# AMR

松本倫明（法政大学人間環境学部）

# 星形成のシナリオ

分子雲→分子雲コア→ファーストコア→原始星

極端に大きなダイナミックレンジ

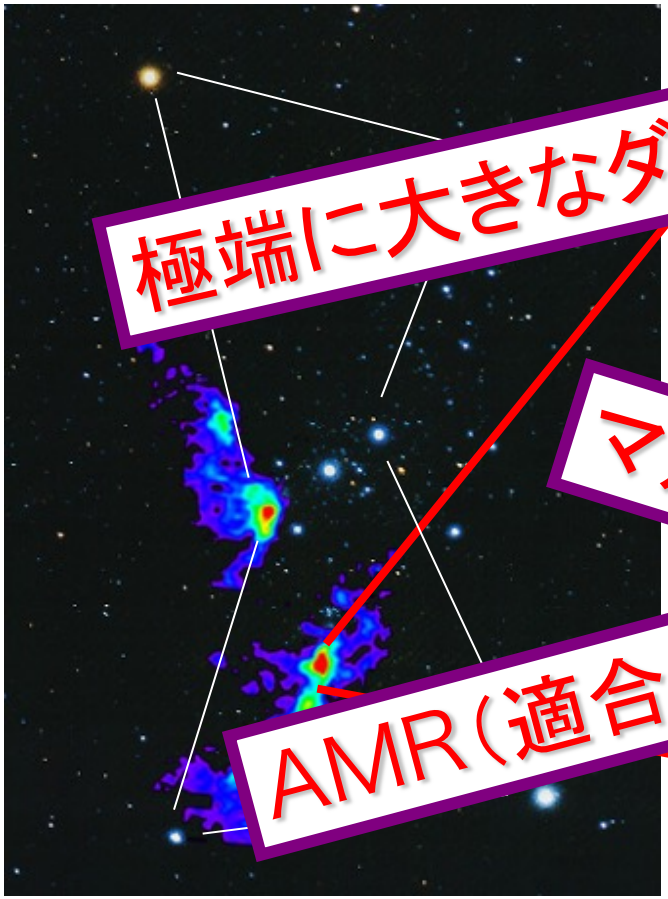
マルチスケール

AMR(適合格子細分化法)

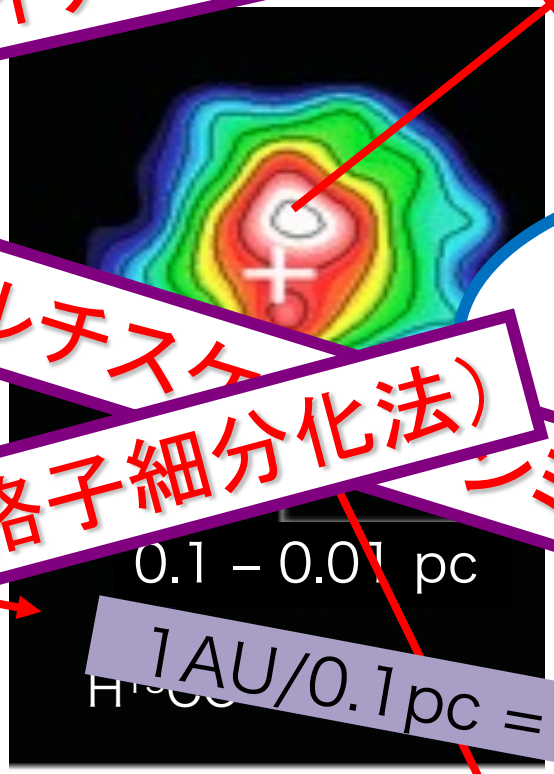
シミュレーション

−5

1AU/0.1pc = 5×10-5



オリオン分子雲
（optical＋radio）
Sakamoto et al. (1994)




Mizuno et al. (1994)

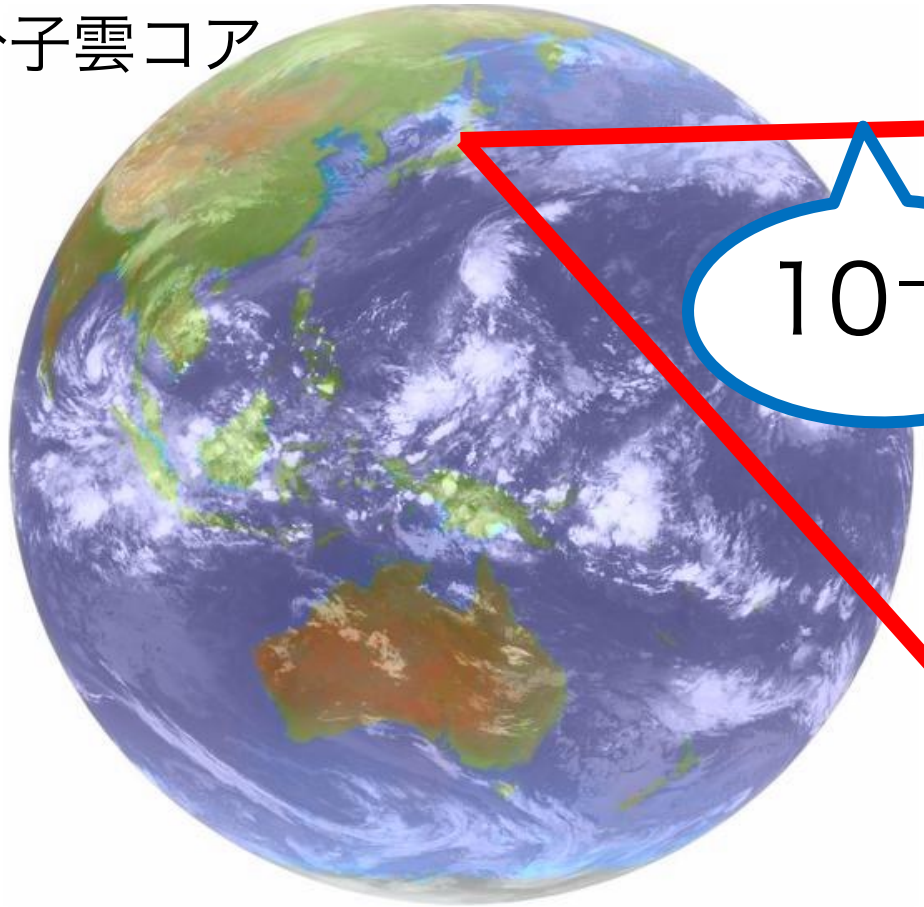
(radio)

Gueth & Guilloteau (1999)

# 分子雲コア→ファーストコア→原始星

再掲

分子雲コア

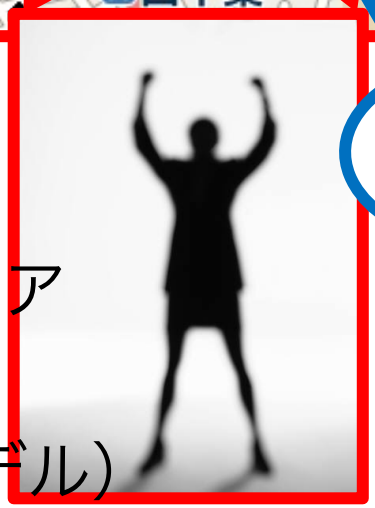


ファーストコア


$10^{-5}$

$10^{-2}$

セカンドコア
（原始星）



シンク粒子（サブグリッドモデル）

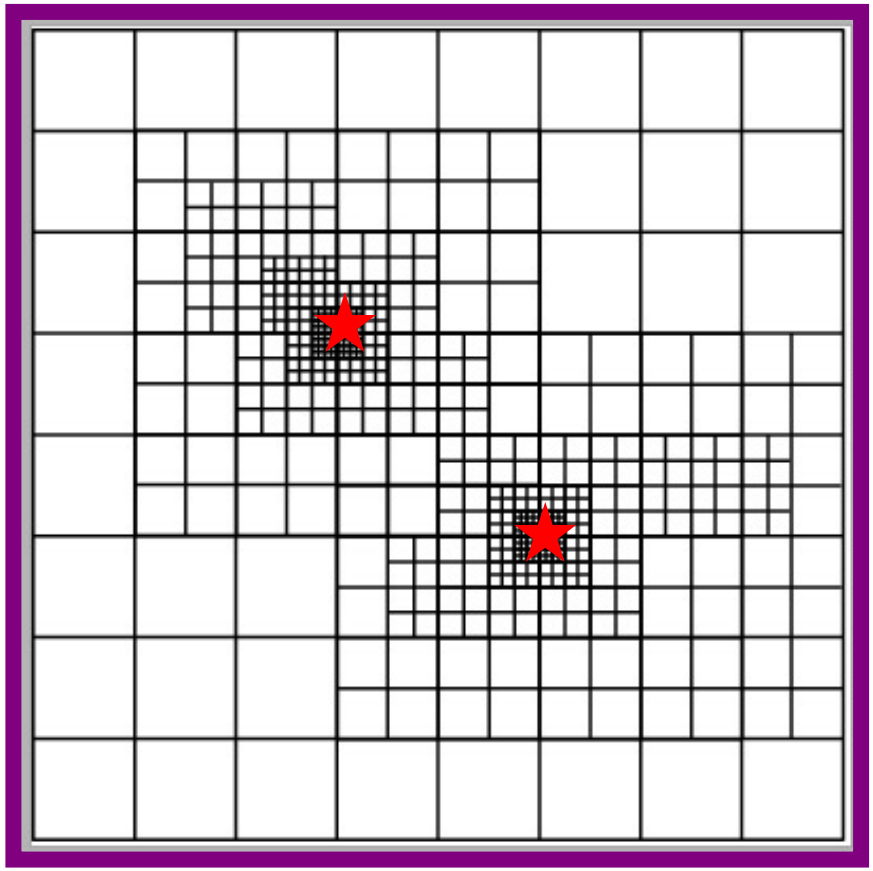
# AMRとは

- 適合格子細分化法 (Adaptive Mesh Refinement; AMR)
- 解適合格子(Solution Adaptive Mesh)

- 骨子
  - 高解像度が必要な領域を高解像度に。
  - それ以外を低解像度に。
  - 「高解像度が必要な領域」を動的に変更する。
  - トータルで格子点数を節約する。
  - 計算機性能の割に、高解像なシミュレーションが可能。
    例：256^3 → 10 level, (256*1024)^3=(1.80144e+16)^3

# AMRの模式図（例）

高解像度が必要な領域を高解像度に



連星系の形成

星の運動とともに
格子を動的に更新する。

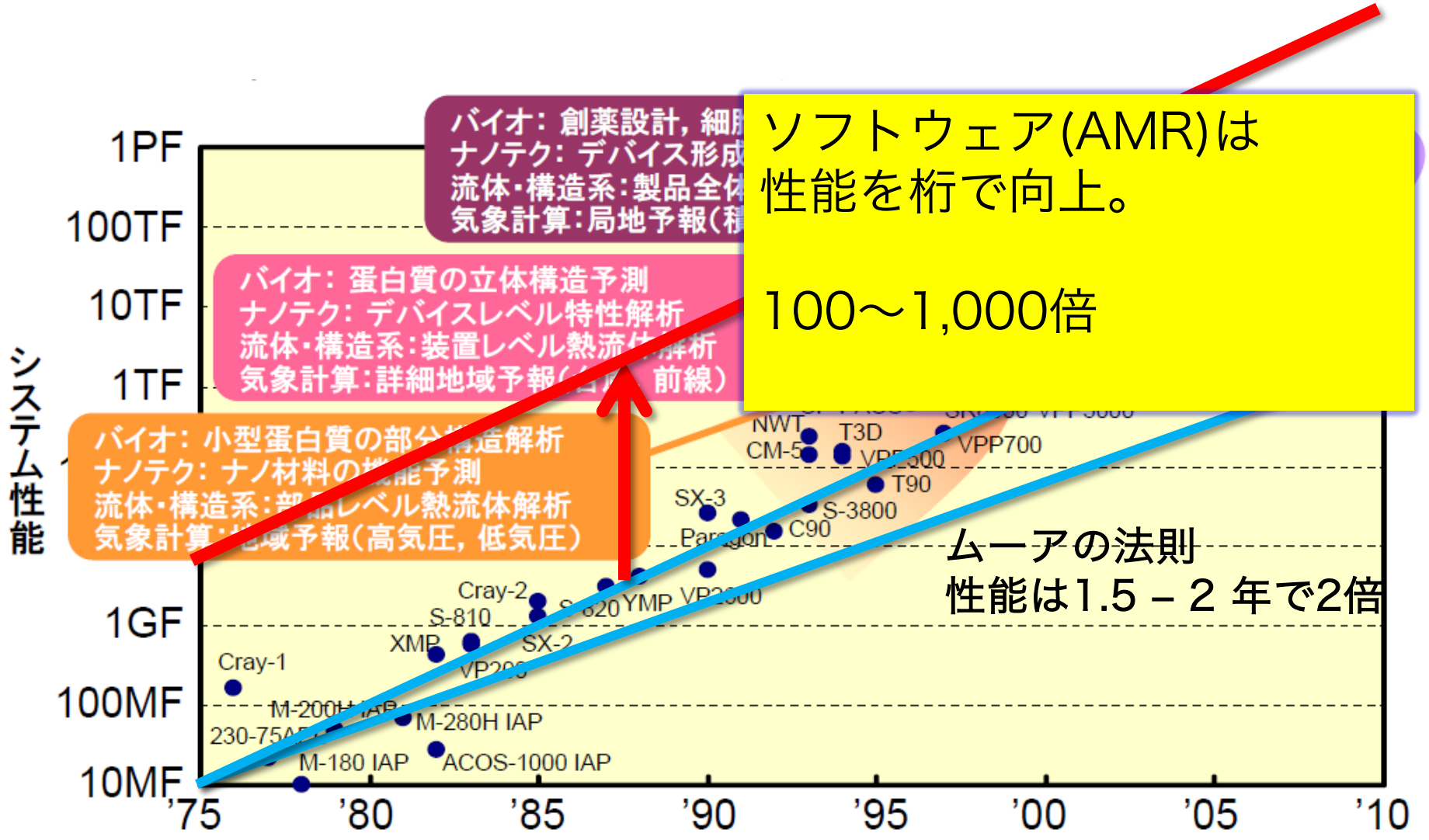
# AMRを使う目的

- 自己重力による構造形成：
  - 形成される天体を分解したい。
  - 空間スケール∝密度$^{-1/2}$
- 惑星間空間のプラズマ：
  - カレントシートを分解したい。
- 超新星爆発：
  - 燃焼波（デトネーション・デフラグレーション）を分解したい。
- 乱流：
  - パワースペクトルを長いレンジで書きたい。
- つまり、ダイナミックレンジが欲しい。
- 格子点数の割りに高解像度が欲しい。

# 自己重力系のシミュレーションにはAMR

- 自己重力系：　大きさはジーンズ長
  - 収縮すると小さくなる
- 一様格子では破綻
- AMR、SPH、非一様格子を導入
- SPHを勧めない理由
  - 質量～ジーンズ質量（どんどん小さくなる）
  - いずれSPHが破綻
- 非一様格子を勧めない理由
  - 形成する天体の位置は決まっていない。
    （決まっていれば非一様格子でもOK）

# なぜAMRか。 ハードウェア vs ソフトウェア




スーパーコンピュータの性能の推移　日立製作所2004

古い資料でごめんなさい

# 分類

JOURNAL OF COMPUTATIONAL PHYSICS 82, 64–84 (1989)

# Local Adaptive Mesh Refinement for Shock Hydrodynamics

M. J. BERGER

*Courant Institute of Mathematical Sciences, New York University, 251 Mercer Street, New York, 10012 New York*

AND

P. COLELLA

*Lawrence Livermore Laboratory, Livermore, 94550 California*

Received September 8, 1987; revised May 20, 1988

The aim of this work is the development of an automatic, adaptive mesh refinement strategy for solving hyperbolic conservation laws in two dimensions. There are two main difficulties in doing this: the first is the [illegible] in the solution and the effect on them of discontinuities in the mesh; the second problem has to do with organizing the algorithm to minimize memory and CPU overhead. This is an important consideration and will continue to be important as more sophisticated algorithms that use data structures other than arrays are developed for use on vector and parallel computers. 

全てのAMRはこの論文起源

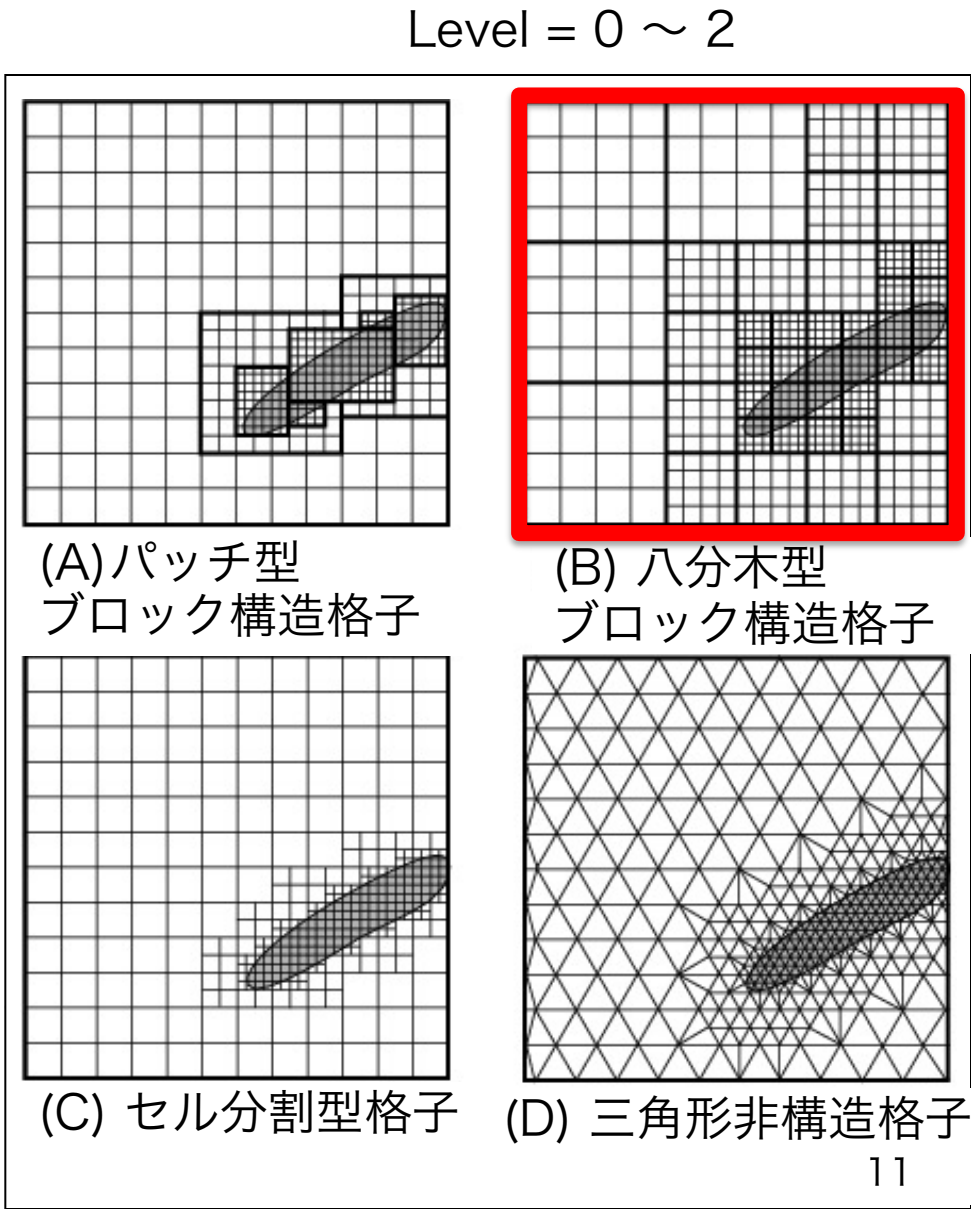
# AMRの分類

A) **パッチ型ブロック構造格子**
- パッチ指向
- 最初のAMR
- Berger & Oliger 1984,
- Berger & Colella 1989

B) **八分木型ブロック構造格子**
- 八分木構造

C) **セル分割型格子**
- 八分木構造

D) **三角形非構造格子**
- 天文学ではあまり用いられていない
- 機体に沿った境界条件に有利。



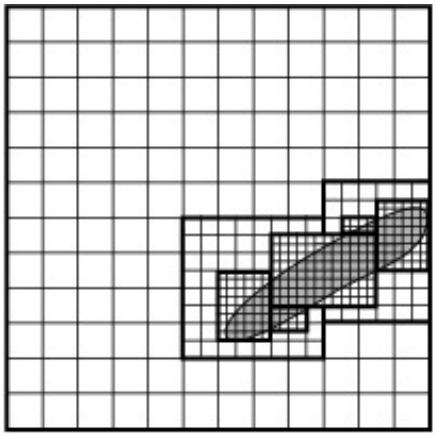
# AMRの分類

A) **パッチ型ブロック構造格子**
- セル数は少なめ。
- ブロック配置のアルゴリズムが複雑。
- 袖のセルが少なく、効率が良い。
- メモリを動的に使う。

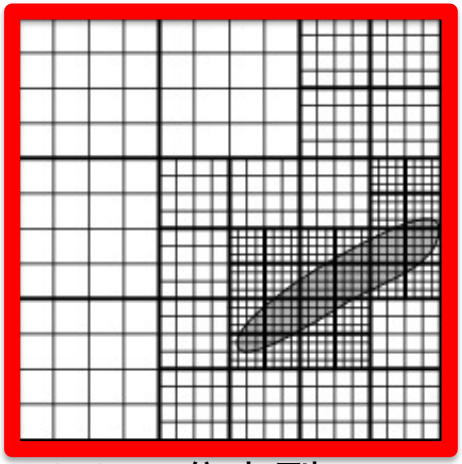
B) **八分木型ブロック構造格子**
- セル数多め。
- ブロック配置のアルゴリズムが単純。
- 袖のセルが多くい。
- メモリを静的に使う。
- キャッシュの有効活用。

C) **セル分割型格子**
- セル数は必要最低限。
- オーバーヘッドが大きいか？
- 一様格子ソルバの流用不可。

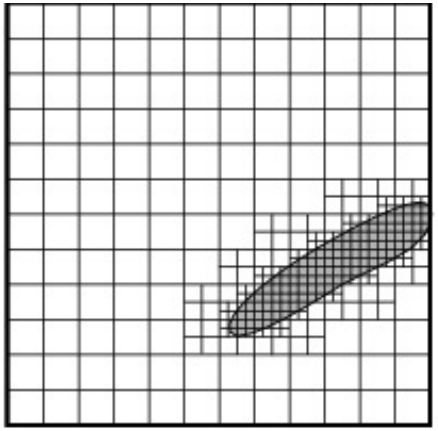
Level = 0 ～ 2



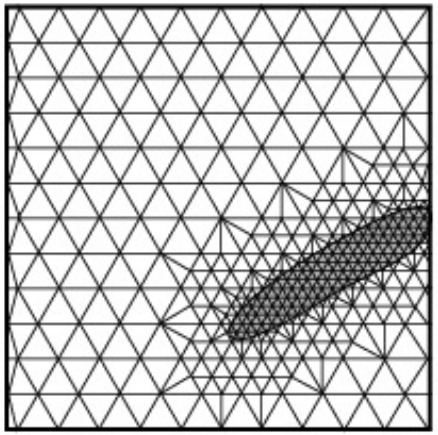
(A)パッチ型
ブロック構造格子



(B) 八分木型
ブロック構造格子



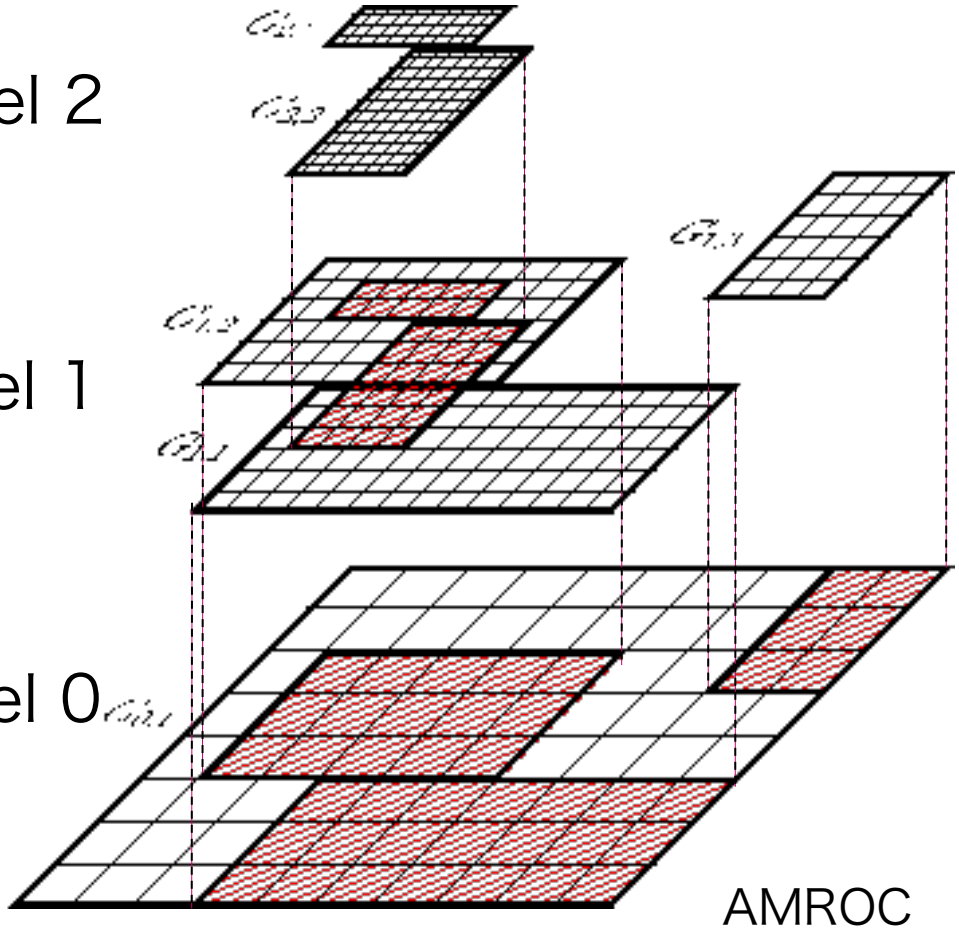
(C) セル分割型格子



(D) 三角形非構造格子

# パッチ型ブロック構造格子

Type A

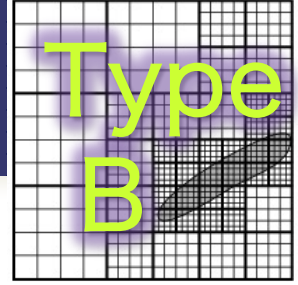


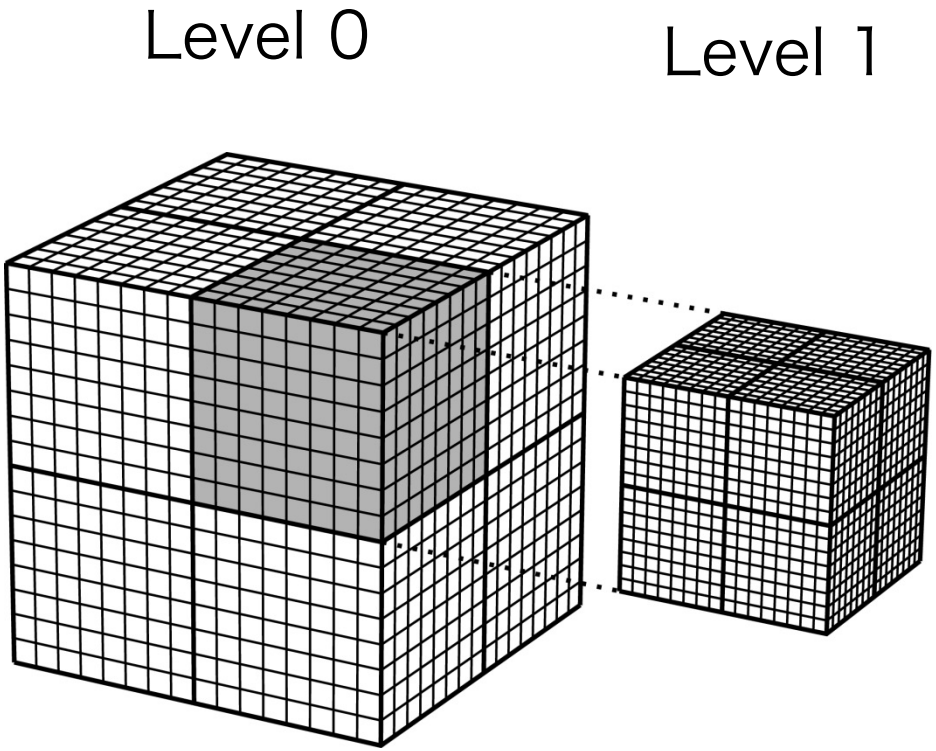

ブロックの大きさは様々（可変長）。

同じレベルのブロックが重なってもOK。

細分化比
= 2, 4, 8…

# 八分木型ブロック構造格子




SFUMATO

Matsumoto 2007
doi:10.1093/pasj/59.5.905

ブロックの大きさは固定（固定長）。

八分木によるデータ管理。

同じレベルのブロックは重ならない。

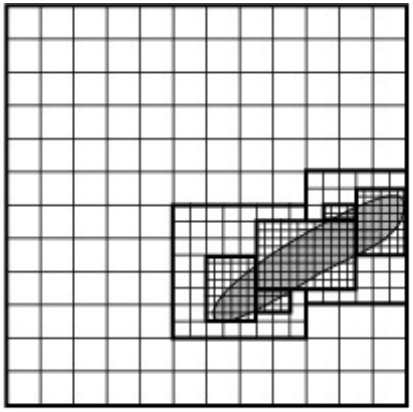
細分比 = 2

# 天体物理学における主なAMR（古いかも？）

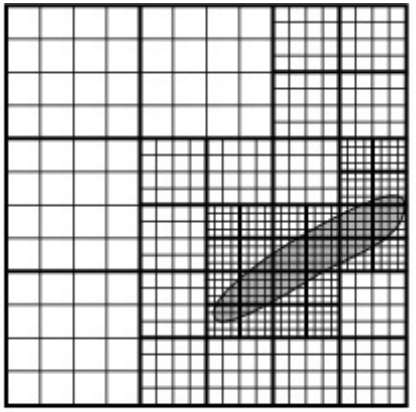
ほかにも沢山。国内にも下記以外に２コード存在。

現在では、MHD と 自己重力は多くのAMRに実装されている。

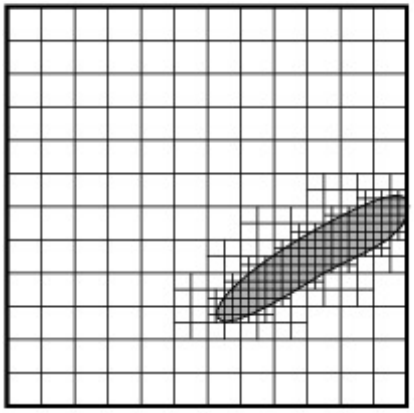
| Code name | Author(s) | Main targets | Grid type |
|---|---|---|---|
| ORION | R. Klein | Star formation | A |
| Enzo | M. Norman | Cosmology | A |
| FLASH | ASC/U-Chicago | Any | B |
| BATS-R-US | K. G. Powell | Space weather | B |
| NIRVANA | U. Ziegler | Any | B |
| RIEMANN | D. Balsara | ISM | A |
| RAMSES | R. Teyssier | Cosmology | C |
| CASTRO | A. S. Almgren CCSE.LBL | Supernovae | A |
| PLUTO | A. Mignone | Any | A |
| Athena++ | J. Stone | Any | B |
| SFUMATO | T. Matsumoto | Star formation | B |



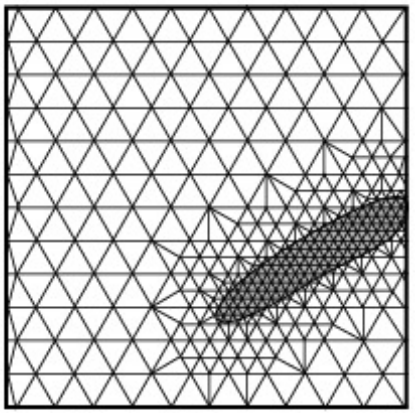
(A) パッチ型
ブロック構造格子



(B) 八分木型
ブロック構造格子

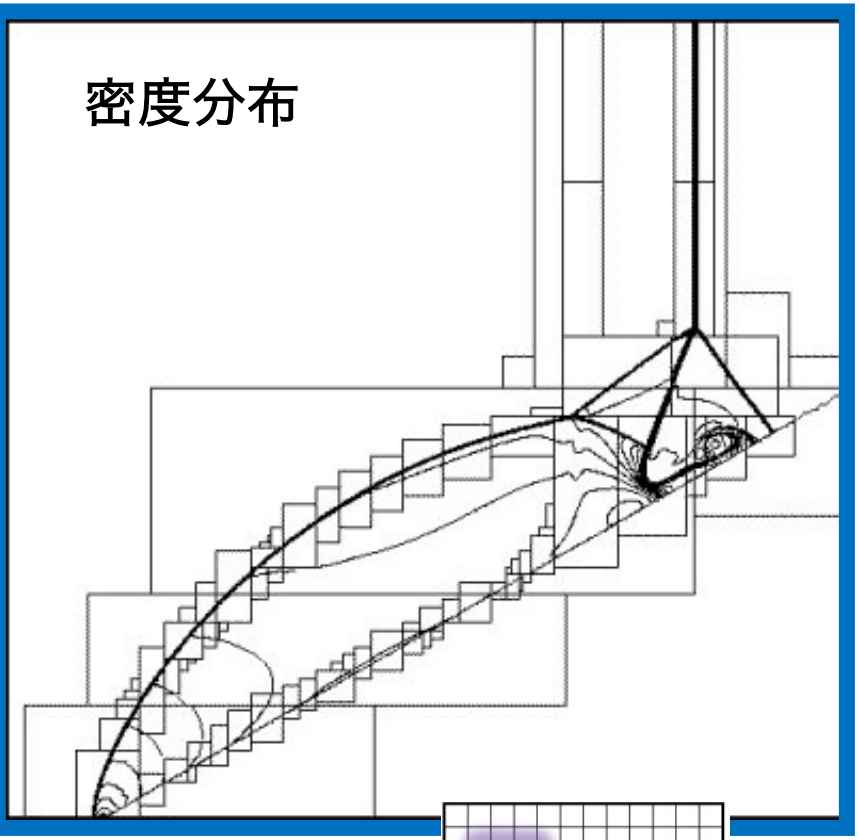


(C) セル分割型格子

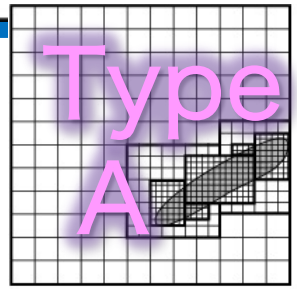


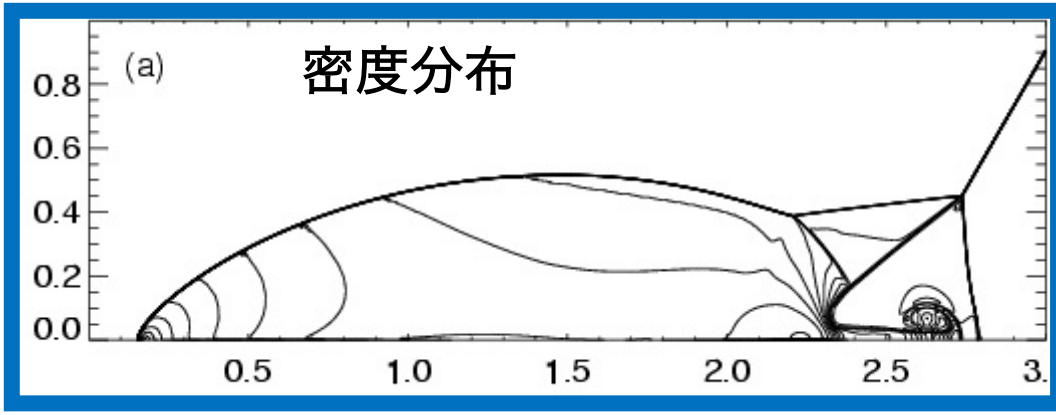
(D) 三角形非構造格子

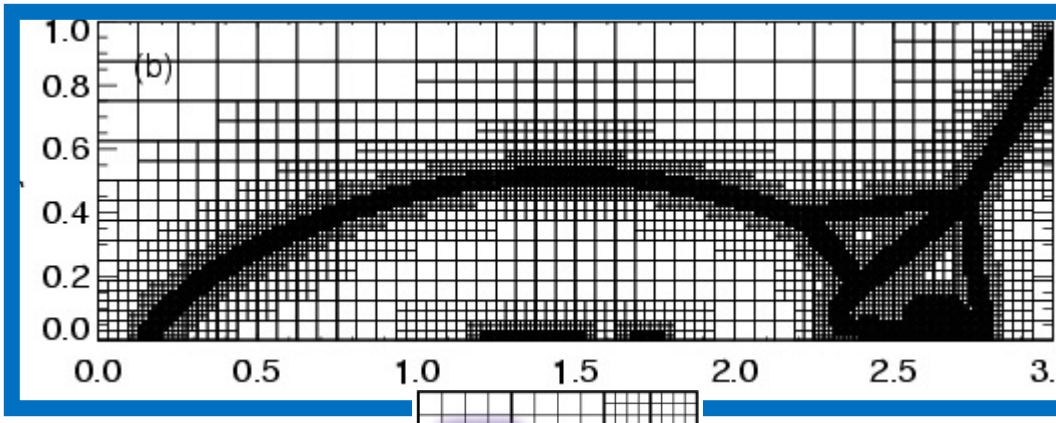
# 二重マッハ反射問題みるタイプ別の解

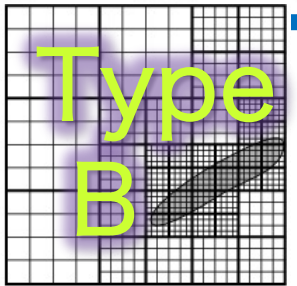



ORION

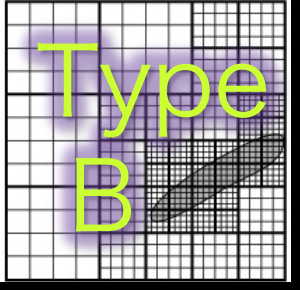


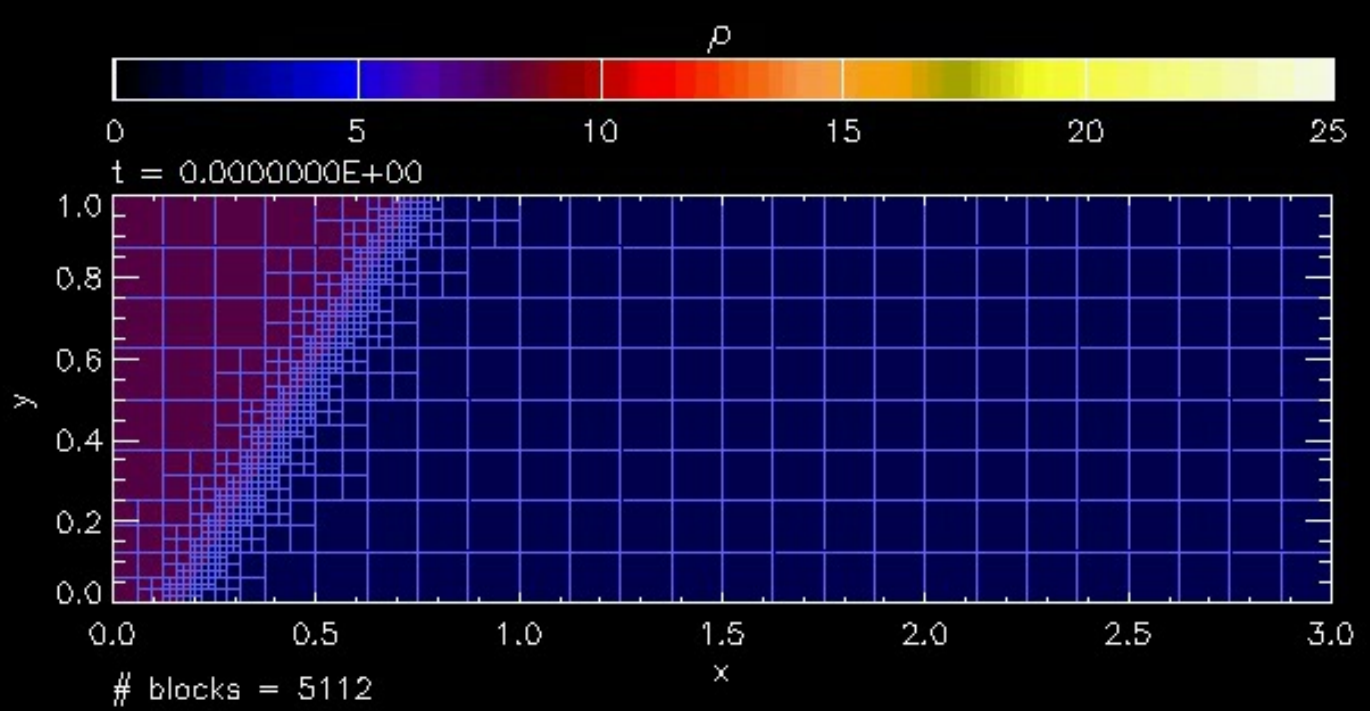



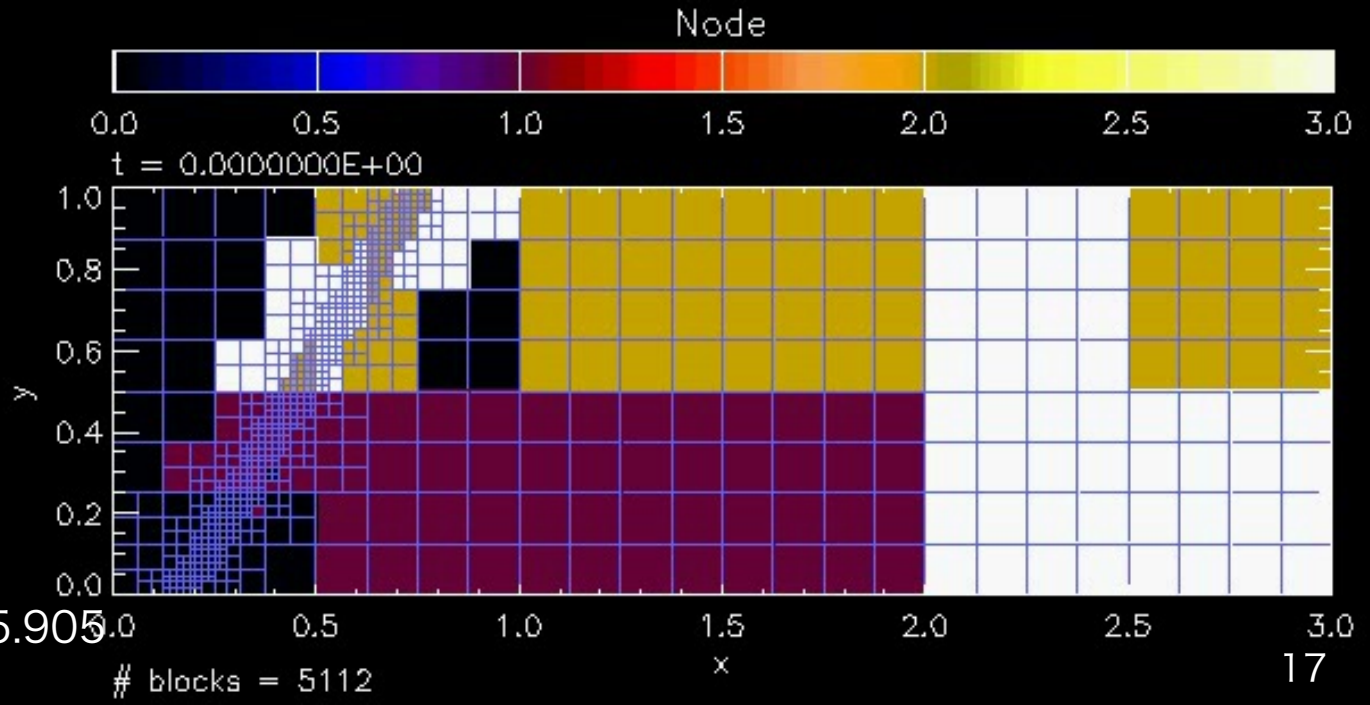

SFUMATO




密度分布




ノード（コア）
0, 1, 2, 3

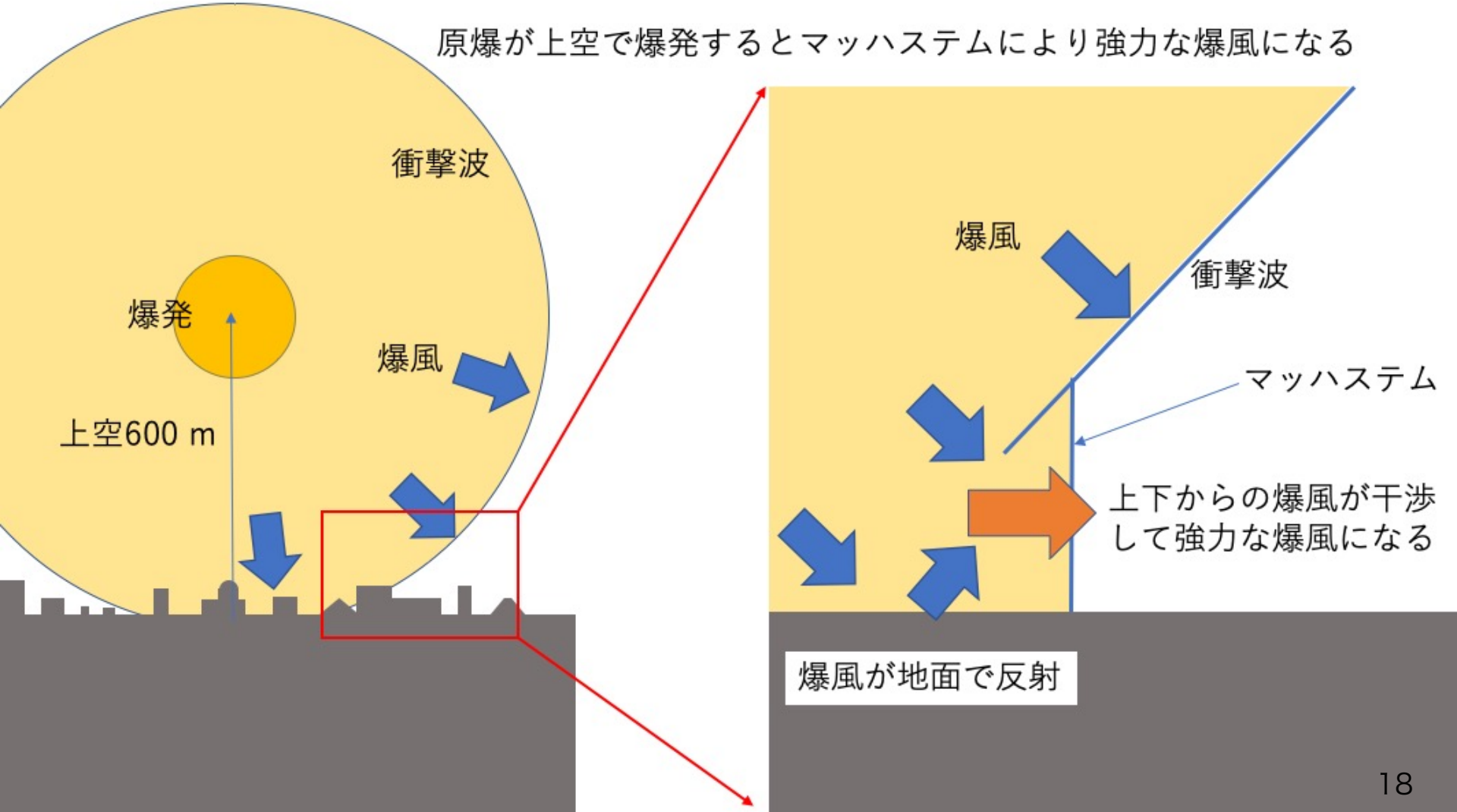



SFUMATO
Matsumoto 2007
doi:10.1093/pasj/59.5.905

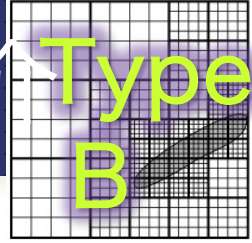
# 余談：二重マッハ反射問題とは

標準的な衝撃波のテスト問題だが、原爆に応用された。

NHKスペシャル 知られざる衝撃波～長崎原爆・マッハステムの脅威～

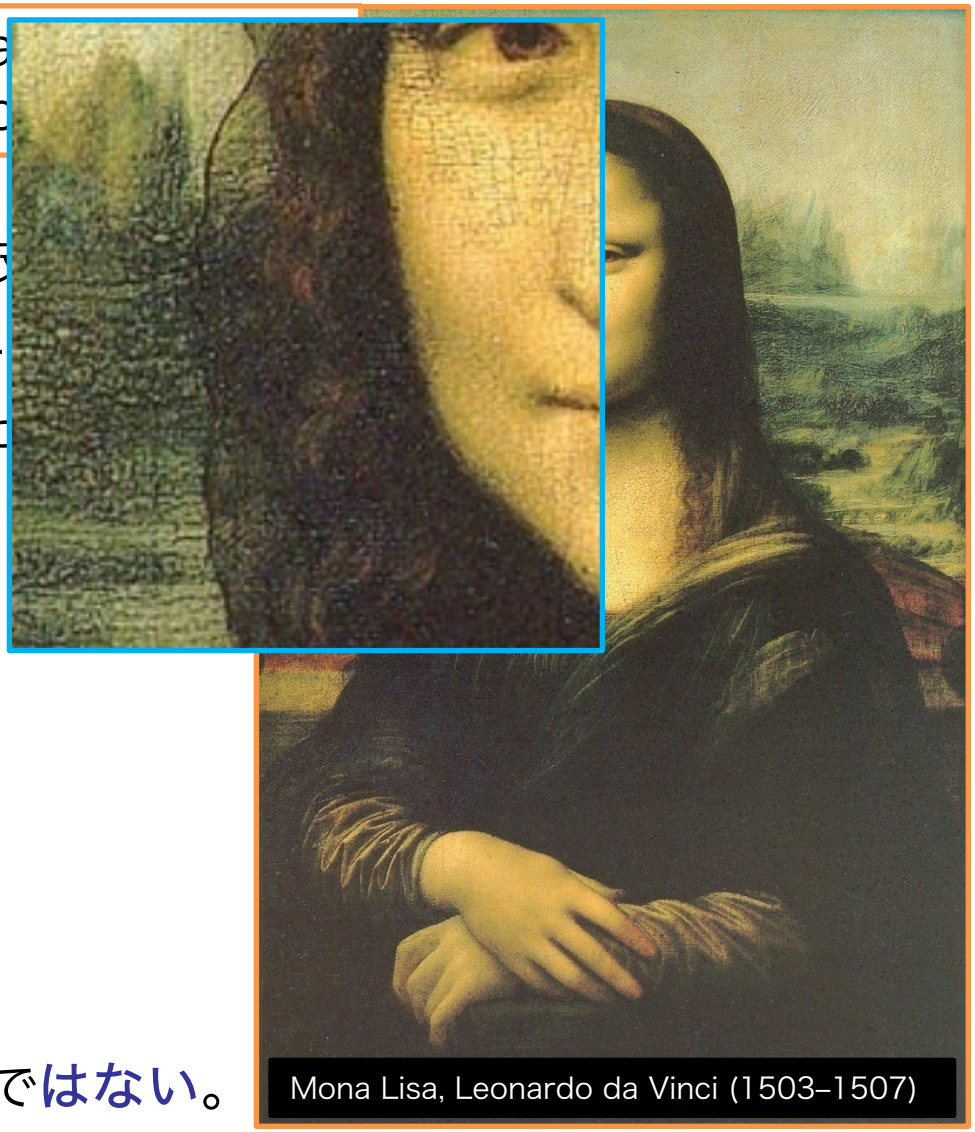


# 我々のAMRコード SFUMATOの紹介

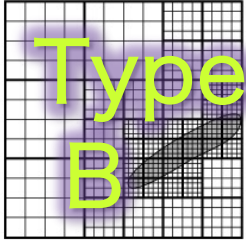


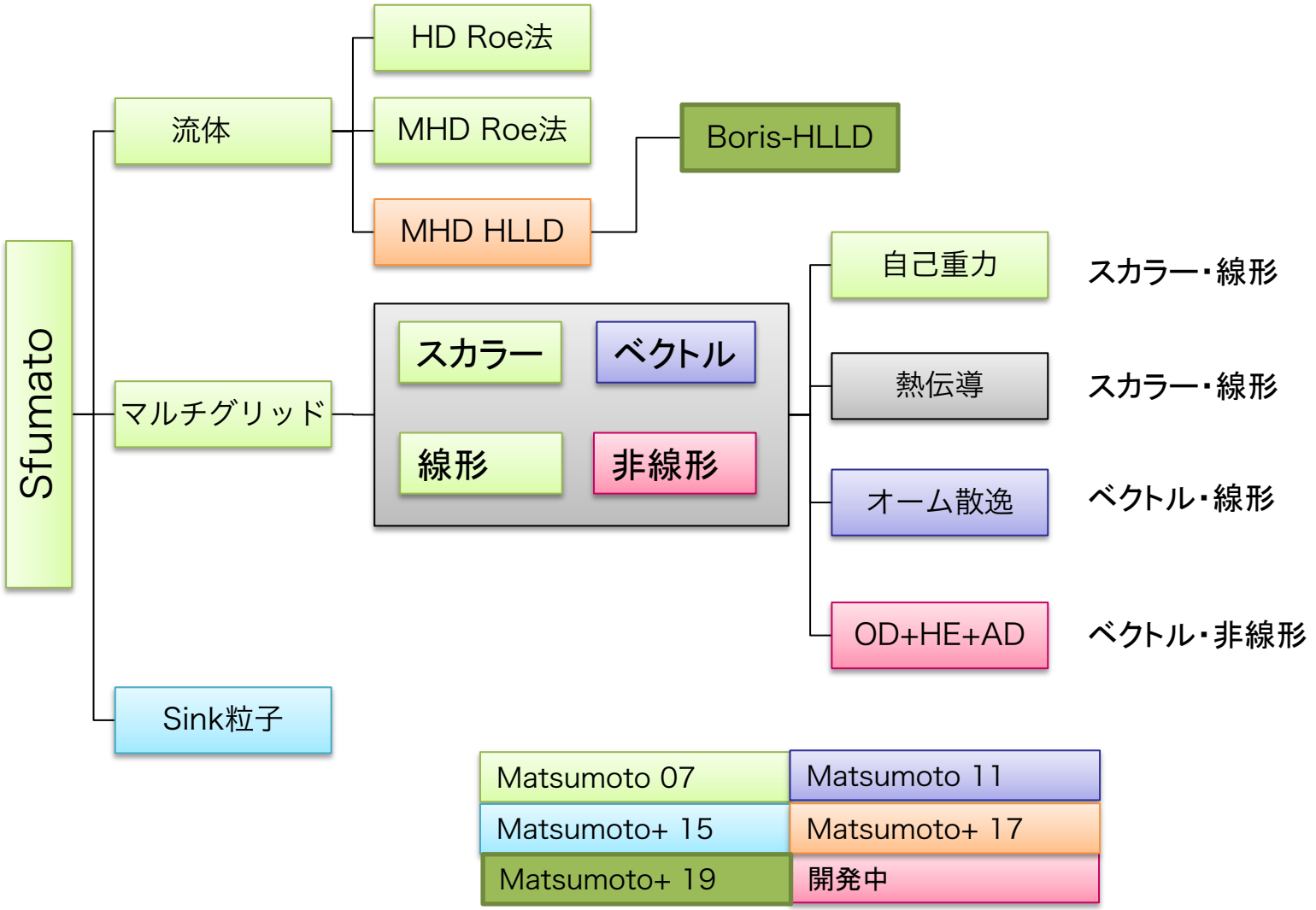

Self-gravitational Fluid-dyna
Utilizing Mesh Adaptive Tec
Oct-tree.

- *Sfumato* は本来、絵画の技
  オナルド・ダ・ビンチ (14
  1519) によって完成された
- その後、ルネサンスーバロ
  多くの画家に用いられた。
- モチーフの輪郭をぼかし、
  **空気を表現**。
- 我々のAMRコードも
  **ガス(空気)を表現**。
- Matsumotoのアナグラムで**はない**。


Mona Lisa, Leonardo da Vinci (1503–1507)

# AMRコード SFUMATO の構成



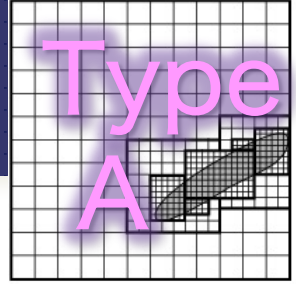



# 天文学における例

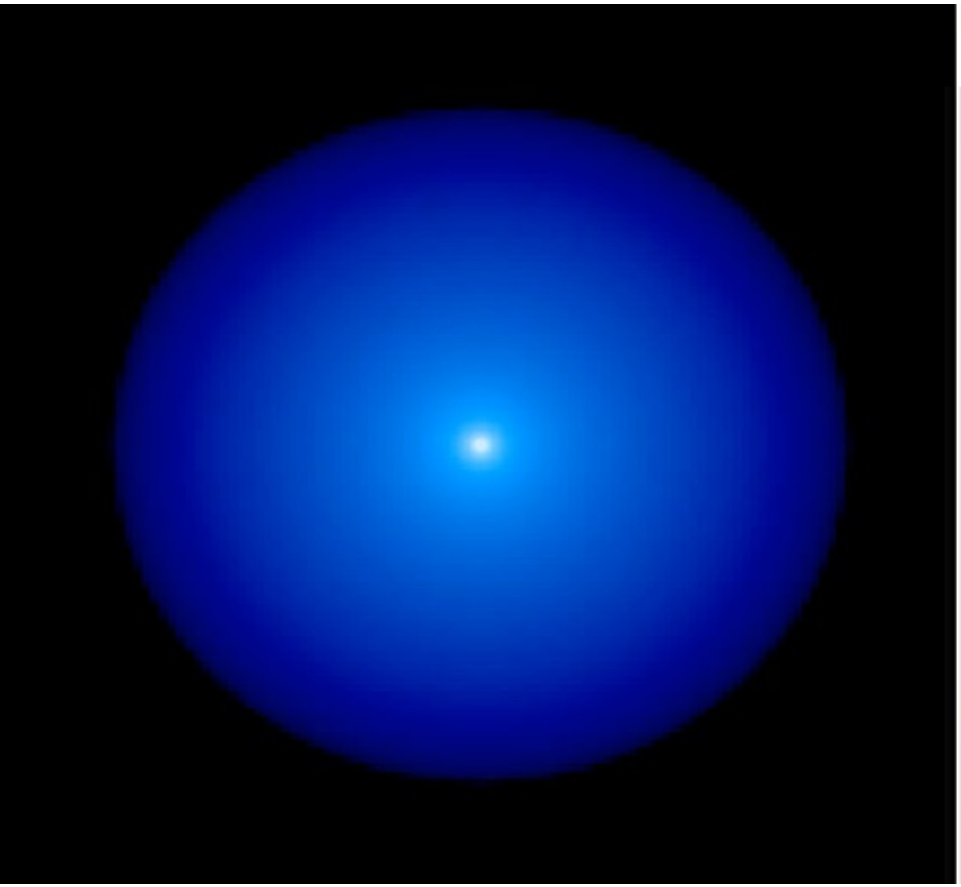
# AMRが当たり前の時代になった。

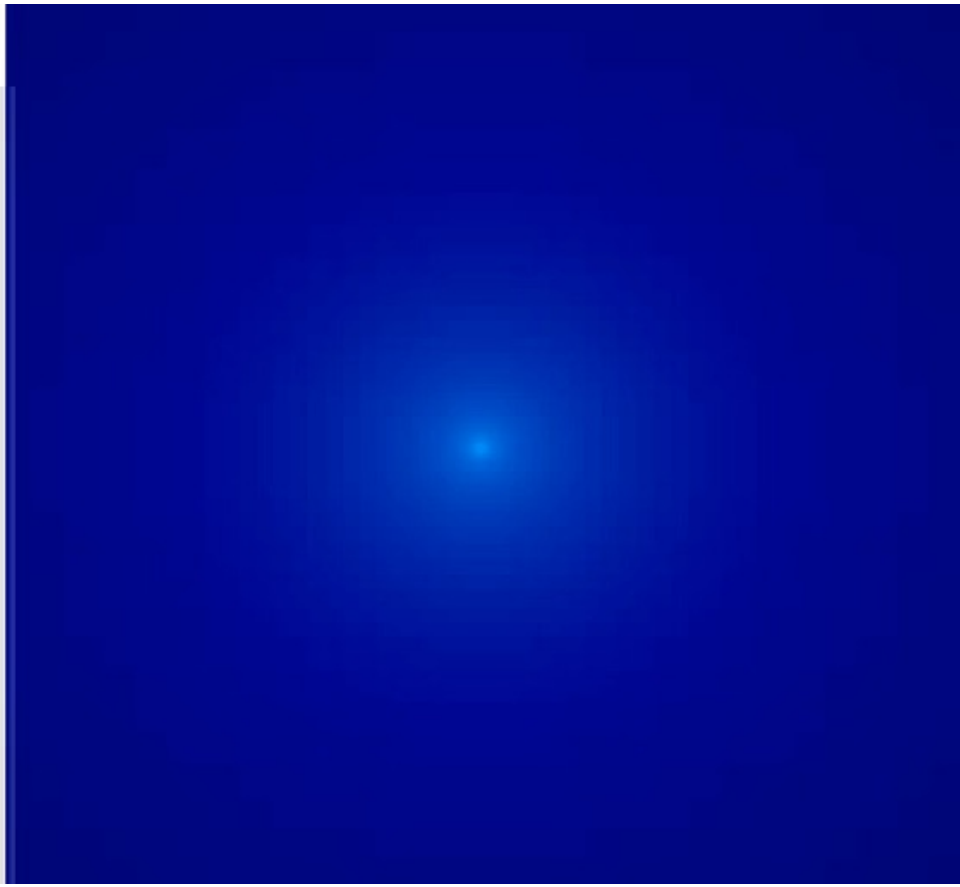
- AMRは特別ではなくなった。
  - 自己重力系のメッシュ法では必須
- 独自性の担保：
  - 新しい物理を導入して勝負
  - 新しいアイディアのモデルで勝負
- AMRの計算例を紹介する。

# 乱流コアでの多重星形成
# ORION: 流体+自己重力+輻射

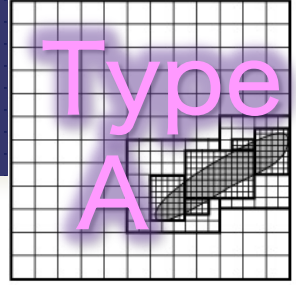



Krumholz, Klein, & McKee (2007

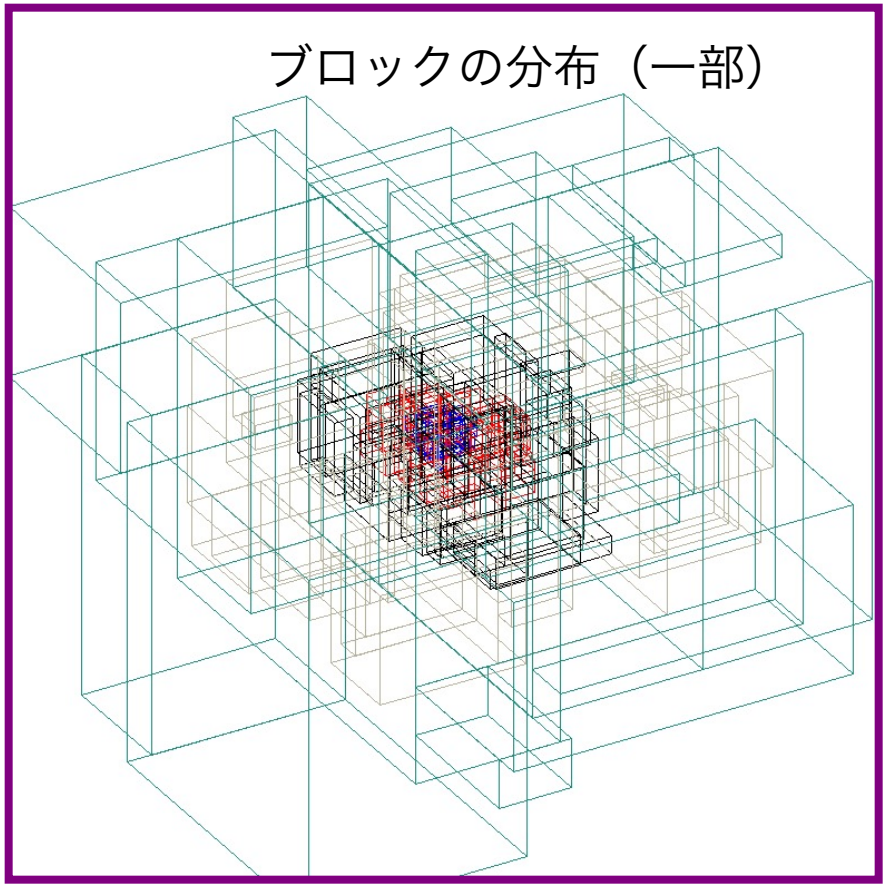


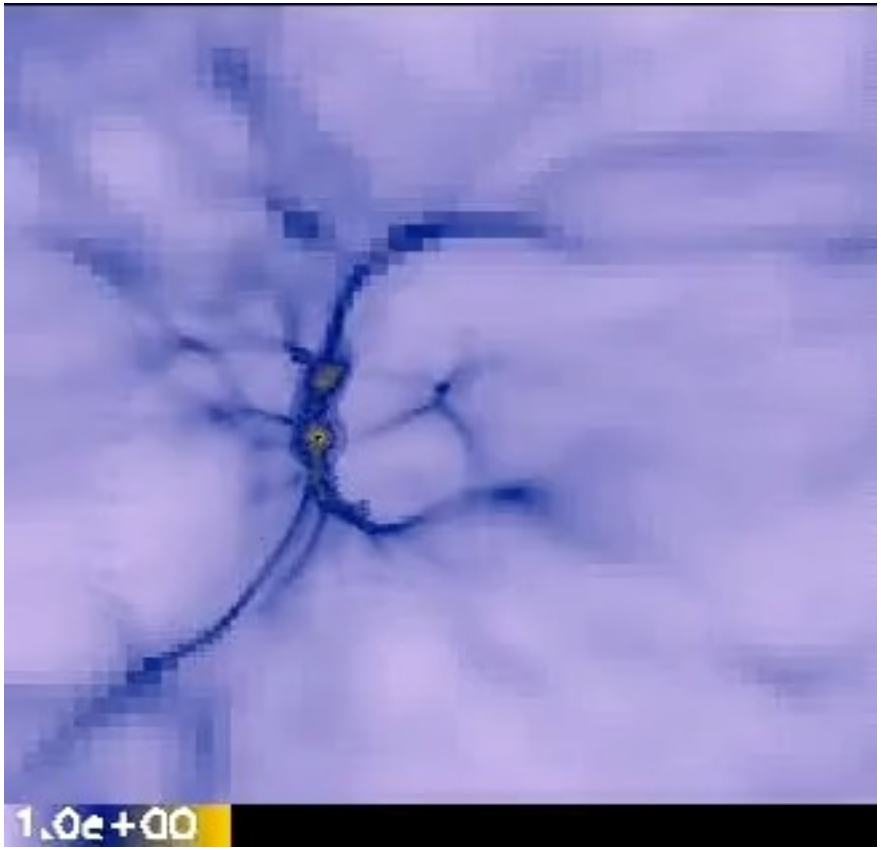


# First star formation by Enzo
## Abel et al. (2003)



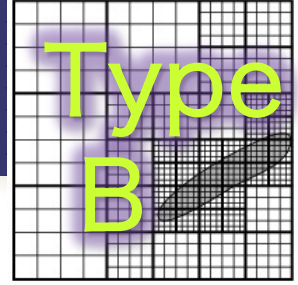

Block-structured grid (patch-oriented type)



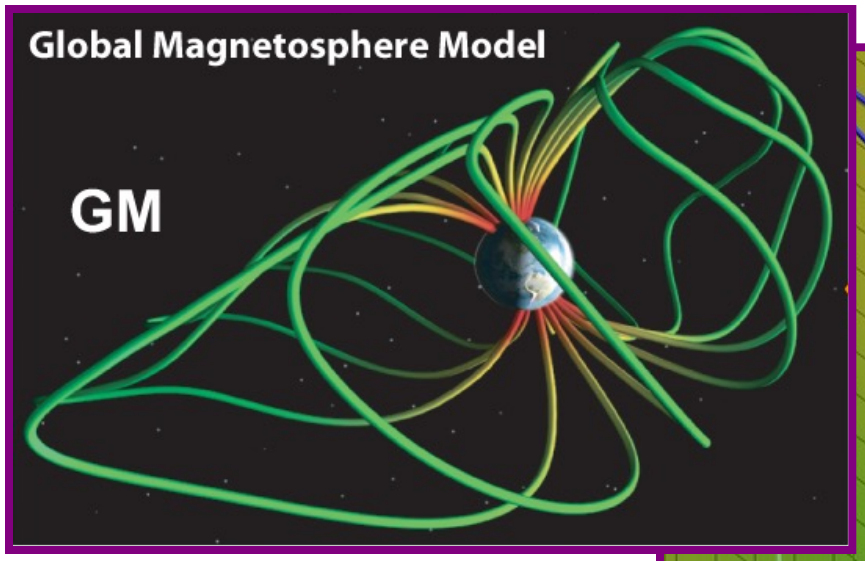



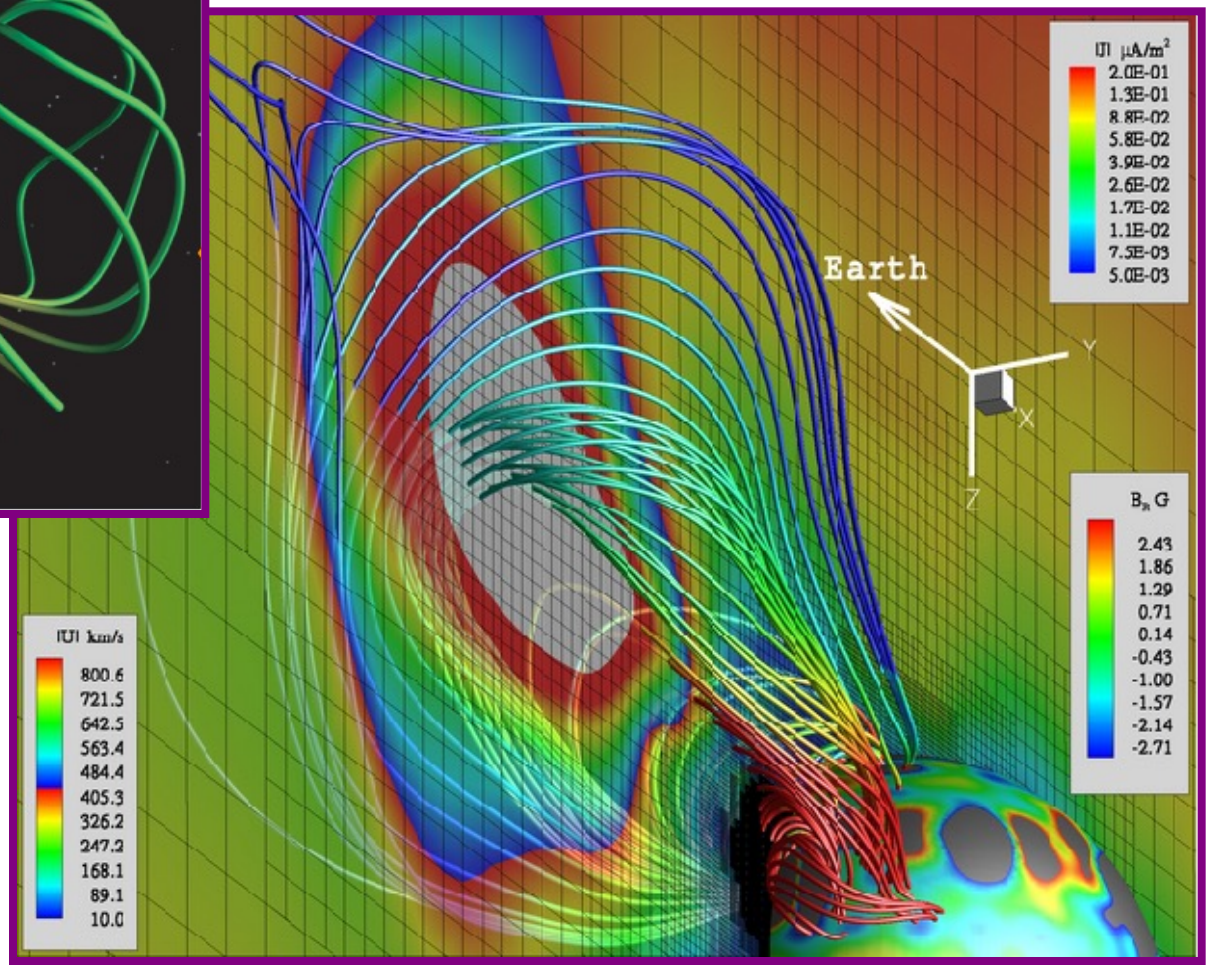

6 kpc ⇒ 100 AU (1,2000倍)

# BATS-R-US (K. G. Powell)
# Space Weather

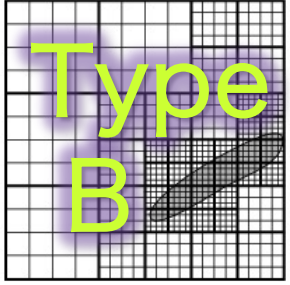


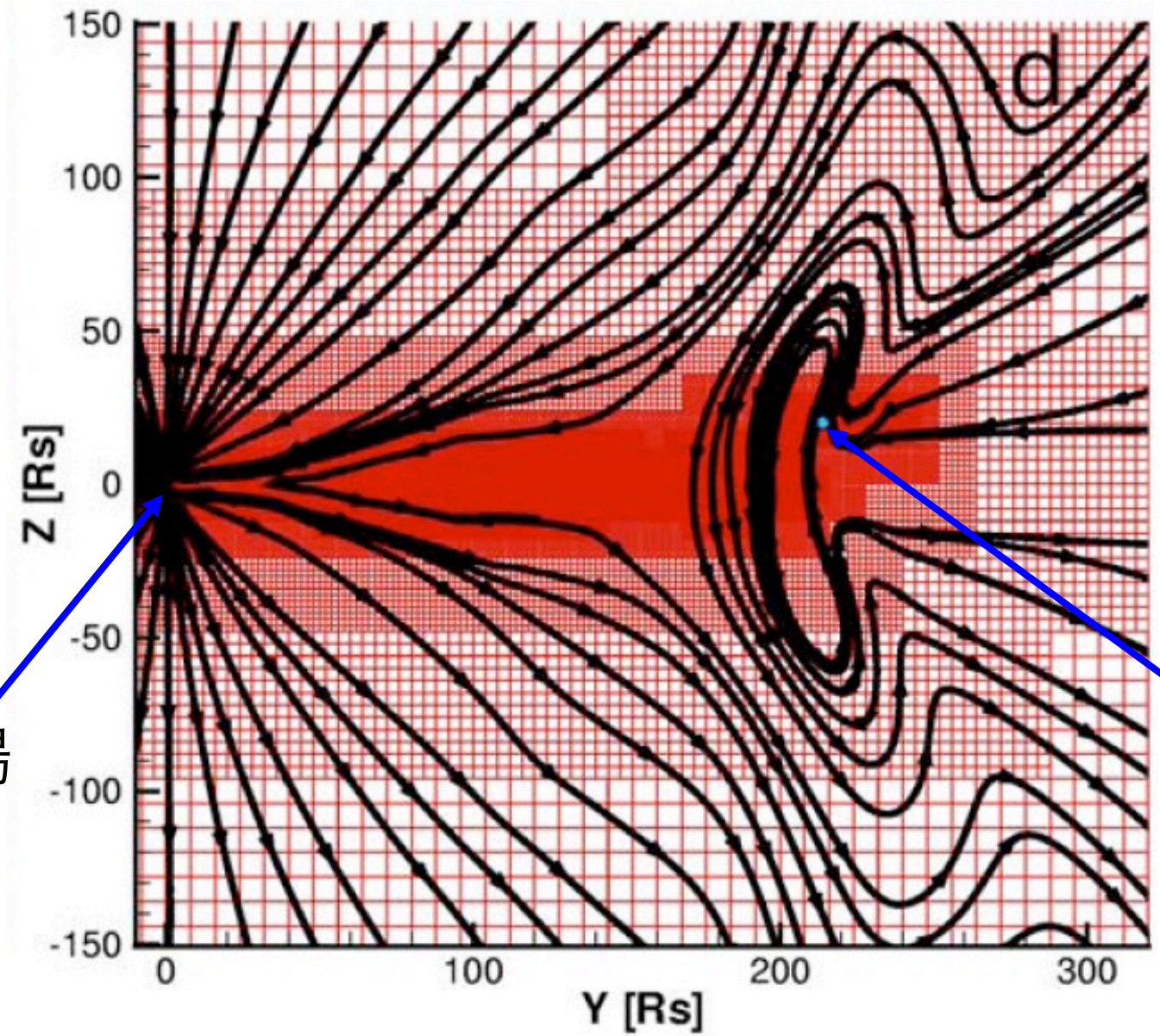

Block-structured grid (oct-tree type)



# Coronal Mass Ejection by BATS-R-US Manchester IV et al. (2004)




太陽

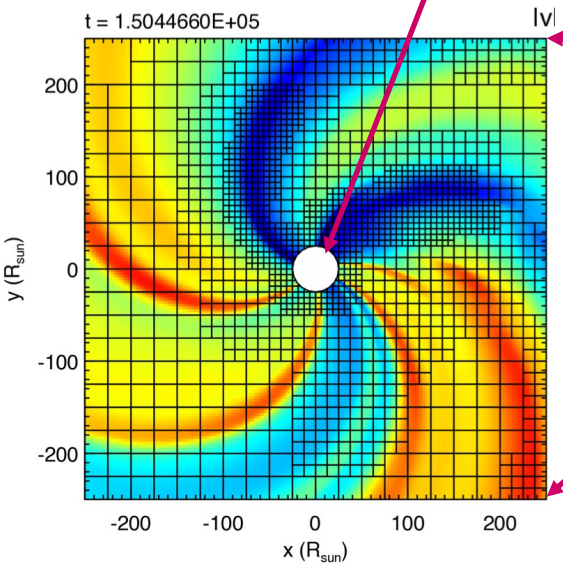
地球

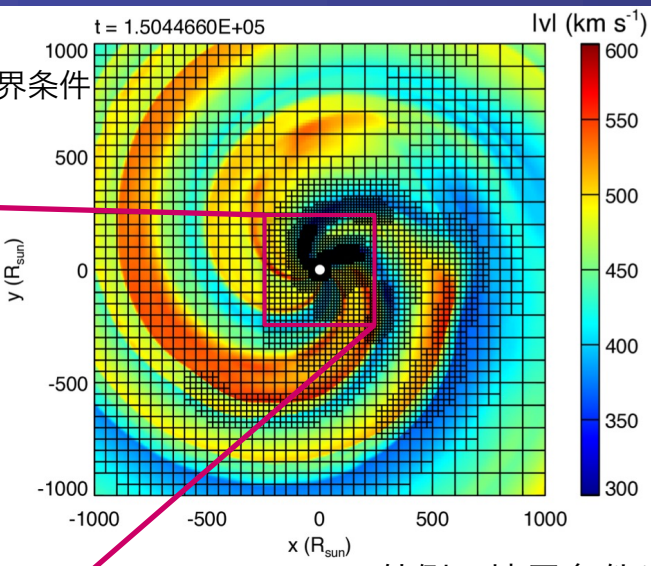
# 太陽圏シミュレーション

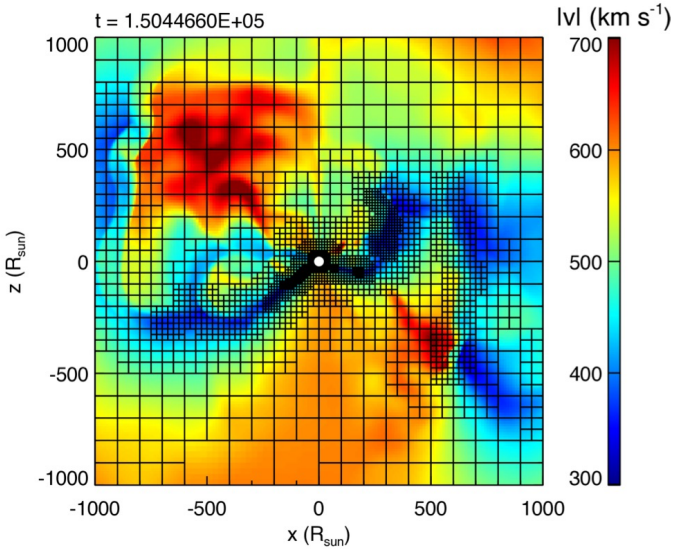
Matsumoto+
doi:10.1088/1742-6596/1225/1/012008

速度

内側の境界条件
25 $R_{sun}$



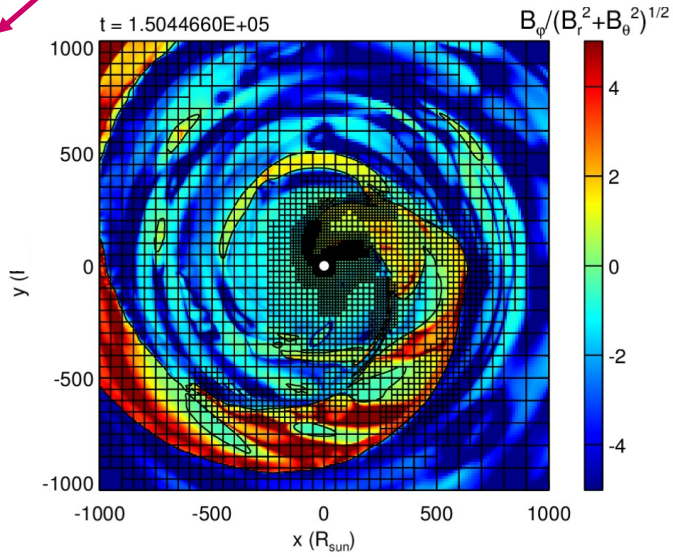



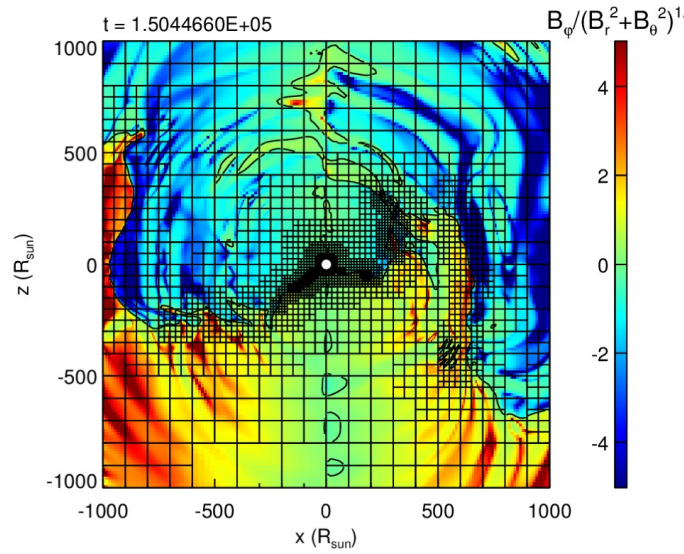




外側の境界条件は3000$R_{sun}$(=14 au)まで

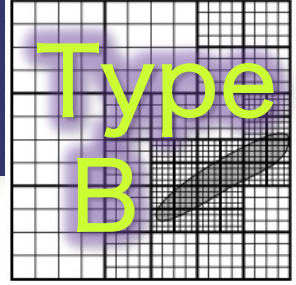
Bϕ

電流シートを分解する

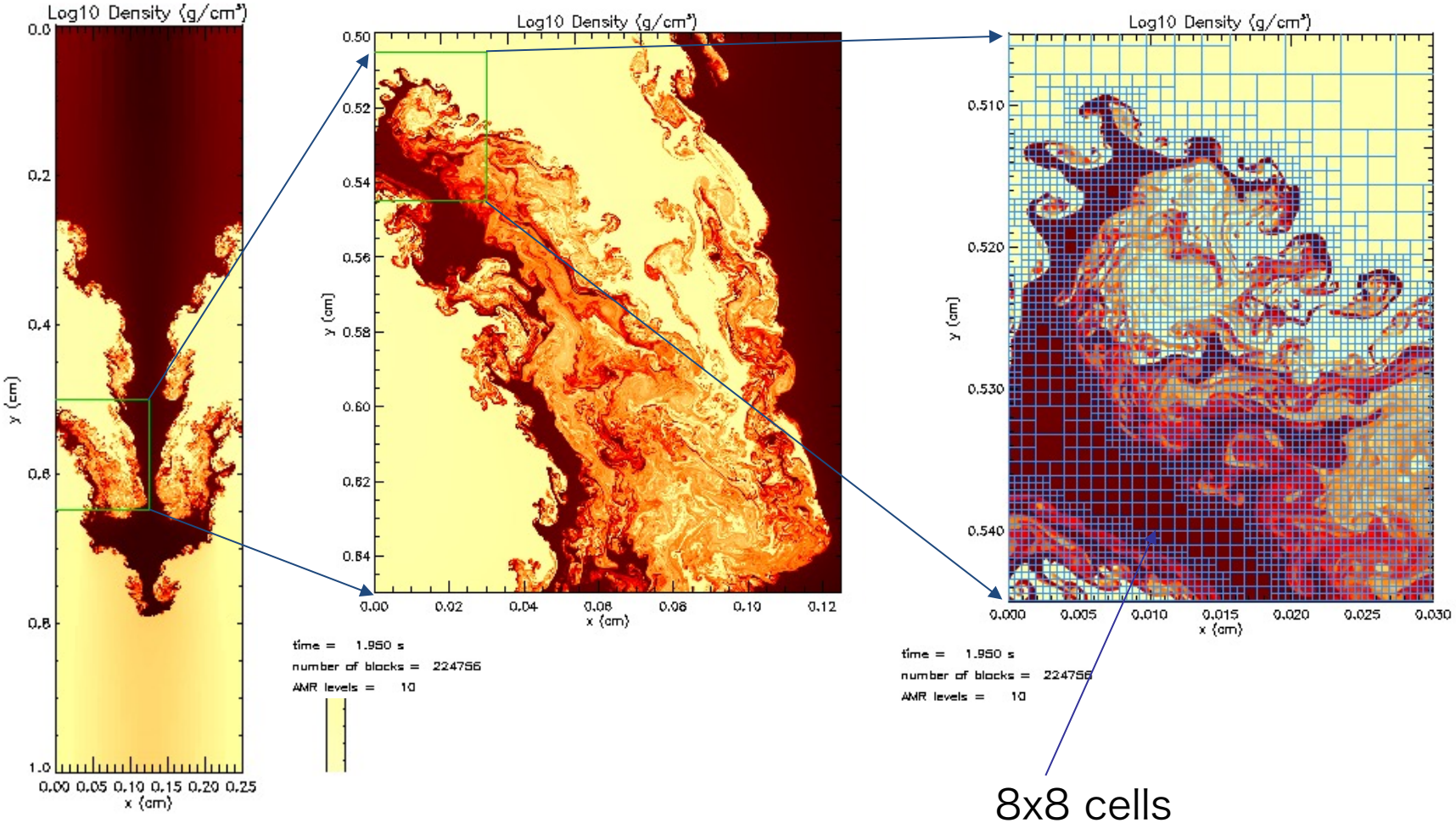


# FLASH (ASC, U-Chicago)
# Rayleigh-Taylor Instability



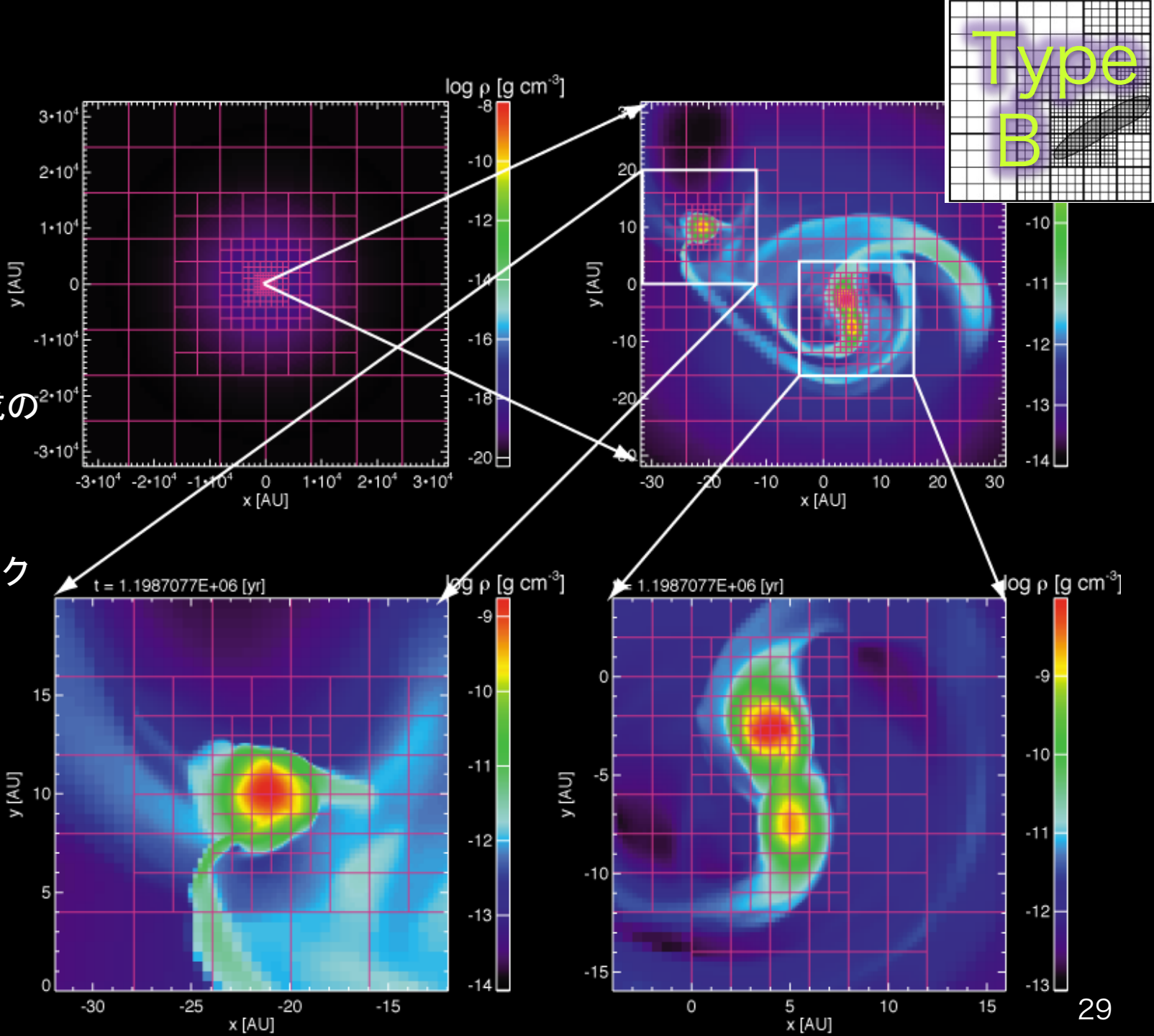

Block-structured grid (oct-tree type)




8x8 cells

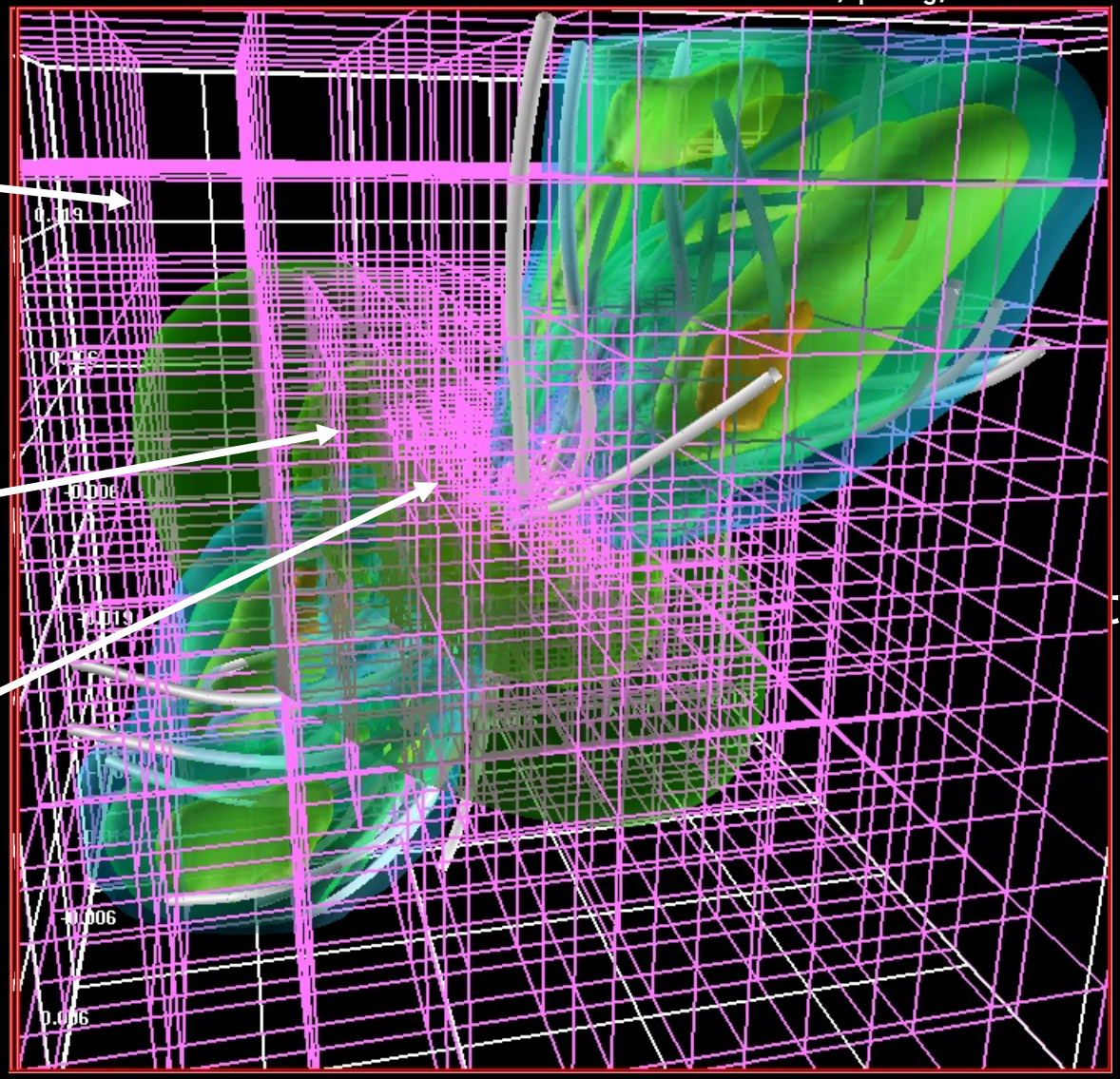
多重星系形成の
計算例
SFUMATO

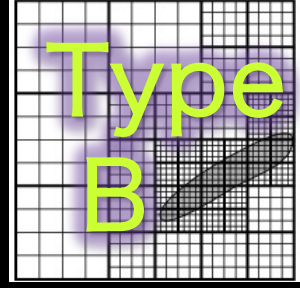
セル数/ブロック
$=8^3$
最大レベル
$=14$



# 分子雲コア⇒原始星（ファーストコア）
## 自己重力 + MHD

Matsumoto 2007
doi:10.1093/pasj/59.5.905



# 本当にAMRが必要ですか？

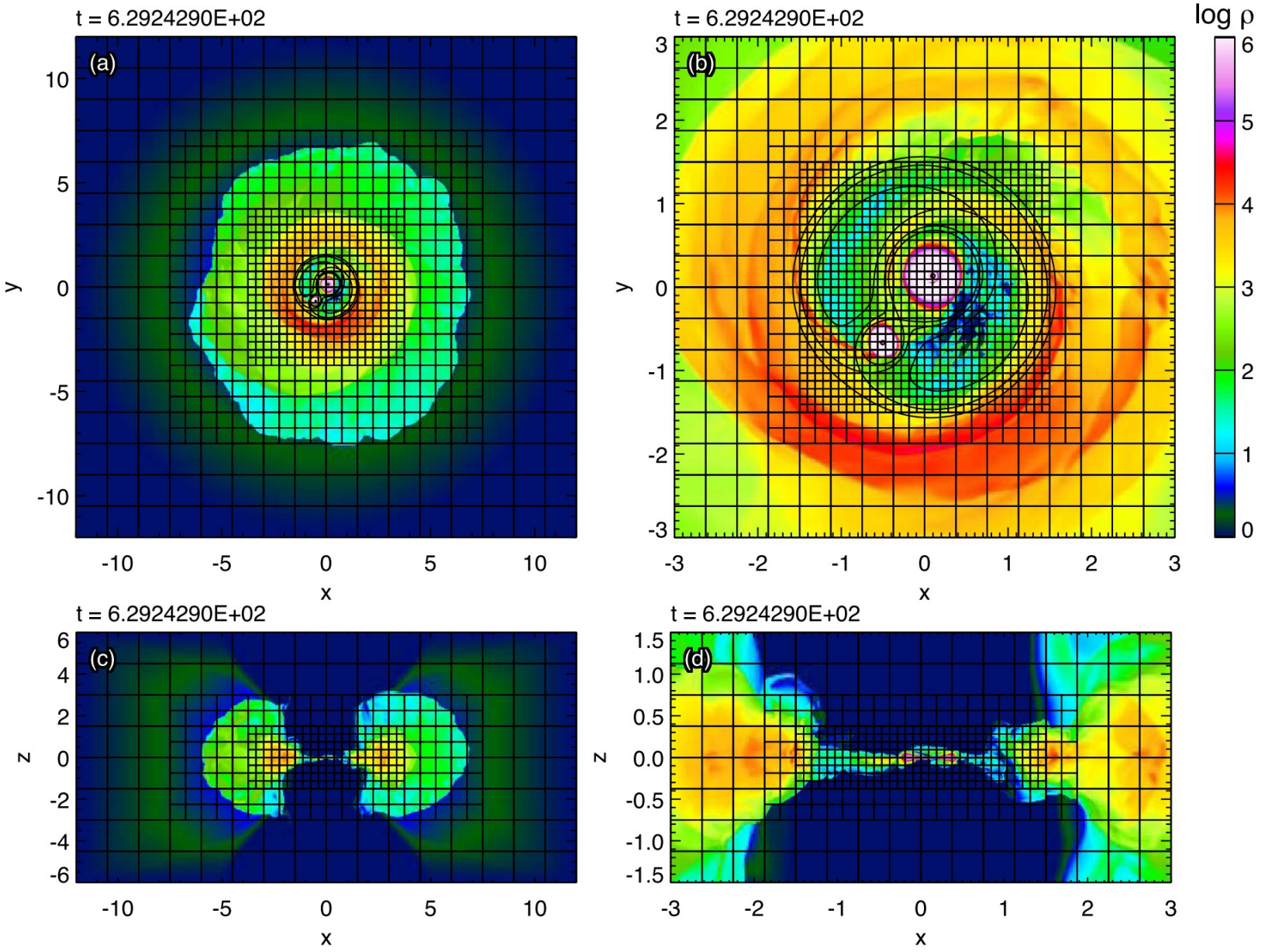
SMR/FMRで十分なのでは？

格子の**貼り替え**の悪さ

| | |
|---|---|
| 細かくする | 高次のノイズ発生<br>エントロピーの減少 |
| 粗くする | 運動エネルギーの熱化 |

# FMR/SMRの使用例

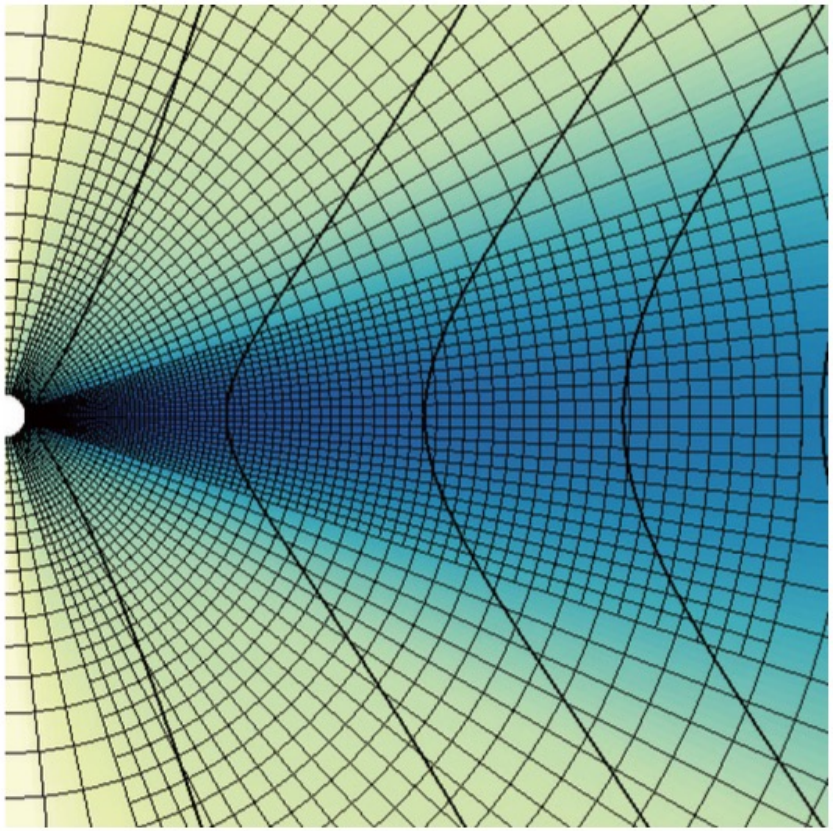
Matsumoto+ 19
doi:10.3847/1538-4357/aaf6ab

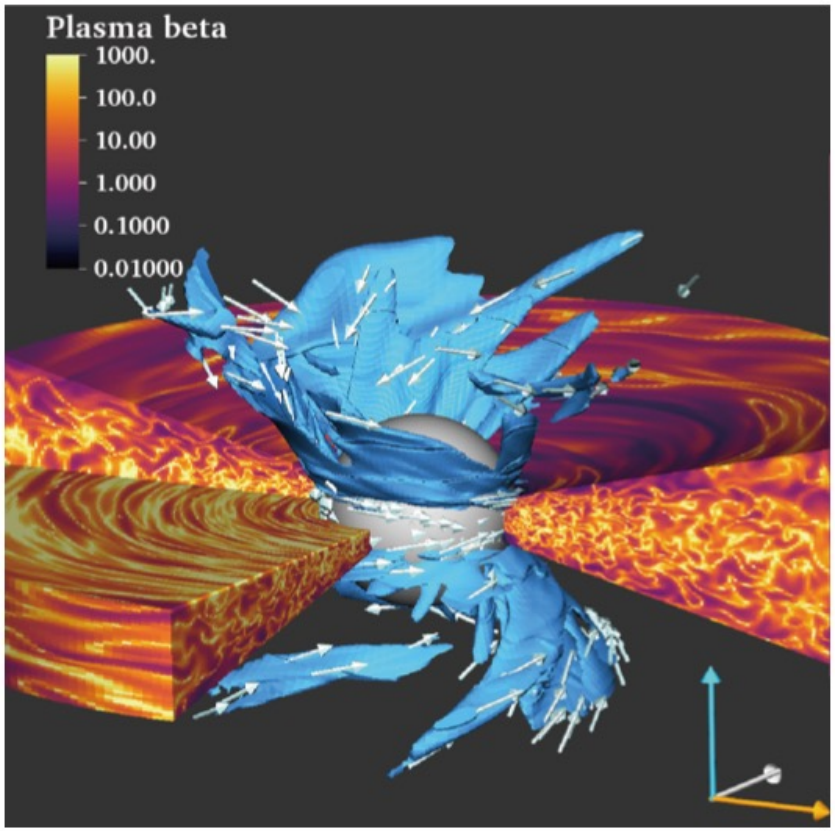


# FMR/SMRの使用例

Athena++ を用いたSMR

$\Delta r_{i+1}/\Delta r_i = 1.007$



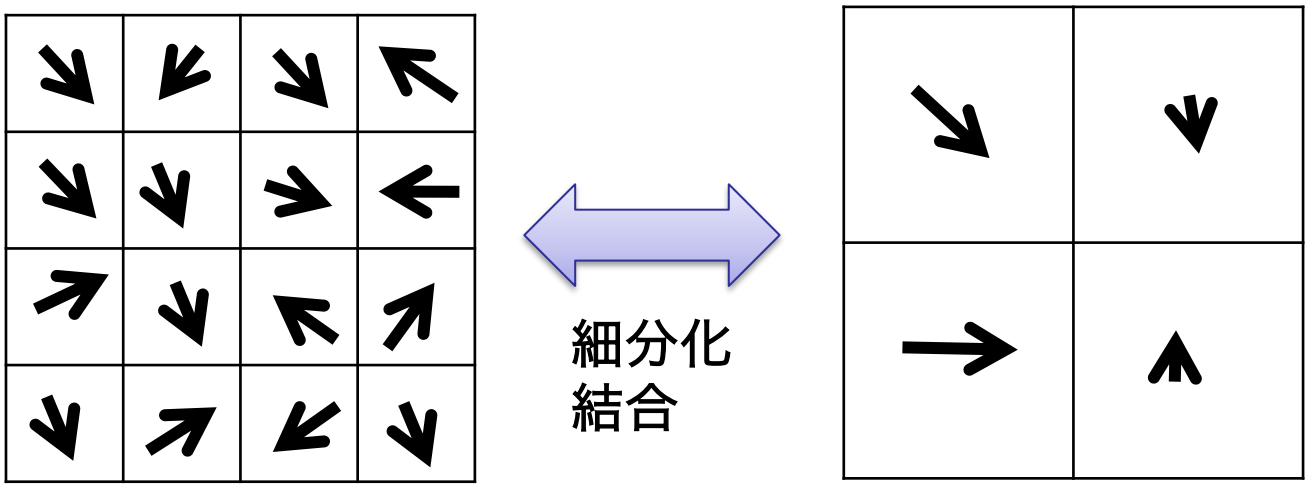



Takasao+ 2018
doi:10.3847/1538-4357/aab5b3

# 定説と実際

# AMRは乱流が不得意であるという「定説」

質量・運動量・エネルギーが保存するように細分化と結合




$\rho\vec{v}$ は保存するが、$|\rho\vec{v}|^2$ は保存しない。

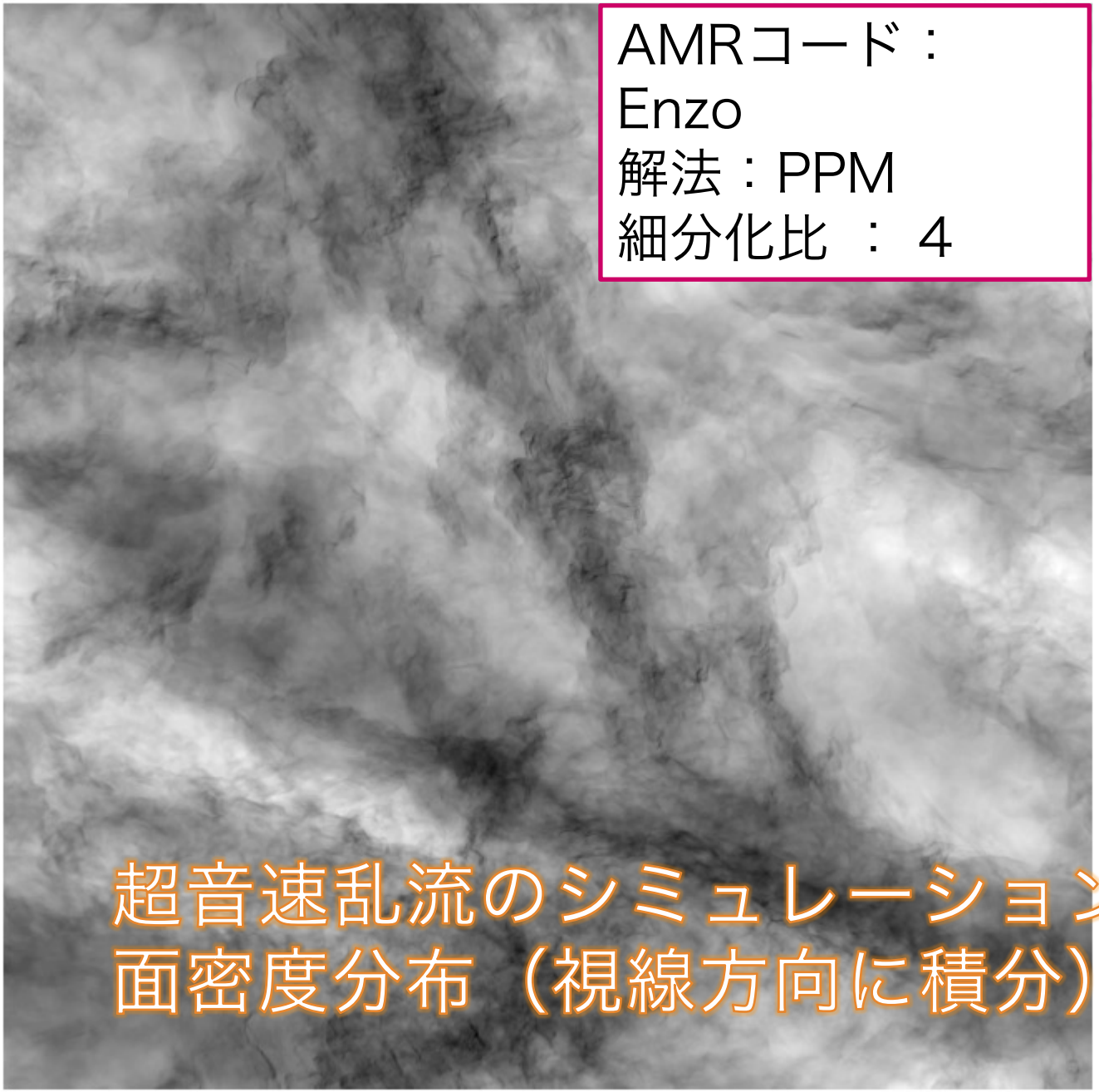
全エネルギー $(E_K+E_{th})$ は保存する。

運動エネルギー$(E_K)$が熱エネルギー$(E_{th})$になる。

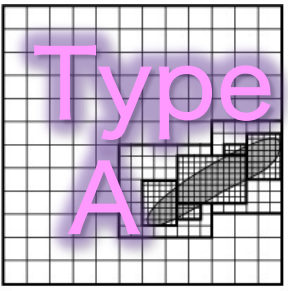
人為的な加熱に相当する。

AMRコード：
Enzo
解法：PPM
細分化比 ： 4

分解能
$1024^3$
相当



超音速乱流のシミュレーション
面密度分布（視線方向に積分）

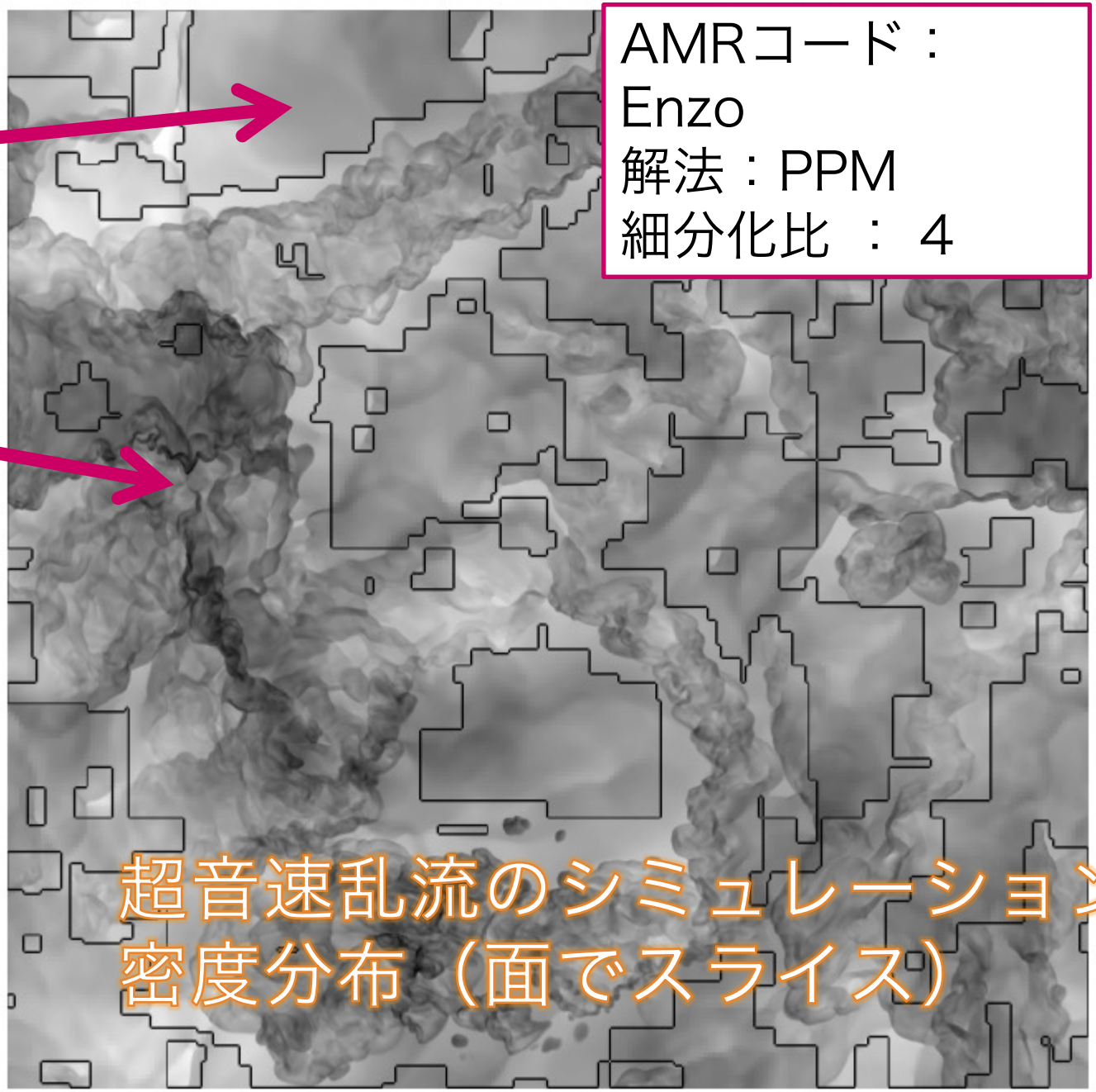



Kritsuk et al. 2006

level 0
$256^3$

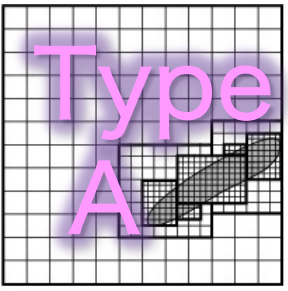
Level 1
$1024^3$
65%

AMRコード：
Enzo
解法：PPM
細分化比 ： 4



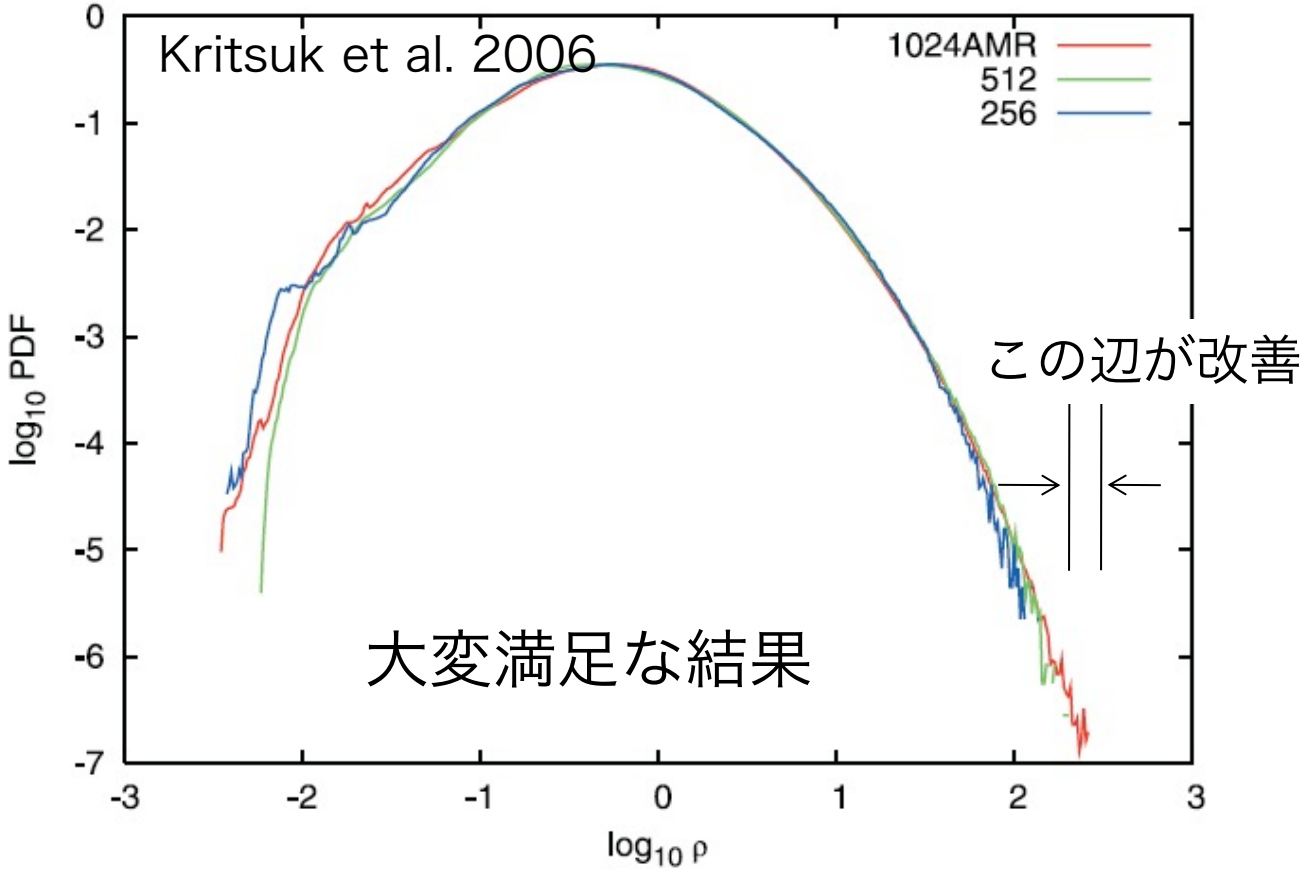
超音速乱流のシミュレーション
密度分布（面でスライス）

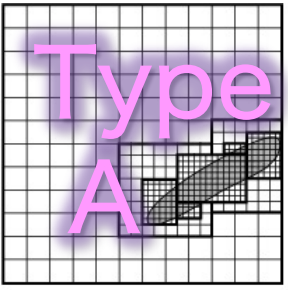
超音速乱流ではOK
亜音速乱流は微妙




Kritsuk et al. 2006

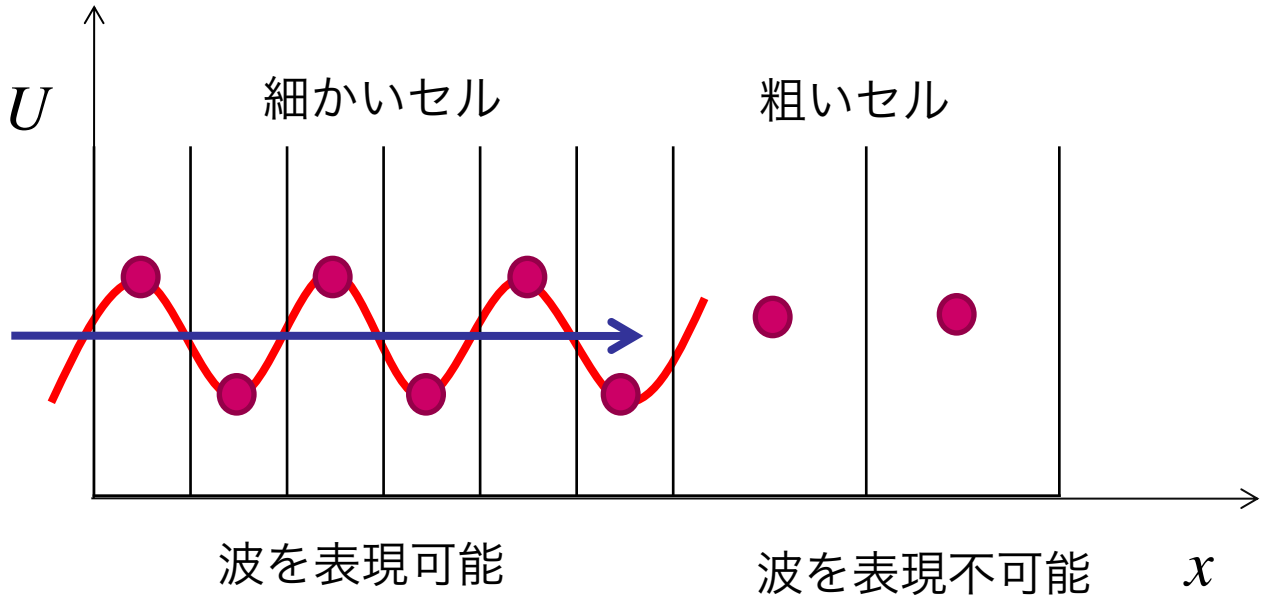
# 密度のPDF (確率分布関数)
# AMRと一様格子の比較



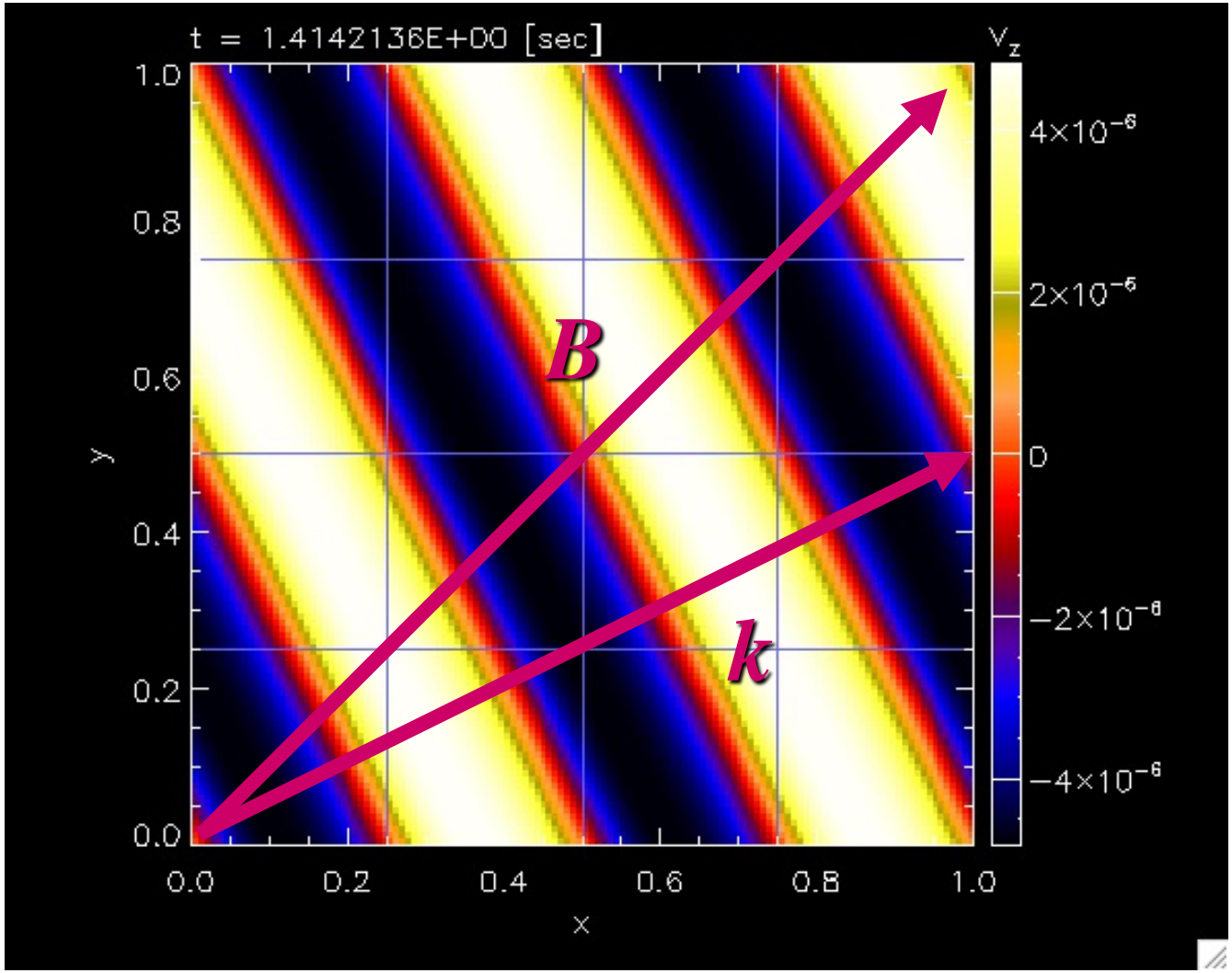

超音速乱流ではAMRは有効（衝撃波の補足）
亜音速乱流では不向き（渦の補足）

# AMRをちゃんと作れば「波は反射しない」という「誤解」



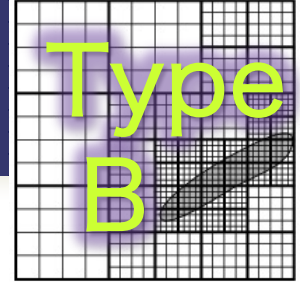

保存形式で解いているけど波のエネルギーはどこへ？
→　反射するしかない

# Convergence test for MHD Alfven wave

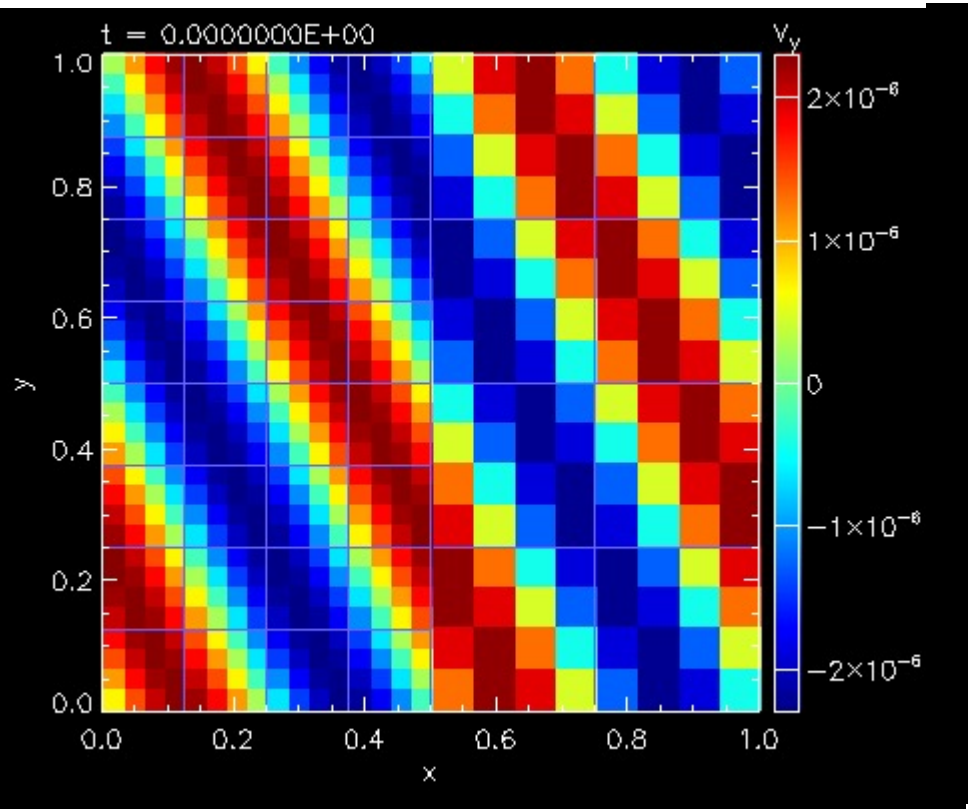


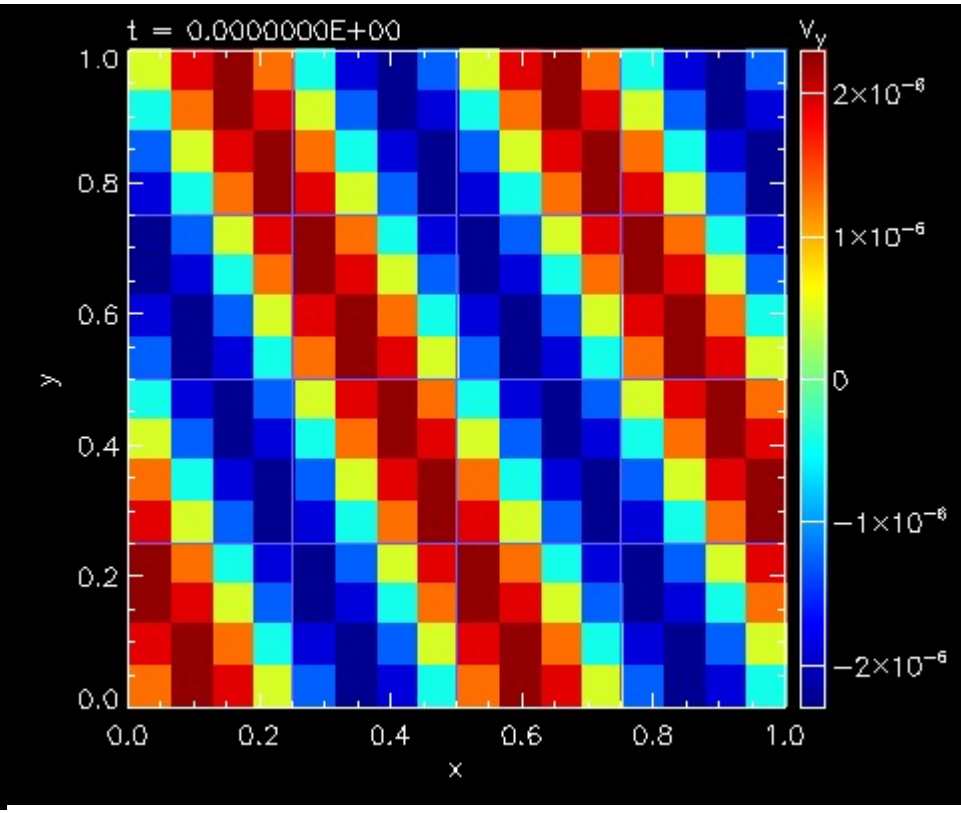

紙面に垂直な
vとBが振動する
sin波

# しかし、波は境界で反射する。
# Fast wave の反射実験。




SFUMATO

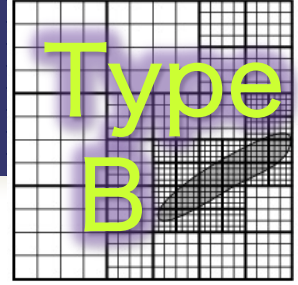


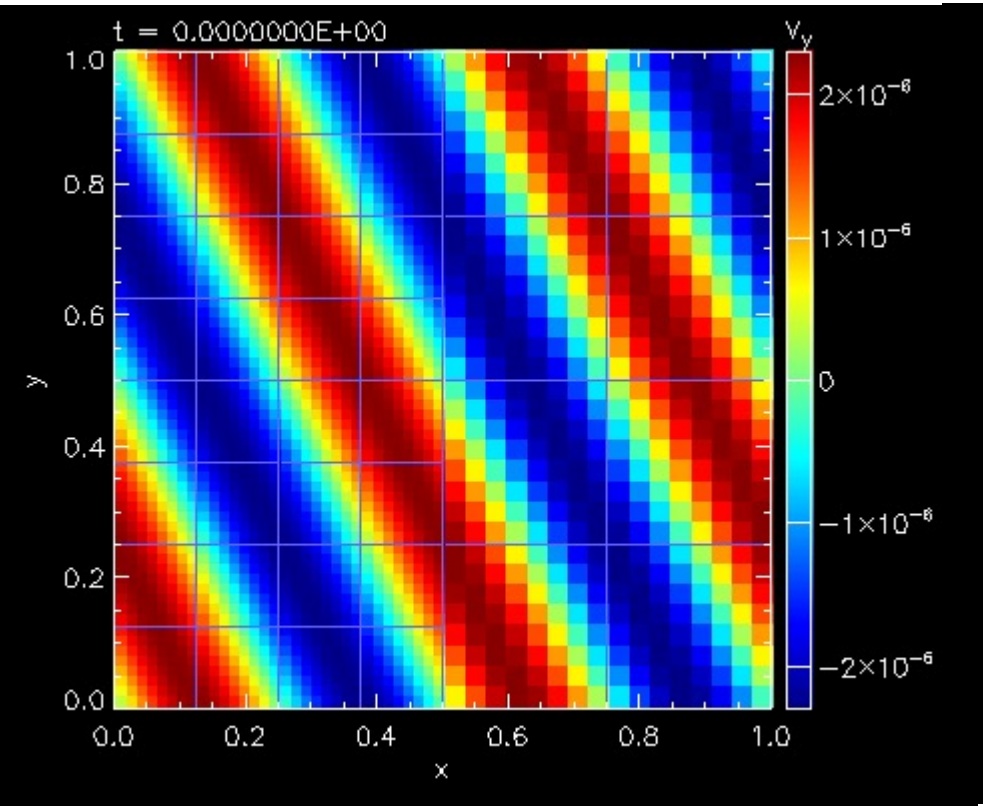

AMR
h = 1/32, 1/16

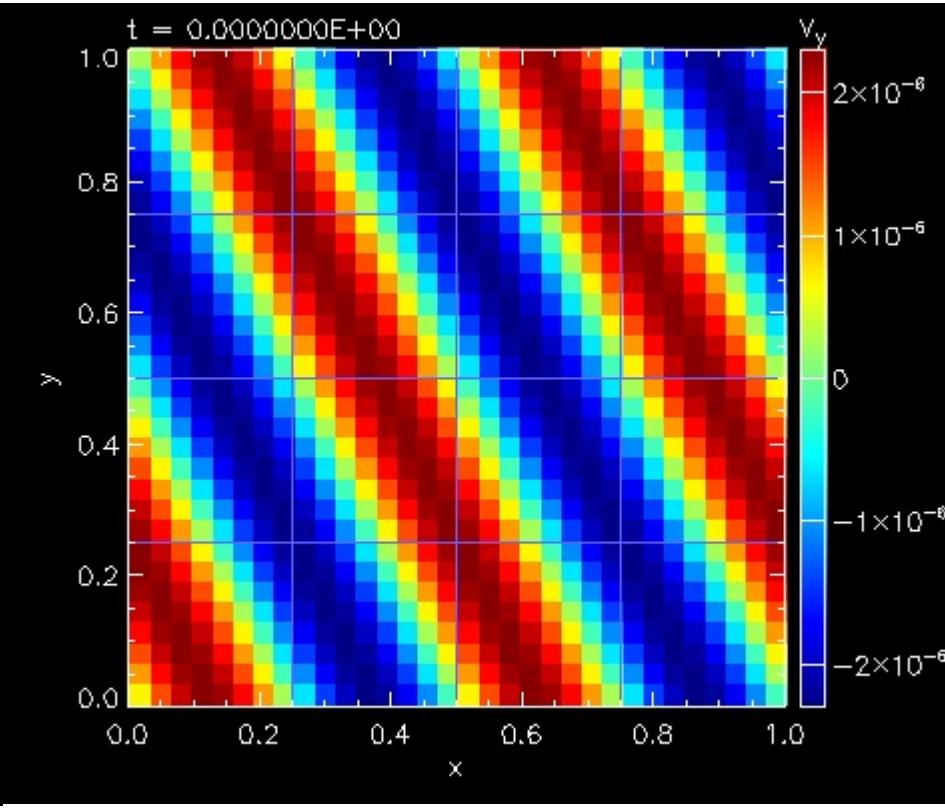
一様格子
h = 1/16

# 波は境界で反射する。
# AMRでは減衰が少ない。




SFUMATO

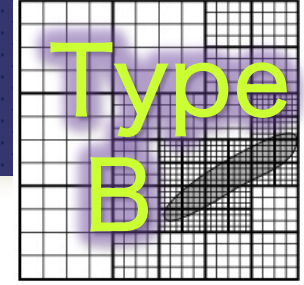


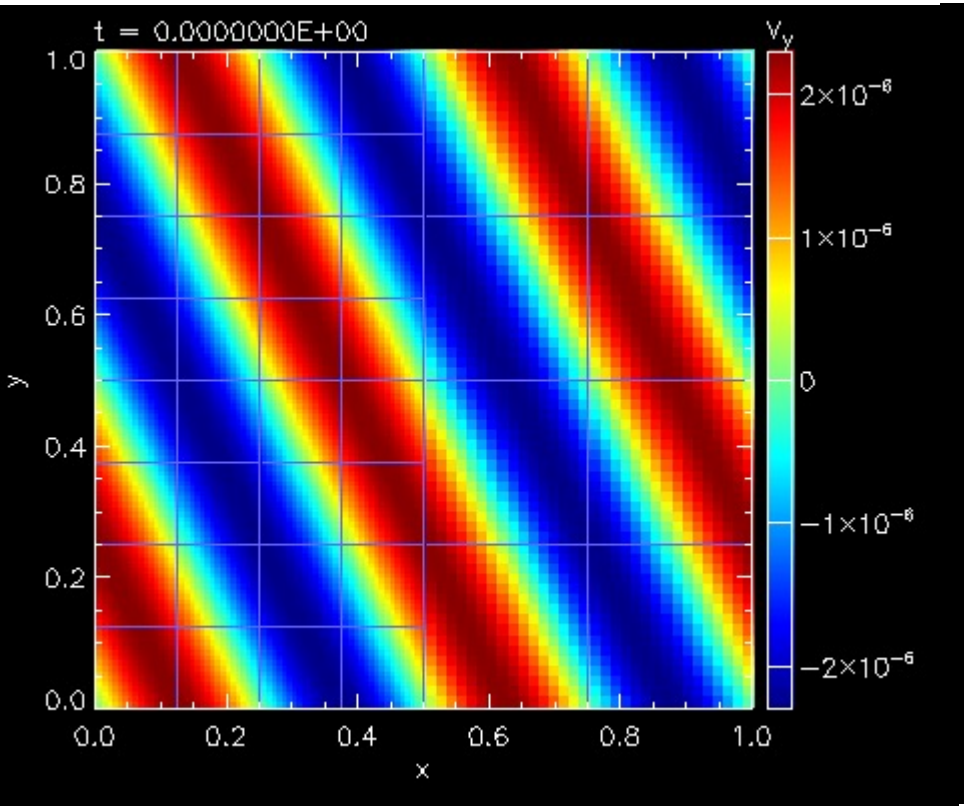

AMR
h = 1/64, 1/32

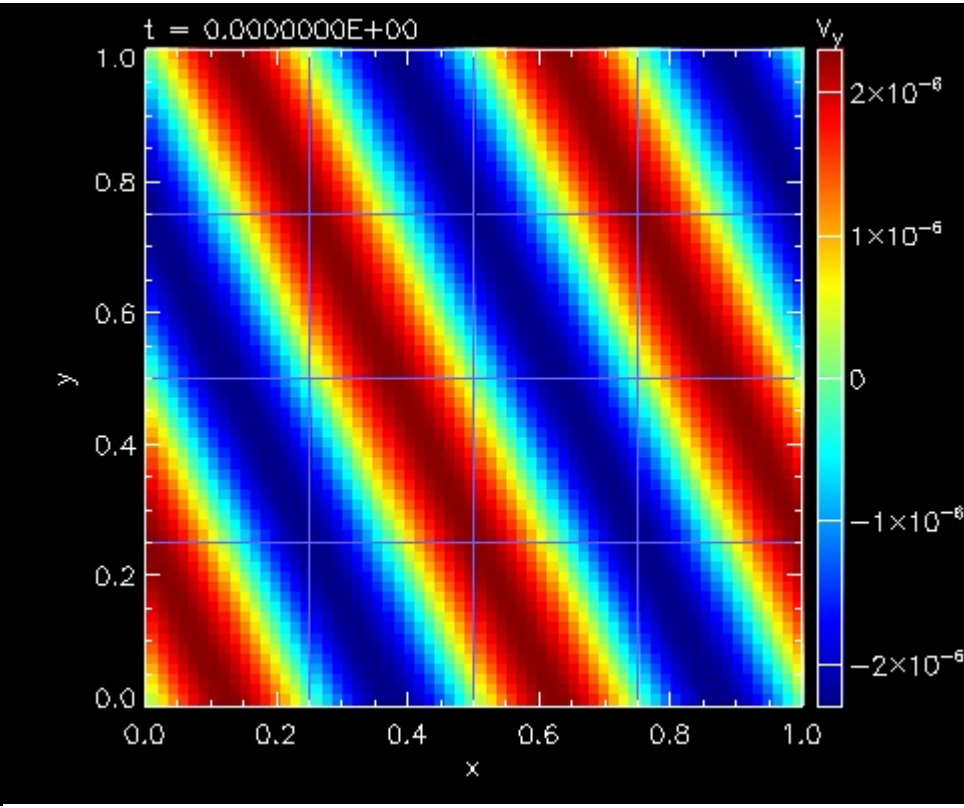
一様格子
h = 1/32

# 格子が細かくなると反射は少なくなる。




SFUMATO

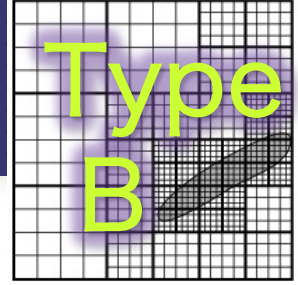


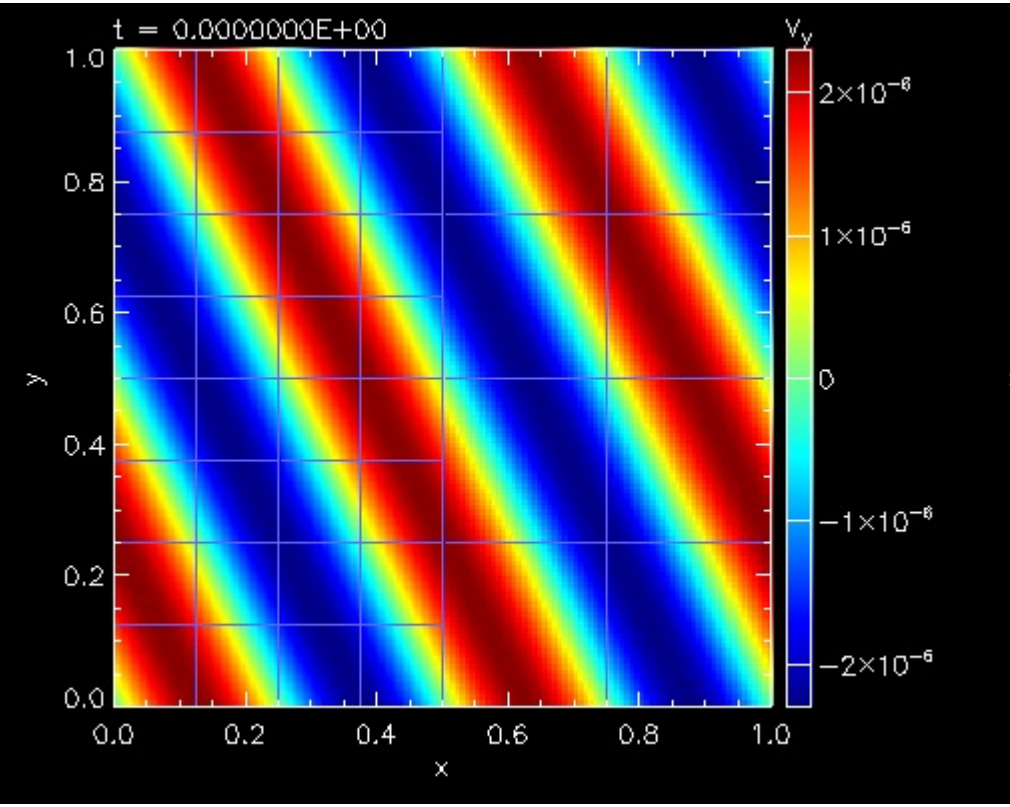

AMR
h = 1/128, 1/64

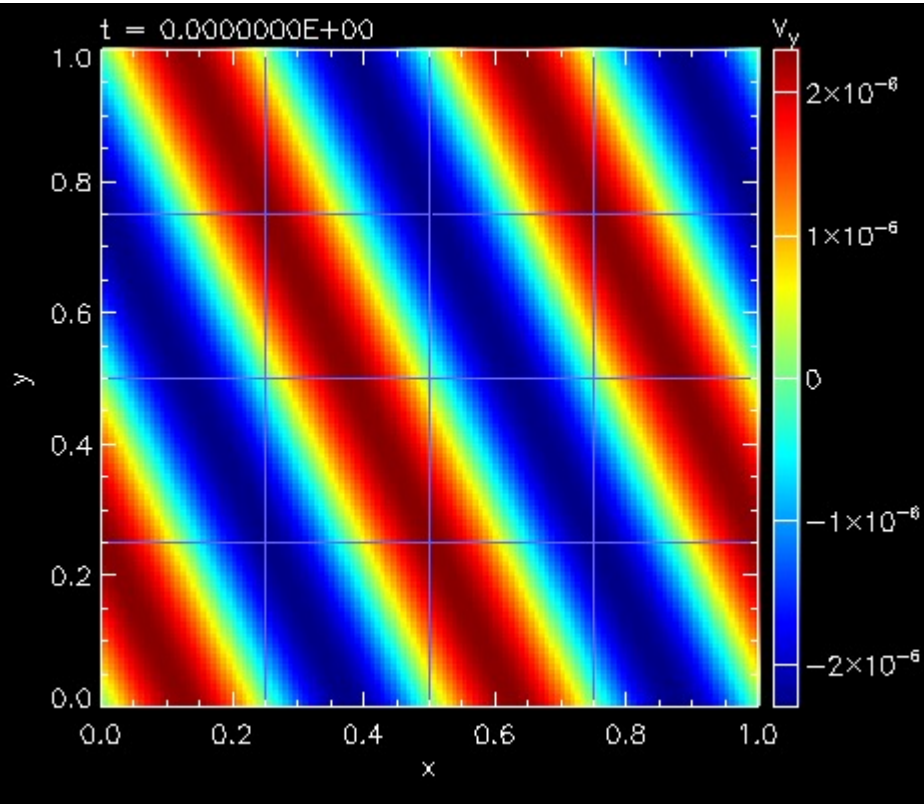
一様格子
h = 1/64

# 格子が細かくなると、
# ほとんど反射は目視できない。




SFUMATO



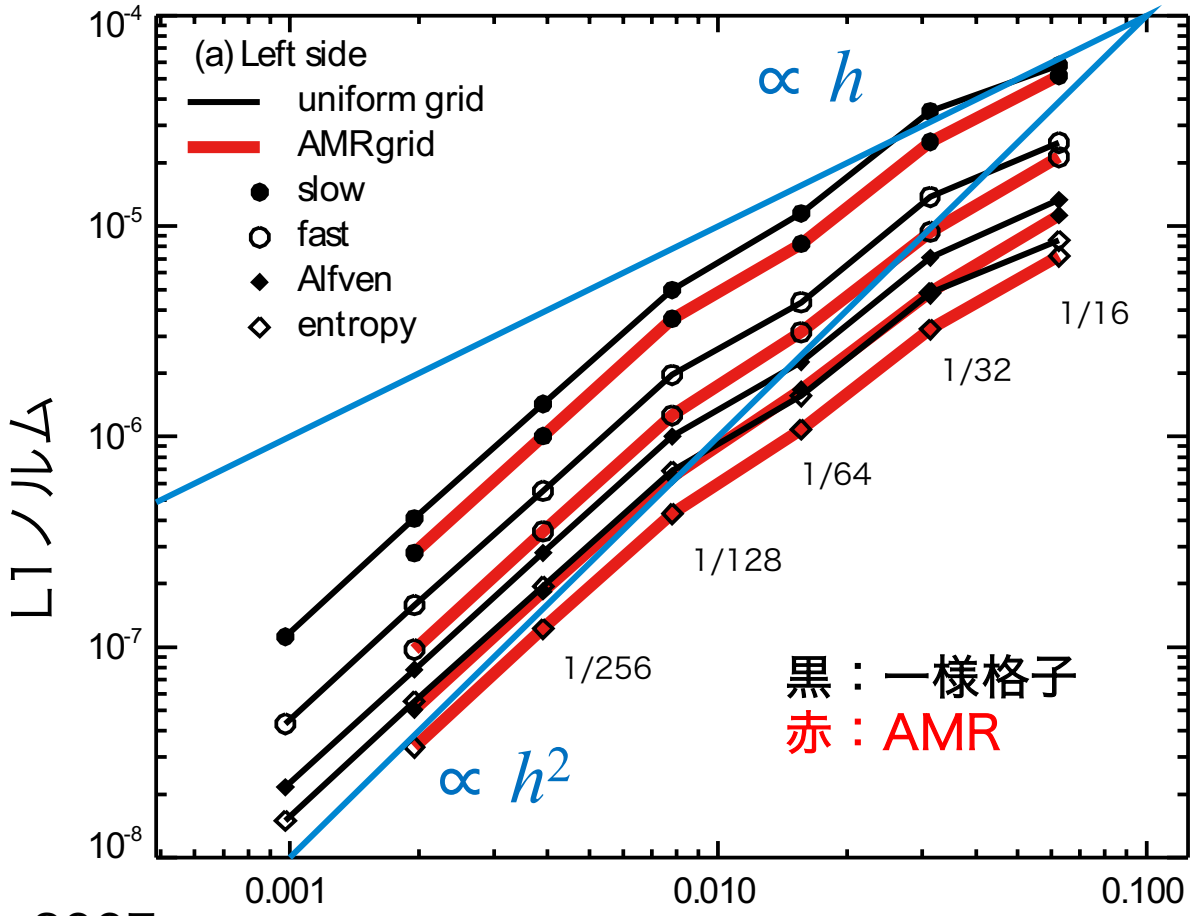

AMR
h = 1/256, 1/128

一様格子
h = 1/128

# 反射してもスキームの精度を達成する。

２次精度 @ h << 1
１次精度 @ h～1

SFUMATO




Matsumoto 2007
doi:10.1093/pasj/59.5.905

Type B

# 実装

# AMRコードを作るか作らないか

## 作る場合

- 利点：　新しい物理の導入が容易。
- 欠点：　膨大なテスト計算の必要性。
  開発時間の必要性
- 開発期間：　流体（MHD）を「書く」だけなら数ヶ月間。
  - 流体部分はラク。松本は流体部分を2週間で集中的に書いた。
  - 自己重力部分の開発は苦労した。
  - 可視化にも苦労した。一番苦労した？
- 効率よく開発するには
  - 流体などの既存のソルバを流用する。
  - はじめから3次元版を作る。
  - はじめから並列版を作る。
  - 既存のAMRライブラリ（paramesh, chombo）を利用するのも一考かと。

# AMRコードを作るか作らないか

## 作らない場合

- 既存の公開コードを利用する。
  現在の主流になりつつある。
  シミュレーションコードの複雑化が原因。

- 利点：　手軽に利用できる。開発の不要（テスト計算は必要）。

- 欠点：　改造の困難さ。新しい物理を導入する場合に苦労する。

- アイディア勝負の場合（隙間産業）は、作らない方が良いと思う。

- 開発者と共同研究する。
  - ブレイクスルーの仕事では、開発者が共同研究者であることが多い。

# AMRコードとその他のコードの比較

| コード種別 | 3dhd | Nested grid | AMR |
|---|---|---|---|
| 行数 | 3,524<br>（自動生成後<br>6,994) | 14,037 | 58,822 |
| 言語 | Fortran77 + Perl<br>コード自動生成 | Fortran77 + 自作<br>cpp | Fortran90 |
| 並列化 | VPP Fortran<br>指示行の挿入 | なし/自動並列<br>指示行の挿入 | MPI並列 |
| バージョン | - | 2.12 | - |
| 実装機能 | 流体, 自己重力 | HD, MHD, registivity,<br>自己重力 | HD, MHD, 自己重力,<br>registivity, sink 粒子 |

# 解法

# AMR格子における解法

- ブロック構造格子
  - ブロック自体は普通の一様格子。
  - 粗細（親子）ブロックの関係（時間的・空間的）を工夫する。それだけ！
- 流体・MHD （双曲型偏微分方程式）

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

  - 基本コンセプトは Berger & Colella (1989) に尽きる。
  - TVD, Roe法, HLLD法, predictor-corrector 法 (時間空間2次精度)
  - MHDでは、∇・B　の処方箋に複数の流儀がある。一長一短。
- 自己重力（楕円型偏微分方程式）

$$\nabla\Phi=4\pi G\rho$$

  - AMRと相性が良いのはマルチグリッド法。
  - お勧め参考書「MULTIGRID」 Trottenberg et al. (2001)
- 拡散方程式（放物型偏微分方程式）

$$\frac{\partial U}{\partial t}=\kappa\nabla^2 U$$

  - 楕円型偏微分方程式と同じ

# 時間発展

# 双曲型方程式の解法：　２種類の時間発展

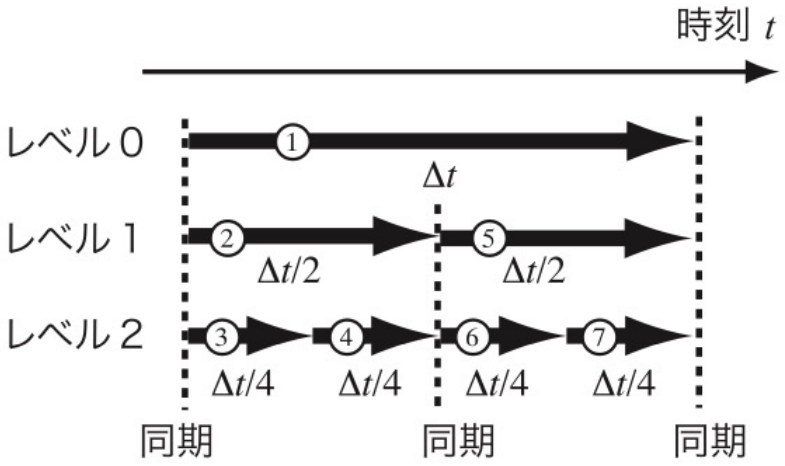
- 適合時間刻み (adaptive time step)
  - 空間だけでなく、時間も細分化する。
  - 親のタイムステップ≧子のタイムステップ
  - C.f., Berger & Cole l la (1989)
  - 自己重力なしの場合

CFL条件

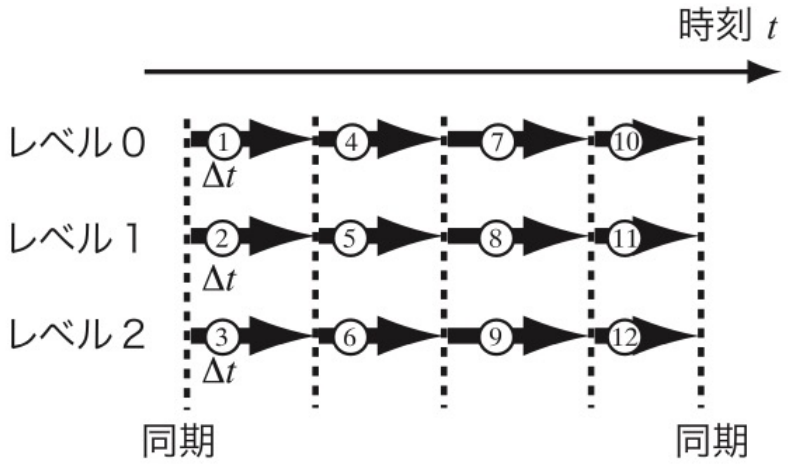
$$\Delta t < \frac{\Delta x}{v}$$

- 共通時間刻み (common time step)
  - 全てのグリッドレベルが同じタイムステップを持つ。
  - 自己重力系、輻射流体の場合　（長距離相互作用）

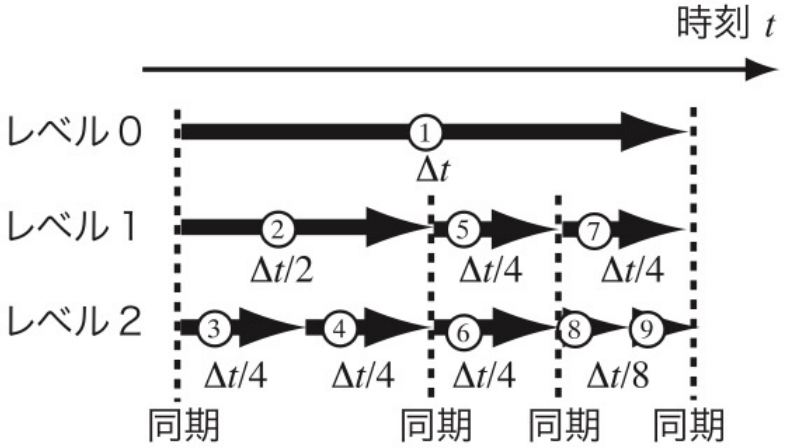
# 2種類の時間発展




(a) 適合時間刻み




(b) 共通時間刻み




(c) 独立時間刻み

独立時間刻は適合時間刻みより自由度大

$\Delta t^{\ell} = \Delta t^{\ell-1} 2^{-n}$ (ただし $n = 0, 1, 2, \cdots$)

$\Delta t^{\ell} \leqq \Delta t^{\ell}_{\mathrm{CFL}}$,

右図の例：⑤で時間ステップを半分

# 適合時間刻み vs 共通時間刻み

**適合時間刻み**

もっとも細かいグリッドレベルが1ステップ進むためのコスト
親のグリッドレベルは1/2ステップ
祖父のグリッドレベルは1/4ステップ

$$1+\frac{1}{2}+\frac{1}{4}+\cdots+\frac{1}{2^{l\max}}=\sum_{l=0}^{l\max}\frac{1}{2^l}\approx 2 \quad (l_{\max}>>1)$$

合計で2倍コストがかかる

**共通時間刻み**

もっとも細かいグリッドレベルが1ステップ進むためのコスト
親のグリッドレベルも1ステップ
祖父のグリッドレベルの1ステップ

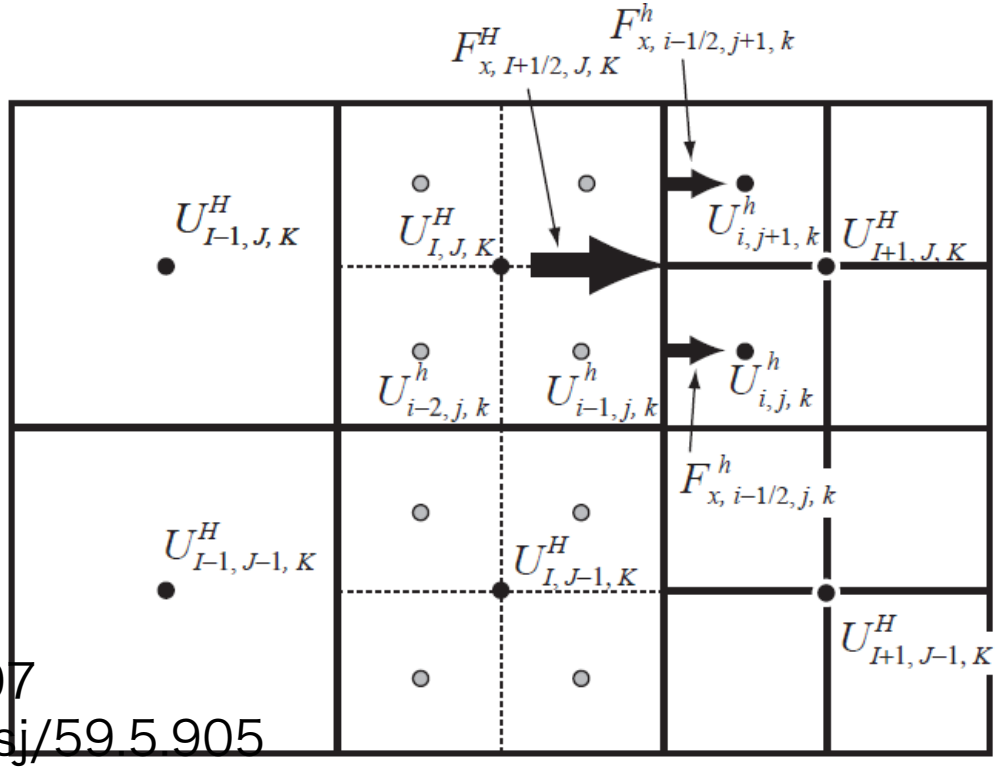
$$1+1+1+\cdots+1=\sum_{l=0}^{l\max}1=l_{\max}+1\approx l_{\max} \quad (l_{\max}>>1)$$

合計で段数倍コストがかかる

特別な理由がない限り、adaptive time step を採用するべき。
特別な理由の例：自己重力、輻射

# 双曲型方程式の解法; 細粗ブロックの境界

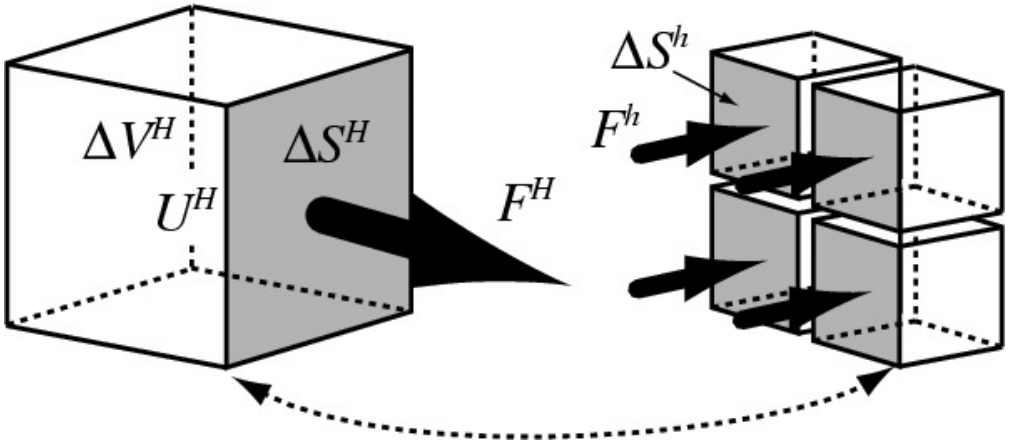
- 粗いブロックと接する**細かいブロック**
  - 粗いブロックを時間的に空間的に補間し、細かいブロックの境界条件にセット。
- 細かいブロックと接する**粗いブロック**
  - 細かいブロックの数値流速を使って、時間発展。




Matsumoto 2007
doi:10.1093/pasj/59.5.905

# 細粗境界での数値流束の保存

Matsumoto 2007
doi:10.1093/pasj/59.5.905




数値流束保存 $F^H \Delta S^H \Delta t^H = \sum_{\text{sub-cycle}} \sum_{\text{surface}} F^h \Delta S^h \Delta t^h$ が成立するように、

細かいブロックと隣接した粗いセルを補正する

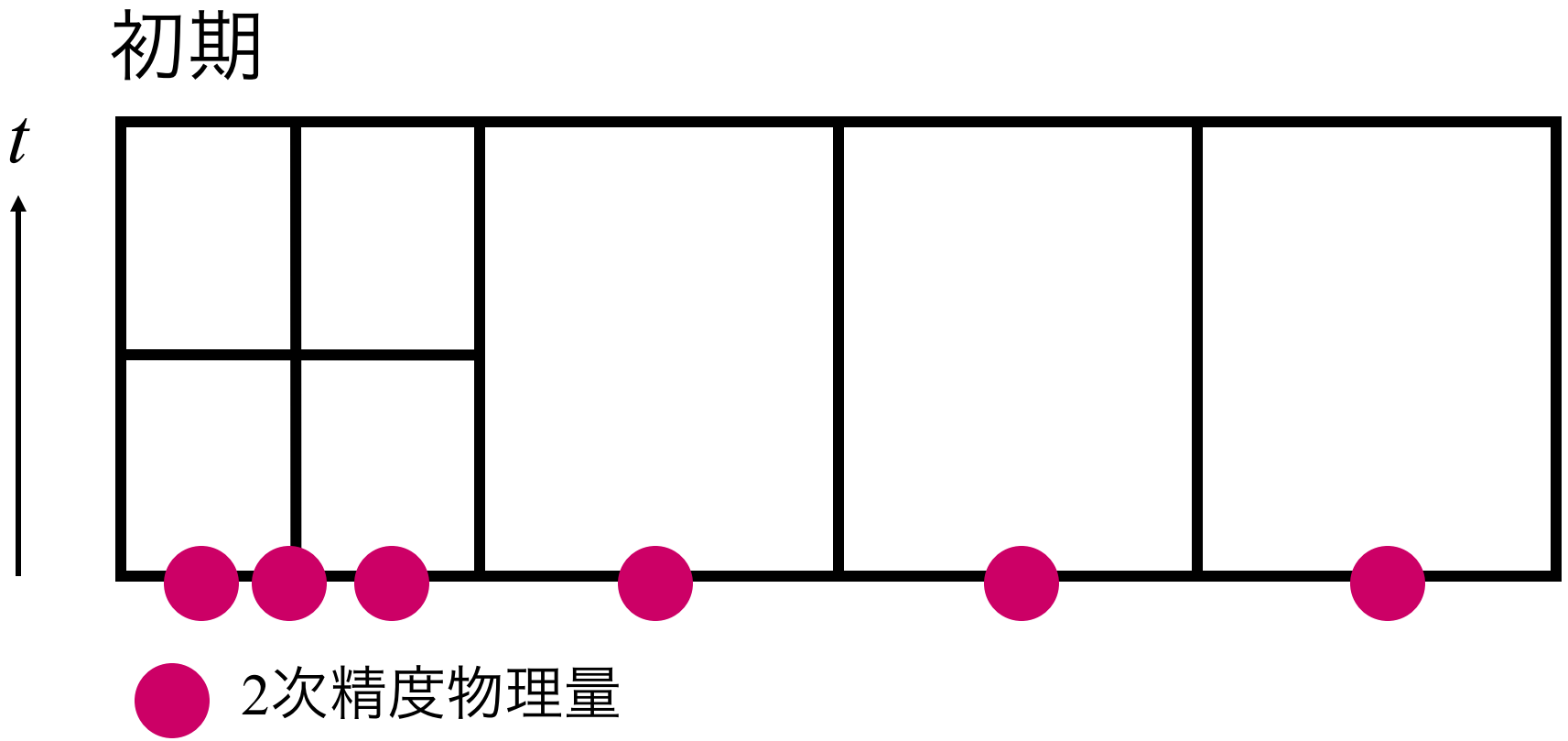
$$U^{H,\text{new}} = U^H - \frac{1}{\Delta V^H} \left( \sum_{\text{sub-cycle}} \sum_{\text{surface}} F^h \Delta S^h \Delta t^h - F^H \Delta S^H \Delta t^H \right)$$

# 時間推進の方法

1. 時間を進めるべきレベルを探す。
   a. もっとも時間が進んでいないレベルのうち、もっとも粗いレベル。
2. 時間刻み幅dtを求める。
   a. レベル0はクーラン条件から。
   b. それ以外は親のdtの半分
3. 境界条件の設定
   a. 兄弟ブロックからコピー。
   b. 親ブロックから時間・空間的に内挿。
   c. ユーザの境界条件。
4. 通常の方法で時間推進する。
   a. たとえば、predictor-corrector 法など。
5. 親レベルと同期した場合：
   a. 自分の解を親セルに代入する。
   b. 隣接する親セルの値を修正する（数値流束保存）。
6. 同期している全てのレベルを用いて細分化をする。

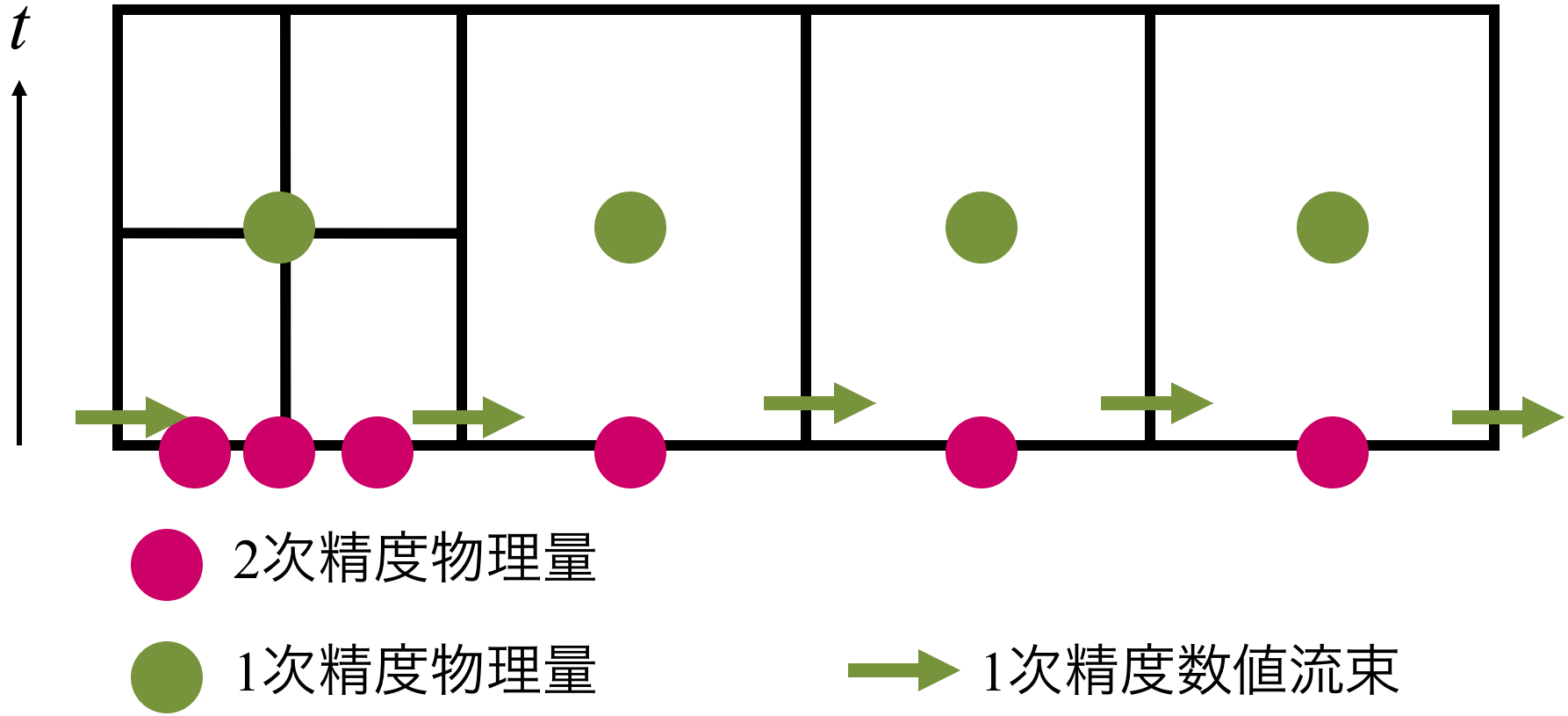
$l_{\mathrm{sync}}$ ～ $l_{\mathrm{max}} - 1$ を細分化して、
$l_{\mathrm{sync}} + 1$ ～ $l_{\mathrm{max}}$ を生成する。
$l_{\mathrm{sync}}$ = 同期している最粗レベル

# コードの詳細：　流体部分
# 数値流束補正・マルチタイムステップ

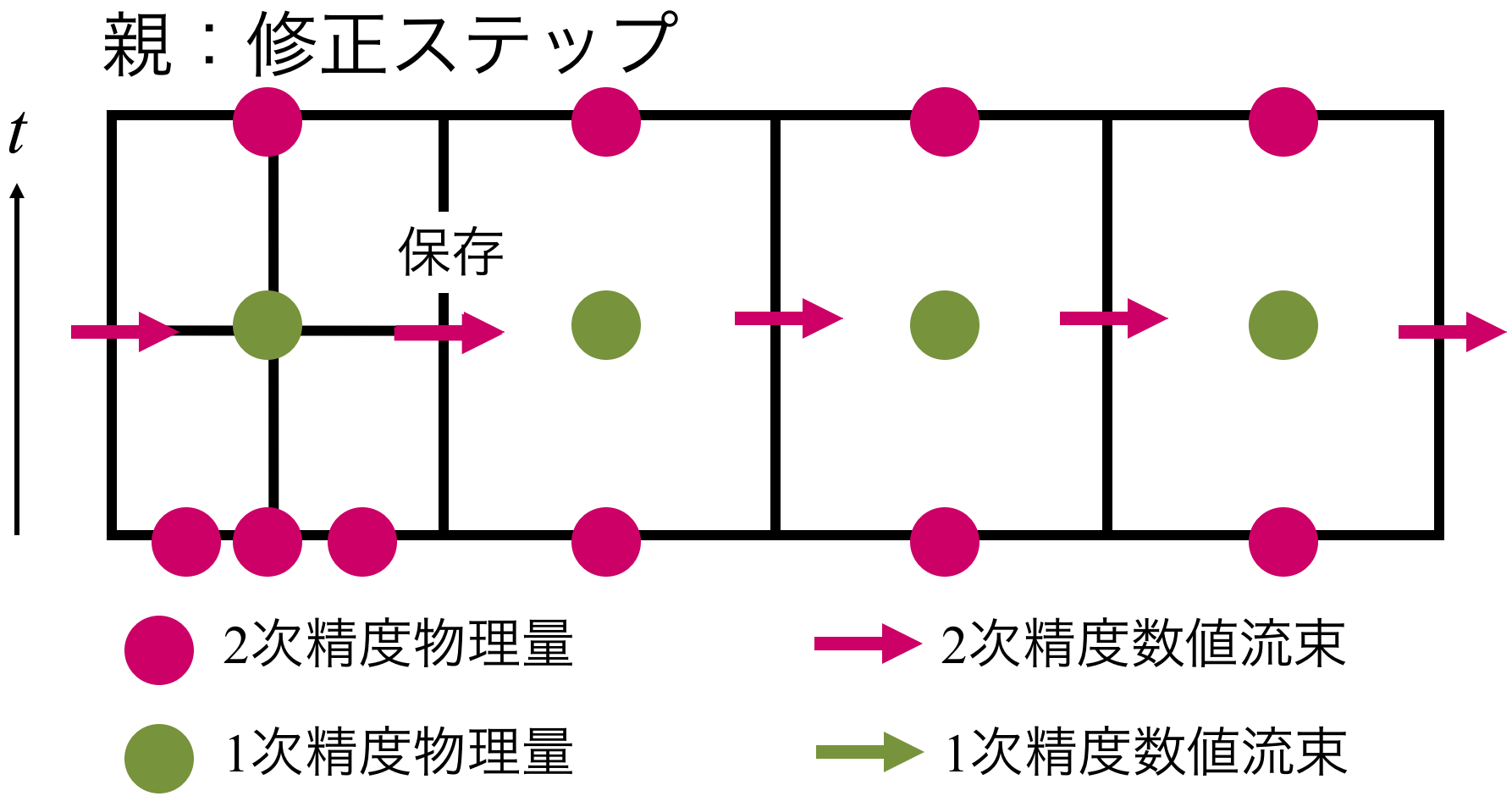


# コードの詳細：　流体部分
# 数値流束補正・マルチタイムステップ

親：予測ステップ

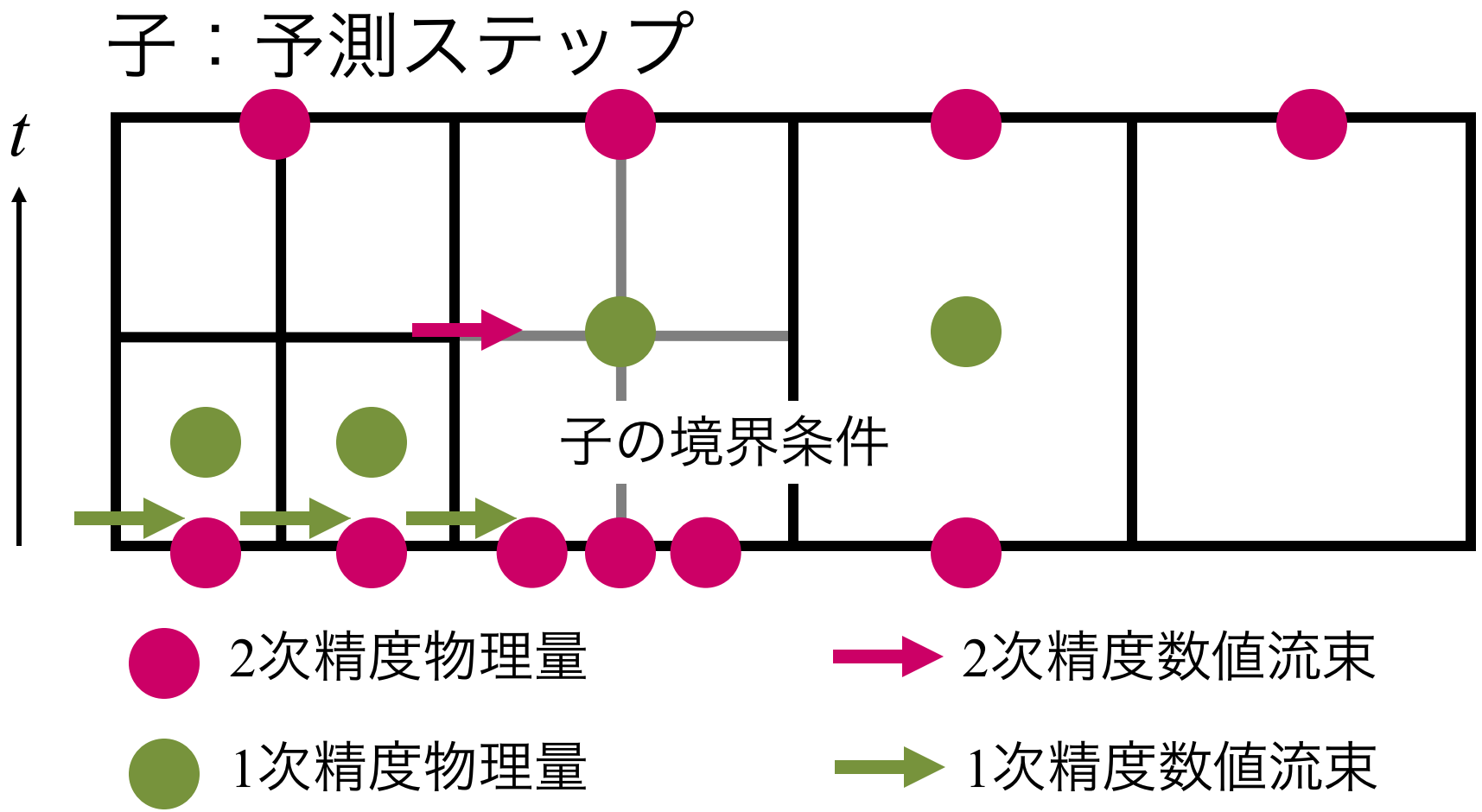


# コードの詳細：　流体部分
# 数値流束補正・マルチタイムステップ

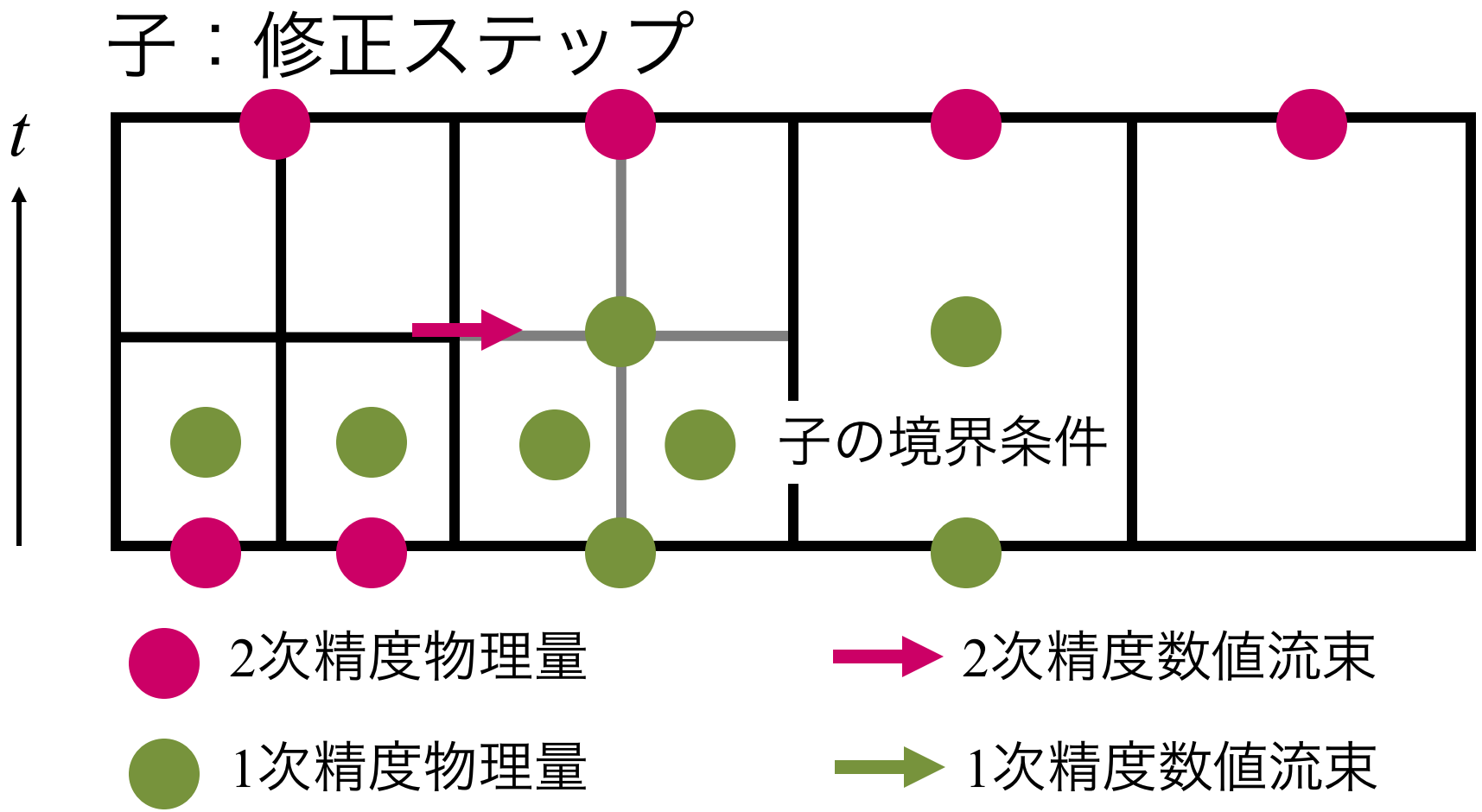



click

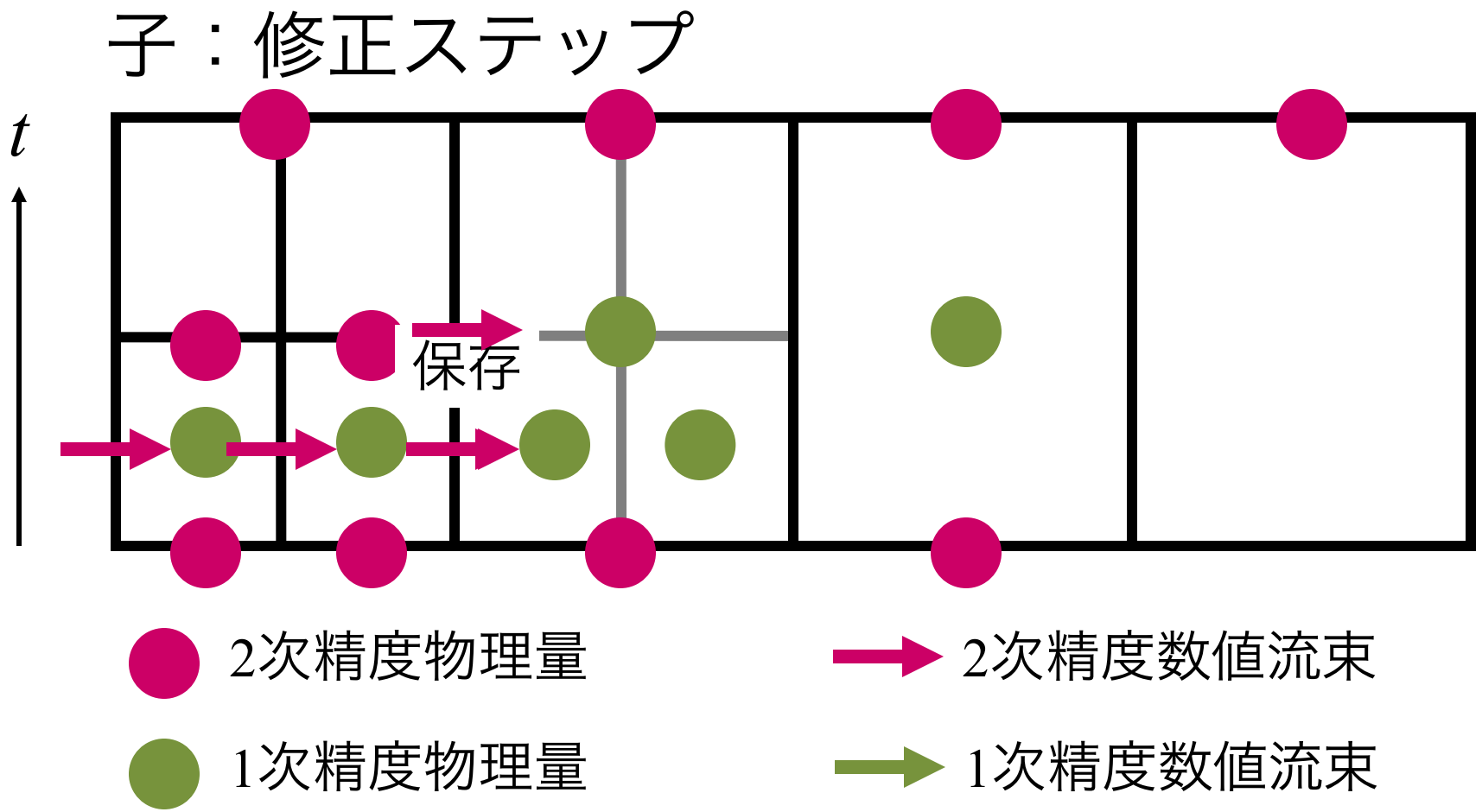
# コードの詳細：　流体部分
# 数値流束補正・マルチタイムステップ



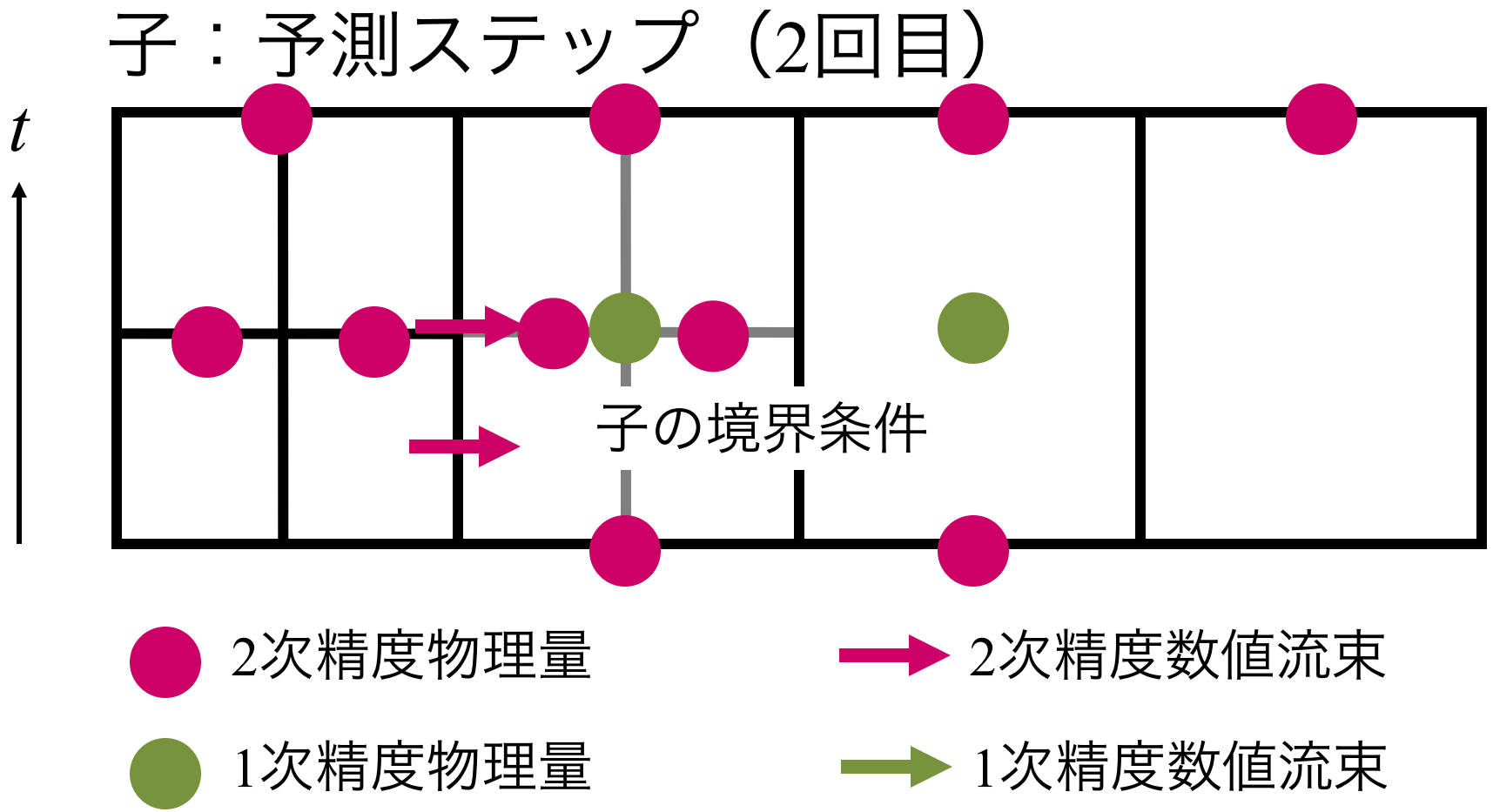
# コードの詳細：　流体部分
# 数値流束補正・マルチタイムステップ



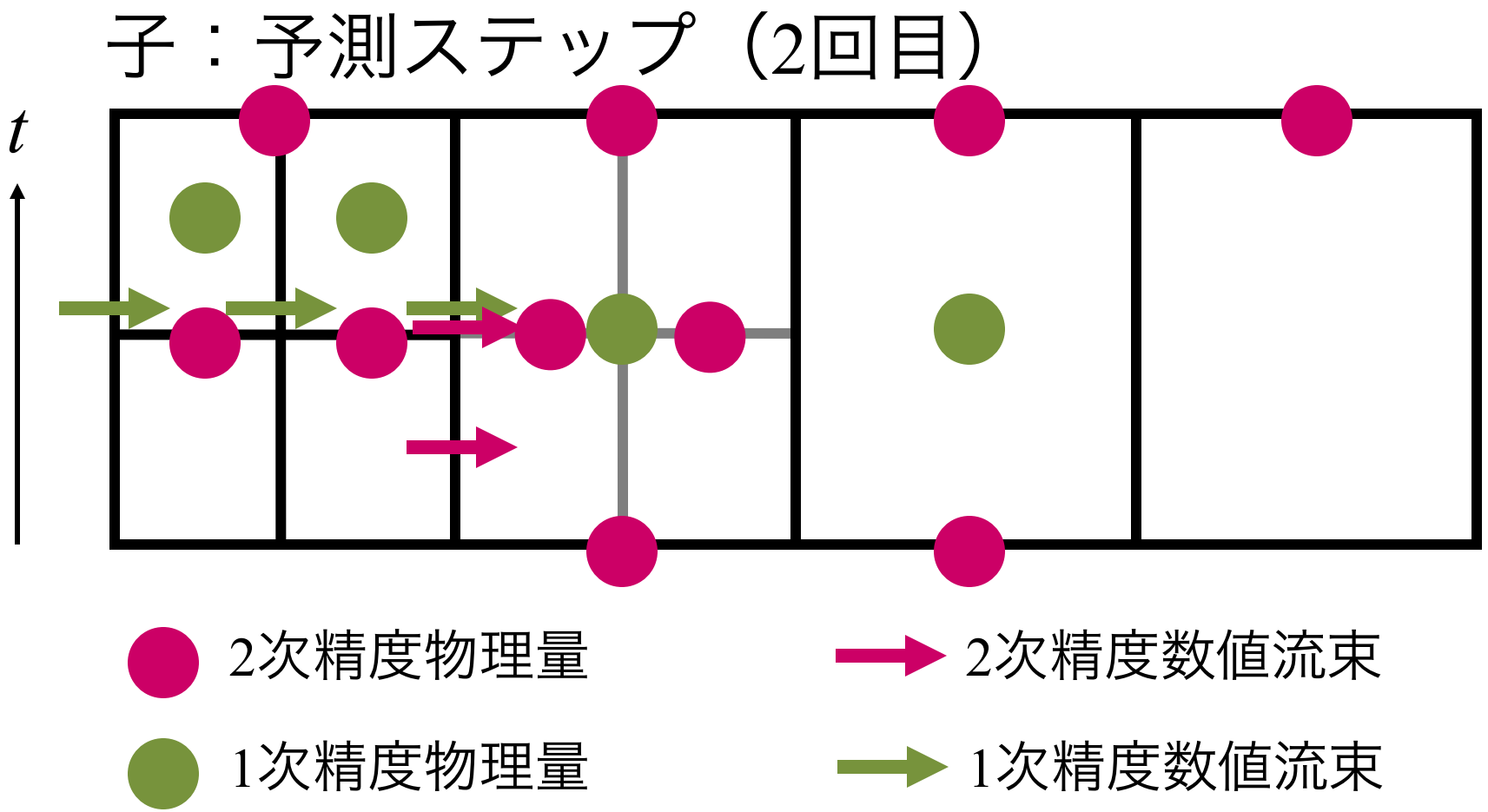
# コードの詳細：　流体部分
# 数値流束補正・マルチタイムステップ



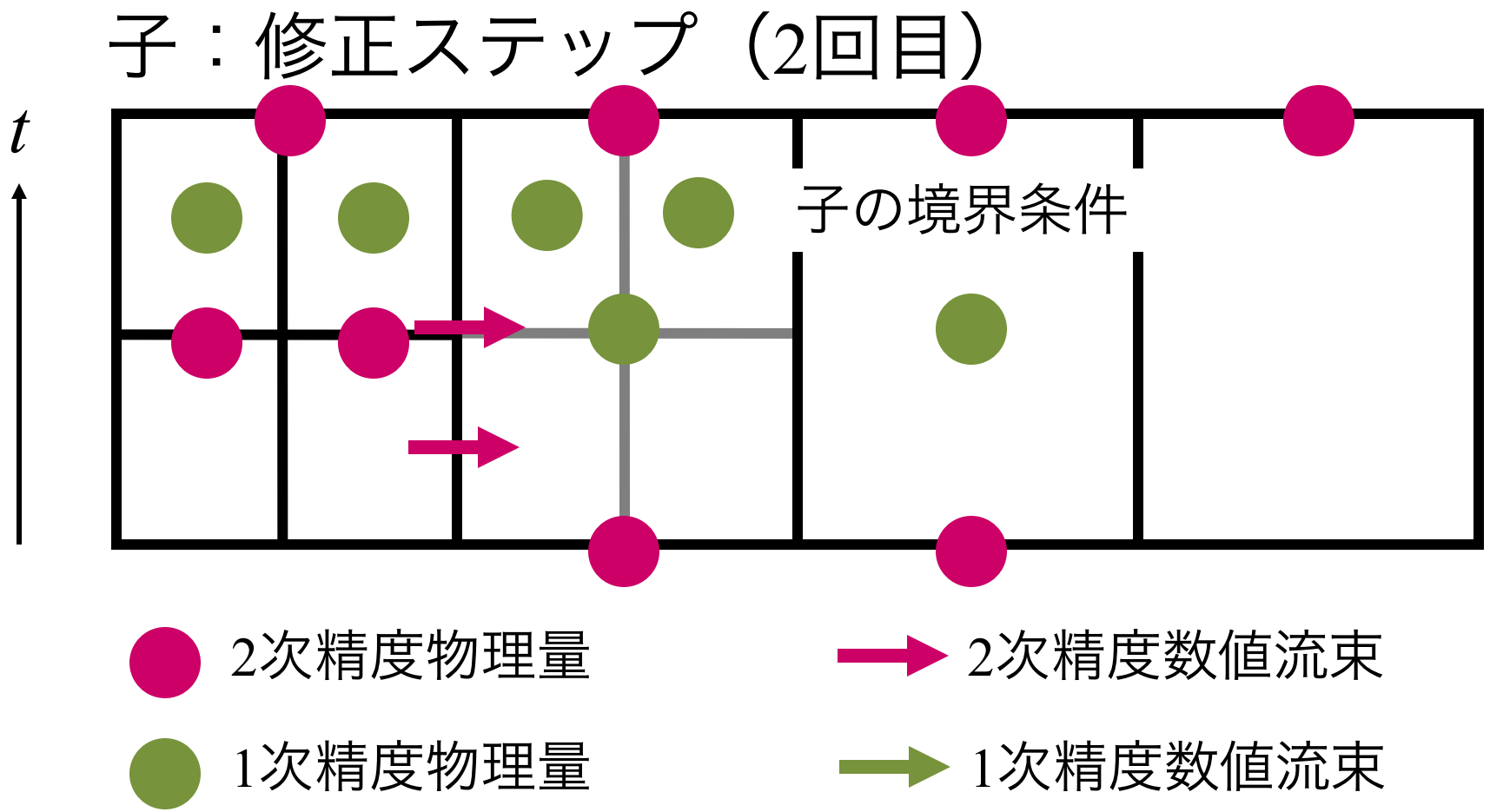
# コードの詳細：　流体部分
# 数値流束補正・マルチタイムステップ



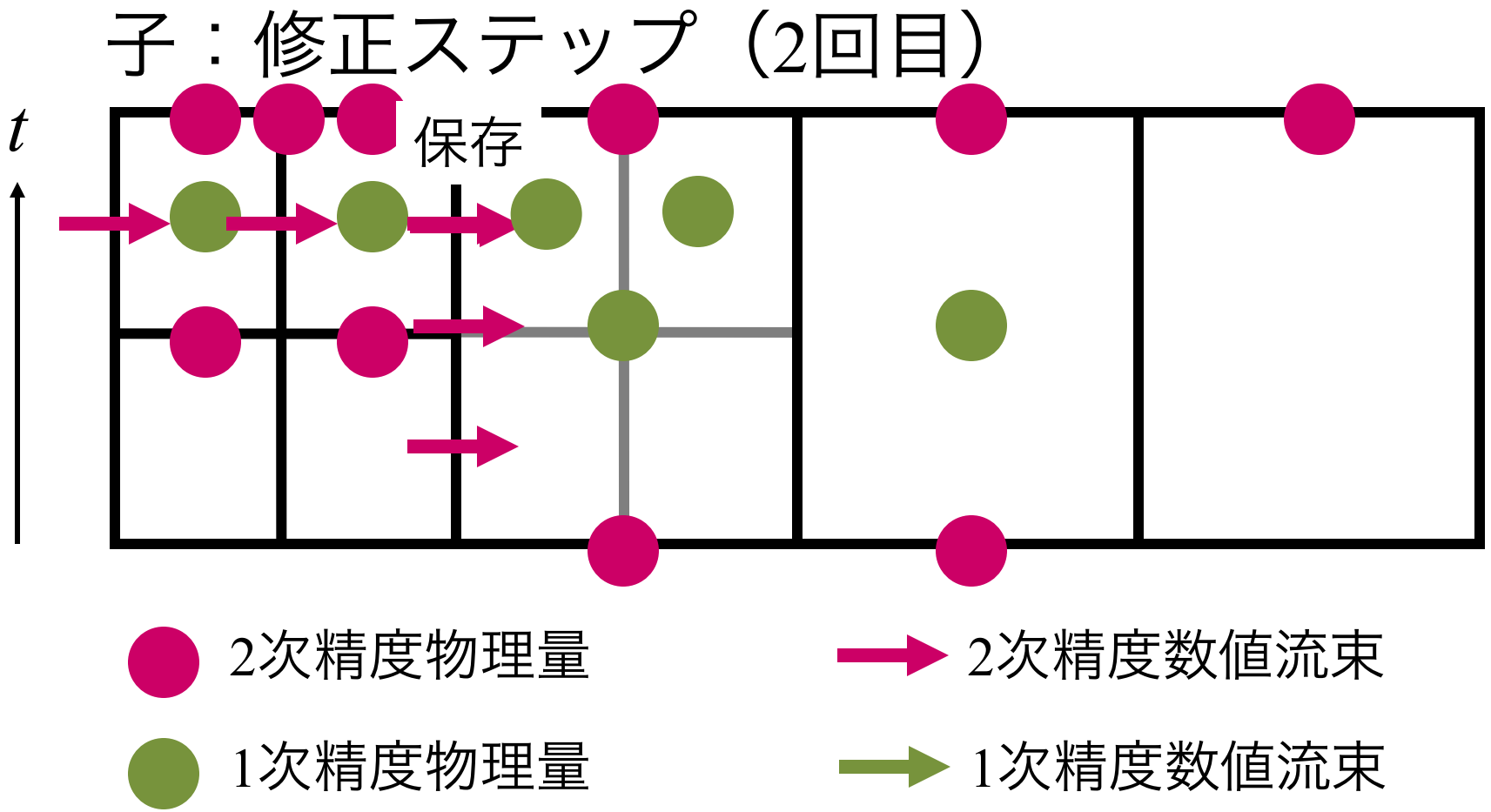
# コードの詳細：　流体部分
# 数値流束補正・マルチタイムステップ



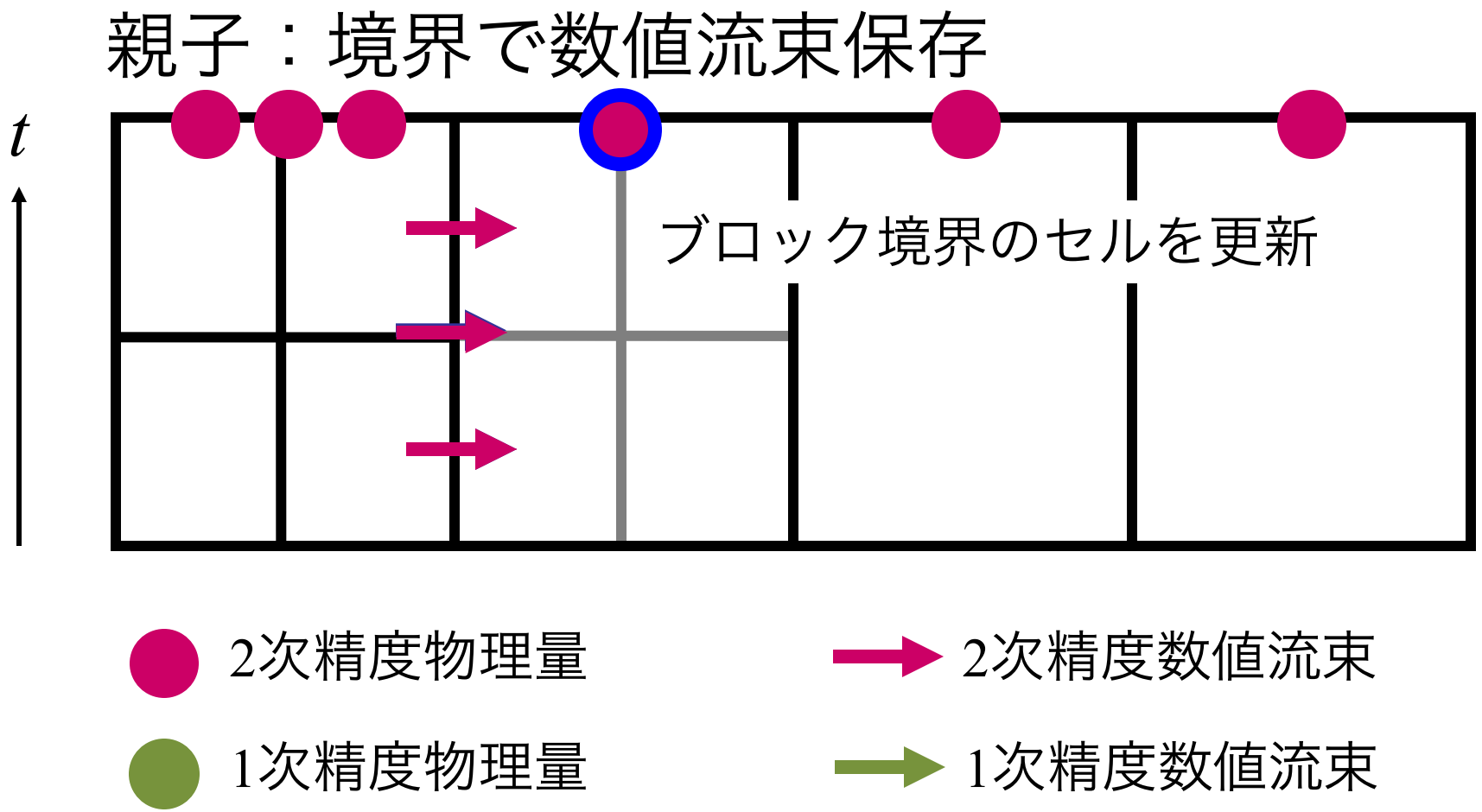
# コードの詳細：　流体部分
# 数値流束補正・マルチタイムステップ



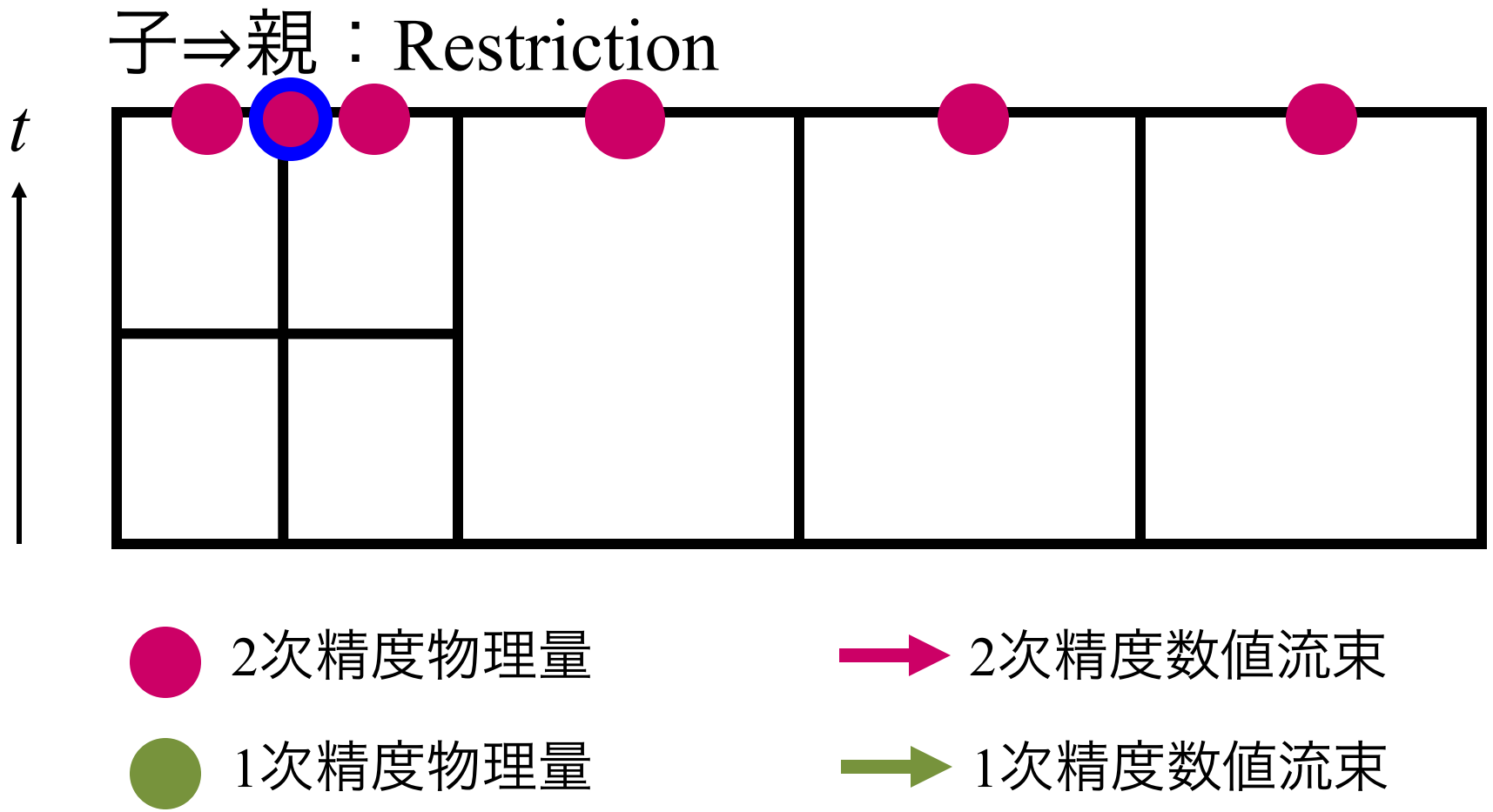
# コードの詳細：　流体部分
# 数値流束補正・マルチタイムステップ



# コードの詳細：　流体部分
# 数値流束補正・マルチタイムステップ



# コードの詳細：　流体部分
# 数値流束補正・マルチタイムステップ



# 細分化

# 細分化条件：　どこを細分化するべきか？

- 何を見たいかに依存する。
- 打切り誤差が閾値を超える部分を細分化 (Berger & Colella 1989)
- 物理量の2階微分が閾値を超える部分を細分化 (Flash など)
- 星形成分野の業界標準
  - Jeans 条件： Jeans 波長を4メッシュ以上で分解する (Truelove et al. )
  - 最近は8や16メッシュが多い。

# 細分化のアルゴリズム

1. 細分化するセルの探査
   a. レベル $l$ に属するセルを順に探査し、細分化条件を満たすセルに印をつける。
   b. 印がついたセルの周囲のセルにも印をつける。
   c. レベル $l+2$ にブロックと重なるセルにも印をつける。

2. ブロック配置の決定
   a. 印がついたセルを含むようにレベル $l+1$ のブロックを作る

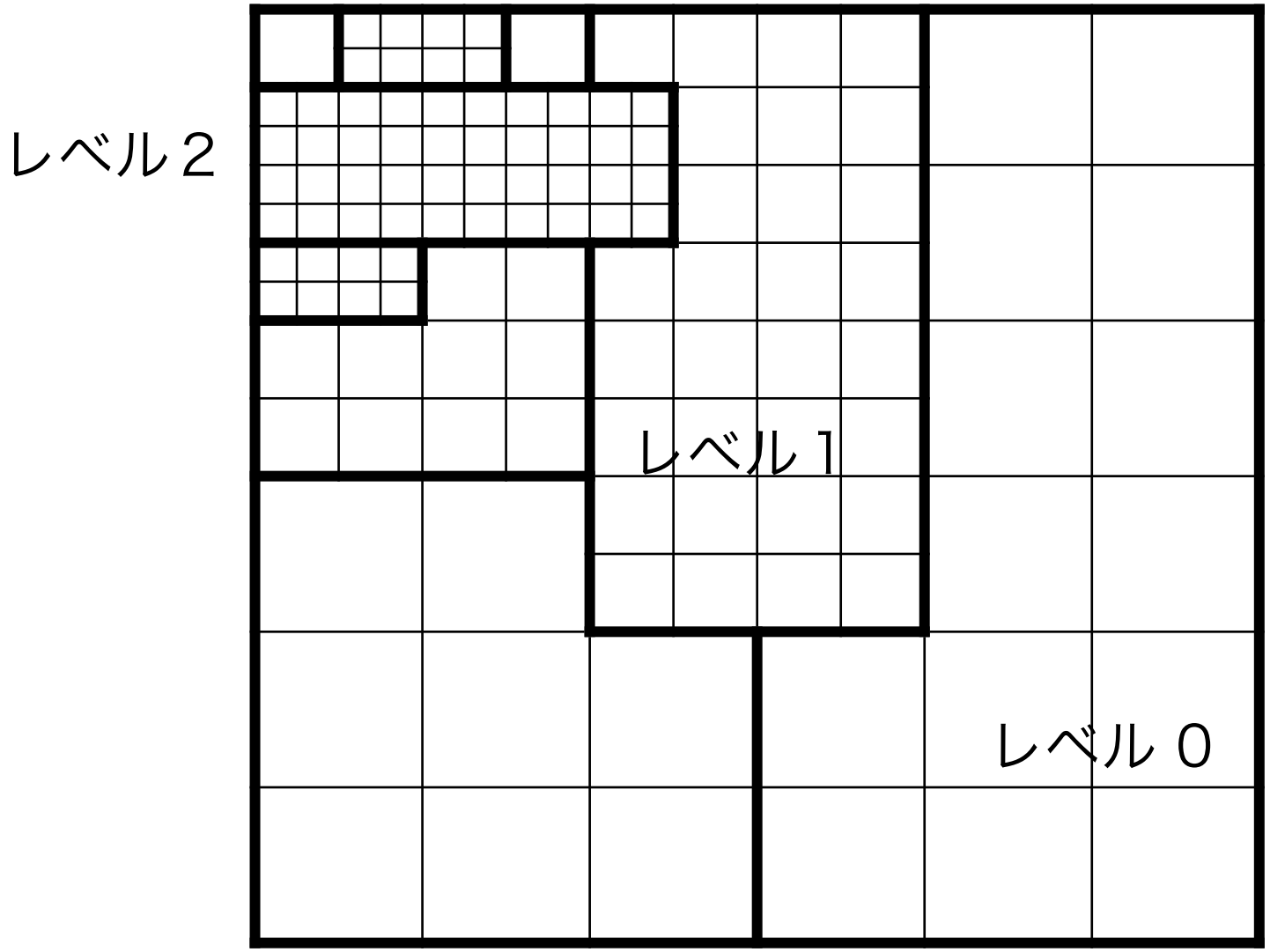
3. ブロックの生成
   a. 当該セルがすでに細分化されていれば、その値をコピーする。
   b. 新たに細分化する場合には、レベル $l$ の値を内挿する。

4. 古いブロックの破棄

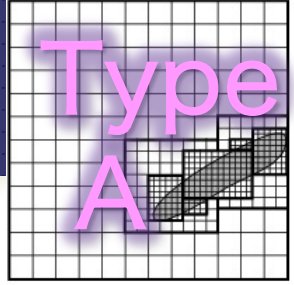
5. 手順1～4を同期しているレベルで行う。
   a. 粗いレベルから細かいレベルの順に繰り返す。

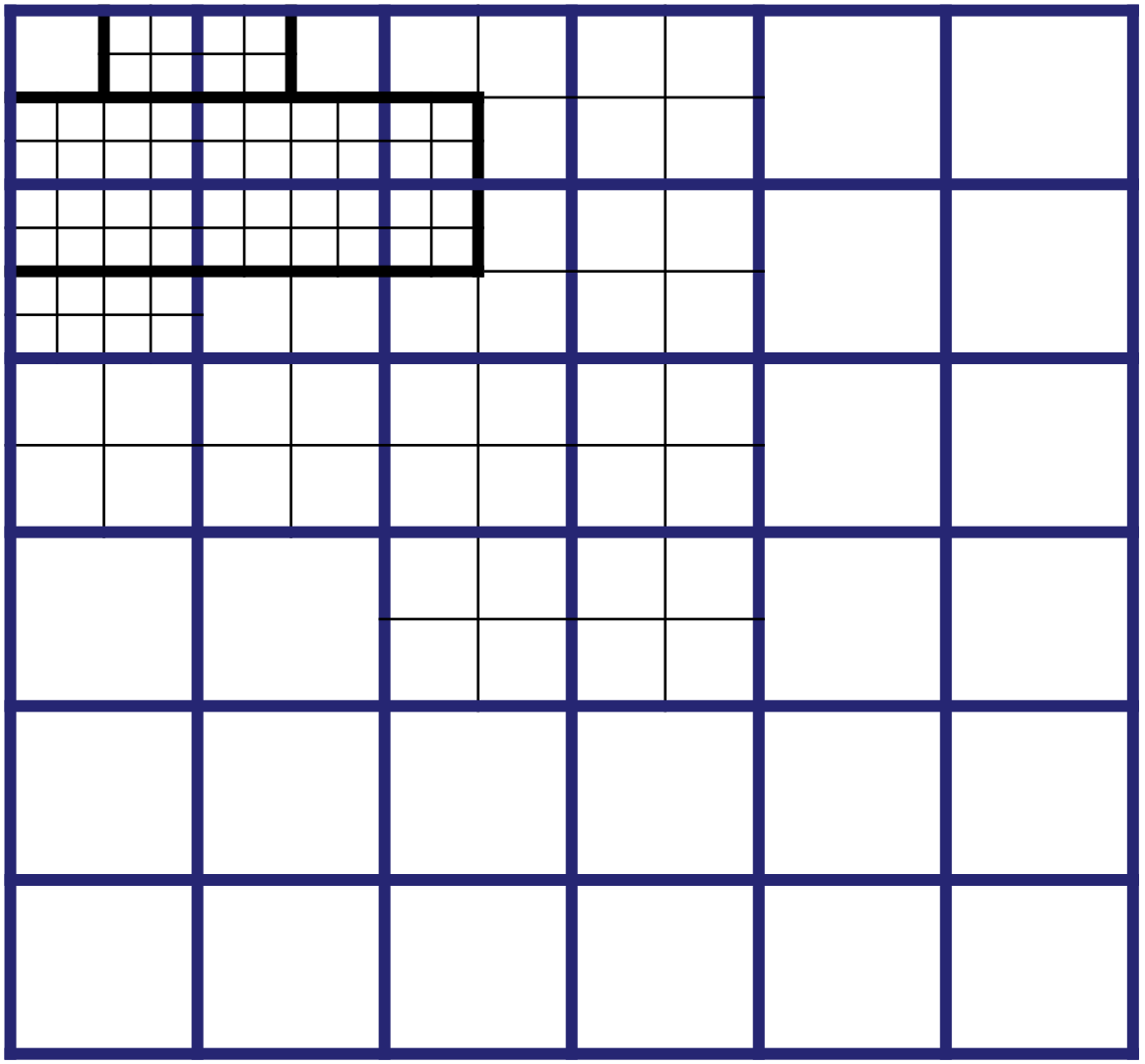
以下、Type A の例を示すが、Type B でもブロックの決め方以外は同じ。

# 細分化のアルゴリズム
# レベル０を細分化してレベル1を生成

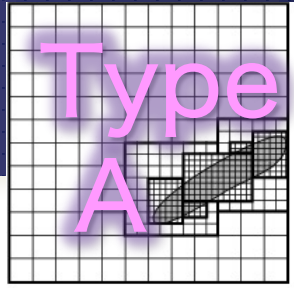


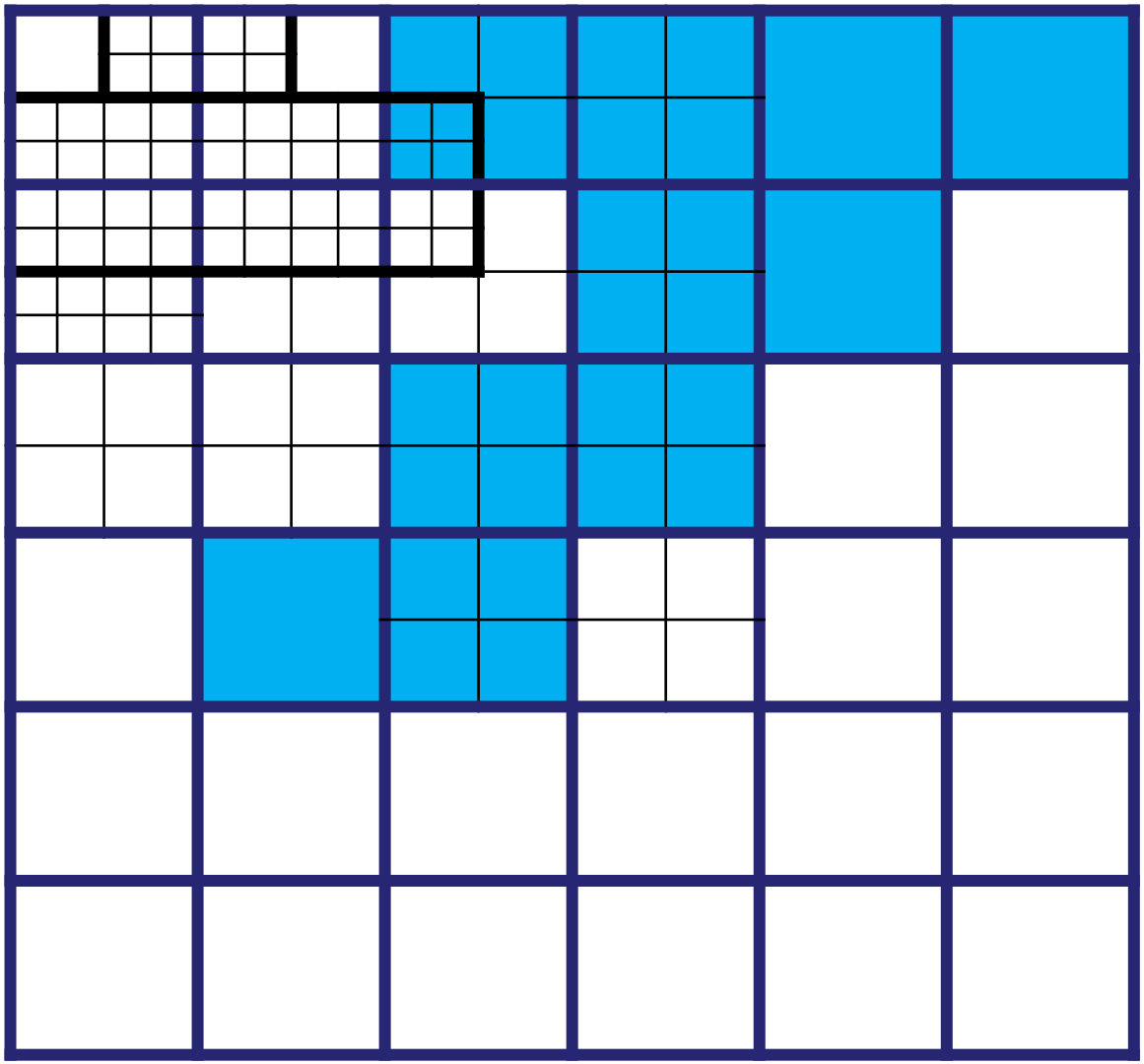
# 1. レベル0のセルに注目する。



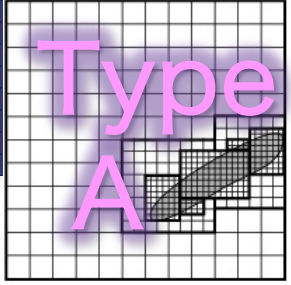



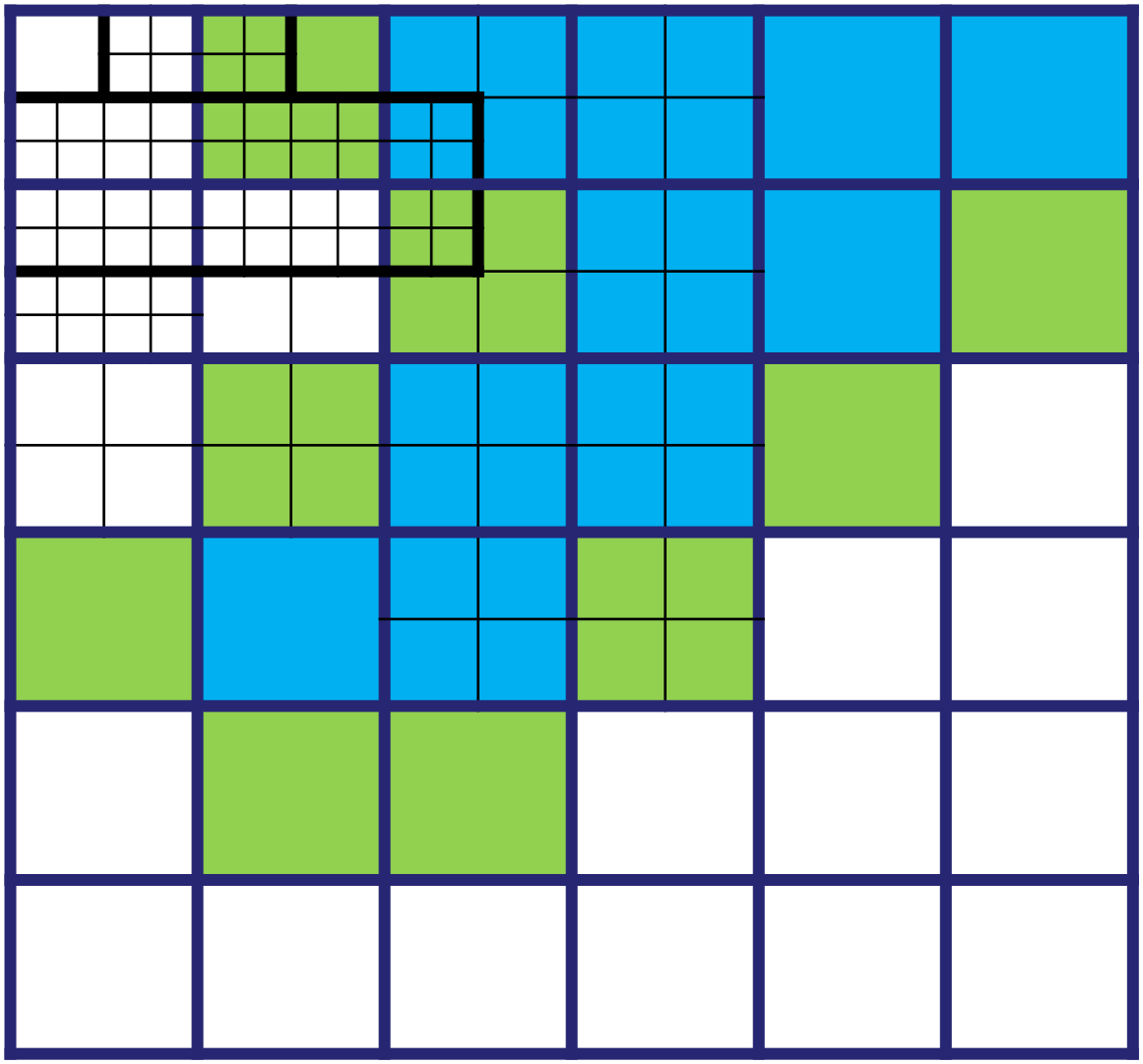
# 1-a. レベル0の細分化。
# レベル0のセルに対して細分化条件を評価する。



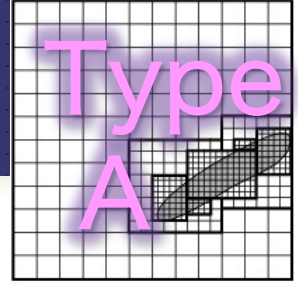



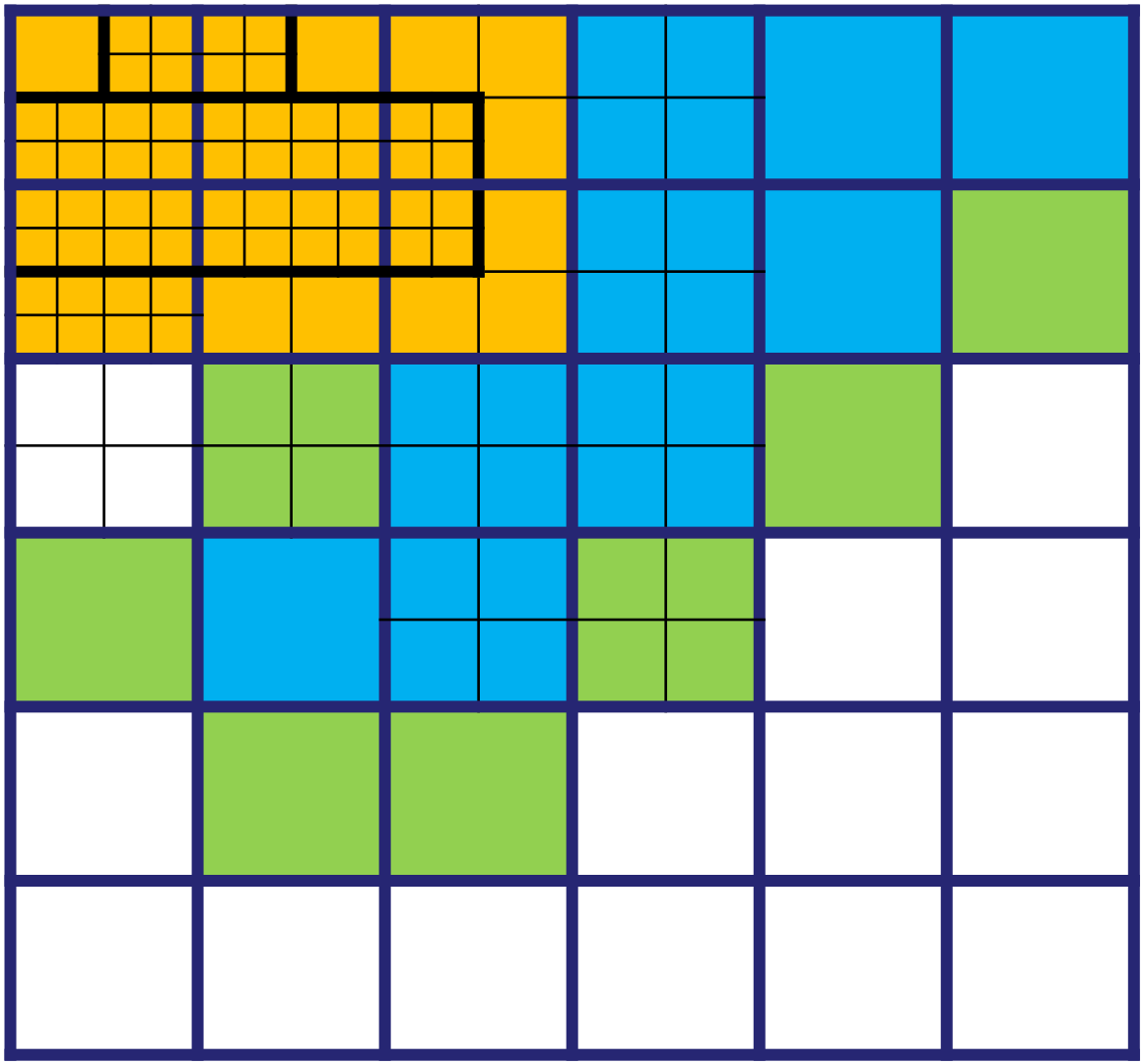
# 1-b. レベル0の細分化。
# 周囲の１～２セルにも印をつける。マージンの確保。



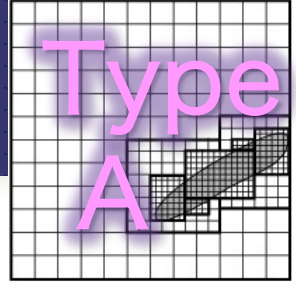



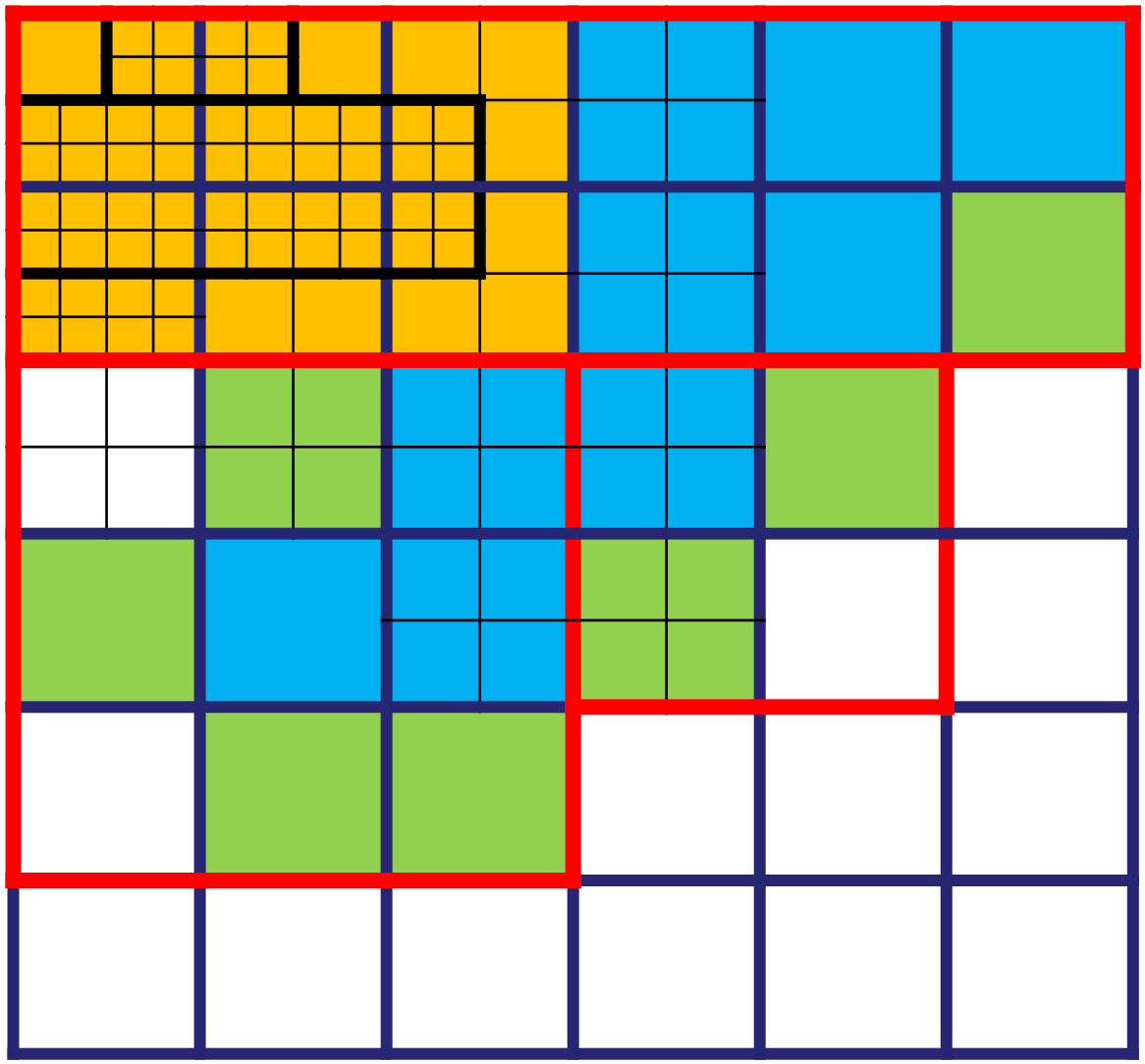
# 1-c. レベル0の細分化。
# レベル2と重なるセルにも印をつける。



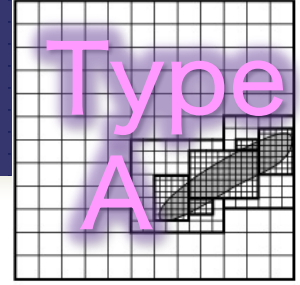



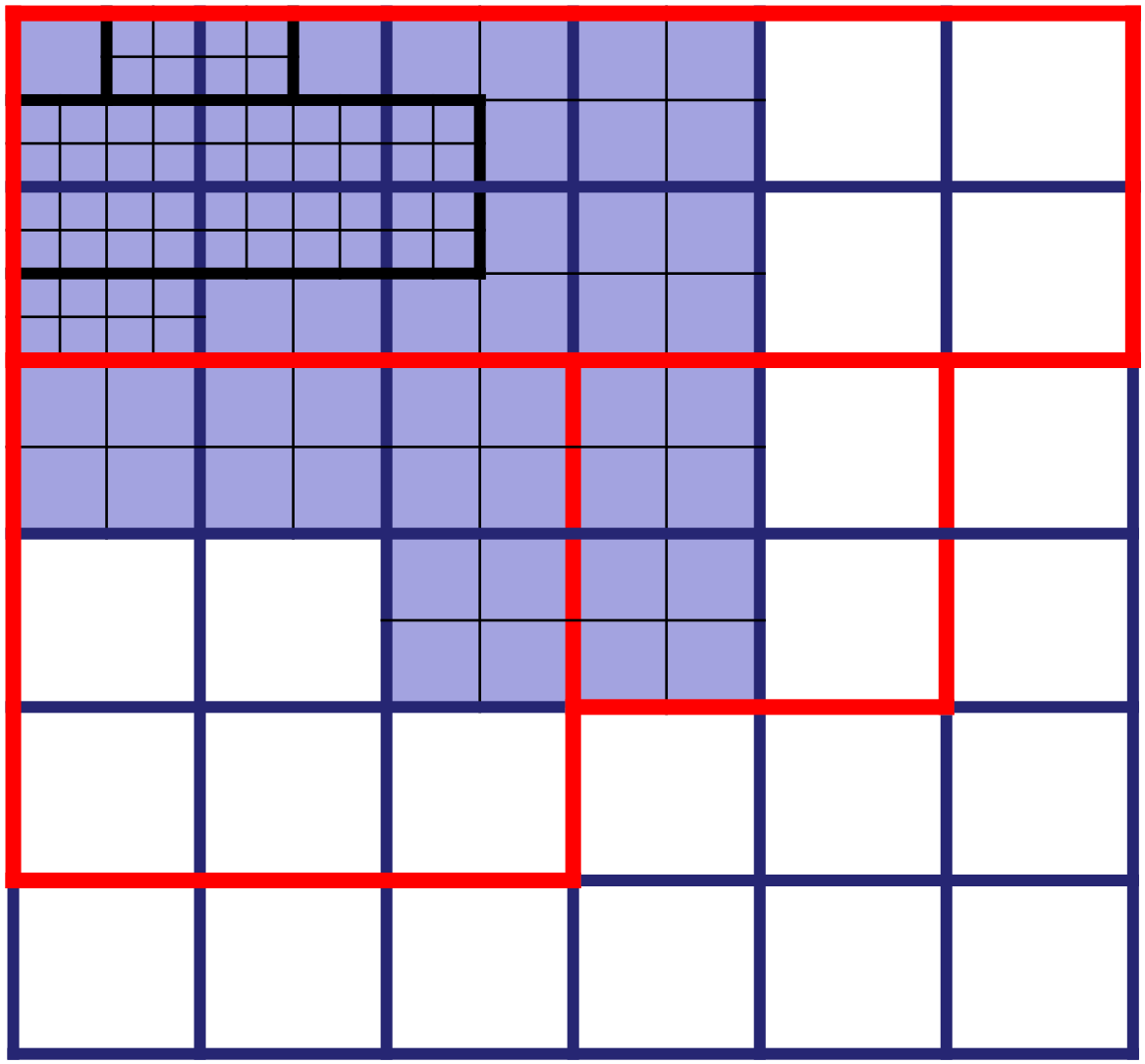
# 2. レベル１のブロック配置を決定。
# ここはいろいろな流儀がある。



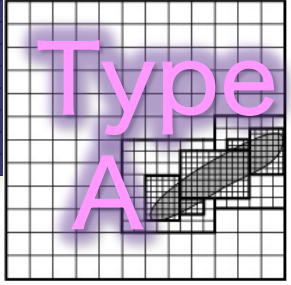



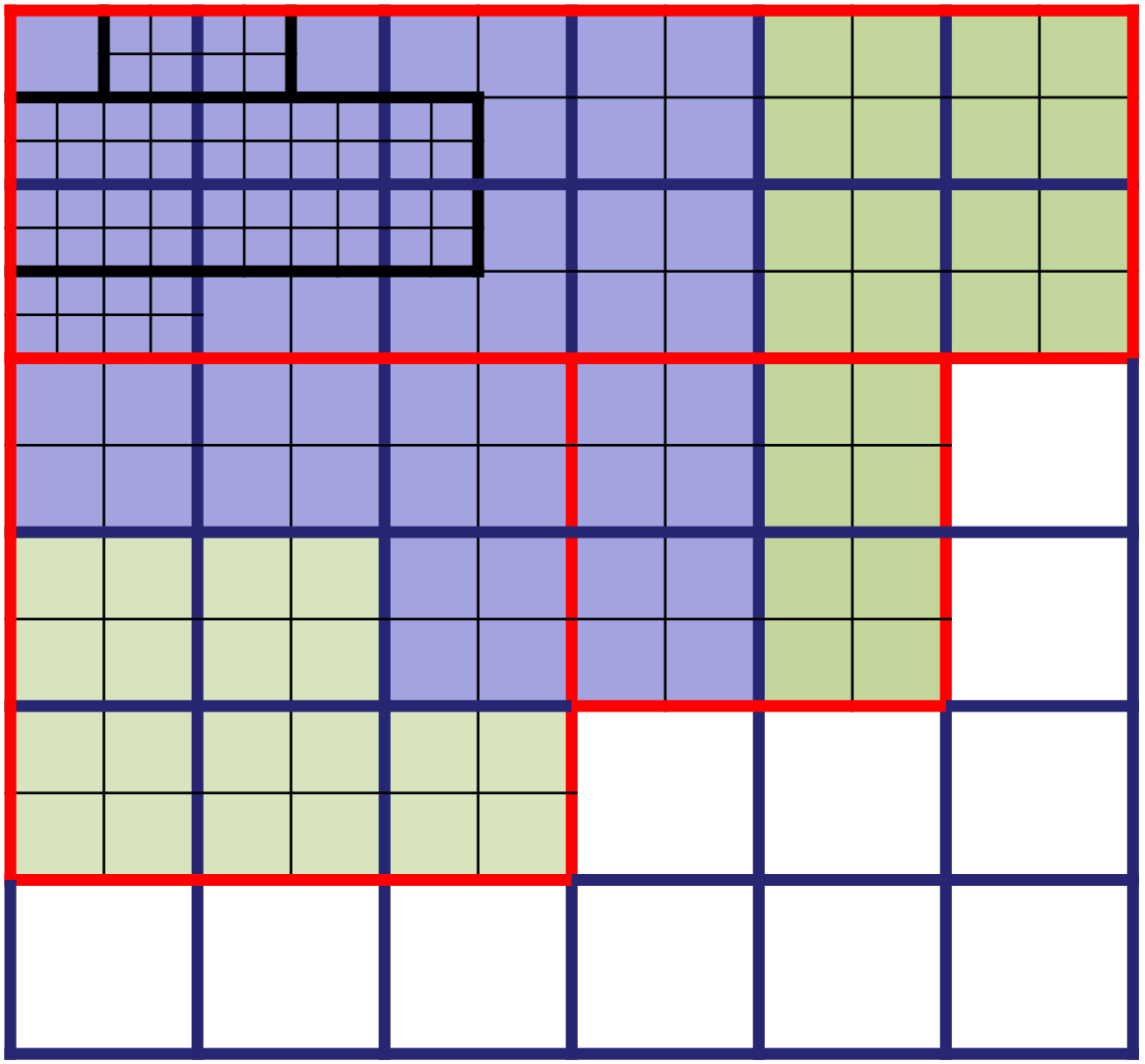
# 3-a. ブロックの生成。
# 既存のセルの値をコピー。



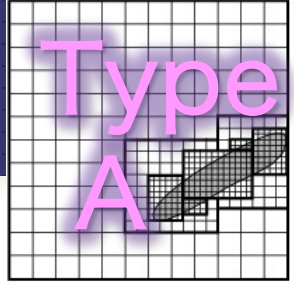



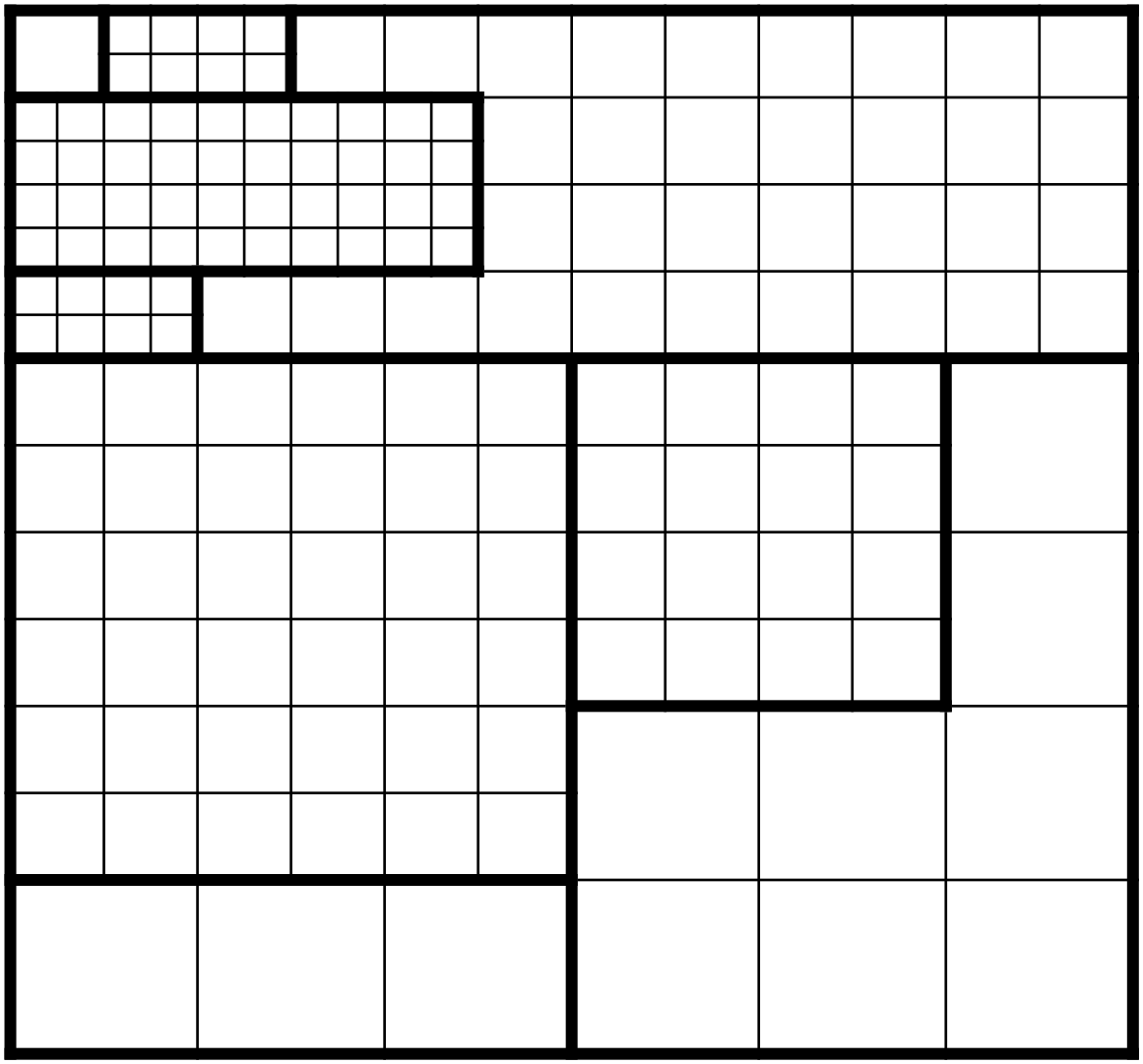
# 3-b. ブロックの生成。
# レベル０から内挿。



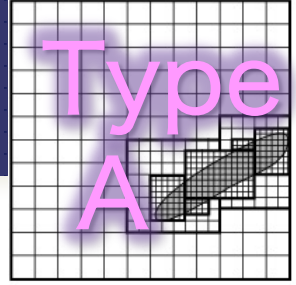



# 4. 古いブロックを破棄して完成。

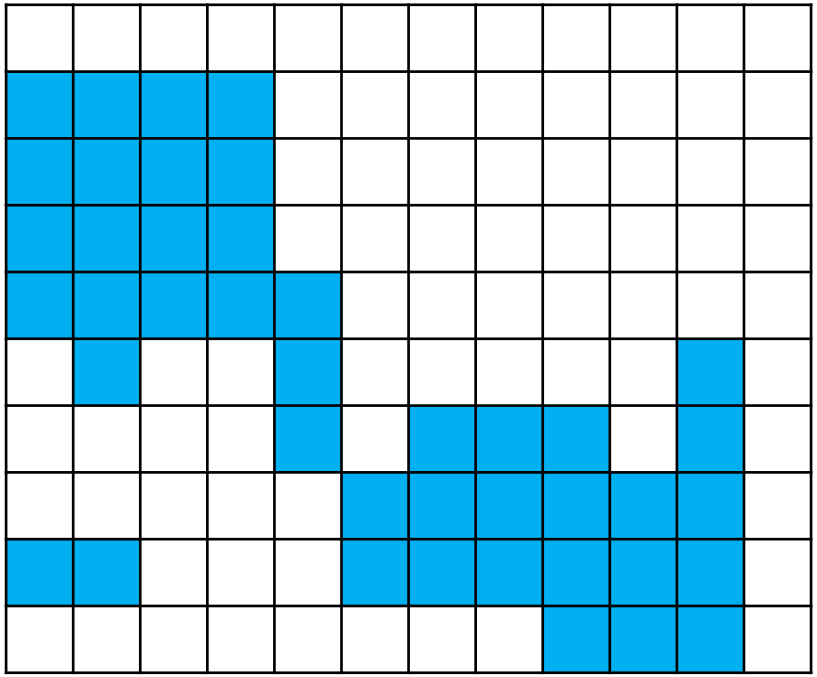


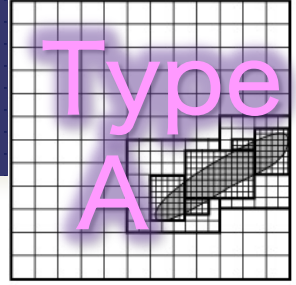
# ブロックの決定方法



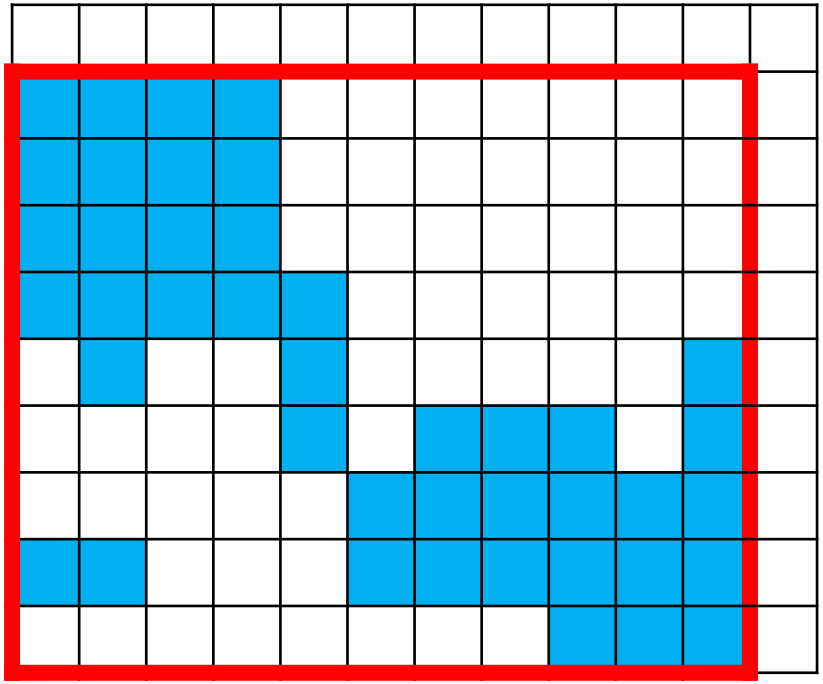

パッチ指向AMRには様々な流儀が存在する。
たとえば、



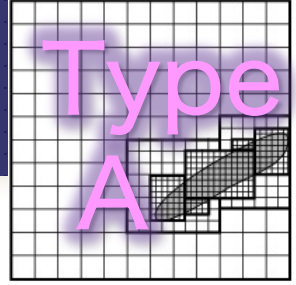
# ブロックの決定方法



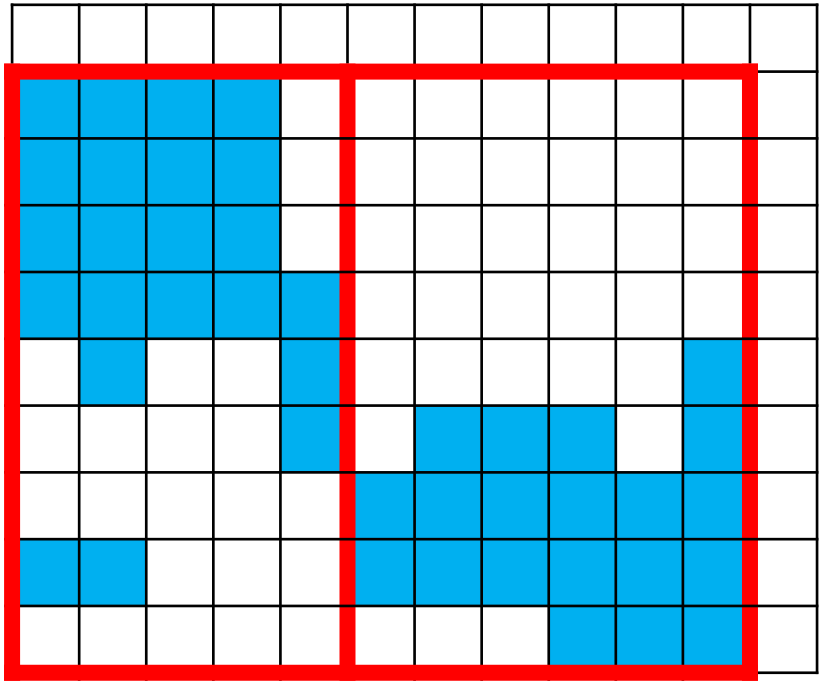

印がついたセルをすべて覆うブロックを考える



# ブロックの決定方法



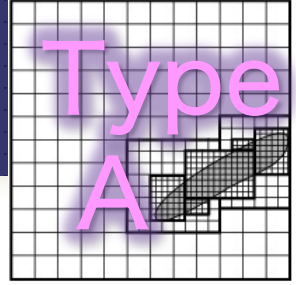

セル数のヒストグラムを書き、最小値部分でブロックを分割する。



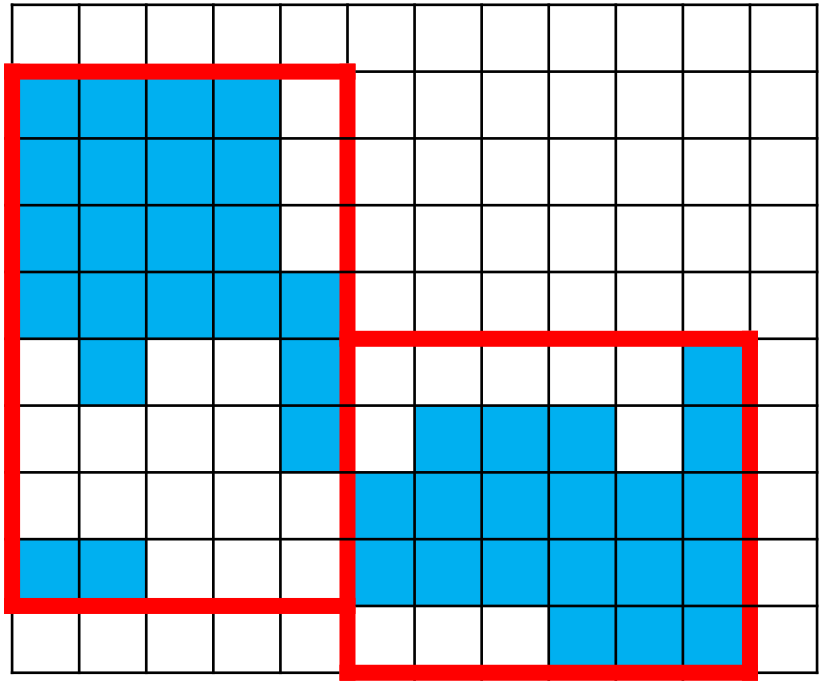
ヒストグラム

最小

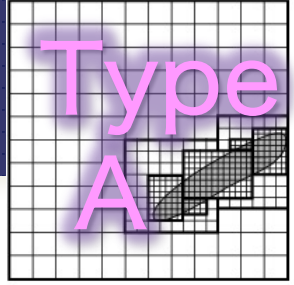
# ブロックの決定方法



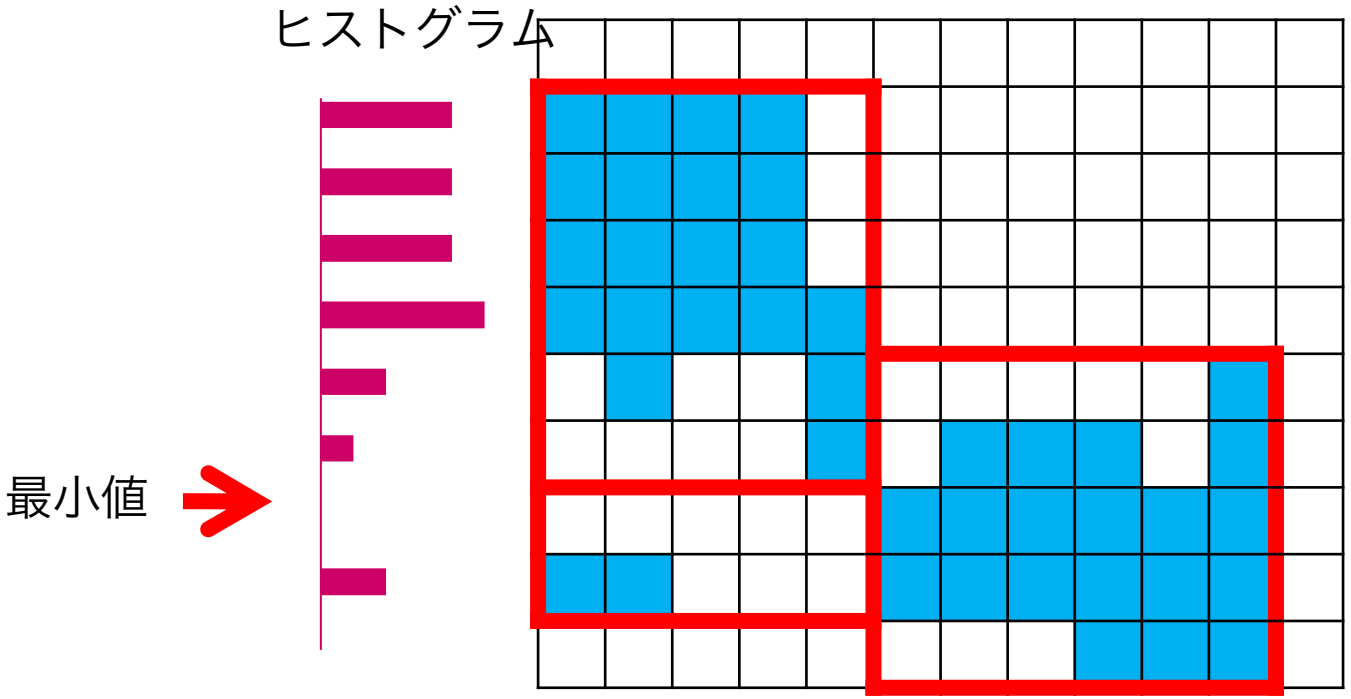

余分な部分を取り除く



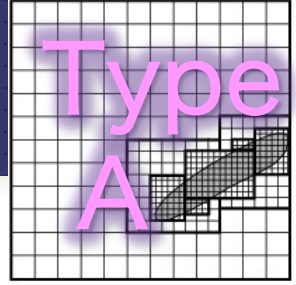
# ブロックの決定方法



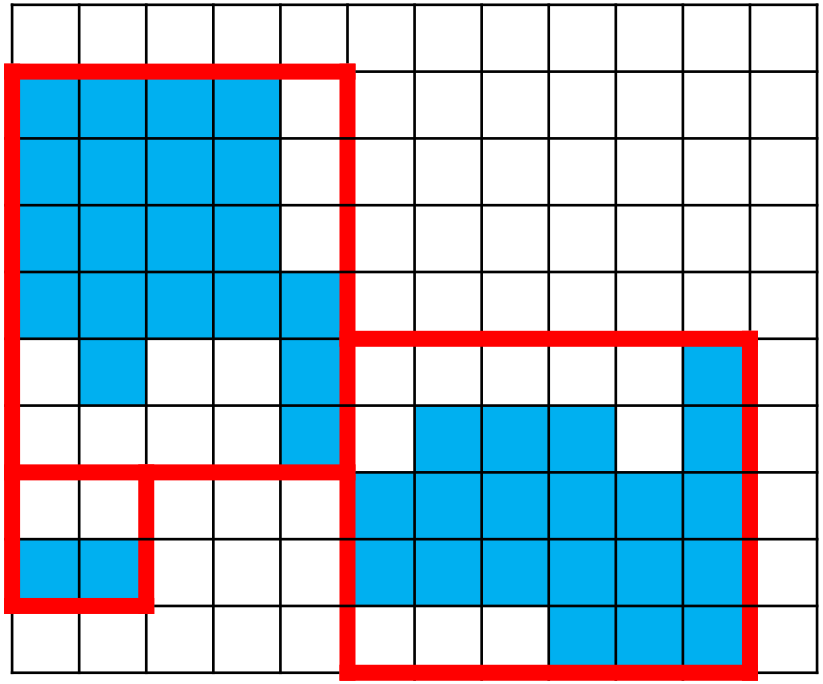

別の方向からセル数のヒストグラムを書き、分割する。



# ブロックの決定方法



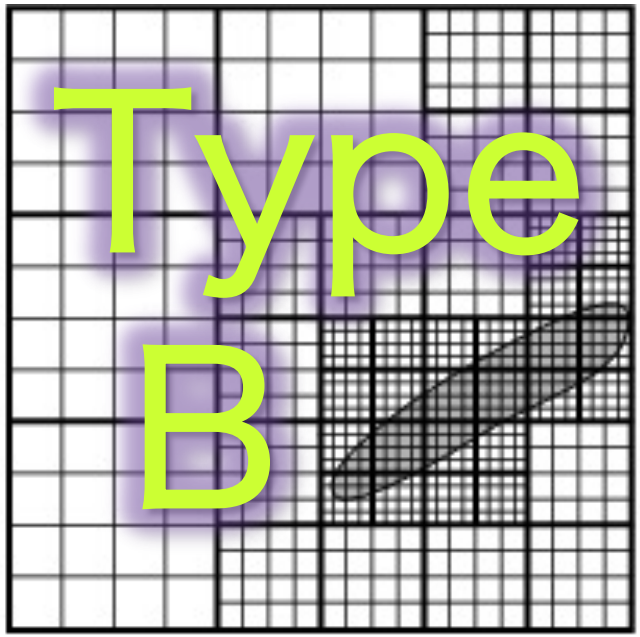

余白を取り除く。



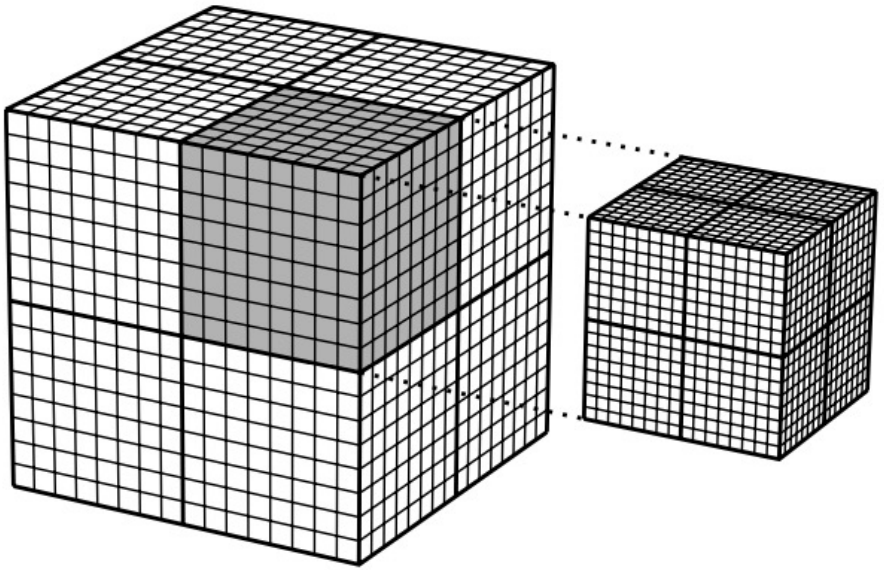
こんなことを繰り返してゆく。

この操作を繰り返すほど、ブロックは小さくなる。
ブロックの最小サイズを決めておこう。

# とっても面倒です




を採用しよう！



Matsumoto 2007
doi:10.1093/pasj/59.5.905

# 自己重力
# マルチグリッド法

# マルチグリッド法

再掲

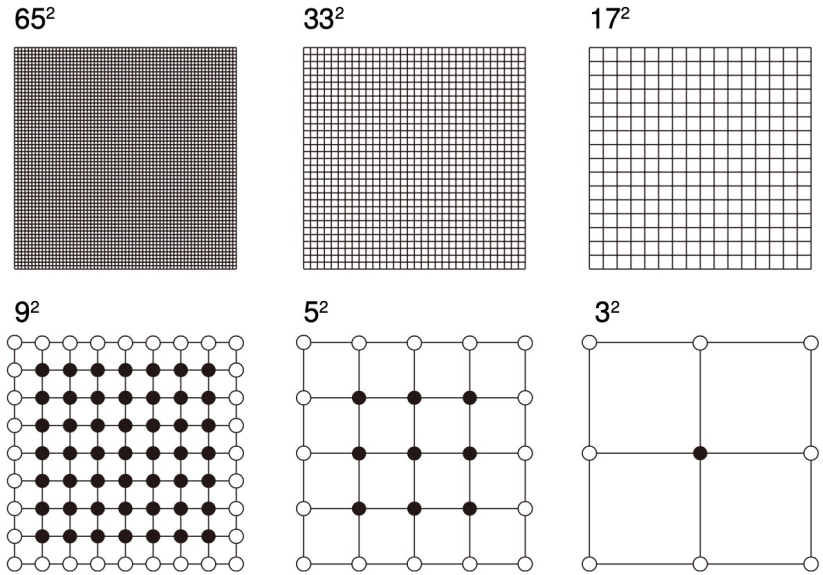



図 **4.1** マルチグリッド法における階層的な格子。2 次元の例を示す。$65^2$ のグリッドの重力ポテンシャルを求めるとき、$33^2 \cdots 3^2$ の作業グリッドを用意する。下段の図では、未知数を定義するグリッドポイントを黒丸で、境界条件を定義するグリッドポイントを白丸で示す。

様々な解像度の格子を準備
様々な波長の誤差を高速に収束させる

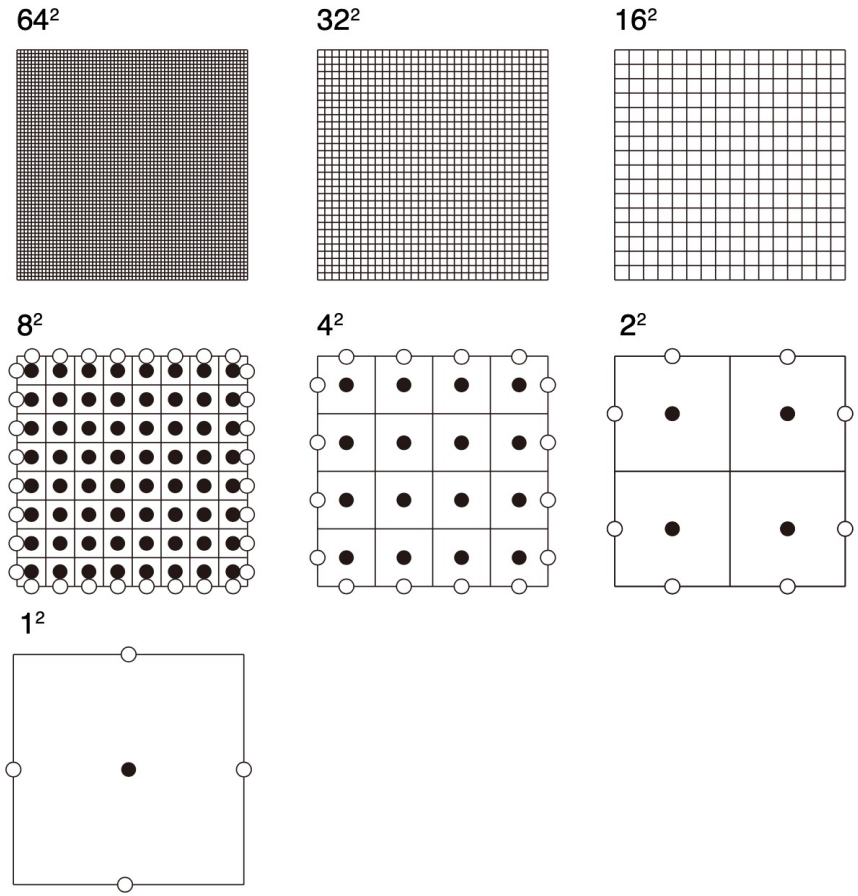



図 **4.2** 有限体積法に適したマルチグリッド法における階層的な格子。2 次元の例を示す。$64^2$ のグリッドの重力ポテンシャルを求めるとき、$32^2 \cdots 1^2$ の作業グリッドを用意する。中断と下段の図では、未知数を定義するセル中心を黒丸で、境界条件を定義するセル境界を白丸で示す。

# 一様格子におけるマルチグリッド法

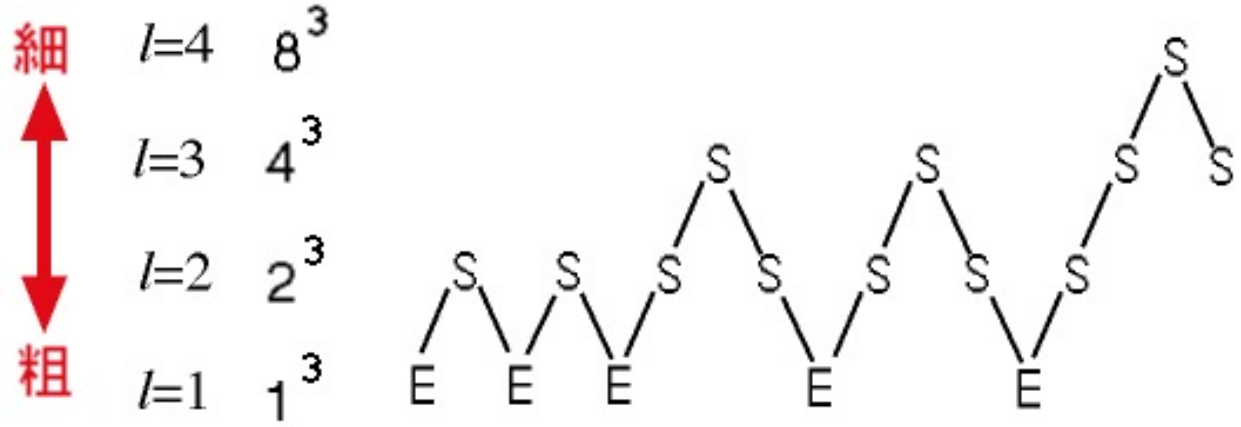
再掲

S = Smoothing
GS法で収束

E = 厳密解を求める

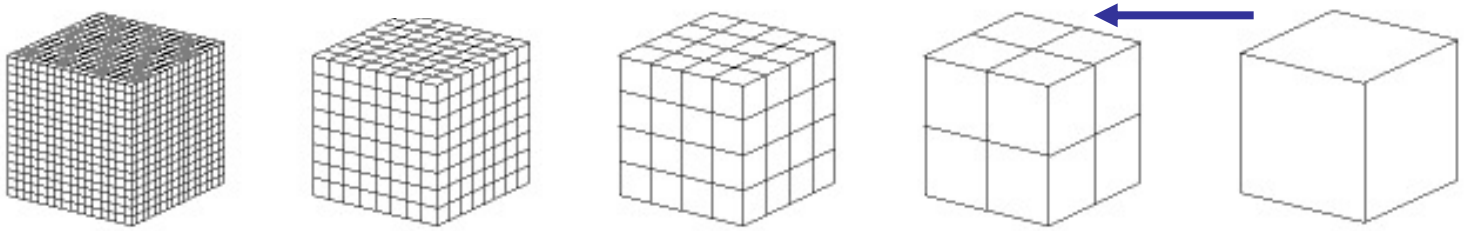
／ = Inerpolation
内挿

＼ = Restriction
間引き




Multigrid Iteration

内装: 良い初期値




実格子 ← → 作業格子

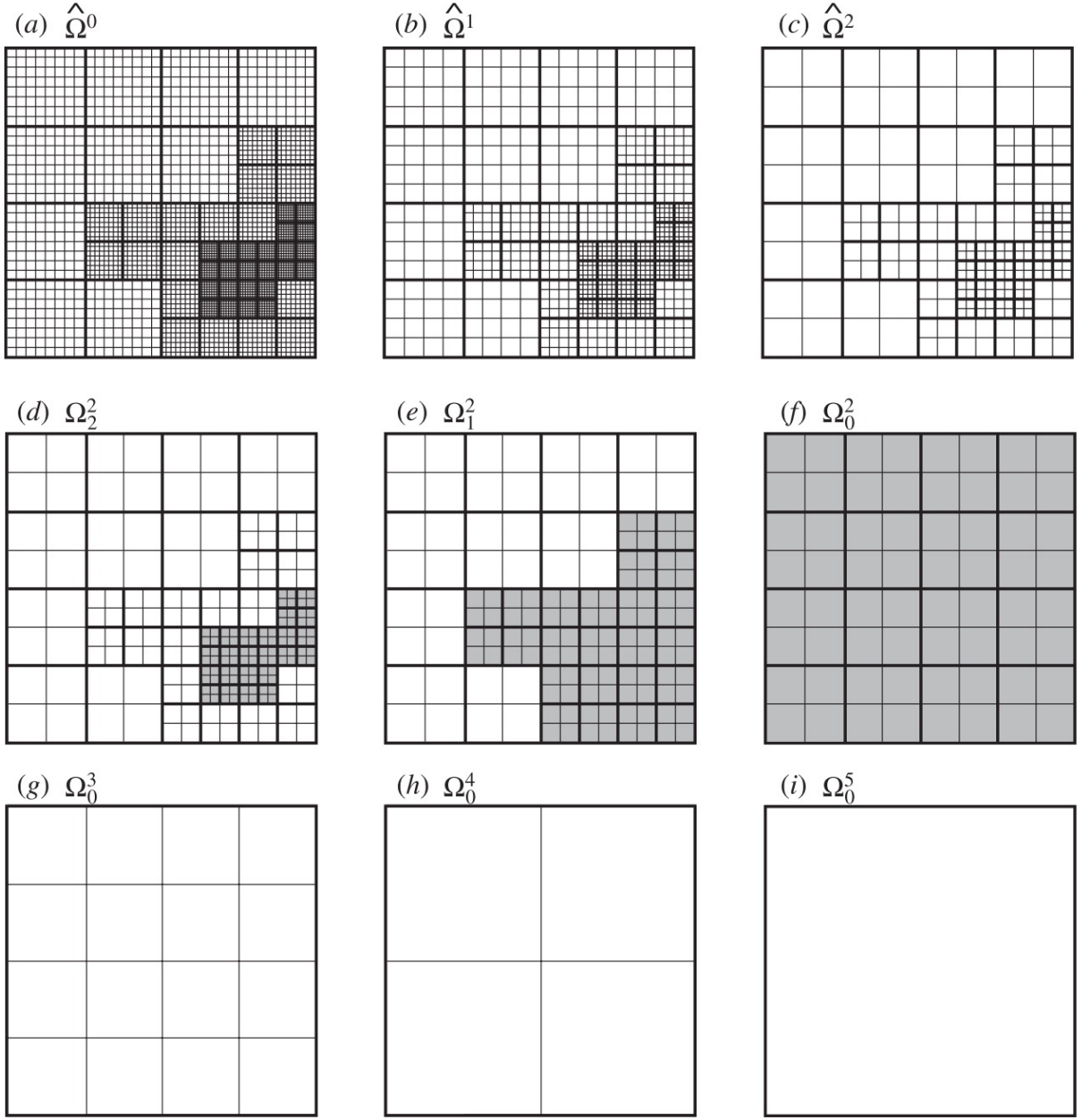
演算回数 ∝ 実格子のセル数

# AMRの場合

1. ブロックごとに粗くする

2. 階層を取り除く

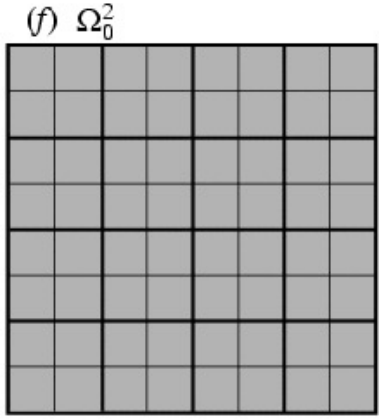
3. ベースの格子を粗くする



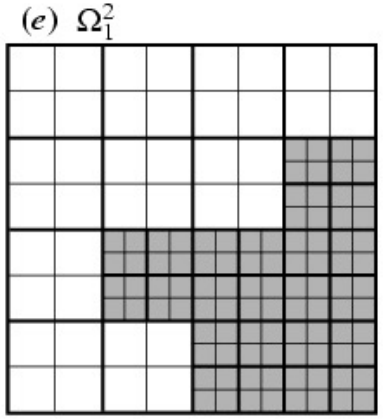

Matsumoto 2007
doi:10.1093/pasj/59.5.905

**Fig. 5.** Schematic diagram of the coarsening of grids in the multigrid method. The cell-boundaries and block-boundaries are denoted by thin and thick lines, respectively. (a–c) Coarsening of grids in the FMG-cycle on the AMR hierarchy. The number of the cells per block decreases up to $2^3$. (d–f) V-cycle on the AMR hierarchy. The solution converges sequentially on the hatched blocks. (g–i) Coarsening of grids in the FMG-cycle on the uniform base grid.

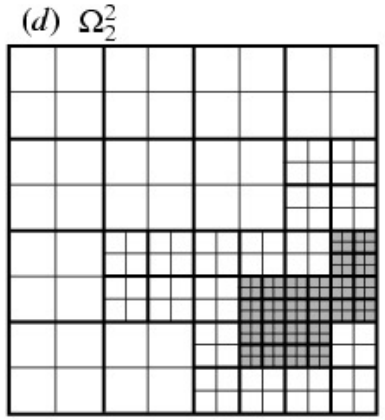
# AMRへの応用：すぐに思いつくがダメな例




1．レベル０の解を求める



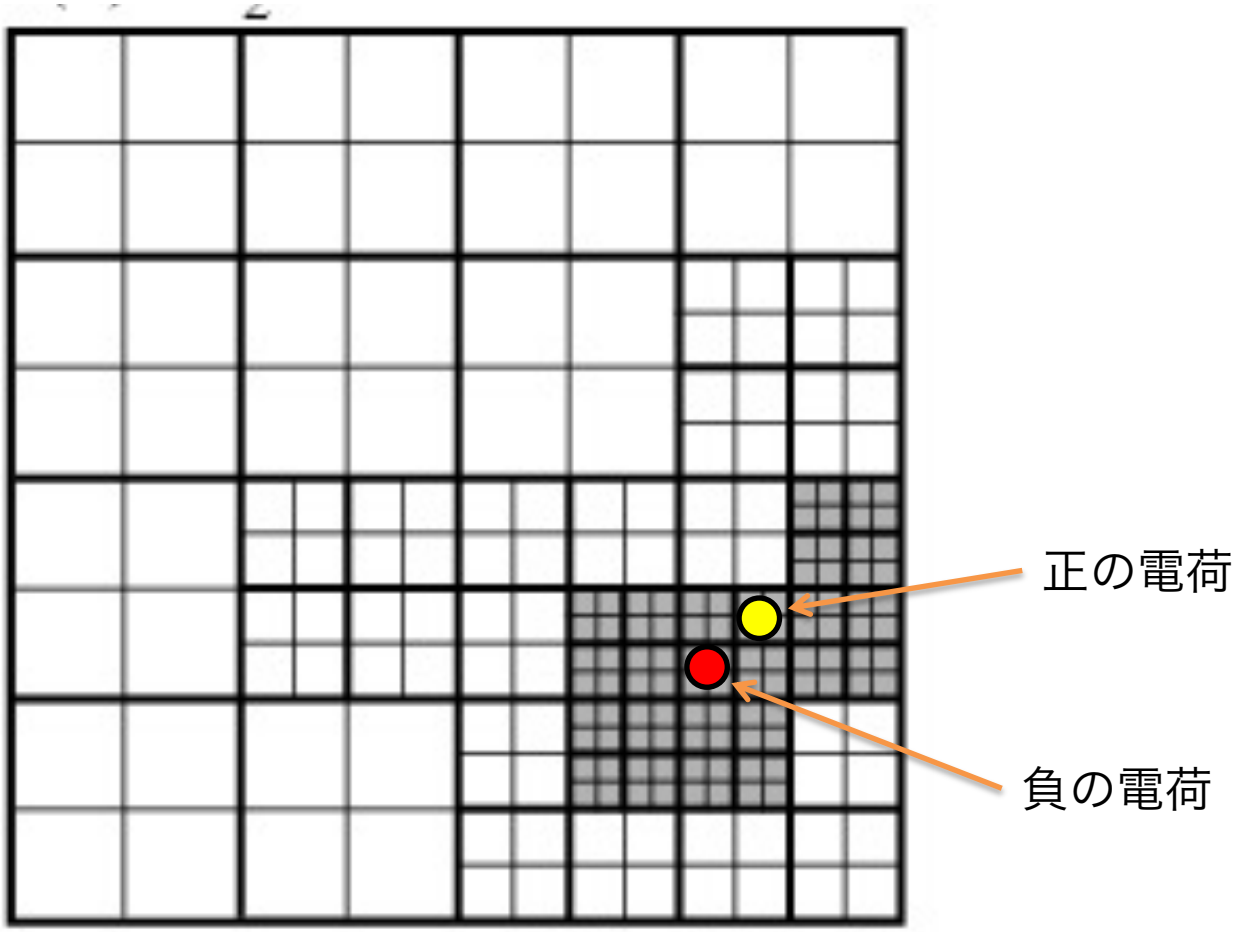

2．レベル０を境界条件にして
レベル１の解を求める



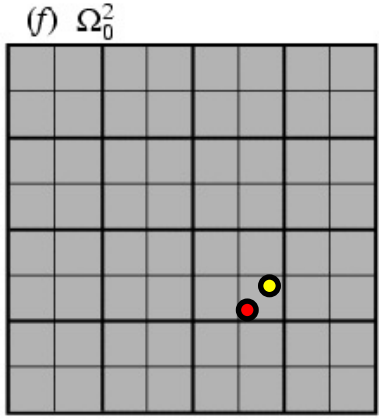

3．レベル１を境界条件にして
レベル２の解を求める

なぜダメか？

# 正負の電荷が作る静電場を求めなさい。（自己重力と同じPoisson 方程式の解）



# AMRへの応用：すぐに思いつくがダメな回答

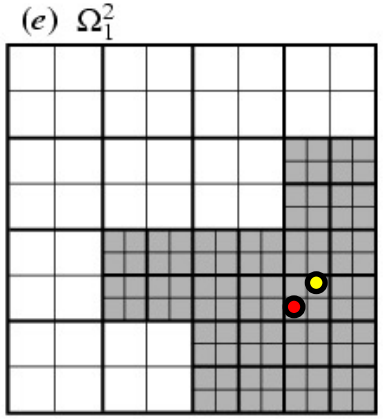



1．レベル０の解を求める

電荷がメッシュで分解できない。

電荷ゼロ。

電場ゼロ。

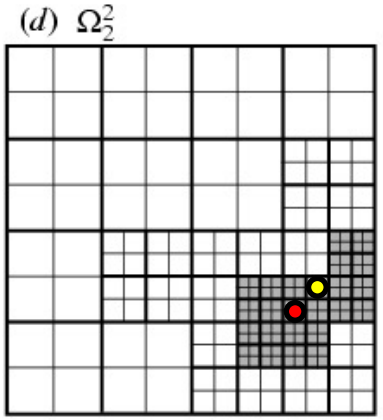



2．レベル０を境界条件にしてレベル１の解を求める

電荷が分解できた。

電場ゼロの解を境界条件に

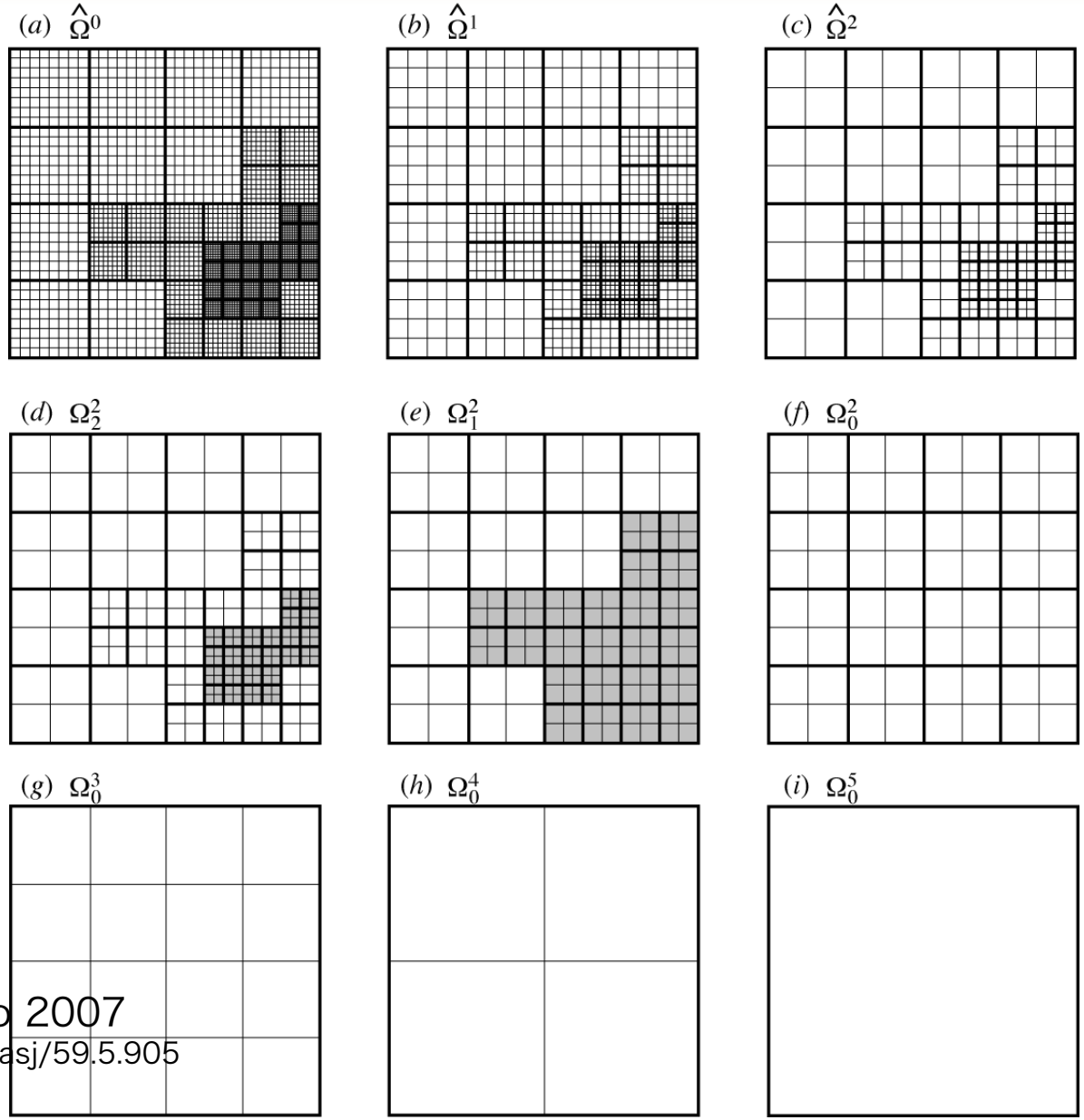
⇒　解が正しくない！




3．レベル１を境界条件にしてレベル２の解を求める

レベルを横断して解を求める必要

# グリッドを粗くする方法




合成グリッド：
AMRレベルを横断して粗くする。

MLAT：
AMRレベルごとに更新する。

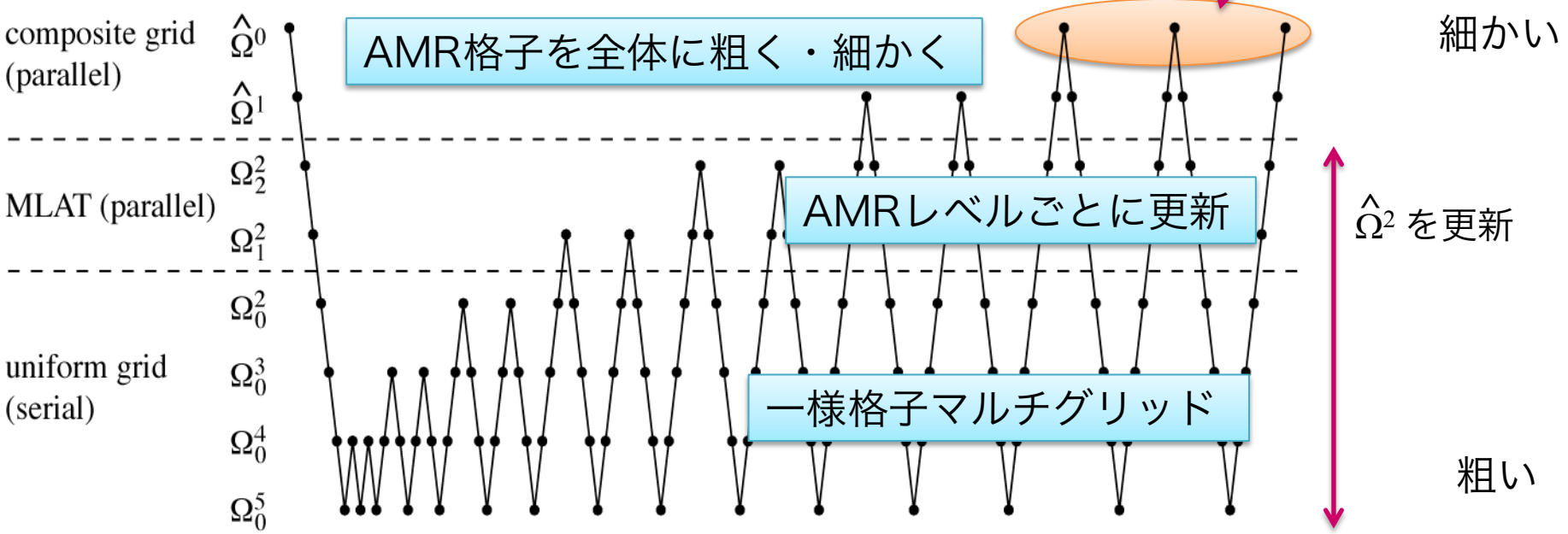
一様格子

Matsumoto 2007
doi:10.1093/pasj/59.5.905

# マルチグリッドサイクル

FMG がベストの選択
（いろいろ試した結果）

ほとんどの計算時間はここ




Ncycle=2の場合

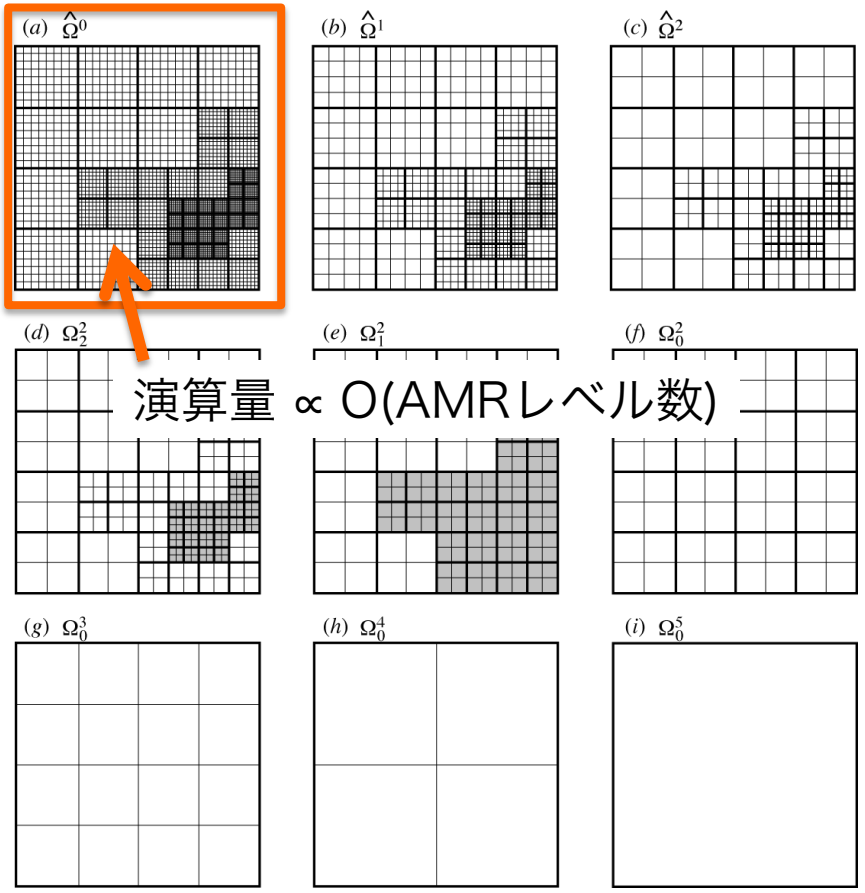
MLAT: Multilevel adaptive technique

Matsumoto 2007
doi:10.1093/pasj/59.5.905

# セルの間引き方の比較

## 合成グリッド

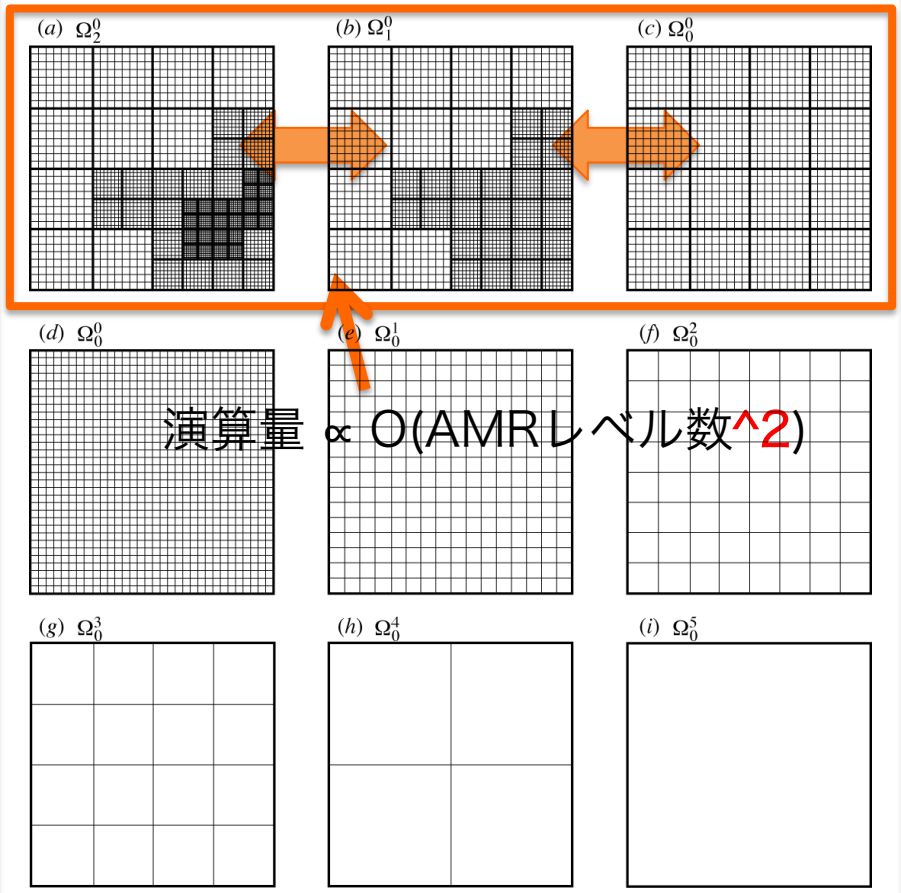
AMRレベルを横断してセルを間引く。
演算量が少ない。




演算量 ∝ O(AMRレベル数)

## MLAT

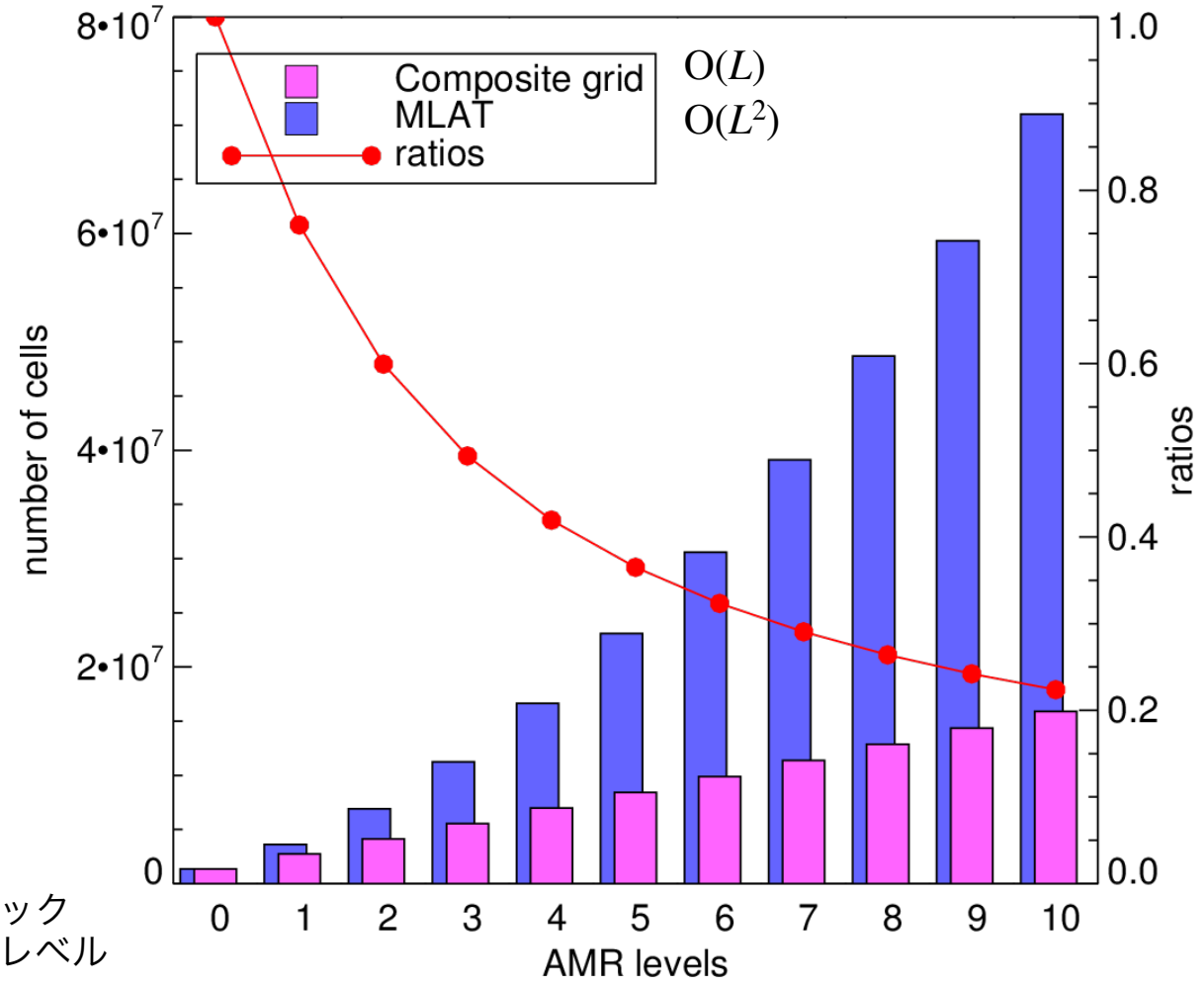
セルを間引かずにAMRレベルを行き来する。
演算量は減らない。




演算量 ∝ O(AMRレベル数^2)

# 合成グリッドの優位性：少ない演算量

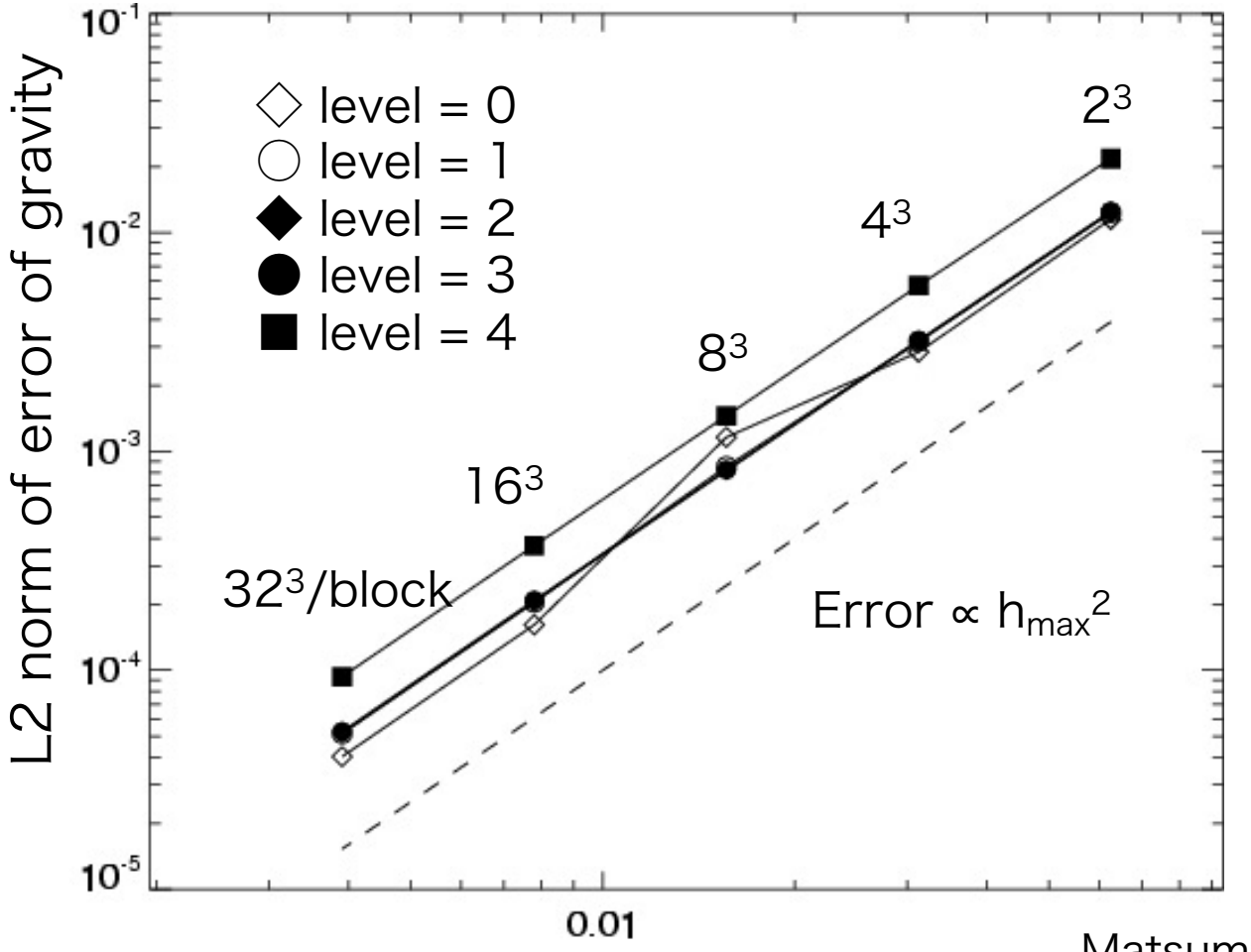
Smoothing が施されるセル数（のべ）




8^3セル/ブロック
8^3ブロック/レベル
$r = 2$
$\nu 1, \nu 2 = 1$

# 自己重力（Multigrid法）の精度

## 空間２次精度




- Source: binary stars
- Maximum level = 4
- Distribution of blocks is fixed.
- Number of cells inside a block is changed.

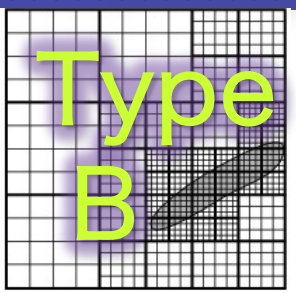
Matsumoto 2007
doi:10.1093/pasj/59.5.905

# おまけ

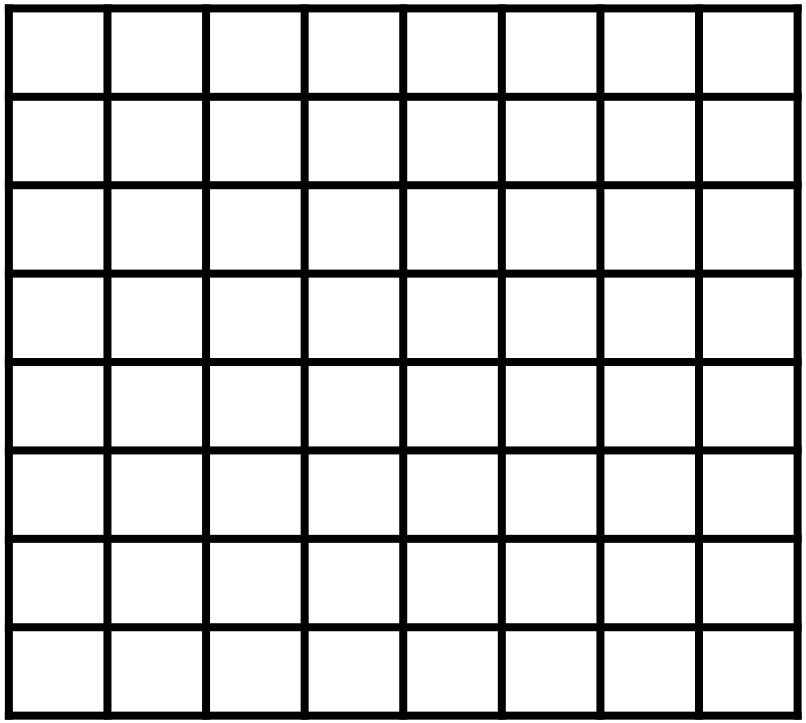
# 並列化の方法

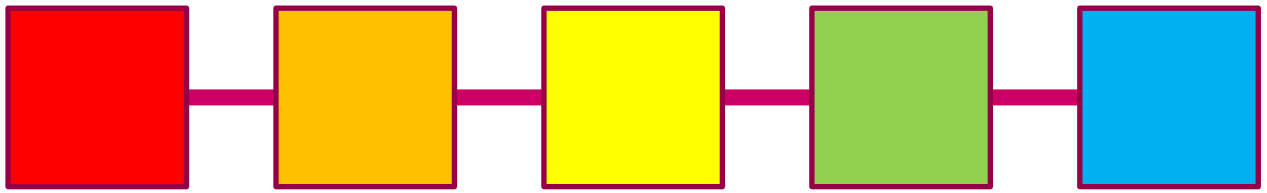
- Block structured gridの場合
  - ブロックをノードに割りつける。
  - 通信量を少なくするために
    - 近所のブロックを同じノードに割り付ける。
    - 近所のブロックを近所のノードに割りつける。

# 並列化： 良くない方法

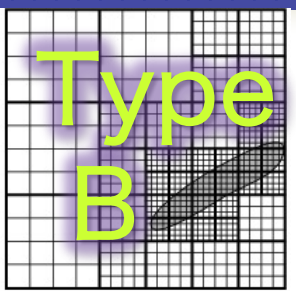



$8^2$ブロック
= 64 ブロック
=13 ブロック/ノード×5ノード
-1ブロック





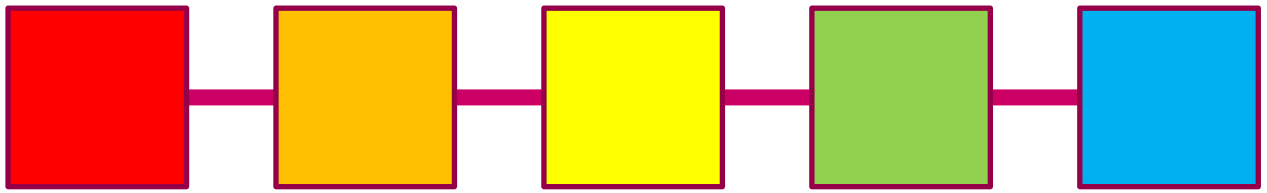
# 並列化： 良くない方法



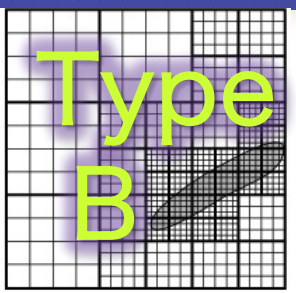

$8^2$ブロック
= 64 ブロック
=13 ブロック/ノード×5ノード
-1ブロック

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
| 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 |
| 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 |
| 25 | 26 | 27 | 28 | 29 | 30 | 31 | 32 |
| 33 | 34 | 35 | 36 | 37 | 38 | 39 | 40 |
| 41 | 42 | 43 | 44 | 45 | 46 | 47 | 48 |
| 49 | 50 | 51 | 52 | 53 | 54 | 55 | 56 |
| 57 | 58 | 59 | 60 | 61 | 62 | 63 | 64 |

ノード



# 並列化： 良くない方法




$8^2$ブロック
= 64 ブロック
=13 ブロック/ノード×5ノード
-1ブロック

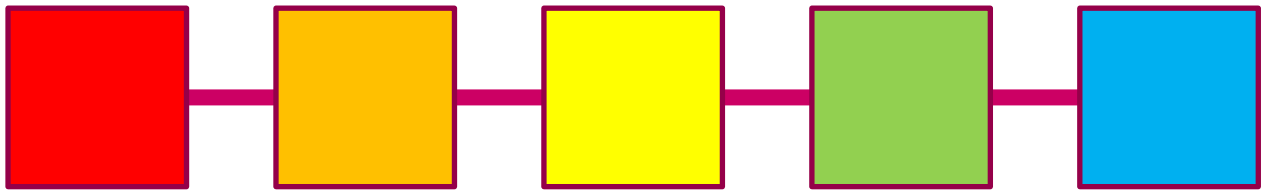
同じノードに割り付けられるブロックが細長く分布。

通信量は多い

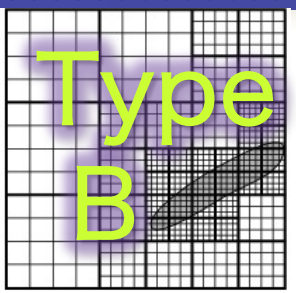
ブロック数・ノード数が多くなると、不利になる。
泣き別れのブロックなど。

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
| 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 |
| 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 |
| 25 | 26 | 27 | 28 | 29 | 30 | 31 | 32 |
| 33 | 34 | 35 | 36 | 37 | 38 | 39 | 40 |
| 41 | 42 | 43 | 44 | 45 | 46 | 47 | 48 |
| 49 | 50 | 51 | 52 | 53 | 54 | 55 | 56 |
| 57 | 58 | 59 | 60 | 61 | 62 | 63 | 64 |

ノード



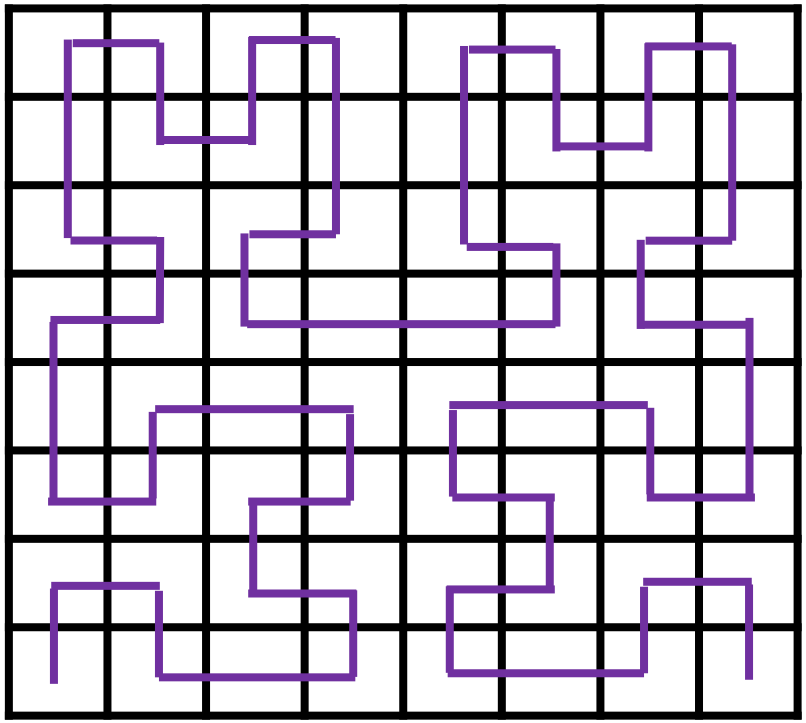
# 並列化： 良い方法



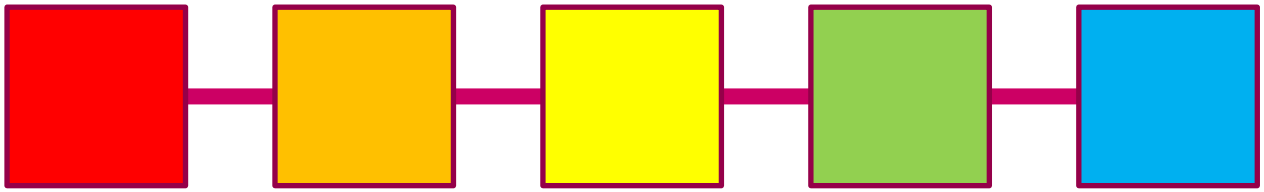

$8^2$ブロック
= 64 ブロック
=13 ブロック/ノード×5ノード
-1ブロック

Peano-Hillbert
空間充填曲線
によるオーダリン
グ

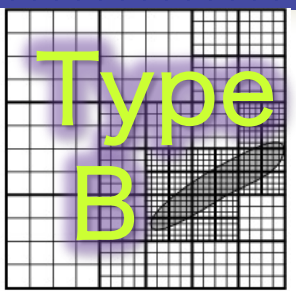
なるべく近くを通る
一筆書き



ノード



# 並列化： 良い方法



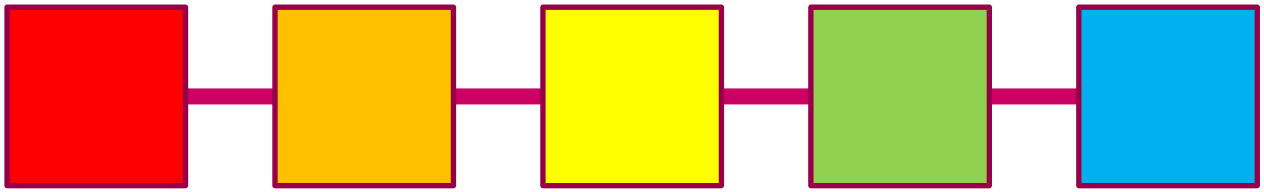

$8^2$ブロック
= 64 ブロック
=13 ブロック/ノード×5ノード
-1ブロック

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 22 | 23 | 26 | 27 | 38 | 39 | 42 | 43 |
| 21 | 24 | 25 | 28 | 37 | 40 | 41 | 44 |
| 20 | 19 | 30 | 29 | 36 | 35 | 46 | 45 |
| 17 | 18 | 31 | 32 | 33 | 34 | 47 | 48 |
| 16 | 13 | 12 | 11 | 54 | 53 | 52 | 49 |
| 15 | 14 | 9 | 10 | 55 | 56 | 51 | 50 |
| 2 | 3 | 8 | 7 | 58 | 57 | 62 | 63 |
| 1 | 4 | 5 | 6 | 59 | 60 | 61 | 64 |

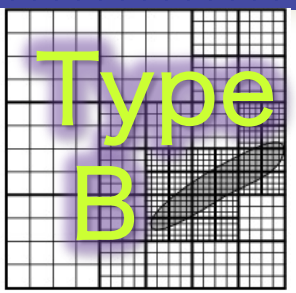
同じノードに割り付けられるブロックがコンパクトに分布。

通信量は少ない。

ノード



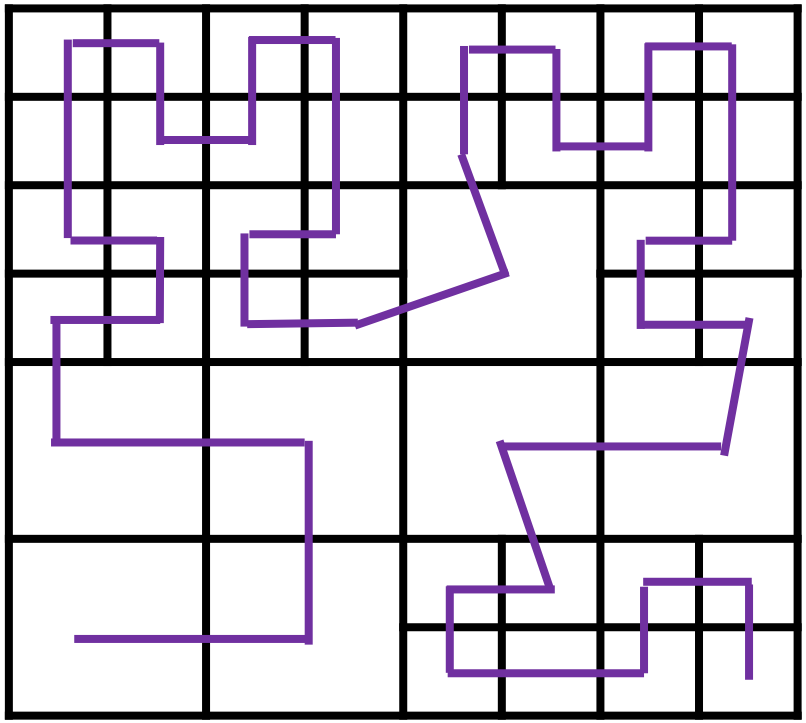
# 並列化： グリッドレベル横断



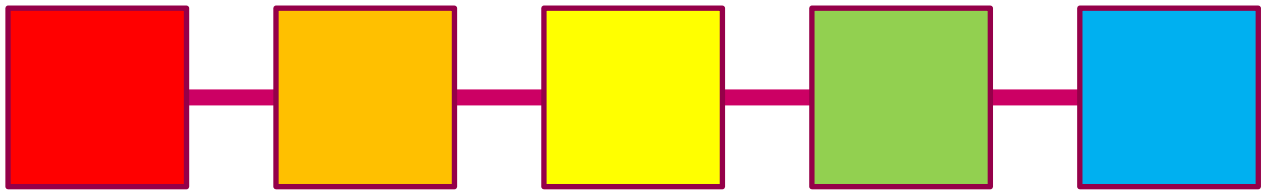

= 43 ブロック
= 8 ブロック/ノード×5ノード
+3ブロック

Peano-Hillbert
空間充填曲線
によるオーダリン
グ

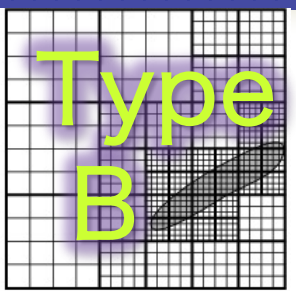
なるべく近くを通る
一筆書き



ノード



# 並列化： グリッドレベル横断

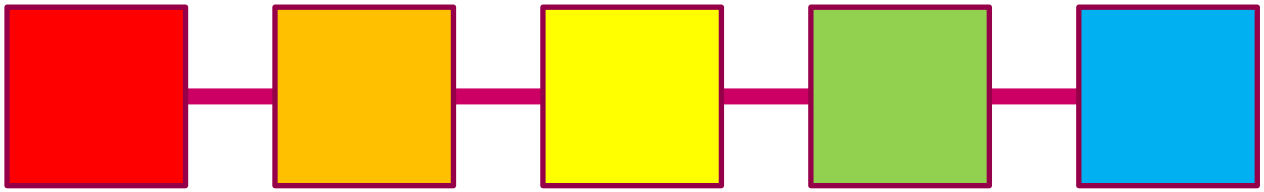



= 43 ブロック
= 8 ブロック/ノード×5ノード
+3ブロック

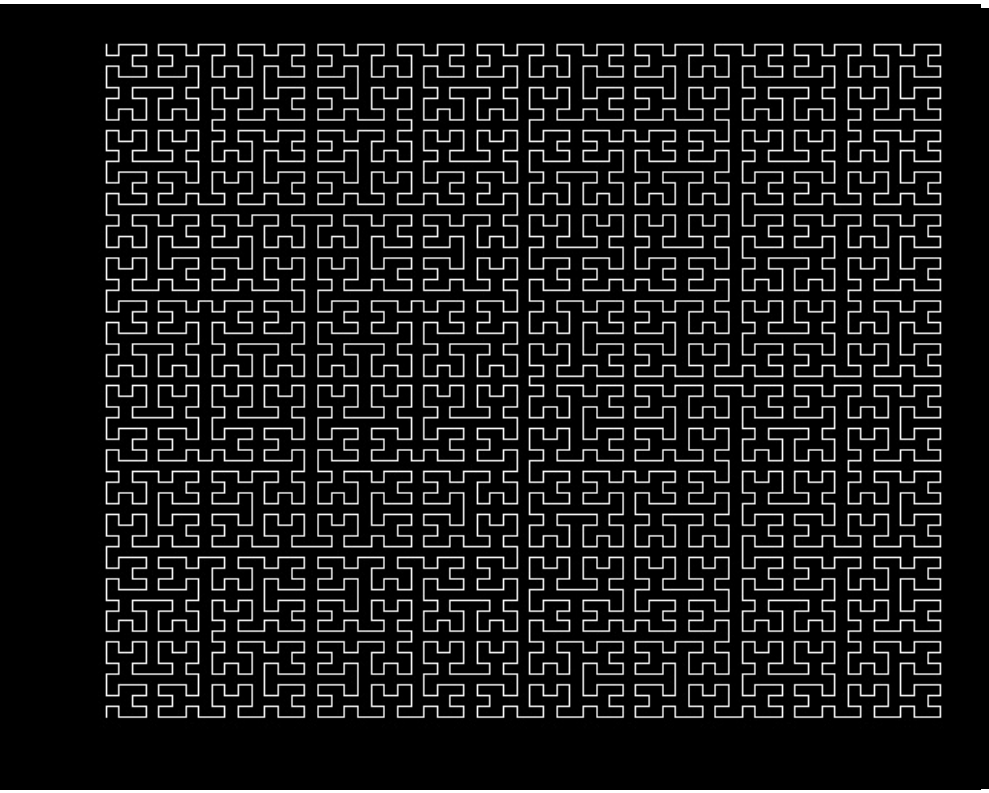
Peano-Hillbert
空間充填曲線
によるオーダリン
グ

なるべく近くを通る
一筆書き

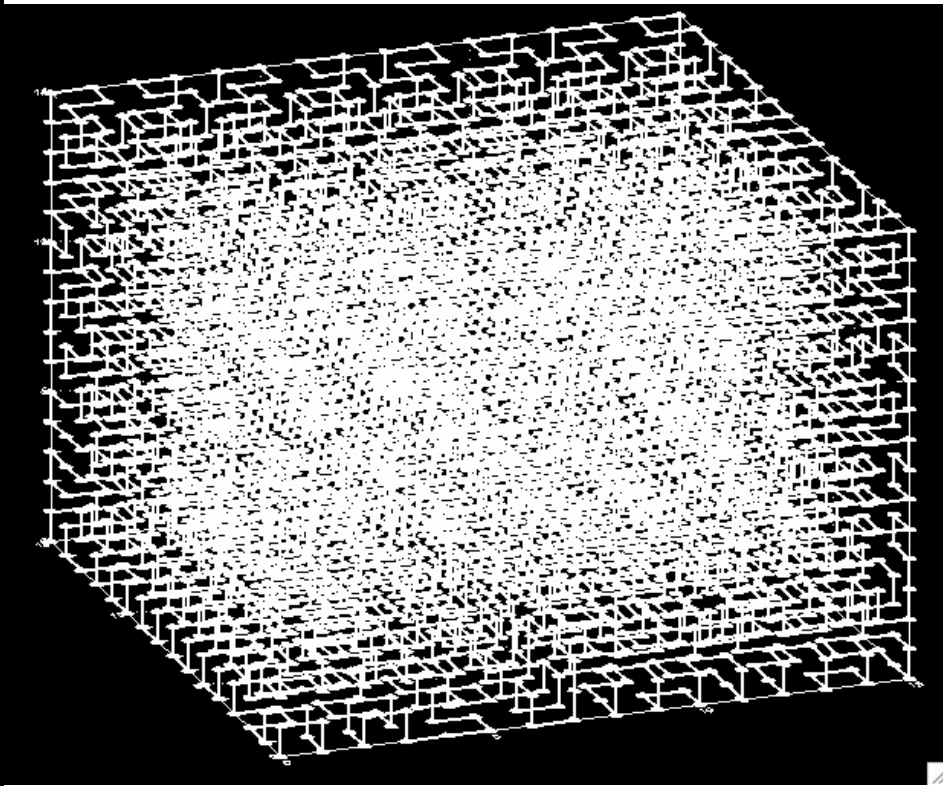

ノード



# 3次元でも同様



2次元のPeano-Hillbert 空間充填曲線



3次元のPeano-Hillbert 空間充填曲線

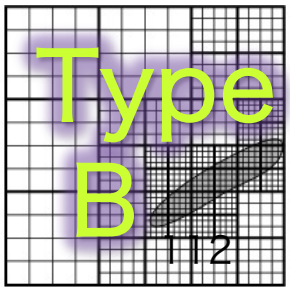
# ブロックのオーダリングまとめ

- Peano-Hillbert空間充填曲線によるオーダリングは、面倒そうですが、意外に簡単な工夫です。

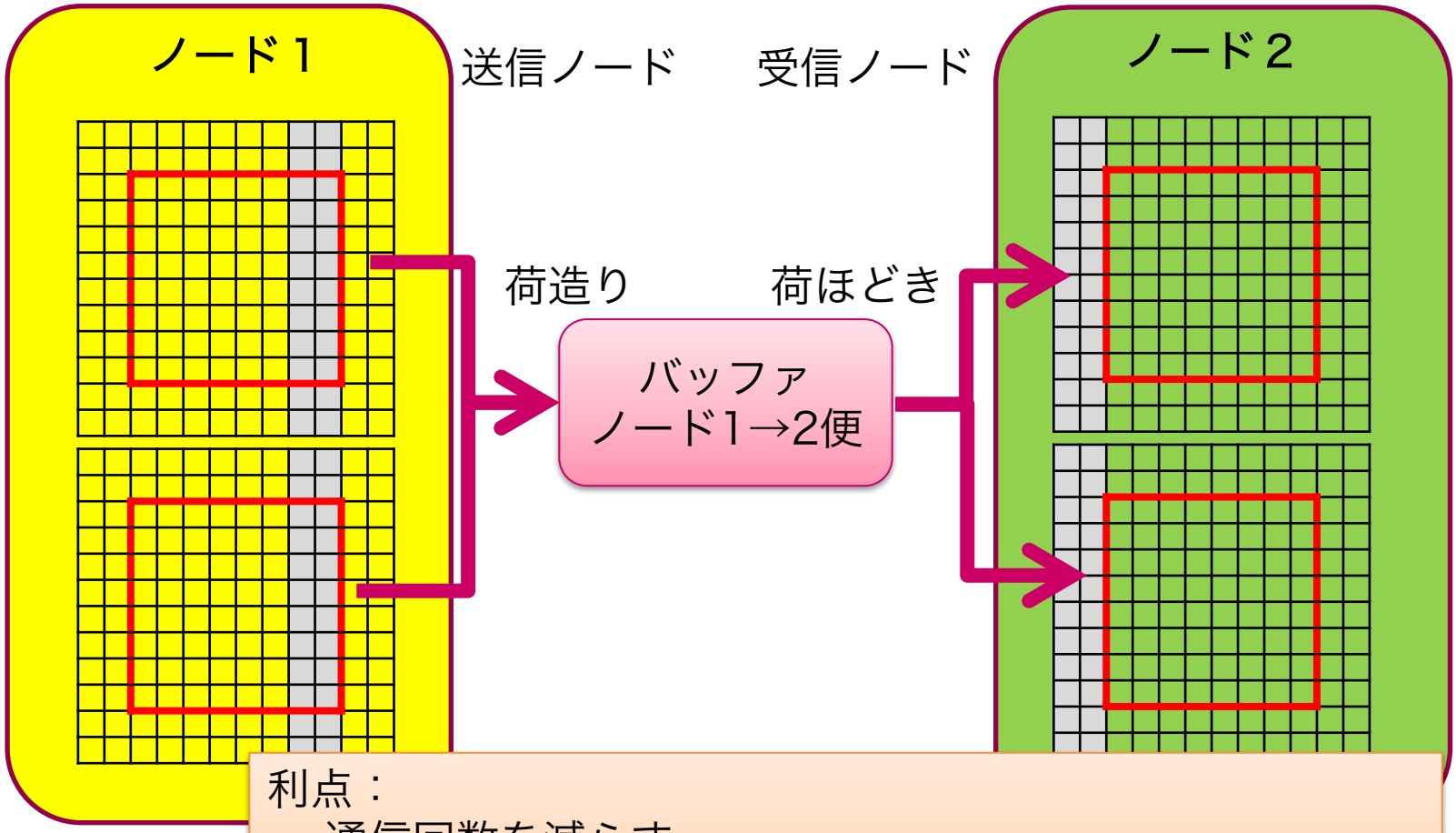
- グリッドレベルごとに行う。
  - Adaptive time step

- グリッドレベルを横断して行う。
  - Synchronous time step

  僕は使い分けてないけど。。。

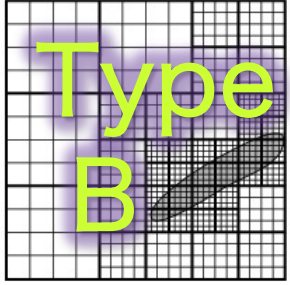


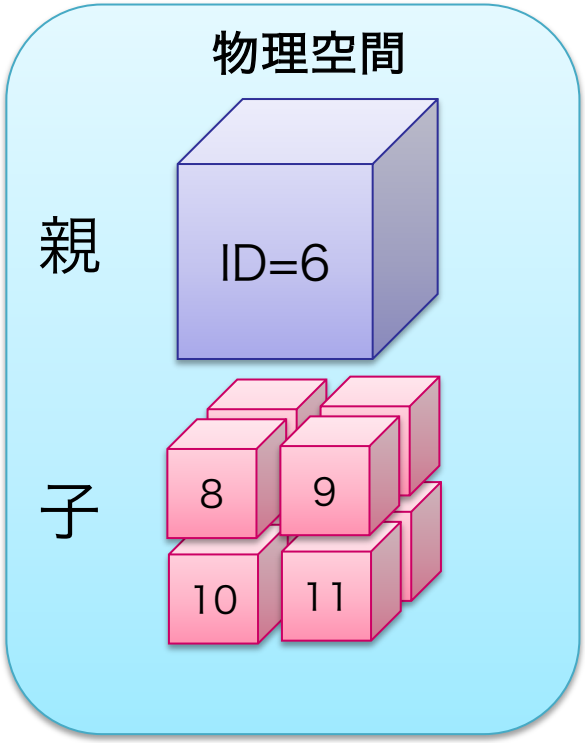
# 袖の転送： ノード間をまとめて転送する

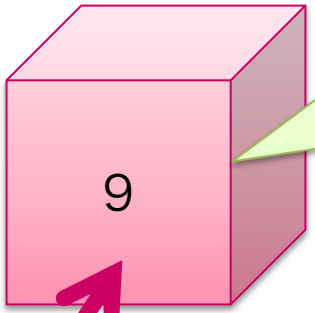



利点：
- 通信回数を減らす。
  性能は通信量ではなく通信回数で決まる。
- MPI通信タグの枯渇を回避する。
  ブロックが個別に通信するとタグ数が上限を超える。

# データ構造：ブロックの八分木構造




親のID×1
子供のID×8
隣のID×6
グリッドレベル
ブロックの位置(i,j,k)

セル Nx×Ny×Nz 個