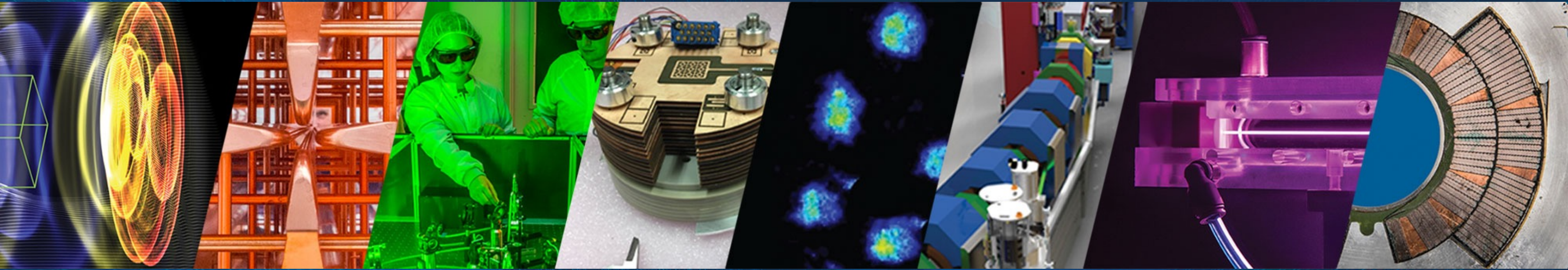
HiPACE++: Ecosystem

Remi Lehe

Accelerator Technology & Applied Physics Division (ATAP)
Lawrence Berkeley National Laboratory (LBNL)



HiPACE++ workshop - July 11th, 2023

BERKELEY LAB

ACCELERATOR TECHNOLOGY & APPLIED PHYSICS DIVISION ATAP

U.S. DEPARTMENT OF ENERGY | Office of Science

HiPACE++ is part of the BLAST ecosystem

BLAST
BEAM PLASMA & ACCELERATOR SIMULATION TOOLKIT

Suite of **open-source** codes
for **plasma & accelerator** simulations,
that are developed **collaboratively**

HiPACE++
Quasi-static PIC, 3D Cartesian

WarpX
Full EM/ES PIC, Cartesian & cylindrical

ImpactX
Electrostatic PIC codes for accelerator physics

use the AMReX framework

(for mesh refinement, GPU portability…)

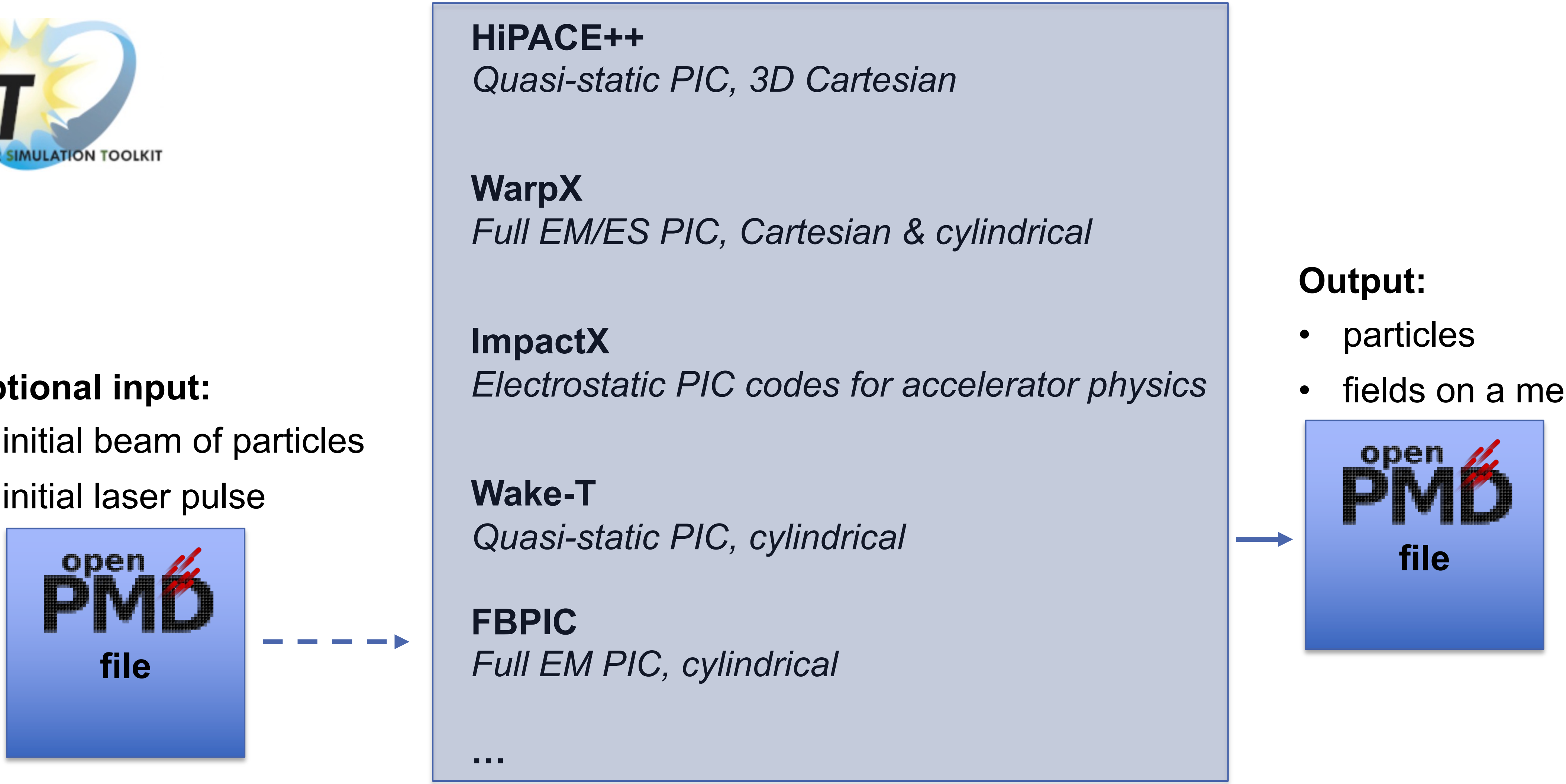
Wake-T
Quasi-static PIC, cylindrical

FBPIC
Full EM PIC, cylindrical

…

BLAST codes use the openPMD format of I/O

BLAST
BEAM PLASMA & ACCELERATOR SIMULATION TOOLKIT



Optional input:

- initial beam of particles
- initial laser pulse

HiPACE++
Quasi-static PIC, 3D Cartesian

WarpX
Full EM/ES PIC, Cartesian & cylindrical

ImpactX
Electrostatic PIC codes for accelerator physics

Wake-T
Quasi-static PIC, cylindrical

FBPIC
Full EM PIC, cylindrical

…

Output:

- particles
- fields on a mesh

openPMD: Open Standard for Particle-Mesh data

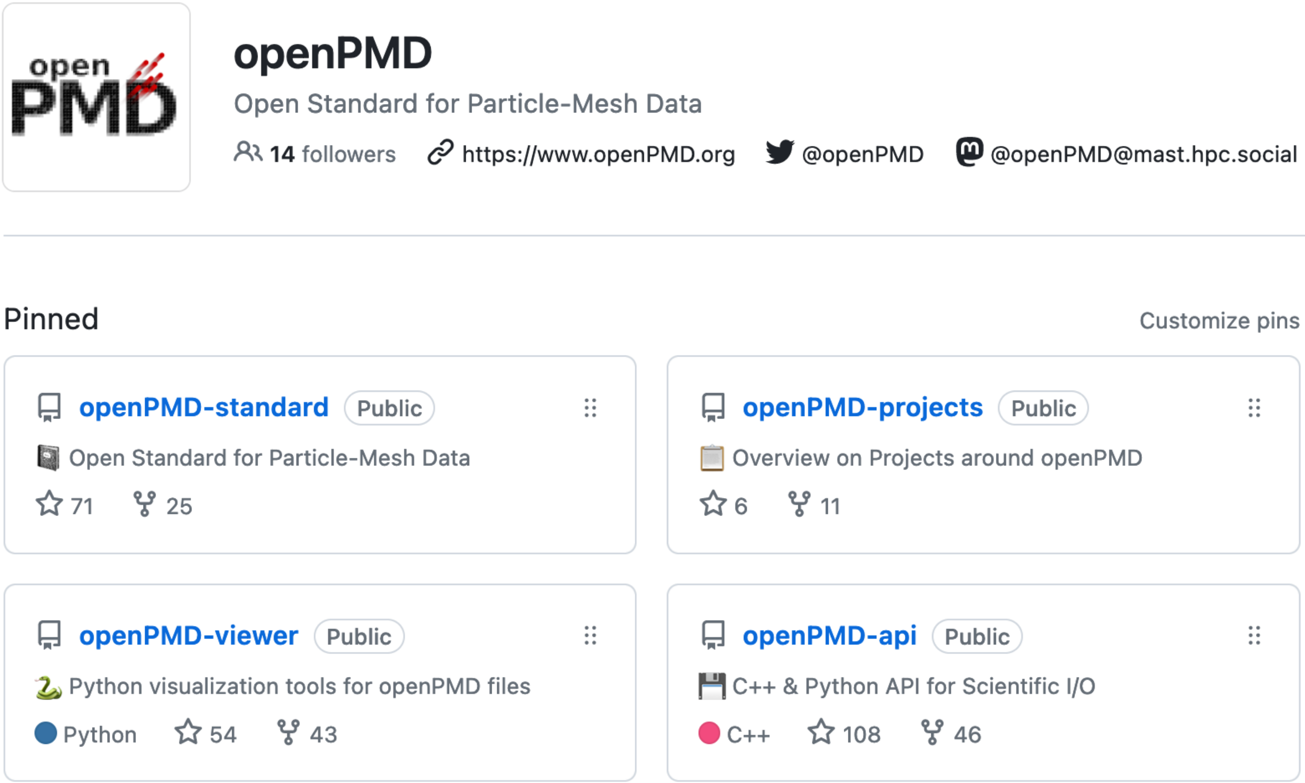
openPMD standard:

specifies how to store
particle and mesh data in:

- HDF5 files
  (end in `.h5`)
- ADIOS files
  (end in `.bp`)
- JSON files
  (end in `.json`)

Files that conform to this standard can use a **rich set of tools.**

github.com/openPMD



openPMD

Open Standard for Particle-Mesh Data

14 followers · https://www.openPMD.org · @openPMD · @openPMD@mast.hpc.social

Pinned | Customize pins

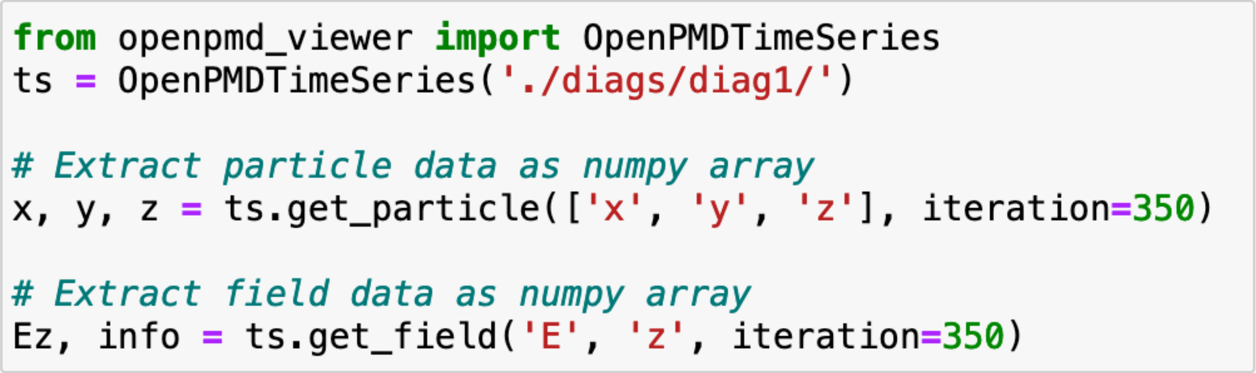
- **openPMD-standard** Public
  Open Standard for Particle-Mesh Data
  ☆ 71 · 25
- **openPMD-projects** Public
  Overview on Projects around openPMD
  ☆ 6 · 11
- **openPMD-viewer** Public
  Python visualization tools for openPMD files
  ● Python · ☆ 54 · 43
- **openPMD-api** Public
  C++ & Python API for Scientific I/O
  ● C++ · ☆ 108 · 46

Data analysis for openPMD files: openPMD-viewer

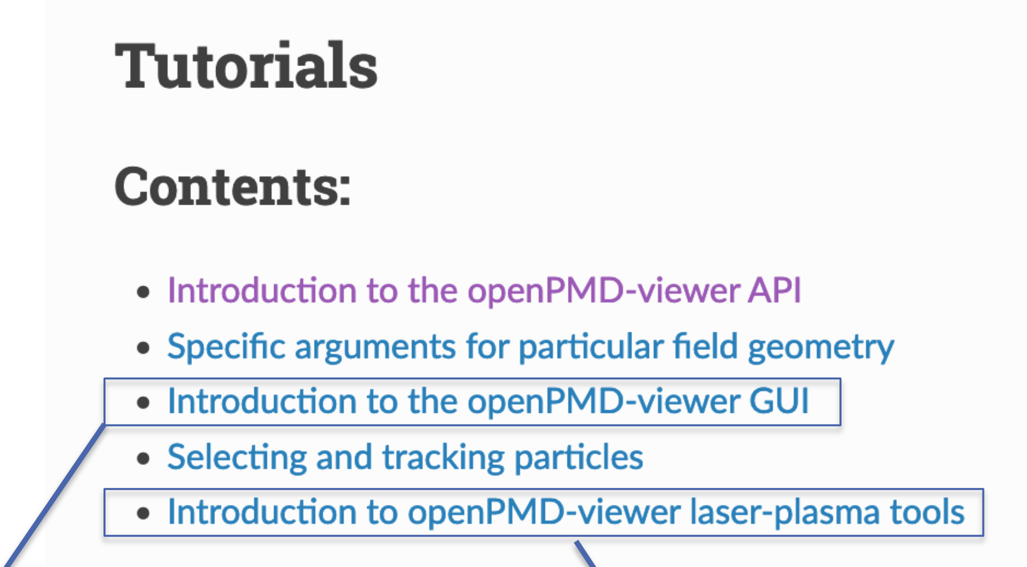
github.com/openPMD/openPMD-viewer

Documentation: openpmd-viewer.readthedocs.io

`openPMD-viewer` is a **Python tool** to **extract/analyze data** from openPMD files:



```python
from openpmd_viewer import OpenPMDTimeSeries
ts = OpenPMDTimeSeries('./diags/diag1/')

# Extract particle data as numpy array
x, y, z = ts.get_particle(['x', 'y', 'z'], iteration=350)

# Extract field data as numpy array
Ez, info = ts.get_field('E', 'z', iteration=350)
```



Tutorials

Contents:

- Introduction to the openPMD-viewer API
- Specific arguments for particular field geometry
- Introduction to the openPMD-viewer GUI
- Selecting and tracking particles
- Introduction to openPMD-viewer laser-plasma tools

Interactive "GUI"
in Jupyter notebooks

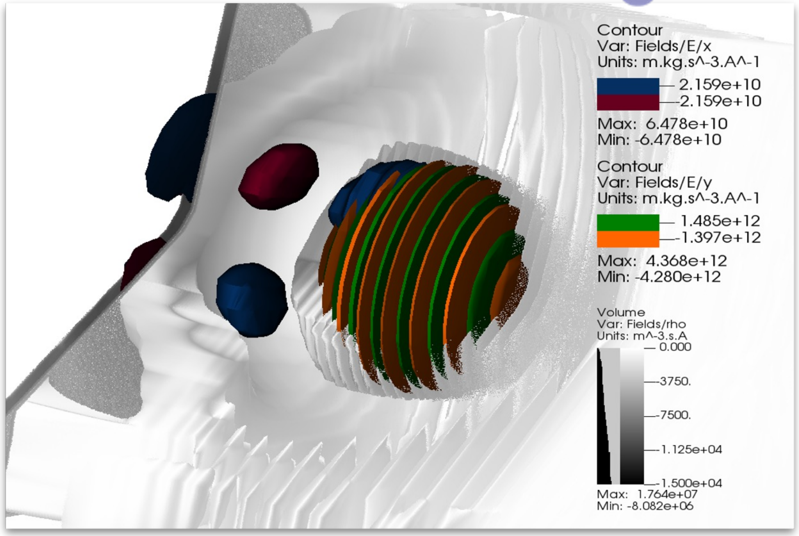
Convenient tools to compute:

- Bunch properties, e.g. emittance, etc.
- Laser properties, e.g. waist, a0, etc.

3D visualization tools of openPMD files

openPMD files can be read with:

ParaView VisIt

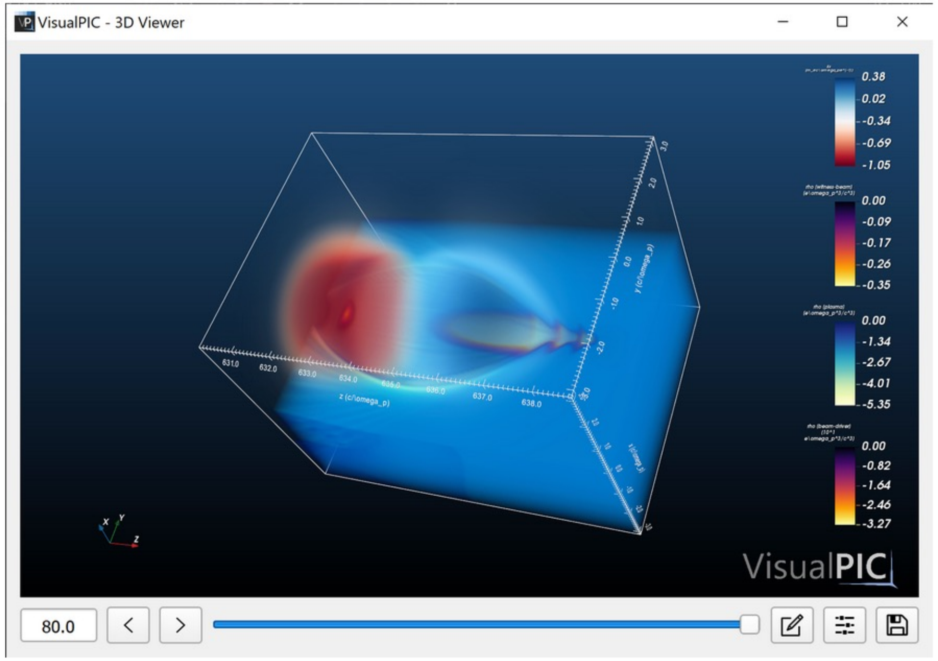


More info:

warpx.readthedocs.io/en/latest/dataanalysis/paraview.html

warpx.readthedocs.io/en/latest/dataanalysis/visit.html

openPMD files can also be read with **VisualPIC**, an in-house 3D visualization tool, for PIC data



github.com/AngelFP/VisualPIC

How to create openPMD files, as input to the simulation?

Optional input:

- **Initial beam of particles:** generate the beam in Python and use **`openPMD-api`** to create the openPMD file openpmd-api.readthedocs.io/
- **Initial laser pulse:** New library: **`lasy`** lasydoc.readthedocs.io/

open PMD file

HiPACE++
Quasi-static PIC, 3D Cartesian

WarpX
Full EM/ES PIC, Cartesian & cylindrical

ImpactX
Electrostatic PIC codes for accelerator physics

Wake-T
Quasi-static PIC, cylindrical

FBPIC
Full EM PIC, cylindrical

…

Output:

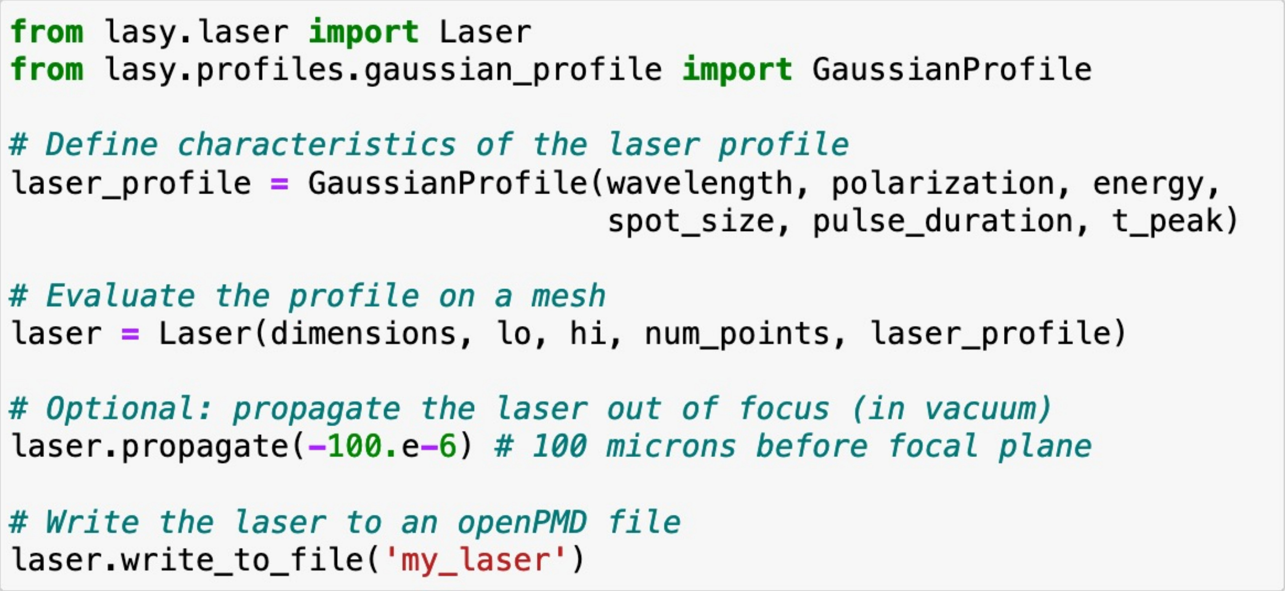
- particles
- fields on a mesh

open PMD file

lasy: a library to initialize complex laser pulses

github.com/LASY-org/lasy

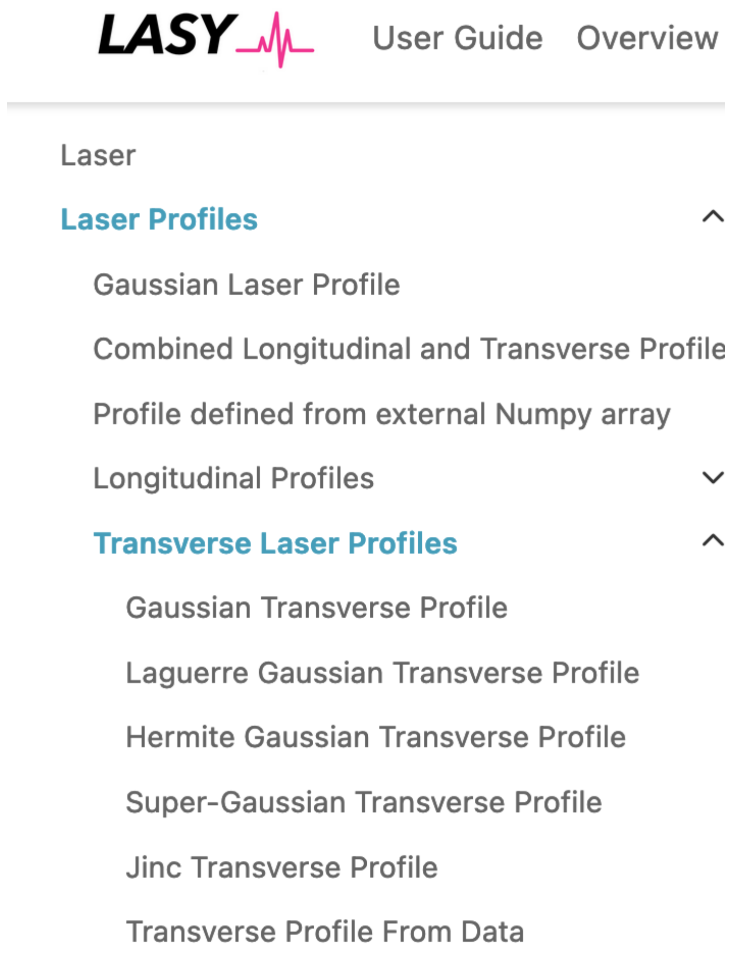
Documentation: lasydoc.readthedocs.io



```python
from lasy.laser import Laser
from lasy.profiles.gaussian_profile import GaussianProfile

# Define characteristics of the laser profile
laser_profile = GaussianProfile(wavelength, polarization, energy,
                                spot_size, pulse_duration, t_peak)

# Evaluate the profile on a mesh
laser = Laser(dimensions, lo, hi, num_points, laser_profile)

# Optional: propagate the laser out of focus (in vacuum)
laser.propagate(-100.e-6) # 100 microns before focal plane

# Write the laser to an openPMD file
laser.write_to_file('my_laser')
```

powered by github.com/hightower8083/axiprop

Growing list of laser profiles implemented:



- Laser
- Laser Profiles
  - Gaussian Laser Profile
  - Combined Longitudinal and Transverse Profile
  - Profile defined from external Numpy array
  - Longitudinal Profiles
  - Transverse Laser Profiles
    - Gaussian Transverse Profile
    - Laguerre Gaussian Transverse Profile
    - Hermite Gaussian Transverse Profile
    - Super-Gaussian Transverse Profile
    - Jinc Transverse Profile
    - Transverse Profile From Data

optimas: design optimization using simulations

github.com/optimas-org/optimas

Documentation: optimas.readthedocs.io/

optimas facilitates this workflow for simulations **on HPC clusters**

In many cases: we search for the **combination of design parameters** (e.g. plasma density, laser intensity, etc.) that **maximize a given objective** (e.g. energy spread of final beam).

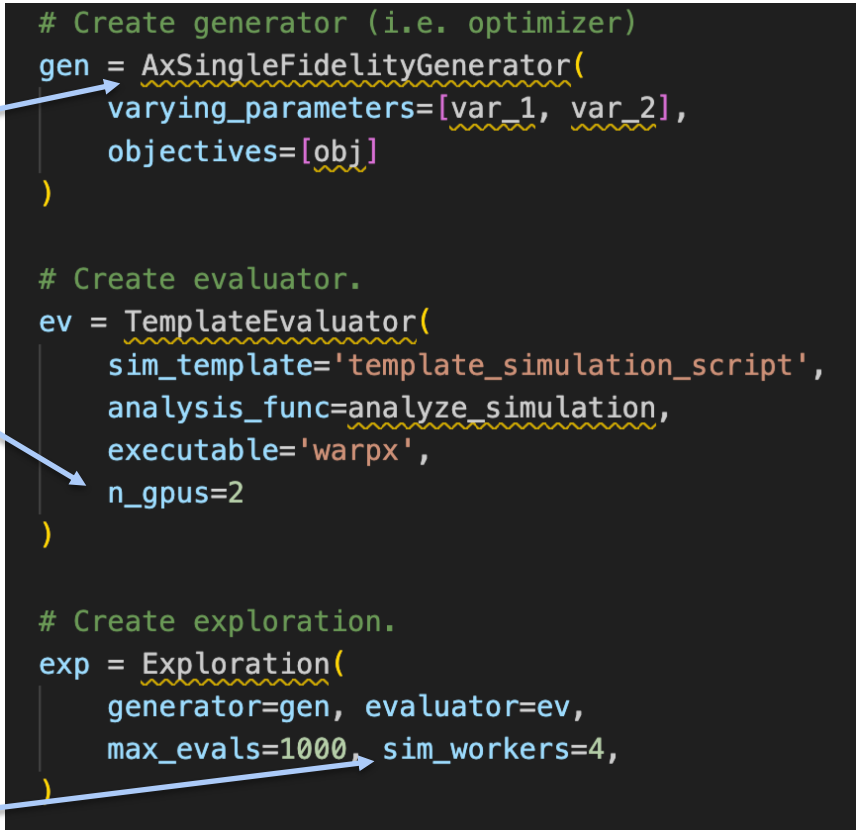
This typically requires to run **many simulations**, to search the space of design parameters.

e.g.
Jalas et al., Bayesian Optimization of a Laser-Plasma Accelerator, PRL, 2021

Bayesian optimizer

Use 2 GPUs per simulation

Run 4 simulations in parallel



```
# Create generator (i.e. optimizer)
gen = AxSingleFidelityGenerator(
    varying_parameters=[var_1, var_2],
    objectives=[obj]
)

# Create evaluator.
ev = TemplateEvaluator(
    sim_template='template_simulation_script',
    analysis_func=analyze_simulation,
    executable='warpx',
    n_gpus=2
)

# Create exploration.
exp = Exploration(
    generator=gen, evaluator=ev,
    max_evals=1000, sim_workers=4,
)
```

Thank you