Accelerating a Spectral Algorithm for Plasma Physics with Python/Numba on GPU

FBPIC: A spectral, quasi-3D, GPU accelerated Particle-In-Cell code

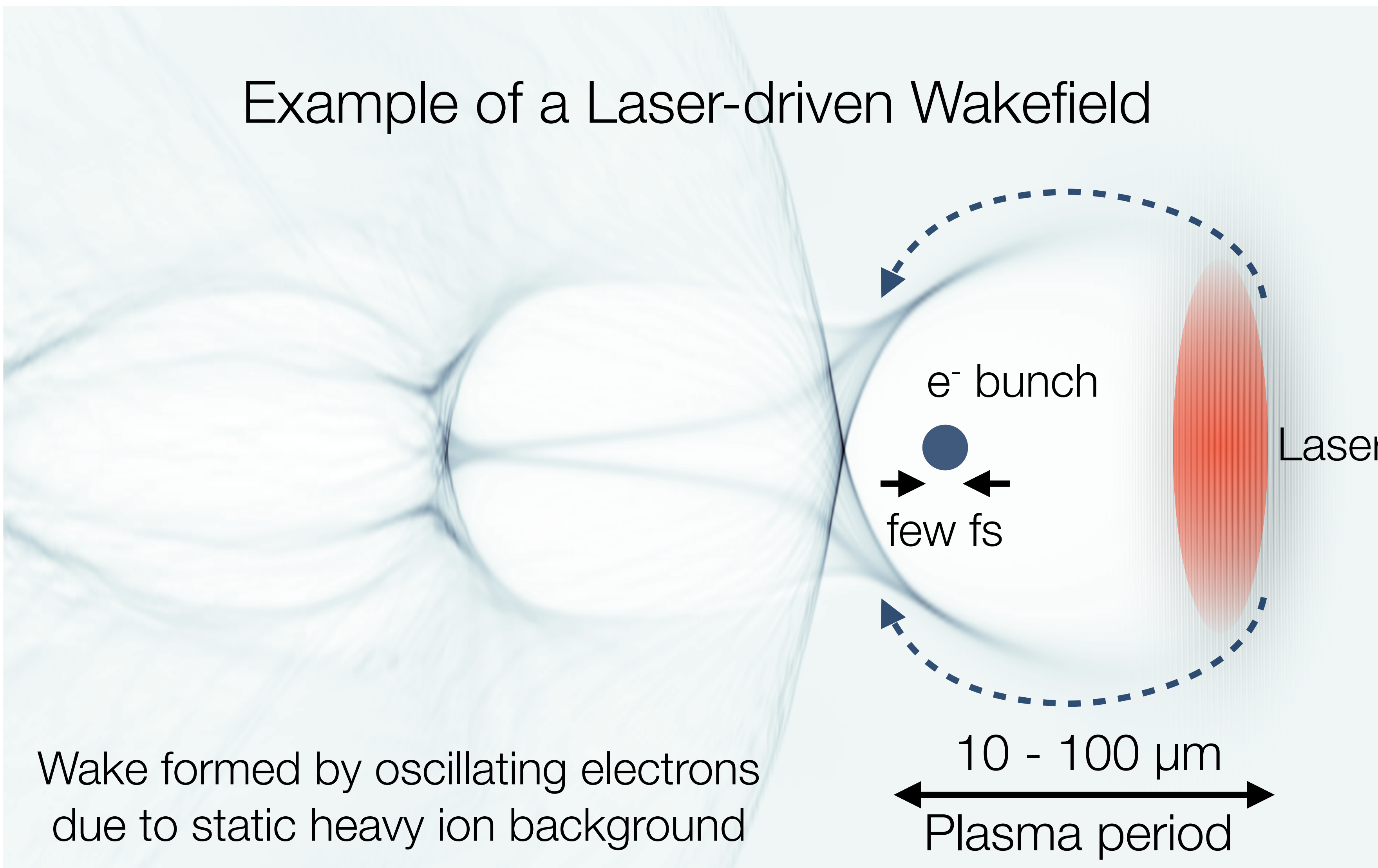
Manuel Kirchen
Center for Free-Electron Laser Science
University of Hamburg, Germany
manuel.kirchen@desy.de

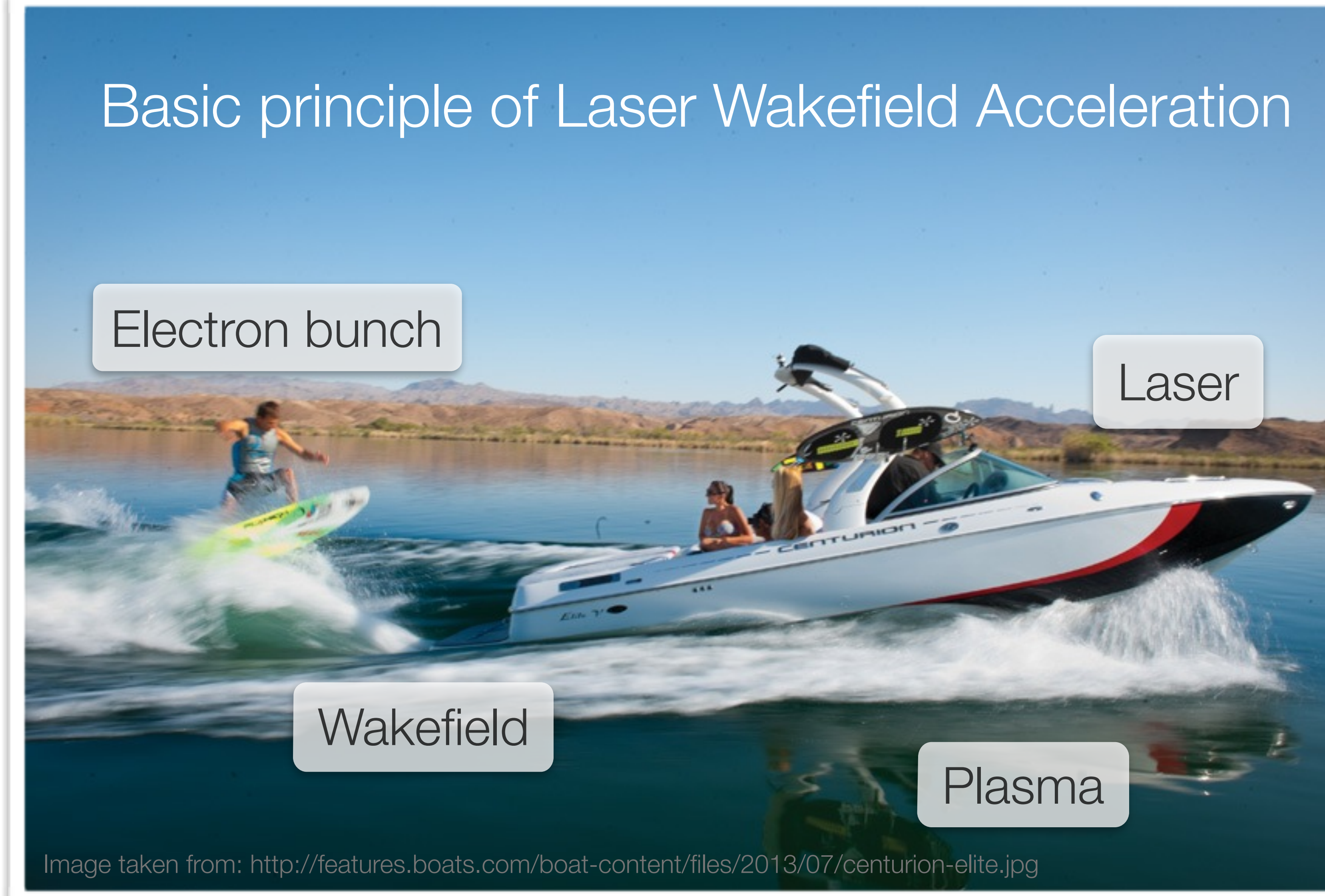
Rémi Lehe
BELLA Center &
Center for Beam Physics,
LBNL, USA
rlehe@lbl.gov

Content

- Introduction to Plasma Accelerators
- Modelling Plasma Physics with Particle-In-Cell Simulations
- A Spectral, Quasi-3D PIC Code (FBPIC)
- Two-Level Parallelization Concept
- GPU Acceleration with Numba
- Implementation & Performance
- Summary

Introduction to Plasma Accelerators

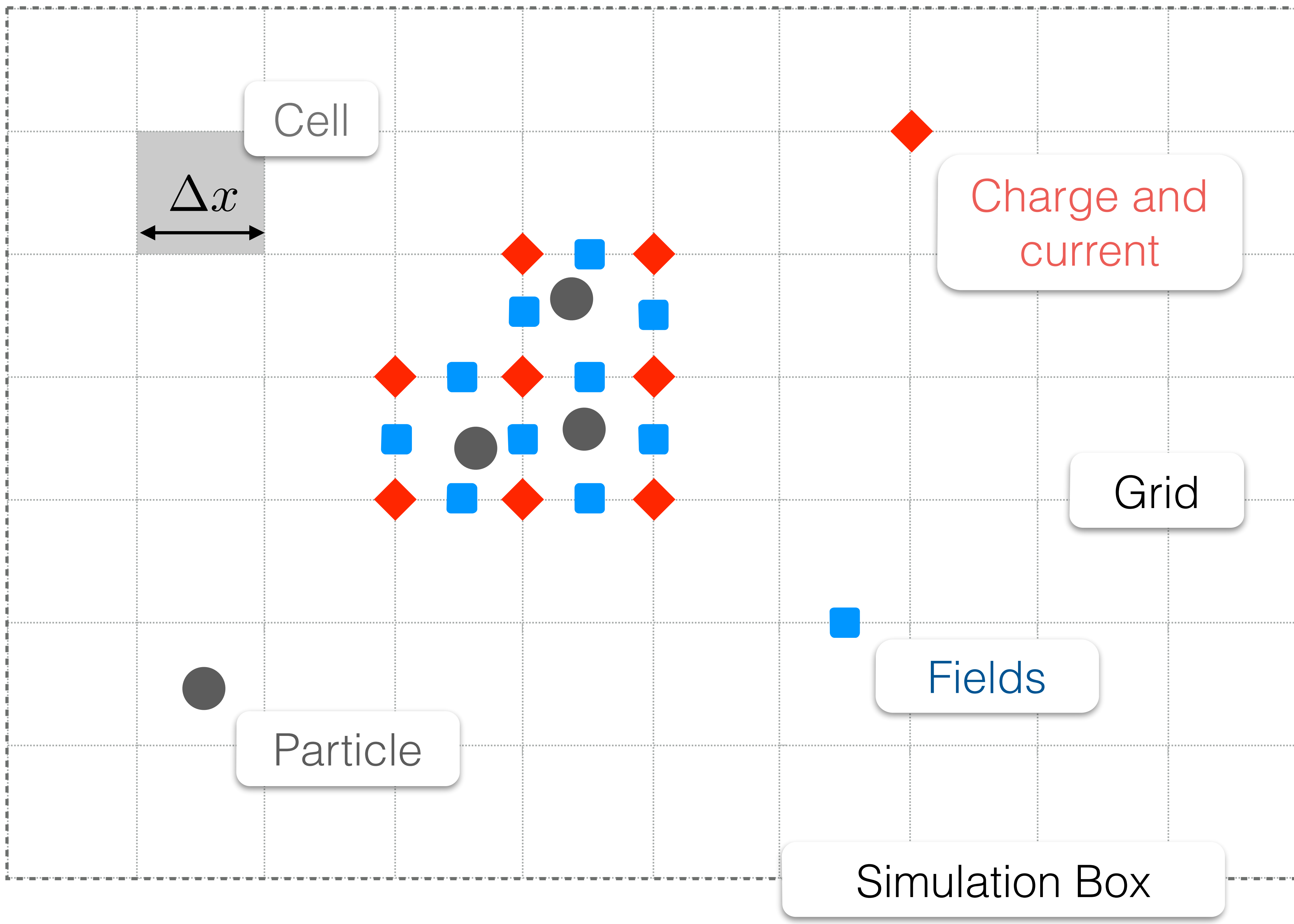




- cm-scale plasma target (ionized gas)
- Laser pulse or electron beam drives the wake
- Length scale of accelerating structure: Plasma wavelength (µm scale)
- Charge separation induces strong electric fields (~100 GV/m)

Shrink accelerating distance from km to mm scale (orders of magnitude)
+ Ultra-short timescales (few fs)

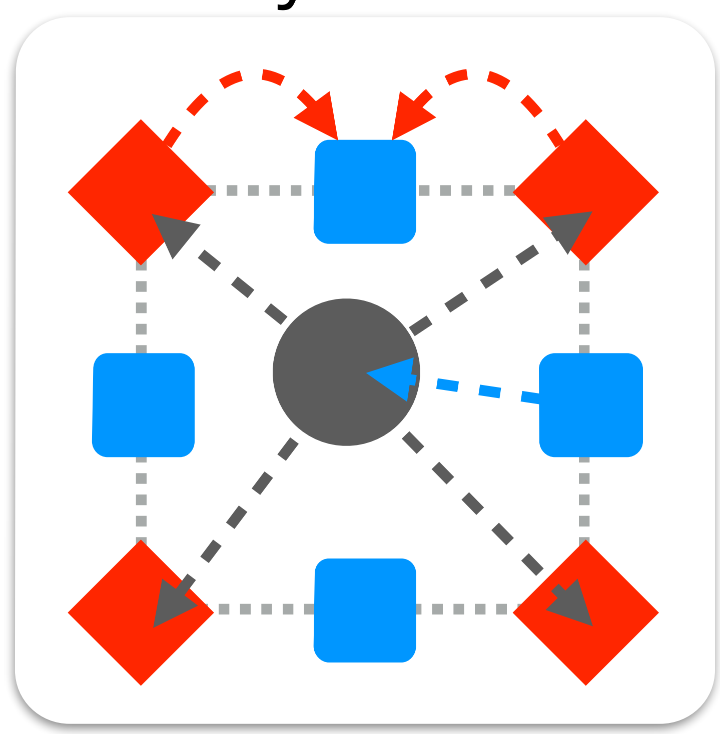
Modelling Plasma Physics with Particle-In-Cell Simulations



Millions of cells, particles and iterations!

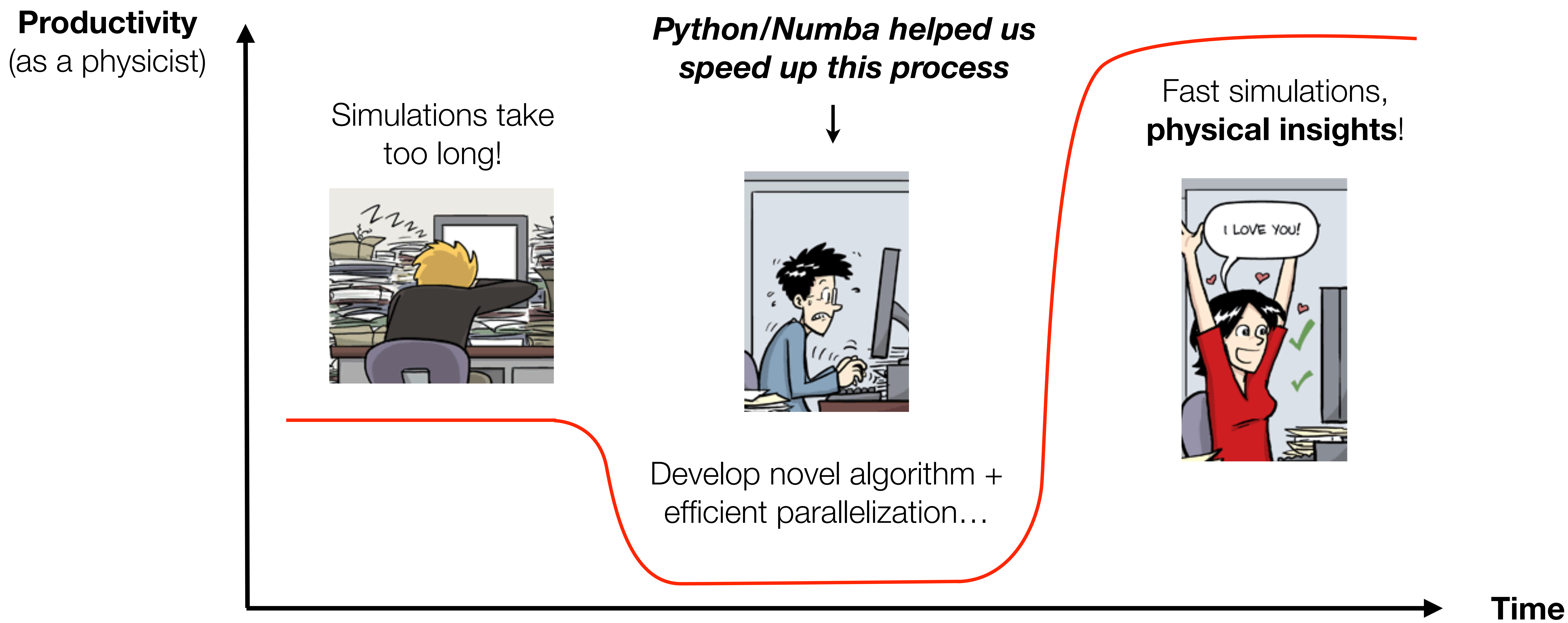
- Fields on discrete grid
- Macroparticles interact with fields

PIC Cycle



- Charge/Current deposition on grid nodes
- Fields are calculated ➔ Maxwell equations
- Fields are gathered onto particles
- Particles are pushed ➔ Lorentz equation

Productivity of a (Computational) Physicist



Our goal: Reasonably fast & accurate code with many features and user-friendly interface

A Spectral, Quasi-3D PIC Code

PIC Simulations in **3D** are essential, but **computationally demanding**

Majority of algorithms are based on **finite-difference algorithms** that introduce numerical artefacts

Quasi-cylindrical symmetry

- **Captures important 3D effects**
  (Lifschitz et al., 2009)
- **Computational cost similar to 2D code**

Spectral solvers

- **Correct evolution of electromagnetic waves**
  PSATD algorithm *(Haber et al., 1973)*
- **Less numerical artefacts**

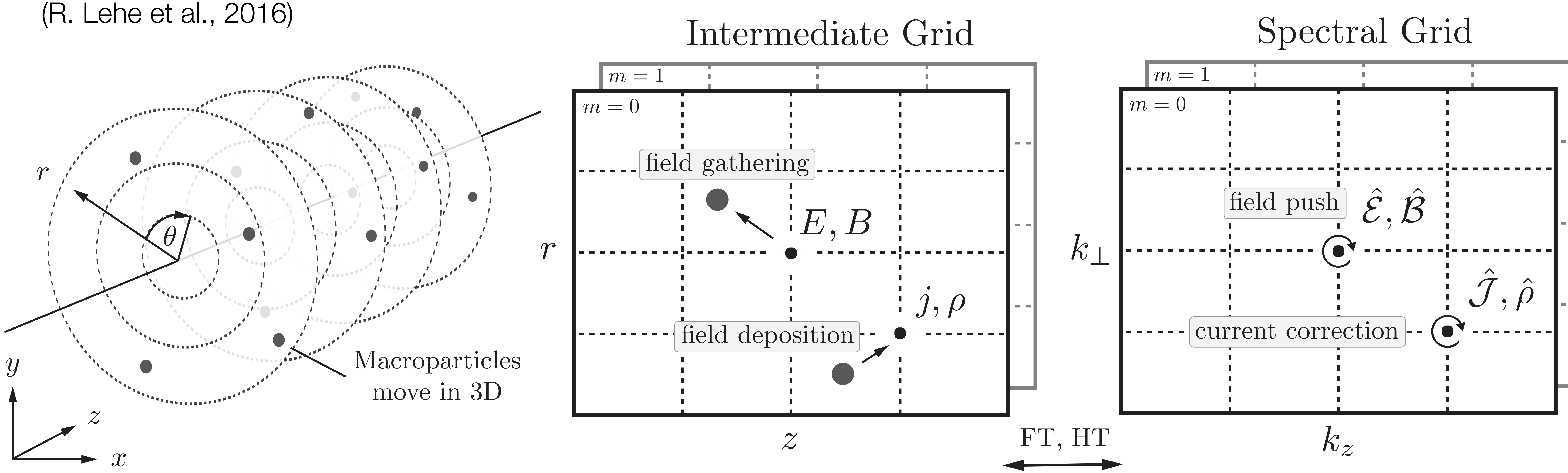
Combine best of both worlds → ***Spectral & quasi-cylindrical algorithm***

A Spectral, Quasi-3D PIC Code

Algorithm developed by **Rémi Lehe**

FBPIC (Fourier-Bessel Particle-In-Cell)

(R. Lehe et al., 2016)

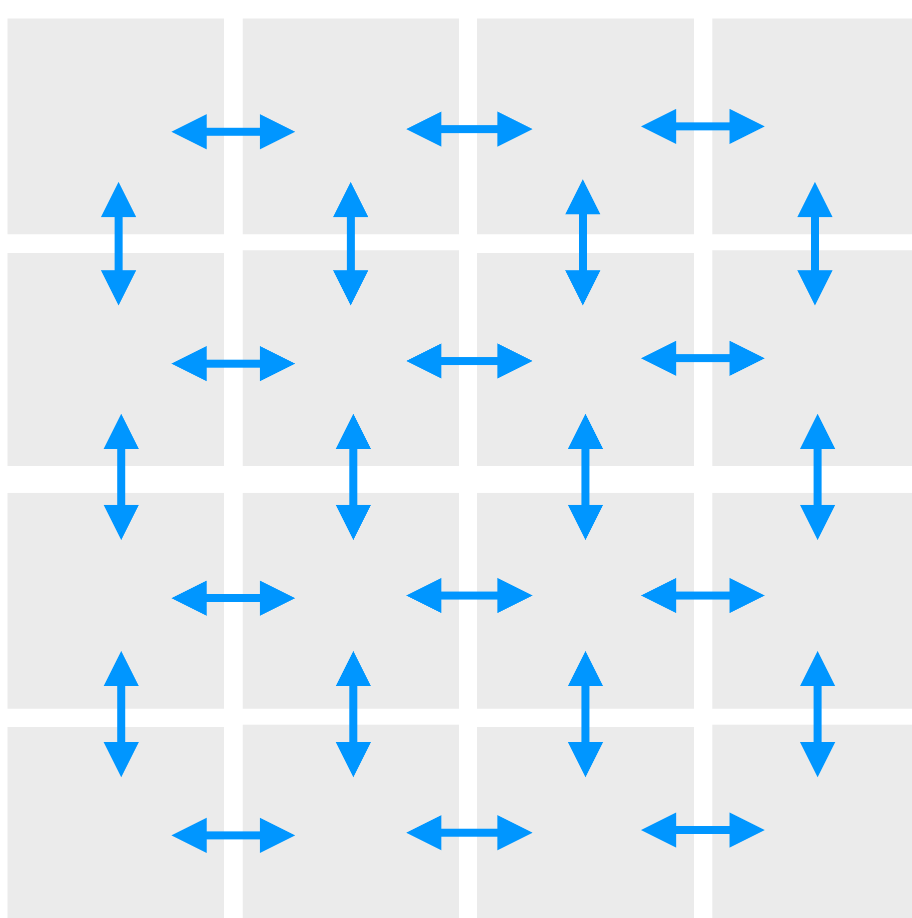


- Written entirely in **Python and uses Numba** Just-In-Time compilation
- Only **single-core** and **not easy to parallelize** due to **global operations** (FFT and DHT)

Parallelization Approach for Spectral PIC Algorithms

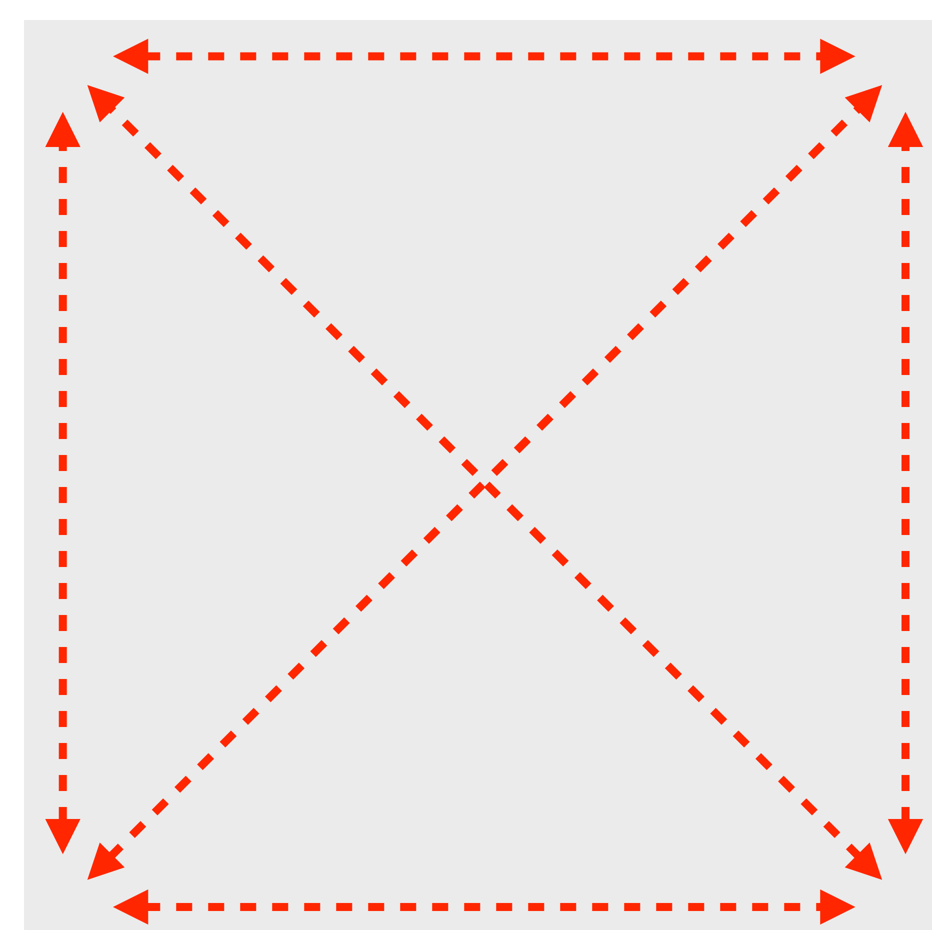
Not easy to parallelize by domain decomposition, due to FFT & DHT.

Standard (FDTD)
Domain Decomposition



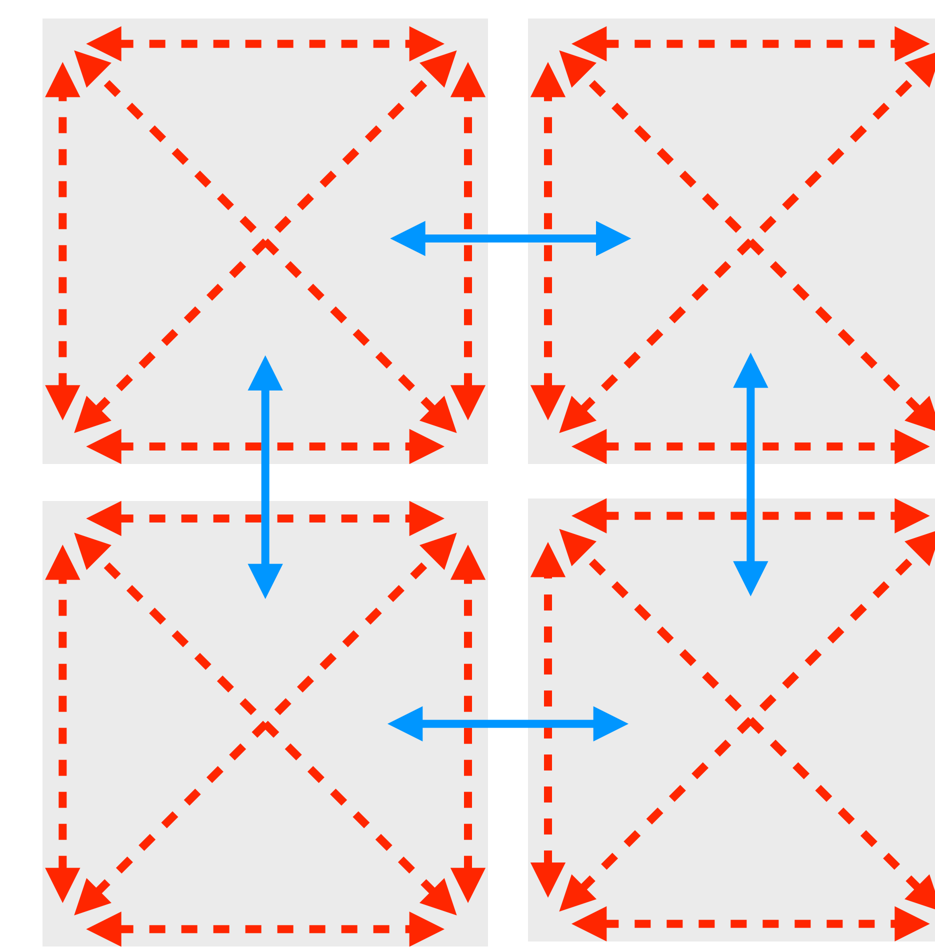
local exchange
low accuracy

Spectral
Transformations



global communication
high accuracy

Local Transformations &
Domain Decomposition



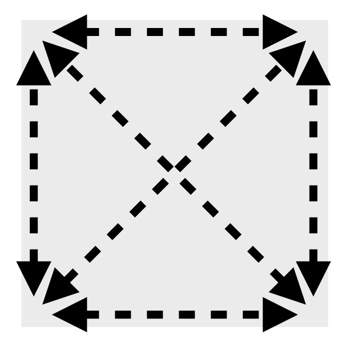
local communication & exchange
arbitrary accuracy

Local parallelization of global operations & global domain decomposition

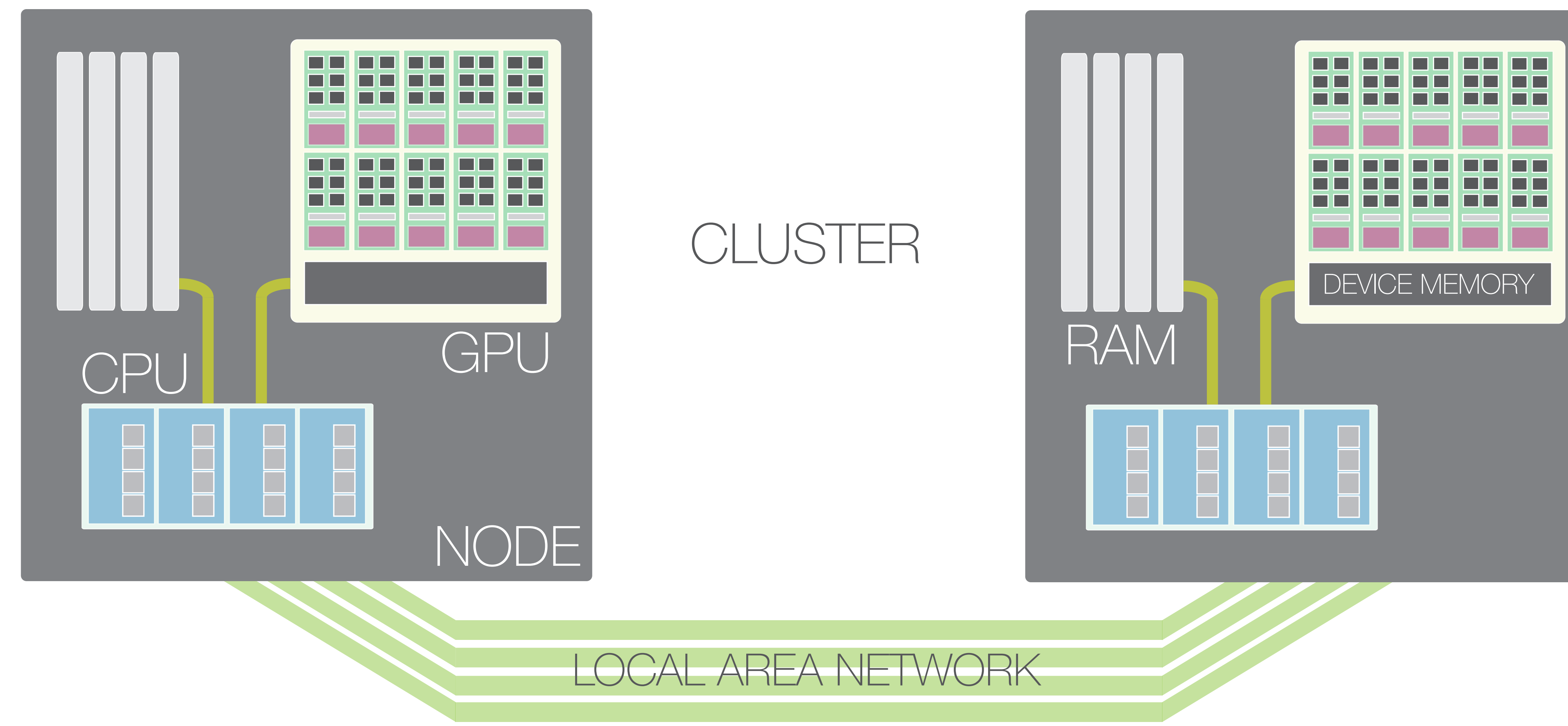
Parallelization Concept

Intra-node parallelization

- Shared memory layout
- GPU (or multi-core CPU)
- *Parallel PIC methods & Transformations*
- ***Numba + CUDA***

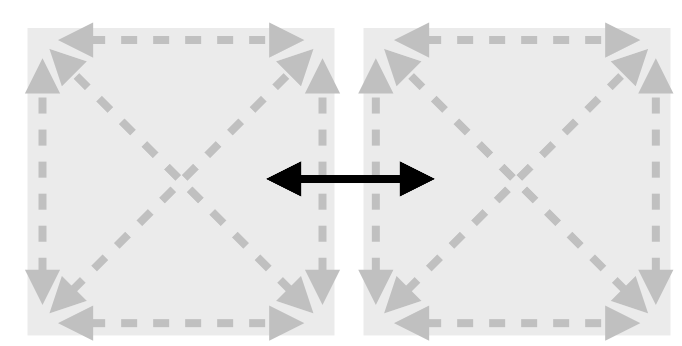


Typical HPC infrastructure



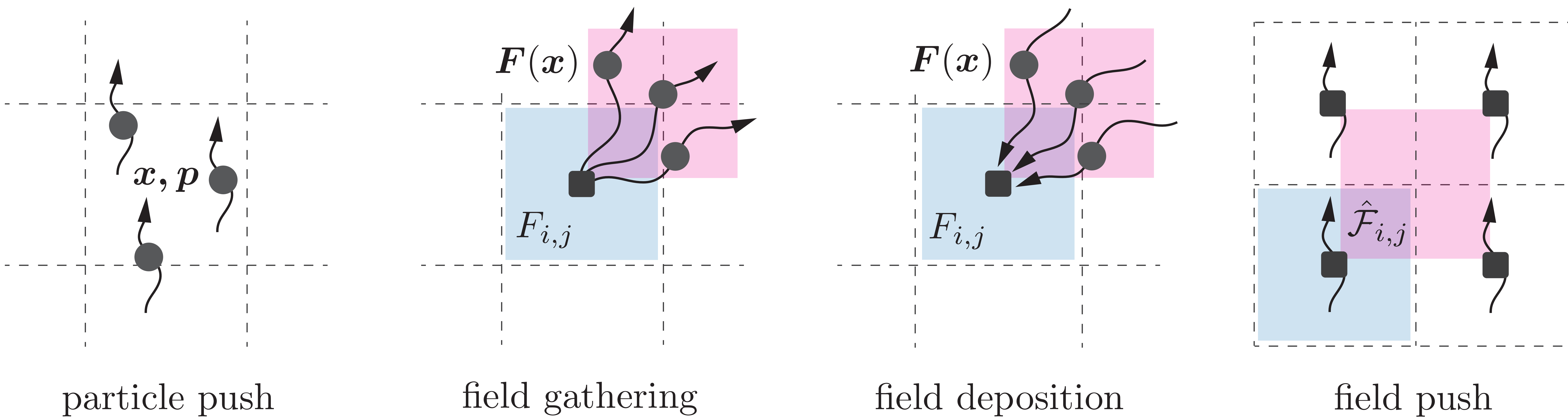
Inter-node parallelization

- Distributed memory layout
- Multi-CPU / Multi-GPU
- *Spatial domain decomposition for spectral codes (Vay et al., 2013)*
- ***mpi4py***



Shared and distributed memory layouts → ***Two-level parallelization entirely with Python***

Intra-Node Parallelization of PIC Methods



Particles

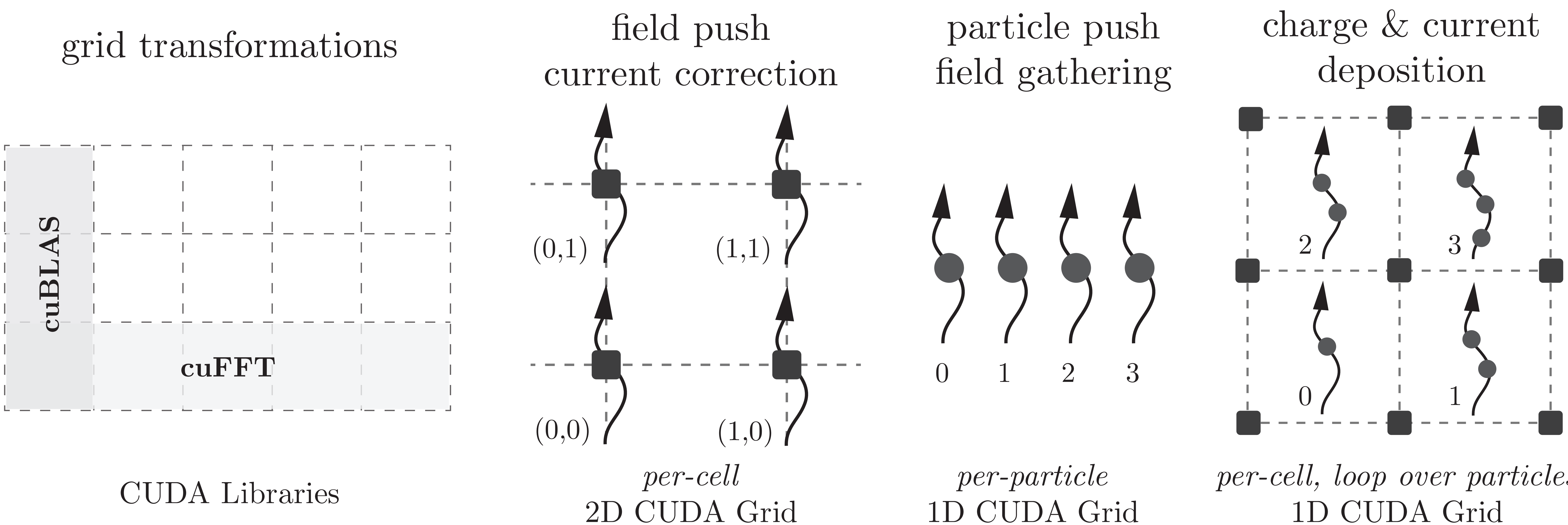
- Particle push: Each thread updates one particle
- Field gathering: Some threads read same field value
- Field deposition: Some threads **write** same field value
  → race conditions!

Fields

- Field push and current correction: Each thread updates one grid value
- Transformations: Use optimized parallel algorithms

Intra-node parallelization → ***CUDA with Numba***

CUDA Implementation with Numba



Fields

- Transformation ➞ CUDA Libraries
- Field push & current correction per-cell

Particles

- Field gathering and particle push per-particle
- Field deposition ➞ Particles are sorted and each thread loops over particles in its cell

CUDA Implementation with Numba

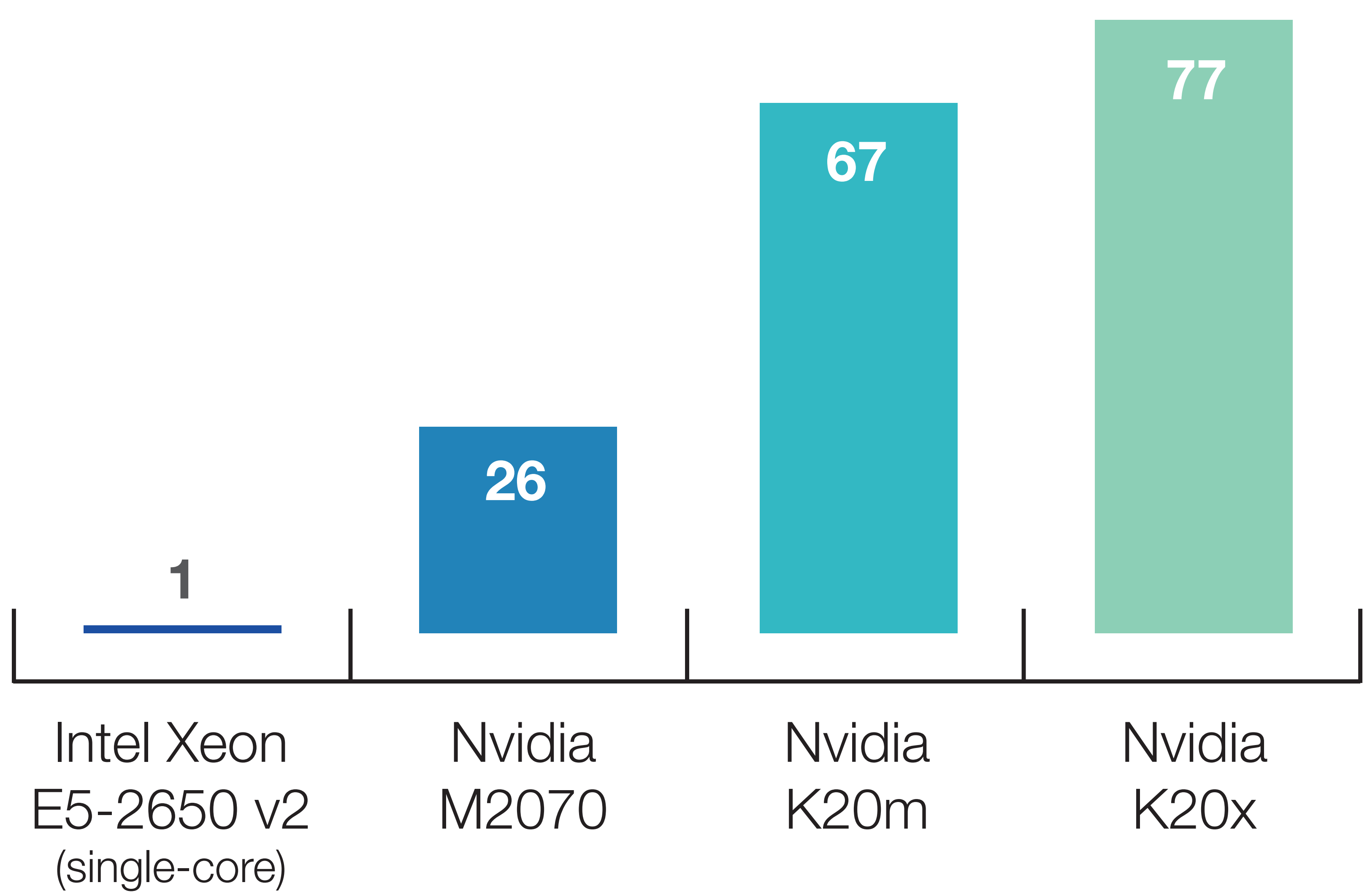
- Simple interface for writing CUDA kernels
- Made use of cuBLAS, cuFFT, RadixSort
- Manual Memory Management
  Data is kept on GPU / only copied to CPU for I/O
- Almost full control over CUDA API
- **Ported code to GPU in less than 3 weeks**

```python
@cuda.jit('void(float64[:], float64[:], float64[:], \
            float64[:], float64[:], float64[:], \
            float64[:], float64)')
def push_x_gpu( x, y, z, ux, uy, uz, inv_gamma, dt ) :
    """
    Advance the particles' positions over one half-timestep

    This assumes that the positions (x, y, z) are initially either
    one half-timestep *behind* the momenta (ux, uy, uz), or at the
    same timestep as the momenta.

    Parameters
    ----------
    x, y, z : 1darray of floats (in meters)
        The position of the particles
        (is modified by this function)

    ux, uy, uz : 1darray of floats (in meters * second^-1)
        The velocity of the particles

    inv_gamma : 1darray of floats
        The inverse of the relativistic gamma factor

    dt : float (seconds)
        The time by which the position is advanced
    """
    # Half timestep, multiplied by c
    chdt = c*0.5*dt

    i = cuda.grid(1)
    if i < x.shape[0]:
        # Particle push
        inv_g = inv_gamma[i]
        x[i] += chdt*inv_g*ux[i]
        y[i] += chdt*inv_g*uy[i]
        z[i] += chdt*inv_g*uz[i]
```

Simple CUDA kernel in FBPIC

Single-GPU Performance Results

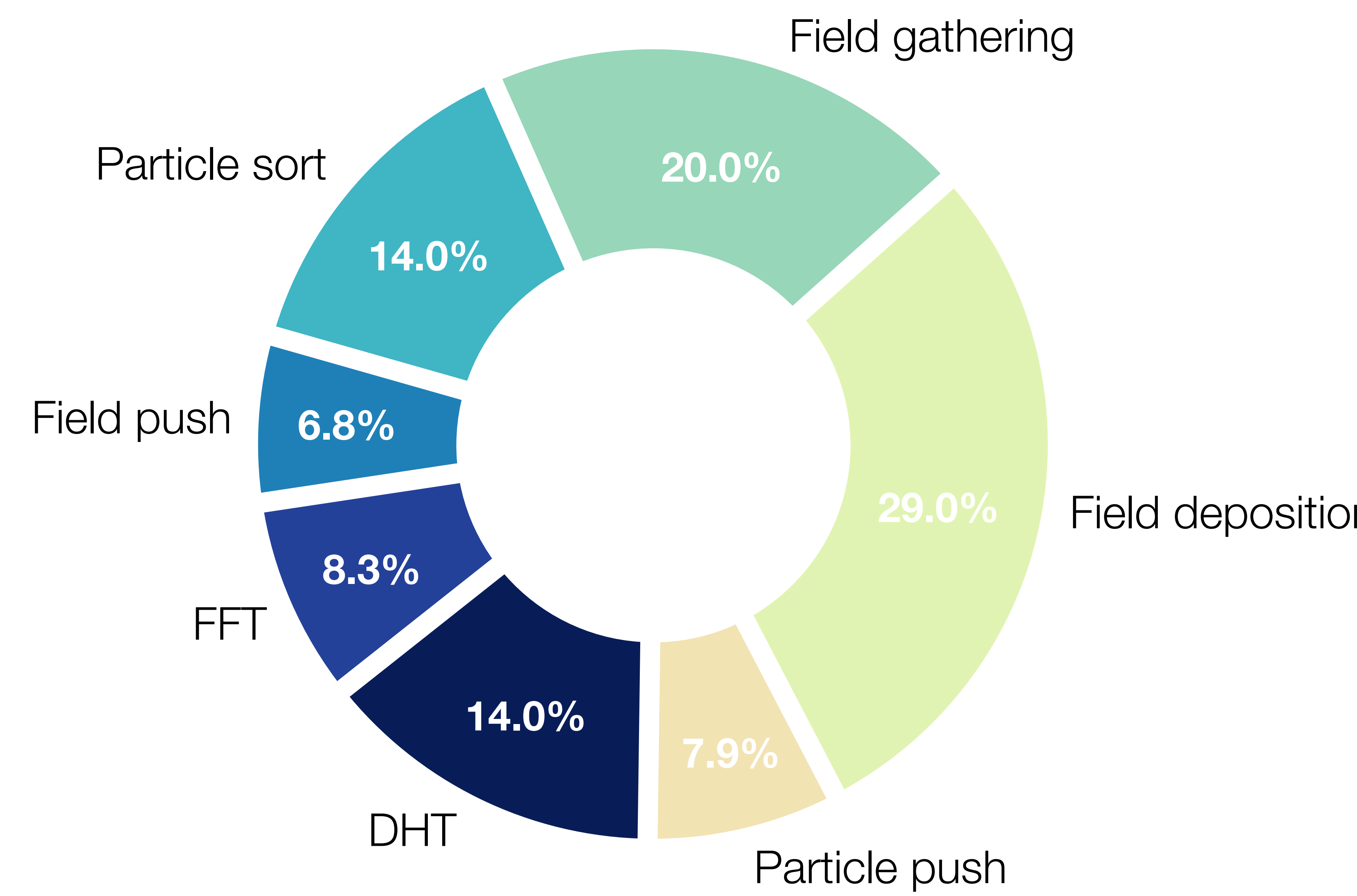
Speed-up on different Nvidia GPUs



Speed-up of up to ~70

compared to single-core CPU version

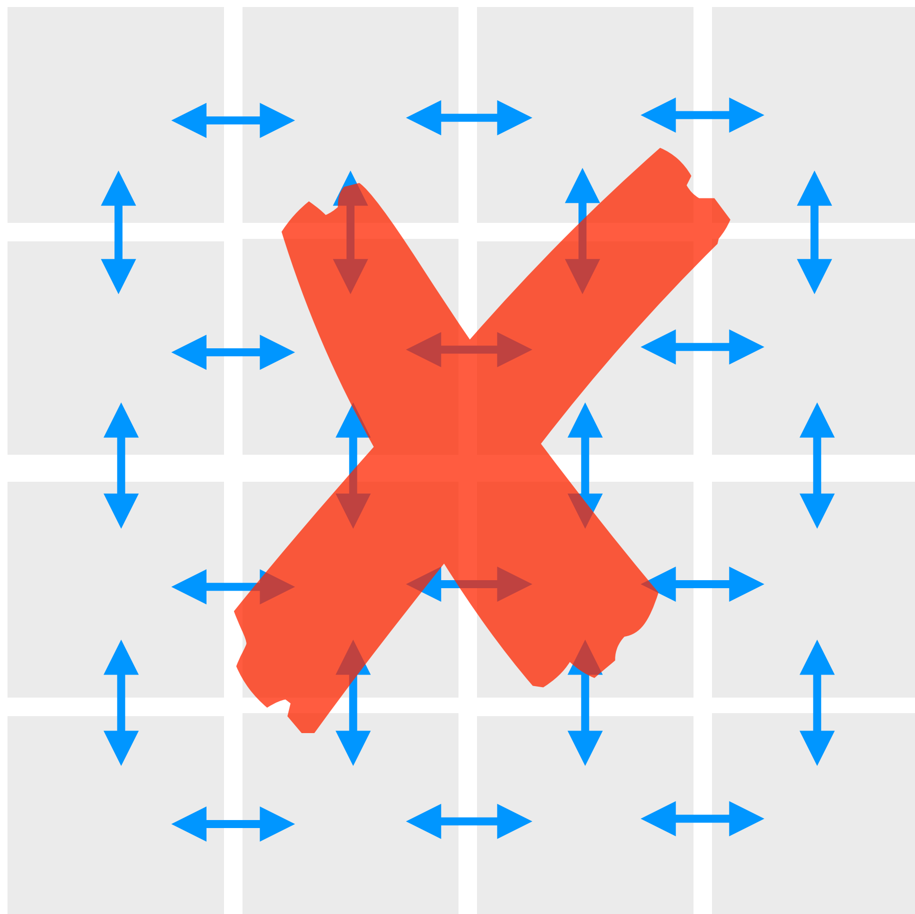
Runtime distribution of the GPU PIC methods



20 ns per particle per step

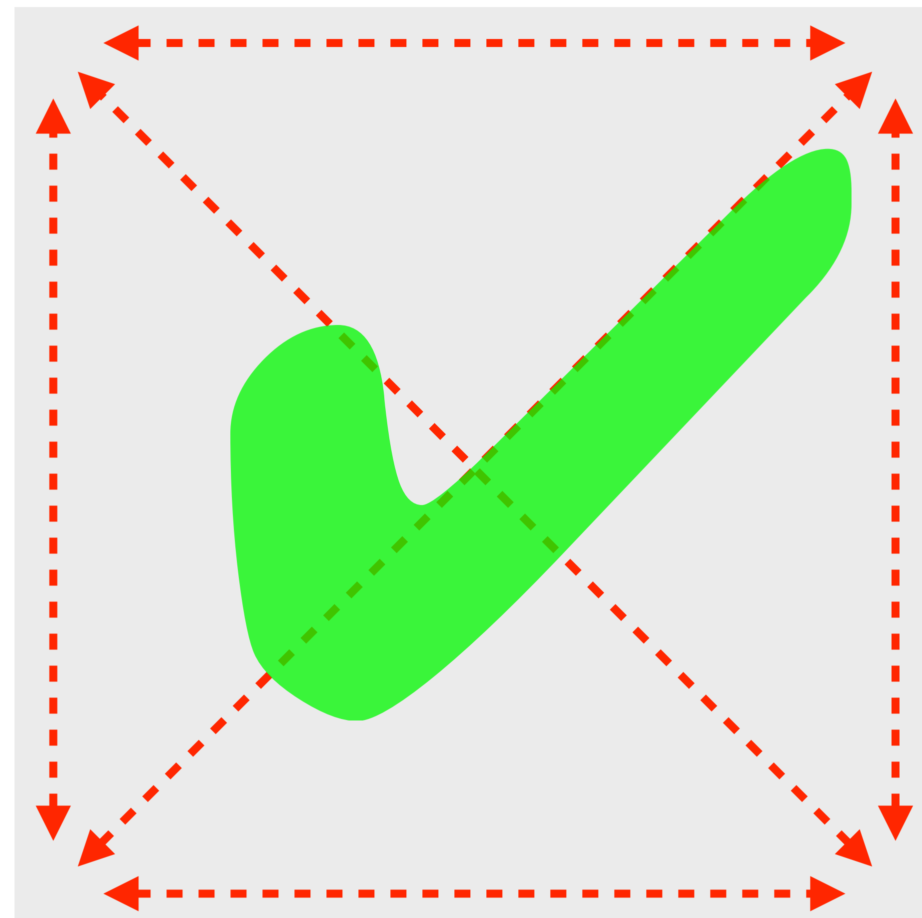
Parallelization of FBPIC

Standard FDTD
Domain Decomposition



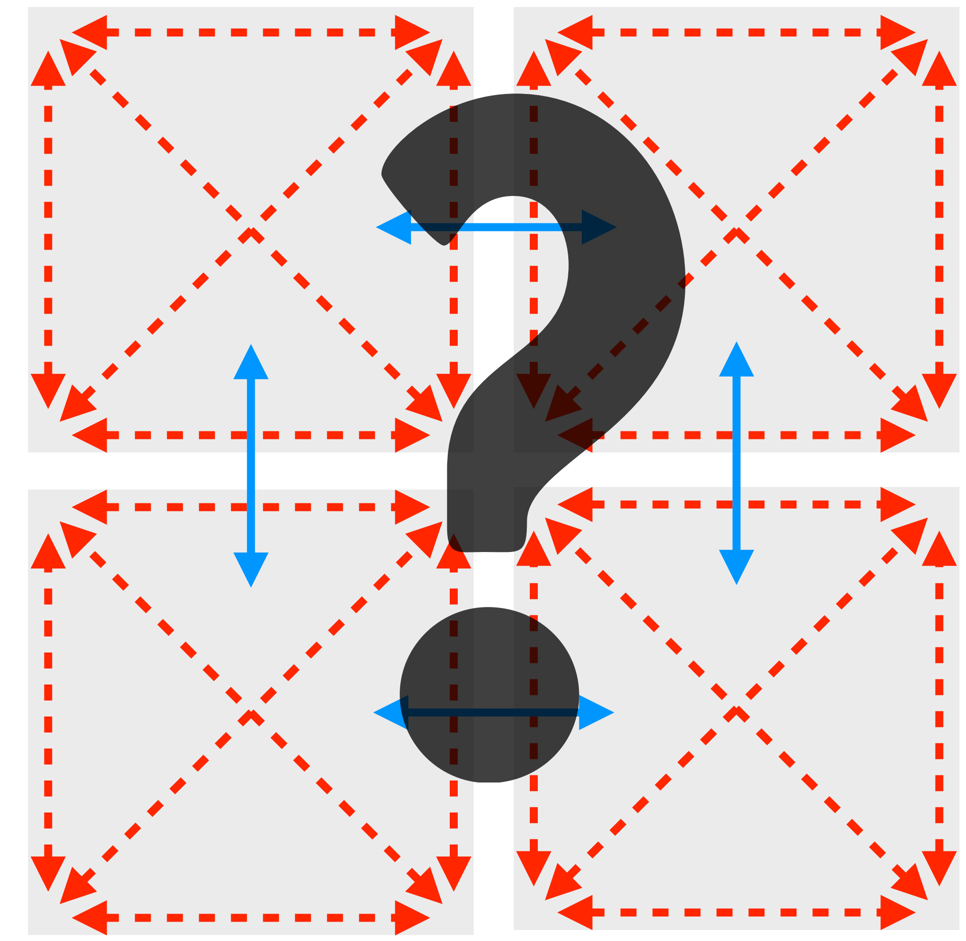
local exchange
low accuracy

PSATD
Transformations



global communication
high accuracy

Local Transformations &
Domain Decomposition



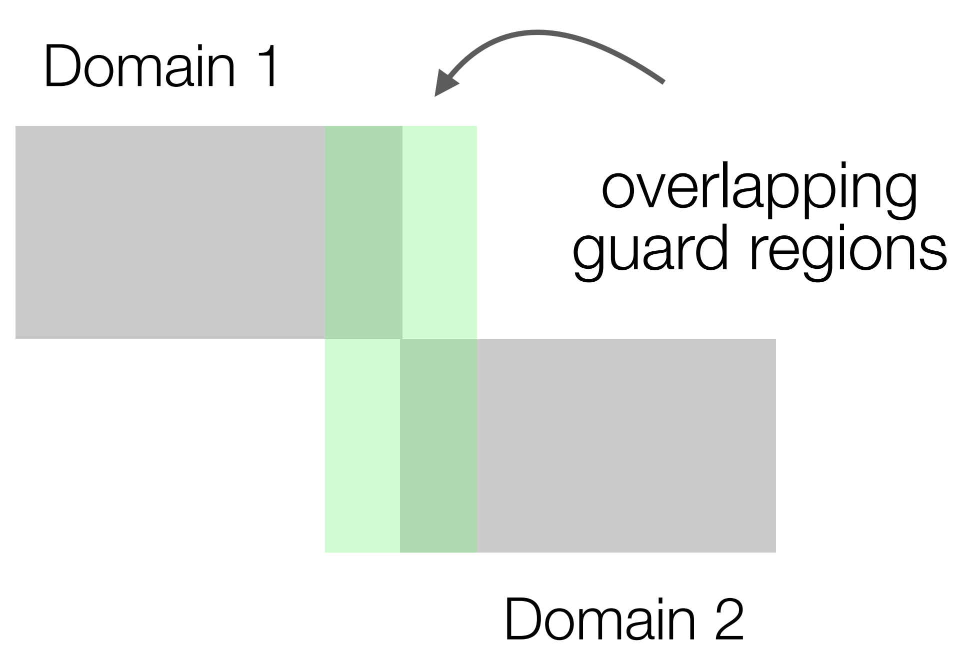
local communication & exchange
limited accuracy

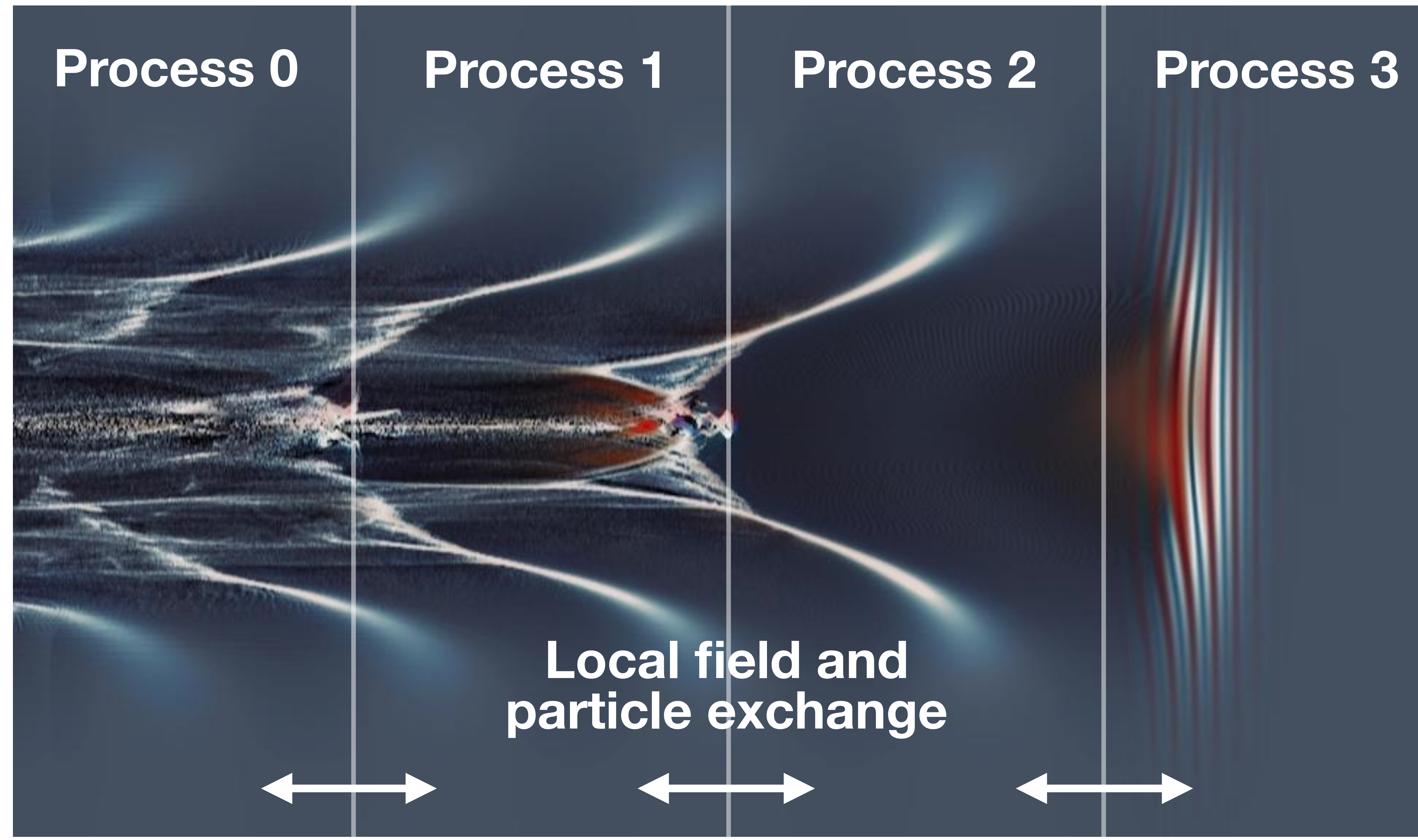
work in progress

Inter-Node Parallelization

Spatial domain decomposition

- Split work by spatial decomposition
- Domains computed in parallel
- Exchange local information at boundaries
- Order of accuracy defines guard region size (**Large guard regions for quasi-spectral accuracy**)



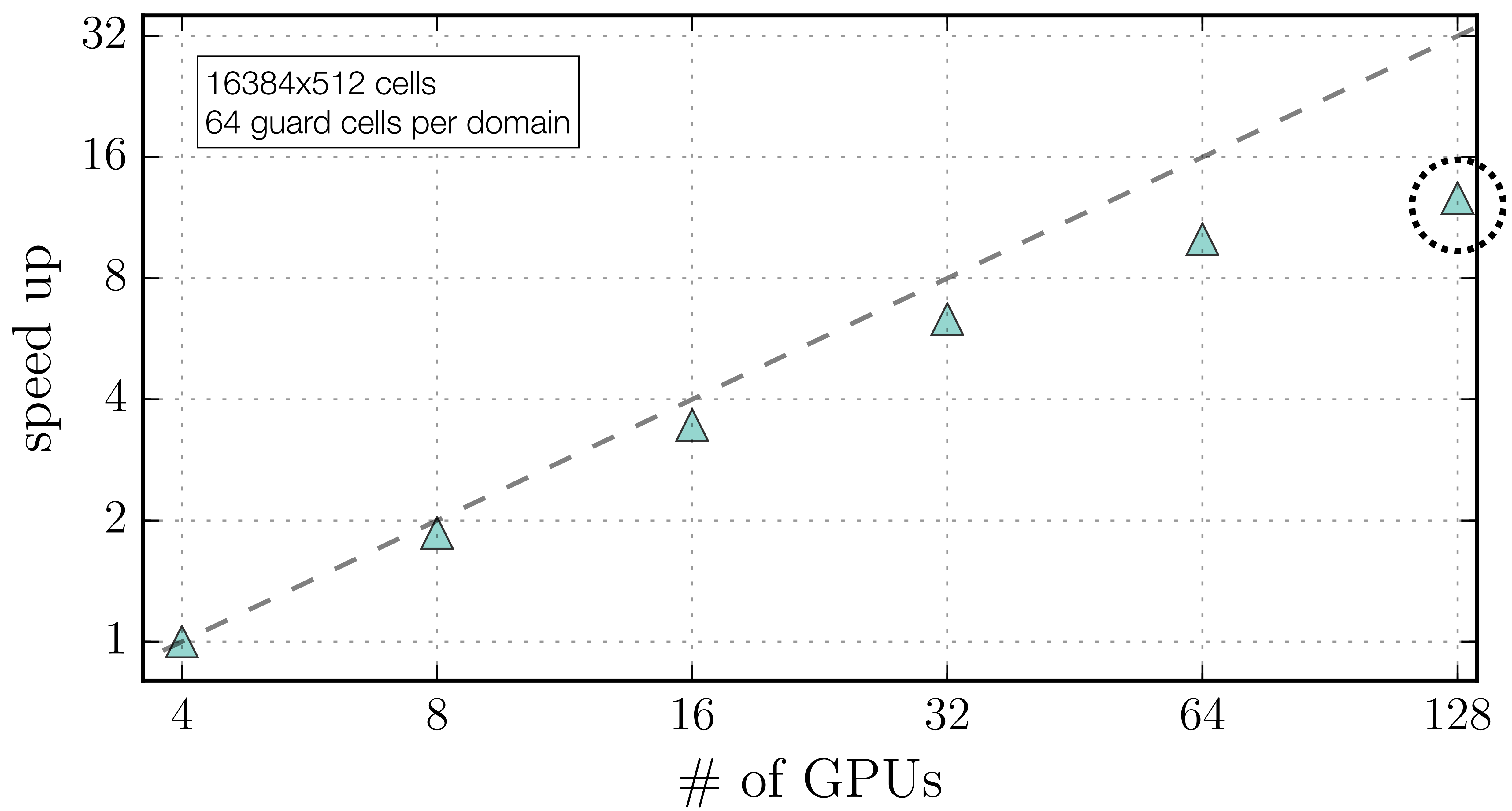


Concept of domain decomposition in the longitudinal direction

Scaling of the MPI version of FBPIC

Strong scaling on JURECA supercomputer (Nvivida K80)

Preliminary results *(not optimized)*



guard region size
=
local domain size

Best strategy for our case:

Extensive Intra-node parallelization on the GPU and only a few Inter-node domains.

For productive and fast simulations: 4-32 GPUs more than enough!

Summary

- **Motivation:** Efficient and easy parallelization of a novel PIC algorithm to combine speed, accuracy and usability in order to work productively as a physicist

- **FBPIC** is entirely **written in Python** (easy to develop and maintain the code)
- Implementation uses **Numba** (JIT compilation and interface for writing CUDA-Python)
- **Intra-** and **Inter-node parallelization** approach suitable for spectral algorithms
- Single **GPU** well suited for global operations (FFT & DHT)
- Enabling CUDA support for the full code took **less than 3 weeks**
- **Multi-GPU parallelization** by spatial domain decomposition with **mpi4py**

- **Outlook:** Finalize Multi-GPU, CUDA Streams, GPU Direct, OpenSourcing of FBPIC

Thanks… Questions?

funding contributed by

UHH DESY CFEL SCIENCE

eli beamlines

FSP302 **BMBF** Bundesministerium für Bildung und Forschung

Special thanks to Rémi Lehe

thanks to

BERKELEY LAB LBNL

JÜLICH FORSCHUNGSZENTRUM JURECA supercomputer

Warp LBNL WARP code

University of Strathclyde Glasgow group Brian McNeil

HZB Helmholtz Zentrum Berlin group Johannes Bahrdt

DESY group Jens Osterhoff