JICS
Joint Institute for Computational Science

Tutorial Workshop

Linear Algebra Library

Kwai Lam Wong
kwong@utk.edu

Joint Institute for Computational Sciences
University of Tennessee, Knoxville
http://www.jics..utk.edu

Contents

- BLAS
- LAPACK
- SCALAPACK
- Aztec
- PETSc
- SuperLU, MUMPS
- Finite Element example
- References

JICS
Joint Institute for Computational Science

Numerical Linear Algebra Libraries

- Generally the most important kernels for scientific computations
- Portable, scalable and optimized state-of-the-art software written and tested by experts or vendors
- Basic Algebraic Calculations :
  - Basic Linear Algebra Subprograms (BLAS, ATLAS, GOTO, ESSL, MKL)
- Dense Matrices Libraries
  - Linear Algebra Package (LAPACK, ESSL)
  - Scalable Linear Algebra Package (ScaLAPACK, PESSL)
- Sparse Direct Solvers
  - www.nersc.gov/~xiaoye/SuperLU : SuperLU
  - www.enseeiht.fr/apo/MUMPS : (MUMPS)
- Sparse Matrices Libraries
  - Portable, Extensible, Toolkit for Scientific Computation (PETSc)
  - Aztec, Trilinos
- ODE – Suite of Nonlinear and Differential/Algebraic equation Solvers (SUNDIALS, PVODE) – www.llnl.gov/CASC/sundials

Basic Linear Algebra Subprograms (BLAS)

JICS
Joint Institute for Computational Science

Basic Linear Algebra Subprograms (BLAS)

- **BLAS is a library of standardized basic linear algebra computational kernels created to perform efficiently on serial computers taking into account the memory hierarchy of modern processors.**
- **BLAS1 does vectors-vectors operations.**
  - **Saxpy = y(i) = a* x(i) + y(i), ddot= $\sum$ x(i) *y(i)**
- **BLAS2 does matrices - vectors operations.**
  - **MV : y = A x + b**
- **BLAS3 operates on pairs or triples of matrices.**
  - **MM : $C = \alpha AB + \beta C$, Triangular Solve : $X = \alpha T^{-1} X$**
- **Level3 BLAS is created to take full advantage of the fast cache memory. Matrix computations are arranged to operate in block fashion. Data residing in cache are reused by small blocks of matrices.**

Basic Linear Algebra Subroutine (BLAS)

- **BLAS is a library of standardized basic linear algebra computational kernels created to perform efficiently on serial computers taking into account the memory hierarchy of modern processors.**

- **The Level 1 BLAS or BLAS1 operate mostly on vectors or pairs of vector. These routines perform O(n) operations, and return either a vector or a scalar. Examples are**
  - **A saxpy operation : y(i) = a* x(i) + y(i)** $sdot = \sum_{i=1}^{n} x^T y$
  - **A scaling operation : y (i) = a* x (i)**
  - **A dot product :**

- **Examples : saxpy, scaling, and dot product**
  - **SUBROUTINE SAXPY(N, ALPHA, X, INCX, Y, INCY)**
  - **SUBROUTINE SSCAL(N, ALPHA, X, INCX)**
  - **SUBROUTINE SDOT(N, X, INCX, Y, INCY)**

BLAS2

- **The ratio of floating-point operations to data movement is too low to achieve effective use of most vector or parallel machines, even scalar machines. Thus, Level2 BLAS was designed to optimize reuse of data in registers and reduction in memory access.**

- **The Level2 BLAS or BLAS2 operate mostly on a matrix (2D array) and a vector (or vectors),returning a matrix or a vector. If the array is n-by-n, O(n^2) operations are performed. Examples are:**
  - **Matrix-vector multiplication :** $y = \alpha Ax + \beta y$
  - **A rank-one update :** $A = \alpha xy^T + A$
  - **A triangular solve :** $x = T^{-1}x$ **, where T is a triangular matrix**

- **Examples : matrix vector multiplication, rank-one update, and triangular solve**
  - **SUBROUTINE SGEMV(TRANS, M, N, ALPHA, A, LDA, X, INCX, BETA, Y, INCY)**
  - **SUBROUTINE GER(M, N, ALPHA, X, INCX, Y, INCY, A, LDA)**
  - **SUBROUTINE TRSV(UPLO, TRANS, DIAG, N, A, LDA, X, INCX)**

BLAS3

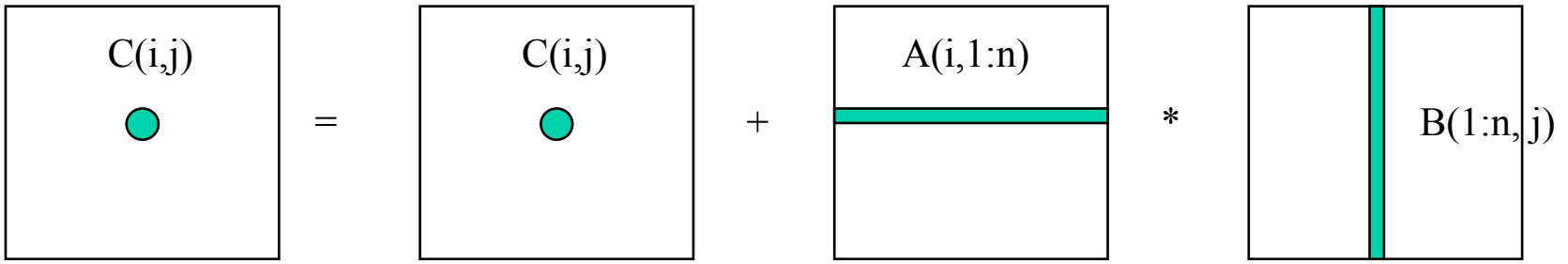
- **In order to maximize the ratio of floating-point operations to memory references, Level3 BLAS is create to take full advantage of modern computer architecture. Data residing in cache or local memory are reused by breaking up matrixes to smaller blocks. Matrix computations are arranged to operate in block fashion.**

- **The level3 BLAS or BLAS3 operate on pairs or triples of matrices, returning a matrix. Examples are:**
  - **Matrix-matrix multiplication :** $C = \alpha AB + \beta C$
  - **Multiple triangular solve :** $X = \alpha T^{-1} X$ **, where T is a triangular matrix, and X is a rectangular matrix**

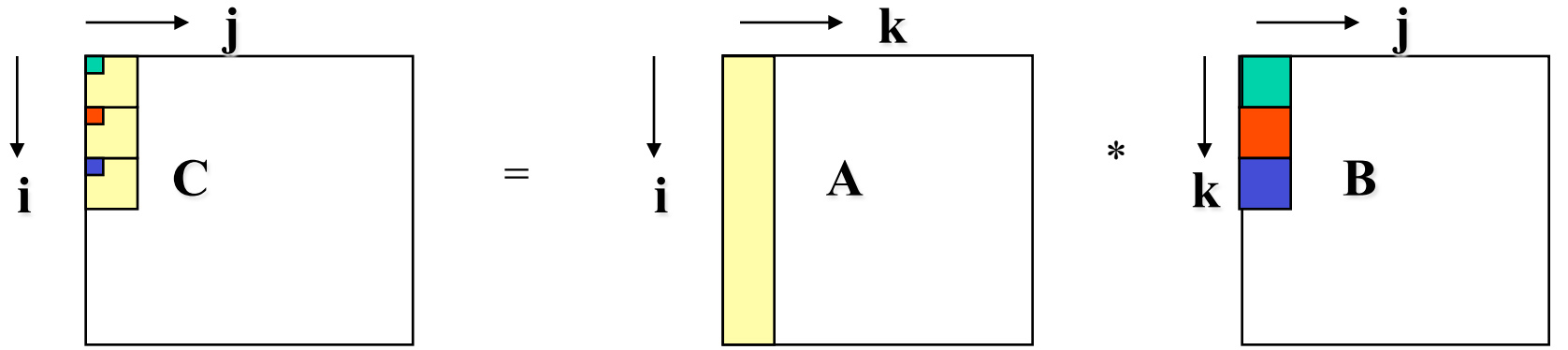
- **Examples : matrix matrix multiplication and multiple triangular solve**
  - **SUBROUTINE SGEMM(TRANSA, TRANSB, M, N, K, ALPHA, A, LDA, B, LDB, BETA, C, LDC)**
  - **SUBROUTINE STRSM(SIDE, UPLO, TRANSA, DIAG, M, N, ALPHA, A, LDA, B, LDB)**

MM Multiplication

- **Simple MM - q = average number of flops per memory reference ~ 2**



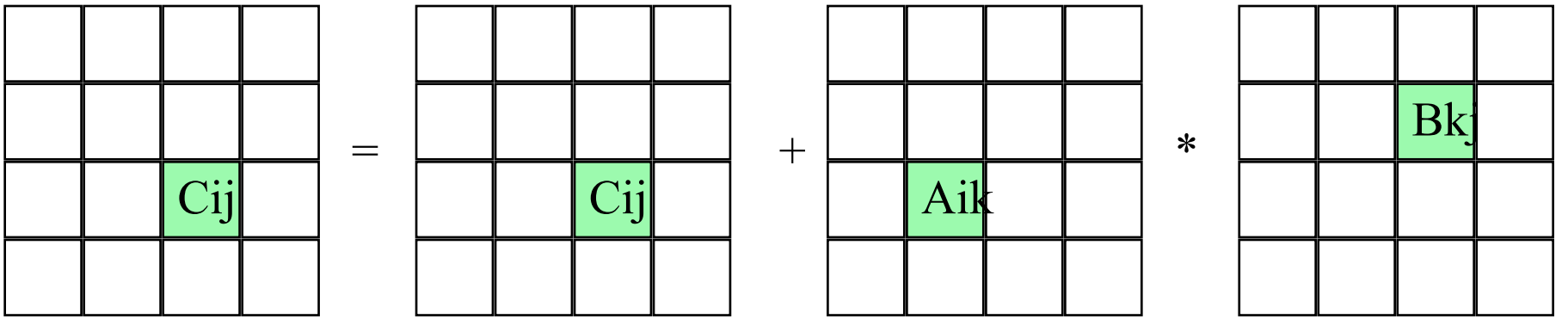
- **Performance of MM can be improved by rearranging the order of multiplication indices in column fashion in Fortran (kji) or in row fashion in C.**



k - j - i ordering for FORTRAN

Block MM

- q = f/m = (2*n^3) / ((2*N + 2) * n^2) ~ n / N , N=1 => q=n/2
- If N is equal to 1, the algorithm is ideal. However, N is bounded by the amount of fast cache memory. However, N can be taken independently to the size of matrix, n.
- The optimal value of N = sqrt (size of fast memory / 3)



$$Cij = Cij + \sum_{k=1}^{n} Aik * Bkj$$

ATLAS

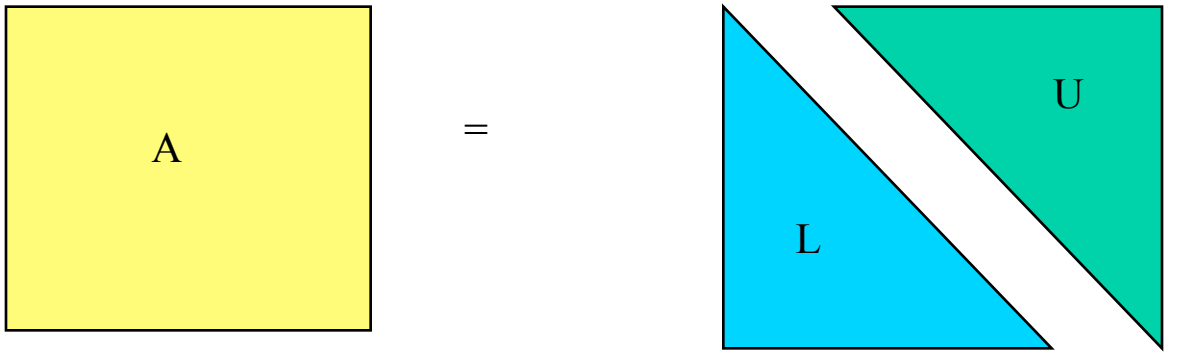
- Automatically Tuned Linear Algebra Software
- It generates a set of optimized linear algebra routines on different computer architectures taking the advantages of their specific memory hierarchies and pipelined functional units.
- In version 3.0, it supports all level of BLAS kernels as well as some LAPACK routines.
- It also provides interfaces to standard C (need cblas.h) and fortran 77.
- Prebuilt ATLAS for various computer architectures are readily available on the web.
- Good for Linux Platform
- www.netlib.org/atlas

Linear Algebra Package (LAPACK)

Gausssian Elimination

$$Ax = b$$

change A into A = L U



so $LUx = b$

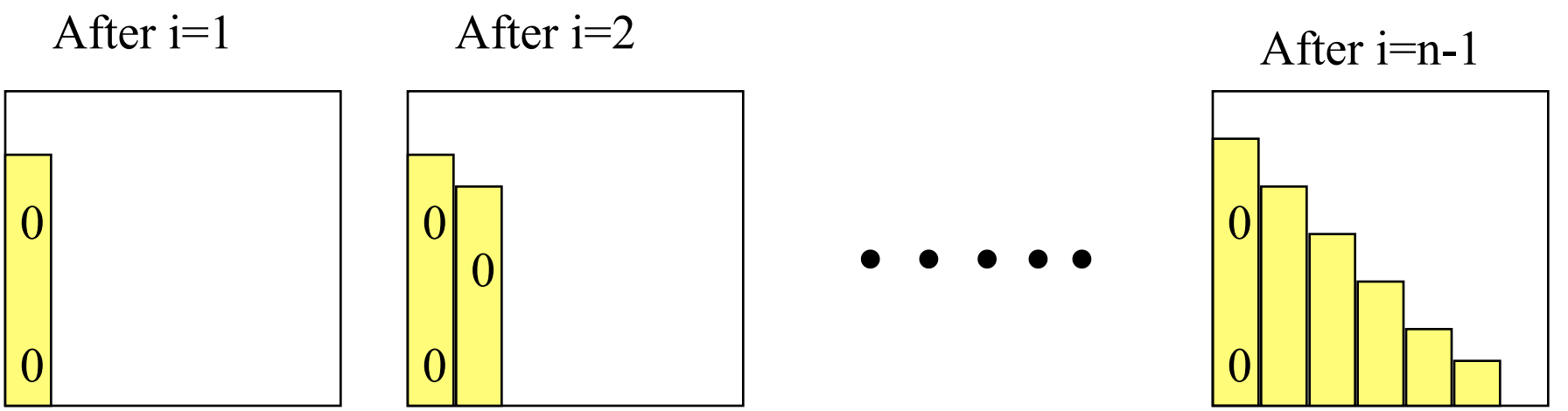
first solve $Ly = b$ by direct downward solve

then solve $Ux = y$ by direct upward solve

Gaussian Elimination

- **For each column i, zero out the element below the diagonal by adding multiples of row i to later rows**

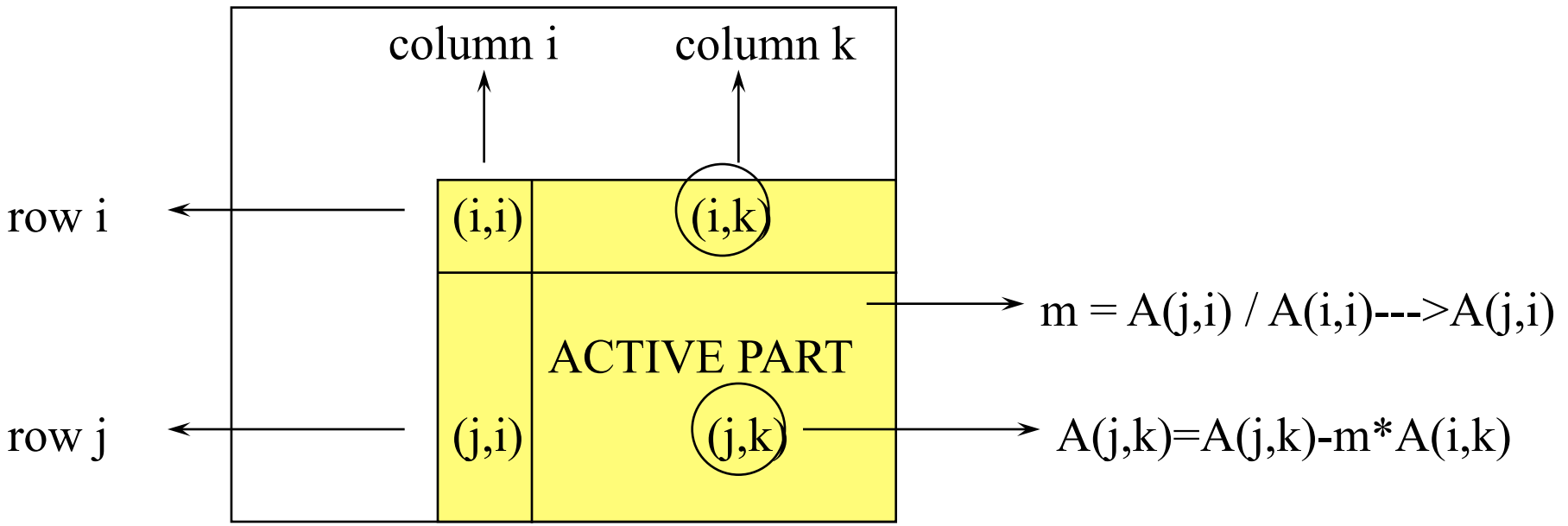
```
for i= 1 to n-1
    for j = i+1 to n
        for k = i to n
            A(j,k) = A(j,k) - (A(j,i) / A(i,i)) * A(i,k)
```



Gaussian Elimination (2)

- To improve the implementation, the constant A(j,i) / A(i,i) is removed from the innermost loop. Zeros below the diagonal is ignored

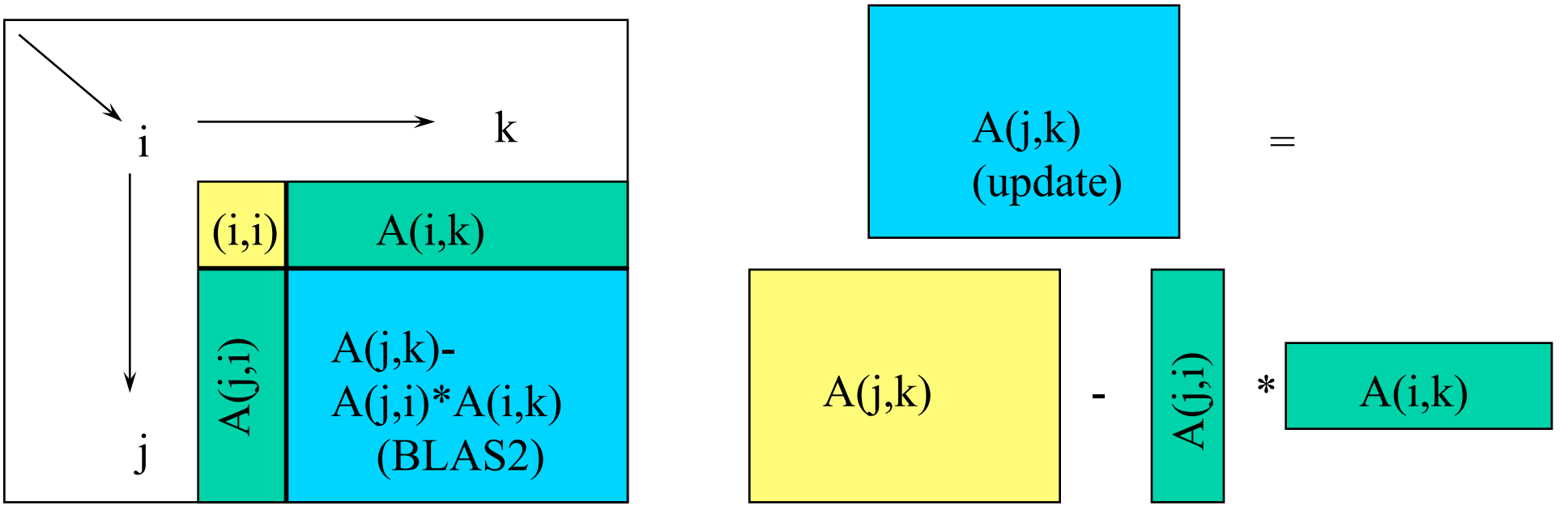
```
for i = 1 to n-1
    for j = i+1 to n
        m = A(j,i) / A(i,i) ----> m = A(j,i)
        for k = i to n
        A(j,k) = A(j,k) - m * A(i,k)
```



LAPACK (LU)

- **The inner loop consists of BLAS1 and one BLAS 2 operations.**

```
for i = 1 to n-1
        for j = i+1 to n
            A(j,I) = A(j,i) / A(i,i)   <------ BLAS1 ( to BLAS2)
            for k = i+1 to n
            A(j,k) = A(j,k) - A(j,i)* A(i,k)   <----BLAS2 ( to BLAS3)
```



LAPACK GE Block Algorithm

- **The block size of bk columns will depend on the machine architectures. It is generally small enough so that bk columns currently used for factorization fit in the fast memory of the machine, and bk is also large enough to make matrix matrix multiplication perform effectively.**
- **The principle is the same as in the ordinary GE algorithm above. Instead of working with one column, A(j,i) or one pivot entry, A(I,I), a block of columns and a square block of matrix are used. Hence, BLAS1 operations will become BLAS2 operations, and BLAS2 operations will become BLAS3 operation**

choose a block size bk

for ib = 1 , n , bk

1) **L U factorize the column block of bk**
2) **compute the pivoting block of rows**
3) **update the remaining block of the square matrix**

LAPACK GE Block Algorithm

I) Choose bk

II) for ib = 1 to n-1 step bk

Work the colored portion of A

1) LU factorize A22+A32

A32 <--(UU, LL, A32)

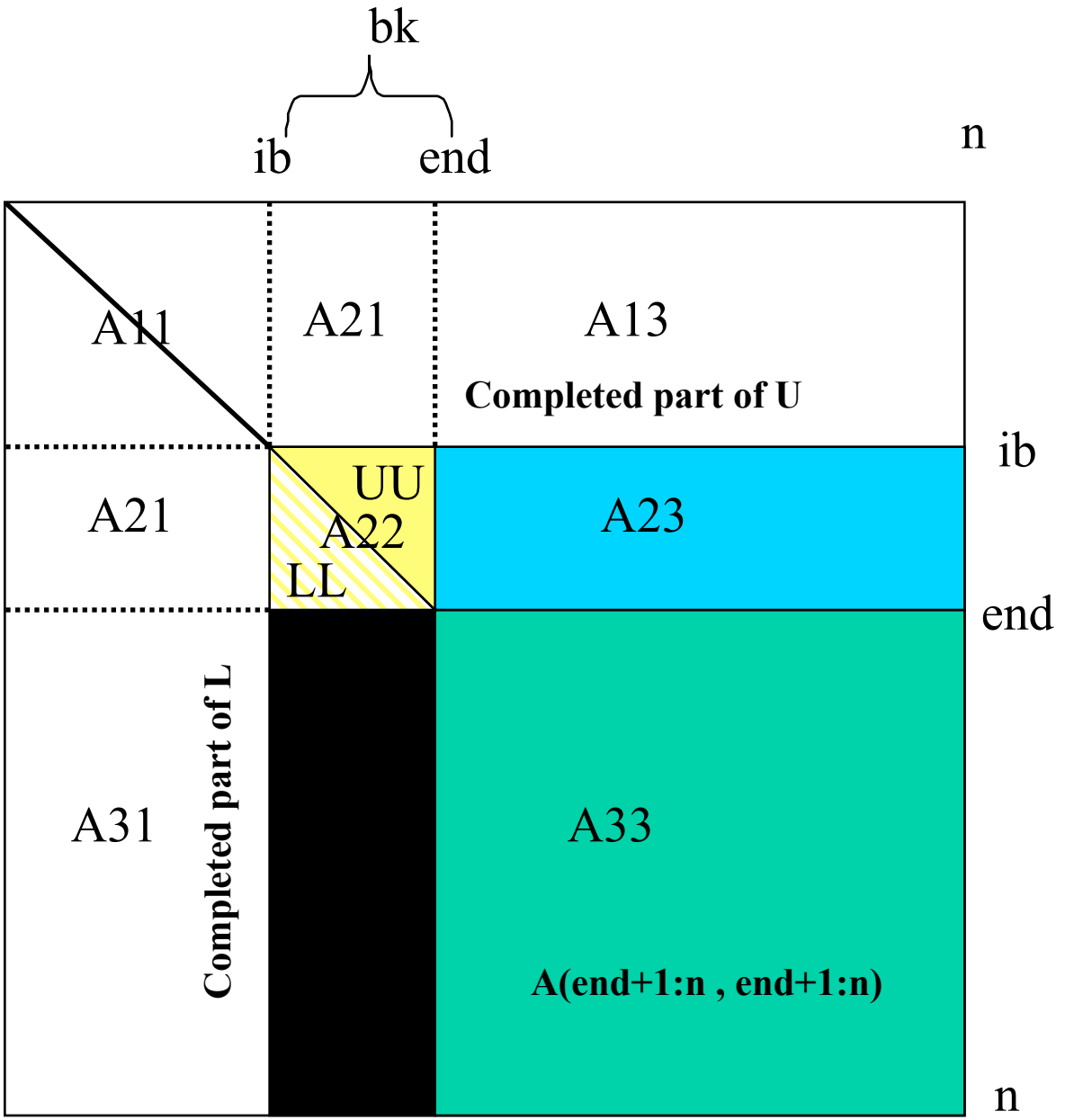
2) Update A23 : triangular solve

(A23) <-- LL \ A23

3) Update A33

A33 <-- (A33, A23, A32)

III) Triangular solve for unknown



LAPACK GE Algorithm

```
Choose appropriate size for bk
for ib = 1 to n-1 step bk
      point to the end of block of bk columns
      end = min (ib+bk-1,n)

      for I = ib to end
       find and record k where
                  |A(k,i) | = max | A(j,i)|
                  if |A(k,i)| = 0, exit with a warning, A is singular
                  if I not equal to k, swap rows of i and k of A
                   A(i+1:n,i) = A(i+1:n,i) /A(i,i)
                    A(i+1:n,I+1:end) = A(i+1:n,i+1:end) - A(i+1:n,i)* A(i,i+1:end)

      Let LL be the bk-by-bk lower triangular matrix whose subdiagonal entries are
      stored in A(ib:end, ib:end), and with 1s on the diagonal. Do delayed update
      of A(ib:end , end +1 : n) by solving n-end triangular system
      A(ib:end,end+1:n) =  LL \ A(ib:end, end+1 : n)

      Do delayed update of the rest of matrix using matrix-matrix multiplication
      A(end+1:n,end+1:n) = A(end+1:n,end+1:n)-
                                    A(end+1:n,ib:end)*A(ib:end,end+1:n)
```

Example (1)

- **Solve the following system of linear equations**

2	0	-1	1	0	2
4	1	-3	4	1	4
0	-1	2	-1	-3	-1
2	-1	0	0	0	0
-2	0	2	-2	-3	0
-2	0	-1	-3	5	0

x1
x2
x3
x4
x5
x6

=

-2
-3
8
3
0
-18

$$2x_1 - x_3 + x_4 + 2x_6 = -2$$
$$4x_1 + x_2 - 3x_3 + 4x_4 + x_5 + 4x_6 = -3$$
$$-x_2 + 2x_3 - x_4 - 3x_5 - x_6 = 8$$
$$2x_1 - x_2 = 3$$
$$-2x_1 + 2x_3 - 2x_4 - 3x_5 = 0$$
$$-2x_1 - x_3 - 3x_4 + 5x_5 = -18$$

Example (2)

- **Choose the column block size bk = 2, so ib = 1, 3, and 5**
- **For b = 2, ib = 1, n = 6, end = 2**

1) For i = ib to end (i = 1,2)

i = 1 a) **A(i+1: n,I) = A(i+1: n, I) / A(i,i) => A(2:6, 1)=A(2:6, 1) / A(1,1)**

A(i+1:n,i+1:end) = A(i+1:n, i+1:end) - A(i+1:n, I) * A(i, i+1:end)

b) **only update columns i+1 (2) to end (2) , so only column 2**

A(2:6, 2:2) = A(2:6, 2:2) - A(2:6, 1) * A(1, 2:2)

i = 2 a) **A(3:6, 1) = A(3:6, 1) / A(2, 2), since A(2,2) = 1 =>** DONE

1a)

4		2
0		0
2	/ 2 =	1
-2		-1
-2		-1

1b)

1		2	0	1
-1		0		-1
-1	-	1	=	-1
0		-1		0
0		-1		0

A(2:6,1) / A(1,1) = A(2:6,1)

A(2:6, 2:2) - A(2:6, 1) * A(1, 2:2)= A(2:6,2)

Example (3)

- **For bk = 2 , n = 6, end = 2,** ib=1

  2) Do delayed update of A(ib:end, end+1:n) by solving n-end triangular system

  A(1:2, 3:6) = LL \ A(1:2, 3:6)

LL =

1	0
2	1

UU=

2	0
0	1

1	0
2	1

?	?	?	?
?	?	?	?

=

-1	1	0	2
-3	4	1	4

LL * new A(1:2, 3:6) = A(1:2, 3:6)

=>

?	?	?	?
?	?	?	?

=

-1	1	0	2
-1	2	1	0

= A(1:2, 3:6)

Example (4)

- **For bk = 2, ib = 1, n = 6, end = 2**

  2) Do delayed update of rest of matrix using matrix-matrix multiplication

  A(end+1:n, end+1:n) = A(end+1:n, end+1:n) -A(end+1:n, ib:end) *A(ib:end, end+1:n)

  A(3:6, 3:6) = A(3:6, 3:6) -A(3:6, 1:2) * A(1:2, 3 : 6)

2	-1	-3	-1
0	0	0	0
2	-2	-3	0
-1	-3	-3	0

-

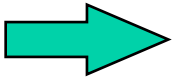
0	-1
1	-1
-1	0
-1	0

-1	1	0	2
-1	2	1	0

=

1	1	-2	-1
0	1	1	-2
1	-1	-3	2
-2	-2	-3	2

2	0	-1	1	0	2
4	1	-3	4	1	4
0	-1	2	-1	-3	-1
2	-1	0	0	0	0
-2	0	2	-2	-3	0
-2	0	-1	-3	-3	0



2	0	-1	1	0	2
2	1	-1	2	1	0
0	-1	1	1	-2	-1
1	-1	0	1	1	-2
-1	0	1	-1	-3	2
-1	0	-2	-2	-3	2

Example (5)

- **Choose the column block size bk = 2, so ib = 1, 3, and 5**
- **For bk = 2, ib = 3, n = 6, end = 4**

1) For I = ib to end (I = 3,4)

i= 3 a) **A(i+1: n, i) = A(i+1: n, i) / A(i,i) => A(4:6, 3)=A(4:6, 3) / A(3,3)**

b) **only update columns i+1 (4) to end (4) , so only column 4**

A(i+1:n,i+1:end) = A(i+1:n, i+1:end) - A(i+1:n, i) * A(i, i+1:end)

A(4:6, 4:4) = A(4:6, 4:4) - A(4:6, 3) * A(3, 4:4)

i = 4 a) **A(5:6, 4) = A(5:6, 4) / A(4, 4), A(4,4) = 1, => DONE**

1a)

0
1
-2

/ 1 =

0
1
-2

1b)

1
-1
-2

-

0
1
-2

1

=

1
-2
0

A(4:6,3) / A(3,3) = A(4:6,3)

A(4:6,4)-A(4:6,3) * A(3,4)=A(4:6,4)

2a)

-2
0

/ 1 =

-2
0

A(5:6,4) = A(5:6,4)/A(4,4)

Example (6)

- **For bk = 2 , n = 6, end = 2, ib=3**

  2) Do delayed update of A(ib:end, end+1:n) by solving n-end triangular system

A(3:4, 5:6) = LL \ A(3:4, 5:6)

LL = | 1 | 0 |
| --- | --- |
| 0 | 1 |

UU= | 1 | 1 |
| --- | --- |
| 0 | 1 |

1	0
0	1

?	?
?	?

=

-2	-1
1	-2

LL * new A(3:4, 5:6) = A(3:4, 5:6)

=>

?	?
?	?

=

-2	-1
1	-2

= A(3:4, 5:6)

Example (7)

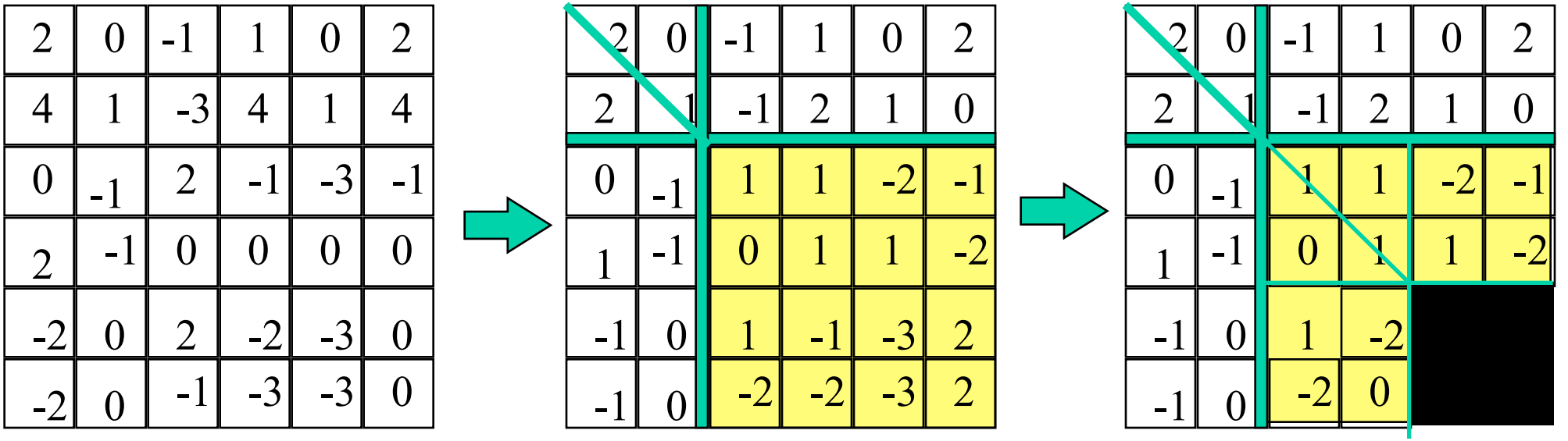
- **For bk = 2, ib = 3, n = 6, end = 2**

  2) Do delayed update of rest of matrix using matrix-matrix multiplication

  A(end+1:n, end+1:n) = A(end+1:n, end+1:n) -A(end+1:n, ib:end)*A(ib:end, end+1:n)

  A(5:6, 5:6) = A(5:6, 5:6) -A(5:6, 3:4) * A(3:4, 5 : 6)

-3	2	-	1	-2	*	-2	-1	=	1	-1
-3	2		-2	0		1	-2		1	0



Example (8)

- **For bk = 2, ib = 5, n = 6, end = 6**

  1) For i = ib to end (i = 5,6)

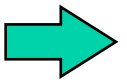
  i= 1 a) A(i+1: n,i) = A(i+1: n, i) / A(i,i) => A(6, 5)=A(6, 5) / A(5,5)

  b) only update columns i+1 (2) to end (2) , so only column 2

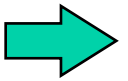
  A(i+1:n, i+1:end) = A(i+1:n, i+1:end) - A(i+1:n, i) * A(i, i+1:end)

  A(6, 6) = A(6, 6) - A(6, 5) * A(5, 6)= 1 => DONE

2	0	-1	1	0	2
4	1	-3	4	1	4
0	-1	2	-1	-3	-1
2	-1	0	0	0	0
-2	0	2	-2	-3	0
-2	0	-1	-3	-3	0



2	0	-1	1	0	2
2	1	-1	2	1	0
0	-1	1	1	-2	-1
1	-1	0	1	1	-2
-1	0	1	-1	-3	2
-1	0	-2	-2	-3	2



2	0	-1	1	0	2
2	1	-1	2	1	0
0	-1	1	1	-2	-1
1	-1	0	1	1	-2
-1	0	1	-2		
-1	0	-2	0		

Example (9)

A = L * U

2	0	-1	1	0	2
4	1	-3	4	1	4
0	-1	2	-1	-3	-1
2	-1	0	0	0	0
-2	0	2	-2	-3	0
-2	0	-1	-3	-3	0

=

1	0	0	0	0	0
2	1	0	0	0	0
0	-1	1	0	0	0
1	-1	0	1	0	0
-1	0	1	-2	1	0
-1	0	-2	0	1	1

*

2	0	-1	1	0	2
0	1	-1	2	1	0
0	0	1	1	-2	-1
0	0	0	1	1	-2
0	0	0	0	1	-1
0	0	0	0	0	1

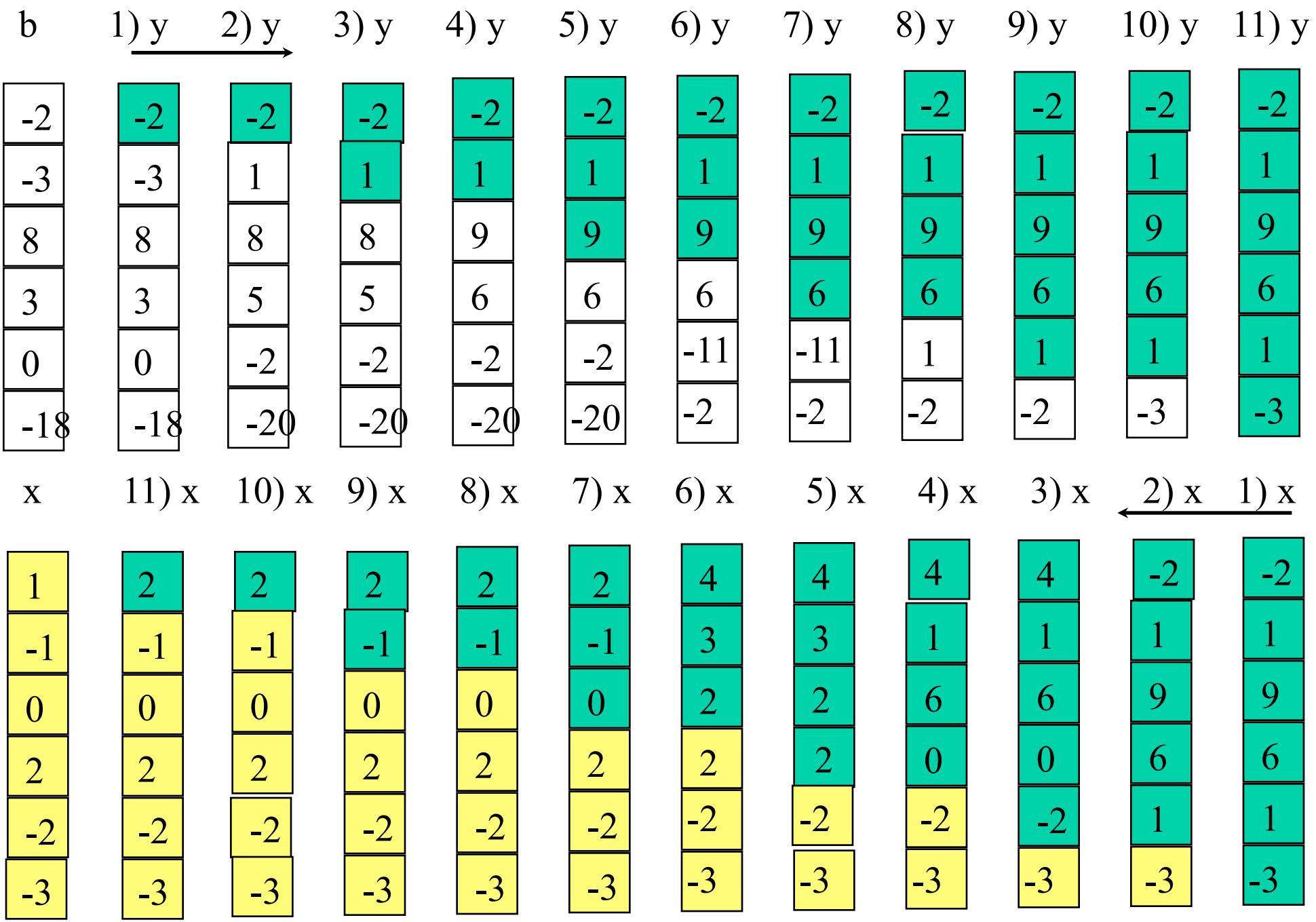
Solve Ax = b

=> L U x = b

=> L y = b , U x = y

```
For  j =  1 to n
y(j) = b(j)
for j = 1 to n-1
y(j) = y(j) / L(j,j)
    for i = j+1 to n
    y(i) = y(i) - y(j)*L(i,j)
```

Triangular Solve



LAPACK GE Solver

- **Driver subroutine to compute the solution of a real system of linear equations, Ax=b**
- **SUBROUTINE DGESV(N, NRHS, A, LDA, IPIV, B, LDB, INFO)**
  - **N : The order of the matrix A**
  - **NRHS : The number of right hand side, the number of columns of b**
  - **A : matrix A, dimension (LDA, N), on entry, the NxN coefficient matrix A, on exit, the factors L and U from factorization**
  - **LDA : The leading dimension of the array A**
  - **IPIV : The pivot indices that define the permutation matrix P**
  - **B : On entry, the right hand side of b, on exit, the solution x**
  - **LDB : The leading dimension of the array b,**
  - **INFO : output info, 0 = successful exit**
- **The DGESV subroutine calls the DGETRF subroutine which does the LU factorization and the DGETRS which solves the triangular systems.**

JICS
Joint Institute for Computational Science

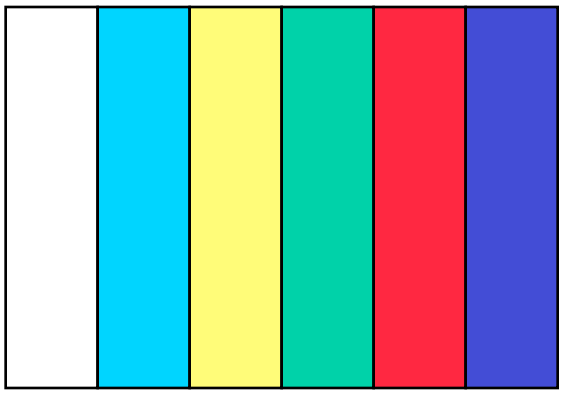
Scalable Linear Algebra Package (ScaLAPACK)
www.netlib.org/scalapack

Data Layout

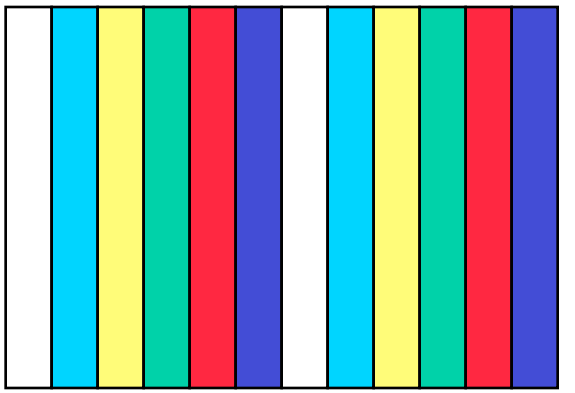
- ScaLAPACK is an extension of the LAPACK subroutines to perform on distributed memory parallel computers or a network of workstations running PVM or/and MPI.
- Data layout of matrices on processors will strongly affect the performance of an algorithm. There are primarily four ways to partition a matrix
- Row-wise block or column-wise block partitioning
- Row-wise block cyclic or column block cyclic partitioning
- 2D block block partitioning
- 2D block cyclic partitioning

Data Distribution

- **Column Blocked Layout:**
  - **In this layout, a block of columns of matrix A is stored per processor as shown below.**
  - **This layout has the same disadvantage of the row-wise stripe partition because as soon as the first few columns have completed the elimination, the processors storing those columns remain idle for the rest of the elimination process.**
- **Column Block Cyclic Layout :**
  - **This layout tries to address the problem of load balancing by assigning blocks of columns of matrix A to processors in a cyclic fashion. However, this layout has the disadvantage that the factorization of A(ib:n, ib:end) will take place perhaps in just one processor. This would be a serial bottleneck**



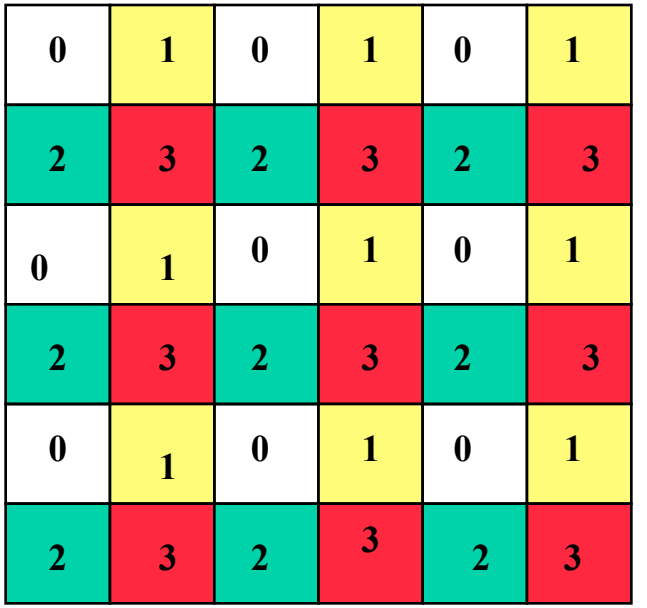
Column block Layout



Column Block Cyclic layout

2D Block Cyclic Layout

- **The Row and Column (2D) Block Cyclic Layout will be a good compromise between the Block and Cyclic Layouts. It will alleviate the problem of load balancing and avoid the situation of a serial bottleneck. Two dimensional block structures allows efficient implementation of BLAS3 update of A(ib;end , end+1:n)**



2D Block Cyclic Layout

ScaLAPACK
(www.netlib.org/scalapack)

- ScaLAPACK (version 1.7) is an extension of LAPACK using PVM or MPI on parallel computers.
- It chooses 2D block cyclic data distribution to optimize BLAS3 operations.
- It is composed of LAPACK, BLAS, PBLAS, and BLACS.
- The BLACS, Blasic Linear Algebra Communication Subprograms, are a message passing library designed for linear algebra.
- PBLAS is a set of parallel basic linear algebra subroutines similar to BLAS.
- There are four basic steps to call a ScaLAPACK routine.
  - **Initialize the process grid (BLACS)**
  - **Distribute the matrix on the process grid (DESCINIT)**
  - **Call ScaLAPACK driver routine**
  - **Release the process grid (BLACS)**

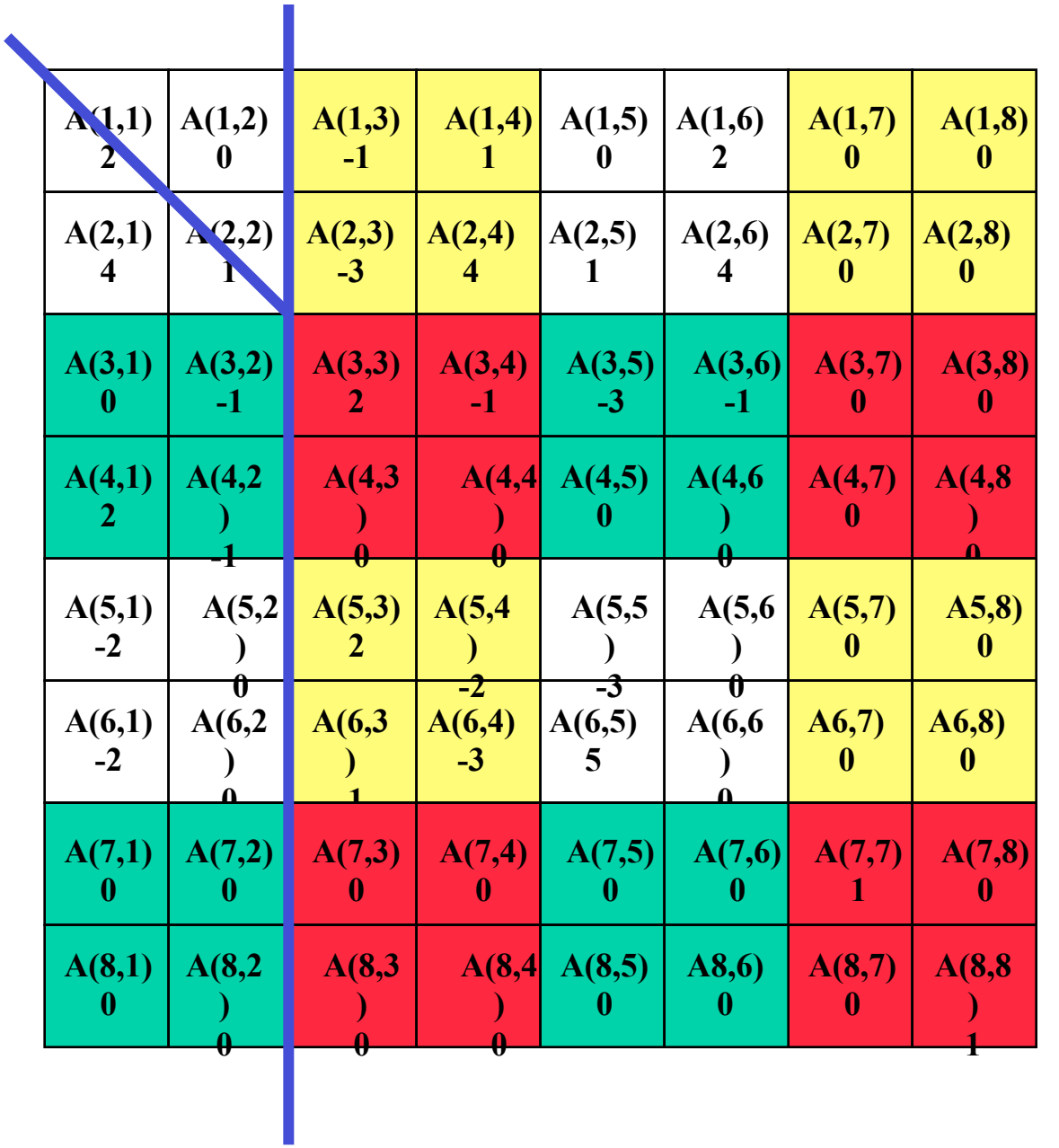
Solve a System of Equations

- **General matrix factorization**
  - **call PDGETRF(M, N, A, IA, JA, DESC_A, IPVT, INFO)**
- **General matrix solve**
  - **call PDGETRS(TRANSA, N, NRHS, A, IA, JA, DESC_A, IPVT, B, IB, JB, ESC_B, INFO)**

$$
\begin{aligned}
2x_1 - x_3 + x_4 + 2x_6 &= -2 \\
4x_1 + x_2 - 3x_3 + 4x_4 + x_5 + 4x_6 &= -3 \\
-x_2 + 2x_3 - x_4 - 3x_5 - x_6 &= 8 \\
2x_1 - x_2 &= 3 \\
-2x_1 + 2x_3 - 2x_4 - 3x_5 &= 0 \\
-2x_1 - x_3 - 3x_4 + 5x_5 &= -18 \\
x_7 &= 1 \\
x_8 &= 1
\end{aligned}
$$

2D Block Cyclic Distribution

- Consider an 8 x 8 system of linear equations using a 2D blocked cyclic data distribution
- Matrix A is first decomposed into 2x2 blocks starting at its upper left corner, bk=2.
- These blocks are then uniformly distributed across a 2x2 processor grid, nprow = npcol =2.
- There are 4 processes in the 2D process grid, nbrow = nbcol = 2.



JICS
Joint Institute for Computational Science

Data Distribution on Local Processors

- The leading dimension of local process grid, LLD, are the same (in this case) and is equal to 4
- The number of rows of matrix A that a process own (in this case) is 4.
- The number of columns of matrix A that a process own is 4.
- Process (0,0) is chosen as the process containing the first matrix entry in its local memory, thus, the process row over which the first row of matrix A is distributed, RSRC=0, and process column over which the first column of matrix A is distributed, CSRC=0

Process grid (0,0)

A(1,1) 2	A(1,2) 0	A(1,5) 0	A(16) 2
A(2,1) 4	A(2,2) 1	A(2,5) 1	A(2,6) 4
A(5,1) -2	A(5,2) 0	A(5,5) -3	A(5,6) 0
A(6,1) -2	A(6,3) 0	A(6,5) 5	A(6,6) -3

Process grid (0,1)

A(1,3) -1	A(1,4) 1	A(1,7) 0	A(1,8) 0
A(2,3) -3	A(2,4) 4	A(2,7) 0	A(2,8) 0
A(5,3) 2	A(5,4) -2	A(5,7) 0	A(5,8) 0
A(6,3) -1	A(6,4) 3	A(6,7) 0	A(5,8) 0

Process grid (1,0)

A(3,1) 0	A(3,2) -1	A(3,5) -3	A(3,6) -1
A(4,1) 2	A(4,2) -1	A(4,5) 0	A(4,6) 0
A(7,1) 0	A(7,2) 0	A(7,5) 0	A(7,6) 0
A(8,1) 0	A(8,2) 0	A(8,5) 0	A(8,6) 0

Process grid (1,1)

A(3,3) 2	A(3,4) -1	A(3,7) 0	A(3,8) 0
A(4,3) 0	A(4,4) 0	A(4,7) 0	A(4,8) 0
A(7,3) 0	A(7,4) 0	A(7,7) 1	A(7,8) 0
A(8,3) 0	A(8,4) 0	A(8,7) 0	A(8,8) 1

ScaLAPACK GE Subroutine

- ScaLAPACK is composed of LAPACK, BLAS, PBLAS, and BLACS.
- The BLACS, Blasic Linear Algebra Communication Subprograms, are a message passing library designed for linear algebra.
- PBLAS is a set of parallel basic linear algebra subroutines similar to BLAS.
- There are four basic steps to call a ScaLAPACK routine.
  - Initialize the process grid
  - Distribute the matrix on the process grid
  - Call ScaLAPACK driver routine
  - Release the process grid
- BLACS routines are used to initialize the process grid
- A ScaLAPACK tools routine, DESCINIT, can be used to distribute the matrix layout (or Iinitializes the Descriptor)
- A ScaLAPACK routine is called to perform a specific task
- A BLACS routine is then used to release the process grid

JICS
Joint Institute for Computational Science

ScaLAPACK GE Algorithm

All processors perform simultaneously

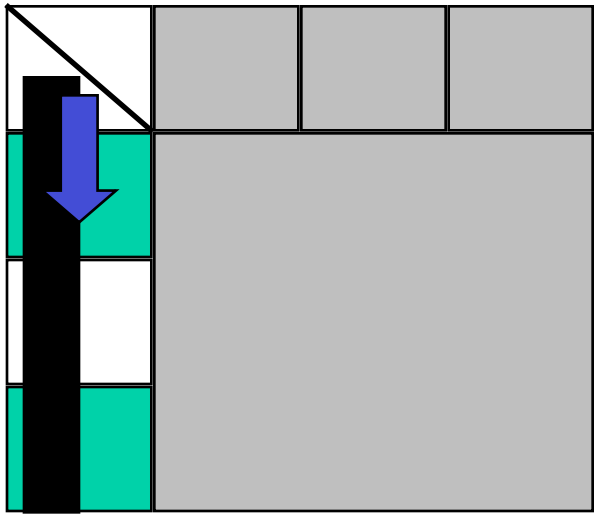
```
For ib = 1 to n-1 step bk
        end = min (ib + bk-1, n)
        For I = ib to end
        (1) Find pivot row k, column broadcast
        (2) Swap rows k and I in block column, broadcast row k
        (3) A(I+1:n, I) = A(I+1:n, I) / A(I, I)
        (4) A(I+1:n, I+1:n) = A(I+1:n, I+1:n) - A(i+1:n, I)*A(I, I+1:end)
        end for
(5) Broadcast all swap information right and left
(6) Apply all rows swaps to other columns
(7) Broadcast LL right
(8) A(ib:end, end+1:n) = LL \ A(ib:end, end+1:n)
(9) Broadcast A(ib:end, end+1:n) down
(10) Broadcast A(end+1:n, ib:end) right
(11) Eliminate A(end+1:n, end+1:n)
end for
```

Distributed GE (1st sweep)

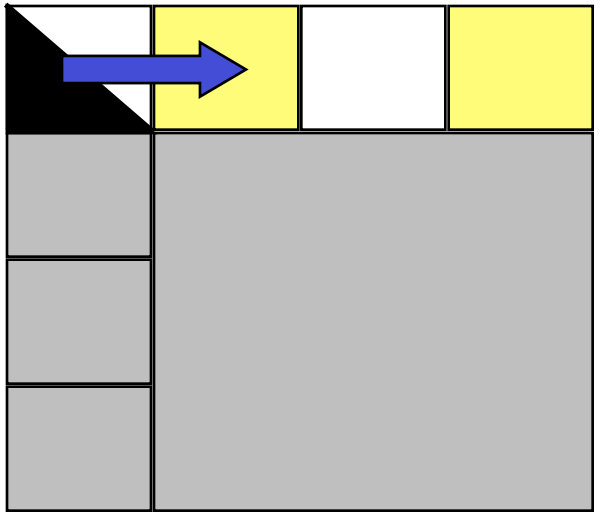
ib = 1, bk=2, end = 2, n=8,

- **Step (1) : pivot is assumed to be the immediate row**
- **Step (2) : i = 1, 2 , Processor 0 to Processor 2**
  - **broadcast the pivot row to all processors in the column**
- **Step (3) : i = 1, 2 , Processor 0 and Processor 2**
  - **A(2:8, 1) = A(2:8, 1) / A(1,1),**
- **Step (4) : i = 1, Processor 0 and Processor 2**
  - **A(2:8, 2) = A(2:8, 2) - A(2:8, 1)*A(1,2)**
- **Step (5) : broadcast all swap information right and left**
- **Step (6) : apply all rows swaps to other columns**
- **Step (7) : Processor 0 to Processor 1**
  - **Broadcast LL right**
- **Step (8) : Processor 0 and Processor 1**
  - **A(1:2, 3:8) = LL \ A(1:2, 3:8)**
- **Step (9) : Processor 0 to Processor 2 , Processor 1 to Processor 3**
  - **Broadcast A(1:2, 3:8) down**
- **Step (10) : Processor 0 to Processor 1, Processor 2 to Processor 3**
  - **Broadcast A(3:8, 1:2) right**
- **Step (11) : Processor 0, Processor 1, Processor 2, and Processor 3**
  - **A(3:8, 3:8) = A(3:8, 3:8) - A(3:8, 1:2) * A(1:2, 3:8)**

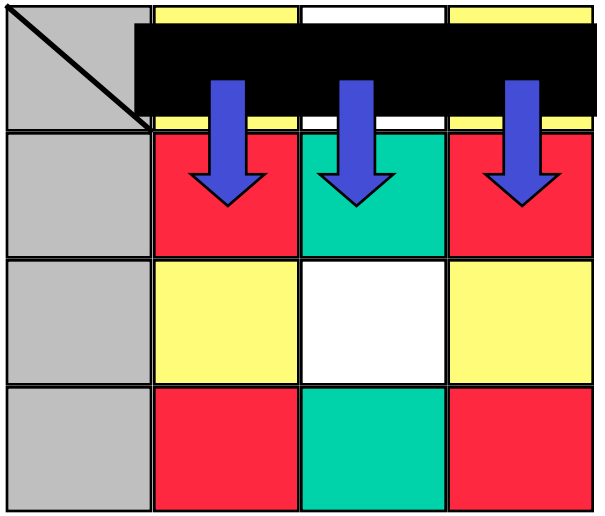
Distributed GE (1st sweep)



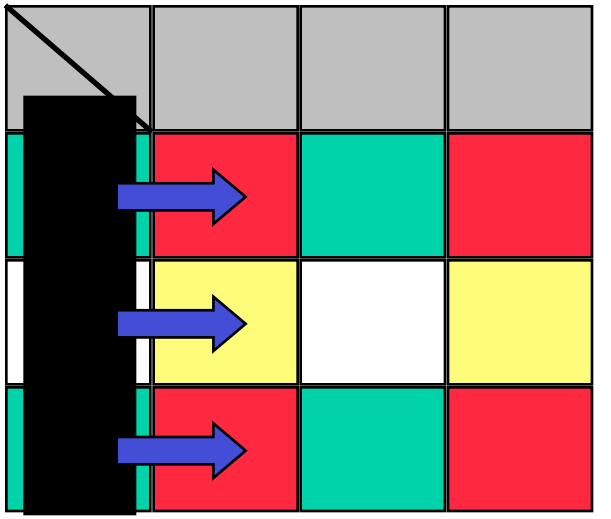
Step (1), (2), (3), (4)



Step (7), (8)



Step (9)



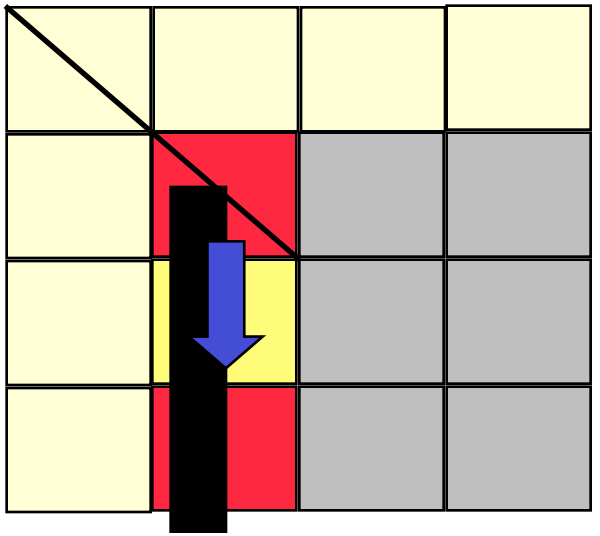
Step (10), (11)

Distributed GE (2nd sweep)

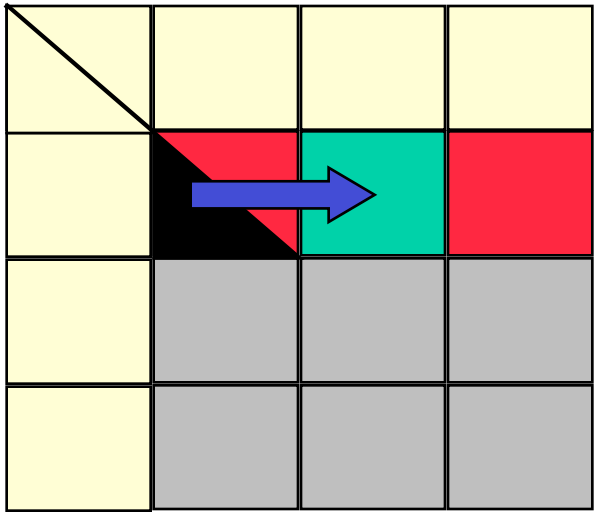
ib = 3, bk=2, end = 4, n=8,

- **Step (1) : pivot is assumed to be the immediate row**
- **Step (2) : i = 3, 4 , Processor 3 to Processor 1**
  - **broadcast the pivot row to all processors in the column,**
- **Step (3) : i = 3, 4 , Processor 1 and Processor 3**
  - **A(4:8, 1) = A(4:8, 1) / A(3,3),**
- **Step (4) : i = 3, Processor 1 and Processor 3**
  - **A(4:8, 4) = A(4:8, 4) - A(4:8, 3)*A(3, 4)**
- **Step (5) : broadcast all swap information right and left**
- **Step (6) : apply all rows swaps to other columns**
- **Step (7) : Processor 3 to Processor 2**
  - **Broadcast LL right**
- **Step (8) : Processor 2 and Processor 3**
  - **A(3:4, 5:8) = LL \ A(3:4, 5:8)**
- **Step (9) : Processor 2 to Processor 0 , Processor 3 to Processor 1**
  - **Broadcast A(3:4, 5:8) down**
- **Step (10) : Processor 1 to Processor 0, Processor 3 to Processor 2**
  - **Broadcast A(5:8, 3:4) right**
- **Step (11) : Processor 0, Processor 1, Processor 2, and Processor 3**
  - **A(5:8, 5:8) = A(5:8, 5:8) - A(5,8, 3:4) * A(3:4, 5:8)**

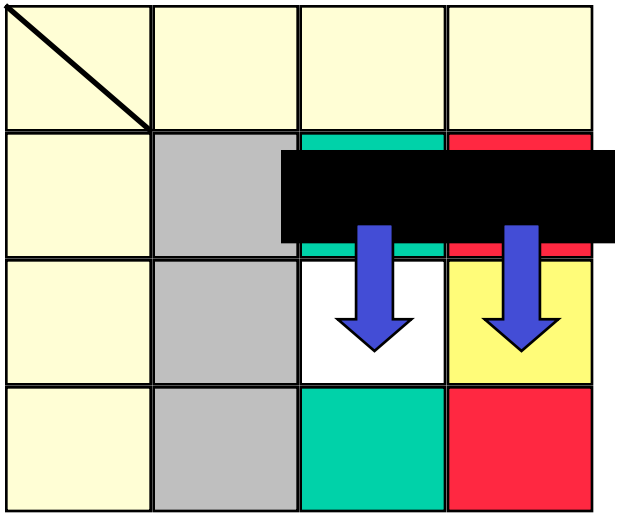
Distributed GE (2nd sweep)



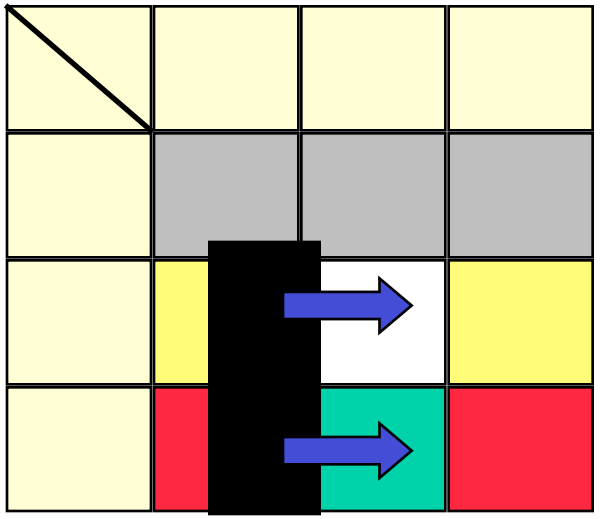
Step (1), (2), (3), (4)



Step (7), (8)



Step (9)



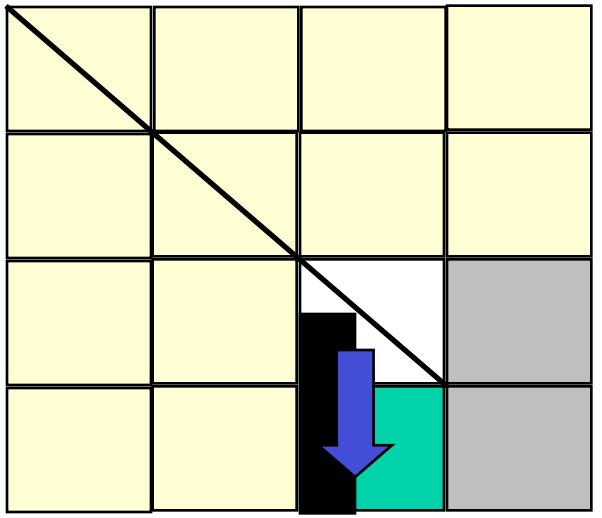
Step (10), (11)

Distributed GE (3rd sweep)

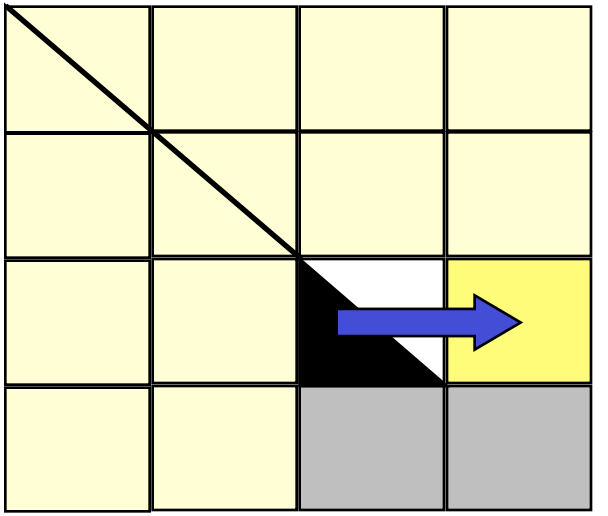
ib = 5, bk=2, end = 6, n=8,

- **Step (1) : pivot is assumed to be the immediate row**
- **Step (2) : i = 5, 6 , Processor 0 to Processor 2**
  - **broadcast the pivot row to all processors in the column, broadcast from**
- **Step (3) : i = 5, 6 , Processor 0 and Processor 2**
  - **A(6:8, 1) = A(6:8, 5) / A(5,5),**
- **Step (4) : i= 5, Processor 0 and Processor 2**
  - **A(6:8, 6) = A(6:8, 6) - A(6:8, 5)*A(5,6)**
- **Step (5) : broadcast all swap information right and left**
- **Step (6) : apply all rows swaps to other columns**
- **Step (7) : Processor 0 to Processor 1**
  - **Broadcast LL right**
- **Step (8) : Processor 0 and Processor 1**
  - **A(5:6, 7:8) = LL \ A(5:6, 7:8)**
- **Step (9) : Processor 1 to Processor 3**
  - **Broadcast A(5:6, 7:8) down**
- **Step (10) : Processor 2 to Processor 3**
  - **Broadcast A(7:8, 5:6) right**
- **Step (11) : Processor 3**
  - **A(7:8, 7:8) = A(7:8, 7:8) - A(7:8, 5:6) * A(5:6, 7:8)**

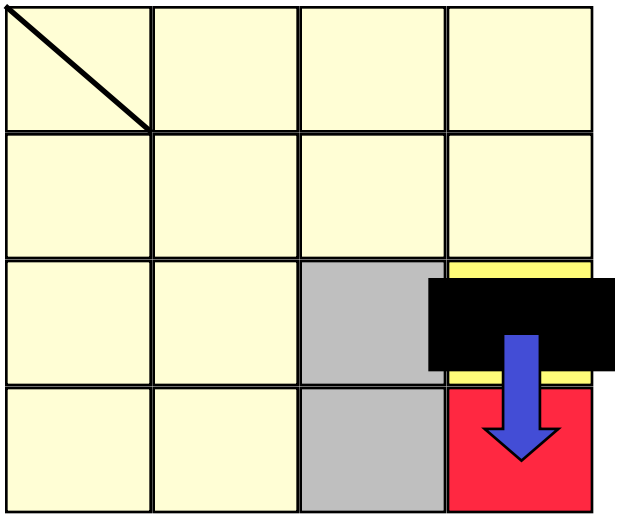
Distributed GE (3rd sweep)



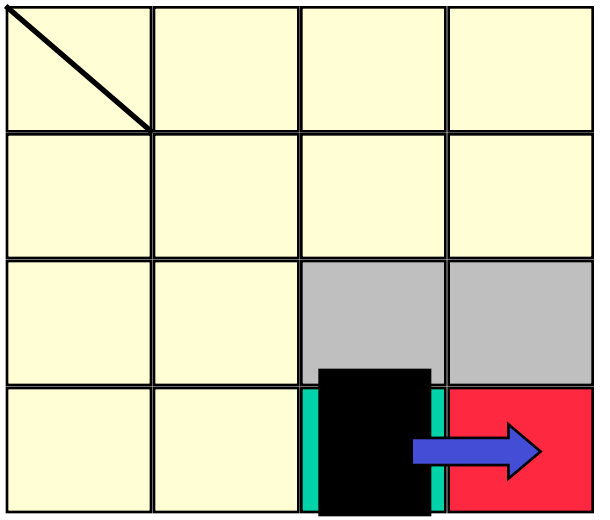
Step (1), (2), (3), (4)



Step (7), (8)



Step (9)



Step (10), (11)

JICS
Joint Institute for Computational Science

ScaLAPACK Linear Solver

Process grid initialization

```
CALL BLACS_PINFO(MYID, NPROCS)
!  Initialize the process grid, obtain system default context
CALL BLACS_GET(-1,0,ICTXT)
!  Map the available processes to a BLACS process grid
CALL BLACS_GRIDINIT(ICTXT,' Row-major' ,NPROW,NPCOL)
!  Query the process grid to identify each process' s coordinate, (MYROW, MYCOL)
CALL BLACS_GRIDINFO(ICTXT,NPROW,NPCOL,MYROW,MYCOL)
```

Data Distribution

- All global matrices must be distributed on the process grid prior
- CALL DESCINIT(DESCA, M, N, MB, NB, RSRC, CSRC, ICTXT, LLDA, INFO)
- CALL DESCINIT(DESCB, N, NRHS, NB, NBRHS, RSRC, CSRC, ICTXT, LLDB, INFO)

Call the solver routine

- CALL PDGESV(N,NRHS,A,IA,JA,DESCA,IPIV,B,IB,JB,DESCB,INFO)
- CALL PDGETRF, CALL PDGETRS

Release the process grid

- CALL BLACS_GRIDEXIT(ICONTXT)
- CALL BLACS_EXIT(0)

JICS
Joint Institute for Computational Science

Setting Up ScaLAPACK Linear Solver

- A number of BLACS routines are needed to initialize the process grid.
- BLACS_PINFO(MYPNUM, NPROCS)
  - Query the number of processes, NPROCS: the number of processes and process identifier, MYPNUM
- BLACS_SETUP(MYPNUM, NPROCS)
  - Same as BLACS_PINFO, only needed for PVM BLACS
- BLACS_GET(ICONTXT, WHAT, VAL)
  - Get default system context (system ID for library reference)
- BLACS_GRIDINIT(ICONTEXT, ORDER, NPROW, NPCOL)
  - This routine assigns the available processes to a BLACS process grid
  - This routine creates a simple NPROW x NPCOL process grid. This process grid will use the first NPROW * NPCOL processes, and assign them to the grid in a row- or column natural ordering depending the input value ORDER.
- BLACS_GRIDINFO(ICONTXT, NPROW, NPCOL, MYPROW, MYPCOL)
  - Query the process grid to identify each process's coordinates
  - MYROW: the calling process's row coordinate in the process grid
  - MYCOL : the calling process's column coordinate in the process grid

ScaLAPACK Linear Solver

```
!   Initialize the process grid, query for the number of processes allocated
    CALL BLACS_PINFO(MYID, NPROCS)
    IF(NPROCS  .LT. 1) THEN
          NPROCS=NPROCS_WANTED
          CALL BLACS_SETUP(MYID, NPROCS)
    END IF
!   Initialize the process grid, obtain system default context
    CALL BLACS_GET(-1,0,ICTXT)
!   Map the available processes to a BLACS process grid
    CALL BLACS_GRIDINIT(ICTXT,' Row' ,NPROW,NPCOL)
!   Query the process grid to identify each process' s coordinate, (MYROW,
    MYCOL)
    CALL BLACS_GRIDINFO(ICTXT,NPROW,NPCOL,MYROW,MYCOL)
```

ScaLAPACK Linear Solver

Data Distribution

- All global matrices must be distributed on the process grid prior to the invocation of a ScaLAPACK routine; typical initalization routines are
- CALL DESCINIT(DESCA, M, N, MB, NB, RSRC, CSRC, ICTXT, LLDA, INFO)
- CALL DESCINIT(DESCB, N, NRHS, NB, NBRHS, RSRC, CSRC, ICTXT, LLDB, INFO)

Call the ScaLAPACK routine

- All ScaLAPACK routines assume that the data has been distributed on the process grid prior to the invocation of the routine, typically, solve the linear system AX=B
- CALL PDGESV(N,NRHS,A,IA,JA,DESCA,IPIV,B,IB,JB,DESCB,INFO)

Release the process grid

- Free the resources associated with a particular context
- CALL BLACS_GRIDEXIT(ICONTXT)
- Exit BLACS
- CALL BLACS_EXIT(0)

JICS
Joint Institute for Computational Science

ScaLAPACK Linear Solver

Process grid initialization

```
CALL BLACS_PINFO(MYID, NPROCS)
!  Initialize the process grid, obtain system default context
CALL BLACS_GET(-1,0,ICTXT)
!  Map the available processes to a BLACS process grid
CALL BLACS_GRIDINIT(ICTXT,' Row-major' ,NPROW,NPCOL)
!  Query the process grid to identify each process' s coordinate, (MYROW, MYCOL)
CALL BLACS_GRIDINFO(ICTXT,NPROW,NPCOL,MYROW,MYCOL)
```

Data Distribution

- All global matrices must be distributed on the process grid prior
- CALL DESCINIT(DESCA, M, N, MB, NB, RSRC, CSRC, ICTXT, LLDA, INFO)
- CALL DESCINIT(DESCB, N, NRHS, NB, NBRHS, RSRC, CSRC, ICTXT, LLDB, INFO)

Call the solver routine

- CALL PDGESV(N,NRHS,A,IA,JA,DESCA,IPIV,B,IB,JB,DESCB,INFO)
- CALL PDGETRF, CALL PDGETRS

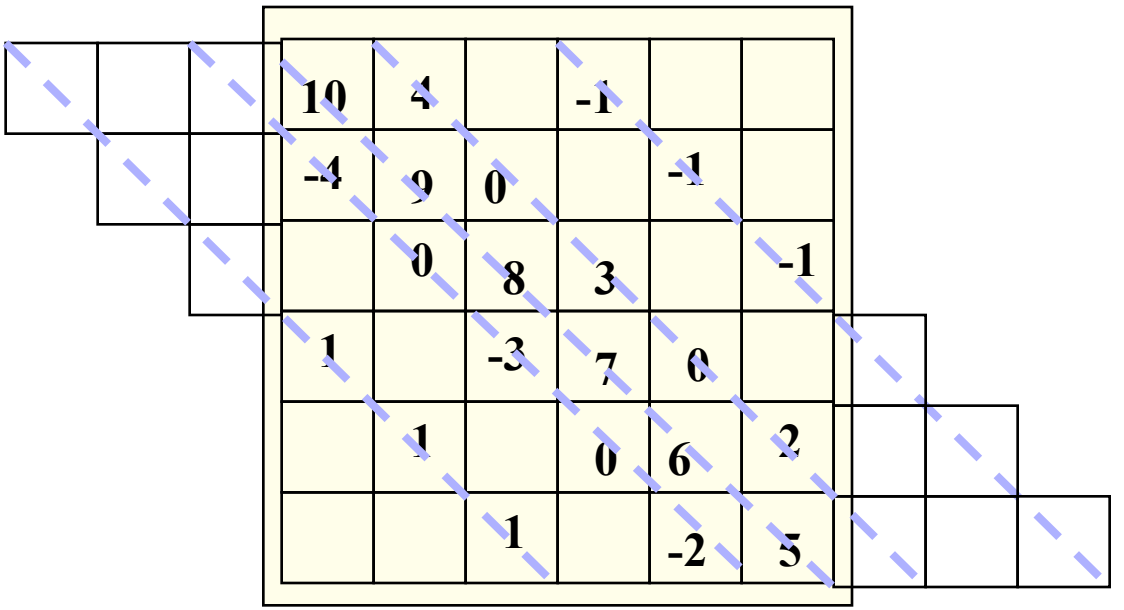
Release the process grid

- CALL BLACS_GRIDEXIT(ICONTXT)
- CALL BLACS_EXIT(0)

Libraries for Sparse Matrices

Storage Schemes of Sparse Matrix

- **There are a lot of different sparse matrix storage schemes. We will introduce a few common types which can be used for general sparse matrix. Sparse storage generally consists of several vectors which stores the nonzero values of the matrix and pointers of location of the nonzero values. Obviously, the most logical and efficient storage scheme for this block tridiagonal matrix will be the** Diagonal Storage scheme. **The scheme stores the values of the matrix using individual vector array for each diagonal and a position pointer relative to the main-diagonal of the matrix.**



aval(:,1)=(0,0,0,1,1,1) , apos(1)=-3
aval(:,2)=(0,-4,0,-3,0,-2), apos(2)=-1
aval(:,3)=(10,9,8,7,6,5), apos(3)=0
aval(:,4)=(4,0,3,0,2,0), apos(4)=1
aval(:,5)=(-1,-1,-1,0,0,0), apos(5)=3

Matrix vector product:

```
do I=1,N
    do k=1,5
    w(I) = w(I) + aval(I,k) * p(I-apos(k))
    enddo
enddo
```

Coordinate Storage Scheme

- **The Coordinate Storage scheme consists of three vector arrays, one stores the nonzero values, one stores the row locations of the nonzero entries, and the last one stores the the column locations of the nonzero entries. The order of storing the nonzero entries can be arbitrary, however, rowwise or columnwise storing orders are used for computing efficiency. As can be observed later, storage of one of the location pointer can be reduced.**

10	4		-1		
-4	9	0		-1	
	0	8	3		-1
1		-3	7	0	
	1		0	6	2
		1		-2	5

Coordinate Storage Scheme:
aval(I) = (10,4,-1,-4,9,-1,8,3,-1,1,-3,7,1,6,2,1,-2,5)
irow(I) = (1,1,1,2,2,2,3,3,3,4,4,4,5,5,5,6,6,6)
jcol(I) = (1,2,4,1,2,5,3,4,6,1,3,4,2,5,6,3,5,6)

Matrix vector multiplication

```
do i=1,N
    w(irow(i))=w(irow(i)+aval(i)*p(icol(I))
end do
```

Compressed Row and Column Storage Schemes

- The Compressed Row Storage (CRS) scheme put the subsequent non-zeros of the matrix row in contiguous memory locations. Three vectors are used. One contains the values of the nonzero entries (aval), one stores the column number of each nonzero entries (icol), and the last one stores the pointers to the first entry of the ith row in aval and icol (jprow)
- The Compressed column Row Storage (CCS) scheme is identical to CRS scheme except the matrix nonzero entries are stored in columnwise fashion.
- Due the structural symmetry of the following example, the position indicators of the CRS and CCS are the same!

10	4		-1		
-4	9	0		-1	
	0	8	3		-1
1		-3	7	0	
	1		0	6	2
		1		-2	5

Compressed Row Storage:
aval(I) = (10,4,-1,-4,9,-1,8,3,-1,1,-3,7,1,6,2,1,-2,5)
icol(I) = (1,2,4,1,2,5,3,4,6,1,3,4,2,5,6,3,5,6)
jprow(I) = (1,4,7,10,13,16,19)

Compressed Column Storage:
aval(I) = (10,-4,1,4,9,1,8,-3,1,-1,3,7,-1,6,-2,-1,2,5)
jrow(I) = (1,2,4,1,2,5,3,4,6,1,3,4,2,5,6,3,5,6)
ipcol(I) = (1,4,7,10,13,16,19))

Matrix Vector Product for CRS and CCS

- Compressed Row Storage : w=A*p

```
do I=1,NROW
    w(I)=0
    do j = jprow(I), jprow(I+1) -1
            w(I) = w(I) + aval(j) * p(icol(j))
    end do
end do
```

- Compressed Row Storage

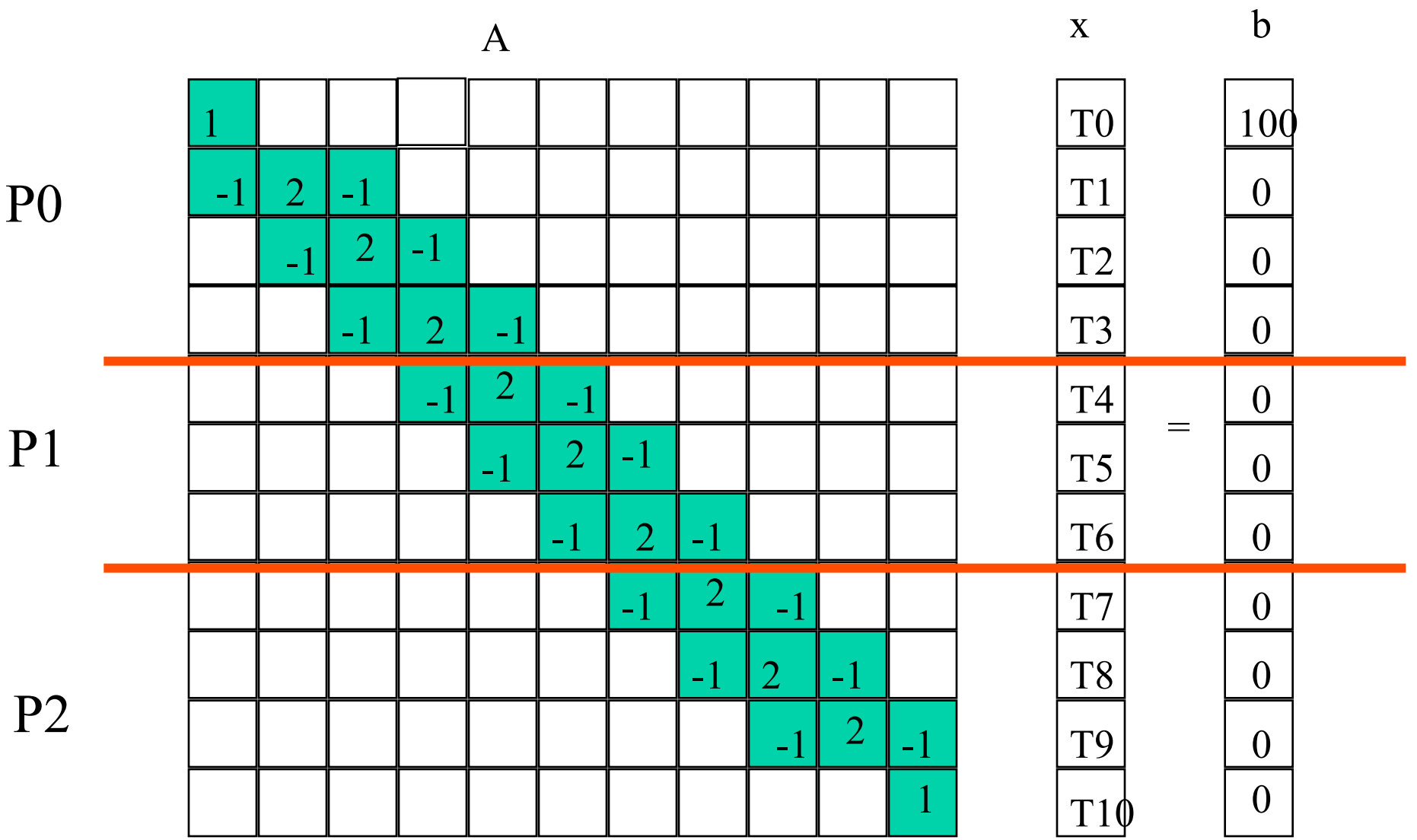
```
Do I=1,NROW
    w(I)=0
end do
do I=1,NCOLUMN
    do j = ipcol(I), ipcol(I+1) -1
            w(jrow(j)) = w(jrow(j)) + aval(j) * p(I)
    end do
end do
```

Resultant Matrix

$$\frac{d^2t}{dx^2} = 0$$

FEM/FD

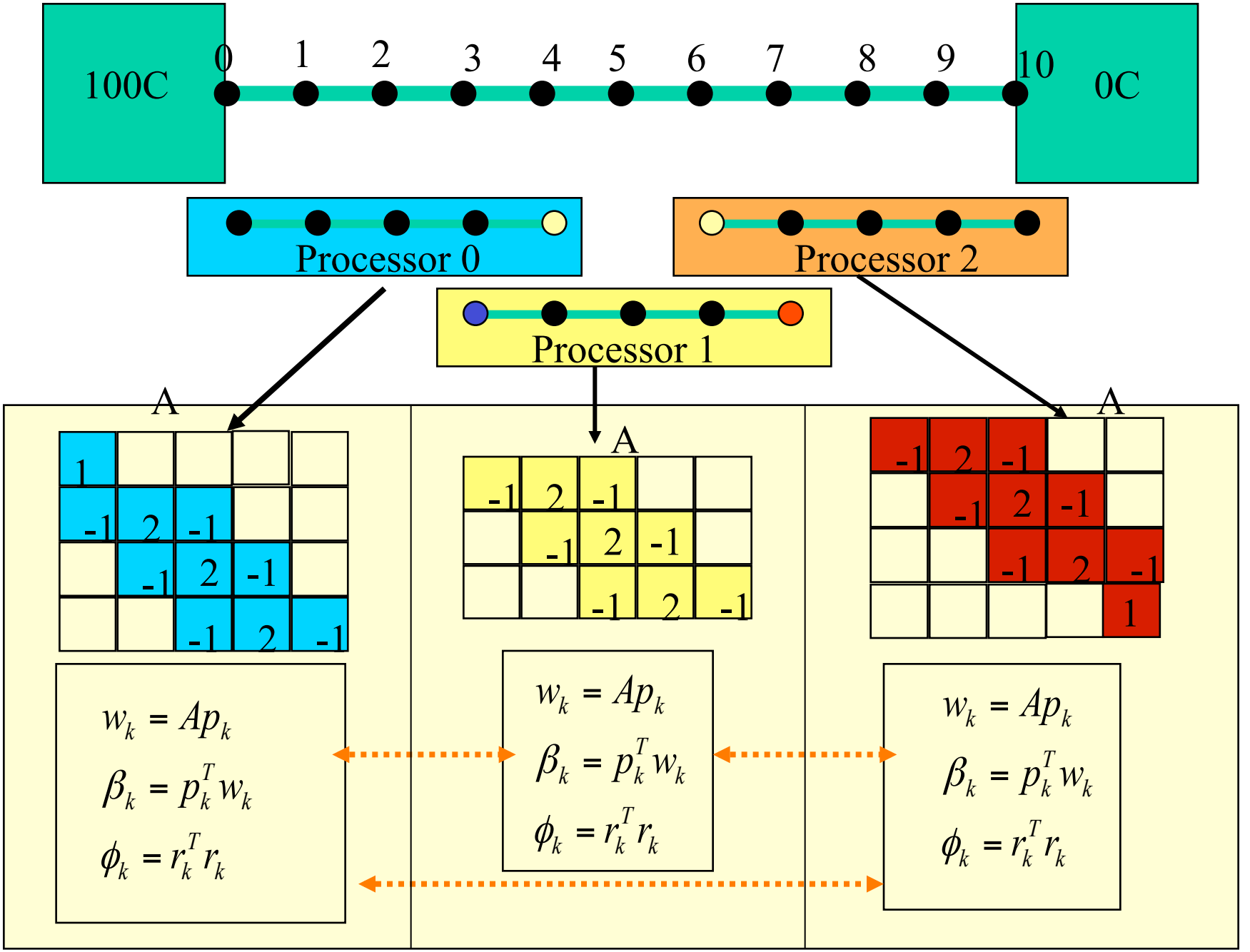
Solve A x = b



CG Algorithm

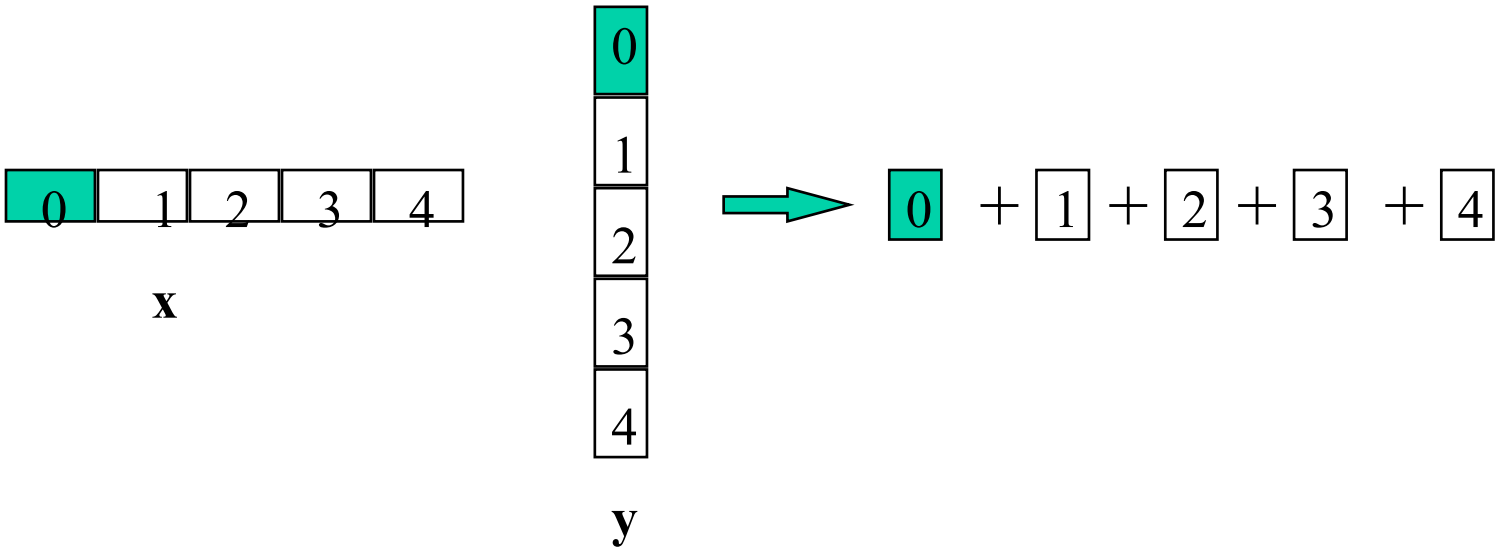
```
i     = 0
x(0)  = 0
r(0)  = b - A*x(0) = b
ϕ(0)  = r^T(0)r(0)
while (f(i) > tolerance) and (i < maximum iteration)
do
        if (i = 0) then p(1) = r(0)
        else p(i+1) = r(i) + ϕ(i)*p(i) / ϕ(i-1)
        i     = i + 1
        - matrix-vector multiplication
        w(i) = A*p(i)
        - vector dot product
        α(i) = ϕ(i-1) / p^T(i)*w(i)
        x(i) = x(i-1) + α(i)*p(i)
        r(i) = r(i-1) - α(i)*w(i)
        - vector dot product
        ϕ(i) = r^T(i)*r(i)
end while
x = x(i)
```

CG in Parallel



Parallel Kennel - Inner Product

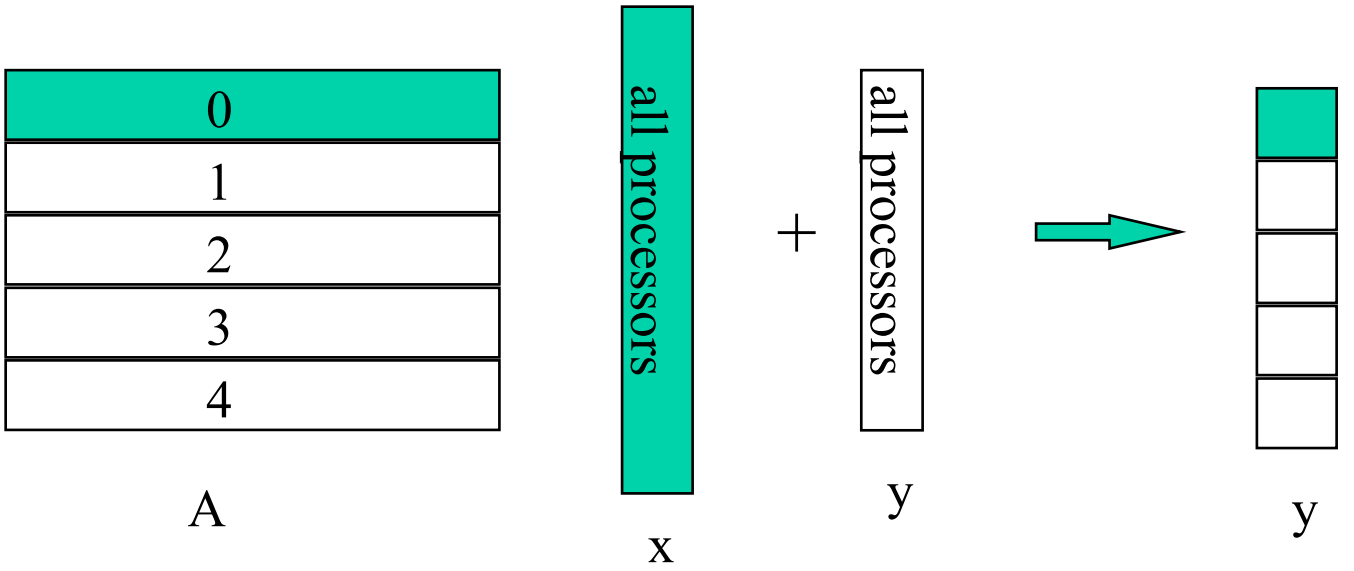
- **Compute the inner product of two vectors,** $sdot = \sum_{i=1}^{n} x^T y$



- Blockwise distribution of the vectors is used. Each processor computes a portion of the value of the inner product. The result is obtained by summing the partial values together.

Parallel Kennel - Matrix Vector Product

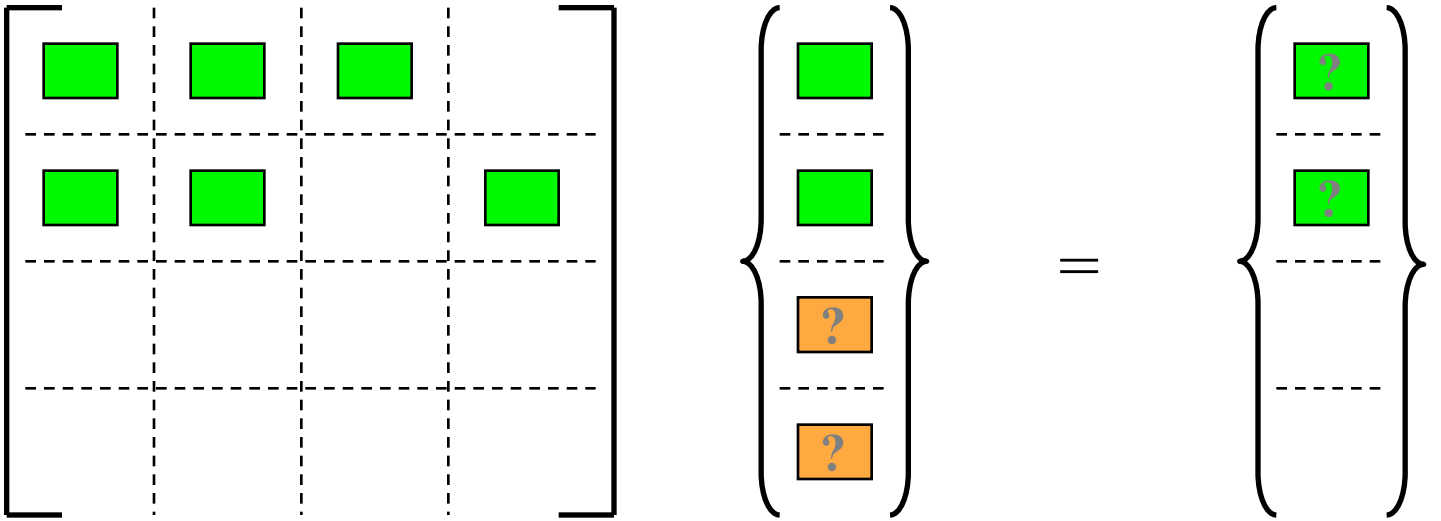
- **Compute the product of a matrix, A, and a vector, x, y=y+Ax**



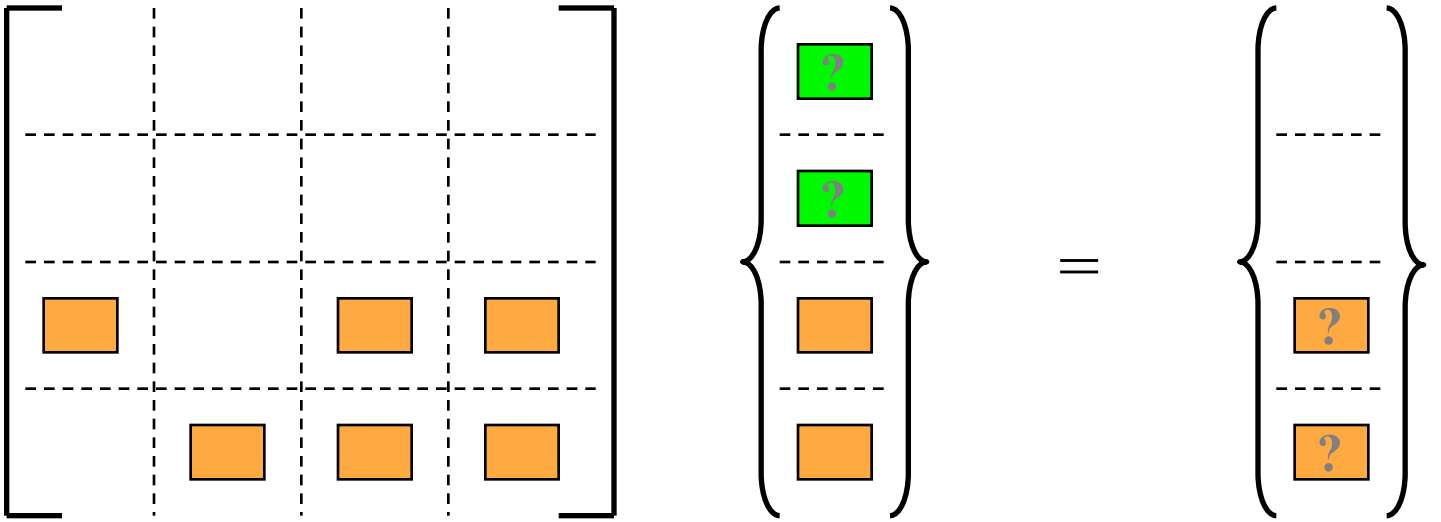
- Matrix A can be distributed to processors with a block-striped partition scheme or a block cyclic scheme. Entire vector x will be needed for calculation of y in every processor. Vector x may be distributed or duplicated among processors. Each processor compute the value of a portion of the matrix vector product The resultant y vector is obtained by summing the partial values.

Parallel M-V Multiplication

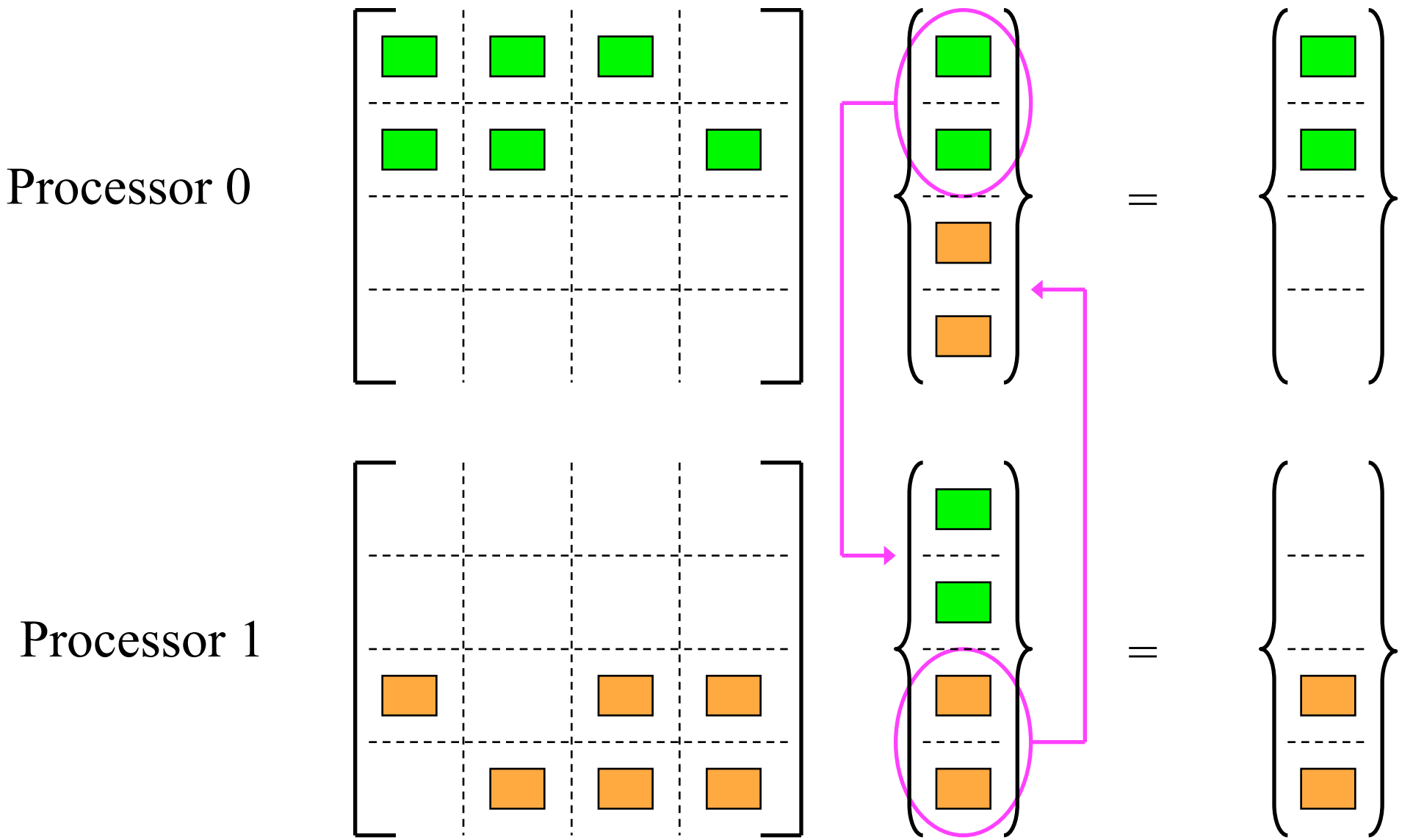
Processor 0



Processor 1



Parallel M-V Multiplication



PINEAPL & Aztec

- Parallel Industrial NumErical Applications and Portable Libraries
- Coordinated efforts by NAG with a group of 8 institutions including British Aerospace, CERFACS, Manchester University, Piaggio, Thomson LCR, Danish Hydraulic Institute, …
- Final version is incorporated as Chapter 11 (F11xxx) of NAG Math Library.
- Aztec is a 'limited freeware' from Sandia National Laboratory - http://www.cs.sandia.gov/CRF/aztec1.html
- Krylov Subspace Solvers :
  - **Conjugate Gradient (CG)**
  - **Generalised Minimal Residual (GMRES)**
  - **Conjugate Gradient Square (CGS)**
  - **Bi-Conjugate Gradient Stabilized (Bi-CGSTAB)**
- Preconditioners:
  - **Additive Schwarz**
  - **Jacobi, SOR, SSOR**
  - **Blocked ILU**

Aztec Data Storage Schemes – Distributed Modified Sparse Row (DMSR)

Prcoess 0 –
nupdate = 3
Iupdate(i)= {2, 3, 4}
Process 1 –
nupdate = 2
Iupdate(i)= { 0, 1}

$$\left[\begin{array}{ccccc} 11 & 12 & 0 & 0 & 15 \\ 21 & 22 & 0 & 0 & 25 \\ \hline 0 & 0 & 33 & 0 & 35 \\ 0 & 0 & 0 & 44 & 0 \\ 51 & 52 & 53 & 0 & 55 \end{array}\right]$$

processor 1

processor 0

ibindx(0) = nupdate + 1

ibindx(i+1) – ibindx(i) = number of nonzero non-diagonal entries in rows, i = 0,..,nupdate-1

ibindx(i) = column indices of iupdate(i), i=nupdate+1

processor 0

I:	0	1	2	3	4	5	6	7
BINDX(I):	4	5	5	8	4	0	1	2
GSM(I):	33	44	55	0	35	51	52	53

processor 1

I:	0	1	2	3	4	5	6
BINDX(I):	3	5	7	1	4	0	4
GSM(I):	11	22	0	12	15	21	25

Aztec Example

```
 /*-------------- setup grid parameters ----------------*/
AZ_processor_info(proc_config);
/*------------------- setup data -------------------------*/
AZ_transform(proc_config, &external, ibindx_jacb, smata_jacb,  iupdate_locl,
     &update_index, &extern_index, &data_org,  nupdate, NULL, NULL, NULL, NULL,
     AZ_MSR_MATRIX);
/* ------------------- setup options --------------------*/
 AZ_defaults(options,params);
options[AZ_solver] = AZ_cg;
params[AZ_tol] = 0.0000000000001;
/* ------------------- solve ---------------------------------*/
AZ_solve(fsysdq, b, options, params, NULL, ibindx_jacb,
        NULL, NULL, NULL, smata_jacb, data_org, status, proc_config);
```

```
CC_SP       = mpcc
FC_SP       = mpxlf
CLIBS_SP   = -lm -lxlf90 /sphome/klwong/Aztec/lib/libaztec.a
CFLAGS_SP =  –O4 -qstrict -bmaxdata=256000000

% poe exe -procs 3 -infolevel 3  ( llsubmit solve.cmd)
```

JICS
Joint Institute for Computational Science

The Portable, Extensible, Toolkit for Scientific Computation (PETSc)

Http://www.mcs.anl.gov/petsc/petsc.html

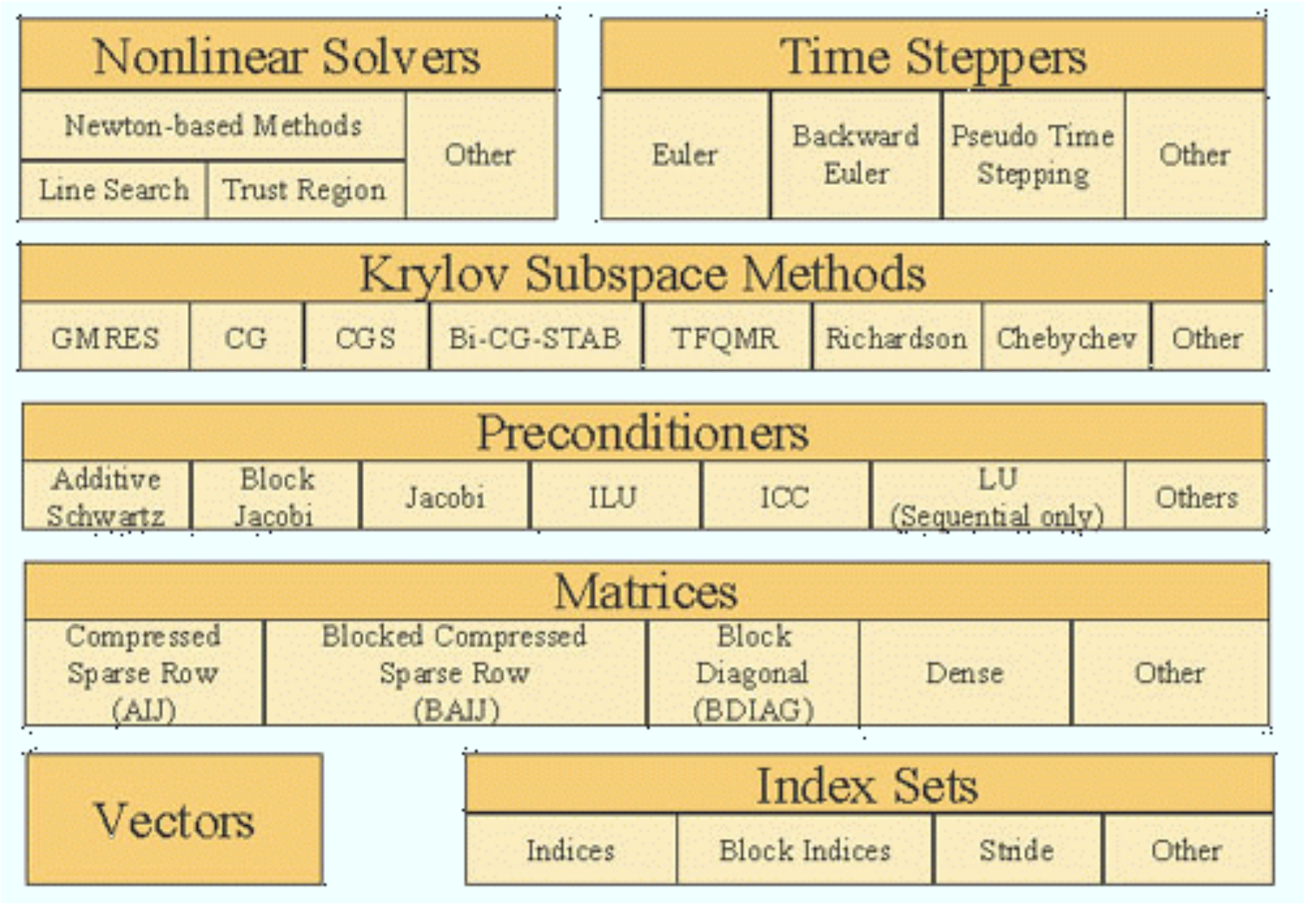
PETSc
(www.mcs.anl.gov/petsc)

- PETSc is a suite of data structures and routines that provide the building blocks for the solution of large-scale application codes on parallel and serial computers
- PETSc includes an expanding set of parallel linear and nonlinear equation solvers with support routines for numerical solutions of partial differential equations on distributed memory machines, clusters of workstations, and non-uniform memory access shared-memory machines.
- It uses the MPI standard for all message passing communication and build on efficient basic linear algebra kernels such as BLAS-type operations. It supports F77, C, C++, and Fortran 90.
- The PETSc distribution contains all source code, installation instructions, a users guide, and a collection of examples.
- PETSc is developed and supported by Argonne National Laboratory.

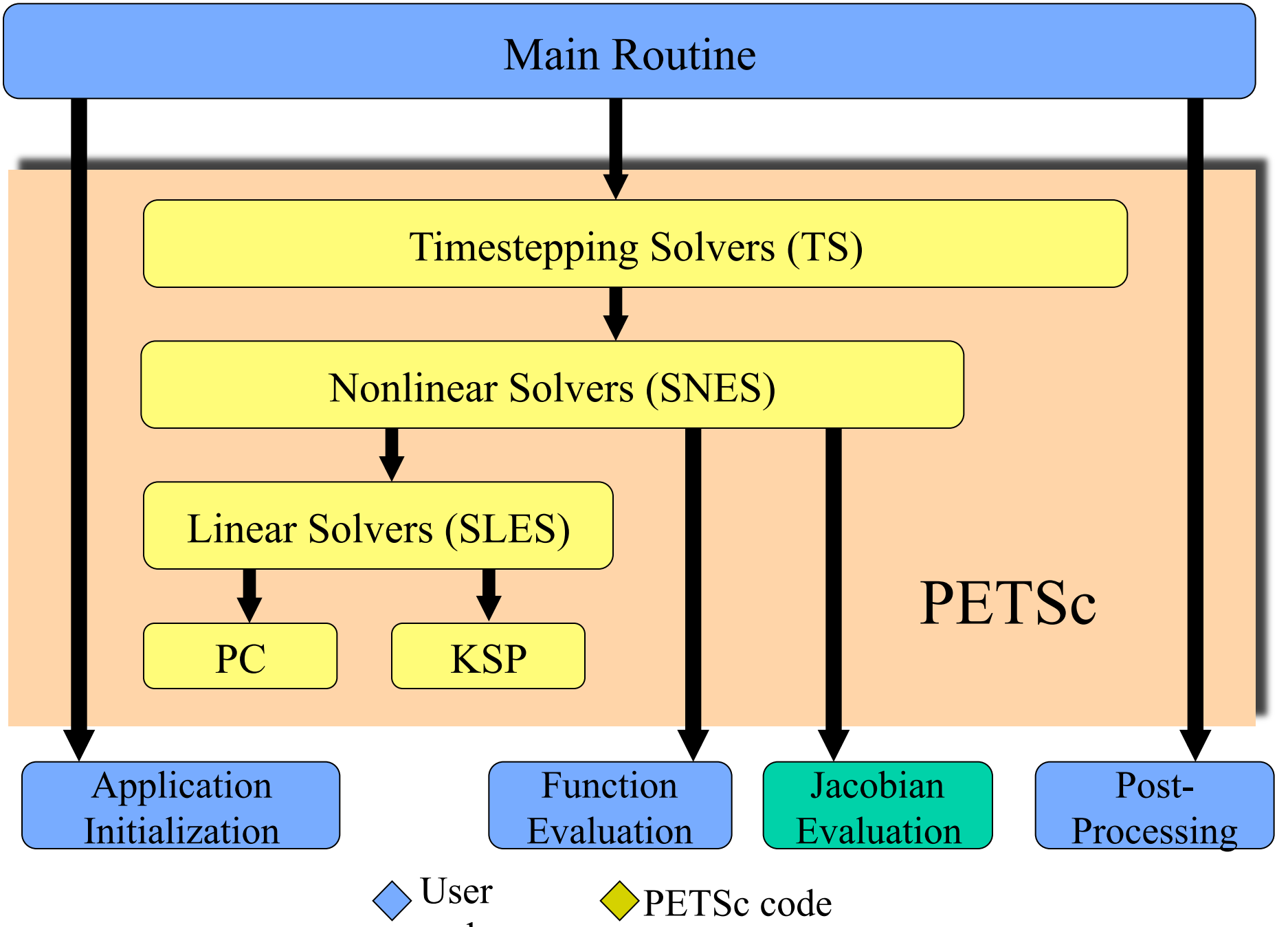
Major Components of PETSc

- **PETSC uses a set of hierarchical tools (mathematical objects) to construct a solution procedure in the PDE problem solving environment**
  - **Vectors (VEC)**
  - **Index sets (IS)**
  - **Distributed Arrays (DA)**
  - **Matrices (MAT)**
  - **Krylov Subspace Solvers (KSP)**
  - **Preconditioners (PC)**
  - **Linear system solvers (SLES)**
  - **Non-linear system solvers (SNES)**
  - **Time-stepping methods (TS)**
  - **Graphics devices**

Petsc Components



User Code/PETSc Library Interactions



References - Books

- **BOOKS (in noparticular order):**
- **MATRIX Computations, G. Golub & C. Van Loan, 2nd Edition, John Hopkins University Press, 1989/93**
- **Introduction to Parallel Computing, Design and Analysis of Algorithms, by Kumar, Grama, Gupta and Karypis, Benjamin/Cummings Publ. Co., 1994**
- **Numerical Analysis, 3rd Edition, by R, Burden and J.D. Faires, PWS Publishers, 1985**
- **Numerical Recipes in C/FORTRAN; The Art of Scientific Computing, 2nd Edition, by Press Teukolsky, Vetterling and Flannery, Cambridge University Press, 1992/94**
- **A Scientist's and Engineer's Guide to Workstations and Supercomputers; Coping with Unix, RISC, vectors and programming, by R. Landau and P. Fink, Jr., John Wiley 1993**

References - WWW URLs

- MAUI's Performance Tuning for the RS6000 Architecture http://www.mhpcc.edu/training/workshop/html/speed/
- performance_optimization.html http://www.mhpcc.edu/training/workshop/html/performance/ and http://www.mhpcc.edu/doc/perf.html
- HPF Educational Materials http://www.npac.syr.edu/projects/cpsedu/hpfe/
- Field simulations - grid techniques www.npac.syr.edu/projects/cpsedu/hpfe/module7/index.html
- Particle Applications - Pipeline Computing www.npac.syr.edu/projects/cpsedu/hpfe/module5/index.html
- Numerical Methods, Computational Physics, University of Carleton, Ottwa, Canada http://www.physics.carleton.ca/courses
- Boston University Origin 2000 Site http://scv.bu.edu/SCV/Origin2000/
- MPI document - http://www-unix.mcs.anl.gov/mpi/

Reference - WWW URLs

- CS325: High Performance Scientific Computing http://comped1.cas.vanderbilt.edu/cs325/cs325.html
- Computational Science Education Projecthttp://compsci.cas.vanderbilt.edu/csep.html http://csep1.phy.ornl.gov/ode/ode.html http://csep1.phy.ornl.gov/pde/pde.html
- Mathematical Optimization, from the CSEP
- e-book http://csep1.phy.ornl.gov/mo/mo.html
- Designing and Building Parallel Programs, by Ian Foster http://www.mcs.anl.gov:80/dbpp/web-tours/
- Massively Parallel SCF Chemistry Calculationshttp://www.mcs.anl.gov/home/minkoff/CHEM/wagner.html
- High Performance Computing Projects at Liverpool University http://www.liv.ac.uk/HPC/HPCpage.html
- Aztec Document - http://www.cs.sandia.gov/CRF/aztec1.html
- PETSc document - http://www.mcs.anl.gov/petsc

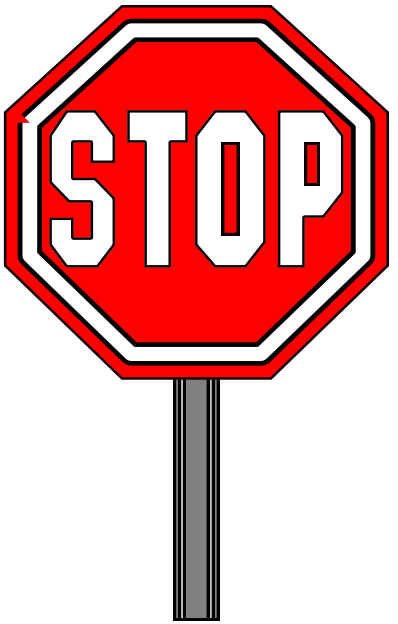
References - WWW URLs

- System Performance Tuning, by Mike Loukides, 336pages (How can I get my computer to do more work without buying more hardware?) http://www.ora.com/gnn/bus/ora/item/spt.html
- Sun Performance and Tuning: SPARC and Solaris, by Adrian Cockcroft, 280pages www.sun.com/smi/ssoftpress/books/Cockcroft/Cockcroft.html
- The National High Performance Computing and Communications Software Exchange (NHSE) http://www.netlib.org/nhse/sw_catalog/ppt.html
- The Parallel Tools Consortium: http://www.llnl.gov/ptools/ptools.html
- Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods; 10 authors; SIAM http://www.netlib.org/linalg/html_templates/Templates.html
- Parallel Numerical Algerbra by Jim Demmel at UC Berkeley http://www.cs.berkeley.edu/~demmel/cs267_Spr99/
- Parallel Numerical Algorithms by Mike Heath at UIUC http://www.cse.uiuc.edu/cse412/index.html

References - WWW URLs

- General NetLib material: http://www.netlib.org
- ScaLAPACK http://www.netlib.org/scalapack/index.html
- LAPACK -- Linear Algebra PACKage http://www.netlib.org/lapack/
- LINPACK http://www.netlib.org/linpack/
- blas http://www.netlib.org/blas/
- BLACS http://www.netlib.org/blacs/Blacs.html
- ATLAS http://www.netlib.org/atlas/atlas.html
- A Parallel Primer http://ibm.tc.cornell.edu/ibm/pps/doc/primer
- Numerical Methods on Parallel Computers by Lyle Long http://cac.psu.edu/~lnl/597d/
- The EPCC Training and Education Centre at the Unversity of Edinburgh http://www.epcc.ed.ac.uk/epcc-tec/courses
- Internet Parallel Computing Archive at the University of Kent at Canterbury http://unix.hensa.ac.uk/parallel/index.html
- JICS Lecture Notes in Parallel Computing http://www-jics.cs.utk.edu/PCUE

The End



- The End!