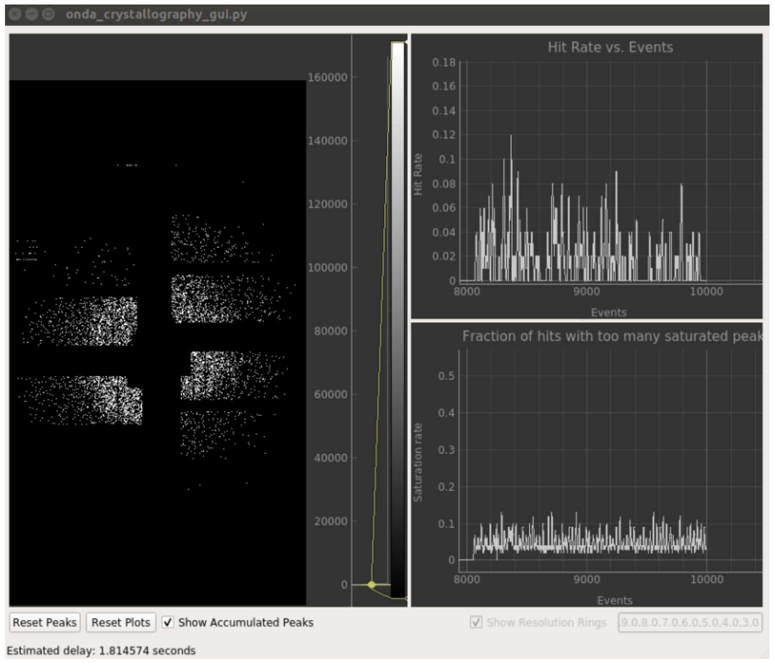
Usage

```
#include "kb_client.hpp"

karabo_bridge::Client client;
client.connect("tcp://localhost:1234")

auto payload = client.next()
for (auto it = payload.begin(); it != payload.end(); ++it) {
    if (it->first == "SPB_DET_AGIPD1M-1/DET/detector") {
        karabo_bridge::kb_data data(it->second);
        auto images = data.array["image.data"].as<std::vector<uint32_t>>();
    }
}
```

Source, documentation, examples: https://github.com/European-XFEL/karabo-bridge-cpp/

Karabo Bridge – in use

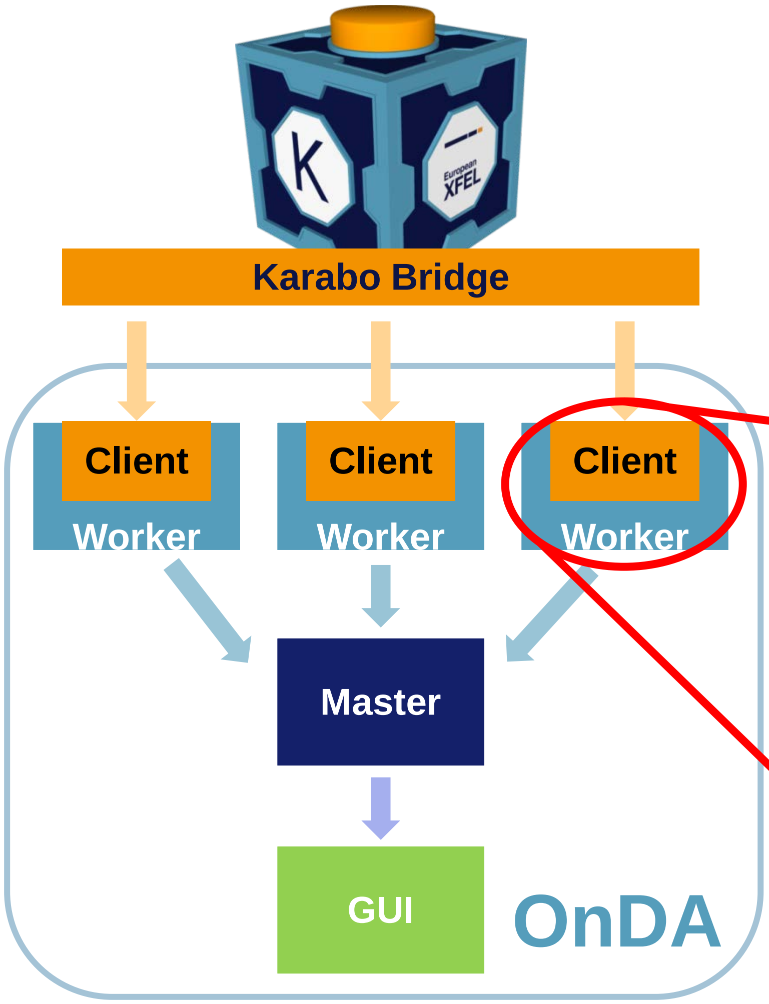


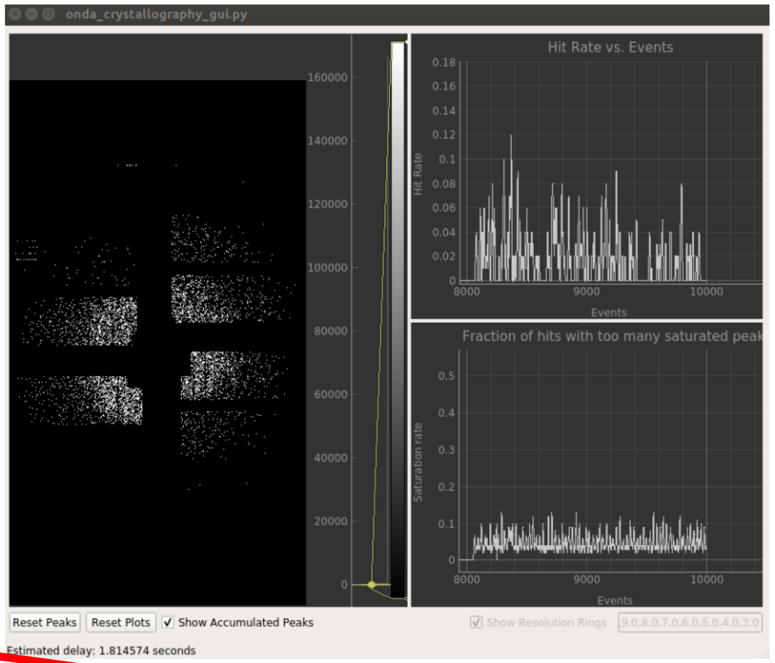


Estimated delay: 1.814574 seconds

- 🟧 Tool integration example
  - 🟦 OnDA – Serial Crystallography

Karabo Bridge – in use



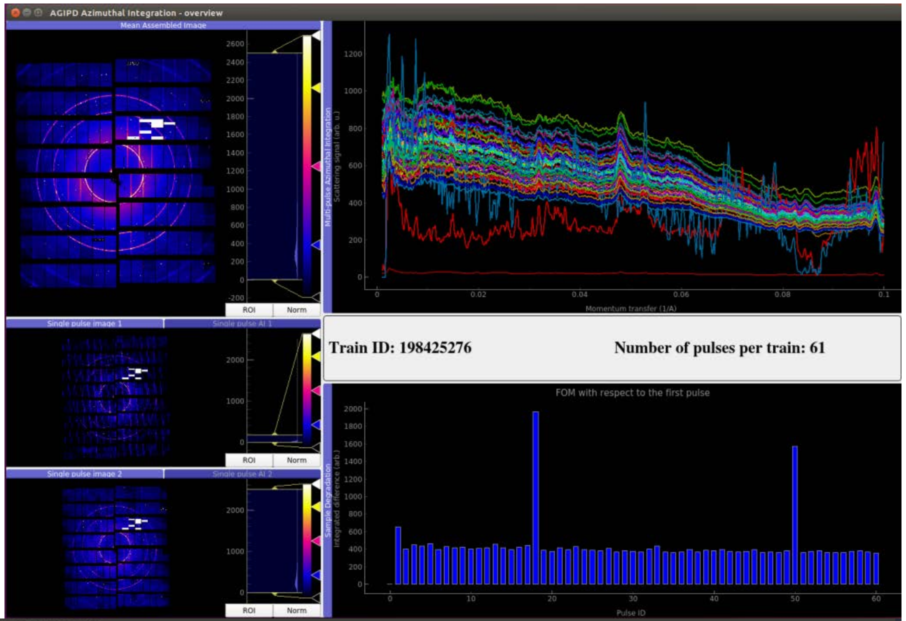


Estimated delay: 1.814574 seconds

```
def __init__(self, interface, **kwargs):
    ...
    self.client = Client(inteface)
    ...

...

def run(self):
    while True:
        data, metadata = self.client.next()

        #process data
        ...
```

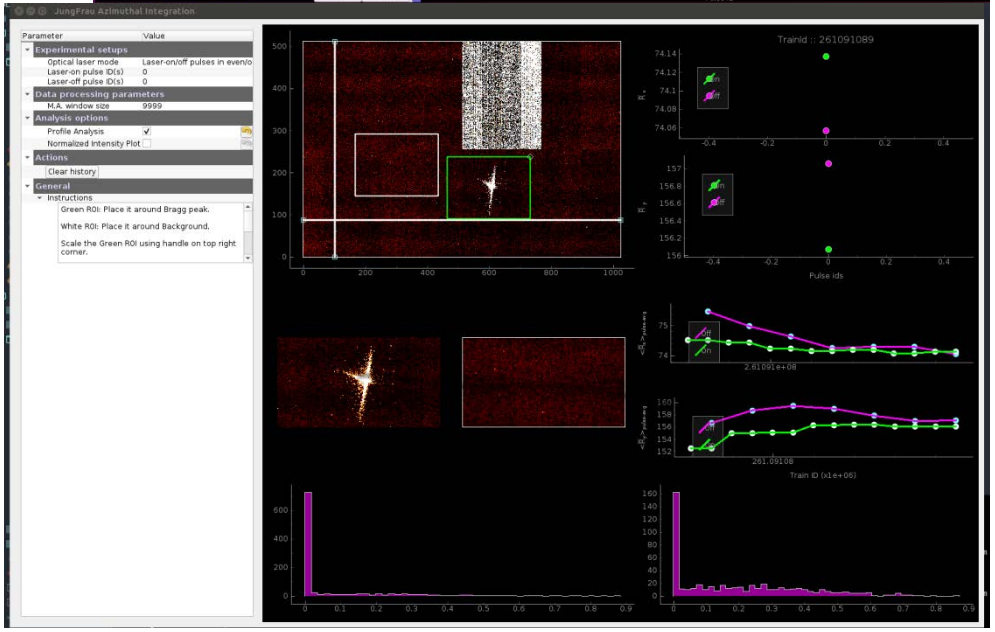
Karabo Bridge – In use

- Crystallography hit finding
  - OnDA
  - Hummingbird
  - CASS (C++ client)
- Fast azimuthal integration
  - Karabo FAI
- Bragg spot analysis
  - KaraboFAI
- X-Ray absorption spectroscopy
- Laser time drifting


AGIPD Azimuthal Integration - overview
Mean Assembled Image
Single pulse image 1
Single pulse AI 1
Single pulse image 2
Single pulse AI 2
ROI
Norm
Multi-pulse Azimuthal Integration
Scattering signal (arb. u.)
Momentum transfer (1/A)
FOM with respect to the first pulse
Sample Degradation
Integrated difference (arb.)
Pulse ID

Train ID: 198425276

Number of pulses per train: 61



Parameter	Value
Experimental setups	
Optical laser mode	Laser-on/off pulses in even/o
Laser-on pulse ID(s)	0
Laser-off pulse ID(s)	0
Data processing parameters	
M.A. window size	9999
Analysis options	
Profile Analysis	✓
Normalized Intensity Plot	☐
Actions	
Clear history	
General	
Instructions	

Green ROI: Place it around Bragg peak.

White ROI: Place it around Background.

Scale the Green ROI using handle on top right corner.

Summary

- Outlined basics for **near real time** data analysis
  - **Karabo** Data **Pipelines**
  - **Karabo Bridge**
    - Network **interface** to access scientific data during experiment in near real time
    - Easy set-up to **export** any **data** pipeline from Karabo
    - Python client: https://github.com/European-XFEL/karabo-bridge-py
    - C++ client: https://github.com/European-XFEL/karabo-bridge-cpp
- Performances for calibrated detector data (AGIPD / LPD)
  - Latency below 3 seconds
  - Stable throughput of 128 image per seconds
- Keen to work with users
  - → Get in touch
- Literature
  - H. Fangohr et al, Data Analysis support in Karabo at European XFEL, ICALEPSC 2017, online: http://icalepcs2017.vrws.de/papers/tucpa01.pdf
  - S. Hauf et al., “The Karabo Distributed Control System”, submitted to Journal of Synchrotron Radiation

Documentation

- Open source projects
  - https://github.com/European-XFEL
- User documentation for Data Analysis at the European XFEL
  - https://in.xfel.eu/readthedocs/docs/data-analysis-user-documentation/en/latest
- Karabo-data documentation
  - https://karabo-data.readthedocs.io/en/latest
- Karabo bridge
  - Python client: https://github.com/European-XFEL/karabo-bridge-py
  - C++ client: https://github.com/European-XFEL/karabo-bridge-cpp
  - documentation: https://in.xfel.eu/readthedocs/docs/data-analysis-user-documentation/en/latest/online.html#data-stream-to-user-tools