PSI Center for Scientific Computing, Theory and Data

A glimpse into BEC's event and scan logic

Christian Appel
AWI department meeting, 2024/09/03

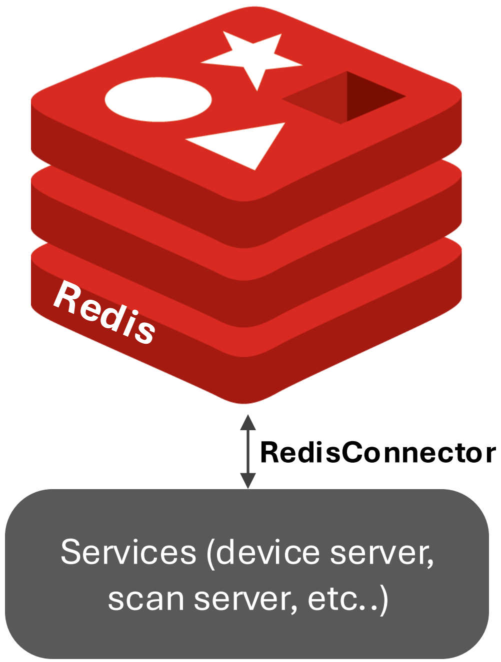
Outline

- Introduction to the event system
  - RedisConnector, MessageEndpoints, BECMessage
- Scan logic
  - Step and fly scans
- Integration with hardware and secondary services
  - Jungfraujoch@cSAXS
  - PandaBox@Pollux
  - NiDAQ motor controller@Debye

PSI Center for Scientific Computing, Theory and Data

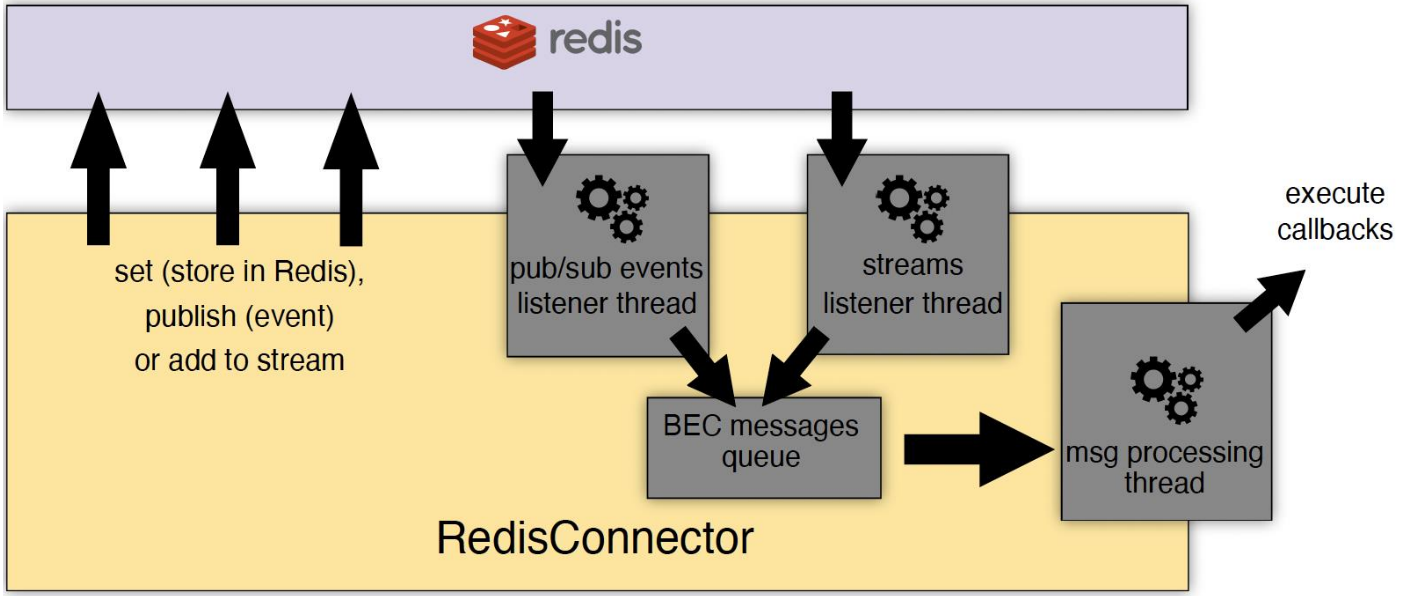
BEC Events

Redis as central component



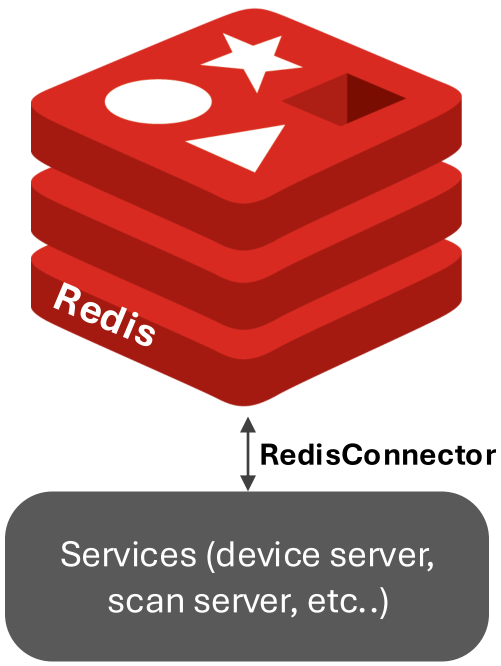
Redis as message broker, events are published to Redis

I. **RedisConnector**



Image/Content courtesy of Matias

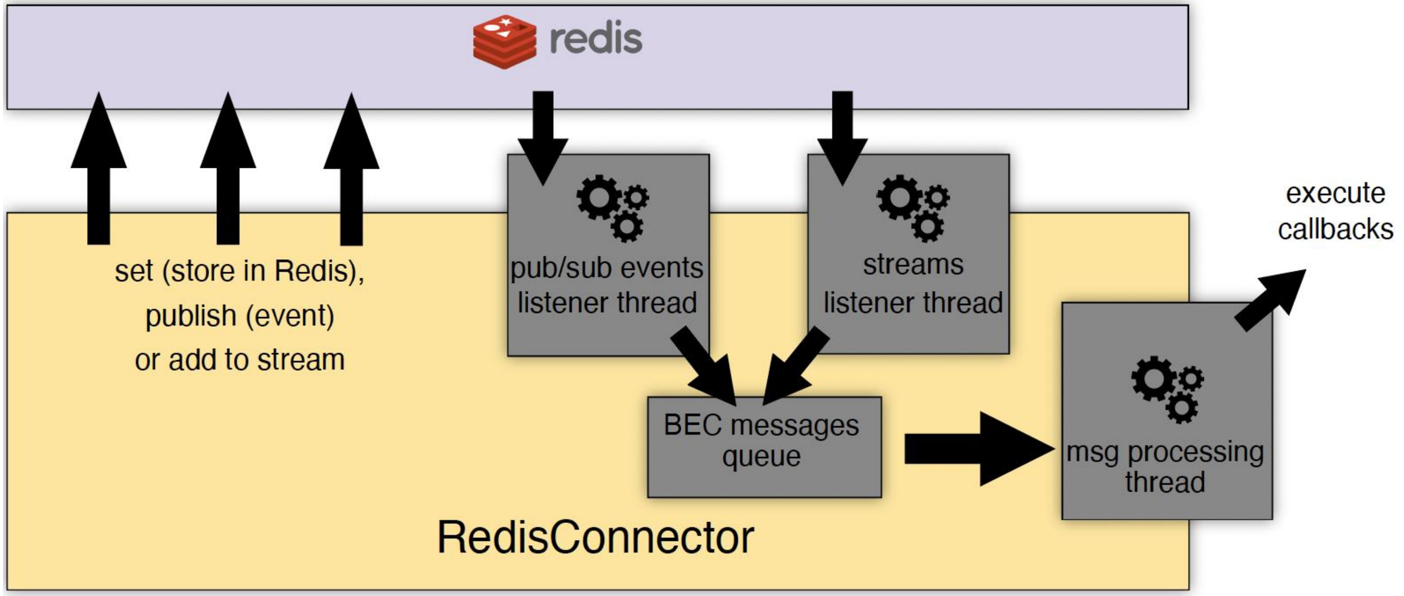
Redis as central component




```
Examples:
    >>> def my_callback(msg, **kwargs):
    ...     print(msg)
    ...
    >>> connector.register("test", my_callback)
    >>> connector.register(topics="test", cb=my_callback)
    >>> connector.register(patterns="test:*", cb=my_callback)
    >>> connector.register(patterns="test:*", cb=my_callback, start_thread=False)
    >>> connector.register(patterns="test:*", cb=my_callback, start_thread=False, my_arg="test")
"""
```

Redis as message broker, events are published to Redis

I. RedisConnector



Image/Content courtesy of Matias

Redis as central component

Redis as message broker, events are published to Redis

I. RedisConnector

II. MessageEndpoints

III. BECMessage *(Pydantic models)*

```
from bec_lib import messages
from bec_lib.endpoints import MessageEndpoints

MessageEndpoints.file_event("samx")
EndpointInfo(endpoint='public/file_event/samx',
             message_type=<class 'bec_lib.messages.FileMessage'>,
             message_op=<MessageOp.SET_PUBLISH: ['register', 'set_and_publish', 'delete', 'get', 'keys']>)
```

Redis as central component

Redis as message broker, events are published to Redis

I. RedisConnector

II. MessageEndpoints

III. BECMessage *(Pydantic models)*

```python
from bec_lib import messages
from bec_lib.endpoints import MessageEndpoints

MessageEndpoints.file_event("samx")
EndpointInfo(endpoint='public/file_event/samx',
             message_type=<class 'bec_lib.messages.FileMessage'>,
             message_op=<MessageOp.SET_PUBLISH: ['register', 'set_and_publish', 'delete', 'get', 'keys']>)

# create a new instance of a BECMessage class, e.g. a FileMessage
msg = messages.FileMessage(file_path='path/to/file', done=False, successful=True, metadata={'scan_id': "1234"})

# Publish the message to the file_event endpoint
bec.connector.set_and_publish(MessageEndpoints.file_event("samx"), msg)

# Get the message from the file_event endpoint
bec.connector.get(MessageEndpoints.file_event("samx"))

# Subscribe to the file_event endpoint
bec.connector.register(MessageEndpoints.file_event("samx"), my_callback)
```

PSI Center for Scientific Computing, Theory and Data

Scan Logic

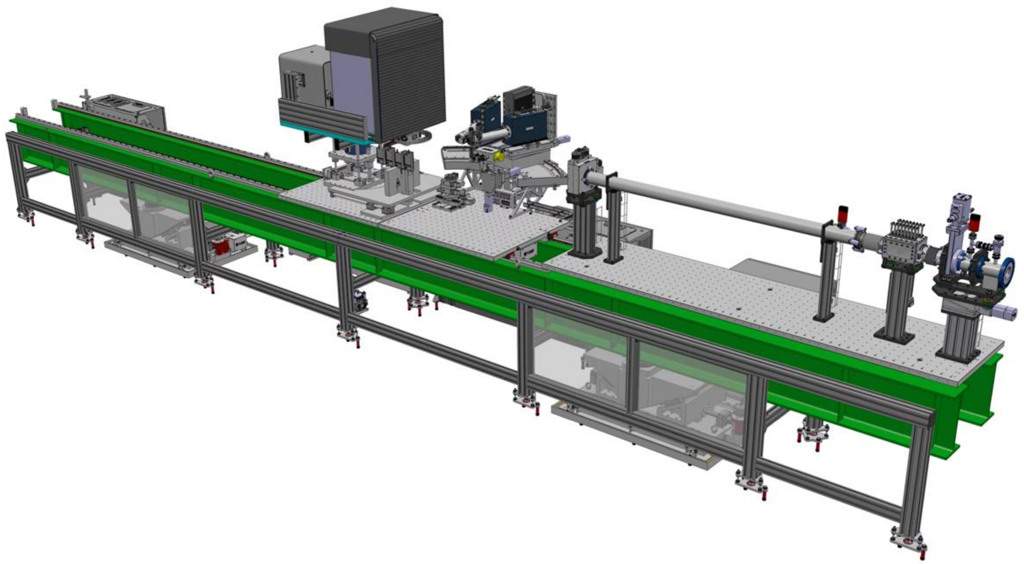
Scan Logic within BEC

PSI

BEC

Must have:

The option to scan (almost) all motors, detectors etc at any beamline…



https://www.psi.ch/en/sls/debye/experimental-station

Type of scans

Step scan:

Array with N points, move scan motors from point to point, measure and repeat.
BEC controls triggering monitored devices for each point of the scan.

Type of scans

PSI

Step scan:

Array with N points, move scan motors from point to point, measure and repeat.
BEC controls triggering monitored devices for each point of the scan.

Fly scan:

More flexibility to implement dedicated (potentially device specific) logic. Typically, a single device drives the scan and triggers the relevant detectors (usually > 100Hz).
Monitored devices are read out at lower frequency (~1-10Hz).

So how do we implement this?

Hardware abstraction:

Clean interface to positioner, signals & devices → ophyd_devices

positioner

Devices
(i.e. detector)

signals

So how do we implement this?

Hardware abstraction:

Clean interface to positioner, signals & devices → ophyd_devices

Build scans following a well-established chain of commands (selection shown):

open → stage → pre → “scan core” → complete → unstage → close

positioner

Devices (i.e. detector)

signals

So how do we implement this?

Hardware abstraction:

Clean interface to positioner, signals & devices → ophyd_devices

Build scans following a well-established chain of commands (selection shown):

open → stage → pre → “scan core” → complete → unstage → close

Most of the time, all relevant information from the scan is available at the first step.
Leverage from BEC events to inform all services & devices about upcoming scan.

positioner

Devices
(i.e. detector)

signals

So how do we implement this?

PSI

Hardware abstraction:

Clean interface to positioner, signals & devices → ophyd_devices

Build scans following a well-established chain of commands (selection shown):

open → stage → pre → "scan core" → complete → unstage → close

Most of the time, all relevant information from the scan is available at the first step.
Leverage from BEC events to inform all services & devices about upcoming scan.

positioner

Devices
(i.e. detector)

signals

Devices request scaninfo during stage, prepare themselves autonomeously.
Afterwards they report whether they succesfully complete the scan!

PSI Center for Scientific Computing, Theory and Data

Use Cases

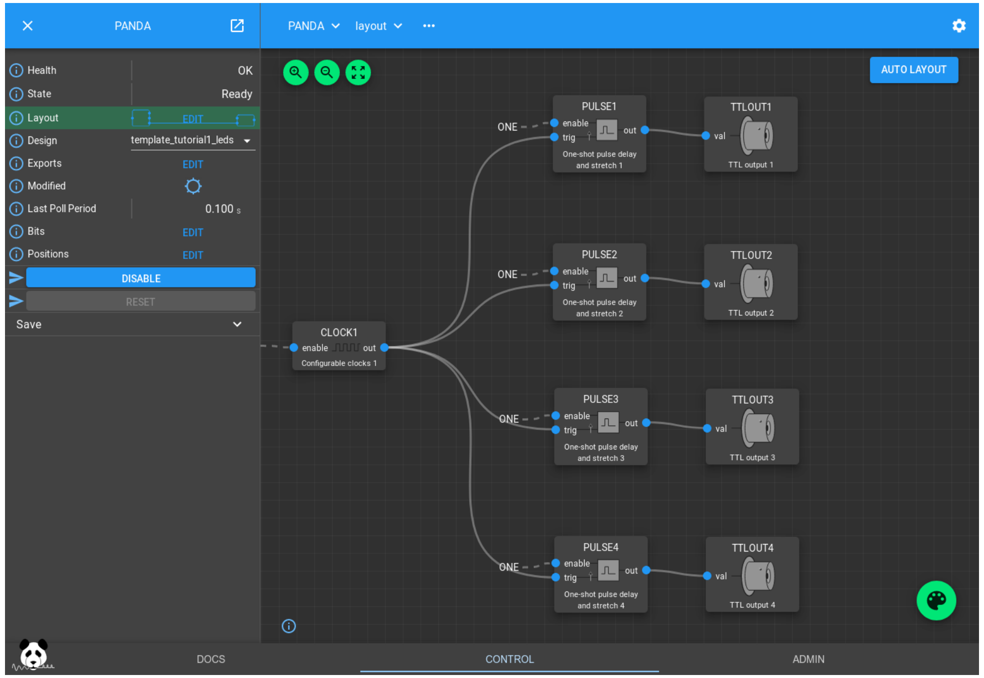
PandaBox at Pollux

FPGA based device to compare, record and act on encoder and digital inputs. Can also produce TTL outputs (trigger signals)

Established mode for automated switching between 4 layouts (step & fly) for low (<1kHz) and high frequency (50kHz) sampling

Outlook:

Live updates of encoder readings can be pushed to Redis and allow BEC to react.



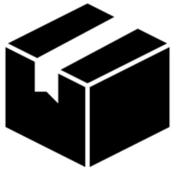
Jungfraujoch at cSAXS

“Detector” control and backend, hardware installed but further integration work is on hold

Data analysis pipelines (DAP) can be configured, on FPGA boards or GPUs. High throughput/data volume solution

Outlook:

BEC can be configured to react on results of the live feedback



“Black box” design
Inspired by DECTRIS control unit
Unit: all-in-one
Tailored for SLS detectors
(JUNGFRAU, EIGER, MATTERHORN)

HW and SW platform
FPGA smart network card
Image analysis on GPU
Compression on CPU

BEC at Debye

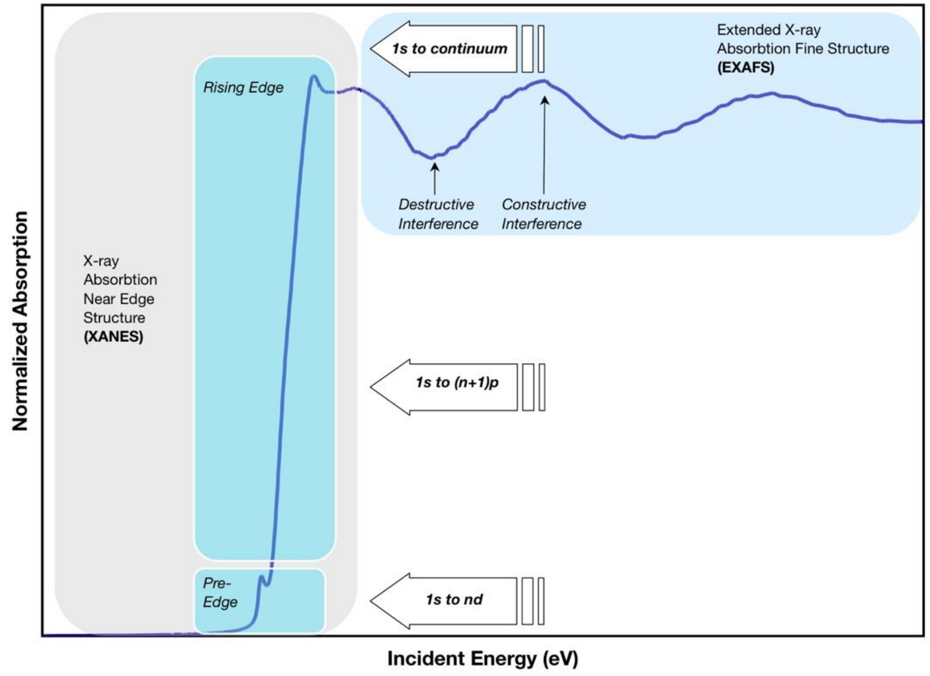
NIDAQ controller steers oscillation of the monochromator. ~1-2keV at 1Hz.

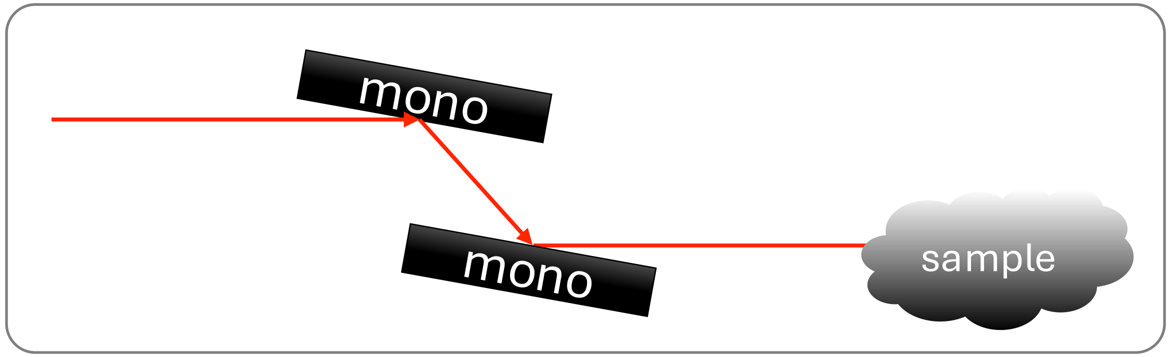
Device control & scans fully integrated, data backend (NIDAQ board) still with controls.

Outlook:

NIDAQ data will be sent to Redis, DAP processing will be expected by BEC, i.e. visualisation, normalisation, background subtraction and filtering.

https://en.wikipedia.org/wiki/Extended_X-ray_absorption_fine_structure

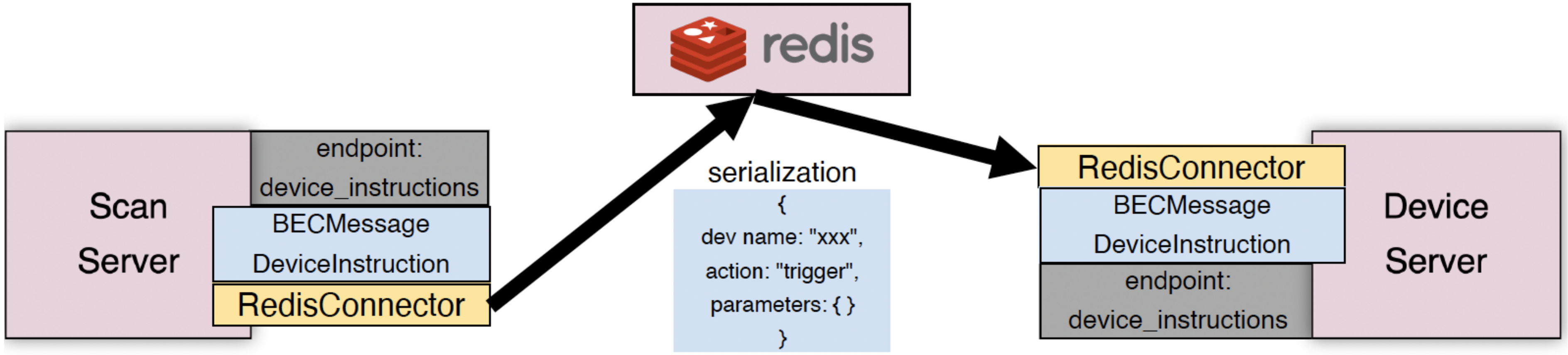




PSI Center for Scientific Computing, Theory and Data

The End!
Enjoy your lunch!

MessageEndpoints



Image/Content courtesy of Matias

Summary

Redis as message broker

- Events published to specific **MessageEndpoints**
  - Allowed operations:
    - Set, get, set_and_publish, xread, …
  - **BEC Message** (*Pydantic models* -> validation)
    - DeviceInstructionMessage, ScanStatusMessage, …

How does this integrate with other hardware, service and software solutions?

- Device Integration
  - Debye Beamline (XAS, EXAFS measurements)
  - *On The Fly focus group:* PandaBox
- DAQ service integration
  - Jungfraujoch integration at cSAXS