# A Taurus GUI for Automated BioSAXS Experiements at the CoSAXS Beamline

Lukas Wittenbecher[1], Hanno Perrey[2], Lin Zhu[1], Miaoxin Gong[3], Benjamin Folsom[1]

(1) MAX IV Laboratory, Lund, Sweden

(2) European Spallation Source, Lund, Sweden

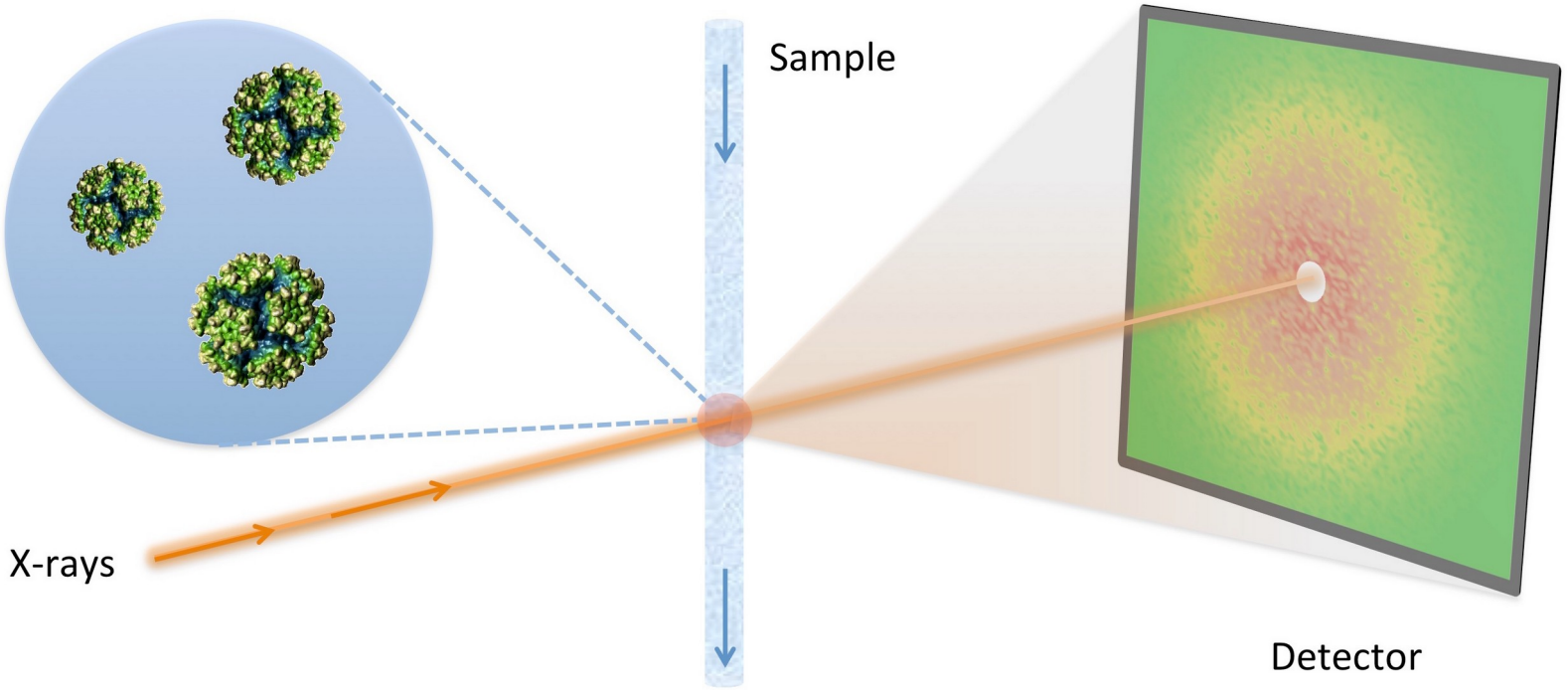
(3) Prevas Test & Measurement, Lund

# Outline

- What is BioSAXS?
- GUI Motivation
- Demo
- Control system integration
- How we use Taurus

# BioSAXS

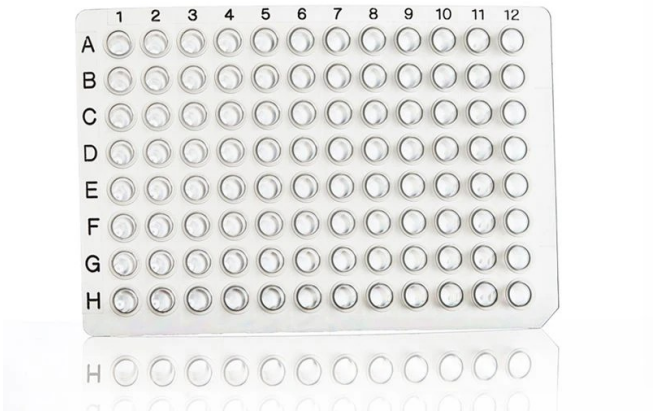
- Small-angle X-ray scattering (SAXS) from Biological samples




https://en.wikipedia.org/wiki/Fluctuation_X-ray_scattering

# BioSAXS

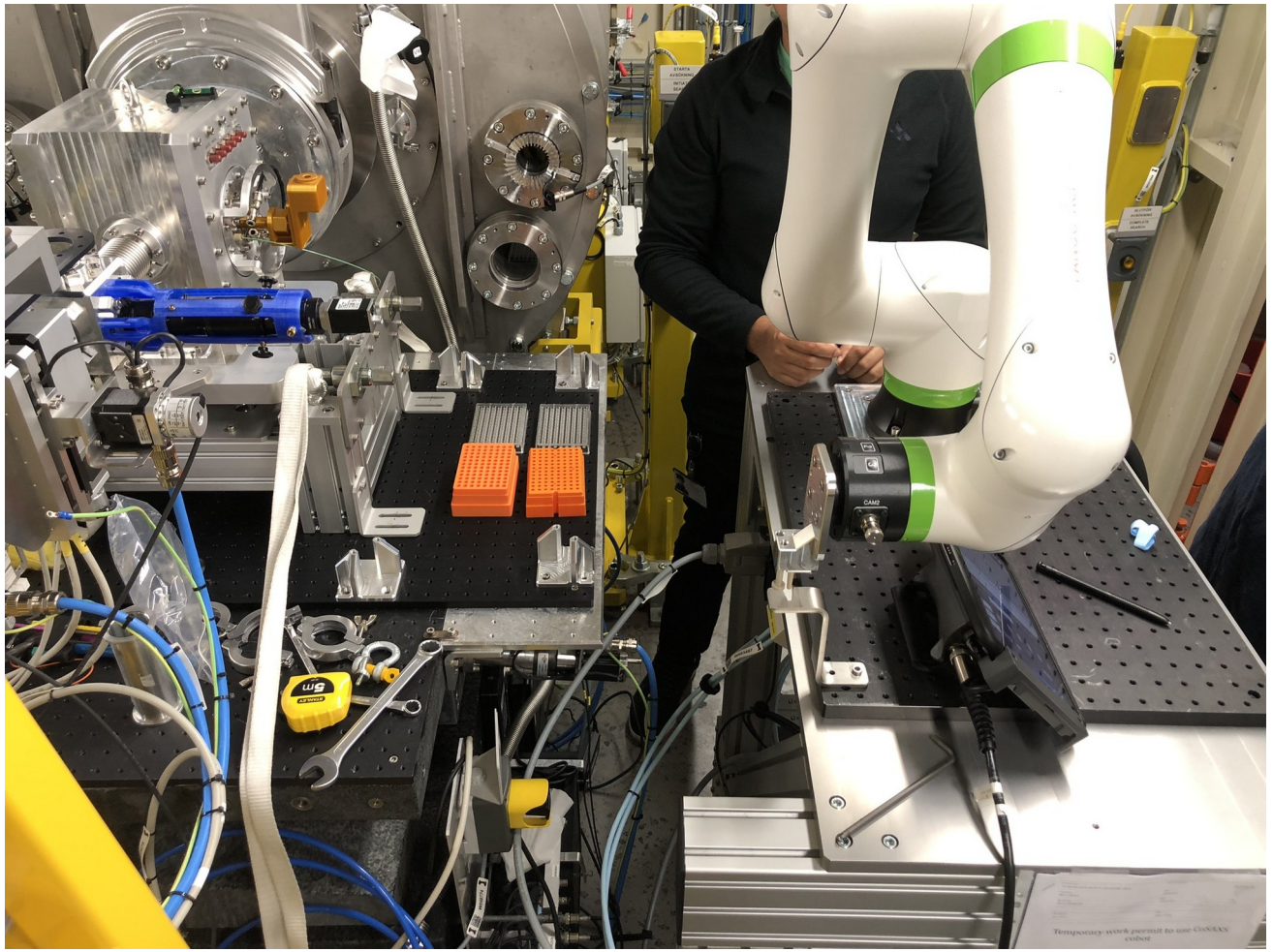
- ~20 seconds per measurement
- Dozens to hundreds of samples
- **Automation**




https://www.azenta.com/products/96-well-non-skirted-pcr-plate

https://crx.fanuc.eu/bg/fanuc-cobot-crx-5ia/

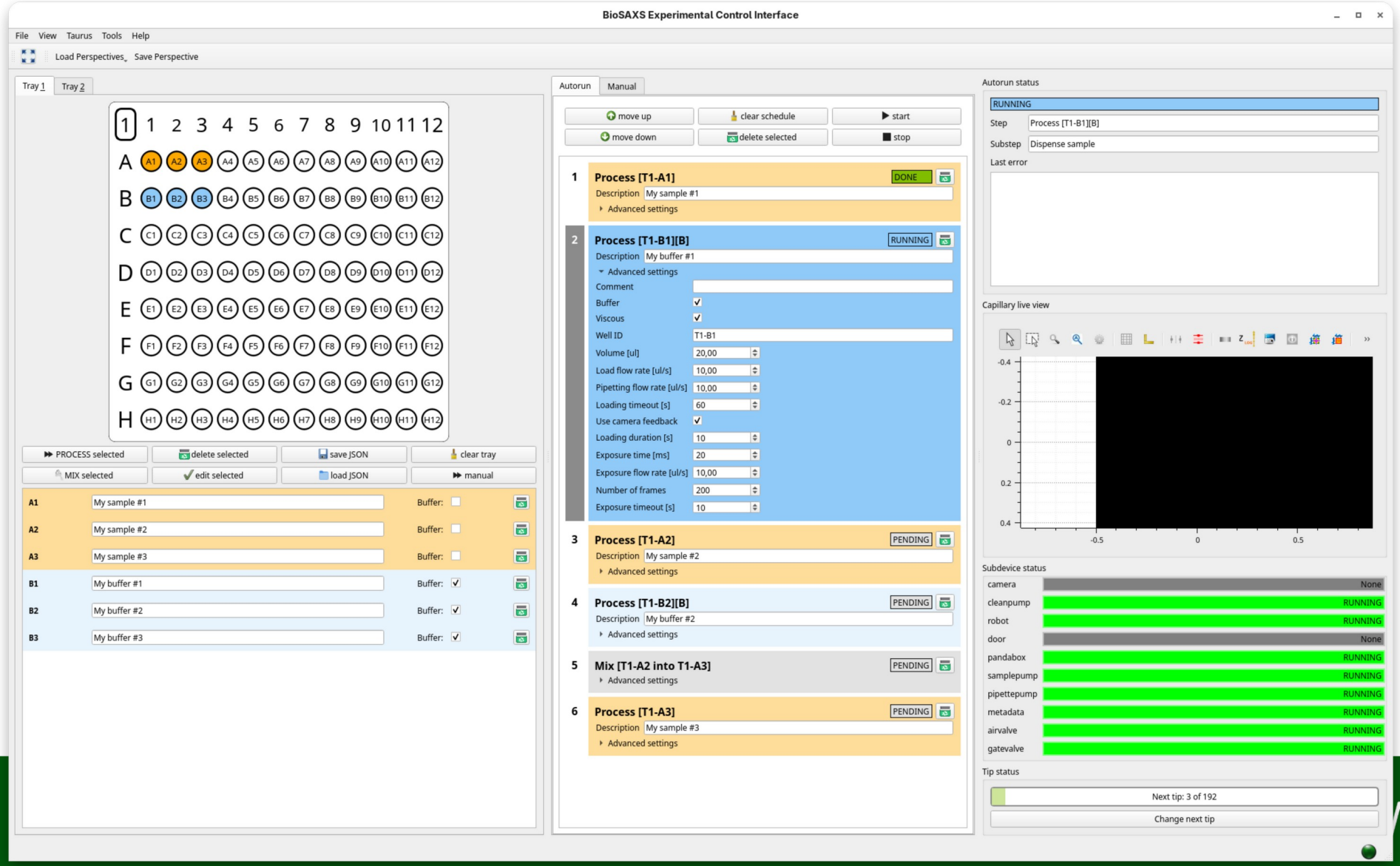
# BioSAXS @ CoSAXS



# BioSAXS GUI – basic requirements

- **Sample management**
  - Create a pool of samples available for measurements
- **Scheduling**
  - Compile an experimental schedule that will be run automatically (“Autorun”)
- **Configuration**
  - Set parameters such as sample volume, exposure time, …
- **Feedback**
  - Status updates from an ongoing experiment

# GUI demo

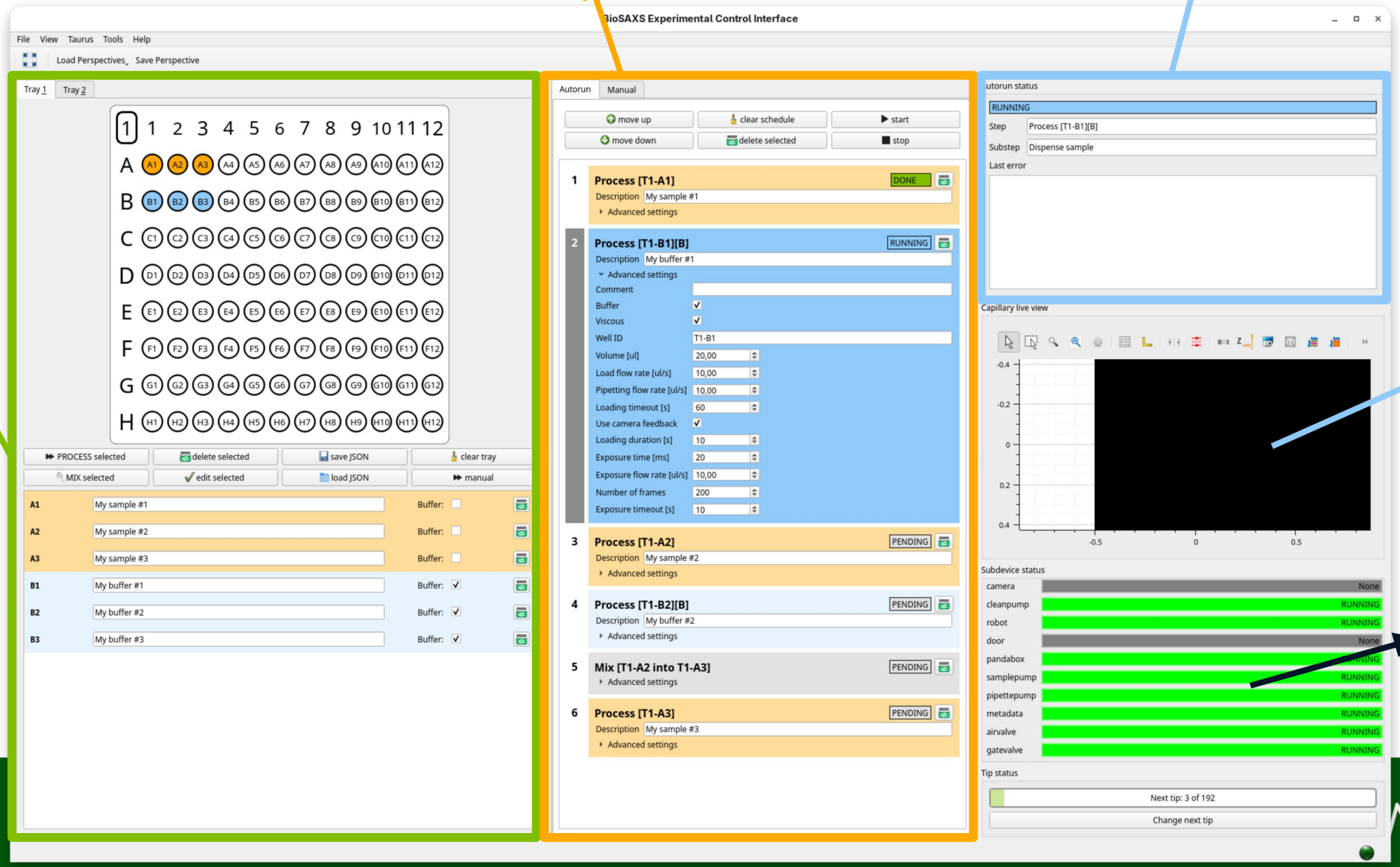

BioSAXS Experimental Control Interface
File View Taurus Tools Help
Load Perspectives Save Perspective
Tray 1 Tray 2
PROCESS selected
delete selected
save JSON
clear tray
MIX selected
edit selected
load JSON
manual
A1 My sample #1 Buffer:
A2 My sample #2 Buffer:
A3 My sample #3 Buffer:
B1 My buffer #1 Buffer:
B2 My buffer #2 Buffer:
B3 My buffer #3 Buffer:
Autorun Manual
move up
clear schedule
start
move down
delete selected
stop
1 Process [T1-A1] DONE
Description My sample #1
Advanced settings
2 Process [T1-B1][B] RUNNING
Description My buffer #1
Advanced settings
Comment
Buffer
Viscous
Well ID T1-B1
Volume [ul] 20,00
Load flow rate [ul/s] 10,00
Pipetting flow rate [ul/s] 10,00
Loading timeout [s] 60
Use camera feedback
Loading duration [s] 10
Exposure time [ms] 20
Exposure flow rate [ul/s] 10,00
Number of frames 200
Exposure timeout [s] 10
3 Process [T1-A2] PENDING
Description My sample #2
Advanced settings
4 Process [T1-B2][B] PENDING
Description My buffer #2
Advanced settings
5 Mix [T1-A2 into T1-A3] PENDING
Advanced settings
6 Process [T1-A3] PENDING
Description My sample #3
Advanced settings
Autorun status
RUNNING
Step Process [T1-B1][B]
Substep Dispense sample
Last error
Capillary live view
Subdevice status
camera None
cleanpump RUNNING
robot RUNNING
door None
pandabox RUNNING
samplepump RUNNING
pipettepump RUNNING
metadata RUNNING
airvalve RUNNING
gatevalve RUNNING
Tip status
Next tip: 3 of 192
Change next tip

# GUI demo

"Autorun schedule" – a list of items to be processed. An item could be either processing a sample (loading, measuring, cleaning), or mixing the contents of two sample wells.

Status widget, provides information about the ongoing experiment

Tray editor – allows the user to specify the wells in which sample has been loaded and to build a pool of samples. For each sample, the user may enter a description and set the 'buffer' flag.




Sampel live view

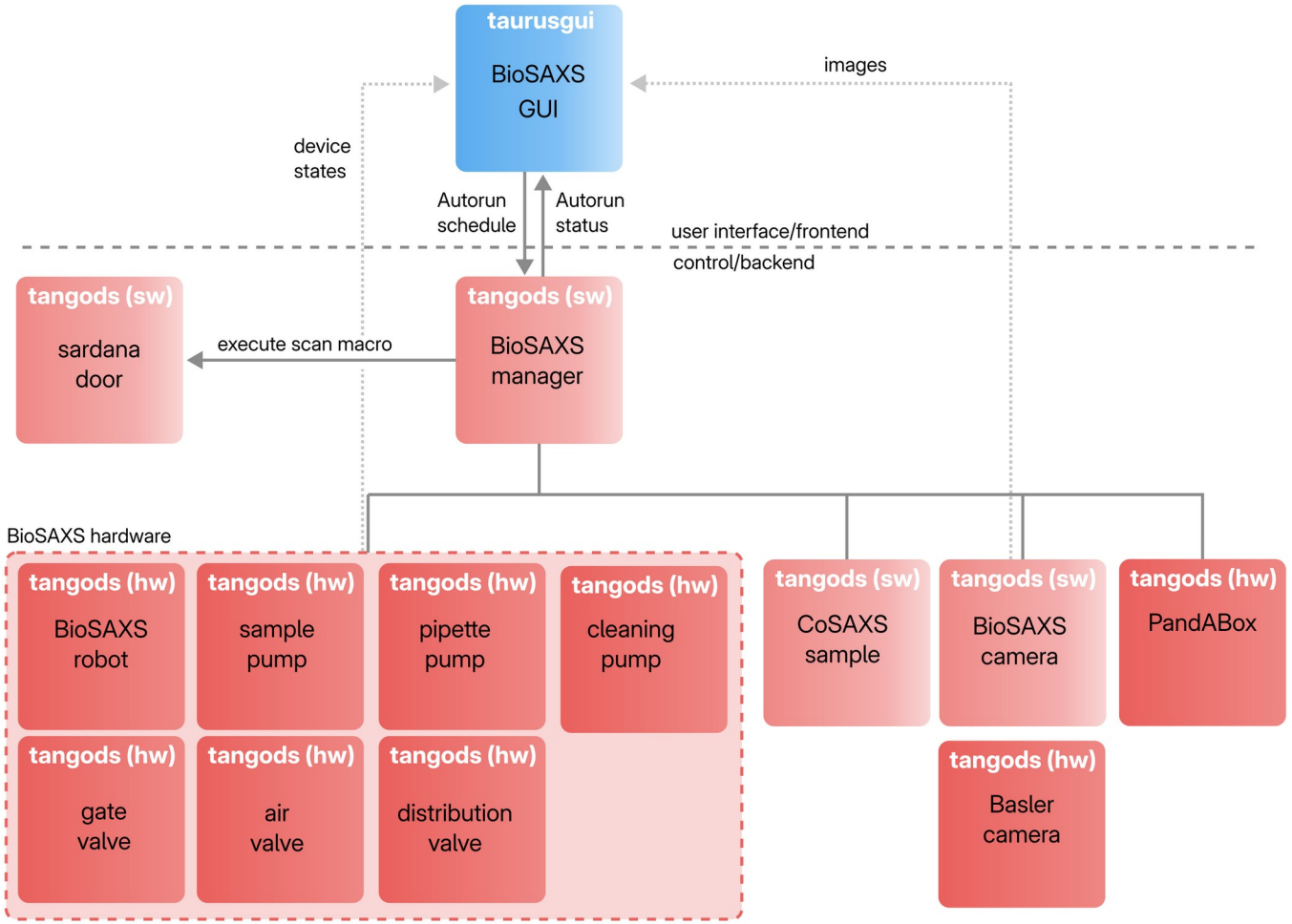
States of sub-devices (valves, pumps, robot, ...)

# Control system integration

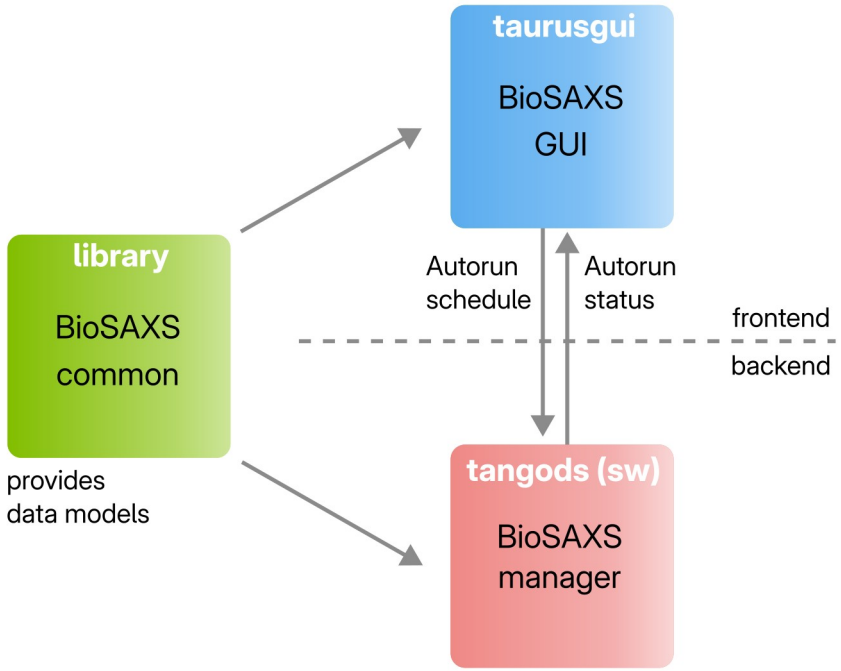
**User interface**

**Orchestration (sardana/Tango)**

**Tango devices (components)**



# GUI and Manager Device

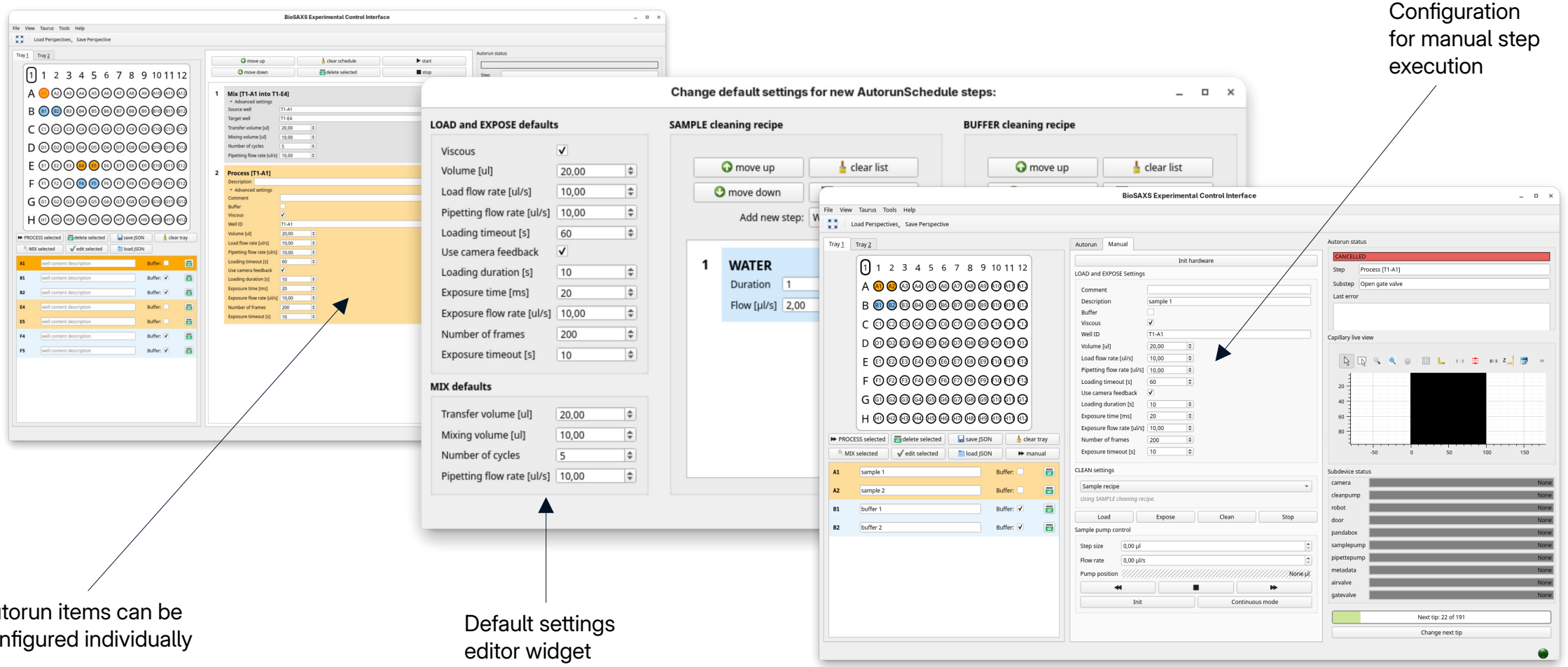



- GUI-manager communication primarily via
  - `AutorunSchedule` – the list of items to process and their configuration
  - `AutorunStatus` – information about current step/item, sub-step and exception

```python
from pydantic import BaseModel

class AutorunStatus(BaseModel):
    """
    Provides information about the current status of an autorun (ongoing or finished).
    Provides the data structure for the `AutorunStatus` attribute of the
    BioSAXS manager tango device.
    """
    status: Status | None = None
    step: AutorunStep | None = None
    substep: SubStepInfo | None = None
    error: AutorunExcInfo | None = None
```

- Definition of configuration parameters in shared lib
  - Configuration: used in manager device to configure hardware
  - GUI: Various widgets for editing configuration paramters

# Configuration Parameters




Configuration for manual step execution

Autorun items can be configured individually

Default settings editor widget

# Configuration Parameters

common library

Configuration for manual step execution

GUI

```python
from pydantic import BaseModel, Field


class MixSettings(BaseModel):

    well_id_from: str = ""
    well_id_to: str = ""
    transfer_volume: float = 20.0  # in µl
    mixing_volume: float = 10.0  # in µl
    num_cycles: int = 5  # how often to aspirate and dispense
    pipetting_flow: float = 10.0  # in µl/s
```

- Initial approach: settings models in common library define parameter **name**, **type** and **default value**
- Parameter **meta-information** (min, max, step size, read only, …) in a separate list on the GUI side
- Disadvantages:
  - Spreads information about config parameters across two repositories
  - Creates hidden dependency between lib and GUI

```python
class MixSamplesItem(AutorunBaseItem):
    """
    Widget for representing a 'MIX samples' step in the autorun schedule.
    """
    configParams: list[ConfigParameter] = [
        ConfigParameter(
            name="well_id_from",
            category="mix",
            label="Source well",
            dataType=str,
            defaultEditable=False,
            expert=True,
            readOnly=True,
        ),
        ConfigParameter(
            name="well_id_to",
            category="mix",
            label="Target well",
            dataType=str,
            defaultEditable=False,
            expert=True,
            readOnly=True,
        ),
        ...
    ]
```

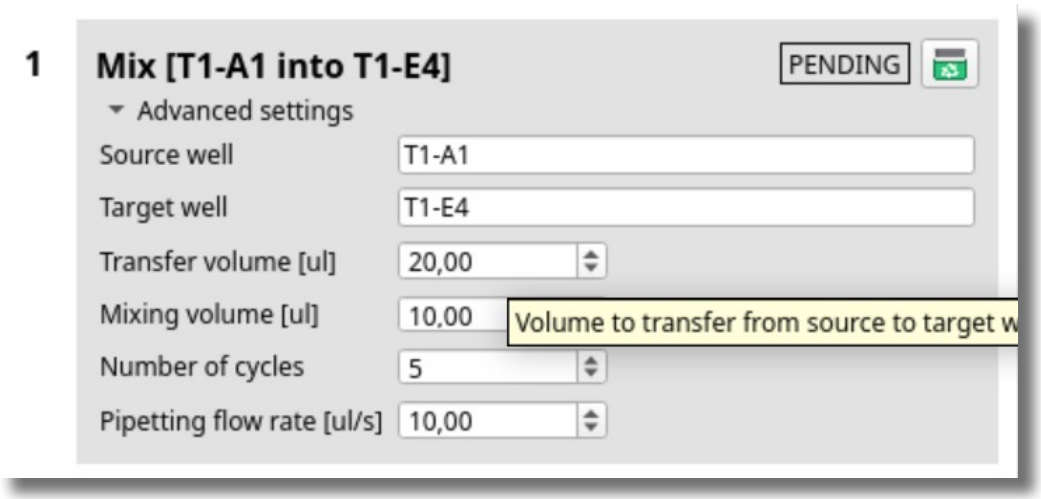
# Configuration Parameters

common library

```python
from pydantic import BaseModel, Field


class MixSettings(BaseModel):

    ...

    transfer_volume: float = Field(
        default=20.0,
        ge=0.0,
        le=200.0,
        title="Transfer volume",
        description="Volume to transfer from source to target well.",
        json_schema_extra={
            "category": "mix",
            "unit": "ul",
            "read_only": False,
            "default_editable": True,
            "step_size": 0.1,
            "expert": True,
        },
    )

    ...
```

GUI

```python
class MixSamplesItem(AutorunBaseItem):
    """
    Widget for representing a 'MIX samples' step in the autorun schedule.
    """
    configParams: list[ConfigParameter] = paramsFromModel(MixSettings)
```

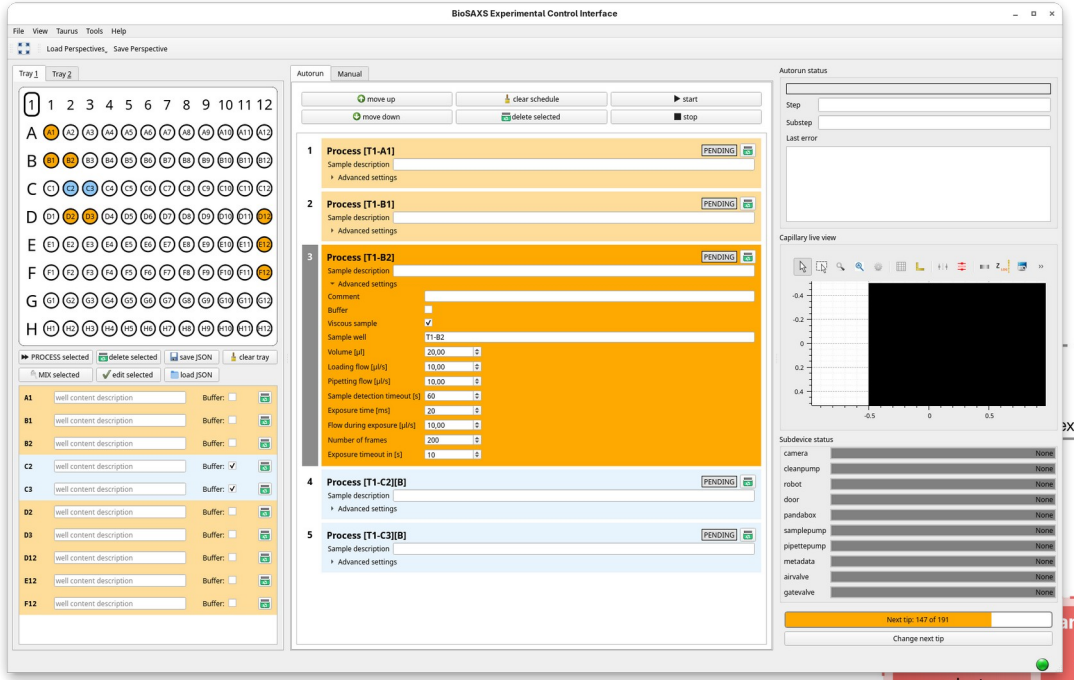


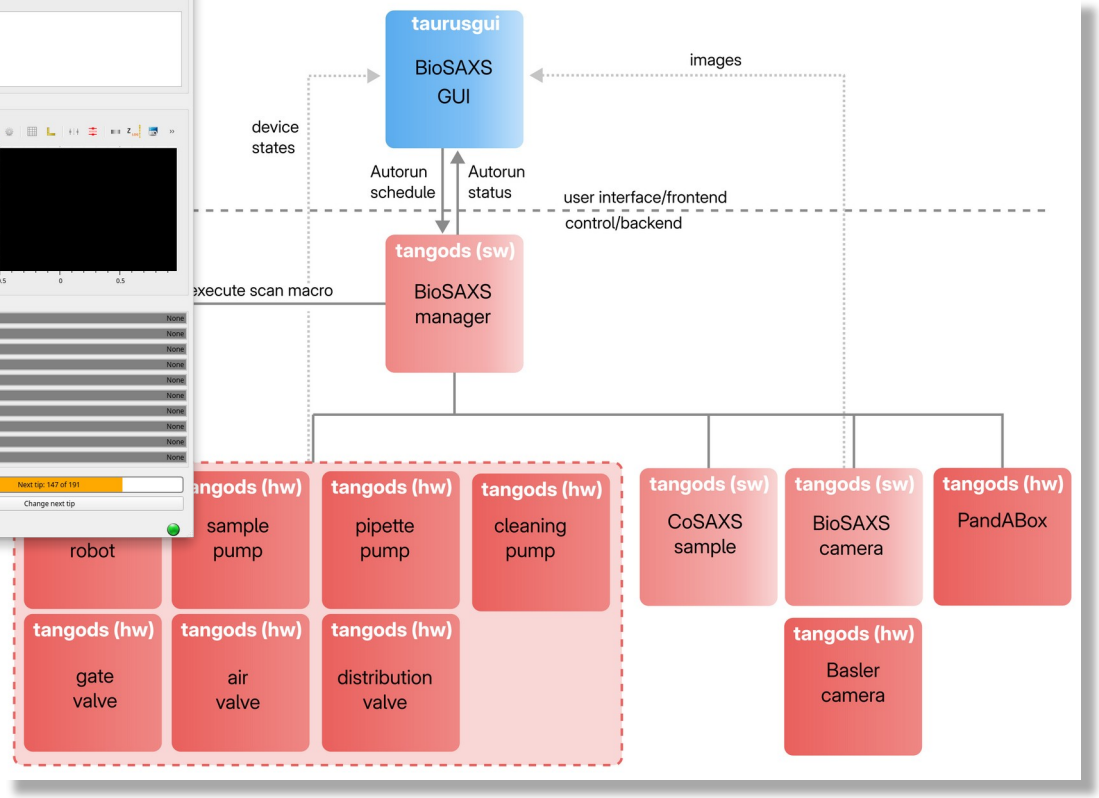

- Current approach: settings models in common library extended to contain parameter meta information
- GUI: widgets that expose configuration parameters to user are build from the model
- Single source of truth for configuration parameters
- Easy to add configuration parameters

# Taurus

- `TaurusApplication` and `TaurusMainWindow` as base
- Display values from Tango control system
  - Live images of capillary
  - Device states
- Handle change events in custom widgets
  - Status updates during “Autorun”
  - Dynamic GUI configuration

# Thank you for you attention!




MAXIV