# Divergence Cleaning for MHD

賴伯熏 黃品睿 曾映舫 吳冠霖

# Overview

1.A (really) brief review on MHD and the project

2.How cleaning works

3.1D Riemann solver for MHD & Test Problems

4.2D Riemann solver & revision

5.MPI Parallelization

# Task Required

- Implement a divergence cleaning method for MHD - done!(in some sense)
- Compare the divergence errors with and without this correction - done!
- No need to adopt the constraint transport technique - done!
- Apply to a MHD test problem (e.g., MHD Riemann problems) - done!
- Measure the performance scaling - done!

# Ideal MHD Equations

$$\partial_t \rho + \nabla \cdot (\rho \mathbf{u}) = 0$$

$$\partial_t (\rho \mathbf{u}) + \nabla \cdot \left[ \rho \mathbf{u}\mathbf{u}^T + \left( p + \frac{1}{2}|\mathbf{B}|^2 \right) \mathcal{I} - \mathbf{B}\mathbf{B}^T \right] = 0$$

$$\partial_t \mathbf{B} + \nabla \cdot (\mathbf{u}\mathbf{B}^T - \mathbf{B}\mathbf{u}^T) = 0$$

$$\partial_t e + \nabla \cdot \left[ \left( e + p + \frac{1}{2}|\mathbf{B}|^2 \right) \mathbf{u} - \mathbf{B}(\mathbf{u} \cdot \mathbf{B}) \right] = 0$$

**With the additional constraint on B (also our favorite one)**

$$\nabla \cdot \mathbf{B} = 0$$

combine the divergence free constraint and the third equation,

$\partial t B + \nabla \times (B \times u) = 0$

and we have $\nabla \cdot (\nabla \times \cdot) \equiv 0$.

$$\frac{\partial}{\partial t} \boldsymbol{U} + \nabla \cdot \boldsymbol{F} = \boldsymbol{S}$$

# Divergence error

However!! For divergence to be discretized $\nabla \cdot \boldsymbol{B} \neq 0$

$\nabla \cdot$ B errors arise $\frac{\partial}{\partial t}(\nabla \cdot \mathbf{B}) = 0 + O(\Delta x^m, \Delta t^n)$

magnetic flux, momentum, and energy are not conserved

Effect:

Unphysical force

The magnetic topology is distorted

# Divergence Cleaning Method - GLM

Generalized Lagrange Multiplier - Boris(1971), Munz *et al*(2000)

Introducing a Lagrange multiplier to couple the divergence constraints

Operator-splitting procedure:

First step: Equations are solved numerically;

Second step: Divergence constraint is enforced by solving an possion equation

Divergence errors are transported to the boundary

$$\partial_t \mathbf{B} + \nabla \cdot (\mathbf{u}\mathbf{B}^T - \mathbf{B}\mathbf{u}^T) + \nabla\psi = 0$$

$$\mathcal{D}(\psi) + \nabla \cdot \mathbf{B} = 0$$

Motivated by C.-D. Munz 1999

# Advantage

Straightforward way

Efficient

Highly flexible

```
# GLM_hyperbolic
def hyperbolic_secondstep(flux, L, R):
    flux_hyp = flux.copy()
    b_xm = L[5] + 0.5*( R[5] - L[5]) - ( R[8] - L[8] ) / ( 2*ch )
    psi_m = L[8] + 0.5*( R[8] - L[8]) - 0.5*ch*( R[5] - L[5] )
    flux_hyp[8] = flux[8] + psi_m
    flux_hyp[5] = flux[5] + ch**2*b_xm

    return flux_hyp
```

$$\partial_t \rho + \nabla \cdot (\rho \mathbf{u}) = 0$$

$$\partial_t (\rho \mathbf{u}) + \nabla \cdot \left[ \rho \mathbf{u}\mathbf{u}^T + \left( p + \frac{1}{2}|\mathbf{B}|^2 \right) \mathcal{I} - \mathbf{B}\mathbf{B}^T \right] = 0$$

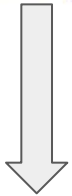
$$\partial_t \mathbf{B} + \nabla \cdot (\mathbf{u}\mathbf{B}^T - \mathbf{B}\mathbf{u}^T) + \nabla \psi = 0$$

$$\partial_t e + \nabla \cdot \left[ \left( e + p + \frac{1}{2}|\mathbf{B}|^2 \right) \mathbf{u} - \mathbf{B}(\mathbf{u} \cdot \mathbf{B}) \right] = 0$$

$$\mathcal{D}(\psi) + \nabla \cdot \mathbf{B} = 0$$

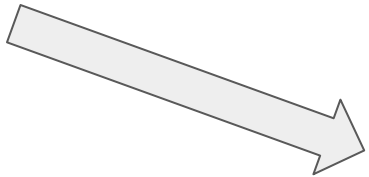
$$\partial_t \mathbf{B} + \nabla \cdot (\mathbf{u}\mathbf{B}^T - \mathbf{B}\mathbf{u}^T) + \nabla\psi = 0$$

$$\mathcal{D}(\psi) + \nabla \cdot \mathbf{B} = 0$$



$$\partial_t(\nabla \cdot \mathbf{B}) + \Delta\psi = 0$$

$$\partial_t \mathcal{D}(\psi) + \partial_t(\nabla \cdot \mathbf{B}) = 0$$



$$\partial_t \mathcal{D}(\psi) - \Delta\psi = 0$$

# Correction Methods - A Dedner *et al*(2002)

Proposed three different choices for the linear operator *D*.

$$\mathcal{D}(\psi) := 0 \qquad \Delta\psi = 0$$

$$\mathcal{D}(\psi) := \frac{1}{c_p^2}\psi \quad \text{with } c_p \in (0,\infty) \qquad \partial_t\psi - c_p^2\Delta\psi = 0$$

$$\mathcal{D}(\psi) := \frac{1}{c_h^2}\partial_t\psi \quad \text{with } c_h \in (0,\infty) \qquad \partial_{tt}^2\psi - c_h^2\Delta\psi = 0$$

$$\mathcal{D}(\psi) := \frac{1}{c_h^2}\partial_t\psi + \frac{1}{c_p^2}\psi \qquad \partial_{tt}^2\psi + \frac{c_h^2}{c_p^2}\partial_t\psi - c_h^2\Delta\psi = 0$$

$$\partial_t\mathcal{D}(\psi) - \Delta\psi = 0$$

# Elliptic $\mathcal{D}(\psi) := 0$

then ψ is just a Lagrange multiplier

$$\partial_t \mathbf{B} + \nabla \cdot (\mathbf{u}\mathbf{B}^T - \mathbf{B}\mathbf{u}^T) + \nabla\psi = 0, \qquad \partial_t(\nabla \cdot \mathbf{B}) + \Delta\psi = 0$$

- First step: solve the hyperbolic system (Resulting in B_n)
- 1.5 step : Solve the possion equation (Resulting in psi)

$$-\Delta\psi^{n^*} = \frac{1}{\Delta t_n}(\nabla \cdot \mathbf{B}^{n^*} - \nabla \cdot \mathbf{B}^n) = \frac{1}{\Delta t_n}\nabla \cdot \mathbf{B}^{n^*}.$$

- Second step:update the magnetic field with psi

$$\mathbf{B}^{n+1} = \mathbf{B}^{n^*} - \Delta t_n \nabla\psi^{n^*}.$$

# Elliptic $\mathcal{D}(\psi) := 0,$

**Projection method**

- Possesses elliptical character
- Bn+1 is simply the projection of Bn$*$ into the space of divergence-free vector fields.

# Parabolic

$$\mathcal{D}(\psi) := \frac{1}{c_p^2}\psi \quad \text{with } c_p \in (0, \infty),$$

$$\partial_t \psi - c_p^2 \Delta \psi = 0$$

**Heat Equation**

- Coupled with a strength
- Local divergence errors are dissipated and smoothed out

$$\partial_t \mathbf{B} + \nabla \cdot (\mathbf{u}\mathbf{B}^T - \mathbf{B}\mathbf{u}^T) = c_p^2 \nabla(\nabla \cdot \mathbf{B})$$

$$\partial_t \mathbf{B} + \nabla \cdot (\mathbf{u}\mathbf{B}^T - \mathbf{B}\mathbf{u}^T) + \nabla\psi = 0$$

$$\mathcal{D}(\psi) + \nabla \cdot \mathbf{B} = 0$$

$$\partial_t \rho + \nabla \cdot (\rho \mathbf{u}) = 0$$

$$\partial_t (\rho \mathbf{u}) + \nabla \cdot \left[ \rho \mathbf{u}\mathbf{u}^T + \left( p + \frac{1}{2}|\mathbf{B}|^2 \right) \mathcal{I} - \mathbf{B}\mathbf{B}^T \right] = 0$$

$$\partial_t \mathbf{B} + \nabla \cdot (\mathbf{u}\mathbf{B}^T - \mathbf{B}\mathbf{u}^T) + \nabla \psi = 0$$

$$\partial_t e + \nabla \cdot \left[ \left( e + p + \frac{1}{2}|\mathbf{B}|^2 \right) \mathbf{u} - \mathbf{B}(\mathbf{u} \cdot \mathbf{B}) \right] = 0$$

$$\mathcal{D}(\psi) + \nabla \cdot \mathbf{B} = 0$$

# Hyperbolic

$$\mathcal{D}(\psi) := \frac{1}{c_h^2}\partial_t \psi \quad \text{with } c_h \in (0, \infty)$$

$$\partial_{tt}^2 \psi - c_h^2 \Delta \psi = 0$$

$$\Delta t_n = c_{cfl} \min_{1 \le j \le N} \min_{1 \le l \le 3} \frac{h_{jl}}{c_h}.$$

**wave equation**

- local divergence error are propagated to the boundary with finite speed ch> 0

# Hyperbolic

$$\partial_t \rho + \partial_x(\rho u_x) = 0,$$

$$\partial_t(\rho u_x) + \partial_x\left(\rho u_x^2 + p + \frac{1}{2}\left(B_y^2 + B_z^2 - B_x^2\right)\right) = 0,$$

$$\partial_t(\rho u_y) + \partial_x(\rho u_x u_y - B_x B_y) = 0,$$

$$\partial_t(\rho u_z) + \partial_x(\rho u_x u_z - B_x B_z) = 0,$$

$$\partial_t B_x + \partial_x \psi = 0,$$

$$\partial_t B_y + \partial_x(B_y u_x - B_x u_y) = 0,$$

$$\partial_t B_z + \partial_x(B_z u_x - B_x u_z) = 0,$$

$$\partial_t e + \partial_x\left[u_x\left(e + p + \frac{1}{2}\mathbf{B}^2\right) - B_x(u_x B_x + u_y B_y + u_z B_z)\right] = 0,$$

$$\partial_t \psi + c_h^2 \partial_x B_x = 0,$$

$$\frac{\partial}{\partial t}\boldsymbol{U} + \frac{\partial}{\partial x}\boldsymbol{F} + \frac{\partial}{\partial y}\boldsymbol{G} + \frac{\partial}{\partial z}\boldsymbol{H} = 0$$

$$\partial_t \begin{pmatrix} B_x \\ \psi \end{pmatrix} + \begin{pmatrix} 0 & 1 \\ c_h^2 & 0 \end{pmatrix} \partial_x \begin{pmatrix} B_x \\ \psi \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \end{pmatrix}$$

Decoupled from the other seven equations

It's actually a **Riemann problem**

$$\begin{pmatrix} B_{x,m} \\ \psi_m \end{pmatrix} = \begin{pmatrix} B_{x,l} \\ \psi_l \end{pmatrix} + \begin{pmatrix} \frac{1}{2}(B_{x,r} - B_{x,l}) - \frac{1}{2c_h}(\psi_r - \psi_l) \\ \frac{1}{2}(\psi_r - \psi_l) - \frac{c_h}{2}(B_{x,r} - B_{x,l}) \end{pmatrix}$$

we obtain (ψm, c2 h Bx,m)T as numerical flux.

$$\mathbf{G}\left(\mathcal{R}(\mathbf{n}_{jl})\mathbf{U}_j^n, \mathcal{R}(\mathbf{n}_{jl})\mathbf{U}_{jl}^n\right)$$
$$:= \mathbf{G}_{mhd}\left(\mathcal{R}(\mathbf{n}_{jl})\mathbf{U}_j^n, \mathcal{R}(\mathbf{n}_{jl})\mathbf{U}_{jl}^n; B_{jl}^n\right) + \left(0, 0, 0, 0, \psi_{jl}^n, 0, 0, 0, c_h^2 B_{jl}^n\right)^T.$$

# Mixed

$$\mathcal{D}(\psi) := \frac{1}{c_h^2}\partial_t \psi + \frac{1}{c_p^2}\psi.$$

- Offers both dissipation and propagation of divergence errors

The “divergence constraint”takes the form $\partial_t \psi + c_h^2 \nabla \cdot \mathbf{B} = -\frac{c_h^2}{c_p^2}\psi.$

**Initial value problem**

First step :Solve the homogenous system

1.5step :solve the initial value problem, with ψj(0) = ψn*

$$\partial_t \psi_j = -\frac{c_h^2}{c_p^2}\psi_j \quad \Longrightarrow \quad \psi_j^{n+1} := e^{-\Delta t_n c_h^2/c_p^2}\psi_j^{n^*}.$$

Second step: $\mathbf{B}^{n+1} = \mathbf{B}^{n^*} - \Delta t_n \nabla \psi^{n^*}.$

## $c_p$ is still a free parameter

We can set it as a function of c_h.

1. $c_d := e^{-\Delta t_n c_h^2 / c_p^2}$ fix a constant $c_d \in (0, 1)$

$$c_p(c_d, c_h, \Delta t_n) = \sqrt{-\Delta t_n \frac{c_h^2}{\ln(c_d)}}.$$

2. $c_r := c_p^2 / c_h \in (0, \infty)$

   Mirrors the ratio between hyperbolic and parabolic effects

# MHD Riemann Solver in 1D

# Riemann Solvers

On our way to make a Riemann solver for MHD, we've tried the following solvers.

1. **Roe Solver**
2. **HLL - Einfeldt, et al (1991)**
3. **HLLC - Toro, et al (1994)**

These three solvers have been successfully implemented for 1D hydro problems.

For MHD, we choose to use the HLL solver.

# Riemann Solvers

$$\mathbf{U} = \begin{pmatrix} \rho \\ \rho u \\ \rho v \\ \rho w \\ B_y \\ B_z \end{pmatrix}, \quad \mathbf{F} = \begin{pmatrix} \rho u \\ \rho u^2 + p_T - B_x^2 \\ \rho v u - B_x B_y \\ \rho w u - B_x B_z \\ B_y u - B_x v \\ B_z u - B_x w \end{pmatrix}$$

# HLL Solver

Advantages

- No need to deal with the Roe matrix.
- The entropy condition is satisified.
- **Easier to code.**
- Because that's also what Athena did.
  (but HLLD is a bit too hard, so we just stayed with HLL)

| Choice | Comment | Physics | Reference |
|---|---|---|---|
| force | Toro's FORCE flux | Hydro and MHD | Toro section 7.4.2 |
| two-shock | Two-shock approximation | Hydro | Toro section 9.4.2 |
| exact | eact solver | Hydro | Toro chapter 4 |
| hlle | Harten-Lax-van Leer with Einfeldt fix | Hydro and MHD | Toro section 10.3 |
| hllc | Harten-Lax-van Leer with contact | Hydro | Toro section 10.4 |
| hlld | Harten-Lax-van Leer with contact and Alfven mode | MHD | Miyoshi & Kusano, JCP, 208, 305 |
| Roe | Roe's linearized solver | Hydro and MHD | Toro chapter 11 |

For **hydrodynamics**, use of the **Roe or HLLC solver** is strongly recommended.

For **MHD**, use of the **Roe or HLLD solver** is strongly recommended.

Taken from the Athena documentation

# (Really) Basic Ideas of HLL Solver

Roe : solve the exact solutions to the approximated equations.

HLL- : solve the approximated solutions to the original equations.

- HLL : Using **one intermediate state** to approximate the solutions.
- HLLC : Using **two intermediate states** to approximate the solutions.
- HLLD : Using **four intermediate states** to approximate the solutions.

We just stayed with the one state approximate.

$$\mathbf{U}(x,0) = \begin{cases} \mathbf{U}_\mathrm{l}, & \text{if } x < 0 \\ \mathbf{U}_\mathrm{r}, & \text{if } x > 0 \end{cases} \qquad \mathbf{U}_\mathrm{HLL} = \begin{cases} \mathbf{U}_\mathrm{l}, & \text{if } S_\mathrm{L} > 0, \\ \mathbf{U}^*, & \text{if } S_\mathrm{L} \leqslant 0 \leqslant S_\mathrm{R} \\ \mathbf{U}_\mathrm{r}, & \text{if } S_\mathrm{R} < 0, \end{cases}$$

where SL and SR are numerical approximations for the largest and smallest physical signal-velocities - B Einfeldt.(1991)

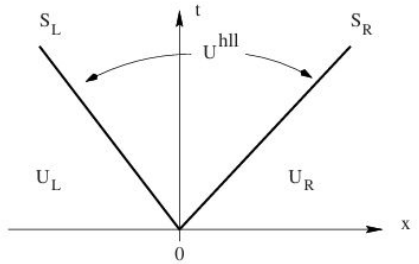



**Fig. 10.3.** Approximate HLL Riemann solver. Solution in the *Star Region* consists of a single state $\mathbf{U}^{hll}$ separated from data states by two waves of speeds $S_L$ and $S_R$.

E.F. Toro. (1999)

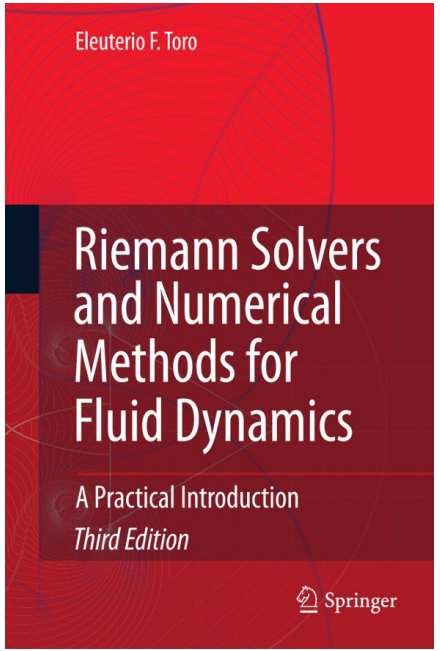
# (Really) Basic Ideas of HLL Solver

After some mathematics, physics, assumptions and approximations, we have the following formulas. Check out **E.F. Toro. (1999)** for more details.

$$\mathbf{U}^* = \frac{S_\mathrm{R}\mathbf{U}_\mathrm{r} - S_\mathrm{L}\mathbf{U}_\mathrm{l} - (F_\mathrm{r} - F_\mathrm{l})}{S_\mathrm{R} - S_\mathrm{L}} \qquad \mathbf{U}_\mathrm{HLL} = \begin{cases} \mathbf{U}_\mathrm{l}, & \text{if } S_\mathrm{L} > 0, \\ \mathbf{U}^*, & \text{if } S_\mathrm{L} \leqslant 0 \leqslant S_\mathrm{R} \\ \mathbf{U}_\mathrm{r}, & \text{if } S_\mathrm{R} < 0, \end{cases}$$

Or in terms of fluxes, L and R denote the states or fluxes of the left and right side.

$$\mathbf{F}^{hll}_{i+\frac{1}{2}} = \begin{cases} \mathbf{F}_L & \text{if} \quad 0 \leq S_L \ , \\ \dfrac{S_R\mathbf{F}_L - S_L\mathbf{F}_R + S_LS_R(\mathbf{U}_R - \mathbf{U}_L)}{S_R - S_L} & \text{if} \quad S_L \leq 0 \leq S_R \\ \mathbf{F}_R & \text{if} \quad 0 \geq S_R \ . \end{cases}$$



# On the Estimation of SL and SR

Final step is to find SL and SR. Several methods for estmating SL and SR exist, we adapt the one proposed by B Einfeldt, et al.(1991).

$$S_L = min\{b_L, 0\} \quad b_L = min\{\lambda_i, u_L - c_L\}$$

$$S_R = max\{b_R, 0\} \quad b_R = max\{\lambda_i, u_R + c_R\}$$

where the lambdas are the eigenvalues of **the Roe matrix**. CL and CR are the sound speed in the left and right state in hydro, but for MHD, we uses the "fast wave speed".

$$\lambda_{f_\pm} = u \pm c_f\,, \quad \lambda_{s_\pm} = u \pm c_s\,, \quad \lambda_{a_\pm} = u \pm c_a\,,$$

$$b_x := \sqrt{\frac{B_x^2}{\rho}}, \quad b^2 := \frac{|\mathbf{B}|^2}{\rho}, \quad a := \sqrt{\frac{\gamma p}{\rho}}$$

$$c_a = |b_x|,$$

$$c_f = \sqrt{\frac{1}{2}\left(a^2 + b^2 + \sqrt{(a^2+b^2)^2 - 4a^2b_x^2}\right)},$$

$$c_s = \sqrt{\frac{1}{2}\left(a^2 + b^2 - \sqrt{(a^2+b^2)^2 - 4a^2b_x^2}\right)}$$

# The Roe Matrix (The Fun Stuff)

The 7 eigenvalues correspond to 7 waves.

- Two fast magnetosonic waves - Cs
- Two fast magnetosonic waves - Cf
- One Alfvén wave - Ca

$$\begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ \delta_{21} & \delta_{22} & \delta_{23} & \delta_{24} & \delta_{25} & \delta_{26} & \delta_{27} \\ -\overline{u}\,\overline{v} & \overline{v} & \overline{u} & 0 & -B_x & 0 & 0 \\ -\overline{u}\,\overline{w} & \overline{w} & 0 & \overline{u} & 0 & -B_x & 0 \\ \frac{-\underline{B_y}}{\underline{\rho}}\overline{u} + \frac{B_x}{\underline{\rho}}\overline{v} & \frac{\underline{B_y}}{\underline{\rho}} & \frac{-B_x}{\underline{\rho}} & 0 & \overline{u} & 0 & 0 \\ \frac{-\underline{B_z}}{\underline{\rho}}\overline{u} + \frac{B_x}{\underline{\rho}}\overline{w} & \frac{\underline{B_z}}{\underline{\rho}} & 0 & \frac{-B_x}{\underline{\rho}} & 0 & \overline{u} & 0 \\ \delta_{71} & \delta_{72} & \delta_{73} & \delta_{74} & \delta_{75} & \delta_{76} & \delta_{77} \end{bmatrix}$$

$$\delta_{21} = -\overline{u}^2 + (2-\gamma)X + \frac{\gamma-1}{2}\overline{\boldsymbol{V}}^2,\ \delta_{22} = 2\overline{u} - (\gamma-1)\overline{u}$$

$$\delta_{23} = -(\gamma-1)\overline{v},\ \delta_{24} = -(\gamma-1)\overline{w}$$

$$\delta_{25} = (2-\gamma)\underline{B_y},\ \delta_{26} = (2-\gamma)\underline{B_z},\ \delta_{27} = \gamma - 1$$

$$\delta_{71} = -\overline{u}\overline{H^*} + \overline{u}(\delta_{21} + \overline{u}^2) + \frac{B_x}{\underline{\rho}}(\overline{\boldsymbol{V}}\cdot\underline{\boldsymbol{B}})$$

$$\delta_{72} = \overline{H^*} + \overline{u}(\delta_{22} - 2\overline{u}) - \frac{B_x^2}{\underline{\rho}},\ \delta_{73} = \overline{u}\delta_{23} - \frac{B_x}{\underline{\rho}}\underline{B_y},$$

$$\delta_{74} = \overline{u}\delta_{24} - \frac{B_x}{\underline{\rho}}\underline{B_z}$$

$$\delta_{75} = \overline{u}\delta_{25} - B_x\overline{v},\ \delta_{76} = \overline{u}\delta_{26} - B_x\overline{w},\ \delta_{77} = \overline{u} + \overline{u}\delta_{27}$$

P Cargo. et al(1997)

# Roe-averaged Values

The wave speeds are calculated using Roe-averaged values. B and u are **vectors**.

$$\bar{u} = \frac{\sqrt{\rho_L}u_L + \sqrt{\rho_R}u_R}{\sqrt{\rho_L} + \sqrt{\rho_R}}$$

$$\bar{H} = \frac{\sqrt{\rho_L}H_L + \sqrt{\rho_R}H_R}{\sqrt{\rho_L} + \sqrt{\rho_R}}$$

$$\bar{B} = \frac{\sqrt{\rho_L}B_L + \sqrt{\rho_R}B_R}{\sqrt{\rho_L} + \sqrt{\rho_R}}$$

$$\bar{\rho} = \sqrt{\rho_L}\sqrt{\rho_R}$$

Recall that:

$$\lambda_{f_\pm} = u \pm c_f\,, \quad \lambda_{s_\pm} = u \pm c_s\,, \quad \lambda_{a_\pm} = u \pm c_a\,,$$

$$c_a = |b_x|,$$

$$c_f = \sqrt{\frac{1}{2}\left(a^2 + b^2 + \sqrt{(a^2+b^2)^2 - 4a^2b_x^2}\right)},$$

$$c_s = \sqrt{\frac{1}{2}\left(a^2 + b^2 - \sqrt{(a^2+b^2)^2 - 4a^2b_x^2}\right)}$$

$$b_x := \sqrt{\frac{B_x^2}{\rho}}, \quad b^2 := \frac{|\mathbf{B}|^2}{\rho}, \quad a := \sqrt{\frac{\gamma p}{\rho}}$$

# Python Code for HLL

See the function **“HLL”** in our code for details. I’ll just go through the process here.

- Given the left and right state and compute the Roe-averaged values.
- Use the Roe-averaged values to compute the eigenvalues.
- Estimate SL and SR.
- Determine which state to use using SL and SR.
- Compute the HLL-estimated flux.
- Return flux and update.(we use MUSCL-Hancock, van-leer limiter and PLM)

$$\mathbf{U}_{\mathrm{HLL}} = \begin{cases} \mathbf{U}_{\mathrm{l}}, & \text{if } S_{\mathrm{L}} > 0, \\ \mathbf{U}^{*}, & \text{if } S_{\mathrm{L}} \leqslant 0 \leqslant S_{\mathrm{R}} \\ \mathbf{U}_{\mathrm{r}}, & \text{if } S_{\mathrm{R}} < 0, \end{cases} \qquad \mathbf{F}_{i+\frac{1}{2}}^{hll} = \begin{cases} \mathbf{F}_L & \text{if} \quad 0 \leq S_L\,, \\ \dfrac{S_R\mathbf{F}_L - S_L\mathbf{F}_R + S_L S_R(\mathbf{U}_R - \mathbf{U}_L)}{S_R - S_L}\,, & \text{if} \quad S_L \leq 0 \leq S_R \\ \mathbf{F}_R & \text{if} \quad 0 \geq S_R\,. \end{cases}$$

# Test Problems for Our Solver

# Test Problems

To verify if our solver yields correct result, we use several test problems, including hydro and MHD problems.

- **Hydro: Sod's Shock Tube**
- **Hydro: Strong Shock Problem**
- **MHD: Brio-Wu Shock Tube**

We also verify the conserved quantities (Brio-Wu Shock Tube).

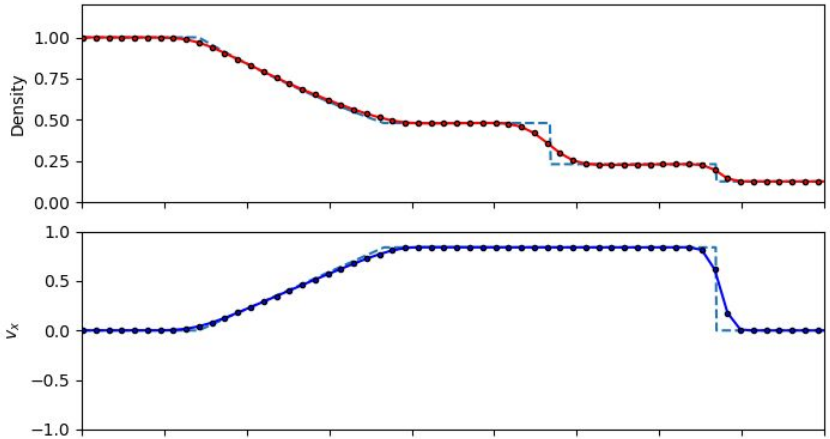
- **Mass conservation**
- **Energy conservation**

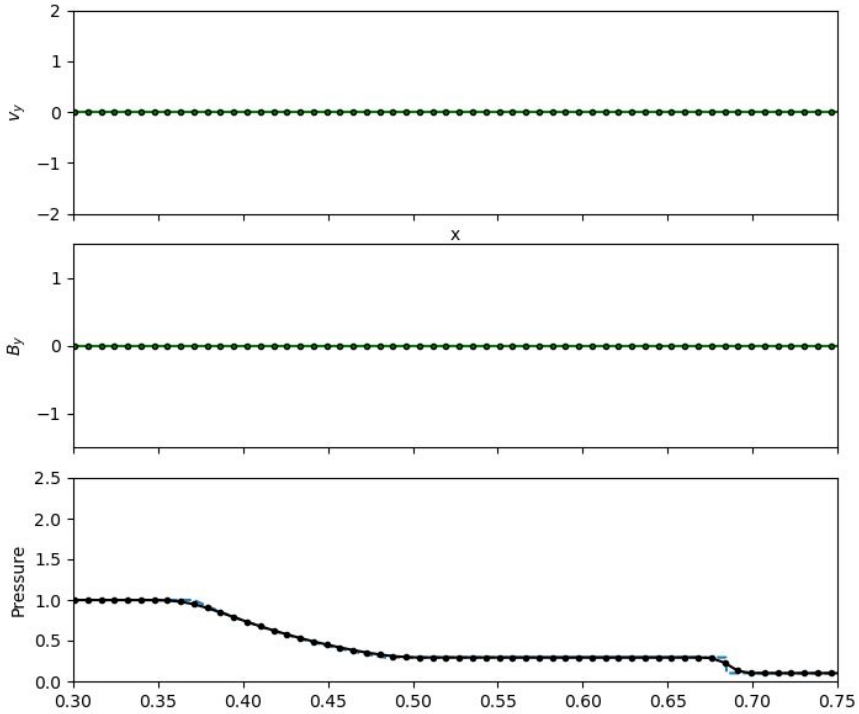
Lastly, we will show the performance scaling (with some unsolved mysteries).

# Sod's Shock Tube

## Hydro test problem. (BC: Outflow)

- Left state: ρ = 1.0, **u** = 0.0, **B** = 0.0, P = 1.0
- Right state: ρ = 0.125, **u** = 0.0, **B** = 0.0, P = 0.1
- End time = 0.1, cfl = 0.475, 128 cells were used

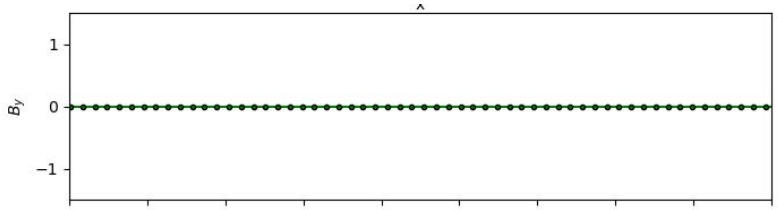


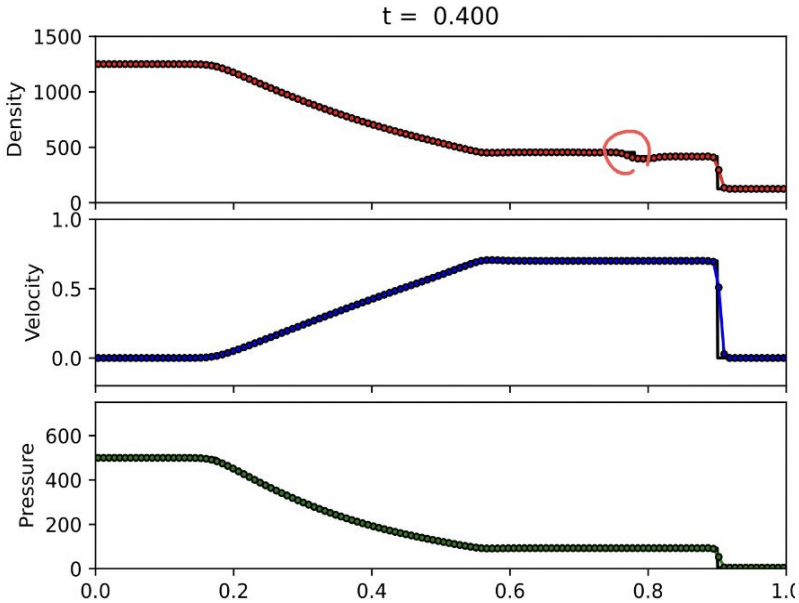



# Strong Shock Tube

## Hydro test problem. (BC: Outflow)

- Left state: ρ = 1.25e+3, **u** = 0.0, **B** = 0.0, P = 5.0e+2
- Right state: ρ = 1.25e+2, **u** = 0.0, **B** = 0.0, P = 5.0
- End time = 0.4, cfl = 0.475, 128 cells were used.

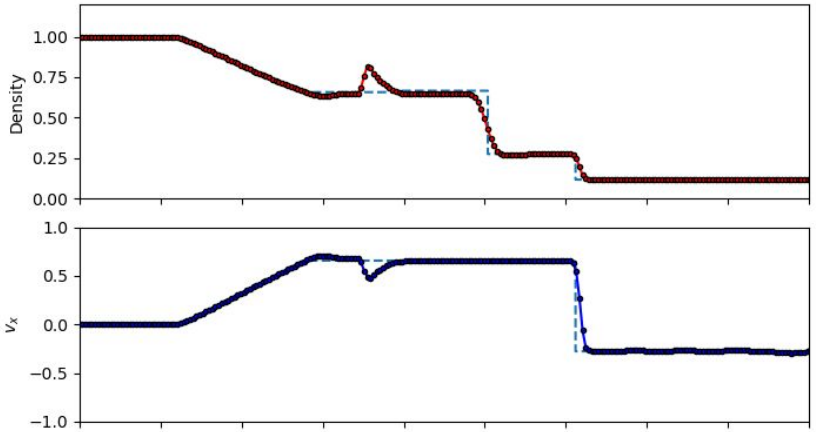


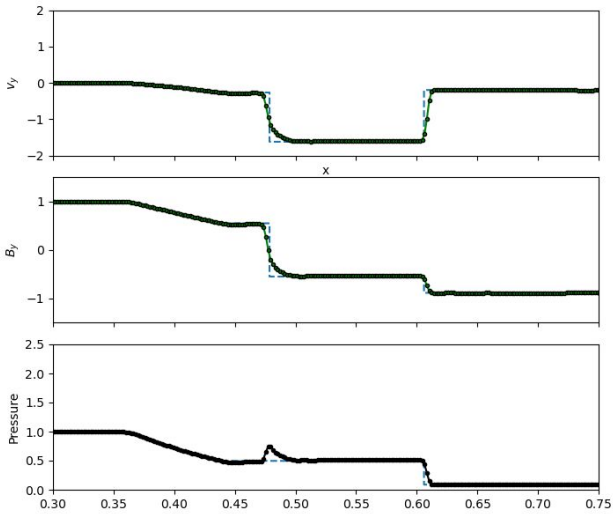



# Brio-Wu shock tube

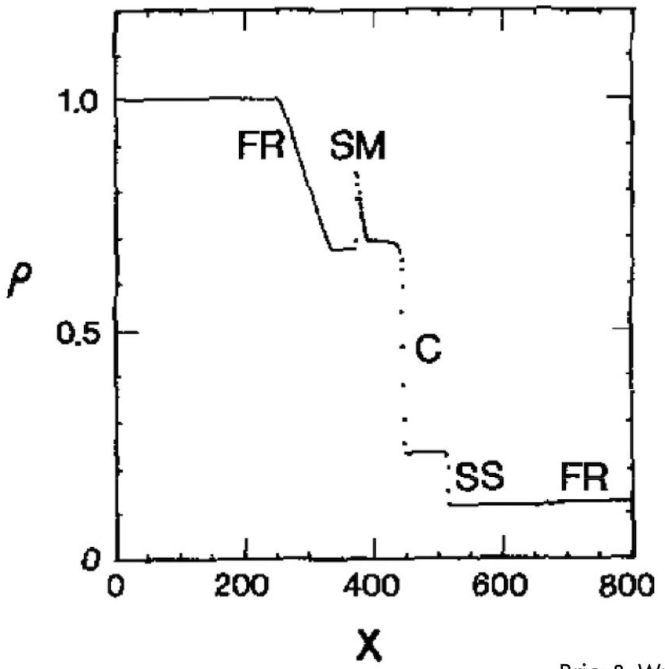
## MHD test problem. (BC: Dirichlet)

- Left state: ρ = 1.0, **u** = 0.0, **B** = (0.75, 1.0, 0.0), P = 1.0
- Right state: ρ = 0.125, **u** = 0.0, **B** = (0.75, -1.0, 0.0), P = 0.1
- End time = 0.08, cfl = 0.475, 512 cells were used.

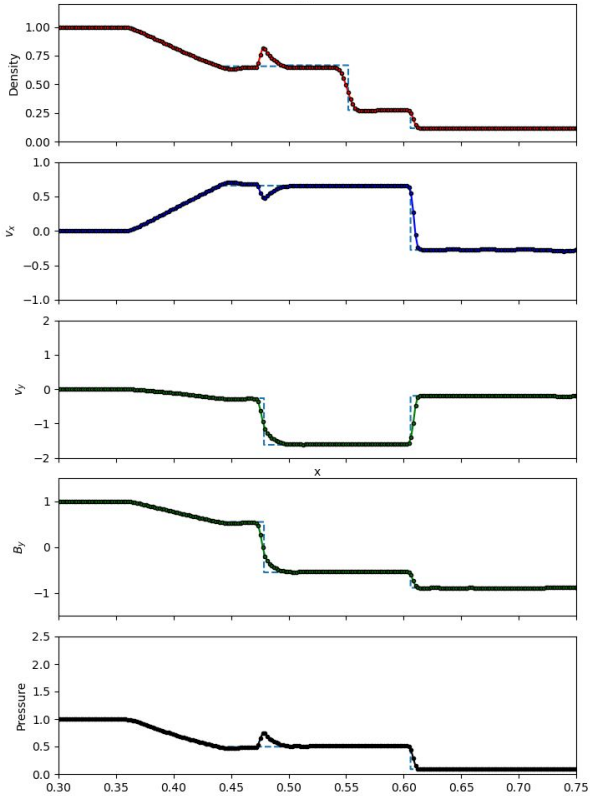


# Brio-Wu shock tube



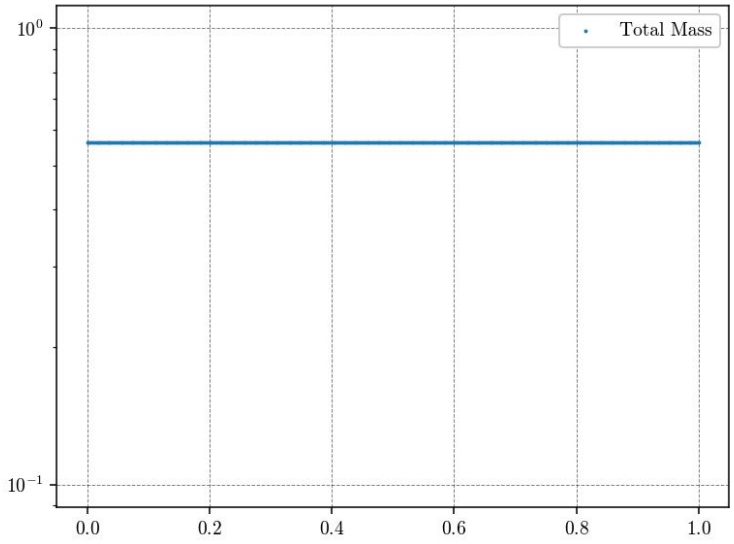

Brio & Wu 1988

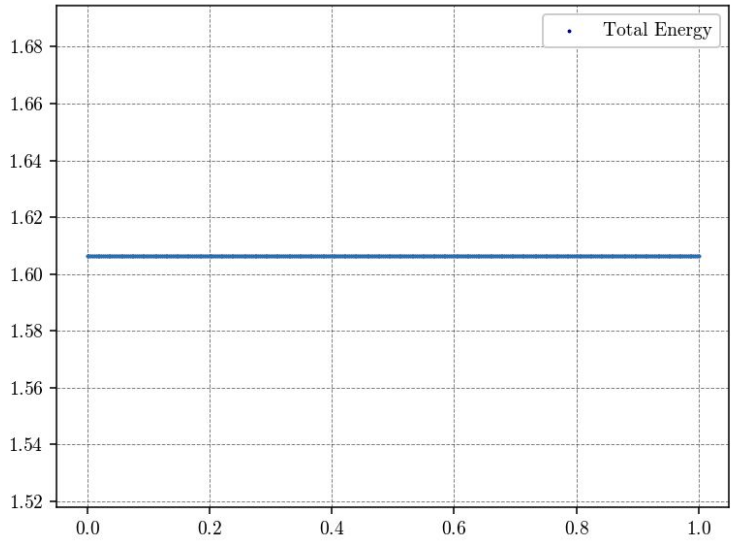


# Conserved Quantities

Using the Brio-Wu tube as an example. The result seems a bit too good.

The energy error has an order of magnitude of -16 (double precision).

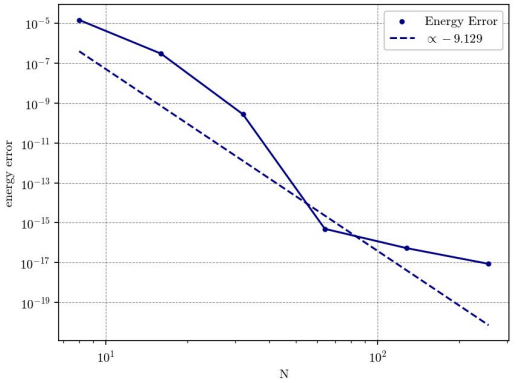


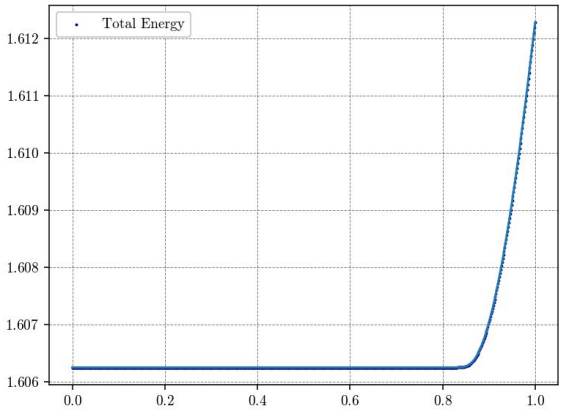



# Performance Scaling

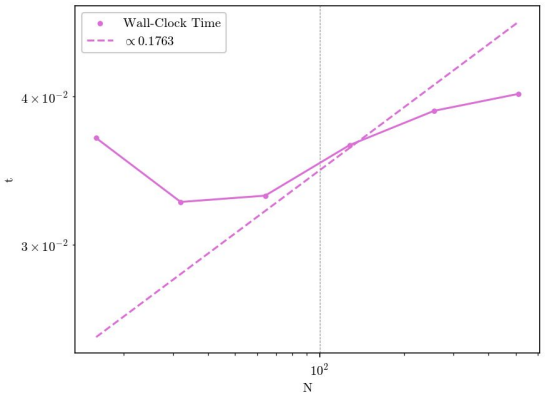
Using the Brio-Wu tube as an example. Endtime = 0.16.

- In theory, MUSCL-Hancock should give us a **second-order accuracy**.
- The error decrease with the number of computing cells(**ninth-order ?**).
- The wall-clock is roughly the same, despite the increasing number of cells.
- The error starts increasing after t=0.8. (due to Dirichlet boundary conditions).



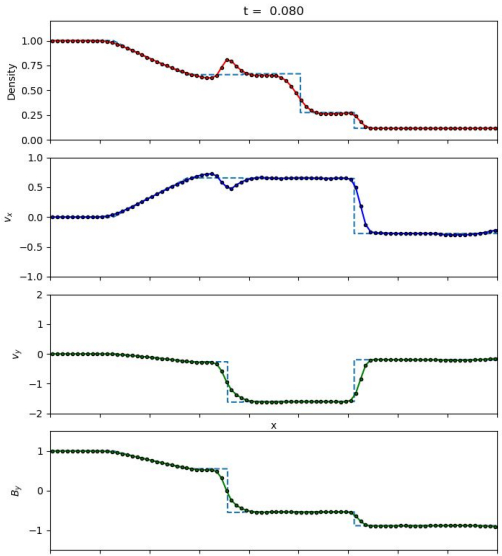

$$Error = \left| \frac{E - E_0}{E_0} \right|$$



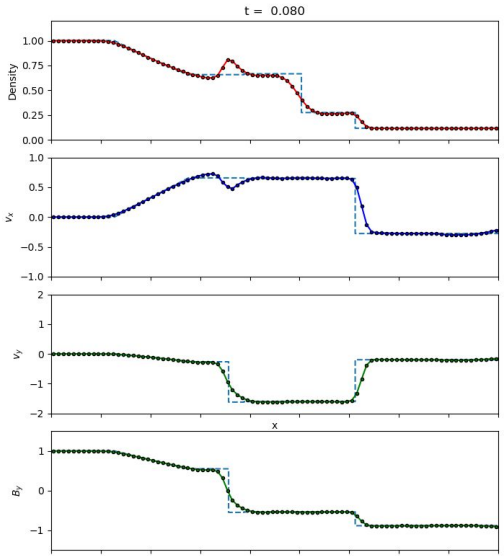
# Compare the Divergence Errors

**But there's no error.** With or without the cleaning, we all obtained the same result.

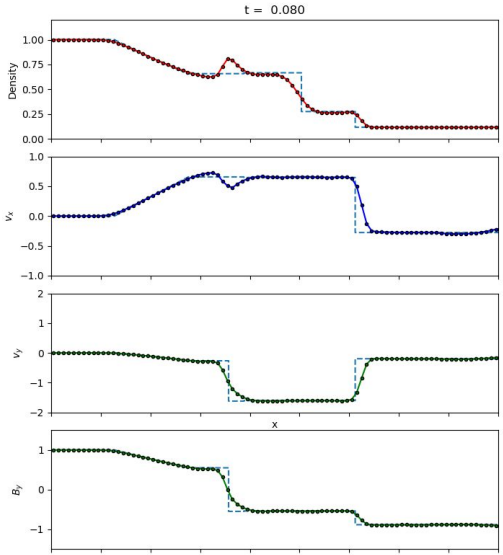
Since the divergence free condition **is trivially satisfied in 1D** …… (We have more on this later.)




**No correction**




**Hyperbolic correction**




**Mixed correction**

# 2D MHD ?

$$\mathbf{U}_t + \mathbf{G}(\mathbf{U})_y = 0$$

$$\mathbf{U} = \begin{Bmatrix} \rho \\ \rho u \\ \rho v \\ E \end{Bmatrix}; \mathbf{G}(\mathbf{U}) = \begin{Bmatrix} \rho v \\ \rho uv \\ \rho v^2 + p \\ v(E+p) \end{Bmatrix}$$

$$\mathbf{U}_B = \bar{\mathbf{U}}_n$$
$$\mathbf{U}_T = \bar{\mathbf{U}}_s$$
$$\mathbf{G}_B = \mathbf{G}(\bar{\mathbf{U}}_n)$$
$$\mathbf{G}_T = \mathbf{G}(\bar{\mathbf{U}}_s)$$

$$s_B = \min(v_B - a_B, v_T - a_T)$$
$$s_T = \min(v_B + a_B, v_T + a_T)$$

$$\mathbf{G} = \begin{cases} \mathbf{G}_B & \text{if } 0 \le s_B \\ \dfrac{s_T\mathbf{G}_B - s_B\mathbf{G}_T + s_B s_T(\mathbf{U}_T - \mathbf{U}_B)}{s_T - s_B} & \text{if } s_B \le 0 \le s_T \\ \mathbf{G}_T & \text{if } 0 \ge s_T \end{cases}$$

$$\mathbf{U}_t + \mathbf{F}(\mathbf{U})_x = 0$$

$$\mathbf{U} = \begin{Bmatrix} \rho \\ \rho u \\ \rho v \\ E \end{Bmatrix}; \mathbf{F}(\mathbf{U}) = \begin{Bmatrix} \rho u \\ \rho u^2 + p \\ \rho uv \\ u(E+p) \end{Bmatrix}$$

$$\mathbf{U}_L = \bar{\mathbf{U}}_e$$
$$\mathbf{U}_R = \bar{\mathbf{U}}_w$$
$$\mathbf{F}_L = \mathbf{F}(\bar{\mathbf{U}}_e)$$
$$\mathbf{F}_R = \mathbf{F}(\bar{\mathbf{U}}_w)$$

$$s_L = \min(u_L - a_L, u_R - a_R)$$
$$s_R = \min(u_L + a_L, u_R + a_R)$$

$$\mathbf{F} = \begin{cases} \mathbf{F}_L & \text{if } 0 \le s_L \\ \dfrac{s_R\mathbf{F}_L - s_L\mathbf{F}_R + s_L s_R(\mathbf{U}_R - \mathbf{U}_L)}{s_R - s_L} & \text{if } s_L \le 0 \le s_R \\ \mathbf{F}_R & \text{if } 0 \ge s_R \end{cases}$$

# 2D MHD ?

HLL in 2D - DS Balsara(2011)

$$\partial_t \mathbf{U} + \partial_x \mathbf{F} + \partial_y \mathbf{G} = 0$$

$$\mathbf{F}^* = 2\,\mathbf{U}^*\,S_R - \frac{S_U}{S_U - S_D}\mathbf{F}_U^{\mathrm{HLL}} + \frac{S_D}{S_U - S_D}\mathbf{F}_D^{\mathrm{HLL}}$$

$$+ 2\,\frac{\mathbf{F}_{RU}\,S_U - \mathbf{F}_{RD}\,S_D + \left(\mathbf{G}_{RU} - \mathbf{G}_{RD}\right)S_R - \left(\mathbf{U}_{RU}\,S_U - \mathbf{U}_{RD}\,S_D\right)S_R}{S_U - S_D}$$

$$+ \frac{\left(\mathbf{F}_{RD} - \mathbf{F}_R^*\right)S_D^R - \left(\mathbf{F}_{RU} - \mathbf{F}_R^*\right)S_U^R - \left(\mathbf{G}_{RU} - \mathbf{G}_U^*\right)S_R^{U+} + \left(\mathbf{G}_{LU} - \mathbf{G}_U^*\right)S_L^{U+} + \left(\mathbf{G}_{RD} - \mathbf{G}_D^*\right)S_R^{D+} - \left(\mathbf{G}_{LD} - \mathbf{G}_D^*\right)S_L^{D+}}{S_U - S_D}$$

$$S_R^U = \max\left(\lambda_x^N(\mathbf{U}_{RU}), \bar{\lambda}_x^N(\mathbf{U}_{LU}, \mathbf{U}_{RU})\right) \quad ; \quad S_L^U = \min\left(\lambda_x^1(\mathbf{U}_{LU}), \bar{\lambda}_x^1(\mathbf{U}_{LU}, \mathbf{U}_{RU})\right) \quad S_U^R = \max\left(\lambda_y^N(\mathbf{U}_{RU}), \bar{\lambda}_y^N(\mathbf{U}_{RD}, \mathbf{U}_{RU})\right) \quad ; \quad S_D^R = \min\left(\lambda_y^1(\mathbf{U}_{RD}), \bar{\lambda}_y^1(\mathbf{U}_{RD}, \mathbf{U}_{RU})\right)$$

$$S_R^D = \max\left(\lambda_x^N(\mathbf{U}_{RD}), \bar{\lambda}_x^N(\mathbf{U}_{LD}, \mathbf{U}_{RD})\right) \quad ; \quad S_L^D = \min\left(\lambda_x^1(\mathbf{U}_{LD}), \bar{\lambda}_x^1(\mathbf{U}_{LD}, \mathbf{U}_{RD})\right) \quad S_U^L = \max\left(\lambda_y^N(\mathbf{U}_{LU}), \bar{\lambda}_y^N(\mathbf{U}_{LD}, \mathbf{U}_{LU})\right) \quad ; \quad S_D^L = \min\left(\lambda_y^1(\mathbf{U}_{LD}), \bar{\lambda}_y^1(\mathbf{U}_{LD}, \mathbf{U}_{LU})\right)$$

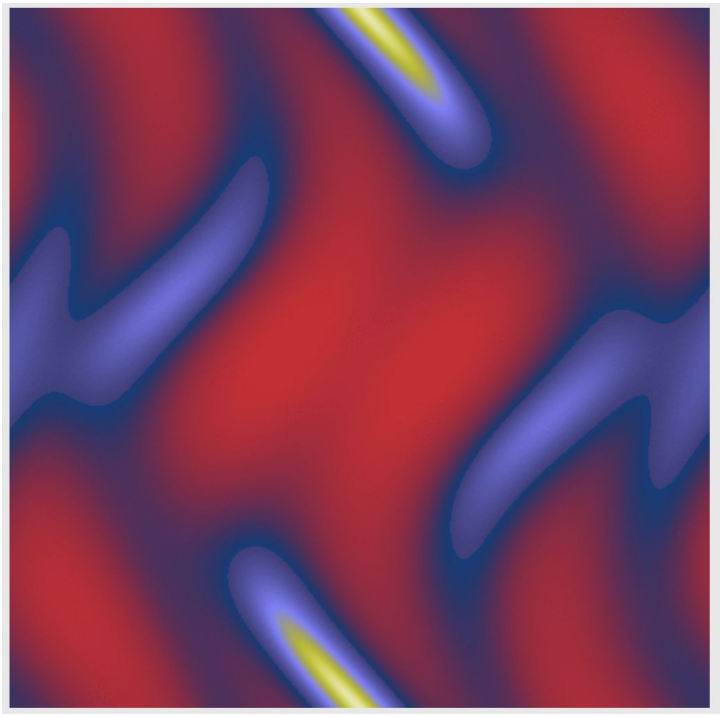
$$S_R = \max\left(S_R^U, S_R^D\right) \quad ; \quad S_L = \min\left(S_L^U, S_L^D\right) \qquad S_U = \max\left(S_U^R, S_U^L\right) \quad ; \quad S_D = \min\left(S_D^R, S_D^L\right)$$

# Orszag-Tang Vortex

**2D test problem. (BC: Periodic)**

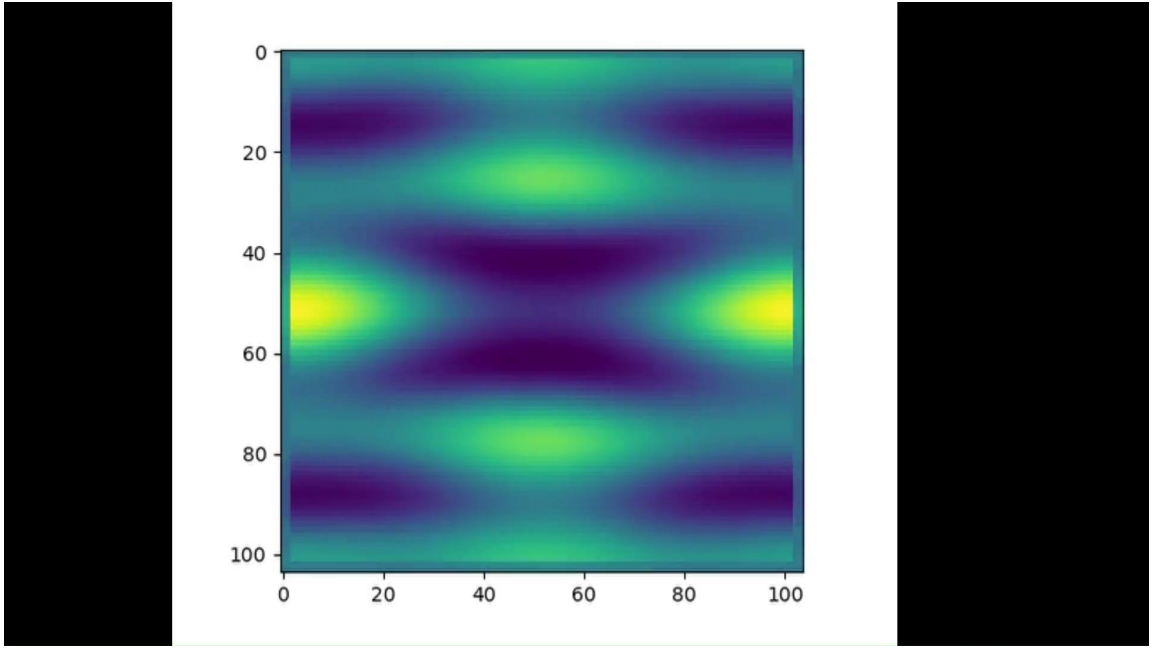
$$\rho(\mathbf{r},0) = \frac{\gamma^2}{4\pi},$$
$$p(\mathbf{r},0) = \frac{\gamma}{4\pi},$$
$$\mathbf{v}(\mathbf{r},0) = (-\sin(2\pi y), \sin(2\pi x)),$$
$$\mathbf{B}(\mathbf{r},0) = \frac{1}{2\pi\sqrt{4\pi}} \nabla_\perp \left( \frac{\cos(4\pi x)}{2} + \cos(2\pi y) \right)$$

Negative pressure ?!



openMHD

# Current Progress

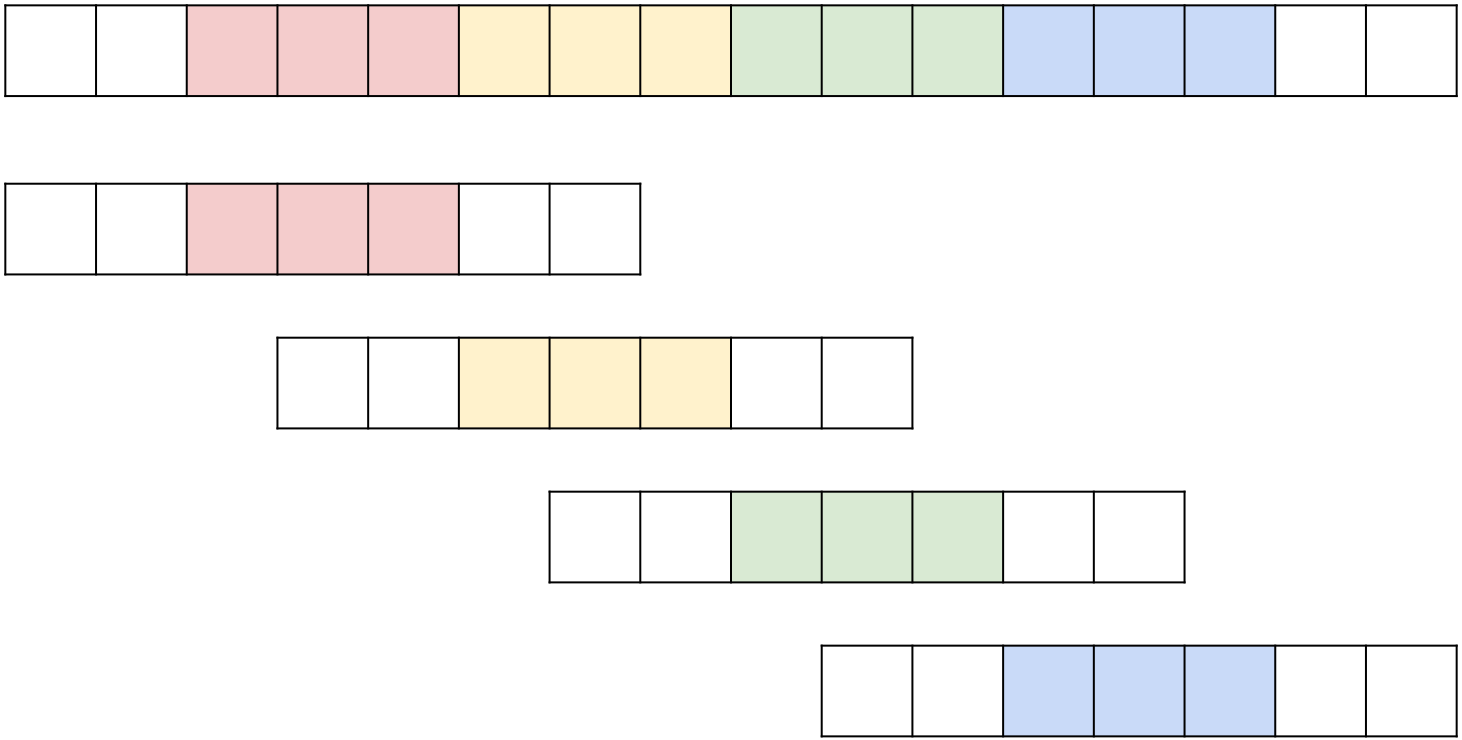


# MPI Parallelization

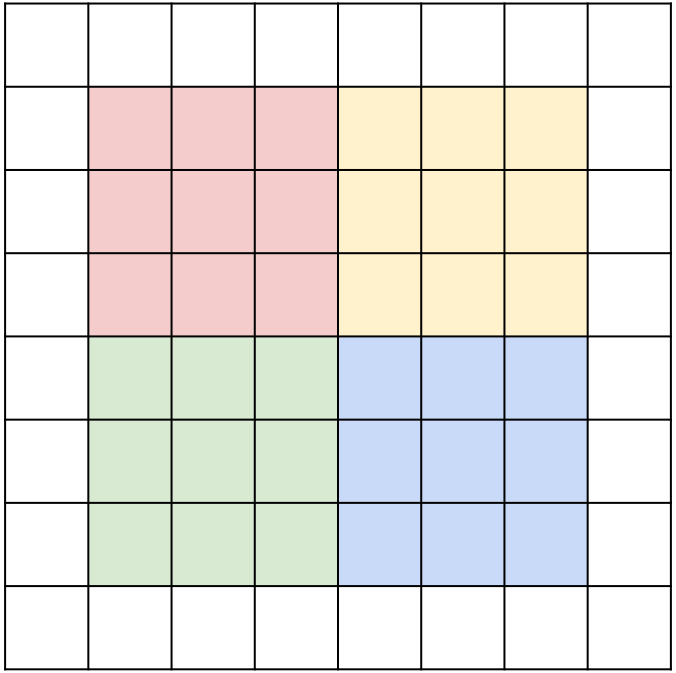
# Syntax

- comm = MPI.COMM_WORLD
- rank = comm.Get_rank()
- nrank = comm.Get_size()
- comm.Send()
- comm.Recv()
- comm.gather()
- comm.bcast()

# Split cells - 1D

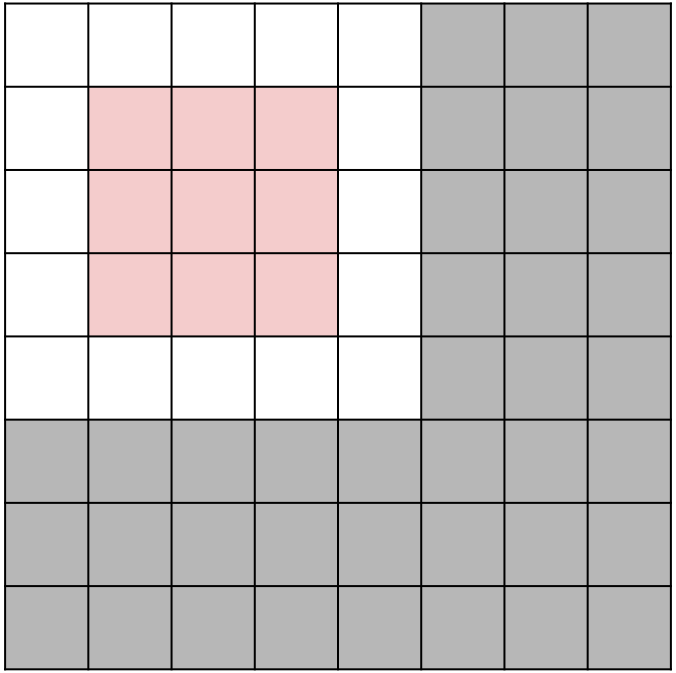


# Split cells - 2D

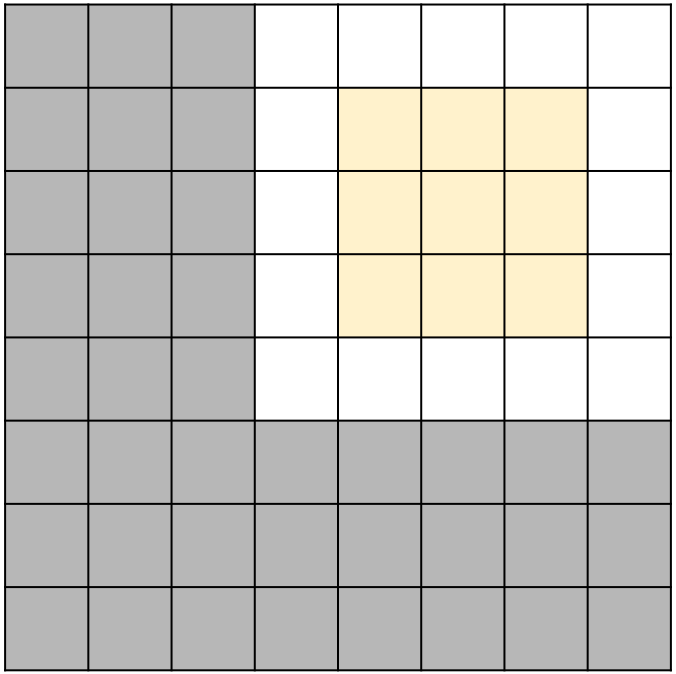


# Split cells - 2D

Rank = 0



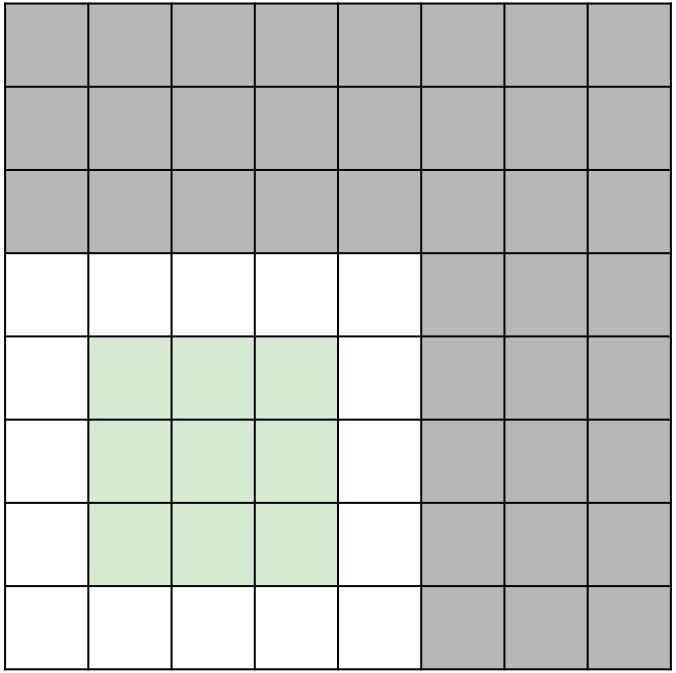
# Split cells - 2D



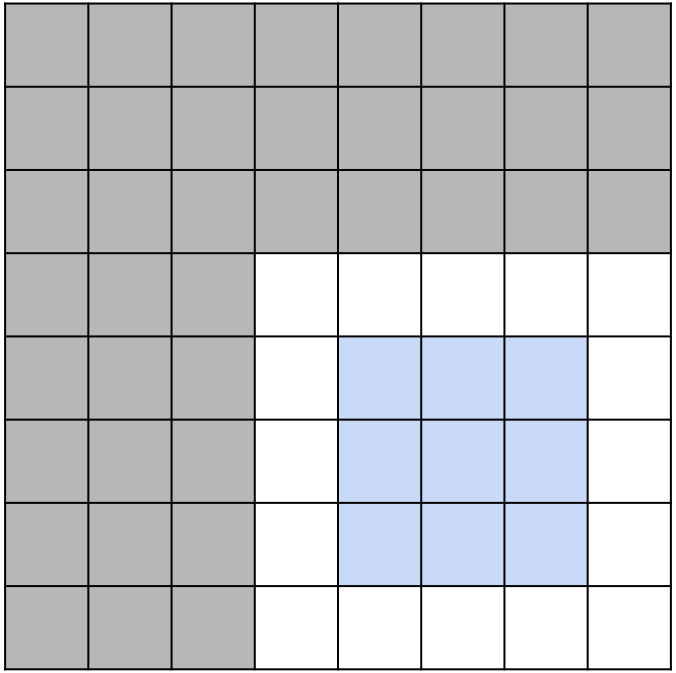
Rank = 1

# Split cells - 2D

Rank = 2

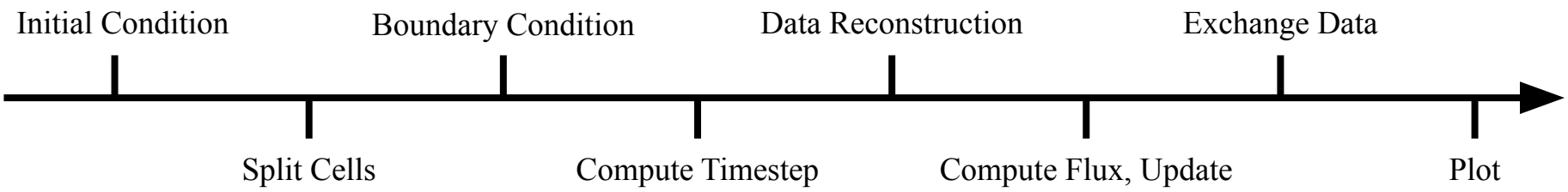


# Split cells - 2D

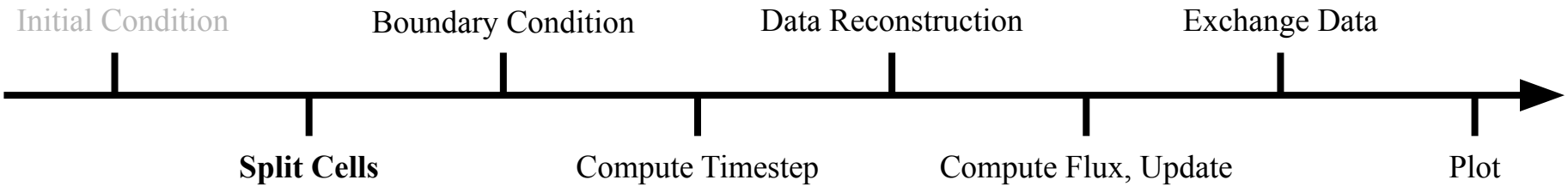


Rank = 3

# Modifications



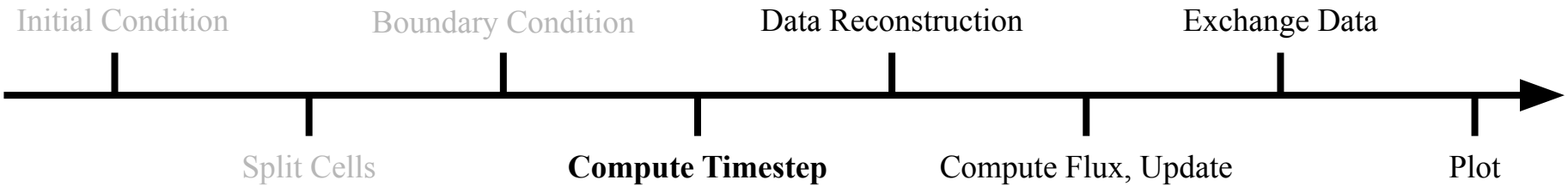
# Modifications




```
u = np.empty((n,9))
U[0:nghost] = U[nghost]
U[N-nghost:N] = U[N-nghost-1]
u = U[rank*n_In:(rank+1)*n_In+4]
```

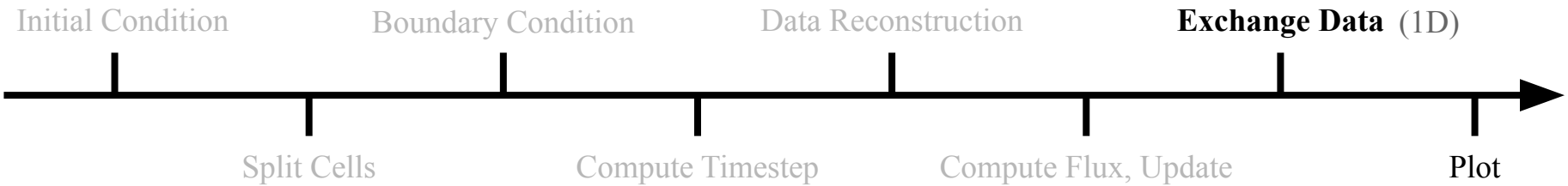
```
u = np.empty((n,n,9))
if rank == 0:
    u = U[:n,:n]
elif rank == 1:
    u = U[:n,-n:]
elif rank == 2:
    u = U[-n:,:n]
elif rank == 3:
    u = U[-n:,-n:]
```

# Modifications




```
max_info_speed = np.amax( u + cf )
dt_cfl         = cfl*dx/max_info_speed
dt_end         = end_time - t

dt_l = comm.gather(dt_cfl, root = 0)
DT = comm.bcast(dt_l, root = 0)
DT.append(dt_end)
dt = min(DT)
```

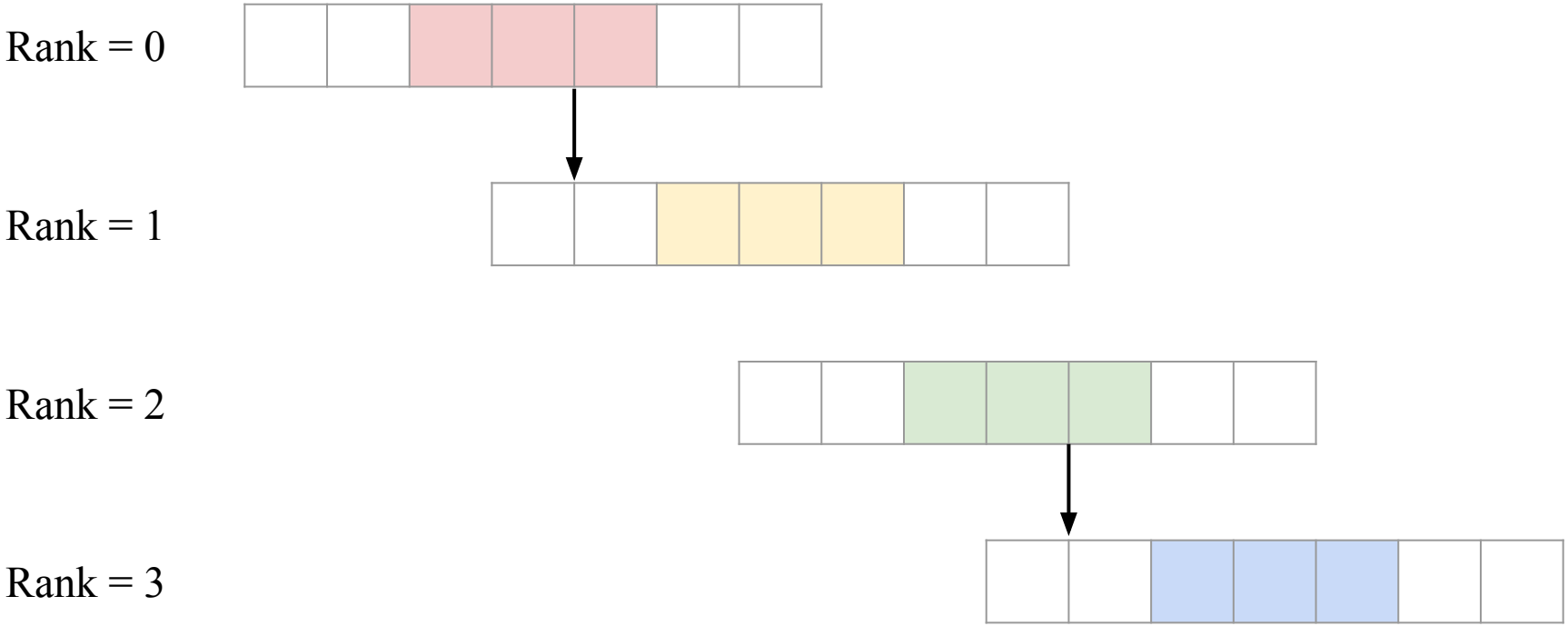
# Modifications



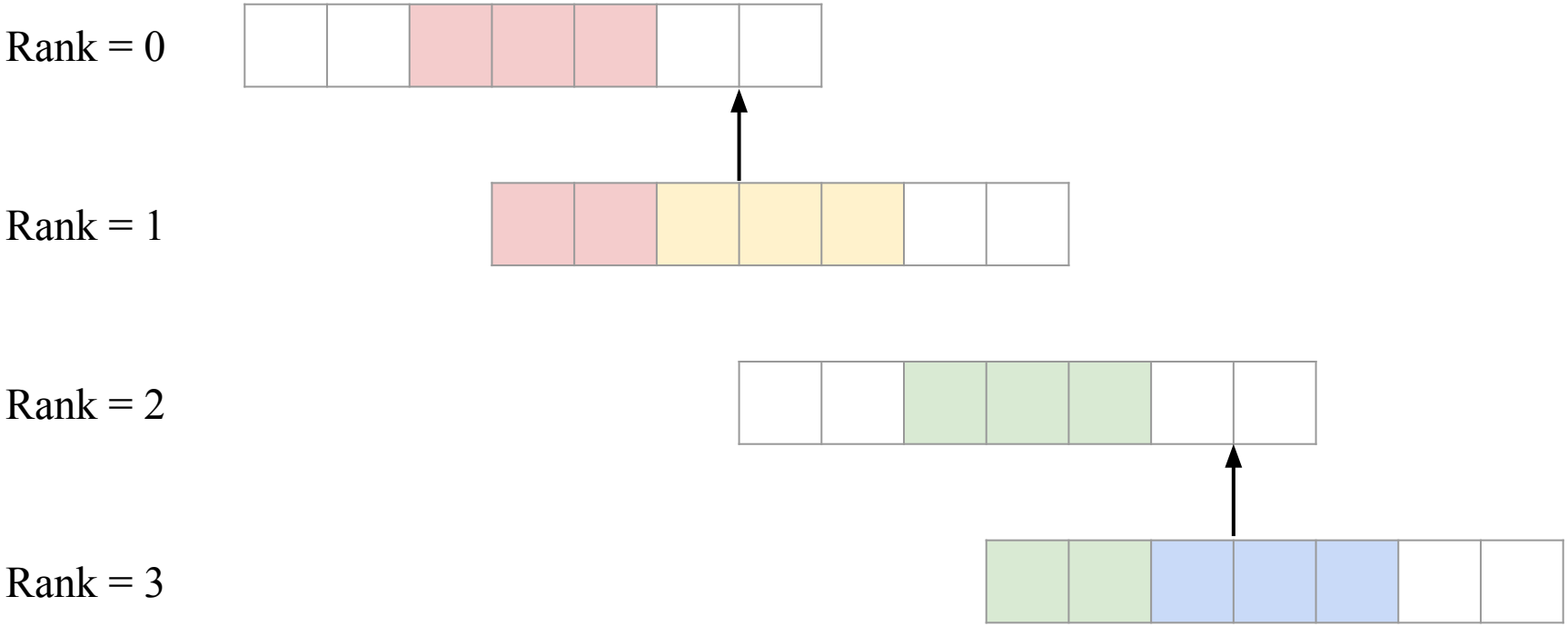

```
recv_l = np.empty((nghost,9))
recv_r = np.empty((nghost,9))
send_l = u[nghost:nghost+nghost]
send_r = u[n-nghost-nghost:n-nghost]

if rank%2 == 0:
    comm.Send(send_r, rank+1, tag = it+rank)
    comm.Recv(recv_r, rank+1, tag = it+rank+1)
    bd_r = recv_r
else:
    comm.Recv(recv_l, rank-1, tag = it+rank-1)
    comm.Send(send_l, rank-1, tag = it+rank)
    bd_l = recv_l
```

```
if rank != 0 and rank != nrank-1:
    if rank%2 == 0:
        comm.Send(send_l, rank-1, tag = it+rank)
        comm.Recv(recv_l, rank-1, tag = it+rank-1)
        bd_l = recv_l
    else:
        comm.Recv(recv_r, rank+1, tag = it+rank+1)
        comm.Send(send_r, rank+1, tag = it+rank)
        bd_r = recv_r
elif rank == 0:
    bd_l = u[0:nghost]
elif rank == nrank-1:
    bd_r = u[n-nghost:n]
```

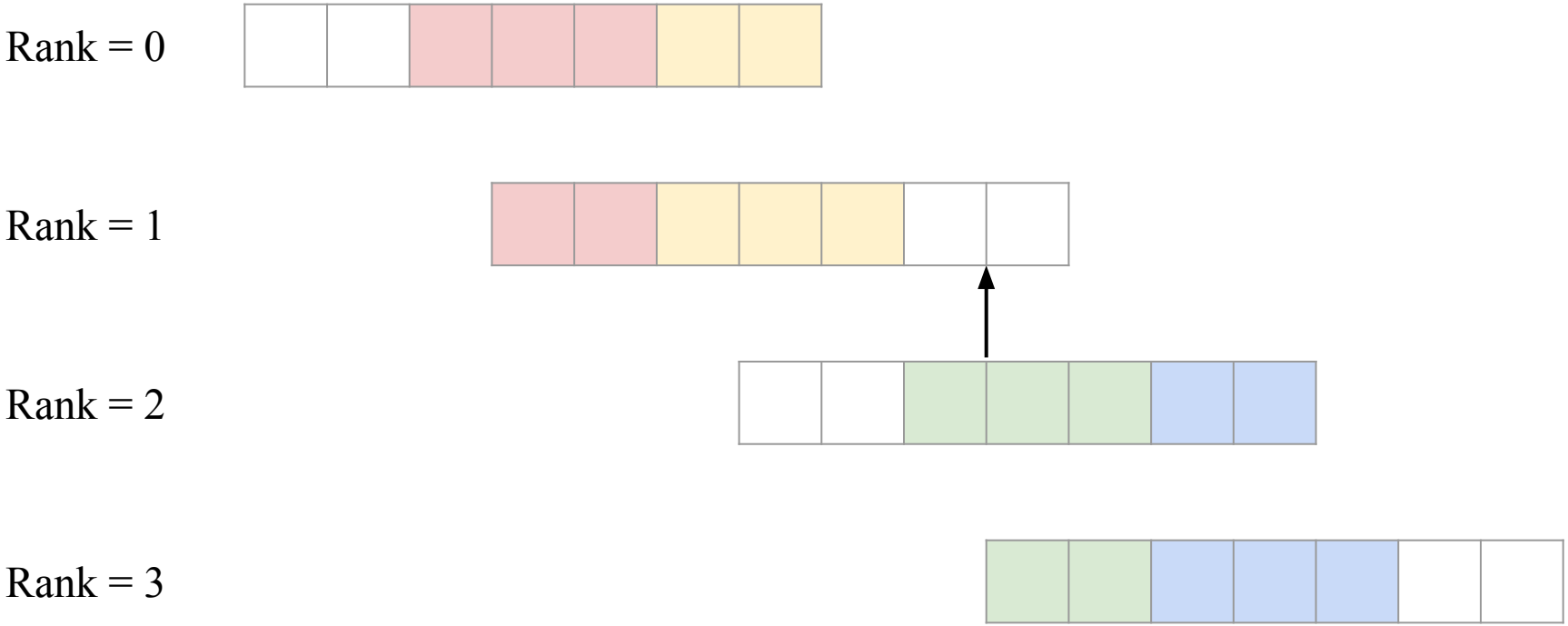
# Sending & Receiving data



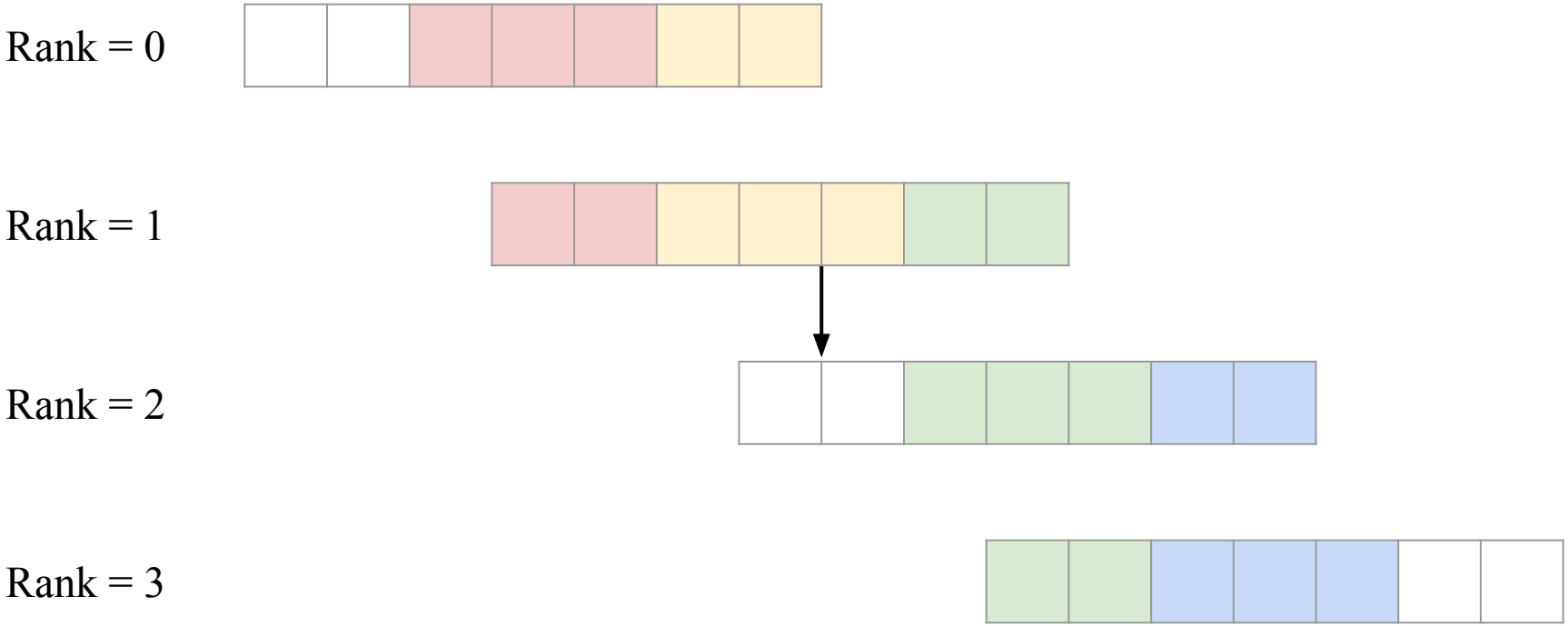
# Sending & Receiving data



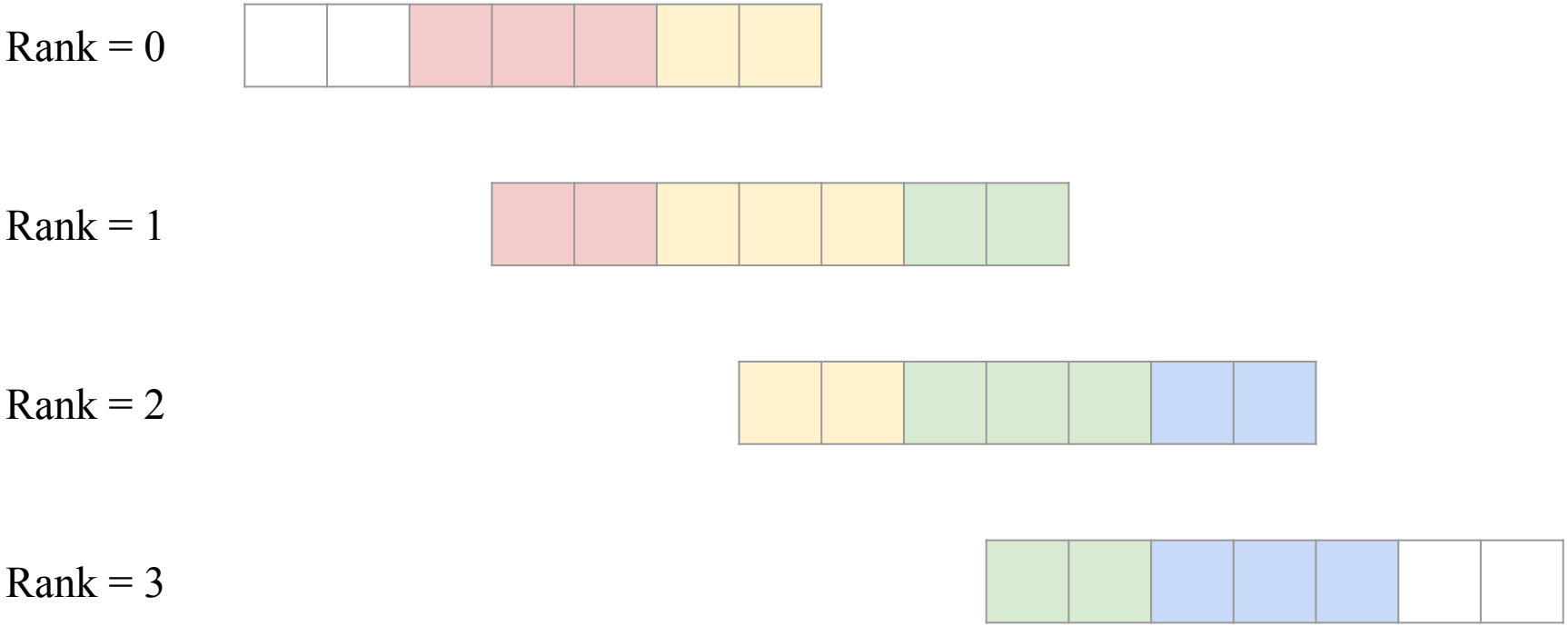
# Sending & Receiving data



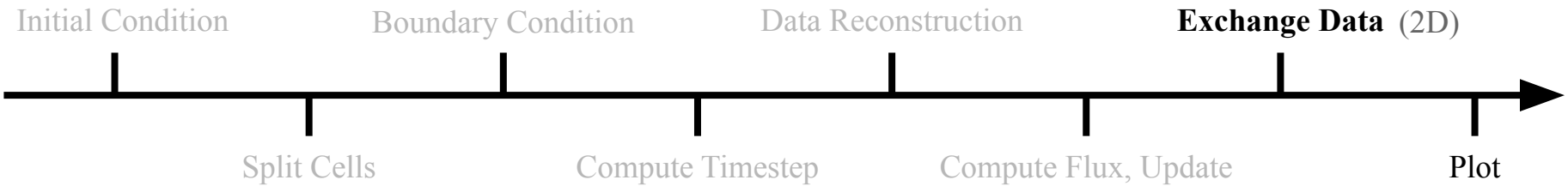
# Sending & Receiving data



# Done!



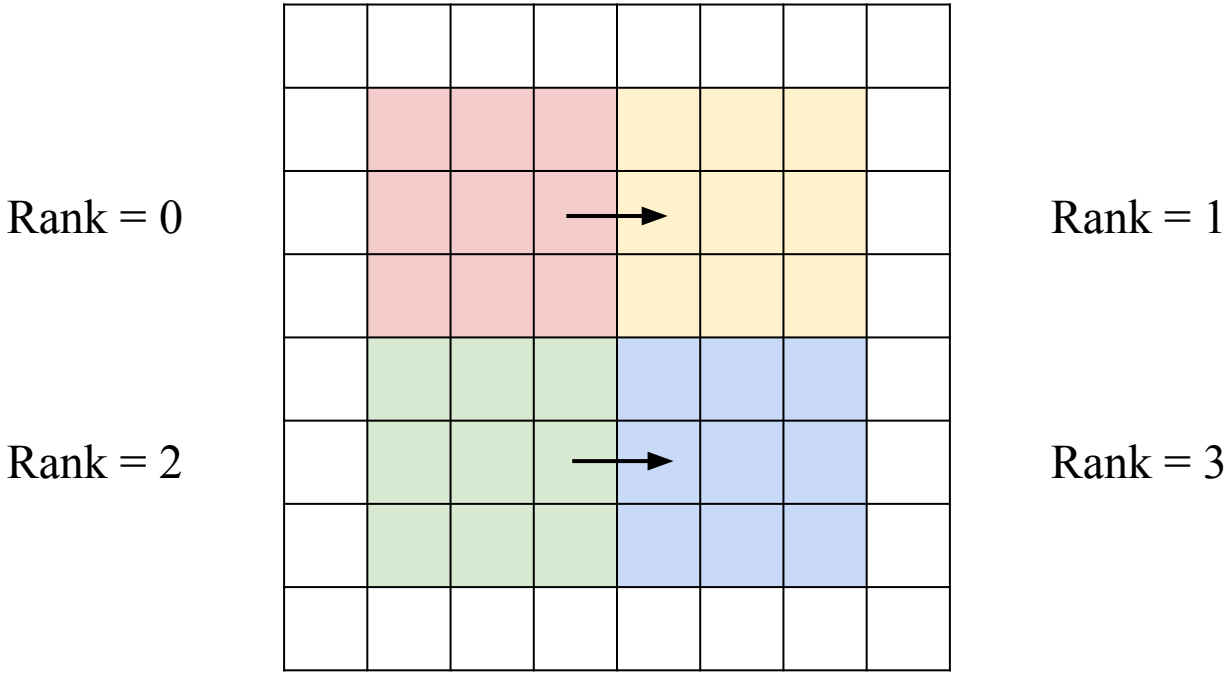
# Modifications



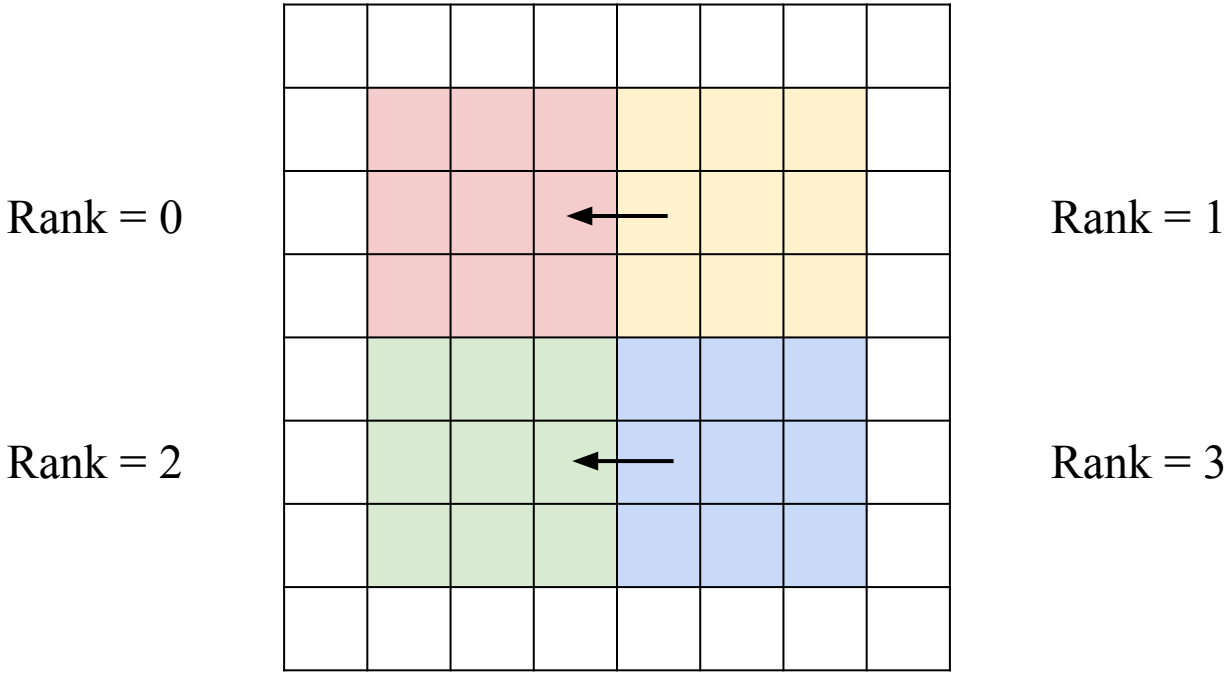

```
recv_u = np.empty((nghost*n,9))
recv_d = np.empty((nghost*n,9))
recv_r = np.empty((n*nghost,9))
recv_l = np.empty((n*nghost,9))
send_u = u[nghost:nghost+nghost].reshape(nghost*n,9)
send_d = u[n-nghost-nghost:n-nghost].reshape(nghost*n,9)
send_l = u[:, nghost:nghost+nghost].reshape(n*nghost,9)
send_r = u[:, n-nghost-nghost:n-nghost].reshape(n*nghost,9)
```

```
if rank%2 == 0:
    comm.Send(send_r, rank+1, tag = it+rank)
    comm.Recv(recv_r, rank+1, tag = it+rank+1)
    bd_r = recv_r.reshape(n,nghost,9)
    bd_l = u[:, :nghost]
else:
    comm.Recv(recv_l, rank-1, tag = it+rank-1)
    comm.Send(send_l, rank-1, tag = it+rank)
    bd_l = recv_l.reshape(n,nghost,9)
    bd_r = u[:, -nghost:]
if int(rank/2) == 0:
    comm.Send(send_d, rank+2, tag = it+rank+10)
    comm.Recv(recv_d, rank+2, tag = it+rank+12)
    bd_d = recv_d.reshape(nghost,n,9)
    bd_u = u[:nghost, :]
else:
    comm.Recv(recv_u, rank-2, tag = it+rank+8)
    comm.Send(send_u, rank-2, tag = it+rank+10)
    bd_u = recv_u.reshape(nghost,n,9)
    bd_d = u[-nghost:, :]
```

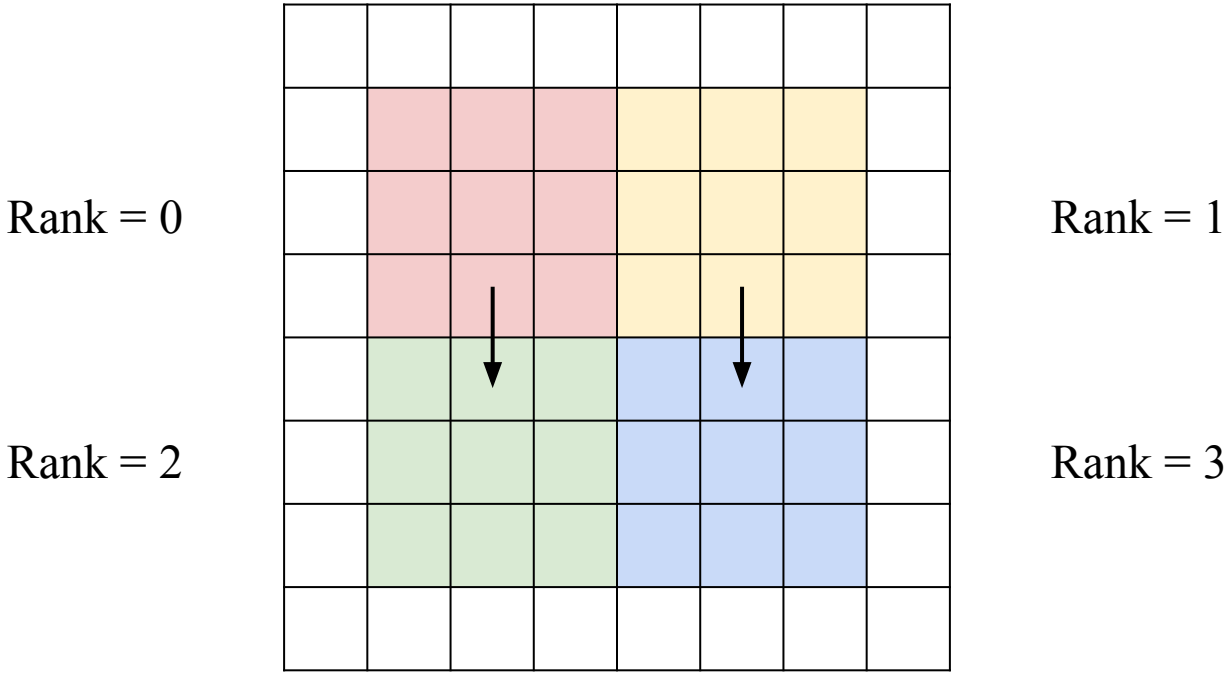
# Sending & Receiving data



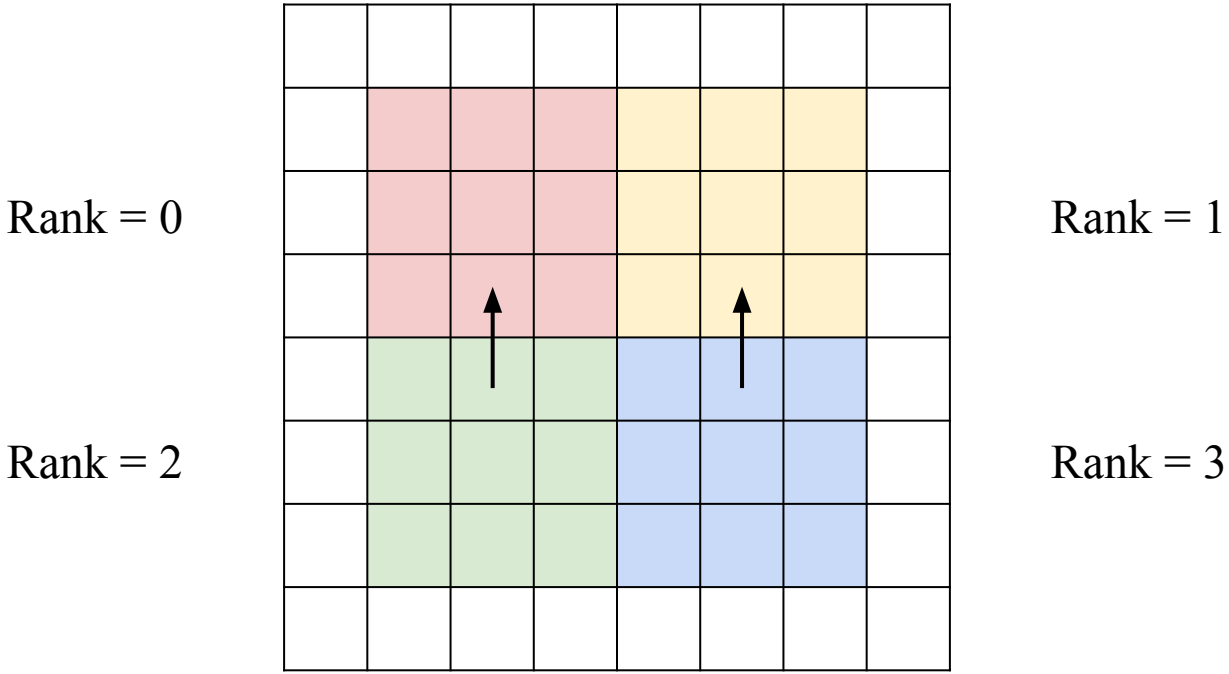
# Sending & Receiving data



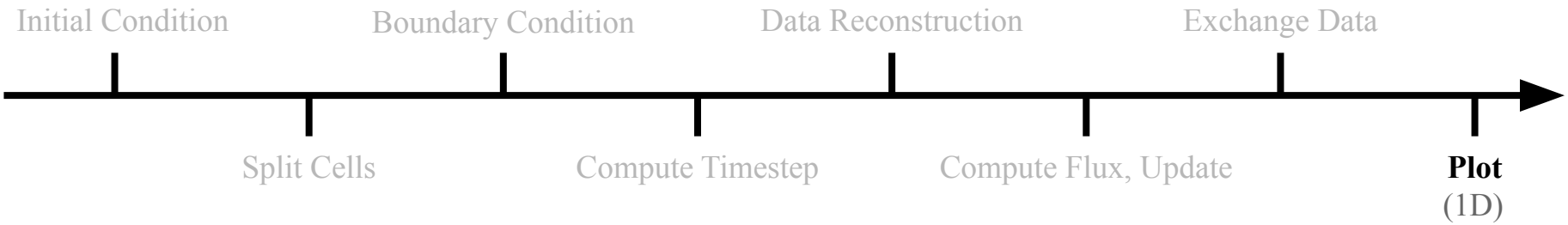
# Sending & Receiving data



# Sending & Receiving data

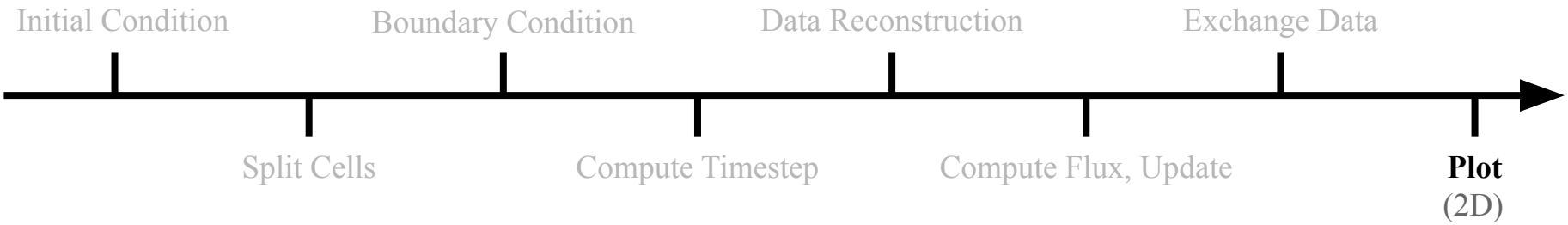


# Modifications



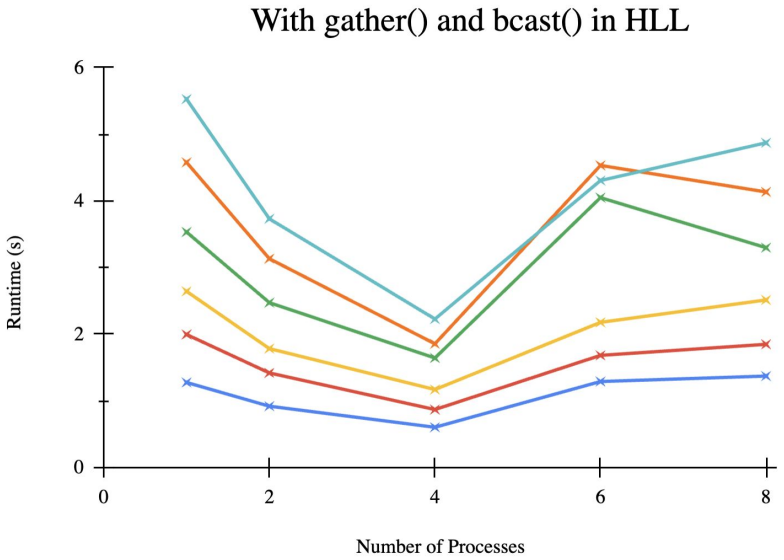

```
Send = list(u[nghost:n-nghost])
Recv = comm.gather(Send, root = 0)
Bcast = comm.bcast(Recv, root = 0)
Cells = np.array( np.concatenate( np.array(Bcast), axis = None ) ).reshape((N_In,9))
U[nghost:N-nghost] = Cells
```

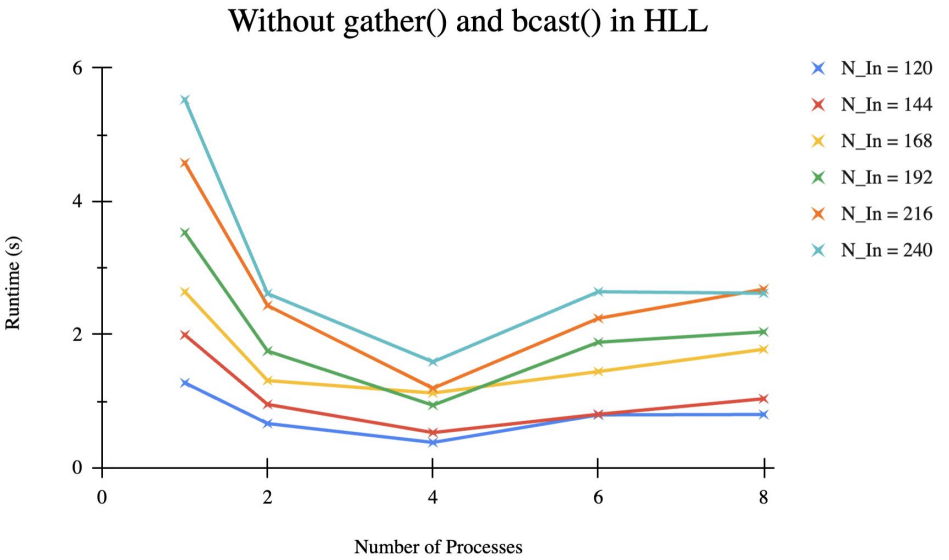
# Modifications



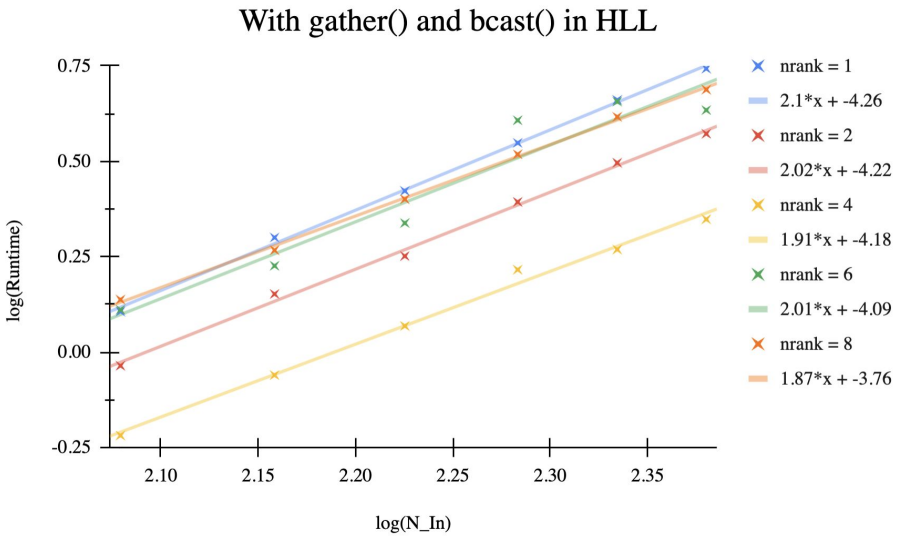

```
Send = list(u[nghost:n-nghost,nghost:n-nghost])
Bcast0 = comm.bcast(Send, root = 0)
Bcast1 = comm.bcast(Send, root = 1)
Bcast2 = comm.bcast(Send, root = 2)
Bcast3 = comm.bcast(Send, root = 3)
Cell0 = np.array( np.concatenate( np.array(Bcast0), axis = None ) ).reshape((n_In,n_In,9))
Cell1 = np.array( np.concatenate( np.array(Bcast1), axis = None ) ).reshape((n_In,n_In,9))
Cell2 = np.array( np.concatenate( np.array(Bcast2), axis = None ) ).reshape((n_In,n_In,9))
Cell3 = np.array( np.concatenate( np.array(Bcast3), axis = None ) ).reshape((n_In,n_In,9))
U[nghost:int(N/2),nghost:int(N/2)] = Cell0
U[nghost:int(N/2),int(N/2):N-nghost] = Cell1
U[int(N/2):N-nghost,nghost:int(N/2)] = Cell2
U[int(N/2):N-nghost,int(N/2):N-nghost] = Cell3
```

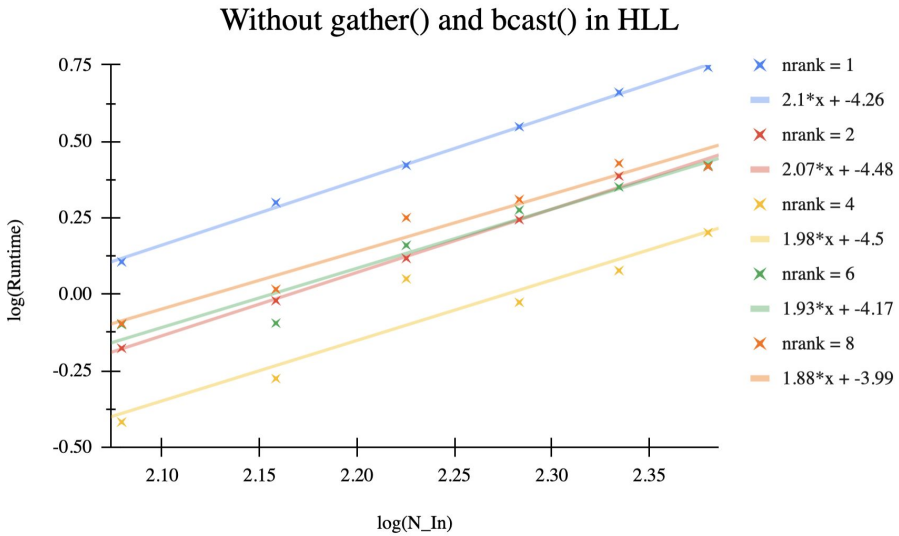
# Performance



# Performance



# Discussion

Most time consuming step: gather and bcast

Turns out that the gather and bcast was unnecessary inside **def HLL**, because the signs of SL and SR stayed the same

gather and bcast are still necessary in **ComputeTimestep**, and when nrank went from 4 to 6, the average time it takes to compute went up a whopping 30 times.

The scaling of runtime with N is as expected for other steps

Sending and receiving data took less time than expected

# (Possible) Far Future Works

- HLL in 2D - DS Balsara(2011) - WIP ('bout to RIP)
- Compare different correction methods.
- Compare with CT method.
- HLLC, HLLD, Roe…….
- GPU parallel
- Update our Github pages.
- Lots of thing to do, but not lots of time……
- Have some good sleeps finally…….

# Reference

[1] Brio, M. ; Wu, C. C. An Upwind Differencing Scheme for the Equations of Ideal Magnetohydrodynamics. Journal of Computational Physics, Volume 75, Issue 2, p. 400-422.

[2] Li, Shengtai. An HLLC Riemann solver for magneto-hydrodynamics. Journal of Computational Physics, Volume 203, Issue 1, p. 344-357.

[3] Einfeldt, B. ; Roe, P. L. ; Munz, C. D. ; Sjogreen, B. On Godunov-Type Methods near Low Densities. Journal of Computational Physics, Volume 92, Issue 2, p. 273-295.

[4] Eleuterio F. Toro. Riemann Solvers and Numerical Methods for Fluid Dynamics: A Practical Introduction.

[5] A. Dedner, F. Kemm, D. Kröner, C.-D. Munz, T. Schnitzer, and M. Wesenberg. Hyperbolic Divergence Cleaning for the MHD Equations. Journal of Computational Physics 175, 645–673.

[6] Patricia Cargo, Gérard Gallice. Roe Matrices for Ideal MHD and Systematic Construction of Roe Matrices for Systems of Conservation Laws. Journal of Computational Physics 136, 446–466.

[7] Dinshaw S. Balsara. A two-dimensional HLLC Riemann solver for conservation laws: Application to Euler and magnetohydrodynamic flows. Journal of Computational Physics 231,  7476-7503.

# Contribution Table

Divergence Cleaning: 賴伯熏

1D MHD Riemann Solver: 黃品睿

2D MHD: 吳冠霖

MPI Parallelization: 曾映舫

# Github Page

We didn’t really have time to sort out our Github pages. Most of the code is still in our personal branches, but they are public. You may take a look if wanted, and sorry for the inconvenience.

- pipimei: Hydro Riemann solver and MHD Riemann solver (single process)
- grid: MPI parallelization, supporting 1D hydro and MHD (both 1D and 2D). Cleaning included for 1D MHD.
- JW: 2D MHD and CT test code

# Project-V: Divergence Cleaning for MHD

1. Implement a divergence cleaning method for MHD
   a. Compare the divergence errors with and without this correction
   b. No need to adopt the constraint transport technique
   c. Apply to a MHD test problem (e.g., MHD Riemann problems)
   d. Measure the performance scaling (i.e., wall-clock time vs. number of cells)

2. Bonus points
   a. You get bonus points automatically since this topic is not covered by our lecture
   b. Incorporate with a high-resolution shock capturing scheme

3. Reference
   a. Hyperbolic Divergence Cleaning for the MHD Equations: https://www.sciencedirect.com/science/article/pii/S002199910196961X

1D —> Grade : F

# Thank you !

And wish you all a great summer.