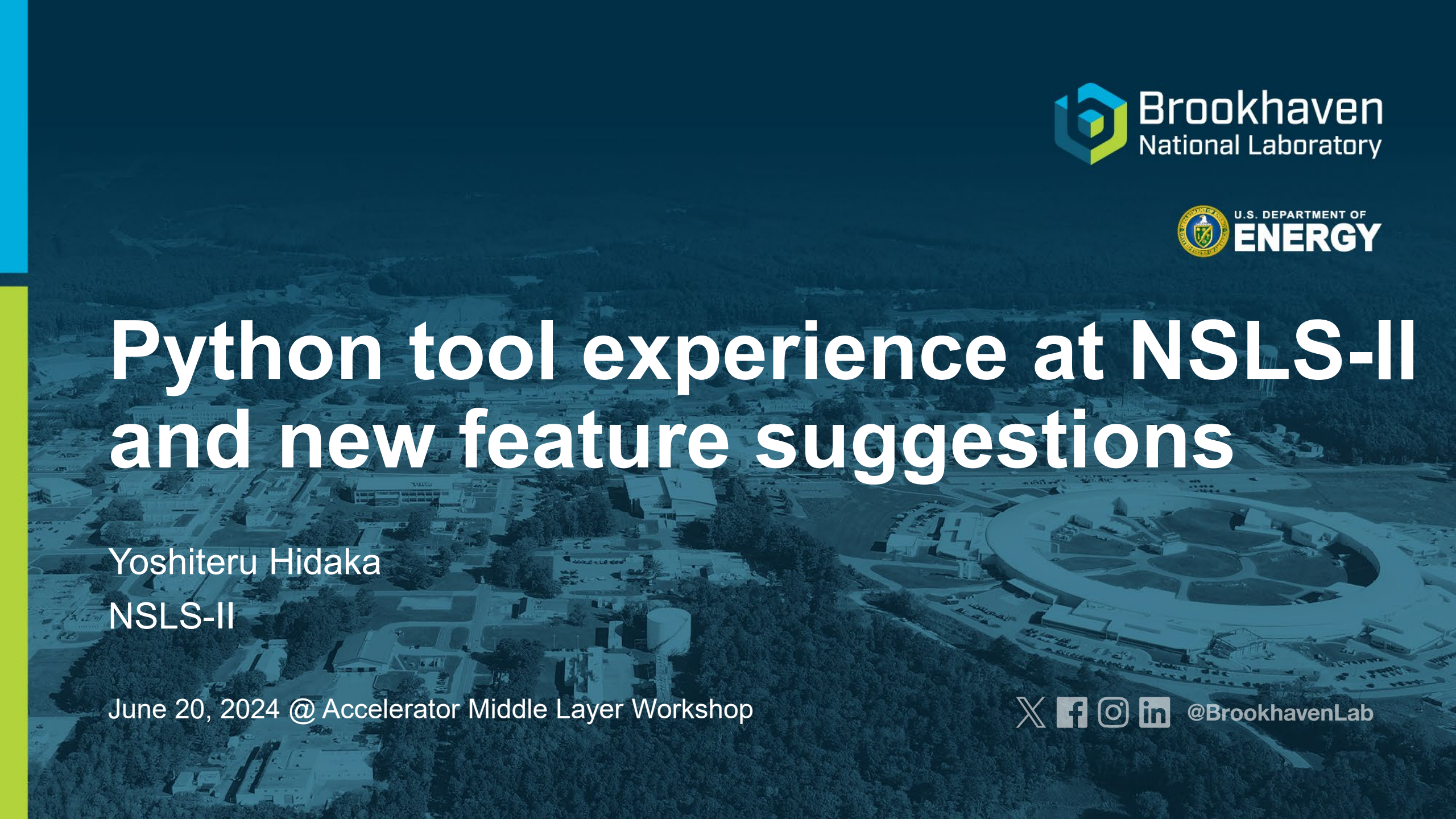

Brookhaven National Laboratory

U.S. DEPARTMENT OF ENERGY

Python tool experience at NSLS-II and new feature suggestions

Yoshiteru Hidaka
NSLS-II

June 20, 2024 @ Accelerator Middle Layer Workshop

@BrookhavenLab

Outline

- Python middle layer at NSLS-II
  - History
  - Example
  - Interfaces / Services
  - Thoughts / Lessons Learned
- New Feature Suggestions

Brief History of Python Middle Layer @ NSLS-II

- `aphla`: Accelerator Physics High-Level Applications
  - Developed for NSLS-II commissioning and beam studies mainly by Lingyun Yang & Guobao Shen, starting in 2010 (also contributions by J. Choi, Y. Hidaka, G. Wang).
    - https://accelconf.web.cern.ch/ipac2012/papers/thppr018.pdf
  - Python selected as main language, though MATLAB Middle Layer (MML) and ELEGANT/SDDS kept for transition, collaboration, and fallback.
  - Simulation engines (AT, ELEGANT, Tracy, etc.) were not integrated / interfaceable.
  - NSLS-II SR commissioning in 2014.
  - Only used at NSLS-II.
  - Active development stopped when L. Yang left in 2016.
  - `pytac` development @ Diamond, influenced by `aphla`, starting in 2017.
- Disclaimer about myself:
  - Opinions expressed in this talk are mine, not NSLS-II as a whole.
  - No formal computer science education.
  - MATLAB main until switching to Python in 2011. Only limited experience with MML.
  - Maintaining NSLS-II `aphla` database since 2016.
    - Plan to switch this year to v2 for easier database update. No plan for continued development.

Example: Dispersion Measurement

```
import aphla as ap
ap.machines.init("nsls2")
ap.machines.use("SR")
bpmobj = ap.getElements('p*c0[3-6]*')
bpmnames = [b.name for b in bpmobj]
f0 = ap.getRfFrequency()
f = np.linspace(f0 - 1e-5, f0 + 1e-5, 5)
for i,f1 in enumerate(f):
    ap.putRfFrequency(f1)
    time.sleep(6)
    obt2 = ap.getOrbit(bpmnames)
    codx[i,:] = obt2[:,0]
    cody[i,:] = obt2[:,1]

ap.putRfFrequency(f0)
dxc = codx - codx0
df = -(f - f0)/f0/eta
p = np.polyfit(df, dxc, 1)
plt.plot(s1, p[0,:], 'o--', label="Fit")
```

ap.machines.load("nsls2", "SR") [Loading from database, construct machine object]

[Search and collect elements of your interest]

L. Yang, et at. ICALEPCS2013, TUPPC130 (2013)

Interfaces

- Jupyter Notebooks, IPython (interactive consoles)
  - Most interactive / easy to modify
  - Notebooks
    - Act as experiment log books
    - Most-often-used code-exchange format between physicists
    - Hard to debug interactively
- Scripts
  - Very interactive & easy to debug, if paired with IDE (e.g., VS Code).
  - Easy to port finished scripts to run on IOC servers as production programs.
- GUI
  - A handful of Python GUIs developed, but no longer used except for machine snapshot/restore & booster ramp manager.
    - Replaced by CSS (Control System Studio, a.k.a. CS-Studio, or Phoebus) panels
    - CSS / MATLAB GUI: development / maintenance much easier (though limited customizability)
    - Python GUI:
      - Too many frameworks – hesitant to invest in one.
      - Full-featured frameworks like PyQt: Steep learning curve => time sink & most users cannot modify / debug.
      - Hard to: debug (non-Python parts, multi-threads, etc.), test (difficult to automate), maintain (keeping up with dependency upgrades)

Services

- MASAR (Machine Snapshot, Archiving, and Retrieve: https://github.com/epics-base/masarService)
  - Used daily for operation and beam studies.
  - Still in Py2/PyQt4 => need upgrade to Py3/PyQt6
    - or switch to Phoebus Save-and-Restore or a Bluesky-based solution?
- Virtual accelerator
  - Only utilized for tests before commissioning
  - Planned to be used as online calculator after commissioning, but not being used (most likely offline)
- Unit conversion (for magnet strengths: A, T, 1/m, etc.)
  - Not being used (offline).
  - Locally converted with `aphla`.
- Lattice / model (Twiss based on power supply setpoint / readback)
  - Not being used (still online, but PVs not updated)
  - Most of the time, pre-compute quantities of interest and save / load from files.
- Python high-level application services still running on IOCs:
  - Dispersion measurement, chromaticity measurement / adjustment, tune measurement / adjustment / feedback, slow orbit feedback, ...

Thoughts on Services

- Model / Simulator on remote server: personally prefer local over remote
  - Local:
    - Full knowledge on what's happening with the model.
    - Under full control (somebody else could be controlling a shared virtual accelerator).
    - Can be offline.
  - Remote:
    - Necessary for heavy calculations (particularly if in need of parallel computations with clusters)
    - Necessary if computed quantities need to be used by many people or other services.
      - On-demand calc. or caching is better; continuous calc. wasteful, if not being used continuously.
    - Need to learn another interface (i.e., extra friction for development).
    - All features / controls may not be made available.
- If developing services (or any investment-heavy programs like GUIs), make sure they will be used by many and/or part of critical infrastructure.
  - Otherwise, they will be abandoned quickly once there is no expert / time.
- "Service-later" approach seems better
  - First develop local features, then later elevate to services if deemed useful / necessary.
  - More resilient against service outage / obsolescence.

Other Thoughts / Lessons Learned

- `aphla` not easy to add/modify machine elements into centralized database (`pytac` seems more intuitive)
  - One reason switching from SQLite to dict/list/numpy in YAML/pickle as database.
  - Will encourage undesirable PV-centric (vs. element-centric) script writing.
- Utilize caching:
  - Faster `aphla` startup by loading machine object from a `pickle` cache file to avoid slow object construction from database files.
- Methods (object-oriented prog.) vs. functions for high-level applications (HLA)
  - A many-step HLA can require lots of variables to be passed btw. each func. -> Fewer inputs/outputs with methods -> Easier to read
  - Method overriding should be useful for customization at a different facility.
- `caget` / `caput`
  - for-loop vs. parallelization – reduce initial connection penalty
  - `caput` (`cothread` package) occasionally not taking effect in interactive consoles until next `caget` – extra `caget` line after every `caput`

New Feature Suggestions

Nice-to-have Features

- `caput` live/offline switch – avoid accidental writing during operation
- Customizable "wait" functions for different programs / facilities
  - Allow different methods: Fixed time (w/ timeout), "ready" PVs, timestamp change, setpoint-readback diff., etc.
  - Easiest to implement functions as class methods with method overriding?
  - Time flow speedup for simulation mode.
- Interactive documentation w/ customized LLM (e.g., ChatGPT)
  - May miss existing functions if right keywords are not used in traditional doc.
  - Customized tutorial based on what you want to do.
- Enforcement of units (dimensions)
- Scenario injections

Units

- Enforce to use “unit” objects for every float value of input/output function arguments
  - NOT about unit conversions between engineering and physics units (e.g., Ampere to/from mrad)
  - NSLS-II: RF frequency (Hz, GHz), BPM (mm, um, nm), ID gaps (mm, um), etc.
  - `aphla`: Can associate units, but info stored as strings
  - Pros for developers:
    - No more manual unit conversion.
    - Avoid logical coding errors.
  - Pros for users:
    - No more wondering (or reading help) about whether to convert BPM data from [mm] to [m], etc. before passing as input arguments.
  - Cons for developers / users:
    - Frustrating before getting used to always associating units.
    - Potentially slow down computations for large arrays.

```
In [71]: ap.getRfFrequency()
Out[71]: 0.49968275679000007

In [72]: ap.getRfFrequency(unitsys=None)
Out[72]: 0.49968275679000007

In [73]: ap.getRfFrequency(unitsys='phy')
Out[73]: 499682756.7900001

In [74]: e = ap.getElements('rfcavity')[0]

In [75]: e.getUnit('f', unitsys=None)
Out[75]: 'GHz'

In [76]: e.getUnit('f', unitsys='phy')
Out[76]: 'Hz'
```

Units (cont'd)

- Potential packages for unit handling:
  - Pint [https://github.com/hgrecco/pint]
    - Used in `aphla-v2`, simple to use, pure Python
  - astropy.units
    - A sub-package of astropy, requires installation of astropy
  - unyt [https://arxiv.org/pdf/1806.02417]
    - Good summary & comparison among these packages

```
In [6]: quads = ap.getElements('QUAD')

In [7]: e = quads[0]

In [8]: e.get('b1', unitsys='phy', handle='readback')
Out[8]: -0.1552004157805968 <Unit('1 / meter')>

In [9]: e.get('b1', unitsys='phy', handle='readback').dimensionality
Out[9]: <UnitsContainer({'[length]': -1})>

In [10]: e.get('b1', unitsys=None, handle='readback')
Out[10]: 34.560410615421546 <Unit('ampere')>
```

Scenario Injections

- Inject artificial machine states by applying a pre-defined sequence of state modifications at checkpoints embedded in high-level applications.
- Very useful for offline testing of complex programs that:
  - are typically operation-oriented
    - as opposed to measurement scripts run by physicists only during beam study shifts: sequential, under full control, not as many exception handling, mistakes tolerable
    - Example: Unified orbit feedback (UOFB) at NSLS-II
  - require handling of
    - on-demand user requests
      - Examples:
        - Enabling / disabling beamline local bump feedback
        - Switching between RF-BPM and X-ray BPM bump feedback
        - Local bump adjustments, etc.
    - machine condition changes (both expected & unexpected)
      - Examples:
        - Sudden beam dump or dropout => Need to abort the whole program.
        - ID gap moves too much while in X-ray BPM bump feedback mode => Need disabling of the bump feedback (not the entire program).

```
while not turn_off_requested:
    iIter = get_iter_index()

    update_online_status()

    scenarios.checkpoint(f"iter_{iIter}")

    try:
        if iIter != 0:
            update_logger()
```

Summary

- NSLS-II has been using in-house developed package `aphla` as our Python middle layer since commissioning in 2014.
- Provided some thoughts and lessons learned based on our 10-year Python experiences for machine control.
- Ready to contribute and migrate to a new standardized Python middle layer in this collaborative effort.
- New potentially useful features have been also suggested / requested as part of the new development.

Thank You!