CSE 6220/CX 4220
Introduction to HPC

Lecture 18: Hierarchical Methods for the N-Body Problem

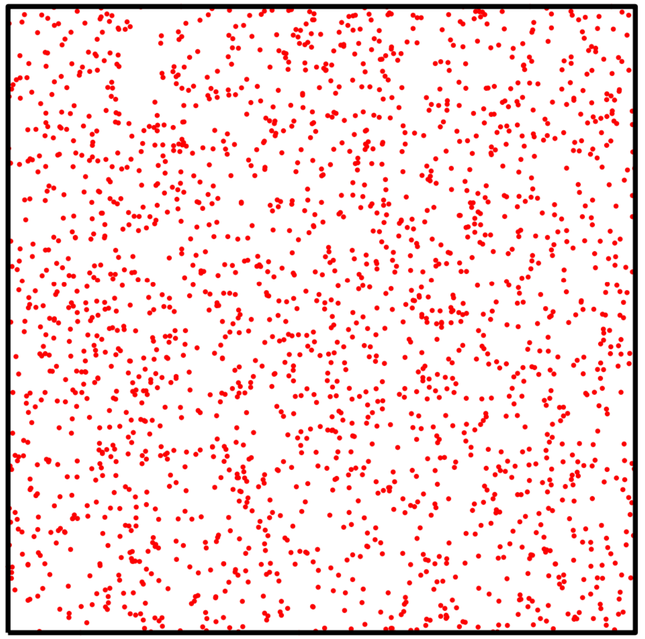
Helen Xu
hxu615@gatech.edu

Georgia Tech College of Computing
School of Computational Science and Engineering

Slide credits mostly to Prof. James Demmel

Motivation

- **Computational simulation** is becoming an accepted paradigm for scientific discovery - but can involve several million variables
- Most large problems boil down to solution of linear systems or matrix-vector product
- Regular product requires $O(n^2)$ time and memory



Applying the idea recursively: cost drops to $O(n \log n)$ or even $O(n)$

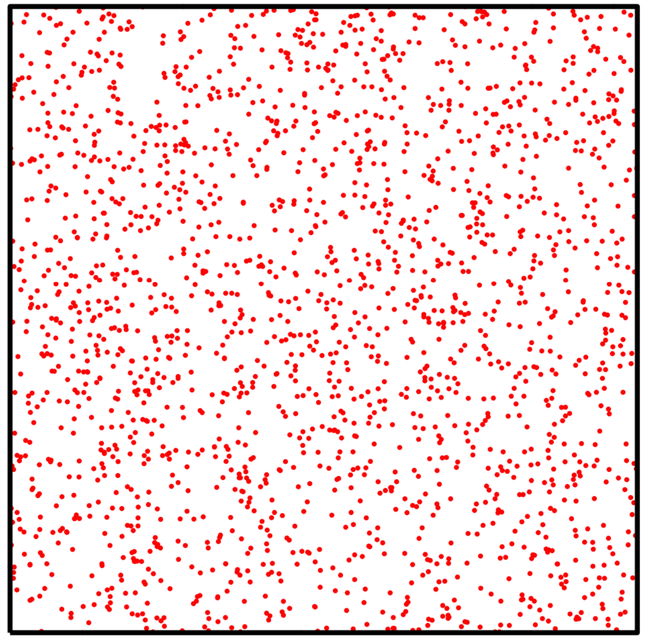
https://www.cscamm.umd.edu/programs/fam04/dg_lecture1.pdf

Diagram from https://amath.colorado.edu/faculty/martinss/2014_CBMS/Lectures/lecture02.pdf

Motivation

- Intuition - **compress dependence on "distant" data**
- Compressing data of groups at nearby points can cut cost (work, communication) at distant points

- Examples:
  - multigrid for elliptic PDE
  - multilevel graph partitioning (METIS, Chaco, …)
  - Barnes-Hut or Fast Multipole Method (FMM) for electrostatics/gravity/…



Applying the idea recursively: cost drops to $O(n \log n)$ or even $O(n)$

https://www.cscamm.umd.edu/programs/fam04/dg_lecture1.pdf

Diagram from https://amath.colorado.edu/faculty/martinss/2014_CBMS/Lectures/lecture02.pdf

Particle Simulation

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

Most complicated component in computing `f(i)` is **n-body force**, e.g., gravity or electrostatics

$$f(i) = \sum_{k \neq i} f(i,k)$$

f(i, k) = force on i from k

Obvious algorithm requires computing all-to-all interactions for $O(n^2)$, but we can do better

Motivation

- Listed as one of the top 10 algorithms of the 20th century (IEEE magazine)
- Frequently used in scientific computing

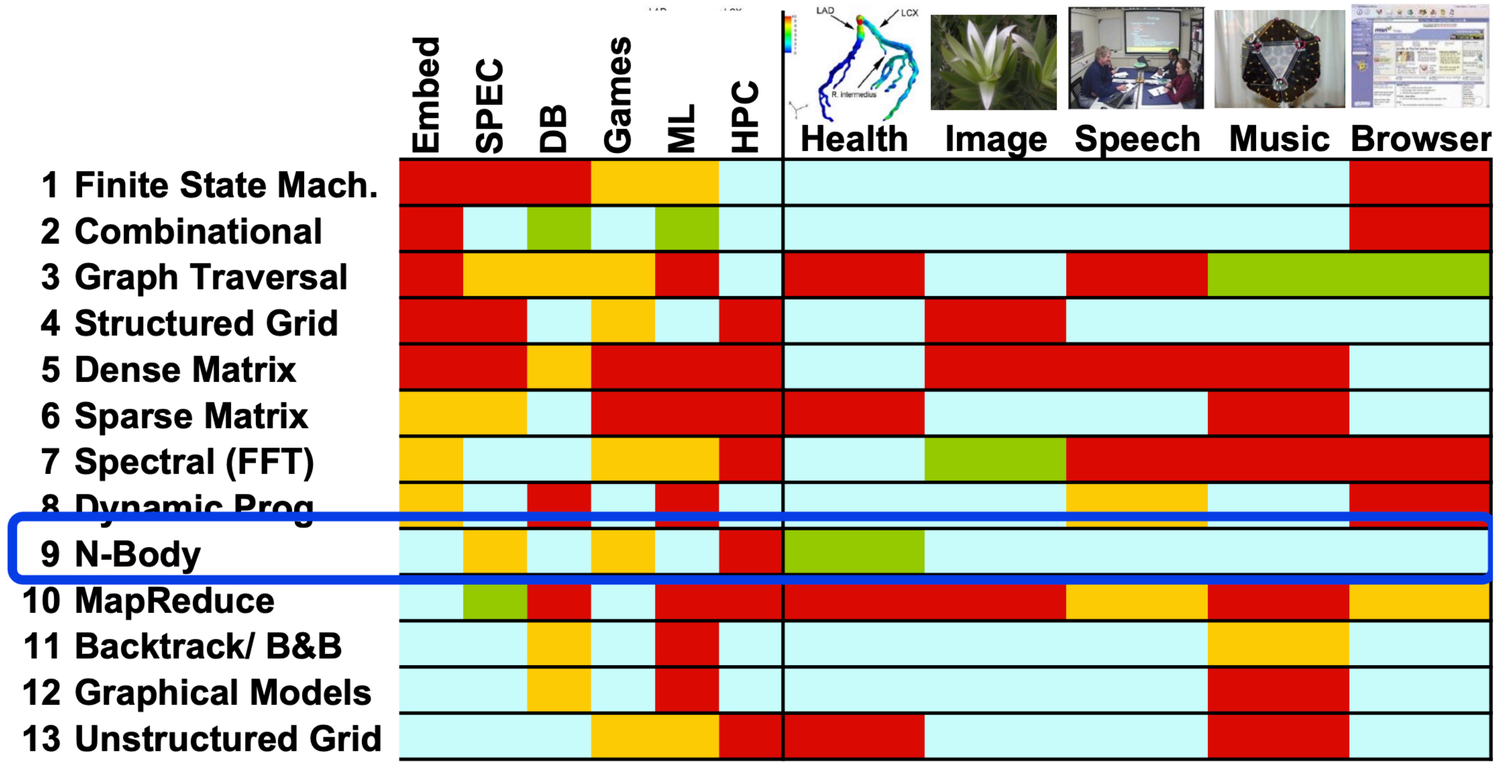


diagram from UC Berkeley CS267

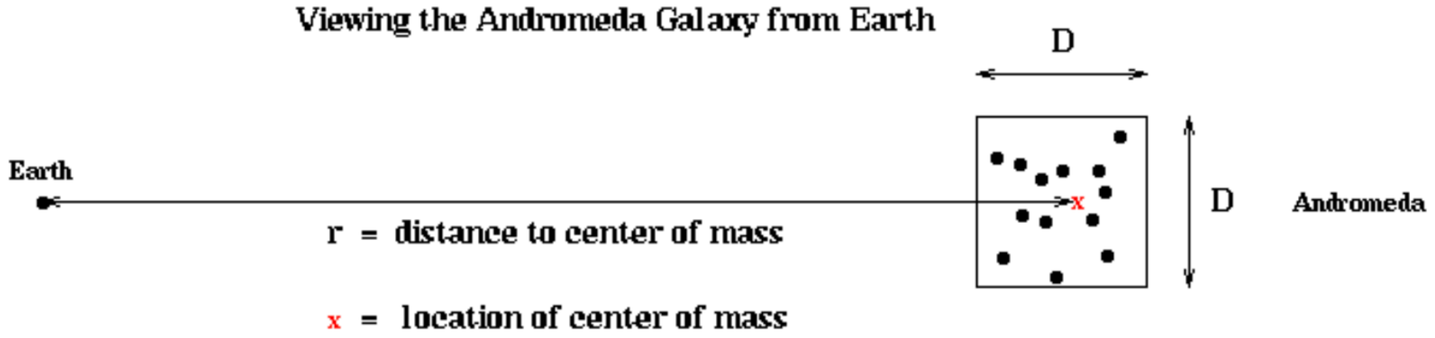
from “The Landscape of Parallel Computing Research: A View from Berkeley” by Asanović et al. ’06

Applications

- **Astrophysics and Celestial Mechanics - 1992**
  - Intel Delta = 1992 supercomputer, 512 Intel i860s
  - 17 million particles, 600 time steps, 24 hours elapsed time
    - M. Warren and J. Salmon
    - Gordon Bell Prize at Supercomputing 1992
  - Sustained 5.2 Gigaflops = 44K Flops/particle/time step
  - 1% accuracy
  - Direct method (17 Flops/particle/time step) at 5.2 Gflops would have taken 18 years, 6570 times longer
- **Vortex particle simulation of turbulence – 2009**
  - Cluster of 256 NVIDIA GeForce 8800 GPUs
  - 16.8 million particles
    - T. Hamada, R. Yokota, K. Nitadori. T. Narumi, K. Yasoki et al
    - Gordon Bell Prize for Price/Performance at Supercomputing 2009
  - Sustained 20 Teraflops, or $8/Gigaflop

Reducing the number of particles in the force sum

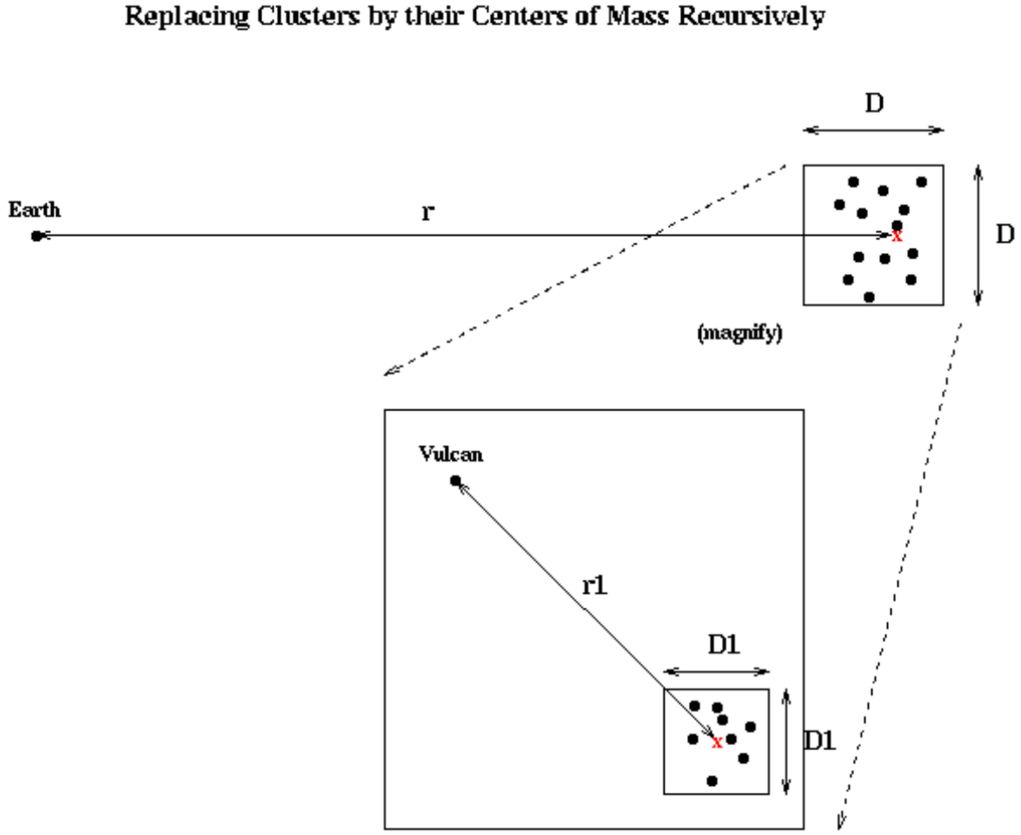
- All later divide and conquer algorithms use same intuition
- Consider computing force on earth due to all celestial bodies
  - Look at night sky, # terms in force sum ≥ number of visible stars
  - Oops! One "star" is really the Andromeda galaxy, which contains billions of real stars
    - Seems like a lot more work than we thought …
- 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

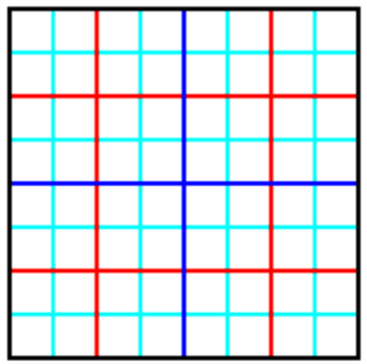
- From Andromeda's point of view, Milky Way is also a point mass
- Within Andromeda, picture repeats itself
  - As long as D1/r1 is small enough, stars inside smaller box can be replaced by their CM to compute the force on Vulcan
  - Boxes nest in boxes recursively

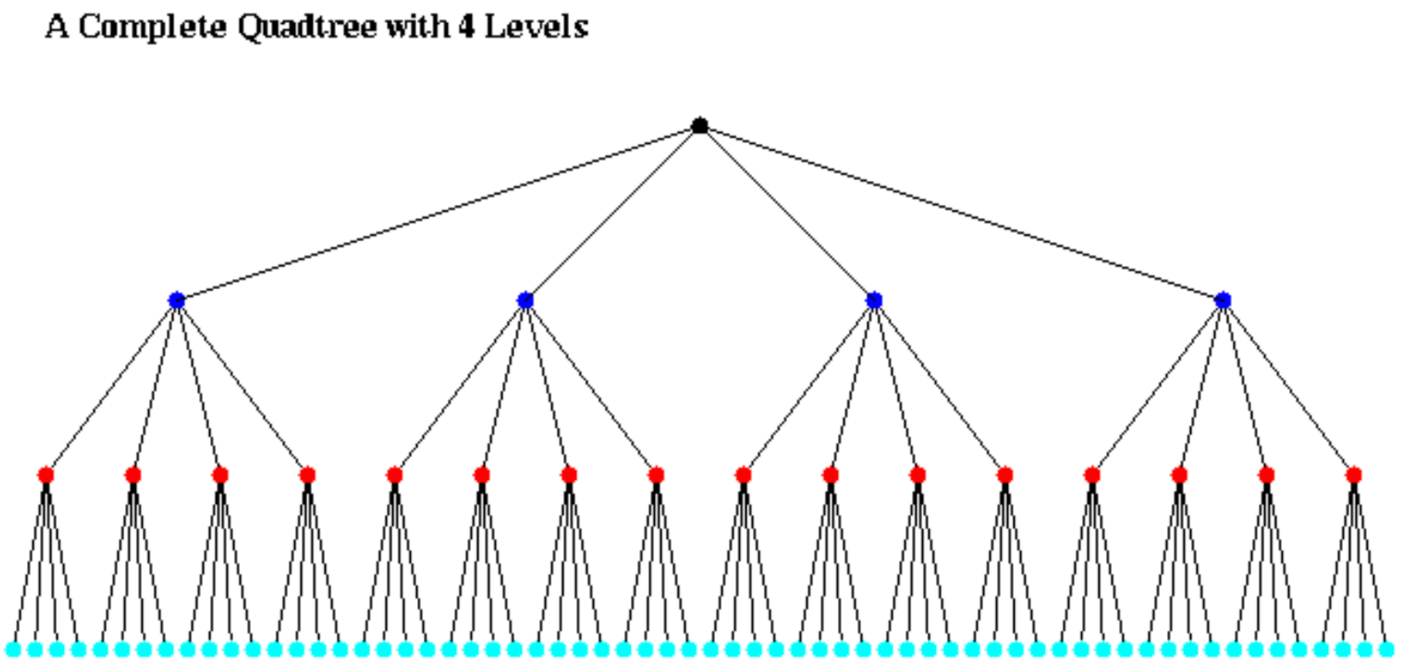


Basic Data Structures: Quad Trees and Oct Trees

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**

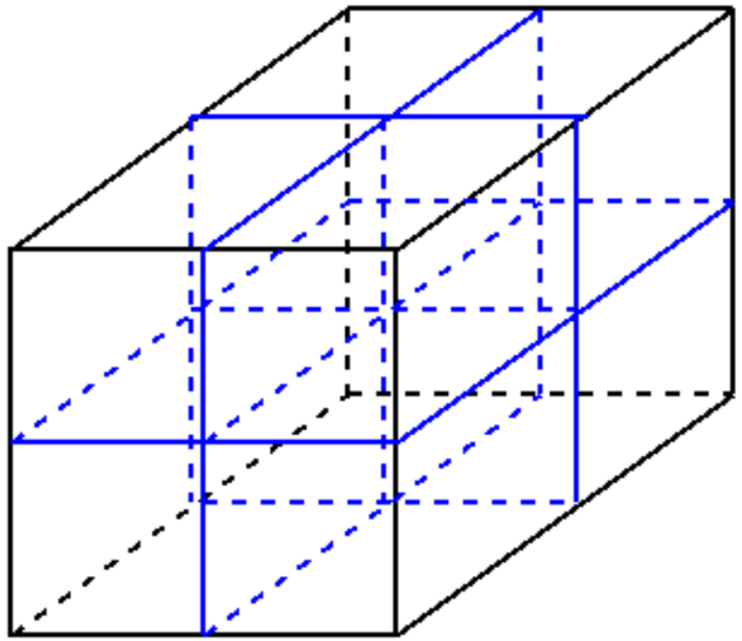


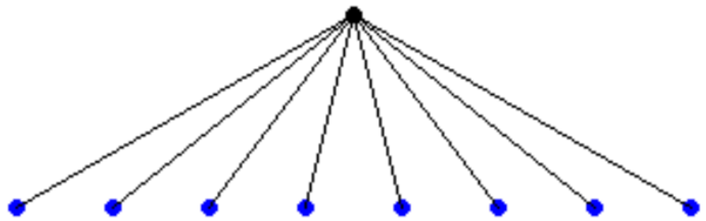


Oct Trees

- **Similar Data Structure to subdivide space**

2 Levels of an Octree

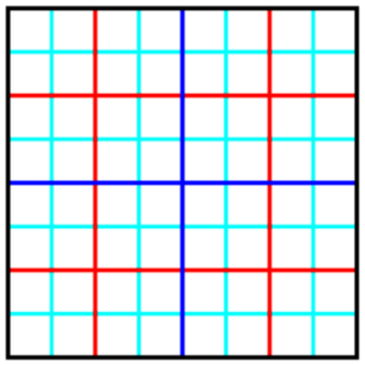


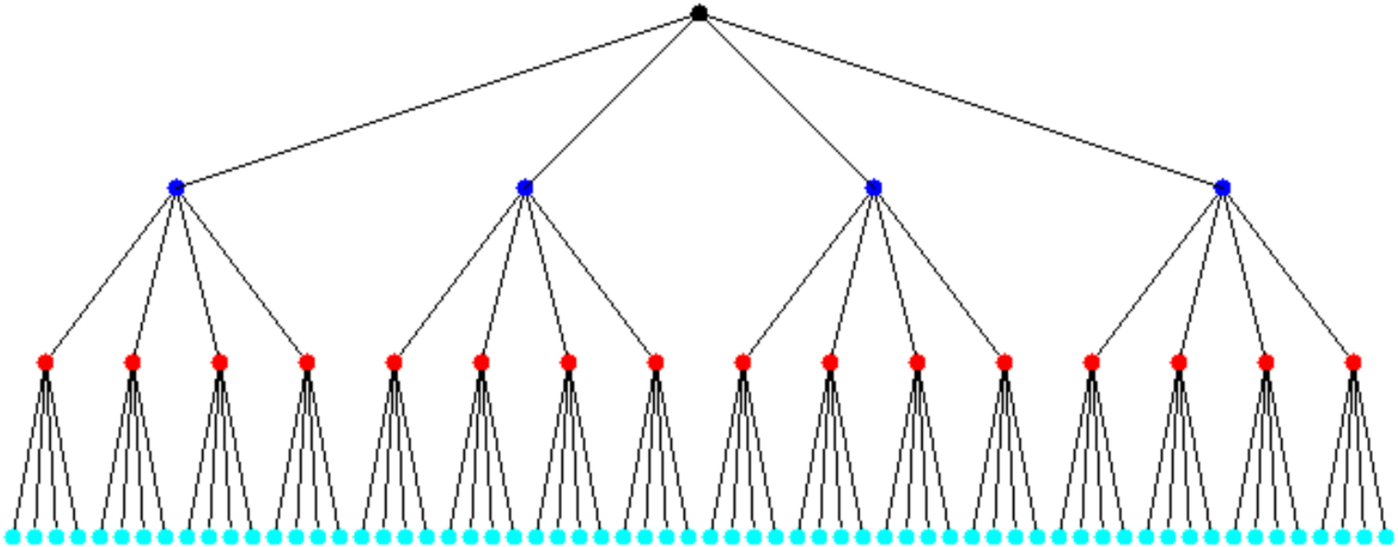


Using Quad Trees and Oct Trees

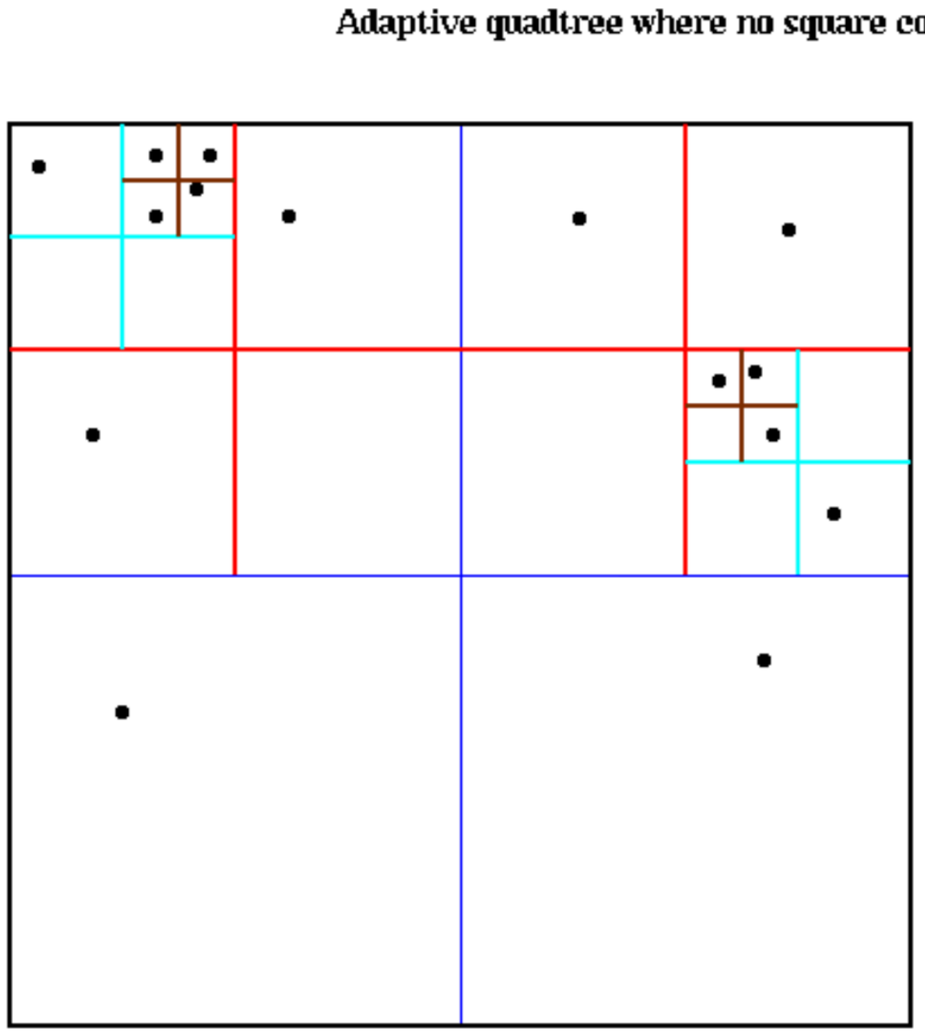
- All our algorithms begin by constructing **a tree to hold all the particles**
- Interesting cases have nonuniformly distributed particles

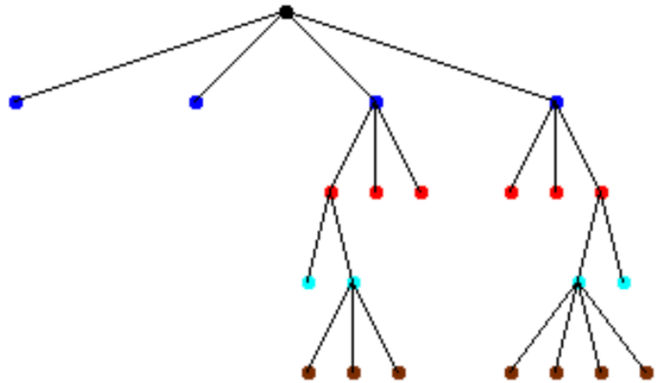
  In a complete tree most nodes would be empty, wasting space and time
- **Adaptive** Quad (Oct) Tree only subdivides space where particles are located





Example of an Adaptive Quad Tree





Child nodes enumerated counterclockwise from SW corner, empty ones excluded

In practice, have q>1 particles/square; tuning parameter

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 (BH)

An $O(n \log n)$ approximation algorithm for the N-Body Problem

Barnes-Hut Algorithm

- **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)

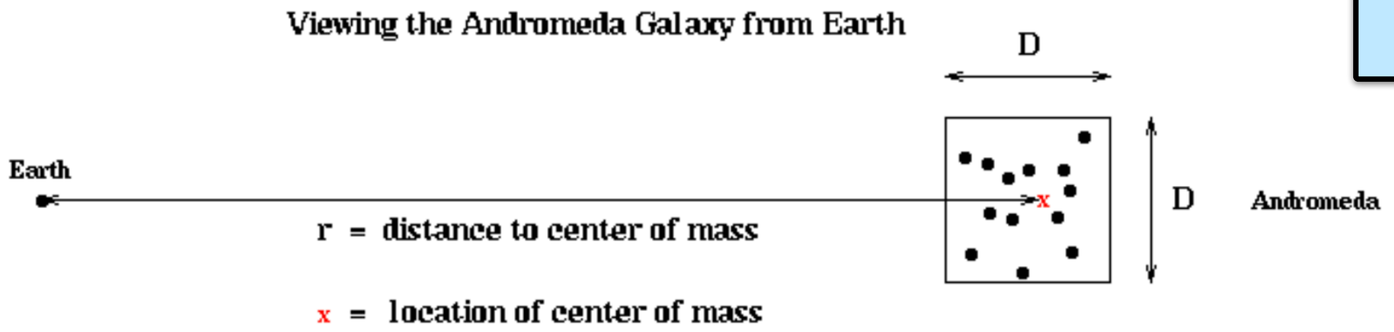
Center of mass of a cell is the weighted average of centers of its children

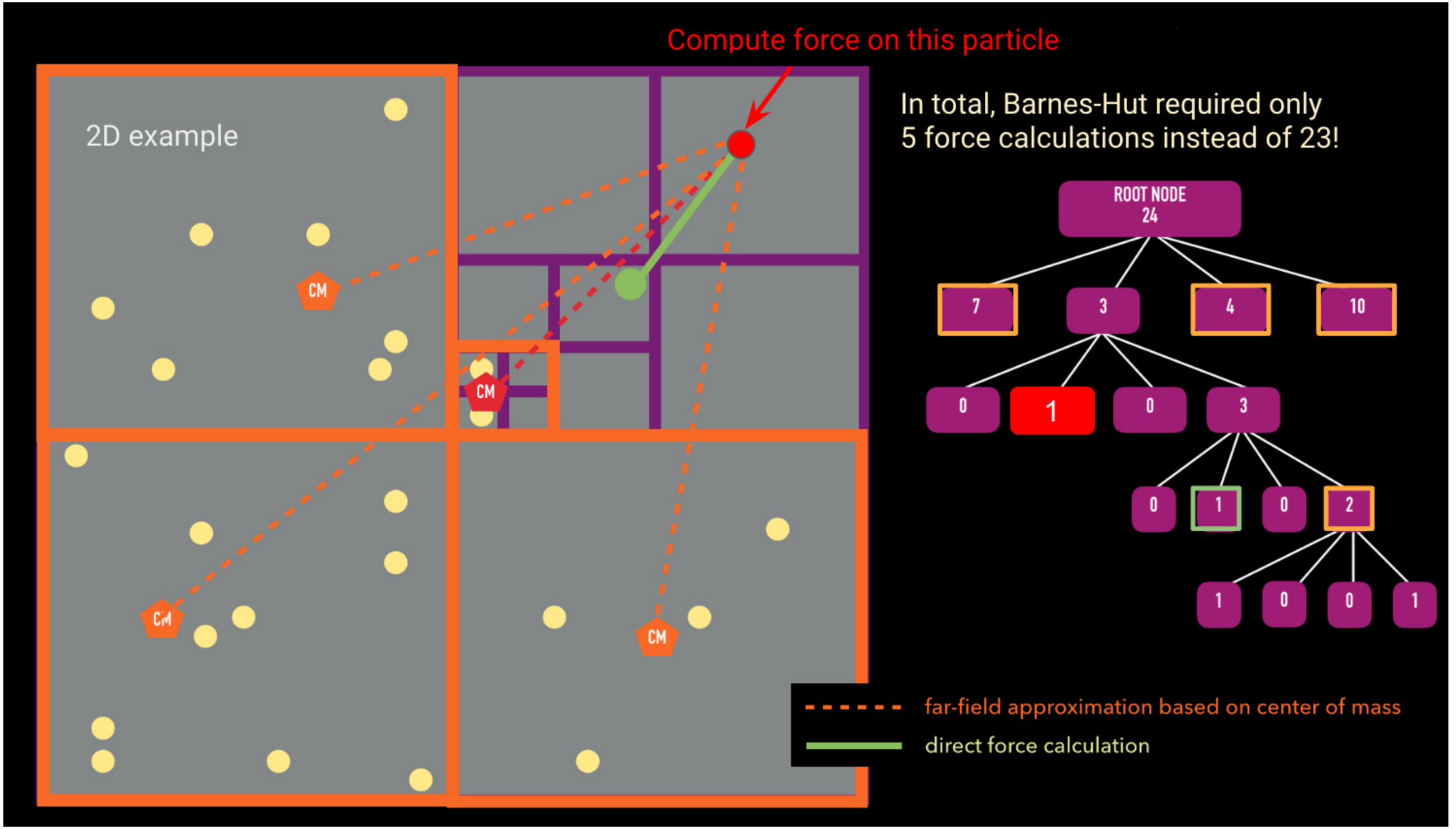
"A Hierarchical O(n log n) force calculation algorithm", J. Barnes and P. Hut, Nature, v. 324 (1986)

Step 3 of BH: Compute force on each particle

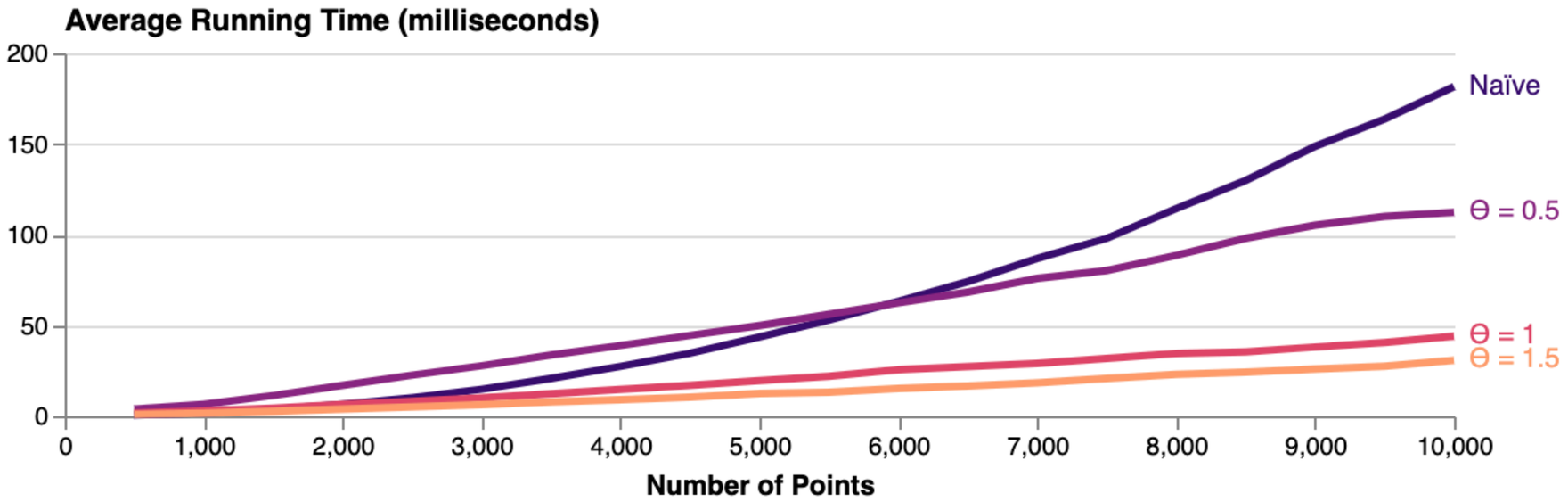
- For each node = square, can approximate force on particles outside the node due to particles inside node by using the node's CM and TM
- This will be accurate enough if the node if "far away enough" from the particle
- For each particle, use as few nodes as possible to compute force, subject to accuracy 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 < θ

If ratio of (cell width / node's center of mass) is more than Θ




Compute force on this particle
2D example
CM
CM
CM
CM
In total, Barnes-Hut required only
5 force calculations instead of 23!
ROOT NODE
24
7
3
4
10
0
1
0
3
0
1
0
2
1
0
0
1
far-field approximation based on center of mass
direct force calculation

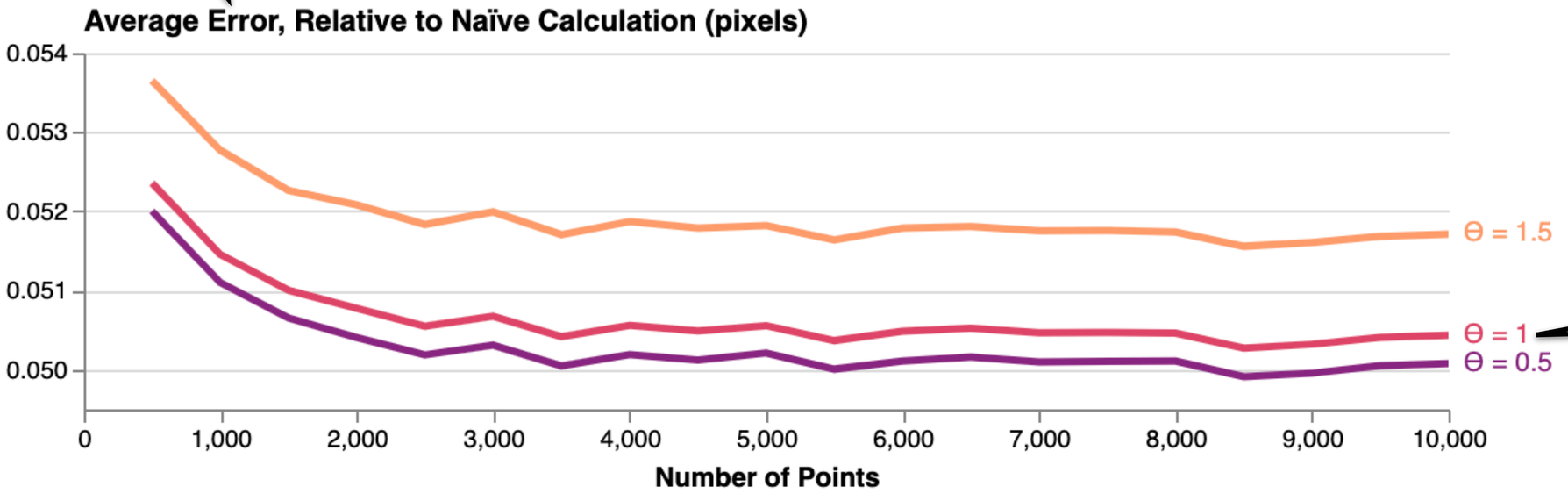
Example Performance Analysis



https://jheer.github.io/barnes-hut/

Example Error Analysis

Average vector distance between naive method and Barnes-Hut



In practice, 1.0 is a good default value

Fast Multipole Method (FMM): An O(N) approximate algorithm for the N-Body Problem

Most slides by Prof. Lexing Ying (talk here: https://www.youtube.com/watch?v=qMLIyZi8Sz0&ab_channel=IBMResearch)

Some diagrams from https://web.stanford.edu/~lexing/pfmm.pdf

Fast Multipole Method (FMM)

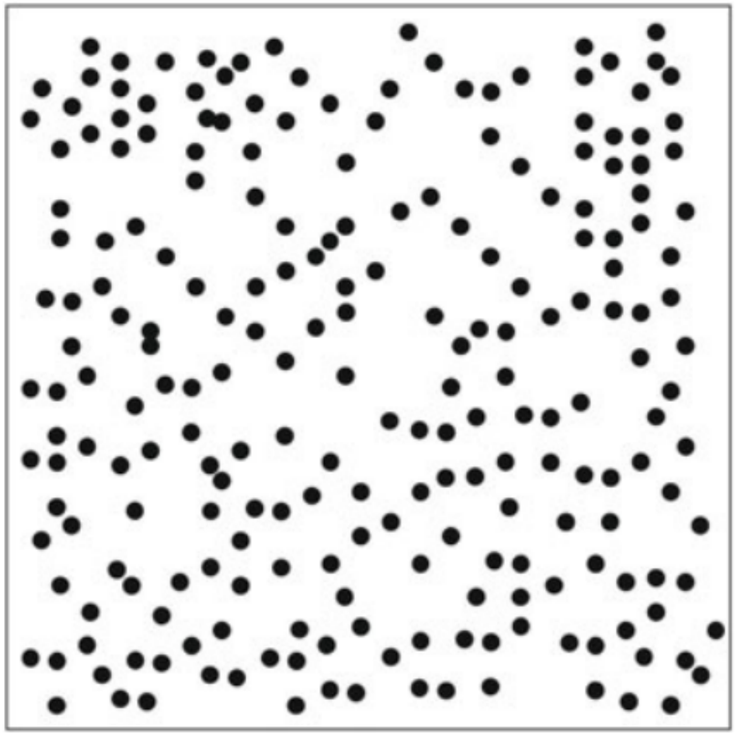
- "A fast algorithm for particle simulation", L. Greengard and V. Rokhlin, J. Comp. Phys. V. 73, 1987, many later papers
  - Many awards
- Differences from Barnes-Hut
  - FMM computes the *potential* at every point, not just the force
  - FMM uses more information in each box than the CM and TM, so it 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 increases with accuracy.
- FMM uses two kinds of expansions
  - Outer expansions represent potential outside node due to particles inside, analogous to (CM,TM)
  - Inner expansions represent potential inside node due to particles outside; *Computing this for every leaf node is the computational goal of FMM*
- First review potential, then return to FMM

N-body Problem Formulation

- Points $\{x_i, i = 1, \ldots, N\}$
- Charges $\{f_i, i = 1, \ldots, N\}$
- Kernel $G(x, y)$, for example
  - $G(x, y) = \ln(|x - y|)$ in 2D
  - $G(x, y) = 1/|x - y|$ in 3D
- Compute potentials for x_i for $i = 1, \ldots, N$

$$u_i = \sum_{j=1}^{N} G(x_i, x_j) f_j$$

Can be expressed with matrix-vector multiplication



Computational Cost

$$u_i = \sum_{j=1}^{N} G(x_i, x_j) f_j$$

- Direct computation is $O(N^2)$ steps : too costly
- Goal: lower the complexity

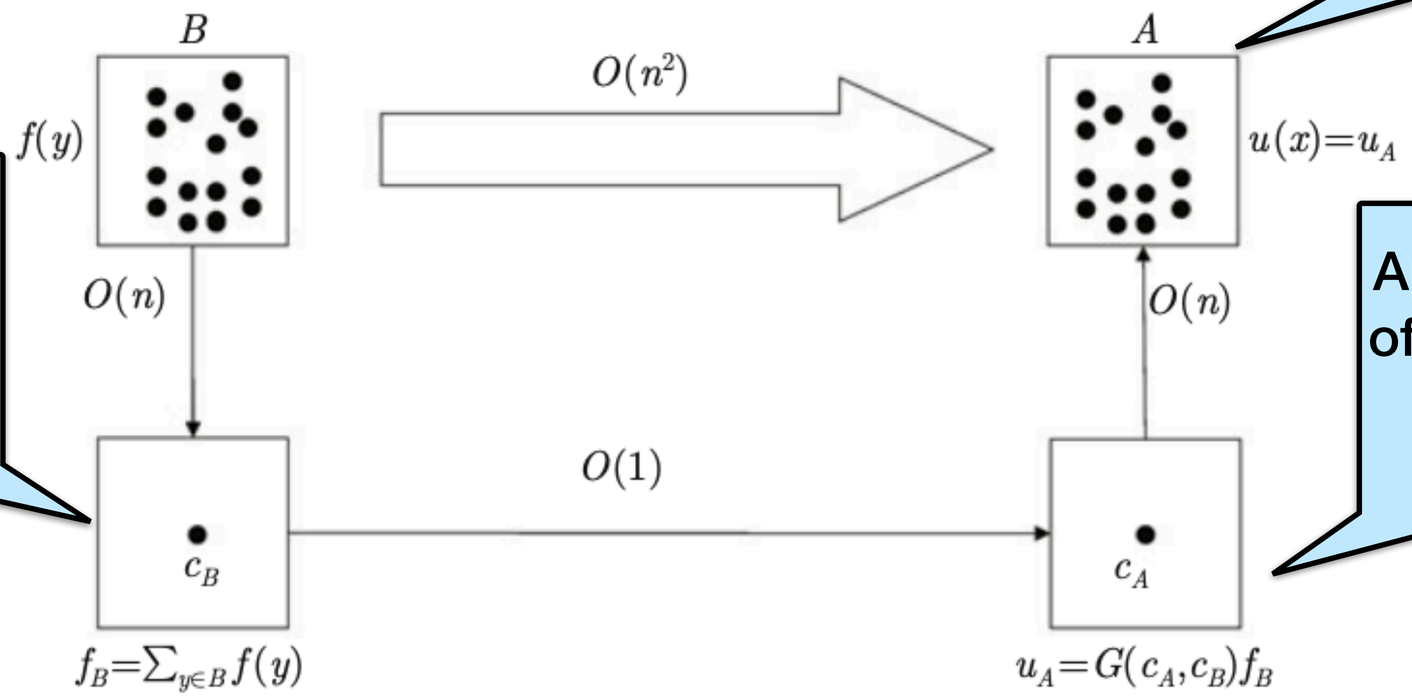
- Fast multipole method - Greengard and Rokhlin in 1987
  - For $G(x, y) = \ln|x - y|$ or $1/|x - y|$
  - Bring the complexity to O(N) for some fixed accuracy ϵ

- Based on four key ideas

Idea 1: Well-separated Regions

- Suppose boxes B and A are well-separated
- Consider the influence from B to A: at each x_i in A, evaluate

$$u_i = \sum_{x_j \in B} G(x_i, x_j) f_j$$

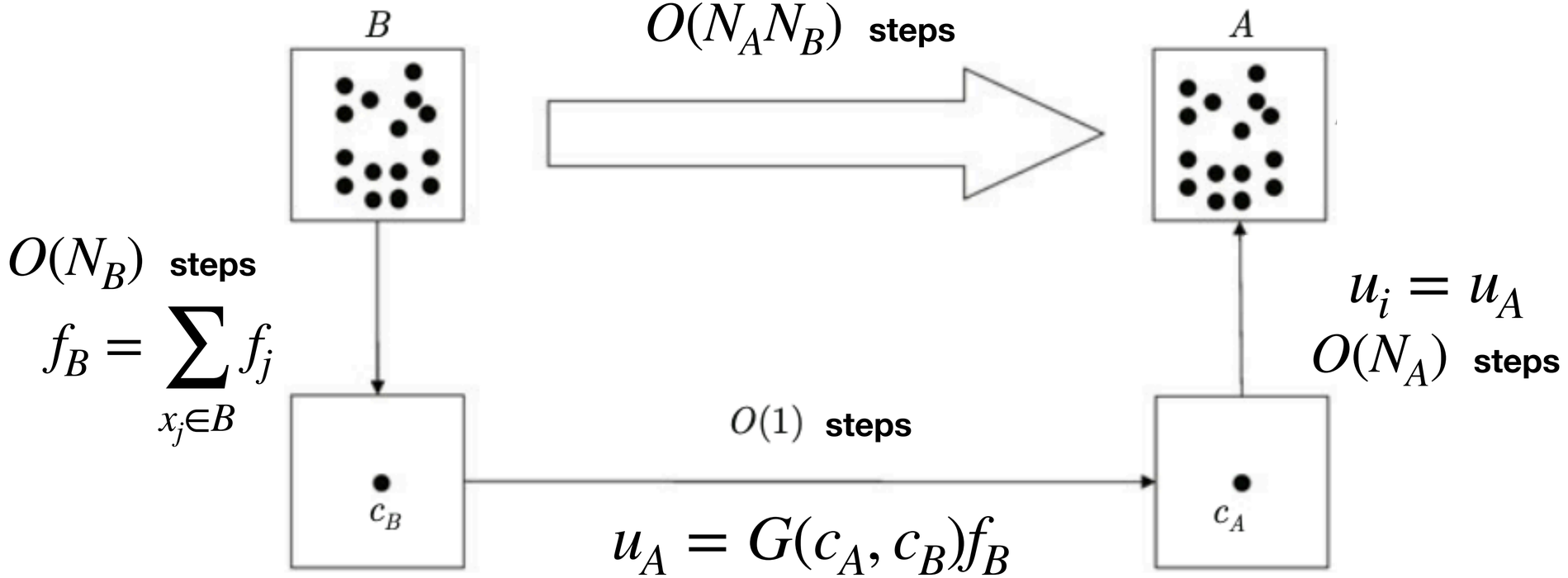


Consider boxes rather than individual points

Sum mass, get center, evaluate potential at center of both A and B

Approximate effect of B on all of A with one single multiplication

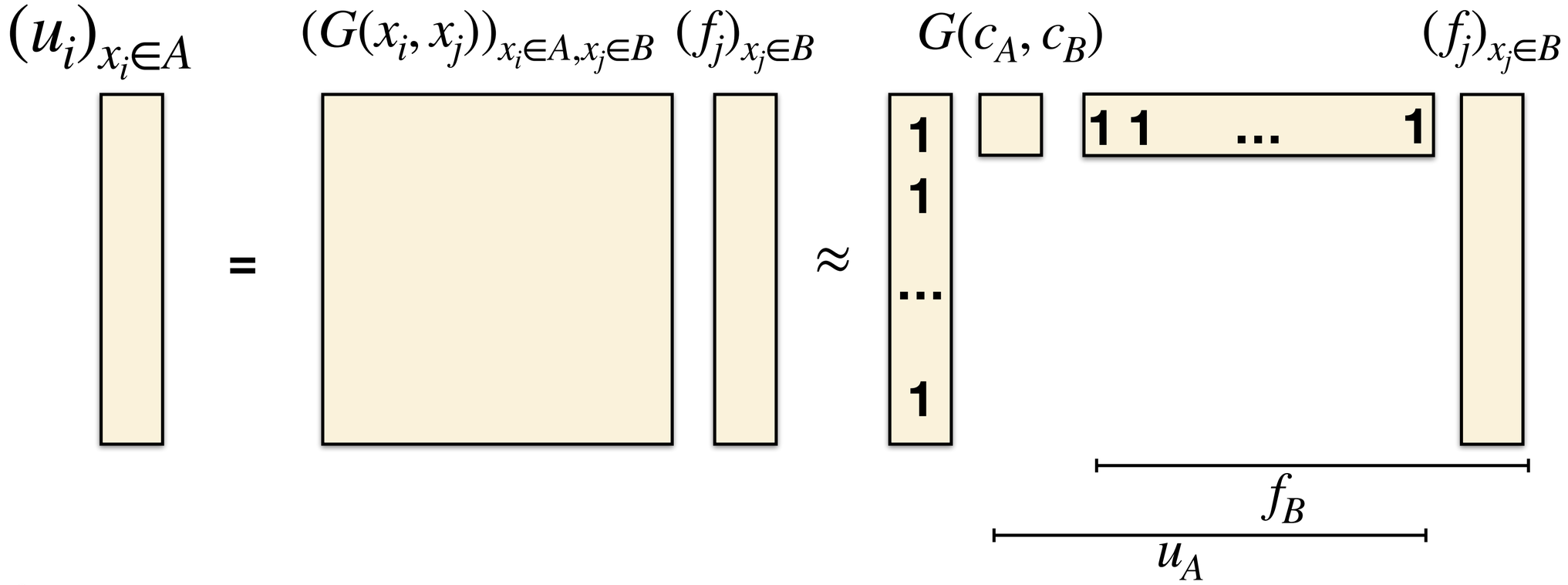
Cost of 3 step approximation



- Reduce $O(N_A N_B)$ steps to $O(N_A + N_B)$ steps
- Good accuracy when **A, B are far away**
- We will use it whenever A and B are well separated

Low-rank matrix approximation

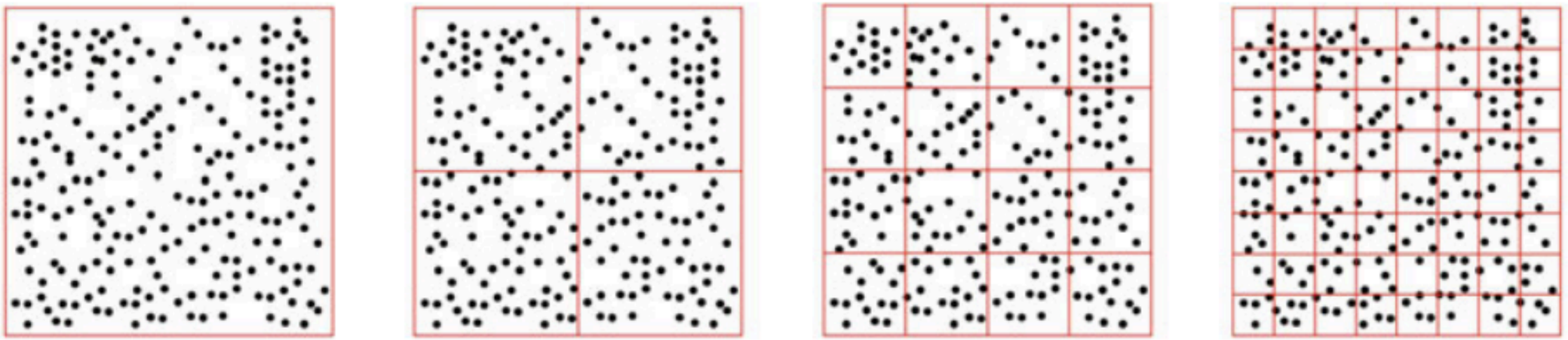
Algebraic Interpretation



- f_B = source representation
- u_A = target representation
- Problem: but each point x_i serves as **both a source and target**

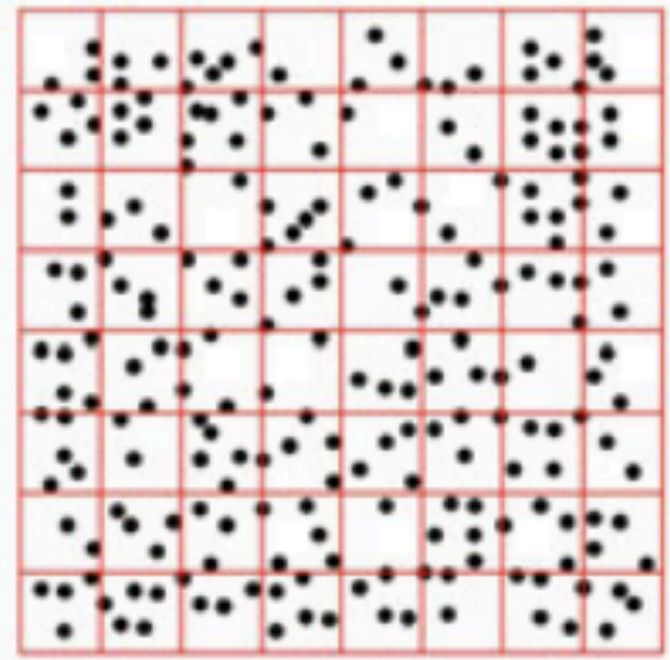
Idea 2: Hierarchical Decomposition

- Partition the domain hierarchically until **each leaf box contains O(1) number of points**

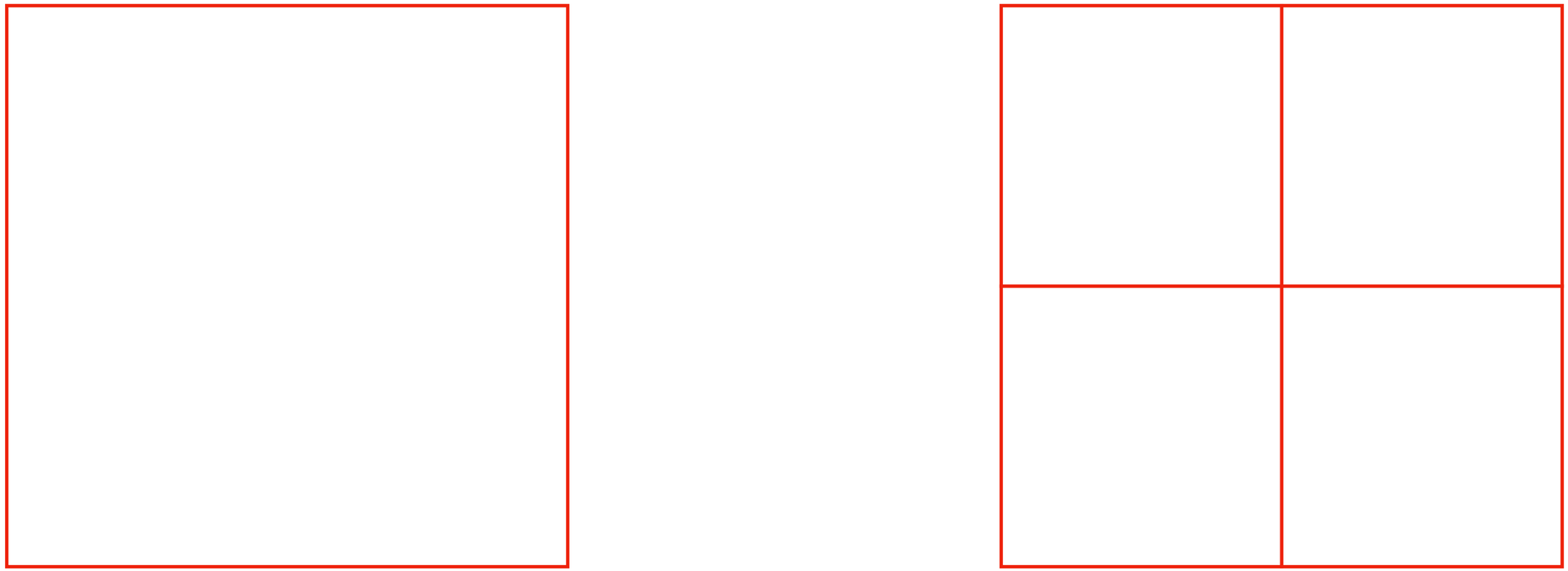


Idea 2: Hierarchical Decomposition

- Partition the domain hierarchically until **each leaf box contains O(1) number of points**
- N = # points
- # levels = $O(\log_4 N)$
- # boxes on level ℓ = $O(4^\ell)$
- Total # boxes = $O(N)$

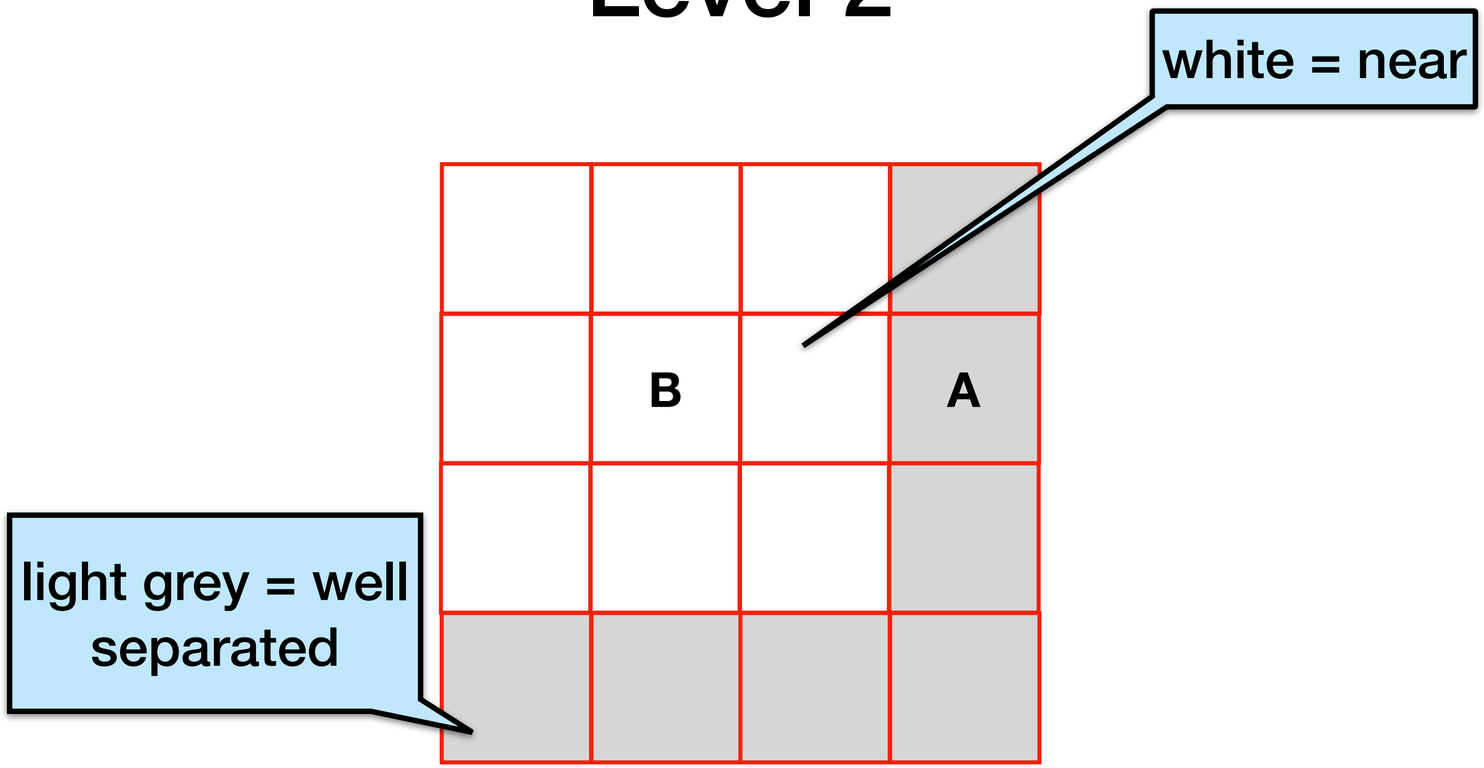


Levels 0 and 1



At levels 0 and 1, the boxes are adjacent to each other, so **we cannot use the 3 step approximation** as the boxes are not well separated

Level 2

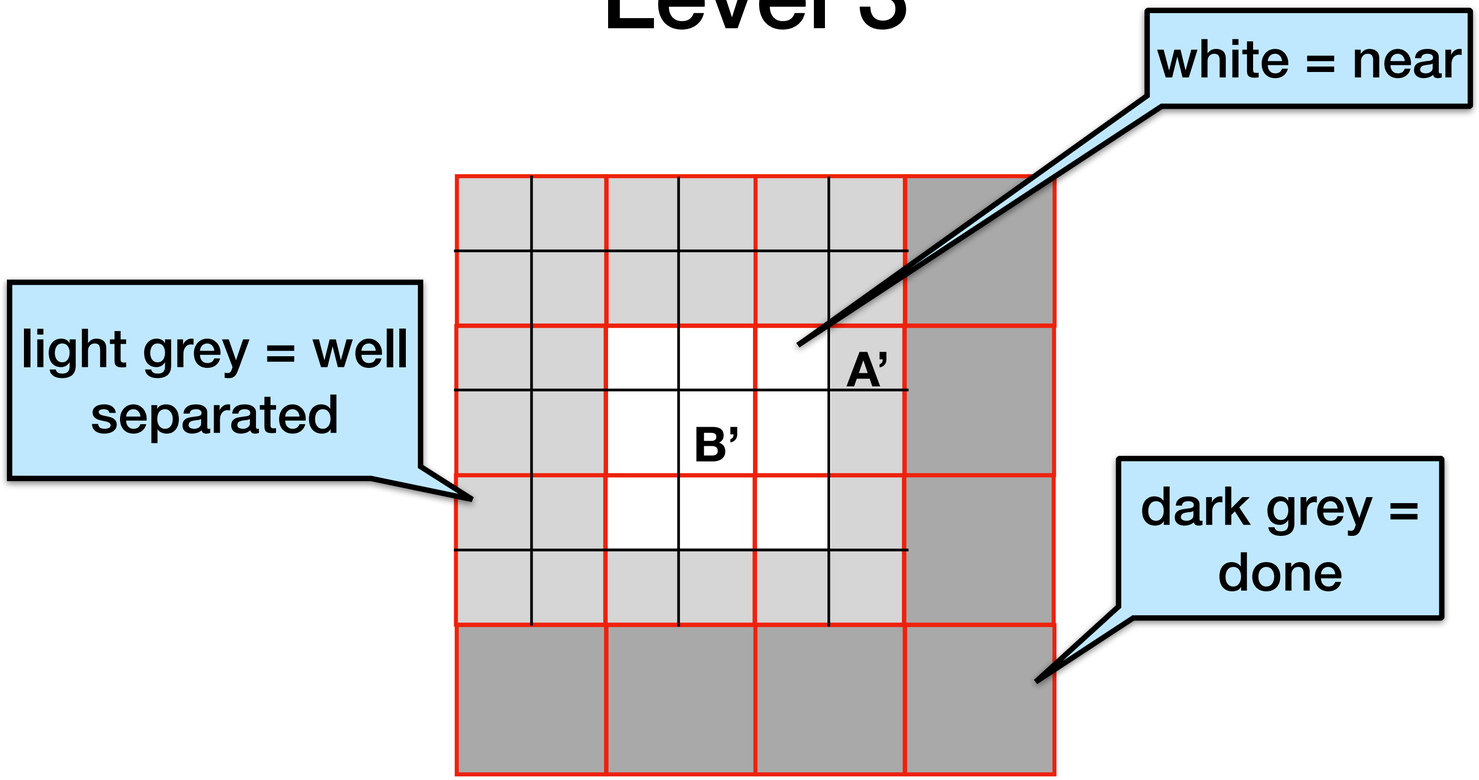


B and A are well separated, so use 3 step approximation for influence from B to A

Q: What about interaction with B its neighbors?

- **Go down one level in hierarchical decomposition**

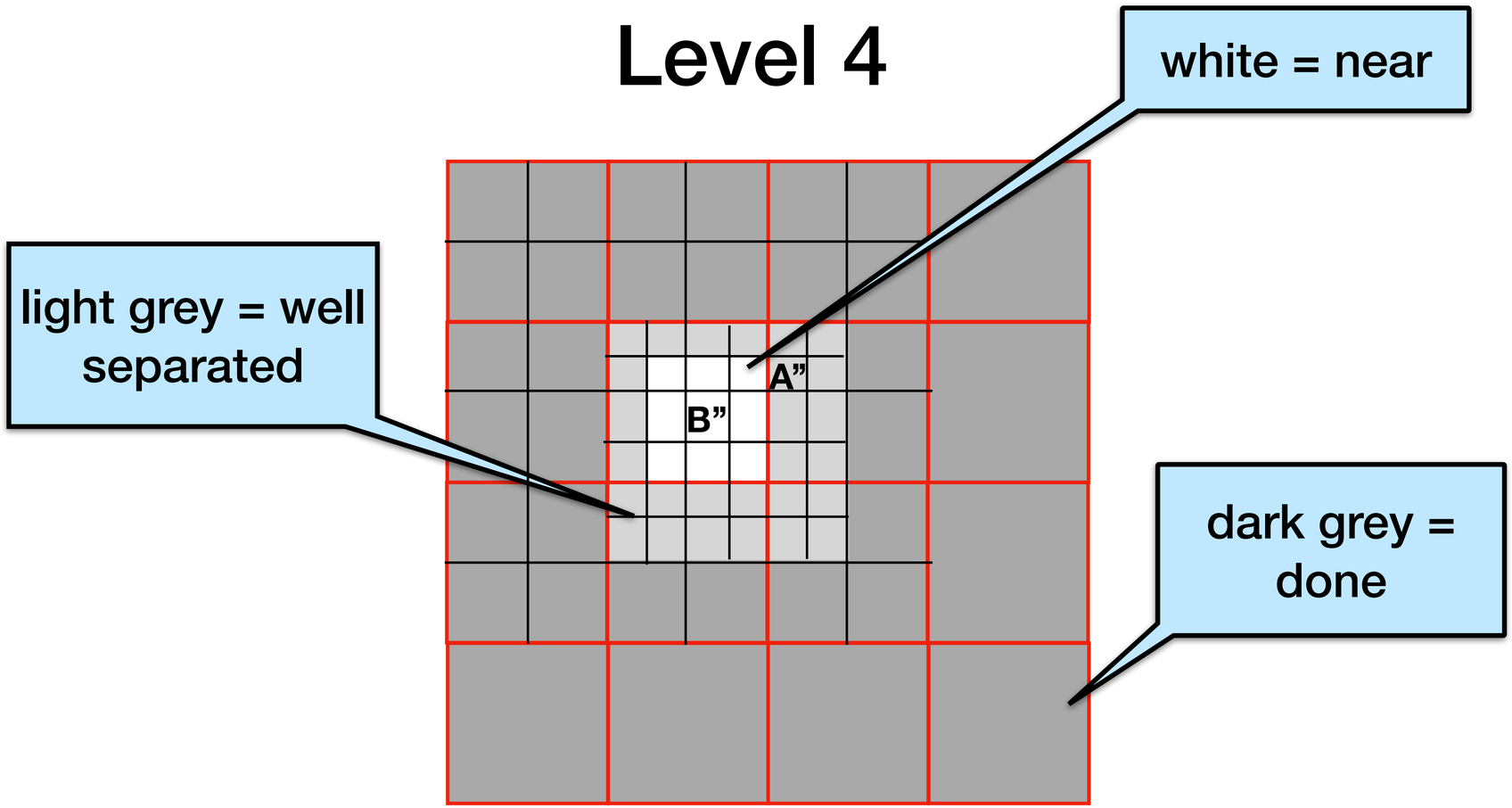
Level 3



B’ and A’ are well-separated

Use the 3-step approx. for the influence from B’ to A’

Interaction list of all B’ = all such A’, at most $6^2 - 3^2 = 27$ boxes



- B” and A” are well-separated, so use 3-step approximation for influence from B” to A”
- Q: influence from B” to neighbors?
  - Suppose we are at the leaf level, so we cannot keep going down
  - **Evaluate neighbor influence directly**

Algorithm 1

1. For each box B in the tree, compute $f_B = \sum_{x_j \in B} f_j$

2. For L = 2 to last level
   for each box B at level L
      for each A in B's interaction list

O(1) boxes

$$u_A = u_A + G(c_A, c_B) f_B$$

3. for each box A in the tree

$u_i = u_i + u_A$ for $x_i \in A$

4. For each box B on the last level

$$u_i = u_i + \sum_{x_j \in nghs(B)} G(x_i, x_j) f_j, \text{ for } x_i \in B$$

Step 1

1. For each box B in the tree, compute $f_B = \sum_{x_j \in B} f_j$

For a given level ℓ, there 4^ℓ boxes and $O(N/4^\ell)$ points in each of the boxes at that level (assuming uniformity)

So the work to calculate the sum of all the masses at each level is O(N)

for O(N log N) total work

Step 2

2. For L = 2 to last level
   for each box B at level L
      for each A in B's interaction list

$$u_A = u_A + G(c_A, c_B) f_B$$

For a given level ℓ, there 4^ℓ boxes and each has O(1) neighbors in the interaction list. So the work to do this approximation, assuming you have already calculated f_B, is $\sum_{\ell=0}^{\log_4 N} 4^\ell = O(N)$

Algorithm 1 Analysis

1. For each box B in the tree, compute $f_B = \sum_{x_j \in B} f_j$

$O(N \log N)$

2. For L = 2 to last level
   for each box B at level L
      for each A in B's interaction list

O(1) boxes

$O(N)$

$$u_A = u_A + G(c_A, c_B) f_B$$

3. for each box A in the tree

$$u_i = u_i + u_A \text{ for } x_i \in A$$

$O(N \log N)$

4. For each box B on the last level

$$u_i = u_i + \sum_{x_j \in nghs(B)} G(x_i, x_j) f_j, \text{ for } x_i \in B$$

$O(N)$

can we do better?

Total = $O(N \log N)$

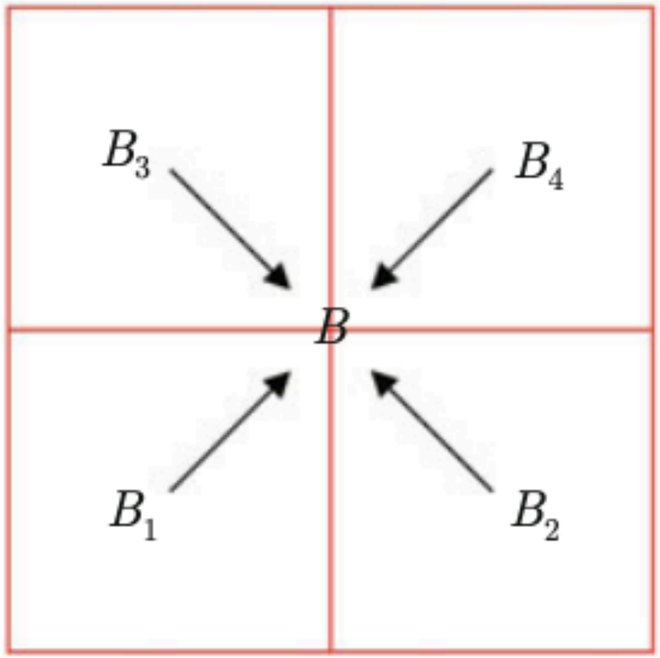
Idea 3: reuse the computation

$$f_B = \sum_{x_j \in B} f_j = \sum_{x_j \in B_1} f_j + \sum_{x_j \in B_2} f_j + \sum_{x_j \in B_3} f_j + \sum_{x_j \in B_4} f_j = f_{B_1} + f_{B_2} + f_{B_3} + f_{B_4}$$

Idea: Compute f_B from its children

O(1) cost

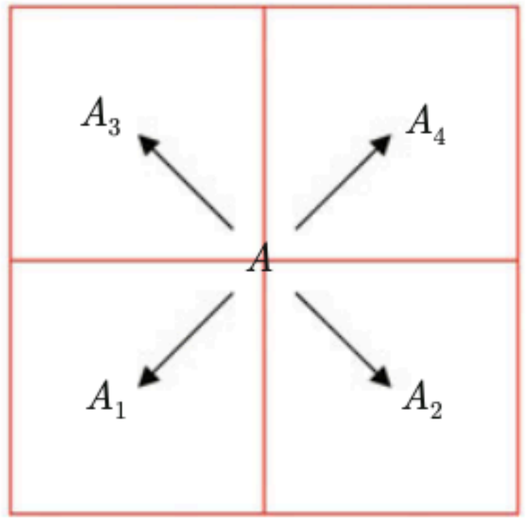
But you have to traverse the quadtree bottom-up



Idea 3: reuse the computation

$u_i = u_i + u_A$, for $x_i \in A$

- Similarly, add u_A to only its children - gradually pass down to the lower generations
  - O(1) cost
  - Traverse quadtree top down



Algorithm 2 (FMM)

1. Go up the tree, for each box B

   if B is a leaf box, $f_B = \sum_{x_j \in B} f_j$

   else, $f_B = f_{B_1} + f_{B_2} + f_{B_3} + f_{B_4}$

2. Same as step 2 of algorithm 1

3. Go down the tree, for each box A

   if A is a leaf box, $u_i = u_i + u_A$, for $x_i \in A$

   else, $u_{A_1} = u_{A_1} + u_A$, same for A_2, A_3, A_4

4. Same as step 4 of algorithm 1

Algorithm 2 (FMM)

1. Go up the tree, for each box B

   if B is a leaf box, $f_B = \sum_{x_j \in B} f_j$

   else, $f_B = f_{B_1} + f_{B_2} + f_{B_3} + f_{B_4}$

   $O(N)$

2. Same as step 2 of algorithm 1

   $O(N)$

3. Go down the tree, for each box A

   if A is a leaf box, $u_i = u_i + u_A$, for $x_i \in A$

   else, $u_{A_1} = u_{A_1} + u_A$, same for A_2, A_3, A_4

   $O(N)$

4. Same as step 4 of algorithm 1

   $O(N)$

Total = $O(N)$

Three Main Ideas So Far

- When things are far away, use 3-step procedure (low-rank approximation)
- Use hierarchical decomposition to identify what is far away and what is nearby
- Reuse computation to do calculation of the total mass

what about accuracy? not just the total mass, but the mass distribution

Idea 4: Better Source/Target Representations

So far

- f_B is the sum of charges in B
- u_A is the potential at c_A from well-separated sources
- Low accuracy if A and B are not far

More generally

- f_B is a **compact representation** of the charge for B in well-separated targets
- Similarly, u_A is a **compact representation** of the potential in A by well-separated targets

- Better accuracy requires better compact reps.
  - In 2D, treat x, y as points in the complex plane
  - G(x, y) = RE(ln(x-y))

More Fine-grained Representations

- In practice, to get good accuracy, we can use low-rank approximations rather than rank-1 approximations.
- Instead of representing f_B, u_A as a single value, represent them as a **series of coefficients rather than a single term** - first one is the initial approximation
- Keep $p = \lg(1/\epsilon)$ for accuracy ϵ
- Calculating the total mass now goes to $O(pn_B + pn_A)$. Still linear if p = O(1)

Summary of FMM

- The FMM relies on 4 main ideas:
  - Low-rank interaction
  - Hierarchical decomposition
  - Reuse of computation
  - More accurate compact representation for f_B and u_A
- So far we assumed the points are uniform, if they are not, adaptive algorithm is also available

Parallelizing BH, FMM, and related algorithms

Parallelizing Hierarchical N-Body Codes

Barnes-Hut, FMM and related algorithms have similar computational structure:

1) Build the QuadTree
2) Traverse QuadTree from leaves to root and build outer expansions

(just (TM,CM) for Barnes-Hut)
3) Traverse QuadTree from root to leaves and build any inner expansions (FMM only)

4) Traverse QuadTree to accumulate forces for each particle

Parallelizing Hierarchical N-Body Codes

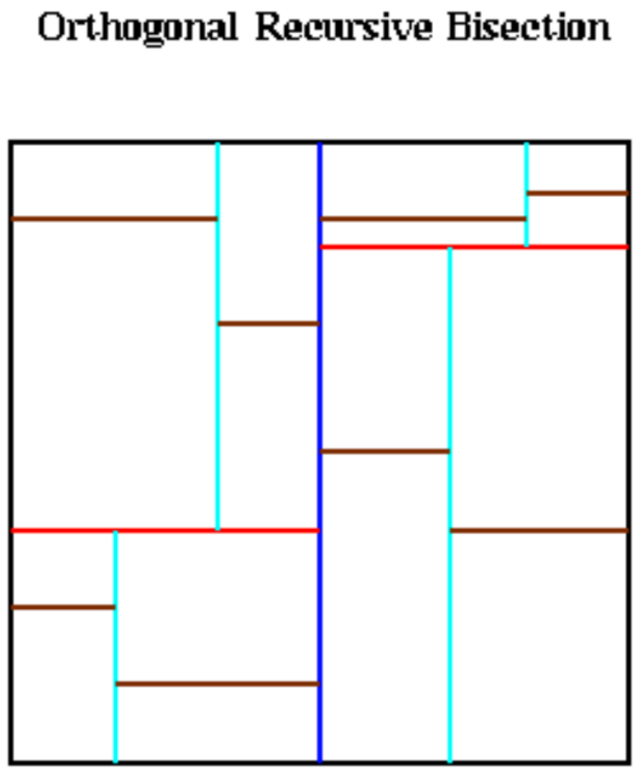
One parallelization scheme will work for them all [Blackston and Suel, SC '97]

- Assign regions of space to each processor
- Regions may have different shapes, to get load balance
  - Each region will have about N/p particles
- Each processor will store part of Quadtree containing all particles (=leaves) in its region, and their ancestors in Quadtree
  - Top of tree stored by all processors, lower nodes may also be shared
- Each processor will also store adjoining parts of Quadtree needed to compute forces for particles it owns
  - Subset of Quadtree needed by a processor called the **Locally Essential Tree (LET)**
- Given the LET, all force accumulations (step 4)) are done in parallel, without communication

Load Balancing Scheme 1: Orthogonal Recursive Bisection (ORB) [Warren and Salmon, SC '92]

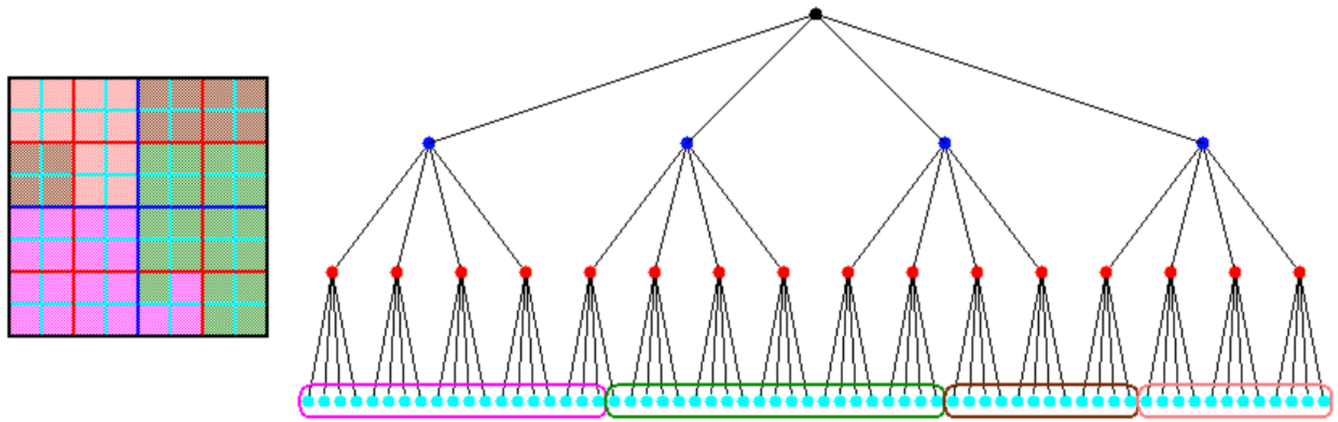
- Recursively split region along axes into regions containing equal numbers of particles

Partitioning for 16 procs:



Load Balancing Scheme 2: Costzones

- Called Costzones for Shared Memory
  - PhD thesis, J.P. Singh, Stanford, 1993
- Called “Hashed Oct Tree” for Distributed Memory
  - Warren and Salmon, Supercomputing 93
- We will use the name Costzones for both; also in Pbody
- Idea: partition QuadTree instead of space
  - Estimate work for each node, call total work W



Example Performance Results

- **512 Proc Intel Delta**
  - **Warren and Salmon, Supercomputing 92, Gordon Bell Prize**
  - **8.8 M particles, uniformly distributed**
  - **.1% to 1% RMS error, Barnes-Hut**
  - **114 seconds = 5.8 Gflops**
    - **Decomposing domain 7 secs**
    - **Building the OctTree 7 secs**
    - **Tree Traversal 33 secs**
    - **Communication during traversal 6 secs**
    - **Force evaluation 54 secs**
    - **Load imbalance 7 secs**
  - **Rises to 160 secs as distribution becomes nonuniform**