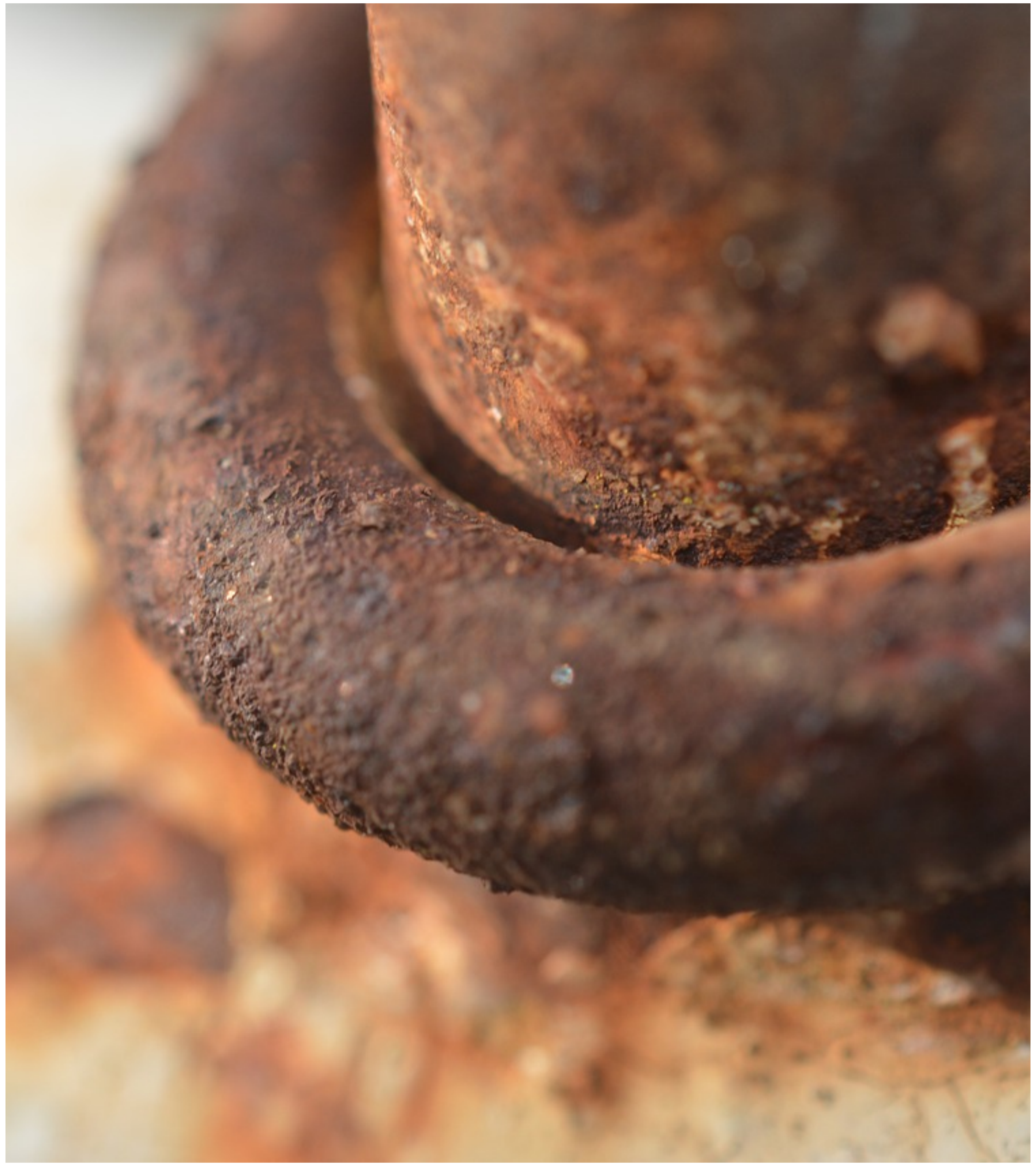
Particle in Cell Simulations

...with Rust and crustaceans

Hi

- Christoph Beberweil
- MSc in 2017
- Software Developer since 2017
- Part time PhD student since 2023 (~ 1m)
- Musician since 1997



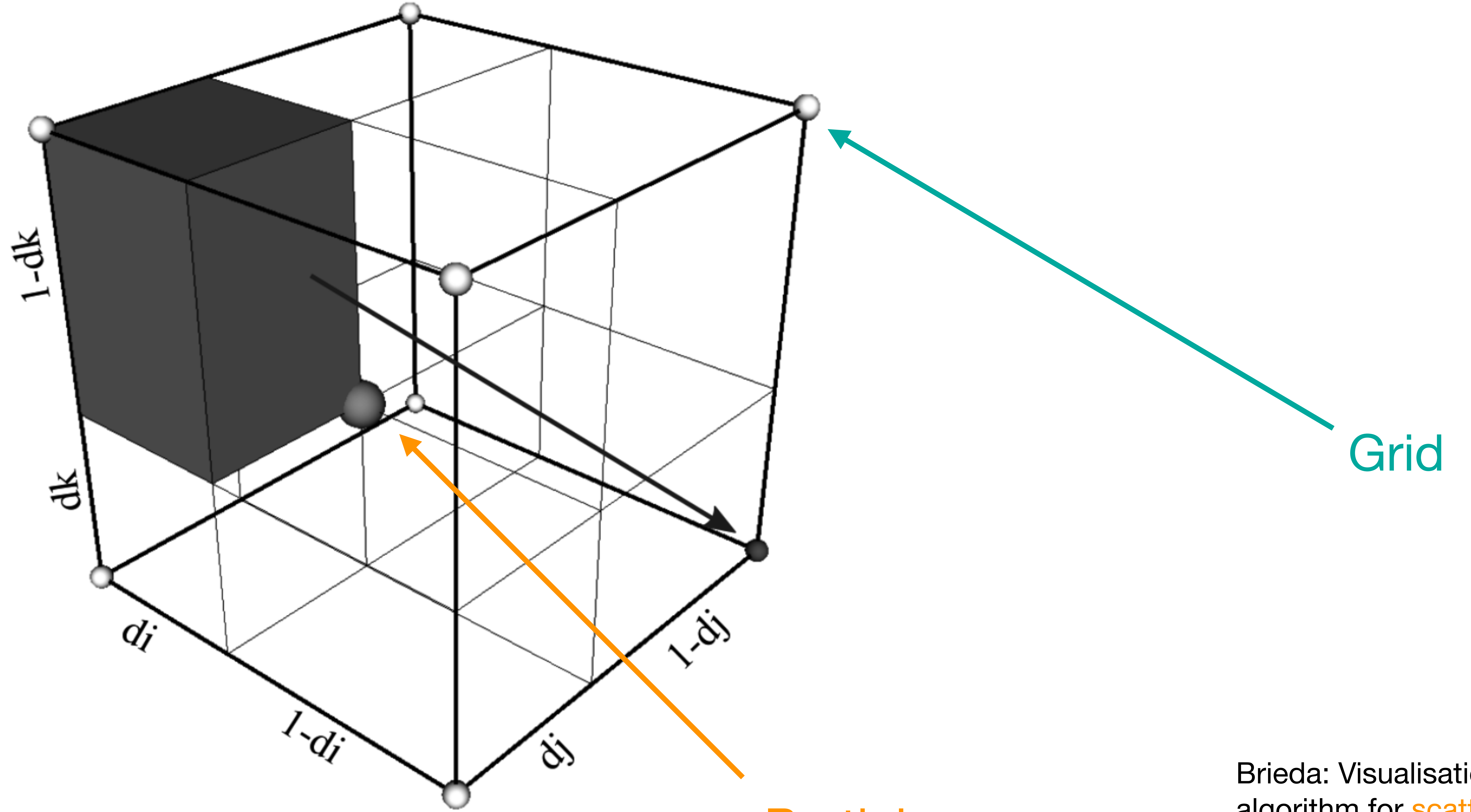
Particle in Cell simulations

1. Overlay the computational domain with a grid
2. Calculate charge density using Poission's equation on the **grid** points, scatter data from the particles
3. Calculate fields on the grid
4. Extrapolate (gather) the field data to the **particle** positions
5. Move particles
6. Rinse and repeat

Grid

- Large enough to resolve the geometry
- Small enough to fit in memory and to finish in acceptable time
- Resolve Debye length
- …

Gather

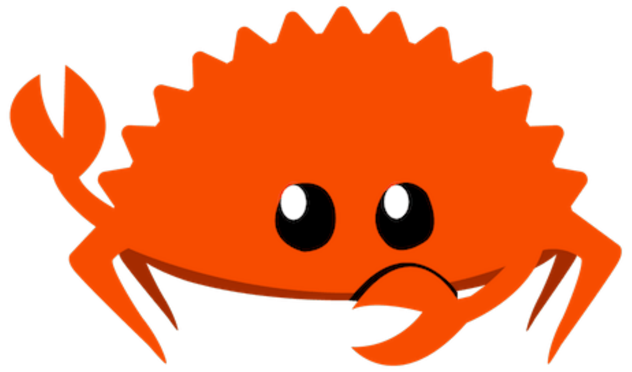


Brieda: Visualisation of a 3d interpolation algorithm for scattering particle properties to the grid or **gathering grid properties to particle positions**

Calculate Fields

$$\nabla^2 \phi = -\frac{\rho}{\epsilon_0}$$

Poisson's equation

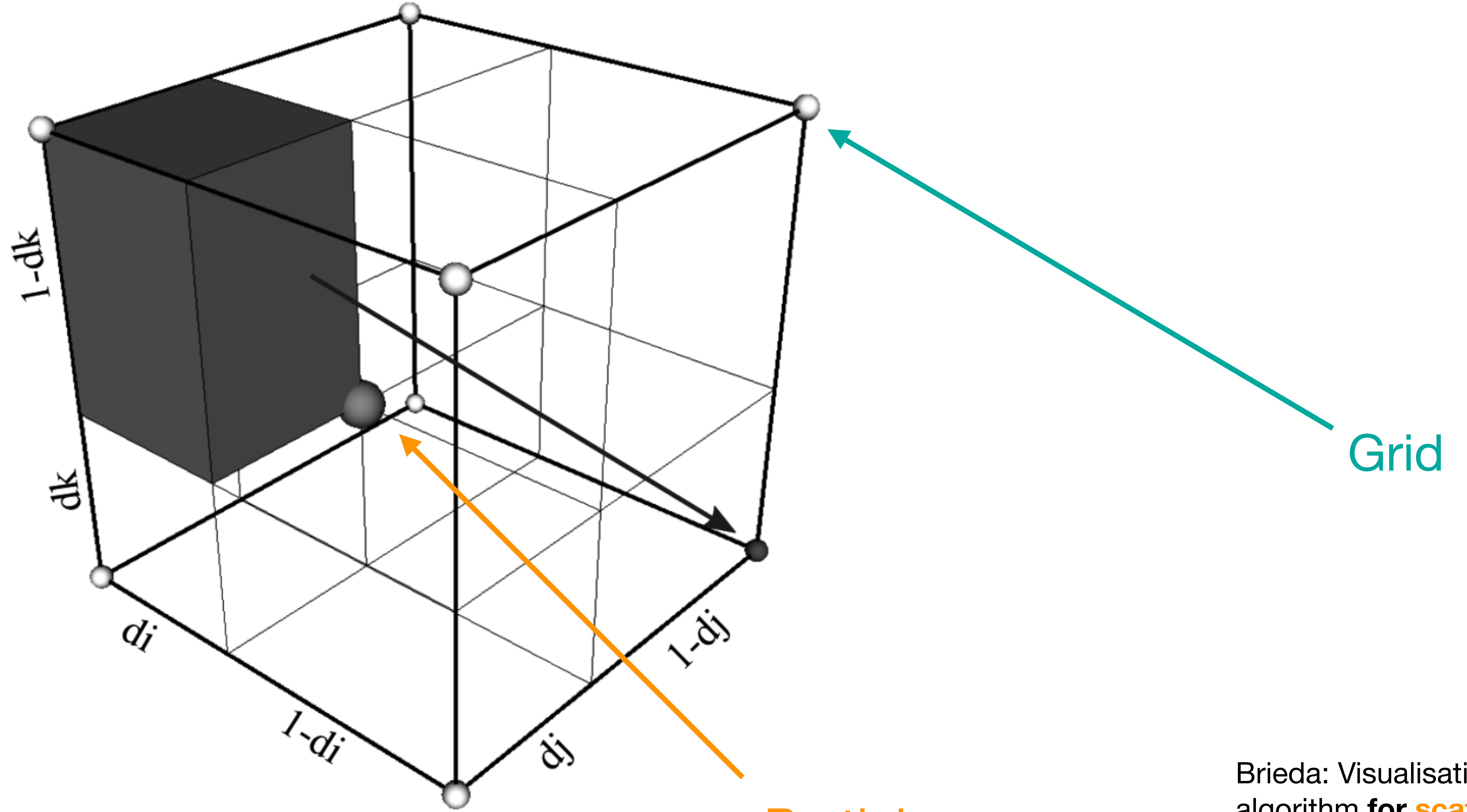


Finite Difference Method

Finite Element Method

Finite Volume Method

Scatter

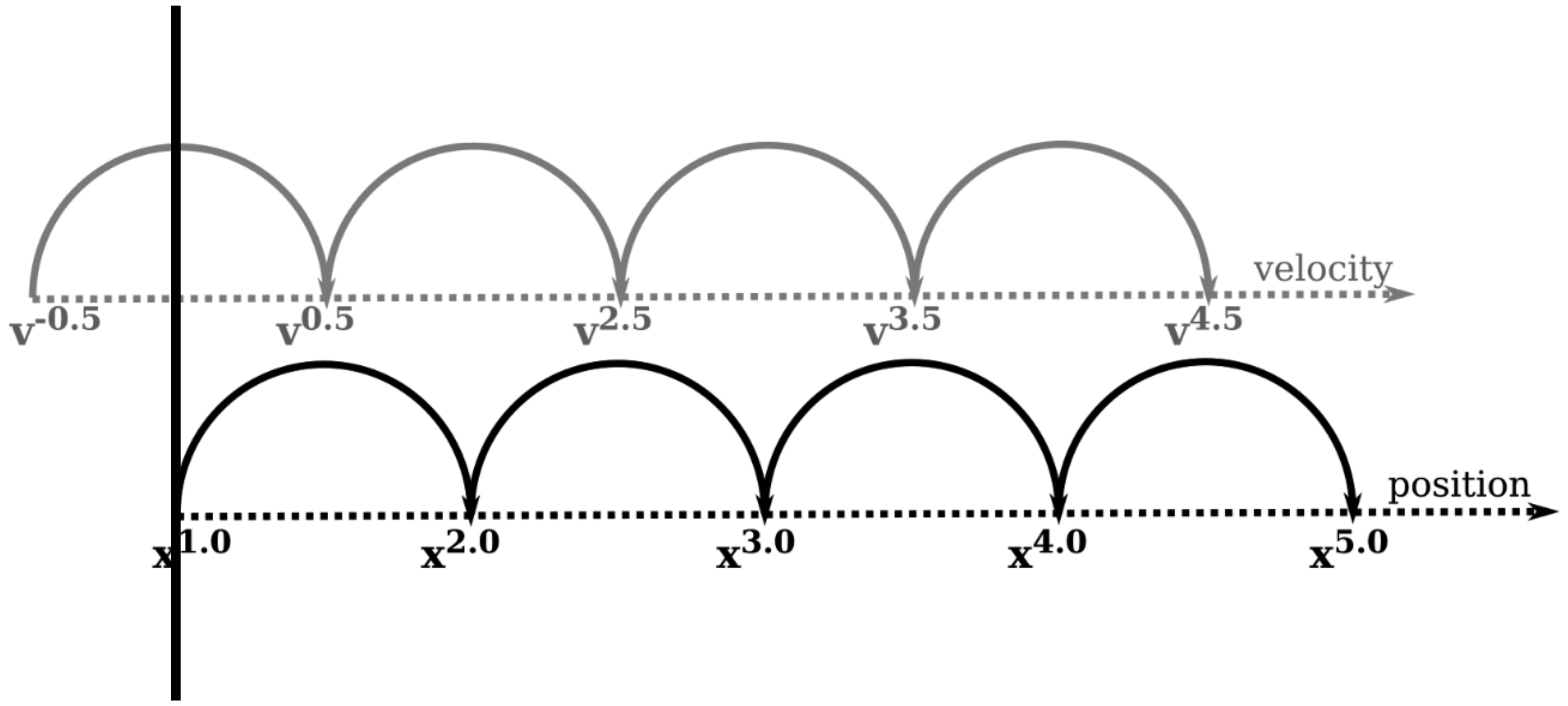


Brieda: Visualisation of a 3d interpolation algorithm **for scattering particle properties to the grid** or gathering grid properties to particle positions

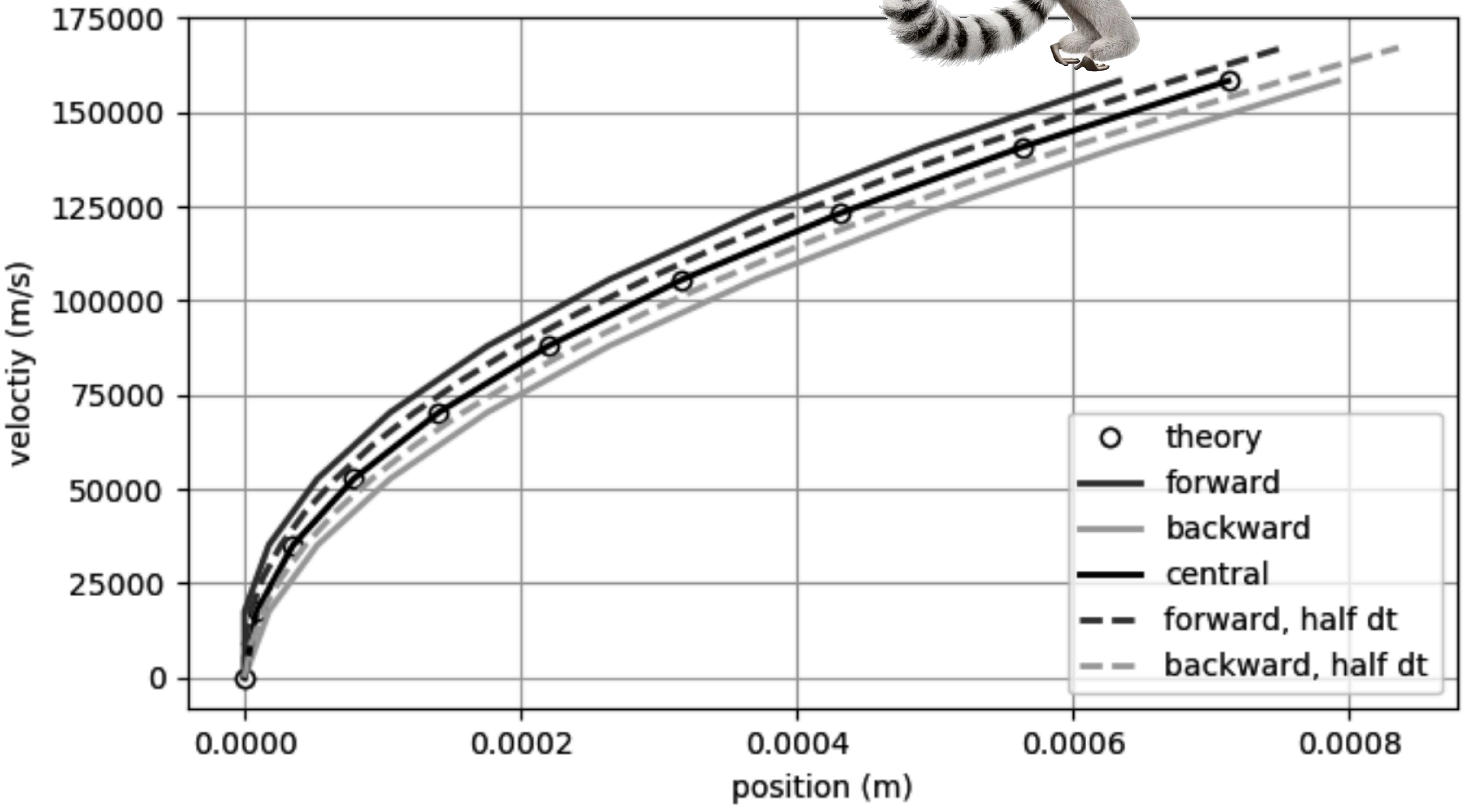
Move Particles

$$x^{k+1} = x^k + v^k \Delta t$$

$$v^{k+1} = v^k + (q/m)E^k \Delta t$$



Brieda: Leapfrog method



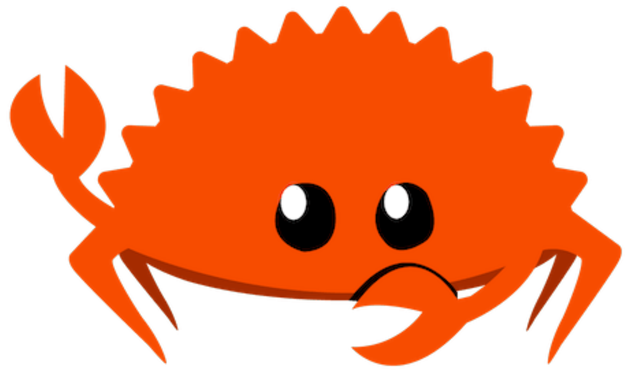
Brieda: Comparison of different particle pushing algorithms

Repeat

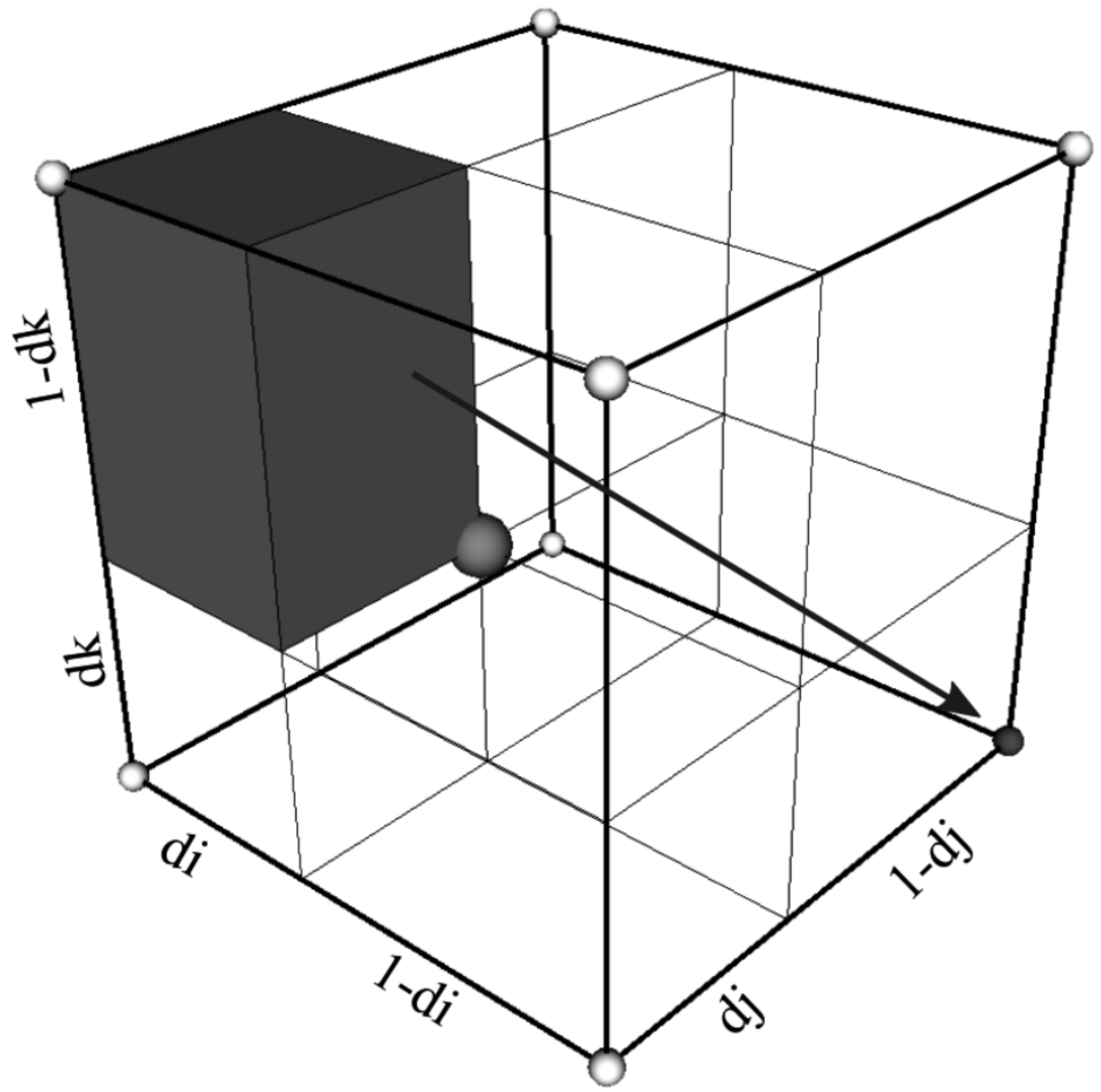
```
for i in 1..simulation.steps{
    do_simulation();
}
```

Goals

- Learn about modern plasma simulation methods and codes by writing a generic particle in cell simulation program
- Learn about Rust by writing it in Rust

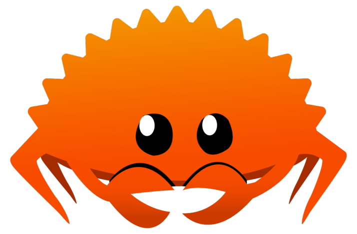


- Use the program to investigate ion beam transport and interactions in unusual and interesting geometries
    - Electron coolers
    - Gabor lenses
    - And your interesting application :)



Brieda: Visualisation of a 3d interpolation algorithm for scattering particle properties to the grid or gathering grid properties to particle positions

The Code

- Portable (run anywhere reasonable that is a rust compile target, probably not on microcontrollers…)
- Simple to use
- Blazingly fast 
- Parallel execution, horizontal scalability
- Define data aggregation and plots to be created during the simulation automatically
- Orchestrate simulations across multiple systems
- Web UI to show live simulation progress

… will run here one day ☝️

Challenges/Chances

- Physical correctness
- Performance
- Parallelisation #MooresLawIsDead

- Technical correctness (Option, Result, compiler assistance, build system)
- User Experience (Simple compilation, different input file formats, database backends, live view, scheduler across different servers and services with self service web frontend)
- Library: Different Frontends (Web, CLI, GUI, Mobile App, …)
- Parallelisation
- Community
- …

Performance Improvements

```rust
impl<T: Clone + Mul<Output = T>> Field<T> {
    pub fn scale_for(&mut self, factor: f64)
    where
        f64: Mul<T, Output = T>,
    {
        for i in 0..self.data.len() {
            for j in 0..self.data[i].len() {
                for k in 0..self.data[i][j].len() {
                    self.data[i][j][k] = factor * self.data[i][j][k].clone();
                }
            }
        }
    }
}
```

```rust
pub struct Field<T: Clone> {
    pub data: Vec<Vec<Vec<T>>>,
}
```

↓

```rust
impl<T: Clone + Mul<Output = T> + Debug> Field<T> {
    pub fn scale(&mut self, factor: f64)
    where
        f64: Mul<T, Output = T>,
    {
       self.data
            .iter_mut()
            .flatten()
            .flatten()
            .for_each(|k_d| *k_d = factor * k_d.clone());
    }
}
```

Performance Improvements 2

```rust
pub struct FieldFlat<T: Clone> {
    pub data: Vec<T>,
    pub dim: (usize, usize, usize),
}
```

```rust
impl<T: Clone + Mul<Output = T>> FieldFlat<T> {
    pub fn scale_for(&mut self, factor: f64)
    where
        f64: Mul<T, Output = T>,
    {
        for i in 0..self.data.len() {
           self.data[i] = factor * self.data[i].clone();
        }
    }
}
```

↓

```rust
impl<T: Clone + Mul<Output = T> + Debug> FieldFlat<T> {
    pub fn scale(&mut self, factor: f64)
    where
        f64: Mul<T, Output = T>,
    {
       self.data
            .iter_mut()
            .for_each(|i_d| *i_d = factor * i_d.clone());
    }
}
```

Parallelisation

```
//Standard: Took 0.347186s
data.into_iter()
    .map(|d| work_on_instance(d, config))
    .collect()
```

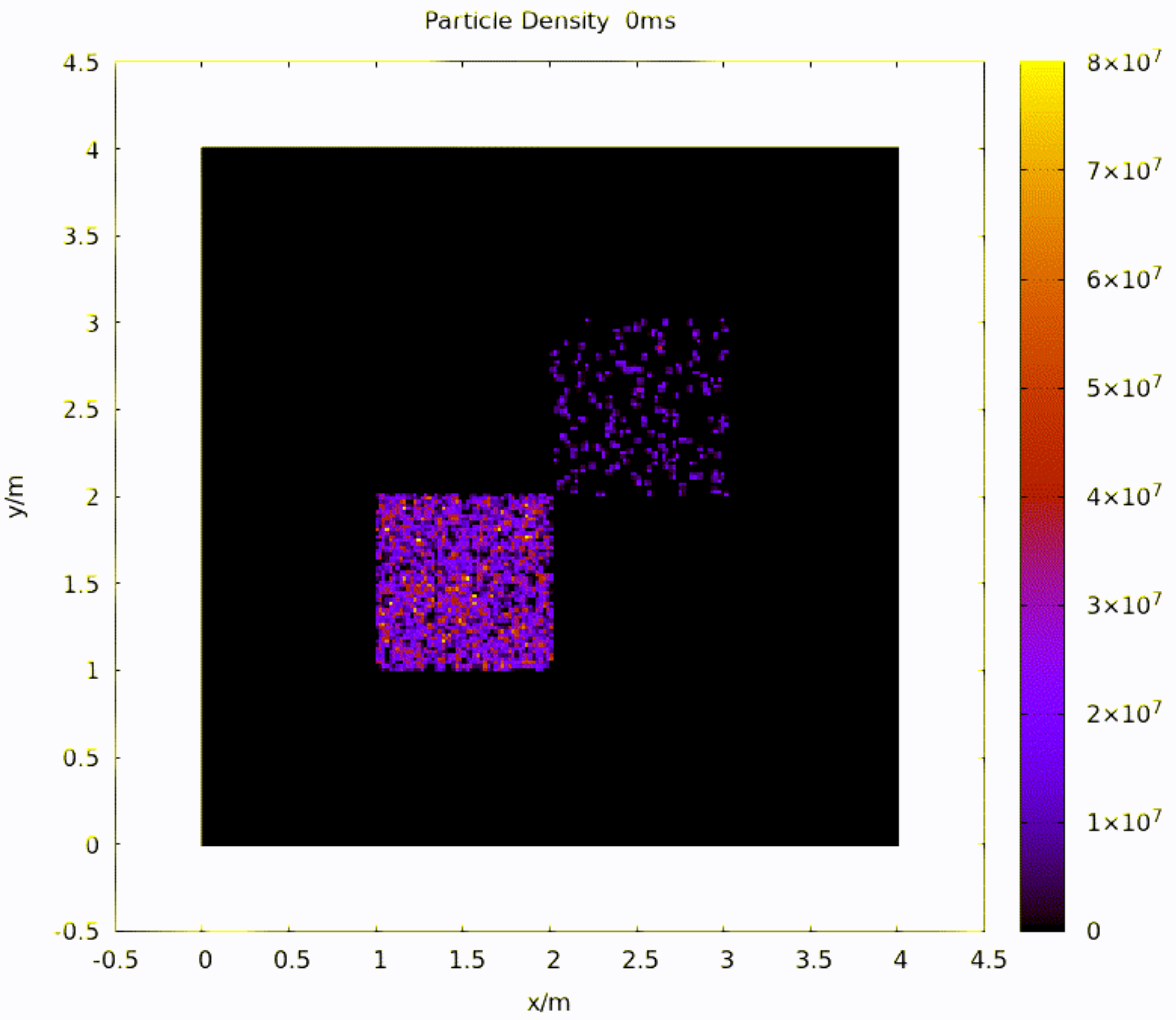
Works always

```
//Rayeon: Took 0.1634s
data.into_par_iter()
    .map(|d| work_on_instance(d, config))
    .collect()
```

Works only if d implements the IntoParallelIterator trait

OS Threads

Tokio


Particle Density 0ms
y/m
x/m
4.5
4
3.5
3
2.5
2
1.5
1
0.5
0
-0.5
-0.5
0
0.5
1
1.5
2
2.5
3
3.5
4
4.5
8×10^7
7×10^7
6×10^7
5×10^7
4×10^7
3×10^7
2×10^7
1×10^7
0

Infrastructure

- Homepage https://nnp.physik.uni-frankfurt.de
- Published Talks https://talks.nnp.physik.uni-frankfurt.de
- Internal Wiki https://wiki.nnp.physik.uni-frankfurt.de
- Git Hosting https://git.nnp.physik.uni-frankfurt.de