Parallel Programming
in C with MPI and OpenMP

Michael J. Quinn

McGraw Hill

Chapter 12

Solving Linear Systems

Outline

- Terminology
- Back substitution
- Gaussian elimination
- Jacobi method
- Conjugate gradient method

Terminology

- System of linear equations
  - Solve $Ax = b$ for x
- Special matrices
  - Upper triangular
  - Lower triangular
  - Diagonally dominant
  - Symmetric

Upper Triangular

4	2	-1	5	9	2
0	-4	5	6	0	-4
0	0	3	2	4	6
0	0	0	0	9	2
0	0	0	0	8	7
0	0	0	0	0	2

Lower Triangular

4	0	0	0	0	0
0	0	0	0	0	0
5	4	3	0	0	0
2	6	2	3	0	0
8	-2	0	1	8	0
-3	5	7	9	5	2

Diagonally Dominant

19	0	2	2	0	6
0	-15	2	0	-3	0
5	4	22	-1	0	4
2	3	2	13	0	-5
5	-2	0	1	16	0
-3	5	5	3	5	-32

Symmetric

3	0	5	2	0	6
0	7	4	3	-3	5
5	4	0	-1	0	4
2	3	-1	9	0	-5
0	-3	0	0	5	5
6	5	4	-5	5	-3

Back Substitution

- Used to solve upper triangular system $Tx = b$ for x
- Methodology: one element of x can be immediately computed
- Use this value to simplify system, revealing another element that can be immediately computed
- Repeat

Back Substitution

$$\begin{array}{rrrrcr}
1x_0 & +1x_1 & -1x_2 & +4x_3 & = & 8 \\
 & -2x_1 & -3x_2 & +1x_3 & = & 5 \\
 & & 2x_2 & -3x_3 & = & 0 \\
 & & & 2x_3 & = & 4
\end{array}$$

Back Substitution

$1x_0$	$+1x_1$	$-1x_2$	$+4x_3$	$=$	8
	$-2x_1$	$-3x_2$	$+1x_3$	$=$	5
		$2x_2$	$-3x_3$	$=$	0
			$2x_3$	$=$	4

$$x_3 = 2$$

Back Substitution

$$
\begin{array}{ccccc}
1x_0 & +1x_1 & -1x_2 & & = & 0 \\
 & -2x_1 & -3x_2 & & = & 3 \\
 & & 2x_2 & & = & 6 \\
 & & & 2x_3 & = & 4
\end{array}
$$

Back Substitution

$1x_0$	$+1x_1$	$-1x_2$		$=$	0
	$-2x_1$	$-3x_2$		$=$	3
		$2x_2$		$=$	6
			$2x_3$	$=$	4

$$x_2 = 3$$

Back Substitution

$$
\begin{array}{llllll}
1x_0 & +1x_1 & & & = & 3 \\
 & -2x_1 & & & = & 12 \\
 & & 2x_2 & & = & 6 \\
 & & & 2x_3 & = & 4
\end{array}
$$

Back Substitution

$$
\begin{array}{llllll}
1x_0 & +1x_1 & & & = & 3 \\
 & -2x_1 & & & = & 12 \\
 & & 2x_2 & & = & 6 \\
 & & & 2x_3 & = & 4
\end{array}
$$

$$x_1 = -6$$

Back Substitution

$$
\begin{array}{rrrrcr}
1x_0 & & & & = & 9 \\
 & -2x_1 & & & = & 12 \\
 & & 2x_2 & & = & 6 \\
 & & & 2x_3 & = & 4
\end{array}
$$

Back Substitution

$$
\begin{array}{lllll}
1x_0 & & & = & 9 \\
 & -2x_1 & & = & 12 \\
 & & 2x_2 & = & 6 \\
 & & \quad 2x_3 & = & 4
\end{array}
$$

$$x_0 = 9$$

Pseudocode

for $i \leftarrow n - 1$ down to 1 do

  $x[i] \leftarrow b[i] / a[i, i]$

  for $j \leftarrow 0$ to $i - 1$ do

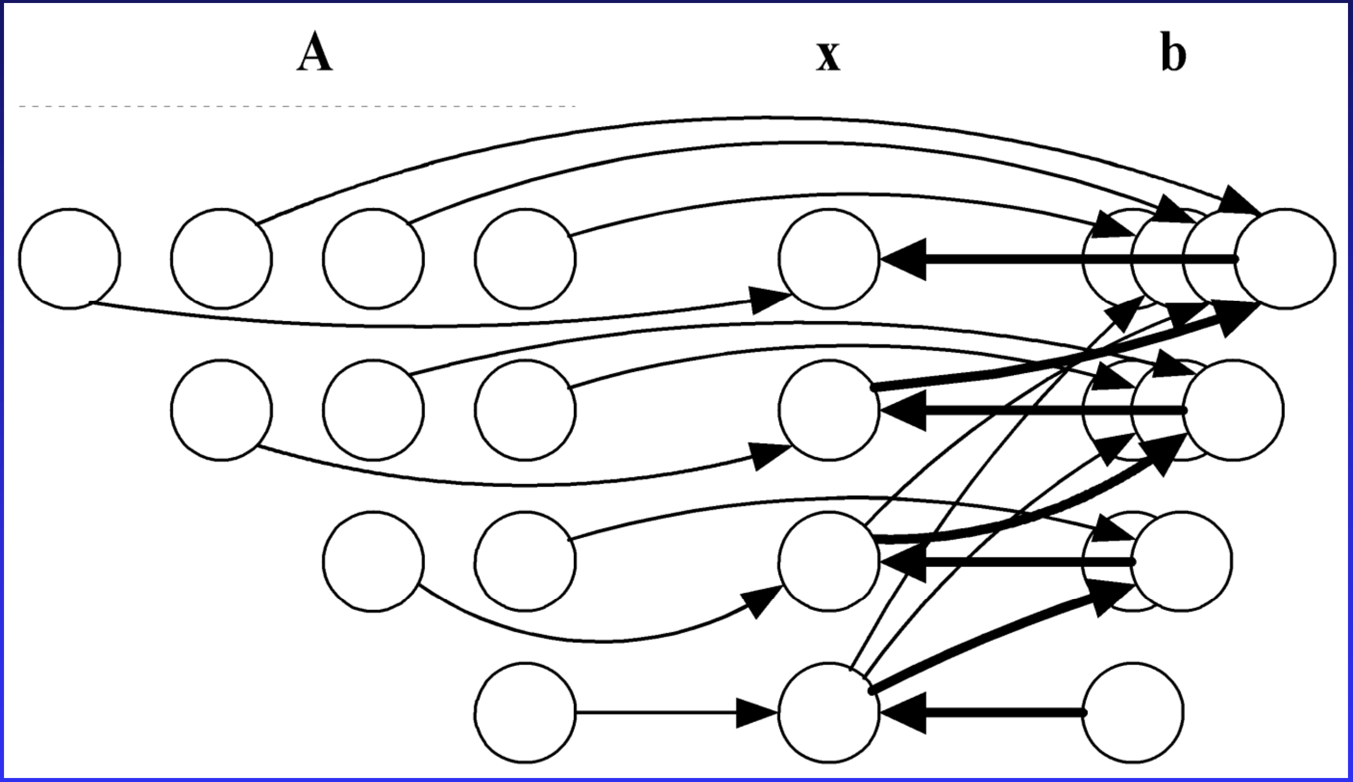
    $b[j] \leftarrow b[j] - x[i] \times a[j, i]$

  endfor

endfor

Time complexity: $\Theta(n^2)$

Data Dependence Diagram

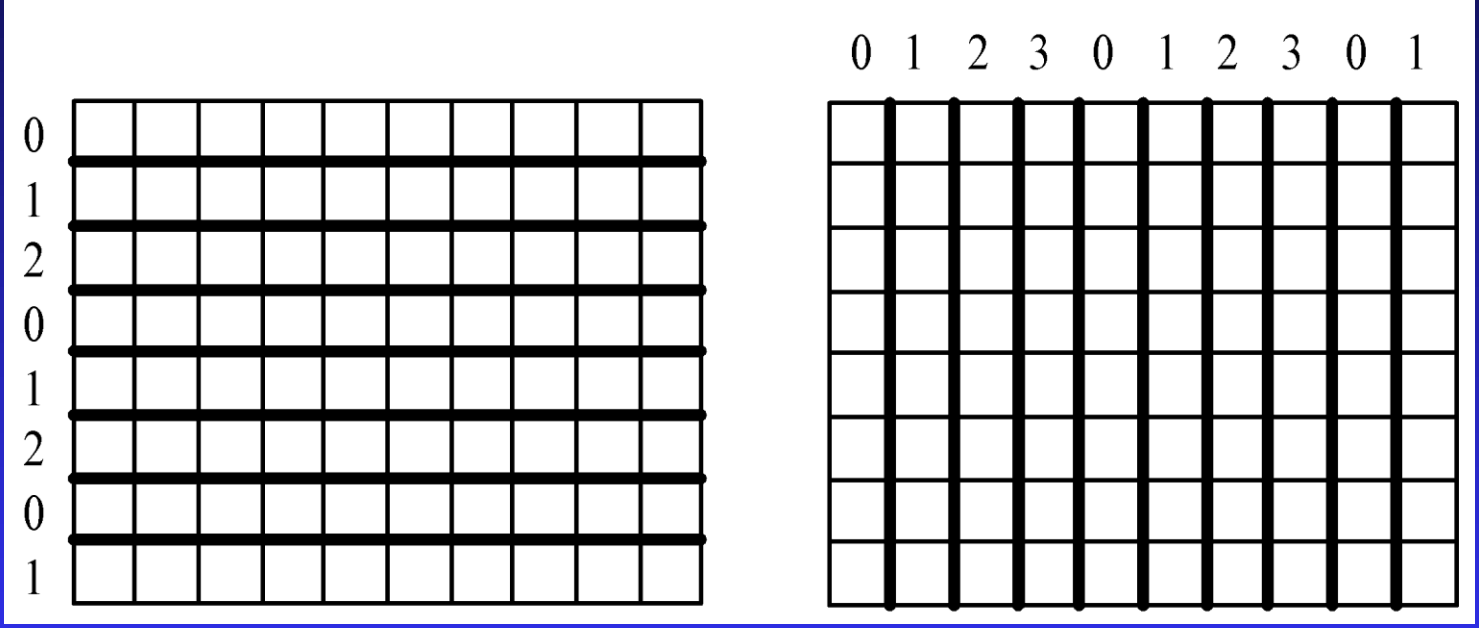


We cannot execute the outer loop in parallel.
We can execute the inner loop in parallel.

Row-oriented Algorithm

- Associate primitive task with each row of A and corresponding elements of x and b
- During iteration i task associated with row j computes new value of b_j
- Task i must compute x_i and broadcast its value
- Agglomerate using rowwise interleaved striped decomposition

Interleaved Decompositions



Rowwise interleaved striped decomposition

Columnwise interleaved striped decomposition

Complexity Analysis

- Each process performs about n / (2p) iterations of loop j in all
- A total of n -1 iterations in all
- Computational complexity: $\Theta(n^2/p)$
- One broadcast per iteration
- Communication complexity: $\Theta(n \log p)$

Gaussian Elimination

- Used to solve $Ax = b$ when A is dense
- Reduces $Ax = b$ to upper triangular system $Tx = c$
- Back substitution can then solve $Tx = c$ for x

Gaussian Elimination

$4x_0$	$+6x_1$	$+2x_2$	$-2x_3$	$=$	8
$2x_0$		$+5x_2$	$-2x_3$	$=$	4
$-4x_0$	$-3x_1$	$-5x_2$	$+4x_3$	$=$	1
$8x_0$	$+18x_1$	$-2x_2$	$+3x_3$	$=$	40

Gaussian Elimination

$4x_0$	$+6x_1$	$+2x_2$	$-2x_3$	$=$	8
	$-3x_1$	$+4x_2$	$-1x_3$	$=$	0
	$+3x_1$	$-3x_2$	$+2x_3$	$=$	9
	$+6x_1$	$-6x_2$	$+7x_3$	$=$	24

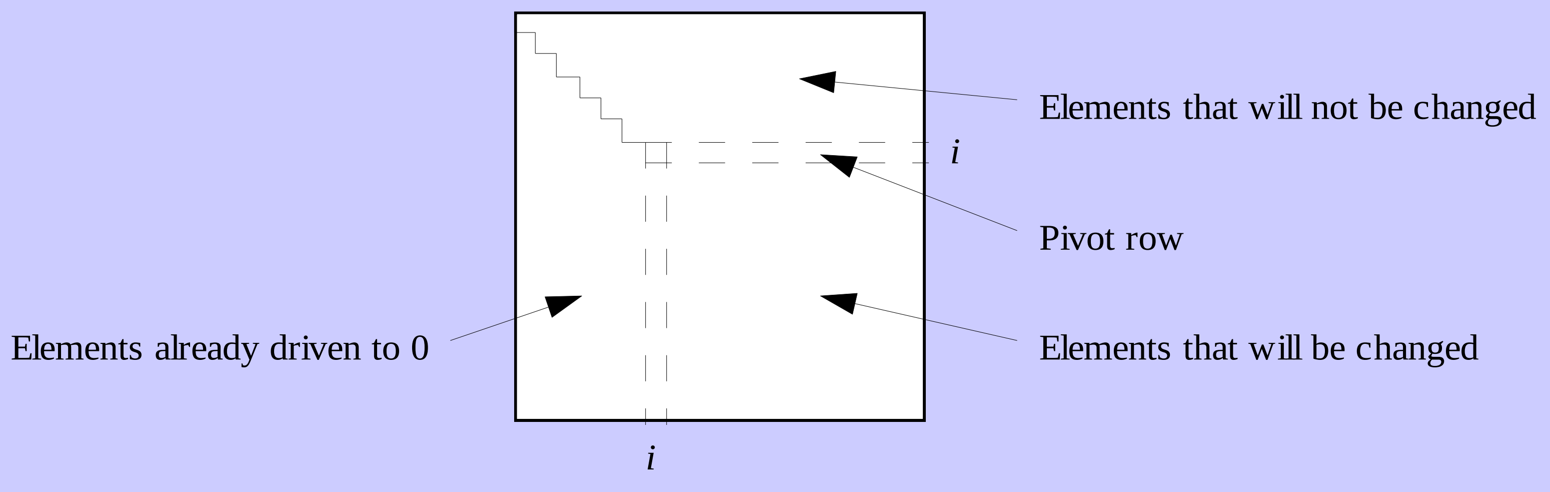
Gaussian Elimination

$$
\begin{array}{cccccc}
4x_0 & +6x_1 & +2x_2 & -2x_3 & = & 8 \\
 & -3x_1 & +4x_2 & -1x_3 & = & 0 \\
 & & 1x_2 & +1x_3 & = & 9 \\
 & & 2x_2 & +5x_3 & = & 24
\end{array}
$$

Gaussian Elimination

$$
\begin{array}{lllll}
4x_0 & +6x_1 & +2x_2 & -2x_3 & = 8 \\
 & -3x_1 & +4x_2 & -1x_3 & = 0 \\
 & & 1x_2 & +1x_3 & = 9 \\
 & & & 3x_3 & = 6
\end{array}
$$

Iteration of Gaussian Elimination



Numerical Stability Issues

- If pivot element close to zero, significant roundoff errors can result
- Gaussian elimination with partial pivoting eliminates this problem
- In step *i* we search rows *i* through *n*-1 for the row whose column *i* element has the largest absolute value
- Swap (pivot) this row with row *i*

Row-oriented Parallel Algorithm

- Associate primitive task with each row of *A* and corresponding elements of *x* and *b*
- A kind of reduction needed to find the identity of the pivot row
- Tournament: want to determine identity of row with largest value, rather than largest value itself
- Could be done with two all-reductions
- MPI provides a simpler, faster mechanism

MPI_MAXLOC, MPI_MINLOC

- MPI provides reduction operators MPI_MAXLOC, MPI_MINLOC
- Provide datatype representing a (value, index) pair

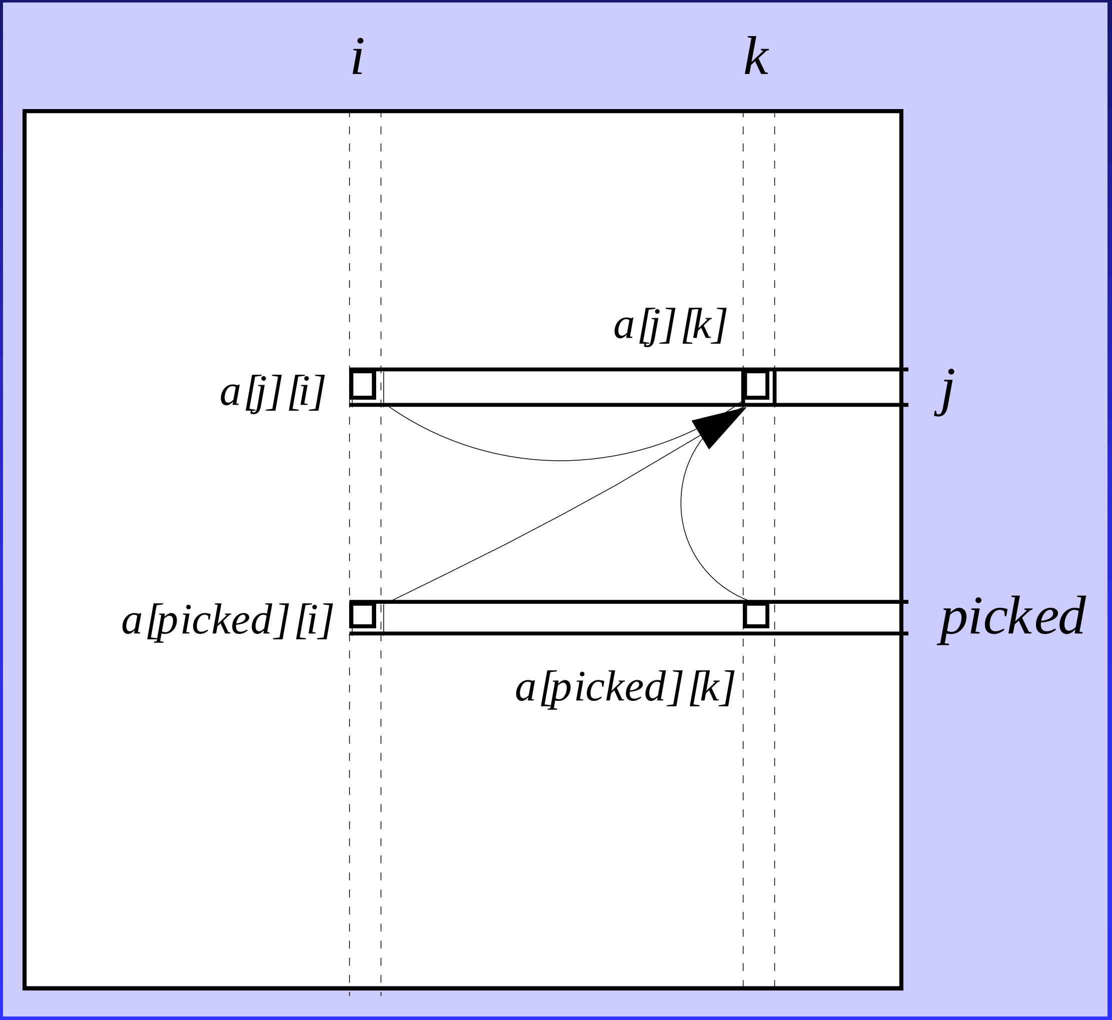
MPI (value,index) Datatypes

MPI_Datatype	*Meaning*
MPI_2INT	Two ints
MPI_DOUBLE_INT	A double followed by an int
MPI_FLOAT_INT	A float followed by an int
MPI_LONG_INT	A long followed by an int
MPI_LONG_DOUBLE_INT	A long double followed by an int
MPI_SHORT_INT	A short followed by an int

Example Use of MPI_MAXLOC

```
struct {
    double value;
    int    index;
} local, global;
...
local.value = fabs(a[j][i]);
local.index = j;
...
MPI_Allreduce (&local, &global, 1,
    MPI_DOUBLE_INT, MPI_MAXLOC,
    MPI_COMM_WORLD);
```

Second Communication per Iteration



Communication Complexity

- Complexity of tournament: $\Theta(\log p)$
- Complexity of broadcasting pivot row: $\Theta(n \log p)$
- A total of $n - 1$ iterations
- Overall communication complexity: $\Theta(n^2 \log p)$

Column-oriented Algorithm

- Associate a primitive task with each column of *A* and another primitive task for *b*
- During iteration *i* task controlling column *i* determines pivot row and broadcasts its identity
- During iteration *i* task controlling column *i* must also broadcast column *i* to other tasks
- Agglomerate tasks in an interleaved fashion to balance workloads
- Isoefficiency same as row-oriented algorithm

Comparison of Two Algorithms

- Both algorithms evenly divide workload
- Both algorithms do a broadcast each iteration
- Difference: identification of pivot row
  - Row-oriented algorithm does search in parallel but requires all-reduce step
  - Column-oriented algorithm does search sequentially but requires no communication
- Row-oriented superior when *n* relatively larger and *p* relatively smaller

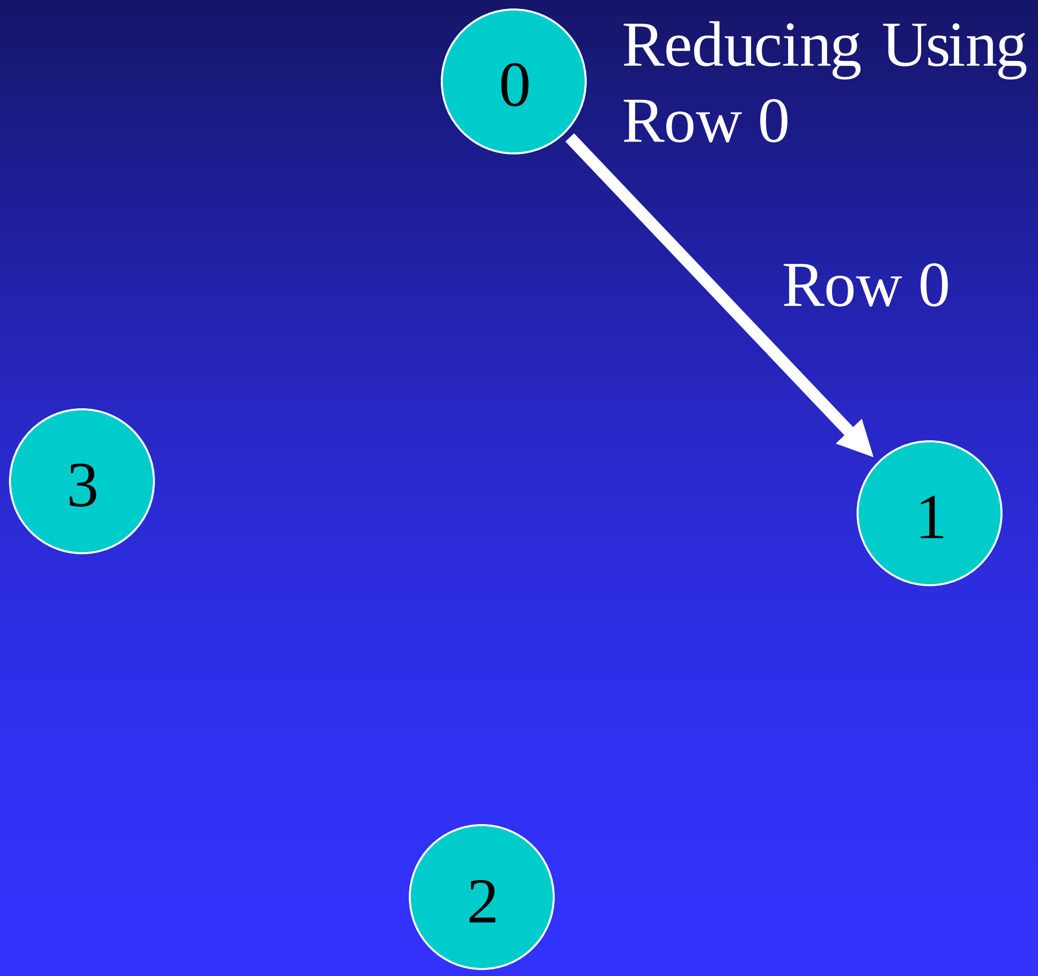
Problems with These Algorithms

- They break parallel execution into computation and communication phases
- Processes not performing computations during the broadcast steps
- Time spent doing broadcasts is large enough to ensure poor scalability

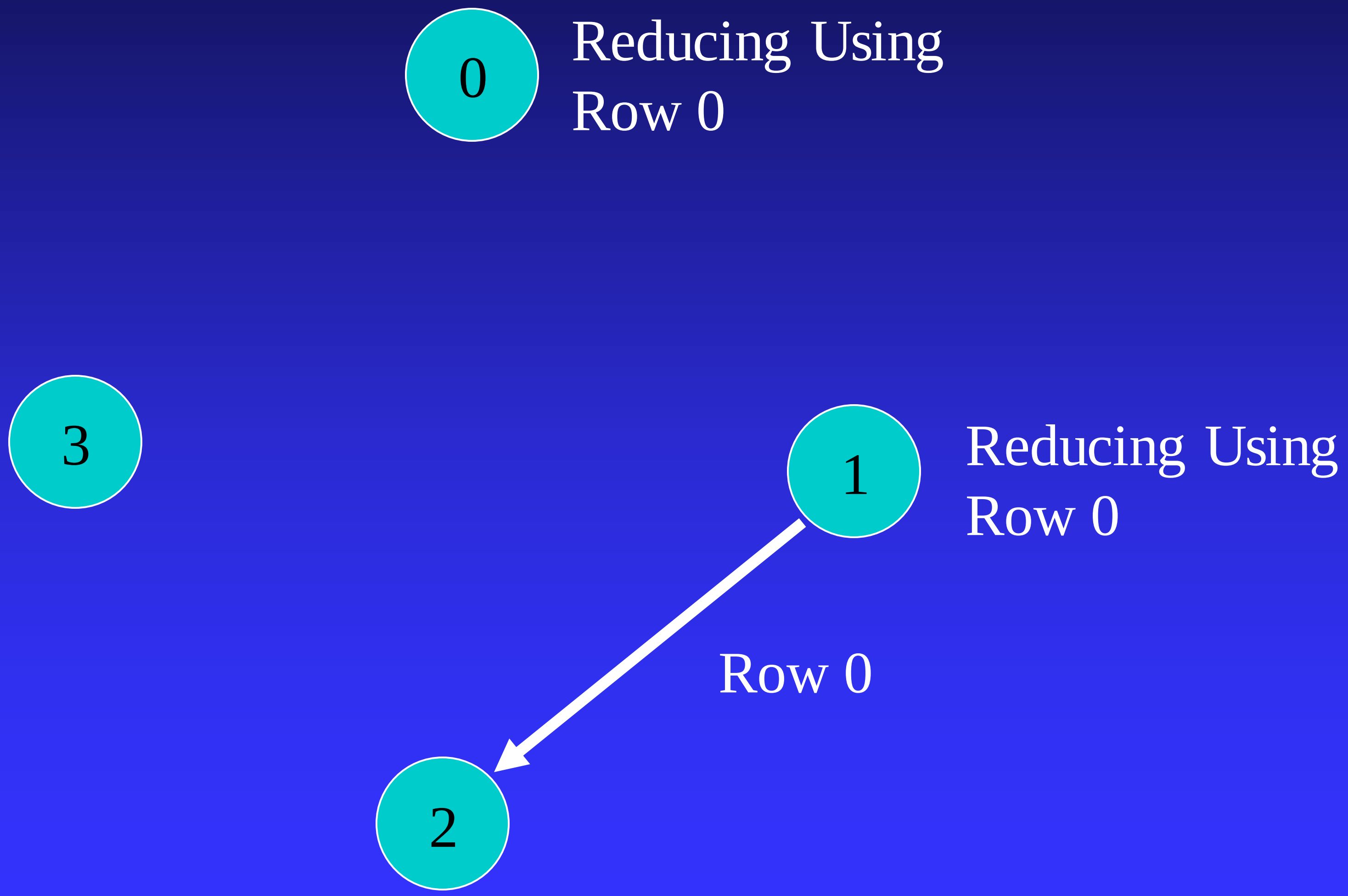
Pipelined, Row-Oriented Algorithm

- Want to overlap communication time with computation time
- We could do this if we knew in advance the row used to reduce all the other rows.
- Let’s pivot columns instead of rows!
- In iteration *i* we can use row *i* to reduce the other rows.

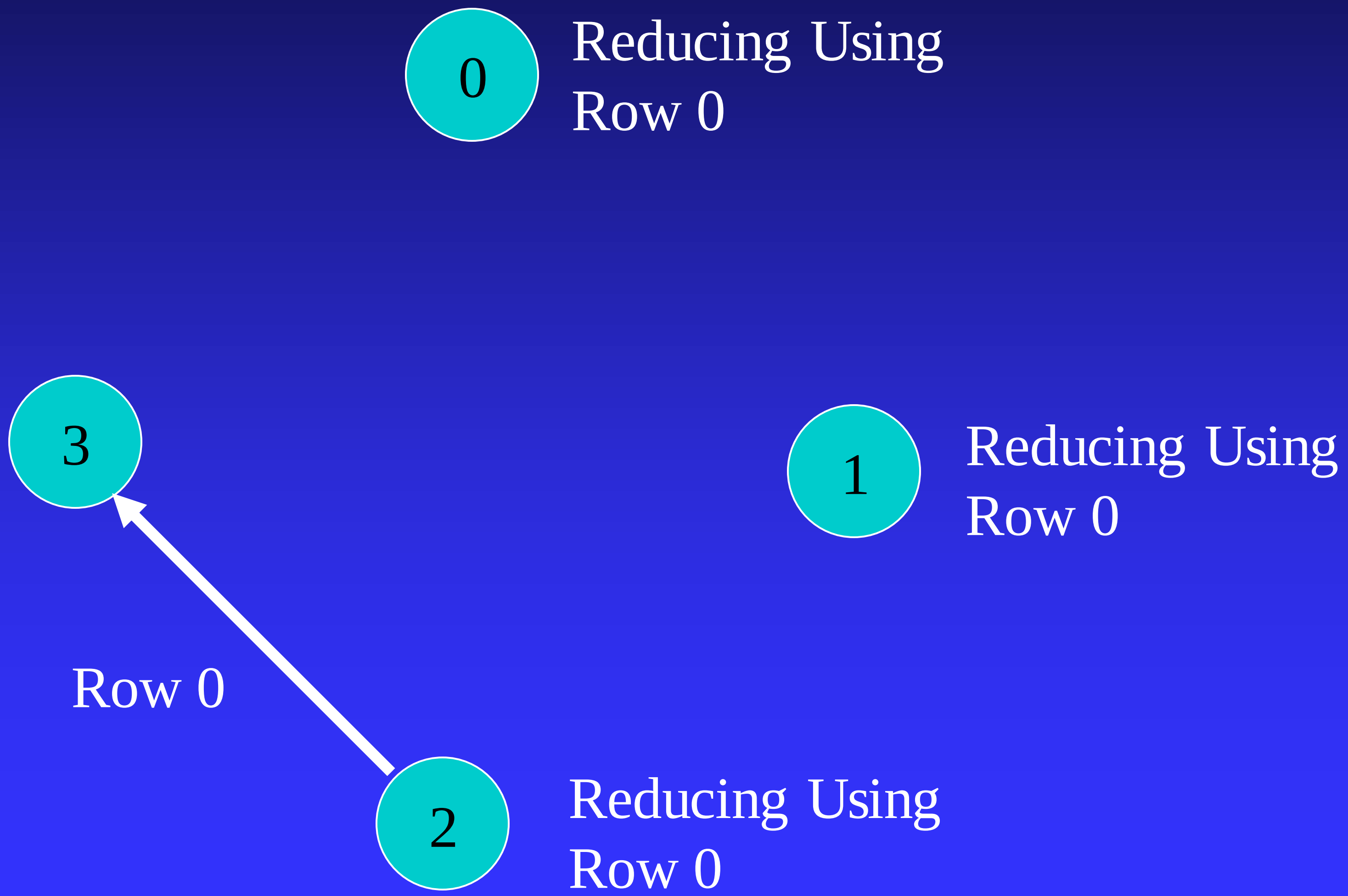
Communication Pattern



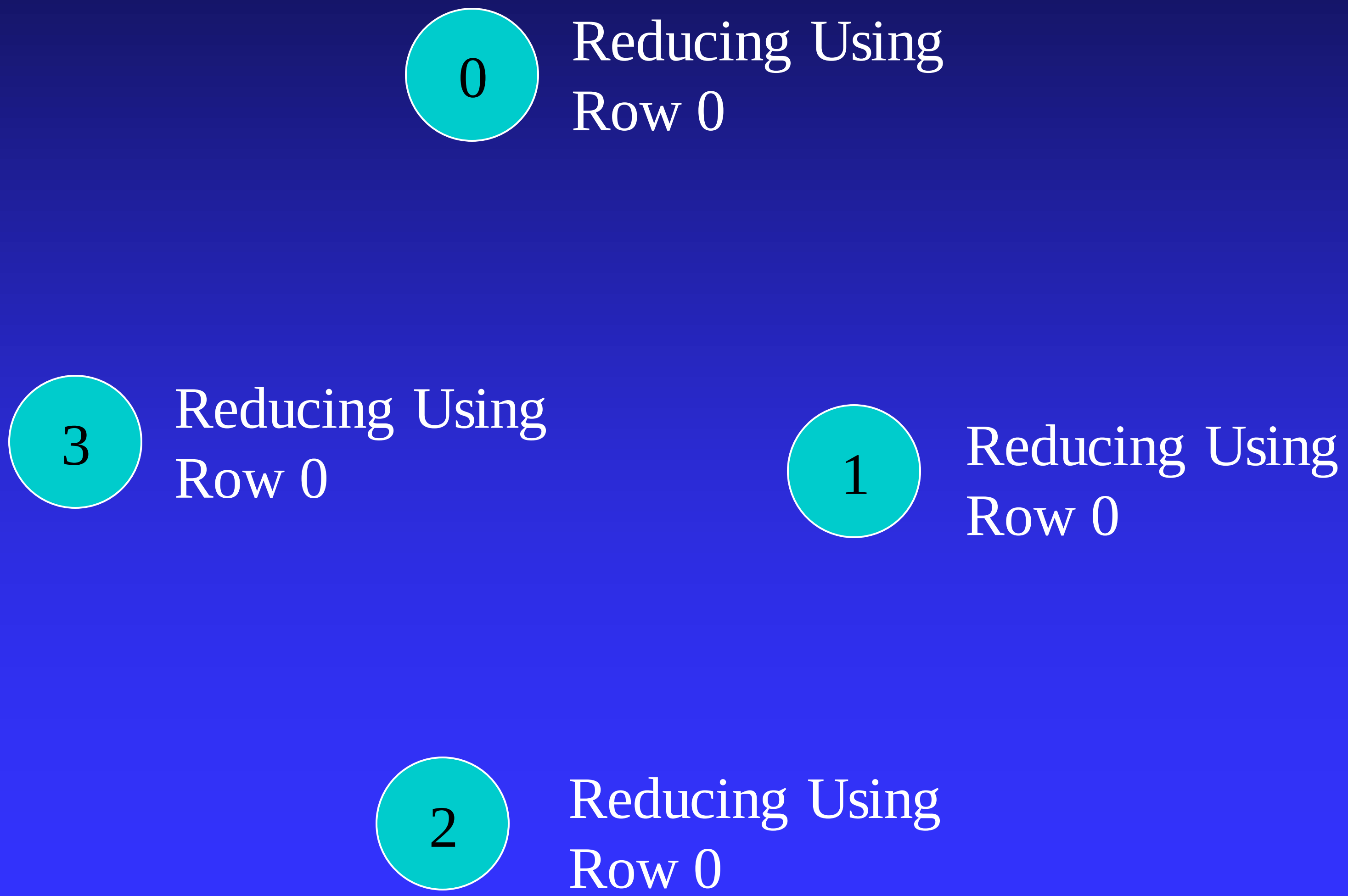
Communication Pattern



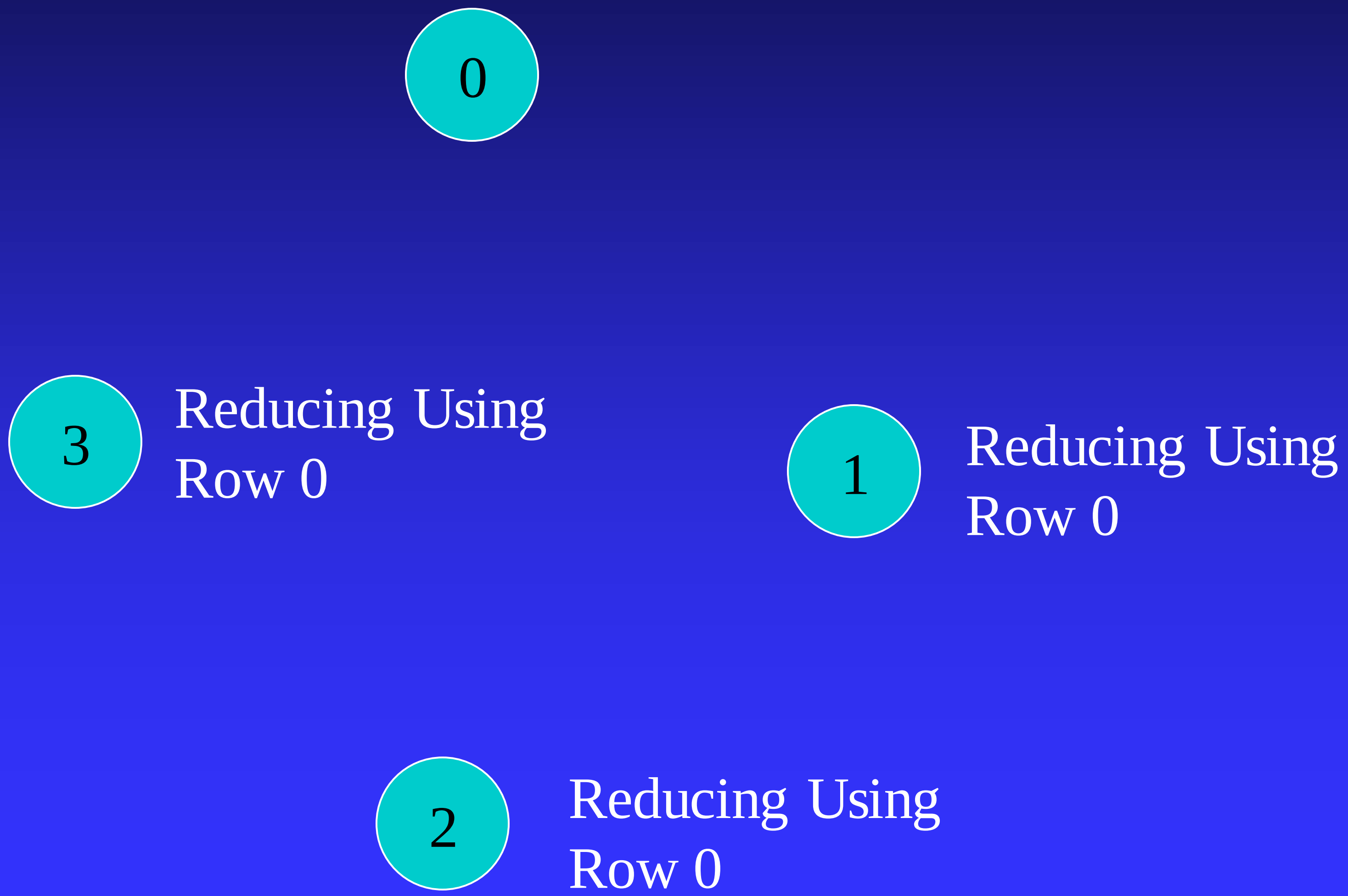
Communication Pattern



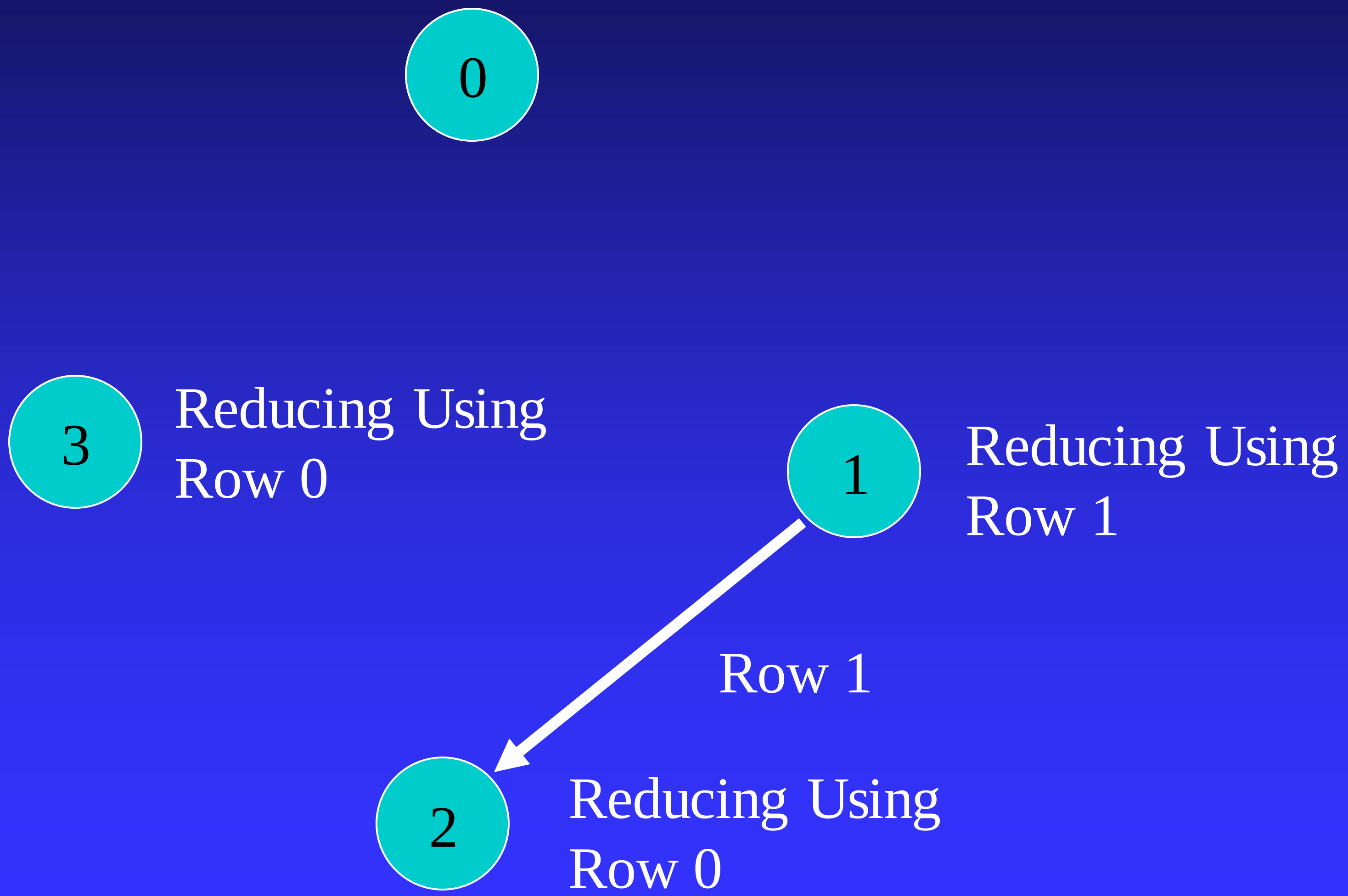
Communication Pattern



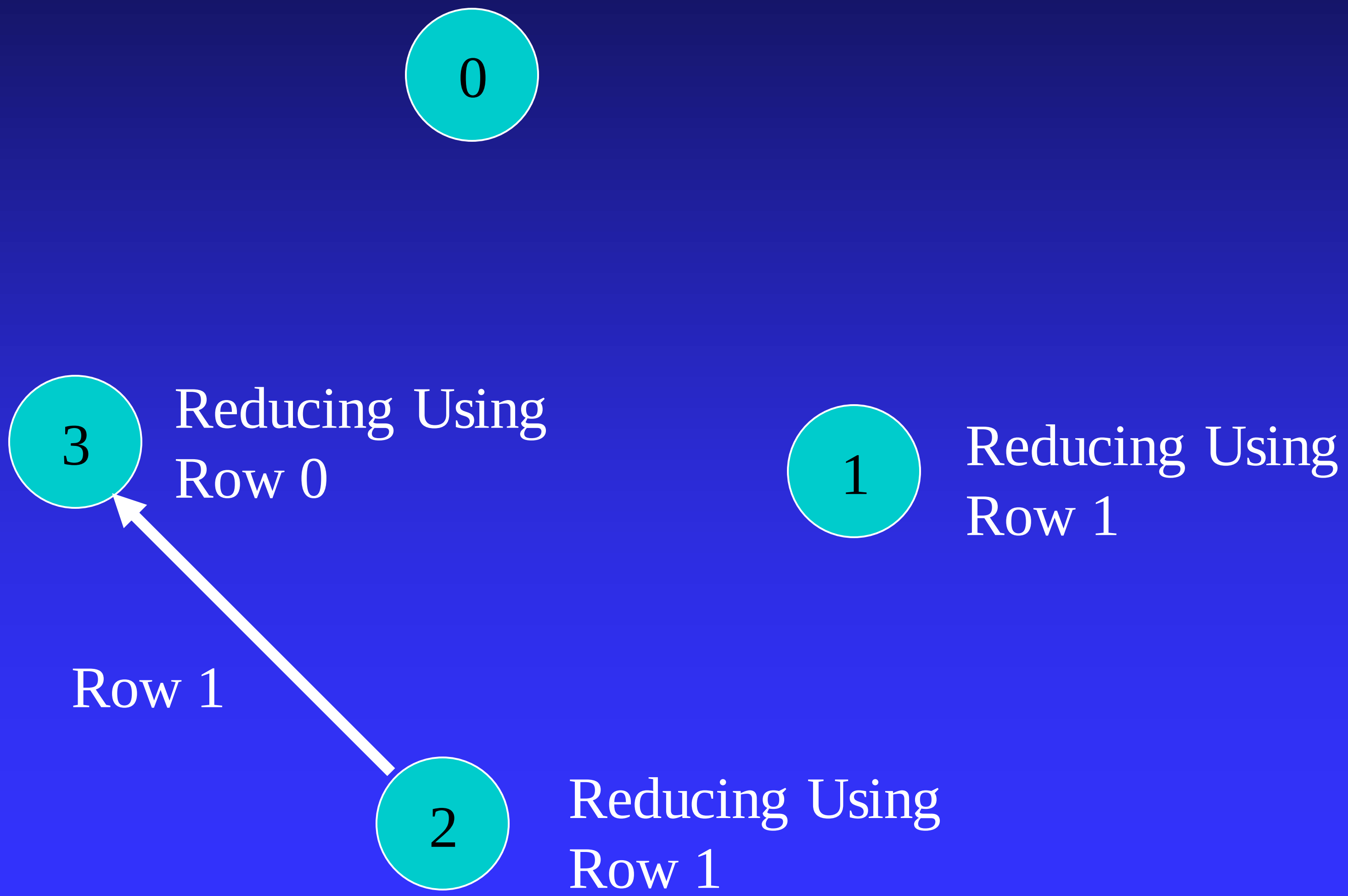
Communication Pattern



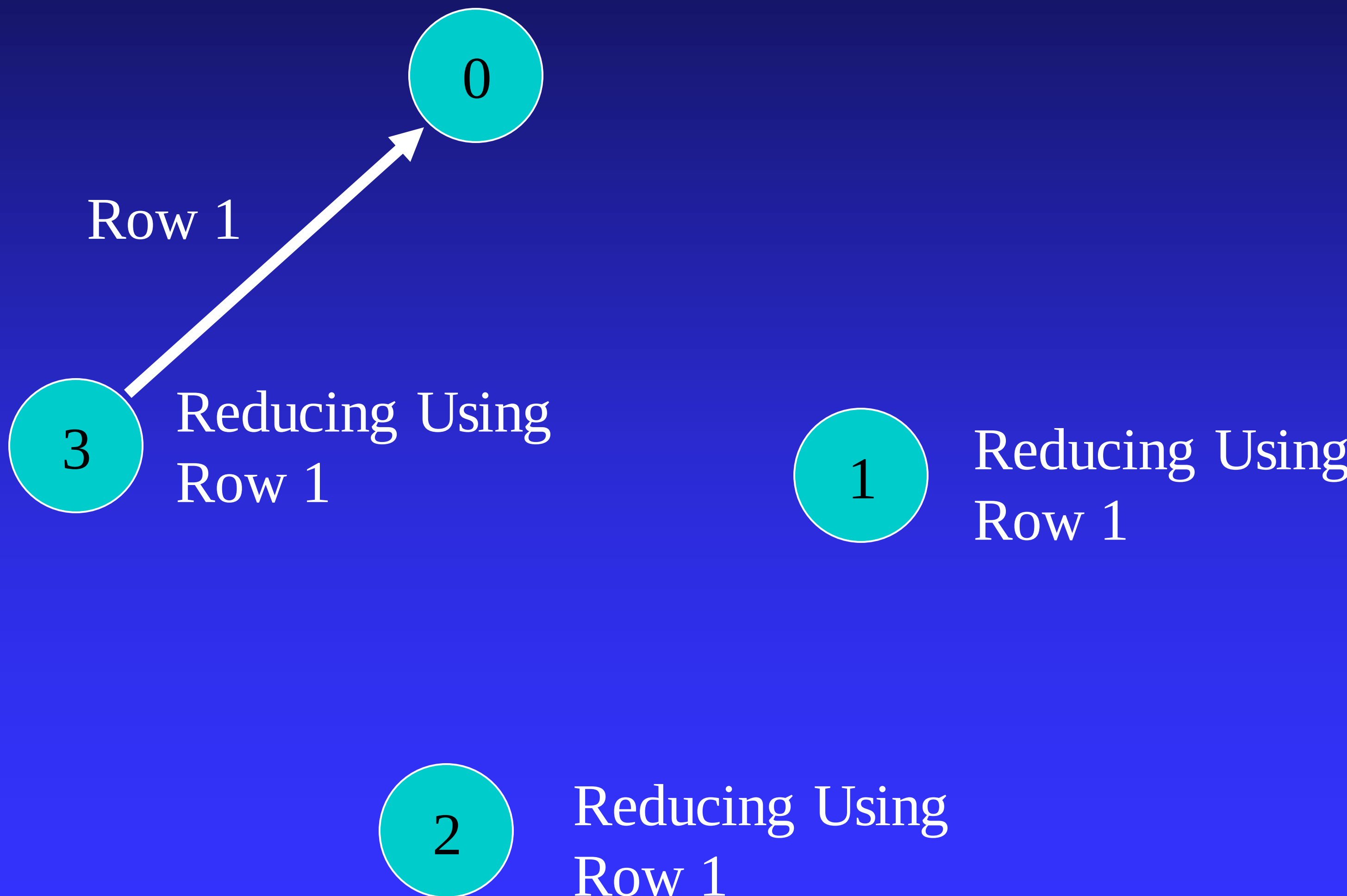
Communication Pattern



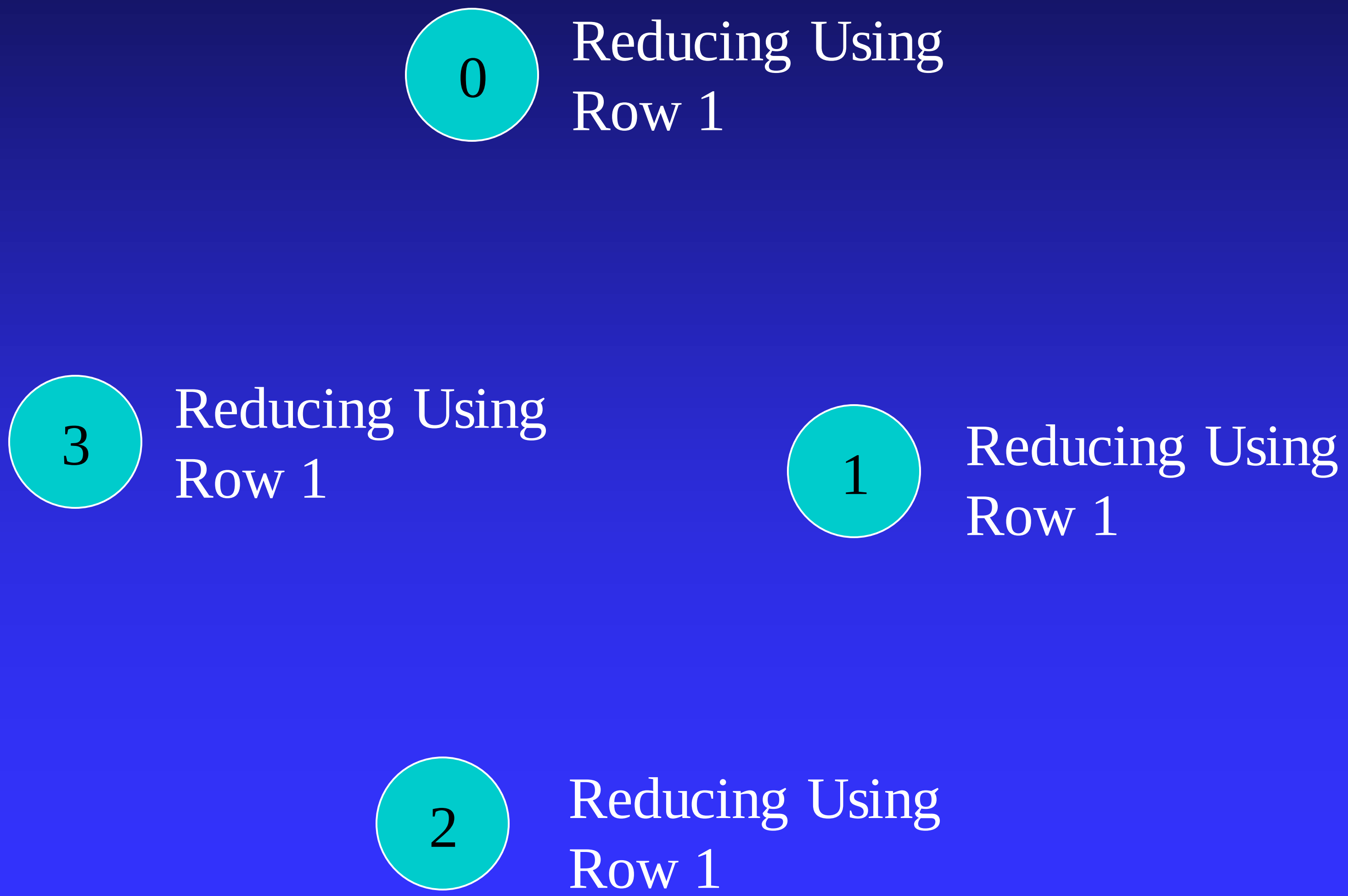
Communication Pattern



Communication Pattern



Communication Pattern



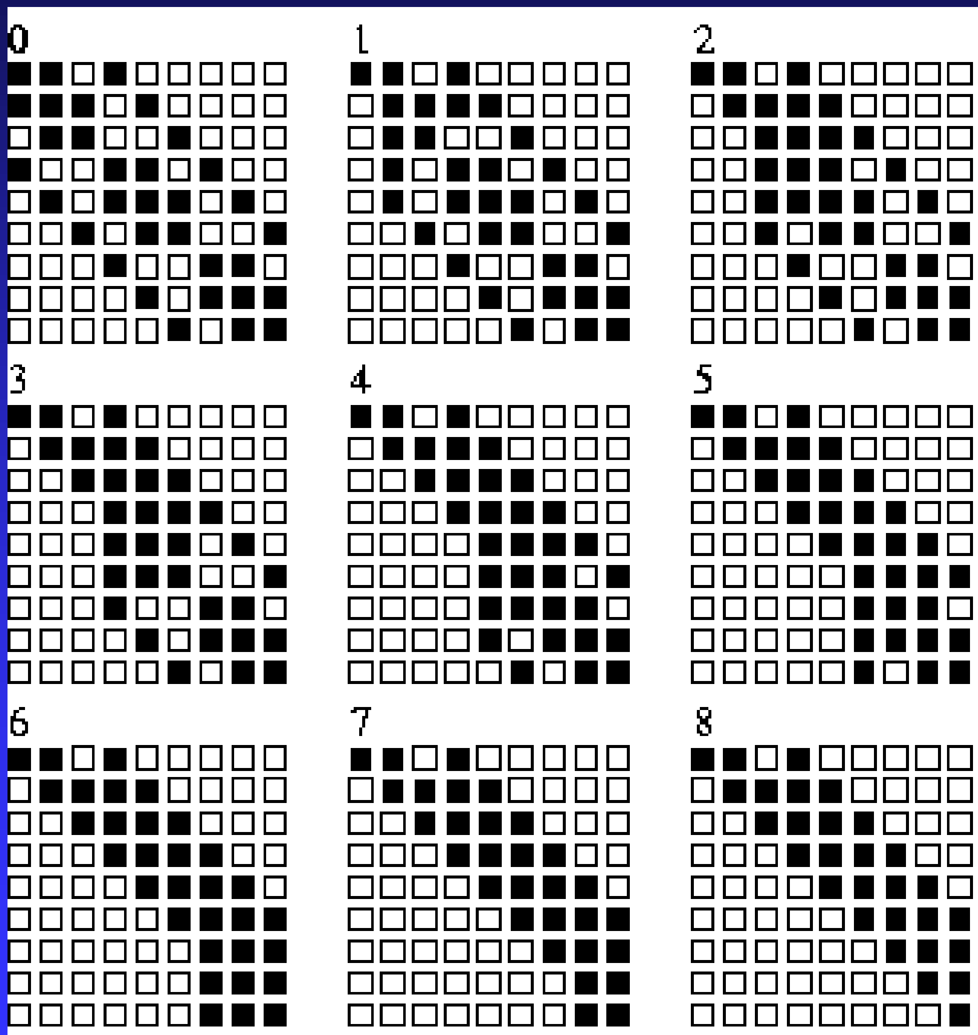
Analysis

- Total computation time: $\Theta(n^3/p)$
- Total message transmission time: $\Theta(n^2)$
- When n large enough, message transmission time completely overlapped by computation time
- Message start-up not overlapped: $\Theta(n)$
- Parallel overhead: $\Theta(np)$

Sparse Systems

- Gaussian elimination not well-suited for sparse systems
- Coefficient matrix gradually fills with nonzero elements
- Result
  - Increases storage requirements
  - Increases total operation count

Example of “Fill”



Iterative Methods

- Iterative method: algorithm that generates a series of approximations to solution’s value
- Require less storage than direct methods
- Since they avoid computations on zero elements, they can save a lot of computations

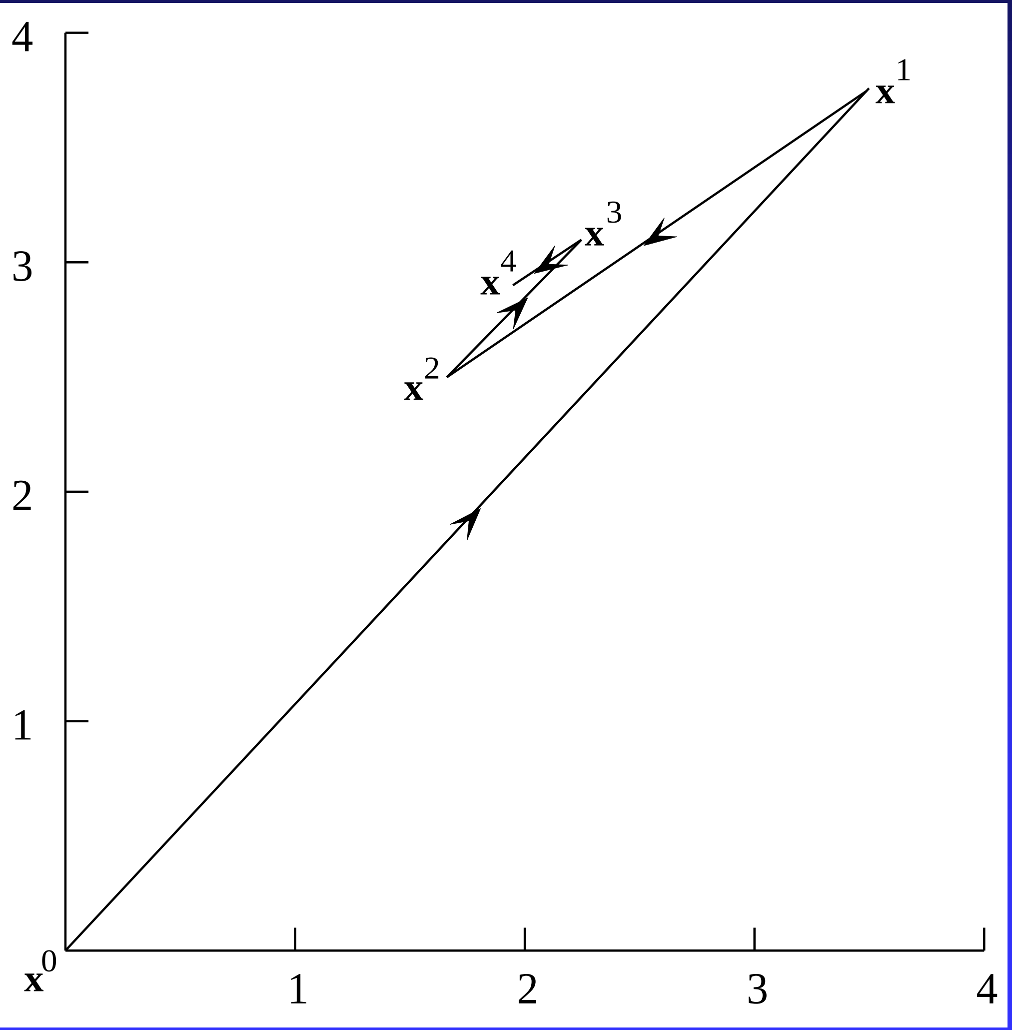
Jacobi Method

$$x_i^{k+1} = \frac{1}{a_{i,i}}\left(b_i - \sum_{j \neq i} a_{i,j} x_j^k\right)$$

Values of elements of vector *x* at iteration *k*+1 depend upon values of vector *x* at iteration *k*

Gauss-Seidel method: Use latest version available of x_i

Jacobi Method Iterations



Rate of Convergence

- Even when Jacobi method and Gauss-Seidel methods converge on solution, rate of convergence often too slow to make them practical
- We will move on to an iterative method with much faster convergence

Conjugate Gradient Method

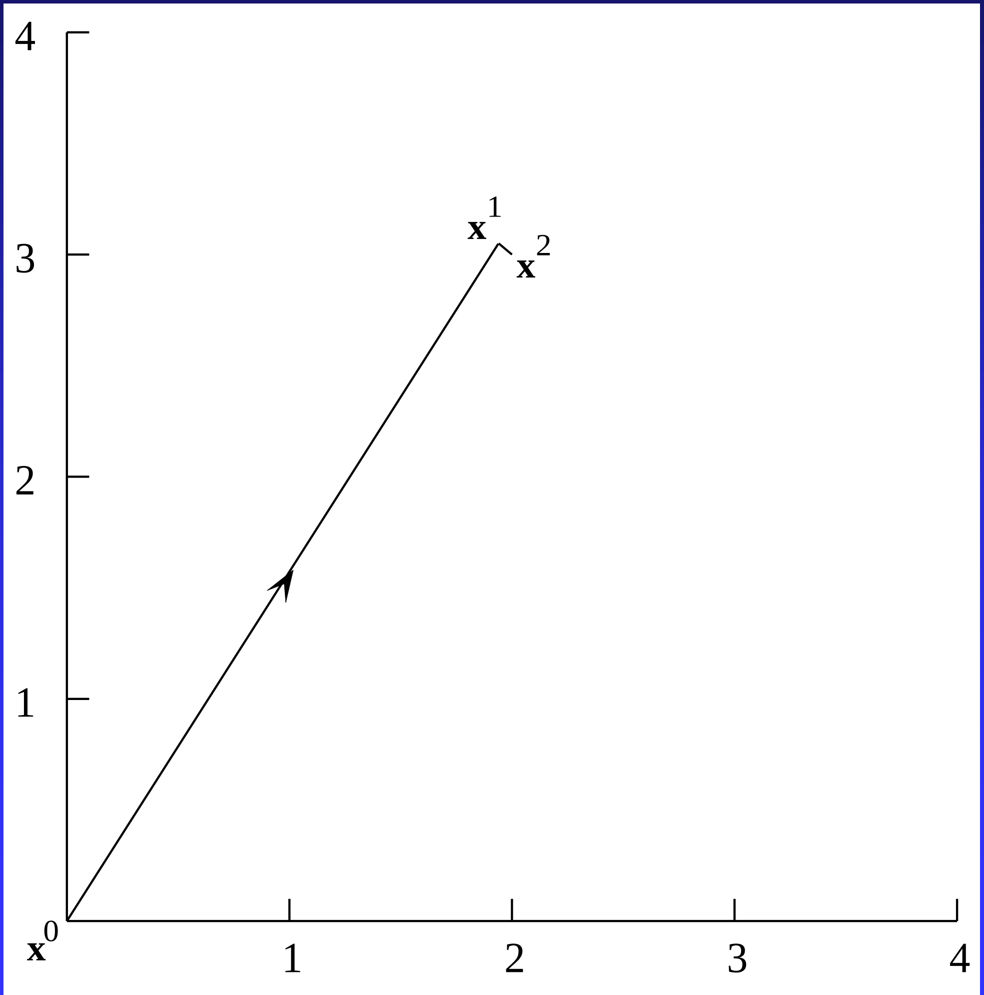
- A is positive definite if for every nonzero vector x and its transpose x^T, the product $x^T A x > 0$
- If A is symmetric and positive definite, then the function

$$q(x) = \tfrac{1}{2} x^T A x - x^T b + c$$

has a unique minimizer that is solution to $Ax = b$

- Conjugate gradient is an iterative method that solves $Ax = b$ by minimizing $q(x)$

Conjugate Gradient Convergence

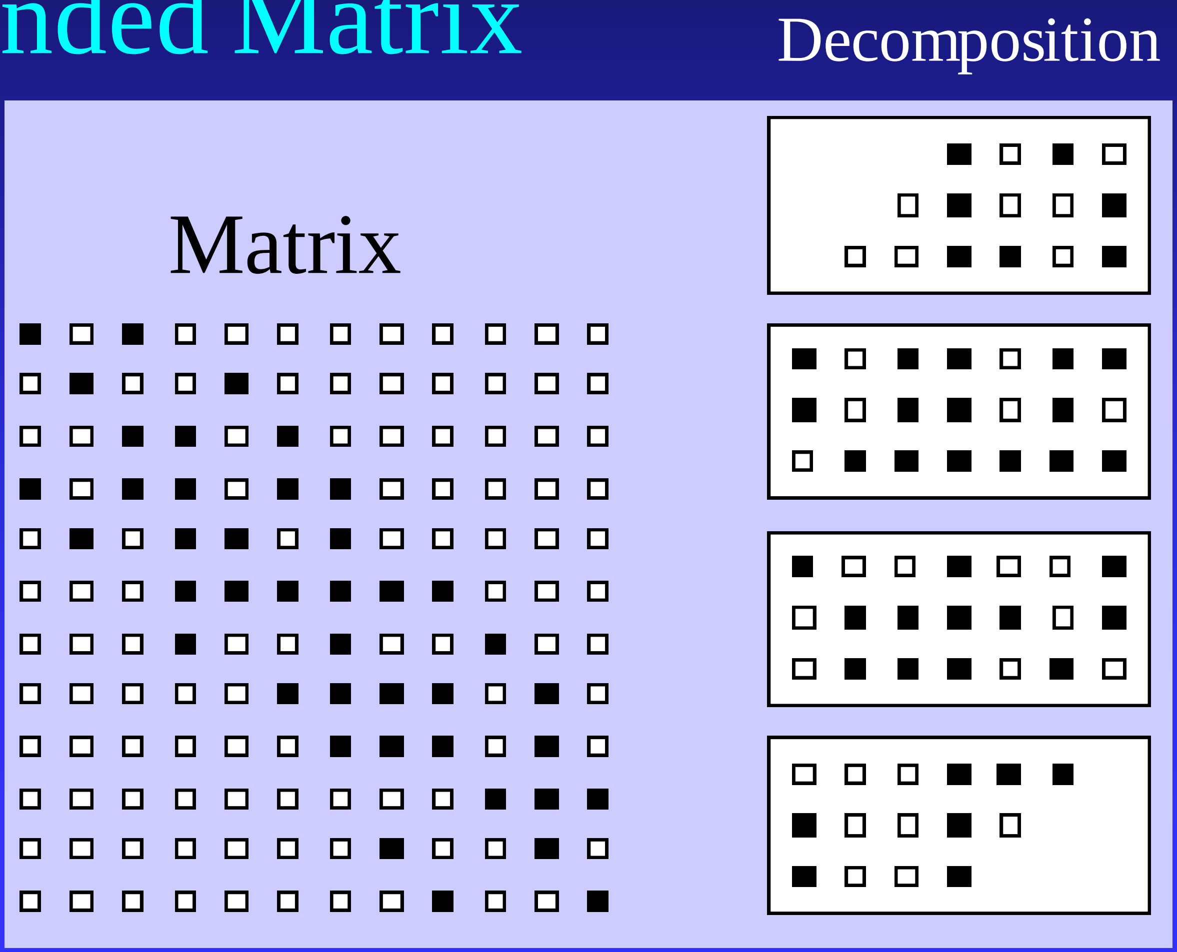


Finds value of *n*-dimensional solution in at most *n* iterations

Conjugate Gradient Computations

- Matrix-vector multiplication
- Inner product (dot product)
- Matrix-vector multiplication has higher time complexity
- Must modify previously developed algorithm to account for sparse matrices

Rowwise Block Striped Decomposition of a Symmetrically Banded Matrix



Representation of Vectors

- Replicate vectors
  - Need all-gather step after matrix-vector multiply
  - Inner product has time complexity $\Theta(n)$
- Block decomposition of vectors
  - Need all-gather step before matrix-vector multiply
  - Inner product has time complexity $\Theta(n/p + \log p)$

Summary

- Solving systems of linear equations
  - Direct methods
  - Iterative methods
- Parallel designs for
  - Back substitution
  - Gaussian elimination
  - Conjugate gradient method