LONGITUDINAL TRACKING HANDS-ON

PART 1: INTRODUCTION TO TRACKING

RF CAS 2023

S. Albright, A. Lasheen,
F. Batsch, K. Iliakis, G. Papotti, D. Quartullo, H. Timko,
L. Intelisano, B. Karlsen-Baeck, I. Karpov, O. Naumenko, A. Vanel, M. Zampetakis

1983
—2023
years

The CERN Accelerator School

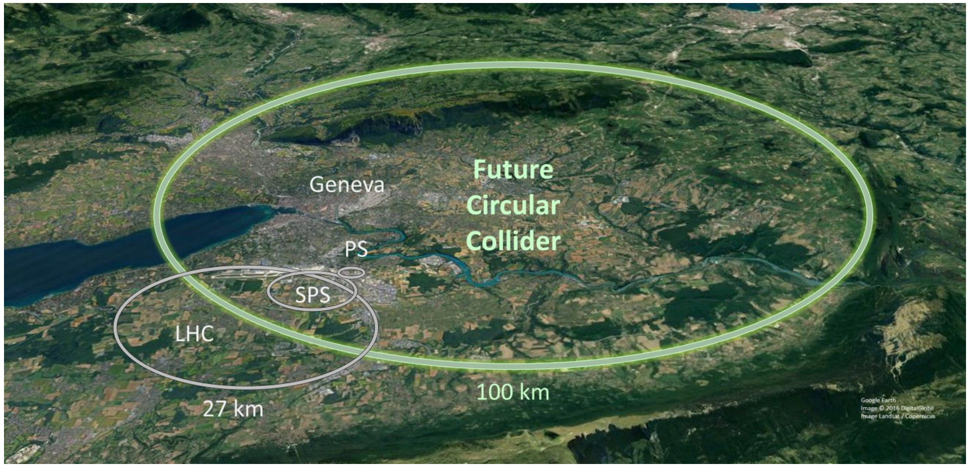
COPYRIGHT

Copyright statement and speaker's release for video publishing

SCENARIO

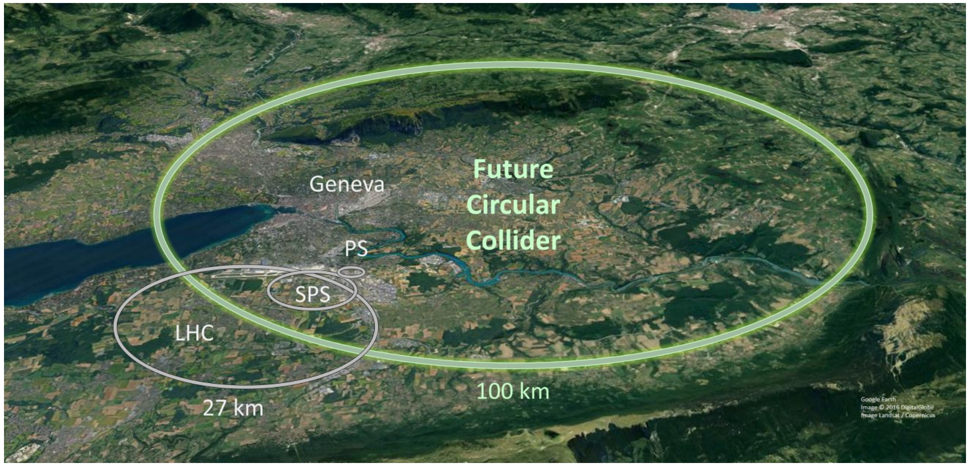
You've been enrolled to work for the FCC-hh project, and more specifically on its injector complex...

We need to decide whether we use the LHC as a High Energy Booster, or upgrade the SPS into a superconducting machine at 1.3 TeV: the scSPS...



FCC-hh Conceptual Design Report, Section 6, p.940

SCENARIO



You are part of the team working on the scSPS. Your colleague from beam transfer says there are many benefits in terms of transfer line design...

One question remains: what about longitudinal beam stability due to wakefields?...

The task is complex, so are the RF system and the impedance model, the analytical approach difficult... How can we evaluate beam stability?

ONE TOOL YOU SHOULD TRY: BEAM TRACKING!

OUTLINE OF THE HANDS-ON

FOR EACH HANDS-ON BLOCK

FIRST AFTERNOON: INTRODUCTION TO TRACKING

- Develop a multiparticle tracking code, without intensity effects
- Two options are offered:
    - Make your own tracker!
    - Use the BLonD simulation code

SECOND AFTERNOON: ADVANCED TRACKING

- Learn how to include intensity effects (wakefields) using BLonD
- Check if instabilities are a limitation for scSPS
- Get insights on advanced topics like potential well distortion, as you would measure it in a real accelerator

OUTLINE OF THE HANDS-ON

FOR EACH HANDS-ON BLOCK

FIRST AFTERNOON: INTRODUCTION TO TRACKING

- Develop a multiparticle tracking code, without intensity effects
- Two options are offered:
  - Make your own tracker!
  - Use the BLonD simulation code

TODAY!

INTRODUCTION TO TRACKING

ALL YOU NEED TO GET STARTED

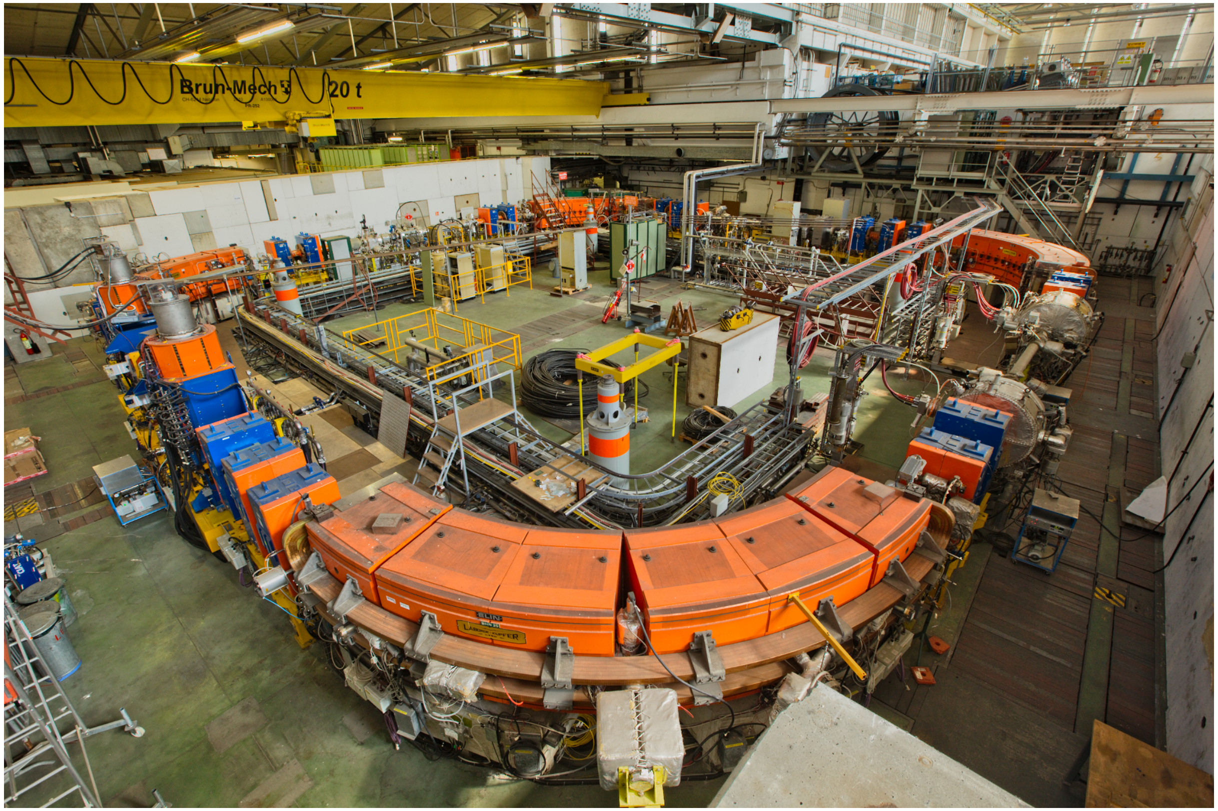
→ **Introduction to particle acceleration in a synchrotron**

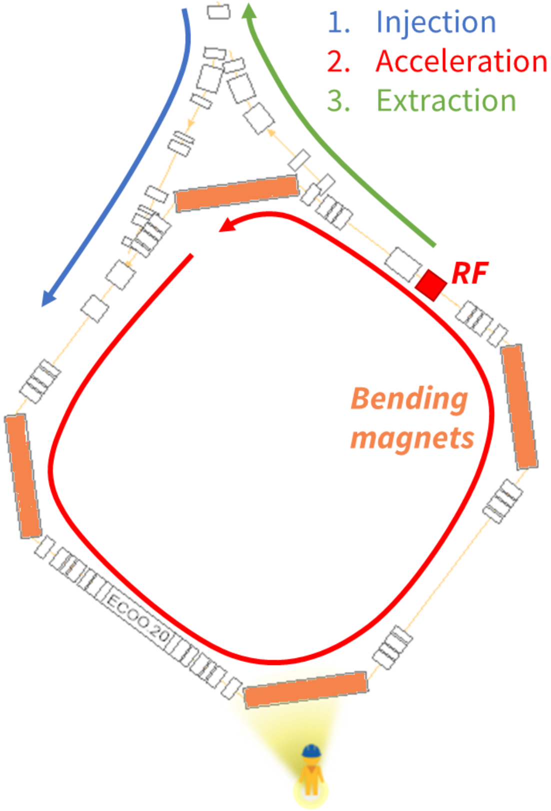
→ **Recap on longitudinal equations of motion**

→ **An example tracking code**

LAYOUT OF A REAL ACCELERATOR

THE LOW ENERGY ION RING (LEIR) AT CERN

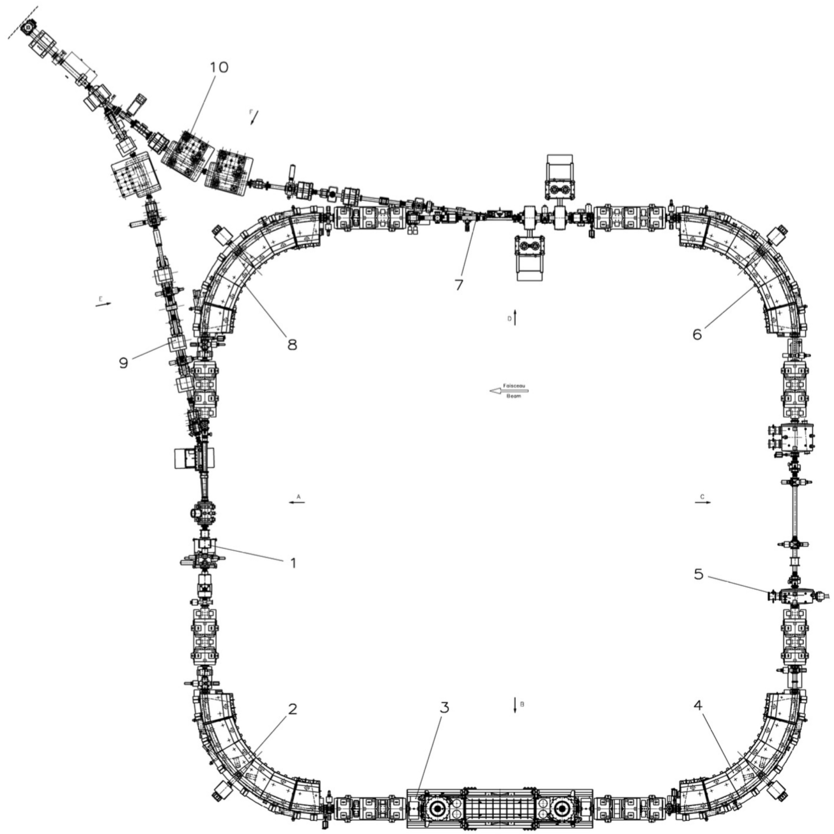


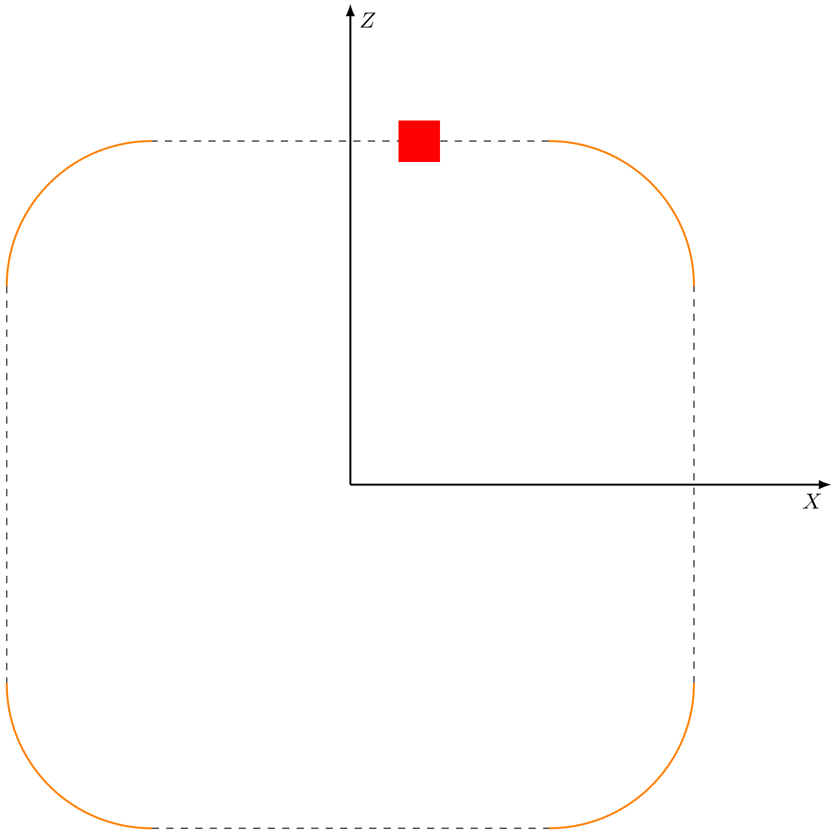


- Click to see other accelerators at CERN…

LEIR SEEN FROM THE ABOVE

And simplifying the layout...

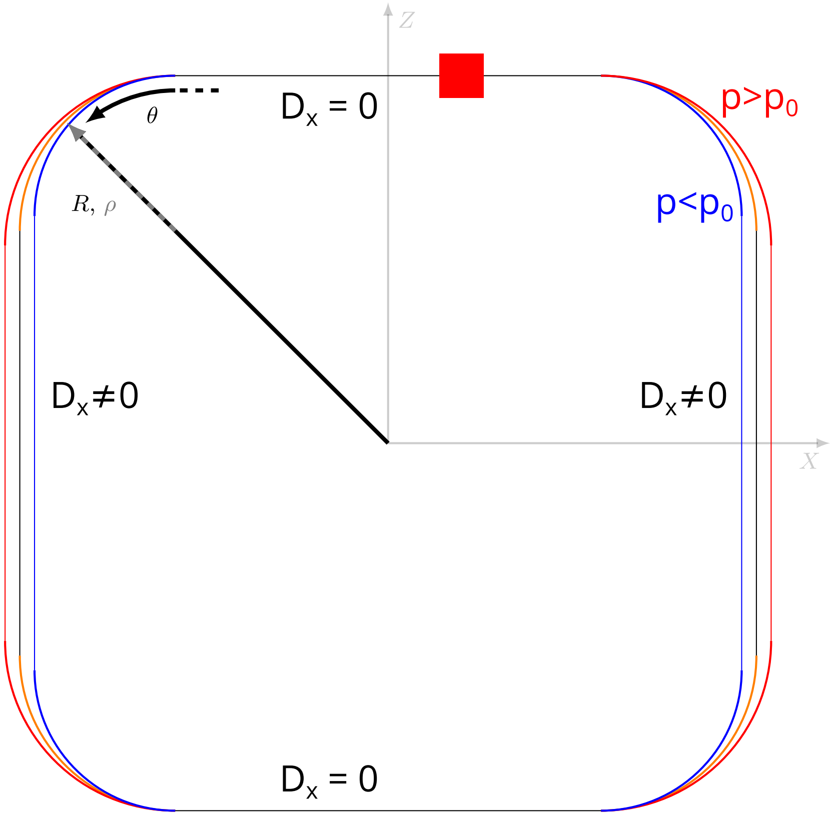


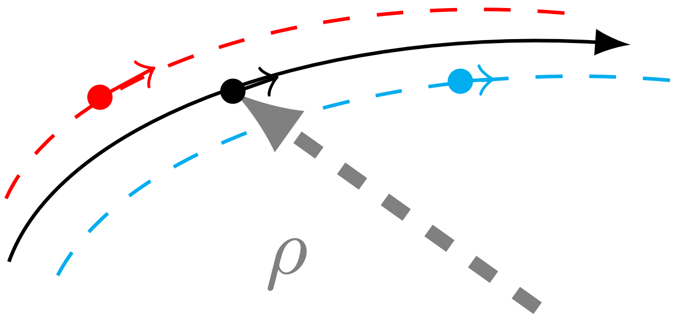


Bending magnets Accelerating RF cavities

PARTICLE TRAJECTORIES

EVOLUTION OF THE RELATIVE PARTICLE POSITIONS

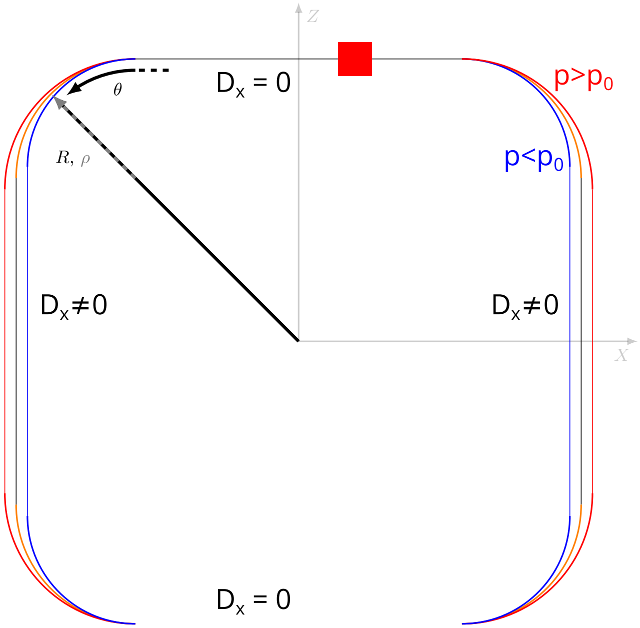




- Longitudinal dynamics is about **acceleration of particles**, and **relative motion in the longitudinal plane**.

- In the longitudinal direction, a particle can be in front (in advance), or behind (late) with respect to the **ideal particle (on time)**.

MOMENTUM COMPACTION AND PHASE SLIP FACTOR



Is the blue particle (slower, but shorter path) arriving before or after the red one (faster, but longer path) after a turn?

- Particles with larger momentum p will have a different orbit R, defined by the **momentum compaction factor**

$$\alpha_p = \frac{dR/R}{dp/p}$$

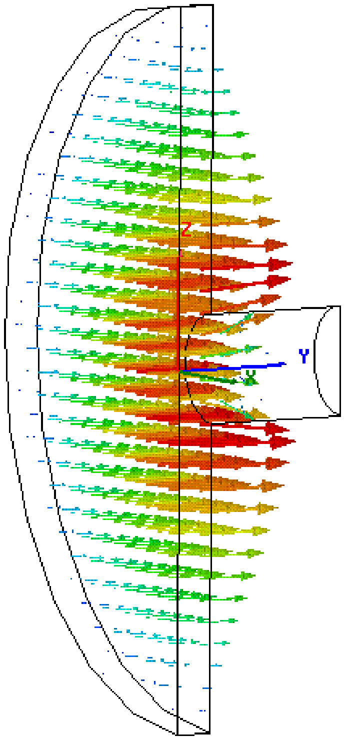
- This parameter is linked to the relative change in angular revolution frequency ω_0, **the phase slip factor**

$$\eta = -\frac{d\omega_0/\omega_0}{dp/p} = \alpha_p - \frac{1}{\gamma^2}$$

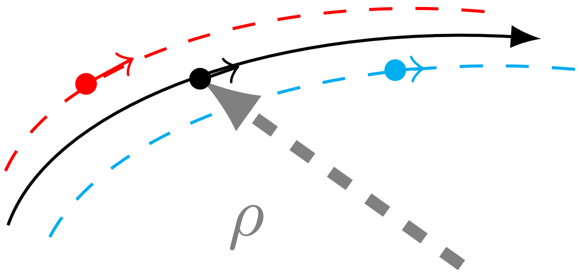
where γ is the relativistic Lorentz factor

PARTICLE TRAJECTORIES

ACCELERATION IN THE RF SYSTEM



Animation:
E. Jensen



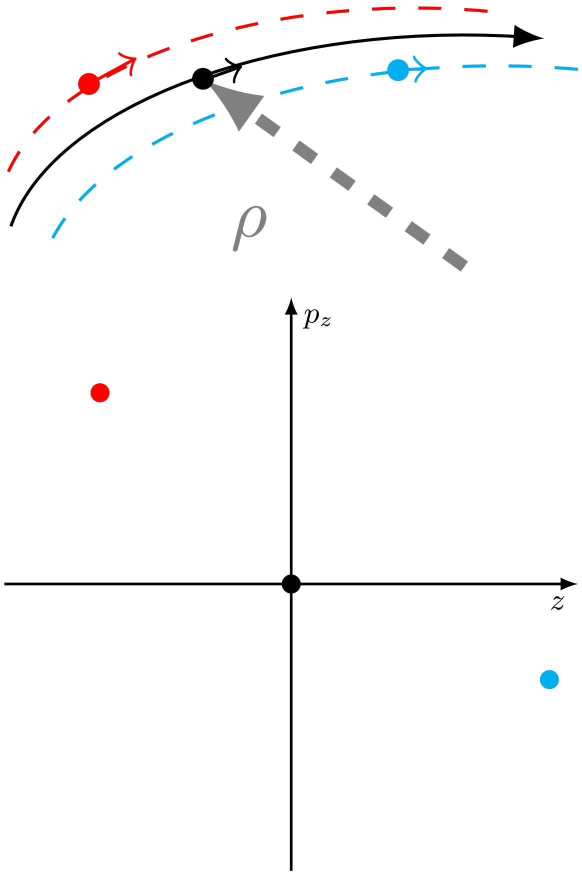
- Each particle arrives at a different time τ in the RF system, where it will gain (or lose) a different amount of energy.

- The energy change at each passage is

$$\delta E_{\mathrm{rf}}\left(\tau\right) = qV_{\mathrm{rf}}\cos\left(\omega_{\mathrm{rf}}\tau\right)$$

where q is the charge, V_{rf} the RF potential, ω_{rf} the angular resonant frequency of the cavity.

LONGITUDINAL PHASE SPACE



- Particles and their motion can be represented in **longitudinal phase space.** (z, p_z)

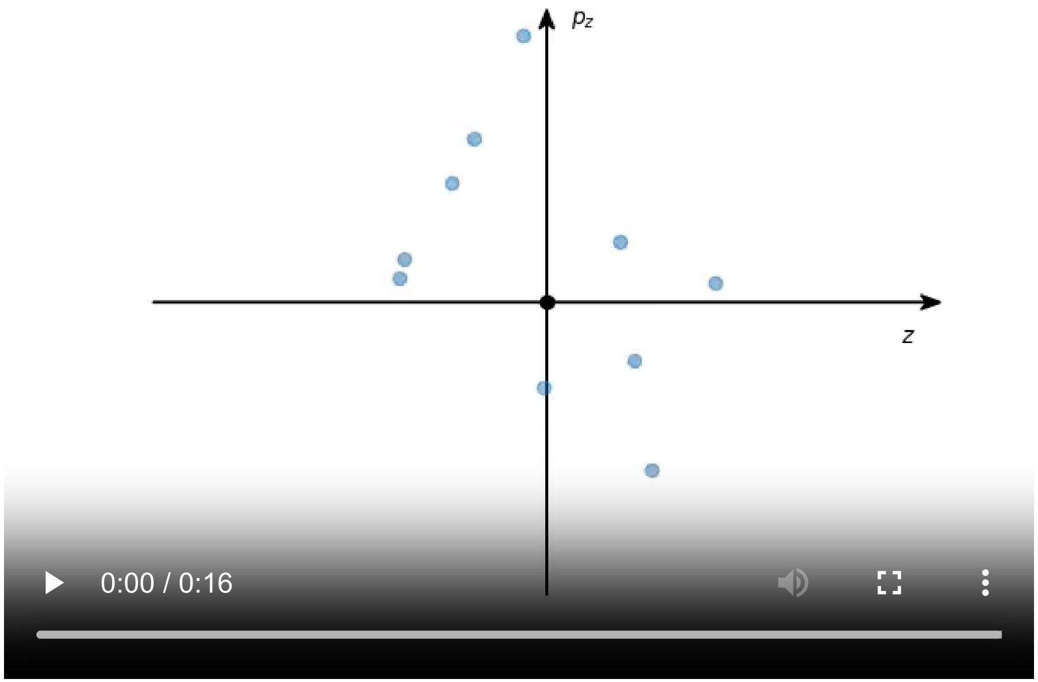
- z is the longitudinal position linked to τ, the relative time of arrival between two particles, $z = -\beta c \tau$

- As described before other particles can be
  - In front, or in advance in time (right)
  - In the back, or delayed in time (left)
  - Have higher momentum/velocity (top)
  - Have lower momentum/velocity (bottom)

- The motion of the particles in the longitudinal phase space is called **synchrotron motion**.

SYNCHROTRON OSCILLATIONS

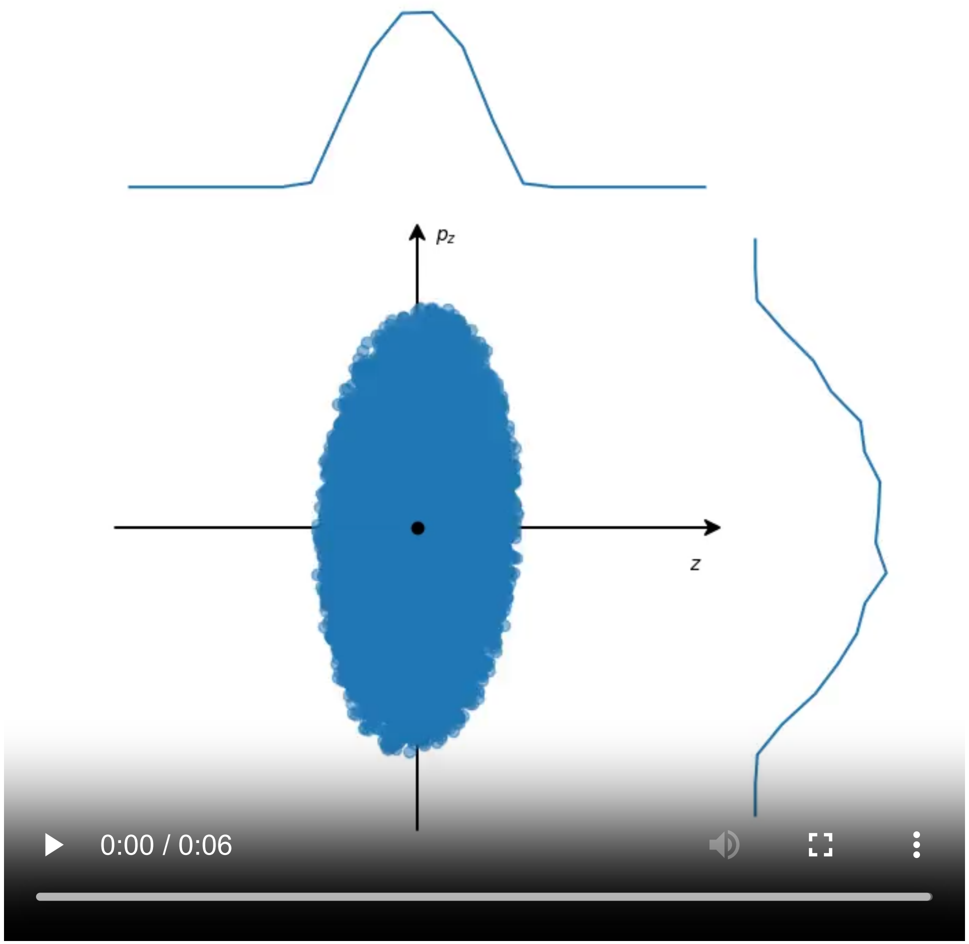
WITH A FEW PARTICLES



Stable conditions can be found where particles oscillate around the ideal particle in black. This motion is called **synchrotron oscillations**.

SYNCHROTRON OSCILLATIONS

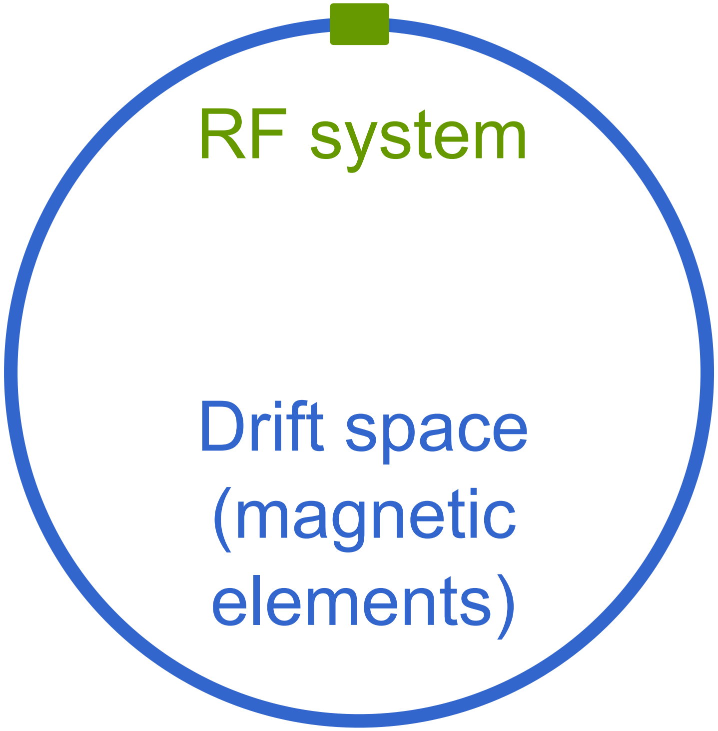
WITH MANY PARTICLES



- A bunch is usually composed of a very large number of particles, typically $\mathcal{O}\left(10^{10} - 10^{12}\right)$.
- In a real machine, the coherent motion of a bunch can be measured and analyzed from the longitudinal bunch density (top line, projection along the p_z axis, instantaneous beam current).
- **Let's simulate synchrotron motion with tracking!**

SYNCHROTRON DEFINITION

Simplest configuration, starting with a single RF system and assuming no longitudinal energy gain/loss in the rest of the ring



- Definition of "design" energy (momentum)

$$p_d = q\mathcal{B}_y \rho_d$$

- Definition of "design" revolution period

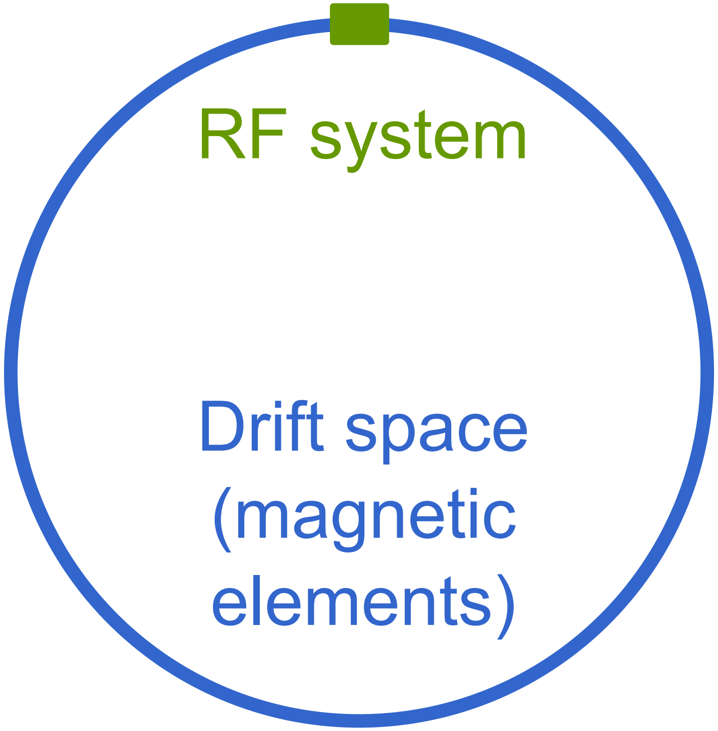
$$\mathcal{B}_y, p_d \rightarrow T_{0,d} = \frac{2\pi}{\omega_{0,d}} = \frac{C_d}{\beta_d c}$$

- Definition of RF parameters

$$\omega_{\mathrm{rf}} = h\omega_{0,d}$$

SYNCHROTRON DEFINITION

Simplest configuration, starting with a single RF system and assuming no longitudinal energy gain/loss in the rest of the ring



- In the context of the tracking code, we will use the $(\Delta t, \Delta E)$ coordinate system
- Relative energy of an arbitrary particle

$$\Delta E = E - E_d$$

- Time of arrival in the RF system, relative to $T_{0,d}$ at each turn n

$$\Delta t = \tau - \sum_n T_{0,d}$$

ENERGY GAIN IN RF CAVITY

- The energy gain in the cavity gap is essentially discrete, the relative energy gain every turn $T_{0,d}$ is

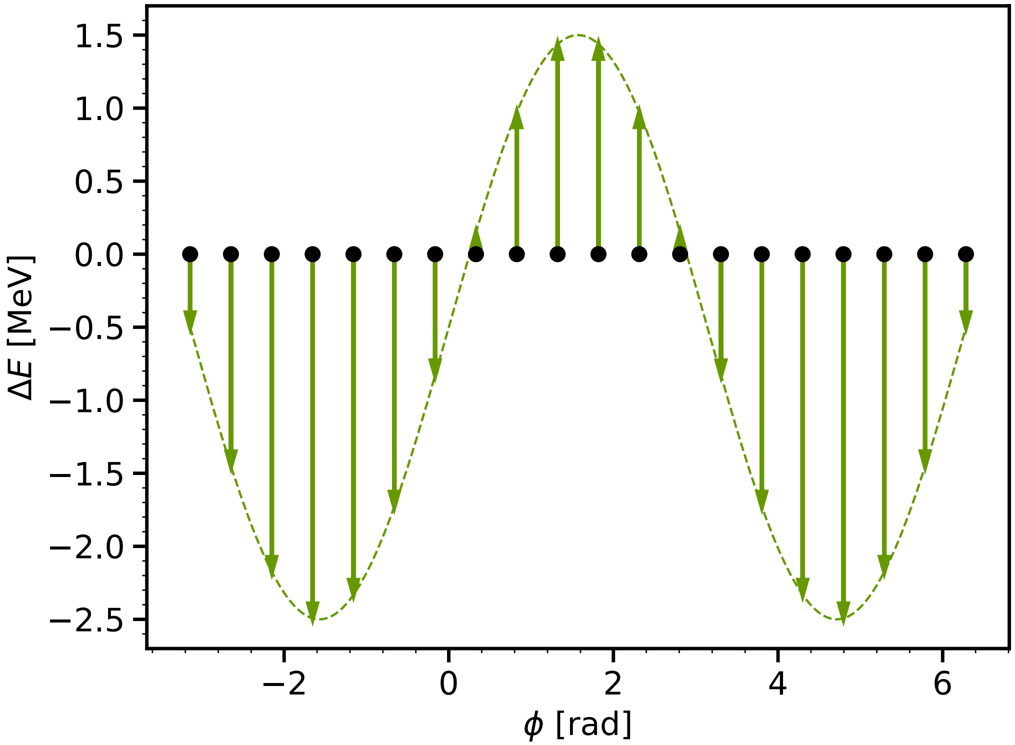
$$\Delta E^{n+1} = \Delta E^{n} + qV_{\mathrm{rf}} \sin\left(\omega_{\mathrm{rf}} \Delta t + \phi\right) - \delta E_d^{n\rightarrow n+1}$$

- The acceleration per turn is

$$\delta E_d^{n\rightarrow n+1} = 2\pi q \rho_d R_d \frac{\mathcal{B}_y^{n+1} - \mathcal{B}_y^{n}}{T_{0,d}}$$

where $\mathcal{B}_y^{n+1} - \mathcal{B}_y^{n}$ is the increment in the bending magnet field at each turn with a design bending radius ρ_d.

ENERGY GAIN IN RF CAVITY



- Example for protons passing in RF system with $V = 2\mathrm{MV}$, $f_{\mathrm{rf}} = 200\mathrm{MHz}$, $\delta E_d^{n\rightarrow n+1} = 0.5\mathrm{MeV}$ (note the vertical offset of the RF waveform with respect to 0).

DRIFT

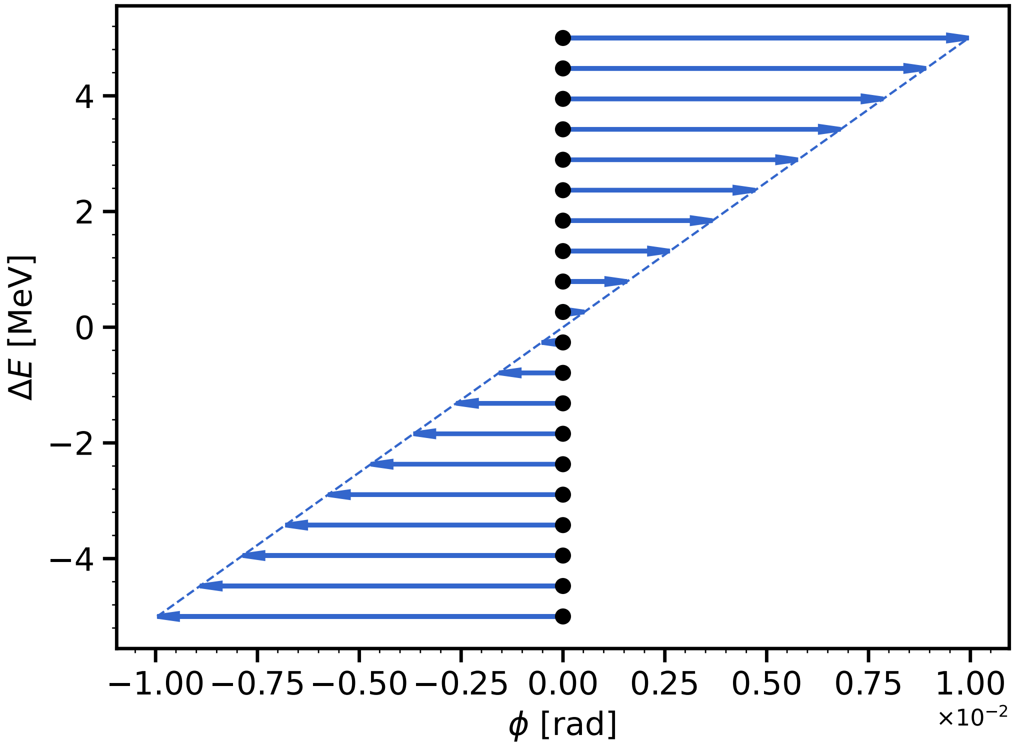
- The relative slippage in time of the particles is commonly called the "drift" equation, the drift after $T_{0,d}$ is

$$\Delta t^{n+1} = \Delta t^{n} + \frac{\eta_0 T_{0,\mathrm{d}}}{\beta_d^2 E_d} \Delta E^{n}$$

with

$$\eta_{0,d} = \frac{1}{\gamma_{\mathrm{tr}}^2} - \frac{1}{\gamma_d^2} = \alpha_c - \frac{1}{\gamma_d^2}$$

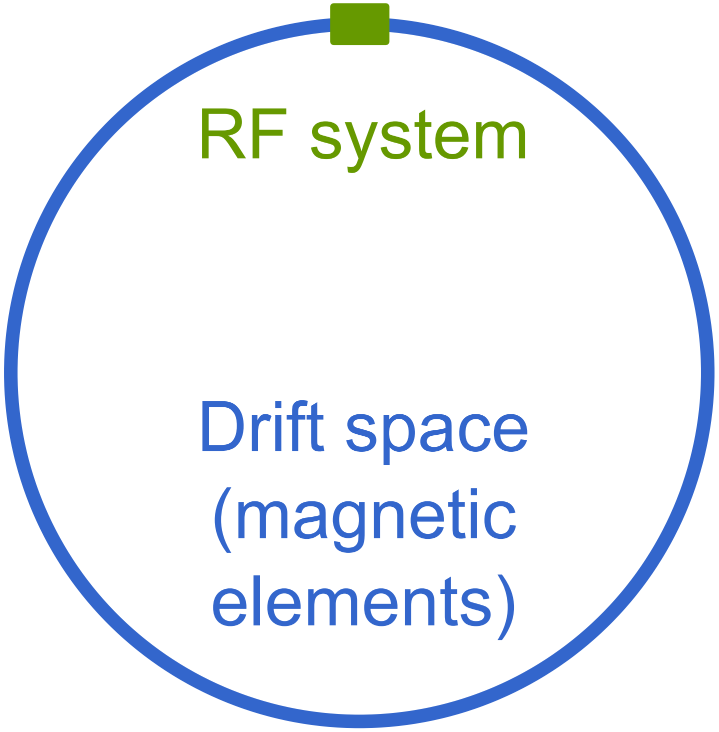
DRIFT



- Example for protons passing along a ring with $C_d = 6911.50\mathrm{m}$, $E_d = 26\mathrm{GeV}$, $\gamma_d = 27.7$, $\gamma_{\mathrm{tr}} = 18$

TRACKING

The two equations of motion are sufficient to build a simple **(yet very useful)** tracking code



- The tracking can be coded as

```
for n_turns:
    dE += rf_kick(phi)
    phi += drift(dE)
```

- Where

```
def rf_kick(dt):
    return q*Vrf*sin(omega_rf*dt+phase) \
           - E_gain
```

```
def drift(dE):
   return (eta*t_rev)/(beta**2*energy)*dE
```

TRACKING

A REALISTIC WORKING CODE IN PYTHON

The present example is using Python, but any language could be used following the code layout from the previous slide.

- We import useful libraries

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.constants import m_p, c, e
```

- Numpy is a numerical computation package, perfect to handle arrays of particles
- `Matplotlib` is a versatile package for plotting

TRACKING

A REALISTIC WORKING CODE IN PYTHON

- Define our machine parameters

```
Ekin = 26e9  # eV
q = 1
E0 = m_p * c**2. / e
circumference = 6911.5  # m
energy = Ekin + E0
momentum = np.sqrt(energy**2.
    - E0**2.)
beta = momentum / energy
gamma = energy / E0

t_rev = circumference/(beta*c)
f_rev = 1 / t_rev

harmonic = 4620
Vrf = 4.5e6  # V
f_rf = harmonic * f_rev
omega_rf=2*np.pi*f_rf
t_rf = 1 / f_rf

gamma_t = 18
alpha_c = 1 / gamma_t**2.
eta = alpha_c - 1 / gamma**2.
```

- Print the parameters of the machine

```
print("Beta: " +
      str(beta))
print("Gamma: " +
      str(gamma))
print("Revolution period: " +
      str(t_rev * 1e6) + " mus")
print("RF frequency: " +
      str(f_rf / 1e6) + " MHz")
print("RF period: " +
      str(t_rf * 1e9) + " ns")
print("Momentum compaction \
       factor: " +
      str(alpha_c))
print("Phase slippage \
       factor: " +
      str(eta))
```

TRACKING

A REALISTIC WORKING CODE IN PYTHON (AND NUMPY)

- Define our machine parameters

```
Ekin = 26e9  # eV
q = 1
E0 = m_p * c**2. / e
circumference = 6911.5  # m
energy = Ekin + E0
momentum = np.sqrt(energy**2.
    - E0**2.)
beta = momentum / energy
gamma = energy / E0

t_rev = circumference/(beta*c)
f_rev = 1 / t_rev

harmonic = 4620
Vrf = 4.5e6  # V
f_rf = harmonic * f_rev
omega_rf=2*np.pi*f_rf
t_rf = 1 / f_rf

gamma_t = 18
alpha_c = 1 / gamma_t**2.
eta = alpha_c - 1 / gamma**2.
```

- Define your tracking functions

```
def drift(dE, t_rev, eta, beta,
          energy):

    return (eta*t_rev) / \
        (beta**2*energy)*dE

def rf_kick(dt, q, Vrf, omega_rf,
            phase=0, E_gain=0):

    return q*Vrf*np.sin(
        omega_rf*dt+phase)-E_gain
```

- Define your initial particle positions (test example)

```
n_parts = 10
dt_coordinates = np.linspace(
    0, t_rf, n_parts)
dE_coordinates = np.zeros(n_parts)
```

TRACKING

A REALISTIC WORKING CODE IN PYTHON (AND NUMPY)

- Track!!!

```
n_turns = 10

for idx_turn in range(n_turns):
    dE_coordinates += rf_kick(
        dt_coordinates, q, Vrf, omega_rf, phase=0, E_gain=0)

    dt_coordinates += drift(
        dE_coordinates, t_rev, eta, beta, energy)
```

TRACKING

A REALISTIC WORKING CODE IN PYTHON (AND NUMPY)

- The particle coordinates can then be monitored at each turn during the tracking.

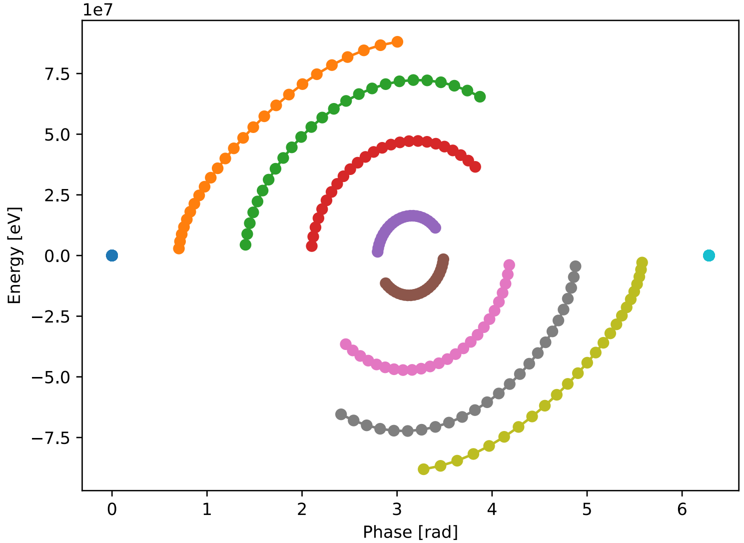
```python
n_turns = 25

saved_positions_dt = np.zeros((n_parts, n_turns))
saved_positions_dE = np.zeros((n_parts, n_turns))

for idx_turn in range(n_turns):
    dE_coordinates +=rf_kick(
        dt_coordinates, q, Vrf, omega_rf, phase=0, E_gain=0)

    dt_coordinates +=  drift(
        dE_coordinates, t_rev, eta, beta, energy)

    saved_positions_dE[:, idx_turn] = dE_coordinates
    saved_positions_dt[:, idx_turn] = dt_coordinates
```

TRACKING

A REALISTIC WORKING CODE IN PYTHON (AND NUMPY)

- The trajectory of the particles can be visualized

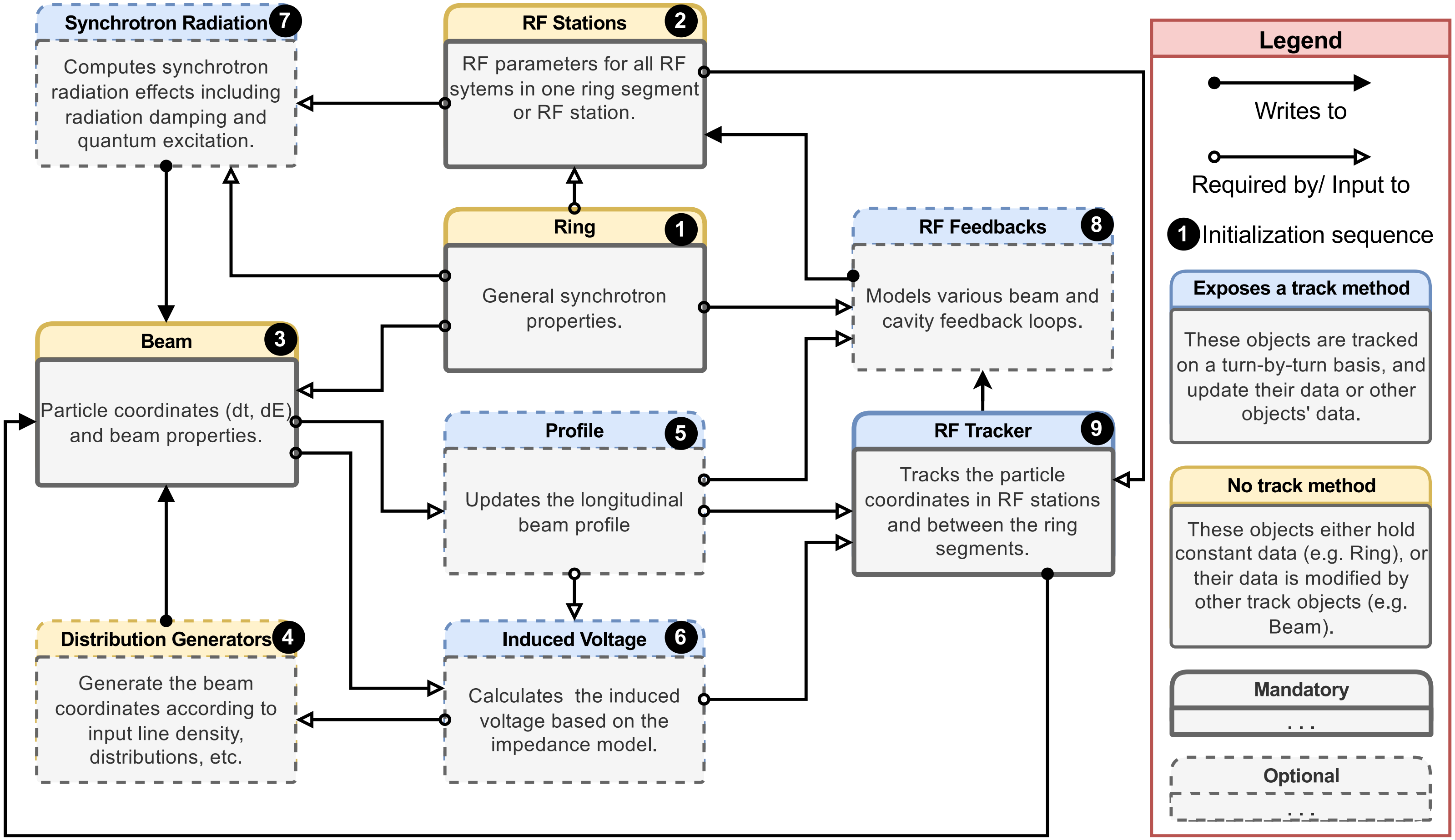
```
plt.figure('phase space')
plt.clf()
for idx_part in range(n_parts):
    plt.plot(
        saved_positions_dt[idx_part, :],
        saved_positions_dE[idx_part, :],
        '-o')
plt.xlabel('Phase [rad]')
plt.ylabel('Energy [eV]')
```



- The final script is less than 100 lines long!

THE BLOND SIMULATION CODE

OBJECT ORIENTED SIMULATOR WITH PYTHON INTERFACE



The BLonD code repository

THE BLOND SIMULATION CODE

EXAMPLE TRACKING USING BLOND (1/2)

- Defining the accelerator parameters

```
particle=Proton()
ring = Ring(circumference, alpha_c, momentum, particle, n_turns=n_turns)
rf_station = RFStation(ring, harmonic, Vrf, phase)
```

- Defining the initial bunch distribution (parabolic)

```
beam = Beam(ring, n_parts, intensity)
parabolic(ring, rf_station, beam, bunch_length)
```

- Defining the tracking objects, acting on the beam

```
rf_tracker = RingAndRFTracker(rf_station, beam)
full_tracker = FullRingAndRF([rf_tracker])
```

THE BLOND SIMULATION CODE

EXAMPLE TRACKING USING BLOND (2/2)

- Saving trajectories

```
saved_dt = np.zeros((n_turns, n_parts))
saved_dE = np.zeros((n_turns, n_parts))
```

- **TRACKING!!!**, and saving trajectories

```
for turn in range(ring.n_turns):

    '''Our two equations of motion
     are applied here!!'''
    full_tracker.track()

    saved_dt[turn] = beam.dt
    saved_dE[turn] = beam.dE
```

- At each track() method, the equations of motion are applied once for the turn. The for loop applies the equations for the whole simulation.
- A Profile object can be made to monitor the evolution of the longitudinal profile.

WHERE TO FIND THE EXERCISES

- Longitudinal Tracking
  - 0_cheat_sheet.html
  - 0_cheat_sheet.pdf
  - 0_Foreword.html
  - 0_Foreword.pdf
  - 0_Installation_instructions
  - 0_PythonExample.ipynb
  - 1_Introduction.html
  - 1_Introduction_optionA_empty.ipynb
  - 1_Introduction_optionA_solutions.ipynb
  - 1_Introduction_optionB_empty.ipynb
  - 1_Introduction_optionB_solutions.ipynb
  - 1_Introduction.pdf
  - 2_Advanced.html
  - 2_Advanced.pdf
  - 2_AdvancedTracking_empty.ipynb
  - 2_AdvancedTracking_solutions.ipynb
  - all_files.tar.gz
  - all_files.zip
  - all_in_one_tracker.py
  - PythonExampleFunctions.py
  - support_functions.py

Find the exercises in
Indico (Timetable → Introduction Afternoon courses)

We will need

- 1_Introduction_optionA_empty.ipynb

OR

- 1_Introduction_optionB_empty.ipynb

AND

- support_functions.py

The all_files archive files allows to get them all at once!

YOUR MISSION

WE WOULD LIKE TO SIMULATE THE BUNCH EVOLUTION IN SCSPS WITH TRACKING.

TWO OPTIONS ARE OFFERED IN THE EXERCISE

→ **OPTION A: Using the BLonD simulation code!**

→ **OPTION B: Coding your own tracker!** *(check the cheat sheet for formulas)*

CHOOSE ONE AND DON'T HESITATE TO ASK US FOR HELP! TEAM-UP WITH YOUR NEIGHBORS IF YOU WANT.

YOU'VE GOT ALL IT TAKES!!