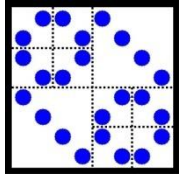
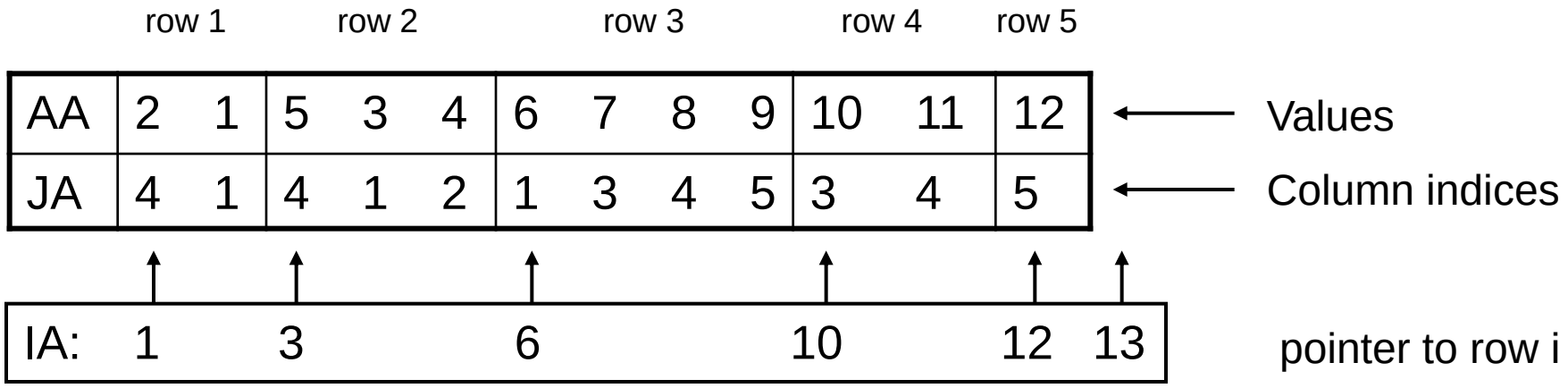
4.1.2 Compressed Sparse Row Format: CSR



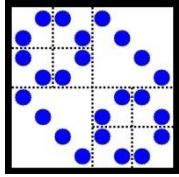


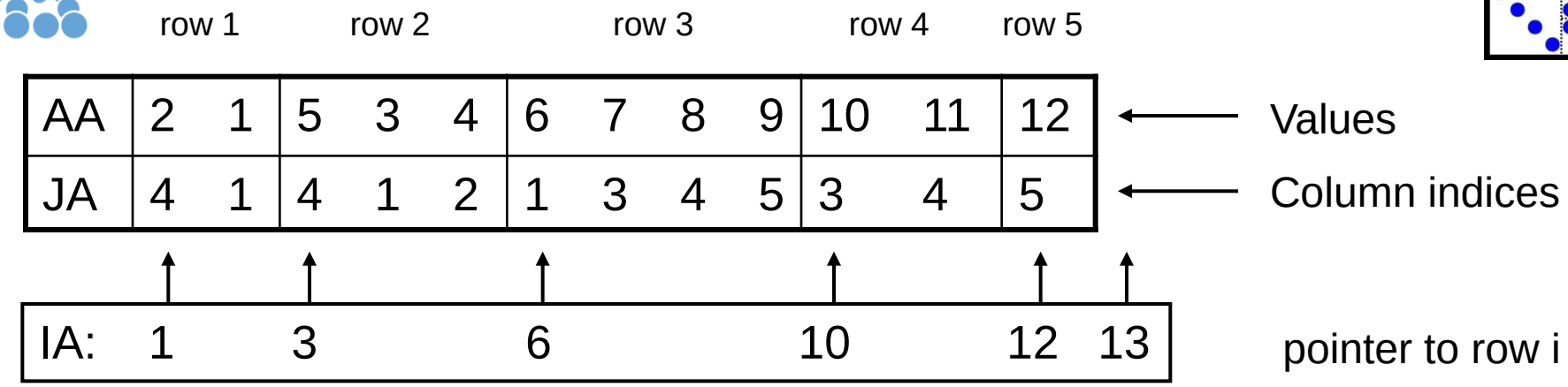
Storage:

n and nnz,

n+nnz+1 integer,

nnz float.





Code for computing c = A*b:

```
c = 0;
for i = 1 : n
    for  j = IA(i) : IA(i+1)-1
         c_i = c_i + AA(j)*b_JA(j) ;
    end
end
```

Indirect addressing only in b.

Columnwise → compressed sparse column format

CSR with extracted main diagonal

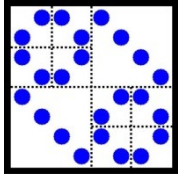
main diagonal entries | nondiagonal entries in CSR

AA	1	4	7	11	12	*	2	3	5	6	8	9	10
JA	7	8	10	13	14	14	4	2	4	1	4	5	3

Pointer to begin of i-th row:

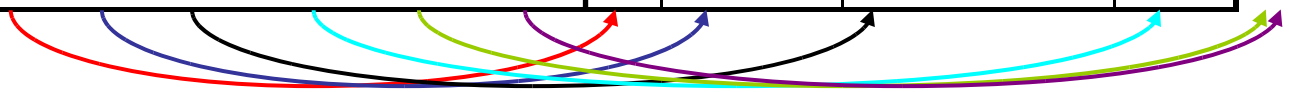
Storage:

n and nnz,
nnz + 1 integer,
nnz + 1 float



main diagonal entries | nondiagonal entries in CSR

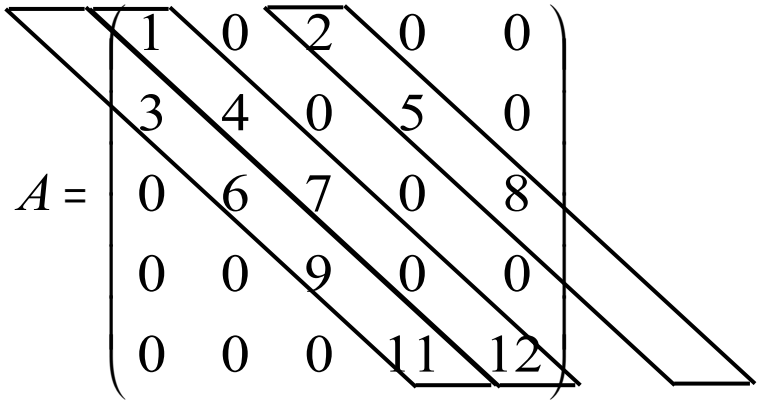
AA	1	4	7	11	12	*	2	3	5	6	8	9	10
JA	7	8	10	13	14	14	4	2	4	1	4	5	3



Code for c=A*b

```
for  i = 1 : n
       c_i = AA_i * b_i ;
       for  j = JA(i) : JA(i+1)-1
            c_i = c_i + AA_j*b_JA(j) ;
       end
end
```

4.1.4 Diagonalwise storage



$$A = \begin{pmatrix} 1 & 0 & 2 & 0 & 0 \\ 3 & 4 & 0 & 5 & 0 \\ 0 & 6 & 7 & 0 & 8 \\ 0 & 0 & 9 & 0 & 0 \\ 0 & 0 & 0 & 11 & 12 \end{pmatrix}$$

Diagonal number -1 0 2

Values in: $DIAG = \begin{pmatrix} * & 1 & 2 \\ 3 & 4 & 5 \\ 6 & 7 & 8 \\ 9 & 10 & * \\ 11 & 12 & * \end{pmatrix}$, $IOFF = \begin{pmatrix} -1 & 0 & 2 \end{pmatrix}$

Storage: n, nd = number of diagonals,
nd integers for IOFF and n*nd float

4.1.5 Rectangular Storage Scheme by Pressing from the Right

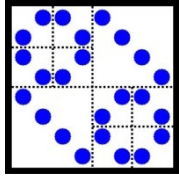
$$\begin{vmatrix} 1 & 0 & 2 & 0 & 0 \\ 3 & 4 & 0 & 5 & 0 \\ 0 & 6 & 7 & 0 & 8 \\ 0 & 0 & 9 & 10 & 0 \\ 0 & 0 & 0 & 11 & 12 \end{vmatrix} \longleftarrow gives \qquad COEF = \begin{pmatrix} 1 & 2 & 0 \\ 3 & 4 & 5 \\ 6 & 7 & 8 \\ 9 & 10 & 0 \\ 11 & 12 & 0 \end{pmatrix}$$

$$JCOEF = \begin{pmatrix} 1 & 3 & * \\ 1 & 2 & 4 \\ 2 & 3 & 5 \\ 3 & 4 & * \\ 4 & 5 & * \end{pmatrix}$$

nl := nnz of longest row.

Storage: n,

n*nl integer and float



Code for c = A b:

```
c = 0;
for  i = 1 : n
      for  j = 1 : nl
            c_i = c_i + COEFF(i,j) * b(JCOEFF(i,j));
      end
end
```

This format was used in ELLPACK
(package of subroutines for elliptic PDE).

Coordinate form is used by MATLAB.

4.1.6 Jagged Diagonal Form

Prestep: Sort rows after their length. Long rows first.

$$A = \begin{pmatrix} 1 & 0 & 2 & 0 & 0 \\ 3 & 4 & 0 & 5 & 0 \\ 0 & 6 & 7 & 0 & 8 \\ 0 & 0 & 9 & 10 & 0 \\ 0 & 0 & 0 & 11 & 12 \end{pmatrix} \Rightarrow PA = \begin{pmatrix} 3 & 4 & 0 & 5 & 0 \\ 0 & 6 & 7 & 0 & 8 \\ 1 & 0 & 2 & 0 & 0 \\ 0 & 0 & 9 & 10 & 0 \\ 0 & 0 & 0 & 11 & 12 \end{pmatrix}$$

Length 3

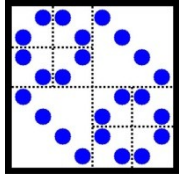
Length 2

Values of PA: $DJ = (3 \quad 6 \quad 1 \quad 9 \quad 11 \mid 4 \quad 7 \quad 2 \quad 10 \quad 12 \mid 5 \quad 8 \mid)$

First jagged diagonal second jagged diag.

Column indices: $JDIAG = (1 \quad 2 \quad 1 \quad 3 \quad 4 \mid 2 \quad 3 \quad 3 \quad 4 \quad 5 \mid 4 \quad 5 \mid)$

Pointer to beginning of j-th diagonal: $IDIAG = (1 \quad 6 \quad 11 \quad 13)$



NDIAG = number of jagged diagonals

Storage: n, NDIAG, nnz float, nnz + NDIAG integer

Code for c = A b:

```
c = 0;
for  j = 1 : NDIAG
     for  i = 1 : IDIAG(j+1) - IDIAG(j)
          k = IDIAG(j) + i - 1;
          c_i = c_i + DJ(k) * b(JDIAG(k));
     end
end
```

← Length of j-th jagged diag.

Advantages:

- Always start with row 1.
- More operations on neighboring data.
- Less indirect addressing.
- Pre-permutation changes only rows. Can be done implicitly.

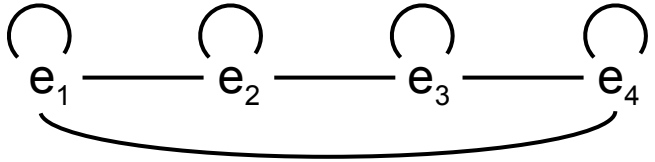
4.2 Sparse Matrices and Graphs

4.2.1 Graph G(A) for symmetric positive definite spd $A=A^T>0$

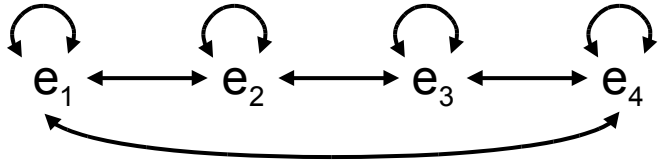
n x n –matrix: vertices $e_1, \ldots, e_n$ with edges (e_i,e_k) for $a_{ik} \neq 0$, undirected Graph

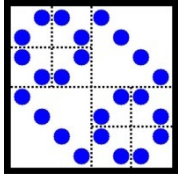
$$A = \begin{pmatrix} * & * & 0 & * \\ * & * & * & 0 \\ 0 & * & * & * \\ * & 0 & * & * \end{pmatrix} \longrightarrow$$

G(A):



G(A) as directed graph:





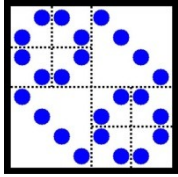
Adjacency Matrix for G(A) or A:

$$A(G(A)) = \begin{pmatrix} 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 \end{pmatrix}$$

can be obtained directly by replacing in A each nonzero entry by 1.

Symmetric permutations of A in the form $P A P^T$ change the ordering of the rows and columns of A simultaneously.

Therefore, the graph of $P A P^T$ can be obtained by the graph of A by renumbering the vertices:

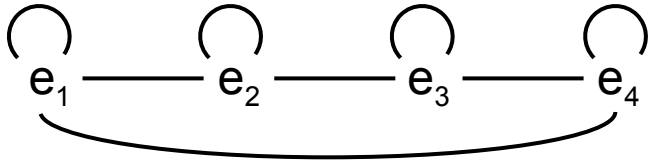


Matrix A with graph G(A)

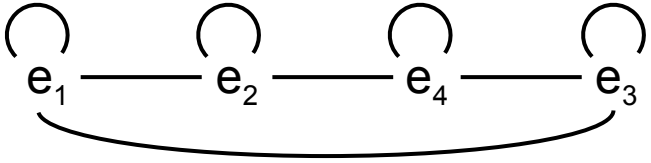
Symmetric permutation P A P^T with graph G(P A P^T)

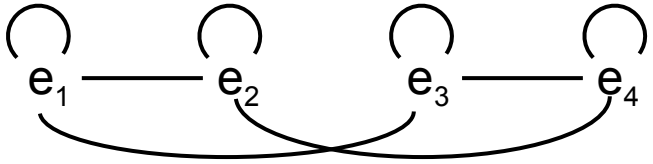
Example: P permutation that changes 3 ←→4:

G(A):



G(PAP^T):

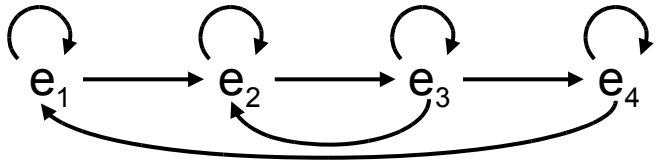




4.2.2 Matrix A nonsymmetric, G(A) directed

$$A = \begin{pmatrix} * & * & 0 & 0 \\ 0 & * & * & 0 \\ 0 & * & * & * \\ * & 0 & 0 & * \end{pmatrix} \quad \Rightarrow \quad A(G(A)) = \begin{pmatrix} 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 \\ 1 & 0 & 0 & 1 \end{pmatrix}$$

G(A):



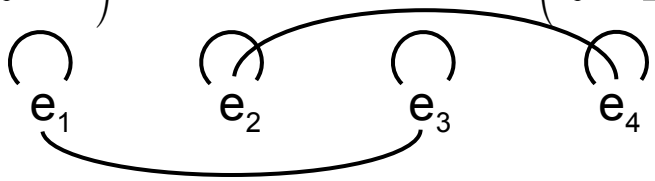
How can we characterize „good“ sparsity patterns?

„good“: Gaussian Elimination can be reduced to smaller subproblems or produces no (or small) fill-in.

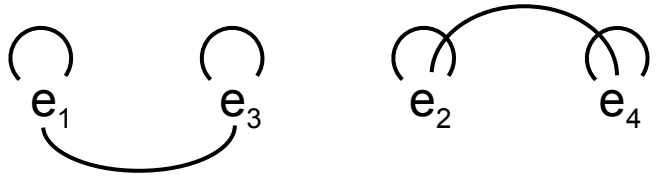
Block Diagonal Pattern

$$A = \begin{pmatrix} * & 0 & * & 0 \\ 0 & * & 0 & * \\ * & 0 & * & 0 \\ 0 & * & 0 & * \end{pmatrix} \Rightarrow A(G(A)) = \begin{pmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \end{pmatrix}$$

G(A):



2 ←→ 3:

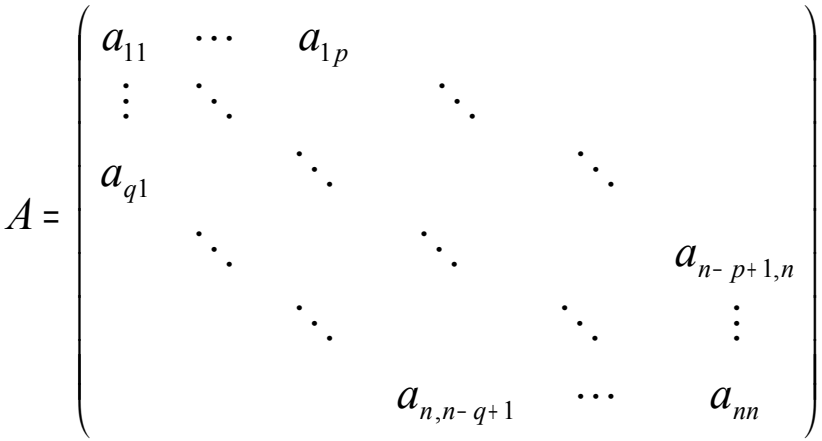


$$PAP^T = \begin{pmatrix} * & * & 0 & 0 \\ * & * & 0 & 0 \\ 0 & 0 & * & * \\ 0 & 0 & * & * \end{pmatrix} = \begin{pmatrix} A_1 & 0 \\ 0 & A_2 \end{pmatrix}$$

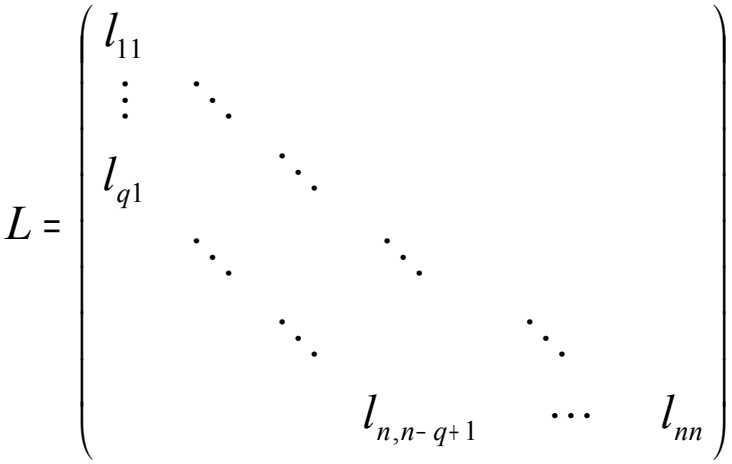
By this permutation, A can be transformed into block diagonal form → easy to solve!

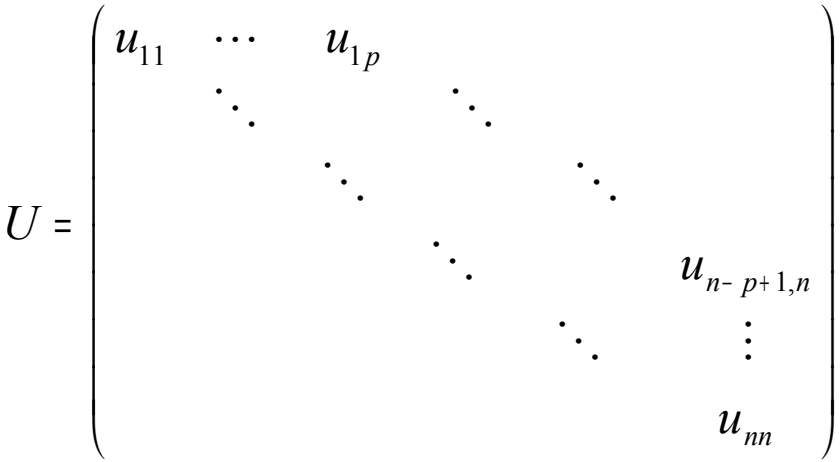
$$\begin{pmatrix} A_1 & 0 \\ 0 & A_2 \end{pmatrix}^{-1} = \begin{pmatrix} A_1^{-1} & 0 \\ 0 & A_2^{-1} \end{pmatrix}$$

Banded Pattern



Gauss Elimination without pivoting preserves the sparsity pattern

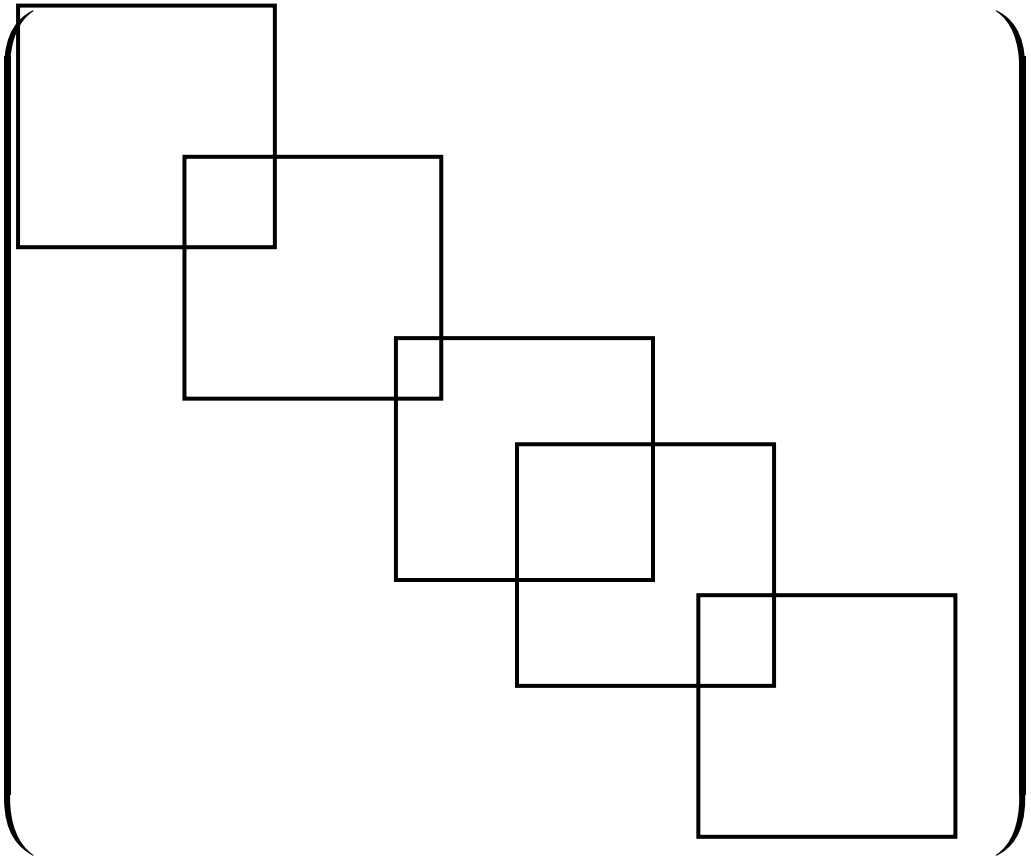




With pivoting the bandwidth in U grows, but remains <= p+q.

Overlapping Block Diagonal

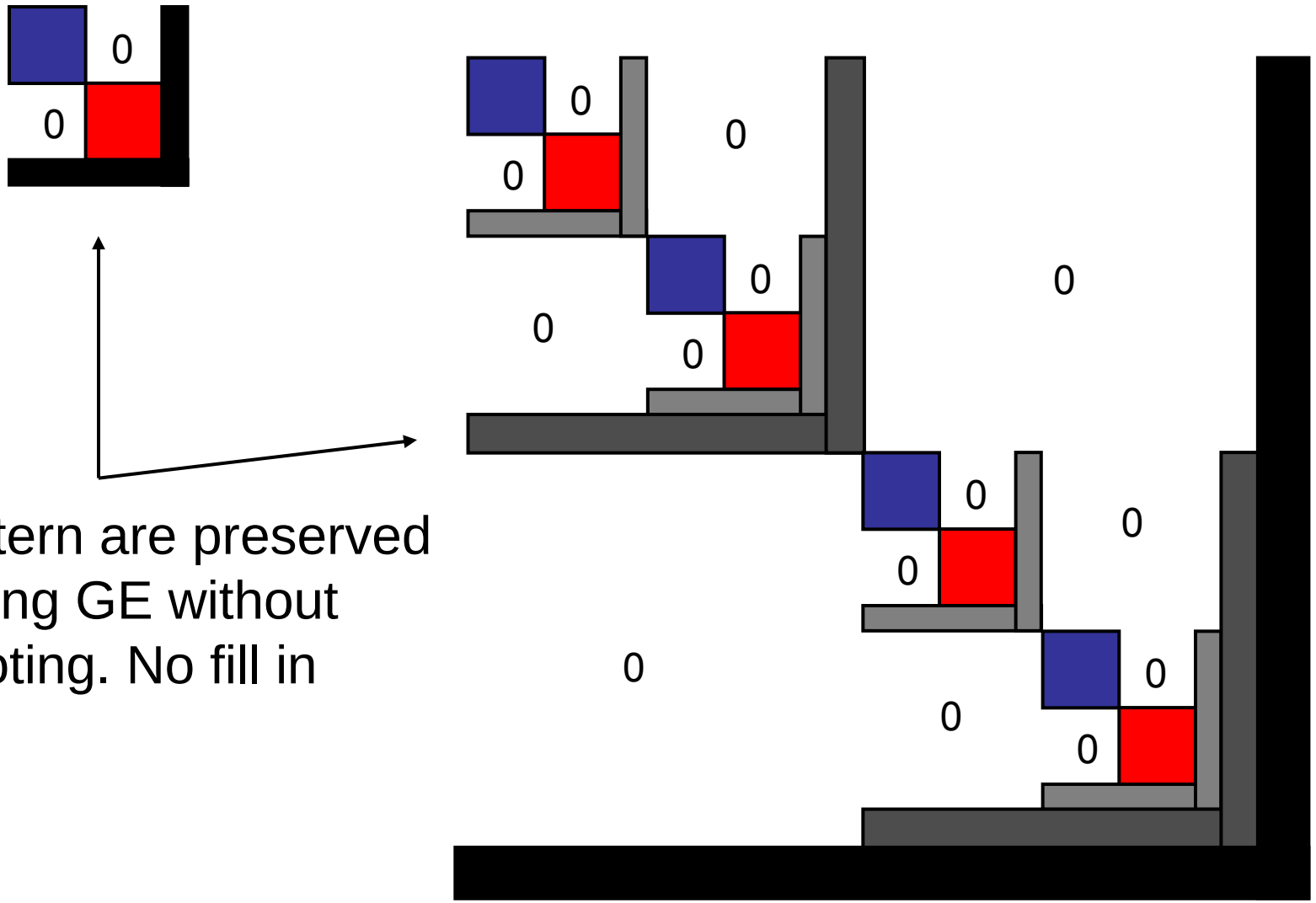
$A =$



Pattern is preserved by Gaussian Elimination (in case of restricted pivoting).

Dissection Form

Nested (recursive) dissection:



Pattern are preserved during GE without pivoting. No fill in

Schur Complement Reduction

Write matrix B in terms of smaller submatrices:

$$\begin{pmatrix} B_1 & B_2 \\ B_3 & B_4 \end{pmatrix} \cdot \begin{pmatrix} B_1^{-1} & D \\ 0 & S^{-1} \end{pmatrix} = \begin{pmatrix} I & B_1 D + B_2 S^{-1} \\ B_3 B_1^{-1} & B_3 D + B_4 S^{-1} \end{pmatrix} \overset{!}{=} \begin{pmatrix} I & 0 \\ * & I \end{pmatrix}$$

To satisfy this equation we have to set:

$$B_1 D + B_2 S^{-1} = 0 \Rightarrow D = -B_1^{-1} B_2 S^{-1}$$

$$B_3 D + B_4 S^{-1} = I \Rightarrow I = -B_3 B_1^{-1} B_2 S^{-1} + B_4 S^{-1}$$

$$\Rightarrow \underbrace{S = B_4 - B_3 B_1^{-1} B_2}_{\text{S Schur Complement}} \quad \text{and} \quad D = -B_1^{-1} B_2 S^{-1}$$

$$B = \begin{pmatrix} B_1 & B_2 \\ B_3 & B_4 \end{pmatrix} = \begin{pmatrix} I & 0 \\ B_3 B_1^{-1} & I \end{pmatrix} \cdot \begin{pmatrix} B_1 & B_2 \\ 0 & S \end{pmatrix}$$

Therefore, solving linear system in B is reduced to solving two smaller linear systems, one in B_1 and the other in the Schur complement S.

B sparse → B_1 also sparse, but S usually dense!

Example: Schur complement and dissection form:

$$A = \left(\begin{array}{cc|c} A_1 & 0 & F_1 \\ 0 & A_2 & F_2 \\ \hline G_1 & G_2 & A_3 \end{array}\right)$$

Schur complement:

$$S = A_3 - \begin{pmatrix} G_1 & G_2 \end{pmatrix} \cdot \begin{pmatrix} A_1^{-1} & 0 \\ 0 & A_2^{-1} \end{pmatrix} \cdot \begin{pmatrix} F_1 \\ F_2 \end{pmatrix} =$$

$$= A_3 - G_1 A_1^{-1} F_1 - G_2 A_2^{-1} F_2$$

Direct derivation of Schur complement:

$$\begin{pmatrix} A_1 & 0 & F_1 \\ 0 & A_2 & F_2 \\ G_1 & G_2 & A_3 \end{pmatrix} \cdot \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \\ b_3 \end{pmatrix} \Rightarrow \begin{array}{lllll} A_1x_1 + & & F_1x_3 & = & b_1 \\ & A_2x_2 + & F_2x_3 & = & b_2 \\ G_1x_1 + & G_2x_2 + & A_3x_3 & = & b_3 \end{array}$$

$$x_1 = A_1^{-1}b_1 - A_1^{-1}F_1x_3$$

$$\Rightarrow$$

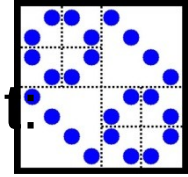
$$x_2 = A_2^{-1}b_2 - A_2^{-1}F_2x_3$$

$$\Rightarrow (G_1A_1^{-1}b_1 - G_1A_1^{-1}F_1x_3) + (G_2A_2^{-1}b_2 - G_2A_2^{-1}F_2x_3) + A_3x_3 = b_3$$

$$\Rightarrow (A_3 - G_1A_1^{-1}F_1 - G_2A_2^{-1}F_2)x_3 = b_3 - G_1A_1^{-1}b_1 - G_2A_2^{-1}b_2$$

$$\Rightarrow Sx_3 = \tilde{b}_3$$

Algorithm for solving Ax=b based on Schur complement:



1. Compute S by using inv(A_1) and inv(A_2)

2. Solve $Sx_3 = \tilde{b}_3$

3. Compute x_1 and x_2 by using inv(A_1) and inv(A_2)

The explicit computation of S can be avoided by solving the linear System in S iteratively, e.g. Jacobi, pcg, ….
Then we need only part of S and in every iteration step we have to compute S * intermediate vector.

To achieve fast convergence, a preconditioner (approximation) for S has to be used!

Iterative methods and preconditioning will be subject of later chapters.