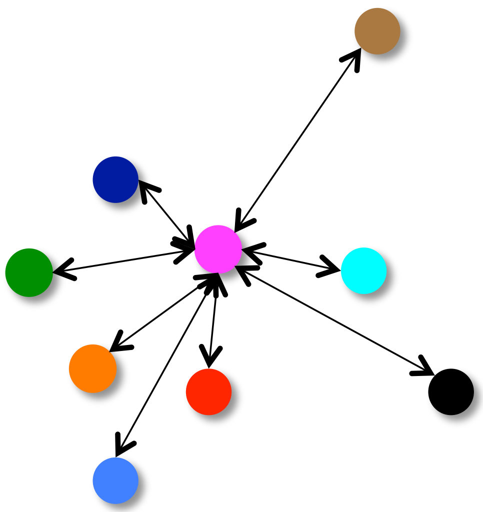
Unsynchronized Techniques for Approximate Parallel Computing

Martin Rinard
MIT EECS, MIT CSAIL
Massachusetts Institute of Technology
Cambridge, MA 02139

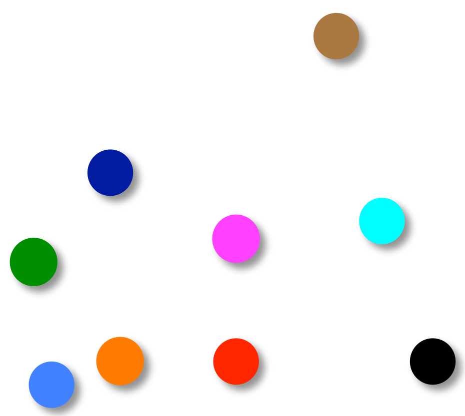
Barnes Hut N-Body Simulation



N Interacting Bodies (Stars, Molecules)

- At each step
  - Compute force
    - Acting on each body
    - From all other bodies

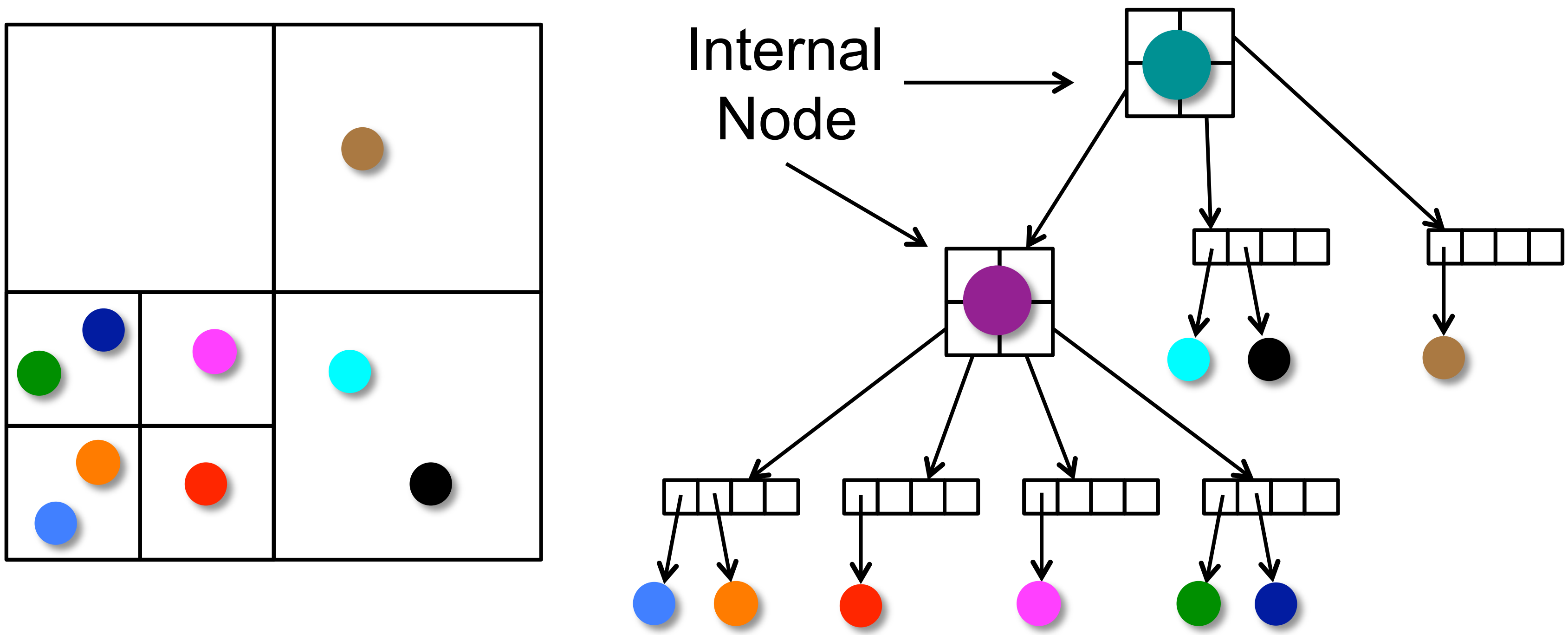
Barnes Hut N-Body Simulation



N Interacting Bodies (Stars, Molecules)

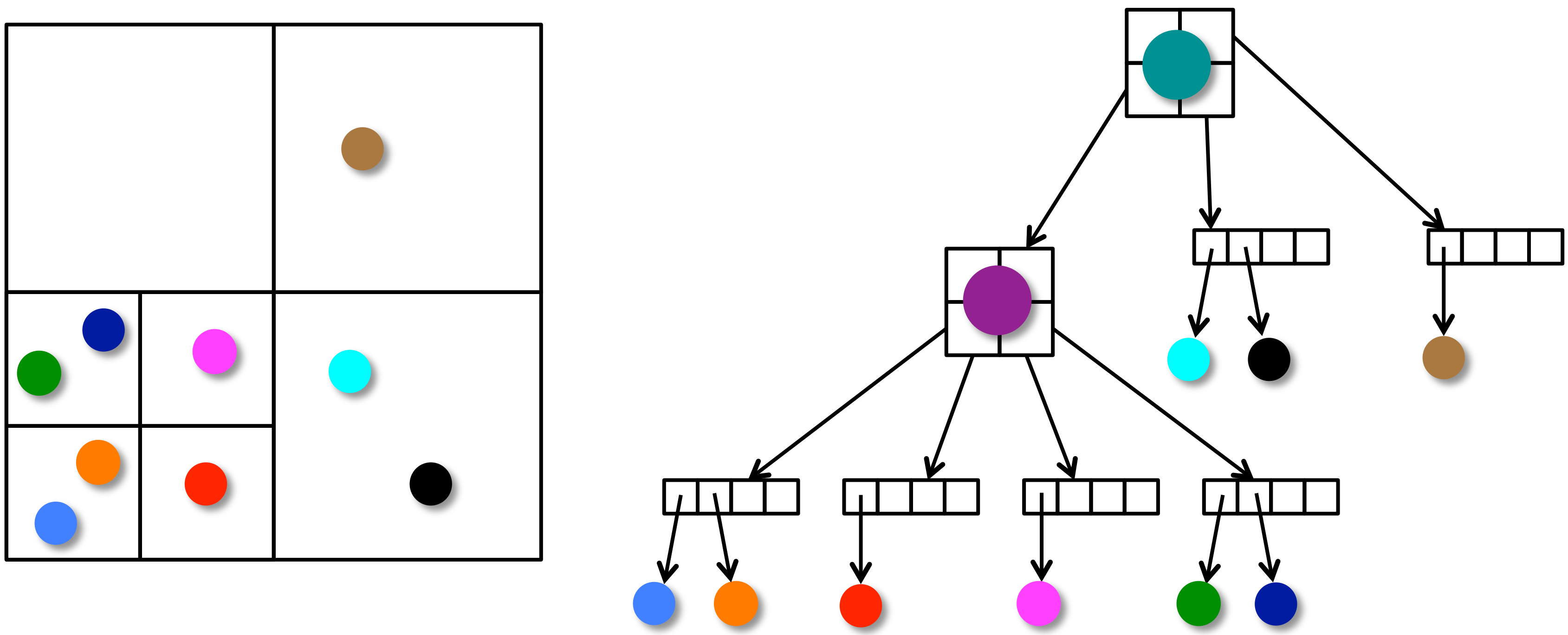
- At each step
  - Compute force
    - Acting on each body
    - From all other bodies
  - Use forces to move bodies
- Repeat

Space Subdivision Tree



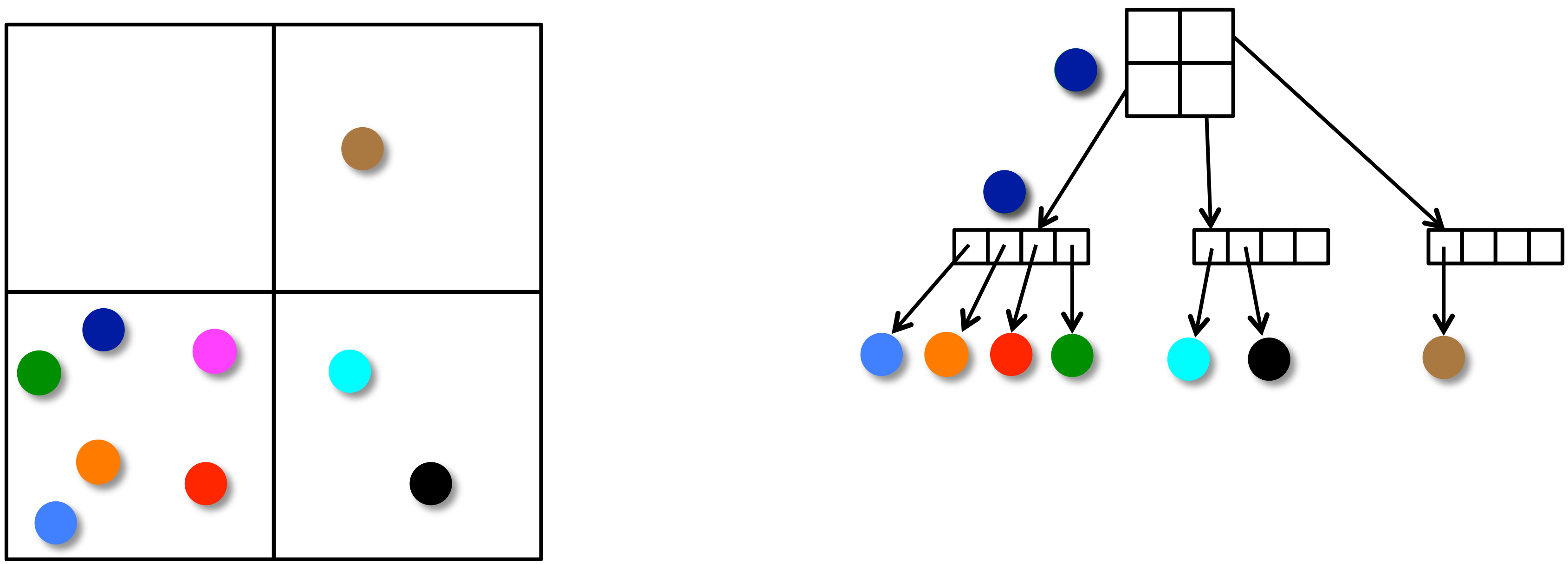
- Internal node contains center of mass for subtree
- Center of mass approximation for distant bodies
- Changes N^2 algorithm into N log N algorithm

Barnes-Hut Algorithm



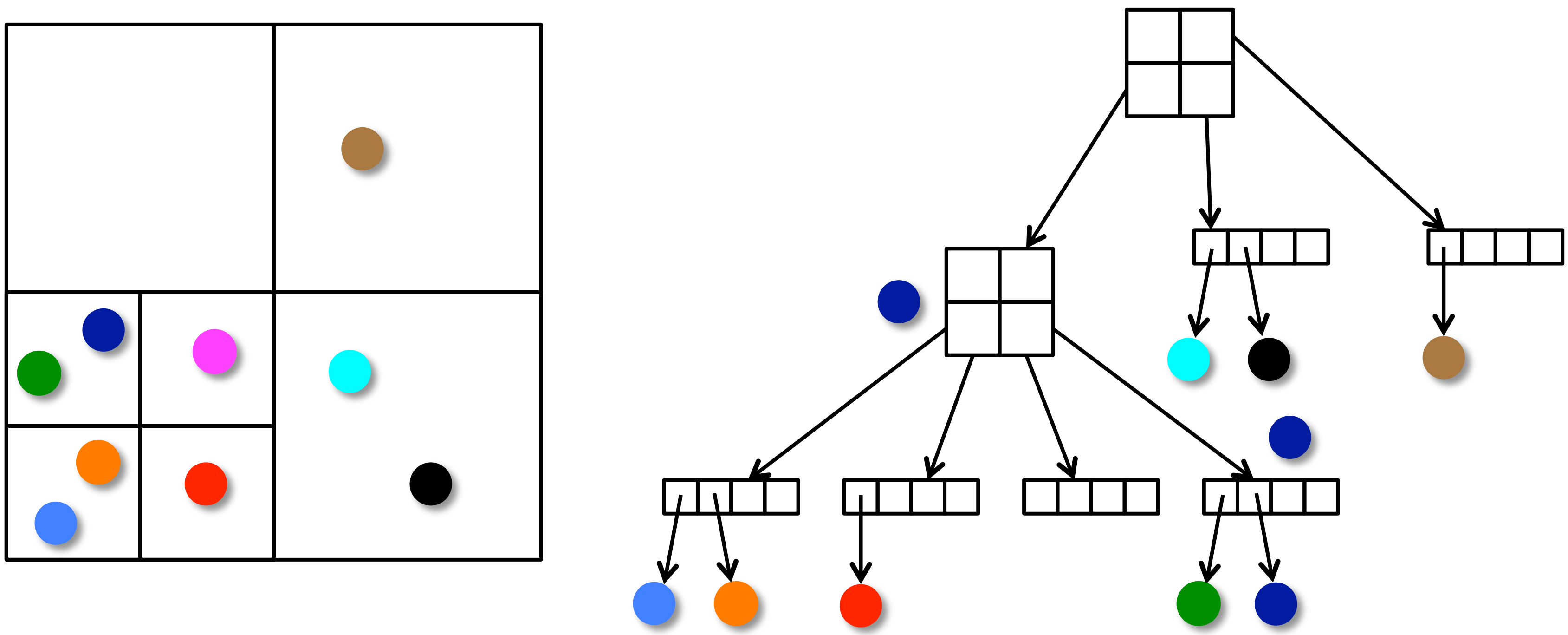
1) Build space subdivision tree
2) Use tree to compute forces acting on each body
3) Move bodies
4) Repeat

Tree Construction Algorithm



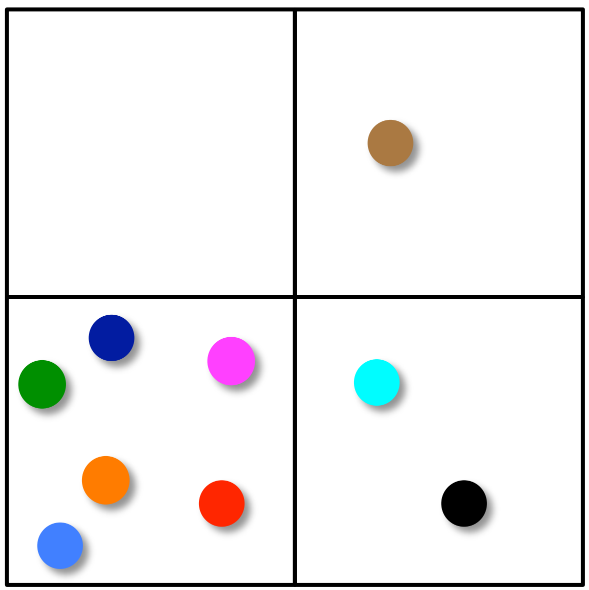
1) Drop bodies into the tree from the top
2) Bodies percolate down the tree
3) Insert each body in correct position
4) Add internal nodes as necessary

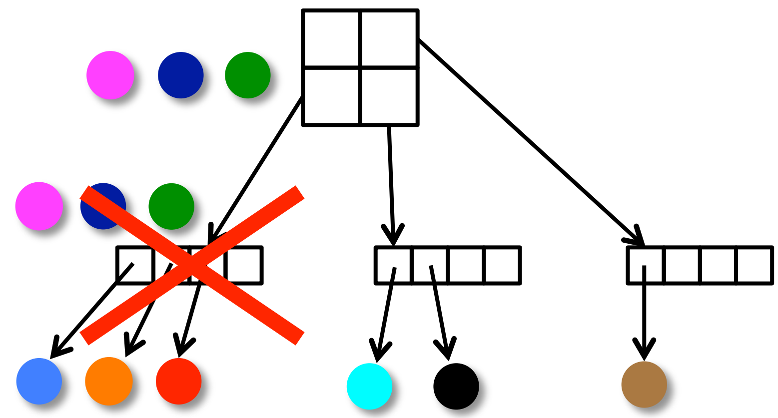
Tree Construction



1) Drop bodies into the tree from the top
2) Bodies percolate down the tree
3) Insert each body in correct position
4) Add internal nodes as necessary

Parallel Tree Construction Algorithm, Version 1 – No Synchronization, Just Insert Bodies in Parallel





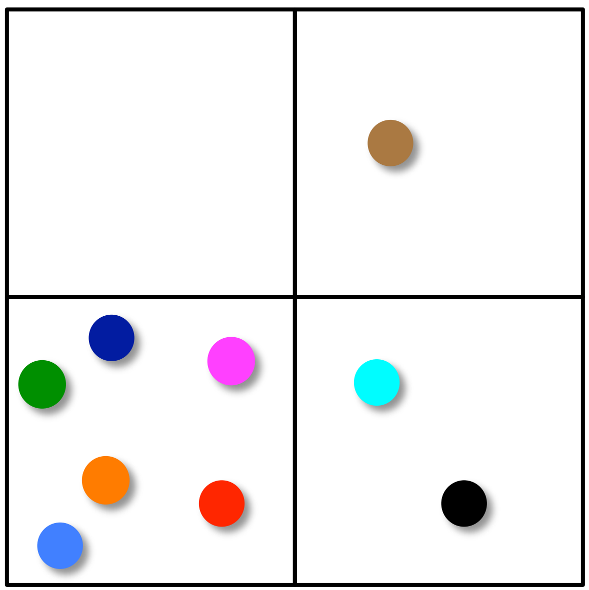
Data Races
Corruption
Crash!

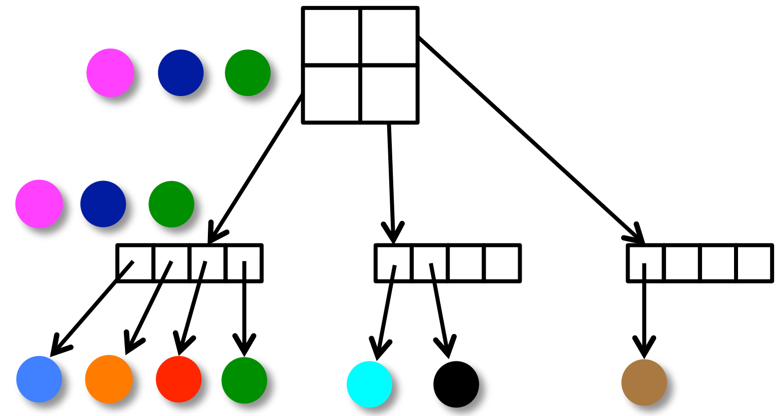
1) Drop bodies into the tree from the top
2) Bodies percolate down the tree
3) Insert each body in correct position
4) Add internal nodes as necessary

Standard Solution

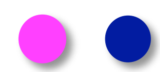
- Add synchronization
  - Complicate program
  - Add synchronization overhead
  - Add space overhead
  - Maybe contention, even deadlock too
- We aren't going to do this

Parallel Tree Construction Algorithm, Version 2 – No Synchronization, No Crash!



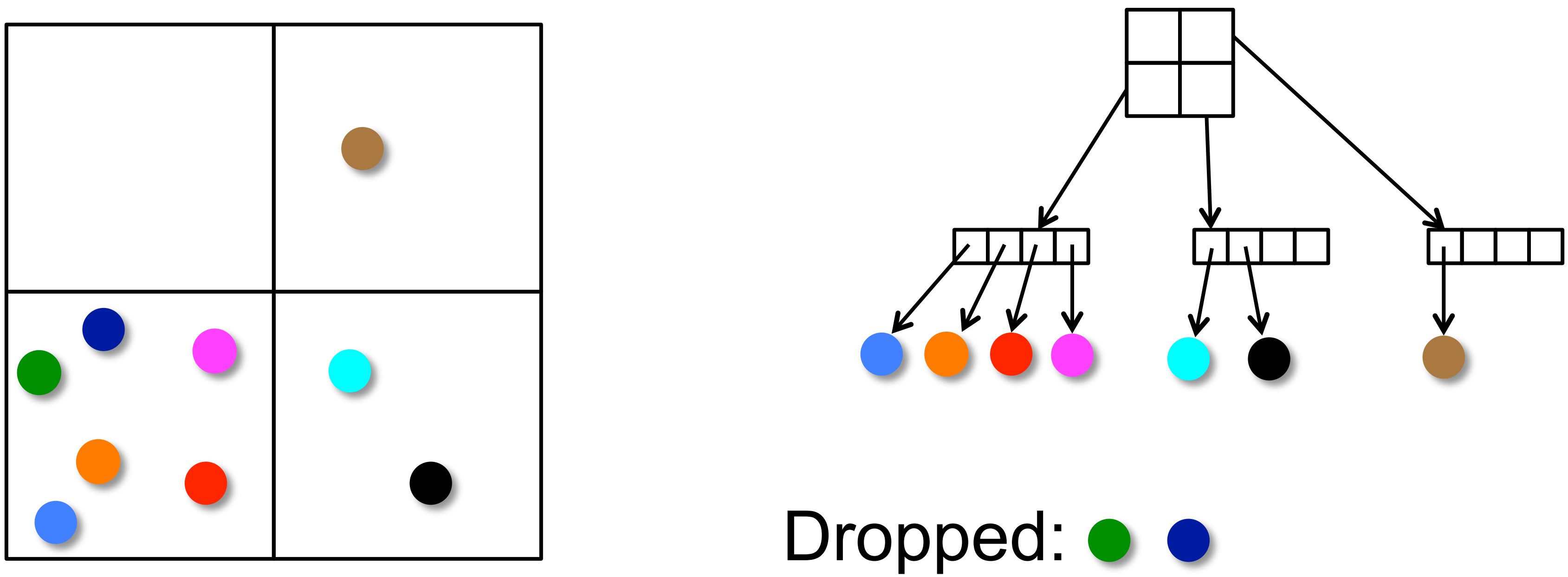


Potential Outcome:

Drop 

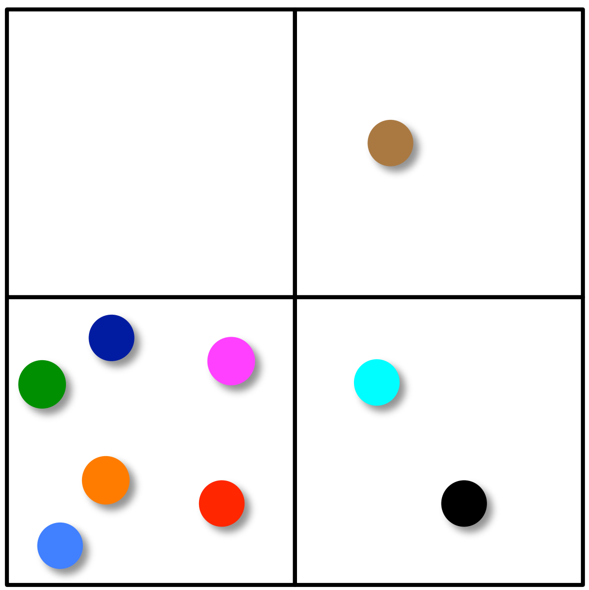
1) Drop bodies into the tree from the top
2) Bodies percolate down the tree
3) Insert each body in correct position
4) Add internal nodes as necessary

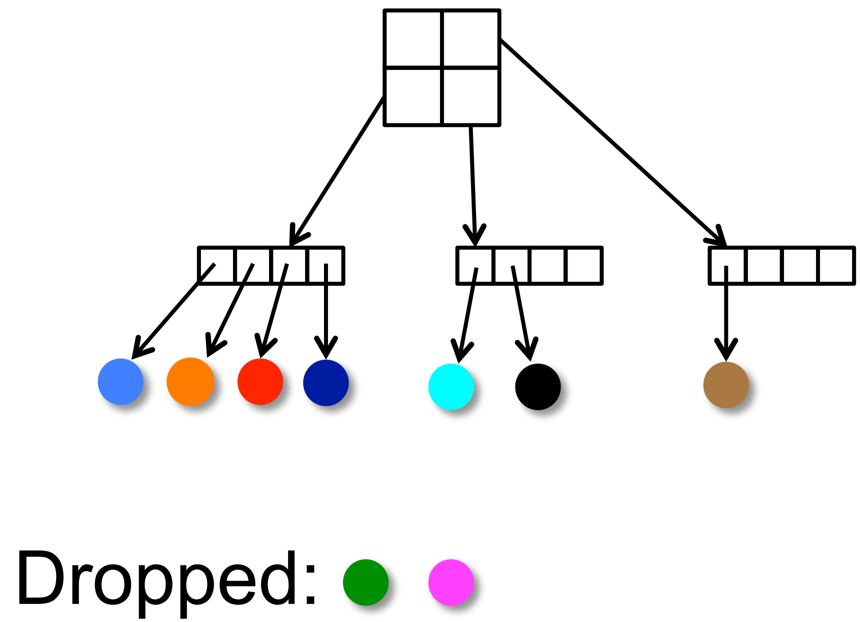
Parallel Tree Construction Algorithm, Version 2 – No Synchronization, No Crash!



1) Drop bodies into the tree from the top
2) Bodies percolate down the tree
3) Insert each body in correct position
4) Add internal nodes as necessary

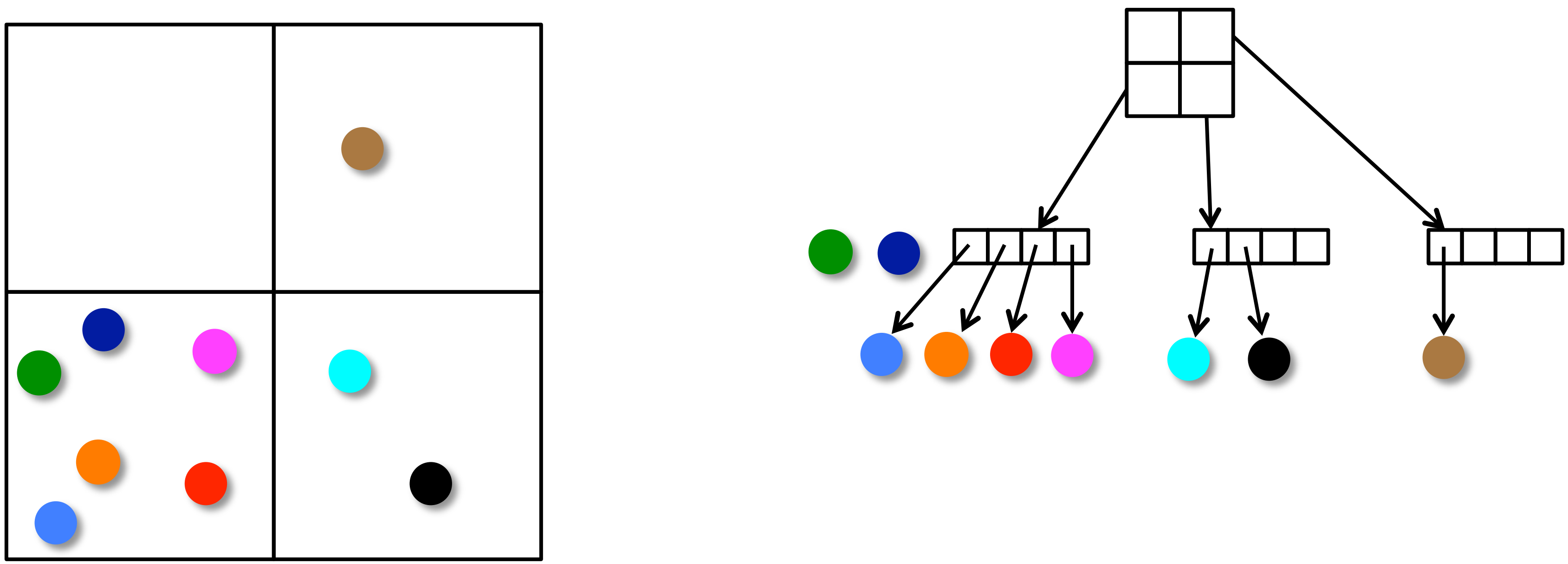
Parallel Tree Construction Algorithm, Version 2 – No Synchronization, No Crash!





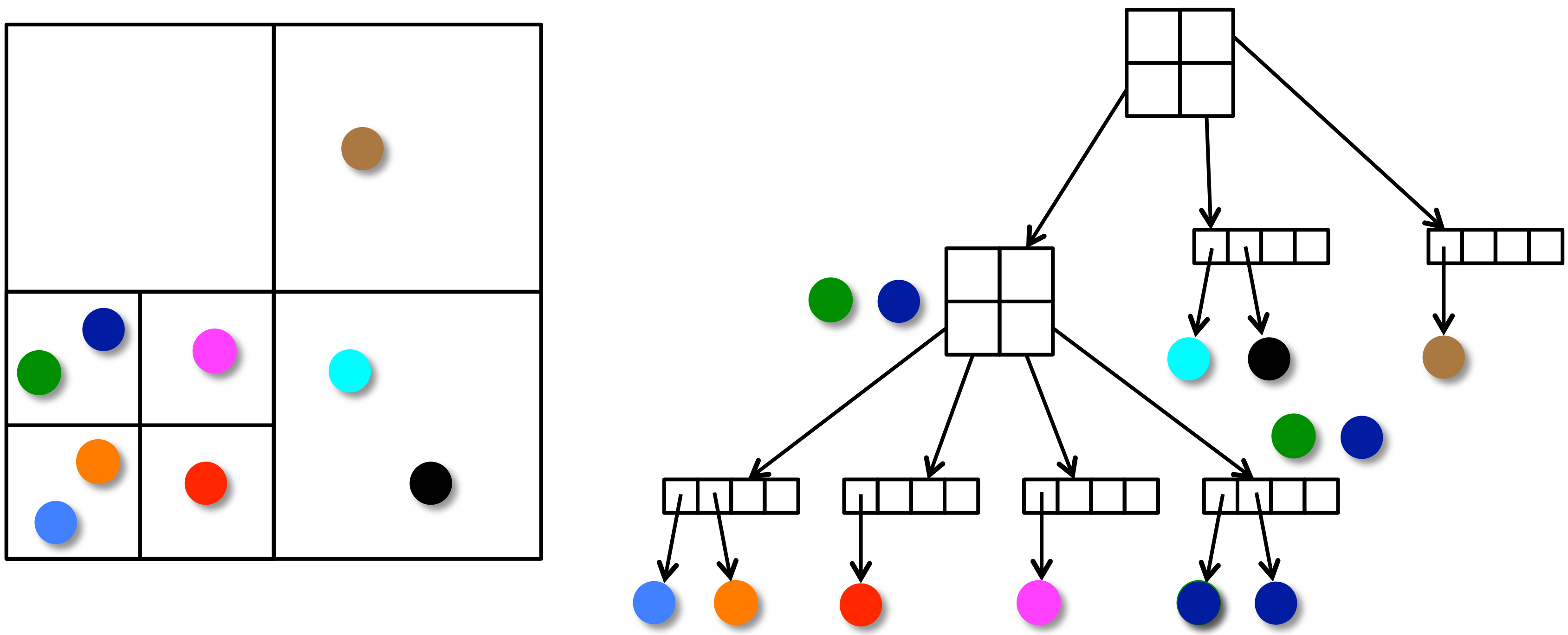
1) Drop bodies into the tree from the top
2) Bodies percolate down the tree
3) Insert each body in correct position
4) Add internal nodes as necessary

Parallel Tree Construction Algorithm, Version 2 – No Synchronization, No Crash!



1) Drop bodies into the tree from the top
2) Bodies percolate down the tree
3) Insert each body in correct position
4) Add internal nodes as necessary

Parallel Tree Construction Algorithm, Version 2 – No Synchronization, No Crash!



1) Drop bodies into the tree from the top
2) Bodies percolate down the tree
3) Insert each body in correct position
4) Add internal nodes as necessary

Effect of Unsynchronized Construction

- Always produces a tree that force computation calculation can use
- But tree may not contain all bodies
  - Forces computed as if dropped bodies don't exist
  - Get an approximate force computation
  - But force computation is already approximate because of center of mass approximation
- Preserves integrity
- May affect accuracy

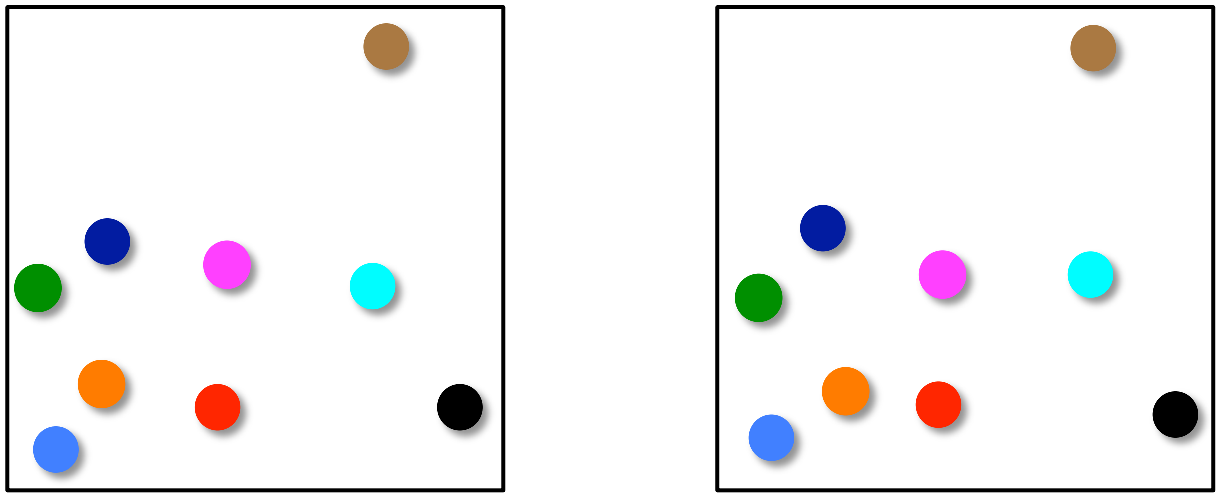
Parallel Tree Construction Options

- **Unsynchronized**
  - Data races
  - Always produces a usable tree, but may drop bodies
    - Preserves integrity of computation
    - May affect accuracy
- **Tree Locking (standard approach)**
  - No data races, no dropped bodies
  - Lots of synchronization, lots of contention at top of tree
- **Update Locking (synchronize updates but not reads)**
  - Data races, no dropped bodies
  - Some synchronization, little contention
  - Complex, scary algorithm

Accuracy Evaluation

- Need a comparison point
- Full N^2 computation too expensive
- Compare with **hyperaccurate** version
  - Uses a center of mass approximation
  - But goes deeper into the tree before using the center of mass approximation
  - So more accurate

Accuracy Metric

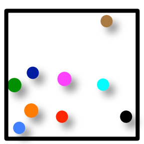


- Start with two corresponding configurations
  - Reference (hyperaccurate)
  - Comparison (tree locking, update locking, unsynchronized)
- Compute sum of distances between corresponding bodies
- Divide sum by distance between corners of bounding box

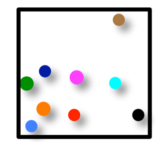
Accuracy Result

- Accuracy metric between
  - Hyperaccurate version and
  - All other versions
  - All other numbers of processors
- Is **1.02%** (to three significant digits)

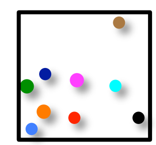
Visually



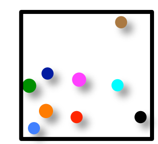
Hyperaccurate



Unsynchronized



Tree Locking



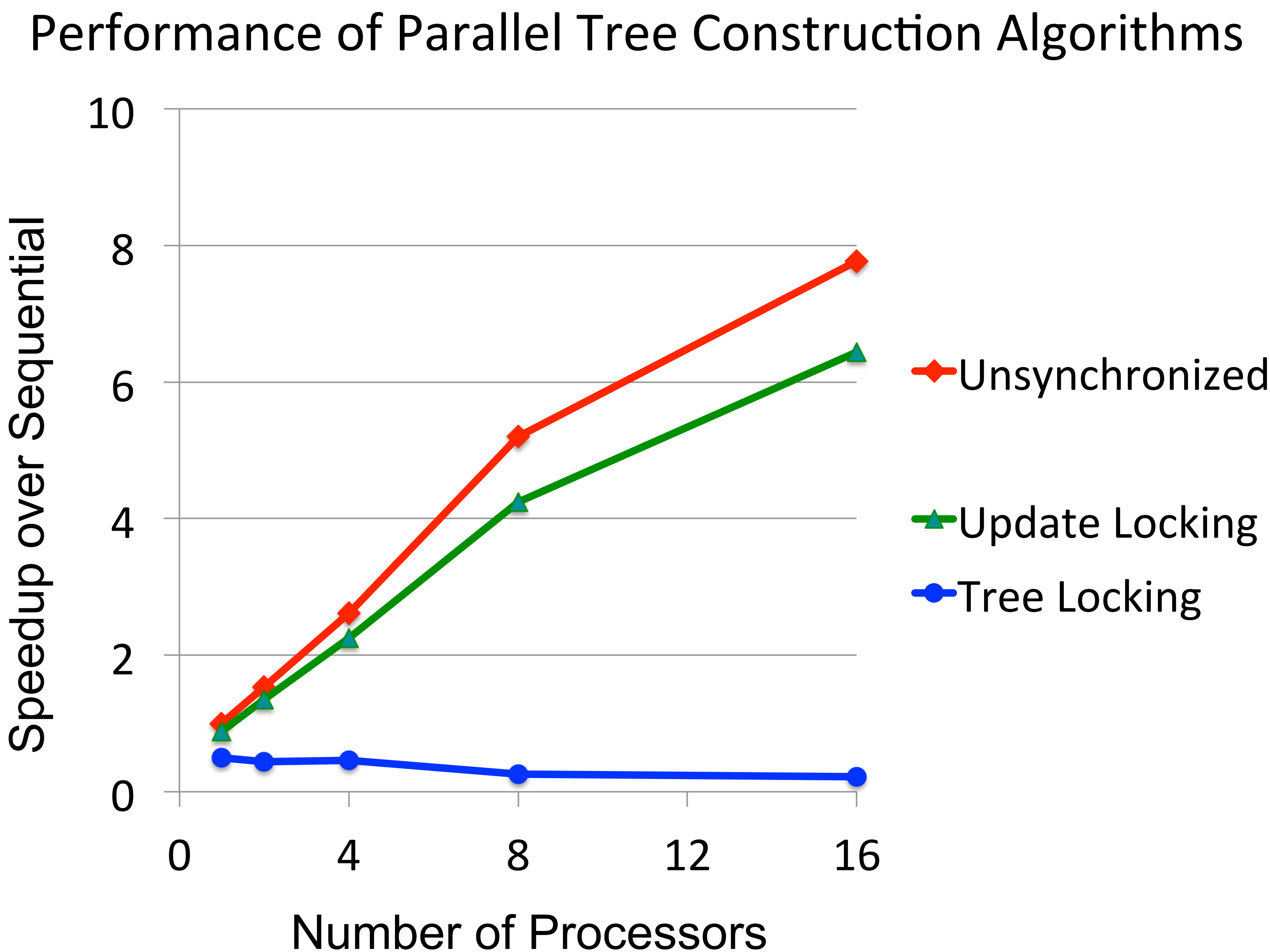
Update Locking

Accuracy Result in Context

- Difference between hyperaccurate and synchronized version is **two orders of magnitude larger than** difference between synchronized and unsynchronized versions
- Changing center of mass approximation has an accuracy effect **two orders of magnitude more than** removing synchronization

Performance Evaluation

- Implemented different versions
  - Tree locking
  - Update locking
  - Unsynchronized
  - Evaluated performance of different versions
- Measured speedup over sequential tree construction algorithm


Performance of Parallel Tree Construction Algorithms
Speedup over Sequential
Number of Processors
0
2
4
6
8
10
0
4
8
12
16
Unsynchronized
Update Locking
Tree Locking

Key Technical Concepts and Results

- Safe approximate unsynchronized data structures
  - Look just like standard sequential data structures
  - Give programmer sense of security and comfort
  - Building blocks in paper
- Accuracy evaluation methodology
  - To evaluate effect of unsynchronized data structures
  - Adjust accepted accuracy knobs
  - Compare with effect of removing synchronization
- Evidence shows programs are oversynchronized

Bigger Picture

- Field has traditionally aspired to perfection
- But perfection is increasingly unavailable
  - Huge systems, huge data sets
  - Something always broken
- Software is inherently resilient and malleable
- Enables more mature and productive approaches
- Goal is acceptability, not perfection
  - Integrity (program does not crash)
  - Accuracy (accurate enough, often enough)
  - End-to-end perspective
- Opens up new, counterintuitive possibilities