Designing Parallel Programs

Abhinav Bhatele, Department of Computer Science

UNIVERSITY OF MARYLAND

Writing parallel programs

Writing parallel programs

- Decide the serial algorithm first

Writing parallel programs

SPMD model

- Decide the serial algorithm first

Writing parallel programs

SPMD model

- Decide the serial algorithm first
- Data: how to distribute data among threads/processes?
  - Data locality: assignment of data to specific processes to minimize data movement

Writing parallel programs

SPMD model

- Decide the serial algorithm first
- Data: how to distribute data among threads/processes?
  - Data locality: assignment of data to specific processes to minimize data movement
- Computation: how to divide work among threads/processes?

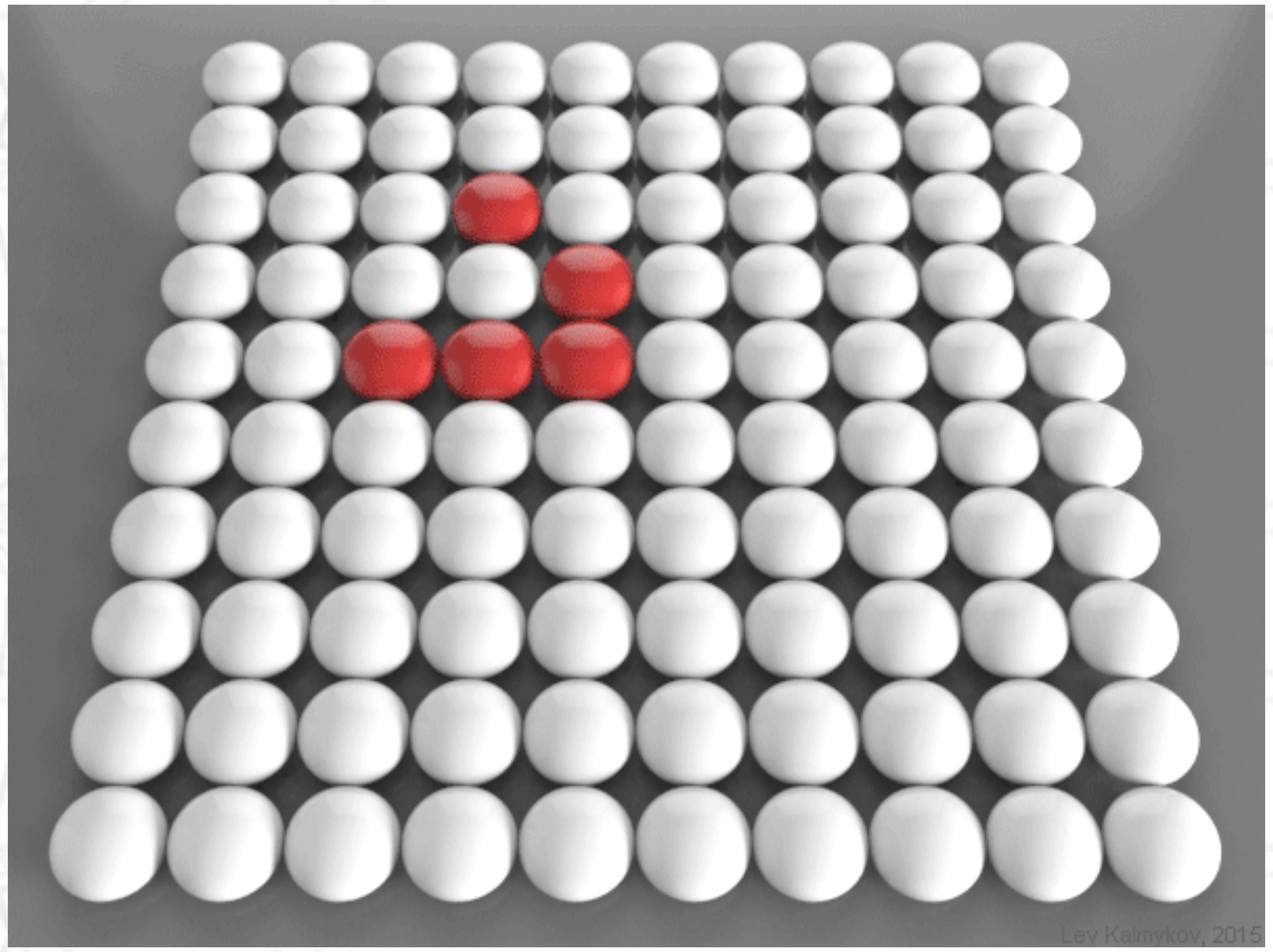
Writing parallel programs

SPMD model

- Decide the serial algorithm first
- Data: how to distribute data among threads/processes?
  - Data locality: assignment of data to specific processes to minimize data movement
- Computation: how to divide work among threads/processes?
- Figure out how often communication will be needed

Conway’s Game of Life

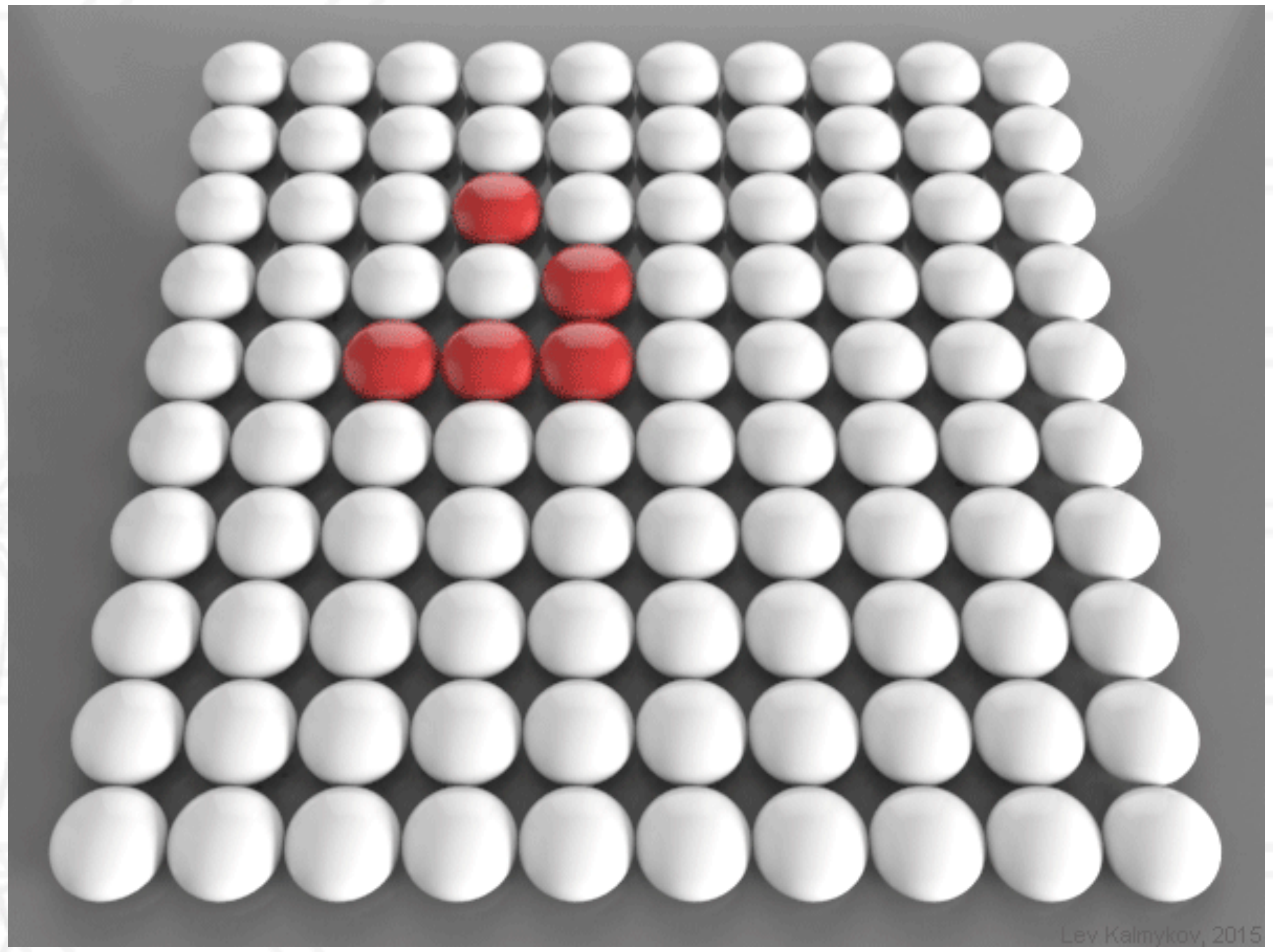
- Two-dimensional grid of (square) cells
- Each cell can be in one of two states: live or dead
- Every cell only interacts with its eight nearest neighbors
- In every generation (or iteration or time step), there are some rules that decide if a cell will continue to live or die or be born (dead ➜ live)



https://en.wikipedia.org/wiki/Conway's_Game_of_Life

Conway’s Game of Life

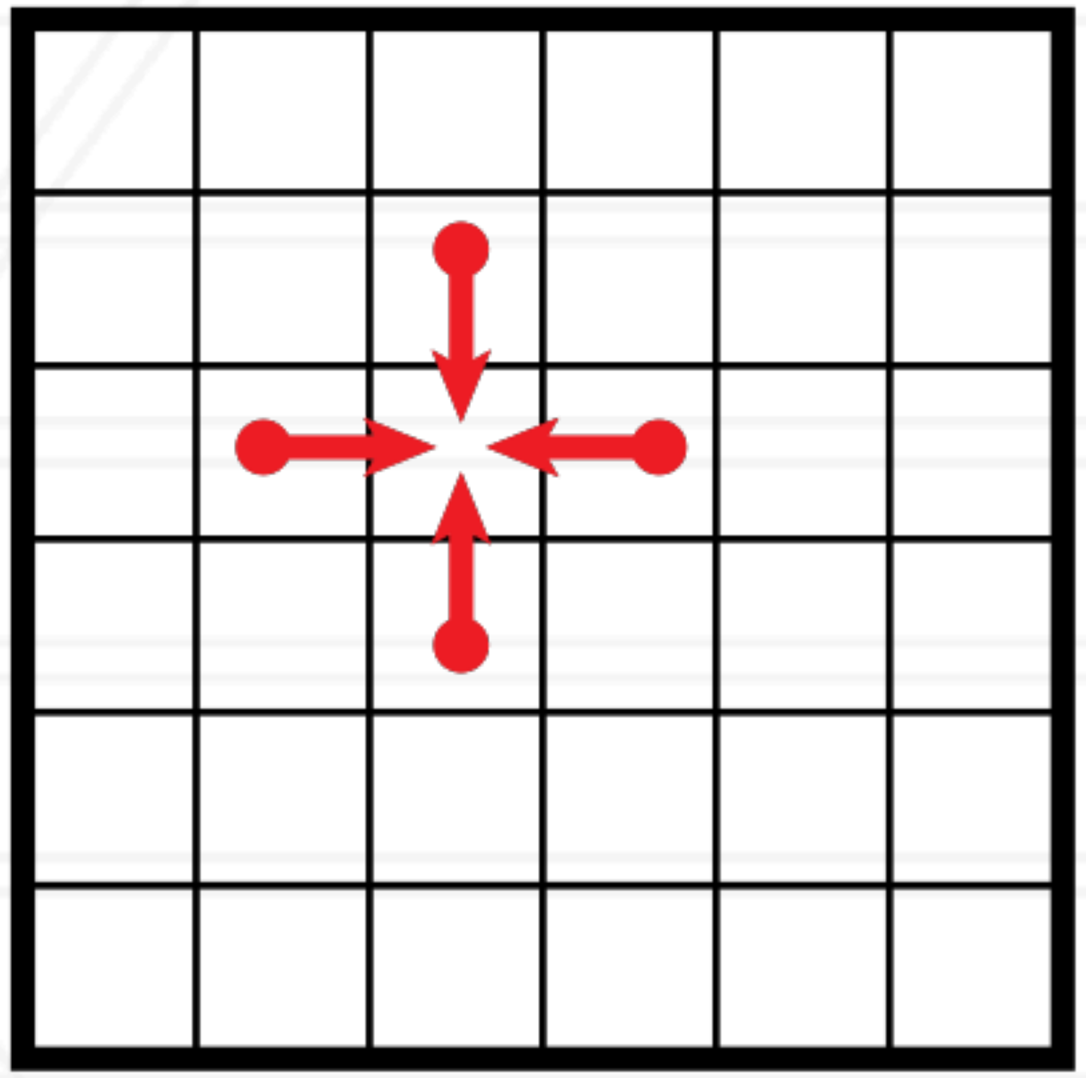
- Two-dimensional grid of (square) cells
- Each cell can be in one of two states: live or dead
- Every cell only interacts with its eight nearest neighbors
- In every generation (or iteration or time step), there are some rules that decide if a cell will continue to live or die or be born (dead ➜ live)



https://en.wikipedia.org/wiki/Conway's_Game_of_Life

Two-dimensional stencil computation

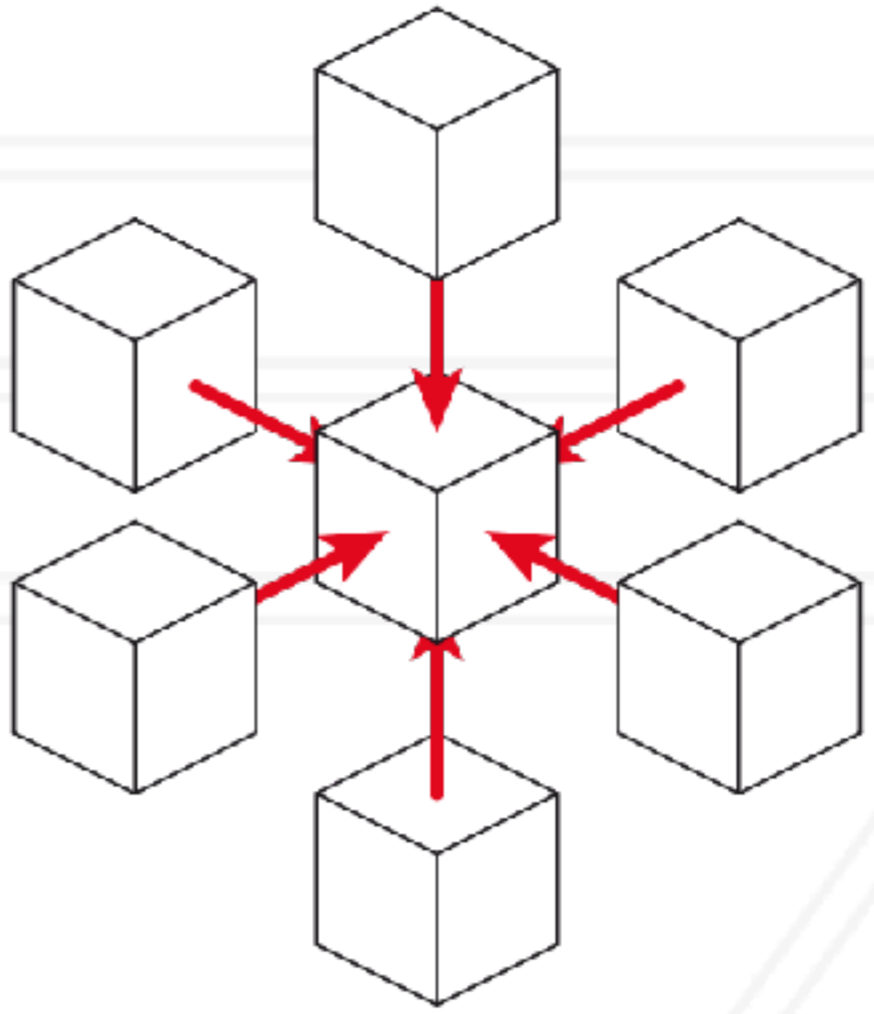
- Commonly found kernel in computational codes
- Heat diffusion, Jacobi method, Gauss-Seidel method

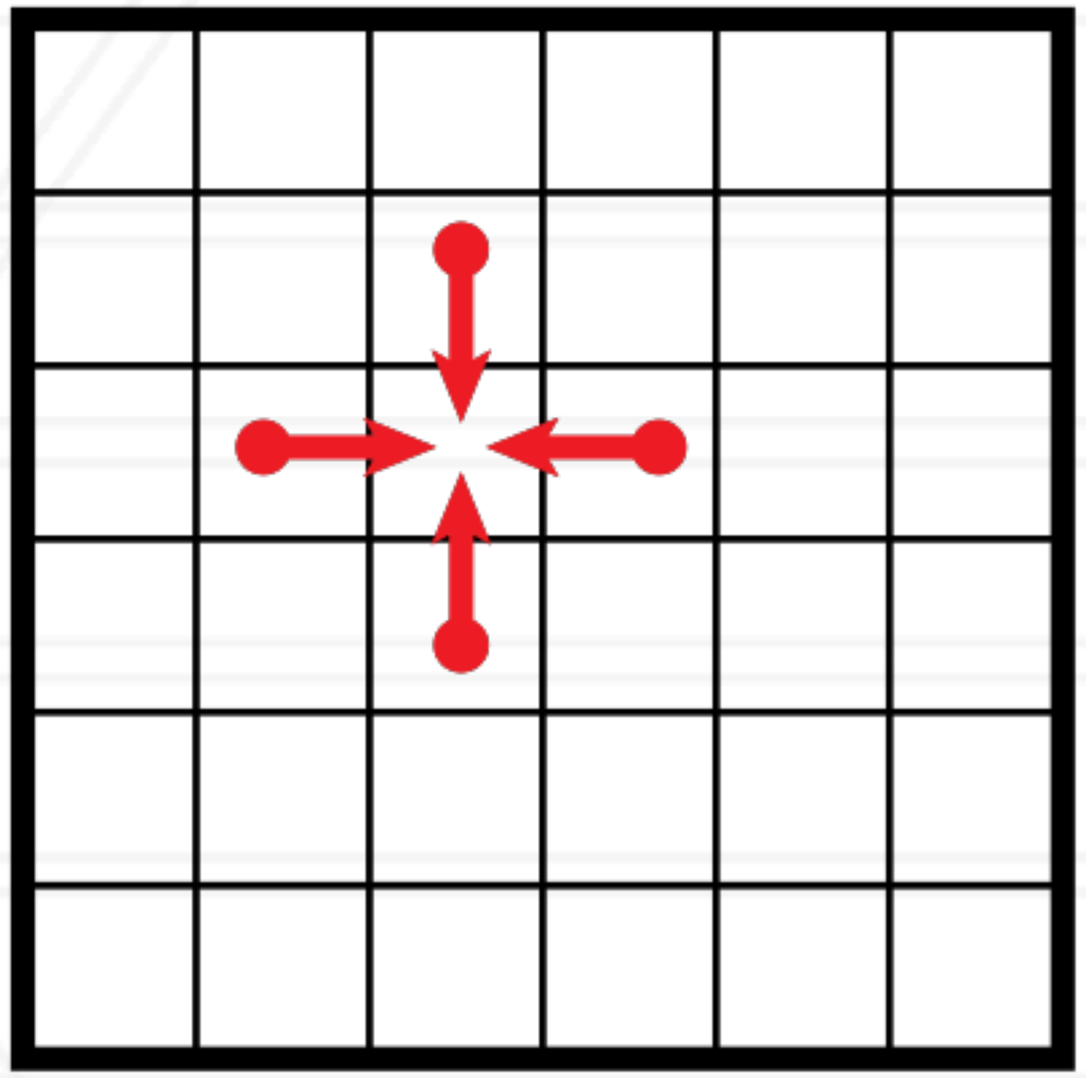


$$A[i,j] = \frac{A[i,j] + A[i-1,j] + A[i+1,j] + A[i,j-1] + A[i,j+1]}{5}$$

Two-dimensional stencil computation

- Commonly found kernel in computational codes
- Heat diffusion, Jacobi method, Gauss-Seidel method





$$A[i,j] = \frac{A[i,j] + A[i-1,j] + A[i+1,j] + A[i,j-1] + A[i,j+1]}{5}$$

Serial code

```
for(int t=0; t<num_steps; t++) {
  ...

  for(i ...)
    for(j ...)
      A_new[i, j] = (A[i, j] + A[i-1, j] + A[i+1, j] + A[i, j-1] + A[i, j+1]) * 0.2

  // copy contents of A_new into A
  ...
}
```

Serial code

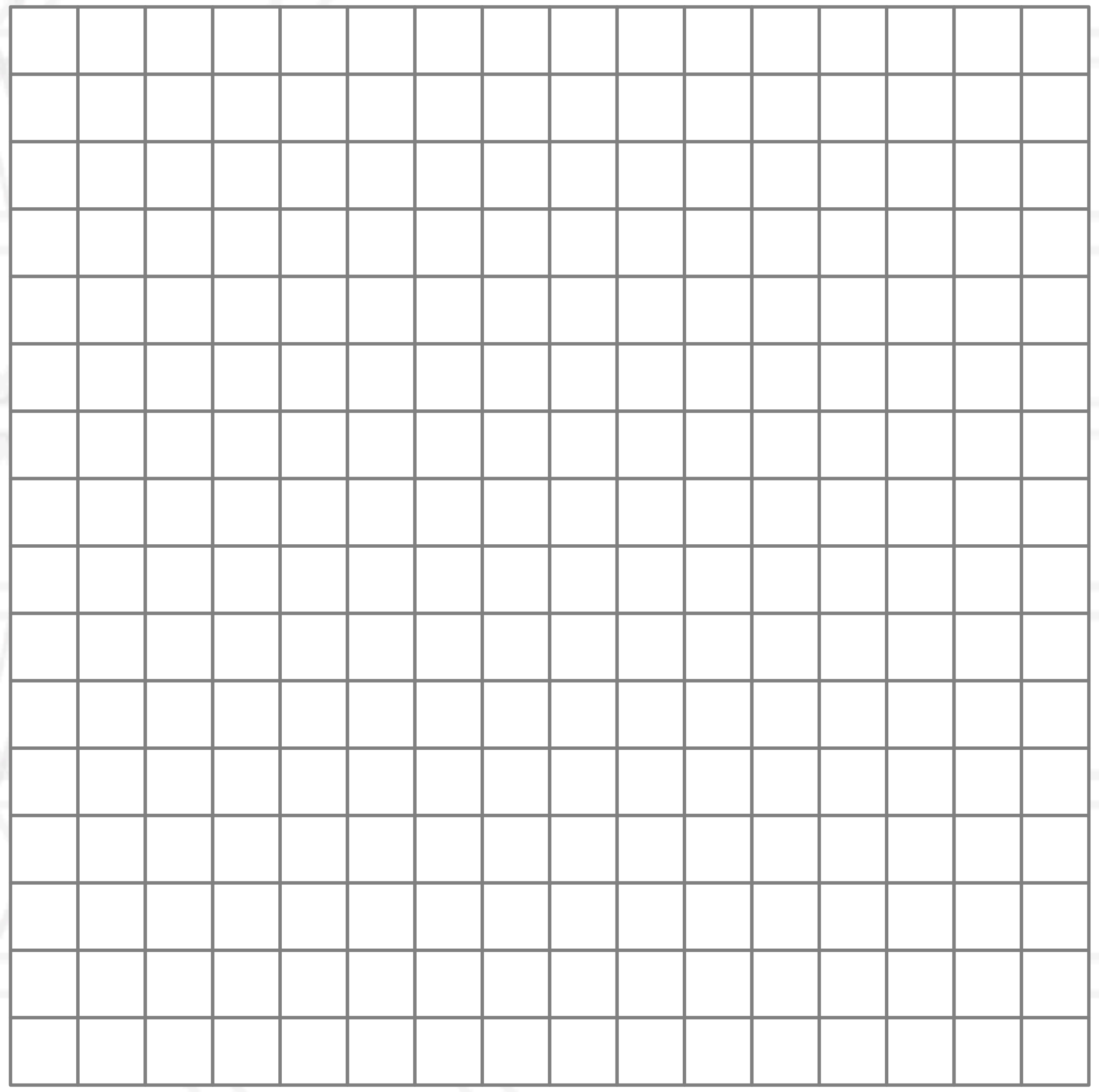
```
for(int t=0; t<num_steps; t++) {
  ...

  for(i ...)
    for(j ...)
      A_new[i, j] = (A[i, j] + A[i-1, j] + A[i+1, j] + A[i, j-1] + A[i, j+1]) * 0.2

  // copy contents of A_new into A
  ...
}
```

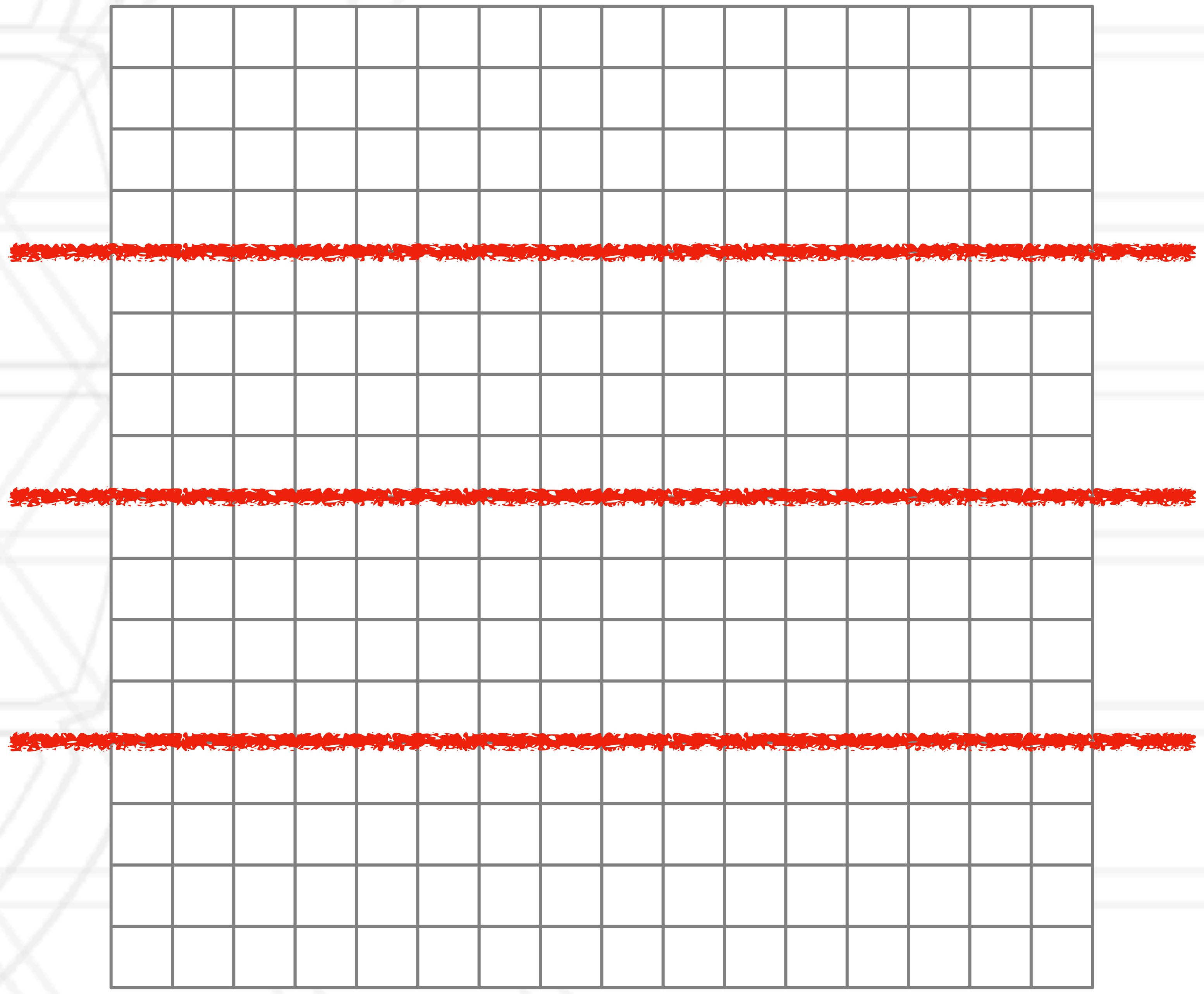
Why do we keep two copies of A?

2D stencil computation in parallel



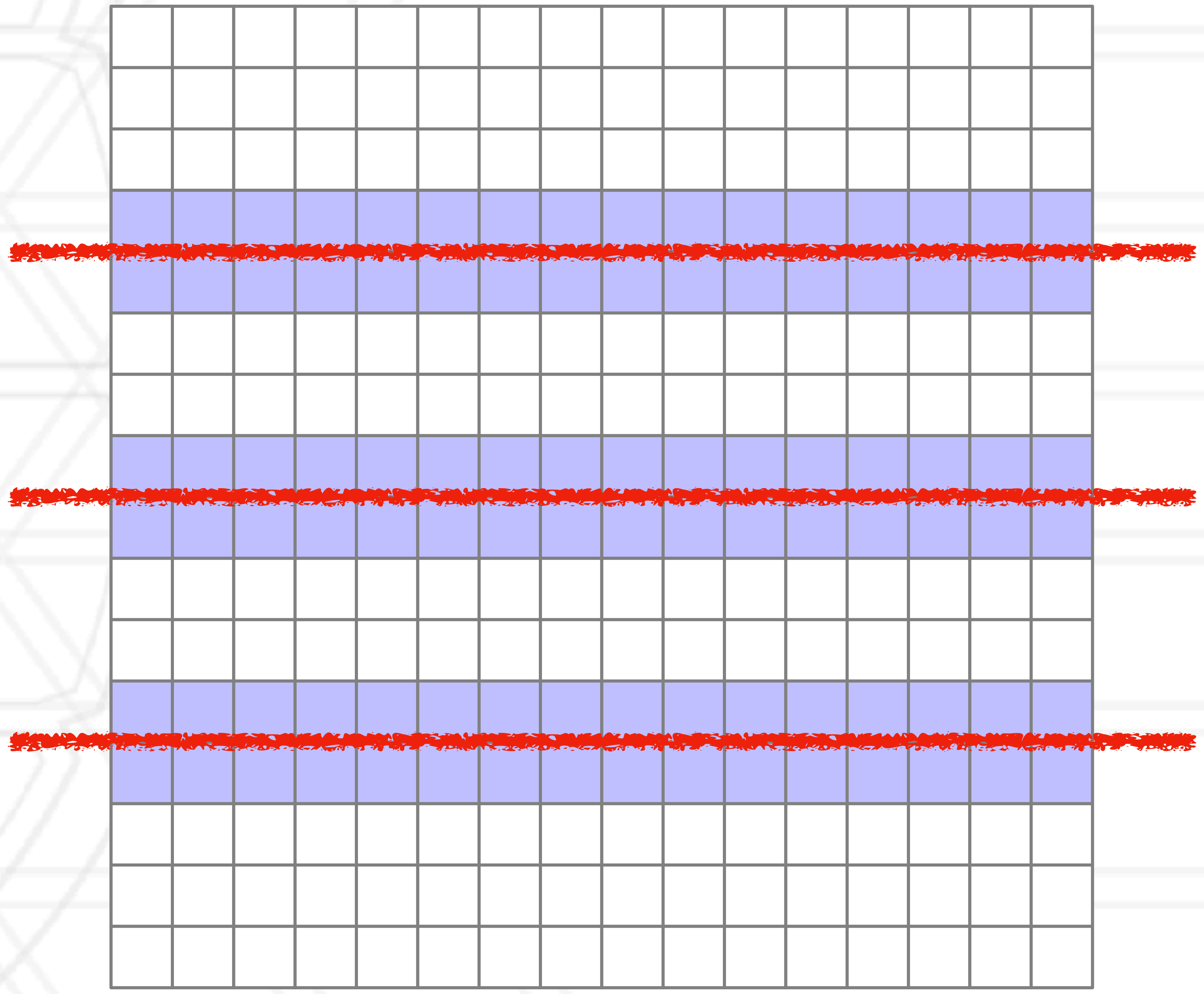
2D stencil computation in parallel

- 1D decomposition
  - Divide rows (or columns) among processes



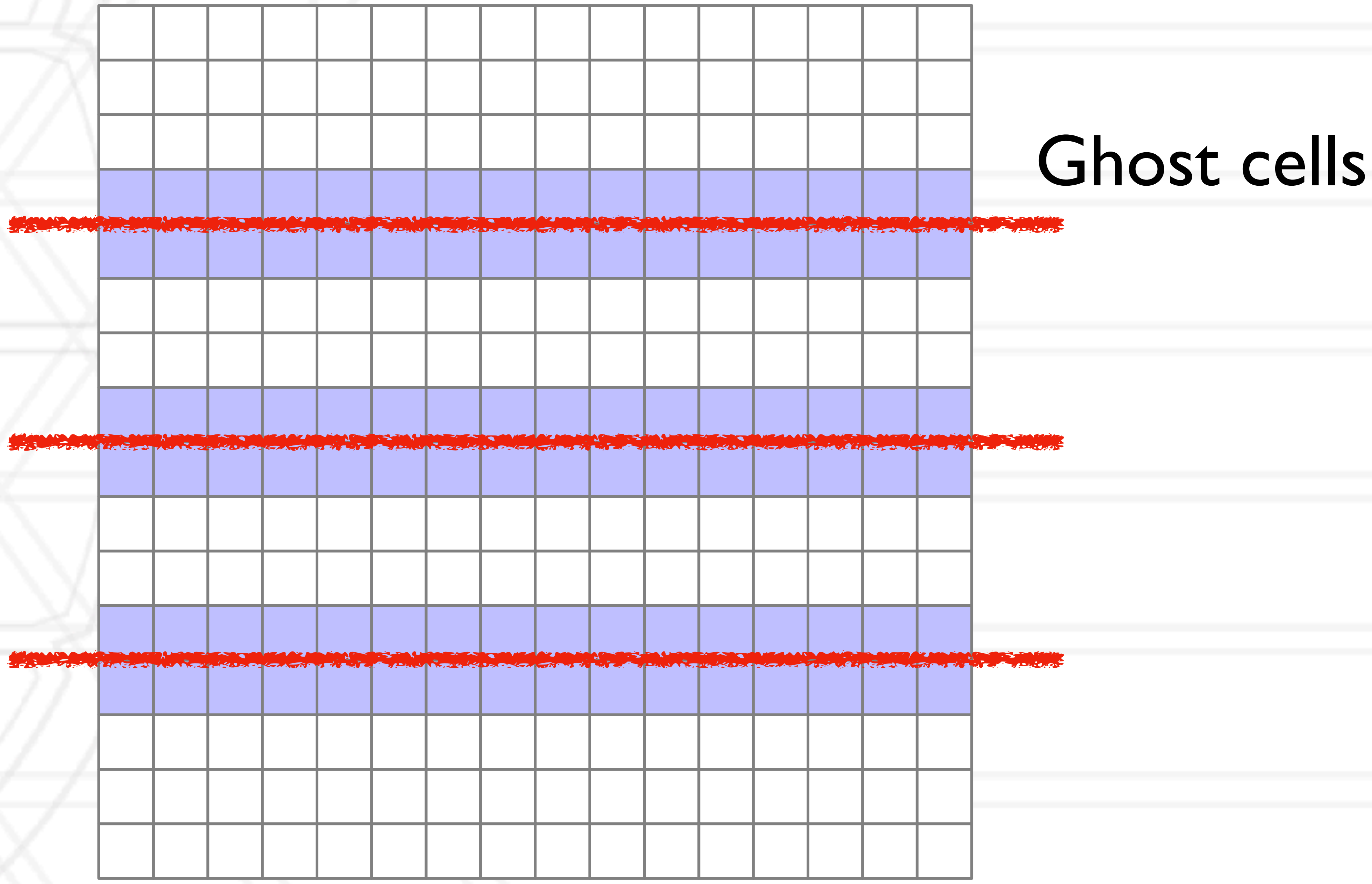
2D stencil computation in parallel

- 1D decomposition
  - Divide rows (or columns) among processes



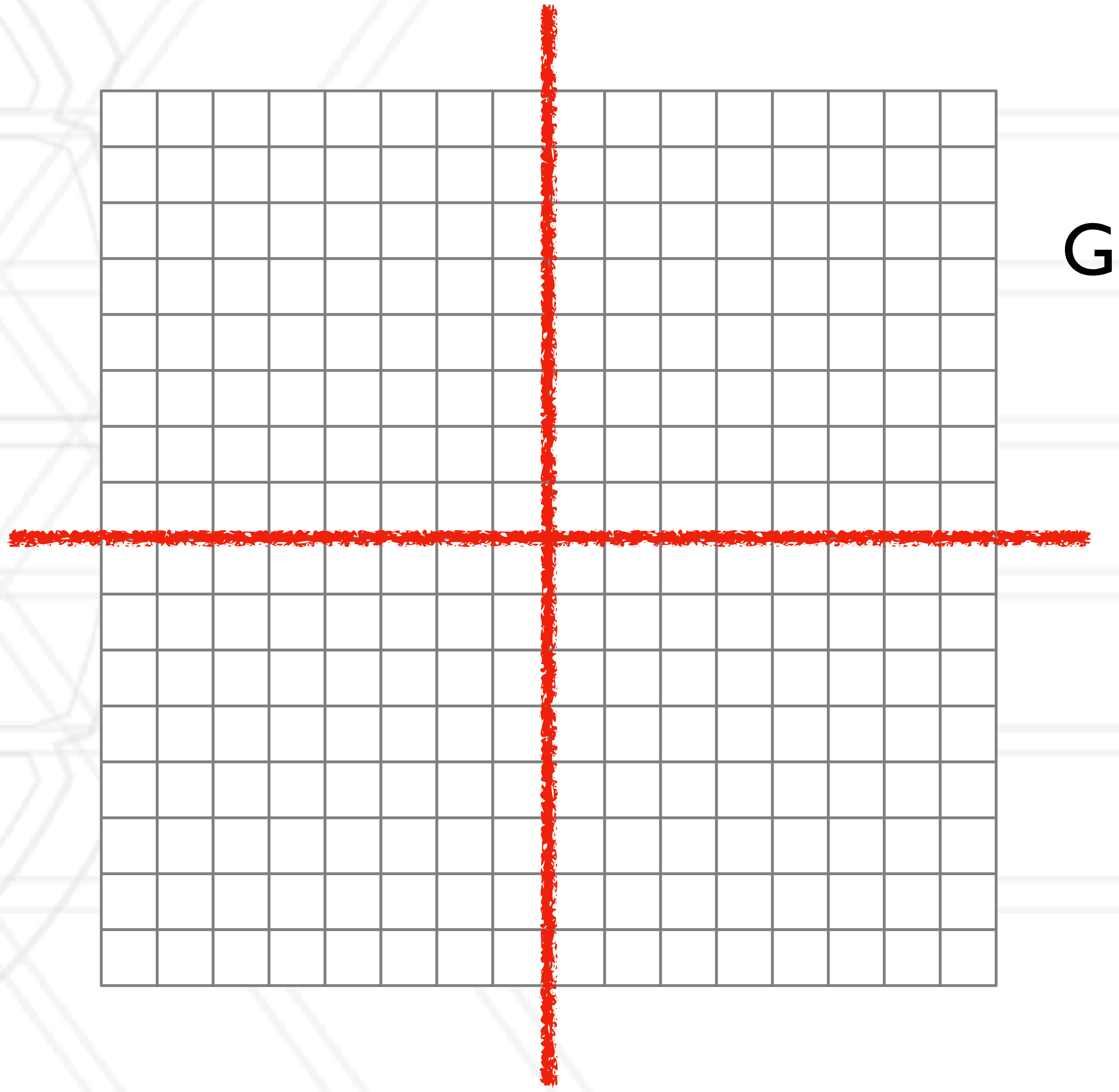
2D stencil computation in parallel

- 1D decomposition
  - Divide rows (or columns) among processes



2D stencil computation in parallel

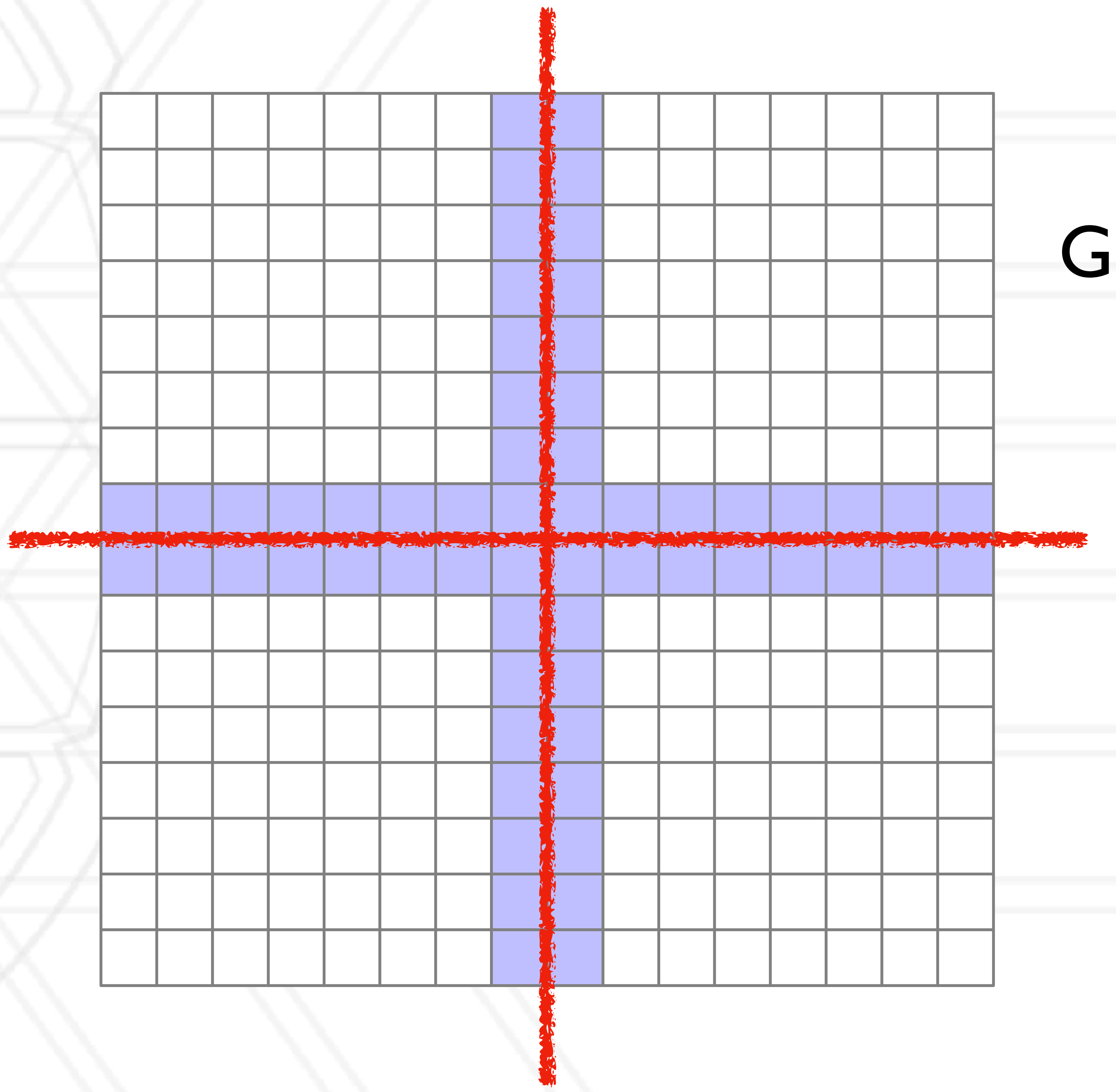
- 1D decomposition
  - Divide rows (or columns) among processes
- 2D decomposition
  - Divide both rows and columns (2d blocks) among processes



Ghost cells

2D stencil computation in parallel

- 1D decomposition
  - Divide rows (or columns) among processes
- 2D decomposition
  - Divide both rows and columns (2d blocks) among processes



Prefix sum

- Calculate sums of prefixes (running totals) of elements (numbers) in an array
- Also called a "scan" sometimes

```
pSum[0] = A[0]

for(i=1; i<N; i++) {
    pSum[i] = pSum[i-1] + A[i]
}
```

A	1	2	3	4	5	6	...
pSum	1	3	6	10	15	21	...

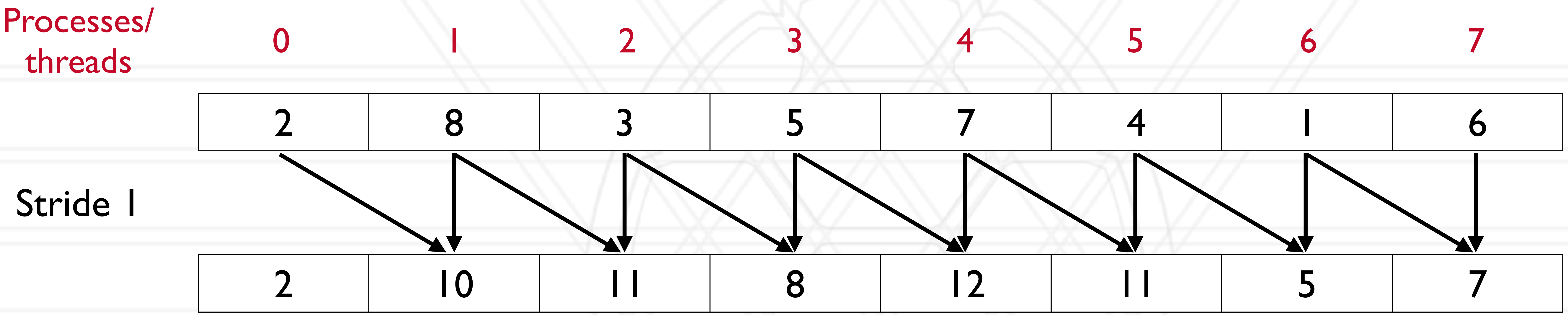
Parallel prefix sum

2	8	3	5	7	4	1	6

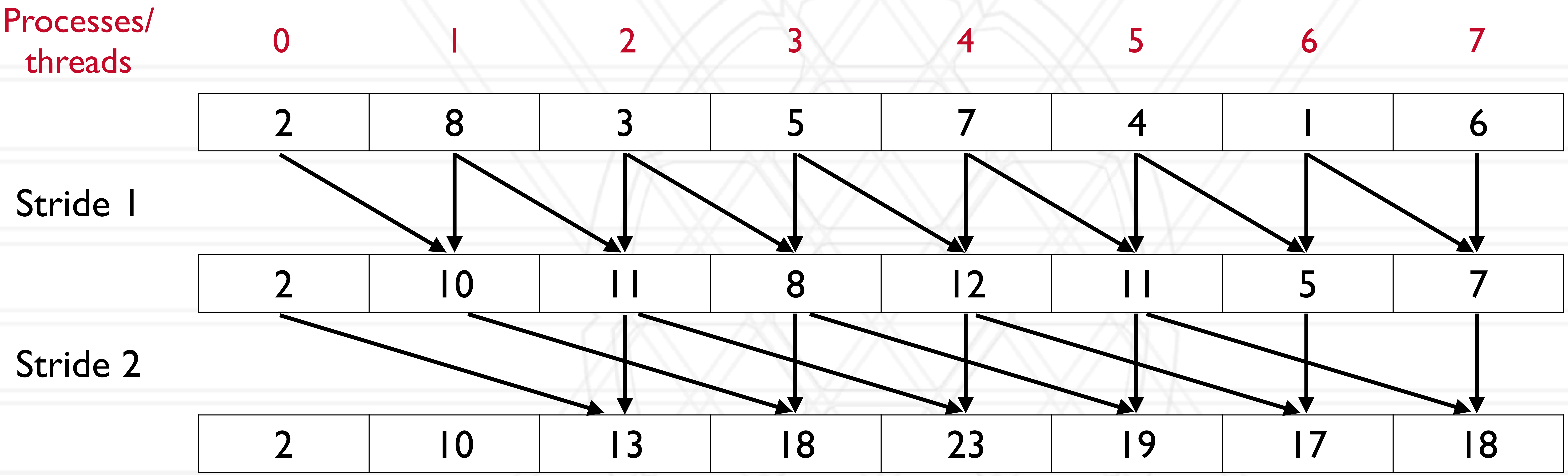
Parallel prefix sum

Processes/ threads	0	1	2	3	4	5	6	7
	2	8	3	5	7	4	1	6

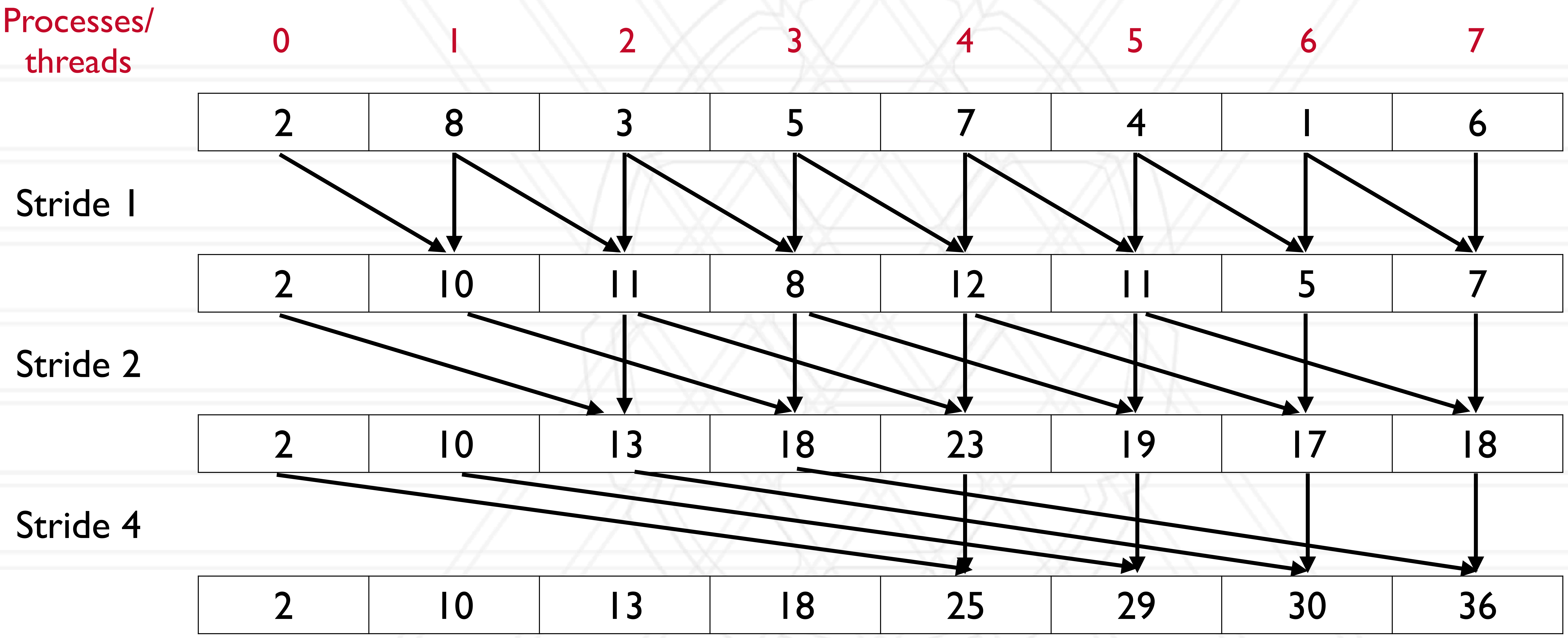
Parallel prefix sum



Parallel prefix sum



Parallel prefix sum



In practice

In practice

- You have N numbers and p processes, $N >> p$

In practice

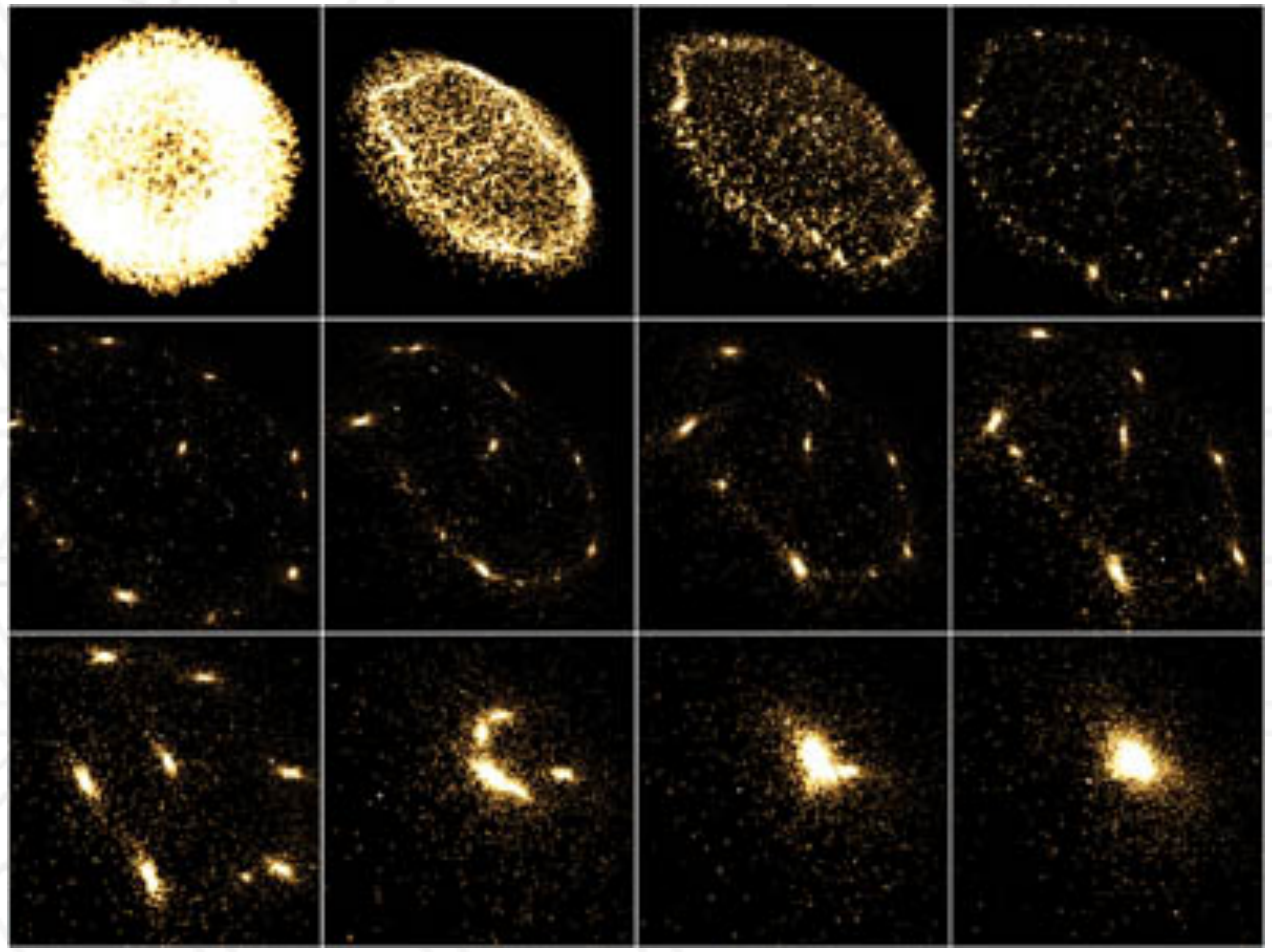
- You have N numbers and p processes, $N >> p$
- Assign a N/p block to each process
  - Do the serial prefix sum calculation for the blocks owned on each process locally

In practice

- You have *N* numbers and *p* processes, *N* >> *p*
- Assign a *N*/*p* block to each process
  - Do the serial prefix sum calculation for the blocks owned on each process locally
- Then do parallel algorithm with partial prefix sums (using the last element from each local block)
  - Last element from sending process is added to all elements in receiving process' sub-block

The *n*-body problem

- Simulate the motion of celestial objects interacting with one another due to gravitational forces
- Naive algorithm: $O(n^2)$
  - Every body calculates forces pair-wise with every other body (particle)



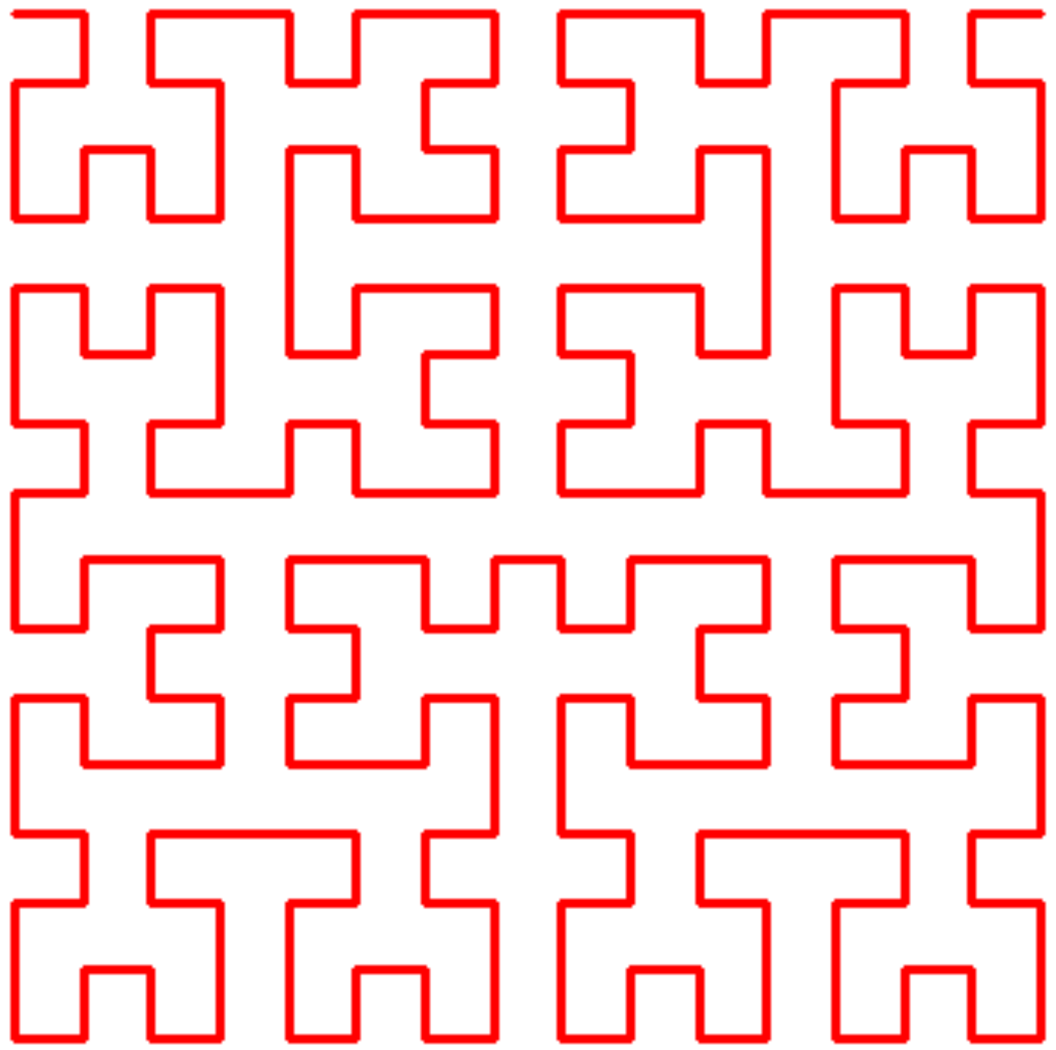
https://developer.nvidia.com/gpugems/gpugems3/part-v-physics-simulation/chapter-31-fast-n-body-simulation-cuda

Data distribution in *n*-body problems

- Naive approach: Assign n/p particles to each process
- Other approaches?

Data distribution in *n*-body problems

- Naive approach: Assign n/p particles to each process
- Other approaches?



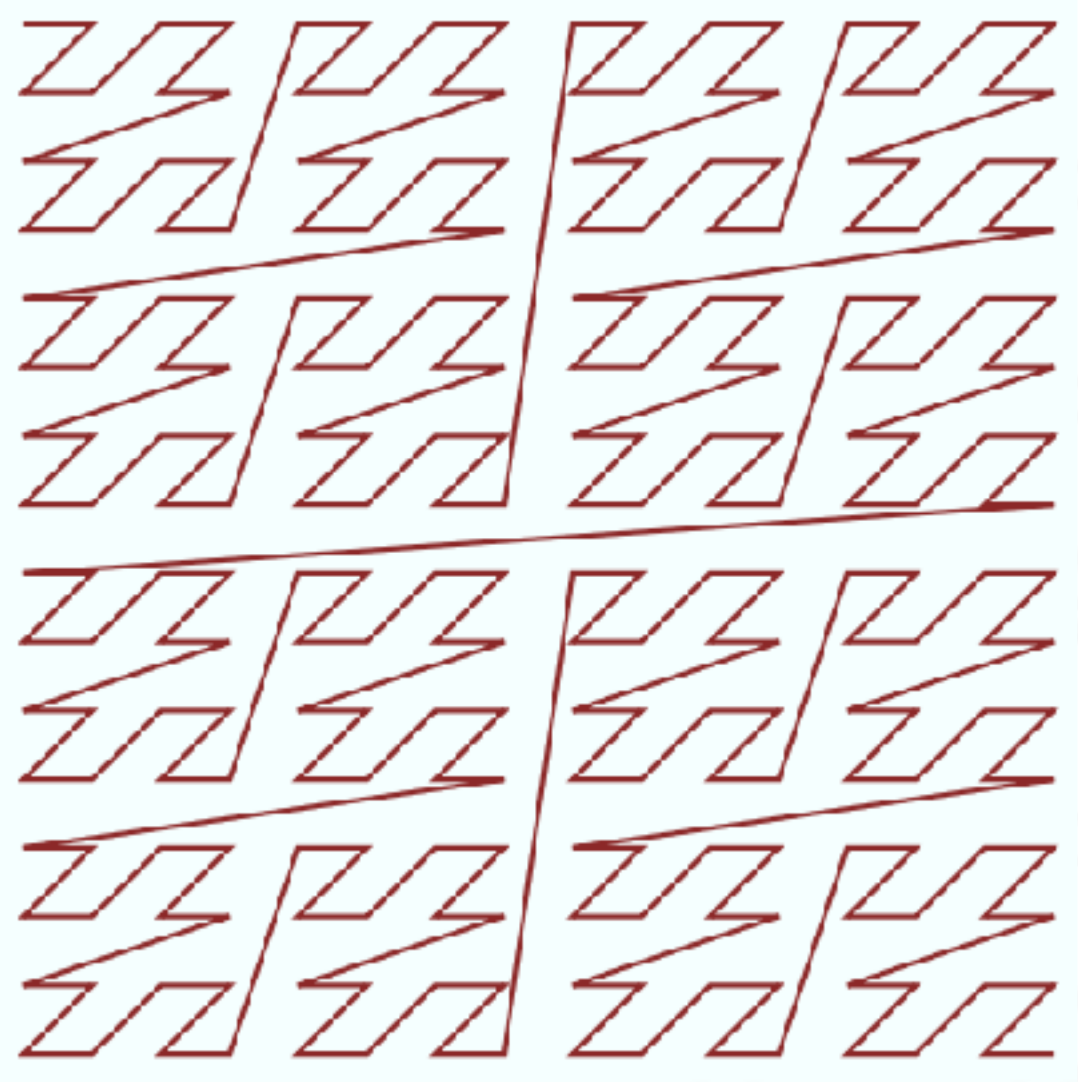
Space-filling curves

http://datagenetics.com/blog/march22013/

https://en.wikipedia.org/wiki/Z-order_curve

Data distribution in *n*-body problems

- Naive approach: Assign n/p particles to each process
- Other approaches?



Space-filling curves

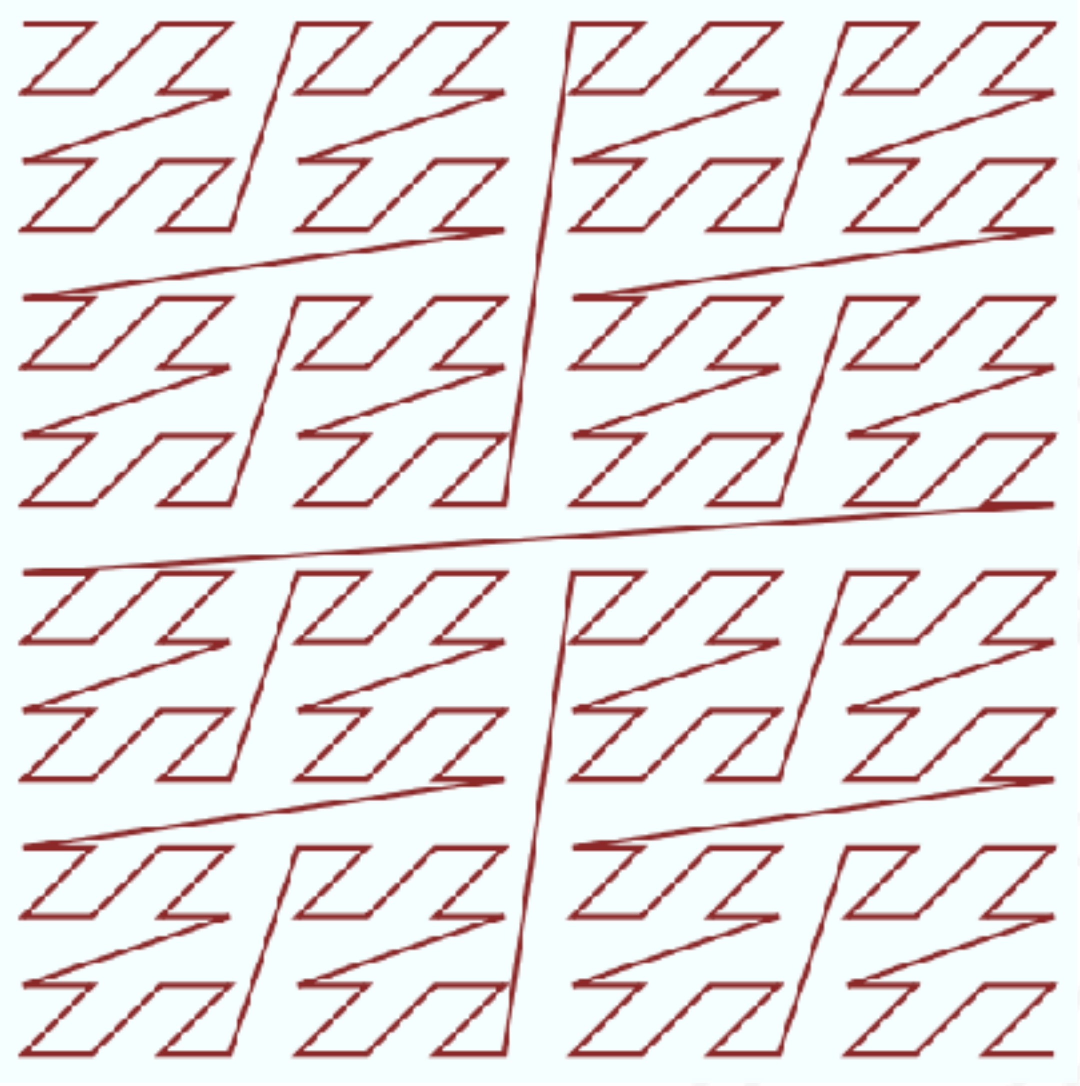
http://datagenetics.com/blog/march22013/

https://en.wikipedia.org/wiki/Z-order_curve

Data distribution in *n*-body problems

- Naive approach: Assign n/p particles to each process
- Other approaches?

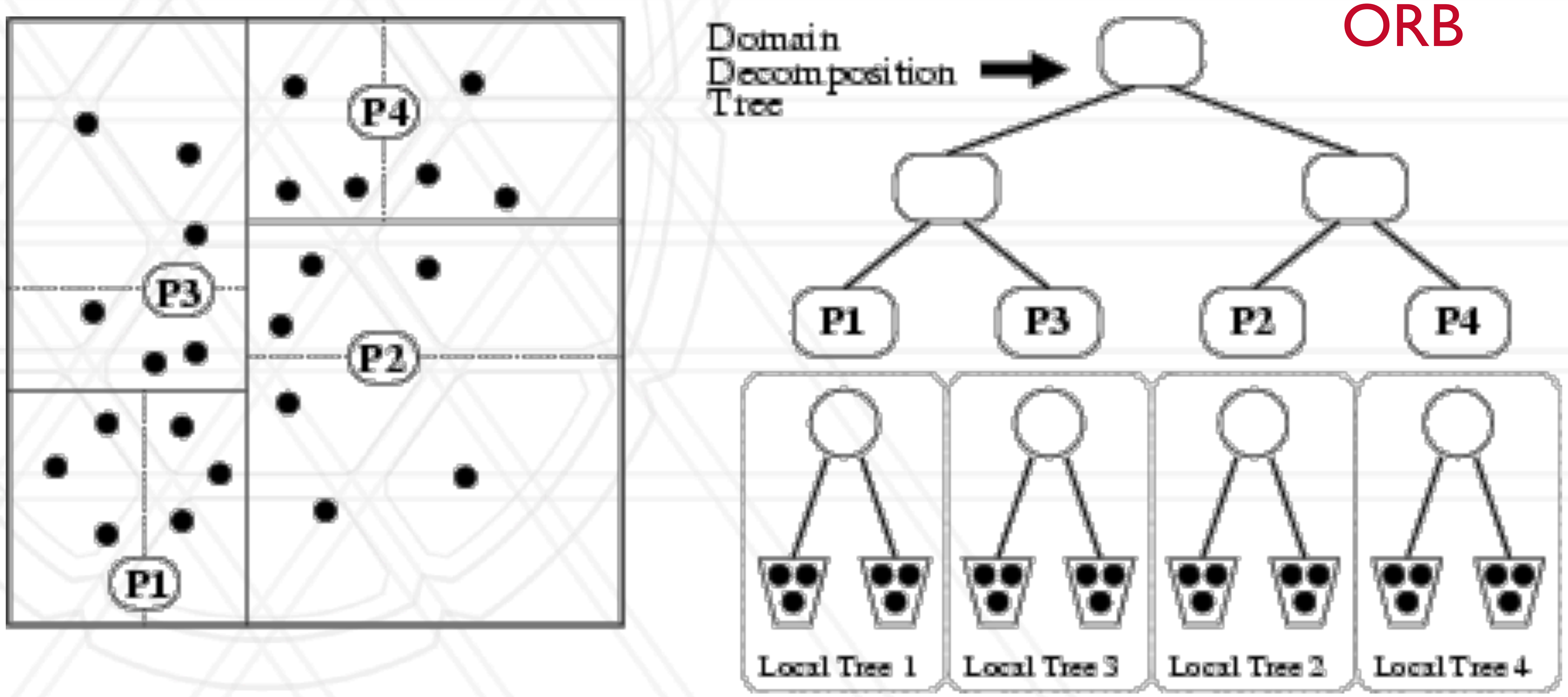
Space-filling curves



http://datagenetics.com/blog/march22013/

https://en.wikipedia.org/wiki/Z-order_curve

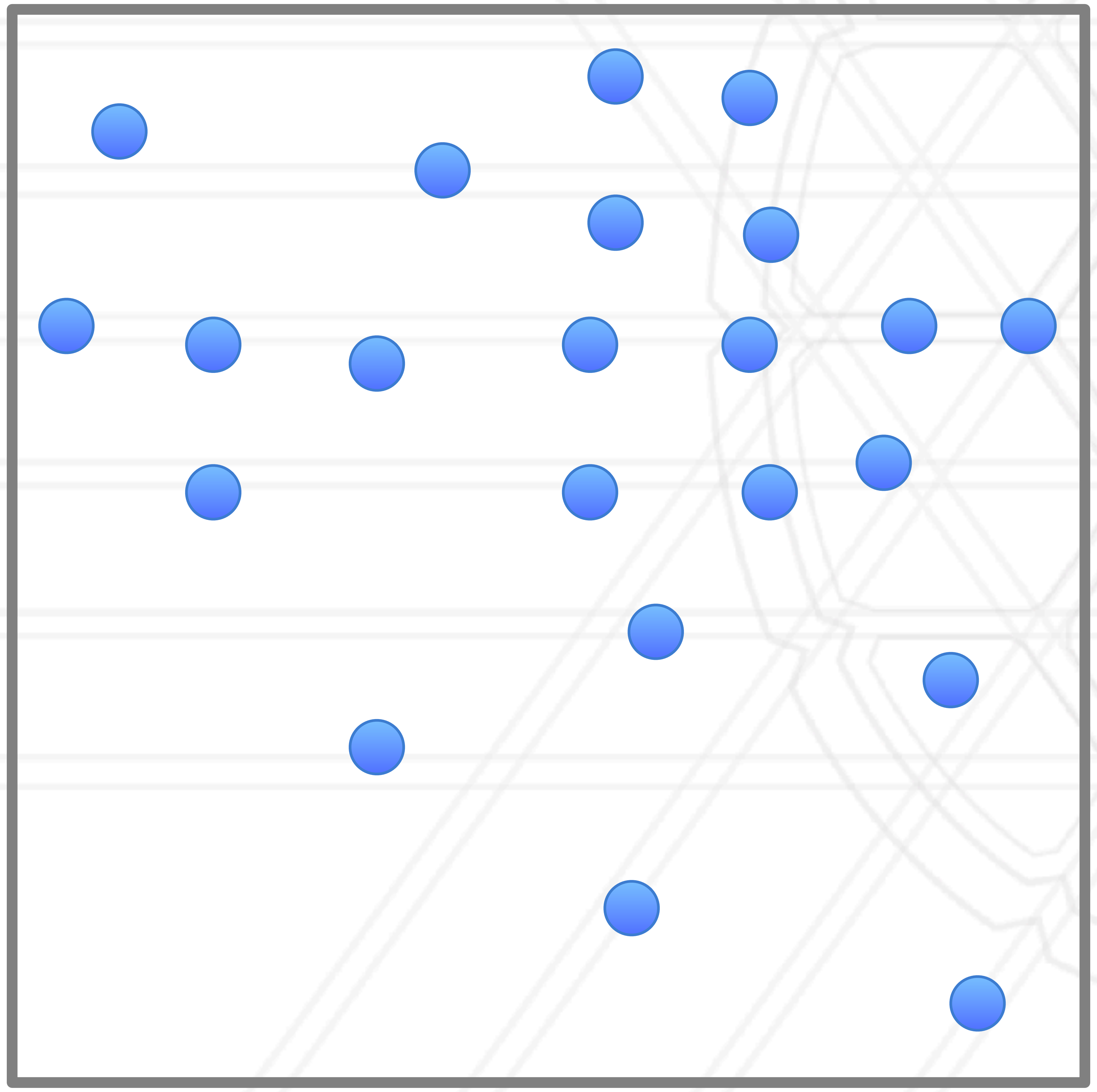
ORB



http://charm.cs.uiuc.edu/workshops/charmWorkshop2011/slides/CharmWorkshop2011_apps_ChaNGa.pdf

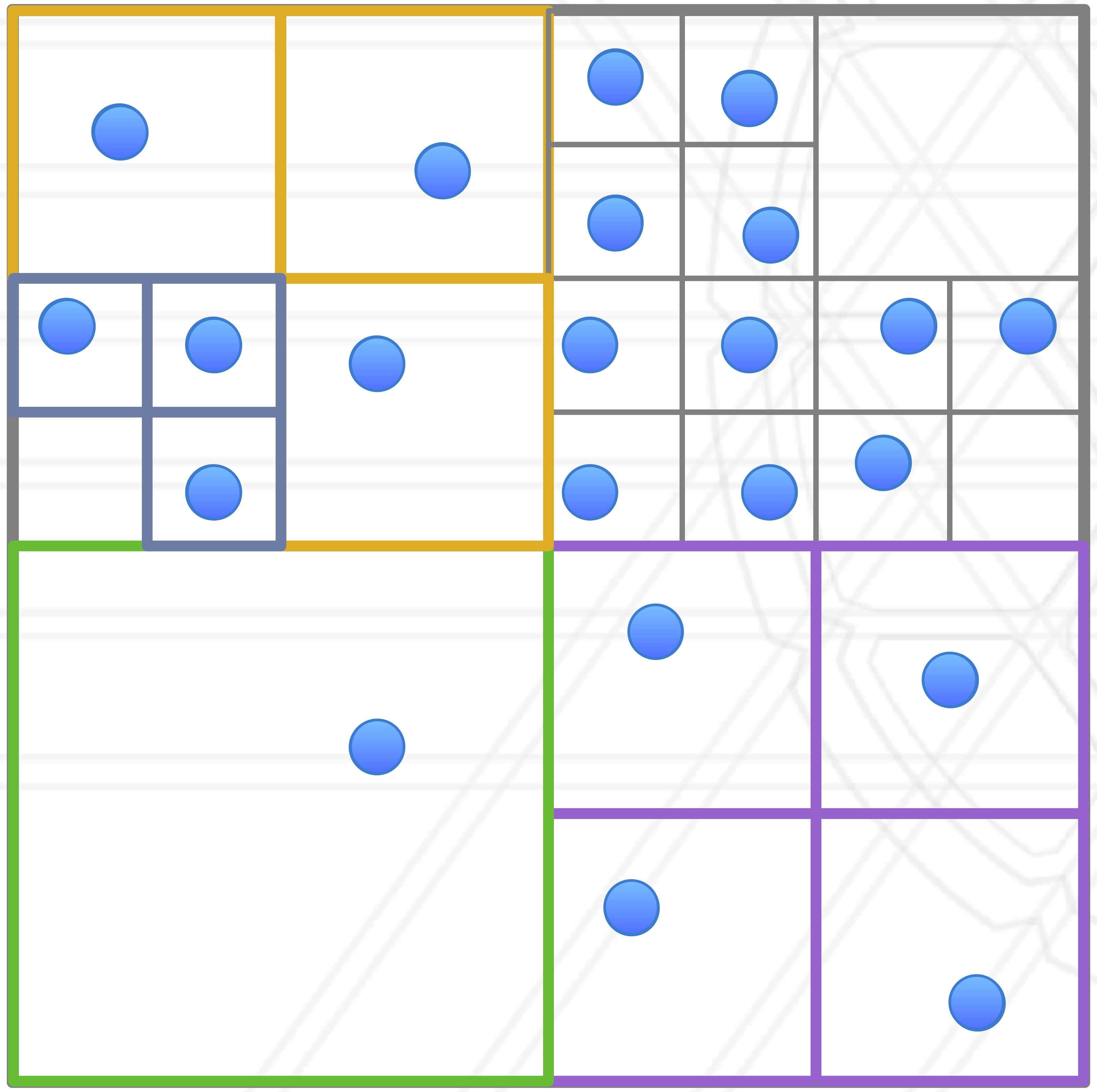
Data distribution in *n*-body problems

- Let us consider a two-dimensional space with bodies/particles in it



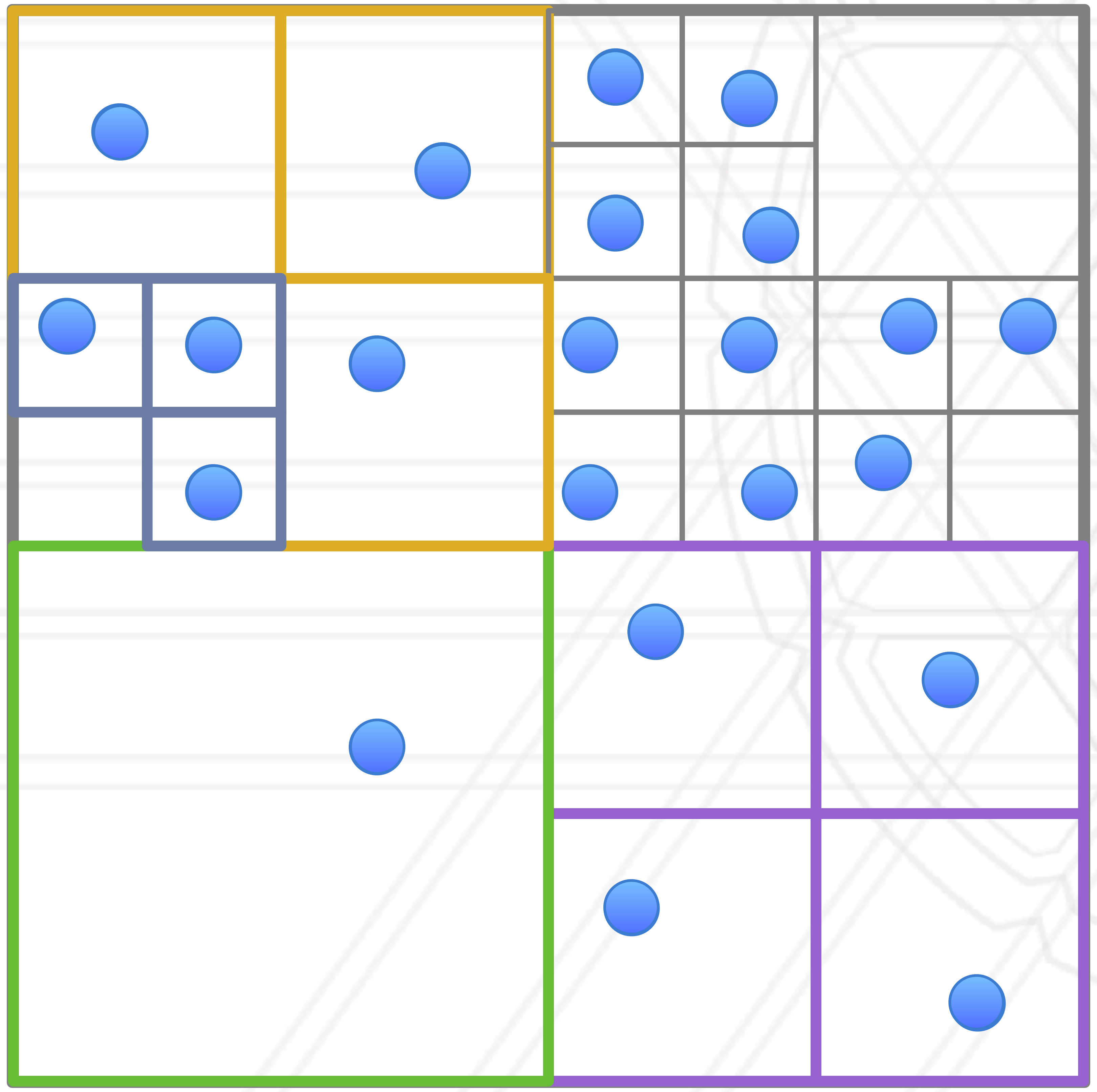
Data distribution in *n*-body problems

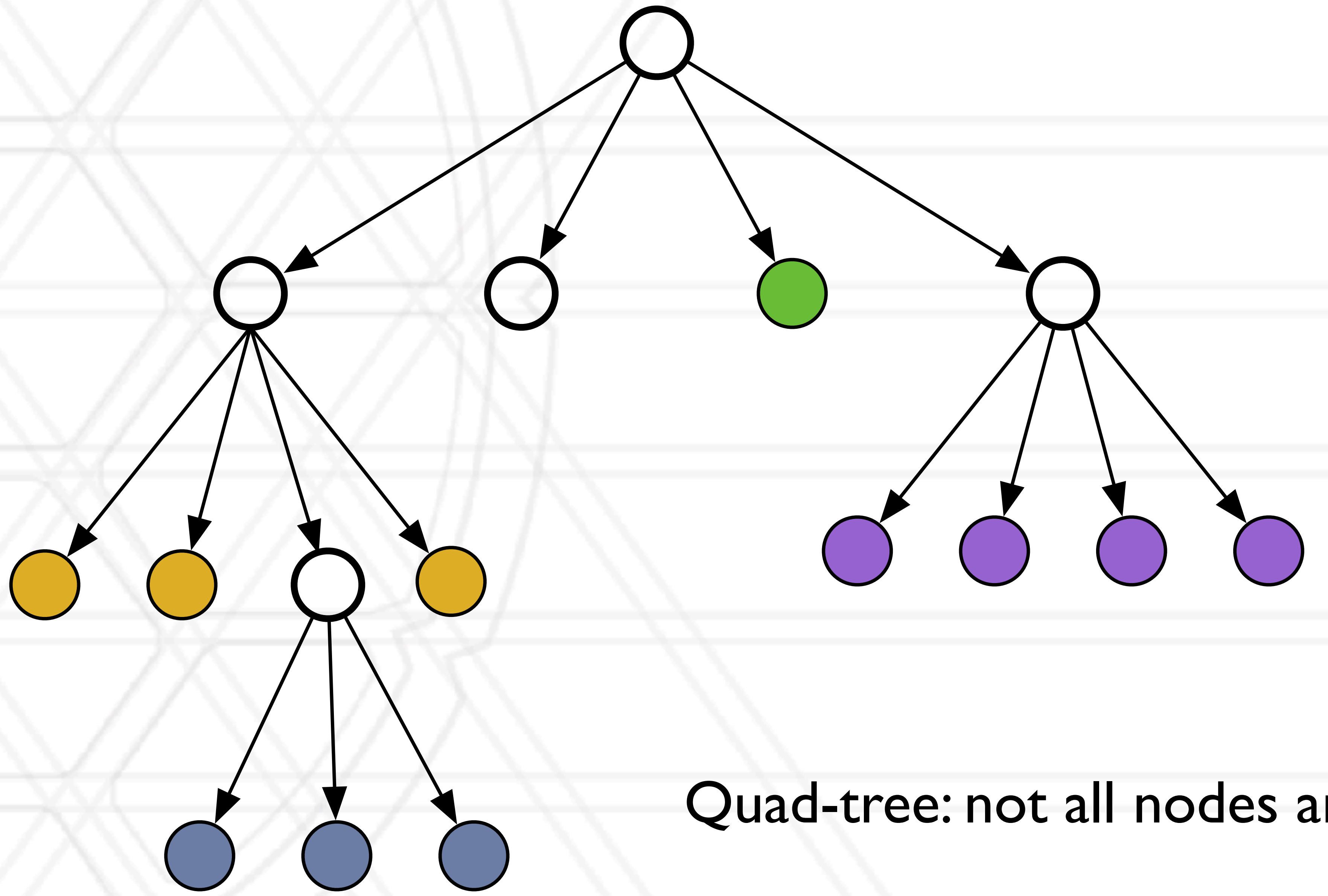
- Let us consider a two-dimensional space with bodies/particles in it



Data distribution in *n*-body problems

- Let us consider a two-dimensional space with bodies/particles in it





Quad-tree: not all nodes are shown

Load balance and grain size

- Load balance: try to balance the amount of work (computation) assigned to different threads/ processes
  - Bring ratio of maximum to average load as close to 1.0 as possible
  - Secondary consideration: also load balance amount of communication
- Grain size: ratio of computation-to-communication
  - Coarse-grained (more computation) vs. fine-grained (more communication)

UNIVERSITY OF MARYLAND

Abhinav Bhatele

5218 Brendan Iribe Center (IRB) / College Park, MD 20742

phone: 301.405.4507 / e-mail: bhatele@cs.umd.edu