Online Data Analysis at European XFEL

Hans Fangohr
Control and Analysis Software Group
Senior Data Analysis Scientist

DESY, 25 January 2018

Outline

- Introduction & European XFEL status
- Overview online analysis
- “Karabo Bridge” (0MQ)
- Summary

Introduction

Hans Fangohr

- Diplom in Physics (1999), Hamburg
- PhD in High Performance Computing & Computer simulation (2002) in Computer Science, Southampton (UK)
- Lecturer (2002), Senior Lecturer (2006) in Computational Methods, Engineering, Southampton
- Professor of Computational Modelling (2010), Southampton
  - Head of Computational Modelling Group (2010-2017)
- Since September 2017 Data Analysis Scientist at XFEL
- Research interests:
  - Data Analysis & Computational Modelling
  - Software for Science
  - Use of software for science

European XFEL

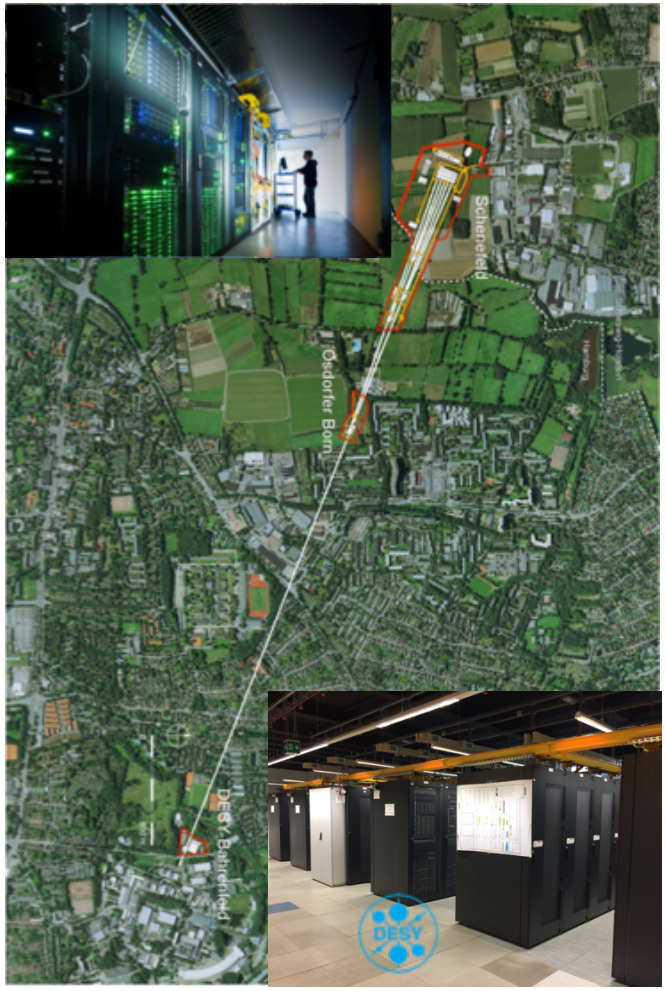
- Official opening 1 September 2017
- 2 of 6 scientific instruments live
- First experiments started 14 Sept 2017
- 12 proposals collected ~450 TB raw data
- Positive feedback

Prof. Dr. Johanna Wanka, Bundesministerin für Bildung und Forschung, visits SPB hutch

Data analysis infrastructure

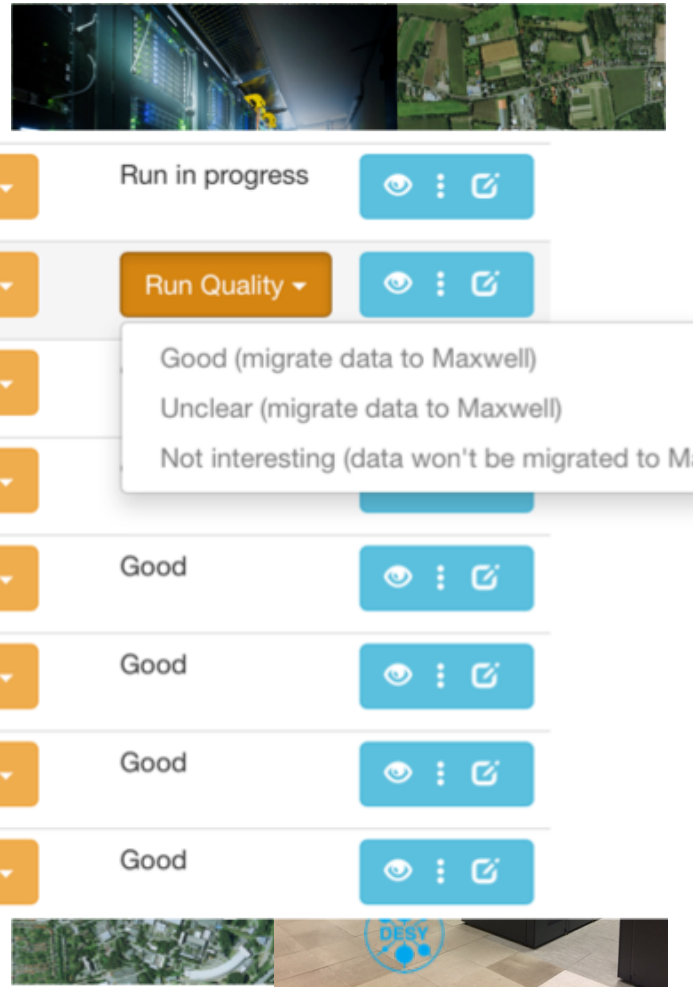
- Hardware: “Online cluster”,
  - 8 nodes x (20 cores, 256GB RAM) dedicated to users
  - Additional nodes for control and XFEL provided calibration and processing

- Hardware: “Offline cluster” = Maxwell cluster (DESY)
  - 80 nodes/3200 cores (Intel Xeon E5-2698v4)
  - ~112 TFlops
  - 512GB RAM each node
  - +20 nodes with other spec
  - 7 GPU nodes available



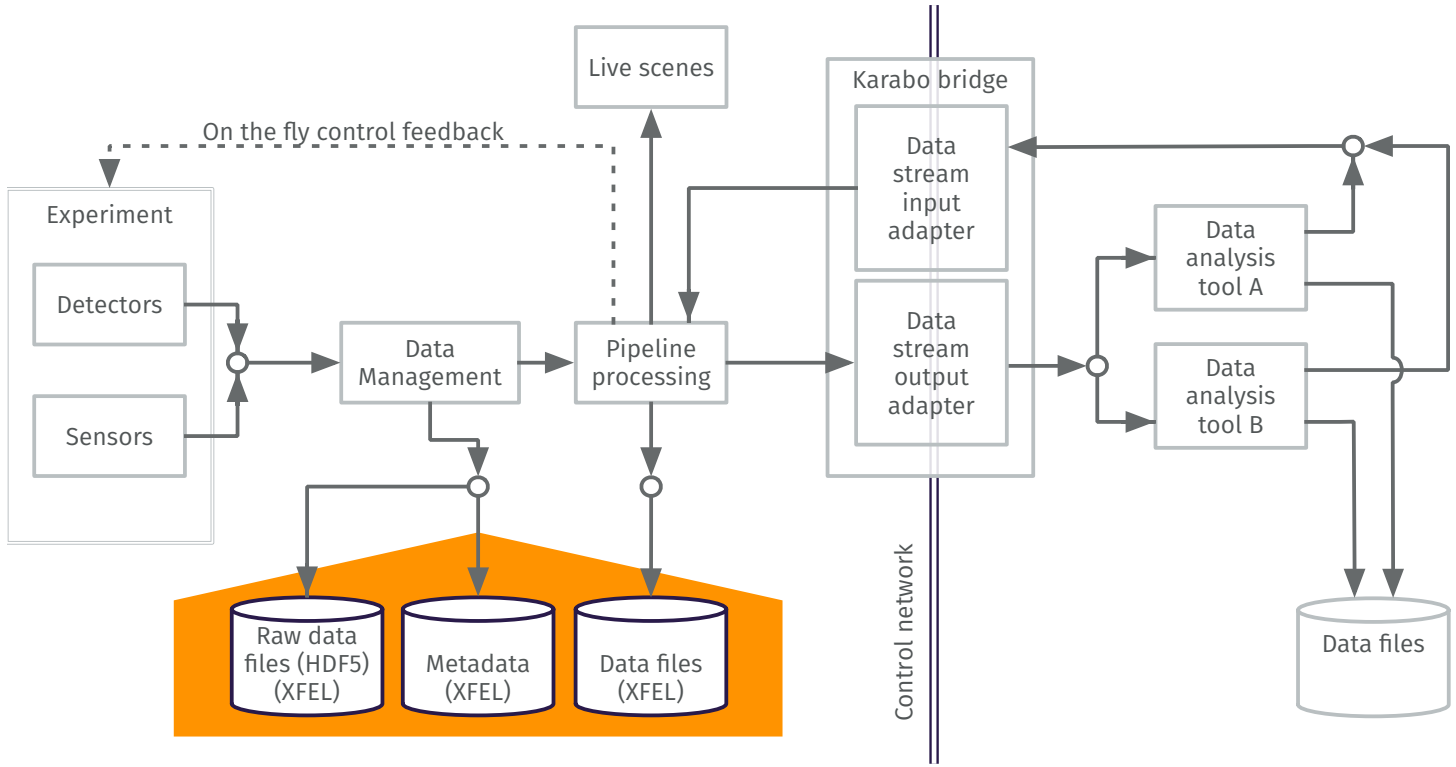
Data management online -> offline

- During measurement (run)
  - Calibrated and raw data available in hutch (GUI, online)
- Data migration after each run
  - After each run, data manager decides on quality of the data: “good”, “unclear”, “not interesting”
  - “good” and “unclear” data transferred to “Offline cluster”
  - Migration triggers computation of calibrated data at online cluster
- After experiment
  - Raw and calibrated data available
  - Analysis on ”Offline cluster” (Maxwell @ DESY)
- No automatic online data reduction



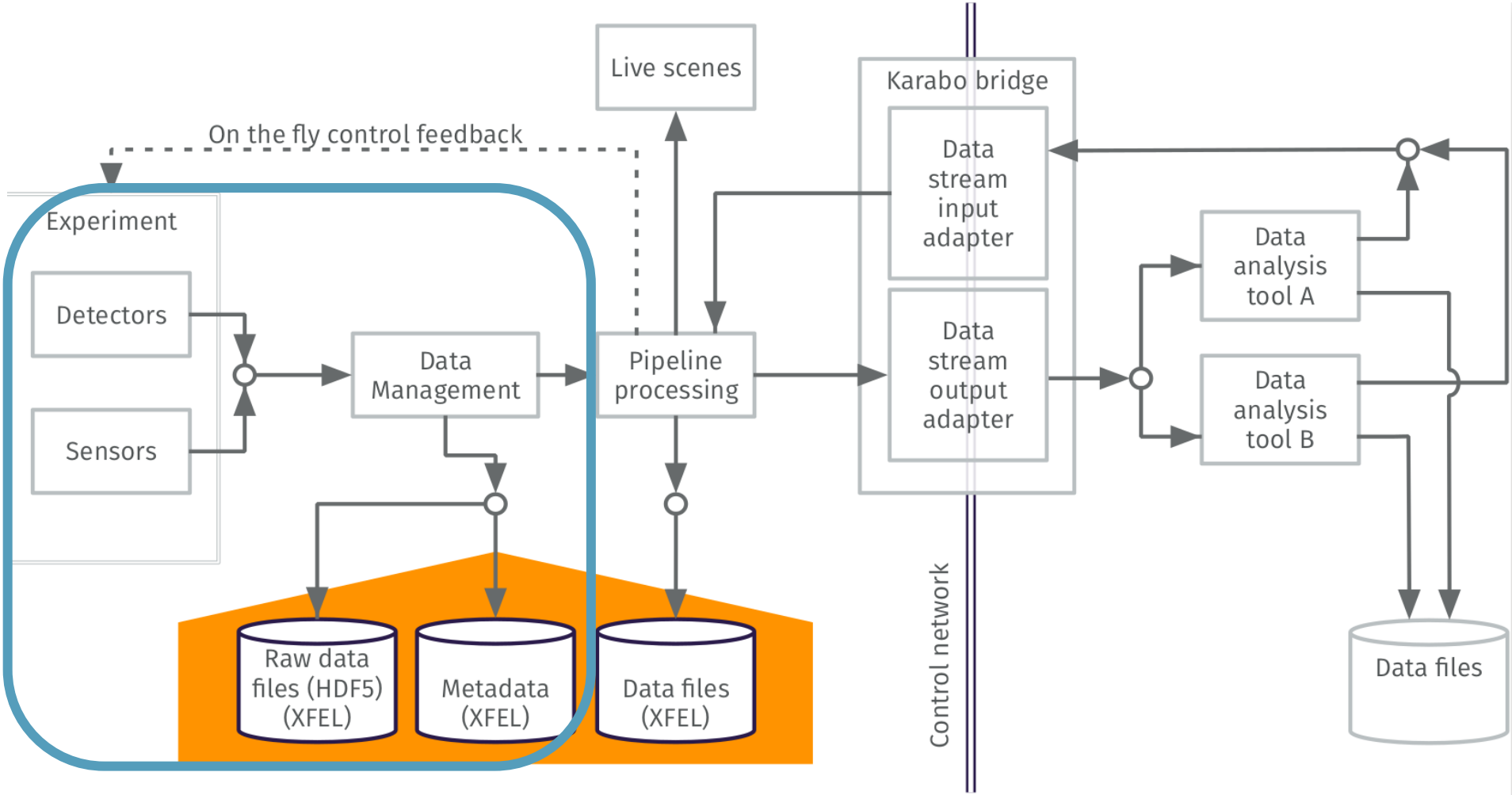
Online Analysis

Online data analysis



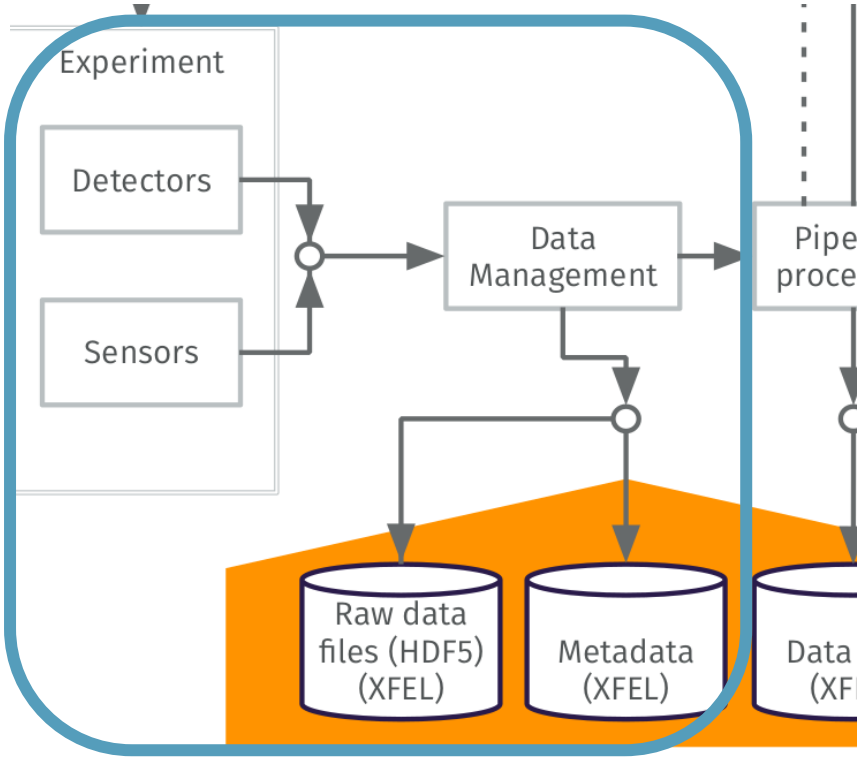
Online data analysis

Data Acquisition

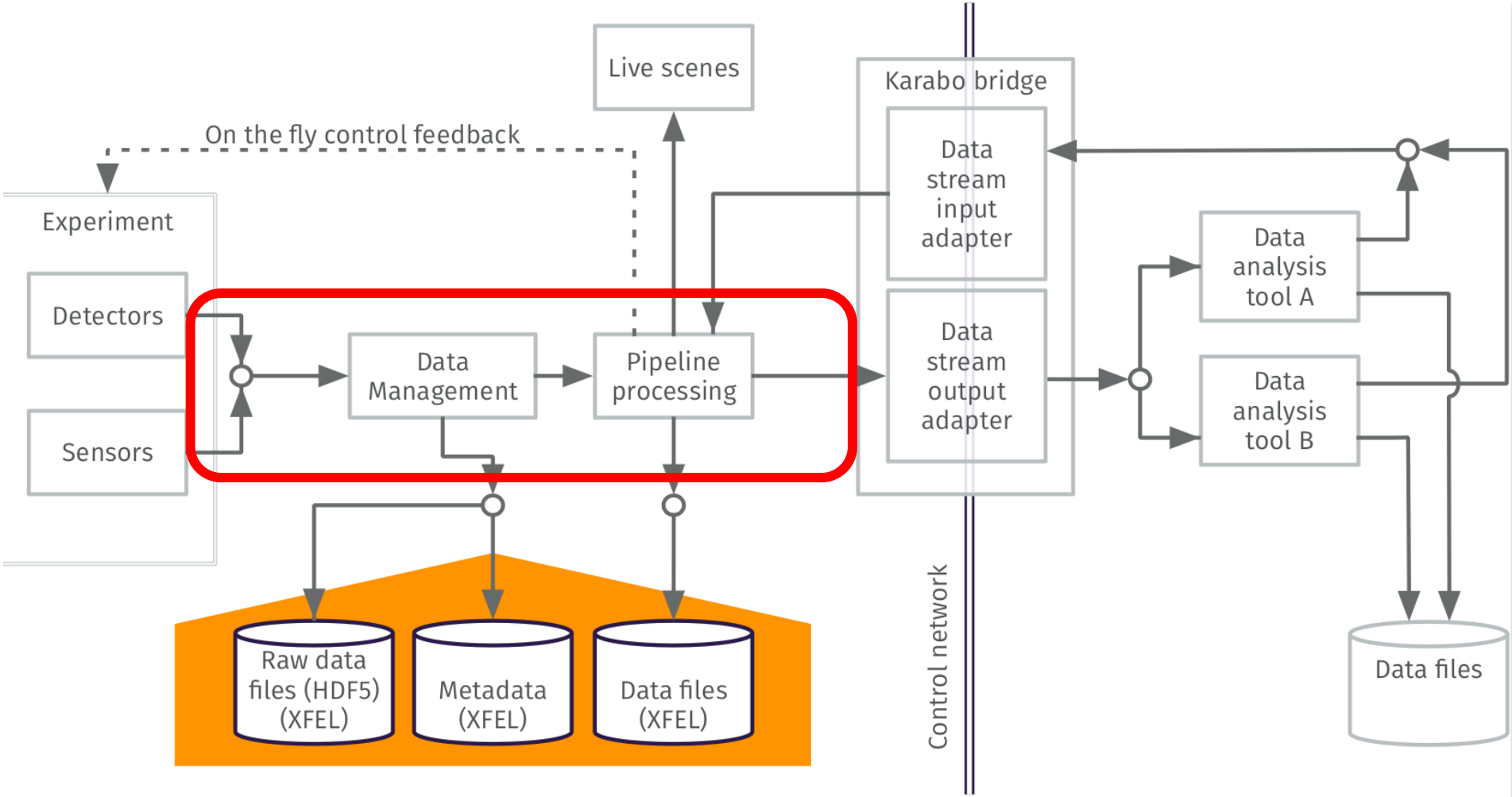


Data Acquisition (DAQ)

- 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

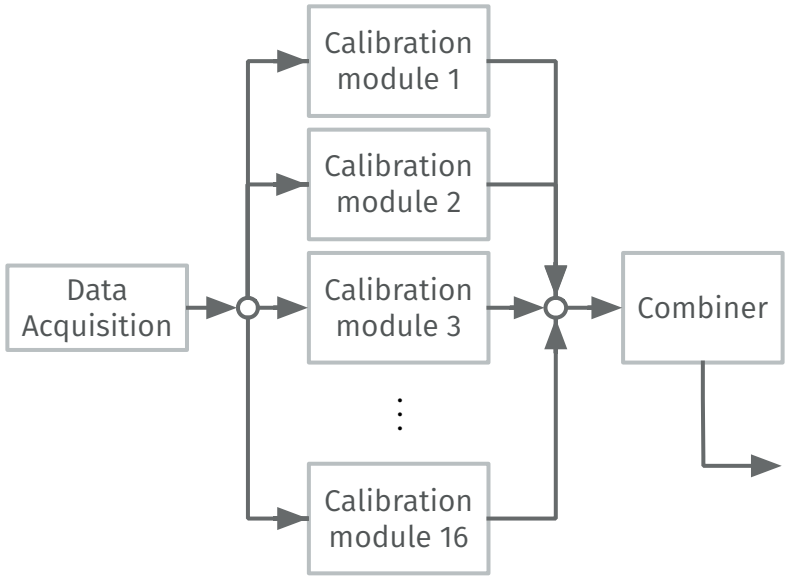


Karabo Data Pipeline



Karabo processing pipeline example

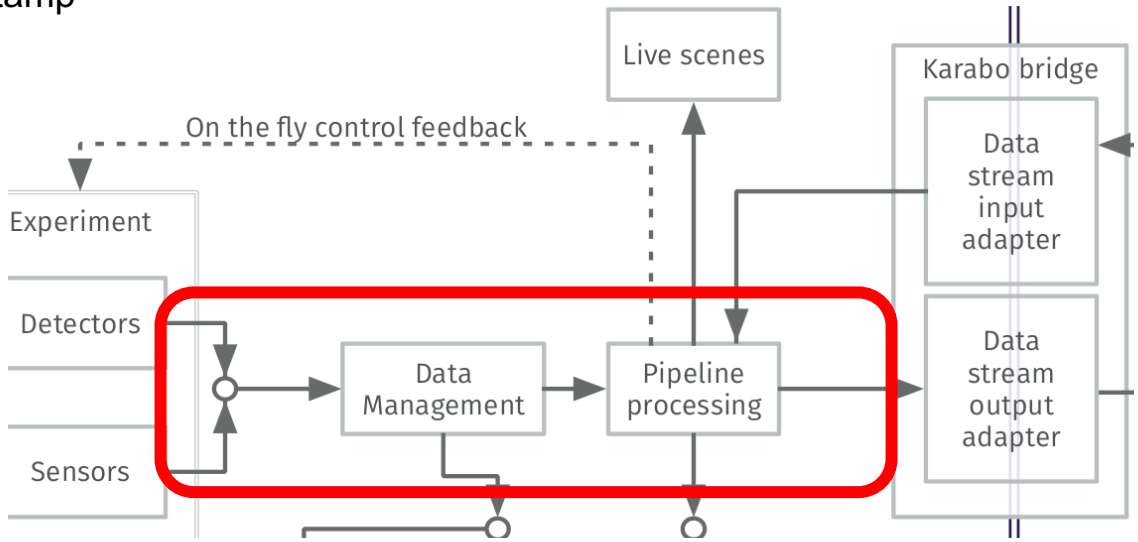
- Karabo [1] is framework for control and data
  - Data tokens pass through pipeline
  - Processing units called “devices”
  - Devices can be distributed over hardware
  - Simplified example in figure: calibration for detector modules carried out in parallel



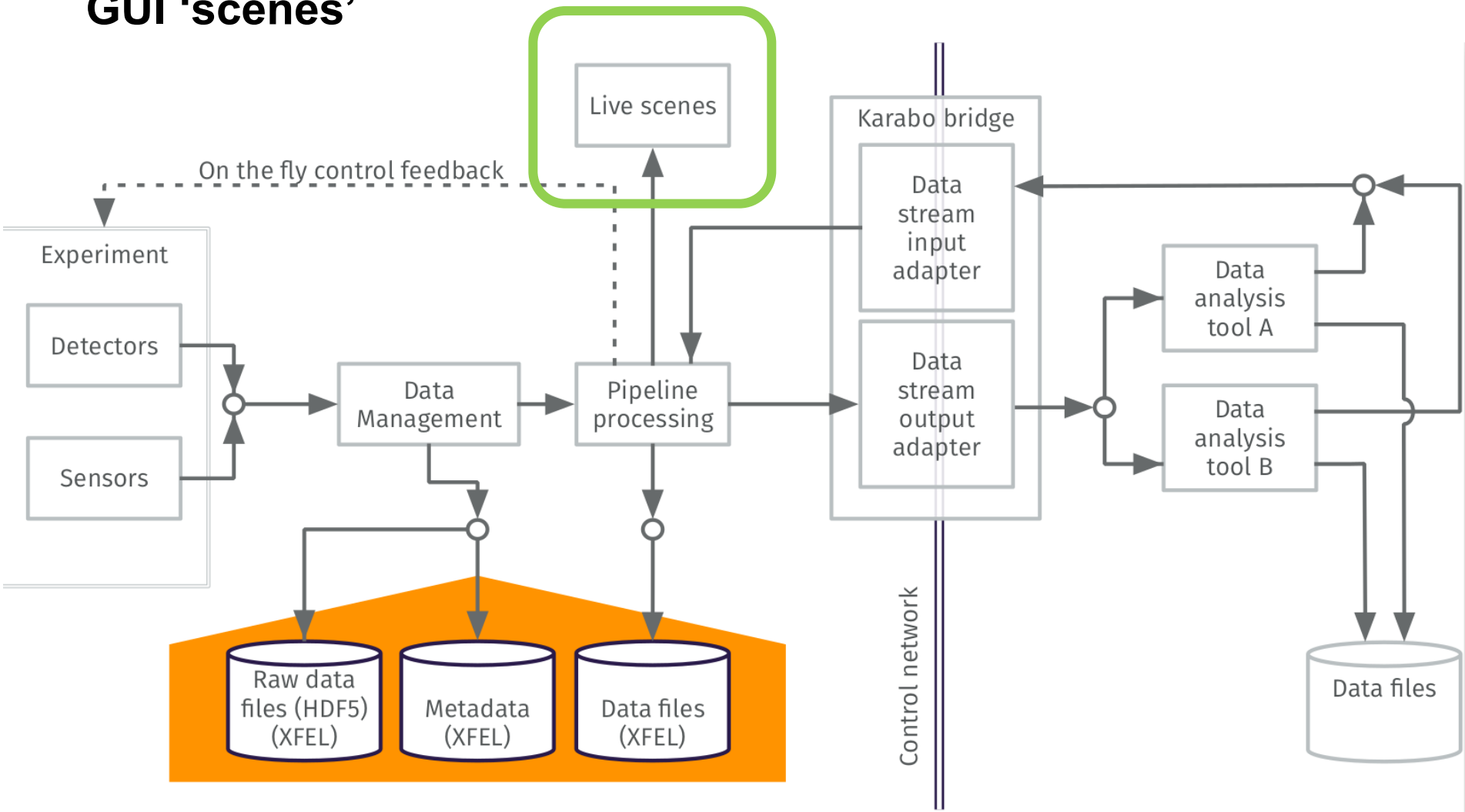
[1] B. Heisen et al: “Karabo: An integrated software framework combining control, data management, and scientific computing tasks,” in14th ICALEPCS2013. San Francisco, CA, 2013.

Karabo Data Pipeline

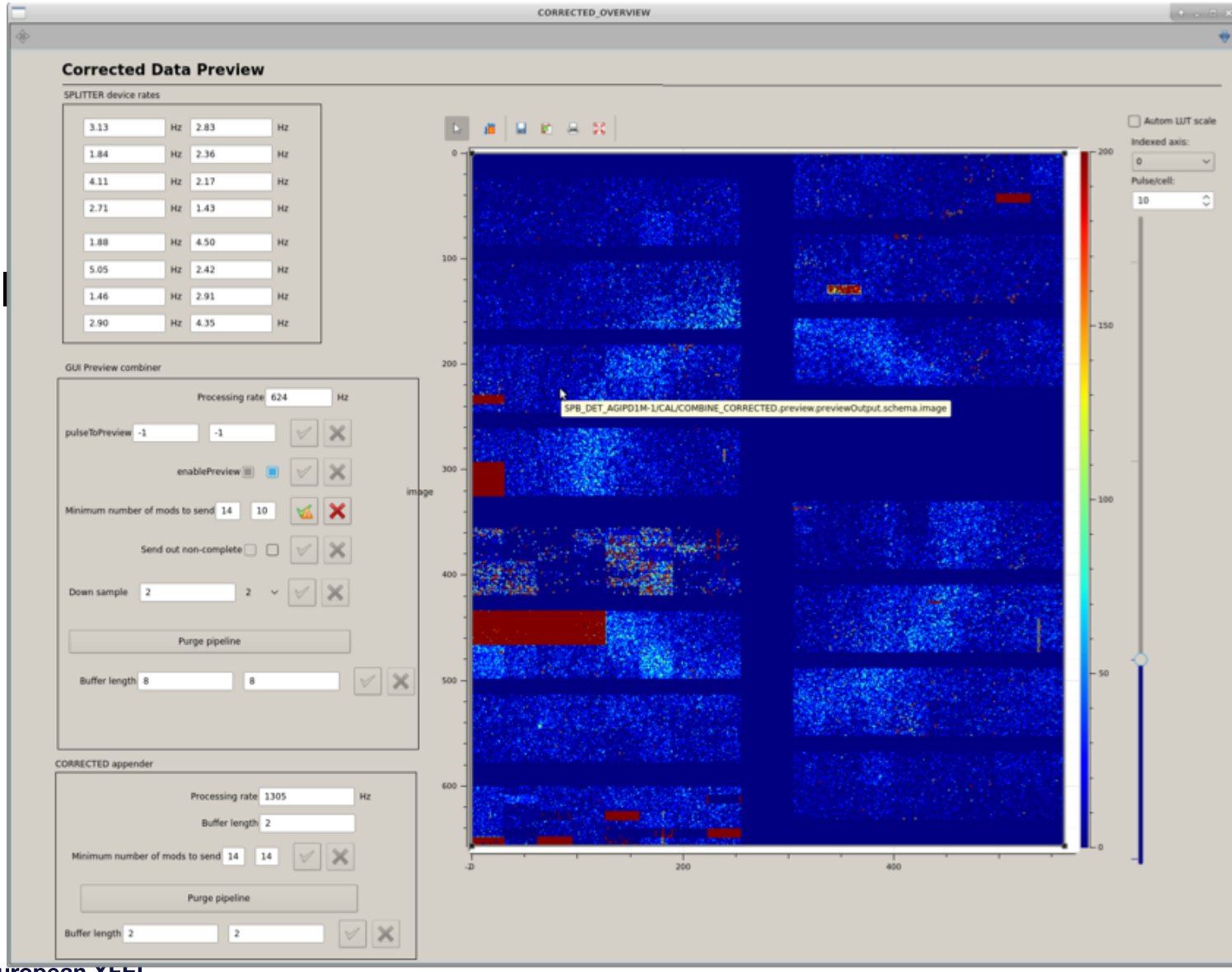
- Peer-to-peer model with **TCP** protocol
- Direct **data channels** between Karabo devices
  - Can dispatch data 1-to-n / n-to-1
  - Copy data to n clients
  - Policy on busy client: wait, queue, drop, exception
- Standardized format and **data container** (Karabo Hash)
- Provides data source and timestamp

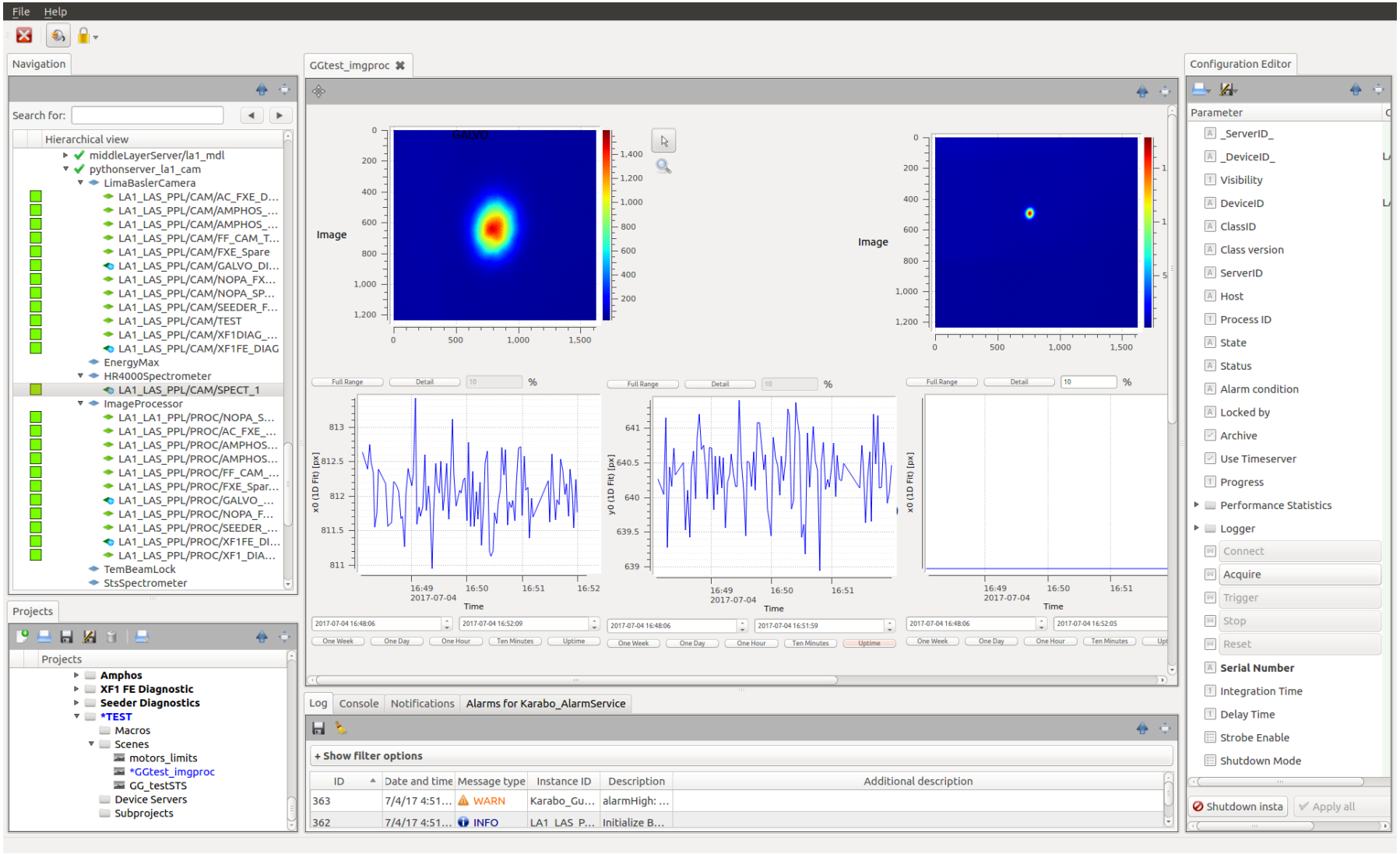


GUI ‘scenes’

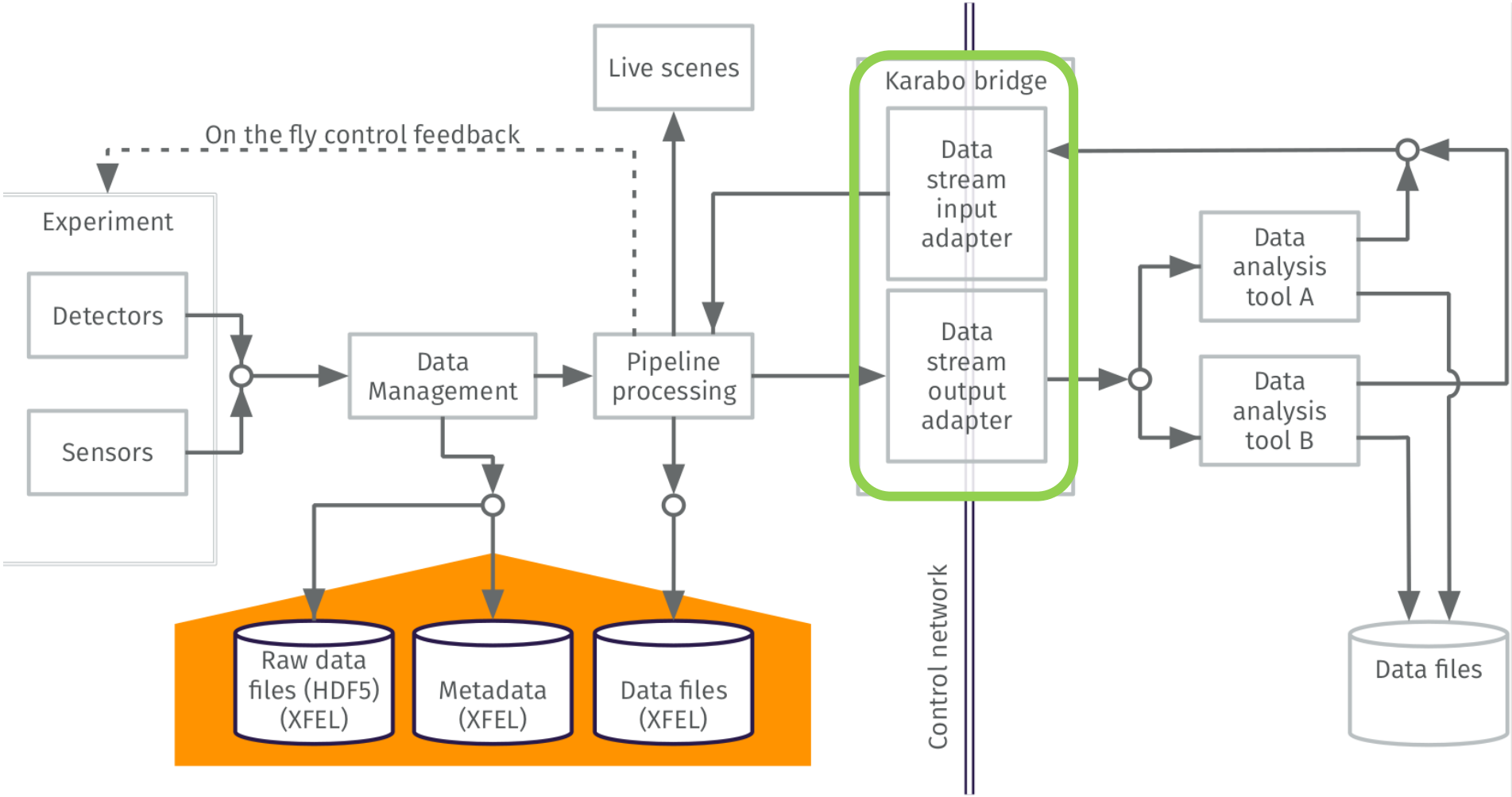

Live scenes
Karabo bridge
On the fly control feedback
Experiment
Detectors
Sensors
Data Management
Pipeline processing
Data stream input adapter
Data stream output adapter
Data analysis tool A
Data analysis tool B
Raw data files (HDF5) (XFEL)
Metadata (XFEL)
Data files (XFEL)
Control network
Data files

Online data analysis: Rapid feedback through GUI




File
Help
Navigation
Search for:
Hierarchical view
middleLayerServer/la1_mdl
pythonserver_la1_cam
LimaBaslerCamera
LA1_LAS_PPL/CAM/AC_FXE_D...
LA1_LAS_PPL/CAM/AMPHOS_...
LA1_LAS_PPL/CAM/AMPHOS_...
LA1_LAS_PPL/CAM/FF_CAM_T...
LA1_LAS_PPL/CAM/FXE_Spare
LA1_LAS_PPL/CAM/GALVO_DI...
LA1_LAS_PPL/CAM/NOPA_FX...
LA1_LAS_PPL/CAM/NOPA_SP...
LA1_LAS_PPL/CAM/SEEDER_F...
LA1_LAS_PPL/CAM/TEST
LA1_LAS_PPL/CAM/XF1DIAG_...
LA1_LAS_PPL/CAM/XF1FE_DIAG
EnergyMax
HR4000Spectrometer
LA1_LAS_PPL/CAM/SPECT_1
ImageProcessor
LA1_LA1_PPL/PROC/NOPA_S...
LA1_LAS_PPL/PROC/AC_FXE_...
LA1_LAS_PPL/PROC/AMPHOS...
LA1_LAS_PPL/PROC/AMPHOS...
LA1_LAS_PPL/PROC/FF_CAM_...
LA1_LAS_PPL/PROC/FXE_Spar...
LA1_LAS_PPL/PROC/GALVO_...
LA1_LAS_PPL/PROC/NOPA_F...
LA1_LAS_PPL/PROC/SEEDER_...
LA1_LAS_PPL/PROC/XF1FE_DI...
LA1_LAS_PPL/PROC/XF1_DIA...
TemBeamLock
StsSpectrometer
Projects
Projects
Amphos
XF1 FE Diagnostic
Seeder Diagnostics
*TEST
Macros
Scenes
motors_limits
*GGtest_imgproc
GG_testSTS
Device Servers
Subprojects
GGtest_imgproc
GALVO
Image
Image
Full Range
Detail
Time
Log
Console
Notifications
Alarms for Karabo_AlarmService
+ Show filter options
ID
Date and time
Message type
Instance ID
Description
Additional description
363
7/4/17 4:51...
WARN
Karabo_Gu...
alarmHigh: ...
362
7/4/17 4:51...
INFO
LA1_LAS_P...
Initialize B...
Configuration Editor
Parameter
ServerID
DeviceID
Visibility
DeviceID
ClassID
Class version
ServerID
Host
Process ID
State
Status
Alarm condition
Locked by
Archive
Use Timeserver
Progress
Performance Statistics
Logger
Connect
Acquire
Trigger
Stop
Reset
Serial Number
Integration Time
Delay Time
Strobe Enable
Shutdown Mode
Shutdown insta
Apply all

Karabo Bridge



Karabo Bridge collaboration

16th Int. Conf. on Accelerator and Large Experimental Control Systems ICALEPCS2017, Barcelona, Spain JACoW Publishing
ISBN: 978-3-95450-193-9 doi:10.18429/JACoW-ICALEPCS2017-TUCPA01

DATA ANALYSIS SUPPORT IN KARABO AT EUROPEAN XFEL

H. Fangohr†, M. Beg, V. Bondar, D. Boukhelef, S. Brockhauser, C. Danilevski, W. Ehsan, S. G. Esenov, G. Flucke, G. Giovanetti, D. Göries, S. Hauf, B. Heisen, D. G. Hickin, D. Khakhulin, A. Klimovskaia, M. Kuster, P. M. Lang, L. Maia, L. Mekinda, T. Michelat, A. Parenti, G. Previtali, H. Santos, A. Silenzi, J. Sztuk-Dambietz, J. Szuba, M. Teichmann, K. Weger, J. Wiggins, K. Wrona, C. Xu
European XFEL GmbH, Holzkoppel 4, 22869 Schenefeld, Germany

S. Aplin, A. Barty, M. Kuhn, V. Mariani
Centre for Free Electron Laser Science, DESY, Notkestrasse 85, 22607 Hamburg, Germany

T. Kluyver
University of Southampton, SO17 1BJ Southampton, United Kingdom

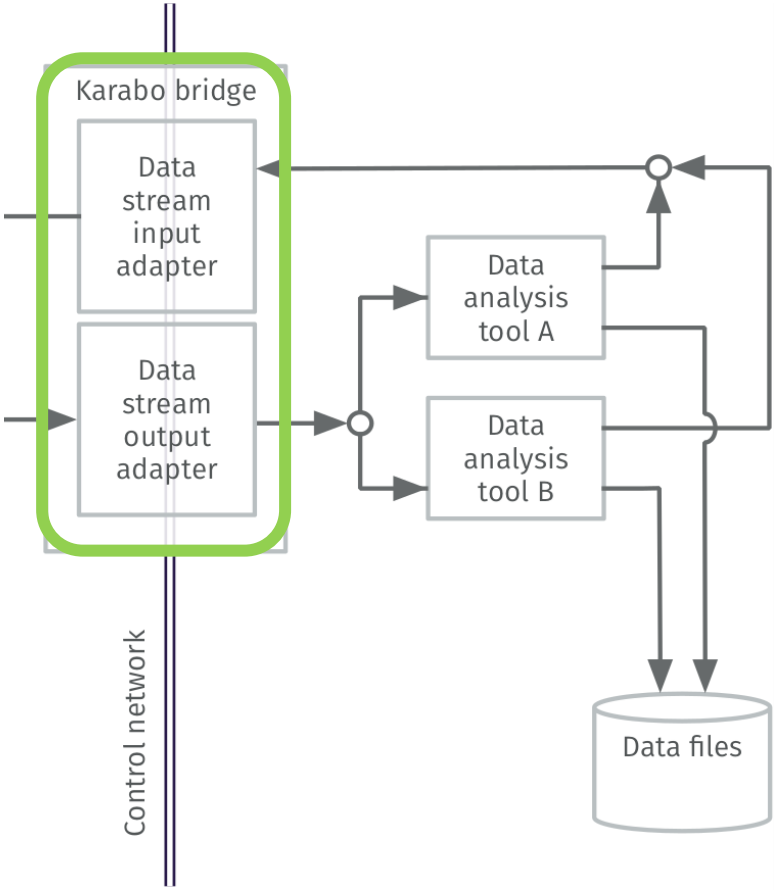
Abstract

We describe the data analysis structure that is integrated into the Karabo framework to support scientific experiments and data analysis at European XFEL. The photon science experiments have a range of data analysis requirements, including online (i.e. near real-time during the actual measure-

per second per detector at European XFEL [2] demand an efficient concurrent approach of performing experiments and data analysis: Data analysis must already start whilst data is still being acquired and initial analysis results must immediately be usable to feedback into and re-adjust the current experiment setup. The Karabo control system [3]

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 from CFEL)



Karabo Bridge Client

- Install the client

```
pip install -e git+https://github.com/European-XFEL/karabo-bridge-py.git#egg=karabo-bridge-py
```

- How to use it

Import Karabo bridge client

```
In [1]: from karabo_bridge import KaraboBridge
```

How to use it?

```
In [2]: help(KaraboBridge)
```

```
Help on class KaraboBridge in module karabo_bridge.KaraboBridge:

class KaraboBridge(builtins.object)
 |  Karabo bridge client for Karabo pipeline data.
 |
 |  This class can request data to a Karabo bridge server.
 |  Create the client with::
 |
 |      krb_client = KaraboBridge("tcp://153.0.55.21:12345")
 |
 |  then call ``data = krb_client.next()`` to request next available data
 |  container.
 |
 |  Parameters
 |  ----------
 |  endpoint : str
 |      server socket you want to connect to (only support TCP socket).
 |  sock : str, optional
 |      socket type - supported: REQ.
 |  ser : str, optional
 |      Serialization protocol to use to decode the incoming message (default
 |      is msgpack) - supported: msgpack,pickle.
 |
```

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 [3]: kb = KaraboBridge('tcp://max-exfl093:45632')
```

request the next data available on this server.

```
In [4]: train = kb.next()
```

The data container is a dictionary.

```
In [5]: type(train)
Out[5]: dict
```

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

```
In [6]: train.keys()
Out[6]: dict_keys(['detector', 'DETLAB_LAB_LPD-1/FPGA/FEM_Q2M0', 'DETLAB_LAB_LP
D-1/FPGA/FEM_Q1M0', 'DETLAB_LAB_LPD-1/FPGA/FEM_Q0M0', 'DETLAB_LAB_LPD-
1/FPGA/FEM_Q3M0'])
```

Karabo Bridge Client

- Each data source is a dictionary
  - It contains device parameters
  - And source metadata
- All data are python built-in types
- Big array are Numpy array
- Requesting data will return the latest available train in the pipeline

find the parameters keys for a specific data source.

```
In [7]: train['detector'].keys()
```

```
Out[7]: dict_keys(['image.data', 'image.trainId', 'ignored_keys', 'metadata',
'trainId', 'image.pulseId', 'image.cellId'])
```

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

```
In [8]: train['detector']['metadata']
```

```
Out[8]: {'source': 'DETLAB_LAB_DAQ-0/DET/0:xtdf',
 'timestamp': {'frac': 0, 'sec': 1516198931, 'tid': 1516199957}}
```

Example, getting detector image (numpy array) and display array informations.

```
In [9]: im = train['detector']['image.data']
print('shape:', im.shape)
print('dtype:', im.dtype)
print(im[0, 0, 0:2, 0:2])
```

```
shape: (5, 4, 256, 256)
dtype: float64
[[3910.05812037 4041.62319897]
 [4113.51273402 4056.11034393]]
```

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

```
In [18]: for i in range(5):
    data = kb.next()
    print(data['detector']['trainId'])
```

```
1516200478
1516200481
1516200483
1516200483
1516200484
```

Karabo Bridge Client

- You can instantiate many clients
  - Data can be dispatched among them

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

In [52]:
```
client_2 = KaraboBridge('tcp://max-exfl093:45632')
data = client_2.next()
print(data['detector']['trainId'])
```

```
1516380752
```

In [53]:
```
client_3 = KaraboBridge('tcp://max-exfl093:45632')
data = client_3.next()
print(data['detector']['trainId'])
```

```
1516380753
```

  - Or copy to all
    - ▶ PUB-SUB sockets

Karabo Bridge Client – Try this at home!

- Karabo Bridge server simulation
  - Does not require Karabo
  - Helps testing integration of the client to your tool

```
Terminal
tmichela@exflpcx17673 ~/projects/karabo-bridge-py/examples
 % ./demo.sh
demo.sh: starting (simulated) server
demo.sh: starting client
Client : received train ID 15163874924
Client : - detector image shape is (32, 16, 512, 128), 64.0 Mbytes
Client : received train ID 15163874931
Client : - detector image shape is (32, 16, 512, 128), 64.0 Mbytes
Server : buffered train: 15163875269
Client : received train ID 15163874939
Client : - detector image shape is (32, 16, 512, 128), 64.0 Mbytes
Client : received train ID 15163874945
Client : - detector image shape is (32, 16, 512, 128), 64.0 Mbytes
Server : buffered train: 15163875274
Client : received train ID 15163874950
Client : - detector image shape is (32, 16, 512, 128), 64.0 Mbytes
Client : received train ID 15163874955
Client : - detector image shape is (32, 16, 512, 128), 64.0 Mbytes
Server : buffered train: 15163875280
Client : received train ID 15163874960
Client : - detector image shape is (32, 16, 512, 128), 64.0 Mbytes
Client : received train ID 15163874965
Client : - detector image shape is (32, 16, 512, 128), 64.0 Mbytes
Client : received train ID 15163874970
```

```python
# server.py

from karabo_bridge import server_sim

# start a simulated karabo bridge server
# and bind a socket on port 4545 of this machine (localhost).
server_sim(4545)
```

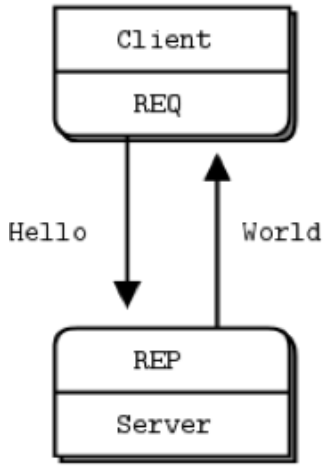
```python
# client.py

from karabo_bridge import KaraboBridge

# connect the client to localhost if running on the same machine as the server.
client = KaraboBridge('tcp://localhost:4545')

while True:
    data = client.next()
    det_data = data['SPB_DET_AGIPD1M-1/DET/detector']
    print("Client : received train ID", str(det_data['header.trainId']))
    print("Client : - detector image shape is {}, {} Mbytes".format(
          det_data['image.data'].shape, det_data['image.data'].nbytes/1024**2))
```

Karabo Bridge – technical details
Networking library



- ZeroMQ
  - Intelligent **socket library** for messaging
  - Many kind of connection **patterns**
  - Multiplatform, **multi-language** (30+)
  - Open source LGPL
  - **large user community** (including Jupyter)

ØMQ Hello World

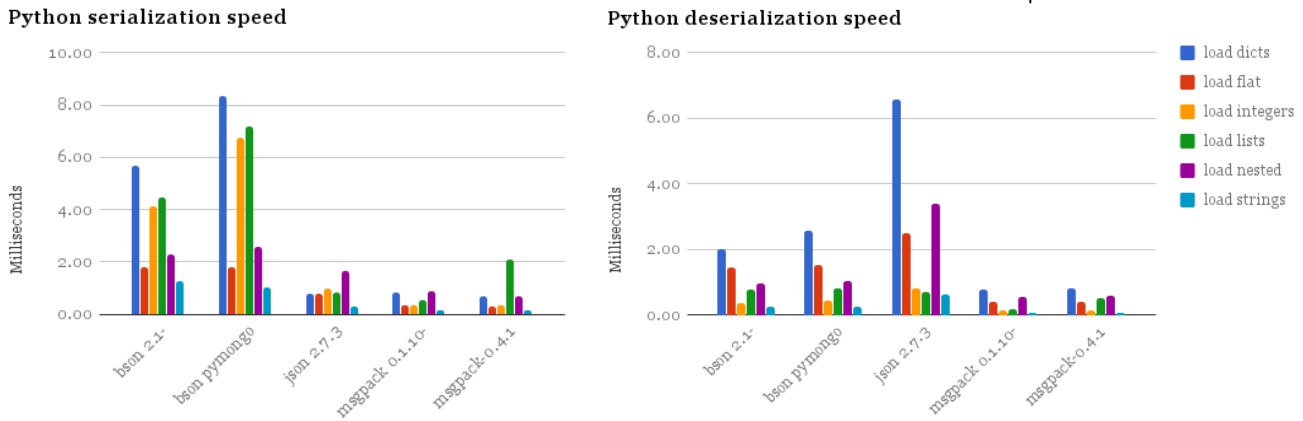
```
import org.zeromq.ZMQ;
public class hwclient {
    public static void main (String[] args){
        ZMQ.Context context = ZMQ.context (1);
        ZMQ.Socket socket = context.socket (ZMQ.REQ);
        socket.connect ("tcp://localhost:5555");
        socket.send ("Hello", 0);
        System.out.println (socket.recv(0));
    }
}
```

```
import org.zeromq.ZMQ;
public class hwserver {
    public static void main (String[] args) {
        ZMQ.Context context = ZMQ.context(1);
        ZMQ.Socket socket =
  context.socket(ZMQ.REP);
        socket.bind ("tcp://*:5555");
        while (true) {
            byte [] request = socket.recv (0);
            socket.send("World", 0);
        }
    }
}
```

- Message blobs of 0 to N bytes
- One socket to many socket connection
- Queuing at sender and receiver
- Automatic TCP (re)connect
- **Easy** to use

Karabo Bridge – technical details
Message serialization

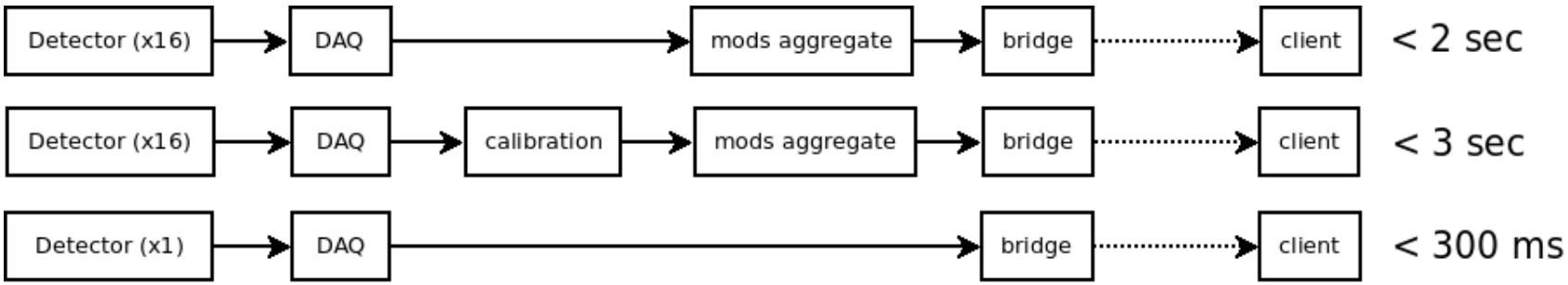
Source: https://wtanaka.com/node/8100



- Serialization
  - Pickle
  - boost::serialization
  - **MessagePack**
  - Protobuf
  - …
- MessagePack
  - Simple and open source design: https://github.com/msgpack/msgpack/
  - **JSON-like** binary format
  - But **faster** and **smaller**
  - **Multi-language** (80+ implementation available)
  - Easy implementation if need to support new language

Summary Karabo Bridge

- Network interface to access scientific data during experiment in near real time
  - Keep the same data structure and names as in Karabo Hash and HDF5 files
  - Easy set-up to export any data pipeline from Karabo
- Client implementation and simulator
  - Python: https://github.com/European-XFEL/karabo-bridge-py
  - C++: implementation existing
- Successful use during first experiments
  - OnDA, Hummingbird, CASS, custom
- Performance (SPB experiment, AGIPD detector)



Containers & Jupyter

- Jupyter Notebook
  - Executable document
  - Code, output, interpretation
- Jupyter Ecosystem
  - Docker, Binder
  - Reproducibility -> better science
- Potential to support Online DA?

jupyter jupyterdemo Logout

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3

Code cells show code input and output:

```
In [1]: 1 + 2
Out[1]: 3
```

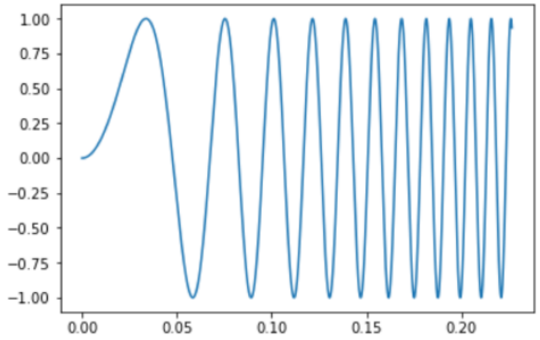
Cells can contain text and latex equations such as $f(x) = \sin(2\pi\omega t^2)$ and $\omega = 220$ Hz. We can use code to define the corresponding functions:

```
In [2]: import numpy as np
        def f(t):
            omega = 220
            return np.sin(2 * np.pi * omega * t**2)
```

```
In [3]: f(0)   # call the function
Out[3]: 0.0
```

Let's compute the data and plot the beginning of it:

```
In [4]: t = np.linspace(0, 2, 44100)
        y = f(t)
        ## Show plots inside the notebook
        %matplotlib inline
        import pylab
        pylab.plot(t[0:5000], y[0:5000])
Out[4]: [<matplotlib.lines.Line2D at 0x10a267898>]
```

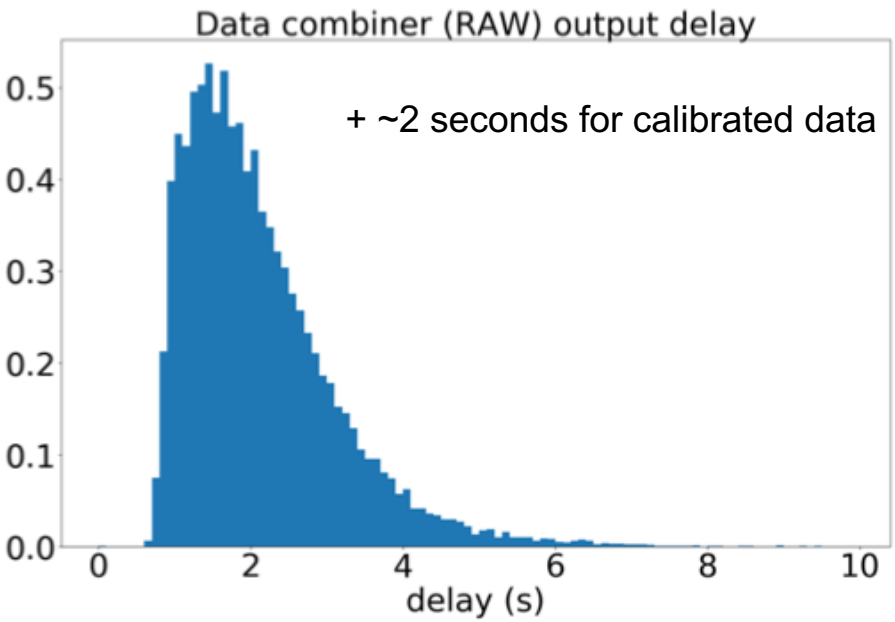


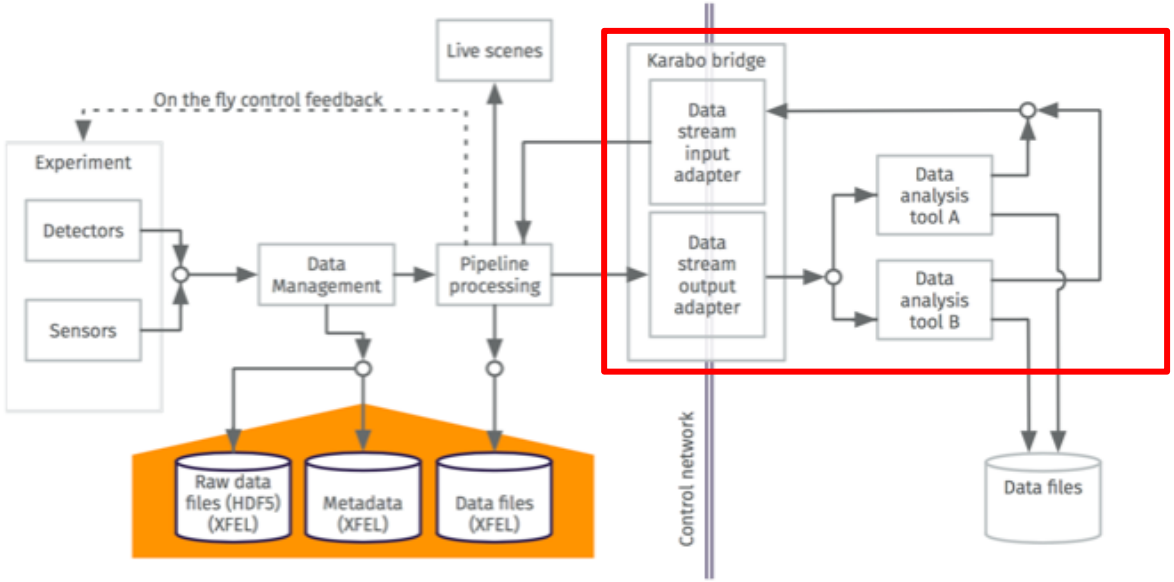
Summary

- Outlined basics of online data analysis at European XFEL
  - Quasi real time analysis within Karabo
  - Online GUI elements
  - Lightweight (0MQ) interface to integrate external applications
- Very early stages
- Development of growing set of open source tools
- Contact
  - Thomas.Michelat@xfel.eu, Hans.Fangohr@xfel.eu, Sandor.Brockhauser@xfel.eu
- Acknowledgements: CFEL, XFEL groups Detectors, ITDM, SPB, FXE, CAS
- Reference
  - H. Fangohr et al, Data Analysis support in Karabo at European XFEL, ICALEPSC 2017, online: http://icalepcs2017.vrws.de/papers/tucpa01.pdf

Online Analysis performance
Early User Experiment (S Hauf)

- Feeds user-provided online tools via a Karabo-Bridge device
  - 3-5 Hz rate at 64 cells measured
  - 2-4s latency with 64 memory cells





- Latency includes:
  - Data acquisition
  - Data formatting on DAQ
  - Data forwarding to pipelines
  - Data selection at pipeline entry points:
    - ► Every nth train
  - Combining of 16 streams from modules
  - Data advertising on ZMQ