Code Development for Collective Effects

K. Li, H. Bartosik, G. Iadarola, L. Mether, A. Oeftiger,
A. Passarelli, A. Romano, G. Rumolo, M. Schenk

HB2016, 3-8 July, Malmo, Sweden

CDCE2016!?

K. Li, H. Bartosik, G. Iadarola, L. Mether, A. Oeftiger,
A. Passarelli, A. Romano, G. Rumolo, M. Schenk

HB2016, 3-8 July, Malmo, Sweden

Context & Outline

Context:

Collective effects pose **important limitations** in modern **high brightness circular accelerators**. Studying and understanding these effects can help overcoming these limitations.

Numerical methods form one of the **fundamental contemporary tools** used for this purpose. We discuss some modern approaches for numerical modeling of collective effects and show some use-cases employing the example of the **PyHEADTAIL framework**.

Outline:

1. Introduction
2. Basic model of the accelerator-beam system
3. Modern approaches and program architectures
4. Performance considerations
5. Applications, present status and perspectives

Context & Outline

Summary:

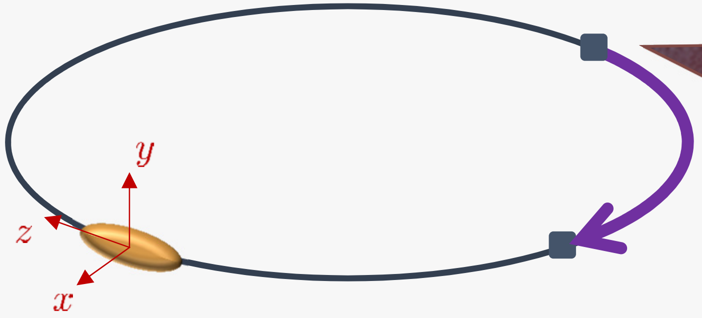
Brief outline of the nature of collective effects and why we study them with computer simulation programs.

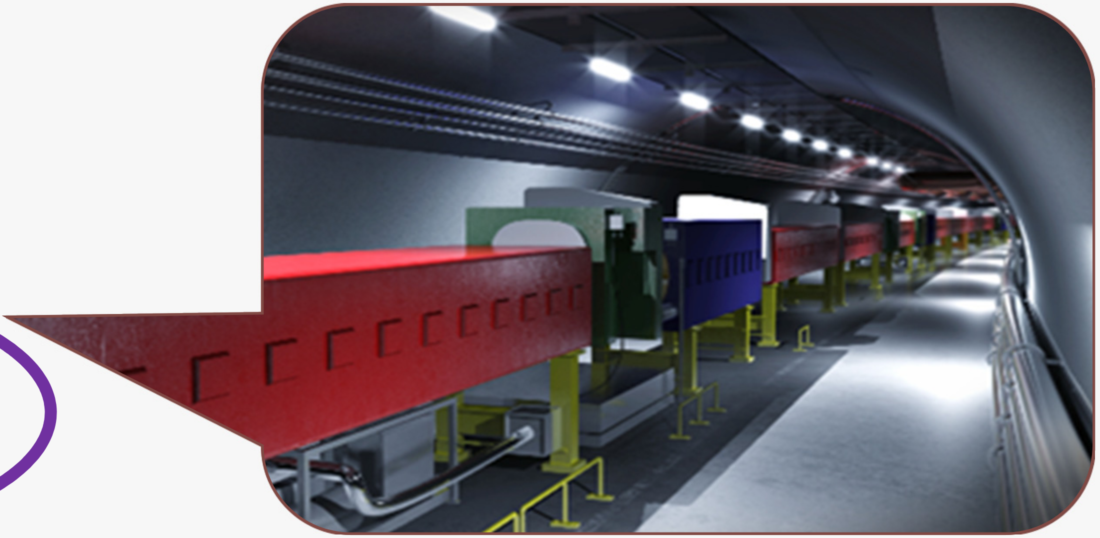
Outline:

1. Introduction
2. Basic model of the accelerator-beam system
3. Modern approaches and program architectures
4. Performance considerations
5. Applications, present status and perspectives

Modeling collective effects

- Beam dynamics deals with studying the evolution of the phase space variables i.e., the generalized coordinates and canonically conjugate momenta of a beam in an accelerator
- Generally this evolution is determined by external force fields (magnets, electrostatic fields, RF fields)

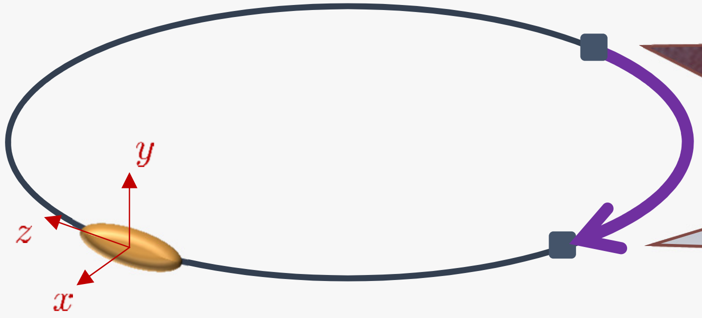


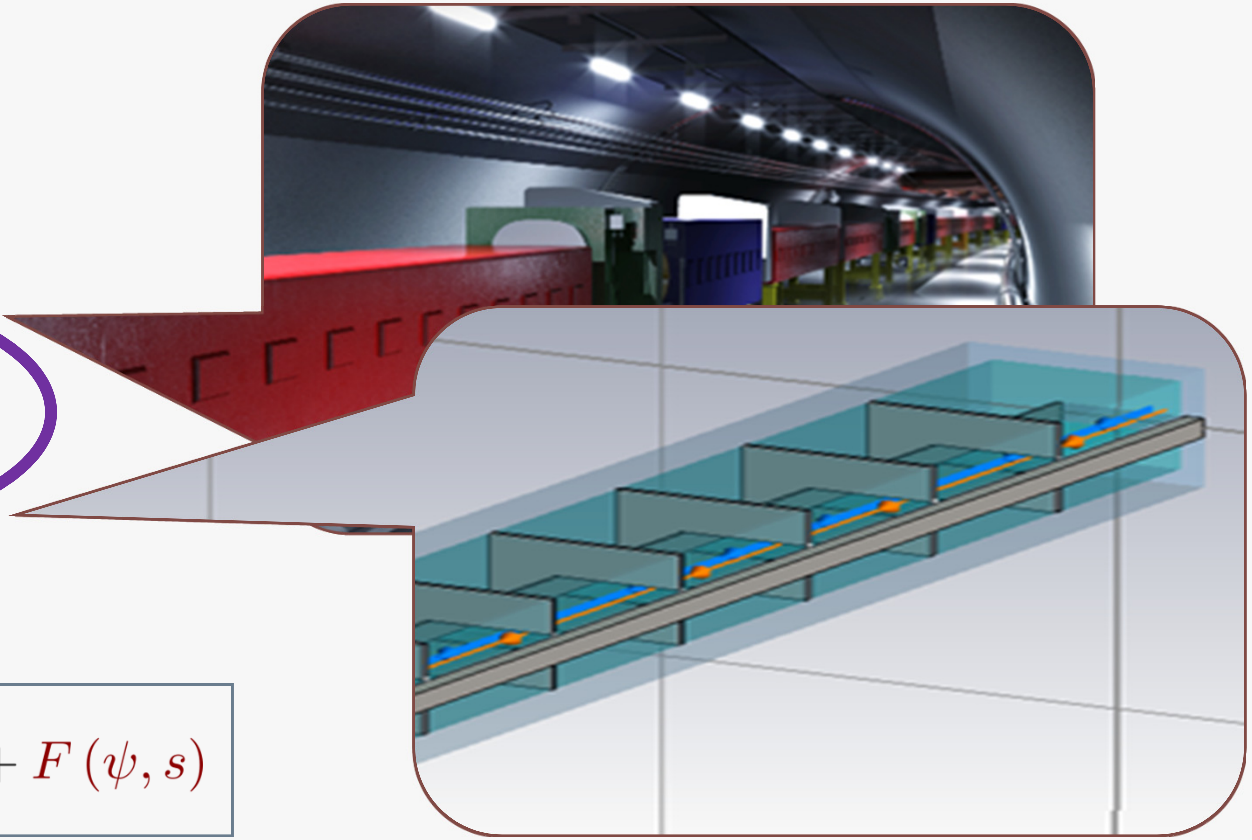


$$\frac{d}{dt}\psi\left(x, x', y, y', z, \delta\right) = F_{\text{extern}}(x, y, z, s)$$

Modeling collective effects

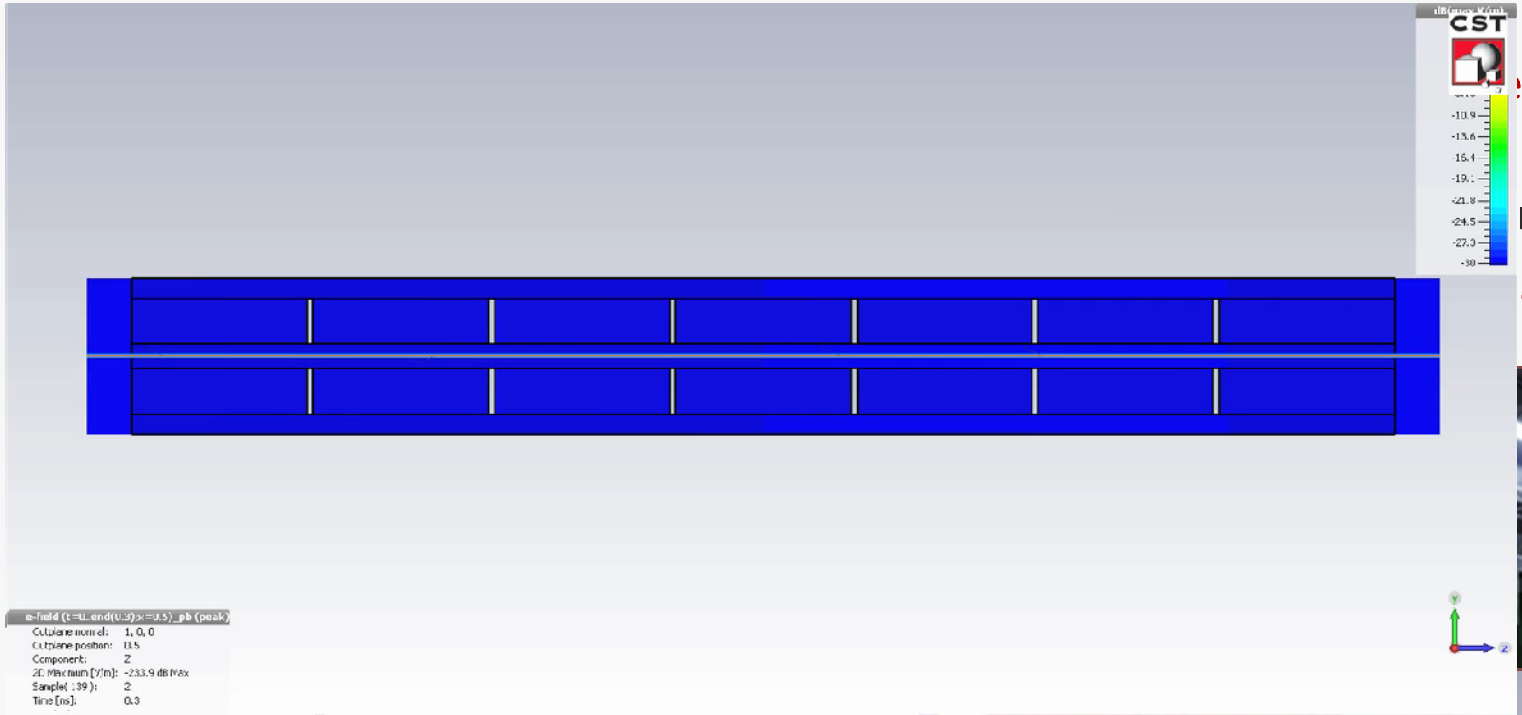
- Beam dynamics deals with studying the evolution of the phase space variables i.e., the generalized coordinates and canonically conjugate momenta of a beam in an accelerator
- Generally this evolution is determined by external force fields (magnets, electrostatic fields, RF fields)
- Collective effects add to this fields that depend on the phase space distribution function itself (space charge, wake fields)



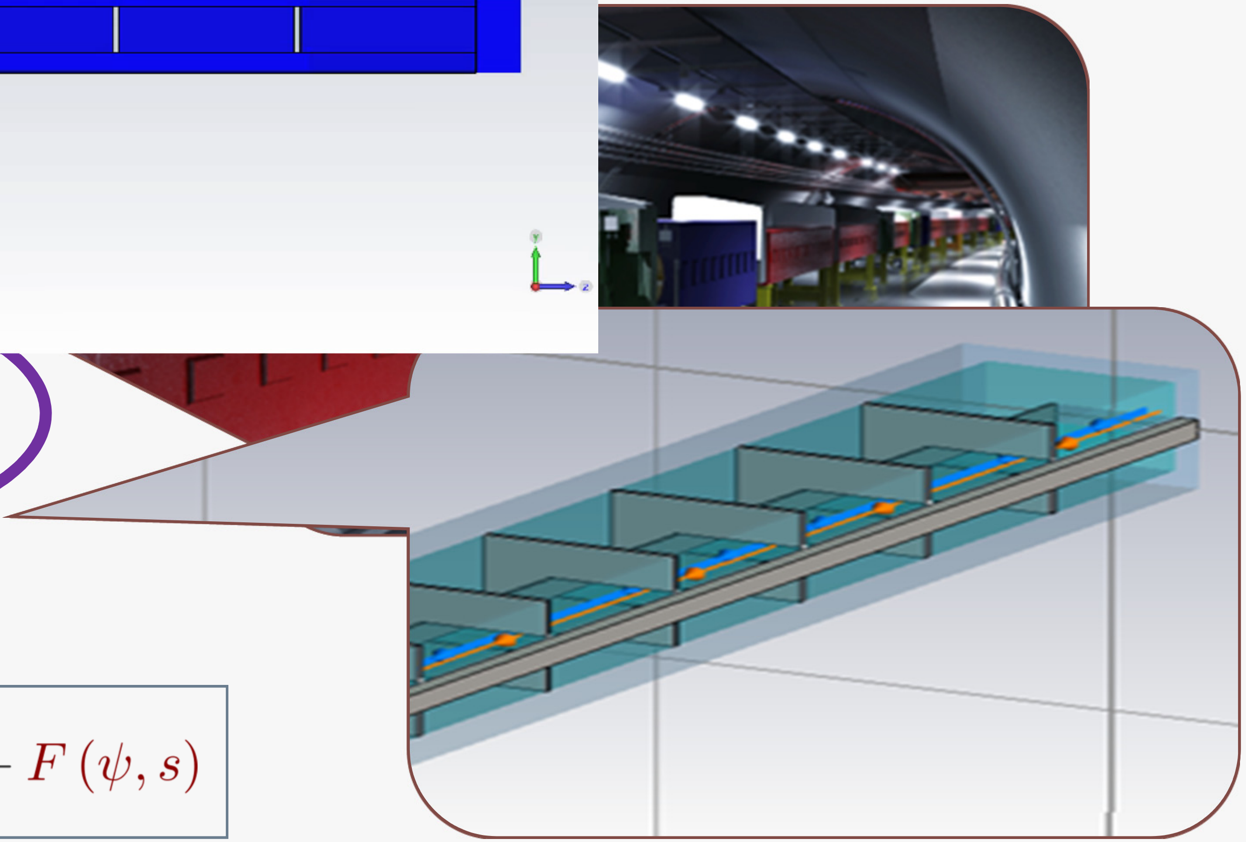


$$\frac{d}{dt}\psi\left(x, x', y, y', z, \delta\right) = F_{\mathrm{extern}}(x, y, z, s) + F\left(\psi, s\right)$$

Modeling collective effects



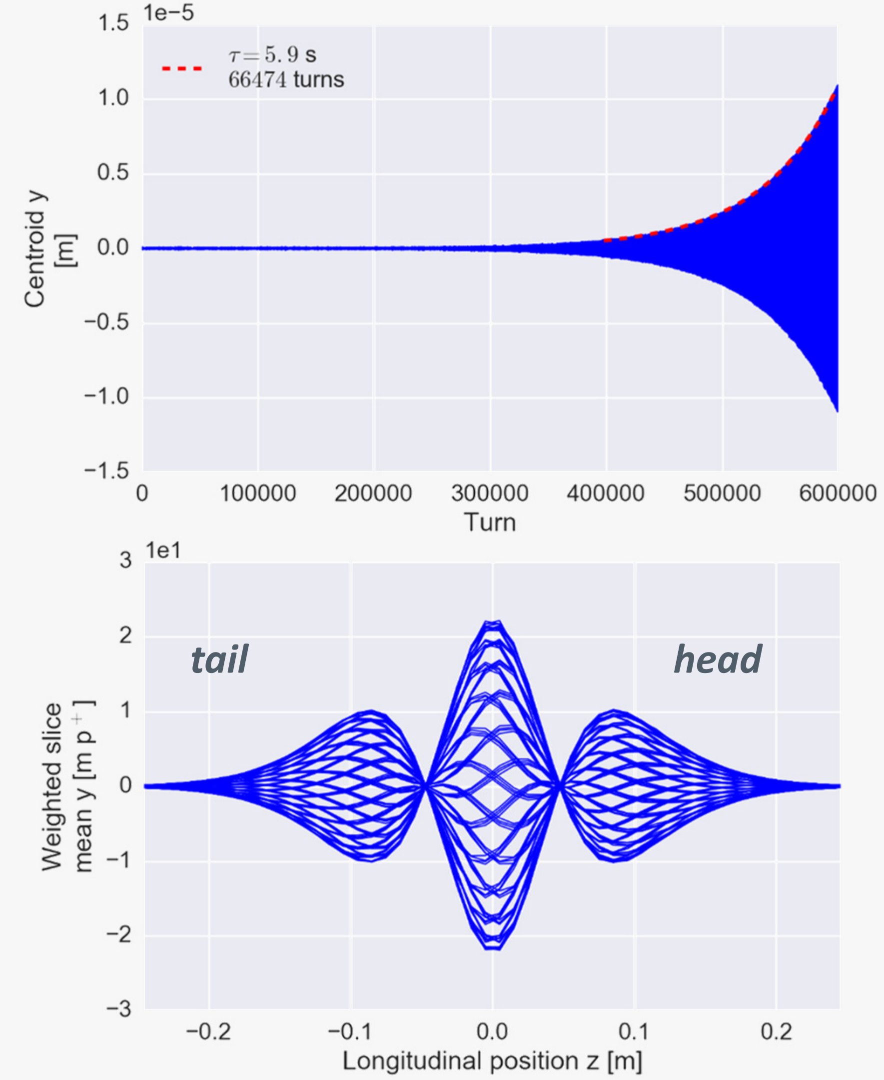
- …es i.e., the generalized coordinates and
- …rostatic fields, RF fields)
- …on function itself (space charge, wake



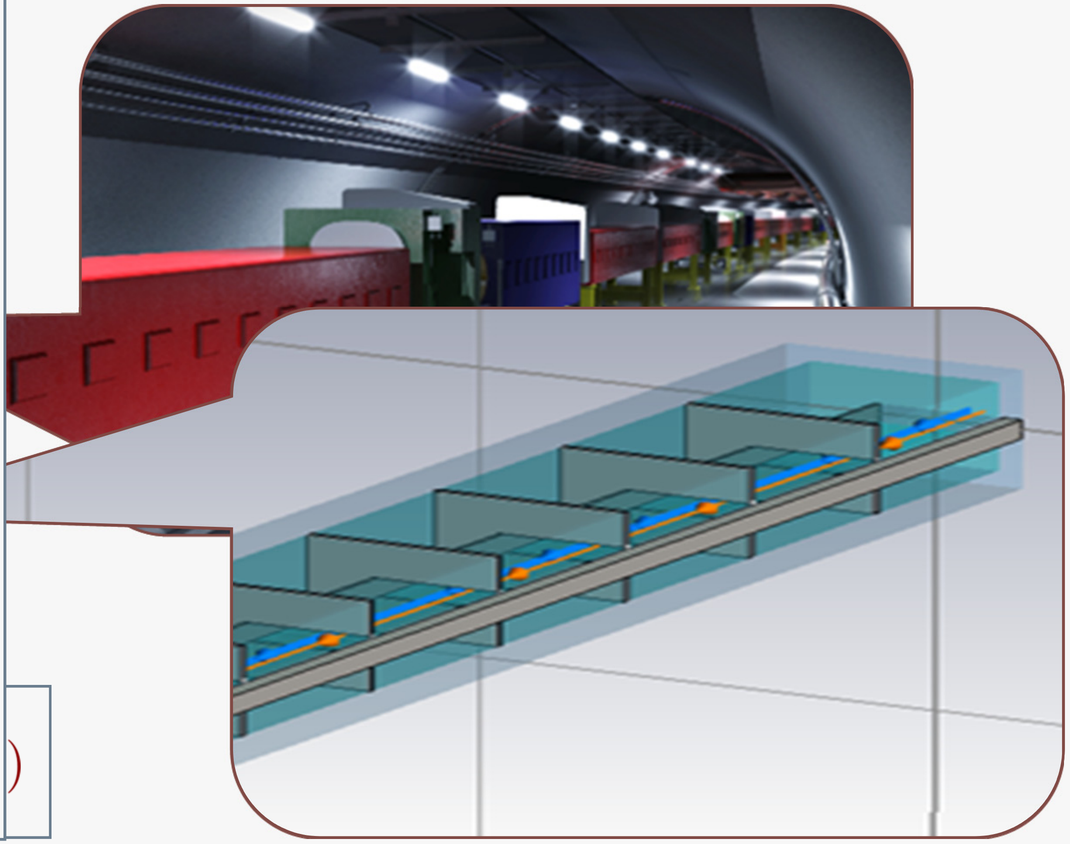
$$\frac{d}{dt}\psi\left(x, x', y, y', z, \delta\right) = F_{\text{extern}}(x, y, z, s) + F\left(\psi, s\right)$$

Modeling collective effects

- Be… phase space variables i.e., the generalized coordinates and ca… …ator
- Ge… fields (magnets, electrostatic fields, RF fields)
- Co… phase space distribution function itself (space charge, wake fie…



Courtesy Michael Schenk

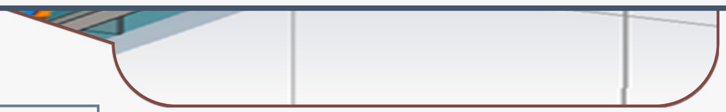


$\frac{d}{dt}\psi$ …)

Modeling collective effects

- Beam dynamics deals with studying the evolution of the phase space variables i.e., the generalized coordinates and canonically conjugate momenta of a beam in an accelerator
- Generally this evolution is determined by external force fields (magnets, electrostatic fields, RF fields)
- Collective … charge, wake fields)

- For a **multi-particle system** this self-consistency equation becomes arbitrarily complex and practically **impossible to solve**
- Obtaining the **multi-particle dynamics** requires **computer simulation codes**



$$\frac{d}{dt}\psi\left(x, x', y, y', z, \delta\right) = F_{\text{extern}}(x, y, z, s) + F\left(\psi, s\right)$$

Context & Outline

Summary:

Brief explanation of the numerical modelling of the accelerator-beam system employed for computer simulation programs to study collective effects in circular accelerators.

Outline:

1. Introduction
2. Basic model of the accelerator-beam system
3. Modern approaches and program architectures
4. Performance considerations
5. Applications, present status and perspectives

The PyHEADTAIL Model

Coordinate system

$$\left[\begin{pmatrix} x_i \\ x'_i \end{pmatrix} \begin{pmatrix} y_i \\ y'_i \end{pmatrix} \begin{pmatrix} z_i \\ \delta_i \end{pmatrix}\right]_{i=1,\ldots,N} \in \mathbb{R}^{2N}$$

- Bunch acquires and transports information from elements actions (messenger!)
- Elements imprint their action on bunch.

The PyHEADTAIL Model

Betatron motion

- Bunch acquires and transports information from elements actions (messenger!)
- Elements imprint their action on bunch.

Chromaticity: coupling to longitudinal

Detuning with amplitude: continuous detuning

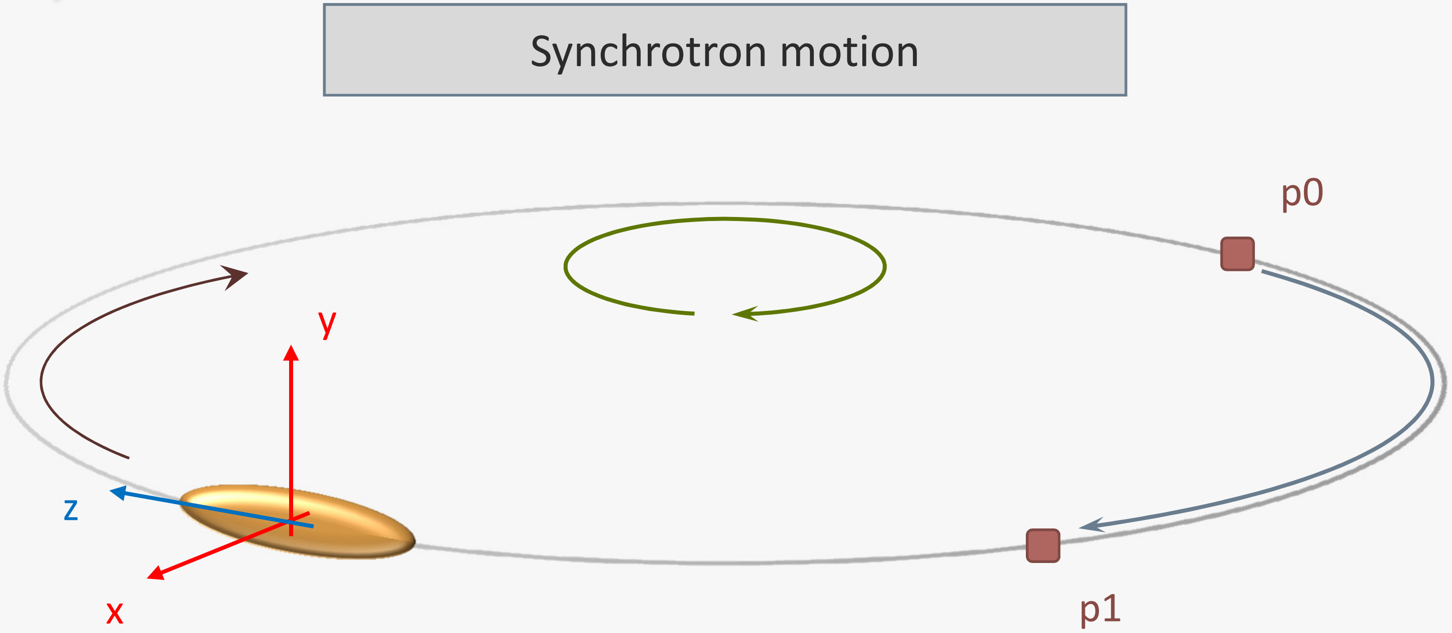
$$\Delta\mu_i \sim \Delta\mu_0 + \left(\xi\,\delta_i + \alpha_{xx}\,J_{x,\,i} + \alpha_{xy}\,J_{y,\,i}\right)\frac{\Delta\mu_0}{2\pi Q}$$

$$\mathcal{M}_i = \begin{pmatrix} \sqrt{\beta_1} & 0 \\ -\frac{\alpha_1}{\sqrt{\beta_1}} & \frac{1}{\sqrt{\beta_1}} \end{pmatrix} \begin{pmatrix} \cos(\Delta\mu_i) & \sin(\Delta\mu_i) \\ -\sin(\Delta\mu_i) & \cos(\Delta\mu_i) \end{pmatrix} \begin{pmatrix} \frac{1}{\sqrt{\beta_0}} & 0 \\ \frac{\alpha_0}{\sqrt{\beta_0}} & \sqrt{\beta_0} \end{pmatrix}$$

$$\left.\begin{pmatrix} x_i \\ x'_i \end{pmatrix}\right|_1 = \mathcal{M}_i \left.\begin{pmatrix} x_i \\ x'_i \end{pmatrix}\right|_0$$

$$i = 1, \ldots, N$$

The PyHEADTAIL Model



$$z_{i,k+1/2} = z_{i,k} - \frac{\eta C}{2}\,\delta_{i,k}$$

$$\delta_{i,k+1} = \delta_{i,k} + \frac{e\,V_{\mathrm{RF}}}{m\gamma\beta^2 c^2}\sin\left(\frac{2\pi h}{C}\,z_{i,k+1/2}\right)$$

$$z_{i,k+1} = z_{i,k+1/2} - \frac{\eta C}{2}\,\delta_{i,k+1}$$

$$i = 1,\ldots,N$$

k: iteration/turn

- Bunch acquires and transports information from elements actions (messenger!)
- Elements imprint their action on bunch.

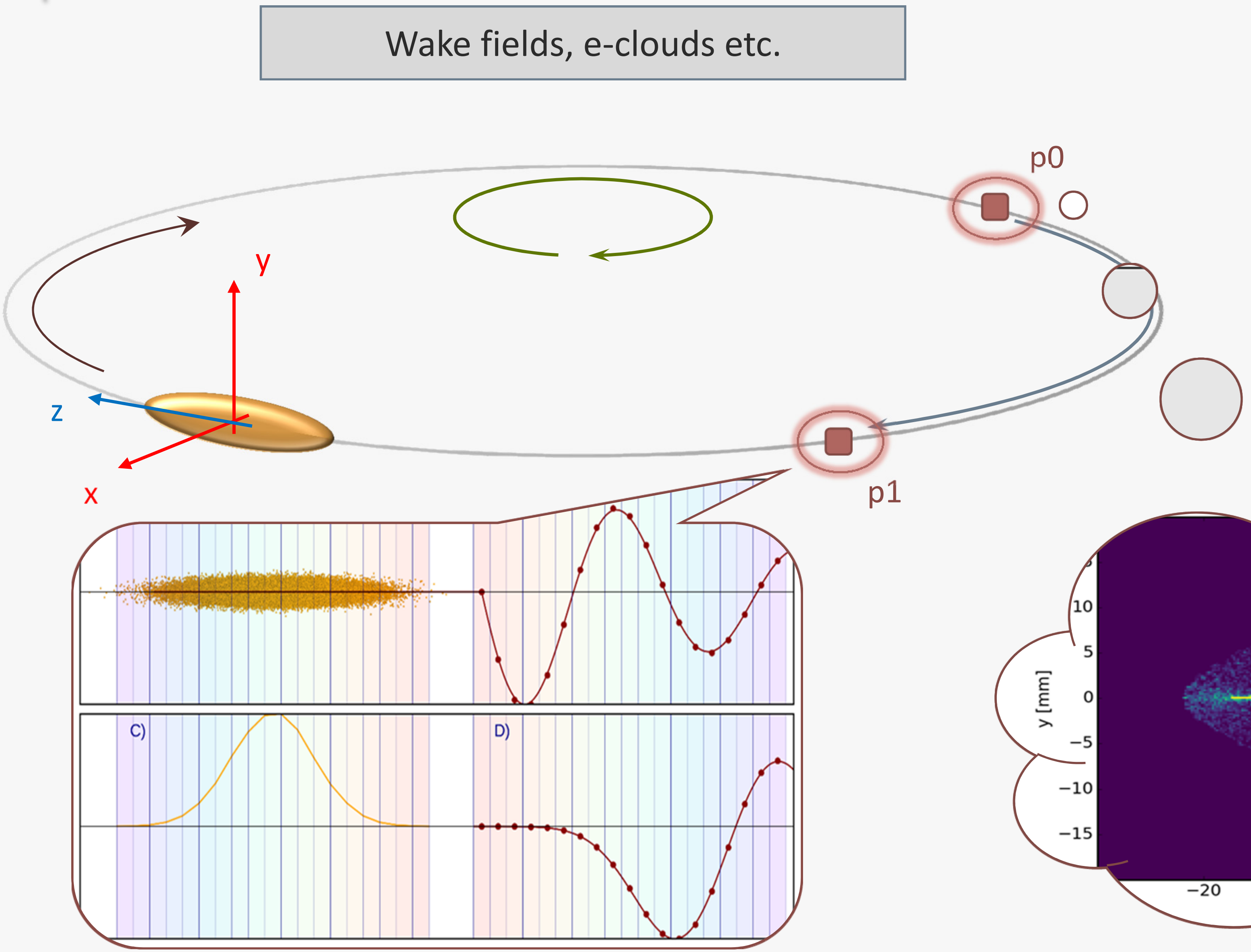
$$z' = -\eta\,\delta$$

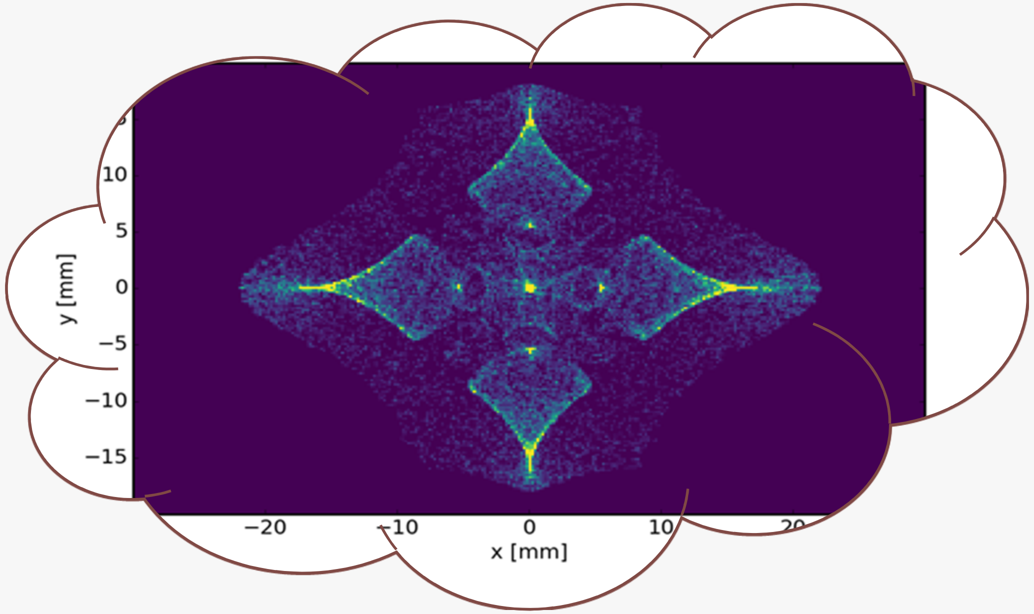
$$\delta' = \frac{e\,V_{\mathrm{RF}}}{m\gamma\beta^2 c^2\,C}\sin\left(\frac{2\pi h}{C}\,z\right)$$

- V_{RF}: RF voltage
- $h = \frac{\omega_{\mathrm{RF}}}{\omega_0}$: harmonic number
- ω_0: Revolution frequency
- C: circumference

The PyHEADTAIL Model

- Bunch acquires and transports information from elements actions (messenger!)
- Elements imprint their action on bunch.





Context & Outline

Summary:

Some guidelines on programming strategies with their advantages and limitations.

Outline:

1. Introduction
2. Basic model of the accelerator-beam system
3. Modern approaches and program architectures
4. Performance considerations
5. Applications, present status and perspectives

Modeling collective effects

- What are the demands of modern computer codes?

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

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

Dynamic
- Fast
- Flexible
- Interactive

… and each individual module becomes a carefully engineered piece of software.

Simple

Be minimalist

- keep number of lines to the minimum necessary

Be pragmatic

- use available libraries and don‘t re-invent the wheel

Be paranoid

- write clear and well-formatted code

“Always code as if the guy who ends up maintaining your code will be a violent psychopath who knows where you live”

John Woods

Modular

Repository (PyHEADTAIL)

- ...
- feedback
- gpu
- impedances
- machines
- monitors
- multipoles
- particles
- radiation
- spacecharge
- testing
- trackers
- ...

Modular

Repository (PyHEADTAIL)

- ...
- feedback
- gpu
- impedances
  - impedances/wake_kicks.py
  - impedances/wakes.py
- machines
- monitors
- multipoles
- particles
  - particles/generators.py
  - particles/particles.py
  - particles/...
- radiation
- spacecharge
- testing
- trackers
  - trackers/longitudinal_tracking.py
  - trackers/transverse_tracking.py
  - trackers/...
- ...

Modular

Repository (PyHEADTAIL)

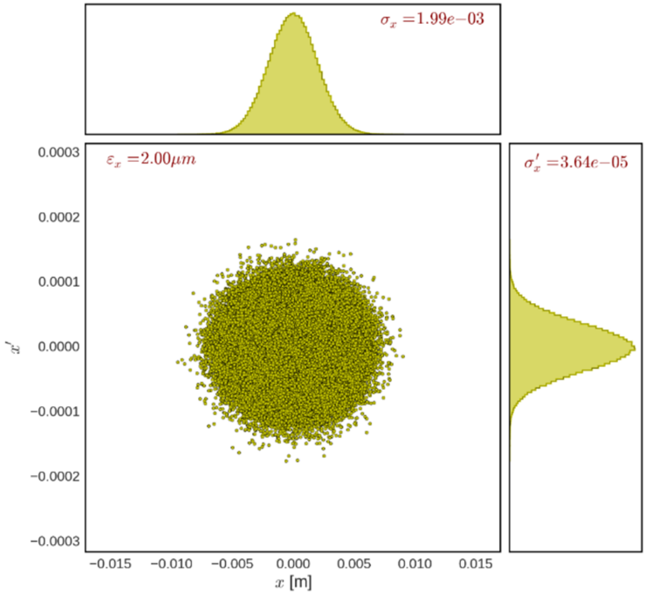
... | feedback | g... | impedances | machines | monitors | multipoles

particles | radiation | ...esting | trackers

- particles/generators.py
- particles/particles.py
- particles/...

- impedances/wake_kicks.py
- impedances/wakes.py

- trackers/longitudinal_tracking.py
- trackers/transverse_tracking.py
- trackers/...



$$\varepsilon_\perp = \beta\gamma\sqrt{\langle x^2\rangle\langle x'^2\rangle - \langle xx'\rangle^2}$$
$$= \beta\gamma\sigma_x\sigma_{x'}$$
$$\varepsilon_\parallel = 4\pi\sigma_z\sigma_\delta\frac{p_0}{e}$$

Modular

Repository (PyHEADTAIL)

...
feedback
impedances
machines
monitors
multipoles

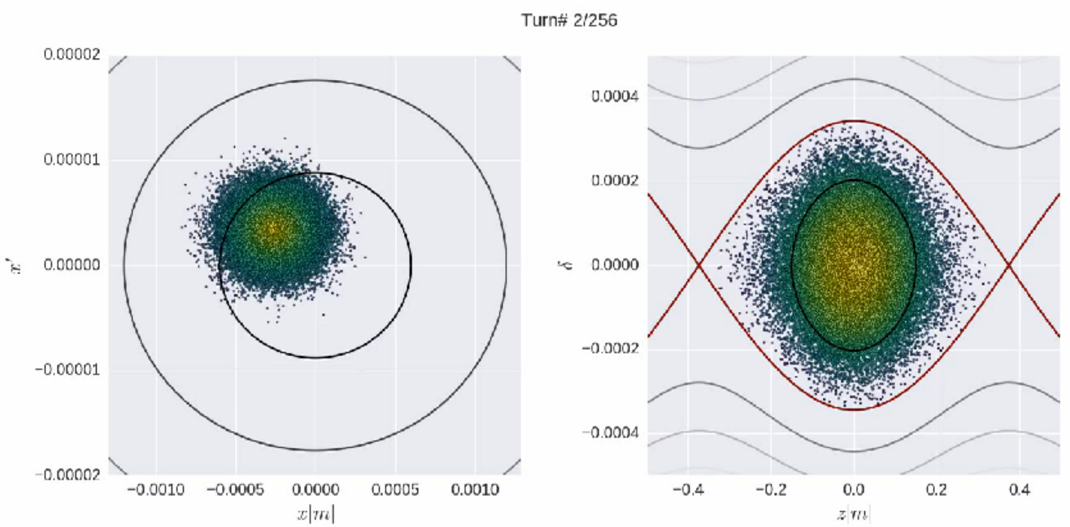
particles
radiation
testing
trackers

particles/generators.py
particles/particles.py
particles/...

impedances/wake_kicks.py
impedances/wakes.py

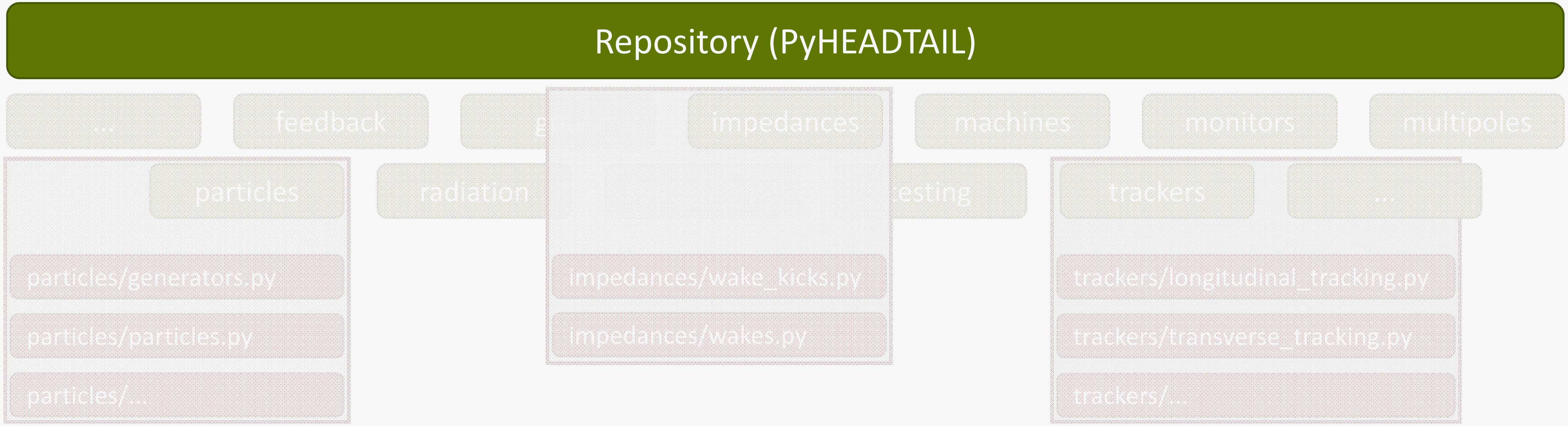
trackers/longitudinal_tracking.py
trackers/transverse_tracking.py
trackers/...

Transverse



Longitudinal

Modular



"Just as Code Style, API Design, and Automation are essential for a healthy development cycle, Repository structure is a crucial part of your project's architecture."

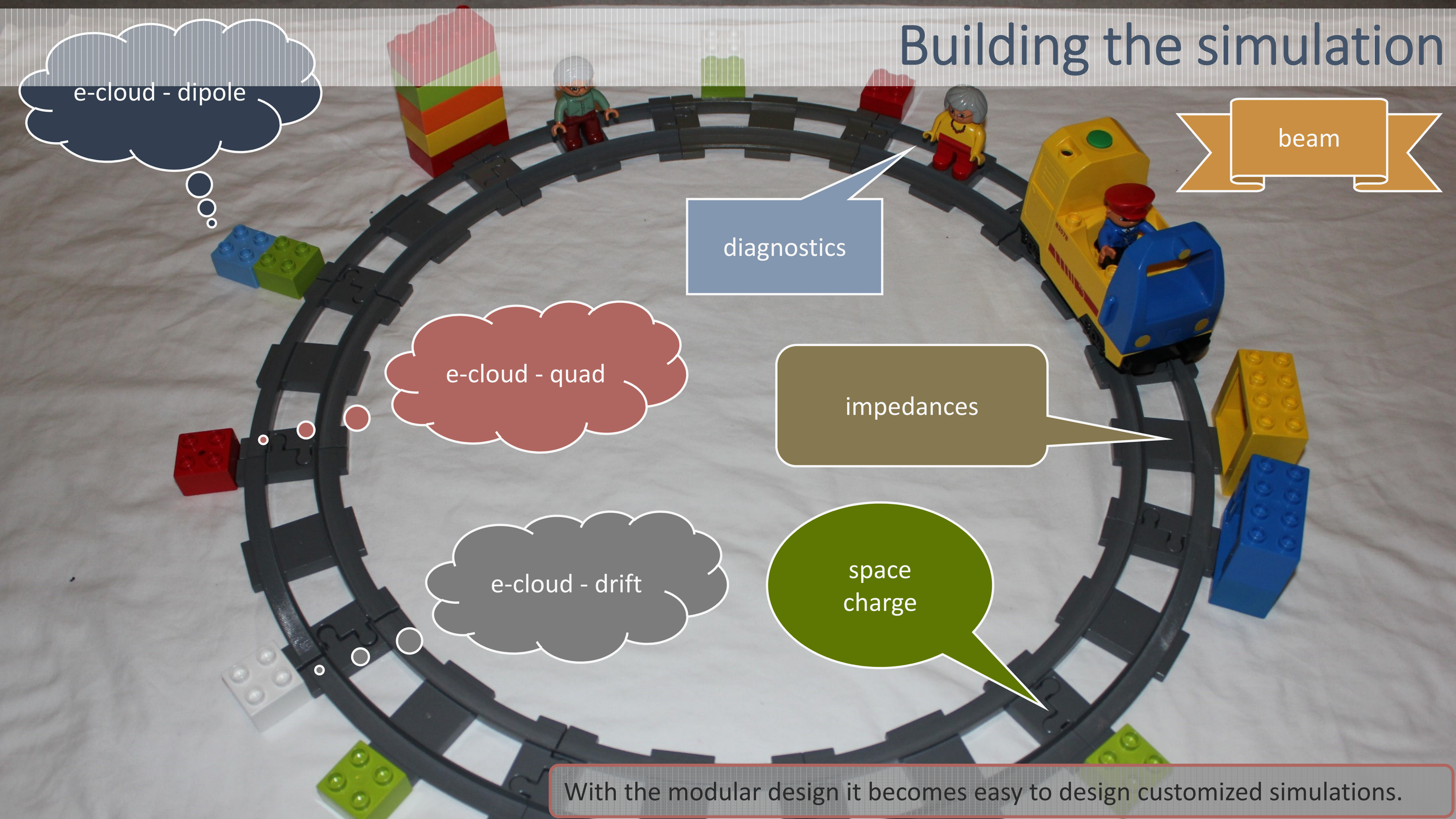
The Hitchhiker's Guide to Python

A good repository structure forms the backbone of a project. Object-oriented programming adds to this a defined programming structure.

Building the simulation

With the modular design it becomes easy to design customized simulations.

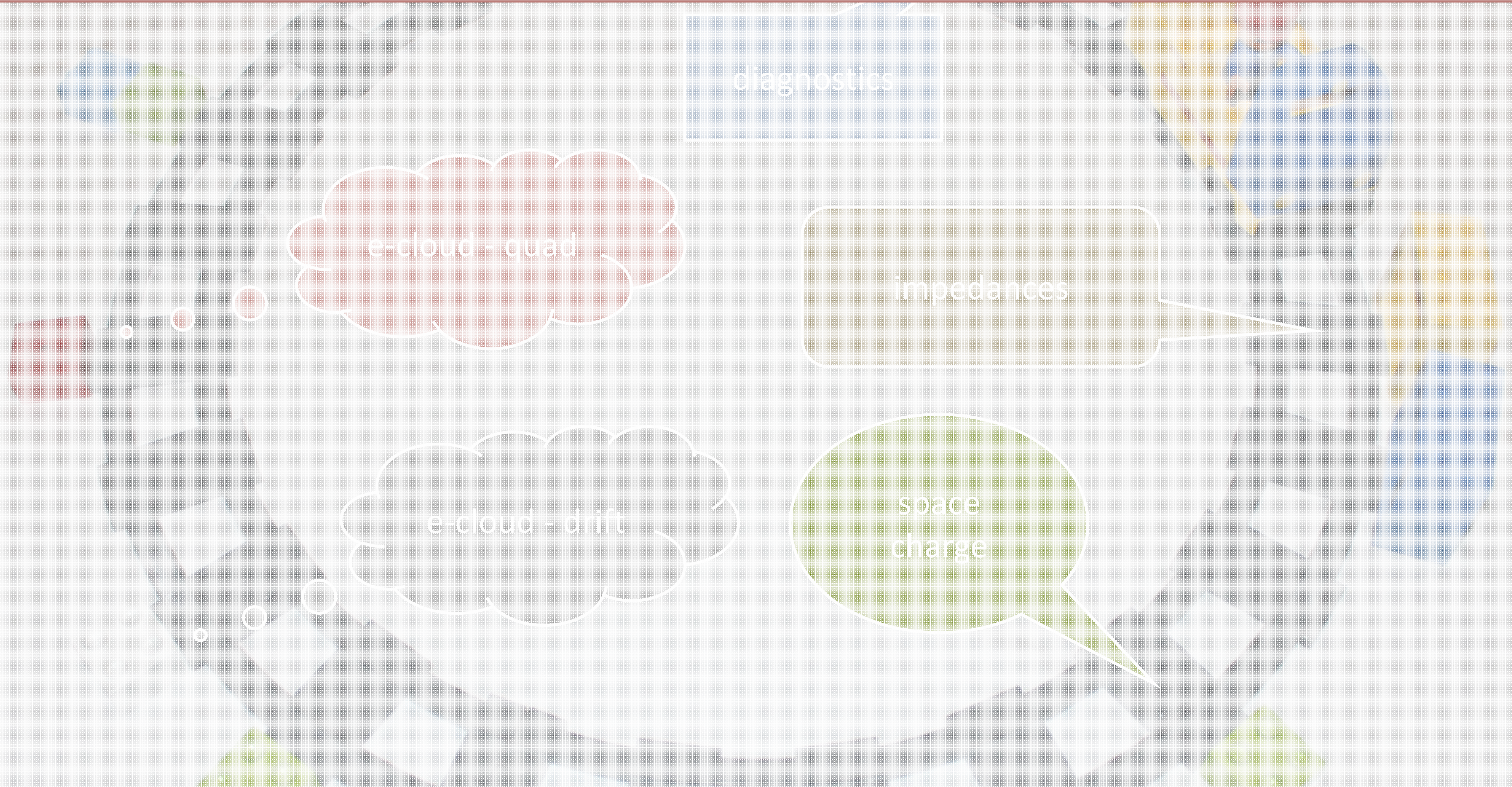
Building the simulation



With the modular design it becomes easy to design customized simulations.

Building the simulation

We need to think of a clever inputfile language that reflects this modularity (example of MAD-X, MAFIA etc.).



Building the simulation

~~We need to think of a clever inputfile language that reflects this modularity (example of MAD-X, MAFIA etc.).~~

Why invent yet another language/syntax when there are established and popular languages most programmers already know:

"It was nice to learn Python;
a nice afternoon."

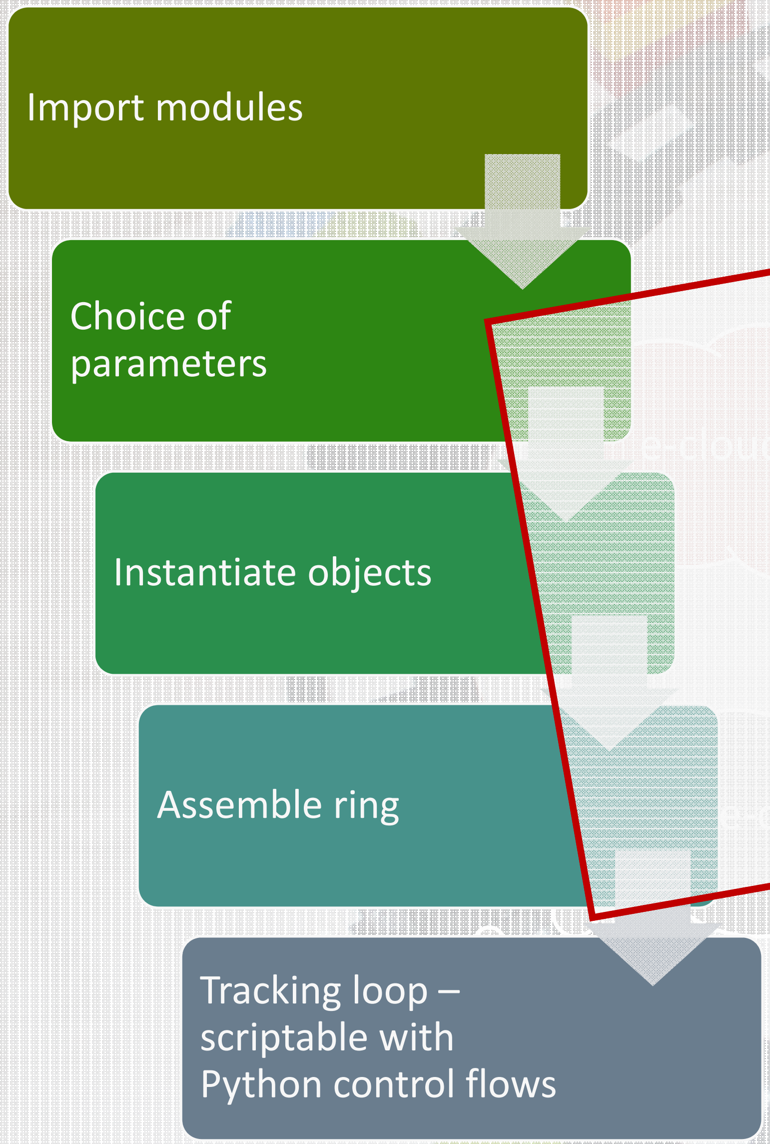
Donald Knuth

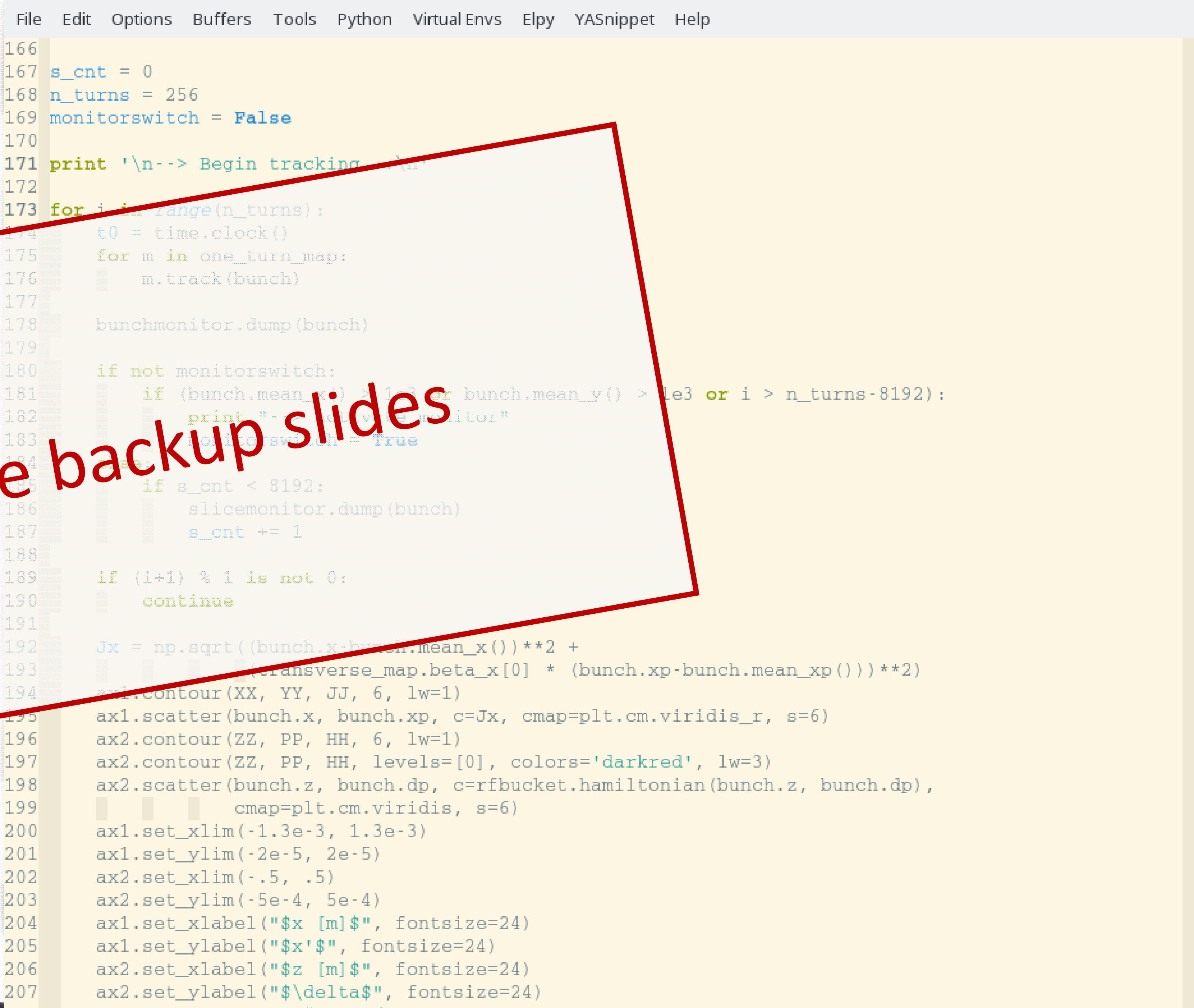
Input file becomes simply **a Python script** → inject the full power of the Python programming language to the input file

- Control flows
- Third party libraries
- Data analysis and visualization
- ...

Setting up a simulation becomes writing a small program/script – simple, highly customizable, interactive

Building the simulation





Summary

Summary:

Some ideas and examples how to improve performance limitations encountered when working with interpreted languages (Python).

Outline:

1. Introduction
2. Basic model of the accelerator-beam system
3. Modern approaches and program architectures
4. Performance considerations
5. Applications, present status and perspectives

Performance considerations

An important hitch remains on the **FAST** requirement:

"Python was not made to be fast… … but to make developers fast."

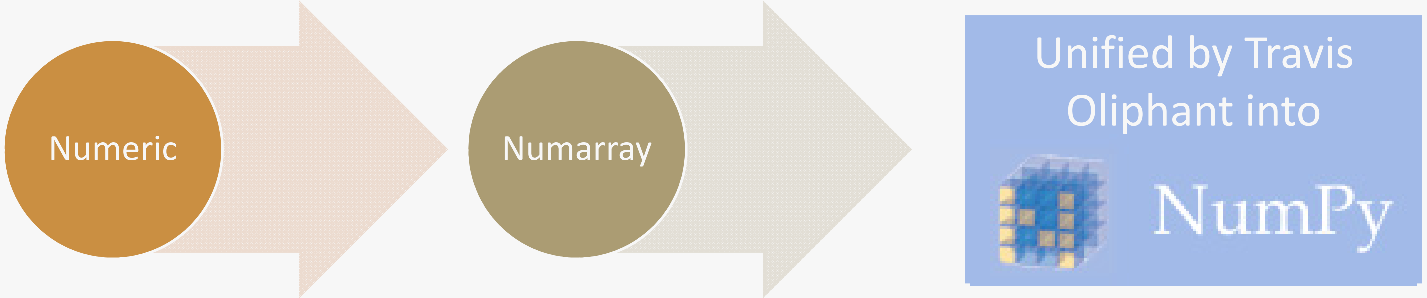
Writing faster Python,
Sebastian Witowski

Being an interpreted language, strongly but dynamically typed, **Python can be slow in execution**! This needs to be handled.

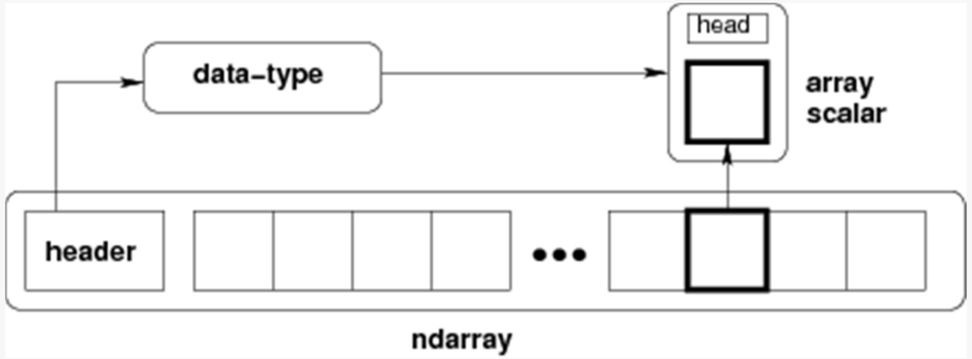
Performance considerations

Python used as it is usually is slow for number crunching!

In the late 90s the Python scientific community worked on bringing the computationally intensive parts to a lower level:



The core functionality of NumPy is its "ndarray", for *n*-dimensional array, data structure.



Today, NumPy is the unrivaled basis for nearly all scientific tools in Python.

A very simple example for speed-up

- Profile the evaluation of the sine function over an array
- x is an array with 6M entries ranging uniformly from 0 to 2 pi

PYHEADTAIL notebook

012b_optimisation_example Last Checkpoint: 4 hou

File Edit View Insert Cell Kernel Help

Profile

```
%timeit pysin(x)
%timeit npsin(x)
%timeit cmsin(x, a)
```

```
1 loop, best of 3: 171 ms per loop
```

PYHEADTAIL notebook

012b_optimisation_example Last Checkpoint: 4 minutes ago (autosaved)

File Edit View Insert Cell Kernel Help

Python sin

for loop in Python – never do this!

```
In [4]: def pysin(x):
            y = [math.sin(i) for i in x]
            return y
```

A very simple example for speed-up

- Profile the evaluation of the sine function over an array
- x is an array with 6M entries ranging uniformly from 0 to 2 pi

PYHEADTAIL notebook

012b_optimisation_example Last Checkpoint: 4 hou

File Edit View Insert Cell Kernel Help

Profile

```
%timeit pysin(x)
%timeit npsin(x)
%timeit cmsin(x, a)
```

```
1 loop, best of 3: 171 ms per loop
10 loops, best of 3: 26.4 ms per loop
```

PYHEADTAIL notebook

012b_optimisation_example Last Checkpoint: 4 minutes ago (autosaved)

File Edit View Insert Cell Kernel Help

Python sin

In [4]:
```
def pysin(x):
    y = [math.sin(i) for i in x]
    return y
```

Numpy sin

In [5]:
```
import numpy as np

def npsin(x):
    return np.sin(x)
```

Vectorization: the array is interpreted by NumPy and the loop is executed at C level

A very simple example for speed-up

- Profile the evaluation of the sine function over an array
- x is an array with 6M entries ranging uniformly from 0 to 2 pi

PYHEADTAIL notebook

012b_optimisation_example Last Checkpoint: 4 hou

File Edit View Insert Cell Kernel Help

Profile

```
%timeit pysin(x)
%timeit npsin(x)
%timeit cmsin(x, a)
```

```
1 loop, best of 3: 171 ms per loop
10 loops, best of 3: 26.4 ms per loop
10 loops, best of 3: 26.8 ms per loop
```

PYHEADTAIL notebook

012b_optimisation_example Last Checkpoint: 11 minutes ago (unsaved cha

File Edit View Insert Cell Kernel Help

Generated by Cython 0.23.4

Yellow lines hint at Python interaction.
Click on a line that starts with a "+" to see the C code that Cython genera

```
 01:
+02: import numpy as np
 03: cimport numpy as np
 04: cimport cython
 05: from cython.parallel import prange
 06: from libc.math cimport sin
 07:
 08: @cython.boundscheck(False)
 09: @cython.wraparound(False)
 10: @cython.cdivision(False)
+11: def cmsin(double[::1] x, double[::1] s):
 12:
+13:     cdef int n = x.shape[0]
 14:     cdef int i
+15:     for i in prange(n, nogil=True, num_threads=4):
+16:         s[i] = sin(x[i])
```

Cython

Small low-level extensions for Python accessing native Python data structures – core coding in Python

Typed Memoryviews: efficient data access and handling

Explicit loop at C level; yellow lines indicate Python overhead

A very simple example for speed-up

- Profile the evaluation of the sine function over an array
- x is an array with 6M entries ranging uniformly from 0 to 2 pi

PYHEADTAIL notebook

012b_optimisation_example Last Checkpoint: 4 hou

File Edit View Insert Cell Kernel Help

Profile

```
%timeit pysin(x)
%timeit npsin(x)
%timeit cmsin(x, a)
```

```
1 loop, best of 3: 171 ms per loop
10 loops, best of 3: 26.4 ms per loop
10 loops, best of 3: 26.8 ms per loop
```

```
%timeit vdt_sin(x, a)
%timeit vdt_sinv(x, a)
%timeit vdt_sincos(x, a, b)
```

```
100 loops, best of 3: 11 ms per loop
100 loops, best of 3: 8.23 ms per loop
100 loops, best of 3: 17.2 ms per loop
```

PYHEADTAIL notebook

012b_optimisation_example Last Checkpoint: 8 minutes ago (autosaved)

File Edit View Insert Cell Kernel Help

Generated by Cython 0.23.4

Yellow lines hint at Python interaction.
Click on a line that starts with a "+" to see the C code that Cyth

```
#export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/home/Kli/wo
#cython boundscheck=False

cimport cython
import numpy as np
cimport numpy as np
from cython.parallel import prange

cdef extern from "sin.h" namespace "vdt":
    cdef double fast_sin(double x) nogil
    cdef void fast_sinv(int n, double *x, double *s) nogil

cdef extern from "sincos.h" namespace "vdt":
    cdef void fast_sincos(double x, double &s, double &c) nogil

@cython.boundscheck(False)
def vdt_sin(double[::1] x, double[::1] s):

    cdef int n = x.shape[0]
    cdef int i
    for i in prange(n, nogil=True, num_threads=1):
        s[i] = fast_sin(x[i])

@cython.boundscheck(False)
def vdt_sincos(double[::1] x, double[::1] s, double[::1] c):

    cdef int n = x.shape[0]
    cdef int i
    for i in prange(n, nogil=True, num_threads=1):
        fast_sincos(x[i], s[i], c[i])

@cython.boundscheck(False)
def vdt_sinv(double[::1] x, double[::1] s):

    cdef int n = x.shape[0]
    fast_sinv(n, &x[0], &s[0])
```

Cython

Small low-level extensions for Python accessing native Python data structures – core coding in Python

Import from an external library and use it in Cython

(here, vdtmath:
D. Piparo et al. 2014 J. Phys.: Conf. Ser. 513 052027 "Speeding up HEP experiment software with a library of fast and auto-vectorisable mathematical functions")

A very simple example for speed-up

- Profile the evaluation of the sine function over an array
- x is an array with 6M entries ranging uniformly from 0 to 2 pi

PYHEADTAIL notebook

012b_optimisation_example Last Checkpoint: 4 hou

File Edit View Insert Cell Kernel Help

Profile

```
%timeit pysin(x)
%timeit npsin(x)
%timeit cmsin(x, a)
```

```
1 loop, best of 3: 171 ms per loop
10 loops, best of 3: 26.4 ms per loop
10 loops, best of 3: 26.8 ms per loop
```

```
%timeit vdt_sin(x, a)
%timeit vdt_sinv(x, a)
%timeit vdt_sincos(x, a, b)
```

```
100 loops, best of 3: 11 ms per loop
100 loops, best of 3: 8.23 ms per loop
100 loops, best of 3: 17.2 ms per loop
```

PYHEADTAIL notebook

012b_optimisation_example Last Checkpoint: 8 minutes ago (autosaved)

File Edit View Insert Cell Kernel Help

Generated by Cython 0.23.4

Yellow lines hint at Python interaction.
Click on a line that starts with a "+" to see the C code that Cyth

```
#export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/hom
#cython boundscheck=False

cimport cython
import numpy as np
cimport numpy as np
from cython.parallel import prange

cdef extern from "sin.h" namespace "vdt":
    cdef double fast_sin(double x) nogil
    cdef void fast_sinv(int n, double *x, double *s) nogil

cdef extern from "sincos.h" namespace "vdt":
    cdef void fast_sincos(double x, double &s,

@cython.boundscheck(False)
def vdt_sin(double[::1] x, double[::1] s):

    cdef int n = x.shape[0]
    cdef int i
    for i in prange(n, nogil=True, num_threads=1):
        s[i] = fast_sin(x[i])

@cython.boundscheck(False)
def vdt_sincos(double[::1] x, double[::1] s, double[::1] c):

    cdef int n = x.shape[0]
    cdef int i
    for i in prange(n, nogil=True, num_threads=1):
        fast_sincos(x[i], s[i], c[i])

@cython.boundscheck(False)
def vdt_sinv(double[::1] x, double[::1] s):

    cdef int n = x.shape[0]
    fast_sinv(n, &x[0], &s[0])
```

Cython

Small low-level extensions for Python accessing native Python data structures – core coding in Python

Import from an external library and use it in Cython

(here, vdtmath:
D. Piparo et al. 2014 J. Phys.: Conf. Ser. 513 052027 "Speeding up HEP experiment software with a library of fast and auto-vectorisable mathematical functions")

OpenMP multithreading parallelization; move from e.g. 1 to 4 threads

A very simple example for speed-up

- Profile the evaluation of the sine function over an array
- x is an array with 6M entries ranging uniformly from 0 to 2 pi

PYHEADTAIL notebook

012b_optimisation_example Last Checkpoint: 4 hou

File Edit View Insert Cell Kernel Help

Profile

```
%timeit pysin(x)
%timeit npsin(x)
%timeit cmsin(x, a)
```

```
1 loop, best of 3: 176 ms per loop
10 loops, best of 3: 26.4 ms per loop
100 loops, best of 3: 13 ms per loop
```

```
%timeit vdt_sin(x, a)
%timeit vdt_sinv(x, a)
%timeit vdt_sincos(x, a, b)
```

```
100 loops, best of 3: 6.12 ms per loop
100 loops, best of 3: 8.19 ms per loop
100 loops, best of 3: 8.26 ms per loop
```

PYHEADTAIL notebook

012b_optimisation_example Last Checkpoint: 8 minutes ago (autosaved)

File Edit View Insert Cell Kernel Help

Generated by Cython 0.23.4

Yellow lines hint at Python interaction.
Click on a line that starts with a "+" to see the C code that Cyth

```
#export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/home/Kli/wo
#cython boundscheck=False

cimport cython
import numpy as np
cimport numpy as np
from cython.parallel import prange

cdef extern from "sin.h" namespace "vdt":
    cdef double fast_sin(double x) nogil
    cdef void fast_sinv(int n, double *x, double *s) nogil

cdef extern from "sincos.h" namespace "vdt":
    cdef void fast_sincos(double x, double &s

@cython.boundscheck(False)
def vdt_sin(double[::1] x, double[::1] s):

    cdef int n = x.shape[0]
    cdef int i
    for i in prange(n, nogil=True, num_threads=1):
        s[i] = fast_sin(x[i])

@cython.boundscheck(False)
def vdt_sincos(double[::1] x, double[::1] s, double[::1] c):

    cdef int n = x.shape[0]
    cdef int i
    for i in prange(n, nogil=True, num_threads=1):
        fast_sincos(x[i], s[i], c[i])

@cython.boundscheck(False)
def vdt_sinv(double[::1] x, double[::1] s):

    cdef int n = x.shape[0]
    fast_sinv(n, &x[0], &s[0])
```

Cython

Small low-level extensions for Python accessing native Python data structures – core coding in Python

Import from an external library and use it in Cython

(here, vdtmath:
D. Piparo et al. 2014 J. Phys.: Conf. Ser. 513 052027 "Speeding up HEP experiment software with a library of fast and auto-vectorisable mathematical functions")

OpenMP multithreading parallelization; move from e.g. 1 to 4 threads

4

A very simple example for speed-up

- Profile the evaluation of the sine

There are an abundance of tools out there for embedding and extending Python:

CPython API, ctypes, f2py, SWIG, Boost::Python, Numexpr, Cython, PyCUDA, Numba, cffi…

Today, Cython provides a very most modern and flexible approach. However the world is changing fast. Writing simple (and little) code will enable us to remain adaptable.

PYHEADTAIL notebook

Cython

…nsions for … …rary and …

… 513 052027 … th a library of … unctions")

…lelization; move from e.g. 1 to 4 threads

```
%timeit pysin(x)
%timeit npsin(x)
%timeit cmsin(x, a)

1 loop, best of 3: 176 ms per loop
10 loops, best of 3: 26.4 ms per loop
100 loops, best of 3: 13 ms per loop
```

```
%timeit vdt_sin(x, a)
%timeit vdt_sinv(x, a)
%timeit vdt_sincos(x, a, b)

100 loops, best of 3: 6.12 ms per loop
100 loops, best of 3: 8.19 ms per loop
100 loops, best of 3: 8.26 ms per loop
```

```
def vdt_sin(double[::1] x, double[::1] s):

    cdef int n = x.shape[0]
    cdef int i
    for i in prange(n, nogil=True, num_threads=1):
        s[i] = fast_sin(x[i])

@cython.boundscheck(False)
def vdt_sincos(double[::1] x, double[::1] s, double[::1] c):

    cdef int n = x.shape[0]
    cdef int i
    for i in prange(n, nogil=True, num_threads=1):
        fast_sincos(x[i], s[i], c[i])

@cython.boundscheck(False)
def vdt_sinv(double[::1] x, double[::1] s):

    cdef int n = x.shape[0]
    fast_sinv(n, &x[0], &s[0])
```

4

A very simple example for speed-up

- Profile the evaluation of the sine

PYHEADTAIL notebook

Cython

nsions for

brary and

. 513 052027
th a library of
unctions")

lelization;
move from e.g. 1 to 4 threads

There are an abundance of tools out there for embedding and extending Python:

CPython API, ctypes, f2py, SWIG, Boost::Python, Numexpr, Cython, PyCUDA, Numba, cffi…

Today, Cython provides a very most modern and flexible approach. However the world is changing fast. Writing simple (and little) code will enable us to remain adaptable.

```
%timeit pysin(x)
%timeit npsin(x)
%timeit cmsin(x,
```

```
1 loop, best of
10 loops, best o
100 loops, best
```

```
%timeit vdt_sin(
%timeit vdt_sinv
%timeit vdt_sinc
```

```
100 loops, best
100 loops, best
100 loops, best of 3: 8.26 ms per loop
```

```
+17: def vdt_sin(double[::1] x, double[::1] s):
```

The next Travis Oliphant:

“Yep. Theano, Numba, numexpr, Cython and PyPy shall one day all merge to form Numtron, defender of sanity, runner of fast numerics.”

Dave Warde-Farley

```
 32: @cython.boundscheck(False)
+33: def vdt_sinv(double[::1] x, double[::1] s):
 34:
+35:     cdef int n = x.shape[0]
+36:     fast_sinv(n, &x[0], &s[0])
```

Summary:

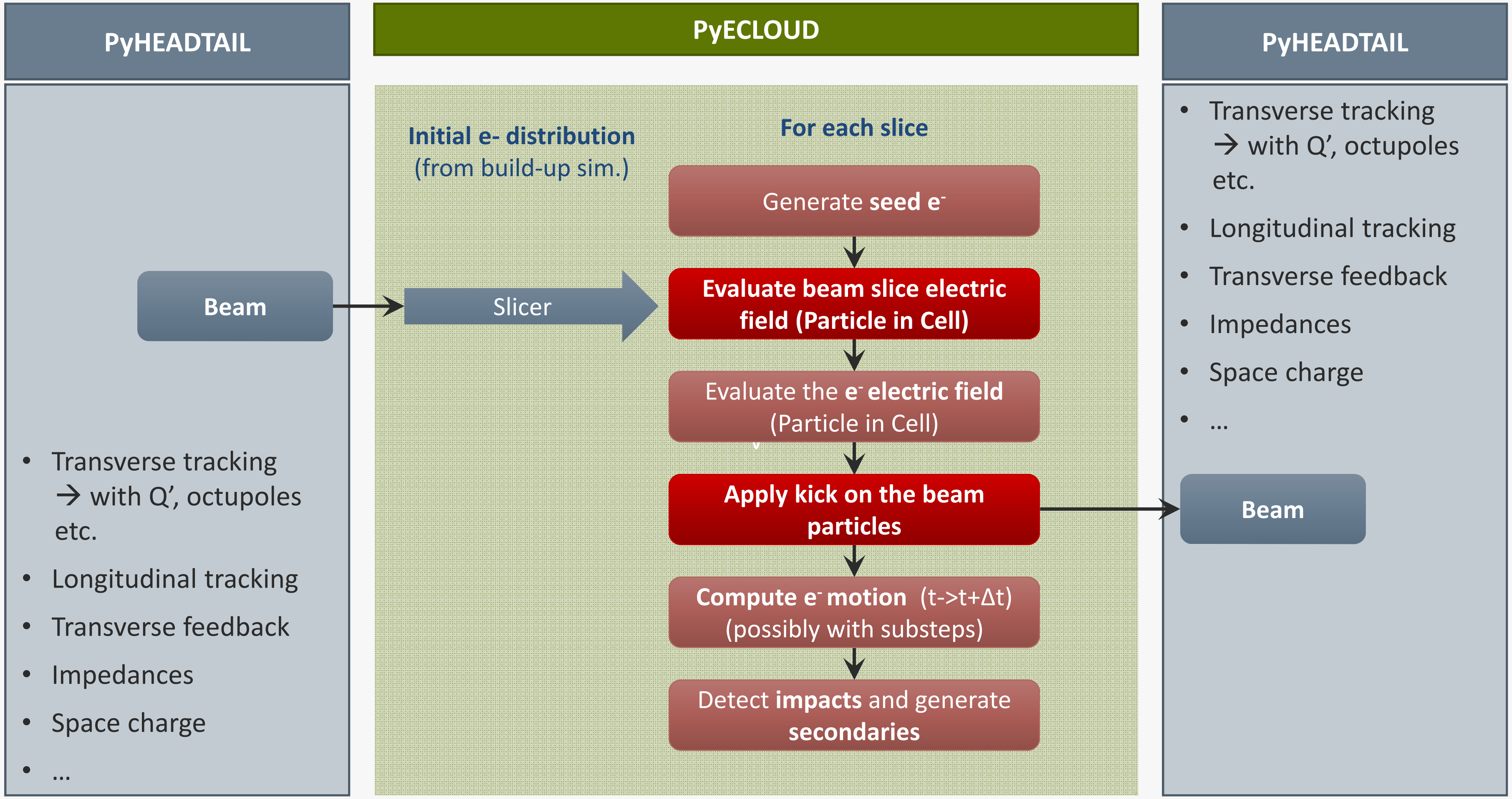
Illustration of a few examples which exploit the architecture and design promoted earlier.

Outline:

1. Introduction
2. Basic model of the accelerator-beam system
3. Modern approaches and program architectures
4. Performance considerations
5. Applications, present status and perspectives

Interfacing with PyPIC and PyELCOUD

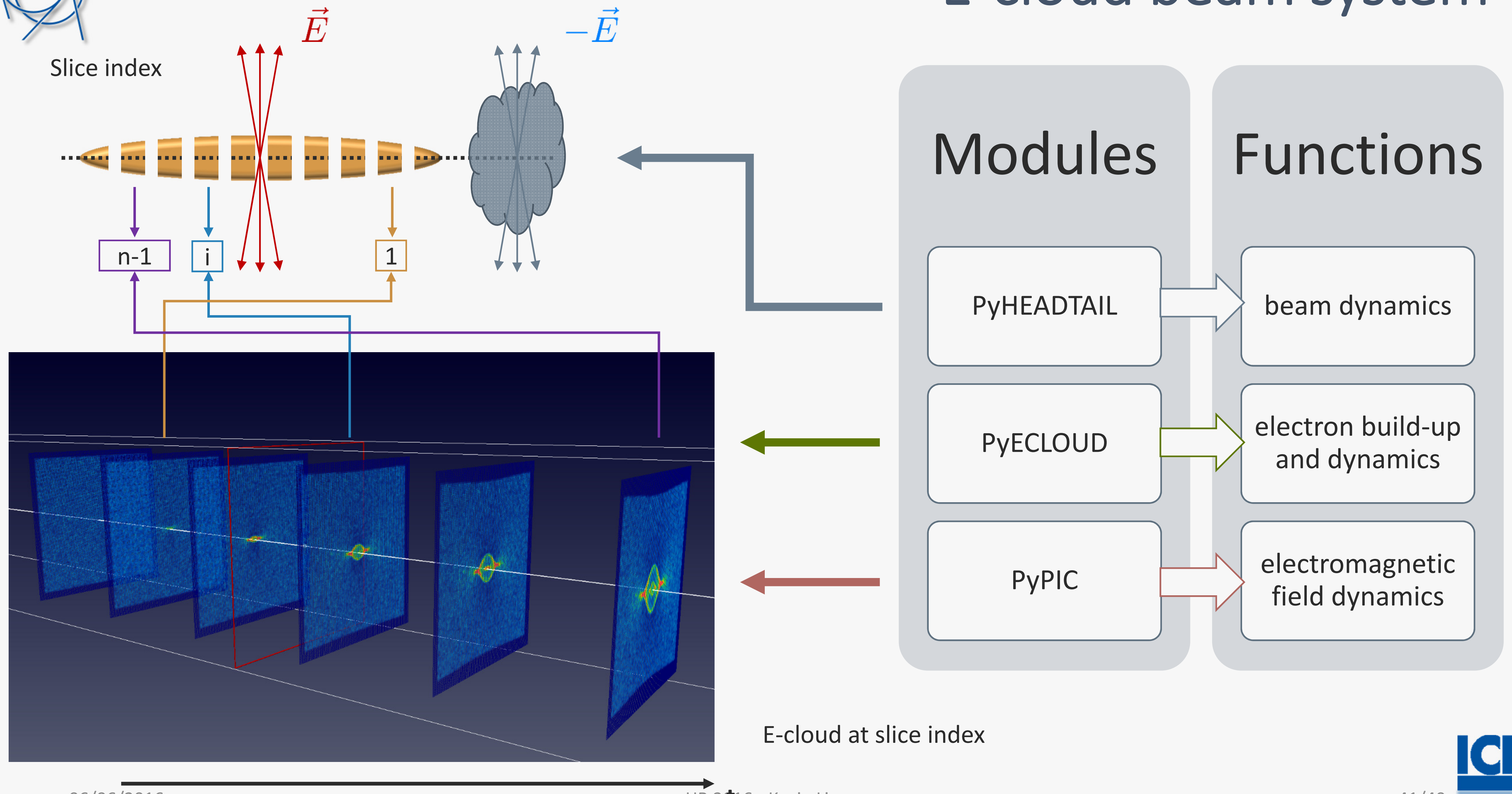
- A self-consistent treatment requires the combination of an instability and a build-up code
- Becomes easily possible with modular structure and good design of codes (e.g. object orientation)



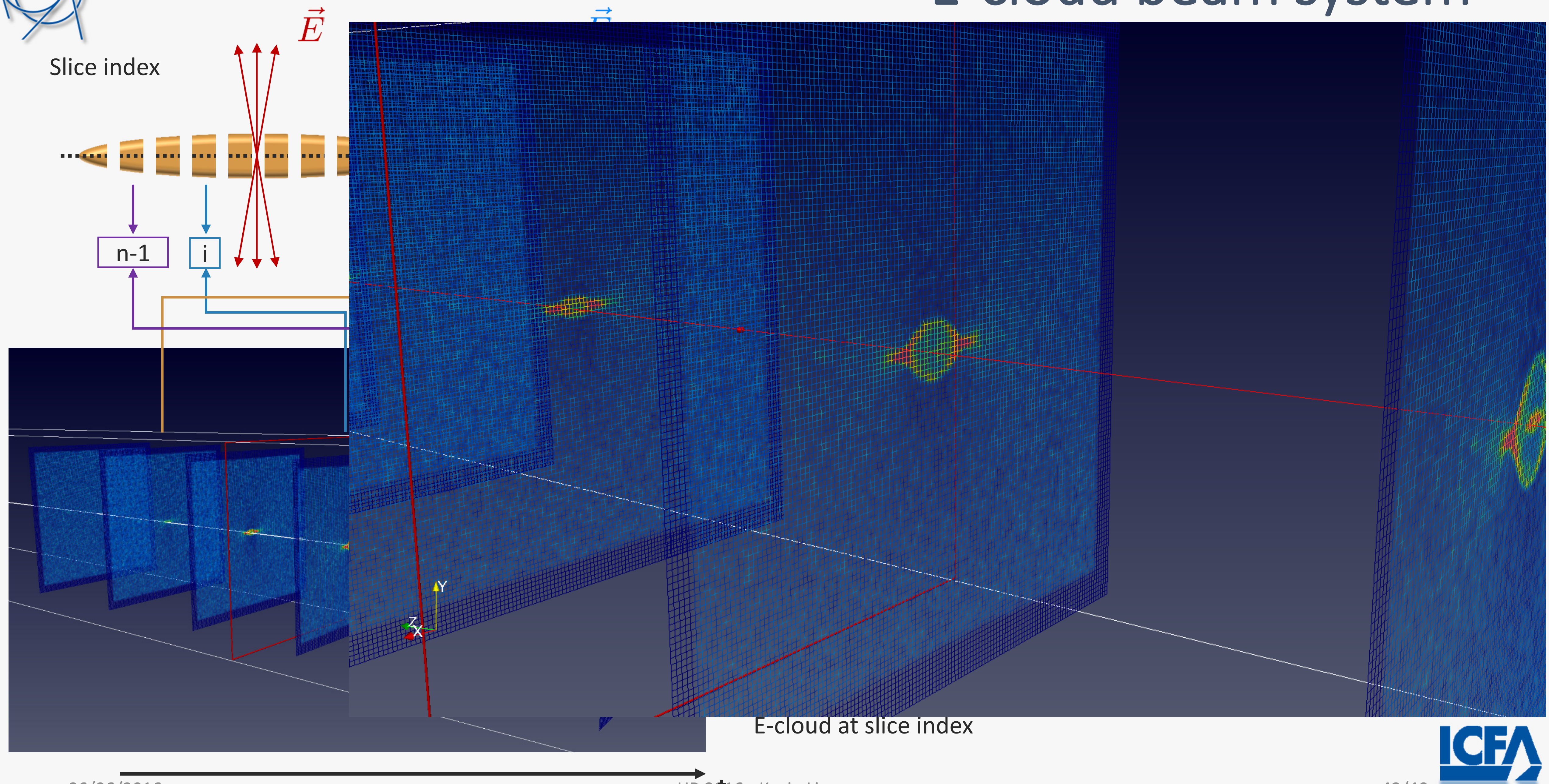
Courtesy Giovanni Iadarola

Legend: From instability code – From build-up code – Interaction between the two codes

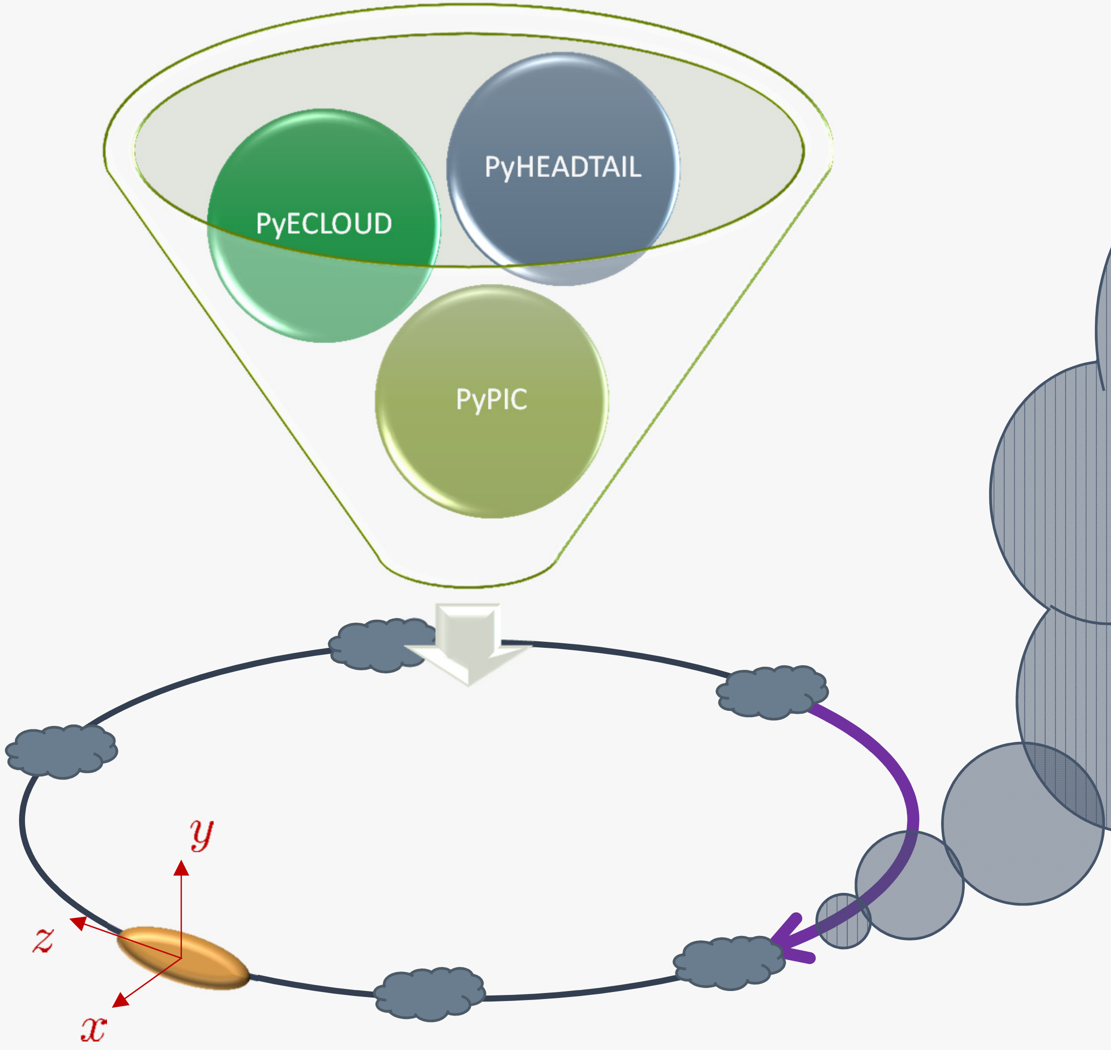
E-cloud beam system



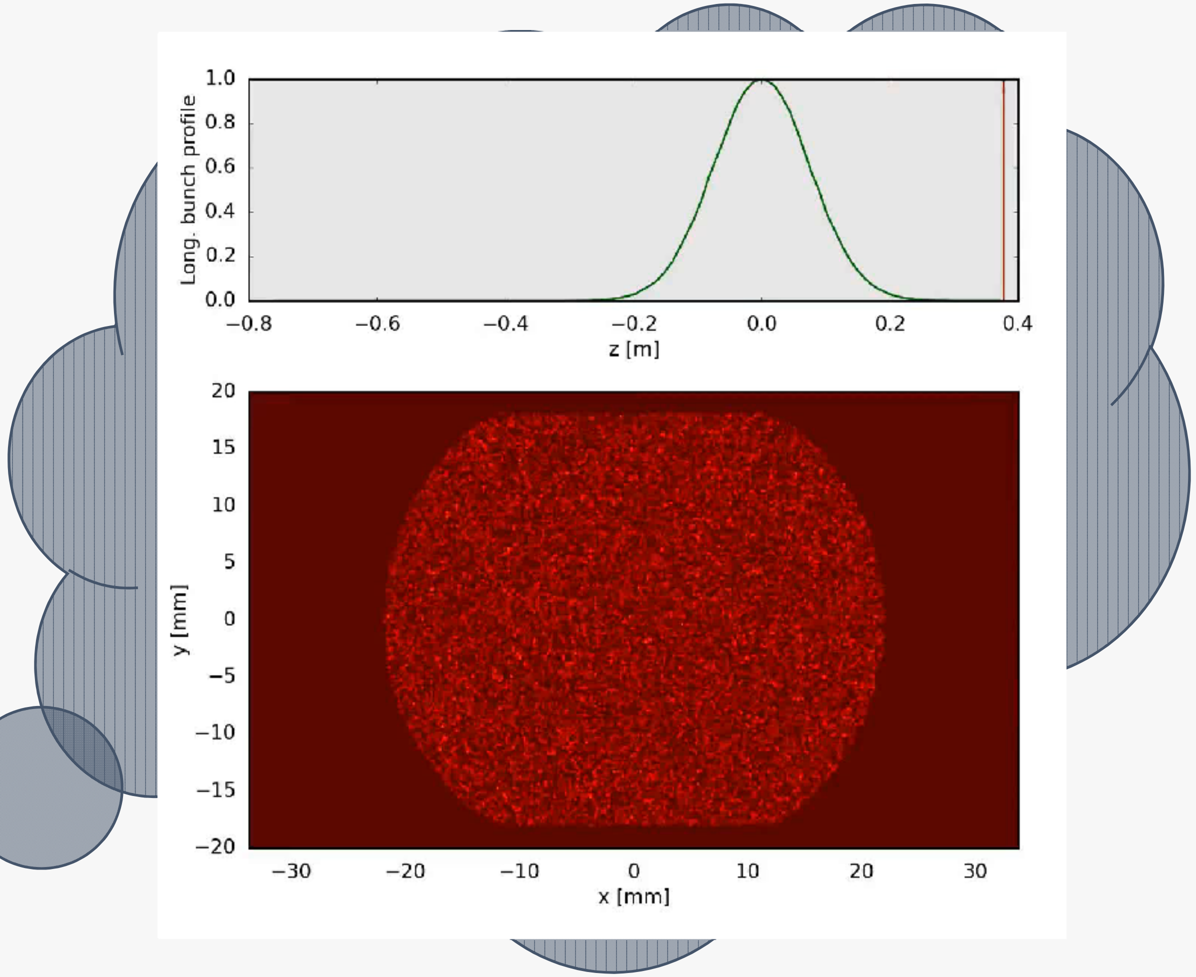
E-cloud beam system



Electron clouds in a quadrupole magnet



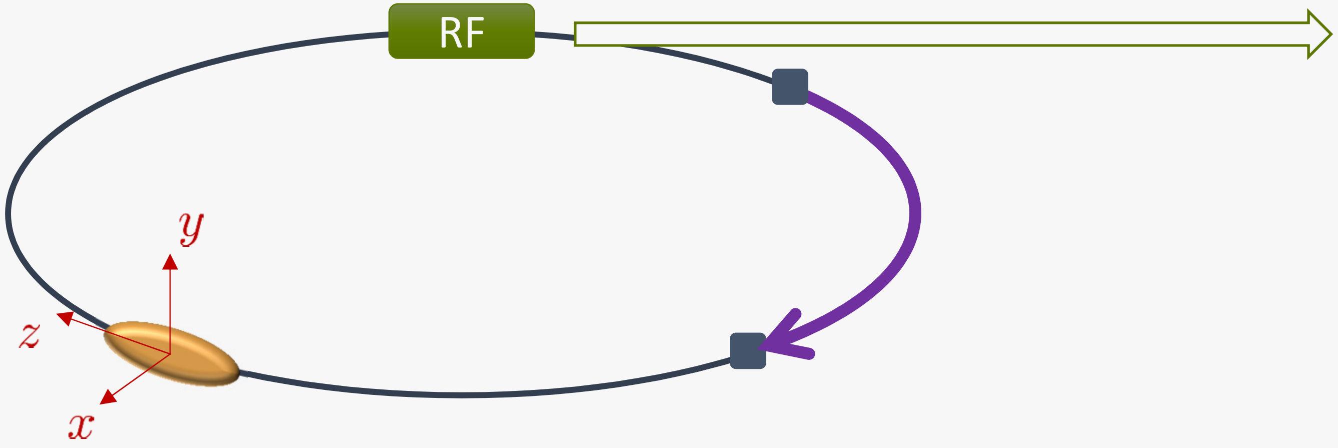
- Two stream collective interaction – much more involved

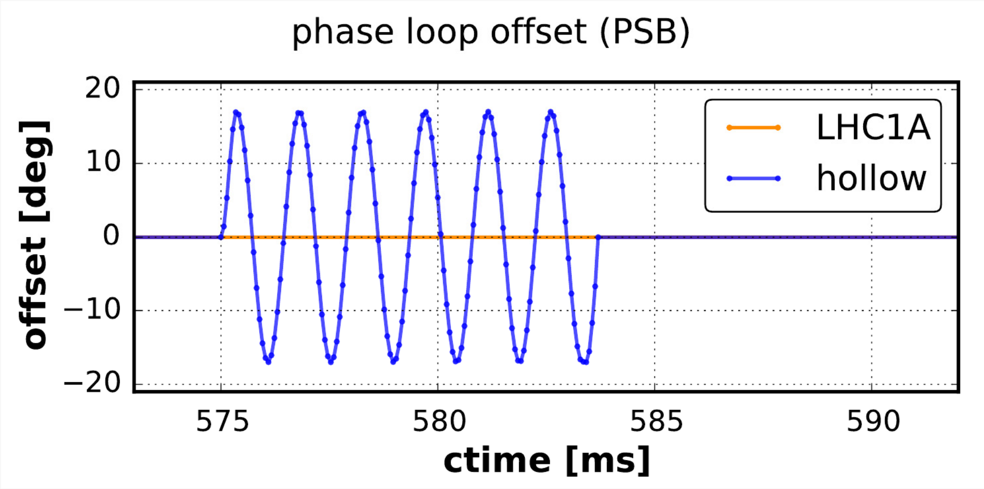


- Beam passage leads to a pinch of the cloud which in turn acts back on the beam – differently each turn

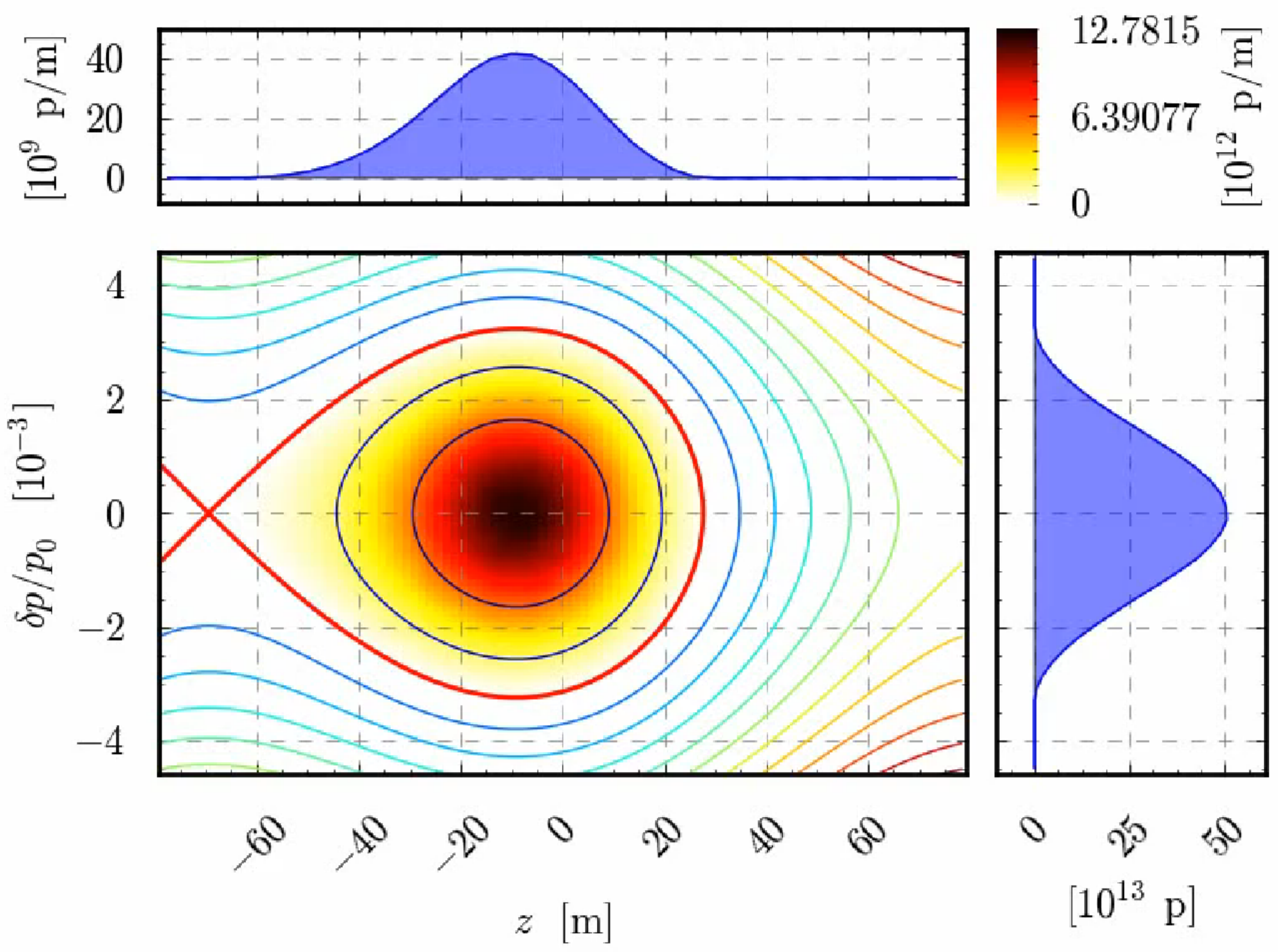
Hollow Bunches

Trim function:
dynamically change property of RF cavity

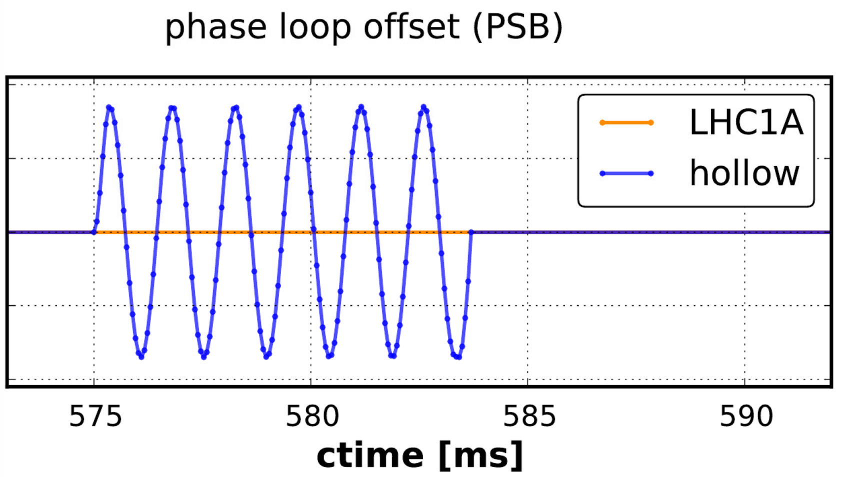




Hollow Bunches

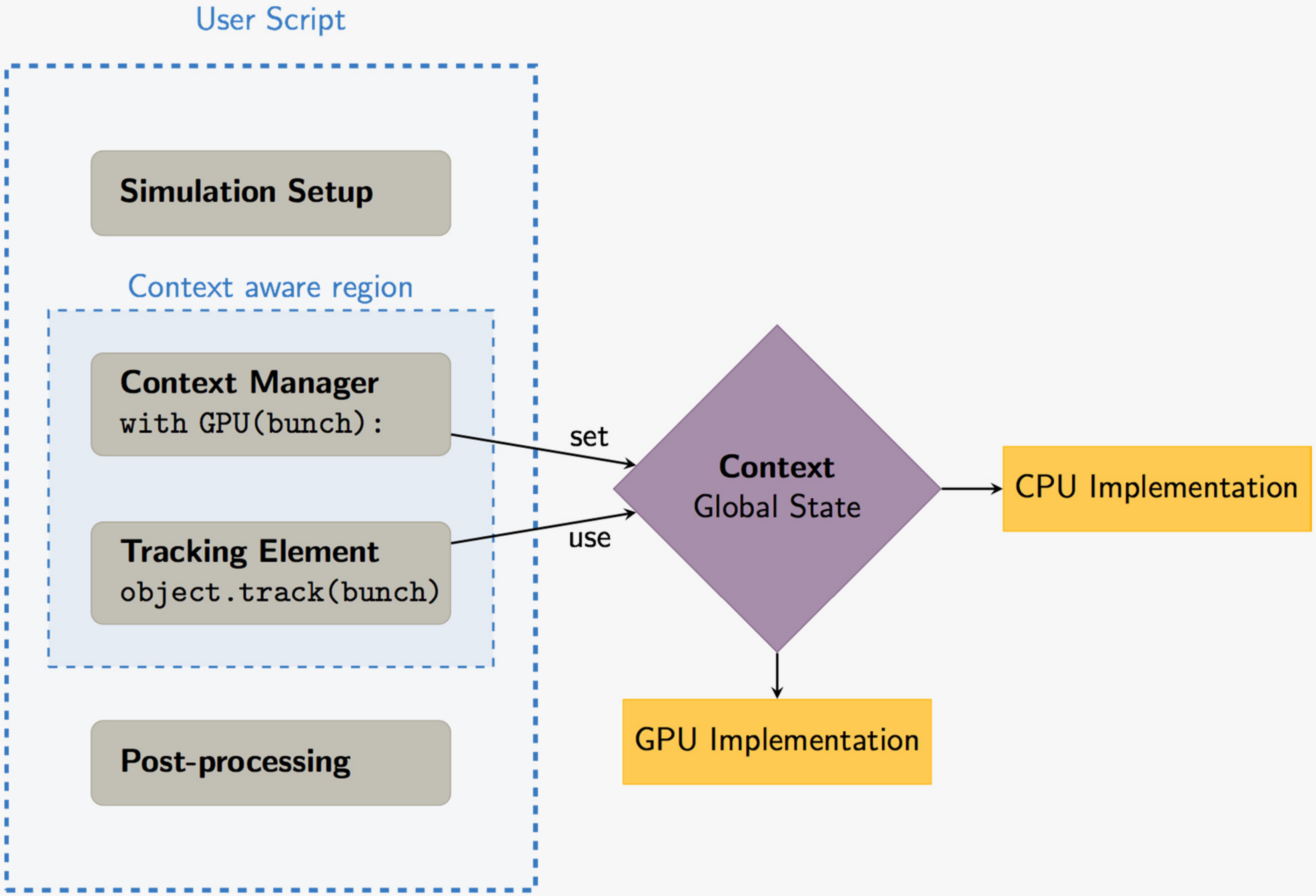


Courtesy Adrian Oeftiger



Context manager for GPU supported functions

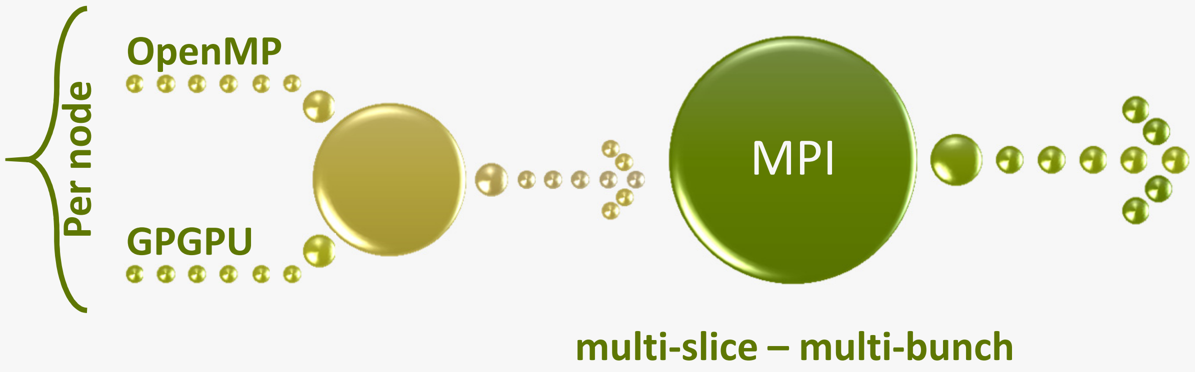
1. Setup simulation
2. Set context with context manager
3. Track bunch: correct implementation automatically chosen
4. Move data back to CPU



A user does not need to care about the system internals

Summary & Outlook

- Introduced collective effects and basic model of accelerator-beam systems used for simulations
- Highlighted some requirements for developers of modern computer simulation codes
- Highlighted the importance of performance and possibilities for improvement
- Showed some examples and applications
- Did not go into parallelisation techniques which **for collective effects become challenging**
- Parallelisation strategy would be to exploit all technics using a **hybrid approach**:



BACKUP

BACKUP

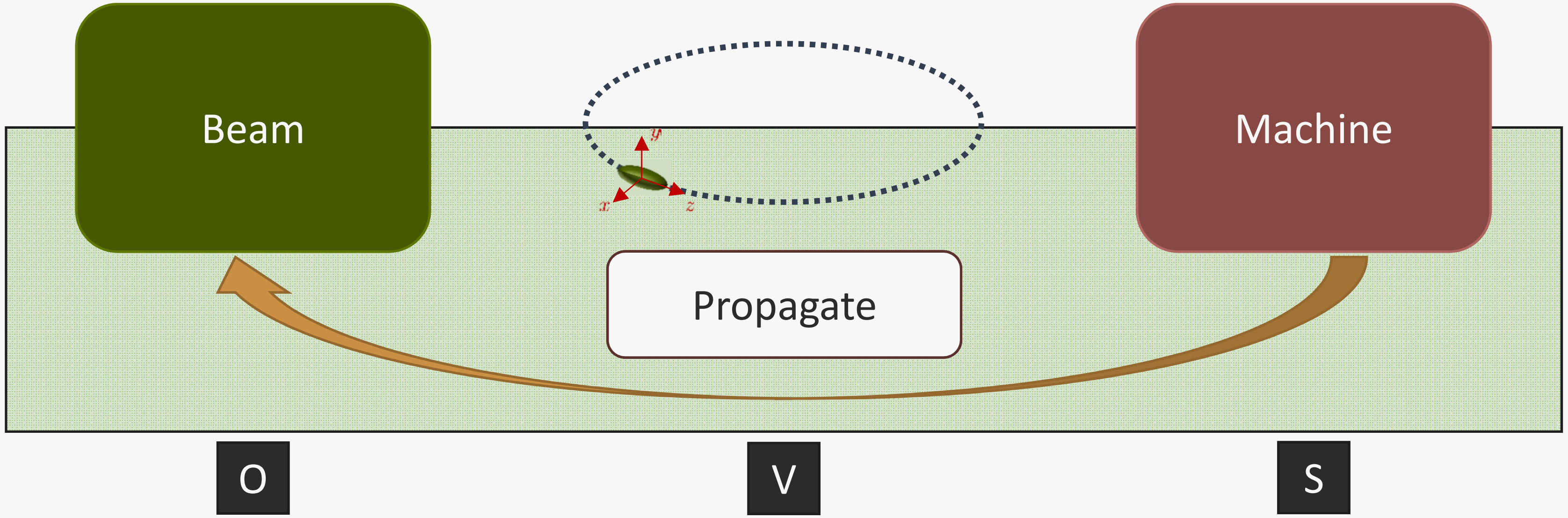
Backup

Outline:

1. Introduction
2. Basic model of the accelerator-beam system
3. Modern approaches and program architectures
4. Performance considerations
5. Applications, present status and perspectives

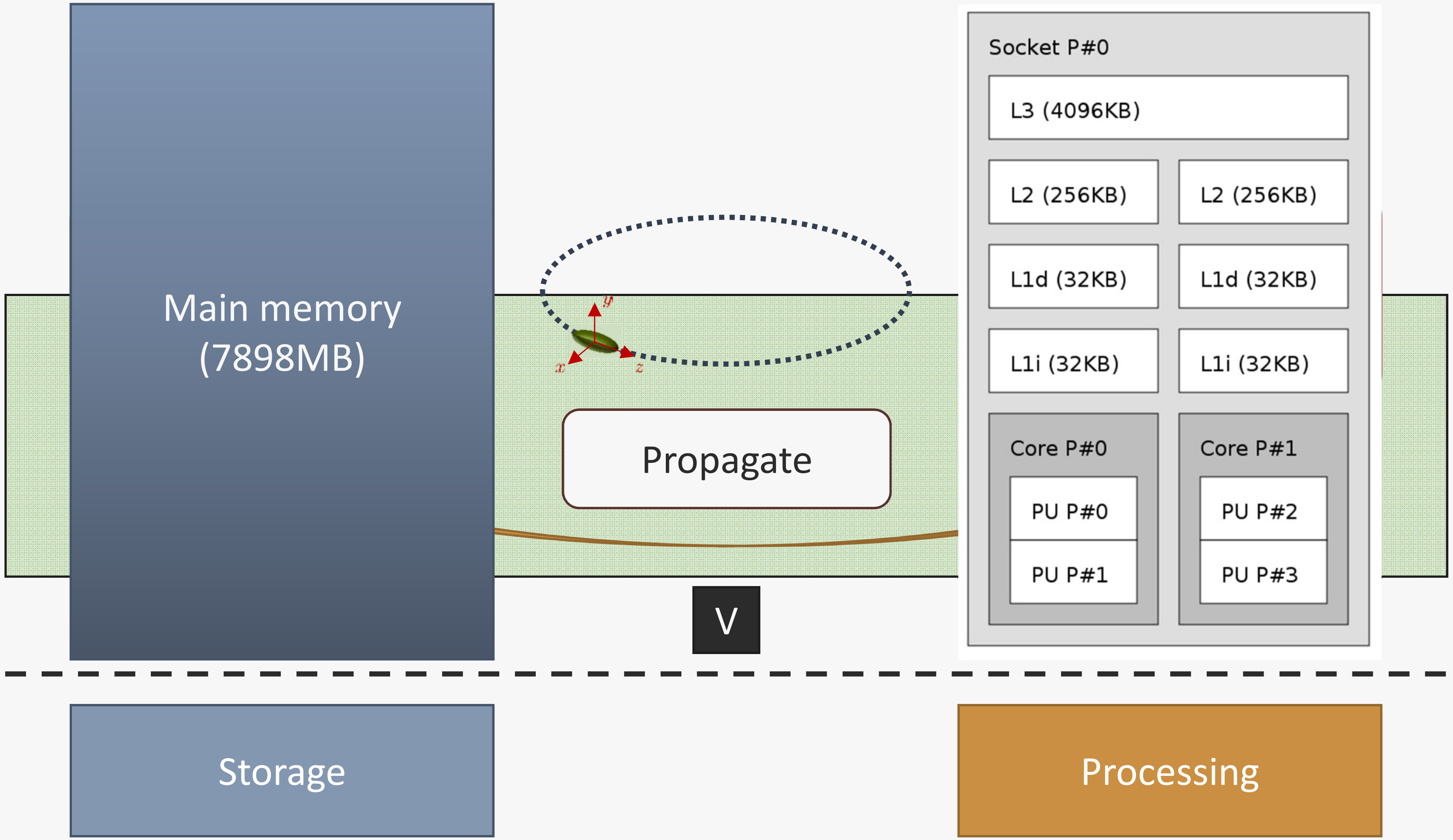
Mapping accelerator-beam to computer system

- Possible program layout



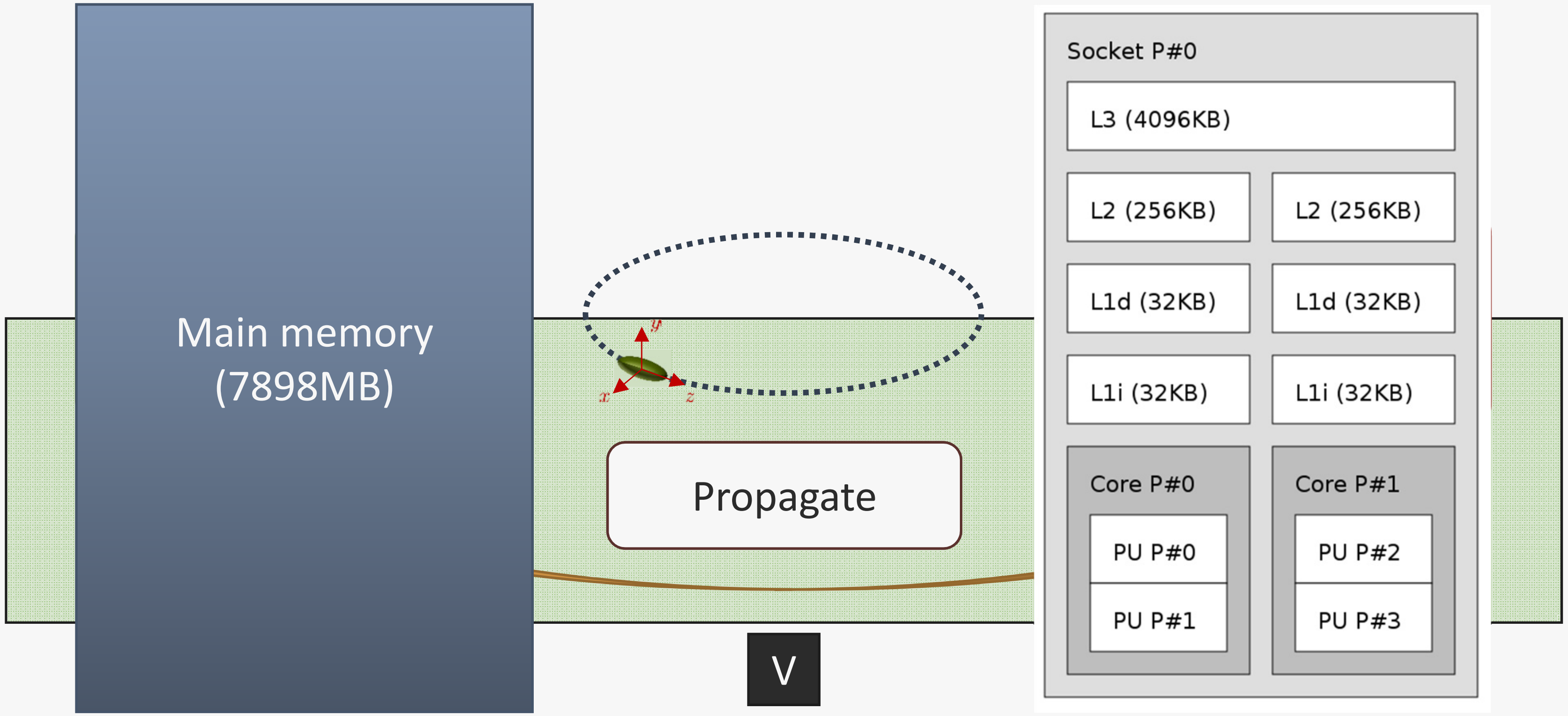
Mapping accelerator-beam to computer system

- Possible program layout



Mapping accelerator-beam to computer system

- Possible program layout

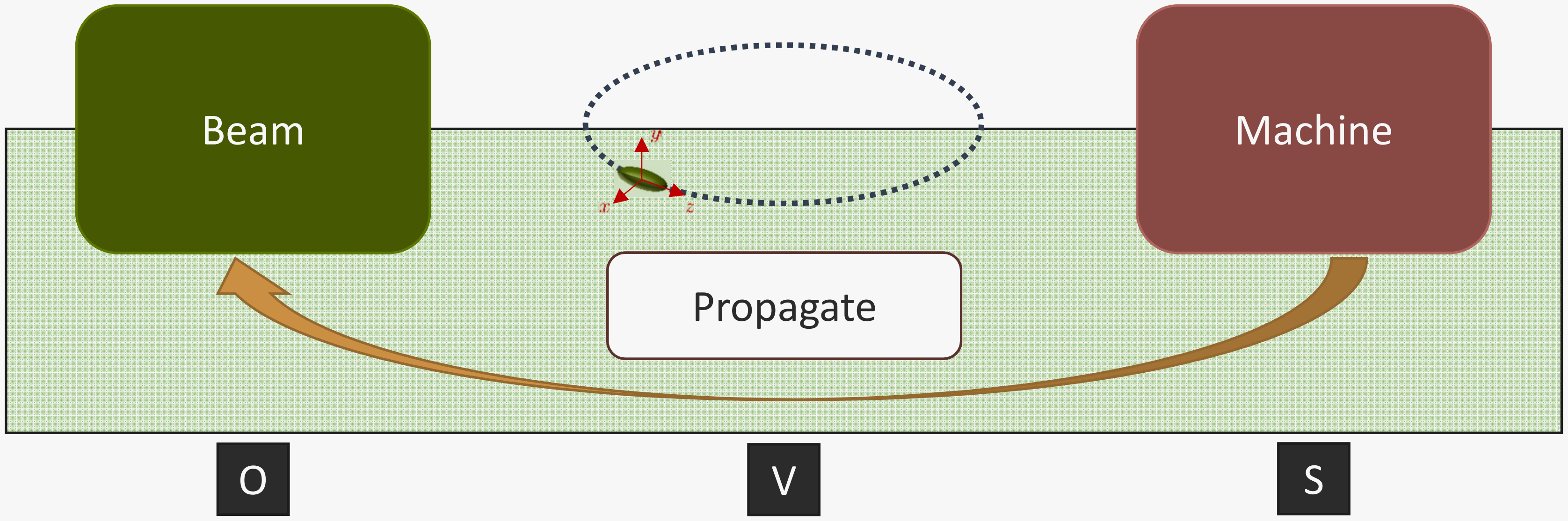


A data structure containing all coordinates and momenta — e.g.: ***class***

A function that knows how to update the coordinates and momenta — ***class A***, ***class B***, …

Mapping accelerator-beam to computer system

- Possible program layout



A data structure containing all coordinates and momenta

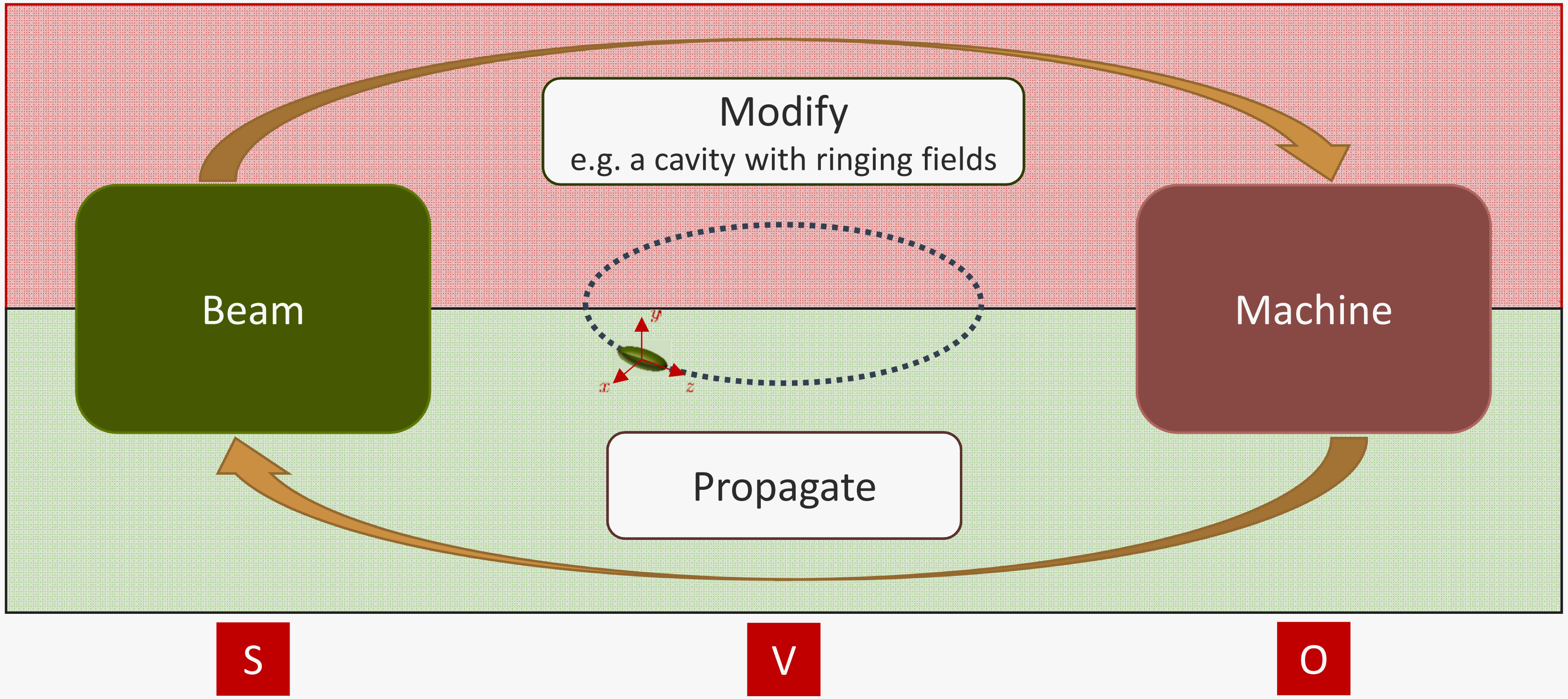
e.g.:
class

A function that knows how to update the coordinates and momenta

class A
class B
…

Mapping accelerator-beam to computer system

- Possible program layout



A data structure containing all coordinates and momenta	e.g.: ***class***

A function that knows how to update the coordinates and momenta	***class A*** ***class B*** …

Mapping accelerator-beam to computer system

- Possible program layout

By closing the loop, we introduce collective effects which can act as a feedback mechanism:

- New equilibrium solutions
- Damping or growth of the stationary solutions → instabilities

containing all coordinates and momenta — *class*

knows how to update the coordinates and momenta — *class B*

…

Backup

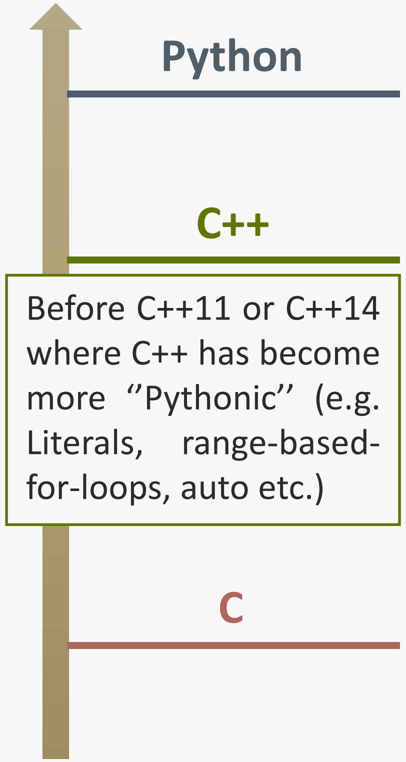
Outline:

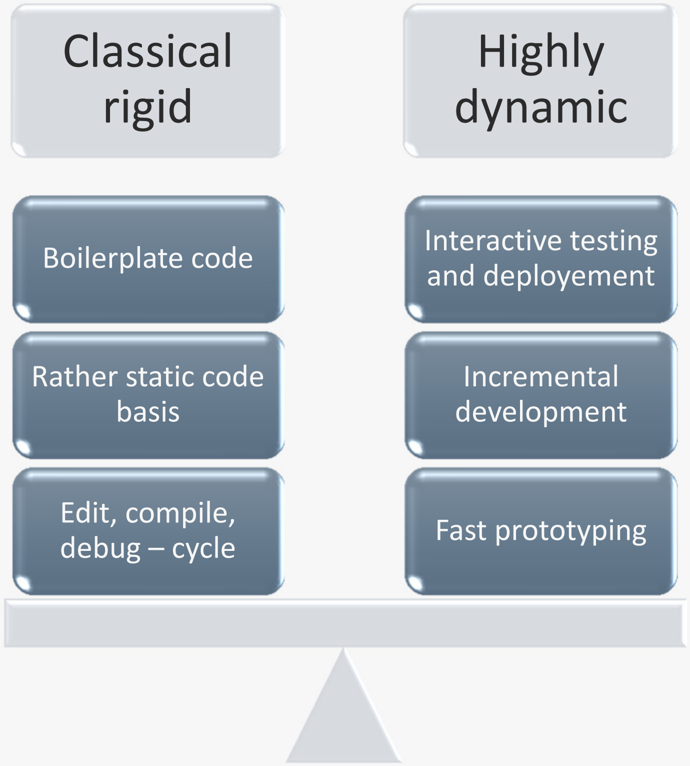
1. Introduction
2. Basic model of the accelerator-beam system
3. Modern approaches and program architectures
4. Performance considerations
5. Applications, present status and perspectives

Dynamic

- We work today with rapidly changing simulation demands – this requires frequent and rapid code development which should preferably be close to physics rather than close to the hardware.

Abstraction (example)

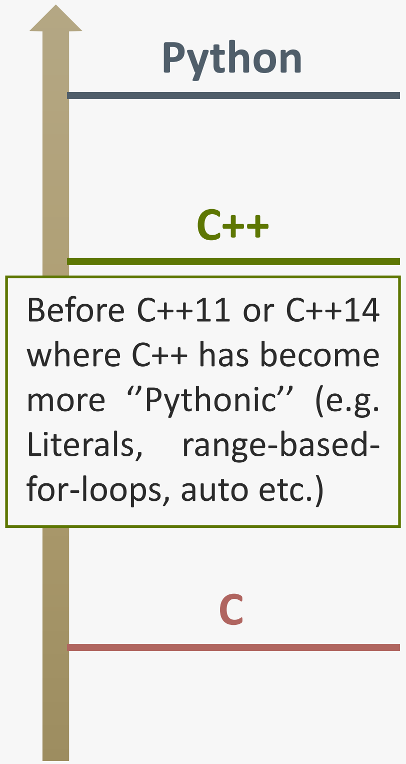




Dynamic

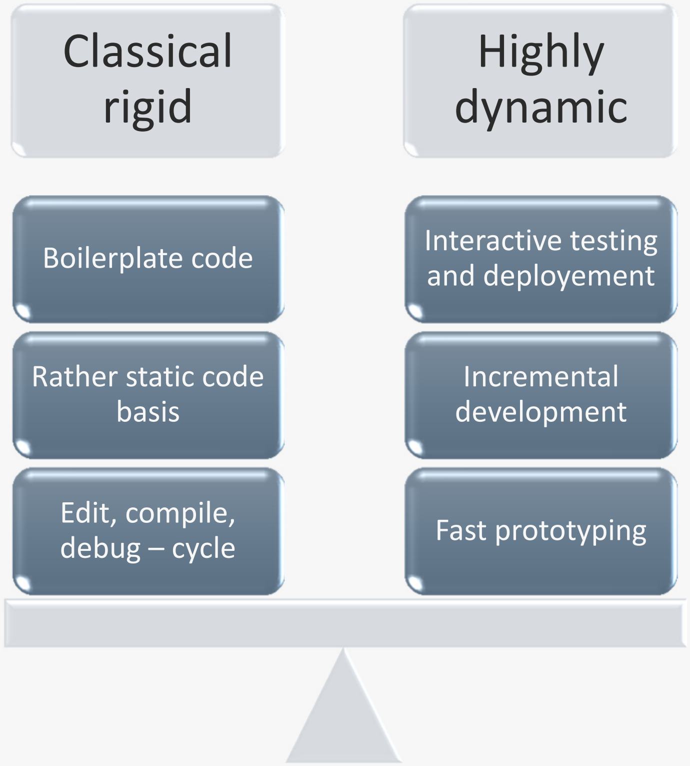
- We work today with rapidly changing simulation demands – this requires frequent and rapid code development which should preferably be close to physics rather than close to the hardware.

Abstraction (example)



Python is an object oriented script language while C++ is an object oriented compiling language.

Python code is … often 5-10 times shorter than equivalent C++ code! Anecdotal evidence suggests that one Python programmer can finish in two months what two C++ programmers can't complete in a year.



- We work today with rapidly changing simulation demands – this requires frequent and rapid code development which should preferably be close to physics rather than close to the hardware.

Abstraction (example)

An important hitch remains on the **FAST** requirement:

"Python was not made to be fast… … but to make developers fast."

Writing faster Python,
Sebastian Witowski

Being an interpreted language, strongly but dynamically typed, **Python can be slow in execution**! This needs to be handled.

Dynamic

- We work today with rapidly changing simulation demands – this requires frequent and rapid code development which should preferably be close to physics rather than close to the hardware.

Abstraction

An im

Being

be slo

Coding. Crawler

Java

```
public class CrawlerExample {

    public static void main(String[] args) throws IOException {
        PrintWriter textFile = null;
        try {
            textFile = new PrintWriter("result.txt");
            System.out.println("Enter the URL you wish to crawl..");
            System.out.print("@> ");
            String myUrl = new Scanner(System.in).nextLine();

            String response = getContentByUrl(myUrl);

            Matcher matcher = Pattern
              .compile("href=[\"'](.[^\"']+)[\"']").matcher(response);
            while (matcher.find()) {
                String url = matcher.group(1);
                System.out.println(url);
                textFile.println(url);
            }
        } finally {
            if(textFile != null) {
                textFile.close();
            }
        }
    }

    private static String getContentByUrl(String myUrl)
              throws IOException {
        URL url = new URL(myUrl);
        URLConnection urlConnection = url.openConnection();
        BufferedReader in = null;
        StringBuilder response = new StringBuilder();
        try {
            in = new BufferedReader(new InputStreamReader
              (urlConnection.getInputStream()));
            String inputLine;
            while ((inputLine = in.readLine()) != null) {
                response.append(inputLine);
            }
        } finally {
            if(in != null) {
                in.close();
            }
        }
        return response.toString();
    }
}
```

Python

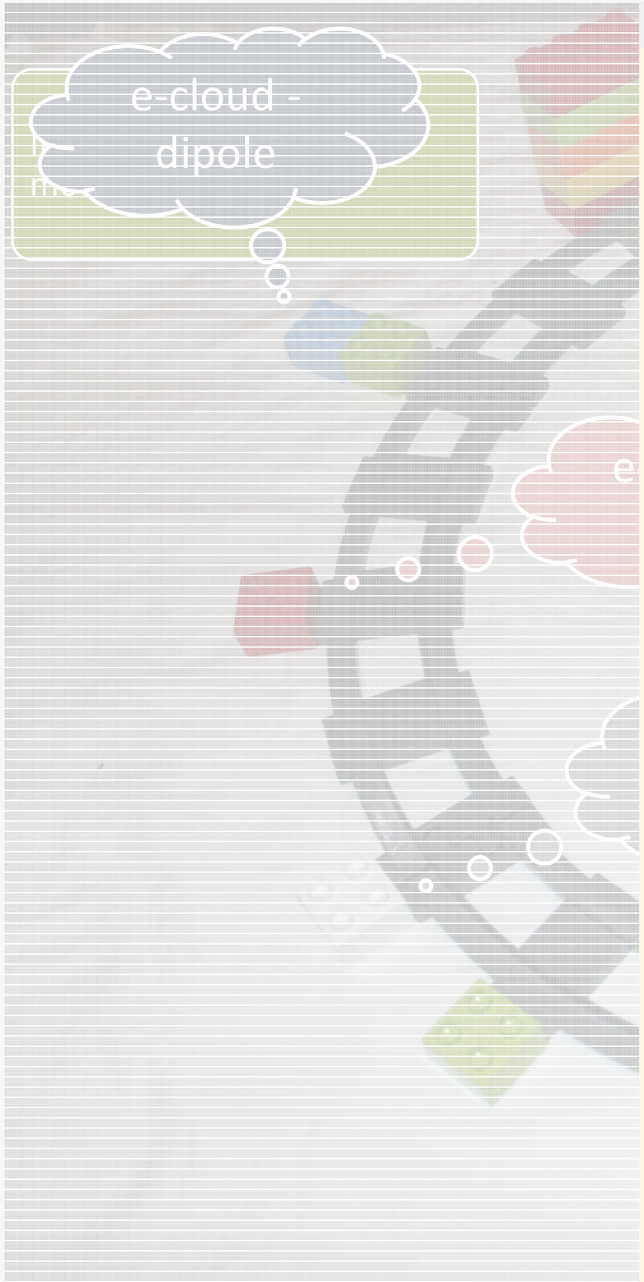
```
if __name__ == '__main__':
    with open("result.txt","wt") as textFile:
        print("Enter the URL you wish to crawl..")
        myUrl = input("@> ")
        for i in re.findall("href=[\"'](.[^\"']+)[\"']",
            urllib.request.urlopen(myUrl).read().decode(), re.I):
            print(i)
            textFile.write(i+'\n')
```

Python,

/itowski

on can

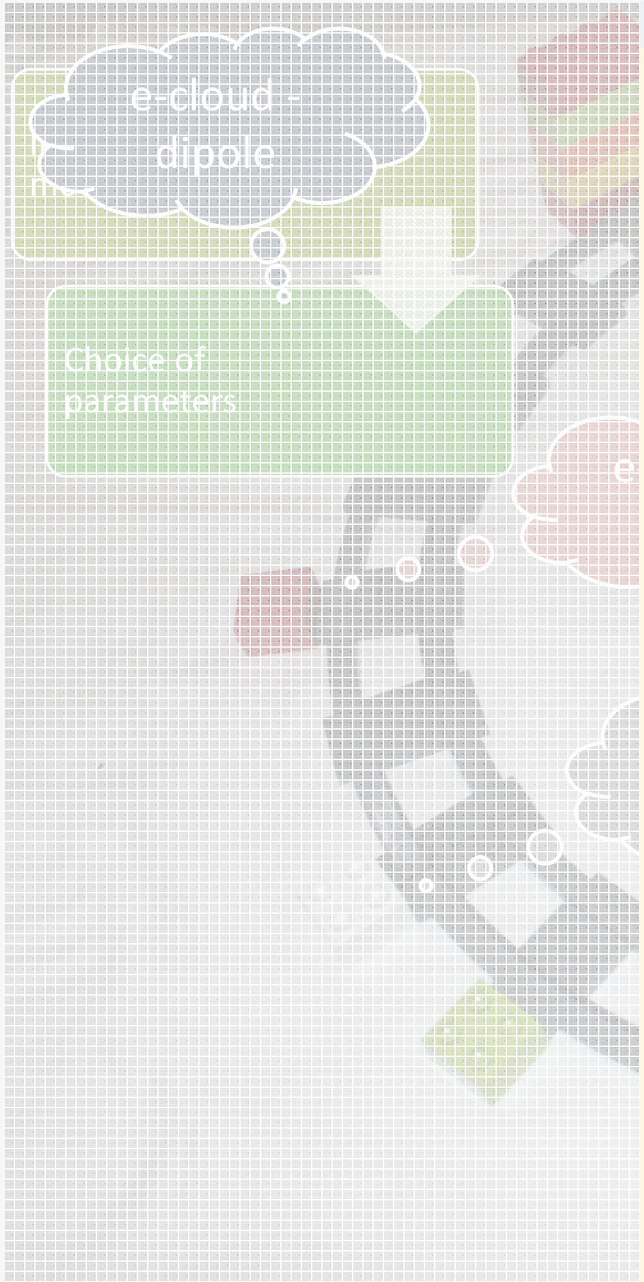
Building the simulation



File Edit Options Buffers Tools Python Virtual Envs Elpy YASnippet Help

```python
from __future__ import division

import time
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from scipy.constants import c, e, m_p

from PyHEADTAIL.trackers.rf_bucket import RFBucket
from PyHEADTAIL.particles.generators import ParticleGenerator
from PyHEADTAIL.particles.generators import gaussian2D, RF_bucket_distribution

from PyHEADTAIL.trackers.transverse_tracking import TransverseMap
from PyHEADTAIL.trackers.detuners import Chromaticity, AmplitudeDetuning

from PyHEADTAIL.trackers.simple_long_tracking import RFSystems

from PyHEADTAIL.particles.slicing import UniformBinSlicer
from PyHEADTAIL.impedances.wakes import CircularResonator, WakeField

from PyHEADTAIL.feedback.transverse_damper import TransverseDamper

from PyHEADTAIL.monitors.monitors import BunchMonitor, SliceMonitor


plt.switch_backend('TkAgg')
sns.set_context('talk', font_scale=1.5,
                rc={'lines.markeredgewidth': 1})
sns.set_style('darkgrid', {
    'axes.linewidth': 2,
    'legend.fancybox': True})


# PARAMETERS
# ==========
p0 = 7000e9 * e/c
E0 = p0*c
gamma = np.sqrt((p0/(m_p*c))**2 + 1)
beta = np.sqrt(1 - gamma**-2)
betagamma = np.sqrt(gamma**2 - 1)

C = 26658.883
R = C/(2*np.pi)
T0 = C/(beta*c)
omega0 = 2*np.pi/T0
```

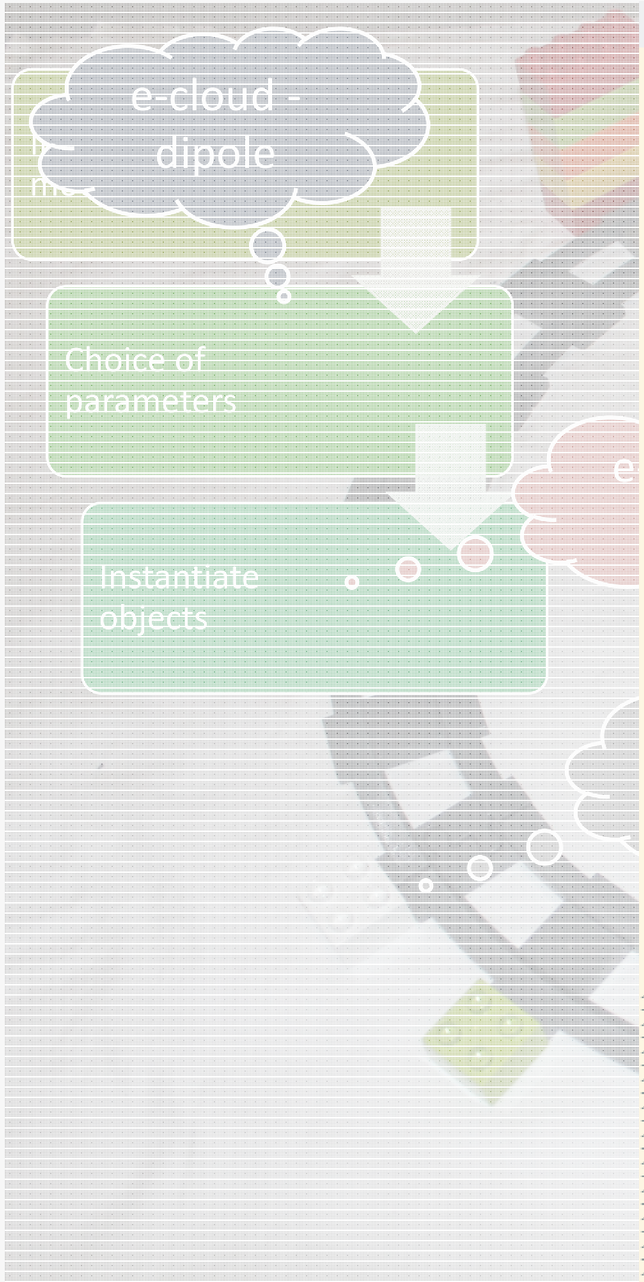
Building the simulation



File Edit Options Buffers Tools Python Virtual Envs Elpy YASnippet Help

```python

# PARAMETERS
# ==========
p0 = 7000e9 * e/c
E0 = p0*c
gamma = np.sqrt((p0/(m_p*c))**2 + 1)
beta = np.sqrt(1 - gamma**-2)
betagamma = np.sqrt(gamma**2 - 1)

C = 26658.883
R = C/(2*np.pi)
T0 = C/(beta*c)
omega0 = 2*np.pi/T0
alpha = 53.86**-2
eta = alpha - gamma**-2

V_RF = [16e6, 0*8e6]
h_RF = [35640, 71280]
dphi_RF = [0, 0*np.pi]

macroparticlenumber = 50000
intensity = 1.3e11
epsn_x = 2.2e-6
epsn_y = 2.2e-6
epsn_z = 2.5

Q_x = 62.31
Q_y = 60.32
Qp_x = 0
Qp_y = 0
beta_x = R/Q_x
beta_y = R/Q_y


# PARTICLE DISTRIBUTION
# =====================
rfbucket = RFBucket(
    circumference=C, charge=e, mass=m_p, gamma=gamma,
    alpha_array=[alpha], p_increment=0,
    harmonic_list=h_RF, voltage_list=V_RF, phi_offset_list=dphi_RF)

bunch = ParticleGenerator(
    macroparticlenumber=macroparticlenumber, intensity=intensity,
    charge=e, mass=m_p, gamma=gamma, circumference=C,
    distribution_x=gaussian2D(epsn_x/betagamma), beta_x=beta_x,
```

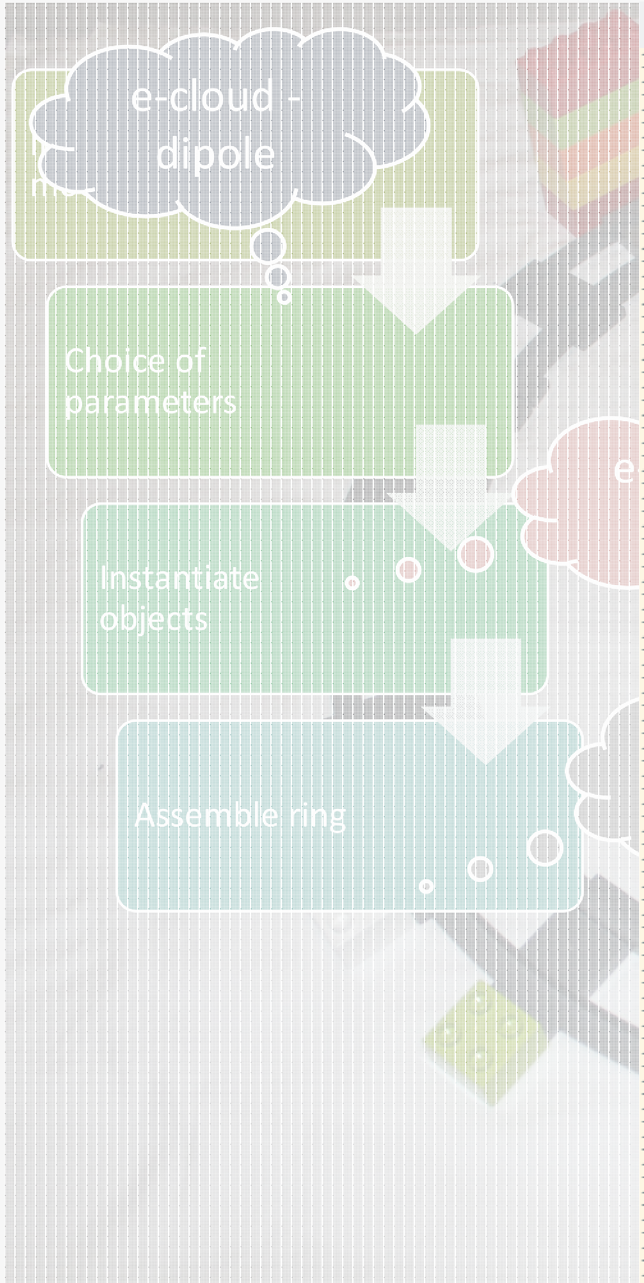
Building the simulation



File Edit Options Buffers Tools Python Virtual Envs Elpy YASnippet Help

```python

# PARTICLE DISTRIBUTION
# =====================
rfbucket = RFBucket(
    circumference=C, charge=e, mass=m_p, gamma=gamma,
    alpha_array=[alpha], p_increment=0,
    harmonic_list=h_RF, voltage_list=V_RF, phi_offset_list=dphi_RF)

bunch = ParticleGenerator(
    macroparticlenumber=macroparticlenumber, intensity=intensity,
    charge=e, mass=m_p, gamma=gamma, circumference=C,
    distribution_x=gaussian2D(epsn_x/betagamma), beta_x=beta_x,
    distribution_y=gaussian2D(epsn_y/betagamma), beta_y=beta_y,
    distribution_z=RF_bucket_distribution(rfbucket, epsn_z=epsn_z)).generate()
bunch.x += 3.5e-4


# TRANSVERSE MAP
# ==============
n_segments = 3

s = np.array([i * C/n_segments for i in range(n_segments + 1)])
alpha_x = np.zeros(n_segments + 1)
beta_x = np.ones(n_segments + 1) * beta_x
D_x = np.zeros(n_segments + 1)
alpha_y = np.zeros(n_segments + 1)
beta_y = np.ones(n_segments + 1) * beta_y
D_y = np.zeros(n_segments + 1)

detuners = [Chromaticity(Qp_x, Qp_y),
            AmplitudeDetuning(-4e-9, 4e-9, -2e-10)]

transverse_map = TransverseMap(
    s=s,
    alpha_x=alpha_x, beta_x=beta_x, D_x=D_x,
    alpha_y=alpha_y, beta_y=beta_y, D_y=D_y,
    accQ_x=Q_x, accQ_y=Q_y,
    detuners=detuners)
one_turn_map = [m for m in transverse_map]


# LONGITUDINAL MAP
# ================
longitudinal_map = RFSystems(
    C, h_RF, V_RF, dphi_RF, [alpha], gamma, charge=e, mass=m_p)
```

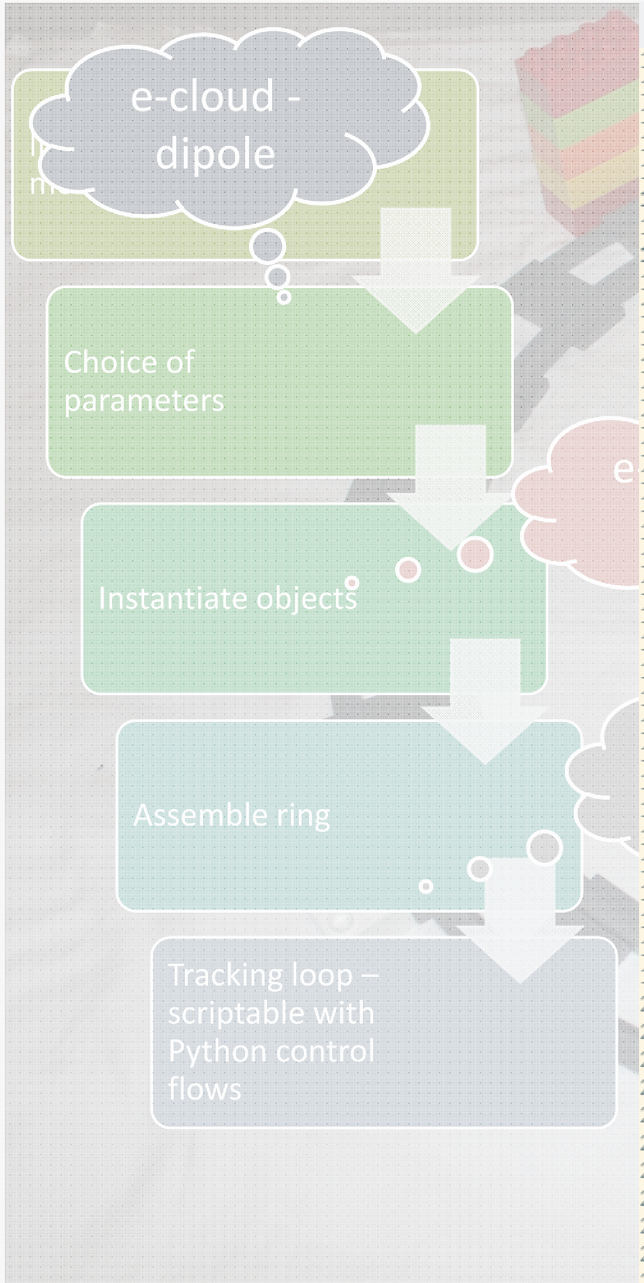
Building the simulation



File Edit Options Buffers Tools Python Virtual Envs Elpy YASnippet Help

```python

# CREATE WAKEFIELDS
# =================
slicer_for_wakefields = UniformBinSlicer(300, n_sigma_z=3)
wake = CircularResonator(R_shunt=1e6, frequency=700e6, Q=350, n_turns_wake=4)
wakefields = WakeField(slicer_for_wakefields, wake, circumference=C)


# CREATE DAMPER
# =============
dampingrate_x = 50
dampingrate_y = 50
damper = TransverseDamper(dampingrate_x, dampingrate_y)


# CREATE MONITORS
# ===============
sigma_z = bunch.sigma_z()
slicer_for_slicemonitor = UniformBinSlicer(50, z_cuts=(-3*sigma_z, 3*sigma_z))
simulation_parameters_dict = {
    'gamma': gamma,
    'intensity': intensity,
    'Qx': Q_x,
    'Qy': Q_y,
    'Qs': rfbucket.Qs,
    'beta_x': bunch.beta_Twiss_x(),
    'beta_y': bunch.beta_Twiss_y(),
    'beta_z': rfbucket.beta_z,
    'epsn_x': bunch.epsn_x(),
    'epsn_y': bunch.epsn_y(),
    'sigma_z': bunch.sigma_z(),
}
bunchmonitor = BunchMonitor('./bunchmonitor', 8192, simulation_parameters_dict)
slicemonitor = SliceMonitor('./slicemonitor',  8192, slicer_for_slicemonitor,
                            simulation_parameters_dict)


# TRACKING LOOP
# =============
xx = np.linspace(-3e-3, 3e-3, 400)
yy = np.linspace(-3e-5, 3e-5, 400)
XX, YY = np.meshgrid(xx, yy)
JJ = -np.sqrt(XX**2 + (transverse_map.beta_x[0] * YY)**2)

zz = np.linspace(-6, 6, 400)
```

Building the simulation



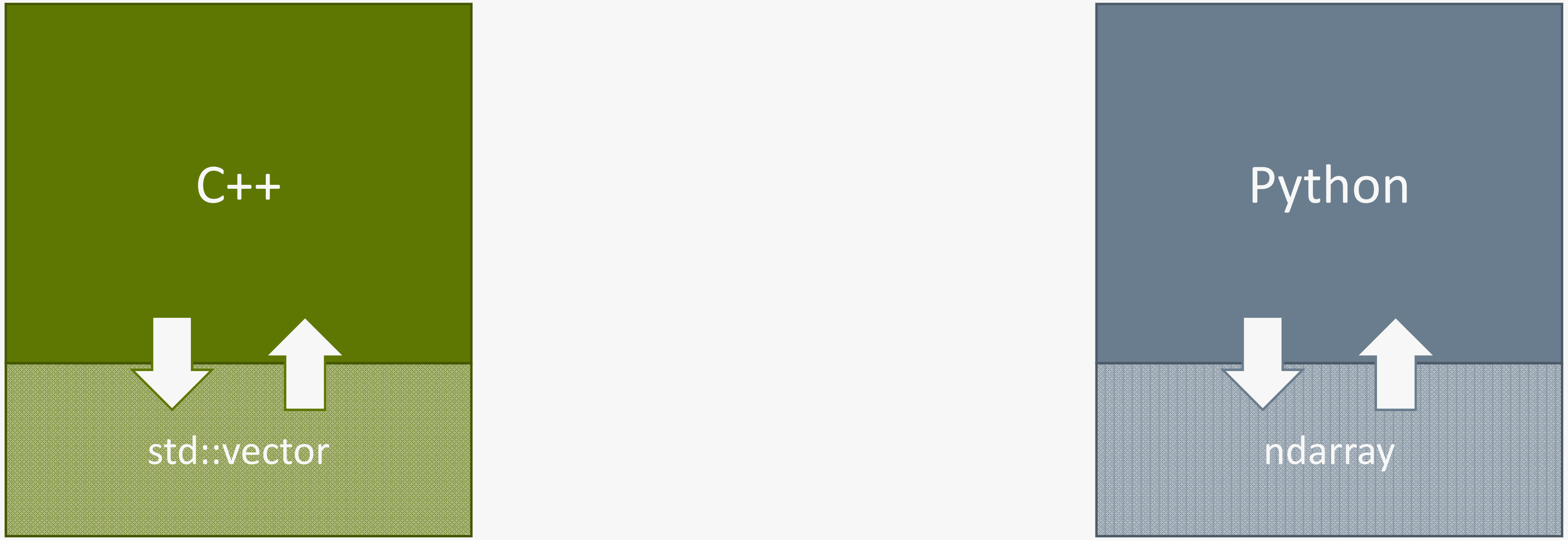
File Edit Options Buffers Tools Python Virtual Envs Elpy YASnippet Help

```
s_cnt = 0
n_turns = 256
monitorswitch = False

print '\n--> Begin tracking...\n'

for i in range(n_turns):
    t0 = time.clock()
    for m in one_turn_map:
        m.track(bunch)

    bunchmonitor.dump(bunch)

    if not monitorswitch:
        if (bunch.mean_x() > 1e3 or bunch.mean_y() > 1e3 or i > n_turns-8192):
            print "--> Activate monitor"
            monitorswitch = True
    else:
        if s_cnt < 8192:
            slicemonitor.dump(bunch)
            s_cnt += 1

    if (i+1) % 1 is not 0:
        continue

    Jx = np.sqrt((bunch.x-bunch.mean_x())**2 +
                 (transverse_map.beta_x[0] * (bunch.xp-bunch.mean_xp()))**2)
    ax1.contour(XX, YY, JJ, 6, lw=1)
    ax1.scatter(bunch.x, bunch.xp, c=Jx, cmap=plt.cm.viridis_r, s=6)
    ax2.contour(ZZ, PP, HH, 6, lw=1)
    ax2.contour(ZZ, PP, HH, levels=[0], colors='darkred', lw=3)
    ax2.scatter(bunch.z, bunch.dp, c=rfbucket.hamiltonian(bunch.z, bunch.dp),
                cmap=plt.cm.viridis, s=6)
    ax1.set_xlim(-1.3e-3, 1.3e-3)
    ax1.set_ylim(-2e-5, 2e-5)
    ax2.set_xlim(-.5, .5)
    ax2.set_ylim(-5e-4, 5e-4)
    ax1.set_xlabel("$x [m]$", fontsize=24)
    ax1.set_ylabel("$x'$", fontsize=24)
    ax2.set_xlabel("$z [m]$", fontsize=24)
    ax2.set_ylabel("$\delta$", fontsize=24)
    fig.suptitle("Turn# {:g}/{:g}".format(i+1, n_turns), fontsize=22)
    plt.savefig('Movies/movie-{:03d}.png'.format(i+1), dpi=72)
    # plt.draw()
```

Outline:

1. Introduction
2. Basic model of the accelerator-beam system
3. Modern approaches and program architectures
4. Performance considerations
5. Applications, present status and perspectives

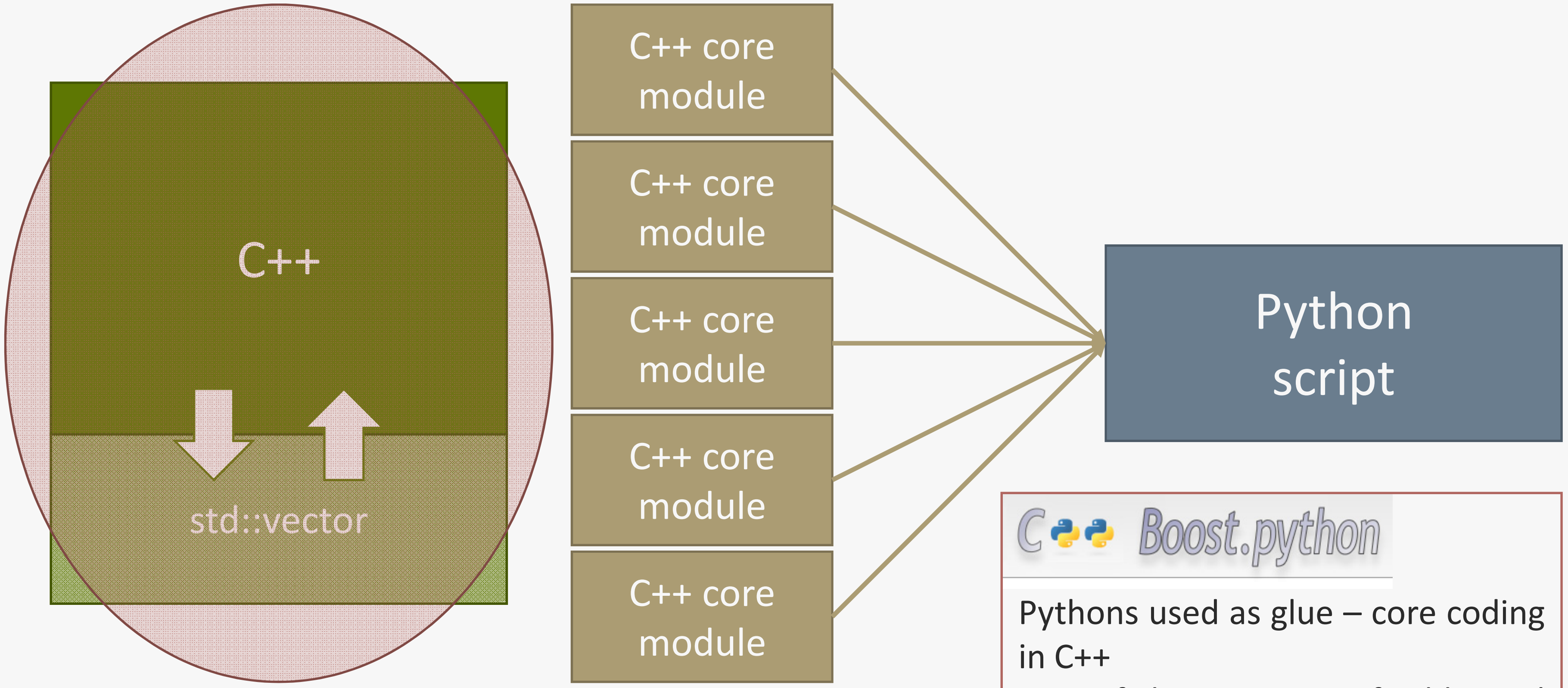
Where do I want to spend most of my time coding in – in which world do I want to live?



This is where my native data structures should reside.

Python extensions

Where do I want to spend most of my time coding in – in which world do I want to live?

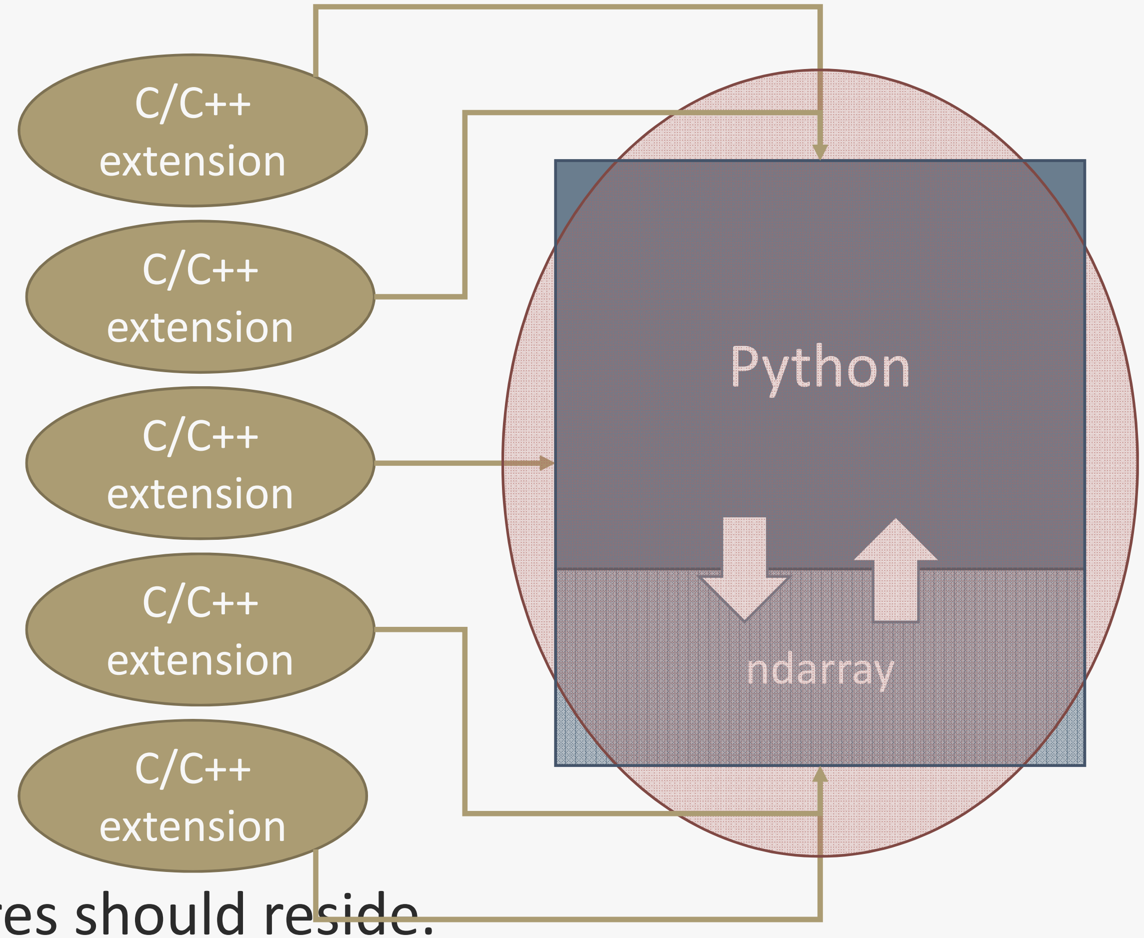


This is where my native data structures should reside.

Pythons used as glue – core coding in C++
Lots of data copying of additional coding is to be done in Python

Where do I want to spend most of my time coding in – in which world do I want to live?

C/C++
module



Cython

Small low-level extensions for Python accessing native Python data structures – core coding in Python

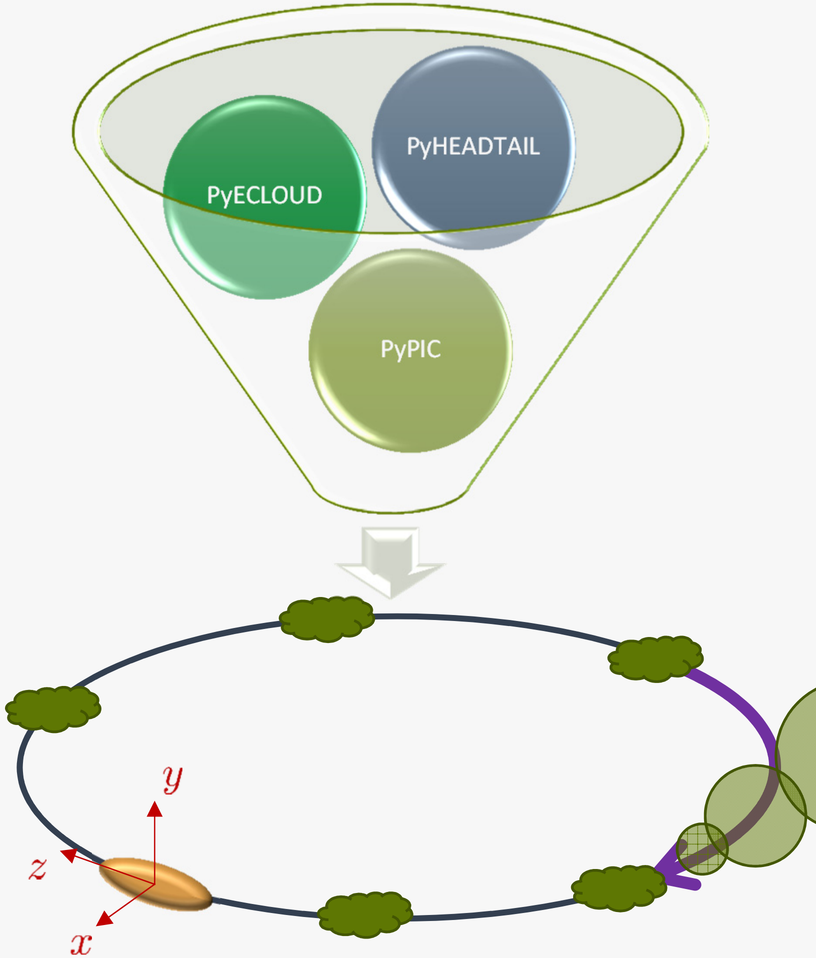
This is where my native data structures should reside.

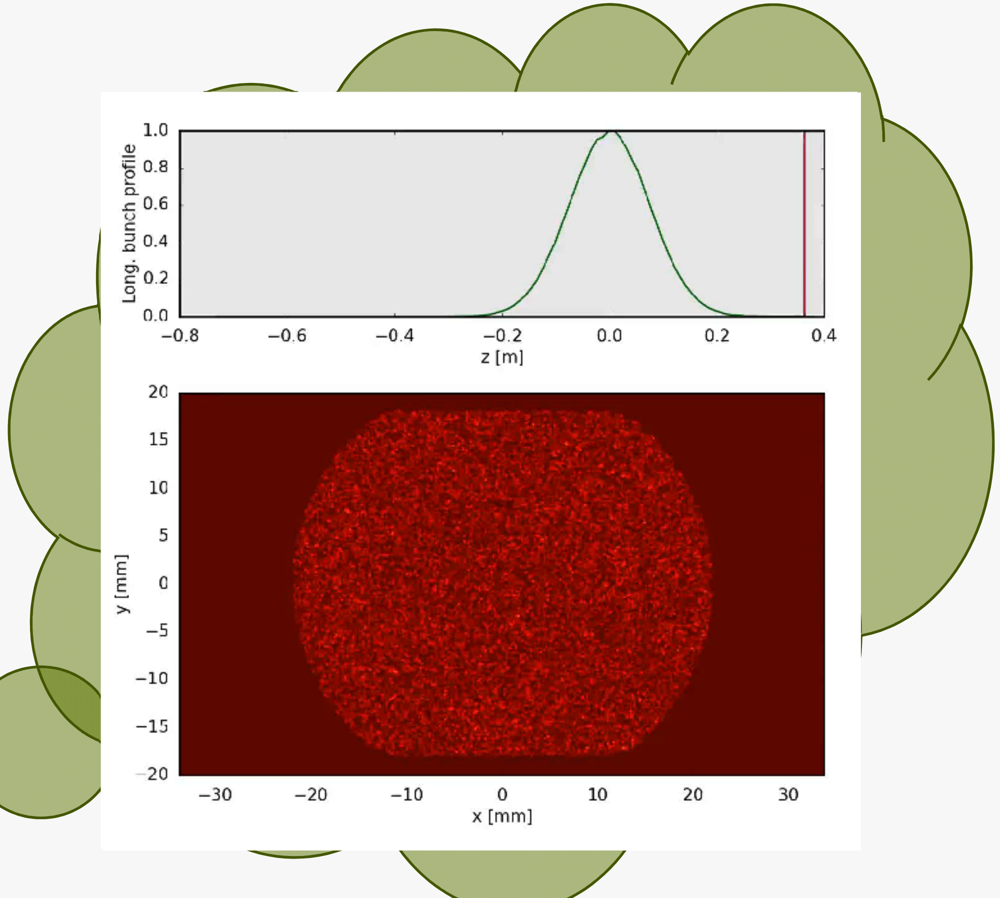
Outline:

1. Introduction
2. Basic model of the accelerator-beam system
3. Modern approaches and program architectures
4. Performance considerations
5. Applications, present status and perspectives

Electron clouds in a bending magnet

- Two stream collective interaction – much more involved





- Beam passage leads to a pinch of the cloud which in turn acts back on the beam – differently each turn

Results: Benchmark Study

Typical application:

LHC@injection instability,
impedance + transverse damper

CPU Time: ~ 1day
GPU Time: 5x less

Two lines of code added to script

Benchmark for CPU
Results agree