



BLISS: New experiments control for ESRF beamlines

PCaPAC 2018

*presented by Vincent Michel
Beamline Control Unit, Software Group
ESRF, Grenoble, France*



BLISS

BeamLine Instrumentation Support Software

The new control system for ESRF beamlines
currently under development

A global approach to run synchrotron experiments

- centralized configuration services
- support for many kinds of equipments
- advanced scanning engine

BLISS key concepts



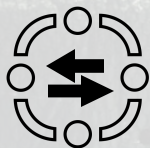
Relies heavily on **python**
and its ecosystem



All I/O based on **gevent**
cooperative multi-tasking



Direct hardware control



**Distributed control shared
state**



**Persistent settings and
transient data store**



**Scan acquisition chain,
represented as a tree**

User interfaces

Different interfaces
for the different
parts of the projects

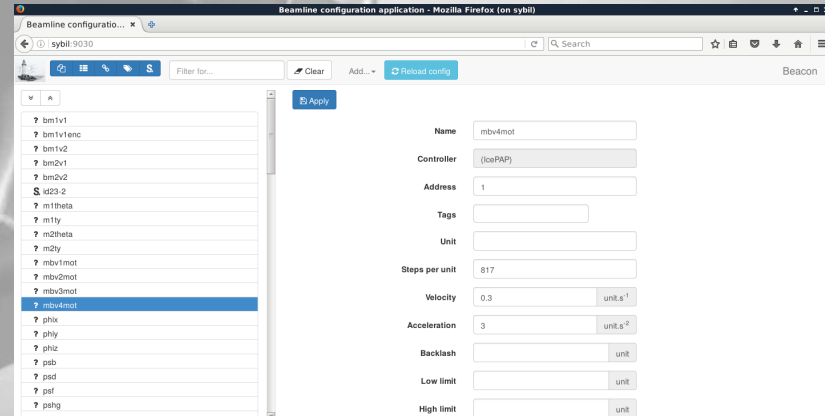
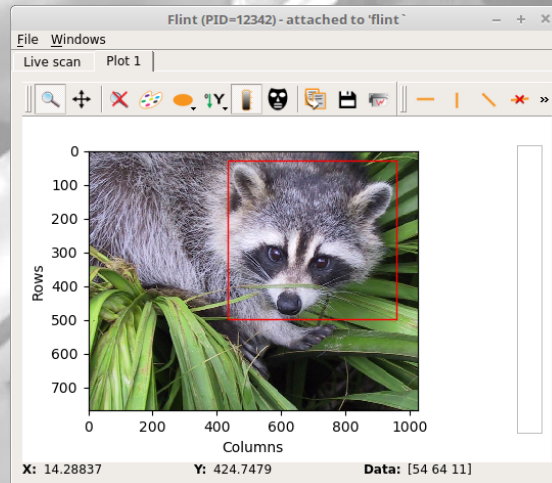
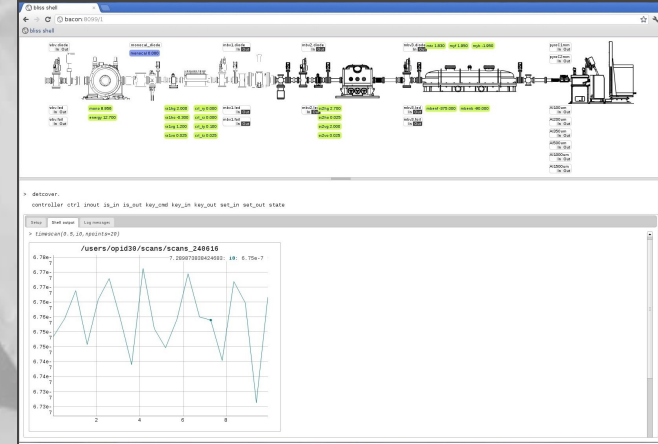
- Python shell
- Qt silx apps
- Web apps

```
matias@kashyyyk:~ % bliss -s test_session
test_session: Executing setup...
Initializing 'heater'
....
Initializing 'slhg'
Done.

>>> ascan(m1, 0, 10, 30, 0.1, diode, save=False)
Total 30 points, 3.0 seconds

Scan 4 Mon Sep 11 11:58:03 2017 <no file> test_session user = guijarro
ascan m1 0 10 30 0.1

# timestamp m1 diode
0 1.50512e+09 0 499.112
1 1.50512e+09 0.345 500.799
...
28 1.50512e+09 9.655 505.622
29 1.50512e+09 10 499.883
```



Project status

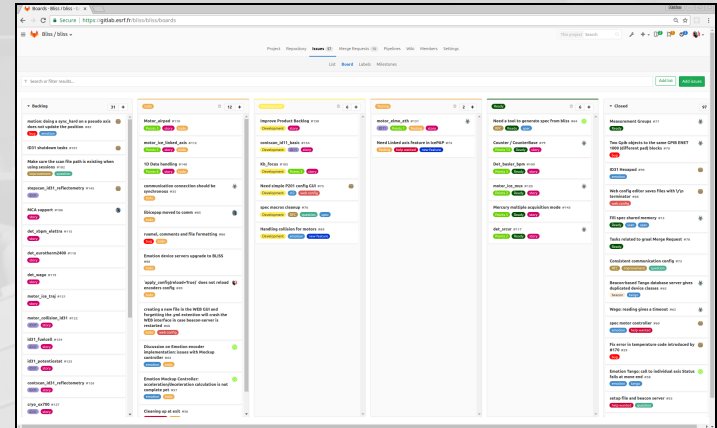
Current state of deployment

- Development version already deployed on several beamlines
- Full deployment planned for the end of the EBS program in 2021



Project is in active development

- First release - beginning of 2019
- Repository - gitlab.esrf/bliss/bliss



**Slides from
ICALEPCS 2017**



Extremely Brilliant Source (ESRF - EBS) project

- **150 M€ investment over the period 2015-2022**
- **4th generation light source**
- **100x improved brilliance and coherence of X-ray beams**
- **New state-of-the-art beamline portfolio**



Full details in talk: FRAPL07



BLISS

BeamLine Instrumentation Support Software



Why BLISS ?

spec: 26 years driving experiments at ESRF

- **Direct control of devices**
 - easier to debug
 - restarting = reset
- **Integrated tool**
 - configuration
 - controllers for all kinds of devices
 - plotting
- **Server mode to connect with external processes (GUI...)**
- **Commercial support**



spec: 26 years driving experiments at ESRF

- Poor macro language
- No extensibility
- Single task operation
- Exclusive hardware control
- Per-session configuration, no sharing
- No built-in continuous scan framework
- Limited data management
- No code ownership, less freedom



Limitations



Workarounds



Maintenance cost



The path to BLISS

- **Python library + tools**
- **Technical choices**
- **Beacon: services for BLISS**
- **Hardware control**
- **Scanning & data acquisition**
- **Data management**
- **Sequences as genuine Python functions**



BLISS Python library and tools

BLISS Python library and tools

Embed into any Python program

```
>>> from bliss.common.axis import Axis
>>> from bliss.controllers.motors import IcePAP
>>> iceid2322 = IcePAP.IcePAP("iceid2322",
                             {"host": "iceid2322"},
                             [{"mbv4mot", Axis, { "address": 1,
                                                  "steps_per_unit": 80,
                                                  "velocity": 125,
                                                  "acceleration": 500
                                                  }
                              }], [])

>>> iceid2322.initialize()
>>> m = iceid2322.get_axis("mbv4mot")
>>> m.velocity()
125.0
>>> m.acceleration()
500.0
>>> m.position()
252.23750000000001
>>>
```



BLISS Python library and tools

Command Line Interface based on [ptpython](#)

```
matias@kashyyyk:~ % bliss -s test_session
test_session: Executing setup...
Initializing `heater`
...
Initializing `slhg`
Done.

>>> ascan(m1, 0, 10, 30, 0.1, diode, save=False)
Total 30 points, 3.0 seconds

Scan 4 Mon Sep 11 11:58:03 2017 <no file> test_session user = guijarro
ascan m1 0 10 30 0.1

# timestamp m1 diode
0 1.50512e+09 0 499.112
1 1.50512e+09 0.345 500.799
...
28 1.50512e+09 9.655 505.622
29 1.50512e+09 10 499.883
```

BLISS Python library and tools

Configuration web application

The screenshot shows a web browser window titled "Beamline configuration application - Mozilla Firefox (on sybil)". The address bar shows "sybil:9030". The page has a navigation bar with a search box, a "Filter for..." input, a "Clear" button, an "Add..." dropdown, and a "Reload config" button. The main content area is divided into two sections. On the left is a list of motor configurations, with "mbv4mot" selected and highlighted in blue. On the right is a configuration form for the selected motor, with fields for Name, Controller, Address, Tags, Unit, Steps per unit, Velocity, Acceleration, Backlash, Low limit, and High limit. Each field has a corresponding input box or dropdown menu. The "Velocity" and "Acceleration" fields include unit indicators: $\text{unit}\cdot\text{s}^{-1}$ and $\text{unit}\cdot\text{s}^{-2}$ respectively. The "Backlash", "Low limit", and "High limit" fields also have unit indicators: "unit".

Motor Name	Controller	Address	Steps per unit	Velocity	Acceleration	Backlash	Low limit	High limit
mbv4mot	(IcePAP)	1	817	0.3 $\text{unit}\cdot\text{s}^{-1}$	3 $\text{unit}\cdot\text{s}^{-2}$	unit	unit	unit

BLISS Python library and tools

Graphical interface for users: interactive web shell

The screenshot displays the BLISS web shell interface. The top section shows a detailed schematic diagram of a particle detector system, including various components like diodes, magnets, and sensors, each with associated parameters and labels. Below the diagram, there is a command prompt where the user has entered `> detcover.` and a table of controller parameters:

controller	ctrl	inout	is_in	is_out	key_cmd	key_in	key_out	set_in	set_out	state
------------	------	-------	-------	--------	---------	--------	---------	--------	---------	-------

The bottom section shows a plot titled `/users/opid30/scans/scans_240616`. The plot displays a time scan with the following parameters: `7.289873838424683; i0: 6.75e-7`. The x-axis represents time points from 0 to 10, and the y-axis represents a value ranging from $6.73e-7$ to $6.78e-7$. The plot shows a fluctuating signal with a peak at approximately x=4.

The image shows a vast industrial interior, likely a power plant or refinery. The ceiling is high with a complex network of steel trusses and numerous overhead pipes. In the foreground, a long, narrow walkway with metal railings runs alongside a large, rectangular tank wrapped in silver insulation. The floor is polished and reflects the overhead lights. The overall atmosphere is industrial and technical.

BLISS technical choices

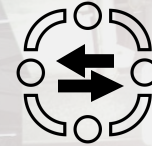
BLISS key concepts



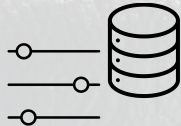
All I/O based on **gevent**
cooperative multi-tasking



Direct hardware control



Distributed control
ownership & shared state



Persistent settings
cache



Scan acquisition chain,
represented as a tree

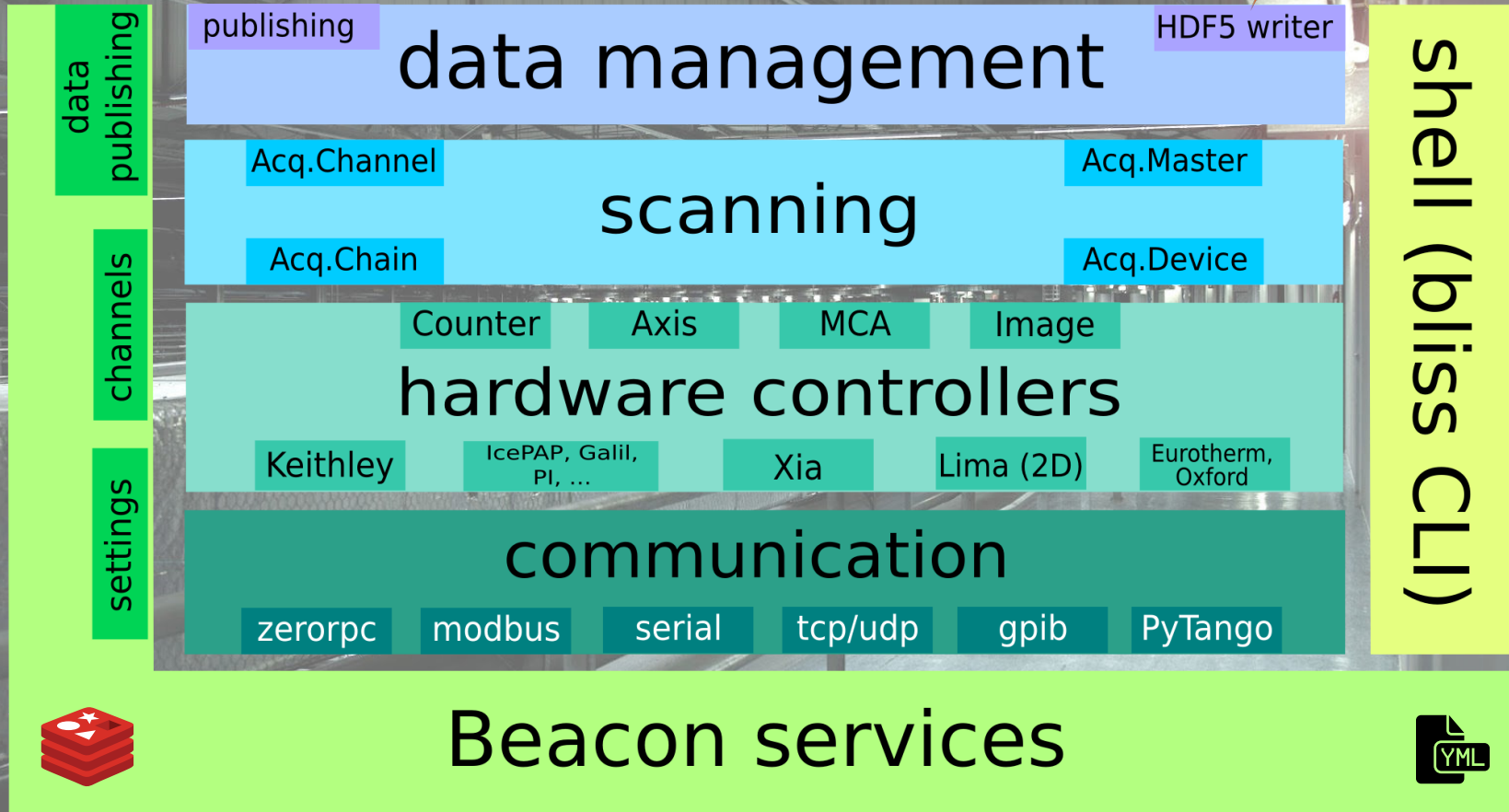


Transient data store

BLISS modular architecture

online data analysis
data visualisation

data archiving



**Beacon:
services for BLISS**



Beacon static configuration service

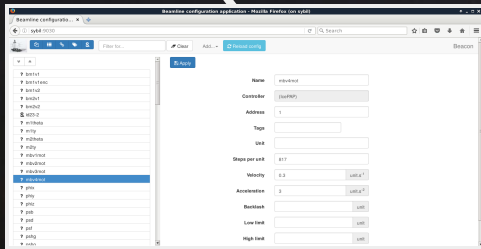
Devices & sequences configuration in YAML format



Beacon server



Sessions to group objects
Python **setup file**
User scripts



Web interface for configuration editing



Can **replace TANGO DB**
Conversion script provided

Beacon: example configuration

```
sybil:~/local/beamline_configuration % tree
```

```
.
├── beacon.rdb
├── eh
│   ├── diode.yml
│   ├── __init__.yml
│   └── motors
│       ├── bv.yml
│       ├── DtoX.yml
│       ├── __init__.yml
│       ├── md2.yml
│       ├── mirror1.yml
│       ├── slits.yml
│       └── table.yml
├── oh
│   ├── bpm.yml
│   ├── __init__.yml
│   └── motors
│       ├── bv.yml
│       ├── __init__.yml
│       ├── mono.yml
│       ├── slits.yml
│       └── transfocators.yml
├── wagos.yml
└── sessions
    ├── id232_setup.py
    ├── id232.yml
    └── __init__.yml
```

bv.yml: motor object

```
- controller:
  class: IcePAP
  host: iceid2322
  axes:
    - name: mbv4mot
      address: 1
      steps_per_unit: 817
      velocity: 0.3
      acceleration: 3
```

Beacon dynamic services

- Message broker
- state sharing
 - distributed lock

Beacon server,
services built on top of
 redis

Persistent
settings cache

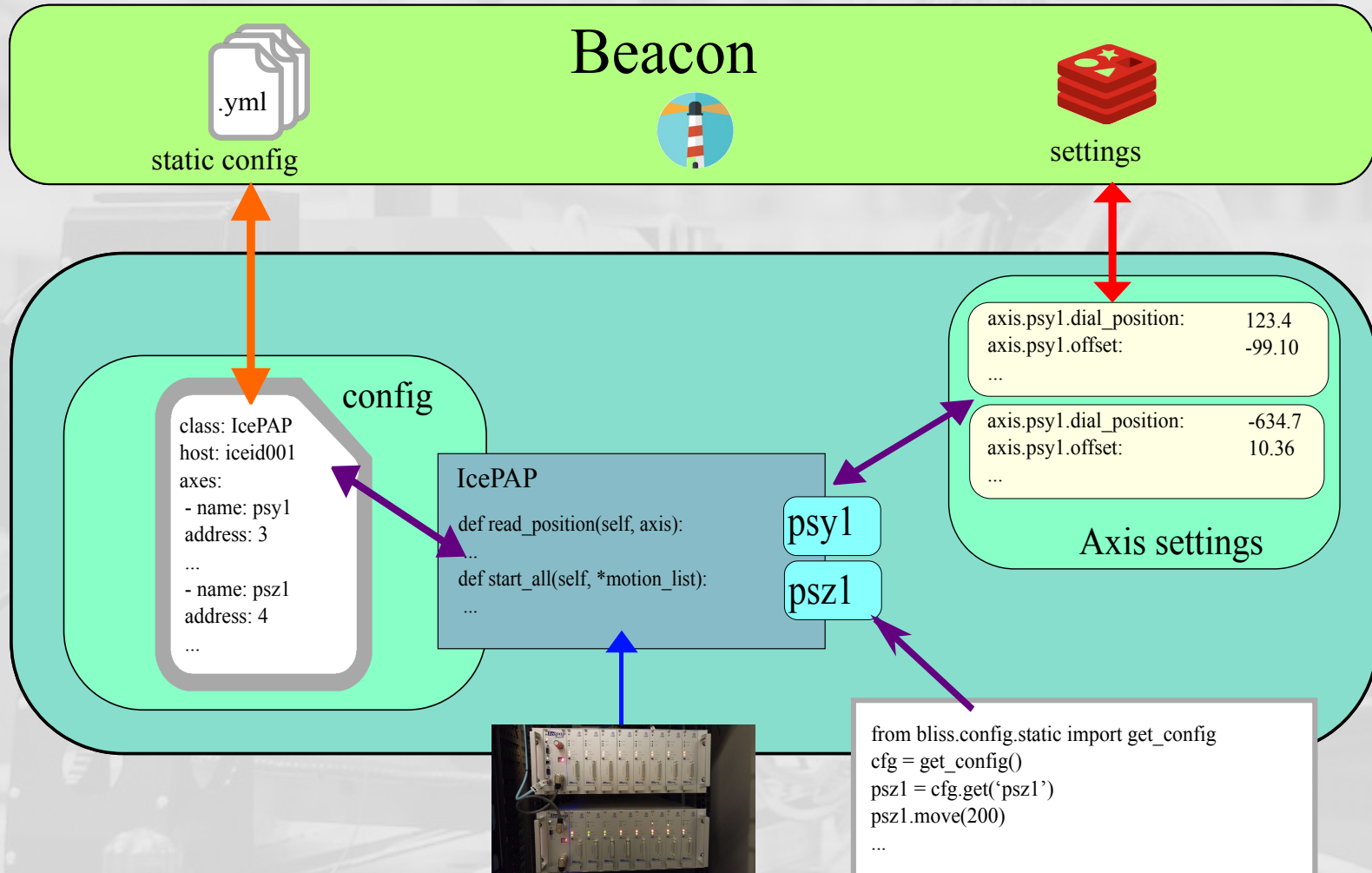
Transient data store



A black and white photograph of a man in a hoodie and glasses working on a complex piece of industrial hardware in a factory setting. The hardware is a large, dark metal structure with various cables and components. The man is looking down at the device, and his hands are near a large, rectangular metal mesh enclosure. The background is a blurred industrial environment with overhead lights and structural elements.

BLISS Hardware Control

Direct hardware control



Management of concurrent access

- Multiple BLISS processes means **concurrent access**
 - distributed control ownership
 - based on a protocol: ask Beacon for permission
- **State coherence**
 - hardware state is shared between all peers via **channels**

Management of concurrent access

BLISS process A

IcePAP
controller

psy1



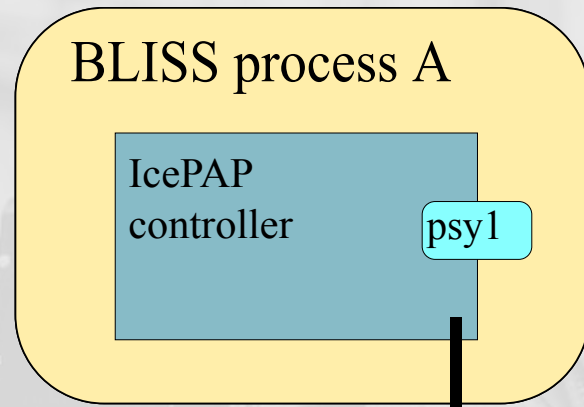
IcePAP
controller

psy1

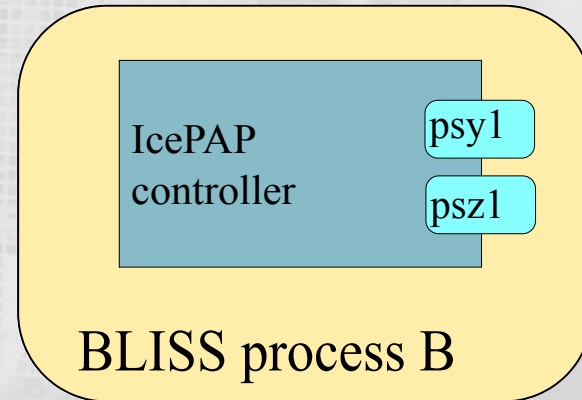
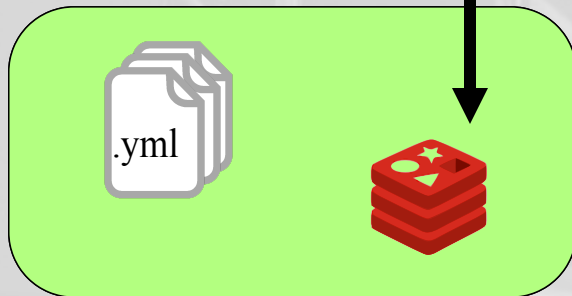
psz1

BLISS process B

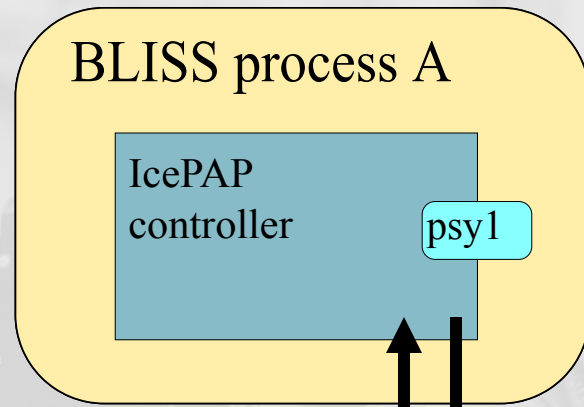
Management of concurrent access



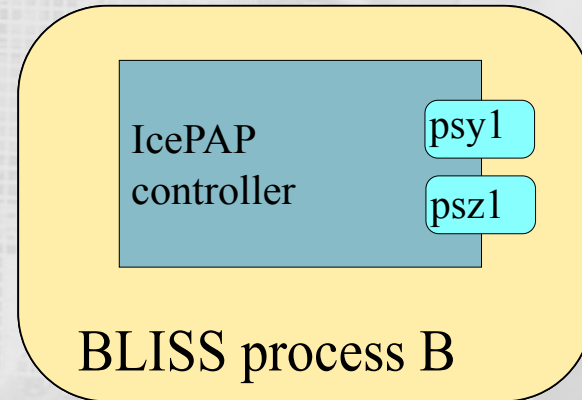
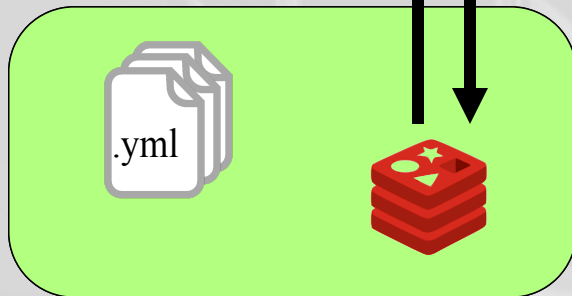
acquire lock



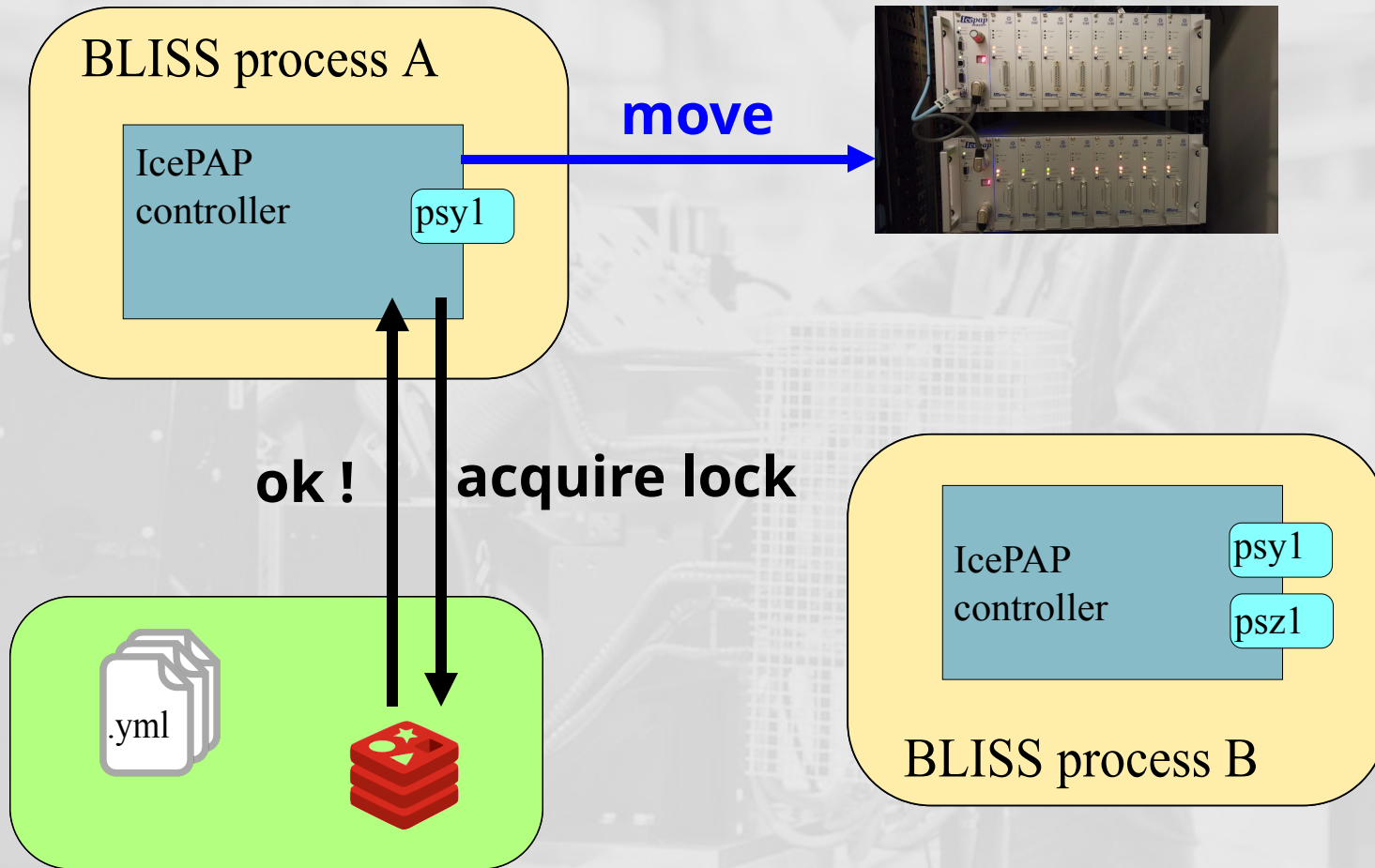
Management of concurrent access



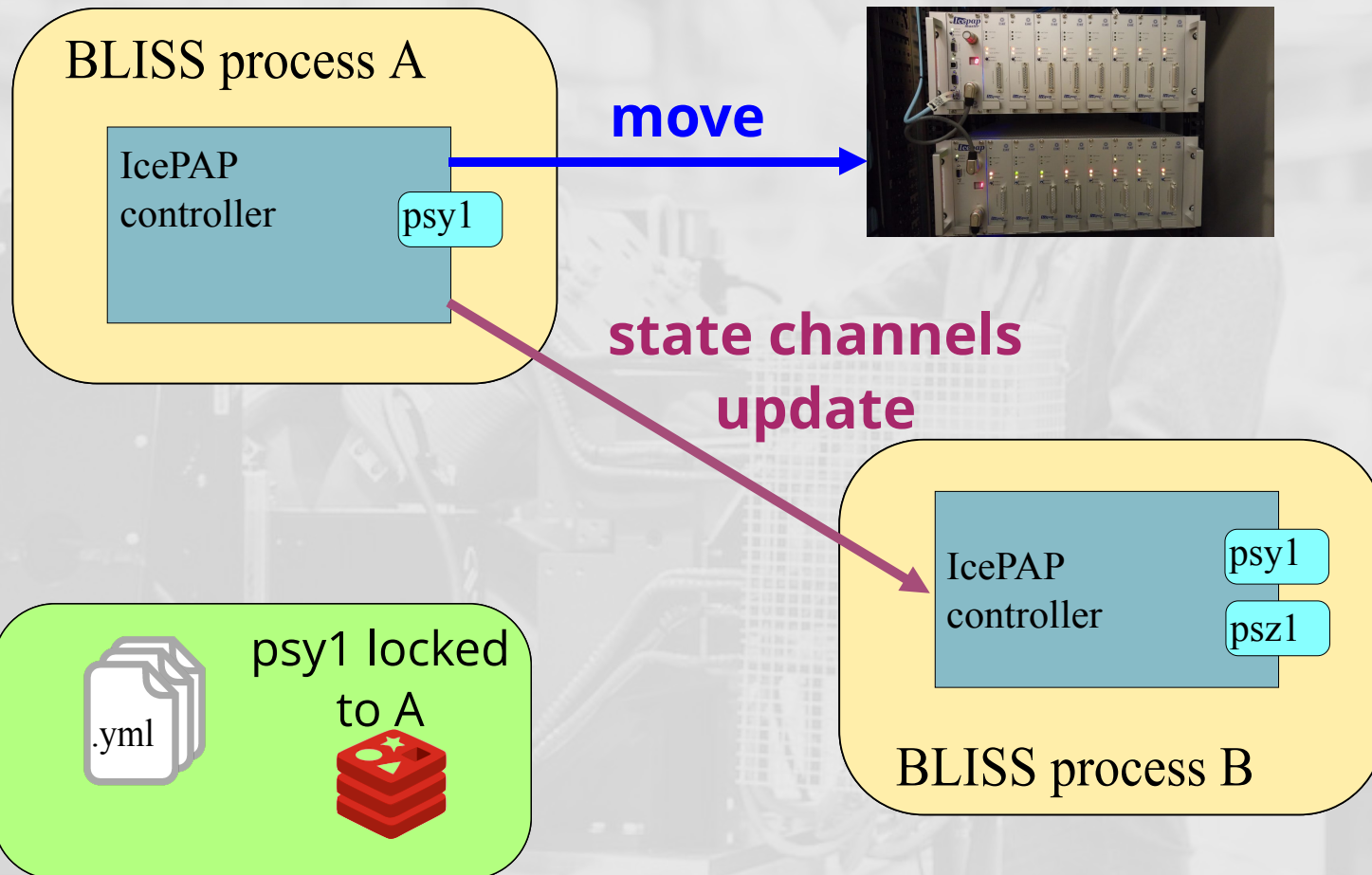
ok ! acquire lock




Management of concurrent access



Management of concurrent access



A black and white photograph of five ballerinas in a dance studio, captured in a synchronized leap. They are wearing black leotards and white tights. The studio has a wooden floor, a ballet barre, and a mirror. The text "BLISS scans" is overlaid in the center in a bold, blue font.

BLISS
scans

BLISS scans



- **Acquisition chain**

- a tree with master & slave nodes
- master triggers data acquisition
- slave takes data

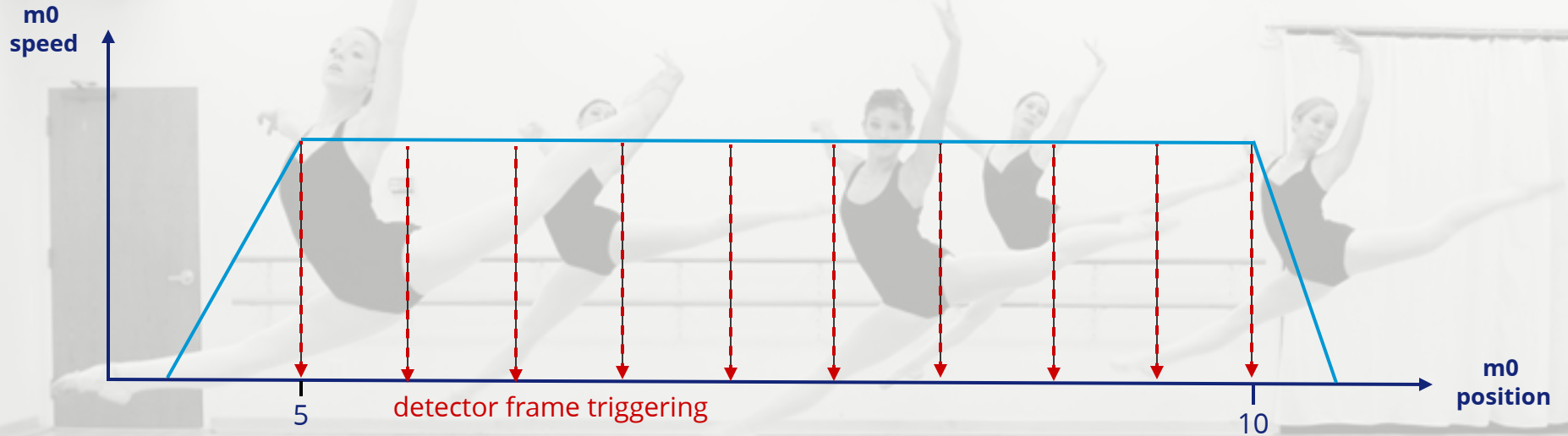
- **AcquisitionMaster, AcquisitionDevice**

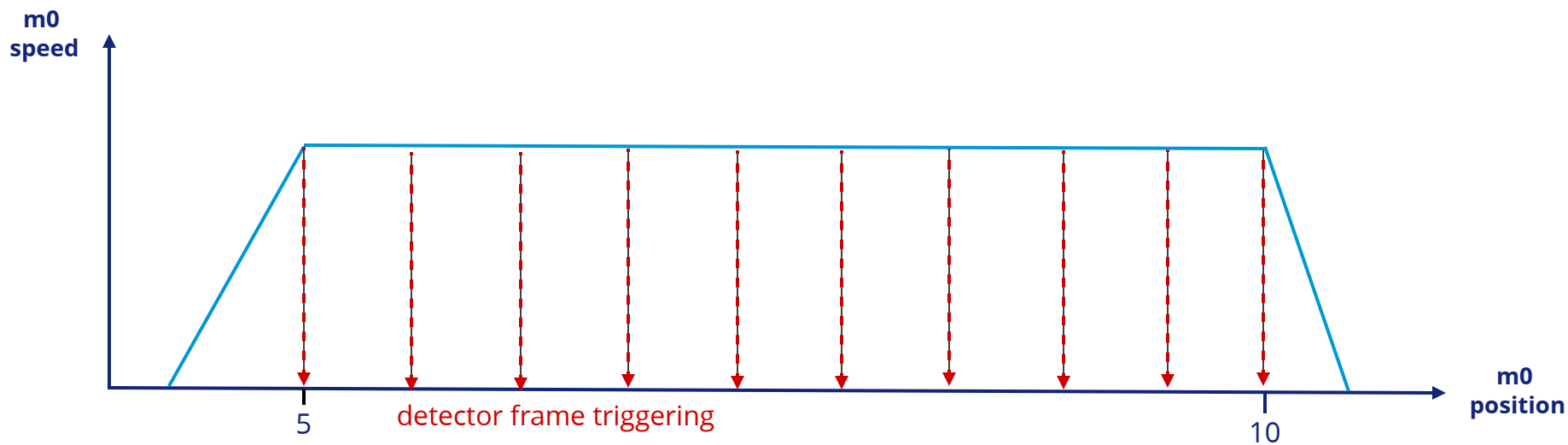
- wrappers around BLISS control objects

- **Data writer**

- HDF5

Continuous scan example





Continuous scan example

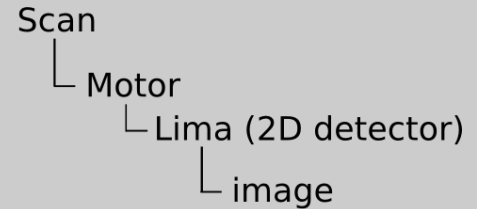
```
sybil:~ % bliss
>>> from bliss.scanning.chain import AcquisitionChain
>>> from bliss.scanning.acquisition.motor import SoftwarePositionTriggerMaster
>>> from bliss.scanning.acquisition.lima import LimaAcquisitionDevice
>>> from PyTango.gevent import DeviceProxy

>>> m0 = config.get("m0")

>>> lima_dev = DeviceProxy("id30a3/limaccd/simulation")

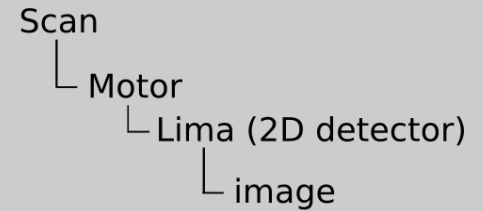
>>> chain = AcquisitionChain()

>>> chain.add(SoftwarePositionTriggerMaster(m0, start=5, end=10,
                                             npoints=10, time=5),
              LimaAcquisitionDevice(lima_dev, acq_nb_frames=5, acq_expo_time=0.03,
                                     acq_trigger_mode="INTERNAL_TRIGGER_MULTI"))
```



Continuous scan example

```
>>> SCAN_SAVING.template = '/data/id23eh2/inhouse/{date}/{sample}'
>>> SCAN_SAVING.sample = 'HAK1234'
>>> SCAN_SAVING.get_path()
"/data/id23eh2/inhouse/20170324/HAK1234"
>>> from bliss.scanning.scan import Scan
>>> my_continuous_scan = Scan(chain)
>>> my_continuous_scan.start()
```



Classic step-by-step scans

- **Directly available as functions** from '*bliss.common.standard*'
 - Example: *ascan(axis, start, stop, npoints, count_time, *counters)*
- **Default acquisition chain**
- **Use the same underlying framework** as continuous scans


A photograph of a warehouse interior. The scene is filled with white metal shelving units arranged in rows. Each unit has three shelves, and they are all filled with stacks of grey cardboard boxes. The boxes are of various sizes and are stacked in a somewhat organized but dense manner. The floor is a smooth, light-colored concrete. The lighting is bright and even, coming from above. In the center of the image, the words "Data Management" are written in a large, bold, blue sans-serif font, overlaid on the boxes and shelves.


Data Management

Model for organizing acquired data

- **Mirroring of the Acquisition Chain tree**
 - each device in the chain **has a name**
 - each device define 1 or more **'AcquisitionChannel' objects**
- **Acquisition channels**
 - must have **a name, a type and a shape**
- **Metadata**
 - *scan_info* dictionary ({ key: value, ... }) associated with scans

Online data publishing

- While a scan is running, **data is published** to the redis database provided by Beacon 
 - scalar values are **stored directly**
 - bigger data (images, spectra) is **just referenced**
 - configurable time to live (TTL)
- Any external process can access redis data to perform **online data analysis**, for example



User Sequences

Sequences as Python functions

```
from bliss import * # imports generic scans, cleanup functions, etc
from bliss.setup_globals import * # imports objects from session (setup)
import numpy # I know you dreamt of it
import gevent

def set_detector_cover(in):
    wcidxx.set('detcover', in)

    # 5 seconds timeout waiting for detector cover to move
    with gevent.Timeout(5):
        while wcidxx.get('detcover_in') == in:
            time.sleep(0.1)

def my_super_experiment(name):
    safety_shutter.open()

    old_att = attenuators.get()

    def restore_beamline():
        set_detcover_open(False)
        attenuators.set(old_att)

    with cleanup(safety_shutter.close): # cleanup is always called at the end
        with error_cleanup(restore_beamline): # this will only be called in case of error
            attenuators.set(50)
            set_detcover_open(True)

            SCAN_SAVING.name = name
            MEASUREMENT_GROUP.enable('diode')

            data_node = dscan(m0, -5, 5, 10, 0.1)

            for data in data_node.walk_data():
                # do something useful with data...
```


Sequences as Python functions

```
from bliss import * # imports generic scans, cleanup functions, etc
from bliss.setup_globals import * # imports objects from session (setup)
import numpy # I know you dreamt of it
import gevent
```

Easy timeouts with gevent.Timeout

```
def set_detector_cover(in):
```

```
    # 5 seconds timeout waiting for detector cover to move
    with gevent.Timeout(5):
        while wcidxx.get('detcover_in') == in:
            time.sleep(0.1)
```

```
def my_super_experiment(name):
    safety_shutter.open()
```

Normal Python functions

```
    old_att = attenuators.get()
```

```
    def restore_beamline():
        set_detcover_open(False)
        attenuators.set(old_att)
```

Use of Python context managers for cleanup

```
    with cleanup(safety_shutter.close): # cleanup is always called at the end
        with error_cleanup(restore_beamline): # only called in case of error
            ...
```

```
    data_node = dscan(m0, -5, 5, 10, 0.1)
```

```
    for data in data_node.walk_data():
        # do something useful with data...
```

A scenic view of a snow-capped mountain range under a clear blue sky. In the foreground, a stone staircase leads up towards the base of the mountains. The word "Conclusion" is overlaid in the center of the image.

Conclusion

Project state

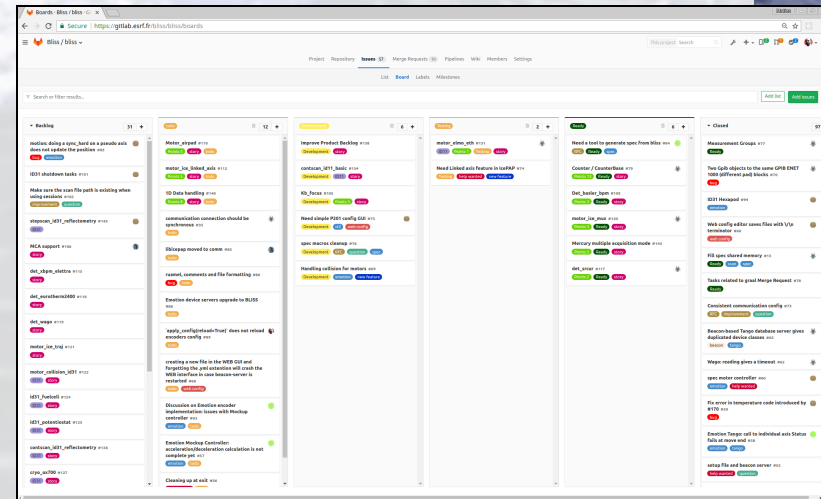
- **Current state of deployment**

- MX beamlines are already running BLISS
- 3 more beamlines (Materials Science) for the end of the year

- **Full deployment in 2020**

- **Project is in active development**

- Not ready for use outside ESRF yet
- [git repository](#)



Conclusion

- **Long term project** for EBS beamlines
- Control paradigm: **keep what works, add new concepts**
- Python **scanning framework**
- Prepared for **current and future challenges**
 - scans with online feedback
 - data management
 - evolutive platform

Acknowledgements



+ ESRF BCU contributing members: A. Beteva, M.C.Dominguez, M. Perez, J. Meyer

ESRF Software Group: A. Goetz