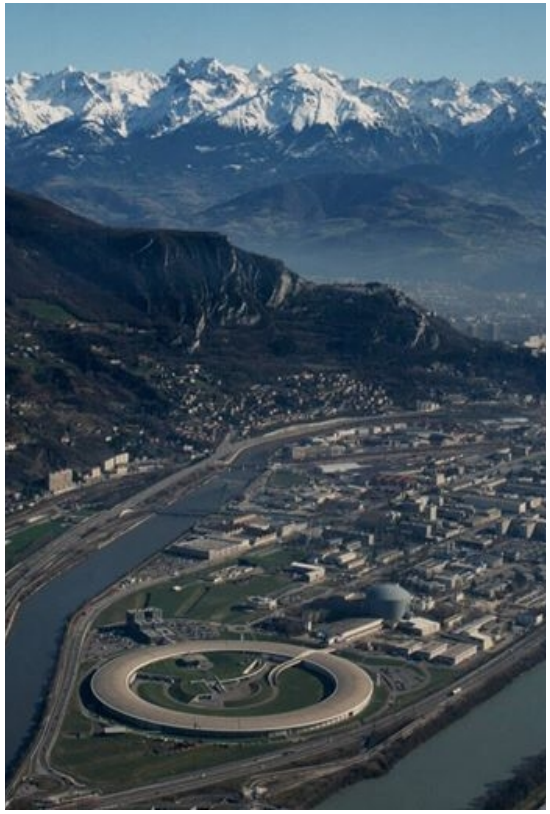
The European Synchrotron

BLISS PROJECT OVERVIEW



Outline

- Bliss package content
- Bliss installation
- Bliss core and frameworks
- Bliss scanning engine

BLISS PACKAGE: EASY TO INSTALL

- Open repository on gitlab: https://gitlab.esrf.fr/bliss/bliss
- A pure Python package
- Install with a single command line

```
(base) pguillou@linguillou:~$ cd local/bliss.git/
(base) pguillou@linguillou:~/local/bliss.git$ make
################################################################################
MAKEFILE TO CREATE AND CONFIGURE CONDA ENVIRONMENTS FOR YOU.

IMPORTANT: All commands are LAZY, no package is updated unless it is strictly
required. It is up to you to update a particular package, or to start a fresh
environment to get most recent version of everything.

COMMANDS:
    make bl_env [NAME=<name>] [PYTHON_VERSION=<version>]
        (typical ESRF beamline installation)
        - Creates a conda environment for BLISS with the name you provide (default: bliss_env).
        - Bliss and blissdata are installed from sources (pip -e).
    make dev_env [NAME=<name>] [LIMA_NAME=<name>] [LIMA2_NAME=<name>] [MOSCA_NAME=<name>] [PYTHON_VERSION=<version>]
        (use this command to set up an ideal environment to develop with BLISS)
        - Creates a conda environment for BLISS with the name you provide (default: bliss_env).
        - Bliss and blissdata are installed from sources (pip -e)
          with dev and test tools like black and pytest.
        - Creates a conda environment with the name you provide (default: lima_env)
        - Lima simulator plugins are installed from source (pip -e).
        - Creates a conda environment with the name you provide (default: lima2_env)
        - Creates a conda environment with the name you provide (default: mosca_env)
    make demo_env [NAME=<name>] [LIMA_NAME=<name>] [LIMA2_NAME=<name>] [MOSCA_NAME=<name>] [PYTHON_VERSION=<version>]
        (use this command to set up an ideal environment to demonstration of BLISS)
        - Creates a conda environment for BLISS with the name you provide (default: bliss_env).
        - Bliss, blissdata and blissdemo are installed from sources (pip -e).
        - Creates a conda environment with the name you provide (default: lima_env)
        - Lima simulator plugins are installed from source (pip -e).
        - Creates a conda environment with the name you provide (default: lima2_env)
        - Creates a conda environment with the name you provide (default: mosca_env)
    make demo_with_oda_env [NAME=<name>] [LIMA_NAME=<name>] [LIMA2_NAME=<name>] [MOSCA_NAME=<name>] [ODA_NAME=<name>] [PYTHON_VERSION=<version>]
        (use this command to set up an ideal environment to demonstration of BLISS)
        - Creates a conda environment for BLISS with the name you provide (default: bliss_env).
        - Bliss, blissdata and blissdemo are installed from sources (pip -e).
        - Creates a conda environment with the name you provide (default: lima_env)
        - Lima simulator plugins are installed from source (pip -e).
        - Creates a conda environment with the name you provide (default: lima2_env)
        - Creates a conda environment with the name you provide (default: mosca_env)
        - Creates a conda environment with the name you provide (default: oda_env).
        - Blissdemo is installed from source (pip -e) with worker dependencies.
################################################################################
(base) pguillou@linguillou:~/local/bliss.git$
```

BLISS PACKAGE: A LOT MORE THAN JUST A SCANNING ENGINE

A **complete and modular package**, providing tools for:

- **Configuration files** management
- **Hardware control** and **data acquisition**
- Data **storage in RAM**
- Data **archiving** (writing files)
- Live **data access**
- Live **data visualization**
- Interactive **terminal**
- **Web** terminal and UI

Beacon Server

Serves bliss objects configuration files

Bliss Core

Device control and frameworks

Scanning engine

Data publication to REDIS

BlissData

Common API to access data from

- REDIS
- HDF files
- Other locations via plugins

BlissWriter

Data archiving from REDIS to HDF5 (Nexus)

FLINT

GUI for live data visualization

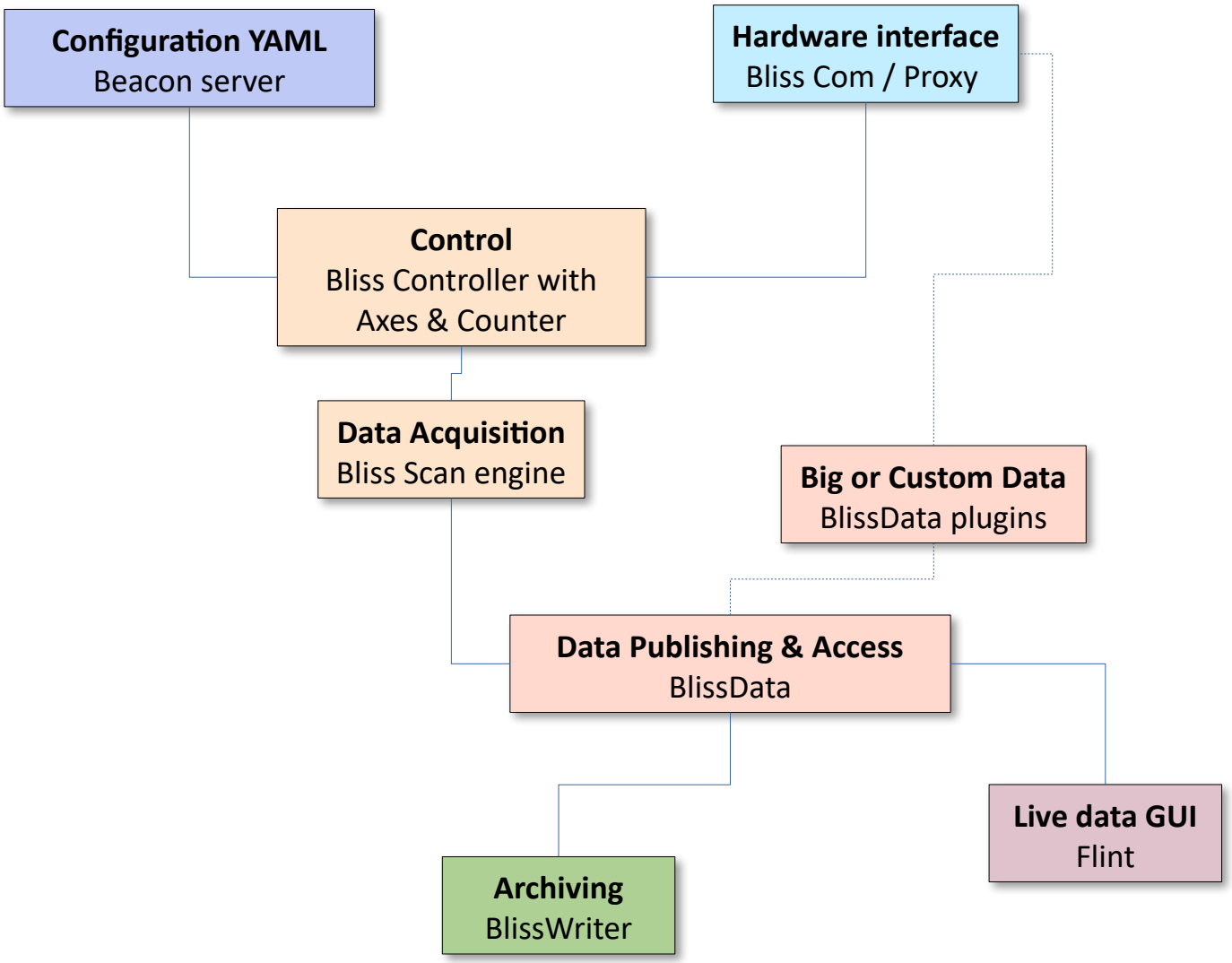
BlissTerm

Bliss as a server

Terminal as a web application

Simple web UIs

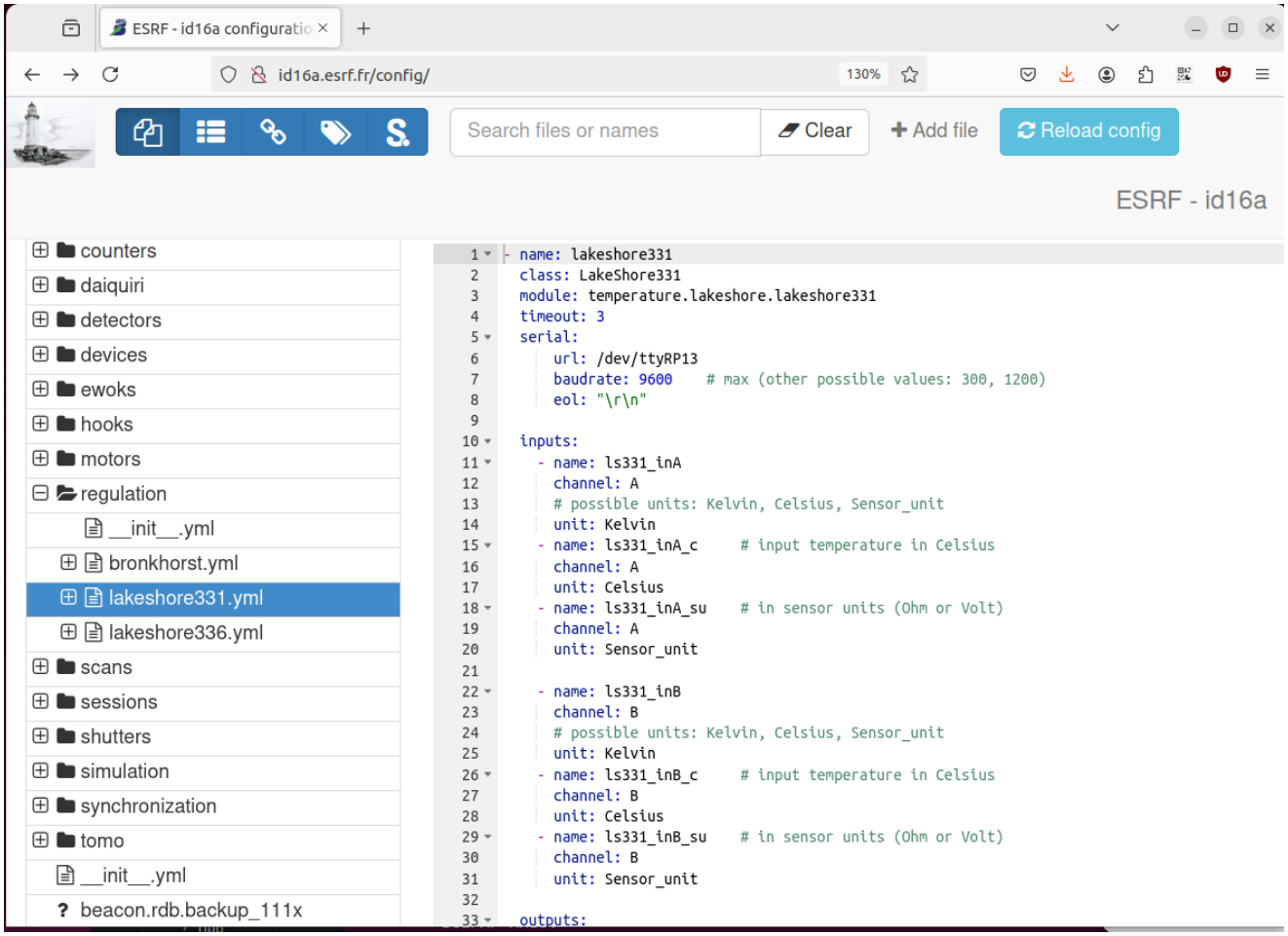
BLISS A MODULAR PACKAGE



BEACON SERVER: BLISS OBJECTS CONFIGURATION FILES

A **unique data base** holding all **BLISS objects configuration files** (YAML)

Directly **edit YAML** files **or** use the **web interface** to declare devices and user sessions



BLISS CORE: VARIOUS PLUG AND PLAY CONTROLLERS

Many **hardware** controllers already **implemented** and **ready to use**

- **Motors:** Aerotech, Newport, PI, Zaber, Elmo, Galil, flex, micos, …
- **Pressure, Temperature, Power supply:** Eurotherm, Lakeshore, Oxford, Linkam, ...
- **1D Multi-channels:** FalconX, Mercury, OceanOptics, Hamamatsu, …
- **2D Lima detectors:** Eiger, Andor, Prosilica, Basler, Pilatus, Maxipix, ...

```
- class: icepap
  plugin: emotion
  host: iceid163
  axes:
    - name: mot3
      address: 28
      steps_per_unit: 6555.5555
      velocity: 2
      acceleration: 4
      user_tag:
        - dial.anticlockwise

- name: ximea
  plugin: bliss
  class: Lima
  tango_url: id16ni/limaccd/ximea
  saving:
    file_format: HDF5
  disable_bpm: true
  prepare_timeout: 60
  tango_timeout: 60
  disable_bpm: True
```

Simply declare the object in a YML configuration file and it will be ready to use in your Bliss session

```
PERC [4]: mot3
 Out [4]: AXIS mot3

          position   dial      offset   sign  steps_per_unit  tolerance
          46.7200    -46.7200  0.0000   -1    100.00          0.0001

                                        CURRENT VALUE   |  CONFIG VALUE
                                        -------------   |  ------------
          limits [low ; high]           [ inf ; -inf]   |  [ inf ; -inf]
          velocity                      1.0             |  1.0
          velocity limits [low ; high]  [ inf ; inf]    |  [ inf ; inf]
          acceleration                  1.0             |  1.0
          acctime                       1.0             |  1.0
          backlash                      0.0000          |  0.0000

          STATE(s): READY

          ICEPAP AXIS:
               address: 3
               axis status: POWER: ON    CLOOP: OFF    WARNING: SSI error    ALARM: NO
               IcepapEncoders(ENCIN='0', ABSENC='16744082', INPOS='305533071', MOTOR='-4672', AXIS='-4672', SYNC='-3262')
```

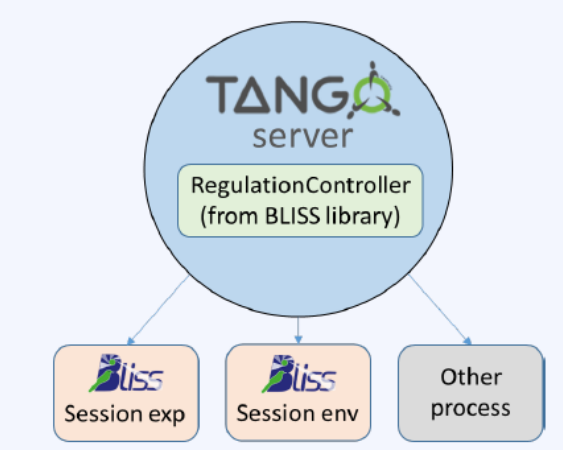
BLISS CORE: CONTROLLERS FRAMEWORK

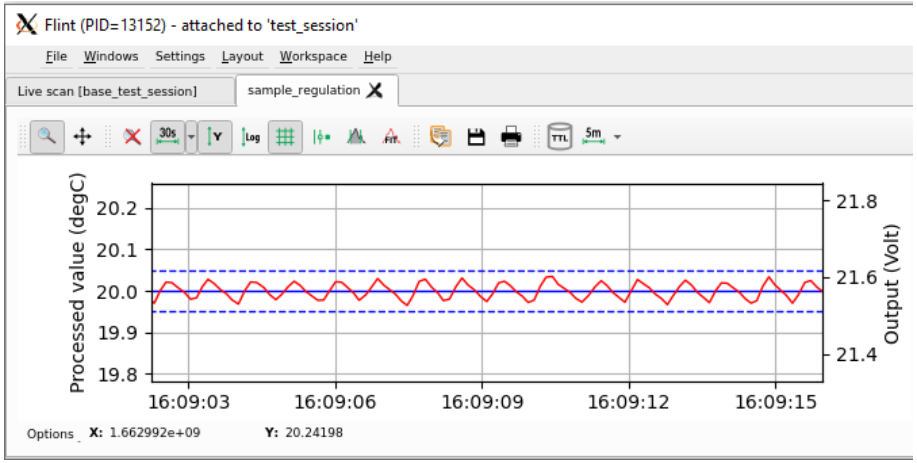
Similar devices regrouped into **frameworks** with a **common API**

Hides each controller **specific interface** and provides **dedicated features**.

- **Axis**: real axes, calculated axes, soft-axes
- **MCA / MOSCA**: Multi-channels 1D detectors
- **LIMA / LIMA2**: 2D detectors
- **Regulation**: PID-Loops / Inputs / Outputs

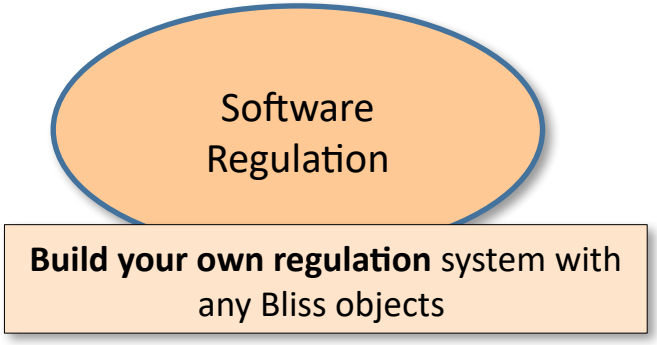
- ✓ **Special Feature examples**
- ✓ Generic Tango server for multi-client access
- ✓ Monitoring plots
- ✓ Axis-like set-point for scanning

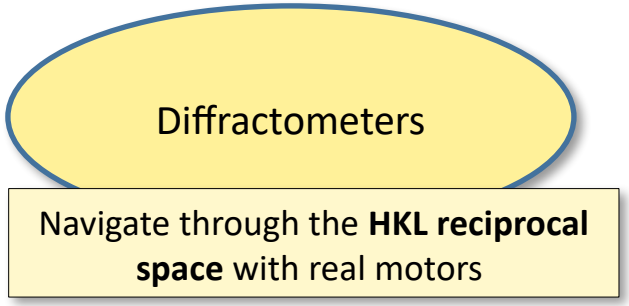


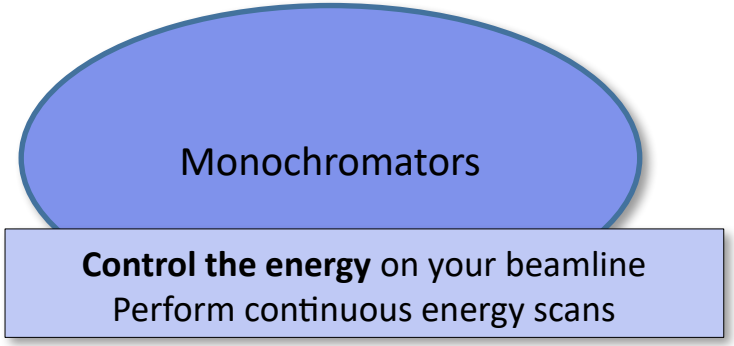


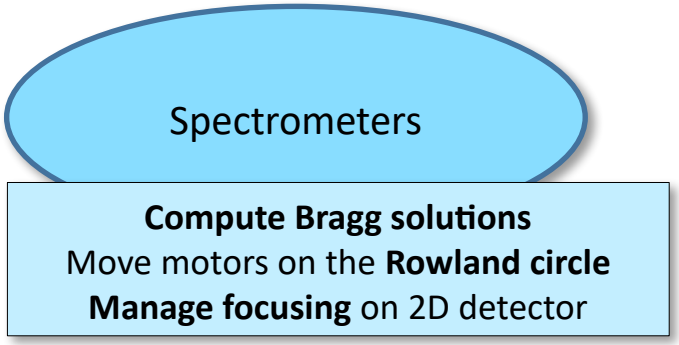
BLISS CORE: ADVANCED SCIENTIFIC FRAMEWORKS

Complex Beamline control **setup** that can be generalized are **integrated into Bliss** core for the benefit of all the community and for a better software maintenance.









BLISS CORE: MANY USEFUL OBJECT AND HELPERS

BLISS provides a **lot of objects designed for Beamline** purposes.

Users can **add** to their **sessions** and customize **via** the **configuration** files or at runtime:

- **Slits** / Shutters / Interlocks / FilterWheel / Switches ...
- Motion**Hooks** / ScanPreset
- Encoders / **VirtualAxis** / SoftAxis / MultiPositions
- **Logging** tools / **Debugging** tools / Prdef / Pprint
- Smart **refreshed prints** (context manager) / Formatted prints / Colors
- **Interaction with** plotted **curves** via Flint (goto-peak, ROIs creation, …)
- Connection to **Tango** servers (Device**Proxy**)
- Loading and playing **users macros** (globals protected, user env_dict)

BLISS CORE: COMMUNICATION LAYER

Bliss already implements **various communication protocols**:

- Serial line
- Ser2net
- TCP socket
- UDP socket
- GPIB
- Modbus
- VXI11
- SCPI

YAML example

```yaml
class: FooController
name: foo
serial:
  url: /dev/ttyS0
```

```python
from bliss.comm.util import get_comm, SERIAL

class FooController:
    def __init__(self, config):
        self.config = config
        self._comm = None

    def _init_com(self):
        default_options = {'baudrate': 19200}
        self._comm = get_comm(self.config, ctype=SERIAL, **default_options)
```

BLISS CORE: ONE CMD TO INSTANTIATE THEM ALL

Instantiate all **communication** object **from** YML **configuration** easily with a unique helper command

- Usual default parameters provided
- Overwrite defaults via the configuration
- Check/validate com type and options passed via the configuration

```
gpib:
  url: tango_gpib_device_server://id42/gpib_lid421/0
  pad: 13
  timeout: 10.
```

```
class: FooController
name: foo
serial:
  url: /dev/ttyS0
```

"com = get_comm(config)"

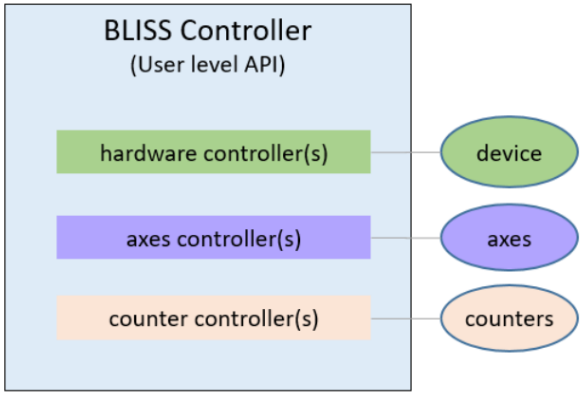
```
tcp:
  url: 160.103.99.42
  eol: "\r\n"
```

Insta ation easily with '**get_comm()**' **helper**

```
plugin: keithley
keithleys:
  - name: k_pico1
    model: 6485
    tcp-proxy:
      external: False
      tcp:
          url: id14serial1.esrf.fr:9001
          timeout: 3
```

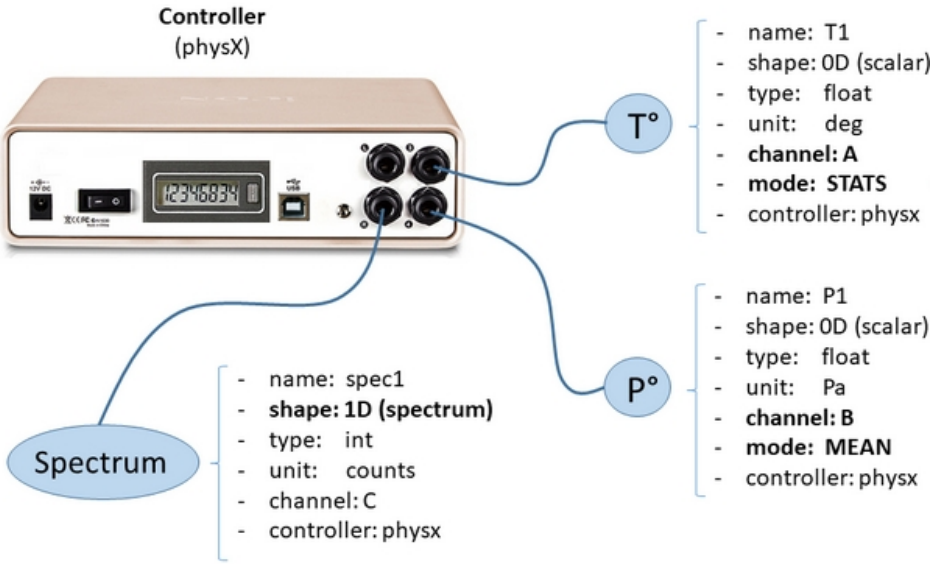
BLISS CORE: HW DEVICE AND BLISS CONTROLLER

To be used in a ***Scan***, the **controller** of a device must **expose *Axes*** or ***Counters***



Users can **declare** and customize **counter names** from controller **configuration**

```
plugin: ct2
class: CT2
name: p201
address: tcp://lid00c:8909
clock: CLK_100_MHz
type: P201
external sync: {}
timer:
  counter name: sec
counters:
- address: 3
  clock source: CLK_10_KHz
channels:
- address: 1
  level: NIM
- counter name: freq_gen
  address: 2
- counter name: clk
  address: 3
```



BLISS CORE: BASE OBJECTS FOR SCANNING

Axis

Anything that can be **moved** to a target position

Counter

Defines something that can be **measured**
(ex: temperature 0D, spectrum 1D, image 2D)

Scan

The scanning procedure

Moves **axes** to a list of **positions**
Performs measurements with a list of **counters** for a given **counting time**

Ex: **ascan(axis, start, stop, steps, count_time, cnt1, cnt2, ...)**

AcquisitionChain

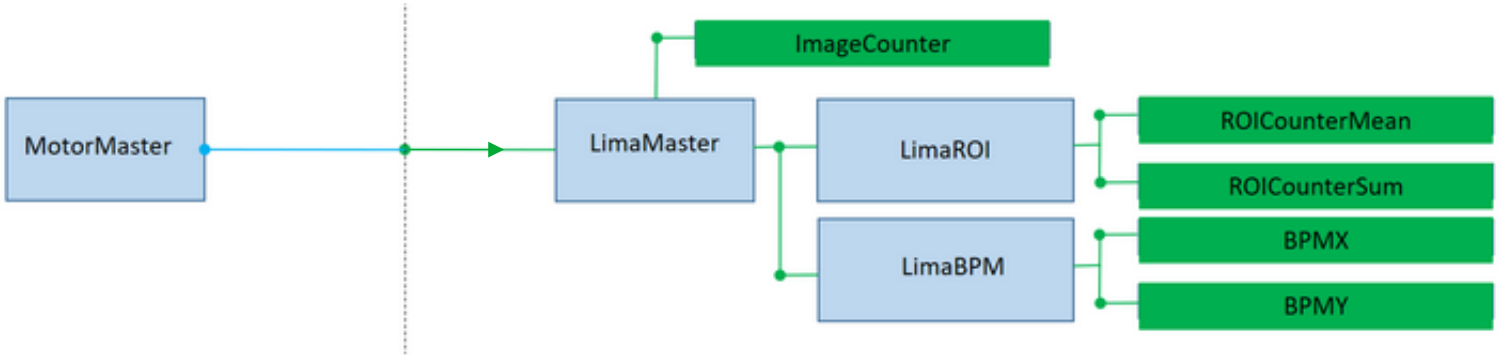
Trigger the devices and **gather** acquisition **data**

BlissData

Publish acquisition **data** (buffering / streaming to clients)

BLISS CORE: STEP BY STEP SCANS

- A **wide range of default step by step scans** (Count, Loop, Mesh, LookUp, ...)
- **Auto-resolution** of the **acquisition parameters** of involved devices
- **Customize default acquisition parameters** of each device
- **Synchronization devices** can be **automatically introduced** on top of any device
- **Scan presets** can be easily added to perform actions BEFORE and AFTER

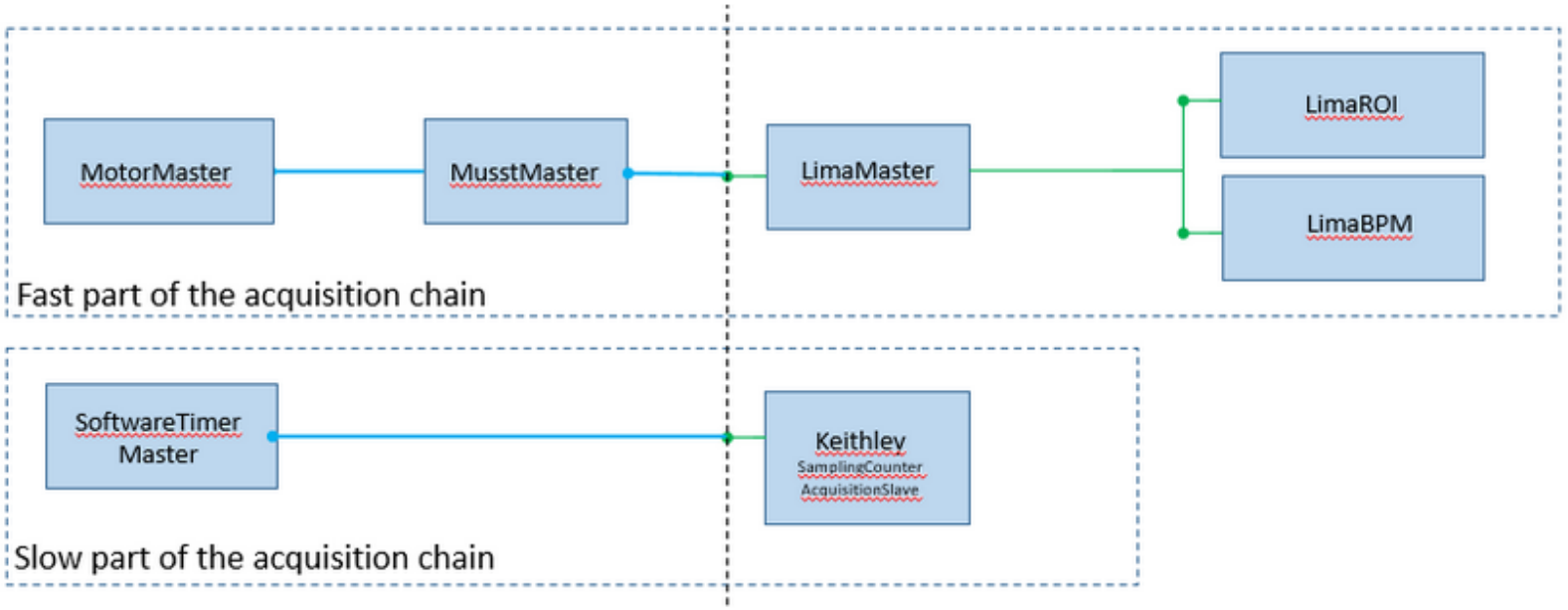


AcquisitionChain reflects the devices **triggering hierarchy**

Bliss standard scans automatically build the chain in the step by step context

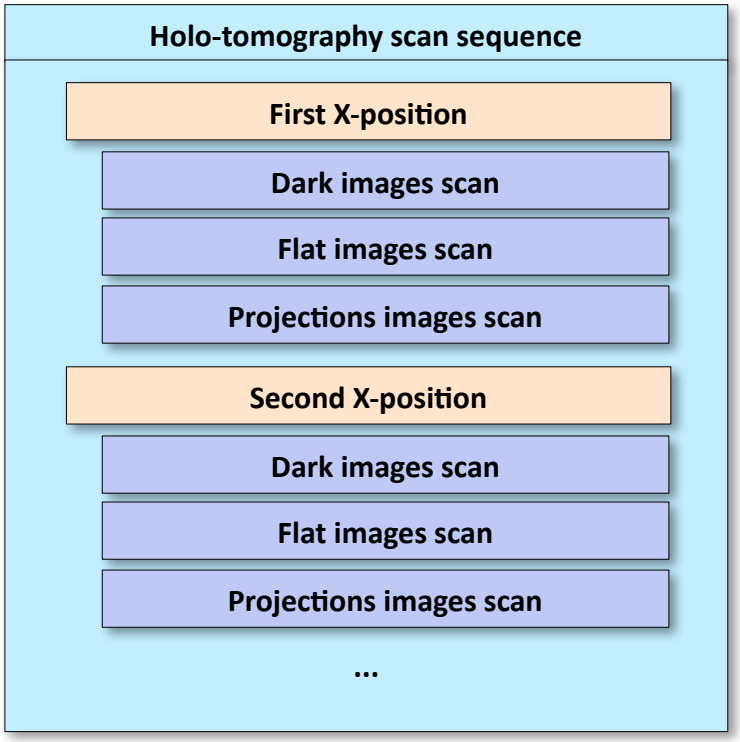
BLISS CORE: CONTINUOUS SCANS

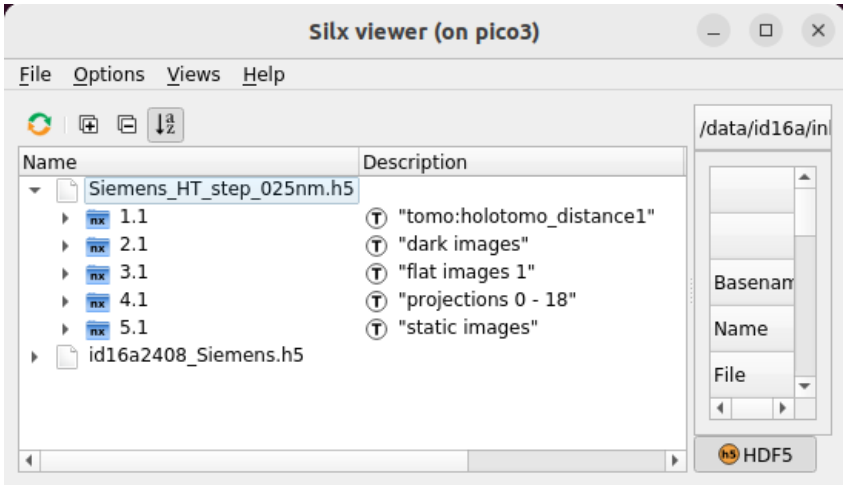
- During a **continuous scan**, **data** is acquired **while** axes are **moving**
- All the device **triggering** is **done via hardware signals**
- **Data** are **gathered** and published **asynchronously** by BLISS
- **Continuous** scans **use same objects** as any scan and are fully customizable
- Acquisition chain **can mix Fast and Slow chain**



BLISS CORE: SCANS SEQUENCE

- A **scan sequence is a scan performing sub-scans**
- Scans sequences can be **nested**
- Saved **data** automatically **reflects sequence structure**
- Allow **complex nested scanning procedures**





CONCLUSION

- BLISS is a complete python package for control, acquisition and storage
- BLISS provides many plug and play hardware controllers
- BLISS provides many tools for experimental science
- BLISS decouples data acquisition from data access
- BlissData open the door to the community