COMP90025 Parallel and Multicore Computing

Systolic Algorithms

Aaron Harwood

School of Computing and Information Systems
The University of Melbourne

2019 Semester II

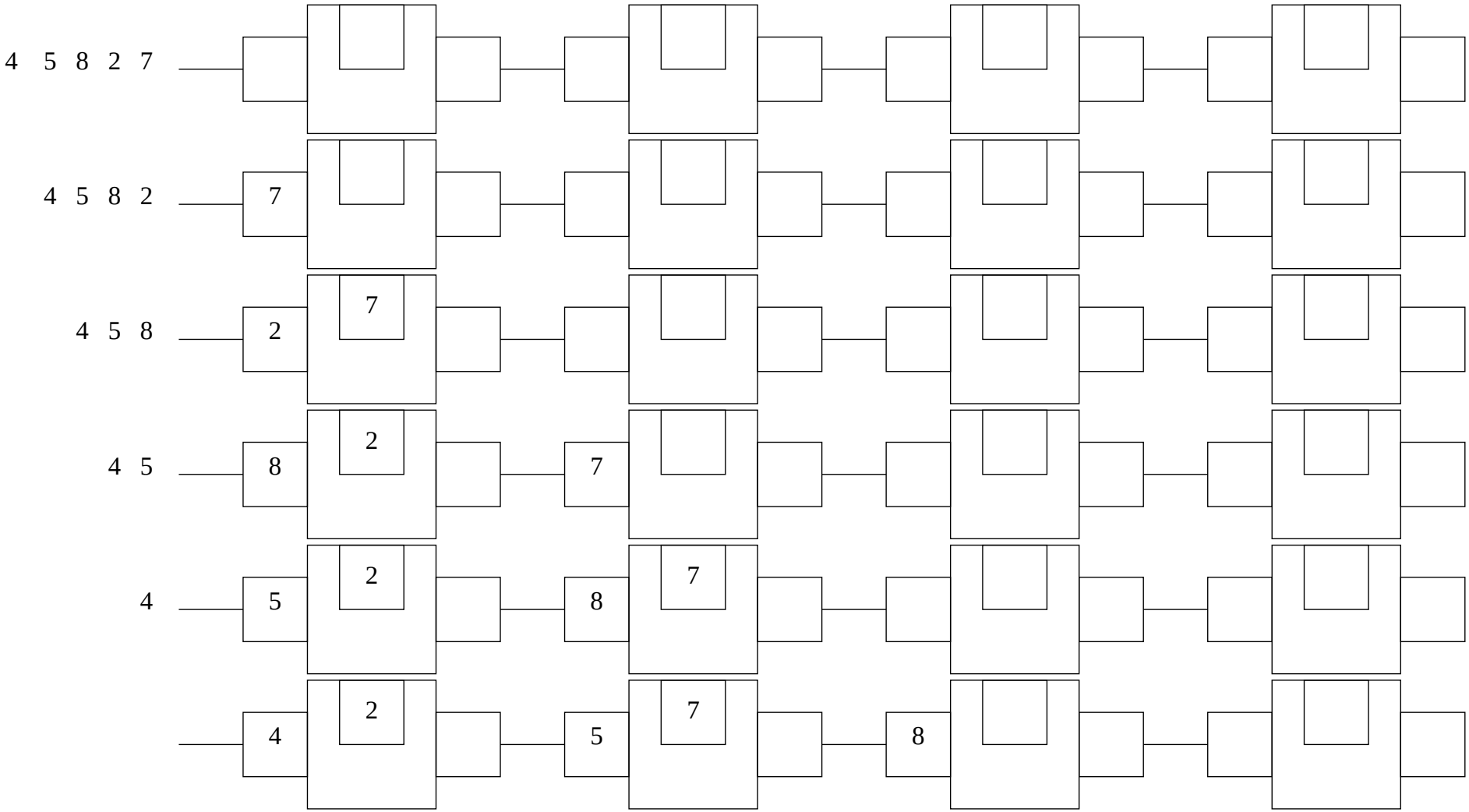
Systolic Algorithms

Systolic algorithms consider the flow of data through an array of processing elements. The processing elements themselves do not have access to any global memory, but rather just to values from adjacent processing elements.
From an implementation perspective, each processing element consists of input buffers, output buffers and local memory. Flow of data and computation takes place in two phases:

1. Processing elements read from their input buffers and local memory, undertake some computation and write results to their output buffers.
2. Output buffers are copied across to the input buffers of adjacent processing elements.

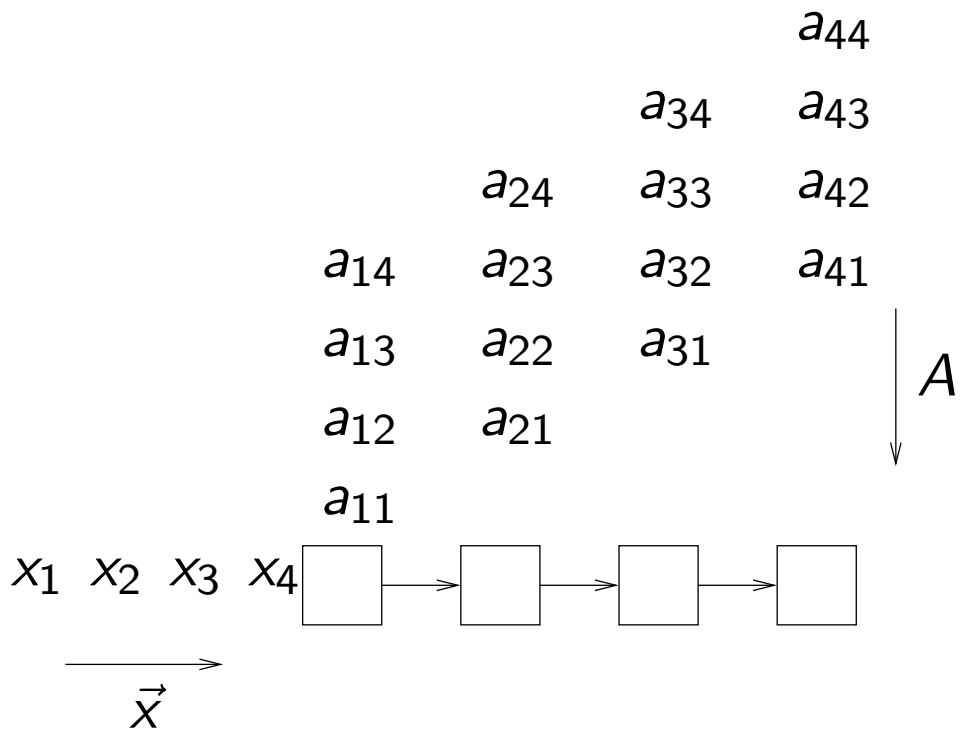
Sorting on a systolic linear array

Only the computation phase is shown below. A speedup of $\Theta(\log N)$ is achieved using an N-cell linear array.



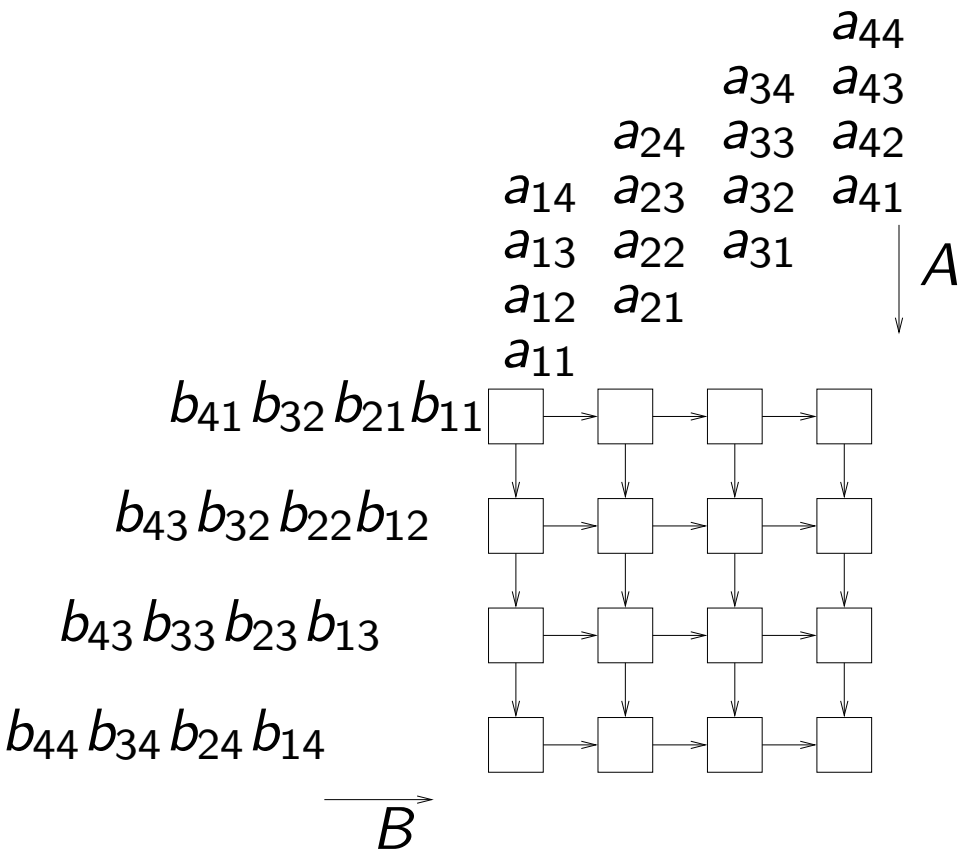
Matrix-vector multiplication

Computing a matrix-vector product $\vec{y} = A\vec{x}$ in seven steps on a 4-cell linear array. The values of $\vec{x}$ move rightward after each multiply/add step. A speedup of $\Theta(N)$ is achieved using an N-cell linear array.



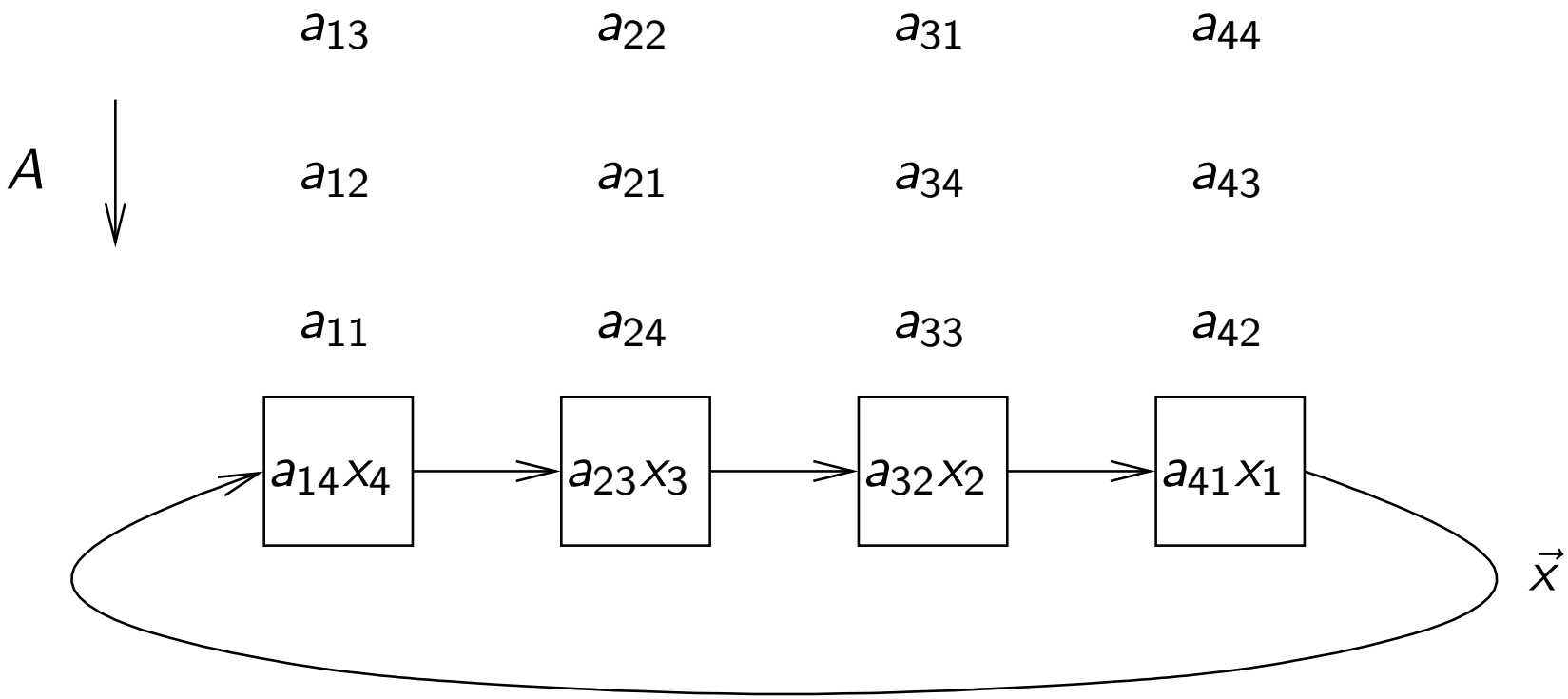
Matrix-matrix multiplication

Computing a matrix-matrix product $C = A \times B$ in 10 steps on a 4×4 mesh. A speedup of $\Theta(N^2)$ is achieved using an $N \times N$ mesh.



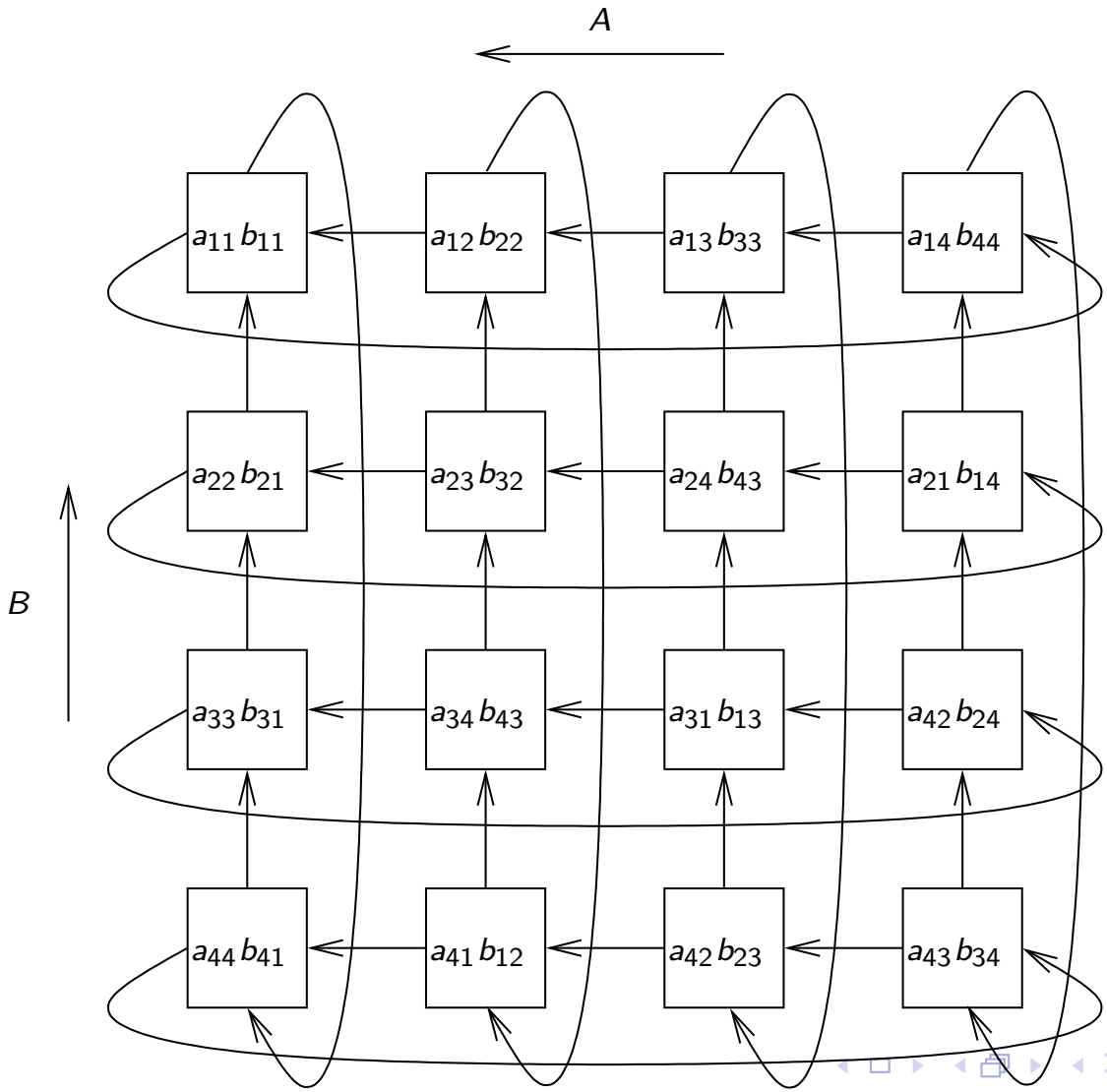
Matrix-vector multiplication

The first step in computing a matrix-vector product $\vec{y} = A\vec{x}$ in four steps on a 4-cell ring. The values of $\vec{x}$ move rightward after each multiply/add step.



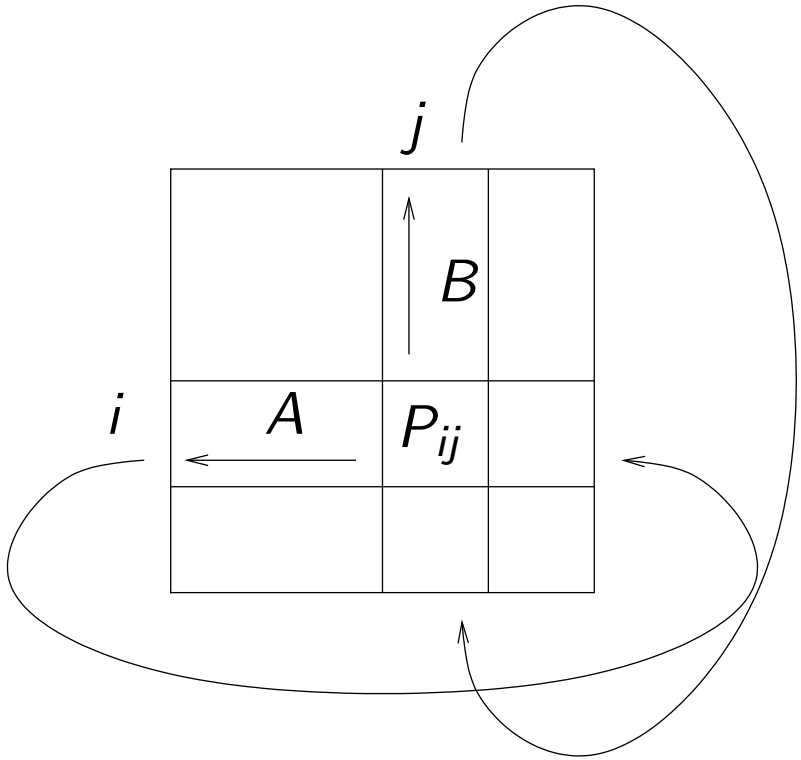
Matrix-matrix multiplication

Computing a matrix-matrix product $C = A \times B$ in 4 steps on a 4×4 torus. A speedup of $\Theta(N^2)$ is achieved using an $N \times N$ mesh.



Cannon's algorithm

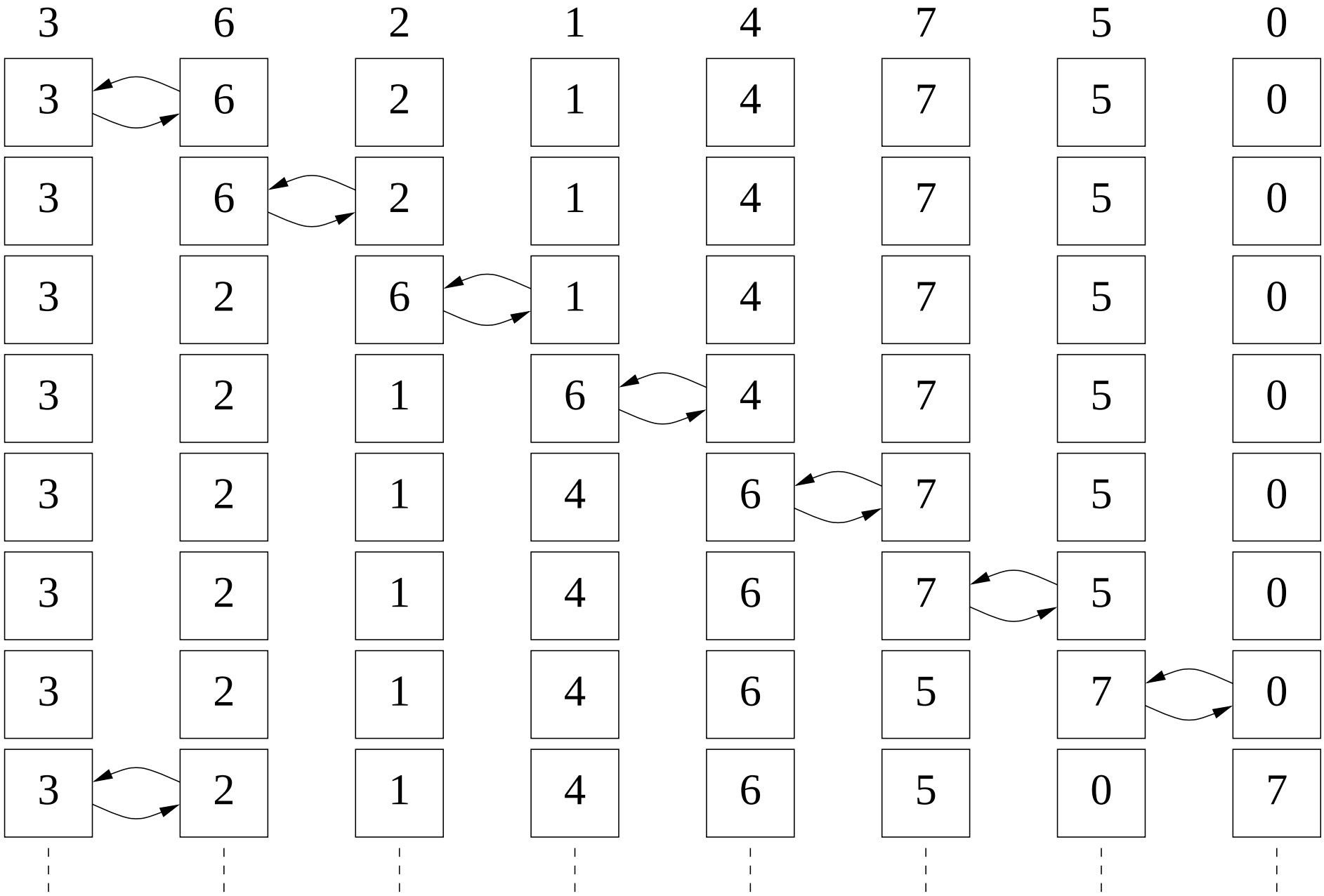
Cannon's algorithm (1969) describes the matrix-matrix multiplication on a torus.
Cannon's algorithm uses a $N \times N$ torus of processors. Processor (i, j) at location (i, j) initially begins with elements or submatrices a_{ij} and b_{ij}, for $i, j = 1, 2, \ldots, N$. As the algorithm progresses, the submatrices are passed left and upwards.


j
B
i
A
P_{ij}

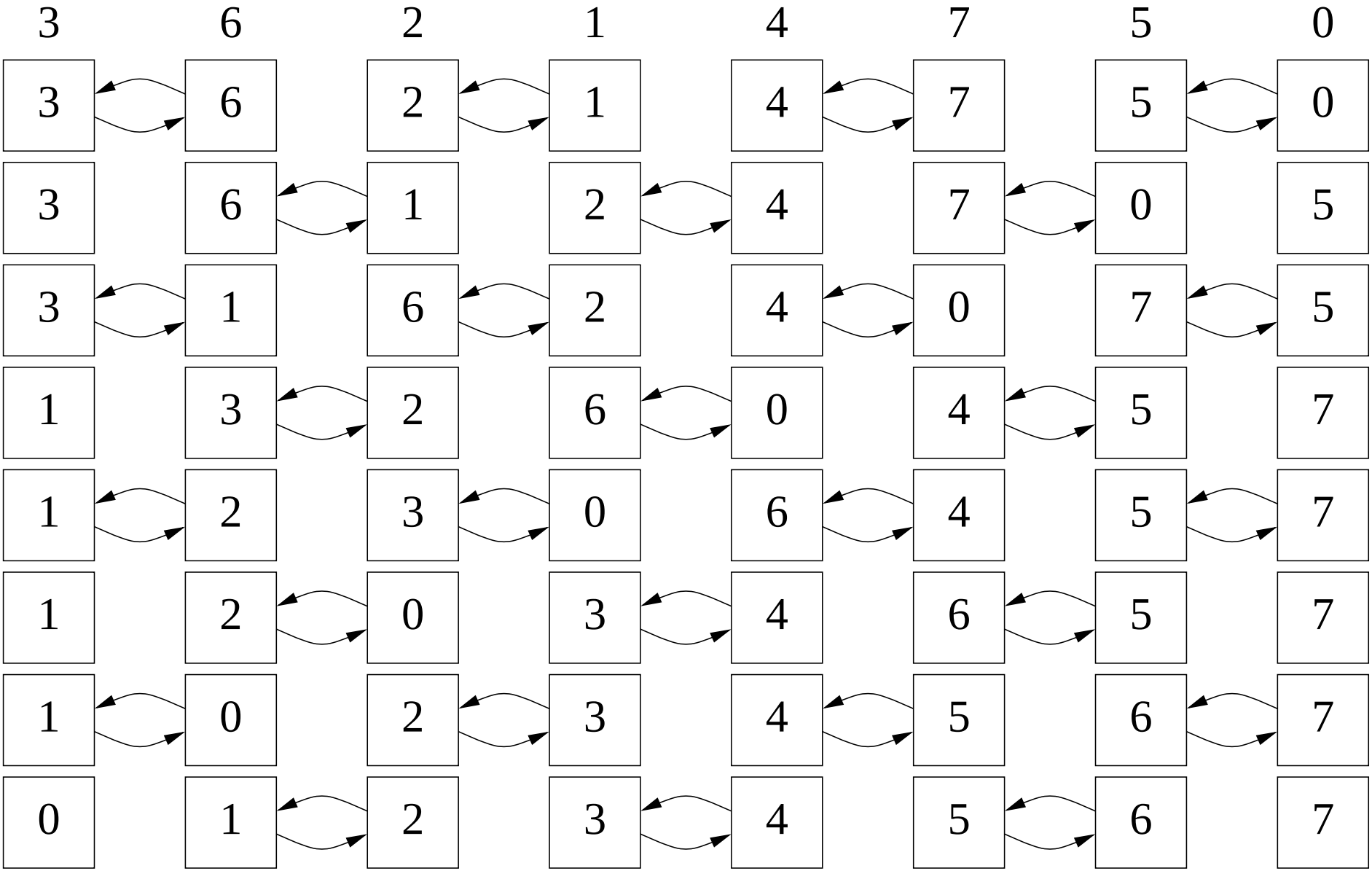
1. Initially P_{ij} begins with a_{ij} and b_{ij}.
2. Elements are moved from their initial positions to align them so that the correct submatrices are multiplied with one another. Note that submatrices on the diagonal don't actually require alignment. Alignment is done by shifting the i-th row of A $i-1$ positions left and the j-th column of B $j-1$ positions up.
3. Each processor, P_{ij} multiplies its current submatrices and adds to a cumulative sum.
4. The submatrices are shifted left and upwards.
5. The above two steps are repeated through the remaining submatrices.

Odd-even Transposition Sort

Recall the operation of a bubble sort algorithm. For n elements, a first phase iterates through the elements, comparing and exchanging neighbors, with $n-1$ operations. After the first phase the largest element will have "bubbled" to the end of the array.
Since the largest element appears at the end of the array after the first phase, the second phase need only iterate $n-2$ times. The bubble sort however takes $O(n^2)$ sequential steps.


3 6 2 1 4 7 5 0
3 6 2 1 4 7 5 0
3 6 2 1 4 7 5 0
3 2 6 1 4 7 5 0
3 2 1 6 4 7 5 0
3 2 1 4 6 7 5 0
3 2 1 4 6 7 5 0
3 2 1 4 6 5 7 0
3 2 1 4 6 5 0 7

The odd-even transposition sort makes use of a pipelining technique to ultimately run many phases of the bubble sort in parallel.
The second phase of the bubble sort algorithm can begin after the second iteration of the first phase.
The third phase can begin after the fourth iteration, the fourth phase after the sixth iteration and so on.
In effect each parallel computational step can pair off either the odd or even neighboring pairs.


3 6 2 1 4 7 5 0
3 6 2 1 4 7 5 0
3 6 1 2 4 7 0 5
3 1 6 2 4 0 7 5
1 3 2 6 0 4 5 7
1 2 3 0 6 4 5 7
1 2 0 3 4 6 5 7
1 0 2 3 4 5 6 7
0 1 2 3 4 5 6 7

Assuming n is an even number.

Processor P_i, i is odd:

```
send(A, Pi-1);
recv(B, Pi-1);
if(A<B) A=B;
if(i<=n-3){
send(A, Pi+1);
recv(B, Pi+1);
if(A>B) A=B;}
```

Processor P_i, i is even:

```
recv(A, Pi+1);
send(B, Pi+1);
if(A<B) B=A;
if(i>=2){
recv(A,Pi-1);
send(B,Pi-1);
if(A>B) B=A;}
```

Shear sort

The Shear sort is for mesh architectures that use message passing. The lower bound for a mesh architecture sort is $O(\sqrt{n})$ since it takes at most $2(\sqrt{n}-1)$ steps in the worst case to move a number from one node to another.
The Shear sort uses $\log_2 n + 1$ phases. In odd phases (1, 3, 5 ...) the following is done:

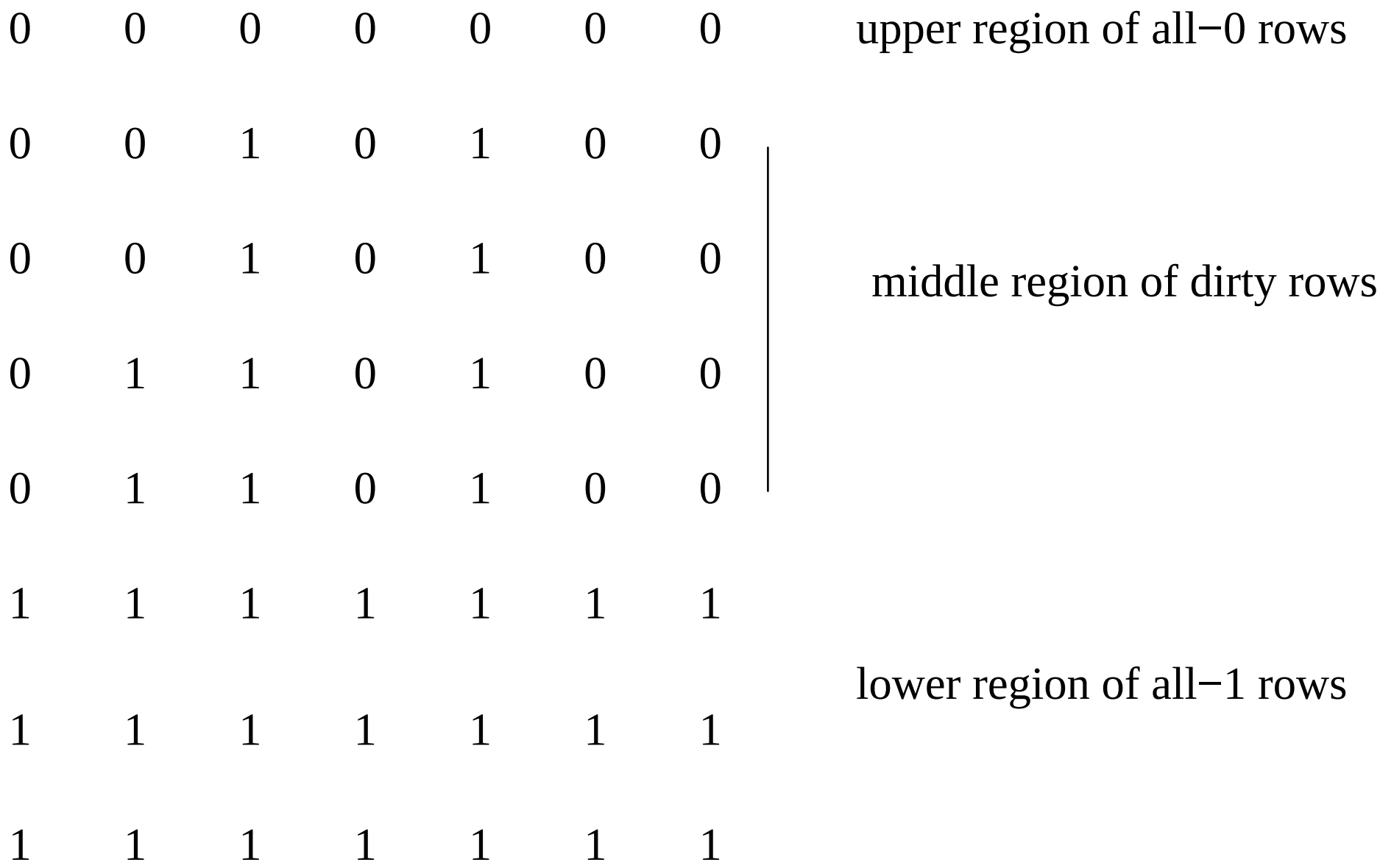
- even rows – the row is sorted with the smallest number placed at the right.
- odd rows – the row is sorted with the smallest number placed at the left.

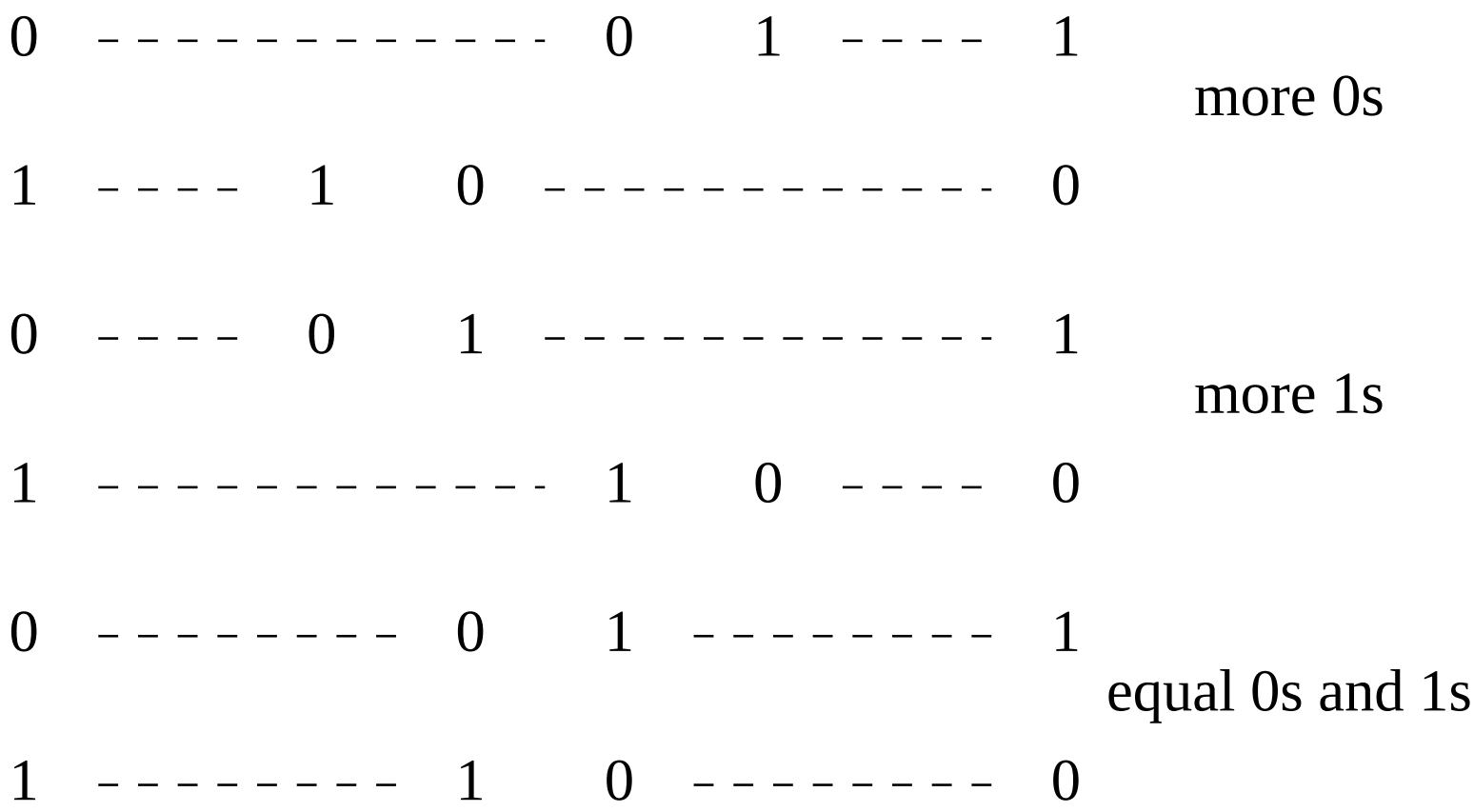
In even phases (2, 4, 6, ...) the following is done:

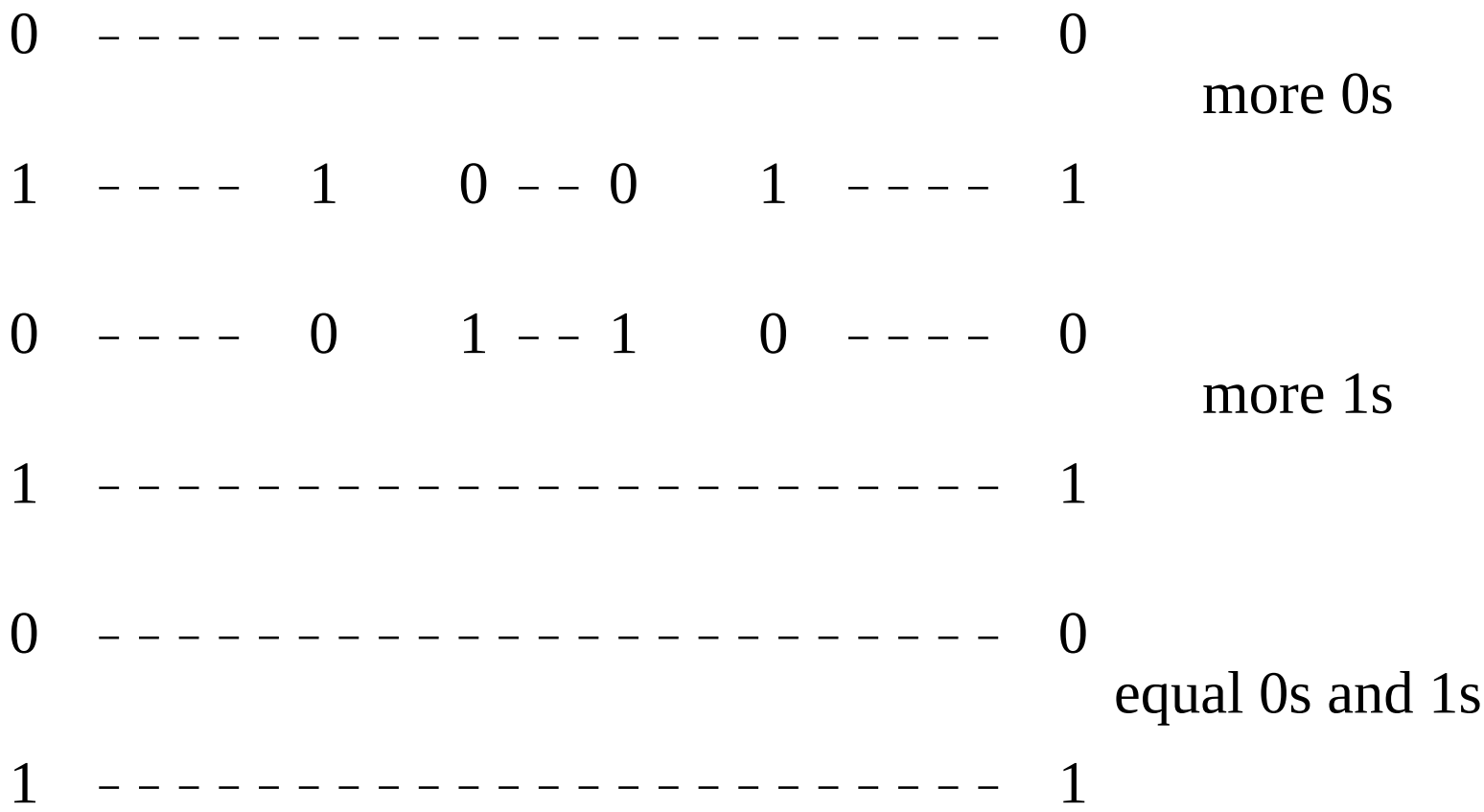
- each column of number is sorted independently with the smallest number placed at the top.

Sorting of rows and columns requires $\sqrt{n}$ time to complete using for example the odd-even transposition sort. The total time is then $\sqrt{n}(\log_2 n + 1)$.
There are other algorithms that reach the lower bound for meshes.

Shear Sort Intuitive Proof




0
0
1
1
more 0s
1
1
0
0
0
0
1
1
more 1s
1
1
0
0
0
0
1
1
equal 0s and 1s
1
1
0
0


0
0
more 0s
1
1
0
0
1
1
0
0
more 1s
1
1
0
0
equal 0s and 1s
1
1

Since at least one row in each pair becomes all-0 or all-1 and is moved out of the middle region after sorting the rows and columns, the middle region decreases in size by at least one-half for each pair of phases. Hence after $2 \log \sqrt{N} = \log N$ phases, the numbers are sorted, except for one row, and the algorithm is concluded by sorting this row in the last phase.