

# cobra-HeadTail

**ICE Meeting**  
6. November 2013, CERN

Kevin Li,  
Adrian Oeftiger, Giovanni Rumolo



# Overview

- Motivation
- Model
- Design
- Benchmark
- Demo
- Conclusions

# Motivation

# What is cobra-HeadTail?

A little success story:

**E CLOUD**

# What is cobra-HeadTail?

A little success story:



Written in Python:

- Interpreted language (open source), **allowing incremental and interactive development** of the code, encouraging an highly modular structure
- **Libraries for scientific computation** (e.g Numpy, Scipy, Pylab)
- Extensible with **C/C++ or FORTRAN compiled** modules for computationally intensive parts

# What is cobra-HeadTail?

A little success story:

E CLOUD  $\xrightarrow{\text{Python}}$  PyE CLOUD

HeadTail

# What is cobra-HeadTail?

A little success story:

E CLOUD  $\xrightarrow{\text{Python}}$  PyE CLOUD

HeadTail  $\longrightarrow$  PyHeadTail

# What is cobra-HeadTail?

A little success story:

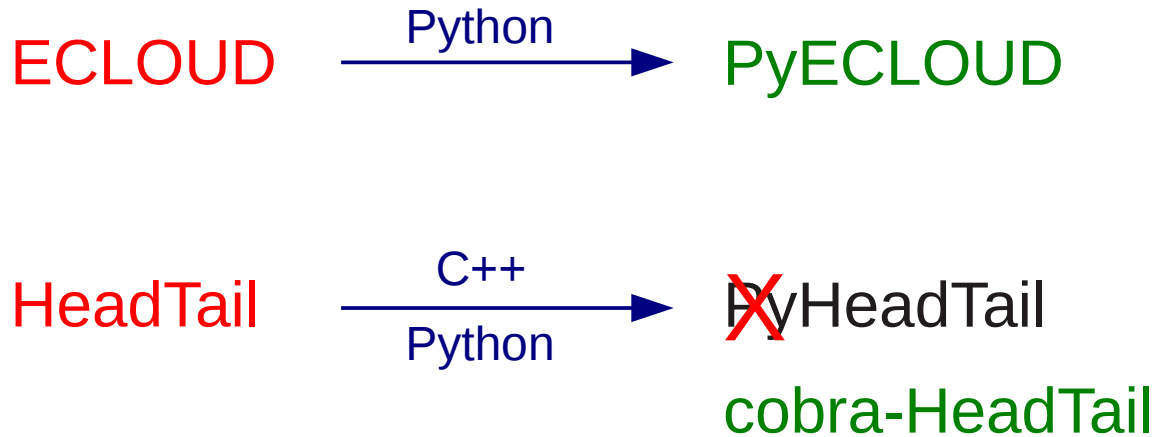
**E**CLOUD  $\xrightarrow{\text{Python}}$  **Py**ECLOUD

**Head**Tail  $\xrightarrow[\text{Python}]{\text{C++}}$  ~~Py~~HeadTail



# What is cobra-HeadTail?

A little success story:

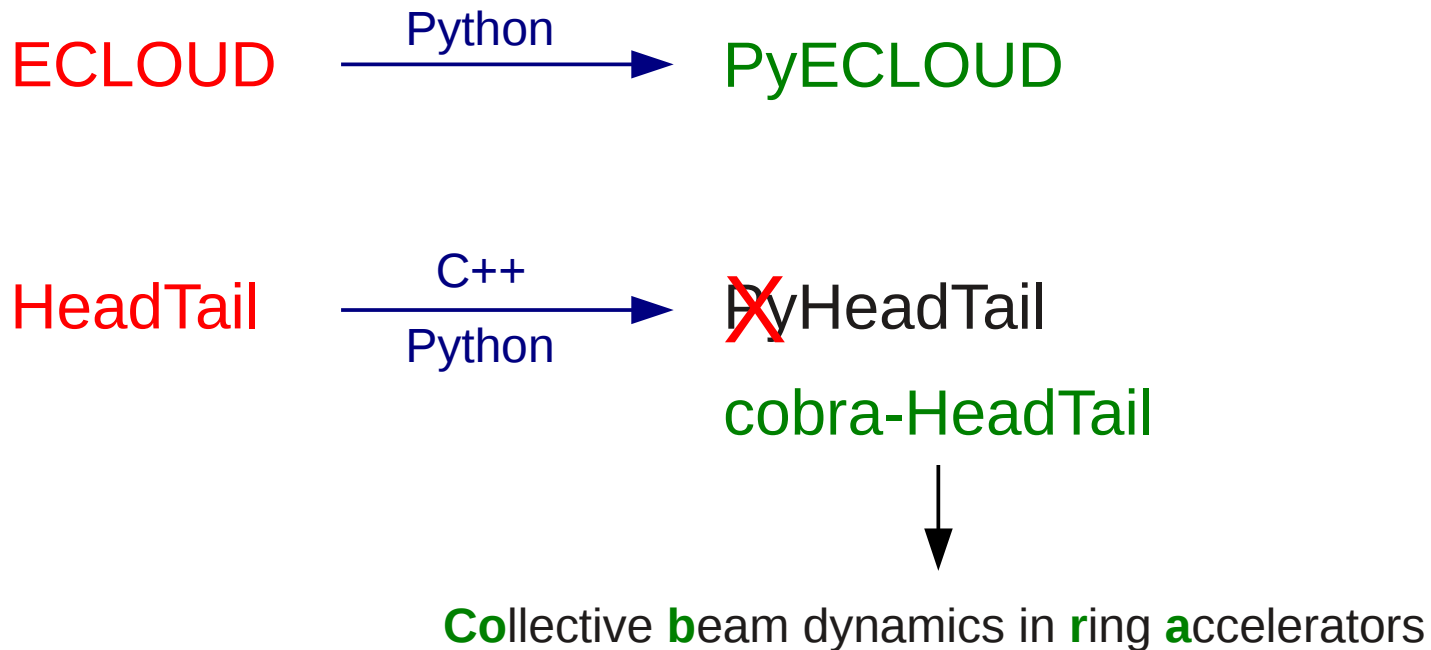


Written in C++ interfaced by Python:

- The whole effort for modularization started with the choice for C++
- C++ is fairly high level, generic and combines modularity and object-orientation with the high performance of a compiled language
- C++ lacks the flexibility provided by a script language
  - provide an interface to Python via boost libraries
  - we get the best of both worlds!

# What is cobra-HeadTail?

A little success story:



# What is cobra-HeadTail?

A little success story:

ECLLOUD  $\xrightarrow{\text{Python}}$  PyECLLOUD

HeadTail  $\xrightarrow[\text{Python}]{\text{C++}}$  ~~Py~~HeadTail  
cobra-HeadTail



Collective beam dynamics in ring accelerators  
&  
Analysis of cobra-HeadTail numerical simulation data

# What is cobra-HeadTail?

A little success story:

ECLLOUD  $\xrightarrow{\text{Python}}$  PyECLLOUD

HeadTail  $\xrightarrow[\text{Python}]{\text{C++}}$  ~~Py~~HeadTail  
cobra-HeadTail

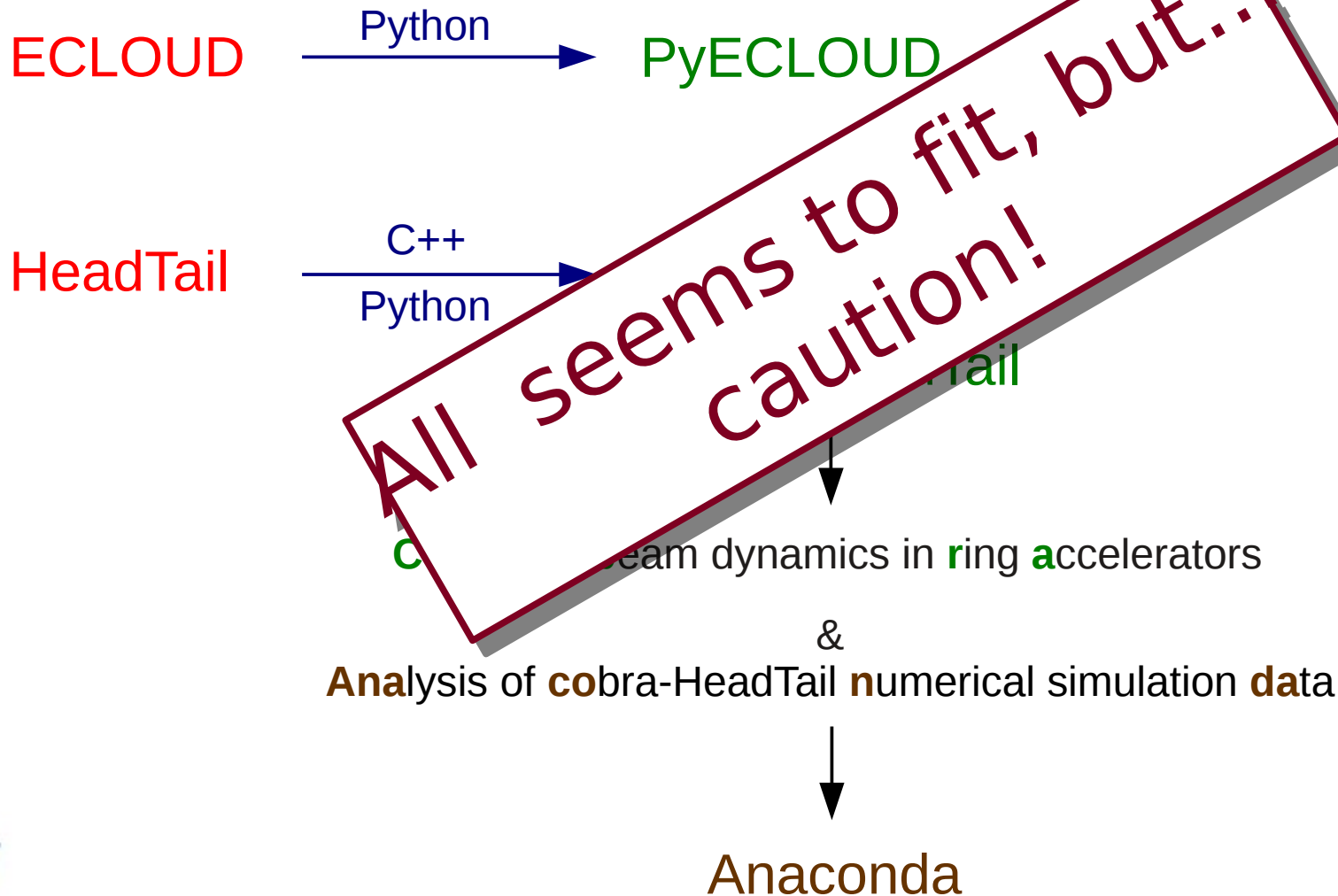
↓  
Collective beam dynamics in ring accelerators  
&  
Analysis of cobra-HeadTail numerical simulation data

↓  
Anaconda



# What is cobra-HeadTail?

A little success story:



# What is cobra-HeadTail?

## Cobra Programming Language

not to be mixed-up


[Downloads](#)
[Documentation](#)
[How To](#)
[Samples](#)
[Discussion](#)
[Wiki](#)
[Contact](#)

```
class SmallSample

  var _random = Random()

  def randomString(length as Int, alphabet as String) as String
    require
      length > 0
      alphabet <> ''
    ensure
      result.length == length
    test
      utils = SmallSample()
      assert utils.randomString(5, 'ab').length == 5
      s = utils.randomString(1000, 'a')
      for c in s, assert c == 'a'
    body
      sb = StringBuilder()
      for i in length
        c = alphabet[_random.next(alphabet.length)]
        sb.append(c)
      return sb.toString

  def main
    alphabet = 'abcdefghijklmnopqrstuvwxyz'
    for i in 10, print .randomString(10, alphabet)
```

Clean, high-level syntax

Static and dynamic binding

First class support for unit tests and contracts

Compiled performance with scripting conveniences

Lambdas and closures

Extensions and mixins

... and more



Download Cobra

### Recent News

2013-05-24 Cobra 0.9.4 has been released.

2012-11-26 Cobra 0.9.2 has been released with 30 refinements and fixes.



# What is cobra-HeadTail?

## Cobra Programming Language


[Downloads](#)
[Documentation](#)
[How To](#)
[Samples](#)
[Discussion](#)
[Wiki](#)
[Contact](#)


### Why Cobra?

(This article was translated to the [Serbo-Croatian](#) language by WHGeeks.)

There are plenty of object-oriented programming languages in existence including C#, Python and Ruby. So why use Cobra? This document addresses that question.

Right now, if you want software contracts in your language, how can you get them? The answer is to use Eiffel or D. What if you want static and dynamic binding? Use Objective-C or Boo. What if you want expressiveness and quick coding? Use Python. Ruby or Smalltalk. What if you want runtime performance? Use C#, Java, C++ etc. What if you want first class language support for unit tests? Use D.

But what if you want *all* of those? ... You can't get them! And that's frustrating because none of those productivity-boosting features are incompatible with each other. You shouldn't have to choose between C++'s speed, Python's expressiveness and Eiffel's contracts. There's no theoretical reason that you can't have it all. There's "just" a lot of work required to make it happen.

One way to characterize Cobra is with these high level points:

1. Quick, expressive coding
2. Fast execution
3. Static and dynamic binding
4. Language level support for quality

Cobra achieves 1 by following Python and Ruby (but not religiously). It achieves 2 by favoring static typing ("i = 5" means "i" is an integer and always will be) and leveraging .NET/Mono for machine code generation. It does 3 by using the .NET typing system at compile-time for static types, and using the .NET run-time for dynamic binding. It takes language features for 4 from multiple sources including Eiffel, Python and its own compile-time nil tracking.

There are additional refinements in Cobra such as defaulting to accurate decimal math and providing detailed postmortem exception reports.

So the "what's new" in Cobra is not the individual elements such as contracts, classes, etc. It's the combination of everything that goes into it.



# What is cobra-HeadTail?

## Cobra Programming Language


[Downloads](#)
[Documentation](#)
[How To](#)
[Samples](#)
[Discussion](#)
[Wiki](#)
[Contact](#)


### Why Cobra?

(This article was translated to the [Serbo-Croatian](#) language by [WHGeeks](#).)

There are plenty of object-oriented programming languages in existence including C#, Python and Ruby. So why use Cobra? This document addresses that question.

Right now, if you want software contracts in your language, how can you get them? The answer is to use Eiffel or D. What if you want static and dynamic binding? Use Objective-C or Boo. **What if you want expressiveness and quick coding? Use Python.** Ruby or Smalltalk. What if you want runtime performance? Use C#, Java, C++ etc. What if you want first class language support for unit tests? Use D.

But what if you want *all* of those? ... You can't get them! And that's frustrating because none of those productivity-boosting features are incompatible with each other. You shouldn't have to choose between C++'s speed, Python's expressiveness and Eiffel's contracts. There's no theoretical reason that you can't have it all. There's "just" a lot of work required to make it happen.

One way to characterize Cobra is with these high level points:

1. Quick, expressive coding
2. Fast execution
3. Static and dynamic binding
4. Language level support for quality

Cobra achieves 1 by following Python and Ruby (but not religiously). It achieves 2 by favoring static typing ("`i = 5`" means "`i`" is an integer and always will be) and leveraging .NET/Mono for machine code generation. It does 3 by using the .NET typing system at compile-time for static types, and using the .NET run-time for dynamic binding. It takes language features for 4 from multiple sources including Eiffel, Python and its own compile-time nil tracking.

There are additional refinements in Cobra such as defaulting to accurate decimal math and providing detailed postmortem exception reports.

So the "what's new" in Cobra is not the individual elements such as contracts, classes, etc. It's the combination of everything that goes into it.

**Actually, we are doing exactly that, but sticking to well-known Python as our scripting language... so maybe there is actually some truth in this disambiguation**





# Why cobra-HeadTail?

Why cobra-HeadTail? HeadTail works just fine!

# Why cobra-HeadTail?

Why cobra-HeadTail? HeadTail works just fine!

The reference:

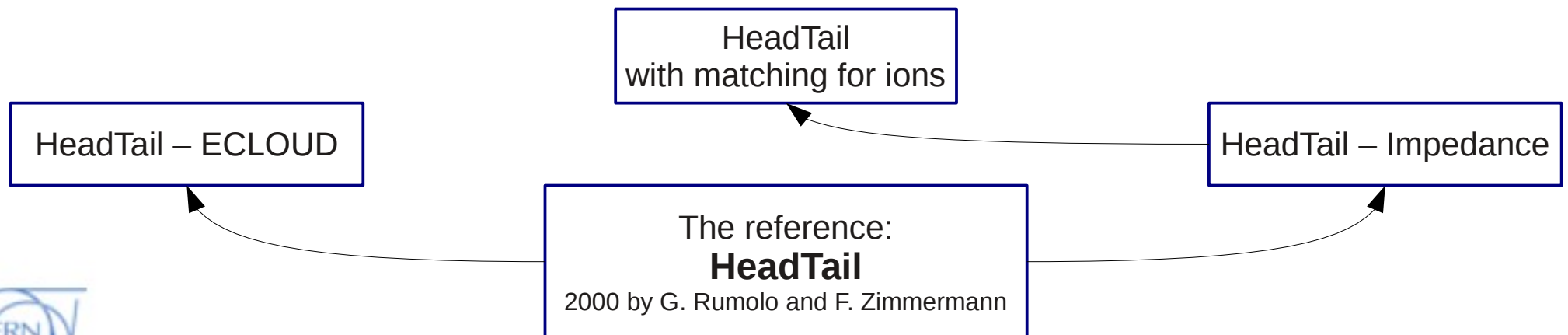
**HeadTail**

2000 by G. Rumolo and F. Zimmermann



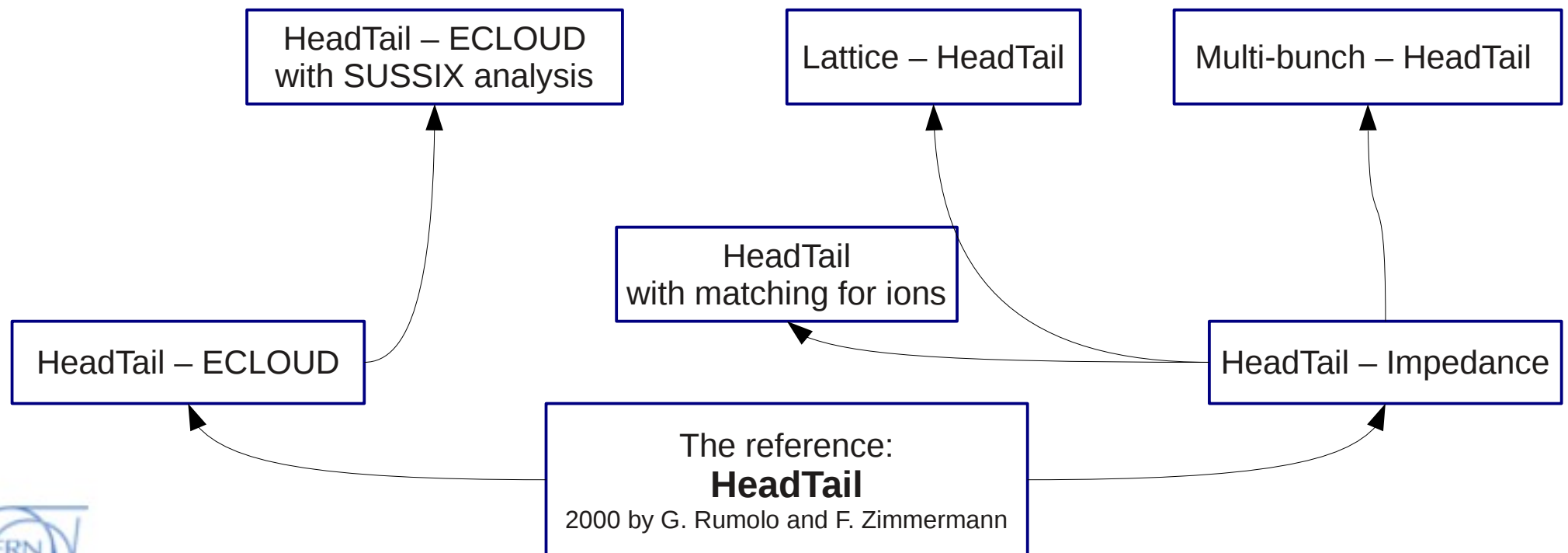
# Why cobra-HeadTail?

Why cobra-HeadTail? HeadTail works just fine!



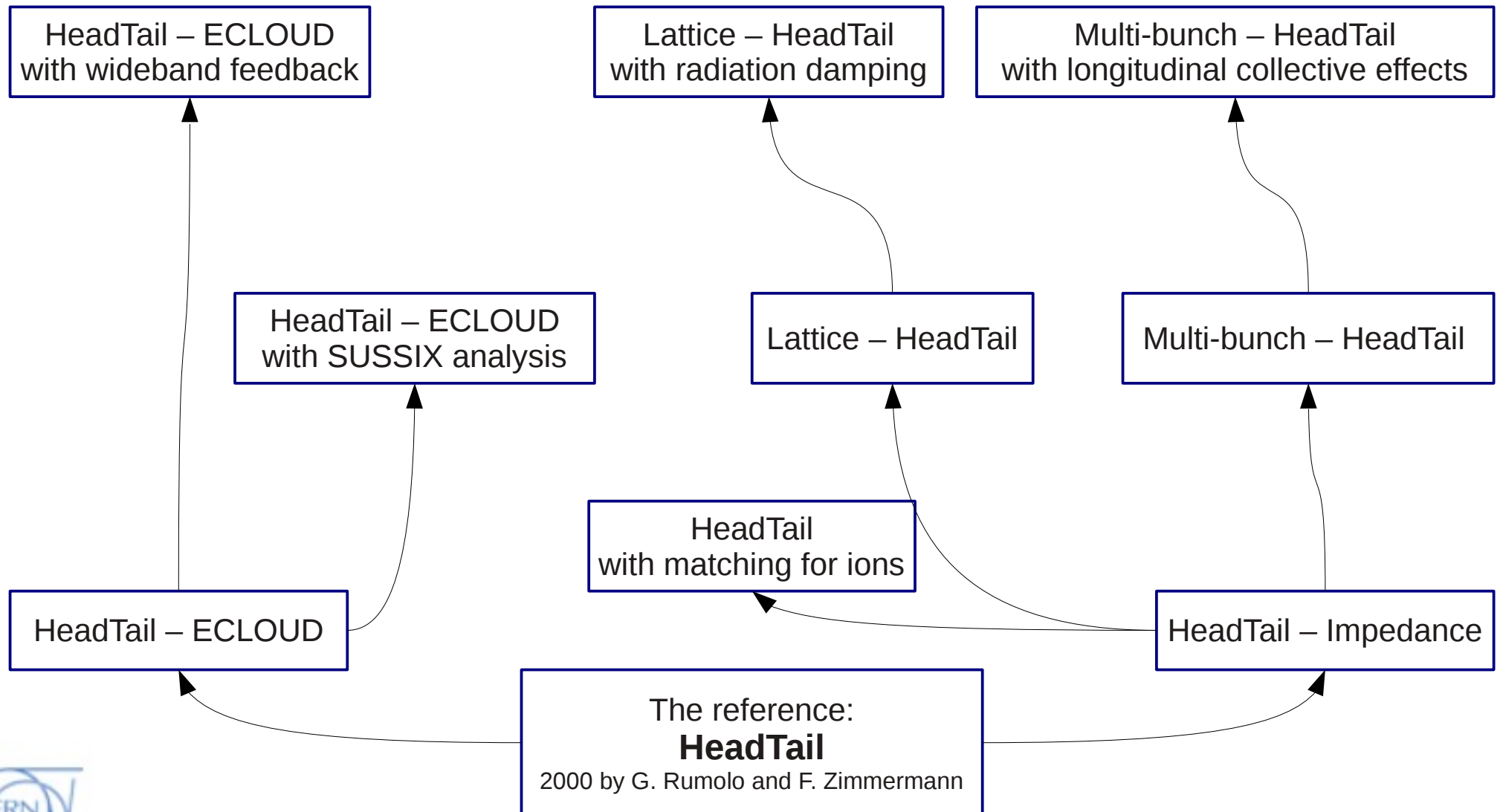
# Why cobra-HeadTail?

Why cobra-HeadTail? HeadTail works just fine!



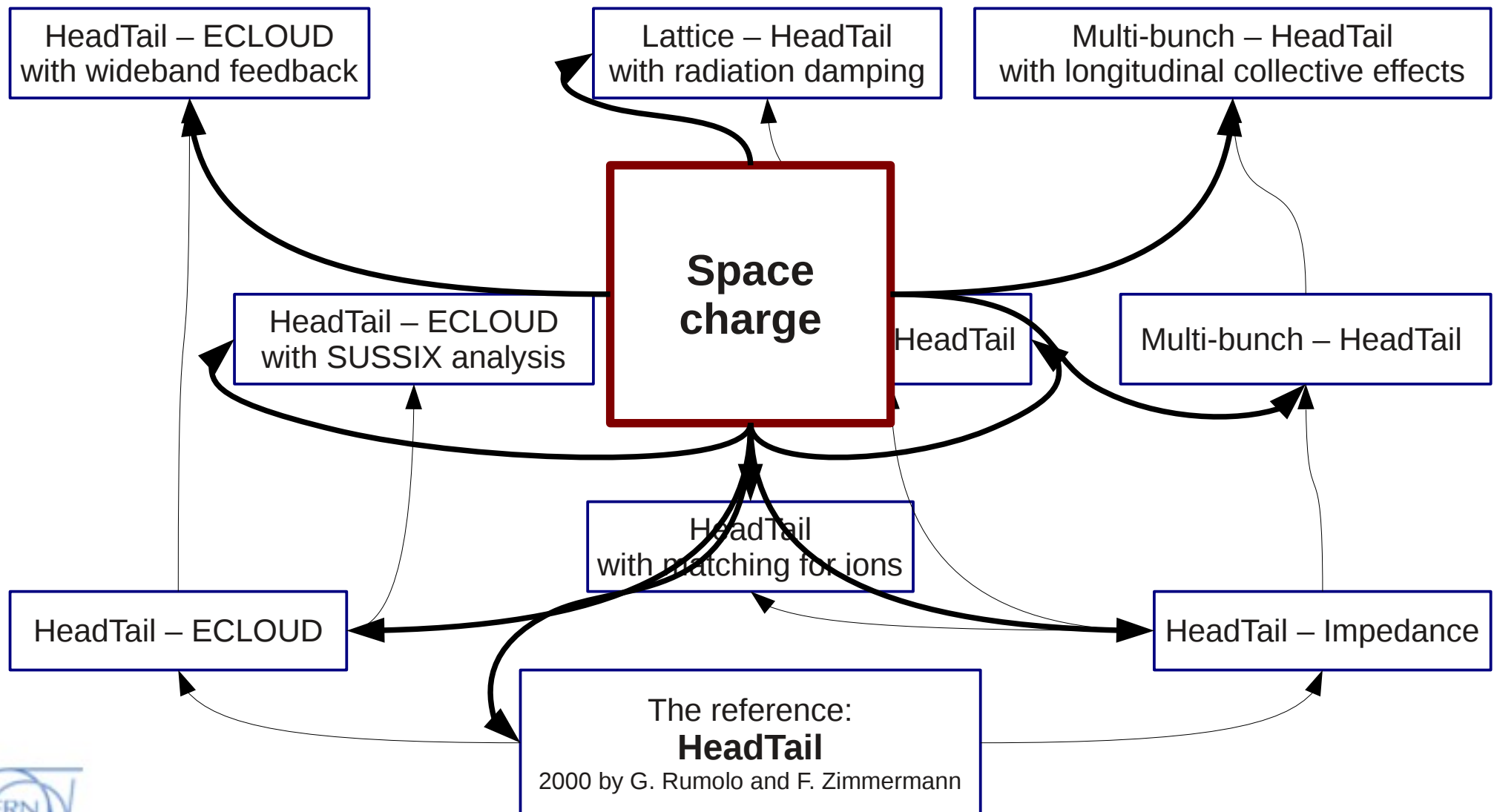
# Why cobra-HeadTail?

Why cobra-HeadTail? HeadTail works just fine!



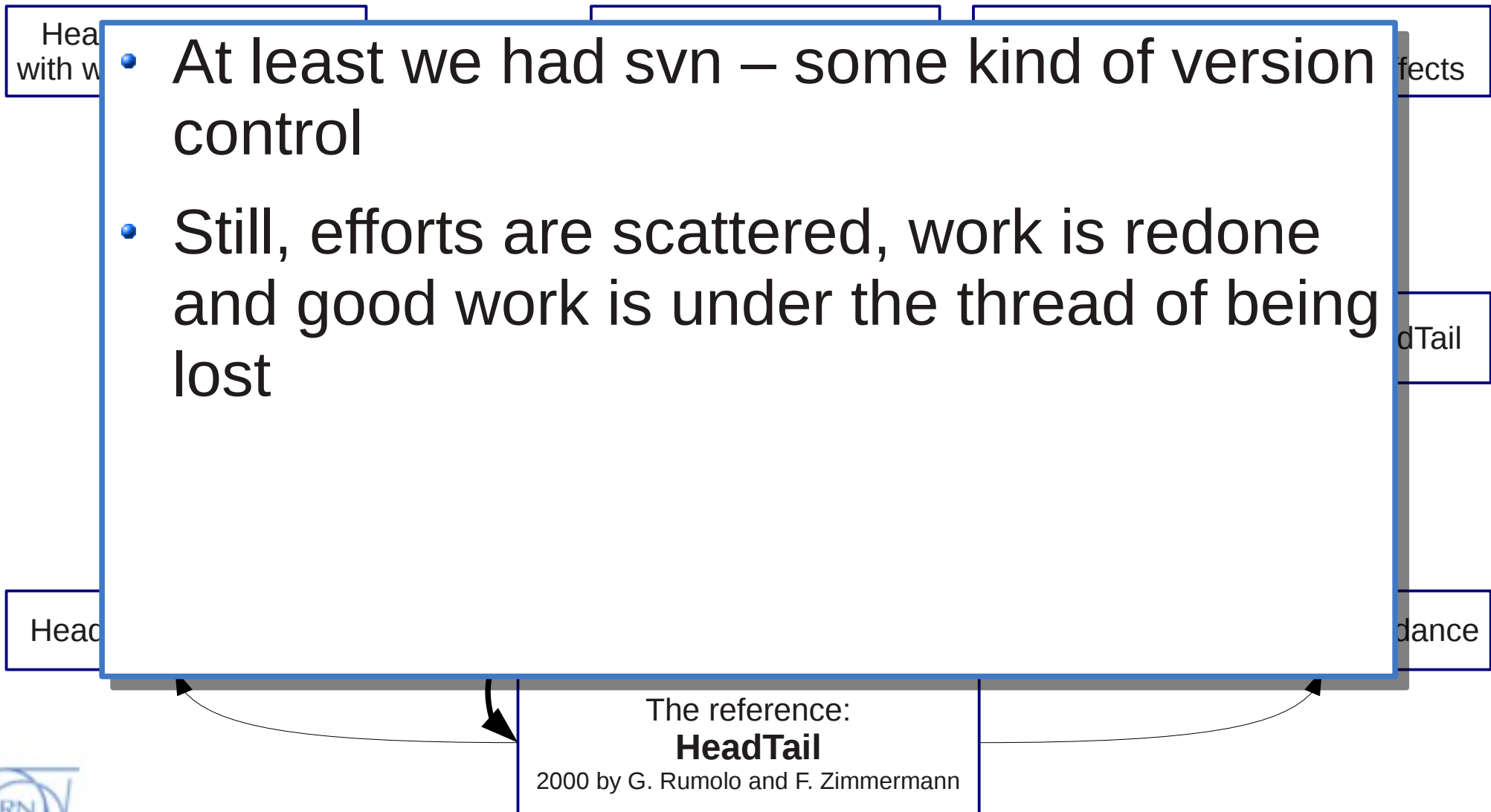
# Why cobra-HeadTail?

Why cobra-HeadTail? HeadTail works just fine!



# Why cobra-HeadTail?

Why cobra-HeadTail? HeadTail works just fine!



# Why cobra-HeadTail?

Possible reasons for scattered efforts:

- People like to understand what they write; a non-modular code is full of dependencies with global variables scattered all over the place
  - sometimes very hard to understand
  - Okay, I will now write my own version of the code that only I can understand!
- It is simple and quick (at least, it seems so at first) to hardcode some simple modifications for one specific purpose and add them within some switch statement (hoping it will not interfere with anything else...)





# Why cobra-HeadTail?

Possible reasons for scattered efforts:

- People like to understand what they write; a non-modular code is full of dependencies with global variables scattered all over the place
  - sometimes very hard to understand
  - Okay, I will now write my own version of the code that only I can understand!
- It is simple and quick (at least, it seems so at first) to hardcode some simple modifications for one specific purpose and add them within some switch statement (hoping it will not interfere with anything else...)

Possible solutions

- Extendible – Flexible – Manageable – Translatable
- How about having a basic platform to which anyone can contribute his own independent part of code which anyone can use just by somehow plugging it in?



# Why cobra-HeadTail?

Possible reasons for scattered efforts:

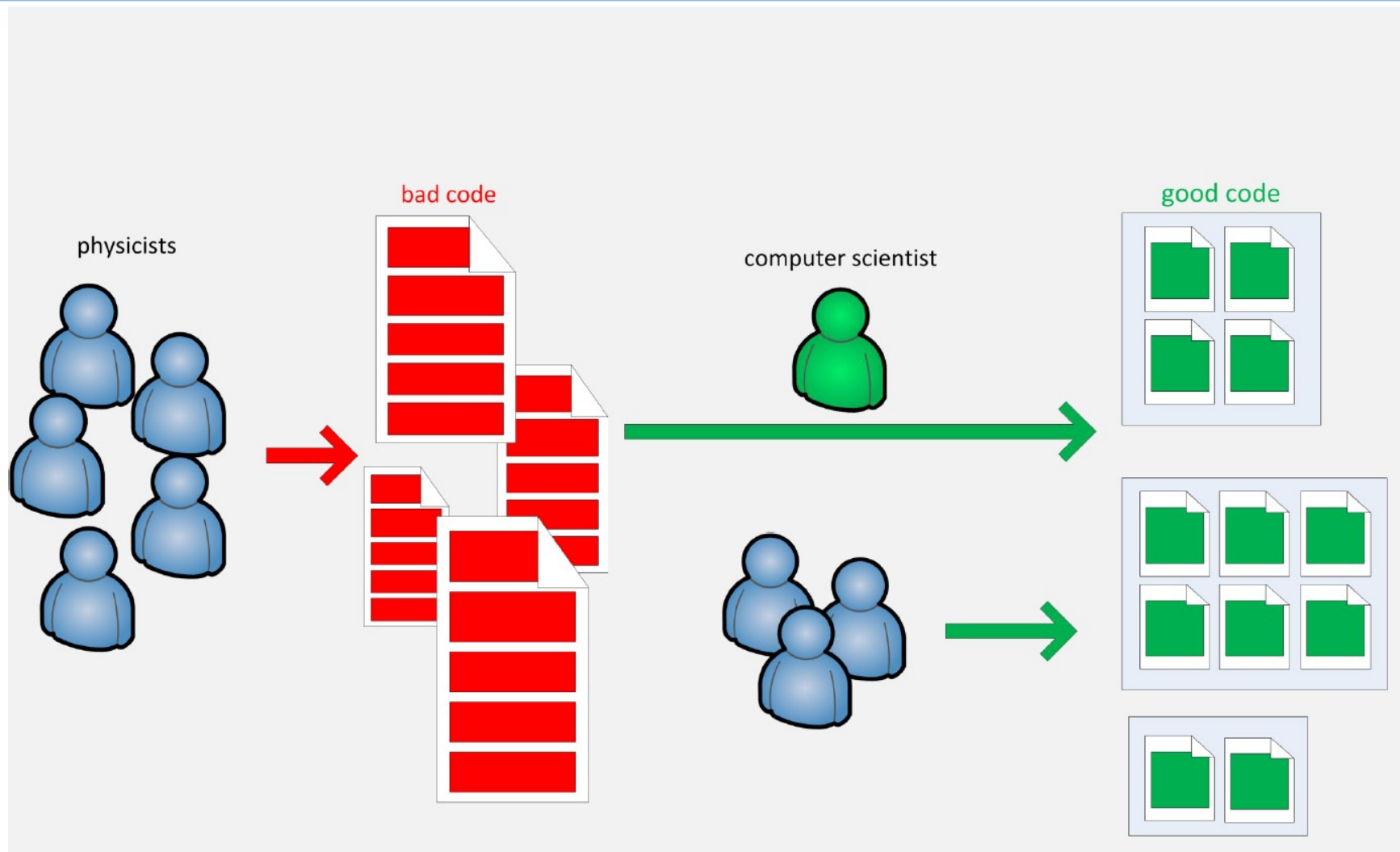
- People like to understand what they write; a non-modular code is full of dependencies with global variables scattered all over the place
  - sometimes very hard to understand
  - Okay, I will now write my own version of the code that only I can understand!
- It is simple and quick (at least, it seems so at first) to hardcode some simple modifications for one specific purpose and add them within some switch statement (hoping it will not interfere with anything else...)

Possible solutions

- Extendible – Flexible – Manageable – Translatable
- How about having a fully modular scriptable HeadTail with Python syntax written in a compiled language prepared for HPC?



# Also followed by other groups



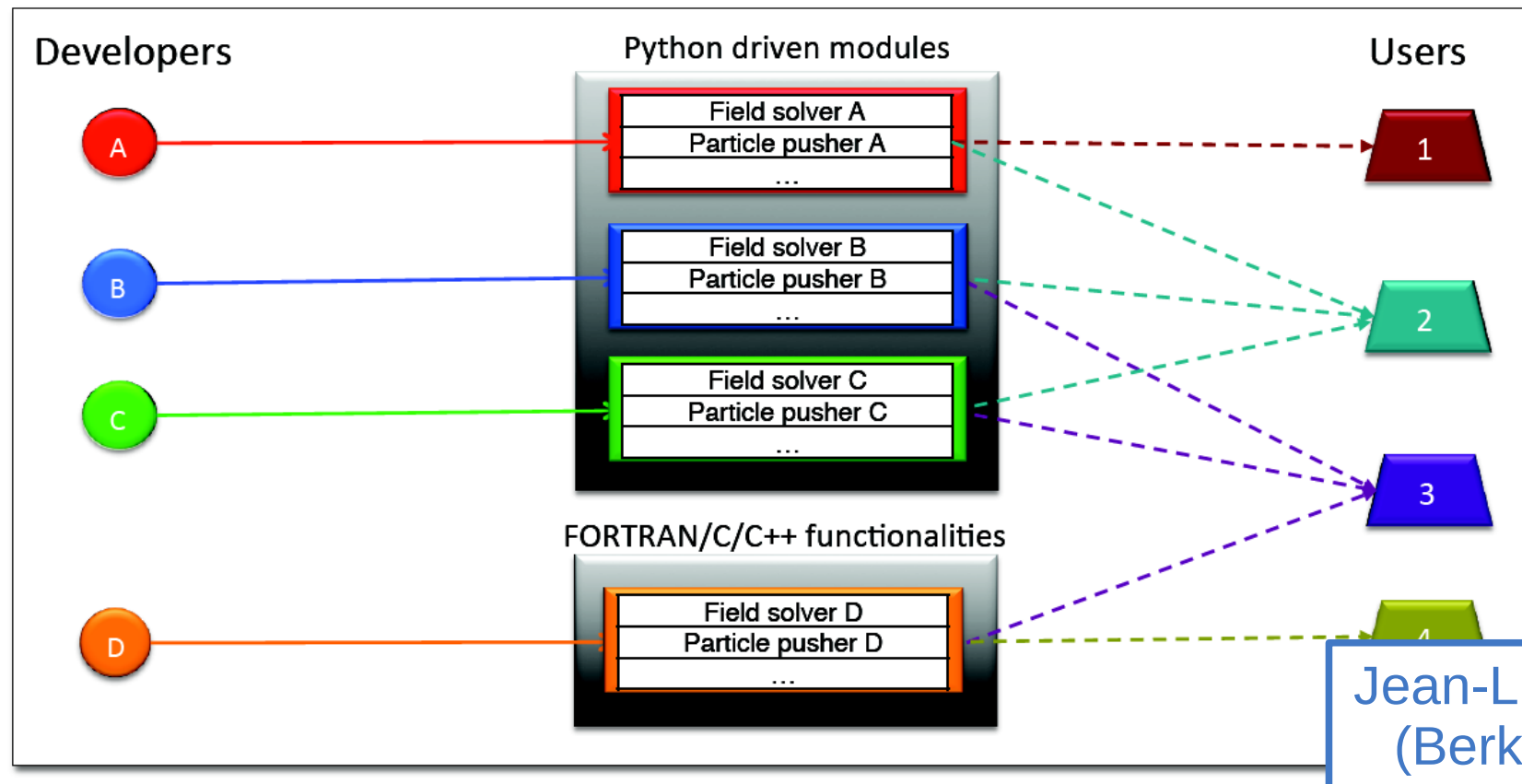
Victor Maier  
(CERN)

# Also followed by other groups

Further integration to increase capabilities, reduce duplication

With modular programming, we can share modules like Lego pieces.

Having Python based codes makes it especially easy!



# Model

# The HeadTail model

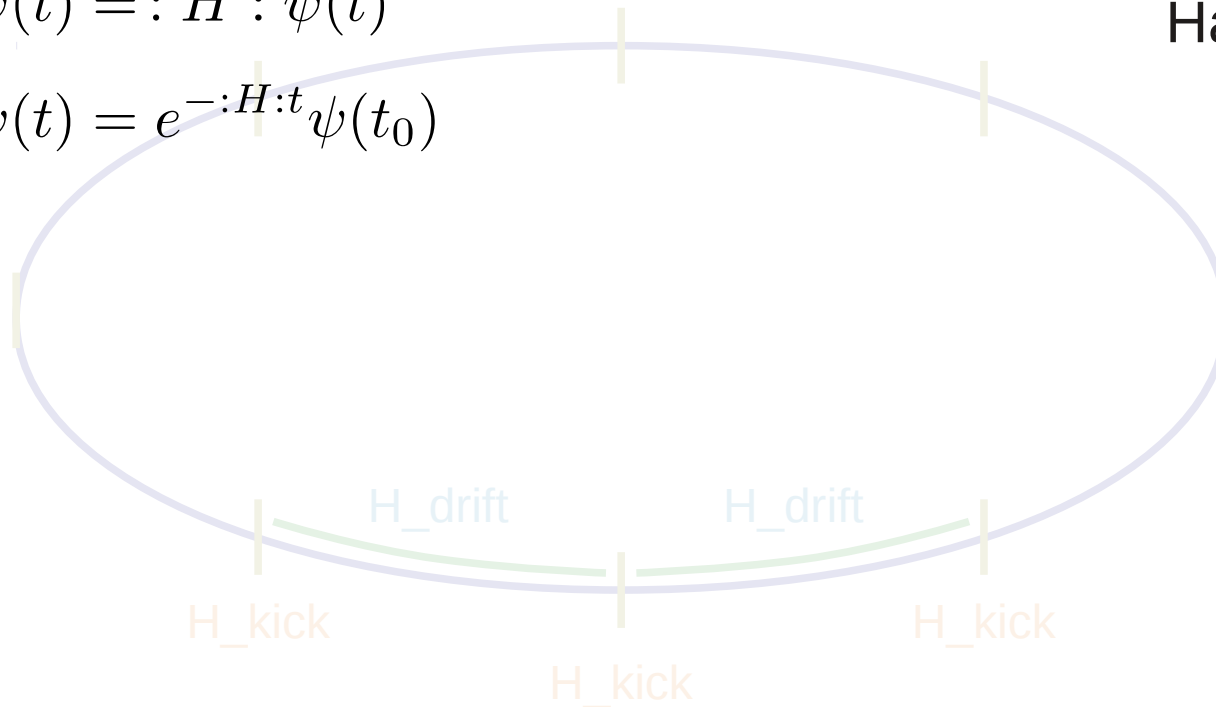
```
class Ensemble { std::vector }
```

$$\frac{d}{dt}\psi(t) = -[H, \psi(t)]$$

$$\frac{d}{dt}\psi(t) = :H: \psi(t)$$

$$\psi(t) = e^{-:H:t} \psi(t_0)$$

- Time evolution of a particle ensemble is generated via a Poisson bracket with the Hamiltonian



# The HeadTail model

```
class Ensemble { std::vector }
```

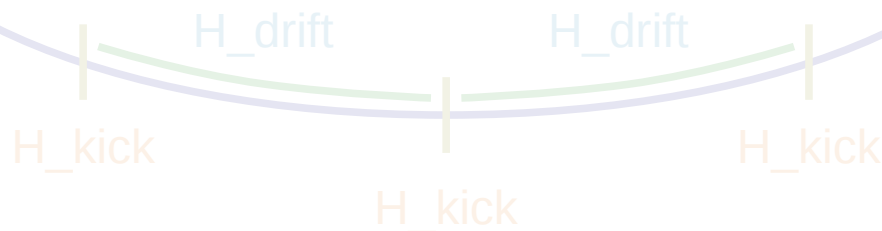
$$\frac{d}{dt}\psi(t) = -[H, \psi(t)]$$

$$\frac{d}{dt}\psi(t) = :H: \psi(t)$$

$$\psi(t) = e^{-:H:t} \psi(t_0)$$

$$H = \sum H_i$$

$$\mathcal{M} = e^{-:H_1:\frac{\Delta t}{2}} e^{-:H_2:\frac{\Delta t}{2}} \dots e^{-:H_2:\frac{\Delta t}{2}} e^{-:H_1:\frac{\Delta t}{2}}$$



- Time evolution of a particle ensemble is generated via a Poisson bracket with the Hamiltonian
- Several Hamiltonian are formally easily concatenated

# The HeadTail model

```
class Ensemble { std::vector }
```

$$\frac{d}{dt}\psi(t) = -[H, \psi(t)]$$

$$\frac{d}{dt}\psi(t) = :H: \psi(t)$$

$$\psi(t) = e^{-:H:t} \psi(t_0)$$

$$H = \sum H_i$$

$$\mathcal{M} = e^{-:H_1:\frac{\Delta t}{2}} e^{-:H_2:\frac{\Delta t}{2}} \dots e^{-:H_2:\frac{\Delta t}{2}} e^{-:H_1:\frac{\Delta t}{2}}$$

$$H = A(p) + V(q) = H_D + H_K$$

$$\mathcal{M} = e^{-:H_D:\frac{\Delta t}{2}} e^{-:H_K:\Delta t} e^{-:H_D:\frac{\Delta t}{2}}$$

- Time evolution of a particle ensemble is generated via a Poisson bracket with the Hamiltonian

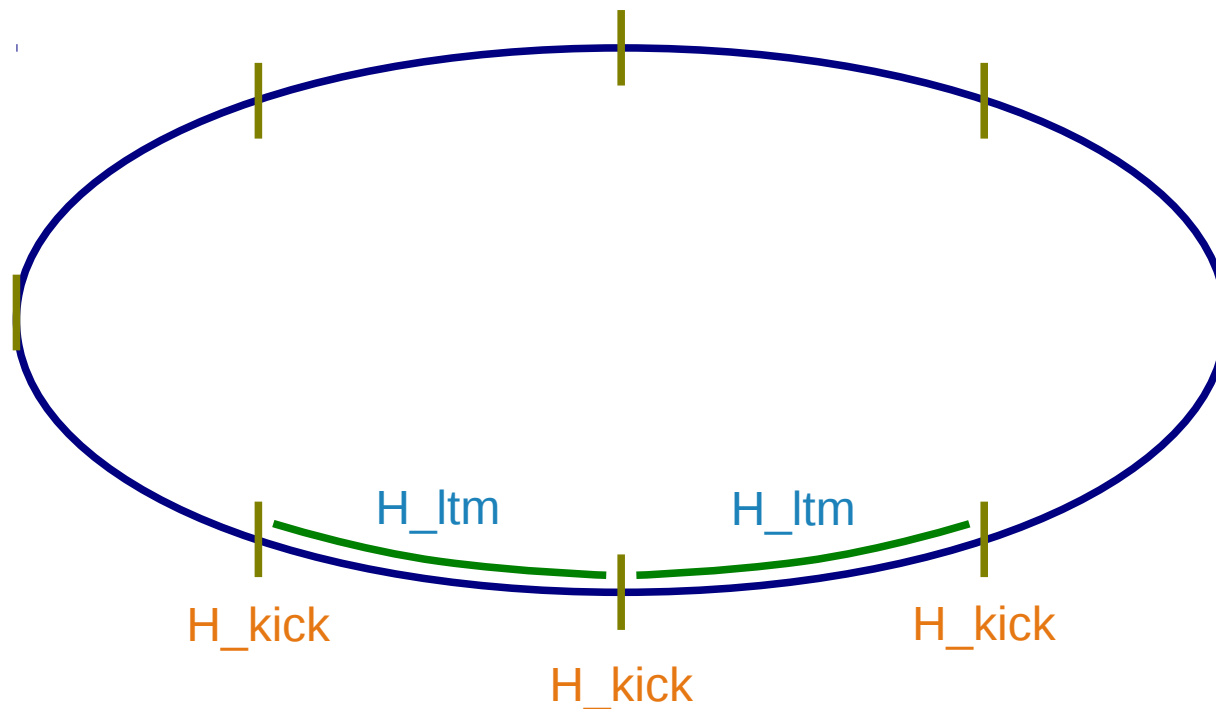
- Several Hamiltonian are formally easily concatenated

- Symmetrized split operator method immediately yields a second order symplectic tracking algorithm (Leap-frog)



# The HeadTail model – transverse

$$\mathcal{M} = e^{-:H_D:\frac{\Delta t}{2}} e^{-:H_K:\Delta t} e^{-:H_D:\frac{\Delta t}{2}}$$



- Transverse plane
- H\_ltm: linear transfer map
- H\_kick:
  - Beam-beam
  - Damper
  - Electron cloud
  - Multipolar wakefields
  - Space-charge
  - ...

# H\_drift or linear focusing channel

$$\begin{aligned}
 M(s_0|s_1) &= I \cos(\mu) + JA \sin(\mu) \\
 &= B(s_1) R(\mu) B^{-1}(s_0) \\
 &= \begin{pmatrix} \sqrt{\beta_1} & 0 \\ -\frac{\alpha_1}{\sqrt{\beta_1}} & \frac{1}{\sqrt{\beta_1}} \end{pmatrix} \begin{pmatrix} \cos(\mu) & \sin(\mu) \\ -\sin(\mu) & \cos(\mu) \end{pmatrix} \begin{pmatrix} \frac{1}{\sqrt{\beta_0}} & 0 \\ \frac{\alpha_0}{\sqrt{\beta_0}} & \sqrt{\beta_0} \end{pmatrix}
 \end{aligned}$$



Chromaticity, amplitude detuning, ...  
over one full turn

$$\Delta Q_x = Q'_x \delta + \alpha_{xx} J_x + \alpha_{xy} J_y + \dots$$

$$\Delta Q_y = Q'_y \delta + \alpha_{yx} J_x + \alpha_{yy} J_y + \dots$$

# One-turn-map

- Model allows simple implementation e.g. of lumped impedances

$$\begin{aligned}
 {}^{(1)} M(s_0|s_n) &= M(s_0|s_1) e^{:V_1:} M(s_1|s_2) e^{:V_2:} \dots M(s_{n-2}|s_{n-1}) e^{:V_{n-1}:} M(s_{n-1}|s_n) \\
 &= M(s_0|s_1) e^{:V_1:} M(s_0|s_1)^{-1} M(s_0|s_1) M(s_1|s_2) e^{:V_2:} \dots \\
 &\dots e^{:V_{n-1}:} M(s_0|s_{n-1})^{-1} M(s_0|s_{n-1}) M(s_{n-1}|s_n) \\
 &= e^{:M(s_0|s_1) V_1:} e^{:M(s_0|s_2) V_2:} \dots e^{:M(s_0|s_{n-1}) V_{n-1}:} M(s_0|s_n) \\
 &= B_0^{-1} B_0 e^{:M(s_0|s_1) V_1:} B_0^{-1} B_0 e^{:M(s_0|s_2) V_2:} \dots e^{:M(s_0|s_{n-1}) V_{n-1}:} B_0^{-1} R(s_0|s_n) B_n \\
 &= B_0^{-1} e^{:\hat{V}_1:} e^{:\hat{V}_2:} \dots e^{:\hat{V}_{n-1}:} e^{:\hat{V}_n:} R(s_0|s_n) B_n
 \end{aligned}$$

$$\hat{V}_i = R(s_0|s_i) B_i V_i$$

$$A:f:A^{-1}g = A[f, A^{-1}g] = [AF, g] =: Af : g$$

(1): J. Bengtsson, SLS Note 9/97, Paul Scherrer Institut (PSI), Villigen, Schweiz

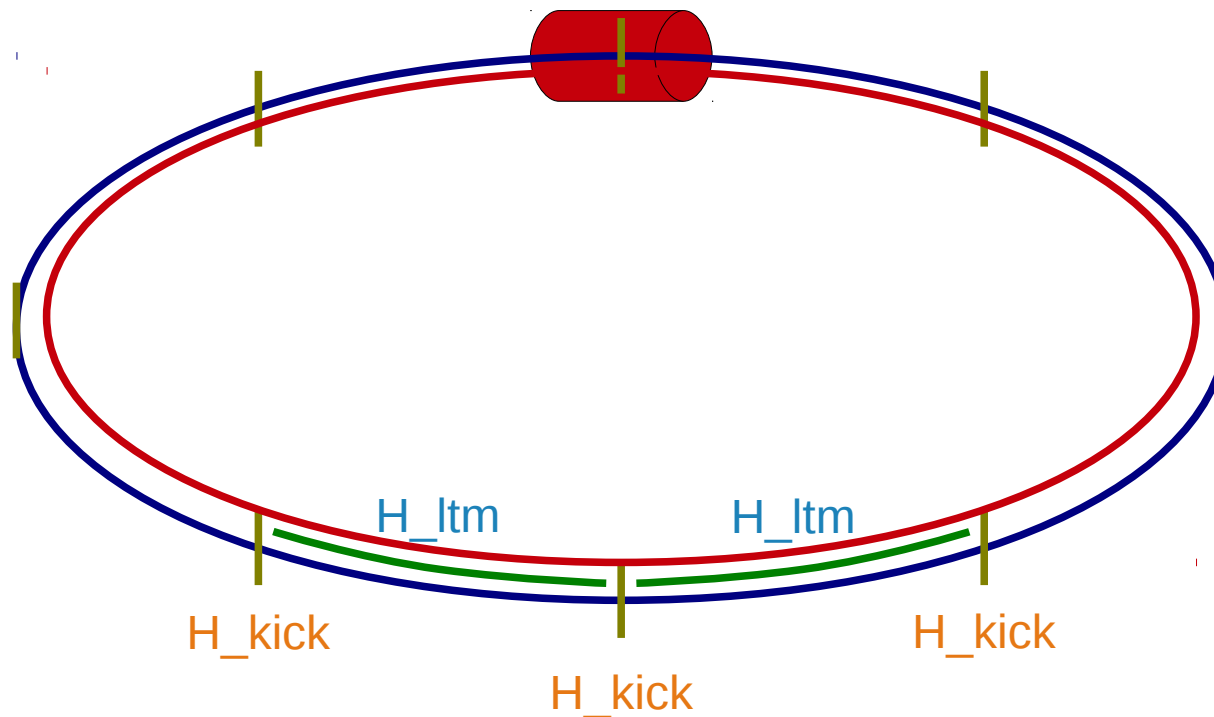


# The HeadTail model – longitudinal

RK4: classical Runge-Kutta

$$\mathcal{M} = e^{-:H_D:\frac{\Delta t}{2}} e^{-:H_K:\Delta t} e^{-:H_D:\frac{\Delta t}{2}} \quad (*)$$

$$\mathcal{M} = e^{d_1:H_D:\Delta t} e^{c_1:H_K:\Delta t} e^{d_2:H_D:\Delta t} e^{c_2:H_K:\Delta t} e^{d_2:H_D:\Delta t} e^{c_1:H_K:\Delta t} e^{d_1:H_D:\Delta t} \quad (**)$$



- Longitudinal plane:
  - (\*)  
2nd order symplectic via split operators - Leap-frog
  - (\*\*)  
4th order symplectic via split operators (Ron Ruth, SLAC)

# HeadTail split

$$H_{\text{transverse}} = \frac{p_x^2}{2} + K_x(s)^2 \frac{x^2}{2} + \frac{p_y^2}{2} + K_y(s)^2 \frac{y^2}{2}$$

$$H_{\text{longitudinal}} = -\frac{1}{2}\eta\delta^2 + \frac{Q_s^2}{\eta h^2} \left( 1 - \cos \left( \frac{h}{R}\sigma + \varphi_s \right) \right) + \frac{Q_s^2}{\eta h R} \sin(\varphi_s)\sigma$$

$$H_{\text{impedance}} = -\frac{r_0}{T_0\gamma c} \sum_{m,n} \left( \frac{x^n}{n!} \int \rho_m(\sigma') W_{mn}(\sigma - \sigma') d\sigma' \right)$$

$$H_{\text{feedback}} = -\frac{r_0}{T_0\gamma c} x \sum_{k=1}^{\text{N-taps}} D(s - kC) K(s)$$

beam dipolar moment

$$H_{\text{beam-beam e-cloud space-charge}} = -\frac{r_0}{T_0\gamma c} \int \rho(x', y')_{\text{beam-beam e-cloud space-charge}} G(x - x', y - y') dx' dy'$$

# HeadTail split

$$H_{\text{transverse}} = \frac{p_x^2}{2} + K_x(s)^2 \frac{x^2}{2} + \frac{p_y^2}{2} + K_y(s)^2 \frac{y^2}{2}$$

$$H_{\text{longitudinal}} = -\frac{1}{2}\eta\delta^2 + \frac{Q_s^2}{\eta h^2} \left( 1 - \cos\left(\frac{h}{R}\sigma + \varphi_s\right) \right) + \frac{Q_s^2}{\eta h R} \sin(\varphi_s)\sigma$$

$$H_{\text{impedance}} = -\frac{r_0}{T_0\gamma c} \sum_{m,n} \left( \frac{x^n}{n!} \int \rho_m(\sigma') W_{mn}(\sigma - \sigma') d\sigma' \right)$$

$$H_{\text{feedback}} = -\frac{r_0}{T_0\gamma c} x \sum_{k=1}^{\text{N-taps}} D(s - kC) K(s)$$

beam dipolar moment

$$H_{\text{beam-beam e-cloud space-charge}} = -\frac{r_0}{T_0\gamma c} \int \rho(x', y') G(x - x', y - y') dx' dy'$$

beam-beam e-cloud space-charge

external sources – self-consistent solution

# HeadTail split

$$\begin{aligned}
 & p_x^2 + \frac{1}{2} x^2 + p_y^2 + \frac{1}{2} y^2 \\
 & H_{\text{SC}} = T_1 + \int d^3x \rho_1 V_1 + \int d^3x \rho_2 V_1 + \frac{1}{4} \int d^3x F_{\mu\nu;1} F^{\mu\nu;1} \\
 & \quad + T_2 + \int d^3x \rho_2 V_2 + \int d^3x \rho_1 V_2 + \frac{1}{4} \int d^3x F_{\mu\nu;2} F^{\mu\nu;2}
 \end{aligned}$$

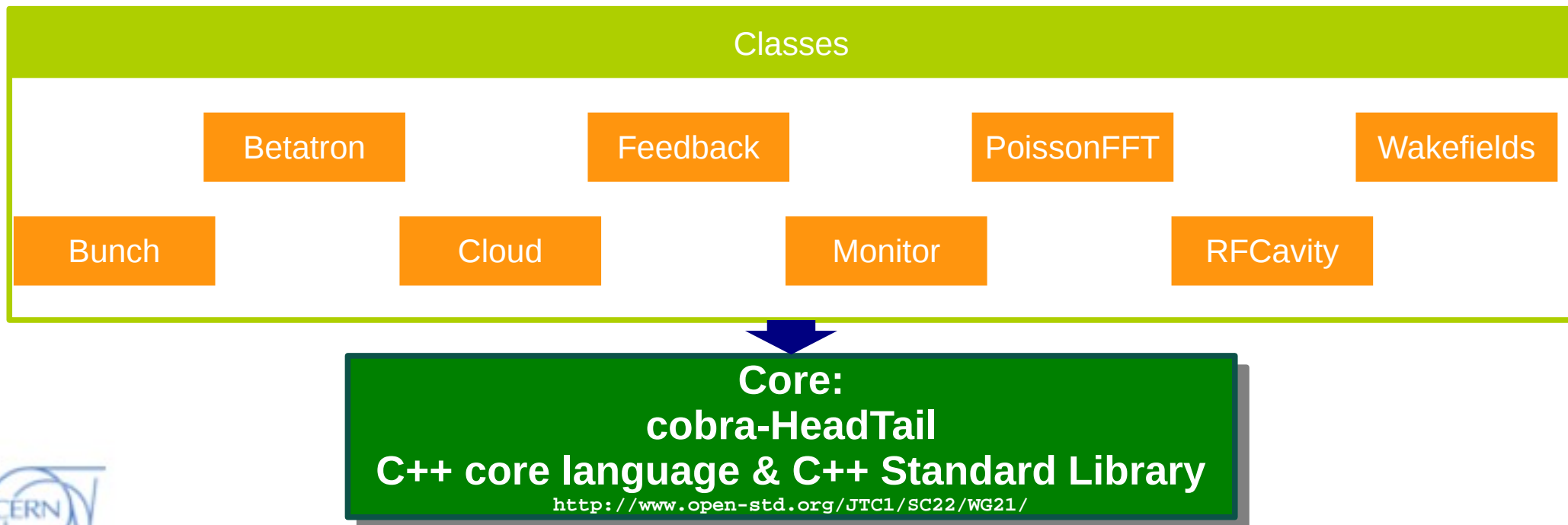
$H$   $\sigma$   
 $H$   
 e-cloud space-charge  $10^7 \text{ C J}$  e-cloud space-charge

external sources – self-consistent solution

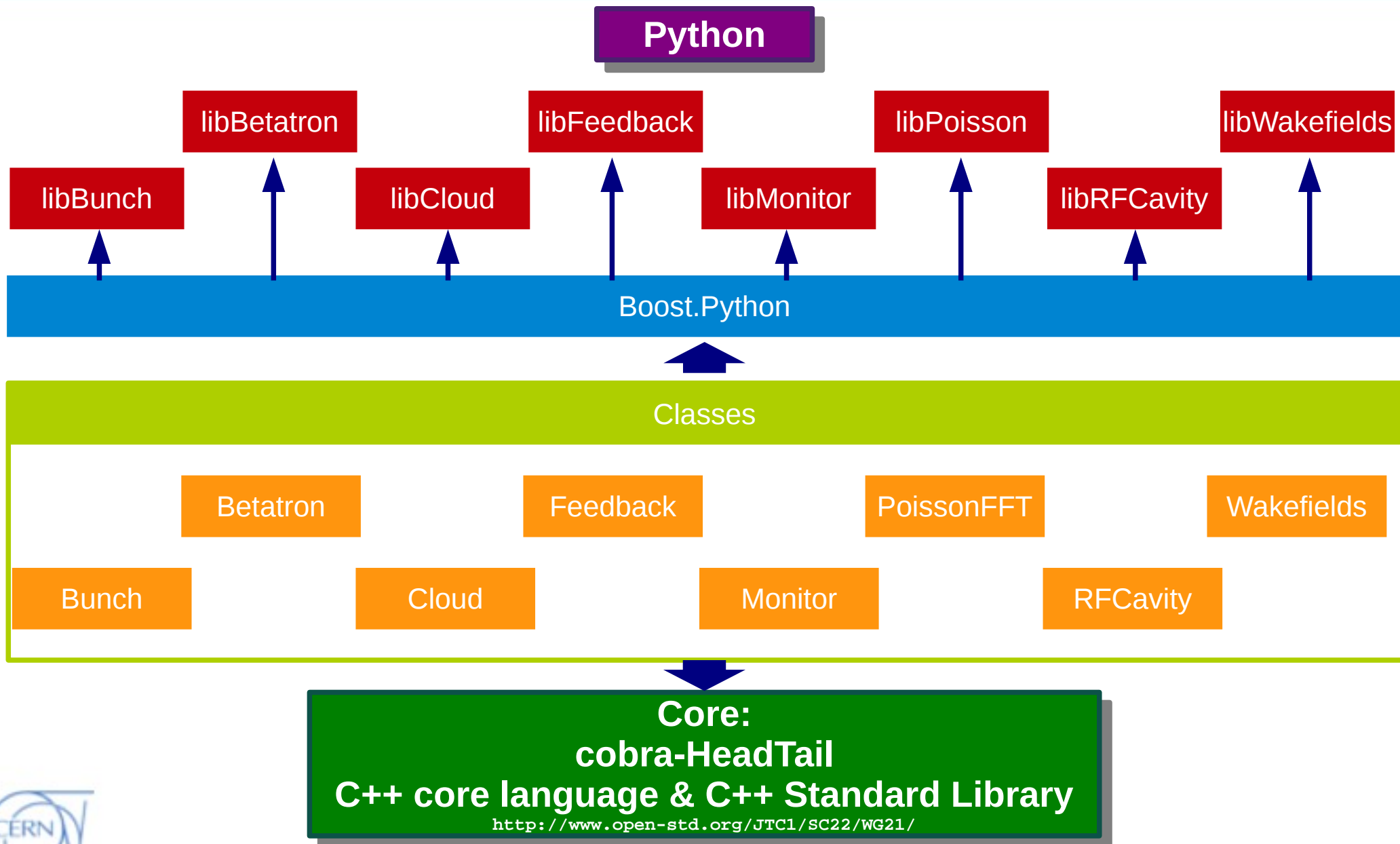
# Design



# cobra-HeadTail design

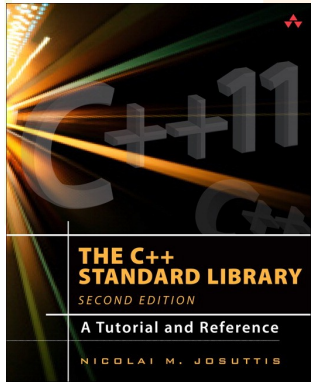
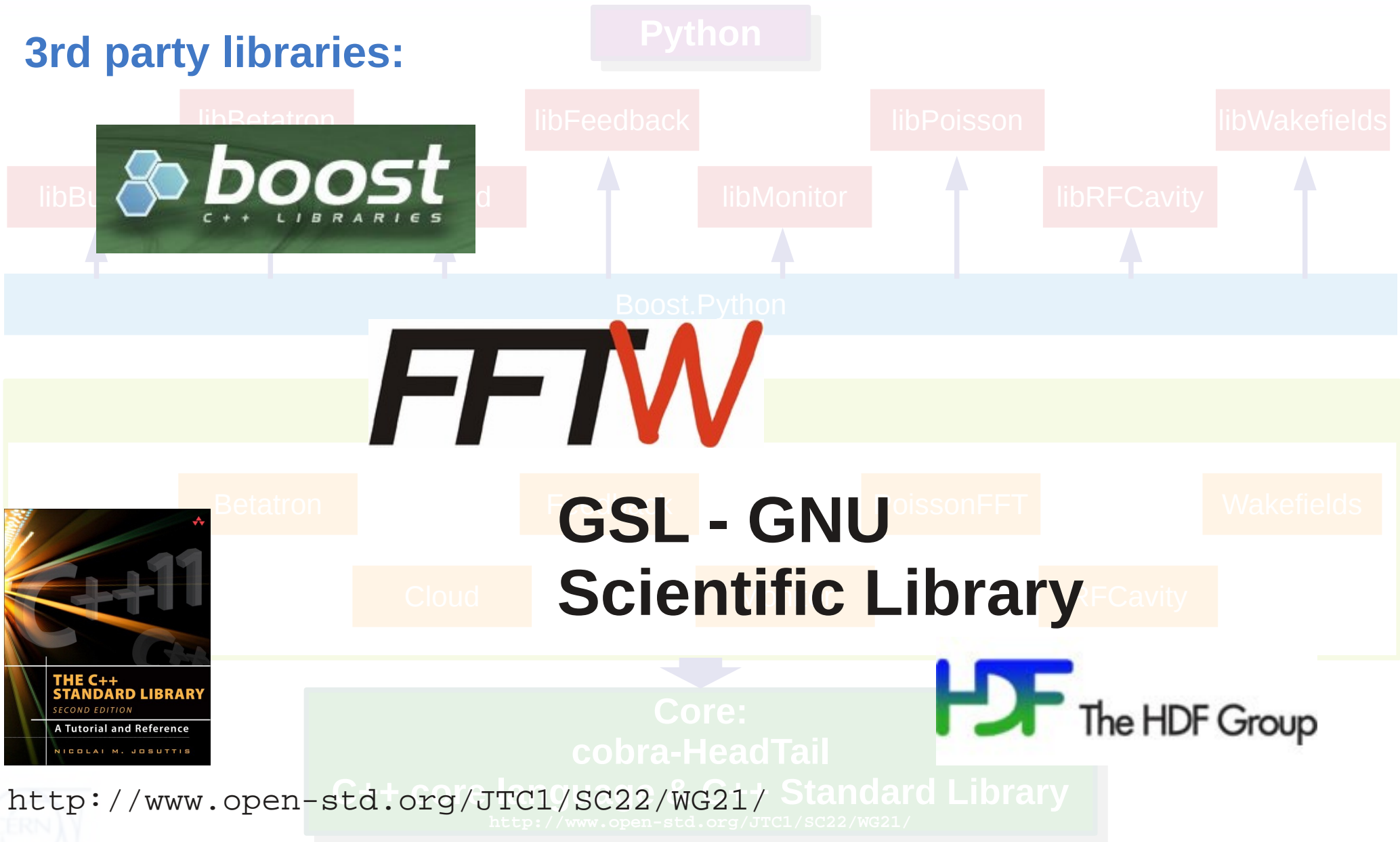


# cobra-HeadTail design



# cobra-HeadTail design

3rd party libraries:



<http://www.open-std.org/JTC1/SC22/WG21/>

# Documentation

We should try to write the Documentation along with the code

- Class hierarchy

The screenshot shows a web browser window titled "cobra-HeadTail: Class Hierarchy - Mozilla Firefox". The address bar shows the file path: "file:///home/kili/workspace/cobra-docs/html/hierarchy.html". The browser's toolbar includes "Meistbesucht", "openSUSE", "Getting Started", "Latest Headlines", "Mozilla Firefox", "SLAC Research Lib...", "SLAC National Accel...", and "SLAC Café - Home".

The main content area features a logo for "cobra-HeadTail" which is a green cobra with red 'X' marks on its head and tail, and the text "cobra-HeadTail A Python interfaced HeadTail Library". Below the logo is a navigation menu with tabs for "Main Page", "Classes", "Files", "Class List", "Class Index", "Class Hierarchy", and "Class Members". The "Class Hierarchy" tab is active.

The "Class Hierarchy" section is titled "Class Hierarchy" and includes the text: "This inheritance list is sorted roughly, but not completely, alphabetically:". Below this is a list of classes and their descriptions:

- AbstractCavity
  - CSCavity
  - RFcavity\_RK4
  - RFcavity\_symplectic2
  - RFcavity\_symplectic4
- Betatron: Class for creation and management of the betatron transport matrices
- Cavity: Class for creation and management of the synchrotron transport matrices
- Cloud: Class for creation and storage of electron clouds
- Ensemble: Class for general representation of particle ensembles / multi-particle states
  - Bunch: Class for creation and storage of particle bunches
- Feedback: Class for modeling a generic feedback system
- Kicker
- Betatron::Map: Single drift map A single drift map extracted from the list of drift maps created by the parent class. Accessed via the operator []
- Monitor: Class for management of the HDF5 file i/o
- MonitorBase
  - BunchMonitor
  - FieldMonitor
  - ParticleMonitor
  - SliceMonitor
- Pickup
- PoissonBase: Base class for creation and management of Poisson solvers
  - PoissonFFT: Class for management of the integrated Green's function FFT Poisson solver
- Slices::Slice
- Slices
  - Bunch: Class for creation and storage of particle bunches
- Wakefields: Class for creation and management of wak-fields from impedance sources

Generated by [doxygen](#) 1.8.1



# Documentation

The screenshot displays the 'cobra-HeadTail' class reference documentation in a Mozilla Firefox browser. The page title is 'cobra-HeadTail: Betatron Class Reference - Mozilla Firefox'. The browser's address bar shows the file path: `file:///home/kli/workspace/cobra-docs/html/classBetatron.html`. The page content includes a logo of a green snake with red 'X' marks, the text 'cobra-HeadTail A Python interfaced HeadTail Library', and a navigation menu with tabs for 'Main Page', 'Classes', and 'Files'. The 'Classes' tab is active, showing a 'Class List' and 'Class Index'. The main content area is titled 'Betatron Class Reference' and contains the following sections:

- Class for creation and management of the betatron transport matrices. More...**
- `#include <betatron.h>`
- List of all members.
- Classes**
  - `class Map`: Single drift map A single drift map extracted from the list of drift maps created by the parent class. Accessed via the operator []. More...
- Public Member Functions**
  - `Betatron (list s, list alpha_x, list beta_x, list D_x, list alpha_y, list beta_y, list D_y, double Q_x, double xi_x, double app_x, double Q_y, double xi_y, double app_y)`: Constructor to create a list of drift maps for linear periodic systems.
  - `Betatron (double C, double L, double Qx, double xi_x, double app_x, double Qy, double xi_y, double app_y, double beta_x, double beta_y)`: Constructor to create a drift map in smooth approximation for a ring with equally spaced elements.
  - `~Betatron ()`
  - `size_t get_npoints ()`: Get number of sections along the ring.
  - `Map operator[] (const int i_point)`: Operator to extract a single drift map from the list of drift maps created by the parent class.
- Public Attributes**
  - `class Betatron::Map M`
  - Betatron quantities**
  - Vectors of betatron quantities along the ring*
  - `std::vector< double > s`
  - `std::vector< double > alpha_x`
  - `std::vector< double > beta_x`
  - `std::vector< double > mu_x`
  - `std::vector< double > D_x`
  - `std::vector< double > alpha_y`
  - `std::vector< double > beta_y`

We should try to write the Documentation along with the code

- Class hierarchy

- Class members overview



# Documentation

Detailed Description

Class for creation and management of the betatron transport matrices.

**Author:**  
Kevin Li

**Date:**  
October 2012

**Copyright:**  
CERN

Constructor & Destructor Documentation

```
Betatron::Betatron ( list s,
                    list alpha_x,
                    list beta_x,
                    list D_x,
                    list alpha_y,
                    list beta_y,
                    list D_y,
                    double Q_x,
                    double xi_x,
                    double app_x,
                    double Q_y,
                    double xi_y,
                    double app_y
                    )
```

Constructor to create a list of drift maps for linear periodic systems.

**Parameters:**

- s** List of positions of elements along the ring in m, starting from the first position  $\neq 0$  and ending at the ring circumference.
- alpha\_x** List of Courant-Snyder parameter  $\alpha_x$  at corresponding locations s
- beta\_x** List of horizontal beta-function in m at corresponding locations s
- D\_x** List of horizontal dispersion in m at corresponding locations s
- alpha\_y** List of Courant-Snyder parameter  $\alpha_y$  at corresponding locations s
- beta\_y** List of vertical beta-function in m at corresponding locations s
- D\_y** List of vertical dispersion in m at corresponding locations s
- Q\_x** Horizontal tune
- xi\_x** Horizontal chromaticity  $\frac{dQ_x}{d(dp/p0)}$
- app\_x** Horizontal detuning parameter
- Q\_y** Vertical tune
- xi\_y** Vertical chromaticity  $\frac{dQ_y}{d(dp/p0)}$
- app\_y** Vertical detuning parameter

**Returns:**  
List of drift maps

We should try to write the Documentation along with the code

- Class hierarchy

- Class members overview

- Class member documentation



# cobra-HeadTail design

- **Extendible:**

Write a new class respecting the interface conventions and export via boost.python – or – write it all directly as a Python module which naturally merges into cobra-HeadTail

- **Flexible:**

Write your own personal main function using all the modules provided by cobra-HeadTail and write it in Python!

- **Manageable:**

The code is fully modular, dependencies are minimized. Each module author can be a module keeper of his own part of the code without breaking the global structure of the program or other independent modules

- **Translatable:**

Python modules can be interfaced from a vast amount of languages → Fortran, C/C++, Matlab

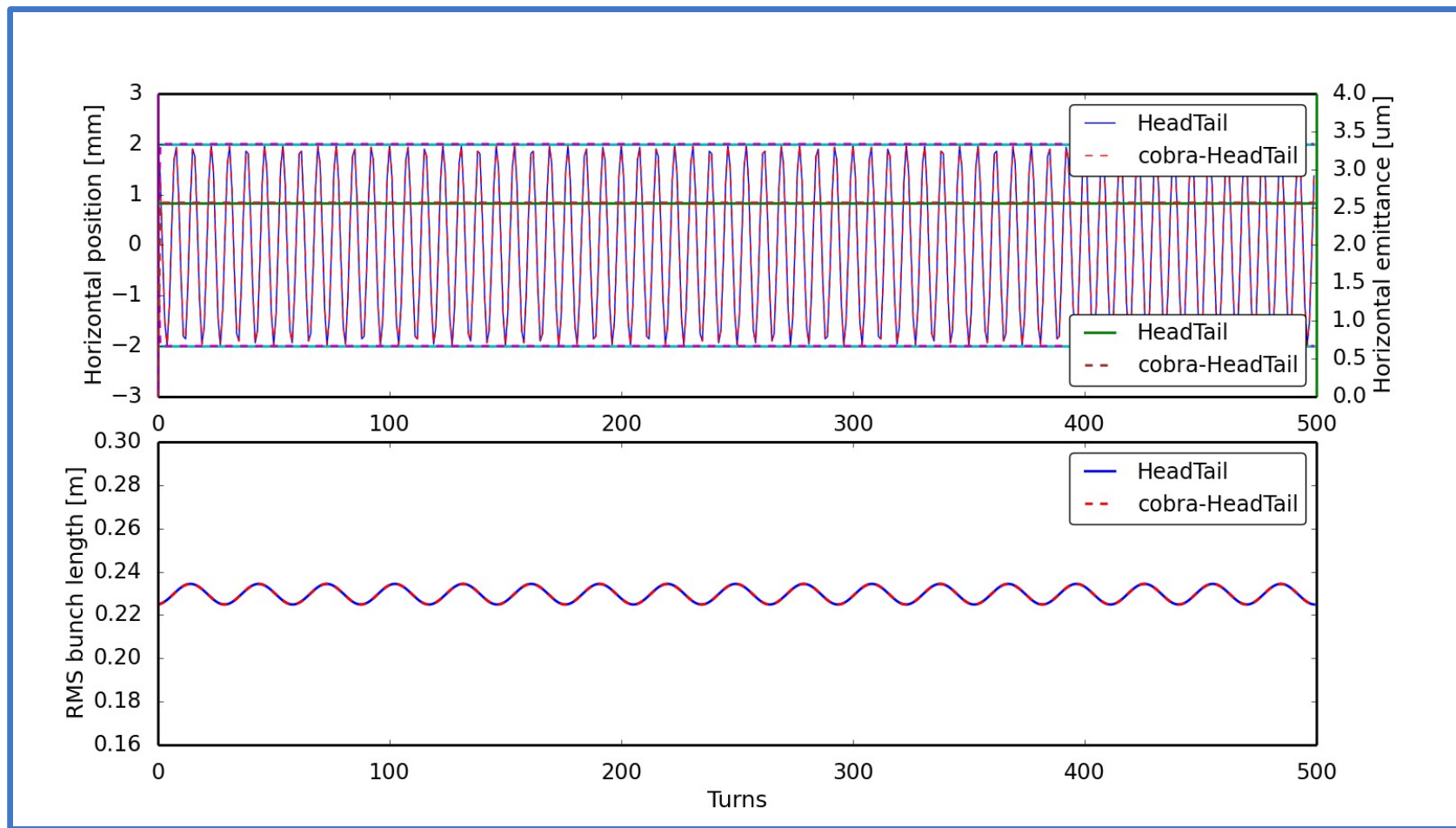


# Benchmarks



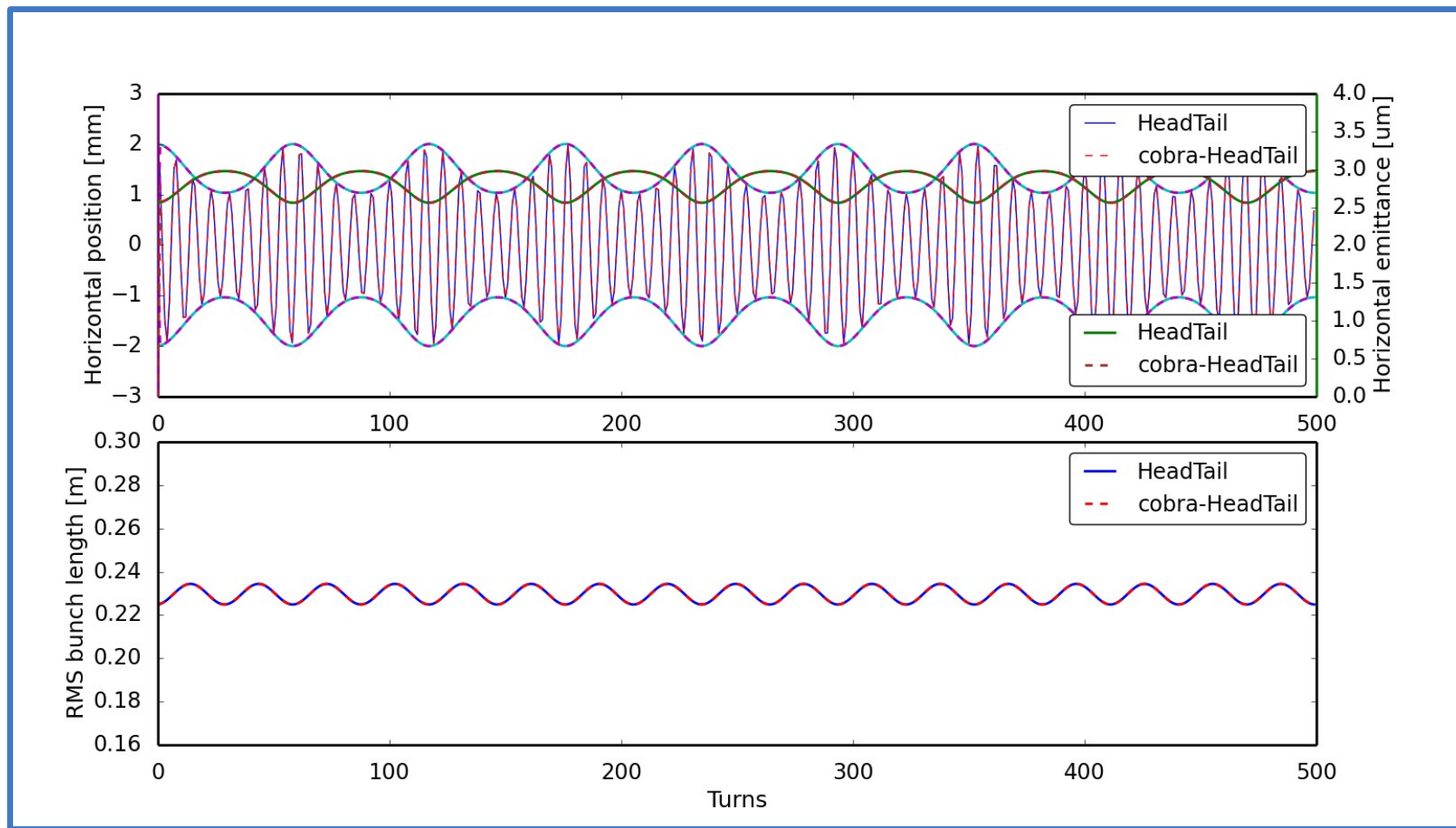
# Benchmarks - tracking

## Linear bucket



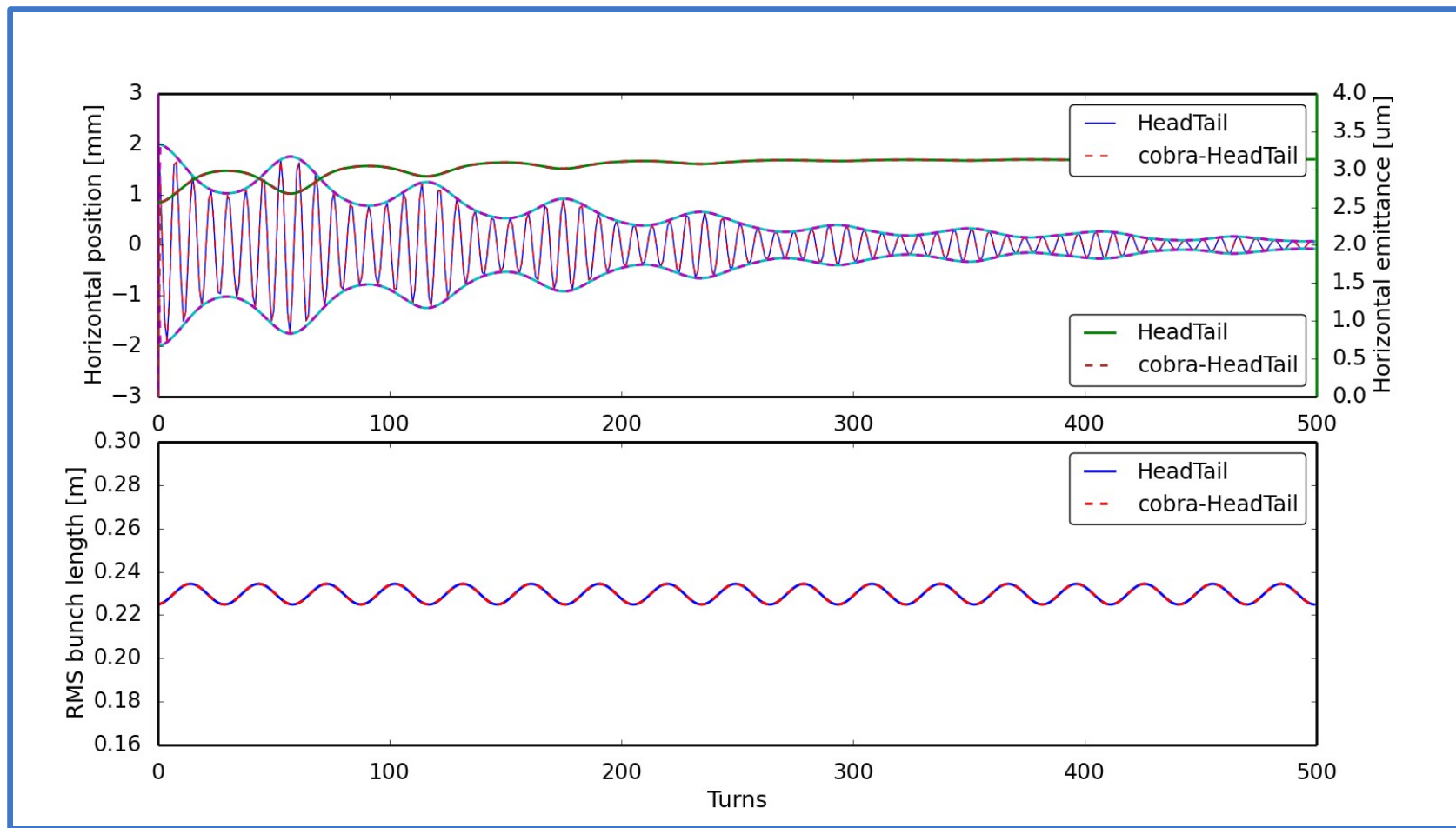
# Benchmarks - tracking

## Linear bucket – chromaticity



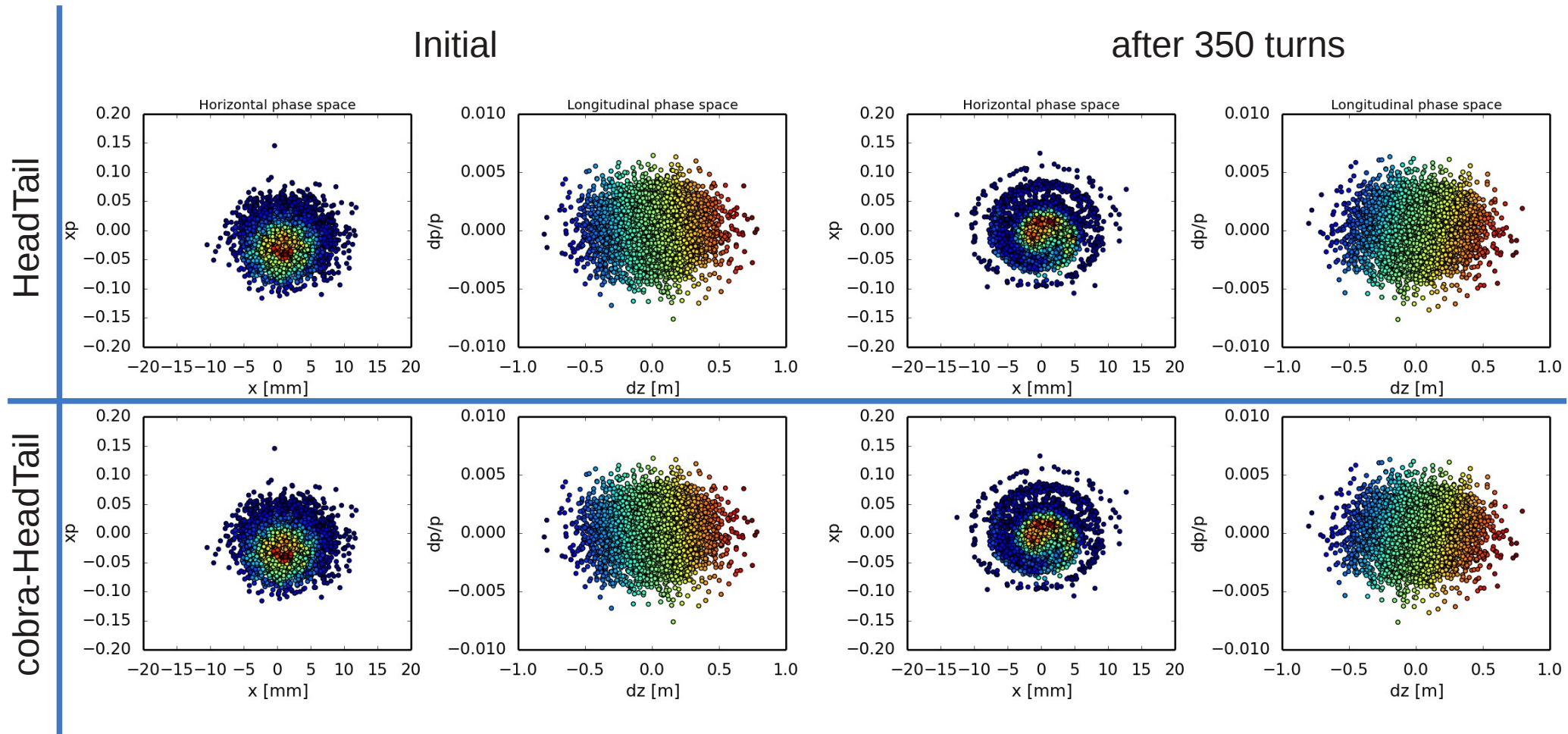
# Benchmarks - tracking

Linear bucket – chromaticity – amplitude detuning



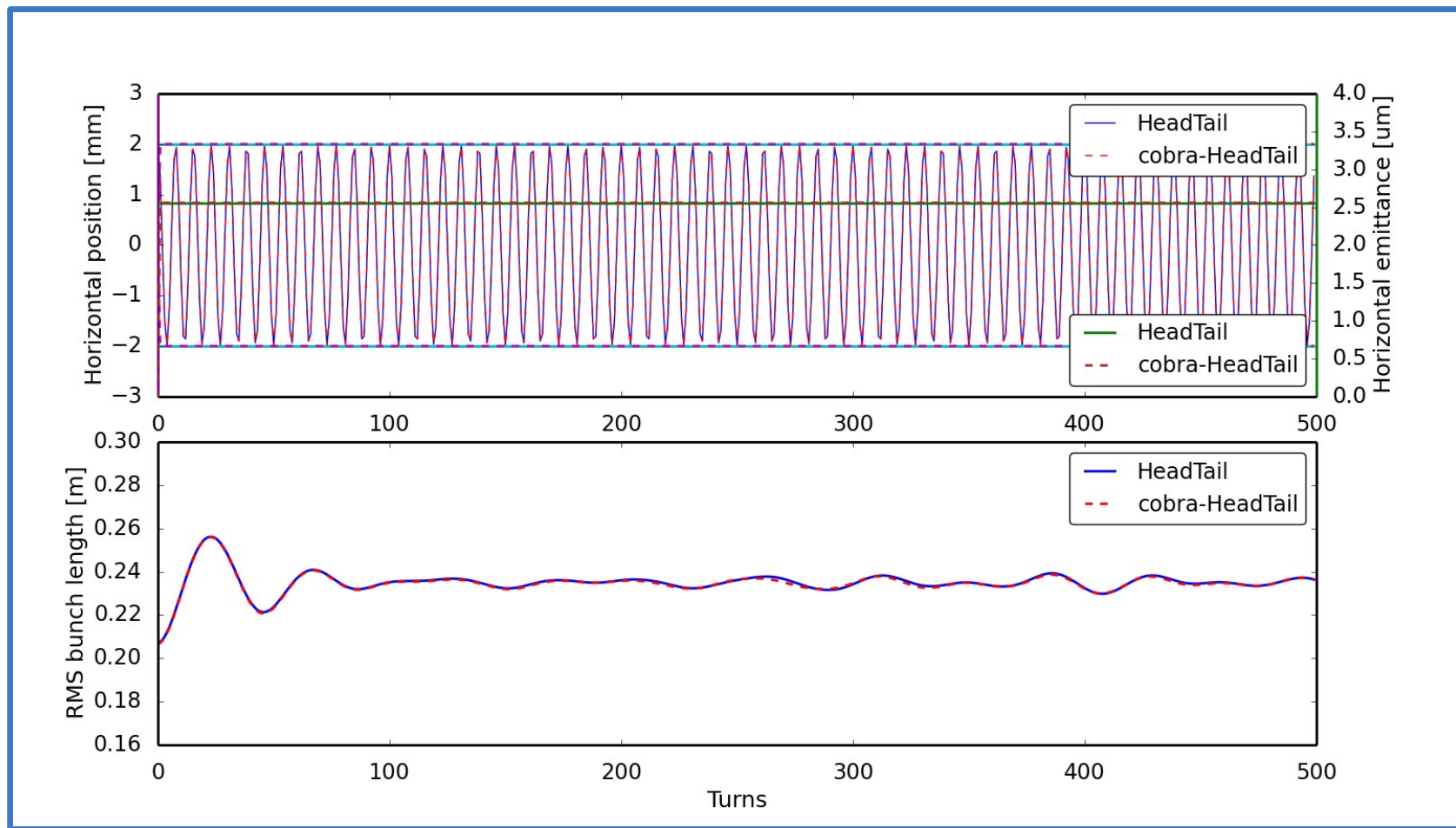
# Benchmarks - tracking

Linear bucket – chromaticity – amplitude detuning



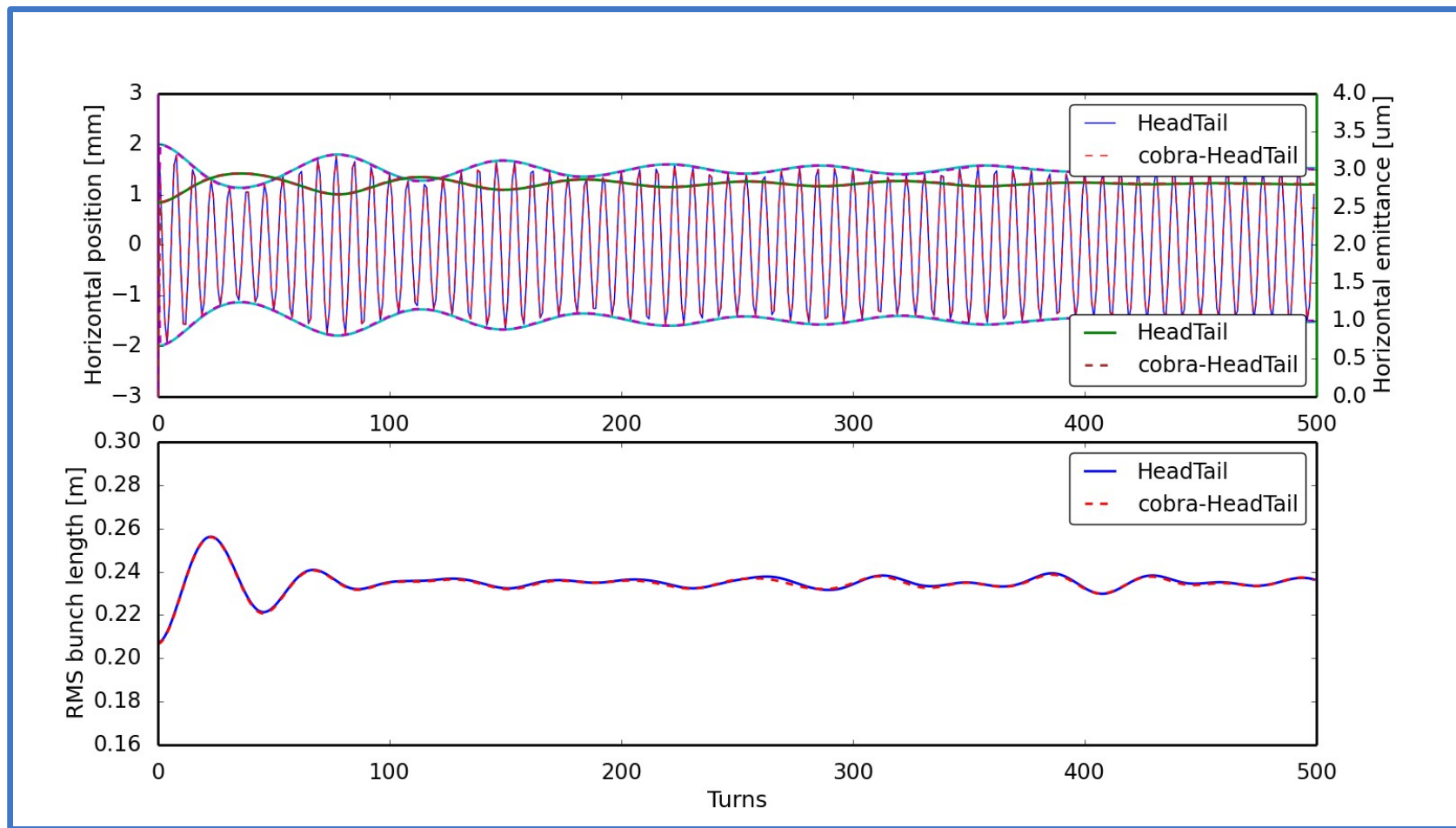
# Benchmarks - tracking

## Nonlinear bucket



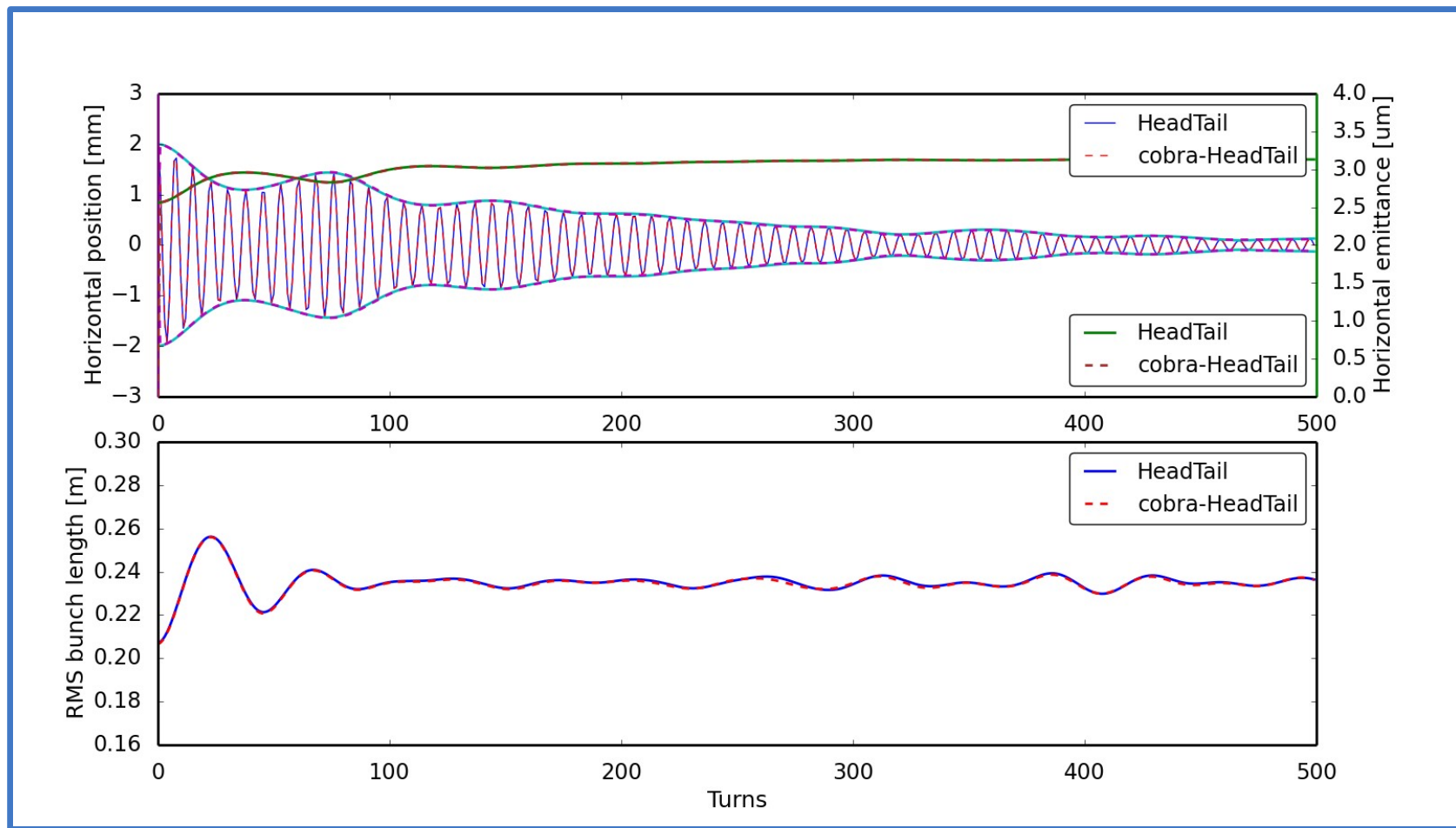
# Benchmarks - tracking

## Nonlinear bucket – chromaticity



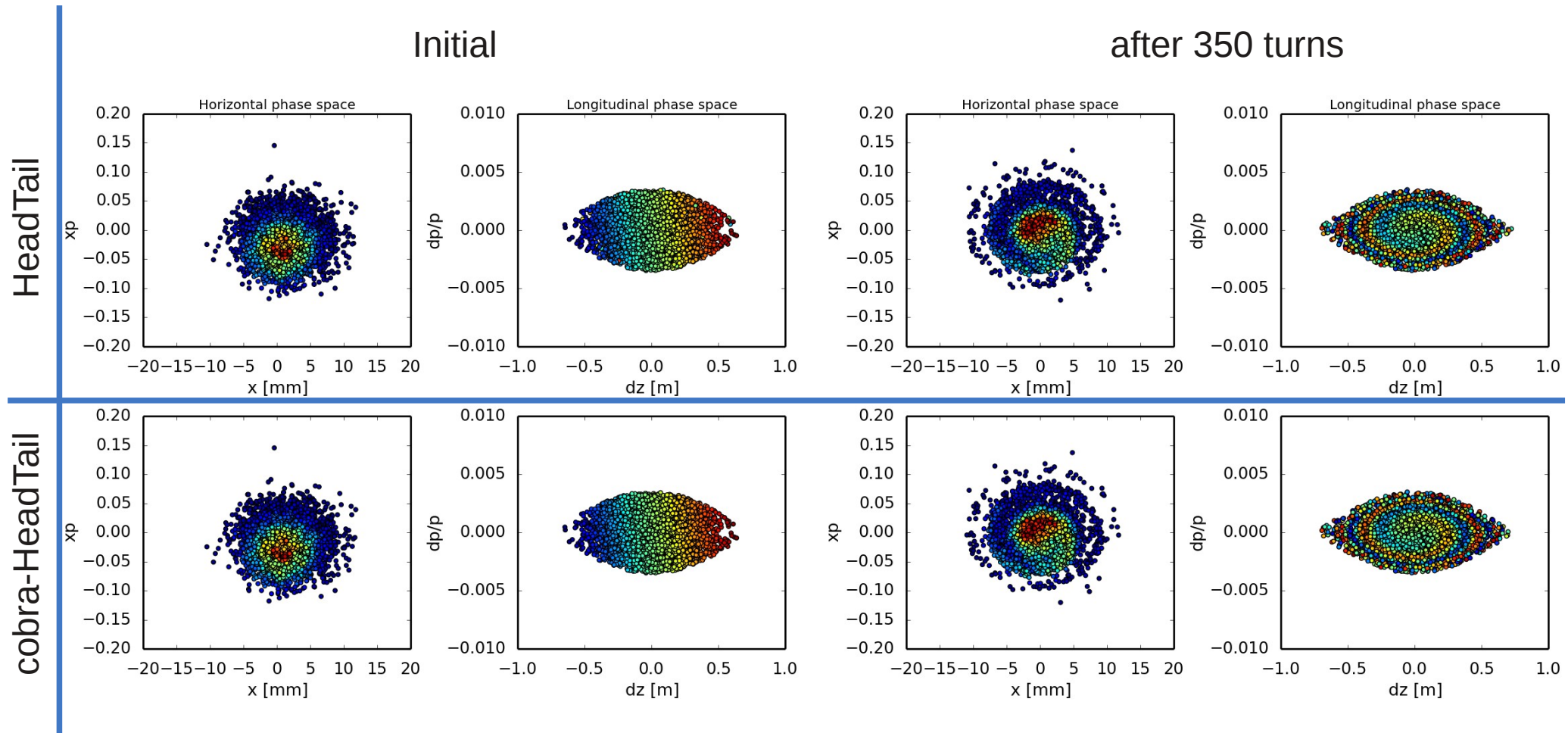
# Benchmarks - tracking

Nonlinear bucket – chromaticity – amplitude detuning



# Benchmarks - tracking

Nonlinear bucket – chromaticity – amplitude detuning





# Poisson solver

- New FFT-type 2D integrated Green's function Poisson solver – similar to Ji Qiang's<sup>1</sup> BeamBeam3D or Impact
- Base class functions highly generic
- Prepared for simple parallelisation

<sup>1</sup> J. Qiang et al., "A parallel particle-in-cell model for beam-beam interaction in high energy ring colliders", <http://www.sciencedirect.com/science/article/pii/S0021999104000282>

```

betatron.h  main.cpp  betatron.cpp  cloud.cpp
590
591 void Cloud::fastpush(Bunch& bunch, int i_slice)
631
632 void Cloud::track(Bunch& bunch)
633 {
634     if (bunch.is_sliced)
635     {
636         rebuild(bunch);
637         poisson.build<Bunch&, Cloud&>(bunch, *this);
638         poisson.integratedGreen();
639
640         //         t0 = 0;
641         for (size_t i=1; i<bunch.get_nslices() + 1; i++)
642         {
643             #pragma omp parallel sections
644             {
645                 #pragma omp section
646                 {
647                     poisson.fastgather<Bunch&>(bunch, i);
648                 #pragma omp critical
649                 {
650                     poisson.computePotential<Bunch&>(bunch, i);
651                     poisson.computeFields<Bunch&>(bunch, i);
652                 }
653             }
654             #pragma omp section
655             {
656                 poisson.fastgather<Cloud&>(*this, i);
657             #pragma omp critical
658             {
659                 poisson.computePotential<Cloud&>(*this, i);
660                 poisson.computeFields<Cloud&>(*this, i);
661             }
662         }
663     }
664     poisson.parallelspace<Bunch&, Cloud&>(bunch, *this, i);
665     poisson.write<Bunch&, Cloud&>(bunch, *this, i);
666     fastpush(bunch, i);
669 }

```

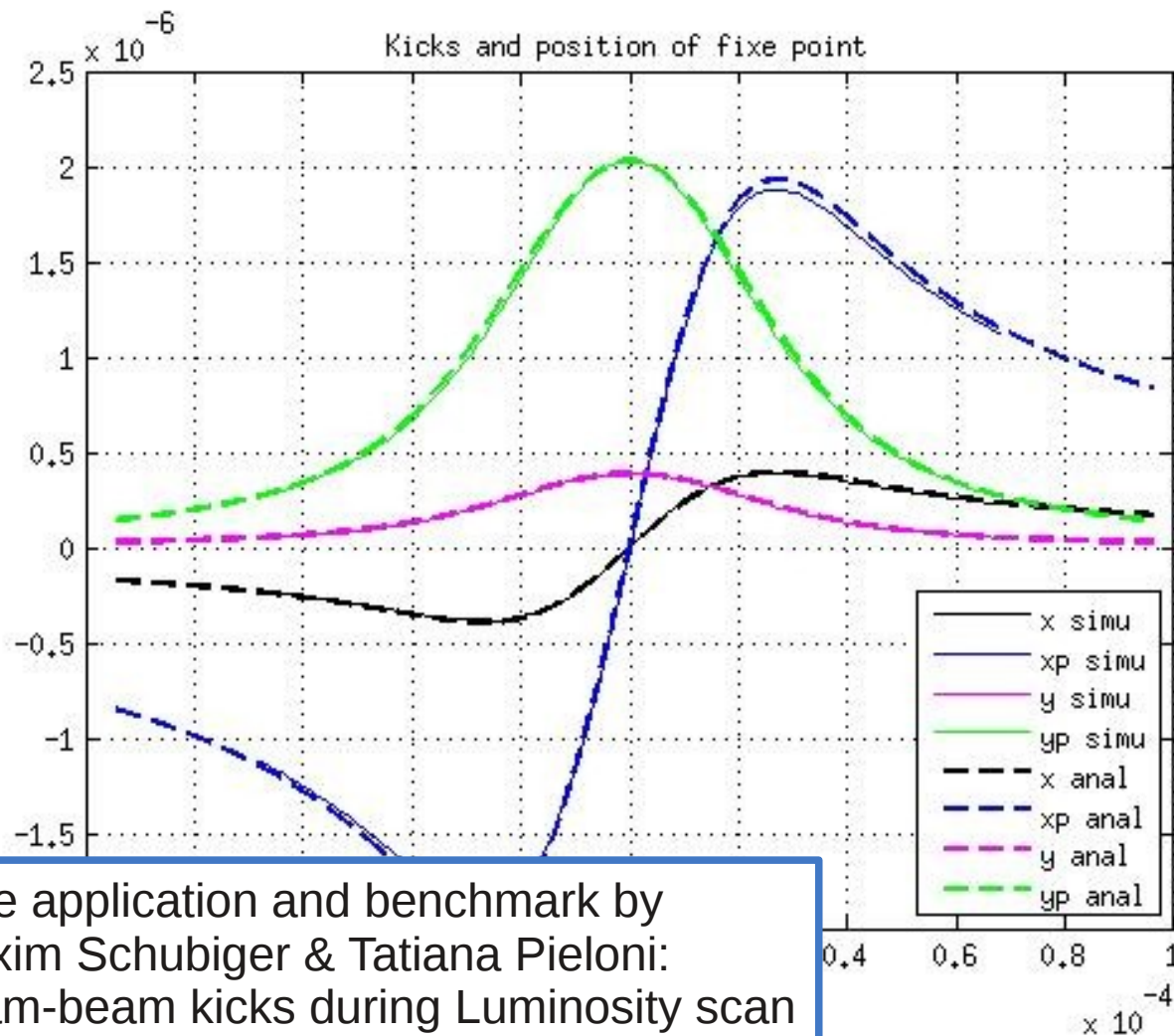
# Poisson solver

- New FFT-type 2D

integrate  
Poisson  
Ji Qiang'  
Impact

- Base cla  
generic

- Prepared  
parallelis



Nice application and benchmark by  
Maxim Schubiger & Tatiana Pieloni:  
beam-beam kicks during Luminosity scan

<sup>1</sup> J. Qiang et al. "Particle-in-particle interaction in colliders",  
<http://www.sciencedirect.com/science/article/pii/S0021999104000282>

```
667
668     fastpush(bunch, i);
669 }
```

```
    *this, i);
```

# Demo

# cobra-HeadTail workflow

- Select the collective kick elements that we would like to treat and position them along the ring
- Connect all kick elements with linear transfer maps (create a betatron object)

Import libraries  
we want to use

Create objects  
we want to use  
for our simulation

Build some maps

Track

```
for i in xrange(nturns):  
    for m in map:  
        m.track(bunch)
```

Postprocess



# cobra-HeadTail demo

- Let's see how we can use this code... we will go through the following steps:
  - Fetch using git
  - Build using CMake
  - Initialize a bunch
  - Build a lattice
  - Transverse tracking
  - Longitudinal tracking
  - Implement an ideal damper
  - Introduce some electron cloud



# cobra-HeadTail demo

- Required packages to build
  - C++ compiler
  - C++ libraries and development package
  - boost libraries and development package
  - FFTW3 libraries and development package
  - GSL libraries and development package
  - HDF5 libraries and development package
  - cmake
  - Up-to-date versions of Python, NumPy, Scipy, Matplotlib etc.

```
• git clone https://git.cern.ch/repos/cobra cobra-demo
• mkdir cobra-demo-build; cd cobra-demo-build
• cmake ../cobra-demo/src      Makefile
• make      Libraries
```



# cobra-HeadTail demo

- Requirements

- C++

- C++

- boost

- FFTW

- GSL

- HD

- cmake

- Up-

- git

- mkdir

- cmake

- Make

```
kli@pcbel3989:~/workspace> git clone https://git.cern.ch/repos/cobra cobra-demo
Klone nach 'cobra-demo'...
Username for 'https://git.cern.ch': kli
Password for 'https://kli@git.cern.ch':
remote: Counting objects: 1000, done.
remote: Compressing objects: 100% (209/209), done.
remote: Total 1000 (delta 663), reused 1000 (delta 663)
Receiving objects: 100% (1000/1000), 268.05 KiB, done.
Resolving deltas: 100% (663/663), done.
```

```
kli@pcbel3989:~/workspace> mkdir cobra-demo-build
kli@pcbel3989:~/workspace> cd cobra-demo-build/
kli@pcbel3989:~/workspace/cobra-demo-build> CXX=icpc cmake ../cobra-demo/src/
-- The CXX compiler identification is Intel 13.1.0.20130313
-- Check for working CXX compiler: /home/kli/Software/Intel/opt/intel/composer_xe_2013.3.163/bin/intel64/icpc
-- Check for working CXX compiler: /home/kli/Software/Intel/opt/intel/composer_xe_2013.3.163/bin/intel64/icpc -- works
-- Detecting CXX compiler ABI info
-- Detecting CXX compiler ABI info - done
**** Building cobra ****
Default options for Debug: -g
Default options for Release: -O3 -DNDEBUG
*****
Using compilers:
> CXX: Intel
```

```
*****
-- Found PythonLibs: /usr/lib64/libpython2.7.so (found version "2.7.3")
--
Using Python libraries:
  /usr/include/python2.7 /usr/lib64/libpython2.7.so
-- Boost version: 1.49.0
-- Found the following Boost libraries:
--   python
--
Using boost libraries:
  /usr/include /usr/lib64/libboost_python-mt.so
-- Configuring done
-- Generating done
-- Build files have been written to: /home/kli/workspace/cobra-demo-build
kli@pcbel3989:~/workspace/cobra-demo-build> make
Scanning dependencies of target Bunch
[ 4%] Building CXX object CMakeFiles/Bunch.dir/bunch.cpp.o
```

a-demo



# Conclusions

- Developed a modular and scriptable, Python interfaced version of HeadTail – written in C++ and interfaced via Boost.Python
- In principle, any effect can now be added as module with relative ease
- Pre-beta version is ready for download, build and test on the CERN git server
- Contributors now needed as we are missing:
  - Impedance model
  - MAD-X interface
  - Perhaps some form of beam-beam (Poisson solvers are there)
  - Space charge (Poisson solvers are there)
- Should we open a HeadTail-developers-and-users (HEDEUS) group?  
Who would want to participate?

