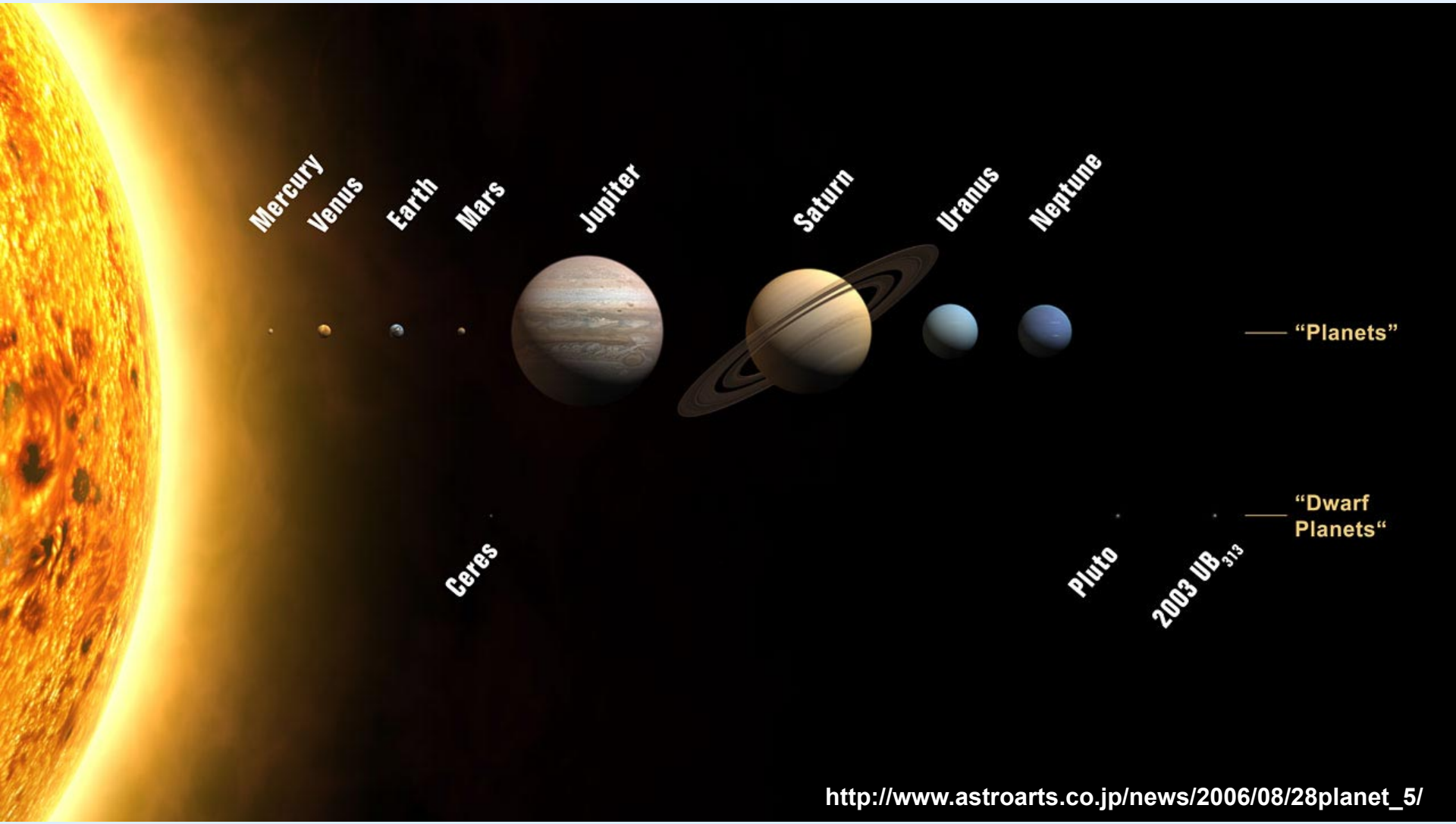
太陽系における地球型惑星の水の起源
――惑星形成の大域シミュレーション――

小南　淳子
（東京工業大学地球生命研究所）

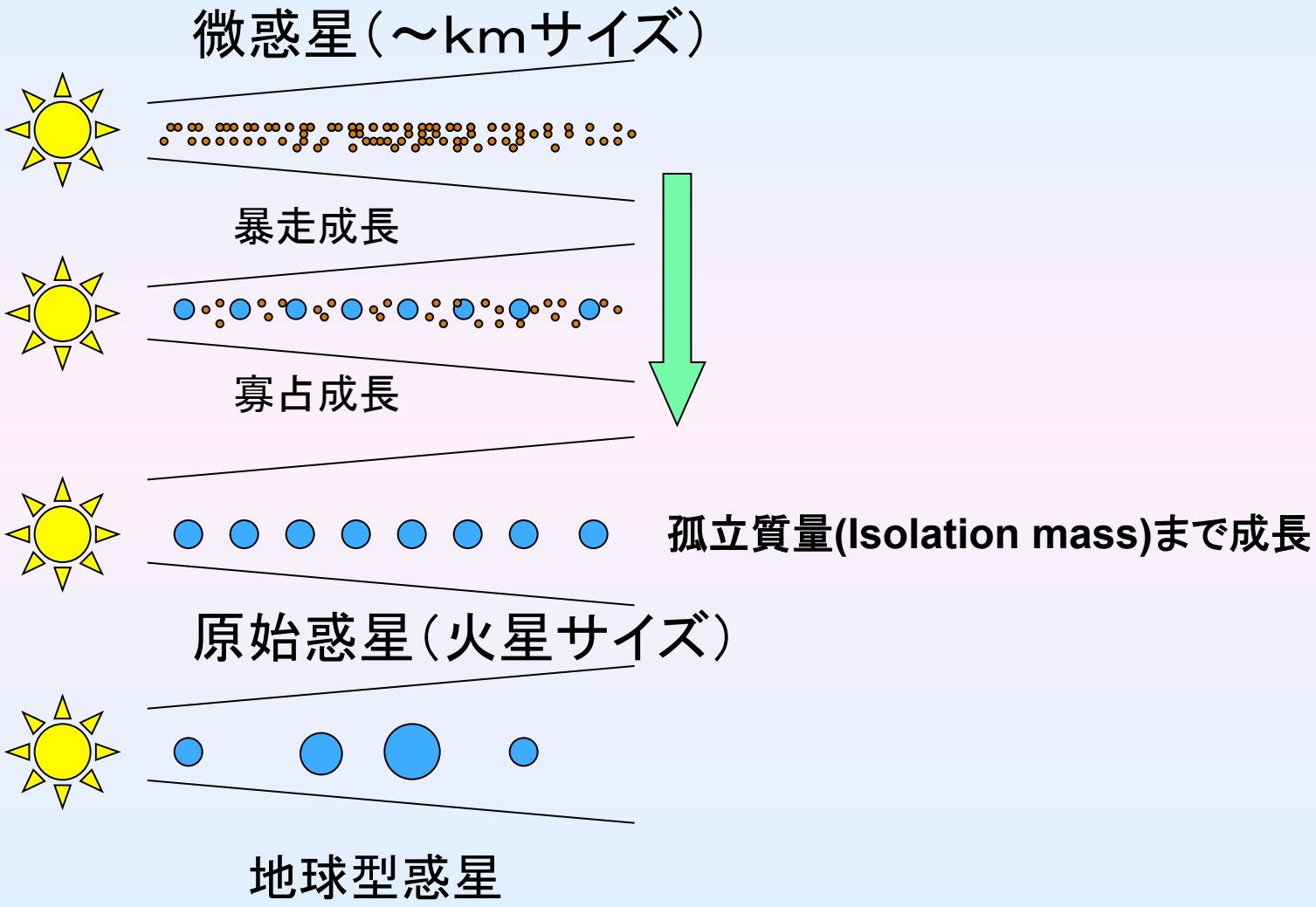
台坂博（一橋）、似鳥啓吾（理研）、
牧野淳一郎（東工大）

太陽系



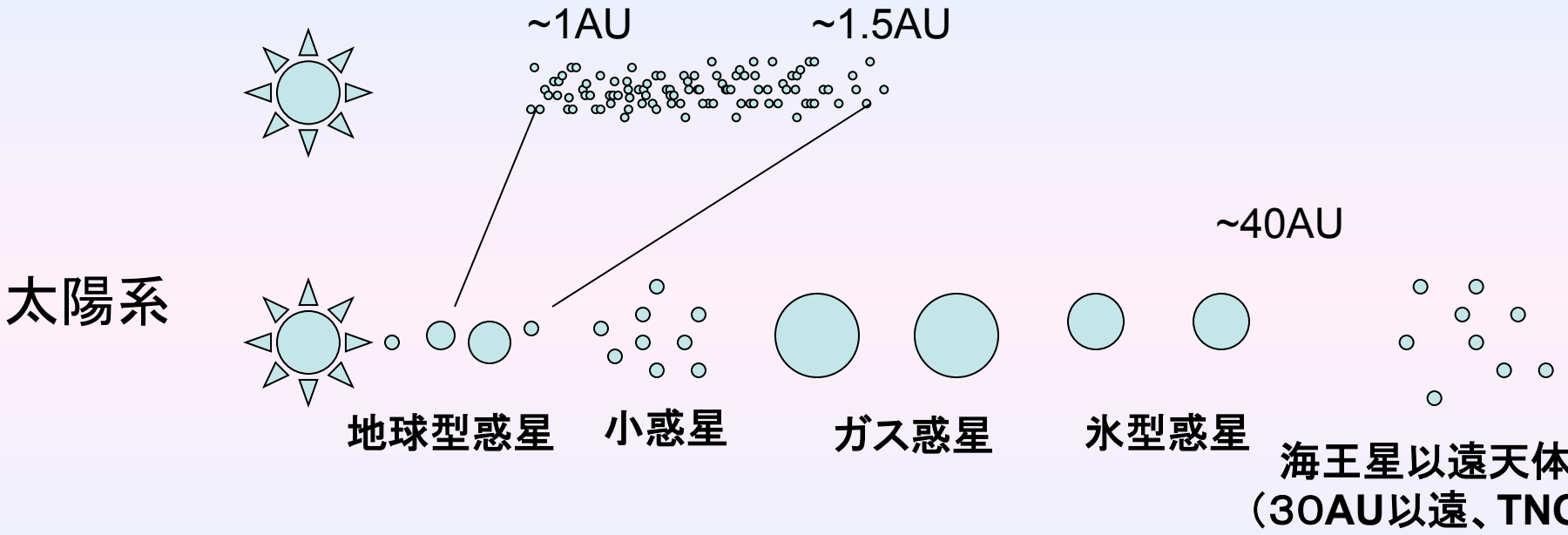
http://www.astroarts.co.jp/news/2006/08/28planet_5/

惑星の形成シナリオ


微惑星（～kmサイズ）
暴走成長
寡占成長
孤立質量(Isolation mass)まで成長
原始惑星（火星サイズ）
地球型惑星

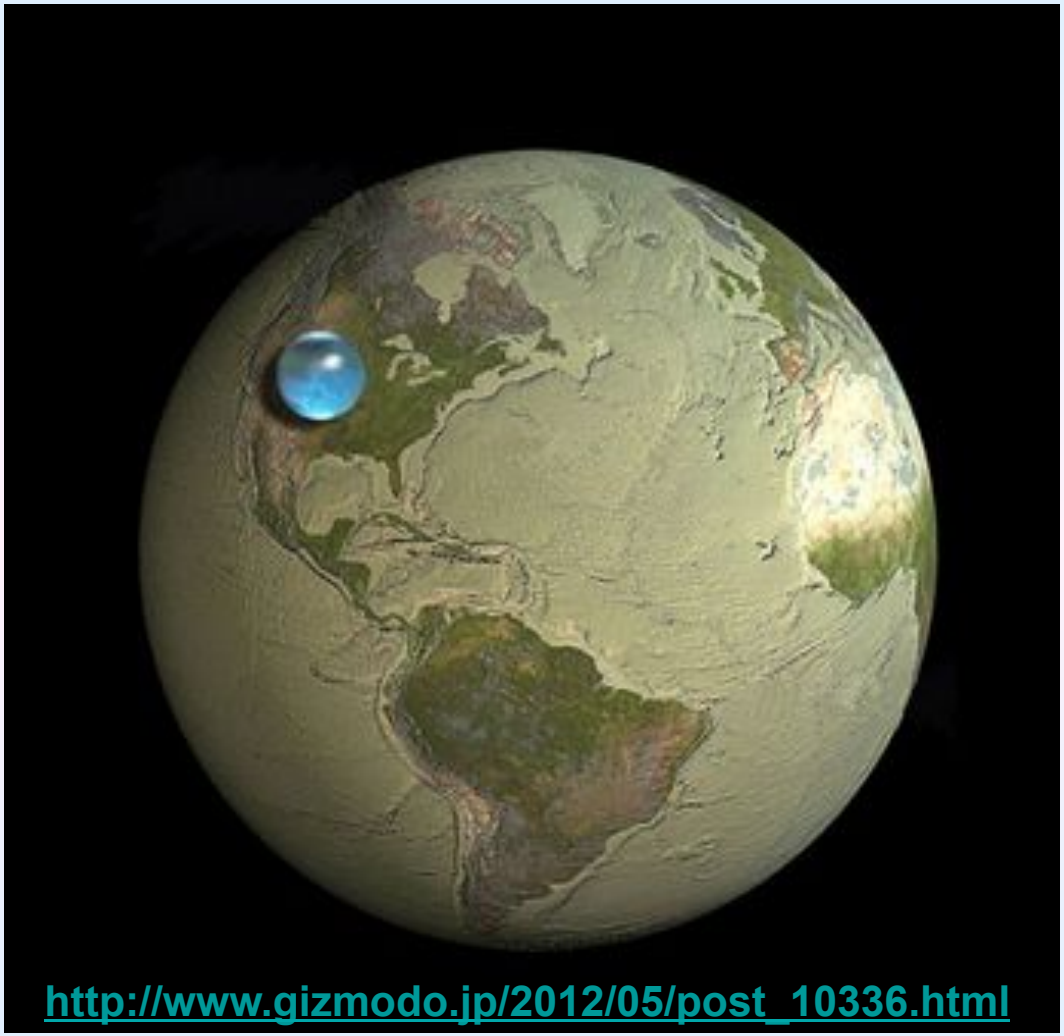
集積計算領域

現在までのN体計算領域（地球型惑星付近に限られる）



粒子数の制約によりN体計算領域は狭く限られていた

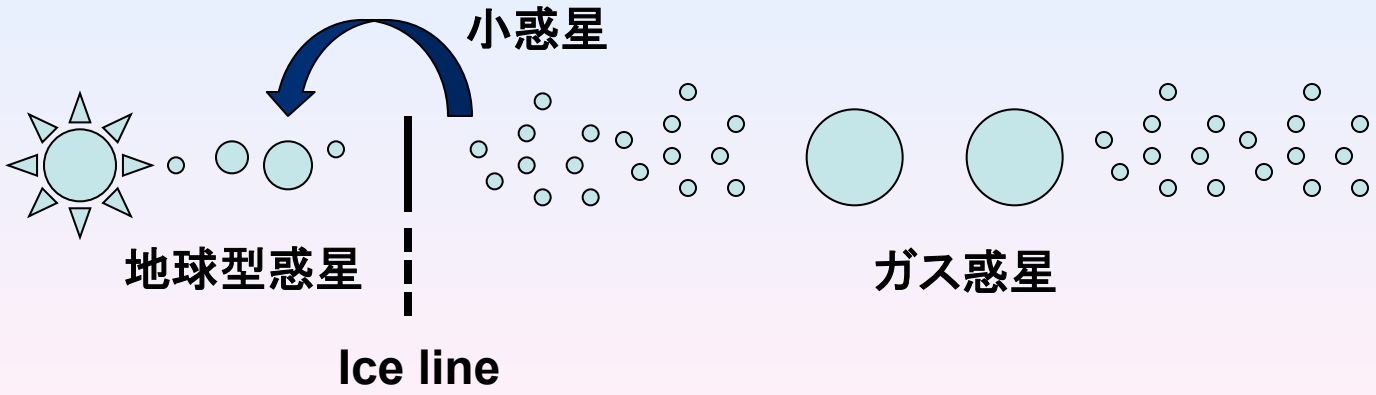
地球と水



http://www.gizmodo.jp/2012/05/post_10336.html

地球の水の重さは、地球の重さの0.02%しかない

惑星形成論の問題点：水の存在



- Ice line より外側のものが降り、地球に水が存在
- 効率よく地球に小惑星が衝突したら地上の水は現在の10倍以上というシミュレーション結果
- 「効率」はガス惑星の移動や原始惑星の質量などにもよる

なぜ地球上の水が0.02 重量% なのかはまだ不明

なぜ太陽系形成論で
水の存在量が説明できないか

局所的な計算しか行われていない

グローバルな計算の必要性

木星土星の形成時に小惑星帯から外側の微惑星が
地球型惑星へ及ぼす影響が明らかになる

本研究の目的

グローバルな惑星形成シナリオの構築
地球上の水の起源の解明

「京」での惑星形成計算

従来の研究　：　GRAPE-DRやGPU単体での計算

——独立時間刻みは並列化に向かない

スケーラビリティの高いアルゴリズムは提案されているが
惑星形成計算向けの実装としては、
「京」向けの実装はまだない

今回新規開発した

計算手法(KninjaX)

- 並列版N体計算（エルミート積分＋ブロックタイムステップ）
- 6次精度（今回は4次）
- 衝突合体あり（今回は入れていません）
- 並列化アルゴリズムはninja法（Nitadori et al. ,2006）を参考にコード作成
- 並列はMPI と OpenMP

Block timestep アルゴリズム

独立時間刻み法

それぞれの粒子が時間　t_i　とタイムステップ Δt_i を持っている

①一番小さい $t_i + \Delta t_i$を選ぶ

②選んだものを積分し、新たな時間 $t_{i,new} = t_i + \Delta t_i$ を設定する

③繰り返す

Block timestep 法

時間刻みを2の整数乗にすることで同じ時間刻みを持つものが
同じ時間をもつことができるようにする

同じ時間をもったものが積分される（Active particle とよぶ）

N体並列計算

従来の方法（粒子を分割して持つ）

- ノードあたりの通信量がノード数によらない：
  通信がボトルネックになる
- Active 粒子が少ないときにロードバランスが悪化

Ninja アルゴリズム

- ノードあたりの通信量がノード数の平方根に反比例して減る
- Active 粒子数に無関係に、ほぼ完璧なロードバランス

ベンチマーク計算で使った条件

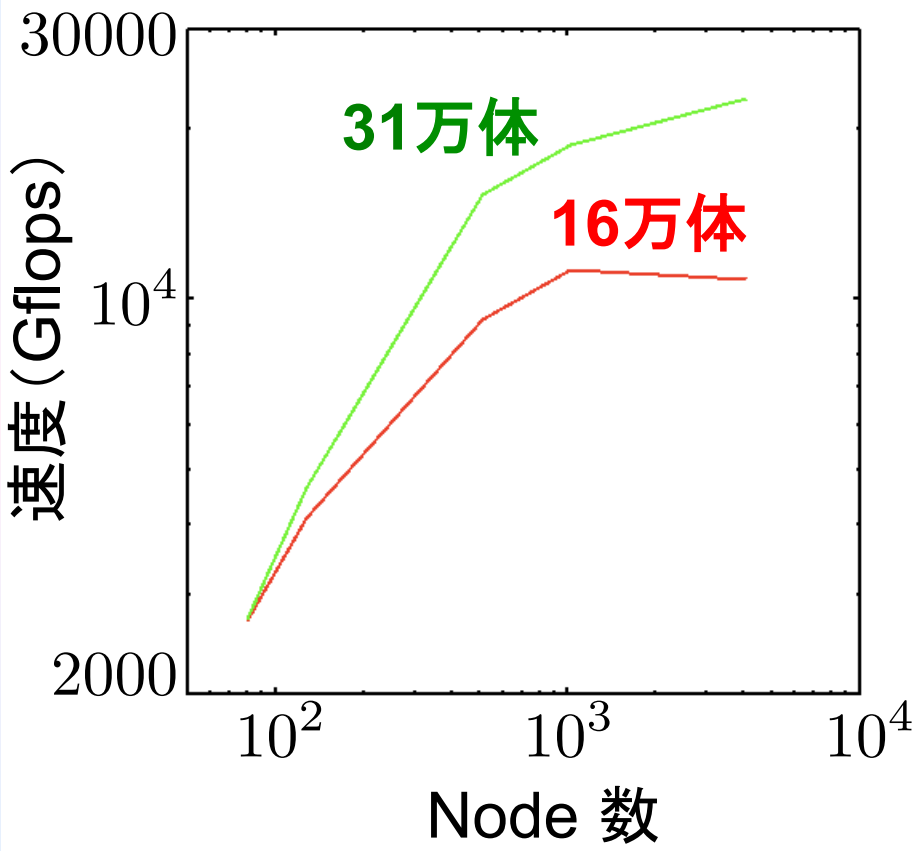
- 初期微惑星円盤　0.5 – 40AU
- 初期微惑星質量　$2 \times 10^{23} g, 10^{23} g$
- 積分時間　　2年
- 微惑星の数　　16万体、31万体
- 面密度分布　林モデル　$\Sigma = 7(r/1\mathrm{AU})^{-3/2} g/cm^2$
- 速度分散　Rayleigh 分布

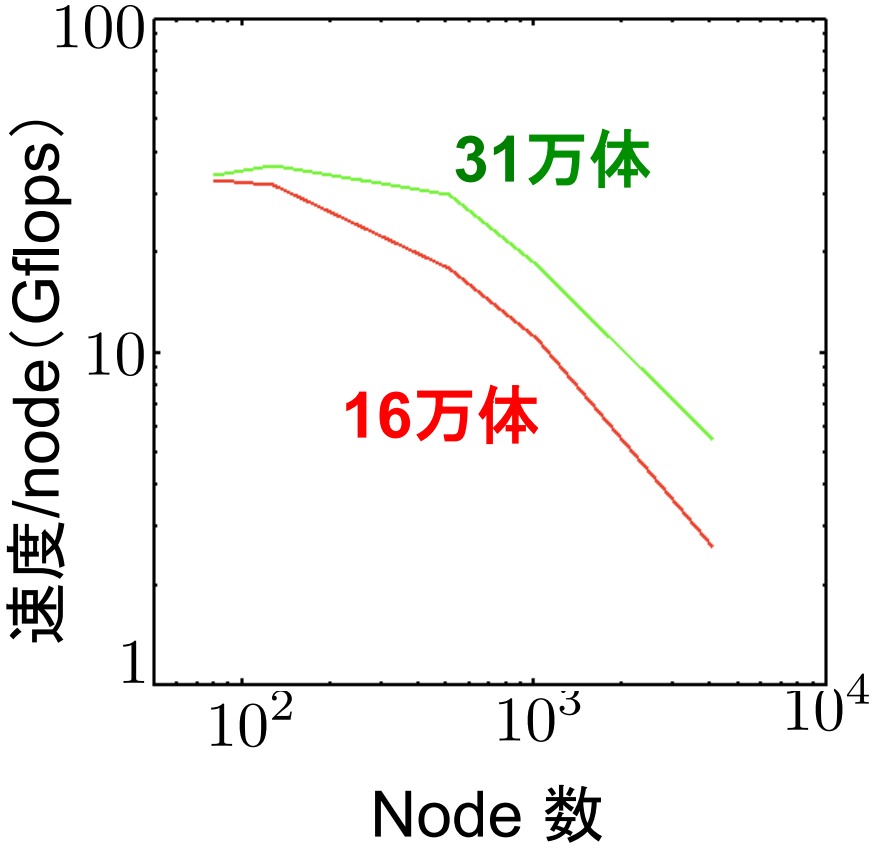
$$< e^2 >^{1/2} = 0.4(m/3M_\odot)^{1/3} = 2 < i^2 >^{1/2}$$

ベンチマーク計算方法

- １次元のプロセッサーのアロケーション
- 計算時間を実測、update particle のトータルを計算し、相互作用数をだし、Gflops を出す

性能評価１

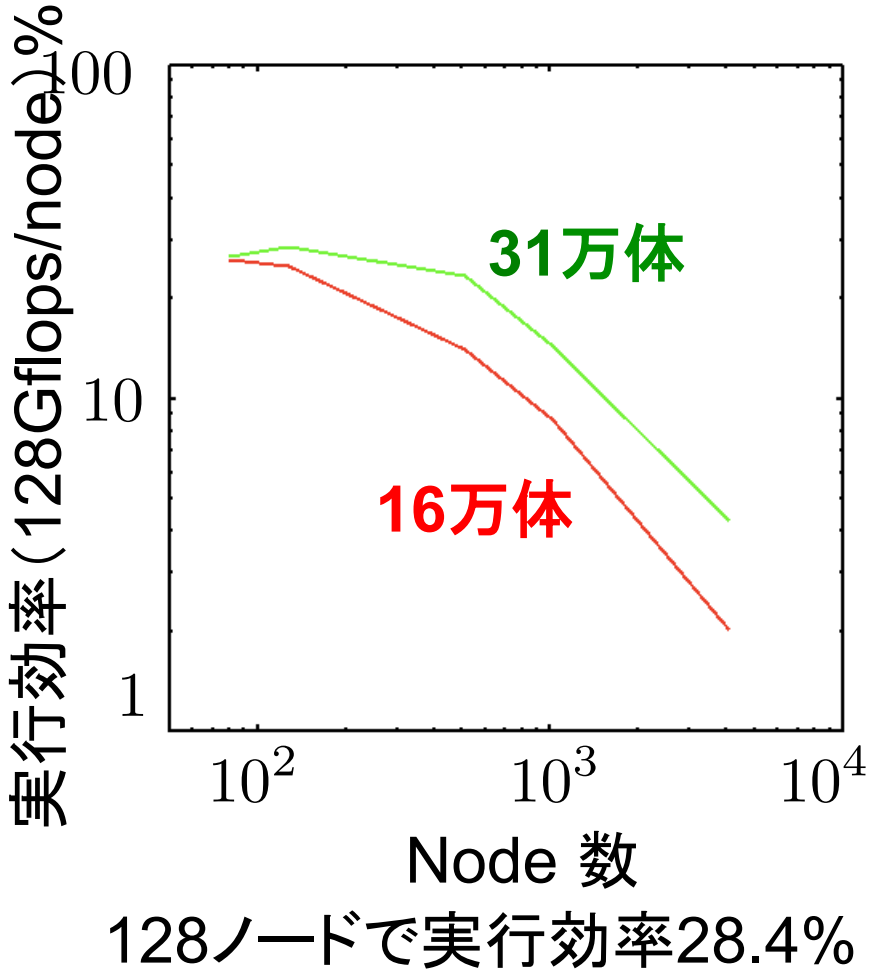




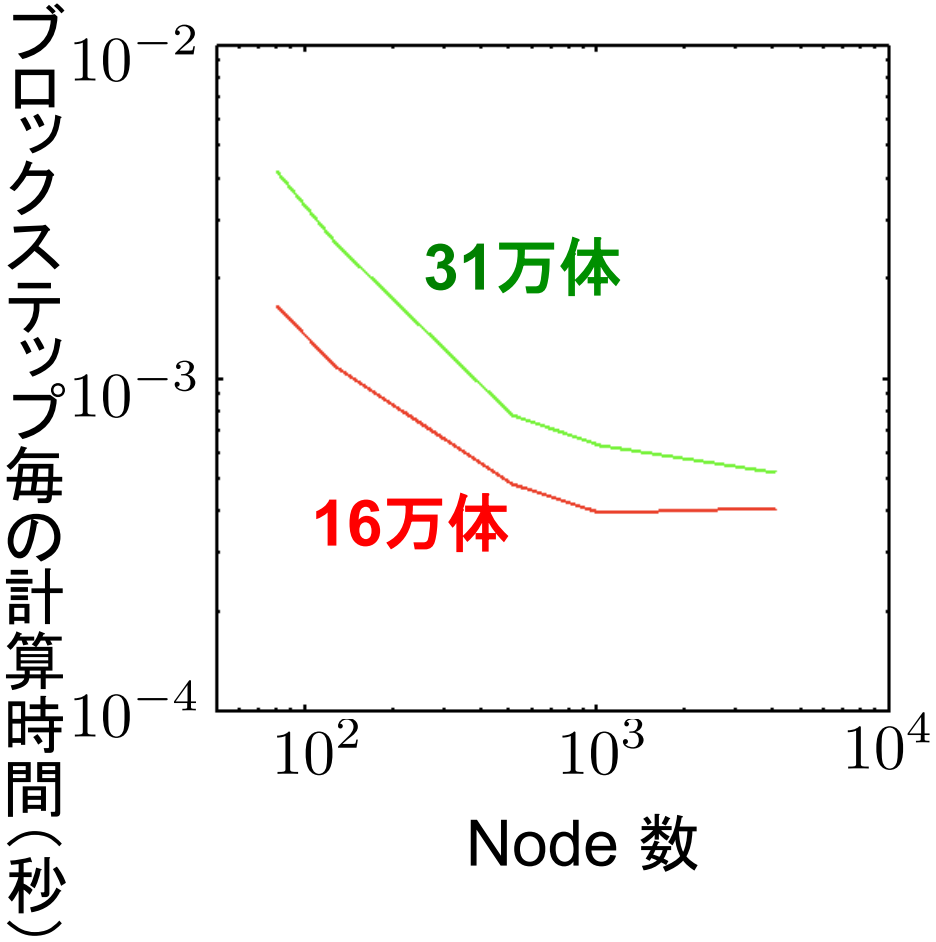
今回の初期条件では～512ノードで速度が飽和

ノード数の少ない所で、～33Gくらいでている

性能評価2



128ノードで実行効率28.4%



ノード数少ない所では実行効率
～30%の十分な速度

1タイムステップあたり0.5ミリ秒
程度でMPI通信のオーバーヘッドが
支配的と思われる

現在まで到達した点

- 京で走る並列4次エルミート（KninjaX）のコードを作成
- 16万体、31万体で性能評価
  - ノード数の少ない所では理論ピークの30%と十分な速度
  - 今回の初期条件では～512ノードで速度が飽和
  - 1タイムステップあたり0.5ミリ秒程度でMPI通信のオーバーヘッドが支配的

次年度に行う計算の見積もり

- 面密度　$\Sigma = 7(1/1\mathrm{AU})^{-3/2} g/cm^2$
- 粒子数　31万体
- 積分時間　10万年

→ 512ノードで3ヶ月で計算可能

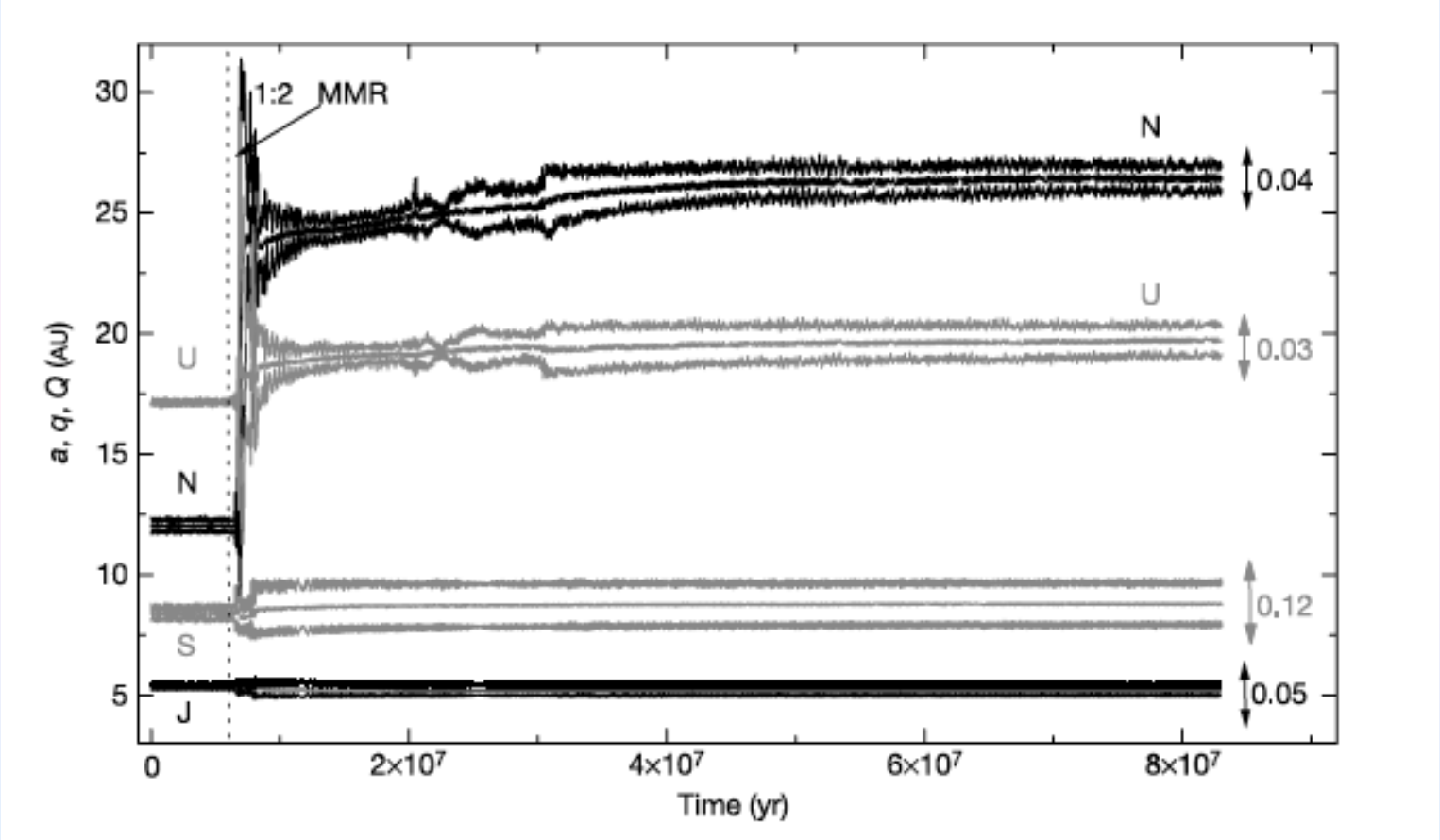
今後の方針

- 6次エルミートの導入
- 2次元でプロセッサのアロケーションをしてみる
- 衝突合体の導入
- 木星土星をいれる

おわり

太陽系を再現しようとしたモデル

ニースモデル（Tsiganis et al. 2005）



しかし、まだ問題点は存在する

ニースモデルの問題点

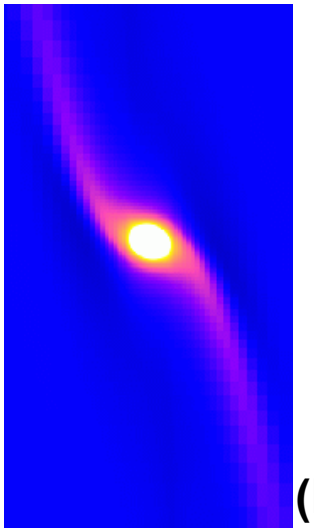
- 天王星、海王星の外側にしか微惑星を置いていない
- 初期のガス惑星の配置に結果が強く依存している

惑星形成論の大きな問題点

惑星形成論の問題：ガス円盤からの効果

（1）ガス抵抗：*a,e,i* の減衰
(Adachi et al. 1976, Tanaka & Ida 1999)

（2）円盤からの重力相互作用：*a,e,i* の減衰
(e.g. Ward 1986, Tanaka et al. 2002, Tanaka and Ward 2004)



惑星が重力でガス円盤に波をたてる

↓

両側の波からトルクがかかる

↓

角運動量が取られて内側に移動

(Morohoshi and Tanaka 2003)

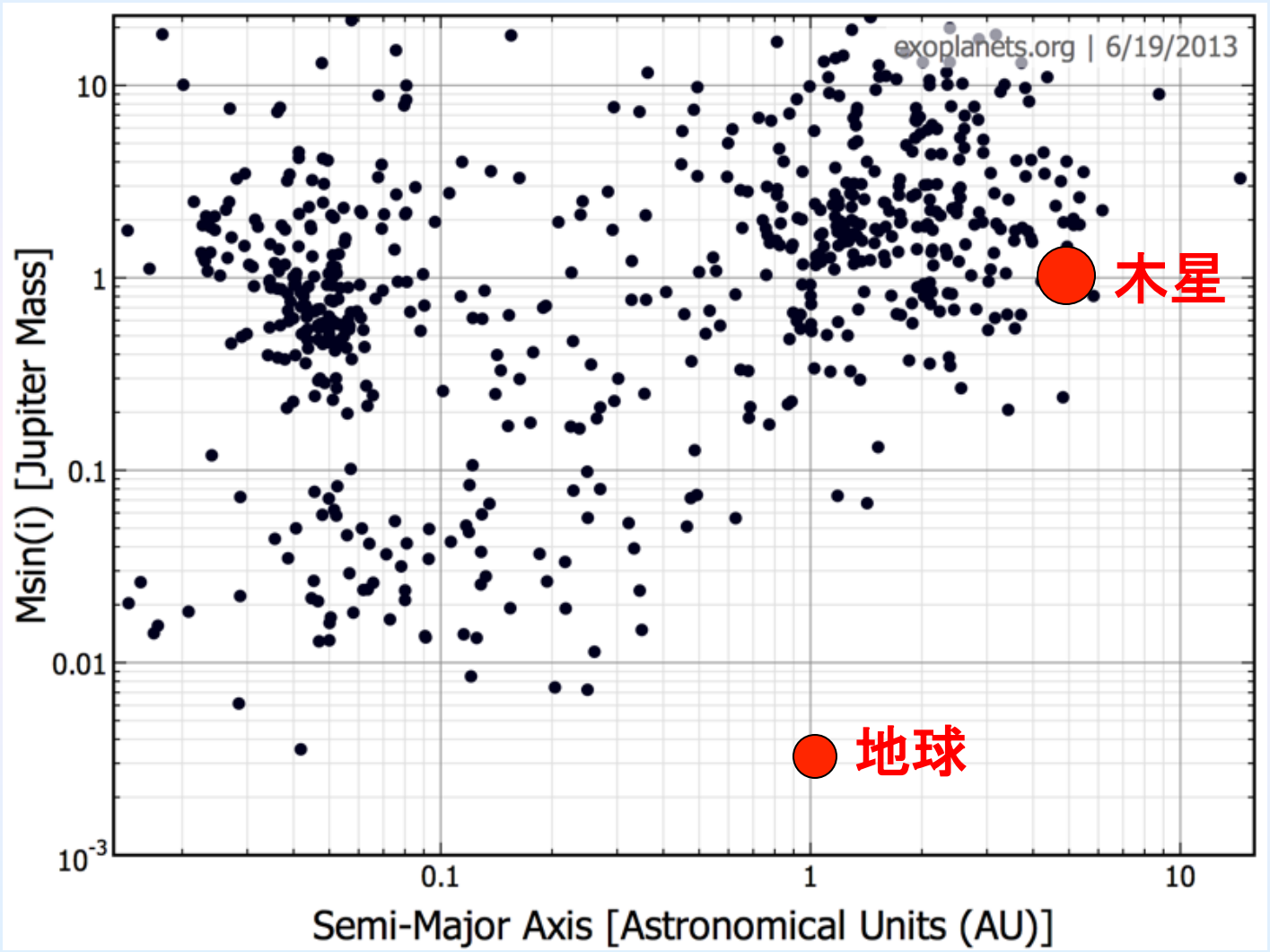
原始惑星の「Type-I 移動」

（1）ガス抵抗 → $\tau_{\rm gas} \propto m^{1/3}$

（2）円盤からの重力相互作用 → $\tau_{\rm grav} \propto m^{-1}$

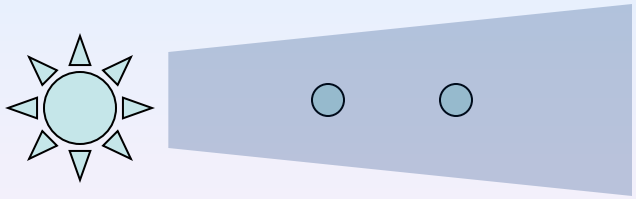
月質量以上になると原始惑星はタイプ1移動で中心星に落ちる

系外惑星と太陽系の比較

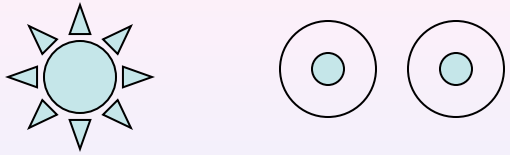


太陽系以外にも様々な系が存在

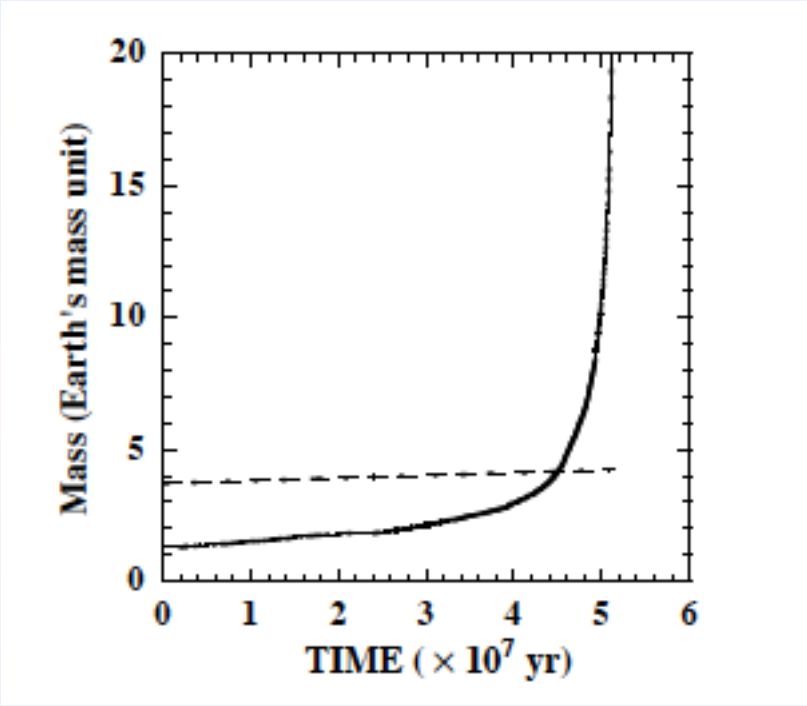
巨大惑星の形成過程



コアが数地球質量まで増加



ガスを暴走的にまとう

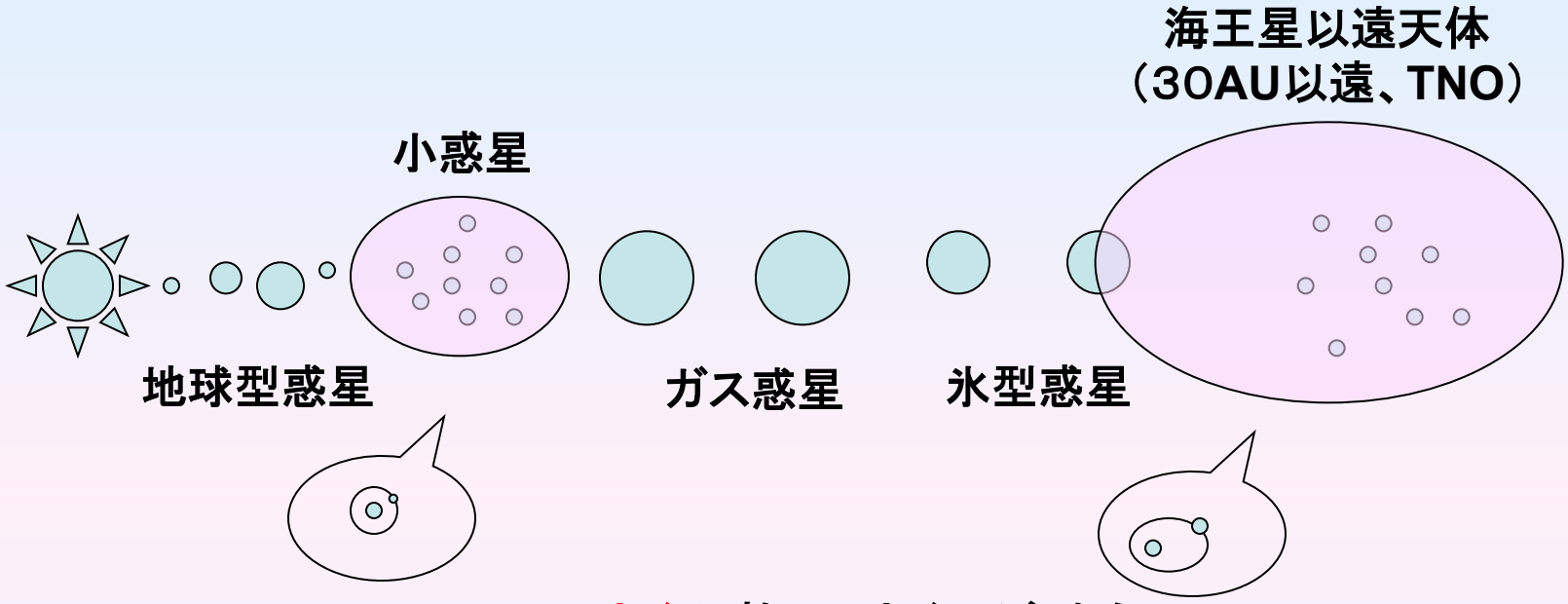


Ikoma et al. (1998)

～5-20地球質量くらいでガス惑星に進化（Ikoma et al. 1998）

ガス円盤がいつ晴れるかというのも重要

円盤外側での集積



ヒル半径/物理半径が違う

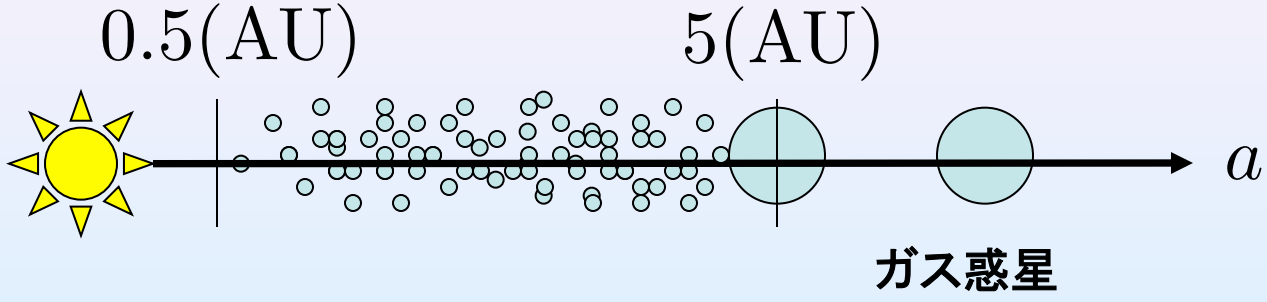
微惑星連星は集積を促進（Kominami et al. 2011）

惑星形成論の問題点

- 微惑星形成、微惑星落下問題
- 原始惑星落下問題（タイプ1惑星移動問題）
- 地球の水の量の問題　　本研究
- 太陽系の外縁が～50AU という問題

初期設定

- 初期微惑星円盤　0.5 – 5AU
- 初期微惑星質量　$10^{23}g$
- 積分時間　$10^{5}\mathrm{years}$
- 微惑星の数　５０万体



N体計算での大領域計算の難しさ

積分時間が長い（１千万年計算したい）ため、
粒子数が多いと計算が終わらない

GRAPE-DR での典型的な計算時間
　→　３万体を１万年計算するのに１週間

本研究で必要な粒子と積分時間
　→　１００万体を数百万年計算したい
　→　～９００００週間

なんとかしたいので並列計算が必要

エルミート積分法（Makino & Aarseth 1992）

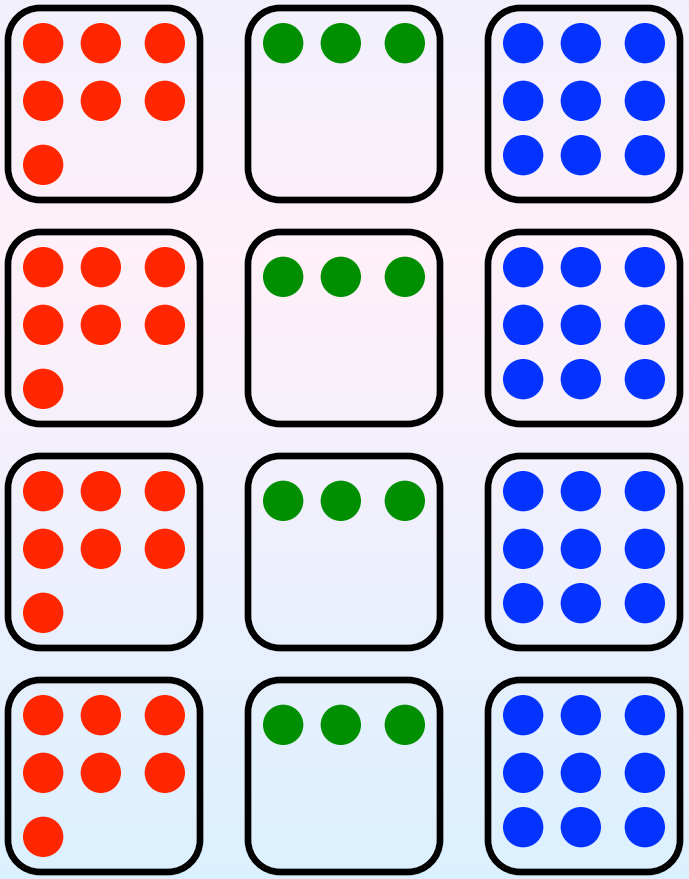
予測子修正子法

加速度、加加速度をもってエルミート補完

過去の１点の情報のみで公式が作れる

Ninjaアルゴリズム

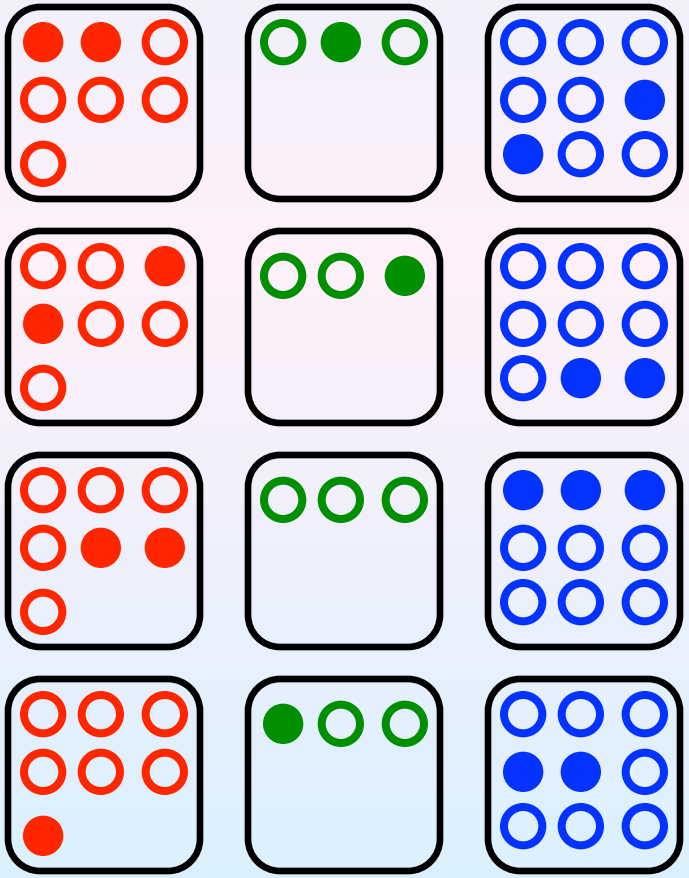
独立時間刻みでも並列化効率が良くなるようにした並列化方法
全プロセッサーを2次元の配列にする（ここでは3x4）。
横方向に一旦粒子を分割、縦方向には同じデータが入る



それぞれの行で、7個、3個、9個の
Active particle が見つかる

Ninjaアルゴリズム

独立時間刻みでも並列化効率が良くなるようにした並列化方法
全プロセッサーを2次元の配列にする（ここでは3x4）。
横方向に一旦粒子を分割、縦方向には同じデータが入る

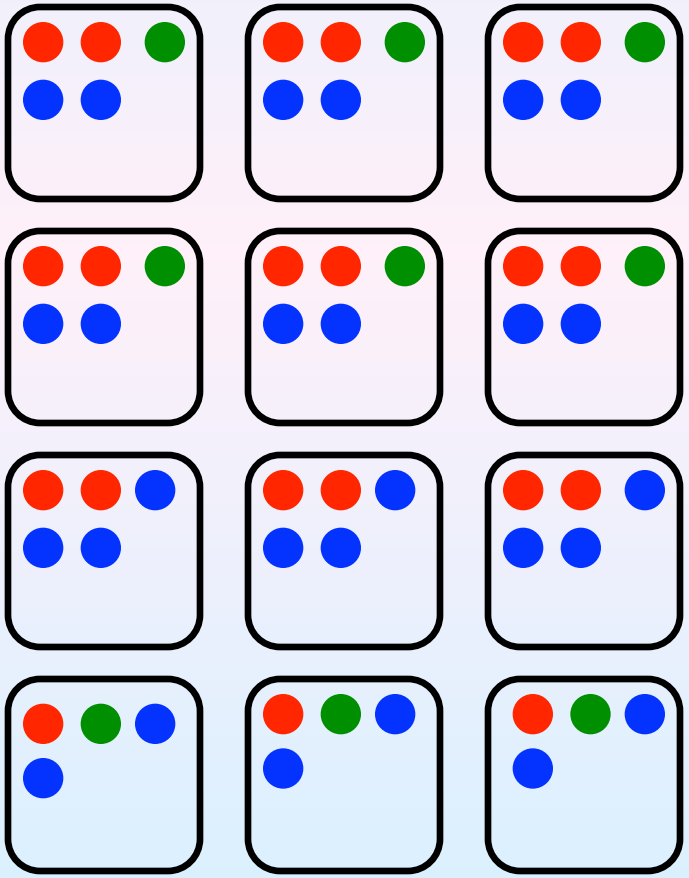


それぞれの行で、7個、3個、9個の
Active particle が見つかる

それぞれのプロセッサーがどの粒子を担当
するのか決める

Ninjaアルゴリズム

独立時間刻みでも並列化効率が良くなるようにした並列化方法
全プロセッサーを2次元の配列にする（ここでは3x4）。
横方向に一旦粒子を分割、縦方向には同じデータが入る



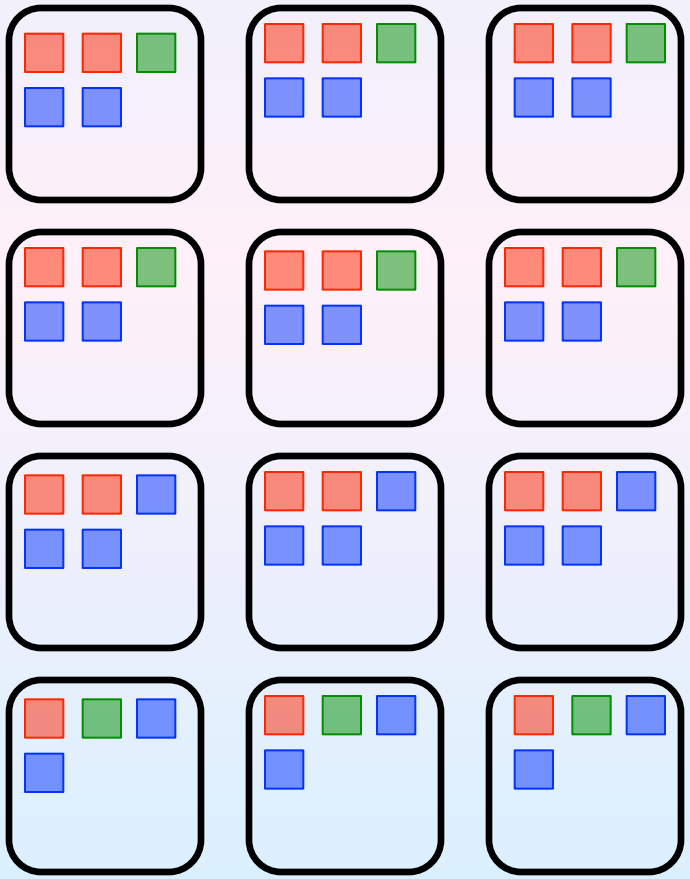
それぞれの行で、7個、3個、9個の
Active particle が見つかる

それぞれのプロセッサーがどの粒子を担当するのか決める

横方向に位置の情報をブロードキャストする

Ninjaアルゴリズム

独立時間刻みでも並列化効率が良くなるようにした並列化方法
全プロセッサーを2次元の配列にする（ここでは3x4）。
横方向に一旦粒子を分割、縦方向には同じデータが入る



それぞれの行で、7個、3個、9個の
Active particle が見つかる

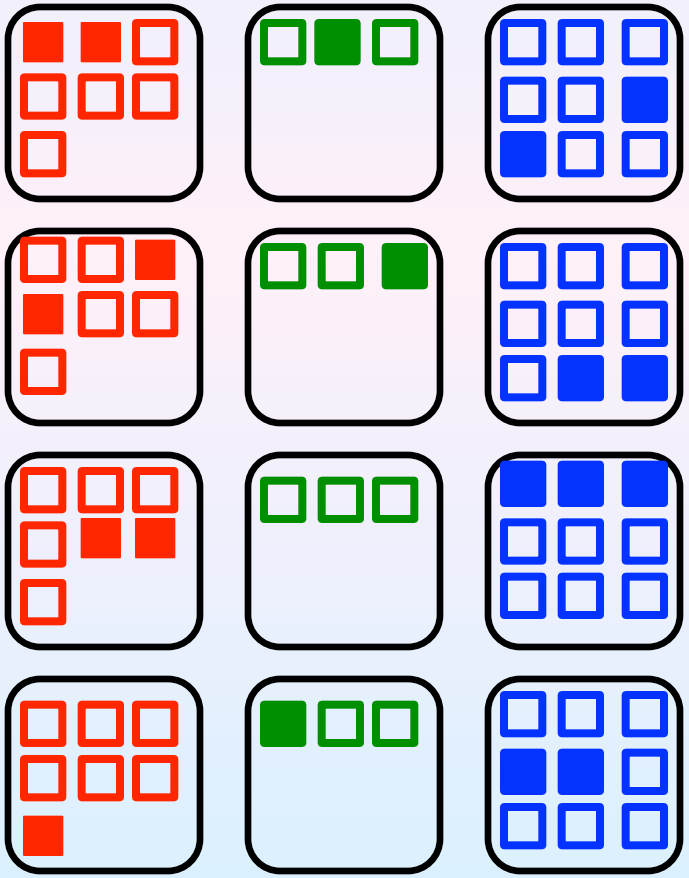
それぞれのプロセッサーがどの粒子を担当するのか決める

横方向に位置の情報をブロードキャストする

Active particle とj粒子の力(部分和)を計算
力の計算はSIMD化

Ninjaアルゴリズム

独立時間刻みでも並列化効率が良くなるようにした並列化方法
全プロセッサーを2次元の配列にする（ここでは3x4）。
横方向に一旦粒子を分割、縦方向には同じデータが入る



それぞれの行で、7個、3個、9個の
Active particle が見つかる

それぞれのプロセッサーがどの粒子を担当するのか決める

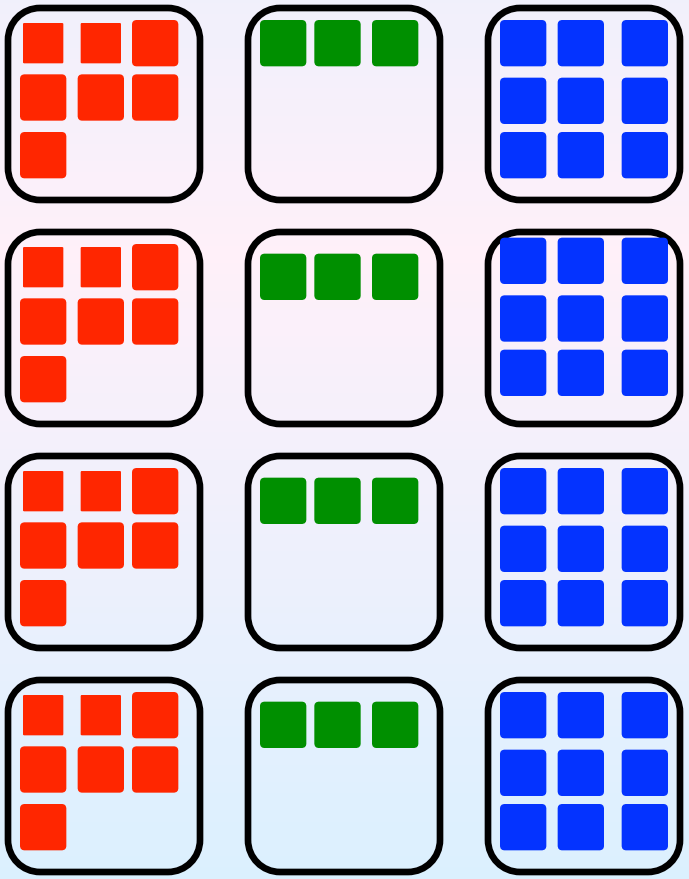
横方向に位置の情報をブロードキャストする

Active particle とj粒子の力(部分和)を計算
力の計算はSIMD化

部分和を足す

Ninjaアルゴリズム

独立時間刻みでも並列化効率が良くなるようにした並列化方法
全プロセッサーを2次元の配列にする（ここでは3x4）。
横方向に一旦粒子を分割、縦方向には同じデータが入る



それぞれの行で、7個、3個、9個の
Active particle が見つかる

それぞれのプロセッサーがどの粒子を担当するのか決める

横方向に位置の情報をブロードキャストする

Active particle とj粒子の力(部分和)を計算
力の計算はSIMD化

部分和を足す

縦方向に情報をブロードキャスト

Active 粒子とj 粒子を両方分割することで
計算能率を上げるアルゴリズム