# CS601: Software Development for Scientific Computing

Autumn 2023

Week15 : Particle Methods (N-Body Problems), Misc Topics

# Particle (Simulation) Methods

- N-Body Simulation – Problem

System of N-bodies (e.g. galaxies, stars, atoms, light rays etc.) interacting with each other continuously

- Problem:
  - Compute force acting on a body due to all other bodies in the system
  - Determine position, velocity, at various times for each body
- Objective:
  - Determine the (approximate) evolution of a system of bodies interacting with each other simultaneously

# Particle (Simulation) Methods

- N-Body Simulation - Examples
  - Astrophysical simulation: E.g. each body is a star/galaxy https://commons.wikimedia.org/w/index.php?title=File%3AGalaxy_collision.ogv
  - Graphics: E.g. each body is a ray of light emanating from the light source. https://www.fxguide.com/fxfeatured/brave-new-hair/
    - Here each body is a point on a strand of hair

# N-Body Simulation

- All-pairs Method
  - Naïve approach. Compute *all pair-wise interactions*
- Hierarchical Methods
  - Optimize. Reduce the number of pair-wise force calculations. How? dependence on ‘distant’ particle(s) can be *compressed*
  - Examples:
    - Barnes-Hut
    - Fast Multipole Method

# N-Body Simulation

- Three fundamental simulation approaches
  - Particle-Particle (PP)
  - Particle-Mesh (PM)
  - Particle-Particle-Particle-Mesh (P3M)
- Hybrid approaches
  - Nested Grid Particle Scheme
  - Tree Codes
  - Tree Code Particle Mesh (TPM)
- Self Consistent Field (SCF), Smoothed-Particle Hydrodynamics (SPH), Symplectic etc.

# Particle-Particle method

- Simplest. Adopts an all-pairs approach.
- State of the system at time $t$ given by particle positions $x_i(t)$ and velocity $v_i(t)$ for $i=1$ to $N$

$$\{x_i(t), v_i(t); i = 1, N\}$$

  – Steps:

1. Compute forces
2. Integrate equations of motion
3. Update time counter
   Each iteration updates $x_i(t)$ and $v_i(t)$ to compute $x_i(t + \Delta t)$ and $v_i(t + \Delta t)$

# Particle-Particle Method

1. Compute forces

```
//initialize forces
for i=1 to N
  F_i = 0

//Accumulate forces
for i=1 to N-1
  for j=i+1 to N
      F_i = F_i + F_ij
      F_j = F_j - F_ij
```

$F_{ij}$ is the force on particle $i$ due to particle $j$

Typically: $F_i = F_{external} + F_{nearest\_neighbor} + F_{N\text{-}Body}$

# Particle-Particle Method

## 2. Integrate equations of motion

for i=1 to N

$$v_i^{new} = v_i^{old} + \frac{F_i}{m_i}\,\Delta t$$ //using a=F/m and v=u+at

$$x_i^{new} = x_i^{old} + v_i\,\Delta t$$

## 3. Update time counter

$$t^{new} = t^{old} + \Delta t$$

# Particle-Particle Method

```
t=0
while(t<t^final) {
//initialize forces
      for i=1 to N
        F_i = 0
//Accumulate forces
      for i=1 to N-1
        for j=i+1 to N
          F[i] = F[i] + F_ij
          F[j] = F[j] - F_ij
//Integrate equations of motion
      for i=1 to N
```

$$v_i^{new} = v_i^{old} + \frac{F_i}{m_i}\,\Delta t$$ //using a=F/m and v=u+at

$$x_i^{new} = x_i^{old} + v_i\,\Delta t$$

// Update time counter

t = t + $\Delta t$

}

# Particle-Particle Method

- Costs (CPU operations)?

```
t=0
while(t<tfinal) {
//initialize forces
        for i=1 to N
          Fi = 0
//Accumulate forces
        for i=1 to N-1
          for j=i+1 to N
            F[i] = F[i] + Fij
            F[j] = F[j] - Fij
//Integrate equations of motion
        for i=1 to N
```

$v_i^{new} = v_i^{old} + \frac{F_i}{m_i}\Delta t$ //using a=F/m and v=u+at

$x_i^{new} = x_i^{old} + v_i\,\Delta t$

// Update time counter

t = t + $\Delta t$

}

# Particle-Particle Method

- Experimental results (then):
  - 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*

# Particle-Particle Method

- Experimental results (now):

  Vortex particle simulation of turbulence

  - 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

# Particle-Particle (PP) Method

- Discussion
  - Simple/trivial to program
  - High computational cost
    - Useful when number of particles are small (few thousands) and
    - We are interested in close-range dynamics when the particles in the range contribute *significantly* to forces
    - Constant time step must be replaced with variable time steps and numerical integration schemes for close-range interactions

# N-Body Simulation

- All-pairs Method
  - Naïve approach. Compute *all pair-wise interactions*
- Hierarchical Methods
  - Optimize. Reduce the number of pair-wise force calculations. How? dependence on ‘distant’ particle(s) can be *compressed*
  - Examples:
    - Barnes-Hut
    - Fast Multipole Method

# Tree Codes

$$F_i = F_{external} + F_{nearest\_neighbor} + F_{N\text{-}Body}$$

- $F_{external}$ can be computed for each body independently. O(N)
- $F_{nearest\_neighbor}$ involve computations corresponding to few nearest neighbors. O(N)
- $F_{N\text{-}Body}$ require all-to-all computations. Most expensive. $O(N^2)$ if computed using all-pairs approach.

```
for(i = 1 to N)
```

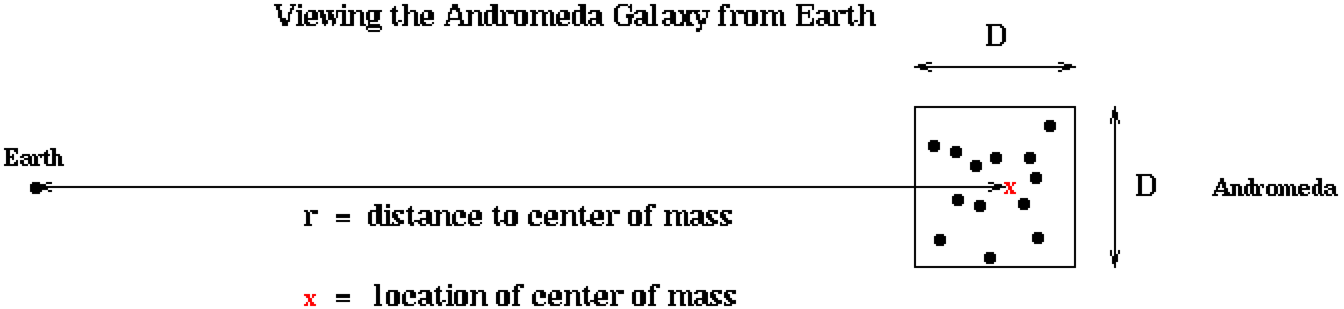
$F_i = \sum_{i \neq j} F_{ij}$ $F_{ij}$= force on i from j

$F_{ij}$ =c*v/||v||$^3$ in 3D, $F_{ij}$ = c*v/||v||$^2$ in 2D
v = vector from particle i to particle j , ||v|| = length of v, c = product of masses or charges

*We can do better.*

# Tree Codes: Divide-Conquer Approach

- Consider computing force on earth due to all celestial bodies
  - Look at the night sky. Number of terms in $\sum_{i \neq j} F_{ij}$ is greater than the number of visible stars
  - One "star" could really be the Andromeda galaxy, which contains billions of real stars. *Seems like a lot more work than we thought …*
  - Idea: Ok to approximate all stars in Andromeda by a single point at its center of mass (CM) with same total mass (TM)




  - Require that `D/r` be "small enough" (`D` = size of box containing Andromeda , r = distance of CM to Earth).

Idea is not new. Newton approximated earth and falling apple by CM

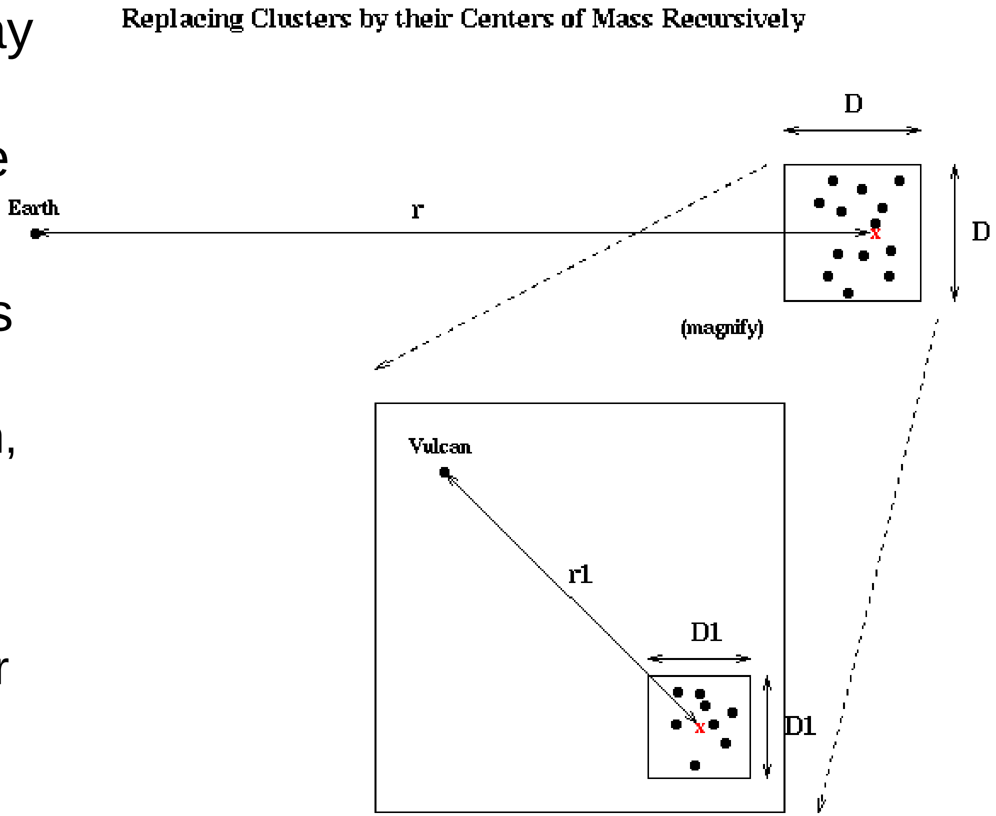
Slide contents based on: CS267 Lecture 24, https://sites.google.com/lbl.gov/cs267-spr2019/

# Tree Codes: Divide-Conquer Approach

- New idea: recursively divide the box.

– If you are in Andromeda, Milky Way (the galaxy we are part of) could appear like a white dot. So, can be approximated by 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
  - If you are on Vulcan, another solar system in Andromeda can be a white dot.
  - Boxes nest in boxes recursively






Slide contents based on: CS267 Lecture 24, https://sites.google.com/lbl.gov/cs267-spr2019/

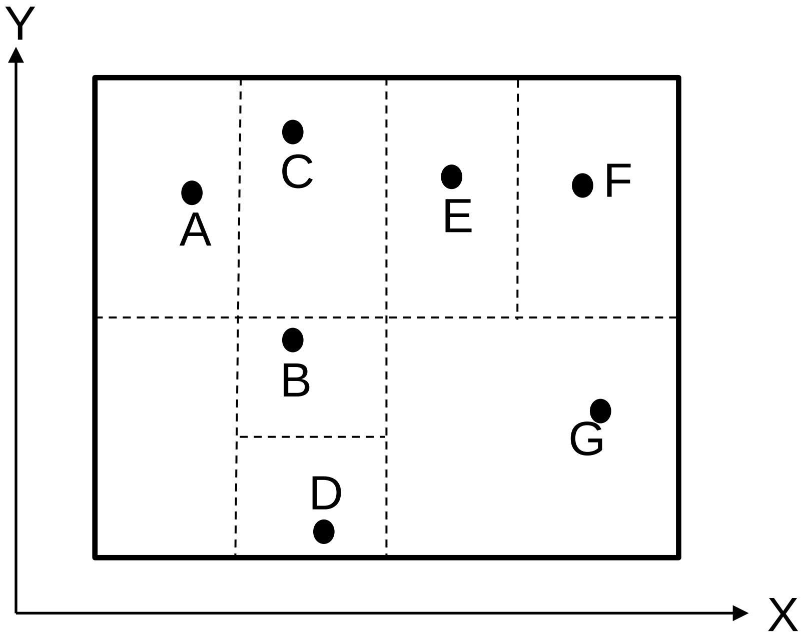
# Tree Codes: Divide-Conquer Approach

- Data structures needed:
  - Quad-trees
  - Octrees

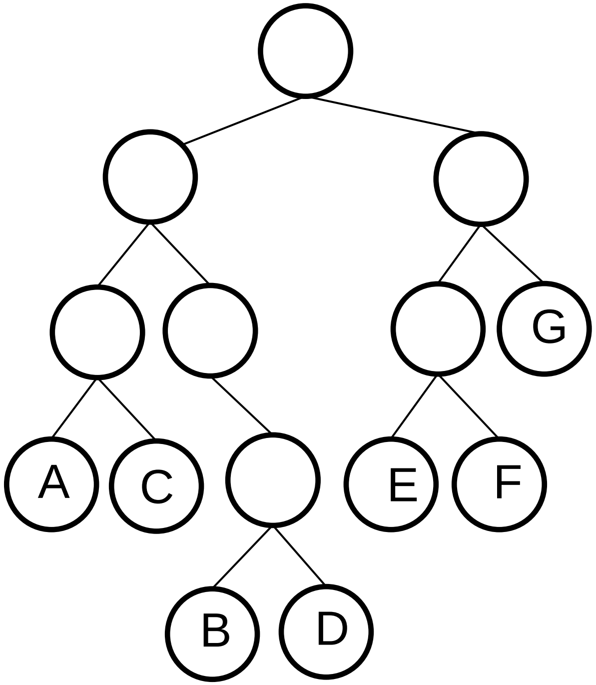
# Background – metric trees

e.g. K-dimensional (kd-), Vantage Point (vp-), quad-trees, octrees, ball-trees

2-dimensional space of points




Binary kd-tree, 1 point /leaf cell



# Background - metric trees

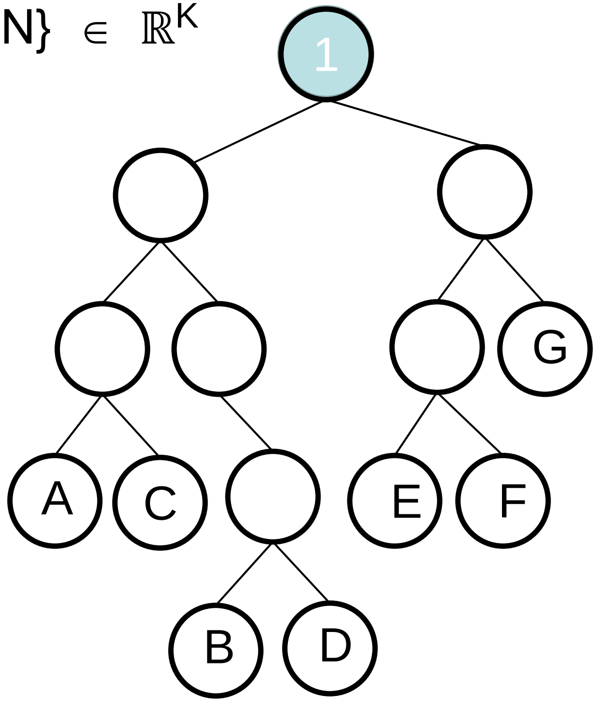
*Typical use:* traverse the tree (often repeatedly), truncate the traversal at some intermediate node if <u>a domain-specific criteria</u> is not met.

E.g. Does the distance from CM to me < D/r?

Input points = {1, 2, … , N} $\in \mathbb{R}^K$

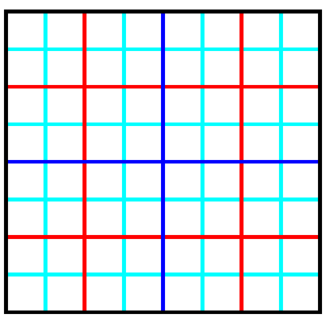
*Cost ???*

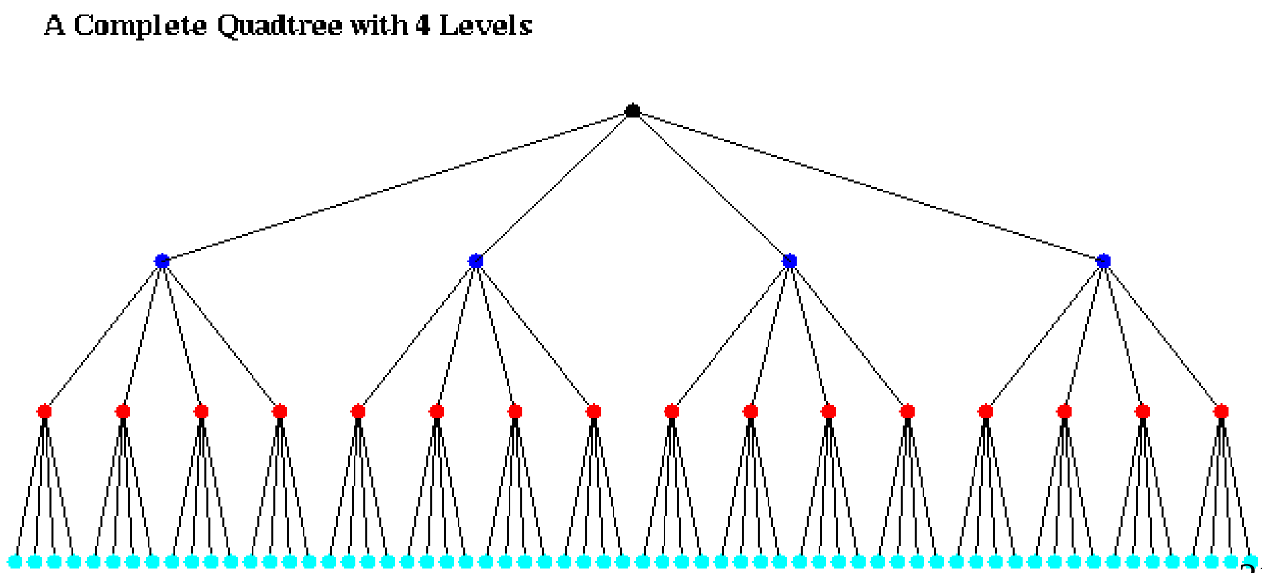
Kd-tree



# Quad Tree

- 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 non-leaf node has 4 children





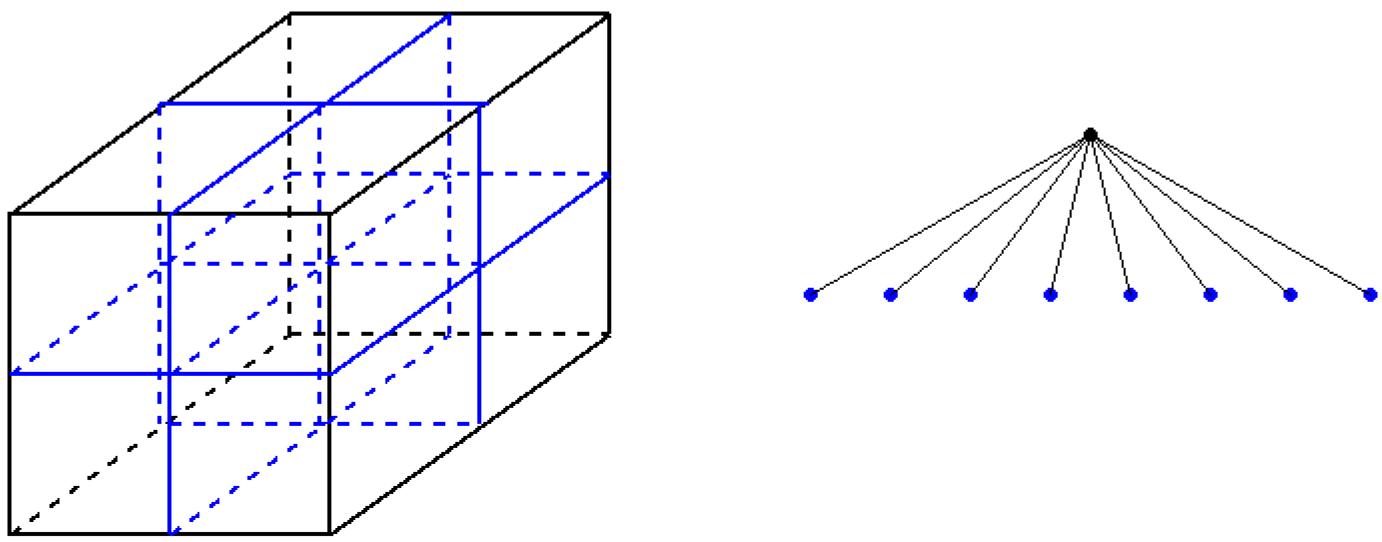



Slide courtesy: CS267 Lecture 24, https://sites.google.com/lbl.gov/cs267-spr2019/

# Octree or Oct Tree

- Similar data structure for subdividing 3D space

2 Levels of an Octree





Slide courtesy: CS267 Lecture 24, https://sites.google.com/lbl.gov/cs267-spr2019/

# Using Quad Tree and Octree

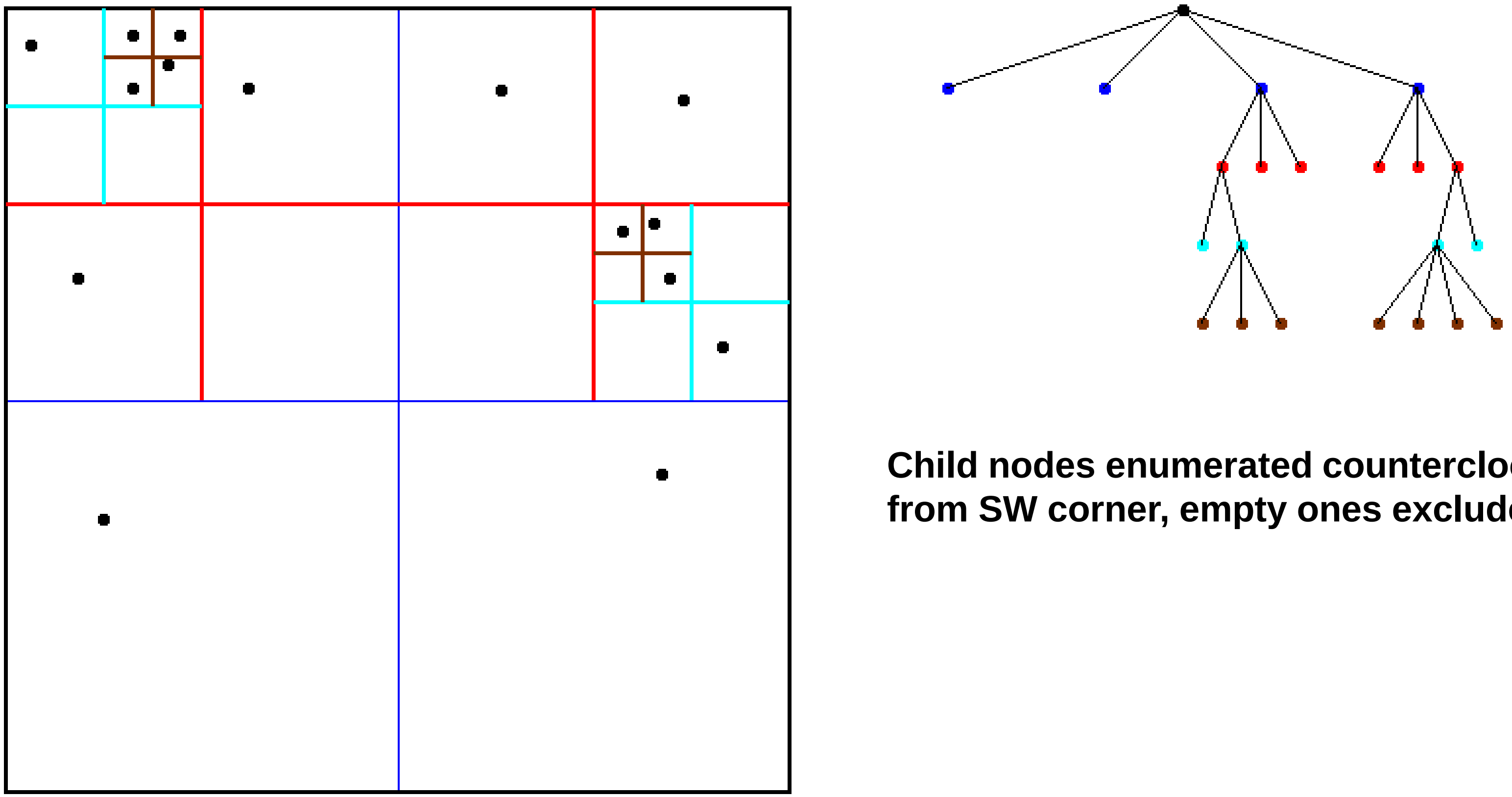
- Begin by constructing a tree to hold all the particles
  - Interesting cases have nonuniformly distributed particles
  - In a complete tree most nodes would be empty, a waste of space and time
  - Adaptive Quad (Oct) Tree only subdivides space where particles are located
- For each particle, traverse the tree to compute force on it



Slide contents based on: CS267 Lecture 24, https://sites.google.com/lbl.gov/cs267-spr2019/

# Using Quad Tree and Octree

Adaptive quadtree where no square contains more than 1 particle



Child nodes enumerated counterclockwise from SW corner, empty ones excluded

- In practice, have q>1 particles/square; tuning parameter (code to build data structure on hidden slide)



Slide courtesy: CS267 Lecture 24, https://sites.google.com/lbl.gov/cs267-spr2019/