Opportunities and Challenges for Domain-Specific Languages in Fusion Applications

Alice Koniges

Lawrence Berkeley National Laboratory

Exascale Research Conference

Arlington, VA

October, 2012

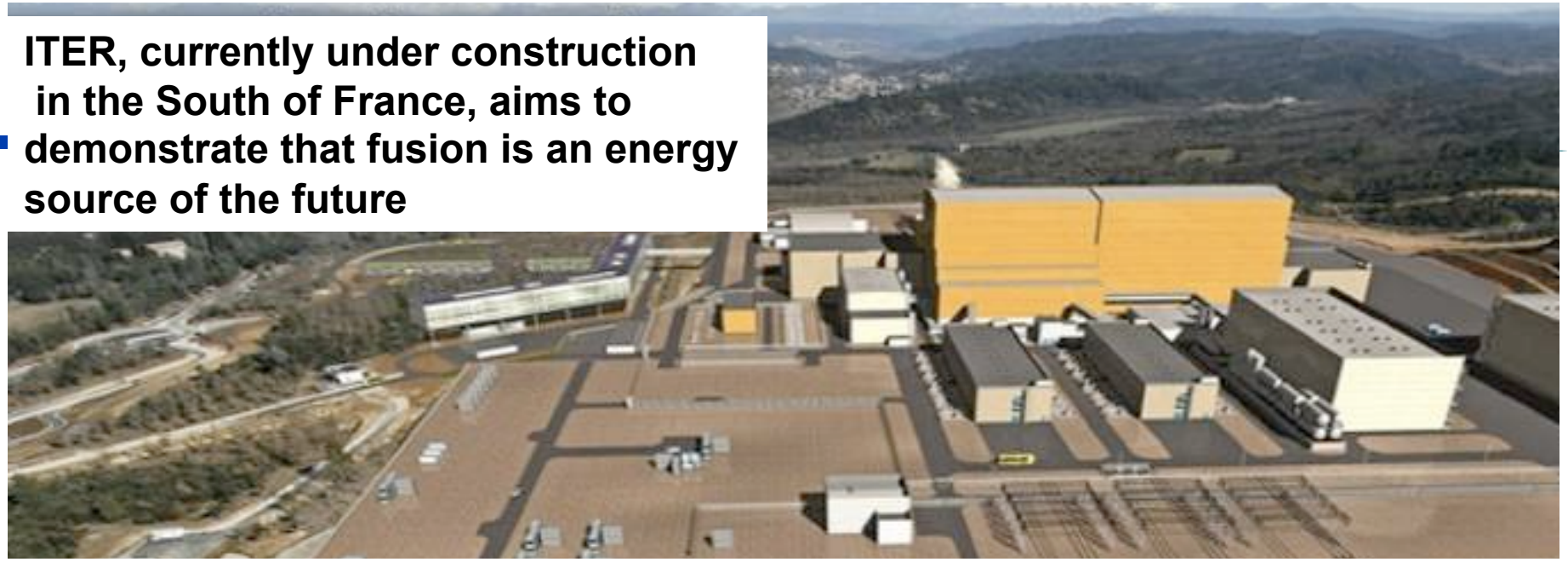
National Energy Research Scientific Computing Center

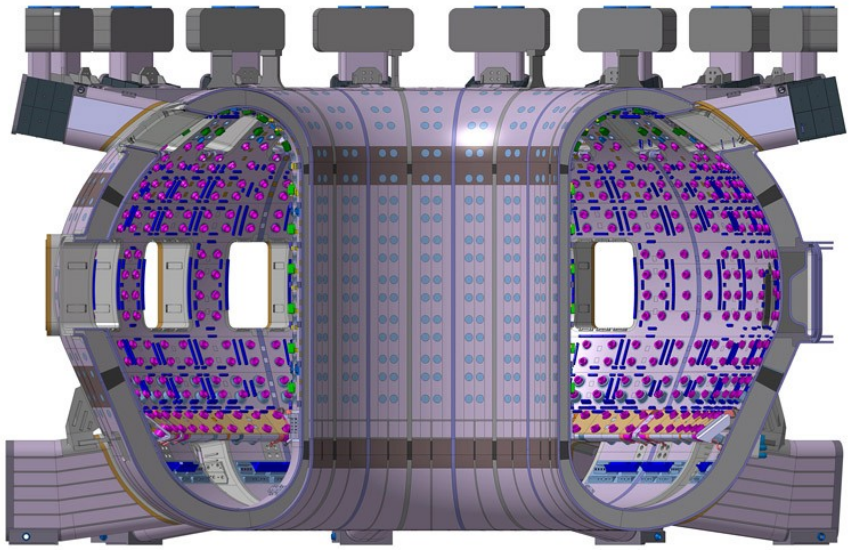
U.S. DEPARTMENT OF ENERGY | Office of Science

A Domain-Specific Language (DSL) for Fusion/Plasma Physics?

- Understanding the field
  - Range of scales
- Opinions in the field
  - Well, maybe a domain-aware library?
  - No thank you, we do fine with MPI, Fortran and our libraries for meshing and solvers
  - If we can design something it can save a lot of duplicative efforts and help use and maintain advanced components
- Possibilities for DSL
  - Low-level, extensions, change MPI calls under hood
  - A high-level construct (abstraction) for meshing or similar operations
  - Orchestrate the DSL to provide abstractions that can be handled by an intermediary (e.g., ROSE-DTEC)
  - Domain specific functionality should be in a human readable form

ITER, currently under construction in the South of France, aims to demonstrate that fusion is an energy source of the future

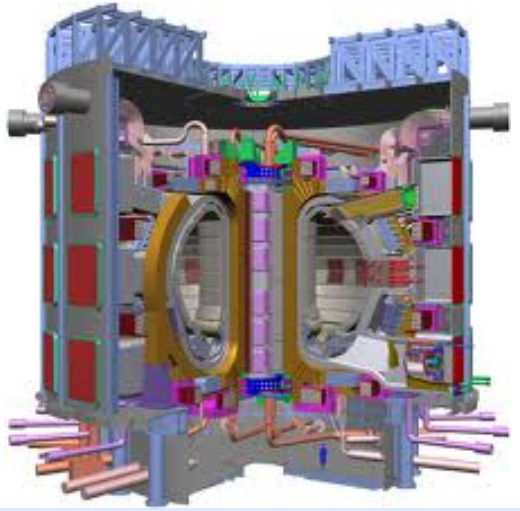




- **Top-to-bottom exascale computer design is essential for efficient design/operation of large-scale experiments**
  - Typical ITER discharge can be estimated at 1M$

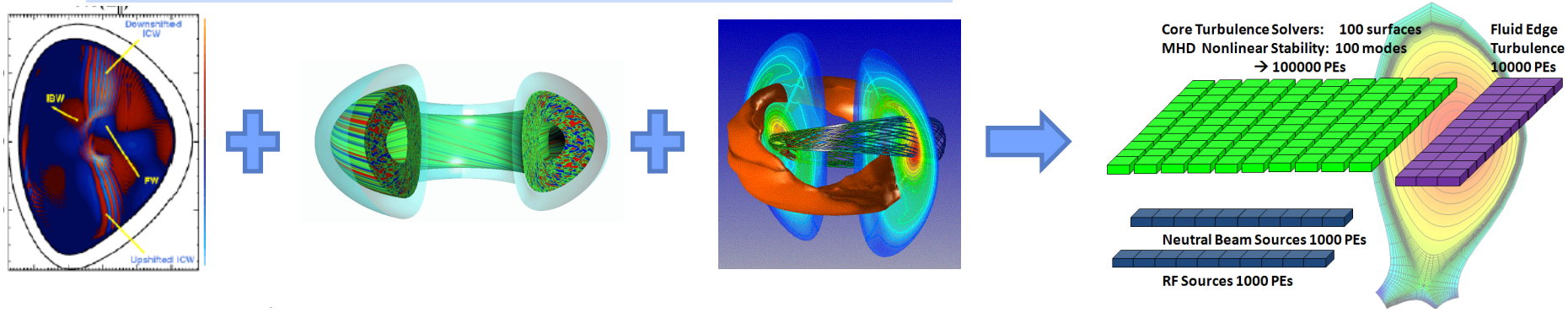
A variety of Fusion apps are required for building and running the ITER

- Fusion program has large suite of petascale applications in use covering many spatial and temporal scales
- The fusion suite of parallel applications brings a wide array of algorithms (implicit, nonlinear fluid, PIC, continuum phase space)



ITER: $10B Reactor

Coupled code set diagram for magnetic fusion



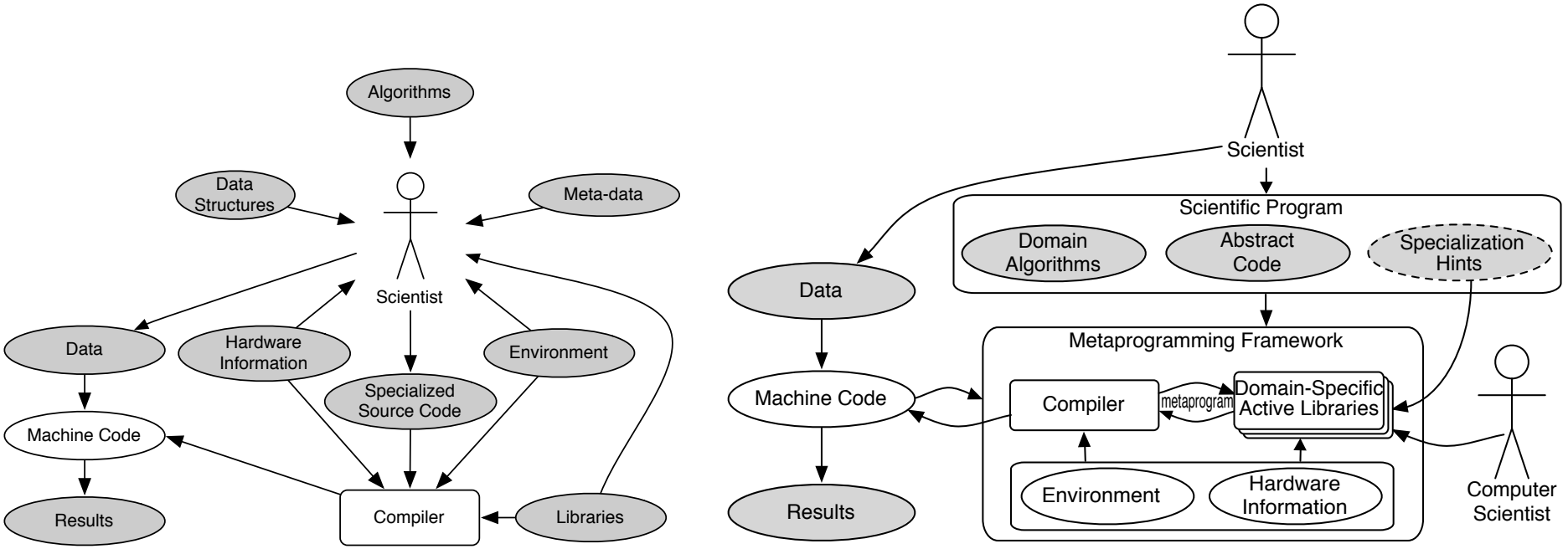
Ω_{ce}^{-1} ω_{LH}^{-1} Ω_{ci}^{-1} τ_A

10^{-10} 10^{-8} 10^{-6} 10^{-4} 10^{-2} 10^{0} 10^{2} 10^{4}

Discharge time ~hour

Time in seconds for full-scale magnetic fusion interactions

Domain Specific Language approaches can benefit the scientist



-Scientist avoids worrying about parallelization, locality, and synchronization
-May really want: “Domain specific concepts” expressible in a variety of languages
-Common set of data abstractions, grid abstractions, functional abstractions

Figures from Xpress Xstack project

PIC as a driver for Domain Specific Languages

- "particle-in-cell" because plasma macro-quantities (number density, current density, etc.) are assigned to simulation particles
- Particles can live anywhere on the domain, but field and macro-quantities are calculated only on the mesh points
- Steps can lead to a domain specific language/concepts
  - Integration of the equations of motion
  - Interpolation of charge and current source terms to the field mesh
  - Computation of the fields on mesh points (field solve)
  - Interpolation of the fields from the mesh to the particle locations
- PIC codes differ from Molecular Dynamics in use of fields on a grid rather than direct binary interactions, goes from N^2 to N
- PIC codes are radically different from standard PDE solver codes and show real promise for the exascale

Specific Components of a DSL might include

- Data Structures/Abstractions
  - Lagrangian particles: x, y, z, Vx, Vy, Vz, q, m, etc.
  - Eulerian fields and sources: Jx, Jy, Jz, Ex, Ey, Ez,
  - Bx, By, Bz on grids for electromagnetics;
  - Rho, Phi (or V) for electrostatics
- Goal – don't care where the particles live, e.g., in terms of the parallel decomposition. Want to hide this from application programmer
- Methods/Functional Abstractions
  - Push particles
  - Deposit (scatter) charge or currents from particles onto grid(s)
  - Solve fields (Can we put this into a library call?)
  - Gather forces from grids onto particles.

Questions for our DSL

- Data abstraction -- Need to ask what do you want to do with the data –e.g., to push the particles. User should not care about where the particles live, what processor, etc. and need to conserve movement of data between procs
- Grid abstraction -- fields have to live on a grid. How much of grid in memory, and how is it distributed? What do you need to pull from it? Maybe don't want to replicate the grid on all procs?
- Functional abstraction --can you generate the move for a variety of different problems and let it (DSL combined with compiler) generate the code for this?

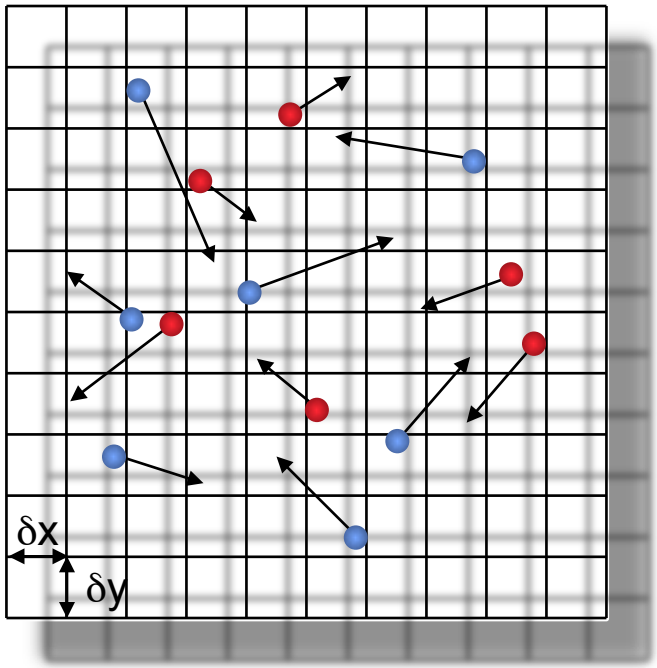
Why can it be difficult to define a PIC DSL

- **Difference in layout of Lagrangian and/or Eulerian quantities in memory**
  - **not a hard barrier per say as layers of translations (copy) between data structure can be added, but usually at the expense of runtime efficiency**
- **Legacy**
- **Competition**

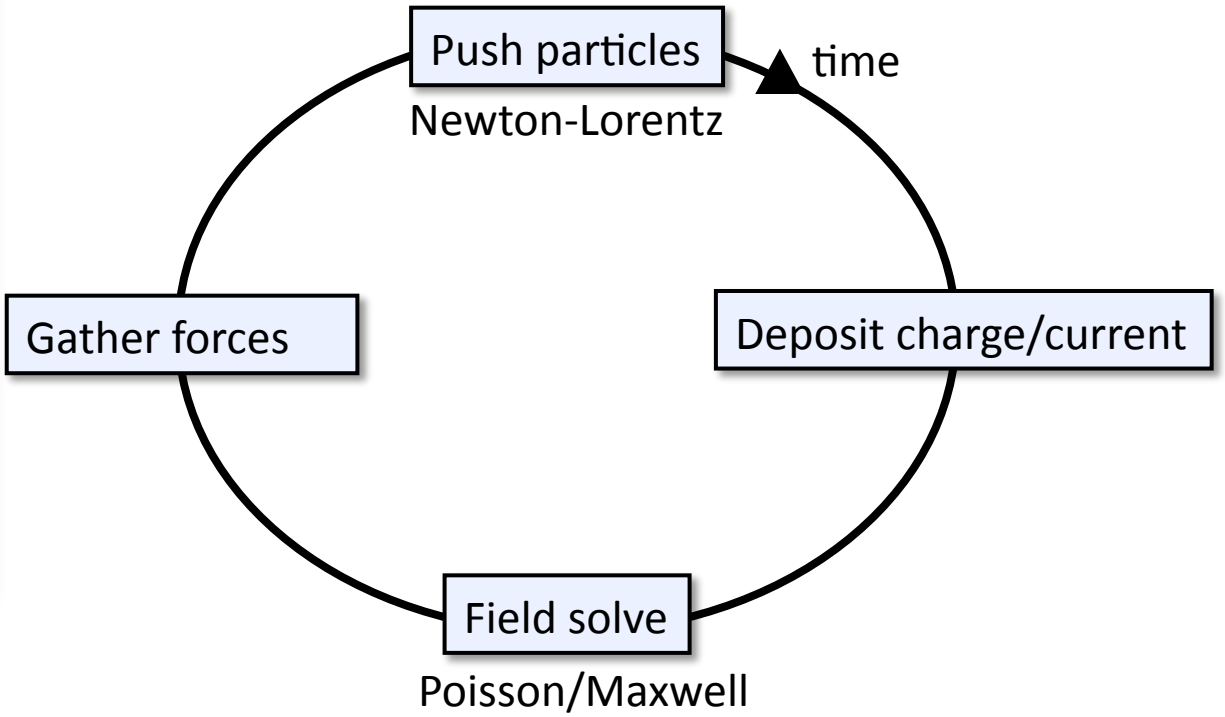
More evolved features like irregular gridding, AMR, complex particle pushers, deposition schemes, or field solvers, call for more sharing as a smaller fraction of developers can effectively maintain such codes.

basic data structures and operations are fairly simple and thus easily reproducible

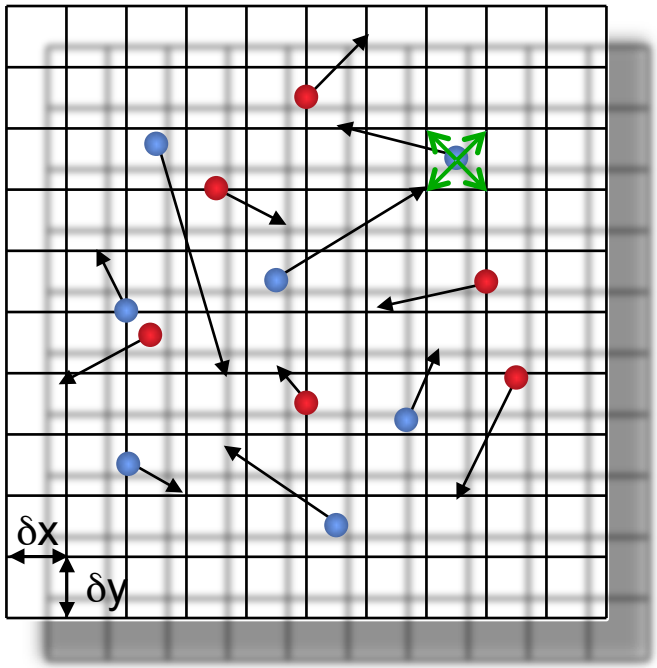
Particle-In-Cell workflow



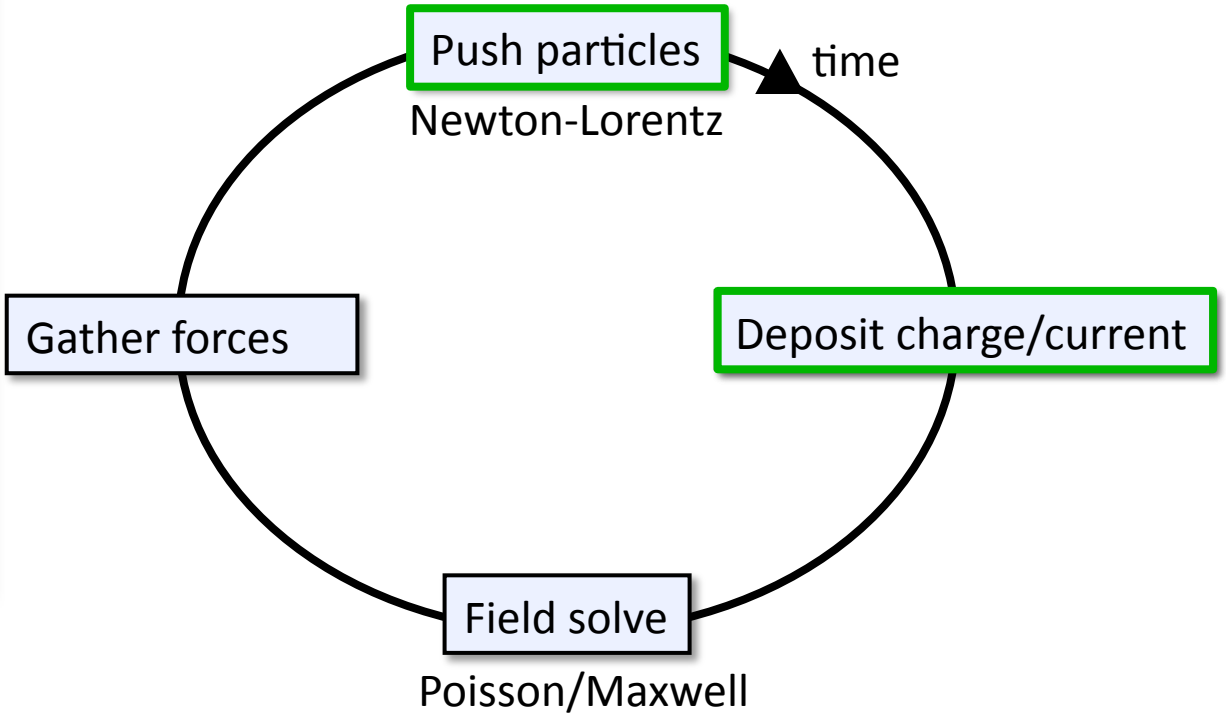
Plasma=collection of interacting charged particles



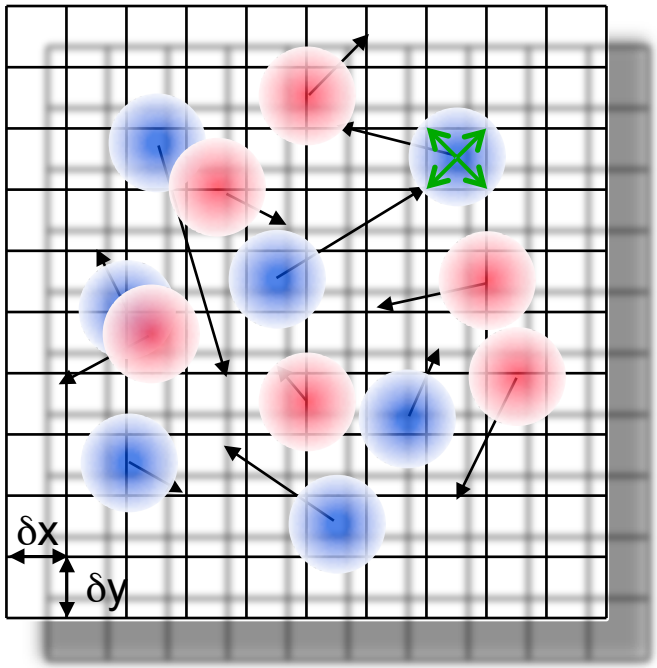
Particle-In-Cell workflow



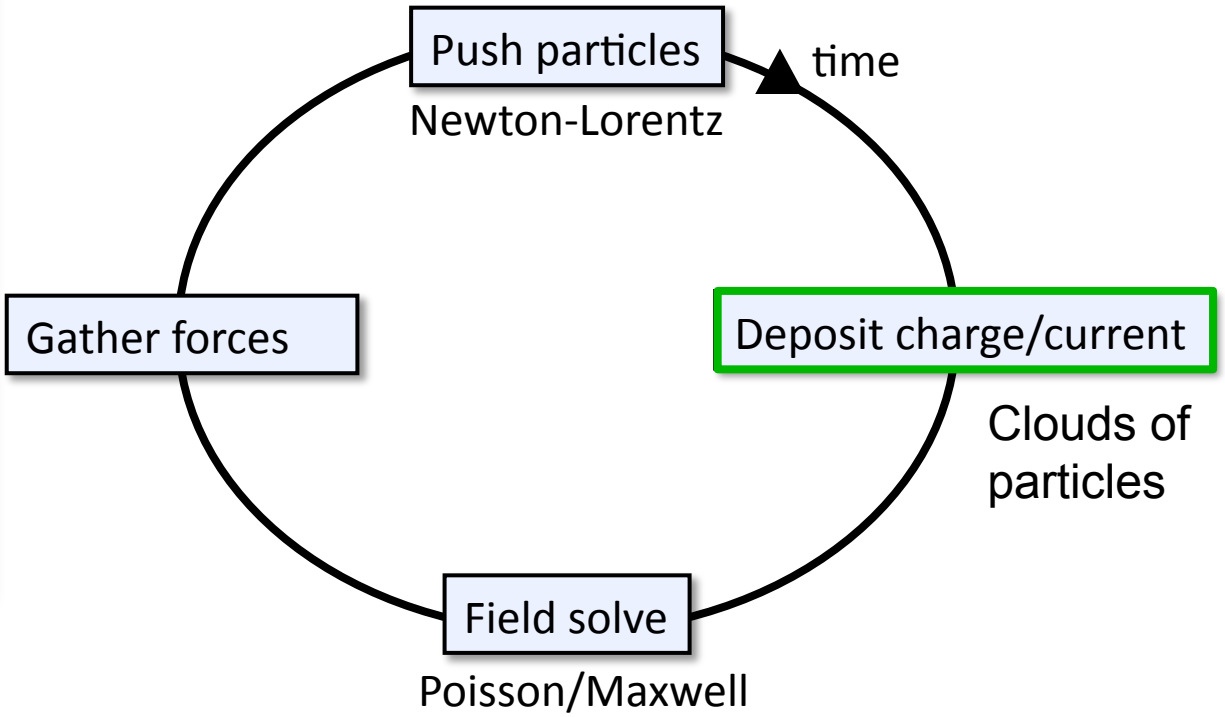
Plasma=collection of interacting charged particles



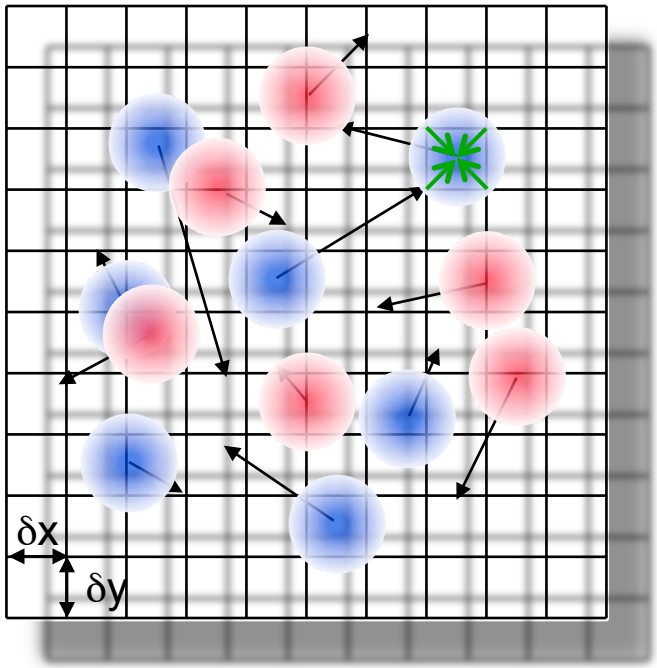
Particle-In-Cell workflow



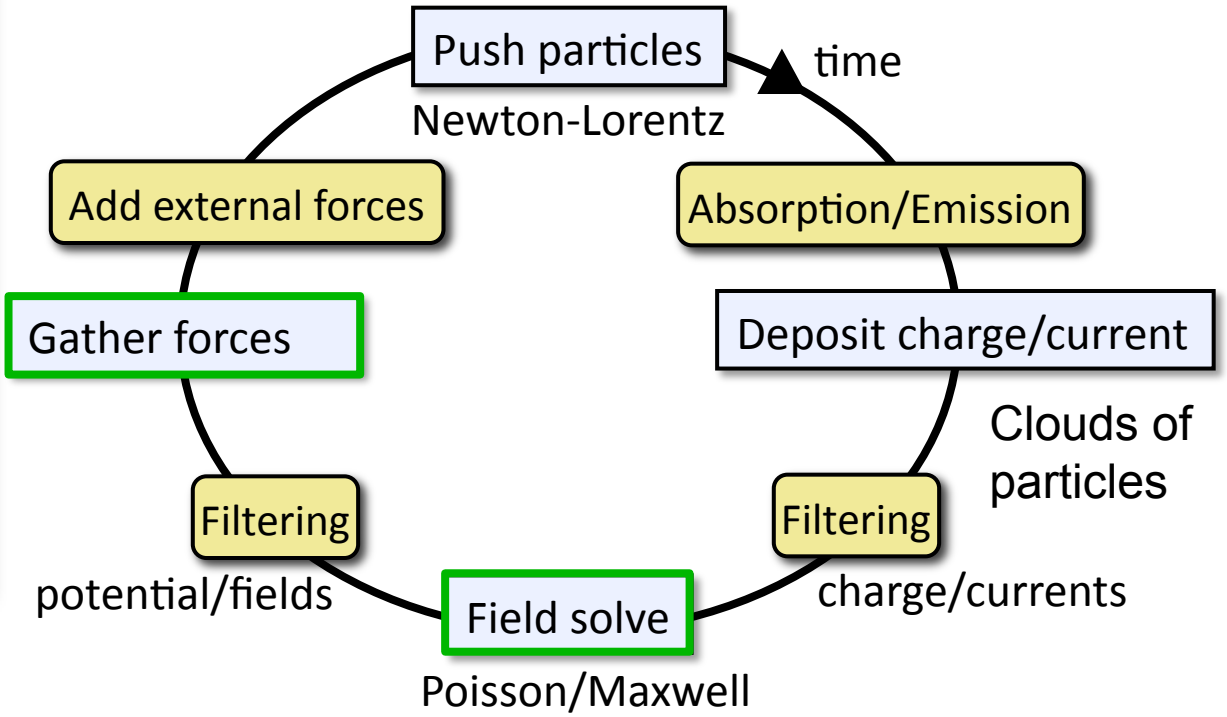
Plasma=collection of interacting charged particles



Particle-In-Cell workflow



Plasma=collection of interacting charged particles



+ filtering (charge,currents and/or potential,fields).

+ absorption/emission (injection, loss at walls, secondary emission, ionization, etc),

+ external forces (accelerator lattice elements),

DSL must allow for different types of push steps according to physics modeled

Particle Push step uses equations of motion. Here, we see a typical Time-difference of eqns of motion: second order leap-frog scheme

$$\frac{d\mathbf{v}_i}{dt} = \frac{q_i}{m_i}\mathbf{E}_s \approx \frac{\mathbf{v}_i(t+\Delta t/2) - \mathbf{v}_i(t-\Delta t/2)}{\Delta t} = \frac{q_i}{m_i}\mathbf{E}_s(t)$$

$$\frac{d\mathbf{x}_i}{dt} \approx \frac{\mathbf{x}_i(t+\Delta t) - \mathbf{x}_i(t)}{\Delta t} = \mathbf{v}_i(t+\Delta t)$$

Solution is explicit time advance:

$$\mathbf{v}_i(t+\Delta t/2) = \mathbf{v}_i(t-\Delta t/2) + \frac{q_i}{m_i}\mathbf{E}_s(\mathbf{x}_i(t))\Delta t$$

$$\mathbf{x}_i(t+\Delta t/2) = \mathbf{x}_i(t) + \mathbf{v}_i(t+\Delta t/2)\Delta t$$

Differences in Particle-In-Cell Codes

- Particle-in-Cell codes are used for a wide variety of applications
- The family of codes known as GTC/GTS implements a Particle method to solve the Gyrokinetic Equations in Tokamaks and other toroidal fusion devices
- Other particle codes are used to model sources for tokamaks, fast ignition concepts for inertial fusion, modeling heavy ion fusion beams

GTS (Gyrokinetic Tokamak Simulation)

- GTS particles are moved along the characteristics in phase space
  - Gyro-averaged Vlasov equation reduced to a simple system of ordinary differential equations for particle push
- Straight-field-line magnetic coordinates in toroidal geometry are employed (natural coordinates for tokamak)
- As before, grid replaces the direct binary interaction between particles by accumulating the charge of those particles on the grid at every time step and solving for the electromagnetic field, which is then gathered back to the particles' positions

The DSL must allow for more complicated particle movers

- **Equations of motion for the particles along the characteristics, slightly more complicated, same type of calculation:**

$$\frac{d\mathrm{R}}{dt} = \mathrm{v}_{\|}\hat{\mathrm{B}} - \left(\frac{q}{m\Omega}\right)\left(\frac{\partial \Psi}{\partial \mathrm{R}} \times \hat{\mathrm{B}}\right)$$

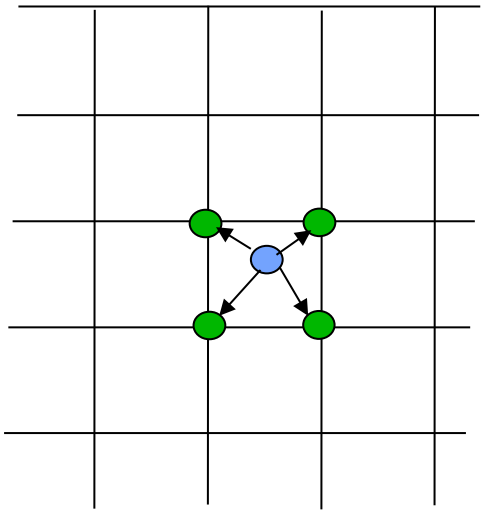
$$\frac{d\mathrm{v}_{\|}}{dt} = -\left(\frac{q}{m}\right)\frac{\partial \Psi}{\partial \mathrm{R}} \cdot \hat{\mathrm{B}}$$

$$\frac{d\mathrm{w}_j}{dt} = -\left[\left(\frac{q}{m\Omega}\right)\frac{\partial \Psi}{\partial \mathrm{R}} \times \hat{\mathrm{B}} \cdot \hat{\mathrm{x}}_k - \left(\frac{q}{m}\right)\frac{\partial \Psi}{\partial \mathrm{R}} \cdot \hat{\mathrm{B}}\frac{1}{f_0}\frac{\partial f_0}{\partial \mathrm{v}_{\|}}\right]_{\mathrm{R}_j, \mu_j}$$

$$\text{with} \quad \mathrm{w}_j = \delta f / f$$

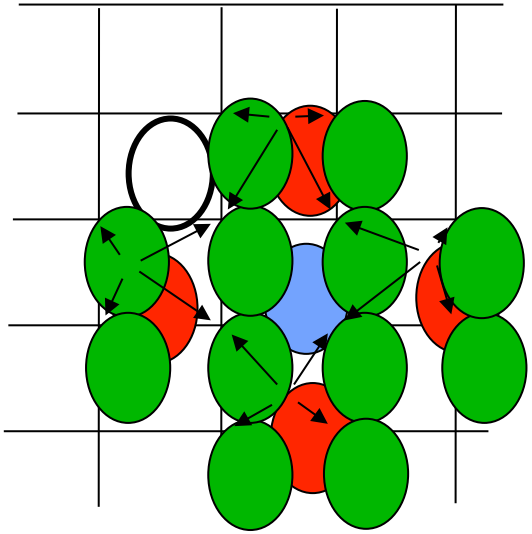
The DSL needs to also have a different Charge Deposition methods

Charge Deposition Step (SCATTER operation)



Classic PIC

GTC



4-Point Average GK
(due to W.W. Lee)

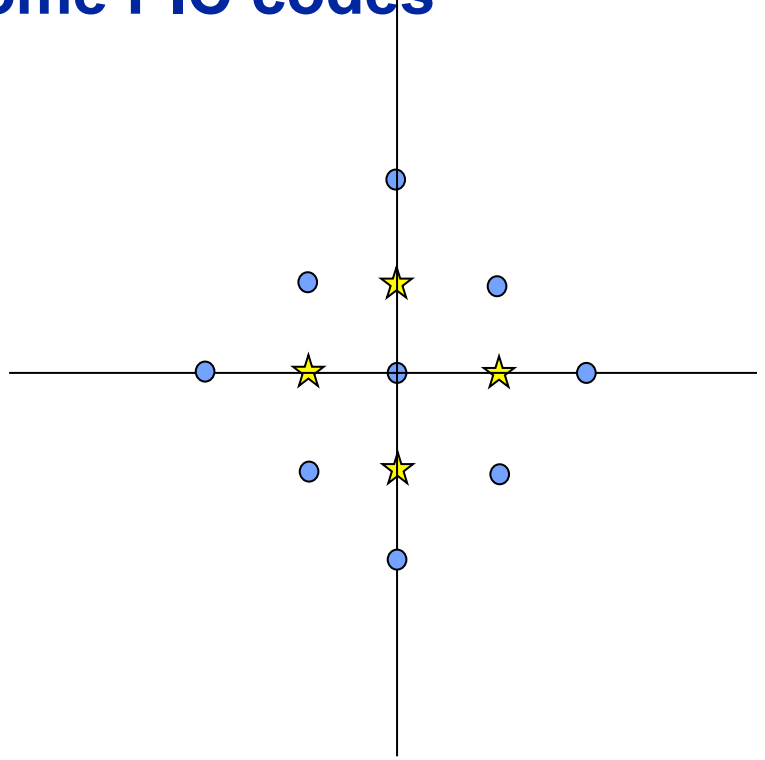
Ethier PPPL

Solvers will also vary in the PIC/DSL (library call?)

- Can be done in real space (iterative solver)
  - –Four or eight-point average method
- Fourier solvers are used in some PIC codes

$$\frac{\tau}{\lambda_D^2}\left(\Phi - \tilde{\Phi}\right) = 4\pi e\left(\bar{n}_i - n_e\right)$$

where $\tilde{\Phi}$ is the second gyrophase-averaged potential

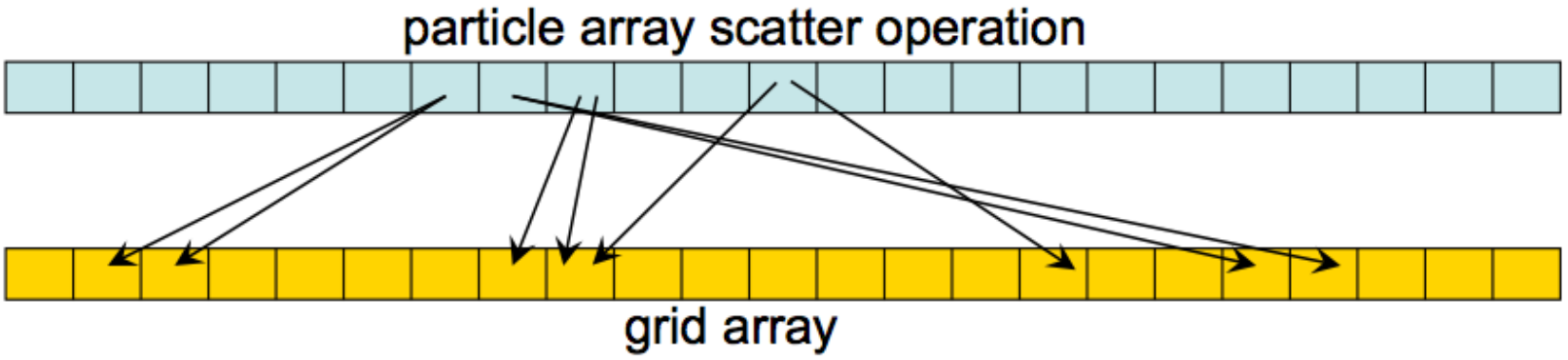


Exascale challenges that DSL can handle:
1D domain decomp. ➔ cache and memory

1D array of particles used to describe data abstraction at lowest level, but they become randomly distributed during the simulation

Grid may be replicated between processes ➔ bad memory footprint

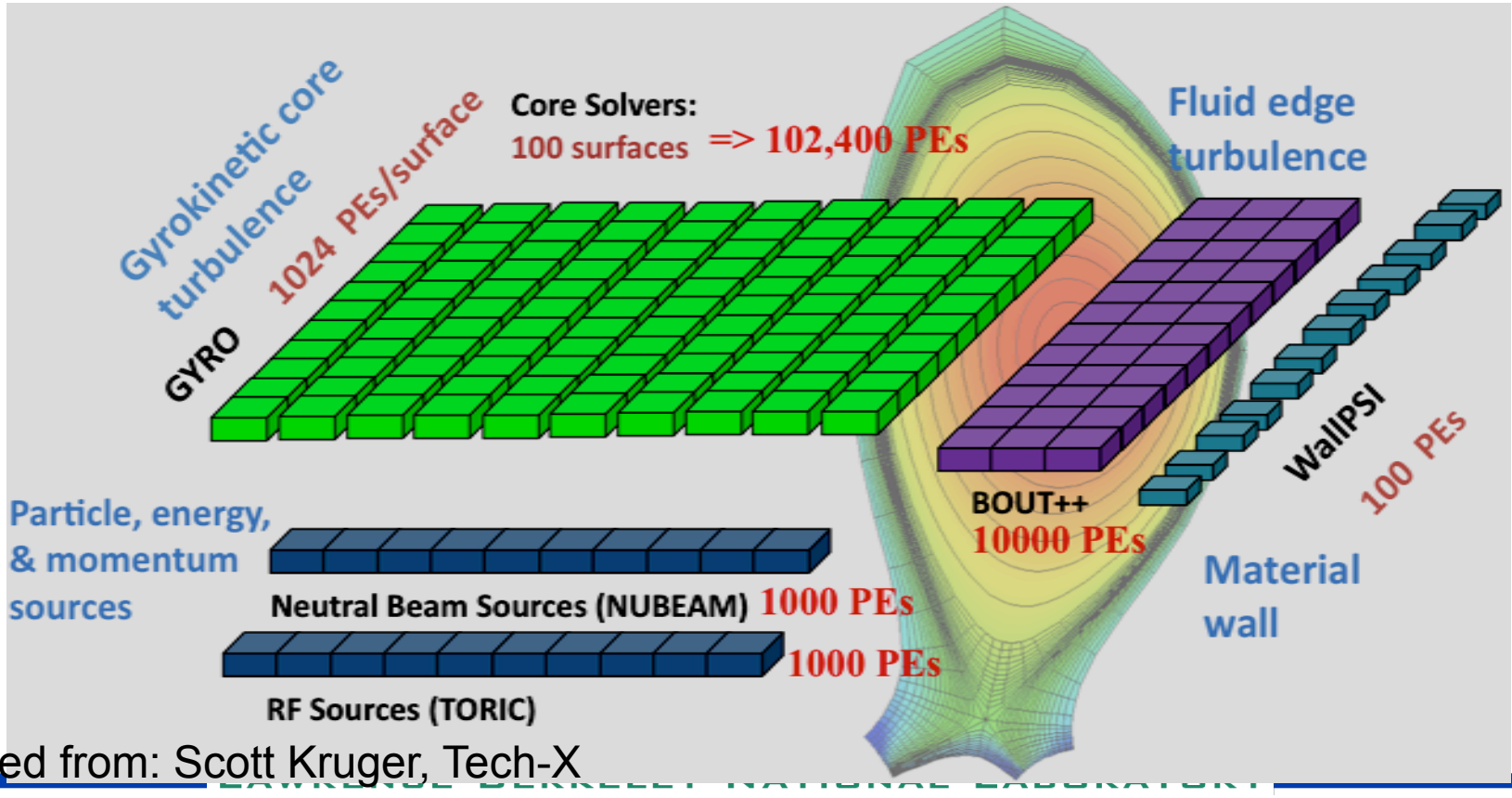
Particle charge deposition on the grid leads to indirect addressing in memory ➔ not cache friendly



PIC codes only part of the story in Fusion

- Core Transport: GYRO/NEO
- Collisional Edge Plasma: BOUT++
- MHD: M3D-C1, NIMROD

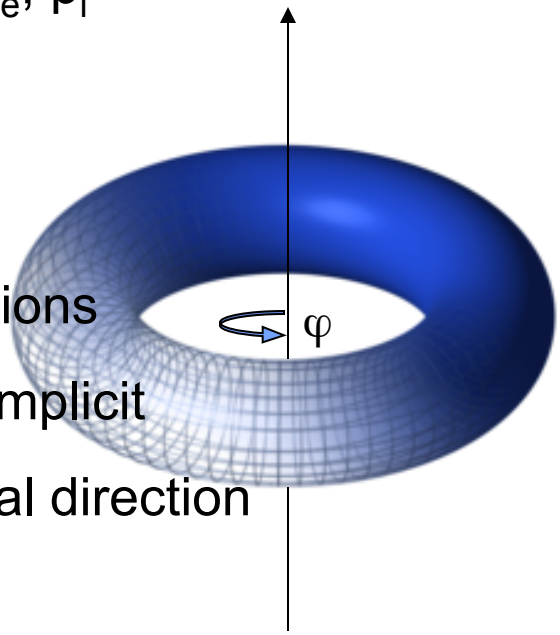
- Explicit PIC Modeling: GTS, VORPAL
- Wave heating, Wall interaction



Adapted from: Scott Kruger, Tech-X

Other opportunities: implicit MHD for tokamaks

~ 8-9 variables per element (or mesh point) **V**, **B**, ρ, p_e, p_i
~ 10^2 toroidal planes (or Fourier modes)
➔Large sparse matrix equations require low latency



Codes vary in:

- single (big) matrix equation or several smaller equations
- non-linearly implicit (NK), linearly implicit, or partial implicit
- spectral, finite element, or finite differences in toroidal direction

Options for DSL:
Something similar to Liszt programming environment?
Lower-level DSL to replace MPI calls
Tokamak-based data structures

Thanks!

- Jean-Luc Vay LBNL
- Viktor Decyk UCLA
- Steve Jardin PPPL
- Stephane Ethier PPPL
- Rebecca Xuefei LBNL