

# Modelling of ultra-intense laser propagation in plasmas and laser-plasma accelerators: fundamentals

*Laserlab-Europe*



Instituto Universitário de Lisboa

**R. A. Fonseca<sup>1,2</sup>, Jorge Vieira<sup>1</sup>**

<sup>1</sup>GoLP/IPFN, Instituto Superior Técnico, Lisboa, Portugal

<sup>2</sup> DCTI, ISCTE-Instituto Universitário de Lisboa, Portugal

- **Session 1 - PIC codes and ZPIC installation**

- Basics of PIC simulations
- ZPIC installation and use

- **Session 2 - Laser dynamics and plasma accelerators**

- Introduction to laser wakefield accelerators
- Numerical modelling of laser-plasma interactions

- **Session 3 - Hands on**

- Laser propagation in plasmas
- Advanced Visualization and Data analysis
- Challenges for participants

- **Session 4 - Participant flash presentations**

- Challenges results
- Wrap-up

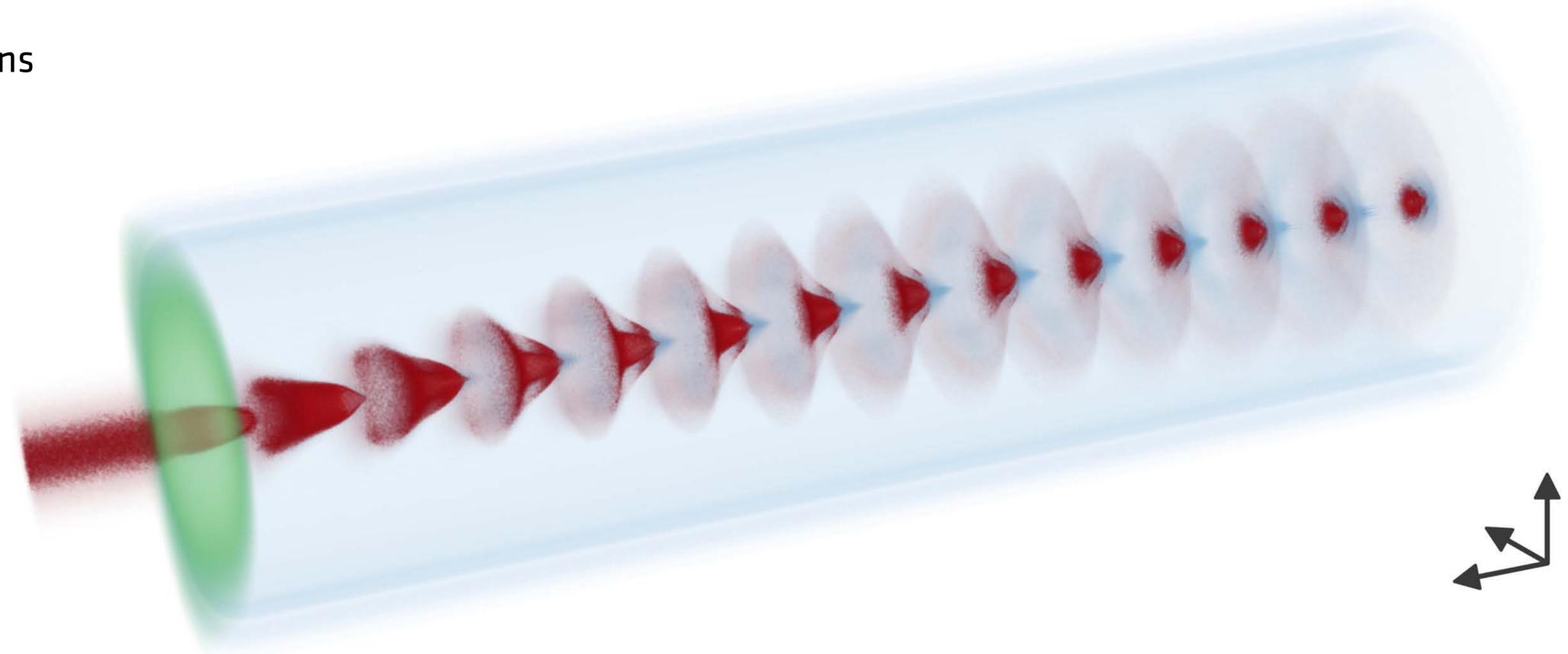
## LETTER

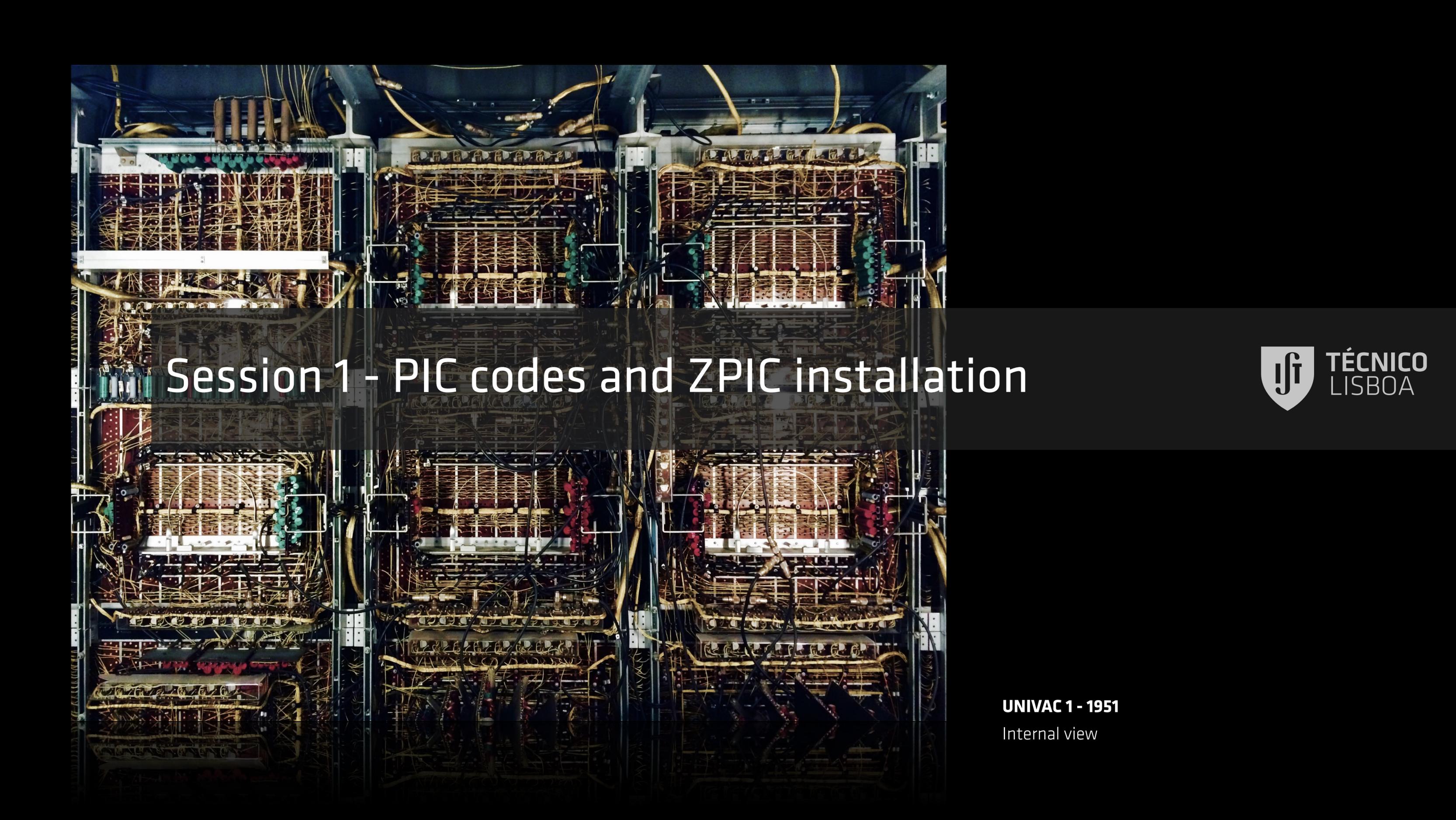
OPEN

<https://doi.org/10.1038/s41586-018-0485-4>

### Acceleration of electrons in the plasma wakefield of a proton bunch

E. Adli<sup>1</sup>, A. Ahuja<sup>2</sup>, O. Apsimon<sup>3,4</sup>, R. Apsimon<sup>4,5</sup>, A.-M. Bachmann<sup>2,6,7</sup>, D. Barrientos<sup>2</sup>, F. Batsch<sup>2,6,7</sup>, J. Bauche<sup>2</sup>, V. K. Berglyd Olsen<sup>1</sup>, M. Bernardini<sup>2</sup>, T. Bohl<sup>2</sup>, C. Bracco<sup>2</sup>, F. Braunmüller<sup>6</sup>, G. Burt<sup>4,5</sup>, B. Buttenschön<sup>8</sup>, A. Caldwell<sup>6</sup>, M. Cascella<sup>9</sup>, J. Chappell<sup>9</sup>, E. Chevallay<sup>2</sup>, M. Chung<sup>10</sup>, D. Cooke<sup>9</sup>, H. Damerau<sup>2</sup>, L. Deacon<sup>9</sup>, L. H. Deubner<sup>11</sup>, A. Dexter<sup>4,5</sup>, S. Doeberl<sup>2</sup>, J. Farmer<sup>12</sup>, V. N. Fedosseev<sup>2</sup>, R. Fiorito<sup>4,13</sup>, R. A. Fonseca<sup>14</sup>, F. Friebe<sup>2</sup>, L. Garolfi<sup>2</sup>, S. Gessner<sup>2</sup>, I. Gorgisyan<sup>2</sup>, A. A. Gorn<sup>15,16</sup>, E. Granados<sup>2</sup>, O. Grulke<sup>8,17</sup>, E. Gschwendtner<sup>2</sup>, J. Hansen<sup>2</sup>, A. Helm<sup>18</sup>, J. R. Henderson<sup>4,5</sup>, M. Hüther<sup>6</sup>, M. Ibson<sup>4,13</sup>, L. Jensen<sup>2</sup>, S. Jolly<sup>9</sup>, F. Keeble<sup>9</sup>, S.-Y. Kim<sup>10</sup>, F. Kraus<sup>11</sup>, Y. Li<sup>3,4</sup>, S. Liu<sup>19</sup>, N. Lopes<sup>18</sup>, K. V. Lotov<sup>15,16</sup>, L. Maricalva Brun<sup>2</sup>, M. Martyanov<sup>6</sup>, S. Mazzoni<sup>2</sup>, D. Medina Godoy<sup>2</sup>, V. A. Minakov<sup>15,16</sup>, J. Mitchell<sup>4,5</sup>, J. C. Molendijk<sup>2</sup>, J. T. Moody<sup>6</sup>, M. Moreira<sup>2,18</sup>, P. Muggli<sup>2,6</sup>, E. Öz<sup>6</sup>, C. Pasquino<sup>2</sup>, A. Pardons<sup>2</sup>, F. Peña Asmus<sup>6,7</sup>, K. Pepitone<sup>2</sup>, A. Perera<sup>4,13</sup>, A. Petrenko<sup>2,15</sup>, S. Pitman<sup>4,5</sup>, A. Pukhov<sup>12</sup>, S. Rey<sup>2</sup>, K. Rieger<sup>6</sup>, H. Ruhl<sup>20</sup>, J. S. Schmidt<sup>2</sup>, I. A. Shalimova<sup>16,21</sup>, P. Sherwood<sup>9</sup>, L. O. Silva<sup>18</sup>, L. Soby<sup>2</sup>, A. P. Sosedkin<sup>15,16</sup>, R. Speroni<sup>2</sup>, R. I. Spitsyn<sup>15,16</sup>, P. V. Tuev<sup>15,16</sup>, M. Turner<sup>2</sup>, F. Velotti<sup>2</sup>, L. Verra<sup>2,22</sup>, V. A. Verzilov<sup>19</sup>, J. Vieira<sup>18</sup>, C. P. Welsch<sup>4,13</sup>, B. Williamson<sup>3,4</sup>, M. Wing<sup>9\*</sup>, B. Woolley<sup>2</sup> & G. Xia<sup>3,4</sup>





# Session 1 - PIC codes and ZPIC installation

**UNIVAC 1 - 1951**

Internal view



# Committed to open science

## Open-access model

- 40+ research groups worldwide are using OSIRIS
- 300+ publications in leading scientific journals
- Large developer and user community
- Detailed documentation and sample inputs files available

## Using OSIRIS 4.0

- The code can be used freely by research institutions after signing an MoU
- Find out more at:

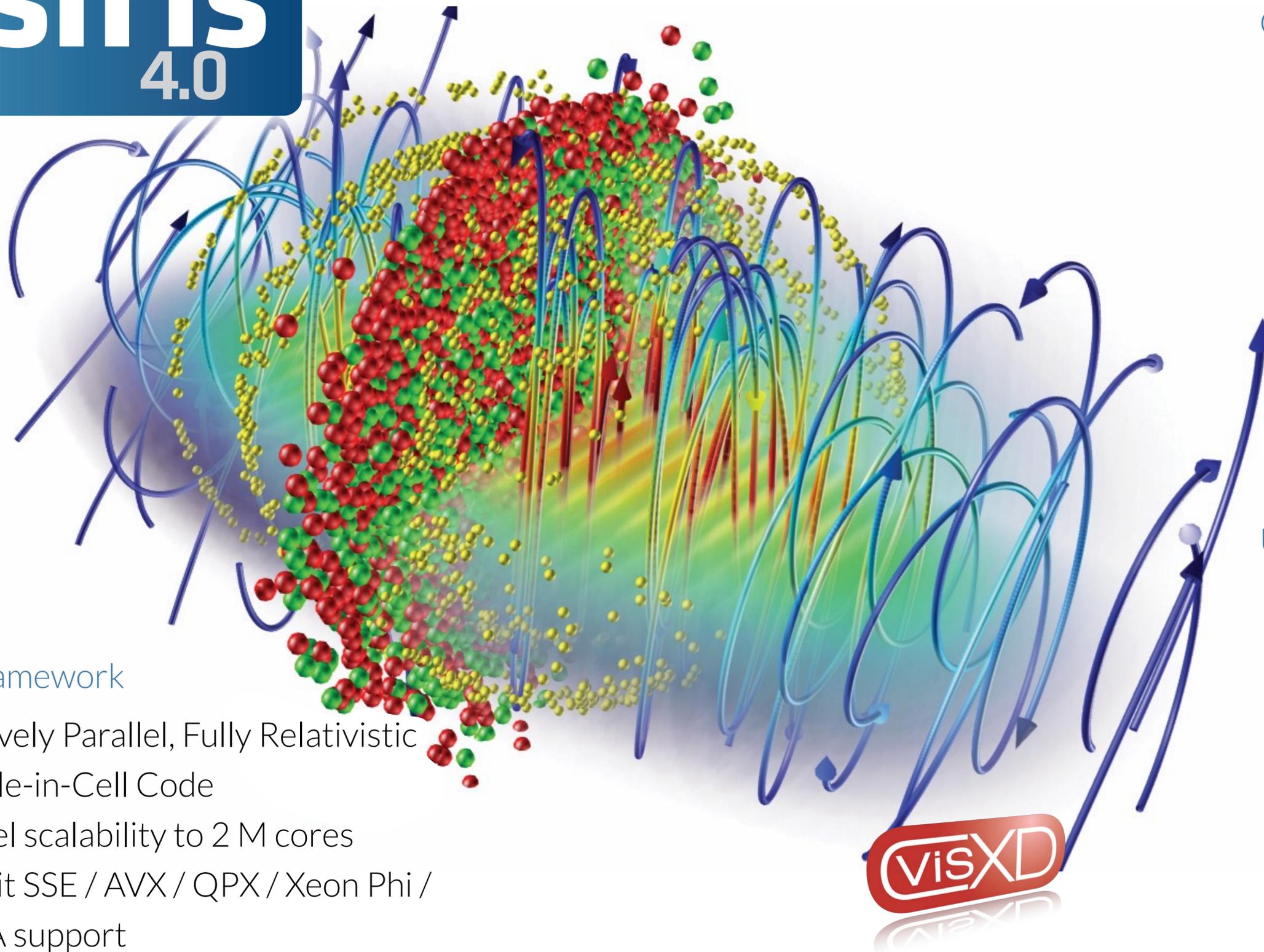
<http://epp.tecnico.ulisboa.pt/osiris>



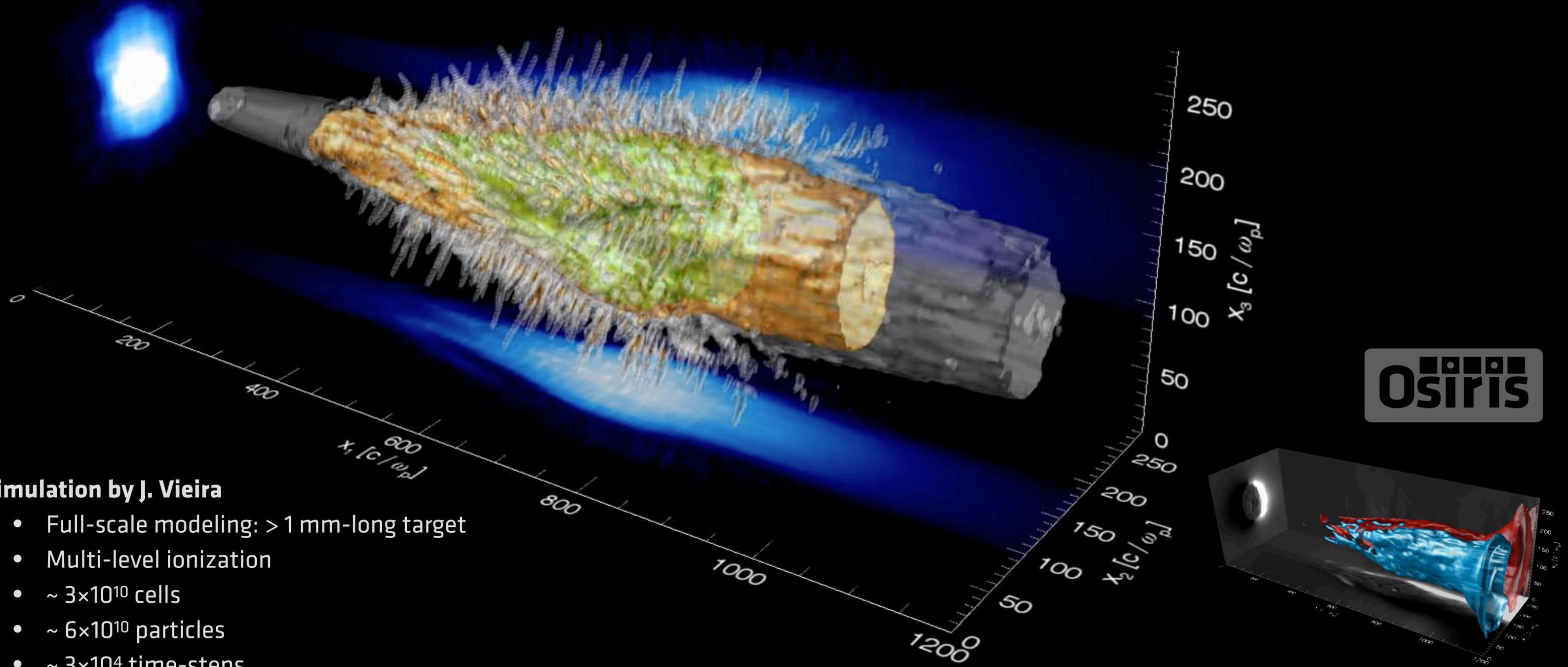
Ricardo Fonseca: [ricardo.fonseca@tecnico.ulisboa.pt](mailto:ricardo.fonseca@tecnico.ulisboa.pt)

## OSIRIS framework

- Massively Parallel, Fully Relativistic Particle-in-Cell Code
- Parallel scalability to 2 M cores
- Explicit SSE / AVX / QPX / Xeon Phi / CUDA support
- Extended physics/simulation models



# Full scale 3D LWFA modeling

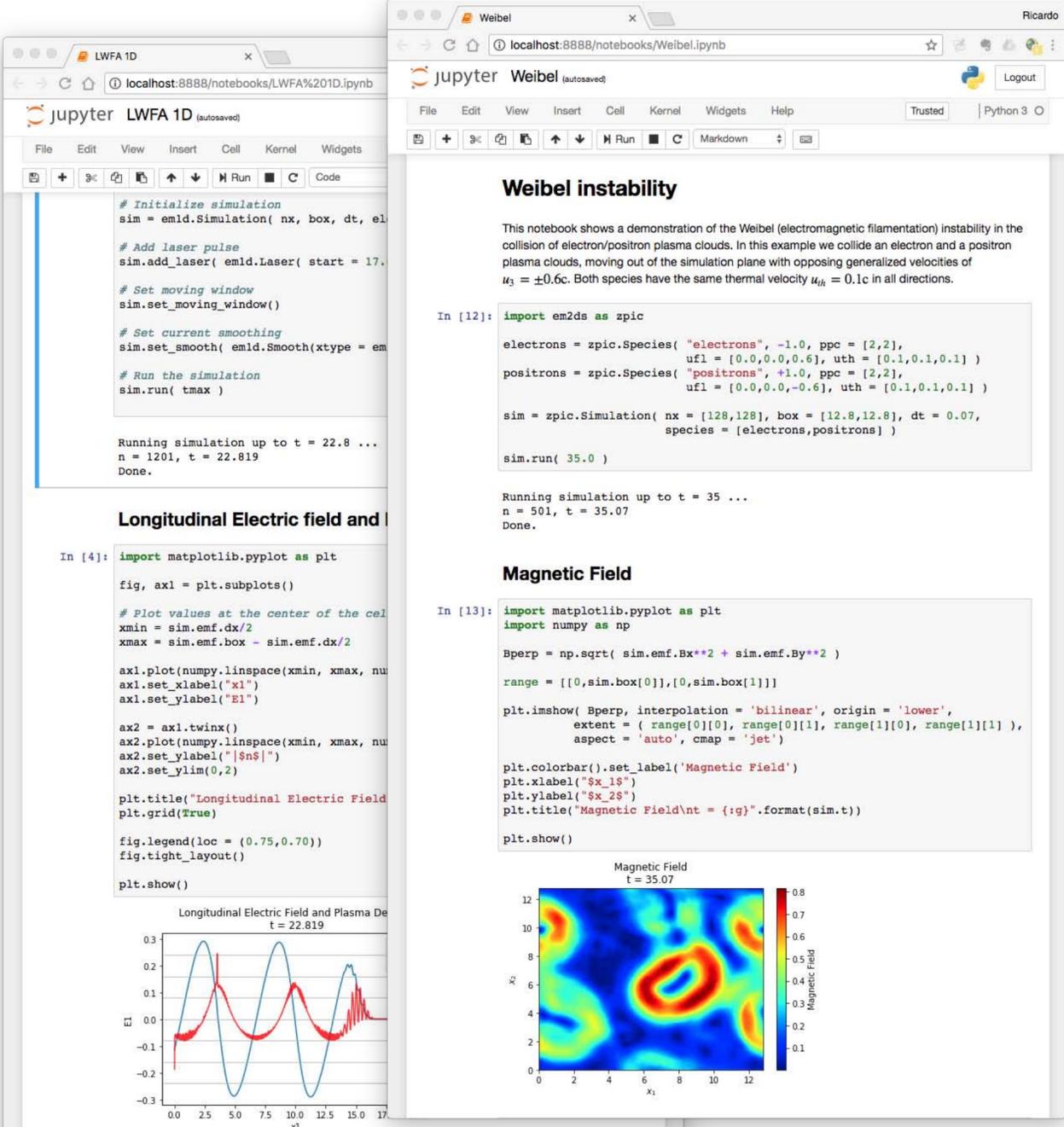


## Simulation by J. Vieira

- Full-scale modeling: > 1 mm-long target
- Multi-level ionization
- $\sim 3 \times 10^{10}$  cells
- $\sim 6 \times 10^{10}$  particles
- $\sim 3 \times 10^4$  time-steps
- $\sim 0.4$  million core h ( $\sim 16$  k€)

# The ZPIC educational code suite

- **ZPIC code suite**
  - Open-source PIC code suit for plasma physics education
  - Fully relativistic 1D and 2D EM-PIC algorithm
  - Electrostatic 1D/2D PIC algorithm
- **Requirements**
  - No external dependencies, requires only C99 compiler
  - Python interface (optional)
- **Jupyter Notebooks**
  - Includes set of Python notebooks with example problems
  - Detailed explanations of code use and physics
- **Also available through Docker**
  - If you just want to run the notebooks you can use a Docker image available on DockerHub: **zamb/zpic**



**LWFA 1D**

```

# Initialize simulation
sim = em1d.Simulation( nx, box, dt, el

# Add laser pulse
sim.add_laser( em1d.Laser( start = 17.

# Set moving window
sim.set_moving_window()

# Set current smoothing
sim.set_smooth( em1d.Smooth(xtype = em

# Run the simulation
sim.run( tmax )

Running simulation up to t = 22.8 ...
n = 1201, t = 22.819
Done.

Longitudinal Electric field and Plasma De

In [4]: import matplotlib.pyplot as plt

fig, ax1 = plt.subplots()

# Plot values at the center of the cell
xmin = sim.emf.dx/2
xmax = sim.emf.box - sim.emf.dx/2

ax1.plot(numpy.linspace(xmin, xmax, nu
ax1.set_xlabel("x1")
ax1.set_ylabel("E1")

ax2 = ax1.twinx()
ax2.plot(numpy.linspace(xmin, xmax, nu
ax2.set_ylabel("$n_1$")
ax2.set_ylim(0,2)

plt.title("Longitudinal Electric Field
plt.grid(True)

fig.legend(loc = (0.75,0.70))
fig.tight_layout()

plt.show()

Longitudinal Electric Field and Plasma De
t = 22.819

```

**Weibel instability**

This notebook shows a demonstration of the Weibel (electromagnetic filamentation) instability in the collision of electron/positron plasma clouds. In this example we collide an electron and a positron plasma clouds, moving out of the simulation plane with opposing generalized velocities of  $u_3 = \pm 0.6c$ . Both species have the same thermal velocity  $u_{th} = 0.1c$  in all directions.

```

In [12]: import em2ds as zpic

electrons = zpic.Species( "electrons", -1.0, ppc = [2,2],
                        ufl = [0.0,0.0,0.6], uth = [0.1,0.1,0.1] )
positrons = zpic.Species( "positrons", +1.0, ppc = [2,2],
                        ufl = [0.0,0.0,-0.6], uth = [0.1,0.1,0.1] )

sim = zpic.Simulation( nx = [128,128], box = [12.8,12.8], dt = 0.07,
                      species = [electrons,positrons] )

sim.run( 35.0 )

Running simulation up to t = 35 ...
n = 501, t = 35.07
Done.

Magnetic Field

In [13]: import matplotlib.pyplot as plt
import numpy as np

Bperp = np.sqrt( sim.emf.Bx**2 + sim.emf.By**2 )

range = [[0,sim.box[0]],[0,sim.box[1]]]

plt.imshow( Bperp, interpolation = 'bilinear', origin = 'lower',
            extent = ( range[0][0], range[0][1], range[1][0], range[1][1] ),
            aspect = 'auto', cmap = 'jet' )

plt.colorbar().set_label('Magnetic Field')
plt.xlabel("$x_1$")
plt.ylabel("$x_2$")
plt.title("Magnetic Field\nt = {:g}".format(sim.t))

plt.show()

Magnetic Field
t = 35.07

```

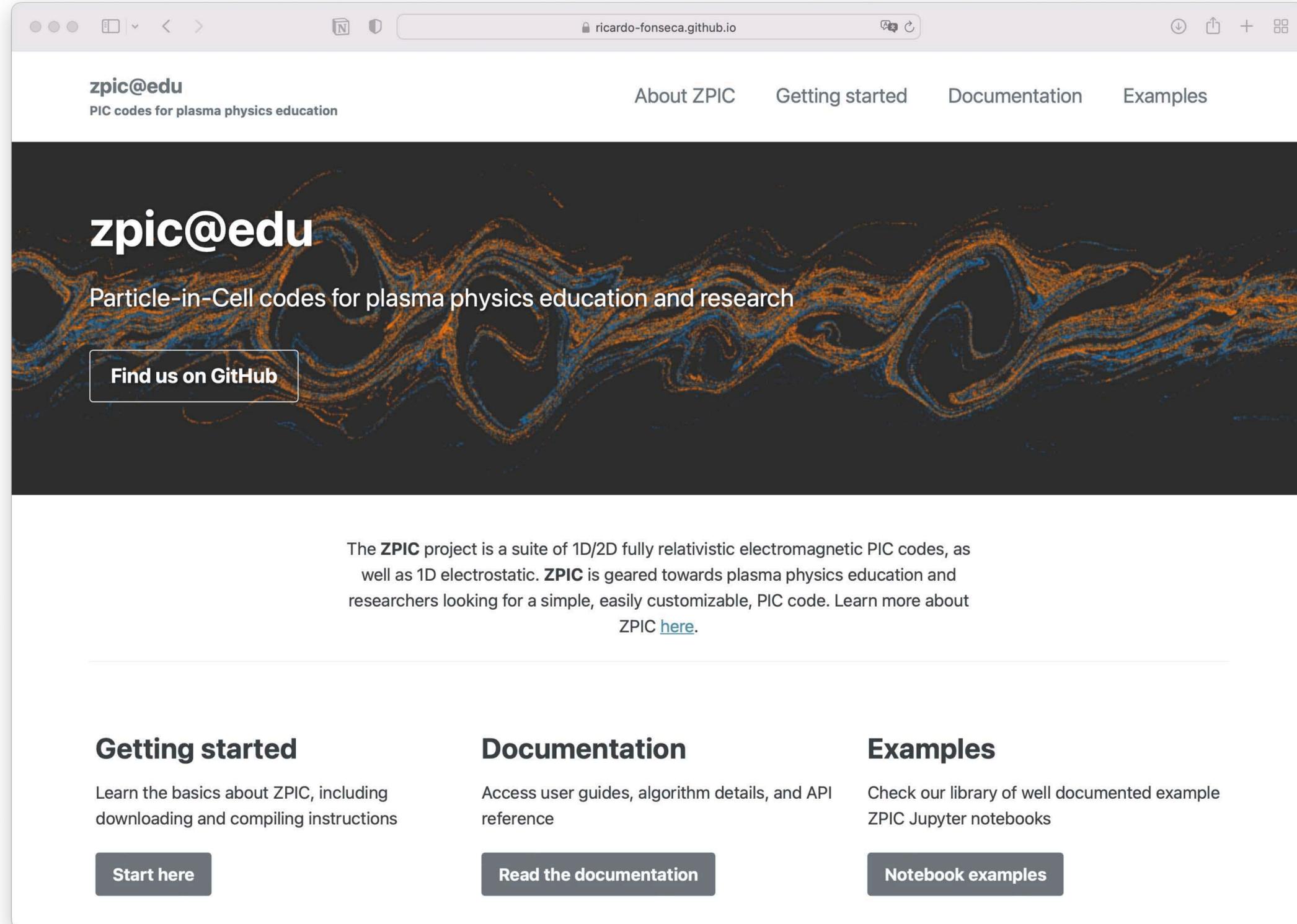
# zpic@edu



**Come find us on GitHub**  
[github.com/ricardo-fonseca/zpic](https://github.com/ricardo-fonseca/zpic)

# ZPIC documentation

<https://ricardo-fonseca.github.io/zpic>



The screenshot shows a web browser displaying the ZPIC documentation website. The browser's address bar shows the URL `ricardo-fonseca.github.io`. The website has a navigation menu with links for "About ZPIC", "Getting started", "Documentation", and "Examples". The main content area features a dark background with a blue and orange plasma visualization. The text "zpic@edu" is prominently displayed, followed by the subtitle "Particle-in-Cell codes for plasma physics education and research". A button labeled "Find us on GitHub" is visible. Below this, a paragraph describes the ZPIC project as a suite of 1D/2D fully relativistic electromagnetic PIC codes, also including 1D electrostatic codes. It is geared towards plasma physics education and researchers seeking a simple, customizable PIC code. A link "here" is provided for more information. At the bottom, there are three columns: "Getting started" with a "Start here" button, "Documentation" with a "Read the documentation" button, and "Examples" with a "Notebook examples" button.

**zpic@edu**  
PIC codes for plasma physics education

About ZPIC Getting started Documentation Examples

**zpic@edu**  
Particle-in-Cell codes for plasma physics education and research

Find us on GitHub

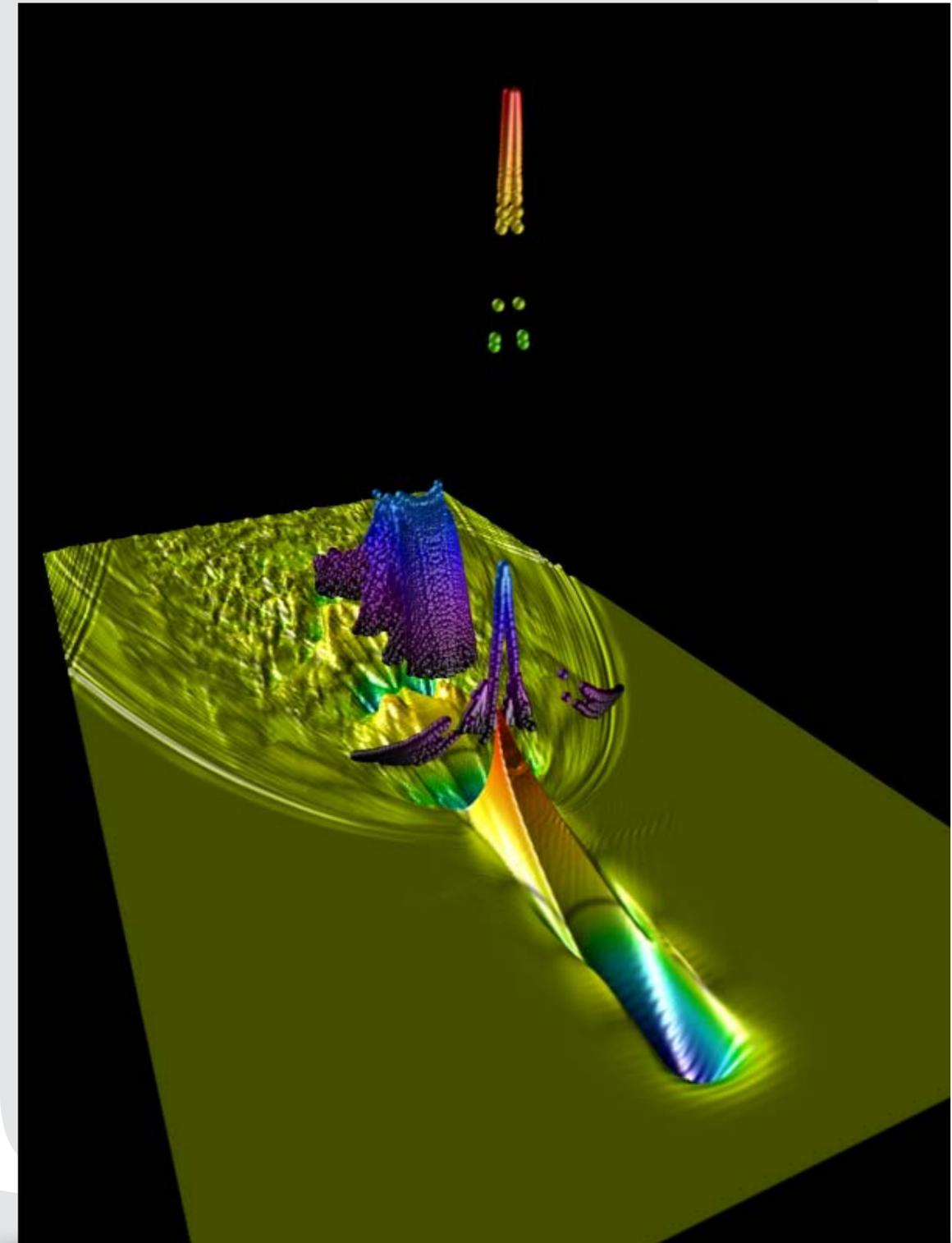
The **ZPIC** project is a suite of 1D/2D fully relativistic electromagnetic PIC codes, as well as 1D electrostatic. **ZPIC** is geared towards plasma physics education and researchers looking for a simple, easily customizable, PIC code. Learn more about ZPIC [here](#).

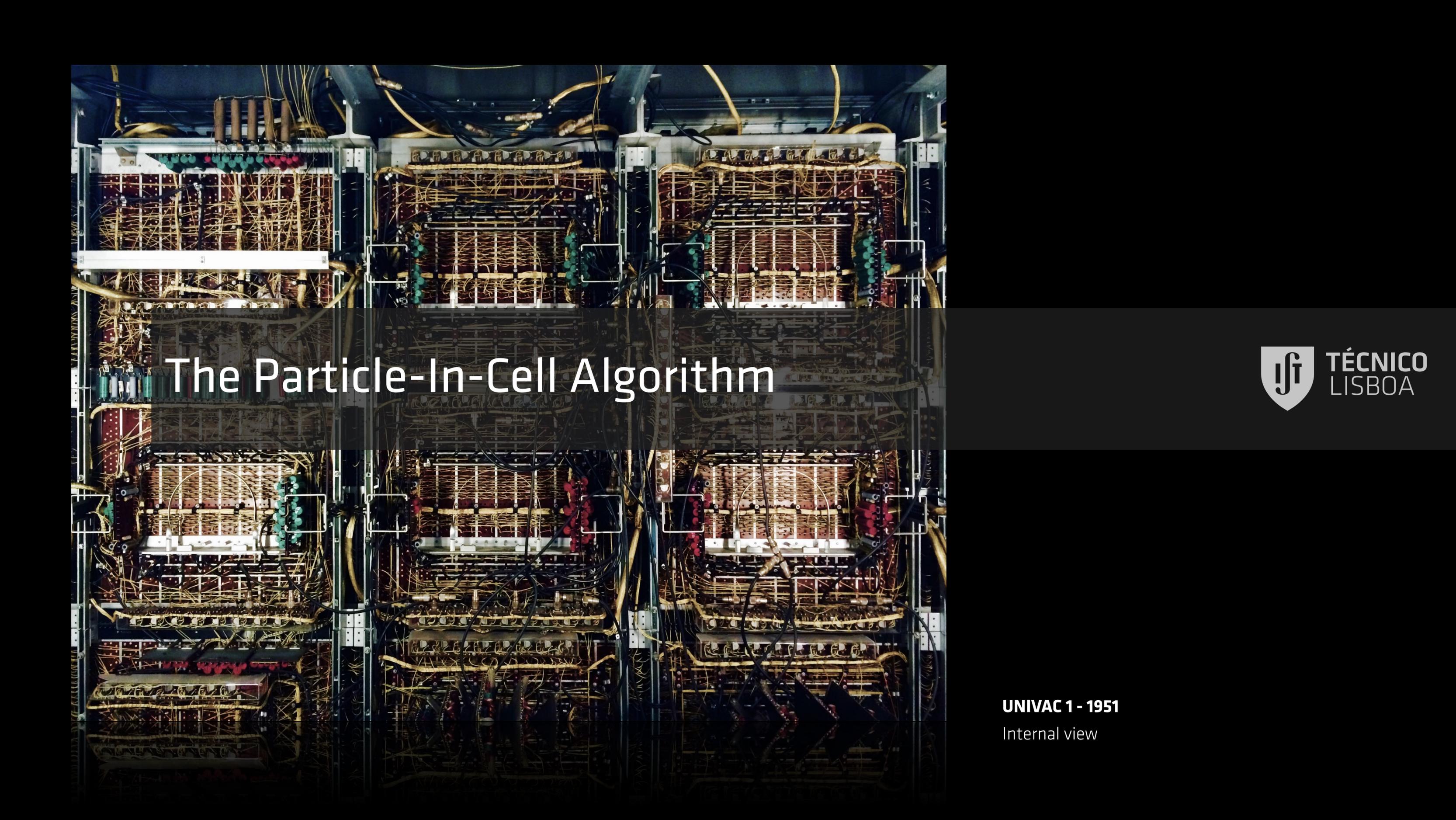
**Getting started**  
Learn the basics about ZPIC, including downloading and compiling instructions  
[Start here](#)

**Documentation**  
Access user guides, algorithm details, and API reference  
[Read the documentation](#)

**Examples**  
Check our library of well documented example ZPIC Jupyter notebooks  
[Notebook examples](#)

- **Review of the Particle-In Cell Algorithm**
  - Plasma simulation using particles
  - The Particle-In-Cell algorithm
  - Units and Normaliation
  - Time-step considerations
- **Installing ZPIC on your Computer**
  - Compiling from source
  - Using a Docker image
- **Running ZPIC**
  - Resolution and box size
  - Simulation Particles
  - Additional useful diagnostics
- **A first laser propagation simulation**
  - Resolution and box size





# The Particle-In-Cell Algorithm

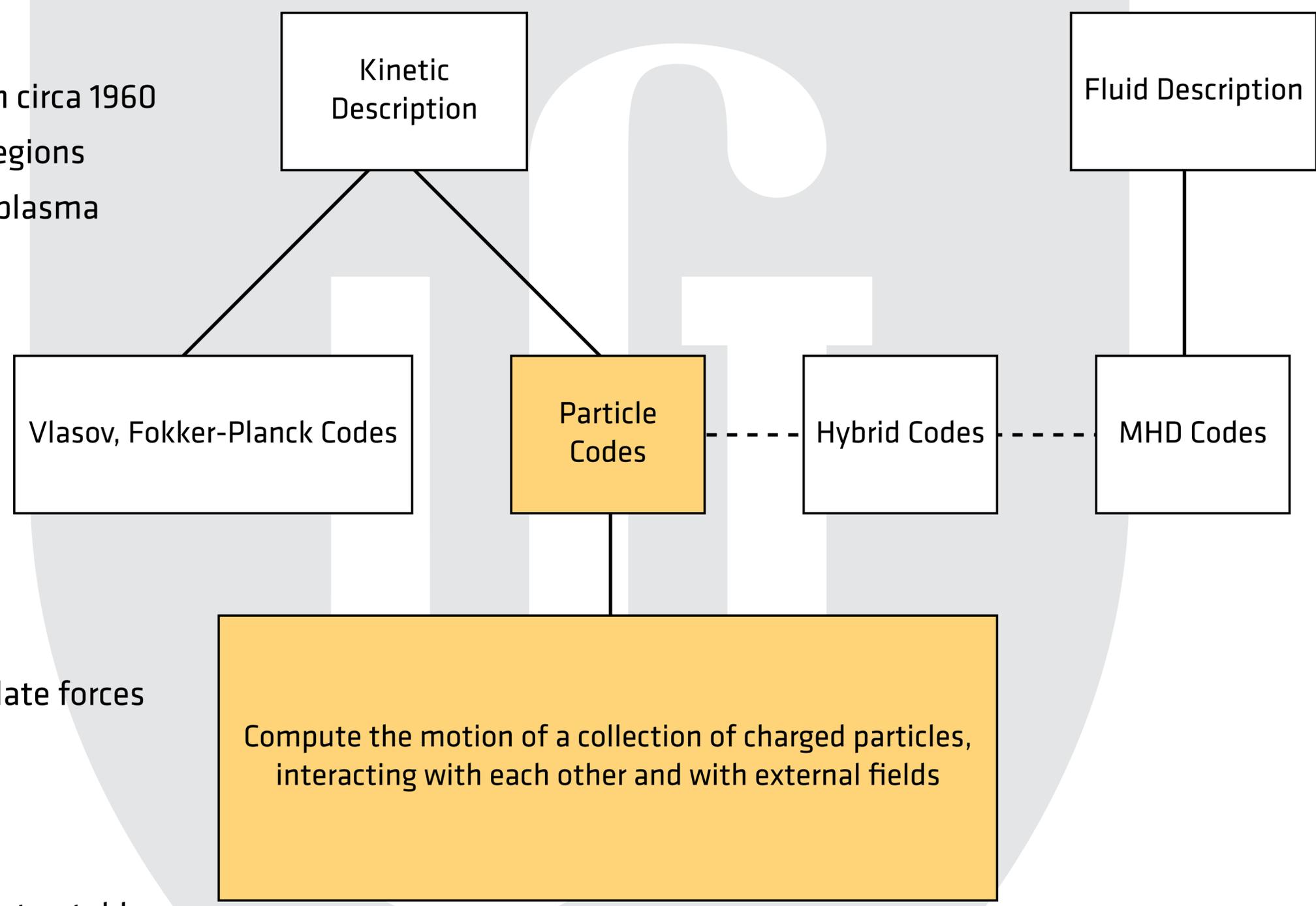
# Overview of Plasma Simulation Algorithms

- **Plasma Simulations Using Particles**

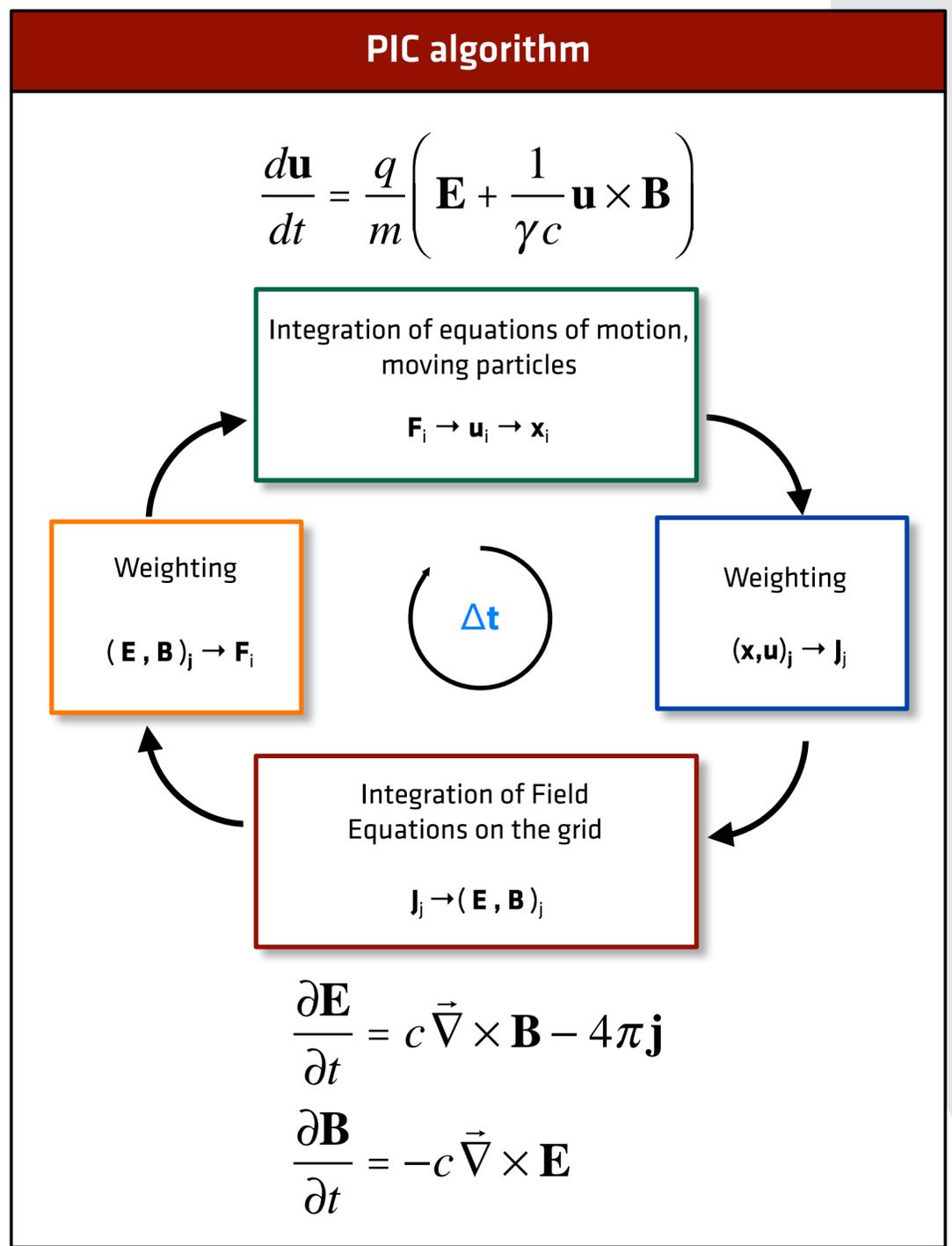
- Pioneered by John Dawson and Oscar Buneman circa 1960
- Use macro particles to simulate large spatial regions
  - 1 simulation particle corresponds to many plasma particles
- Particle-Particle simulations
  - Computations go with  $O(N_p^2)$
  - Computationally very demanding

- **Particle-In-Cell algorithms**

- Interact particles through fields
- Discretize fields on grids
- Interpolate fields at particle positions to calculate forces
- Deposit particle charge/current on a grid
- Particle-Mesh algorithm
  - Computations go with  $O(N_p)$
  - Still computationally heavy but much more tractable



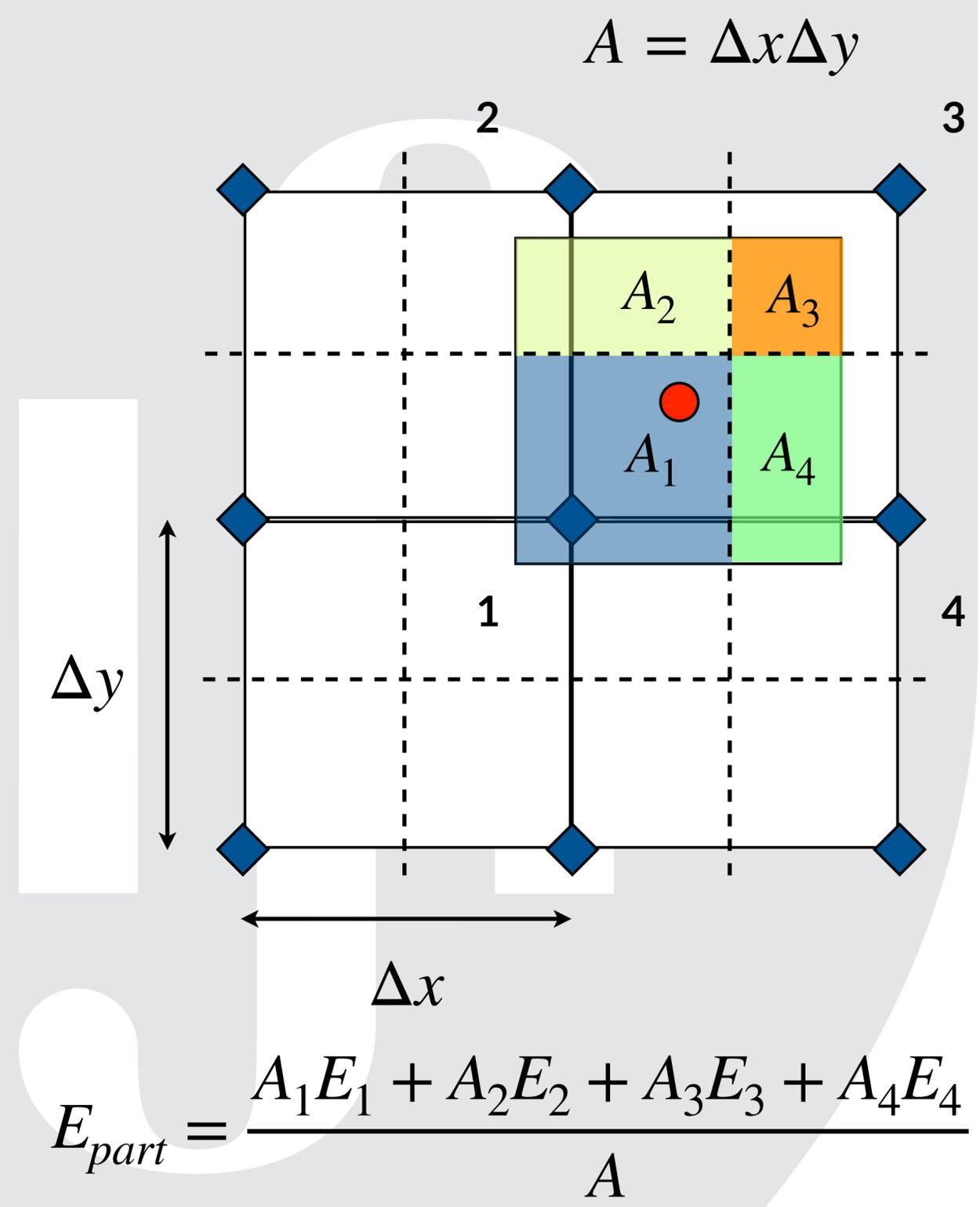
# The particle-in-cell (PIC) Algorithm



- **Fully Relativistic, Electromagnetic Particle-In-Cell algorithm**
  - Discretize Electric and Magnetic fields on a grid
  - Cell size must resolve shortest relevant lengths in the simulation
    - Typically the laser wavelength or the plasma skin depth
  - Represent plasma particles with simulation macro-particles
    - Free to move in entire  $nD-3V$  phasespace
    - Each macro-particle represents several plasma particles
  - Must have enough particles per cell to properly resolve velocity distributions
- **Fields and particles don't exist in the same simulation topology**
  - Field quantities are limited to grid points
  - Field interpolation connects fields  $\rightarrow$  particles
  - Current deposition connects particles  $\rightarrow$  fields
- **Four major steps**
  - Field interpolation
  - Particle advance
  - Current deposition
  - Field advance

# Interpolating the fields

- **Particles are free to move to any position**
  - Field are discretized on a grid
  - Field values at particle positions are required to advance particle momenta
- **Interpolate fields at particle positions**
  - ZPIC uses linear interpolation
  - In 2D this can be viewed as area weighting
  - The interpolating scheme must be consistent with charge / current deposition
- **Momentum conserving algorithm**
  - Avoids self-forces
  - $dp/dt = 0$  for single particle



# Pushing the particles

- **Advance generalized velocity and position of individual particles**
  - ZPIC is a fully relativistic code so we work with  $\mathbf{u} = \gamma \boldsymbol{\beta}$  instead  $\mathbf{v}$ .
  - We use a leap-frog scheme to integrate particle motion:
    - Positions ( $\mathbf{x}$ ) are defined at integral time-steps  $t^n$
    - Velocities ( $\mathbf{u}$ ) are defined at half time-steps  $t^{n+1/2}$
  - Second-order accuracy in time
- **Velocities are integrated using a relativistic Boris pusher**
  - Separate  $\mathbf{E}$  and  $\mathbf{B}$  contributions
    - Accelerate with  $1/2$  electric impulse
    - Full magnetic field rotation
    - Add remaining  $1/2$  electric impulse
  - Fully relativistic, second order time accurate
  - By construction, no work from  $\mathbf{B}$  field
- **Position advance is straightforward**
  - ZPIC stores cell index and position inside cell

**Advance momenta**

- $\mathbf{u}^- = \mathbf{u}^{n-1/2} + \frac{q\mathbf{E}^n}{m} \frac{\Delta t}{2}$
- $\mathbf{u}' = \mathbf{u}^- + \mathbf{u}^- \times \mathbf{t}$
- $\mathbf{u}^+ = \mathbf{u}^- + \mathbf{u}^- \times \mathbf{s}$
- $\mathbf{u}^{n+1/2} = \mathbf{u}^+ + \frac{q\mathbf{E}^n}{m} \frac{\Delta t}{2}$

$$\mathbf{t} = -\frac{q\mathbf{B}^n}{\gamma^n mc} \frac{\Delta t}{2}$$

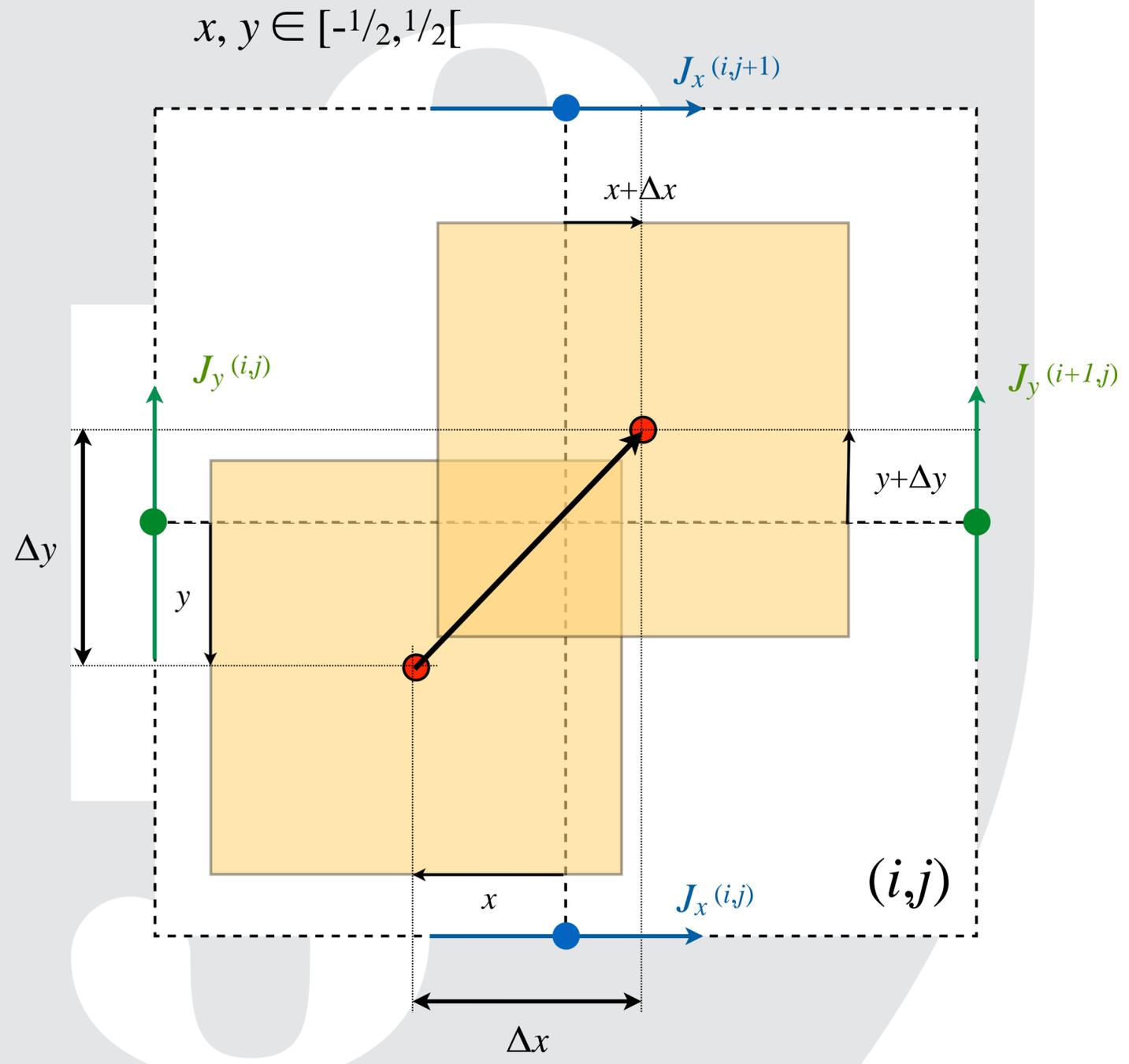
$$\mathbf{s} = \frac{2\mathbf{t}}{1 + \mathbf{t}^2}$$

**Advance positions**

$$\mathbf{x}^{n+1} = \mathbf{x}^n + \frac{\mathbf{u}^{n+1/2}}{\gamma^{n+1/2}} \Delta t$$

# Depositing the current

- **Connects particle motion to field equations**
  - Current deposition must satisfy continuity equation:
 
$$\frac{\partial \rho}{\partial t} = - \nabla' \cdot \mathbf{j}$$
  - The operator  $\nabla'$  corresponds to the finite difference approximation
  - Simply depositing  $\rho \mathbf{v}$  does not conserve charge
  - Critical to guarantee the solutions to Maxwell's equations are self-consistent
- **Exact charge conserving current deposition scheme**
  - Developed by Villaseñor and Buneman for linear interpolation
  - Looks at particle motion, not velocity
  - Limited to motion inside single cell
    - If particles cross cell boundary, motion is split into segments that don't cross boundaries



# Advancing the EM fields

- **EM Fields are advanced in time using Maxwell's equations using the deposited current as source terms**

- Rearrange Ampère's and Faraday's laws:

$$\frac{\partial \mathbf{E}}{\partial t} = \nabla' \times \mathbf{B} - \mathbf{j}$$

$$\frac{\partial \mathbf{B}}{\partial t} = -\nabla' \times \mathbf{E}$$

- Discretize temporal and spatial derivatives using finite differences

- **Careful time and spacial centering of quantities leads to 2<sup>nd</sup> order accuracy**

- ZPIC uses the Finite Difference Time Domain (FDTD) algorithm

- Fields are staggered in time for 2<sup>nd</sup> order accuracy

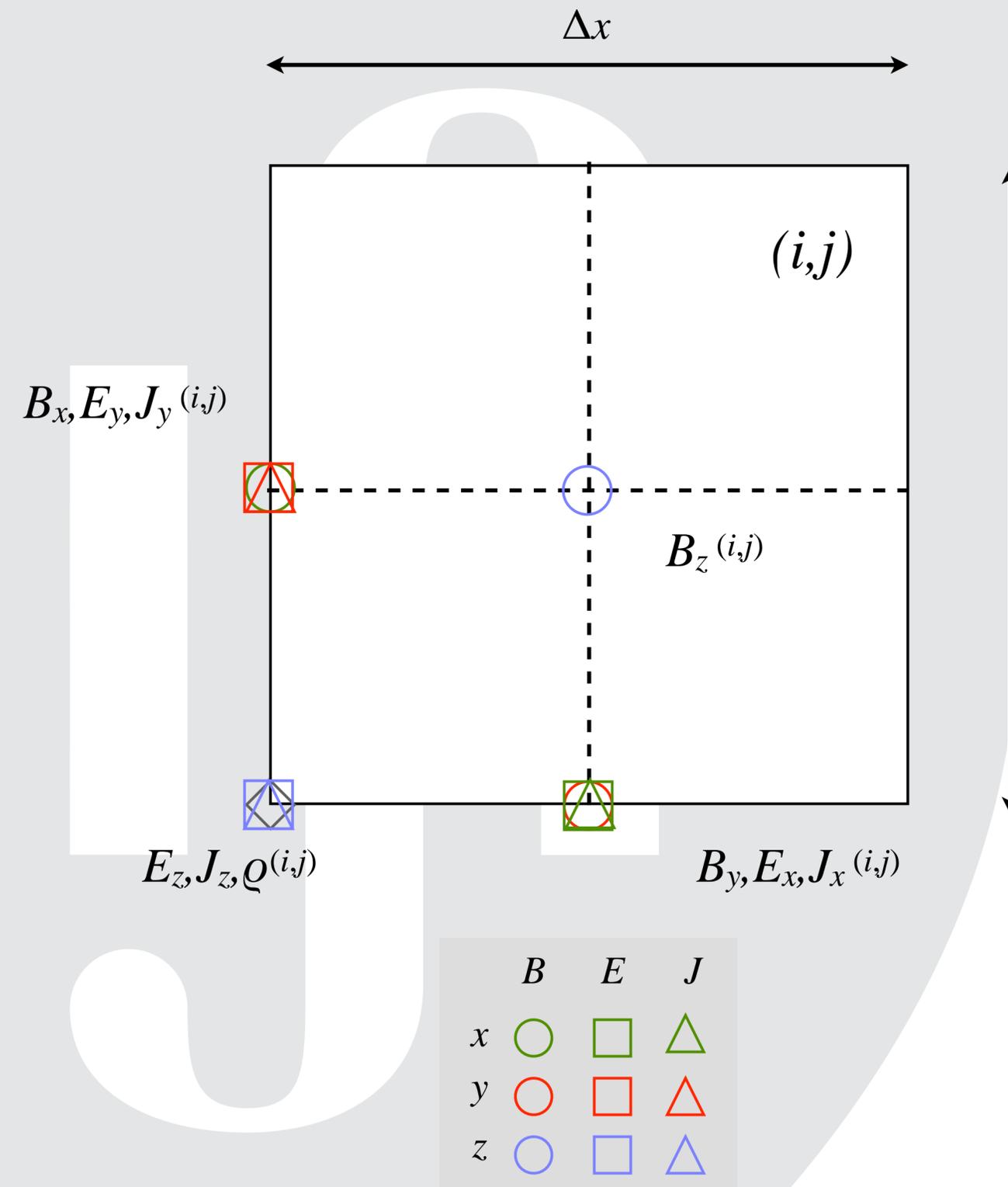
- **E** is defined at times  $t^n$

- **B** and **j** are defined at times  $t^{n+1/2}$

- **B** is later time centered for use in particle advance

- And also in space:

- Spatial derivatives are also 2<sup>nd</sup> order accurate



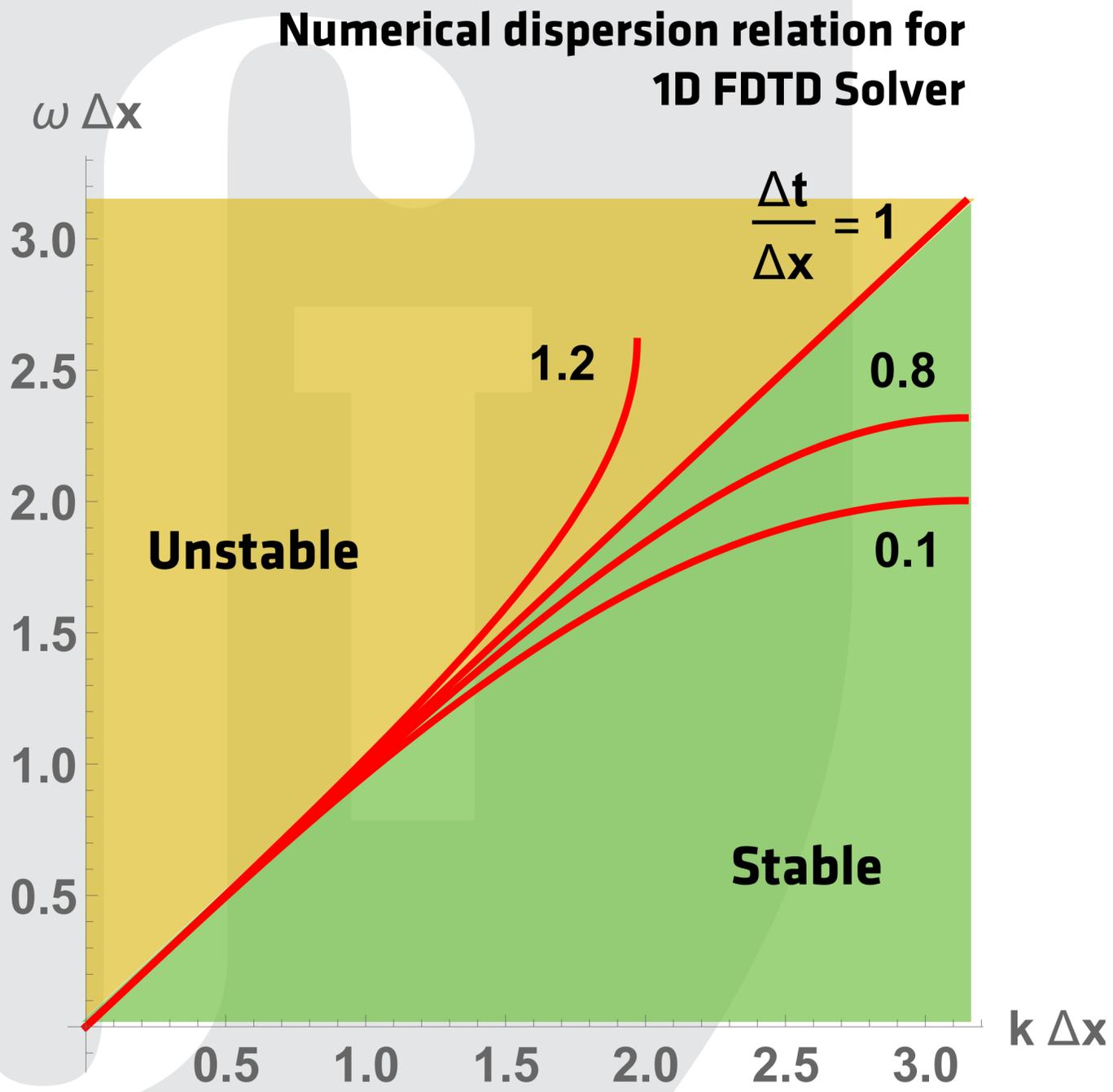
# Choice of time-step

- Choice of time-step is dominated by the FDTD solver (in sim. units):

**1D**      $\Delta t \leq \Delta x$

**2D**      $\Delta t \leq (\Delta x^{-2} + \Delta y^{-2})^{-\frac{1}{2}}$

- If time step is larger than Courant condition the field solver becomes unstable
- If time step is much smaller than courant condition for large  $k$ ,  $v_{ph}$  drops as low as  $2/\pi = 0.637 c$
- Relativistic particles may have  $v > v_0$ 
  - Numerical Cherenkov



# Units and Normalization in ZPIC

- **Careful choice of units and normalization is critical**

- Avoids multiplication by several constants (e.g.  $m_e$ ,  $e$  and  $c$ ) improving performance and numerical accuracy.
- By expressing the simulation quantities in terms of fundamental plasma quantities the results are general and not bound to some specific units we may choose

- **Units and normalization in ZPIC**

- The frequencies are normalized to a normalization frequency,  $\omega_n$ . Time is normalized to  $\omega_n^{-1}$ .
- Proper velocities are normalized to the speed of light,  $c$ . Space is normalized to  $c/\omega_n$ .
- Charge and mass are normalized to the absolute electron charge,  $e$ , and the electron mass,  $m_e$ . The fields are then normalized appropriately.
- The density is normalized to  $\omega_n^2$  (the normalization frequency squared). So if the density is 1 at a given location then the normalization frequency is the plasma frequency at that location.
- If the laser frequency is 1, then the normalization frequency is the laser frequency and the density is normalized to the critical density (for that laser frequency).

## zpic units

$$\mathbf{x}' = \frac{\omega_n}{c} \mathbf{x}$$

$$\mathbf{p}' = \frac{\mathbf{p}}{m_{sp}c} = \frac{\gamma \mathbf{v}}{c} = \frac{\mathbf{u}}{c}$$

$$\mathbf{E}' = e \frac{c/\omega_n}{m_e c^2} \mathbf{E}$$

$$\mathbf{B}' = e \frac{c/\omega_n}{m_e c^2} \mathbf{B}$$

$m_{sp}$  is the mass of the species



# Running ZPIC on your computer

**Harvard Mark I - 1944**

Rear view of Computing Section

# Running ZPIC - Option 1 - compile from source

- **Build from ZPIC source**

- ZPIC itself has no external dependencies, and requires only a C99 compliant C compiler
  - **gcc, clang** and **intel** tested
- The code is open-source and hosted on GitHub
  - <https://github.com/ricardo-fonseca/zpic>

- **Build Python interface**

- The Python interface requires a Python3 installation
- The interface also requires NumPy and Cython packages to be installed
- Just use the Makefile in the python subfolder of the ZPIC distribution
  - This will also compile all of the ZPIC codes

- **Using the Jupyter notebooks**

- Requires a working Jupyter + Python installation
- Launch Jupyter and open one of the example notebooks
- Use either a browser or Visual Studio Code

```
python — fish /Users/zamb/Source/zpic/python — -fish
zamb@zamlap-2 ~/S/z/python> make
python3 setup.py build_ext -if
Compiling em1d.pyx because it changed.
Compiling em2d.pyx because it changed.
Compiling es1d.pyx because it changed.
Compiling em1ds.pyx because it changed.
Compiling em2ds.pyx because it changed.
[1/5] Cythonizing em1d.pyx
[2/5] Cythonizing em1ds.pyx
[3/5] Cythonizing em2d.pyx
[4/5] Cythonizing em2ds.pyx

pes -I/opt/intel/intelpython3/include -I/opt/intel/intelpython3/include -std=c99 -I. -I/opt/intel/intelpython3/include/python3.6m -c ../em2ds/zdf.c -o build/temp.macosx-10.6-x86_64-3.6/./em2ds/zdf.o
/usr/bin/clang -bundle -undefined dynamic_lookup -L/opt/intel/intelpython3/lib -L/opt/intel/intelpython3/lib -arch x86_64 build/temp.macosx-10.6-x86_64-3.6/em2ds.o build/temp.macosx-10.6-x86_64-3.6/./em2ds/charge.o build/temp.macosx-10.6-x86_64-3.6/./em2ds/current.o build/temp.macosx-10.6-x86_64-3.6/./em2ds/emf.o build/temp.macosx-10.6-x86_64-3.6/./em2ds/fft.o build/temp.macosx-10.6-x86_64-3.6/./em2ds/filter.o build/temp.macosx-10.6-x86_64-3.6/./em2ds/grid2d.o build/temp.macosx-10.6-x86_64-3.6/./em2ds/particles.o build/temp.macosx-10.6-x86_64-3.6/./em2ds/random.o build/temp.macosx-10.6-x86_64-3.6/./em2ds/simulation.o build/temp.macosx-10.6-x86_64-3.6/./em2ds/timer.o build/temp.macosx-10.6-x86_64-3.6/./em2ds/zdf.o -L/opt/intel/intelpython3/lib -o /Users/zamb/Source/zpic/python/em2ds.cpython-36m-darwin.so
zamb@zamlap-2 ~/S/z/python>
```

```
python — jupyter /Users/zamb/Source/zpic/python — jupyter-notebook LWFA 2D.ipynb • python
zamb@zamlap-2 ~/S/z/python> jupyter notebook LWFA\ 2D.ipynb
[I 17:13:47.845 NotebookApp] JupyterLab extension loaded from /opt/intel/intelpython3/lib/python3.6/site-packages/jupyterlab
[I 17:13:47.845 NotebookApp] JupyterLab application directory is /opt/intel/intelpython3/share/jupyter/lab
[I 17:13:47.850 NotebookApp] Serving notebooks from local directory: /Users/zamb/Source/zpic/python
[I 17:13:47.850 NotebookApp] 0 active kernels
[I 17:13:47.850 NotebookApp] The Jupyter Notebook is running at:
[I 17:13:47.850 NotebookApp] http://localhost:8888/?token=676ee830df601408ba79a6ecf0c0db560784fc654521b963
[I 17:13:47.850 NotebookApp] Use Control-C to stop this server and shut down all kernels (twice to skip confirmation).
[C 17:13:47.854 NotebookApp]

Copy/paste this URL into your browser when you connect for the first time,
to login with a token:
http://localhost:8888/?token=676ee830df601408ba79a6ecf0c0db560784fc654521b963
[I 17:13:48.855 NotebookApp] Accepting one-time-token-authenticated connection from ::1
[W 17:13:49.797 NotebookApp] 404 GET /static/components/moment/locale/en-gb.js?v=20190314171347 (:::1) 9.97ms referer=http://localhost:8888/notebooks/LWFA%202D.ipynb
[I 17:13:50.348 NotebookApp] Kernel started: 96761370-79fb-4e91-bf01-6c6f0143cea5
[I 17:13:51.092 NotebookApp] Adapting to protocol v5.1 for kernel 96761370-79fb-4e91-bf01-6c6f0143cea5

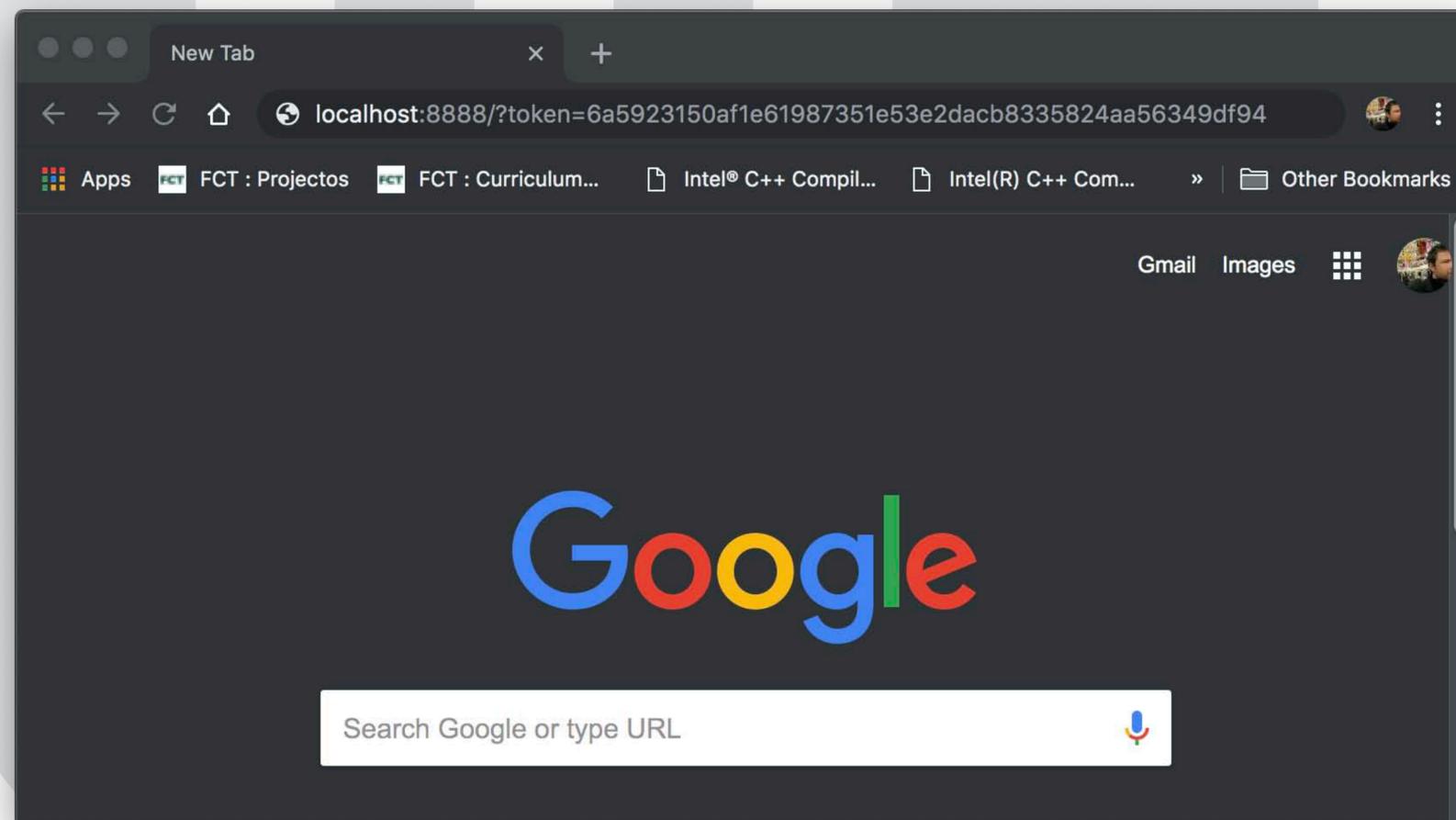
```

# Running ZPIC - Option 2 - use a Docker container

- **Install Docker desktop on your computer**
  - Available for free at:
    - <https://www.docker.com/products/docker-desktop>
- **Run the ZPIC image**
  - The ZPIC container image is hosted on DockerHub
  - Open a terminal window and type the following command
    - `> docker run -p 8888:8888 -t zamb/zpic`
  - The first time you do it, it will download the ZPIC container image. This can take a little time.
- **Open a web browser on your computer and point it to the appropriate port**
  - Type in the following as the address
    - `localhost:8888/?token=[TOKEN]`
  - Get the [TOKEN] value from the output of the docker run command
  - The port number must match the docker run command

```
zamb@zamlap-2 ~> docker run -p 8888:8888 -t --rm zamb/zpic
Executing the command: jupyter notebook
[I 17:06:34.455 NotebookApp] Writing notebook server cookie secret to /home/jovyan/.local/share/jupyter/runtime/notebook_cookie_secret
[I 17:06:34.668 NotebookApp] JupyterLab extension loaded from /opt/conda/lib/python3.7/site-packages/jupyterlab
[I 17:06:34.668 NotebookApp] JupyterLab application directory is /opt/conda/share/jupyter/lab
[I 17:06:34.670 NotebookApp] Serving notebooks from local directory: /home/jovyan
[I 17:06:34.670 NotebookApp] The Jupyter Notebook is running at:
[I 17:06:34.670 NotebookApp] http://(d02798c226cc or 127.0.0.1):8888/?token=0dd946005de0e6db9083ca039ea66faffd24cd51bdd8d55d
[I 17:06:34.671 NotebookApp] Use Control-C to stop this server and shut down all kernels (twice to skip confirmation).
[C 17:06:34.671 NotebookApp]

Copy/paste this URL into your browser when you connect for the first time,
to login with a token:
http://(d02798c226cc or 127.0.0.1):8888/?token=0dd946005de0e6db9083ca039ea66faffd24cd51bdd8d55d
```



- **Option 1 - Compile from source**

- i. Compile the code

- ii. Launch the Jupyter notebook from the source folder:

```
> jupyter notebook LWFA1D.ipynb
```

- **Option 2 - Use a Docker Container**

- i. Install Docker

- ii. Launch the zpic container

```
> docker run -p 8888:8888 -t -v $PWD:/home/jovyan/work zamb/zpic
```

- This mounts the directory **\$PWD** on the directory **work** on your container so you can save changes to the existing notebooks or create new ones

# Using ZPIC Notebooks

- **Jupyter notebooks**

- Similar to Mathematica notebooks but for Python
- Run in a web browser
- Organized in a sequence of cells
- Each cell can contain Python code or annotations

- **The code is runs inside the notebook**

- Initialize the simulation
- Run to specified time
- Access simulation data directly to visualize output
- Several examples provided

- **Saving simulation output not necessary**

- Example simulations run in ~ 1 minute
- Visualize results in the notebook
- Interactively modify simulation parameters
- If required (e.g. for longer simulations) the code can save simulation results to disk
  - Files are saved in the ZDF format
  - a Python module is provided to read these files

The screenshot shows a Jupyter notebook interface for 'R-L Waves'. The code cell contains the following Python code:

```
w = np.linspace(1e-6, wC - 1e-6, num = 512)
k = w * np.sqrt( 1.0 - 1.0/(w**2 * (1-Bx0(w) ) ) )
plt.plot( k, w, label = "R-wave")

plt.ylim(0,12)
plt.xlim(0,12)
plt.xlabel("$k$ [$\omega_n/c$]")
plt.ylabel("$\omega$ [$\omega_n$]")
plt.title("R/L-waves dispersion relation")

plt.legend()

plt.show()
```

Below the code, a plot titled 'R/L-waves dispersion relation' is displayed. The x-axis is labeled  $k [\omega/c]$  and the y-axis is labeled  $\omega [\omega_n]$ . The plot shows three curves: a blue line for 'L-wave' and two red lines for 'R-wave'. The L-wave curve is a straight line starting from the origin. The R-wave curves are slightly curved and start at a non-zero value on the y-axis.

The screenshot shows a Jupyter notebook interface for 'LWFA 2D'. The code cell contains the following Python code:

```
In [3]: import matplotlib.pyplot as plt

range = [[0, sim.box[0]], [0, sim.box[1]]]

plt.imshow( sim.emf.Ex, interpolation = 'bilinear', origin = 'lower',
            extent = ( range[0][0], range[0][1], range[1][0], range[1][1] ),
            aspect = 'auto', cmap = 'jet')

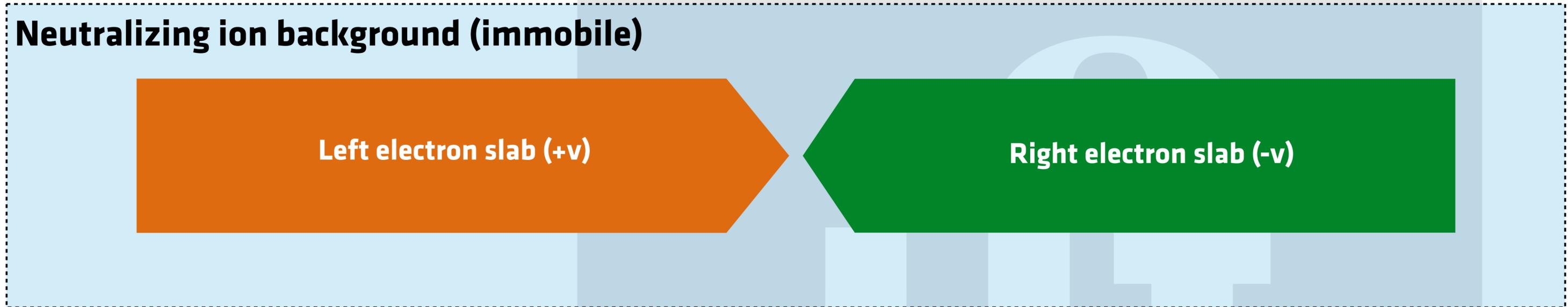
plt.colorbar().set_label('$E_1$')
plt.xlabel("$x_1$")
plt.ylabel("$x_2$")
plt.title("Accelerating Field\n t = {:g}".format(sim.t))

plt.show()
```

Below the code, a heatmap titled 'Accelerating Field' is displayed. The title also includes 't = 22.008'. The x-axis is labeled  $x_1$  and the y-axis is labeled  $x_2$ . The plot shows a grid of colored regions representing the electric field. A color bar on the right indicates the field strength, ranging from -0.2 to 0.2. The plot shows alternating positive (red/yellow) and negative (blue) regions.

A 3D visualization of a laser wakefield acceleration (LWFA) simulation. The image shows a central laser pulse (a small, multi-colored sphere) moving through a plasma channel. Numerous particle trajectories are shown as thin, white lines radiating from the laser pulse, illustrating the acceleration process. The background is dark, with some green and blue highlights representing the plasma density or field distribution.

# Running ZPIC simulations



An Initial seed (e.g. from thermal motions) excites an electrostatic plasma wave



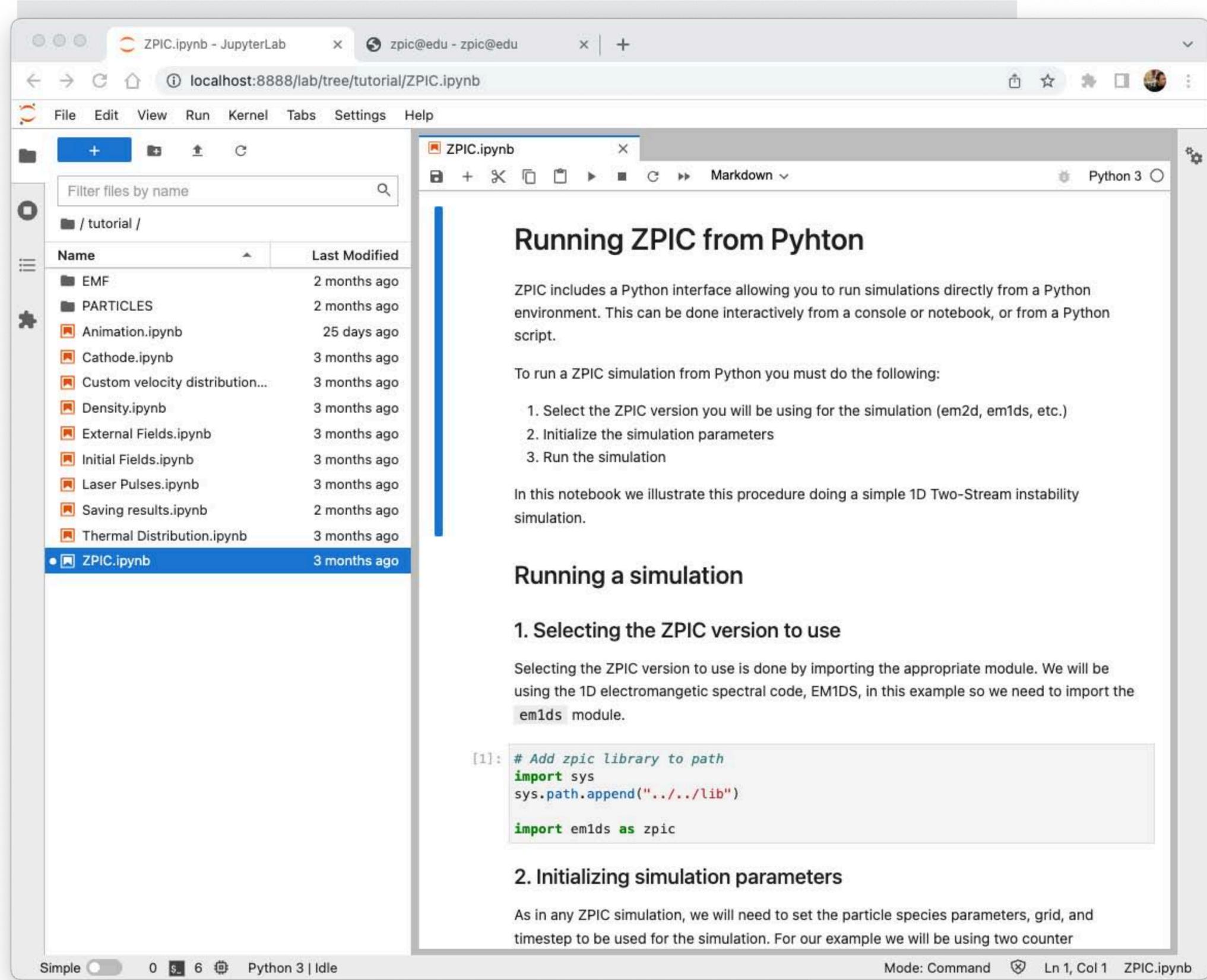
Plasma wave modulates slab electrons



Modulated slab electrons increase amplitude of plasma wave

# Access the ZPIC example notebook

- **Open the ZPIC example notebook**
  - Available at:
    - tutorial/ZPIC.ipynb
- **Run the simulation**
  - Each cell in the notebook can be run by pressing “shift+Enter” or by clicking the “▶” icon.
  - The first cell initializes the simulation
  - The second cell runs the simulation
- **Visualize the results**
  - Additional cells in the notebook allow you to visualize simulation results
  - Just execute any of the visualization cells after running the simulation



# Initializing the Simulation

- **Initializing a ZPIC simulation requires**
  - Selecting the code version
  - Setting up the particle species (sets of particles)
    - The number of species is arbitrary
  - Setting up the simulation
    - Grid / Box size
    - Time step
    - Add species
  
- **Additional (optional) steps**
  - Adding laser pulses
  - Setting up a moving window simulation
  - *We'll look into this later...*

**Code**

```
# Add zpic library to path
import sys
sys.path.append("../../lib")

# Selects EM1DS (EM 1D Spectral) code
import em1ds as zpic
```

**Particles**

```
m_q = -1.0 # mass over charge ratio, in normalized units
           # (1.0 would be a positron)
ppc = 500 # number of particles per cell
ufl = [0.4, 0.0, 0.0] # fluid momenta
uth = [0.001, 0.001, 0.001] # thermal momenta

# Right going electron species
right = zpic.Species( "right", m_q, ppc, ufl = ufl, uth = uth )

# Left going electron species
ufl[0] = -ufl[0]
left = zpic.Species( "left", m_q, ppc, ufl = ufl, uth = uth )
```

**Simulation**

```
import numpy as np

nx = 120 # Number of grid cells
box = 4 * np.pi # Simulation box size
dt = 0.1 # Time step

# Simulation object
sim = zpic.Simulation( nx, box, dt, species = [right, left] )
```

# Running the simulation

- **We can now run our simulation.**
  - The simplest way is to use the `run()` method
  - This will advance the simulation up to the specified time
  
- **This method can be called multiple times**
  - The simulation will still be active when the command completes
  - We can keep calling the same method to further advance the simulation time
  - This allows us to check the evolution of the various results at different time steps.
  
- **The simulation results can be accessed directly**
  - There is no need to store the simulation results to disk
    - *Although this is possible for very long runs*
  - All simulation data is exposed as members of the `Simulation` object used

```

Run # Run up to t = 45 (in simulation units)
    sim.run(45.0)
  
```

# Accessing EM fields and Current density

- **This data is available as properties of the `sim.emf` and `sim.current` objects**
  - Electric field
    - `sim.emf.E[x|y|z]`
  - Magnetic field
    - `sim.emf.B[x|y|z]`
  - Electric current
    - `sim.current.J[x|y|z]`
- **Each of these properties is available as a NumPy array**
  - The array dimensions are the same as the simulation grid
- **Data can be plotted using any Python tool**
  - Matplotlib works fine

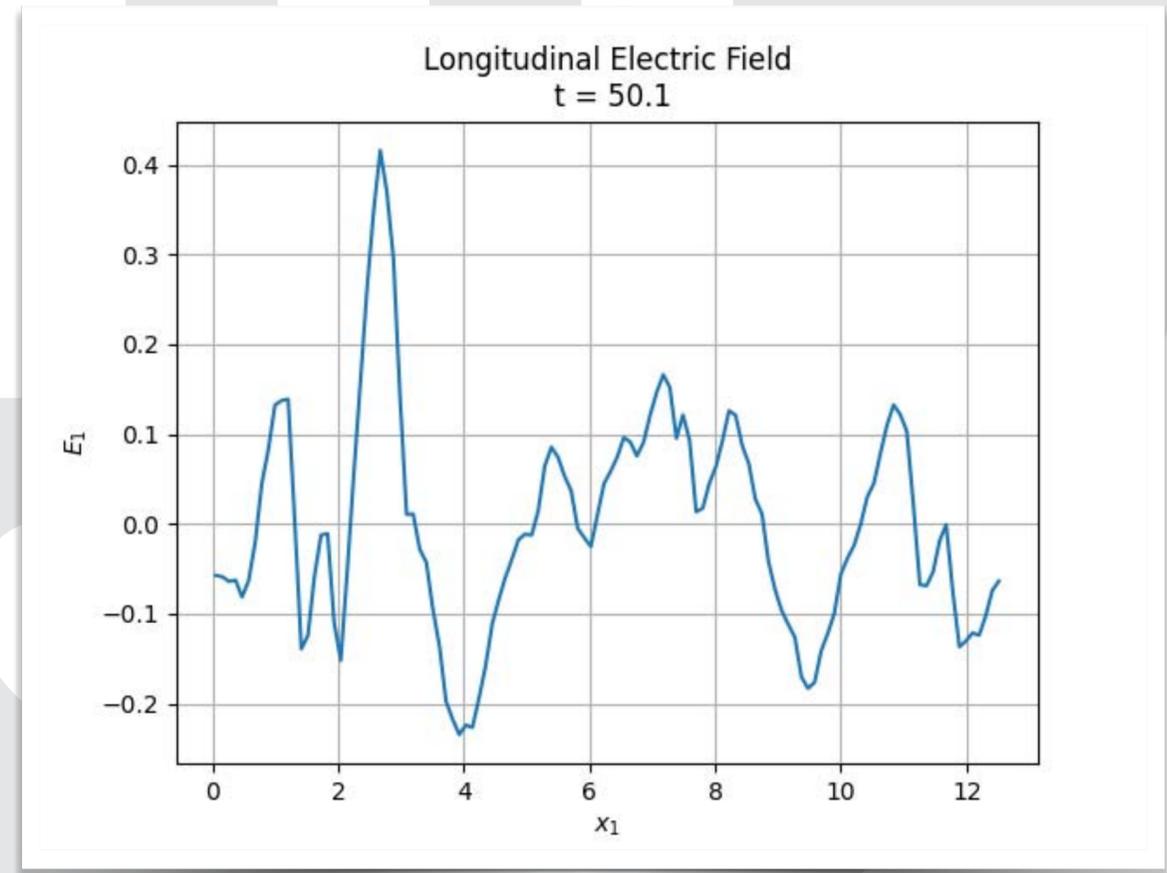
Plot  $E_x$  field

```
import matplotlib.pyplot as plt

# Plot field values at the center of the cells
xmin = sim.dx/2
xmax = sim.box - sim.dx/2

plt.plot(np.linspace(xmin, xmax, num = sim.nx), sim.emf.Ex )
plt.xlabel("$x_1$")
plt.ylabel("$E_1$")
plt.title("Longitudinal Electric Field\n t = {:g}".format(sim.t))
plt.grid(True)

plt.show()
```



# Accessing particle data

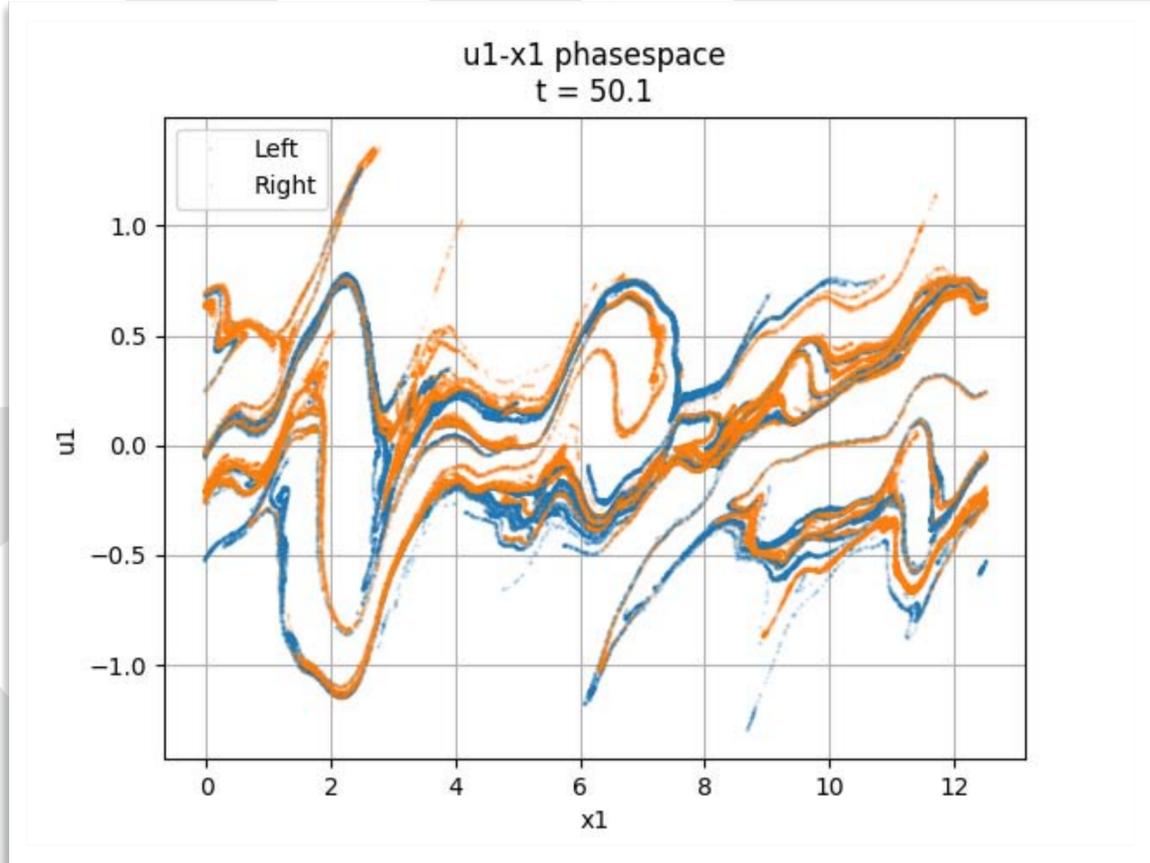
- **Particle data is available using the particles property of each species object**
  - This will be a NumPy array of structures containing
    - ix - the particle cell
    - x - the particle position inside the cell
    - ux, uy, uz - particle generalized velocities
  
- **These can be easily used to produce a phase space plot for the simulation**
  - Note that we have to convert the cell index / position to simulation position

Plot  $u_x-x$

```
import matplotlib.pyplot as plt

# Simple function to convert particle positions
x = lambda s : (s.particles['ix'] + s.particles['x']) * s.dx

plt.plot(x(left), left.particles['ux'], '.', ms=1,alpha=0.2, label = "Left")
plt.plot(x(right), right.particles['ux'], '.', ms=1,alpha=0.2, label = "Right")
plt.xlabel("x1")
plt.ylabel("u1")
plt.title("u1-x1 phasespace\n t = {:g}".format(sim.t))
plt.legend()
plt.grid(True)
plt.show()
```



- **Charge density is not used by the algorithm**
  - But can be easily generated for each particle species using the `charge()` method
- **Again, it is returned as a NumPy array**
  - The array dimensions are the same as the simulation grid

## Plot Charge Density

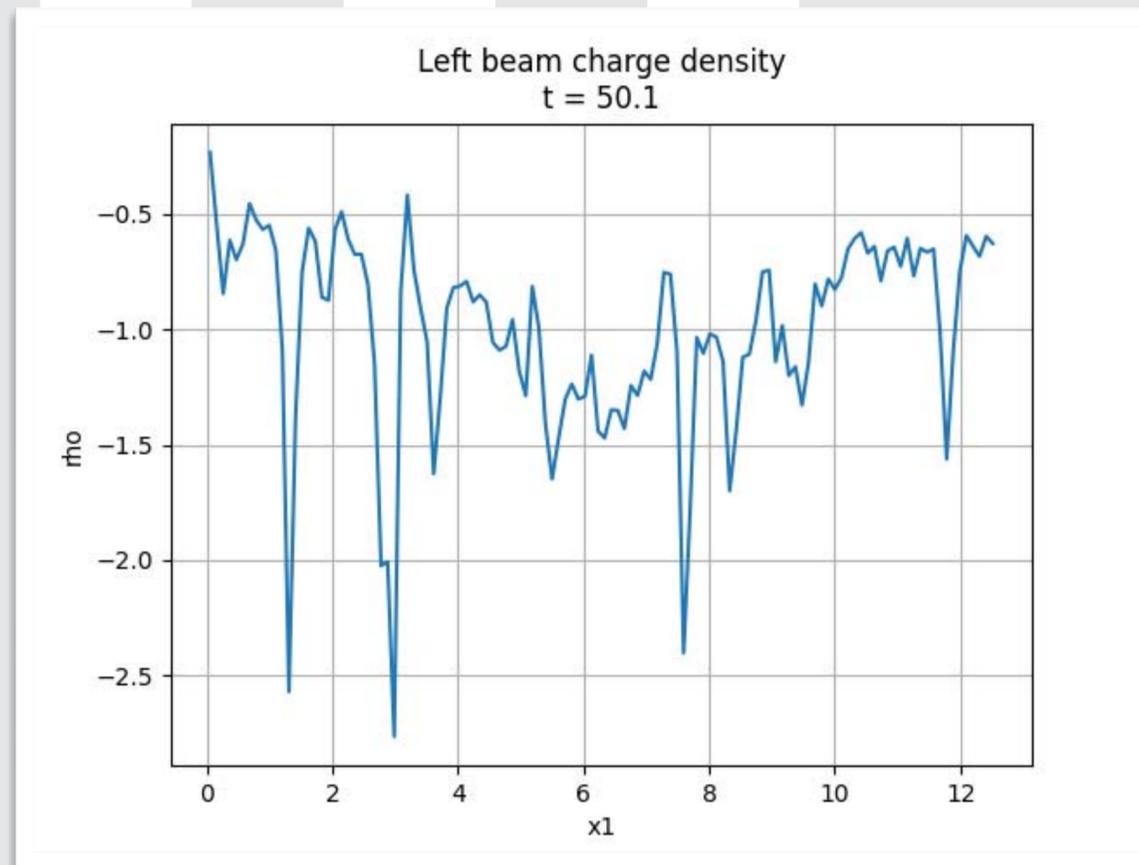
```
import matplotlib.pyplot as plt

charge = left.charge()

xmin = sim.dx/2
xmax = sim.box - sim.dx/2

plt.plot(np.linspace(xmin, xmax, num = sim.nx), left.charge() )
plt.xlabel("x1")
plt.ylabel("rho")
plt.title("Left beam charge density\n t = {:g}".format(sim.t))

plt.grid(True)
plt.show()
```



# Phasespace density

- **ZPIC also includes the possibility of generating 2D phasespace density grids**
  - For each particle species we can use the `phasespace()` method
- **To use this function the user must supply**
  - The variables to be used for each axis
  - The range of values to be used for each axis
  - The size of the phase space grid
- **The result is returned as a NumPy array**
  - The array dimensions will be the size of the phase space grid

## Plot Charge Density

```
import matplotlib.pyplot as plt
import matplotlib.colors as colors

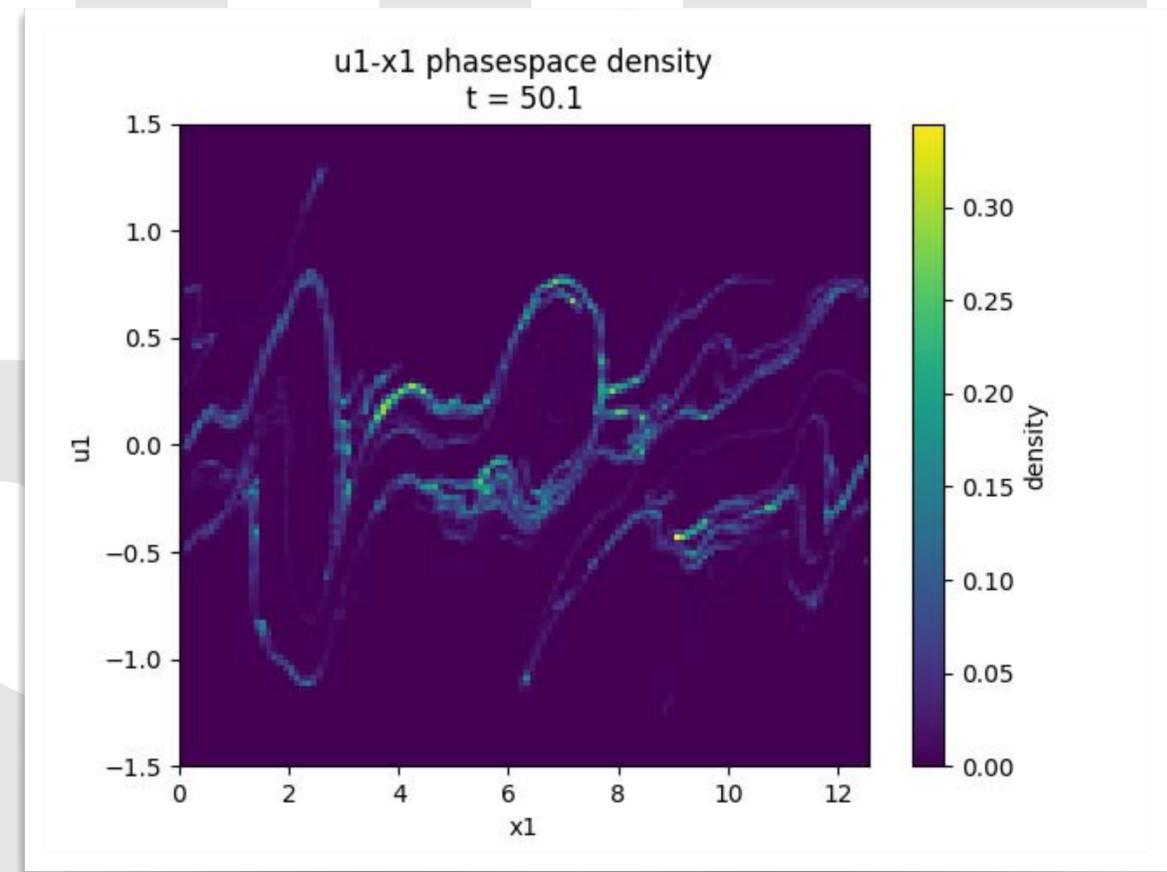
nx    = [120,128]
range = [[0,sim.box],[-1.5,1.5]]

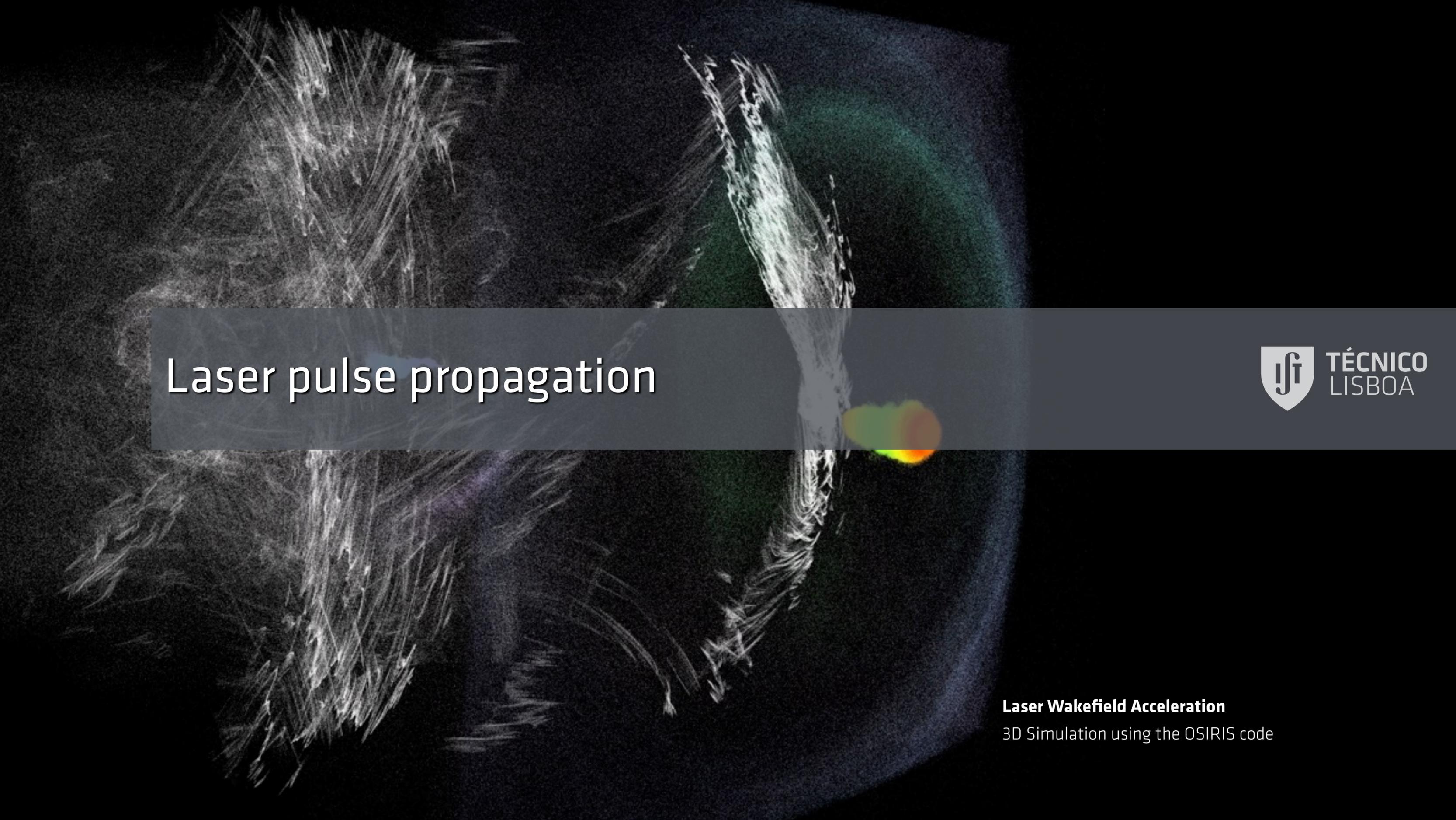
pha = np.abs(left.phasespace( ["x1", "u1"], nx, range ))

plt.imshow( pha, interpolation = 'nearest', origin = 'lower',
            extent = ( range[0][0], range[0][1], range[1][0], range[1][1] ),
            aspect = 'auto')

plt.colorbar().set_label('density')
plt.xlabel("x1")
plt.ylabel("u1")
plt.title("u1-x1 phasespace density\nt = {:g}".format(sim.t))

plt.show()
```



A 3D simulation showing a laser pulse propagating through a plasma channel. The laser pulse is represented by a bright, elongated, and somewhat irregular structure with a color gradient from blue to yellow. It is surrounded by a complex, filamentary structure of plasma filaments. The background is dark, with some faint, diffuse light. The simulation is presented in a perspective view.

# Laser pulse propagation

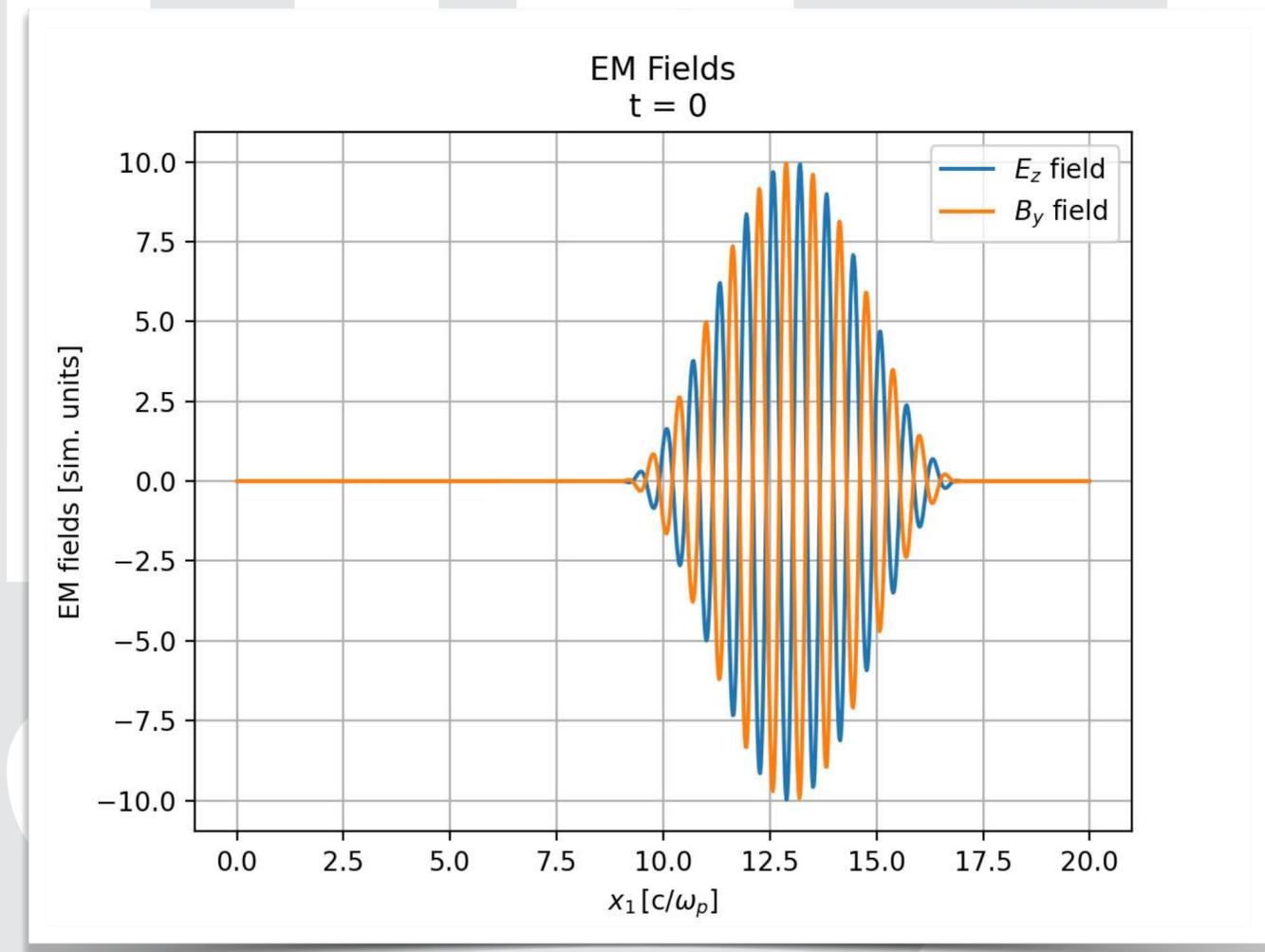
# Laser pulses and moving windows

- **ZPIC includes the ability to launch laser pulses**
  - User defined laser parameters
  - Laser pulses are created at once: the full laser fields are super-imposed (added) on the simulation fields
  - The temporal envelope of the laser pulses is defined as a  $\sin^2$  function
  - Different rise, fall and flat (constant amplitude) times may be selected
- **Laser pulses should be added after creating simulation object**
  - Use the `sim.add_laser()` method
- **2D Laser pulses have additional options**
  - Plane wave / Gaussian beams
  - Gaussian waist, focal plane propagation axis position
- **Check Laser Pulses notebook**
  - Available at:
    - `tutorial/Laser Pulses.ipynb`

```

Laser
# Initialize simulation
sim = em1d.Simulation( nx, box, dt )

# Add laser pulse
sim.add_laser( em1d.Laser( start = 17.0, fwhm = 4.0,
                           a0 = 1.0, omega0 = 10.0,
                           polarization = np.pi/2 ) )
    
```



# Moving simulation window at $c$

- **Most LWFA simulations are not run in a fixed frame**
  - There is a large scale disparity: the simulation needs to resolve both the laser wavelength and the propagation distance
  - This leads to (extremely) large simulations
  - However, most relevant physics happens close behind the laser driver
- **To avoid this issue we can use a moving simulation window**
  - Model a window that includes only the laser pulse and some distance behind it
  - As the simulation progresses the simulation window moves forward at the speed of light
  - Provided nothing happens ahead of the simulation window, relativity ensures we are including all relevant physics
  - Simulation is still run in the lab frame (no Lorentz boost), we just focus on a different region of interest
- **Using a moving window is straightforward**
  - Define simulation parameters the usual way
  - Do `sim.set_moving_window()`

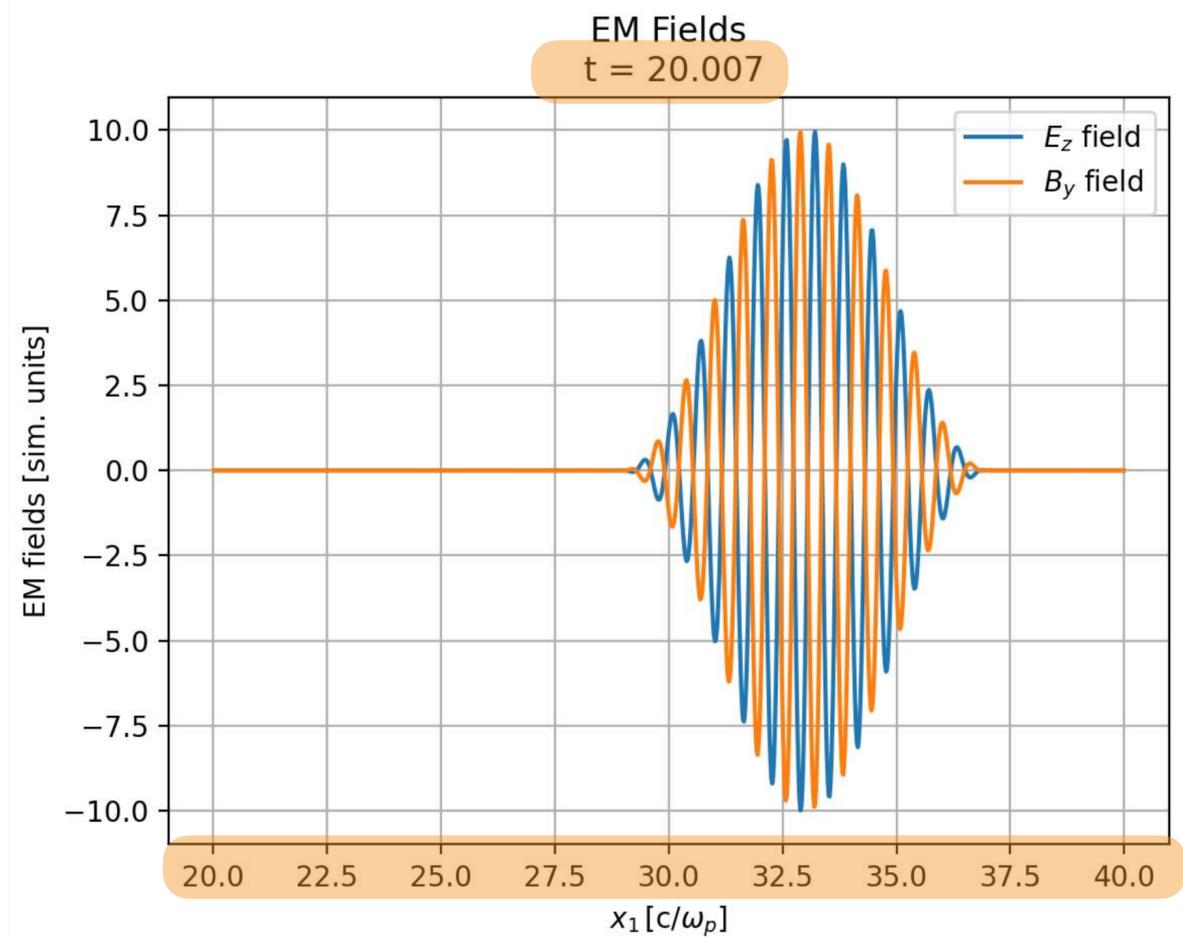
## Moving Window

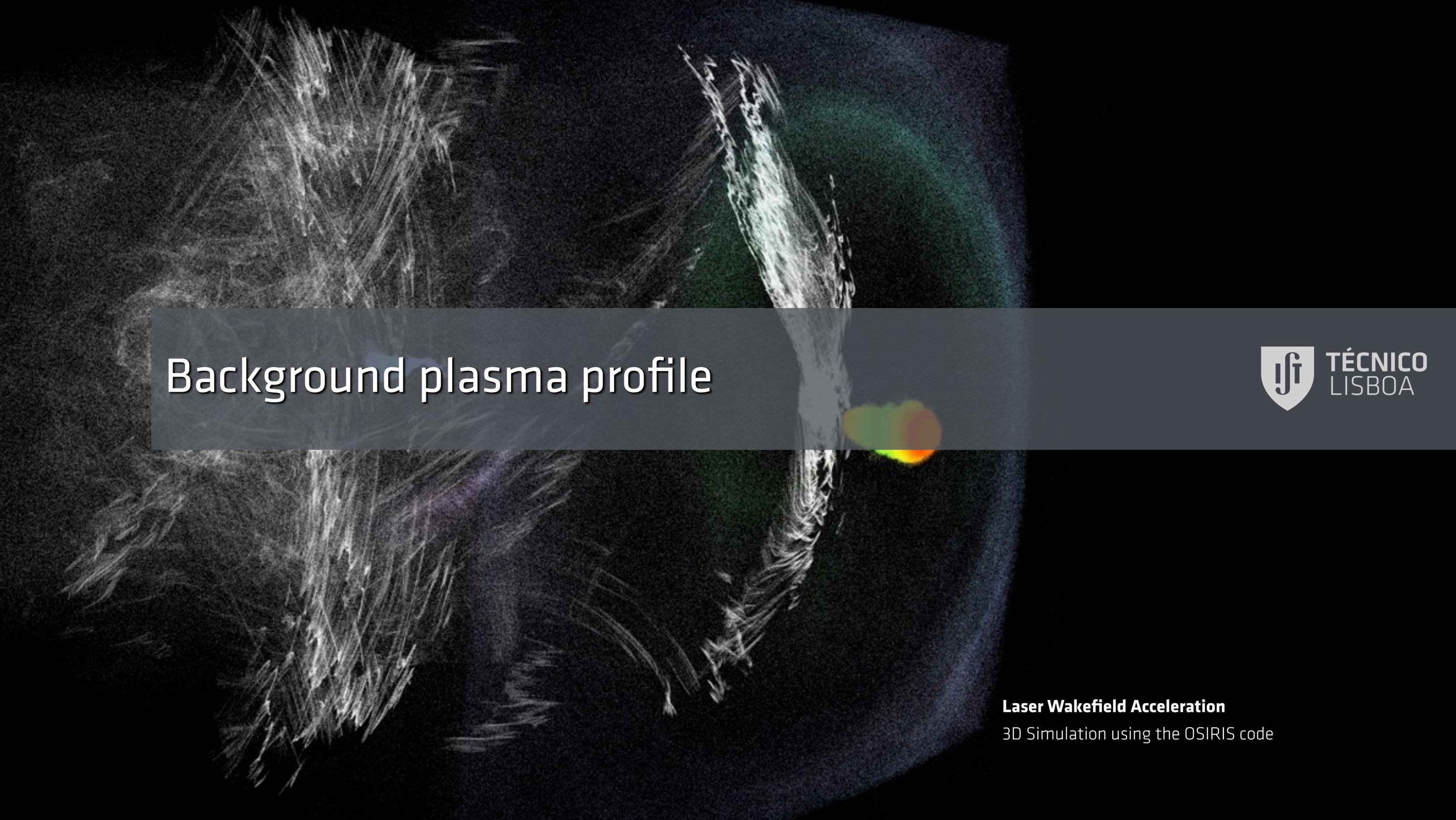
```
# Set moving window
sim.set_moving_window()

# Run the simulation
sim.run( 20.0 )

# Plot field values at the center of the cells
xmin = (sim.n_move - 0.5) * sim.dx
xmax = sim.box + (sim.n_move - 0.5) * sim.dx

# (...)
```



A 3D simulation of a laser wakefield acceleration setup. A laser pulse, represented by a small, bright, multi-colored spot (yellow, green, blue), is shown entering a plasma medium from the right. The plasma is depicted as a dark, textured volume with a central channel. The laser pulse is creating a wakefield, shown as a large, elongated, and textured structure extending to the left. The background is dark, with some faint, glowing structures.

# Background plasma profile

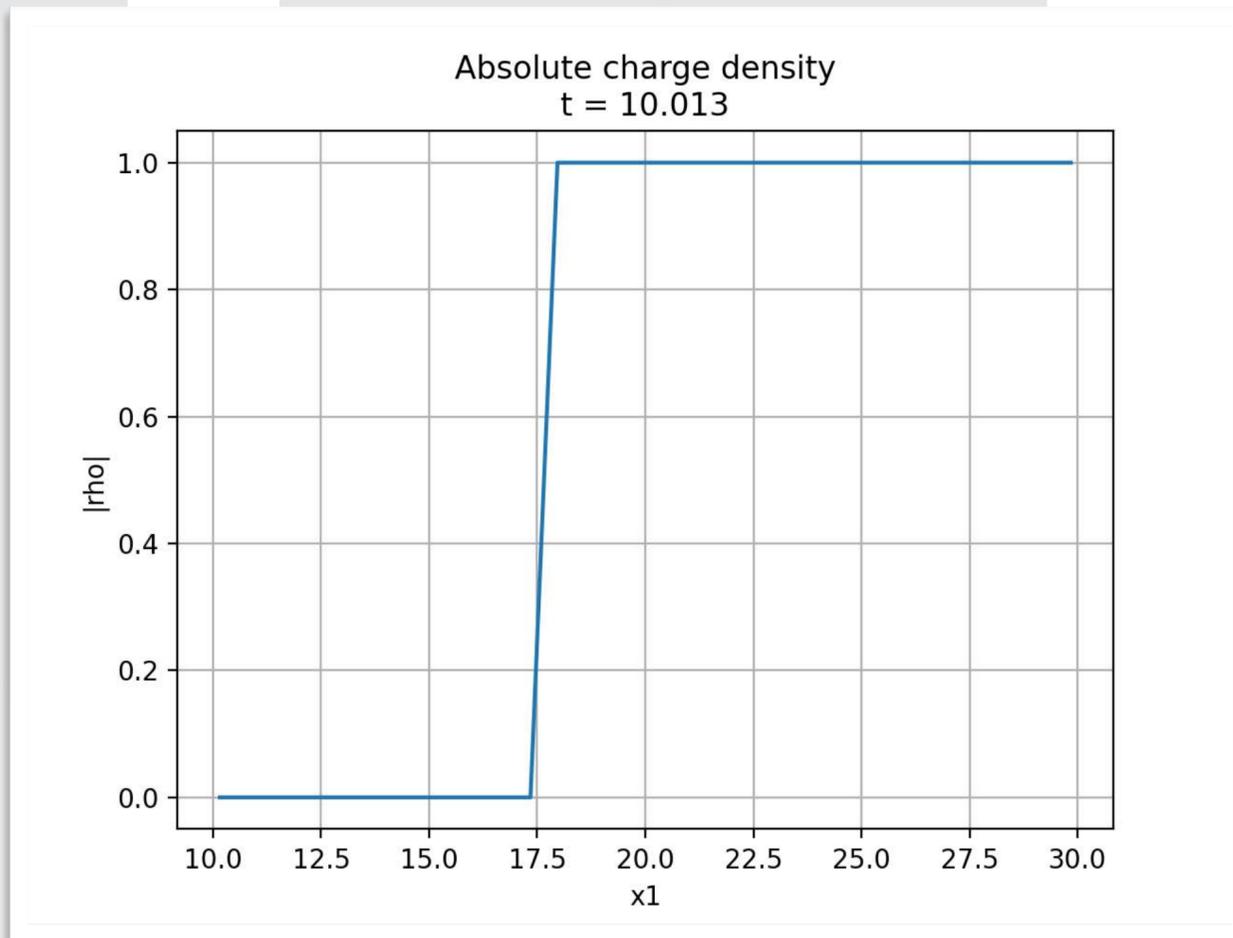
# Plasma density profiles

- **ZPIC uses a fixed charge per particle inside a species**
  - The number of particles per cell option corresponds to the reference density
  - As particles move in the simulation domain, local density will vary
- **By default ZPIC assumes a uniform density profile**
  - The number of particles per cell injected in every cell is constant
  - Different density profiles can be chosen at initialization using the `species.Density` object
- **For accelerators the laser is usually initialized in vacuum before entering some density profile**
  - The most common profile types used are “step”, “slab” and “custom”
  - All work together with moving window
- **Check Density notebook**
  - Available at:
    - [tutorial/Density.ipynb](#)

Step

```
density = em1d.Density( type = "step", start = 17.5 )

# Background plasma
electrons = em1d.Species( "electrons", -1.0, 128,
                          density = density )
```



# Custom density profiles

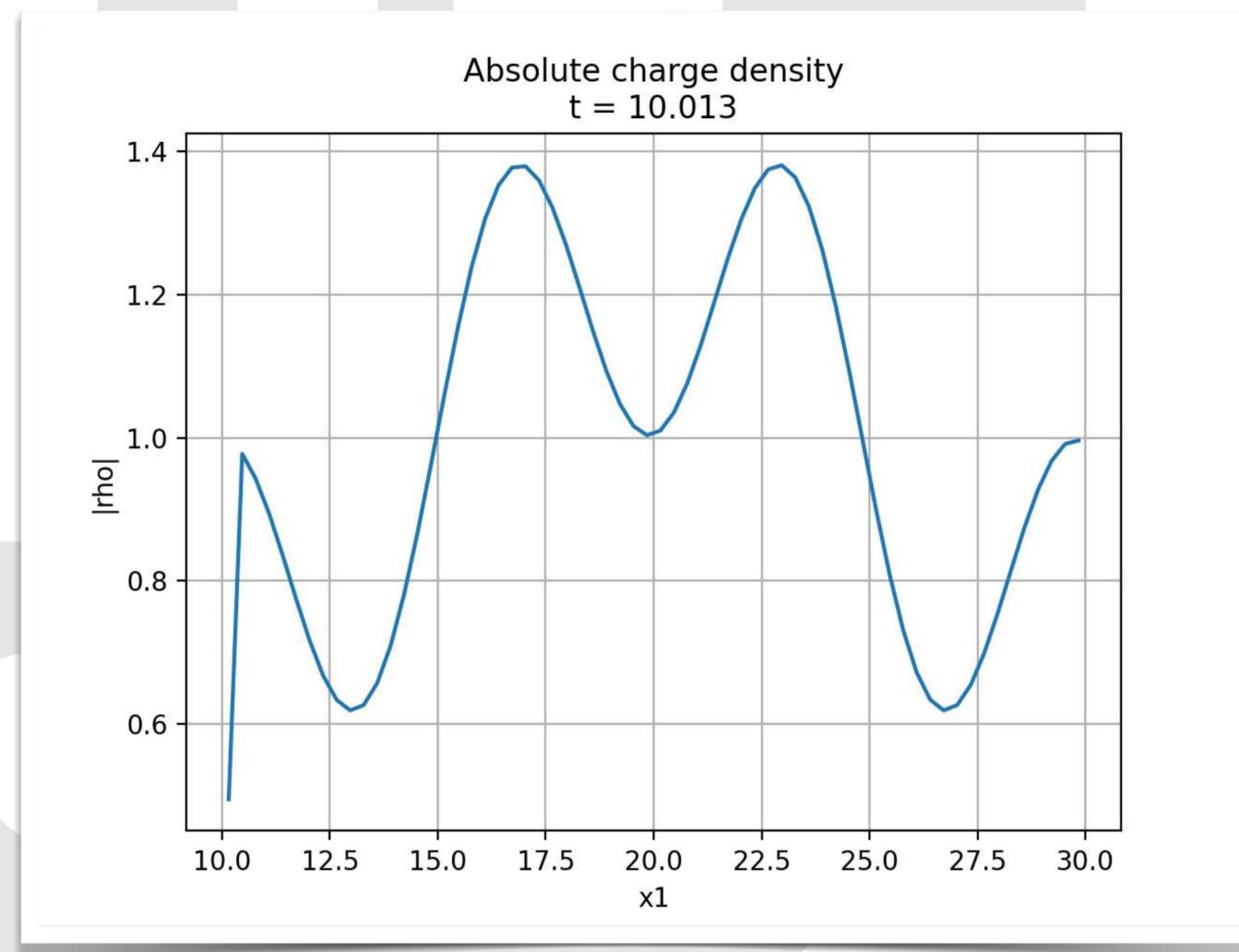
- **ZPIC also allows the use of “custom” density profiles**
  - In this type of profiles the density is defined by a user supplied function
  - This function must take as a single parameter the position (in simulation units) and return the required density for that point
- **This also works in 2D**
  - In this case the density must be defined as the product of 2 separable functions
  - The user must supply a function for the density as a function of  $x$ , and another for the density as a function of  $y$

Custom

```
# Custom density profile
def custom_n0(x):
    return 1.0 + 0.5*np.sin(2*x/np.pi)*np.sin(x/np.pi)

density = em1d.Density( type = "custom", custom = custom_n0 )

# Background plasma
electrons = em1d.Species( "electrons", -1.0, 128, density = density )
```

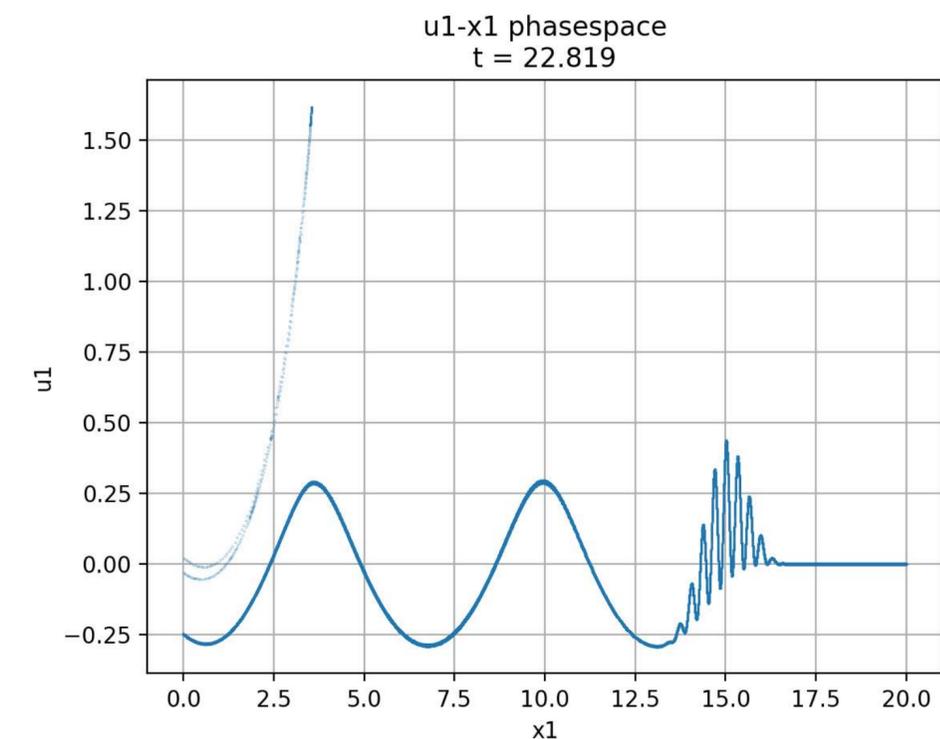
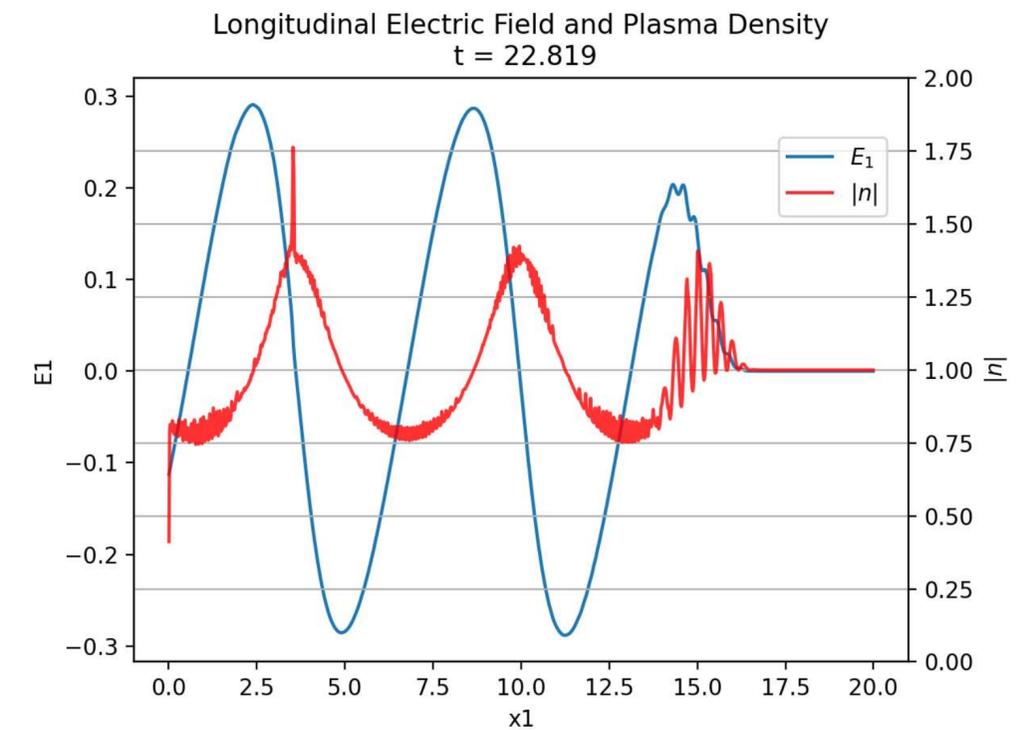


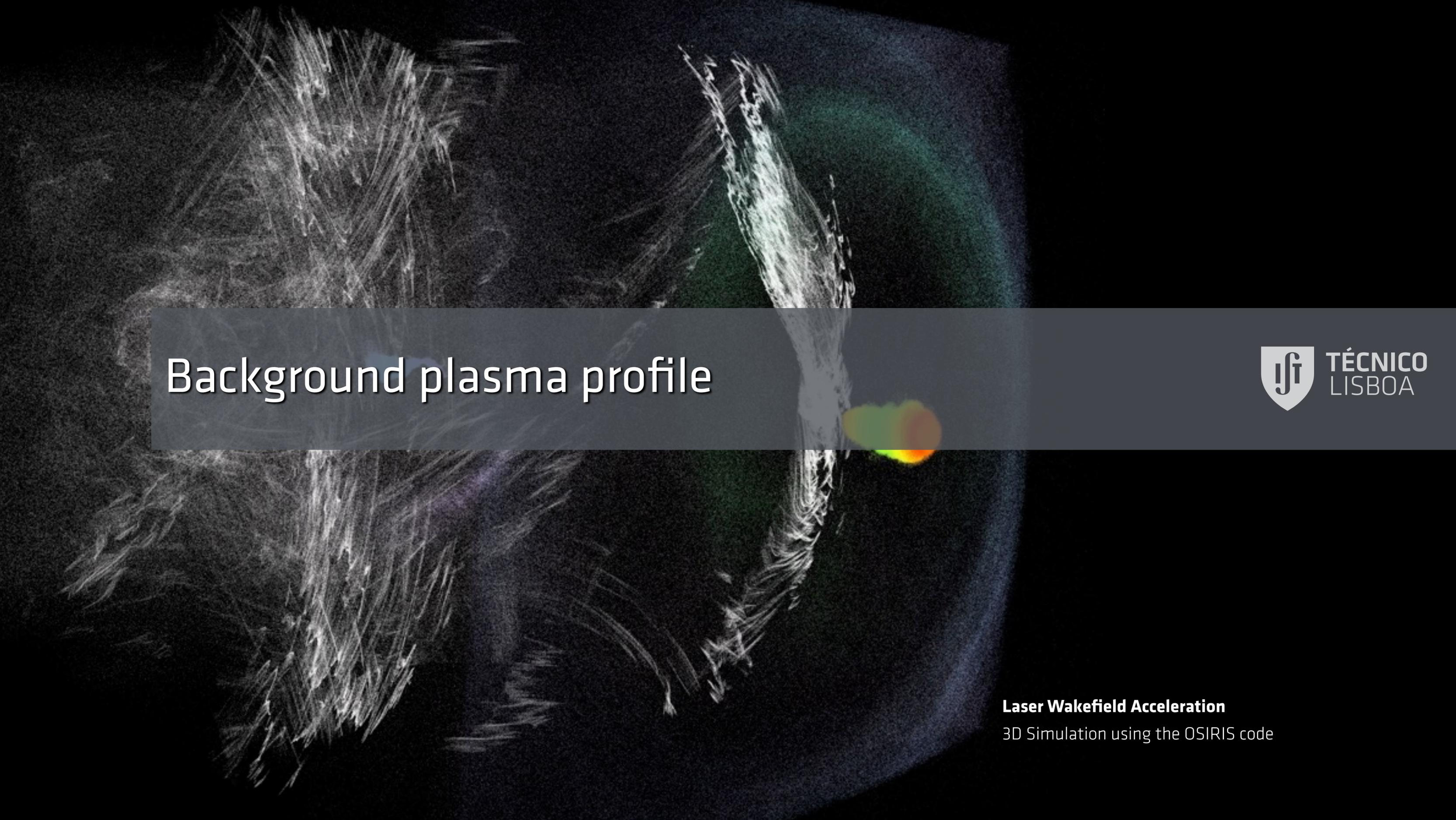


# A first 1D LWFA simulation

# 1D Laser-Wakefield Accelerator

- **Open the 1D LWFA example notebook**
  - Available at:
    - classroom/LWFA 1D.ipynb
- **This notebook presents a simple 1D simulation of a laser wakefield accelerator**
  - Grid: 1000 cells
  - Box size:  $20.0 c/\omega_p$
  - $\Delta t$ :  $0.019 \omega_p^{-1}$
  - Laser frequency:  $10.0 \omega_p$
  - FWHM:  $2.0 \omega_p^{-1}$
  - $a_0$ : 1.0
- **The simulation also uses smoothing (digital filtering)**
  - Keeps noise level low even with small number of particles per cell



A 3D simulation of a laser wakefield acceleration setup. A laser pulse, represented by a small, bright, multi-colored spot (yellow, green, blue), is shown entering a plasma medium from the right. The plasma is depicted as a dark, textured volume with a central channel. The laser pulse is creating a wakefield, shown as a series of bright, elongated structures extending to the left. The background is dark, with some faint, wispy structures on the left side.

# Background plasma profile

- **Understand the fundamentals of PIC simulations**
  - How does the PIC algorithm model kinetic plasma scenarios
  - What are the fundamental parameters on a PIC simulation
- **Get ZPIC up and running on my computer**
  - Either run a Docker image or install it locally
- **Open a ZPIC notebook and run the simulation**
  - Use either the traditional browser interface or VS Code
- **Modify simulation parameters and perform simple data analysis**
  - Explore different scenarios
  - Visualize different quantities
- **Next session**
  - Introduction to laser dynamics and plasma accelerators



**ZPIC website**  
[ricardo-fonseca.github.io/zpic](https://ricardo-fonseca.github.io/zpic)