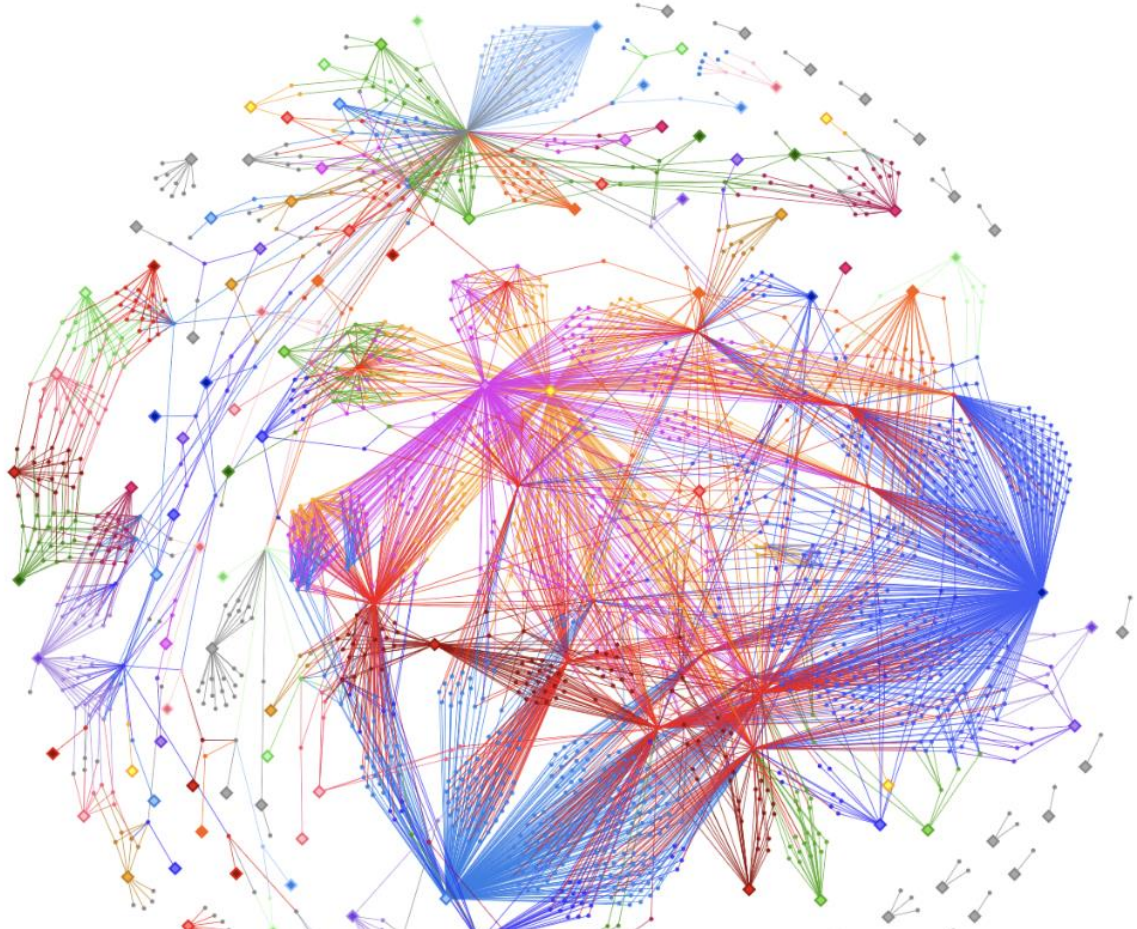
Karabo: Scientific Supervisory Control and Data Acquisition - Automation and High Data Rates

S. Hauf (steffen.hauf@xfel.eu)

8th EIROforum School on Instrumentation
May 13-17th, Garching, Germany

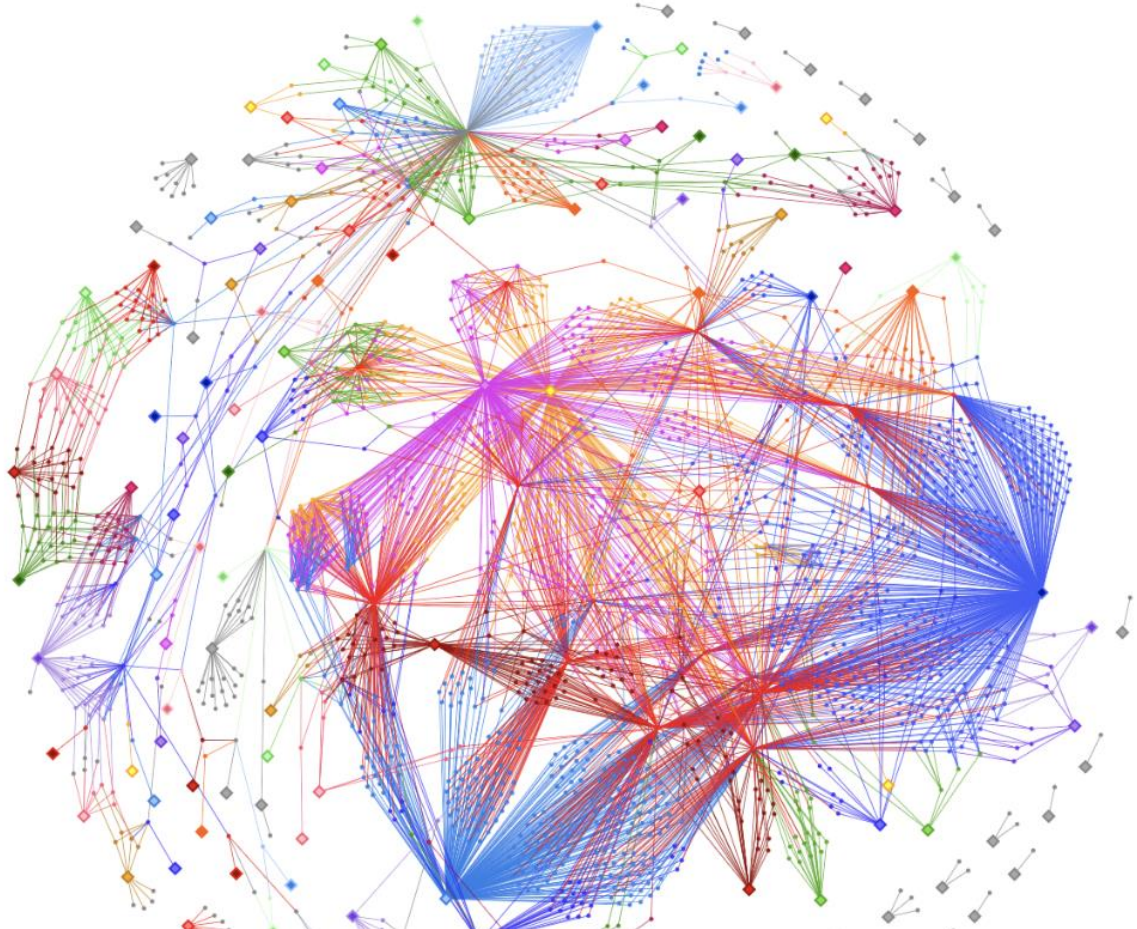


Tswana: **the answer**

Karabo: Scientific Supervisory Control and Data Acquisition - Automation and High Data Rates

S. Hauf (steffen.hauf@xfel.eu)

8th EIROforum School on Instrumentation
May 13-17th, Garching, Germany



The European XFEL – An X-ray Free Electron Laser

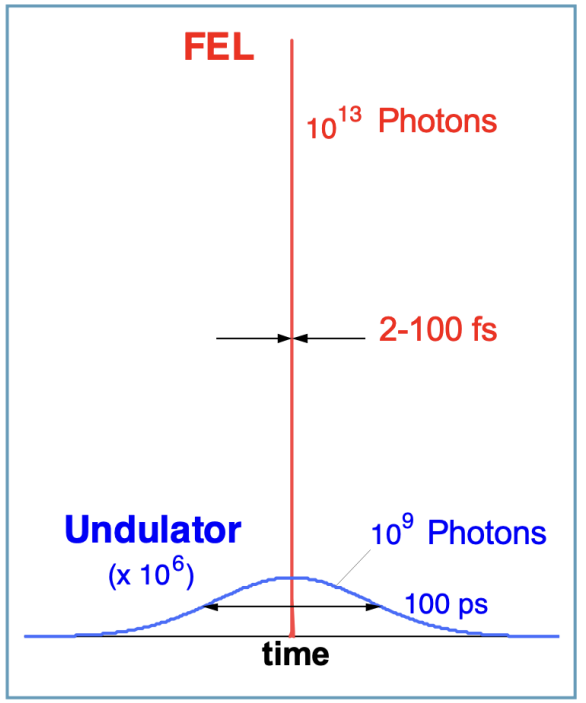
Undulator/Photon Tunnels

Experiment Hall in Schenefeld

Injector at DESY campus

Linear Accelerator
1.9 km
17.5 GeV

The European XFEL – An X-ray Free Electron Laser



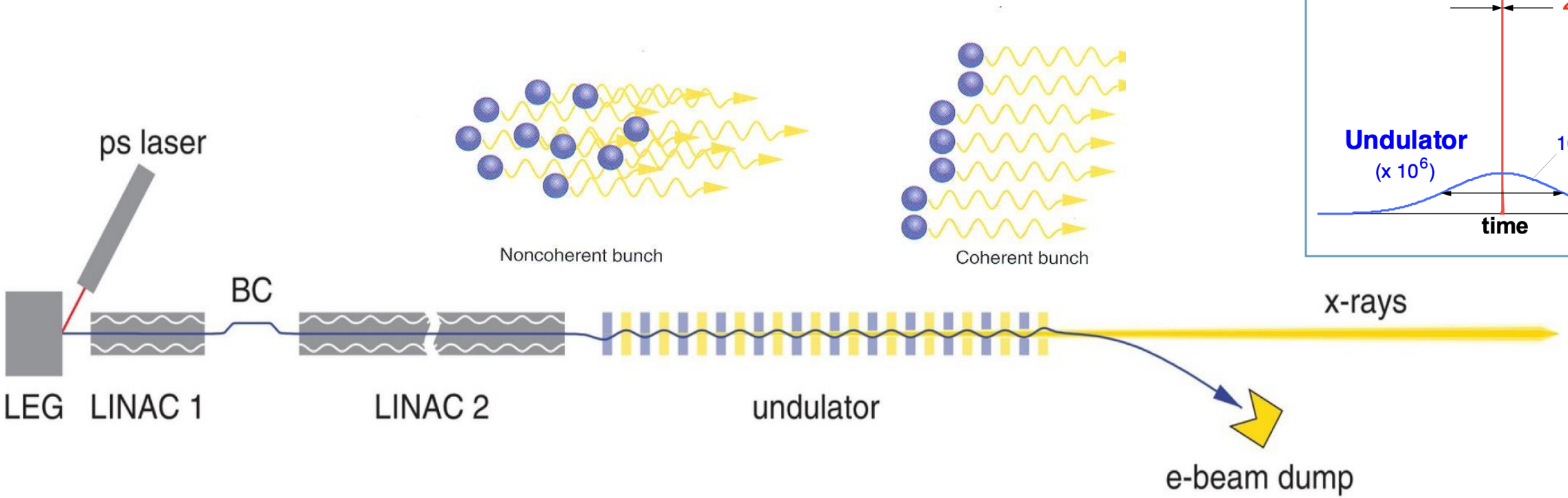


Figure 4.2 Schematic of XFEL facilities. Electron bunches are emitted from a low-emittance gun (LEG) irradiated by picosecond laser pulses. They are then accelerated in a short LINAC (LINAC 1), compressed longitudinally using one or more bunch-compressor magnet chicanes (BC), then further accelerated using a much longer LINAC (LINAC 2) before entering a long undulator, typically a few hundred metres in length. The SASE process along the undulator produces highly intense x-ray pulses with durations of the order of 50 fs. The electrons are deflected after the undulator using a bending magnet and subsequently dumped.

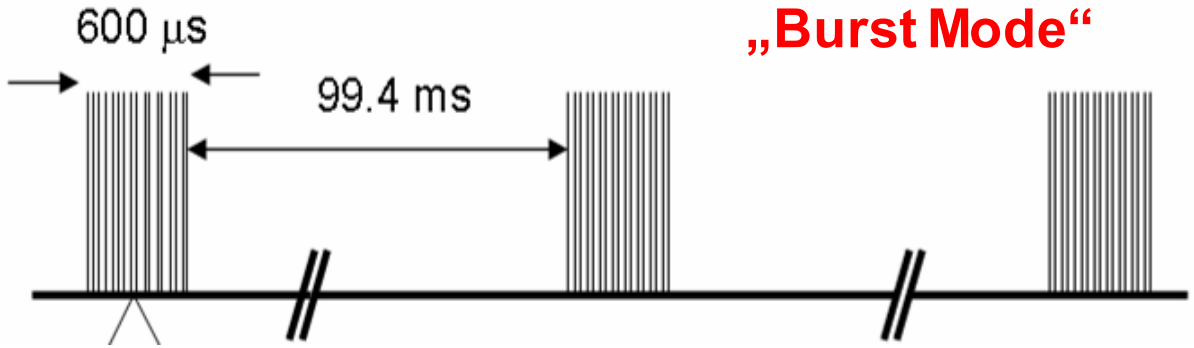
Courtesy R. Feidenhans'l

Light Sources around the World

Courtesy: **International Science Council, https://twitter.com/ISC/status/1246005021155655680/photo/1**

The European XFEL – Key Parameters

Parameter	Value
Electron Energy	8.5 – 17.5 GeV
Photon energy	0.26 - >25 keV
Pulse duration	2 – 100 fs
Seeding	As a special mode
# of pulses	27000 /s
# of FELs	3
# of instruments	7
Start of operation	2017

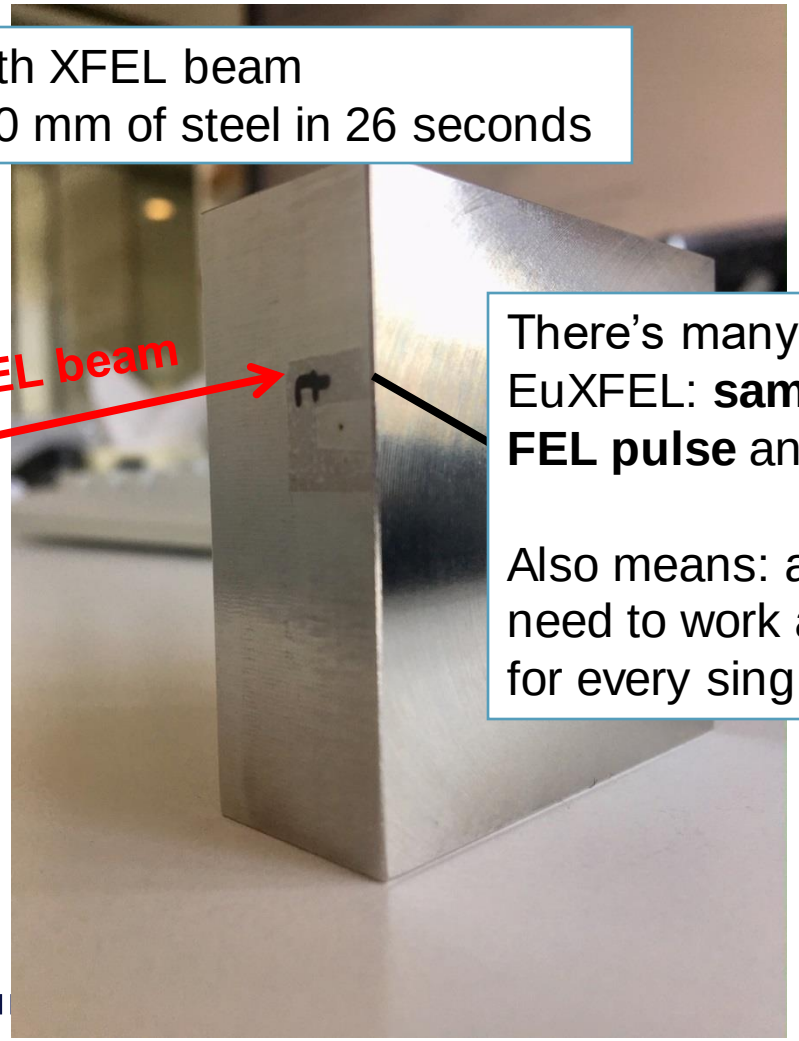


* Specific electron & x-ray beam delivery pattern
  * Follows from pulsed RF system
  * Trains of e-/x-ray pulses
  * Max. = 2.700 per train / 27.000 per sec

  * High average brilliance
  * Feedback & time and space stabilization
  * Dedicated pulse delivery

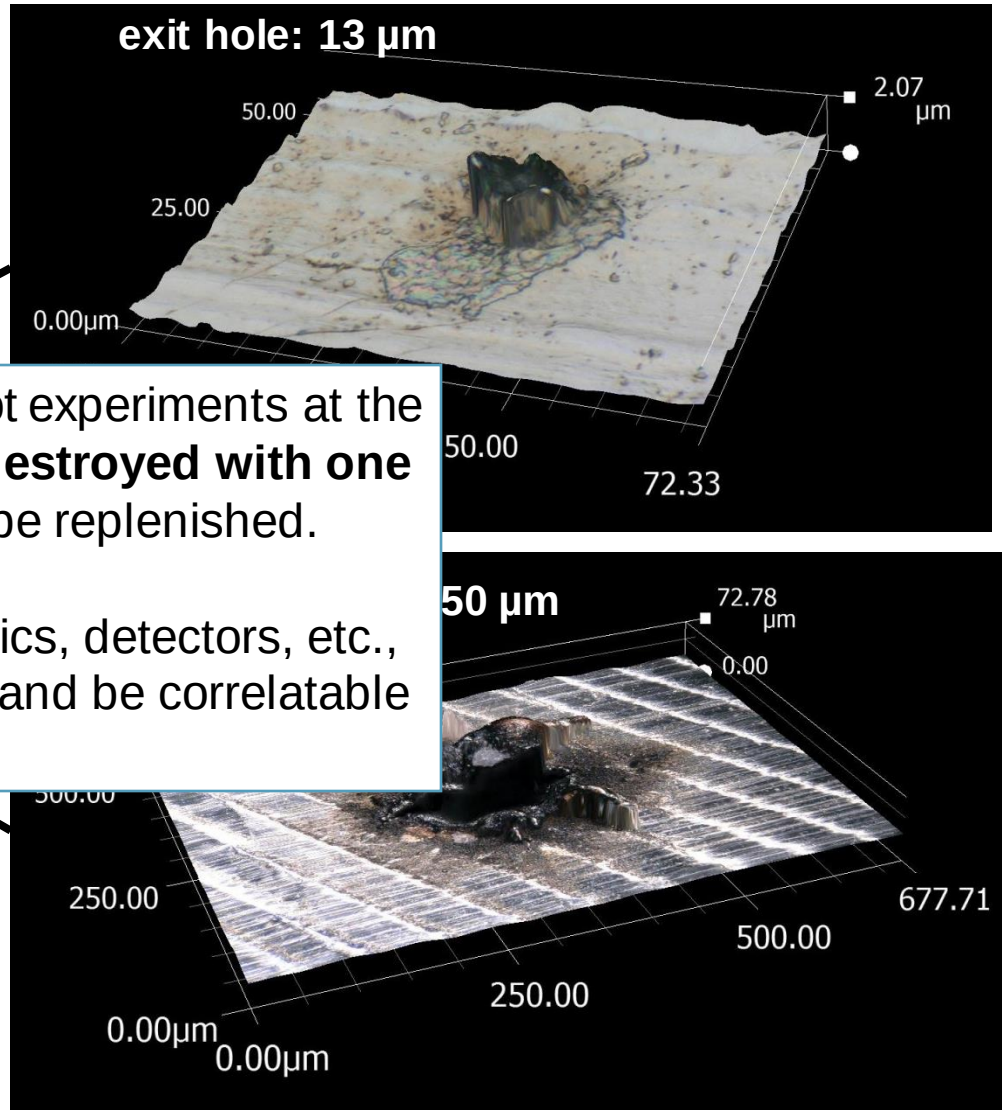
The European XFEL – An Exceptionally Strong X-ray Free Electron Laser

Drilling with XFEL beam
through 50 mm of steel in 26 seconds



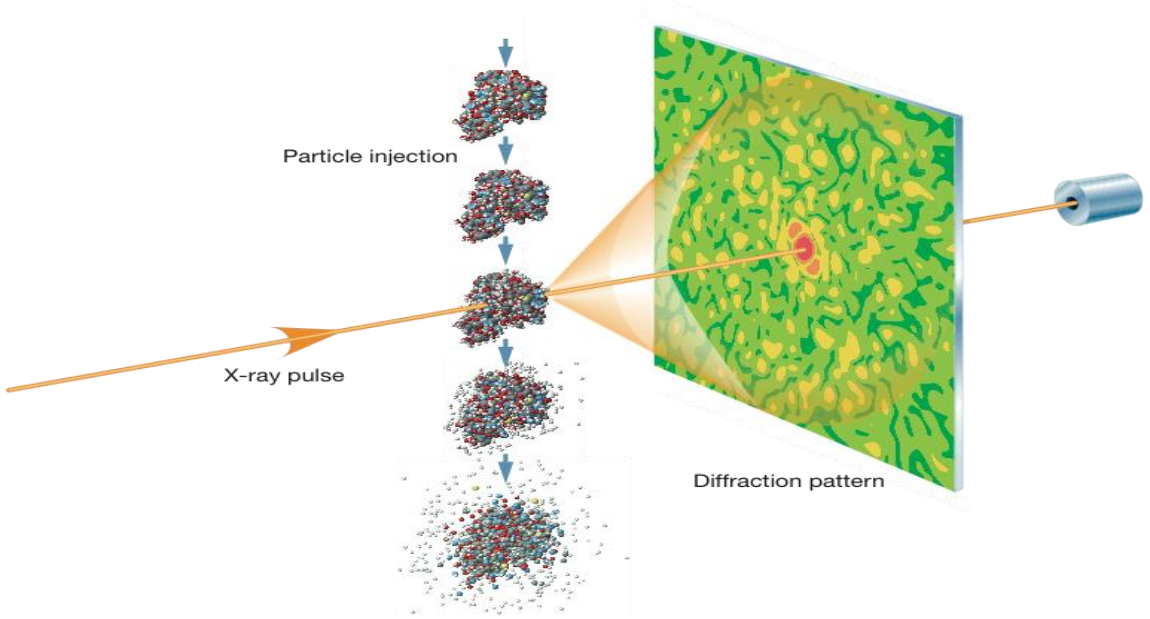
There's many single-shot experiments at the EuXFEL: **samples are destroyed with one FEL pulse** and need to be replenished.

Also means: all diagnostics, detectors, etc., need to work accurately and be correlatable for every single pulse



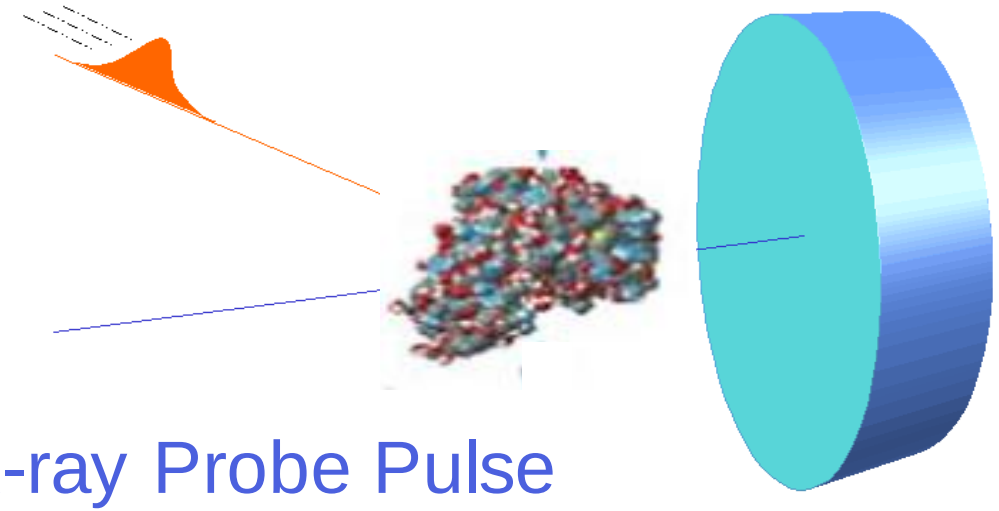
Courtesy Harald Sinn

The European XFEL – Science Case – Molecular Movies



Up to 3520 images/s
Up to ~ 15GB of data/s

Optical Laser Pump Pulse



X-ray Probe Pulse

10 times per second, with ps timing accuracy

Scientific instruments

SASE1

FXE (Femtosecond X-ray Experiments)

* Ultrafast dynamics of liquids and solid matter
* Combination of spec. & scat. techniques

SPB/SFX (Single Part., Bioimaging, & SFX)

- Coherent diffraction imaging from single part.
- Serial fs nano-crystallography

SASE2

MID (Materials Imaging & Dynamics)

- CDI from nano-structured samples
- XPCS of nanoscale dynamic

HED (High Energy Density science)

- Ultrafast dynamics of highly excited matter
- Combinations of scattering, diff. & spectroscopy

SASE3

SQS (Small Quantum Systems)

- Ultrafast dynamics of atoms, ions & clusters
- Combination of spec. & coh. scat. techniques

SCS (Spectroscopy & Coherent Scattering)

- Ultrafast dynamics of complex solids
- Combination of hr-inelastic spec. & coh.scattering

SXP (Soft X-ray Port)

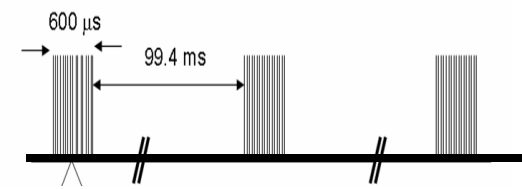
- Flexible port combining intense and tunable soft X-rays with versatile optical laser capabilities

The European XFEL – Complexity x 7 – Can change weekly!

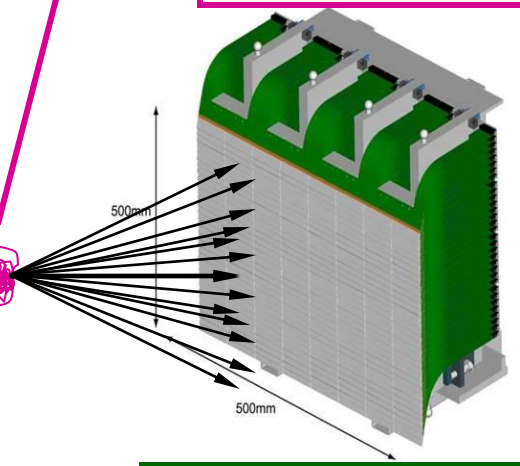
Controls

X-ray optics
Shape x-ray pulse
in space & spectrum

Sample delivery
Refresh & reposition
samples

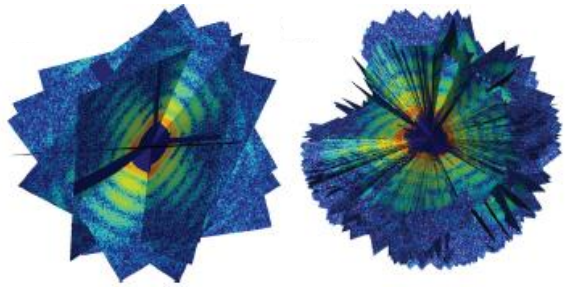


Photon diagnostics
Measure
x-ray pulse properties



Optical laser
Excite sample systems

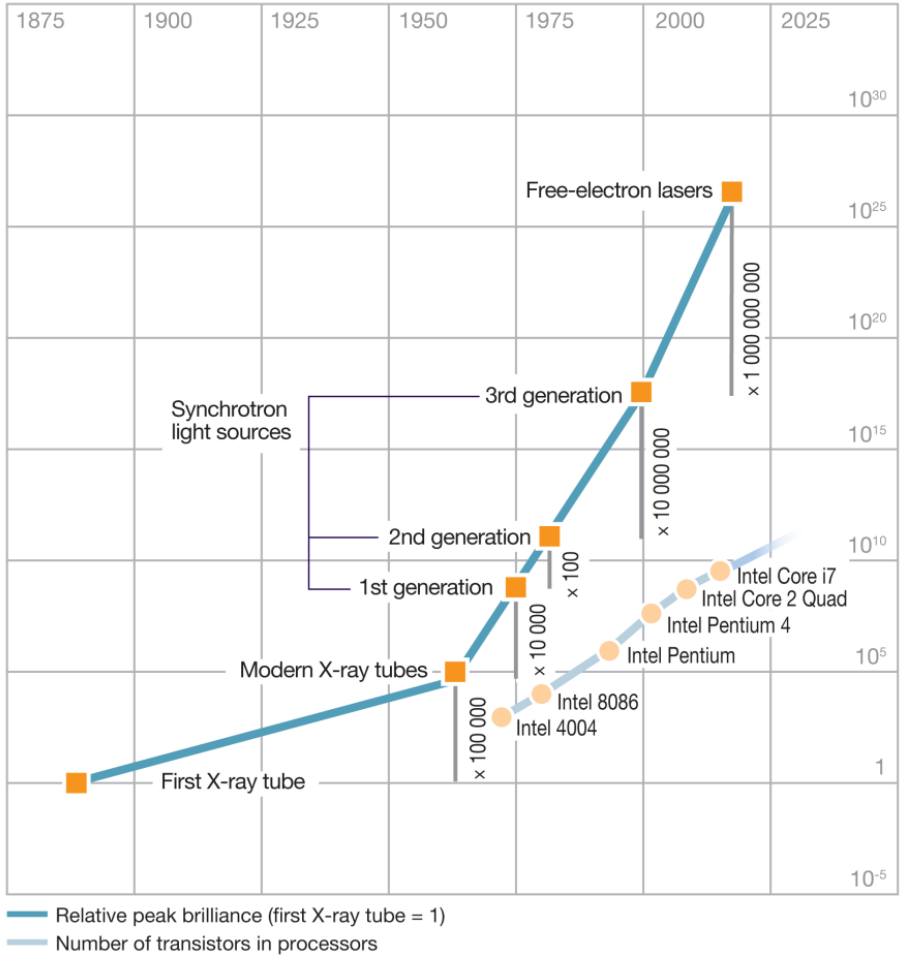
Detectors / DAQ
Take and store large number
of large data data sets



Data Analysis
Analyse immense amounts
of data

IT Infrastructure

The European XFEL - A Data Perspective



* The development of light source facilities has been faster than the increase in computer processing capacity (i.e., Moore's Law)

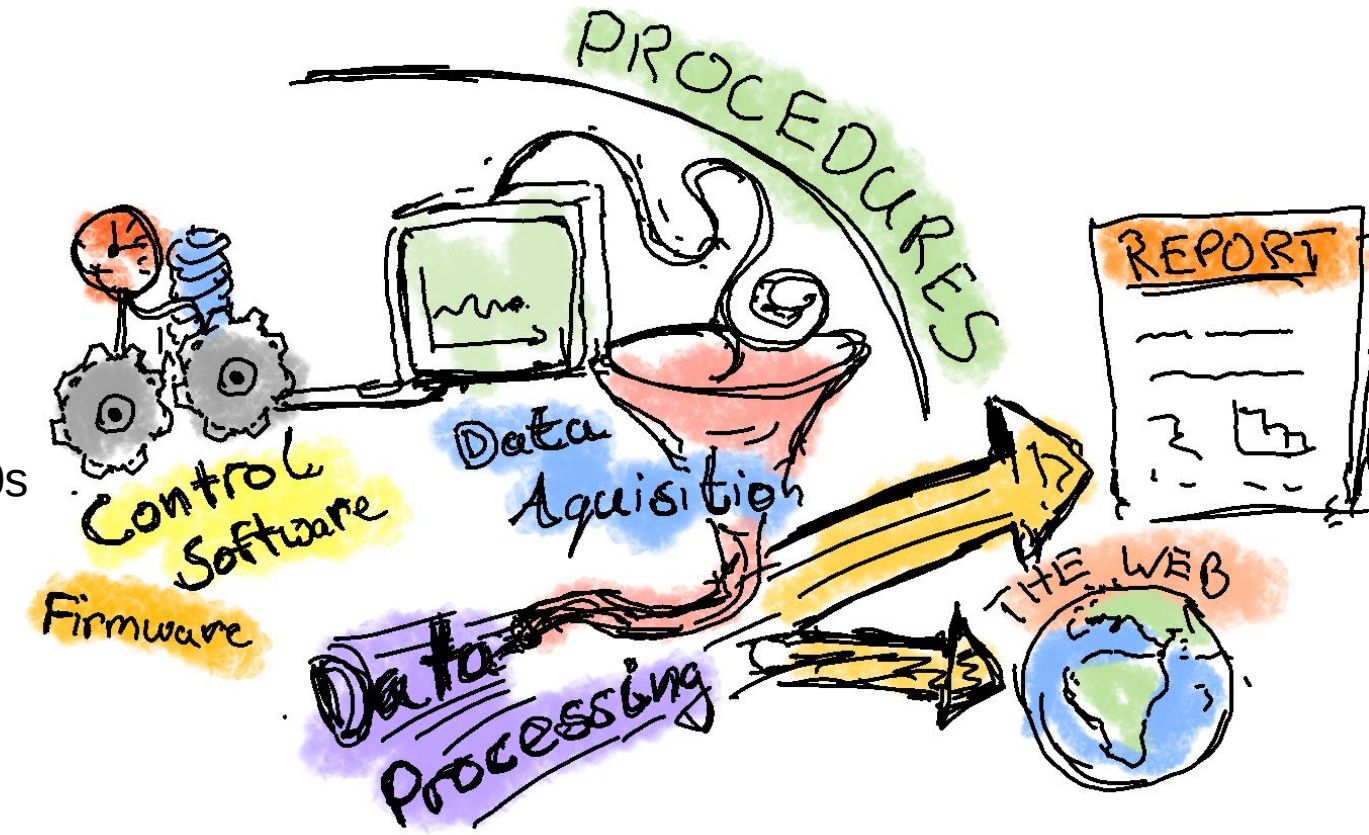
* We see this in the amount of data generated. For EuXFEL this can be multiple **PetaByte/week.** The Data Acquisition System is implemented in Karabo, as are the starting points of the online preview systems which support near-realtime processing of **>3kHz Mpixel images** .

The European XFEL in Software

* At the facility approx. 50 people (out of 450 employees) are tasked with writing software
* Our software can produce up to 5 PB of data per week
* Our software can use more than 10,000 cores, 100s of GPUs and terabytes of RAM simultaneously
* Given these numbers, sustainable software engineering thus can make a difference in the facility's environmental footprint
  * Not to mention that better software gives better scientific advances in fields like biophysics, and material science

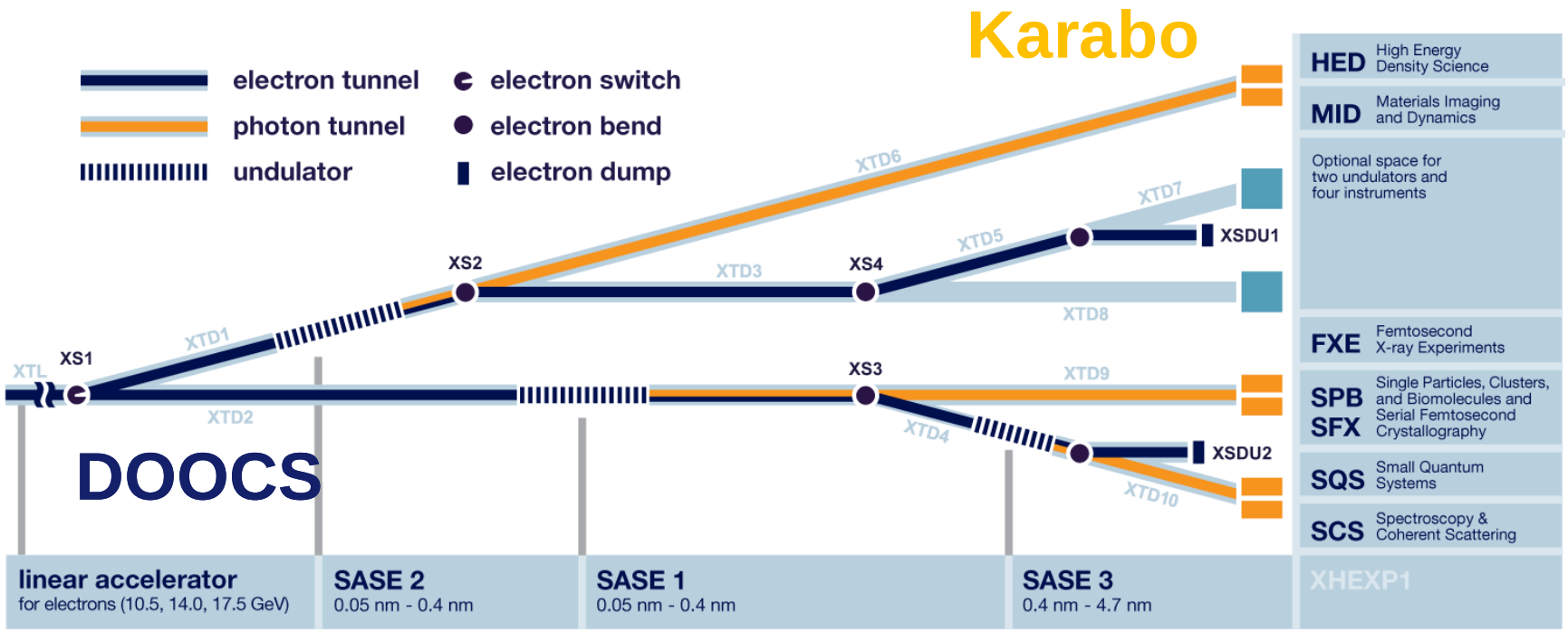
The European XFEL in Software

* At the facility approx. 50 people (out of 450 employees) are tasked with writing software
* Our software can produce up to 5 PB of data per week
* Our software can use more than 10,000 cores, 100s of GPUs and terabytes of RAM simultaneously
* Given these numbers, sustainable software engineering thus can make a difference in the facility's environmental footprint
  * Not to mention that better software gives better scientific advances in fields like biophysics, and material science



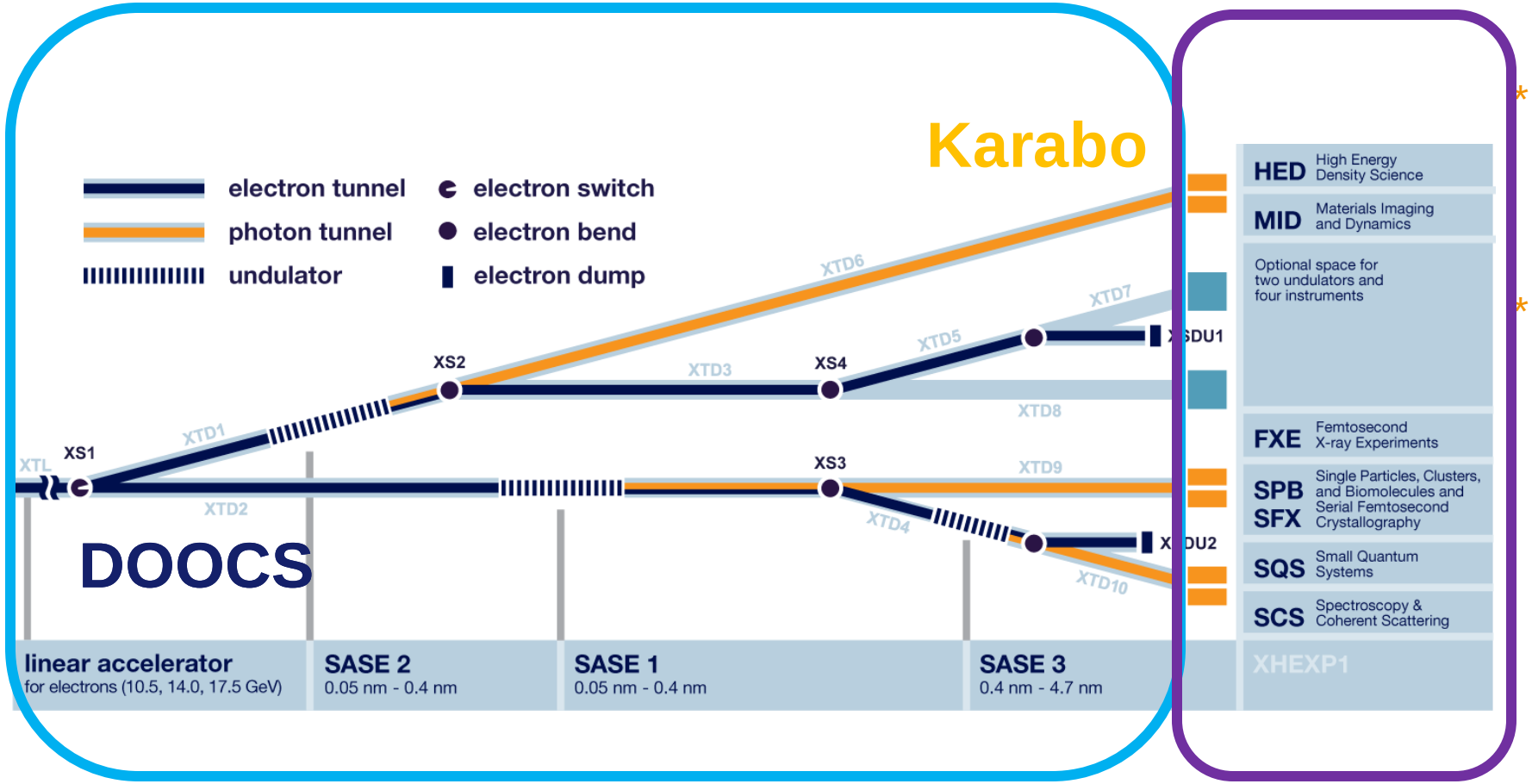
In total in EuXFEL repositories alone: >400k lines of code

Beamline layout and experiment stations



* Diagnostic data from the accelerator is important for experiment analysis
* Bridges between DOOCS and Karabo are collaboratively maintained with DESY and facilitate transfer between the two systems

Beamline layout and experiment stations



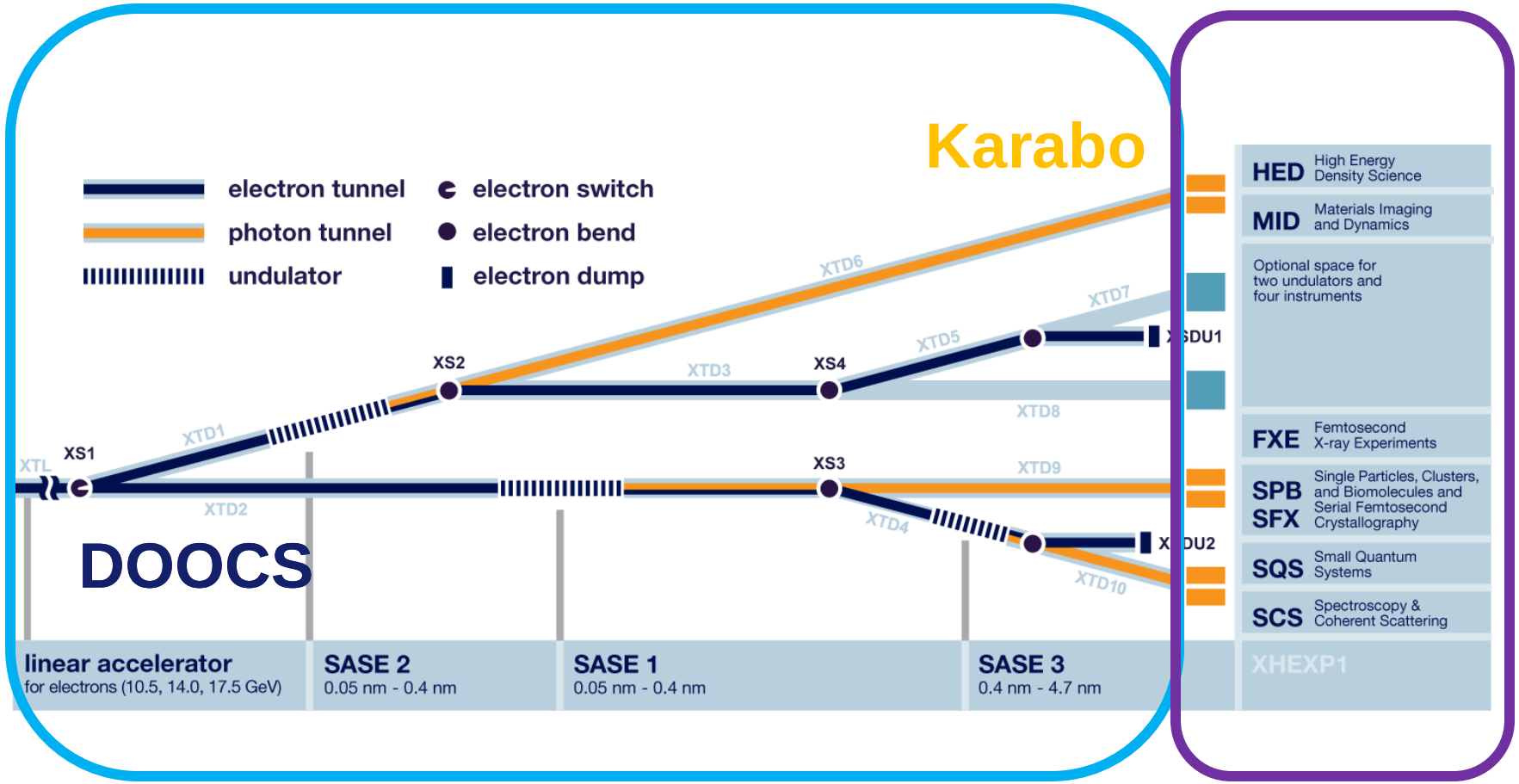
* Diagnostic data from the accelerator is important for experiment analysis

* Bridges between DOOCS and Karabo are collaboratively maintained with DESY and facilitate transfer between the two systems

Rather stable control and DAQ needs

Rather flexible control and DAQ needs

Beamline layout and experiment stations



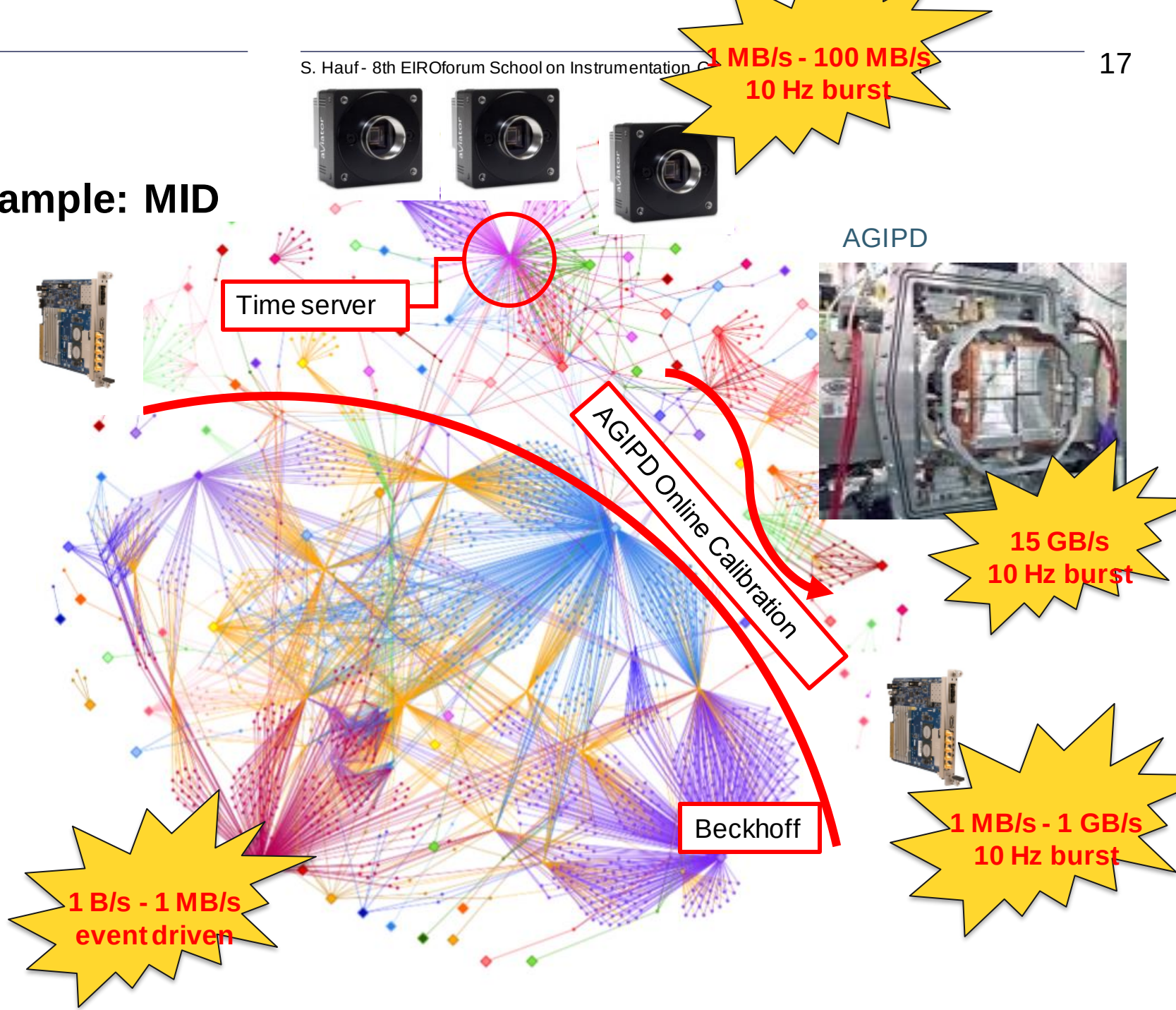
topic	# properties	# devices
LA1	26914.0	257.0
LA2	28103.0	250.0
LA3	32013.0	298.0
SA1	192656.0	1588.0
SA2	274426.0	2260.0
SA3	233216.0	1886.0
FXE	121905.0	905.0
HED	268688.0	1892.0
MID	258094.0	1789.0
SCS	205557.0	1444.0
SPB	267424.0	1872.0
SQS	306817.0	1954.0
Total	2215813.0	16395.0

Rather stable control and DAQ needs

Rather flexible control and DAQ needs

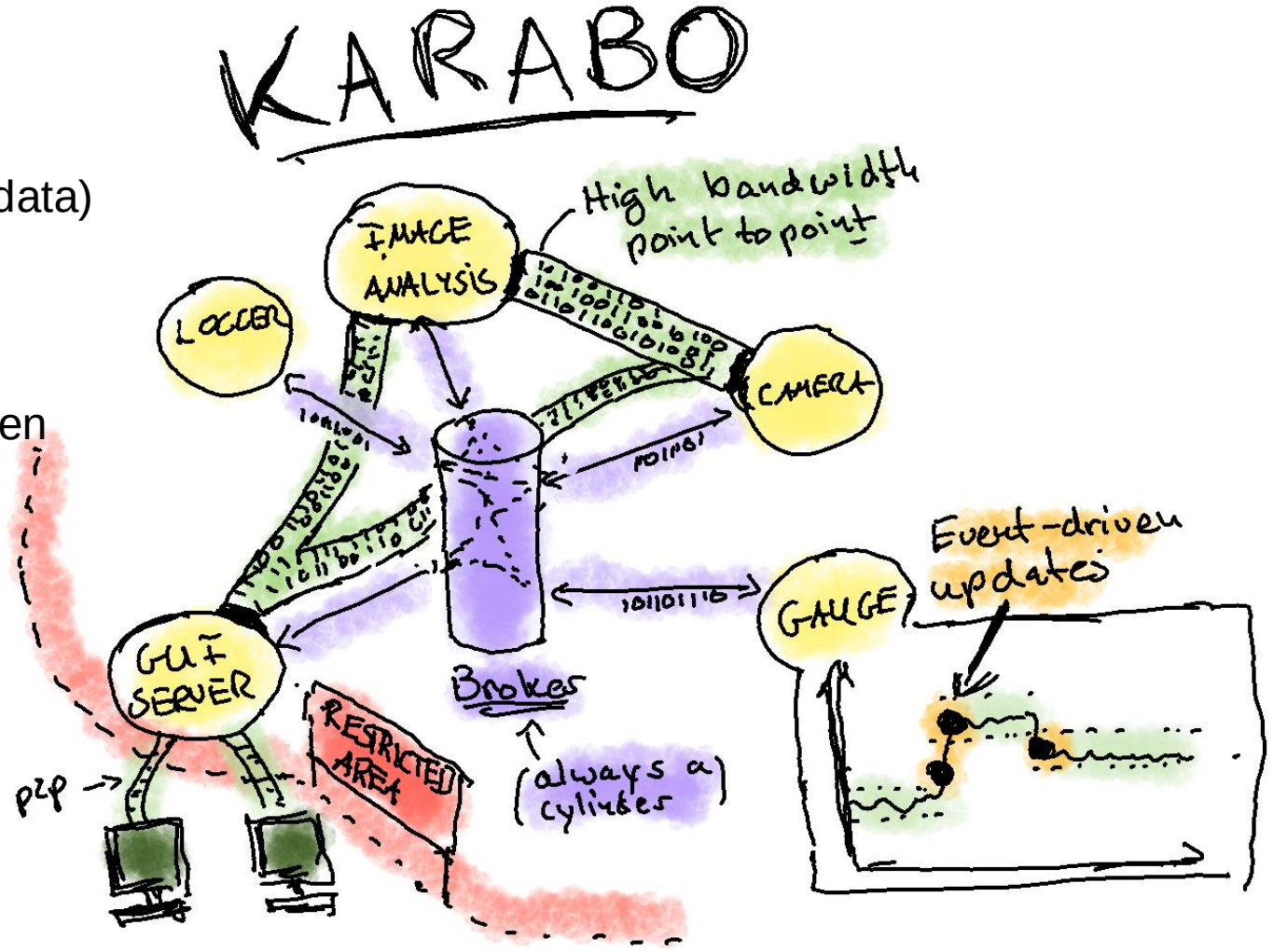
Karabo connects it all – Example: MID

* Key compoenents to look out for in all topics:
  * Timeserver: a central communication point for timing information
  * Bespoke MHz imaging detectors
  * Commercial cameras
  * Fast digitizers
  * Large Beckhoff loops, often interconnected via middlelayer devices and interlock conditions
  * Processing pipelines, e.g. detector calibration



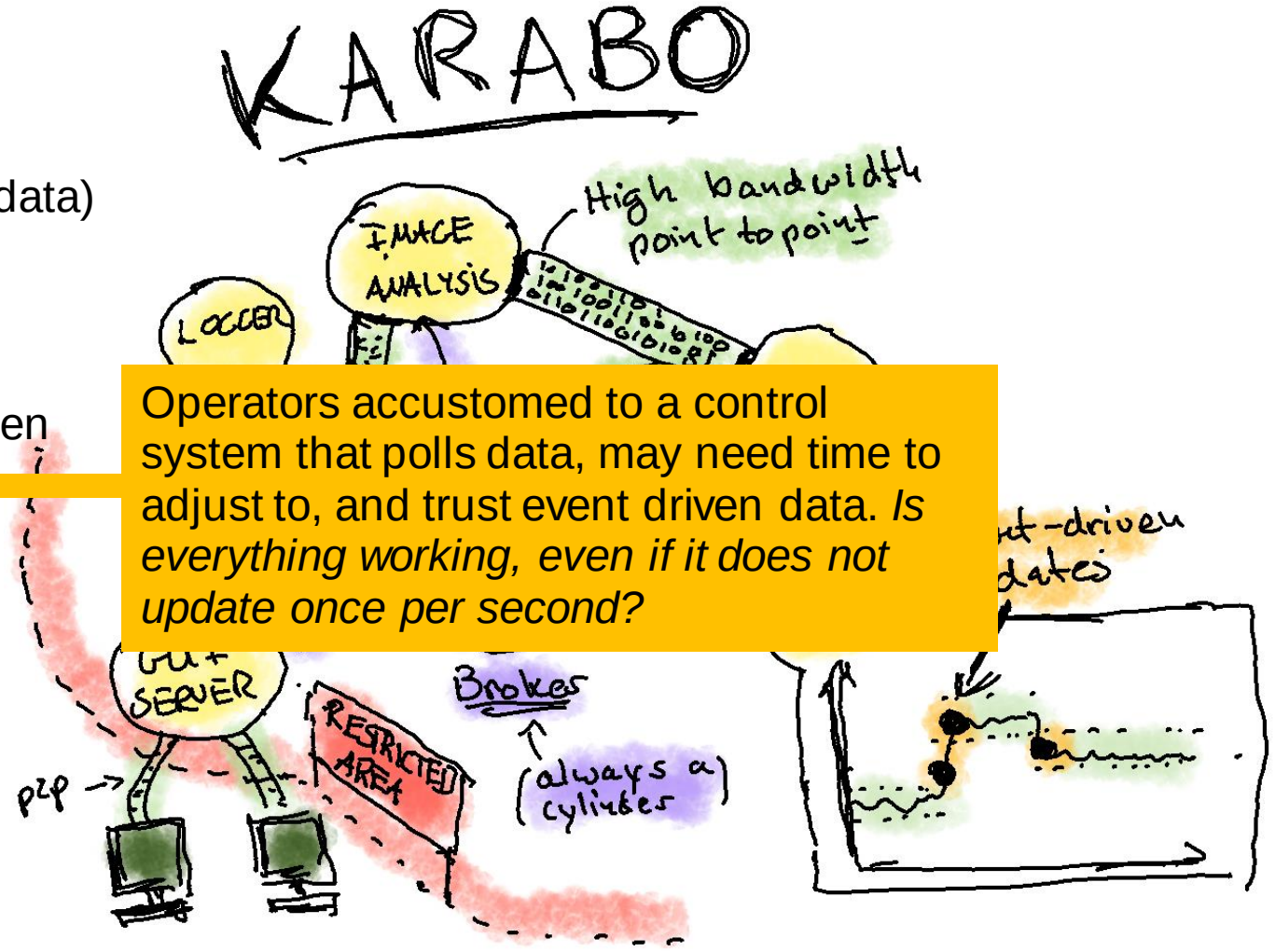
Karabo - Architecture

* Central Message Broker (Control and slow data)
  * AMQP on RabbitMQ
* Event driven:
  * Data propagates through the system when values change – push not poll
* Message driven:
  * Signal – Slot paradigm
  * Asynchronous core, synchronous convenience in middleware

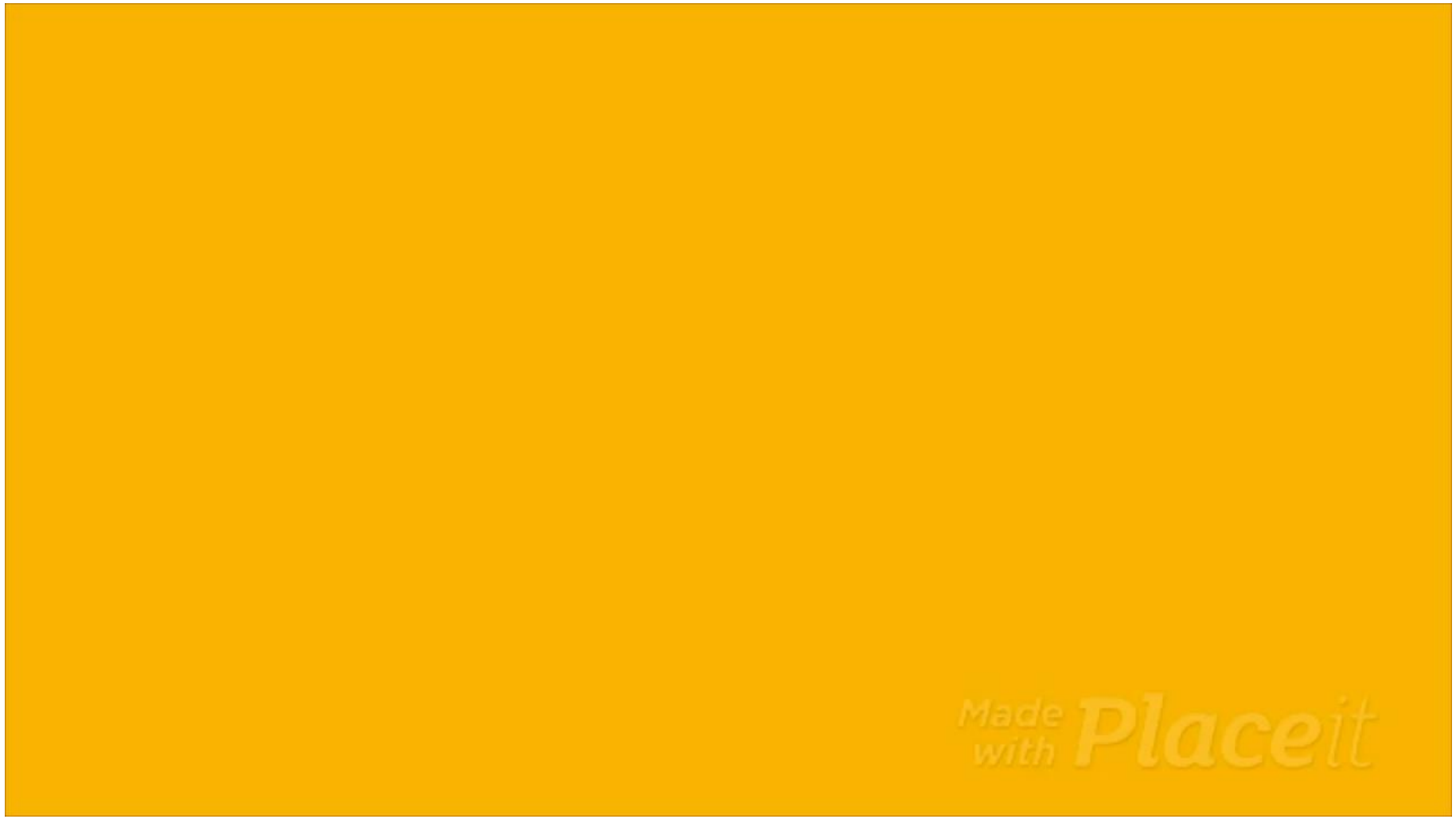


Karabo - Architecture

* Central Message Broker (Control and slow data)
  * AMQP on RabbitMQ
* Event driven:
  * Data propagates through the system when values change – **push not poll**
* Message driven:
  * Signal – Slot paradigm
  * Asynchronous core, synchronous convenience in middleware



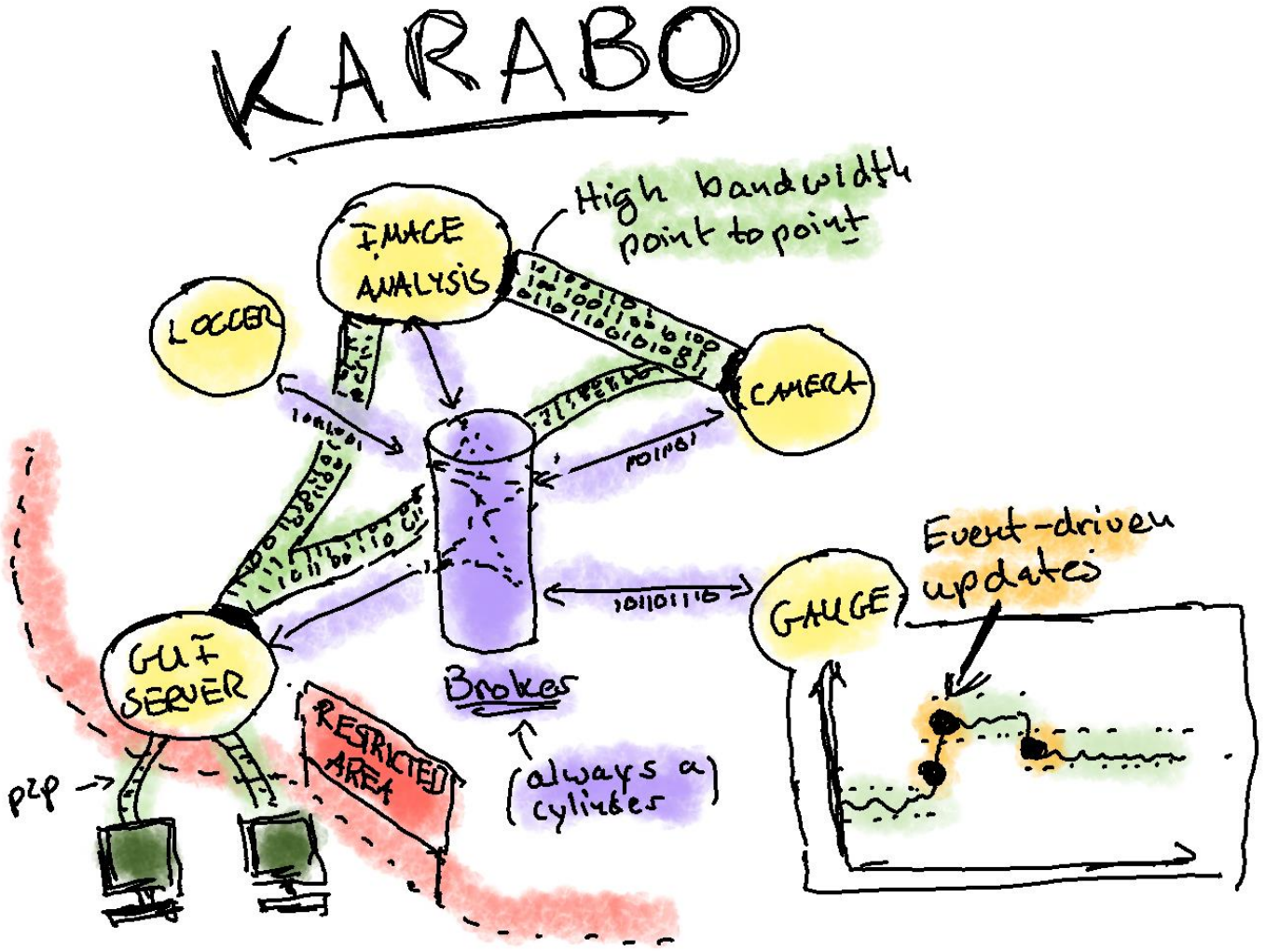
Operators accustomed to a control system that polls data, may need time to adjust to, and trust event driven data. *Is everything working, even if it does not update once per second?*



Pong-like visualization of control system messages over the network. Based on an idea by Mike Smith, using logstalgia, and content from placeit.net

Karabo - Architecture

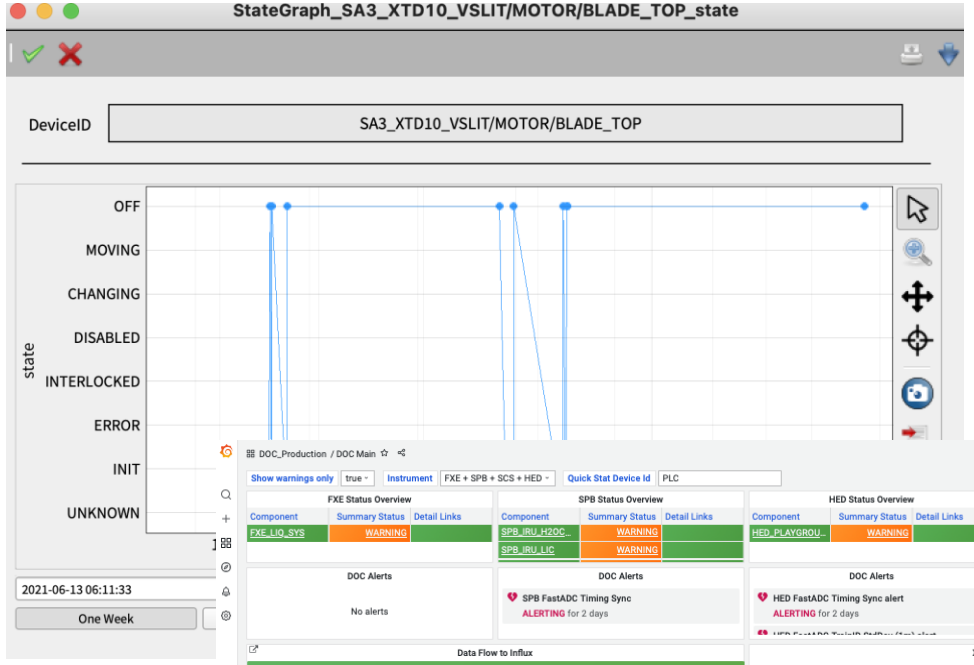
* pipeline (p2p) connections (scientific/large) d
  * Scatter/Gather/Copy/Distribute
  * Block/Drop on congestion
  * TCP
  * Also GUI Server – GUI client
  * Capable of saturating a 10G line
* Dynamic, discoverable topology
  * No central database instance
* GUI Server:
  * Gateway to the Control system



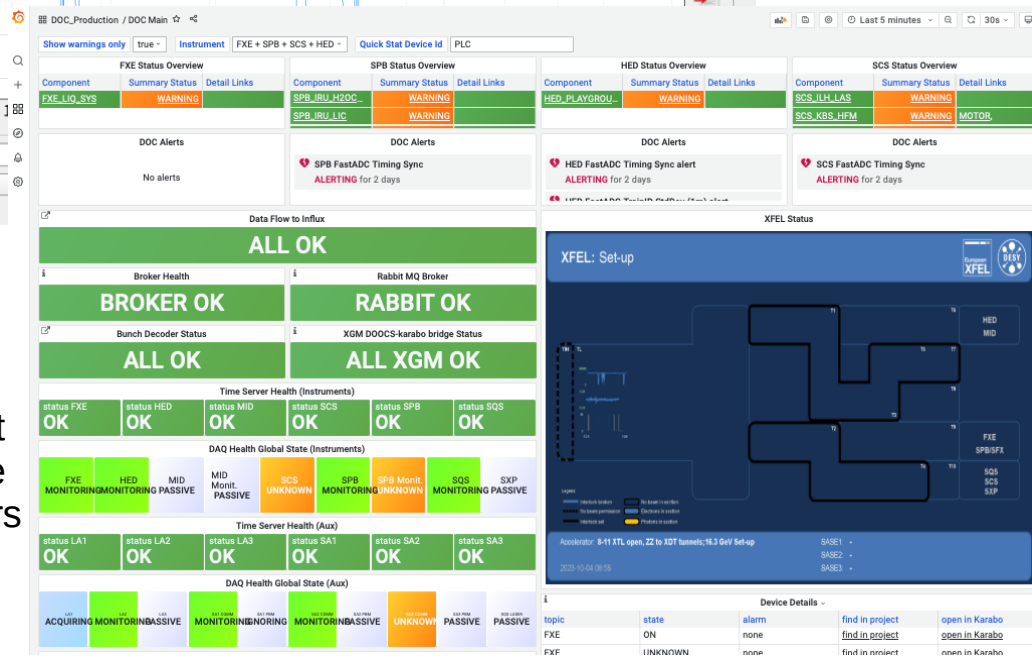
Metrics in Influx: > 240 Billion
Increase per month: ~ 10 Billion

Karabo – Data Logging using the Influx Time Series Database

* Datalogging vs. Data Acquisition
  * Datalogging is continous for slow (broker) data
    - ► It is done by default
    - ► For all devices
    - ► Internal data product for maintenence
  * Data Acquisition is „run“ based
    - ► Explicitely started
    - ► Includes large and fast data
    - ► Subselection of slow data
    - ► Data product for facility users
    - ► Multiple PB per week per instrument
* Karabo dataloggers
  * Proprietary text-based format
  * **Influx Time-series based**



Trendline in Karabo showing time evolution of device state.



Data Operation Center (DOC) main Grafana panel. Most data here is from the Karabo Influx loggers

Karabo – C++, Karabo Bound, and Karabo Middlelayer APIs

General	C++ API	Python Bound API	Middlelayer API
• Event driven • Asynchronous • Self-descriptive • Common, hierarchical data container supporting attributes on leafs: Karabo Hash • Binary and XML serialization • Extensible: core + “Devices“	• C++14 and Boost • Smart pointers • Template-heavy • Boost.asio • Eventloop based • Devices are threads on a single server • Aimed at high-performance devices	• Exposes C++ API via Boost.Python • Devices are separate processes • Was aimed at p2p heavy devices which e.g. need numpy • Not always pythonic	• Python asyncio • Decorators annotate Karabo structures • Emphasis on interaction with other devices • Pythonic

Karabo – C++, Karabo Bound, and Karabo Middlelayer APIs

General	C++ API	Python Bound API	Middlelayer API
• Event driven • Asynchronous • Self-descriptive • Common, hierarchical data container supporting attributes on leafs: Karabo Hash • Binary and XML serialization • Extensible: core + “Devices“	• C++14 and Boost • Smart pointers • Template-heavy • Boost.asio • Eventloop based • Devices are threads on a single server • Aimed at high-performance devices	• Exposes C++ API via Boost.Python • Devices are separate processes • Was aimed at p2p heavy devices which e.g. need numpy • Not always pythonic	• Python asyncio • Decorators annotate Karabo structures • Emphasis on interaction with other devices • Pythonic

Will use in the Automation Hands-On

Common Example: Concurrent Motion

From Bluesky documentation

```
from ophyd.sim import motor1, motor2

# Move motor1 to 1 and motor2 10 units in the positive direction relative
# to their current positions. Wait for both to arrive.
RE(bps.mvr(motor1, 1, motor2, 10))
```

Karabo Middlelayer

```
[2]: motors = [await connectDevice(motorId) for motorId in motorIds]
...: for device, position in zip(motors, positions):
...:     device.targetPosition = position
...:     futures.append(device.move())
...:
...: await gather(*futures)
...: await waitUntil(lambda: all(dev.state != State.MOVING for dev in motors))
```

Other systems:
- Bliss e.g. motor_group
- Sardana: moveMultiple
- ….
- Karabacon: MotorGroup

Common Example: Concurrent Anything…

Motion

```
[2]: motors = [await connectDevice(motorId) for motorId in motorIds]
...: for device, position in zip(motors, positions):
...:     device.targetPosition = position
...:     futures.append(device.move())
...:
...: await gather(*futures)
...: await waitUntil(lambda: all(dev.state != State.MOVING for dev in motors))
```

Power Supplies

```
...:
...:     for device, voltage in zip(mpodGroups, voltage):
...:         device.voltage = voltage
...:         futures.append(device.on())
...:
...:     await gather(*futures)
...:     await waitUntil(lambda: all(dev.state == State.ON for dev in devices))
```

Instantiation

```
...:     for serverId, classId, deviceId, config in offlineDevices:
...:
...:         futures.append(instantiate(serverId, classId, deviceId, config))
...:
...:     await gather(*futures)
...:
```

Common Example: Concurrent Anything…

Motion

```
[2]: motors = [await connectDevice(motorId) for motorId in motorIds]
...: for device, position in zip(motors, positions):
...:     device.targetPosition = position
...:     futures.append(device.move())
...:
...: await gather(*futures)
...: await waitUntil(lambda: all(dev.state != State.MOVING for dev in motors))
```

Karabo Middlelayer prefers using Python ”primitives” where possible over domain specific extensions.

→ Learn once – use often

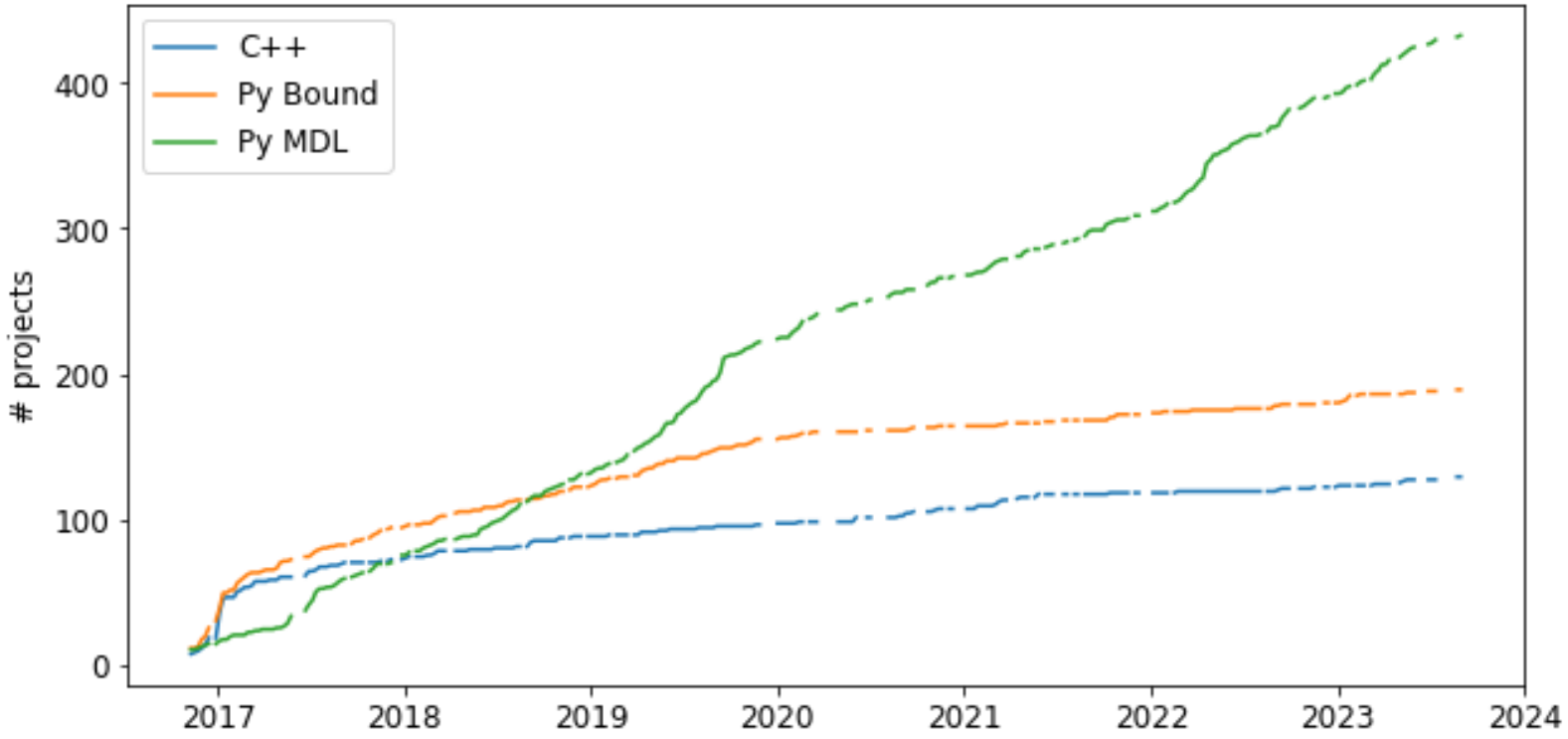
```
...:         futures.append(device.on())
...:
...:     await gather(*futures)
...:     await waitUntil(lambda: all(dev.state == State.ON for dev in devices))
```

Instantiation

```
...:     for serverId, classId, deviceId, config in offlineDevices:
...:
...:         futures.append(instantiate(serverId, classId, deviceId, config))
...:
...:     await gather(*futures)
...:
```

The Karabo Ecosystem – Usage of the three APIs

* Middlelayer API frequently has the most expressive syntax coupled with shortest “time-to-market”.
* C++ and Python Bound are actively maintained and new devices are still being implemented, especially in high-performance fields.

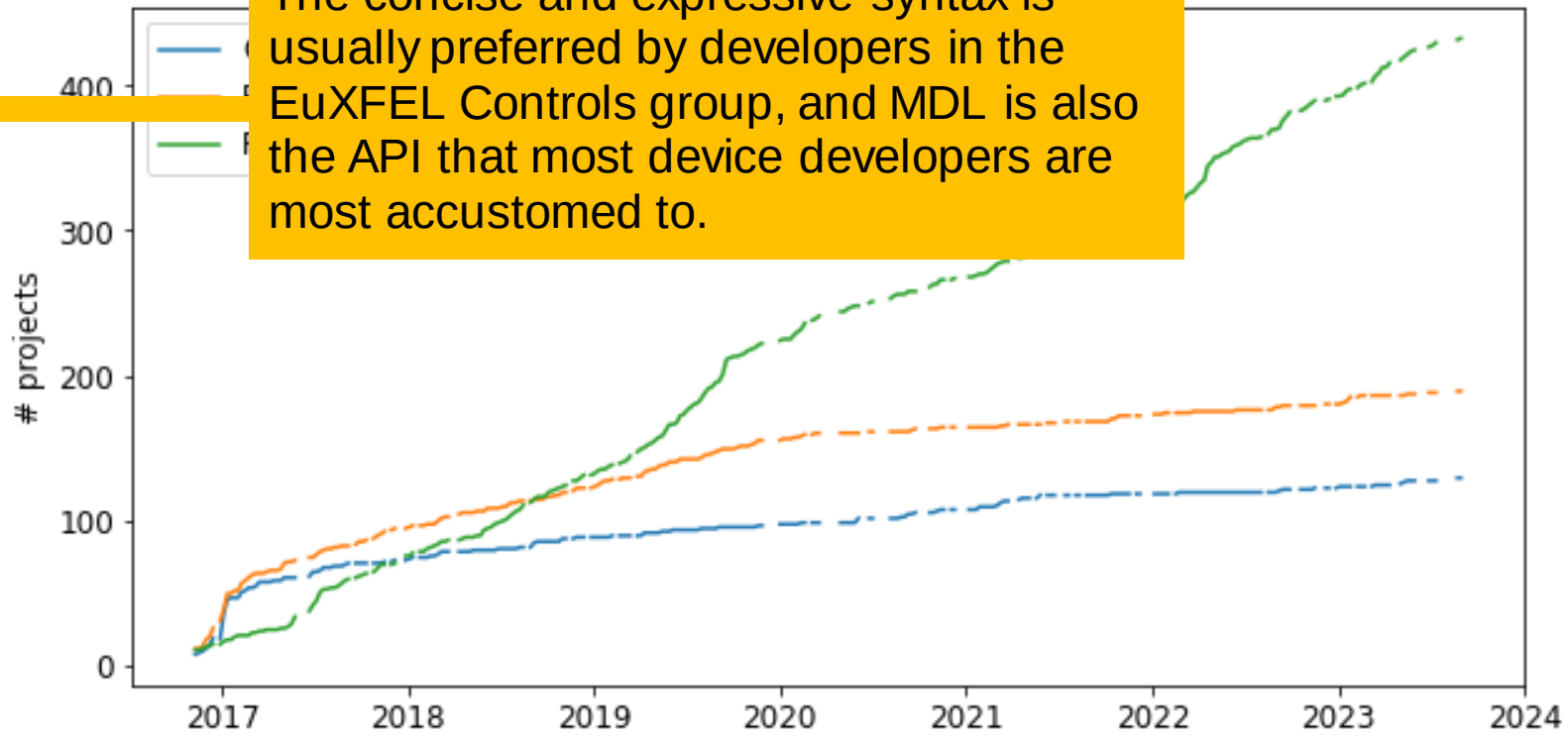


The Karabo Ecosystem – Usage of the three APIs

* Middlelayer API frequently has the most expressive syntax coupled with shortest “time-to-market”.

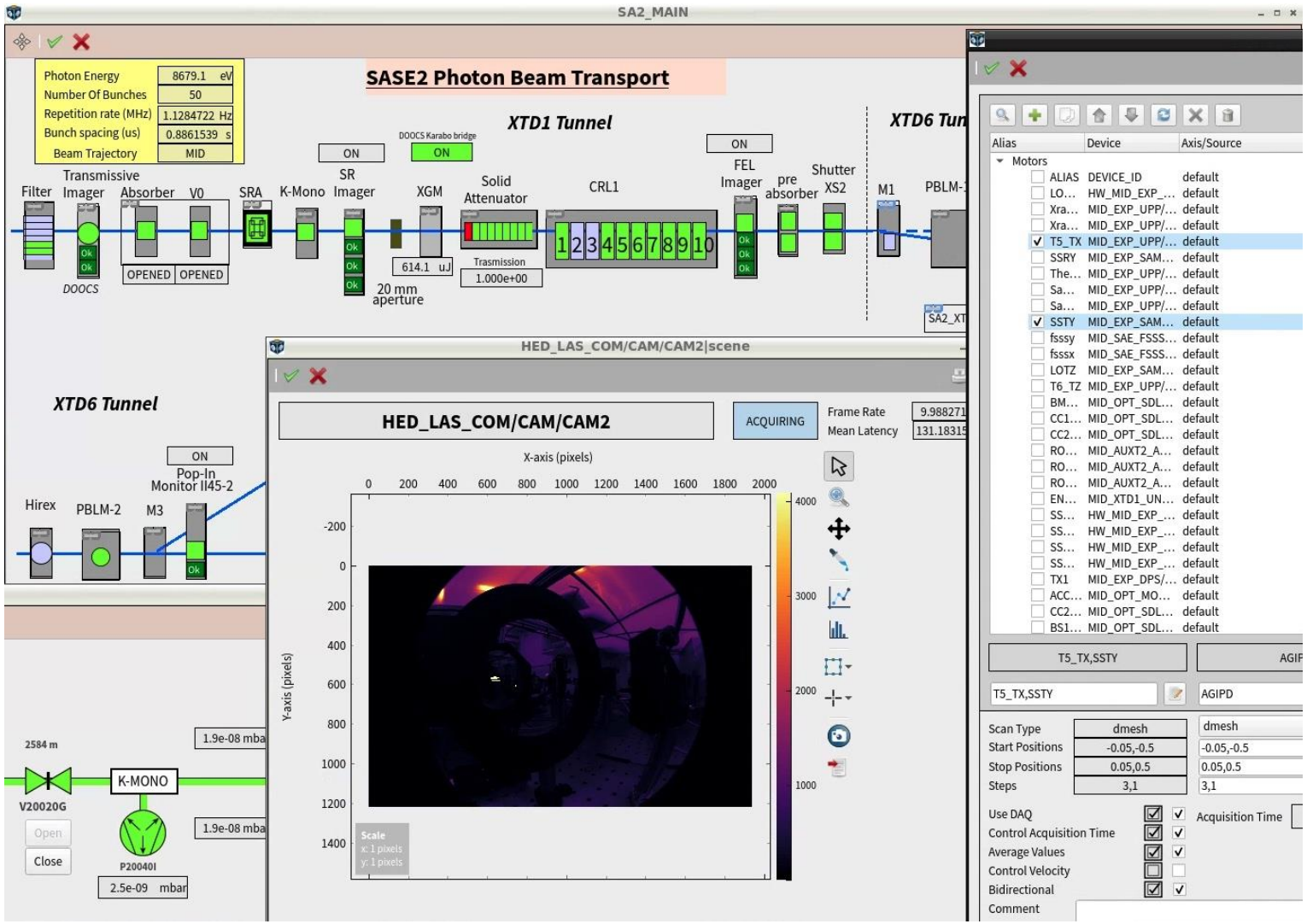
* C++ and Python Bound are actively maintained and new devices are still being implemented, especially in high-performance fields.

The concise and expressive syntax is usually preferred by developers in the EuXFEL Controls group, and MDL is also the API that most device developers are most accustomed to.



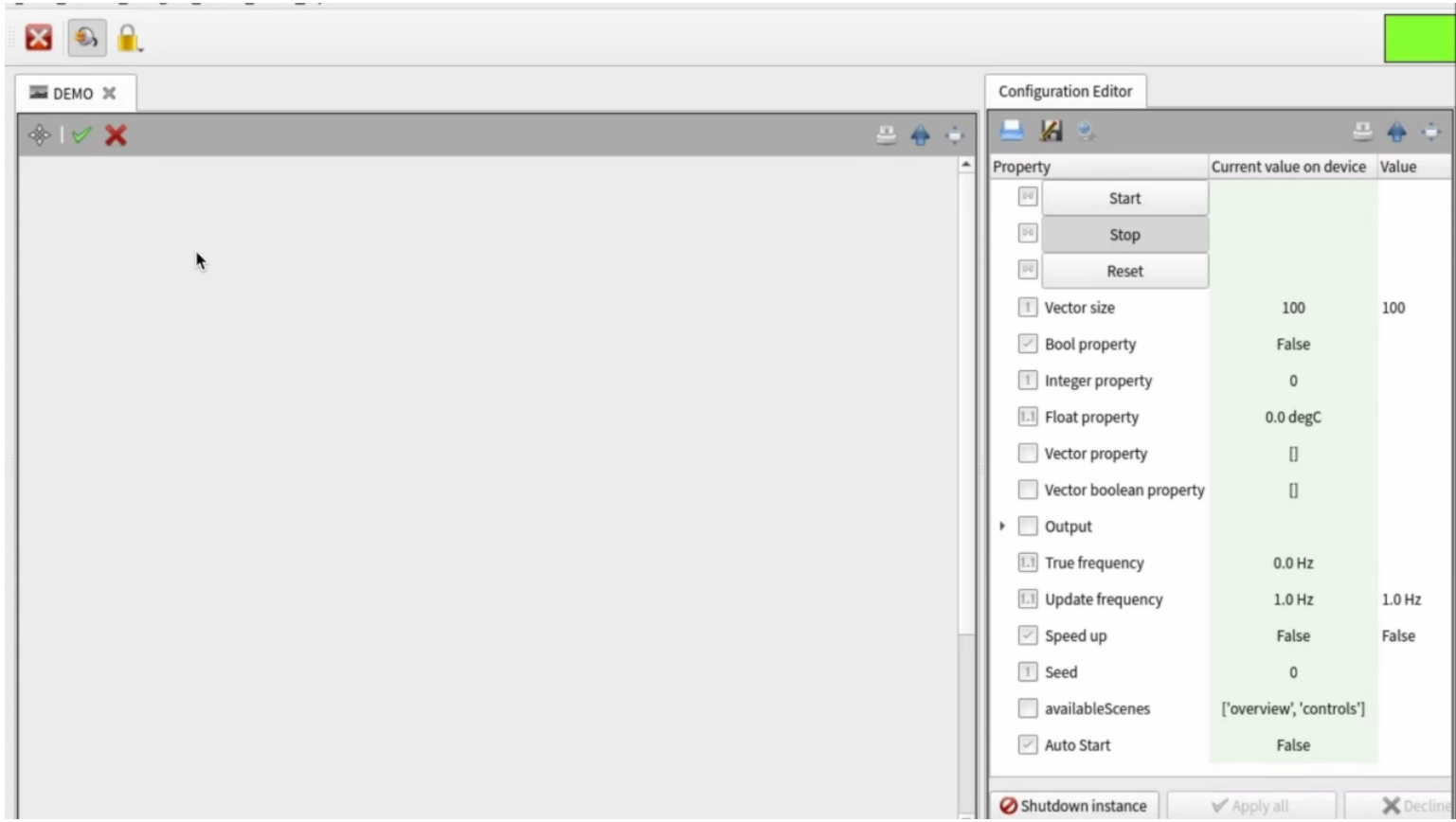
Karabo - The Karabo GUI

* Separate Python Package, well matched to the framework
* PyQt5
* Connects to Karabo via the GUI-server (tcp, p2p)
* Extensible via „gui-extensions“
* **Distinguishing features:**
  * **GUI scene builder**
  * **Projects to logically group devices, scenes and macros**



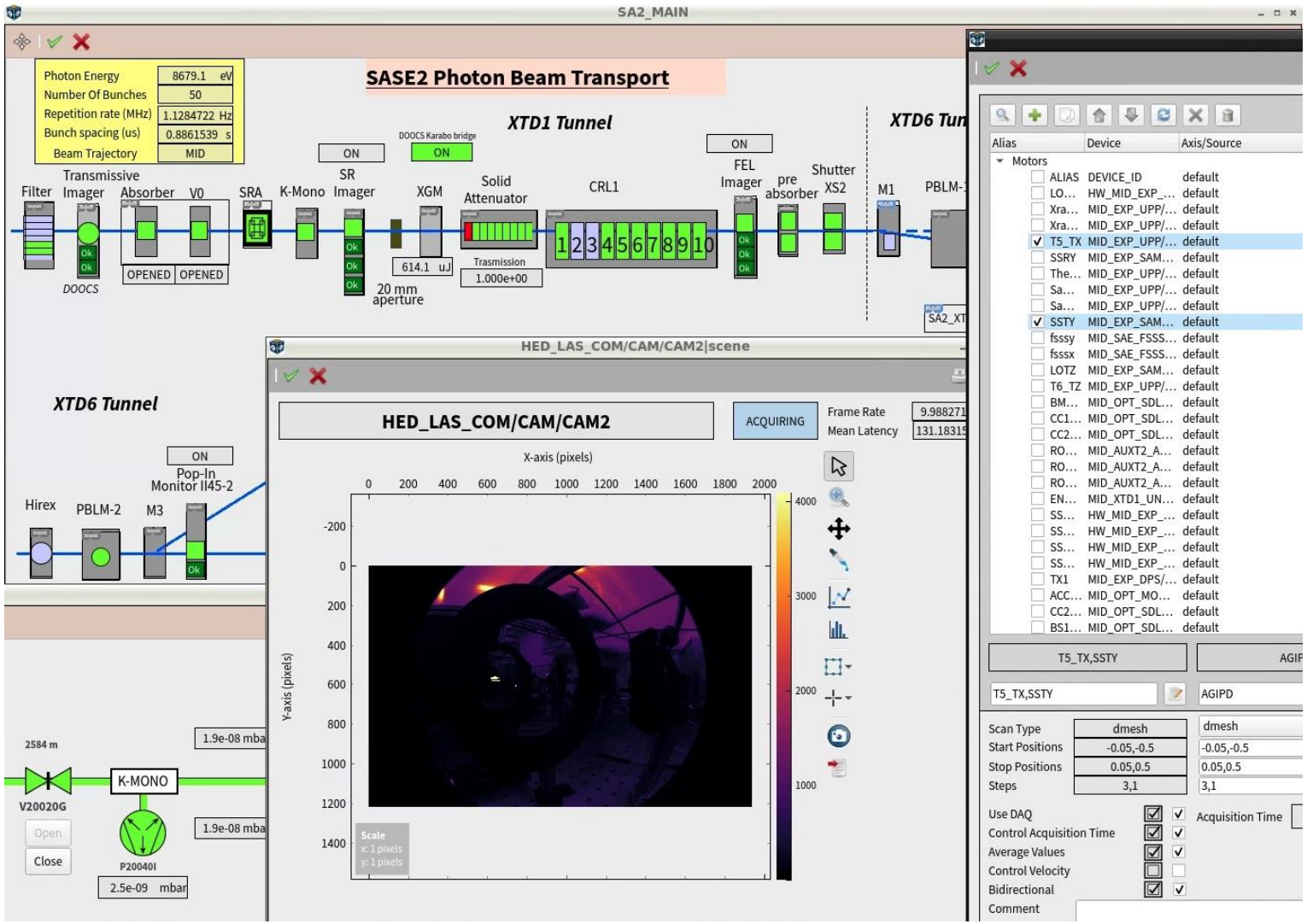
Karabo - The Karabo GUI

* Separate Python Package, well matched to the framework
* PyQt5
* Connects to Karabo via the GUI-server (tcp, p2p)
* Extensible via „gui-extensions“
* **Distinguishing features:**
  * **GUI scene builder**
  * **Projects to logically group devices, scenes and macros**



Karabo - The Karabo GUI

* Separate Python Package, well matched to the framework
* PyQt5
* Connects to Karabo via the GUI-server (tcp, p2p)
* Extensible via „gui-extensions“
* **Distinguishing features:**
  * **GUI scene builder**
  * **Projects to logically group devices, scenes and macros**



Karabo Hardware Devices – an incomplete list of what is supported

Core Services

- Data Logging
- Data Acquisition
- Gui Server
- Alarms & Notification
- Recovery Portal
- Motor Configurator
- Scan Tool

Motion

- Beckhoff MC2*
- Smaract
- Hexapod
- NanoCube
- …

Commercial Cameras

- Basler via Aravis
- Genicam
- GreatEyes
- PI MTE3
- Andor Zylar, …
- Shimadzu HPvx

X-ray Detectors

- Jungfrau
- Gotthard
- Epix
- AGIPD
- LPD
- DSSC
- Timepix3
- …

Scopes and …meters

- OceanOptics
- GENTEC Tetronix
- LeCroy
- MCS Beam Stab.
- …

Digitizers

- SP devices ADQ 412
- SP devices ADQ 7
- SP devices ADQ 14
- FastADC

Power Supplies

- Wiener MPOD
- Keithley
- Agilant

Vacuum Components*

- Adixon
- Pfeiffer
- Infinicon
- Agilant

Chillers & Thermo Contr.*

- Huber
- K2
- Julabo
- Keithley
- Lakeshore
- …

Bridging

- SCPI
- DOOCS
- EPICS (in progress)
- Tango (in progress)

* via Beckhoff PLC integration

Karabo: Ongoing Developments

Move to RabbitMQ and AMQP ✓

RabbitMQ

RDMA – 80% IB link speed

WebUI

https://github.com/European-XFEL/Karabo

ZULIP ✓

Elog Integration

python Package Index

MDL on PyPi

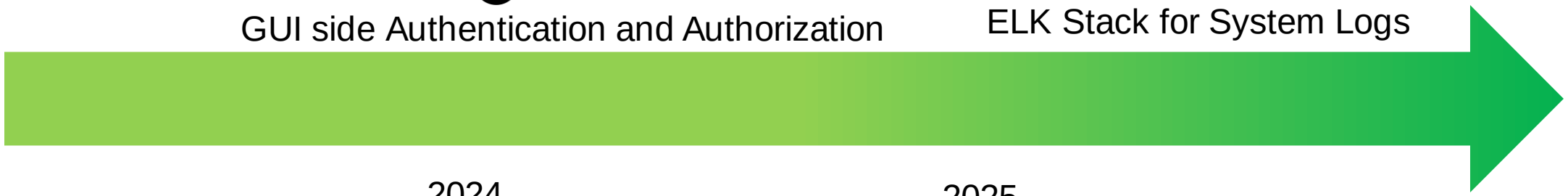
Authenticating & Authorizing Web API

{REST:API}*

elasticsearch

GUI side Authentication and Authorization

ELK Stack for System Logs

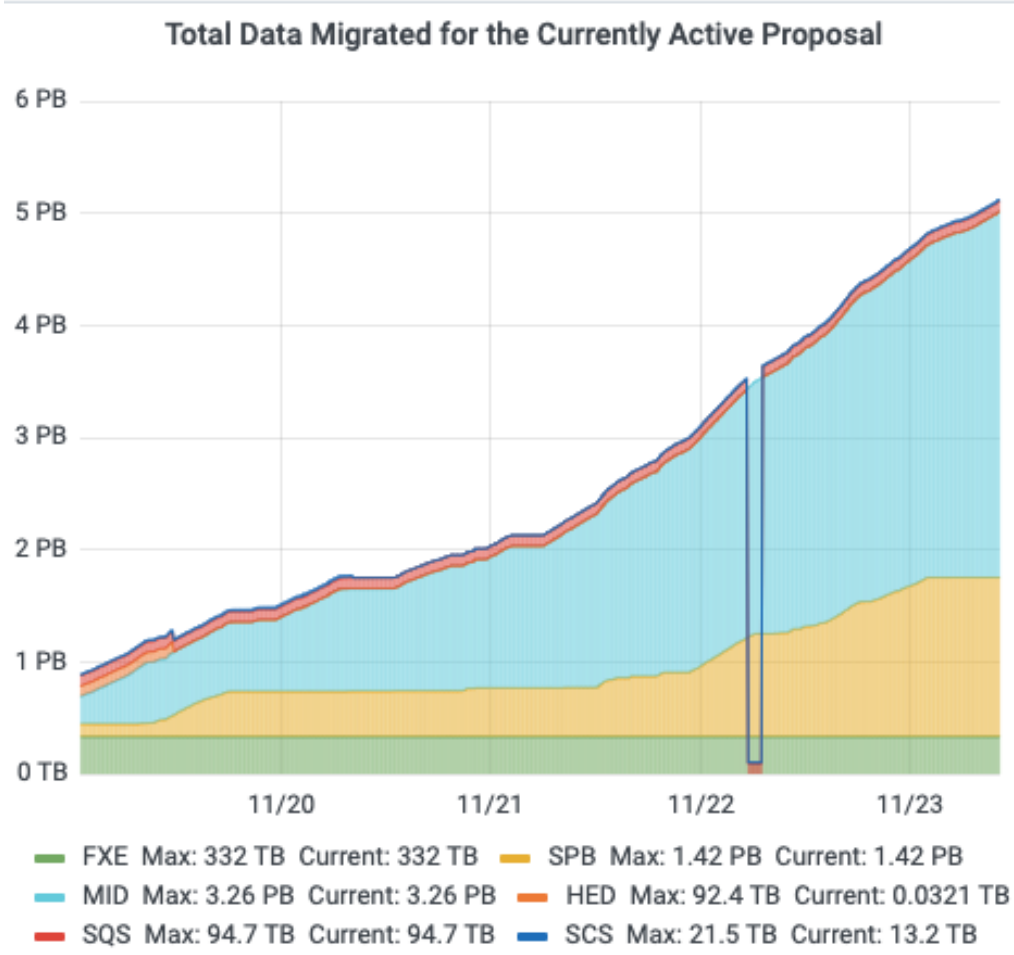


2024

2025

Feeling the Data Pressure – Investments in Data Reduction

Data Produced in CW46 2021 – All Data Systems Working



* EuXFEL can easily produce 1PB of data per week
* Simply buying more disks is not an option
* Major investments in Data Reduction accross the full data stack
  * Before data is written to disk
    ► RDMA will enable smarter algorithms offloaded to distributed hists
  * After data is on disk and as part of analysis
  * Automated procedures and online feedback to experimentalists

E. Sobolev et al.: Data reduction activities at European XFEL: early results

Summary

* Karabo is a flexible SCADA framework with a modern event-driven and asynchronous core.
  * 3 APIs with individual strengths
  * State of the art condition logging using the Influx Time Series database
  * Highly scalable, and no central database authority required
  * Integrated design: Control, Automation, Data Acquistion, and Analysis

* The Karabo GUI is closely matched to the framework
  * Can flexibly visualize all data types the framework provides
  * Synoptic views (scenes) without coding
  * Extensible

Hands-On Session: Machine Learning and Automation

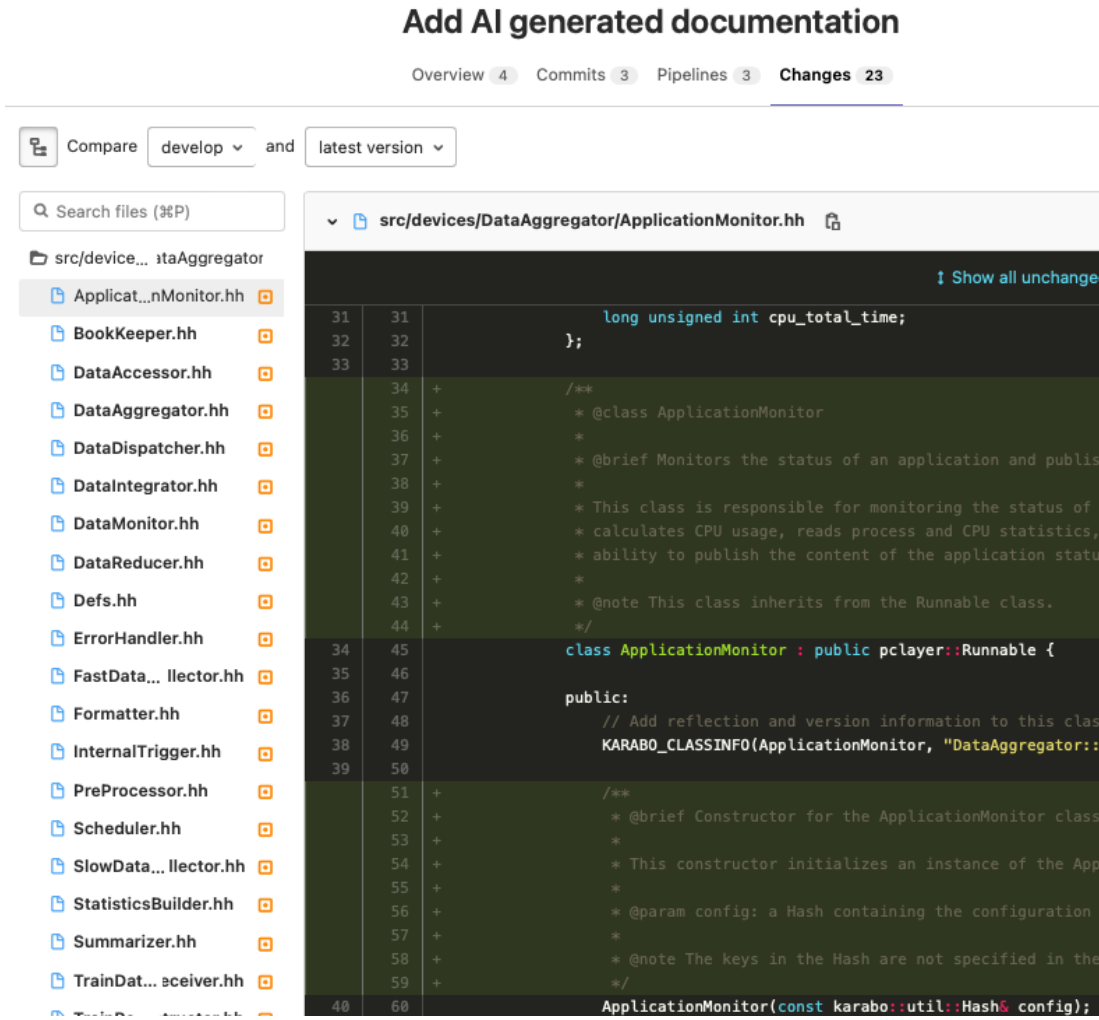
In order to access the virtual machines used during these sessions, sign in here using "Helmholtz AAI“ as authentication choice:
https://visa.xfel.eu/login

Please do so by Wednesday evening!

Karabo and Large Language Model: towards AI assistants

* **LLMs as code documentation assistants**
  * System prompt to ask AI to add or update documentation to code using a diff format that doesn't require counting lines
  * Works well to batch-document code lacking most documentation
  * Karabo agnostic, available at https://github.com/European-XFEL soon

* **LLMs as coding assistants**
  * GPT4 was not trained on Karabo code at the time of tests
  * System prompts describing the MDL API and the scene model suffice
  * Iterative approach, feeding exceptions back into the model
  * **https://syncandshare.xfel.eu/index.php/s/kt6NbSjJfMg7Pf5**



jupyter Test Last Checkpoint: 04/08/2023 (unsaved changes)

File Edit View Insert Cell Kernel Help KaraboAI Kernel

Code CellToolbar

Karabo AI Device Code Generator

Welcome to the Karabo AI device code generator. This notebook will iteratively guide you through the procedure of creating a new Karabo device according to your input prompts.

Prompts will usually forwarded directly to the AI, unless you start them with the following keywords:

- **validate**: will attempt to run the code in the previous code cell and return modified code that either executes, or has reached the interation limit
- **test**: will write unit tests for the latest code cell
- **scene**: will create a device provided scene on top of the source code
- **finalize**: will write out code and tests.

Please enter your prompt below. Good prompts descript what the device interface looks like, if data is to be polled, and which third party libraries and interface methods are prefered. When interfacing with hardware, the concrete model should be given, and a manual can be reference.

In [*]:
```
Write a Karabo Middlelayer Device that monitors the read-back parameters of a Kashiyama NeoDry vacuum pump.
The middlelayer device connects over ethernet to the RS485 serial port on the pump.
The communication protocol to read the pump parameters is Modbus-RTU.

The middlelayer device should poll periodically the following read-only parameters of the pump:
trip counter, Trip info. 1 Factor, Trip info. 1 Inverter status, Trip info. 1 Frequency (High),
Trip info. 1 Frequency (Low), Trip info. 1 Current, Trip info. 1 Voltage, output current,
input power, DC voltage. You provide code for all these parameters, no just stubs or a limited example set.

These should be exposed as read-only Karabo properties. Remember that any Karabo property is camelCased.

The only user provided configuration inputs are the tcp address and port of the serial converter,
and a polling interval in seconds. These too are Karabo properties.

Please use pymodbus python module to implement rtu-over-tcp modbus.
```

Waiting on AI...

In [ ]: