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:

ECLOUD

What is cobra-HeadTail?

A little success story:

ECLOUD $\xrightarrow{\text{Python}}$ PyECLOUD

G. Iadarola
G. Rumolo

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:

ECLOUD $\xrightarrow{\text{Python}}$ PyECLOUD

HeadTail

What is cobra-HeadTail?

A little success story:

ECLOUD $\xrightarrow{\text{Python}}$ PyECLOUD

HeadTail $\longrightarrow$ PyHeadTail

What is cobra-HeadTail?

A little success story:

ECLOUD $\xrightarrow{\text{Python}}$ PyECLOUD

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

What is cobra-HeadTail?

A little success story:

ECLOUD $\xrightarrow{\text{Python}}$ PyECLOUD

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

cobra-HeadTail

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:

ECLOUD $\xrightarrow{\text{Python}}$ PyECLOUD

HeadTail $\xrightarrow[\text{Python}]{\text{C++}}$ ~~P~~yHeadTail

cobra-HeadTail

$\downarrow$

Collective **b**eam dynamics in **r**ing **a**ccelerators

What is cobra-HeadTail?

A little success story:

ECLOUD —Python→ PyECLOUD

HeadTail —C++ / Python→ ~~P~~yHeadTail

cobra-HeadTail

↓

Collective **b**eam dynamics in **r**ing **a**ccelerators

&

Analysis of cobra-HeadTail numerical simulation data

What is cobra-HeadTail?

A little success story:

ECLOUD $\xrightarrow{\text{Python}}$ PyECLOUD

HeadTail $\xrightarrow[\text{Python}]{\text{C++}}$ ~~P~~yHeadTail

cobra-HeadTail

↓

Collective **b**eam dynamics in **r**ing **a**ccelerators

&

Analysis of **co**bra-HeadTail **n**umerical simulation **da**ta

↓

Anaconda

What is cobra-HeadTail?

A little success story:

ECLOUD —Python→ PyECLOUD

HeadTail —C++ / Python→

Tail

All seems to fit, but... caution!

C eam dynamics in **r**ing **a**ccelerators

&

Analysis of **co**bra-HeadTail **n**umerical simulation **da**ta

↓

Anaconda

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

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

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.

Why cobra-HeadTail?

Why cobra-HeadTail? HeadTail works just fine!

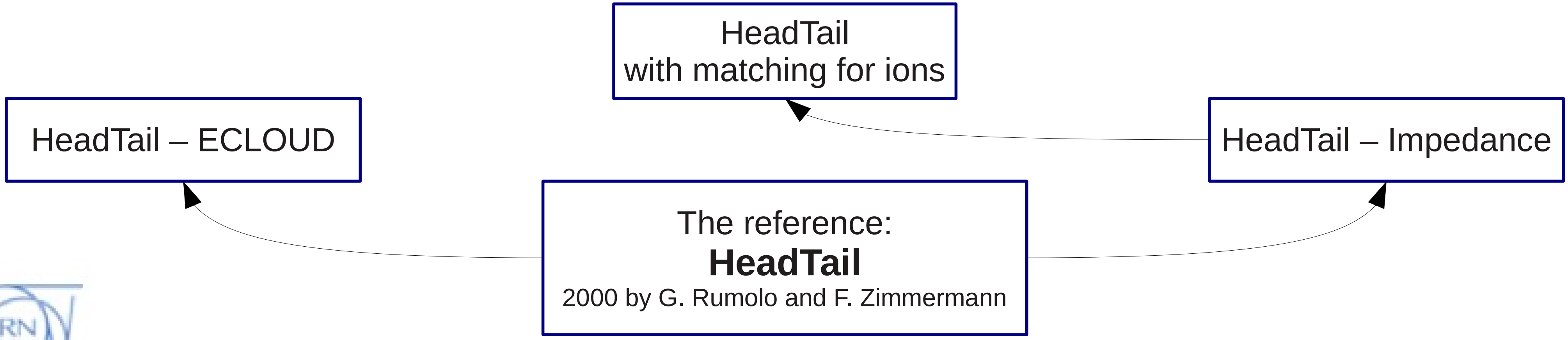
Why cobra-HeadTail?

Why cobra-HeadTail? HeadTail works just fine!

The reference:
HeadTail
2000 by G. Rumolo and F. Zimmermann

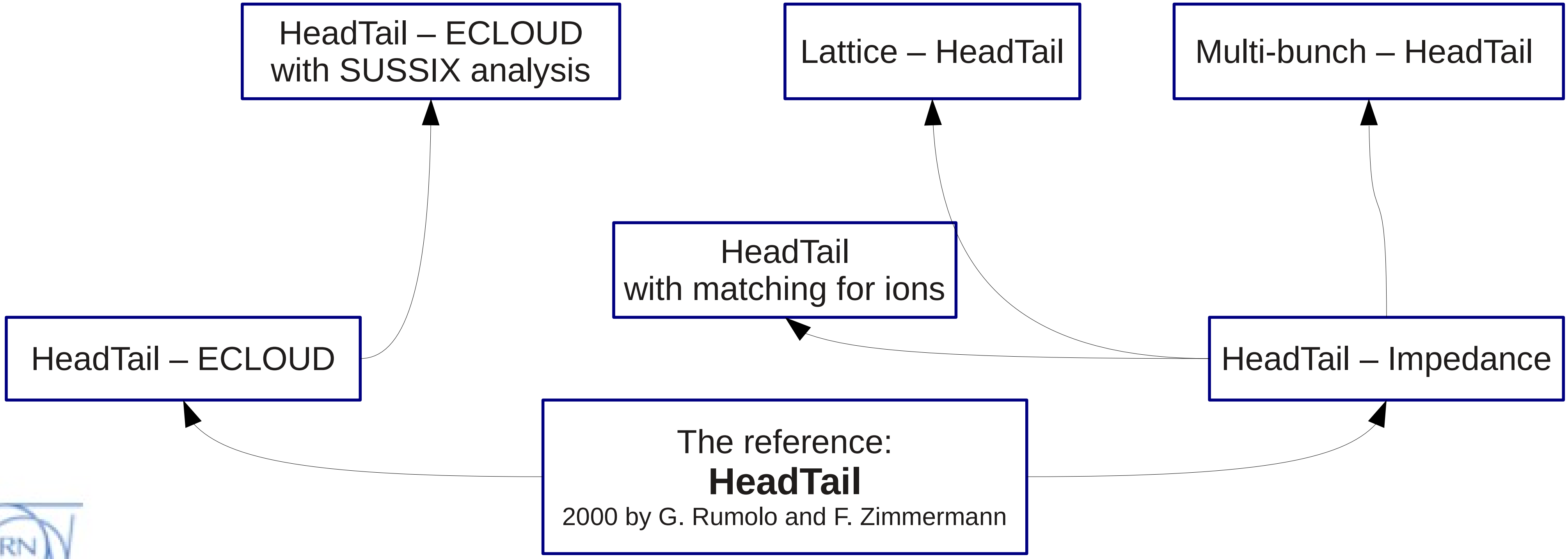
Why cobra-HeadTail?

Why cobra-HeadTail? HeadTail works just fine!



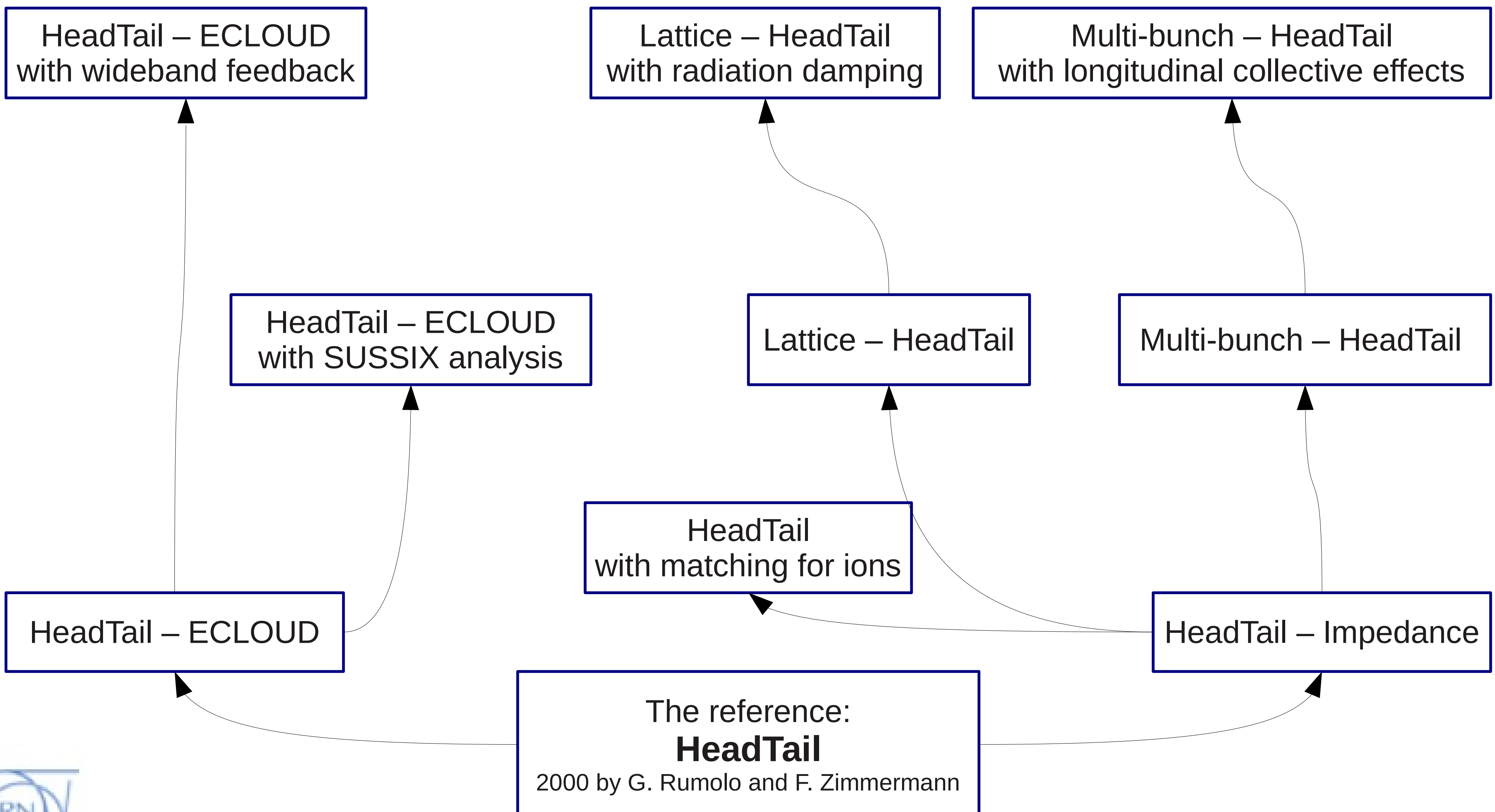
Why cobra-HeadTail?

Why cobra-HeadTail? HeadTail works just fine!



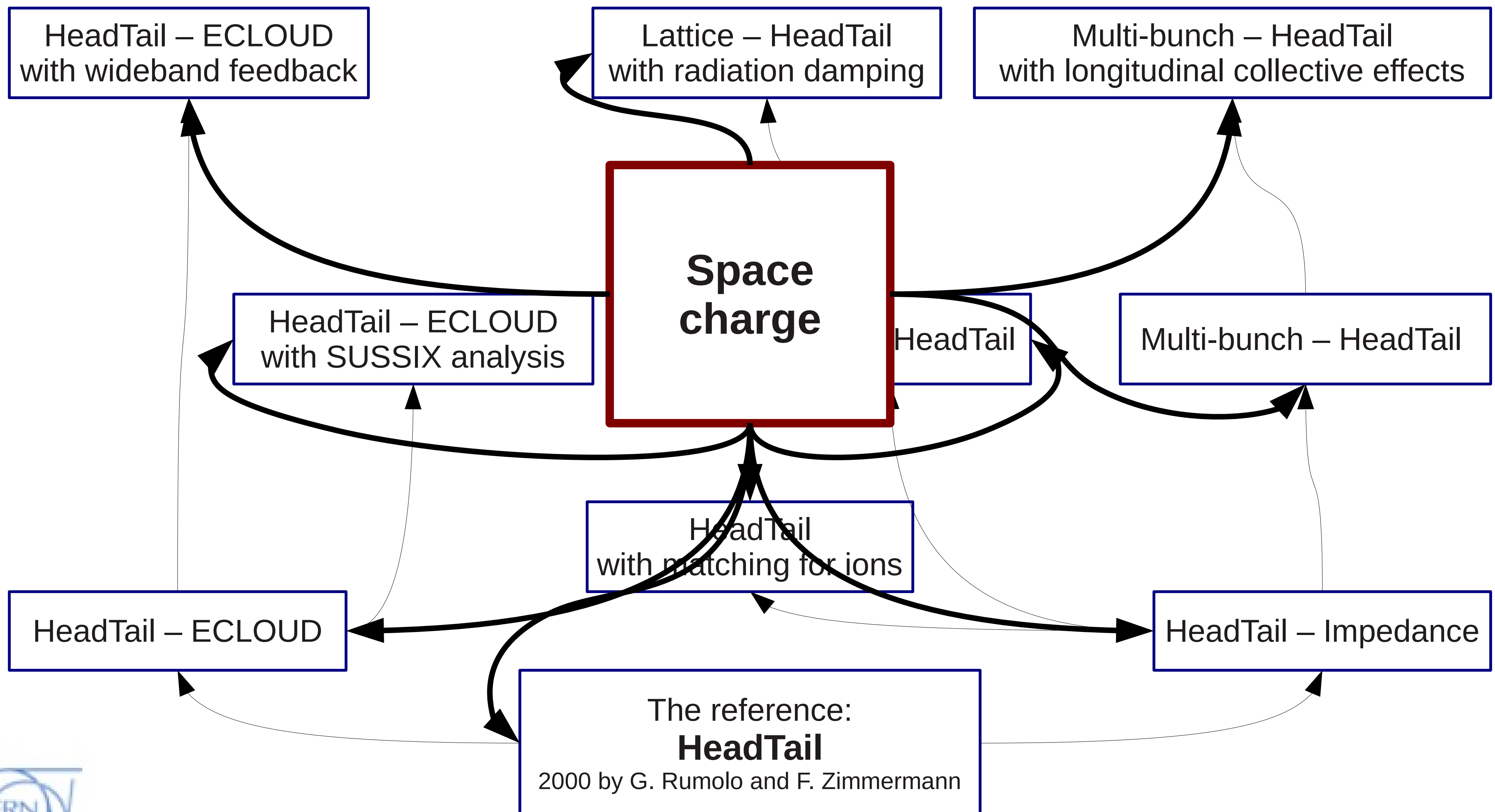
Why cobra-HeadTail?

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!

Hea
with w

fects

- At least we had svn – some kind of version control
- Still, efforts are scattered, work is redone and good work is under the thread of being lost

dTail

Head

dance

The reference:
HeadTail
2000 by G. Rumolo and F. Zimmermann

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?

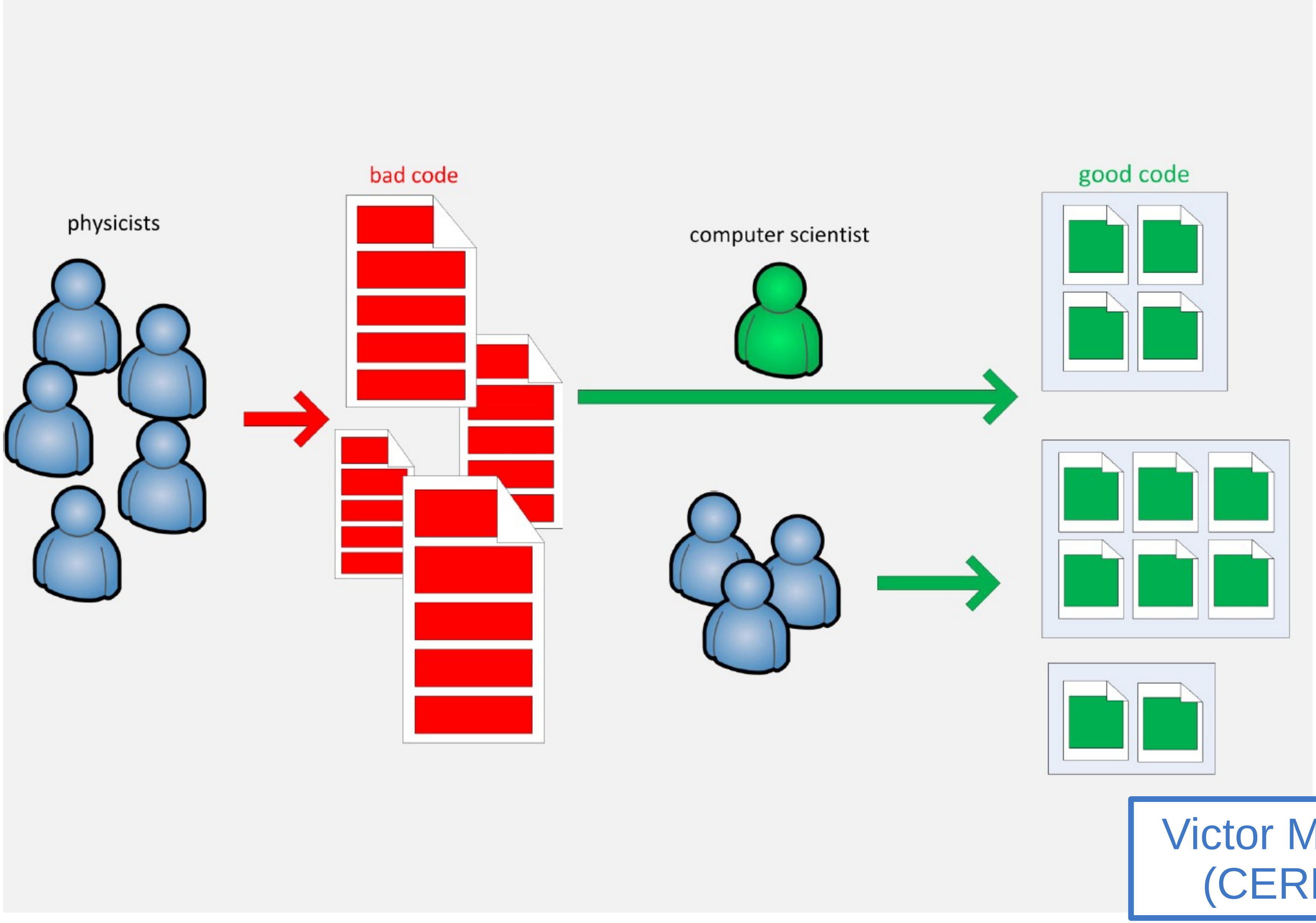
Possible reasons for scattered efforts:

- 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



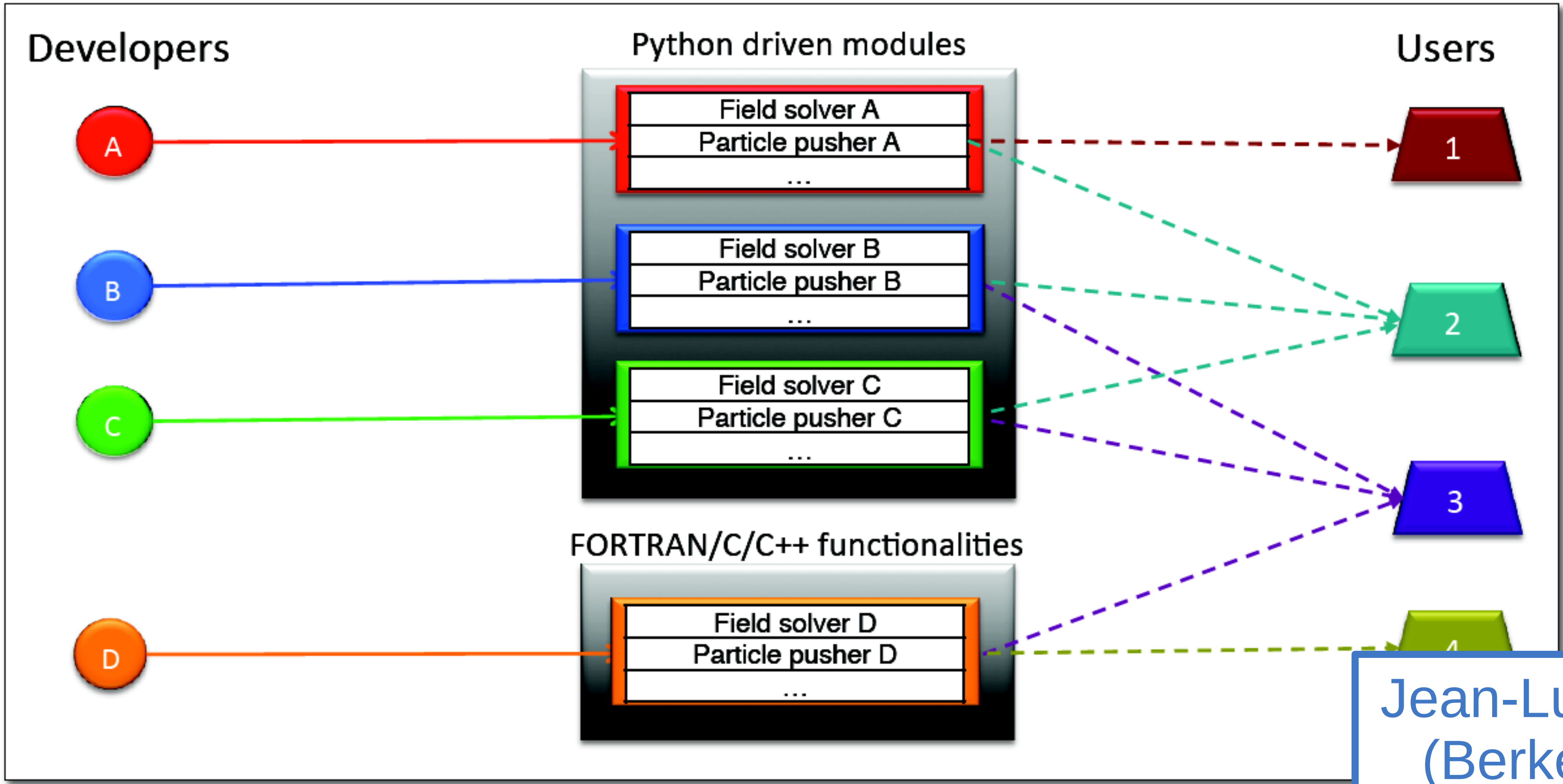
Victor Maier (CERN)

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!



Jean-Luc Vay (Berkeley)

Model

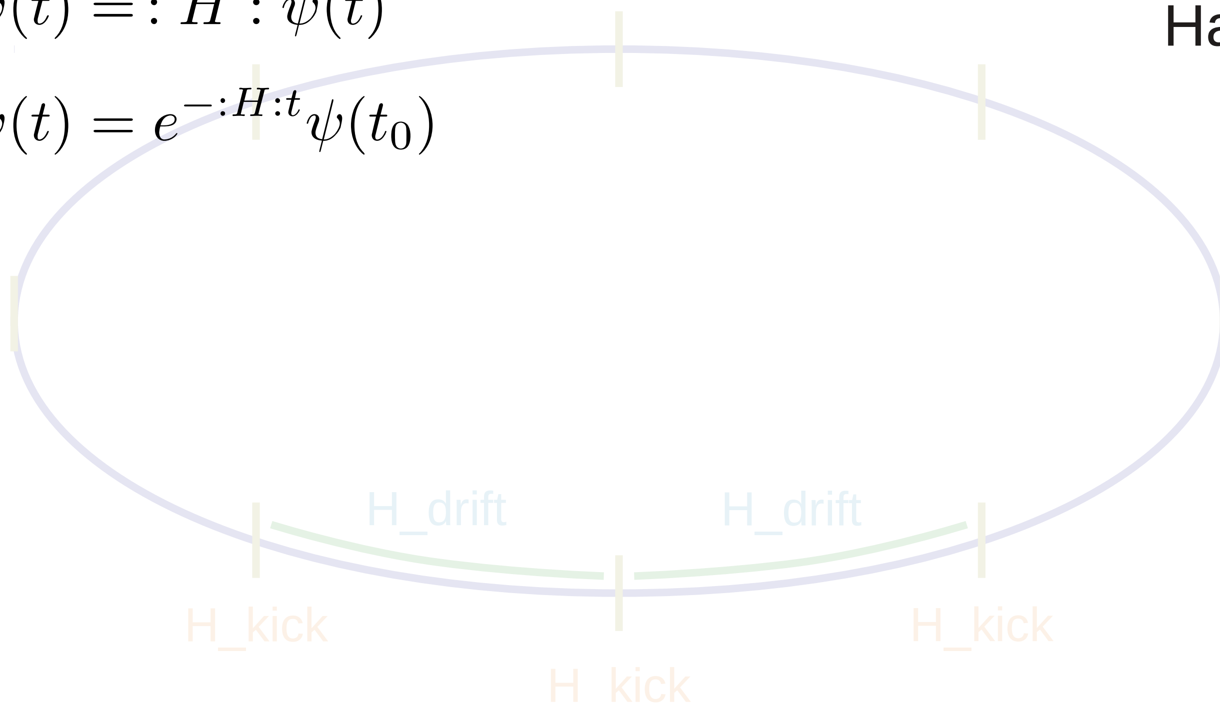
The HeadTail model

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 }

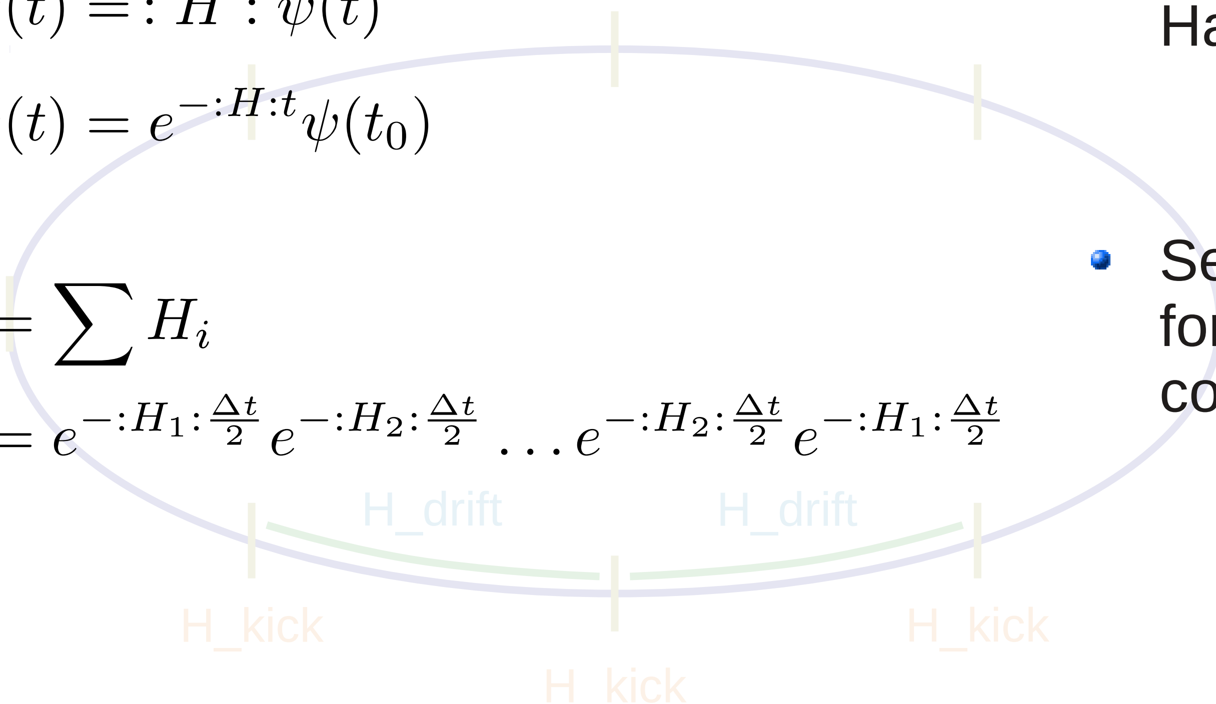
$$\frac{d}{dt}\psi(t) = -[H, \psi(t)]$$

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

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

$$H = \sum H_i$$

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



- 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 }

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

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

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

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

$$H = \sum H_i$$

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

- Several Hamiltonian are formally easily concatenated

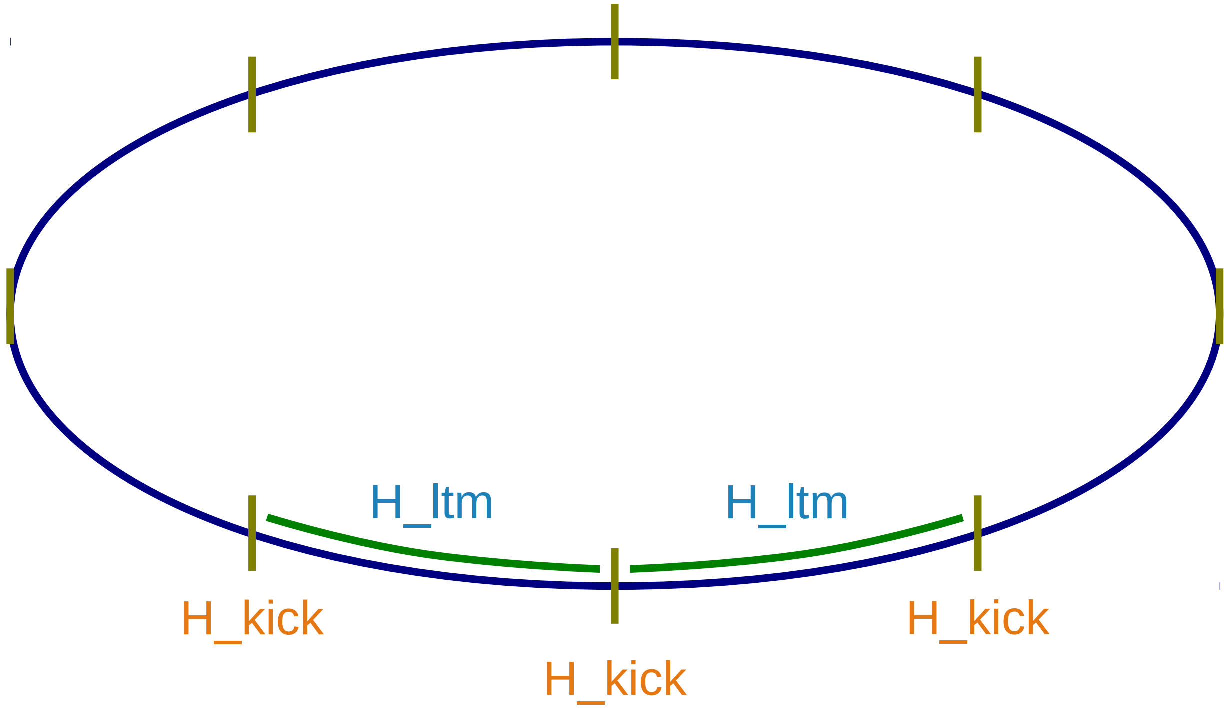
$$H = A(p) + V(q) = H_D + H_K$$

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

- 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

$$
\begin{aligned}
\Delta Q_x &= Q'_x\delta + \alpha_{xx}J_x + \alpha_{xy}J_y + \dots \\
\Delta Q_y &= Q'_y\delta + \alpha_{yx}J_x + \alpha_{yy}J_y + \dots
\end{aligned}
$$

One-turn-map

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

(1)
$$
\begin{aligned}
M(s_0|s_n) &= M(s_0|s_1)\, e^{:V_1:}\, M(s_1|s_2)\, e^{:V_2:} \dots M(s_{n-2}|s_{n-1})\, e^{:V_{n-1}:}\, M(s_{n-1}|s_n) \\
&= M(s_0|s_1)\, e^{:V_1:}\, M(s_0|s_1)^{-1} M(s_0|s_1) M(s_1|s_2)\, e^{:V_2:} \dots \\
&\dots e^{:V_{n-1}:}\, M(s_0|s_{n-1})^{-1} M(s_0|s_{n-1}) M(s_{n-1}|s_n) \\
&= e^{:M(s_0|s_1)\, V_1:}\, e^{:M(s_0|s_2)\, V_2:} \dots e^{:M(s_0|s_{n-1})\, V_{n-1}:}\, M(s_0|s_n) \\
&= B_0^{-1} B_0\, e^{:M(s_0|s_1)\, V_1:}\, B_0^{-1} B_0\, e^{:M(s_0|s_2)\, V_2:} \dots e^{:M(s_0|s_{n-1})\, V_{n-1}:}\, B_0^{-1} R(s_0|s_n) B_n \\
&= B_0^{-1} e^{:\hat{V}_1:}\, e^{:\hat{V}_2:} \dots e^{:\hat{V}_{n-1}:}\, e^{:\hat{V}_n:}\, R(s0|s_n)\, B_n
\end{aligned}
$$

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

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

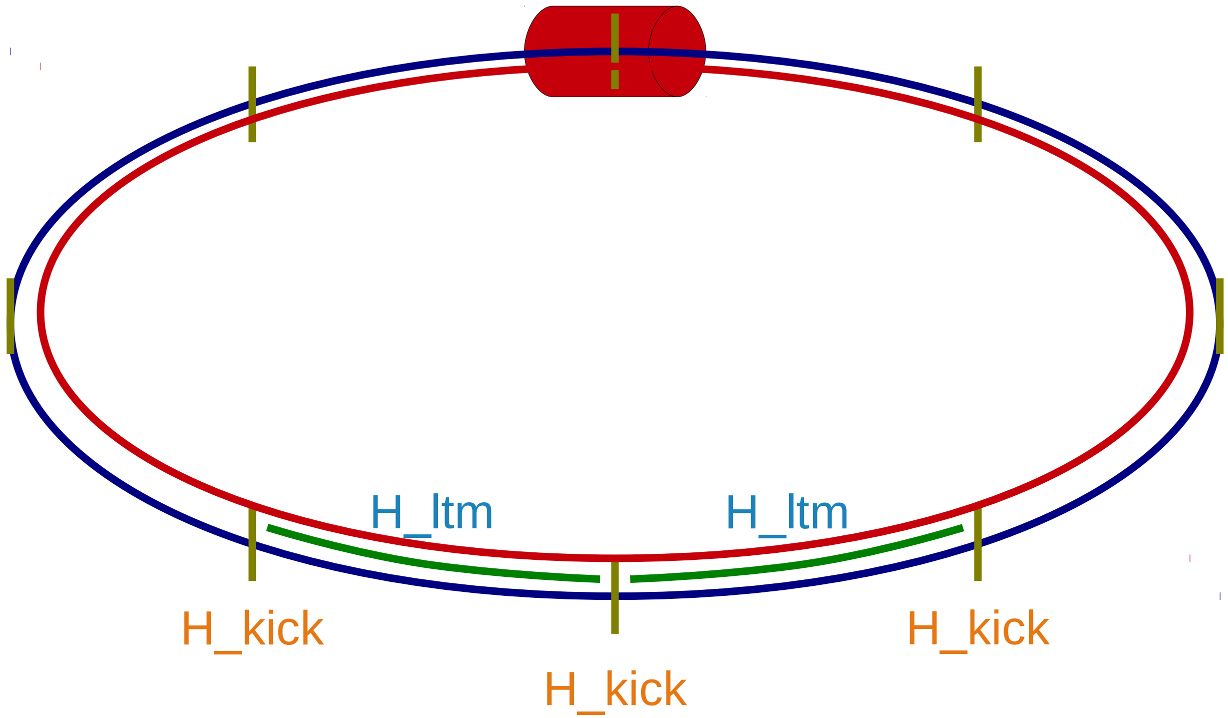
(1): J. Bengtsson, SLS Note 9/97, Paul Scherrer Institut (PSI), Villigen, Schweiz

The HeadTail model – longitudinal

RK4: classical Runge-Kutta

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

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



- 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

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

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

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

external sources – self-consistent solution

HeadTail split

HeadTail self-consistency

$$H_{\mathrm{SC}} = T_1 + \int d^3x\,\rho_1\,V_1 + \int d^3x\,\rho_2\,V_1 + \frac{1}{4}\int d^3x\,F_{\mu\nu;1}F^{\mu\nu;1}$$
$$+ T_2 + \int d^3x\,\rho_2\,V_2 + \int d^3x\,\rho_1\,V_2 + \frac{1}{4}\int d^3x\,F_{\mu\nu;2}F^{\mu\nu;2}$$

e-cloud
space-charge

e-cloud
space-charge

external sources – self-consistent solution

Design

cobra-HeadTail design

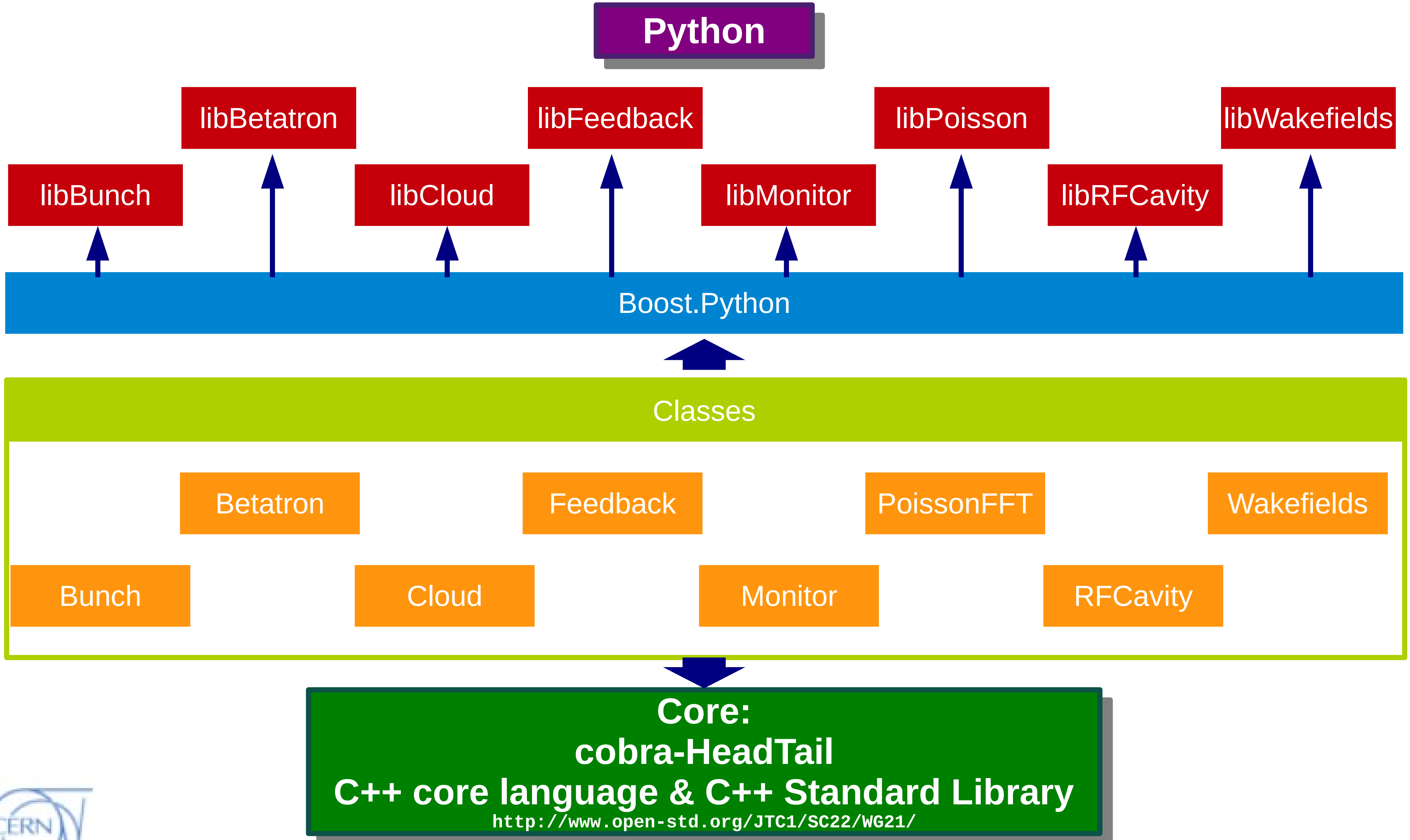
Classes

Betatron | Feedback | PoissonFFT | Wakefields

Bunch | Cloud | Monitor | RFCavity

Core:
cobra-HeadTail
C++ core language & C++ Standard Library
http://www.open-std.org/JTC1/SC22/WG21/

cobra-HeadTail design



cobra-HeadTail design

3rd party libraries:

Python

libBetatron libFeedback libPoisson libWakefields

libMonitor libRFCavity

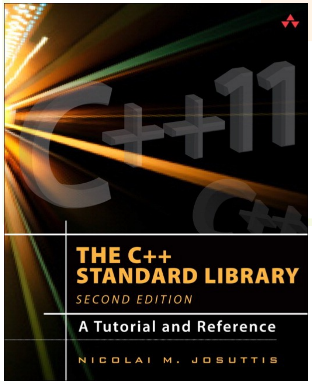
Boost.Python

Betatron PoissonFFT Wakefields

Cloud RFCavity

Core:
cobra-HeadTail
C++ core language & C++ Standard Library

GSL - GNU Scientific Library



The HDF Group

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

Documentation

We should try to write the Documentation along with the code

- Class hierarchy

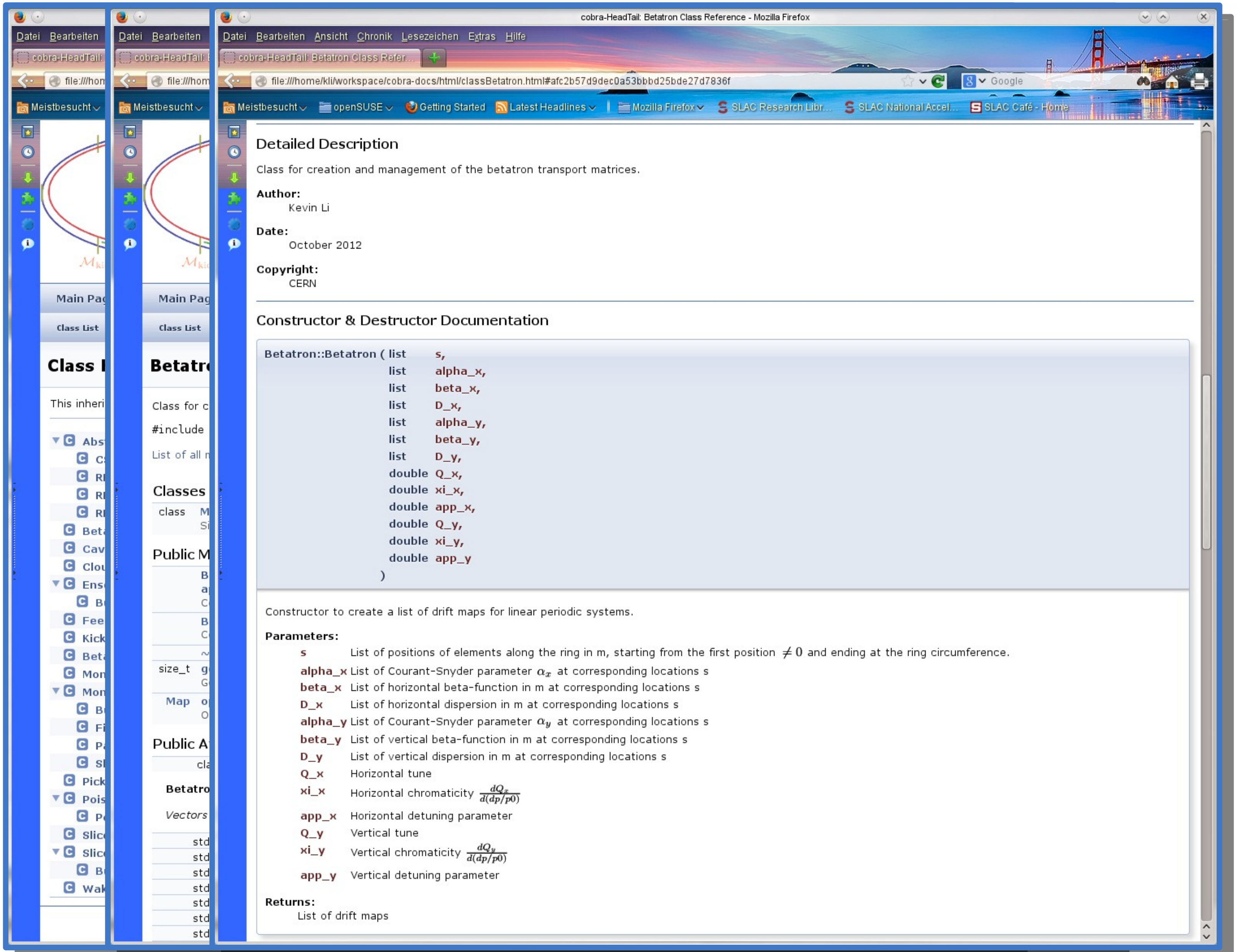
Documentation

We should try to write the Documentation along with the code

- Class hierarchy

- Class members overview

Documentation



We should try to write the Documentation along with the code

- Class hierarchy
- Class members overview
- Class member documentation

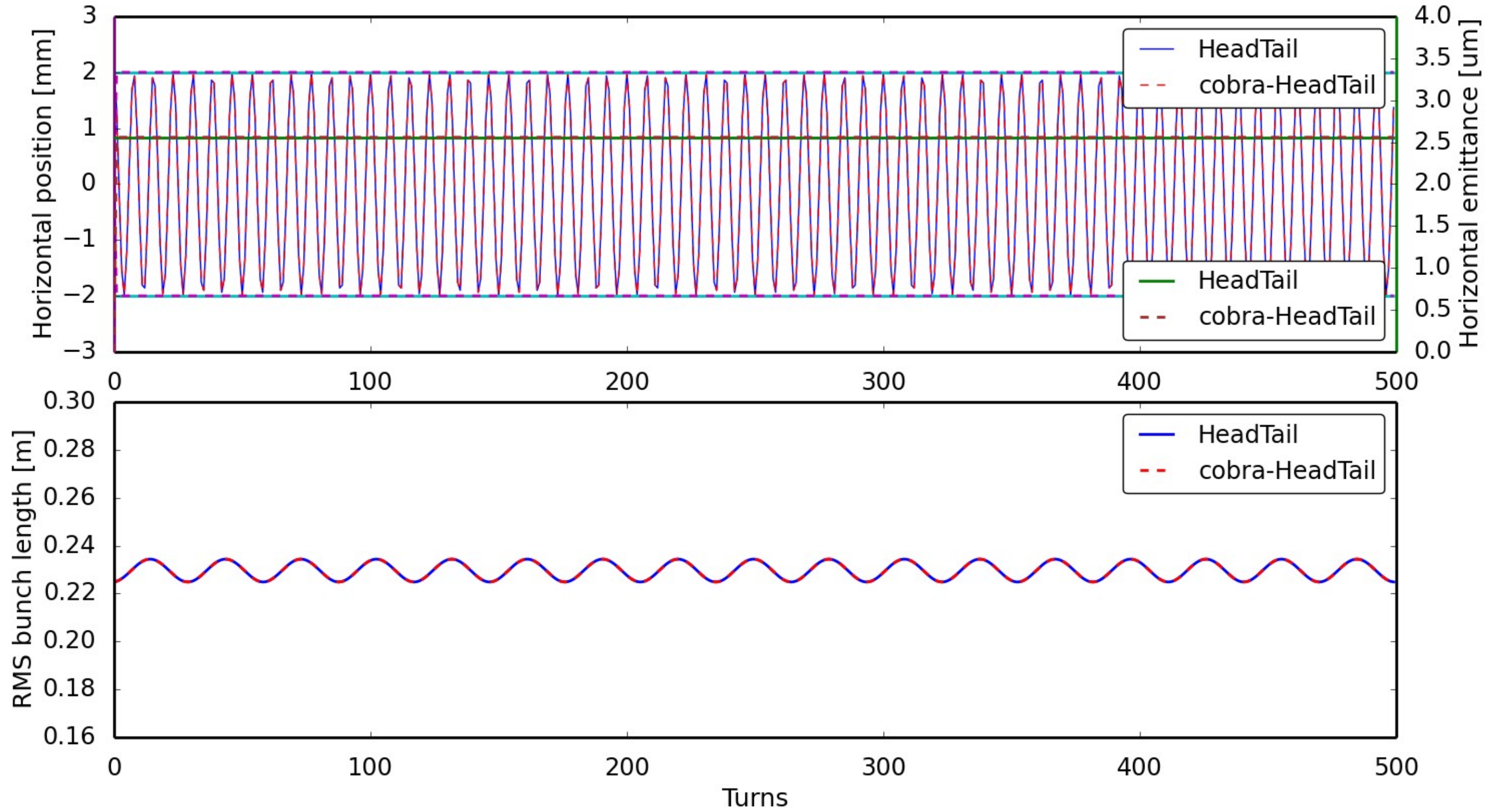
cobra-HeadTail design

- 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

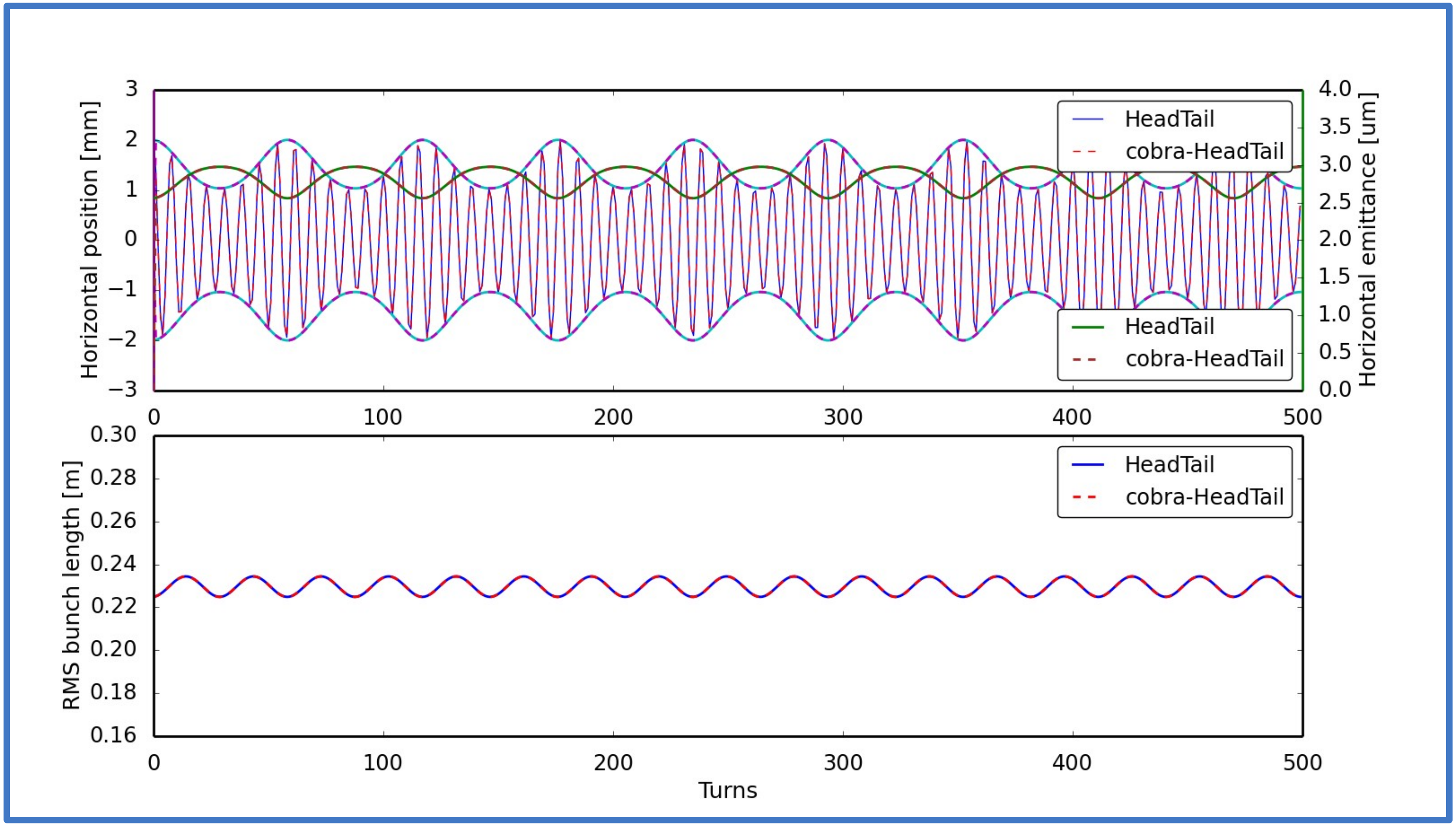
Benchmarks - tracking

Linear bucket



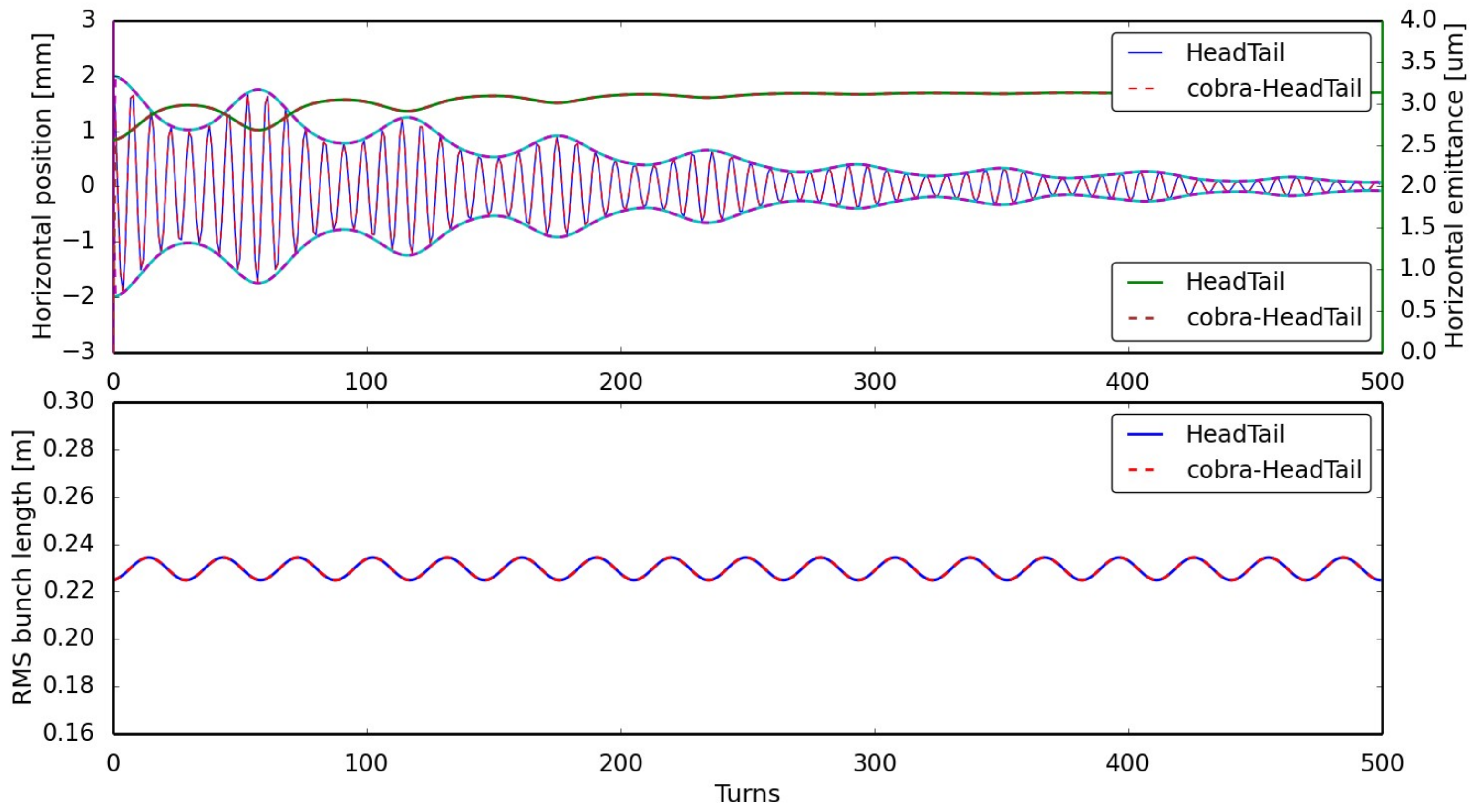
Benchmarks - tracking

Linear bucket – chromaticity



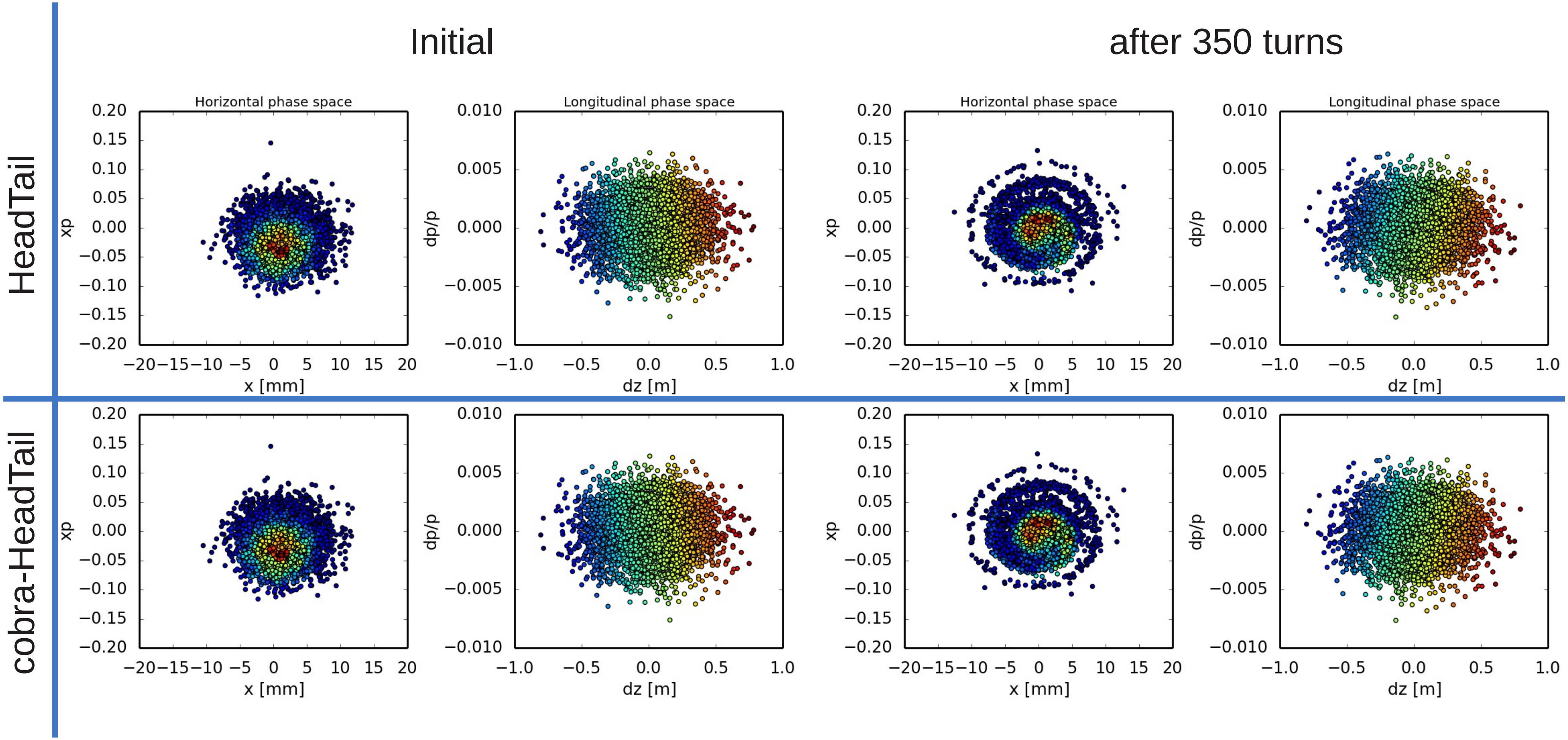
Benchmarks - tracking

Linear bucket – chromaticity – amplitude detuning



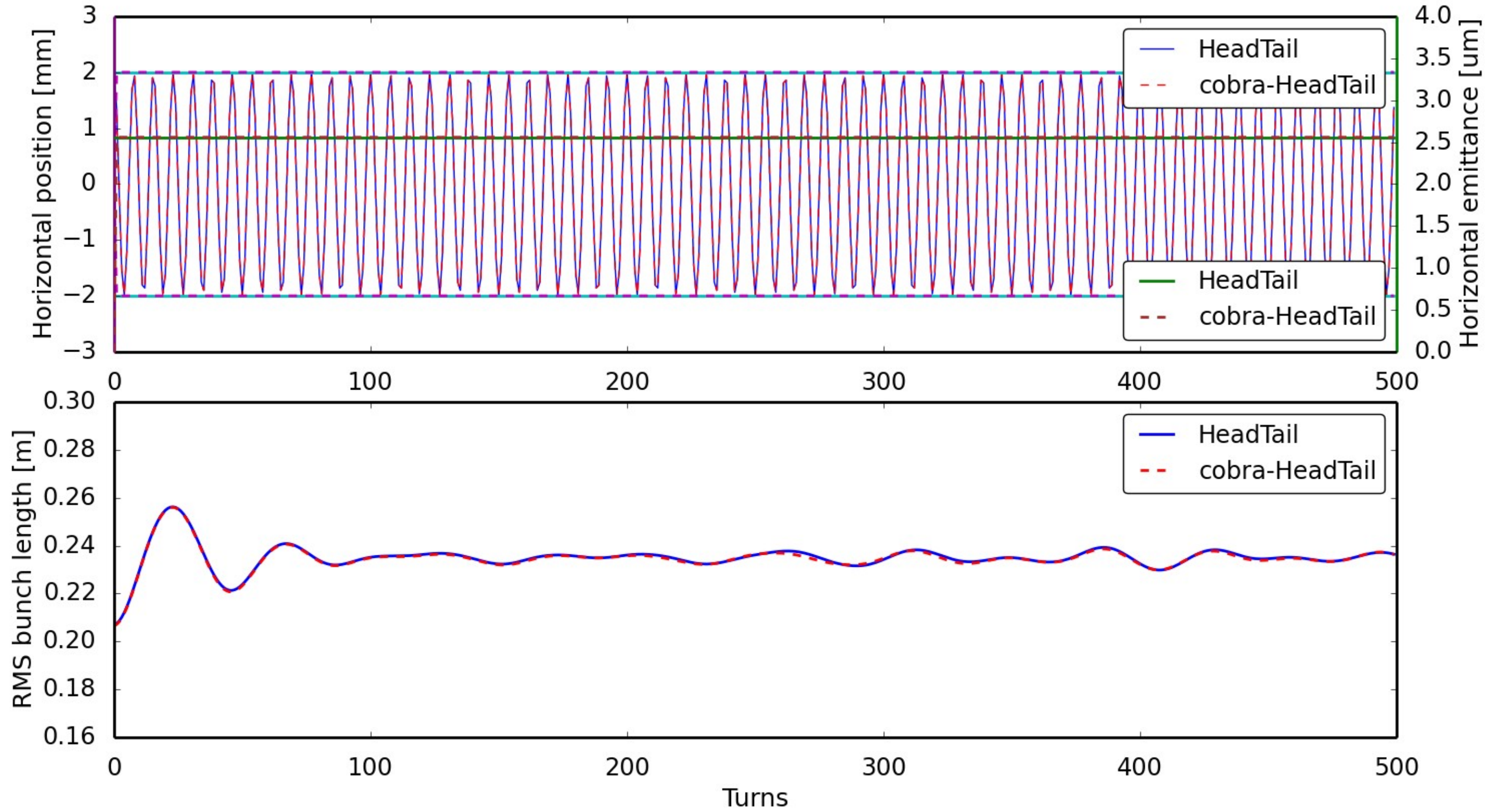
Benchmarks - tracking

Linear bucket – chromaticity – amplitude detuning



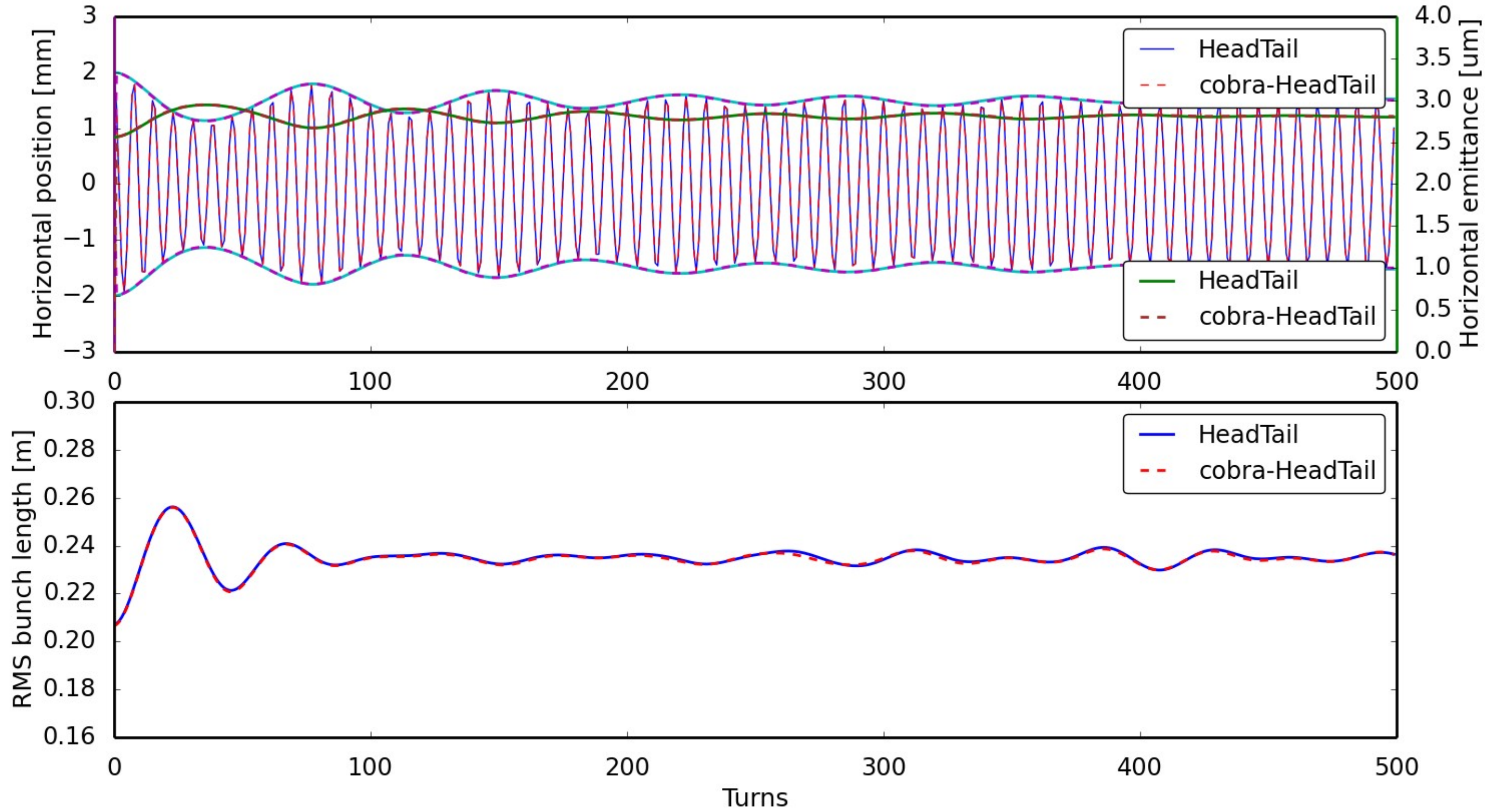
Benchmarks - tracking

Nonlinear bucket



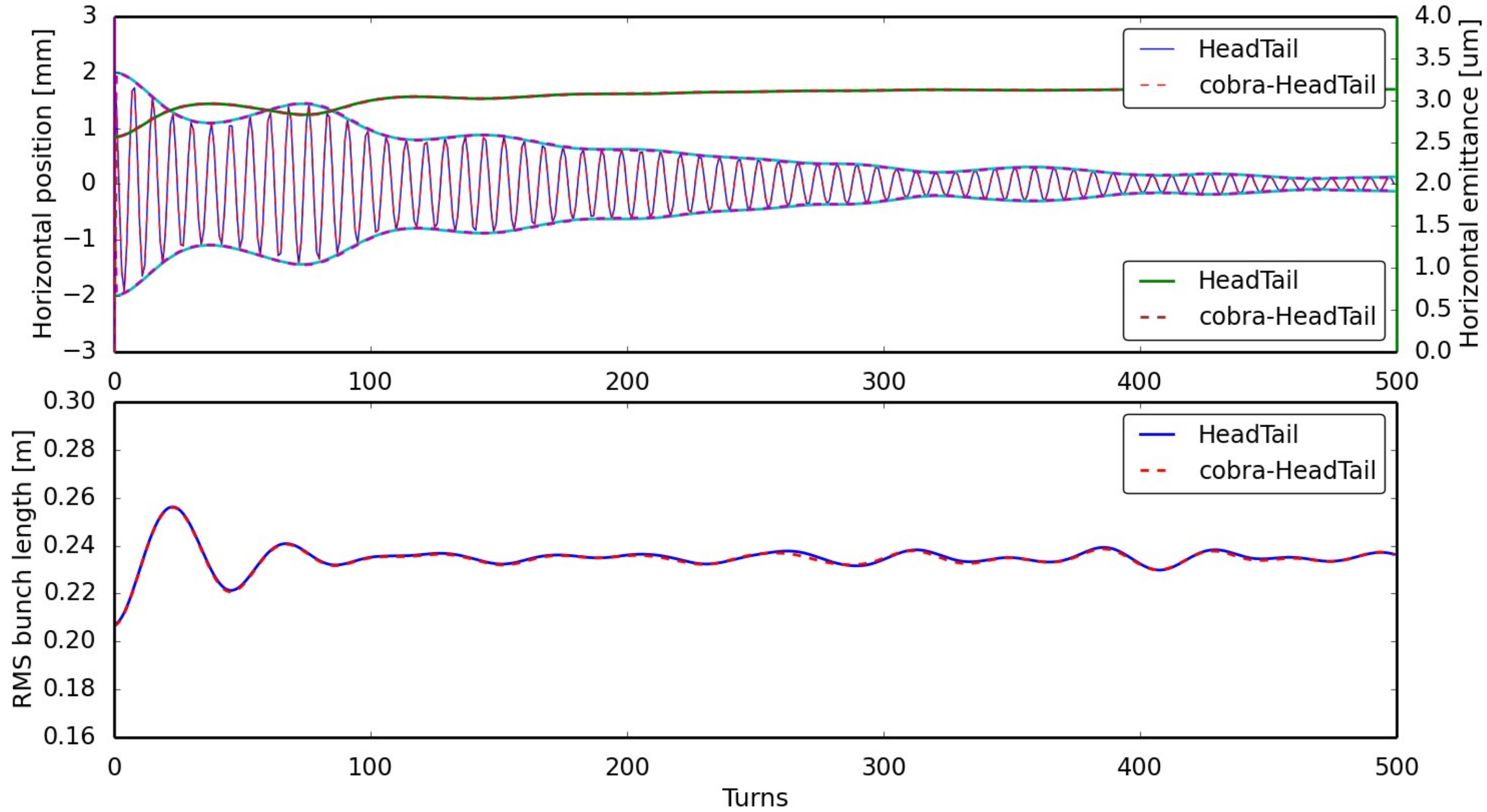
Benchmarks - tracking

Nonlinear bucket – chromaticity



Benchmarks - tracking

Nonlinear bucket – chromaticity – amplitude detuning



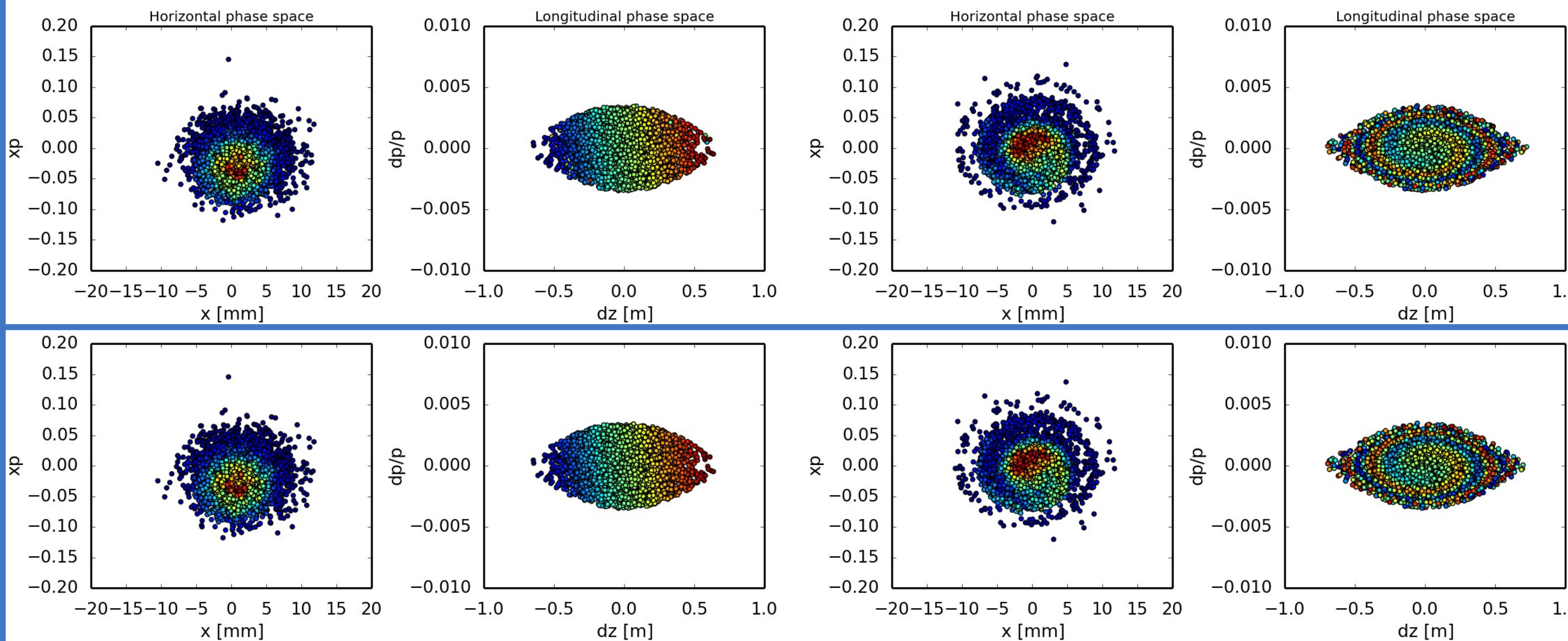
Benchmarks - tracking

Nonlinear bucket – chromaticity – amplitude detuning

Initial | after 350 turns

HeadTail

cobra-HeadTail



Poisson solver

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

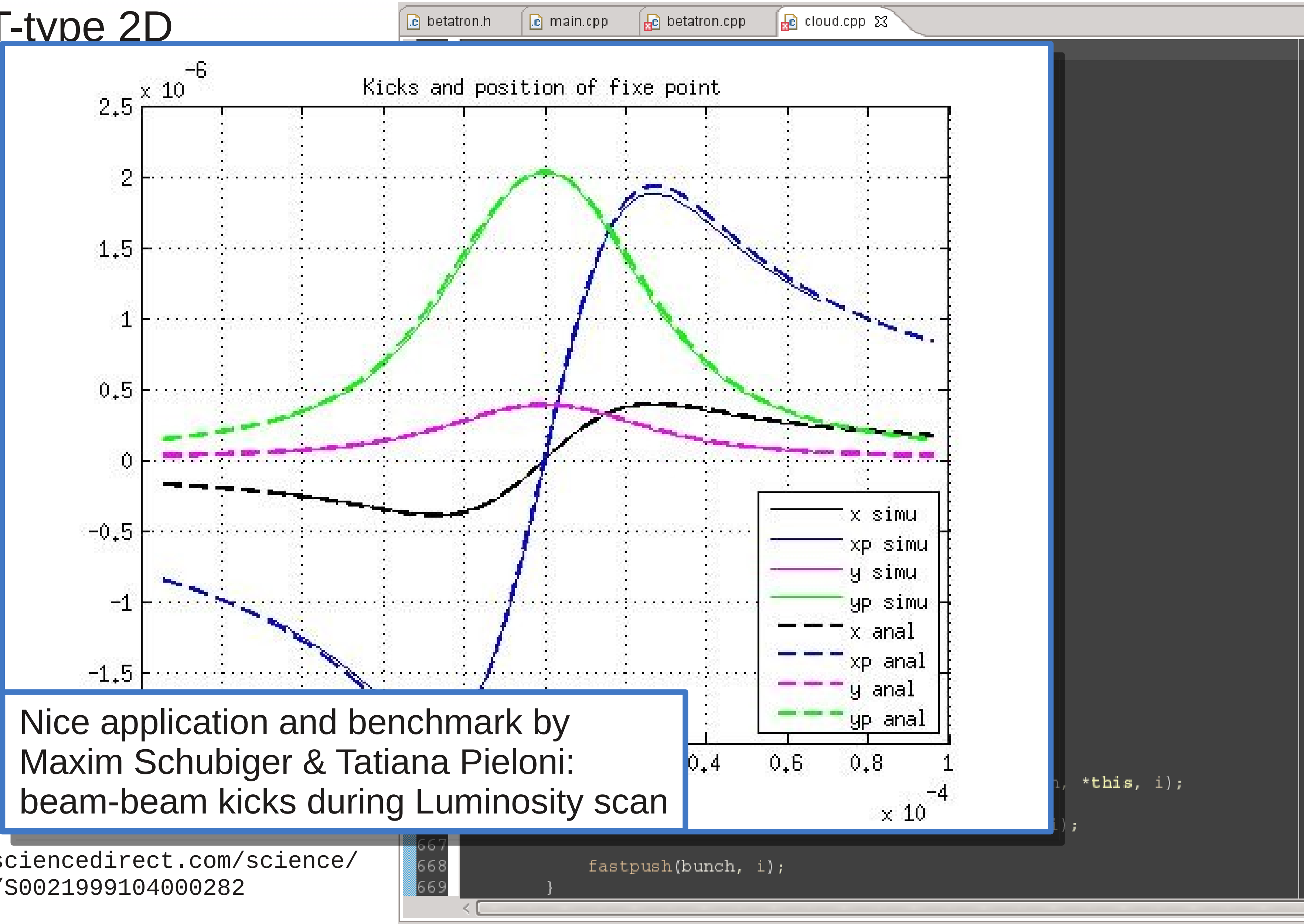
betatron.h | main.cpp | betatron.cpp | cloud.cpp

```
void Cloud::fastpush(Bunch& bunch, int i_slice)
void Cloud::track(Bunch& bunch)
{
    if (bunch.is_sliced)
    {
        rebuild(bunch);
        poisson.build<Bunch&, Cloud&>(bunch, *this);
        poisson.integratedGreen();

//          t0 = 0;
        for (size_t i=1; i<bunch.get_nslices() + 1; i++)
        {
#pragma omp parallel sections
{
#pragma omp section
{
            poisson.fastgather<Bunch&>(bunch, i);
#pragma omp critical
{
            poisson.computePotential<Bunch&>(bunch, i);
            poisson.computeFields<Bunch&>(bunch, i);
}
}
#pragma omp section
{
            poisson.fastgather<Cloud&>(*this, i);
#pragma omp critical
{
            poisson.computePotential<Cloud&>(*this, i);
            poisson.computeFields<Cloud&>(*this, i);
}
}
}
            poisson.parallelscatter<Bunch&, Cloud&>(bunch, *this, i);

            poisson.write<Bunch&, Cloud&>(bunch, *this, i);

            fastpush(bunch, i);
        }
```

[1] 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

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

[1] J. Qiang et particle-in- interaction colliders", http://www.sciencedirect.com/science/article/pii/S0021999104000282

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/reps/cobra cobra-demo`
- `mkdir cobra-demo-build; cd cobra-demo-build`
- `cmake ../cobra-demo/src    Makefile`
- `make    Libraries`

cobra-HeadTail demo

- Requir
  - C+-
  - C+-
  - boo
  - FFT
  - GS
  - HD
  - cma
  - Up-

- git … a-demo
- mkd
- cmak
- Make

```
kli@pcbe13989:~/workspace> git clone https://git.cern.ch/reps/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@pcbe13989:~/workspace> mkdir cobra-demo-build
kli@pcbe13989:~/workspace> cd cobra-demo-build/
kli@pcbe13989:~/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/intel6
4/icpc
-- Check for working CXX compiler: /home/kli/Software/Intel/opt/intel/composer_xe_2013.3.163/bin/intel6
4/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@pcbe13989:~/workspace/cobra-demo-build> make
Scanning dependencies of target Bunch
[  4%] Building CXX object CMakeFiles/Bunch.dir/bunch.cpp.o
```

cobra-demo-build : make

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?