Partitioning and Divide-and-Conquer Strategies

Version:
~~May 2022~~
November 2022

Credit: ITCS 4/5145 Cluster Computing, UNC-Charlotte, B. Wilkinson, 2007
And with some modifications by Guy Tel-Zur, 2017, 2022.

Partitioning

Partitioning simply divides the problem into parts and then compute the parts and combine results.

Divide and Conquer

Characterized by dividing problem into sub-problems of same form as larger problem. Further divisions into still smaller sub-problems, usually done by recursion.

Recursive divide and conquer amenable to parallelization because separate processes can be used for divided parts. Also usually data is naturally localized.

Partitioning/Divide and Conquer Examples

Many possibilities.

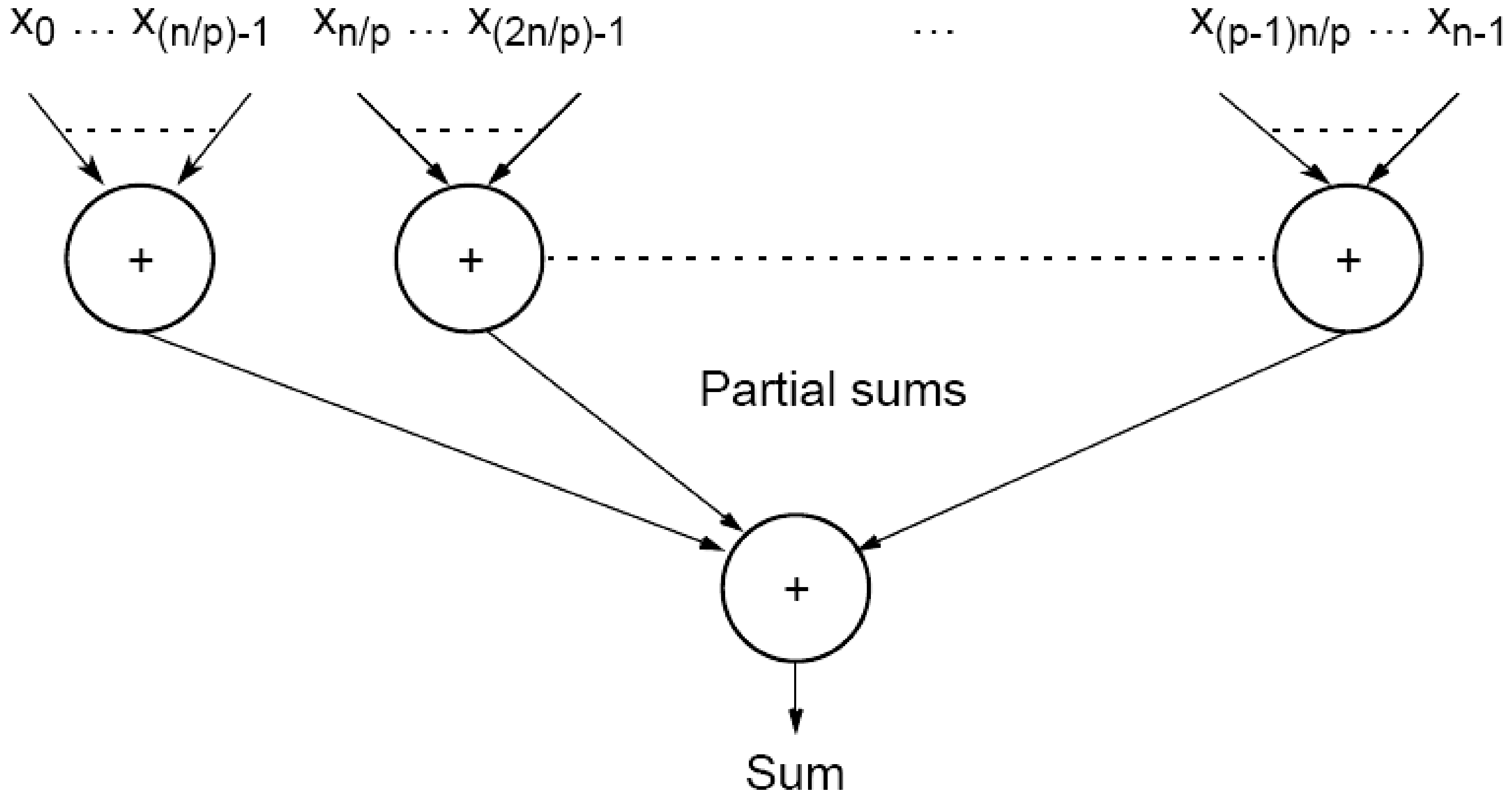
- Operations on **sequences of number** such as simply adding them together

- Several **sorting algorithms** can often be partitioned or constructed in a recursive fashion

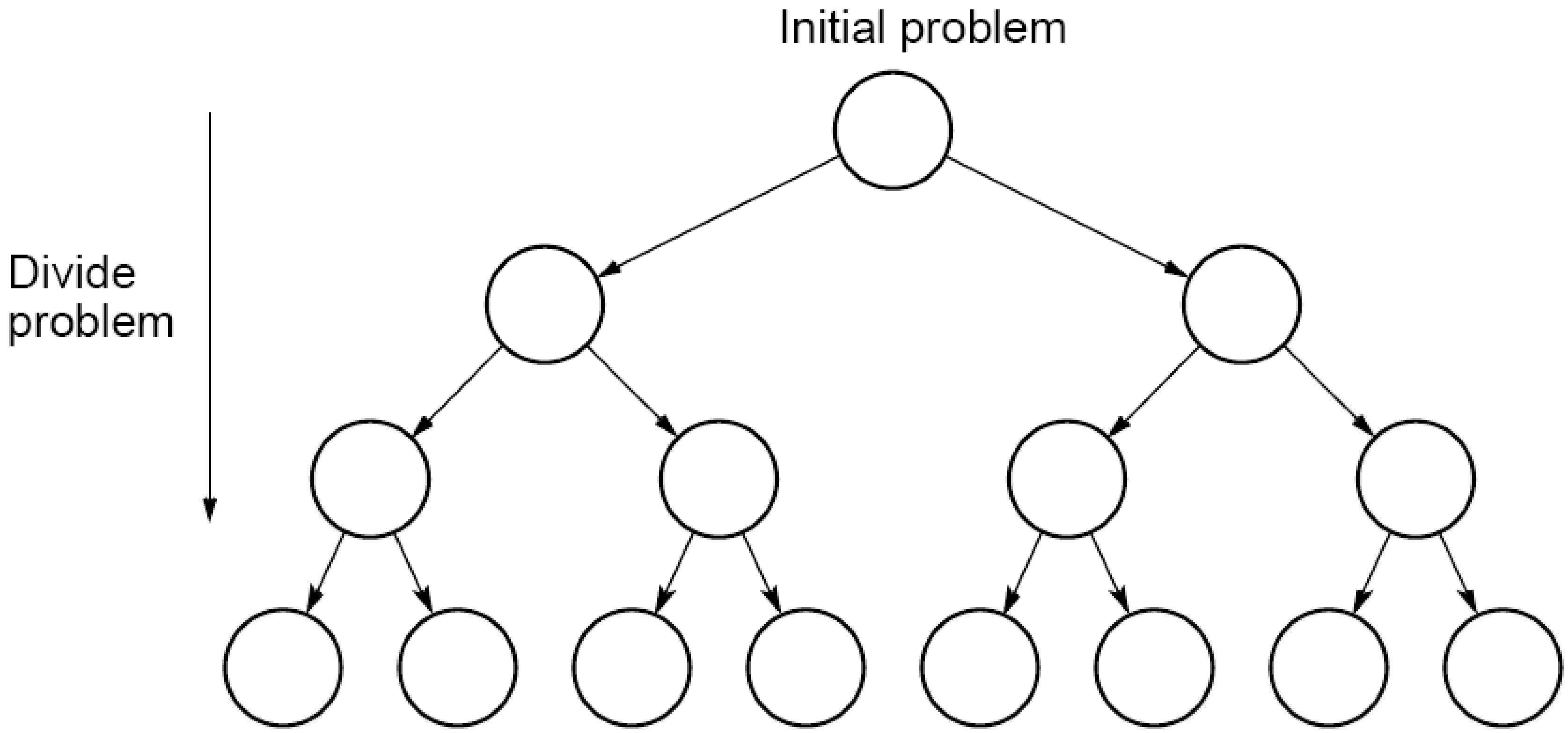
- **Numerical integration**

- ***N*-body problem**

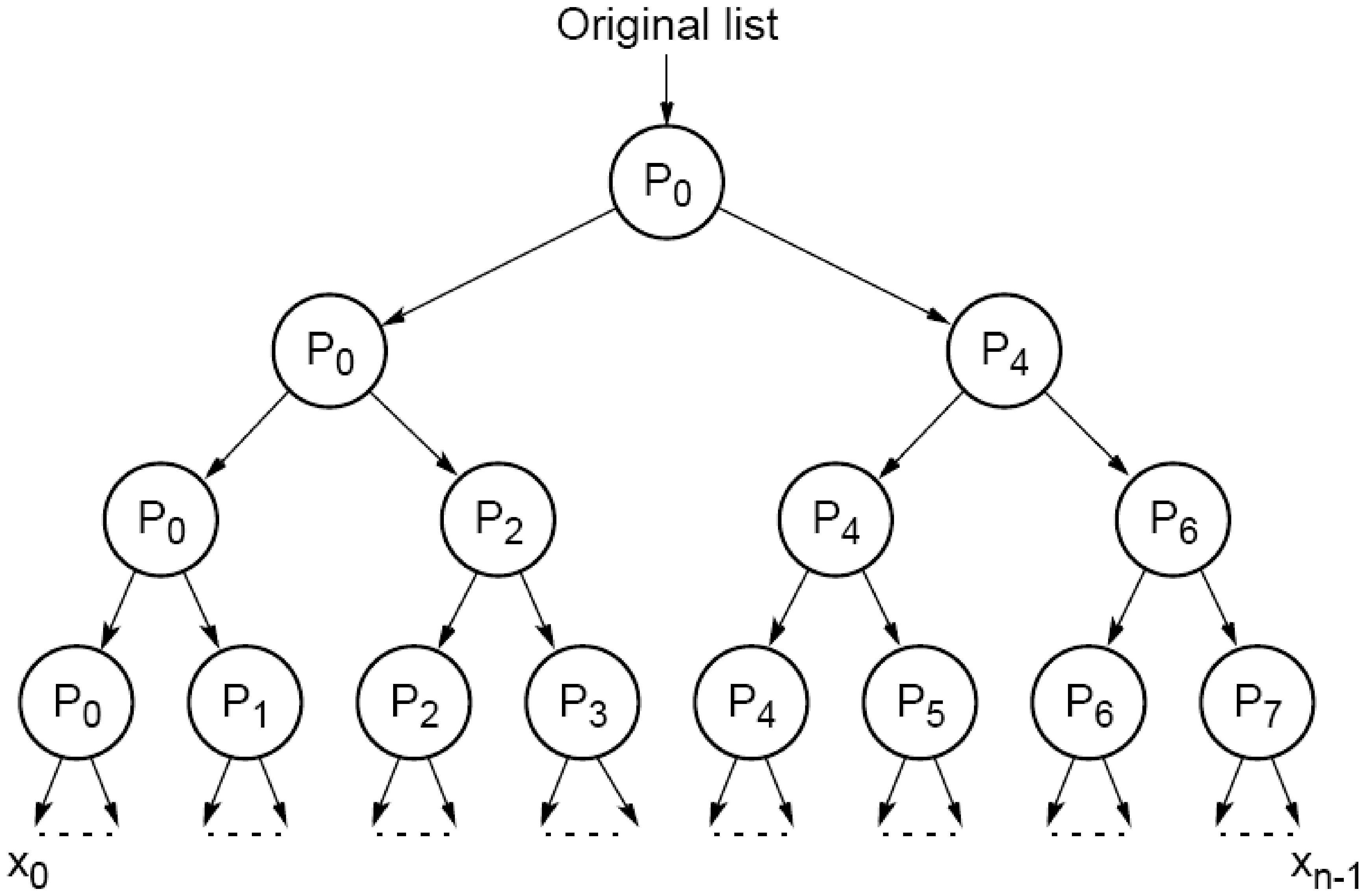
Partitioning a sequence of numbers into parts and adding the parts



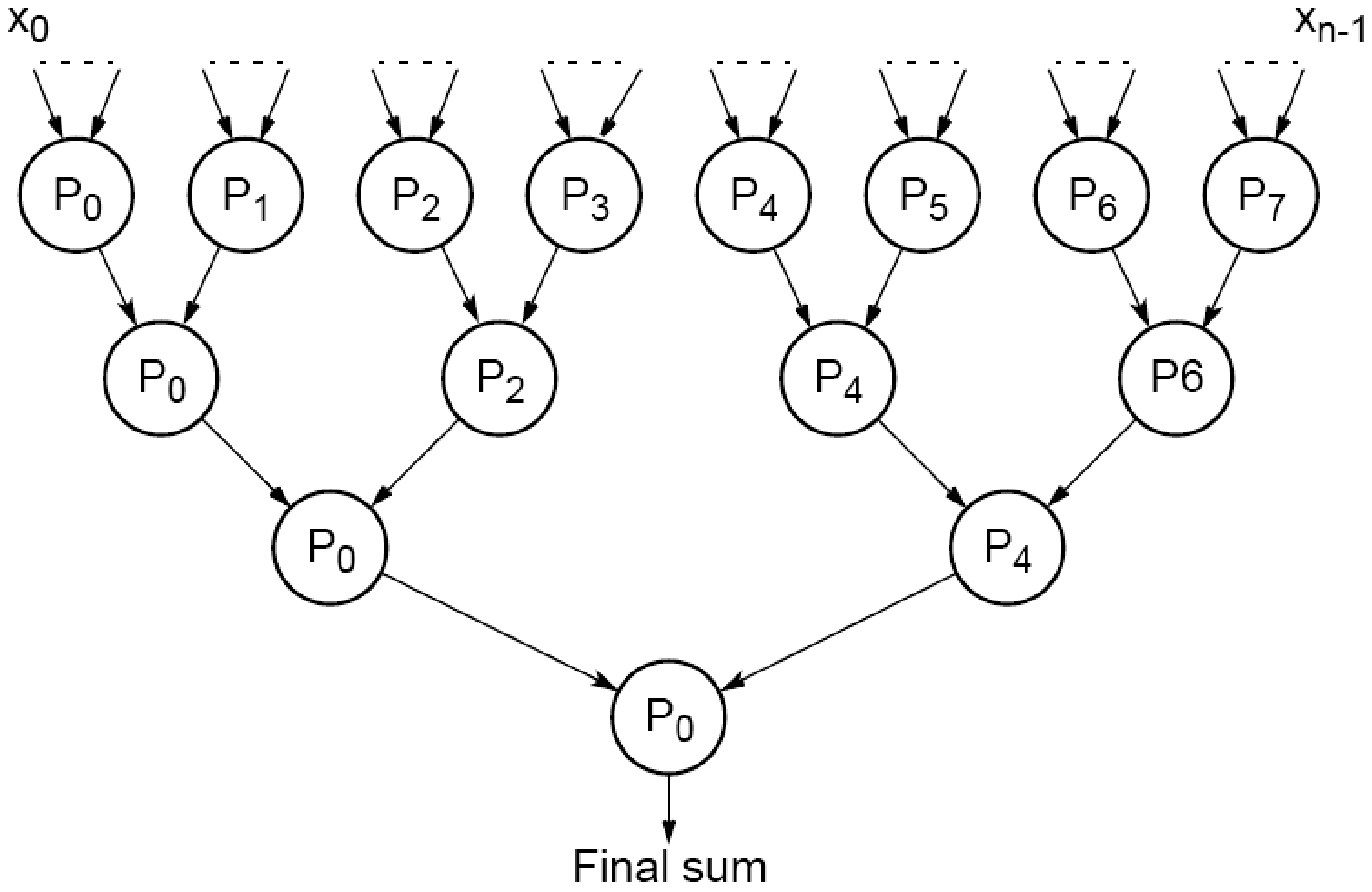
Tree construction



Dividing a list into parts



Partial summation



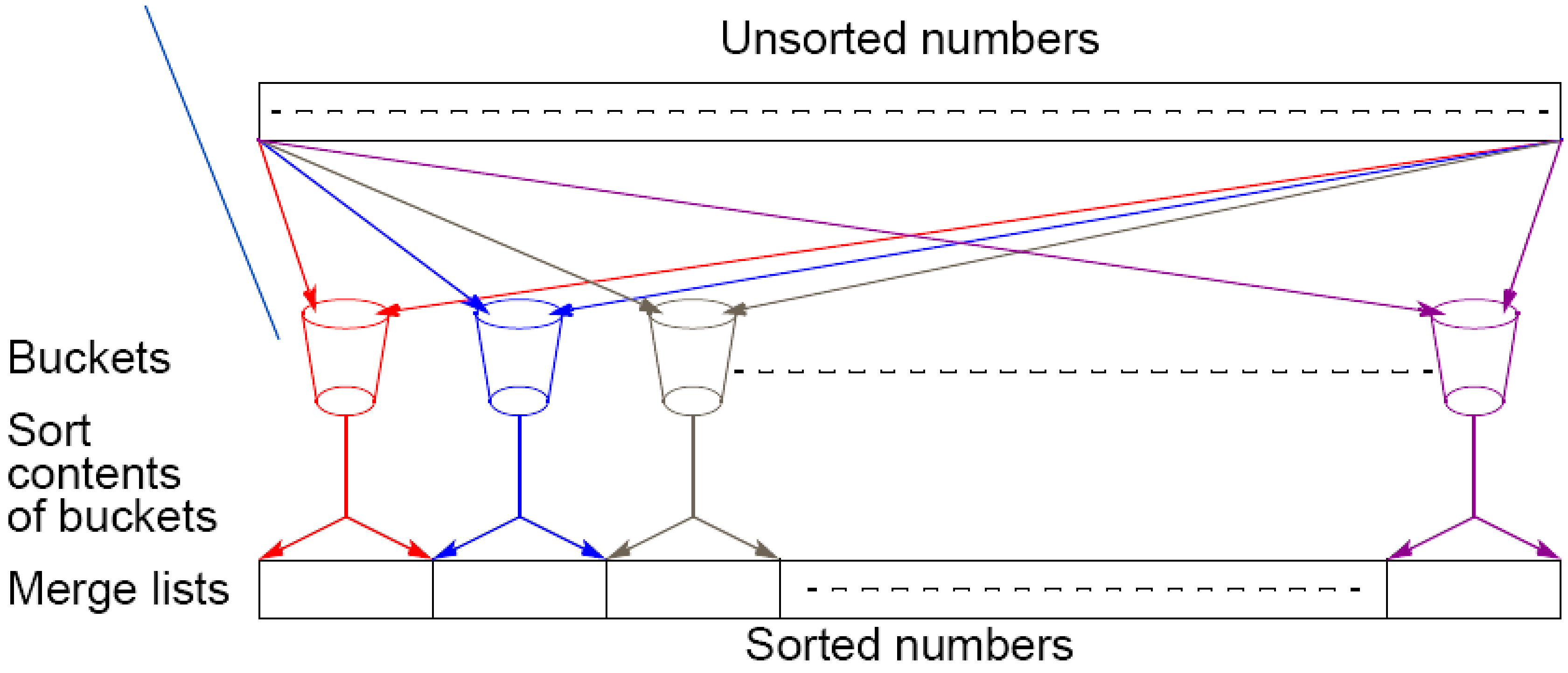
Many Sorting algorithms can be parallelized by partitioning and by divide and conquer.

Example

Bucket sort

Bucket sort

One "bucket" assigned to hold numbers that fall within each region.
Numbers in each bucket sorted using a sequential sorting algorithm.

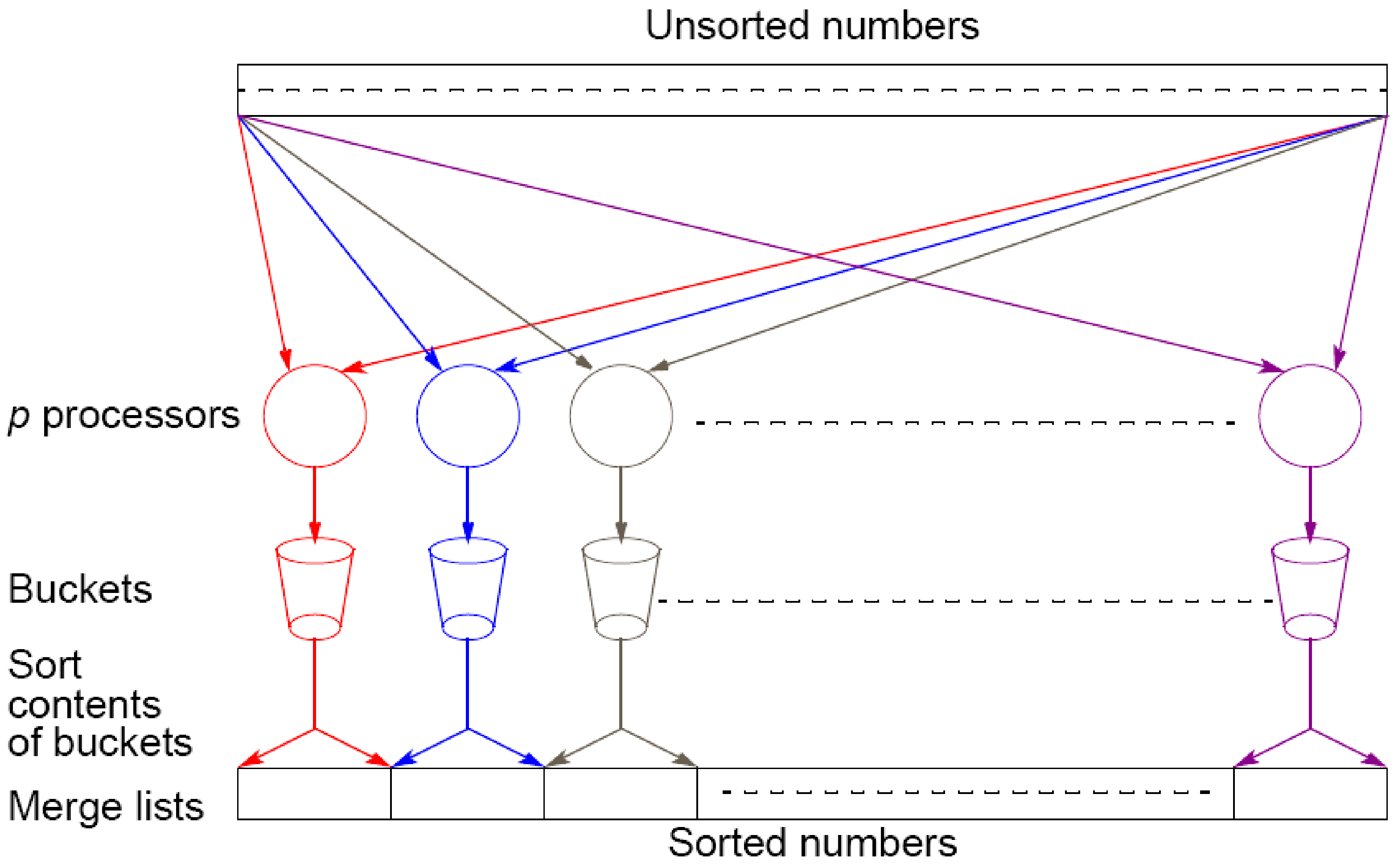


Sequential sorting time complexity: O($n$log(n/m)).
Works well if the original numbers uniformly distributed across a known interval, say 0 to a – 1.
Guy: O($n$log(n/m)) because: O(m·(n/m)log(n/m))

Parallel version of bucket sort

Simple approach

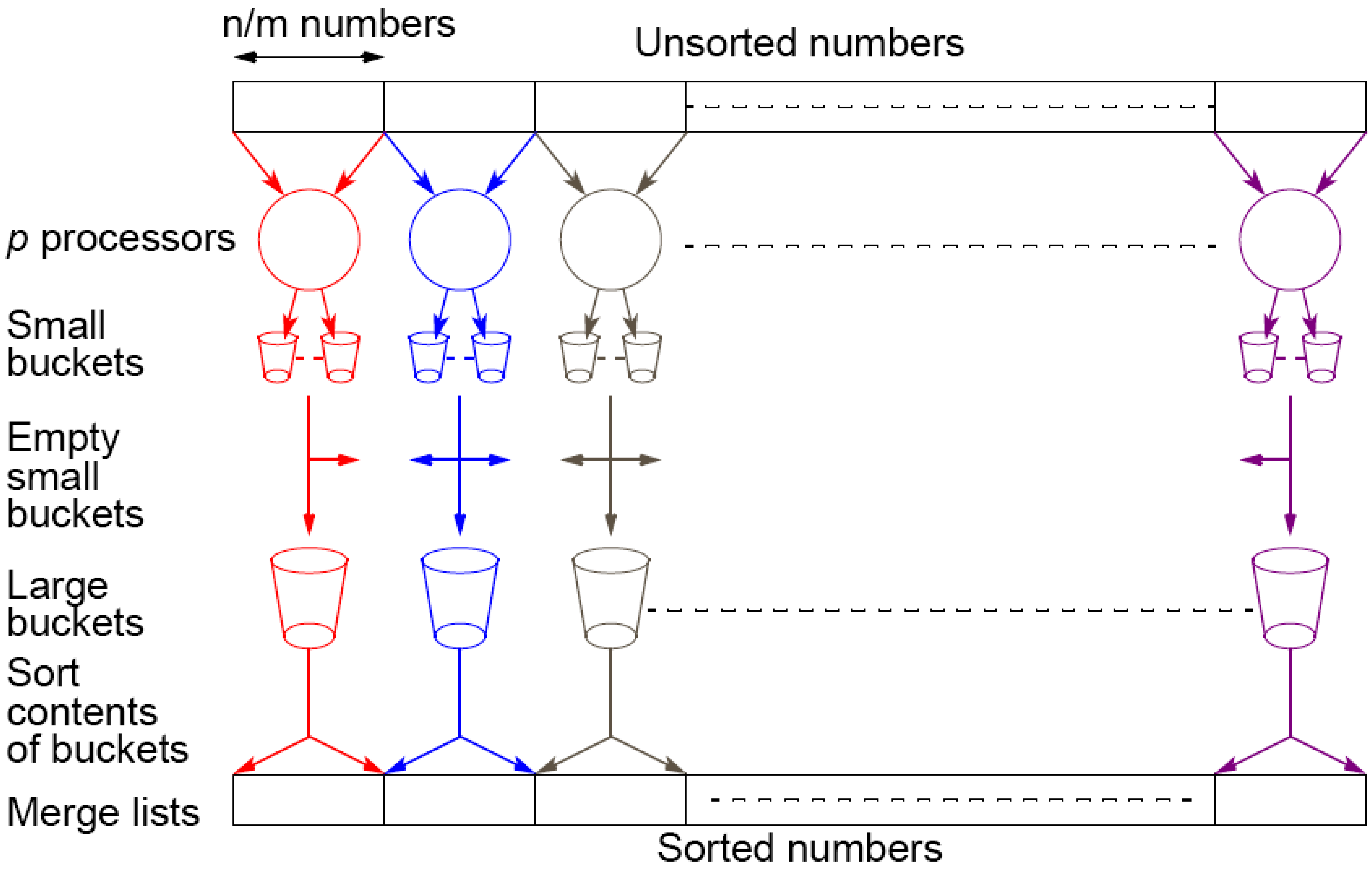
Assign one processor for each bucket.



Further Parallelization

- Partition sequence into *m* regions.
- Each processor assigned one region.
  (Hence number of processors, *p*, equals *m*.)
- Each processor maintains one “big” bucket for its region.
- Each processor maintains *m* “small” buckets, one for each region.
- Each processor separates numbers in its region into its own small buckets.
- All small buckets emptied into *p* big buckets
- Each big bucket sorted by its processor

Another parallel version of bucket sort

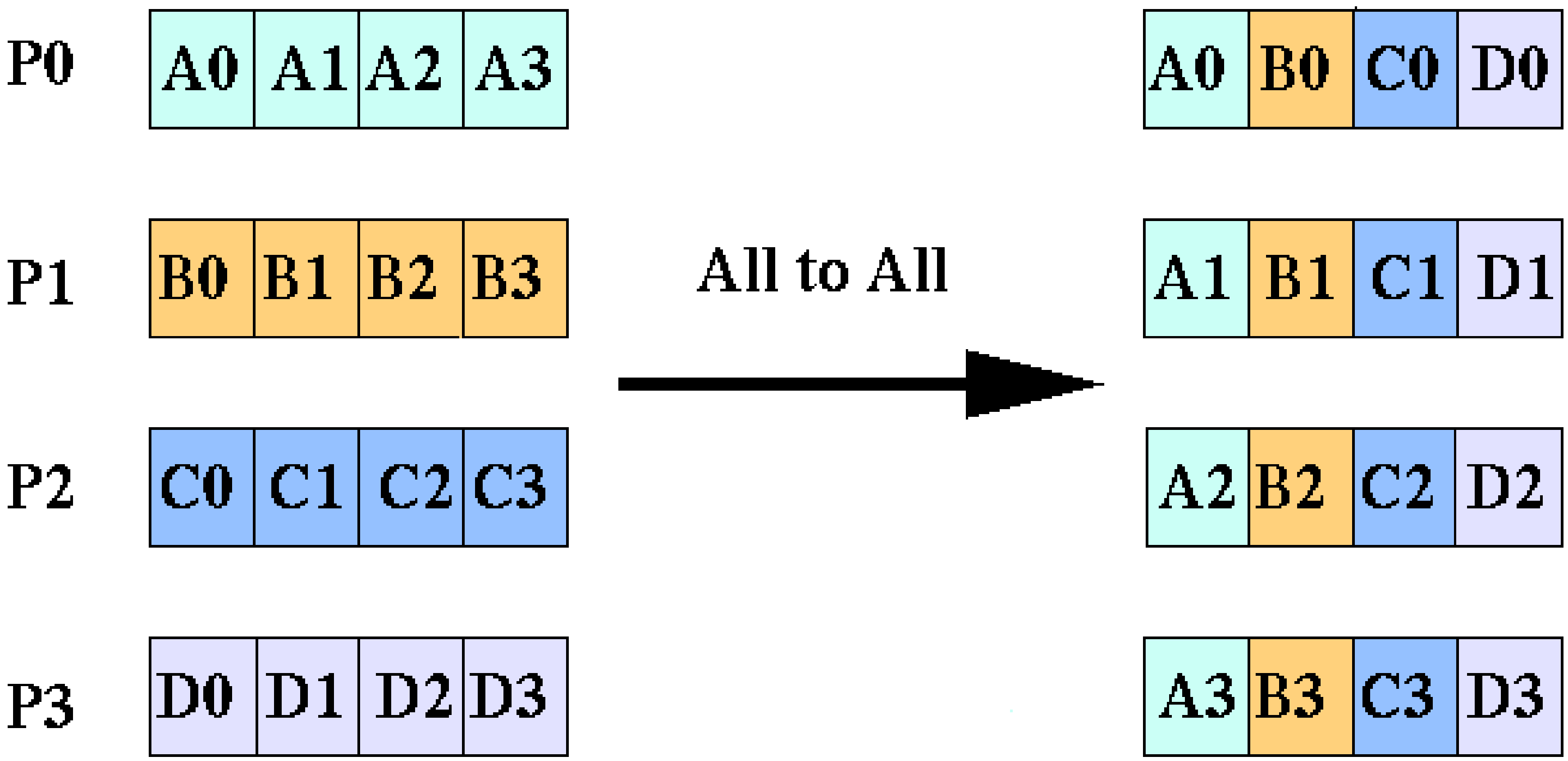


all-to-all broadcast

- An MPI function that can be used to advantage here.

- Sends one data element from each process to every other process.

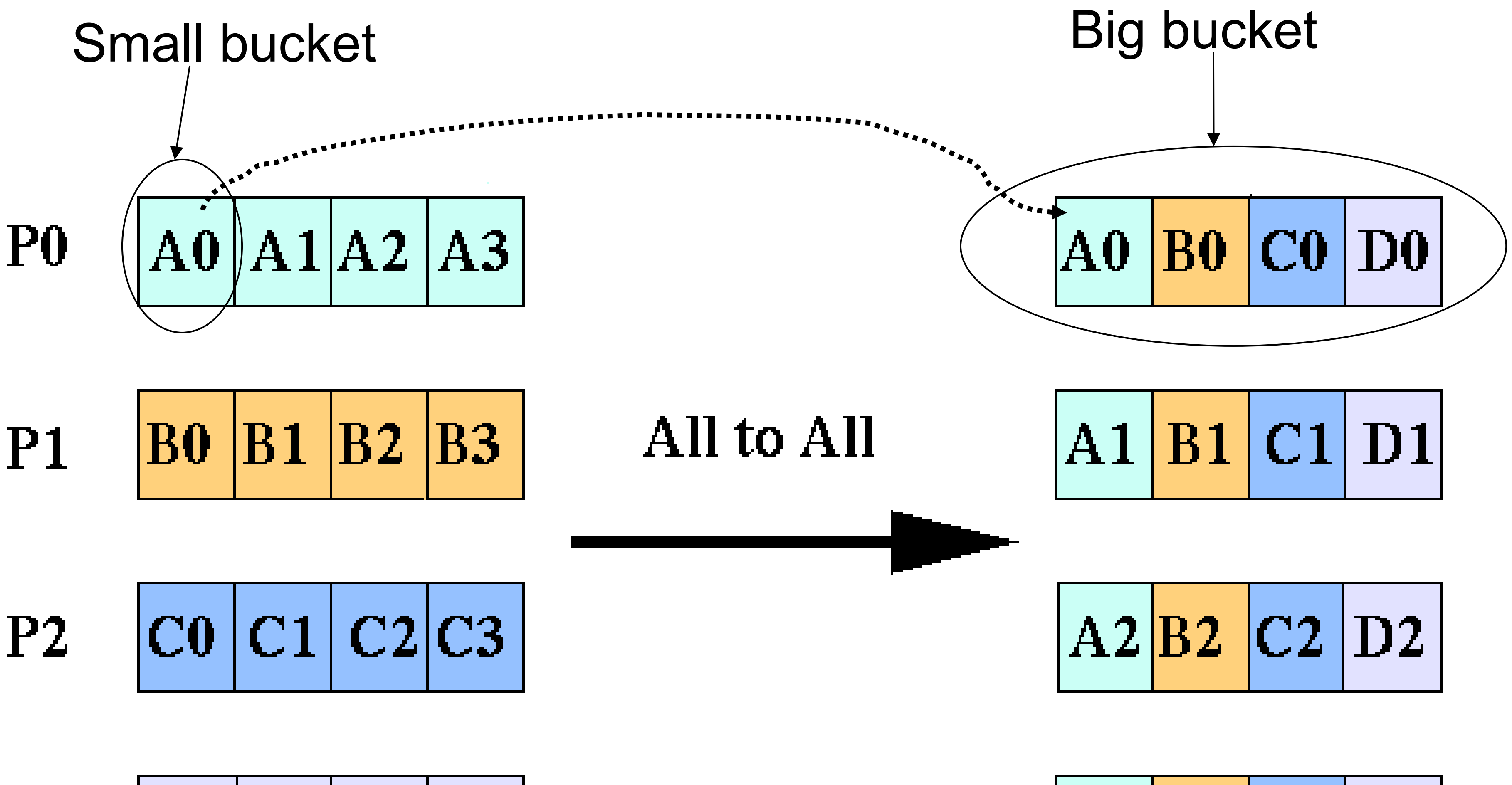
- Corresponds to multiple scatter operations, but implemented together.



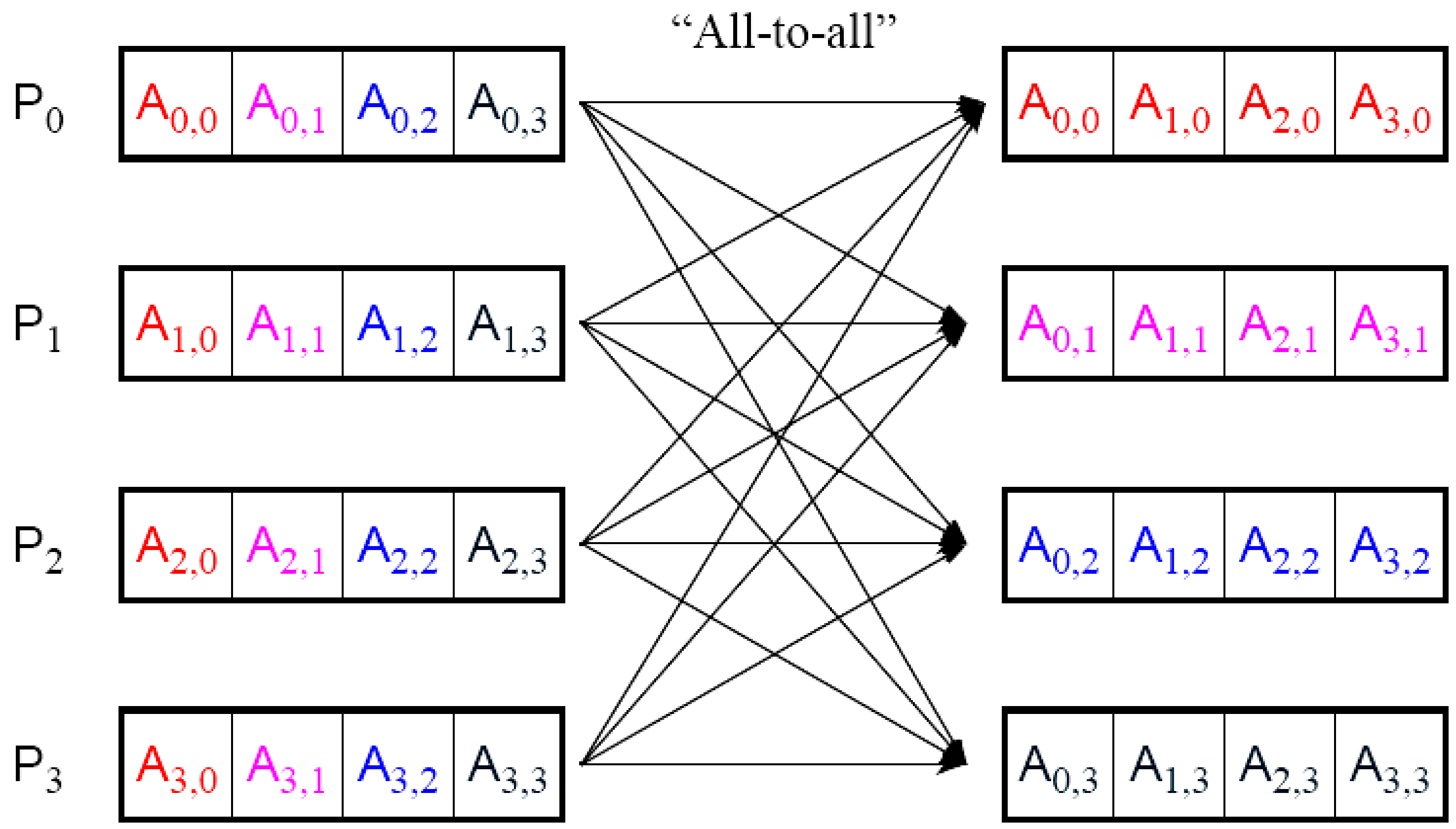
From: http://www.mhpcc.edu/training/workshop/parallel_intro/MAIN.html

Applying “all-to-all” broadcast to emptying small buckets into big buckets

Suppose 4 regions and 4 processors



“all-to-all” routine actually transfers rows of an array to columns: Transposes a matrix.


“All-to-all”
P_0
P_1
P_2
P_3
$A_{0,0}$ $A_{0,1}$ $A_{0,2}$ $A_{0,3}$
$A_{1,0}$ $A_{1,1}$ $A_{1,2}$ $A_{1,3}$
$A_{2,0}$ $A_{2,1}$ $A_{2,2}$ $A_{2,3}$
$A_{3,0}$ $A_{3,1}$ $A_{3,2}$ $A_{3,3}$
$A_{0,0}$ $A_{1,0}$ $A_{2,0}$ $A_{3,0}$
$A_{0,1}$ $A_{1,1}$ $A_{2,1}$ $A_{3,1}$
$A_{0,2}$ $A_{1,2}$ $A_{2,2}$ $A_{3,2}$
$A_{0,3}$ $A_{1,3}$ $A_{2,3}$ $A_{3,3}$

MPI_Alltoall

Sends data from all to all processes

Synopsis

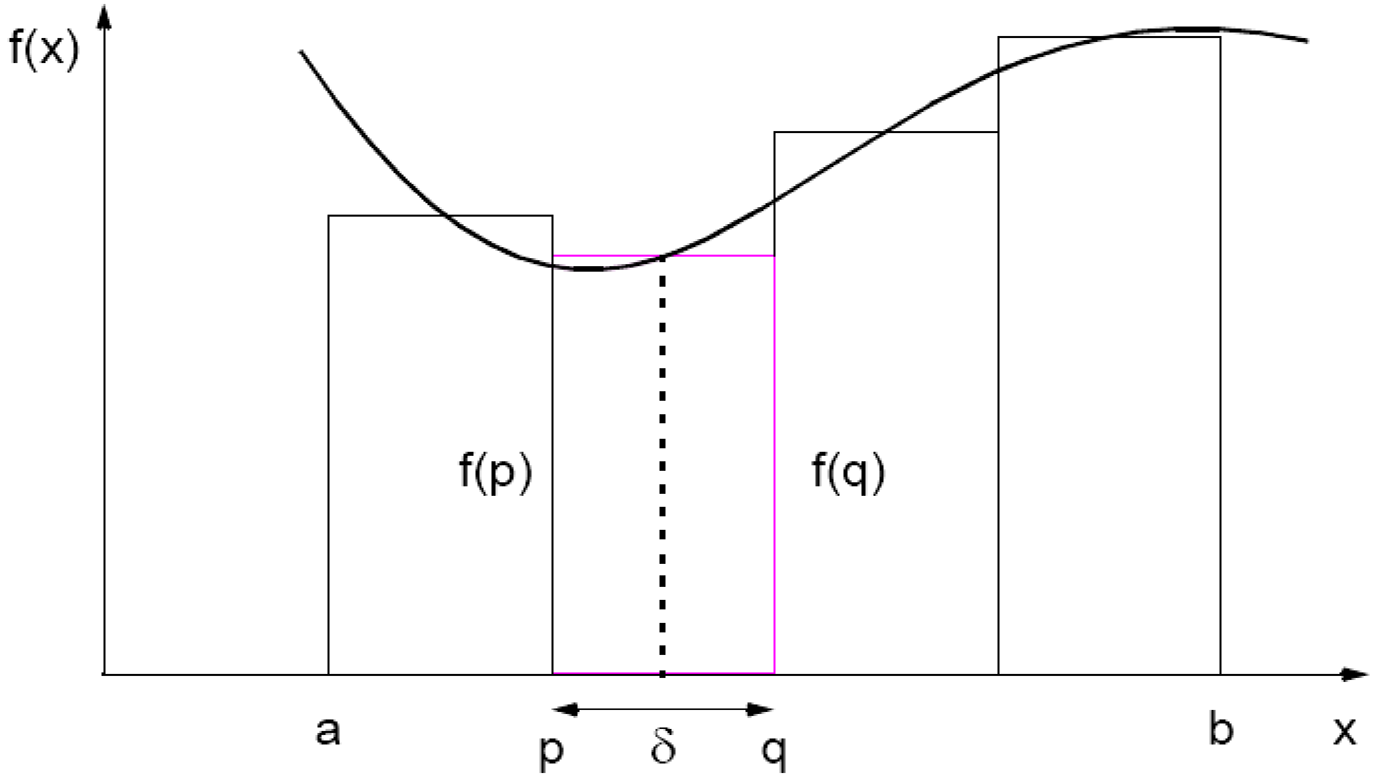
```
int MPI_Alltoall(*sendbuf, int sendcount, MPI_Datatype sendtype,
                 *recvbuf, int recvcount, MPI_Datatype recvtype,
                 MPI_Comm comm)
```

Numerical Integration

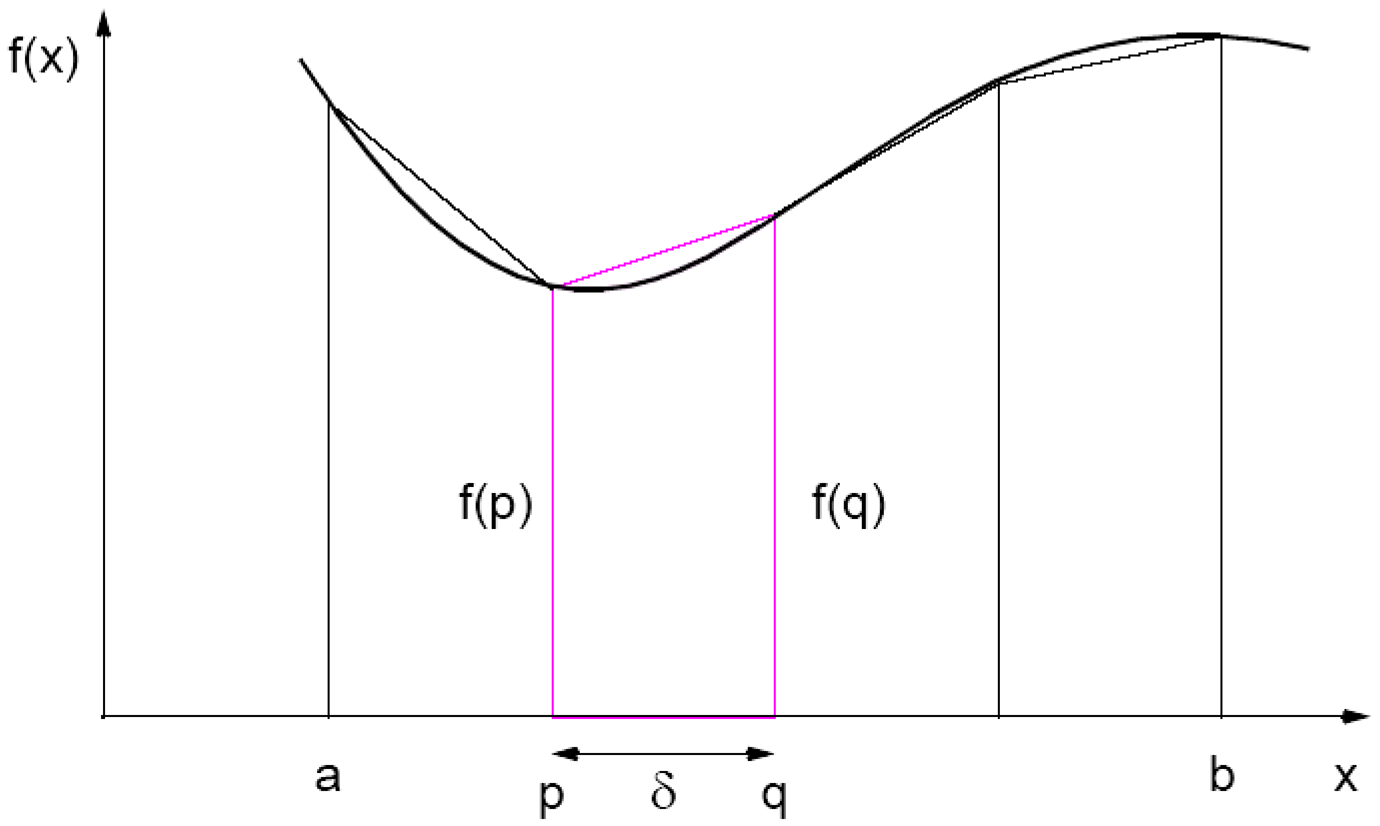
- Computing the “area under the curve.”
- Parallelized by divided area into smaller areas (partitioning).
- Can also apply divide and conquer -- repeatedly divide the areas recursively.

Numerical integration using rectangles

Each region calculated using an approximation given by rectangles:
Aligning the rectangles:



Numerical integration using trapezoidal method

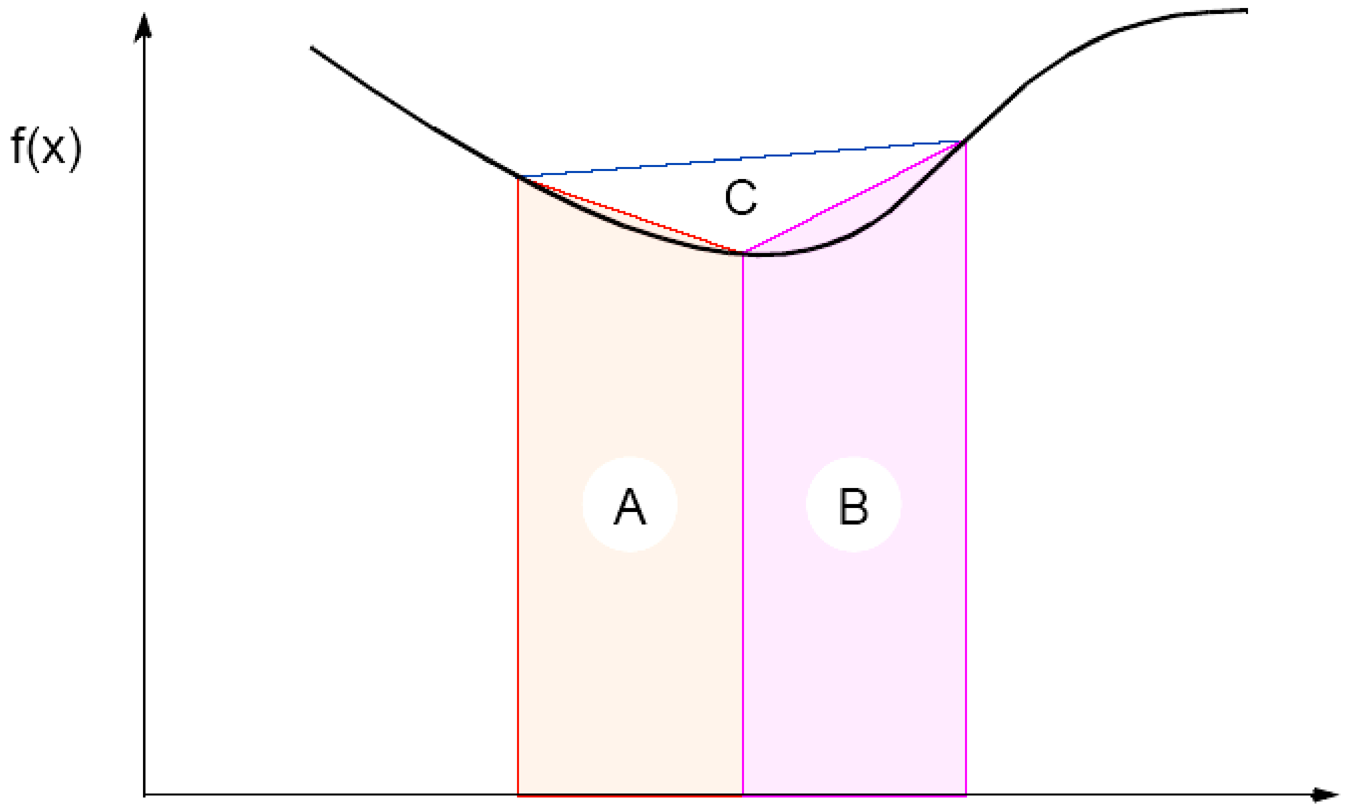


May not be better!

Applying Divide and Conquer

Adaptive Quadrature

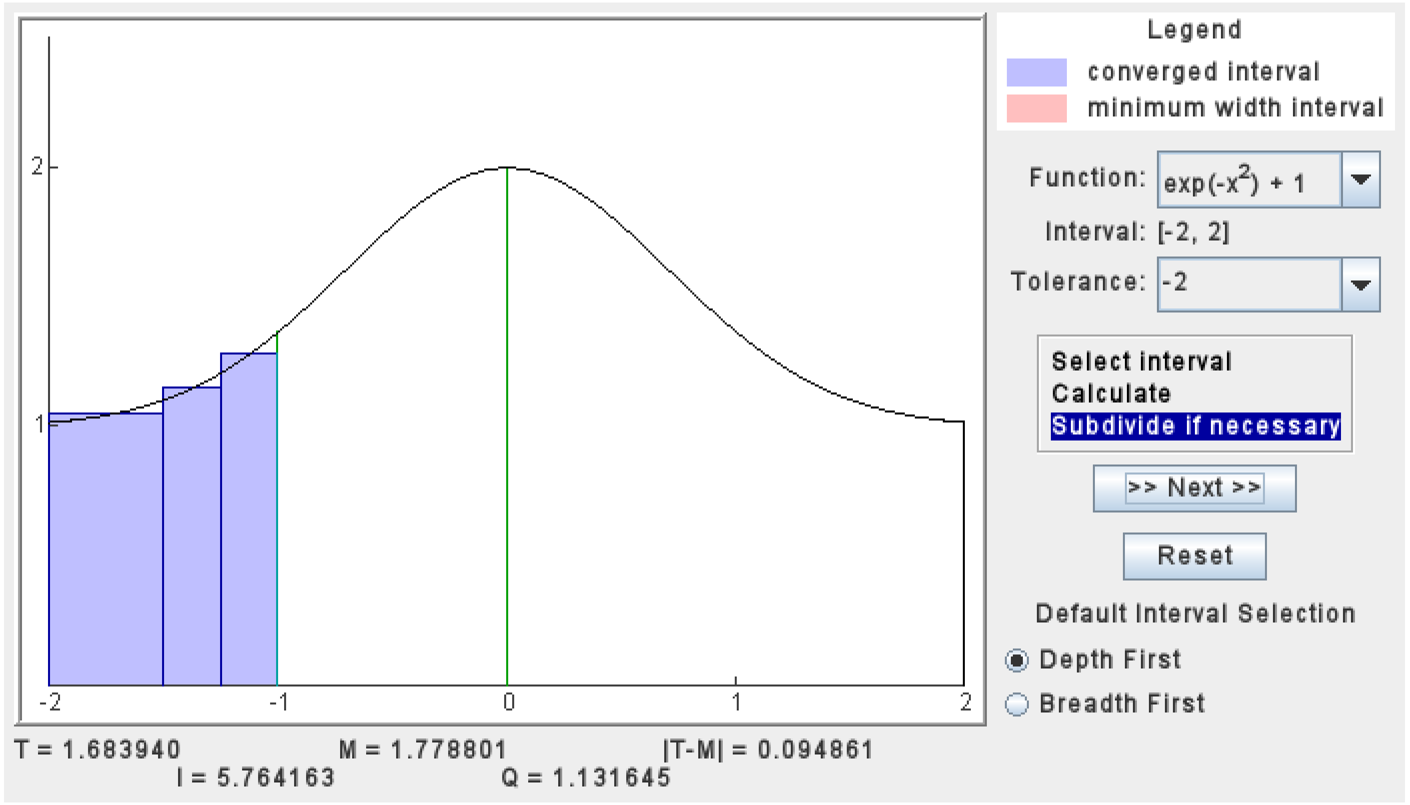
Solution adapts to shape of curve. Use three areas, *A*, *B*, and *C*. Computation terminated when largest of *A* and *B* sufficiently close to sum of remain two areas .



demo

Check this:

http://heath.cs.illinois.edu/iem/integration/adaptivq/



Function: exp(-x^2) + 1

Interval: [-2, 2]

Tolerance: -2

Default Interval Selection

[✓ Depth First]

Breadth First

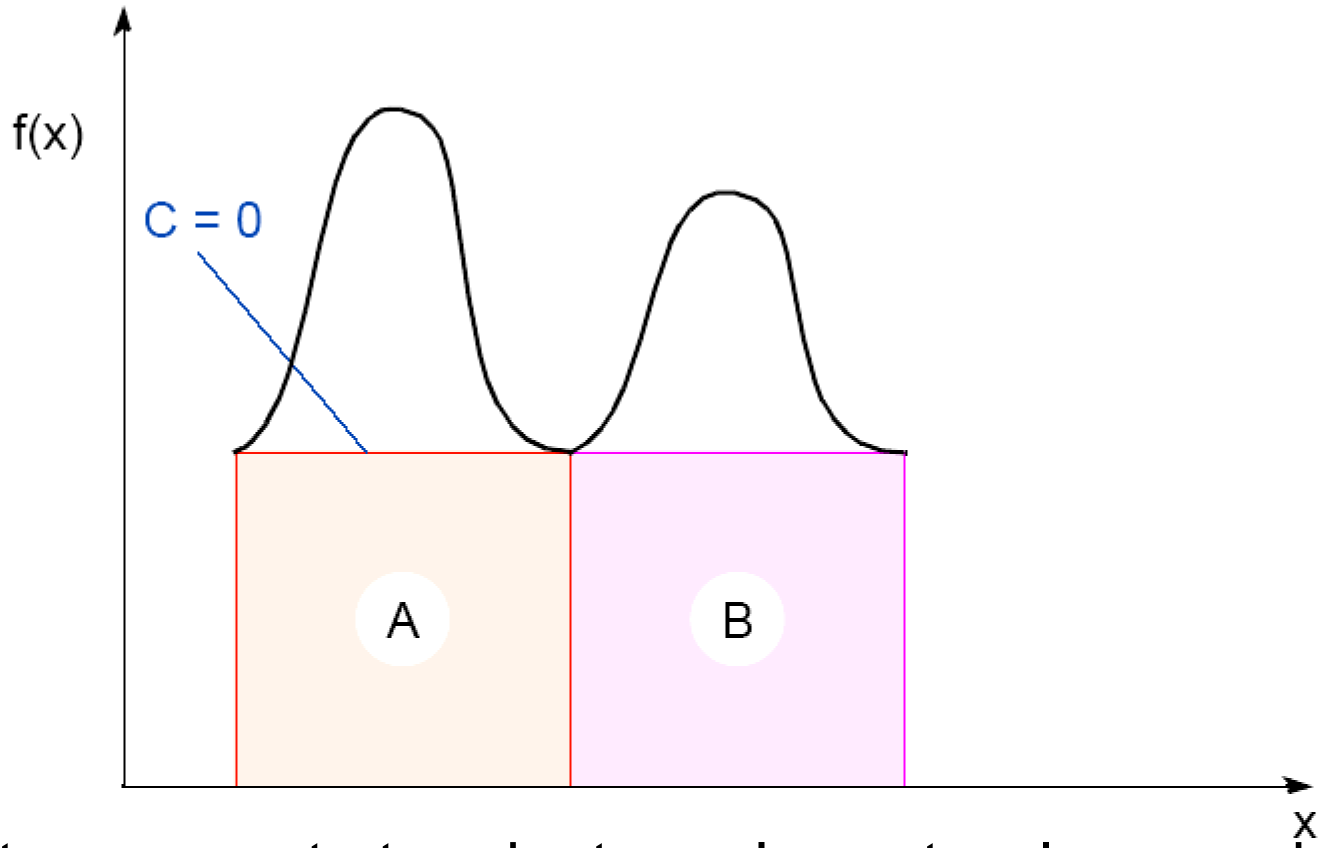
```
T = 1.683940              M = 1.778801              |T-M| = 0.094861
               I = 5.764163               Q = 1.131645
```

T=Trapeze
M=Midpoint
I=Exact
Q=Cumulative

Tolerance:
2 means 10^-2

Adaptive quadrature with false termination.

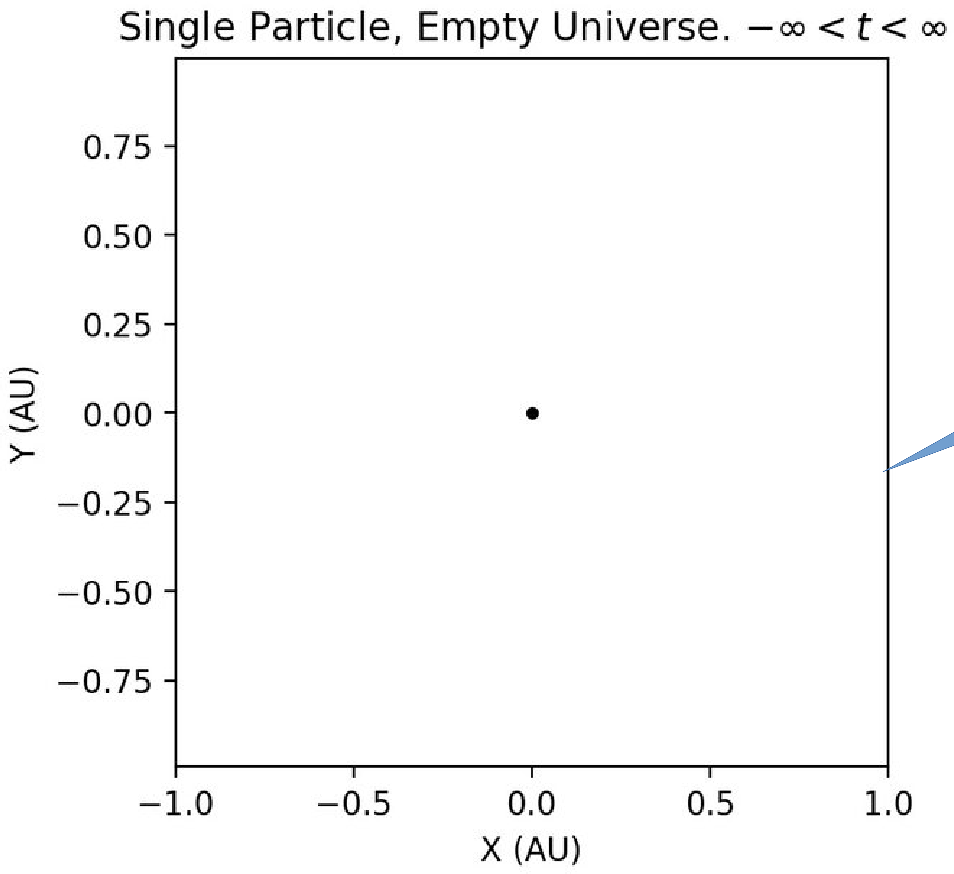
Some care might be needed in choosing when to terminate.



Might cause us to terminate early, as two large regions are the same (i.e., *C* = 0).

Gravitational *N*-Body Problem

Finding positions and movements of bodies in space subject to gravitational forces from other bodies, using Newtonian laws of physics.



זה בהומור כמובן.
חלקיק אחד ביקום ריק...

https://twitter.com/YoniAstro/status/1461179823674601474?s=20

Gravitational *N*-Body Problem Equations

Gravitational force between two bodies of masses m_a and m_b is:

$$F = \frac{G m_a m_b}{r^2}$$

G is the gravitational constant and *r* the distance between the bodies. Subject to forces, body accelerates according to Newton’s 2nd law:

$$F = ma$$

m is mass of the body, *F* is force it experiences, and *a* the resultant acceleration.

Details

Let the time interval be Δt. For a body of mass *m*, the force is:

$$F = \frac{m(v^{t+1} - v^{t})}{\Delta t}$$

New velocity is:

$$v^{t+1} = v^{t} + \frac{F\Delta t}{m}$$

where v^{t+1} is the velocity at time $t + 1$ and v^{t} is the velocity at time *t*.

Over time interval Δt, position changes by

$$x^{t+1} - x^{t} = v\Delta t$$

where x^{t} is its position at time *t*.
Once bodies move to new positions, forces change.
Computation has to be repeated.

Sequential Code

Overall gravitational *N*-body computation can be described by:

```
for (t = 0; t < tmax; t++) {            /* for each time period */
     for (i = 0; i < nmax; i++) {        /* for each body */
       F = Force_routine(i);             /* compute force on ith body */
       v[i]new = v[i] + F * dt / m;      /* compute new velocity */
       x[i]new = x[i] + v[i]new * dt;    /* and new position */
}
for (i = 0; i < nmax; i++) {             /* for each body */
     x[i] = x[i]new;                     /* update velocity & position*/
     v[i] = v[i]new;
}
} // end time loop
```

Parallel Code

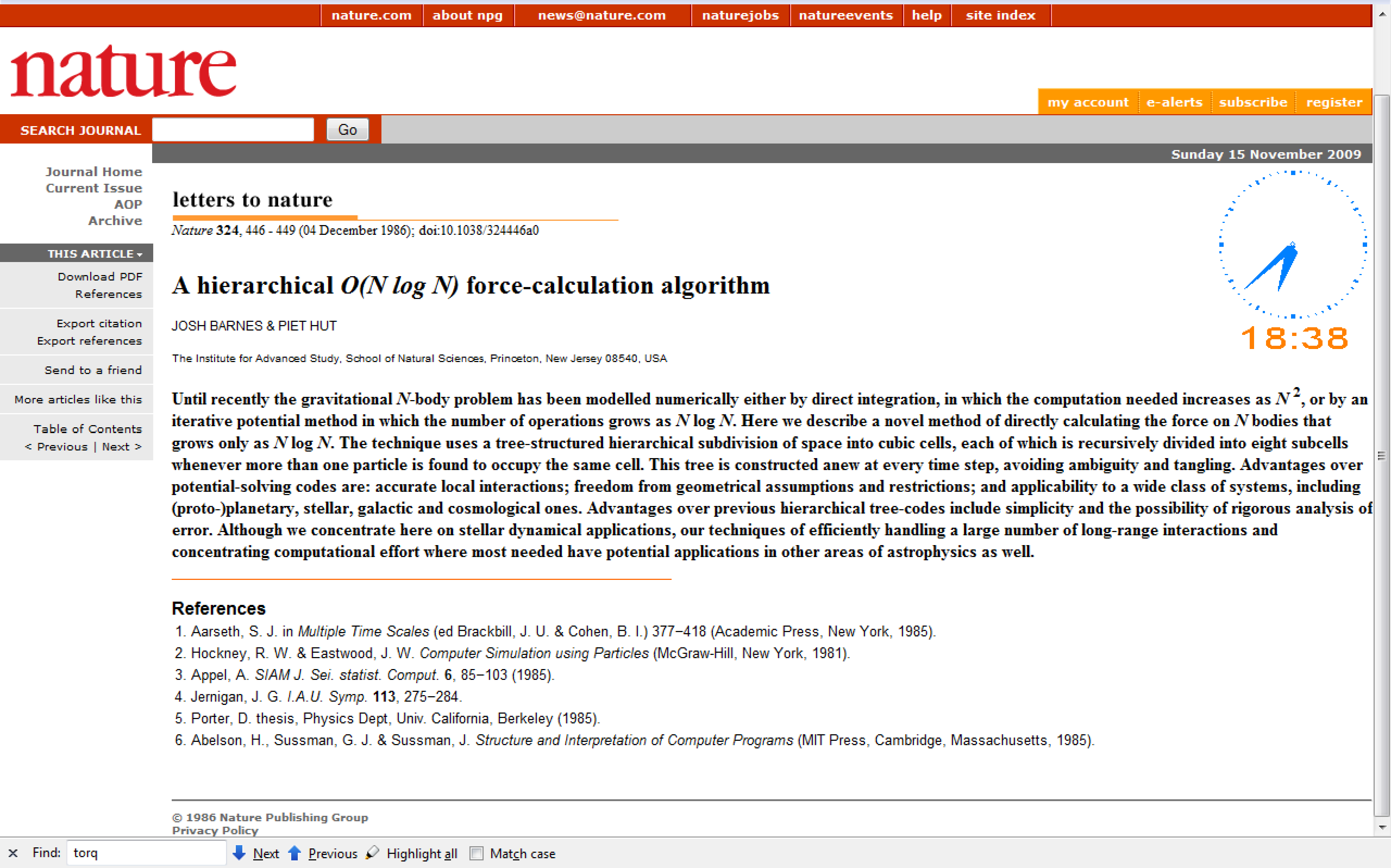
The sequential algorithm is an $O(N^2)$ algorithm (for one iteration) as each of the N bodies is influenced by each of the other $N - 1$ bodies.

Not feasible to use this direct algorithm for most interesting N-body problems where N is very large.

- http://www.nature.com/nature/journal/v324/n6096/abs/324446a0.html

(Guy)

Josh Barnes & Piet Hut



Sunday 15 November 2009

letters to nature

Nature **324**, 446 - 449 (04 December 1986); doi:10.1038/324446a0

A hierarchical *O(N log N)* force-calculation algorithm

JOSH BARNES & PIET HUT

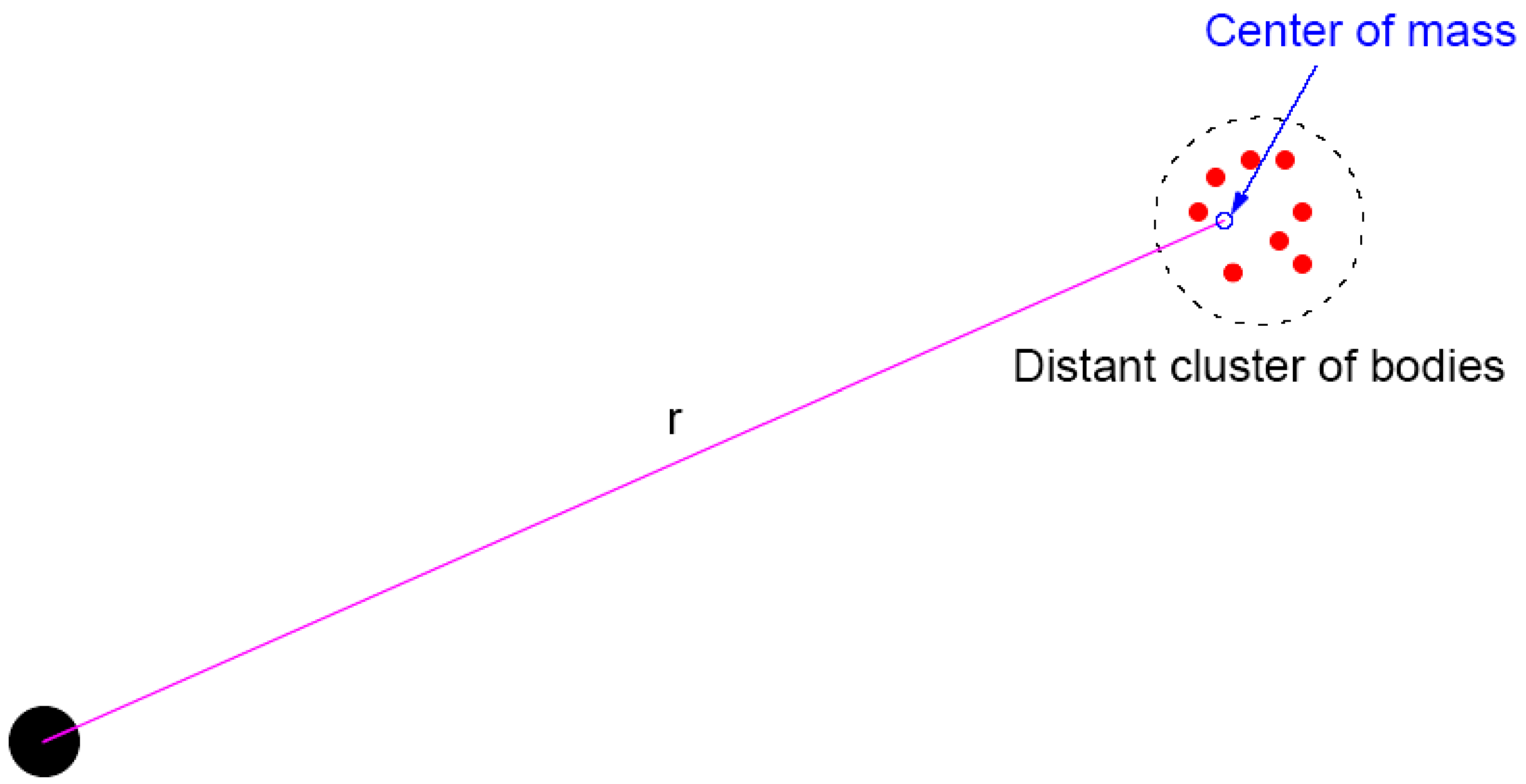
The Institute for Advanced Study, School of Natural Sciences, Princeton, New Jersey 08540, USA

Until recently the gravitational N-body problem has been modelled numerically either by direct integration, in which the computation needed increases as N^2, or by an iterative potential method in which the number of operations grows as $N \log N$. Here we describe a novel method of directly calculating the force on N bodies that grows only as $N \log N$. The technique uses a tree-structured hierarchical subdivision of space into cubic cells, each of which is recursively divided into eight subcells whenever more than one particle is found to occupy the same cell. This tree is constructed anew at every time step, avoiding ambiguity and tangling. Advantages over potential-solving codes are: accurate local interactions; freedom from geometrical assumptions and restrictions; and applicability to a wide class of systems, including (proto-)planetary, stellar, galactic and cosmological ones. Advantages over previous hierarchical tree-codes include simplicity and the possibility of rigorous analysis of error. Although we concentrate here on stellar dynamical applications, our techniques of efficiently handling a large number of long-range interactions and concentrating computational effort where most needed have potential applications in other areas of astrophysics as well.

References

1. Aarseth, S. J. in *Multiple Time Scales* (ed Brackbill, J. U. & Cohen, B. I.) 377–418 (Academic Press, New York, 1985).
2. Hockney, R. W. & Eastwood, J. W. *Computer Simulation using Particles* (McGraw-Hill, New York, 1981).
3. Appel, A. *SIAM J. Sei. statist. Comput.* **6**, 85–103 (1985).
4. Jernigan, J. G. *I.A.U. Symp.* **113**, 275–284.
5. Porter, D. thesis, Physics Dept, Univ. California, Berkeley (1985).
6. Abelson, H., Sussman, G. J. & Sussman, J. *Structure and Interpretation of Computer Programs* (MIT Press, Cambridge, Massachusetts, 1985).

Time complexity can be reduced approximating a cluster of distant bodies as a single distant body with mass sited at the center of mass of the cluster:



Barnes-Hut Algorithm

Start with whole space in which one cube contains the bodies (or particles).

- First, this cube is divided into eight subcubes.
- If a subcube contains no particles, subcube deleted from further consideration.
- If a subcube contains one body, subcube retained.
- If a subcube contains more than one body, it is recursively divided until every subcube contains one body.

Creates an *octtree* - a tree with up to eight edges from each node.

The leaves represent cells each containing one body.

After the tree has been constructed, the total mass and center of mass of the subcube is stored at each node.

Force on each body obtained by traversing tree starting at root, stopping at a node when the clustering approximation can be used, e.g. when:

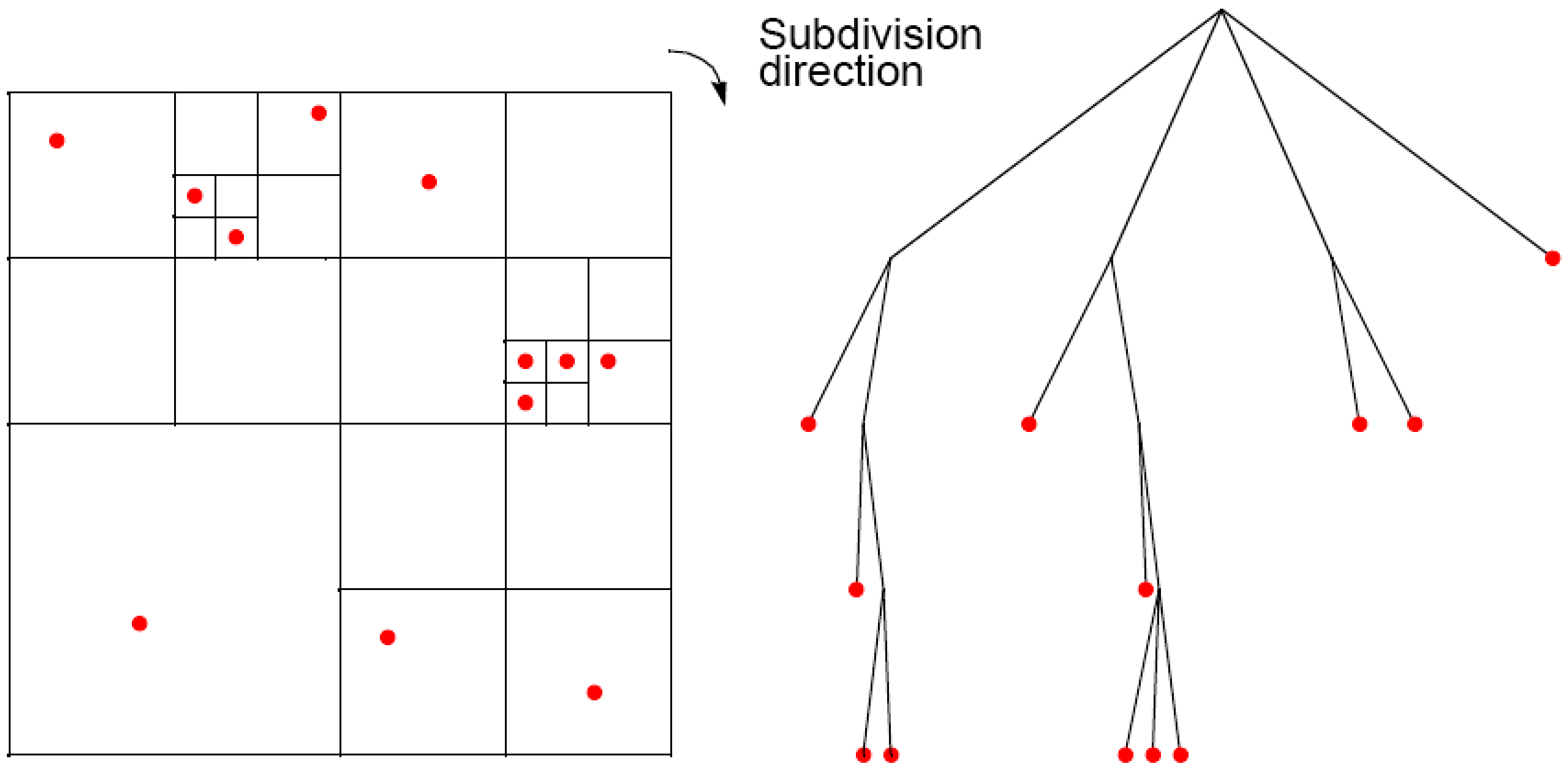
$$r \geq \frac{d}{\theta}$$

where θ is a constant typically 1.0 or less.

Constructing tree requires a time of O(*n*log*n*), and so does computing all the forces, so that overall time complexity of method is O(*n*log*n*).

r = distance to CoM, d = cluster diameter, Θ = scale

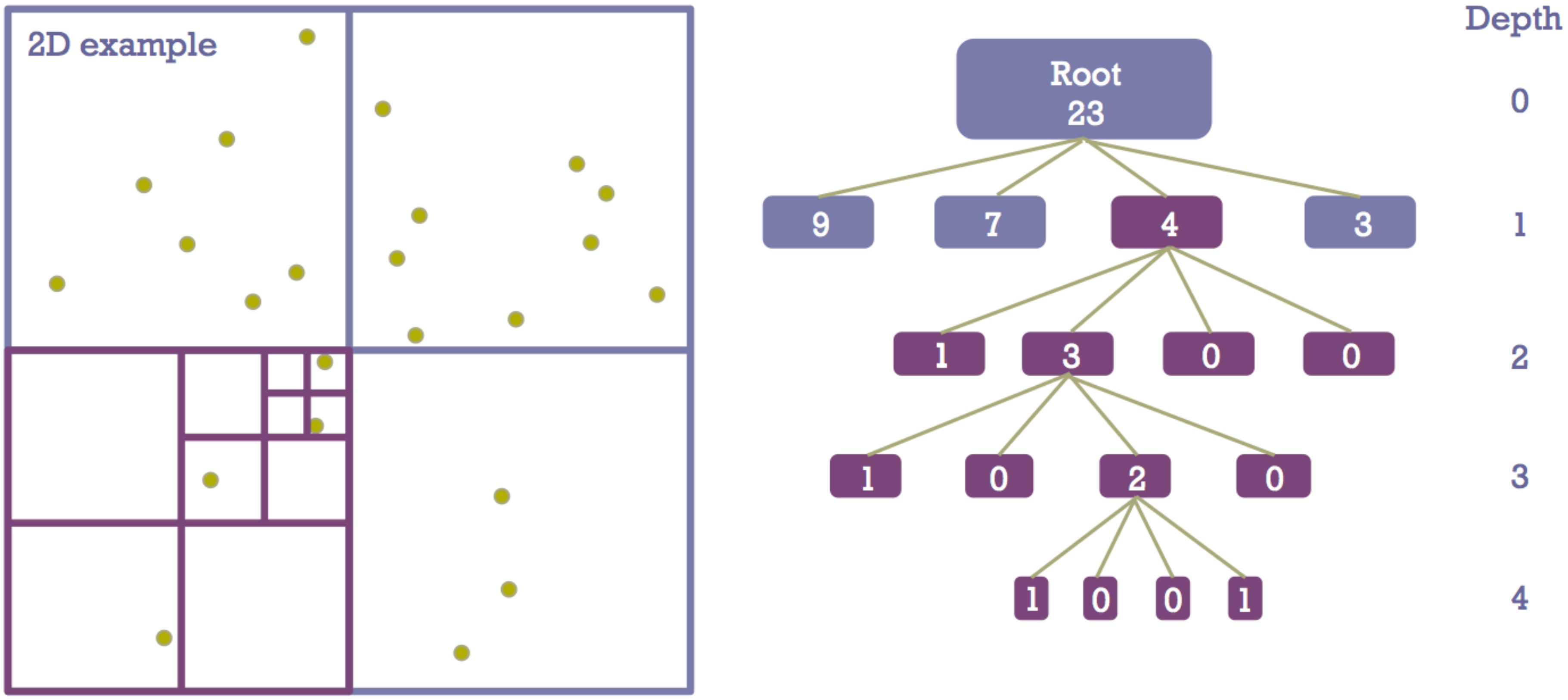
Recursive division of 2-dimensional space



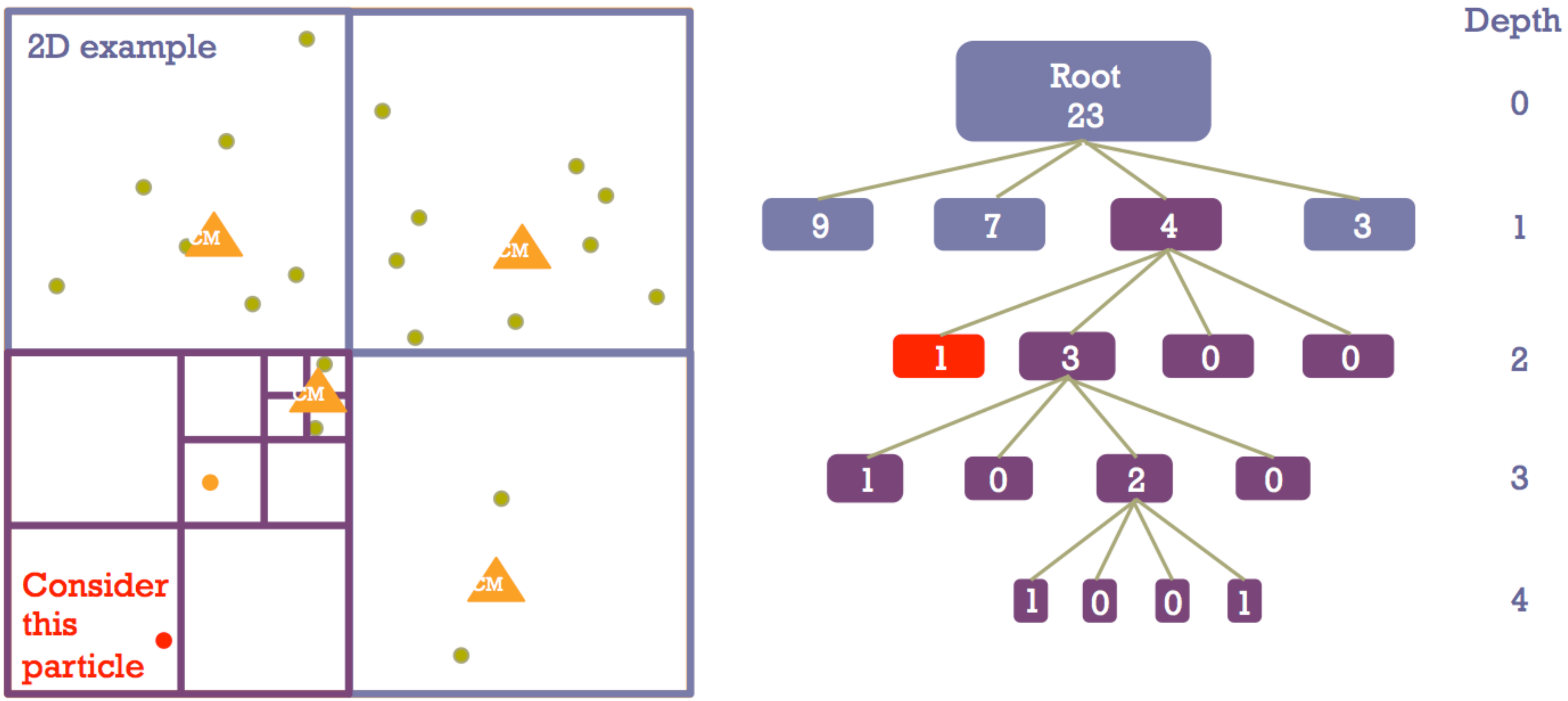
Particles

Partial quadtree

דוגמה נוספת



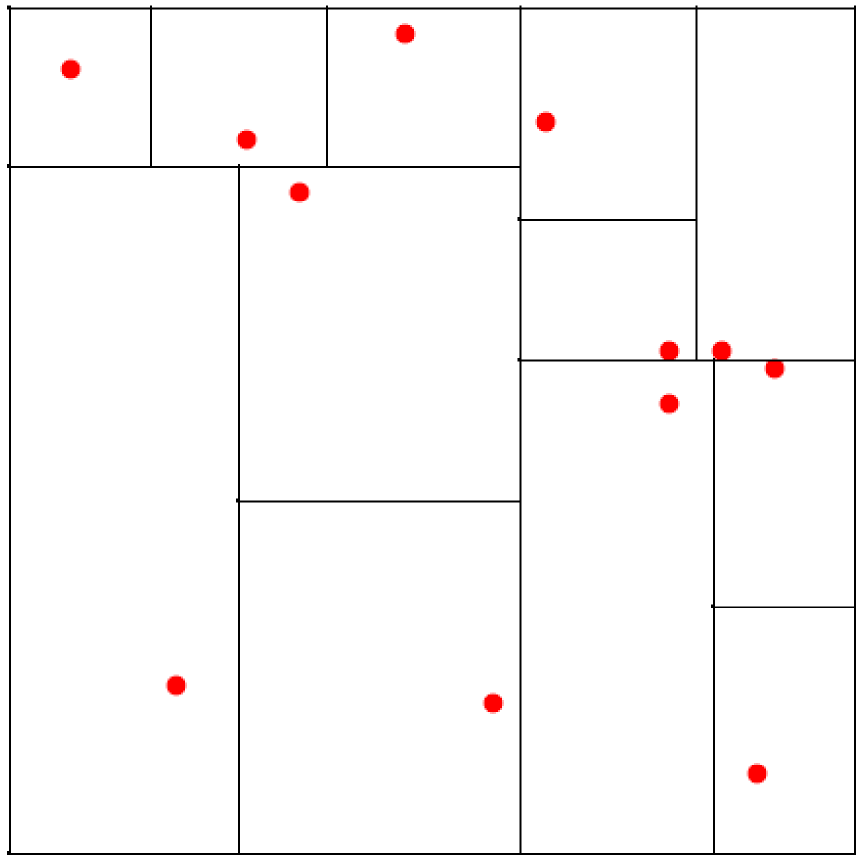
http://portillo.ca/nbody/wp-content/uploads/2013/12/Barnes-Hut_tree1.png



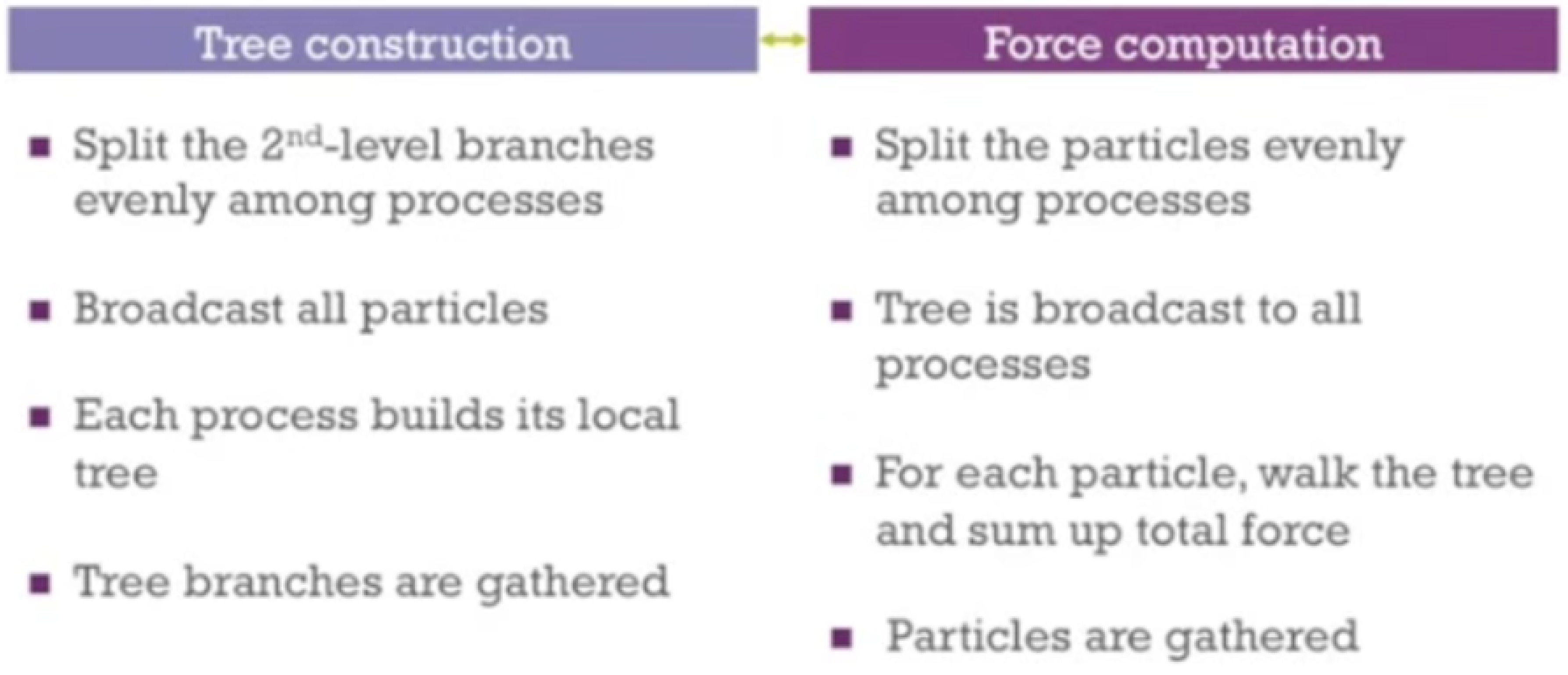
אם נשארנו עם כוכב אחד בתא השטח אז נחשב את הכוח מולו. אחרת נחצה את המרחב ונחשב CoM. אם קריטריון הזוית מתקיים נחשב את הכוח מול מרכז המסה אחרת נחצה את המרחב שוב וחוזר חלילה.

Orthogonal Recursive Bisection

(For 2-dimensional area) First, a vertical line found that divides area into two areas each with equal number of bodies. For each area, a horizontal line found that divides it into two areas each with equal number of bodies. Repeated as required.



איך ממקבלים את האלגוריתם?



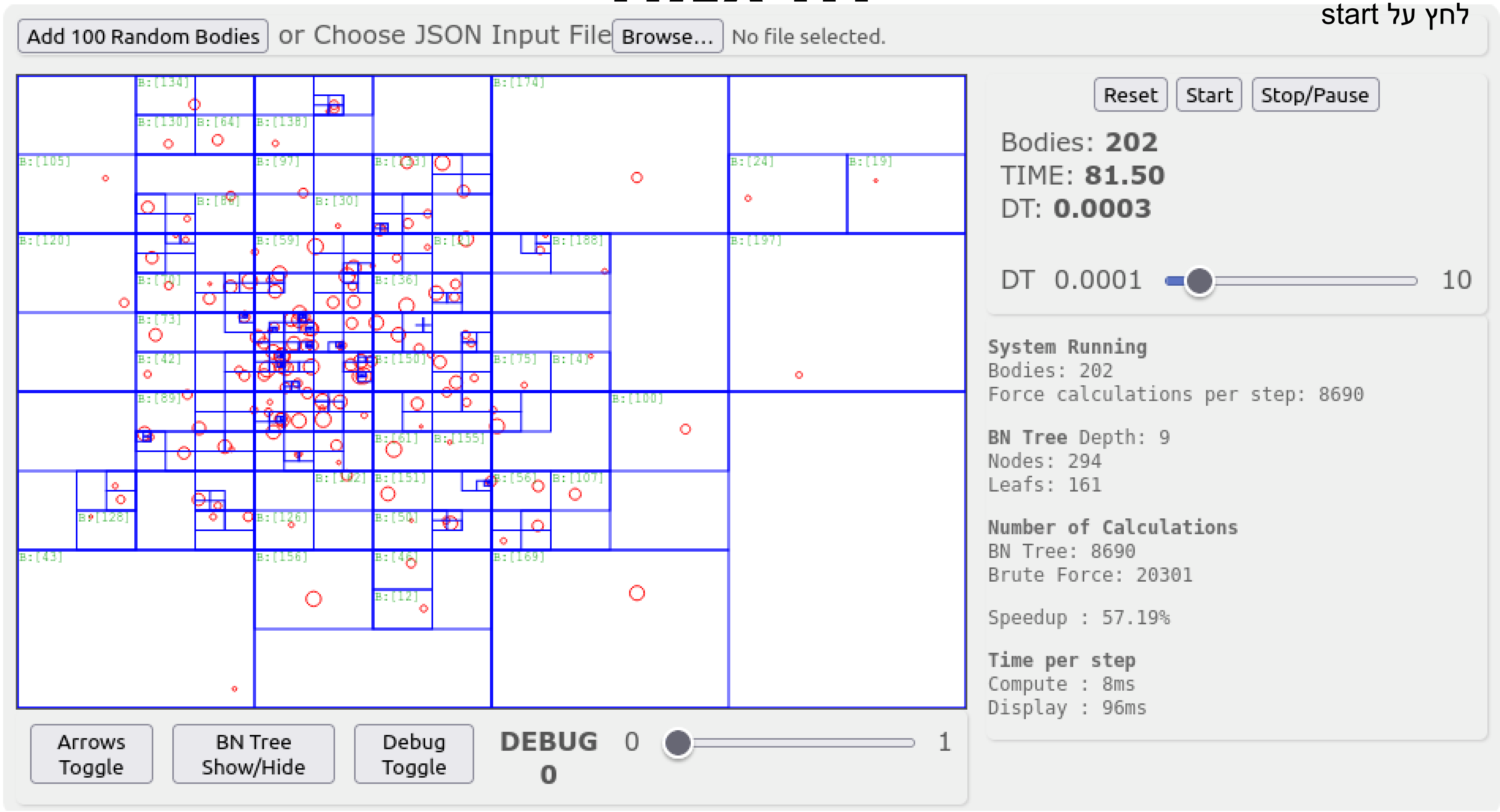
Tree construction	Force computation
▪ Split the 2nd-level branches evenly among processes	▪ Split the particles evenly among processes
▪ Broadcast all particles	▪ Tree is broadcast to all processes
▪ Each process builds its local tree	▪ For each particle, walk the tree and sum up total force
▪ Tree branches are gathered	▪ Particles are gathered

סרטון של 3 דקות:

https://www.youtube.com/watch?v=0eKQXPAcQK8

הדגמה

תזכורת לעצמי: נקה את המגרש. בעזרת העכבר עשה קליק היכן שתרצה חלקיק נוסף. באופן זה הוסף חלקיקים ולבסוף לחץ על start



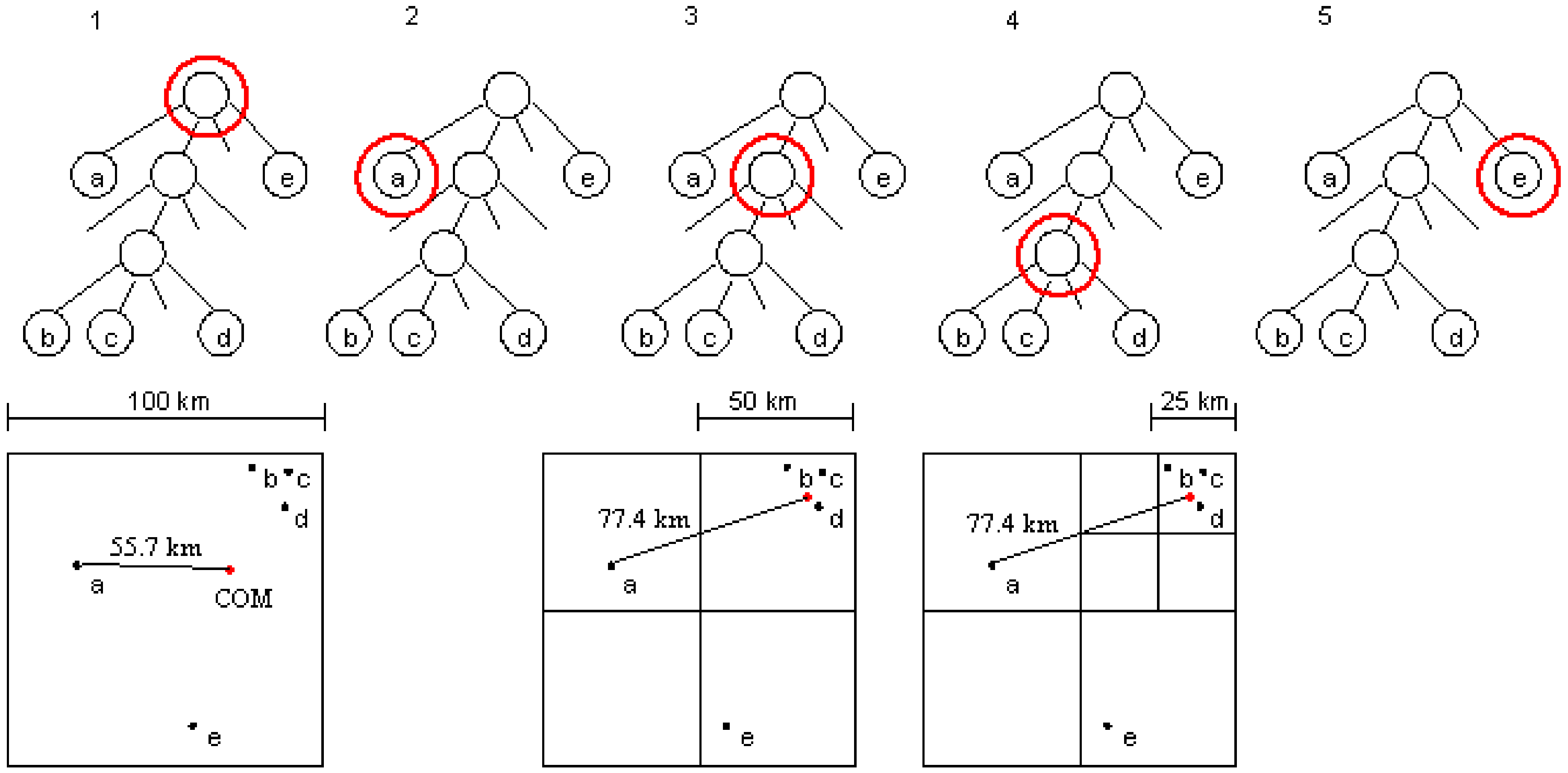
Bodies: **202**
TIME: **81.50**
DT: **0.0003**

```
System Running
Bodies: 202
Force calculations per step: 8690

BN Tree Depth: 9
Nodes: 294
Leafs: 161

Number of Calculations
BN Tree: 8690
Brute Force: 20301

Speedup : 57.19%

Time per step
Compute : 8ms
Display : 96ms
```

https://web.archive.org/web/20140413142523/http://www.andrew.cmu.edu/user/sameera/demos/BNtree/

- Guy

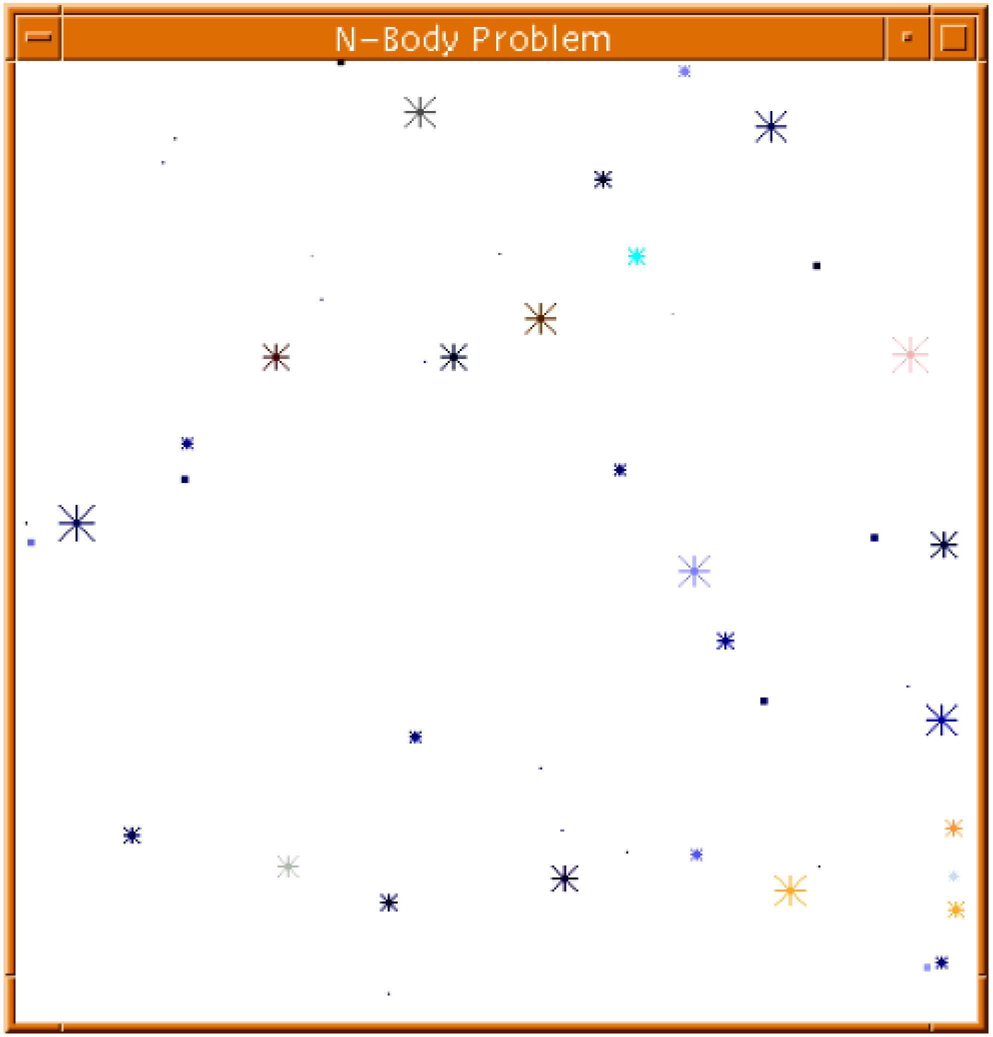
תהליך בניית ה-Quadtree:



Reference: http://www.cs.princeton.edu/courses/archive/fall04/cos126/assignments/barnes-hut.html

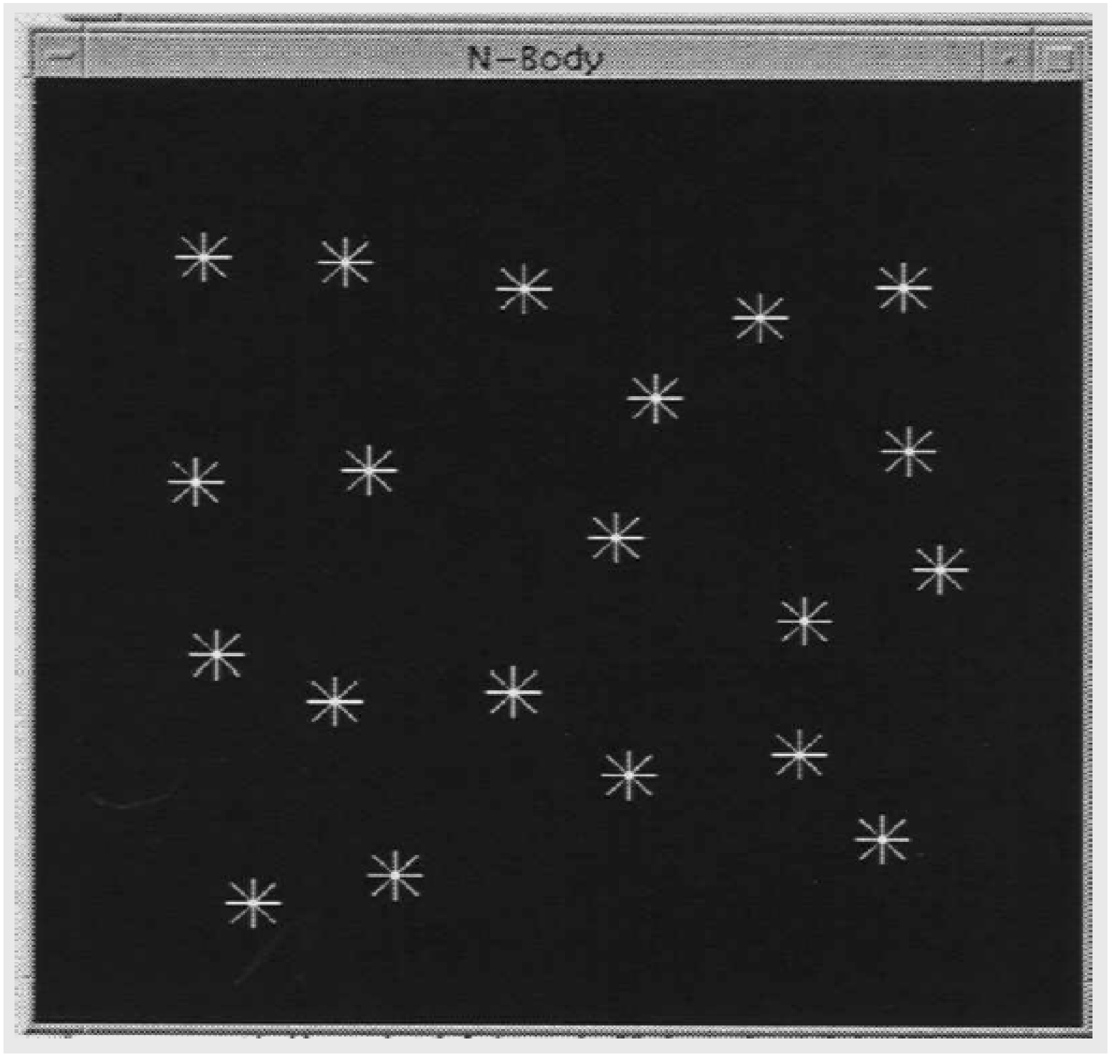
Astrophysical *N*-body simulation

By Scott Linssen (UNCC student, 1997) using O(N²) algorithm.

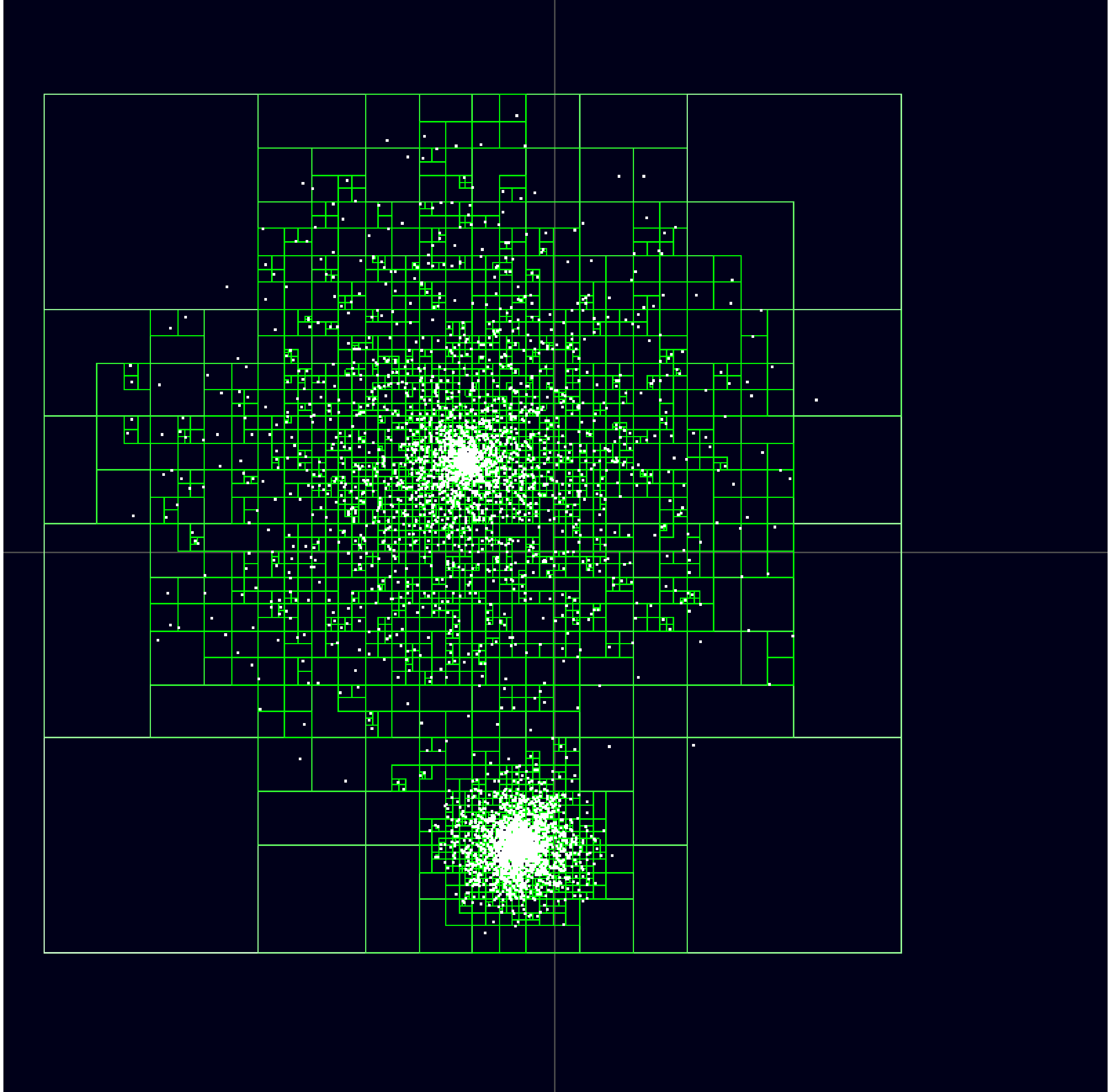


Astrophysical *N*-body simulation

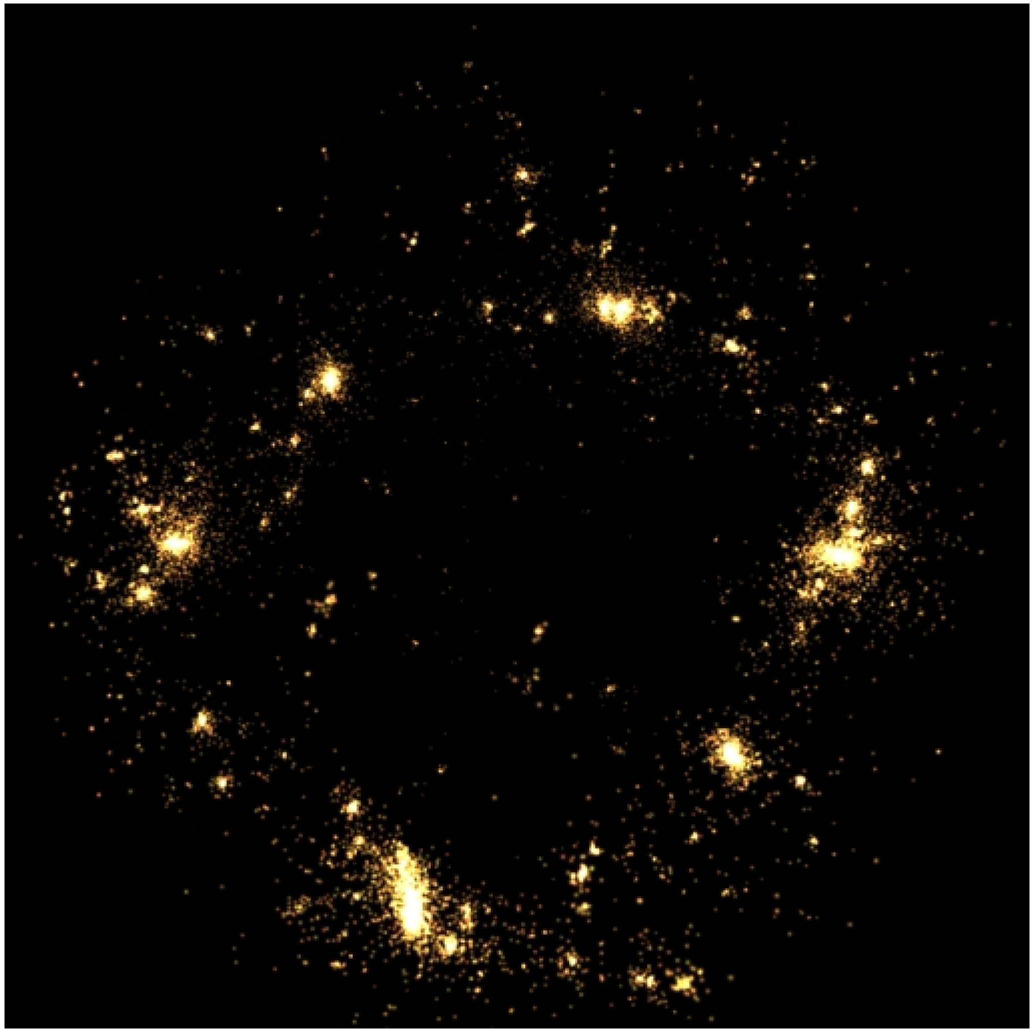
By David Messager (UNCC student 1998) using Barnes-Hut algorithm.



(Guy)



A demo on a GPU



Folder:
~/science/CUDA-10_Samples/5_Simulations/nbody

instructions

```
Run "nbody -benchmark [-numbodies=<numBodies>]" to measure performance.
    -fullscreen       (run n-body simulation in fullscreen mode)
    -fp64             (use double precision floating point values for
simulation)
    -hostmem          (stores simulation data in host memory)
    -benchmark        (run benchmark to measure performance)
    -numbodies=<N>    (number of bodies (>= 1) to run in simulation)
    -device=<d>       (where d=0,1,2.... for the CUDA device to use)
    -numdevices=<i>   (where i=(number of CUDA devices > 0) to use for
simulation)
    -compare          (compares simulation results running once on the
default GPU and once on the CPU)
    -cpu              (run n-body simulation on the CPU)
    -tipsy=<file.bin> (load a tipsy model file for simulation)


Exit the program with: q
```

In one terminal type: ./nbosy -fullscreen
In another terminal start: **`nvidia-smi -l`**