# The n-body problem (3/3)

Prof. Richard Vuduc

Georgia Institute of Technology

CSE/CS 8803 PNA: Parallel Numerical Algorithms

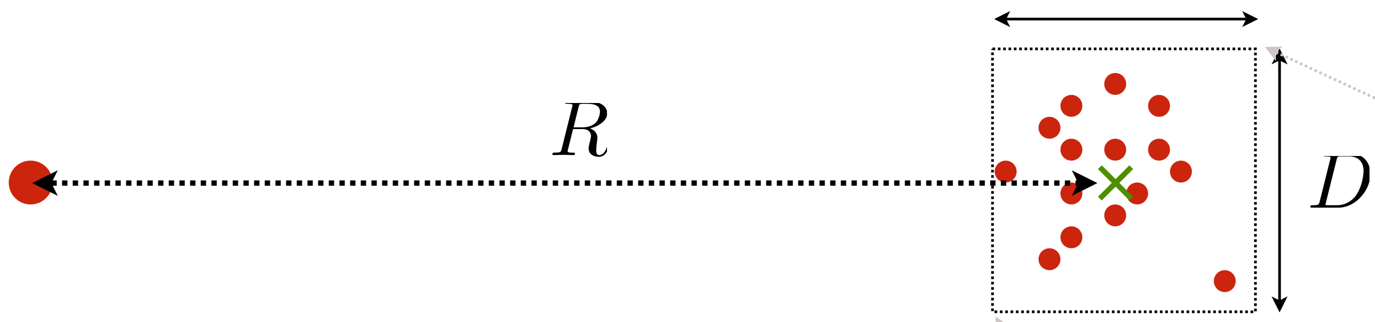
[L.24] Thursday, April 10, 2008

# Today's sources

- CS 267 at UCB (Demmel & Yelick)

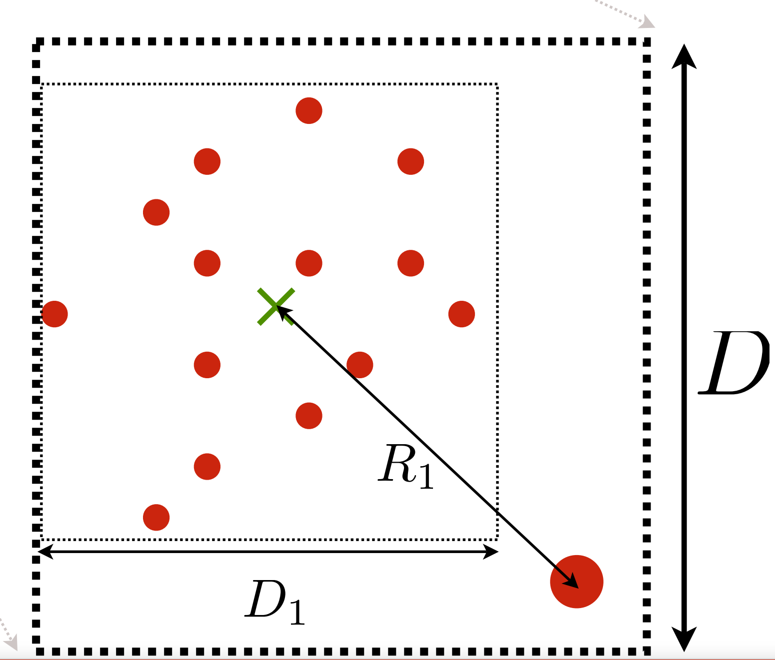
Review:
Tree codes

# Approximate long-distance interactions



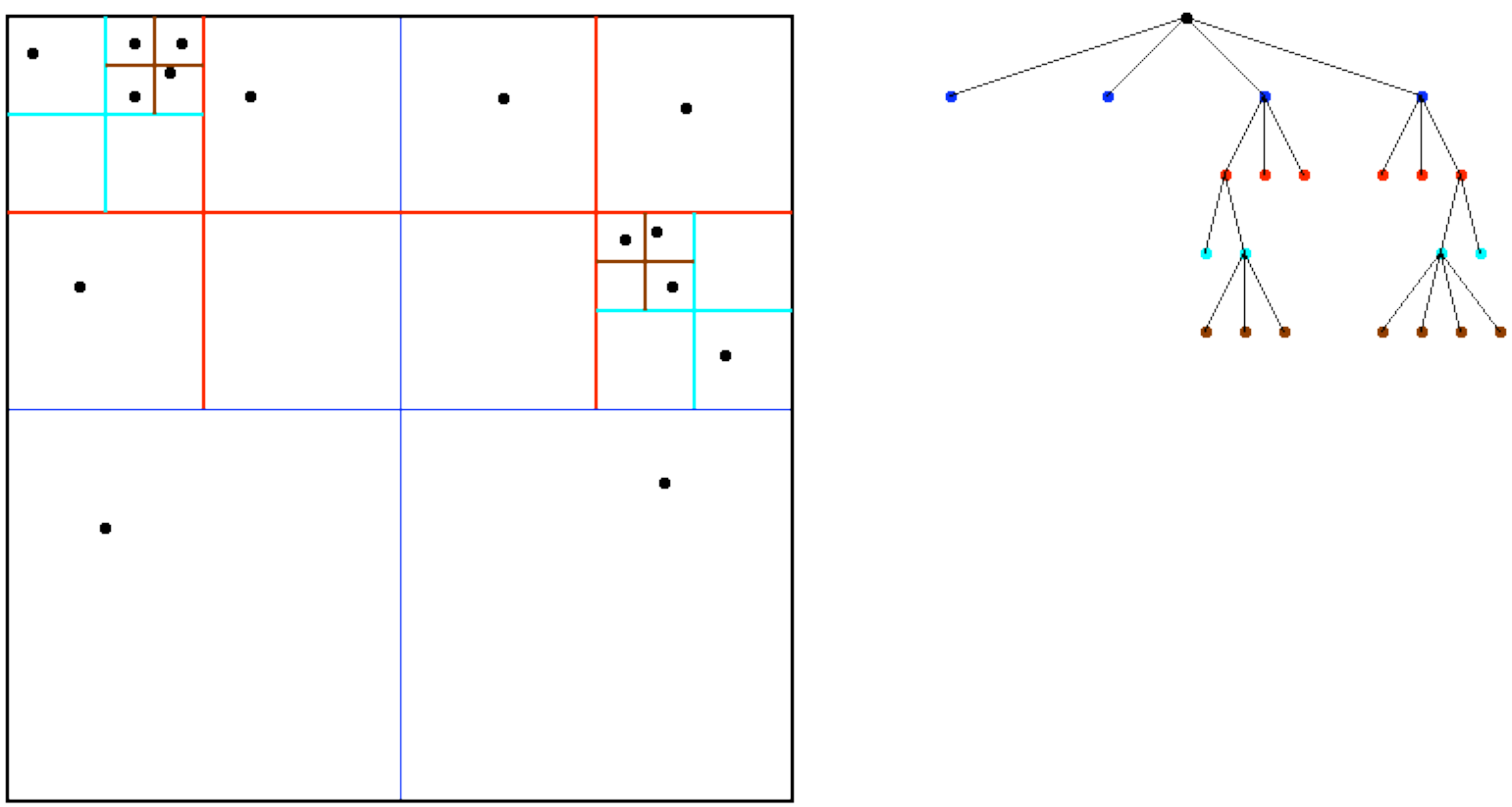

$$\frac{D}{R} < \theta$$

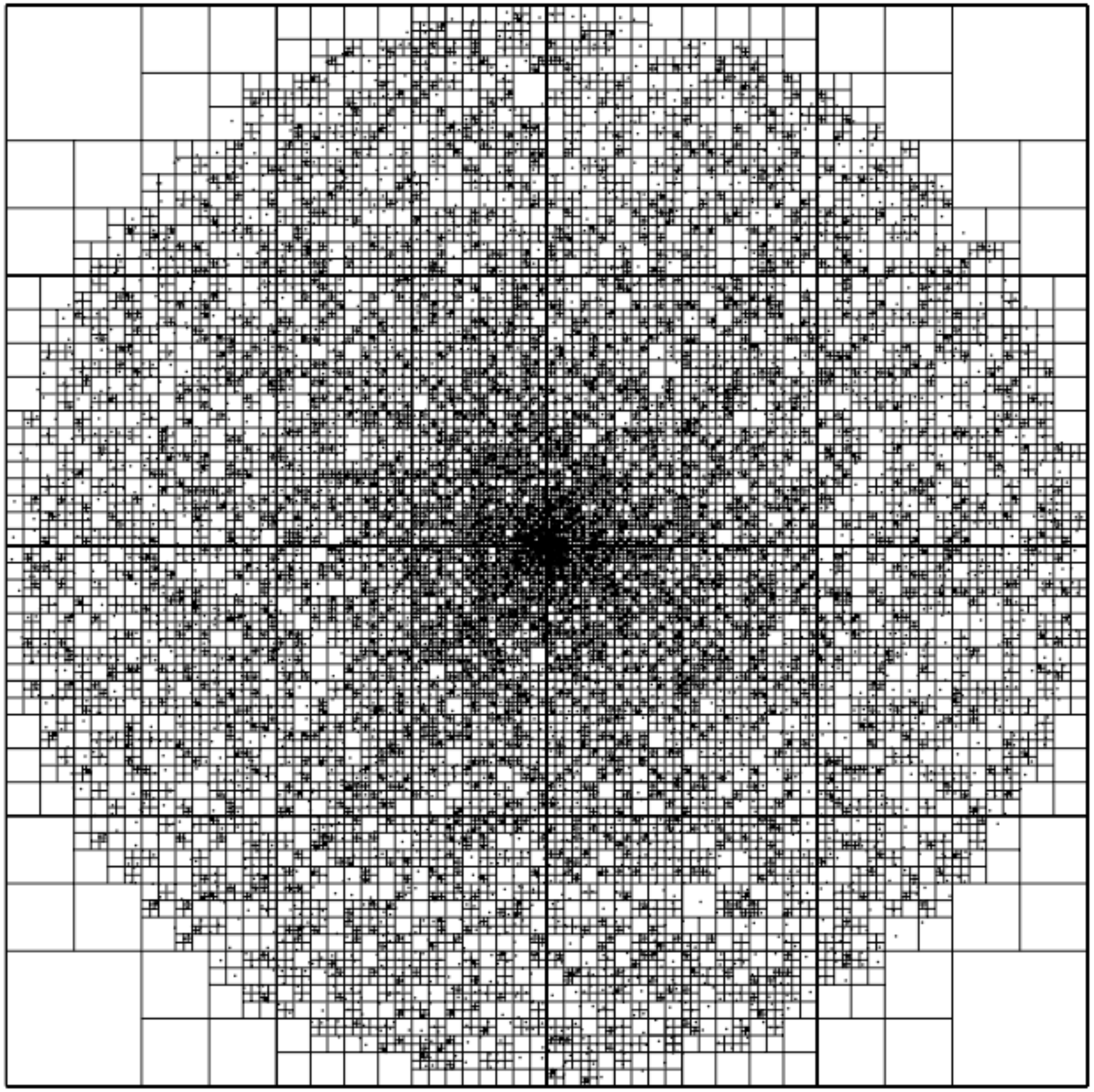
× = **center of mass**



# Idea: Organize particles into a tree

Adaptive quadtree where no square contains more than 1 particle





*Source: M. Warren & J. Salmon, In* Supercomputing 1993.

# Barnes-Hut algorithm (1986)

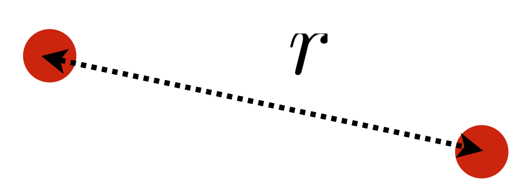
- Algorithm:
  - **Build tree**
  - For each **node**, compute center-of-mass and total mass
  - For each **particle**, traverse tree to compute force on it
    - If D/R < $\theta$, approximate with center-of-mass
    - Else, recurse on node and sum child results

# Fast multipole method of Greengard & Rokhlin (1987)

- Differences from Barnes-Hut
  - Computes potential, not force
  - Uses more than center-of- and total-mass ⇒ more accurate & expensive
  - Accesses fixed set of boxes at every level, independent of “D / R”
- Increasing accuracy
  - BH: Fixed info / box, more boxes
  - FMM: Fixed no. of boxes; more info / box

# FMM computes compact expression for potential

$$
\begin{aligned}
|\mathbf{F}(\mathbf{r})| &= \frac{1}{r^2} \\
&\Downarrow \\
\mathbf{F}(\mathbf{r}) &= -\nabla\phi(\mathbf{r})
\end{aligned}
$$



# Potential in 3-D

**3-D:**

$$\phi(\mathbf{r}) = -\frac{1}{|\mathbf{r}|} = -\frac{1}{\sqrt{x^2 + y^2 + z^2}}$$

$$\mathbf{F}(\mathbf{r}) = -\left(\frac{\partial \phi}{\partial x}, \frac{\partial \phi}{\partial y}, \frac{\partial \phi}{\partial z}\right) = -\left(\frac{x}{r^3}, \frac{y}{r^3}, \frac{z}{r^3}\right)$$

# 2-D multipole expansion

$$
\begin{aligned}
\alpha_d &\equiv \sum_{k=1}^{n} m_k z_k^d \\
\sum_{k=1}^{n} m_k \ln(z - z_k) &= M \ln z + \sum_{d=1}^{\infty} \frac{\alpha_d}{z^d} \\
&\approx M \ln z + \sum_{d=1}^{p} \frac{\alpha_d}{z^d} + \mathrm{Error}(p)
\end{aligned}
$$

**Can approx. by truncation**

$$
\mathrm{Error}(p) \sim \left( \frac{\max |z_k|}{|z|} \right)^{p+1}
$$

# FMM algorithm

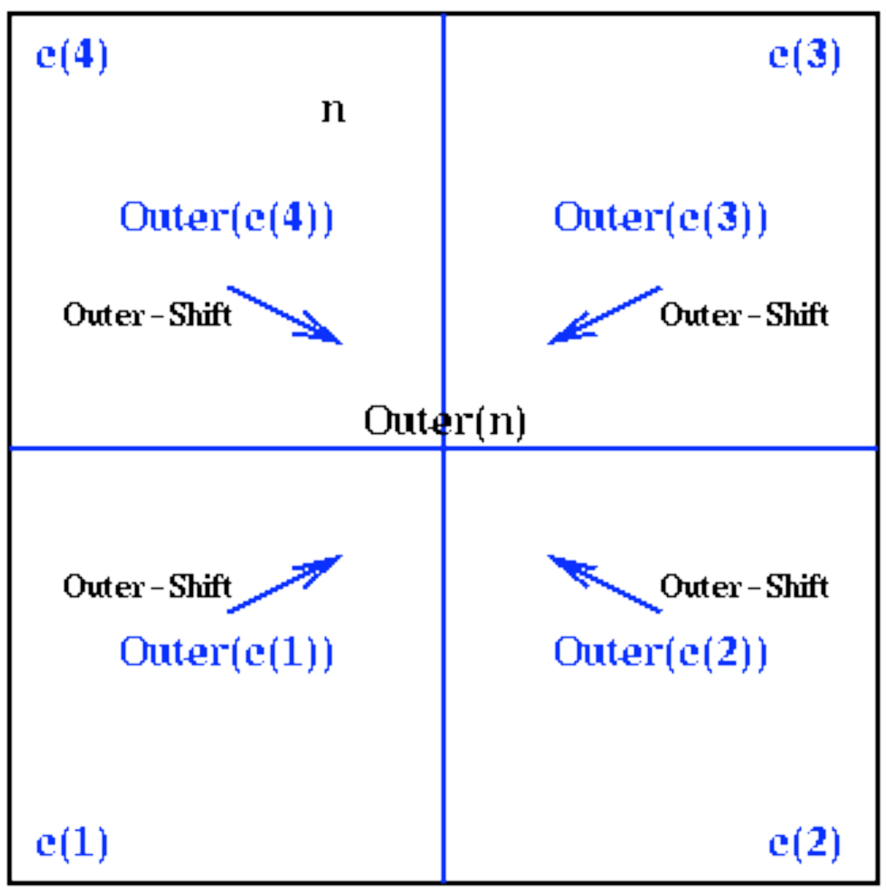
- **Build tree**
- Bottom-up traversal to compute Outer(N)
- Top-down traversal to compute Inner(N)
- For each leaf N, add contributions of nearest particles directly into Inner(N)

# FMM algorithm

- Build tree
- **Bottom-up traversal to compute Outer(N)**
- Top-down traversal to compute Inner(N)
- For each leaf N, add contributions of nearest particles directly into Inner(N)

# Building Outer(N)

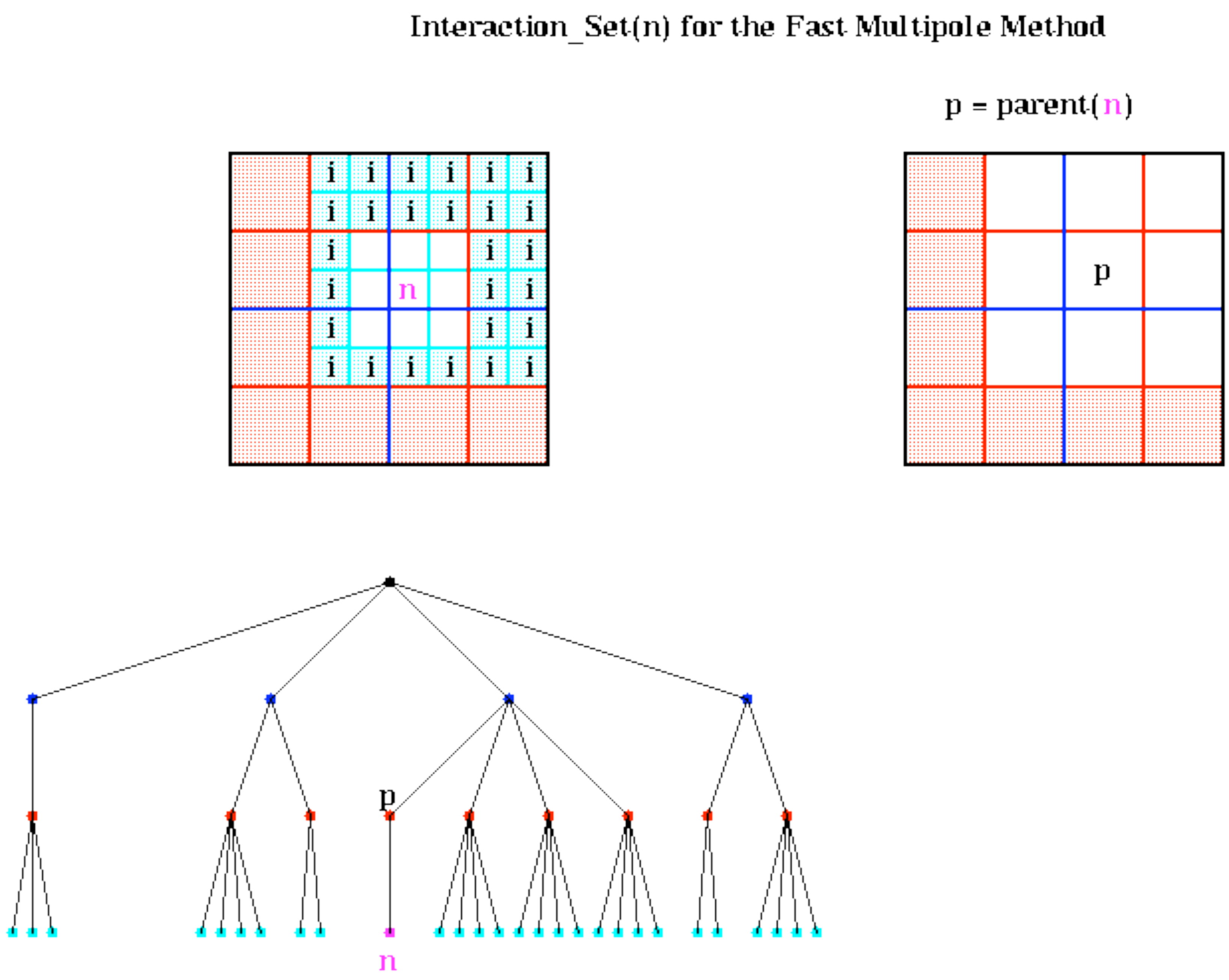
Inner Loop of Build_Outer



# FMM algorithm

- Build tree
- Bottom-up traversal to compute Outer(N)
- **Top-down traversal to compute Inner(N)**
- For each leaf N, add contributions of nearest particles directly into Inner(N)

# Building Inner(N)



# FMM algorithm

- Build tree
- Bottom-up traversal to compute Outer(N)
- Top-down traversal to compute Inner(N)
- **For each leaf N, add contributions of nearest particles directly into Inner(N)**

# Dual-trees

- Build trees for “queries” and “references”
  - Queries = points on which to compute forces
  - References = points contributing to force
  - In physics n-body as we’ve discussed it, these are the same sets of points
- For **pairs of tree nodes**:
  - If “bounds” suggest a result for the pair, use it
  - Else, recurse on all pairs

# Tree code parallelization

# Basic tree-code structure

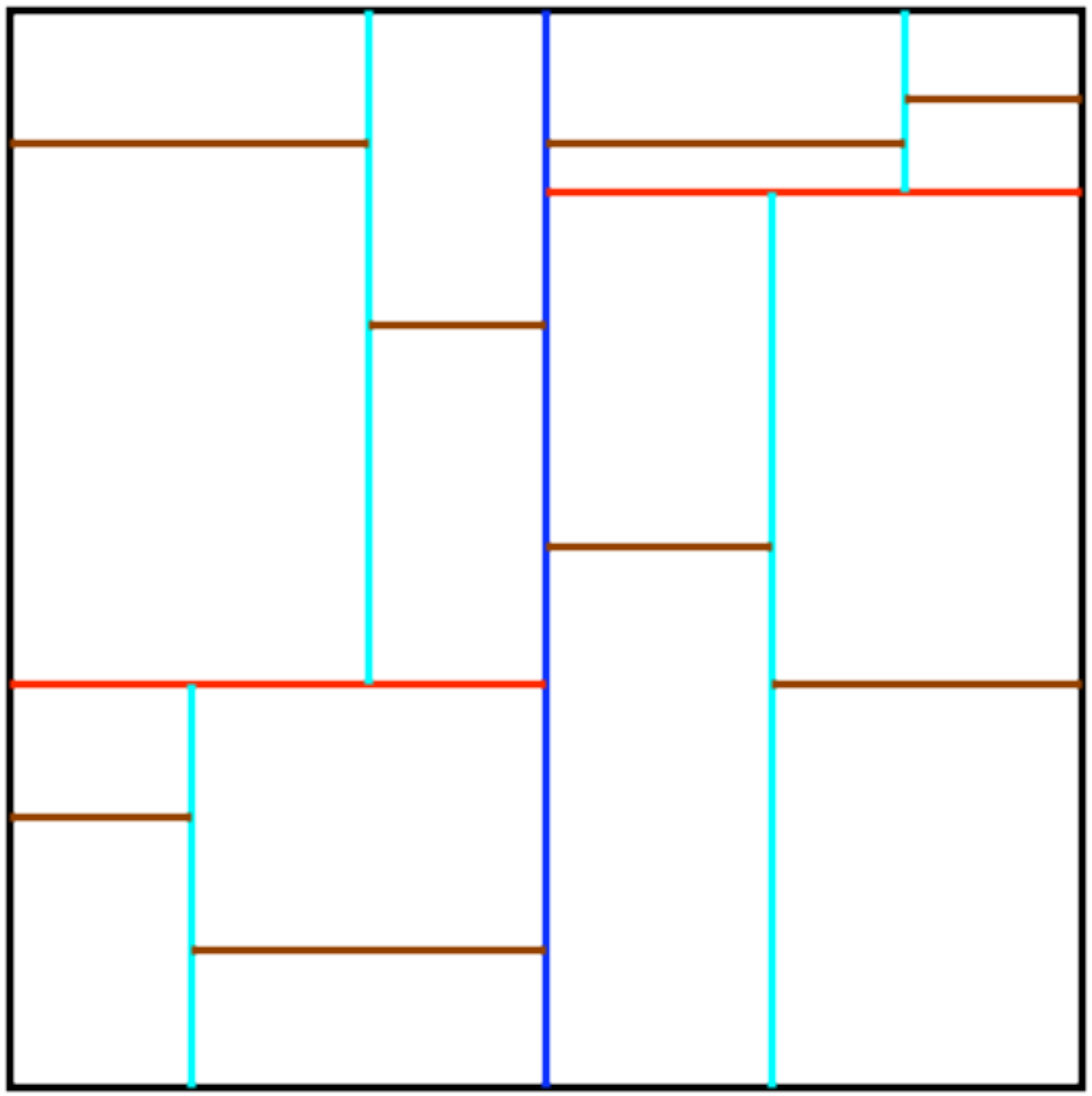
- Build tree
- Traverse from leaves to root to compute outer expansions
  - (In B-H, center-of-mass and total-mass)
- Traverse from root to leaves to compute any inner expansions
- **Traverse to compute forces**

- Question: Load-balancing for force computation?

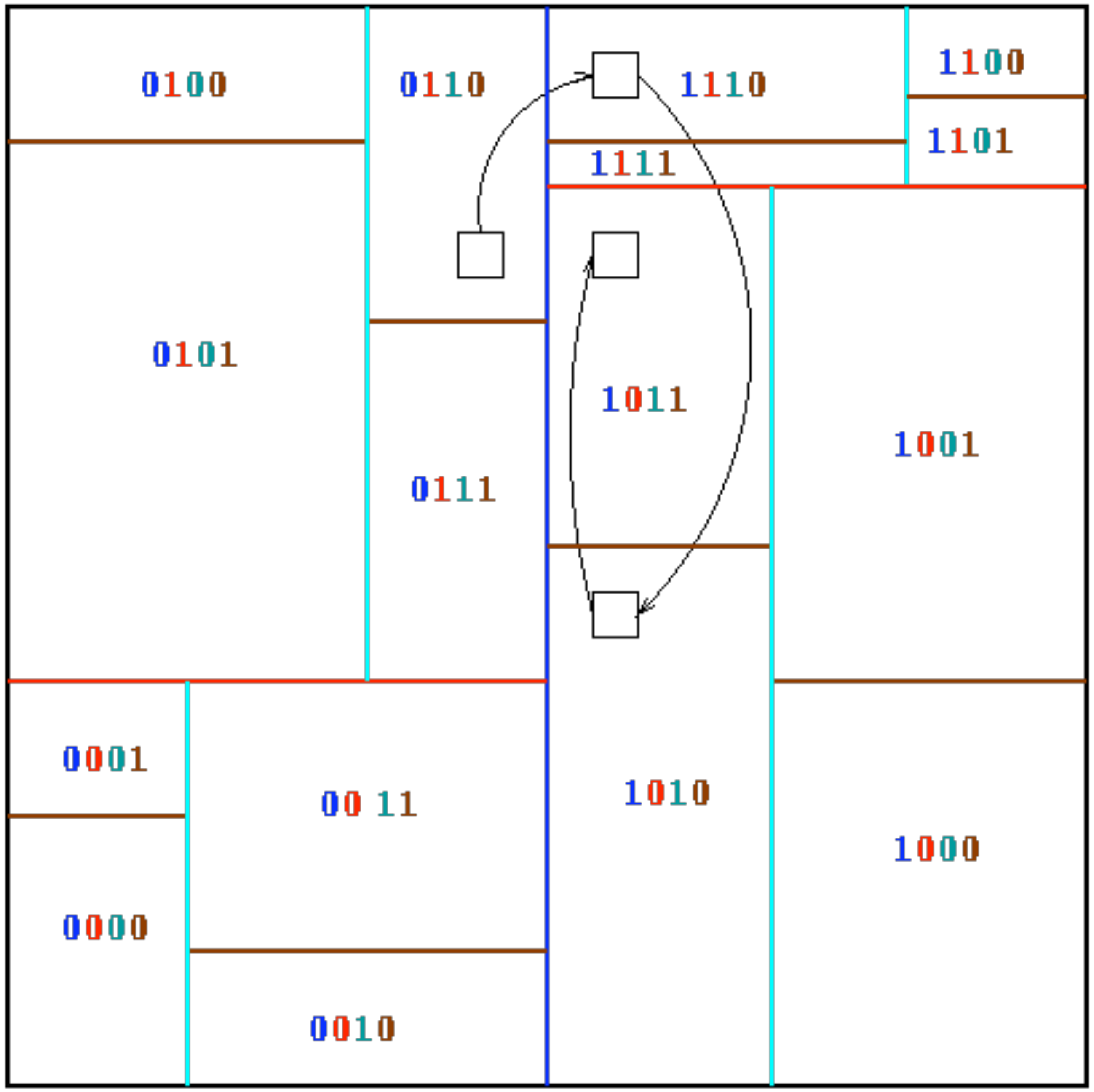
# Scheme 1: Partition space

- Divide space into regions with roughly equal particles in each
- Assign each region to a processor
- Each processor computes **locally essential tree (LET)**

# Orthogonal Recursive Bisection



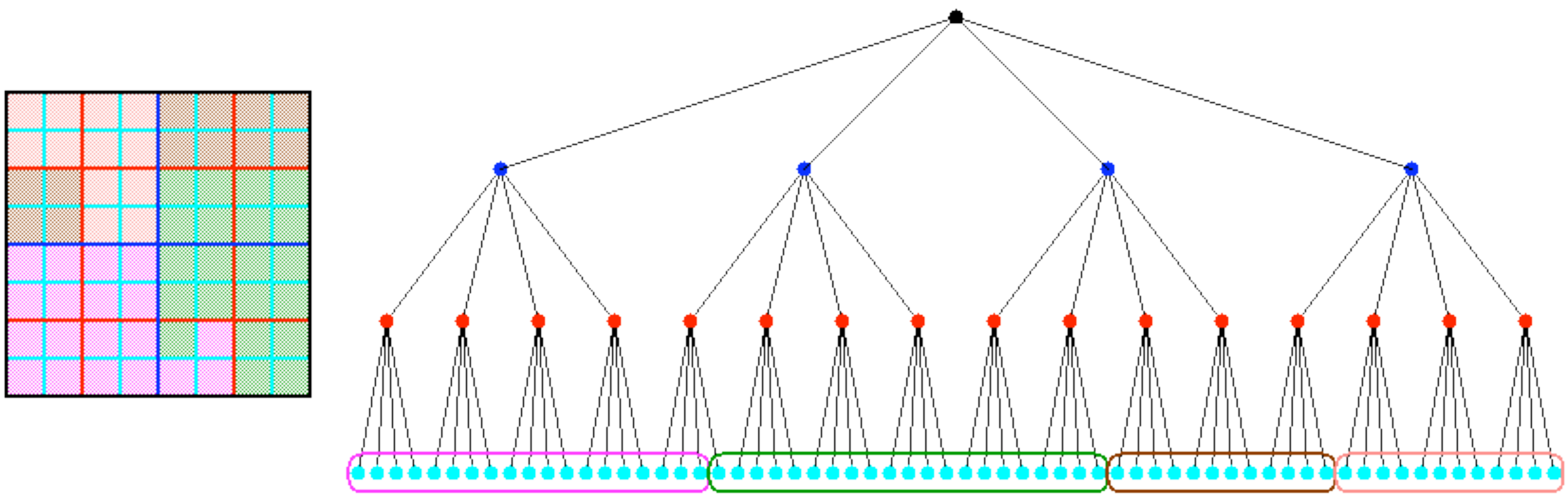
Building a Locally Essential Tree



# Scheme 2: Partition tree

- "Cost-zones" (shared memory); "hashed oct-tree" (distributed)
- Partitioning the tree
  - For each node, estimate work W
  - Linearize tree (many choices)
  - Partition nodes to roughly balance W / p

Using costzones to layout a quadtree on 4 processors
Leaves are color coded by processor color

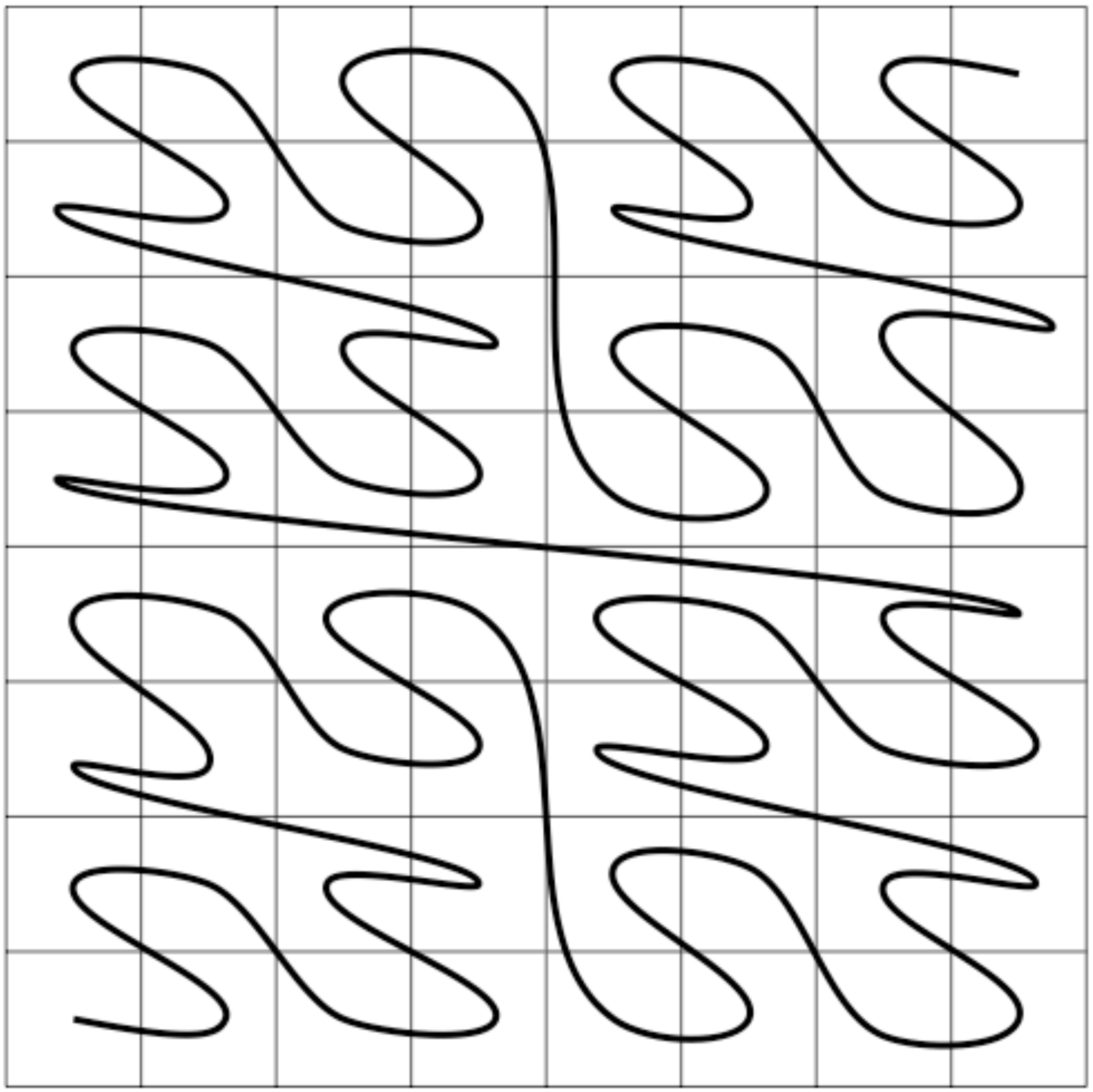


# Scheme 2: Partition tree

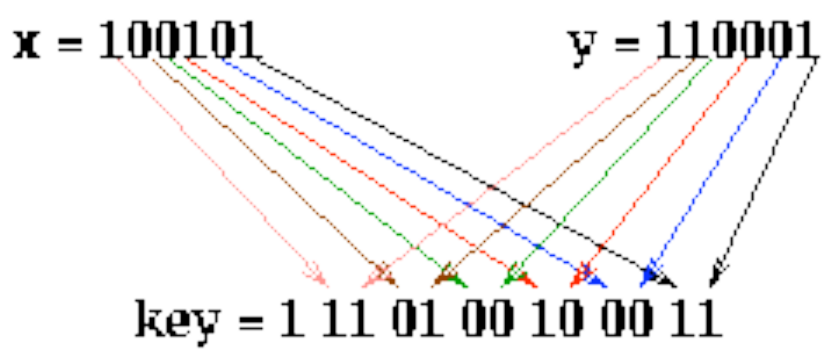
- “Cost-zones” (shared memory); “hashed oct-tree” (distributed)
- Partitioning the tree
  - For each node, estimate work W
  - **Linearize tree (many choices)**
  - Partition nodes to roughly balance W / p

# Linearizing the tree: Hashed quad-/oct-trees

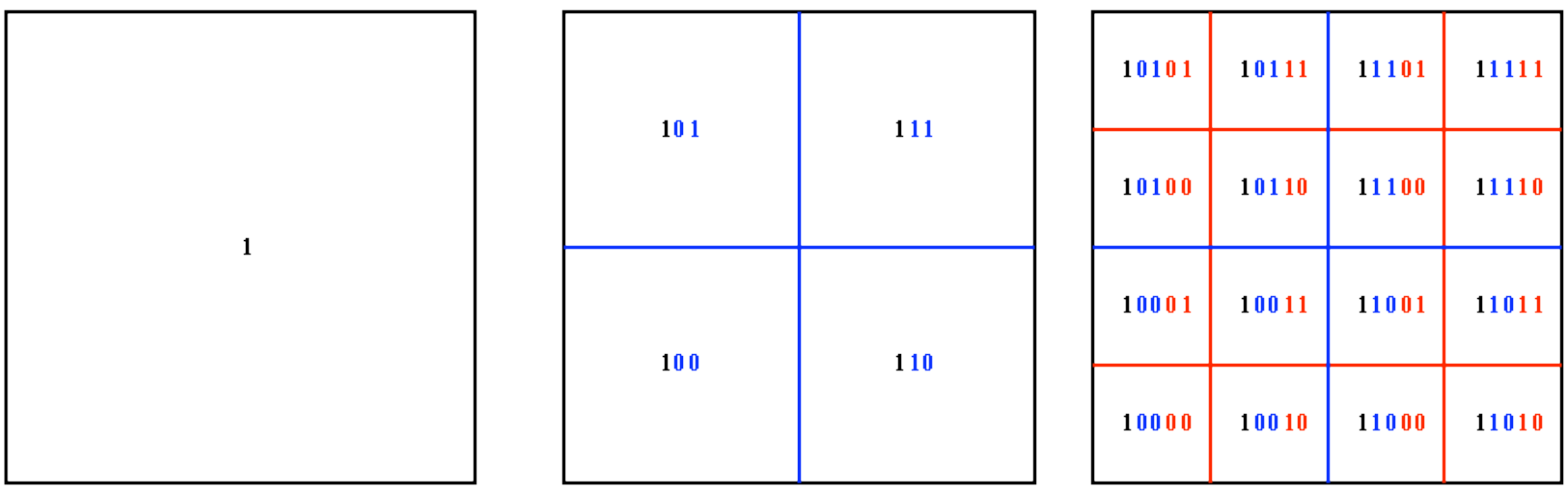
- Scheme:
  - Assign unique key to each node in tree
  - Compute hash(key) : key → global address in hash table
  - Distribute hash table
- Idea: Each processor can find node (with high probability) without traversing links
- Warren & Salmon ’93
  - Key = interleave bits of coordinates
  - hash(key) = bit-mask of bottom ‘h’ bits



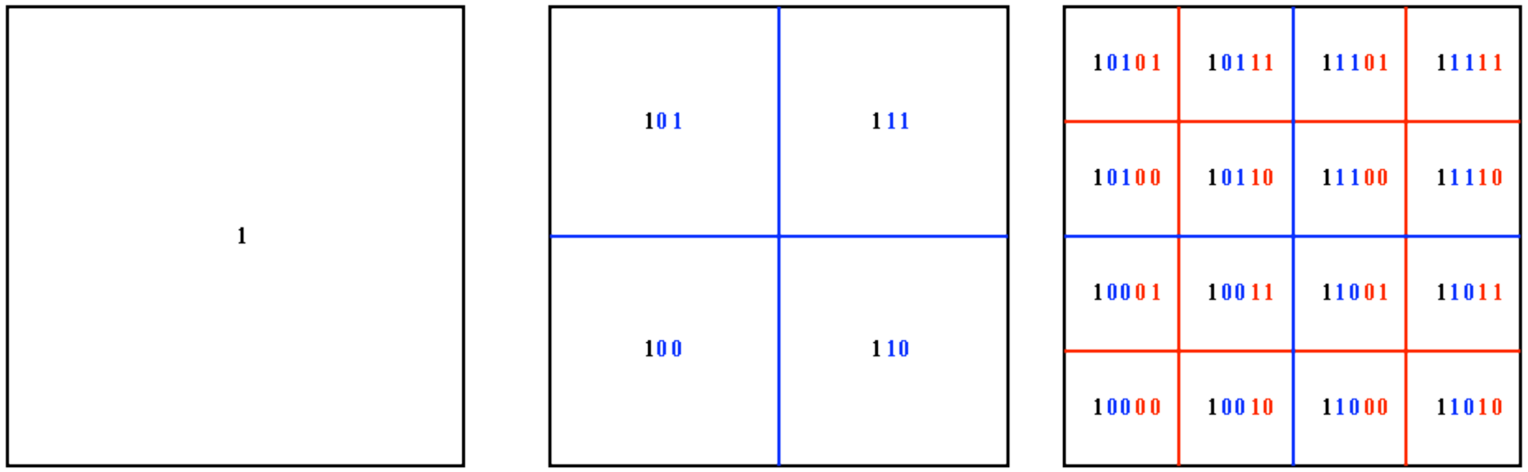
# Building a key for a hashed Quadtree



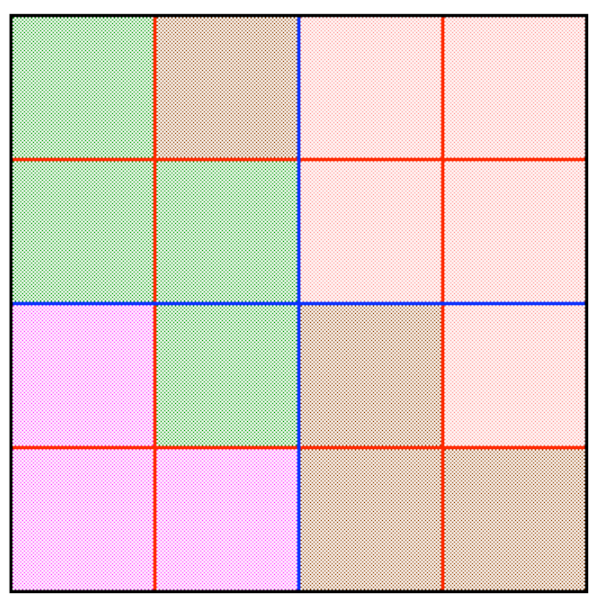

Assigning Keys to Quadtree Nodes



Assigning Keys to Quadtree Nodes




Assigning Hash Table Entries to 4 Processors



# Administrivia

# Final stretch…

- Project checkpoints due already

“In conclusion…”

# Ideas apply broadly

- Physical sciences, *e.g.*,
  - Plasmas
  - Molecular dynamics
  - Electron-beam lithography device simulation
  - Fluid dynamics
- "Generalized" n-body problems: Talk to your classmate, Ryan Riegel

# Backup slides