Simulation and modeling of natural processes

Week 6: Particles and point-like objects

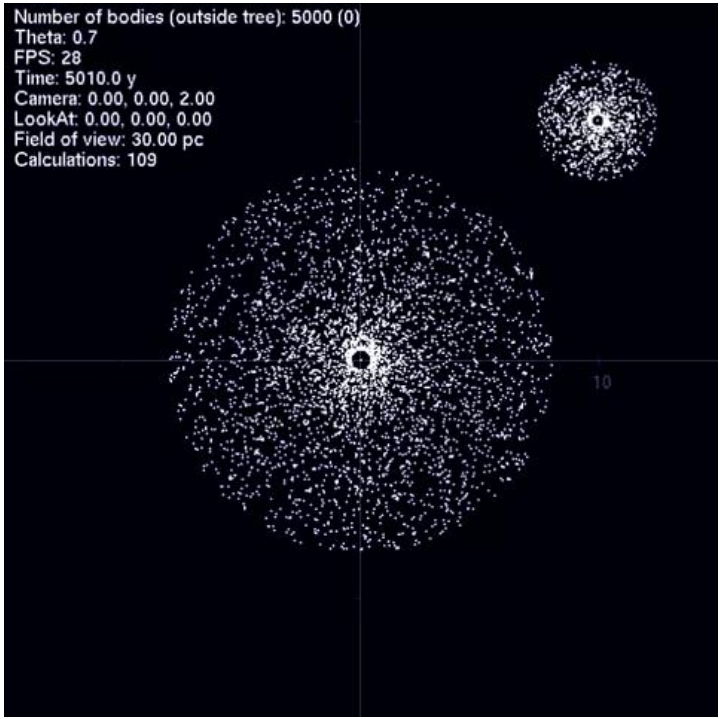
Jonas Latt

Week 6

- This week: Simulation of systems consisting of a large amount of particles, stellar bodies, or other objects.
- Equations of classical Newtonian mechanics. Numerical resolution.
- Different types of potentials: stellar bodies, molecular dynamics.
- Algorithmic complexity of the presented schemes. Workarounds and simplifications.

Particles and point-like objects: Overview

Example 1: n-body problem

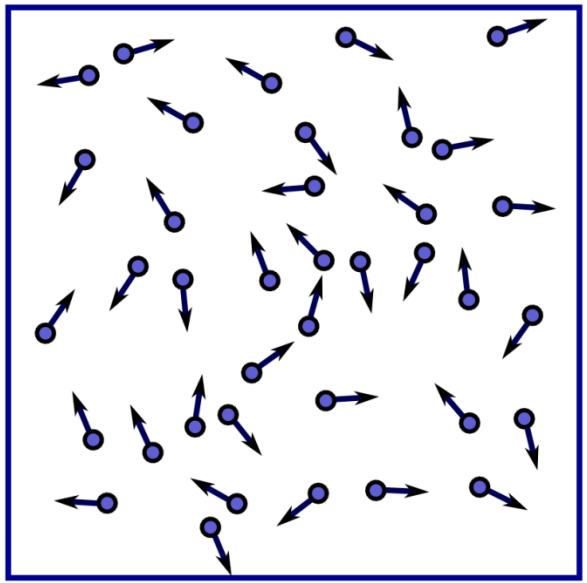


- In the n-body problem, we try to predict the individual motion of celestial bodies.
- In the left example, the bodies are the individual stars of two colliding galaxies.
- There are so many stars that we simply model them as point-like objects, which interact through the laws of gravitation.

Point-like objects

- In this lesson, we consider different physical models, which consist of a large amount of objects with similar properties.
- These objects interact with each other with given classical, Newtonian forces.
- To speed up the calculations, we model the objects as single points, described by their position, speed, and possibly some other simple properties.

Example 2: molecules in a gas



- In this example, we model a (small portion of a) gas by simulating the interaction of single gas molecules.
- Here, the the molecules interact only with other nearby molecules.
- Most of the time, they follow straight paths through empty space.
- But when they get close to other molecules, the interaction forces become important, and they collide (they change their direction and velocity).

Example 3: red blood cells

- In this case, the red blood cells in a section of a human arteries are represented by a point-like model.
- The red blood cells are embedded in a flow. Additionally to particle-particle interaction, a particle-fluid coupling is integrated in the model.
- Actual red blood cells have complicated shape and properties. They are simplified to allow for this large amount of cells (around 1 billion).

End of module

Particles and point-like objects: Overview

Coming next

Newton’s laws of motion,

Potentials and forces

Newton’s laws of motion, potentials and forces

Newton’s laws of motion

First law:	When viewed in an inertial reference frame, an object either remains at rest or continues to move at a constant velocity, unless acted upon by an external force.
Second law:	The vector sum of the forces $\boldsymbol{F}$ on an object is equal to the mass m of that object multiplied by the acceleration vector a of the object: $\boldsymbol{F} = m\boldsymbol{a}$.
Third law:	When one body exerts a force on a second body, the second body simultaneously exerts a force equal in magnitude and opposite in direction on the first body.

Forces

- In the present models, we focus on the forces, acting on an object, which originate from the interaction with the other objects (although other forces may also have their importance in practice).
- We restrict ourselved to **Newtonian mechanics**. If the interaction is non-classical (example: molecular dynamics), it is approximated by a Newtonian-type force.
- In particular, the total interaction is decomposed into a sum of interactions between **pairs of objects**.

Example: gravitational force

- As said, we consider pair-wise interaction between objects (here: stellar bodies).
- A gravitational force is directed to the body that exerts the force, and decreases with the square of the distance between the bodies:

$$\boldsymbol{F}_{ij} = \frac{G\, m_i m_j (\boldsymbol{x}_j - \boldsymbol{x}_i)}{\left|\boldsymbol{x}_j - \boldsymbol{x}_i\right|^3}$$

- The total force is the sum of the contributions from all other bodies:

$$\boldsymbol{F}_i = \sum_{j=1, j\neq i}^{N} \frac{G\, m_i m_j (\boldsymbol{x}_j - \boldsymbol{x}_i)}{\left|\boldsymbol{x}_j - \boldsymbol{x}_i\right|^3}$$

Force vs. potential

- If the force is conservative, it can be derived from a scalar potential:

$$\boldsymbol{F}_i = \frac{\partial V}{\partial \boldsymbol{x}_i}$$

- It is often more enlightening to analyze the potential, rather than the force, to understand the interaction between objects. For gravity, the potential is:

$$V = \sum_{1 \le i \le j \le N} \frac{G m_i m_j}{\left| \boldsymbol{x}_j - \boldsymbol{x}_i \right|}$$

Second example: Coulomb law

- Coulomb's inverse-square law describes the electrostatic interaction between electrically charged particles.

$$\boldsymbol{F}_{ij} = \frac{q_i q_j}{4\pi\epsilon_0} \frac{(\boldsymbol{x}_j - \boldsymbol{x}_i)}{\left|\boldsymbol{x}_j - \boldsymbol{x}_i\right|^3}$$

- Same force term as for gravity, with different constants.
- The Coulomb force can be attractive or repulsive, depending on the charge of the particles.

Third example: Lennard-Jones potential

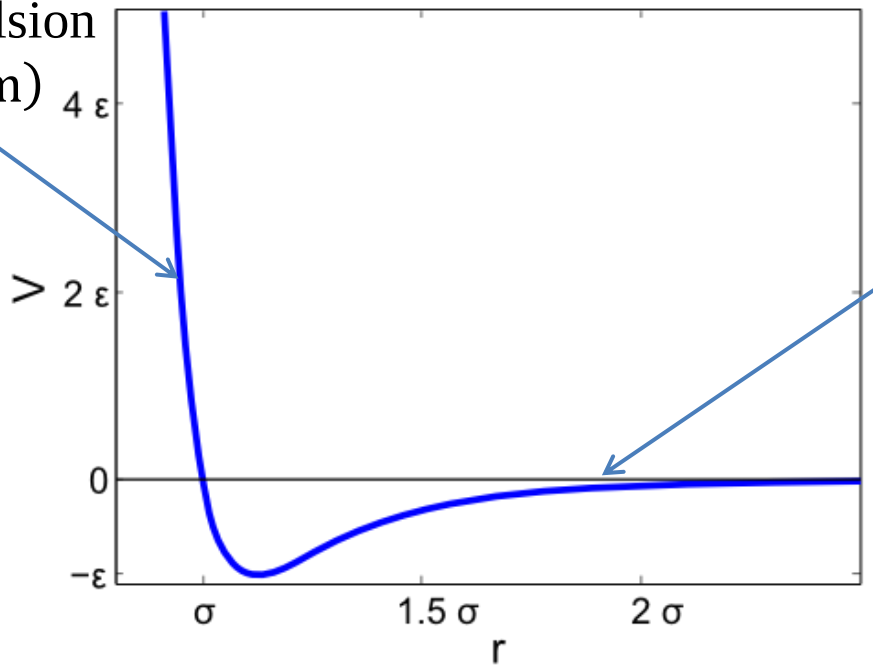
- The Lennard-Jones potential is a simple model which approximates the interaction between a pair of neutral atoms or molecules:

$$V_{ij} = 4\epsilon\left\{\left(\frac{\sigma}{\left|\boldsymbol{x}_i - \boldsymbol{x}_j\right|}\right)^{12} - \left(\frac{\sigma}{\left|\boldsymbol{x}_i - \boldsymbol{x}_j\right|}\right)^{6}\right\}$$

- The power-12 term is the Pauli repulsion.
- The power-6 term is the attractive long-range term (van der Waals force).

Lennard-Jones potential

Pauli repulsion (r^{-12} term)



Van der Waals attraction(r^{-6} term)

- $r = |\boldsymbol{x}_j - \boldsymbol{x}_i|$
- ϵ: depth of potential well

Summary: our three potentials

Force	Long-range behavior	Comment
Gravity	r^{-2}	Long-range force: can be felt at very long distances.
Coulomb	r^{-2}	Long-range force: can be felt at very long distances.
Lennard-Jones	r^{-6}	Short-range force: drops to zero very fast.

End of module

Newton’s laws of motion,

Potentials and forces

Coming next

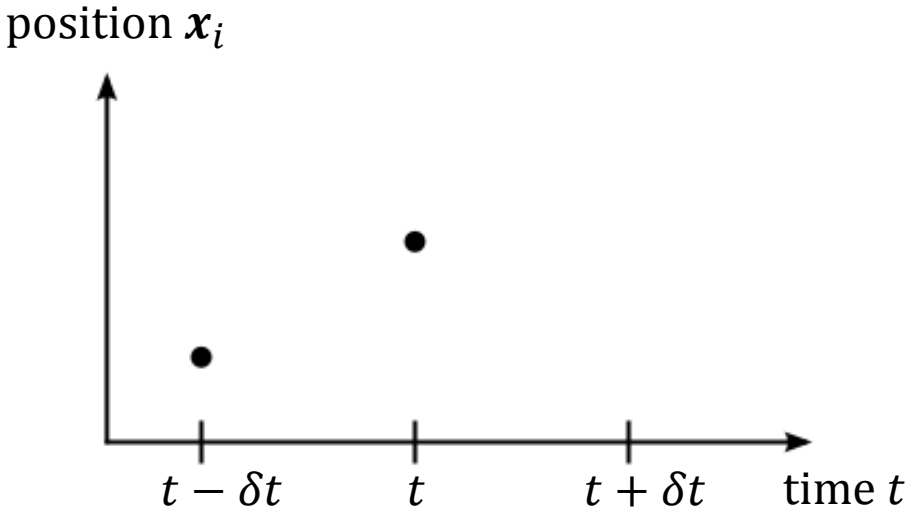
Time-integration of equations of motion

Time-integration of equations of motion

Principles of time-integration

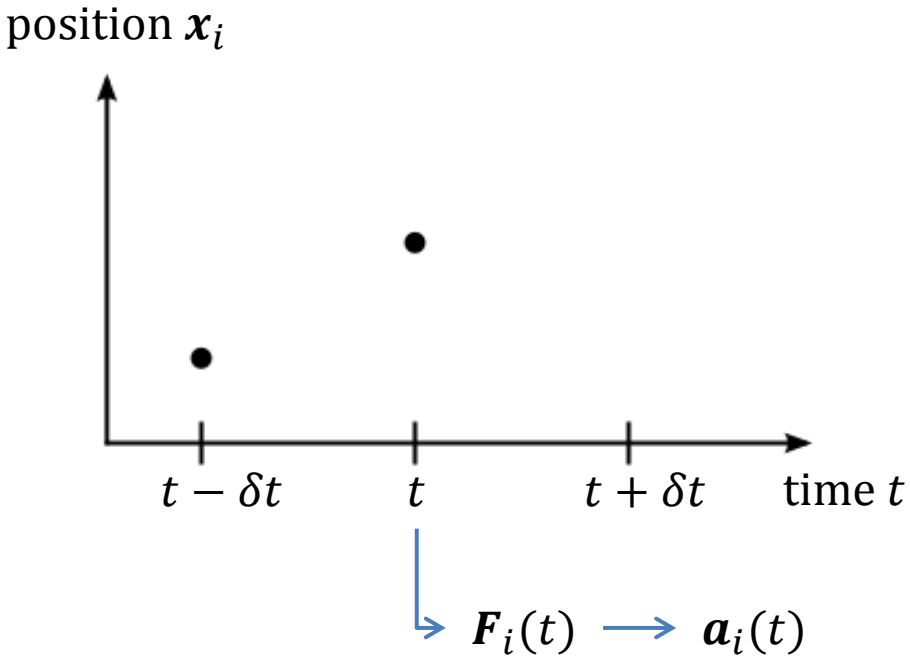
- We discretize the time axis in steps δt. An iteration of the numerical scheme takes the system from time t to time $t + \delta t$.
- At every iteration, the state of the system is fully defined by
  - the position $\boldsymbol{x}_i$ of all individual objects.
  - the velocity $\boldsymbol{v}_i$ of all individual objects.
- In the following, we introduce the Verlet algorithm, which is both efficient and accurate.

Original version of Verlet algorithm



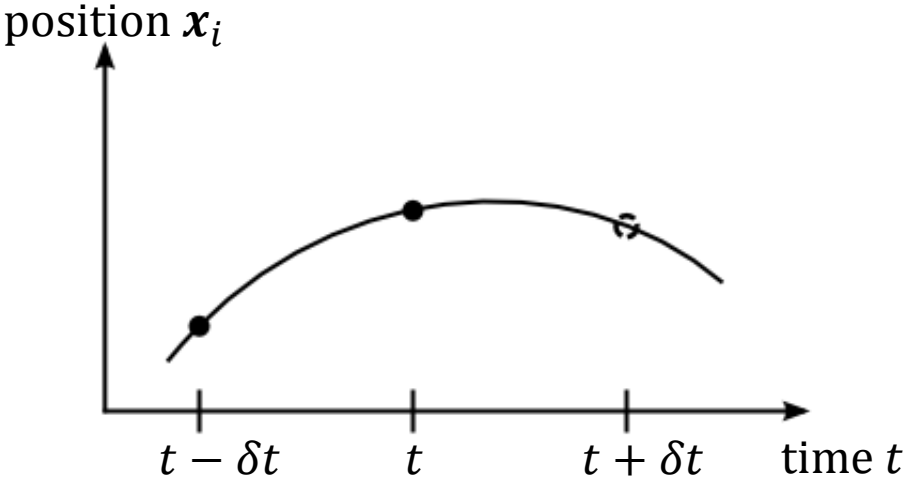
- In the original version of the Verlet algorithm, the velocity $\boldsymbol{v}_i$ of the objects is not stored.
- Instead, the position $\boldsymbol{x}_i$ of all objects is stored at two successive time steps, $t - \delta t$ and t.

Original version of Verlet algorithm



- In the original version of the Verlet algorithm, the velocity $\boldsymbol{v}_i$ of the objects is not stored.
- Instead, the position $\boldsymbol{x}_i$ of all objects is stored at two successive time steps, $t - \delta t$ and t.
- Using the position of all objects at time t, we calculate the force $\boldsymbol{F}_i(t)$, and from Newton's law the acceleration $\boldsymbol{a}_i(t)$ on the *i*-th object.

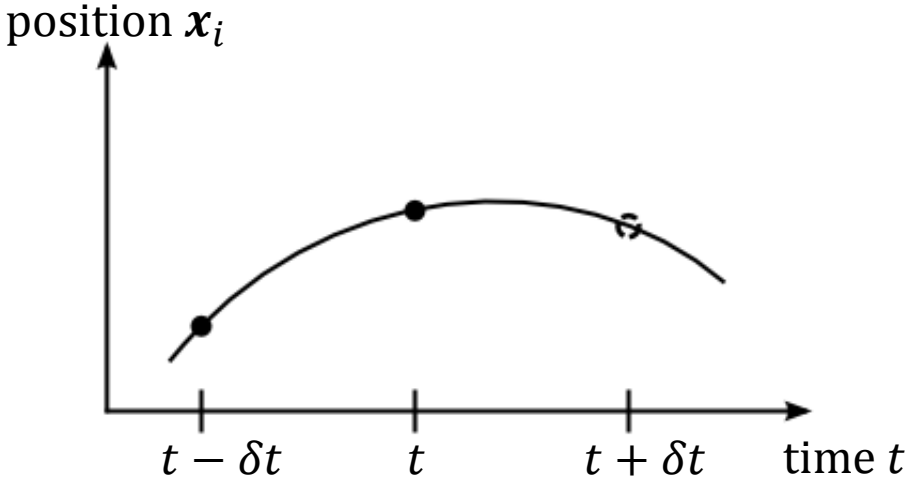
Original version of Verlet algorithm



Through a finite-difference scheme, the acceleration is related to the position with second-order accuracy:

$$\boldsymbol{a_i}(t) \approx \frac{1}{\delta t^2}\left(\boldsymbol{x}_i(t+\delta t) - \boldsymbol{x}_i 2(t) + \boldsymbol{x}_i(t-\delta t)\right)$$

Original version of Verlet algorithm



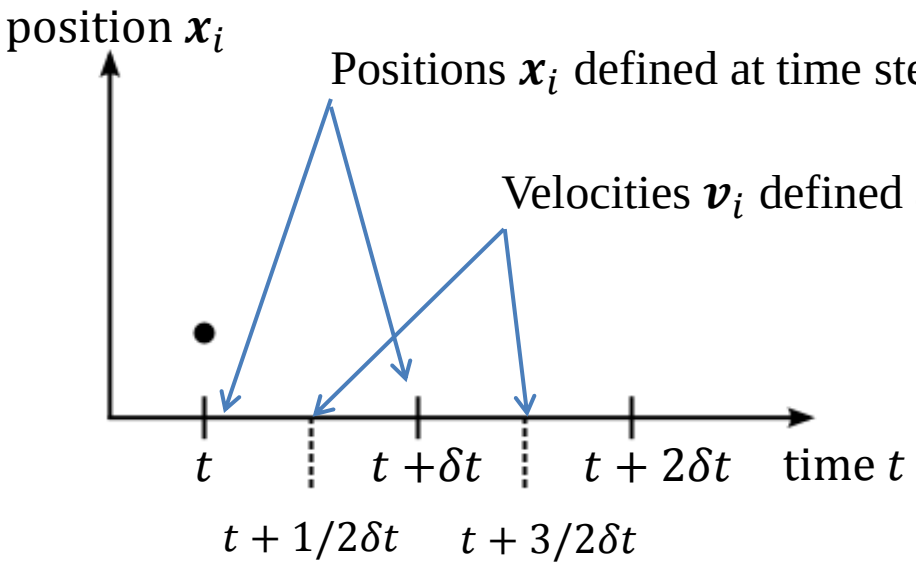
We now solve this equation for the next position $\boldsymbol{x}_i(t + \delta t)$:

$$\boldsymbol{a_i}(t) \approx \frac{1}{\delta t^2}(\boldsymbol{x}_i(t + \delta t) - \boldsymbol{x}_i 2(t) + \boldsymbol{x}_i(t - \delta t))$$

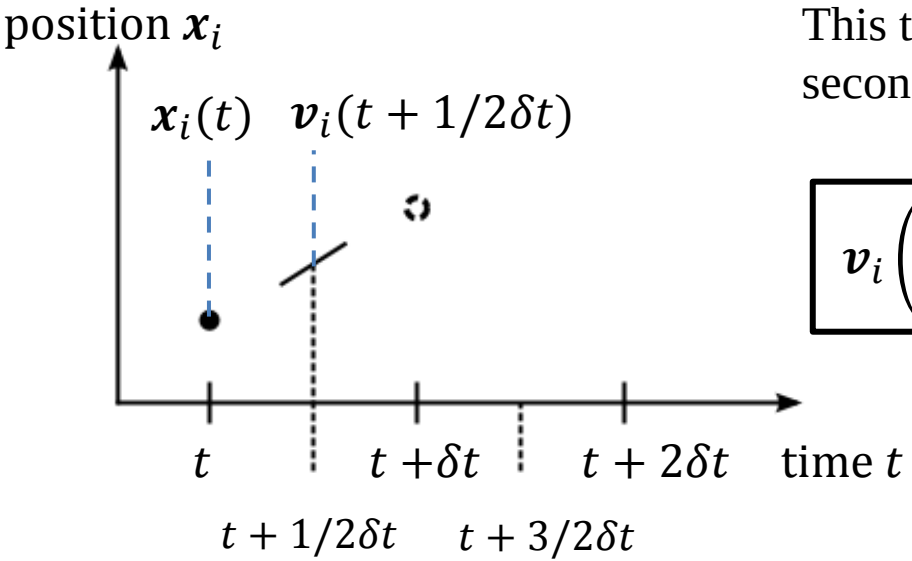
$$\boldsymbol{x}_i(t + \delta t) \approx \delta t^2\, \boldsymbol{a}_i(t) + 2\boldsymbol{x}_i(t) + \boldsymbol{x}_i(t - \delta t)$$

Verlet («leap-frog») algorithm

Idea:



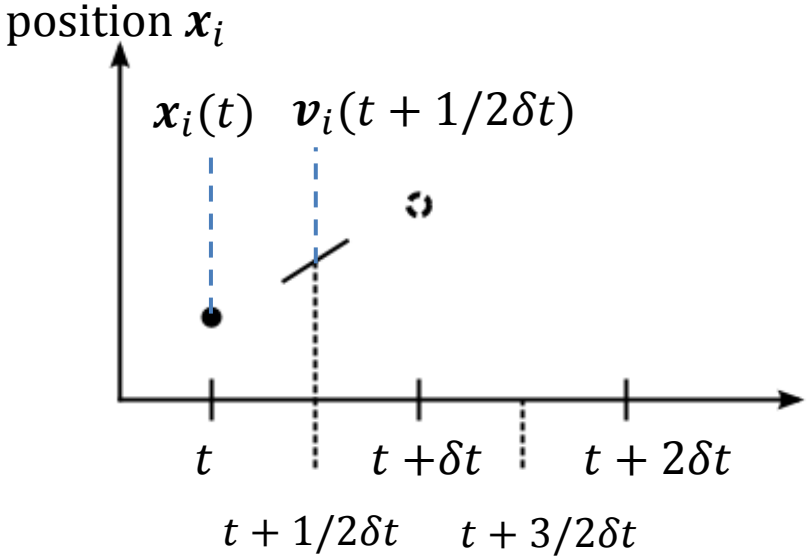
Verlet («leap-frog») algorithm



This time, we approximate the velocity with second-order accuracy:

$$\boldsymbol{v}_i\left(t + \frac{1}{2\delta t}\right) \approx \frac{1}{\delta t}(\boldsymbol{x}_i(t + \delta t) - \boldsymbol{x}_i(t))$$

Verlet («leap-frog») algorithm

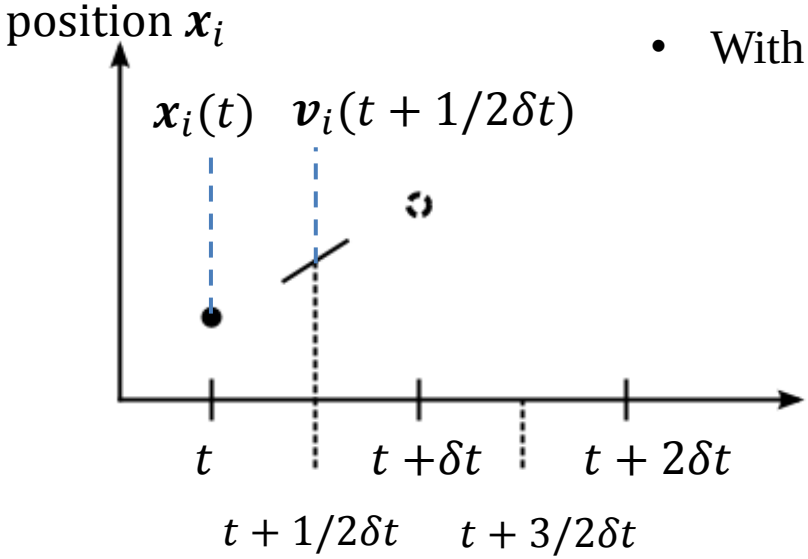


And solve for $\boldsymbol{x}_i(t + \delta t)$:

$$\boldsymbol{v}_i\left(t + \frac{1}{2}\delta t\right) \approx \frac{1}{\delta t}(\boldsymbol{x}_i(t + \delta t) - \boldsymbol{x}_i(t))$$

$$\boldsymbol{x}_i(t + \delta t) \approx \boldsymbol{x}_i(t) + \delta t \boldsymbol{v}_i\left(t + \frac{1}{2}\delta t\right)$$

Verlet («leap-frog») algorithm



- From all $\boldsymbol{x}_i(t + \delta t)$, we find $\boldsymbol{a}_i(t + \delta t)$.
- With the same scheme we find $\boldsymbol{v}_i\left(t + \frac{3}{2}\delta t\right)$:

$$\boldsymbol{x}_i(t + \delta t) \approx \boldsymbol{x}_i(t) + \delta t \boldsymbol{v}_i\left(t + \frac{1}{2}\delta t\right)$$

$$\boldsymbol{v}_i\left(t + \frac{3}{2}\delta t\right) \approx \boldsymbol{v}_i\left(t + \frac{1}{2}\delta t\right) + \delta t \boldsymbol{a}_i(t + \delta t)$$

End of module

Time-integration of equations of motion

Coming next

The Lennard-Jones potential:
Introducing a cut-off distance

The Lennard-Jones potential: Introducing a cut-off distance

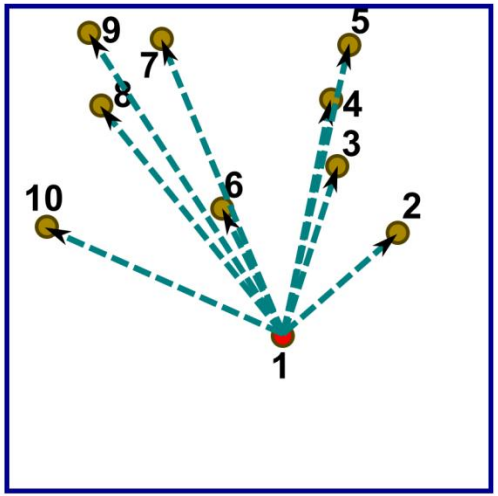
Calculation of a potential: complexity

To solve for N point-like objects, we need to

- Compute the system's time evolution iteratively (see next module).
- At each time iteration, compute the force acting on each object, i.e. compute N force terms.
- For each of the N force terms, compute the force from each of the N-1 other objects.

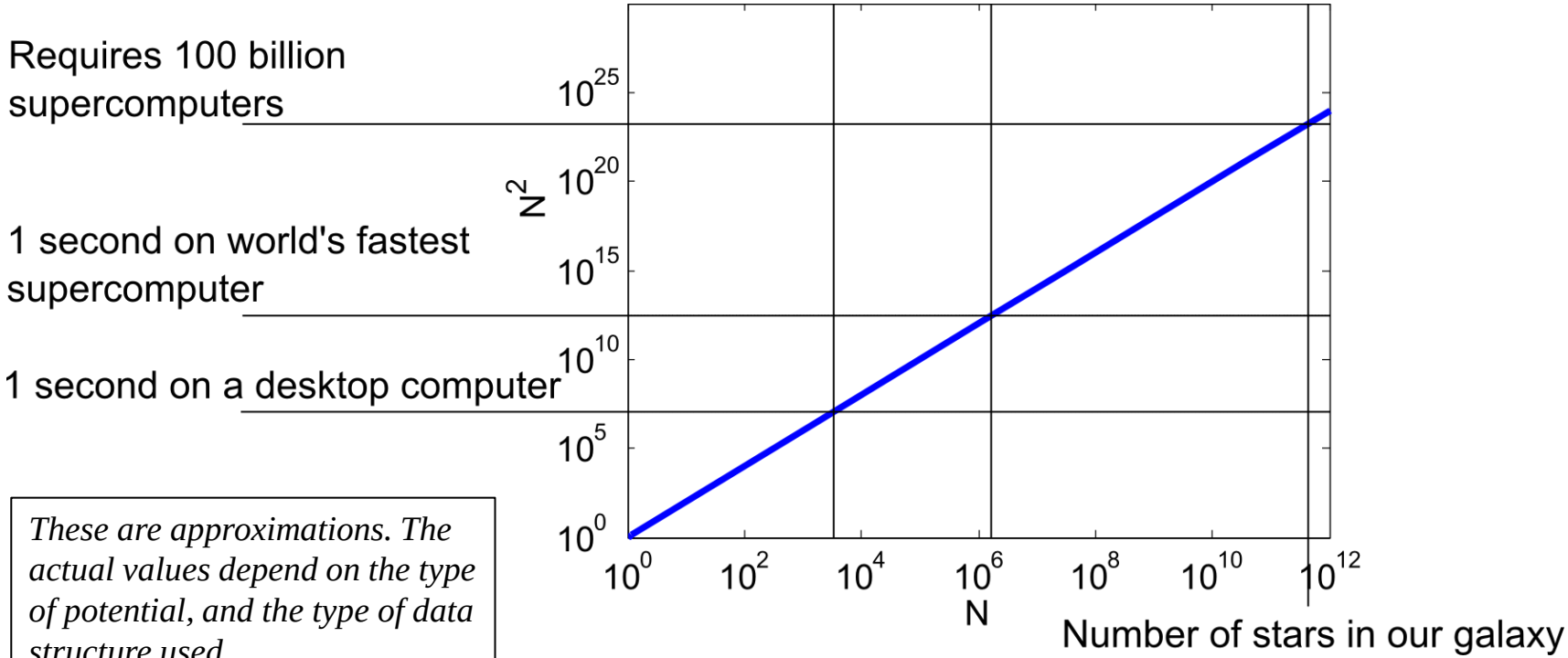
Total number of operations: $N(N-1)$

Without simplification, the calculation of the forces at a given time step has complexity N^2



(N-1) forces acting on a single object.

Can we compute N^2 interactions in 1 second?

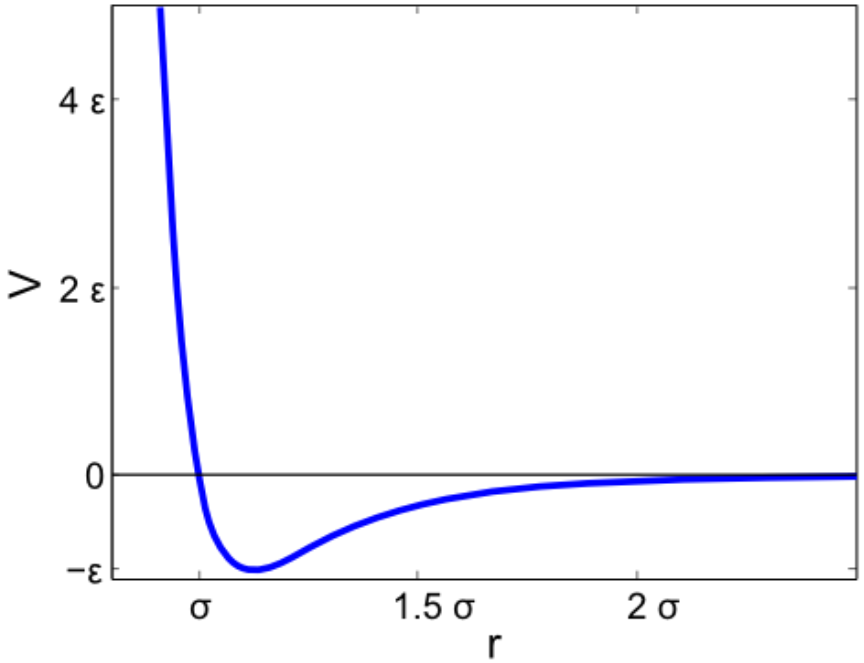


These are approximations. The actual values depend on the type of potential, and the type of data structure used.

Algorithmic complexity

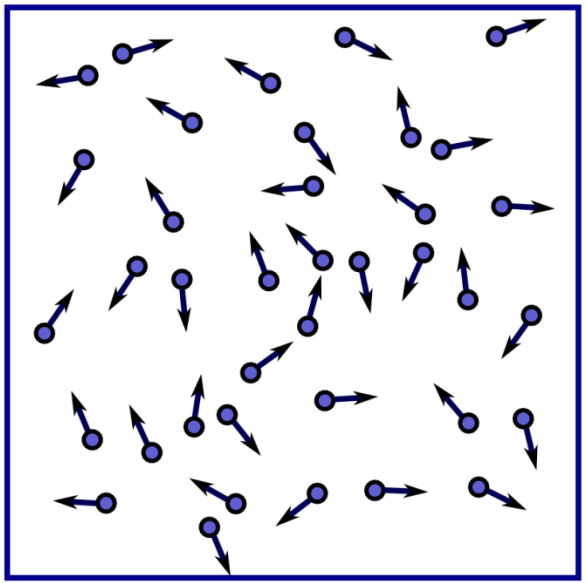
- The algorithmic complexity of solving a particle / n-body system through a brute-force approach is N^2.
- This makes it impossible to simulate a system with more than about a million objects, even on a high-end supercomputer.
- We need strategies to reduce the amount of calculations.

Back to Lennard-Jones potential



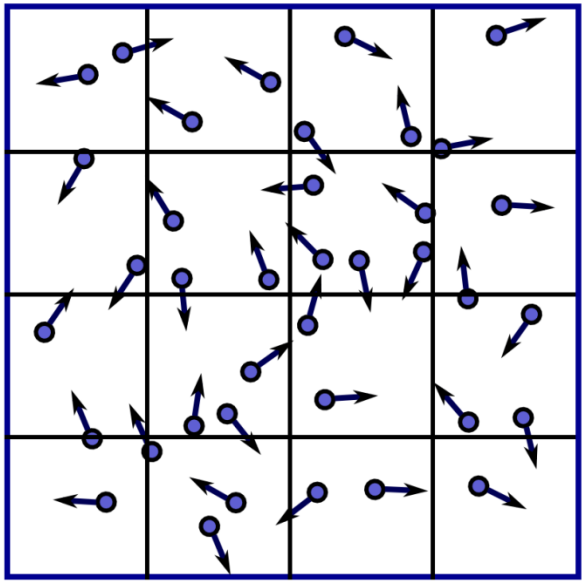
- The potential drops to zero rapidly, as the far-range contains a r^{-6} term.
- At a distance $d_c = 2.5\sigma$, its value is just about 1% of ϵ.
- Idea: we introduce d_c as a cut-off distance, and neglect all particles that are further away, to reduce the computational cost.

But how to find the nearby particles?



- If we were to check all particles to find out if they are within the cutoff length, we would need to do N checks per particle: we gain nothing.

But how to find the nearby particles?



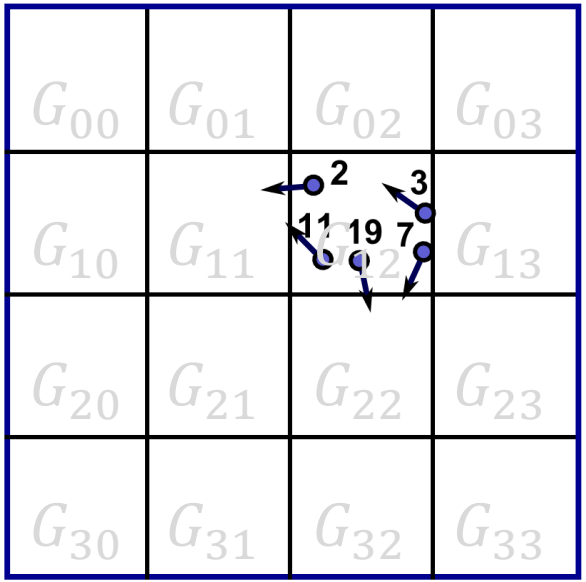
- If we were to check all particles to find out if they are within the cutoff length, we would need to do N checks per particle: we gain nothing.
- Solution: grid method. We overlap an equal-sized grid to the space of the particles.

Grid method

G_{00}	G_{01}	G_{02}	G_{03}
G_{10}	G_{11}	G_{12}	G_{13}
G_{20}	G_{21}	G_{22}	G_{23}
G_{30}	G_{31}	G_{32}	G_{33}

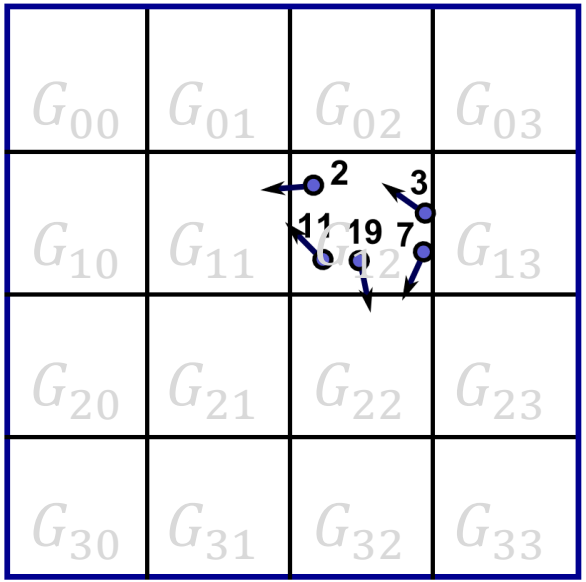
- The grid is represented by a matrix. Each matrix element G_{ij} represents a grid cell.

Grid method



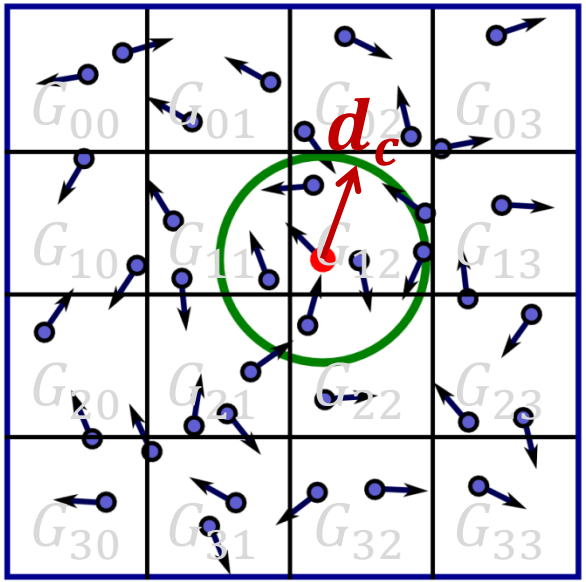
- The grid is represented by a matrix. Each matrix element G_{ij} represents a grid cell.
- Example: cell G_{12} has 5 particles with ID 2, 3, 7, 11, and 19. The corresponding matrix element contains a list with these IDs: $G_{12} = [2,3,7,11,19]$
- We only store the Ids in the matrix. The positions and velocities of a particle with given ID are stored elsewhere.

Grid method: updating the grid



- Whenever a particles changes its position, we reassign it to its proper grid cell.
- Computing the cell to which a particle belongs is a constant-time operation.
- Reassigning all particles to their grid cell is, all in all, an O(N) operation: this is OK.

Grid method: using the grid



- Using the grid, we easily find all particles within the cutoff distance.
- Example: to find particles with distance d_c of the red particle, we consider all particles in the lists of G_{11}, G_{12}, G_{21}, and G_{22}.
- We then perform an additional check on them to verify if they fall within cutoff distance.

Grid method: complexity

- If a single cell is larger, in width, than a cutoff distance, we need to check at most 9 cells (2D), or 27 cells (3D).
- If we have N_c cells in total, the average number of checks are $N \frac{9\,N}{N_c}$.
- If $N_c$is of the order of N (which is the case if cutoff-length is much smaller than the full domain), the algorithm has a complexity O(N).

End of module

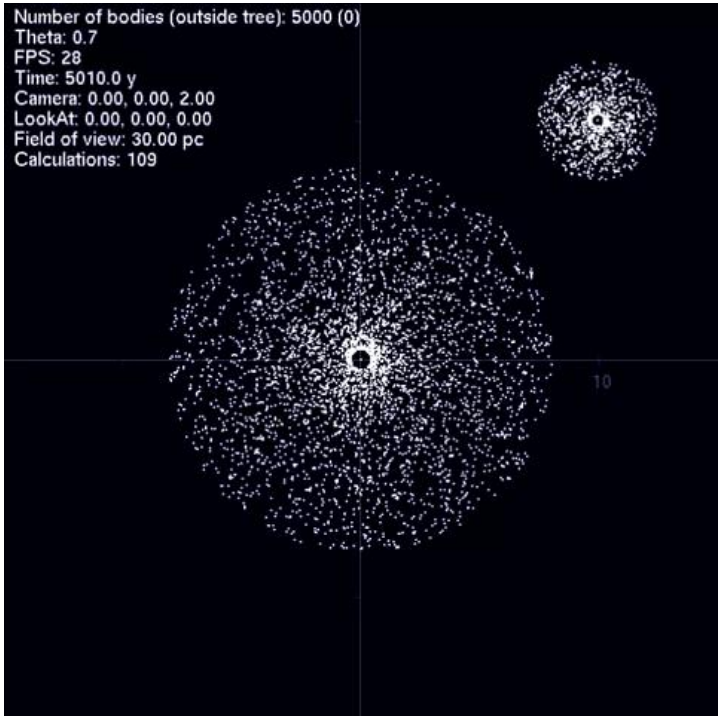
The Lennard-Jones potential:
Introducing a cut-off distance

Coming next

The n-body problem:
Evaluation of gravitational forces

The n-body problem: Evaluation of gravitational forces

The n-body problem

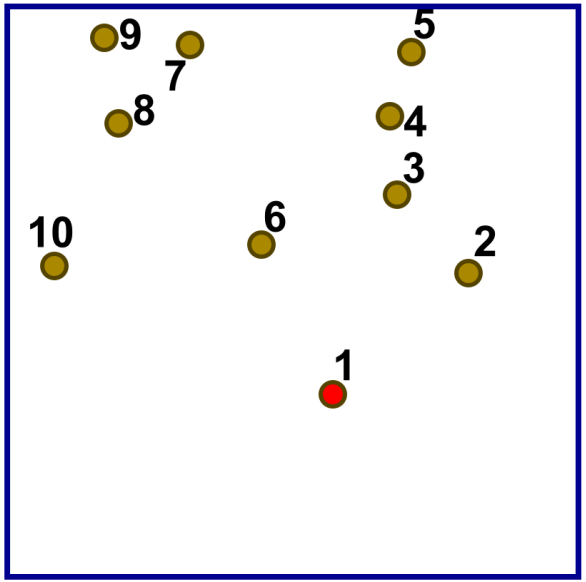


- In the n-body problem, we consider gravitational forces between all objects, i.e. stellar bodies (here: stars).
- Gravitational forces act at long distances: the two galaxies rotate around their common center-of-mass.
- A cut-off distance makes no sense here: all interactions must be accounted for.

Simplification : tree methods

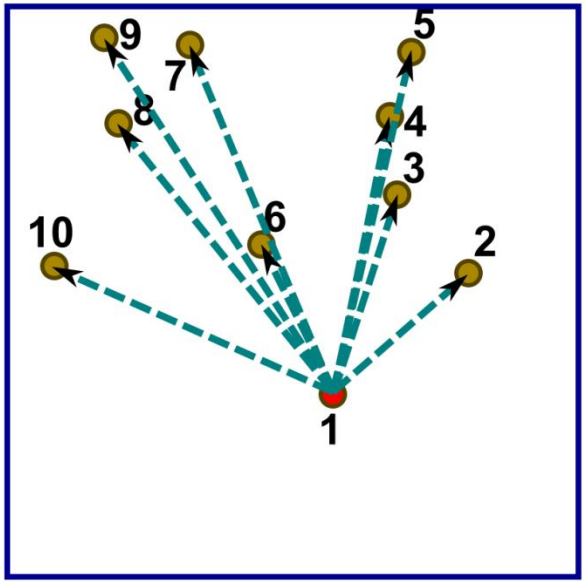
- Idea: if a group of bodies is sufficiently far away, model them as a single body, positioned at their center of mass.
- Example: to simulate the interaction between many galaxies, you don’t consider all stars of each galaxy, but consider a galaxy itself as a single body.
- In tree methods, this idea is generalized to regroup portions of space and speed up calculations.

Example: 10 interacting bodies



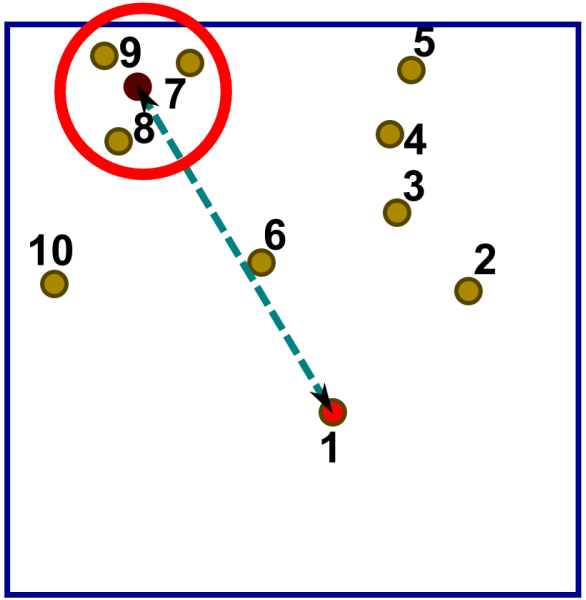
- We'd like to compute the resulting gravitational force acting on body 1.

Example: 10 interacting bodies



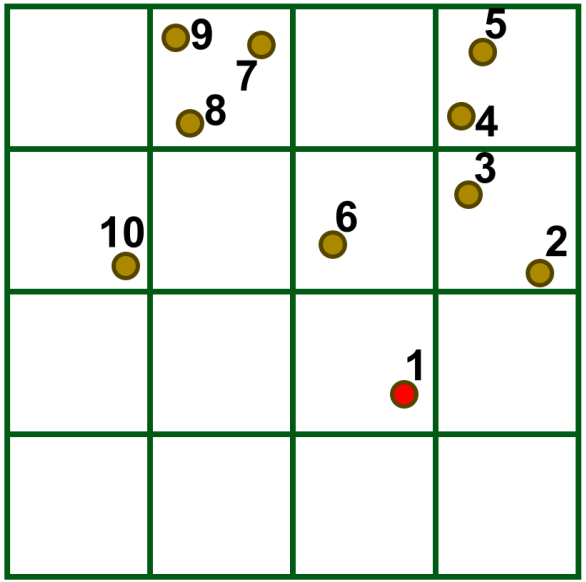
- We’d like to compute the resulting gravitational force acting on body 1.
- Exact result: compute the contribution from the other 9 bodies, and sum them up.

Faster: treat group as single body



- Idea: speed up the results by treating a group of bodies as a single body, located at the group's center of mass.
- This is a reasonable approximation if the group is far away.

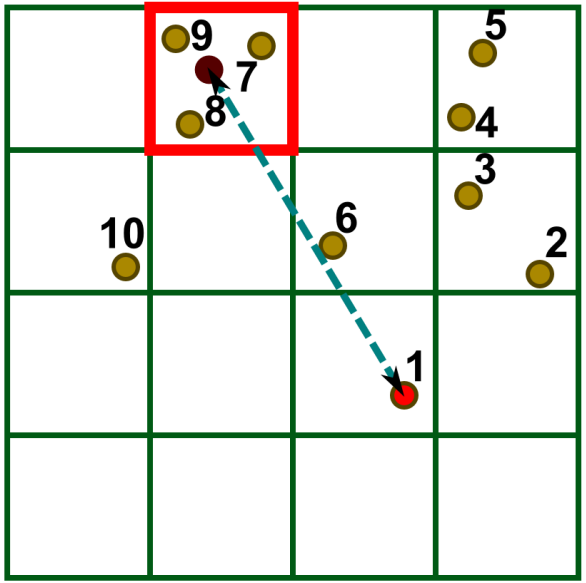
How to define these groups?



First attempt: create a regular grid

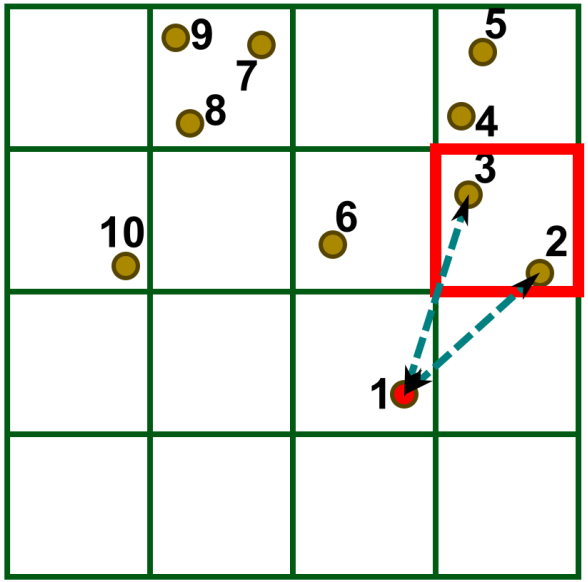
- Subdivide space into equal-sized cells.
- All bodies falling into the same cell contribute to a group.

How to define these groups?



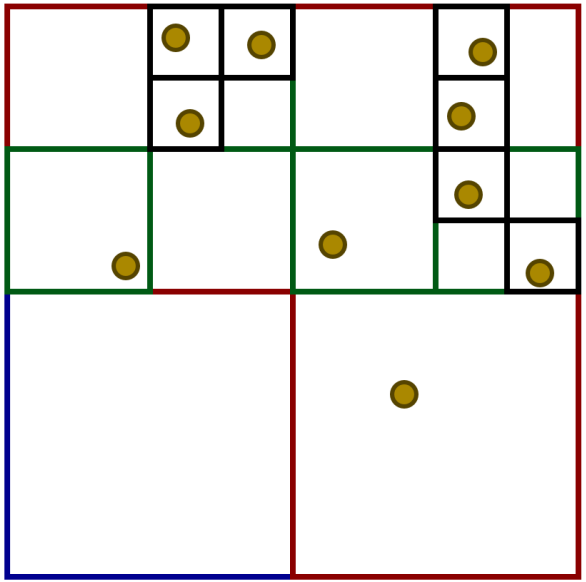
- For some of the bodies, this strategy is perfect.

How to define these groups?



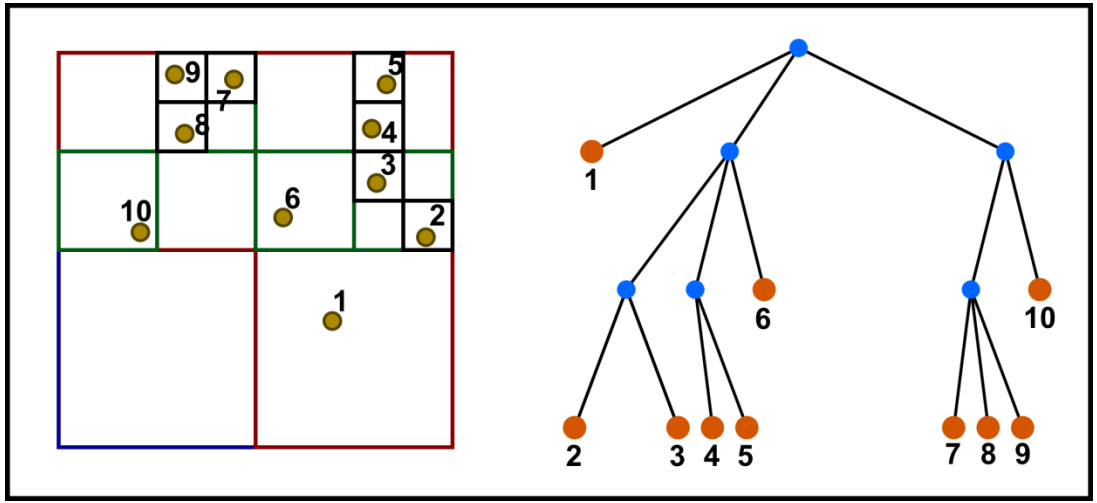
- For some of the bodies, this strategy is perfect.
- But for other bodies, the cell size is wrong.
- Nearby bodies should be grouped into smaller cells.
- Distant bodies can be grouped into larger cells.

Solution: tree algorithms



- Subdivide space recursively, into smaller and smaller cells.
- For close bodies, use the small cells, and for distant bodies the large cells.
- A classic tree algorithm is the **Barnes-Hut** algorithm, which we present here.

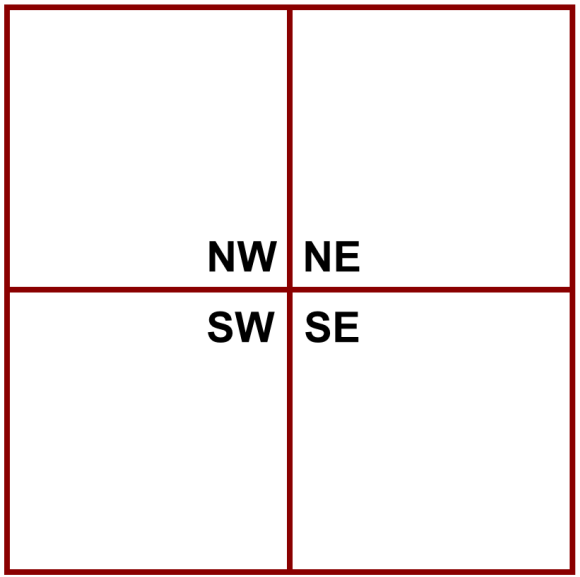
Barnes-Hut algorithm

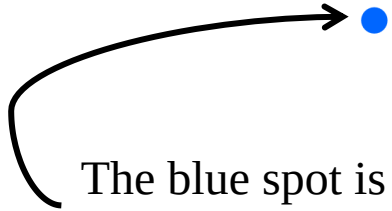


In the Barnes-Hut algorithm, space is recursively subdivided into **quadrants** (in 3D: **octants**).

The relationship between the quadrants is represented by a tree data structure, a **quad-tree** (in 3D: **octree**).

Naming of the quadrants

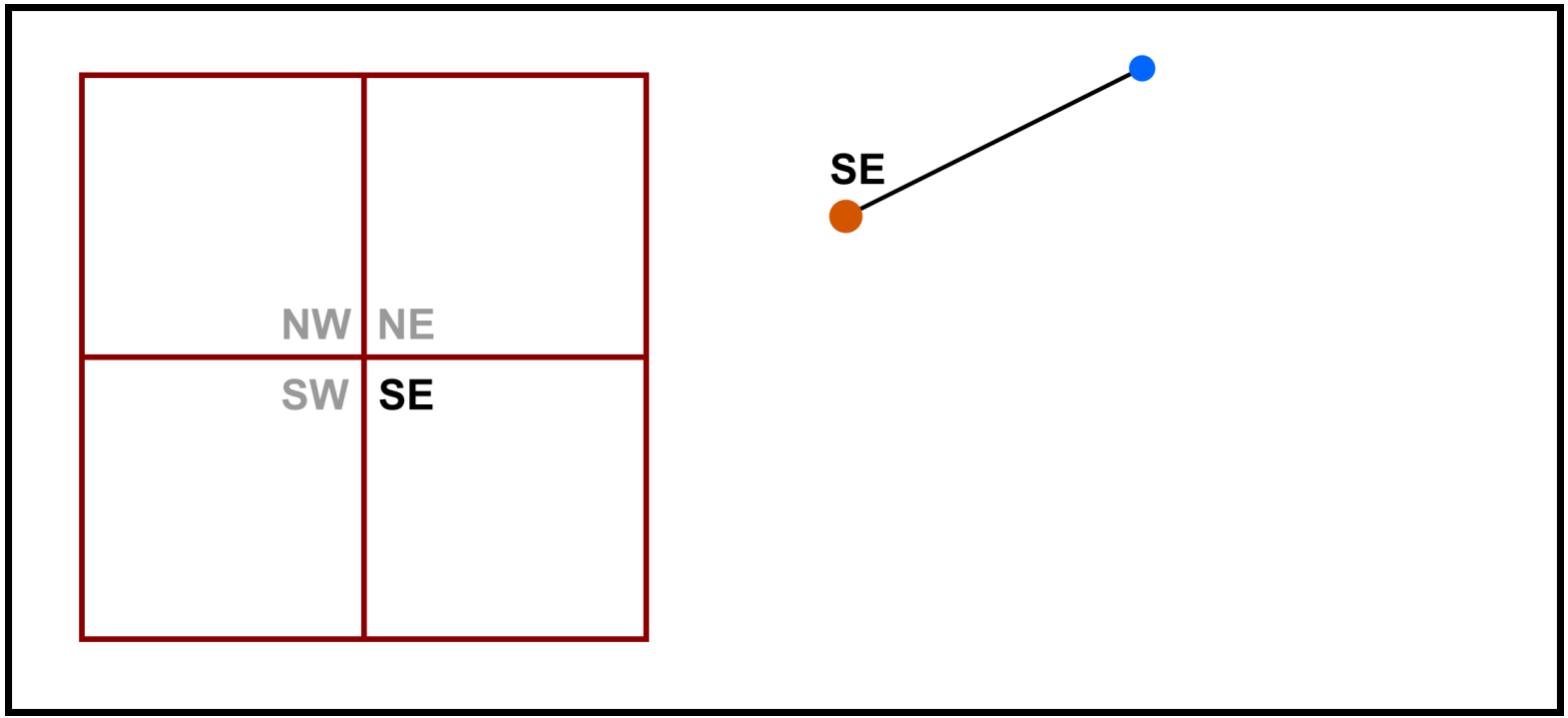




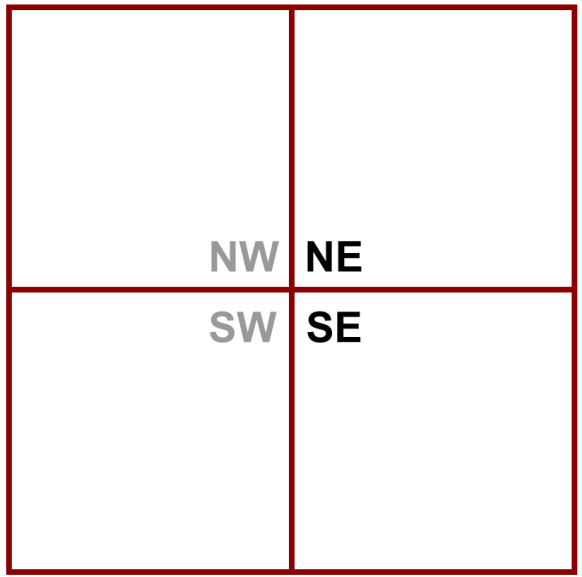
The blue spot is the root of the tree. It represents the full space.

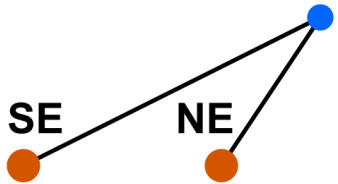
- SE = South-East
- NE = North-East
- SW = South-West
- NW = North-West

South-East quadrant

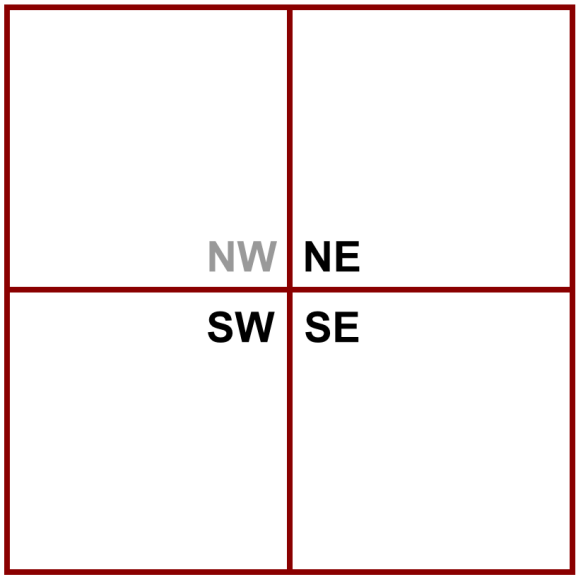


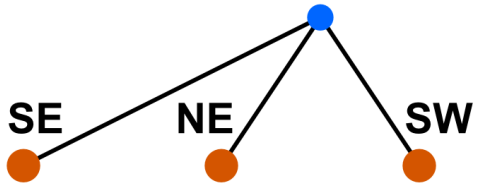
North-East quadrant



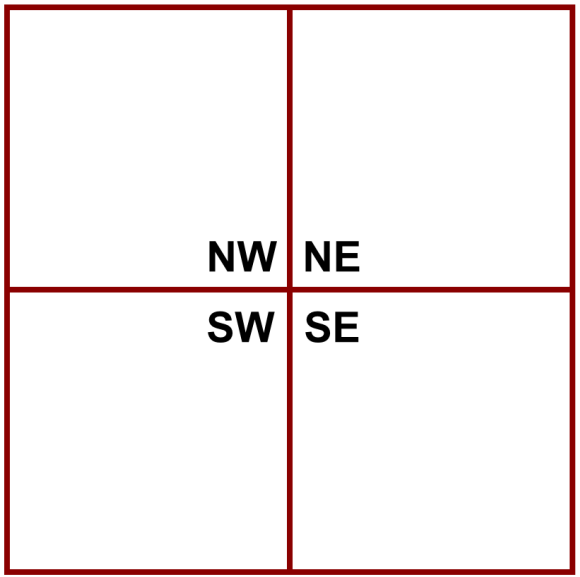


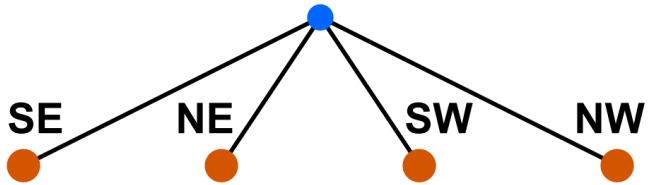
South-West quadrant



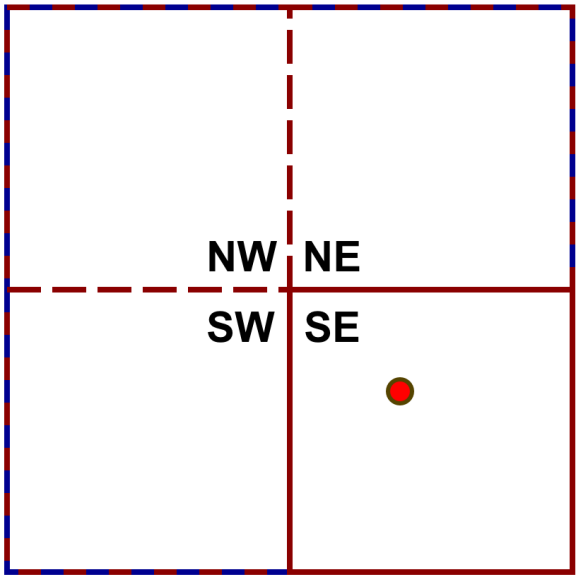


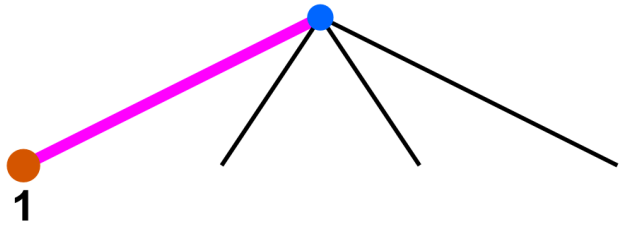
North-West quadrant





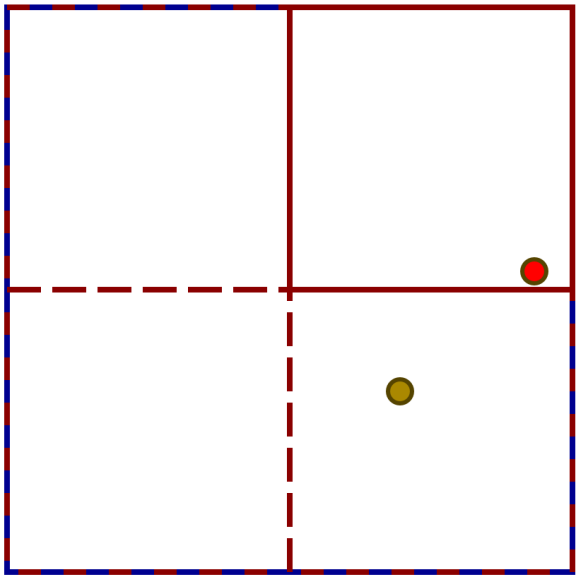
Inserting the first body

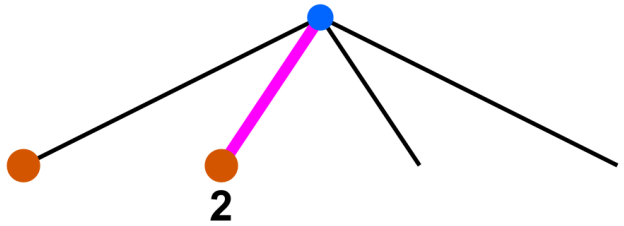




- This is the second node of our tree.
- We placed it on the South-East branch of the root.
- It is an **exterior node**: it has no sub-branches.

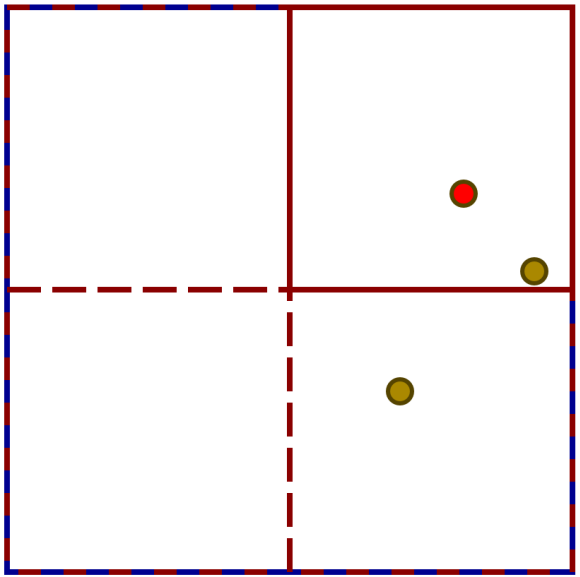
Inserting the second body

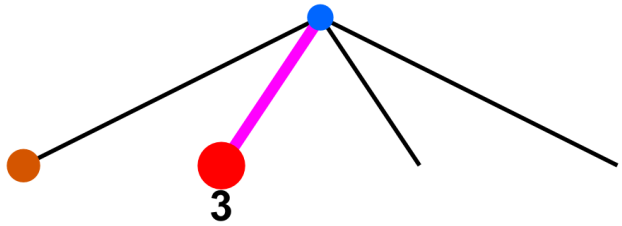




- The second body is located in the North-East quadrant.
- We therefore place it on the North-East branch of the root.

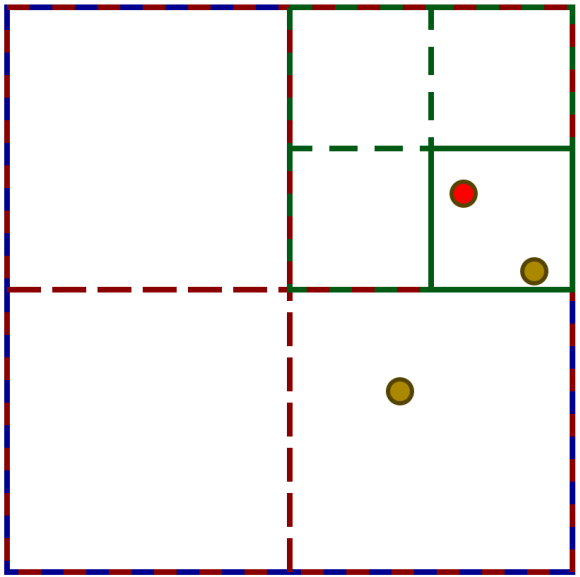
Inserting the third body

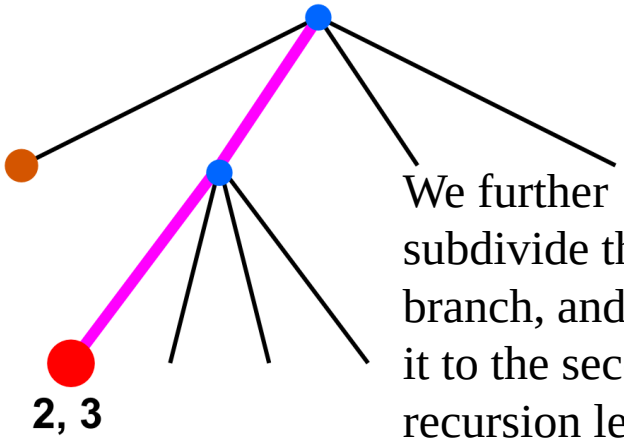




- The third body is also located in the North-East quadrant.
- But body 2 is already located on this node.
- Exterior nodes can have only one body: this is a conflicting situation.

Inserting the third body

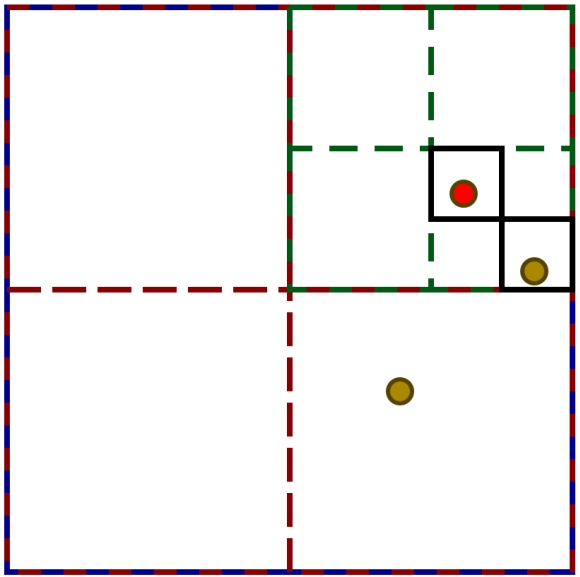


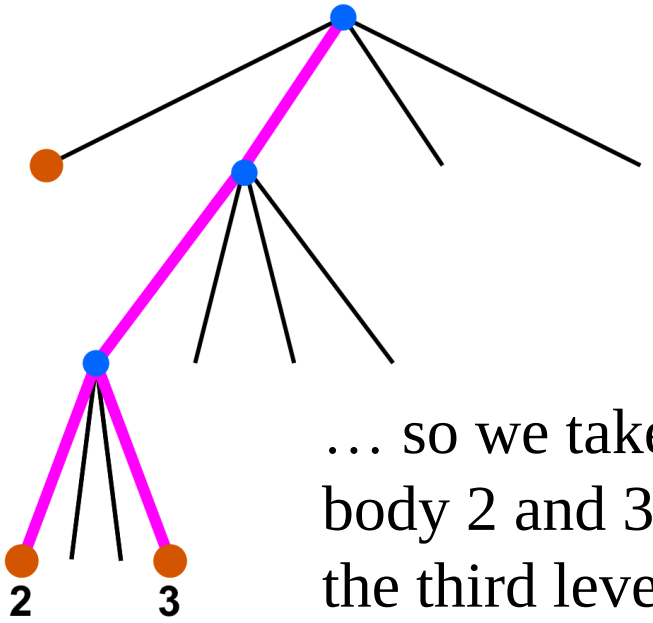


We further subdivide this branch, and take it to the second recursion level.

But at the second level, body 2 and 3 are again in the same quadrant: the conflict is unresolved.

Inserting the third body





… so we take body 2 and 3 to the third level.

Each node has a mass, and center-of-mass

- The internal nodes created during the insertion of the third body represent cells which so far contain two bodies (red cell at first recursion level, green cell at second recursion level).
- They have the same mass and center-of-mass, which we store in the node, for future usage. They are given by:

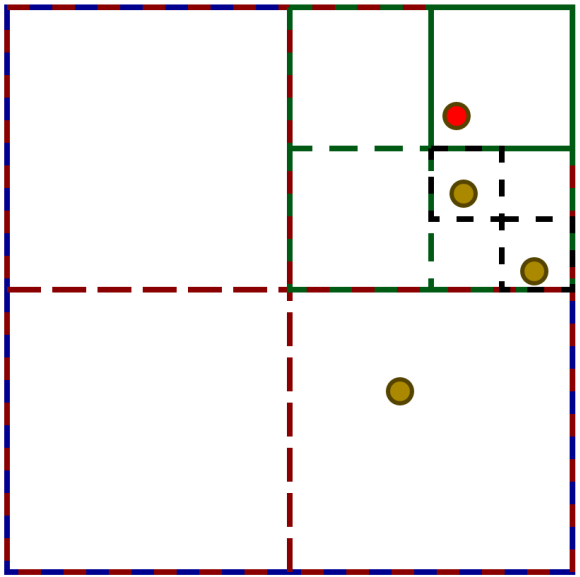
$$m = m_1 + m_2$$
$$\boldsymbol{x} = \frac{(\boldsymbol{x}_1 m_1 + \boldsymbol{x}_2 m_2)}{m}$$

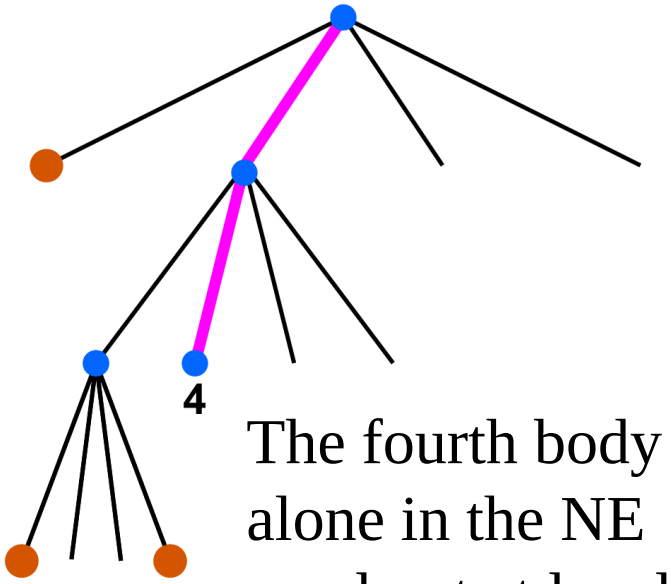
Insertion of a new body: algorithm

Insert a body *b* in the tree recursively, by starting at the root node *n*.

1.	If node *n* does not contain a body, put the new body *b* there.
2.	If node *n* is an internal node (i.e. it has children), update the center-of-mass and total mass of *n*. Recursively insert the body *b* in the appropriate quadrant.
3.	If node *n* is an external node, then the new body *b* is in conflict with a body already present in this region. Subdivide the region further by creating four children. Update the center-of-mass and total mass of *n*. Then, insert both bodies into the appropriate quadrant(s), and proceed recursively as long as there is a conflict.

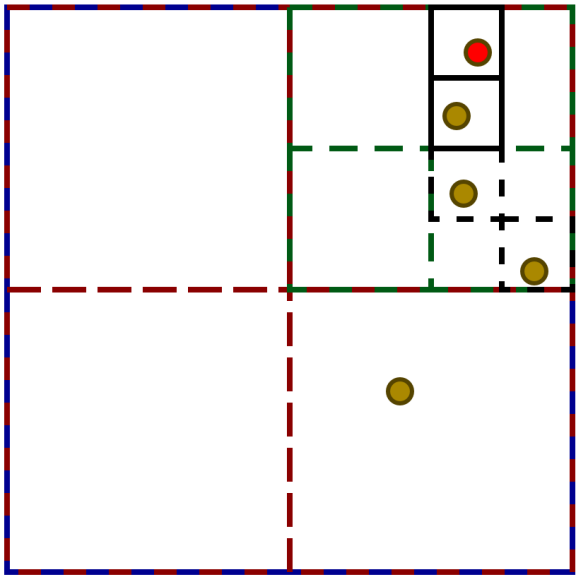
Inserting the fourth body

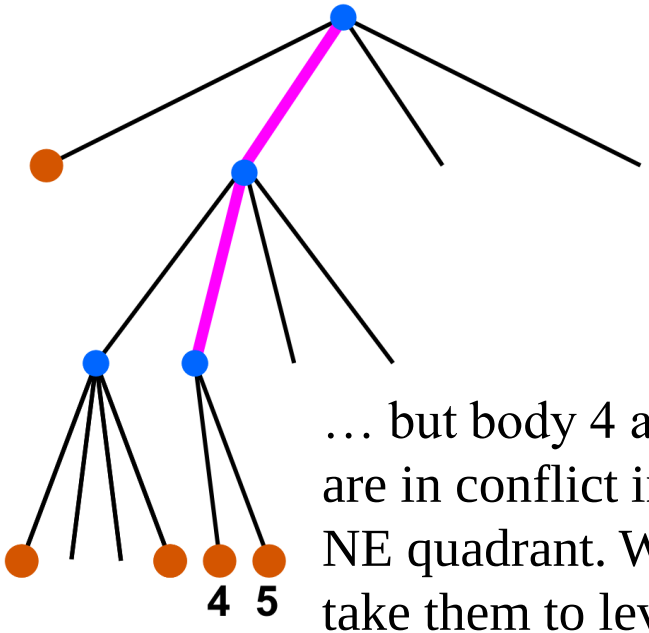




The fourth body is alone in the NE quadrant at level 2.

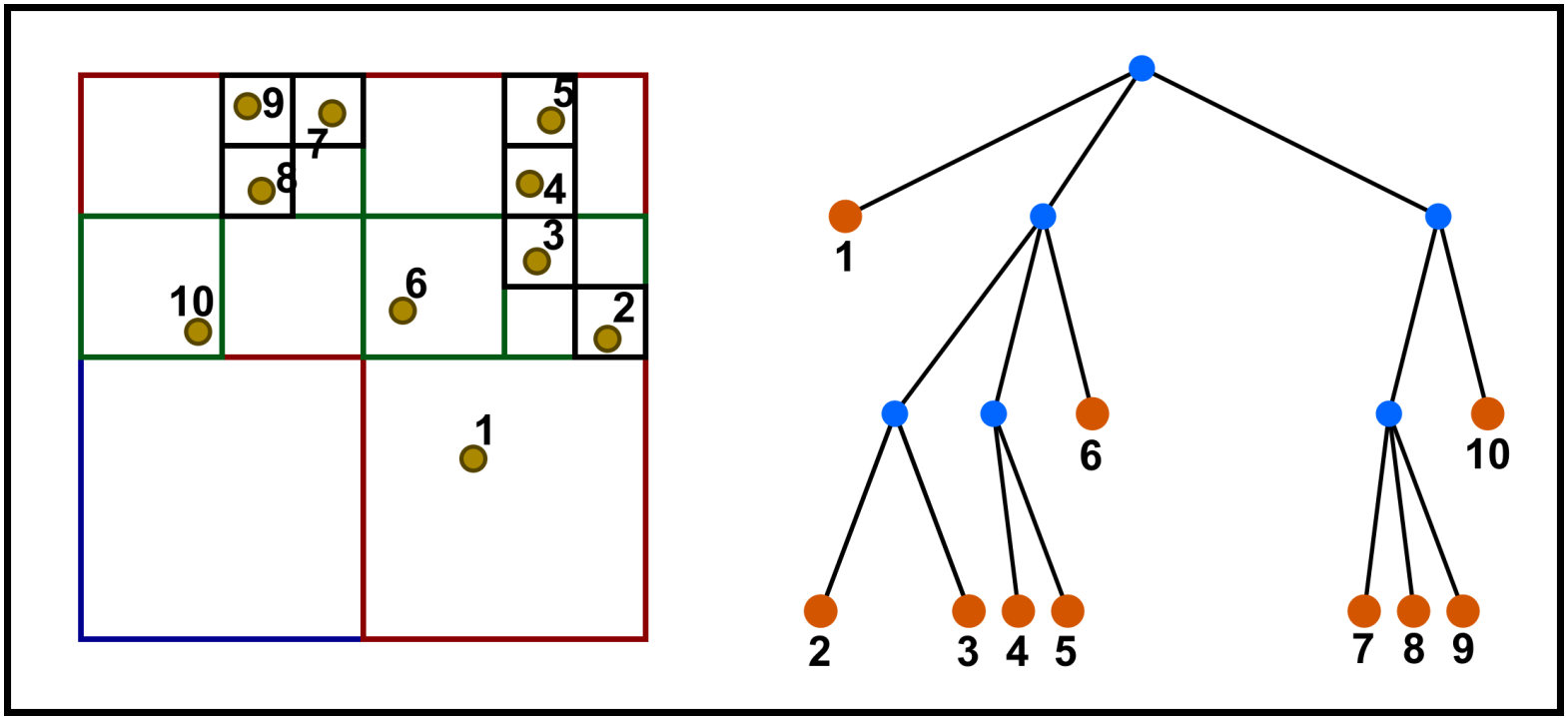
Inserting the fifth body





… but body 4 and 5 are in conflict in the NE quadrant. We take them to level 3.

And so on …



End of module

The n-body problem:
Evaluation of gravitational forces

Coming next

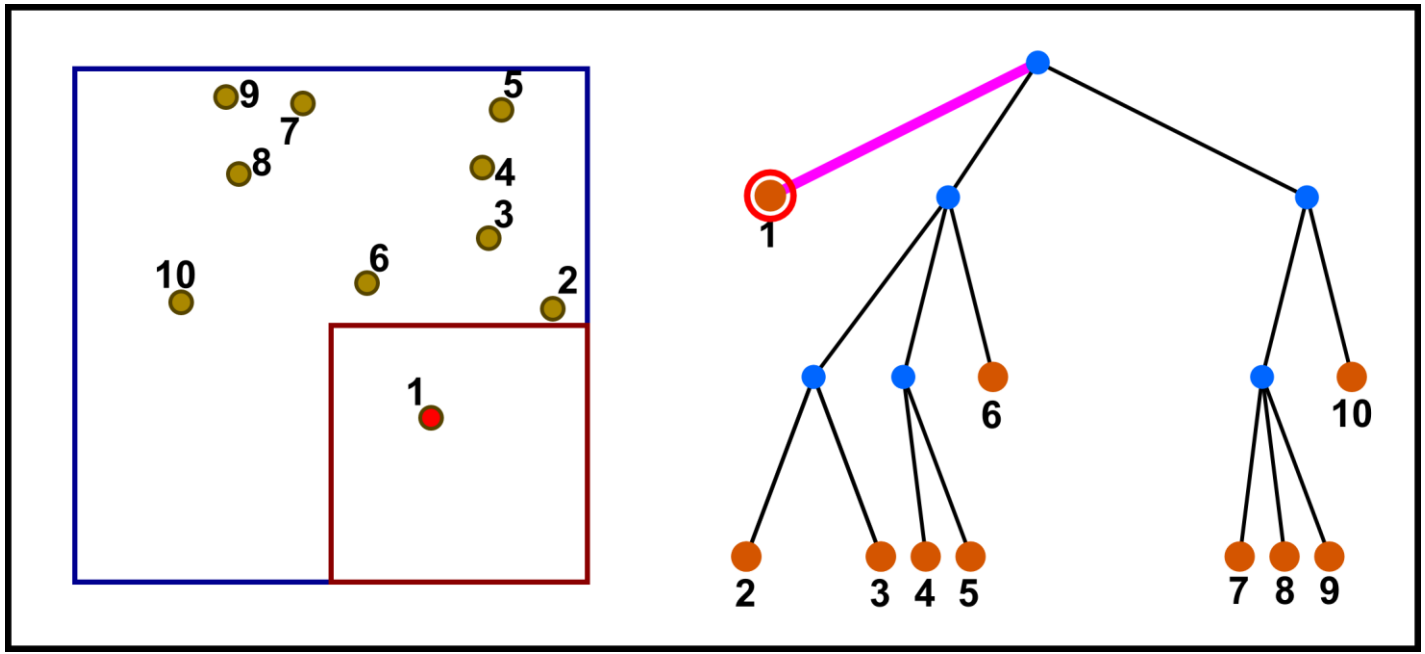
Barnes-Hut algorithm: using the quadtree

Barnes-Hut algorithm: using the quadtree

Computing the force on body 1

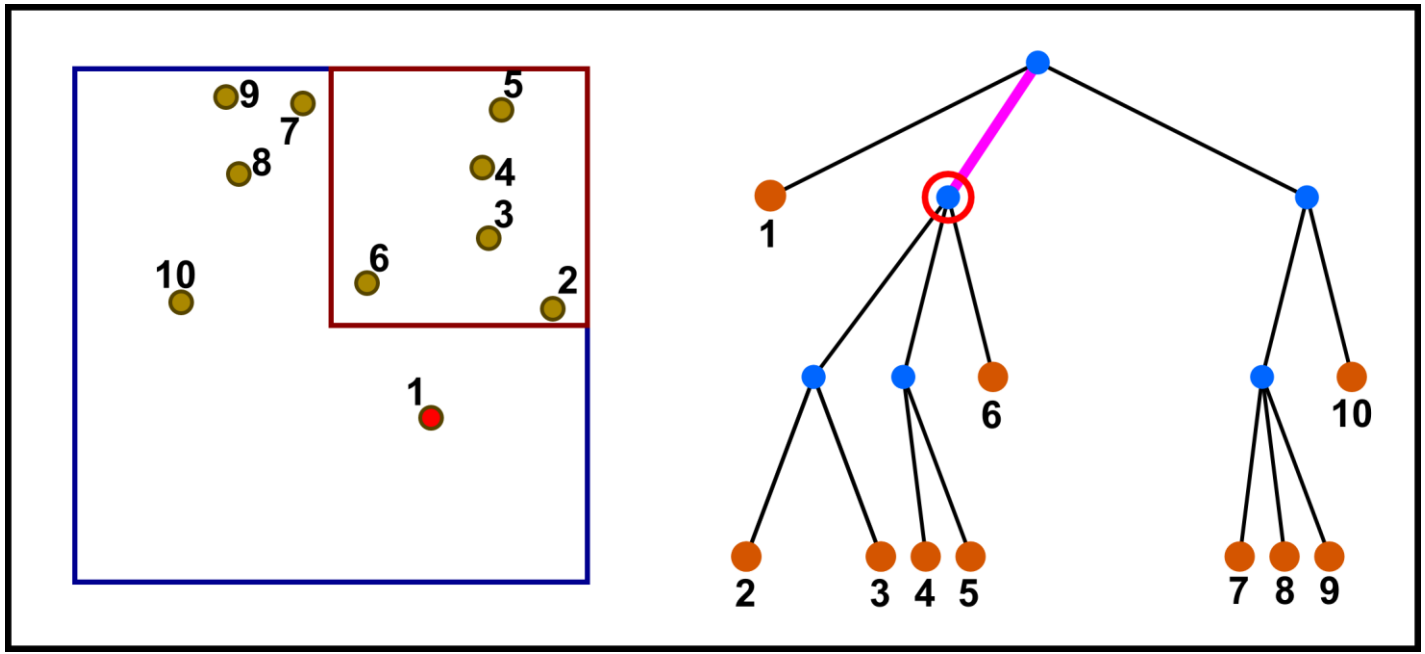
- We will now use our quadtree to compute the forces acting on body 1.
- For this, we traverse the tree recursively. At the first level, we consider the first red quadrants. If they are far enough from body 1, we can use their mass and center of mass.
- If not, we recurse into these quadrants to consider smaller domains.

Tree traversal: SE quadrant



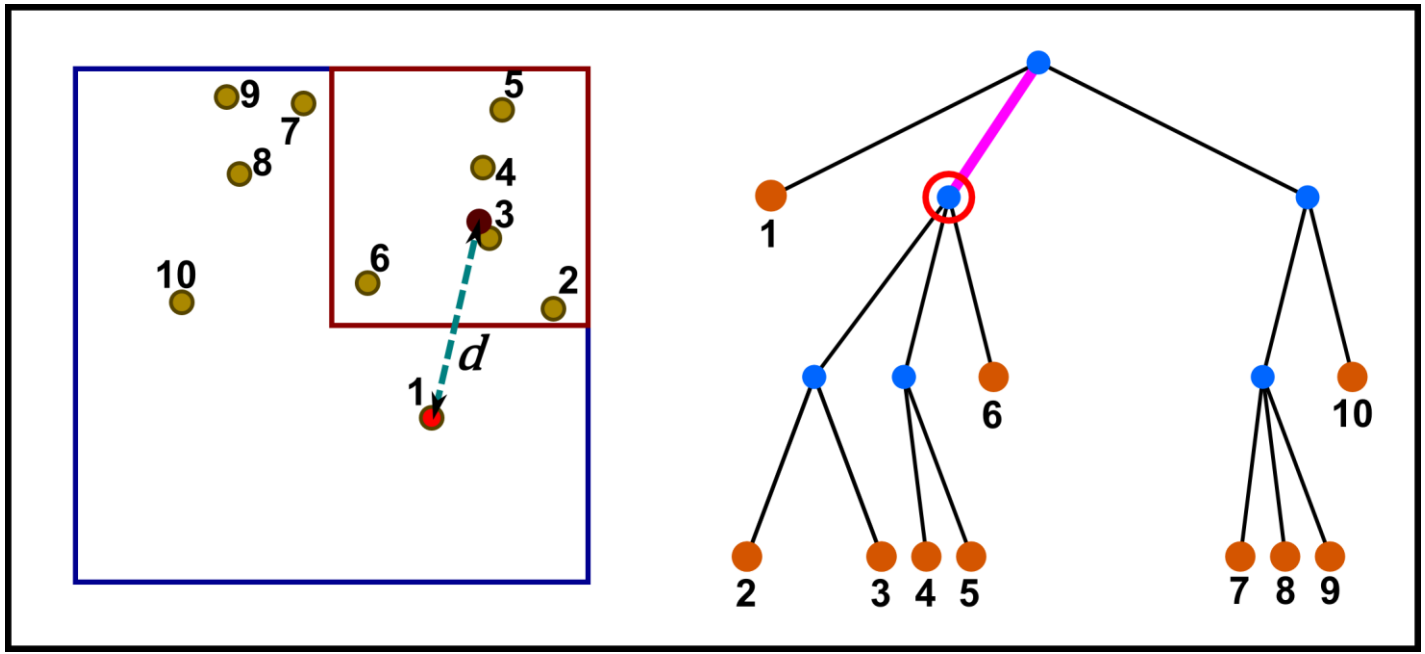
The SE quadrant, at first level, is populated by body 1 itself: No force here.

Tree traversal: NE quadrant



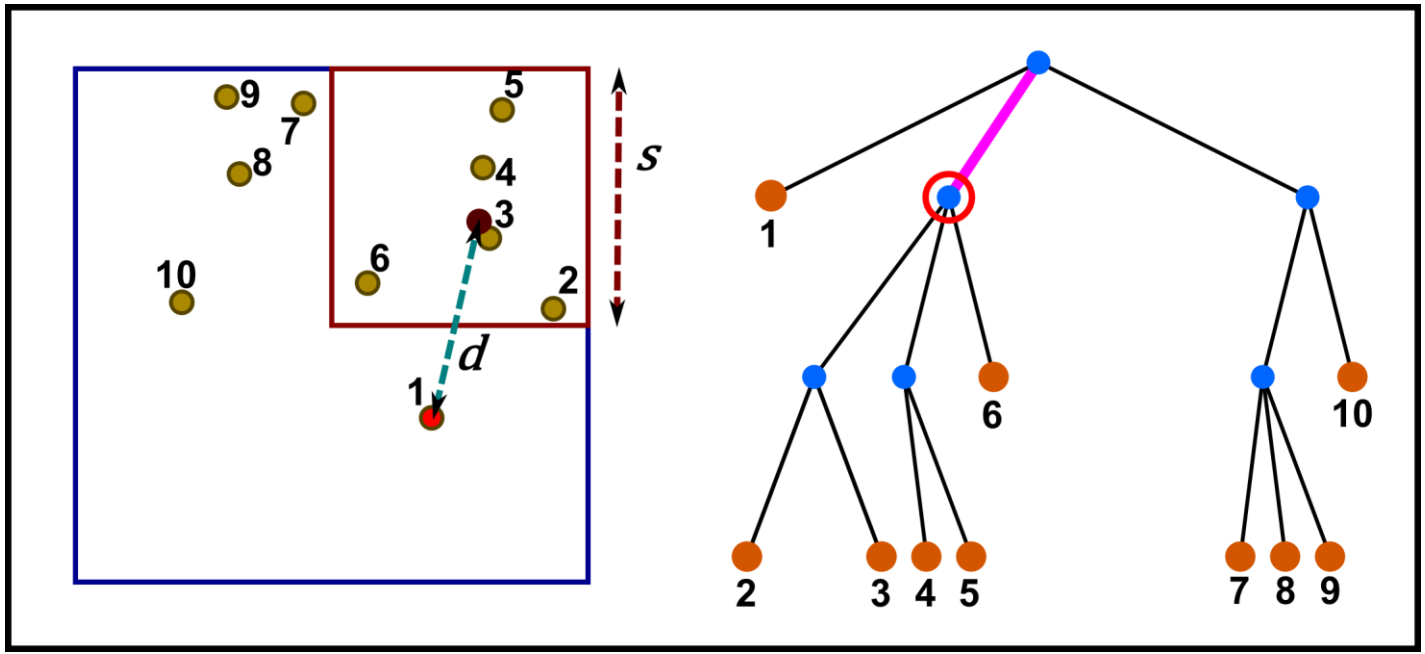
There are 5 bodies in the NE quadrant.

Center of mass of NE quadrant



The center of mass of this quadrant is located at a distance *d* from body 1.

Is this domain far enough?



To decide, we compare the distance *d* with the size *s* of the quadrant.

Theta-criterion

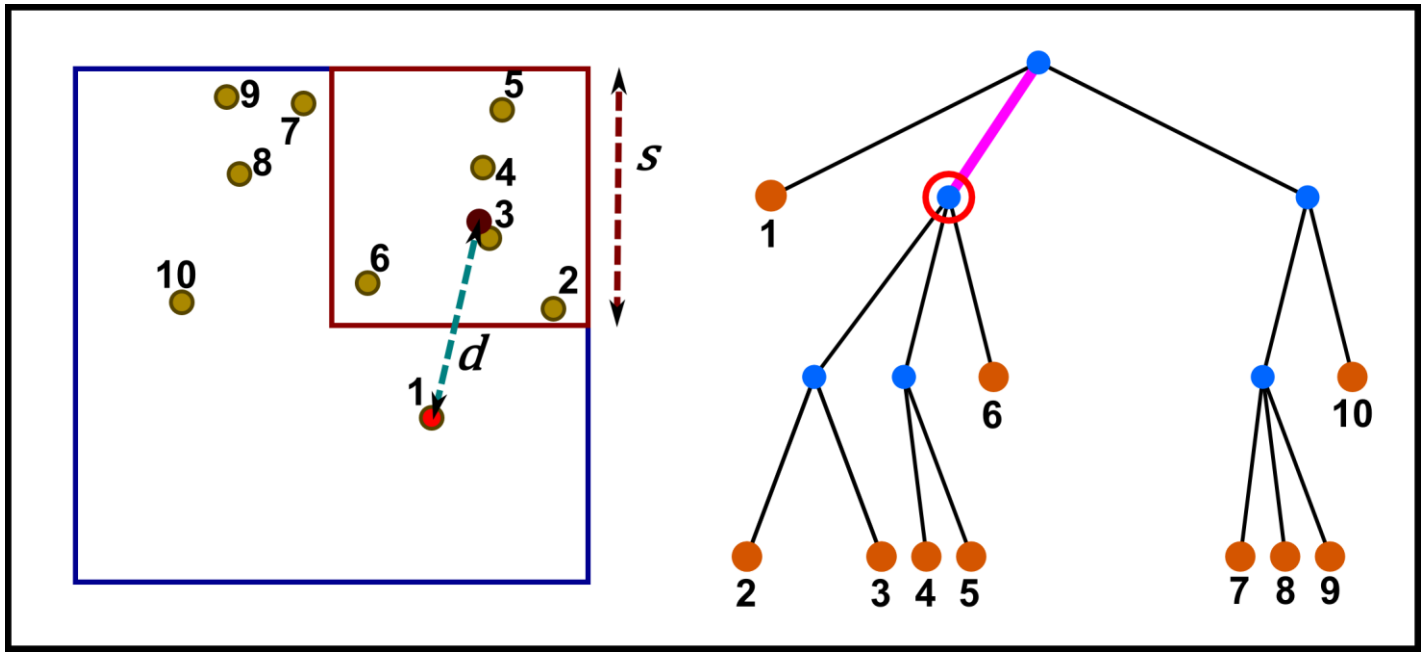
- To decide if a domain is far enough, we must put the domain size *s* in relation with the distance *d* to its center-of-mass.
- We consider that a cell is far enough if

$$\frac{s}{d} < \theta$$

  Where θ is a given threshold value. In the present example, we will work with $\theta = 0.5$, a common value for the Barnes-Hut algorithm.
- In this case, the criterion amounts to

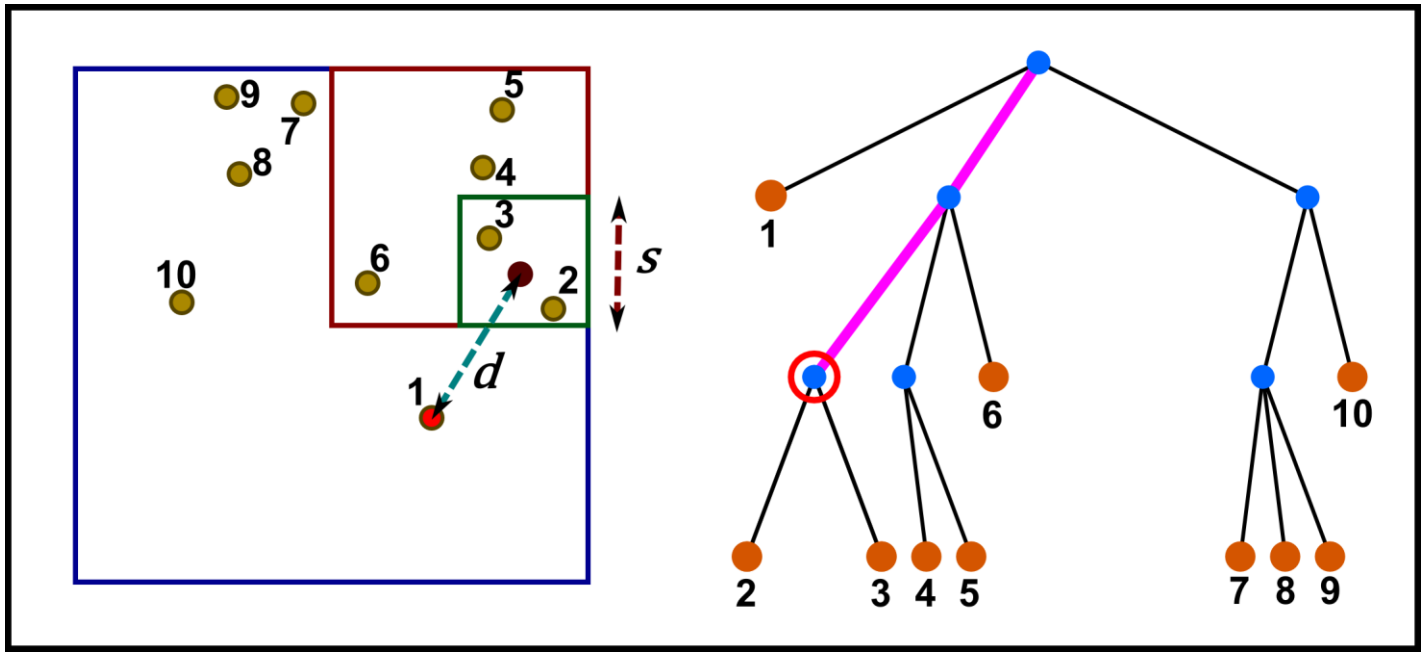
$$d > 2s$$

Applying the theta-criterion



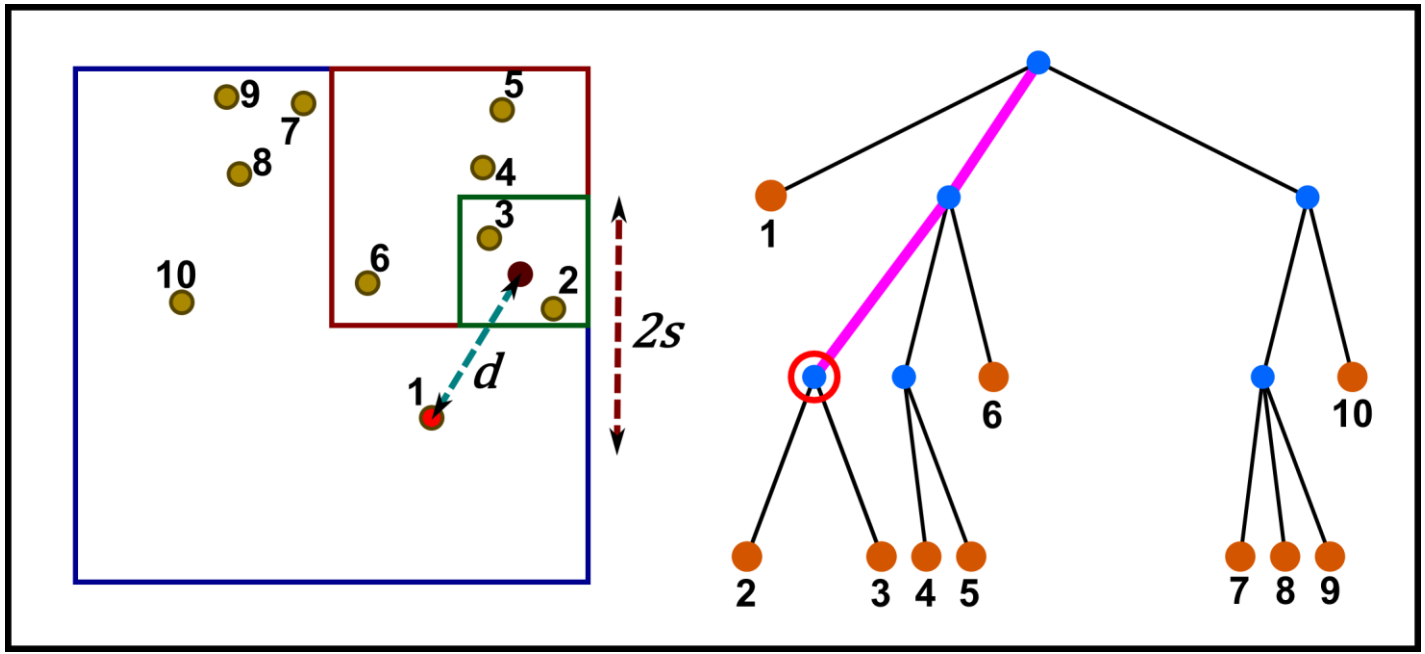
Here, $d < 2s$. Domain rejected!

Trying SE domain at second level



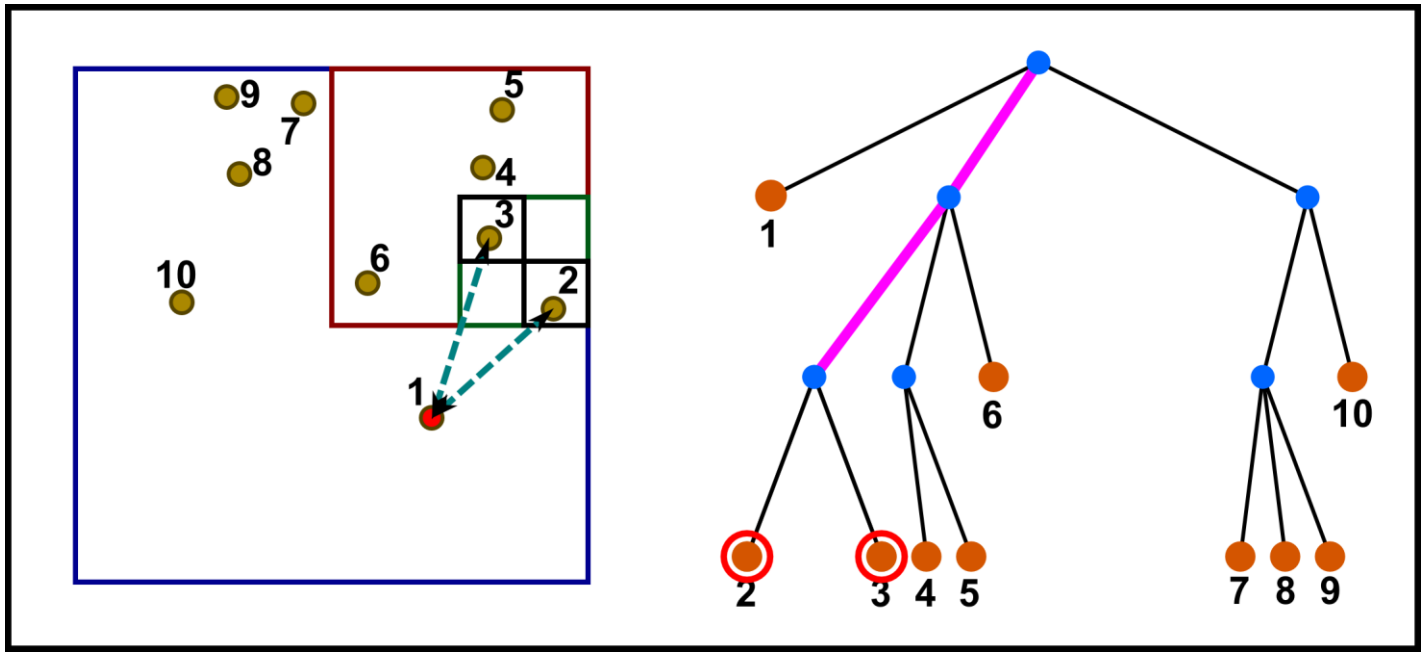
This time, the domain is smaller…

Trying SE domain at second level



… but still, $d < 2s$. Domain rejected again!

We need to go to third level



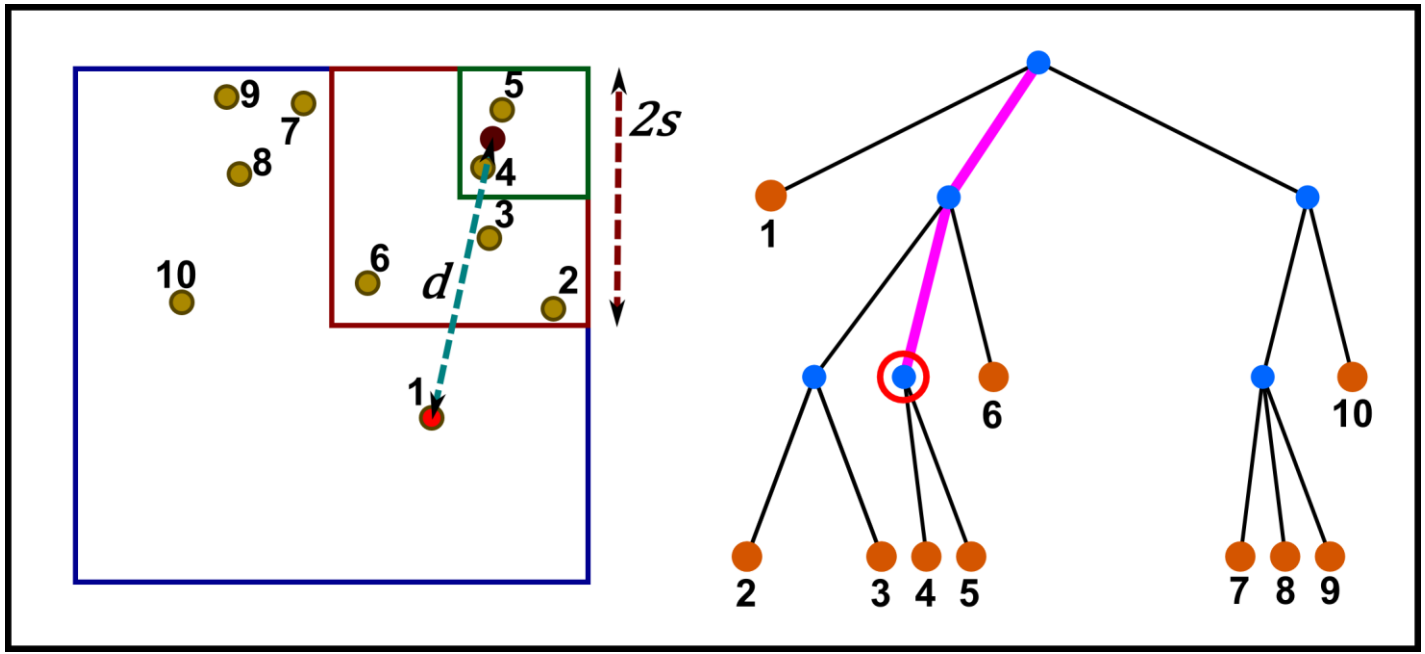
These are exterior nodes, representing single bodies. We accept them by default.

Force on a body: algorithm

To calculate the force on *b*, use the a recursive procedure, starting at the root node.

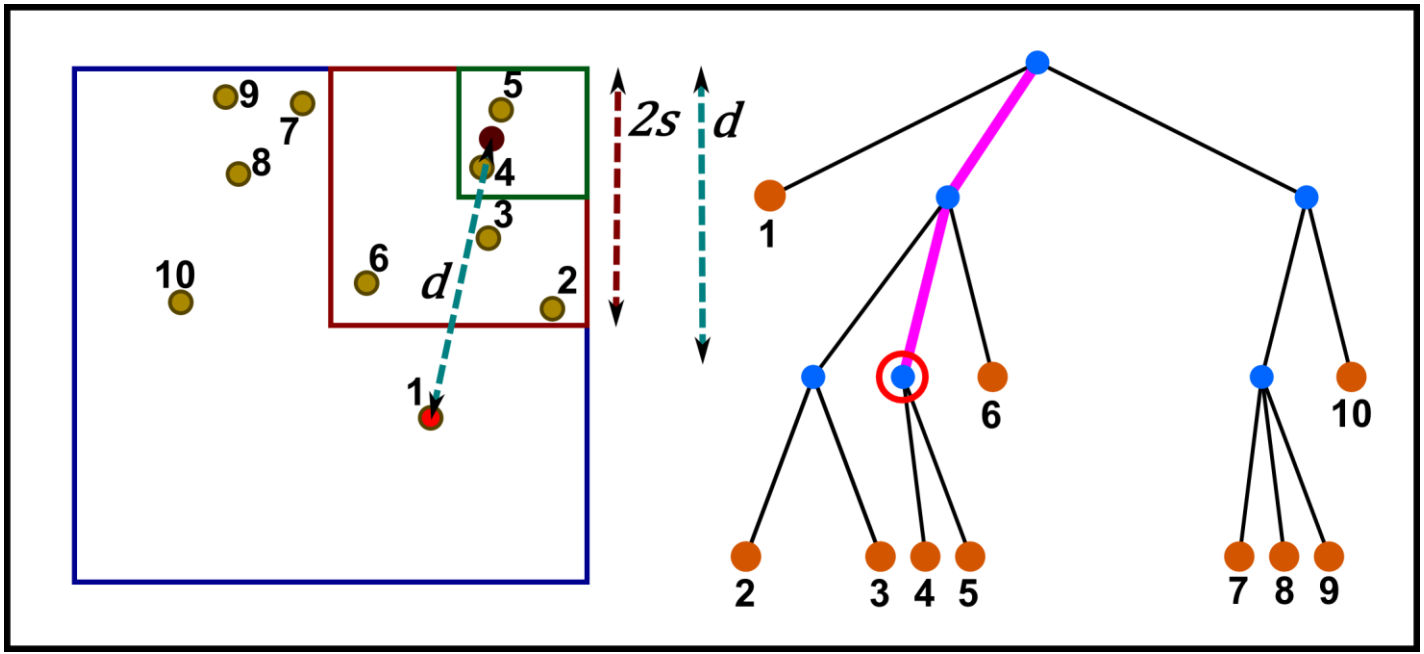
1.	If the current node is an external node (and it is not body *b*), calculate the force exerted by the current node on *b*, and add this amount to *b*'s net force.
2.	Otherwise, calculate the ratio s/d. If $s/d < \theta$, treat this internal node as a single body, and calculate the force it exerts on body *b*, and add this amount to *b*'s net force.
3.	Otherwise, run the procedure recursively on each of the current node's children.

Continue with NE quadrant, level 2



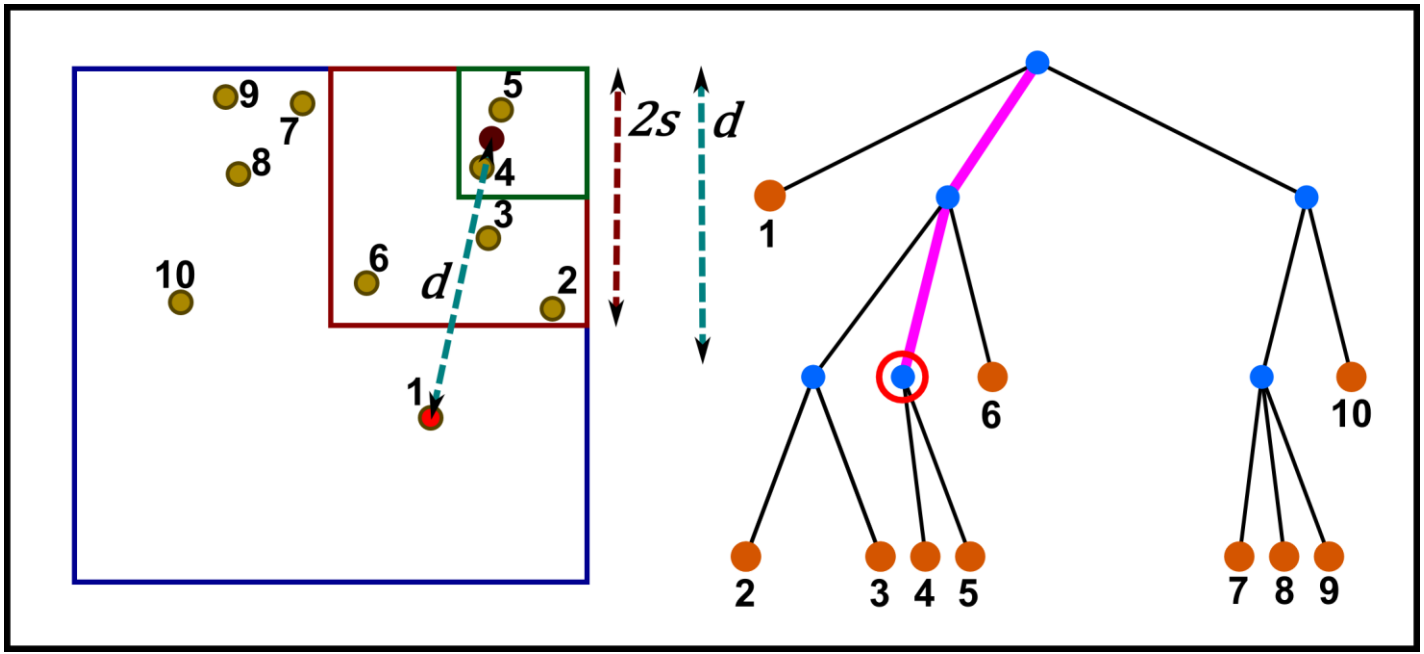
Two bodies in this quadrant. Can we replace them by their center of mass?

Continue with NE quadrant, level 2



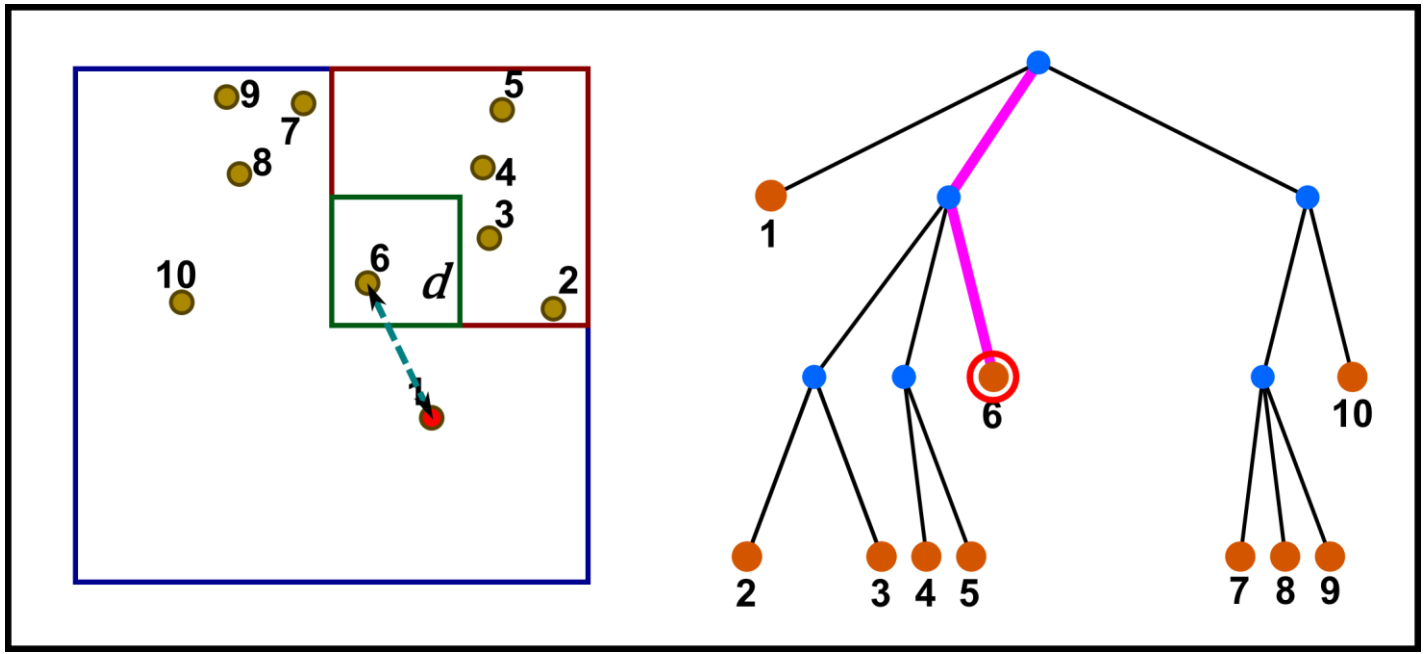
Yes, we can! $d > 2s$.

Continue with NE quadrant, level 2



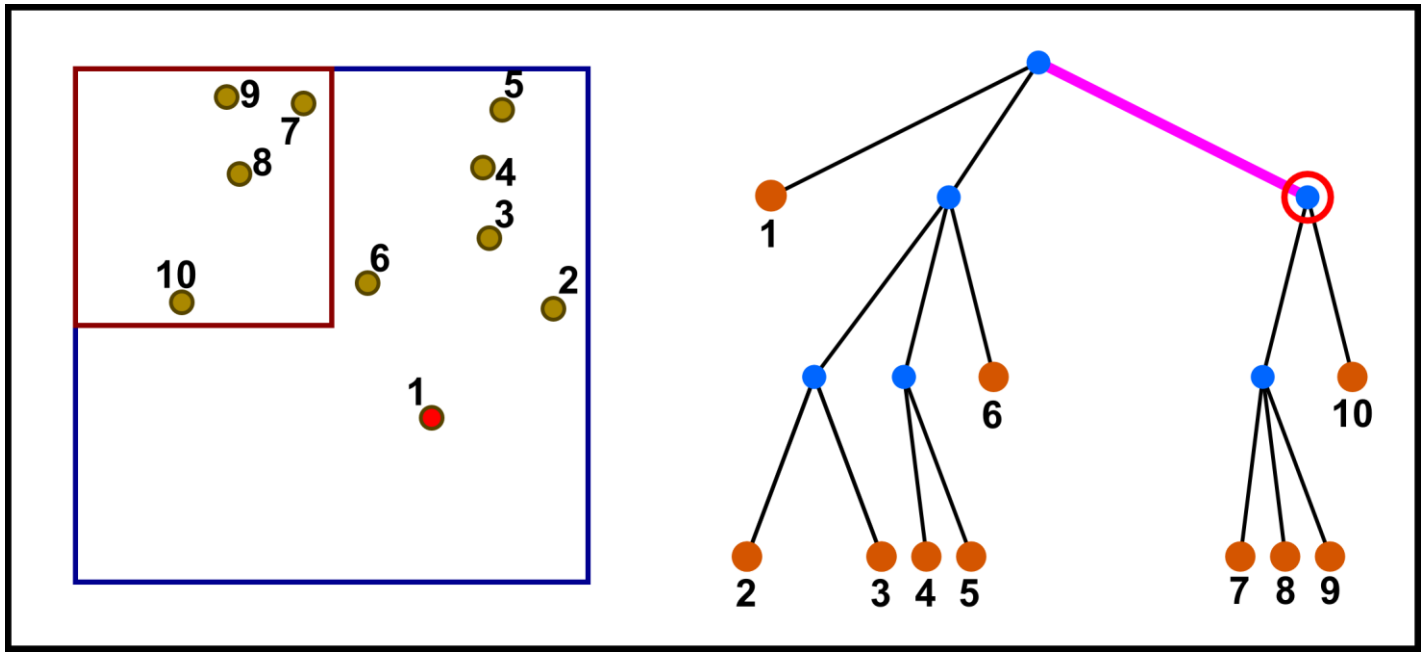
Yes, we can! $d > 2s$.

And SE quadrant, level 2



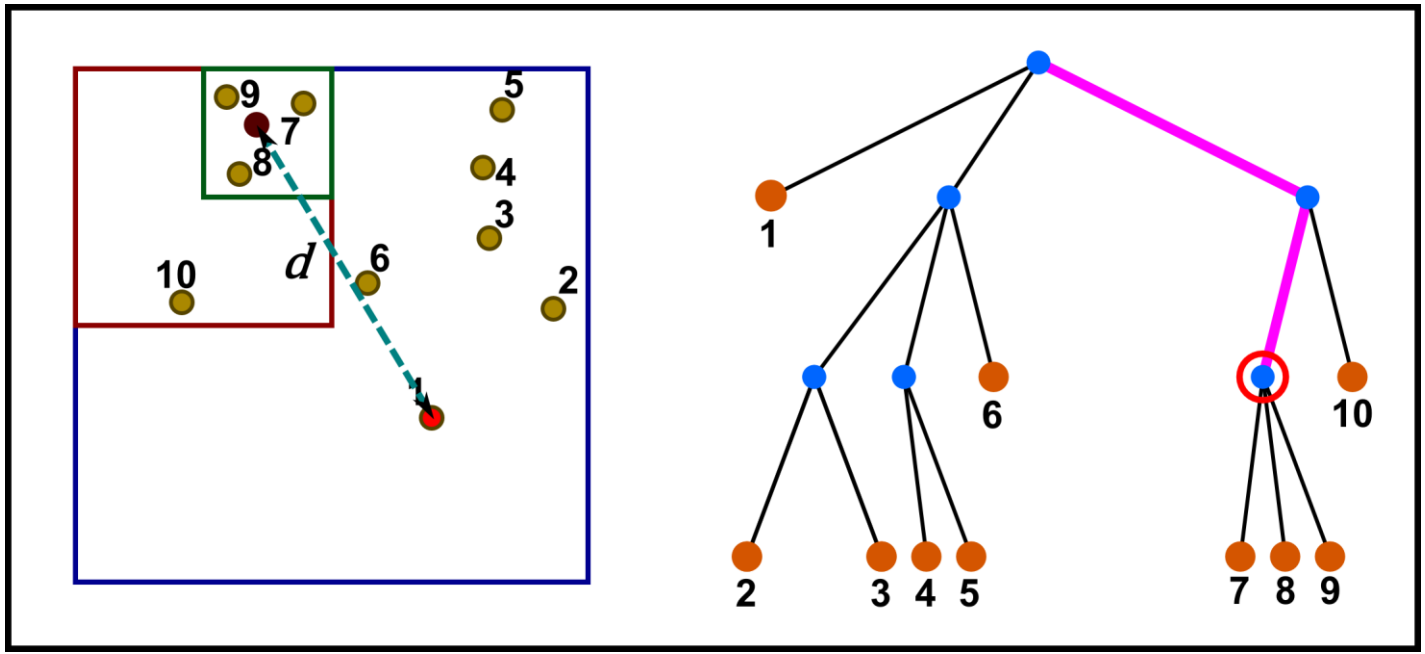
This is an exterior node, which is accepted by default.

Back to level 1: NW quadrant



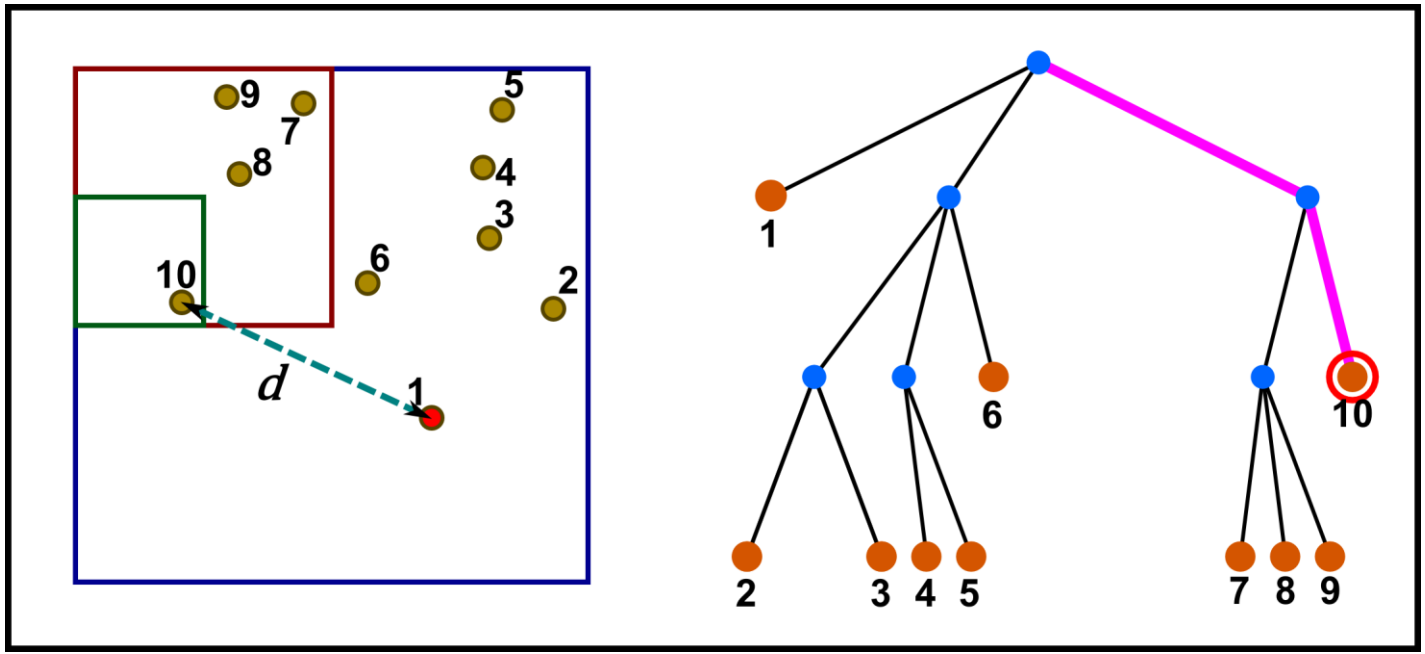
This quadrant doesn’t verify the theta-criterion, obviously …

Level 3, NE quadrant



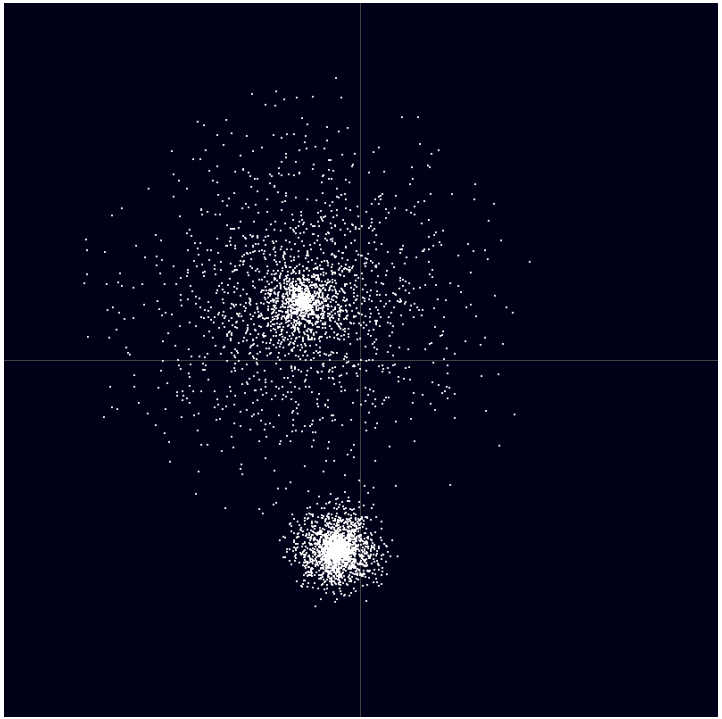
…but the sub-quadrant, at NE position, is fine (believe me).

Final quadrant…



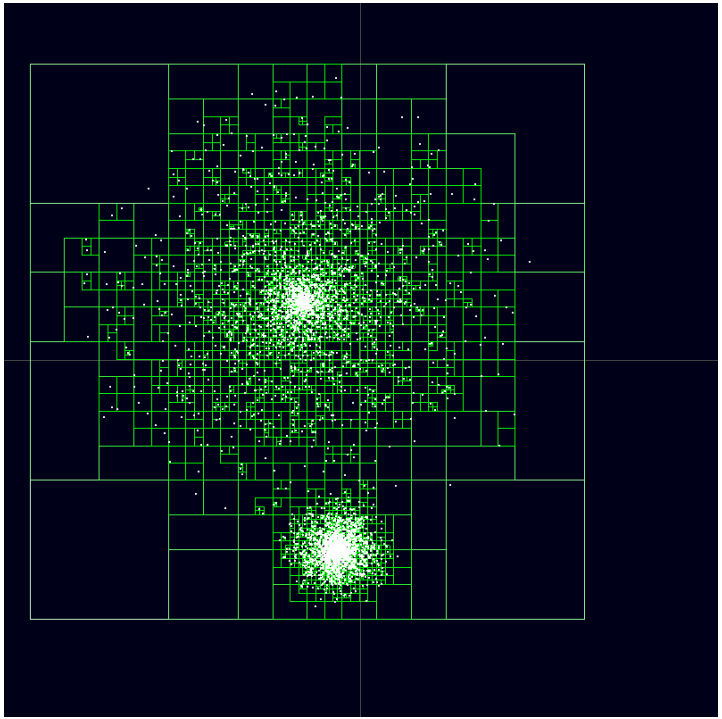
And we end the algorithm with an exterior node, which is default-accepted.

With 5000 bodies: same procedure



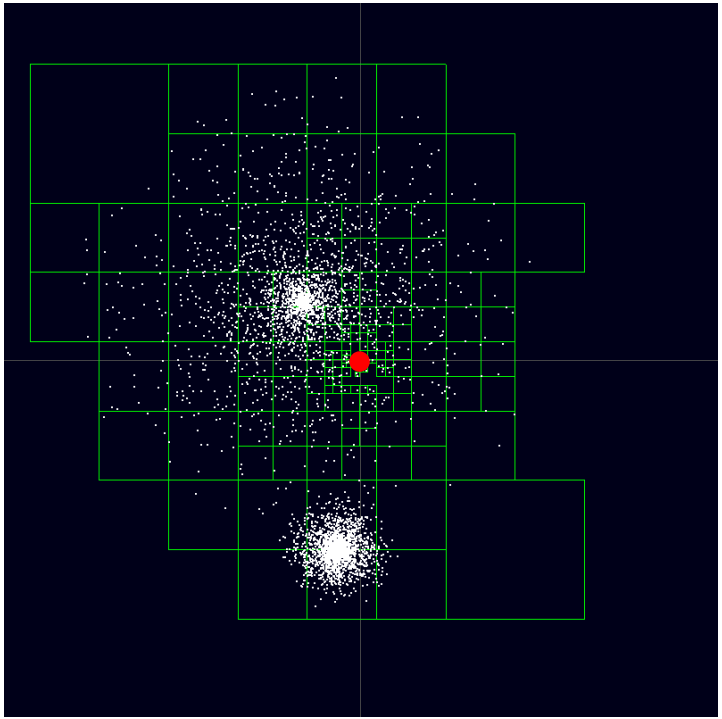
- With 10 bodies, the Barnes-Hut algorithm is an overkill.
- But with more bodies, like the case of 5000 bodies on the left, the algorithm substantially improves calculation times.

With 5000 bodies: same procedure



- Structure of the octree, with 5000 bodies.

With 5000 bodies: same procedure



- Cells used to compute the force on a body in the center of the domain.

End of module

Barnes-Hut algorithm:

Using the quadtree

End of week 6