PYHEADTAIL

Multi-bunch status update

J. Komppula, K. Li, G. Rumolo, M. Schenk

Brief reminder

- First multi-bunch PyHEADTAIL version completed in summer 2016 (s. PyHEADTAIL Meeting #12)

- Parallelized multi-bunch PyHEADTAIL version benchmarked during scrubbing run 2017 (along with test suite) – tests done on laptops/desktop PCs

- With the availability of the CERN HPC cluster, first time deployment of multi-bunch PyHEADTAIL for large scale problems → possibility to evaluate scaling

- Several bottlenecks identified leading to poor scaling. Parallel activities on multi-bunch feedback finally led to synergies and significant optimizations of the parallel multi-bunch PyHEADTAIL version (→ Jani)

- Review:
    - Basic principles
    - Modifications/additions

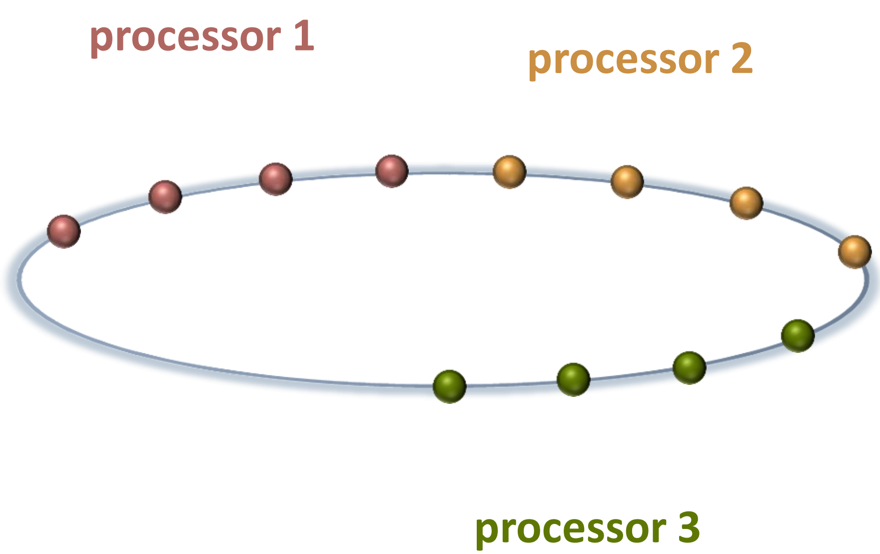
Parallelization

- OpenMP
  - Multithreading **simple loops** (trig. functions evaluation, dot products etc.)

Parallelization

- OpenMP
  - Multithreading **simple loops** (trig. functions evaluation, dot products etc.)
- MPI
  - Where strongly memory limited, e.g. **multi bunch wakes and feedback**

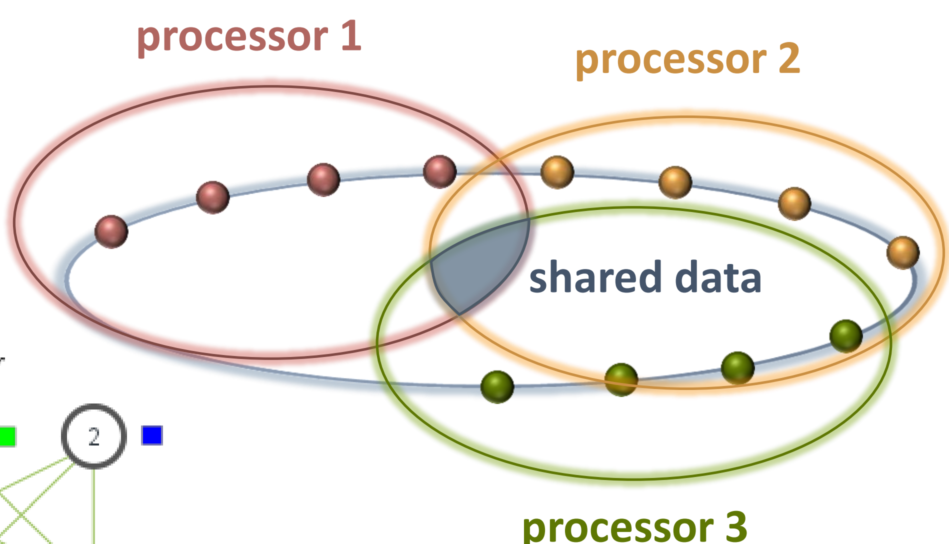
1. Perform parallel tracking for all bunches

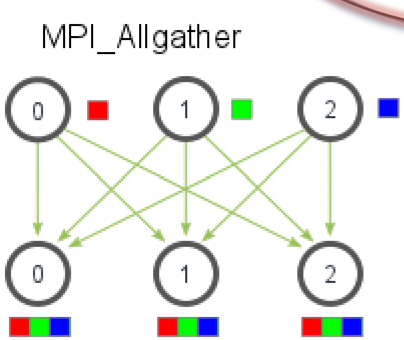


Parallelization

- OpenMP
  - Multithreading **simple loops** (trig. functions evaluation, dot products etc.)
- MPI
  - Where strongly memory limited, e.g. **multi bunch wakes and feedback**

1. Perform parallel tracking for all bunches
2. Collect slice data from all other processes

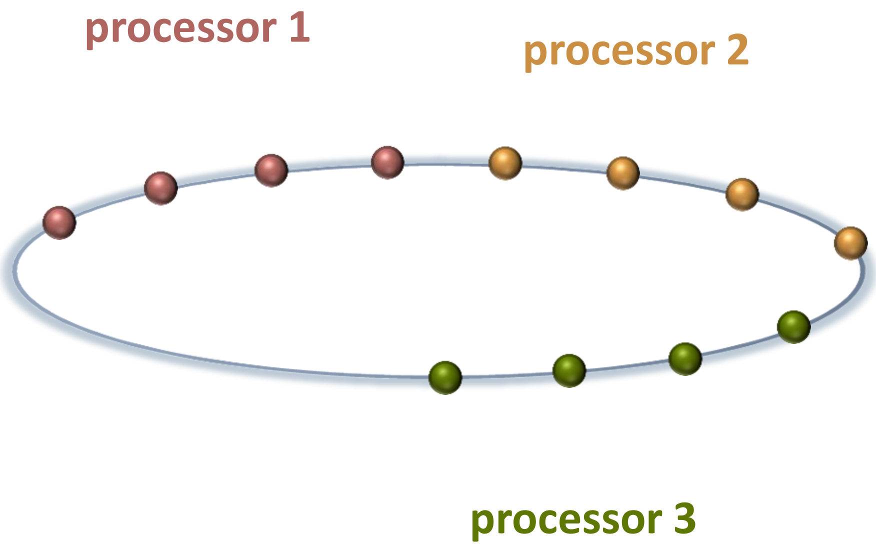




Parallelization

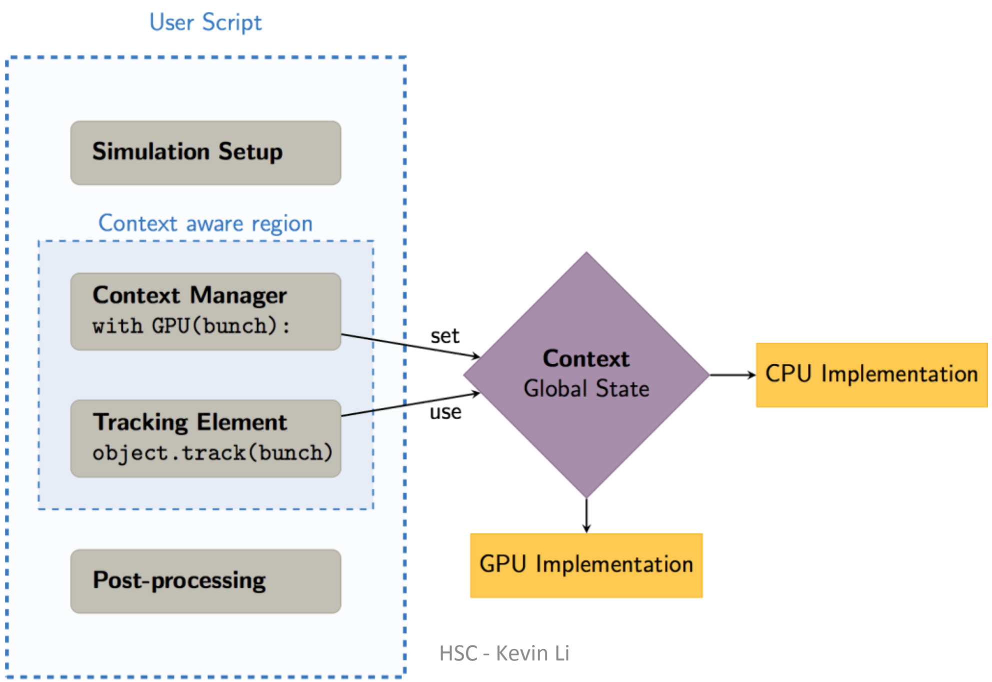
- OpenMP
  - Multithreading **simple loops** (trig. functions evaluation, dot products etc.)
- MPI
  - Where strongly memory limited, e.g. **multi bunch wakes and feedback**

1. Perform parallel tracking for all bunches
2. Collect slice data from all other processes
3. Continue parallel tracking for all bunches



Parallelization

- OpenMP
  - Multithreading **simple loops** (trig. functions evaluation, dot products etc.)
- MPI
  - Where strongly memory limited, e.g. **multi bunch wakes and feedback**
- CUDA
  - Parallelization of simple loops via context manager
  - Space charge PIC solvers



Overview

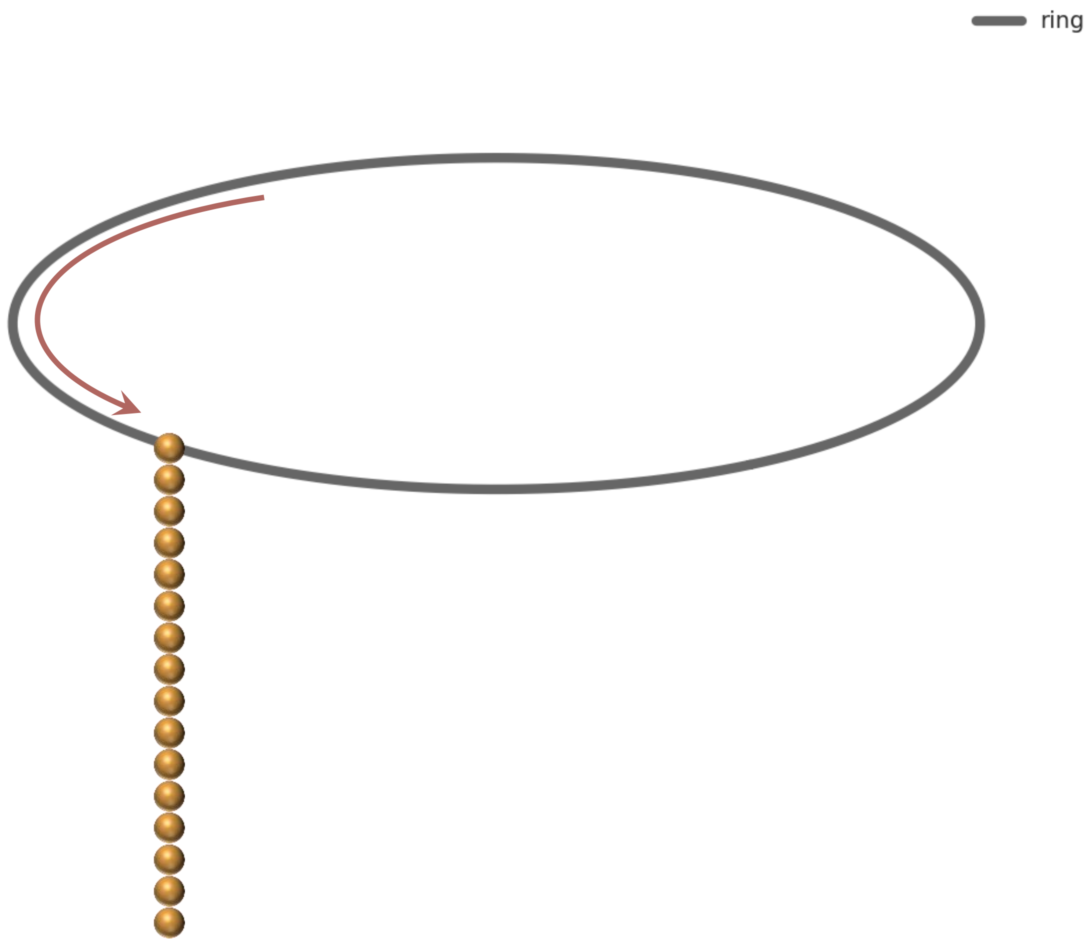
- Particle beams generation based on multiple bunches generation (via **list of bunch parameters** and **filling scheme**)

- Possibility via **bunch_id** to quickly extract and re-insert individual bunches from and back into beam

- Parallel wake kick computations are **broadcasting only slice data** - MPI management done via new mpi_data module which takes care of layout and distribution of this slice data

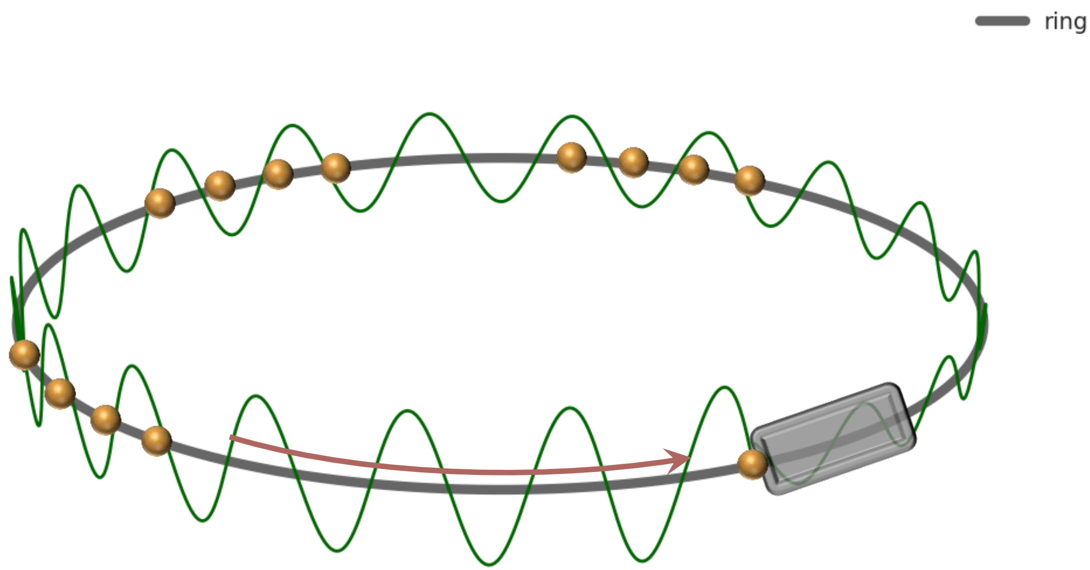
Parallel tracking

- Macroparticle tracking in phase space → **independent and embarrassingly parallel**



Parallel tracking

- Macroparticle tracking in phase space → **independent and embarrassingly parallel**



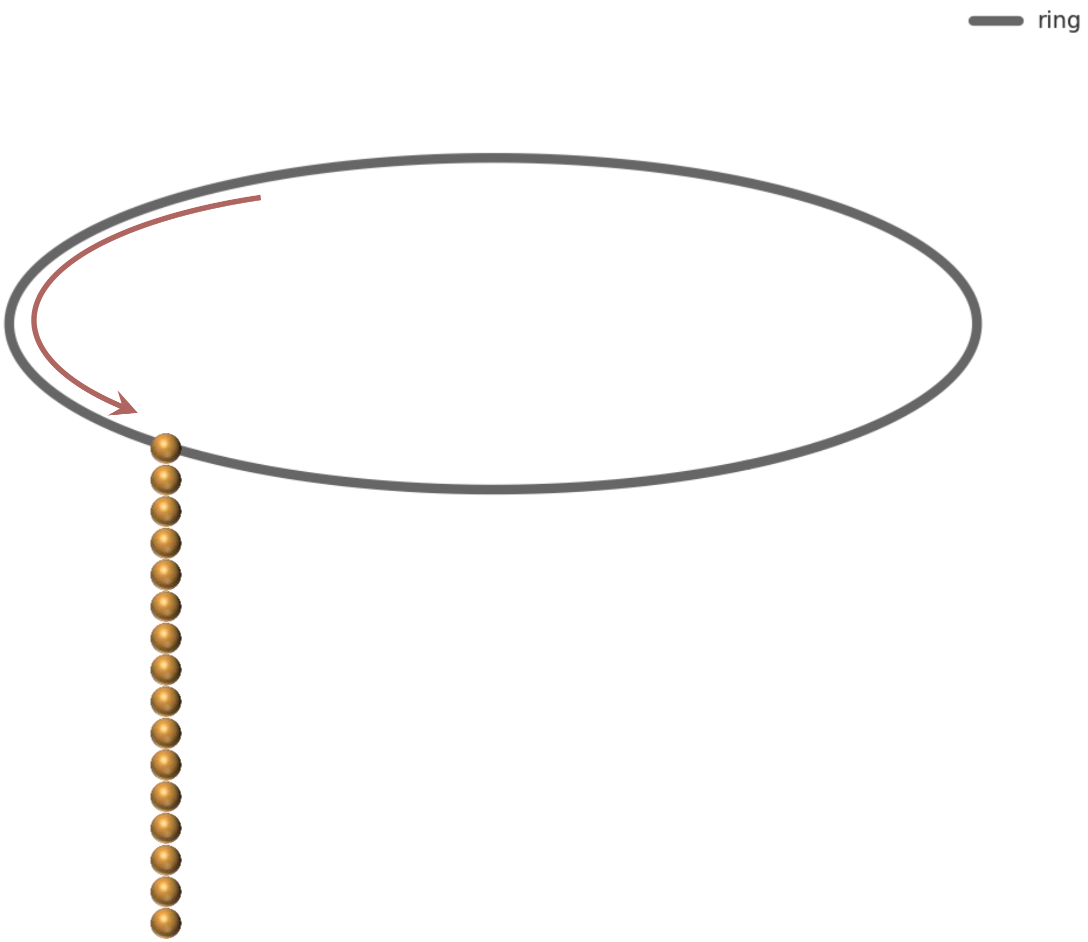
- Wake field kicks → convolution <M, W>
  - Every slice acts as source for every other slice → **collective interaction** among slices via the wake field

$$\Delta x_i' = -\frac{e^2}{m\gamma\beta^2c^2} \times \sum_{j=0}^{\text{n\_slices}} \boxed{M_{x,y}[z_j]} \cdot \boxed{W_\perp[z_i - z_j]} \cdot f(x_i, y_i)$$

knowledge of **slice moments** and **locations** needs to be shared

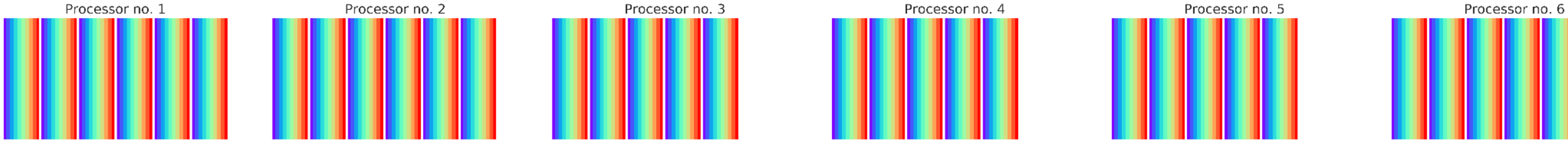
Parallel tracking

- Macroparticle tracking in phase space → **independent and embarrassingly parallel**



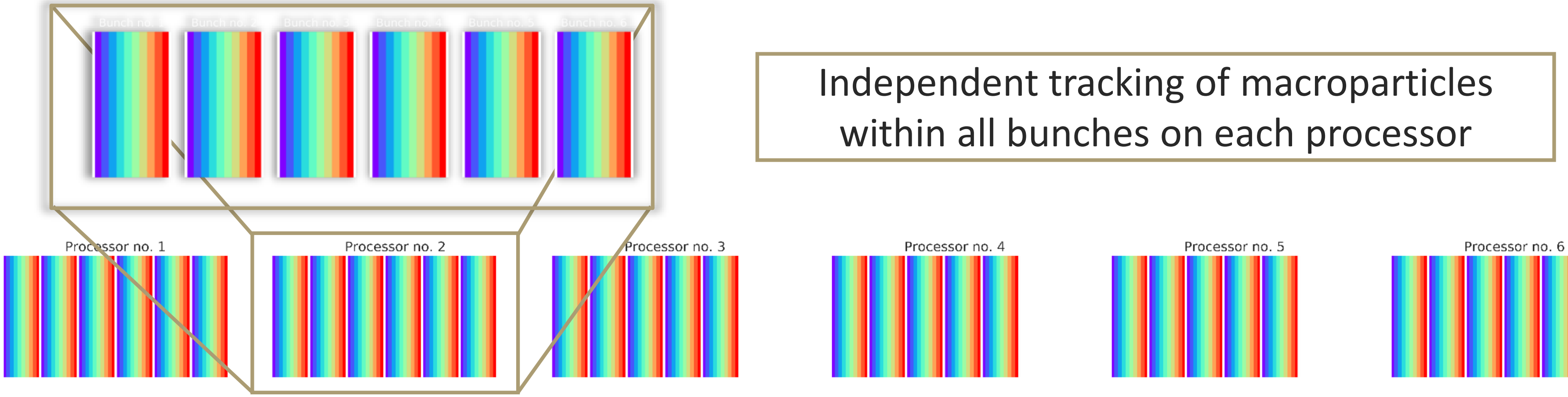
Parallel structures

- Multi-bunch beam generation (via binary-tree algorithm → 10s for full LHC)



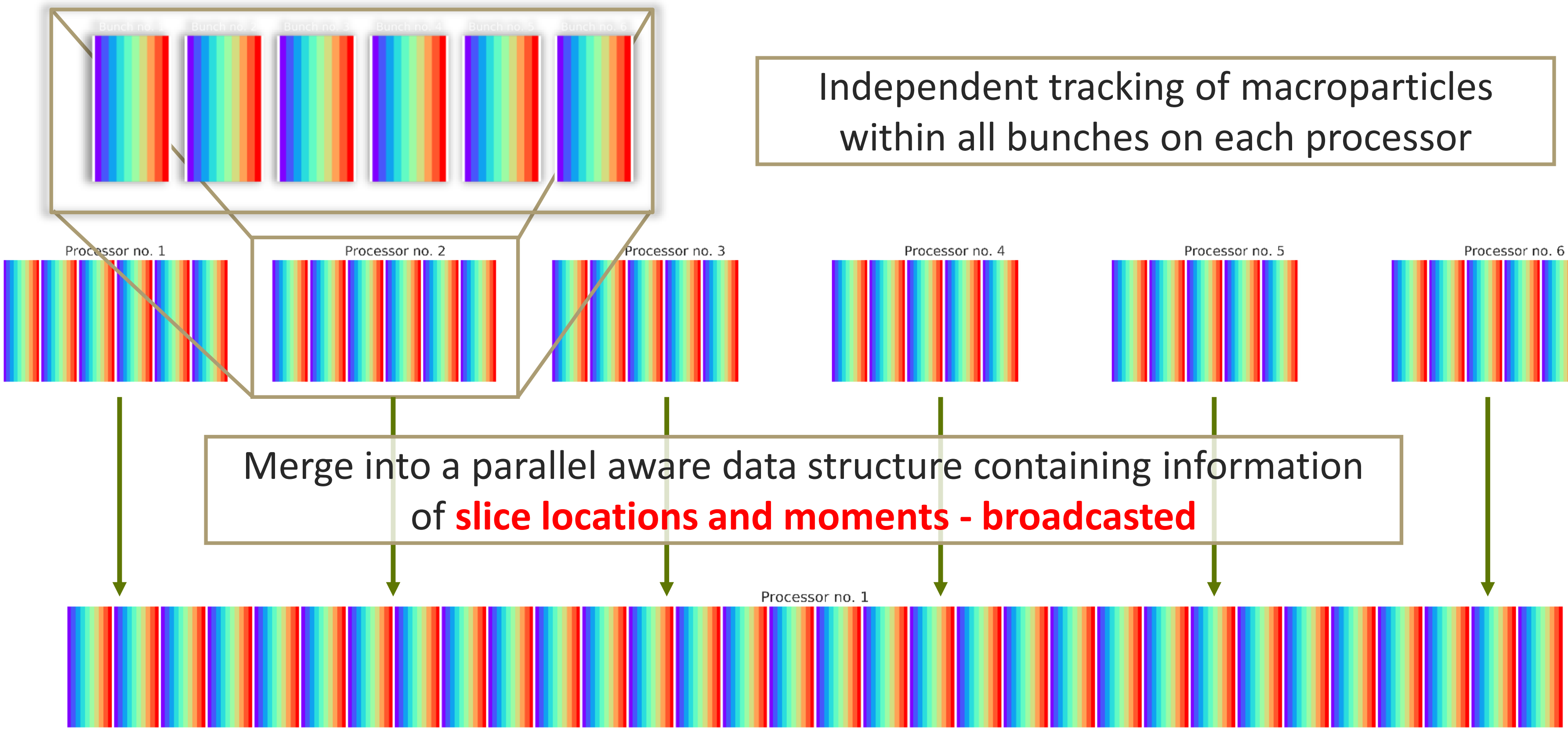
Parallel structures

- Multi-bunch beam generation (via binary-tree algorithm → 10s for full LHC)



Parallel structures

- Multi-bunch beam generation (via binary-tree algorithm → 10s for full LHC)

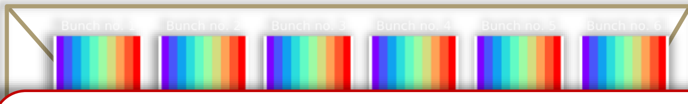


$$\Delta x_i' = -\frac{e^2}{m\gamma\beta^2c^2} \times \sum_{j=0}^{\text{n\_slices}} M_{x,y}[z_j] \cdot W_\perp[z_i - z_j] \cdot f(x_i, y_i)$$

knowledge of **slice locations and moments** needs to be shared

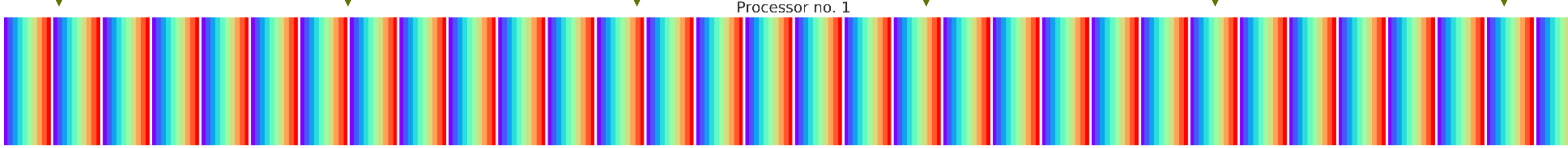
Parallel structures

- Multi-bunch beam generation (via binary-tree algorithm → 10s for full LHC)



Communication via an MPI_Allgather call

Merge into a parallel aware data structure containing information of **slice locations and moments - broadcasted**

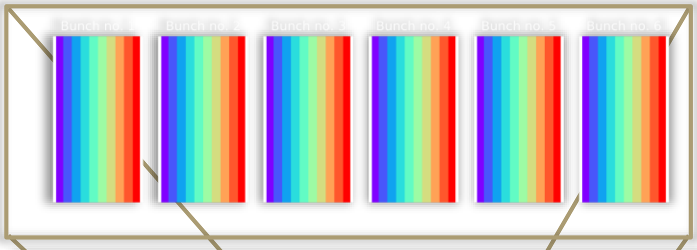


$$\Delta x_i' = -\frac{e^2}{m\gamma\beta^2c^2} \times \sum_{j=0}^{\mathrm{n\_slices}} M_{x,y}[z_j] \cdot W_\perp[z_i - z_j] \cdot f(x_i, y_i)$$

knowledge of **slice locations and moments** needs to be shared

Parallel structures

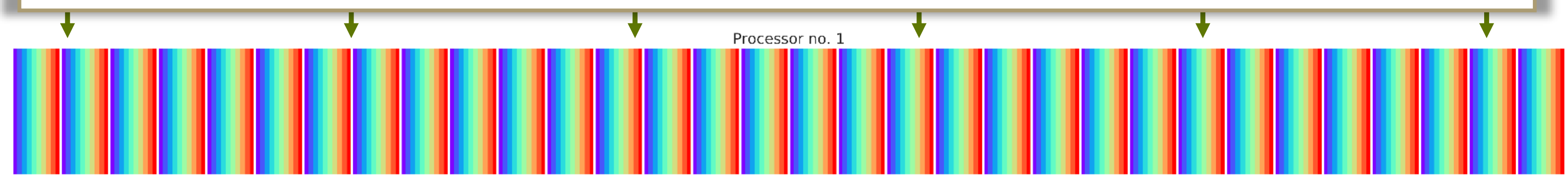
- Multi-bunch beam generation (via binary-tree algorithm → 10s for full LHC)



Independent tracking of macroparticles within all bunches on each processor

Once broadcasted, perform computation of wake kicks for all slices on every processor:

```
for zt in target_bunches:
    for k in xrange(n_turns):
        for zs in source_bunches:
            accumulated_kicks_t += ~ <M(zs), W(zt - zs - kC)>
```

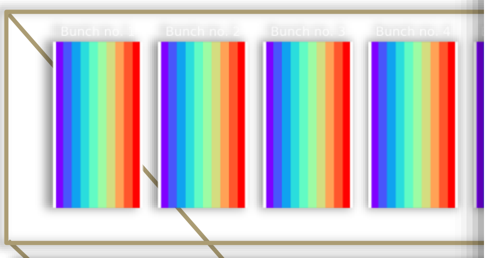


$$\Delta x'_i = -\frac{e^2}{m\gamma\beta^2 c^2} \times \sum_{j=0}^{n\_slices} M_{x,y}[z_j] \cdot W_\perp[z_i - z_j] \cdot f(x_i, y_i)$$

knowledge of **slice locations and moments** needs to be shared

lel structures

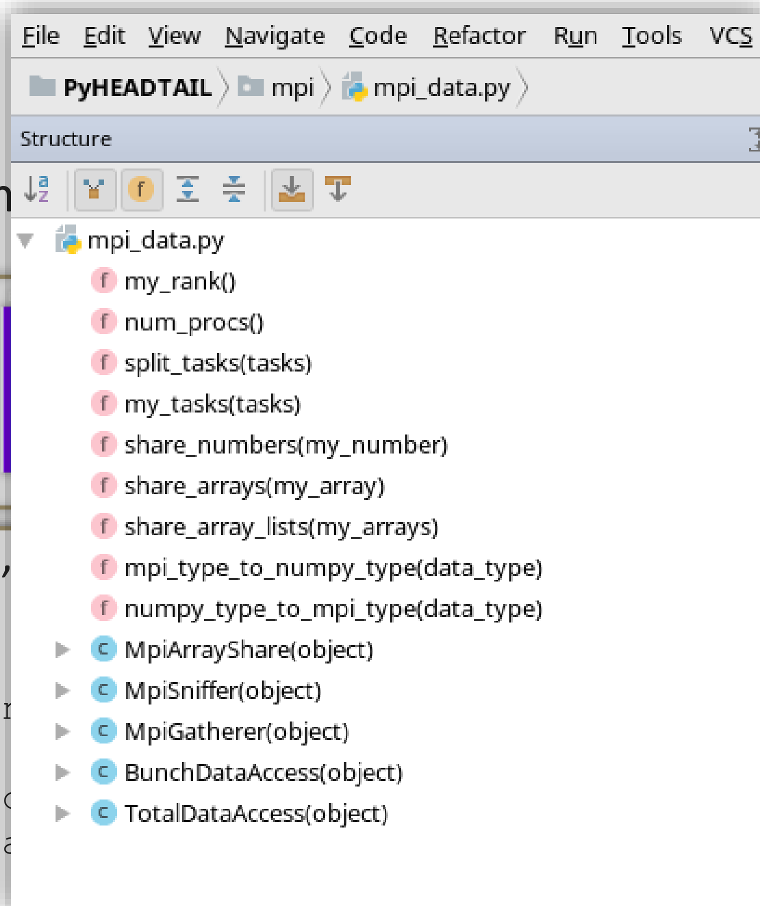
• Multi-bunch beam … nm → 10s for full LHC)



…ing of macroparticles
…s on each processor

Once broadcasted, … r all slices on every

```
for zt in target_bu…
    for k in xrange…
        for zs in sc…
            accumula… - kC)>
```

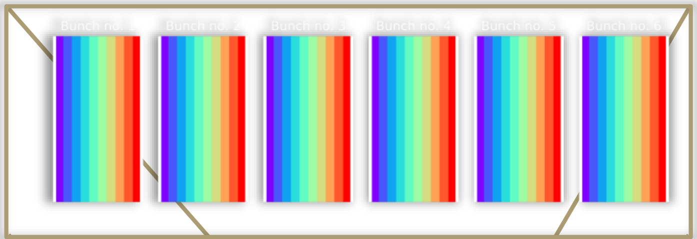


All encapsulated and managed nicely via new MPI_data module

knowledge of slice locations and moments needs to be shared

Parallel structures

- Multi-bunch beam generation (via binary-tree algorithm → 10s for full LHC)



Independent tracking of macroparticles within all bunches on each processor

Once broadcasted, perform computation of wake kicks for all slices on every processor:

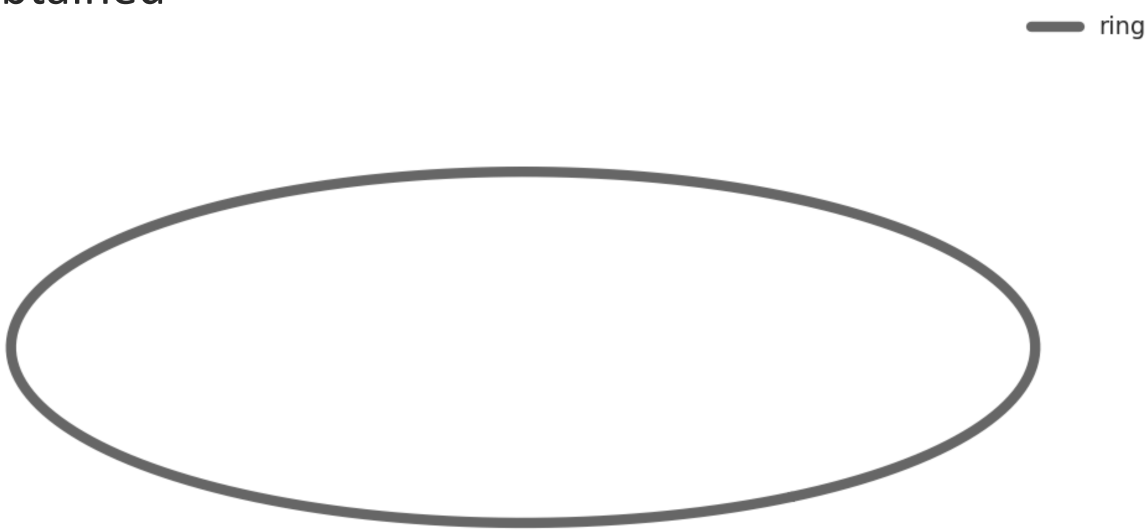
```
for zt in target_bunches:
    for k in xrange(n_turns):
        for zs in source_bunches:
            accumulated_kicks_t += ~<M(zs), W(zt − zs − kC)>
```

Generic and robust solution, but expensive as number of bunches increases. More room for optimization…

knowledge of slice locations and moments needs to be shared

Optimizations

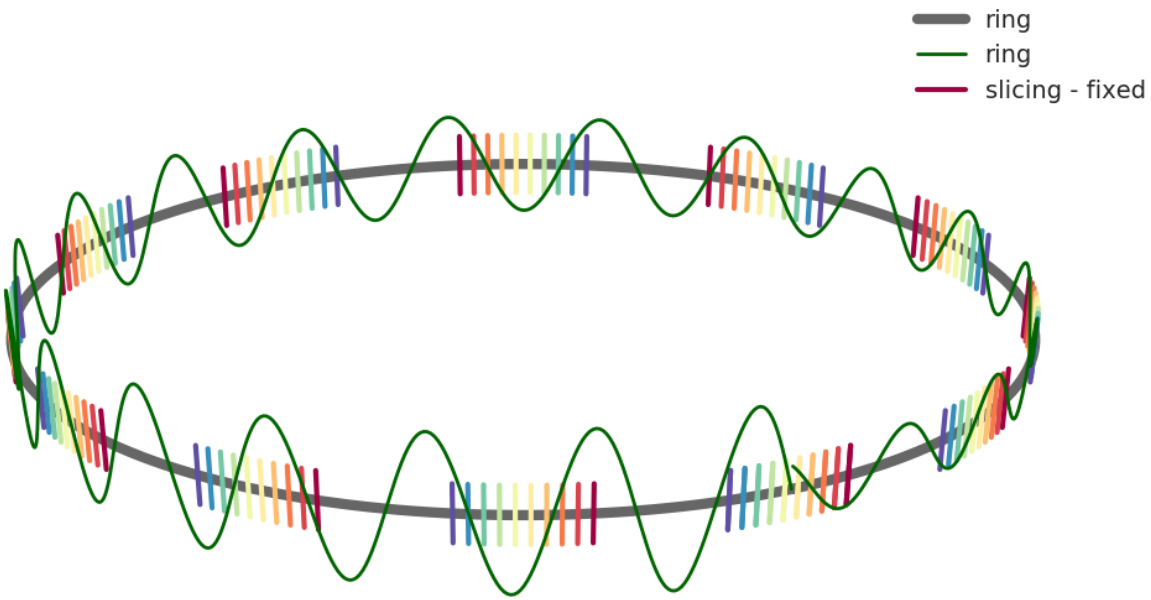
- With a slight **change of perspective/reference frame**, significant speed-ups can be obtained



Change of perspective!
Freeze coordinate system as given by ring geometry.

Optimizations

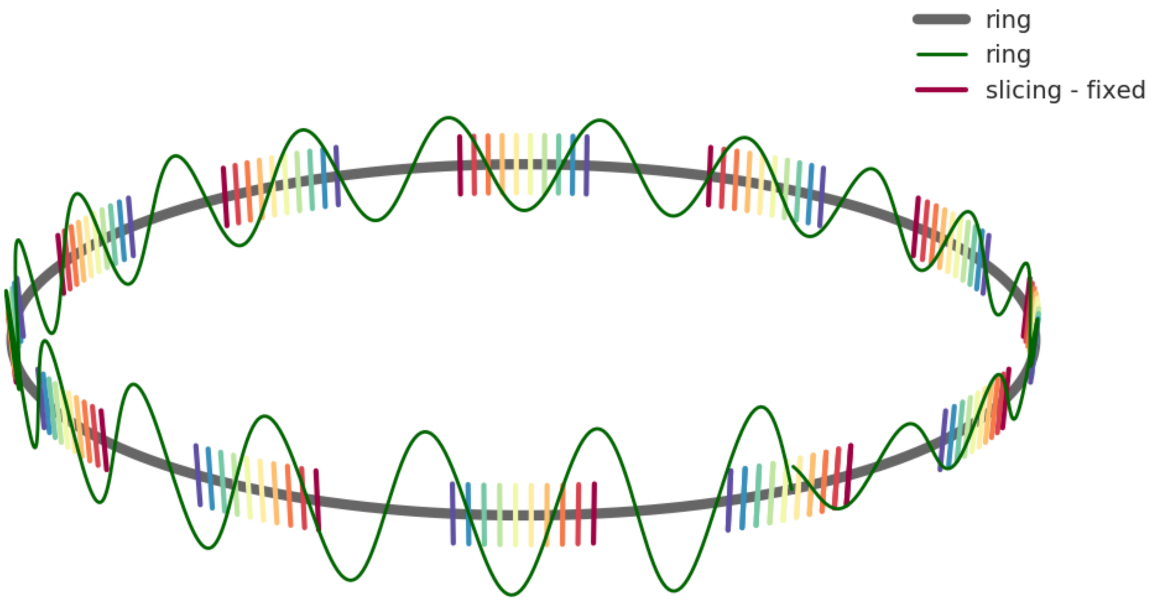
- With a slight **change of perspective/reference frame**, significant speed-ups can be obtained



Change of perspective!
Freeze coordinate system as given by ring geometry.
Slicing determined entirely by ring geometry.

Optimizations

- With a slight **change of perspective/reference frame**, significant speed-ups can be obtained



- We identify a **bunch harmonic number (i.)** defined by the RF frequency (synchrotron harmonic number) and the bunch spacing (i.e. 3564 for the LHC or 924 for the SPS)
- We sample the wake function at **fixed slices (ii.)** around the bunch harmonic number locations. This generates a **reusable wake function look-up table**…

Optimizations

- With a slight **change of perspective/reference frame**, significant speed-ups can be obtained

ring
ring
slicing - fixed

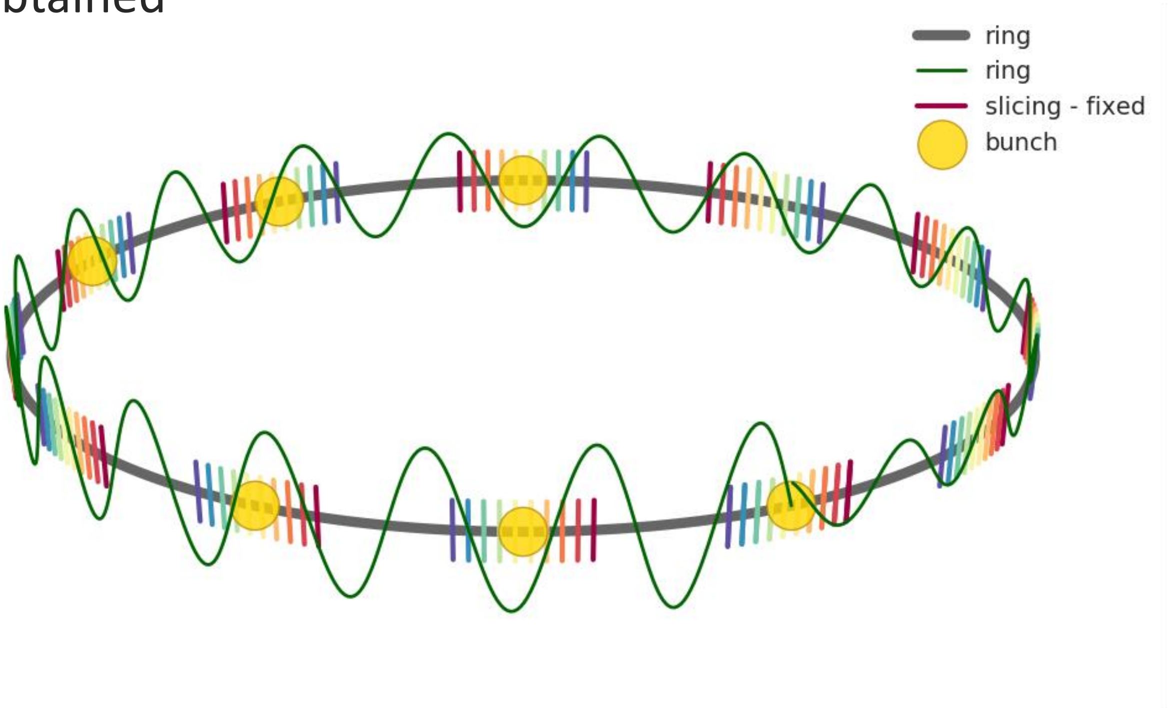
\ slice _ix h_ix	0	1	2	3	4	5	6
0	…	…	…	…	…	…	…
1	…	…	…	…	…	…	…
2	…	…	…	…	…	…	…
3	…	…	…	…	…	…	…
4	…	…	…	…	…	…	…
5	…	…	…	…	…	…	…
6	…	…	…	…	…	…	…

- W … ncy (s … the LHC or 924 for the SPS)

- We sample the wake function at **fixed slices (ii.)** around the bunch harmonic number locations. This generates a **reusable wake function look-up table**…

Optimizations

- With a slight **change of perspective/reference frame**, significant speed-ups can be obtained



- Filling individual buckets with bunches, we can now **deploy the FFT convolution** which scales much better, effectively moving from O(n^2) to O(n log(n))

Parallel version – catches

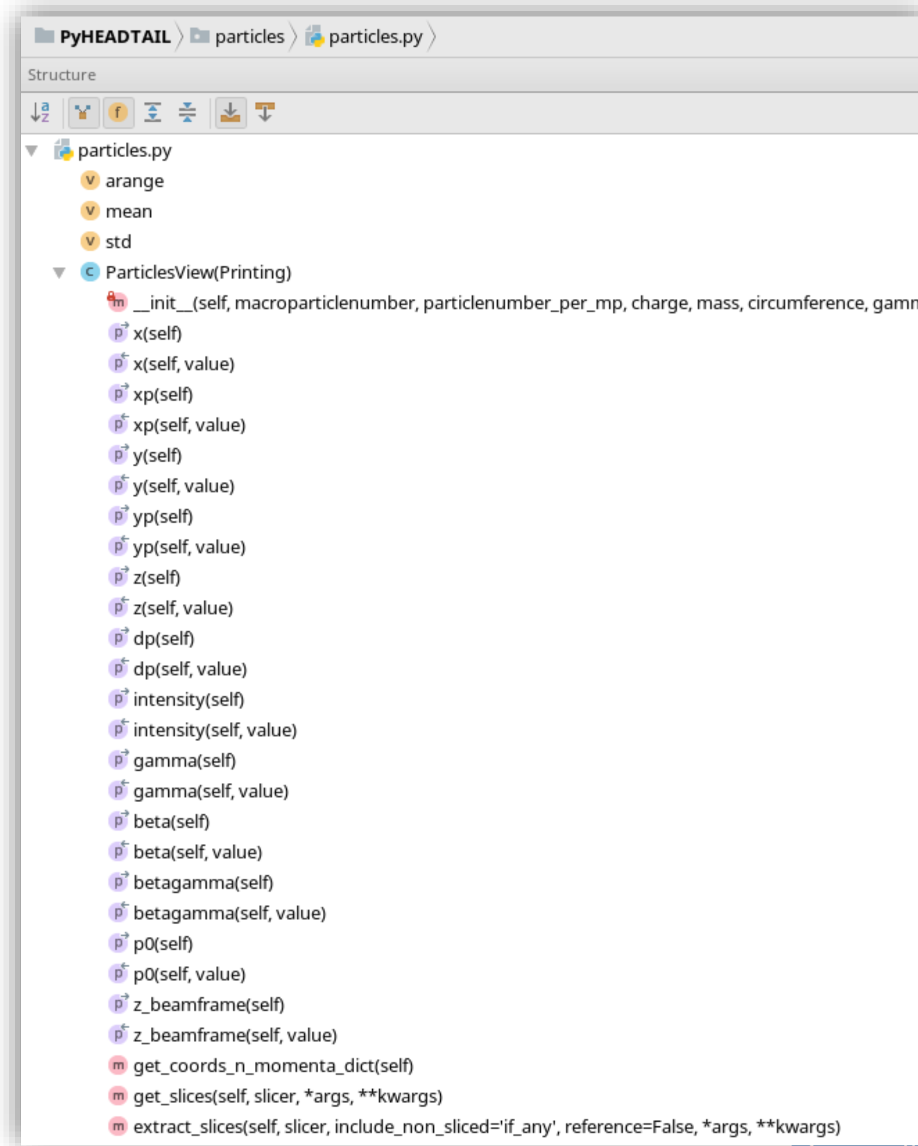
- Some splitting and merging required (→ in particular for slicing)

- Extract bunches was slow… in comparison to rather fast tracking/convolution

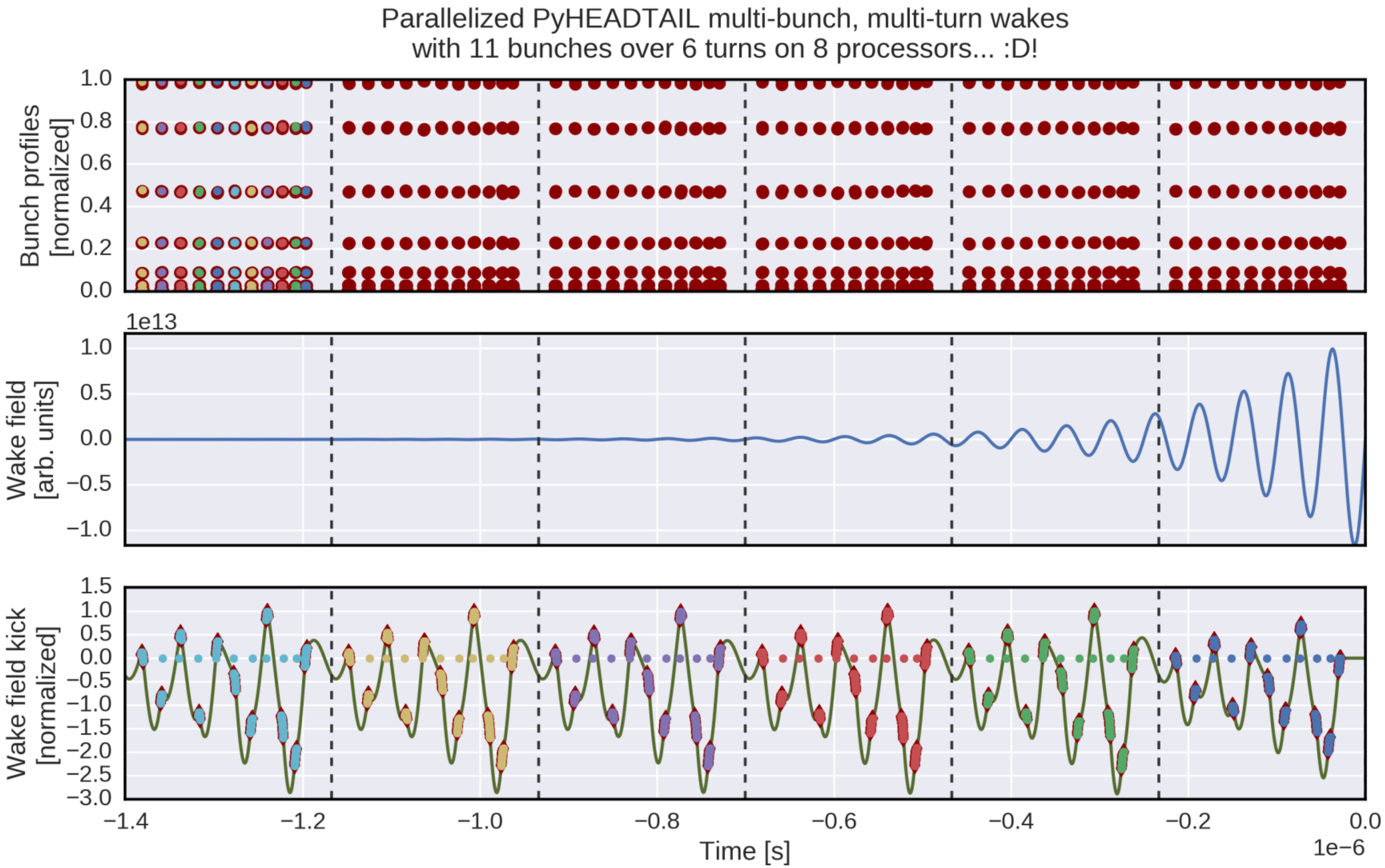
  - Improved using split function which internally deploys masks via a bunch index – still requires copying of lots of data.

  - Final improvement **using memory views** (like pointers), instead → bunch is a specific view on a portion of the beam. No copying required and thus very fast.

  - Need to understand how to handle this when trying to integrate GPU support for these features.



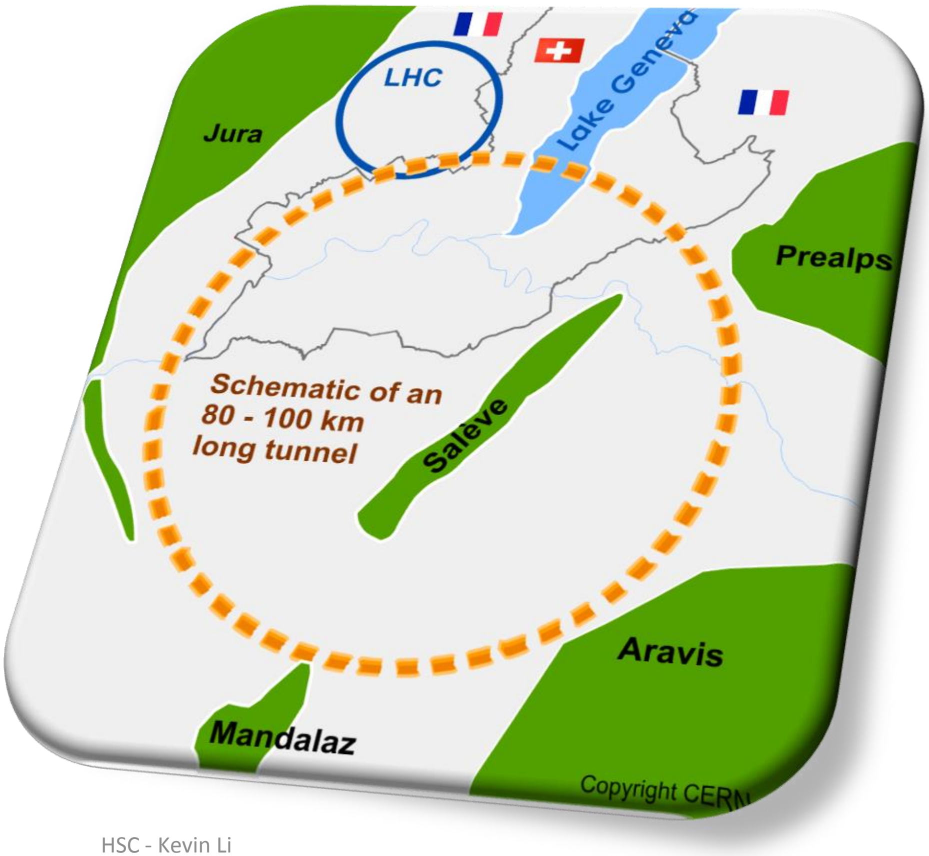
Parallel version – tests



Motivation

- Transverse feedback studies for the FCC-hh
  - Coupled bunch instabilities, injection oscillations, etc

- Realistic models for a beam and a transverse feedback system
  → PyHEADTAIL
  - New feedback module
  - Multibunch PyHEADTAIL



Challenges

Typical single bunch PyHEADTAIL simulation:

- 500k macro particles
- 100 slices
- Qp
- wakes

→ ~0.5 s / turn, 500k turns

	LHC	FCC-hh
Beam energy	7 TeV	50 TeV
Circumference	26.7 km	97.7 km
Betatron tunes	64.28/59.31	111.31/109.32
RF harmonic	35640	130680
Max N bunches	3564	13068

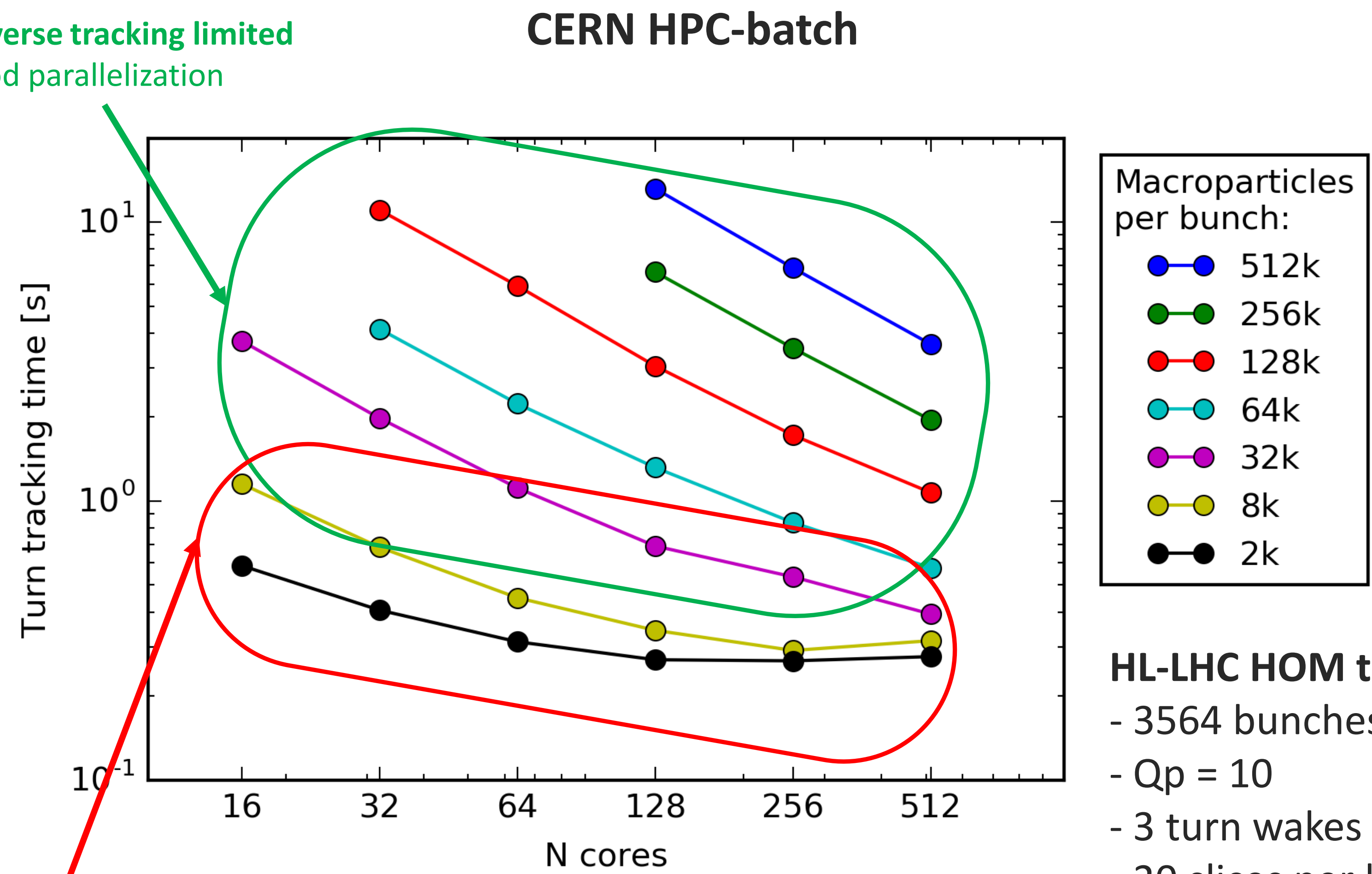
- Linear scaling to the full LHC simulation:
  → 600 cores, 3 s / turn (practical limit)
- Local 4 core simulations → ~1000 bunches per core
  → every 1 ms per bunch → 1 s in the total tracking time

This is not easy but...

- ... O(n^2) algorithms are the absolute evil:
  - Each bunch interacts with all the other bunches → O(n^2) algorithm!
  - 13 068 b ^2 = 170 000 000 times slower! (years per turn)

Scaling – tests

Transverse tracking limited
→ good parallelization

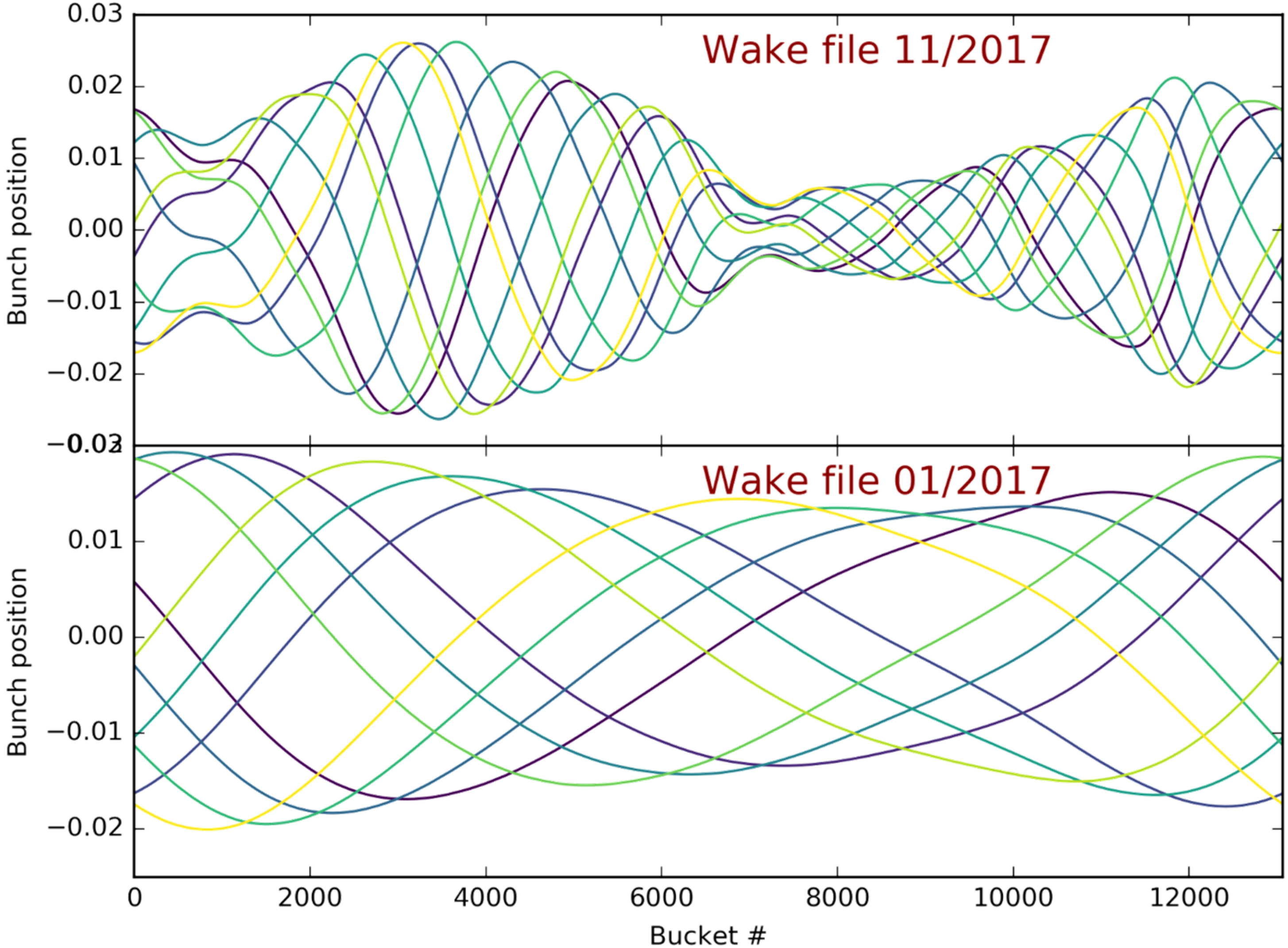


Wake limited
→ poor parallelization

HL-LHC HOM test:
- 3564 bunches
- Qp = 10
- 3 turn wakes
- 20 slices per bunch

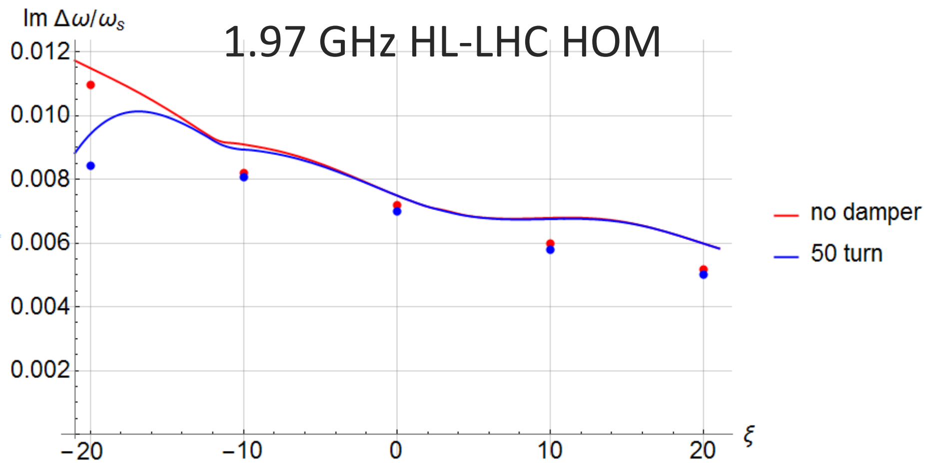
Scaling – tests

FCC impedance model with 13068 bunches – turn-by-turn snapshots



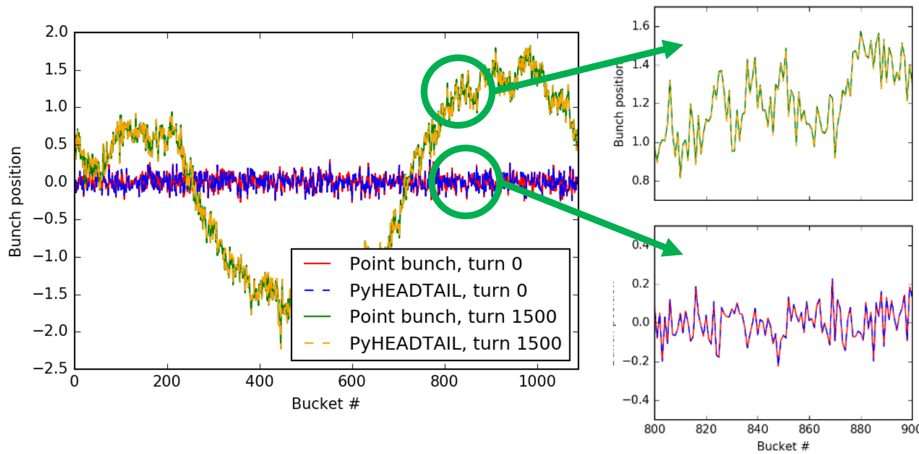
Benchmarking – tests

A) PyHEADTAIL vs NHT vs theory



B) PyHEADTAIL wakes vs math

Minimalistic point-like-bunch code
(tracking and wakes ~10 lines of code)



Cumulative numerical error <0.5% after 1500 turns

C) PyHEADTAIL wakes vs Sacherer

A factor of ~1.5 difference between Sacherer and PyHEADTAIL

However:

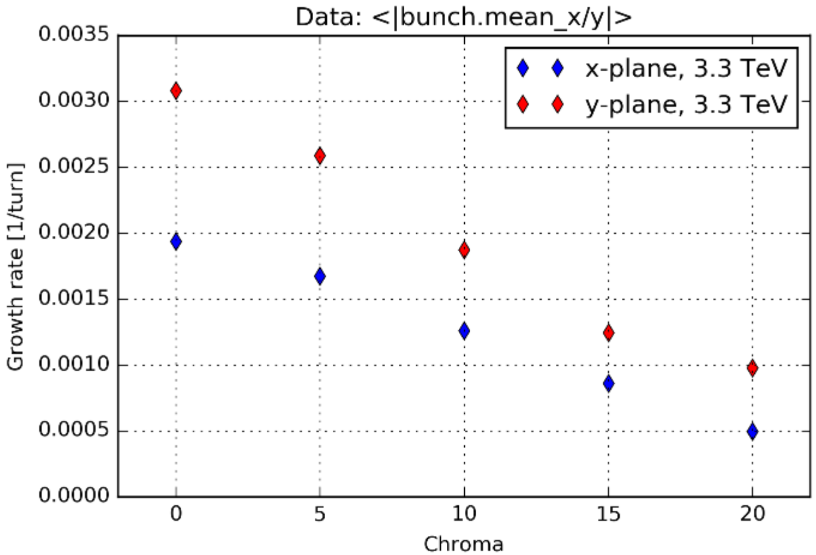
PyHEADTAIL wake kick ≈ coef * iFFT(FFT(mean_x) * FFT(wake function))

≈/= Sacherer modes ≈ beam impedance

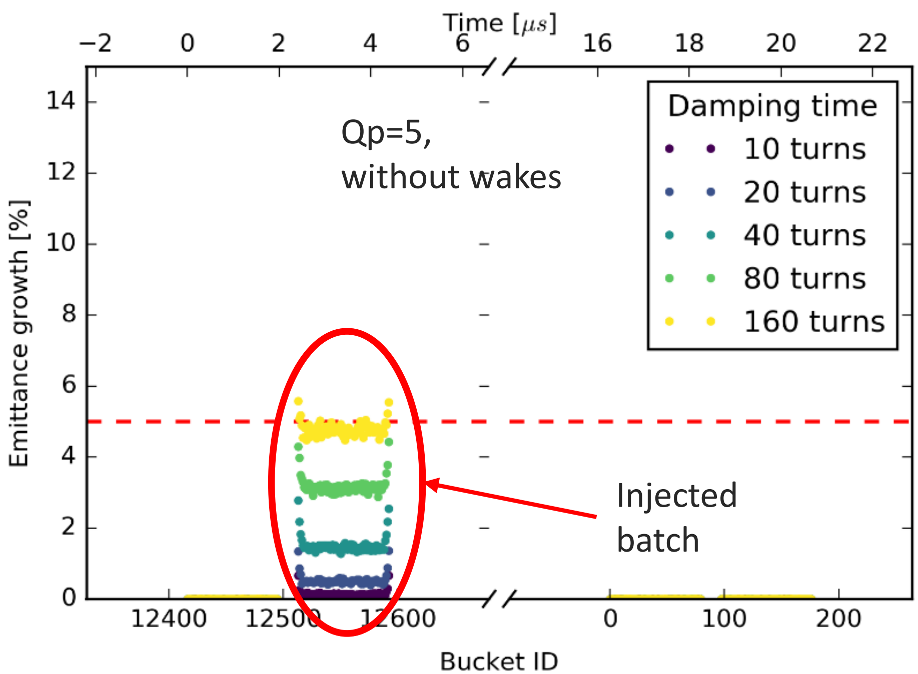
Hypotheses: The difference can be understood analytically

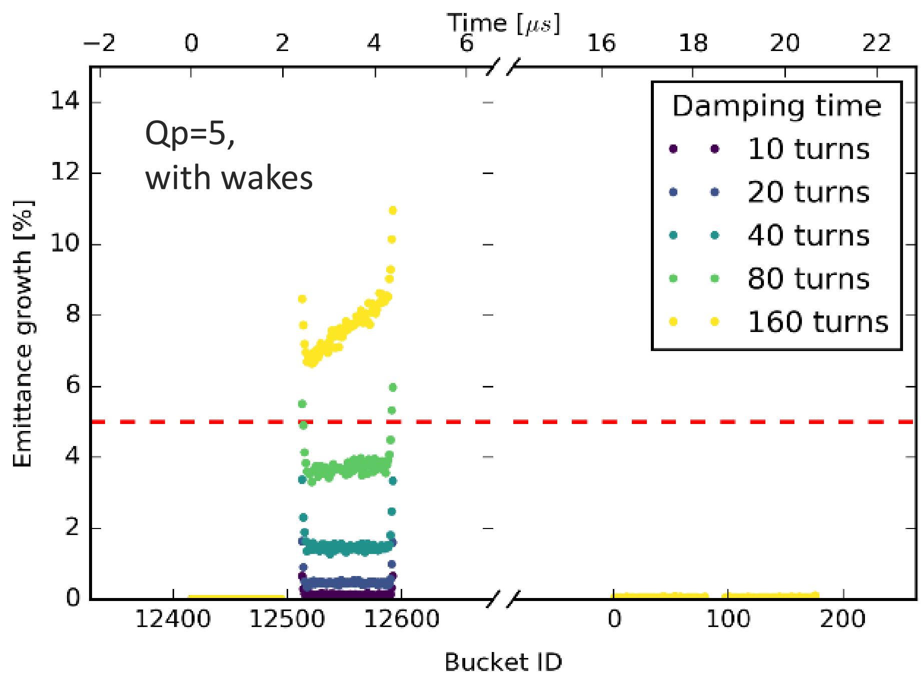
FCC-hh studies

Full FCC-hh beam chroma scan



Injection emittance growth





Status

- Performance is sufficient
  - Full SPS beam → **No problem!**
  - LHC MD studies → **Easy!**
  - LHC full beam → From **easy** (10k ppb)
    to **if you really need** (500k ppb)
  - FCC-hh → **It was the goal!**
- Benchmarking promising
  - Against other codes → **OK**
  - Against math → **OK**
  - Against Sacherer → **In progress**
- … but the multibunch version is still a development branch
  - Bunch monitors are sometimes unstable
  - Behind the master branch
  - Changes to the core functions were needed :(
  - Some parts might need cleaning and reprogramming

→ **merging needs some work**

Backup