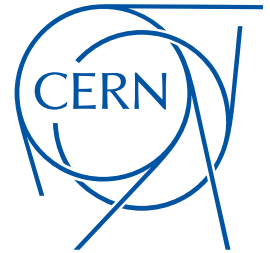
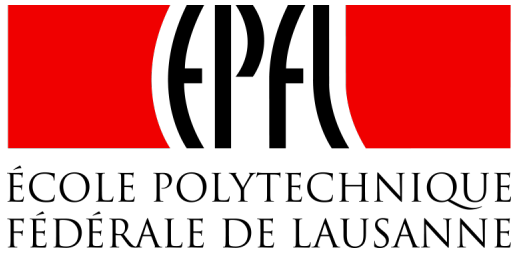




Work supported by the Swiss State Secretariat for Education, Research and Innovation SERI

Python tools for simulating beam dynamics

L. Mether, K. Li and A. Oeftiger
for the PyHEADTAIL developer team

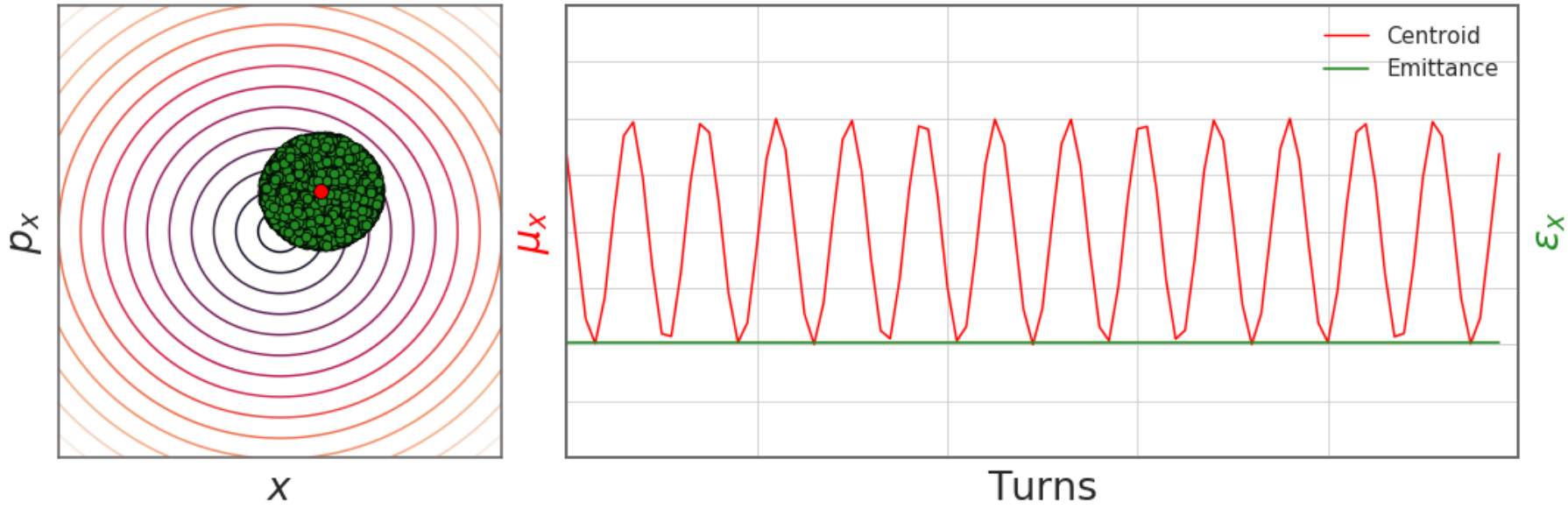
PyHEP 2018 Workshop, Sofia, Bulgaria
June 7th – 8th, 2018

Beam dynamics

- Beam dynamics essentially studies the evolution of the phase space variables i.e. the generalized coordinates and canonically conjugate momenta of a beam in an accelerator

- This evolution is primarily determined by external force fields, e.g.
  - Magnets: bending, focusing…

Filamentation and emittance growth due to initial offset (and non-linearities)

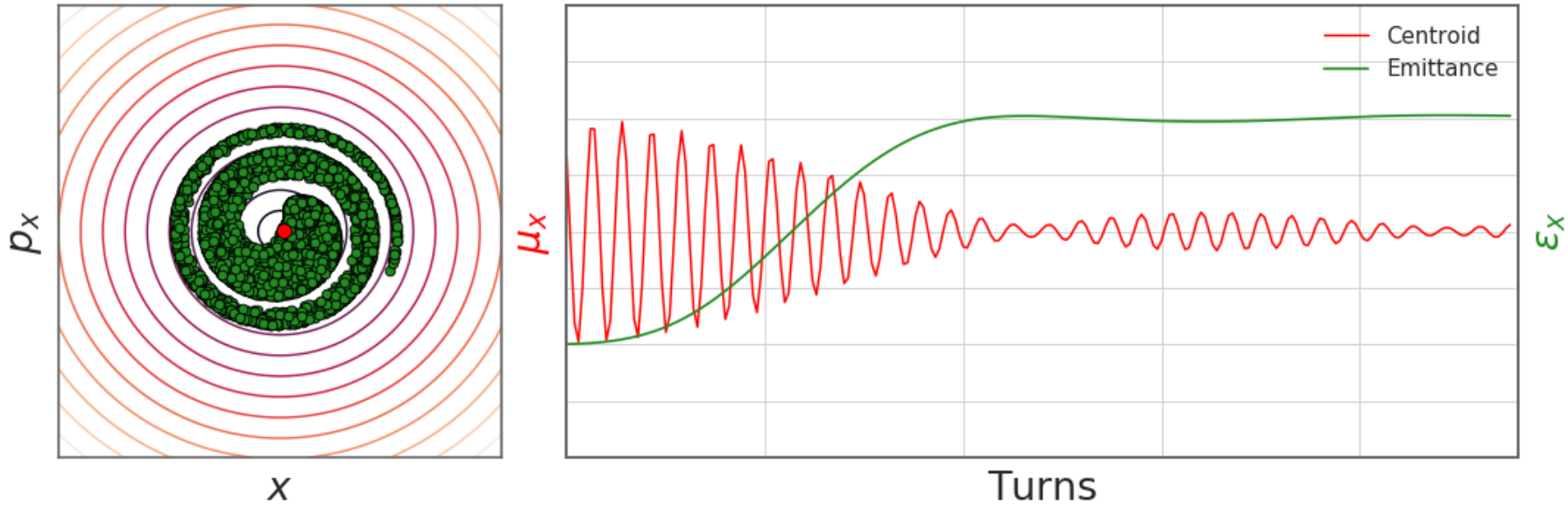


Beam dynamics

- Beam dynamics essentially studies the evolution of the phase space variables i.e. the generalized coordinates and canonically conjugate momenta of a beam in an accelerator

- This evolution is primarily determined by external force fields, e.g.
  - Magnets: bending, focusing…

Filamentation and emittance growth due to initial offset (and non-linearities)



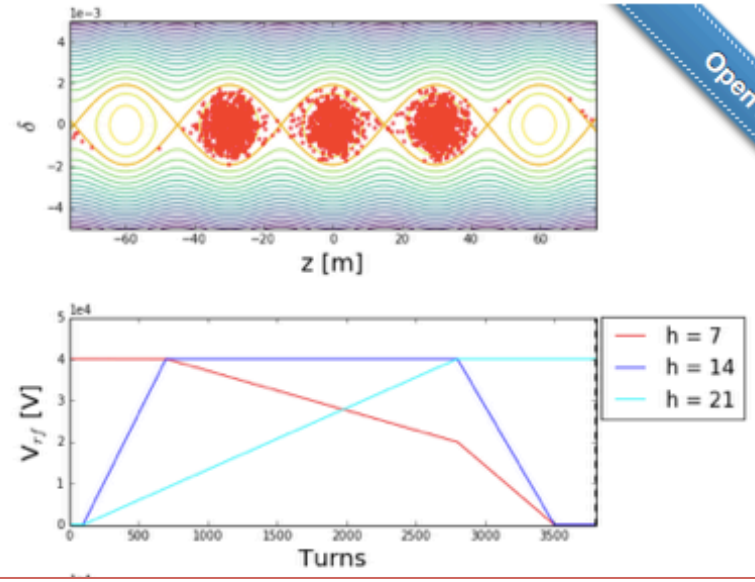
Beam dynamics

- Beam dynamics essentially studies the evolution of the phase space variables i.e. the generalized coordinates and canonically conjugate momenta of a beam in an accelerator

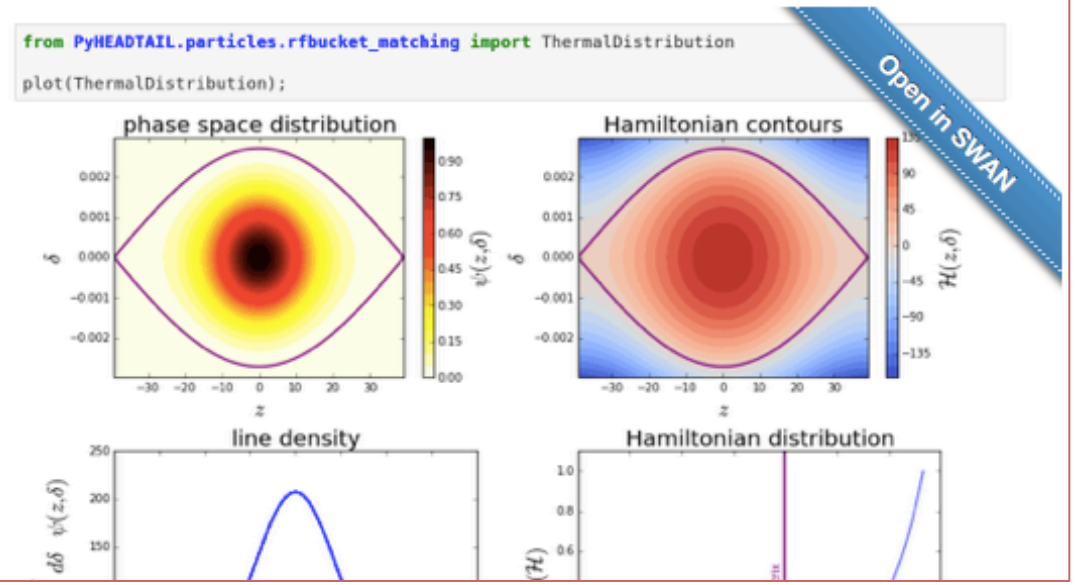
- This evolution is primarily determined by external force fields, e.g.
  - Magnets: bending, focusing…
  - RF fields: confinement in bunches, acceleration…

[SWAN Beam dynamics gallery]

PS Triple Bunch Splitting



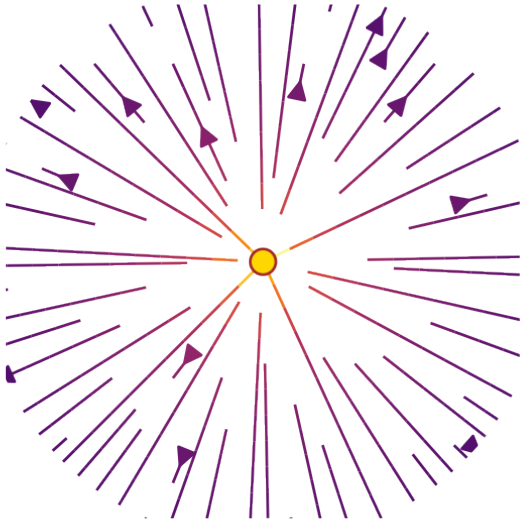
RF Bucket Matching



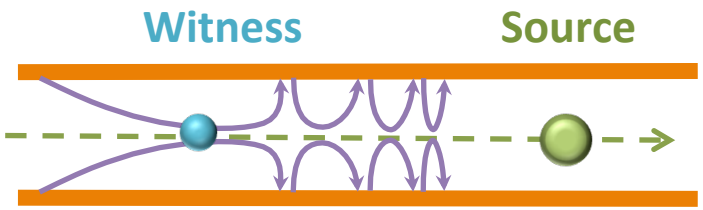
Collective effects

- The evolution can be affected by collective effects – fields that are caused by the beam itself and depend on its phase space distribution, e.g.
  - Space charge: Coulomb repulsion between particles within beam
  - Wake fields: beam-induced image charges and currents in the surrounding structure
  - Electron cloud: beam-induced exponential electron accumulation

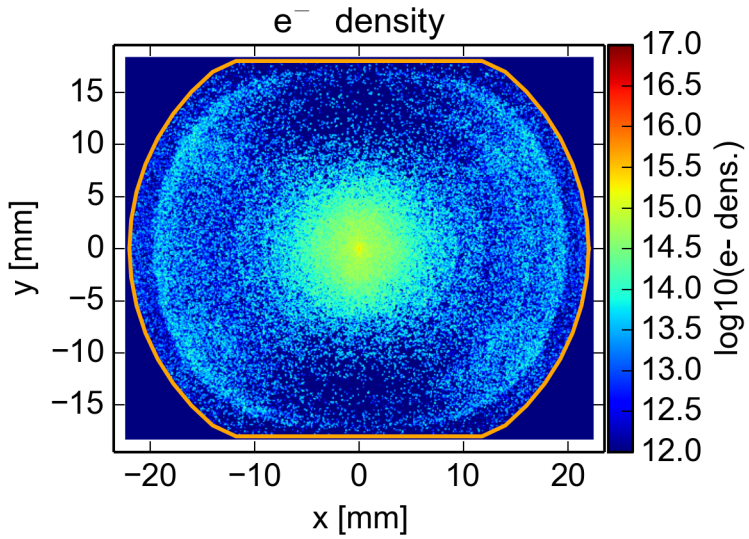
Space charge



Wake fields



Electron cloud

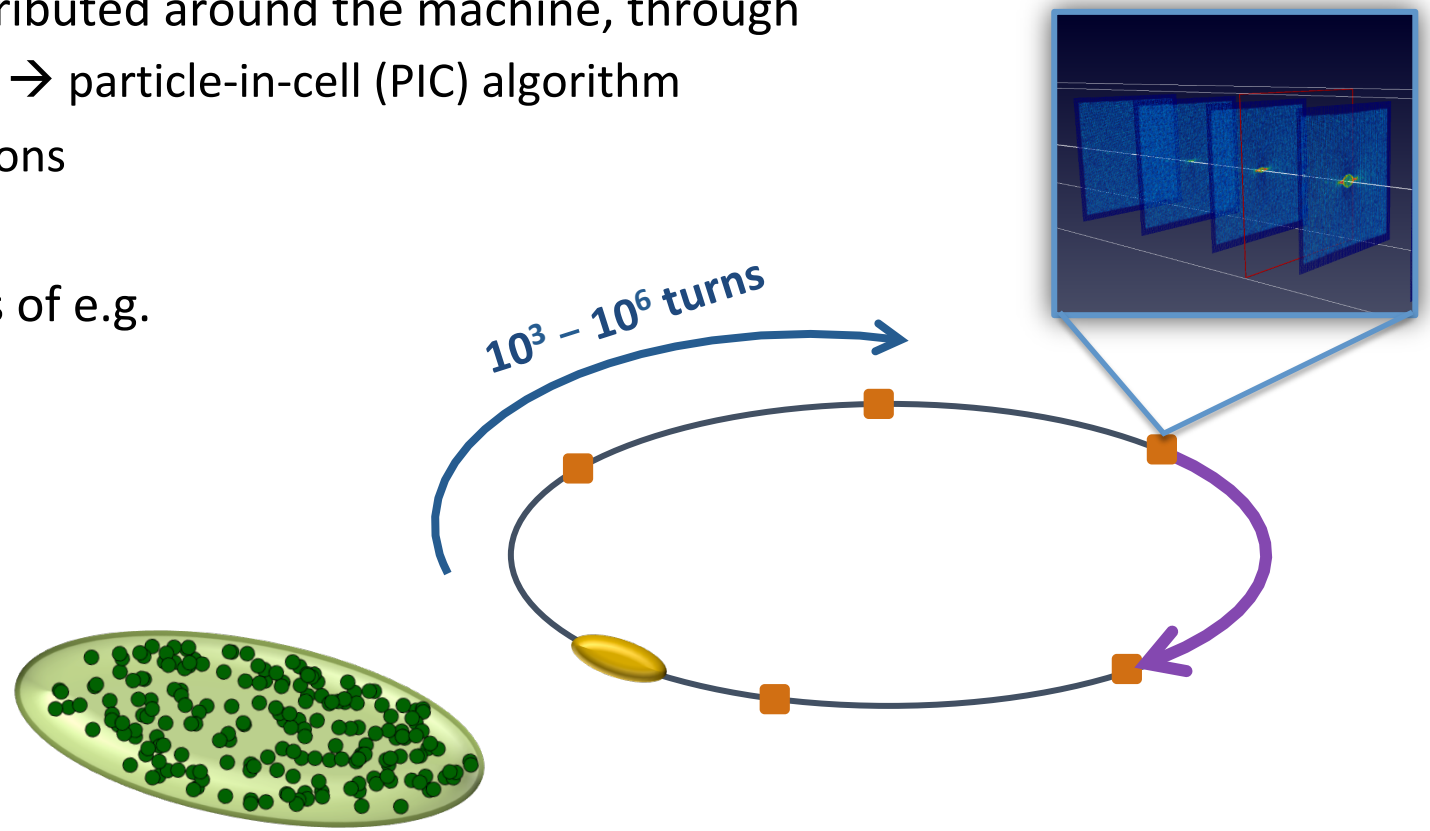


The simulation model

- The beam is modelled as macro-particles, typically $N \approx 10^6$
  - The effect of external force fields can be applied with analytical expressions

- Collective effects are modelled as kicks distributed around the machine, through
  - solving for the electric field of the particles → particle-in-cell (PIC) algorithm
  - or a convolution over macro-particle positions

- Similar models and dynamics to simulations of e.g. large scale structure, plasma physics…



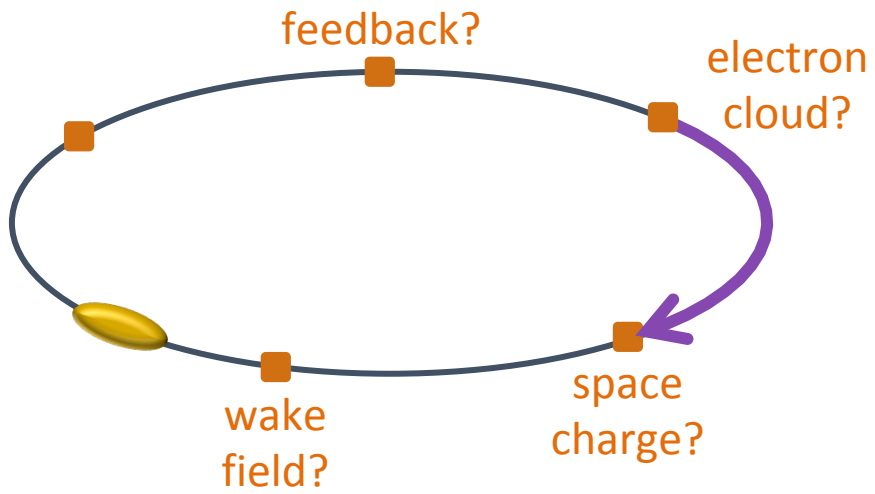
Why Python?

- Are compiled languages not ideal for simulations that rely on repeating a few core algorithms?
  - Indeed, we have (had) such codes, and the good ones are (were) robust, powerful and fast
  - However, we would like our tools to also be versatile, extendible and easily maintainable

- A large part of the users need to modify/extend the code to study specific physics cases
  - Development should be as simple and fast as possible
  - We also want to avoid several independently developing forks of the same code (old oak vs. bonsai)

- Different users want to use different combinations of ingredients
  - We need a modular structure

  → Object-oriented programming



Python

- Because we can make it:

Simple

- Easy to fix
- Easy to read
- Easy to maintain

Modular

- Easy to extend
- Easy to combine
- Easy to maintain

Dynamic

- Fast
- Flexible
- Interactive

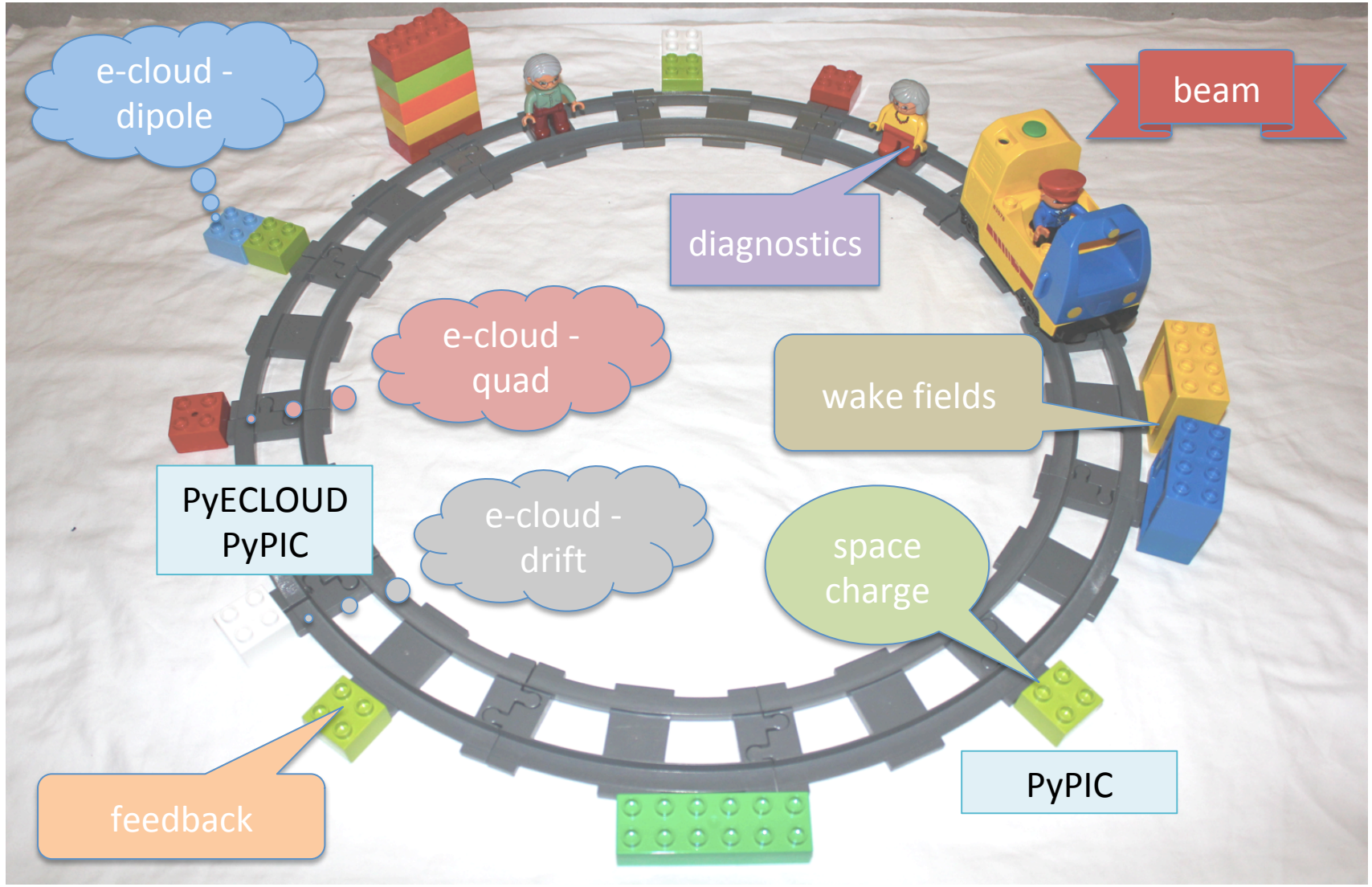
PyHEADTAIL

A library of beam dynamics tools:

- particles
- machines
- trackers
- space charge, wake fields
- monitors…

In fact we have a collection of compatible libraries:

- PyECLOUD: e-cloud dynamics
- PyPIC: PIC routines
- Py(CERN)Machines…



Running PyHEADTAIL

ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE

- Instead of running the simulations using the classic:

  input file(s) + main executable

  we can simply use a Python script

→ The full power of Python is available in the input file

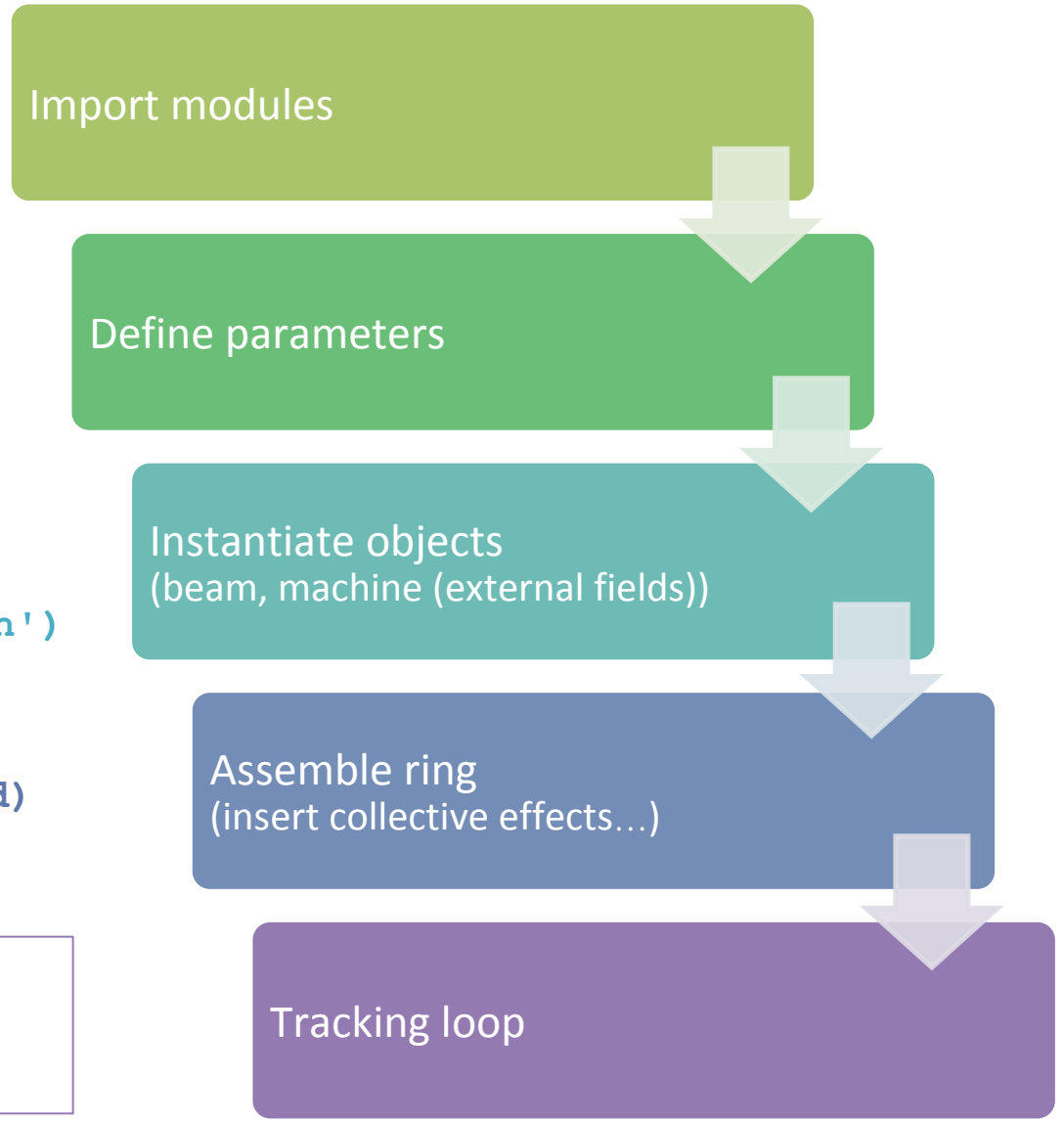
→ Simple, highly customizable, interactive

```
machine = SPS(n_segments = n_seg,
              machine_configuration = 'Q26-injection')

# install eclouds
machine.install_after_each_transverse_segment(ecloud)
```

→ The "simulation":

```
for i in range(nturns):
    for m in one_turn_map:
        m.track(beam)
```

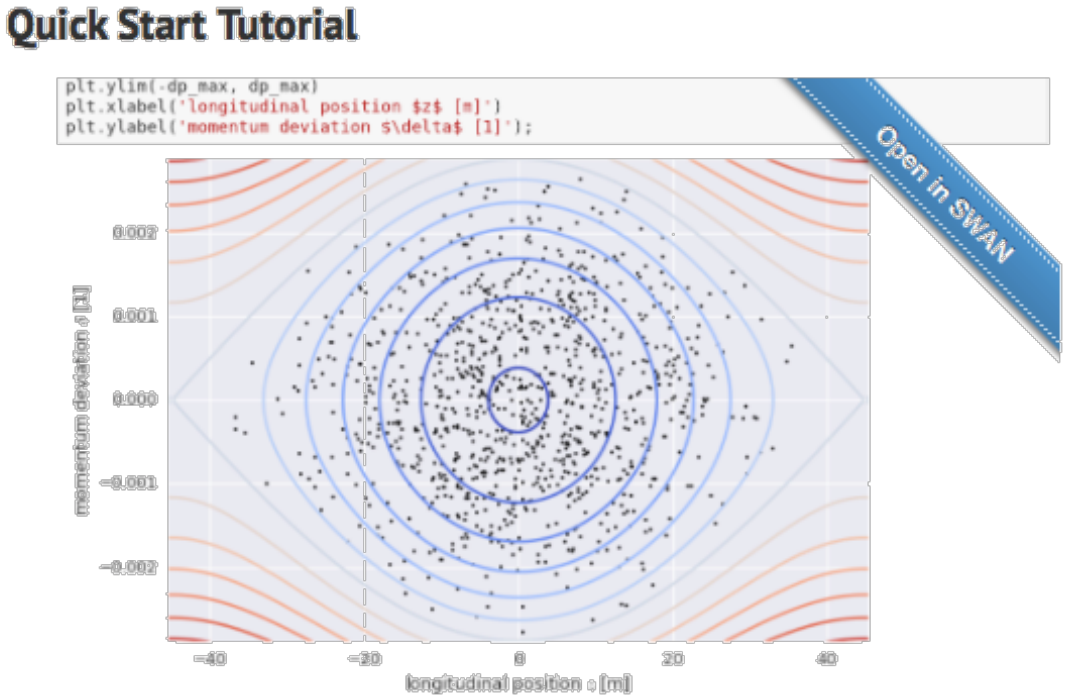


Notebooks and SWAN

- PyHEADTAIL documentation is in the form of Jupyter notebooks: PyHEADTAIL-playground
  - Instructive notebooks that outline the different use cases of the code
  - Have been used at several accelerator physics schools etc.
  - Work also as a testing suite

- The PyHEADTAIL-playground could be put on SWAN after PyHEADTAIL was added to pip
  - Makes the notebooks even more accessible, since no local installation or specific environment needed

- Accompanied by the **beam dynamics gallery**: illustrative notebooks with visual representations
  - Accessible from SWAN front page

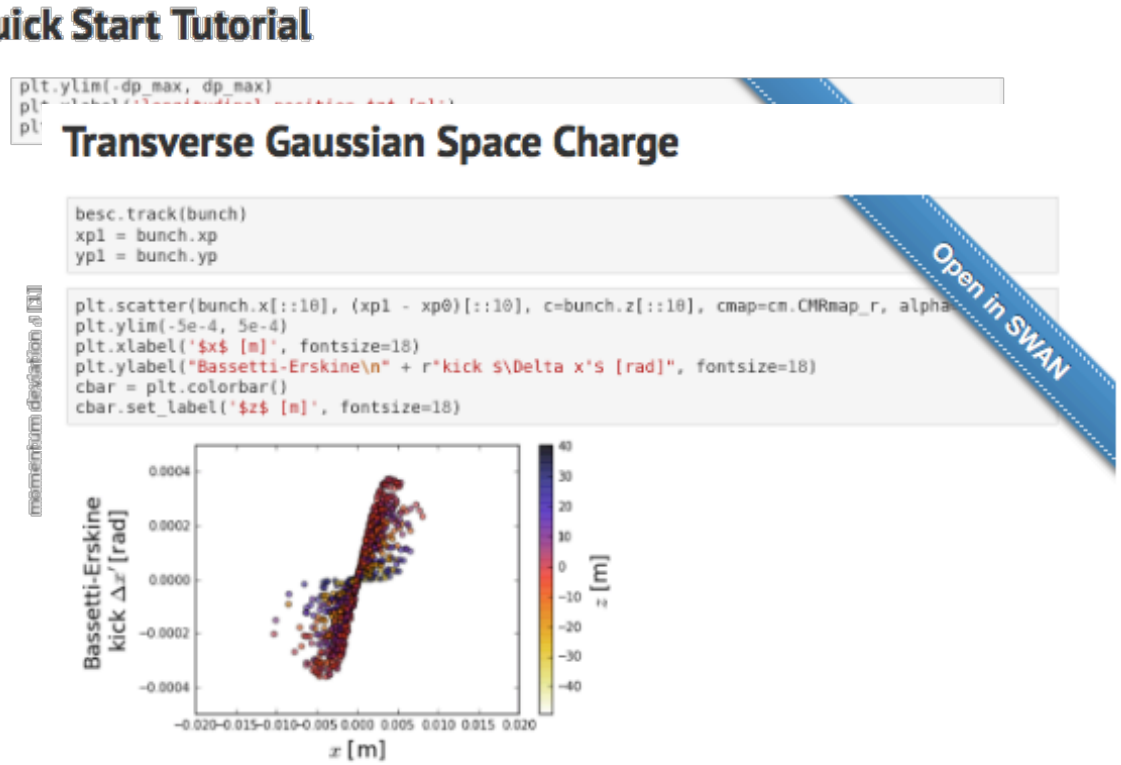


PyHEADTAIL on SWAN

- Beam dynamics gallery and PyHEADTAIL-playground
  - Easy introduction for newcomers, students, etc
  - Can be useful also for non-experts, e.g. machine operators for visualisation
  - Motivation among users to add additional examples

- In addition to developing and using instructive material, more generally PyHEADTAIL on SWAN can be useful for quick simulations and for testing and exploring purposes
  - Full “production” simulations typically done on batch service or HPC clusters

- The prospect of using SWAN as an interface and manager for batch jobs could be interesting for our users (but mainly if access to significant resources)



Performance

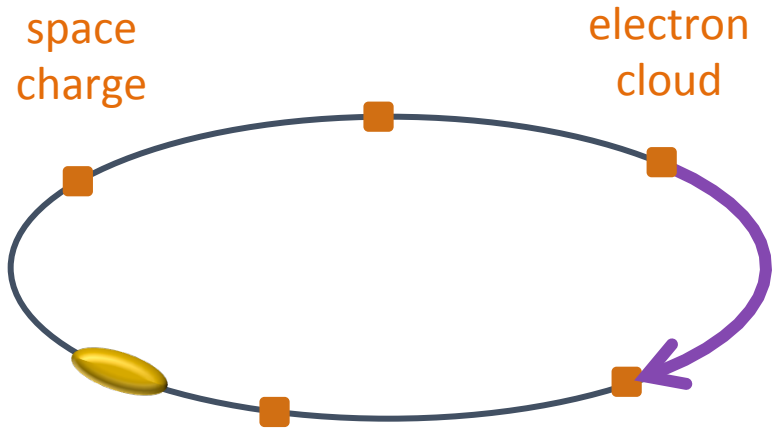
- To try to maximize the performance, we use:
  - NumPy `ndarrays` for basic data structures
  - SciPy where appropriate
  - FFTW with **pyFFTW**
  - Output in HDF5 with **h5py**…
- Some of the core algorithms are implemented in C(++) or Fortran, binding to Python through
  - Cython, **f2py**
- With the caveat that our codes are not necessarily maximally optimized, to be fair, they are not as fast as they could be e.g. in C
  - However, an increase in runtime of some tens of per cent feels like a small price to pay for the other benefits

Parallelisation

- Many of our use cases are quite demanding in terms of computation time, despite optimization
- Abstract parallelisation layers implemented to keep main code unchanged
- Different physics models require different parallelisation methods:

PyCUDA

Space charge instantaneously couples all particles within the bunch → the same calculation to be done for large number of particles − suited for GPU-parallelization



mpi4py

The electron cloud interaction must be done sequentially over several slices of a single bunch → parallelisation only over different kicks, which are treated slice-by-slice

simulations still take weeks/months

- Context manager for GPU-parallelisation handles all dispatches of function calls according to context (GPU/CPU)
- Parallel ring simulator (PyPARIS) distributes the different electron cloud kicks in the machine over several processors and passes beam slices around the ring

Conclusion & final remarks

ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE

- Python allows us to keep our tools concise, neat and organised, yet versatile and dynamic, without compromising much on performance
  - Even if Python isn't the ideal language for the rest of eternity, it provides a good starting point for the next step

- Jupyter Notebooks are great for documentation, education, and exploring
  - PyHEADTAIL on SWAN makes them even more easily accessible and available
  - SWAN as an interface for batch jobs could be very interesting for us

- Python 2 to 3
  - Currently our tools are in Python 2.7
  - Many parts of the codes are compatible with Python 3, but migration needs to be done collectively
  - Foreseen for LS2 when the machines will need less follow-up

- https://github.com/PyCOMPLETE

A brief history of codes