HZB BESSY II Light Source

THE PYTHON ACCELERATOR MIDDLE LAYER PROJECT

Teresia Olsson,
on behalf of the pyAML collaboration

ROCK-IT Bluesky Workshop, 5-7 November 2025

pyAML

PURPOSE

- For commissioning and operation of particle accelerators robust and reliable software tools are required.
- Many of these tools are developed and used by accelerator physicists.
- Some procedures are the same (or at least similar) at different facilities → possible to standardize and share applications.
- When designing/upgrading accelerators need for simulating the commissioning process → ideally use same scripts as will be used during real commissioning.

BACKGROUND: MATLAB MIDDLE LAYER

- Many synchrotron light sources today use Matlab Middle Layer (MML).
- Originally developed at ALS and SLAC in the 1990s and then spread over the world.
- Many useful features:
  - Control system agnostic.
  - Abstraction of hardware devices, independence of naming conventions, easy to group similar devices together.
  - Can switch between real machine and simulator.
  - Allows for both quick tests of new measurements and running more complex applications for standard procedures.
- But…
  - Not using modern software practices.
  - Matlab propriety software requiring a license & decreasingly known in the community.
  - Collaborative development faded out → diverged into lab specific versions.
  - Do not follow FAIR principles for data management.
  - Not easy to integrate with important “new” tools such as machine learning packages.
  - Not integrated with simulated commissioning.

BACKGROUND: MATLAB MIDDLE LAYER

- Many synchrotron light sources today use Matlab Middle Layer (MML).
- Originally developed at ALS and SLAC in the 1990s and then spread over the world.
- Many useful features:
  - Control system agnostic.
  - Abstraction of hardware devices, independence of naming conventions, easy to gr
  - Can switch between real machine and simulator.
  - Allows for both quick tests of new measurements and running more complex a
- But…
  - Not using modern software practices.
  - Matlab propriety software requiring a license & decreasingly known in the comm
  - Collaborative development faded out → diverged into lab specific versions.
  - Do not follow FAIR principles for data management.
  - Not easy to integrate with important “new” tools such as machine learning packages.
  - Not integrated with simulated commissioning.

Development of new tools ongoing at many facilities!

VISION

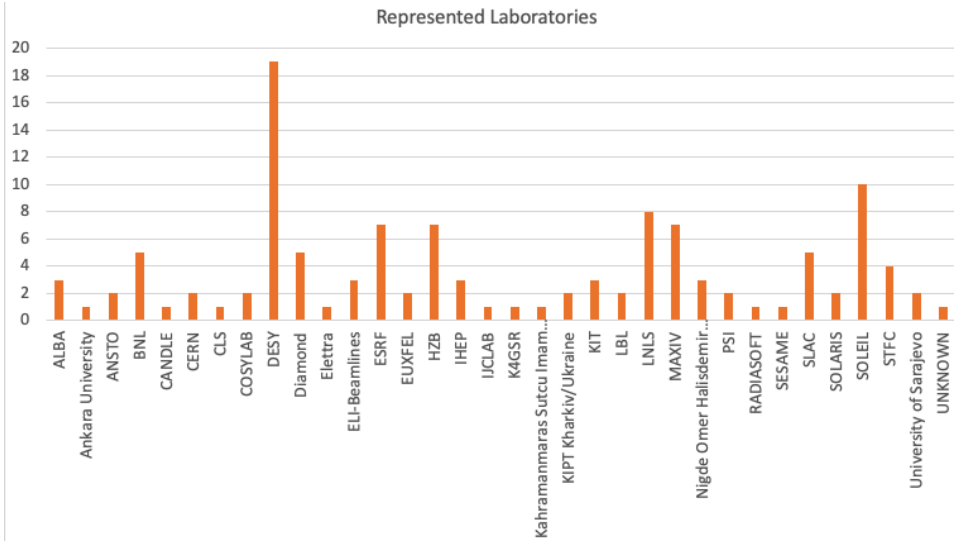
- Join resources together to develop new Python-based tool.
- Developed and maintained in a collaborative way by an open source community consisting of accelerator facilities around the world.
- Including different type of facilities → not just synchrotron light sources.
- Cover the different steps in the life cycle of an accelerator.

→ Joint technology platform for design, commissioning and operation of particle accelerators

PRESENT STATUS

- Status October 2025:
  - Community: 120 people, 32 organisations, 21 countries
  - Maintainers: 20
  - Steering committee (DESY, ESRF, HZB, MAX IV, SOLEIL)
- Big community but little resources → no one works full time on project.

Status email list October 2025



- Currently in the prototype phase.
- Prototype should be finished by the end of the year and evaluated at workshop in February.

- More details: https://python-accelerator-middle-layer.github.io/

MOST IMPORTANT QUALITY ATTRIBUTES

Classification according to Bass, Clements & Kazman, “Software Architecture in Practice, 4th Edition, Pearson Education, 2022.

- Usability
  - Accelerator physicists must be able to use, develop, debug and maintain.
  - Usable for wide range of users (bachelor students to experienced senior scientists).
  - Support the research process: both easy scripting and running complex, standardized applications.
- Modifiability
  - Easy to add new functionality over time.
  - Needs to evolve together with the evolution of accelerator design & technology.
  - Possible to add lab specific functionality.
- Integrability
  - Possible to integrate with existing hardware/control system tools.
  - Leverage the scientific Python ecosystem.
  - Easy integration with HPC and data management resources (clusters, databases etc).

AGNOSTIC INTERFACE

- Support different control systems: DOOCS, EPICS (CA/PVA) and TANGO.
- Possible to write generic devices and the choice of control system is just configuration.

- Same interface to different backends:
  - Live accelerator
  - Virtual accelerator: control system + model
  - Simulator: model only

→ Develop + test applications using one backend which works without modifications for another backend.

MACHINE INDEPENDENCE

- Independent of lab specific naming conventions → fully configurable.
- Physicists must be able to interact with machine using the names they know → hide control system names from the user.

- Possible to use different types of accelerators (storage rings, ramped machines, transfer lines, linacs etc.)
- Default simulator will be pyAT (python accelerator toolbox) but not best choice for all types of accelerators → in the future be possible to extend to other simulators.

→ Configuration layer crucial for use at different facilities.

CONVERSIONS

Classification according to Y. Hidaka, “Particle Accelerator MIddle LAyer (PAMILA)”, presentation at pyAML community meeting 15 November 2024.

- Two type of conversions:
  - Universal (intra-dimensional) conversion:
    - Example: mA ↔ A, mm ↔ m
    - Handle control systems configured for different units.
  - Representation (inter-dimensional) conversion (often called hardware to physics conversions):
    - Example: A ↔ mrad
    - Handle conversions between different backends but also preference of the user → physicists must be able to use the units they know their machine in.

- Needs to be able to handle combined function magnets (m x n mapping) → integration with magnet excitation curves or models.

→ Flexible and customisable two-way conversion interface.

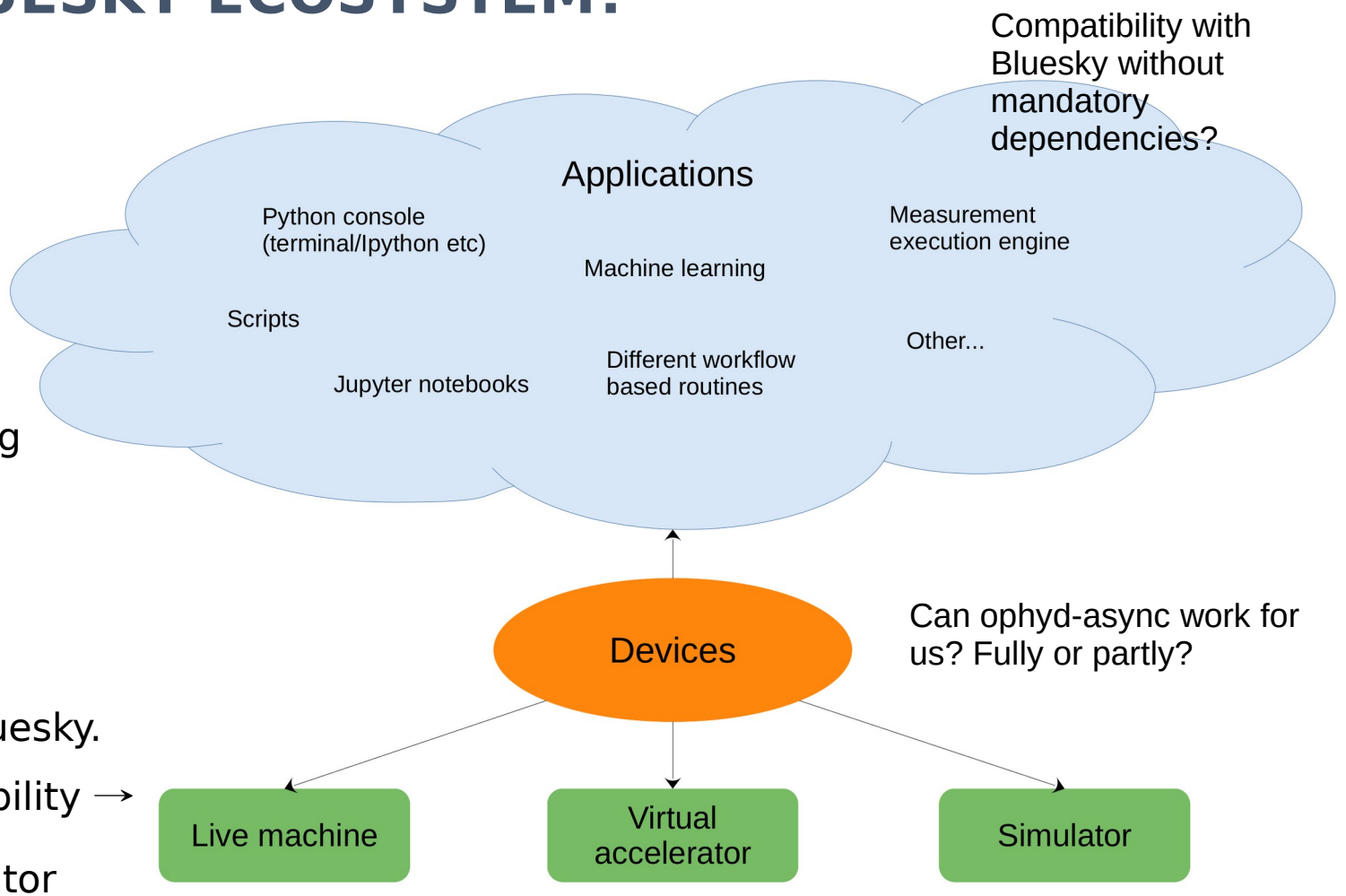
APPLICATIONS

- Possible to write generic applications that work out of the box after configuration.
- Prerequisites:
  - Grouping of devices to use together: e.g. all horizontal correctors grouped together (often called family).
  - Standardised names: e.g. correctors used for orbit correction mapped to the name OrbitCorrectors.
  - Execution should be controlled at device level → no sleeps in applications → when hardware upgraded instant benefit.
- Modular: measurement and analysis independent.
- Metadata automatically generated from devices and configuration.

→ Sufficient abstraction required to write generic applications that can be shared between facilities.

CONNECTION TO THE BLUESKY ECOSYSTEM?

- Many similar requirements.
- Hardware abstraction:
  - Our users want the option to use devices directly → not something like runEngine for all applications.
  - Currently exploring and evaluating ophyd-async for our use cases.
- Application layer:
  - Need flexible options for how to use devices.
  - Some labs do **not** want to use Bluesky.
  - Other labs want Bluesky compatibility → for standard applications or measurements involving accelerator + insertion device + beamline.



THANK YOU!

To contact the pyAML steering committee or subscribe to our email list:
pyaml-steering-committee-contact@esrf.fr.