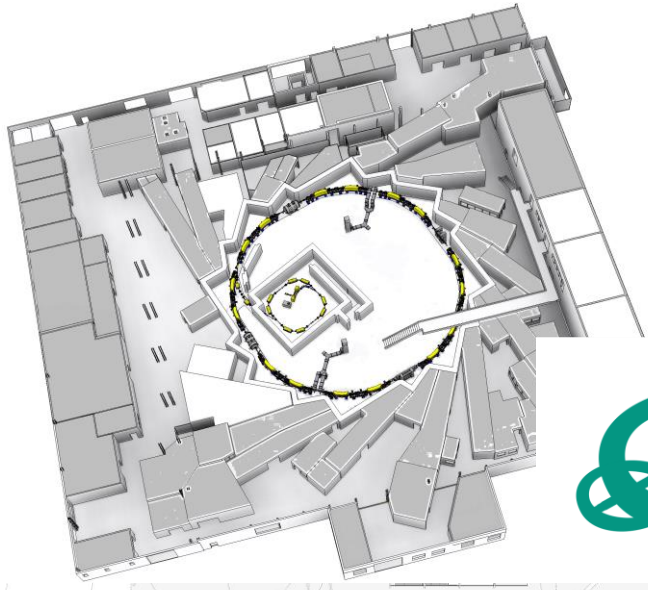
Making controls groups' systems more accessible to scientist with simple Python interfaces and cleaner code

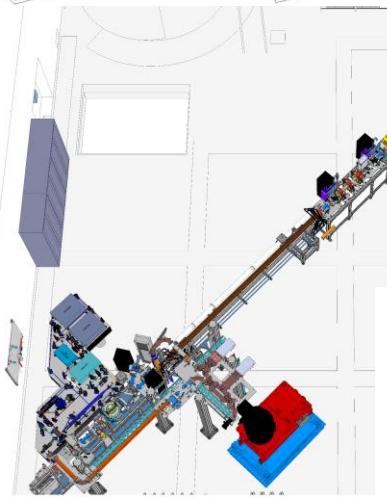
Julian Gethmann, 10th MT ARD ST3 Meeting at HZB

Multiple accelerators

- KARA
- FLUTE
- Furthers to come

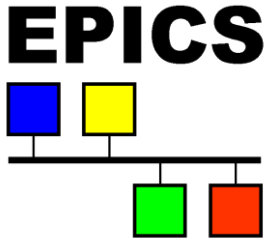
- Similar infrastructure, *but*
  - Different features
  - Different settings
  - Different “namespaces”

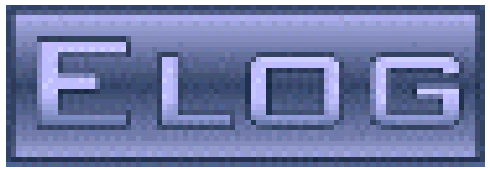


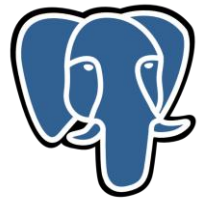


Software infrastructure to control the machines

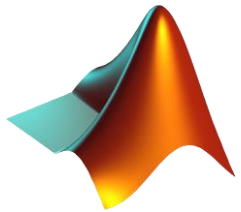
- Diverse software infrastructure for machine controls
  - General data gathering: EPICS (ANL), DOOCS (DESY)
  - Data archiving: cassandra
  - Operator‘s notes: Elog (PSI)
  - Save and restore of machine settings: PostgreSQL
  - Machine optimisation: MML, etc.







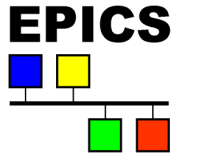
PostgreSQL

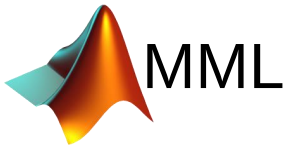


MML

. . .

Software infrastructure to control the machine





- *Doocs via EPICS*, pydoocs
- caget, camonitor, pyepics, *caproto*
- NoSQL, JSON
- SQL
- HTTP individual text format, *pyelog*
- Matlab/HDF5, *scipy.io*
- scikit-learn
- Further to come

Software infrastructure to control the machine

- *Plenty* systems of
- Different *kind* and
- For various *use cases*

Too much to care of for a typical (trainee) scientist doing his own experiment

EPICS

DOOCS

cassandra

ELOG

PostgreSQL

MML

python™

. . .

Why should I care?

Different user groups: Different demands.

User group	Demands on the system
Students and Interns	→Works with little domain knowledge
External Researcher	→Fast setup, limited permissions
PhD researcher	→Access to all the systems
Senior scientists	→Good interface
Machine operators	→Maintainability
Controls group and IT	→Adaption/Usage

Different demands: Different challenges.

Demands on the system

- Works with little domain knowledge
- Fast setup, limited permissions
- Access to all the systems
- Good interface
- Maintainability
- Adaption/Usage

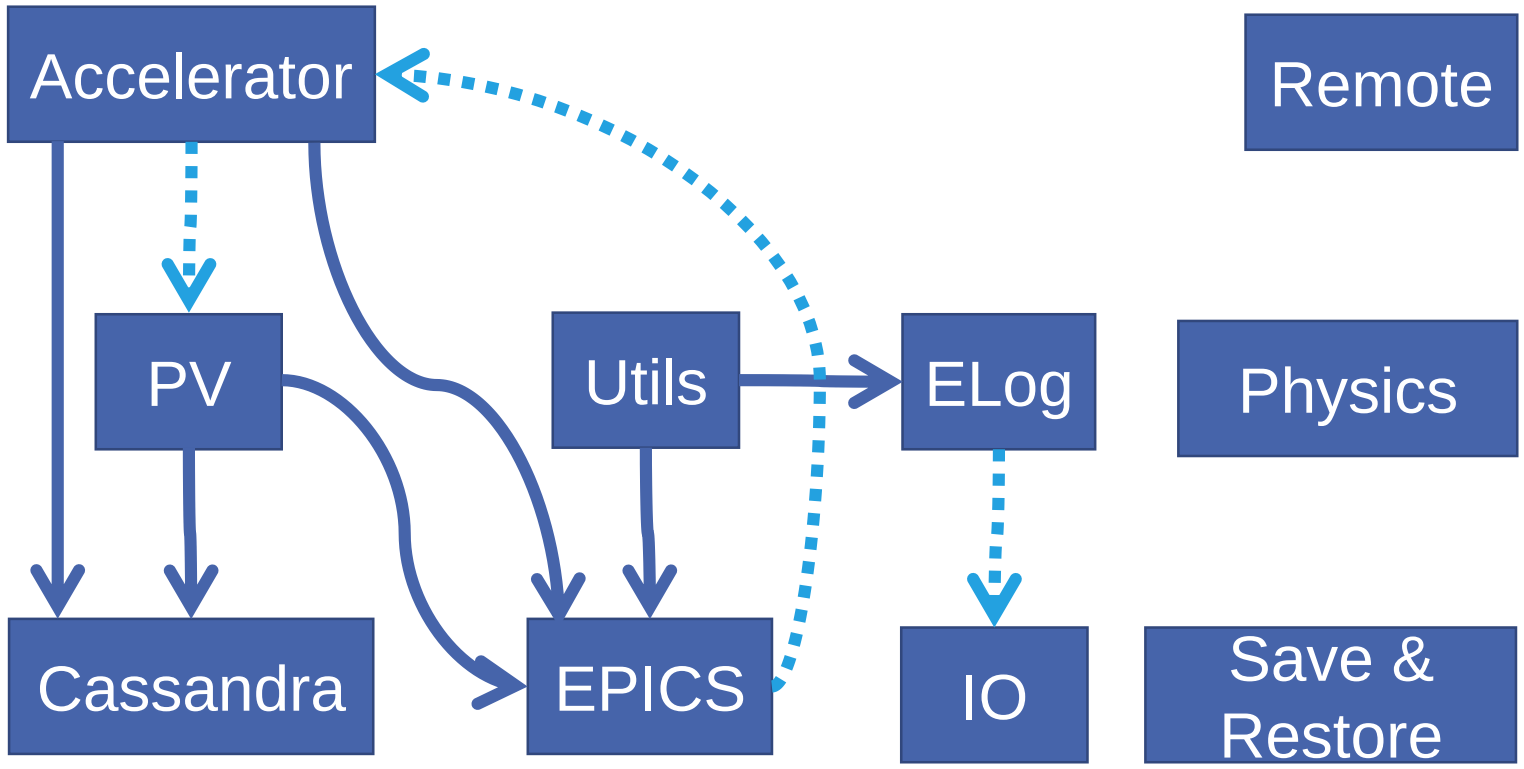
Challenges

- Unmaintainable code/experiments
- Systems get reinvented often
- Errors from unintended use arise
- Too complicated systems in “silos”

Python wrapper as a solution

Package dependencies

- Data aggregation wrapper
  - Cassandra
  - ELog
  - EPICS
  - Save and Restore
  - IO
- Helper libraries
  - Accelerator
  - PVs
  - Utils
  - Physics
  - Remote



Data aggregation wrapper

Wrapper with **sane defaults** for our user's needs as part of **this collection**

So far we have wrappers for

- Archive (Cassandra)
- Operators log (ELog)
- Live data (EPICS)
- Settings archive (Save and Restore)
- Typical machine measurements data formats

- Open for new measurements / instrumentation / data sources

Helper libraries

- Accelerator
- PVs
- Utils

- Physics

```python
from ibpt import accelerator
from ibpt.accelerator import kara
from ibpt.epics import get_pv
from ibpt.pvs import get_pv_string
from ibpt.utils.network import is_internal_network

if is_internal_network():
    energy = epics.get_pv("energy")
    print(f"{accelerator.get} runs with {energy} GeV")
    # -> KARA runs with 2.5 GeV
    energy_pv = get_pv_string("energy")
    print(f"You can cross check it with 'caget {energy_pv}'")
    # -> You can cross check it with 'caget A:SR:BeamInfo:01:Energy'
```

Helper libraries

- Accelerator
- PVs
- Utils

- Physics

The code fits on a page

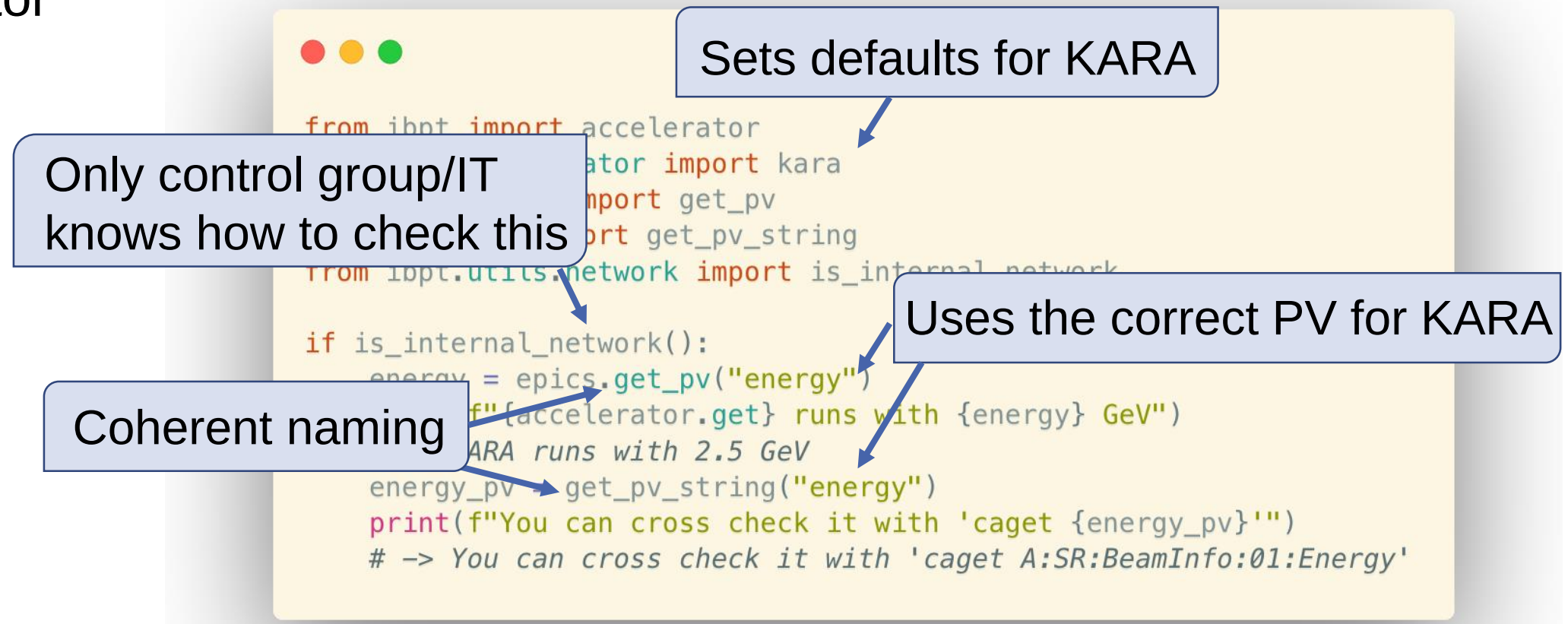
```python
from ibpt import accelerator
from ibpt.accelerator import kara
from ibpt.epics import get_pv
from ibpt.pvs import get_pv_string
                                    nal_network

    print(f"{accelerator.get} runs with {energy} GeV")
    # -> KARA runs with 2.5 GeV
    energy_pv = get_pv_string("energy")
    print(f"You can cross check it with 'caget {energy_pv}'")
    # -> You can cross check it with 'caget A:SR:BeamInfo:01:Energy'
```

Helper libraries

- Accelerator
- PVs
- Utils

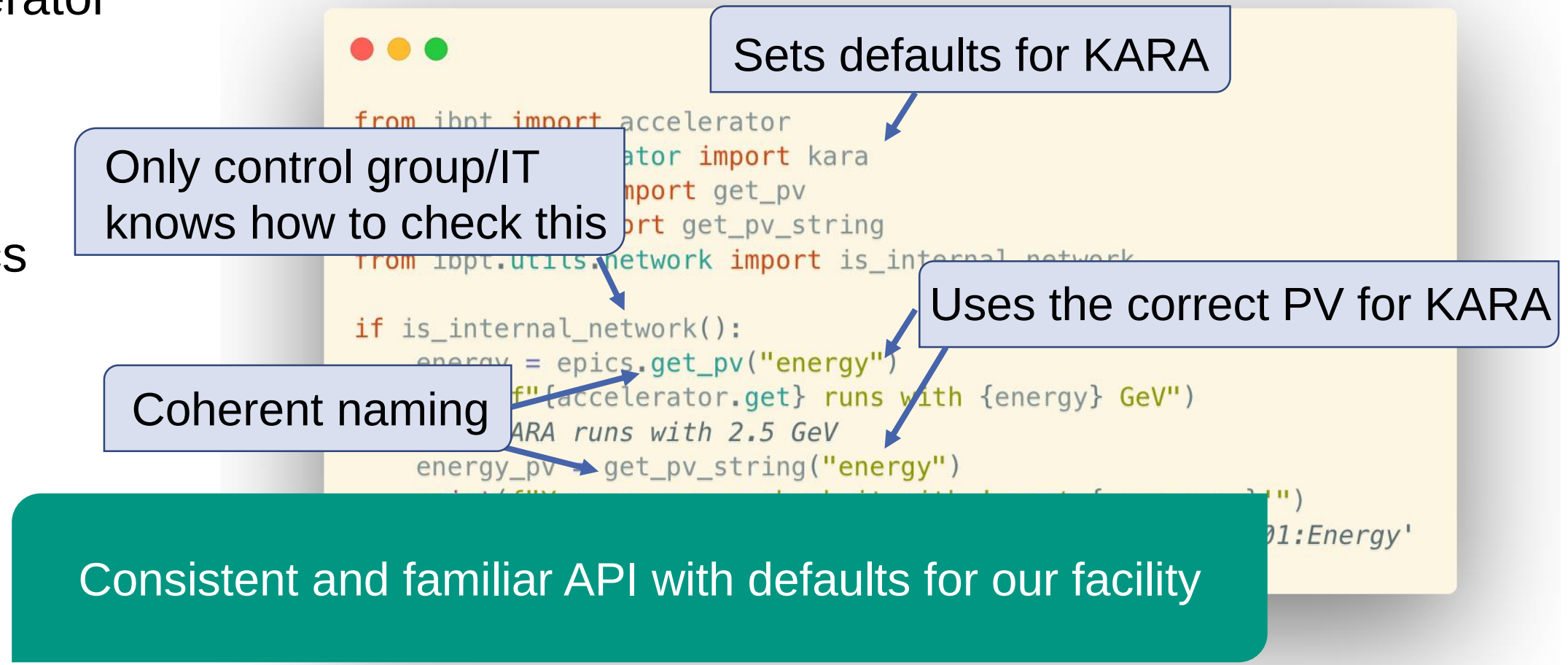
- Physics



Helper libraries

- Accelerator
- PVs
- Utils

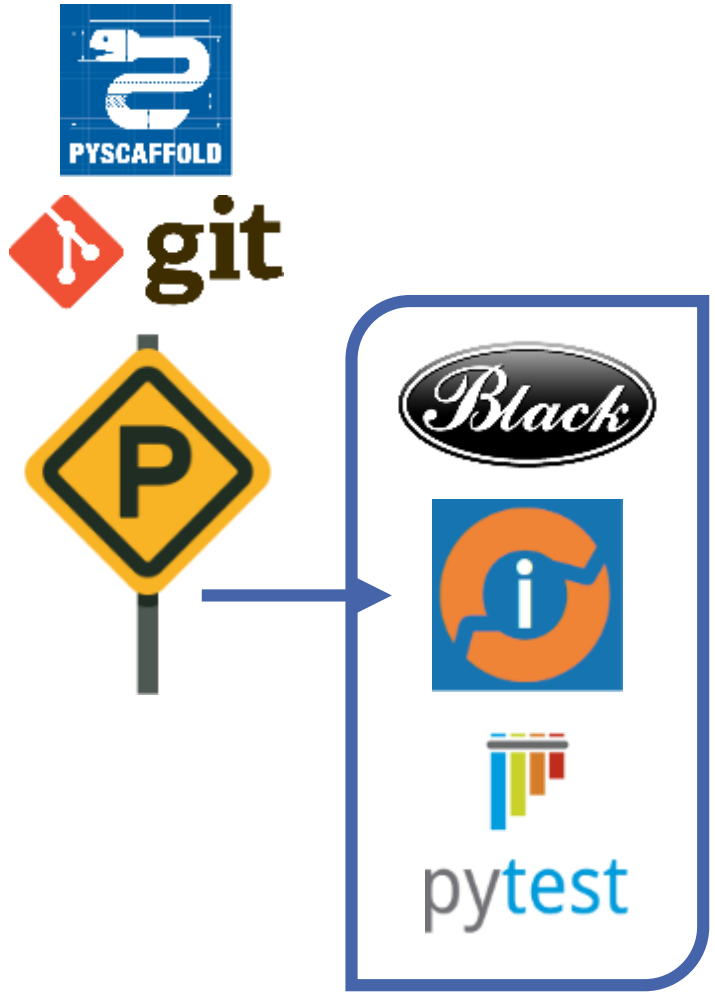
- Physics



Development workflows

Development workflow: client side

- (Create Python project)
- Use **git**
- Activate pre-commit
  - Style formatting
  - Import sorting
  - PyDocStyle
- Write tests
- Create merge request



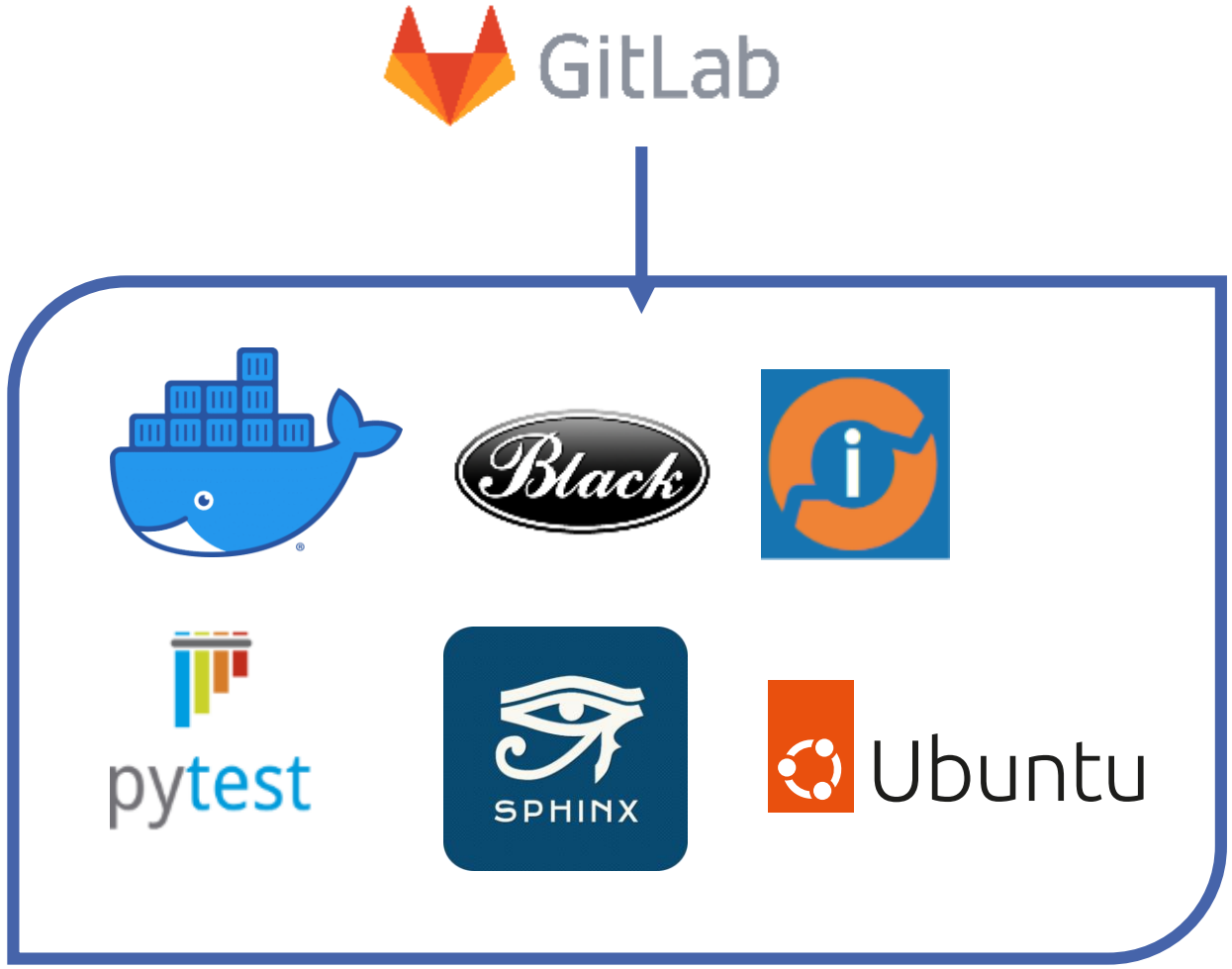
($ putup --ibpt picoharp)
$ git clone git@git.ibpt…
$ cd repo
$ pre-commit install

… write code and tests …

$ git push

Development workflow: server side

- (Create new GitLab project)
- Create merge requests
  - Uses Docker images
  - Checks code formatting
  - Runs tests
  - Generates Documentation
  - Builds Python packages
  - Builds Debian packages
- Code review: learn and educate



Consistence and convenient

- Local side and remote side act similarly: *no surprises*
- Other people‘s code looks familiar: *I can contribute*
- Few steps to setup environment/start: *I do contribute*
- One place to search: *less code duplication*
- Maintainable code: *less headache/more code in use*

Challenges

- Unmaintainable code/experiments
- Systems get reinvented often
- Errors from unintended use arise
- Too complicated systems in “silos”

Approaches

- Code style enforced gently
- Common functionality accessible
- Coherent interfaces exist
- IT expert’s knowledge accessible

Thanks

Thank you for your attention!

Thanks to my colleagues for closing ~300 issues.

- Create your own abstraction layer for sane defaults and settings
- Automate your “not so clean code” to maintainable code pipeline
- Teach best practices with code review and tooling

Resources

- Logos of the respective software projects belong to them and are mostly (registered) trademarks, namely:
  - PyScaffold.org
  - Pre-commit.com / A. Sottile and Yelp
  - Git-scm.com / cc-by Jason Long
  - Github.com/psf/black
  - PyDocStyle.org
  - Python.org / PSF
  - Pycqa.github.com/isort
  - Pytest.org
  - Docker Inc.
  - GitLab B.V.
  - Sphinx-doc.org
  - PostgreSQL Community Association of Canada, and used with their permission
  - Cassandra The Apache Software Foundation
  - EPICS A. Johnson / Uchicago Argonne LLC
  - DOOCS. / DESY
  - ELog S. Ritt / PSI
  - MATLAB MathWorks