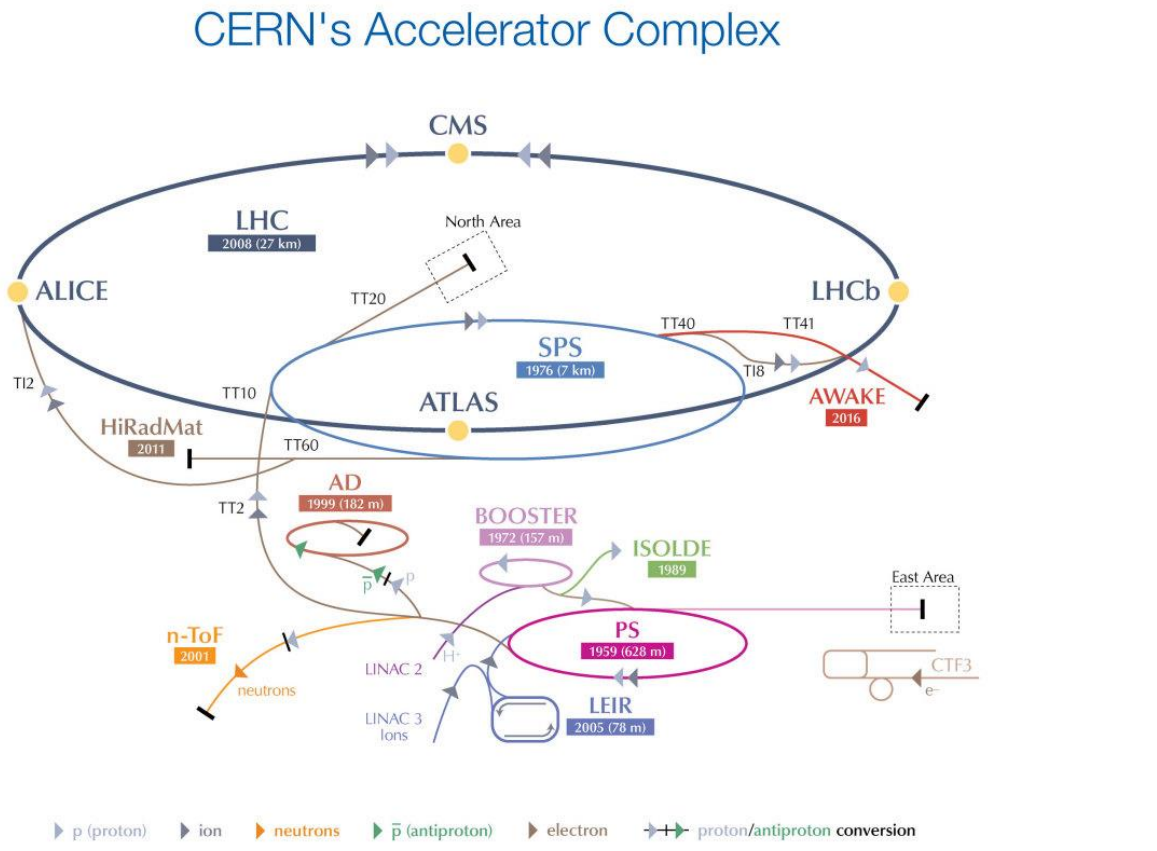
Longitudinal Beam Dynamics Simulations with BLonD

Danilo Quartullo, Helga Timko

CERN, BE-RF

Thematic CERN School of Computing, 18-22 May 2015

The CERN Accelerator Complex

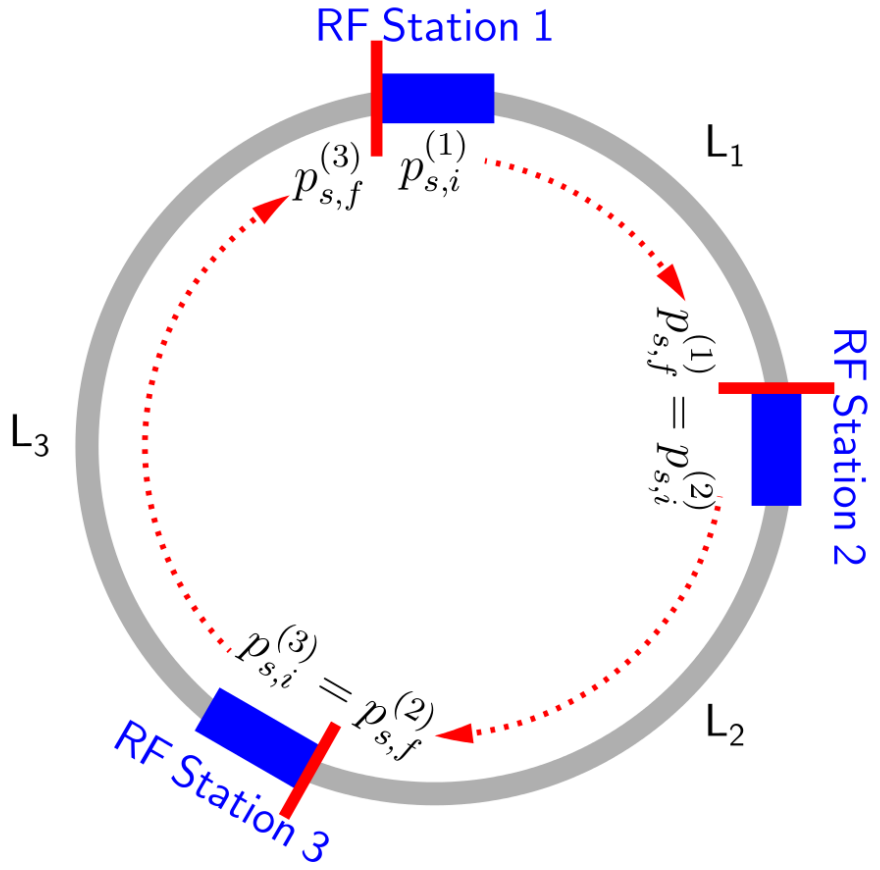


Motivation

- **Why do we care about the beam evolution over time?**
  - To design our machines
  - To control/manipulate the beam (splitting, shaping…)
    - PSB: 1 bunch (800 ns) in each of 4 rings, LHC: 2808 bunches (1.2 ns)
  - To ensure stable beams and safe operation
    - Unstable beams → loss of luminosity, radiation/safety issue
      - Full LHC beam stores 350 MJ, can drill a hole into the beam pipe!
- **What do we need to model?**
  - Charged particles on electromagnetic fields
  - Intensity effects (beam-machine interaction)
  - Feedback loops (machine dependent)
- **Complex machines, numerical models needed!**

Synchrotron model

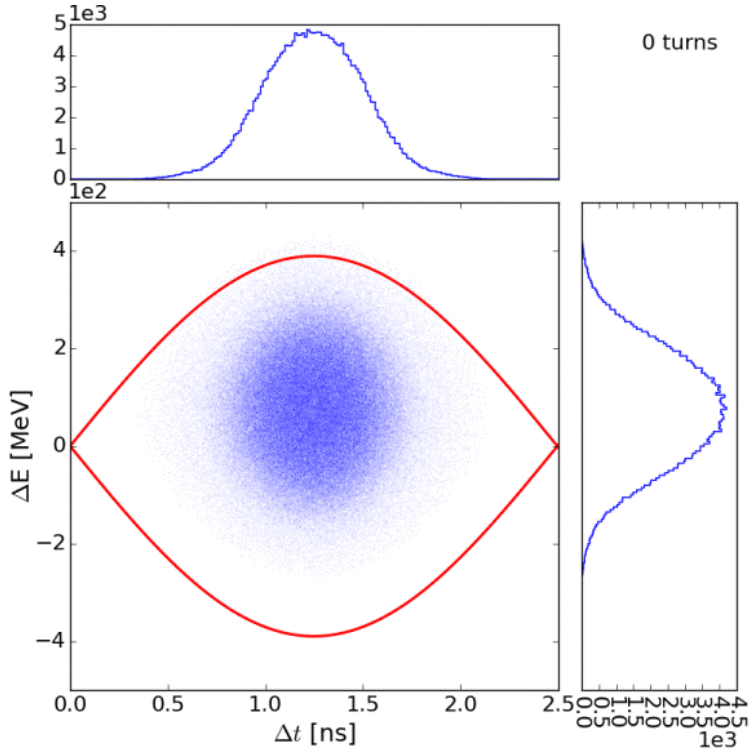
- **Equations of motion**
  - Nonlinear differential equations
  - Sinusoidal potential
  - Independent particle treatment
- **Intensity effects**
  - Accumulated effect from all particles “in front” (convolution)
  - Particles depend on each other
- **Feedback loops**
  - Interaction between RF frequency/phase and particles



Visual example

- **Beam capture in LHC**

  * grey = particles that hit the separatrix



- **Time axis = coordinate along the beam pipe**
- **Energy axis = particle's energy offset from the synchronous particle**
- **"Bucket" (red) = the limit of closed trajectories**
  - De-bunching outside of it

Simulated with BLonD http://blond.web.cern.ch

Histogram in python

- **Python built-in functions can be handy but also expensive**
- **E.g. numpy.histogram always does a sorting**
  - For **M** macroparticles and **S** slices the routine scales linear in **M** and **S** for **M > 65536 (=2^{16})**

```
Numpy.histogram: ( a = coordinate array, bins =e dge array, M = len(a),
                   S = len(bins)-1 )

block = 65536
for i in arange(0, len(a), block):
    sa = sort(a[i:i+block])
    n += np.r_[sa.searchsorted(bins[:-1], 'left'),
         sa.searchsorted(bins[-1], 'right')]

n = np.diff(n)
```

$\sim 2^{16} \log_2 2^{16} \times M/2^{16}$ \

$\sim S \log_2 2^{16} \times M/2^{16}$

$\sim S$

$$\Sigma = 16\,M\,(1 + S/2^{16}) + S$$

Histogram in c++

- **In comparison, the 'hand-made' histogram that uses no sorting is linear in M, but independent of S (because access time doesn't depend on array length)**

```
cpp_histogram: ( input = array of coordinates, output = array of edges,
                 M = len(input), S = len(output) - 1 )

inv_bin_width = n_slices / (cut_right - cut_left);      ~ c1

    for (i = 0; i < n_macroparticles; i++) {
        a = input[i];                                   ~ c2 M
        if ((a < cut_left)||(a > cut_right))            ~ c3 M
            continue;                                   ~ c4 M
        fbin = (a - cut_left) * inv_bin_width;          ~ c5 M
        ffbin = (uint)(fbin);                           ~ c6 M
        output[ffbin] = output[ffbin] + 1.0;            ~ c7 M
    }
```

$$\Sigma = c_1 + C\,M$$

An example of potential speed-up

- **Profiled with M = 50000, S = 100, 1000 iterations on a PC**
- **Improved histogram, VDT sine function, vectorised tracking**

numpy.histogram, python tracker

Function/Module	Total Time	Local Time
track	4.510	0.008
slice_constant_space_histo...	3.477	0.006
histogram	3.476	0.048
beam_coordinates	0.001	0.001
gaussian_fit	1.024	0.014
convert_coordinates	0.001	0.001
plot_long_phase_space	3.064	0.001
track	1.877	0.008
kick	1.456	1.455
drift	0.385	0.216
kick_acceleration	0.027	0.027

C++ histogram, C++ tracker

Function/Module	Total Time	Local Time
track	1.191	0.008
gaussian_fit	0.993	0.014
slice_constant_space_histo...	0.188	0.167
data_as	0.048	0.016
__init__	0.026	0.026
beam_coordinates	0.001	0.001
__getattr__	0.000	0.000
convert_coordinates	0.002	0.002
loadtxt	1.113	0.335
track	0.747	0.611
drift	0.082	0.067
data_as	0.048	0.016
__init__	0.026	0.026
ascontiguousarray	0.016	0.006
__getattr__	0.000	0.000