Training code description

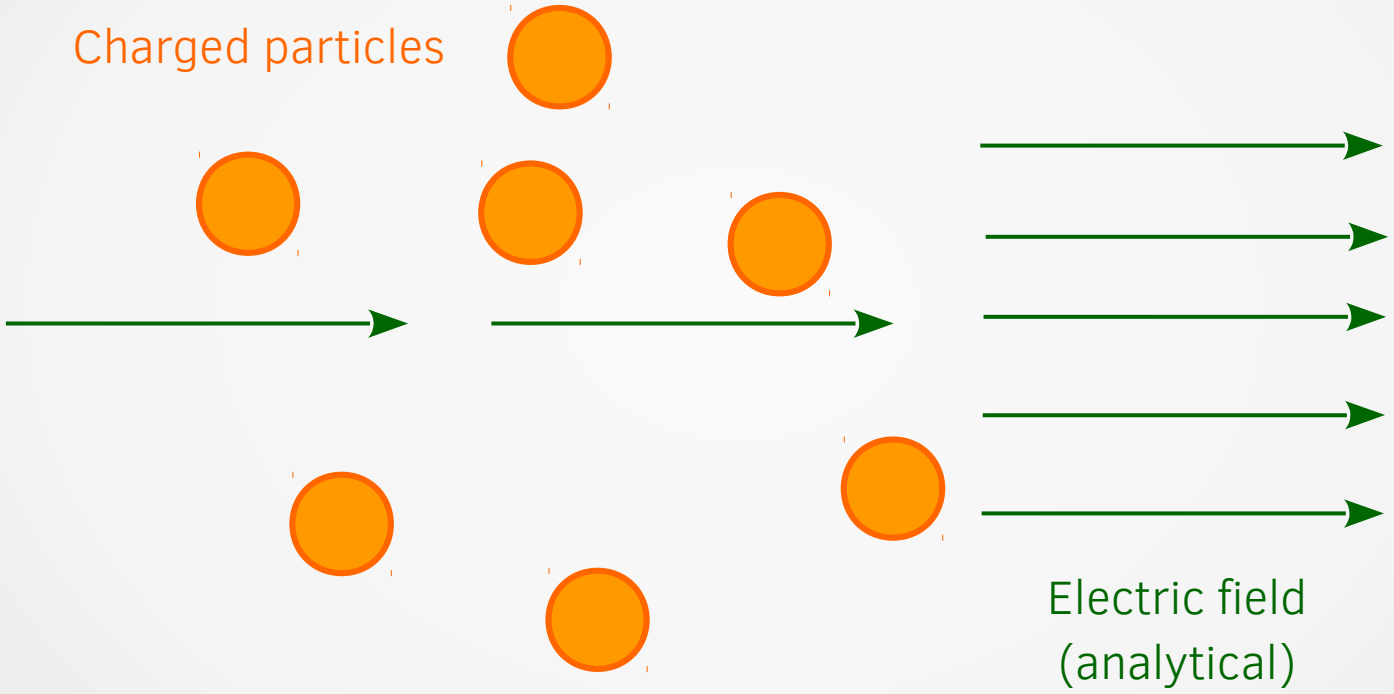
Mathieu Lobet, Matthieu Haefele

Maison de la Simulation, CEA, CNRS, Université Paris-Sud, UVSQ,
Universite Paris-Saclay, F-91191 Gif-sur-Yvette, France
(mathieu.lobet@cea.fr)

Simplified Particle-In-Cell code

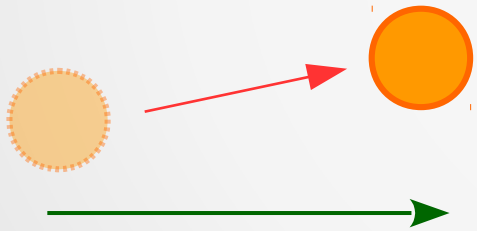
https://gitlab.maisondelasimulation.fr/mlobet/PIC-optimization-tutorial

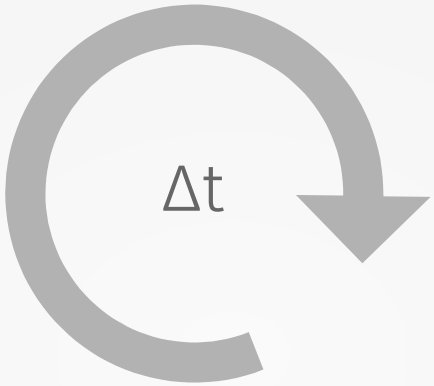
Simplified Particle-In-Cell code



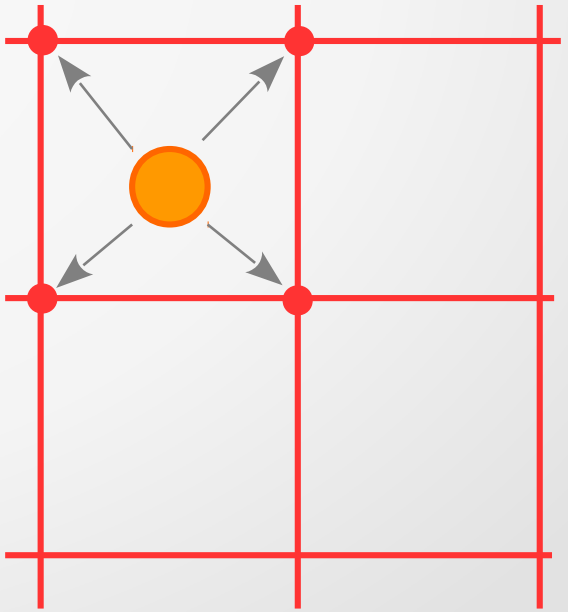
Only 2 phases in the time step

Move the particles using the analytical field





Deposition of the charge on a grid (order 1 interpolation)

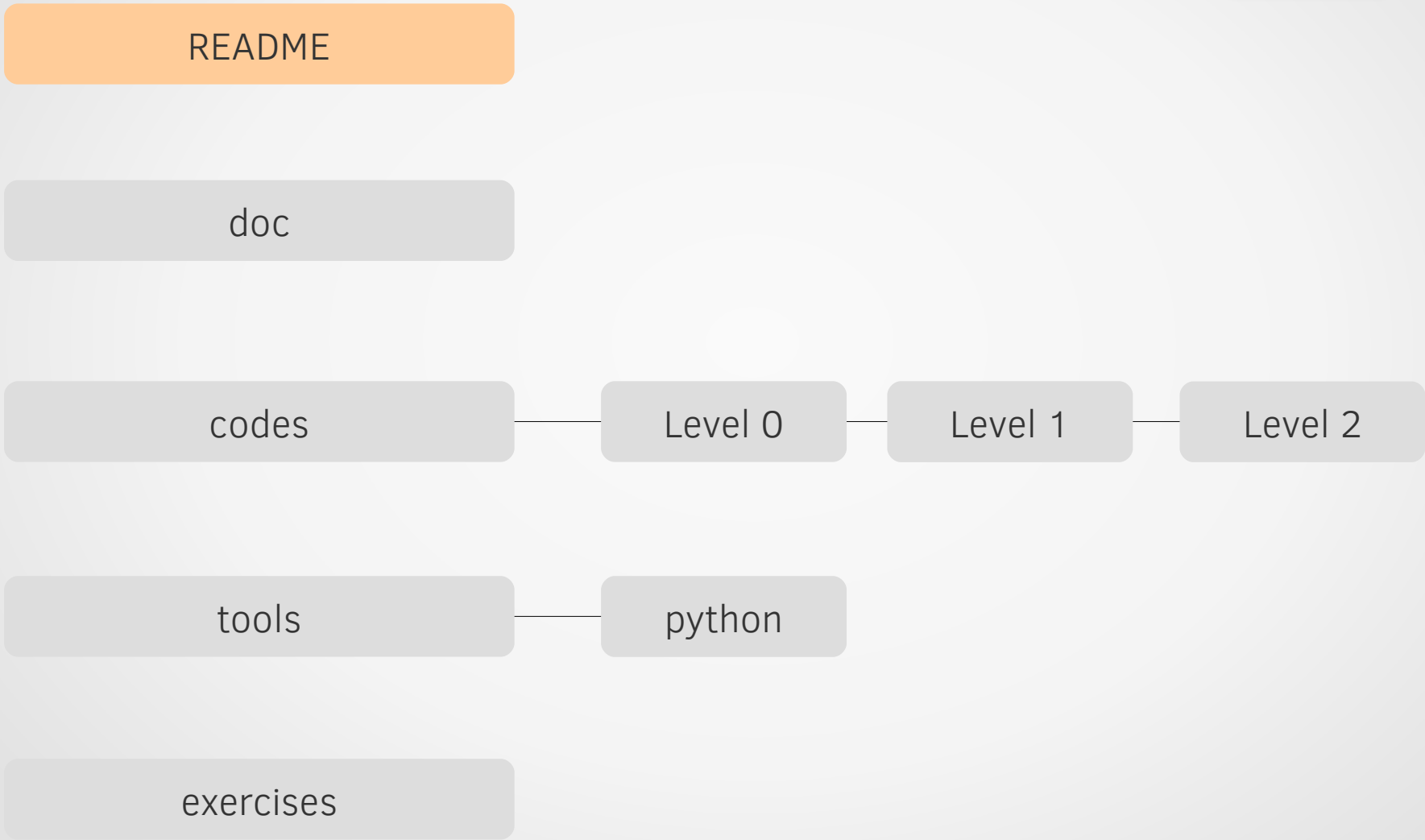


Charge grid

Code properties

- Fortran
- MPI Parallelization
- Sequential or Parallel HDF5 output
- Different levels of optimization in /codes/*

Repository structure



Code file structure

Main.F90	——	Initialization and time loop
Parameters.F90	——	All the parameters and data structures
Parallelism.F90	——	Parallelism, parameters and data initialization
Particles.F90	——	Particle pusher and boundary conditions
Charge.F90	——	Charge deposition and boundary conditions
Diagnosctics.F90	——	Scalar diags and HDF5 output
Tools.F90	——	Tool functions
Timers.F90	——	Advanced timers

Compilation

```
Git clone git@gitlab.maisondelasimulation.fr:mlobet/PIC-optimization-tutorial.git
```

```
Module load intel/19.0.0.nodes
```

```
Export HDF5_DIR=/obs/mlobet/libraries/hdf5-1.10.3_intel19.nodes/build/hdf5/
```

- Use the makefile

```
FC=mpiifort make archi="ivy bridge"
```

Execution and arguments

- The meaning of the command line arguments is reminded in the documentation

```
Mpirun -np 4 ./main
-procs 2 2
-steps 100
-size 1 1
-cells 128 128
-diags 10
-prints 10
-case 2
-verbose
```

Visualization of the results

- Python scripts available in `/tools/python/*`
- Diags generated in the directory `diags`
- To plot a specific time step :

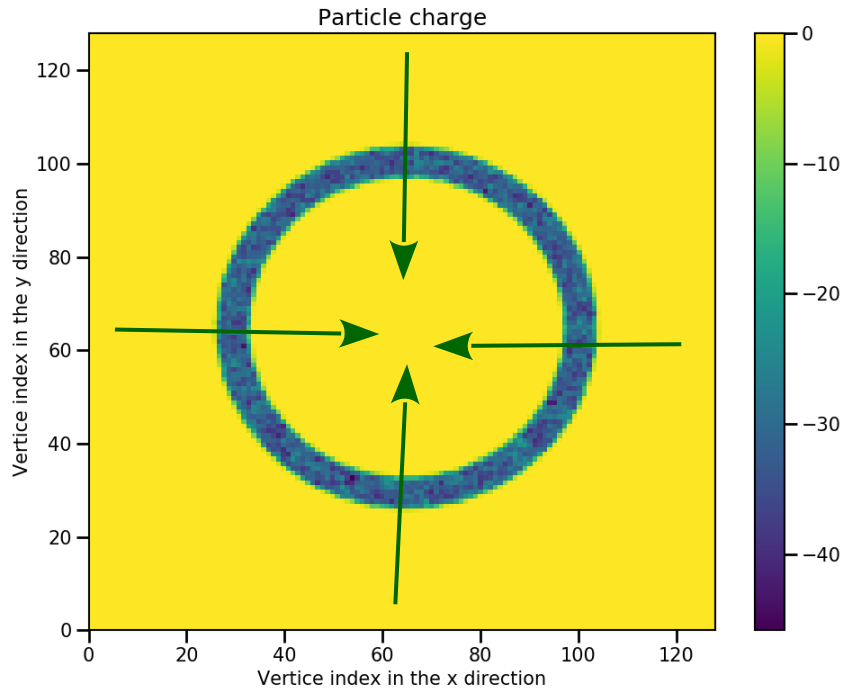
```
Python /tools/python/plot.py diags/charge_00500.h5
```

- To plot an animation of all files :

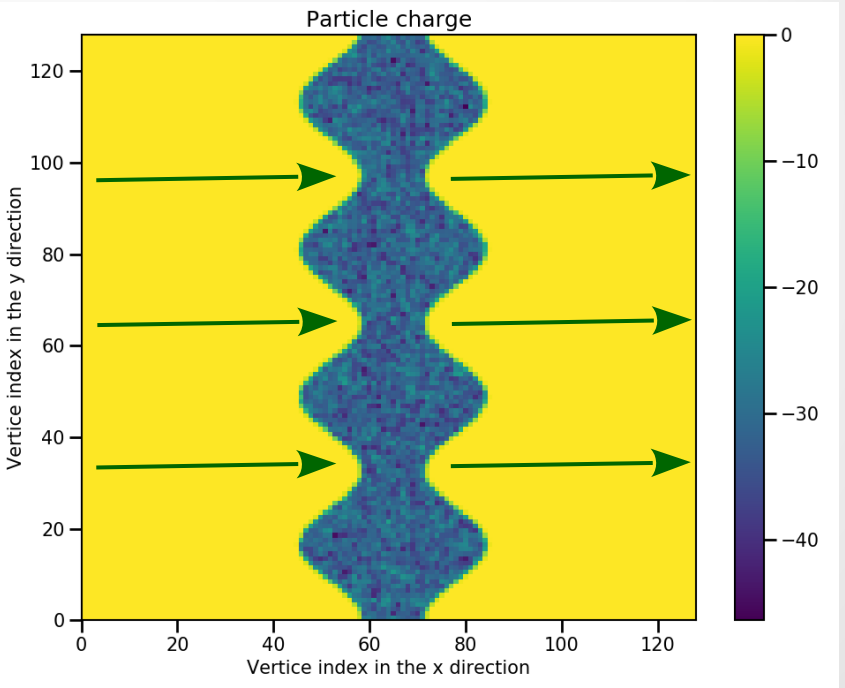
```
Python /tools/python/animate.py diags/
```

Cases

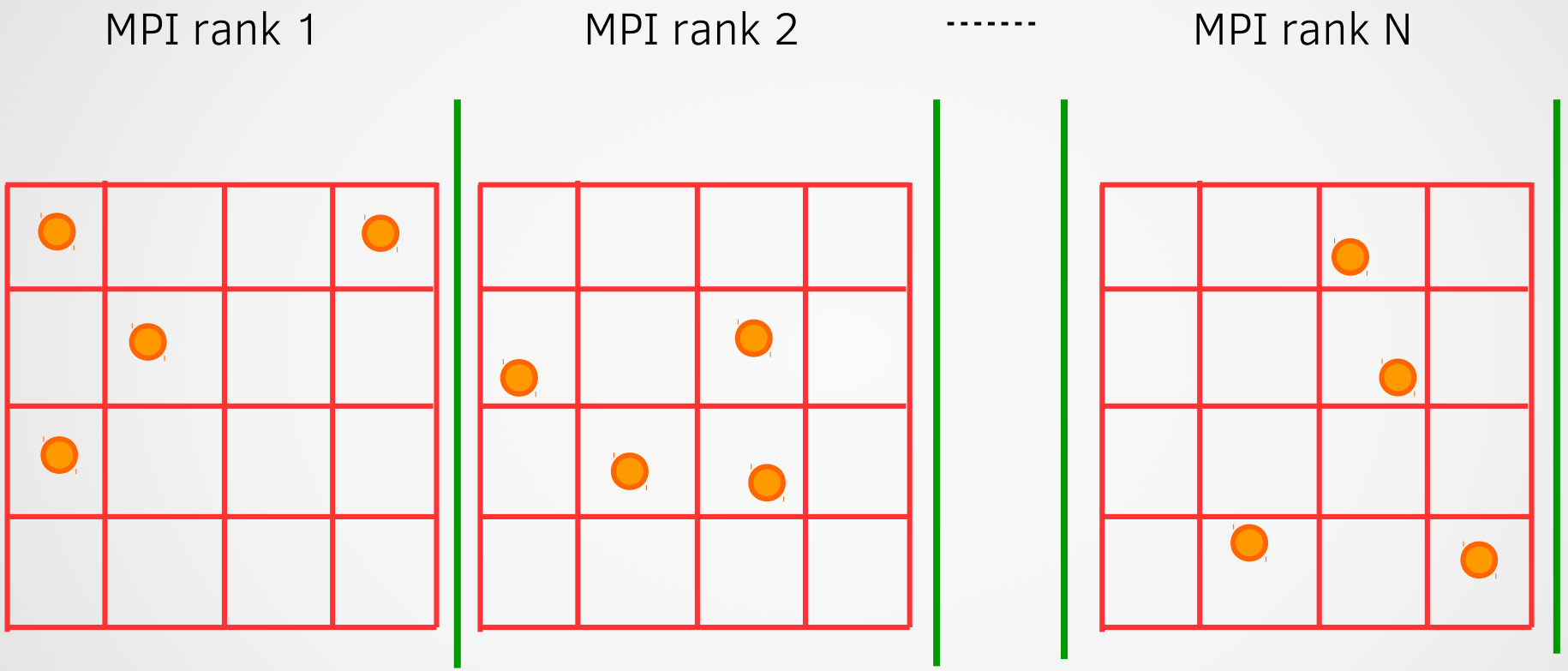
Case 1



Case 2

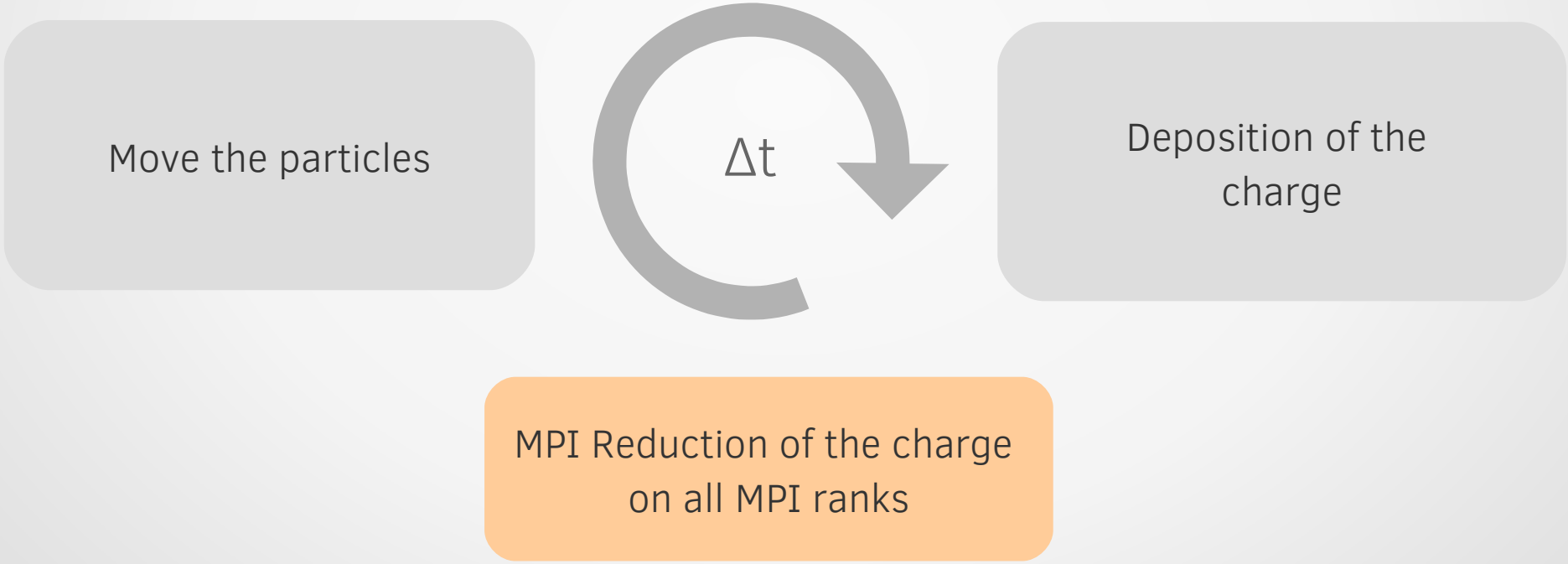


Level 0 – Particle MPI decomposition



- Particles equally shared accross MPI domains
- All the MPI domains have the full charge grid

Level 0 – Particle MPI decomposition



Level 0 – Data structure

Particle structure:

```
Real(8) :: weight
Real(8) :: x, y
Real(8) :: mx, my
```

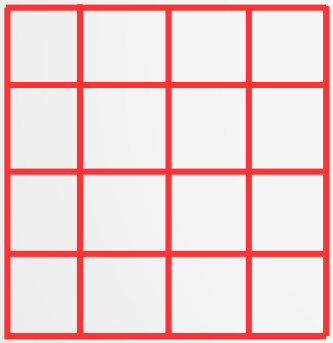
List of particles : array of structures

1 2 ---------- N

w(1), x(1), y(1), mx(1), my(1) ---------- w(N), x(N), y(N), mx(N), my(N)

Data order in memory

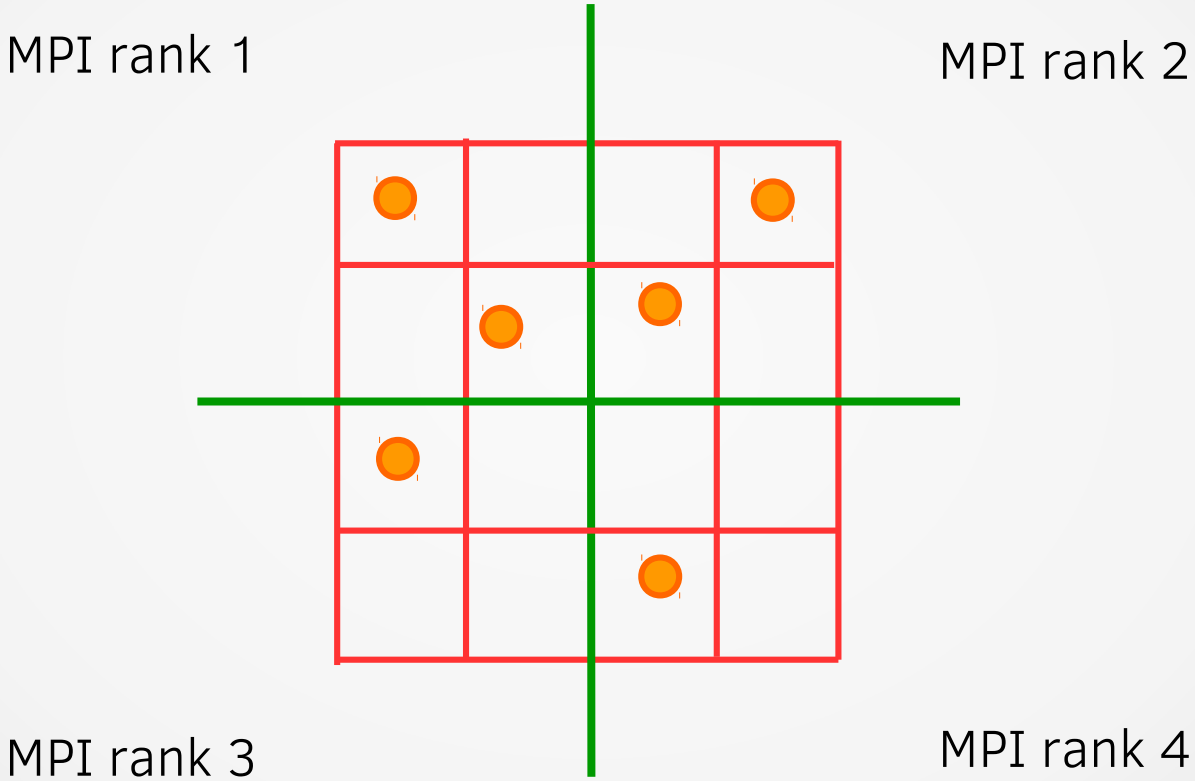
Level 0 – Data structure



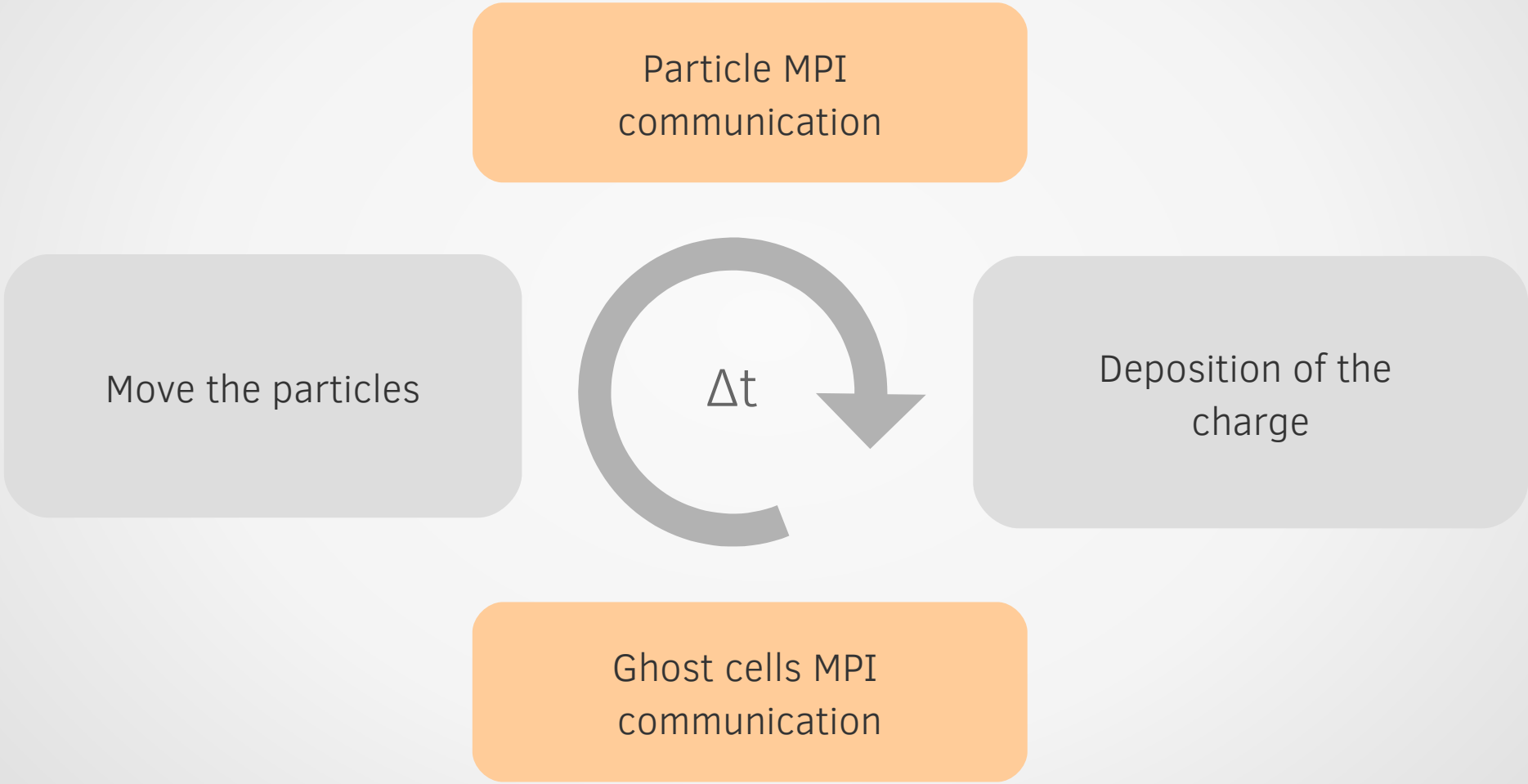
2D array

```
Real(8), dimension(:,:) :: charge
```

Level 1 – MPI domain decomposition



Level 1 – MPI domain decomposition



Level 2 – Structure of Array instead of Array of Structure

List of particles becomes structure of arrays

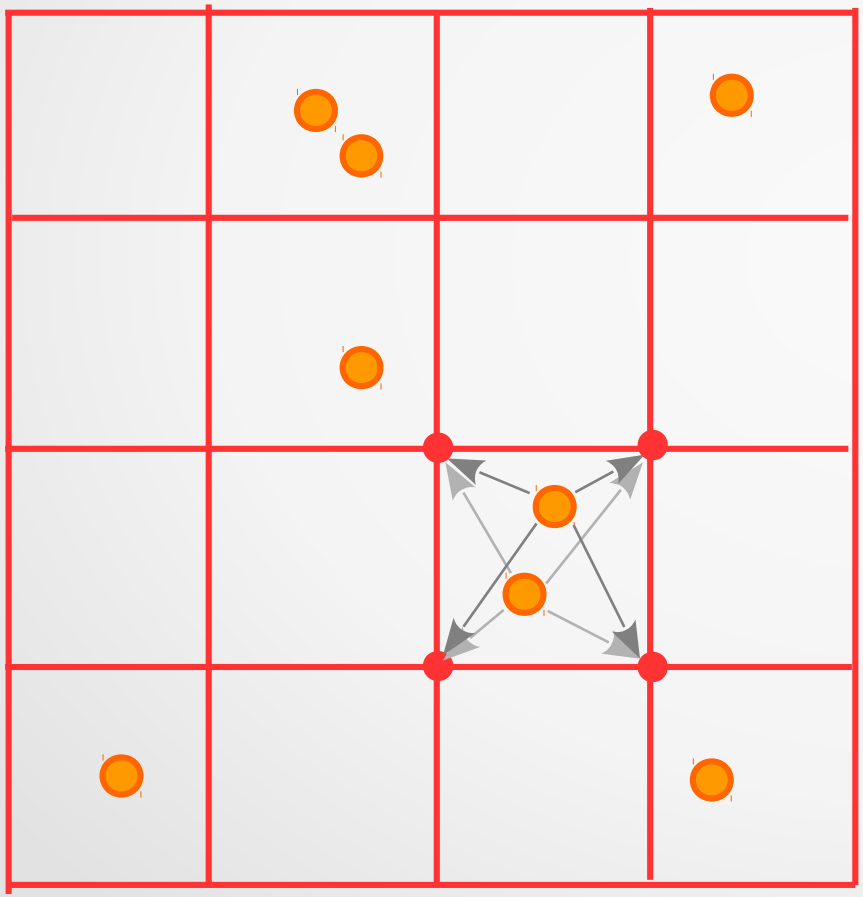
1	2	----------	N
x(1)	x(2)	----------	x(N)
y(1)	y(2)	----------	y(N)
mx(1)	mx(2)	----------	mx(N)

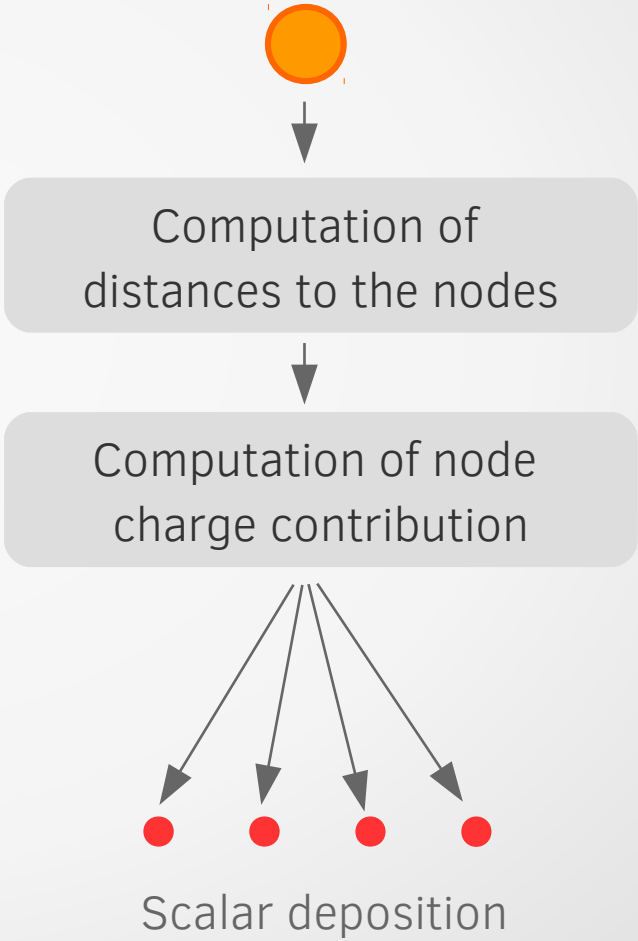
Particle structure:

```
Real(8), dimension(N) :: weight
Real(8), dimension(N) :: x, y
Real(8), dimension(N) :: mx, my
```

Level 3 – Vectorization of the charge deposition

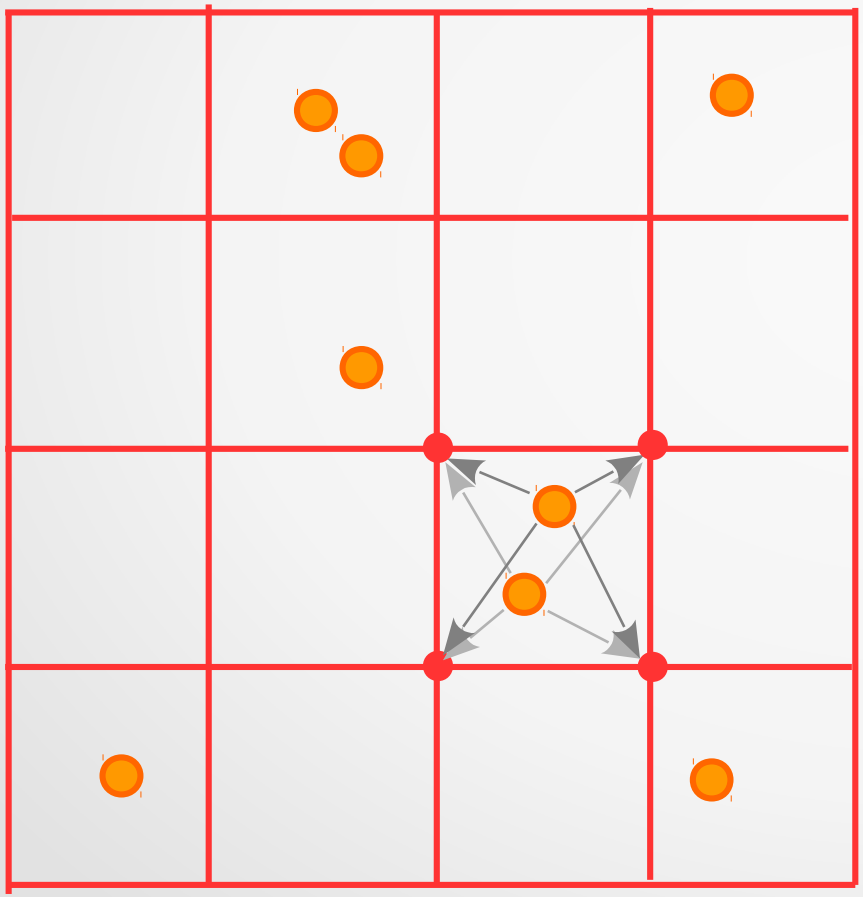
What prevents vectorization?

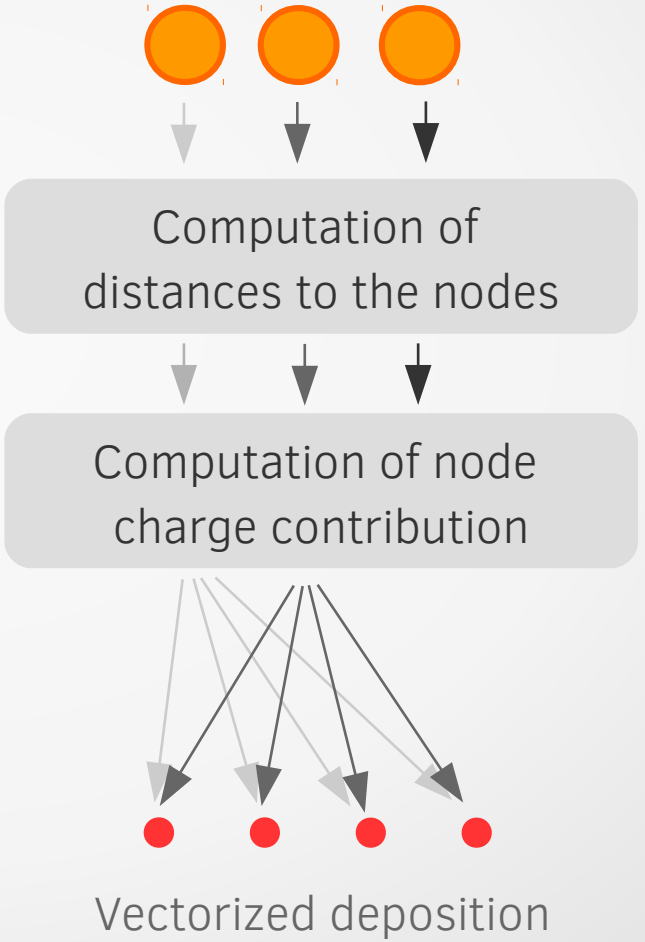




Level 3 – Vectorization of the charge deposition

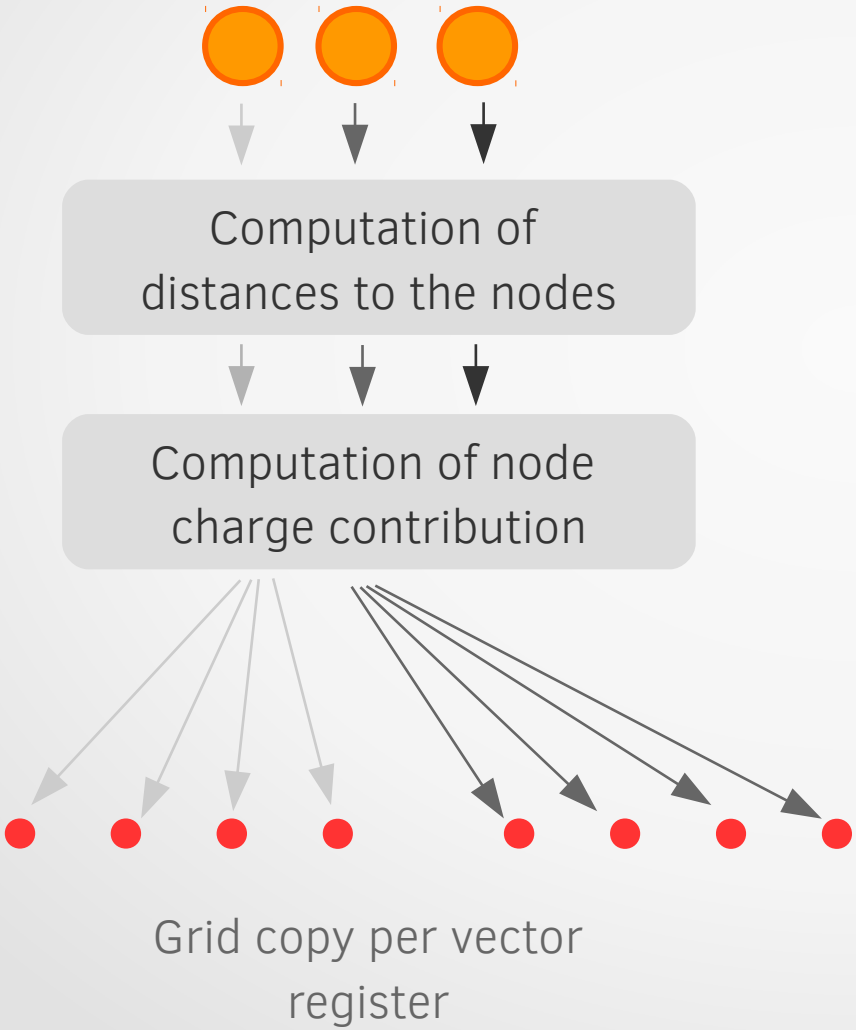
What prevents vectorization: Concurent and indirect memory accesses



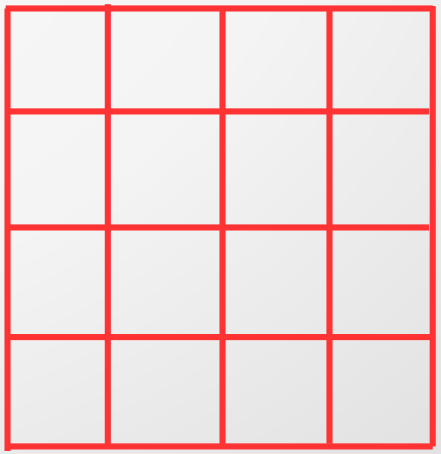


Level 3 – Vectorization of the charge deposition

How to solve this issue?

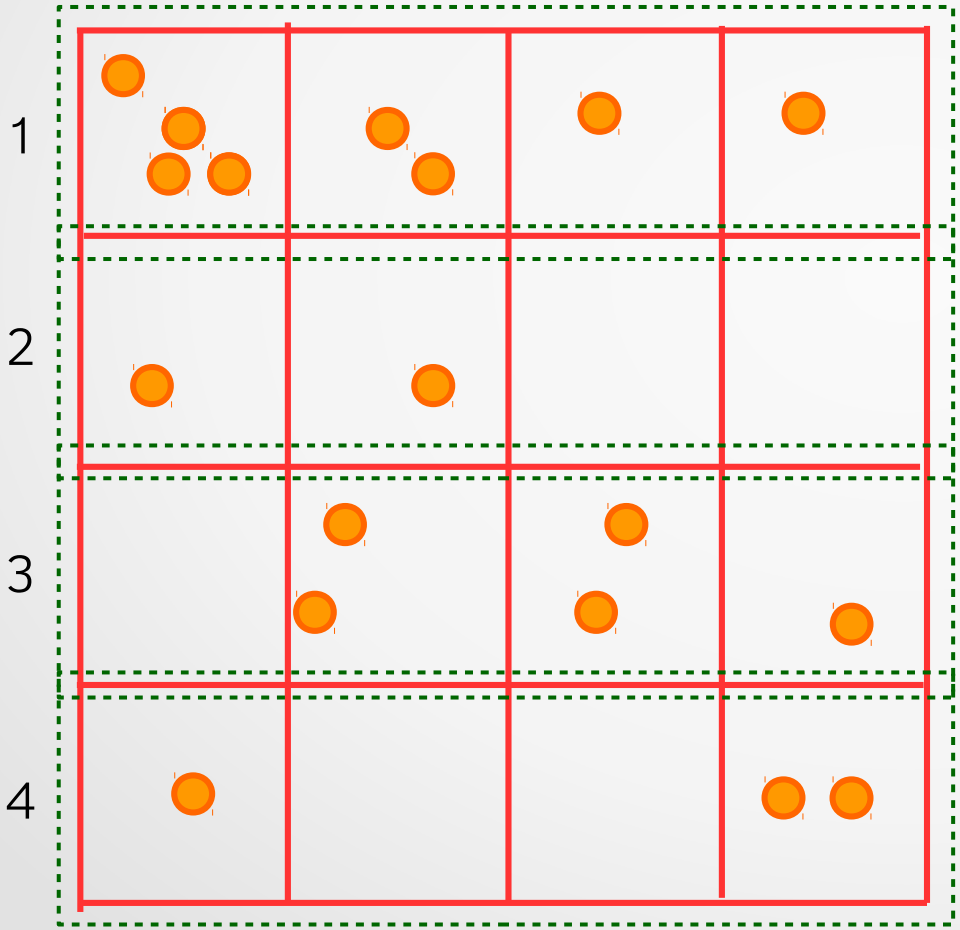


Reduction



Level 3 – Vectorization of the charge deposition

How to solve this issue?



- Particle decomposition into chunks localized in the same region
- Particle sorting
- Cache blocking effects