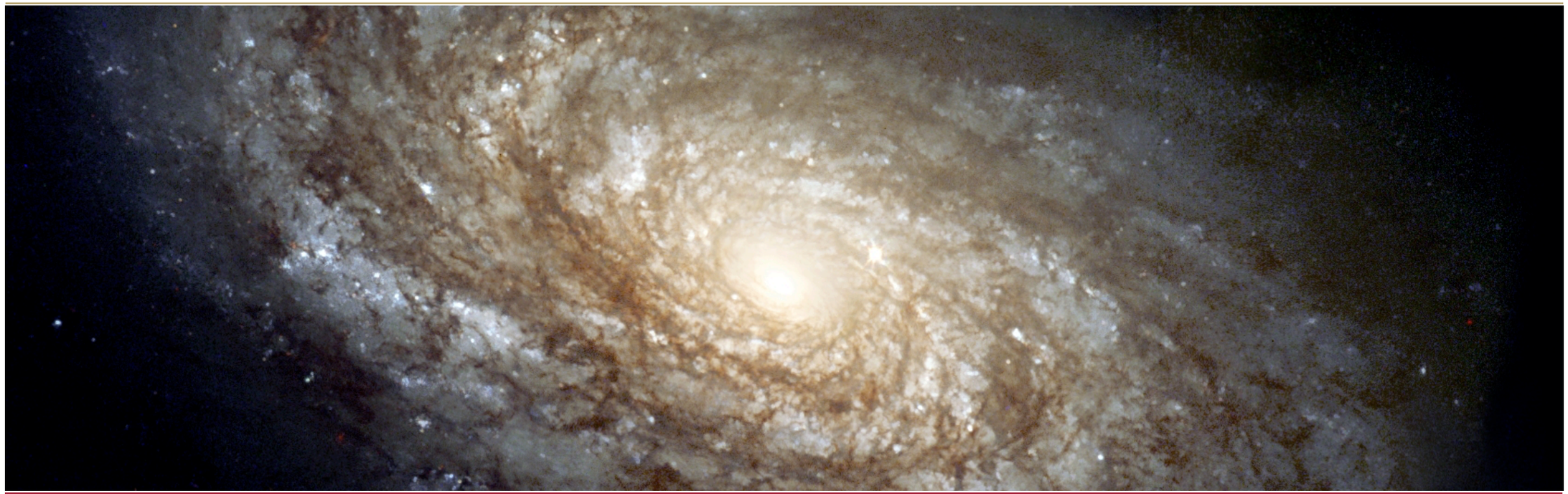
EBERHARD KARLS
UNIVERSITÄT
TÜBINGEN

FACULTY OF
SCIENCE
Institute of Astronomy and Astrophysics



N-Body modelling

Crash Course on Numerical Astrophysics
July 27-31, 2020

28 July 2020 · Christoph Schäfer

Ferdowsi university
of mashhad

Topics

- Theory part
  - The classical astrophysical N-body problem
  - Exact N-body schemes and time integrators
  - Tree algorithms - Barnes & Hut Tree
  - REBOUND integrator package
    by Hanno Rein

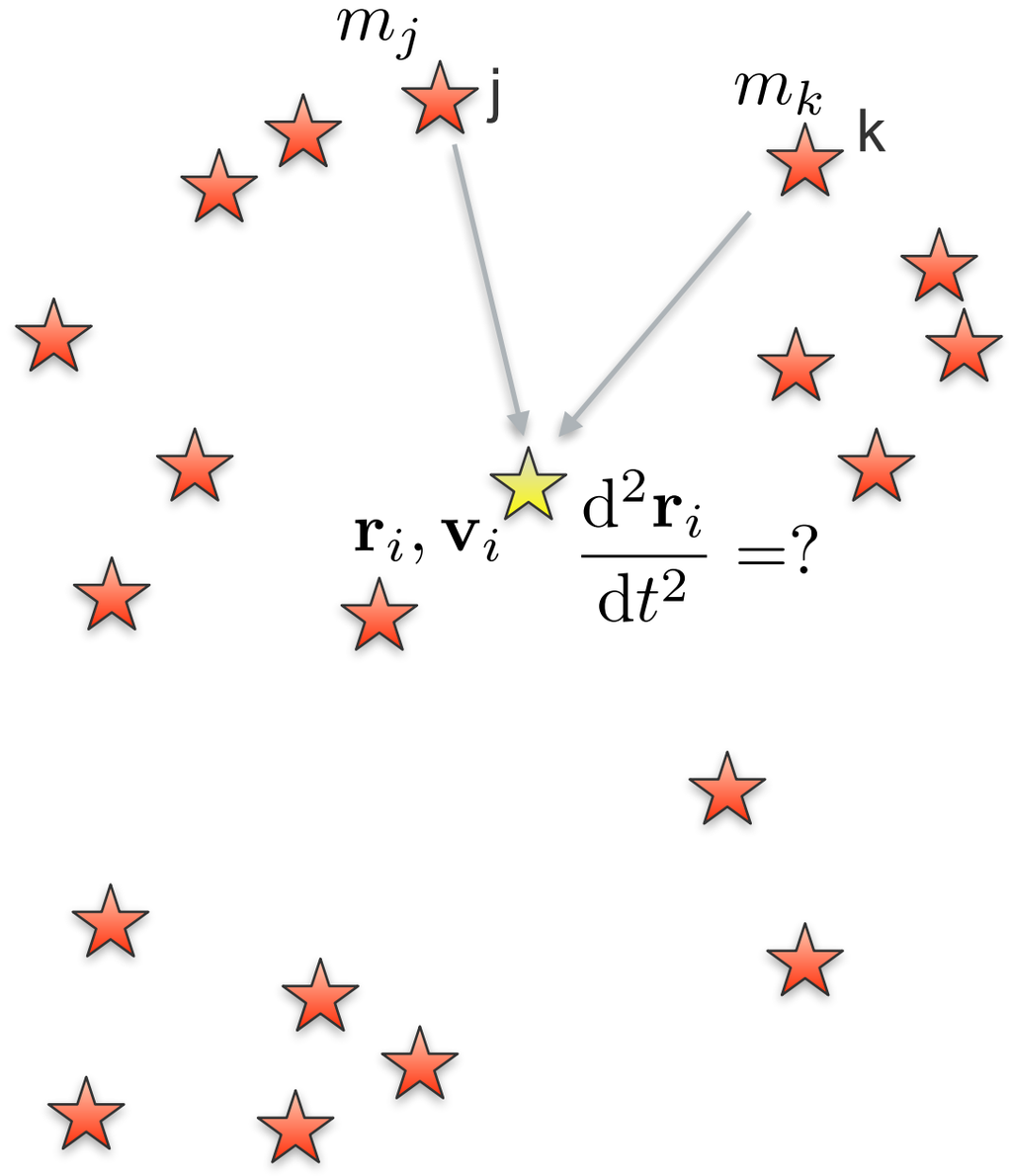
- Hands-on exercises part
  - Two-Body problem
  - Few-body problem
  - Saturn's rings stability
  - Kirkwood gaps

Goals:

- Learn some basics about time integrators
- Get familiar with REBOUND

The classical astrophysical N-body problem

A number of N particles interact classically through Newton's Laws of Motion and Newton's Law of Gravitation.

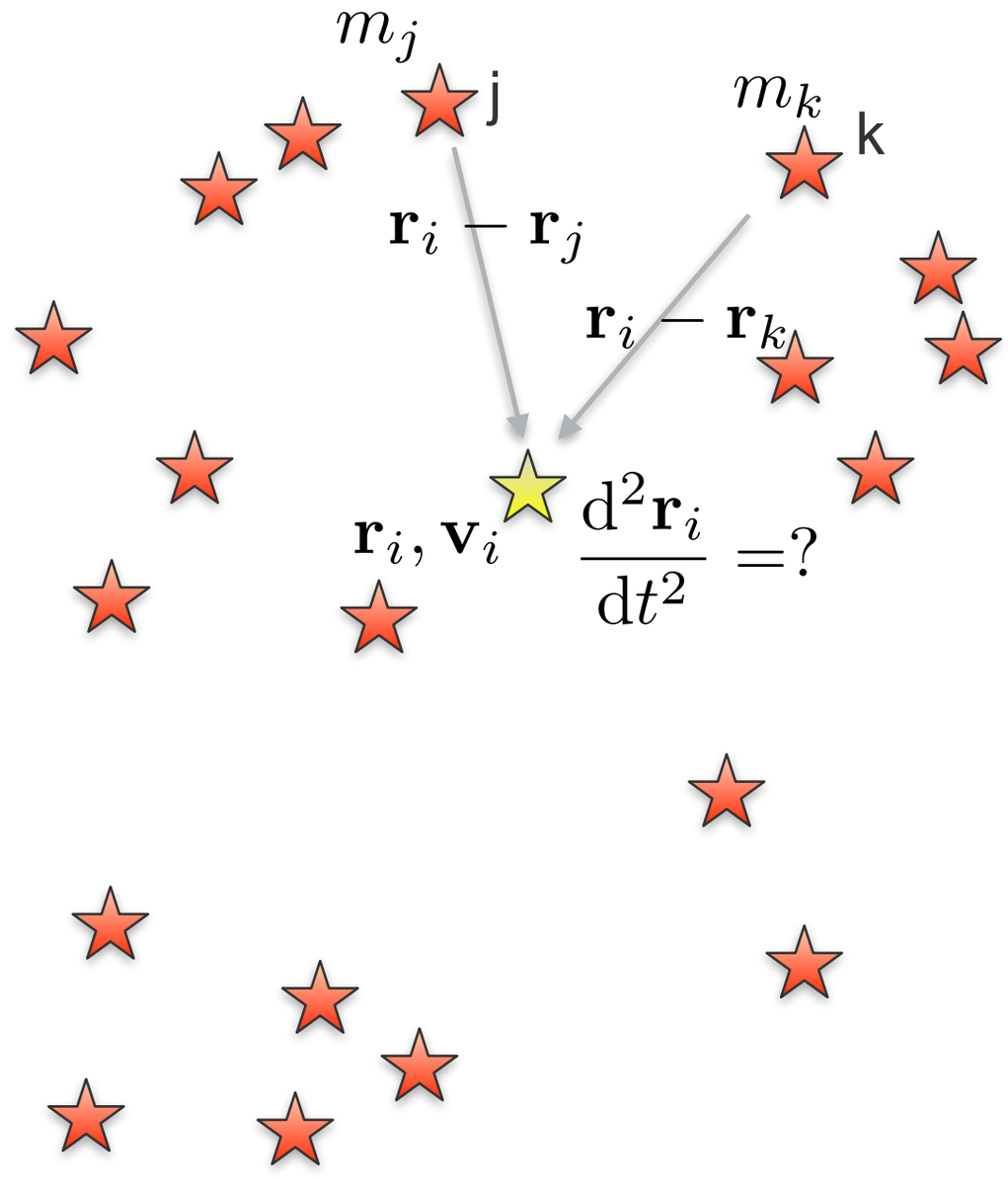


For N=1 and N=2, the equation of motion can be solved analytically.

The classical astrophysical N-body problem

A number of N particles interact classically through Newton's Laws of Motion and Newton's Law of Gravitation.

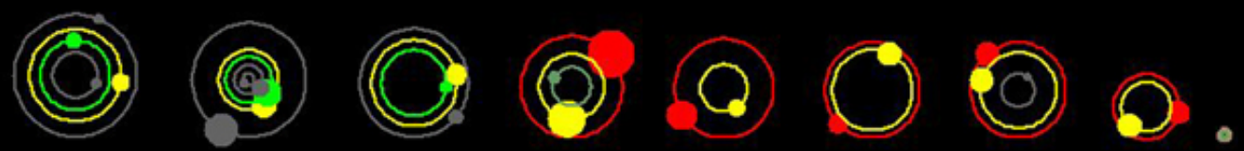
$$\frac{\mathrm{d}^2\mathbf{r}_i}{\mathrm{d}t^2} = -G\sum_{j\neq i}^{N} m_j \frac{\mathbf{r}_i - \mathbf{r}_j}{|\mathbf{r}_i - \mathbf{r}_j|^3}$$

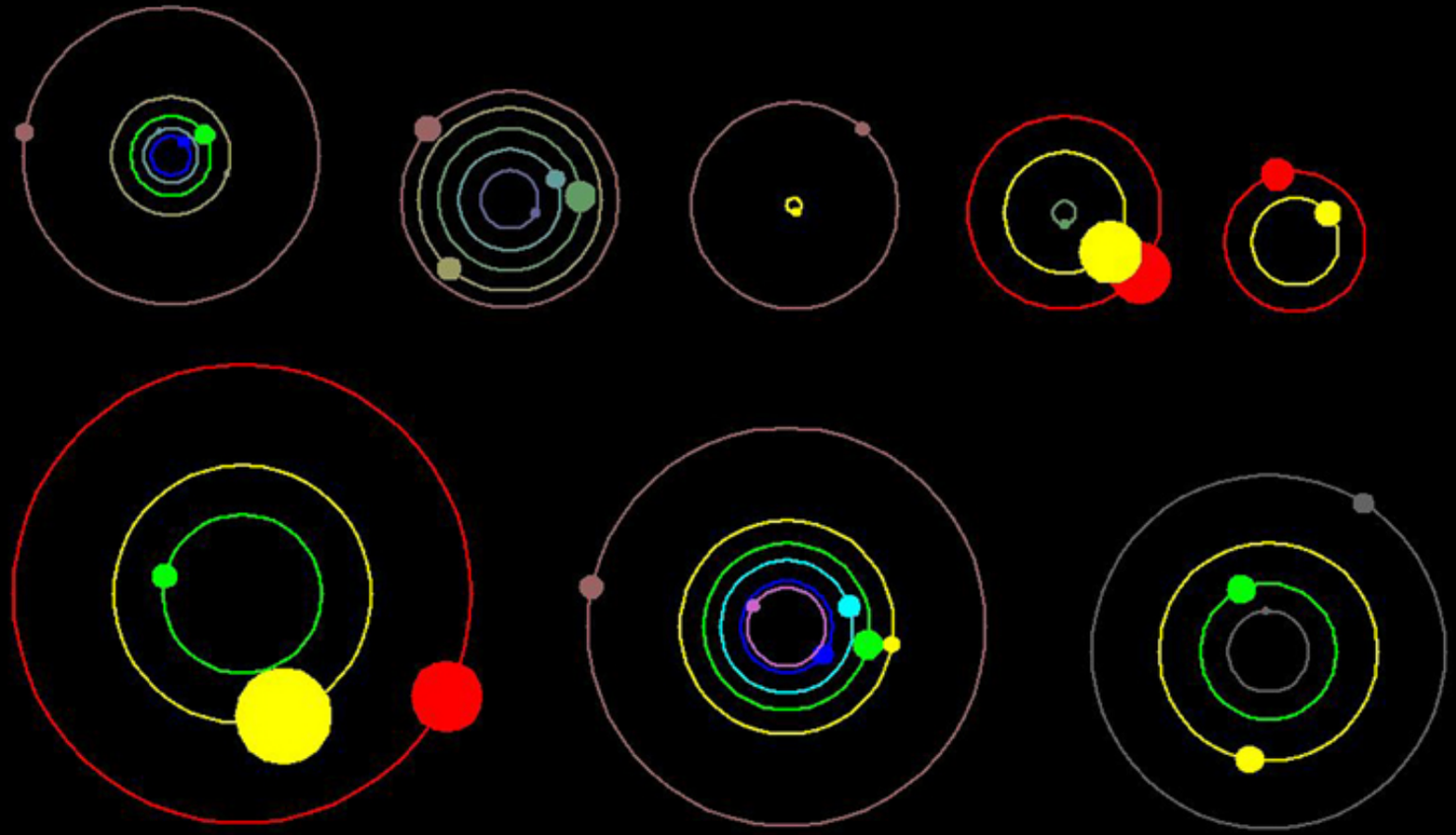


For N=1 and N=2, the equation of motion can be solved analytically.

Applications

- Few-body systems N≅ 3-10
  - planetary systems

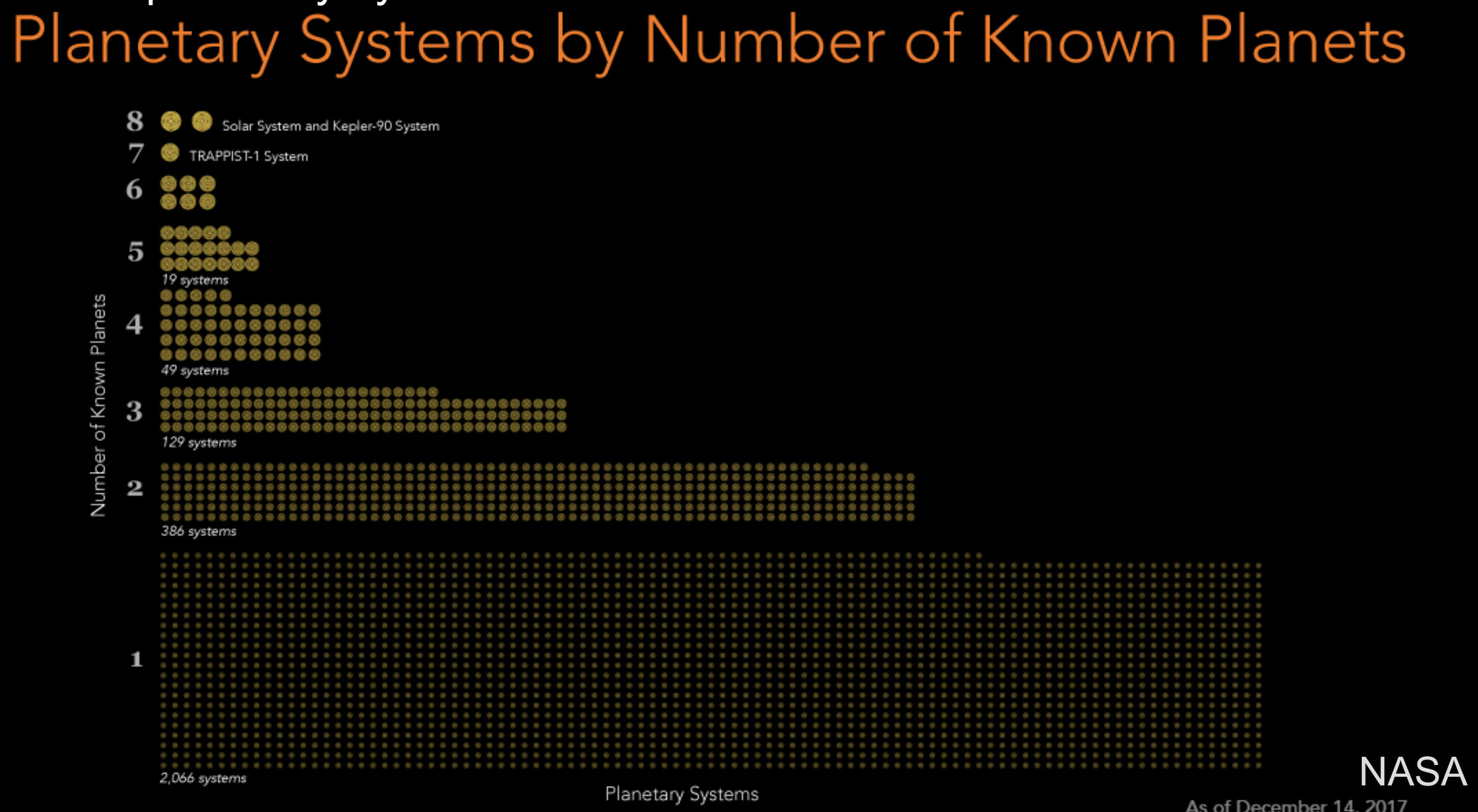




NASA/Ames/UC Santa Cruz 2012

Applications

- Few-body systems N≅ 3-10
  - planetary systems

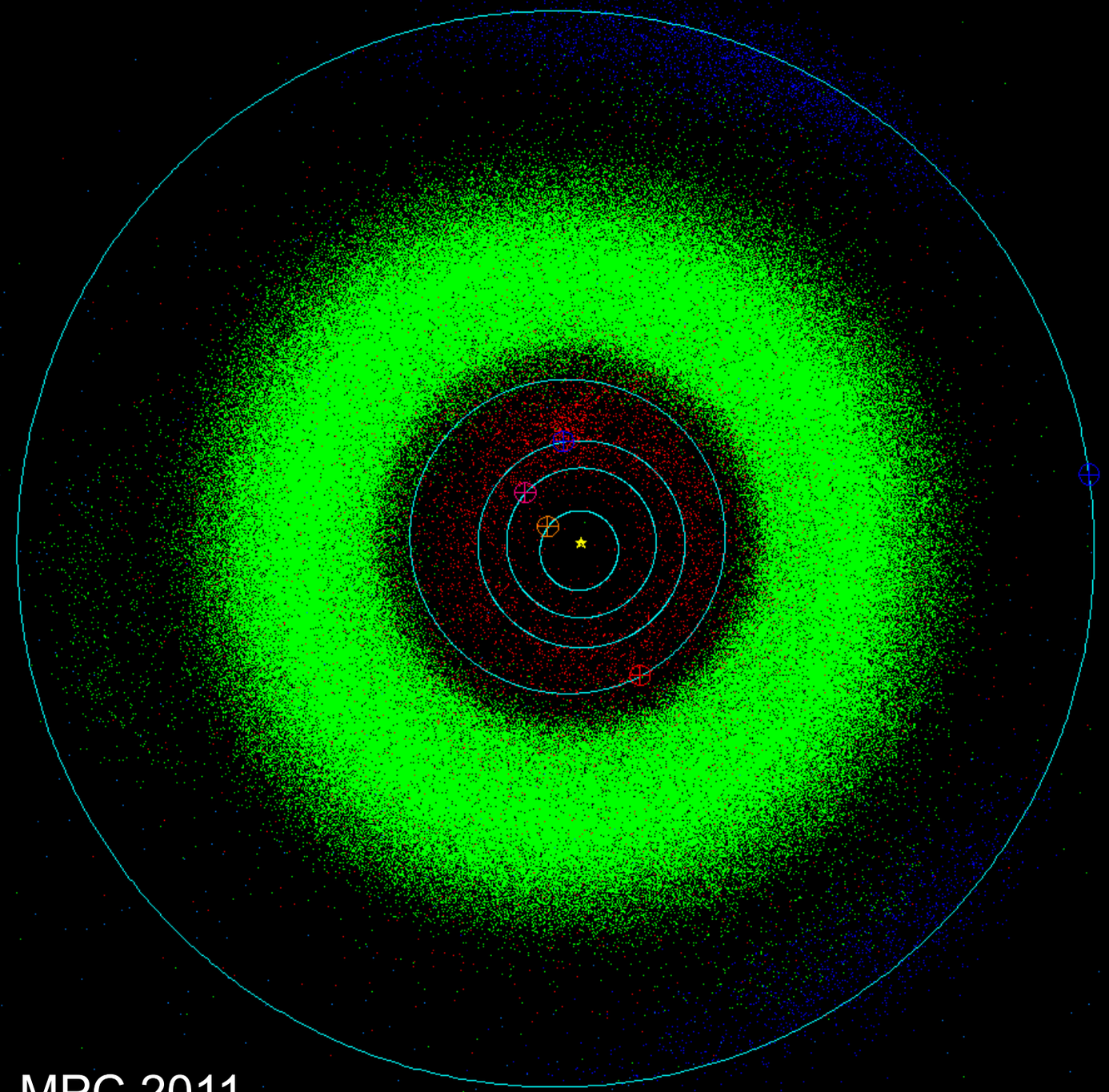


Applications

- Many-body systems
  N ≅ 10-400 and tracers

  planetary systems with minor bodies

  formation of asteroid families



MPC 2011

2011/01/01

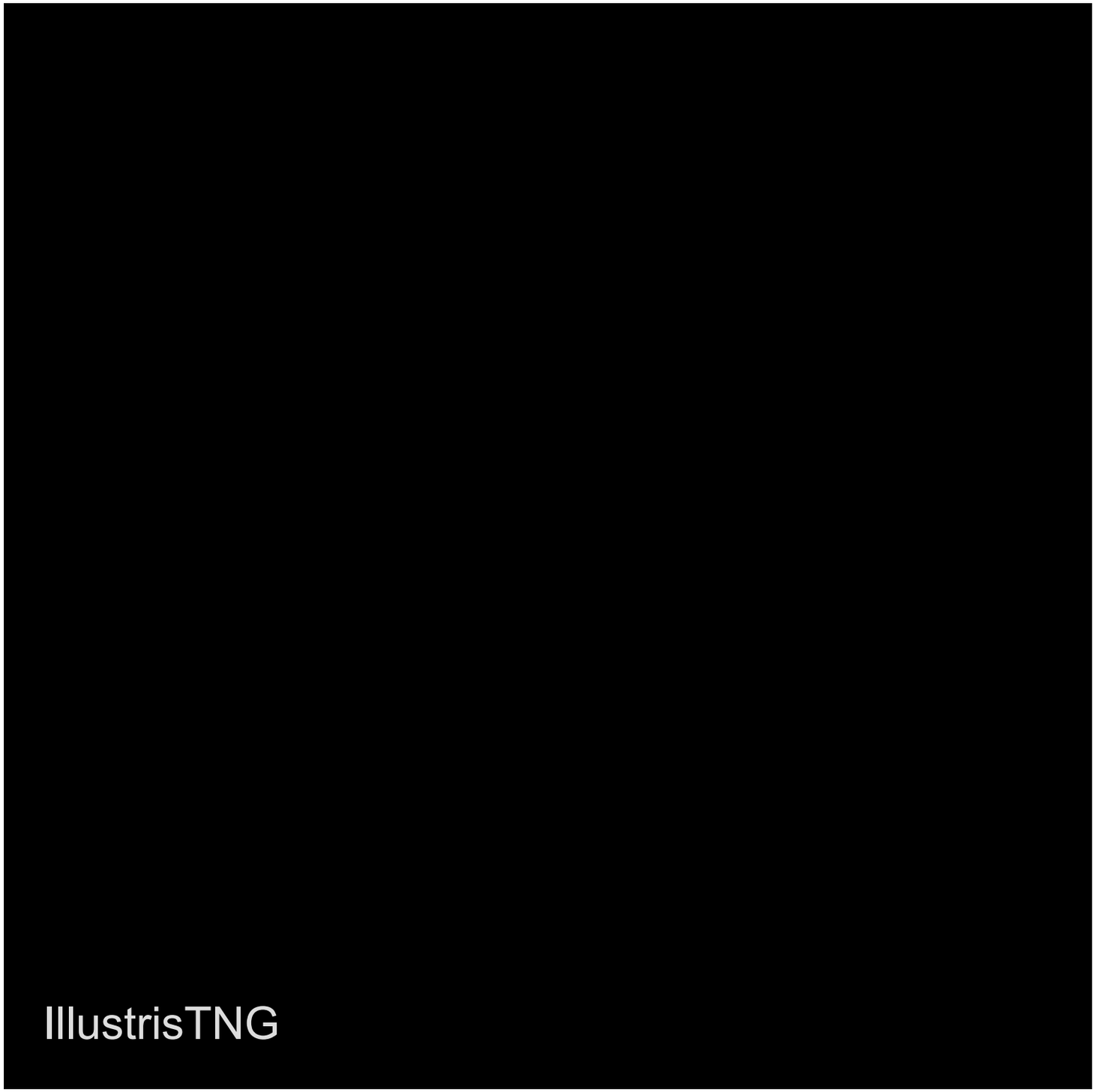
Applications

- Large N-body systems
  - globular star clusters
    $N > 10^3$
  - large-scale structure
    of the universe
  - galactic dynamics and
    cosmology $N > 10^6$

Applications

- Large N-body systems
  - globular star clusters
    $N > 10^3$
  - large-scale structure
    of the universe
  - galactic dynamics and
    cosmology $N > 10^6$

  - coupled hydro-
    nbody simulations,
    Tree-Particle-Mesh
    (Tree-PM)



Applications · Gravity is everywhere

- Few-body systems N≅ 3-10
  - planetary systems

celestial mechanics

- Many-body systems N≅10-400 and tracers
  - planetary systems with minor bodies - formation of asteroid families

- Large N-body systems
  - globular star clusters $N > 10^3$

dense stellar systems

  - large-scale structure of the universe
  - galactic dynamics and cosmology $N > 10^6$

galactic dynamics

Applications · today during the workshop

- Few-body systems N≅ 3-10
  - planetary systems

celestial mechanics

- Many-body systems N≅10-400 and tracers
  - planetary systems with minor bodies - formation of asteroid families

- Large N-body systems
  - globular star clusters $N > 10^3$

dense stellar systems

  - large-scale structure of the universe
  - galactic dynamics and cosmology $N > 10^6$

galactic dynamics

Direct N-Body

set of N·dimension 2nd order differential equations

$$\frac{\mathrm{d}^2\mathbf{r}_i}{\mathrm{d}t^2} = -G\sum_{j\neq i}^{N} m_j \frac{\mathbf{r}_i - \mathbf{r}_j}{\left|\mathbf{r}_i - \mathbf{r}_j\right|^3}$$

⟶ 2N·dimension sets of 1st order differential equations

$$\frac{\mathrm{d}\mathbf{r}_i}{\mathrm{d}t} = \mathbf{v}_i \qquad \frac{\mathrm{d}\mathbf{v}_i}{\mathrm{d}t} = \mathbf{a}_i = -G\sum_{j\neq i}^{N} m_j \frac{\mathbf{r}_i - \mathbf{r}_j}{\left|\mathbf{r}_i - \mathbf{r}_j\right|^3}$$

initial value problem: 2N·dimension additional conditions needed

$$\mathbf{r}_i(t_0), \mathbf{v}_i(t_0)$$

initial positions and velocities have to be known 🤔

Direct N-Body cont'd

THE GLOBAL SOLUTION OF THE *N*-BODY PROBLEM*

WANG QIU-DONG
Department of Mathematical Science, University of Cincinnati, Cincinnati, OH 45221-0025, U.S.A

(Received: 8 November, 1990; accepted: 26 February, 1991)

(2) Some comments: (i) Although the conclusion given here provides a way to integrate the n-body problem, one does not obtain a useful solution in series expansion. The reason for this is because the speed of convergence of the resulting solution is terribly slow. One has to sum, for example, an incredible number of terms, even for an approximate solution of first order in q, p, t. Because we know almost nothing about the complex singular point in the 7-plane, it seems hopeless to try to improve the convergence of such a series expansion.

see the paper by Florin Diacu about the mathematical history

Direct N-Body cont'd

Numerical integration of the equations of motion

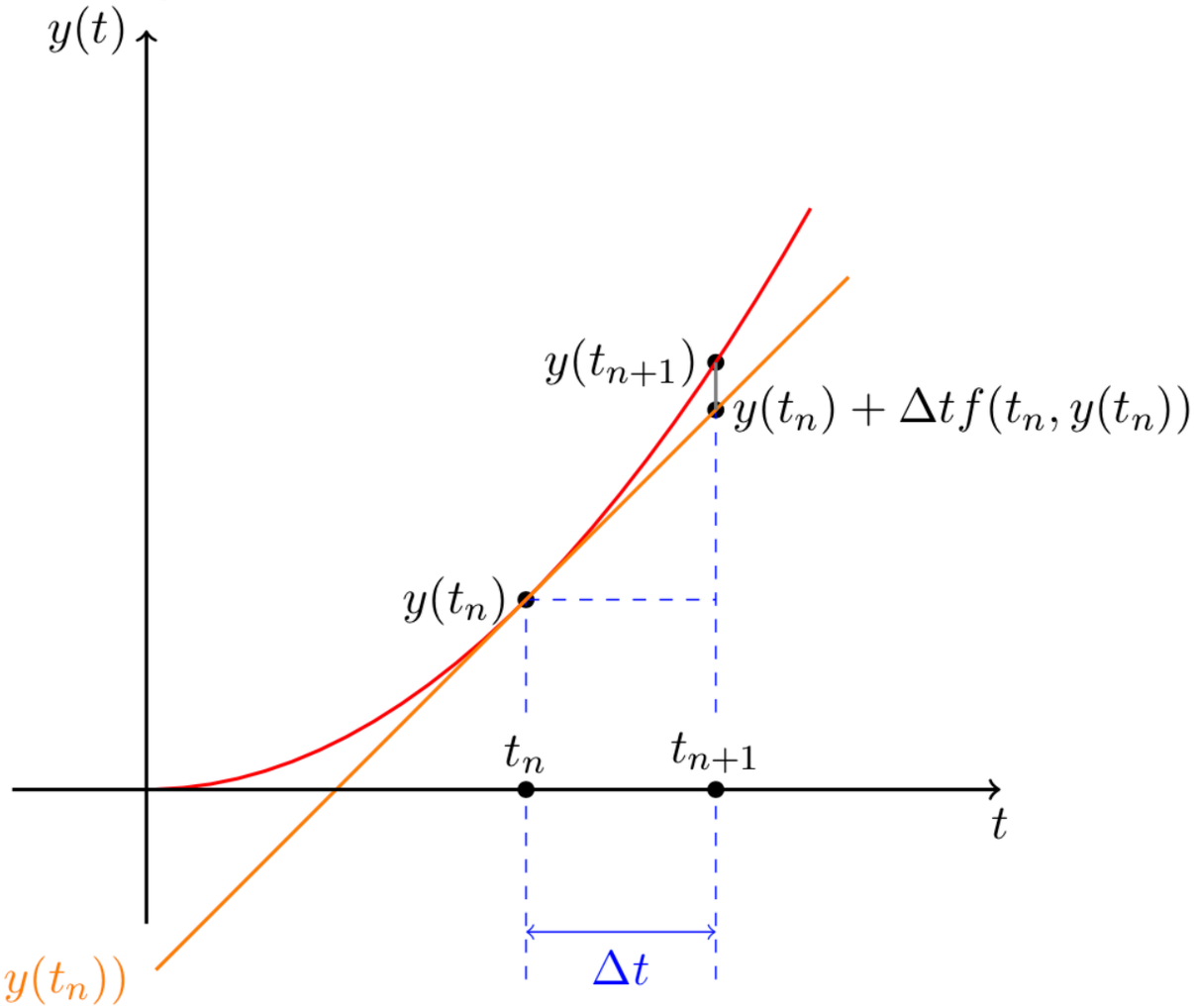
challenge: high accuracy vs. low computational cost

e.g., simple Euler integrator

$$\mathbf{v}_i(t_{n+1}) = \mathbf{v}_i(t_n) + \mathbf{a}_i(t_n)\Delta t$$

$$\mathbf{r}_i(t_{n+1}) = \mathbf{r}_i(t_n) + \mathbf{v}_i(t_n)\Delta t$$

$$\mathbf{a}_i = -G\sum_{j\neq i}^{N} m_j \frac{\mathbf{r}_i - \mathbf{r}_j}{|\mathbf{r}_i - \mathbf{r}_j|^3}$$

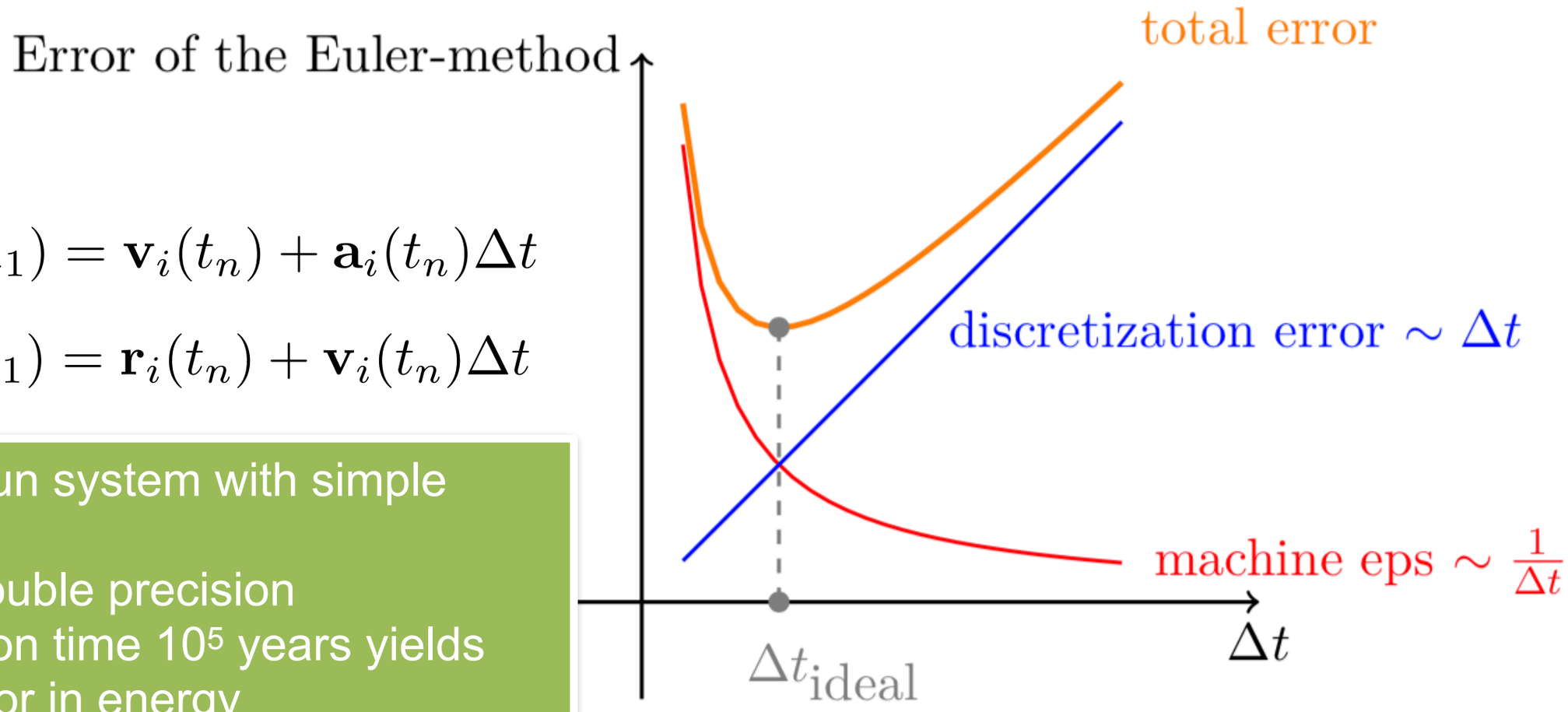


Direct N-Body cont'd

Numerical integration of the equations of motion
Different types of errors
e.g., simple Euler integrator

$$\mathbf{v}_i(t_{n+1}) = \mathbf{v}_i(t_n) + \mathbf{a}_i(t_n)\Delta t$$

$$\mathbf{r}_i(t_{n+1}) = \mathbf{r}_i(t_n) + \mathbf{v}_i(t_n)\Delta t$$



Earth-Sun system with simple Euler:
using double precision
simulation time 10^5 years yields
10% error in energy

Direct N-Body cont'd

Numerical integration of the equations of motion

A vast number of integrators have been developed

one step methods: Euler, Runge-Kutta, …

$$y_{n+1} = y_n + \Delta t \Phi(y, t, \Delta t)$$

e.g., Euler

$$\mathbf{v}_i(t_{n+1}) = \mathbf{v}_i(t_n) + \mathbf{a}_i(t_n)\Delta t$$

$$\mathbf{r}_i(t_{n+1}) = \mathbf{r}_i(t_n) + \mathbf{v}_i(t_n)\Delta t$$

multi step methods: Verlet, Adams–Bashforth, Nyström, Hermite,…

$$y_{n+1} = y_n + \Delta t \sum_{i=0}^{q-1} \beta_i f(t_{n-i}, y_{n-i})$$

e.g., Verlet

$$\mathbf{r}_{n+1} = 2\mathbf{r}_n - \mathbf{r}_{n-1} + \mathbf{a}_n \Delta t^2$$

symplectic integrators: Leap-Frog, Wisdom-Holman, symplectic RKs, …

Hamiltonian systems

energy conservation

e.g., Leap-Frog for N-Body

$$\mathbf{v}_{n+1} = \mathbf{v}_n + \mathbf{a}_{n+1/2}\Delta t$$

$$\mathbf{r}_{n+3/2} = \mathbf{r}_{n+1/2} + \mathbf{v}_{n+1}\Delta t$$

Time Integrators

- Consider the following problem

$$y'(x) = \frac{\mathrm{d}y}{\mathrm{d}x} = f(x, (y(x)), \quad a \leq x \leq b,$$
$$y(a) = y_0 .$$

- Finite interval [a,b] and function f(x,y) are given.

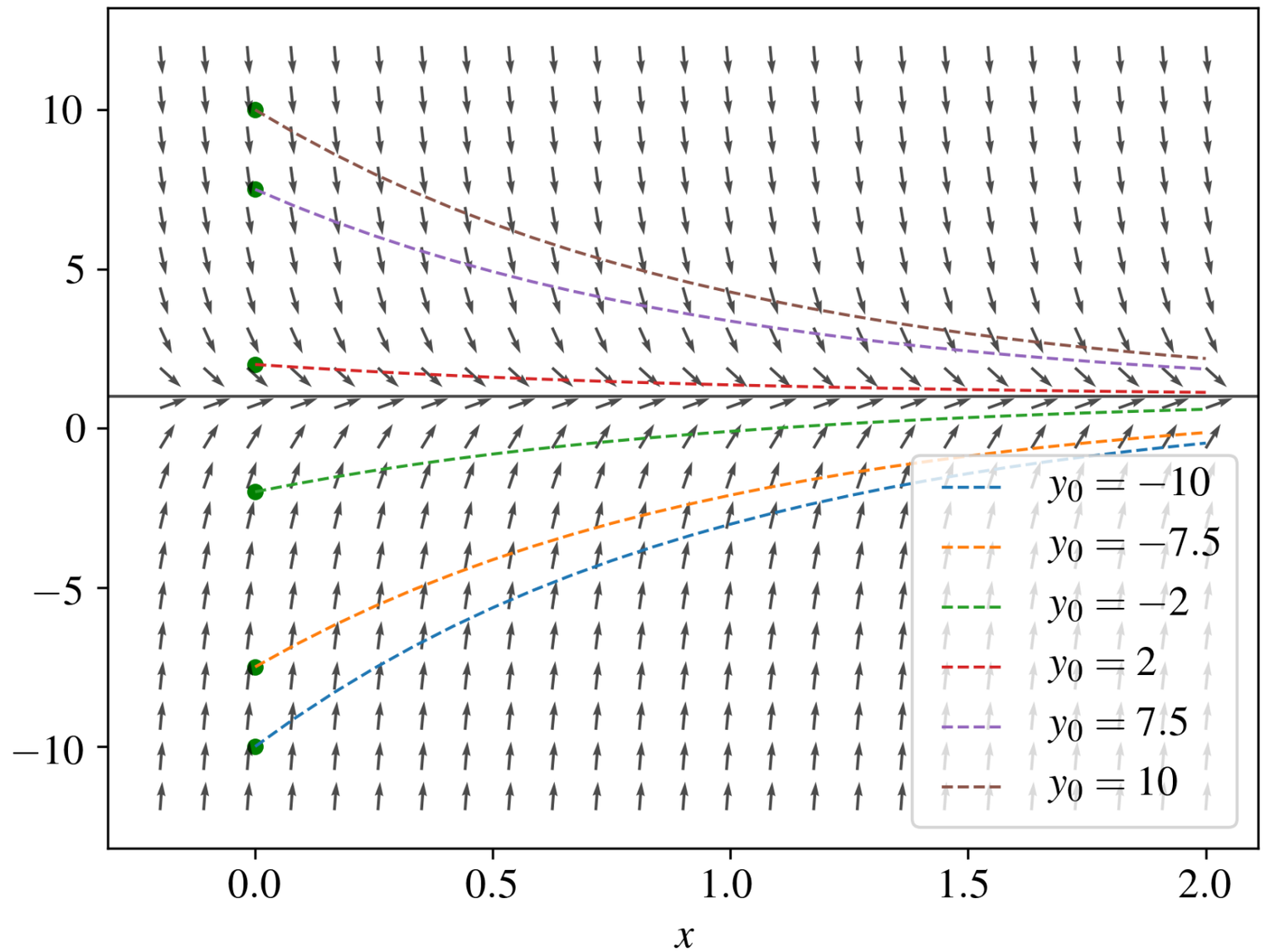
- We seek a function y(x) defined on [a,b], such that $y'(x) = f(x,y(x))$ for $a \leq x \leq b$, and such that y(x) satisfies the initial condition $y(a) = y_0$, where y_0 is some prescribed value.

- This is called an **initial value problem**, cf. N-Body problem.

Time Integrators

- The IVP consists of two parts: the differential equation which gives the relationship between y(x) and y′(x) and the initial condition $y(a) = y_0$.

- Slope field, vector plot at points (x, y) showing the slope [1, f (x, y(x))]. It gives an impression on the type of possible solutions. The choice of y_0 selects the specific, corresponding solution.



Time Integrators

- Basic idea: write dx and dy as finite intervals Δx and Δy. Discretise interval [a,b] with n points, and walk from point to point with a step size Δx=h.

$$\frac{\mathrm{d}y}{\mathrm{d}x} = \frac{\Delta y}{\Delta x} = f(x,y) \quad \longrightarrow \quad \Delta y \equiv y_{k+1} - y_k = \Delta x f(x,y)\,.$$

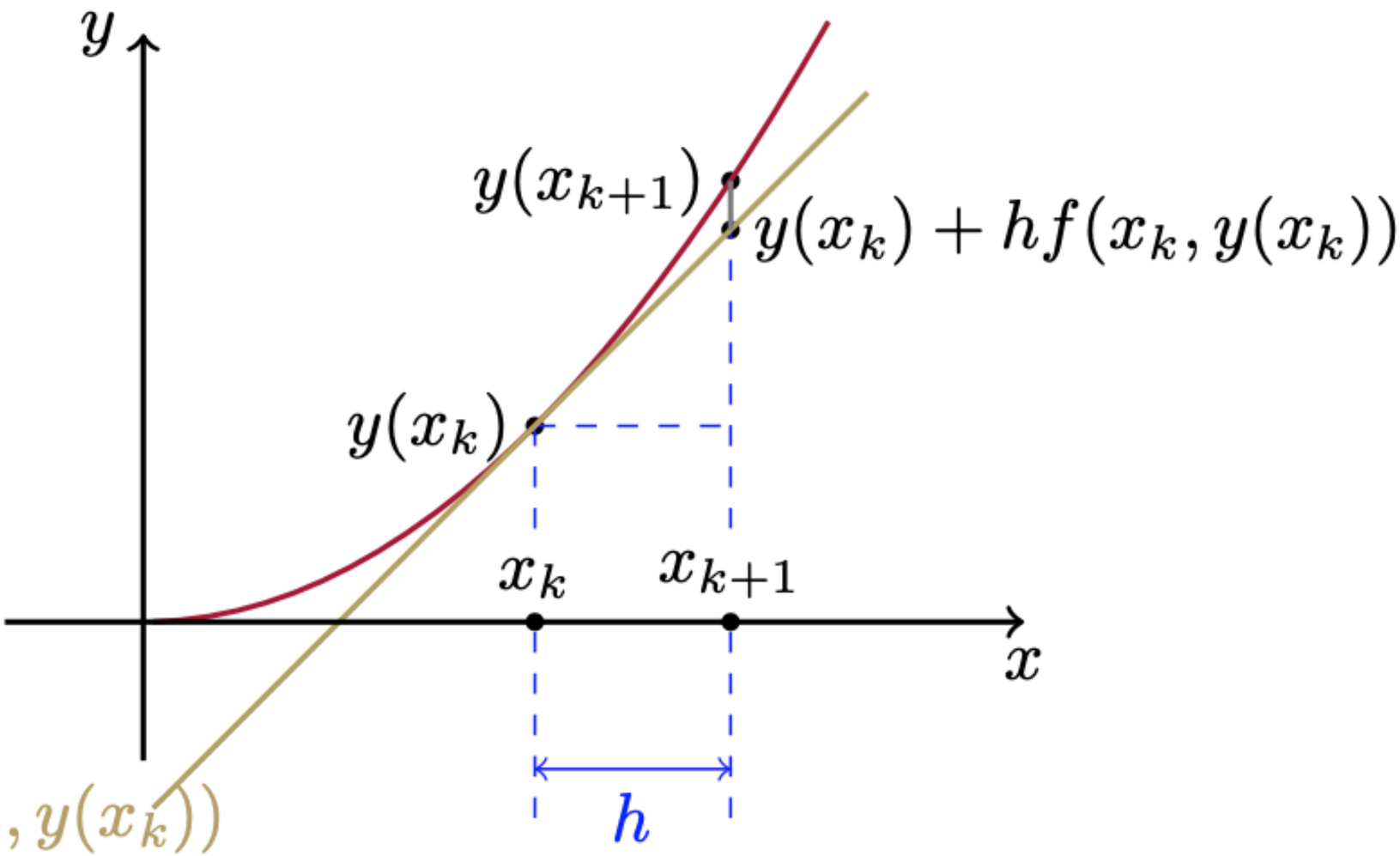
- In general, **one step methods** can be written in the following form

$$y_{k+1} = y_k + h\Phi(x_k, y_k; y_{k+1}, h)\,.$$

  - Start from x_k to x_{k+1} and y_k to y_{k+1}, only use previous values.
  - If the right hand side does not depend on y_{k+1}, i.e. $\Phi = \Phi(x_k, y_k; h)$, **explicit**, otherwise **implicit** method.
  - The function Φ is called **increment function**.

Time Integrators - Euler Integrator

- Increment function is f(x,y): $y_{k+1} = y_k + hy'_k = y_k + hf(x_k, y_k)$.



Time Integrators - Runge-Kutta Integrators

- The Runge-Kutta methods are based on integration by Lagrange polynomials.
- We rewrite the ODE as an integral equation

$$y' = f(x, y(x)) \quad \Leftrightarrow \quad \int_{x_k}^{x_{k+1}} \frac{\mathrm{d}y}{\mathrm{d}x} \mathrm{d}x = \int_{x_k}^{x_{k+1}} f(x, y(x)) \mathrm{d}x,$$

  and obtain

$$y(x_{k+1}) = y(x_k) + \int_{x_k}^{x_{k+1}} f(x, y(x)) \mathrm{d}x$$

- Now, the integral on the right hand side is solved by the help of polynomial expansion. One possibility is to apply the trapezoidal rule with $h = x_{k+1} - x_k$ and use

$$\int_{x_k}^{x_{k+1}} y' \mathrm{d}x \approx \frac{h}{2} \left[y'(x_k) + y'(x_{k+1}) \right].$$

Time Integrators - Runge-Kutta Integrators

- Since $y'(x_{k+1})$ is not known yet, we have to approximate it, e.g., with an Euler integrator step

$$y'(x_k) = K_1 = f(x_k, y_k),$$
$$y'(x_{k+1}) = K_2 = f(x_{k+1}, y_{k+1}) = f(x_k + h, y_k + hK_1)\,.$$

- This gives us a 2nd order Runge-Kutta integrator, knows as **Heun's method**

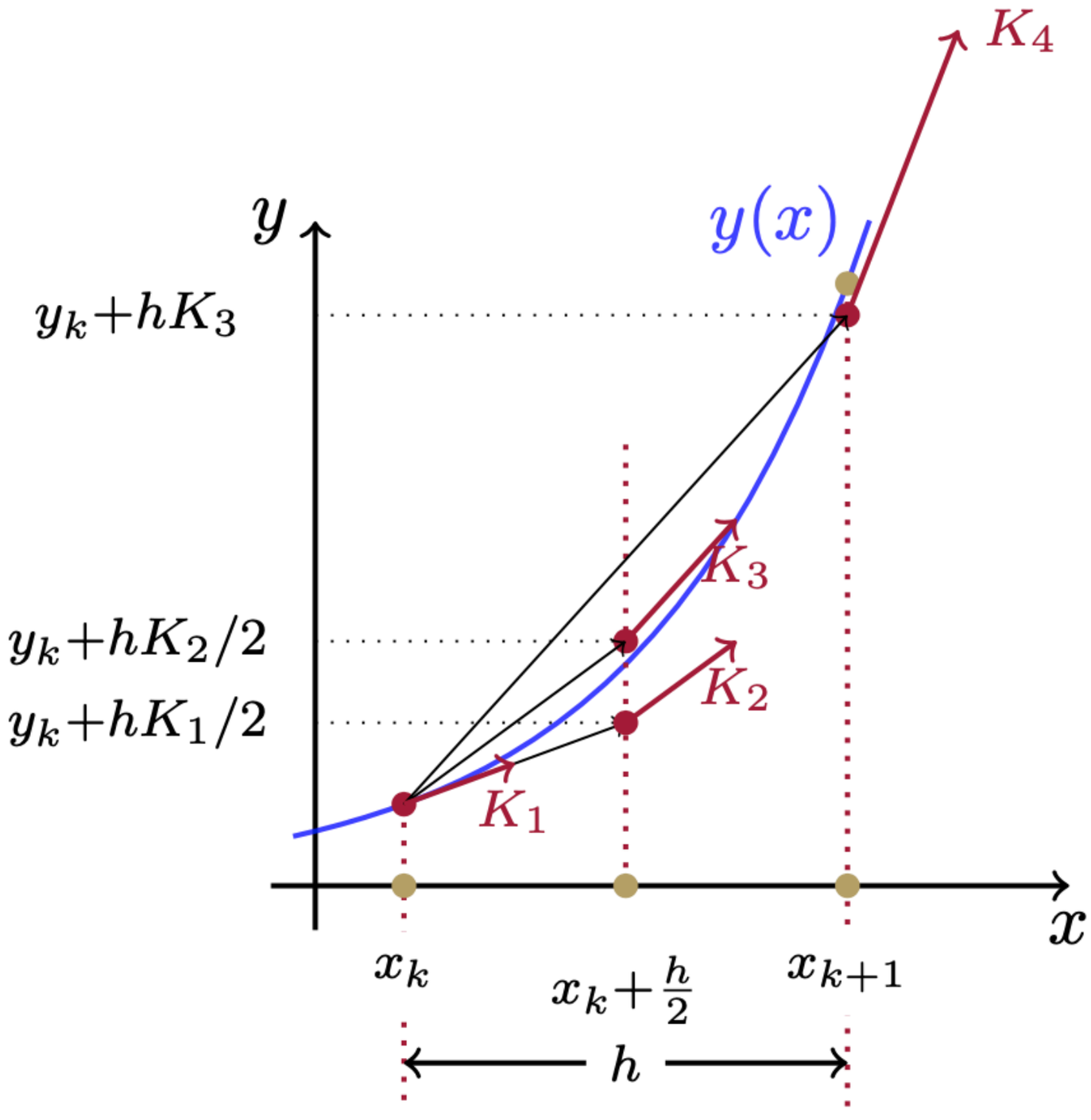
$$K_1 = f(x_k, y_k),$$
$$K_2 = f(x_k + h, y_k + hK_1),$$
$$y_{k+1} = y_k + \frac{h}{2}(K_1 + K_2)\,.$$

- Heun's method is also a vivid example for the diversity in integrators: It is both a **2nd order RK** and a simple **predictor-corrector method**.

Time Integrators - Runge-Kutta Integrators - RK4

- One step from x_k to x_{k+1}

$$K_1 = f(x_k, y_k)$$
$$K_2 = f(x_k + \frac{h}{2}, y_k + \frac{h}{2}K_1)$$
$$K_3 = f(x_k + \frac{h}{2}, y_k + \frac{h}{2}K_2)$$
$$K_4 = f(x_k + h, y_k + hK_3)$$
$$y_{k+1} = y_k + \frac{h}{6}(K_1 + 2K_2 + 2K_3 + K_4)\,.$$



Time Integrators - Runge-Kutta Integrators - RK4

```
# rk4 integrator, fixed step size, f is function which returns derivative
# integration in interval [x0,x1], y0 is initial value
def rk4(f, x0, y0, x1, h):
    N = (x1-x0)//h
    N = int(N)
    x = np.zeros(N+1)
    y = np.zeros(N+1)
    y[0] = yi = y0
    x[0] = xi = x0
    for i in range(1,N+1):
        k1 = h * f(yi,xi)
        k2 = h * f(yi+0.5*k1, xi+0.5*h)
        k3 = h * f(yi+0.5*k2, xi+0.5*h)
        k4 = h * f(yi+k3,xi+h)
        x[i] = xi = x0 + i*h
        y[i] = yi = yi + (k1+k2+k2+k3+k3+k4)/6
    return x,y
```

Time Integrators - multi-step methods

- So far, we have only used the information from the system during one step from k to k+1 and discarded all information from the steps before.

- The multi-step methods also use the information from these former steps.

- Advantage: higher accuracy possible, more efficient.
  Disadvantages: more memory required, more complicated algorithms.

- A **linear** multistep method uses a linear combination of the y_k and $f(x_k,y_k)$ of the previous s steps, its general form is

$$y_{k+s} + a_{s-1}y_{k+s-1} + a_{s-2}y_{k+s-2} + \ldots + a_0 + y_k = h(b_s f(x_{k+s}, y_{k+s}) + b_{s-1}f(x_{k+s-1}, y_{k+s-1}) + \ldots b_0 f(x_k, y_k)\,.$$

Time Integrators - multi-step methods

- Or

$$\sum_{j=0}^{s} a_j y_{k+j} = h \sum_{j=0}^{s} b_j f(x_{k+j}, y_{k+j}) \,.$$

with $a_s = 1$. The coefficients $a_0, \ldots, a_{s-1}$ and $b_0, \ldots, b_s$ determine the method.

- For $b_s = 0$, one obtains an explicit scheme, since y_{k+j} can be directly computed.
- For $b_s \neq 0$, the scheme is implicit and has to be solved iteratively.
- Simple example: Two-step Adams-Bashforth method (cf. to Euler)

$$y_{k+2} = y_{k+1} + \frac{3}{2} h f(x_{k+1}, y_{k+1}) + \frac{1}{2} h f(x_k, y_k) \,.$$

Time Integrators - Symplectic Integrators

- Symplectic integrators are designed for the numerical solution of a Hamiltonian system

$$\dot{q} = \frac{\partial H}{\partial p}, \quad \dot{p} = -\frac{\partial H}{\partial q}$$

- Main idea: The transformation of the coordinates and momenta from t_0 to a later time t_1 is canonical and hence phase space volume preserving
- In the following, we focus on separable Hamiltonians

$$H(p, q) = T(p) + V(q)$$

  with kinetic energy T and potential energy V.

- Introducing $z = (q, p)$, we can write the Hamiltonian eqs using the Poisson bracket {} ($\{f, g\} = \frac{\partial f}{\partial q}\frac{\partial g}{\partial p} - \frac{\partial f}{\partial p}\frac{\partial g}{\partial q}$):

$$\dot{z} = \{z, H\}$$

Time Integrators - Symplectic Integrators

- With the operator $D_H = \{\,\cdot\,, H\}$, the formal solution of this equation is

$$z(t) = \exp(tD_H)z(0).$$

- Or for a separable Hamiltonian

$$z(t) = \exp(t(D_T + D_V))z(0).$$

- We now approximate the time-evolution operator by a product of operators

$$\exp(t(D_T + D_V)) = \Pi_{i=1}^{k} \exp(c_i t D_T) \exp(d_i t D_V) + \mathcal{O}(t^{k+1})$$

  where $\sum_i c_i = 1 = \sum_i d_i$

- Both operators $\exp(c_i t D_T)$ and $\exp(d_i t D_V)$ provide a symplectic map (phase-space volume conserved mappings), and since $D_T^2 z = \{\{z, T\}, T\} = \{(\dot{q},0), T\} = 0$, we find

$$\exp(c_i D_T) = \sum_{n=0}^{\infty} \frac{(c_i D_T)^n}{n!} = 1 + c_i D_T$$

  and likewise for D_V.

Time Integrators - Symplectic Integrators

- We finally end up with a symplectic mapping that conserves both energy and momentum

$$\begin{pmatrix} q \\ p \end{pmatrix} \mapsto \begin{pmatrix} q + tc_i \partial_p T(p) \\ p \end{pmatrix} \text{ from } \exp(tc_i D_T)$$

$$\begin{pmatrix} q \\ p \end{pmatrix} \mapsto \begin{pmatrix} q \\ p - td_i \partial_q V(q) \end{pmatrix} \text{ from } \exp(td_i D_V)$$

- or in coordinates for space and velocity

$$x_{i+1} = x_i + c_i v_{i+1} \Delta t$$
$$v_{i+1} = v_i + d_i \frac{F(x_i)}{m} \Delta t.$$

- Simple example, symplectic Euler with c=d=1

$$x_{i+1} = x_i + v_{i+1} \Delta t$$
$$v_{i+1} = v_i + \frac{F(x_i)}{m} \Delta t.$$

- Further reading: Kinoshita, H., Yoshida, H., & Nakai, H., Symplectic integrators and their application to dynamical astronomy, Celest. Mech. Dyn. Astron., 1991

Time Integrators - Symplectic Integrators

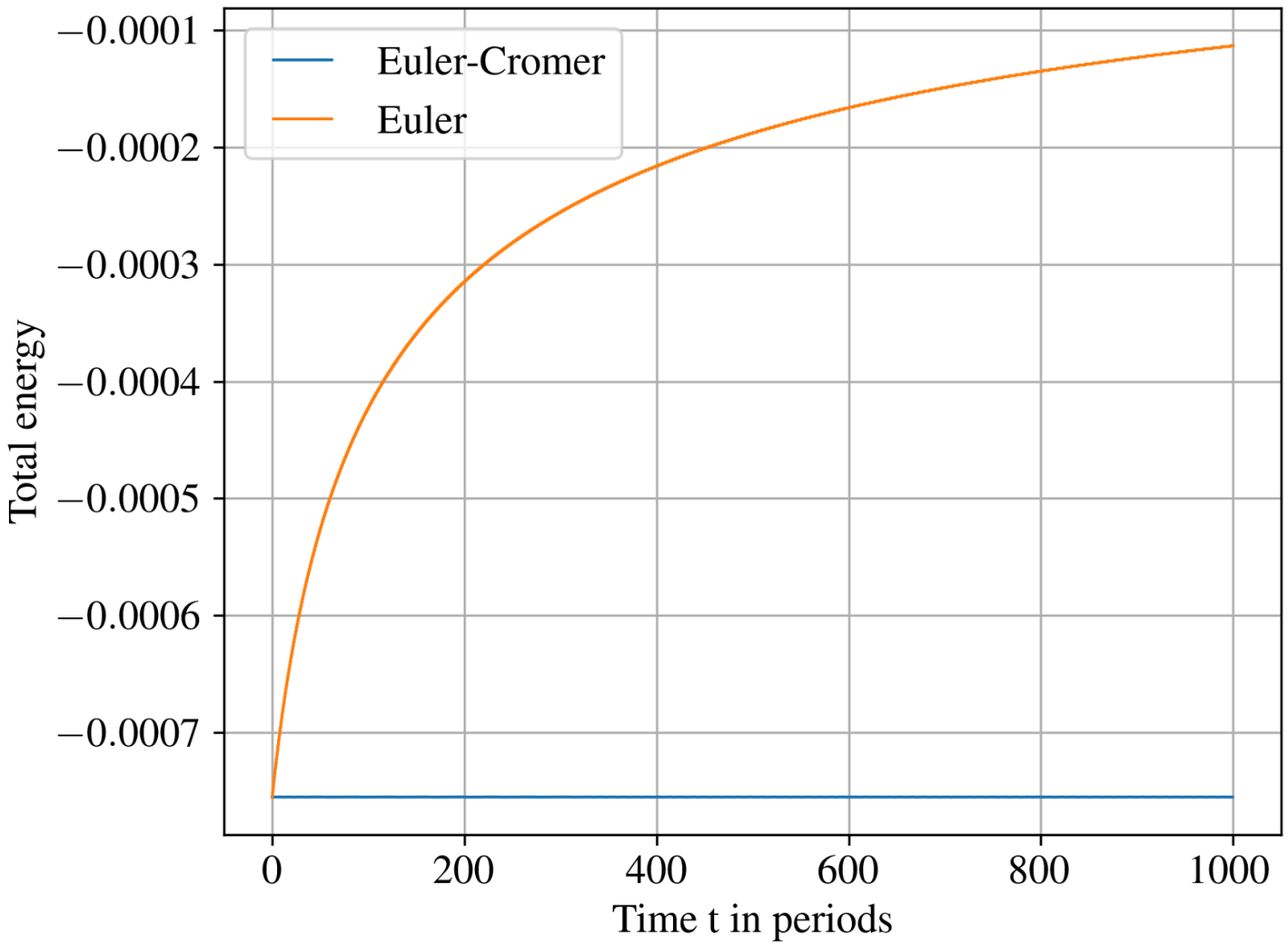
Euler

$$x_{i+1} = x_i + v_i \Delta t$$
$$v_{i+1} = v_i + a_i \Delta t.$$

Euler-Cromer aka symplectic Euler

$$x_{i+1} = x_i + v_{i+1} \Delta t$$
$$v_{i+1} = v_i + a_i \Delta t.$$

Energy,
2-body problem,
integrated for 1000
orbits



Time Integrators - Symplectic Integrators

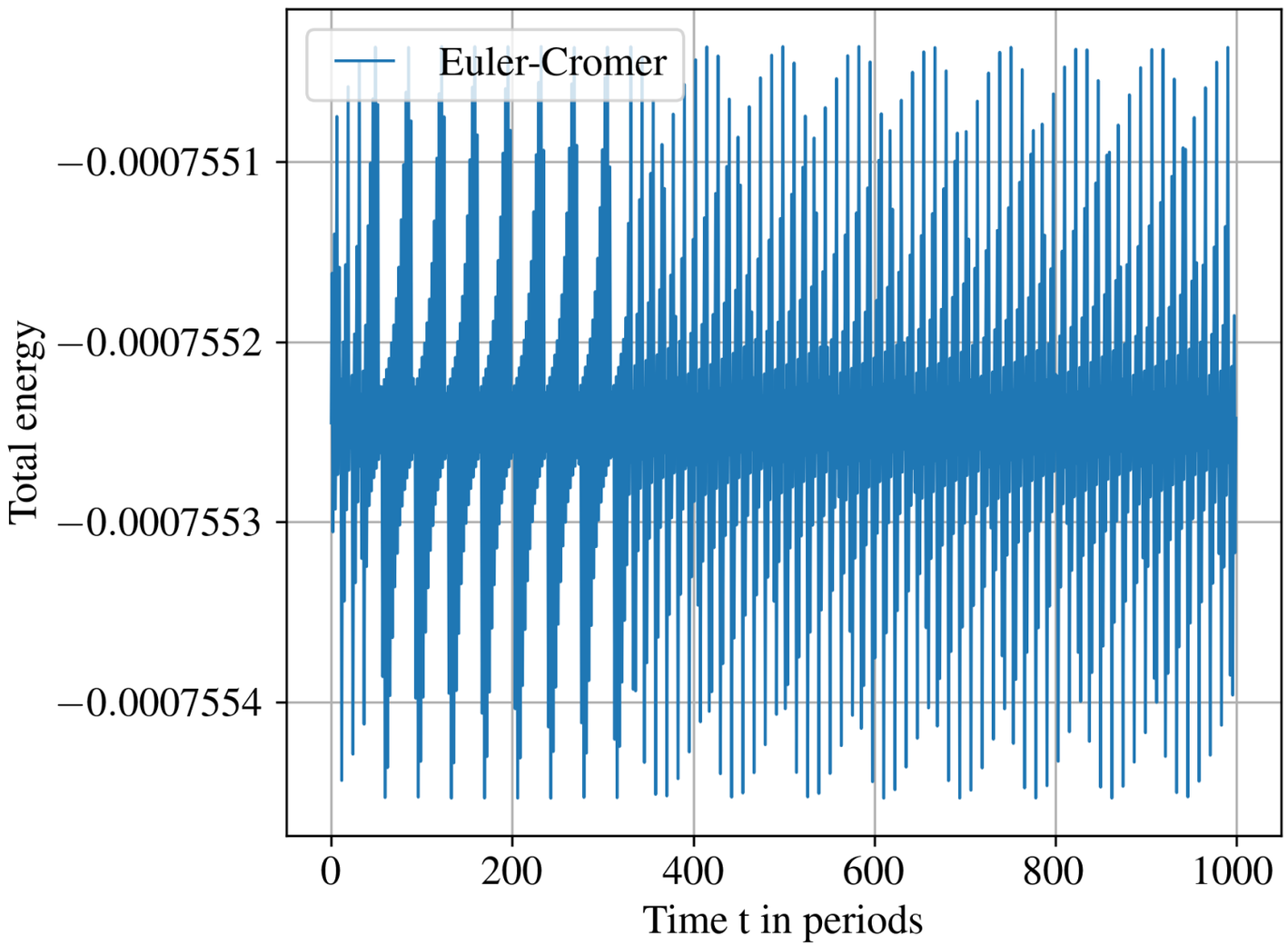
Euler

$$x_{i+1} = x_i + v_i \Delta t$$
$$v_{i+1} = v_i + a_i \Delta t.$$

Euler-Cromer aka symplectic Euler

$$x_{i+1} = x_i + v_{i+1} \Delta t$$
$$v_{i+1} = v_i + a_i \Delta t.$$

Energy,
2-body problem,
integrated for 1000
orbits



Direct N-Body cont'd

Gravitation is long range interaction
calculation of accelerations is N^2

some ideas to speed up simulations

Direct N-Body cont'd

Gravitation is long range interaction
calculation of accelerations is N^2

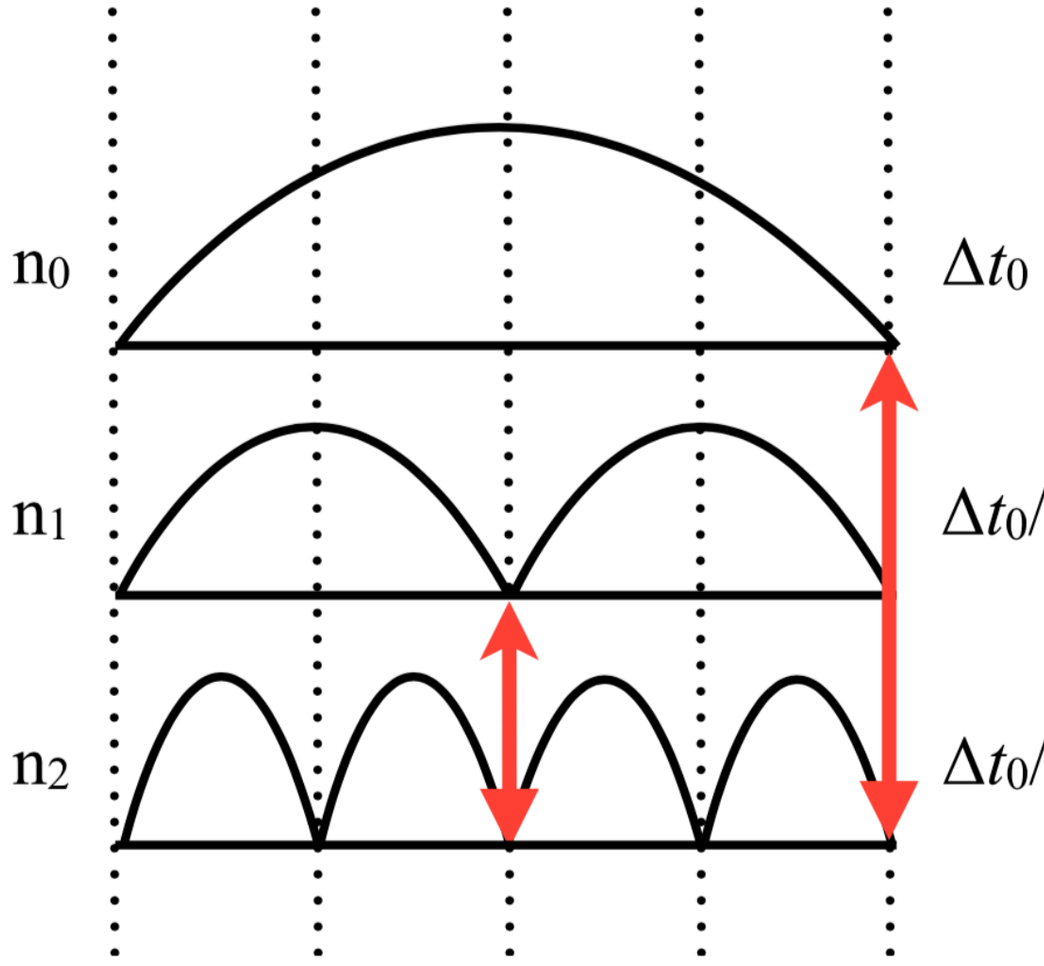
some ideas to speed up simulations

- individual timesteps

$$\Delta t_n = \Delta t_0 / 2^n$$

challenges:
- condition for timestep switch
- individual timesteps break symmetry

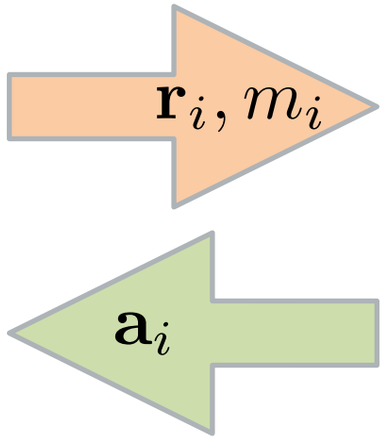
Dehnen & Read 2011

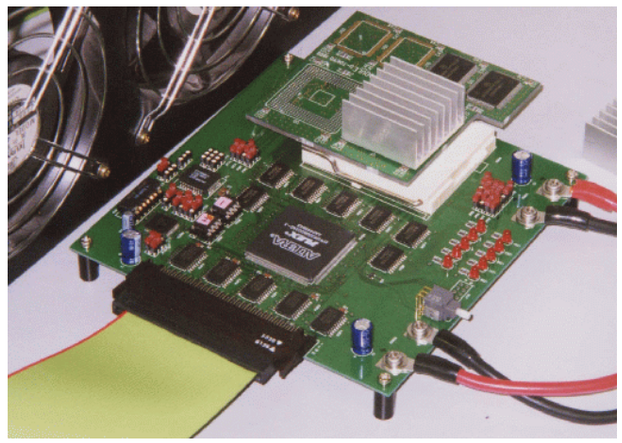


$$\sum_i n_i = N$$

Synchronisation points

Direct N-Body cont'd



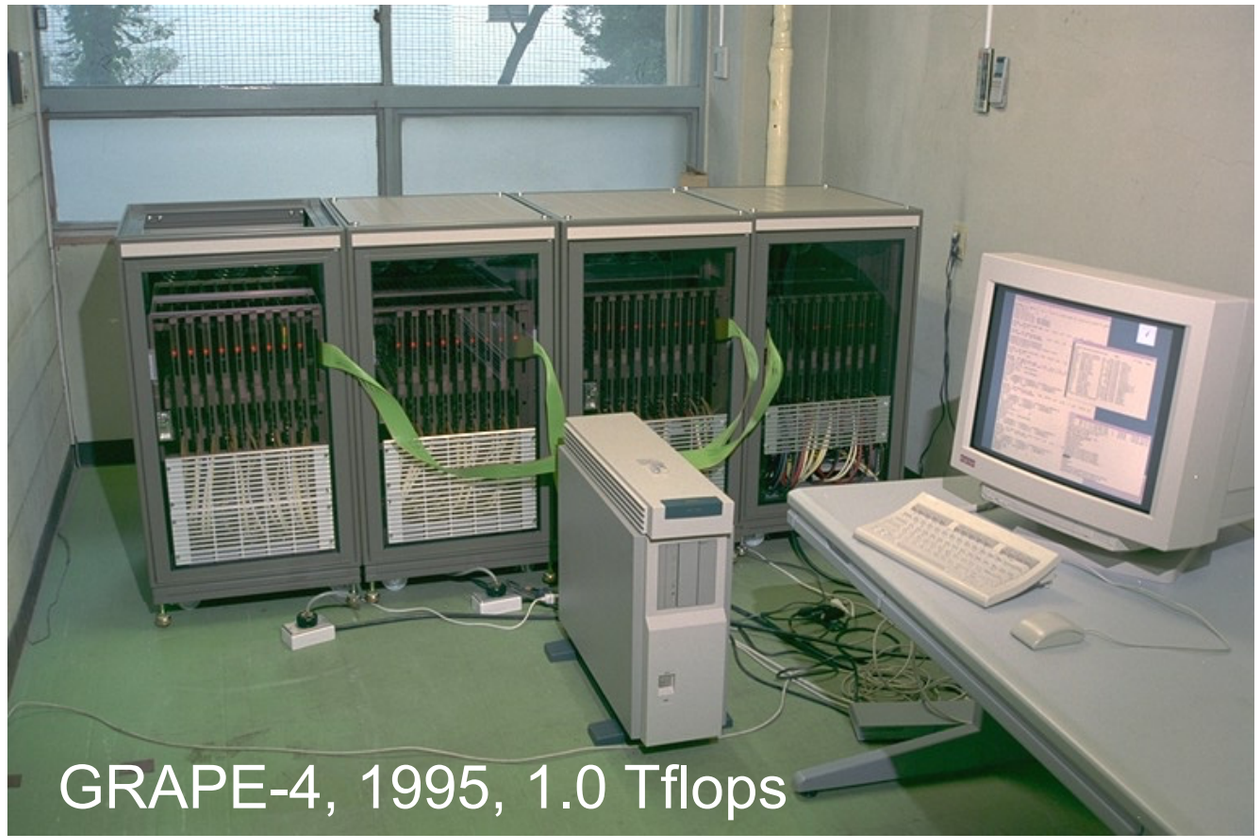


Gravitation is long range interaction
calculation of accelerations is N^2

some ideas to speed up simulations

- special purpose hardware
  GRAvity PipE

last release 200x,
last supercluster 2005:
gravitySimulator, 32 GRAPE boards, 4 Tflops
128'000 particles


GRAPE-4, 1995, 1.0 Tflops

Approximate N-Body

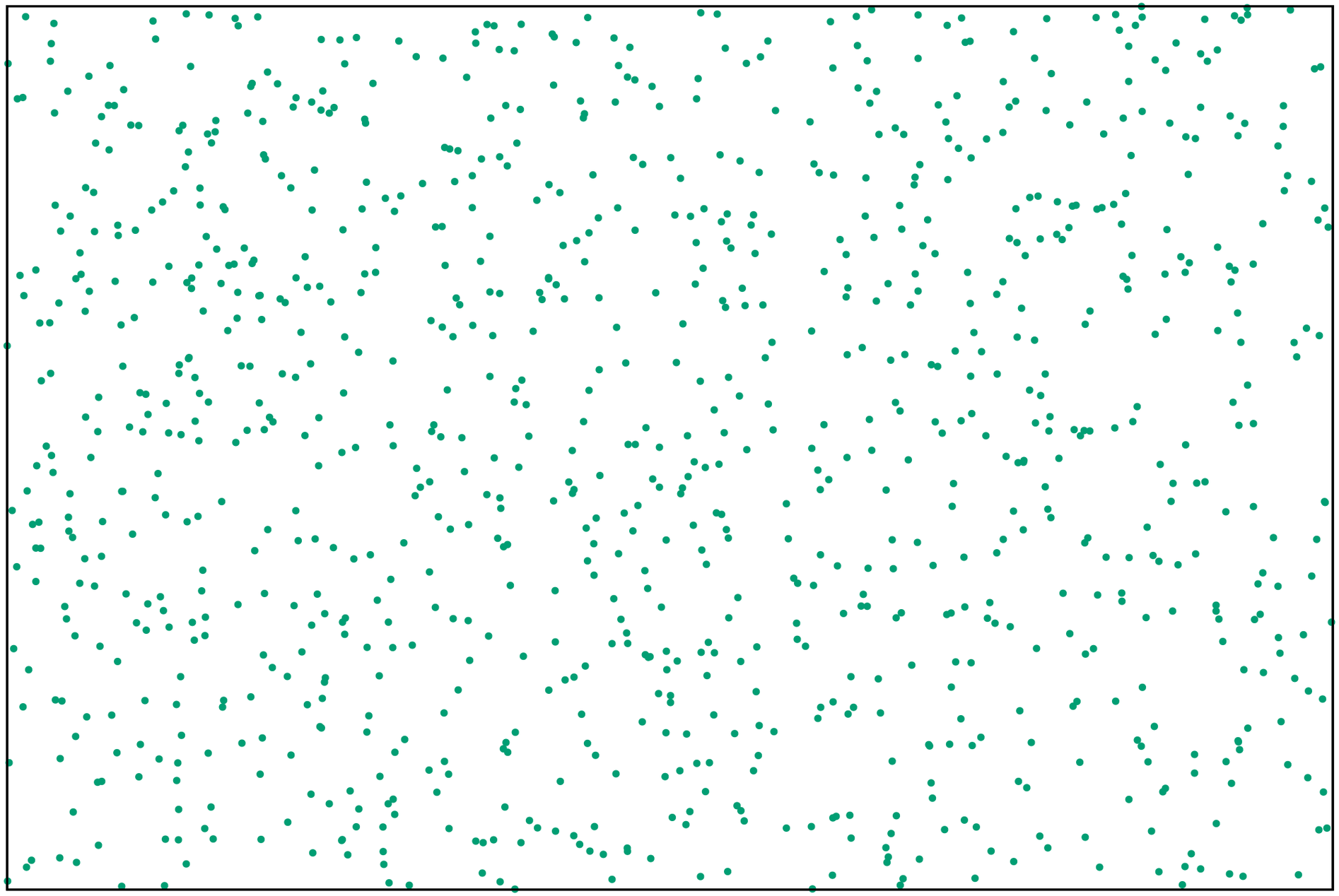
Gravitation is long range interaction
calculation of accelerations is N^2
can we decrease the computations for the cost of lower accuracy?

Barnes-Hut (1986) · hierarchical tree method
basic idea: **reduce** the number of terms in sum

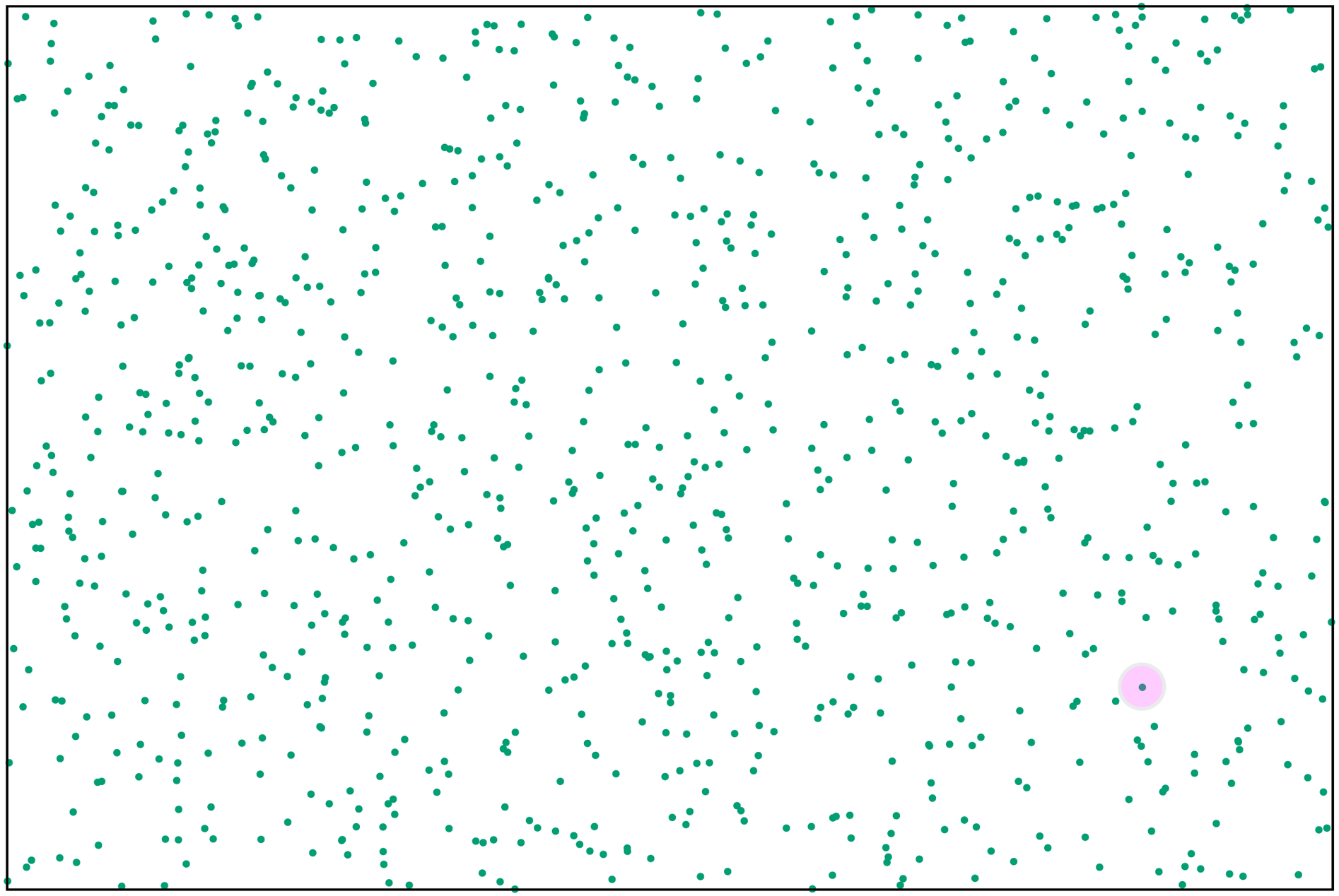
$$\mathbf{a}_i = -G \sum_{j \neq i}^{N} m_j \frac{\mathbf{r}_i - \mathbf{r}_j}{\left|\mathbf{r}_i - \mathbf{r}_j\right|^3}$$

How can we achieve this in a physically correct way?

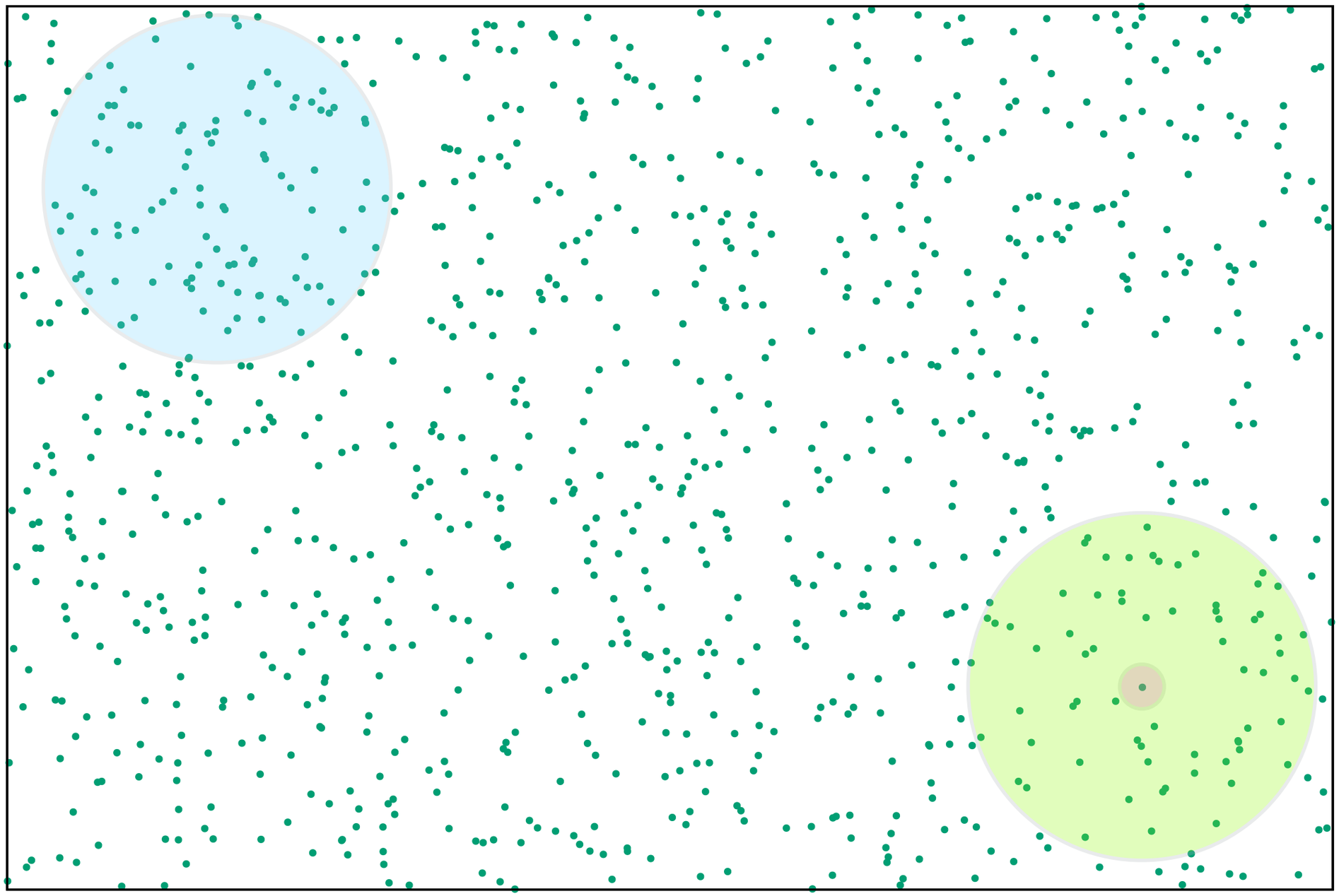
Approximate N-Body - Barnes-Hut Tree



Approximate N-Body - Barnes-Hut Tree

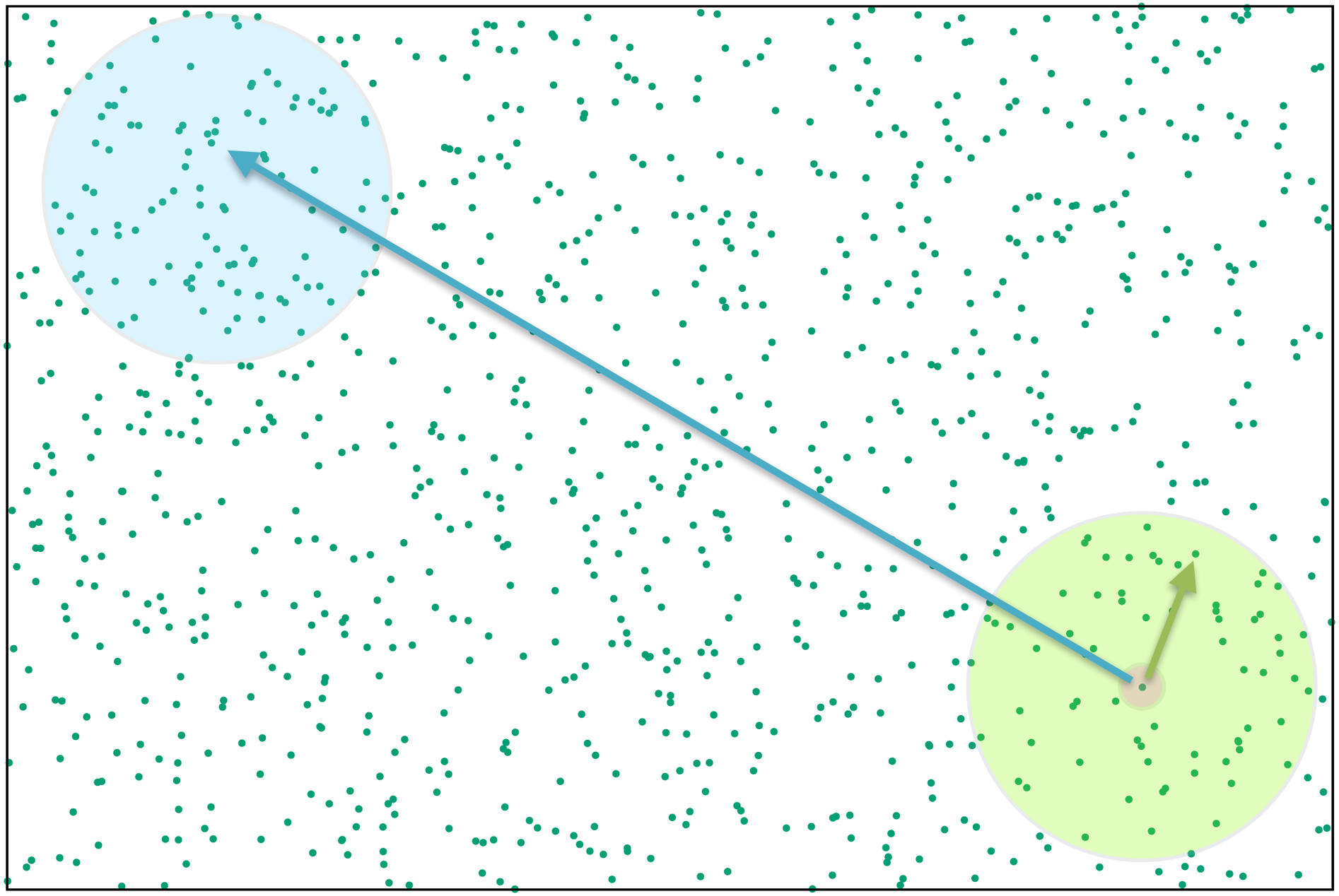


Approximate N-Body - Barnes-Hut Tree



$$\mathbf{a} \sim \frac{1}{r^2}$$

Approximate N-Body - Barnes-Hut Tree



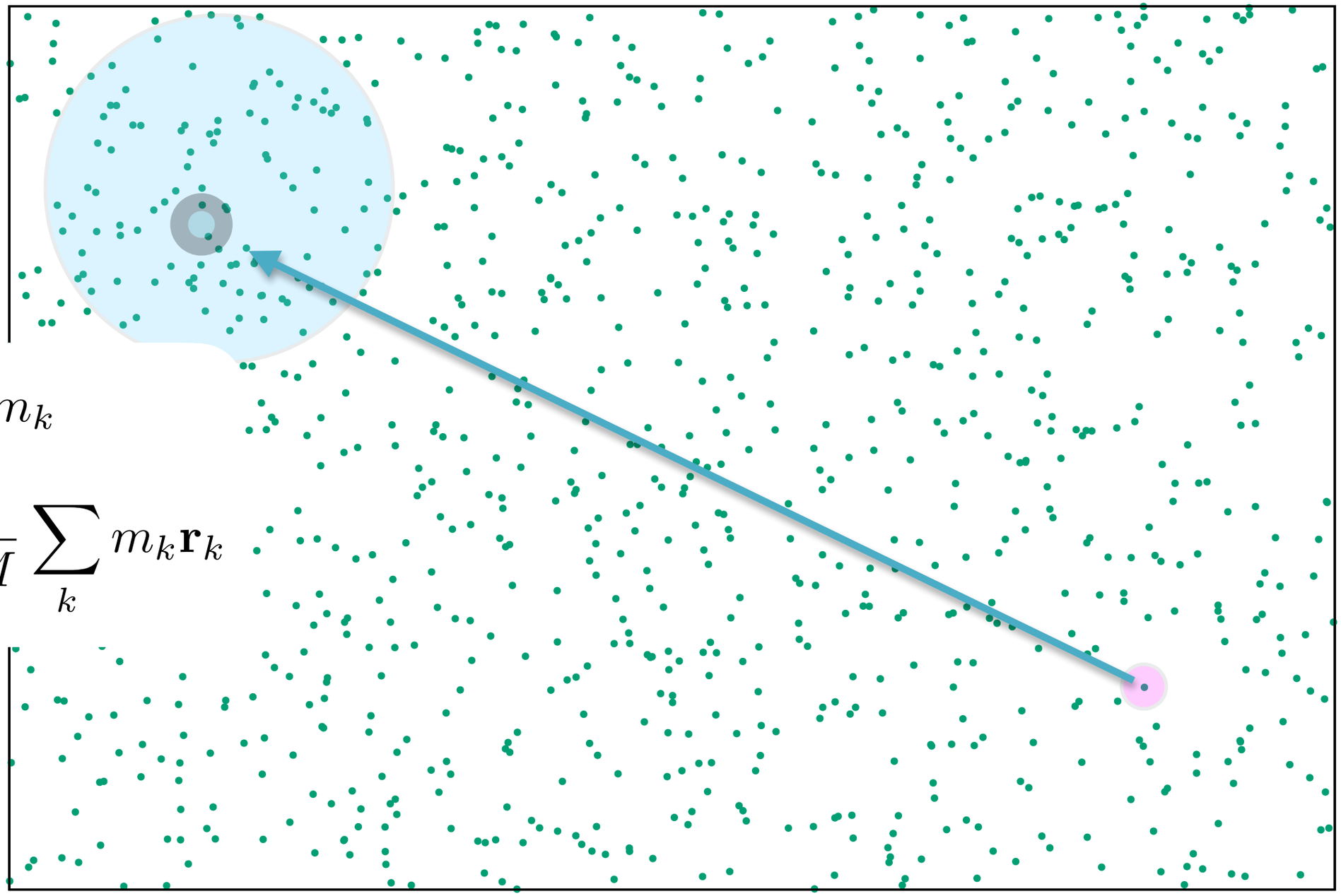
$$\mathbf{a} \sim \frac{1}{r^2}$$

Approximate N-Body - Barnes-Hut Tree

calculate center of mass and mass of blue region

$$M = \sum_k m_k$$

$$\mathbf{r}_{\mathrm{com}} = \frac{1}{M} \sum_k m_k \mathbf{r}_k$$

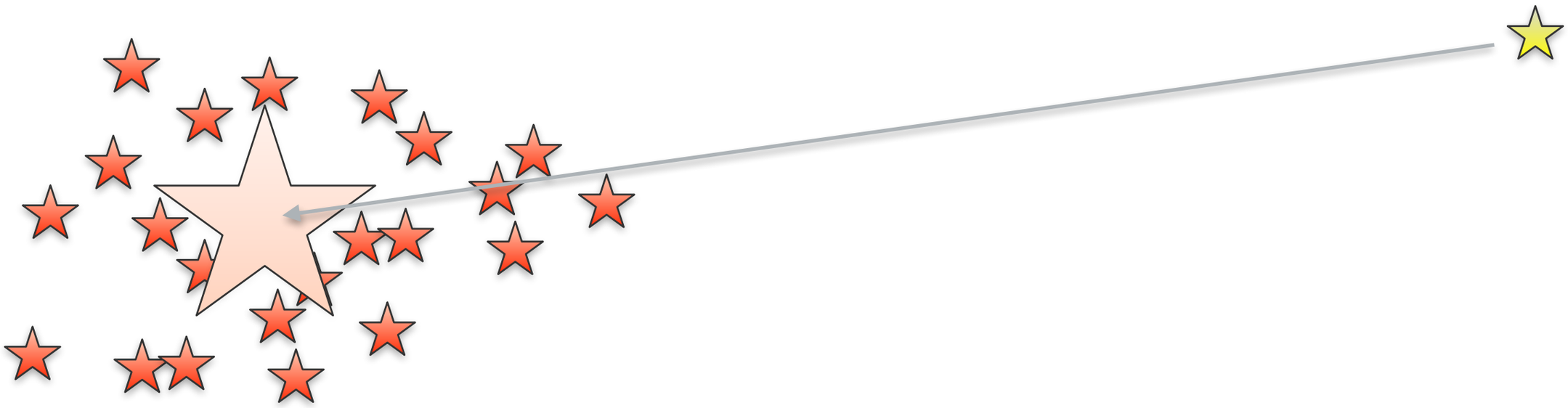


Approximate N-Body - Barnes-Hut Tree

- sort bodies into a hierarchical structure
- add condition for approximation of long range vs. short range

Approximate N-Body - Barnes-Hut Tree

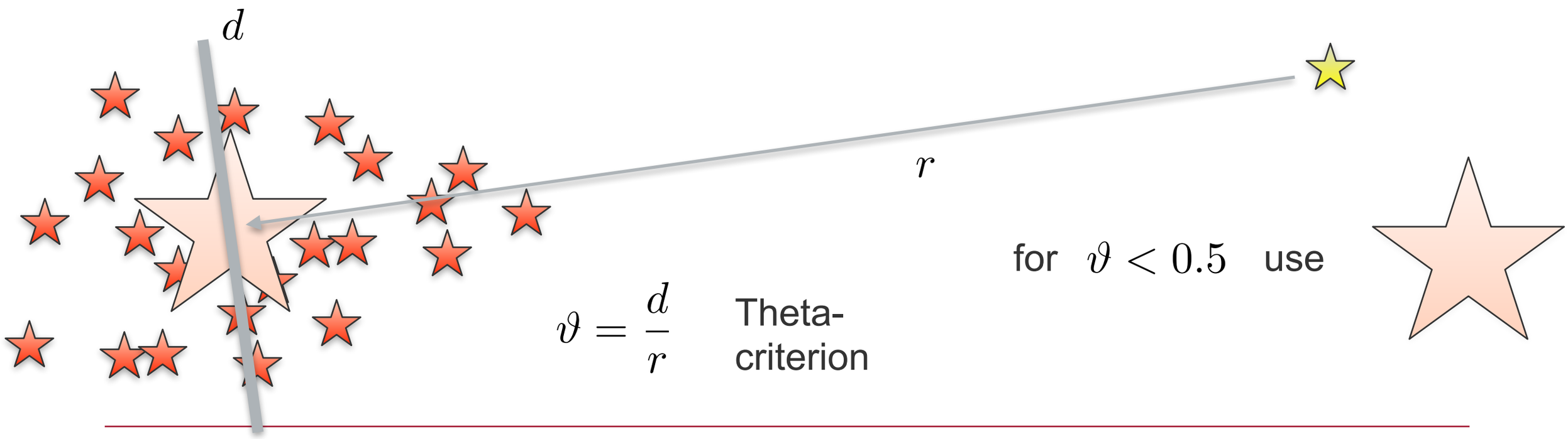
- sort bodies into a hierarchical structure
- add condition for approximation of long range vs. short range



Approximate N-Body - Barnes-Hut Tree

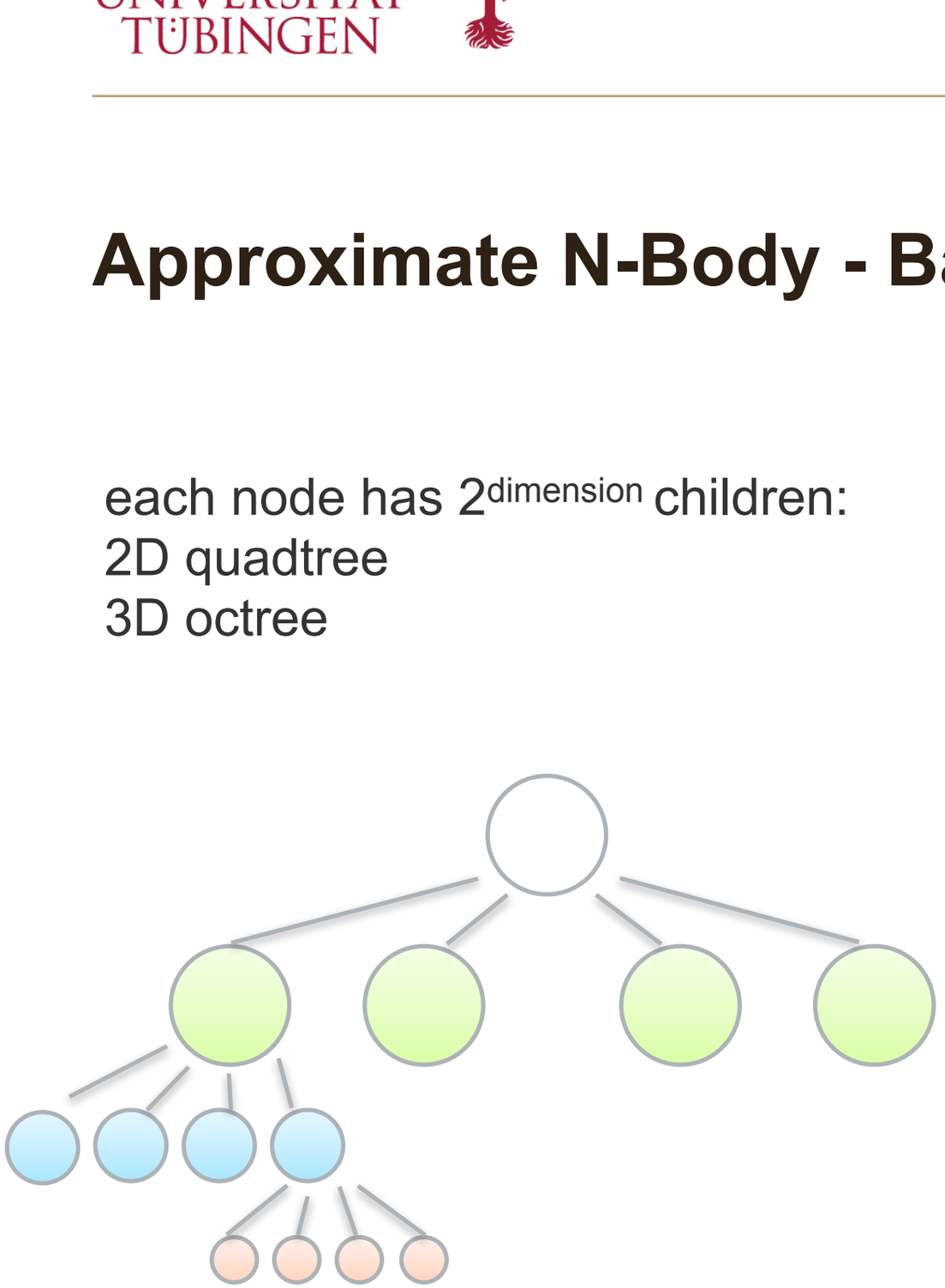
- sort bodies into a hierarchical structure

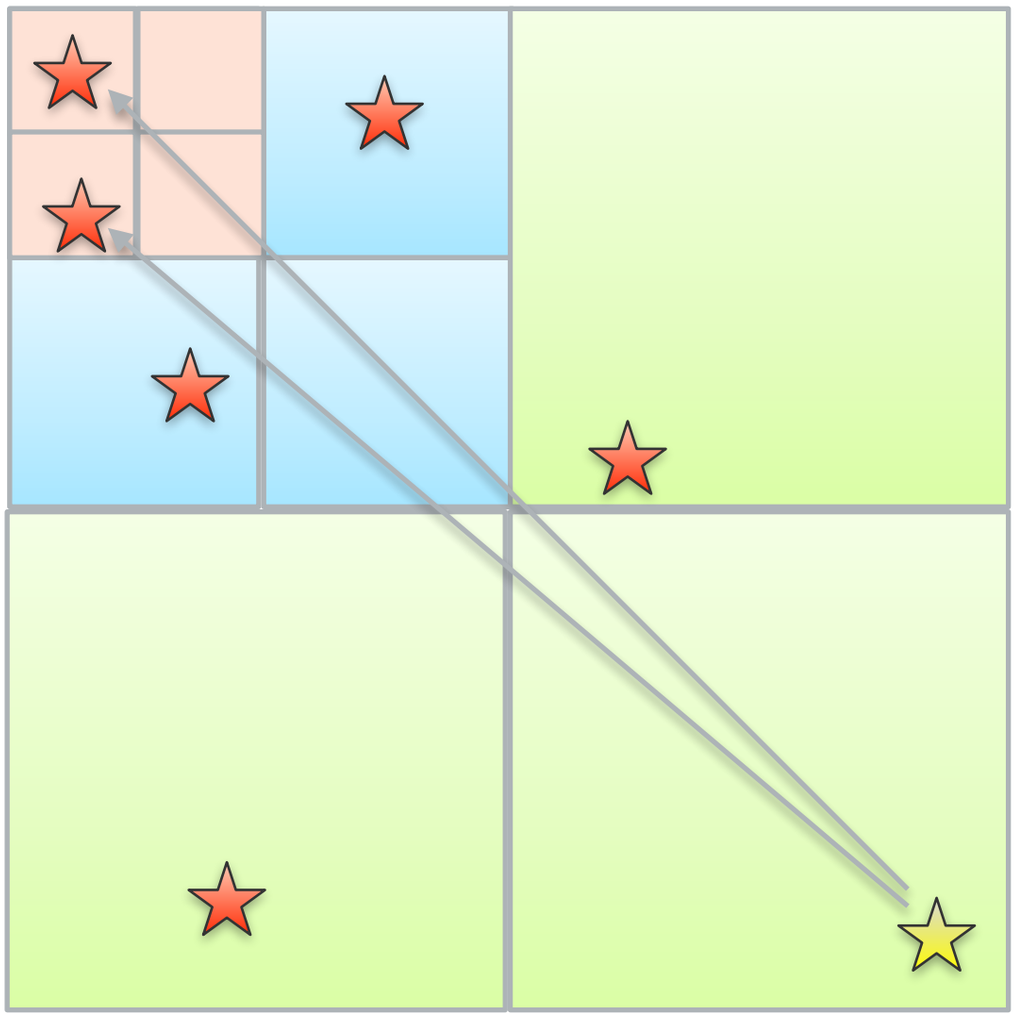
- add condition for approximation of long range vs. short range
  theta criterion



Approximate N-Body - Barnes-Hut Tree

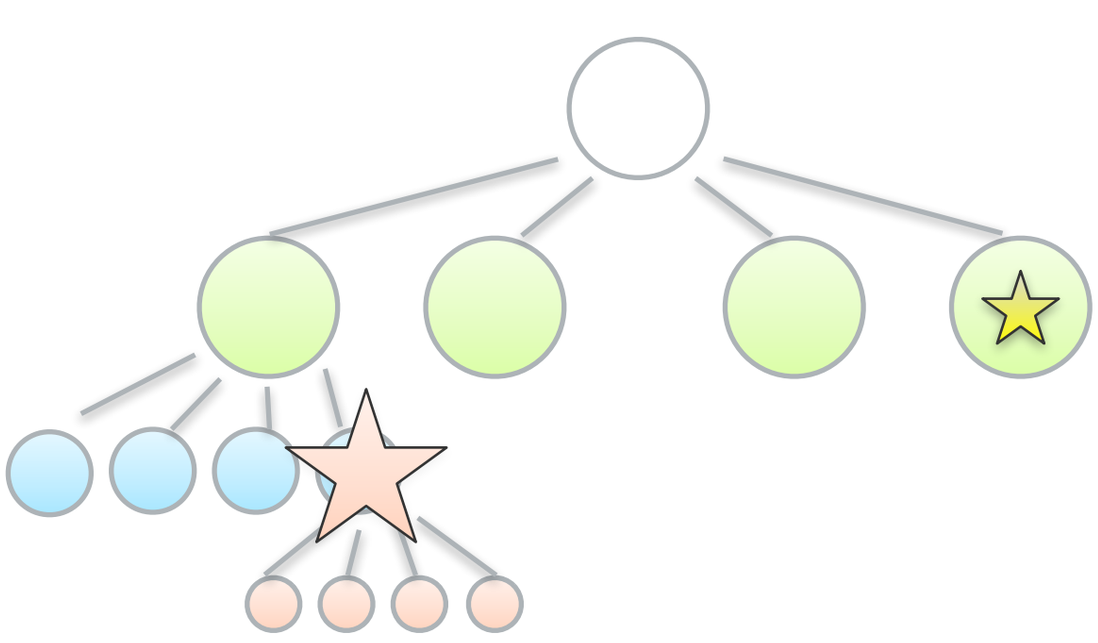
each node has $2^{dimension}$ children:
2D quadtree
3D octree

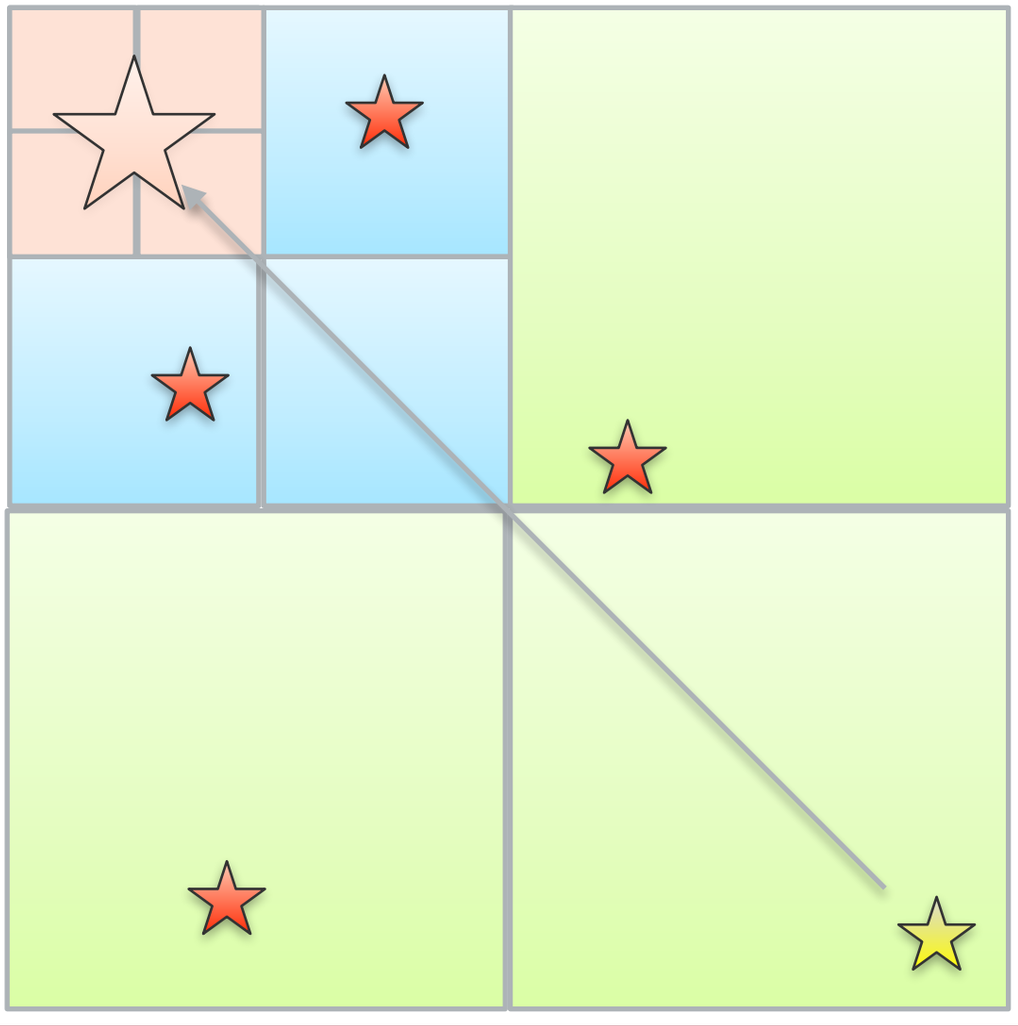




Approximate N-Body - Barnes-Hut Tree

each node has $2^{dimension}$ children:
2D quadtree
3D octree

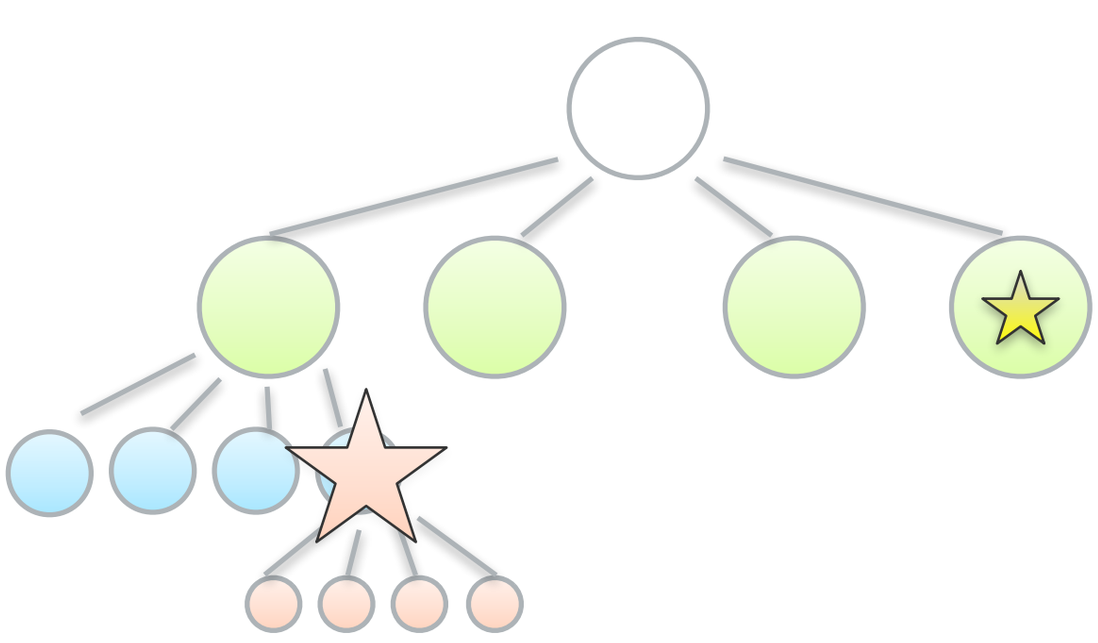


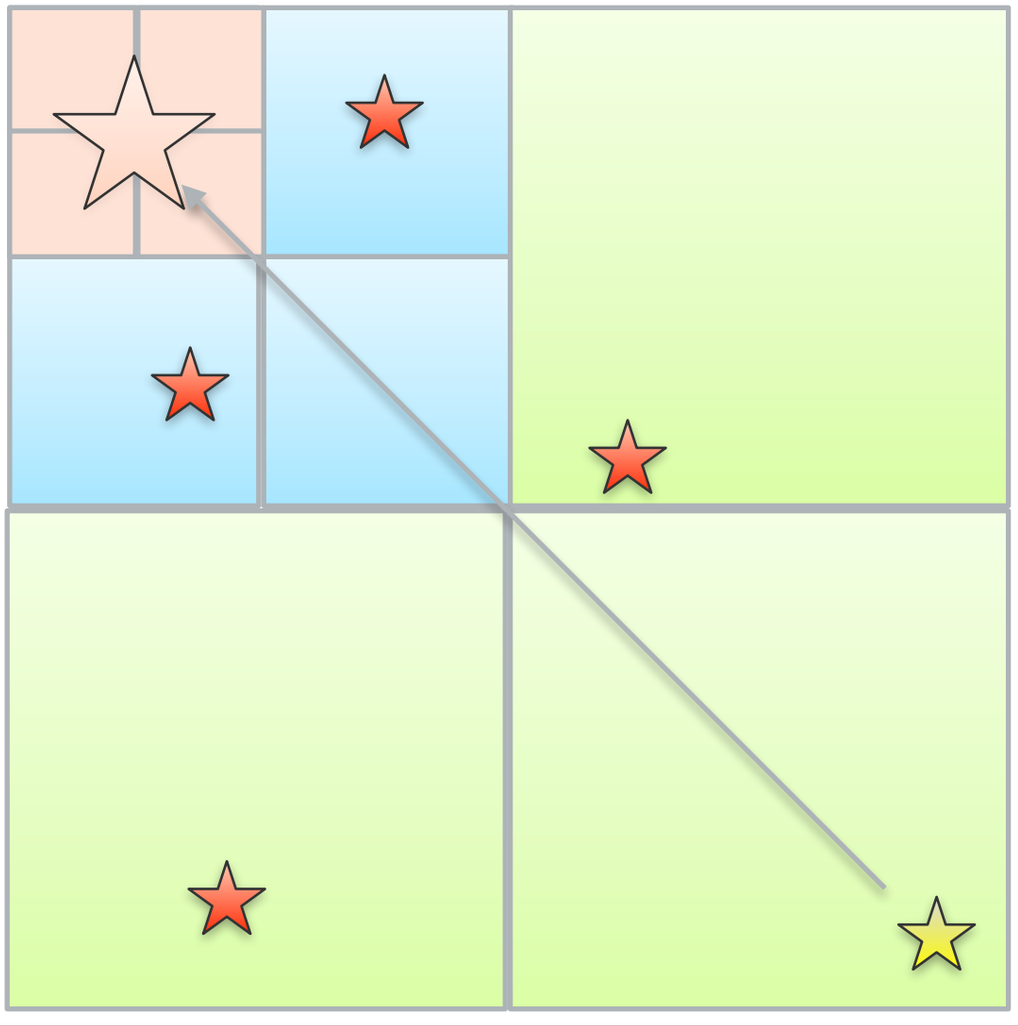


Approximate N-Body - Barnes-Hut Tree

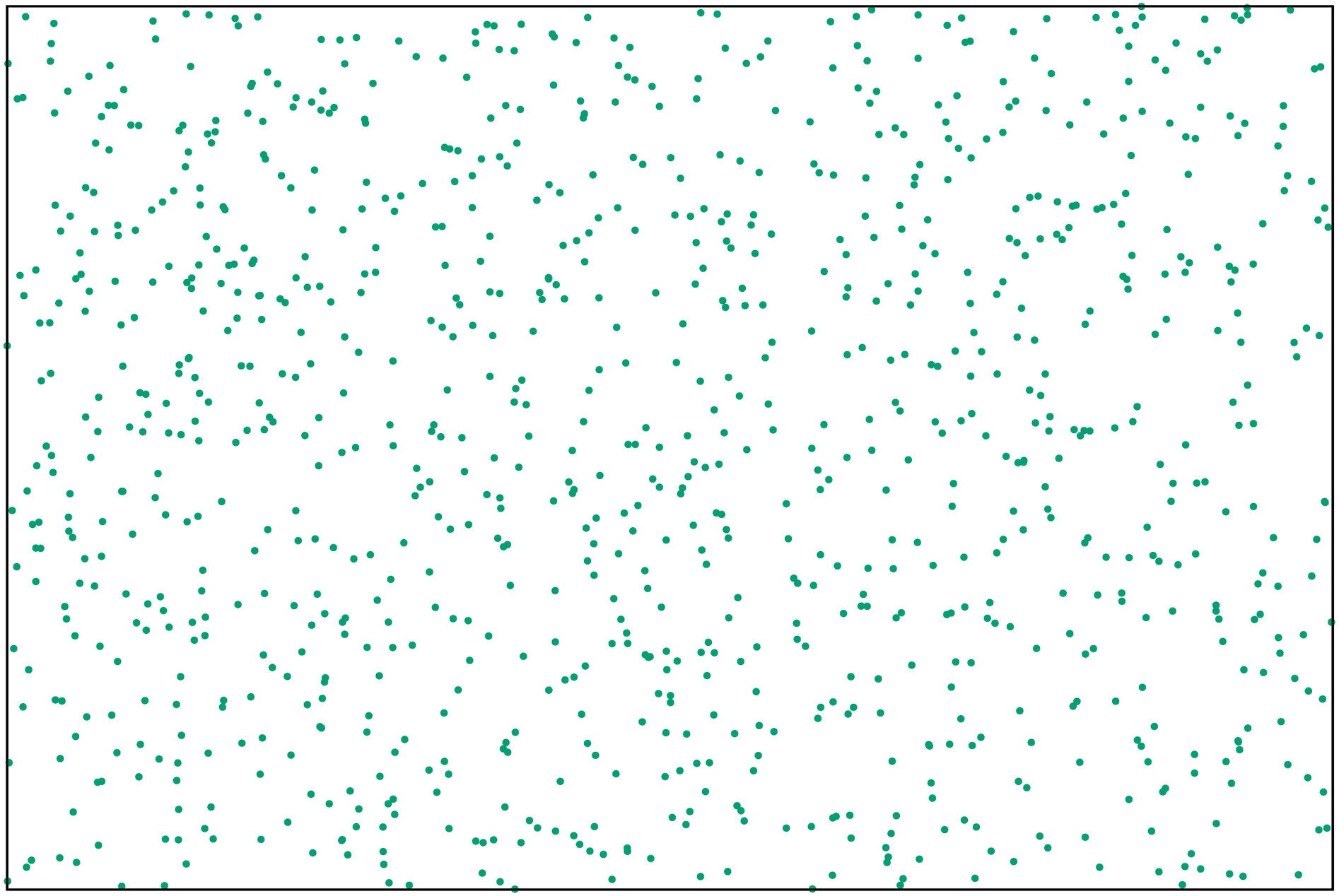
data overhead compared to direct N-Body

computational cost much lower N log(N)

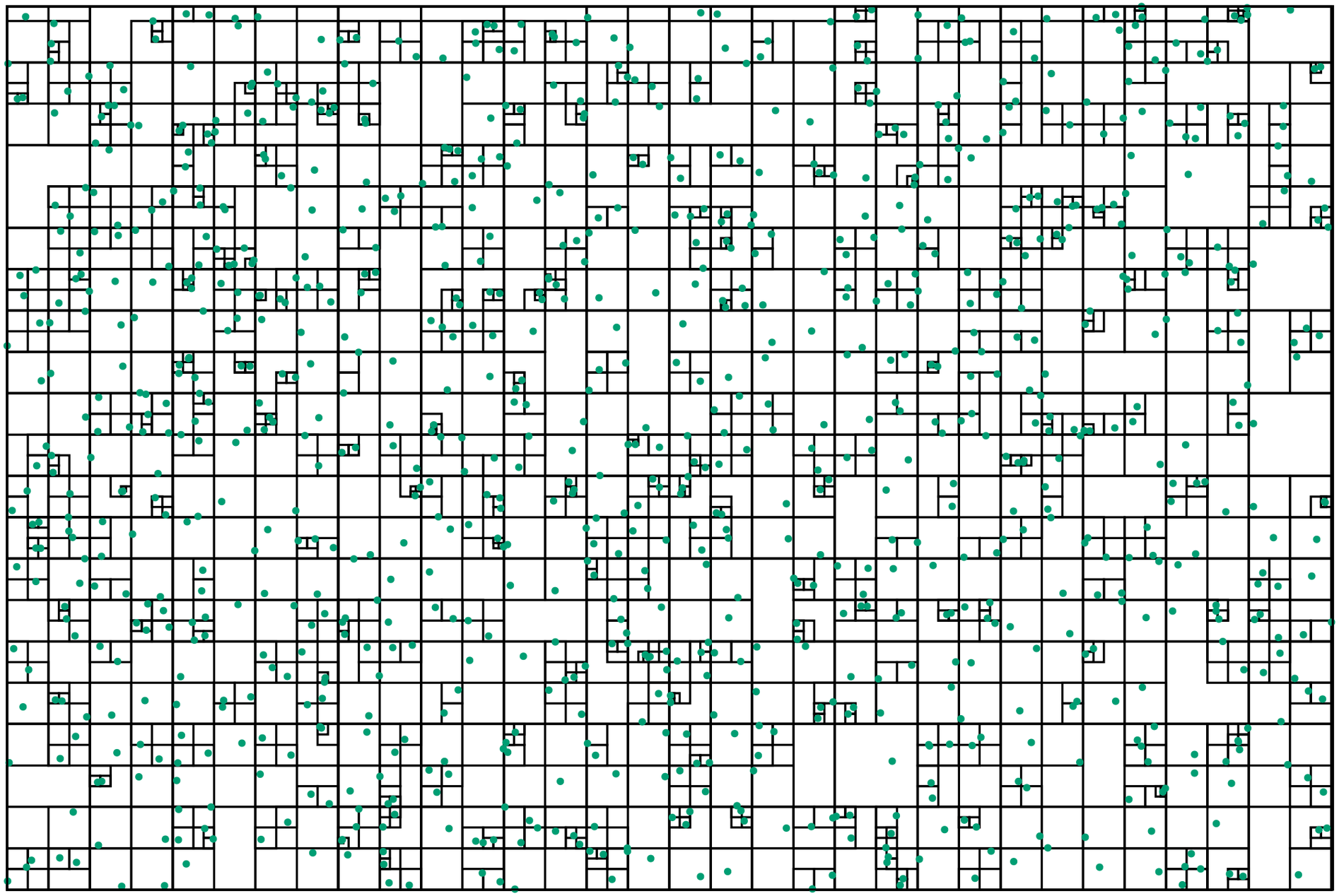




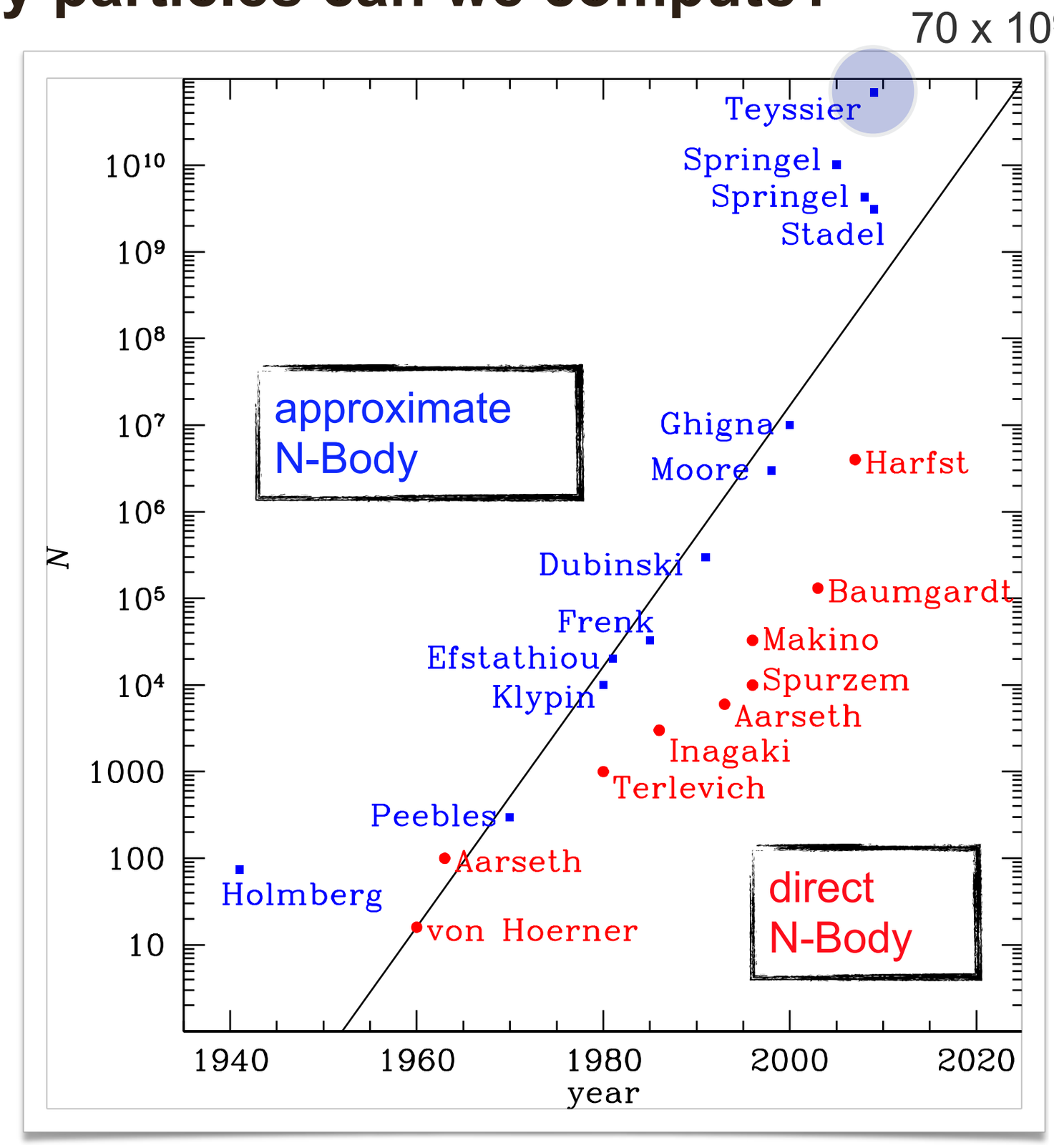
Approximate N-Body - Barnes-Hut Tree



Approximate N-Body - Barnes-Hut Tree



How many particles can we compute?



Dehnen & Read 2011

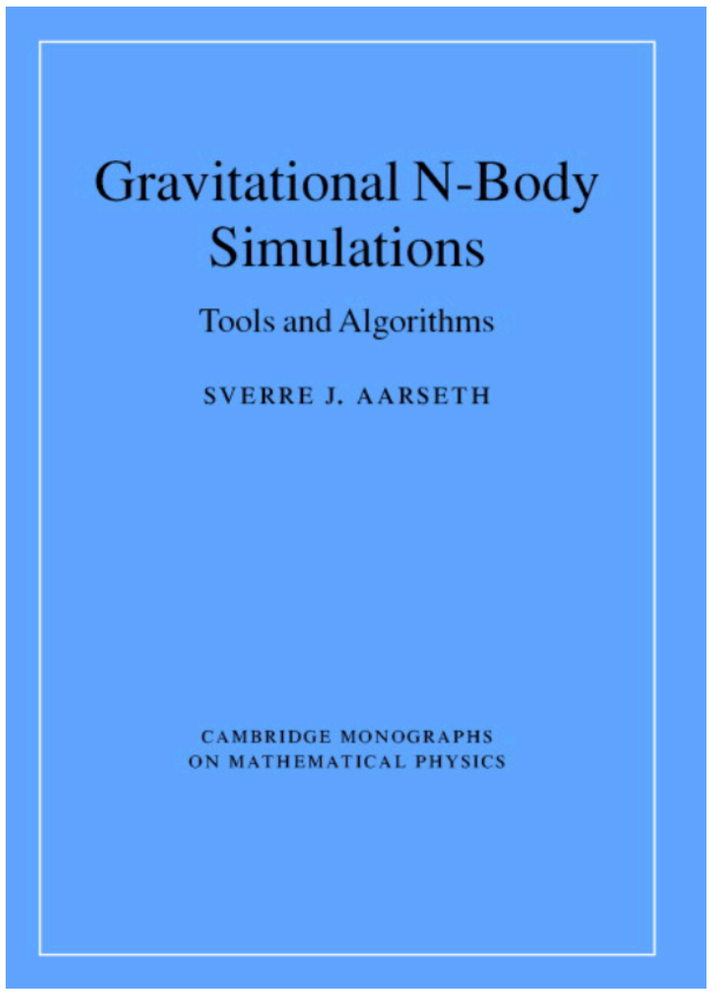
Some N-Body codes you may know

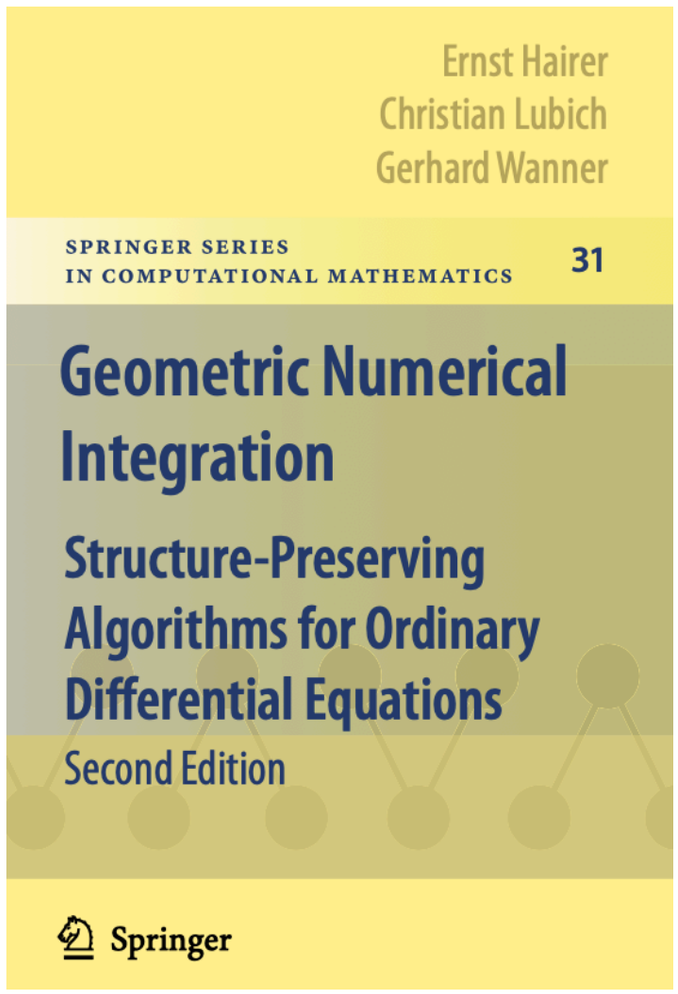
Code	Author	Application	Features
MERCURY	Chambers et al. (2000)	celestial mechanics	symplectic, newer versions from different groups, direct N-Body
NBODY	Aarseth et al. (1985)	star clusters, celestial mechanics	various versions, GPU version, direct N-Body
SyMBA	Duncan et al. (1998)	celestial mechanics	symplectic, direct N-Body
GADGET-2.0	Springel et al. (2005)	galactic dynamics	SPH tree code
AREPO	Springel et al. (2010)	galactic dynamics	moving mesh code
FLASH	Flash consortium	galactic dynamics, accretion discs, star formation	different solvers for gravity: multigrid, tree, …
REBOUND	Rein et al. (2012)	celestial mechanics	😍

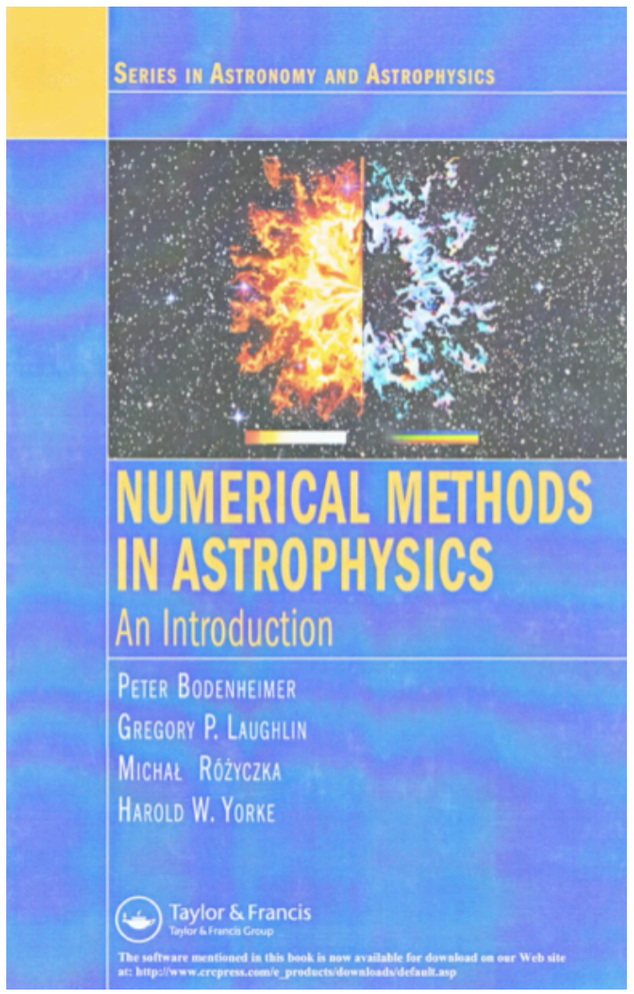
and many more…

More literature for the interested reader:

- Aarseth, Gravitational N-Body Simulations: Tools and Algorithms
- Hairer, Lubich, Wanner, Geometric Numerical Integration
- Bodenheimer, Laughlin, Rozycka, Yorke, Numerical Methods in Astrophysics
- Dehner & Read, N-body simulations of gravitational dynamics, astro-ph







REBOUND

open source multi-purpose N-body code for collisional dynamics written by Hanno Rein (University of Toronto at Scarborough)

- features
  - C (iso C99 standard) programming language
  - **python** frontend
  - modular, use it as an external library
  - OpenMP and MPI (only tree) parallelized
  - various integrators
  - active particles and tracer particles
  - Support for collisional/granular dynamics, various collision detection routines
- **very good documentation**

  https://rebound.readthedocs.io/en/latest
- soon(ish): GPU support via OpenCL

REBOUND

- modular, use it as an external library
  - your nbody simulation is an independent source file which includes `rebound.h`
  - link your binary to `librebound.so`
  - run the code

- basic structure

```
#include “rebound.h”
int main(int argc, char* argv[])
{
      struct reb_simulation *r = reb_create_simulation();
      /* choose integrator */
      r->integrator = REB_INTEGRATOR_LEAPFROG;
      /* here you would add some bodies to the simulation */
      /* start integration */
      reb_integrate(r, INFINITY);
}
```

REBOUND

- install rebound via pip
- import rebound in python scriptfile or jupyter notebook

- basic structure

```
#!/usr/bin/env python3
import numpy as np
import rebound

sim = rebound.Simulation()
sim.add(m=1.)
sim.add(m=1e-3, a=1.0)
sim.move_to_com()
sim.integrate(10*np.pi)
```

REBOUND

- different types of particles
  - active particles and testparticles

    set by `reb_simulation.testparticle_type`

    Type of the particles with an index>=N_active. 0 means particle does not influence any other particle (default), 1 means particles with index < N_active feel testparticles. Testparticles never feel each other.

```
int main(int argc, char* argv[])
{
	struct reb_simulation *r = reb_create_simulation();
	r->N_active = 2;
	(…)
	for (int i = 0; i < 1000; i++) {
		struct reb_particle testparticle = {0};
		testparticle.x = (double) i;
		reb_add(r, testparticle);
	}
	reb_integrate(r, INFINITY);
}
```

REBOUND

- different types of particles
  - active particles and testparticles

    set by `reb_simulation.testparticle_type`

    Type of the particles with an index>=N_active. 0 means particle does not influence any other particle (default), 1 means particles with index < N_active feel testparticles. Testparticles never feel each other.

```
#!/usr/bin/env python3
import rebound
import numpy as np

sim = rebound.Simulation()
sim.add(m=1.0)  # add a sun
sim.add(m=1e-3, a=1.0) # add a jupiter at earth’s orbit
N_testparticle = 1000  # add 1000 test particles
a_initial = np.linspace(1.1, 1.15, N_testparticle)
for a in a_initial:
    sim.add(a=a, f=np.random.rand*2*np.pi)  # random mean anomaly, no mass
sim.N_active = 2 # only sun and jupiterlike planet act on testparticles
```

REBOUND

- each particle can be identified exactly by a particle hash
  - add and remove particles

    ```
    void reb_add(struct reb_simulation *const r, struct
    reb_particle pt)
    int reb_remove(struct reb_simulation *const r, int
    index, int keepSorted)
    int reb_remove_by_hash(struct reb_simulation *const r,
    uint32_t hash, int keepSorted)
    ```

  - identify particles

    ```
    struct reb_particle *reb_get_particle_by_hash(struct
    reb_simulation *const r, uint32_t hash)
    ```

REBOUND

- each particle can be identified exactly by a particle hash
  - add and remove particles

```
sim.add(a=5.2, hash="Jupiter")
sim.remove(hash="Jupiter")
sim.remove(index=i)
```

REBOUND available modules

- integrators
  - REB_INTEGRATOR_IAS15
  - REB_INTEGRATOR_WHFAST
  - REB_INTEGRATOR_JANUS
  - REB_INTEGRATOR_LEAPFROG
  - REB_INTEGRATOR_SEI
  - REB_INTEGRATOR_MERCURIUS
  - REB_INTEGRATOR_SABA
  - REB_INTEGRATOR_EOS

REBOUND available modules

- integrators
  - REB_INTEGRATOR_IAS15
  - REB_INTEGRATOR_WHFAST
  - REB_INTEGRATOR_JANUS
  - REB_INTEGRATOR_LEAPFROG
  - REB_INTEGRATOR_SEI
  - REB_INTEGRATOR_MERCURIUS
  - REB_INTEGRATOR_SABA
  - REB_INTEGRATOR_EOS

IAS15 stands for Integrator with Adaptive Step-size control, 15th order. It is a vey high order, non-symplectic integrator which can handle arbitrary (velocity dependent) forces and is in most cases accurate down to machine precision, see Rein & Spiegel 2015. default

REBOUND available modules

- integrators
  - REB_INTEGRATOR_IAS15
  - REB_INTEGRATOR_WHFAST
  - REB_INTEGRATOR_JANUS
  - REB_INTEGRATOR_LEAPFROG
  - REB_INTEGRATOR_SEI
  - REB_INTEGRATOR_MERCURIUS
  - REB_INTEGRATOR_SABA
  - REB_INTEGRATOR_EOS

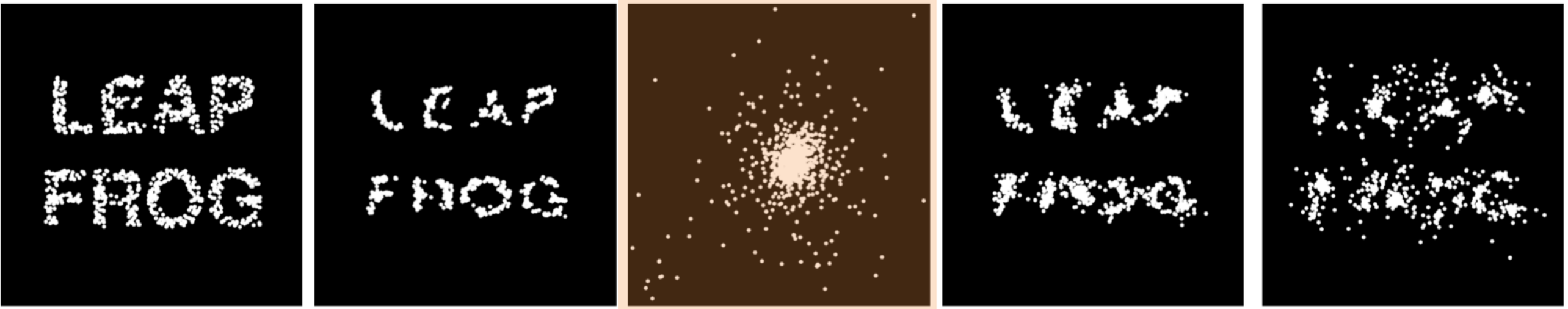
WHFast is the integrator described in Rein & Tamayo 2015, it's a second order symplectic Wisdom & Holman integrator with 11th order symplectic correctors. Good for planetary systems without collisions.

REBOUND available modules

- integrators
  - REB_INTEGRATOR_IAS15
  - REB_INTEGRATOR_WHFAST
  - REB_INTEGRATOR_JANUS
  - REB_INTEGRATOR_LEAPFROG
  - REB_INTEGRATOR_SEI
  - REB_INTEGRATOR_MERCURIUS
  - REB_INTEGRATOR_SABA
  - REB_INTEGRATOR_EOS

Janus is a bit-wise time-reversible high-order symplectic integrator using a mix of floating point and integer arithmetic, see Rein & Tamayo 2017. JANUS is explicit, formally symplectic and satisfies Liouville's theorem exactly. Its order is even and can be adjusted between two and ten.

REBOUND available modules



(a) leap-frog, $t = 0$ (b) leap-frog, $t = 35$ (c) leap-frog, $t = 500$ (d) leap-frog, $t = 965$ (e) leap-frog, $t = 1000$

(f) JANUS, $t = 0$ (g) JANUS, $t = 35$ (h) JANUS, $t = 500$ (i) JANUS, $t = 965$ (j) JANUS, $t = 1000$

Rein & Tamayo 2017

REBOUND available modules

- integrators
  - REB_INTEGRATOR_IAS15
  - REB_INTEGRATOR_WHFAST
  - REB_INTEGRATOR_JANUS
  - REB_INTEGRATOR_LEAPFROG
  - REB_INTEGRATOR_SEI
  - REB_INTEGRATOR_MERCURIUS
  - REB_INTEGRATOR_SABA
  - REB_INTEGRATOR_EOS

Leap frog, second order, symplectic.

REBOUND available modules

- integrators
  - REB_INTEGRATOR_IAS15
  - REB_INTEGRATOR_WHFAST
  - REB_INTEGRATOR_JANUS
  - REB_INTEGRATOR_LEAPFROG
  - REB_INTEGRATOR_SEI
  - REB_INTEGRATOR_MERCURIUS
  - REB_INTEGRATOR_SABA
  - REB_INTEGRATOR_EOS

SEI integrator for shearing sheet simulations, symplectic, needs OMEGA variable.

REBOUND available modules

- integrators
  - REB_INTEGRATOR_IAS15
  - REB_INTEGRATOR_WHFAST
  - REB_INTEGRATOR_JANUS
  - REB_INTEGRATOR_LEAPFROG
  - REB_INTEGRATOR_SEI
  - REB_INTEGRATOR_MERCURIUS
  - REB_INTEGRATOR_SABA
  - REB_INTEGRATOR_EOS

A hybrid integrator very similar to the one found in MERCURY. It uses WHFast for long term integrations but switches over smoothly to IAS15 for close encounters.

REBOUND available modules

- integrators
  - REB_INTEGRATOR_IAS15
  - REB_INTEGRATOR_WHFAST
  - REB_INTEGRATOR_JANUS
  - REB_INTEGRATOR_LEAPFROG
  - REB_INTEGRATOR_SEI
  - REB_INTEGRATOR_MERCURIUS
  - **REB_INTEGRATOR_SABA**
  - REB_INTEGRATOR_EOS

High-order symplectic integrator family after Laskar & Robutel (2001).

REBOUND available modules

- integrators
  - REB_INTEGRATOR_IAS15
  - REB_INTEGRATOR_WHFAST
  - REB_INTEGRATOR_JANUS
  - REB_INTEGRATOR_LEAPFROG
  - REB_INTEGRATOR_SEI
  - REB_INTEGRATOR_MERCURIUS
  - REB_INTEGRATOR_SABA
  - REB_INTEGRATOR_EOS

Embedded Operator Splitting (EOS) integrator family, Rein (2019), especially suited for primitive architectures (GPUs, FPGAs,…)

REBOUND available modules

- Gravity solvers
  - `REB_GRAVITY_COMPENSATED`
  - `REB_GRAVITY_NONE`
  - `REB_GRAVITY_BASIC`
  - `REB_GRAVITY_TREE`

REBOUND available modules

- Gravity solvers
  - `REB_GRAVITY_COMPENSATED`
  - `REB_GRAVITY_NONE`
  - `REB_GRAVITY_BASIC`
  - `REB_GRAVITY_TREE`

Direct summation with compensated summation, $O(N^2)$, default

On your computer

$$\sum_{i=1}^{10000} \frac{1}{i^2} \neq \sum_{i=10000}^{1} \frac{1}{i^2}$$

1.6448340718480651 8
1.6448340718480596 3

REBOUND available modules

- Gravity solvers
  - `REB_GRAVITY_COMPENSATED`
  - `REB_GRAVITY_NONE`
  - `REB_GRAVITY_BASIC`
  - `REB_GRAVITY_TREE`

No self-gravity.

REBOUND available modules

- Gravity solvers
  - `REB_GRAVITY_COMPENSATED`
  - `REB_GRAVITY_NONE`
  - `REB_GRAVITY_BASIC`
  - `REB_GRAVITY_TREE`

Direct summation, $O(N^2)$.

REBOUND available modules

- Gravity solvers
  - REB_GRAVITY_COMPENSATED
  - REB_GRAVITY_NONE
  - REB_GRAVITY_BASIC
  - REB_GRAVITY_TREE

Oct tree, Barnes & Hut 1986, O(N log(N)).

REBOUND available modules

- Gravity solvers
  - `REB_GRAVITY_COMPENSATED`
  - `REB_GRAVITY_NONE`
  - `REB_GRAVITY_BASIC`
  - `REB_GRAVITY_TREE`

- Choose via r->gravity

```
int main(int argc, char* argv[])
{
    struct reb_simulation *r = reb_create_simulation();
    /* choose gravity module */
    r->gravity    = REB_GRAVITY_TREE;
    (…)
    reb_integrate(r, INFINITY);
}
```

REBOUND available modules

- Gravity solvers
  - `REB_GRAVITY_COMPENSATED`
  - `REB_GRAVITY_NONE`
  - `REB_GRAVITY_BASIC`
  - `REB_GRAVITY_TREE`

- Choose via `sim.gravity`

```
#!/usr/bin/env python3
import rebound

sim = rebound.Simulation()
sim.gravity = ‘tree’
```

REBOUND available modules

- Boundary conditions
  - `REB_BOUNDARY_NONE`
  - `REB_BOUNDARY_OPEN`
  - `REB_BOUNDARY_PERIODIC`
  - `REB_BOUNDARY_SHEAR`

REBOUND available modules

- Boundary conditions
  - `REB_BOUNDARY_NONE`
  - `REB_BOUNDARY_OPEN`
  - `REB_BOUNDARY_PERIODIC`
  - `REB_BOUNDARY_SHEAR`

Particles are not affected by boundary conditions, default.

REBOUND available modules

- Boundary conditions
  - REB_BOUNDARY_NONE
  - REB_BOUNDARY_OPEN
  - REB_BOUNDARY_PERIODIC
  - REB_BOUNDARY_SHEAR

Particles are removed from the simulation if they leave the box.

REBOUND available modules

- Boundary conditions
  - REB_BOUNDARY_NONE
  - REB_BOUNDARY_OPEN
  - REB_BOUNDARY_PERIODIC
  - REB_BOUNDARY_SHEAR

Periodic boundary conditions. Particles are re-inserted on the other side if they cross the box boundaries.

REBOUND available modules

- Boundary conditions
  - `REB_BOUNDARY_NONE`
  - `REB_BOUNDARY_OPEN`
  - `REB_BOUNDARY_PERIODIC`
  - `REB_BOUNDARY_SHEAR`

Shear periodic boundary conditions. Similar to periodic boundary conditions, but ghost-boxes are moving with constant speed, set by the shear. see `example/shearing_sheet`.

REBOUND available modules

- Boundary conditions
  - `REB_BOUNDARY_NONE`
  - `REB_BOUNDARY_OPEN`
  - `REB_BOUNDARY_PERIODIC`
  - `REB_BOUNDARY_SHEAR`

- Choose via `r->boundary`

```
int main(int argc, char* argv[])
{
    struct reb_simulation *r = reb_create_simulation();
    /* choose open boundaries */
    r->boundary     = REB_BOUNDARY_OPEN;
    /* define a box for the open boundary, one box with edge length 10 */
    reb_configure_box(r, 10., 1, 1, 1)
    reb_integrate(r, INFINITY);
}
```

REBOUND available modules

- Boundary conditions
  - `REB_BOUNDARY_NONE`
  - `REB_BOUNDARY_OPEN`
  - `REB_BOUNDARY_PERIODIC`
  - `REB_BOUNDARY_SHEAR`

- Choose via `sim.boundary`

```
#!/usr/bin/env python3
import rebound

sim = rebound.Simulation()
boxsize = 1.0
sim.configure_box(boxsize)
sim.boundary = “periodic”
```

REBOUND available modules

- collision detection, assign radius to particles, choose between
  - `REB_COLLISION_NONE`
  - `REB_COLLISION_DIRECT`
  - `REB_COLLISION_LINE`
  - `REB_COLLISION_TREE`

REBOUND available modules

- collision detection
  - `REB_COLLISION_NONE`
  - `REB_COLLISION_DIRECT`
  - `REB_COLLISION_LINE`
  - `REB_COLLISION_TREE`

no collisions detection. default.

REBOUND available modules

- collision detection
  - REB_COLLISION_NONE
  - REB_COLLISION_DIRECT
  - REB_COLLISION_LINE
  - REB_COLLISION_TREE

brute force collision search, $O(N^2)$, checks for instantaneous overlaps only.

REBOUND available modules

- collision detection
  - REB_COLLISION_NONE
  - REB_COLLISION_DIRECT
  - REB_COLLISION_LINE
  - REB_COLLISION_TREE

brute force collision search, $O(N^2)$, checks for overlaps that occured during the last timestep assuming particles travelled along straight lines.

REBOUND available modules

- collision detection
  - REB_COLLISION_NONE
  - REB_COLLISION_DIRECT
  - REB_COLLISION_LINE
  - REB_COLLISION_TREE

uses the Oct tree for particle overlapping, O(N log(N)).

REBOUND available modules

- collision detection
  - `REB_COLLISION_NONE`
  - `REB_COLLISION_DIRECT`
  - `REB_COLLISION_LINE`
  - `REB_COLLISION_TREE`

rebound tracks all collisions and the user can choose between different kinds of collisional outcome:

fully elastic
inelastic
merging (conserves mass, momentum and volume)
user-defined function

REBOUND functions

- some useful functions

REBOUND functions

- heartbeat function, set via reb_simulation.heartbeat pointer
  - is called after each time step

```
int main(int argc, char* argv[])
{
    struct reb_simulation *r = reb_create_simulation();
    /* set heartbeat function */
    r->heartbeat = heartbeat;
    (…)
    reb_integrate(r, INFINITY);
}


void heartbeat(struct reb_simulation* r) {
   if (reb_output_check(r, 1000)) { // <- checks for interval of 1000 time units
       fprintf(stdout, “current time is %e\n", r->t);
   }
}
```

REBOUND functions

- functions for i/o
  - void reb_output_ascii(struct reb_simulation *r, char *filename)

    Append the positions and velocities of all particles to an ASCII file.
  - void reb_output_orbits(struct reb_simulation *r, char *filename)

    Append an ASCII file with orbital paramters of all particles.

```
void heartbeat(struct reb_simulation* r) {
    if (reb_output_check(r, 1000)) { // <- checks for interval of 1000 time units
        reb_output_orbits(r, “cool_orbital_data.txt”);
    }
}
```

REBOUND functions

- many more functions
  - calculate energy
  - calculate angular momentum
  - convert between coordinate system variables and more celestial mechanics related functions
  - print information about time and timestep
  - print information about energy
  - create initial particle distributions, e.g., plummer spheres
  - …

  - see the API documentation
    https://rebound.readthedocs.io/en/latest/c_api.html
    https://rebound.readthedocs.io/en/latest/python_api.html

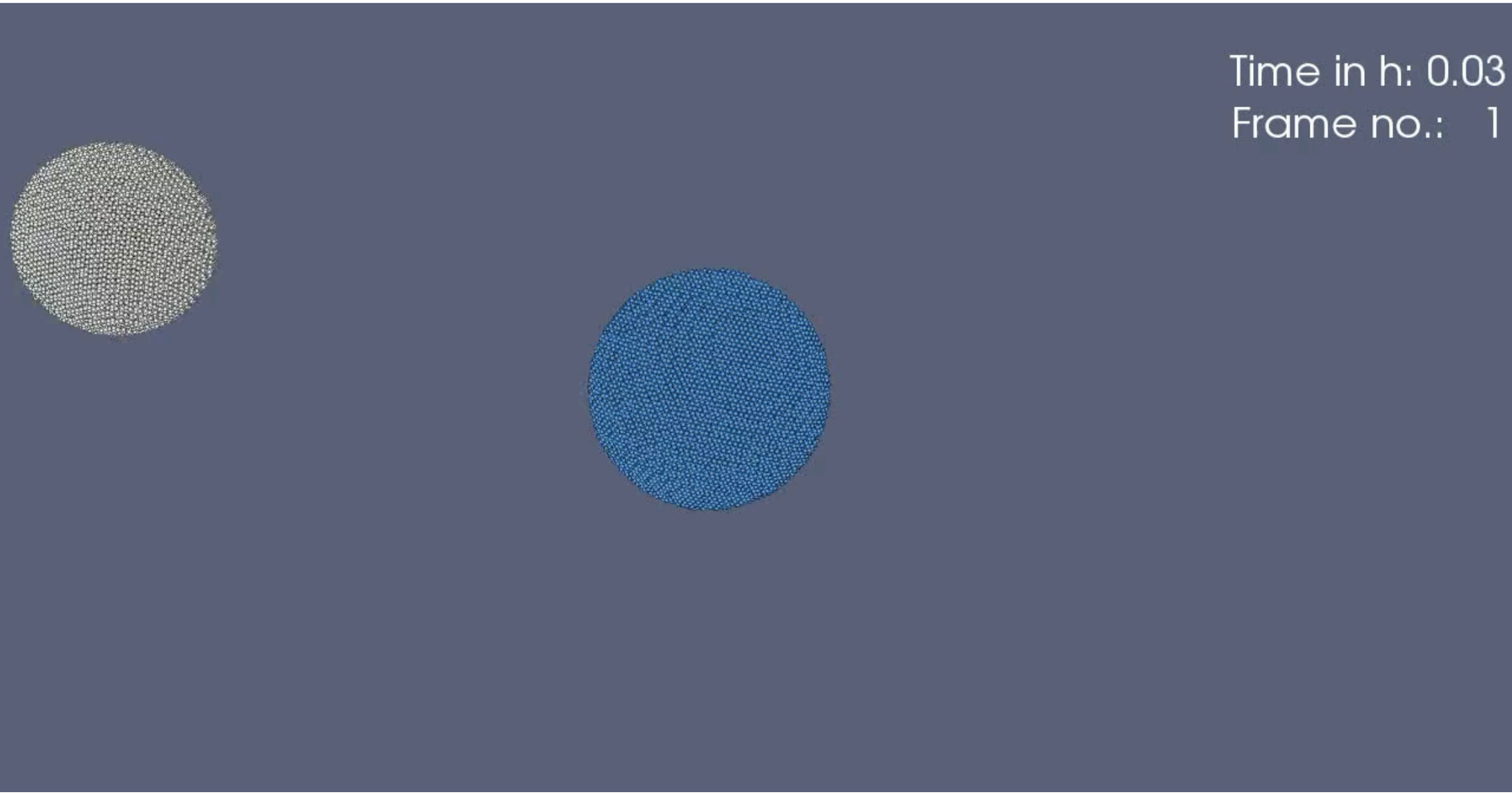
REBOUND adding more physics

- hook function for additional forces (examples/prdrag.c)

```
int main(int argc, char* argv[])
{
        struct reb_simulation *r = reb_create_simulation()
        r->force_is_velocity_dependent    = 1;
        r->additional_forces        = radiation_forces;
        (…)
        reb_integrate(r, INFINITY);
}
void radiation_forces(struct reb_simulation* r){
    struct reb_particle* particles = r->particles;
    const int N = r->N;
    for (int i=0;i<N;i++){
        const struct reb_particle p = particles[i];             // cache
        if (p.m!=0.) continue;                        // only dust particles feel radiation forces
        (…)
        // Equation (5) of Burns, Lamy, Soter (1979), Poynting-Robertson effect
        particles[i].ax += F_r*((1.-rdot/c)*prx/pr - prvx/c);
        particles[i].ay += F_r*((1.-rdot/c)*pry/pr - prvy/c);
        particles[i].az += F_r*((1.-rdot/c)*prz/pr - prvz/c);
    }
}
```

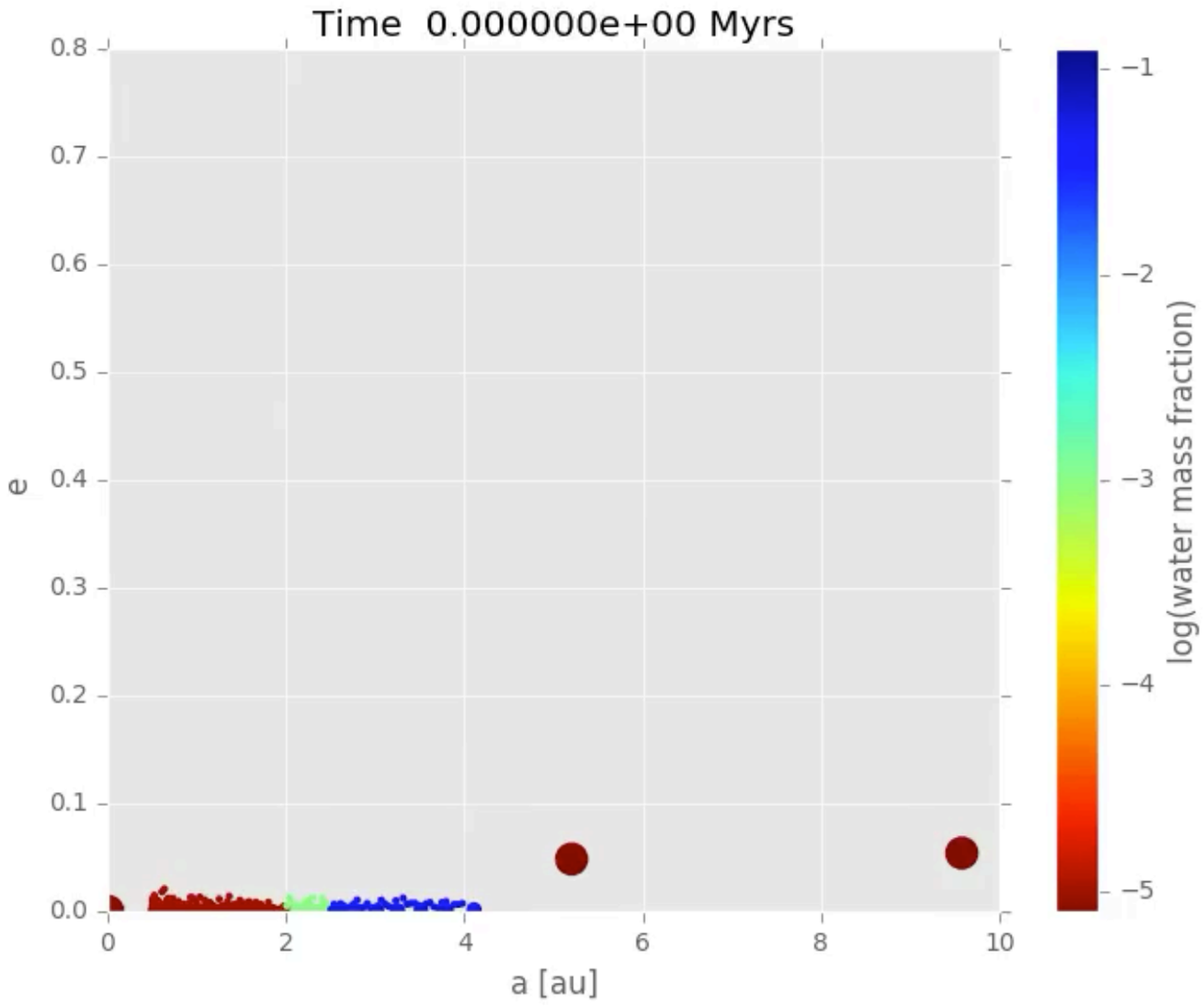
coupling REBOUND to other codes

- Late stage accretion, long term evolution via rebound, collisions via SPH code



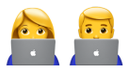
coupling REBOUND to other codes

- Late stage accretion, long term evolution via rebound, collisions via SPH code



REBOUND exercises

https://www.tat.physik.uni-tuebingen.de/~schaefer/teach/fum2020/



- Hands-on exercises part
  - Two-Body problem
  - Few-body problem
  - Saturn’s rings stability
  - Kirkwood gaps