N-body simulations: methods

BT2, §2.9

N-body codes

For general potentials, must use numerical codes to solve gravitational dynamics

<u>Collisional *N*-body:</u> evolve all $N_\star$ particles ($N_\star$ = # particles in system)

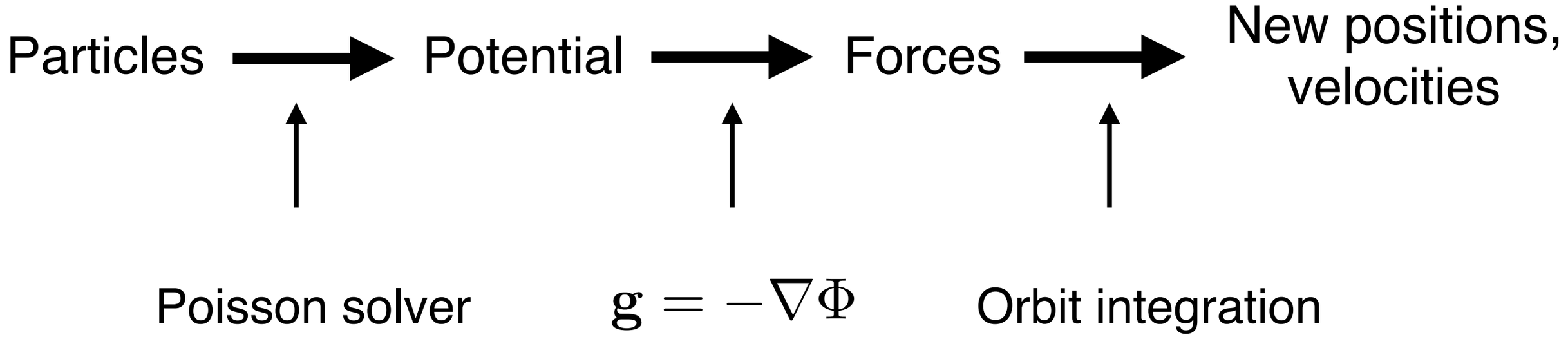
- in principle most accurate, but very expensive

<u>Collisionless *N*-body:</u> follow $N \ll N_\star$ particles (representing the total mass)

- a Monte Carlo (statistical) approach
- necessary when $N_\star$ is very large, e.g. galaxies ($N_\star \sim 10^{11}$), dark matter halos, cosmology
- results accurate when $t_{\rm relax}(N) \sim (0.1N/\ln N)\, t_{\rm cross} \gg$ age of system
- miss close encounters between stars (which can be important, e.g., in globulars)

Steps in *N*-body calculation

At each time step:



Repeat until system is evolved to desired time.

Poisson solvers: direct summation

Directly compute all forces (accelerations)

$$\mathbf{F}_\alpha = \sum_{\beta \neq \alpha} G m_\beta \frac{(\mathbf{r}_\beta - \mathbf{r}_\alpha)}{|\mathbf{r}_\beta - \mathbf{r}_\alpha|^3}$$

acceleration of particle α

Even using Newton's 3rd law ($\mathbf{F}_{ij} = -\mathbf{F}_{ji}$), requires $N(N-1)/2$ force evaluations, i.e. $O(N^2)$.

Largest direct summation simulations still limited to $N \lesssim 10^6$.

(e.g., N~500,000 globular cluster simulation of Heggie [2014] took 2 years and 8 months on 12 CPU cores + 2 GPUs)

Force softening

As $|\mathbf{r}_\beta - \mathbf{r}_\alpha| \to \mathbf{0}$, $|\mathbf{F}_\alpha| \to \infty$

$$t_{\rm dyn} \sim 1/\sqrt{G\rho} \to 0$$

(for accurate integration, need $\Delta t \ll t_{\rm dyn}$)

<u>For collisional systems</u>, this is physical (e.g., when two stars get very close), but can make the computation very slow

- “soften" force to speed up (lose accuracy)

<u>For collisionless systems</u>, divergence is artifact of $N \ll N_\bigstar$ (actual mass distribution is smoother).

- softening can enhance accuracy

Force softening math

Softened force: $$\mathbf{F}_\alpha = \sum_{\beta \neq \alpha} G m_\beta S_F(|\mathbf{r}_\beta - \mathbf{r}_\alpha|) \frac{\mathbf{r}_\beta - \mathbf{r}_\alpha}{|\mathbf{r}_\beta - \mathbf{r}_\alpha|}$$

force softening kernel,
→ r^{-2} for $r > \varepsilon$ = softening length

Softened potential: $$\Phi_\alpha = \sum_{\beta \neq \alpha} G m_\beta S(|\mathbf{r}_\beta - \mathbf{r}_\alpha|)$$

where $$S_F = \frac{dS(r)}{dr}$$

Common choice is Plummer: $$S(r) = -\frac{1}{\sqrt{r^2 + \epsilon^2}}$$

How to choose the softening length?

Many factors, no single recipe:

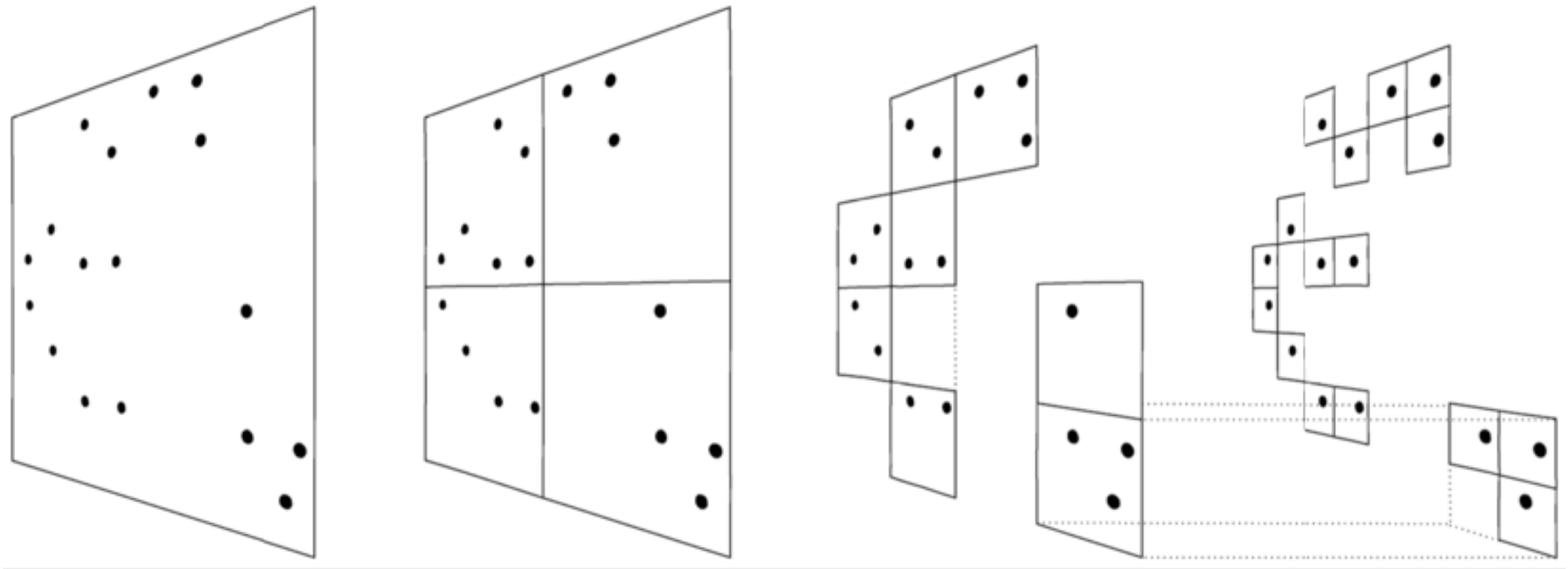
- smaller ε in principle allows the simulation to resolve finer structures
- but the simulation will be more expensive since structures can collapse to higher densities (shorter $t_{\rm dyn}$)
- cannot avoid limitations due to finite N
  - ε should be chosen such that, in regions of interest, each smoothing kernel contains a reasonable number of particles (say 32), otherwise the calculation will be affected by sampling noise
  - if ε is too small then the calculation is effectively not softened and collisional artifacts can be introduced when modeling a collisionless physical system

Common choice in galaxy simulations: $\varepsilon \sim 1/50$-$1/20$ mean particle separation

In general, a numerical convergence analysis is needed to check results.

Poisson solvers: tree method

Barnes & Hut (1986): organize particles in oct-tree



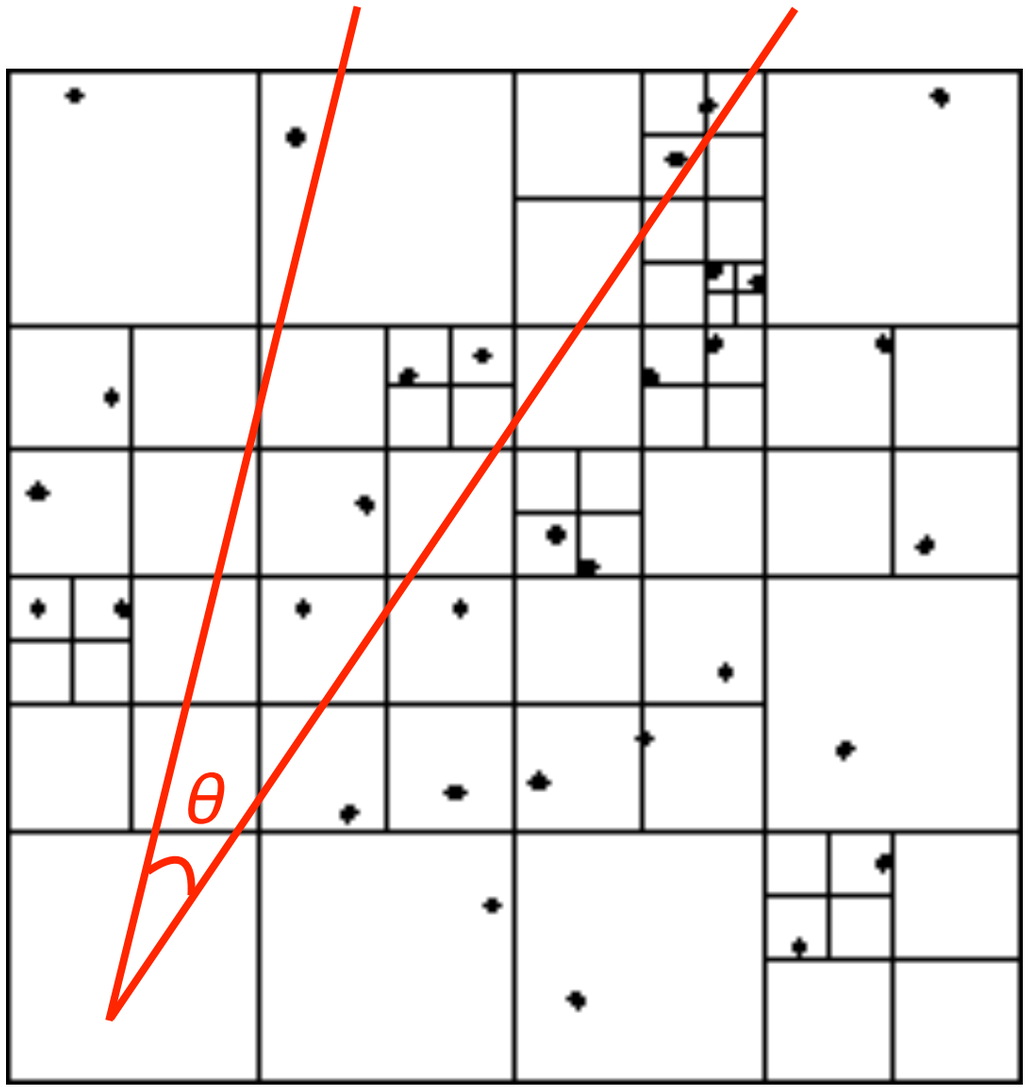
- recursively divide 3D space into cubes
- if cube contains >1 particles, divide into 8 identical cubes
- continue until tree ‘leaves’ each contain 1 particle

Fig credit: Springel+01

Evaluating the gravitational force using the tree

Define opening angle θ

Group contributions from distant particles into largest parent cube that fits within θ



- simplest: approximate cube by its center of mass
- more accurate: use a few Cartesian multipoles (math in BT2, §2.9.2)

Advantages of tree method

Much more efficient than direct summation for large *N:*

- *O(*ln *N)* to evaluate force on single particle, so *O(N* ln *N)* to evaluate force on all *N* particles
- constructing tree is also *O(N* ln *N)*

No grid:

- well suited for dense stellar systems moving through nearly empty space (e.g., galaxy mergers)

Animation credit: John Dubinksi

Poisson solvers: particle-mesh (PM) method

Interpolate particle mass onto grid (simplest: Cartesian)

Solve Poisson eq. using Fourier method to get potential on the grid:

$$\nabla^2\Phi = 4\pi G\rho \quad \longrightarrow \quad (ik)^2\hat{\Phi} = 4\pi G\hat{\rho}$$

$$\Rightarrow \ \hat{\Phi} = -\frac{4\pi G\hat{\rho}}{k^2}$$

where $\hat{f}(\mathbf{k}) = \int d^3\mathbf{x} f(\mathbf{x}) e^{-i\mathbf{k}\cdot\mathbf{x}}$

Using FFT algorithm, this can be done in *O(N* ln *N).*

In 1D, recall: $\mathcal{F}\left\{\frac{d^n f}{dx^n}\right\} = (ik)^n \hat{f}, \quad \hat{f} \equiv \mathcal{F}\{f\} = \int dx e^{-ikx} f(x)$

PM method (continued)

Once we have Φ on the grid, interpolate to compute force on each particle.

Advance particles using orbit integration algorithm.

<u>Advantages of PM method:</u>

- easy to implement
- for applications in which the mass is volume-filling, usually faster than tree (*O(N* ln *N)* pre-factor constant in smaller)
- simpler data structure than tree

<u>Disadvantages of PM method:</u>

- force approximation is anisotropic on the grid (a particle that should produce a spherically symmetric force will be affected by errors that pick out grid-aligned directions)

PM method: mass deposition schemes

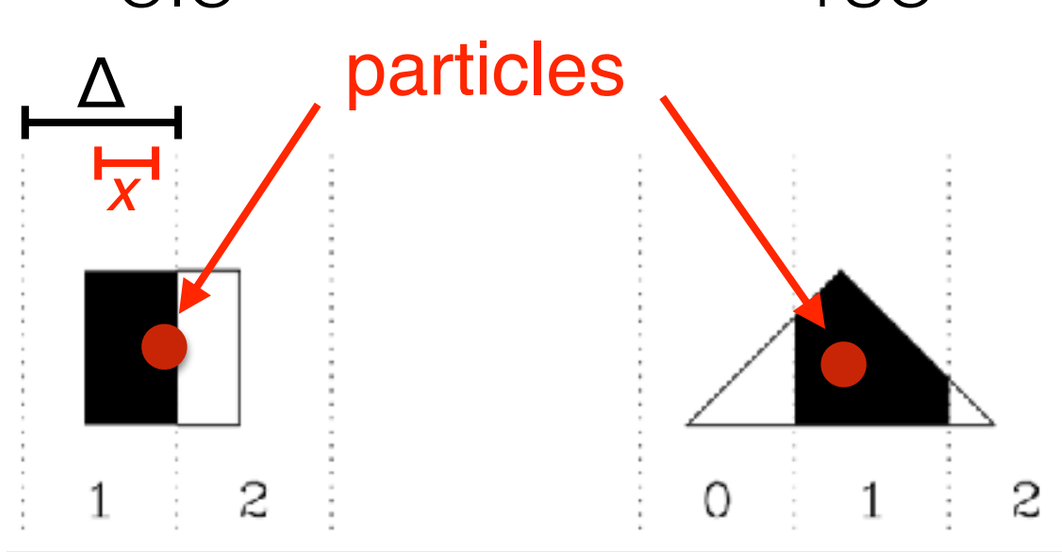
Nearest grid point (NGP): Particle mass all assigned to single cell that contains it. Rarely the method of choice: introduces artificial discontinuities in density field, forces.

Cloud in cell (CIC): Distribute particle mass between 2^D nearest grid nodes (D=# dims), using weights depending linearly on distance. Commonly used.

$$w(x) = \begin{cases} 1 - |x|/\Delta & \text{for } |x| < \Delta, \\ 0 & \text{otherwise} \end{cases}$$

Triangular shaped cloud (TSC): Higher-order scheme distributing particle mass between 3^D nearest grid nodes (D=# dims).

$$w(x) = \begin{cases} \frac{3}{4} - |x|^2/\Delta^2 & \text{for } |x| < \frac{1}{2}\Delta \\ \frac{1}{2}(\frac{3}{2} - |x|/\Delta)^2 & \text{for } \frac{1}{2}\Delta < |x| < \frac{3}{2}\Delta \\ 0 & \text{otherwise} \end{cases}$$



PM method: mass deposition in 3D

CIC example:

If $\Delta x = x_{\rm part} - x_{\rm grid}$ are offsets between Cartesian coordinates of a particle and a grid node,

$\Delta y = y_{\rm part} - y_{\rm grid}$

$\Delta z = z_{\rm part} - z_{\rm grid}$

then a fraction

$$w_{\rm part,grid} = w_{\rm CIC}(\Delta x) w_{\rm CIC}(\Delta y) w_{\rm CIC}(\Delta z)$$

of the particle's mass is assigned to that grid node.

2^D weights for a given particle always sum to 1, so the mass deposition scheme conserves mass.

PM method: computing accelerations

After using FFT to compute potential Φ at grid nodes, need accelerations.

First, accelerations ***at grid nodes***. Use numerical differentiation:

$$\hat{\mathbf{e}}_1 \cdot \mathbf{g_n} = -\frac{1}{2\Delta}(\Phi_{(n_1+1,n_2,n_3)} - \Phi_{(n_1-1,n_2,n_3)})$$

component of gravitational acceleration at node **n** along axis ‘1' (e.g., *x* axis)

minus gradient of potential using node neighbors

Then, interpolate the accelerations ***at the position of each particle***.

Important: BT2 §2.9.3b shows that ensuring net conversation of momentum requires using the **same scheme** to interpolate accelerations from the grid onto particles as is used to deposit mass from particles onto the grid (e.g., CIC).

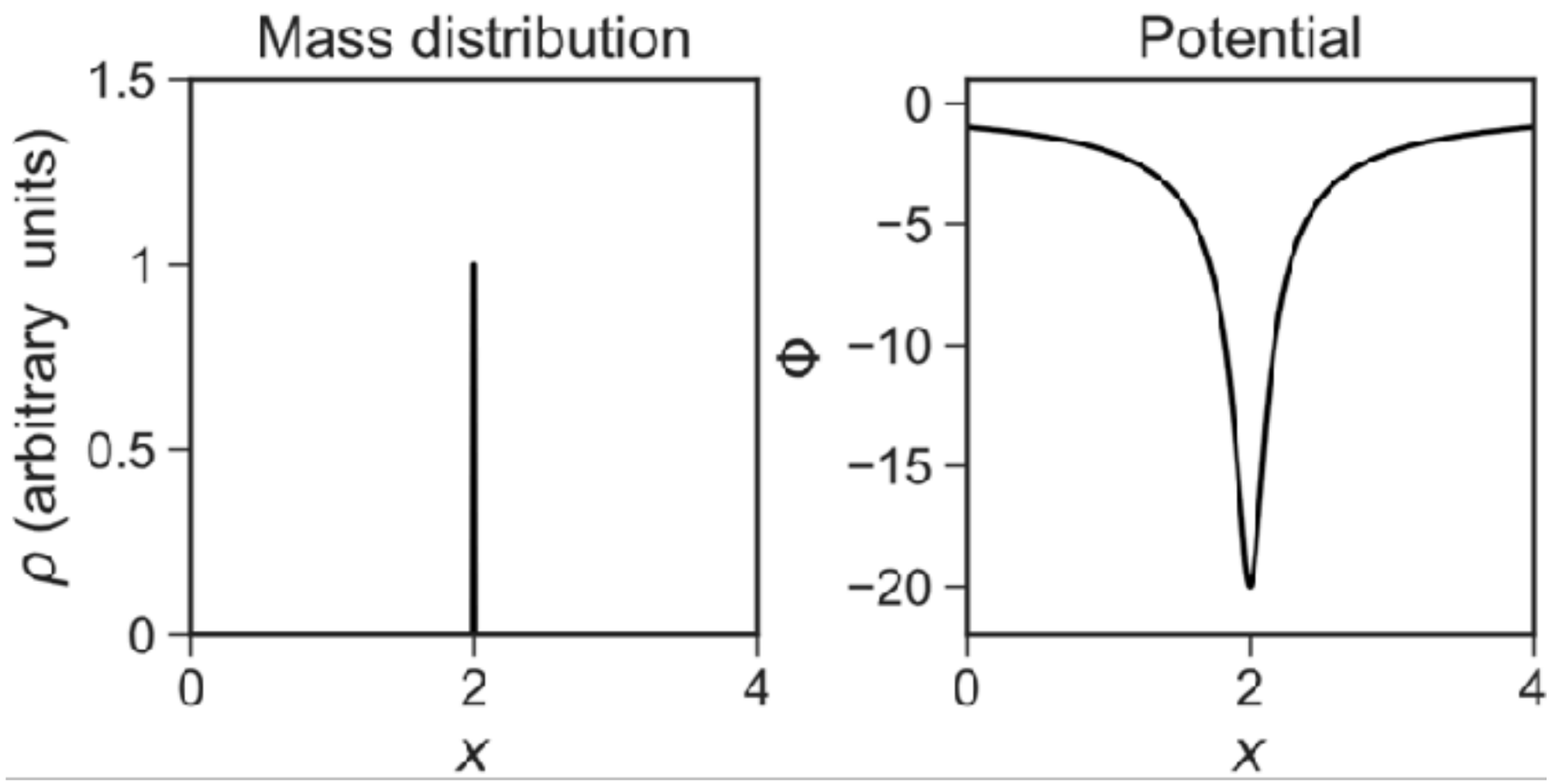
PM method: self-force issue

In a direct summation code, we are trivially guaranteed that the gravitational force on any given particle depends ***only on other particles*** in the system, i.e. particles do not exert any force on themselves.

$$\mathbf{F}_\alpha = \sum_{\beta \neq \alpha} G m_\beta S_F(|\mathbf{r}_\beta - \mathbf{r}_\alpha|) \frac{\mathbf{r}_\beta - \mathbf{r}_\alpha}{|\mathbf{r}_\beta - \mathbf{r}_\alpha|}$$

In a PM code, ***all particles are used*** to compute the gravitational potential everywhere on the grid.

minor coding errors can result in mass and potential grids being slightly offset, inducing spurious self-force — be extra carefully with array indices!



PM method: example of FFT in Python

Write script fft_example.py:

```
import numpy as np

# Make array representing some arbitrary field on a 1D grid.
x_arr = np.arange(0., 10., 0.01)
y_arr = x_arr**2. - np.exp(x_arr/5.)

# rfftn computes FFT of (in general multi-dimensional) array of real numbers.
yk_arr = np.fft.rfftn(y_arr)

# irfftn computes the inverse FFT.
y_arr_iFFT = np.fft.irfftn(yk_arr)

# Check difference between original array and the result of the FFT+inverse FFT.
print "Maximum difference=",
print max(y_arr-y_arr_iFFT)
```

Run script:

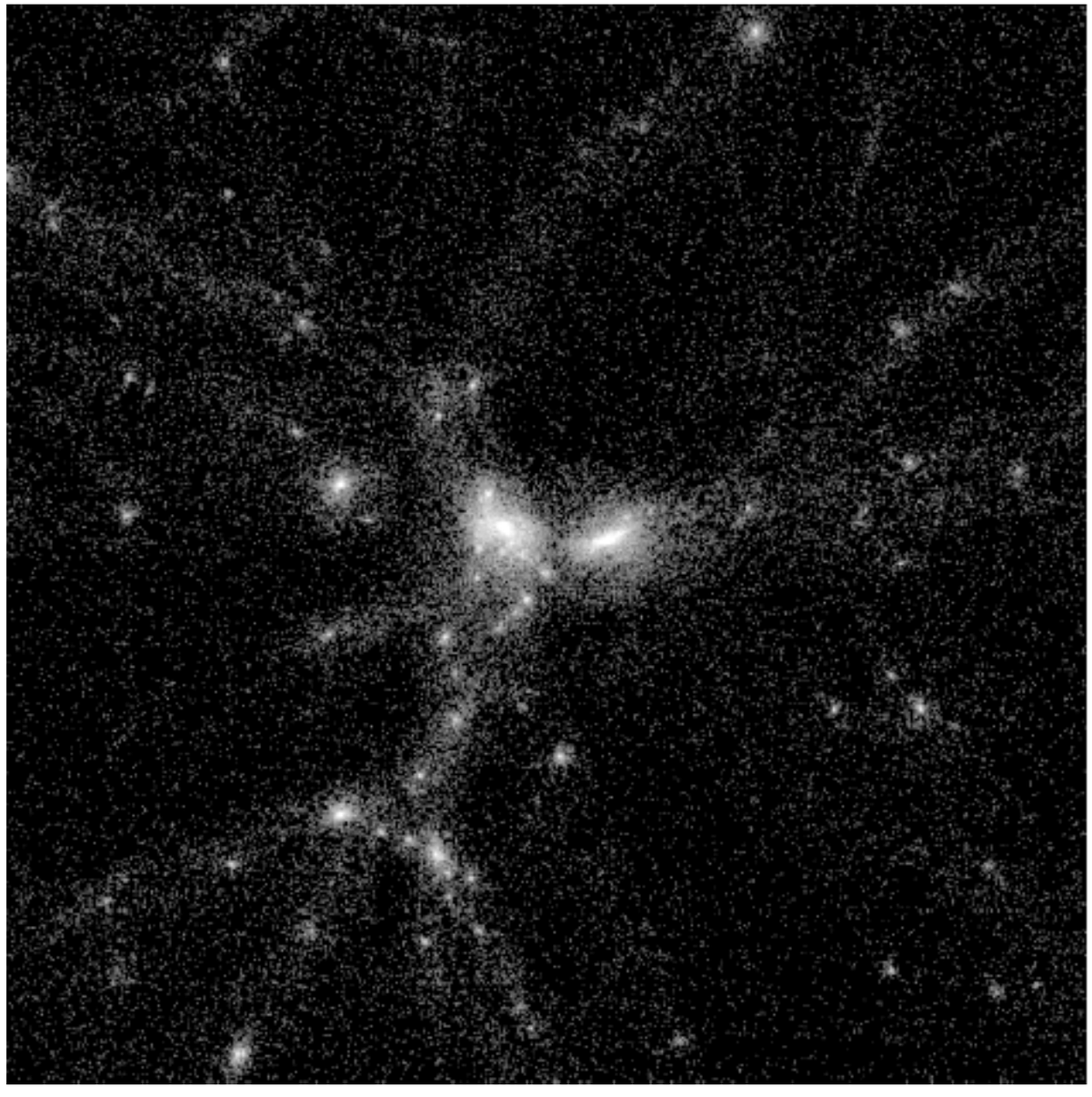
```
claudeandresmbp:code cgiguere$ python fft_example.py
Maximum difference= 4.796163466380676e-14
```

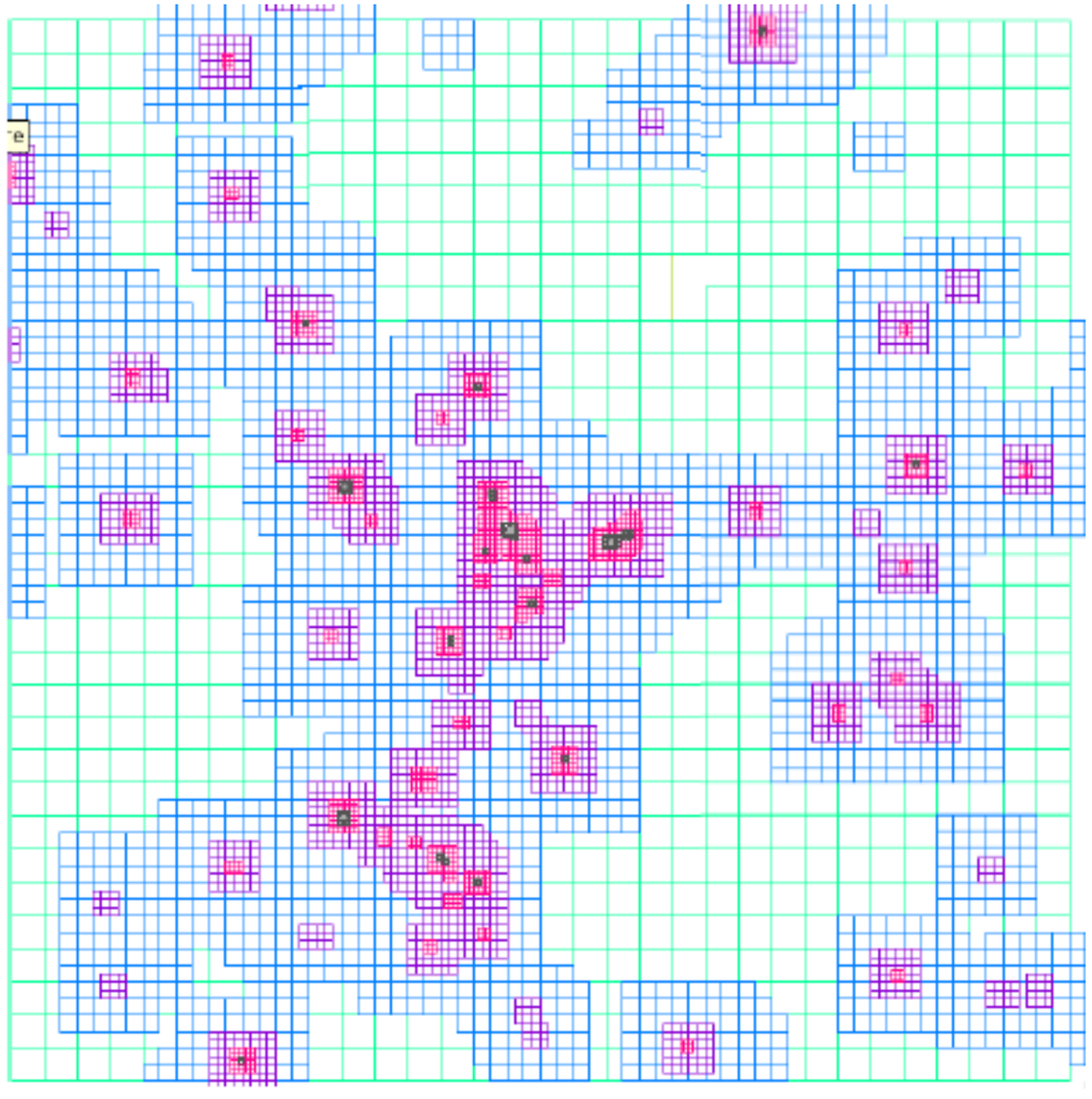
within machine precision of 0

See BT2 §2.9.3b for details on how to implement PM method with FFT and vacuum boundary conditions.

Adaptive mesh refinement

Accuracy of PM can be improved by using an adaptive grid.





Generally not as accurate as tree codes for gravity (grid not well matched to continuous particle orbits) but useful because hydro often solved on a grid, too.

Figure credit: Romain Teyssier

Hybrid methods

Modern codes typically implement hybrid schemes to optimize accuracy-performance balance.

Particle-particle particle-mesh (P3M):

- direct summation for nearby particles
- PM for distant particles

TreePM:

- tree for nearby particles
- PM for distant particles

→ commonly used for cosmological simulations, in which collisional effects are not important (so no need for direct summation), and in which distant gravitational interactions do not need to be computed as accurately as close ones (so PM is adequate)

Orbit integration

BT2, §3.4

Basic orbit integration considerations

Once Φ is known, need to advance particles accurately.

Two main approaches:

<u>High order integrators:</u>

- error grows with a high power of time step *h,* e.g. $O(h^4)$
- *h* is small, so error decreases rapidly with finer time steps

<u>Geometric integrators:</u>

- preserve some properties of the exact solution (e.g., energy, angular momentum, …)
- not necessarily high order, so can require shorter time steps to achieve same accuracy for modest-duration integrations
- but can be much more accurate for long-term integrations because they guarantee that integration does not diverge too much from a realistic solution

Euler integration

Simplest integration method:

$$\mathbf{x}_{i+1} = \mathbf{x}_i + \mathbf{v}_i h$$

$$\mathbf{v}_{i+1} = \mathbf{v}_i + \mathbf{a}_i h$$

$$\mathbf{a}_i = -\nabla \Phi_i$$

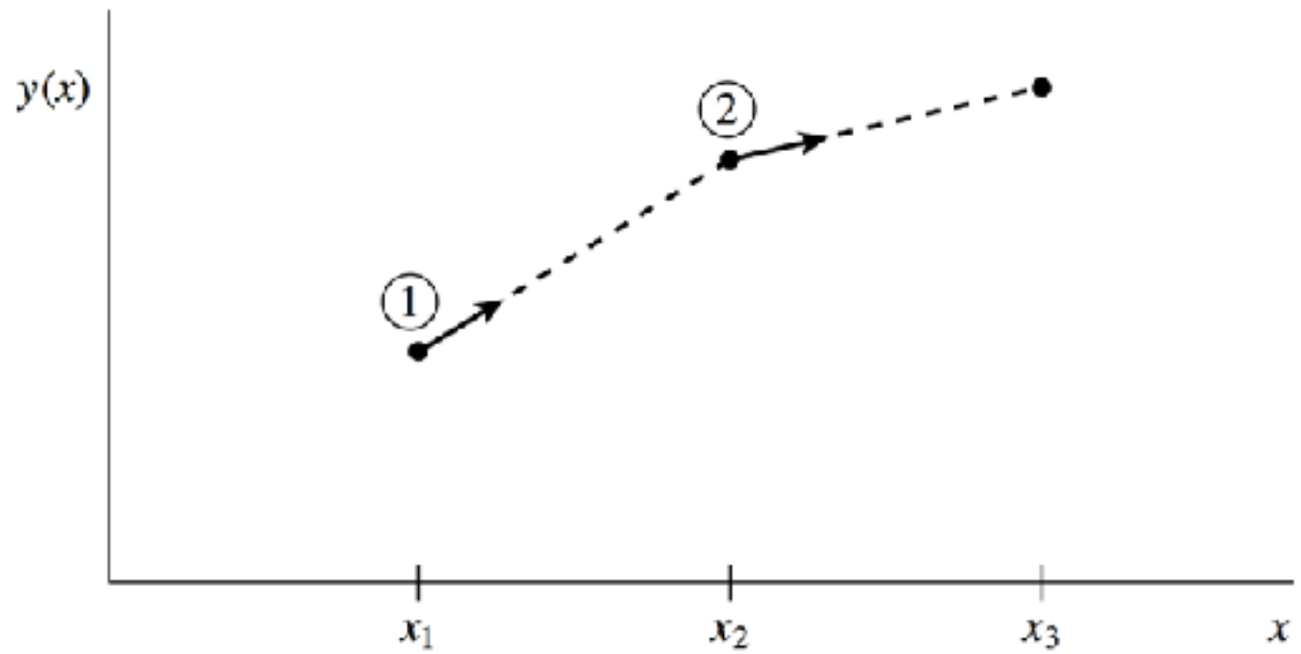
Exact solution would be (Taylor expansion):

$$\mathbf{x}_{i+1} = \underbrace{\mathbf{x}_i + \mathbf{v}_i h}_{\text{matches}} + \underbrace{\frac{1}{2}\mathbf{a}_i h^2}_{\substack{\text{does not} \\ \text{match}}} + \ldots$$

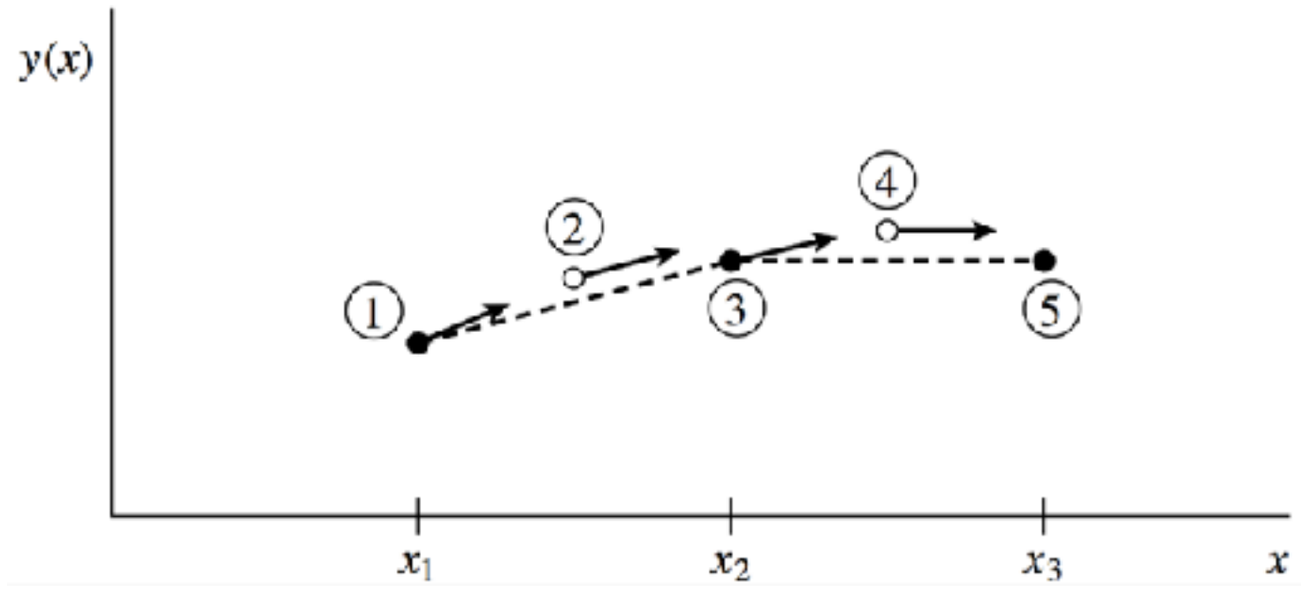
Error is $O(h^2)$, so the method is only first-order accurate.

Using midpoint for integration

In Euler, derivative at the starting point is extrapolated to find the next function value.



More accurate (2nd order) is to use the derivative at the midpoint of each interval.



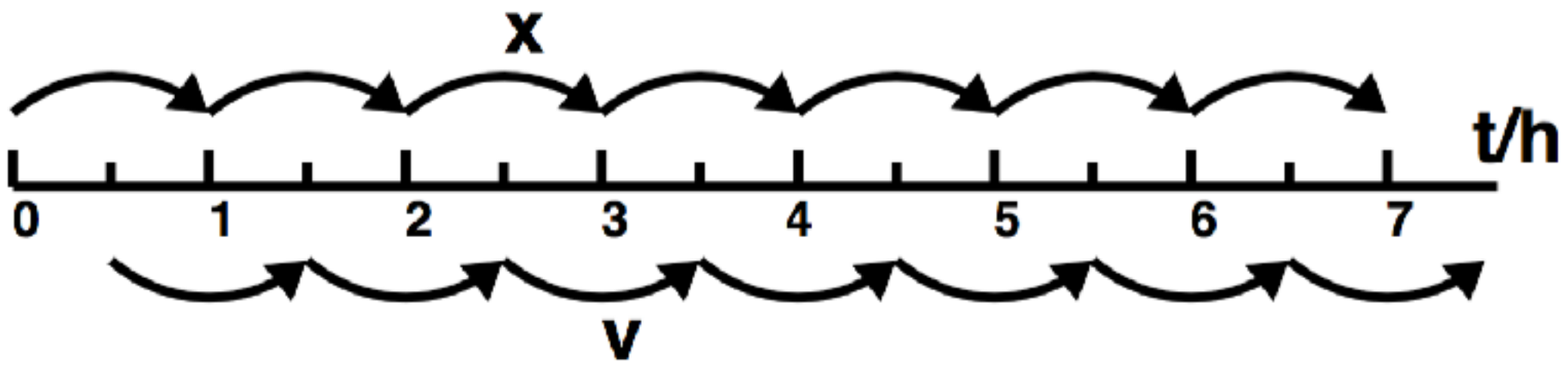
Figures from Numerical Recipes

Leapfrog integration

Better method: $\mathbf{x}_{i+1} = \mathbf{x}_i + \mathbf{v}_{i+1/2} h$ drift

$$\mathbf{v}_{i+1/2} = \mathbf{v}_{i-1/2} + \mathbf{a}_i h$$ kick

$$\mathbf{v}_{1/2} = \mathbf{v}_0 + \mathbf{a}_0 \left(\frac{1}{2} h \right)$$

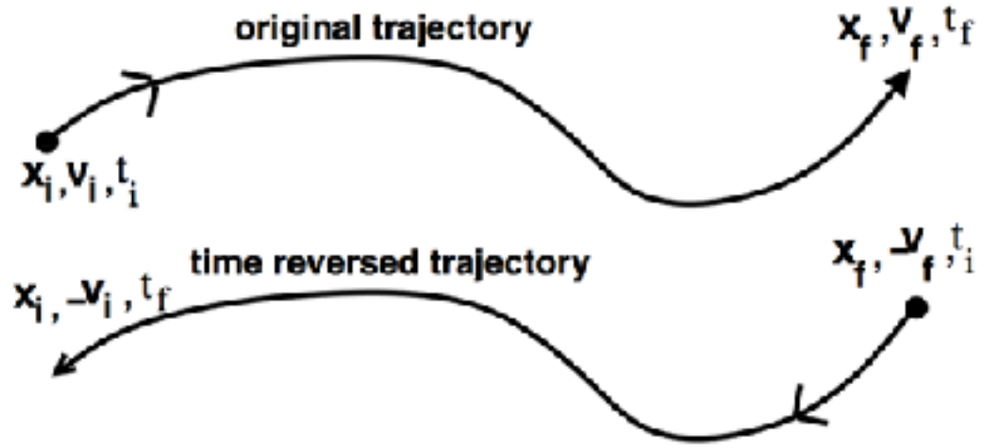


Second-order accurate.

Desirable properties of leapfrog

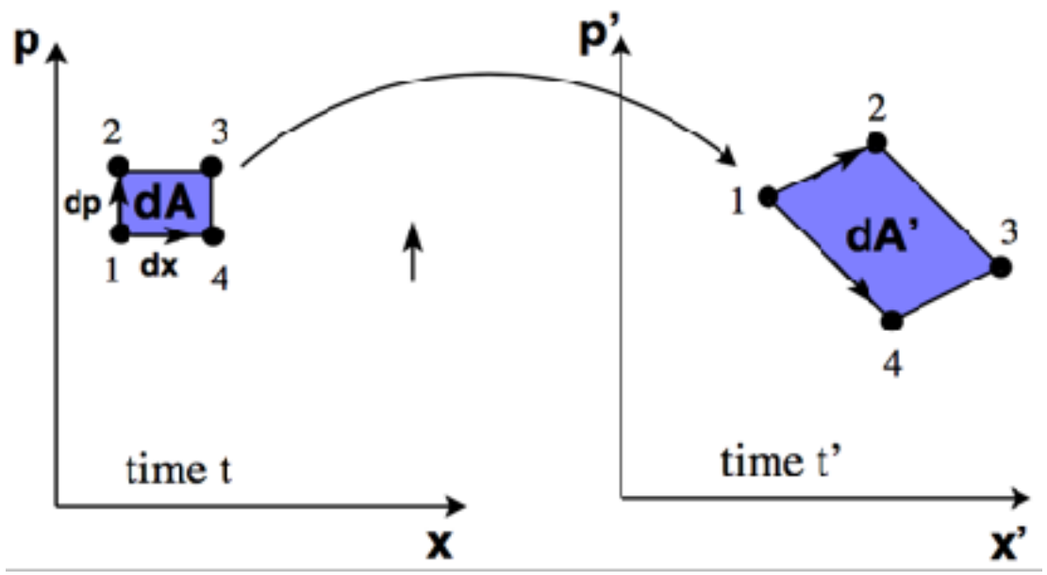
Leapfrog obeys several relations satisfied by exact solutions of the Newtonian EOMs:

1) Time reversible



2) In spherically-symmetric potential, conserves angular momentum exactly

3) Preserves phase-space density (symplectic)



⟶ <u>global stability</u>

see Peter Young's notes linked on web for more details

Fourth-order Runge-Kutta

For general ODE of the form $\dot{\mathbf{y}} = f(t,\ \mathbf{y})$, $\mathbf{y}(t_0) = \mathbf{y}_0$:

$$\mathbf{y}_{i+1} = \mathbf{y}_i + \underbrace{\frac{h}{6}\left(k_1 + 2k_2 + 2k_3 + k_4\right)} + O(h^5)$$

weighted average of 4 estimates of increment,
constructed to yield 4th-order accuracy

$$k_1 = f(t_i,\ \mathbf{y_i})$$ derivative at starting point

$$k_2 = f\left(t_i + \frac{h}{2},\ \mathbf{y_i} + \frac{h}{2}k_1\right)$$ use to estimate derivative at midpoint

$$k_3 = f\left(t_i + \frac{h}{2},\ \mathbf{y_i} + \frac{h}{2}k_2\right)$$ a different derivative at midpoint

$$k_4 = f(t_i + h,\ \mathbf{y_i} + hk_3)$$ use to estimate derivative at endpoint

Fourth-order Runge-Kutta (continued)

For a gravitational system: $\mathbf{y} = [\mathbf{x},\ \mathbf{v}]$

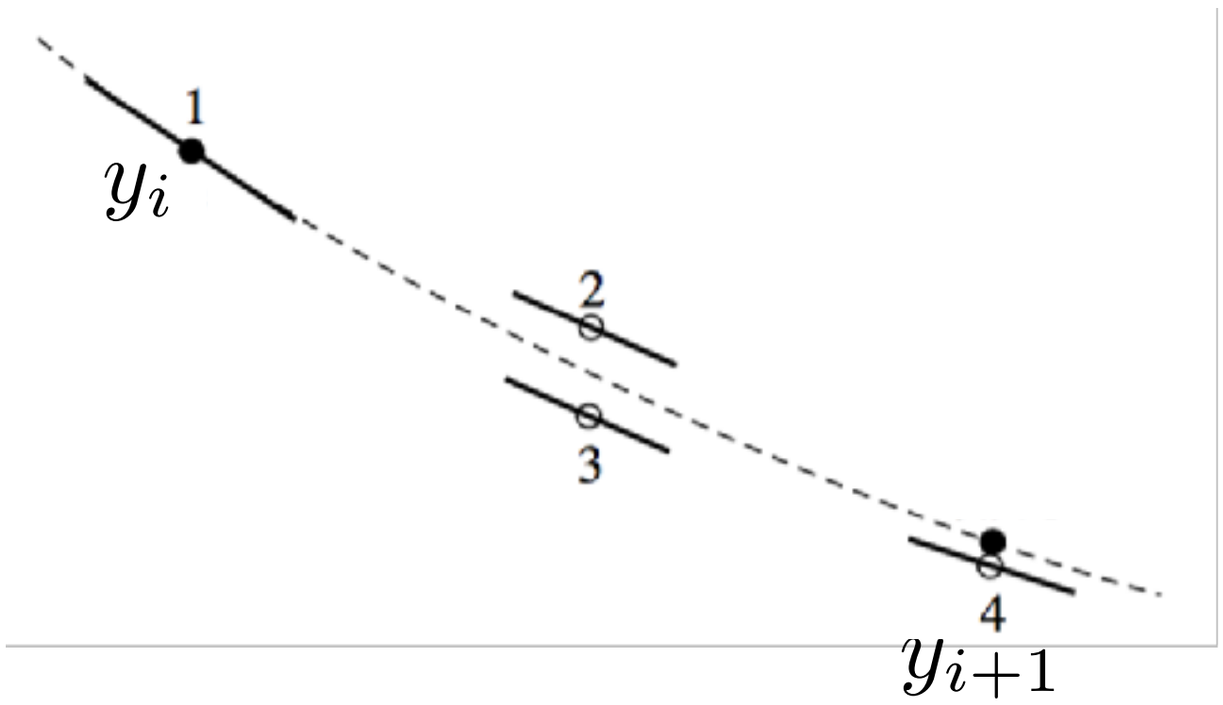
$$\dot{\mathbf{y}} = [\mathbf{v},\ \mathbf{a}] = [\mathbf{v},\ -\nabla\Phi] = f(t,\ \mathbf{y})$$

Since higher order, more accurate than leapfrog for short integrations.

But not symplectic or time-reversible, so errors (e.g., in energy) increase without bound over time.

Four force evaluations per time step, so increased accuracy relative to lower-order methods comes at computational cost.

Schematic of derivative estimates used in 4th order R-K



Comparison of different integrators

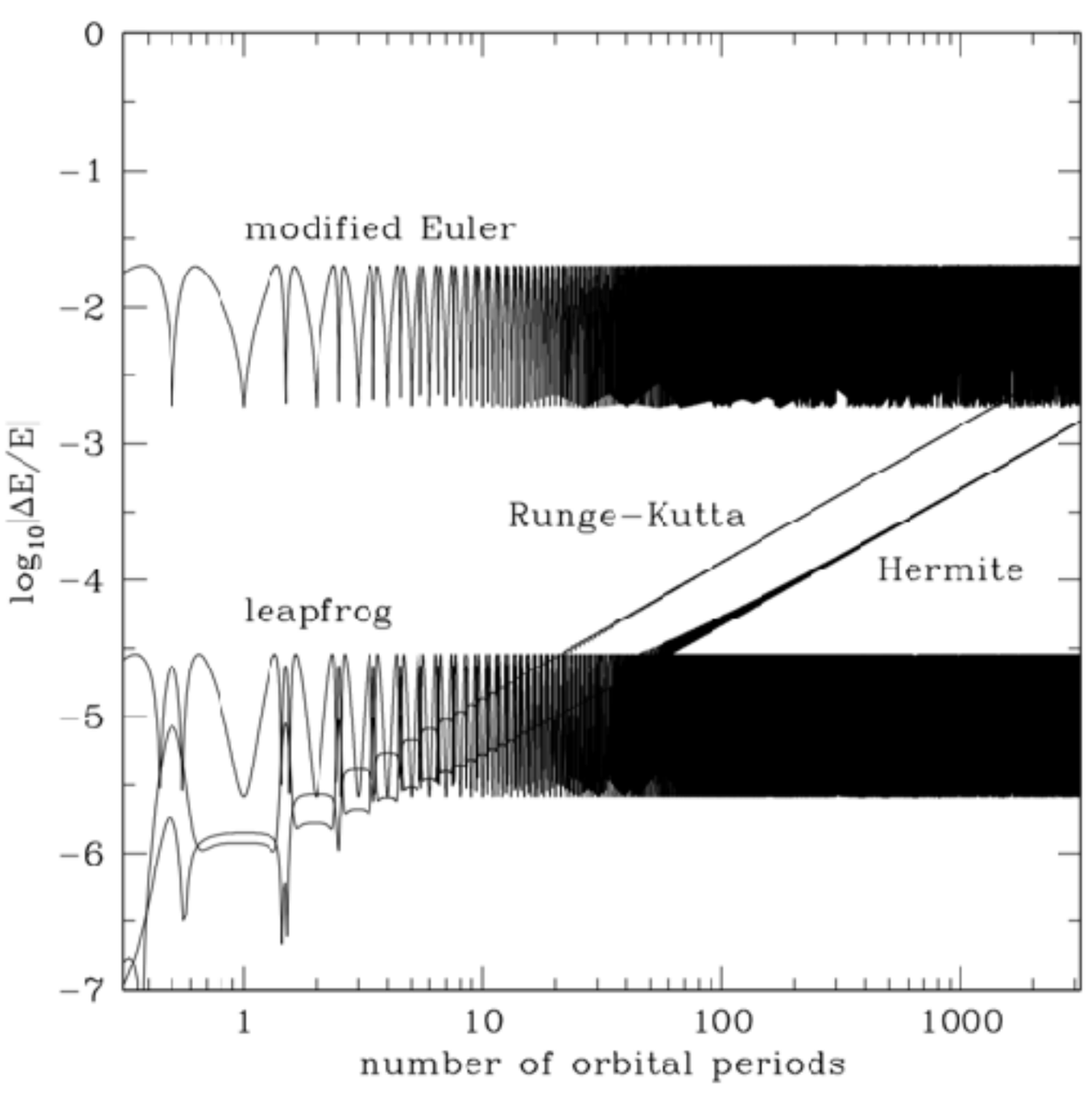
Fractional energy error vs. time for orbit in logarithmic potential, $\Phi(r) = \ln r$

R-K and Hermite are 4th order but not symplectic

- most accurate at first but energy error diverges in long term

Leapfrog and modified Euler are symplectic but only 2nd order

- energy error is bounded



see figure 3.21 in BT2 for more

Hydro simulations: overview of methods

Inviscid fluid equations with gravity

$$\frac{\partial \rho}{\partial t} + \nabla \cdot (\rho \mathbf{v}) = 0$$ mass continuity

$$\frac{\partial \mathbf{v}}{\partial t} + (\mathbf{v} \cdot \nabla)\mathbf{v} = -\nabla\Phi - \frac{1}{\rho}\nabla p$$ Euler (momentum)

$$\nabla^2 \Phi = 4\pi G\rho$$ Poisson

$$\frac{d}{dt} = \frac{\partial}{\partial t} + \mathbf{v} \cdot \nabla$$ Lagrangian or convective derivative

Plus: equation of state describing relation between pressure and density

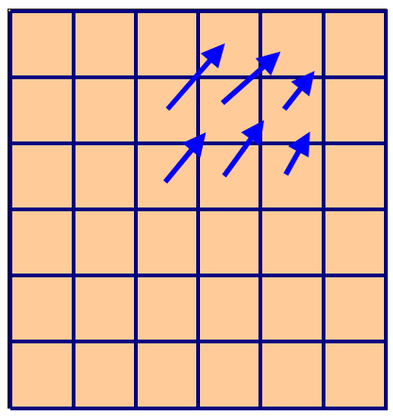
Optionally: energy equation to model gas cooling (affects pressure)

Hydro solvers: classic methods

Eulerian

discretize space

representation on a mesh
(volume elements)



<u>principle advantage:</u>

high accuracy (shock capturing), low numerical viscosity

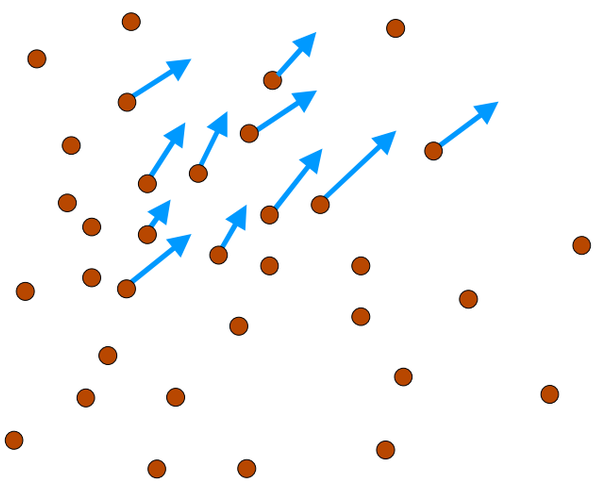
grid-based Godunov schemes

Athena, ENZO, RAMSES, …

Lagrangian

discretize mass

representation by fluid elements
(particles)



<u>principle advantage:</u>

resolutions adjusts automatically to the flow

smooth particle hydrodynamics (SPH)

GADGET, GIZMO, Gasoline, …

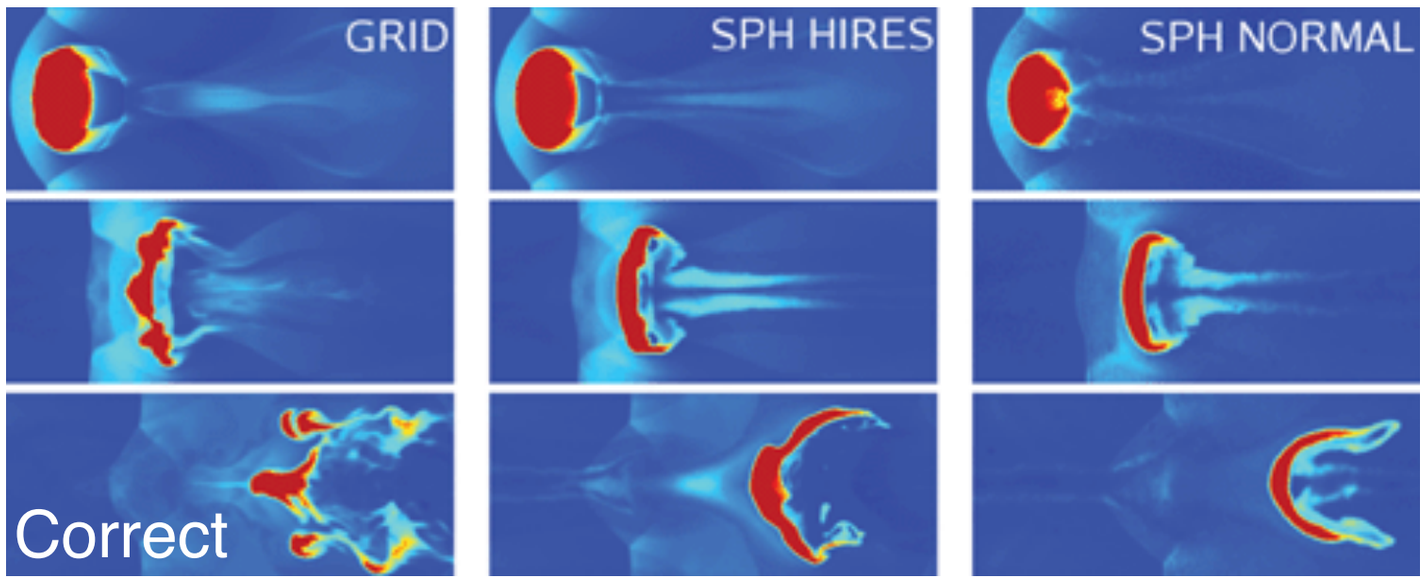
Advantages of grid-based hydro methods

Generally more accurate for pure hydro problems

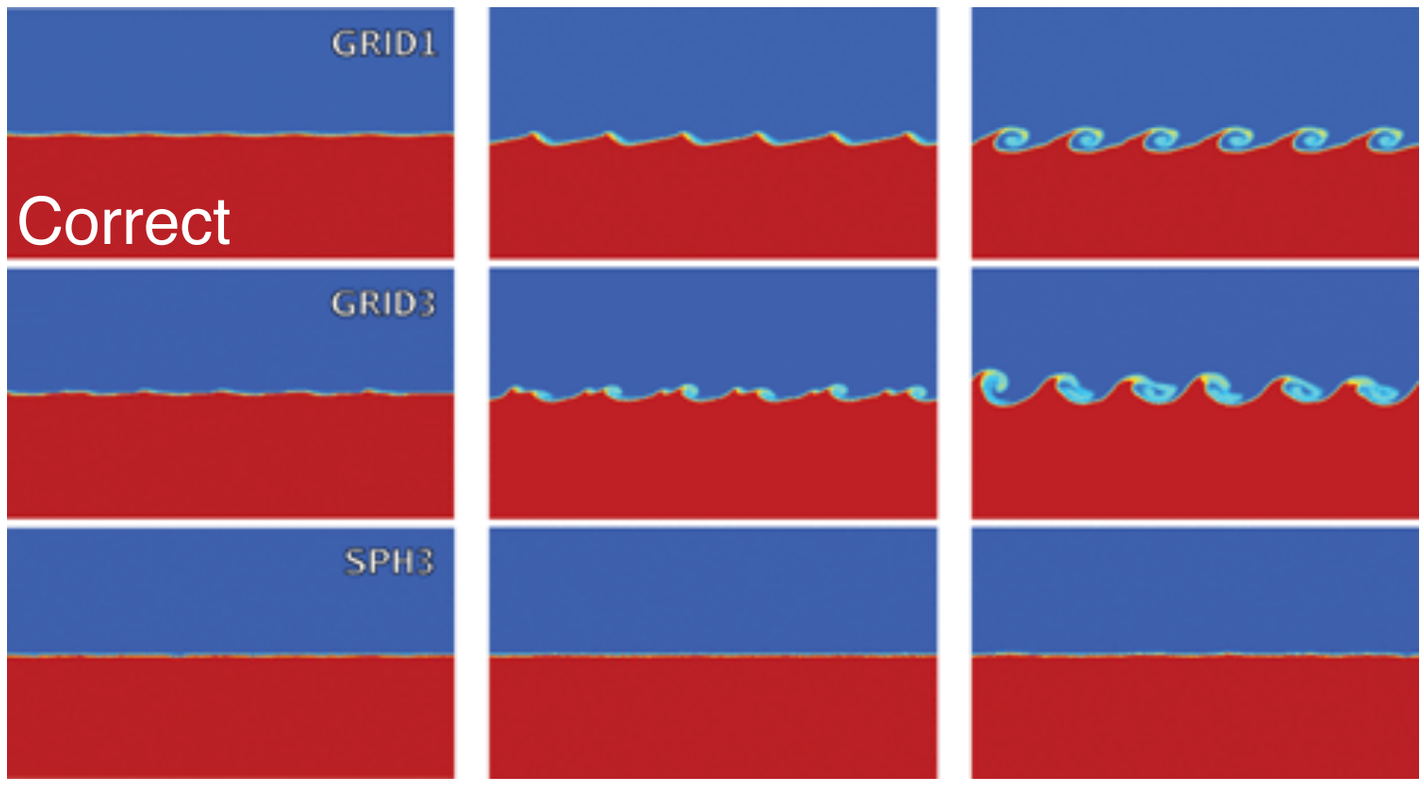
Traditional SPH schemes are known to suppress the development of fluid mixing instabilities owing to artificial surface tension at contact discontinuities (*P* continuous but discontinuous ρ and *T*)

Grid codes can also generally resolve shock discontinuities with fewer resolution elements

"Blob" test



Kelvin-Helmholtz instability test



Agertz+07

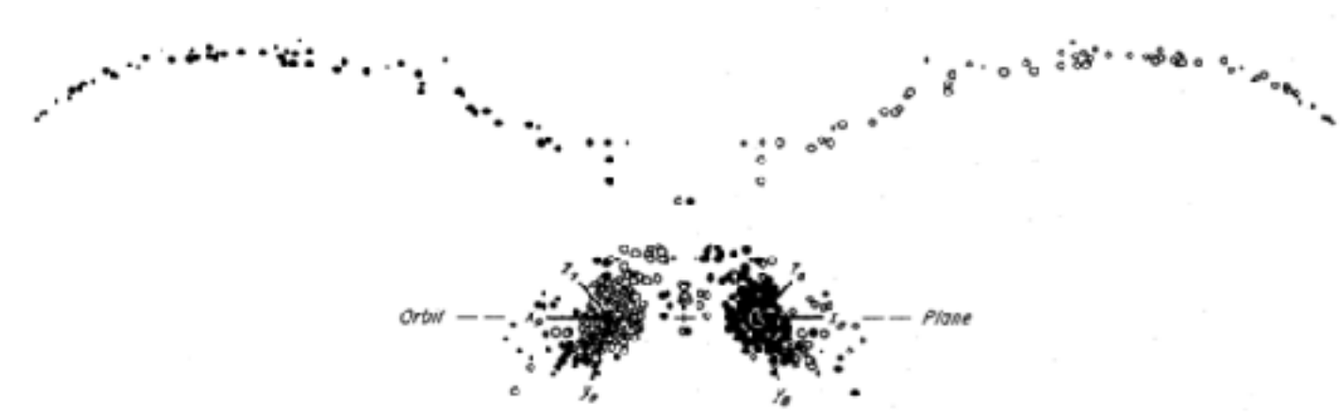
Advantages of particle-based hydro methods

Couple more accurately to gravity solvers (no grid artifacts), so can be preferable when gravitational forces are most important in the problem

e.g., tidal features in galaxy collisions primarily shaped by gravity

SPH widely used for:
- stellar collisions
- galaxy mergers
- cosmology

Antennae galaxies (NGC 4308/4309)



Toomre & Toomre 72
central point masses+test particles model

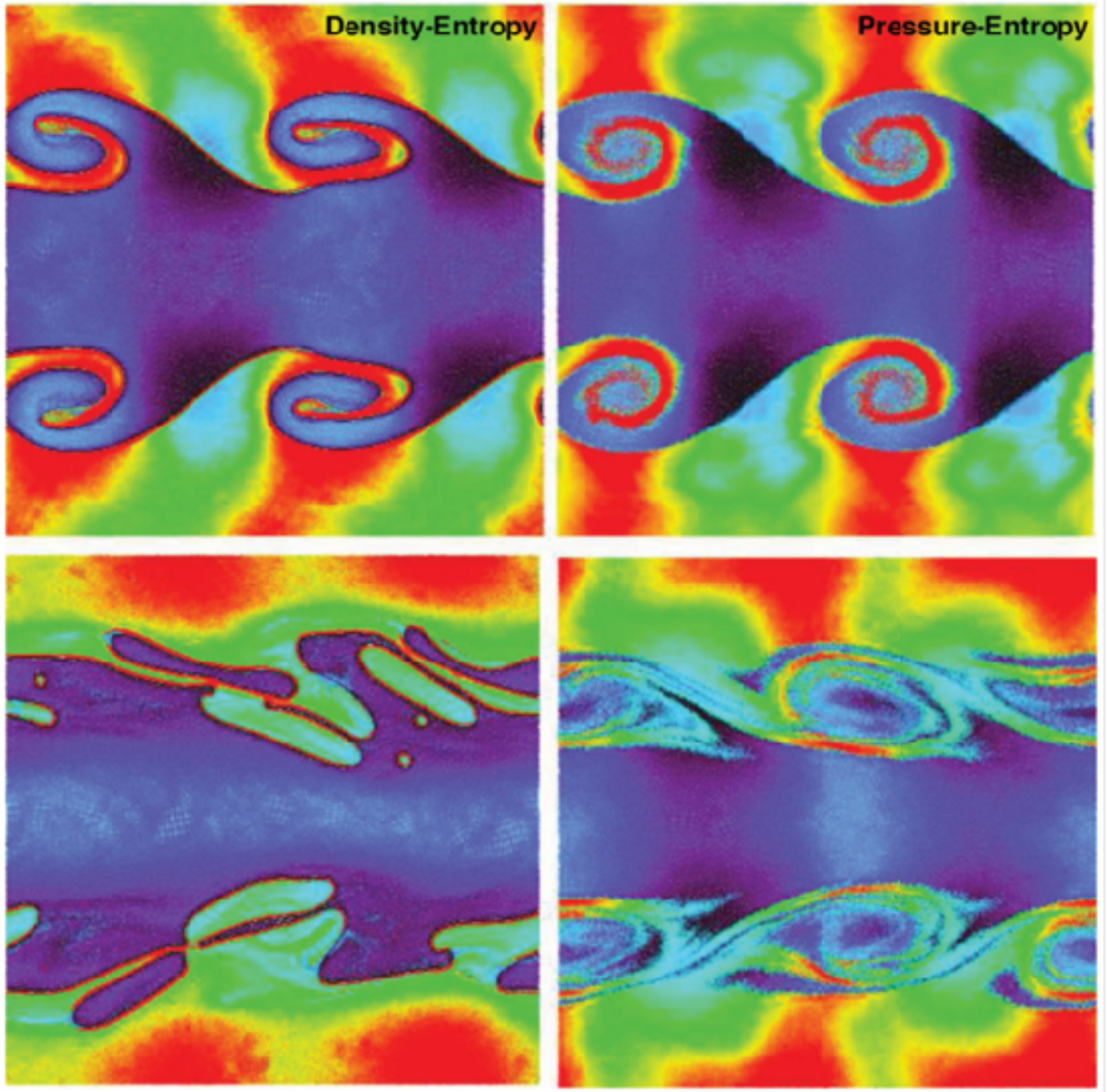
Modern SPH

Most recent SPH simulations use “improved" or “modern" SPH schemes (there are many approaches) that resolve the primary historical discrepancies between grid and SPH codes.

They do so primarily by eliminating artificial surface tension at contact discontinuities.

Traditional (ρ-based) SPH suppresses K-H

Modern (P-based) SPH more accurately follows K-H



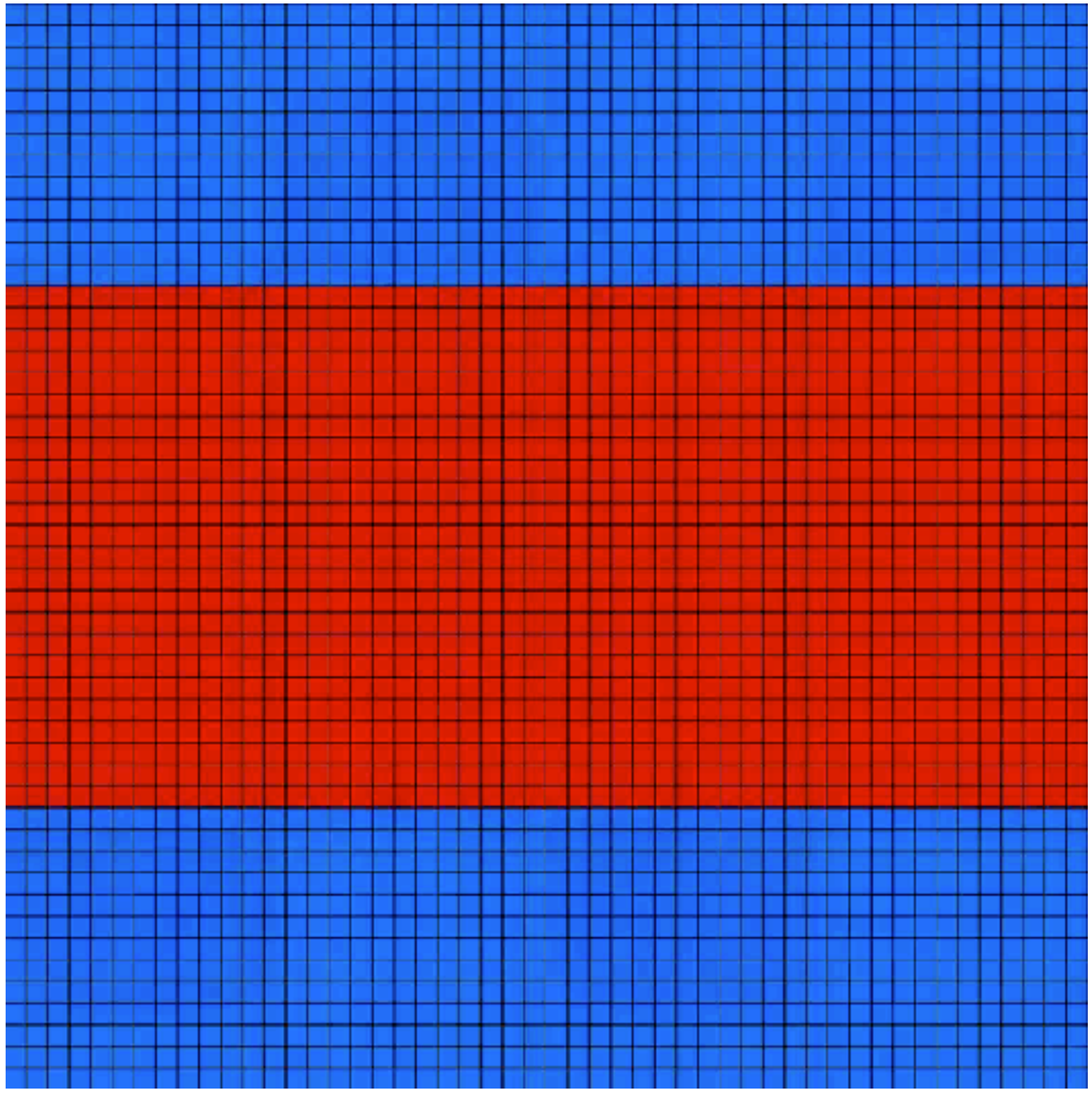
(Relatively) new hybrid hydro methods

<u>Moving mesh:</u>

- like SPH, can be more accurately coupled to gravity
- like fixed grids, more accurate for shocks and fluid mixing instabilities (use same methods to compute fluxes between cells)

<u>Meshless versions:</u>

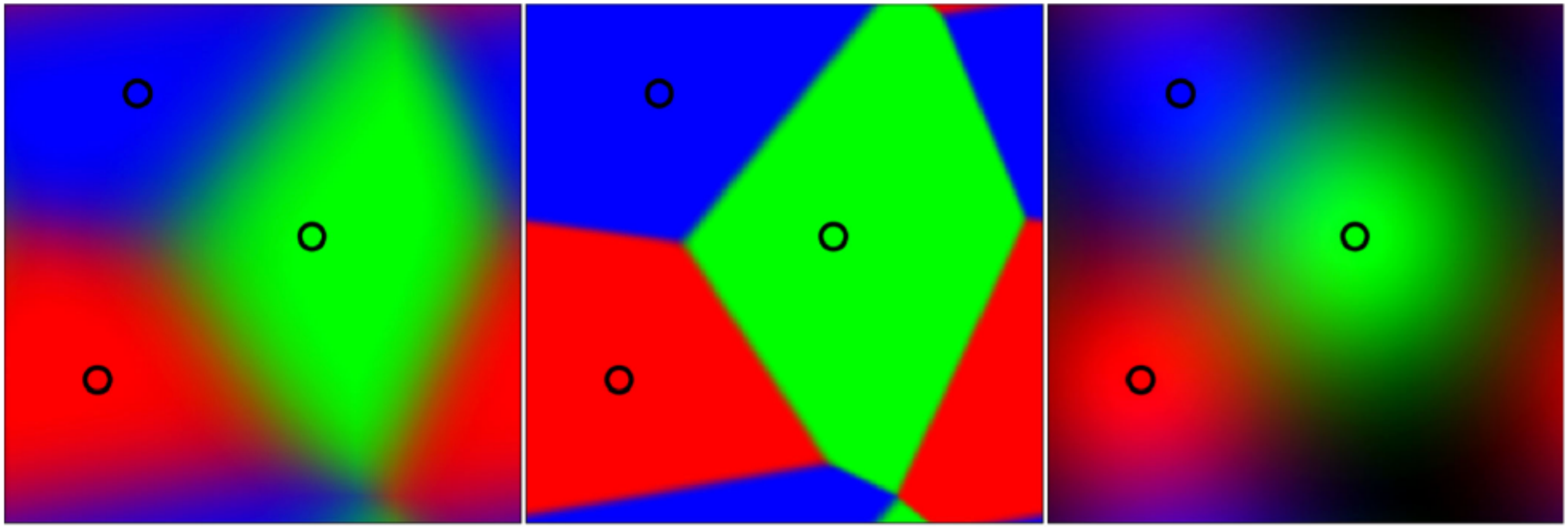
- can be generalized such that well-defined cells boundaries need not be tracked (“fuzzy” cells)
- e.g., the new "meshless finite mass" (MFM) method



Springel 10

Caveat: since relatively new, limitations not yet fully understood (e.g., effects of numerical noise associated with advecting the mesh)

Extra slides



New Meshless Methods Here (MFV, MFM) | Unstructured / Moving−Mesh Methods | Smoothed−Particle Hydrodynamics