Overview of (1D) FEM & Road to Parallel FEM

Kengo Nakajima

Technical & Scientific Computing II (4820-1028)
Seminar on Computer Science II (4810-1205)

- 1D-code for Static Linear-Elastic Problems by Galerkin FEM
- Sparse Linear Solver
  - Conjugate Gradient Method
  - Preconditioning
- Storage of Sparse Matrices
- Program
- Road to Parallel FEM

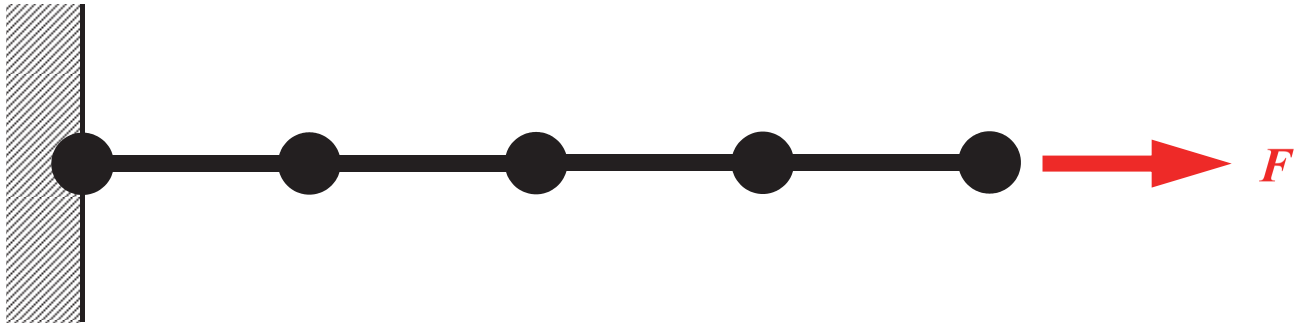
Keywords

- 1D Static Linear Elastic Problems
- Galerkin Method
- Linear Element
- Preconditioned Conjugate Gradient Method

1D Static Linear Elastic Problem （Truss：トラス）

- Only deforms in x-direction (displacement: u)
  - Uniform: Sectional Area A, Young's Modulus E
  - Boundary Conditions (B.C.)
    - x=0 : u=0 (fixed)
    - x=x_{max} : F (axial force)
- Truss: NO bending deformation by G-force

1D Static Linear Elastic Problem



Equilibrium Equation $\frac{\partial \sigma_x}{\partial x} + X = 0$

Strain~Displacement $\varepsilon_x = \frac{\partial u}{\partial x}$

Stress~Strain $\sigma_x = E\varepsilon_x$

$$\frac{\partial}{\partial x}\left(E\frac{\partial u}{\partial x}\right) + X = 0$$

Governing Equation for u

Procedures for Computation

- solve equations for displacement u

$$\frac{\partial}{\partial x}\left(E\frac{\partial u}{\partial x}\right)+X=0$$

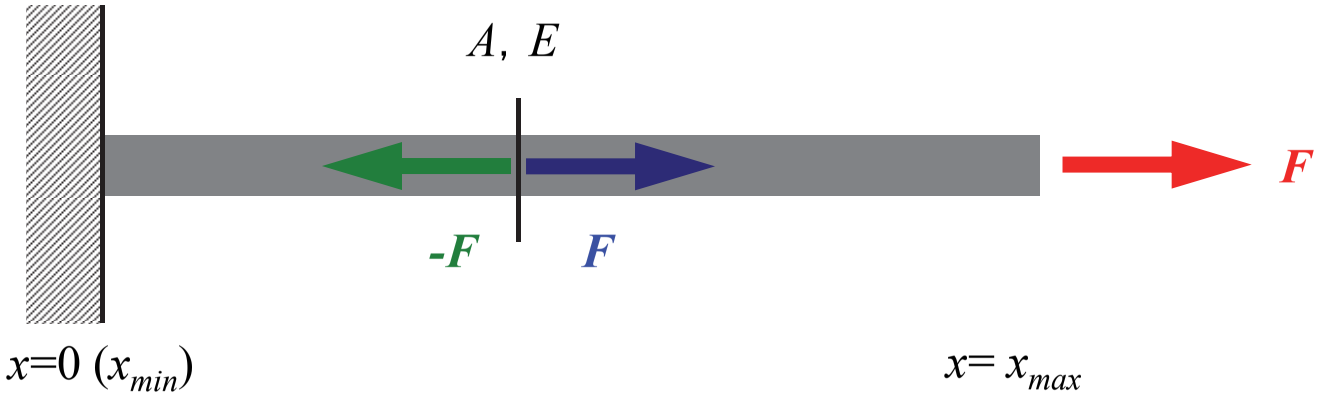
- calc. strain

$$\varepsilon_x=\frac{\partial u}{\partial x}$$

- calc. stress

$$\sigma_x=E\varepsilon_x$$

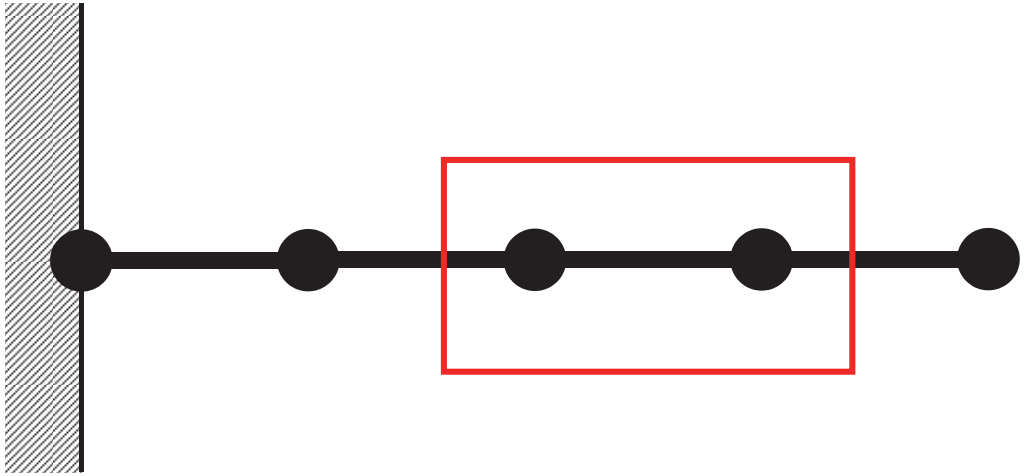
Results Expected



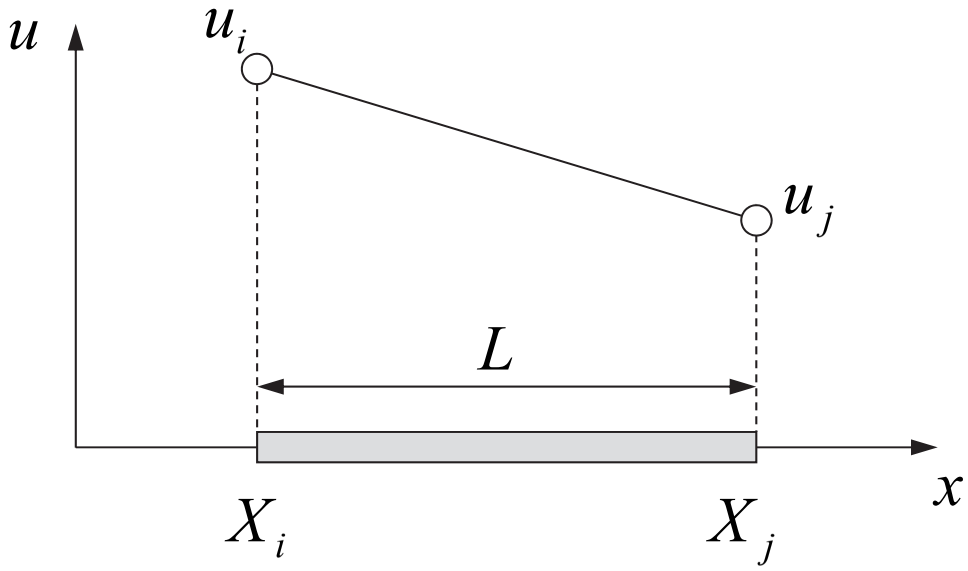
- x_{max}=10, F=5, A=2, E=10.
- Equilibrium State: Uniform axial force at any section
  - Uniform stress: $\sigma = F/A = 2.5$
  - Uniform strain: $\varepsilon = \sigma/E = 0.25$
  - Uniform displacement for each element, if size of each element is uniform (deformation rate= ε) $u = 2.5 @ x = x_{max}$

1D Linear Element (1/4)
一次元線形要素

- 1D Linear Element
  - Length= L
    - Node (Vertex)（節点）
    - Element（要素）



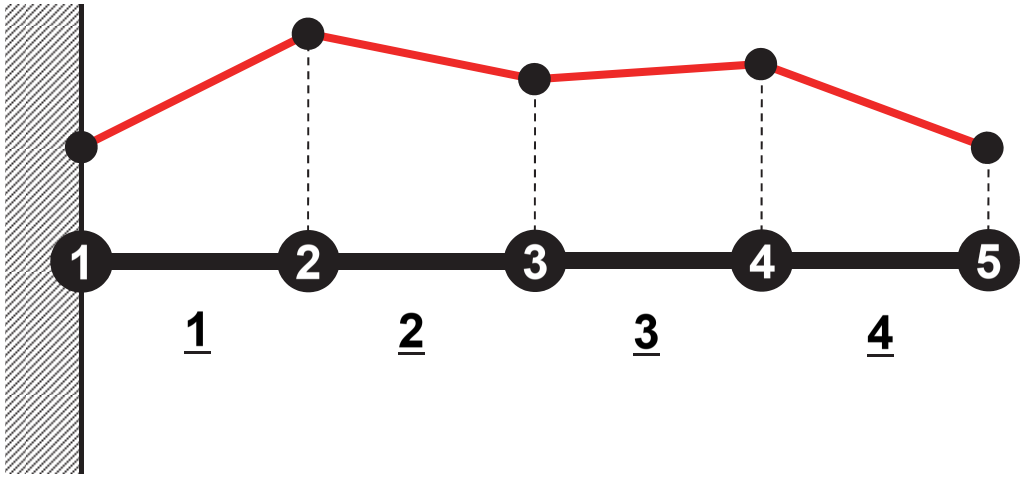
  - u_i Displacement at i
  - u_j Displacement at j



  - Displacement u on each element is linear function of x (Piecewise Linear):

$$u = \alpha_1 + \alpha_2 x$$

Piecewise Linear



Global Node ID
Element ID

Local Node ID
for each elem.

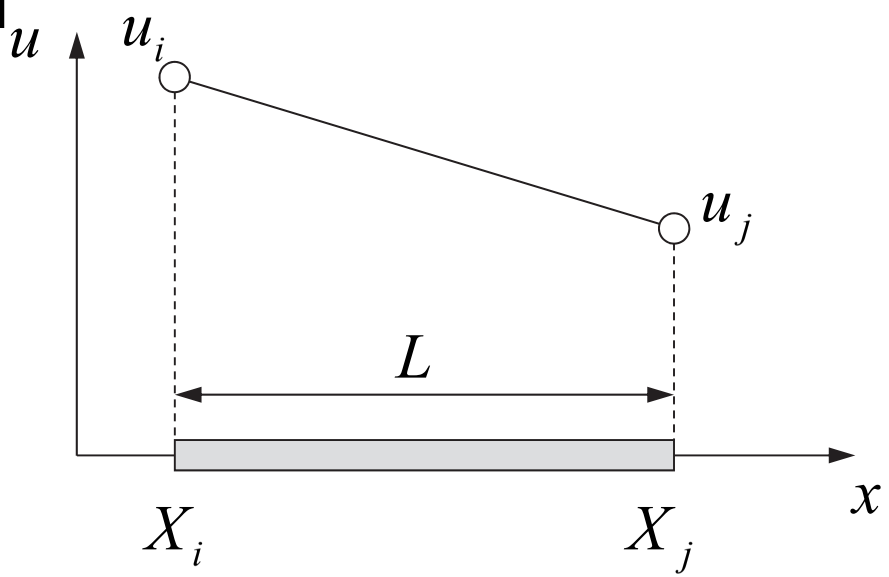
Strain and stress are constant in each element
(might be discontinuous at each “node”)

1D Linear Elem.: Shape Function (2/4)

- Coef's are calculated based on info. at each node

$$u = u_i \,@\, x = X_i, \quad u = u_j \,@\, x = X_j$$

$$u_i = \alpha_1 + \alpha_2 X_i, \quad u_j = \alpha_1 + \alpha_2 X_j$$



- Coefficients:

$$\alpha_1 = \frac{u_i X_j - u_j X_i}{L}, \alpha_2 = \frac{u_j - u_i}{L}$$

- u can be written as follows, according to u_i and u_j:

$$u = \left(\frac{X_j - x}{L}\right) u_i + \left(\frac{x - X_i}{L}\right) u_j$$

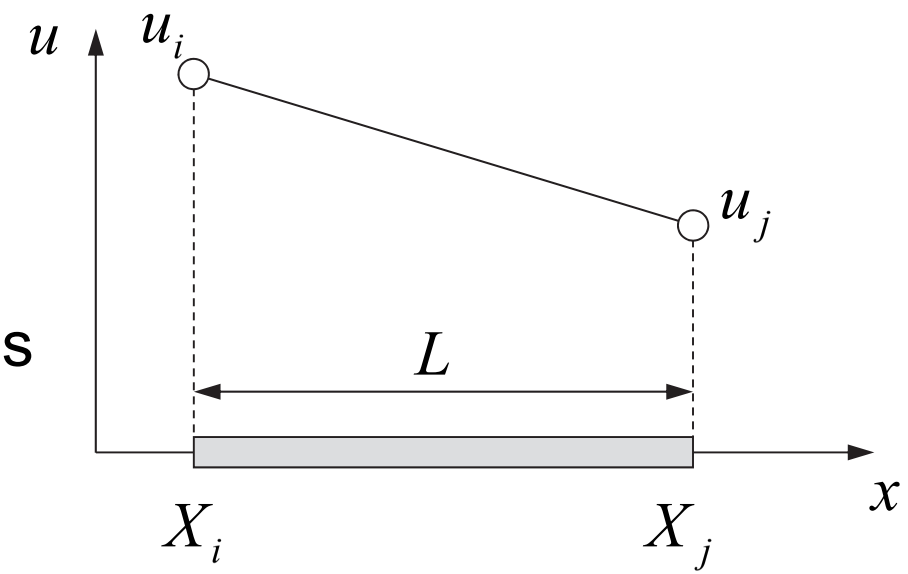
N_i N_j

N_i, N_j
Shape Function or
Interpolation Function
function of x (only)

1D Linear Elem.: Shape Function (3/4)

- Number of Shape Functions = Number of Vertices of Each Element
  - N_i: Function of Position
  - A kind of Test/Trial Functions

$$N_i = \left(\frac{X_j - x}{L} \right), \quad N_j = \left(\frac{x - X_i}{L} \right)$$



- Linear combination of shape functions provides displacement "in" each element
  - Coef's (unknows): Displacement at each node

$$u = N_i u_i + N_j u_j \longleftrightarrow$$

$$u_M = \sum_{i=1}^{M} a_i \Psi_i$$

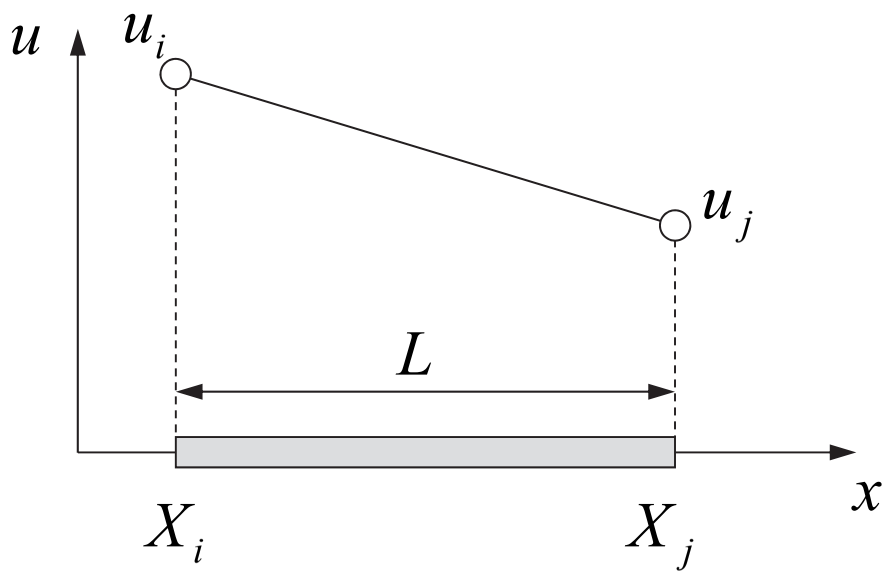
Ψ_i Trial/Test Function (known function of position, defined in domain and at boundary. "Basis" in linear algebra.

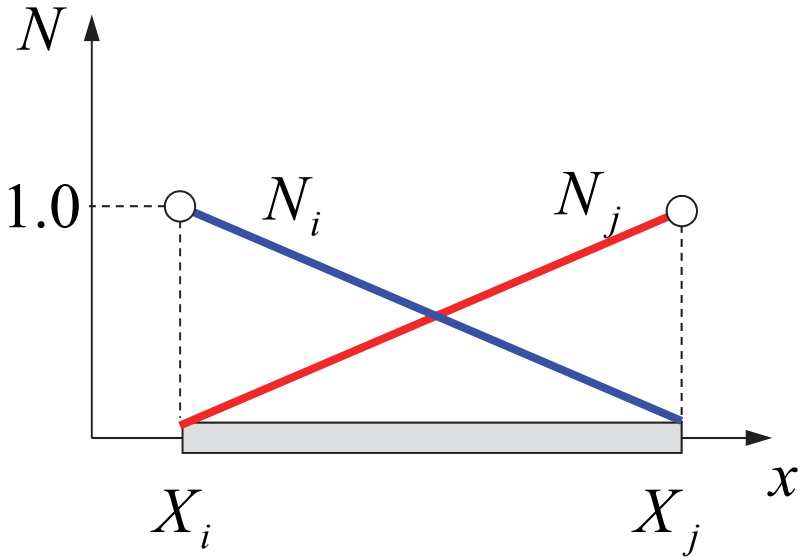
a_i Coefficients (unknown)

1D Linear Elem.: Shape Function (4/4)

- Value of N_i
  - =1 at one of the nodes in element
  - =0 on other nodes

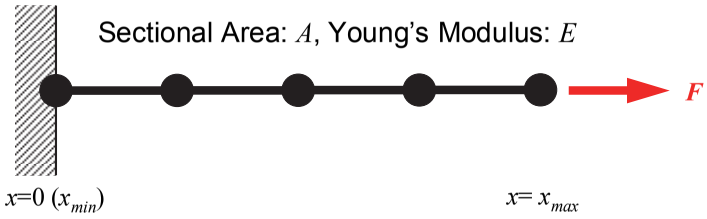
$$N_i = \left(\frac{X_j - x}{L} \right), \quad N_j = \left(\frac{x - X_i}{L} \right)$$





Galerkin Method (1/4)

- Governing Equation for 1D Static Linear-Elastic Problems



$$E\left(\frac{d^2u}{dx^2}\right)+X=0$$

$u=[N]\{\phi\}$ Distribution of Displacement in Each Elem. (Matrix Form), ϕ : Displacement at Each Node

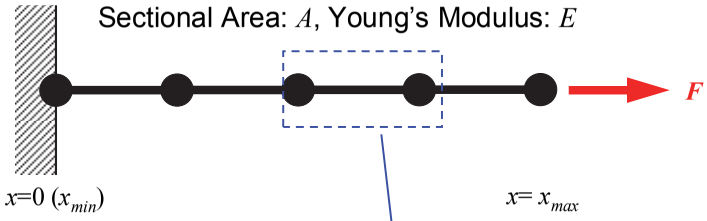
- Following integral equation is obtained at each element by Galerkin method (one of Weighted Residual Methods(WRM)), where $[N]$'s are also weighting functions:

$$\int_V [N]^T\left\{E\left(\frac{d^2u}{dx^2}\right)+X\right\}dV=0$$

Galerkin Method (2/4)

- Green's Theorem (1D)

$$\int_V A\left(\frac{d^2B}{dx^2}\right)dV = \int_S A\frac{dB}{dx}\,dS - \int_V \left(\frac{dA}{dx}\frac{dB}{dx}\right)dV$$



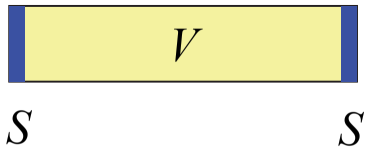
- Apply this to the 1st part of eqn with 2nd-order diff.:

$$\int_V E[N]^T\left(\frac{d^2u}{dx^2}\right)dV = -\int_V E\left(\frac{d[N]^T}{dx}\frac{du}{dx}\right)dV + \int_S E[N]^T\frac{du}{dx}dS$$

- Consider the following terms:

$$u = [N]\{\phi\}, \quad \frac{du}{dx} = \frac{d[N]}{dx}\{\phi\} \quad \bar{\sigma} = E\frac{du}{dx}$$

Stress on Surfaces of Each Element

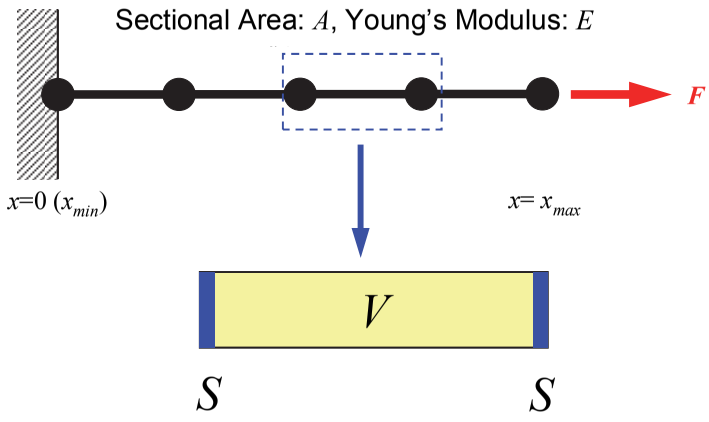


Galerkin Method (3/4)

- Finally following eqn is obtained by considering term due to body forces (X)

$$-\int_V E\left(\frac{d[N]^T}{dx}\frac{d[N]}{dx}\right)dV\cdot\{\phi\}$$

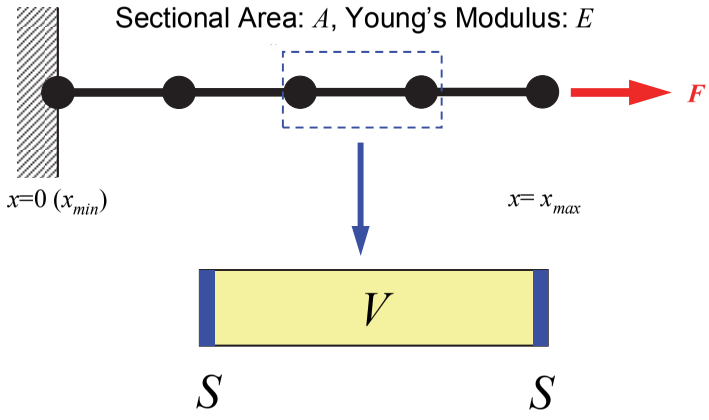
$$+\int_S \bar{\sigma}[N]^T dS+\int_V X[N]^T dV=0$$



- This is called "weak form（弱形式）". Original PDE consists of terms with 2nd-order diff., but this "weak form" only includes 1st-order diff by Green's theorem.
  - Requirements for shape functions are "weaker" in "weak form". Linear functions can describe effects of 2nd-order differentiation.

Galerkin Method (4/4)

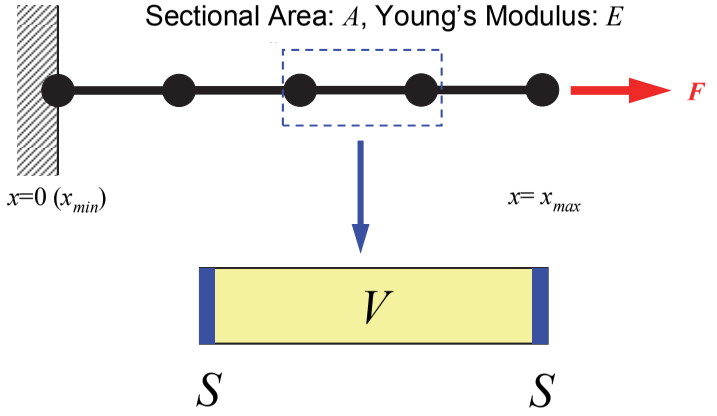
$$-\int_V E\left(\frac{d[N]^T}{dx}\frac{d[N]}{dx}\right)dV\cdot\{\phi\}$$
$$+\int_S \bar{\sigma}[N]^T dS+\int_V X[N]^T dV=0$$



- These terms coincide at element boundaries and disappear. Finally, only terms on the domain boundaries remain.

Weak Form and Boundary Conditions

- Value of dependent variable is defined (Dirichlet)
  - Weighting Function = 0
  - Principal B.C. (Boundary Condition)（第一種境界条件）
  - Essential B.C.（基本境界条件）
- Derivatives of Unknowns (Neumann)
  - Naturally satisfied in weak form
  - Secondary B.C.（第二種境界条件）
  - Natural B.C（自然境界条件）



$$-\int_V E\left(\frac{d[N]^T}{dx}\frac{d[N]}{dx}\right)dV\cdot\{\phi\}$$

$$+\int_S \bar{\sigma}[N]^T dS+\int_V X[N]^T dV=0$$

where $\bar{\sigma}=E\dfrac{du}{dx}$

Weak Form with B.C.: on each elem.

$$[k]^{(e)}\{\phi\}^{(e)} = \{f\}^{(e)}$$

$$[k]^{(e)} = \int_V E\left(\frac{d[N]^T}{dx}\frac{d[N]}{dx}\right)dV$$

$$[f]^{(e)} = \int_V X[N]^T dV + \int_S \bar{\sigma}[N]^T dS$$

Integration over Each Element: $[k]$

$$N_i = \left(\frac{X_j - x}{L}\right), \quad N_j = \left(\frac{x - X_i}{L}\right)$$

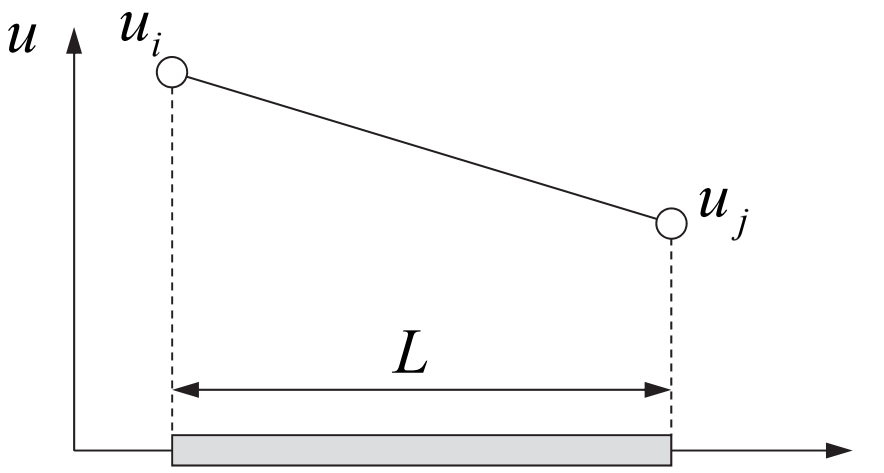
$$\frac{dN_i}{dx} = \left(\frac{-1}{L}\right), \quad \frac{dN_j}{dx} = \left(\frac{1}{L}\right)$$

$$\int_V E\left(\frac{d[N]^T}{dx}\frac{d[N]}{dx}\right)dV$$

$$= E\int_0^L \begin{bmatrix} -1/L \\ 1/L \end{bmatrix}\left[-1/L, 1/L\right] A\, dx$$

2x1 matrix　1x2 matrix

$$= \frac{EA}{L^2}\int_0^L \begin{bmatrix} +1 & -1 \\ -1 & +1 \end{bmatrix} dx = \frac{EA}{L}\begin{bmatrix} +1 & -1 \\ -1 & +1 \end{bmatrix}$$



$X_i \Rightarrow 0$　　$X_j \Rightarrow L$

$$N_i = \left(1 - \frac{x}{L}\right), \quad N_j = \left(\frac{x}{L}\right)$$

A: Sectional Area

L: Length

Integration over Each Element: $\{f\}$ (1/2)

$$N_i = \left(\frac{X_j - x}{L} \right), \quad N_j = \left(\frac{x - X_i}{L} \right) \qquad \frac{dN_i}{dx} = \left(\frac{-1}{L} \right), \quad \frac{dN_j}{dx} = \left(\frac{1}{L} \right)$$

$$N_i = \left(1 - \frac{x}{L} \right), \quad N_j = \left(\frac{x}{L} \right)$$

$$\int_V X [N]^T dV = XA \int_0^L \begin{bmatrix} 1 - x/L \\ x/L \end{bmatrix} dx = \frac{XAL}{2} \begin{Bmatrix} 1 \\ 1 \end{Bmatrix}$$

Body Force

A: Sectional Area

L: Length

Integration over Each Element: $\{f\}$ (2/2)

$$N_i = \left(\frac{X_j - x}{L}\right), \quad N_j = \left(\frac{x - X_i}{L}\right) \qquad \frac{dN_i}{dx} = \left(\frac{-1}{L}\right), \quad \frac{dN_j}{dx} = \left(\frac{1}{L}\right)$$

$$\int_V X[N]^T dV = XA\int_0^L \begin{bmatrix} 1 - x/L \\ x/L \end{bmatrix} dx = \frac{XAL}{2}\begin{Bmatrix} 1 \\ 1 \end{Bmatrix}$$ Body Force

$$\int_S \bar{\sigma}[N]^T dS = \bar{\sigma}A\big|_{x=L} = \bar{\sigma}A\begin{Bmatrix} 0 \\ 1 \end{Bmatrix}$$ Surface Force

when surface force acts on only this surface.

Global Equations

- Accumulate Element Equations:

$$[k]^{(e)}\{\phi\}^{(e)} = \{f\}^{(e)}$$ Element Matrix, Element Equations

$$[K]\cdot\{\Phi\} = \{F\}$$ Global Matrix, Global Equations

$$[K] = \sum [k], \quad \{F\} = \sum \{f\}$$

$$\{\Phi\}: global\ vector\ of\ \{\phi\}$$

This is the final linear equations (global equations) to be solved.

ECCS 2012 System

Creating Directory

```
>$ cd Documents
>$ mkdir fem1
>$ cd fem1
```

create your directory under this (good for Windows)
your favorite name

This is your “top” directory, and is called **<$fem1>** in this class.

1D Code for Static Linear-Elastic Problems

```
>$ cd <$fem1>
>$ cp /home03/skengon/Documents/class/fem1/1d.tar .
>$ tar xvf 1d.tar
>$ cd 1d
```

Compile & GO !

```
>$ cd <$fem1>/1d
>$ cc -O 1d.c       (or g95 -O 1d.f)
>$ ./a.out
```

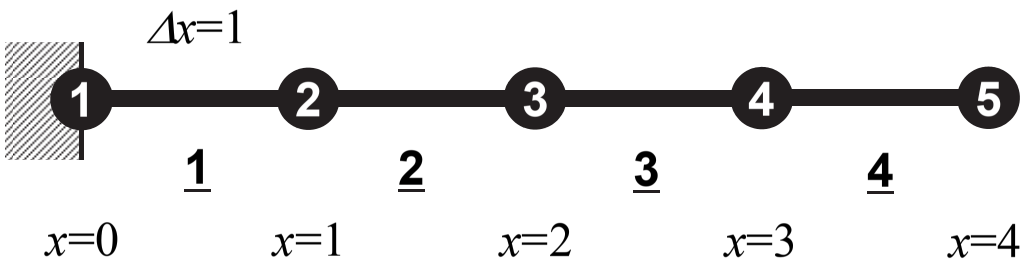
Control Data `input.dat`

```
4
1.0  1.0  1.0  1.0
100
1.e-8
```

NE (Number of Elements)
Δx (Length of Each Elem.: **L**) , **F, A, E**
Number of MAX. Iterations for CG Solver
Convergence Criteria for CG Solver

$$\sigma = \frac{F}{A} = \frac{1}{1} = 1$$

$$\frac{du}{dx} = \varepsilon = \frac{\sigma}{E} = \frac{1}{1} = 1$$



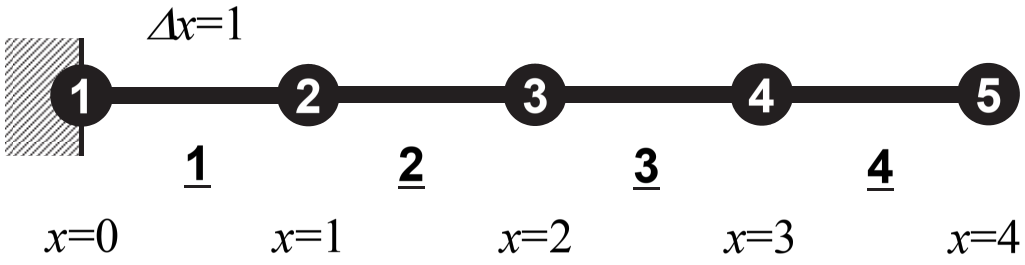
Element ID
Node ID (Global)

Results

```
>$ ./a.out

 4 iters, RESID=    0.000000E+00 U(N)=     4.000000E+00

### DISPLACEMENT      at each node (computed)
       1    0.000000E+00
       2    1.000000E+00
       3    2.000000E+00
       4    3.000000E+00
       5    4.000000E+00

### STRESS            at each element (computed, theoretical)
       1    1.000000E+00    1.000000E+00
       2    1.000000E+00    1.000000E+00
       3    1.000000E+00    1.000000E+00
       4    1.000000E+00    1.000000E+00
```

$$\sigma = \frac{F}{A} = \frac{1}{1} = 1$$

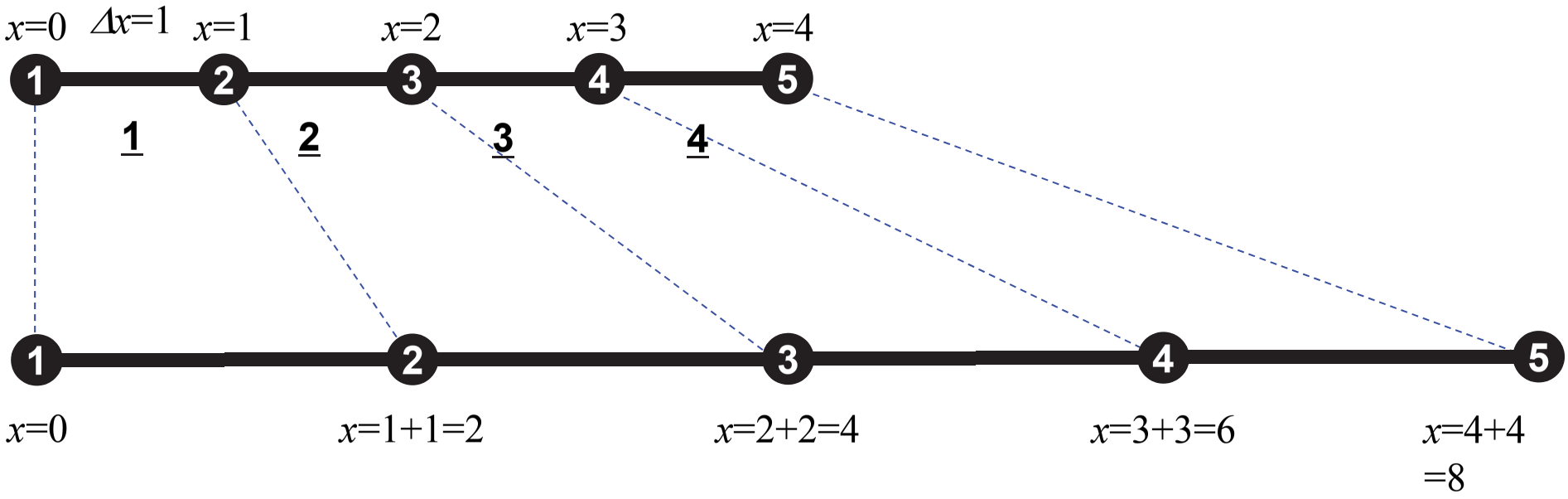
$$\frac{du}{dx} = \varepsilon = \frac{\sigma}{E} = \frac{1}{1} = 1$$



Element ID
Node ID (Global)

Strain=1.00 (100%)

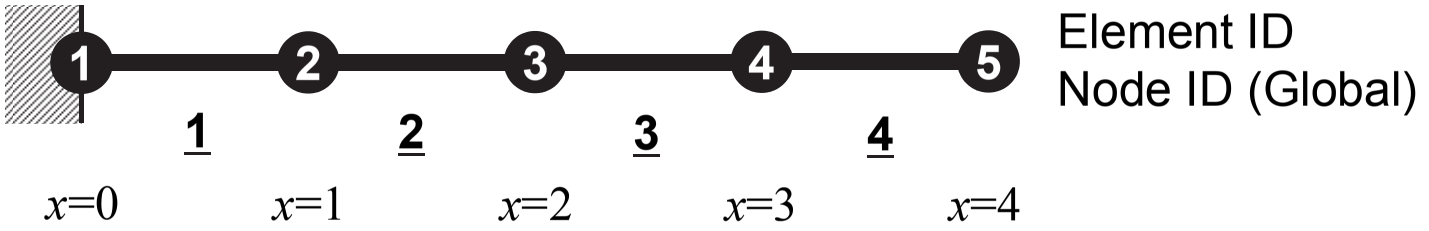
Length of each element is doubled compared to initial condition




```
### DISPLACEMENT      at each node (computed)
         1     0.000000E+00
         2     1.000000E+00
         3     2.000000E+00
         4     3.000000E+00
         5     4.000000E+00
```

Element Eqn's/Accumulation (1/3)

- 4 elements, 5 nodes



- $[k]$ and $\{f\}$ of Element-1:

$$[k]^{(1)} = \frac{EA}{L}\begin{bmatrix} +1 & -1 \\ -1 & +1 \end{bmatrix} \qquad \{f\}^{(1)} = \frac{XAL}{2}\begin{Bmatrix} 1 \\ 1 \end{Bmatrix} + \int_S \bar{\sigma}[N]^T dS = \begin{Bmatrix} 0 \\ 0 \end{Bmatrix}$$

X=0 in this case

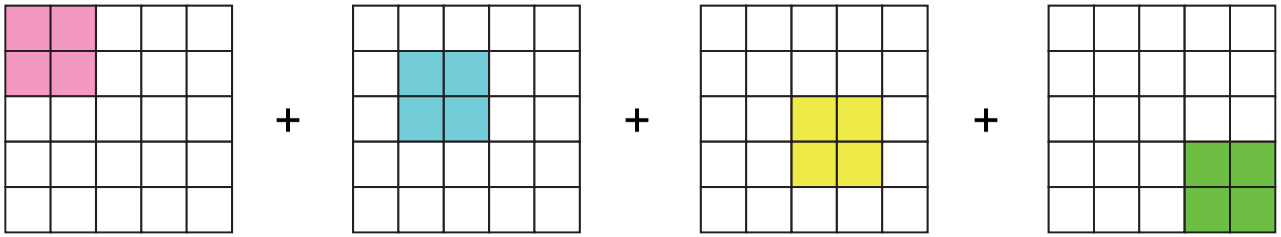
- As for Element-4:

$$[k]^{(4)} = \frac{EA}{L}\begin{bmatrix} +1 & -1 \\ -1 & +1 \end{bmatrix} \qquad \{f\}^{(4)} = \int_S \bar{\sigma}[N]^T dS = \bar{\sigma}A\begin{Bmatrix} 0 \\ 1 \end{Bmatrix} = \begin{Bmatrix} 0 \\ F \end{Bmatrix}$$

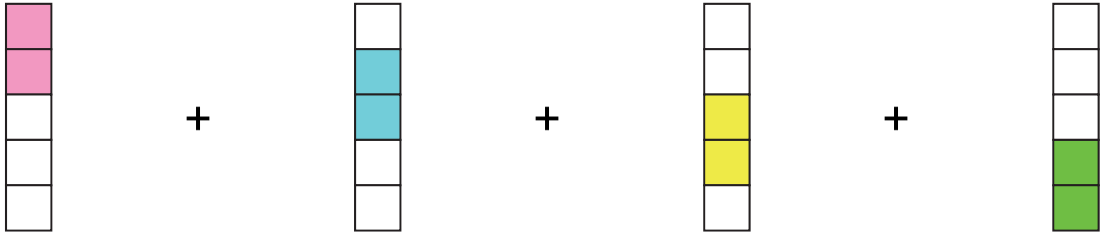
Element Eqn's/Accumulation (2/3)

- Element-by-Element Accumulation:

$$[K] = \sum_{e=1}^{4} [k]^{(e)} =$$



$$\{F\} = \sum_{e=1}^{4} \{f\}^{(e)} =$$

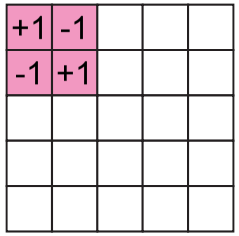


Element Eqn's/Accumulation (3/3)

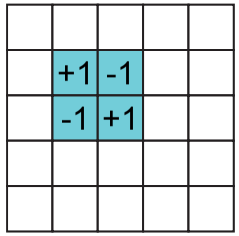
- Relations to FDM

$$[k]^{(e)} = \frac{EA}{L}\begin{bmatrix} +1 & -1 \\ -1 & +1 \end{bmatrix}$$

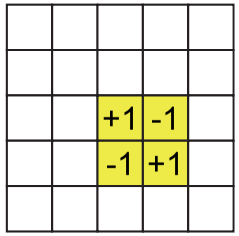
$$[K] = \sum_{e=1}^{4} [k]^{(e)} = \Big[$$



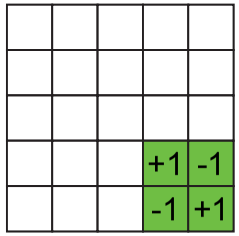
$+$



$+$

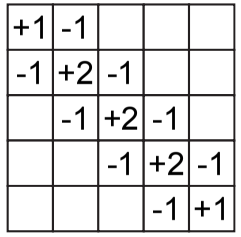


$+$



$\Big] \times \frac{EA}{L}$

$$=$$



$$\times \frac{EA}{L}$$

$$-\int_V \left(\frac{d^2u}{dx^2}\right) dV = -\int_V \left(\frac{u_{i+1} - 2u_i + u_{i-1}}{L^2}\right) dV$$

$$= -\left(\frac{u_{i+1} - 2u_i + u_{i-1}}{L^2}\right) \cdot AL = -(u_{i+1} - 2u_i + u_{i-1}) \cdot \frac{A}{L}$$

Something familiar …

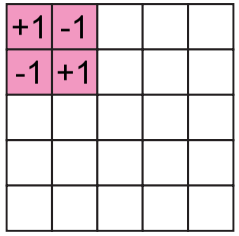
FEM: Coefficient Matrices are generally sparse (many ZERO's)

Element-by-Element Operation

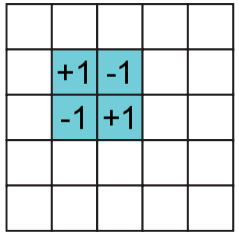
very flexible if each element has different material property, size, etc.

$$[k]^{(e)} = \frac{E^{(e)} A^{(e)}}{L^{(e)}} \begin{bmatrix} +1 & -1 \\ -1 & +1 \end{bmatrix}$$

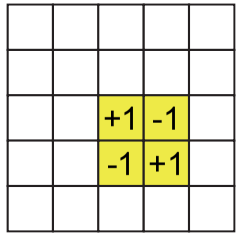
$$[K] = \sum_{e=1}^{4} [k^{(e)}] =$$



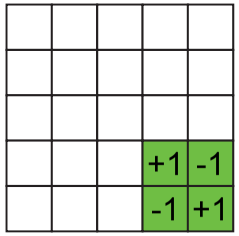
$$\times \frac{E^{(1)} A^{(1)}}{L^{(1)}} +$$



$$\times \frac{E^{(2)} A^{(2)}}{L^{(2)}}$$

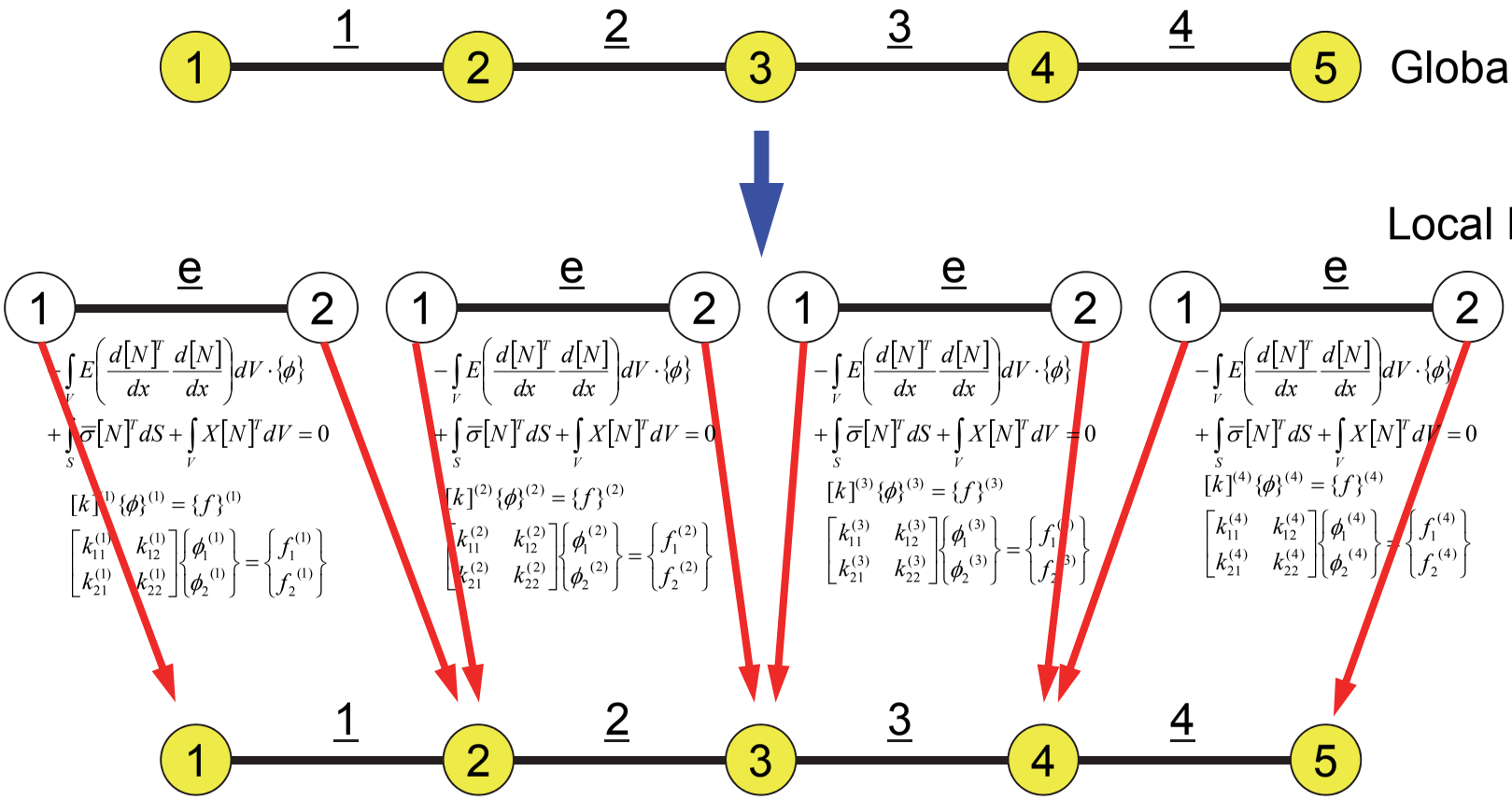


$$\times \frac{E^{(3)} A^{(3)}}{L^{(3)}} +$$



$$\times \frac{E^{(4)} A^{(4)}}{L^{(4)}}$$

Element/Global Operations



$$[K]\{\Phi\}=\{F\}$$

$$\begin{bmatrix} D_1 & AU_{11} & & & \\ AL_{21} & D_2 & AU_{21} & & \\ & AL_{31} & D_3 & AU_{31} & \\ & & AL_{41} & D_4 & AU_{41} \\ & & & AL_{51} & D_5 \end{bmatrix}\begin{Bmatrix}\Phi_1\\ \Phi_2\\ \Phi_3\\ \Phi_4\\ \Phi_5\end{Bmatrix}=\begin{Bmatrix}B_1\\ B_2\\ B_3\\ B_4\\ B_5\end{Bmatrix}$$

Mapping Information needed, from element-matrix to global-matrix.

- 1D-code for Static Linear-Elastic Problems by Galerkin FEM
- **Sparse Linear Solver**
  - **Conjugate Gradient Method**
  - **Preconditioning**
- Storage of Sparse Matrices
- Program
- Road to Parallel FEM

Large-Scale Linear Equations in Scientific Applications

- Solving large-scale linear equations **Ax=b** is the most important and **expensive** part of various types of scientific computing.
  - for both linear and nonlinear applications
- Various types of methods proposed & developed.
  - for dense and sparse matrices
  - classified into ***direct*** and ***iterative*** methods
- Dense Matrices:密行列: Globally Coupled Problems
  - BEM, Spectral Methods, MO/MD (gas, liquid)
- Sparse Matrices:疎行列: Locally Defined Problems
  - FEM, FDM, DEM, MD (solid), BEM w/FMM

Conjugate Gradient Method
共役勾配法

- Conjugate Gradient: CG
  - Most popular “non-stationary” iterative method
- for Symmetric Positive Definite (SPD) Matrices
  - 対称正定
  - $\{x\}^T[A]\{x\}>0$ for arbitrary $\{x\}$
  - All of diagonal components, eigenvaules and leading principal minors > 0（主小行列式・首座行列式）
  - Matrices of Galerkin-based FEM: heat conduction, Poisson, static linear elastic problems
- Algorithm
  - “Steepest Descent Method”
  - $\boldsymbol{x^{(i)}= x^{(i-1)} + \alpha_i p^{(i)}}$
    - $x^{(i)}$: solution, $p^{(i)}$: search direction, α_i: coefficient
  - Solution $\{x\}$ minimizes $\boldsymbol{\{x-y\}^T[A]\{x-y\}}$, where $\{y\}$ is exact solution.

$$\det \begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} & \cdots & a_{1n} \\ a_{21} & a_{22} & a_{23} & a_{24} & \cdots & a_{2n} \\ a_{31} & a_{32} & a_{33} & a_{34} & \cdots & a_{3n} \\ a_{41} & a_{42} & a_{43} & a_{44} & \cdots & a_{4n} \\ \vdots & \vdots & \vdots & \vdots & & \vdots \\ a_{n1} & a_{n2} & a_{n3} & a_{n4} & \cdots & a_{nn} \end{bmatrix}$$

Procedures of Conjugate Gradient

Compute $r^{(0)}= b-[A]x^{(0)}$
for i= 1, 2, …
  $z^{(i-1)}= r^{(i-1)}$
  $\rho_{i-1}= r^{(i-1)}\ z^{(i-1)}$
  if i=1
    $p^{(1)}= z^{(0)}$
  else
    $\beta_{i-1}= \rho_{i-1}/\rho_{i-2}$
    $p^{(i)}= z^{(i-1)} + \beta_{i-1}\ p^{(i-1)}$
  endif
  $q^{(i)}= [A]p^{(i)}$
  $\alpha_i = \rho_{i-1}/p^{(i)}q^{(i)}$
  $x^{(i)}= x^{(i-1)} + \alpha_i p^{(i)}$
  $r^{(i)}= r^{(i-1)} - \alpha_i q^{(i)}$
  check convergence |r|
end

- Mat-Vec. Multiplication
- Dot Products
- DAXPY

$x^{(i)}$: Vector
α_i : Scalar

Preconditioning for Iterative Solvers

- Convergence rate of iterative solvers strongly depends on the spectral properties (eigenvalue distribution) of the coefficient matrix **A**.
  - Eigenvalue distribution is small, eigenvalues are close to 1
  - In "ill-conditioned" problems, "condition number" (ratio of max/min eigenvalue if **A** is symmetric) is large（条件数）.
- A preconditioner **M** (whose properties are similar to those of **A**) transforms the linear system into one with more favorable spectral properties（前処理）
  - **M** transforms **Ax=b** into **A'x=b'** where $\mathbf{A'=M^{-1}A}$, $\mathbf{b'=M^{-1}b}$
  - If **M~A**, $\mathbf{M^{-1}A}$ is close to identity matrix.
  - If $\mathbf{M^{-1}=A^{-1}}$, this is the best preconditioner (Gaussian Elim.)
  - Generally, **A'x'=b'** where $\mathbf{A'=M_L^{-1}AM_R^{-1}}$, $\mathbf{b'=M_L^{-1}b}$, $\mathbf{x'=M_Rx}$
  - $\mathbf{M_L/M_R}$: Left/Right Preconditioning（左／右前処理）

Preconditioned CG Solver

Compute $r^{(0)}= b-[A]x^{(0)}$
for i= 1, 2, …
  solve $[M]z^{(i-1)}= r^{(i-1)}$
  $\rho_{i-1}= r^{(i-1)}\ z^{(i-1)}$
  if i=1
    $p^{(1)}= z^{(0)}$
  else
    $\beta_{i-1}= \rho_{i-1}/\rho_{i-2}$
    $p^{(i)}= z^{(i-1)} + \beta_{i-1}\ p^{(i-1)}$
  endif
  $q^{(i)}= [A]p^{(i)}$
  $\alpha_i = \rho_{i-1}/p^{(i)}q^{(i)}$
  $x^{(i)}= x^{(i-1)} + \alpha_i p^{(i)}$
  $r^{(i)}= r^{(i-1)} - \alpha_i q^{(i)}$
  check convergence |r|
end

ILU(0), IC(0)

- Widely used Preconditioners for Sparse Matrices
  - Incomplete LU Factorization
  - Incomplete Cholesky Factorization (for Symmetric Matrices)

- Incomplete Direct Method
  - Even if original matrix is sparse, inverse matrix is not necessarily sparse.
  - fill-in
  - ILU(0)/IC(0) without fill-in have same non-zero pattern with the original (sparse) matrices

Diagonal Scaling, Point-Jacobi

$$[M]=\begin{bmatrix} D_1 & 0 & \dots & 0 & 0 \\ 0 & D_2 & & 0 & 0 \\ \dots & & \dots & & \dots \\ 0 & 0 & & D_{N-1} & 0 \\ 0 & 0 & \dots & 0 & D_N \end{bmatrix}$$

- `solve [M]z(i-1)= r(i-1)` is very easy.
- Provides fast convergence for simple problems.
- 1d.f, 1d.c

- 1D-code for Static Linear-Elastic Problems by Galerkin FEM
- Sparse Linear Solver
  - Conjugate Gradient Method
  - Preconditioning
- **Storage of Sparse Matrices**
- Program
- Road to Parallel FEM

Coef. Matrix derived from FEM

- Sparse Matrix
  - Many “0”’s
- Storing all components (e.g. $\mathrm{A}(i,j)$) is not efficient for sparse matrices
  - $\mathrm{A}(i,j)$ is suitable for dense matrices

$$
\begin{bmatrix}
D & X & & & X & X & & & & & & & & & & \\
X & D & X & & X & X & X & & & & & & & & & \\
 & X & D & X & & X & X & X & & & & & & & & \\
 & & X & D & & & X & X & & & & & & & & \\
X & X & & & D & X & & & X & X & & & & & & \\
X & X & X & & X & D & X & & X & X & X & & & & & \\
 & X & X & X & & X & D & X & & X & X & X & & & & \\
 & & X & X & & & X & D & & & X & X & & & & \\
 & & & & X & X & & & D & X & & & X & X & & \\
 & & & & X & X & X & & X & D & X & & X & X & X & \\
 & & & & & X & X & X & & X & D & X & & X & X & X \\
 & & & & & & X & X & & & X & D & & & X & X \\
 & & & & & & & & X & X & & & D & X & & \\
 & & & & & & & & X & X & X & & X & D & X & \\
 & & & & & & & & & X & X & X & & X & D & X \\
 & & & & & & & & & & X & X & & & X & D
\end{bmatrix}
\begin{Bmatrix} \Phi_1 \\ \Phi_2 \\ \Phi_3 \\ \Phi_4 \\ \Phi_5 \\ \Phi_6 \\ \Phi_7 \\ \Phi_8 \\ \Phi_9 \\ \Phi_{10} \\ \Phi_{11} \\ \Phi_{12} \\ \Phi_{13} \\ \Phi_{14} \\ \Phi_{15} \\ \Phi_{16} \end{Bmatrix}
=
\begin{Bmatrix} F_1 \\ F_2 \\ F_3 \\ F_4 \\ F_5 \\ F_6 \\ F_7 \\ F_8 \\ F_9 \\ F_{10} \\ F_{11} \\ F_{12} \\ F_{13} \\ F_{14} \\ F_{15} \\ F_{16} \end{Bmatrix}
$$

- Number of non-zero off-diagonal components is O(100) in FEM
  - If number of unknowns is 10^8 :
    - $\mathrm{A}(i,j)$: $O(10^{16})$ words
    - Actual Non-zero Components : $O(10^{10})$ words
- Only (really) non-zero off-diag. components should be stored on memory

Variables/Arrays in 1d.f, 1d.c related to coefficient matrix

name	type	size	description
N	I	-	# Unknowns
NPLU	I	-	# Non-Zero Off-Diagonal Components
Diag(:)	R	N	Diagonal Components
U(:)	R	N	Unknown Vector
Rhs(:)	R	N	RHS Vector
Index(:)	I	0:N N+1	Off-Diagonal Components (Number of Non-Zero Off-Diagonals at Each ROW)
Item(:)	I	NPLU	Off-Diagonal Components (Corresponding Column ID)
AMat(:)	R	NPLU	Off-Diagonal Components (Value)

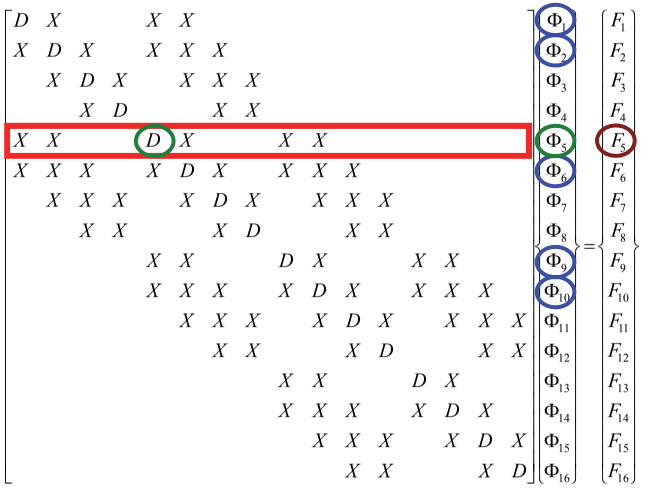
Only non-zero components are stored according to “Compressed Row Storage”.

Mat-Vec. Multiplication for Sparse Matrix

Compressed Row Storage (CRS)

Diag (i)	Diagonal Components (REAL, i=1~N)
Index(i)	Number of Non-Zero Off-Diagonals at Each ROW (INT, i=0~N)
Item(k)	Off-Diagonal Components (Corresponding Column ID) (INT, k=1, index(N))
AMat(k)	Off-Diagonal Components (Value) (REAL, k=1, index(N))

```
{Y}= [A]{X}

do i= 1, N
  Y(i)= Diag(i)*X(i)
  do k= Index(i-1)+1, Index(i)
    Y(i)= Y(i) + Amat(k)*X(Item(k))
  enddo
enddo
```



Mat-Vec. Multiplication for Dense Matrix

Very Easy, Straightforward

$$\begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1,N-1} & a_{1,N} \\ a_{21} & a_{22} & & a_{2,N-1} & a_{2,N} \\ \cdots & & \cdots & & \cdots \\ a_{N-1,1} & a_{N-1,2} & & a_{N-1,N-1} & a_{N-1,N} \\ a_{N,1} & a_{N,2} & \cdots & a_{N,N-1} & a_{N,N} \end{bmatrix} \begin{Bmatrix} x_1 \\ x_2 \\ \vdots \\ x_{N-1} \\ x_N \end{Bmatrix} = \begin{Bmatrix} y_1 \\ y_2 \\ \vdots \\ y_{N-1} \\ y_N \end{Bmatrix}$$

```
{Y}= [A]{X}

do j= 1, N
  Y(j)= 0.d0
  do i= 1, N
    Y(j)= Y(j) + A(i,j)*X(i)
  enddo
enddo
```

Compressed Row Storage (CRS)：C

0	1.1 ◎	2.4 ①,0	3.2 ④,1		
1	3.6 ①	4.3 ◎,2	2.5 ③,3	3.7 ⑤,4	9.1 ⑦,5
2	5.7 ②	1.5 ④,6	3.1 ⑥,7		
3	9.8 ③	4.1 ①,8	2.5 ④,9	2.7 ⑤,10	
4	11.5 ④	3.1 ◎,11	9.5 ①,12	10.4 ②,13	4.3 ⑥,14
5	12.4 ⑤	6.5 ②,15	9.5 ⑥,16		
6	23.1 ⑥	6.4 ①,17	2.5 ②,18	1.4 ⑤,19	13.1 ⑦,20
7	51.3 ⑦	9.5 ①,21	1.3 ②,22	9.6 ③,23	3.1 ⑤,24

Diag (i)	Diagonal Components (REAL, i=1~N)
Index(i)	Number of Non-Zero Off-Diagonals at Each ROW (INT, i=0~N)
Item(k)	Off-Diagonal Components (Corresponding Column ID) （INT, k=1, index(N)）
AMat(k)	Off-Diagonal Components (Value) （ REAL, k=1, index(N) ）

```
{Y}=[A]{X}

for(i=0;i<N;i++){
    Y[i] = Diag[i] * X[i];
    for(k=Index[i];k<Index[i+1];k++){
        Y[i] += AMat[k]*X[Item[k]];
      }
}
```

- 1D-code for Static Linear-Elastic Problems by Galerkin FEM
- Sparse Linear Solver
  - Conjugate Gradient Method
  - Preconditioning
- Storage of Sparse Matrices

- **Program**

- Road to Parallel FEM

Finite Element Procedures

- Initialization
  - Control Data
  - Node, Connectivity of Elements (N: Node#, NE: Elem#)
  - Initialization of Arrays (Global/Element Matrices)
  - Element-Global Matrix Mapping (Index, Item)
- Generation of Matrix
  - Element-by-Element Operations (do icel= 1, NE)
    - Element matrices
    - Accumulation to global matrix
  - Boundary Conditions
- Linear Solver
  - Conjugate Gradient Method
- Calculation of Stress

Program: 1d.c (1/7)
variables and arrays

```
/*
//   1D Solid Mechanics for Truss Elements solved by
//   CG (Conjugate Gradient) Method
//
//   d/dx(EdU/dx) + F = 0
//   U=0@x=0
*/

#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <assert.h>

int main(){
          int NE, N, NPLU, IterMax, errno;
          int R, Z, Q, P, DD;

          double dX, Resid, Eps, Area, F, Young;
          double X1, X2, U1, U2, DL, Strain, Sigma, Ck;
          double *U, *Rhs, *X;
          double *Diag, *AMat;
          double **W;

          int *Index, *Item, *IceInod;
          double Kmat[2][2], Emat[2][2];

          int i, j, in1, in2, k, icel, k1, k2, jS;
          int iter;
          FILE *fp;
          double BNorm2, Rho, Rho1=0.0, C1, Alpha, DNorm2;
          int ierr = 1;
```

Variable/Arrays (1/2)

Name	Type	Size	I/O	Definition
NE	I		I	# Element
N	I		O	# Node
NPLU	I		O	# Non-Zero Off-Diag. Components
IterMax	I		I	MAX Iteration Number for CG
errno	I		O	ERROR flag
R, Z, Q, P, DD	I		O	Name of Vectors in CG
dX	R		I	Length of Each Element
Resid	R		O	Residual for CG
Eps	R		I	Convergence Criteria for CG
Area	R		I	Sectional Area of Element
F	R		I	Axial Force F at X=Xmax
Young	R		I	Young's Modulus
X1,X2,U1,U2	R		O	Location/Displacement at Local Nodes

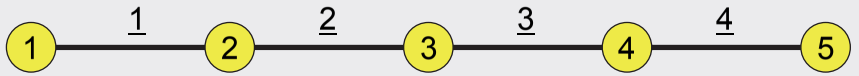
Variable/Arrays (2/2)

Name	Type	Size	I/O	Definition
DL, Ck	R		O	Coef's for Element Matrix
Strain, Stress	R		O	Element Strain, Element Stress
X	R	N	O	Location of Each Node
U	R	N	O	Displacement of Each Node
Rhs	R	N	O	RHS Vector
Diag	R	N	O	Diagonal Components
W	R	[4][N]	O	Work Array for CG
Amat	R	NPLU	O	Off-Diagonal Components (Value)
Index	I	N+1	O	Number of Non-Zero Off-Diagonals at Each ROW
Item	I	NPLU	O	Off-Diagonal Components (Corresponding Column ID)
Icelnod	I	2*NE	O	Node ID for Each Element
Kmat	R	[2][2]	O	Element Matrix [k]
Emat	R	[2][2]	O	Element Matrix

Program: 1d.c (2/7)
Initialization, Allocation of Arrays

```
/*
// +-------+
// | INIT. |
// +-------+
*/
        fp = fopen("input.dat", "r");
        assert(fp != NULL);
        fscanf(fp, "%d", &NE);
        fscanf(fp, "%lf %lf %lf %lf", &dX, &F, &Area, &Young);
        fscanf(fp, "%d", &IterMax);
        fscanf(fp, "%lf", &Eps);
        fclose(fp);

        N= NE + 1;

        U    = calloc(N, sizeof(double));
        X    = calloc(N, sizeof(double));
        Diag = calloc(N, sizeof(double));

        AMat = calloc(2*N-2, sizeof(double));

        Rhs = calloc(N, sizeof(double));
        Index= calloc(N+1, sizeof(int));
        Item = calloc(2*N-2, sizeof(int));

        Icelnod= calloc(2*NE, sizeof(int));
```

```
input.dat
4                       NE (Number of Elements)
1.0  1.0  1.0  1.0      Δx (Length of Each Elem.: L), F, A, E
100                     Number of MAX. Iterations for CG Solver
1.e-8                   Convergence Criteria for CG Solver
```



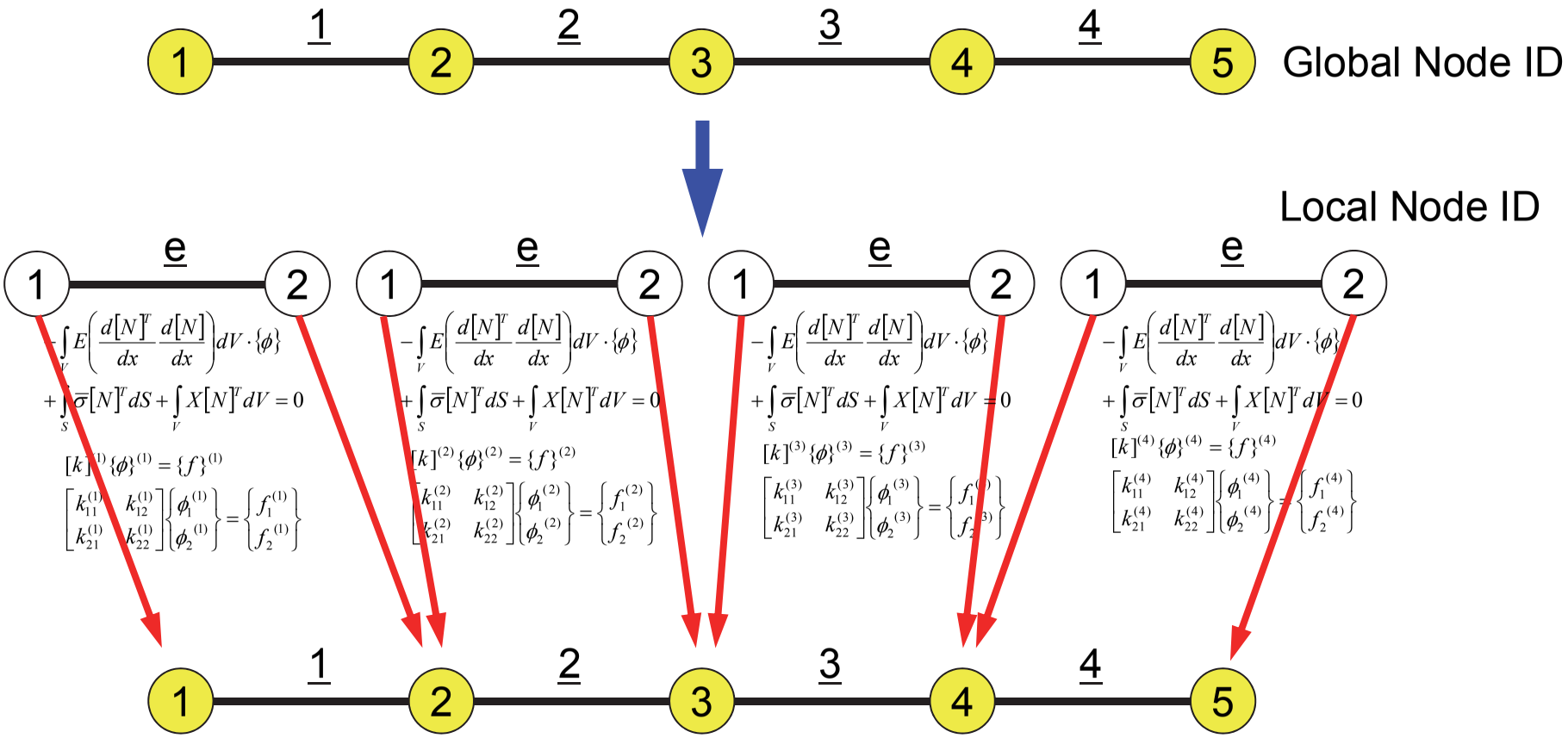
NE: # Element
N : # Node (NE+1)

Program: 1d.c (2/7)
Initialization, Allocation of Arrays

```
/*
// +-------+
// | INIT. |
// +-------+
*/
          fp = fopen("input.dat", "r");
          assert(fp != NULL);
          fscanf(fp, "%d", &NE);
          fscanf(fp, "%lf %lf %lf %lf", &dX, &F, &Area, &Young);
          fscanf(fp, "%d", &IterMax);
          fscanf(fp, "%lf", &Eps);
          fclose(fp);

          N= NE + 1;

          U    = calloc(N, sizeof(double));
          X    = calloc(N, sizeof(double));
          Diag = calloc(N, sizeof(double));

          AMat = calloc(2*N-2, sizeof(double));

          Rhs = calloc(N, sizeof(double));
          Index= calloc(N+1, sizeof(int));
          Item = calloc(2*N-2, sizeof(int));

          Icelnod= calloc(2*NE, sizeof(int));
```

Amat: Non-Zero Off-Diag. Comp.
Item: Corresponding Column ID

Element/Global Operations



$$[K]\{\Phi\}=\{F\}$$

$$\begin{bmatrix} D_1 & AU_{11} & & & \\ AL_{21} & D_2 & AU_{21} & & \\ & AL_{31} & D_3 & AU_{31} & \\ & & AL_{41} & D_4 & AU_{41} \\ & & & AL_{51} & D_5 \end{bmatrix}\begin{Bmatrix}\Phi_1\\ \Phi_2\\ \Phi_3\\ \Phi_4\\ \Phi_5\end{Bmatrix}=\begin{Bmatrix}B_1\\ B_2\\ B_3\\ B_4\\ B_5\end{Bmatrix}$$

Number of non-zero off-diag. components is 2 for each node. This number is 1 at boundary nodes).

Program: 1d.c (2/7)
Initialization, Allocation of Arrays

```
/*
// +-------+
// | INIT. |
// +-------+
*/
          fp = fopen("input.dat", "r");
          assert(fp != NULL);
          fscanf(fp, "%d", &NE);
          fscanf(fp, "%lf %lf %lf %lf", &dX, &F, &Area, &Young);
          fscanf(fp, "%d", &IterMax);
          fscanf(fp, "%lf", &Eps);
          fclose(fp);

          N= NE + 1;

          U    = calloc(N, sizeof(double));
          X    = calloc(N, sizeof(double));
          Diag = calloc(N, sizeof(double));

          AMat = calloc(2*N-2, sizeof(double));

          Rhs = calloc(N, sizeof(double));
          Index= calloc(N+1, sizeof(int));
          Item = calloc(2*N-2, sizeof(int));

          Icelnod= calloc(2*NE, sizeof(int));
```

Amat: Non-Zero Off-Diag. Comp.
Item: Corresponding Column ID

Number of non-zero off-diag. components is 2 for each node. This number is 1 at boundary nodes).

Total Number of Non-Zero Off-Diag. Components:

2*(N-2)+1+1= 2*N-2

Program: 1d.c (3/7)

Initialization, Allocation of Arrays (cont.)

```
W = (double **)malloc(sizeof(double *)*4);
if(W == NULL) {
        fprintf(stderr, "Error: %s¥n", strerror(errno));
        return -1;
}
for(i=0; i<4; i++) {
        W[i] = (double *)malloc(sizeof(double)*N);
        if(W[i] == NULL) {
                fprintf(stderr, "Error: %s¥n", strerror(errno));
                return -1;
        }
}

  for(i=0;i<N;i++)    U[i] = 0.0;
  for(i=0;i<N;i++) Diag[i] = 0.0;
  for(i=0;i<N;i++)  Rhs[i] = 0.0;
  for(k=0;k<2*N-2;k++)  AMat[k] = 0.0;
  for(i=0;i<N;i++) X[i]= i*dX;
  for(icel=0;icel<NE;icel++){
            Icelnod[2*icel  ]= icel;
            Icelnod[2*icel+1]= icel+1;
  }

Kmat[0][0]= +1.0;
Kmat[0][1]= -1.0;
Kmat[1][0]= -1.0;
Kmat[1][1]= +1.0;
```

Program: 1d.c (3/7)
Initialization, Allocation of Arrays (cont.)

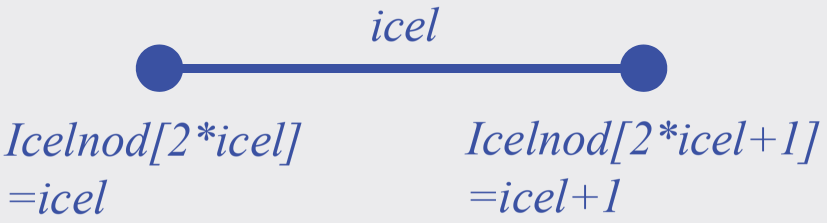
```
W = (double **)malloc(sizeof(double *)*4);
if(W == NULL) {
        fprintf(stderr, "Error: %s¥n", strerror(errno));
        return -1;
}
for(i=0; i<4; i++) {
        W[i] = (double *)malloc(sizeof(double)*N);
        if(W[i] == NULL) {
                fprintf(stderr, "Error: %s¥n", strerror(errno));
                return -1;
        }
}

  for(i=0;i<N;i++)     U[i] = 0.0;
  for(i=0;i<N;i++) Diag[i] = 0.0;
  for(i=0;i<N;i++)  Rhs[i] = 0.0;
  for(k=0;k<2*N-2;k++)  AMat[k] = 0.0;
  for(i=0;i<N;i++) X[i]= i*dX;
  for(icel=0;icel<NE;icel++){
            Icelnod[2*icel  ]= icel;
            Icelnod[2*icel+1]= icel+1;
  }

Kmat[0][0]= +1.0;
Kmat[0][1]= -1.0;
Kmat[1][0]= -1.0;
Kmat[1][1]= +1.0;
```

x: X-coordinate component of each node

Program: 1d.c (3/7)

Initialization, Allocation of Arrays (cont.)

```
W = (double **)malloc(sizeof(double *)*4);
if(W == NULL) {
        fprintf(stderr, "Error: %s¥n", strerror(errno));
        return -1;
}
for(i=0; i<4; i++) {
        W[i] = (double *)malloc(sizeof(double)*N);
        if(W[i] == NULL) {
                fprintf(stderr, "Error: %s¥n", strerror(errno));
                return -1;
        }
}

  for(i=0;i<N;i++)    U[i] = 0.0;
  for(i=0;i<N;i++) Diag[i] = 0.0;
  for(i=0;i<N;i++)  Rhs[i] = 0.0;
  for(k=0;k<2*N-2;k++)  AMat[k] = 0.0;
  for(i=0;i<N;i++) X[i]= i*dX;
  for(icel=0;icel<NE;icel++){
            Icelnod[2*icel  ]= icel;
            Icelnod[2*icel+1]= icel+1;
  }

Kmat[0][0]= +1.0;
Kmat[0][1]= -1.0;
Kmat[1][0]= -1.0;
Kmat[1][1]= +1.0;
```



Program: 1d.c (3/7)

Initialization, Allocation of Arrays (cont.)

```
W = (double **)malloc(sizeof(double *)*4);
if(W == NULL) {
        fprintf(stderr, "Error: %s¥n", strerror(errno));
        return -1;
}
for(i=0; i<4; i++) {
        W[i] = (double *)malloc(sizeof(double)*N);
        if(W[i] == NULL) {
                fprintf(stderr, "Error: %s¥n", strerror(errno));
                return -1;
        }
}

  for(i=0;i<N;i++)    U[i] = 0.0;
  for(i=0;i<N;i++) Diag[i] = 0.0;
  for(i=0;i<N;i++)  Rhs[i] = 0.0;
  for(k=0;k<2*N-2;k++)  AMat[k] = 0.0;
  for(i=0;i<N;i++) X[i]= i*dX;
  for(icel=0;icel<NE;icel++){
            Icelnod[2*icel  ]= icel;
            Icelnod[2*icel+1]= icel+1;
  }

Kmat[0][0]= +1.0;
Kmat[0][1]= -1.0;
Kmat[1][0]= -1.0;
Kmat[1][1]= +1.0;
```

$$[k]^{(e)} = \int_V E\left(\frac{d[N]^T}{dx}\frac{d[N]}{dx}\right)dV = \frac{EA}{L}\begin{bmatrix} +1 & -1 \\ -1 & +1 \end{bmatrix}$$

[Kmat]

Program: 1d.c (4/7)

Global Matrix: Column ID for Non-Zero Off-Diag's

```
/*
// +--------------+
// | CONNECTIVITY |
// +--------------+
*/
        for(i=0;i<N+1;i++)  Index[i] = 2;
        Index[0]= 0;
        Index[1]= 1;
        Index[N]= 1;

        for(i=0;i<N;i++){
                Index[i+1]= Index[i+1] + Index[i];
        }

        NPLU= Index[N];

        for(i=0;i<N;i++){
                jS = Index[i];
                if(i == 0){
                        Item[jS] = i+1;
                }else if(i == N-1){
                        Item[jS] = i-1;
                }else{
                        Item[jS] = i-1;
                        Item[jS+1] = i+1;
                }
        }
```

Number of non-zero off-diag. components is 2 for each node. This number is 1 at boundary nodes).

Total Number of Non-Zero Off-Diag. Components:
2*(N-2)+1+1= 2*N-2= NPLU= Index[N]

						# Non-Zero Off-Diag.	
0	1.1 ◎	2.4 ①	3.2 ④			2	Index[0]= 0
1	3.6 ①	4.3 ◎	2.5 ③	3.7 ⑤	9.1 ⑦	4	Index[1]= 2
2	5.7 ②	1.5 ④	3.1 ⑥			2	Index[2]= 6
3	9.8 ③	4.1 ①	2.5 ④	2.7 ⑤		3	Index[3]= 8
4	11.5 ④	3.1 ◎	9.5 ①	10.4 ②	4.3 ⑥	4	Index[4]= 11
5	12.4 ⑤	6.5 ②	9.5 ⑥			2	Index[5]= 15
6	23.1 ⑥	6.4 ①	2.5 ②	1.4 ⑤	13.1 ⑦	4	Index[6]= 17
7	51.3 ⑦	9.5 ①	1.3 ②	9.6 ③	3.1 ⑤	4	Index[7]= 21
							Index[8]= 25

(Index[i])th~(Index[i+1])th:
Non-Zero Off-Diag. Components corresponding to *i*-th row.

Program: 1d.c (4/7)

Global Matrix: Column ID for Non-Zero Off-Diag's

```
/*
// +--------------+
// | CONNECTIVITY |
// +--------------+
*/
        for(i=0;i<N+1;i++)  Index[i] = 2;
        Index[0]= 0;
        Index[1]= 1;
        Index[N]= 1;

        for(i=0;i<N;i++){
                Index[i+1]= Index[i+1] + Index[i];
        }

        NPLU= Index[N];

        for(i=0;i<N;i++){
                jS = Index[i];
                if(i == 0){
                        Item[jS] = i+1;
                }else if(i == N-1){
                        Item[jS] = i-1;
                }else{
                        Item[jS]   = i-1;
                        Item[jS+1] = i+1;
                }
        }
```

						# Non-Zero Off-Diag.	
0	1.1 ◎	2.4 ①	3.2 ④			2	Index[0]= 0
1	3.6 ①	4.3 ◎	2.5 ③	3.7 ⑤	9.1 ⑦	4	Index[1]= 2
2	5.7 ②	1.5 ④	3.1 ⑥			2	Index[2]= 6
3	9.8 ③	4.1 ①	2.5 ④	2.7 ⑤		3	Index[3]= 8
4	11.5 ④	3.1 ◎	9.5 ①	10.4 ②	4.3 ⑥	4	Index[4]= 11
5	12.4 ⑤	6.5 ②	9.5 ⑥			2	Index[5]= 15
6	23.1 ⑥	6.4 ①	2.5 ②	1.4 ⑤	13.1 ⑦	4	Index[6]= 17
7	51.3 ⑦	9.5 ①	1.3 ②	9.6 ③	3.1 ⑤	4	Index[7]= 21
							Index[8]= 25

(Index[i])th~(Index[i+1])th:
Non-Zero Off-Diag. Components corresponding to *i*-th row.

Program: 1d.c (5/7)
Element Matrix ~ Global Matrix

```
/*
// +-----------------+
// | MATRIX assemble |
// +-----------------+
*/
    for(icel=0;icel<NE;icel++){
        in1= Icelnod[2*icel];
        in2= Icelnod[2*icel+1];
        X1 = X[in1];
        X2 = X[in2];
        DL = fabs(X2-X1);

          Ck= Area*Young/DL;
          Emat[0][0]= Ck*Kmat[0][0];
          Emat[0][1]= Ck*Kmat[0][1];
          Emat[1][0]= Ck*Kmat[1][0];
          Emat[1][1]= Ck*Kmat[1][1];

          Diag[in1]= Diag[in1] + Emat[0][0];
          Diag[in2]= Diag[in2] + Emat[1][1];

              if (icel==0){k1=Index[in1];
                    }else {k1=Index[in1]+1;}
              k2=Index[in2];

              AMat[k1]= AMat[k1] + Emat[0][1];
              AMat[k2]= AMat[k2] + Emat[1][0];
          }
```

Program: 1d.c (5/7)
Element Matrix ~ Global Matrix

```
/*
// +-----------------+
// | MATRIX assemble |
// +-----------------+
*/
    for(icel=0;icel<NE;icel++){
        in1= Icelnod[2*icel];
        in2= Icelnod[2*icel+1];
        X1 = X[in1];
        X2 = X[in2];
        DL = fabs(X2-X1);

          Ck= Area*Young/DL;
          Emat[0][0]= Ck*Kmat[0][0];
          Emat[0][1]= Ck*Kmat[0][1];
          Emat[1][0]= Ck*Kmat[1][0];
          Emat[1][1]= Ck*Kmat[1][1];

          Diag[in1]= Diag[in1] + Emat[0][0];
          Diag[in2]= Diag[in2] + Emat[1][1];

              if (icel==0){k1=Index[in1];
                    }else {k1=Index[in1]+1;}
              k2=Index[in2];

              AMat[k1]= AMat[k1] + Emat[0][1];
              AMat[k2]= AMat[k2] + Emat[1][0];
          }
```

$$[Emat]=[k]^{(e)}=\frac{EA}{L}\begin{bmatrix}+1 & -1\\ -1 & +1\end{bmatrix}=\frac{EA}{L}[Kmat]$$

Program: 1d.c (5/7)
Element Matrix ~ Global Matrix

```
/*
// +-----------------+
// | MATRIX assemble |
// +-----------------+
*/
    for(icel=0;icel<NE;icel++){
        in1= Icelnod[2*icel];
        in2= Icelnod[2*icel+1];
        X1 = X[in1];
        X2 = X[in2];
        DL = fabs(X2-X1);

          Ck= Area*Young/DL;
          Emat[0][0]= Ck*Kmat[0][0];
          Emat[0][1]= Ck*Kmat[0][1];
          Emat[1][0]= Ck*Kmat[1][0];
          Emat[1][1]= Ck*Kmat[1][1];

          Diag[in1]= Diag[in1] + Emat[0][0];
          Diag[in2]= Diag[in2] + Emat[1][1];

              if (icel==0){k1=Index[in1];
                    }else {k1=Index[in1]+1;}
                k2=Index[in2];

              AMat[k1]= AMat[k1] + Emat[0][1];
              AMat[k2]= AMat[k2] + Emat[1][0];
          }
```

$$[Emat] = [k]^{(e)} = \frac{EA}{L}\begin{bmatrix} +1 & -1 \\ -1 & +1 \end{bmatrix}$$

Program: 1d.c (5/7)

Element Matrix ~ Global Matrix

```
/*
// +-----------------+
// | MATRIX assemble |
// +-----------------+
*/
    for(icel=0;icel<NE;icel++){
        in1= Icelnod[2*icel];
        in2= Icelnod[2*icel+1];
        X1 = X[in1];
        X2 = X[in2];
        DL = fabs(X2-X1);

          Ck= Area*Young/DL;
          Emat[0][0]= Ck*Kmat[0][0];
          Emat[0][1]= Ck*Kmat[0][1];
          Emat[1][0]= Ck*Kmat[1][0];
          Emat[1][1]= Ck*Kmat[1][1];

          Diag[in1]= Diag[in1] + Emat[0][0];
          Diag[in2]= Diag[in2] + Emat[1][1];

              if (icel==0){k1=Index[in1];
                   }else {k1=Index[in1]+1;}
                          k2=Index[in2];

              AMat[k1]= AMat[k1] + Emat[0][1];
              AMat[k2]= AMat[k2] + Emat[1][0];
          }
```

Non-zero Off-Diag. at i-th row:

`Index[i], Index[i]+1`

$$[Emat] = [k]^{(e)} = \frac{EA}{L}\begin{bmatrix} +1 & -1 \\ -1 & +1 \end{bmatrix}$$

k1

k2

Program: 1d.c (6/7)
Boundary Conditions

```
/*
// +--------------------+
// | BOUNDARY conditions |
// +--------------------+
*/

/* X=Xmin */
        i=0;
        jS= Index[i];
        AMat[jS]= 0.0;
        Diag[i ]= 1.0;
        Rhs [i ]= 0.0;

          for(k=0;k<NPLU;k++){
            if(Item[k]==0){AMat[k]=0.0;
            }}

/* X=Xmax */
        i=N-1;
        Rhs[i]= F;
```

1D Static Linear Elastic Problem

- Only deforms in x-direction (displacement: u)
  - Uniform: Sectional Area A, Young's Modulus E
  - Boundary Conditions (B.C.)
    - x=0 : u=0 (fixed)
    - x=x_{max} : F (axial force)
- Truss: NO bending deformation by G-force

(Linear) Equation at x=0

$u_1 = 0$ (or $u_0 = 0$)

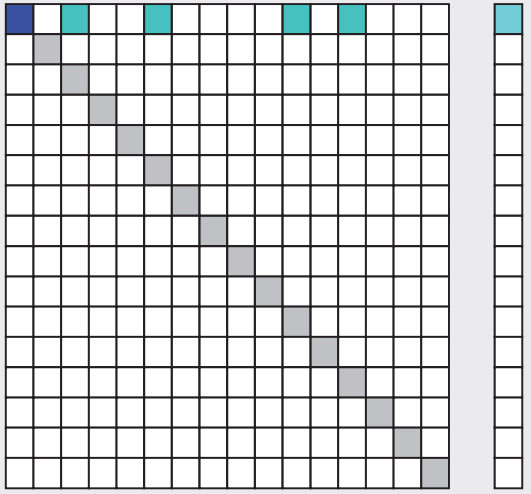
- Only deforms in x-direction (displacement: u)
  - Uniform: Sectional Area A, Young's Modulus E
  - Boundary Conditions (B.C.)
    - x=0 : u=0 (fixed)
    - x=x_{max} : F (axial force)
- Truss: NO bending deformation by G-force

Program: 1d.c (6/7)
Boundary Conditions

```
/*
// +--------------------+
// | BOUNDARY conditions |
// +--------------------+
*/

/* X=Xmin */
        i=0;
        jS= Index[i];
        AMat[jS]= 0.0;
        Diag[i ]= 1.0;
        Rhs [i ]= 0.0;

          for(k=0;k<NPLU;k++){
            if(Item[k]==0){AMat[k]=0.0;
            }}

/* X=Xmax */
        i=N-1;
        Rhs[i]= F;
```

$u_1=0$

Diagonal Component=1
RHS=0
Off-Diagonal Components= 0.



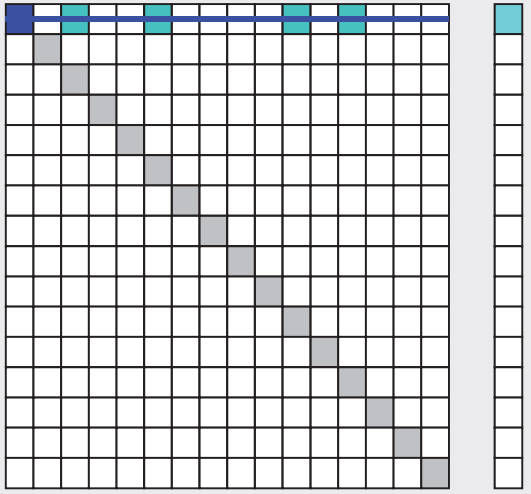
Program: 1d.c (6/7)
Boundary Conditions

```
/*
// +--------------------+
// | BOUNDARY conditions |
// +--------------------+
*/

/* X=Xmin */
        i=0;
        jS= Index[i];
        AMat[jS]= 0.0;
        Diag[i ]= 1.0;
        Rhs [i ]= 0.0;

          for(k=0;k<NPLU;k++){
            if(Item[k]==0){AMat[k]=0.0;
            }}

/* X=Xmax */
        i=N-1;
        Rhs[i]= F;
```

$u_1=0$

Diagonal Component=1
RHS=0
Off-Diagonal Components= 0.

Erase !



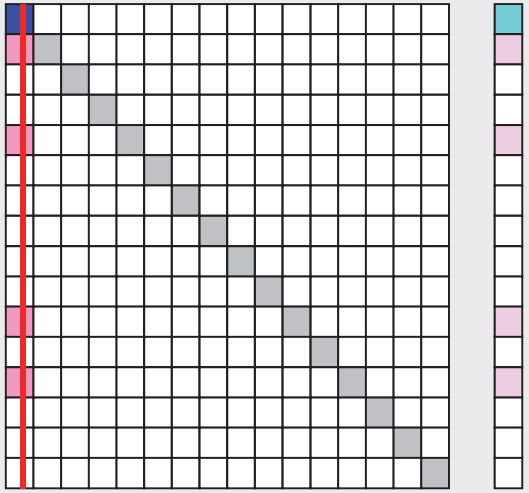
Program: 1d.c (6/7)
Boundary Conditions

```
/*
// +--------------------+
// | BOUNDARY conditions |
// +--------------------+
*/

/* X=Xmin */
        i=0;
        jS= Index[i];
        AMat[jS]= 0.0;
        Diag[i ]= 1.0;
        Rhs [i ]= 0.0;

          for(k=0;k<NPLU;k++){
            if(Item[k]==0){AMat[k]=0.0;
            }}
/* X=Xmax */
        i=N-1;
        Rhs[i]= F;
```

$u_1=0$

Diagonal Component=1
RHS=0
Off-Diagonal Components= 0.

Elimination and Erase



Column components of boundary nodes (Dirichlet B.C.) are moved to RHS and eliminated for keeping symmetrical feature of the matrix (in this case just erase off-diagonal components)

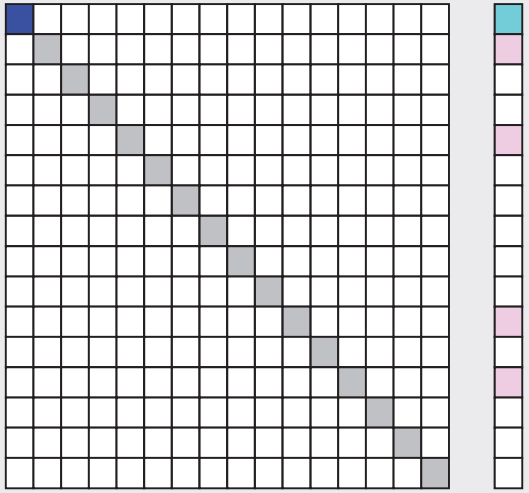
Program: 1d.c (6/7)
Boundary Conditions

```
/*
// +--------------------+
// | BOUNDARY conditions |
// +--------------------+
*/

/* X=Xmin */
        i=0;
        jS= Index[i];
        AMat[jS]= 0.0;
        Diag[i ]= 1.0;
        Rhs [i ]= 0.0;

          for(k=0;k<NPLU;k++){
            if(Item[k]==0){AMat[k]=0.0;
            }}
/* X=Xmax */
        i=N-1;
        Rhs[i]= F;
```

$u_1=0$

Diagonal Component=1
RHS=0
Off-Diagonal Components= 0.

Elimination and Erase



Column components of boundary nodes (Dirichlet B.C.) are moved to RHS and eliminated for keeping symmetrical feature of the matrix (in this case just erase off-diagonal components)

if $u_1 \neq 0$

```
/*
// +--------------------+
// | BOUNDARY conditions |
// +--------------------+
*/

/* X=Xmin */
        i=0;
        jS= Index[i];
        AMat[jS]= 0.0;
        Diag[i ]= 1.0;
        Rhs [i ]= Umin;

          for(j=1;i<N;i++){
            for (k=Index[j];k<Index[j+1];k++){
              if(Item[k]==0){
                Rhs [j]= Rhs[j] - Amat[k]*Umin;
                AMat[k]= 0.0;
            }}
```

Column components of boundary nodes (Dirichlet B.C.) are moved to RHS and eliminated for keeping symmetrical feature of the matrix.

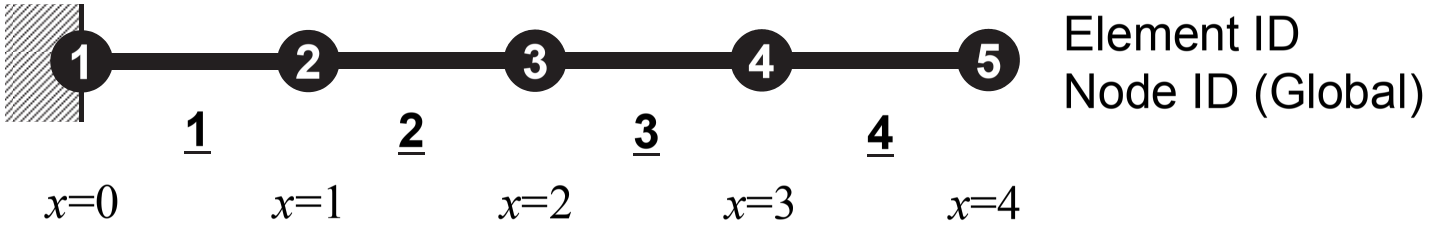
$$Diag_j u_j + \sum_{k=Index[j],k\neq k_s}^{Index[j-1]} Amat_k \, u_{Item[k]}$$

$$= Rhs_j - Amat_{k_s} u_{Item[k_s]} = Rhs_j - Amat_{k_s} u_{\min} \quad where\ Item[k_s] = 0$$

Program: 1d.c (6/7)
Boundary Conditions

```
/*
// +--------------------+
// | BOUNDARY conditions |
// +--------------------+
*/

/* X=Xmin */
        i=0;
        jS= Index[i];
        AMat[jS]= 0.0;
        Diag[i ]= 1.0;
        Rhs [i ]= 0.0;

          for(k=0;k<NPLU;k++){
            if(Item[k]==0){AMat[k]=0.0;
            }}

/* X=Xmax */
        i= N-1;
        Rhs[i]= F;
```

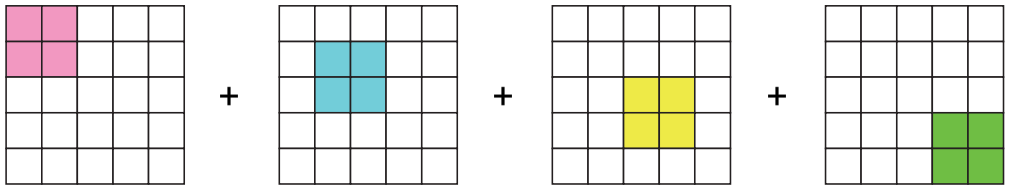
Element Eqn's/Accumulation



- As for Element-4:

$$[k]^{(4)} = \frac{EA}{L}\begin{bmatrix} +1 & -1 \\ -1 & +1 \end{bmatrix} \qquad \{f\}^{(4)} = \int_S \bar{\sigma}[N]^T dS = \bar{\sigma}A\begin{Bmatrix} 0 \\ 1 \end{Bmatrix} = \begin{Bmatrix} 0 \\ F \end{Bmatrix}$$

$$[K] = \sum_{e=1}^{4} [k]^{(e)} =$$



$$\{F\} = \sum_{e=1}^{4} \{f\}^{(e)} =$$

Preconditioned CG Solver

Compute $r^{(0)}= b-[A]x^{(0)}$
for i= 1, 2, …
  solve $[M]z^{(i-1)}= r^{(i-1)}$
  $\rho_{i-1}= r^{(i-1)}\ z^{(i-1)}$
  if i=1
    $p^{(1)}= z^{(0)}$
  else
    $\beta_{i-1}= \rho_{i-1}/\rho_{i-2}$
    $p^{(i)}= z^{(i-1)} + \beta_{i-1}\ p^{(i-1)}$
  endif
  $q^{(i)}= [A]p^{(i)}$
  $\alpha_i = \rho_{i-1}/p^{(i)}q^{(i)}$
  $x^{(i)}= x^{(i-1)} + \alpha_i p^{(i)}$
  $r^{(i)}= r^{(i-1)} - \alpha_i q^{(i)}$
  check convergence |r|
end

$$[M]=\begin{bmatrix} D_1 & 0 & \cdots & 0 & 0 \\ 0 & D_2 & & 0 & 0 \\ \cdots & & \cdots & & \cdots \\ 0 & 0 & & D_{N-1} & 0 \\ 0 & 0 & \cdots & 0 & D_N \end{bmatrix}$$

Diagonal Scaling, Point-Jacobi

$$[M]=\begin{bmatrix} D_1 & 0 & \dots & 0 & 0 \\ 0 & D_2 & & 0 & 0 \\ \dots & & \dots & & \dots \\ 0 & 0 & & D_{N-1} & 0 \\ 0 & 0 & \dots & 0 & D_N \end{bmatrix}$$

- **`solve [M]z(i-1)= r(i-1)`** is very easy.
- Provides fast convergence for simple problems.
- 1d.f, 1d.c

CG Solver (1/6)

```
/*
// +---------------+
// | CG iterations |
// +---------------+
*/
          R = 0;
          Z = 1;
          Q = 1;
          P = 2;
          DD= 3;

          for(i=0;i<N;i++){
                    W[DD][i]= 1.0 / Diag[i];
          }
```

Reciprocal numbers（逆数）of diagonal components are stored in `W[DD][i]`.
Computational cost for division is usually expensive.

CG Solver (1/6)

```
/*
// +---------------+
// | CG iterations |
// +---------------+
*/
          R = 0;
          Z = 1;
          Q = 1;
          P = 2;
          DD= 3;

          for(i=0;i<N;i++){
                    W[DD][i]= 1.0 / Diag[i];
          }
```

```
W[0][i]= W[R][i]  ⇒ {r}
W[1][i]= W[Z][i]  ⇒ {z}
W[1][i]= W[Q][i]  ⇒ {q}
W[2][i]= W[P][i]  ⇒ {p}
W[3][i]= W[DD][i] ⇒ 1/{D}
```

Compute $r^{(0)}= b-[A]x^{(0)}$
for i= 1, 2, …
  solve $[M]z^{(i-1)}= r^{(i-1)}$
  $\rho_{i-1}= r^{(i-1)}\ z^{(i-1)}$
  if i=1
    $p^{(1)}= z^{(0)}$
  else
    $\beta_{i-1}= \rho_{i-1}/\rho_{i-2}$
    $p^{(i)}= z^{(i-1)} + \beta_{i-1}\ p^{(i-1)}$
  endif
  $q^{(i)}= [A]p^{(i)}$
  $\alpha_i = \rho_{i-1}/p^{(i)}q^{(i)}$
  $x^{(i)}= x^{(i-1)} + \alpha_i p^{(i)}$
  $r^{(i)}= r^{(i-1)} - \alpha_i q^{(i)}$
  check convergence |r|
end

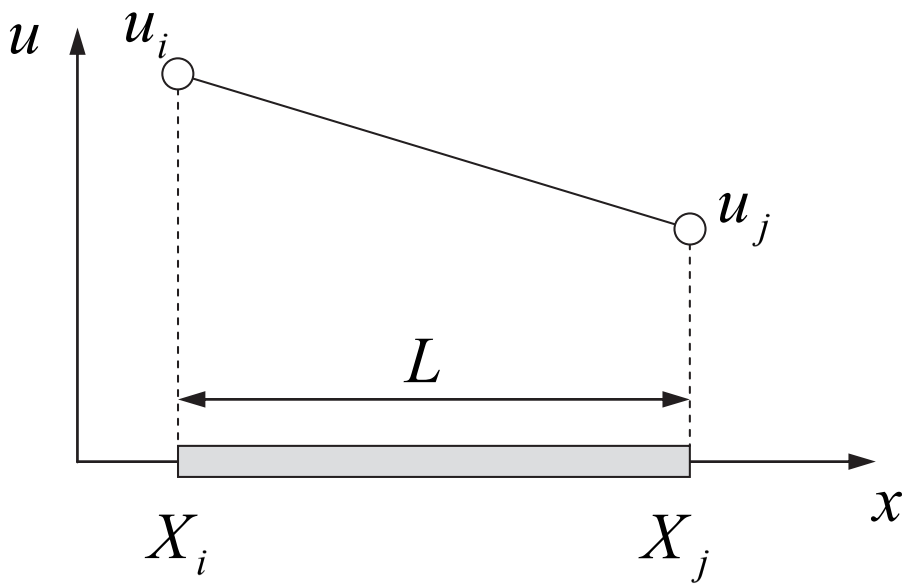
Program: 1d.c (7/7)
Element Stress

```
/*
// +-----------------+
// | STRESS recovery |
// +-----------------+
*/
        printf("¥n%s¥n", "### STRESS");

        for(icel=0;icel<NE;icel++){
            in1= Icelnod[2*icel];
            in2= Icelnod[2*icel+1];
            X1 = X[in1];
            X2 = X[in2];
            U1 = U[in1];
            U2 = U[in2];
           DL = fabs(X2-X1);

          Strain= (U2-U1)/DL;
          Sigma = Young*Strain;

          printf("%8d%16.6E%16.6E¥n", icel+1, Sigma, F/Area);

          }
```

Element Stress

Constant for 1D linear element: Constant Strain

$$N_i = \left(\frac{X_j - x}{L} \right), \quad N_j = \left(\frac{x - X_i}{L} \right) \qquad \frac{dN_i}{dx} = \left(\frac{-1}{L} \right), \quad \frac{dN_j}{dx} = \left(\frac{1}{L} \right)$$



$$u = N_i u_i + N_j u_j$$

$$\varepsilon = \frac{du}{dx} = \frac{d}{dx}\left(N_i u_i + N_j u_j \right)$$

$$= \frac{dN_i}{dx} u_i + \frac{dN_j}{dx} u_j$$

$$= \left(-\frac{1}{L} \right) u_i + \left(\frac{1}{L} \right) u_j = \boxed{\frac{u_j - u_i}{L}}$$

Finite Element Procedures

- Initialization
  - Control Data
  - Node, Connectivity of Elements (N: Node#, NE: Elem#)
  - Initialization of Arrays (Global/Element Matrices)
  - Element-Global Matrix Mapping (Index, Item)
- Generation of Matrix
  - Element-by-Element Operations (do icel= 1, NE)
    - Element matrices
    - Accumulation to global matrix
  - Boundary Conditions
- Linear Solver
  - Conjugate Gradient Method
- Calculation of Stress

- 1D-code for Static Linear-Elastic Problems by Galerkin FEM
- Sparse Linear Solver
  - Conjugate Gradient Method
  - Preconditioning
- Storage of Sparse Matrices

- Program

- **Road to Parallel FEM**

Parallel Computing

- Faster, Larger & More Complicated

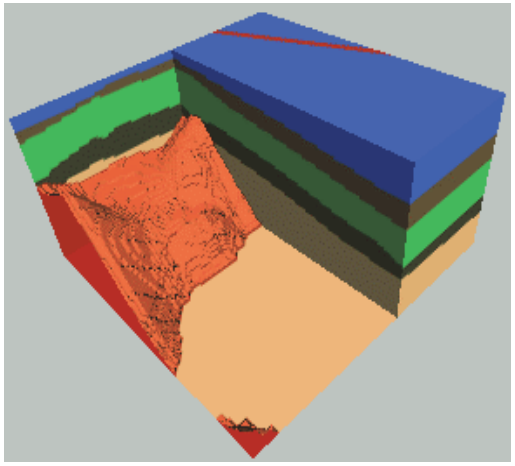
- Scalability
  - Solving N^x scale problem using N^x computational resources during same computation time
    - for large-scale problems: **Weak Scaling**
    - e.g. CG solver: more iterations needed for larger problems
  - Solving a problem using N^x computational resources during 1/N computation time
    - for faster computation: **Strong Scaling**

What is Parallel Computing ? (1/2)

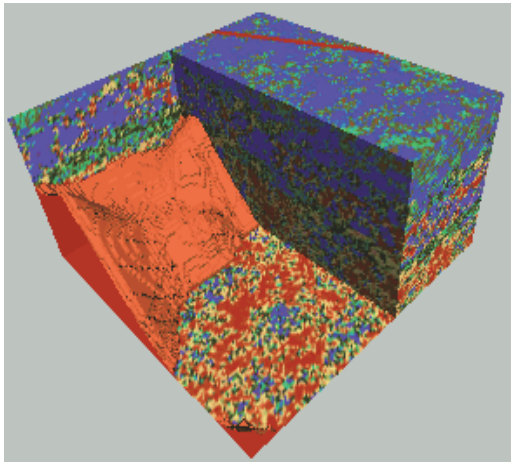
- to solve larger problems faster

Homogeneous/Heterogeneous Porous Media

Lawrence Livermore National Laboratory



Homogeneous

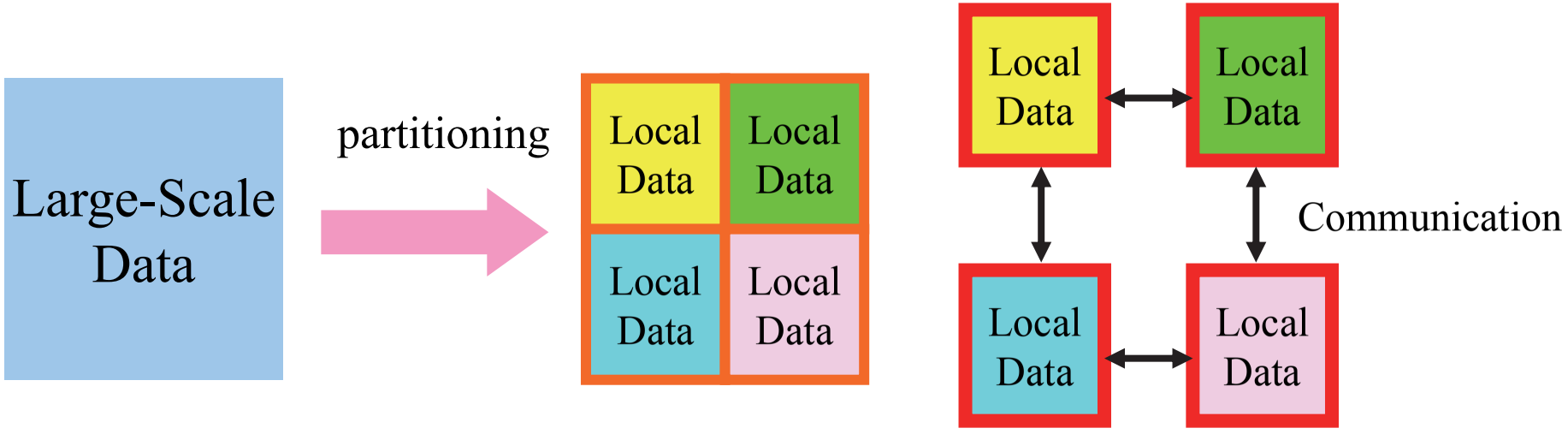


Heterogeneous

very fine meshes are required for simulations of heterogeneous field.

What is Parallel Computing ? (2/2)

- PC with 1GB memory : 1M meshes are the limit for FEM
  - Southwest Japan with 1,000km x 1,000km x 100km in 1km mesh -> 10^8 meshes
- Large Data -> Domain Decomposition -> Local Operation
- Inter-Domain Communication for Global Operation



What is Communication ?

- Parallel Computing -> Local Operations
- Communications are required in Global Operations for Consistency.

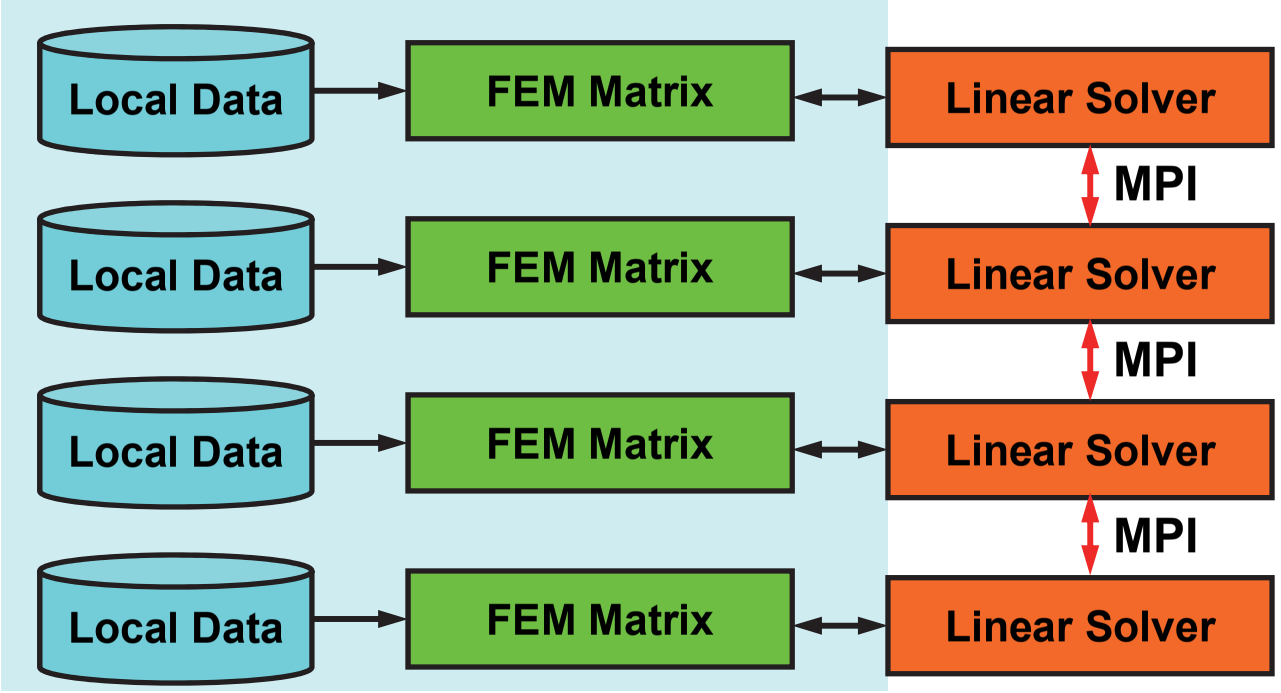
Operations in Parallel FEM

SPMD: Single-Program Multiple-Data

Large Scale Data -> partitioned into Distributed Local Data Sets.

FEM code can assembles coefficient matrix for each local data set : this part could be completely local, same as serial operations

Global Operations & Communications happen only in Linear Solvers dot products, matrix-vector multiply, preconditioning



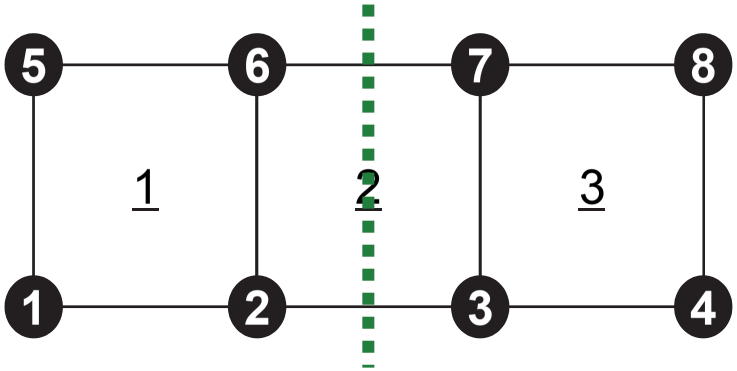
Parallel FEM Procedures

- Design on “Local Data Structure” is important
  - for SPMD-type operations in the previous page

- Matrix Generation
- Preconditioned Iterative Solvers for Linear Equations

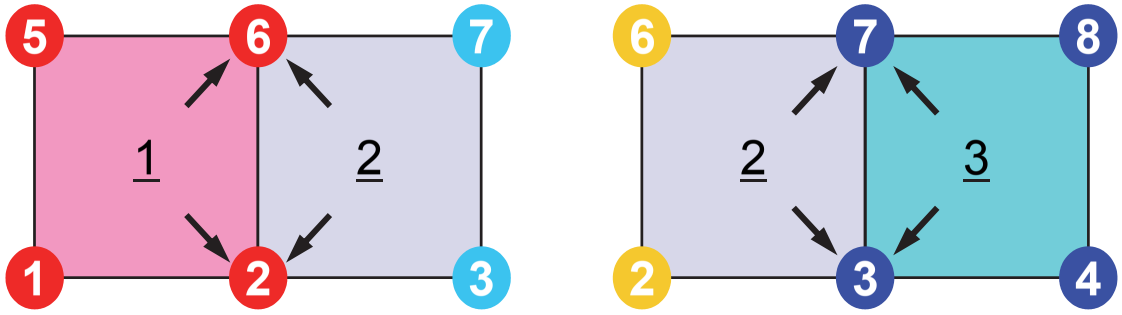
Bi-Linear Square Elements

Values are defined on each node



divide into two domains by "node-based" manner, where number of "nodes (vertices)" are balanced.

Local information is not enough for matrix assembling.



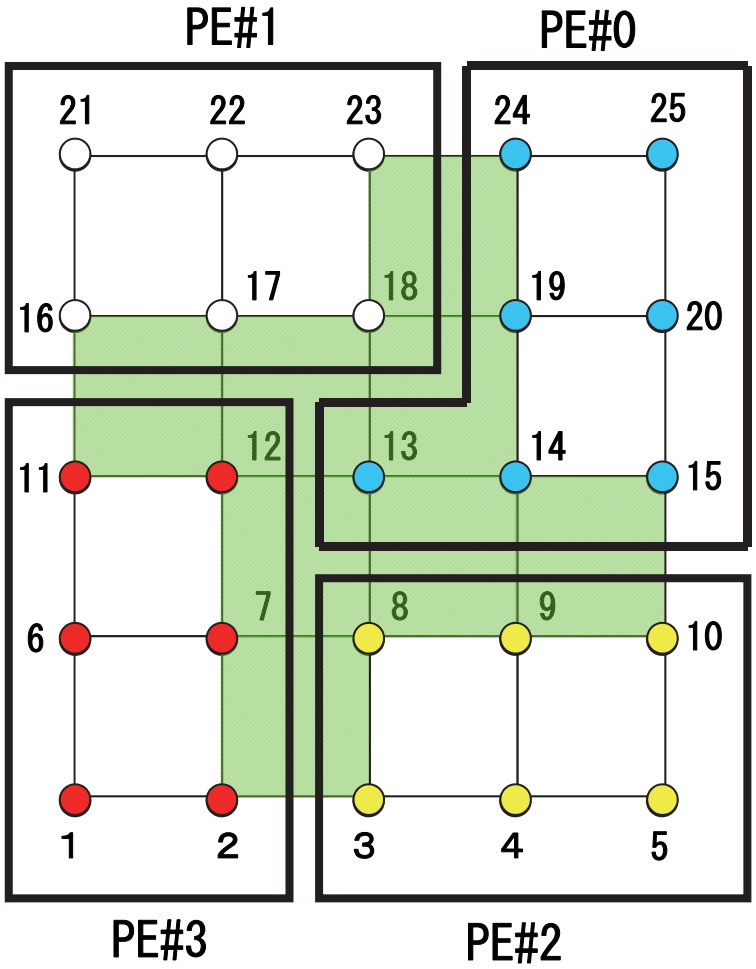
Information of overlapped elements and connected nodes are required for matrix assembling on boundary nodes.

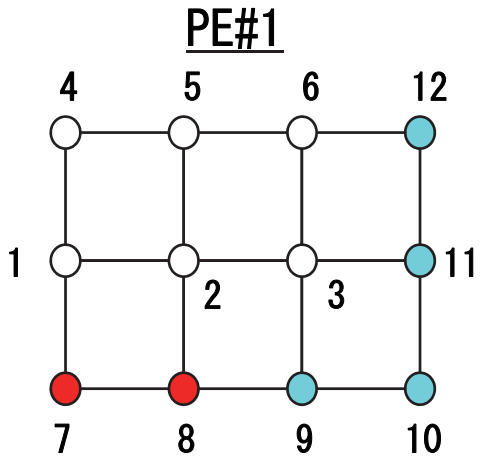
Local Data of Parallel FEM

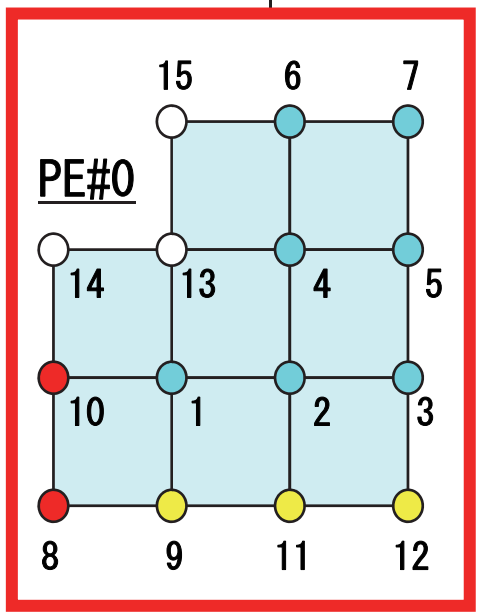
- Node-based partitioning for IC/ILU type preconditioning methods
- Local data includes information for :
  - Nodes originally assigned to the partition/PE
  - Elements which include the nodes : Element-based operations (Matrix Assemble) are allowed for fluid/structure subsystems.
  - All nodes which form the elements but out of the partition
- Nodes are classified into the following 3 categories from the viewpoint of the message passing
  - Internal nodes originally assigned nodes
  - External nodes in the overlapped elements but out of the partition
  - Boundary nodes *external nodes* of other partition
- Communication table between partitions
- NO global information required except partition-to-partition connectivity

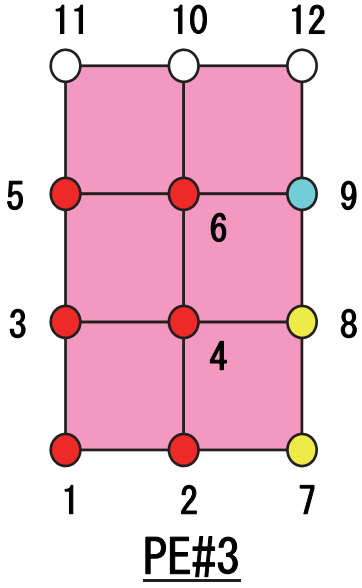
Node-based Partitioning

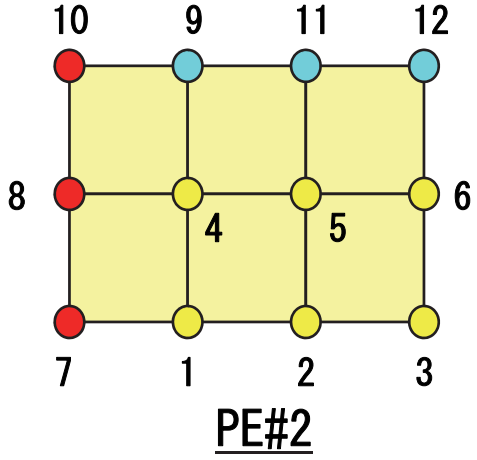
internal nodes - elements - external nodes







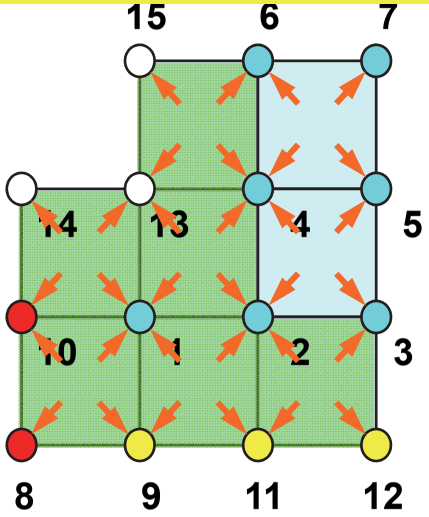


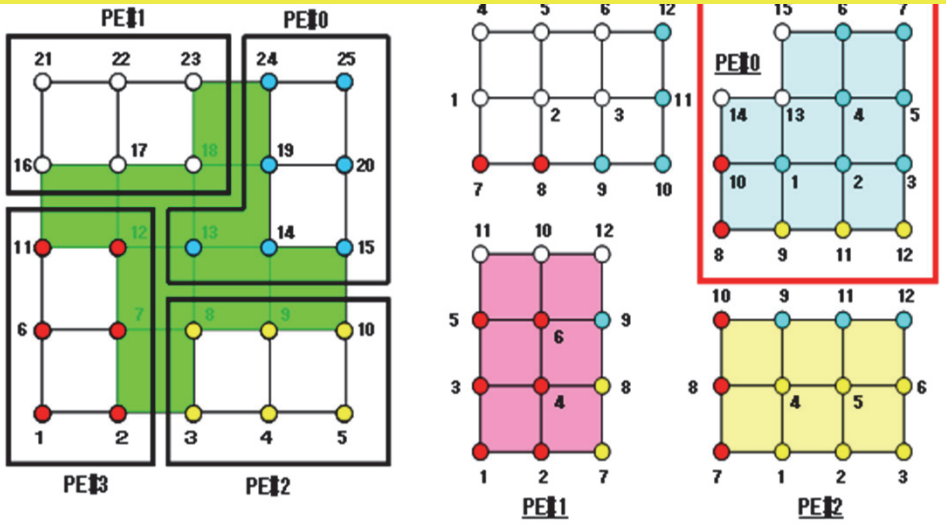


Node-based Partitioning

internal nodes - elements - external nodes

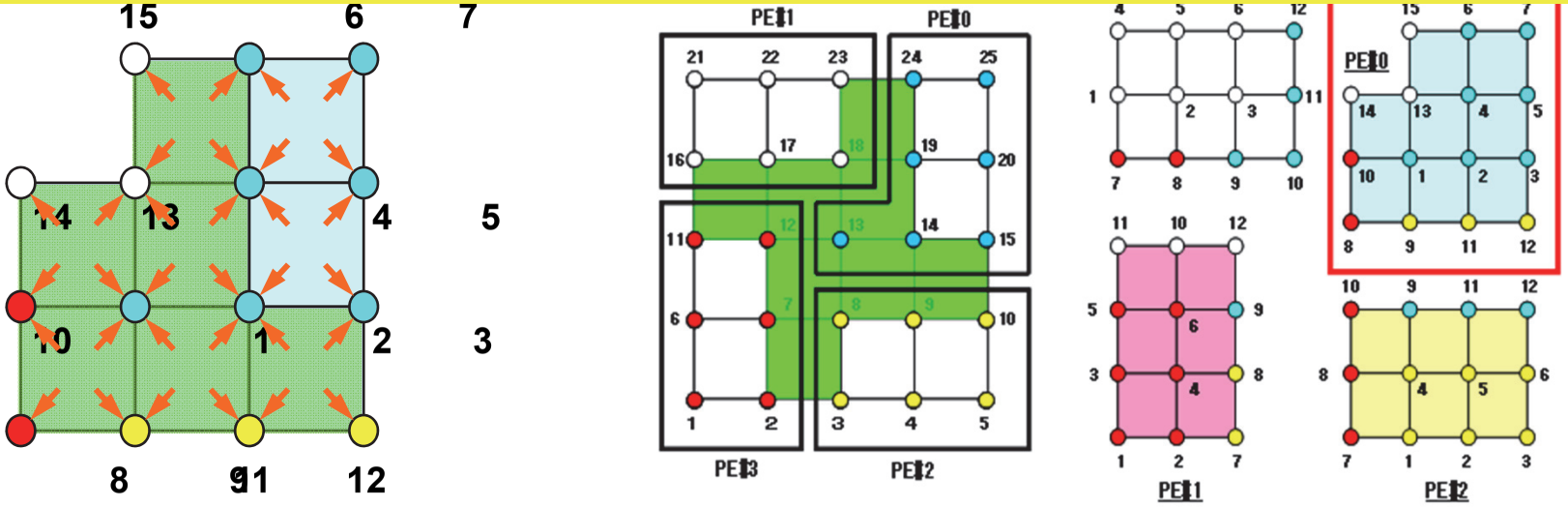
- Partitioned nodes themselves (Internal Nodes) 内点
- Elements which include Internal Nodes 内点を含む要素
- External Nodes included in the Elements 外点
  - in overlapped region among partitions.
- Info of External Nodes are required for completely local element–based operations on each processor.





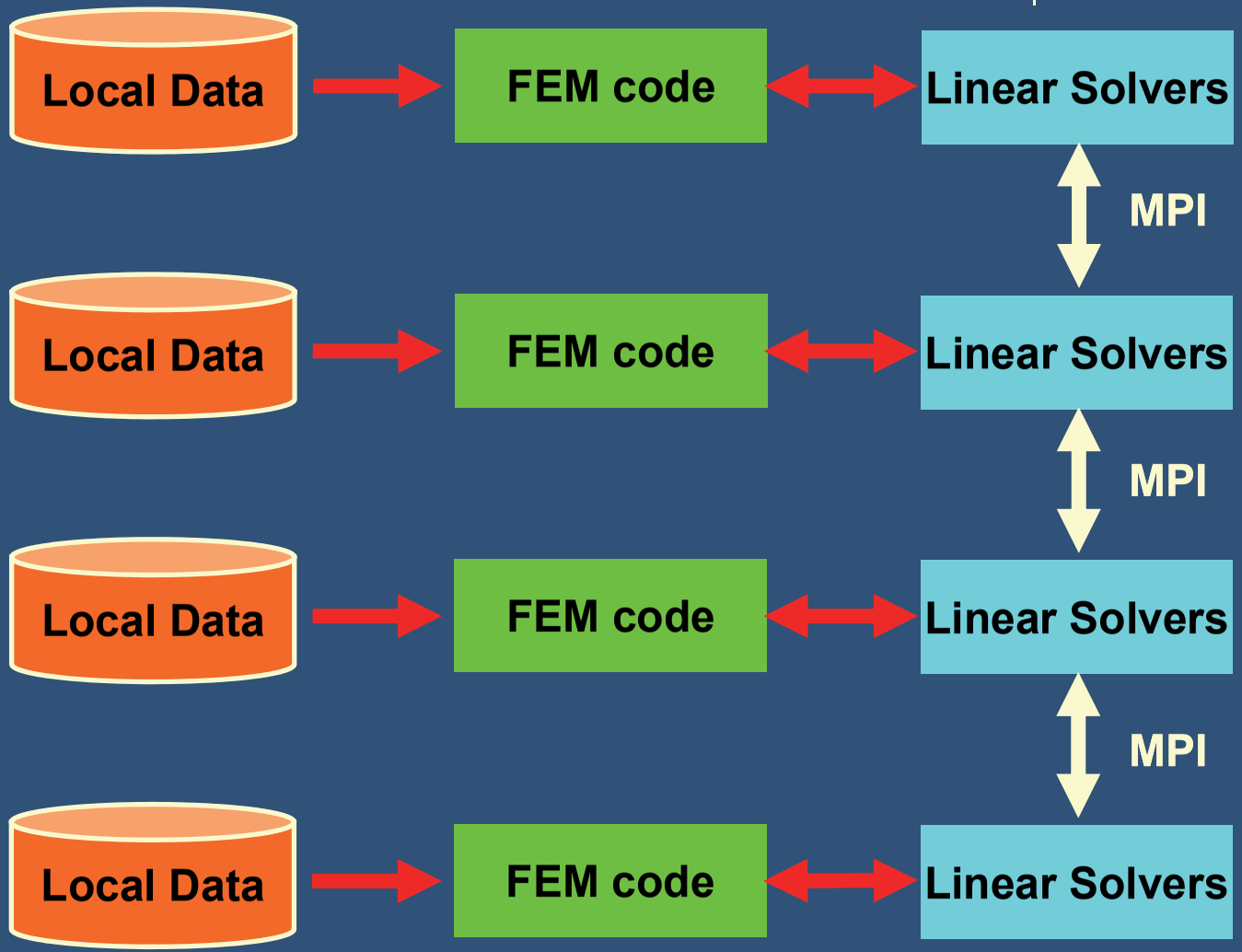
We do not need communication during matrix assemble !!

- Elements which include Internal Nodes
- External Nodes included in the Elements
  in overlapped region among partitions.
- Info of External Nodes are required for completely local element–based operations on each processor.

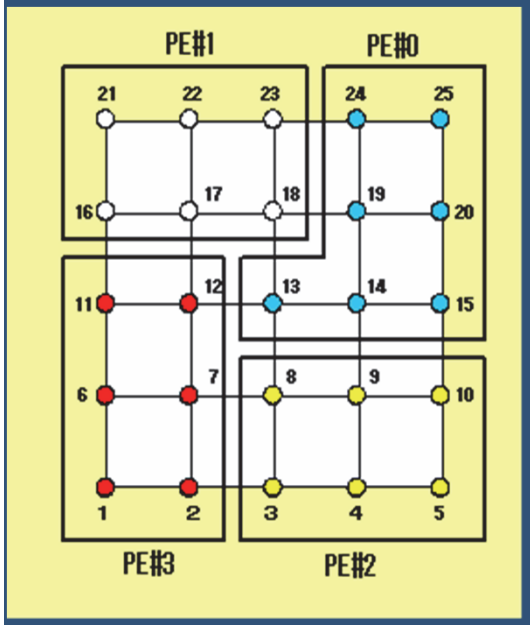


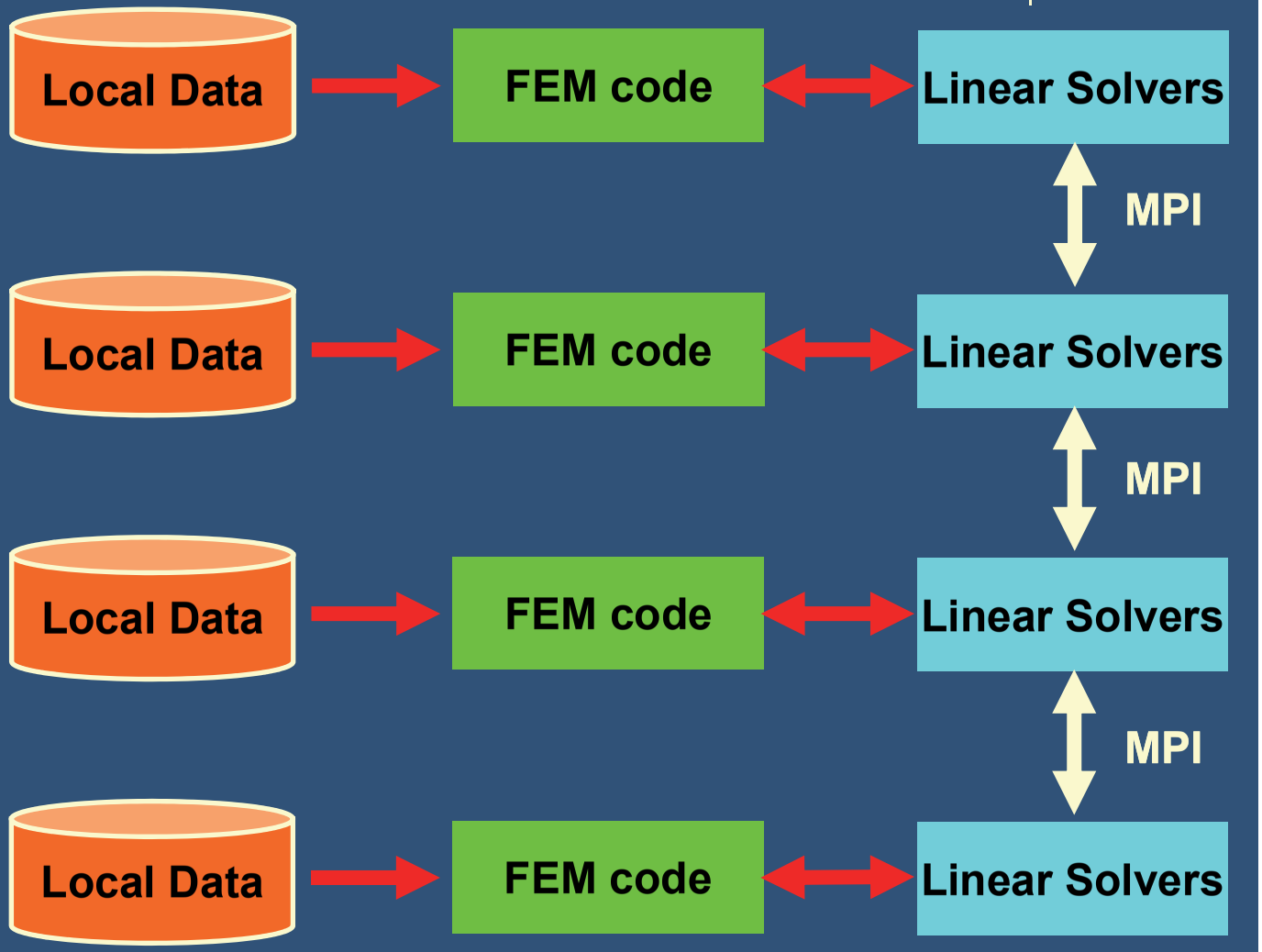
Parallel Computing in FEM

SPMD: Single-Program Multiple-Data



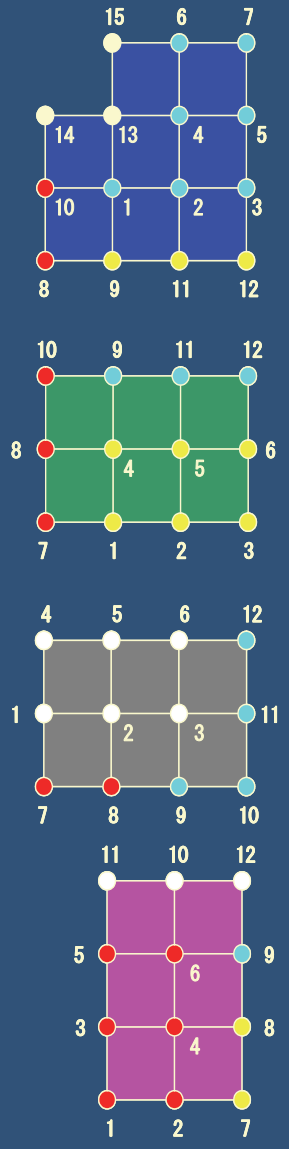
Parallel Computing in FEM
SPMD: Single-Program Multiple-Data

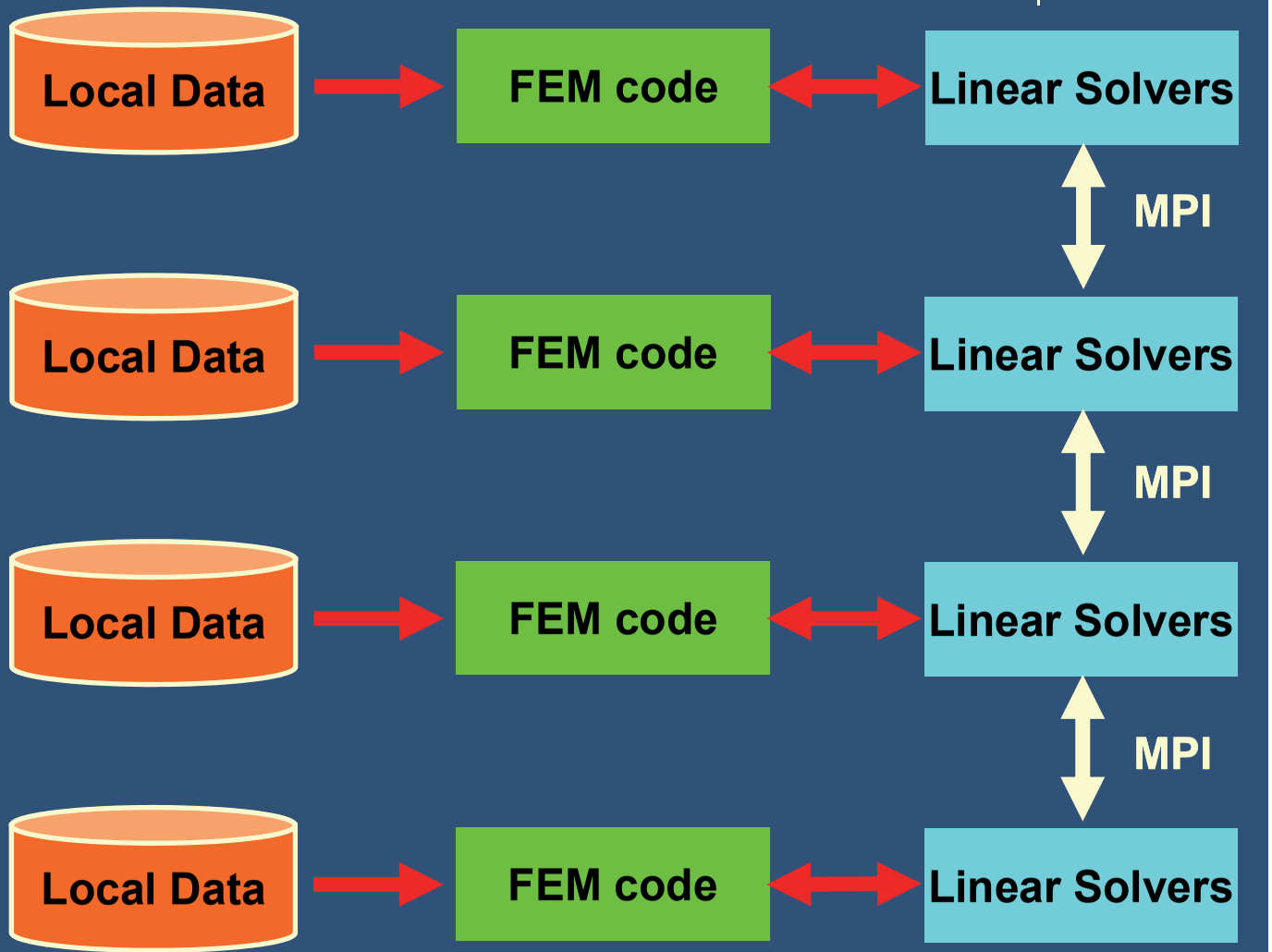




Parallel Computing in FEM

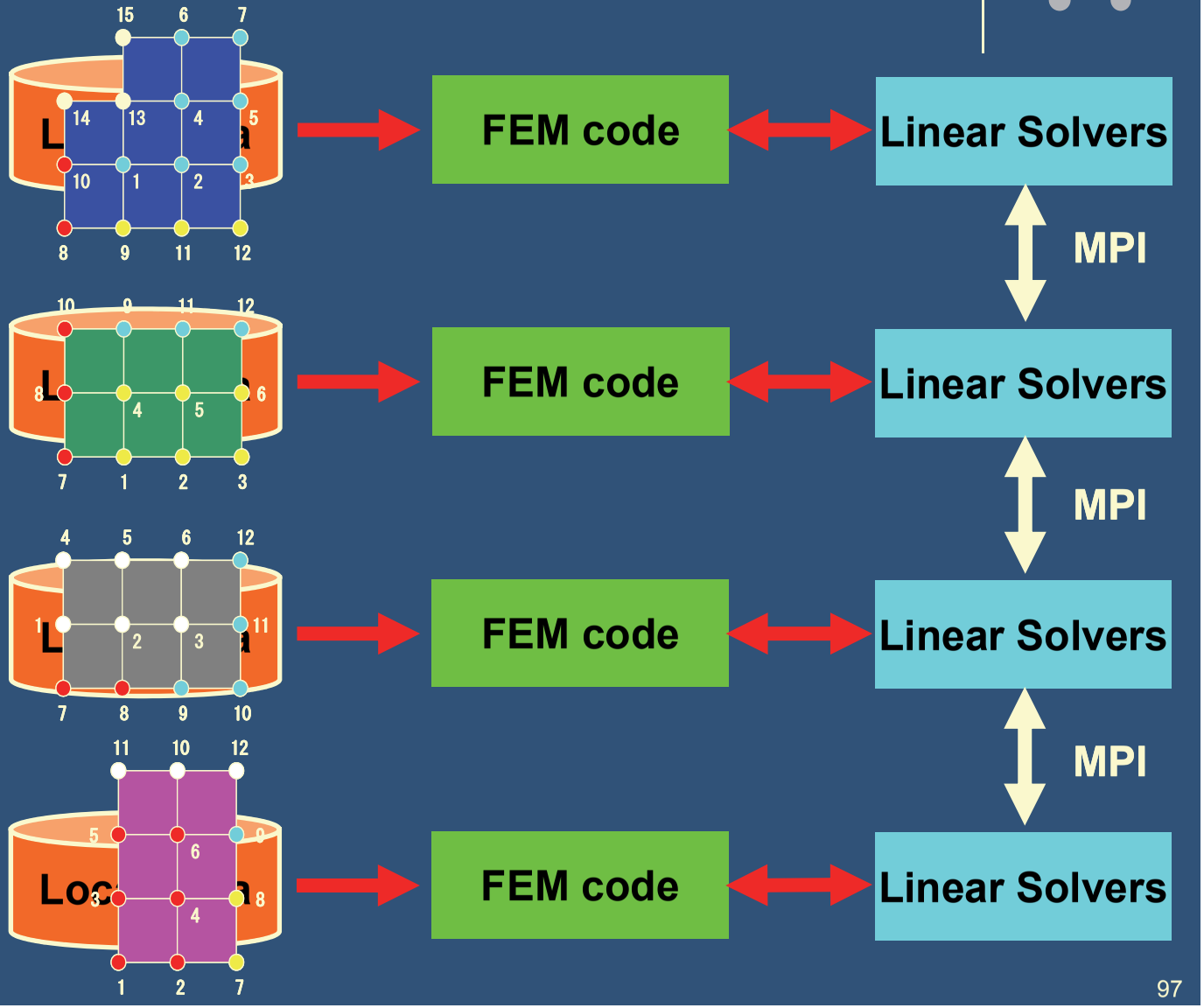
SPMD: Single-Program Multiple-Data





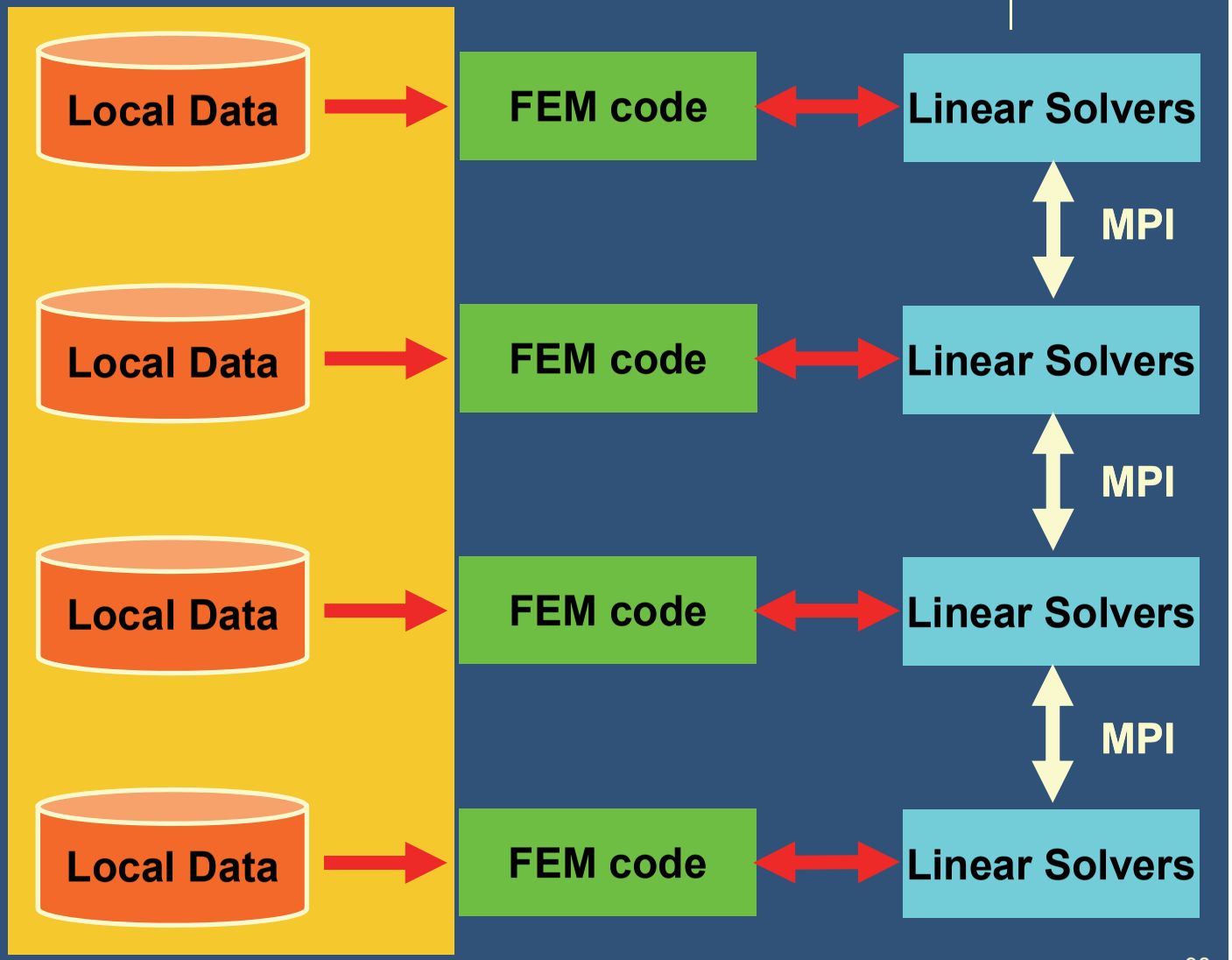
Parallel Computing in FEM

SPMD: Single-Program Multiple-Data



Parallel Computing in FEM

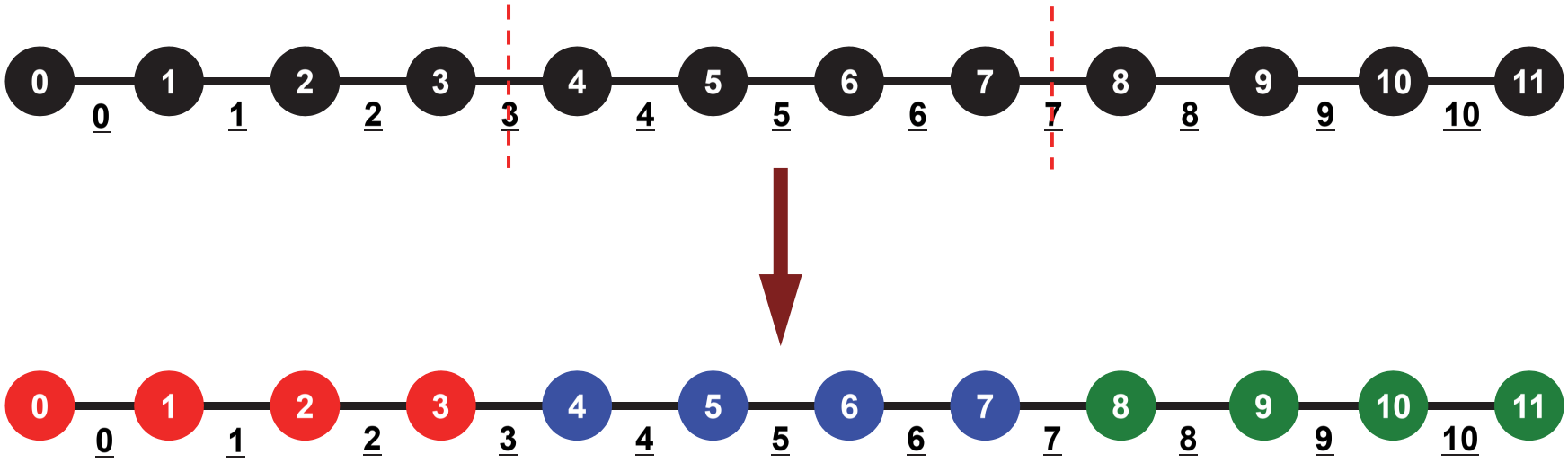
SPMD: Single-Program Multiple-Data



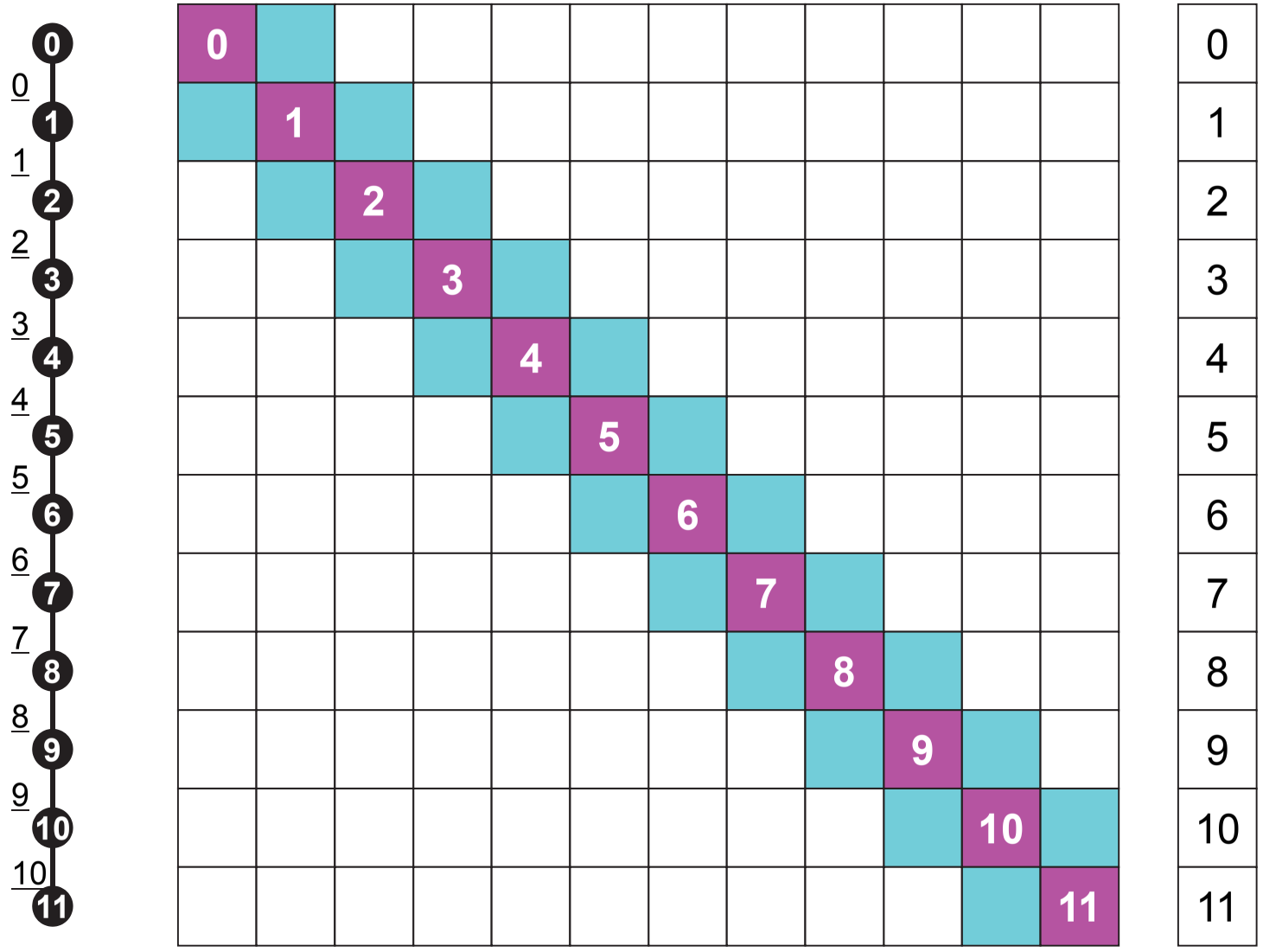
What is Communications ?

- to get information of “external nodes” from external partitions (local data)
- “Communication tables” contain the information

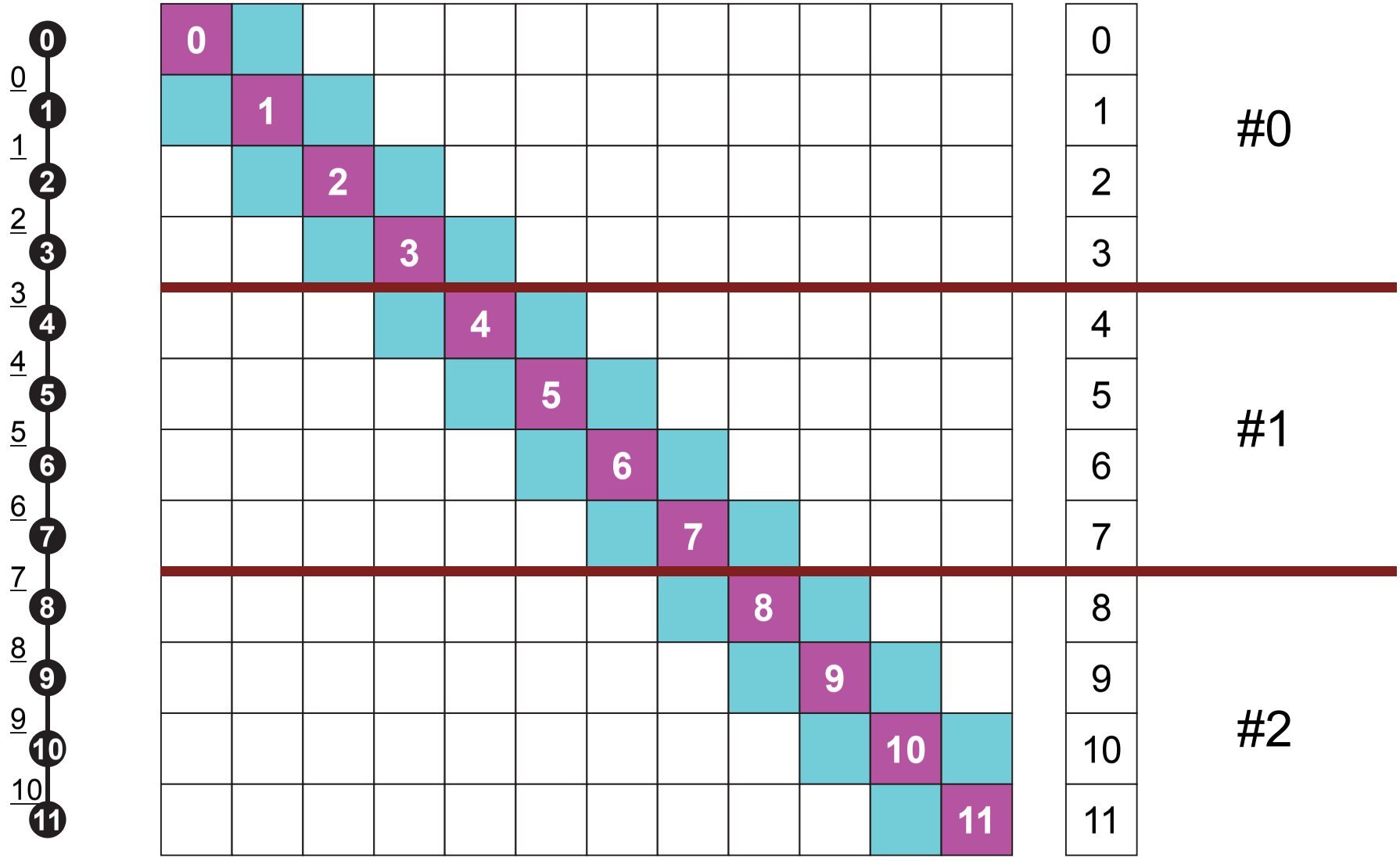
1D FEM: 12 nodes/11 elem's/3 domains



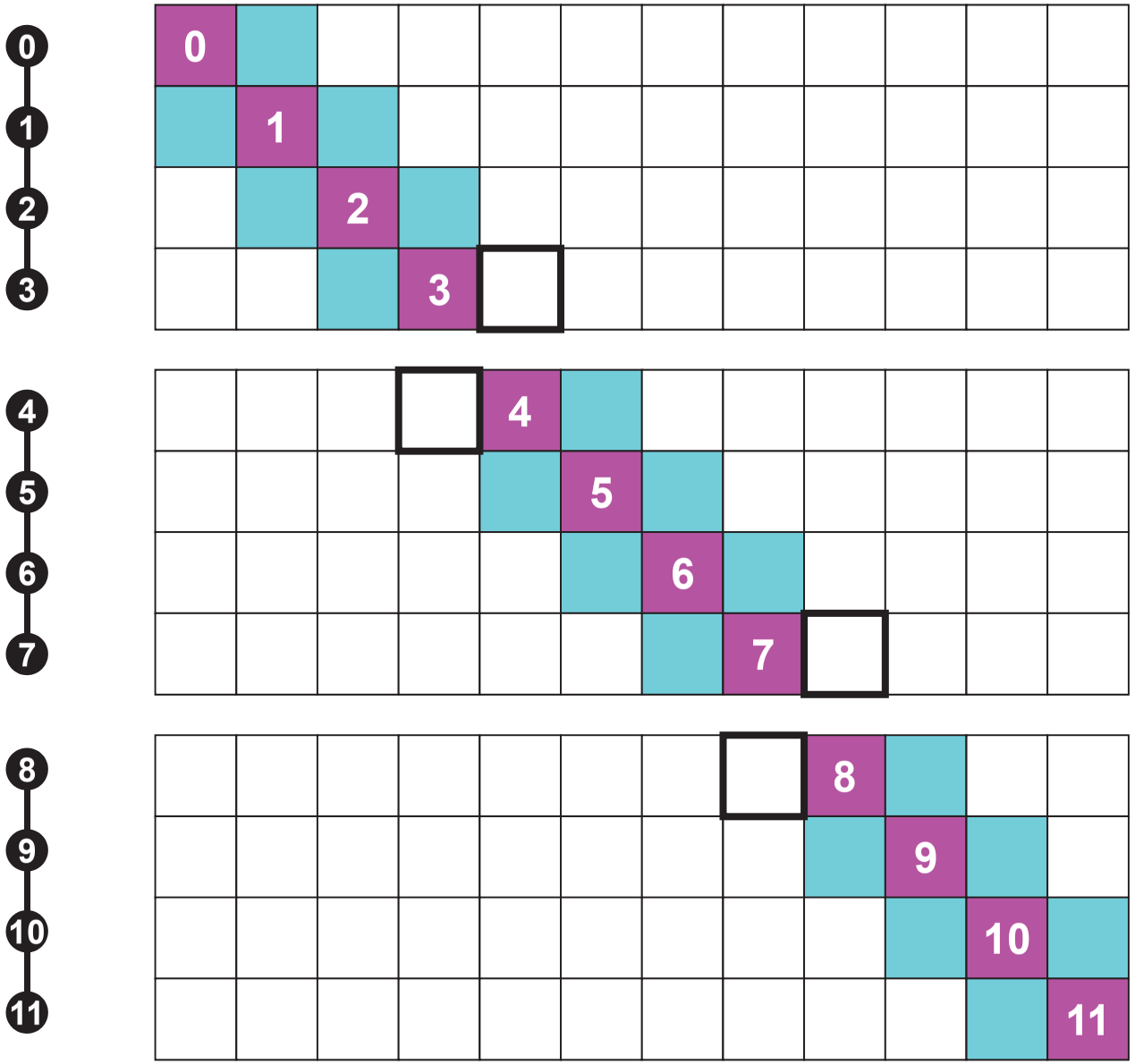
1D FEM: 12 nodes/11 elem's/3 domains



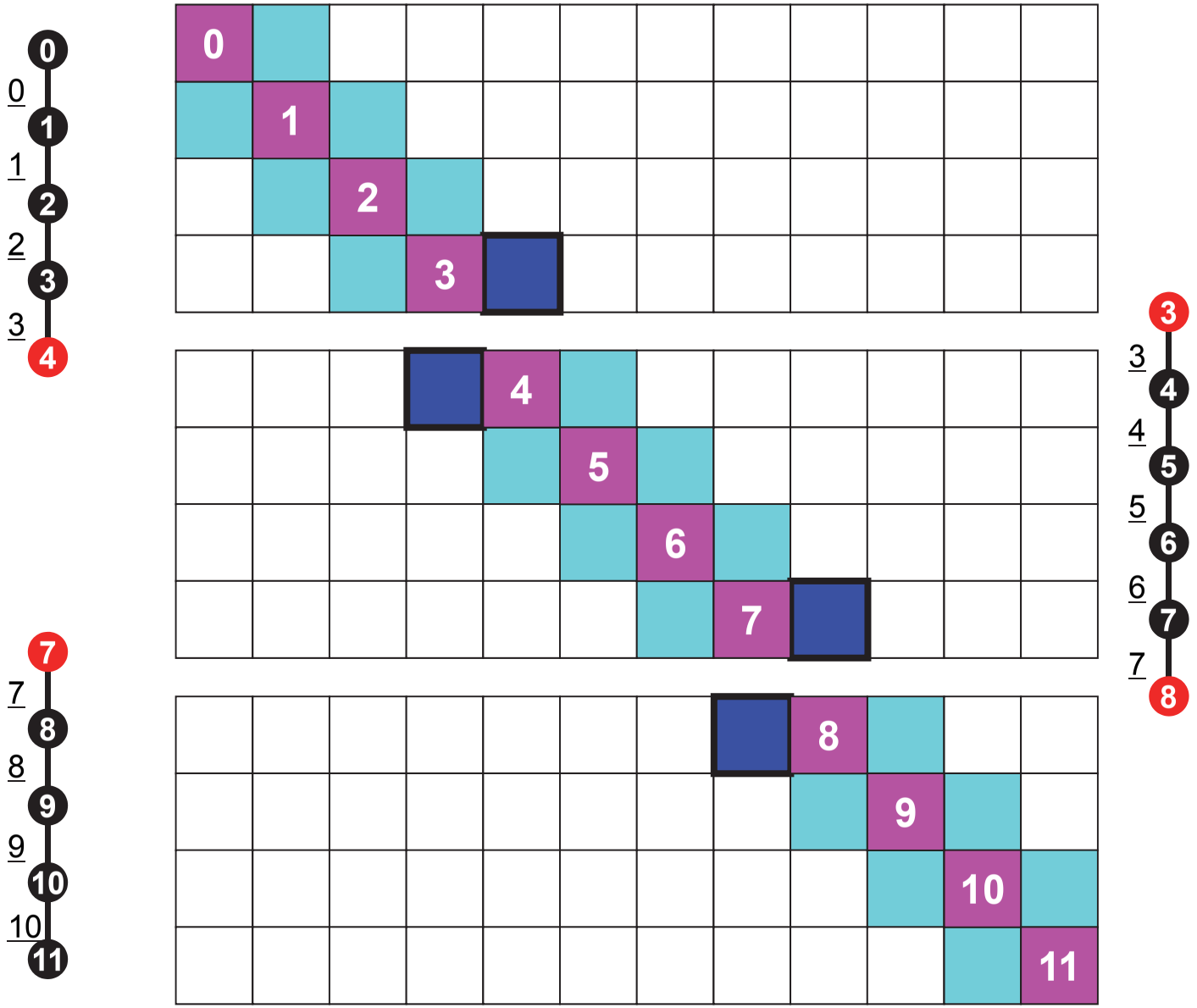
“Internal Nodes” should be balanced



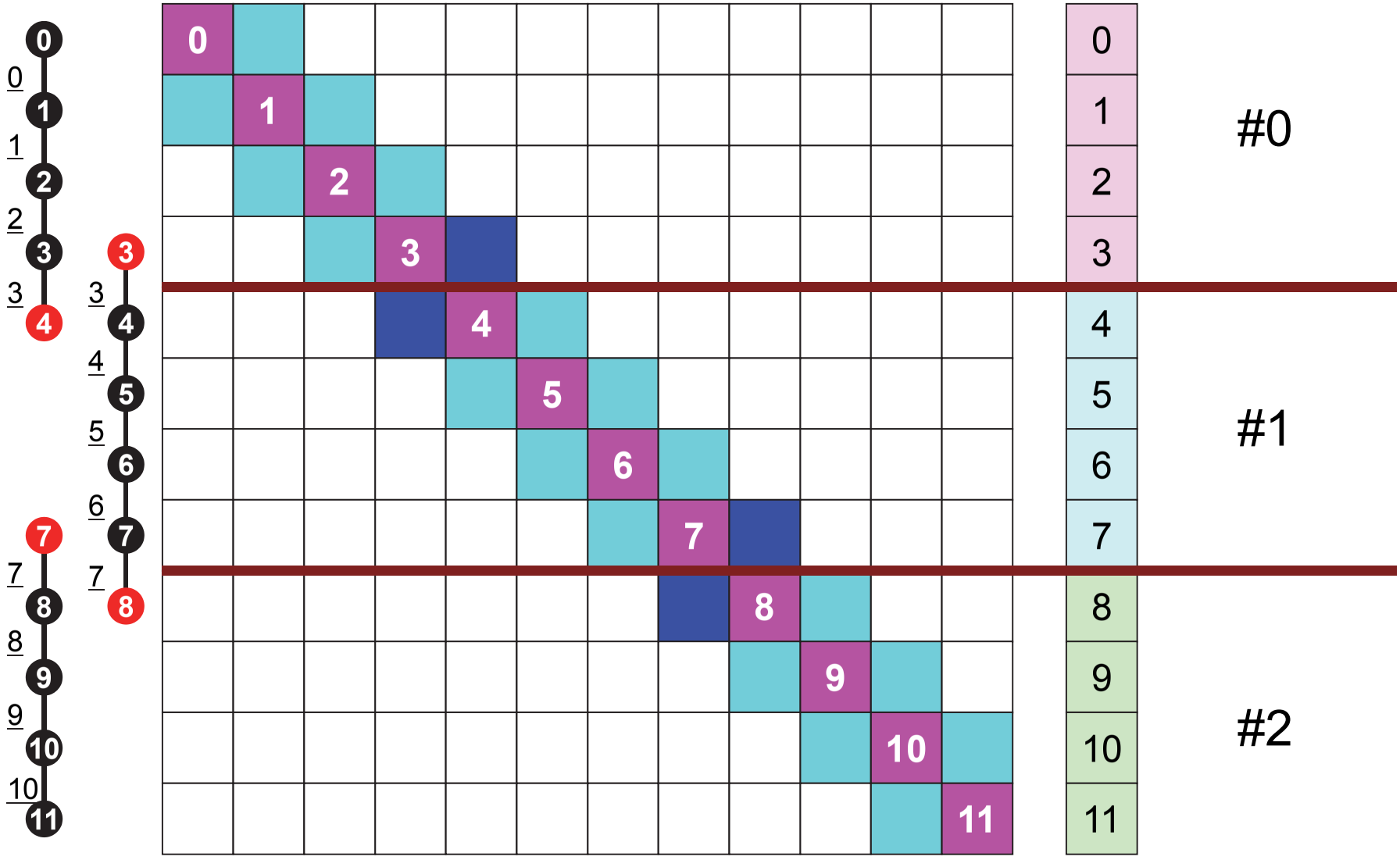
Matrices are incomplete !



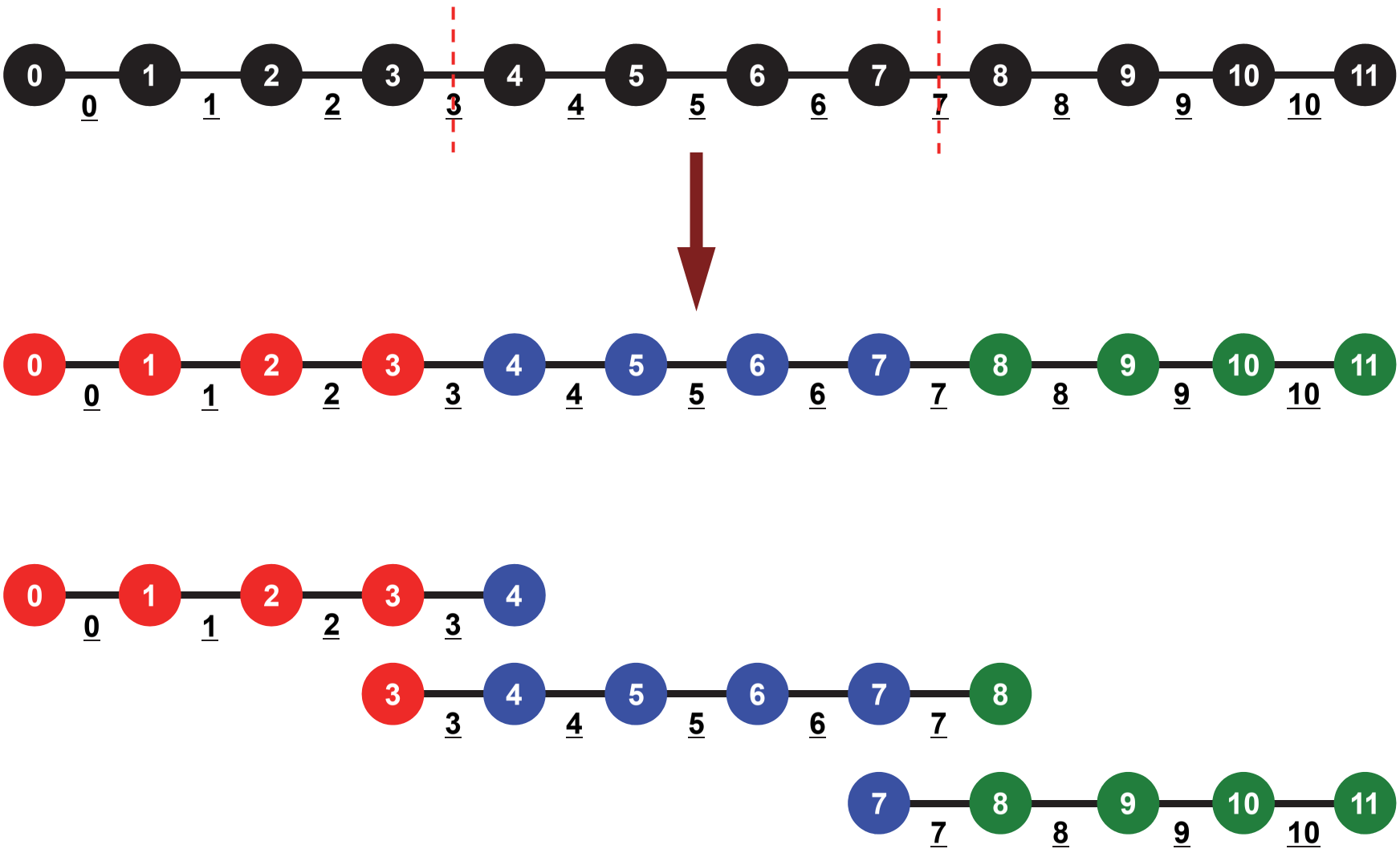
Connected Elements + External Nodes



1D FEM: 12 nodes/11 elem's/3 domains

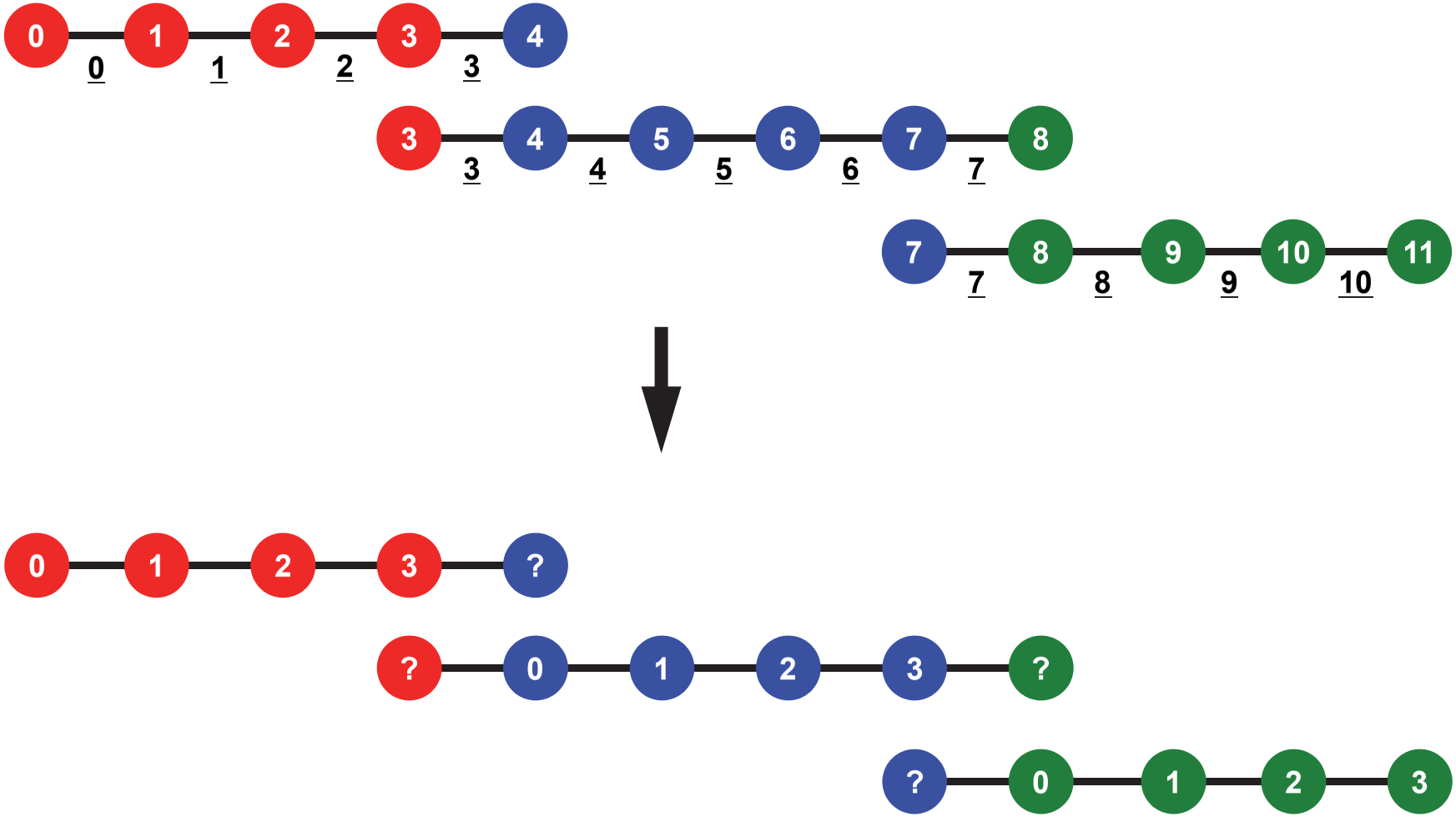


1D FEM: 12 nodes/11 elem's/3 domains



Local Numbering for SPMD

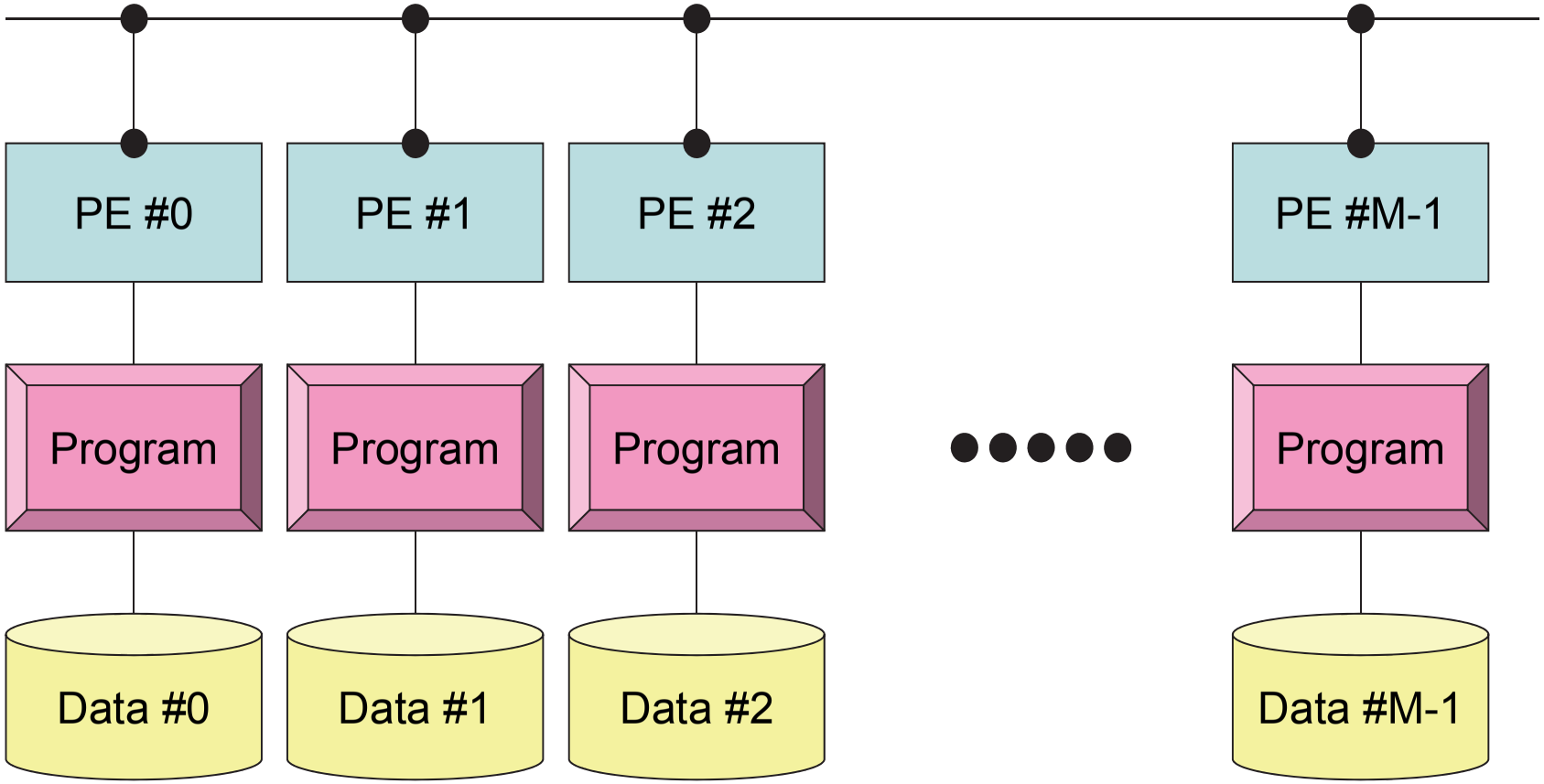
Numbering of internal nodes is 1-N (0-N-1), same operations in serial program can be applied. How about numbering of external nodes ?



PE: Processing Element
Processor, Domain, Process

SPMD

```
mpirun -np M <Program>
```



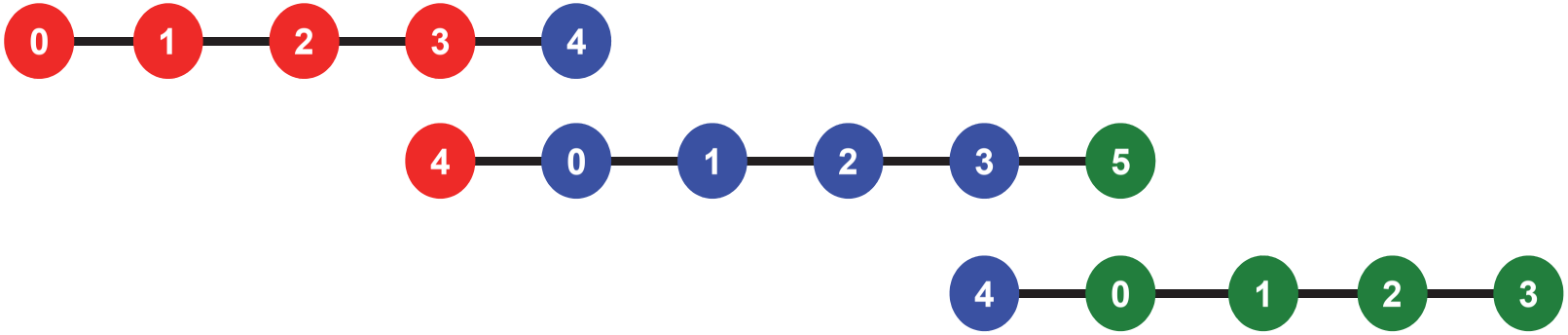
Each process does same operation for different data

Large-scale data is decomposed, and each part is computed by each process

It is ideal that parallel program is not different from serial one except communica

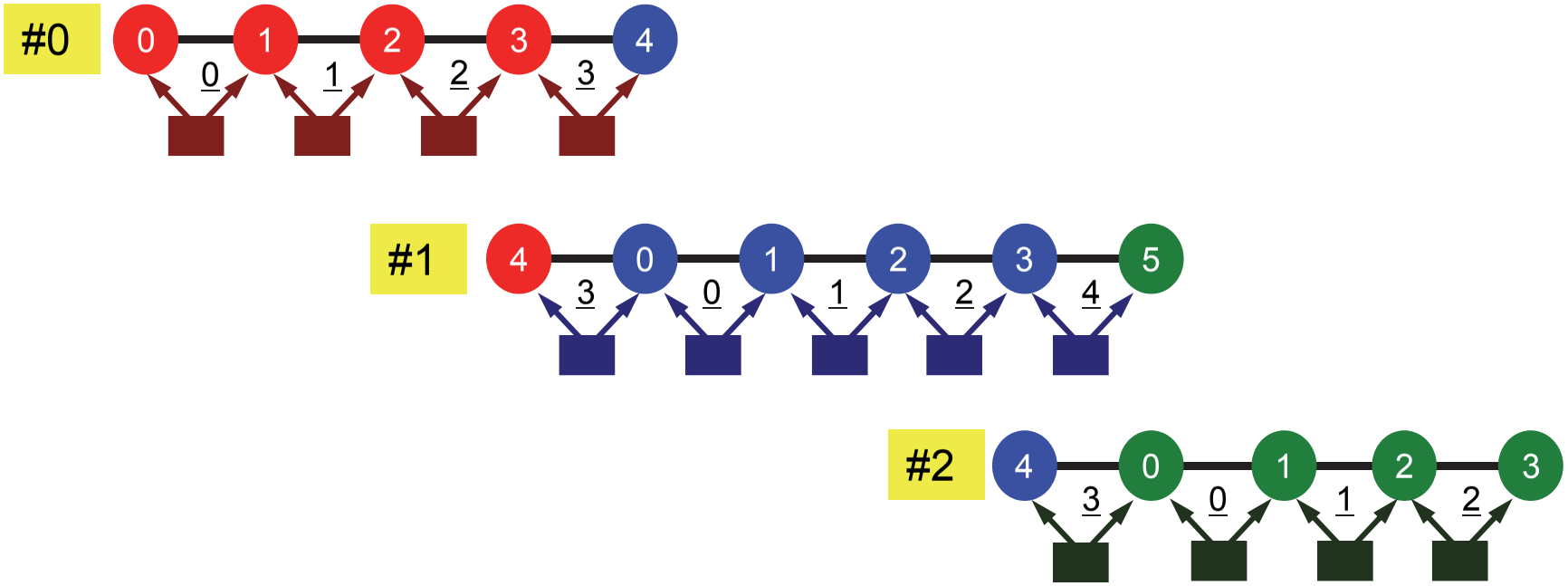
Local Numbering for SPMD

Numbering of external nodes: N+1, N+2 (N,N+1)



1D FEM: 12 nodes/11 elem's/3 domains

Integration on each element, element matrix -> global matrix
Operations can be done by info. of internal/external nodes and elements which include these nodes



Finite Element Procedures

- Initialization
  - Control Data
  - Node, Connectivity of Elements (N: Node#, NE: Elem#)
  - Initialization of Arrays (Global/Element Matrices)
  - Element-Global Matrix Mapping (Index, Item)
- Generation of Matrix
  - Element-by-Element Operations (do icel= 1, NE)
    - Element matrices
    - Accumulation to global matrix
  - Boundary Conditions
- Linear Solver
  - Conjugate Gradient Method

Preconditioned CG Solver

Compute $\mathbf{r}^{(0)}= \mathbf{b}-[\mathbf{A}]\mathbf{x}^{(0)}$
for i= 1, 2, …
  solve $[\mathbf{M}]\mathbf{z}^{(i-1)}= \mathbf{r}^{(i-1)}$
  $\rho_{i-1}= \mathbf{r}^{(i-1)}\ \mathbf{z}^{(i-1)}$
  if i=1
    $p^{(1)}= z^{(0)}$
  else
    $\beta_{i-1}= \rho_{i-1}/\rho_{i-2}$
    $p^{(i)}= z^{(i-1)} + \beta_{i-1}\ p^{(i-1)}$
  endif
  $\mathbf{q}^{(i)}= [\mathbf{A}]\mathbf{p}^{(i)}$
  $\alpha_i = \rho_{i-1}/\mathbf{p}^{(i)}\mathbf{q}^{(i)}$
  $x^{(i)}= x^{(i-1)} + \alpha_i p^{(i)}$
  $r^{(i)}= r^{(i-1)} - \alpha_i q^{(i)}$
  check convergence $|\mathbf{r}|$
end

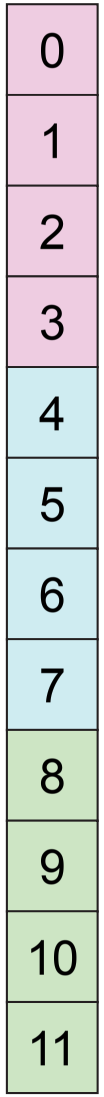
$$[M]=\begin{bmatrix} D_1 & 0 & \cdots & 0 & 0 \\ 0 & D_2 & & 0 & 0 \\ \cdots & & \cdots & & \cdots \\ 0 & 0 & & D_{N-1} & 0 \\ 0 & 0 & \cdots & 0 & D_N \end{bmatrix}$$

Preconditioning, DAXPY

Local Operations by Only Internal Points: Parallel Processing is possible

```
/*
//-- {z}= [Minv]{r}
*/
    for(i=0;i<N;i++){
        W[Z][i] = W[DD][i] * W[R][i];
    }
```

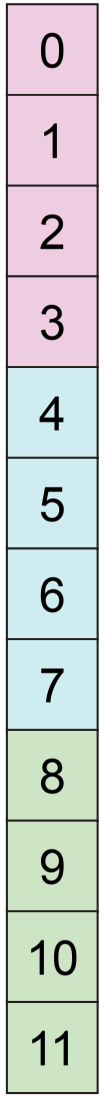
```
/*
//-- {x}= {x} + ALPHA*{p}        DAXPY: double a{x} plus
{y}
//    {r}= {r} - ALPHA*{q}
*/
  for(i=0;i<N;i++){
      U[i]    += Alpha * W[P][i];
      W[R][i] -= Alpha * W[Q][i];
  }
```



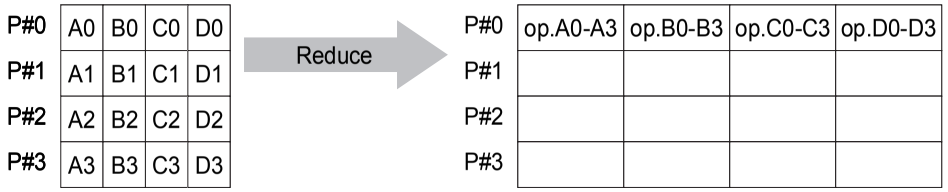
Dot Products

Global Summation needed: Communication ?

```
/*
//-- ALPHA= RHO / {p}{q}
*/
  C1 = 0.0;
  for(i=0;i<N;i++){
      C1 += W[P][i] * W[Q][i];
  }

  Alpha = Rho / C1;
```



MPI_Reduce

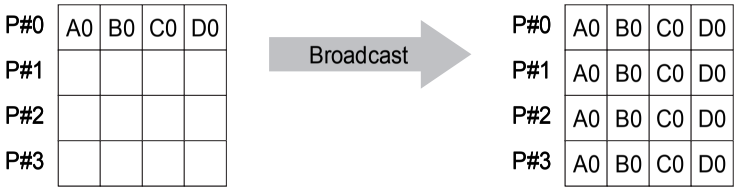


- Reduces values on all processes to a single value
  - Summation, Product, Max, Min etc.

- **MPI_Reduce**
  (sendbuf,recvbuf,count,datatype,op,root,comm)
  - **sendbuf** choice I starting address of send buffer
  - **recvbuf** choice O starting address receive buffer
    type is defined by "**datatype**"
  - **count** int I number of elements in send/receive buffer
  - **datatype** MPI_Datatype I data type of elements of send/recive buffer
    FORTRAN MPI_INTEGER, MPI_REAL, MPI_DOUBLE_PRECISION, MPI_CHARACTER etc.
    C MPI_INT, MPI_FLOAT, MPI_DOUBLE, MPI_CHAR etc
  - **op** MPI_Op I reduce operation
    MPI_MAX, MPI_MIN, MPI_SUM, MPI_PROD, MPI_LAND, MPI_BAND etc
    Users can define operations by **MPI_OP_CREATE**
  - **root** int I rank of root process
  - **comm** MPI_Comm I communicator

C

MPI_Bcast

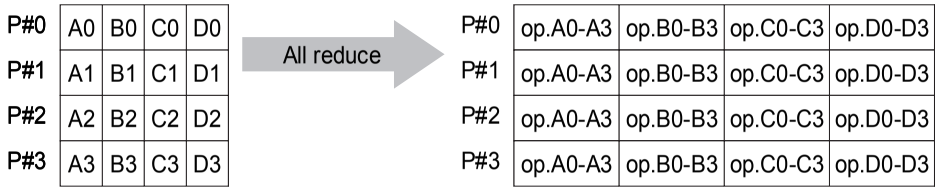


- Broadcasts a message from the process with rank "root" to all other processes of the communicator

- **MPI_Bcast (buffer,count,datatype,root,comm)**
  - **buffer** choice I/O starting address of buffer
    type is defined by "**datatype**"
  - **count** int I number of elements in send/recv buffer
  - **datatype** MPI_Datatype I data type of elements of send/recv buffer
    FORTRAN MPI_INTEGER, MPI_REAL, MPI_DOUBLE_PRECISION, MPI_CHARACTER etc.
    C MPI_INT, MPI_FLOAT, MPI_DOUBLE, MPI_CHAR etc.
  - **root** int I rank of root process
  - **comm** MPI_Comm I communicator

C

MPI_Allreduce



- MPI_Reduce + MPI_Bcast
- Summation (of dot products) and MAX/MIN values are likely to utilized in each process

- **`call MPI_Allreduce`**
  `(sendbuf,recvbuf,count,datatype,op, comm)`
  - **`sendbuf`** choice I starting address of send buffer
  - **`recvbuf`** choice O starting address receive buffer
    type is defined by "**`datatype`**"
  - **`count`** int I number of elements in send/recv buffer
  - **`datatype`** MPI_Datatype I data type of elements of send/recv buffer
  - **`op`** MPI_Op I reduce operation
  - **`comm`** MPI_Comm I communicator

C

C

“op” of MPI_Reduce/Allreduce

```
MPI_Reduce
(sendbuf,recvbuf,count,datatype,op,root,comm)
```

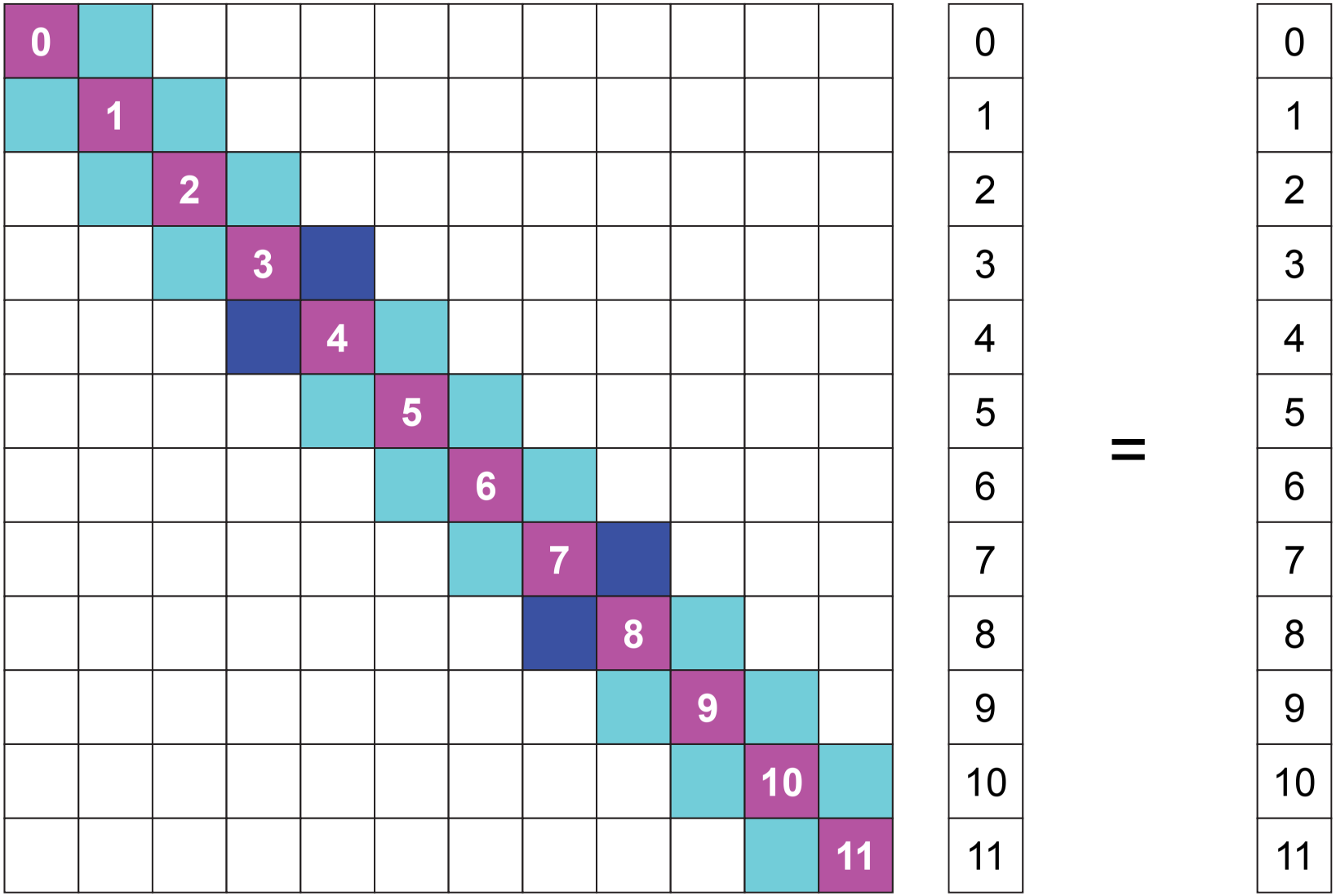
- **MPI_MAX**, **MPI_MIN** Max, Min
- **MPI_SUM**, **MPI_PROD** Summation, Product
- **MPI_LAND** Logical AND

Matrix-Vector Products

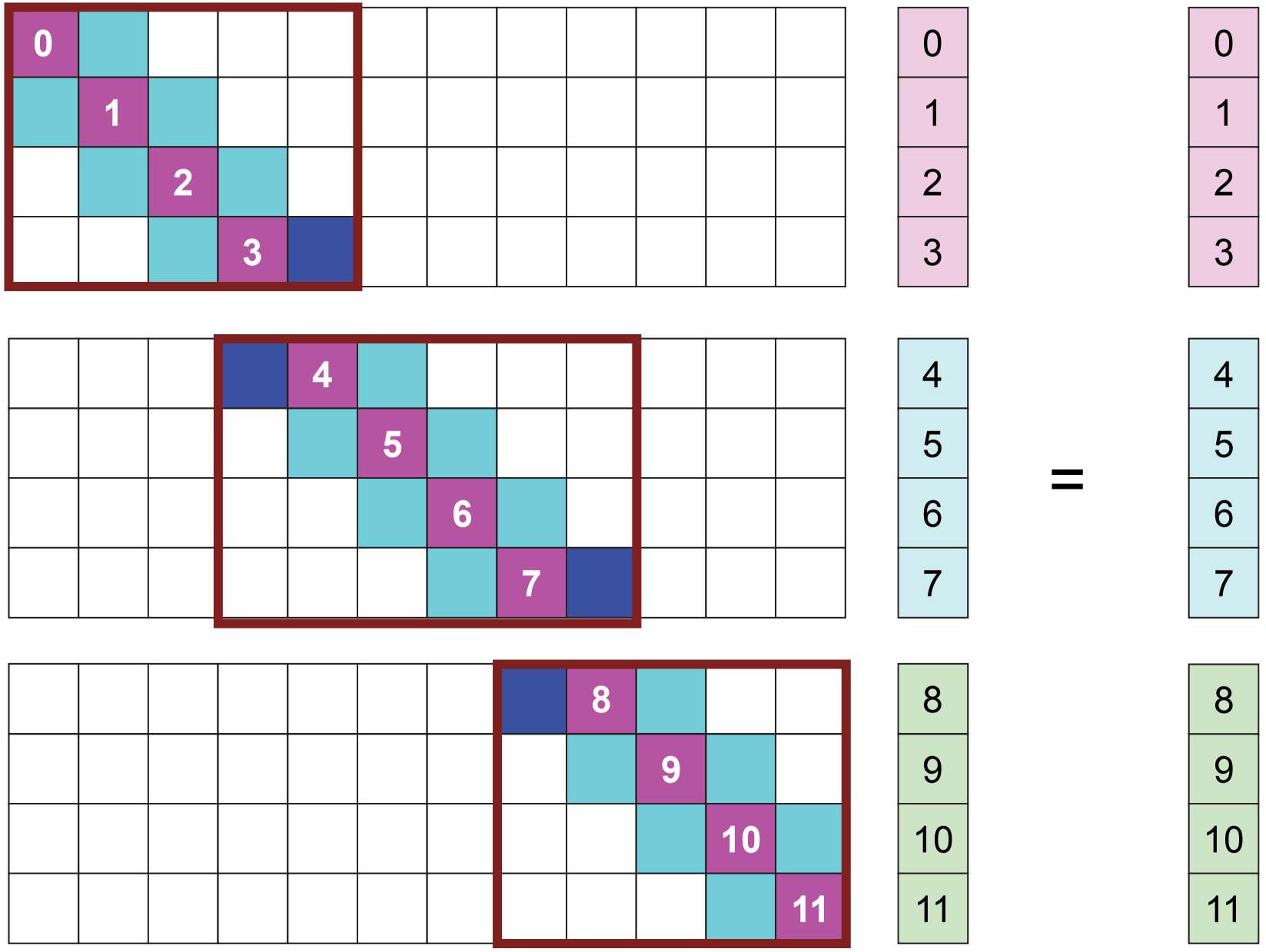
Values at External Points: P-to-P Communication

```
/*
//-- {q}= [A]{p}
*/
  for(i=0;i<N;i++){
      W[Q][i] = Diag[i] * W[P][i];
      for(j=Index[i];j<Index[i+1];j++){
          W[Q][i] += AMat[j]*W[P][Item[j]];
          }
      }
```

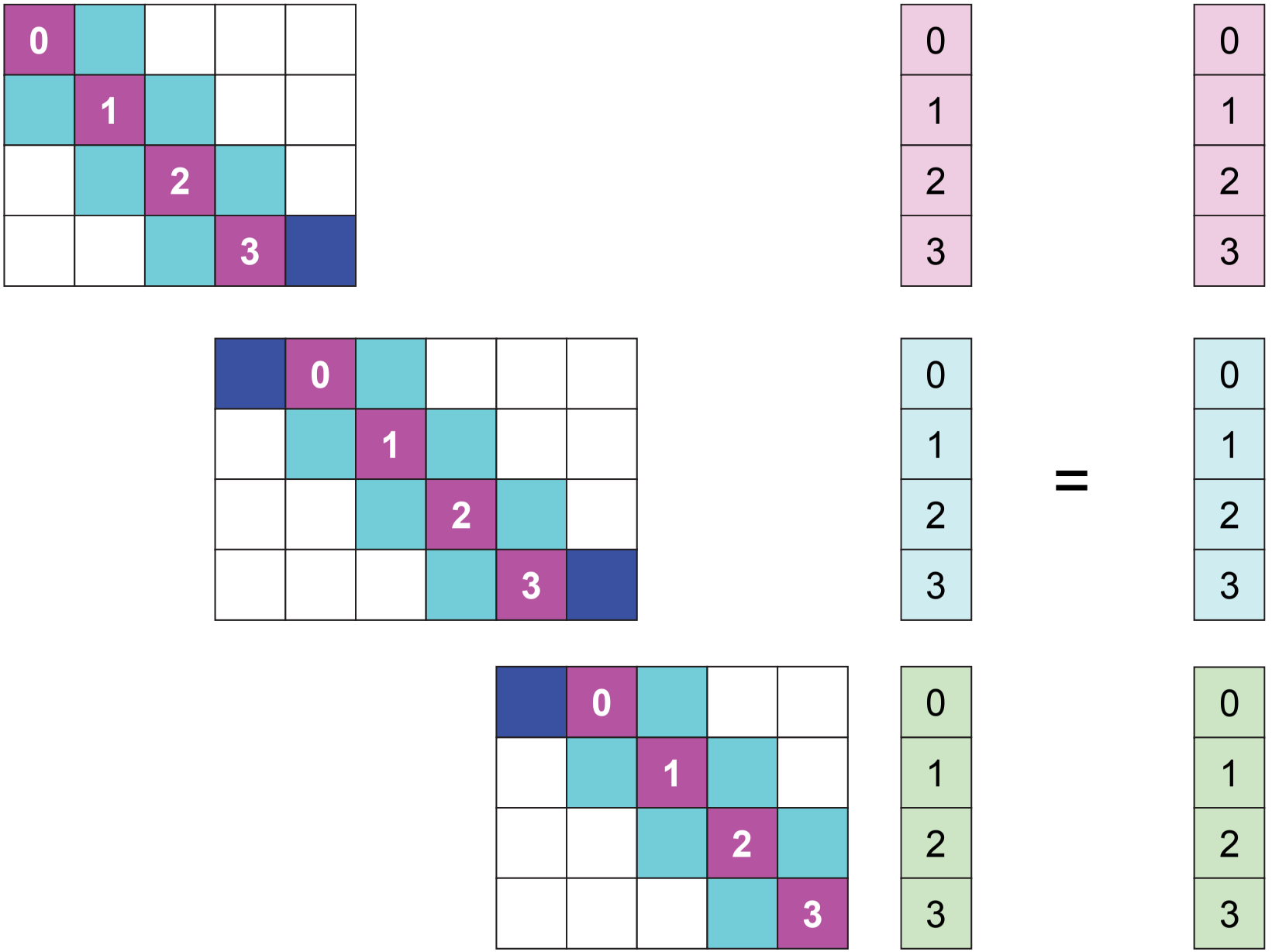
Mat-Vec Products: Local Op. Possible



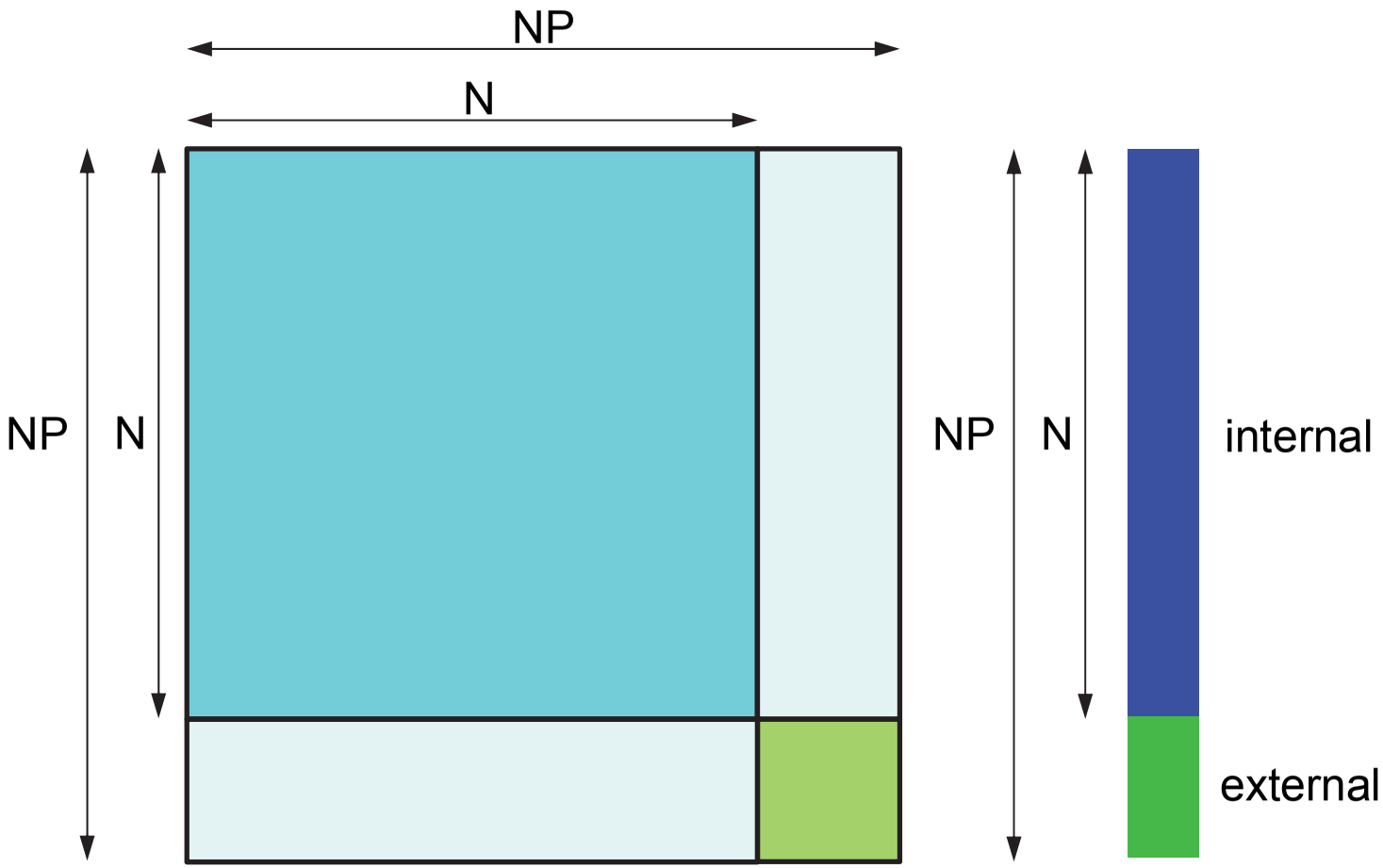
Mat-Vec Products: Local Op. Possible



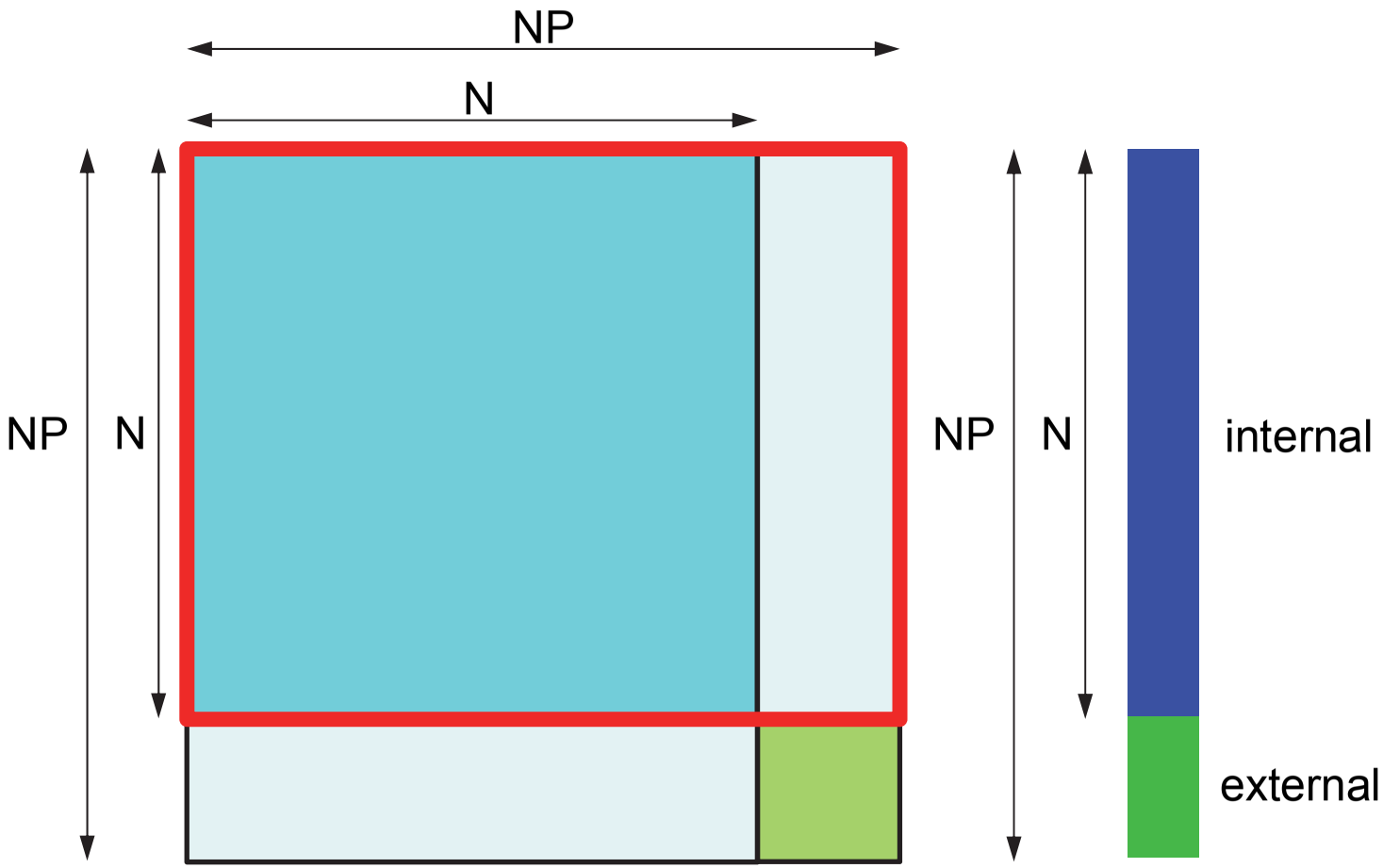
Mat-Vec Products: Local Op. Possible



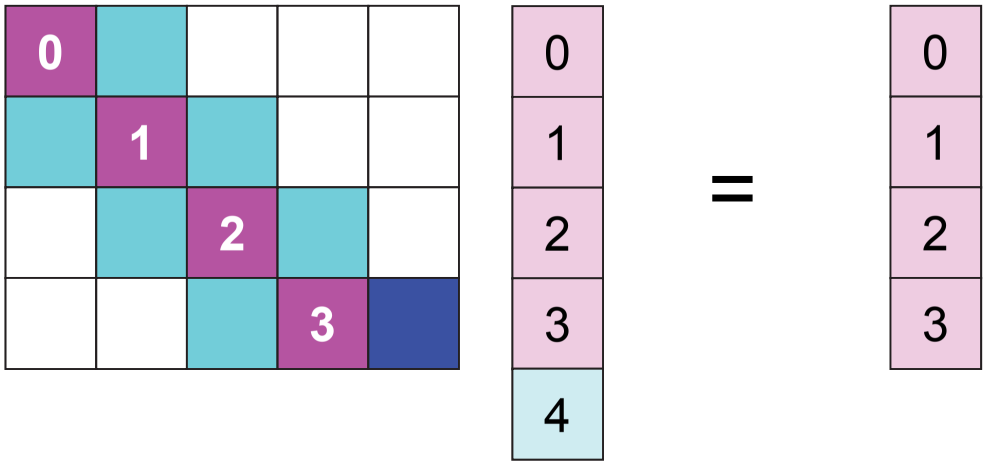
Local Matrix



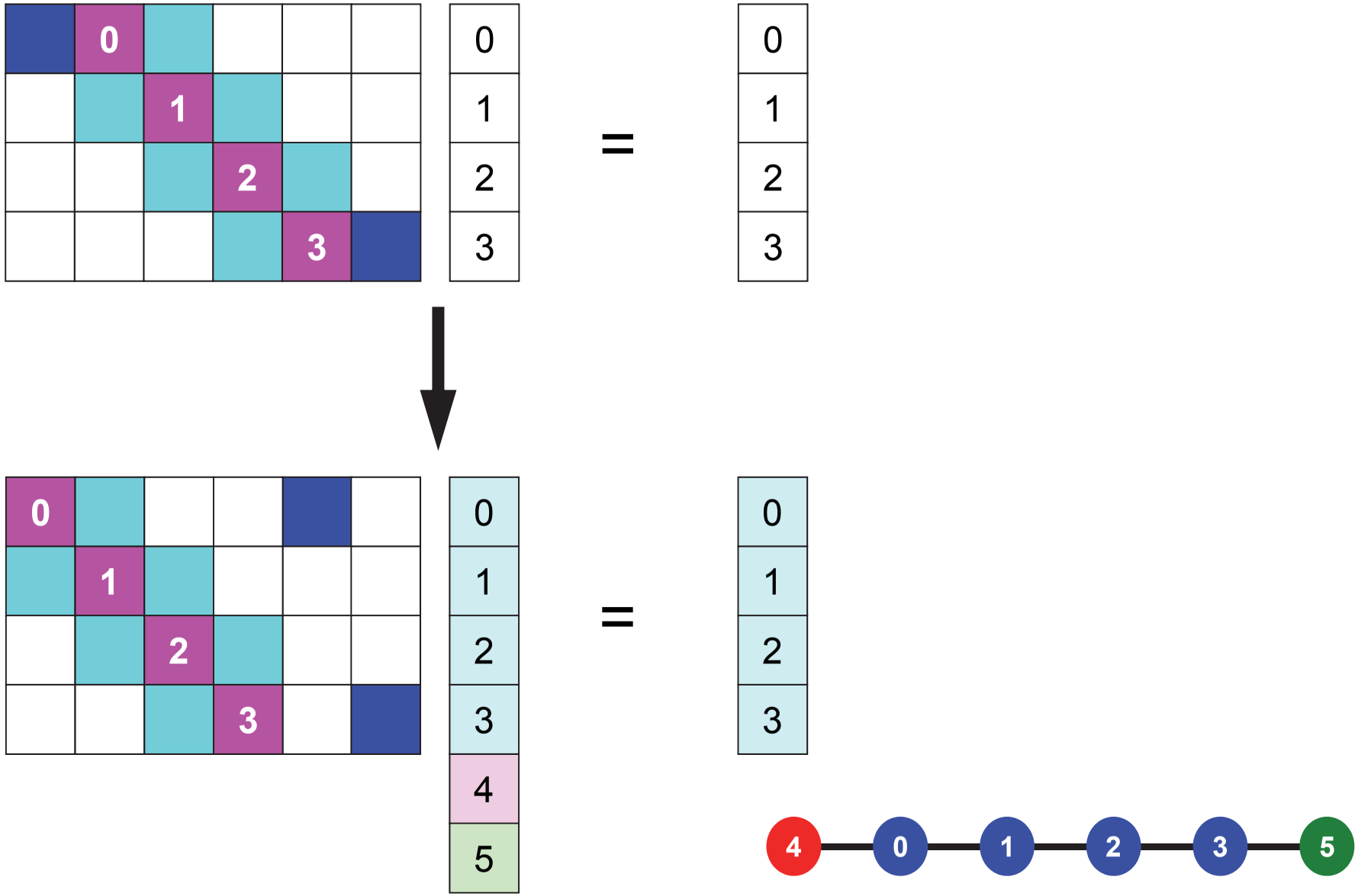
We really need these parts



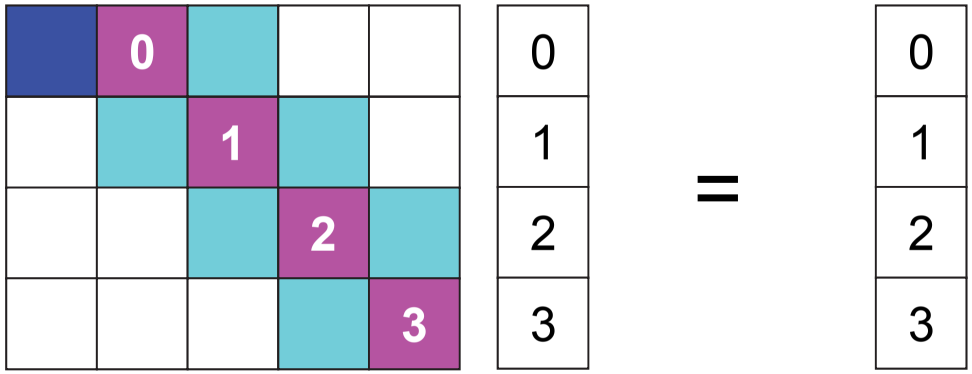
Mat-Vec Products: Local Op. #0

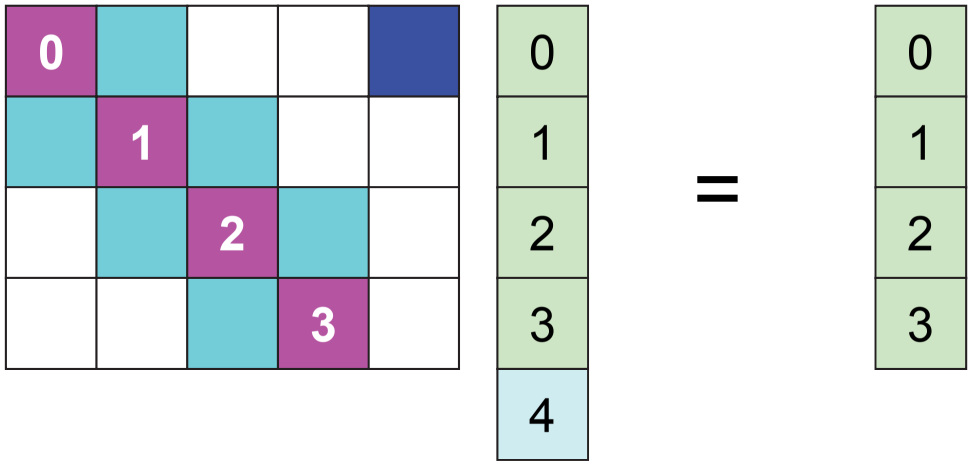


Mat-Vec Products: Local Op. #1

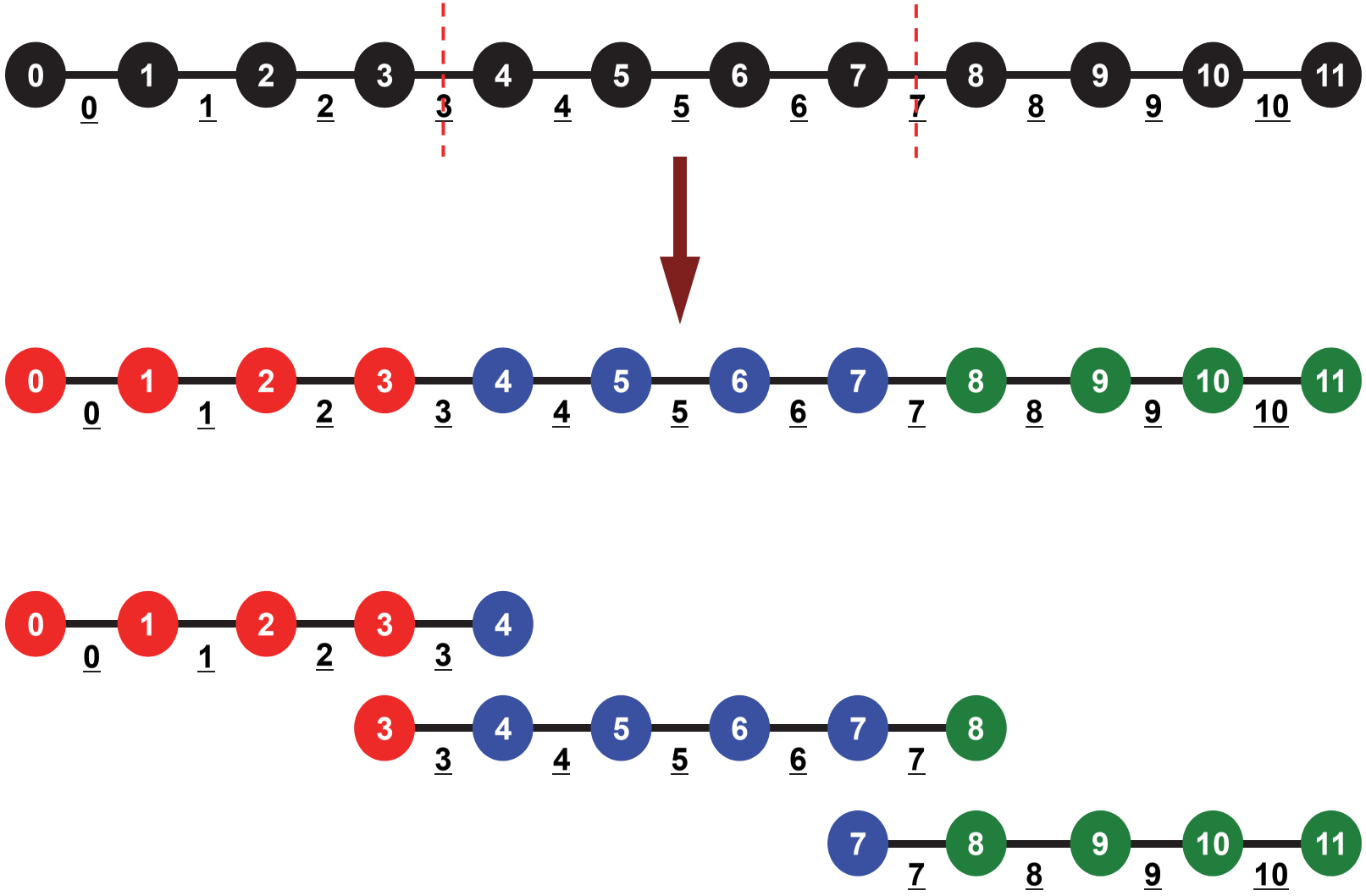


Mat-Vec Products: Local Op. #2



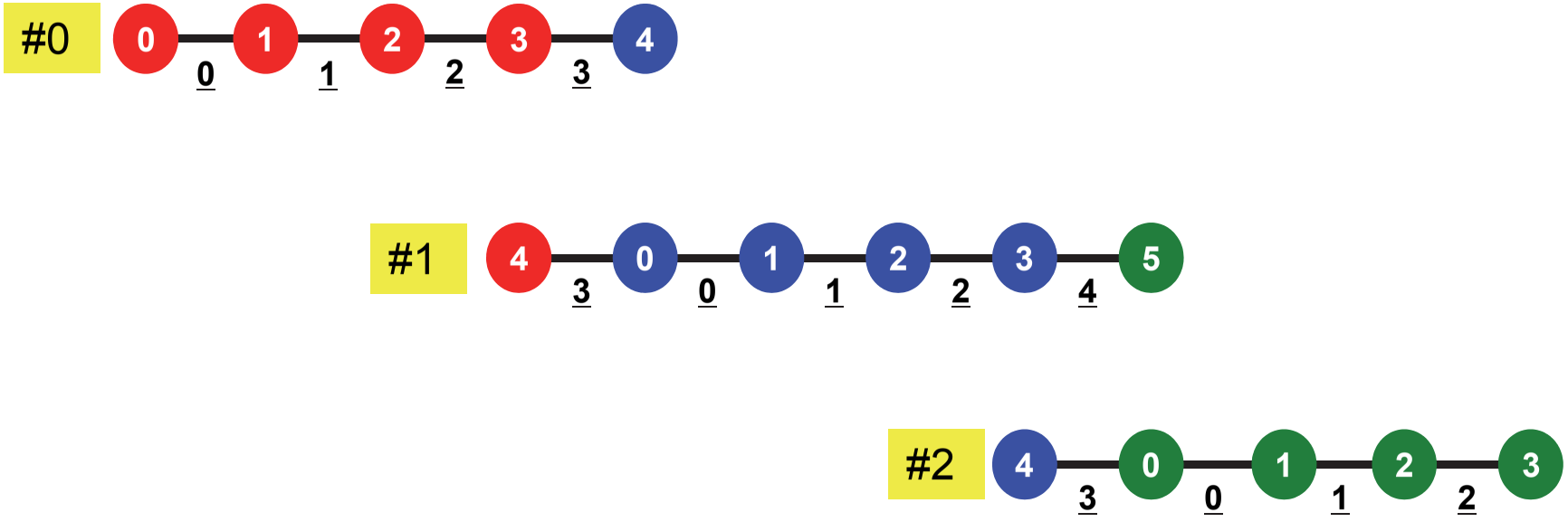


1D FEM: 12 nodes/11 elem's/3 domains



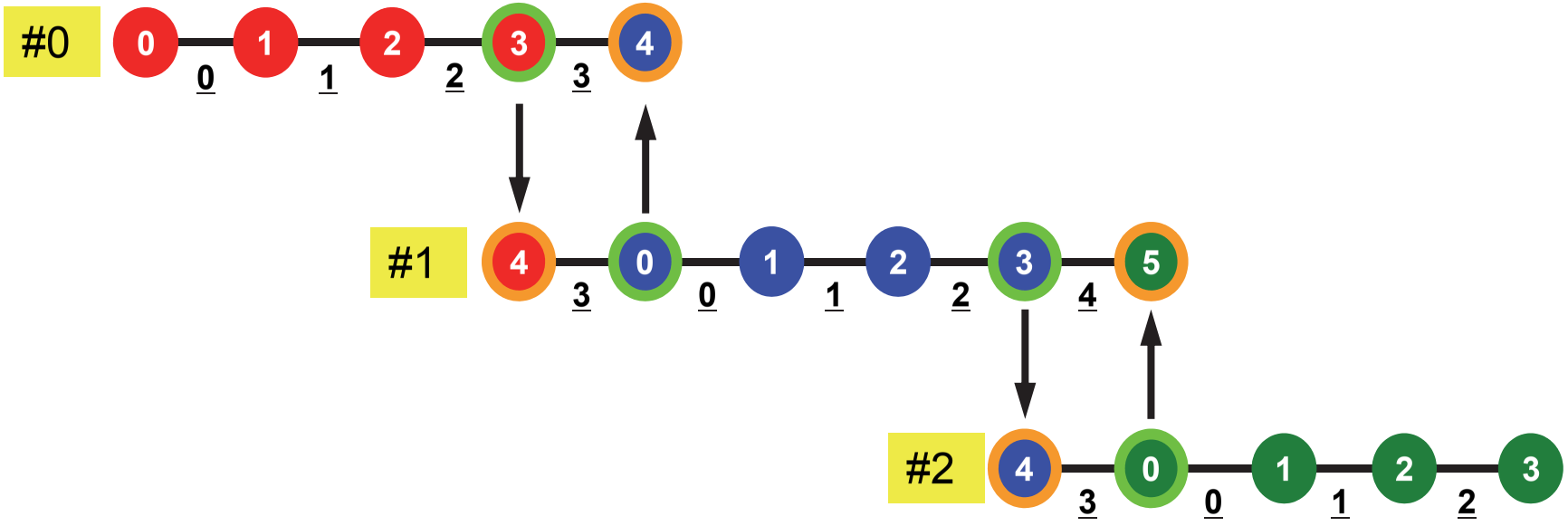
1D FEM: 12 nodes/11 elem's/3 domains

Local ID: Starting from 0 for node and elem at each domain



1D FEM: 12 nodes/11 elem's/3 domains

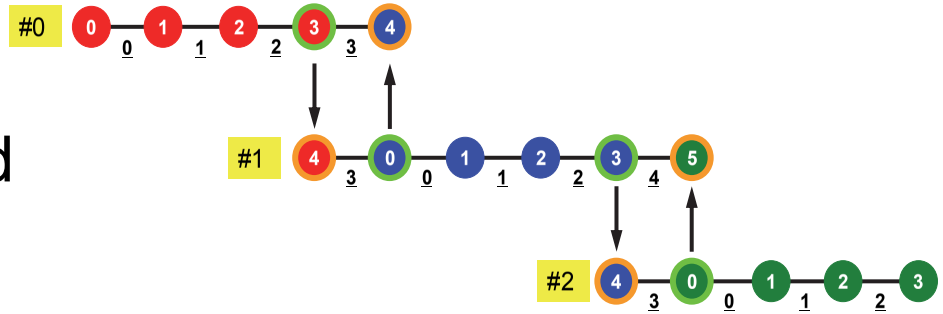
Internal/External Nodes



What is Peer-to-Peer Communication ?

- Collective Communication
  - MPI_Reduce, MPI_Scatter/Gather etc.
  - Communications with all processes in the communicator
  - Application Area
    - BEM, Spectral Method, MD: global interactions are considered
    - Dot products, MAX/MIN: Global Summation & Comparison

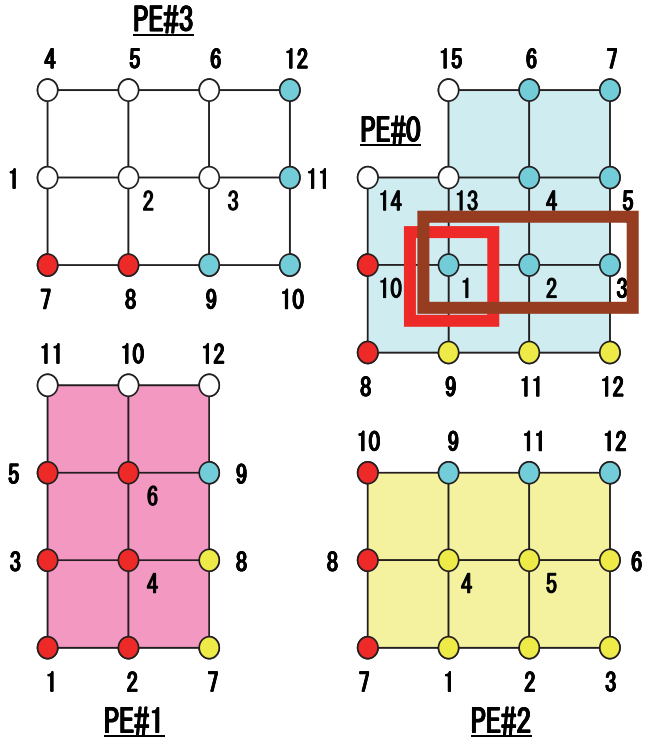
- Peer-toPeer/Point-to-Point
  - MPI_Send, MPI_Receive
  - Communication with limited processes
    - Neighbors
  - Application Area
    - FEM, FDM: Localized Method



SEND: sending from boundary nodes

Send continuous data to send buffer of neighbors

- **`MPI_Isend`**
  `(sendbuf,count,datatype,dest,tag,comm,request)`
  - **`sendbuf`** choice I — starting address of sending buffer
  - **`count`** I I — number of elements sent to each process
  - **`datatype`** I I — data type of elements of sending buffer
  - **`dest`** I I — rank of destination



C

MPI_Isend

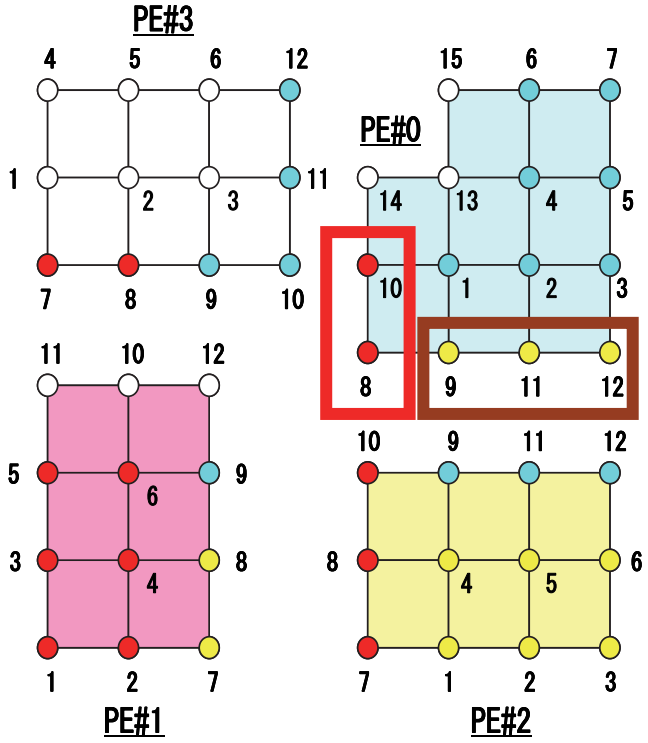
- Begins a non-blocking send
  - Send the contents of sending buffer (starting from **sendbuf**, number of messages: **count**) to **dest** with **tag** .
  - Contents of sending buffer cannot be modified before calling corresponding **MPI_Waitall**.

- **MPI_Isend (sendbuf,count,datatype,dest,tag,comm,request)**
  - **sendbuf** choice I starting address of sending buffer
  - **count** int I number of elements in sending buffer
  - **datatype** MPI_Datatype I datatype of each sending buffer element
  - **dest** int I rank of destination
  - **tag** int I message tag
    This integer can be used by the application to distinguish messages. Communication occurs if `tag's` of `MPI_Isend` and `MPI_Irecv` are matched.
    Usually tag is set to be "0" (in this class),
  - **comm** MPI_Comm I communicator
  - **request** MPI_Request O communication request array used in `MPI_Waitall`

RECV: receiving to external nodes

Recv. continuous data to recv. buffer from neighbors

- **`MPI_Irecv`**
  `(recvbuf,count,datatype,dest,tag,comm,request)`
  - **`recvbuf`** choice I — starting address of receiving buffer
  - **`count`** I I — number of elements in receiving buffer
  - **`datatype`** I I — data type of elements of receiving buffer
  - **`source`** I I — rank of source



C

MPI_Irecv

- Begins a non-blocking receive
  - Receiving the contents of receiving buffer (starting from **recvbuf**, number of messages: **count**) from **source** with **tag** .
  - Contents of receiving buffer cannot be used before calling corresponding **MPI_Waitall**.

- **MPI_Irecv (recvbuf,count,datatype,source,tag,comm,request)**
  - **recvbuf** choice I — starting address of receiving buffer
  - **count** int I — number of elements in receiving buffer
  - **datatype** MPI_Datatype I — datatype of each receiving buffer element
  - **source** int I — rank of source
  - **tag** int I — message tag
    This integer can be used by the application to distinguish messages. Communication occurs if `tag's` of `MPI_Isend` and `MPI_Irecv` are matched. Usually tag is set to be "0" (in this class),
  - **comm** MPI_Comm I — communicator
  - **request** MPI_Request O — communication request array used in `MPI_Waitall`

MPI_Waitall

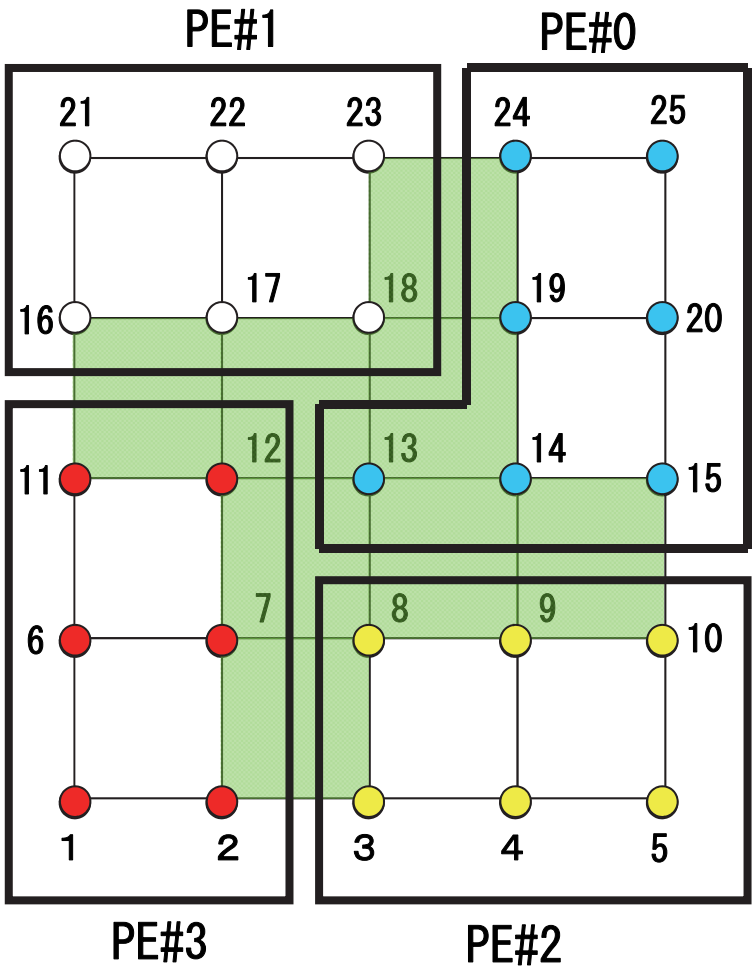
C

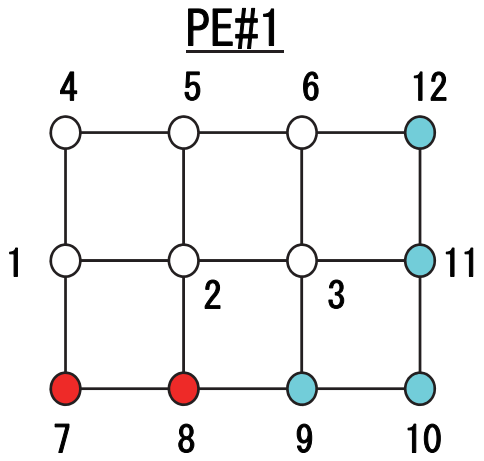
- **MPI_Waitall** blocks until all comm's, associated with **request** in the array, complete. It is used for synchronizing **MPI_Isend** and **MPI_Irecv** in this class.
- At sending phase, contents of sending buffer cannot be modified before calling corresponding **MPI_Waitall**. At receiving phase, contents of receiving buffer cannot be used before calling corresponding **MPI_Waitall**.
- **MPI_Isend** and **MPI_Irecv** can be synchronized simultaneously with a single **MPI_Waitall** if it is consitent.
  - Same **request** should be used in **MPI_Isend** and **MPI_Irecv**.
- Its operation is similar to that of **MPI_Barrier** but, **MPI_Waitall** can not be replaced by **MPI_Barrier.**
  - Possible troubles using **MPI_Barrier** instead of **MPI_Waitall**: Contents of **request** and **status** are not updated properly, very slow operations etc.

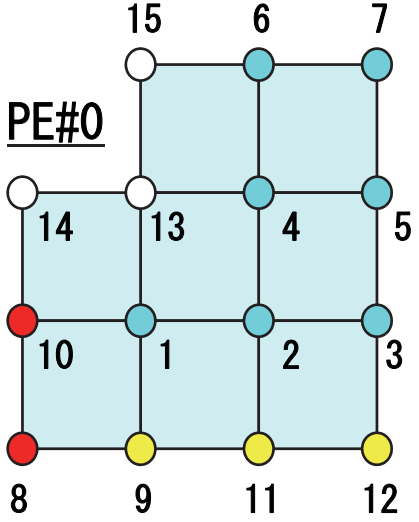
- **MPI_Waitall (count,request,status)**
  - **count** int I number of processes to be synchronized
  - **request** MPI_Request I/O comm. request used in `MPI_Waitall` (array size: `count`)
  - **status** MPI_Status O array of status objects
    `MPI_STATUS_SIZE: defined in 'mpi.h'`

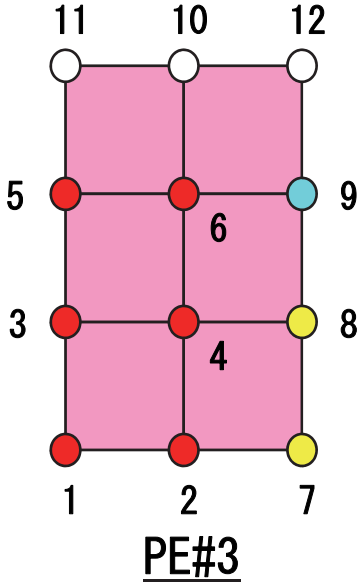
Node-based Partitioning

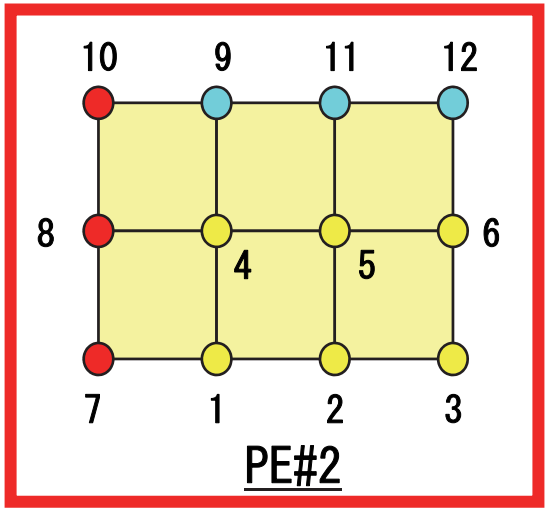
internal nodes - elements - external nodes



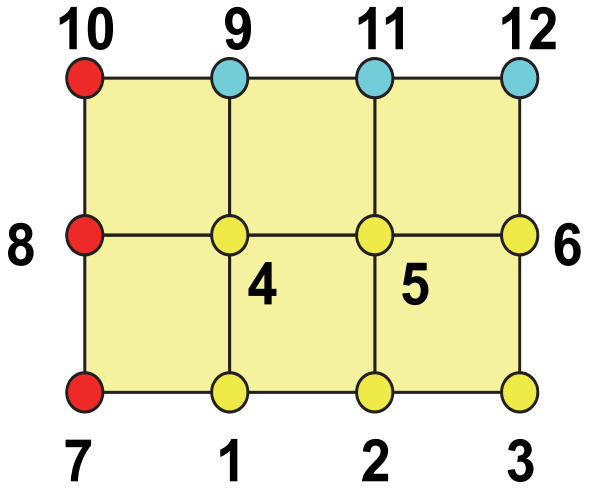








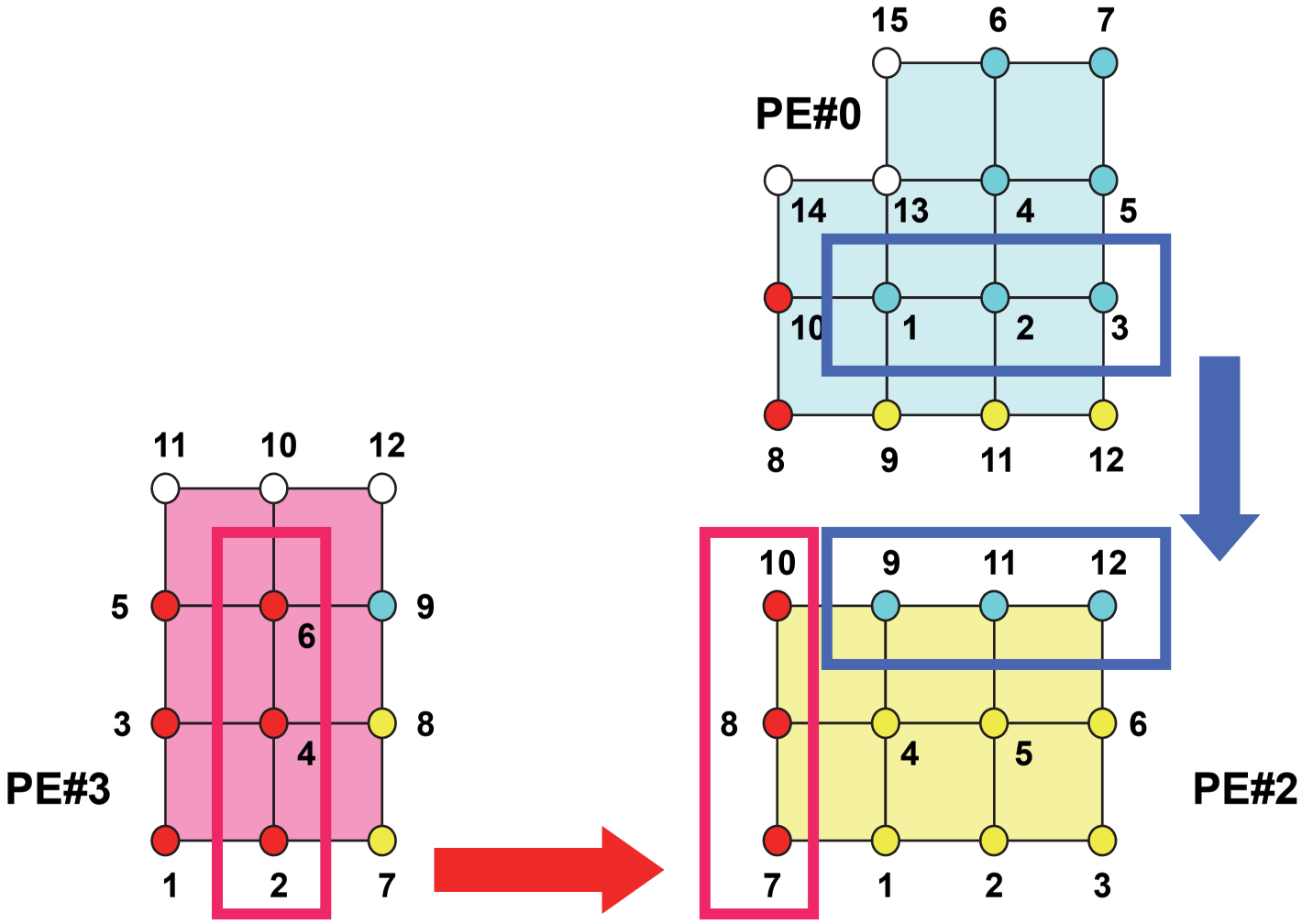
Description of Distributed Local Data



- Internal/External Points
  - Numbering: Starting from internal pts, then external pts after that
- Neighbors
  - Shares overlapped meshes
  - Number and ID of neighbors
- External Points
  - From where, how many, and which external points are received/imported ?
- Boundary Points
  - To where, how many and which boundary points are sent/exported ?

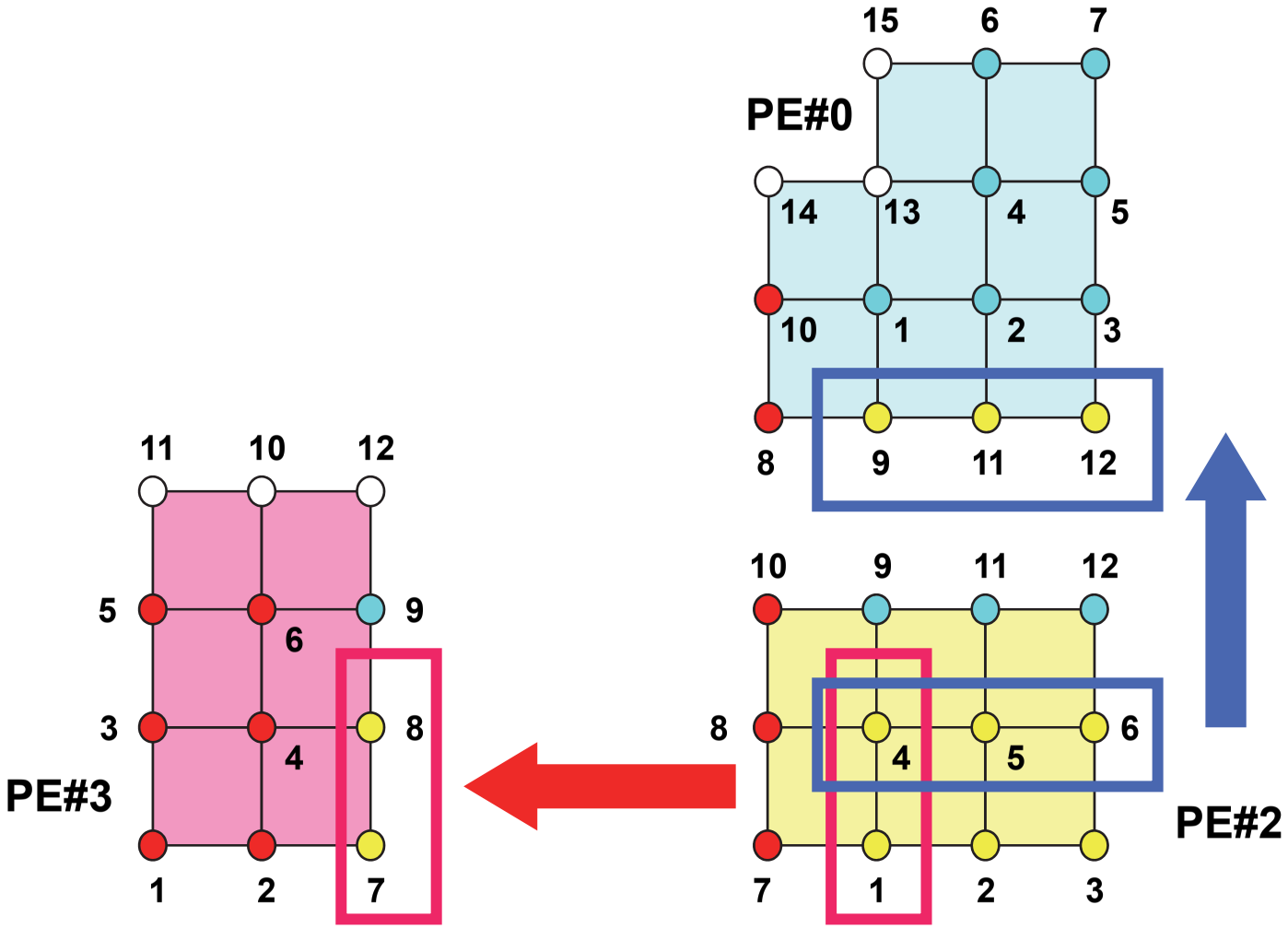
External Nodes（外点）: RECEIVE

PE#2 : receive information for “external nodes”



Boundary Nodes（境界点）：SEND

PE#2 : send information on “boundary nodes”



Distributed Local Data Structure for Parallel Computation

- Distributed local data structure for domain-to-doain communications has been introduced, which is appropriate for such applications with sparse coefficient matrices (e.g. FDM, FEM, FVM etc.).
  - SPMD
  - Local Numbering: Internal pts to External pts
  - Generalized communication table

- Everything is easy, if proper data structure is defined:
  - Values at boundary pts are copied into sending buffers
  - Send/Recv
  - Values at external pts are updated through receiving buffers