



| The European Synchrotron



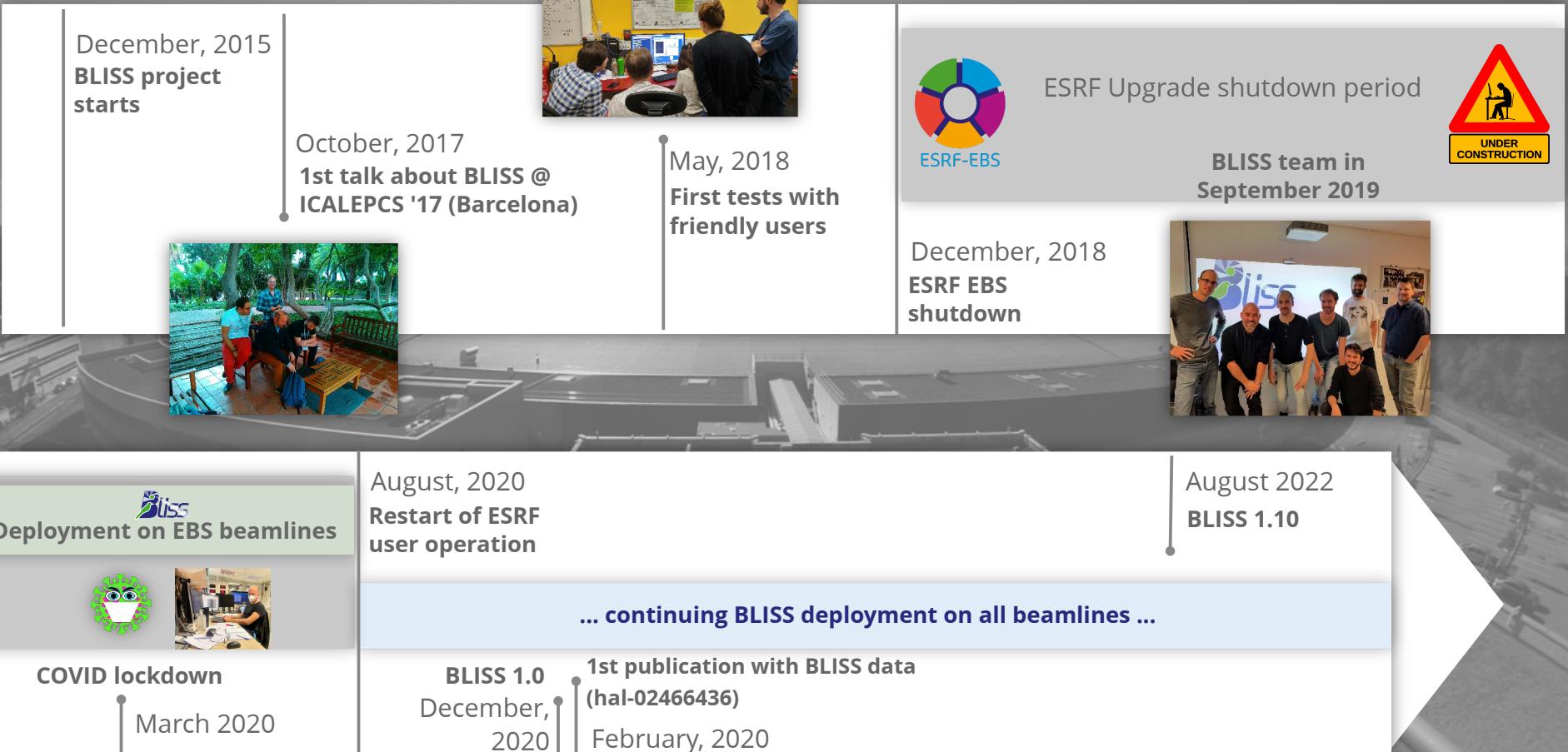
Beamline Instrumentation Support Software

ESRF beamline control system

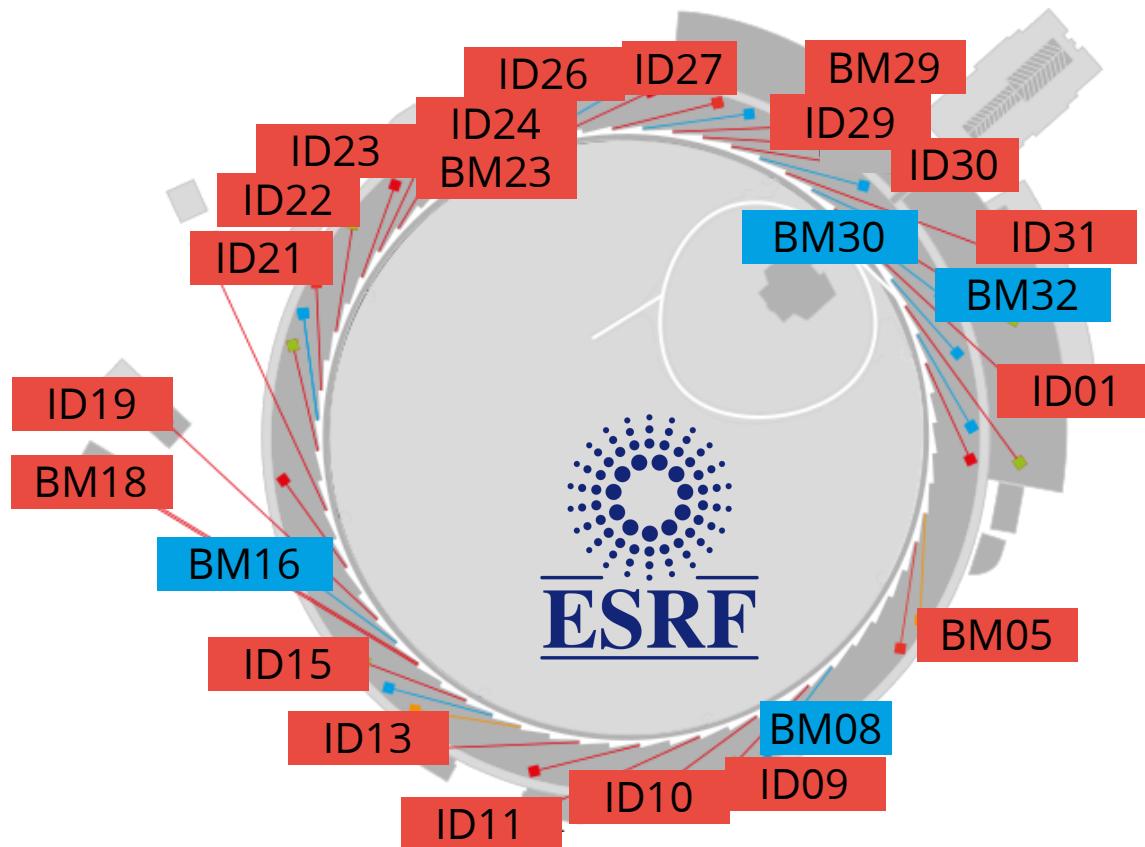
NOBUGS 
19/09/2022
PSI, Villigen, Switzerland

*presented by Matias Guijarro - BLISS team
Beamline Control Unit, Software Group*





BLISS beamlines (september 2022)



— ESRF beamlines
— CRG beamlines

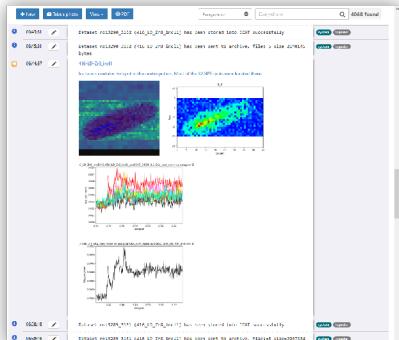

Full deployment
aimed at the
horizon of 2024

Today's experiment control system challenges

Data portal: <https://data.esrf.fr>



E-logbook

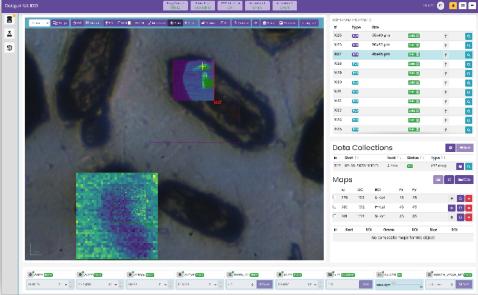


Faster detectors, data production



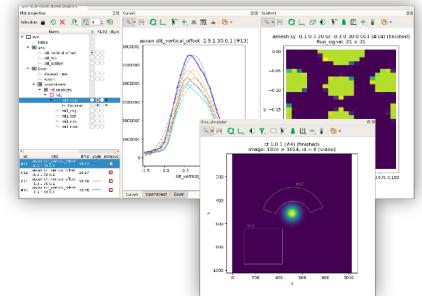
Jungfrau @ ID29

Graphical user interface for data acquisition

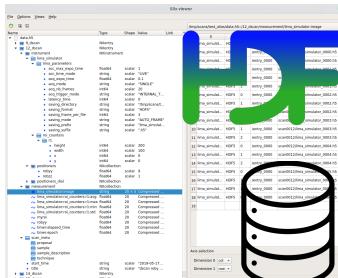


(cf. previous talk by S. Fisher)

Live data visualization

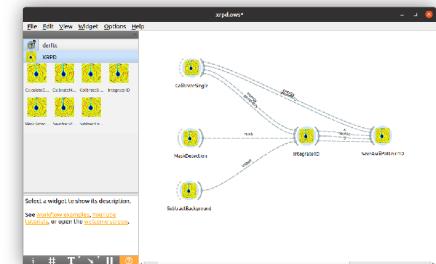


Online data analysis workflows



(see Ewoks presentation by W. De Nolf this afternoon !)

Efficient data format, storage

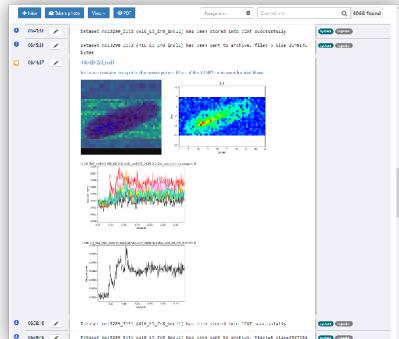


Today's experiment control system challenges

Data portal: <https://data.esrf.fr>

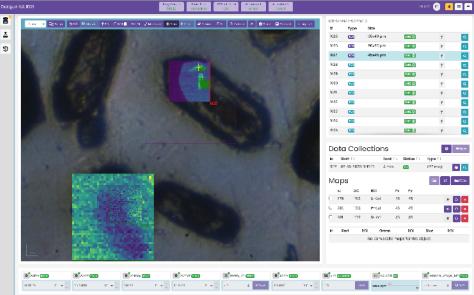


E-logbook



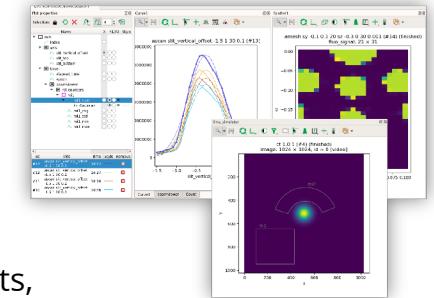
Faster detectors, data production

Graphical user interface for data acquisition

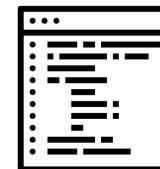


(cf. previous talk by S. Fisher)

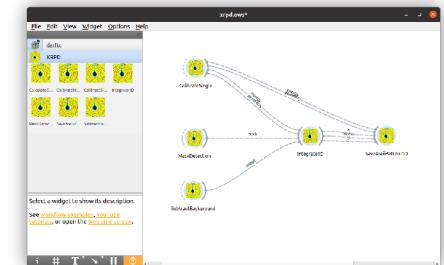
Live data visualization



Experiment scripts,
user sequences



Online data analysis
workflows

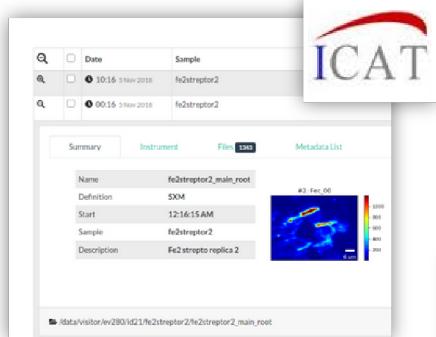


(see Ewoks presentation by
W. De Nolf this afternoon !)

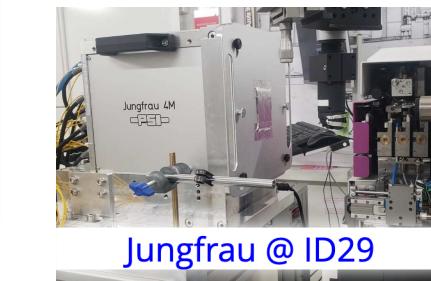
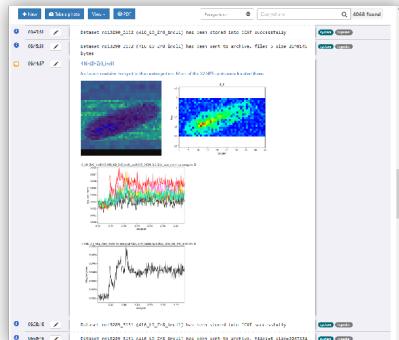
Efficient data format,
storage

Today's experiment control system challenges

Data portal: <https://data.esrf.fr>



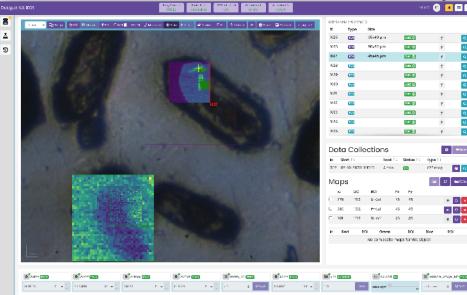
E-logbook



Jungfrau @ ID29

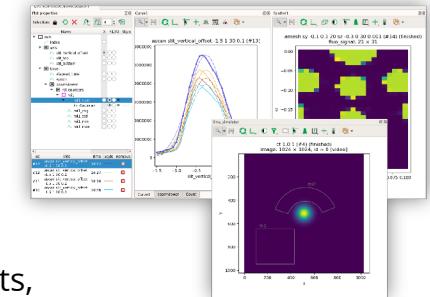
Faster detectors, data production

Graphical user interface for data acquisition

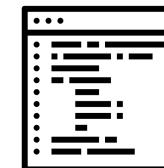


(cf. previous talk by S. Fisher)

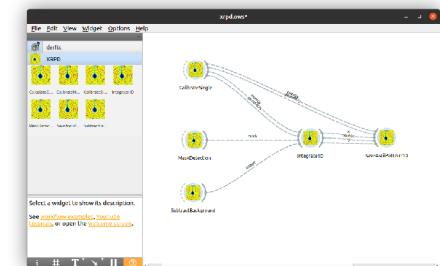
Live data visualization



Experiment scripts,
user sequences

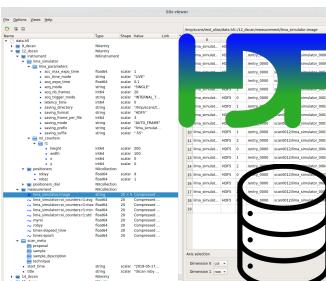


Online data analysis
workflows



(see Ewoks presentation by
W. De Nolf this afternoon !)

Efficient data format,
storage



BLISS technical choices and design principles

Written in Python 

BLISS technical choices and design principles

Written in Python 

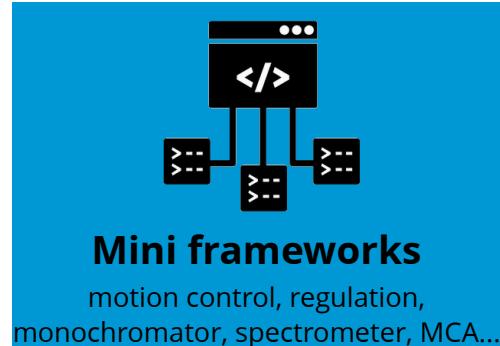


All-in-one solution

configuration management, logging,
communication protocols, hw drivers,
shell (CLI), scanning engine, live data
display, publishing and saving

BLISS technical choices and design principles

Written in Python 

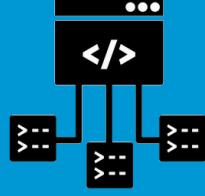


BLISS technical choices and design principles

Written in Python 



All-in-one solution
configuration management, logging,
communication protocols, hw drivers,
shell (CLI), scanning engine, live data
display, publishing and saving



Mini frameworks
motion control, regulation,
monochromator, spectrometer, MCA...



I/O based on gevent
cooperative multi-tasking,
asyncio interoperability

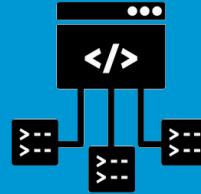
BLISS technical choices and design principles

Written in Python 



All-in-one solution

configuration management, logging,
communication protocols, hw drivers,
shell (CLI), scanning engine, live data
display, publishing and saving



Mini frameworks

motion control, regulation,
monochromator, spectrometer, MCA...



I/O based on `gevent`

cooperative multi-tasking,
asyncio interoperability



Built-in direct hardware control



TANGO extensions for
interoperability with other systems, or
in case of shared hw

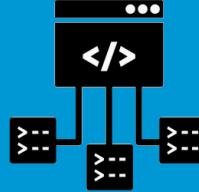
BLISS technical choices and design principles

Written in Python 



All-in-one solution

configuration management, logging, communication protocols, hw drivers, shell (CLI), scanning engine, live data display, publishing and saving



Mini frameworks

motion control, regulation, monochromator, spectrometer, MCA...



I/O based on gevent

cooperative multi-tasking, asyncio interoperability



Built-in direct hardware control



TANGO extensions for interoperability with other systems, or in case of shared hw



Scanning with the acquisition chain

Tree structure to represent "trigger" relationship of devices in a scan

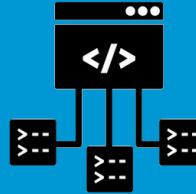
BLISS technical choices and design principles

Written in Python 



All-in-one solution

configuration management, logging, communication protocols, hw drivers, shell (CLI), scanning engine, live data display, publishing and saving



Mini frameworks

motion control, regulation, monochromator, spectrometer, MCA...



I/O based on gevent

cooperative multi-tasking, asyncio interoperability



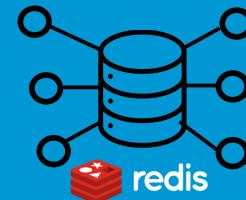
Built-in direct hardware control

 extensions for interoperability with other systems, or in case of shared hw



Scanning with the acquisition chain

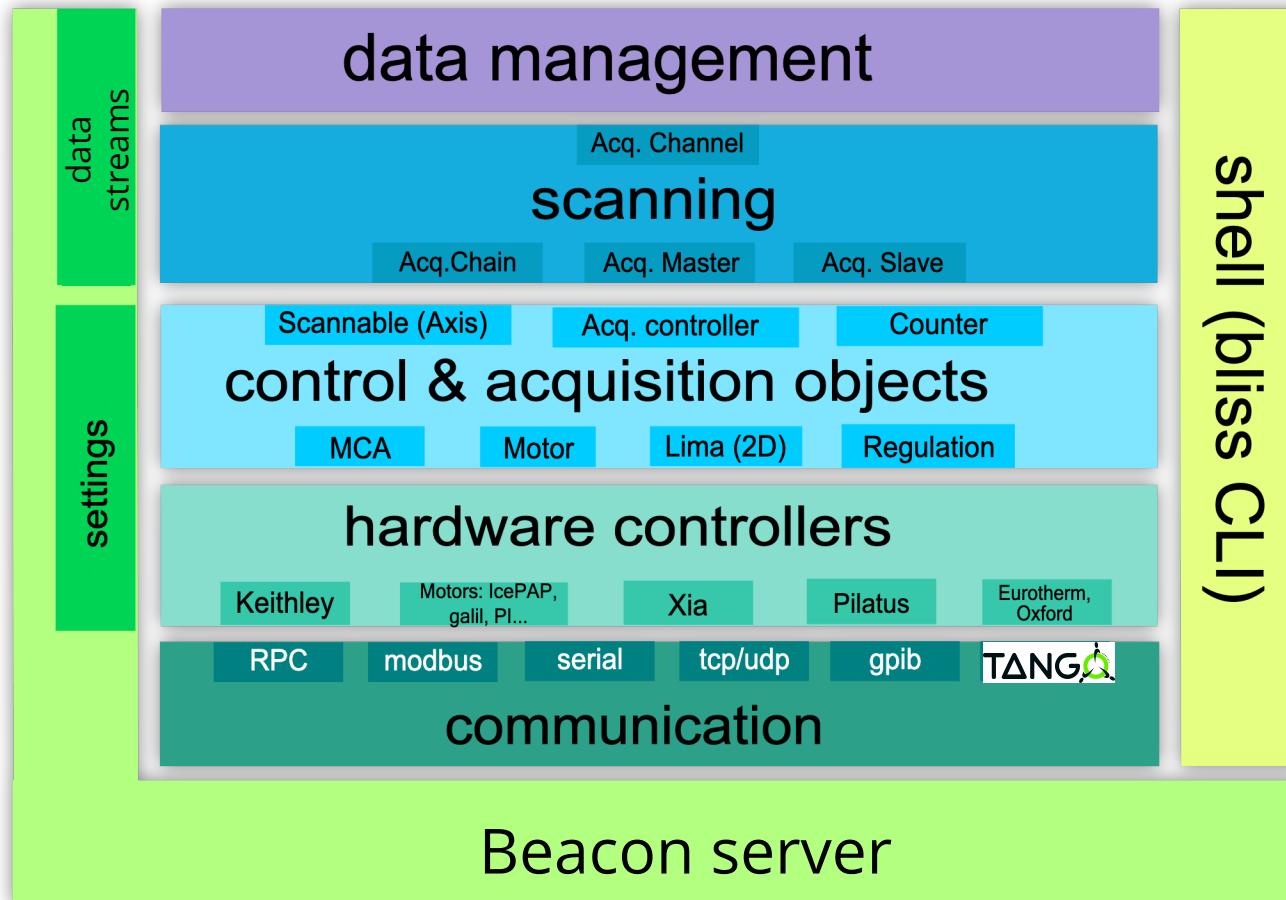
Tree structure to represent "trigger" relationship of devices in a scan



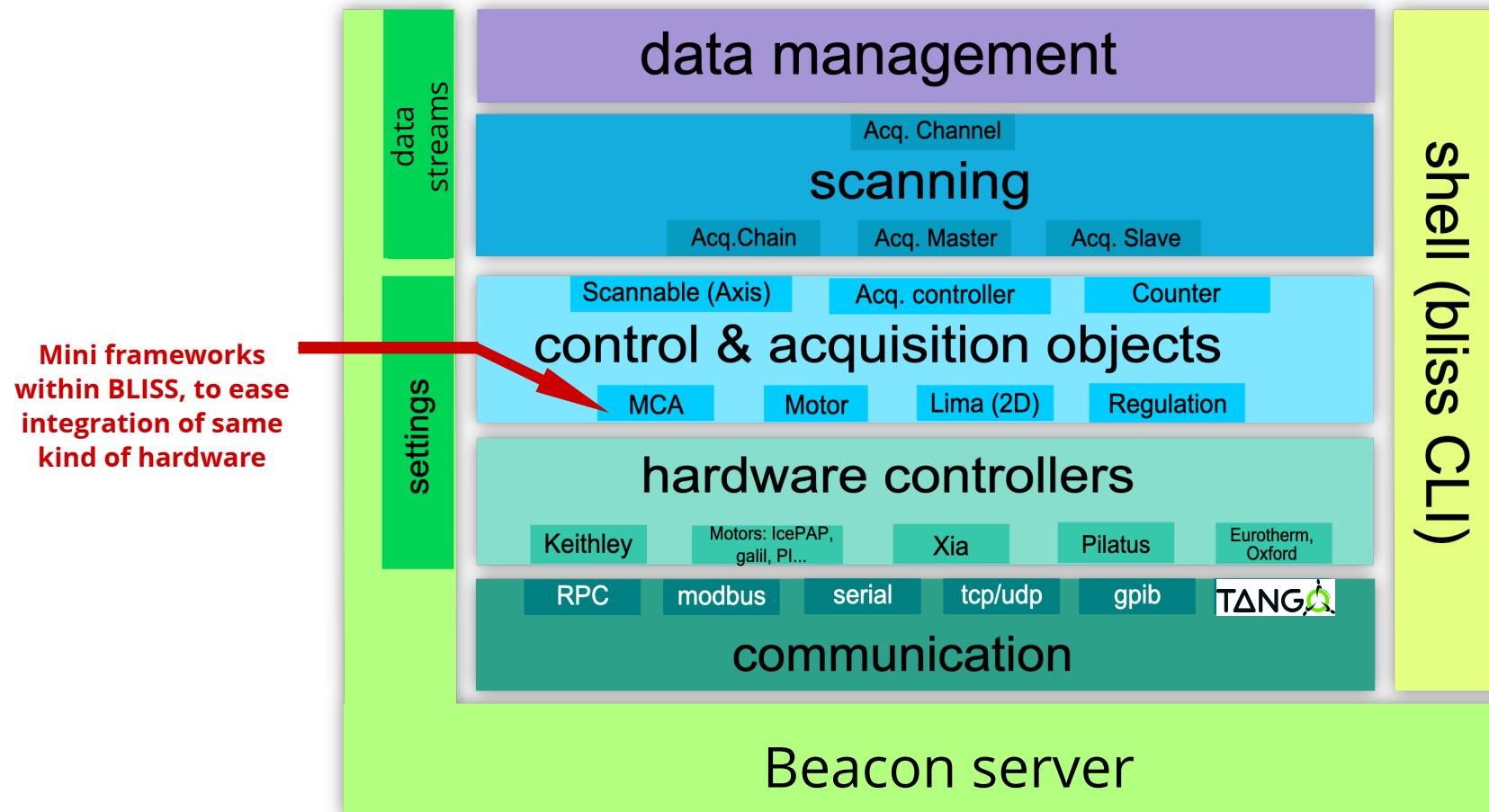
Data publishing,

acquisition made independent of storage

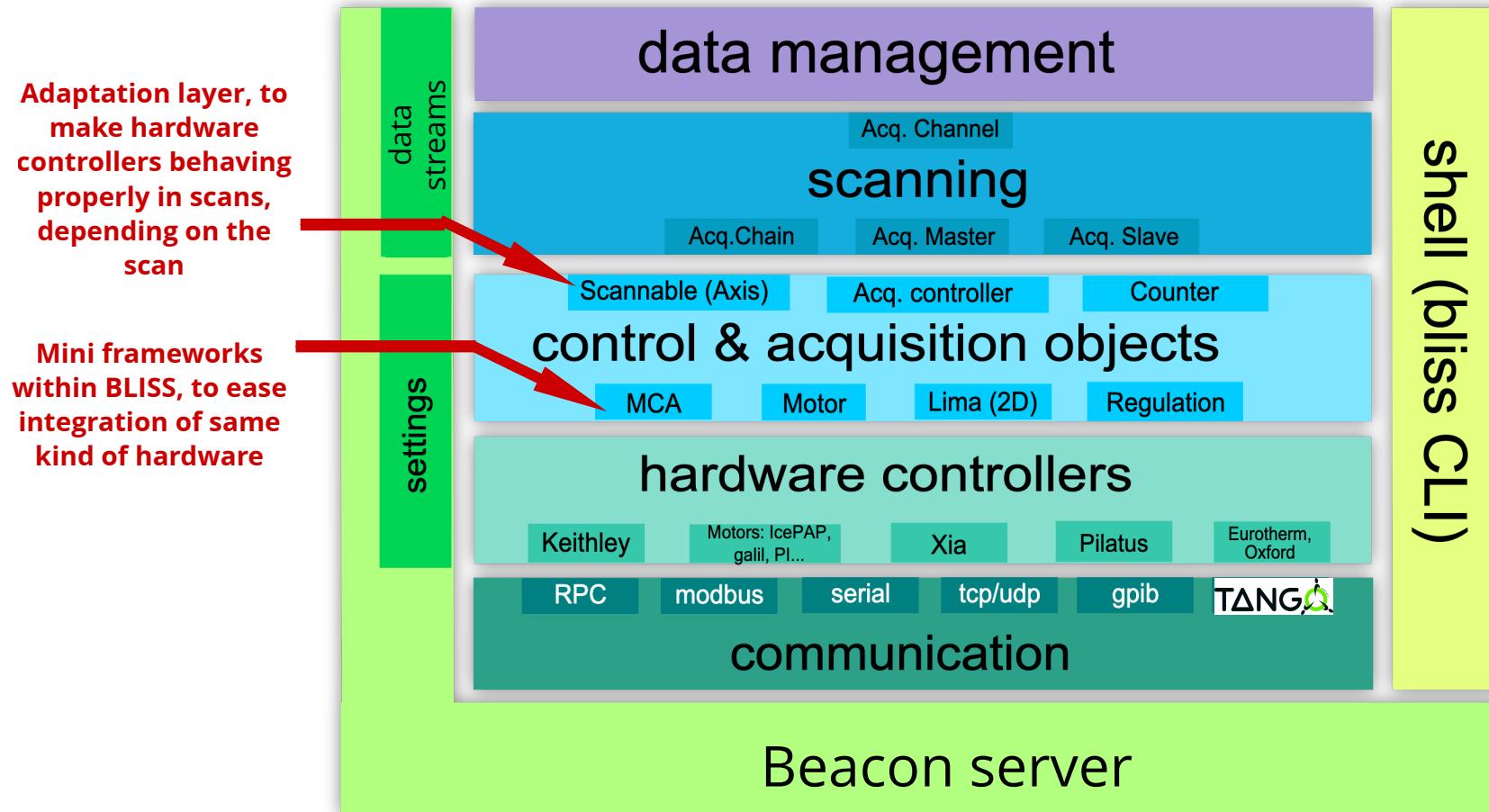
BLISS modular architecture overview



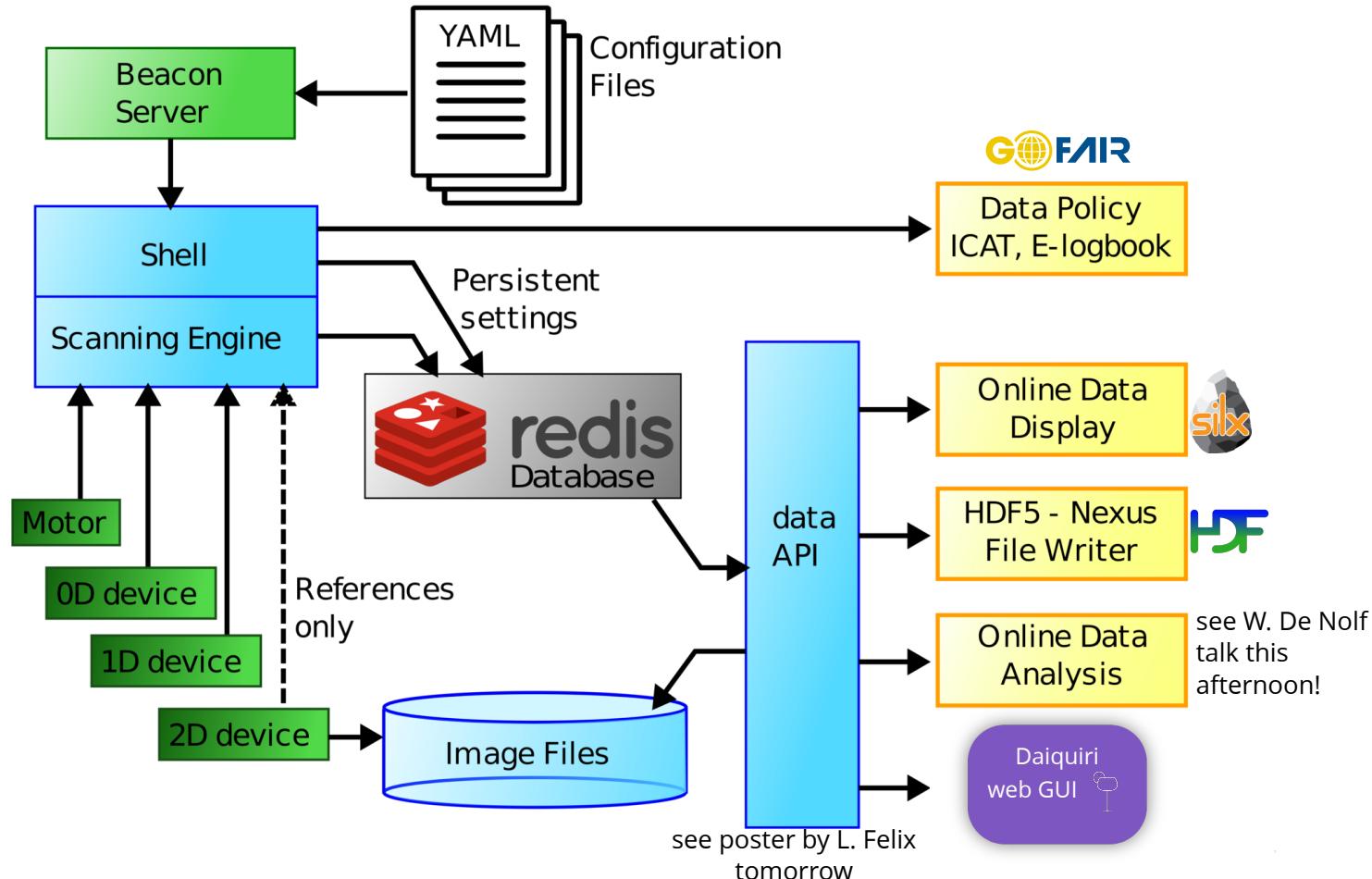
BLISS modular architecture overview



BLISS modular architecture overview



BLISS data flow



BLISS features tour

Most praised by ESRF scientists

1. Python!
2. Flint, BLISS interactive data display
3. Software axes and counters
4. Software regulation loops
5. Accessing BLISS data on the fly

1. Python!

Example from BM23 (Kirill Lomachenko): beamline "health check" for local contacts via Telegram, thanks to Telegram Python API



Scientists can take advantage of the huge Python ecosystem in their own scripts

1. Python!

Example from ID11: tomography @ nanofocus endstation



500 fps

700 GB per
hour of
(compressed)
data

1. Python!

Example from ID11: tomography @ nanofocus endstation



700 GB per
hour of
(compressed)
data

User-written acquisition sequence == night shift macro

```
1 from bliss.setup_globals import fsh, diffrrz
2 import numpy as np
3
4 def ftomo_series(scanname, start, num_scans, sleep_time=0, pars=None):
5     for i in np.arange(start, start+num_scans):
6         newdataset(f"{scanname}{i}")
7         umv(diffrrz, pars['start_pos'])
8         with fsh.closed_context():
9             print("taking dark images")
10            ftimescan(pars['exp_time'], pars['nref'], 0)
11        if i==start or not(i%pars['ref_int']):
12            print("taking flat images")
13            ref_scan(pars['ref_mot'], pars['exp_time'], pars['nref'], pars['ref_step'], pars['start_pos'], pars['scan_mode'])
14        else:
15            print("skipping flat images - stay where we are")
16            ftimescan(pars['exp_time'], 10,0)
17        print("taking projections...")
18        fscan(diffrrz, pars['start_pos'], pars['step_size'], pars['num_proj'], pars['exp_time'], scan_mode = pars['scan_mode'])
19        print("resetting diffrrz to 0")
20        umv(diffrrz, pars['start_pos']+360)
21        diffrrz.position=pars['start_pos']
22        sleep(sleep_time)
```

1. Python!

Example from ID11: tomography @ nanofocus endstation



700 GB per
hour of
(compressed)
data

User-written acquisition sequence == night shift macro

```
1 from bliss.setup_globals import fsh, diffrrz  direct import of beamline objects from BLISS setup in user scripts
2 import numpy as np
3
4 def ftomo_series(scanname, start, num_scans, sleep_time=0, pars=None):
5     for i in np.arange(start, start+num_scans):
6         newdataset(f"{scanname}{i}")
7         umv(diffrrz, pars['start_pos'])
8         with fsh.closed_context():
9             print("taking dark images")
10            ftimescan(pars['exp_time'], pars['nref'], 0)
11        if i==start or not(i%pars['ref_int']):
12            print("taking flat images")
13            ref_scan(pars['ref_mot'], pars['exp_time'], pars['nref'], pars['ref_step'], pars['start_pos'], pars['scan_mode'])
14        else:
15            print("skipping flat images - stay where we are")
16            ftimescan(pars['exp_time'], 10,0)
17        print("taking projections...")
18        fscan(diffrrz, pars['start_pos'], pars['step_size'], pars['num_proj'], pars['exp_time'], scan_mode = pars['scan_mode'])
19        print("resetting diffrrz to 0")
20        umv(diffrrz, pars['start_pos']+360)
21        diffrrz.position=pars['start_pos']
22        sleep(sleep_time)
```

1. Python!

Example from ID11: tomography @ nanofocus endstation



700 GB per
hour of
(compressed)
data

User-written acquisition sequence == night shift macro

```
1 from bliss.setup_globals import fsh, diffrrz direct import of beamline objects from BLISS setup in user scripts
2 import numpy as np whole Python ecosystem can be available
3
4 def ftomo_series(scanname, start, num_scans, sleep_time=0, pars=None):
5     for i in np.arange(start, start+num_scans):
6         newdataset(f"{scanname}{i}")
7         umv(diffrrz, pars['start_pos'])
8         with fsh.closed_context():
9             print("taking dark images")
10            ftimescan(pars['exp_time'], pars['nref'], 0)
11        if i==start or not(i%pars['ref_int']):
12            print("taking flat images")
13            ref_scan(pars['ref_mot'], pars['exp_time'], pars['nref'], pars['ref_step'], pars['start_pos'], pars['scan_mode'])
14        else:
15            print("skipping flat images - stay where we are")
16            ftimescan(pars['exp_time'], 10,0)
17        print("taking projections...")
18        fscan(diffrrz, pars['start_pos'], pars['step_size'], pars['num_proj'], pars['exp_time'], scan_mode = pars['scan_mode'])
19        print("resetting diffrrz to 0")
20        umv(diffrrz, pars['start_pos']+360)
21        diffrrz.position=pars['start_pos']
22        sleep(sleep_time)
```

1. Python!

Example from ID11: tomography @ nanofocus endstation



700 GB per
hour of
(compressed)
data

User-written acquisition sequence == night shift macro

```
1 from bliss.setup_globals import fsh, diffrrz  direct import of beamline objects from BLISS setup in user scripts
2 import numpy as np whole Python ecosystem can be available
3
4 def ftomo_series(scanname, start, num_scans, sleep_time=0, pars=None): Python functions, with named arguments and default values
5     for i in np.arange(start, start+num_scans):
6         newdataset(f"{scanname}{i}")
7         umv(diffrrz, pars['start_pos'])
8         with fsh.closed_context():
9             print("taking dark images")
10            ftimescan(pars['exp_time'], pars['nref'], 0)
11        if i==start or not(i%pars['ref_int']):
12            print("taking flat images")
13            ref_scan(pars['ref_mot'], pars['exp_time'], pars['nref'], pars['ref_step'], pars['start_pos'], pars['scan_mode'])
14        else:
15            print("skipping flat images - stay where we are")
16            ftimescan(pars['exp_time'], 10,0)
17        print("taking projections...")
18        fscan(diffrrz, pars['start_pos'], pars['step_size'], pars['num_proj'], pars['exp_time'], scan_mode = pars['scan_mode'])
19        print("resetting diffrrz to 0")
20        umv(diffrrz, pars['start_pos']+360)
21        diffrrz.position=pars['start_pos']
22        sleep(sleep_time)
```

1. Python!

Example from ID11: tomography @ nanofocus endstation



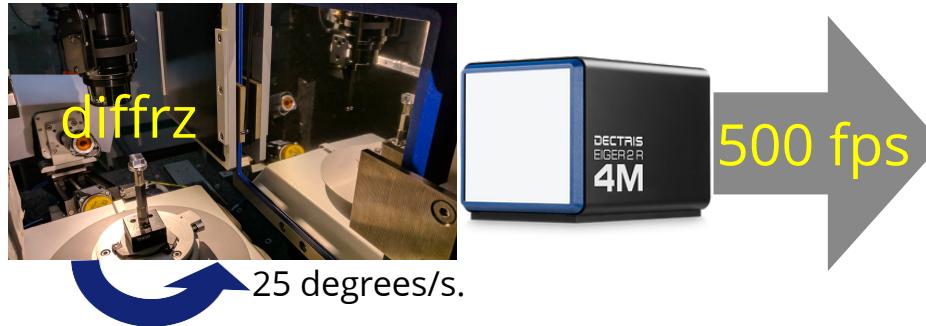
700 GB per
hour of
(compressed)
data

User-written acquisition sequence == night shift macro

```
1 from bliss.setup_globals import fsh, diffrrz  direct import of beamline objects from BLISS setup in user scripts
2 import numpy as np whole Python ecosystem can be available
3
4 def ftomo_series(scanname, start, num_scans, sleep_time=0, pars=None): Python functions, with named arguments and default values
5     for i in np.arange(start, start+num_scans):
6         newdataset(f"{scanname}{i}")
7         umv(diffrrz, pars['start_pos'])
8         with fsh.closed_context():
9             print("taking dark images")
10            ftimescan(pars['exp_time'], pars['nref'], 0)
11        if i==start or not(i%pars['ref_int']):
12            print("taking flat images")
13            ref_scan(pars['ref_mot'], pars['exp_time'], pars['nref'], pars['ref_step'], pars['start_pos'], pars['scan_mode'])
14        else:
15            print("skipping flat images - stay where we are")
16            ftimescan(pars['exp_time'], 10,0)
17        print("taking projections...")
18        fscan(diffrrz, pars['start_pos'], pars['step_size'], pars['num_proj'], pars['exp_time'], scan_mode = pars['scan_mode'])
19        print("resetting diffrrz to 0")
20        umv(diffrrz, pars['start_pos']+360)
21        diffrrz.position=pars['start_pos']
22        sleep(sleep_time)
```

1. Python!

Example from ID11: tomography @ nanofocus endstation



700 GB per
hour of
(compressed)
data

User-written acquisition sequence == night shift macro

```
1 from bliss.setup_globals import fsh, diffrrz direct import of beamline objects from BLISS setup in user scripts
2 import numpy as np whole Python ecosystem can be available
3
4 def ftomo_series(scanname, start, num_scans, sleep_time=0, pars=None): Python functions, with named arguments and default values
5     for i in np.arange(start, start+num_scans):
6         newdataset(f"{scanname}{i}")
7         umv(diffrrz, pars['start_pos'])
8         with fsh.closed_context():
9             print("taking dark images")
10            ftimescan(pars['exp_time'], pars['nref'], 0)
11        if i==start or not(i%pars['ref_int']):
12            print("taking flat images")
13            ref_scan(pars['ref_mot'], pars['exp_time'], pars['nref'], pars['ref_step'], pars['start_pos'], pars['scan_mode'])
14        else:
15            print("skipping flat images - stay where we are")
16            ftimescan(pars['exp_time'], 10.0)
17        print("taking projections...")
18        fscan(diffrrz, pars['start_pos'], pars['step_size'], pars['num_proj'], pars['exp_time'], scan_mode = pars['scan_mode'])
19        print("resetting diffrrz to 0")
20        umv(diffrrz, pars['start_pos']+360)
21        diffrrz.position=pars['start_pos']
22        sleep(sleep_time)
```

continuous, hardware-
synchronized scans

2. Flint, BLISS interactive data display tool

Flint is a GUI companion for BLISS for online data visualization
It built based on [ESRF silx](#) (Scientific Library for eXperimentalists) and [Qt](#) (Qt)

It provides **live display of BLISS scans,**
custom plotting
and **interactive control**
from BLISS scripts

2. Flint, BLISS interactive data display tool

Flint is a GUI companion for BLISS for online data visualization
It built based on [ESRF silx](#) (Scientific Library for eXperimentalists) and [Qt](#) (Qt)

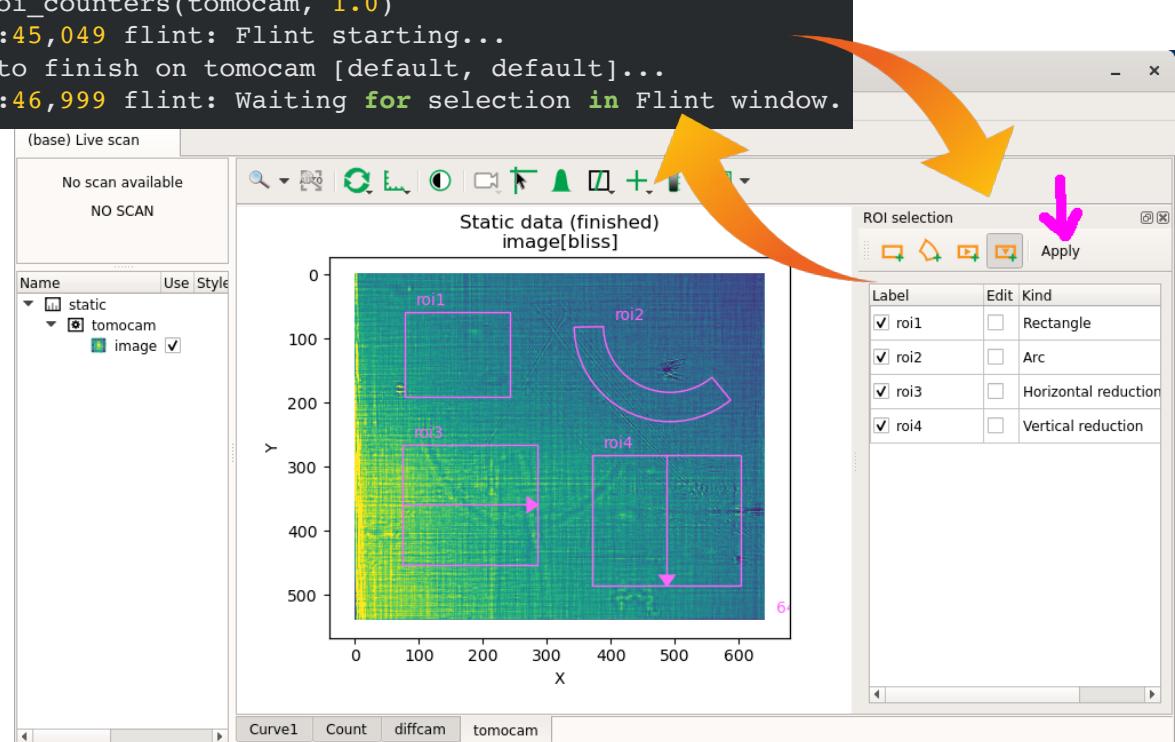
```
1 TOMO_SESSION [1]: edit_roi_counters(tomocam, 1.0)
2 WARNING 2020-10-16 15:39:45,049 flint: Flint starting...
3 Waiting for ROI edition to finish on tomocam [default, default]...
4 WARNING 2020-10-16 15:39:46,999 flint: Waiting for selection in Flint window.
```

It provides **live display of BLISS scans,**
custom plotting
and **interactive control**
from BLISS scripts

2. Flint, BLISS interactive data display tool

Flint is a GUI companion for BLISS for online data visualization
It built based on [ESRF silx](#) (Scientific Library for eXperimentalists) and [Qt](#) (Qt)

```
1 TOMO_SESSION [1]: edit_roi_counters(tomocam, 1.0)
2 WARNING 2020-10-16 15:39:45,049 flint: Flint starting...
3 Waiting for ROI edition to finish on tomocam [default, default]...
4 WARNING 2020-10-16 15:39:46,999 flint: Waiting for selection in Flint window.
```

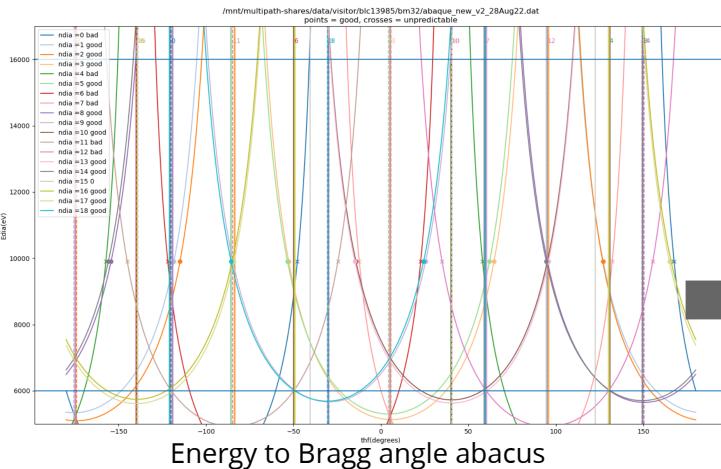


It provides **live display of BLISS scans**,
custom plotting and **interactive control** from BLISS scripts

3.1 Software pseudo axes

Example from BM32: Laue diffraction

Pseudo axis to switch from Energy (HKL reciprocal space)
to crystal angle (real hardware, rotation axis)



Energy to Bragg angle abacus



```
1 - controller:
2   class: EnergyCalc
3   package: bm32.ctrls.pseudo_laue
4
5
6   - name: $thf
7     tags: real thf
8
9   - name: Edia
10    tags: Edia
11    unit: eV
12
13
14
15
16
17
18
```

```
1 file: ~/local/bm32/ctrls/pseudo_laue.py
2
3
4 from bliss.controller.motor import CalcController
5
6 class EnergyCalc(CalcController):
7     def initialize(self):
8         """Load abacus data, to be able to do the calculation"""
9         ...
10
11     def calc_to_real(self, **positions):
12         """Take pseudo axis position, and return real axis position"""
13         ...
14
15     def calc_from_real(self, **positions):
16         """Return calculated axis position from real axis position"""
17         ...
18
```

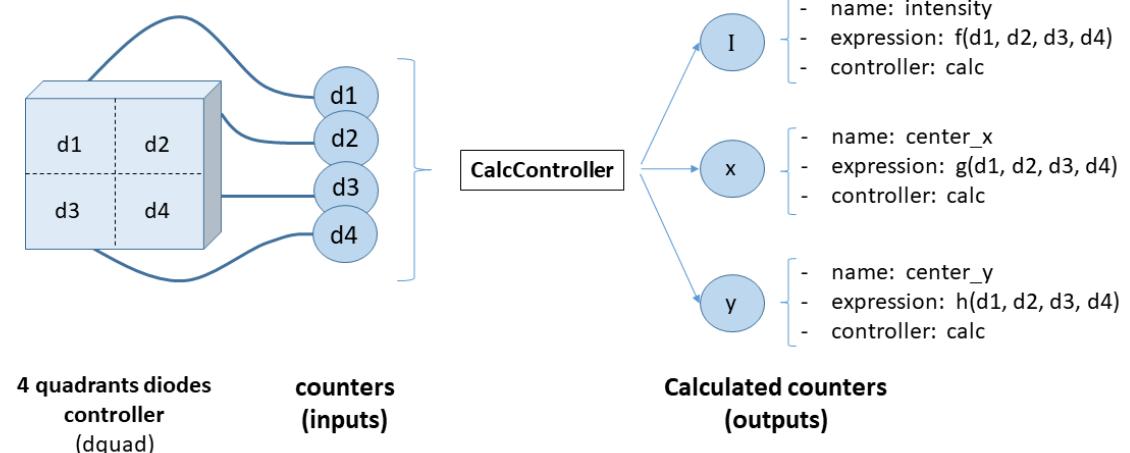
3.2 Software pseudo counters

In BLISS, a counter is a Python object representing an experimental parameter which can be measured during a scan

A Software counter takes multiple counters as **inputs** and produces multiple calculated counters as **outputs**

Software counters can be created in code, or declared from the BLISS configuration:

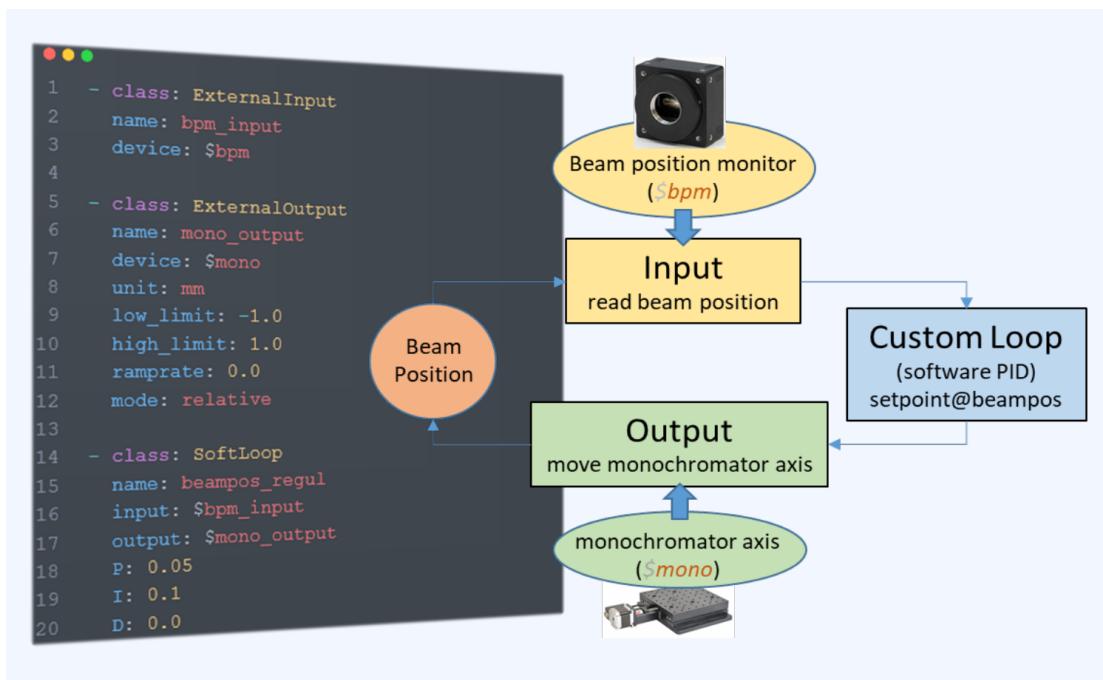
```
1 - class: ExpressionCalcCounterController
2 name: calc_diodes
3 inputs:
4     - counter: $diode1
5         tags: d1
6     - counter: $diode2
7         tags: d2
8     - counter: $diode3
9         tags: d3
10    - counter: $diode4
11        tags: d4
12 constants:
13     m : 0.5
14     n : 0.8
15 outputs:
16     - name: intensity
17         expression: (d1 + d2 + d3 + d4 )
18     - name: cen_x
19         expression: m*(d2-d1) + m*(d4-d3)
20     - name: cen_y
21         expression: m*(d3-d1) + m*(d4-d2)
```



4. Software regulation loops (see poster by P. Guillou tomorrow)

Example: beam position regulation

Compensation of low frequency drifts due to thermal load changes or mechanical instability



BLISS provides a Software Loop object, that knows how to regulate with PID parameters

Existing BLISS objects with "read" interface can be taken as Input, and objects with "set" interface can be taken as Output

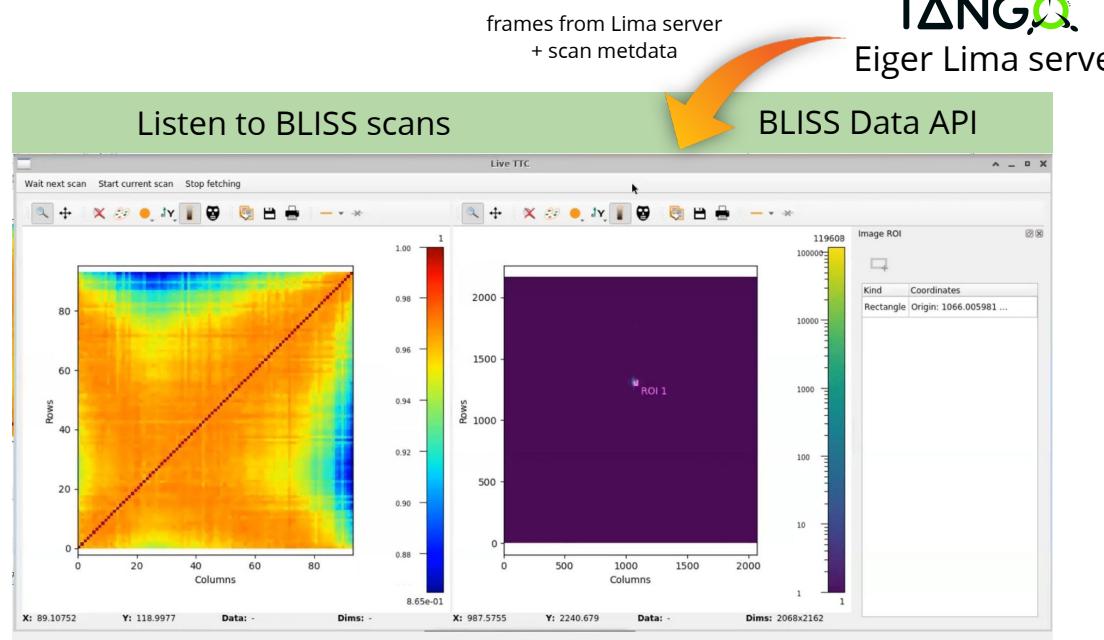
5. Accessing BLISS data (see poster by L. Felix tomorrow)

Example from ID10: measurement of coherent x-ray speckle patterns

User group from Tübingen University (Germany) developed a data acquisition and processing graphical interface on top of customized Flint and **BLISS data API**

Contributors: Linus Pithan, Vladimir Starostin, Prof. F. Schreiber

two-time correlation function (TTC) for XPCS (X-ray photon correlation spectroscopy) calculated online on NICE cluster



Conclusion

Conclusion



Wants to know more ? Come to try BLISS at the
BLISS satellite meeting on Thursday morning

Conclusion



Wants to know more ? Come to try BLISS at the
BLISS satellite meeting on Thursday morning

More than half of ESRF beamlines run
BLISS, all beamlines will be converted
at the horizon of 2024



Conclusion



Wants to know more ? Come to try BLISS at the
BLISS satellite meeting on Thursday morning

More than half of ESRF beamlines run
BLISS, all beamlines will be converted
at the horizon of 2024



BLISS has been designed to fulfill scientists needs and
synchrotron experiments control challenges

Conclusion



Wants to know more ? Come to try BLISS at the
BLISS satellite meeting on Thursday morning

More than half of ESRF beamlines run
BLISS, all beamlines will be converted
at the horizon of 2024



BLISS has been designed to fulfill scientists needs and
synchrotron experiments control challenges

Would be happy to start collaborations
around BLISS with interested people

