Lecture 26: N-Body Problem

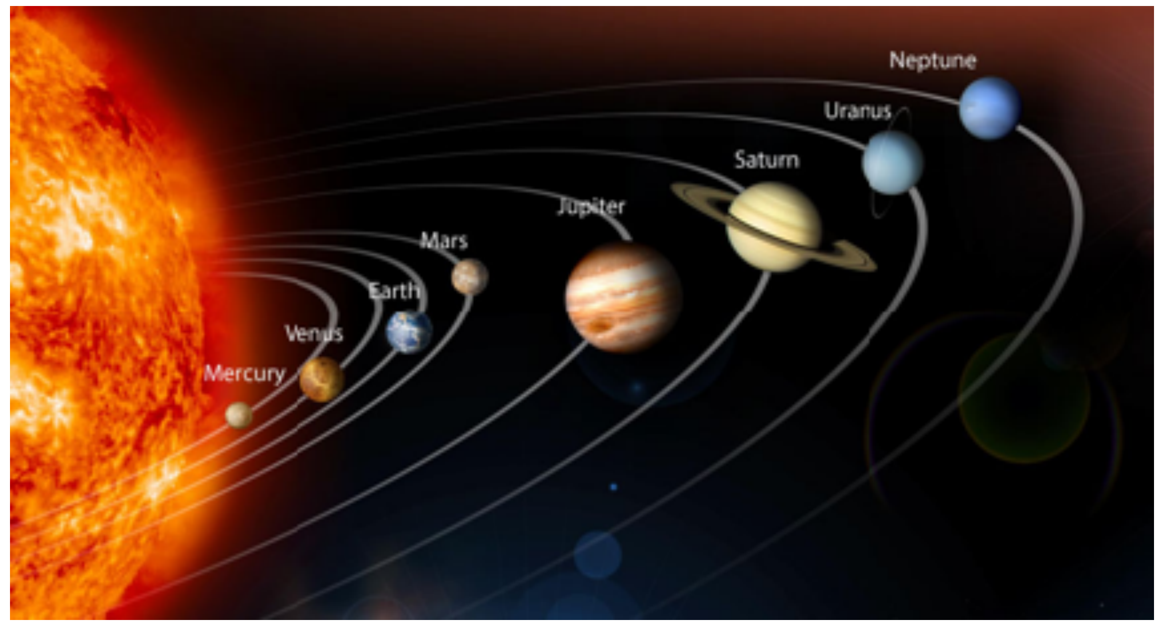
Zoran Budimlić and Mack Joyner
{zoran, mjoyner}@rice.edu

http://comp322.rice.edu

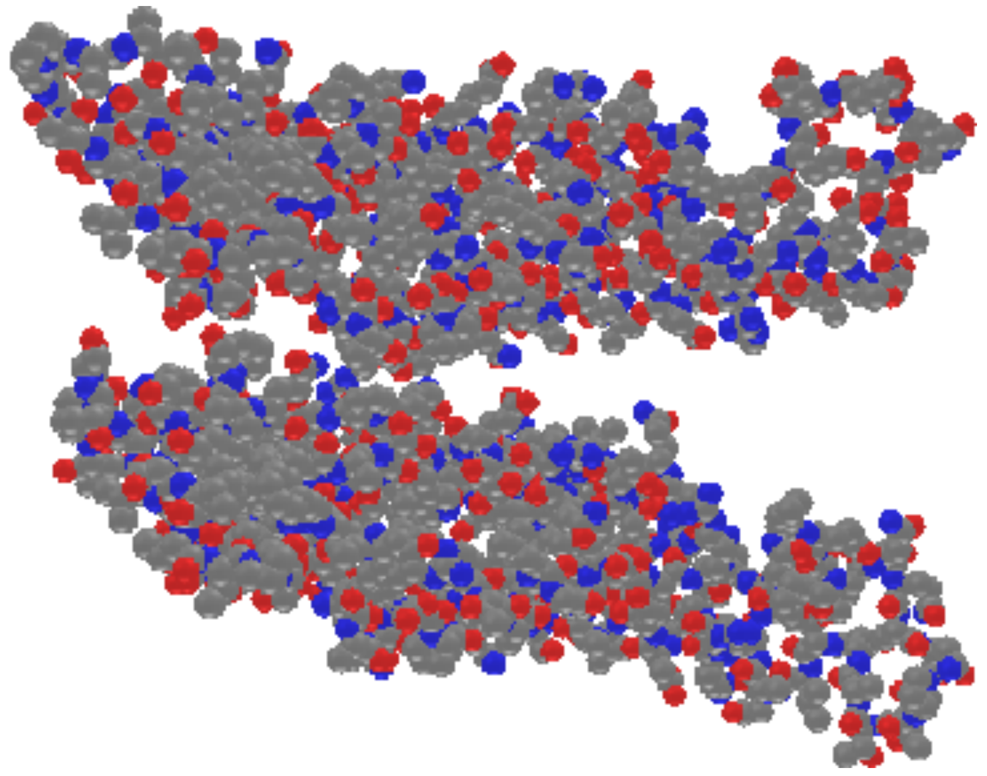
Acknowledgments: James Demmel (UC Berkeley)

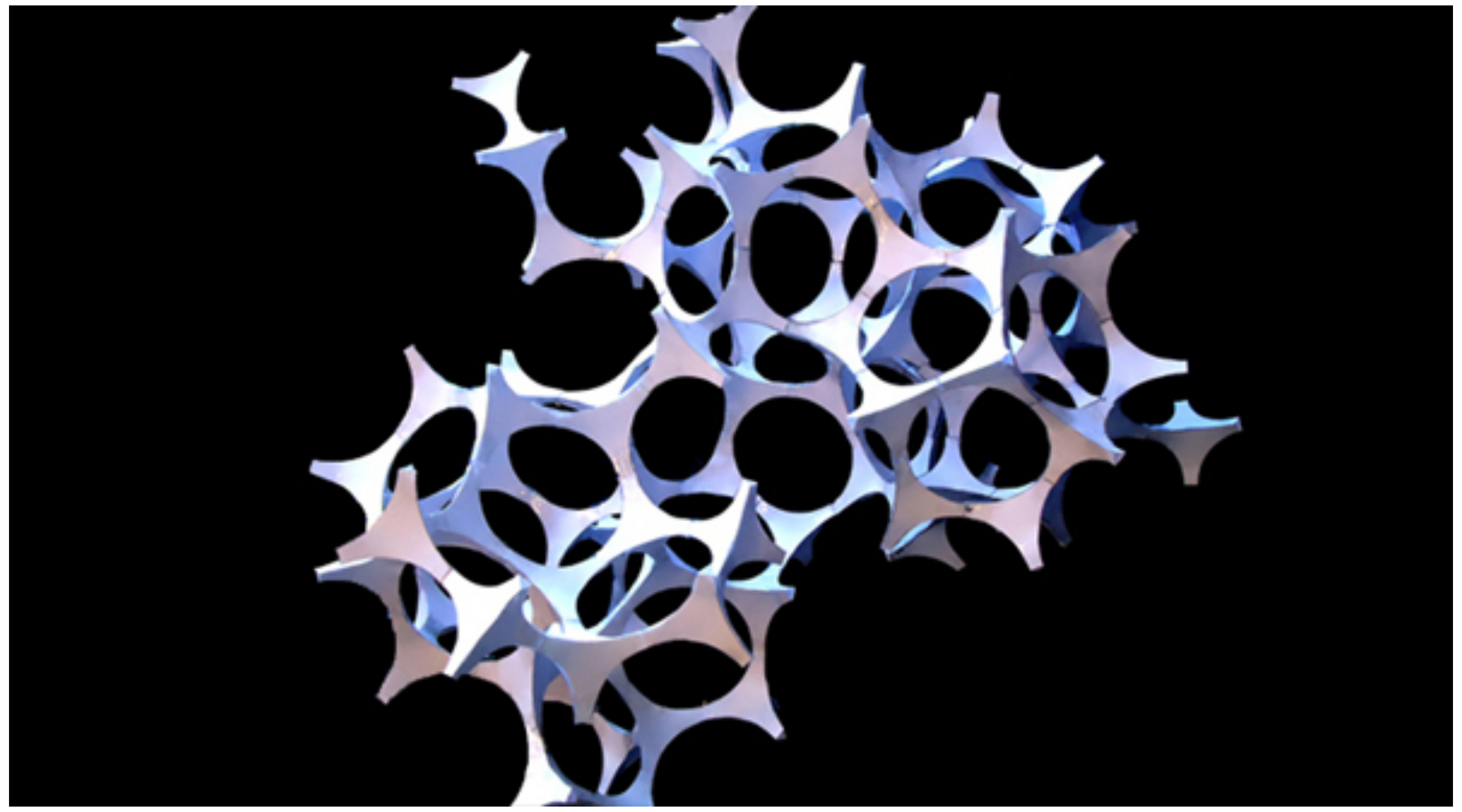
N-Body problems

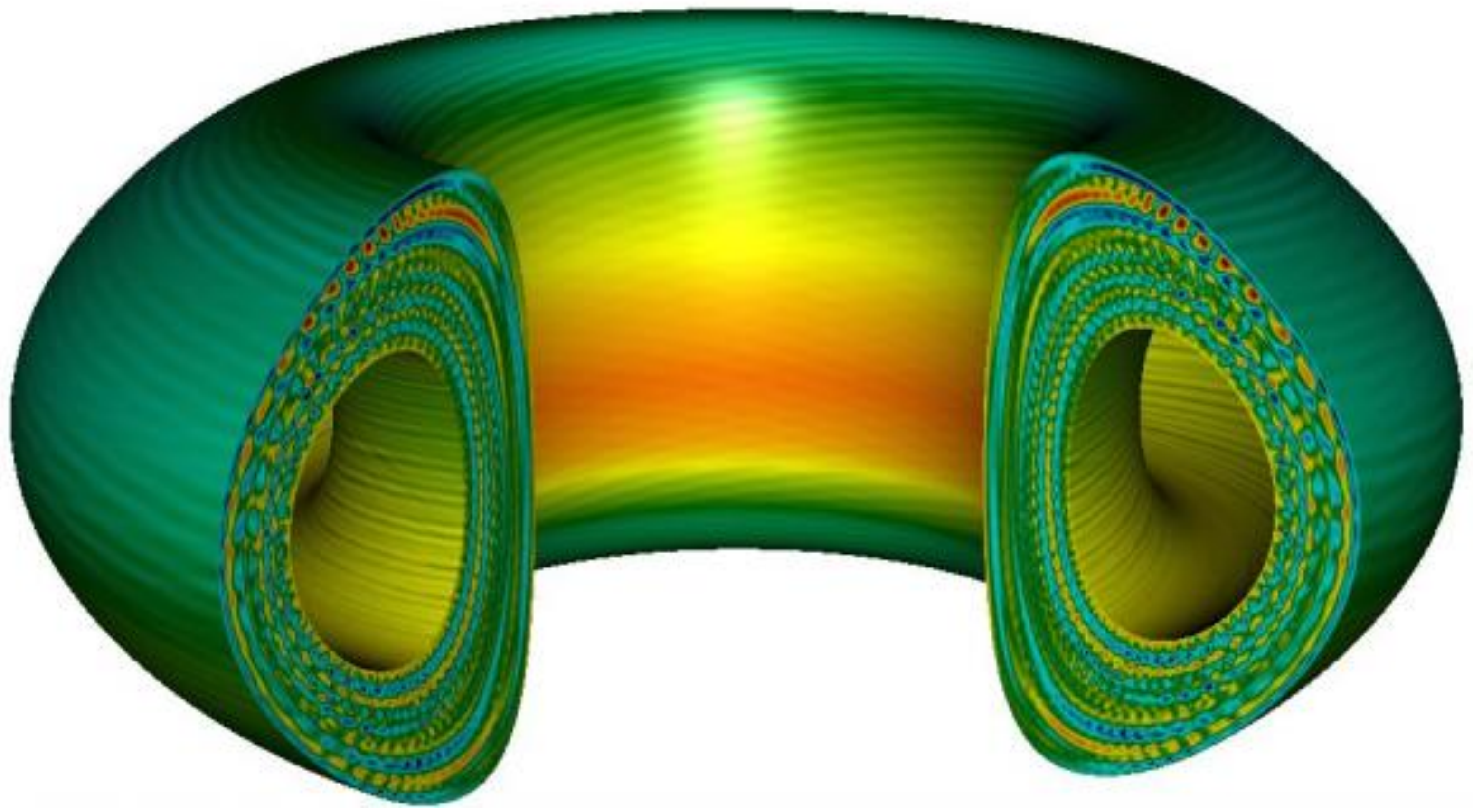
- Gravitational



- Electromagnetic







Big Idea

- Suppose the answer at each point depends on data at all the other points
  - Electrostatic, gravitational force
  - Solution of elliptic PDEs
  - Graph partitioning
- Seems to require at least $O(n^2)$ work, communication
- If the dependence on "distant" data can be compressed
  - Because it gets smaller, smoother, simpler…
- Then by compressing data of groups of nearby points, can cut cost (work, communication) at distant points
  - Apply idea recursively: cost drops to $O(n \log n)$ or even $O(n)$
- Examples:
  - Barnes-Hut or Fast Multipole Method (FMM) for electrostatics/gravity/…
  - Multigrid for elliptic PDE
  - Multilevel graph partitioning (METIS, Chaco,…)

Particle Simulation

```
t = 0
while t < t_final
    for i = 1 to n                ... n = number of particles
        compute f(i) = force on particle i
    for i = 1 to n
        move particle i under force f(i) for time dt    ... using F=ma
    compute interesting properties of particles (energy, etc.)
    t = t + dt
end while
```

- f(i) = external_force + nearest_neighbors_force + N-Body_force
  - External_force is usually embarrassingly parallel and costs O(N) for all particles
  - Nearest_neighbors_force requires interacting with a few (constant # of) neighbors, so still O(N)
  - N-Body_force (gravity or electrostatics) requires all-to-all interactions
    - $f(i) = \sum_{k \neq i} f(i,k)$ … f(i,k) = force on i from k
    -
    - $f(i,k) = c*v/||v||^3$ in 3 dimensions or $f(i,k) = c*v/||v||^2$ in 2 dimensions

      v = vector from particle i to particle k , c = product of masses or charges

      $||v||$ = length of v
    - Obvious algorithm costs $O(n^2)$
    - Can we do better?

Applications

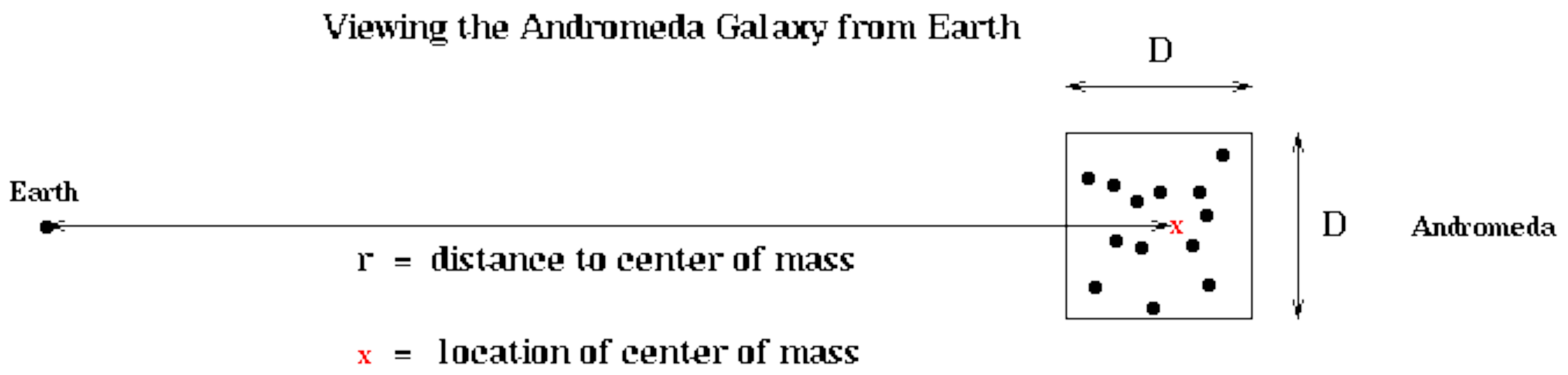
1. Astrophysics and Celestial Mechanics
   1. Intel Delta = 1992 supercomputer, 512 Intel i860s
   2. 17 million particles, 600 time steps, 24 hours elapsed time

      M. Warren and J. Salmon

      Gordon Bell Prize at Supercomputing 92
   3. Sustained 5.2 Gflops = 44K Flops/particle/time step
   4. 1% accuracy
   5. Direct method (17 Flops/particle/time step) at 5.2 Gflops would have taken 18 years, 6570 times longer
2. Plasma Simulation
3. Molecular Dynamics
4. Electron-Beam Lithography Device Simulation
5. Fluid Dynamics (vortex method)
6. Good sequential algorithms too!

Reducing the number of particles in the force sum

1. All later divide and conquer algorithms use same intuition
2. Consider computing force on earth due to all celestial bodies
   1. Look at night sky, # terms in force sum ≥ number of visible stars
   2. Oops! One “star” is really the Andromeda galaxy, which contains billions of real stars
      1. Seems like a lot more work than we thought …
3. Don’t worry, ok to approximate all stars in Andromeda by a single point at its center of mass (CM) with same total mass (TM)
   - D = size of box containing Andromeda , r = distance of CM to Earth
   - Require that D/r be “small enough”

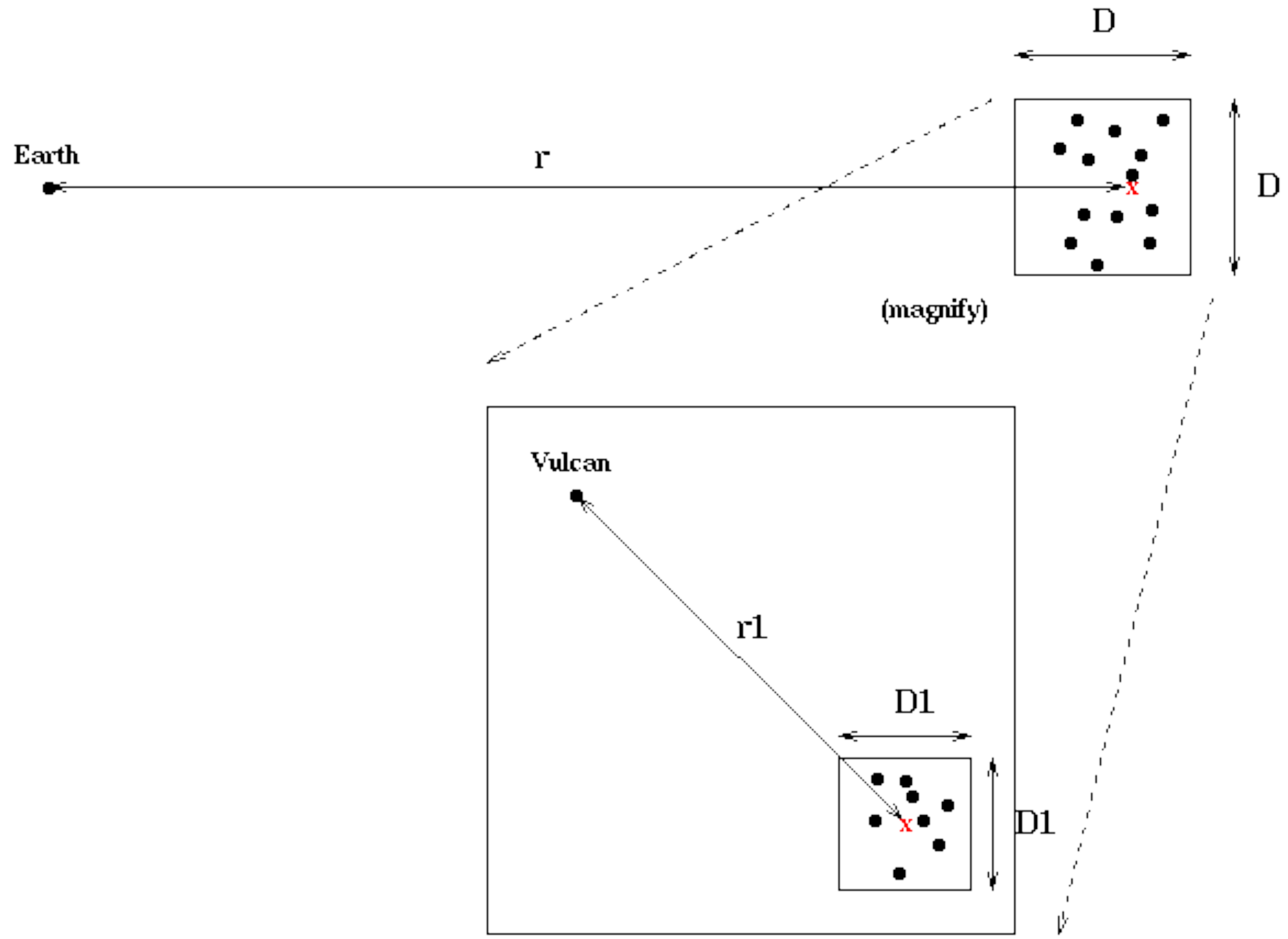


   - Idea not new: Newton approximated earth and falling apple by CMs

What is new: Using points at CM recursively

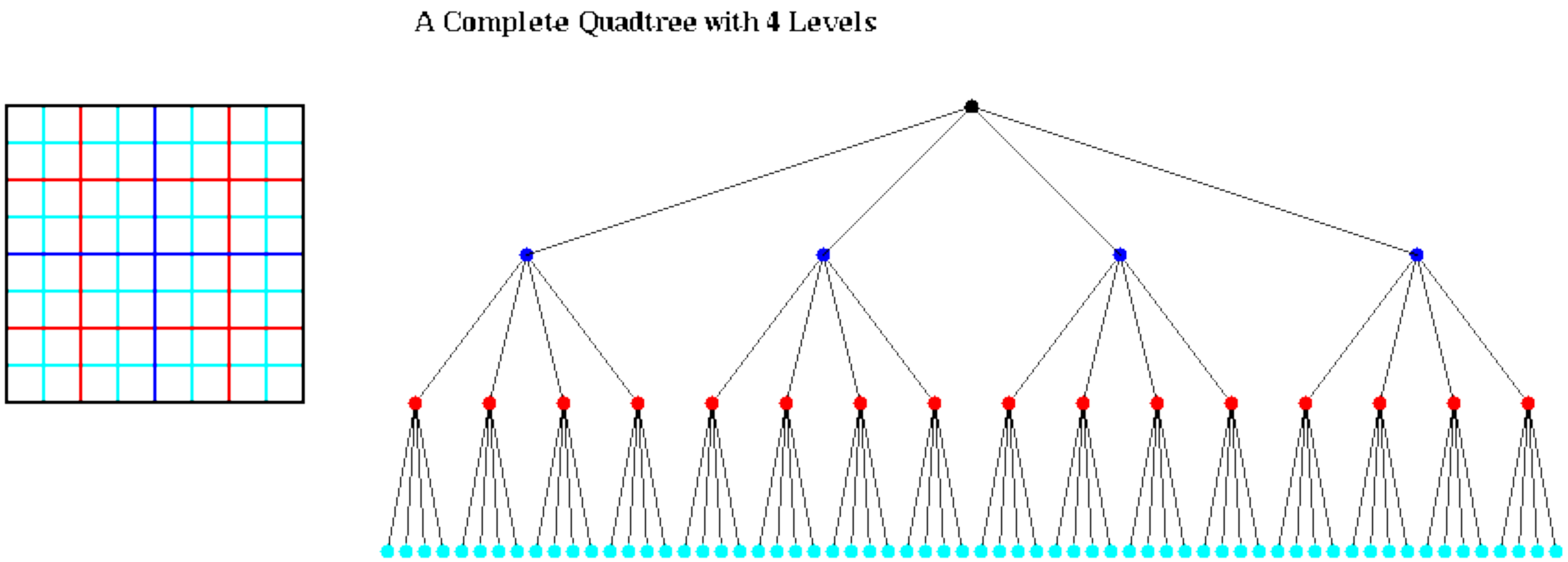
- From Andromeda's point of view, Milky Way is also a point mass
- Within Andromeda, picture repeats itself
  - As long as D1/r1 is small enough, stars inside smaller box can be replaced by their CM to compute the force on Vulcan
  - Boxes nest in boxes recursively

Replacing Clusters by their Centers of Mass Recursively



Quad Trees

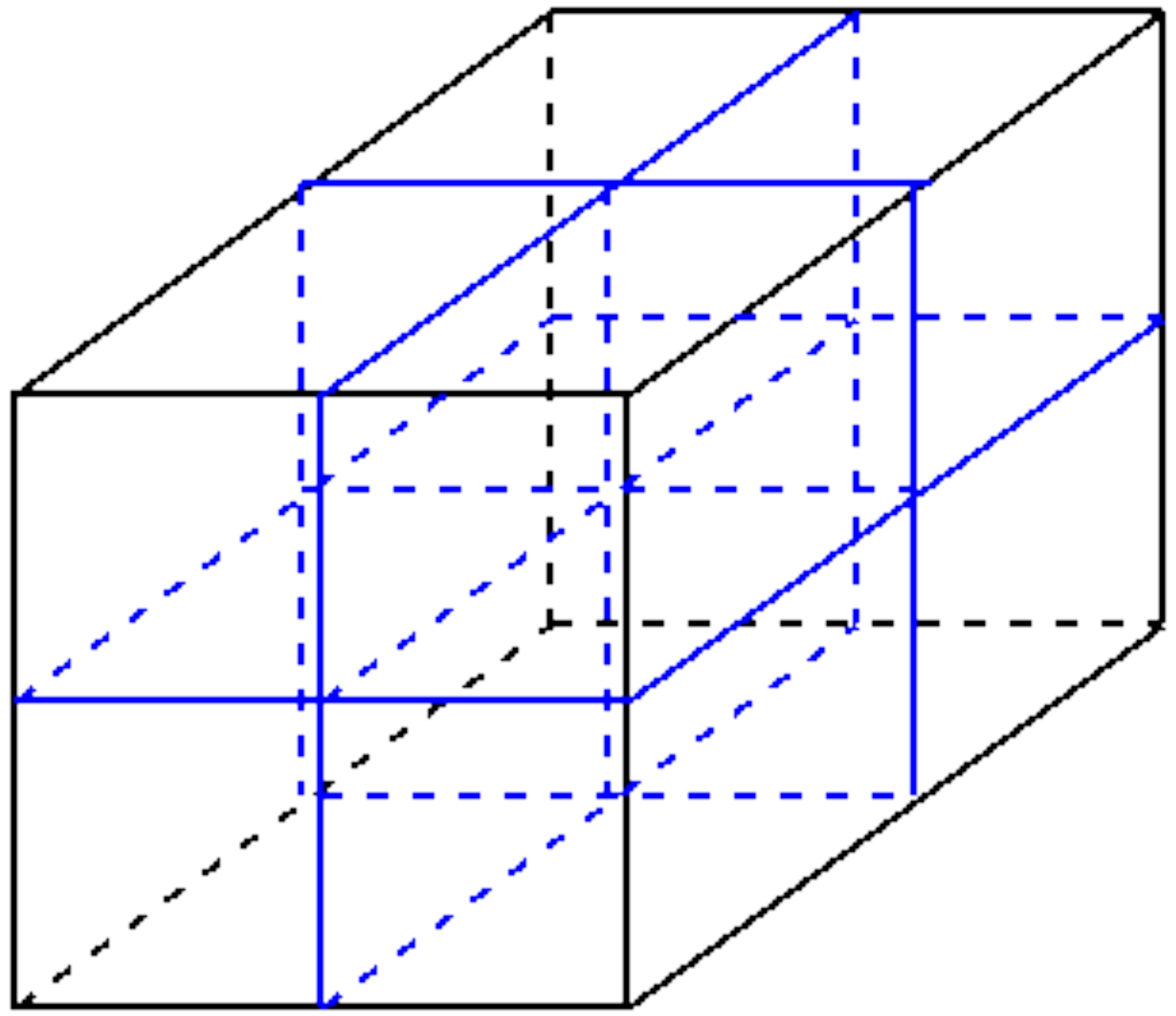
- Data structure to subdivide the plane
  - Nodes can contain coordinates of center of box, side length
  - Eventually also coordinates of CM, total mass, etc.
- In a complete quad tree, each nonleaf node has 4 children

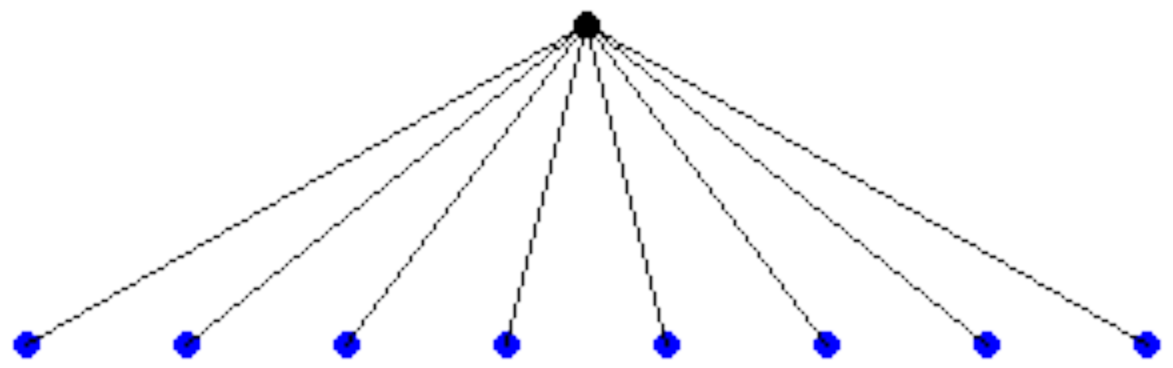


Oct Trees

- Similar Data Structure to subdivide space

2 Levels of an Octree



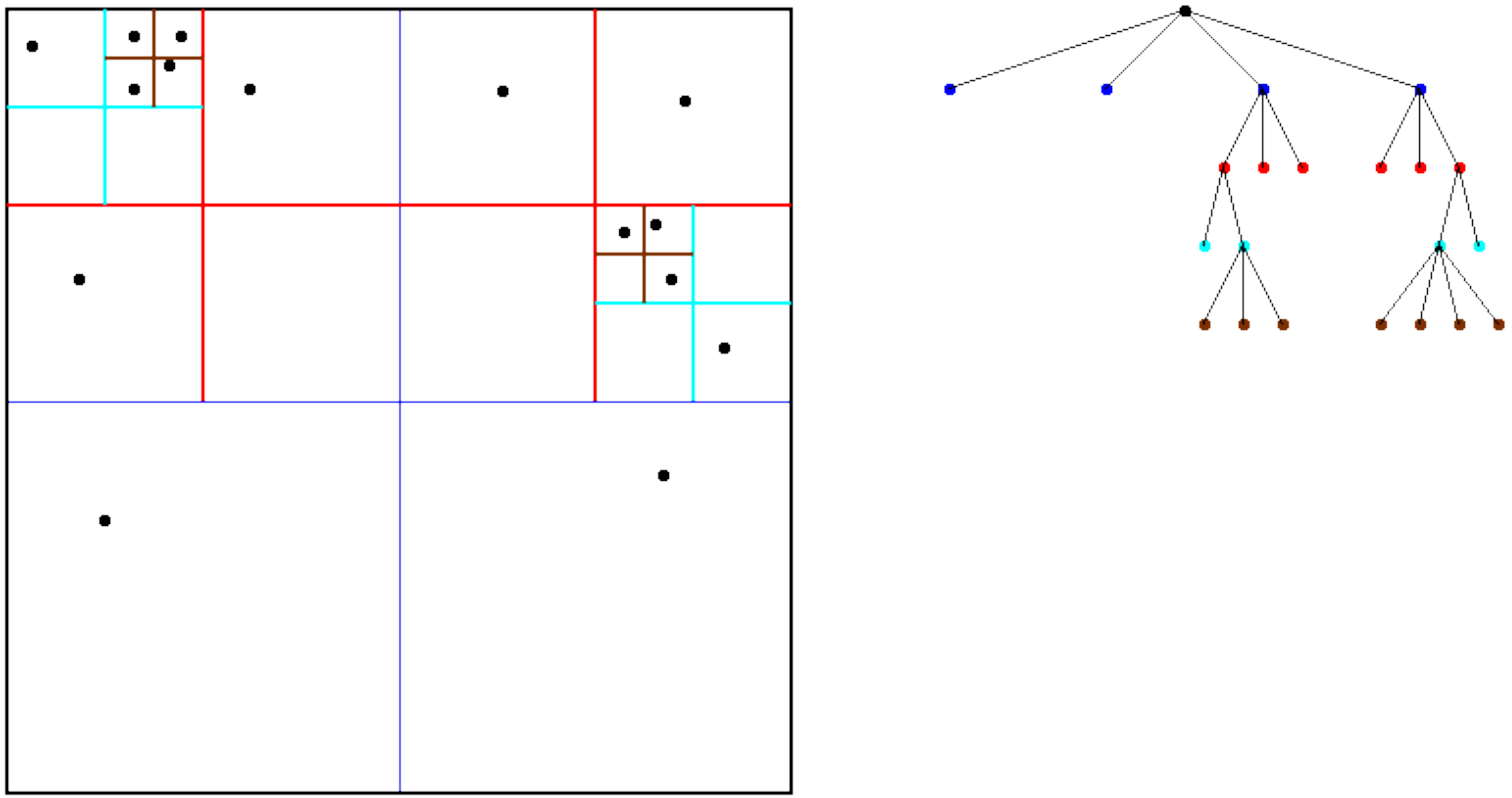


Using Quad Trees and Oct Trees

- All these algorithms begin by constructing a tree to hold all the particles
- Interesting cases have non-uniformly distributed particles
  - In a complete tree most nodes would be empty, a waste of space and time
- ***Adaptive*** Quad (Oct) Tree only subdivides space where particles are located

Example of an Adaptive Quad Tree

Adaptive quadtree where no square contains more than 1 particle



Adaptive Quad Tree Algorithm (Oct Tree analogous)

```
Procedure Quad_Tree_Build
    Quad_Tree = {}
    for j = 1 to N                                    … loop over all N particles
         Quad_Tree_Insert(j, root)           … insert particle j in QuadTree
    endfor
    …   At this point, each leaf of Quad_Tree will have 0 or 1 particles
    …   There will be 0 particles when some sibling has 1
    Traverse the Quad_Tree eliminating empty leaves  … via, say Breadth First Search
```

```
Procedure Quad_Tree_Insert(j, n) … Try to insert particle j at node n in Quad_Tree
    if n is an internal node                 … n has 4 children
        determine which child c of node n contains particle j
        Quad_Tree_Insert(j, c)
    else if n contains 1 particle    …  n is a leaf
        add n's 4 children to the Quad_Tree
        move the particle already in n into the child containing it
        let c be the child of n containing j
        Quad_Tree_Insert(j, c)
    else                                              …  n empty
        store particle j in node n
    end
```

Adaptive Quad Tree Algorithm (Oct Tree analogous)

```
Procedure Quad_Tree_Build
    Quad_Tree = {}
    for j = 1 to N                                  … loop over all N particles
         Quad_Tree_Insert(j, root)           … insert particle j in QuadTree
    endfor
    …   At this point, each leaf of Quad_Tree will have 0 or 1 particles
    …   There will be 0 particles when some sibling has 1
    Traverse the Quad_Tree eliminating empty leaves  … via, say Breadth First Search
```

```
Procedure Quad_Tree_Insert(j, n) … Try to insert particle j at node n in Quad_Tree
    if n is an internal node                … n has 4 children
        determine which child c of node n contains particle j
        Quad_Tree_Insert(j, c)
    else if n contains 1 particle    …  n is a leaf
        add n’s 4 children to the Quad_Tree
        move the particle already in n into the child containing it
        let c be the child of n containing j
        Quad_Tree_Insert(j, c)
    else                                            …  n empty
        store particle j in node n
    end
```

Easy change for > 1 particles/leaf

Cost of Adaptive Quad Tree Construction

- Cost ≤ N * maximum cost of Quad_Tree_Insert

  = O(N * maximum depth of Quad_Tree)

- Uniform Distribution of particles
  - Depth of Quad_Tree = O(log N)
  - Cost ≤ O(N * log N)

- Arbitrary distribution of particles
  - Depth of Quad_Tree = O(# bits in particle coords) = O(b)
  - Cost ≤ O(b N)

Barnes-Hut Algorithm

- "A Hierarchical O(n log n) force calculation algorithm",
  J. Barnes and P. Hut, Nature, v. 324 (1986), many later papers
- Good for low accuracy calculations:

  Root-Mean-Square error $= (\Sigma_k \|\text{approx } f(k) - \text{true } f(k)\|^2 / \|\text{true } f(k)\|^2 / N)^{1/2}$
  $\sim 1\%$

  (other measures better if some true f(k) ~ 0)

- High Level Algorithm (in 2D, for simplicity):

```
1) Build the QuadTree using QuadTreeBuild
     … already described, cost = O( N log N) or O(b N)
2) For each node = subsquare in the QuadTree, compute the
    CM and total mass (TM)  of all the particles it contains
    … "post order traversal" of QuadTree, cost = O(N log N) or O(b N)
3) For each particle, traverse the QuadTree to compute the force on it,
    using the CM and TM of "distant" subsquares
    … core of algorithm
    … cost depends on accuracy desired but still O(N log N) or O(bN)
```

Step 2 of BH: compute CM and total mass of each node

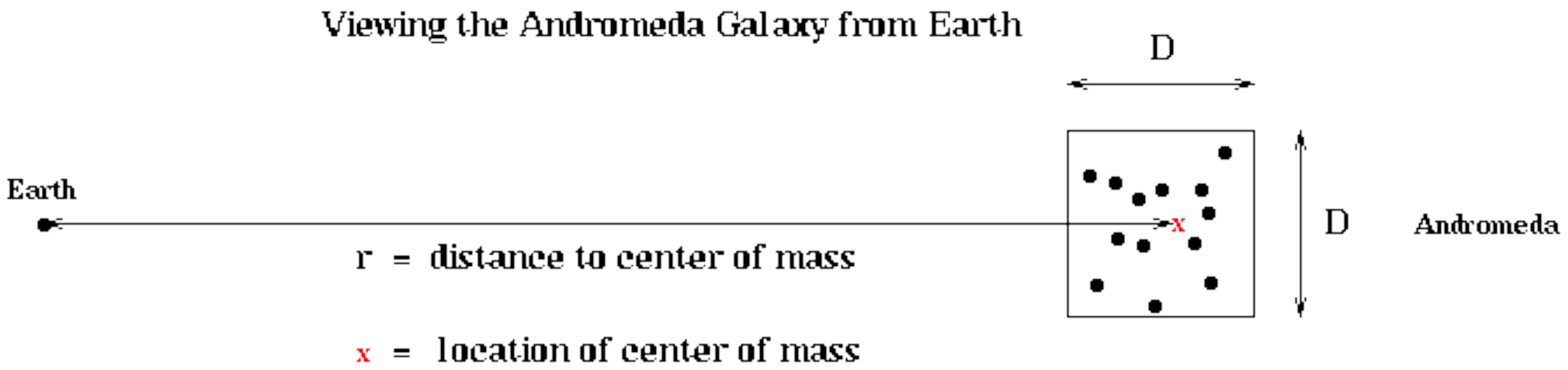
```
… Compute the CM = Center of Mass and TM = Total Mass of all the particles
… in each node of the QuadTree
( TM, CM ) = Compute_Mass( root )
```

```
function ( TM, CM ) = Compute_Mass( n )    … compute the CM and TM of node n
      if n contains 1 particle
           … the TM and CM are identical to the particle's mass and location
           store (TM, CM) at n
           return (TM, CM)
      else       … "post order traversal": process parent after all children
           for all children c(j) of n  … j = 1,2,3,4
                 ( TM(j), CM(j) ) = Compute_Mass( c(j) )
           endfor
           TM = TM(1) + TM(2) + TM(3) + TM(4)
                  … the total mass is the sum of the children's masses
           CM = ( TM(1)*CM(1) + TM(2)*CM(2) + TM(3)*CM(3) + TM(4)*CM(4) ) / TM
                  … the CM is the mass-weighted sum of the children's centers of mass
           store ( TM, CM ) at n
           return ( TM, CM )
       end if
```

Cost = O(# nodes in QuadTree) = O(N log N) or O(b N)

Step 3 of BH: compute force on each particle

- For each node = square, can approximate force on particles outside the node due to particles inside node by using the node's CM and TM
- This will be accurate enough if the node is "far away enough" from the particle
- For each particle, use as few nodes as possible to compute force, subject to accuracy constraint
- Need criterion to decide if a node is "far enough" from a particle
  - D = side length of node
  - r = distance from particle to CM of node
  - **θ = user supplied error tolerance < 1**
  - Use CM and TM to approximate force of node on box if $D/r < \theta$



Computing force on a particle due to a node

- Suppose node n, with CM and TM, and particle k, satisfy $D/r < \theta$
- Let (x_k, y_k, z_k) be coordinates of k, m its mass
- Let (x_{CM}, y_{CM}, z_{CM}) be coordinates of CM
- $r = ((x_k - x_{CM})^2 + (y_k - y_{CM})^2 + (z_k - z_{CM})^2)^{1/2}$
- G = gravitational constant
- Force on k $\approx G * m * TM * (x_{CM} - x_k , y_{CM} - y_k , z_{CM} - z_k) / r^3$

Details of Step 3 of BH

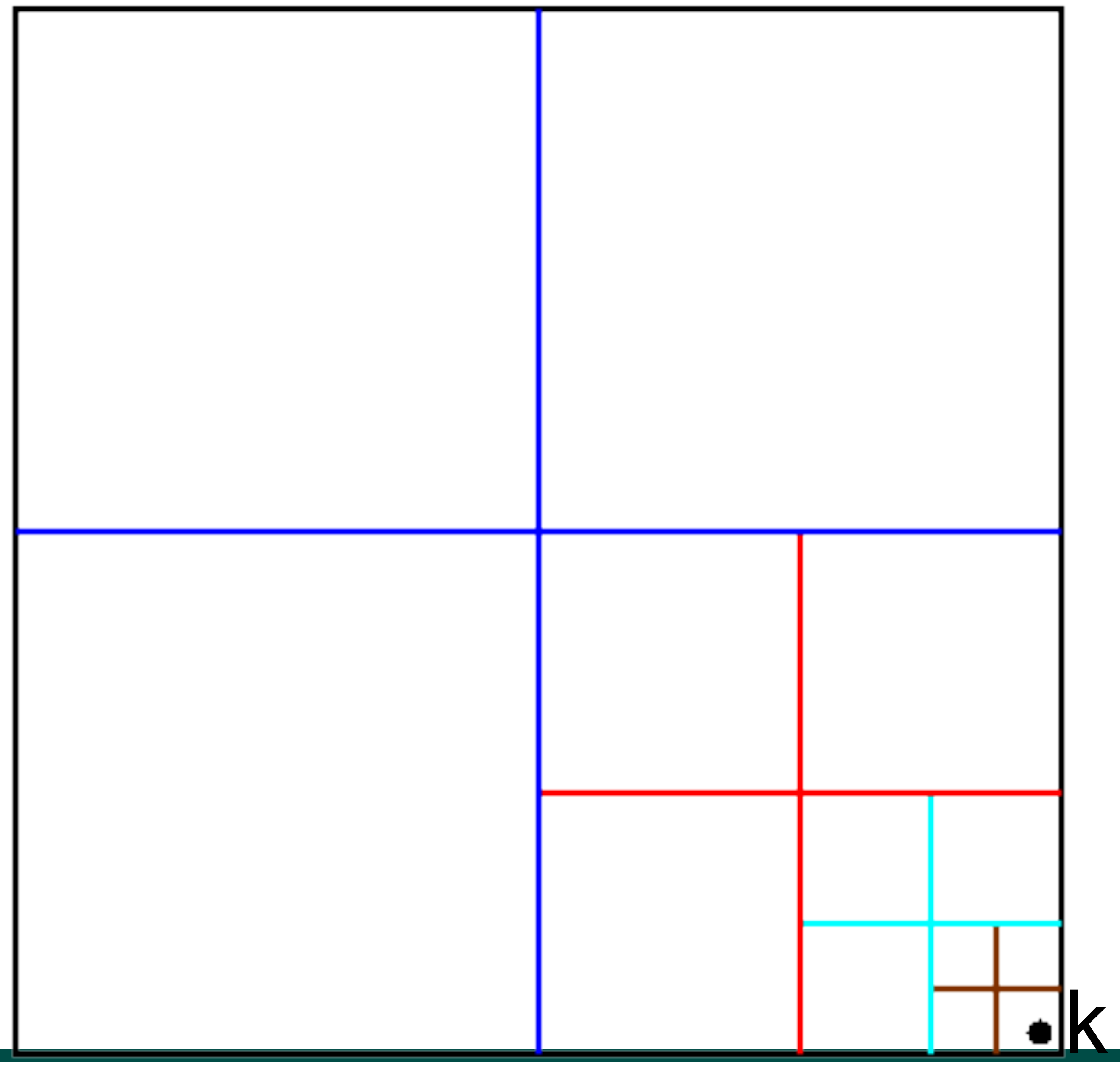
```
… for each particle, traverse the QuadTree to compute the force on it
for k = 1 to N
    f(k) = TreeForce( k, root )
                   … compute force on particle k due to all particles inside root
endfor
```

```
function f = TreeForce( k, n )
    … compute force on particle k due to all particles inside node n
    f = 0
    if n contains one particle  … evaluate directly
        f = force computed using formula on last slide
    else
        r = distance from particle k to CM of particles in n
        D = size of n
        if  D/r  < θ     … ok to approximate by CM and TM
             compute f using formula from last slide
        else              … need to look inside node
             for all children c of n
                   f = f + TreeForce ( k, c )
             end for
        end if
    end if
```

Analysis of Step 3 of BH

- Correctness follows from recursive accumulation of force from each subtree
  - Each particle is accounted for exactly once, whether it is in a leaf or other node
- Complexity analysis
  - Cost of TreeForce(k, root) = O(depth in QuadTree of leaf containing k)
  - Proof by Example (for θ>1):
    - For each undivided node = square,
    - (except one containing k), D/r < 1 < θ
    - There are 3 nodes at each level of
    - the QuadTree
    - There is O(1) work per node
    - Cost = O(level of k)
  - Total cost = O(Σ_k level of k) = O(N log N)
    - Strongly depends on θ

Sample Barnes−Hut Force calculation
For particle in lower right corner
Assuming theta > 1



Other N-Body Algorithms

- Fast Multi-Pole Method: "A fast algorithm for particle simulation", L. Greengard and V. Rokhlin, J. Comp. Phys. V. 73, 1987, many later papers
  - Greengard: 1987 ACM Dissertation Award, 2006 NAE&NAS; Rohklin: 1999 NAS
- Differences from Barnes-Hut
  - FMM computes the *potential* at every point, not just the force
  - FMM uses more information in each box than the CM and TM, so it is both more accurate and more expensive
  - In compensation, FMM accesses a fixed set of boxes at every level, independent of D/r
  - BH uses fixed information (CM and TM) in every box, but # boxes increases with accuracy. FMM uses a fixed # boxes, but the amount of information per box increase with accuracy.
  - O(N log N) for BH, O(N) for FMM

Parallelizing Hierarchical N-Body codes

- Barnes-Hut and other related algorithms have similar computational structure:

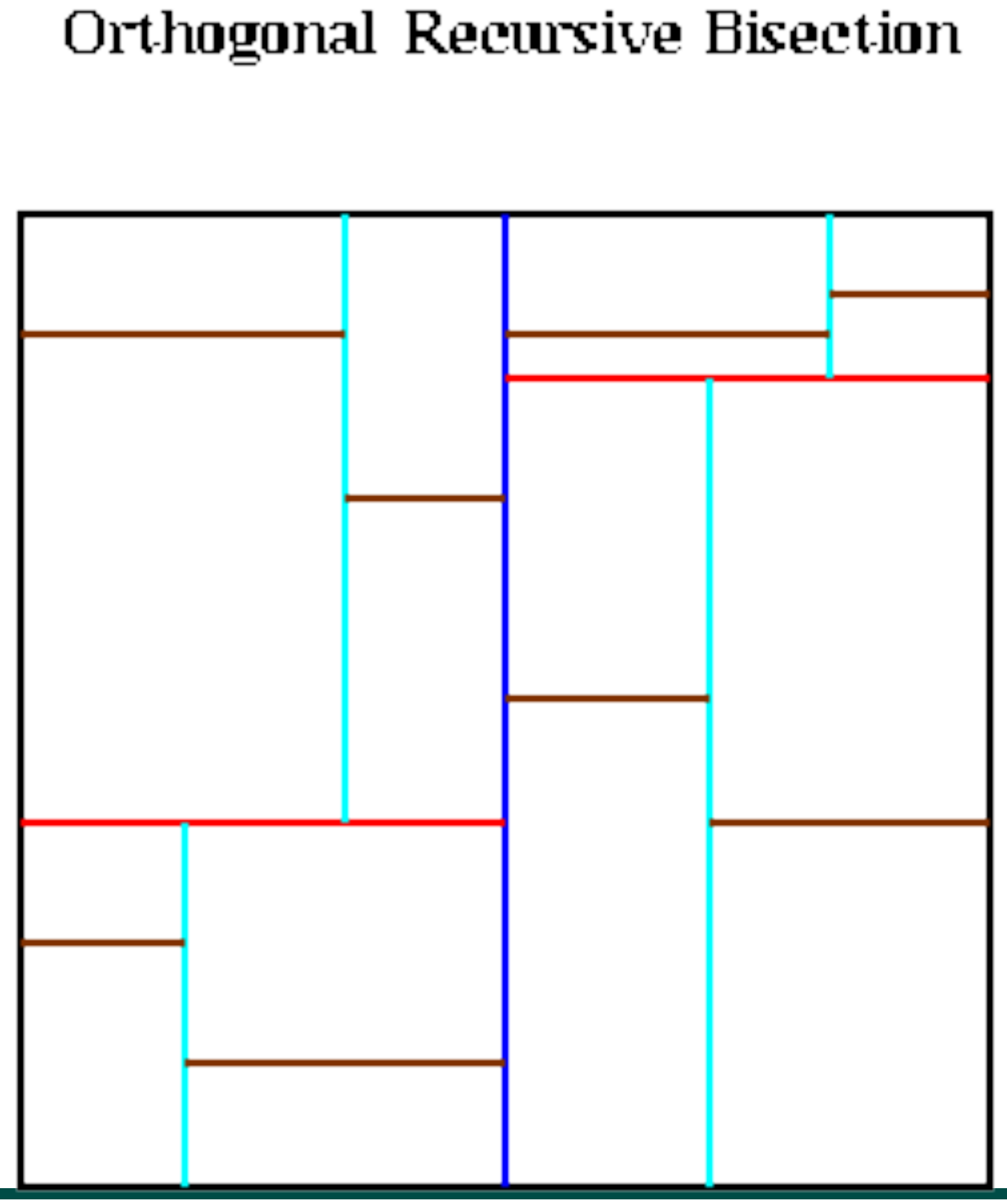
1) Build the QuadTree (OctTree)
2) Traverse QuadTree from leaves to root and build TM and CM
3) (not needed for BH) Traverse QuadTree from root to leaves and build any inner expansions
3) Traverse QuadTree to accumulate forces for each particle

- One parallelization scheme works for all of them
  - Based on D. Blackston and T. Suel, Supercomputing 97
    - UCB PhD Thesis, David Blackston, “Pbody”
    - Autotuner for N-body codes
  - Assign regions of space to tasks. Rule of thumb: one task per core
  - Regions may have different shapes, need to load balance
    - Each region should have around N/p particles
  - Each task will store part of QuadTree containing all particles (=leaves) in its region, and their ancestors in QuadTree
    - Top of tree stored by all tasks, lower nodes may also be shared
  - Each task will also store adjoining parts of QuadTree needed to compute forces for particles it owns
    - Subset of QuadTree needed by a task called the Locally Essential Tree (LET)
  - Given the LET, all force accumulations (step 4)) can be done in parallel, without synchronization

Load Balancing Scheme: Orthogonal Recursive Bisection (ORB)

- Warren and Salmon, Supercomputing 92
- Recursively split region along axes into regions containing equal numbers of particles
- Works well for 2D, not for 3D

Partitioning for 16 procs:



Summary

- N-body problem is common in simulations for astrophysics, electromagnetic, molecular biology, plasma theory, micro-particles, fluid dynamics…
- Same principle holds, with perhaps different physics (gravitational vs. electromagnetic forces)
- Main idea: approximate a group of distant bodies with a single one
- Recursive decomposition of the problem
- Adaptive quad (oct) trees enable problem decomposition
- Parallelization requires splitting the space into equal chunks
- Splitting the QuadTree (OctTree) into equal chunks also requires sharing some data between tasks