Particle-in-cell simulations

Part II: Implementation of Zeltron

Benoît Cerutti

IPAG, CNRS, Université Grenoble Alpes, Grenoble, France.

Plan of the lectures

- **Monday:**

  - *Morning:* The PIC method, numerical schemes and main algorithms.

  - *Afternoon:* Coding practice of the Boris push and the Yee algorithm.

- **Tuesday:**

  - *Morning:* Implementation of Zeltron, structure and methods.

  - *Afternoon:* Zeltron hands on relativistic reconnection simulations

  - *Evening*: Seminar about application of PIC to pulsar magnetospheres.

- ***Wednesday:***

  - *Morning:* *B*oundary conditions and parallelization in Zeltron.

  - *Afternoon:* Zeltron Hands on relativistic collisionless shocks simulations

General presentation

Zeltron is an **explicit, relativistic 3D PIC code** created from scratch in 2012. Originally designed to study particle acceleration in relativistic magnetic reconnection sites applied to astrophysics.

Main developers:

Benoît Cerutti *(CNRS / Univ. Grenoble Alpes)*
Greg Werner *(University of Colorado)*

Some general features

- Written in **Fortran 90**
- **Yee FDTD** algorithm for the fields
- **Boris push** for the particles
- **Efficiently parallelized** with MPI (3D domain decomposition)
- Includes **synchrotron** and **inverse Compton** radiation reaction forces
- **Non Cartesian-mesh**: spherical, cylindrical, Schwarzschild (not public)
- Large set of tools for **data reduction** and **data analysis** on the fly
- Set of **boundary conditions** (absorption, creation, open, reflective, …)
- **No need** for **external libraries**

Global structure

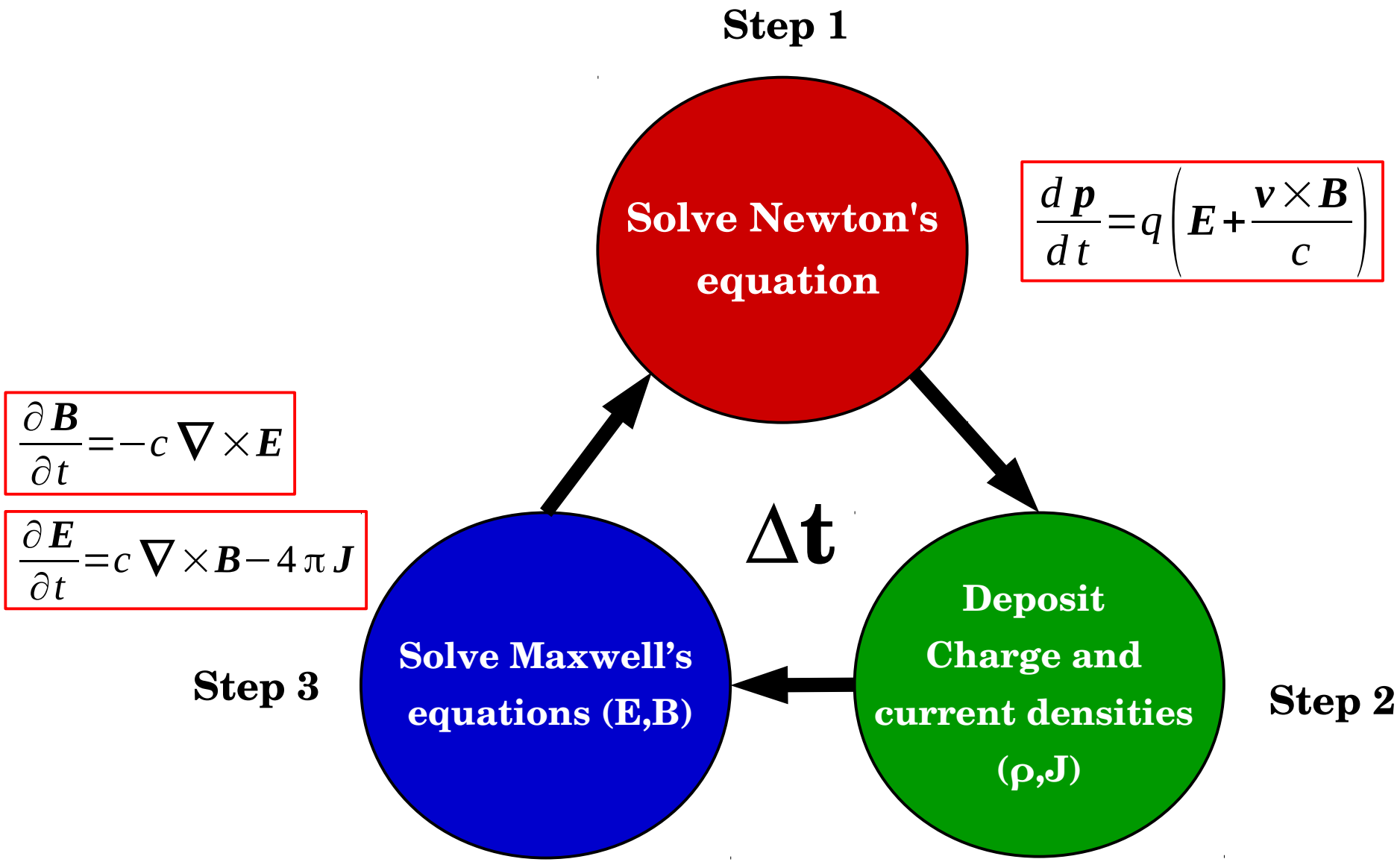
PROGRAM:

main.f90: Contains the main body of the code, initialization, main loop in time.

MODULES:

- *mod_input.f90:* Input file where all the numerical and physical parameters can be set. This is the only file to modify for a given setup.
- *mod_initial.f90*: Initialize the fields and the particles (problem dependent).
- *mod_fields.f90*: Solve Maxwell equations and other operations related to fields.
- *mod_motion.f90:* Contains the particle push, applies boundary conditions and exchange particles at the boundaries between processors
- *mod_rhoj.f90*: For depositing particle charge and current on the grid.
- *mod_analysis.f90*: Data reduction and analysis on the fly and output to the disk
- *Others*: More technical modules related to I/O, interpolation, and parallelization.

Computation procedure per timestep in PIC



General structure of the main.f90 file

INITIALISATION

1. Initialize the MPI environment for parallel computing (see *lecture III*)
2. Initialize the spatial grid, and Yee mesh (this *lecture*)
3. Set the initial conditions (Particles and fields) at t=0 (this *lecture*)
4. Write initial data to disk
5. Leap-frog initialization: evolve $\mathbf{u}^{0} \rightarrow \mathbf{u}^{0-1/2}$ (see *lecture I*)

MAIN LOOP IN TIME

– For each particle species

1. Boris push: $\mathbf{u}^{n-1/2} \rightarrow \mathbf{u}^{n+1/2}$ (see *lecture I*)
2. Deposit currents: $J_0 = 0.5\rho^{n}\mathbf{v}^{n+1/2}$ (see *lecture I*)
3. Evolve particle positions: $\mathbf{r}^{n} \rightarrow \mathbf{r}^{n+1}$
4. Apply boundary conditions and MPI communications (see *lecture III*)
5. Deposit currents: $J = 0.5\rho^{n+1}\mathbf{v}^{n+1/2} + J_0$ (see *lecture I*)

Sample from the main.f90 file (particles)

```
! Update u from t-dt/2 to t+dt/2, particle positions unchanged
CALL BORIS_PUSH(-1d0,me,pcl_ed,pcl_data_ed,Bxg,Byg,Bzg,Exg,Eyg,Ezg,&
                Uph,xgp,ygp,Esyned,Eicsed,NED)

! Computation of rho at t and half of the current density
CALL RHOJ(-1d0,pcl_ed,rhoed,Jx0,Jy0,Jz0,xgp,ygp,NED,id,ngh,TOPO_COMM,ierr)

! Push the particles from t to t+dt
CALL PUSH_PARTICLES(pcl_ed,NED)

! Applying boundary conditions to the particles
CALL BOUNDARIES_PARTICLES(pcl_ed,pcl_data_ed,taged,NED)

! Counting the particles leaving each subdomain
CALL COUNT_ESCAPE(pcl_ed,xminp,xmaxp,yminp,ymaxp,NED,NESC)

! Exchange of particles at the boundaries between processes
CALL COM_PARTICLES(pcl_ed,pcl_data_ed,taged,xminp,xmaxp,yminp,ymaxp,&
                   NED,NESC,id,ngh,TOPO_COMM,ierr)

! Computation of rho at t+dt and the other half of the current density
CALL RHOJ(-1d0,pcl_ed,rhoed,Jxed,Jyed,Jzed,xgp,ygp,NED,id,ngh,TOPO_COMM,ierr)

! Total current density at t+dt/2
Jxed=Jxed+Jx0
Jyed=Jyed+Jy0
Jzed=Jzed+Jz0
```

General structure of the main.f90 file

MAIN LOOP IN TIME (continues)

- For the fields

1. Collect currents from all species, and put current on the Yee mesh.
2. Push B field half time step: $B^n \rightarrow B^{n+1/2}$ (see *lecture I*)
3. Push E field full time step: $E^n \rightarrow E^{n+1}$ (see *lecture I*)
4. Correct the E field (charge conservation) with Poisson solver (this *lecture*)
5. Push B field half time step: $B^{n+1/2} \rightarrow B^{n+1}$ (see *lecture I*)

- Analyze and write data to disk every FDUMP time steps (this *lecture*)

- Create a checkpoint every FSAVE time steps

Sample from the main.f90 file (fields)

```
!=====================================================================
! B FIELD at t=t+dt/2
!=====================================================================

CALL PUSH_BHALF(Bx,By,Bz,Ex,Ey,Ez,xgp,ygp,id,ngh,TOPO_COMM,ierr)

!=====================================================================
! SOLVE MAXWELL'S EQUATIONS at t=t+dt
!=====================================================================

! E FIELD at t=t+dt
CALL PUSH_EFIELD(Bx,By,Bz,Ex,Ey,Ez,Jx,Jy,Jz,xgp,ygp,id,ngh,TOPO_COMM,ierr)

IF (MOD(it,FREQ_POISSON).EQ.0) THEN
! Solve Poisson equation to ensure that div(E)=4*pi*rho
CALL CORRECT_EFIELD(Ex,Ey,xgp,ygp,rho,id,ngh,TOPO_COMM,ierr)
END IF

! B FIELD at t=t+dt
CALL PUSH_BHALF(Bx,By,Bz,Ex,Ey,Ez,xgp,ygp,id,ngh,TOPO_COMM,ierr)

CALL FIELDS_NODES(Bx,By,Bz,Ex,Ey,Ez,Bxg,Byg,Bzg,Exg,Eyg,Ezg,xgp,ygp,&
                  id,ngh,TOPO_COMM,ierr)
```

The grid and the Yee mesh

- Define number of cells and spatial boundaries from the input file (*mod_input.f90*)

```
! Number of cells in X
INTEGER*8, PARAMETER, PUBLIC :: NCX=128

! Number of cells in Y
INTEGER*8, PARAMETER, PUBLIC :: NCY=128

! Spatial boundaries in the X-direction
DOUBLE PRECISION, PARAMETER, PUBLIC :: xmin=0d0,xmax=100.0

! Spatial boundaries in the Y-direction
DOUBLE PRECISION, PARAMETER, PUBLIC :: ymin=0d0,ymax=100.0

! Spatial step
DOUBLE PRECISION, PARAMETER, PUBLIC :: dx=(xmax-xmin)/NCX
DOUBLE PRECISION, PARAMETER, PUBLIC :: dy=(ymax-ymin)/NCY
```

- Build the **spatial arrays** (*main.f90*)

```
! Nodal lattice
DO ix=1,NX
xg(ix)=(ix-1)*1d0/((NX-1)*1d0)*(xmax-xmin)+xmin
ENDDO

DO iy=1,NY
yg(iy)=(iy-1)*1d0/((NY-1)*1d0)*(ymax-ymin)+ymin
ENDDO

! Yee lattice
xyee=xg+dx/2.0
yyee=yg+dy/2.0
```

Particle data structure

The particle distribution function is characterized by 3 arrays (*mod_input.f90*):

```
! BACKGROUND ELECTRONS distribution function components
DOUBLE PRECISION, ALLOCATABLE, PUBLIC :: pcl_eb(:,:)
DOUBLE PRECISION, ALLOCATABLE, PUBLIC :: pcl_data_eb(:,:)
INTEGER*8, ALLOCATABLE, PUBLIC        :: tageb(:)
```

`ALLOCATE(pcl_eb(1:7,1:NP)):` Actual particle distribution function (pb independent)

List of particles ↓

	x	y	z	u_x	u_y	u_z	wgt
Part 1							
Part 2							
...							
Part NP							

`ALLOCATE(pcl_data_eb(1:4,1:NP)):` Additional particle data information (pb dependent)

List of particles ↓

	Data 1	Data 2	Data 3	Data 4
Part 1				
Part 2				
...				
Part NP				

Particle data structure

The particle distribution function is characterized by 3 arrays (*mod_input.f90*):

```
! BACKGROUND ELECTRONS distribution function components
DOUBLE PRECISION, ALLOCATABLE, PUBLIC :: pcl_eb(:,:)
DOUBLE PRECISION, ALLOCATABLE, PUBLIC :: pcl_data_eb(:,:)
INTEGER*8, ALLOCATABLE, PUBLIC        :: tageb(:)
```

`ALLOCATE(tag_eb(1:NP)):` Each particle is identified by a **unique integer number**

=> *Useful for particle tracking*

List of particles

	tag
Part 1	tag1
Part 2	tag2
...	...
Part NP	tag3

Fields/Currents data structure

- **Fields** defined on the Yee lattice (*main.f90*): Used to **evolve the fields** (*see Lecture I*)

```
!*****************************************************
! Magnetic and Electric fields components Yee lattice
DOUBLE PRECISION, DIMENSION(1:NX,1:NY) :: Bx,By,Bz
DOUBLE PRECISION, DIMENSION(1:NX,1:NY) :: Ex,Ey,Ez
```

- The **current** density is needed on the Yee mesh (*main.f90*)

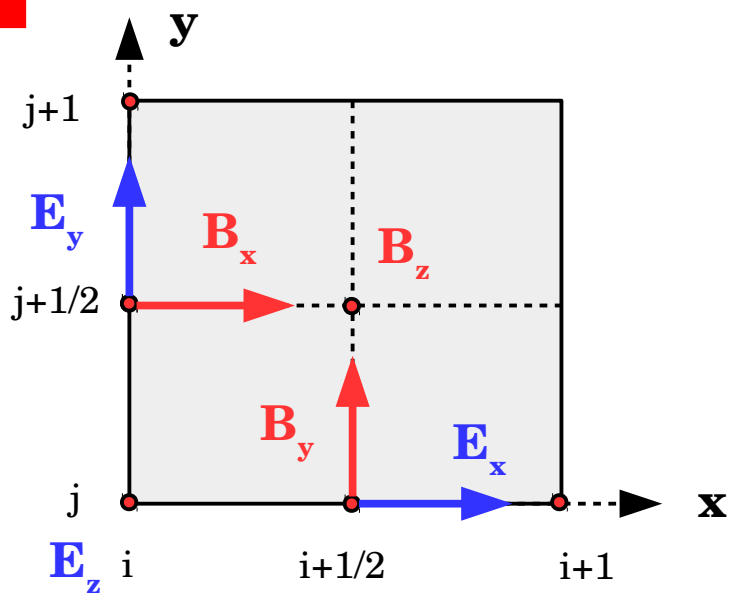
```
!*****************************************************
! Current density components Yee lattice
DOUBLE PRECISION, DIMENSION(1:NX,1:NY) :: Jx,Jy,Jz
```

- **Fields** defined on the nodes of the grid (*main.f90*): Used to **evolve the particles** (*see Lecture I*)

```
!*****************************************************
! Magnetic and Electric fields components at nodes
DOUBLE PRECISION, DIMENSION(1:NX,1:NY) :: Bxg,Byg,Bzg
DOUBLE PRECISION, DIMENSION(1:NX,1:NY) :: Exg,Eyg,Ezg
```

From Yee to nodes

2D



- **Electric field**

$$Exg_{i,j} = \frac{Ex_{i+1/2,j} + Ex_{i-1/2,j}}{2}$$

$$Eyg_{i,j} = \frac{Ey_{i,j+1/2} + Ey_{i,j-1/2}}{2}$$

$$Ezg_{i,j} = Ez_{i,j}$$

- **Magnetic field**

$$Bxg_{i,j} = \frac{Bx_{i,j+1/2} + Bx_{i,j-1/2}}{2}$$

$$Byg_{i,j} = \frac{By_{i+1/2,j} + By_{i-1/2,j}}{2}$$

$$Bzg_{i,j} = \frac{Bz_{i+1/2,j+1/2} + Bz_{i-1/2,j+1/2} + Bz_{i+1/2,j-1/2} + Bz_{i-1/2,j-1/2}}{4}$$

Correction of the electric field

Zeltron does not use a charge conserving deposition scheme

=> **Poisson equation must be solved!**

Consider that: $\vec{E}' = \vec{E} + \delta\vec{E}$

Small correction

Correct — Known field

$$\delta\vec{E} = -\nabla\delta\varphi \rightarrow \nabla^2\delta\varphi = -\left(4\pi\rho - \nabla\cdot\vec{E}\right)$$

Solve this equation using a Gauss-Seidel iterative method

"Empirically", Zeltron uses **500 iterations**, every **25 time steps**.

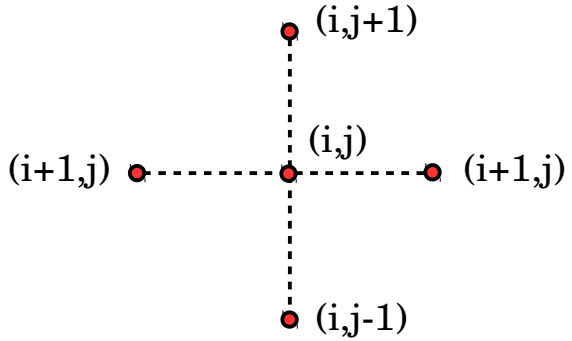
Relevant parameters in *mod_input.f90*

```
! Poisson solver calling frequency in terms of timesteps
INTEGER, PARAMETER, PUBLIC :: FREQ_POISSON=25

! Number of iterations to solve Poisson's equation
INTEGER, PARAMETER, PUBLIC :: NIT=500
```

Poisson solver

$$\nabla^2 \delta\varphi = -\left(4\pi\rho - \nabla\cdot\vec{E}\right)$$

2D Example: 5 points stencil



$$\nabla^2 \delta\varphi = \frac{\partial^2 \delta\varphi}{\partial x^2} + \frac{\partial^2 \delta\varphi}{\partial y^2} \approx \frac{\delta\varphi_{i+1,j} - 2\delta\varphi_{i,j} + \delta\varphi_{i-1,j}}{\Delta x^2} + \frac{\delta\varphi_{i,j+1} - 2\delta\varphi_{i,j} + \delta\varphi_{i,j-1}}{\Delta y^2}$$

Injecting this into Poisson and after some rearrangements yields (and if $\Delta x = \Delta y$):

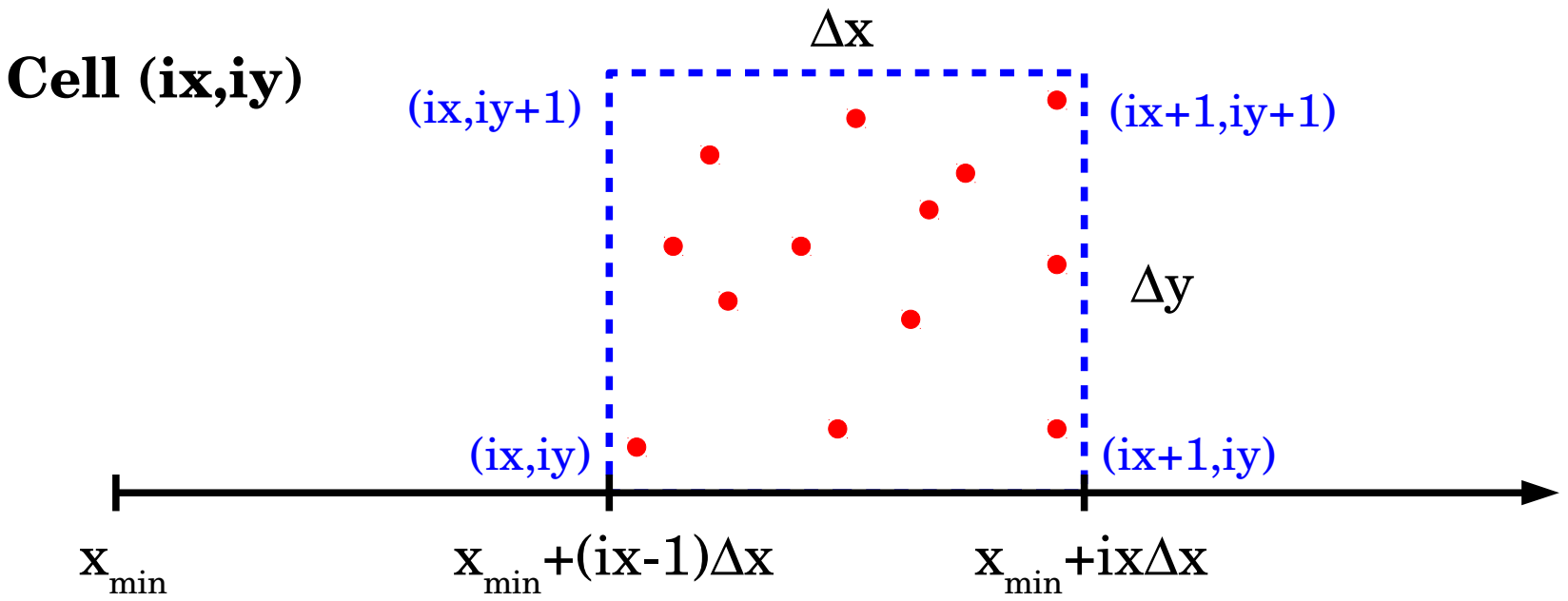
$$\delta\varphi_{i,j} = \frac{1}{4}\left(\delta\varphi_{i+1,j} + \delta\varphi_{i-1,j} + \delta\varphi_{i,j+1} + \delta\varphi_{i,j-1} + \left(4\pi\rho - \nabla\cdot\vec{E}\right)\Delta x^2\right)$$

Calling the subroutine in *main.f90*

```
IF (MOD(it,FREQ_POISSON).EQ.0) THEN
! Solve Poisson equation to ensure that div(E)=4*pi*rho
CALL CORRECT_EFIELD(Ex,Ey,xgp,ygp,rho,id,ngh,TOPO_COMM,ierr)
END IF
```

Particle initialization: Spatial distribution

Let's consider a **uniform distribution** in 2D (*mod_initial.f90*)




```
!*****************************************************
! Definition of the initial position (x0,y0) as a uniform
random number defined between 0 and 1

x0c=0.0
y0c=0.0

CALL RANDOM_NUMBER(x0c)
CALL RANDOM_NUMBER(y0c)

x0c=xmin+(ix-1)*dx+x0c*dx
y0c=ymin+(iy-1)*dy+y0c*dy

!*****************************************************
```

Particle weight

- A macroparticle represents **a large number** of physical particles following the exact same trajectory in phase space.

  => The particle weight gives the **normalization factor** to connect between numerical and physical plasma densities.

- For a **uniform** **physical** and **numerical** plasma density, all the particles have the same weight given by:

$$weight = \frac{N_{phys}}{N_{num}} = \frac{n \times (L_x L_y L_z)}{N_{num}}$$

Physical density

Sample code in 2D from *main.f90:*

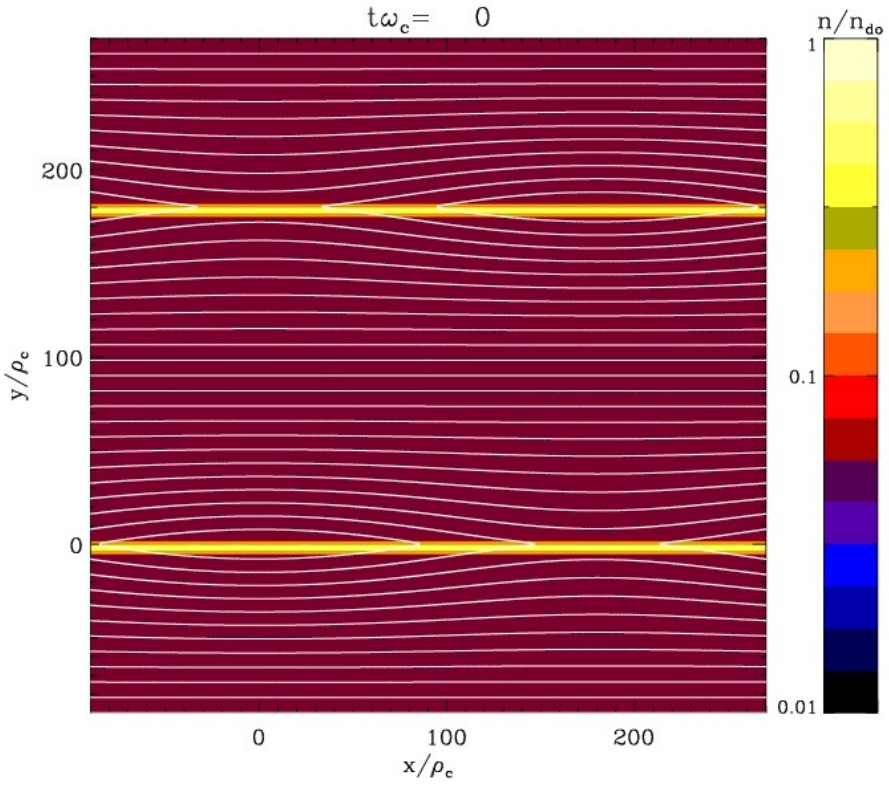
```
! Weight of background particles
pcl_eb(7,:)=density_ratio*nd0*(xmax-xmin)*(ymax-ymin)/NP
pcl_pb(7,:)=density_ratio*nd0*(xmax-xmin)*(ymax-ymin)/NP
```

Variable particle weight

Large density contrast or **sharp density profiles** are hard to model with a constant particle weight.

=> Result in a **bad sampling** of low density regions

Example: *Relativistic reconnection studies where factor >10 between the background and the sheet plasmas.*



Solution: Variable particle weighting: $weight(y_i) = n(y_i) \times \frac{L_x L_y L_z}{N_{num}}$

Macroscopic quantities reconstruction

One can reconstruct global and fluid quantities from the particles: (*mod_analysis.f90*)

- **Particle spectrum:** $\frac{dN}{d\gamma} \approx \frac{1}{\Delta\gamma} \sum_{k=1}^{N_{cell}} w_k$ `SUBROUTINE SPECTRUM_ANGULAR`
- **Plasma density:** $\frac{dN}{dV} \approx \frac{1}{\Delta V} \sum_{k=1}^{N_{cell}} w_k$ `SUBROUTINE MAP_XY`
- **Stress-energy tensor:** $T^{\mu\nu} = \left(\rho + \frac{P}{c^2}\right) V^\mu V^\nu - P\eta^{\mu\nu}$ `SUBROUTINE MAP_FLUID`

  - *Energy density*: $T^{00} \equiv U_e \approx \frac{1}{\Delta V} \sum_{k=1}^{N_{cell}} w_k \gamma_k mc^2$

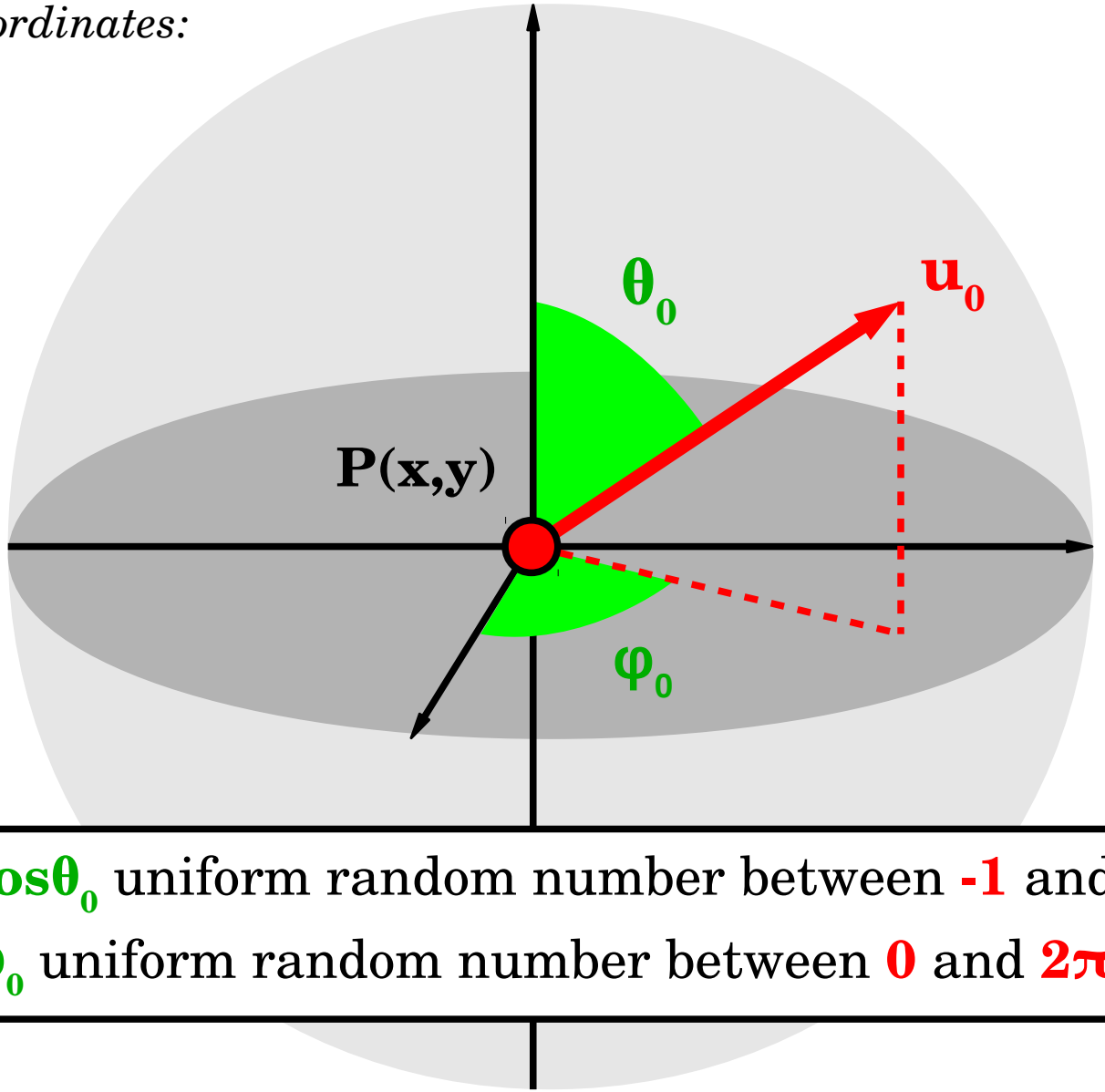
  - *Momentum density:* $T^{0i} \equiv \vec{U}_p \approx \frac{1}{\Delta V} \sum_{k=1}^{N_{cell}} w_k \vec{u}_k mc$

  - *Pressure tensor:* $T^{ij} \equiv P^{ij} \approx \frac{1}{\Delta V} \sum_{k=1}^{N_{cell}} \left(\frac{w_k m c^2}{\gamma_k}\right) u_k^i u_k^j$

Particle initialization: Angular distribution

Let's consider an **isotropic angular distribution** in 3D

In spherical coordinates:



- $\cos\theta_0$ uniform random number between **-1** and **1.**
- φ_0 uniform random number between **0** and **2π.**

Particle initialization: Angular distribution

Let's consider an **isotropic angular distribution** in 3D (*mod_initial.f90*)

```
!*****************************************************
! Definition of the initial phi0 as a uniform random value
between 0 and 1

phi0=0.0

CALL RANDOM_NUMBER(phi0)

phi0=phi0*2.0*pi

! Definition of the initial cth0=cos(theta0) as a uniform
random value between 0 and 1

cth0=0.0

CALL RANDOM_NUMBER(cth0)

cth0=cth0*2.0-1.0

! Initial 4-velocity components
ux0c=u0*sqrt(1.0-cth0*cth0)*cos(phi0)
uy0c=u0*sqrt(1.0-cth0*cth0)*sin(phi0)
uz0c=u0*cth0
```

Particle initialization: Energy distribution

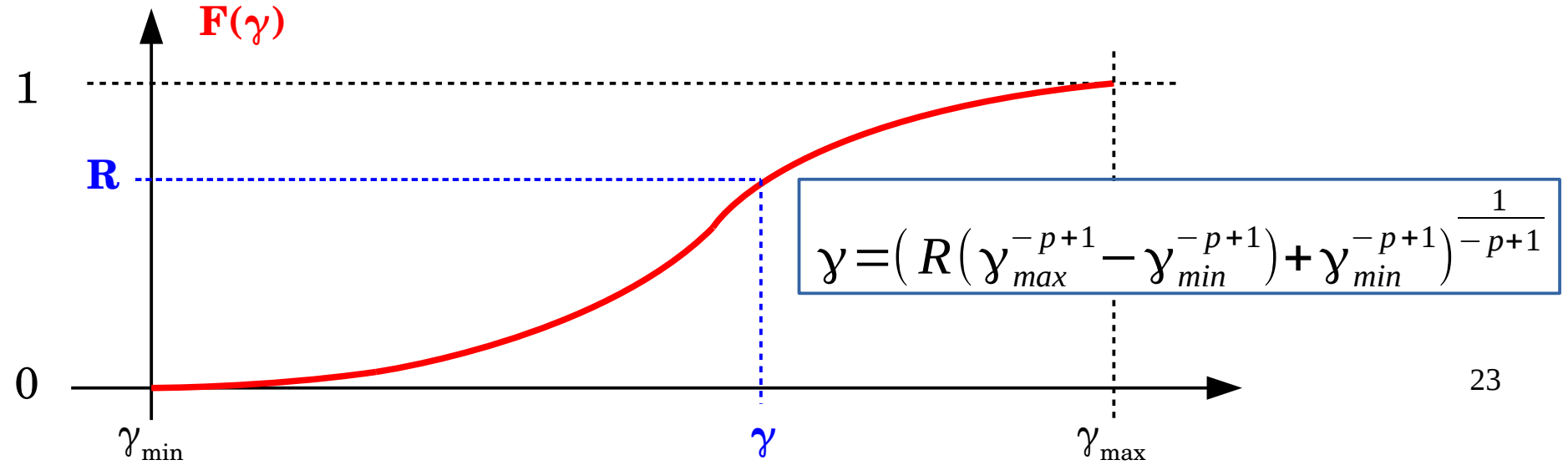
Let's consider a **Power-law distribution:**

$$f(\gamma) = \frac{dN}{d\gamma} \propto \gamma^{-p} \qquad \text{with} \qquad \gamma \in [\gamma_{min}, \gamma_{max}]$$

The trick is to use the **cumulative distribution:**

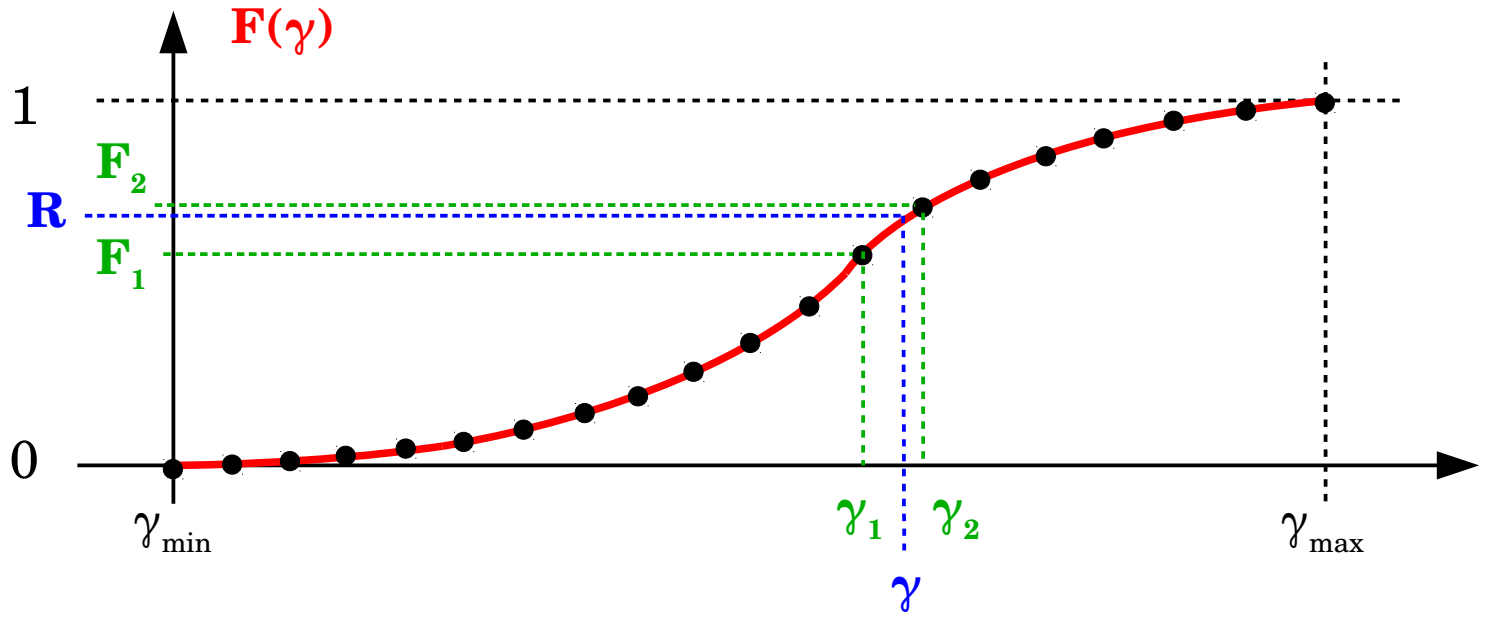
$$F(\gamma) \equiv \frac{\int_{\gamma_{min}}^{\gamma} f(\gamma')\, d\gamma'}{\int_{\gamma_{min}}^{\gamma_{max}} f(\gamma')\, d\gamma'} = \frac{\gamma^{-p+1} - \gamma_{min}^{-p+1}}{\gamma_{max}^{-p+1} - \gamma_{min}^{-p+1}}$$

Next, define a **uniform random number R between 0 and 1**, and sample F to invert the distribution:



Particle initialization: Energy distribution

However, the cumulative distribution is **often a tabulated** (no analytical expression)



We need to interpolate, a **linear interpolation** usually suffices:

$$R \approx (1-\Gamma) F_1 + \Gamma F_2 \qquad \Gamma = \frac{\gamma - \gamma_1}{\gamma_2 - \gamma_1}$$

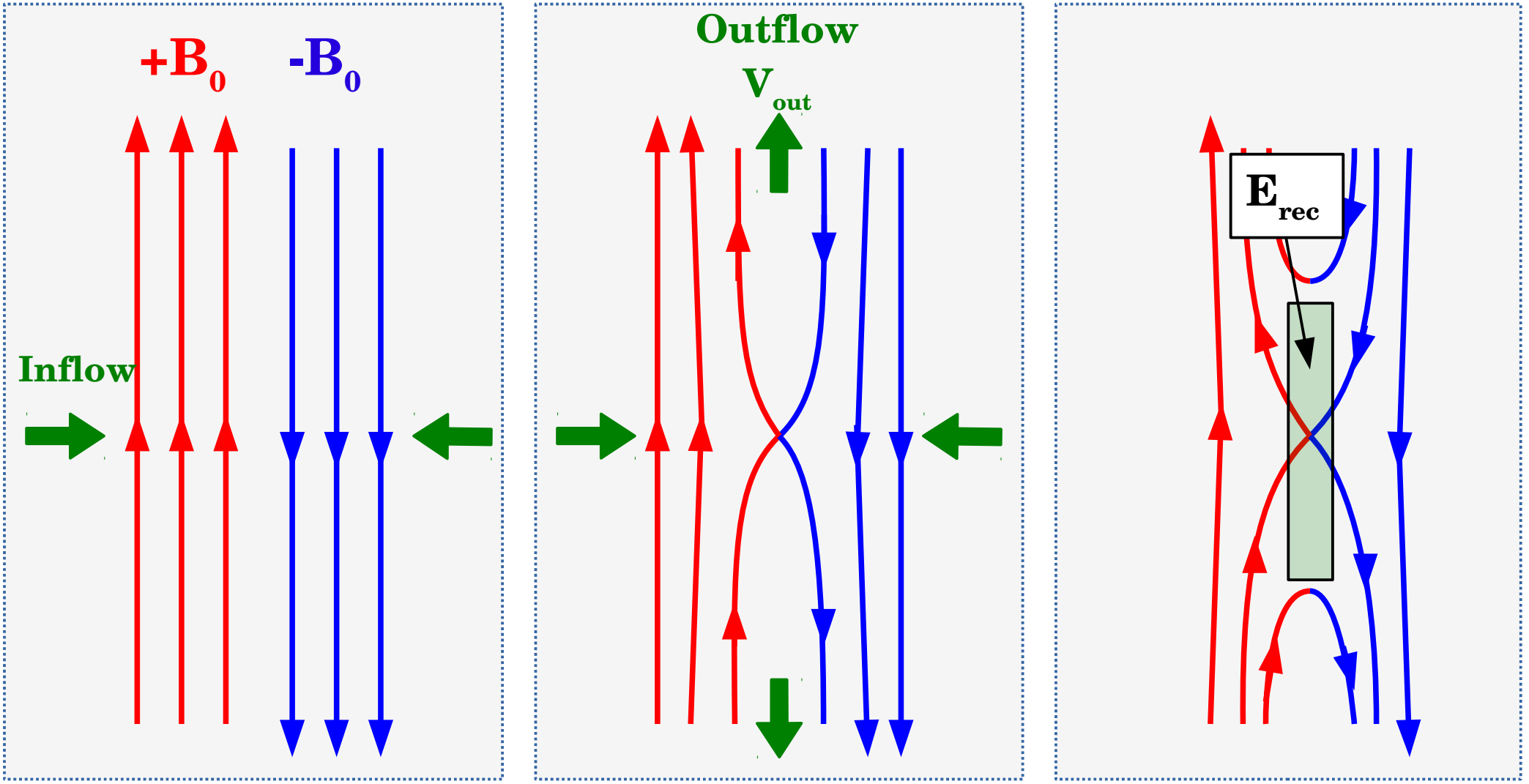
$$\boxed{\gamma \approx \frac{R(\gamma_2 - \gamma_1) - (\gamma_2 F_1 - \gamma_1 F_2)}{F_2 - F_1}}$$

Particle initialization: Energy distribution

In Zeltron, it looks like this (*mod_initial.f90*):

```
Ru=0.0

CALL RANDOM_NUMBER(Ru)
Ru=Ru*0.9999

! Sum over all particles in a the cell
DO ic=1,PPC

! Localize closest known values of cumulative distribution gFu
minu=minloc(abs(gFu-Ru(ic)))
iu=minu(1)

! F1 and F2
gFu1=gFu(iu)
gFu2=gFu(iu+1)

! u1 and u2
u1=ud(iu)
u2=ud(iu+1)

! Linear interpolartion in the log-linear plane (numerically
more accurate)
u0(ic)=exp((Ru(ic)*(log(u2)-log(u1))-
log(u2)*gFu1+log(u1)*gFu2)/(gFu2-gFu1))

ENDDO
```

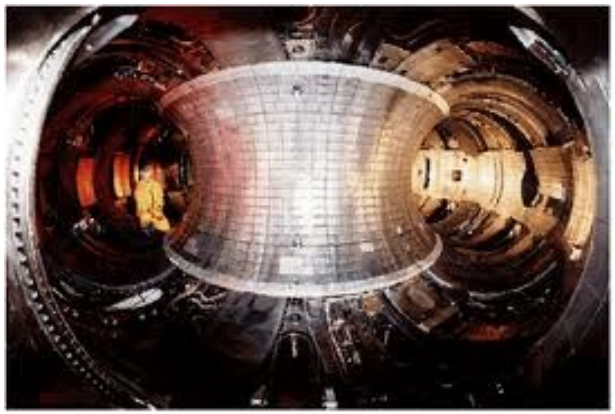
Hands-on II: Relativistic reconnection



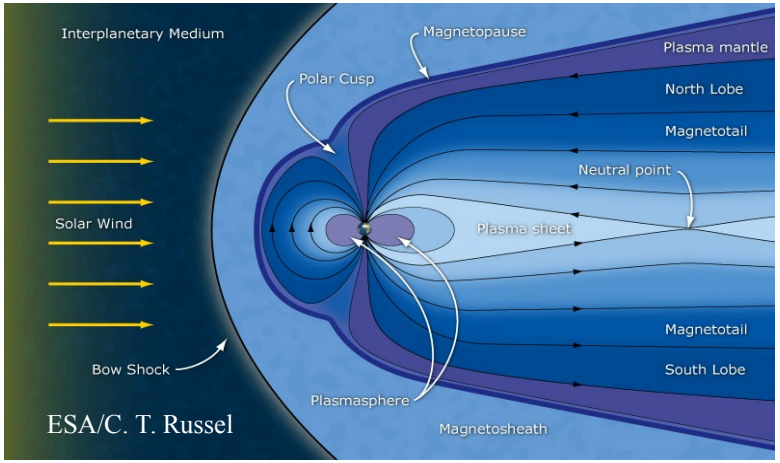
Magnetic energy =? Plasma kinetic energy (heating+non-thermal particles)

How fast does reconnection proceed?
How efficient at accelerating particles?
What are the main acceleration mechanisms?

A common phenomenon

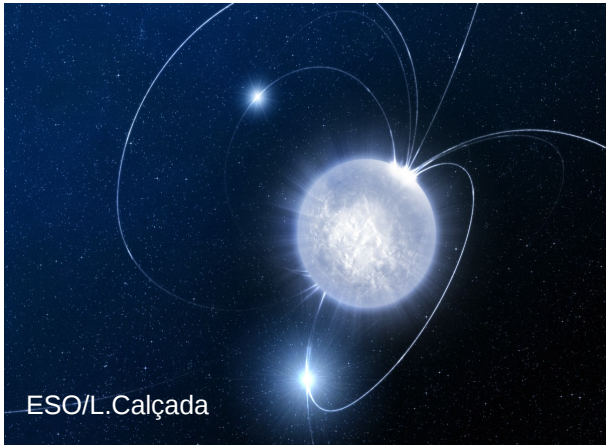


Man-made plasma devices
(e.g., Tokamaks, MRX at PPPL)

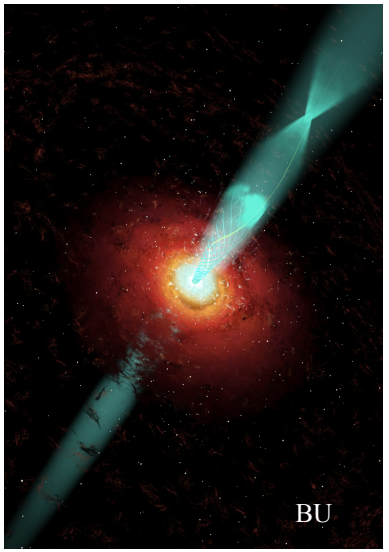


Planet's magnetosphere
(e.g., Earth, Jupiter, Saturn)

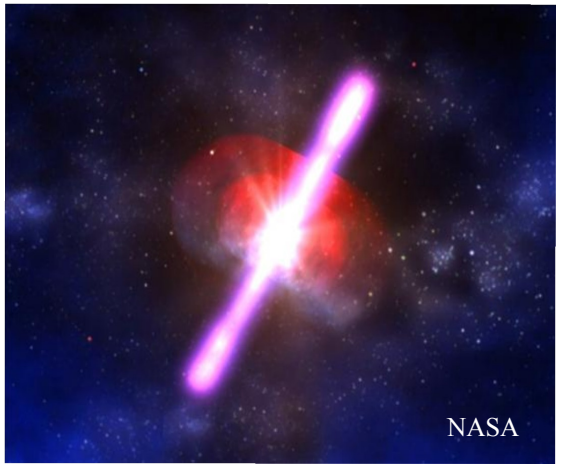
Solar corona



Pulsars/Magnetars



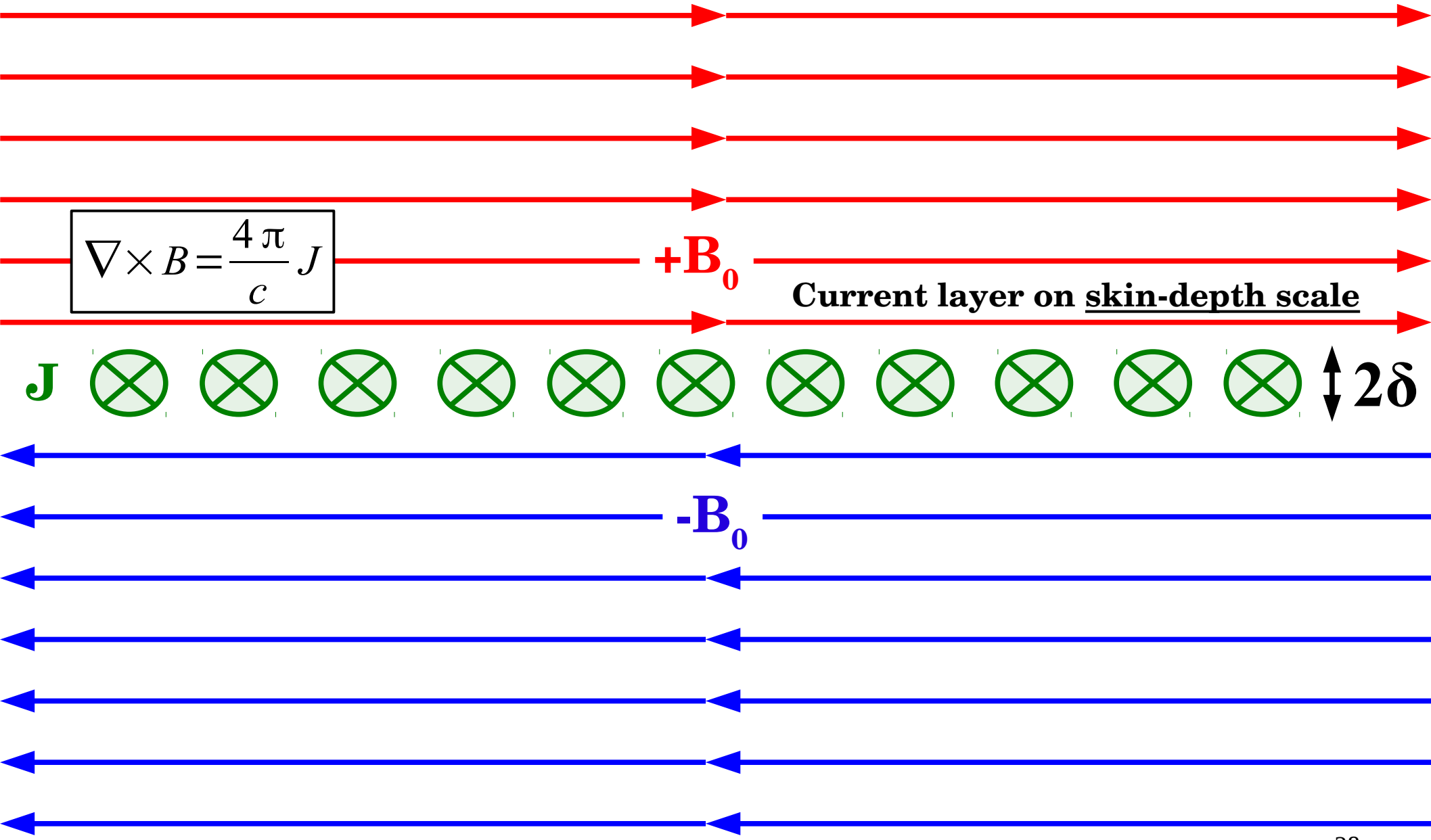
Astrophysical jets



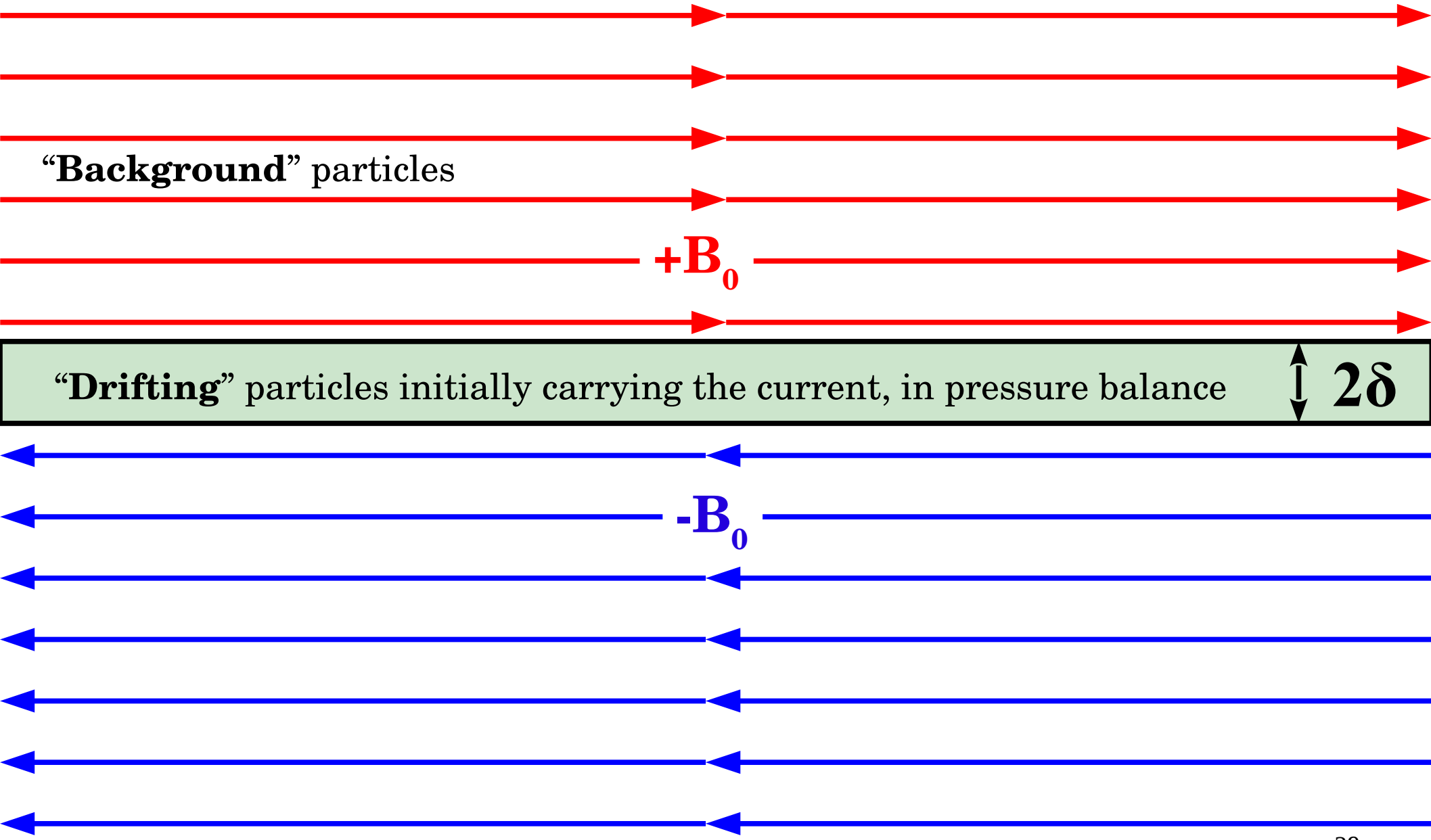
Gamma-ray burst[27]

This evening's talk!

Usual PIC setup: The Harris sheet



Usual PIC setup: The Harris sheet



The relativistic Harris solution

[See Kirk & Skjæraasen 2003]

The Harris solution is a **kinetic equilibrium** between the upstream **magnetic pressure** and a **hot plasma** concentrated inside the current sheet.

The solution:

The **reconnecting** field: $\boldsymbol{B} = B_0 \tanh\left(\frac{y}{\delta}\right)\boldsymbol{e}_x$

The "**drifting**" plasma density: $n_d = n_0 \cosh^{-2}\left(\frac{y}{\delta}\right)$ $\qquad n_0 = \frac{kT}{4\pi e^2 \Gamma_d \beta_d^2 \delta^2}$

Pressure balance condition across the sheet: $\frac{B_0^2}{8\pi} = n_d kT$

The "**background**" plasma: $n_b = \epsilon n_{0,} \quad (\epsilon \ll 1 \quad \text{for} \quad \sigma \gg 1)$

$$\sigma = \frac{B_0^2}{4\pi n_b m_e c^2} \gg 1$$

Particle initialization: Drifting Maxwellian

The **Harris** solution assumes a relativistic **drifting Maxwellian** distribution, i.e., the plasma has a relativistic motion of **Lorentz factor Γ_d.**

In the lab frame: $$f(\vec{u}) \propto \exp\left(\frac{-(\gamma' - 1)}{\theta}\right)$$

Where $$\theta = \frac{kT'}{mc^2} \qquad \gamma' = \Gamma_d\left(\gamma - \vec{\beta}_d \cdot \vec{u}\right)$$

Following *Swisdak (2013)*: $$\vec{u} = \vec{u}_{\parallel} + \vec{u}_{\perp}$$

Generation of $\mathbf{u}_{\parallel}$ via the cumulative distribution of $F(\mathbf{u}_{\parallel})$

$\mathbf{u}_{\perp}$ is generated, but cannot be chosen independently of $\mathbf{u}_{\parallel}$

The full procedure is in *mod_initial.f90*
Subroutines: SET_DRIFT_MAXWELLIAN, INIT_DRIFT_MAXWELLIAN, GEN_UP, GEN_PS

Initial numerical setup

e^+/e^- pairs