Designing Parallel Programs

Abhinav Bhatele, Department of Computer Science

UNIVERSITY OF MARYLAND

Reminders / Announcements

- Office hours are posted on the website
- Most queries specially regarding issues with working on programming assignments should be posted on piazza
- Any other logistical issues should be sent to email: cmsc416@cs.umd.edu
  - Do not send the same email twice
- When emailing, please mention your course and section number:
  - Example: 416 / Section 0101

Announcements

- Reminder: Assignment 0.1 is due on: Sep 10, 11:59 pm ET
- Good-faith attempt of each assignment is required
- Reminders:
  - Do not run/execute code on the login node
  - Do not run sudo on zaratan
  - Best not to use vscode to ssh into zaratan
  - Best way to report issues: What command did you run, and the full error

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?
  - Could be based on what input data each process holds or based on what output data we want each process to hold

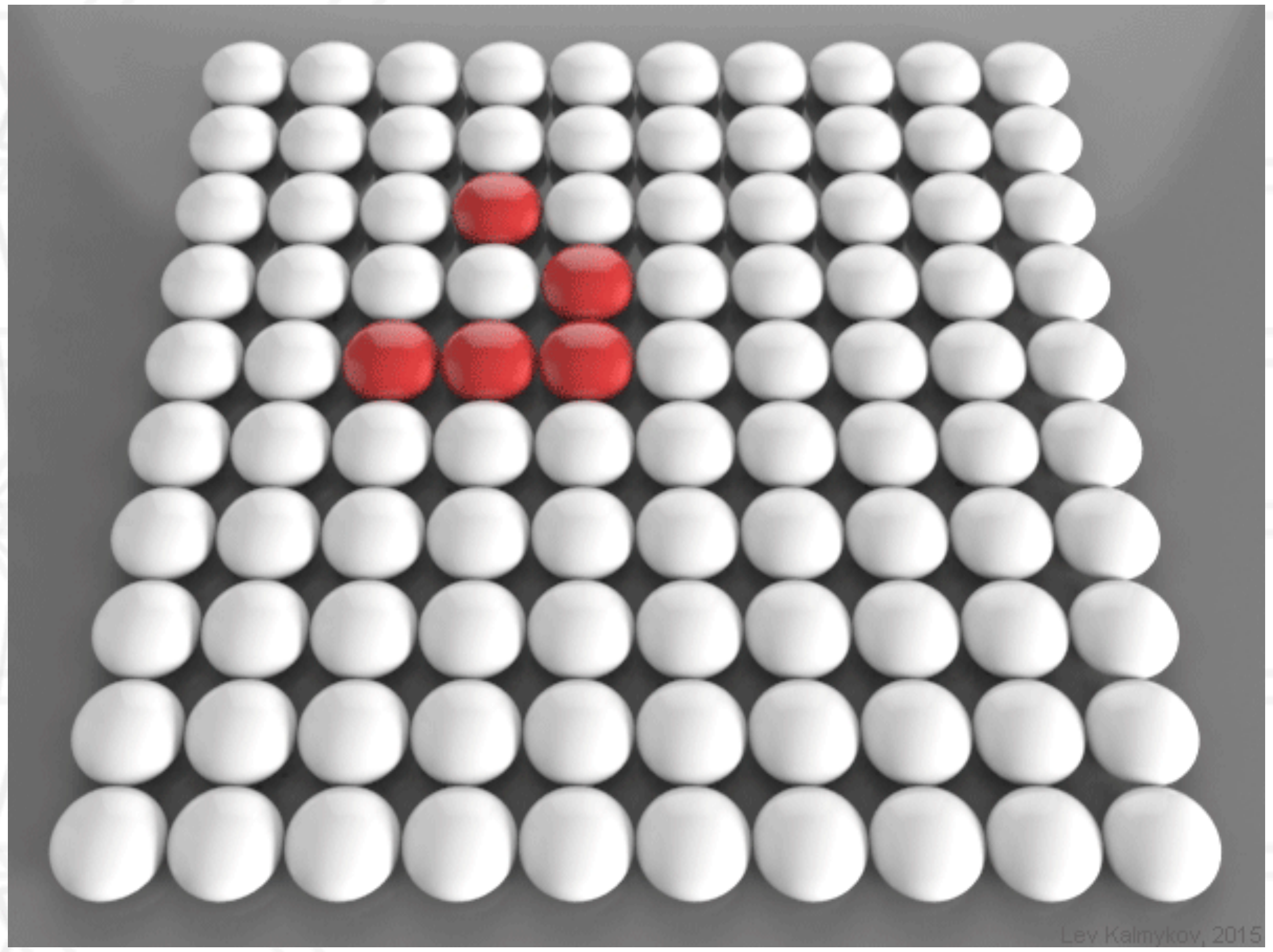
Writing parallel programs

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?
  - Could be based on what input data each process holds or based on what output data we want each process to hold
- Figure out how often communication will be needed

Conway's Game of Life

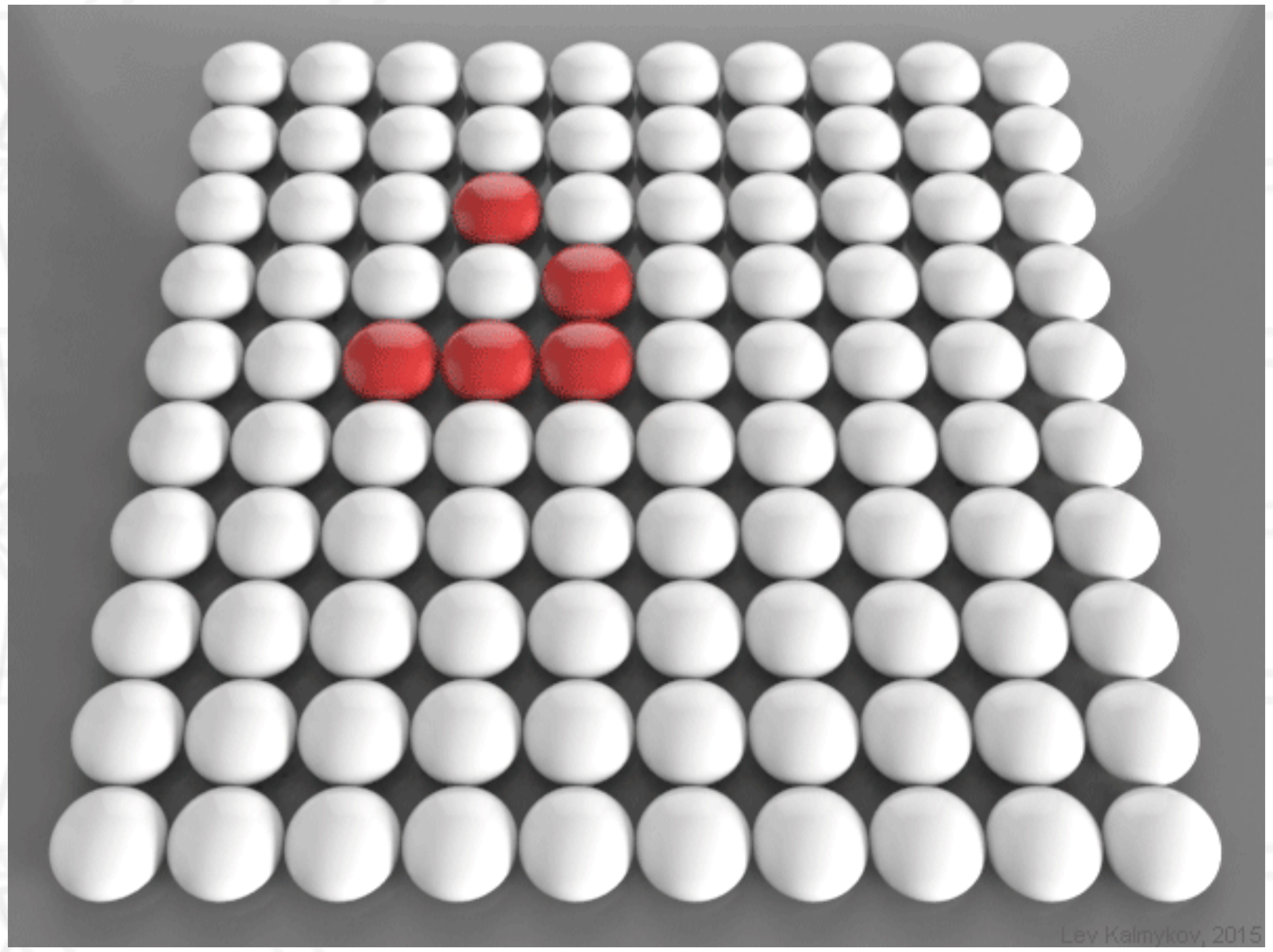
- Two-dimensional grid of cells
- Each cell can be in one of two states: live or dead
- Every cell only interacts with its eight nearest neighbors
- In every generation (or iteration or time step), there are some rules that decide if a cell will continue to live or die or be born (dead ➜ live)



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

Conway’s Game of Life

- Two-dimensional grid of 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)

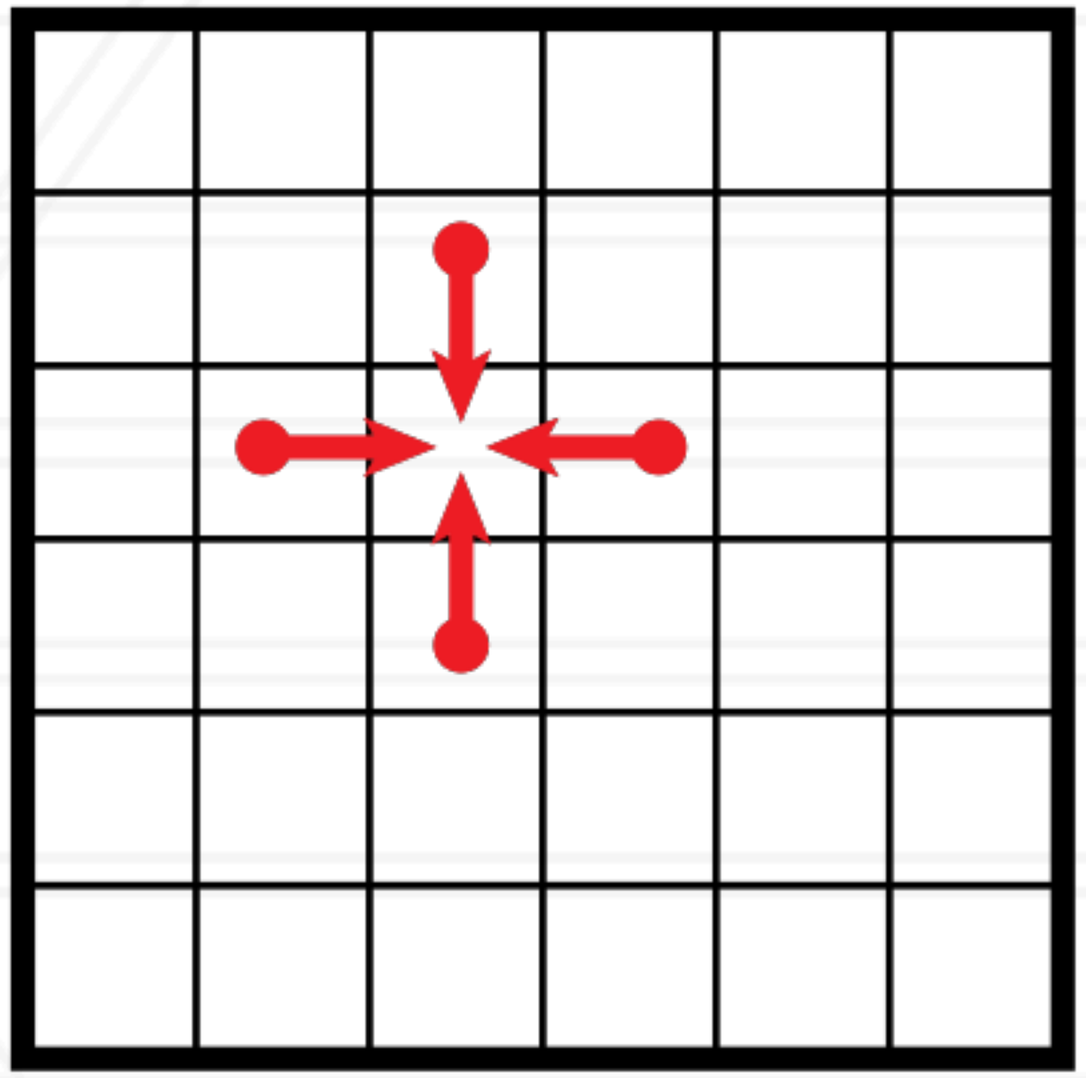


https://en.wikipedia.org/wiki/Conway's_Game_of_Life

Two-dimensional stencil computation

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

2D 5-point Stencil

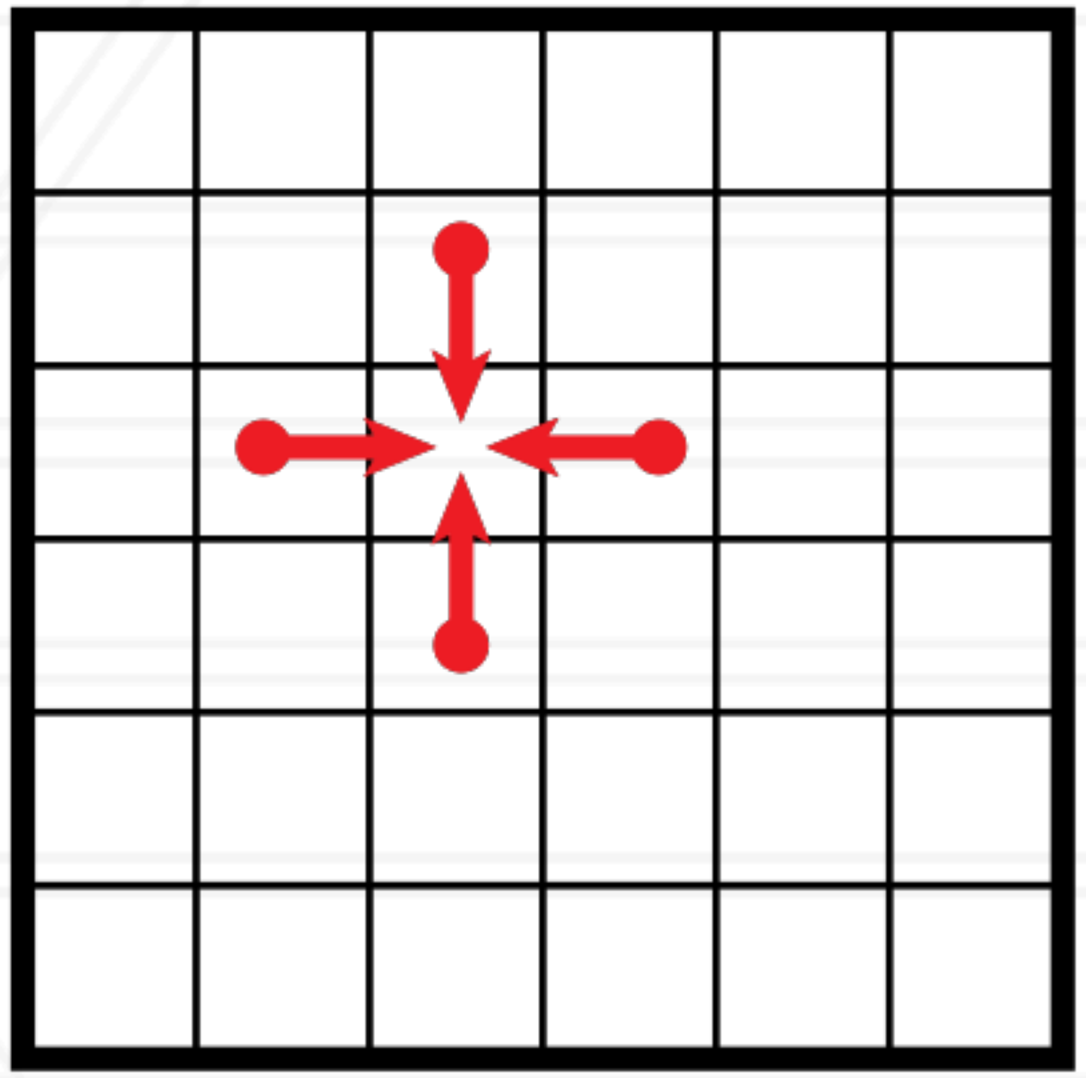


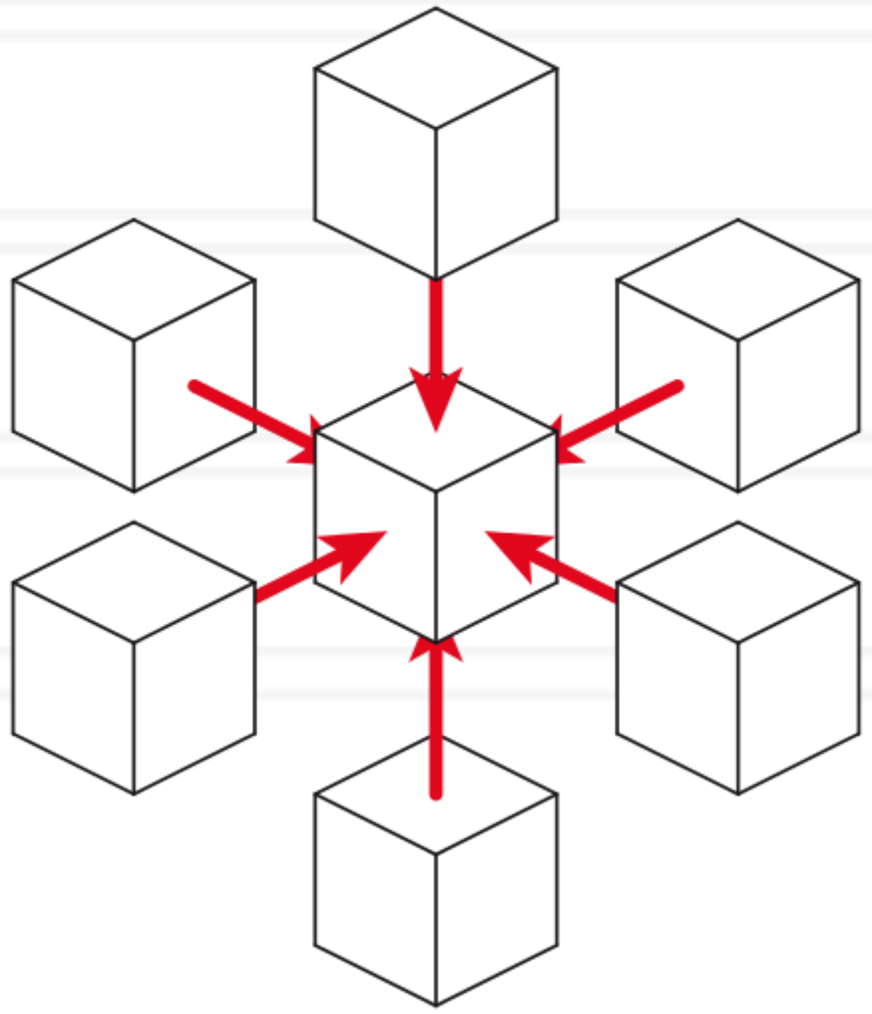
$$A[i,j] = \frac{A[i,j] + A[i-1,j] + A[i+1,j] + A[i,j-1] + A[i,j+1]}{5}$$

Two-dimensional stencil computation

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

2D 5-point Stencil





3D 7-point Stencil

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

Serial code

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

  double A[N][M];
  double A_new[N][M];

  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++) {
  ...

  double A[N][M];
  double A_new[N][M];

  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?

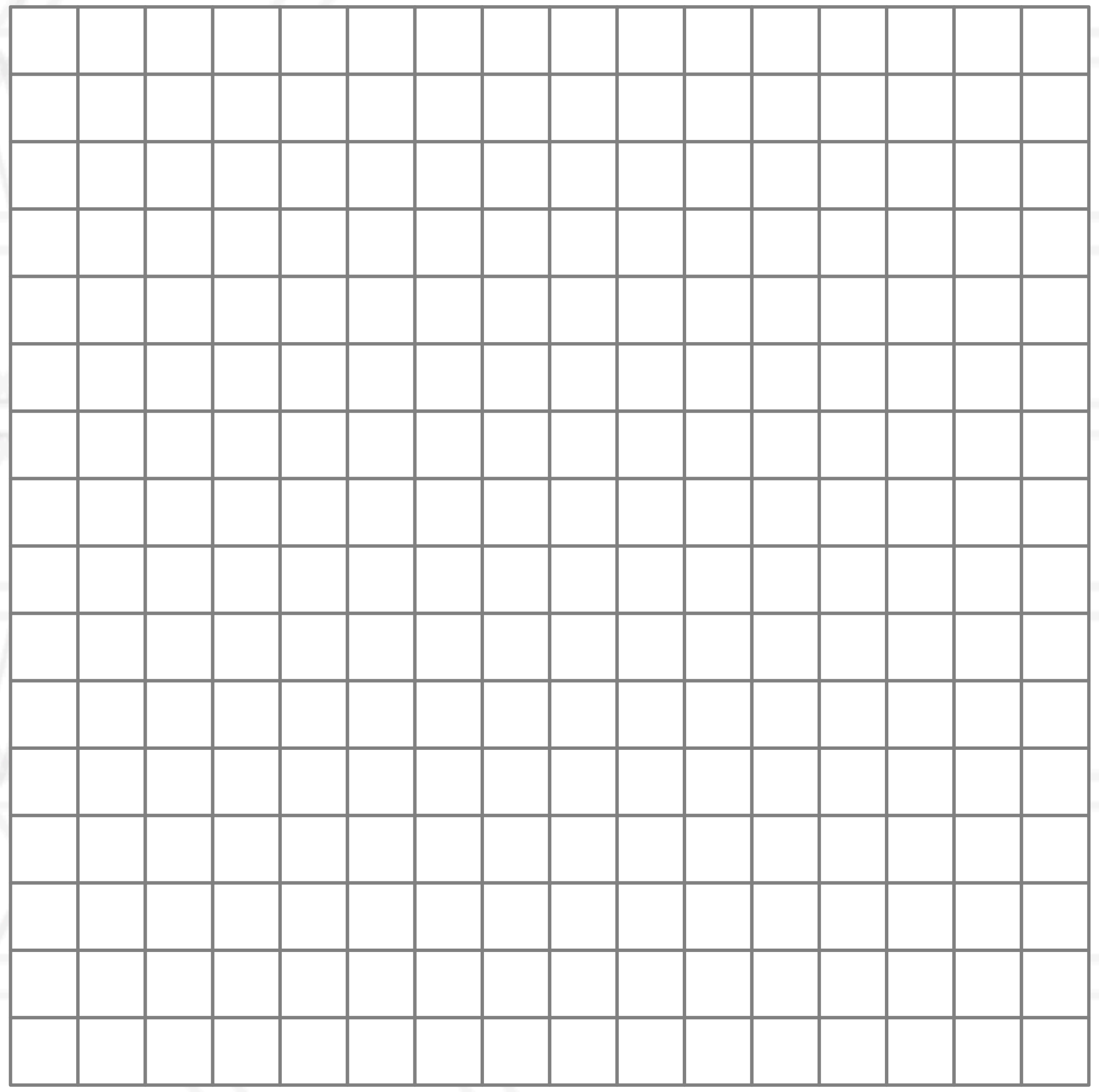
Serial code

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

  double A[N][M];
  double A_new[N][M];

  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?

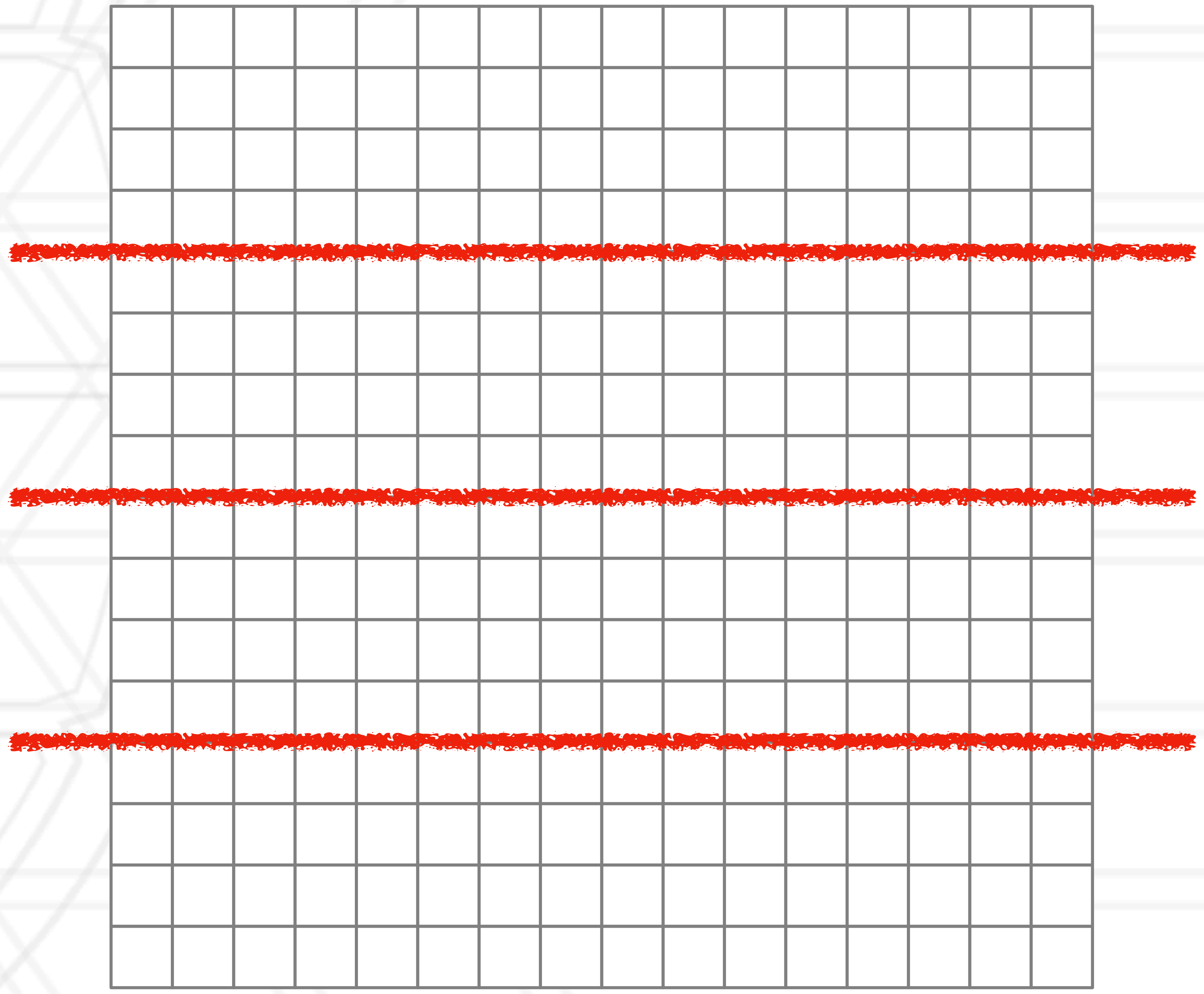
For correctness, we have to ensure that elements in A are not written into before they are read in the same timestep / iteration

2D stencil computation in parallel



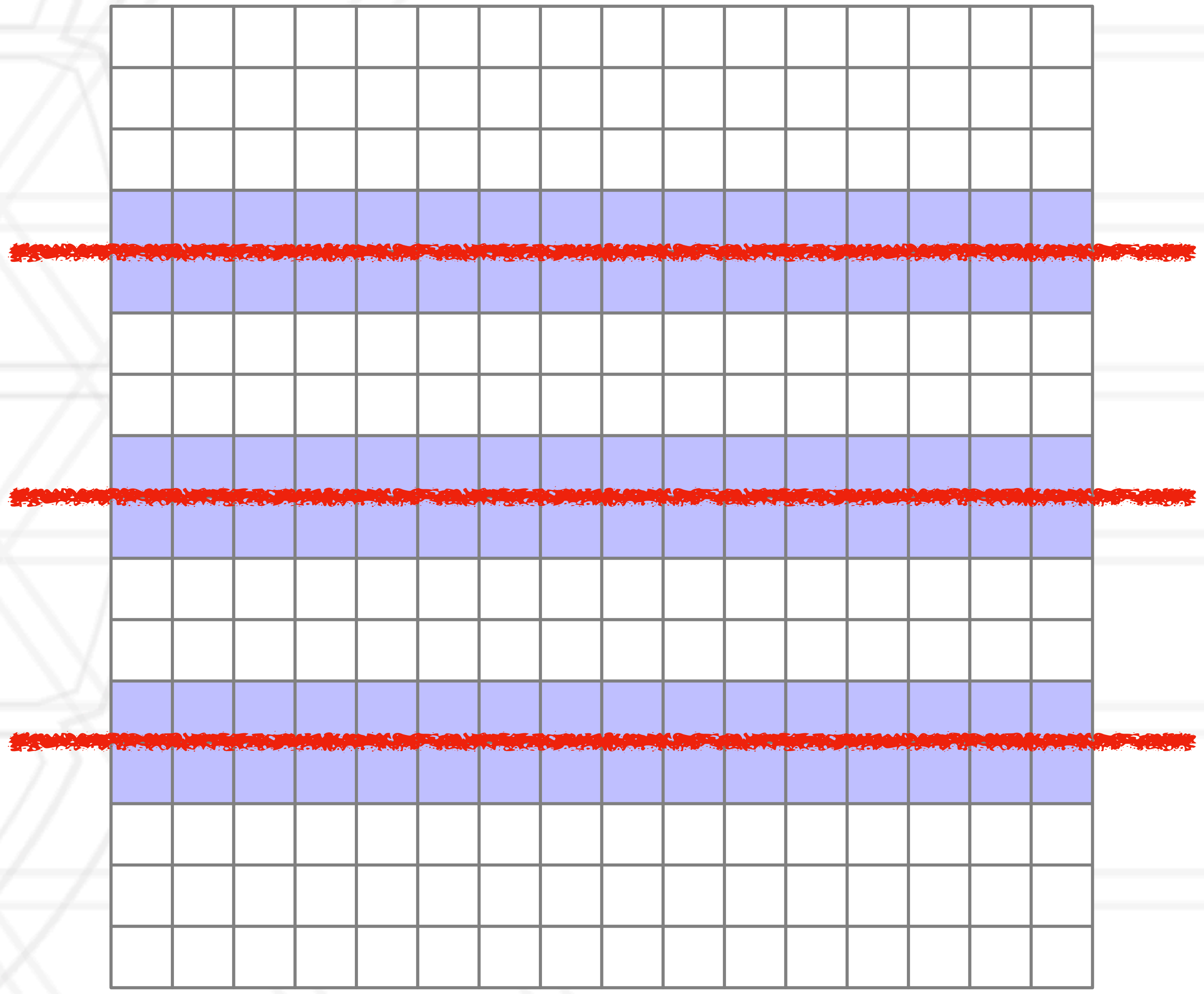
2D stencil computation in parallel

- 1D decomposition
  - Divide rows (or columns) among processes
  - Each process has to communicate with two neighbors (above and below)



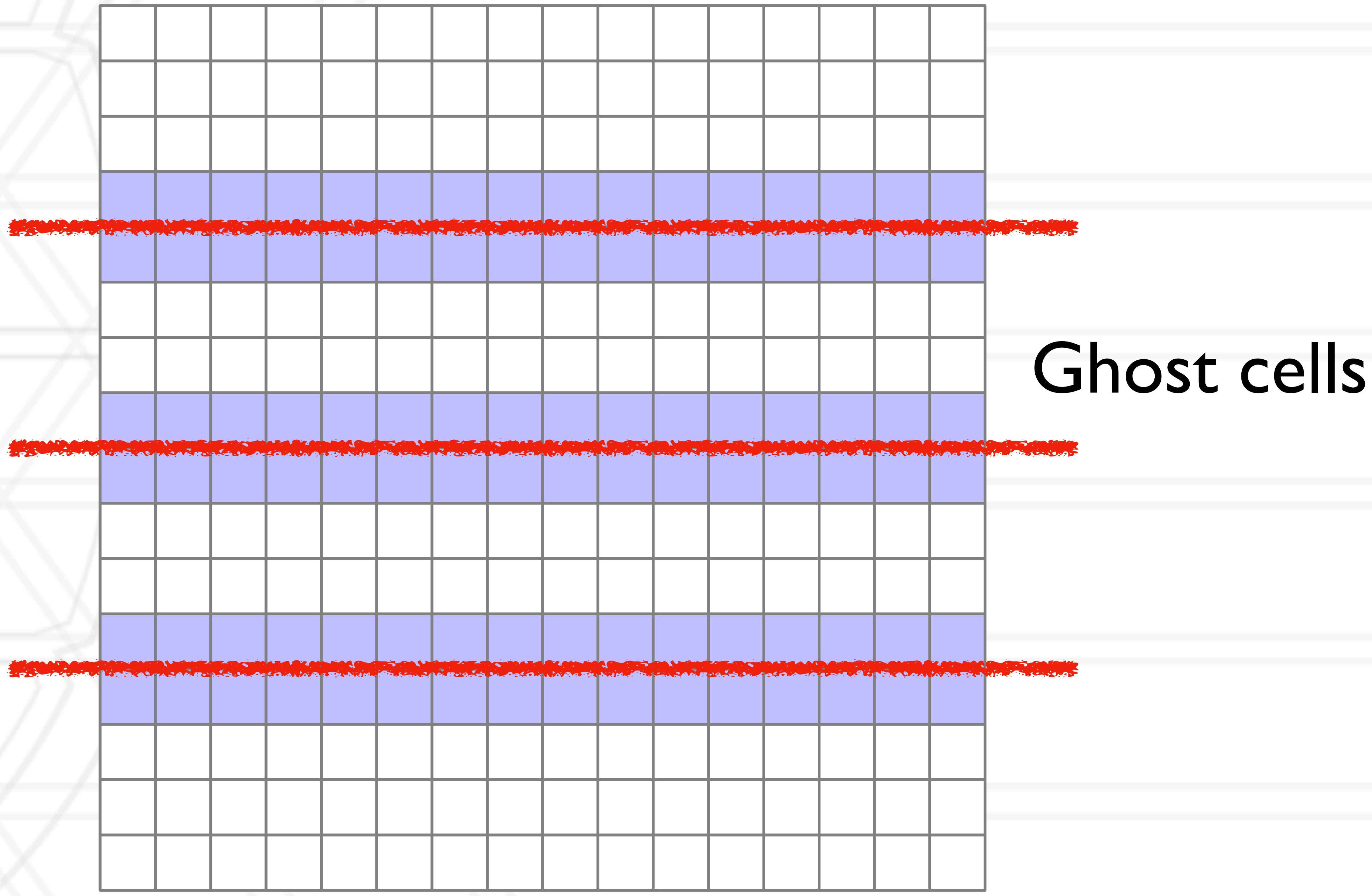
2D stencil computation in parallel

- 1D decomposition
  - Divide rows (or columns) among processes
  - Each process has to communicate with two neighbors (above and below)



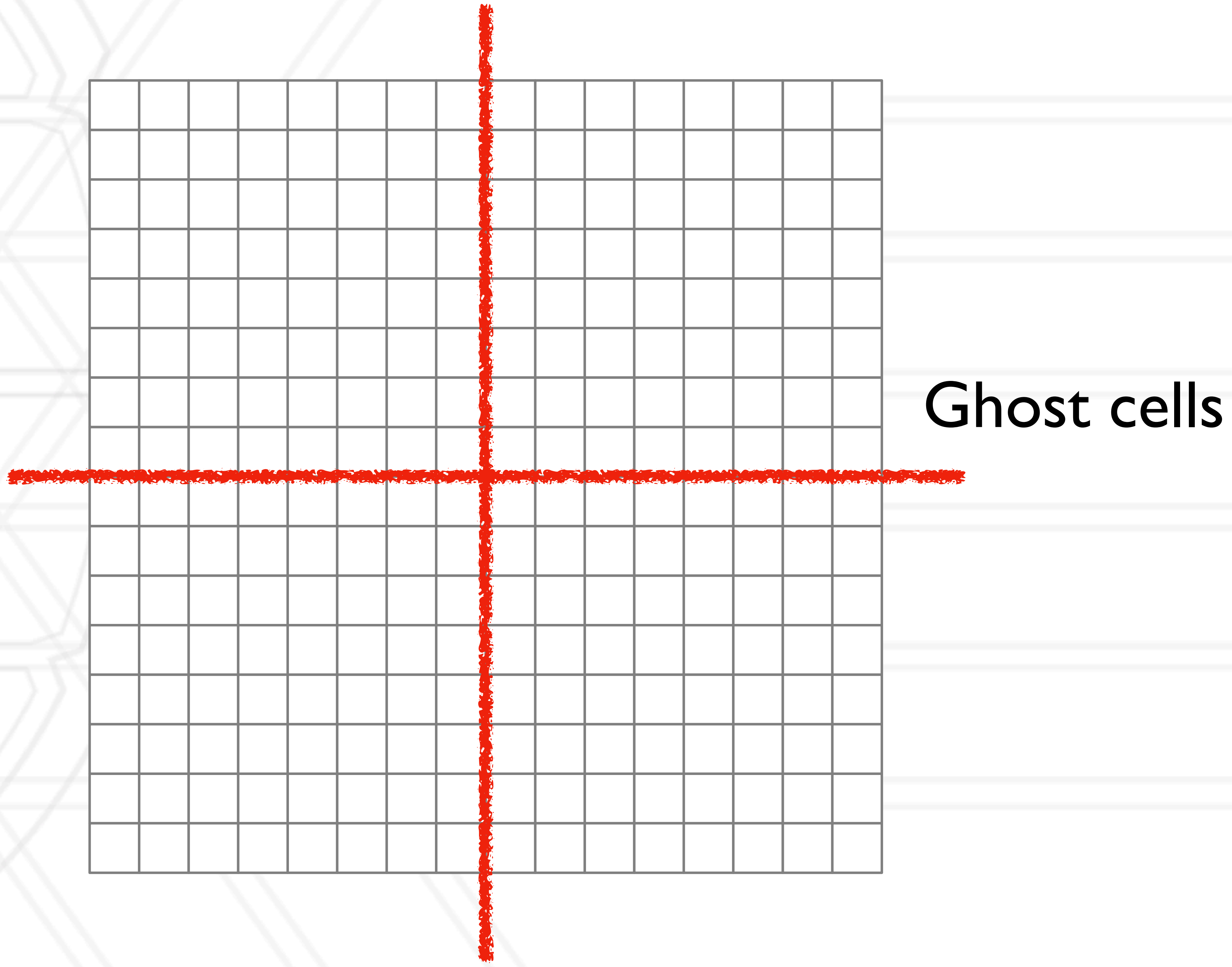
2D stencil computation in parallel

- 1D decomposition
  - Divide rows (or columns) among processes
  - Each process has to communicate with two neighbors (above and below)



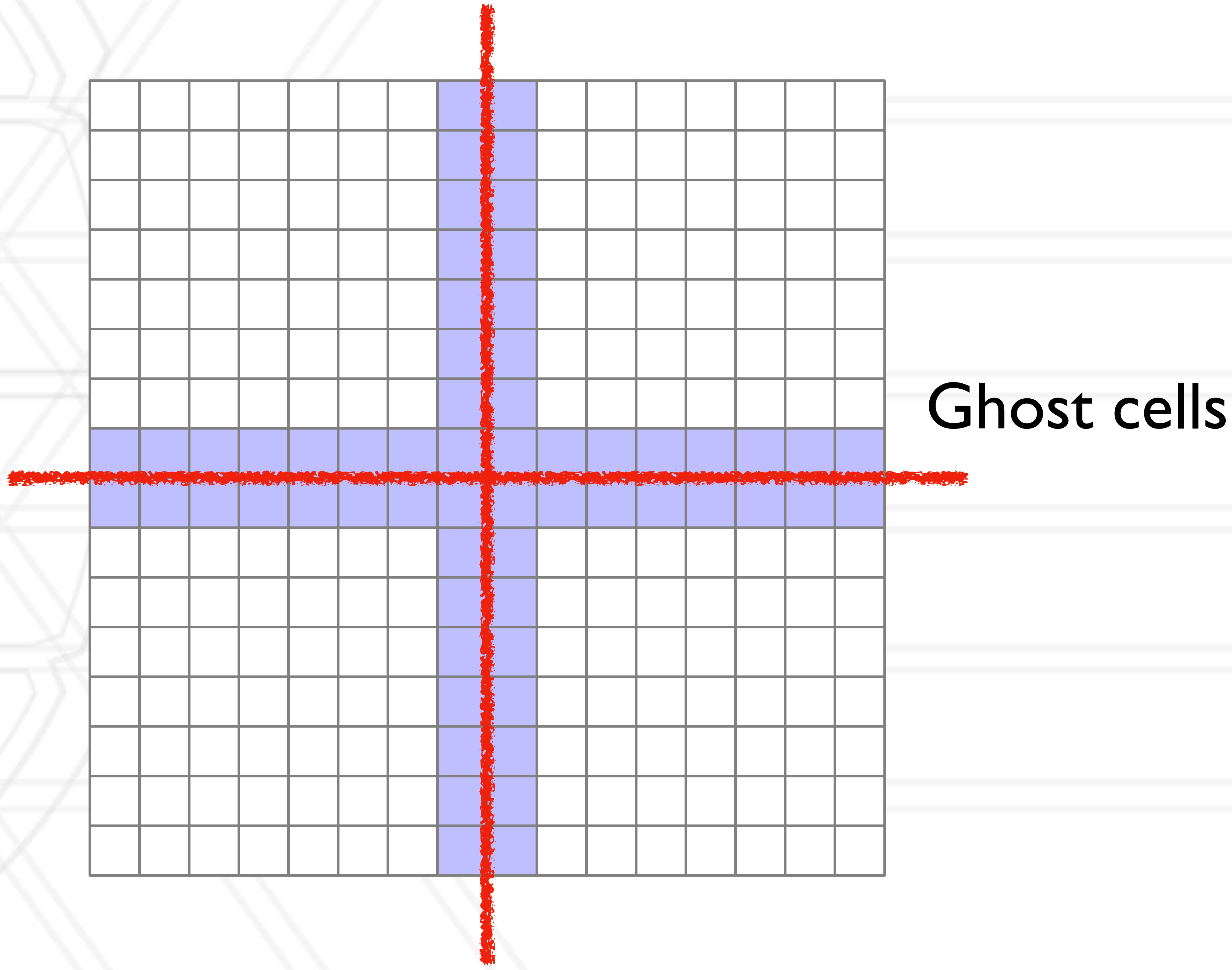
2D stencil computation in parallel

- 1D decomposition
  - Divide rows (or columns) among processes
  - Each process has to communicate with two neighbors (above and below)
- 2D decomposition
  - Divide both rows and columns (2d blocks) among processes
  - Each process has to communicate with four neighbors



2D stencil computation in parallel

- 1D decomposition
  - Divide rows (or columns) among processes
  - Each process has to communicate with two neighbors (above and below)
- 2D decomposition
  - Divide both rows and columns (2d blocks) among processes
  - Each process has to communicate with four neighbors



Message passing parallel code

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

  double A[N/p][M];
  double A_new[N/p][M];

  // send and receive data from neighbors

  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

 ...
}
```

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	…

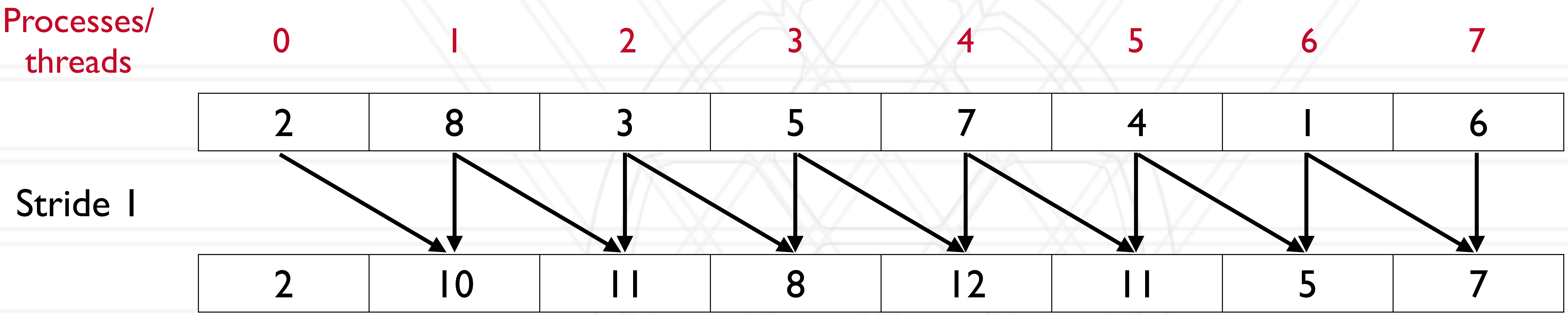
Parallel prefix sum

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

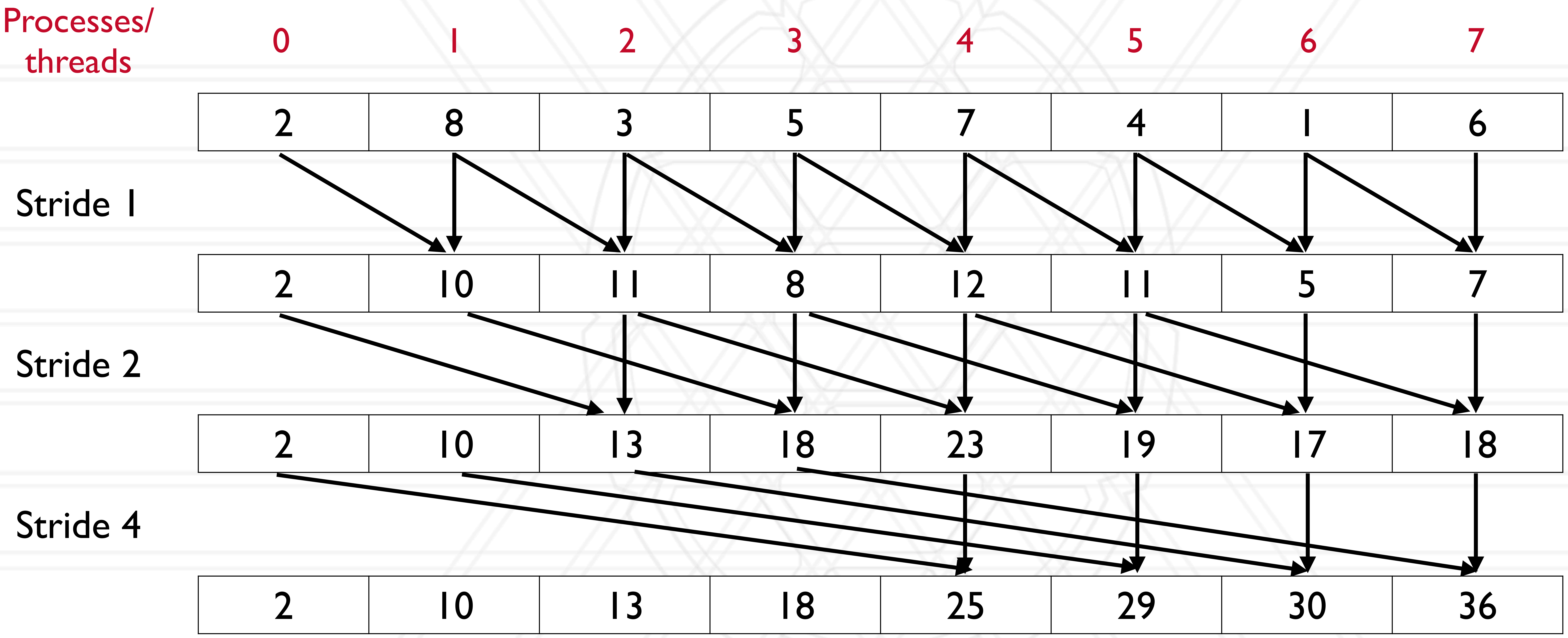
Parallel prefix sum

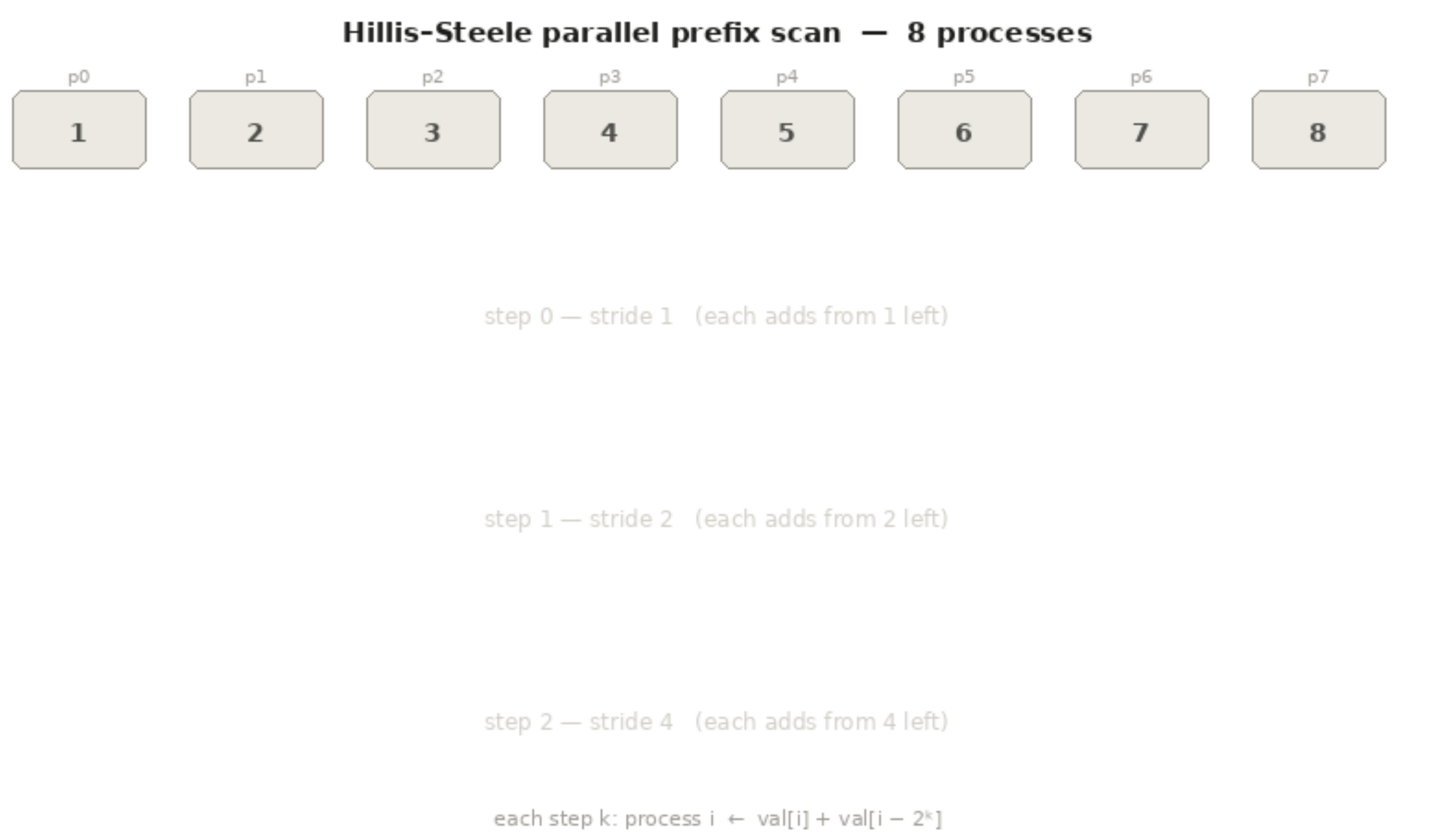


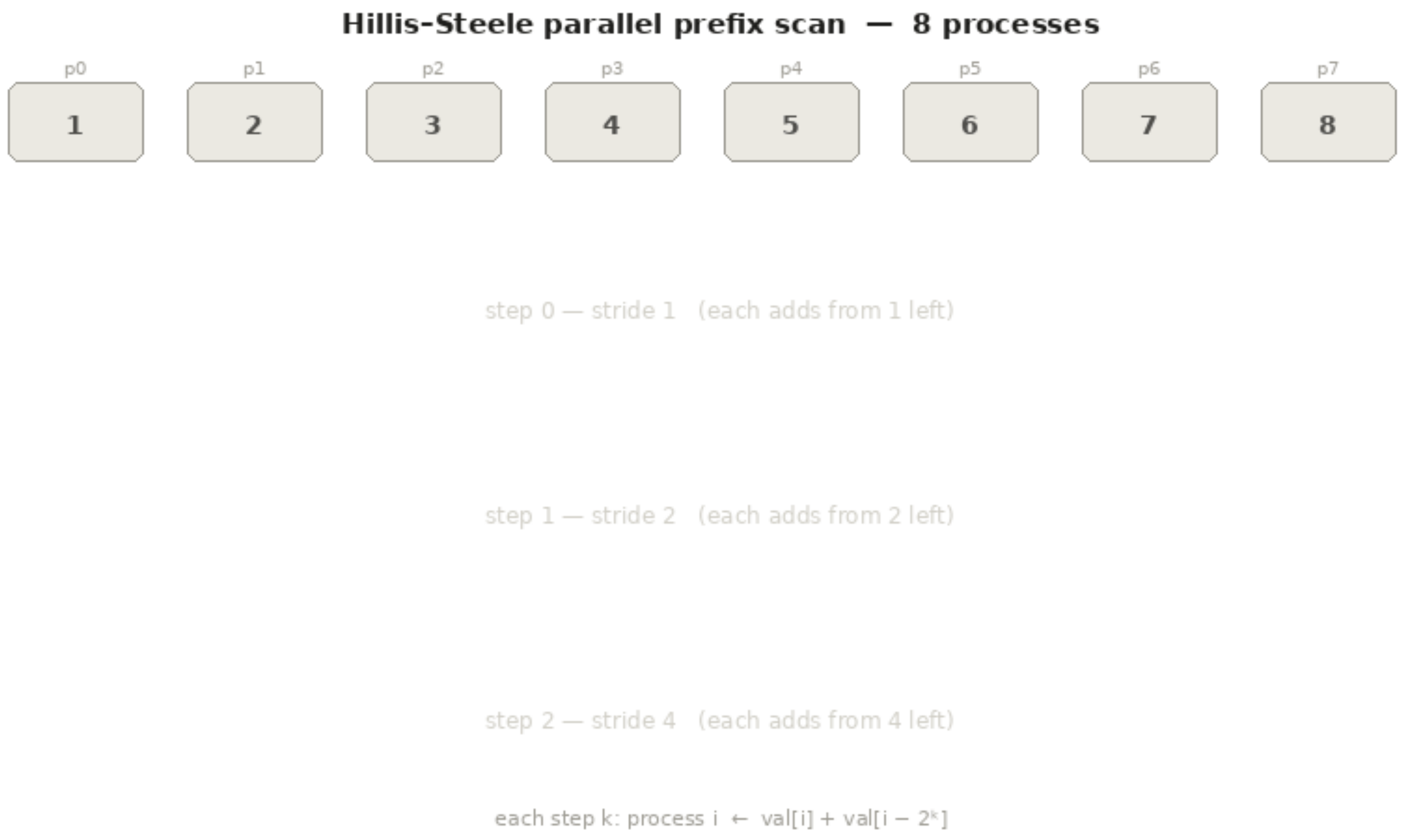
Parallel prefix sum

Processes/ threads	0	1	2	3	4	5	6	7
	2	8	3	5	7	4	1	6
Stride 1								
	2	10	11	8	12	11	5	7
Stride 2								
	2	10	13	18	23	19	17	18

Parallel prefix sum




Hillis-Steele parallel prefix scan — 8 processes
p0
p1
p2
p3
p4
p5
p6
p7
1
2
3
4
5
6
7
8
step 0 — stride 1 (each adds from 1 left)
step 1 — stride 2 (each adds from 2 left)
step 2 — stride 4 (each adds from 4 left)
each step k: process i ← val[i] + val[i − 2ᵏ]


Hillis-Steele parallel prefix scan — 8 processes
p0
1
p1
2
p2
3
p3
4
p4
5
p5
6
p6
7
p7
8
step 0 — stride 1 (each adds from 1 left)
step 1 — stride 2 (each adds from 2 left)
step 2 — stride 4 (each adds from 4 left)
each step k: process i ← val[i] + val[i − 2^k]

In practice

In practice

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

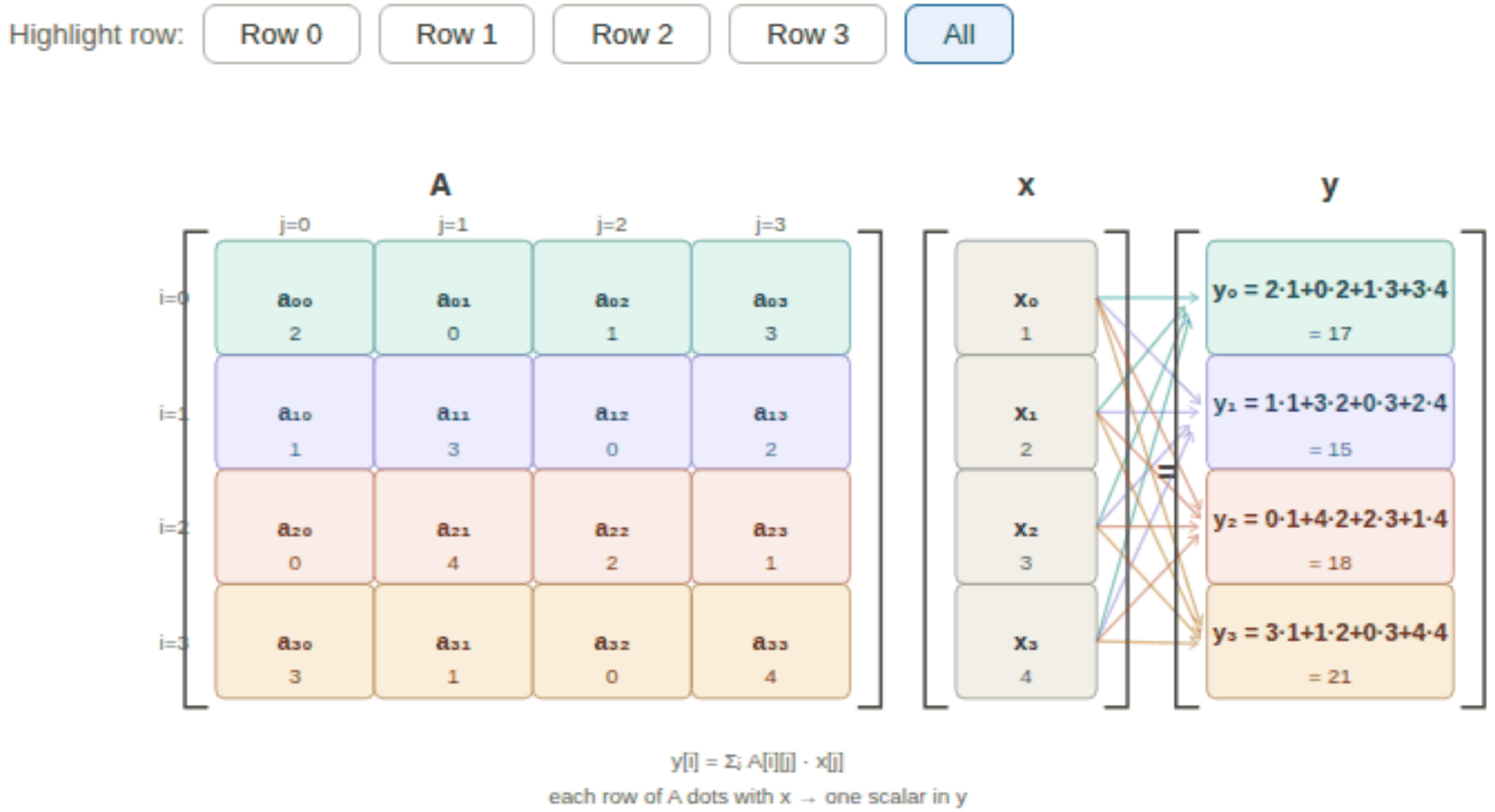
In practice

- 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

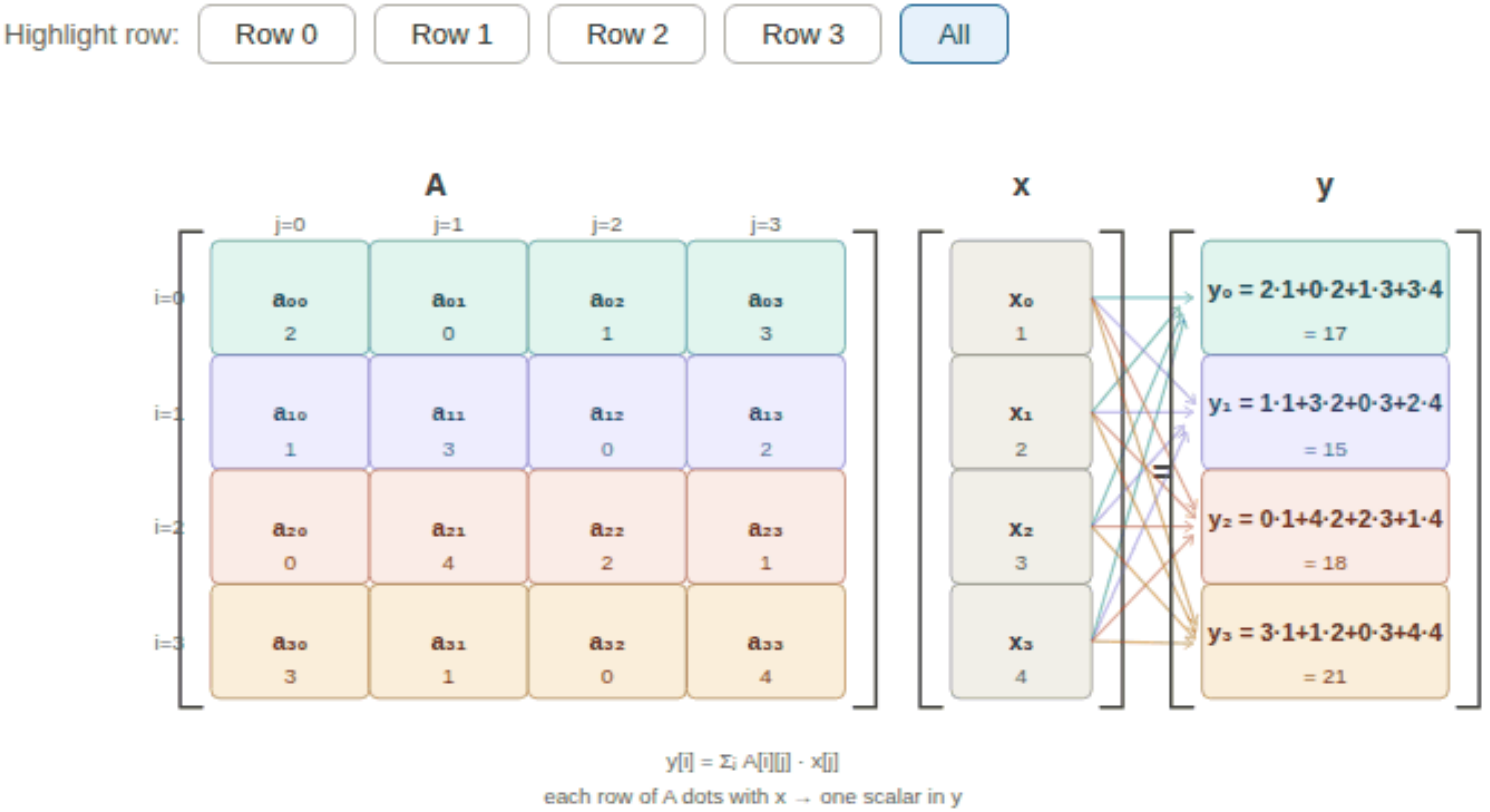
In practice

- 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

Matrix-vector product



Matrix-vector product



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