Status of the *Karabo* Control and Data Processing Framework

Dr. Gero Flucke *et al.*
European XFEL GmbH

17th International Conference on Accelerator and Large Experimental Physics Control Systems
(ICALEPCS 2019)
New York, USA

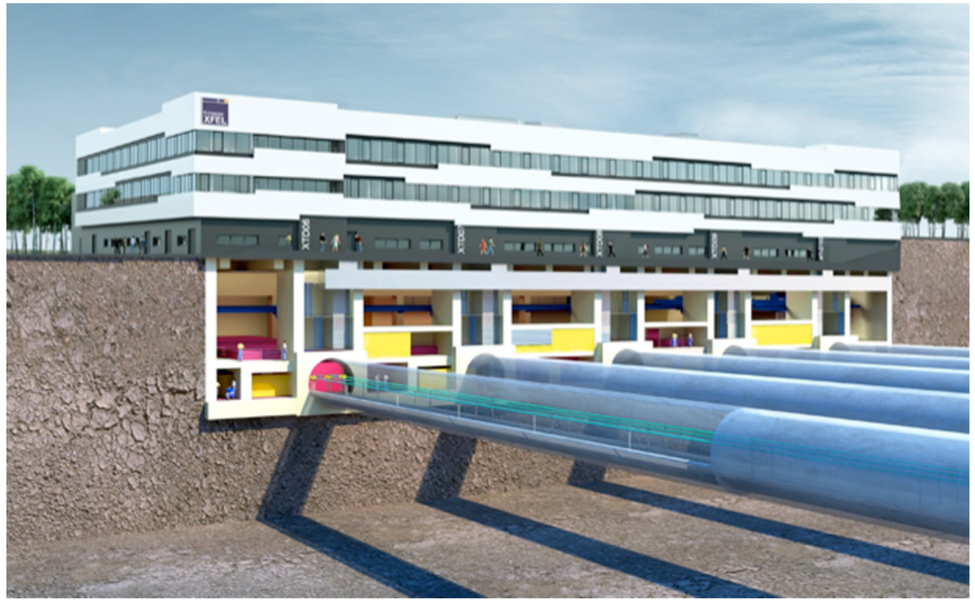
October 7-11, 2019

Outline

- Why there is Karabo?
  - The European XFEL
- Karabo Overview
  - Communication
  - Services and Interfaces
- Achievements and Lessons Learned
- Future Directions

The Home of Karabo: European X-ray Free Electron Laser (XFEL):

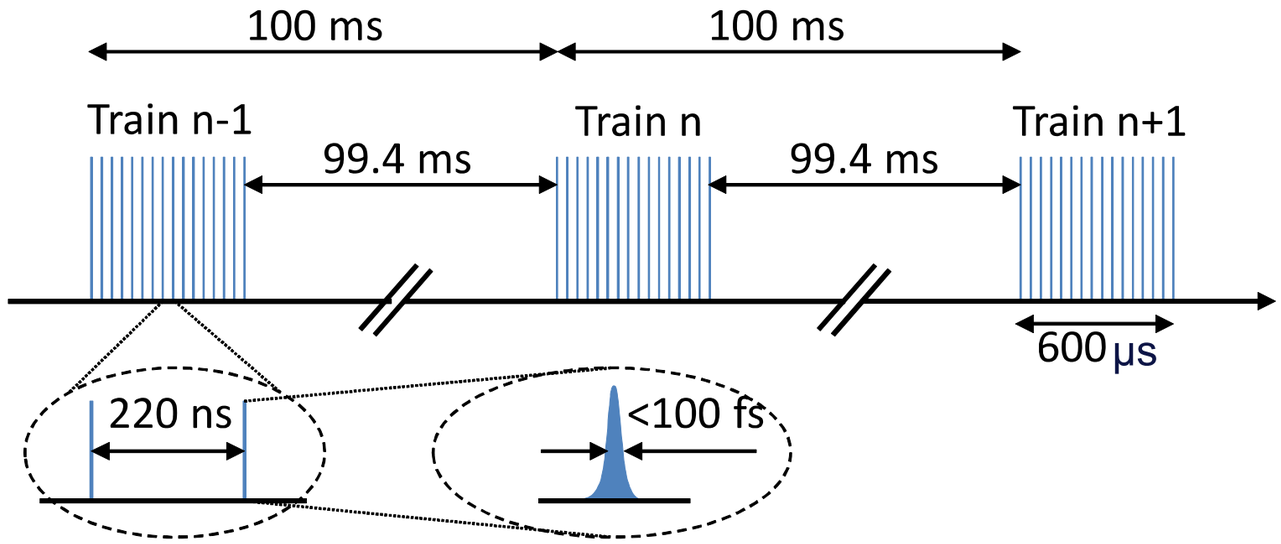
- Linear electron accelerator
  - run by DESY
- Undulators creating X-ray laser photons
- Photon beam steered
  - through 3 tunnels
  - to 6 instruments



Karabo:
Designed and developed for control, data acquisition, analysis

Why a New Control System?

- Photon beam at European XFEL has a particular pattern:
  - 10 Hz of "trains" of up to 2700 pulses
  - Inter-pulse spacing 220 ns (4.5 MHz)
- Custom-made MHz-capable 2D detectors
  - Single pulse resolution: 16 GB/s
  - Calibration required for online preview



- ➔Tight integration of control and online analysis required

European XFEL decided in 2011 to develop a custom *control and data processing* framework

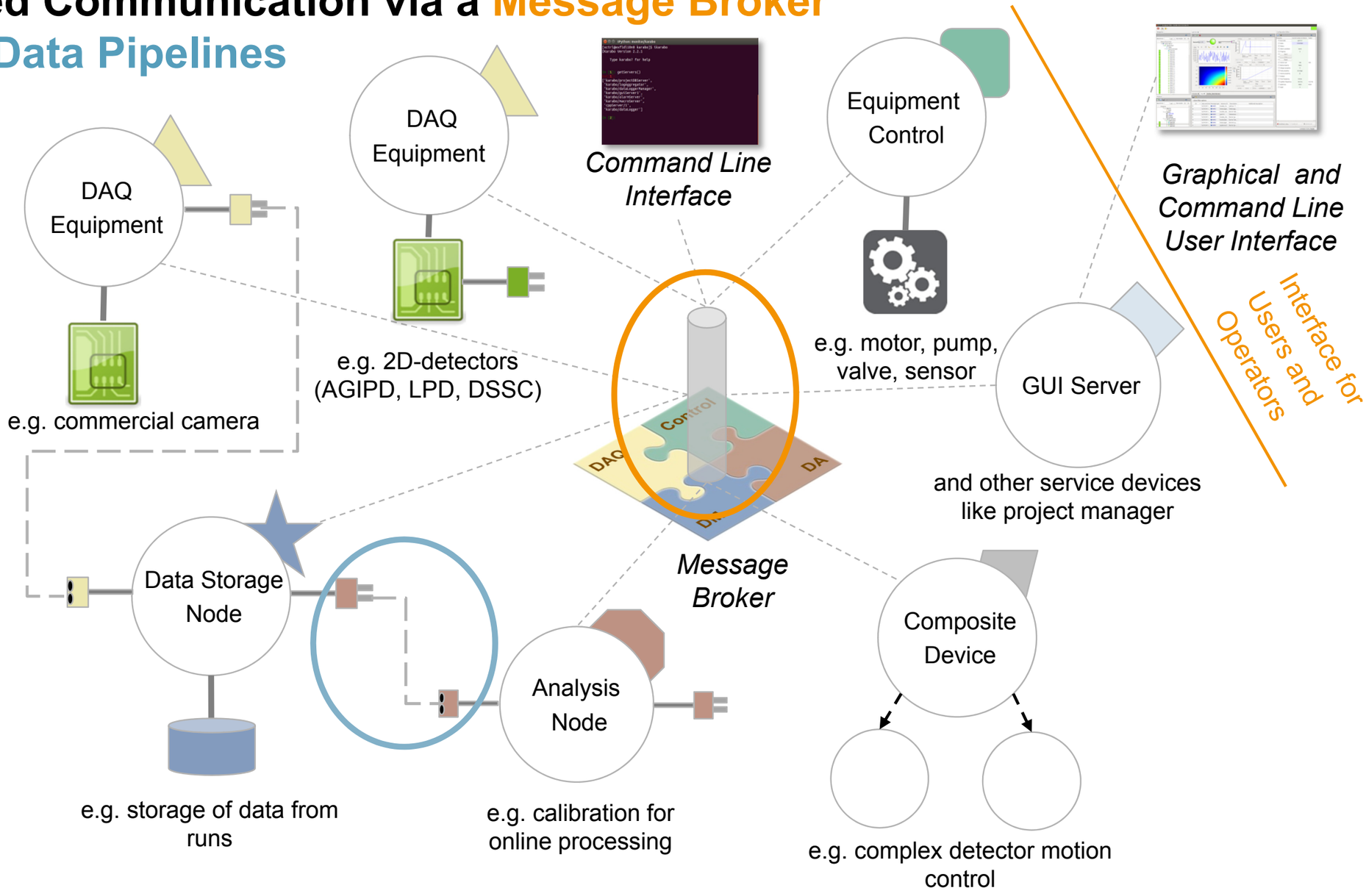
➔Karabo

(presented at ICALEPCS2013)

Karabo: Device Based Communication via a Message Broker and TCP/IP Data Pipelines

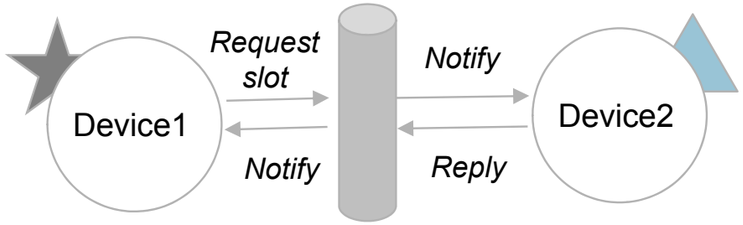
Self-describing
Karabo Devices

- Equipment control, e.g. motors, valves,…
- Detectors like
  - 2D detectors,
  - cameras
- Data storage
- Online data analysis, e.g. calibration
- System services

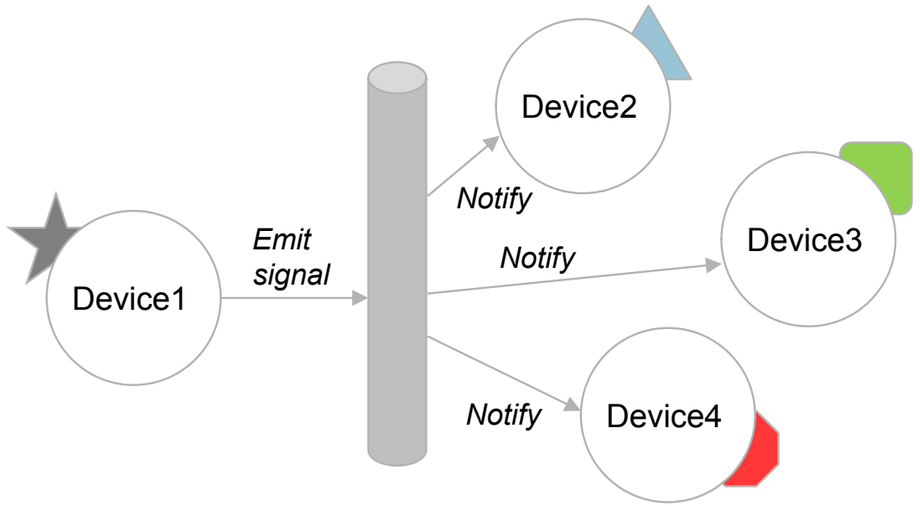


Karabo Communication Patterns

- Request/reply
  - Device registers methods as "slots".
  - Call from remote
    - ► with up to four arguments and return values.

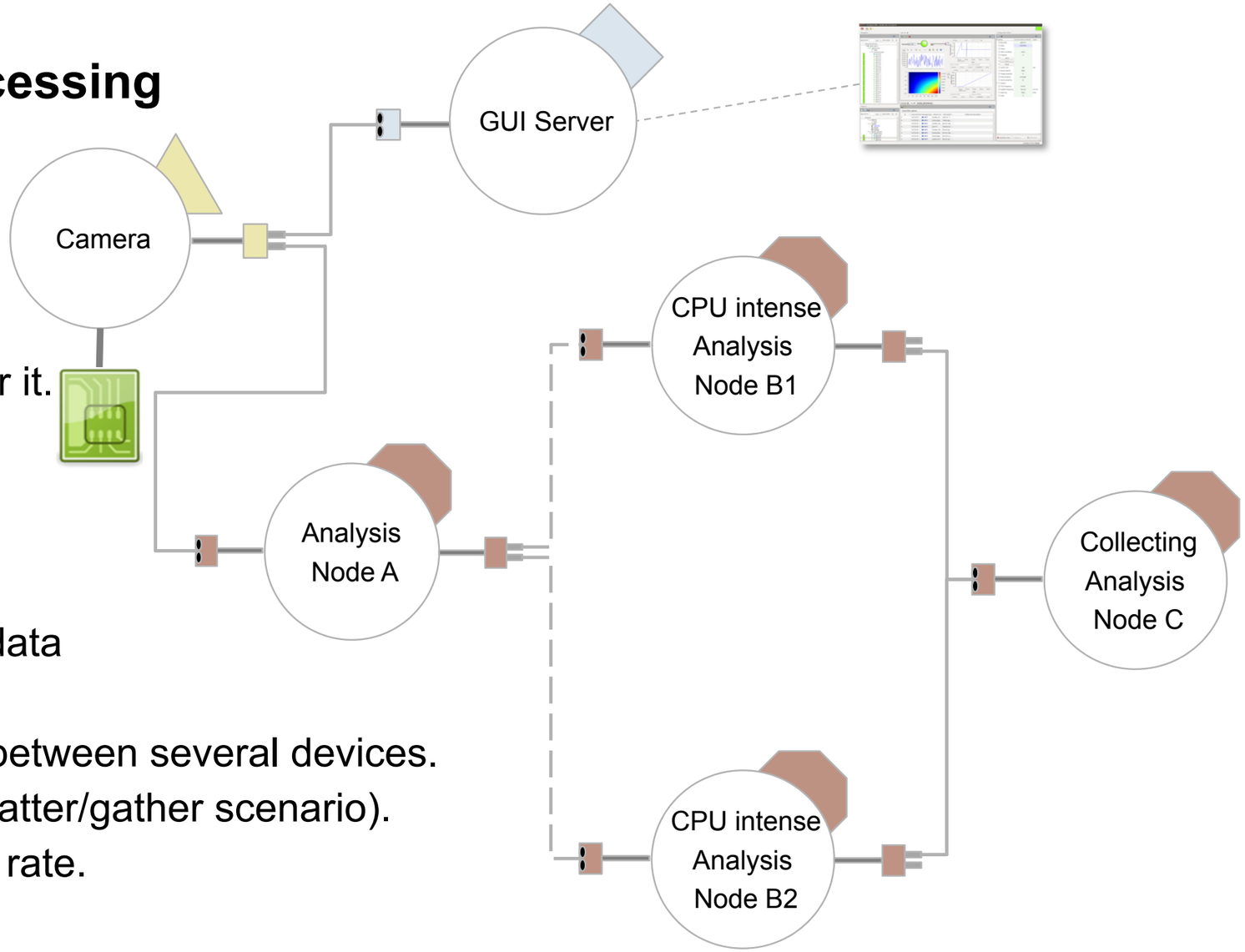


- Publish/subscribe
  - Devices subscribe slots to a remote "signal".
  - When signal is "emitted",
    all subscribed slots are called.
    - ► No publishing overhead for "popular" devices
    - ► **Karabo framework is completely event-driven**:
      regular **polling obsolete**.



Karabo Pipelines for Data Processing

- Complement broker communication
  - Using direct TCP/IP connections.
- Designed for (large) multi-D data.
  - Sent only when receiver ready for it.
  - But processing and transferring in parallel.
- Offer flexible configuration:
  - Sender can *wait*, *drop* or *queue* data when receiver not ready yet.
  - Individual *copy* vs. data *shared* between several devices.
  - Collect from several senders (scatter/gather scenario).
  - Delay readiness report to reduce rate.



Hash: Karabo's Flexible Data Container

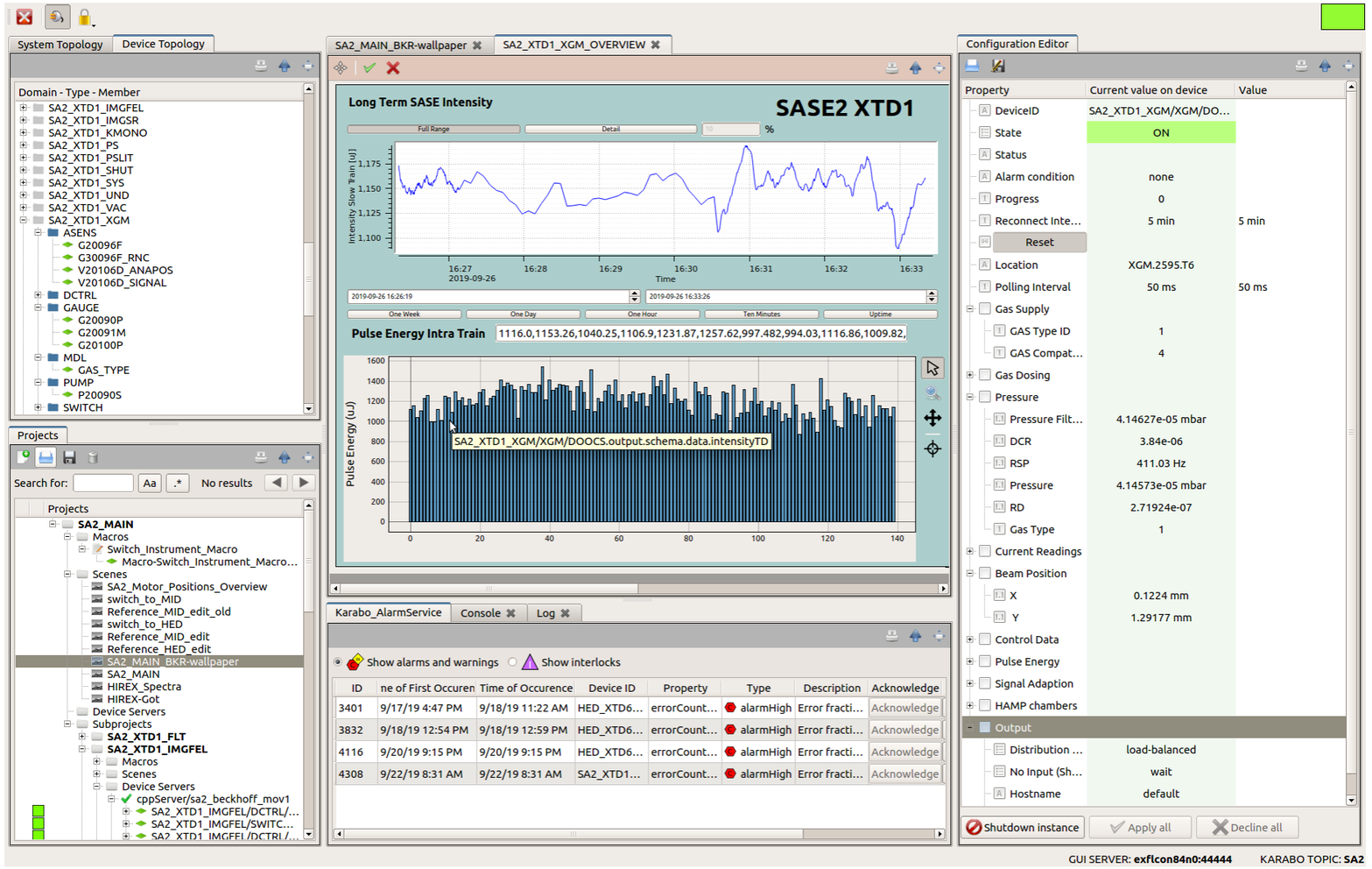
- A nested key-value container with attributes:
  - key: string
    - ▶ direct nested access: separate key levels by dot: `h.get("key1.key2.key3")`,
  - value: any type,
  - attributes per value: key-value container.

- Hash available in all three Karabo APIs:
  - C++
  - Python
    - ▶ "Bound" (C++ bindings)
    - ▶ "Middlelayer" (pythonic)

- Serialisation to Hdf5, XML and binary format.
  - Supported data types:
    - ▶ Scalars, complex, strings and vectors thereof,
    - ▶ "NDArray" for pipelines,
    - ▶ "ImageData": NDArray and meta data
      - not yet (!) or "Middlelayer" API.

```
In [1]: from karabo.bound import Hash

In [2]: h = Hash('a', 'square')

In [3]: h['b.c'] = 42

In [4]: h
Out[4]:
'a' => square STRING
'b' +
  'c' => 42 INT32

In [5]: h.setAttribute('a', 'colour','red')

In [6]: h
Out[6]:
'a' colour="red" => square STRING
'b' +
  'c' => 42 INT32
```

Graphical User Interface (PyQt)



Unified integrated cockpit:

- For experts:
  - Full system view.
  - Detailed access to configurations and commands.
  - Integrated command line.
- For everybody:
  - Customizable "scenes"
    - ► no coding required,
    - ► drag-and-drop properties.
  - Rich set of widgets.
- For operation:
  - Standalone scenes.

System Services

- Shipped as part of Karabo framework
  - GUI server device
  - Server to centrally run macros and GUI command lines
  - Project Database
    - ► Joint storage of GUI scenes, macros, device configurations
  - Alarm service: 

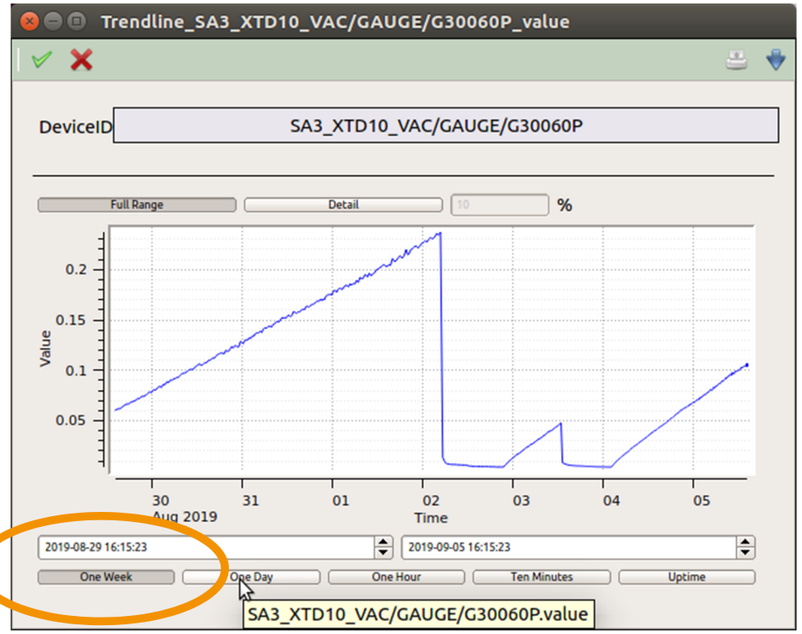
    - ► Track device properties outside normal operation value
  - **Data Logging**

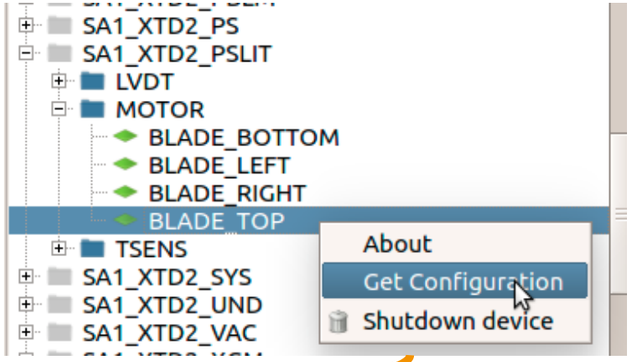
- Outside Karabo framework by independent software devices
    - ► E.g. to allow faster release cycle
  - **Data acquisition**
  - Scan tool

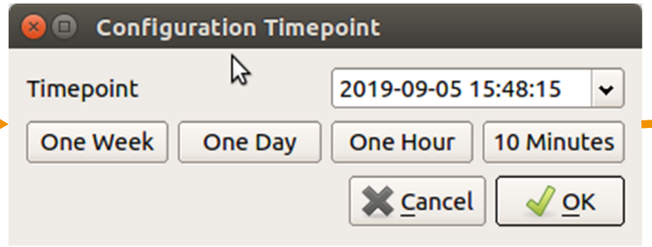
Karabo Data Logging

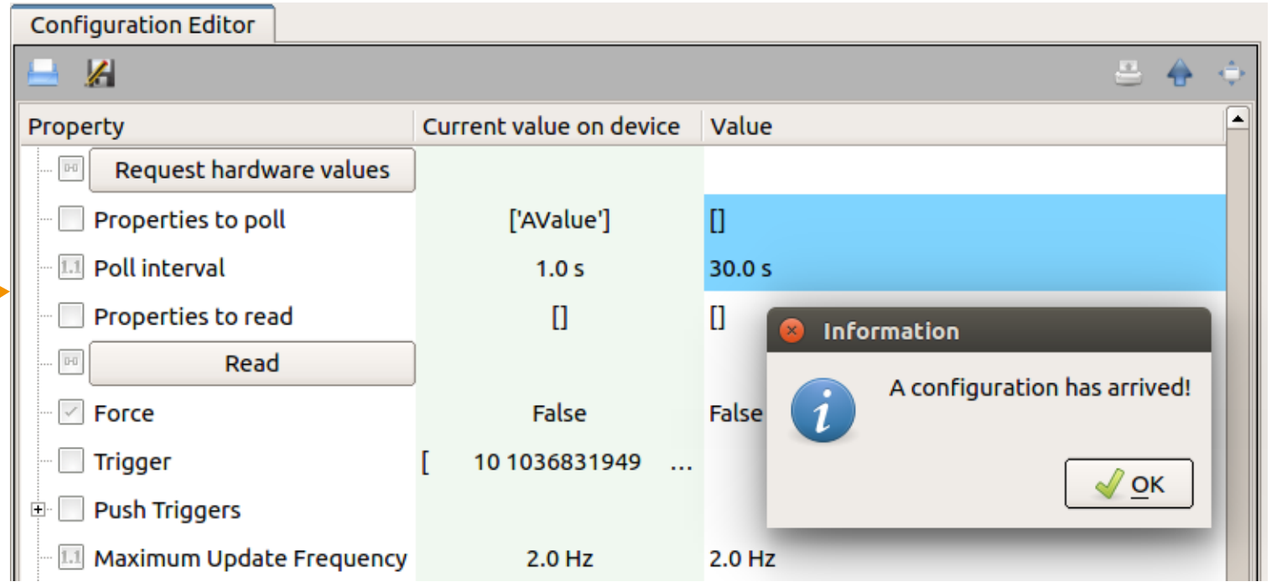
- In-built data logging and retrieval mechanism.
  - Control data only, no pipelines.

- Main control use cases:
  - Past data for trendlines: single scalar property vs time.
  - Past configurations: all device properties at point in time.



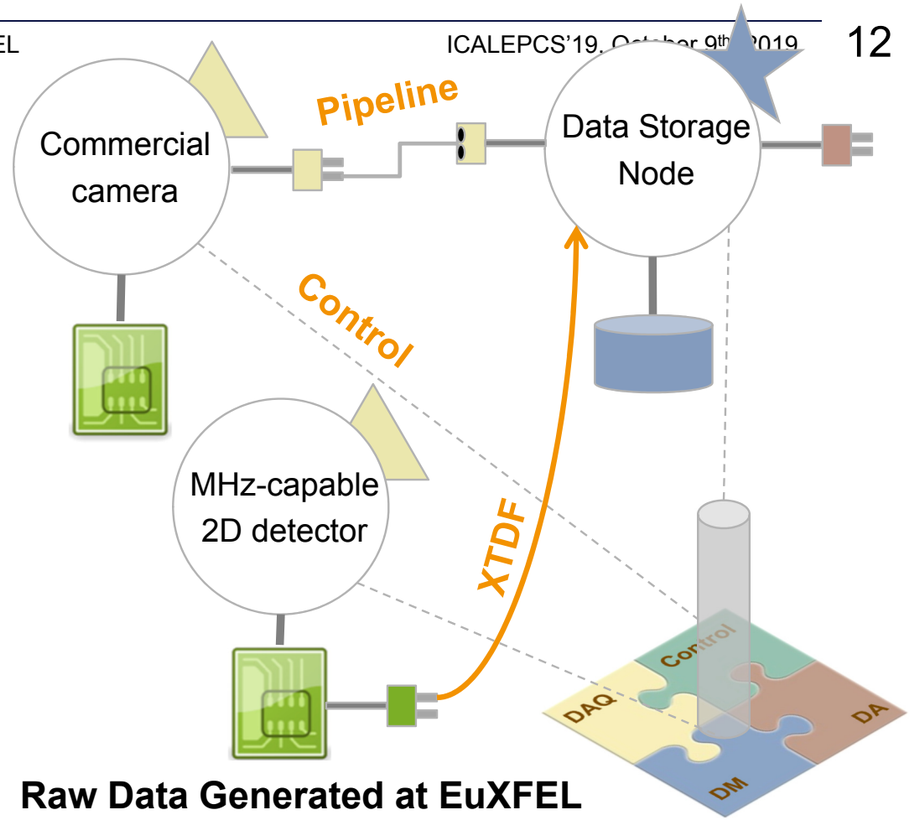


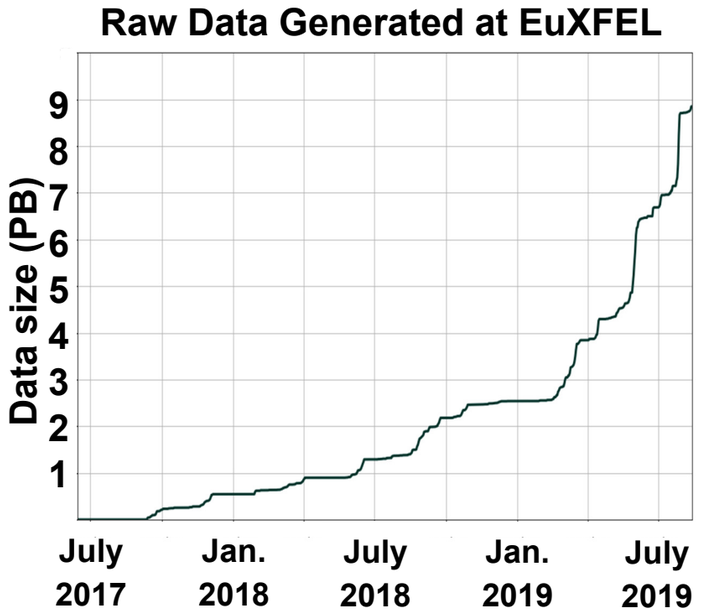




Karabo Data Acquisition (DAQ) Integration

- Focus on scientific instrument data with long term storage: EuXFEL data policy
- Support for different types of data sources:
  - **Control** data with train resolution: e.g. sensors, motors → **slow data**
  - 2D or pulse resolved data: e.g. **pipeline** from cameras, digitizers → **fast and/or medium sized data**
  - MHz-capable 2D detectors (XFEL train data format - **XTDF**) → **big & fast data**
- Data stored in HDF5 files, indexed per train
  - 9 PB raw data stored since experiments started
  - 12 GB/s achieved (600 images per train)
- Provide data stream for online display and analysis:
  - Calibration of big 2D detectors (1.8 GB/s, 2s latency),
  - External tool via Karabo-to-ZeroMQ bridge

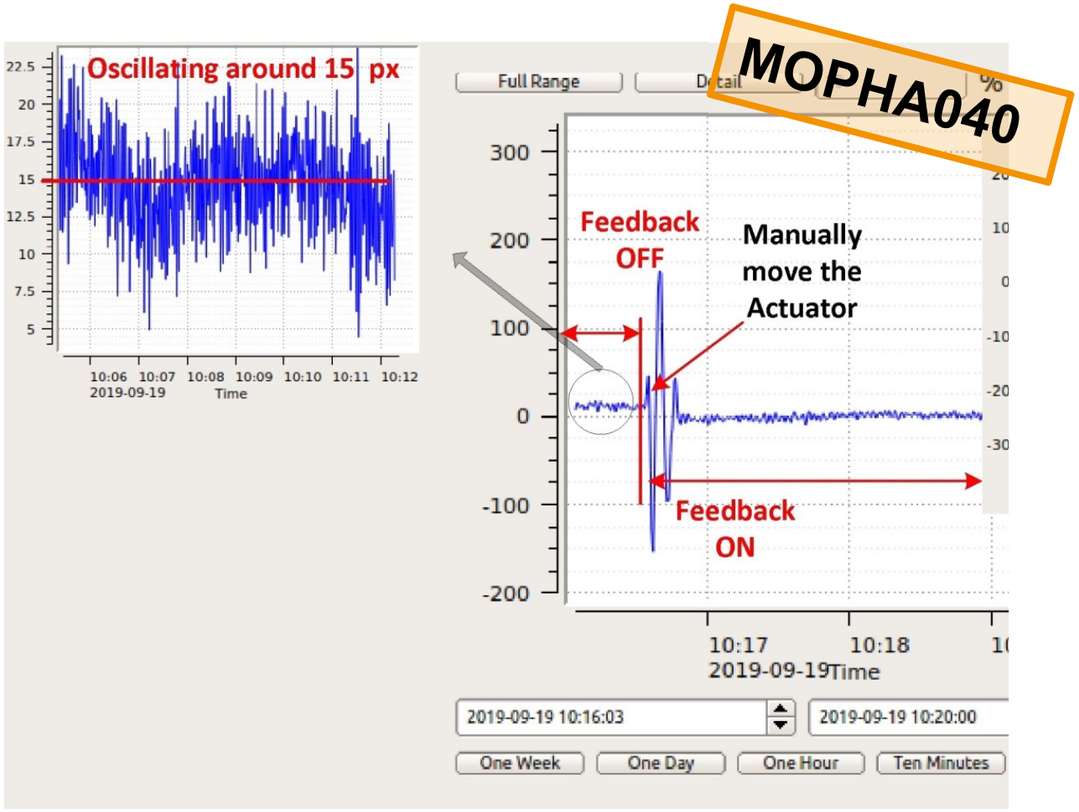




Achievements

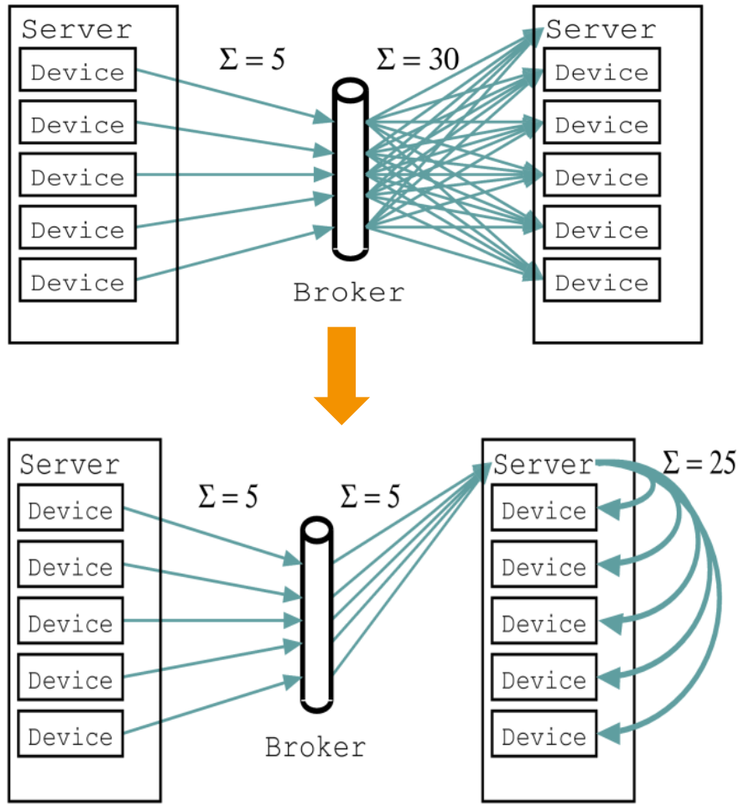
- Karabo in use at EuXFEL:
  - Commissioning since 2016
  - First user experiments 2017
    - ► meanwhile on all 6 instruments
  - Almost 14,000 devices,
    - ► more than 1.6 million "control points" (properties, commands)
    - ► average broker rate per installation: up to 1 kHz

- Example application combining processing and control: "Beam Position Feedback System Supported by Karabo at European XFEL"
  - Compensates drift for EuXFEL beam pointing stability
  - See poster MOPHA040.



Lessons Learned

- Early versions of Karabo had serious communication delays
  - ► Especially for C++ servers hosting many hundred devices in one process
  - Measure 1: No blocking calls in shared event loop,
    - ► use asynchronous coding patterns
  - Measure 2: Broadcast messages only once per process
    - ► Before: 500 devices start, 500 other devices receive
      - ➔ 250 k messages



- Calibration pipelines suffered from slow (de-)serialisation
  - Now avoid copies of NDArray data

- Rare, but serious issues with message ordering
  - Respect that posting functions on multi-threaded event loop has no order guarantee

```
set("motor/A",
    "targetPosition", 2);
execute("motor/A", "move");
```

Developing and Releasing Karabo Framework

- Unit and integration tests are vital:
  - Running with continuous integration
  - No new functionality without test
  - Good test coverage
    - ► 67% of C++ and 74% of Python,
  - Final release test cycle includes device classes
- Code review mandatory
- Karabo shall be released with an open source license
  - Planned since the beginning
  - Discovered conflicting licenses in dependencies: GPLv2 *only* and Apache 2.0
  - Cleaning this up has progressed, but still steps to go…

Future Directions

Steered by experiences gained in European XFEL operation:

- Data logging:
  - Replace custom text and index file based backend
  - Time series database **InfluxDB** tested
    - ► Will need enterprise edition in production
- Device configuration storage
  - Current solution with independent "projects" very flexible
  - Central storage more applicable for commissioned system
- Authentication and authorisation foreseen in original design, but not implemented
  - Becomes a limiting factor in more and more stable system: Who did what?
- JMS broker is very stable
  - but openMQc client library not maintained,
    - ► C++ prototype of communication unit for MQTT broker developed

Summary

- European XFEL endeavored the development of a **new control and data processing framework: Karabo**
  - Broker based control communication
  - TCP/IP pipelines for big data transport
  - Fully event-driven
  - C++ and two Python APIs
- Many **lessons learned** to make it scale:
  - Now operate reliably about 14,000 devices with 1.6 million control points at European XFEL
  - Online calibration of MHz-capable 2D detectors with < 2 s latency, 1.8 GB/s throughput
- **Future directions** driven by needs to control European XFEL instruments
  - Improve **data logging**, more **central configuration** storage, add **authentication/authorisation**
- Goal: publish with an open source license
  - Even before we manage that: If interested, do not hesitate to contact us.

Authors

- Members of the European XFEL Groups for
  Controls, Data analysis, IT and Data Management, and Detector Operation:
  - N. Al-Qudami, M. Beg, M. Bergemann, V. Bondar, D. Boukhelef, S. Brockhauser, C. Carinan, R. Costa, F. Dall'Antonia, C. Danilevski, W. Ehsan, S. G. Esenov, R. Fabbri, H. Fangohr, G. Flucke, D. Fulla Marsa, G. Giovanetti, D. Goeries, S. Hauf, D. G. Hickin, E. Kamil, Y. Kirienko, A. Klimovskaia, T. A. Kluyver, D. Mamchyk, T. Michelat, I. Mohacsi, A. Muennich, A. Parenti, R. Rosca, D. B. Rück, H. Santos, R. Schaffer, A. Silenzi, K. Wrona, C. Youngman, J. Zhu

- Key contributions from people that have left:
  - B. Heisen, M. Teichmann, K. Weger, J. Wiggins