Introduction to PyHEADTAIL

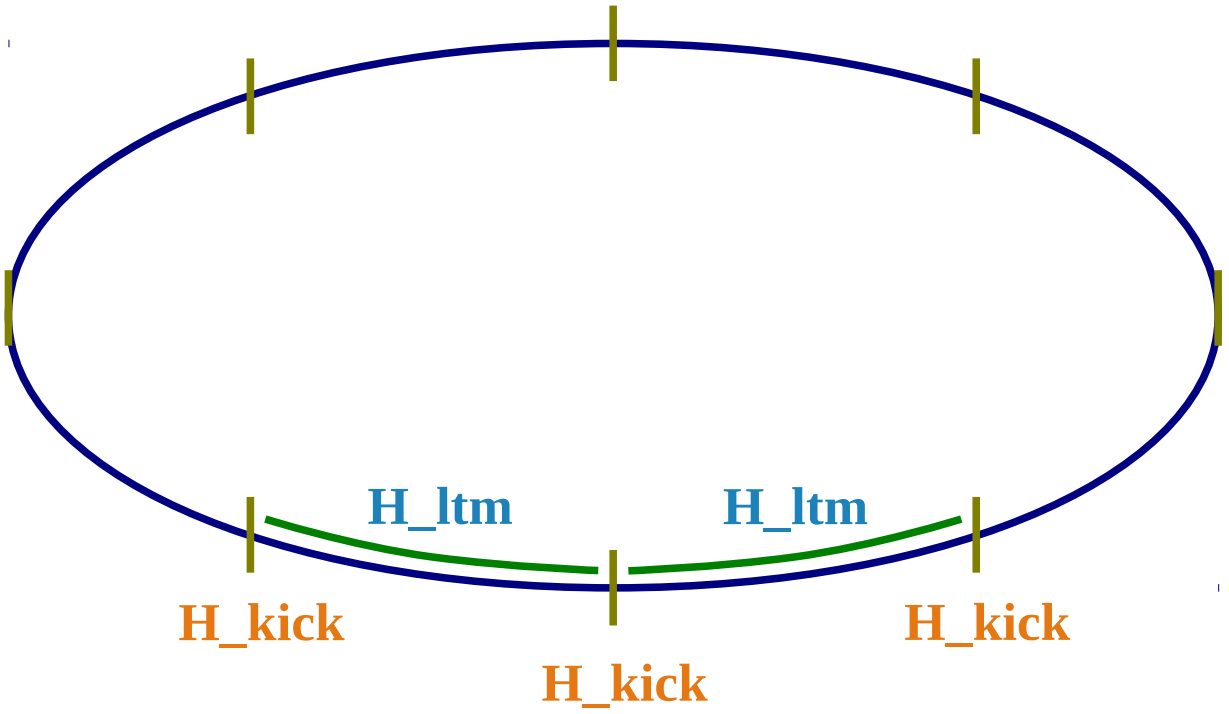
What is PyHEADTAIL?

- PyHEADTAIL is a macroparticle tracking code designed specifically to simulate collective effects in circular accelerators

How does PyHEADTAIL work?

- PyHEADTAIL is a macroparticle tracking code designed specifically to simulate collective effects in circular accelerators

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



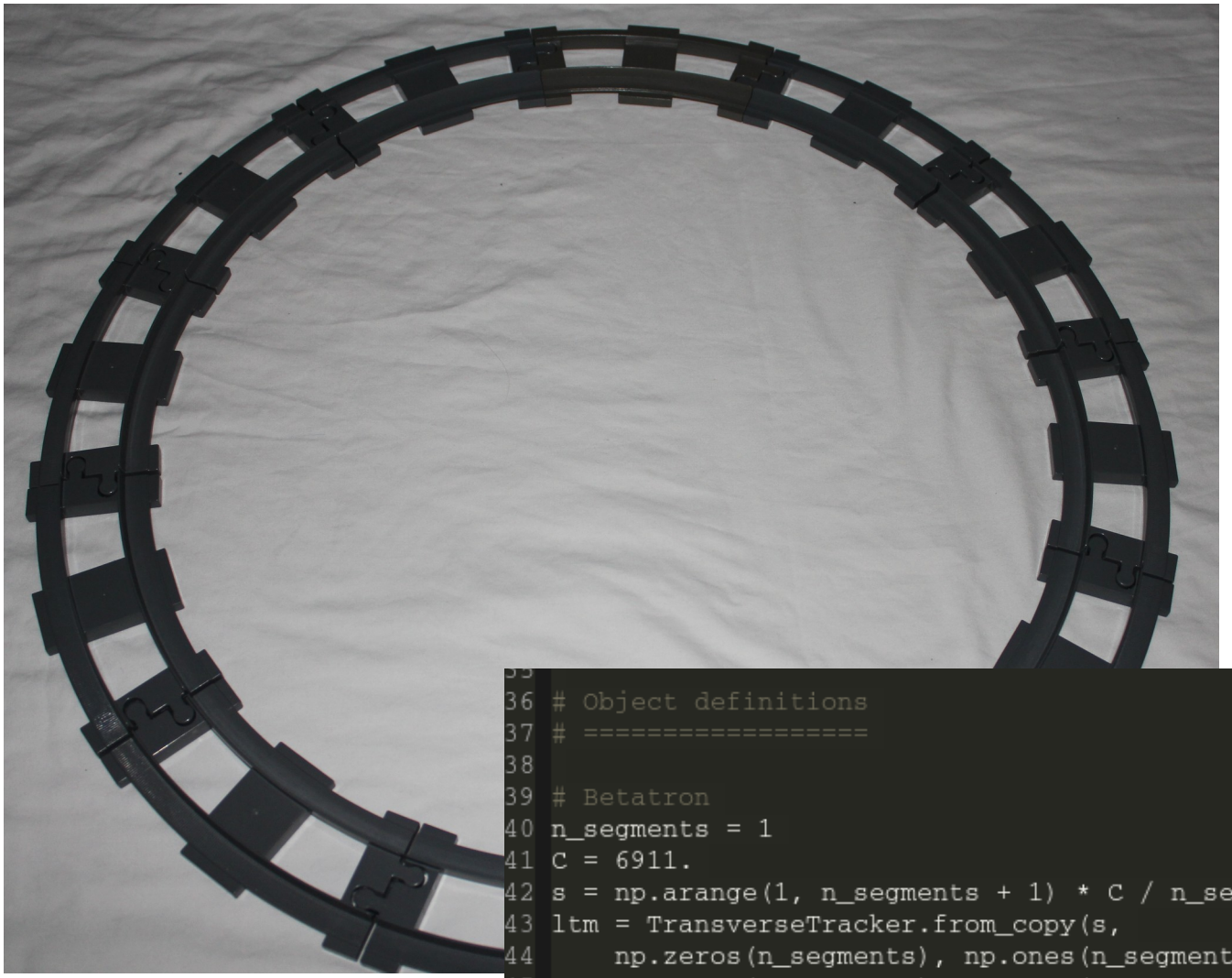
- H_ltm: linear transfer map
  - Chromaticity
  - Amplitude detuning
  - ...
- H_kick: collective interaction
  - Wakefields
  - Electron cloud
  - Feedback
  - Space-charge
  - ...

A real world example

- Load Python packages and modules

```
from __future__ import division
import cProfile, itertools, sys, time, timeit

from scipy.constants import c, e, m_p

from cobra_functions import stats, random
from beams.beams import *
from monitors.monitors import *
from spacecharge.spacecharge import *
from trackers.transverse_tracker import *
from trackers.longitudinal_tracker import *

```

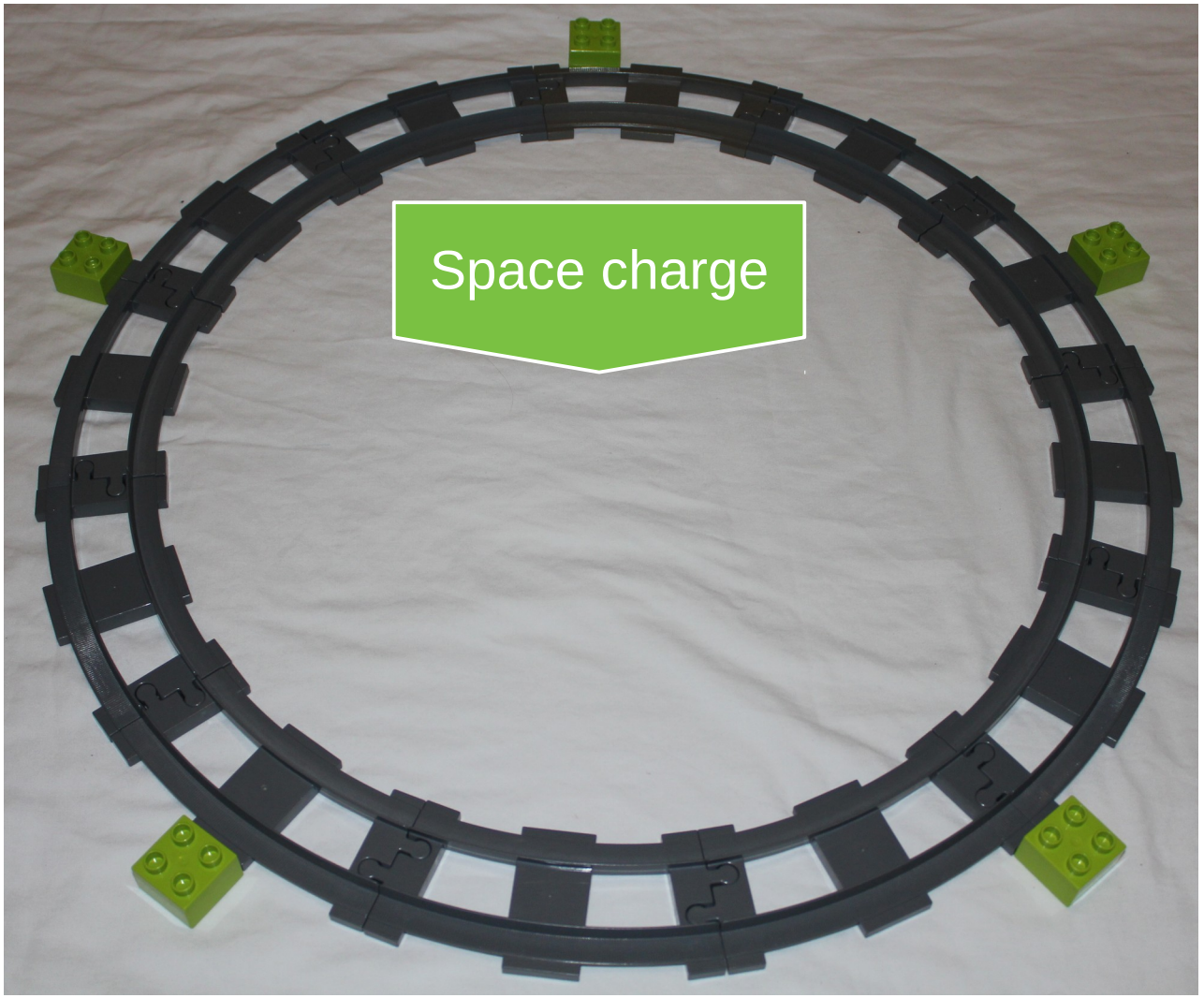
A real world example



- Load Python packages and modules
- Build linear periodic transfer maps

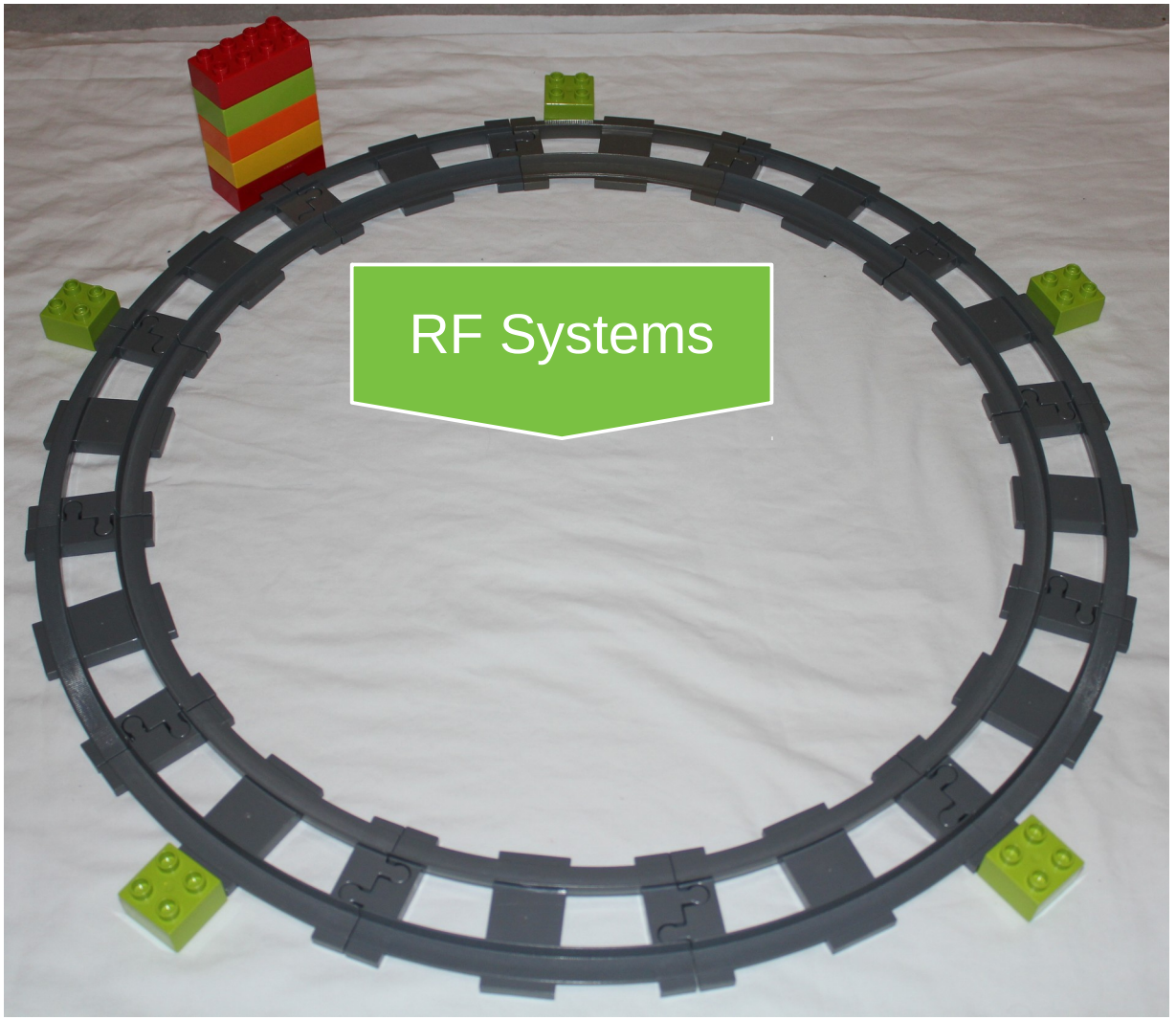
```
# Object definitions
# ==================

# Betatron
n_segments = 1
C = 6911.
s = np.arange(1, n_segments + 1) * C / n_segments
ltm = TransverseTracker.from_copy(s,
    np.zeros(n_segments), np.ones(n_segments) * beta_x, np.zeros(n_segments),
    np.zeros(n_segments), np.ones(n_segments) * beta_y, np.zeros(n_segments),
    Qx, 0, 0, Qy, 0, 0)

```

A real world example



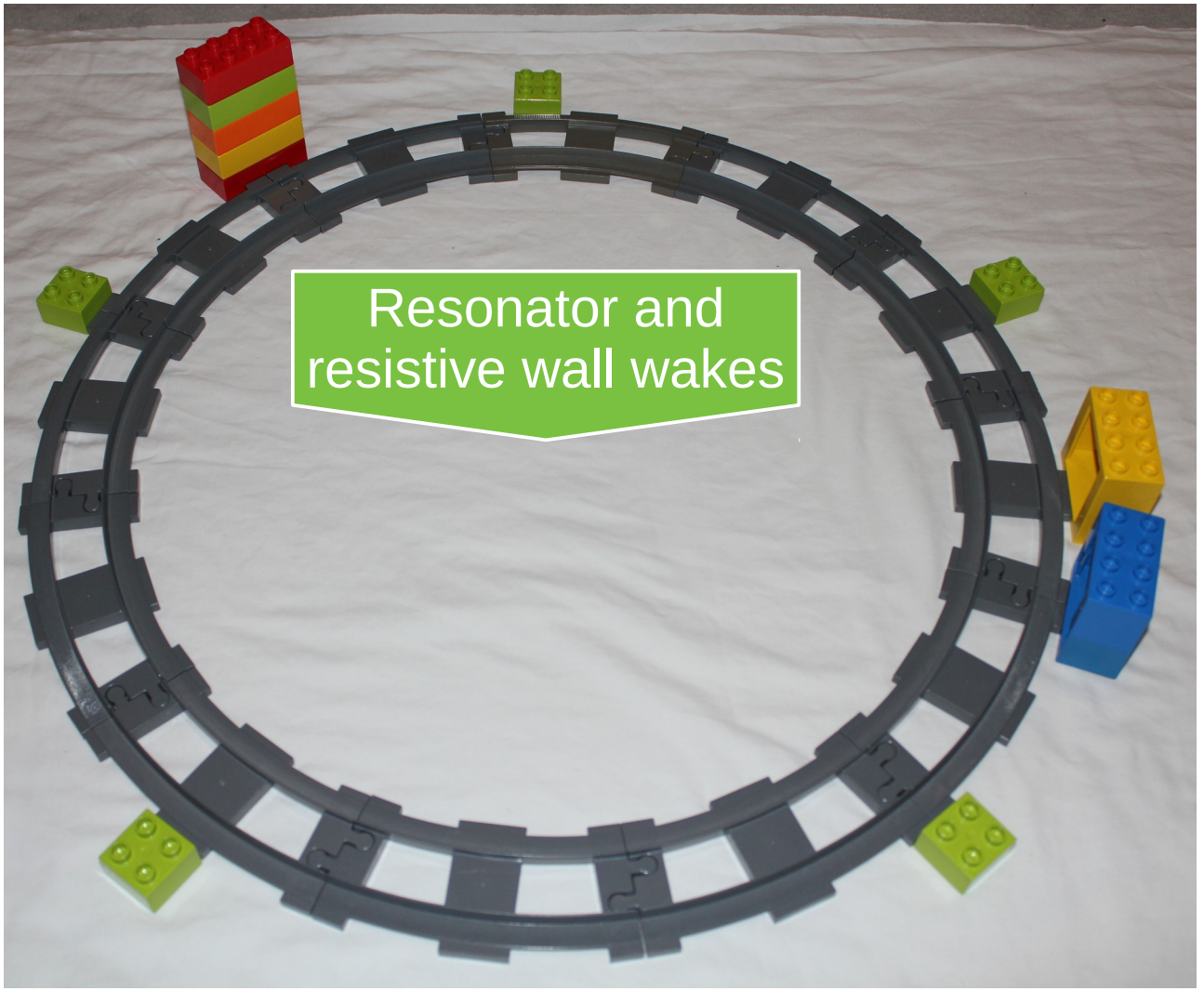
- Load Python packages and modules
- Build linear periodic transfer maps
- Add (collective) kick elements

A real world example



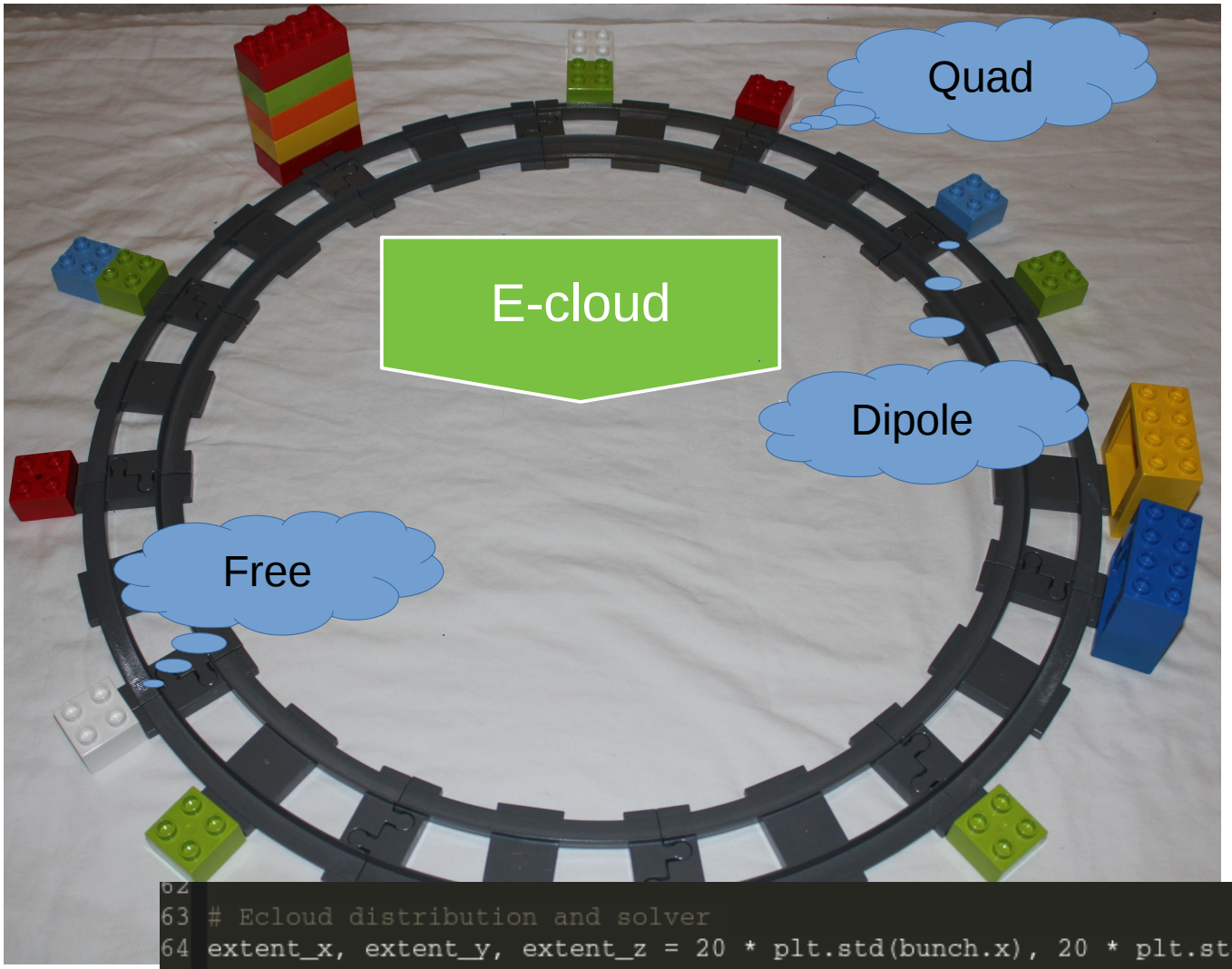
- Load Python packages and modules
- Build linear periodic transfer maps
- Add (collective) kick elements

A real world example



- Load Python packages and modules
- Build linear periodic transfer maps
- Add (collective) kick elements

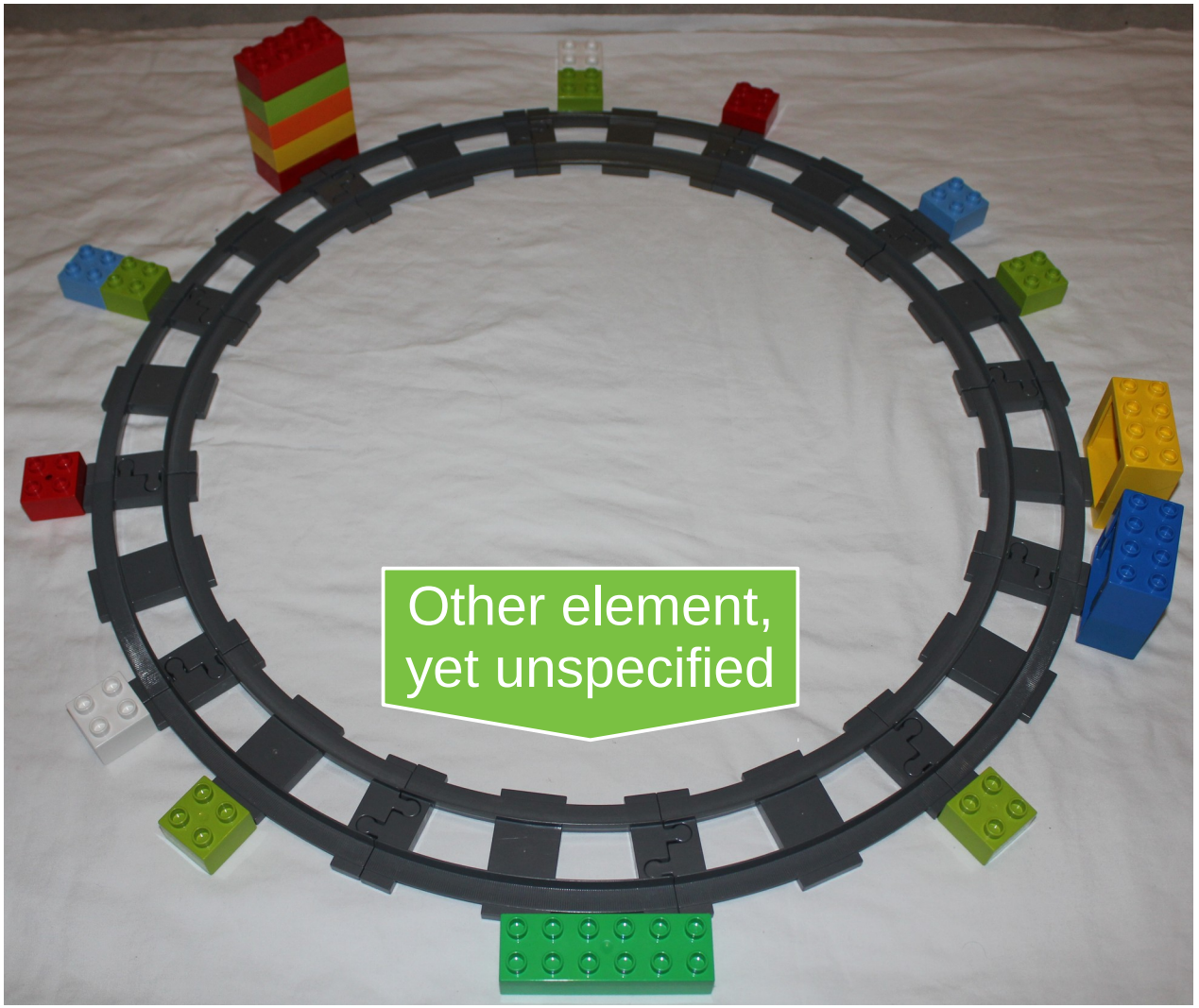
A real world example



- Load Python packages and modules
- Build linear periodic transfer maps
- Add (collective) kick elements

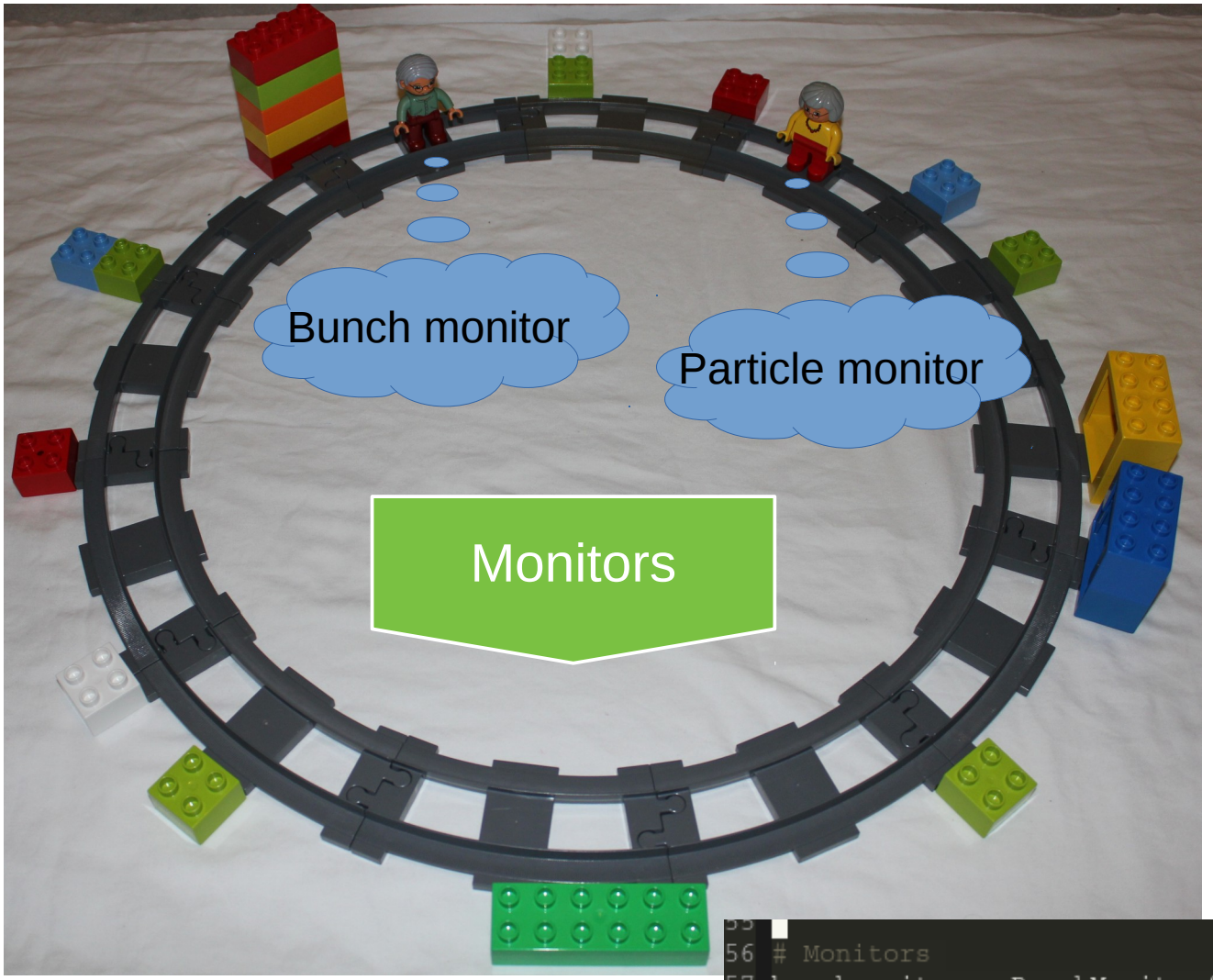
```
# Ecloud distribution and solver
extent_x, extent_y, extent_z = 20 * plt.std(bunch.x), 20 * plt.std(bunch.y), C / n_segments
cloud = Cloud(100000, 1e11, extent_x, extent_y, extent_z)
ecloud = SpaceCharge(cloud, 'cloud', extent_x, extent_y, 128, 128, slices)
```

A real world example



- Load Python packages and modules
- Build linear periodic transfer maps
- Add (collective) kick elements

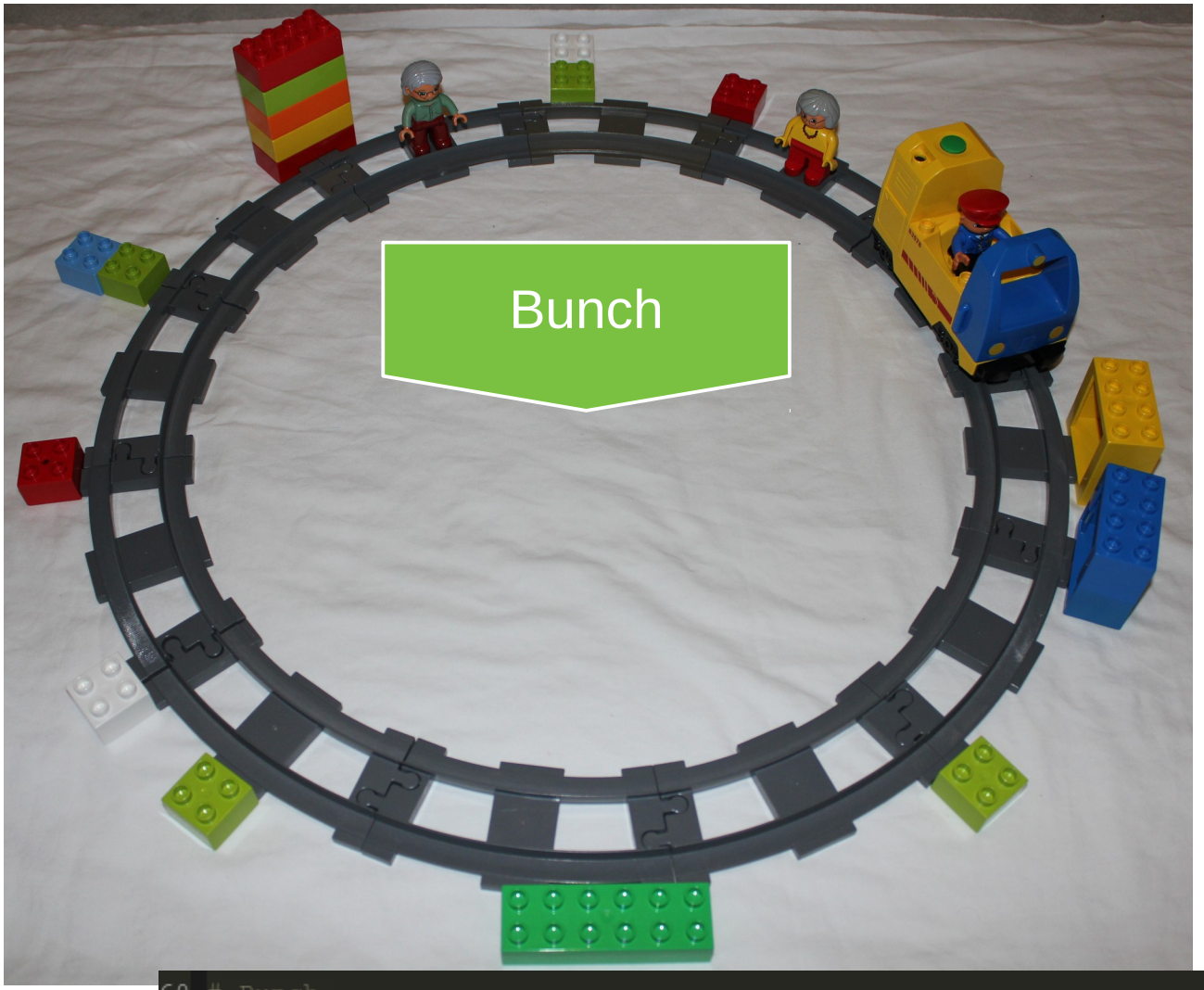
A real world example



- Load Python packages and modules
- Build linear periodic transfer maps
- Add (collective) kick elements

```
56 # Monitors
57 bunchmonitor = BunchMonitor('bunch', n_turns, slices)
58 particlemonitor = ParticleMonitor('particles', n_turns, slices)
```

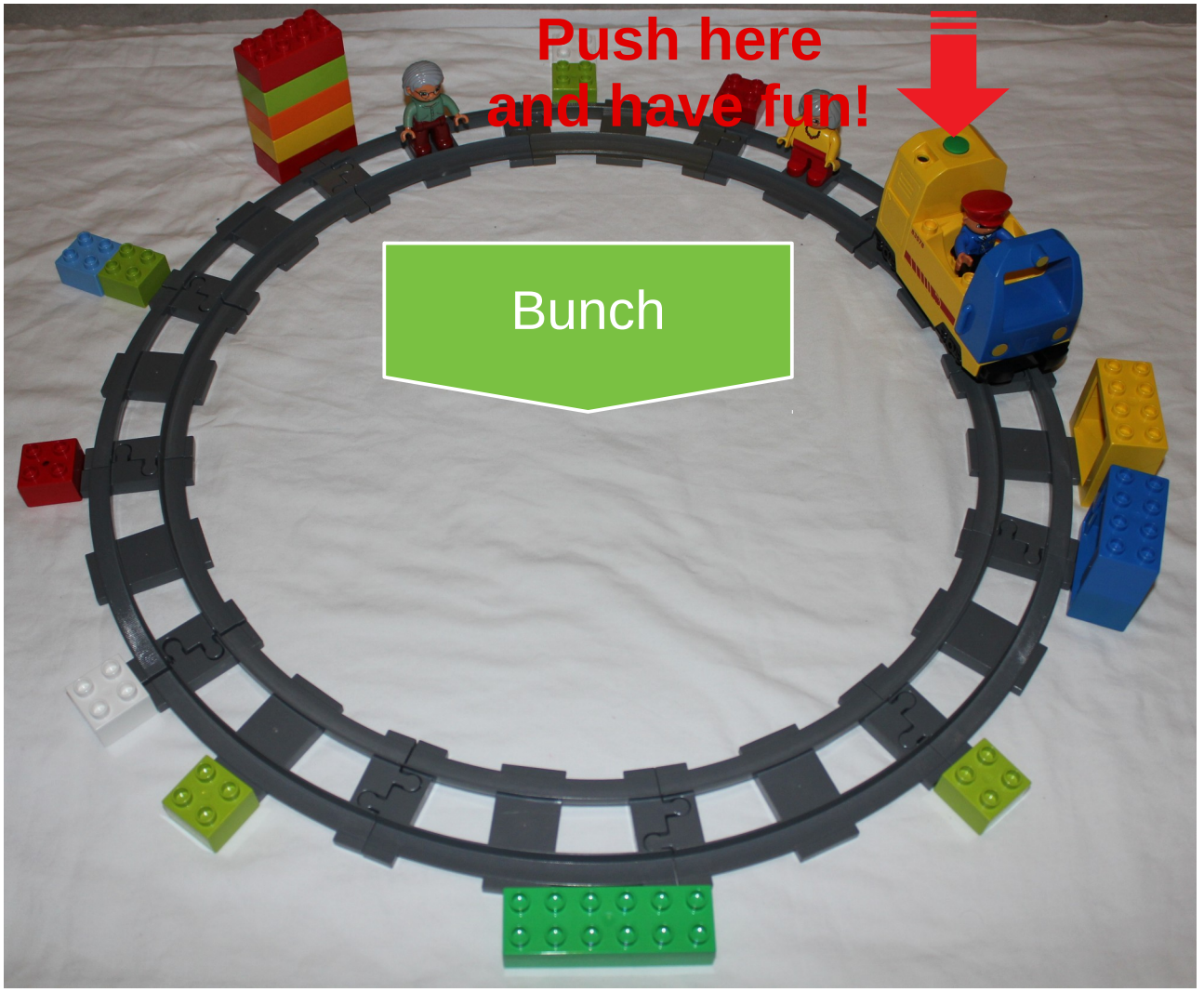
A real world example



- Load Python packages and modules
- Build linear periodic transfer maps
- Add (collective) kick elements
- Place beam

```
60 # Bunch
61 bunch = Bunch(500000, e, gamma, 1.15e11, m_p, 0, beta_x, epsn_x, 0, beta_y, epsn_y, beta_z, sigma_z)
```

A real world example



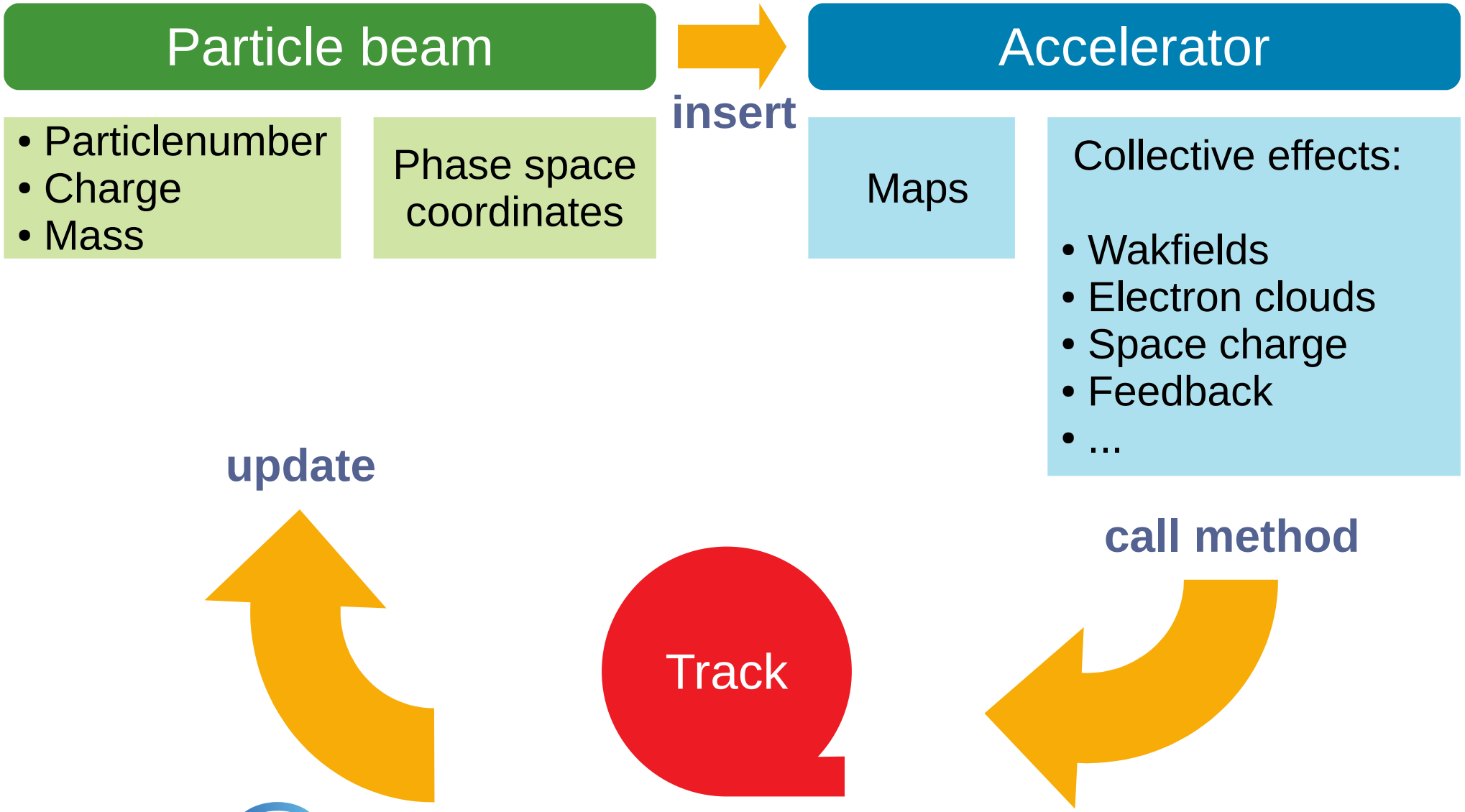
- Load Python packages and modules
- Build linear periodic transfer maps
- Add (collective) kick elements
- Place beam

```
86 for i in range(n_turns):
87     for m in map_:
88         m.track(bunch)
89
```

Model of beam and accelerator

- We have a beam in some initial state
- We want to transform this beam to some final state
- What is a beam?
  - A collection of particles
  - A set of generalized coordinates
  - A set of canonically conjugate momenta
- What is an accelerator?
  - A collection of elements that modifies a subset of beam attributes
- We continuously (periodically) pass the beam through the accelerator, changing the beam attributes – our job is design the accelerator in order to balance the change of attributes to our favour

PyHEADTAIL workflow



PyHEADTAIL workflow

Particle beam

- Particlenumber
- Charge
- Mass

Phase space coordinates

insert

Accelerator

Maps

Collective effects:

- Wakfields
- Electron clouds
- Space charge
- Feedback
- ...

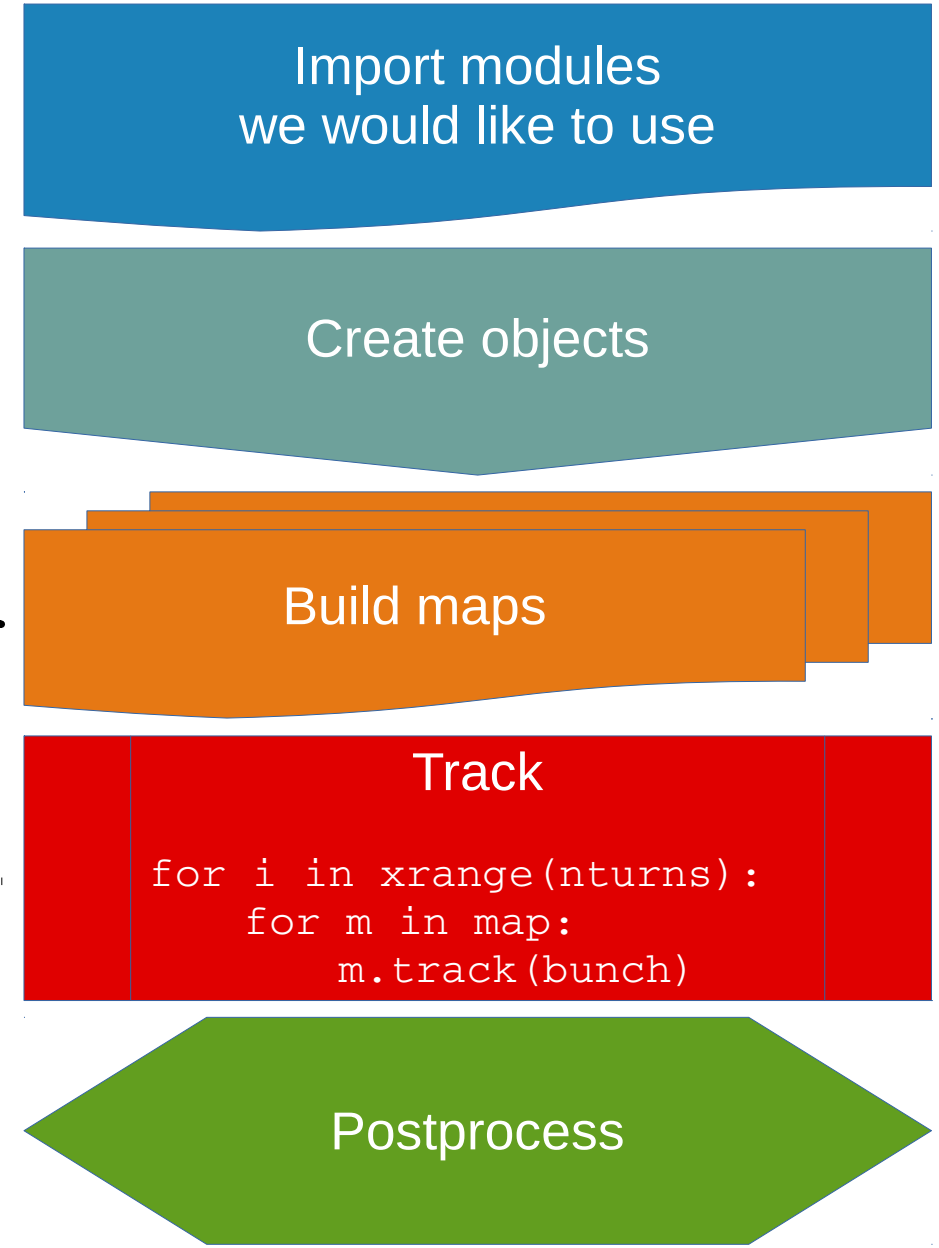
PyHEADTAIL modules

- **Independent**
- **Maintainable**
- **Extendible**

call method

PyHEADTAIL 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



Summary

- What is PyHEADTAIL and how does it work?
- We have shown a real world example
- We have learned about the PyHEADTAIL workflow

THE END