Lecture 14: Parallel Algorithms

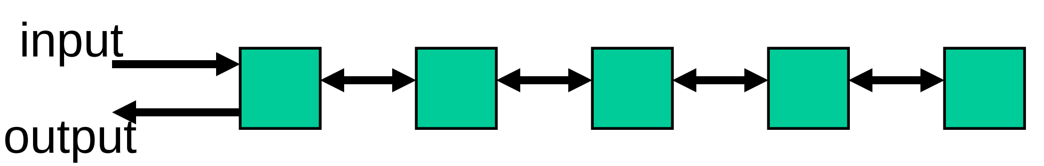
- Topics: sort, matrix, graph algorithms

Processor Model

- High communication latencies → pursue coarse-grain parallelism (the focus of the course so far)
- For upcoming lectures, focus on fine-grain parallelism
- VLSI improvements → enough transistors to accommodate numerous processing units on a chip and (relatively) low communication latencies
- Consider a special-purpose processor with thousands of processing units, each with small-bit ALUs and limited register storage

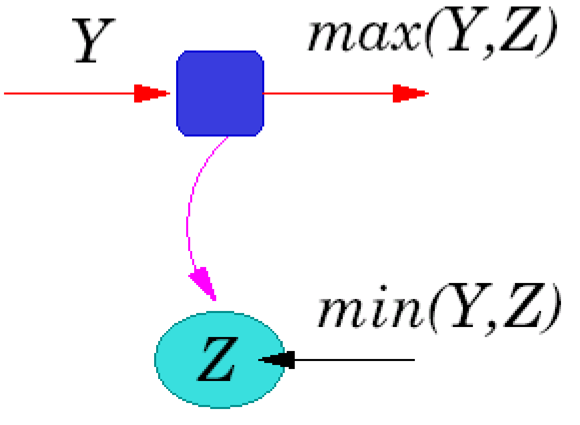
Sorting on a Linear Array

- Each processor has bidirectional links to its neighbors
- All processors share a single clock (asynchronous designs will require minor modifications)
- At each clock, processors receive inputs from neighbors, perform computations, generate output for neighbors, and update local storage

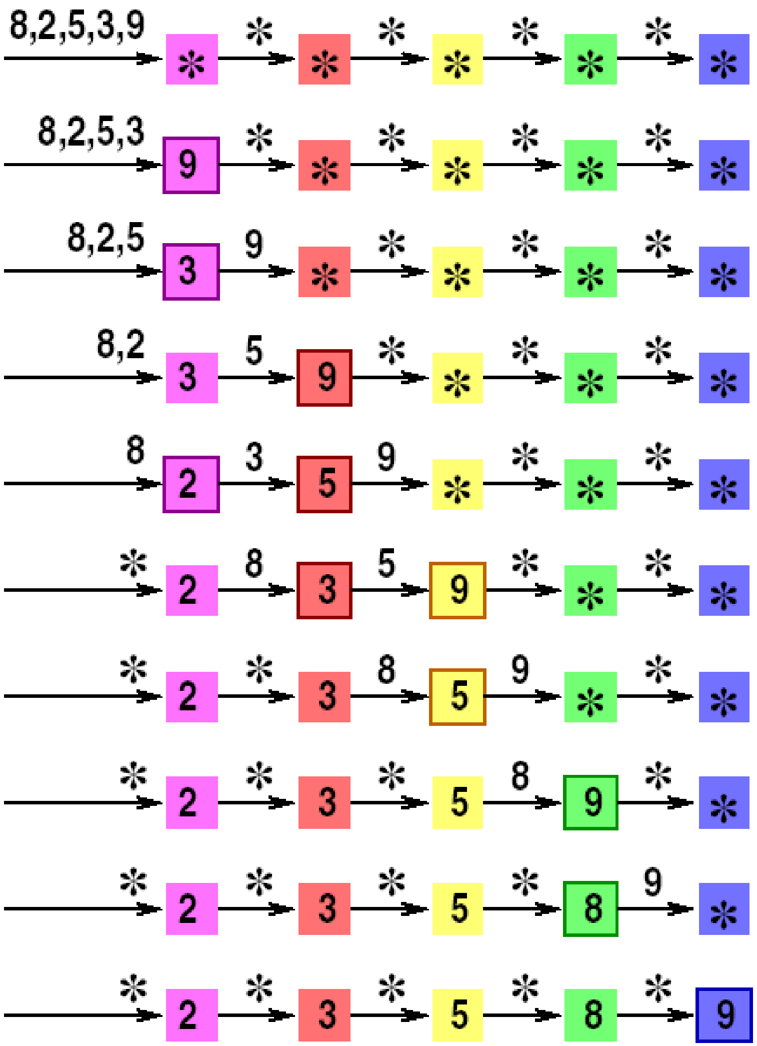


Control at Each Processor

- Each processor stores the minimum number it has seen
- Initial value in storage and on network is “$*$”, which is bigger than any input and also means “no signal”
- On receiving number Y from left neighbor, the processor keeps the smaller of Y and current storage Z, and passes the larger to the right neighbor



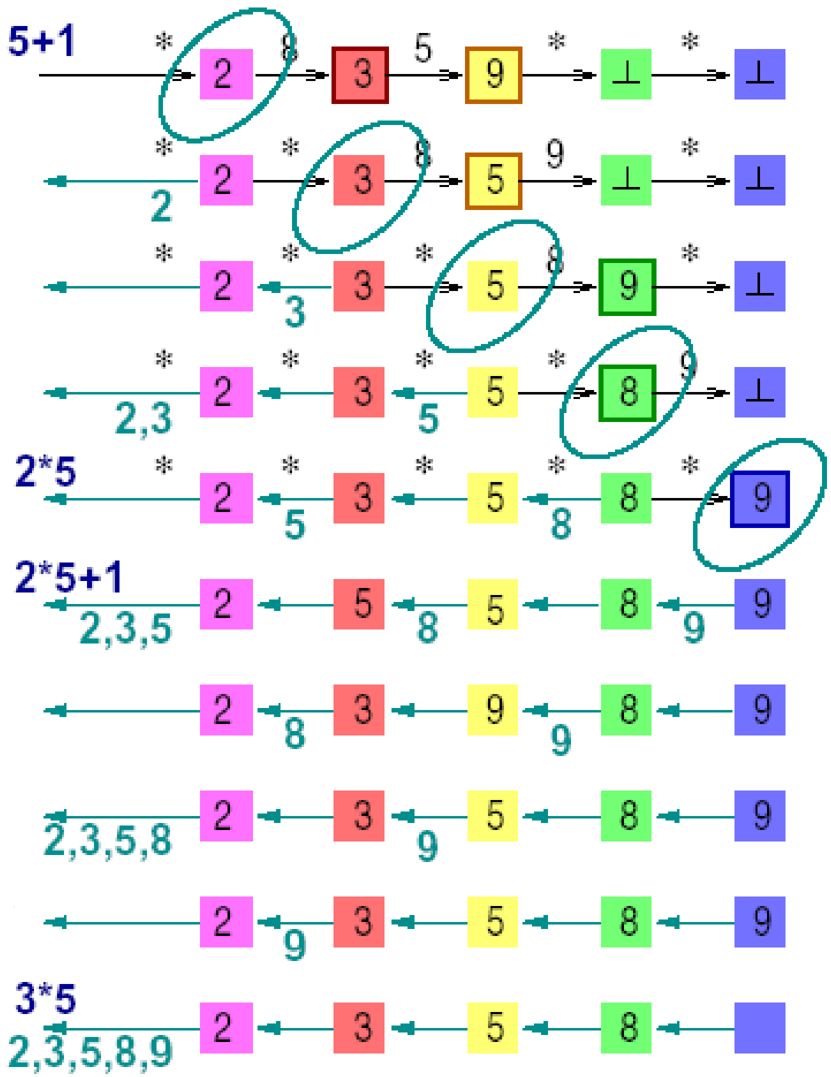
Sorting Example



Result Output

- The output process begins when a processor receives a non-$*$, followed by a "$*$"
- Each processor forwards its storage to its left neighbor and subsequent data it receives from right neighbors
- How many steps does it take to sort N numbers?
- What is the speedup and efficiency?

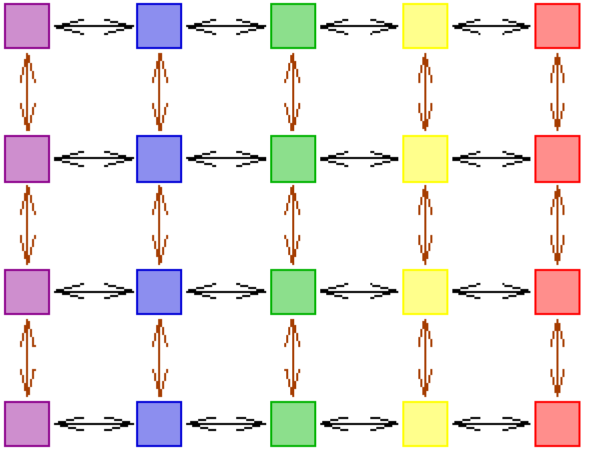
Output Example



Bit Model

- The bit model affords a more precise measure of complexity – we will now assume that each processor can only operate on a bit at a time

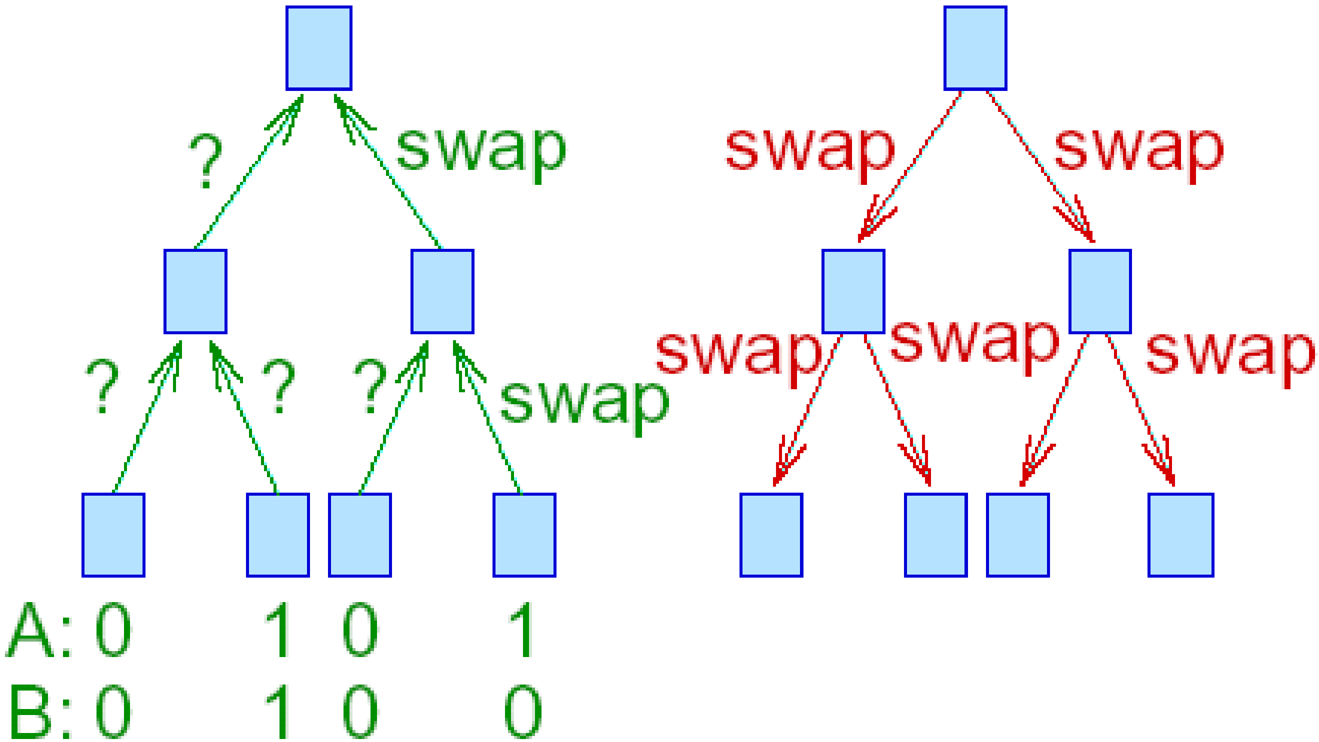
- To compare N k-bit words, you may now need an N x k 2-d array of bit processors



Comparison Strategies

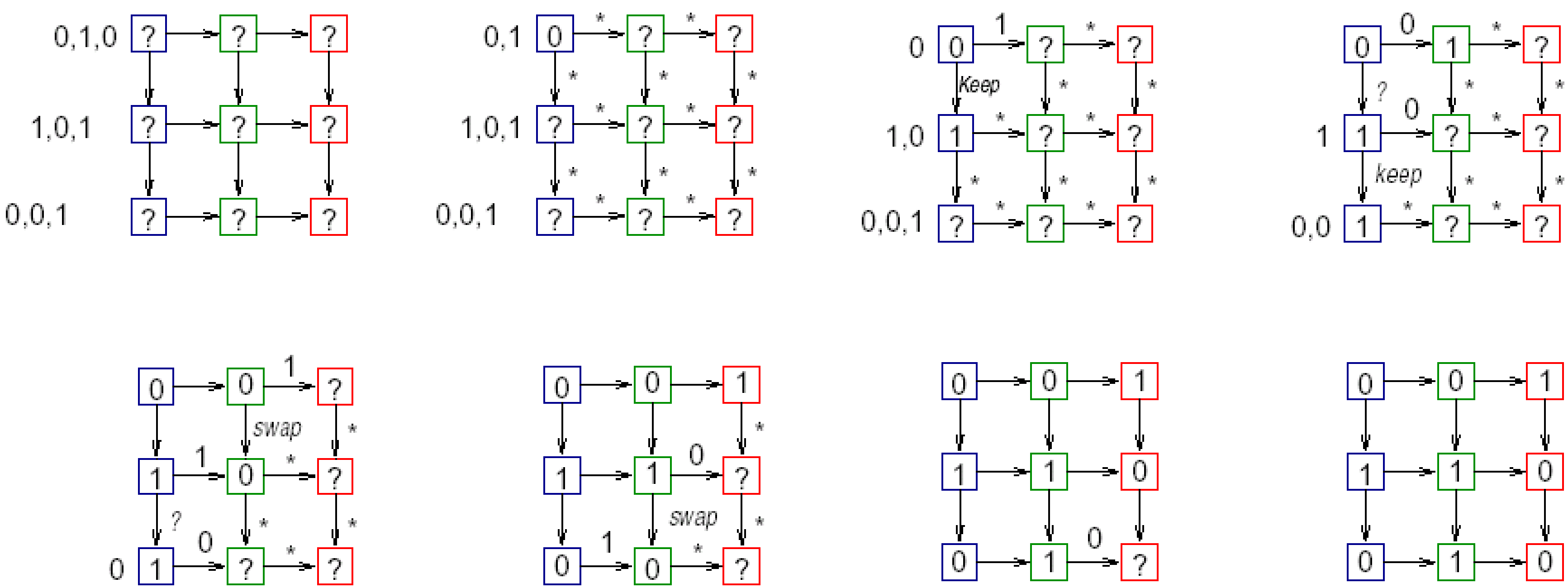
- Strategy 1: Bits travel horizontally, keep/swap signals travel vertically – after at most 2k steps, each processor knows which number must be moved to the right – 2kN steps in the worst case
- Strategy 2: Use a tree to communicate information on which number is greater – after 2logk steps, each processor knows which number must be moved to the right – 2Nlogk steps
- Can we do better?

Strategy 2: Column of Trees



Pipelined Comparison

Input numbers:			
	3	4	2
	0	1	0
	1	0	1
	1	0	0



Complexity

- How long does it take to sort N k-bit numbers?
  $(2N - 1) + (k - 1) + N$ (for output)
- (With a 2d array of processors) Can we do even better?
- How do we prove optimality?

Lower Bounds

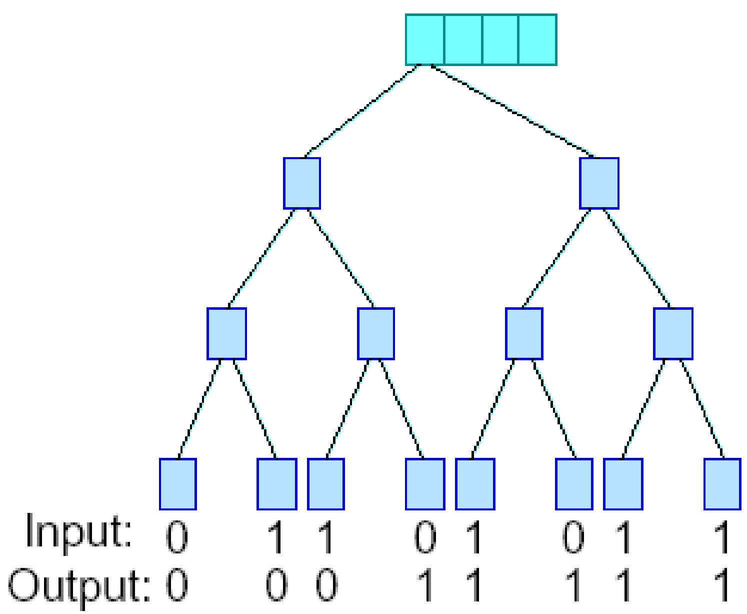
- Input/Output bandwidth: Nk bits are being input/output with k pins – requires Ω(N) time

- Diameter: the comparison at processor (1,1) influences the value of the bit stored at processor (N,k) – for example, N-1 numbers are 011..1 and the last number is either 00…0 or 10…0 – it takes at least N+k-2 steps for information to travel across the diameter

- Bisection width: if processors in one half require the results computed by the other half, the bisection bandwidth imposes a minimum completion time

Counter Example

- N 1-bit numbers that need to be sorted with a binary tree
- Since bisection bandwidth is 2 and each number may be in the wrong half, will any algorithm take at least N/2 steps?



Counting Algorithm

- It takes O(logN) time for each intermediate node to add the contents in the subtree and forward the result to the parent, one bit at a time

- After the root has computed the number of 1's, this number is communicated to the leaves – the leaves accordingly set their output to 0 or 1

- Each half only needs to know the number of 1's in the other half (logN-1 bits) – therefore, the algorithm takes Ω(logN) time

- Careful when estimating lower bounds!

Matrix Algorithms

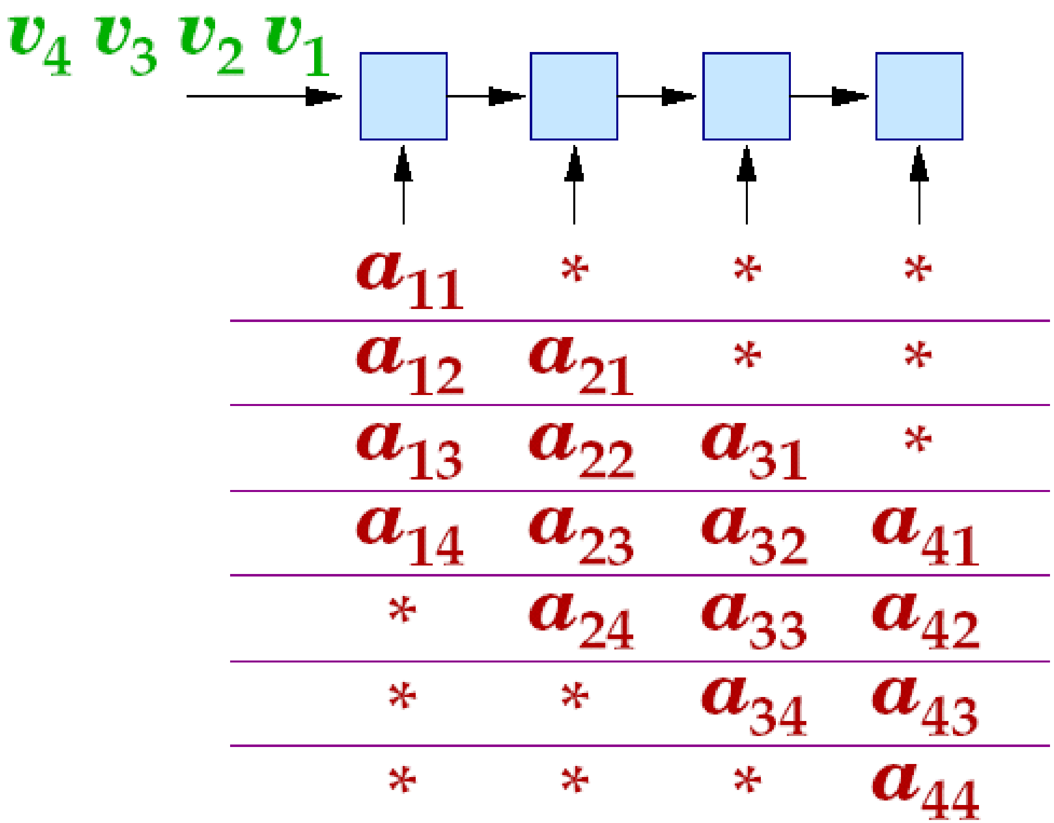
- Consider matrix-vector multiplication:

$$y_i = \Sigma_j\, a_{ij}x_j$$

- The sequential algorithm takes $2N^2 – N$ operations

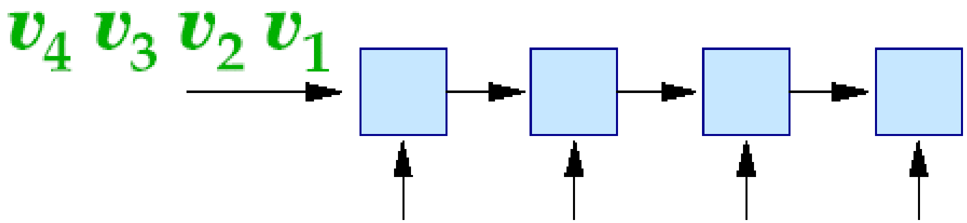
- With an N-cell linear array, can we implement matrix-vector multiplication in O(N) time?

Matrix Vector Multiplication



Number of steps = ?

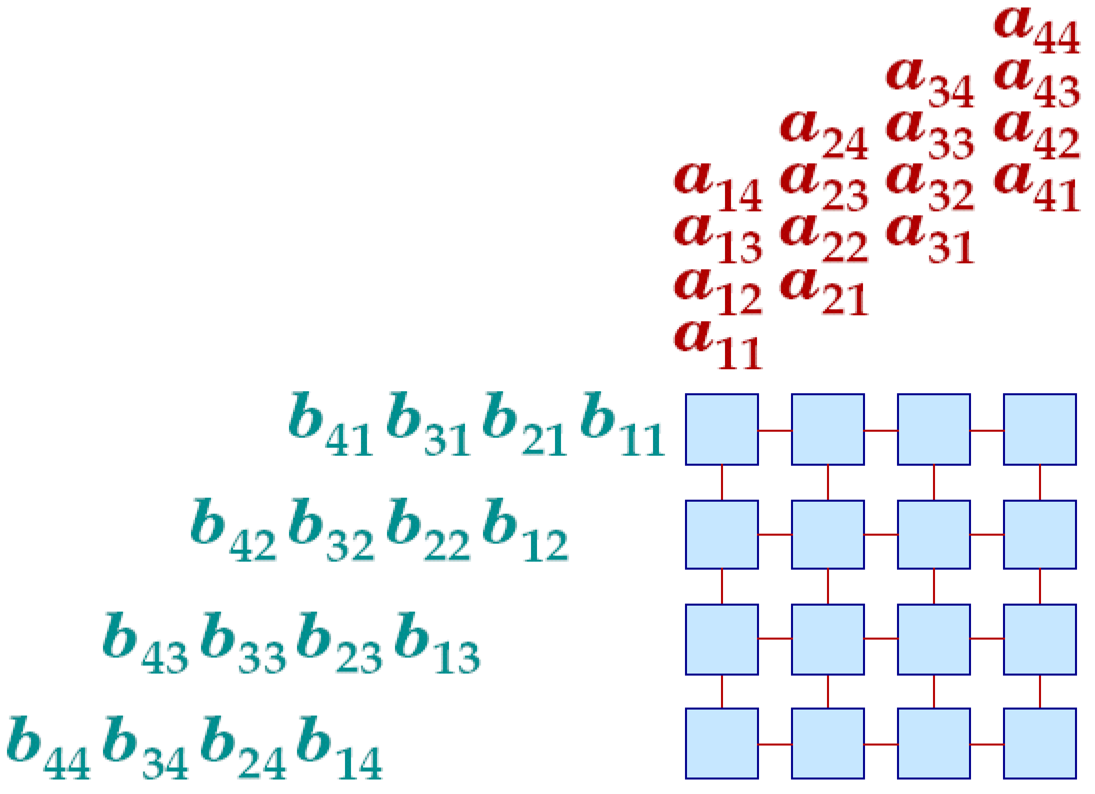
Matrix Vector Multiplication



a_{11}	*	*	*
a_{12}	a_{21}	*	*
a_{13}	a_{22}	a_{31}	*
a_{14}	a_{23}	a_{32}	a_{41}
*	a_{24}	a_{33}	a_{42}
*	*	a_{34}	a_{43}
*	*	*	a_{44}

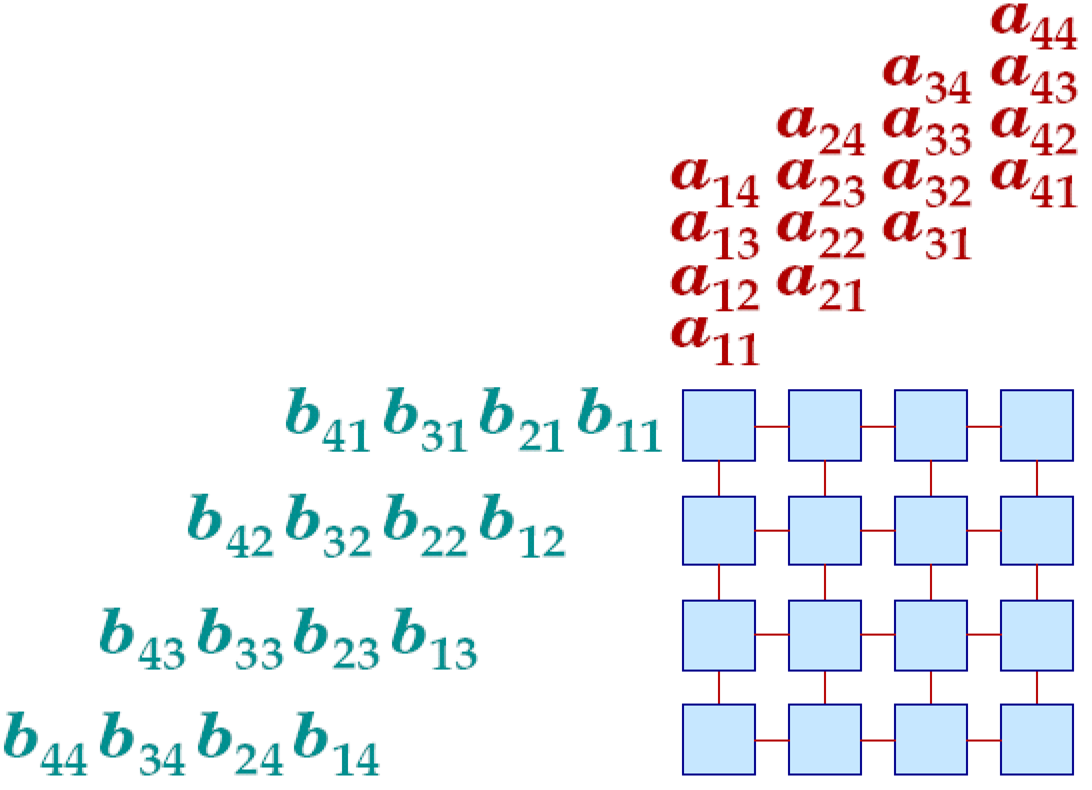
Number of steps = 2N – 1

Matrix-Matrix Multiplication



Number of time steps = ?

Matrix-Matrix Multiplication

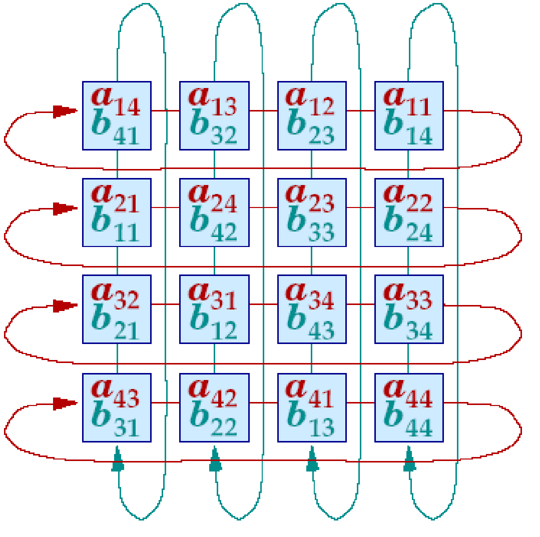


Number of time steps = 3N – 2

Complexity

- The algorithm implementations on the linear arrays have speedups that are linear in the number of processors – an efficiency of O(1)

- It is possible to improve these algorithms by a constant factor, for example, by inputting values directly to each processor in the first step and providing wraparound edges (N time steps)



Solving Systems of Equations

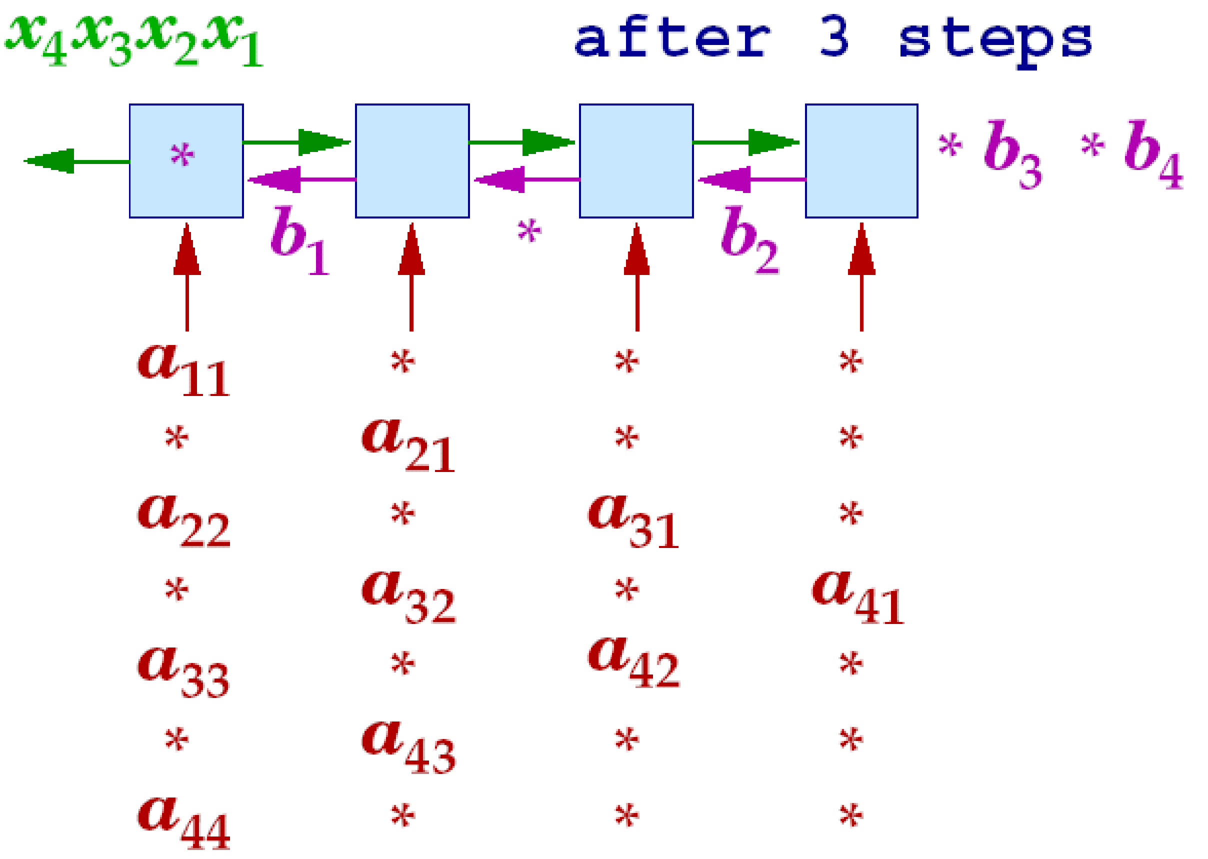
- Given an N x N lower triangular matrix A and an N-vector *b*, solve for *x*, where A*x* = *b* (assume solution exists)

$a_{11}x_1 = b_1$
$a_{21}x_1 + a_{22}x_2 = b_2$, and so on…

Define $t_1 =_{\text{def}} b_1$, $t_i =_{\text{def}} b_i - \sum_{j=1}^{i-1} a_{ij}x_j, 2 \leq i \leq N$. Then $x_i = t_i/a_{ii}$.

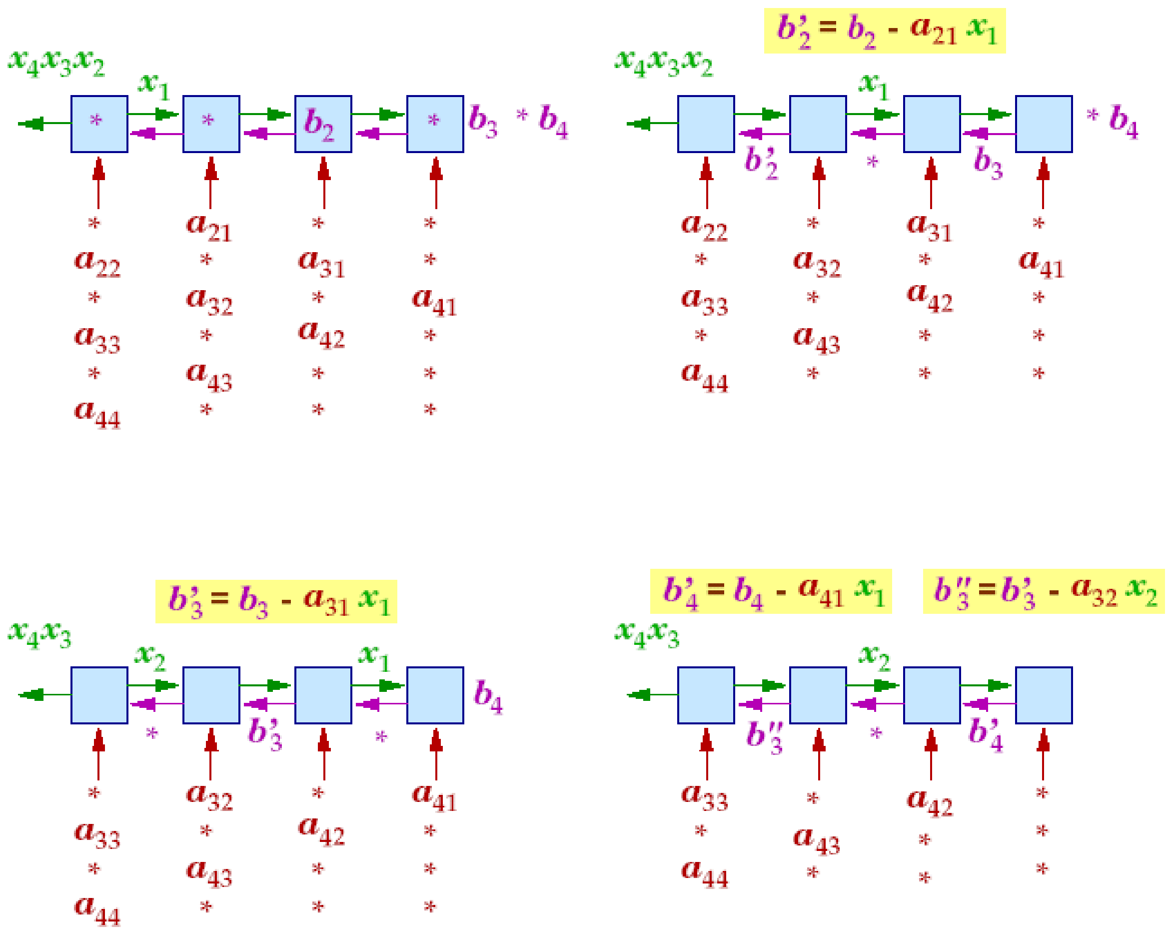
Equation Solver

Define $t_1 =_{\mathrm{def}} b_1$, $t_i =_{\mathrm{def}} b_i - \sum_{j=1}^{i-1} a_{ij}x_j, 2 \leq i \leq N$. Then $x_i = t_i / a_{ii}$.



Equation Solver Example

- When an *x, b,* and *a* meet at a cell, *ax* is subtracted from *b*
- When *b* and *a* meet at cell 1, *b* is divided by *a* to become *x*



Complexity

- Time steps = 2N – 1
- Speedup = O(N), efficiency = O(1)
- Note that half the processors are idle every time step – can improve efficiency by solving two interleaved equation systems simultaneously

Gaussian Elimination

- Solving for x, where Ax=b and A is a nonsingular matrix
- Note that $A^{-1}Ax = A^{-1}b = x$; keep applying transformations to A such that A becomes I ; the same transformations applied to b will result in the solution for x
- Sequential algorithm steps:
  - Pick a row where the first (i^{th}) element is non-zero and normalize the row so that the first (i^{th}) element is 1
  - Subtract a multiple of this row from all other rows so that their first (i^{th}) element is zero
  - Repeat for all i

Sequential Example

$$\begin{array}{ccc} 2 & 4 & -7 \\ 3 & 6 & -10 \\ -1 & 3 & -4 \end{array} \quad \begin{array}{c} x1 \\ x2 \\ x3 \end{array} \quad = \quad \begin{array}{c} 3 \\ 4 \\ 6 \end{array}$$

$$\begin{array}{ccc} 1 & 2 & -7/2 \\ 3 & 6 & -10 \\ -1 & 3 & -4 \end{array} \quad \begin{array}{c} x1 \\ x2 \\ x3 \end{array} \quad = \quad \begin{array}{c} 3/2 \\ 4 \\ 6 \end{array}$$

$$\begin{array}{ccc} 1 & 2 & -7/2 \\ 0 & 0 & 1/2 \\ -1 & 3 & -4 \end{array} \quad \begin{array}{c} x1 \\ x2 \\ x3 \end{array} \quad = \quad \begin{array}{c} 3/2 \\ -1/2 \\ 6 \end{array}$$

$$\begin{array}{ccc} 1 & 2 & -7/2 \\ 0 & 0 & 1/2 \\ 0 & 5 & -15/2 \end{array} \quad \begin{array}{c} x1 \\ x2 \\ x3 \end{array} \quad = \quad \begin{array}{c} 3/2 \\ -1/2 \\ 15/2 \end{array}$$

$$\begin{array}{ccc} 1 & 2 & -7/2 \\ 0 & 5 & -15/2 \\ 0 & 0 & 1/2 \end{array} \quad \begin{array}{c} x1 \\ x2 \\ x3 \end{array} \quad = \quad \begin{array}{c} 3/2 \\ 15/2 \\ -1/2 \end{array}$$

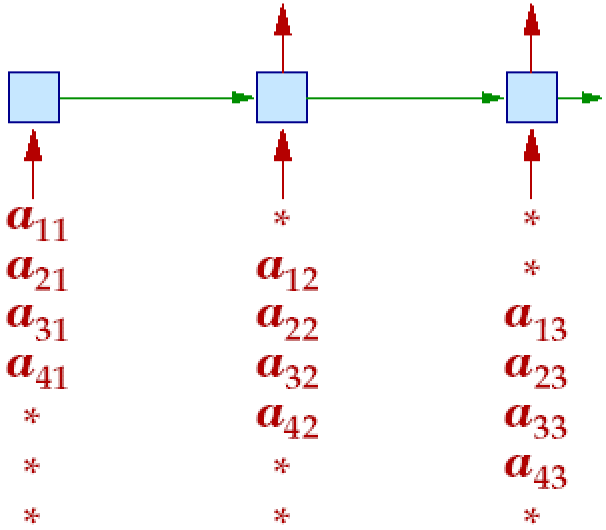
$$\begin{array}{ccc} 1 & 2 & -7/2 \\ 0 & 1 & -3/2 \\ 0 & 0 & 1/2 \end{array} \quad \begin{array}{c} x1 \\ x2 \\ x3 \end{array} \quad = \quad \begin{array}{c} 3/2 \\ 3/2 \\ -1/2 \end{array}$$

$$\begin{array}{ccc} 1 & 0 & -1/2 \\ 0 & 1 & -3/2 \\ 0 & 0 & 1/2 \end{array} \quad \begin{array}{c} x1 \\ x2 \\ x3 \end{array} \quad = \quad \begin{array}{c} -3/2 \\ 3/2 \\ -1/2 \end{array}$$

$$\begin{array}{ccc} 1 & 0 & -1/2 \\ 0 & 1 & -3/2 \\ 0 & 0 & 1 \end{array} \quad \begin{array}{c} x1 \\ x2 \\ x3 \end{array} \quad = \quad \begin{array}{c} -3/2 \\ 3/2 \\ -1 \end{array}$$

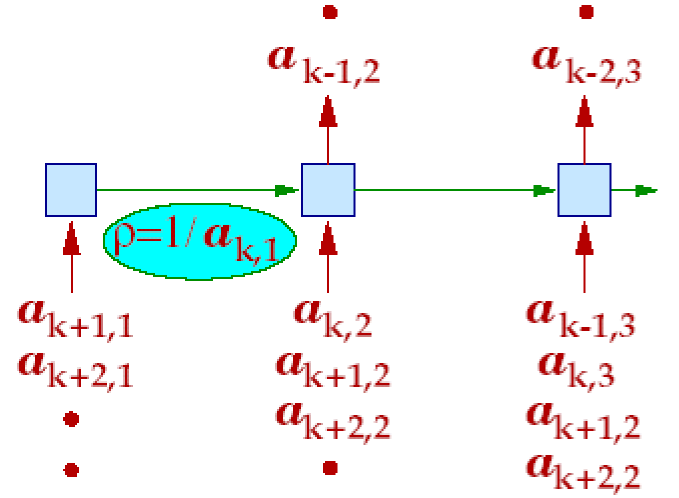
$$\begin{array}{ccc} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{array} \quad \begin{array}{c} x1 \\ x2 \\ x3 \end{array} \quad = \quad \begin{array}{c} -2 \\ 0 \\ -1 \end{array}$$

Algorithm Implementation

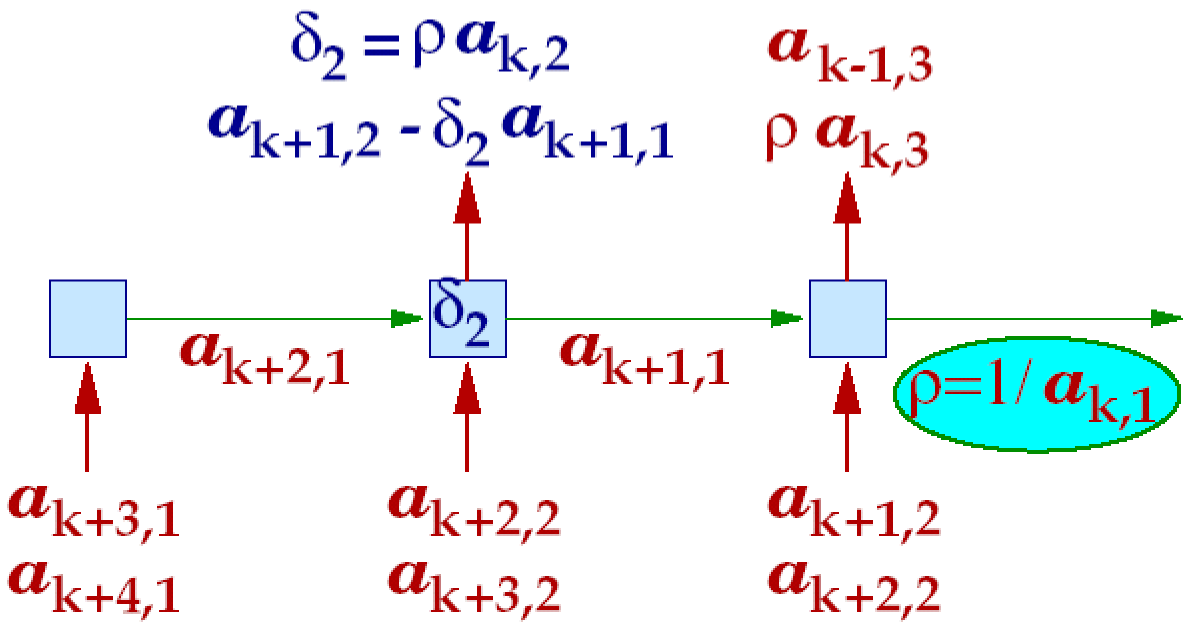


- The matrix is input in staggered form
- The first cell discards inputs until it finds a non-zero element (the pivot row)

- The inverse ρ of the non-zero element is now sent rightward
- ρ arrives at each cell at the same time as the corresponding element of the pivot row



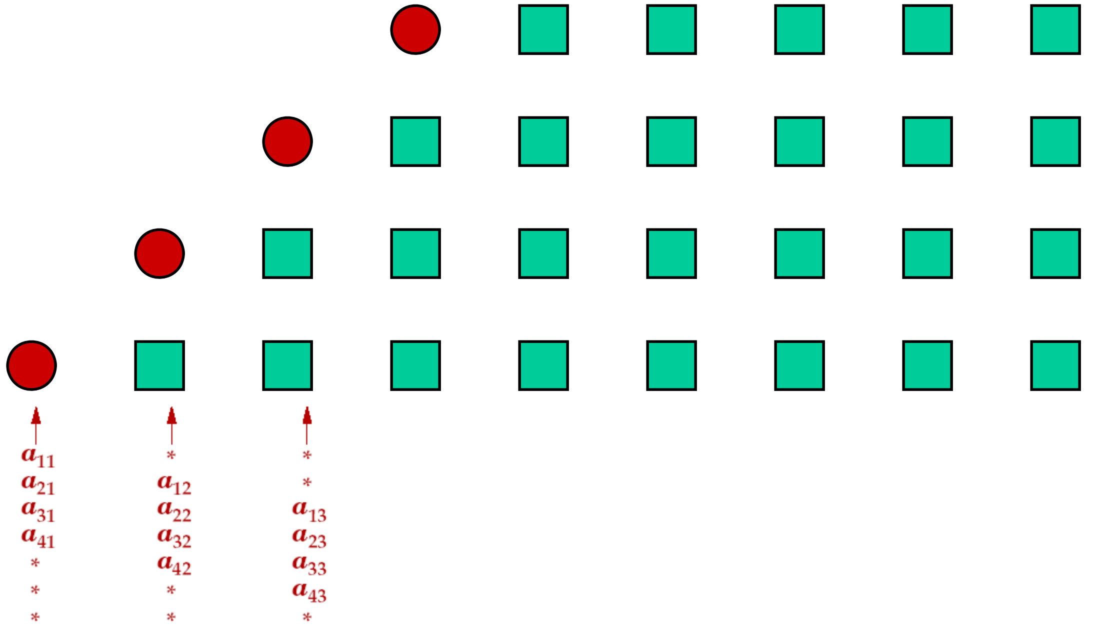
Algorithm Implementation



- Each cell stores $\delta_i = \rho\ a_{k,l}$ – the value for the normalized pivot row
- This value is used when subtracting a multiple of the pivot row from other rows
- What is the multiple? It is $a_{j,1}$
- How does each cell receive $a_{j,1}$? It is passed rightward by the first cell
- Each cell now outputs the new values for each row
- The first cell only outputs zeroes and these outputs are no longer needed

Algorithm Implementation

- The outputs of all but the first cell must now go through the remaining algorithm steps
- A triangular matrix of processors efficiently implements the flow of data
- Number of time steps?
- Can be extended to compute the inverse of a matrix



Graph Algorithms

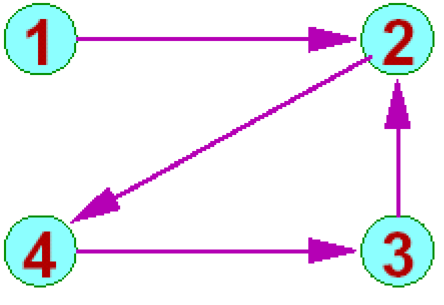
$G = (V, E)$: a directed graph, $V = \{1, \ldots, N\}$

The *adjacency matrix* $A = (a_{ij})$ of G is

$$a_{ij} = \begin{cases} 1 & \text{if either } (i,j) \in E \text{ or } i = j, \\ 0 & \text{otherwise.} \end{cases}$$

The transitive closure of G is $G^* = (V, E^*)$,

$$E^* = \{(i,j) \mid j \text{ is reachable from } i \text{ in } G\}.$$



$$A = \begin{pmatrix} 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 \end{pmatrix} \qquad A^* = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 \end{pmatrix}$$

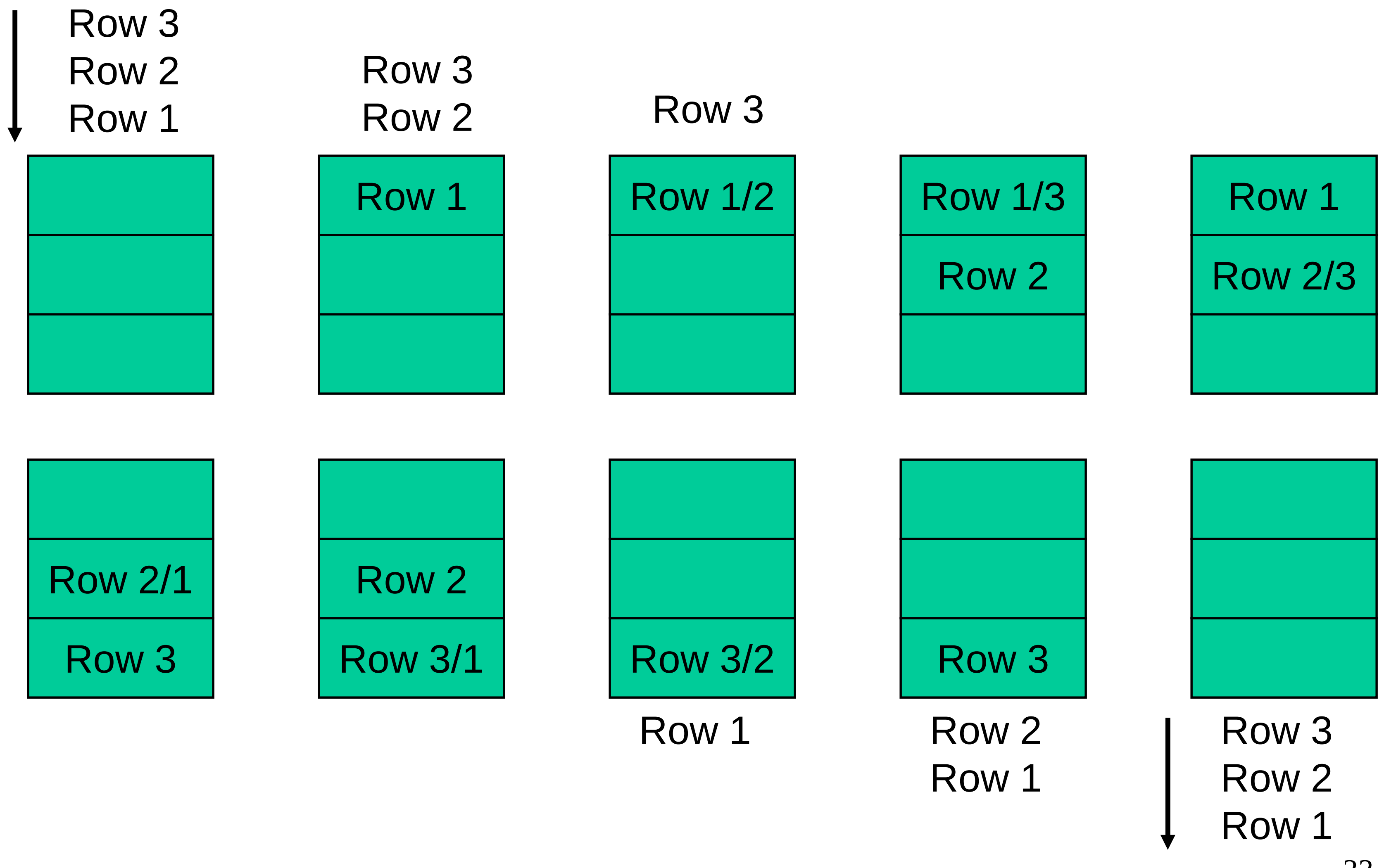
Floyd Warshall Algorithm

$A^{(k)} =_{\text{def}} (a_{ij}^{(k)})$, where for each $k, 0 \leq k \leq N$, $a_{ij}^{(k)} = 1$ if j is reachable from i *passing through* only nodes $\leq k$ and 0 otherwise.

Then $A^{(N)} = A^*$, $A^{(0)} = A$, and for all $k \geq 1$,

$$a_{ij}^{(k)} = a_{ij}^{(k-1)} \vee \left(a_{ik}^{(k-1)} \wedge a_{kj}^{(k-1)} \right).$$

Implementation on 2d Processor Array



Algorithm Implementation

- Diagonal elements of the processor array can broadcast to the entire row in one time step (if this assumption is not made, inputs will have to be staggered)
- A row sifts down until it finds an empty row – it sifts down again after all other rows have passed over it
- When a row passes over the 1st row, the value of a_{i1} is broadcast to the entire row – a_{ij} is set to 1 if $a_{i1} = a_{1j} = 1$ – in other words, the row is now the i^{th} row of $A^{(1)}$
- By the time the k^{th} row finds its empty slot, it has already become the k^{th} row of $A^{(k-1)}$

Algorithm Implementation

- When the i^{th} row starts moving again, it travels over rows a_k ($k > i$) and gets updated depending on whether there is a path from i to j via vertices < k (and including k)

Shortest Paths

- Given a graph and edges with weights, compute the weight of the shortest path between pairs of vertices
- Can the transitive closure algorithm be applied here?

Shortest Paths Algorithm

$D = (d_{ij})$: the distance matrix, where $d_{ij} = \infty$ if there is no edge from i to j

Define $D^{(k)} = (d_{ij}^{(k)})$, where $d_{ij}^{(k)}$ is the length of the shortest path from i to j that passes through only nodes $\leq k$.

Then we have only to compute $D^{(N)}$. Note $D^{(0)} = D$ and for all $k \geq 1$,

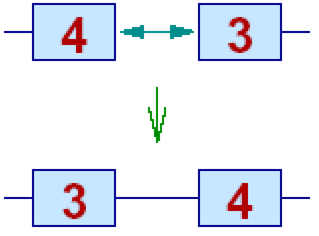
$$d_{ij}^{(k)} = \min(d_{ij}^{(k-1)}, d_{ik}^{(k-1)} + d_{kj}^{(k-1)}).$$

The above equation is very similar to that in transitive closure

Sorting with Comparison Exchange

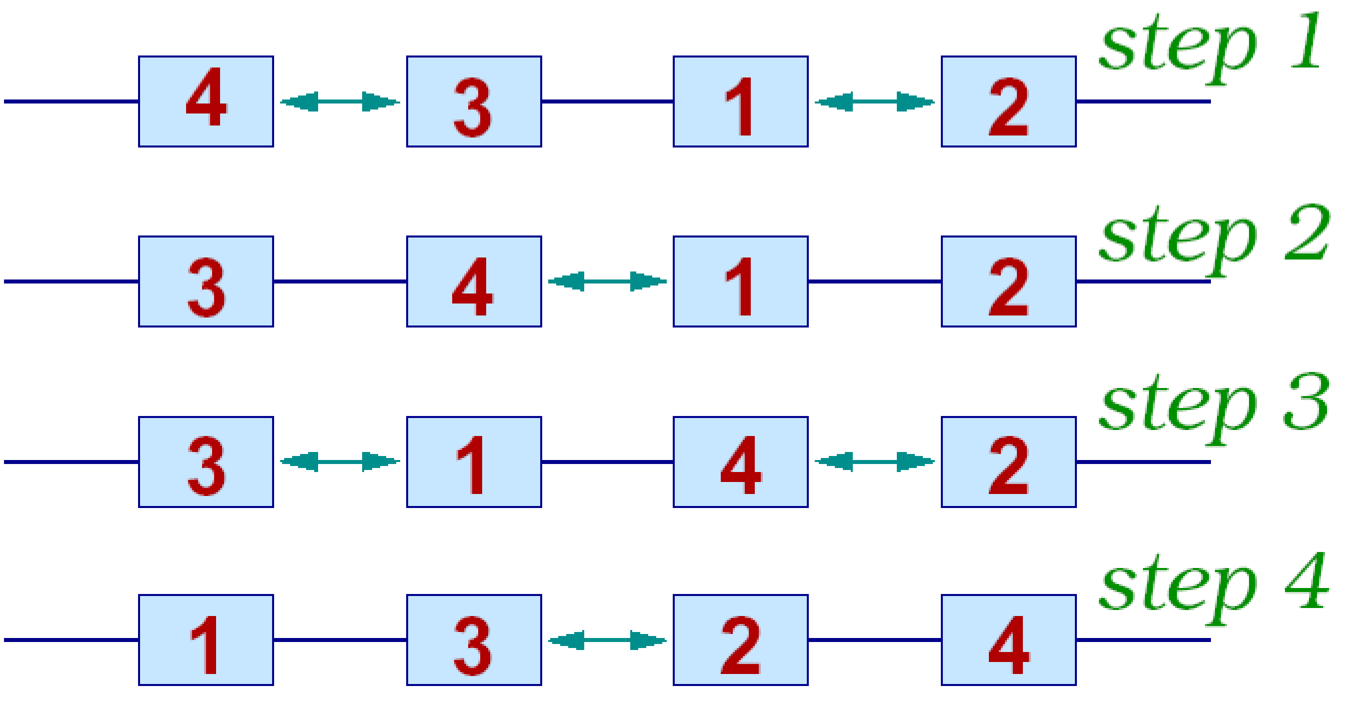
- Earlier sort implementations assumed processors that could compare inputs and local storage, and generate an output in a single time step

- The next algorithm assumes comparison-exchange processors: two neighboring processors I and J (I < J) show their numbers to each other and I keeps the smaller number and J the larger



Odd-Even Sort

- N numbers can be sorted on an N-cell linear array in O(N) time: the processors alternate operations with their neighbors



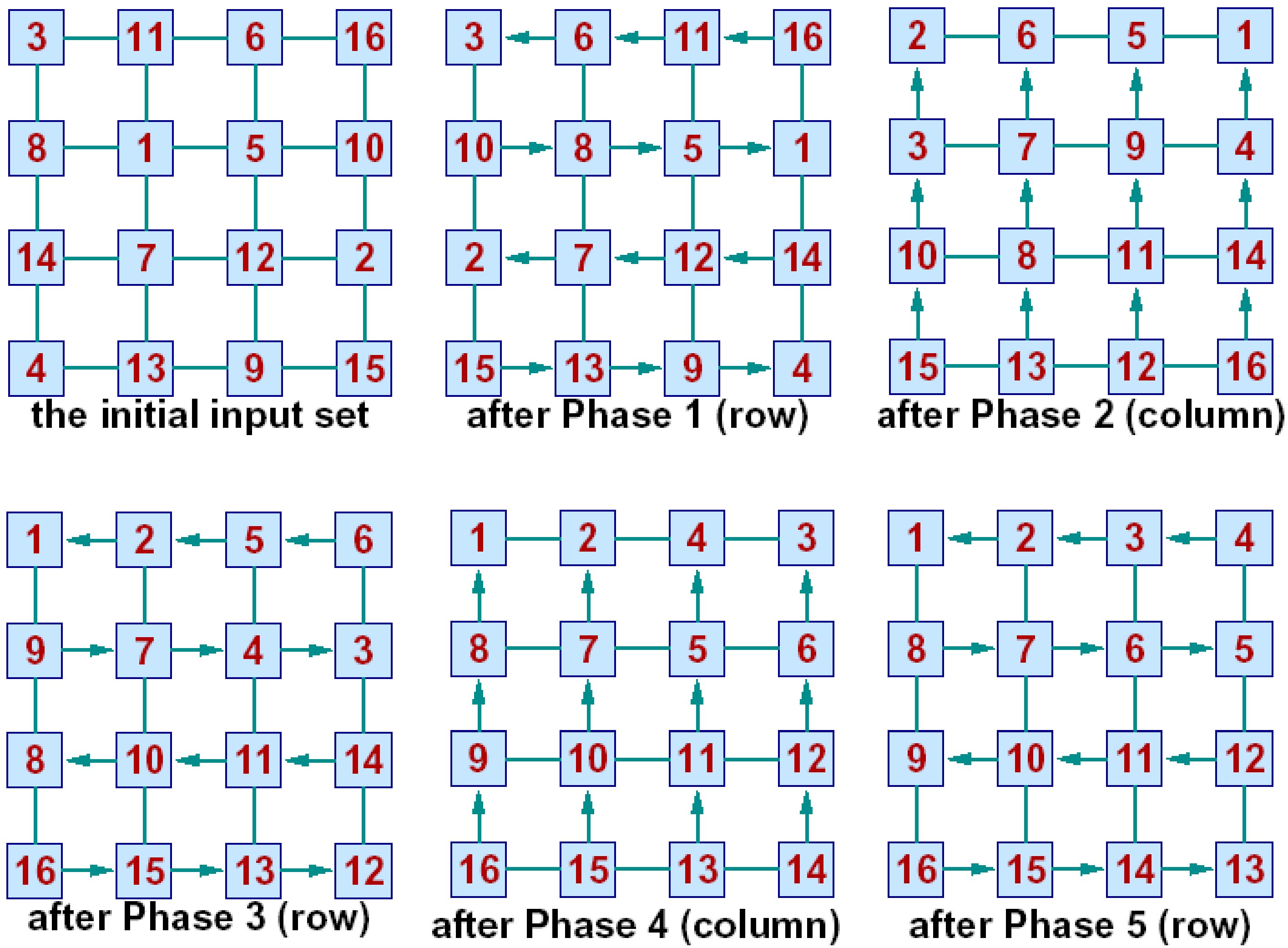
Shearsort

- A sorting algorithm on an N-cell square matrix that improves execution time to O(sqrt(N) logN)

- Algorithm steps:
  Odd phase: sort each row with odd-even sort (all odd rows are sorted left to right and all even rows are sorted right to left)
  Even phase: sort each column with odd-even sort
  Repeat

- Each odd and even phase takes O(sqrt(N)) steps – the input is guaranteed to be sorted in O(logN) steps

Example


3 11 6 16
8 1 5 10
14 7 12 2
4 13 9 15
the initial input set
3 6 11 16
10 8 5 1
2 7 12 14
15 13 9 4
after Phase 1 (row)
2 6 5 1
3 7 9 4
10 8 11 14
15 13 12 16
after Phase 2 (column)
1 2 5 6
9 7 4 3
8 10 11 14
16 15 13 12
after Phase 3 (row)
1 2 4 3
8 7 5 6
9 10 11 12
16 15 13 14
after Phase 4 (column)
1 2 3 4
8 7 6 5
9 10 11 12
16 15 14 13
after Phase 5 (row)

The 0-1 Sorting Lemma

If a comparison-exchange algorithm sorts input sets consisting solely of 0's and 1's, then it sorts all input sets of arbitrary values

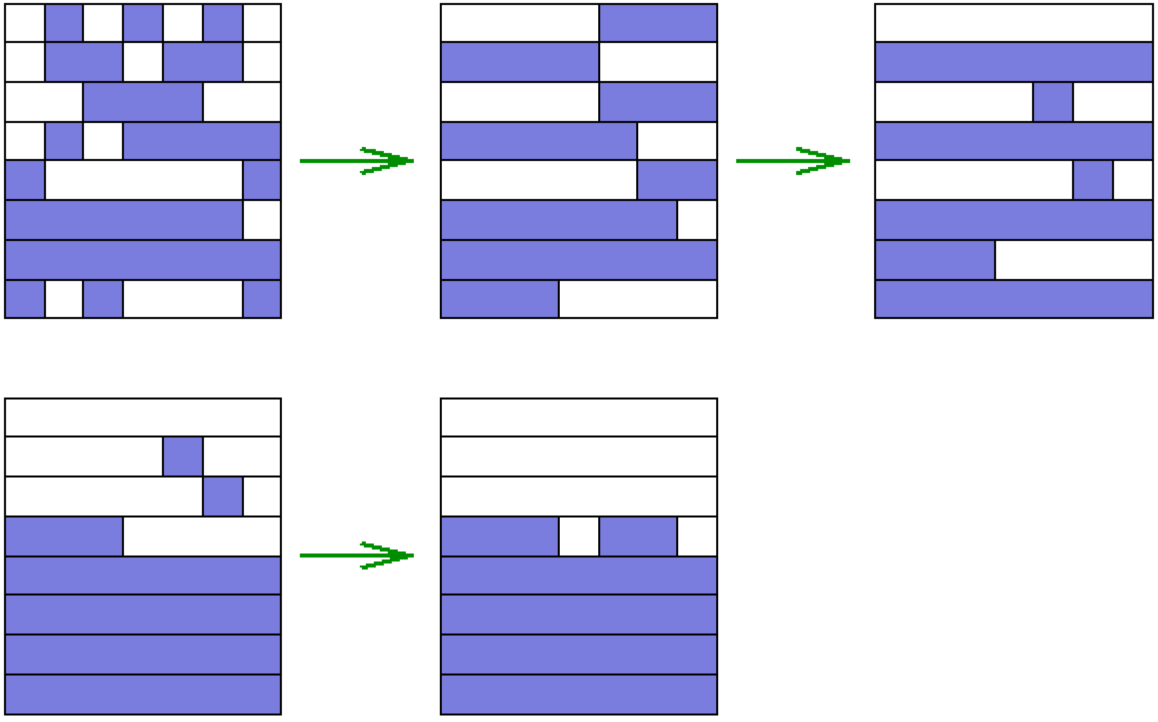
Proof Let an o.c.e. algorithm $\mathcal{A}$ for input size N be given. Suppose that $\mathcal{A}$ works on all 0-1 inputs. Assume that $\mathcal{A}$ fails on an input $[x_1, \ldots, x_N]$ with $[y_1, \ldots, y_N]$ be the output. Then $y_1 \leq \cdots \leq y_m > y_{m+1}$ for some m. Define mapping F by $F(x) = 0$ if $x < y_m$ and 1 otherwise. Since F preserves the order $\leq$, the output of $\mathcal{A}$ on $[F(x_1), \ldots, F(x_N)]$ is $[F(y_1), \ldots, F(y_N)]$, and is of the form $[\ldots, 1, 0, \ldots]$ because of $y_m > y_{m+1}$. This is a contradiction.

Complexity Proof

- How do we prove that the algorithm completes in O(logN) phases? (each phase takes O(sqrt(N)) steps)
- Assume input set of 0s and 1s
- There are three types of rows: all 0s, all 1s, and mixed entries – we will show that after every phase, the number of mixed entry rows reduces by half
- The column sort phase is broken into the smaller steps below: move 0 rows to the top and 1 rows to the bottom; the mixed rows are paired up and sorted within pairs; repeat these small steps until the column is sorted

Example

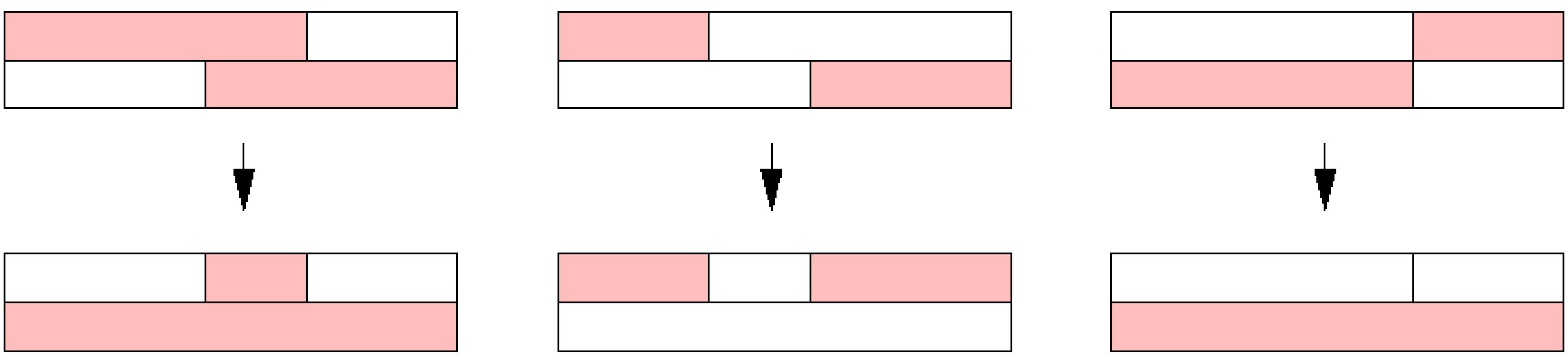
- The modified algorithm will behave as shown below: white depicts 0s and blue depicts 1s



Proof

- If there are N mixed rows, we are guaranteed to have fewer than N/2 mixed rows after the first step of the column sort (subsequent steps of the column sort may not produce fewer mixed rows as the rows are not sorted)

Each pair of mixed rows produces at least one pure row when sorted



Title

- Bullet