Electron-Cloud Module for the ORBIT Code

A. Shishlo, Y. Sato, J. Holmes,
S. Danilov, S. Henderson,
SNS project, ORNL

(April 21, 2004)

Outline

1. Why another electron cloud model
2. Why the ORBIT code
3. Simulation Approach
4. E-Cloud Module in the ORBIT Structure
5. Base Classes of the Electron Cloud Module
6. Algorithm
7. Conclusions

SNS Project , Oak Ridge, TN

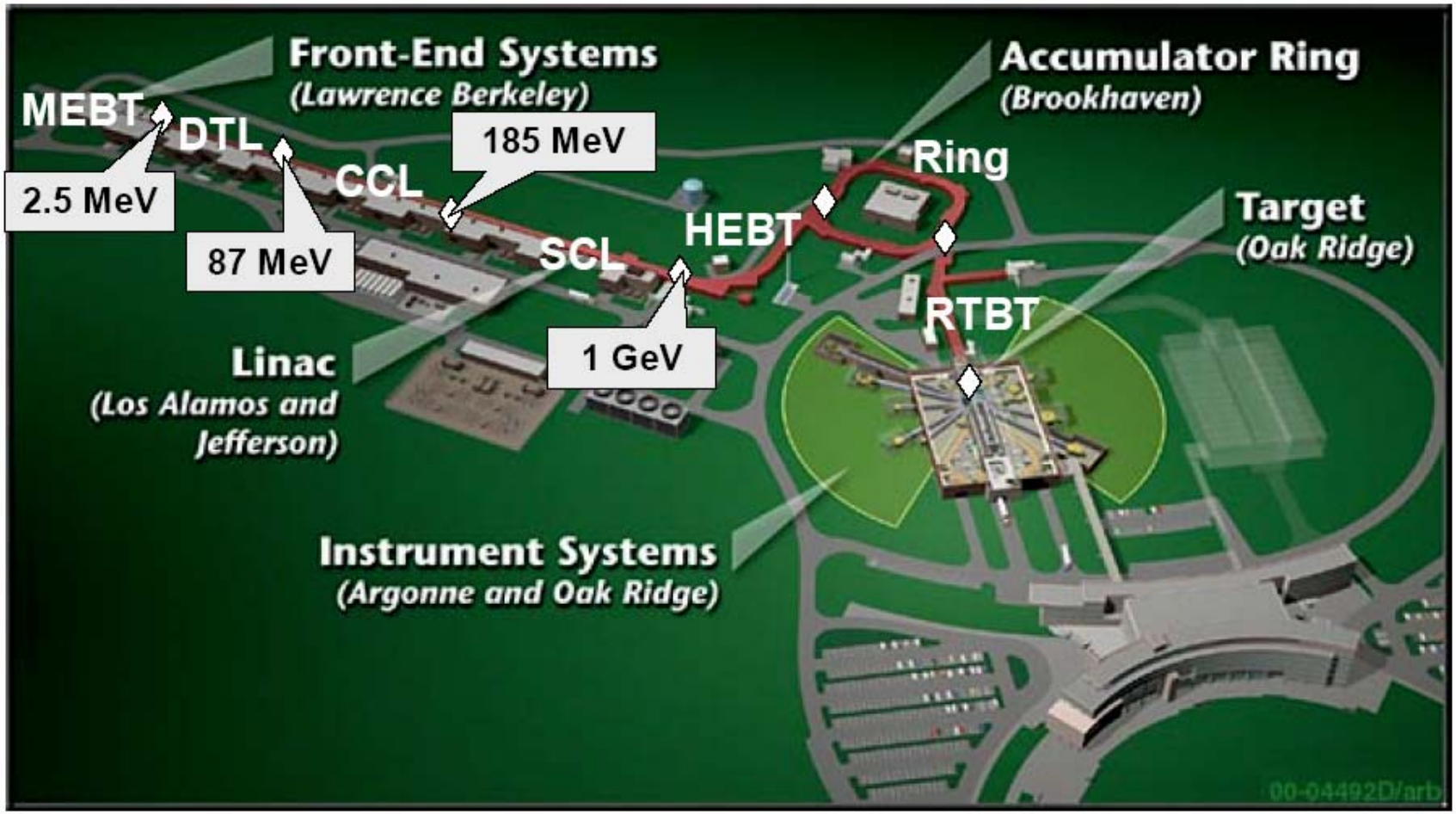


Figure 1: Layout of the SNS facility

Beam Energy	**1 Gev**
Ring Beam Intensity	**1.5×10^{14}**
Repetition Rate	**60 Hz**

Accumulated Turns	**1060**
Accumulation Time	**1.0 msec**
Beam Power on Target	**1.44 MW**

Why Another Electron Cloud Model?

Present status:

The ORBIT E-Cloud Module is now fully implemented in ORBIT. Benchmarking of the secondary electron emission model and electrons-protons dynamics have been carried out (as a first step to apply ORBIT with e-cloud module to PSR and SNS cases)

Why Another Electron Cloud Model?

To study the effect of electron clouds on the dynamics of the proton beam.

This electron cloud model :

- **should be embedded in an existing accelerator code**
- **Includes the interactions of the electron cloud and the proton beam in both directions**
- **describes the electron cloud build up and includes a secondary emission surface model**
- **Should be a parallel code. For a PIC code there is no hope to simulate this type of dynamics by using only one CPU.**

Why the ORBIT Code

ORBIT: Developers and Collaborators

- **SNS at ORNL, FermiLab, SNS at Brookhaven, Indiana University, LANL, TRIUMF**

ORBIT is a particle tracking code in 6D phase space. Its purpose is the design and analysis of high intensity rings.

ORBIT is designed to simulate real machines: it has detailed models for

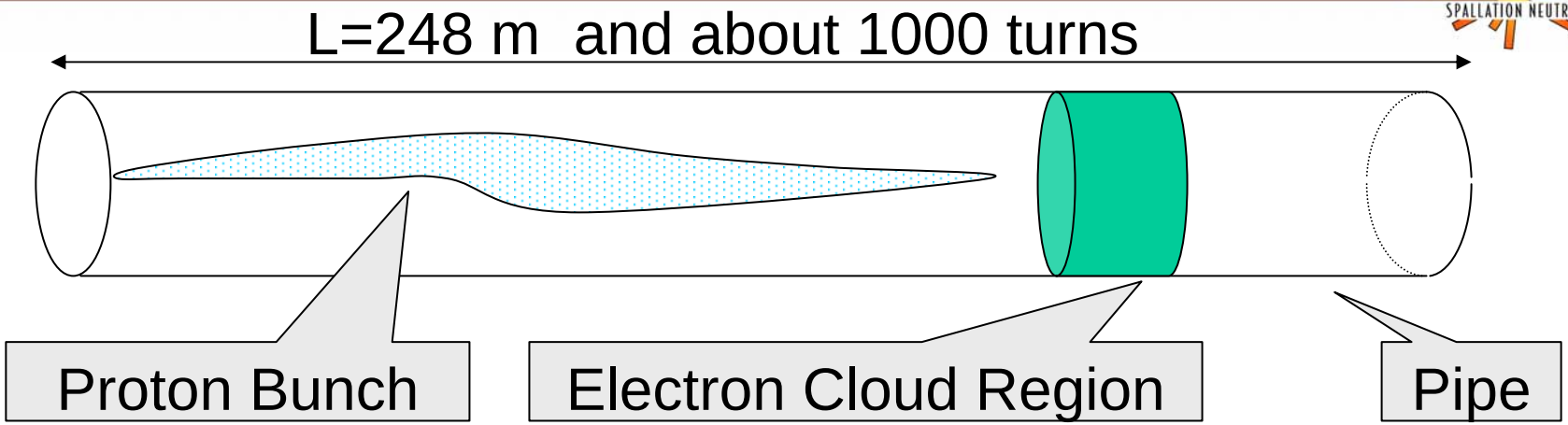
- **Injection foil and painting**
- **Single particle transport through various types of lattice elements**
- **Magnet Errors, Closed Orbit Calculation, Orbit Correction**
- **RF and acceleration**
- **Longitudinal impedance and 1D longitudinal space charge**
- **Transverse impedance**
- **2.5D space charge with or without conducting wall beam pipe**
- **3D space charge**
- **Apertures and collimation**
- **ORBIT has an excellent suite of routines for beam diagnostics**
- **. . . more**

ORBIT supports parallel processing based on MPI

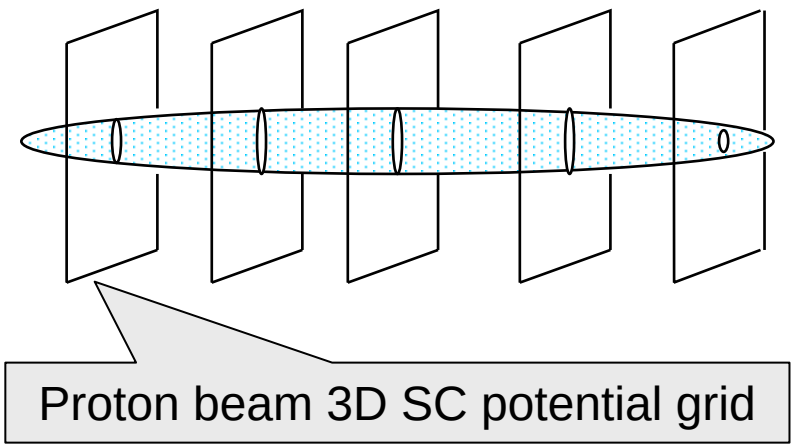
ORBIT is open source code

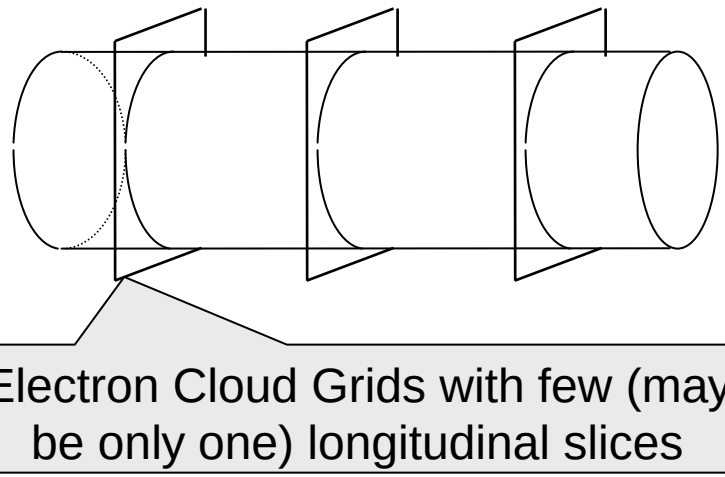
- (contact person: Jeff Holmes, SNS project, ORNL)

Simulation Approach

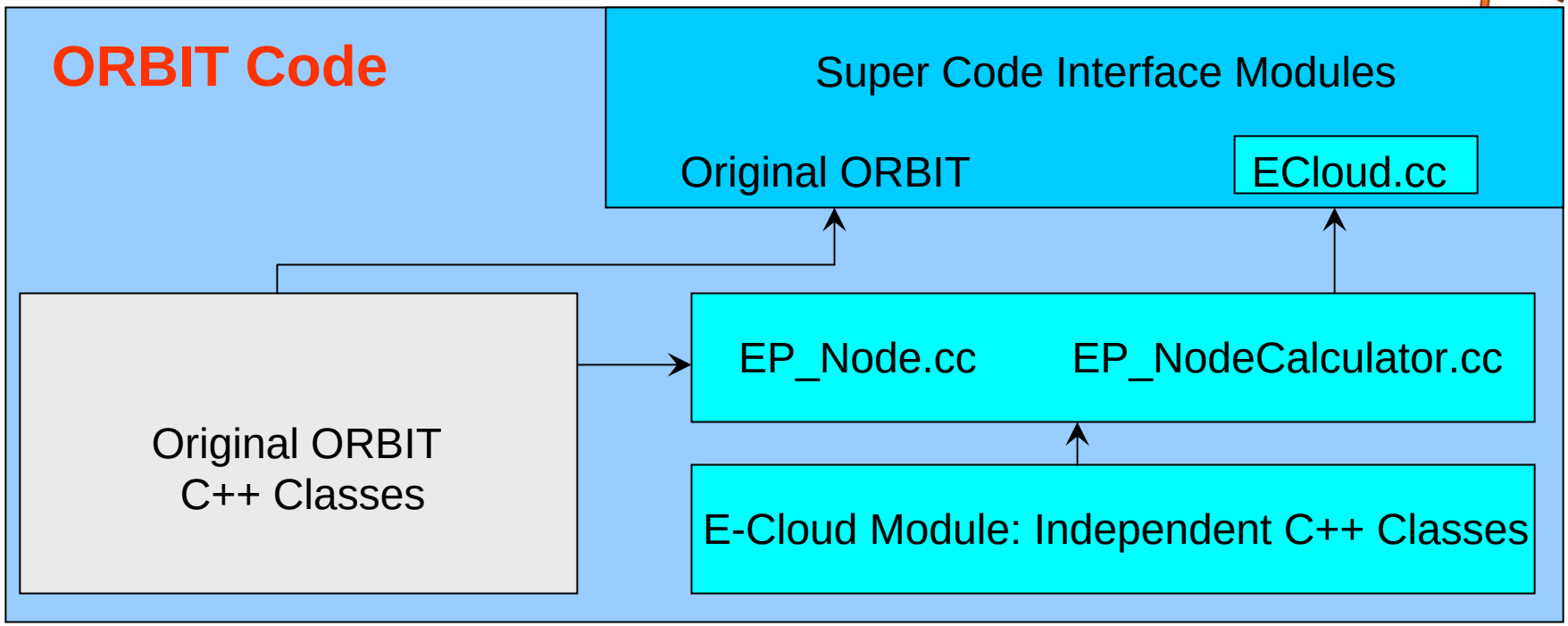


- **We have to simulate a building up an electron cloud, its dynamics, its effect on a proton bunch during the whole accumulation period or at least for several turns to detect the development of instability.**
- **We are going to use PIC method for both p-Bunch and e-Cloud.**





E-Cloud Module in the ORBIT Structure



- ORBIT Electron Cloud Module

- The ORBIT E-Cloud Module is a collection of C++ classes. Only three classes connect the E-Cloud module with the original ORBIT code, so the module can be easily modified to use in other accelerator code or independently.
- The special efforts have been made to provide the possibilities for an extension of existing classes and improvement of the models.

Base Classes of the Electron Cloud Module

eBunch	**Keeps 6D – coordinates of the macro-electrons, provides method to add and to delete macro-particles. It has parallel capabilities.**
EP_Boundary	**It is a 2D SC solver and keeps the transverse 2D grid parameters.**
Grid3D	**3D grid with references to the EP_Boundary class. It has parallel capabilities. The subclasses deal with SC density and potential.**
Surfaces Classes	**Collection of the classes describing different surfaces.**
Field Source Classes	**The collection of classes specifying electrostatic and magnetic fields. Now it includes p-Bunch, e-Bunch, and uniform fields.**
Tracker	**Tracks macro-particles by using arbitrary set of field sources.**
EP_Node EP_NodeCalculator	**These classes connect the base electron-cloud classes to the rest of ORBIT code.**

eBunch Class

eBunch Class is a resizable container that keeps information about macro-electrons:

- **6D coordinates – x, y, z, px, py, pz**
- **macro-size**
- **dead/alive flag**

It provides the following methods to operate with macro-electrons:

- **access to each of the 6D coordinates**
- **add macro-electron**
- **delete macro-electron**
- **print all information into a file**
- **create all macro-electrons by reading the external file**

Its parallel capabilities are used when it reads and writes the content of the electron bunch into or from the external file.

We tried to do it as faster as possible.

EP_Boundary Class

EP_Boundary class :

– keeps information about 2D transverse grid, beam-pipe shape and size in the XY-plane (the shape could be a circle, ellipse, or rectangle)

– is a 2D Poisson solver (Convolution method). It has the method that accepts a 3D grid with space charge density and returns another 3D grid with potential values at the grid points. Each XY-slice of the potential 3D grid is a solution of the 2D space charge problem for the XY-slice of the space-charge density grid.

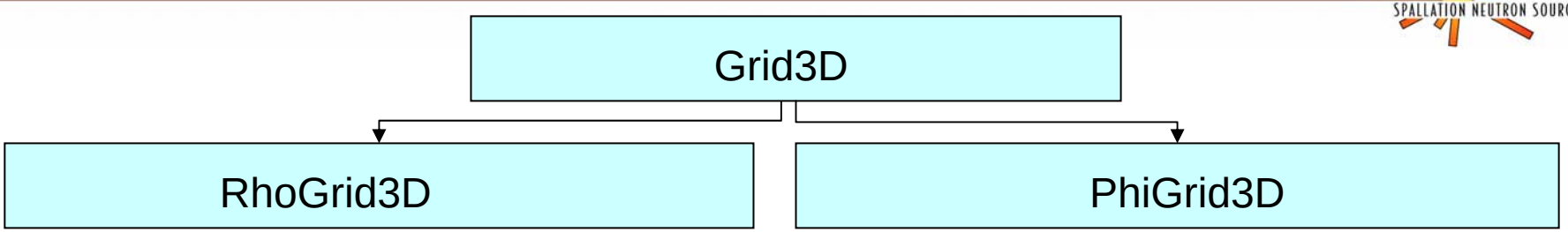
– uses FFTW library and keeps necessary arrays inside

– Can add a boundary conditions (zero potential on the beam-pipe) to the potential 3D grid by using the Capacity Matrix Method

– for an electron hitting the surface of the beam-pipe it finds an impact point on the surface and calculates its normal vector by using internal geometry information

– does not have any parallel capabilities

Grid3D Classes

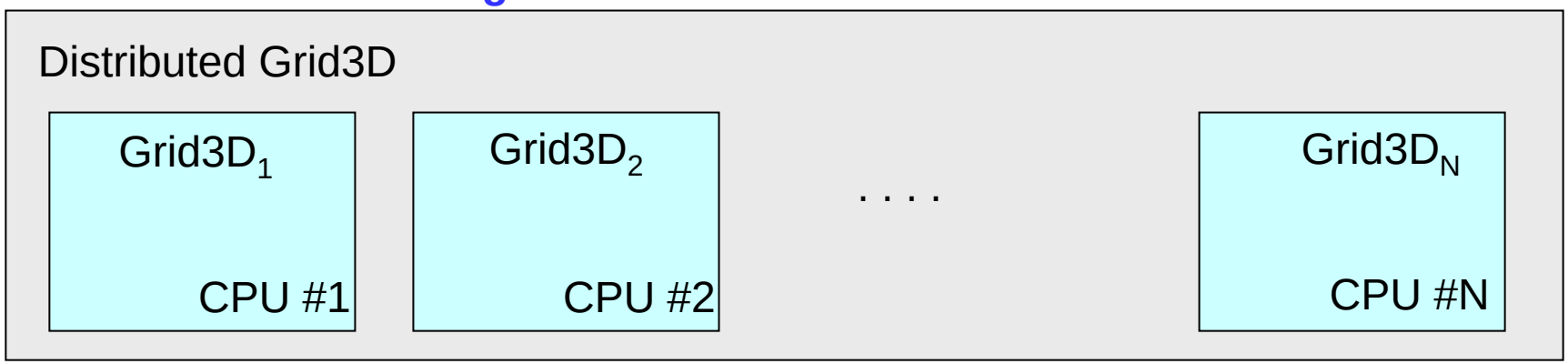


The Grid3D class is the parent class for a Grid3D class hierarchy :

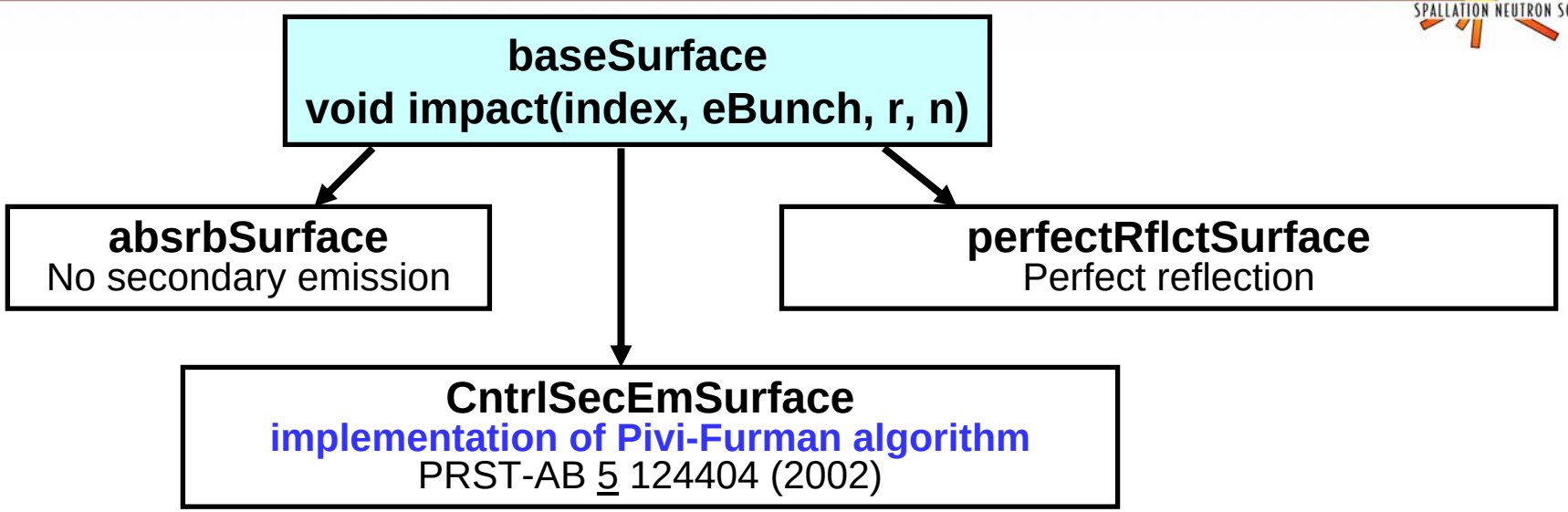
- **keeps the 3D array of double values inside itself**
- **provides direct access to the 3D array and to the 2D slices**
- **calculates a value and gradient at an arbitrary point inside the 3D grid and bins macro-particles by using 3x3x3 points scheme**

The RhoGrid3D and PhiGrid3D classes are the subclasses of the Grid3D class and provide methods specific for space charge density and potential grids

The parallel capabilities: the Grid3D class has methods that transform it into a distributed 3D grid



Surface Classes



A surface class should implement impact method of the abstract baseSurface class. It removes the macro-electron with a particular index from the eBunch and adds the emitted new macro-electrons to the eBunch at the specific point on the beam-pipe surface.

We can create easily new surface classes if we need new models.

The details of the implementation of the Pivi-Furman algorithm will be discussed on the next slide.

Implementation of Furman-Pivi Algorithm (1)

Rationale:

Furman-Pivi algorithm of secondary emission assumes that all primary and secondary macro-electrons have the same macro-size. As result the number of macro-electrons increases exponentially during the electron cloud build up for single bunched proton beam passage. Calculation time grows significantly, so we wanted to avoid this effect and gained control over the macro-electrons population in the electron cloud.

The definition of the variables in the secondary emission algorithm:

M_{in} – macro-size of the electron hitting the surface

M_{out} – macro-size of the resulting electron

$$\delta = \delta_e + \delta_r + \delta_{ts}$$

Total SEY is the sum of "elastic", "rediffused", and "true secondary" components

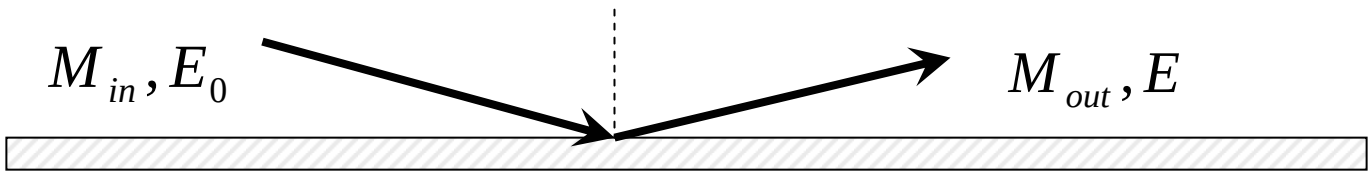
$$p_e = \delta_e / \delta; \quad p_r = \delta_r / \delta; \quad p_{ts} = \delta_{ts} / \delta$$

$$M_{out} = 0 \quad \text{if} \quad G < f_{death}(E_0)$$

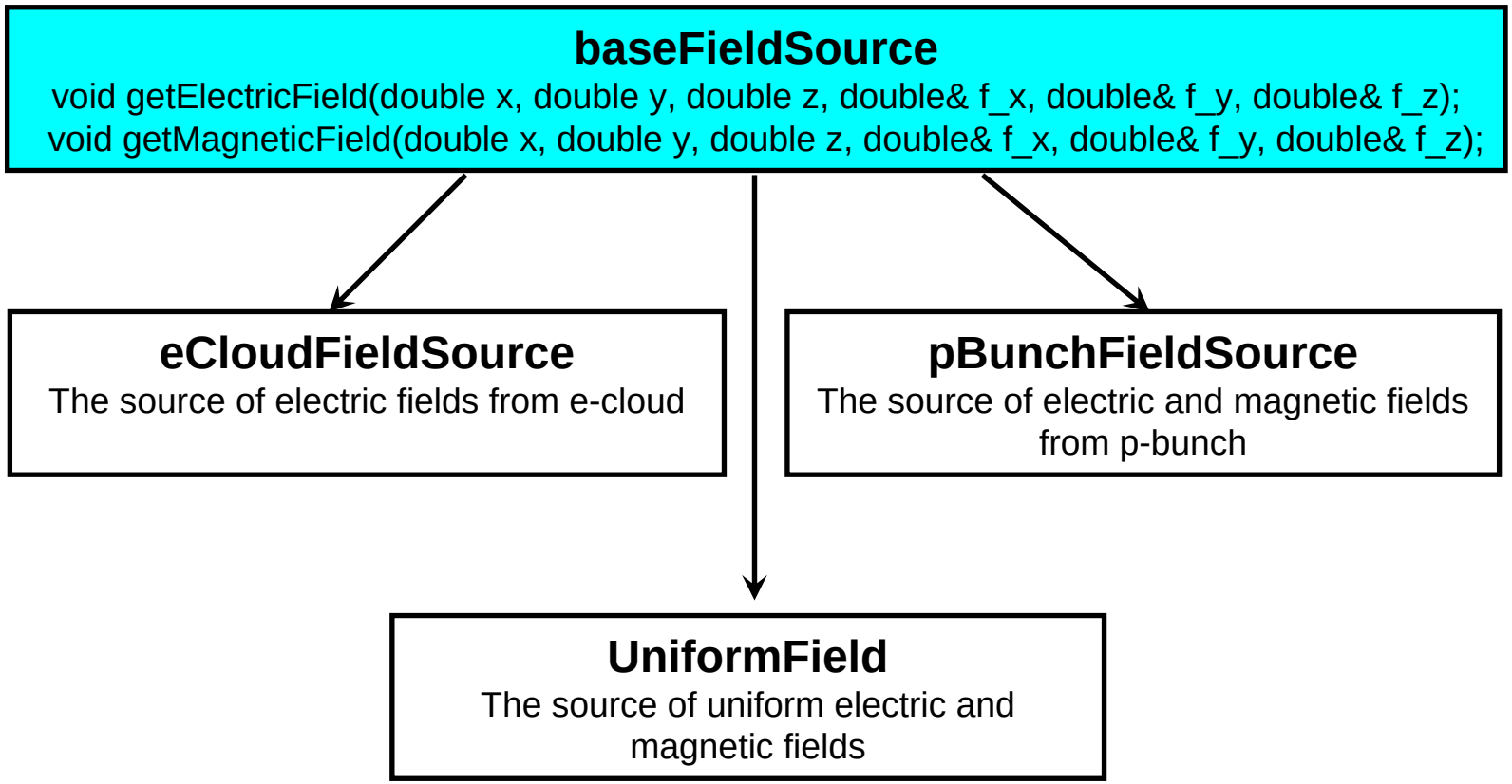
$$M_{out} = M_{in} \cdot \delta / (1 - f_{death}(E_0))$$

G – random number between 0-1

$f_{death}(E_0)$ – user defined function, usually it is equal to 0 if E_0> 1 eV



Field Source Classes



We can create additional sources. For instance, It can be a magnetic field of dipoles, quads etc.

Tracker Class

baseParticleTracker – the parent class of all trackers

The subclasses should implement only one method: the method that will move macro-electrons.

The main method:

```
moveParticles(double time,
              eBunch* eb,
              Grid3D* rhoGrid3D,
              baseSurface* surface)
```

Where “time” - time of tracking, “eb” – macro-electrons bunch

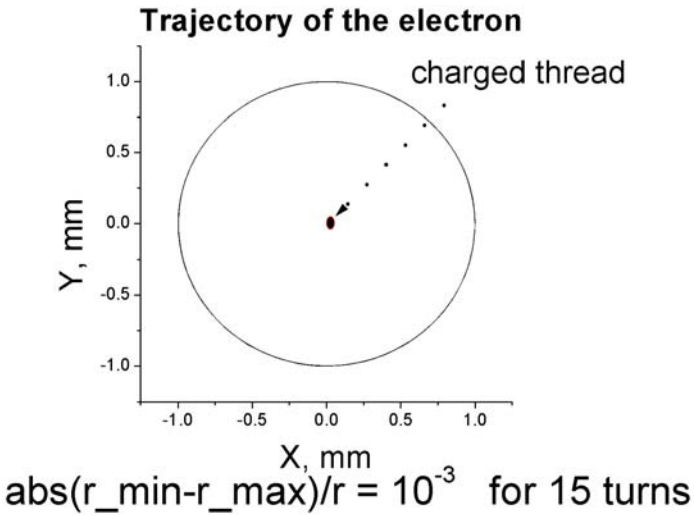
We can add as many sources as we want:

```
addMagneticFieldSource(BaseFieldSource* fs)
addElectricFieldSource(BaseFieldSource* fs)
```

At this moment there are two methods implemented to calculate the nonrelativistic electron’s motion:

- can be integrated symplectically using a leapfrog method
- can be integrated analytically, using a constant local field approximation

Tracker Benchmark

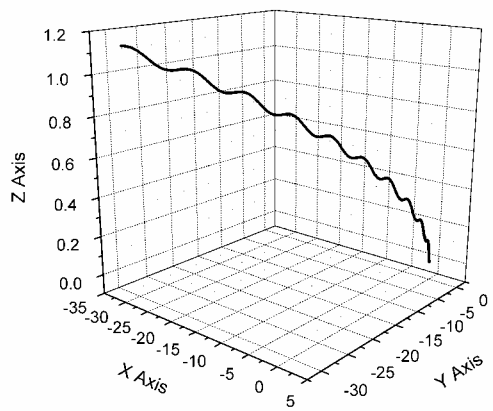


abs(r_min-r_max)/r = 10^{-3} for 15 turns

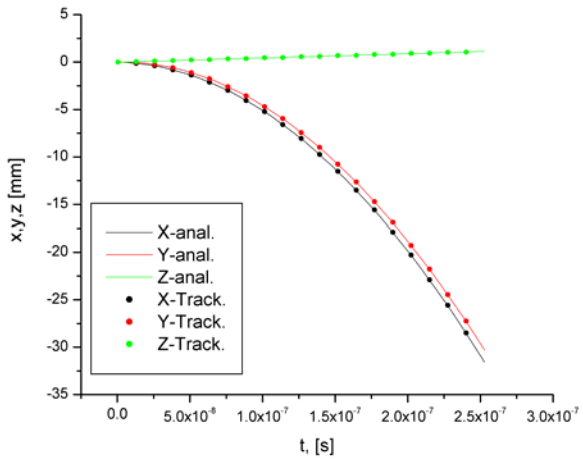
$$\frac{mv^2}{R} = \frac{\lambda}{2 \cdot \pi \cdot \varepsilon_0 \cdot R}$$

This is the test for 2D solver and the tracker.

Trajectory of the Electron in B and E



Motion of the Electron in B and E



EP_Node and EP_NodeCalculator

EP_Node – the subclass of the Node class of the ORBIT code:

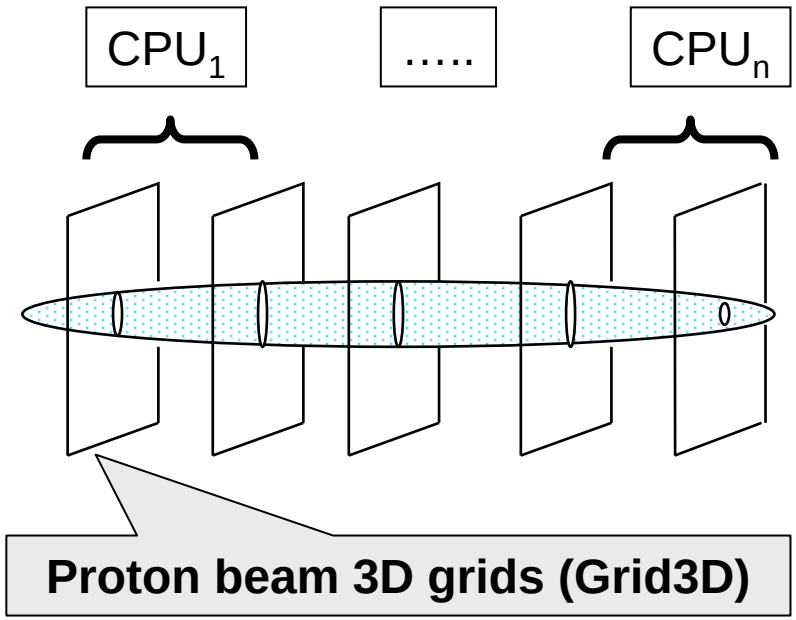
- it represents a accelerator lattice element through which the proton bunch should be propagated.
- there could be an arbitrary number of this nodes in the lattice. Each node has its own set of macro-electrons in the e-bunch.
- it uses EP_NodeCalculator to propagate the proton bunch through the e-cloud region

EP_NodeCalcularor – the class that actually combines all classes together and implements our algorithm

- 1. preparation for calculation:
  - checking the sizes of the arrays and resizing if necessary
  - proton bunch analysis, fields calculation
- 2. simulation of the electron cloud build up and the electron cloud field having an effect on the proton bunch
- 3. applying accumulated kicks from electrons to the protons

Algorithm (1)

Stage 1. Preparations



At this stage we dealing with the proton bunch only. The macro-particles of this bunch are distributed between CPUs.

- **get information about distribution of the slices between CPUs. This data are provided by a special class – ParticleDistributor**
- **resize if necessary 3D arrays: space charge density, space charge potential, x, y, z kicks – 5 distributed 3D grids**
- **bin macro-particles of the proton bunch into the 3D space charge density grid. Provide necessary communication between CPUs to produce distributed 3D grid**
- **find the proton bunch potential**
- **calculates proton line density**

The space charge density grid and proton line density are used in the future calculation if electrons are produced by residual gas ionization or by proton losses on the chamber wall

Memory $M=N_x*N_y*N_z*5$ 2000x64x64 x 5 -> 200 Mb – 1 Gb

Algorithm (2)

Stage 2. Propagation p-bunch through e-cloud

During the turn, as time progresses the proton grid is moved through the electron cloud region at the beam velocity. It is done by three nested loops.

1-st nested loop (upper level)

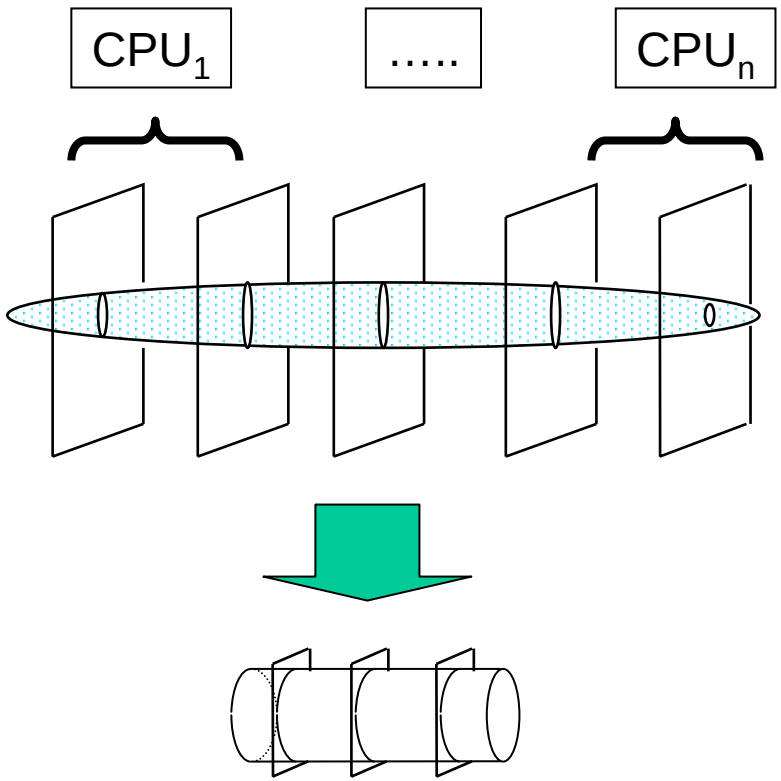
$T1_{step} = T_{rev}/N1_{step}$ **$N1_{step}$ = 2000 – 10000 for PSR or SNS case**

- **The number of steps is defined by the requirement adiabatic changes in the electron cloud potential.**
- **In the beginning of this iteration, primary electrons are generated by routines simulating protons grazing the vacuum chamber or residual gas ionization. The generated macro-electrons are distributed randomly between CPUs. During the calculations they reside at the same CPU where they have been generated. No macro-electrons distributor is needed.**
- **the space charge potential of the electrons is calculated. This potential is sum of all potentials trough all CPUs, so communications between CPUs are needed. This potential is using as one of the field sources for the Tracker.**
- **execute the 2-nd nested loop (intermediate level)**
- **accumulate kicks on the proton bunch into the grids for x,y,z directions**

Algorithm (3)

Stage 2. Propagation p-bunch through e-cloud

2-st nested loop (intermediate level)



$T2_{step} = T1_{step}/N2_{step}$ $N2_{step} = 1 – 5$

The potential of proton beam at the specific position is used to update the proton beam field source for the Tracker.

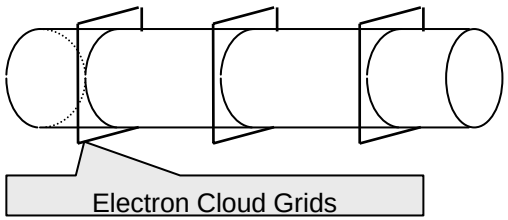
The adiabatic changes in the proton beam potential could be more important, so there is a possibility to update the proton beam field source more frequently comparing to e-cloud field source. Usually it is enough to update fields simultaneously (N2 = 1).

This step require communications between CPUs, because the p-beam potential grid is distributed.

Algorithm (4)

Stage 2. Propagation p-bunch through e-cloud (Continue)

3-rd nested loop (lower level)



$T3_{step} = T2_{step}/N3_{step}$ $N3_{step} = 5 – 20$

$N_{total} = N1xN2xN3$ – tens of thousands

Inside this loop we integrate the equation of motion of the electrons and consider hitting the surface of the beam pipe.

- **Each electron must be tested to determine whether it remains in the electron cloud region**
- **If an electron crosses the beam pipe boundary, the Pivi-Furman electron-wall interaction routines must be run for that electron.**
- **If an electron leaves the longitudinal node end, its longitudinal velocity is reflected, i.e. it bounces off the end.**

Stage 3. Propagation p-bunch through e-cloud

We are applying accumulated kicks from electrons to the protons.

Conclusions

- The electron cloud module has been inserted into the ORBIT code
- The dynamics of proton-electrons interaction can be included into simulations
- The real calculations can be carried out only on parallel computers
- The initial benchmarks has been performed (next talk, Y. Sato)