

有限要素法：復習+並列データ構造

2011年度冬学期

中島 研吾

科学技術計算Ⅱ（4820-1028）・コンピュータ科学特別講義（4810-1205）

（並列有限要素法）

- ガラーキン法による一次元弾性問題の解法
- 連立一次方程式の解法
 - 共役勾配法
 - 前処理手法
- 疎行列格納法
- プログラムの内容
- 並列化

キーワード

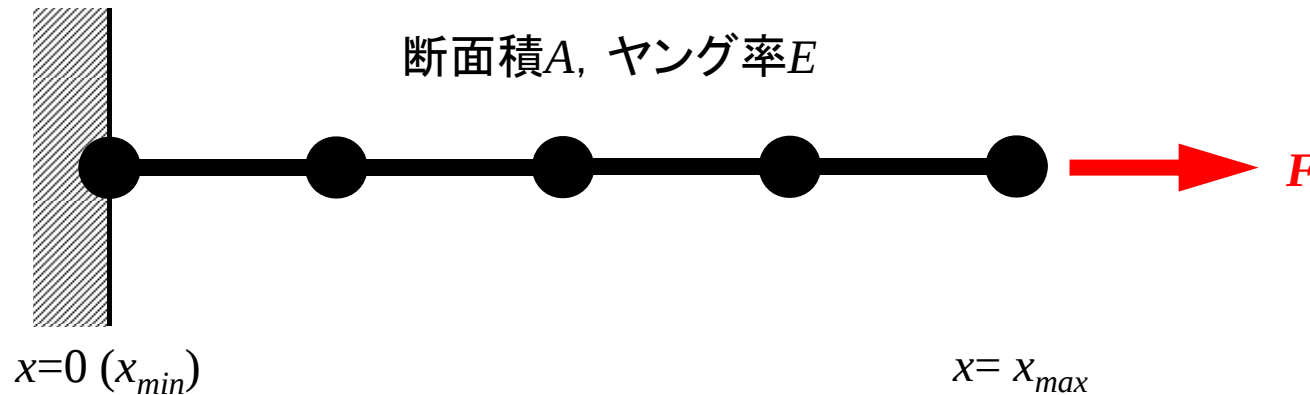
- 一次元弾性問題
- ガラーキン法
- 線形一次要素
- 前処理付共役勾配法

対象とする問題： 一次元弾性体（トラス）



- x 方向にのみ自由度（変位 u ）
 - 一様な：断面積 A , ヤング率 E
 - 境界条件
 - $x=0$: $u=0$ （固定）
 - $x=x_{max}$: 大きさ F の力（軸力）
- 自重によるたわみ等はナシ：バネと同じ

対象とする問題： 一次元弾性体（トラス）



つりあい式

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

ひずみ～変位
関係式

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

ひずみ～応力
関係式

$$\sigma_x = E \varepsilon_x$$



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

変位 u を自由度とする
支配方程式

計算の手順

- まず変位を求め

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

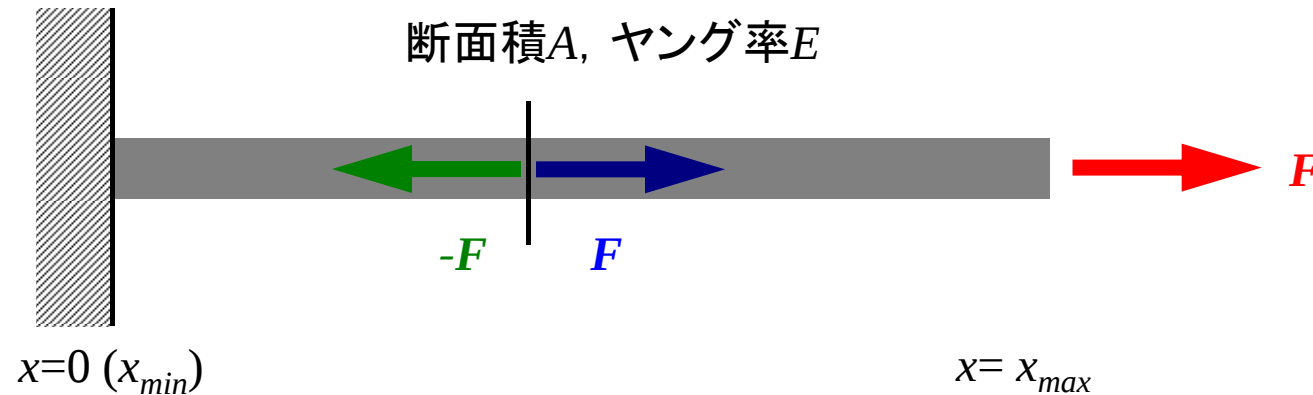
- 「ひずみ」を計算し

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

- 「応力」を計算する

$$\sigma_x = E \varepsilon_x$$

予測される結果



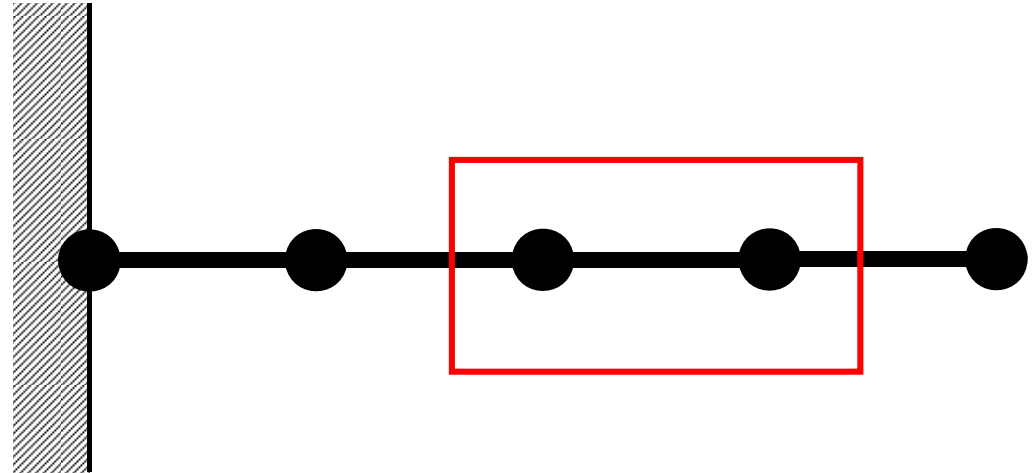
- $x_{max}=10, F=5, A=2, E=10$.とすると
- つりあい：任意断面で一様な軸力
 - 一様な応力 $\sigma = F/A = 2.5$
 - すなわち一様なひずみ $\varepsilon = \sigma/E = 0.25$
 - 各要素の長さが同じであれば、各要素一様な変形（変形率 $= \varepsilon$ ） $x=x_{max}$ での変位は $+2.5$

一次元線形要素 (1/4)

一次元線形要素

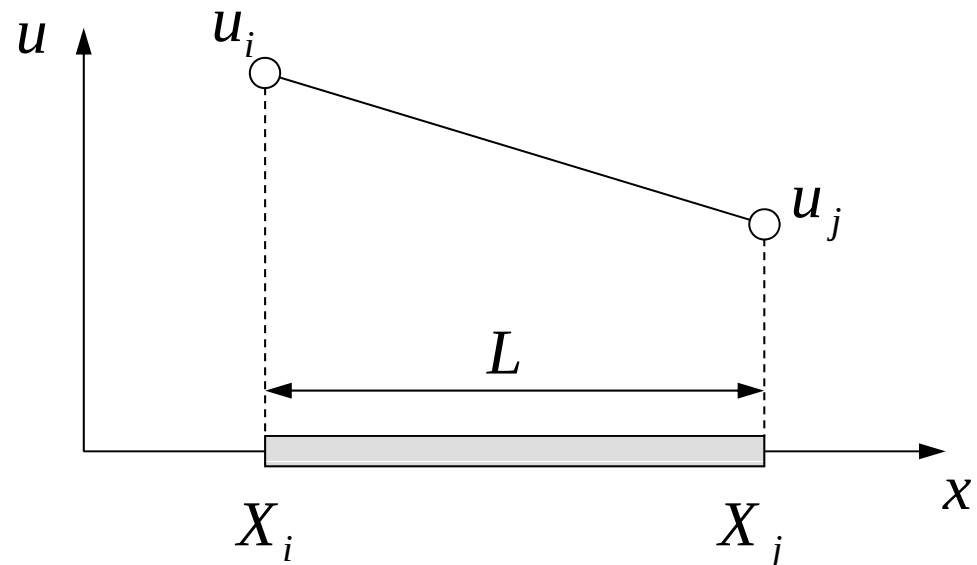
- 長さ L の両端に節点 (node) を持つ線分

- 節点 : node
- 要素 : element



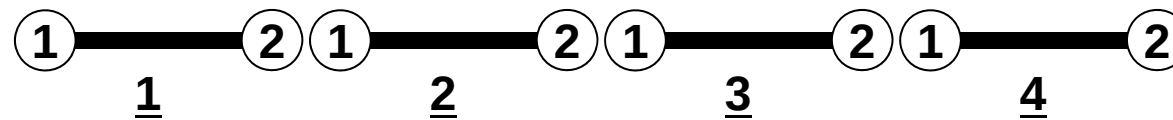
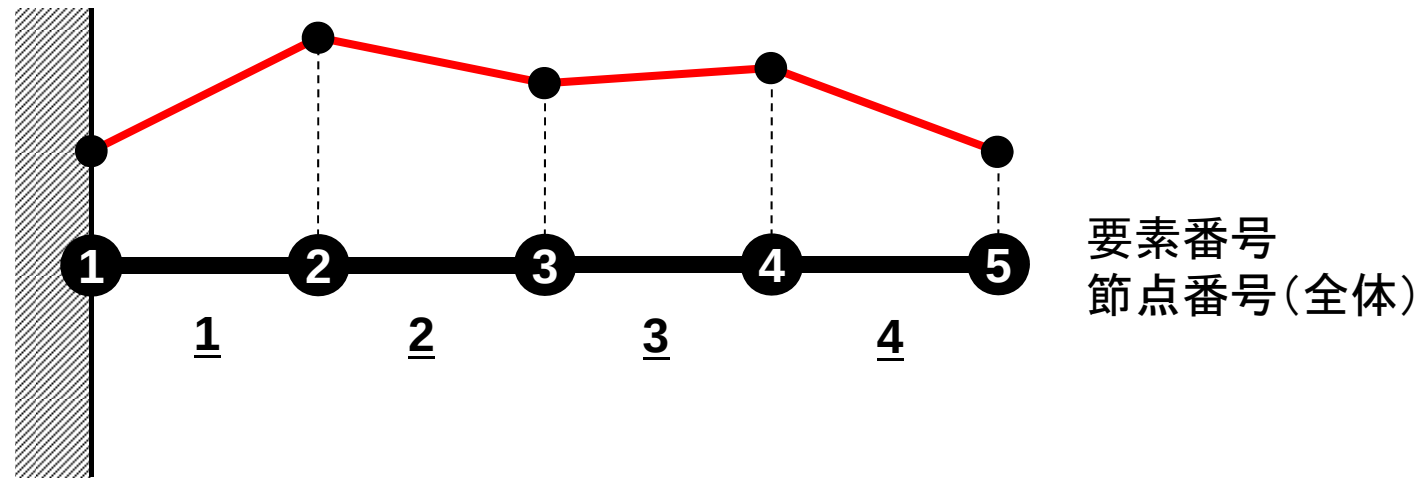
- 節点 i, j における変位を u_i, u_j
- 要素内での変位 u は以下のように表される (座標 x の一次関数, Piecewise Linear) :

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



Piecewise Linear

各要素内で「変位 u の分布」が線形



各要素における
「局所」節点番号

ひずみ, 応力(変位の勾配に比例)は要素内で一定
(節点で不連続となる可能性あり)

一次元線形要素：形状関数（2/4）

- 節点での条件から，係数は以下のように求められる：

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

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

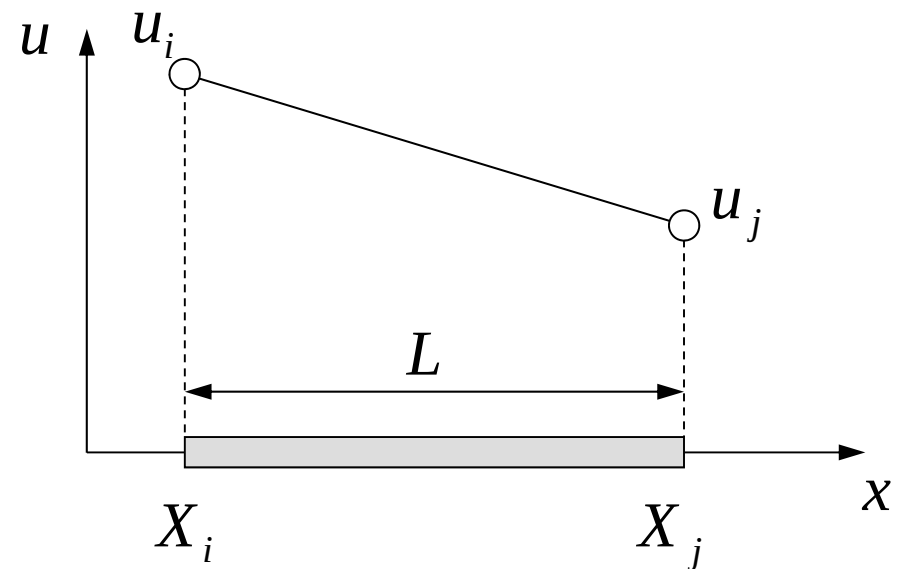
- 従って：

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

- 元の式に代入して，書き直すと以下のようなになる

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

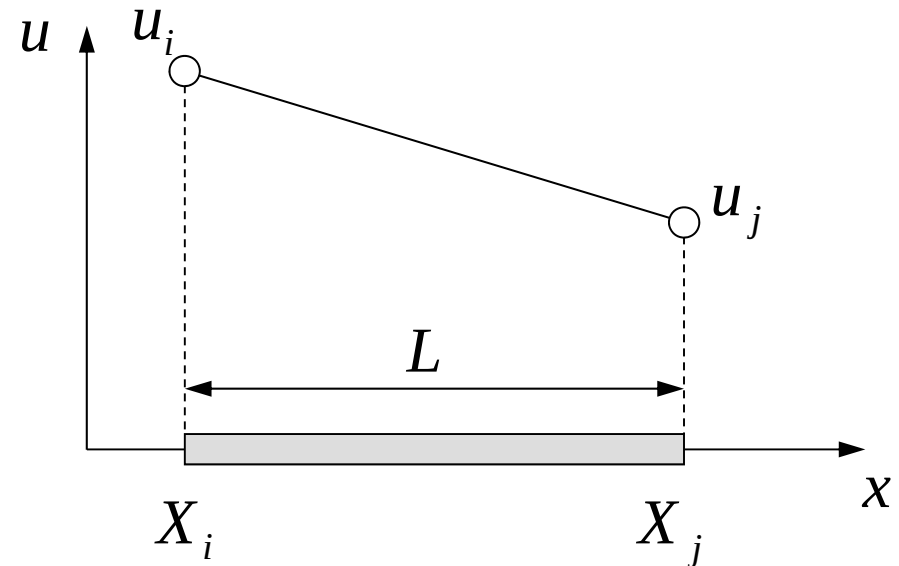
これらのxに関する一次式を形状関数 (shape function) または内挿関数 (interpolation function) と呼ぶ (N_i , N_j と表す)



一次元線形要素：形状関数（3/4）

- 形状関数 N_k は要素を構成する節点数と同じ数だけ存在する：
 - 位置座標のみの関数である
 - 「試行関数」の一種

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



- 形状関数の一次結合により要素内の変位を表す
 - 係数（＝未知数）が節点における変位

$$u = N_i u_i + N_j u_j \longleftrightarrow u_M = \sum_{i=1}^M a_i \Psi_i$$

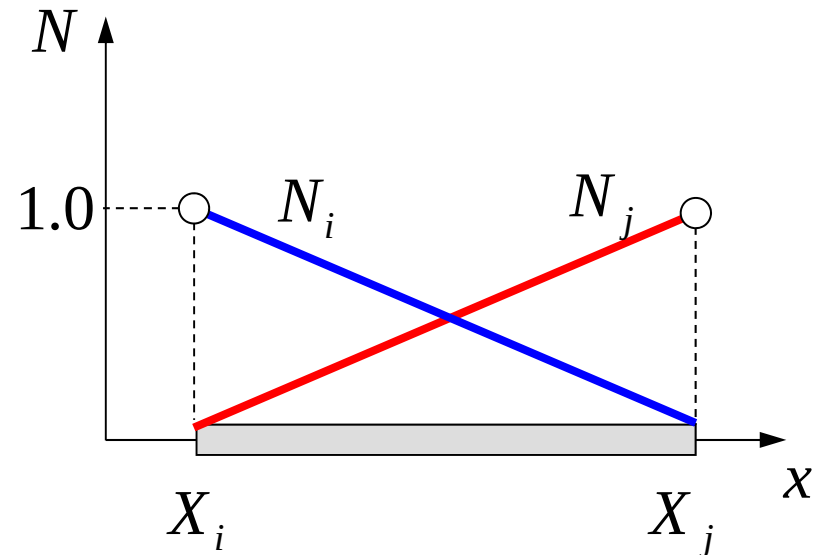
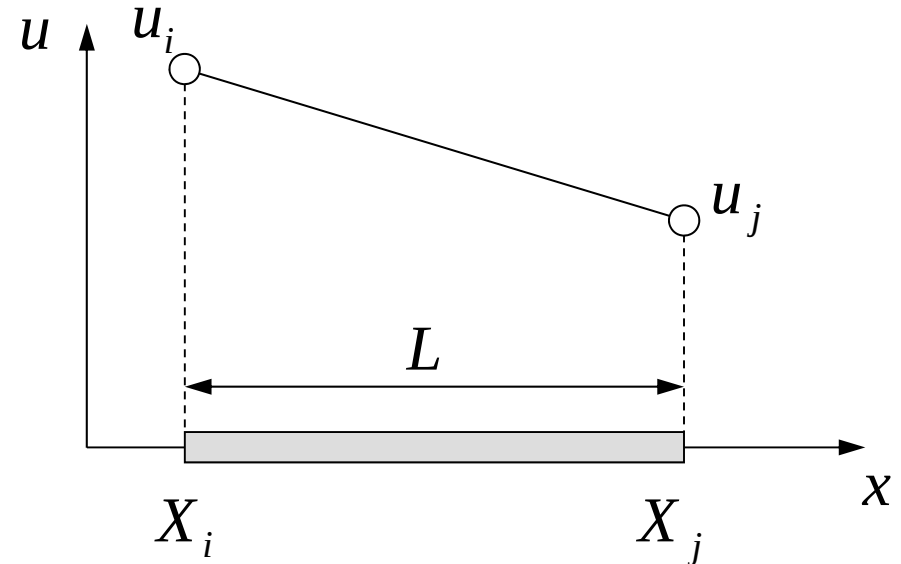
Ψ_i 領域, 境界において定義される, 位置座標のみ既知関数, 互いに独立である: 試行関数 (trial/test function) と呼ばれる。線形代数における基底 (basis) に相当する

a_i 係数 (未知数)

一次元線形要素：形状関数（4/4）

- 形状関数はある節点で1の値をとり，他の節点では必ず0の値をとる：

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

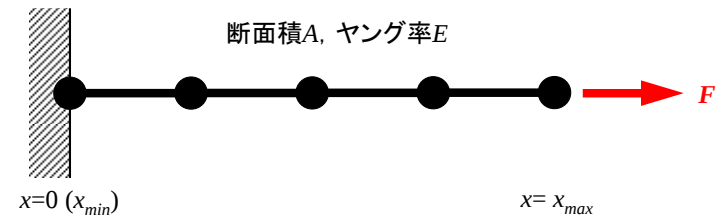


ガラーキン法の適用 (1/4)

- 以下のような一次元弾性力学方程式を考慮する：

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

$u = [N] \{\phi\}$ 要素内の変位分布
(マトリクス形式), 節点における変位を ϕ としてある。



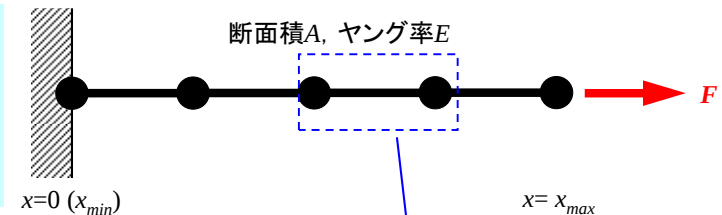
- ガラーキン法に従い, 重み関数を $[N]$ とすると, 各要素において以下の積分方程式が得られる：

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

ガラーキン法の適用 (2/4)

- 一次元のグリーンの定理

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

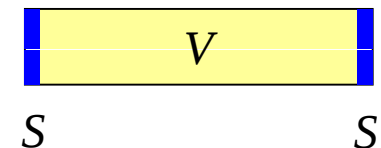


- これを前式の2階微分の部分に適用すると：

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

- これに以下を代入する：

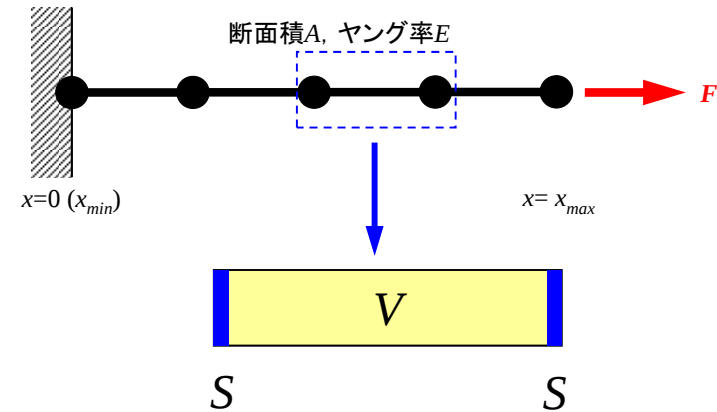
$$u = [N]\{\phi\} \quad \bar{\sigma} = E \frac{du}{dx} : \text{要素表面応力}$$



ガラーキン法の適用 (3/4)

- 更に体積力（物体力）の項 X を加えて次式が得られる：

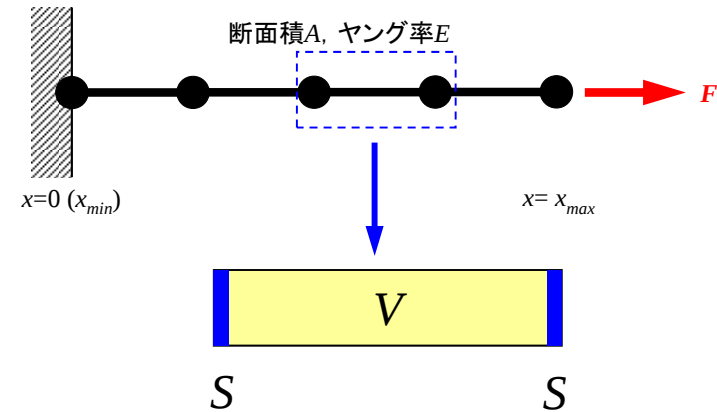
$$\begin{aligned}
 & - \int_V E \left(\frac{d[N]^T}{dx} \frac{d[N]}{dx} \right) dV \cdot \{\phi\} \\
 & + \int_S \bar{\sigma} [N]^T dS + \int_V X [N]^T dV = 0
 \end{aligned}$$



- この式を弱形式（weak form）と呼ぶ。元の微分方程式では2階の微分が含まれていたが、上式では、グリーンの定理によって1階微分に低減されている。
 - 弱形式によって近似関数（形状関数，内挿関数）に対する要求が弱くなっている：すなわち線形関数で2階微分の効果を記述できる。

ガラーキン法の適用 (4/4)

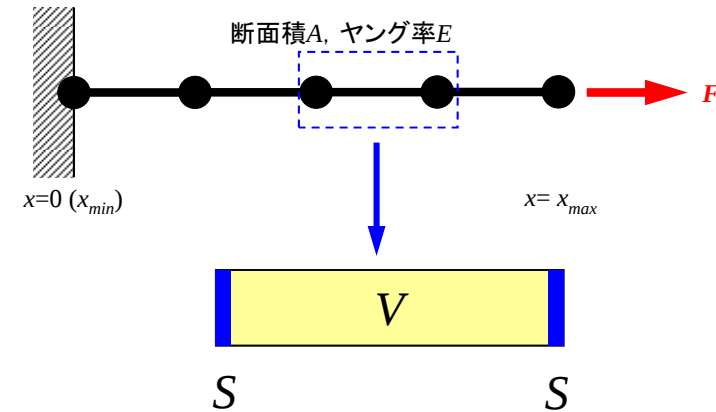
$$\begin{aligned}
 & - \int_V E \left(\frac{d[N]^T}{dx} \frac{d[N]}{dx} \right) dV \cdot \{\phi\} \\
 & + \int_S \bar{\sigma} [N]^T dS + \int_V X [N]^T dV = 0
 \end{aligned}$$



- この項は要素境界で相殺するため、領域境界における項のみが残る。

弱形式と境界条件

- 未知数の値が直接与えられる (Dirichlet)
 - 重み関数=0となる
 - 第一種境界条件
 - 基本境界条件
 - essential boundary condition
- 未知数の導関数が与えられる (Neumann)
 - 弱形式中で自然に考慮される
 - 第二種境界条件
 - 自然境界条件
 - natural boundary condition



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

$$\bar{\sigma} = E \frac{du}{dx} \text{ から得られる}$$

境界条件を考慮した弱形式：各要素

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

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

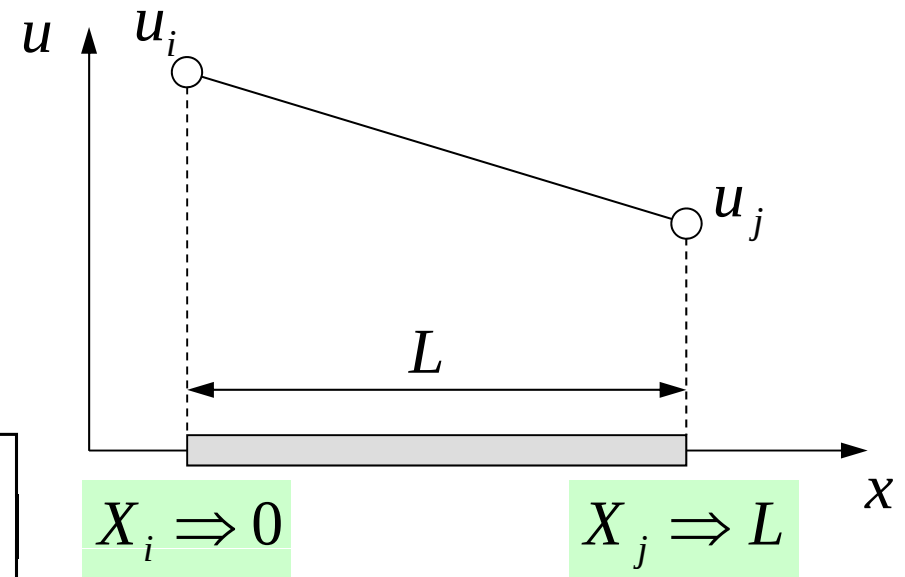
$$[f]^{(e)} = \int_V X [N]^T dV + \int_S \bar{\sigma} [N]^T dS$$

要素単位での積分：[k]

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

$$\begin{aligned} & \int_V E \left(\frac{d[N]^T}{dx} \frac{d[N]}{dx} \right) dV \\ &= E \int_0^L \begin{bmatrix} -1/L \\ 1/L \end{bmatrix} \begin{bmatrix} -1/L & 1/L \end{bmatrix} A dx \\ &= \frac{EA}{L^2} \int_0^L \begin{bmatrix} +1 & -1 \\ -1 & +1 \end{bmatrix} dx = \frac{EA}{L} \begin{bmatrix} +1 & -1 \\ -1 & +1 \end{bmatrix} \end{aligned}$$

A: 断面積, L: 要素長さ



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

要素単位での積分： $\{f\}$ (1/2)

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

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

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

物体力



A : 断面積, L : 要素長さ

要素単位での積分： $\{f\}$ (2/2)

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

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

$$\int_S \bar{\sigma} [N]^T dS = \bar{\sigma} A \Big|_{x=L} = \bar{\sigma} A \begin{Bmatrix} 0 \\ 1 \end{Bmatrix} \quad \text{表面力}$$

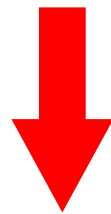


表面力がこの断面のみに
作用しているとする

全体方程式

- 要素単位の方程式を全体で足し合わせ,

$$[k]^{(e)} \{\phi\}^{(e)} = \{f\}^{(e)} \quad \text{要素マトリクス, 要素方程式}$$



$$[K] \cdot \{\Phi\} = \{F\} \quad \text{全体マトリクス, 全体方程式}$$

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

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

この連立一次方程式(全体方程式)
を解いてやればよい

ECCS2008ログイン, ディレクトリ作成, ファイルコピー un001.ecc.u-tokyo.ac.jp

ディレクトリ作成

```
>$ cd Documents          この下に作る  
>$ mkdir fem2 (好きな名前でもいい)  
>$ cd fem2
```

このディレクトリを本講義では **<\$fem2>** と呼ぶ
基本的にファイル類はこのディレクトリにコピー, 解凍する

一次元弾性解析コード

```
>$ cd <$fem2>  
>$ cp /home03/skengon/Documents/class/fem1/1d.tar .  
>$ tar xvf 1d.tar  
>$ cd 1d
```

T2K上では ディレクトリ作成, ファイルコピー

ディレクトリ作成

```
>$ mkdir fem2 (好きな名前でもいい)
```

```
>$ cd fem2
```

このディレクトリを本講義では **<\$T-fem2>** と呼ぶ
基本的にファイル類はこのディレクトリにコピー, 解凍する

一次元弾性解析コード

```
>$ cd <$fem2>
```

```
>$ cp /home/t000000/fem1/1d.tar .
```

```
>$ tar xvf 1d.tar
```

```
>$ cd 1d
```

実行

```
>$ cd <$fem2>/1d
>$ cc -O 1d.c          (or g95 -O 1d.f)
>$ ./a.out
```

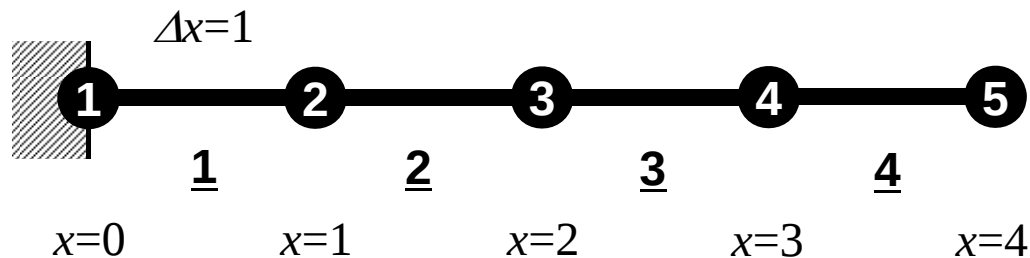
制御ファイル input.dat

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

NE (要素数)
 Δx (要素長さL), F, A, E
 反復回数 (CG法後述)
 CG法の反復打ち切誤差

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

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



要素番号
 節点番号(全体)

結果

```
>$ ./a.out
```

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

```
### DISPLACEMENT
```

各節点における変位(計算結果)

```
1 0.000000E+00
2 1.000000E+00
3 2.000000E+00
4 3.000000E+00
5 4.000000E+00
```

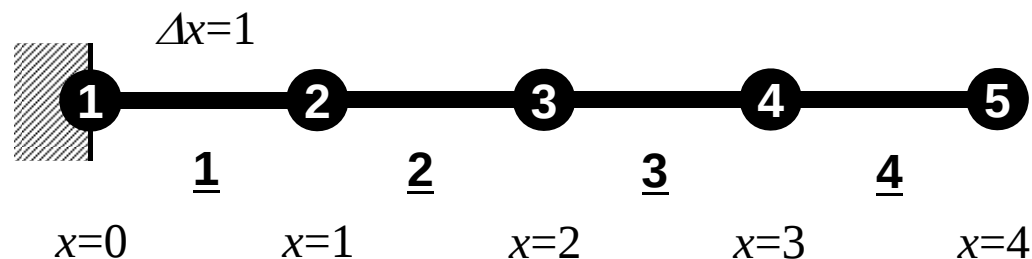
$$\sigma = \frac{F}{A} = \frac{1}{1} = 1$$

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

```
### STRESS
```

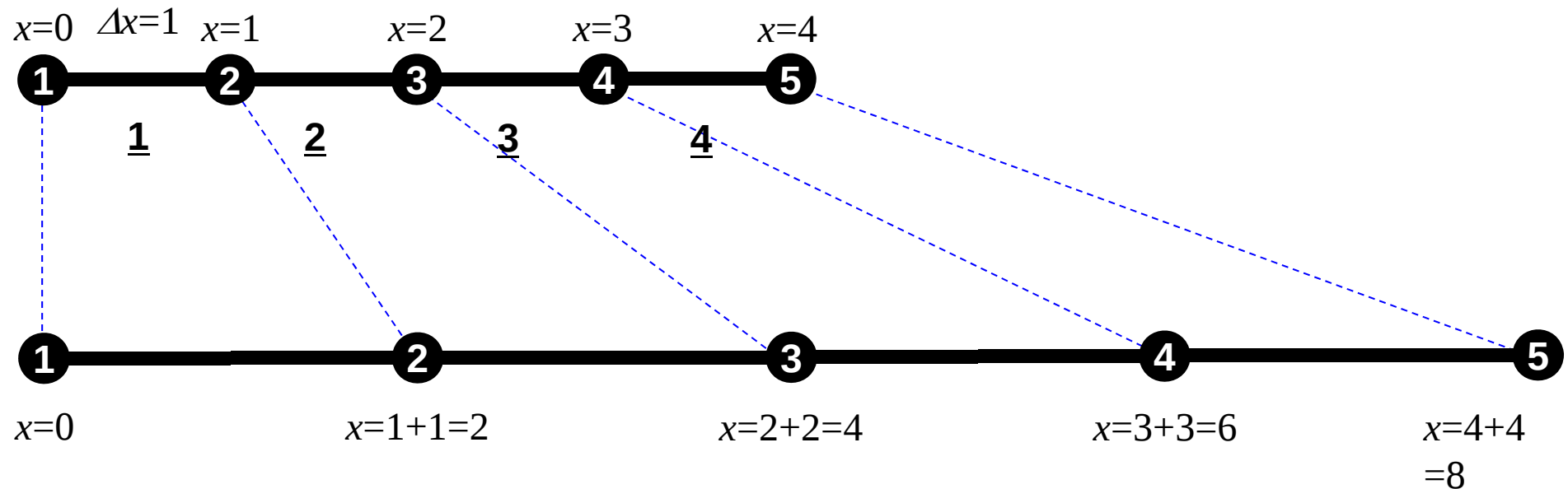
各「要素」における応力(計算結果, 解析解)

```
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
```



要素番号
節点番号(全体)

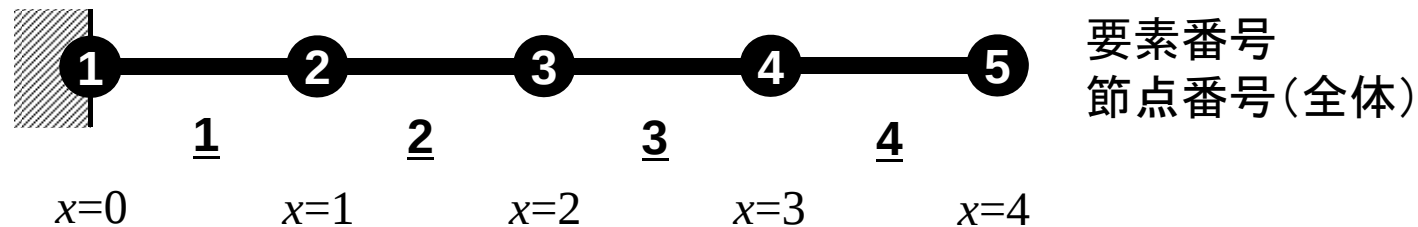
ひずみ=1（100%）：各要素の長さが
2倍に伸びている



###	DISPLACEMENT	各節点における変位(計算結果)
1	0.000000E+00	
2	1.000000E+00	
3	2.000000E+00	
4	3.000000E+00	
5	4.000000E+00	

要素方程式とその重ね合わせ (1/3)

- 4要素, 5節点の例題



- 要素1の $[k], \{f\}$ は以下のようになる :

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

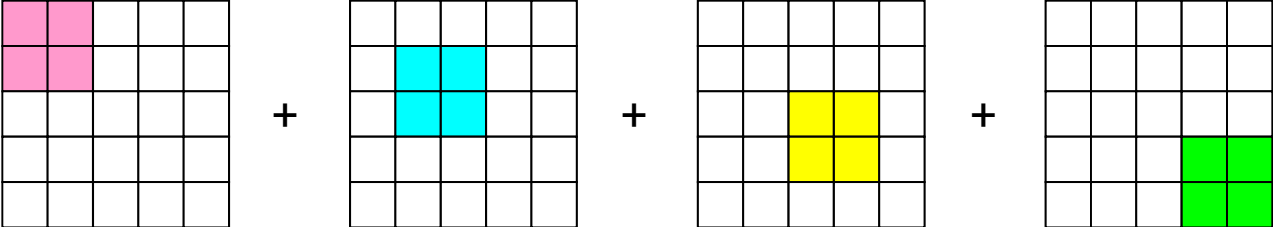
この問題では $X=0$

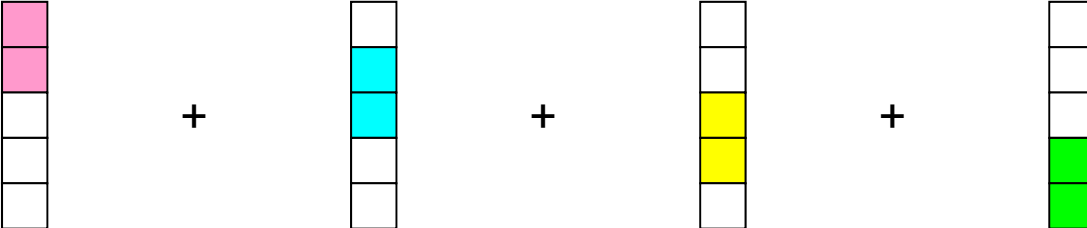
- 要素4については :

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

要素方程式とその重ね合わせ (2/3)

- これを順番に足していけばよい

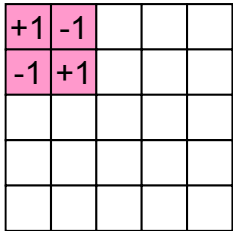
$$[K] = \sum_{e=1}^4 [k]^{(e)} =$$


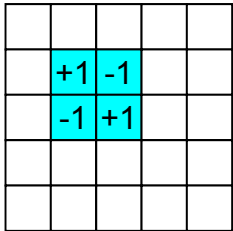
$$\{F\} = \sum_{e=1}^4 \{f\}^{(e)} =$$


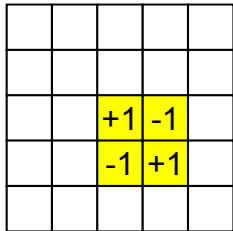
要素ごとに物性，寸法が
異なっても簡単に対応が可能

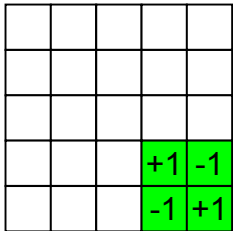
$$[k]^{(e)} = \frac{E^{(e)} A^{(e)}}{L^{(e)}} \begin{bmatrix} +1 & -1 \\ -1 & +1 \end{bmatrix}$$

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

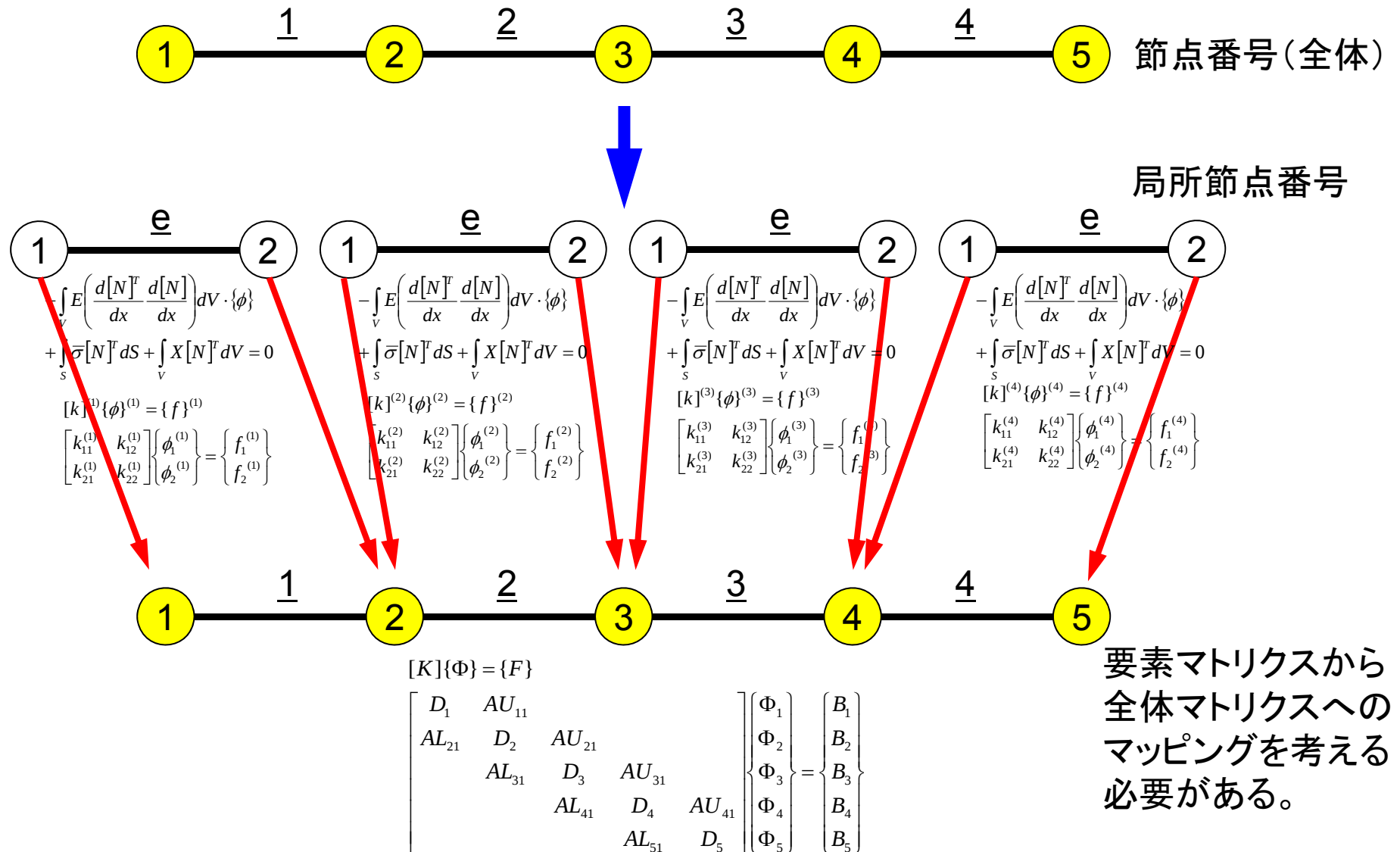

 $\times \frac{E^{(1)} A^{(1)}}{L^{(1)}}$

$+$

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


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

$+$

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

要素処理と全体処理



- ガラーキン法による一次元弾性問題の解法
- 連立一次方程式の解法
 - 共役勾配法
 - 前処理手法
- 疎行列格納法
- プログラムの内容
- 並列化

あとは出てきた全体方程式 (連立一次方程式)を解けばよい

- 多くの科学技術計算は、最終的に大規模線形方程式 $Ax=b$ を解くことに帰着される。
- 様々な手法が提案されている
 - 疎行列(sparse), 密行列(dense)
 - 直接法(direct), 反復法(iterative)
- 密行列(dense)
 - 境界要素法, スペクトル法など
- 疎行列(sparse): 0の部分が多い
 - FEM, FDMなど

代表的な反復法：共役勾配法

- Conjugate Gradient法, 略して「CG」法
 - 最も代表的な「非定常」反復法
- 対称正定値行列 (Symmetric Positive Definite: SPD)
 - 任意のベクトル $\{x\}$ に対して $\{x\}^T[A]\{x\} > 0$
 - 全対角成分 > 0 , 全固有値 > 0 , 全部分行列式 > 0 と同値
 - (ガラーキン法) 熱伝導, 弾性, ねじり: 本コードの場合もSPD
- アルゴリズム
 - 最急降下法 (Steepest Descent Method) の変種
 - $x^{(i)} = x^{(i-1)} + \alpha_i p^{(i)}$
 - $x^{(i)}$: 反復解, $p^{(i)}$: 探索ベクトル, α_i : 定数
 - 厳密解を y とするとき $\{x-y\}^T[A]\{x-y\}$ を最小とするような $\{x\}$ を求める。
 - 詳細は参考文献参照
 - 例えば: 森正武「数値解析(第2版)」(共立出版)

共役勾配法のアルゴリズム

```

Compute  $r^{(0)} = b - [A]x^{(0)}$ 
for  $i = 1, 2, \dots$ 
     $z^{(i-1)} = r^{(i-1)}$ 
     $\rho_{i-1} = r^{(i-1)} z^{(i-1)}$ 
    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

```

- 行列ベクトル積
- ベクトル内積
- ベクトル定数倍の加減

$x^{(i)}$: ベクトル

α_i : スカラー

前処理付き共役勾配法のアルゴリズム

```

Compute  $r^{(0)} = b - [A]x^{(0)}$ 
for  $i = 1, 2, \dots$ 
    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

```

対角スケーリング, 点ヤコビ前処理

- 前処理行列として, もとの行列の対角成分のみを取り出した行列を前処理行列 $[M]$ とする。
 - 対角スケーリング, 点ヤコビ (point-Jacobi) 前処理

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

- **solve $[M]z^{(i-1)} = r^{(i-1)}$** いう場合に逆行列を簡単に求めることができる。
- 簡単な問題では収束する。
- 1d.f, 1d.cはこの手法を使用している

- ガラーキン法による一次元弾性問題の解法
- 連立一次方程式の解法
 - 共役勾配法
 - 前処理手法
- 疎行列格納法
- プログラムの内容
- 並列化

有限要素法で得られるマトリクス

- 疎行列

- 0が多い

- $A(i,j)$ のように正方行列の全成分を記憶することは疎行列では非効率的

- 「密」行列向け

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

- 有限要素法: 非零非対角成分の数は高々「数百」規模

- 例えば未知数が 10^8 個あるとすると記憶容量(ワード数)は

- 正方行列: $O(10^{16})$

- 非零非対角成分数: $O(10^{10})$

- 非零成分のみ記憶するのが効率的

1d.f, 1d.cにおけるマトリクス関連変数

変数名	型	サイズ	内容
N	I	–	未知数総数
NPLU	I	–	連立一次方程式係数マトリクス非対角成分総数
Diag(:)	R	N	連立一次方程式係数マトリクス対角成分
U(:)	R	N	連立一次方程式未知数ベクトル
Rhs(:)	R	N	連立一次方程式右辺ベクトル
Index(:)	I	0:N N+1	係数マトリクス非対角成分要素番号用一次元圧縮配列(非対角成分数)
Item(:)	I	NPLU	係数マトリクス非対角成分要素番号用一次元圧縮配列(非対角成分要素(列)番号)
AMat(:)	R	NPLU	係数マトリクス非対角成分要素番号用一次元圧縮配列(非対角成分)

非零非対角成分のみを格納する
Compressed Row Storage法を使用している。

行列ベクトル積への適用

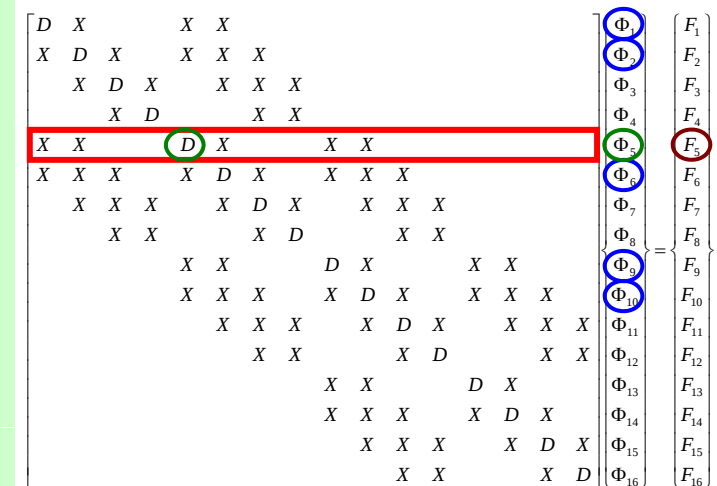
(非零) 非対角成分のみを格納, 疎行列向け方法
Compressed Row Storage (CRS)

Diag (i) 対角成分(実数, $i=1, N$)
 Index(i) 非対角成分に関する一次元配列(通し番号)
 (整数, $i=0, N$)
 Item(k) 非対角成分の要素(列)番号
 (整数, $k=1, \text{index}(N)$)
 AMat(k) 非対角成分
 (実数, $k=1, \text{index}(N)$)

$$\{Y\} = [A] \{X\}$$

```

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
  
```



行列ベクトル積：密行列⇒とても簡単

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

$$\{Y\} = [A] \{X\}$$

```
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] 対角成分(実数, [N])
 Index[i] 非対角成分に関する一次元配列
 (通し番号)(整数, [N+1])
 Item[k] 非対角成分の要素(列)番号
 (整数, [Index[N]])
 AMat[k] 非対角成分
 (実数, [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]];
    }
}
  
```

- ガラーキン法による一次元弾性問題の解法
- 連立一次方程式の解法
 - 共役勾配法
 - 前処理手法
- 疎行列格納法
- プログラムの内容
- 並列化

有限要素法の処理: プログラム

- 初期化
 - 制御変数読み込み
 - 座標読み込み⇒要素生成
 - 配列初期化(全体マトリクス, 要素マトリクス)
 - 要素⇒全体マトリクスマッピング
- マトリクス生成
 - 要素単位の処理
 - 要素マトリクス計算
 - 全体マトリクスへの重ね合わせ
 - 境界条件の処理
- 連立一次方程式
 - 共役勾配法(CG)
- 応力計算

プログラム: 1d.c(1/7)

諸変数

```
/*  
// 1D Solid Mechanics for Truss Elements solved by  
// CG (Conjugate Gradient) Method  
//  
//  $d/dx(E dU/dx) + F = 0$   
//  $U=0@x=0$   
*/  
  
#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, *Icelnod;  
    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;
```

変数表(1/2)

変数名	種別	サイズ	I/O	内 容
NE	I		I	要素数
N	I		O	節点数
NPLU	I		O	非零非対角成分数
IterMax	I		I	最大反復回数
errno	I		O	エラー戻り値
R, Z, Q, P, DD	I		O	CG法ベクトル名
dX	R		I	要素長さ
Resid	R		O	CG法残差
Eps	R		I	CG法反復打ち切り残差
Area	R		I	要素断面積
F	R		I	軸力F（境界条件）
Young	R		I	ヤング率
X1, X2, U1, U2	R		O	局所節点1,2の座標, 変位

変数表(2/2)

変数名	種別	サイズ	I/O	内 容
DL, Ck	R		O	要素マトリクス計算用定数
Strain, Stress	R		O	要素ひずみ, 要素応力
X	R	N	O	節点座標
U	R	N	O	節点変位
Rhs	R	N	O	右辺ベクトル
Diag	R	N	O	全体マトリクス：対角成分
W	R	[N] [4]	O	CG法のwork配列
Amat	R	NPLU	O	全体マトリクス：非零非対角成分
Index	I	N+1	O	全体マトリクス：各行の非零非対角成分数
Item	I	NPLU	O	全体マトリクス：列番号
IceInod	I	2*NE	O	各要素節点番号
Kmat	R	[2] [2]	O	要素マトリクス[k]
Emat	R	[2] [2]	O	要素マトリクス

プログラム: 1d.c (2/7)

初期設定, 配列宣言

```
/*
// +-----+
// |  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));

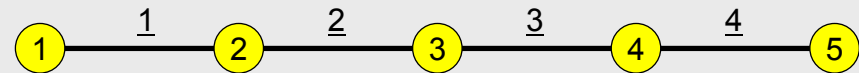
AMat = calloc(2*N-2, sizeof(double));

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

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

制御ファイル input.dat

4	NE (要素数)
1.0 1.0 1.0 1.0	Δx (要素長さL), F, A, E
100	反復回数
1.e-8	CG法の反復打切誤差



NE : 要素数
N : 節点数 (=NE+1)

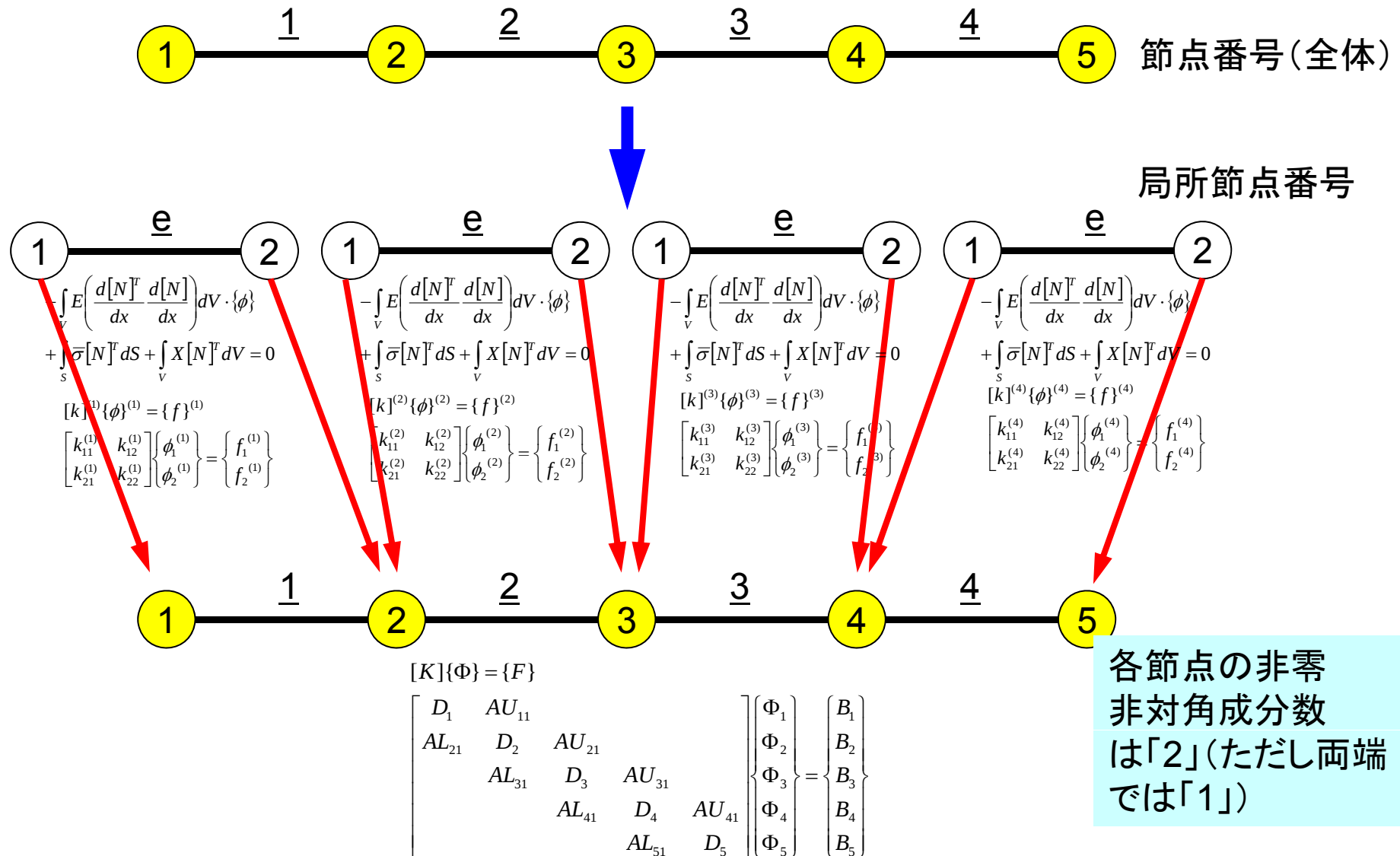
プログラム: 1d.c (2/7)

初期設定, 配列宣言

```
/*  
// +-----+  
// |  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 : 非零非対角成分
Item : 対応する列番号

要素処理と全体処理



プログラム: 1d.c (2/7)

初期設定, 配列宣言

```

/*
// +-----+
// |  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 : 非零非対角成分

Item : 対応する列番号

各節点の非零非対角成分数は「2」
(ただし両端では「1」)

総数 : $2*(N-2)+1+1 = 2*N-2$

プログラム: 1d.c (3/7)

配列宣言(続き), 初期化

```
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;
```

プログラム: 1d.c(3/7)

配列宣言(続き), 初期化

```
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 : 各節点の座標

プログラム: 1d.c (3/7)

配列宣言(続き), 初期化

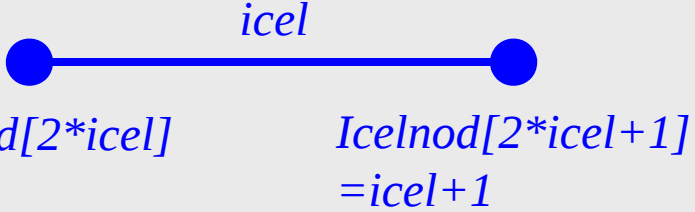
```

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;

```



$icel$
 $Icelnod[2*icel] = icel$ $Icelnod[2*icel+1] = icel+1$

プログラム: 1d.c(3/7)

配列宣言(続き), 初期化

```

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]

プログラム: 1d.c(4/7)

全体マトリクス: 非零非対角成分に対応する列番号

```

/*
// +-----+
// | 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;
    }
}

```

各節点の非零非対角成分数は「2」
(ただし両端では「1」)

総数: $2*(N-2)+1+1 = 2*N-2$
 $Index[N] = 2*N-2 = NPLU$

		非対角成分数	
0	1.1 ⊙ ①	2	Index[0] = 0
1	3.6 ①	4	Index[1] = 2
2	5.7 ②	2	Index[2] = 6
3	9.8 ③	3	Index[3] = 8
4	11.5 ④	4	Index[4] = 11
5	12.4 ⑤	2	Index[5] = 15
6	23.1 ⑥	4	Index[6] = 17
7	51.3 ⑦	4	Index[7] = 21
			Index[8] = 25

Index[i]~Index[i+1]-1番目がi行目の非対角成分

プログラム: 1d.c(4/7)

全体マトリクス: 非零非対角成分に対応する列番号

```

/*
// +-----+
// | 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];
}

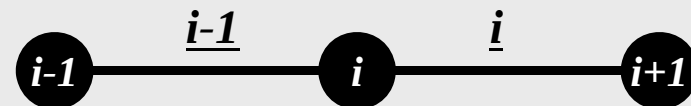
NPLU= Index[N];

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;
    }
}

```

		非対角 成分数	
0	1.1 ⊙	2	Index[0]= 0
1	3.6 ①	4	Index[1]= 2
2	5.7 ②	2	Index[2]= 6
3	9.8 ③	3	Index[3]= 8
4	11.5 ④	4	Index[4]= 11
5	12.4 ⑤	2	Index[5]= 15
6	23.1 ⑥	4	Index[6]= 17
7	51.3 ⑦	4	Index[7]= 21
			Index[8]= 25

Index[i]~Index[i+1]-1番目がi行目の非対角成分



プログラム: 1d.c (5/7)

全体マトリクス生成: 要素マトリクス $\Rightarrow$ 全体マトリクス

```

/*
// +-----+
// | 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);

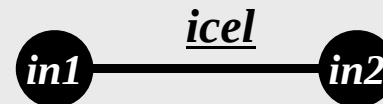
    Ck= Area*Young/DL;
    Emat[0][0]= Ck*Kmat[0][0];
    Emat[0][1]= Ck*Kmat[0][1];
    Emat[1][0]= Ck*Kmat[1][0];
    Emat[1][1]= Ck*Kmat[1][1];

    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];
}

```



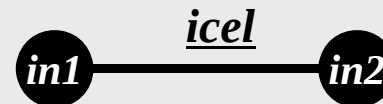
プログラム: 1d.c (5/7)

全体マトリクス生成: 要素マトリクス $\Rightarrow$ 全体マトリクス

```

/*
// +-----+
// | 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];

```

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

```

    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];

```

```

}

```

プログラム: 1d.c (5/7)

全体マトリクス生成: 要素マトリクス $\Rightarrow$ 全体マトリクス

```

/*
// +-----+
// | 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);

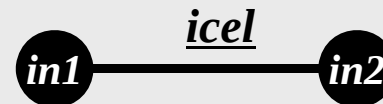
    Ck= Area*Young/DL;
    Emat[0][0]= Ck*Kmat[0][0];
    Emat[0][1]= Ck*Kmat[0][1];
    Emat[1][0]= Ck*Kmat[1][0];
    Emat[1][1]= Ck*Kmat[1][1];

    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];
}

```



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

プログラム: 1d.c (5/7)

全体マトリクス生成: 要素マトリクス $\Rightarrow$ 全体マトリクス

```

/*
// +-----+
// | 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);

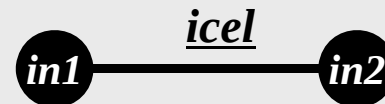
    Ck= Area*Young/DL;
    Emat[0][0]= Ck*Kmat[0][0];
    Emat[0][1]= Ck*Kmat[0][1];
    Emat[1][0]= Ck*Kmat[1][0];
    Emat[1][1]= Ck*Kmat[1][1];

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

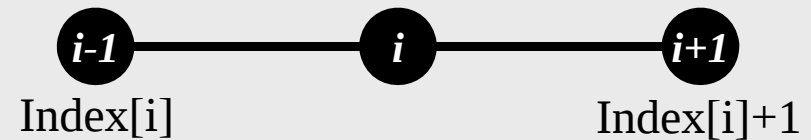
    if (icel==0) {k1=Index[in1];
                  }else {k1=Index[in1]+1;}
    k2=Index[in2];

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

```



「i」 行の非対角成分：
Index[i], Index[i]+1



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

プログラム: 1d.c (6/7)

境界条件

```
/*  
// +-----+  
// | 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;
```

対象とする問題： 一次元弾性体（トラス）



- x 方向にのみ自由度（変位 u ）
 - 一様な：断面積 A , ヤング率 E
 - 境界条件
 - $x=0$: $u=0$ （固定）
 - $x=x_{max}$: 大きさ F の力（軸力）
- 自重によるたわみ等はナシ：バネと同じ

$x=0$ で成立する方程式

$$u_1=0$$



- x 方向にのみ自由度（変位 u ）
 - 一様な：断面積 A , ヤング率 E
 - 境界条件
 - $x=0$: $u=0$ （固定）
 - $x=x_{max}$: 大きさ F の力（軸力）
- 自重によるたわみ等はナシ：バネと同じ

プログラム: 1d.c (6/7)

境界条件

```

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

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

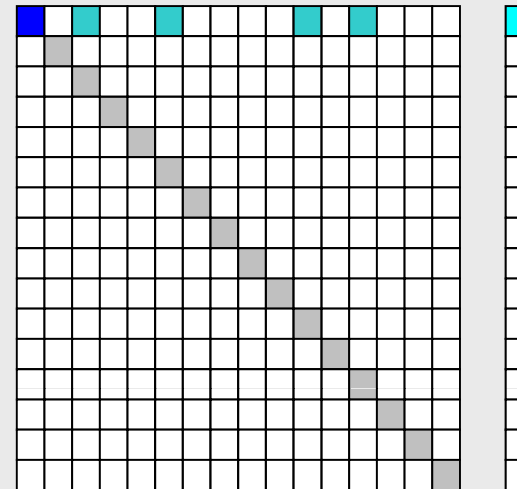
    for (k=0;k<NPLU;k++) {
        if (Item[k]==0) {AMat[k]=0.0;
        }}

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

```

$u_1=0$

対角成分=1, 右辺=0, 非対角成分=0



プログラム: 1d.c (6/7)

境界条件

```

/*
// +-----+
// | 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;
        }}

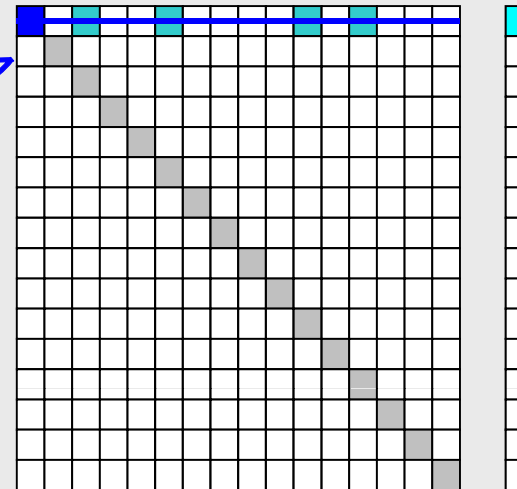
/* X=Xmax */
i=N-1;
Rhs[i]= F;

```

 $u_1=0$

対角成分=1, 右辺=0, 非対角成分=0

ゼロクリア



プログラム: 1d.c (6/7)

境界条件

```

/*
// +-----+
// | 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;
        }}

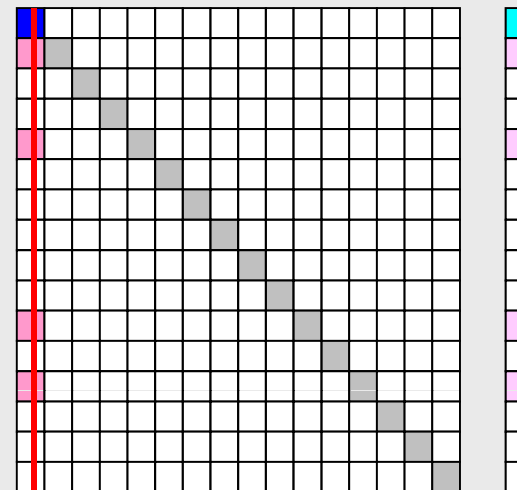
/* X=Xmax */
i=N-1;
Rhs[i]= F;

```

行列の対称性を保つため、第一種境界条件を適用している節点に対応する「列」を、右辺に移項して消去する(今の場合は非対角成分を0にするだけで良い)

$u_1=0$
対角成分=1, 右辺=0, 非対角成分=0

消去, ゼロクリア



プログラム: 1d.c (6/7)

境界条件

```

/*
// +-----+
// | 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;
        }}

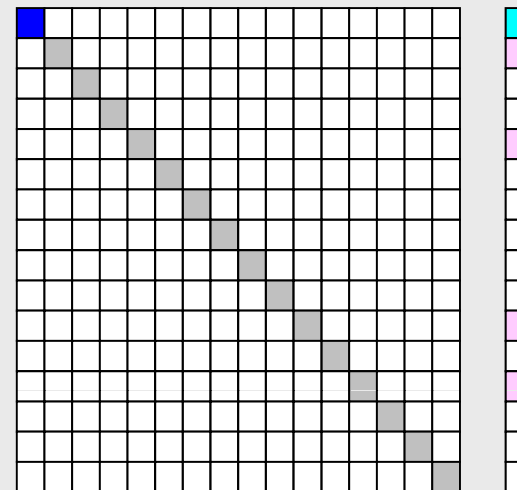
/* X=Xmax */
i=N-1;
Rhs[i]= F;

```

行列の対称性を保つため、第一種境界条件を適用している節点に対応する「列」を、右辺に移項して消去する(今の場合は非対角成分を0にするだけで良い)

$u_1=0$
対角成分=1, 右辺=0, 非対角成分=0

消去, ゼロクリア



第一種境界条件が $u \neq 0$ の場合

```
/*
// +-----+
// | 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;
    }
  }
}
```

行列の対称性を保つため, 第一種境界条件を適用している節点に対応する「列」を, 右辺に移項して消去する

$$Diag_j u_j + \sum_{k=Index[j]}^{Index[j+1]-1} AMat_k u_{Item[k]} = Rhs_j$$

プログラム: 1d.c (6/7)

境界条件

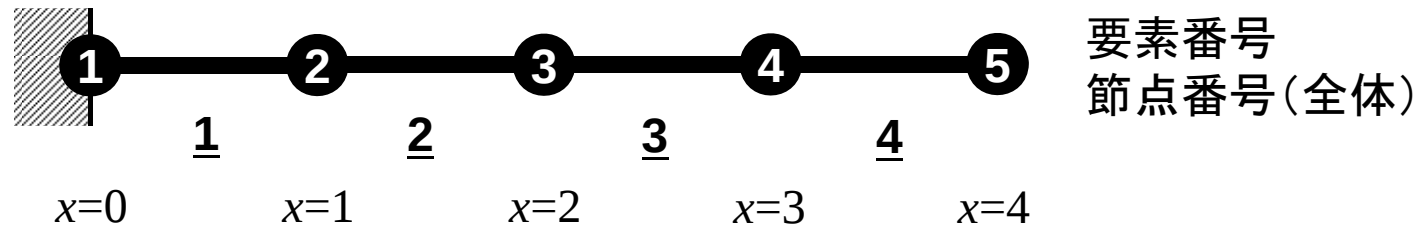
```
/*
// +-----+
// | BOUNDARY conditions |
// +-----+
*/

/* 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;
```

要素方程式とその重ね合わせ



- 要素4については:

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

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

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

前処理付き共役勾配法

Preconditioned Conjugate Gradient Method (CG)

```

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

```

前処理: 対角スケーリング

対角スケーリング, 点ヤコビ前処理

- 前処理行列として, もとの行列の対角成分のみを取り出した行列を前処理行列 $[M]$ とする。
 - 対角スケーリング, 点ヤコビ (point-Jacobi) 前処理

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

- **solve** $[M]z^{(i-1)} = r^{(i-1)}$ いう場合に逆行列を簡単に求めることができる。

CGソルバー(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];  
    }
```

対角成分の逆数(前処理用)
その都度、除算をすると効率が
悪いため、予め配列に格納

CGソルバー(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];
}

```

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

```

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

```

プログラム: 1d.c(7/7)

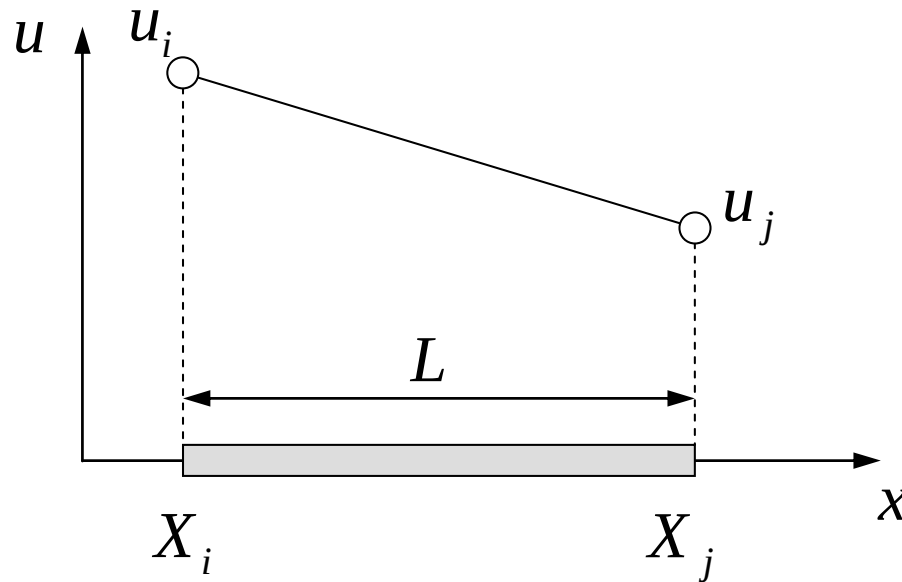
要素応力計算

```
/*  
// +-----+  
// | 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);  
}
```

要素内応力

一次元線形要素の場合は一定

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



$$u = N_i u_i + N_j u_j$$

$$\begin{aligned} \varepsilon &= \frac{du}{dx} = \frac{d}{dx} (N_i u_i + N_j u_j) \\ &= \frac{dN_i}{dx} u_i + \frac{dN_j}{dx} u_j \\ &= \left(-\frac{1}{L} \right) u_i + \left(\frac{1}{L} \right) u_j = \boxed{\frac{u_j - u_i}{L}} \end{aligned}$$

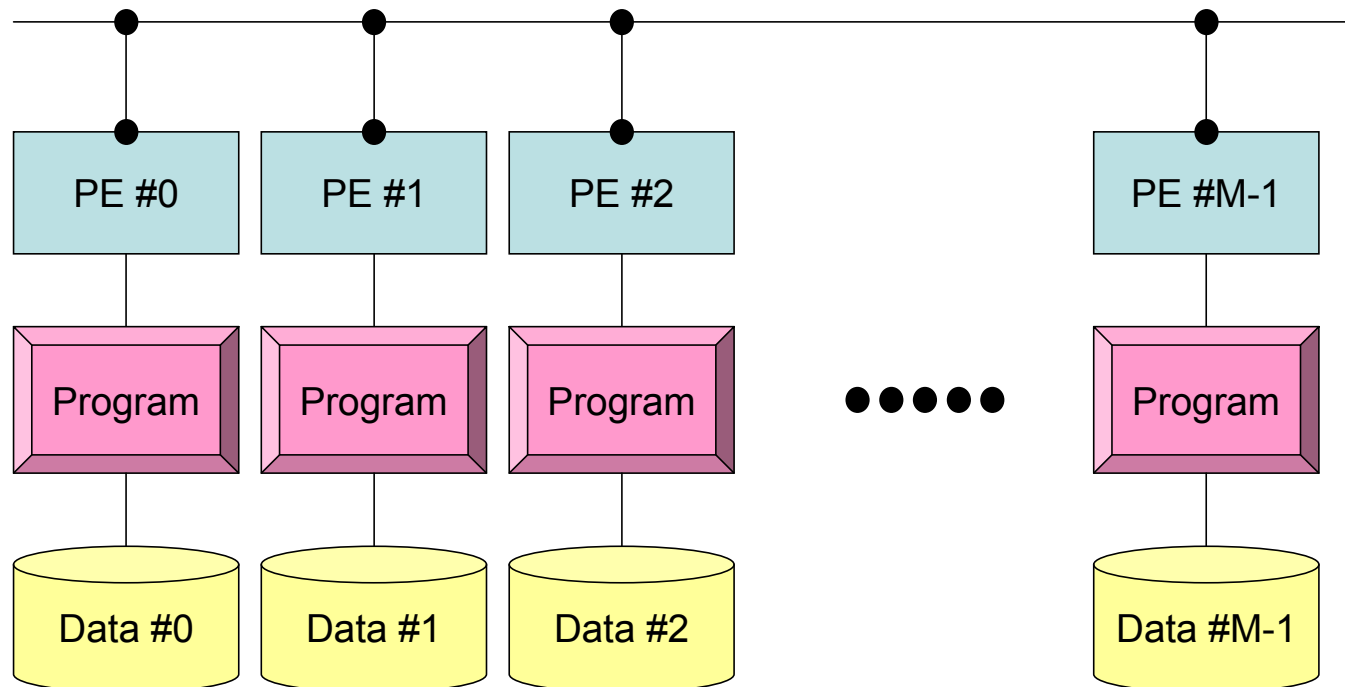
有限要素法の処理: プログラム

- 初期化
 - 制御変数読み込み
 - 座標読み込み⇒要素生成(N:節点数, NE:要素数)
 - 配列初期化(全体マトリクス, 要素マトリクス)
 - 要素⇒全体マトリクスマッピング(Index, Item)
- マトリクス生成
 - 要素単位の処理(do icel= 1, NE)
 - 要素マトリクス計算
 - 全体マトリクスへの重ね合わせ
 - 境界条件の処理
- 連立一次方程式
 - 共役勾配法(CG)
- 応力計算

- ガラーキン法による一次元弾性問題の解法
- 連立一次方程式の解法
 - 共役勾配法
 - 前処理手法
- 疎行列格納法
- プログラムの内容
- 並列化

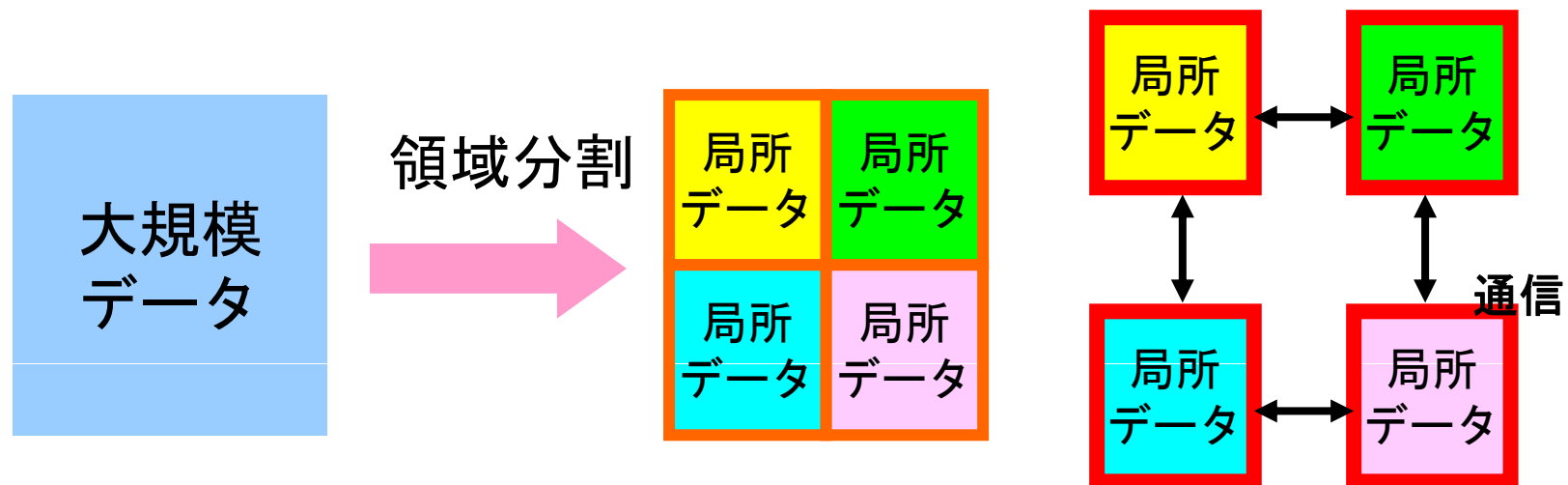
MPIによる並列化: SPMD

- Single Program/Instruction Multiple Data
- 基本的に各プロセスは「同じことをやる」が「データが違う」
 - 大規模なデータを分割し、各部分について各プロセス(プロセッサ)が計算する
- 全体データと局所データ, 全体番号と局所番号
- 通信以外は単体CPUと同じ, というのが理想

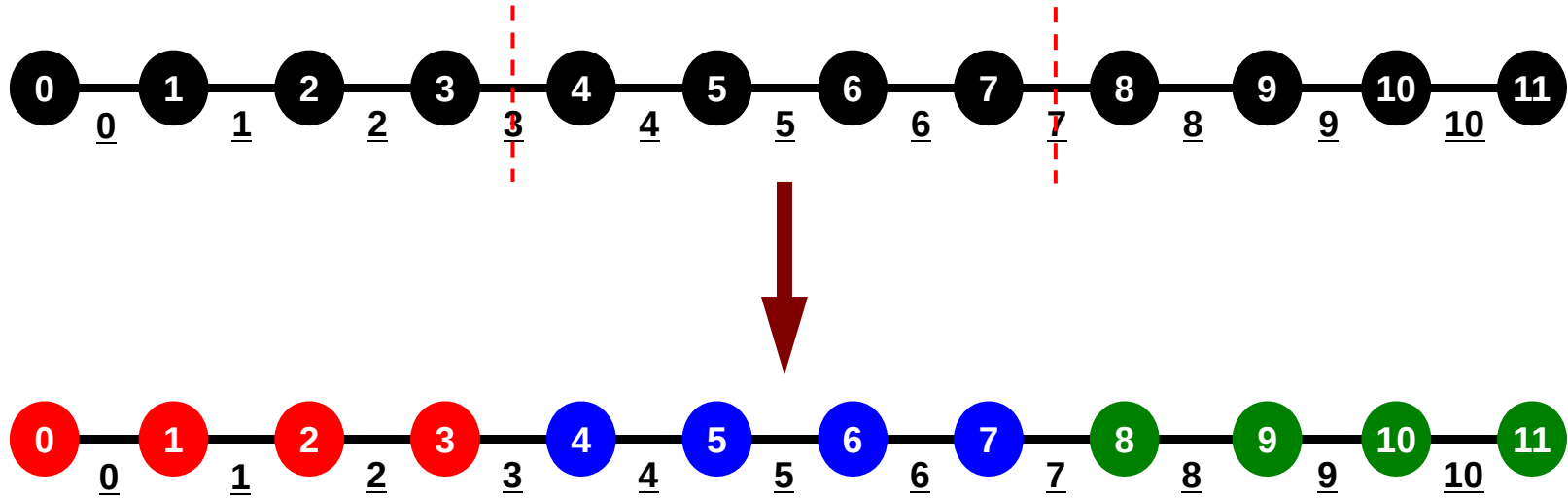


並列計算とは？

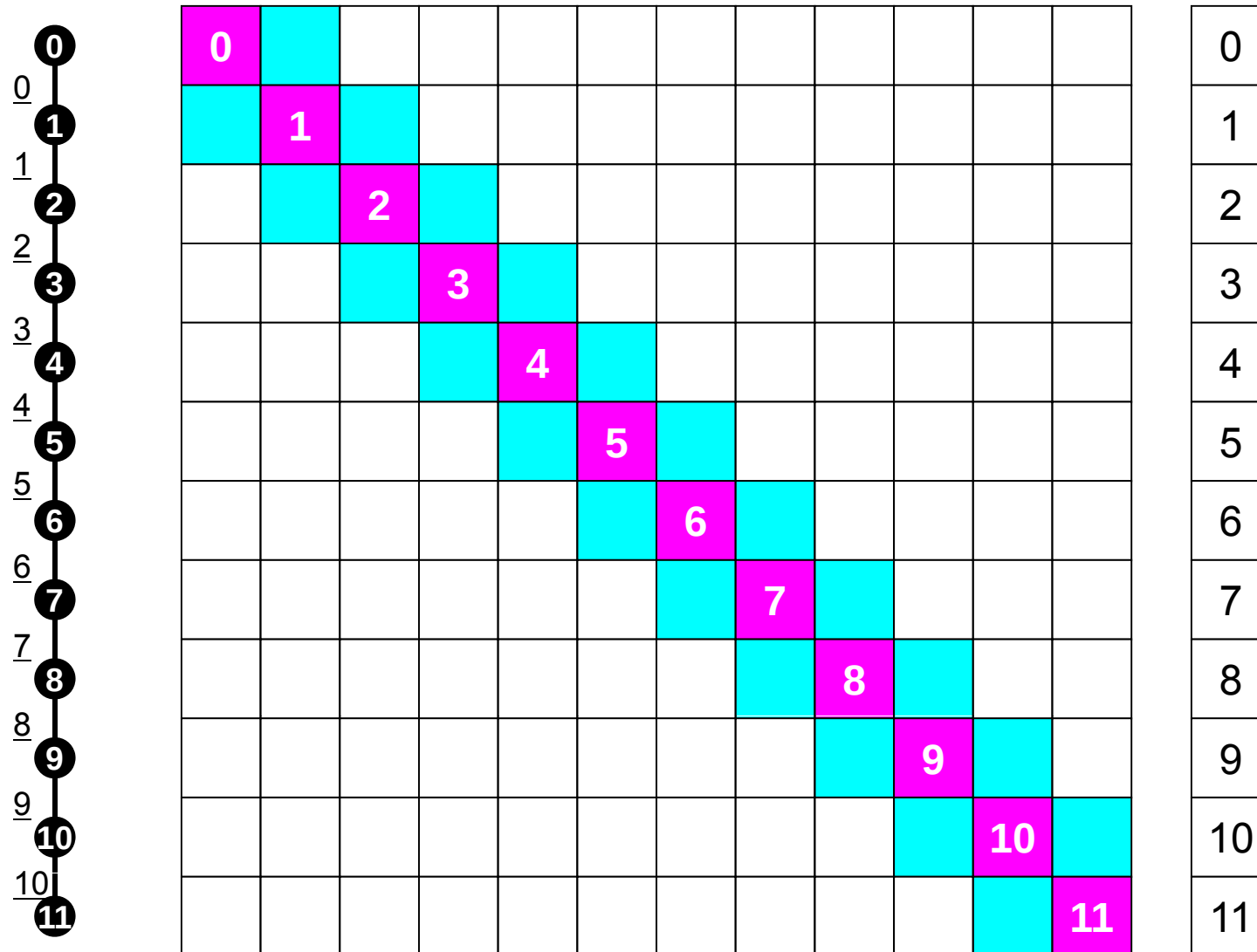
- 大規模データ → 領域分割, 局所データ並列処理
- 全体系計算 → 領域間の通信が必要



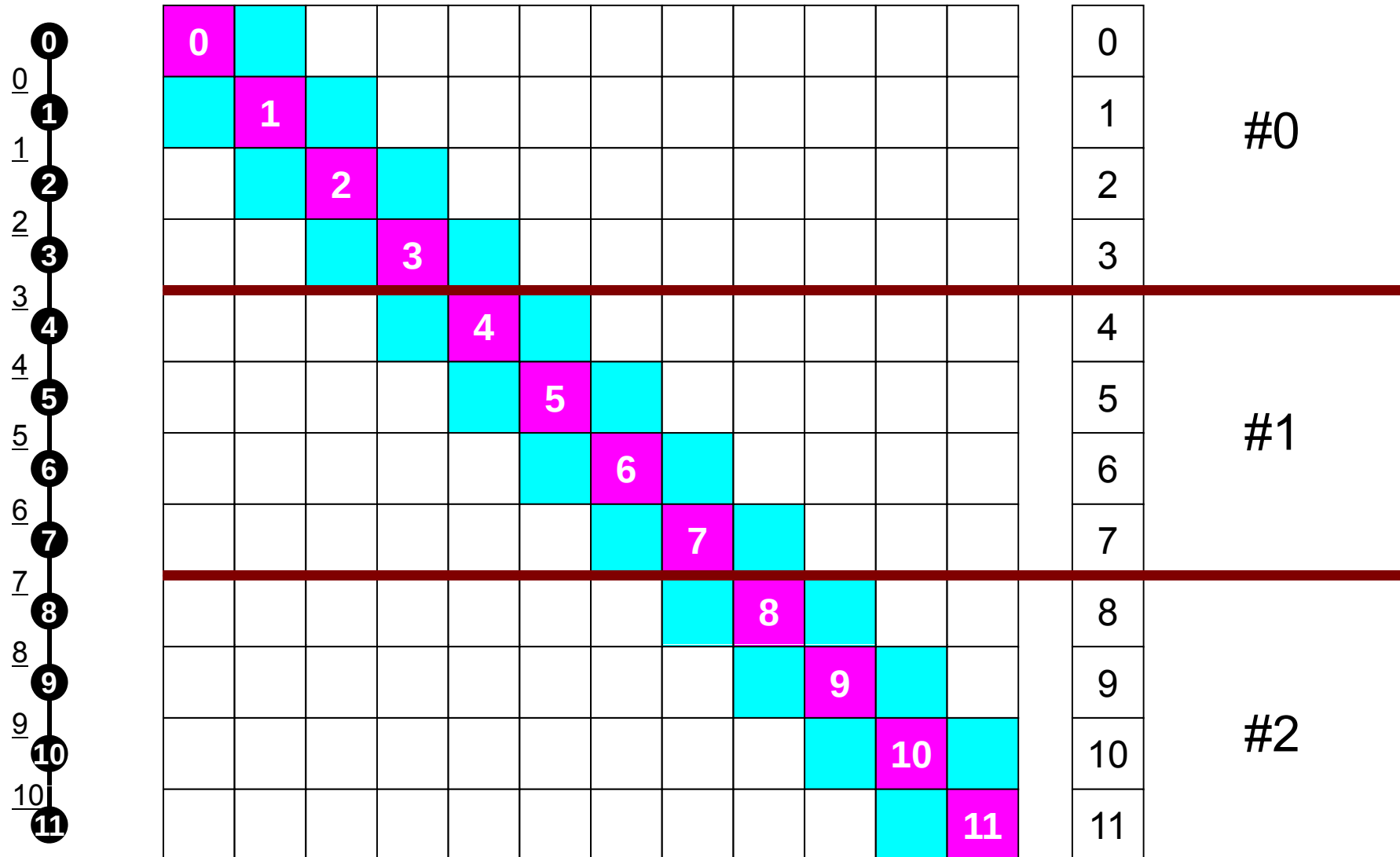
一次元問題: 11要素, 12節点, 3領域



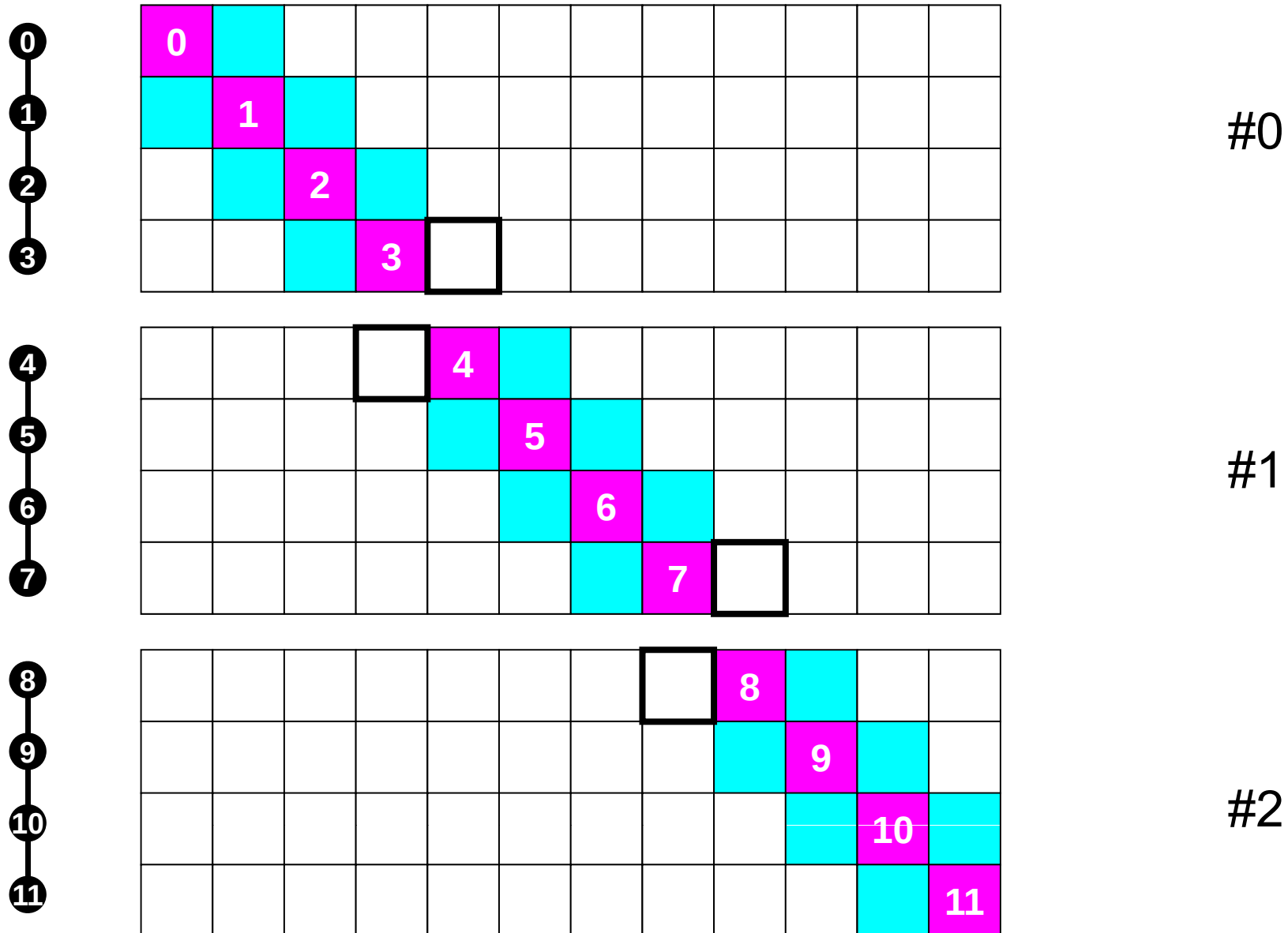
一次元問題: 11要素, 12節点, 3領域



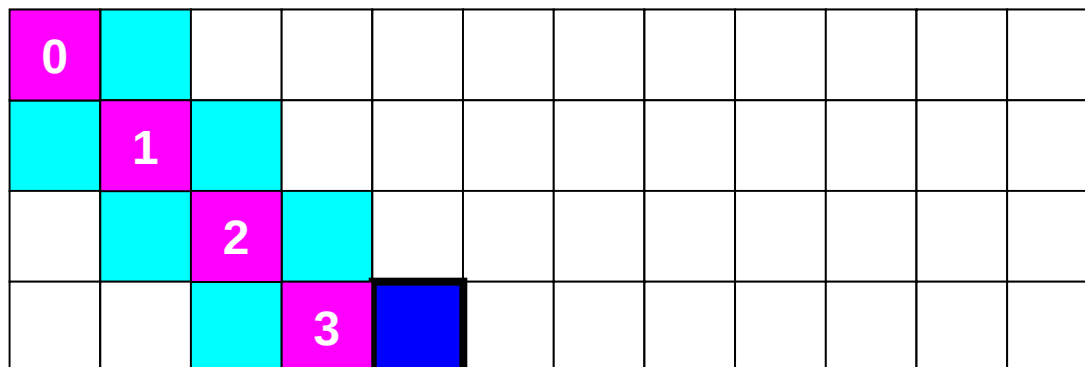
節点がバランスするように分割: 内点



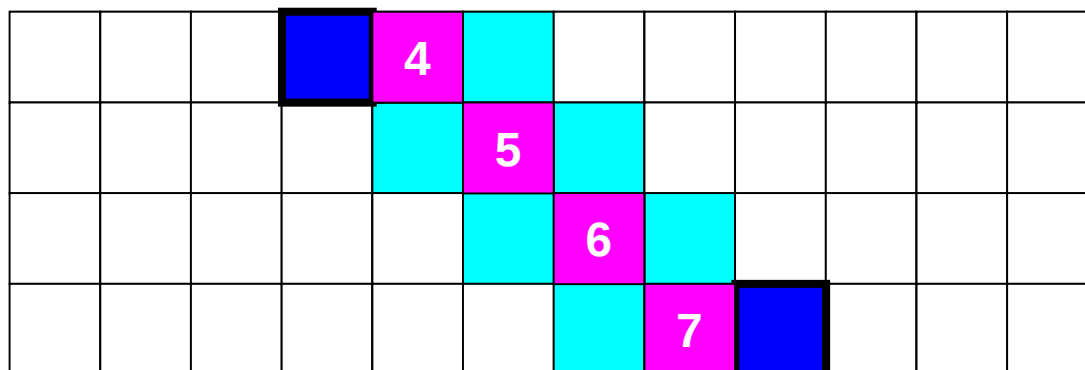
内点だけで分割するとマトリクス不完全



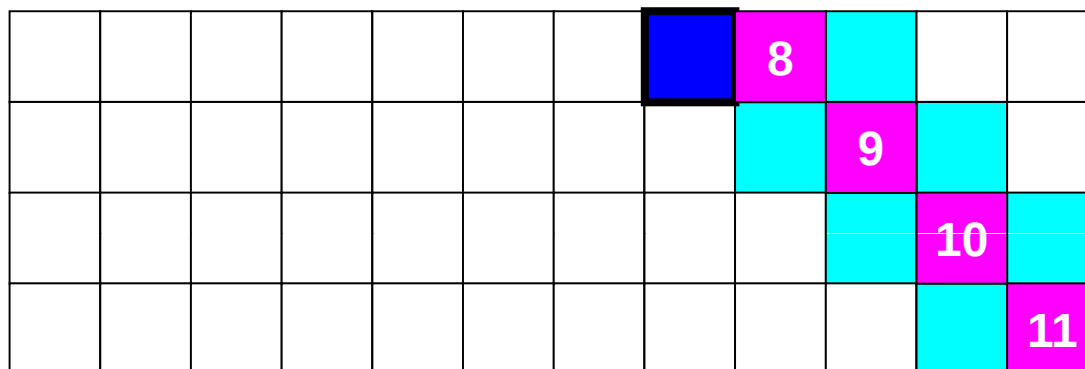
要素＋外点



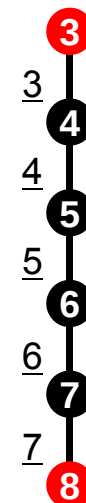
#0



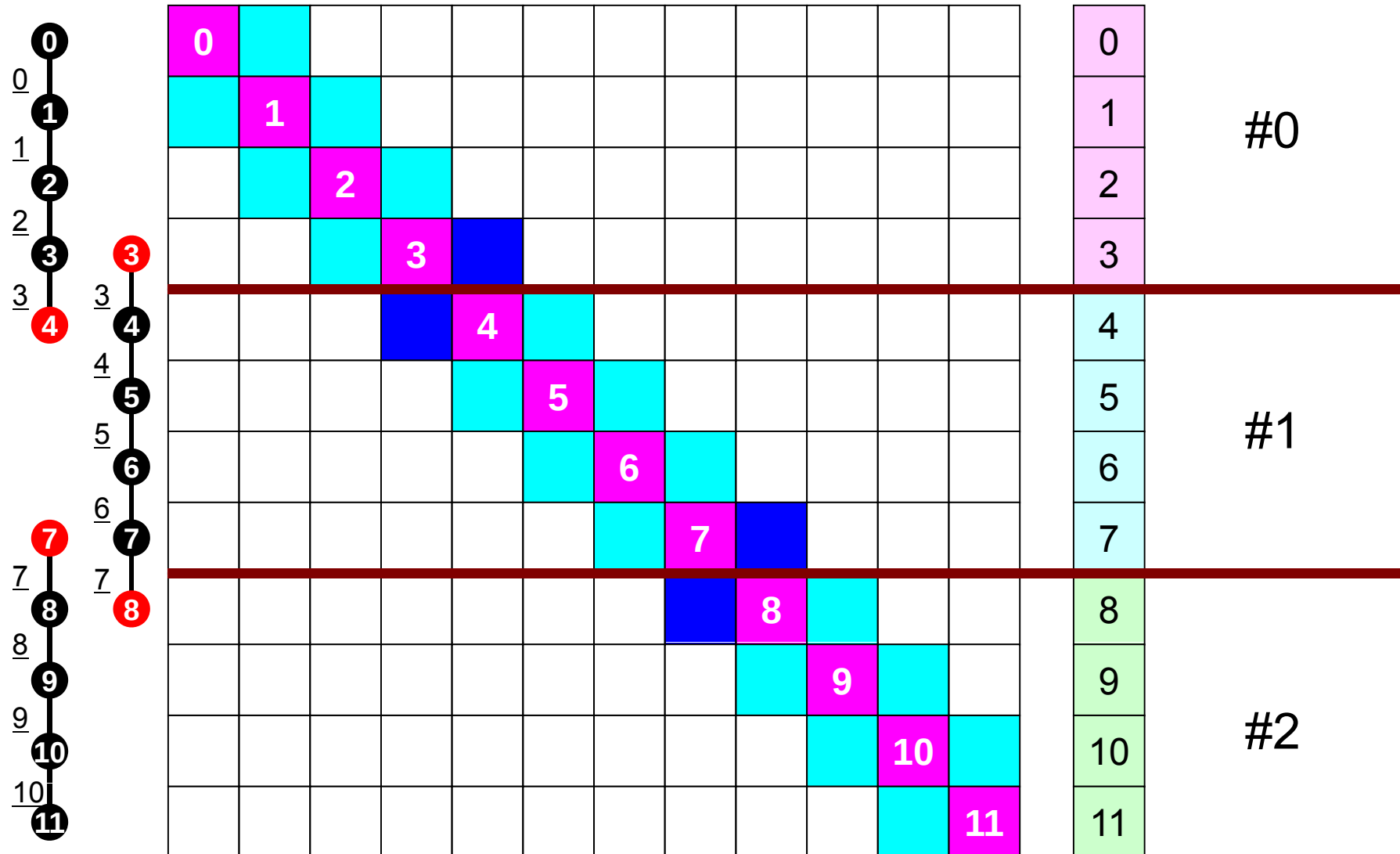
#1



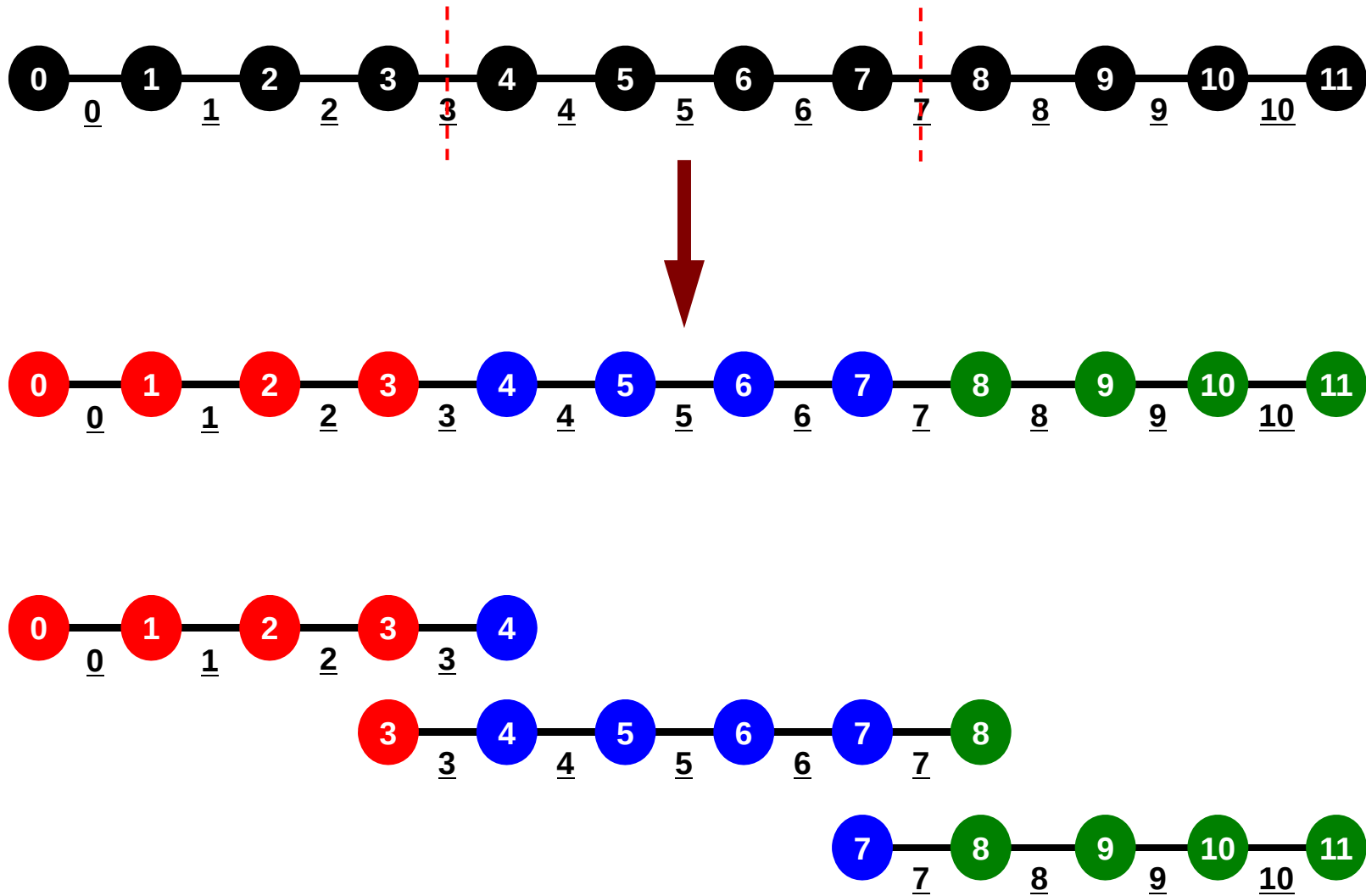
#2



一次元問題: 11要素, 12節点, 3領域

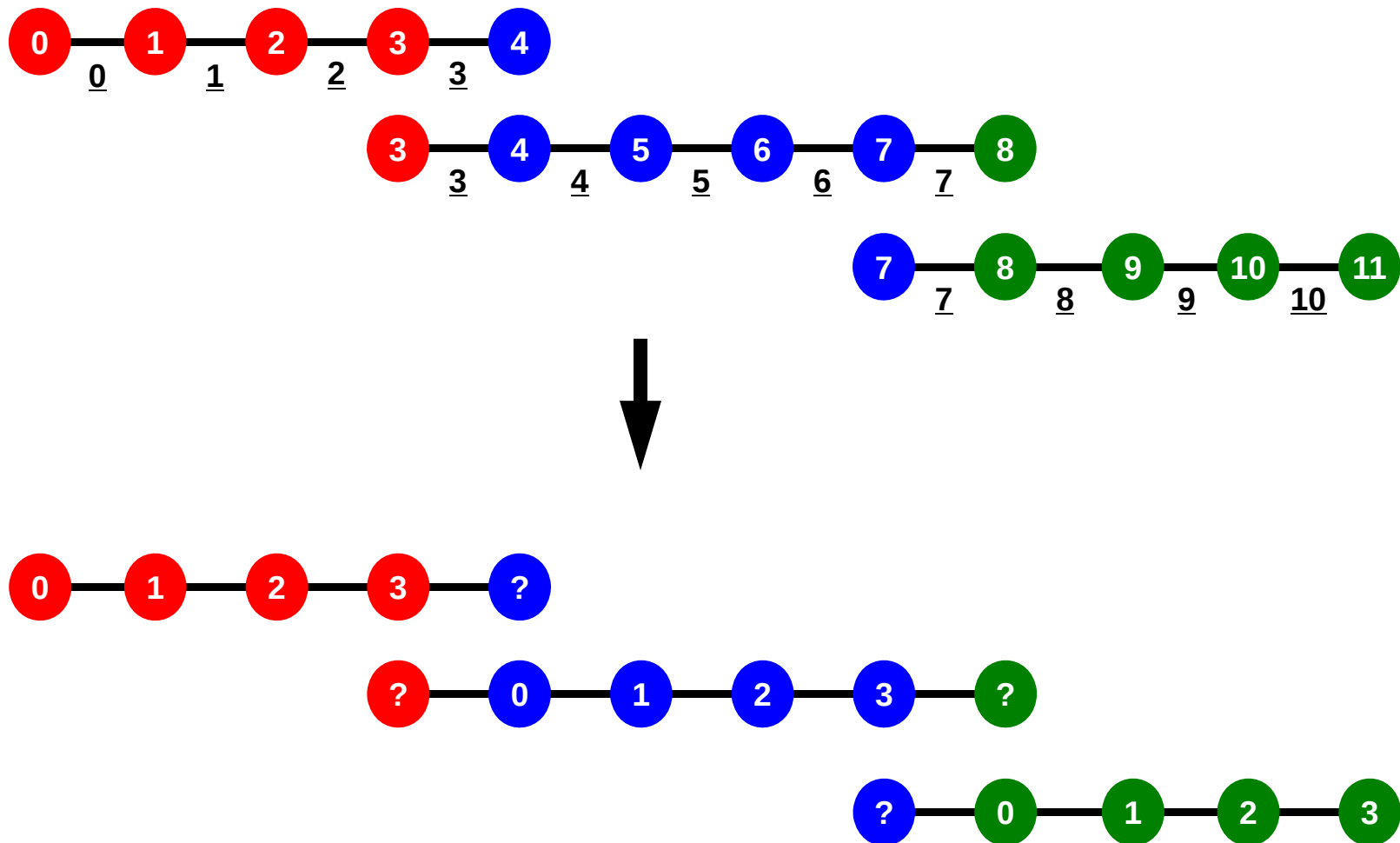


一次元問題: 11要素, 12節点, 3領域



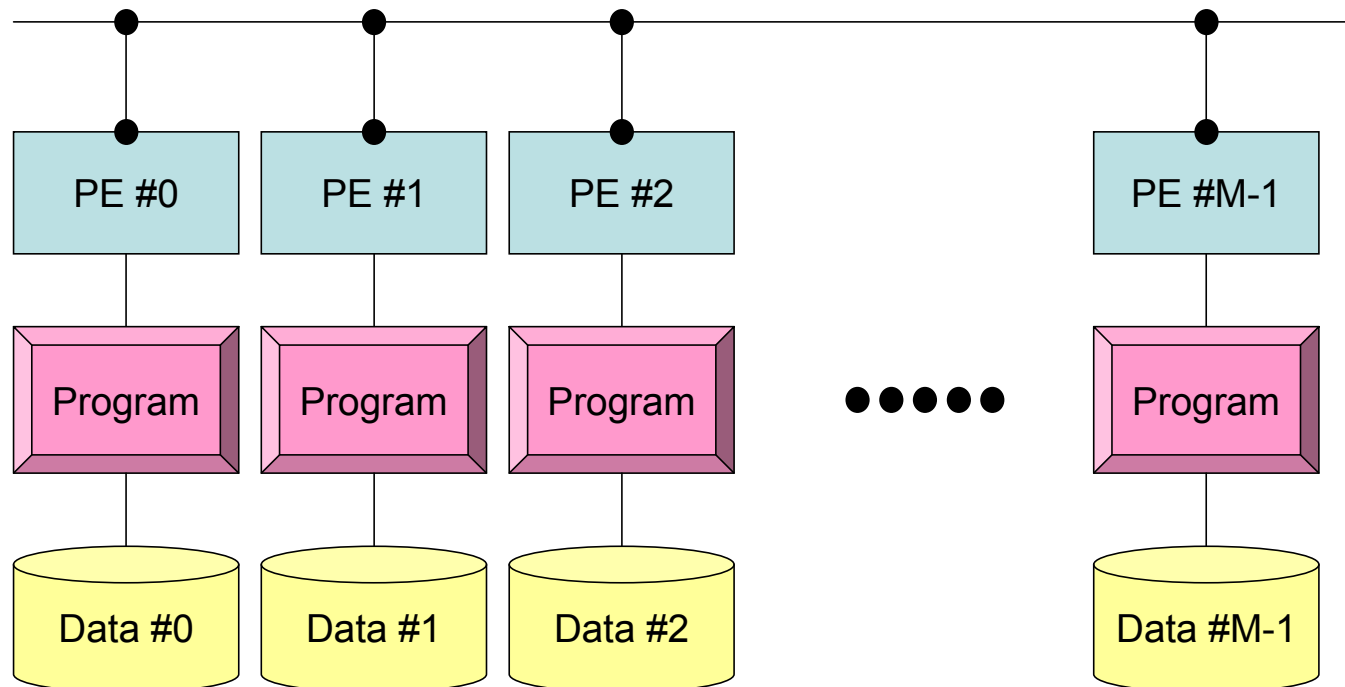
SPMD向け局所番号付け

内点が1~N(0~N)となっていれば, もとのプログラムと同じ
外点の番号は?



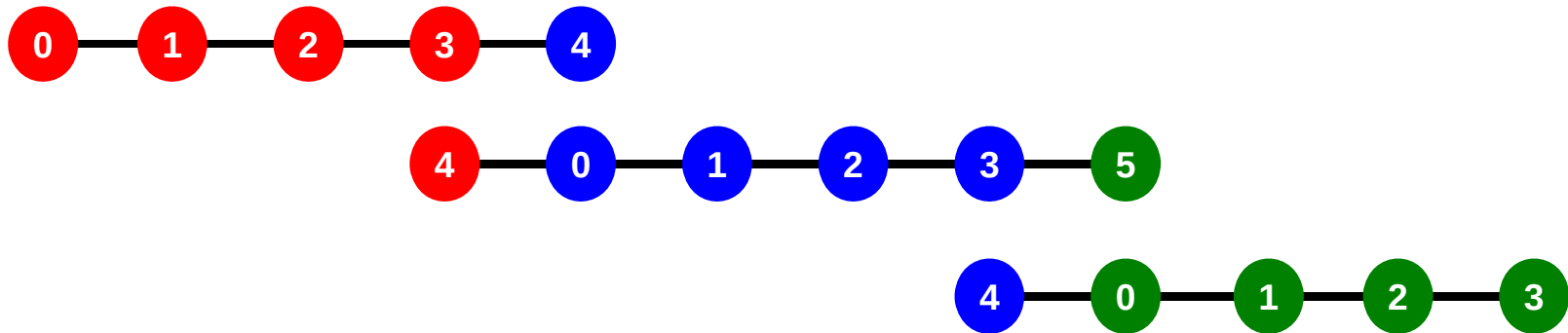
MPIによる並列化: SPMD

- Single Program/Instruction Multiple Data
- 基本的に各プロセスは「同じことをやる」が「データが違う」
 - 大規模なデータを分割し、各部分について各プロセス(プロセッサ)が計算する
- 全体データと局所データ, 全体番号と局所番号
- 通信以外は単体CPUと同じ, というのが理想



SPMD向け局所番号付け

内点が1~N(0~N)となっていれば, もとのプログラムと同じ
外点の番号は?, $N+1, N+2$ ($N, N+1$)



有限要素法の処理: プログラム

- 初期化: 並列計算可
 - 制御変数読み込み
 - 座標読み込み⇒要素生成(N:節点数, NE:要素数)
 - 配列初期化(全体マトリクス, 要素マトリクス)
 - 要素⇒全体マトリクスマッピング(Index, Item)
- マトリクス生成: 並列計算可
 - 要素単位の処理(do icel= 1, NE)
 - 要素マトリクス計算
 - 全体マトリクスへの重ね合わせ
 - 境界条件の処理
- 連立一次方程式: ?
 - 共役勾配法(CG)
- 応力計算

前処理付き共役勾配法

Preconditioned Conjugate Gradient Method (CG)

```

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

```

前処理: 対角スケーリング

前処理, ベクトル定数倍の加減

局所的な計算(内点のみ)が可能⇒並列処理

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

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

0
1
2
3
4
5
6
7
8
9
10
11

内積

全体で和をとる必要がある⇒通信？

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

0
1
2
3
4
5
6
7
8
9
10
11

行列ベクトル積

外点の値が必要⇒通信？

```
/*  
/-- {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]];  
    }  
}
```



行列ベクトル積：ローカルに計算実施可能

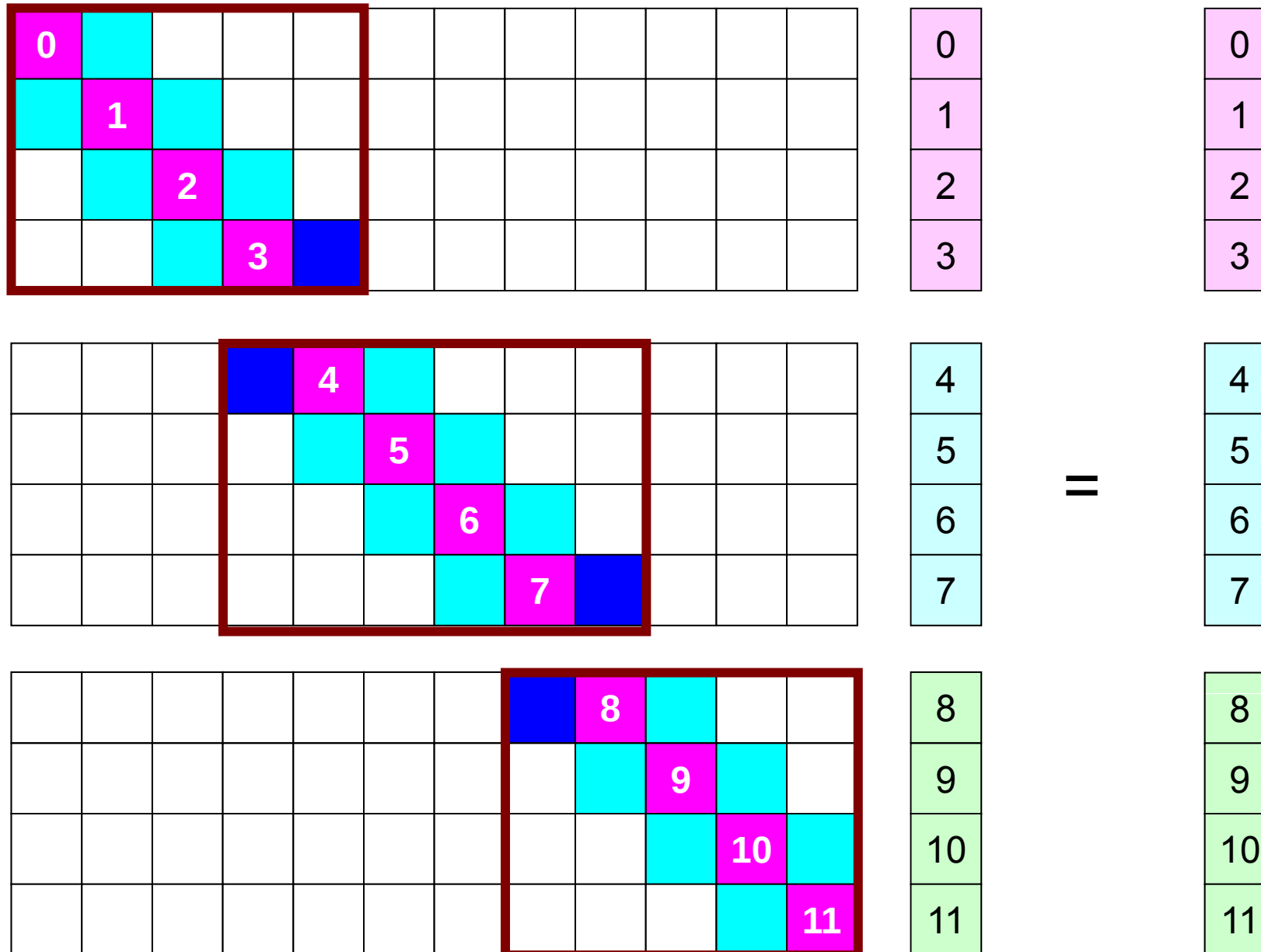
0											
	1										
		2									
			3								
				4							
					5						
						6					
							7				
								8			
									9		
										10	
											11

0
1
2
3
4
5
6
7
8
9
10
11

=

0
1
2
3
4
5
6
7
8
9
10
11

行列ベクトル積: ローカルに計算実施可能



行列ベクトル積：ローカルに計算実施可能

0				
	1			
		2		
			3	

0
1
2
3

0
1
2
3

	0				
		1			
			2		
				3	

0
1
2
3

=

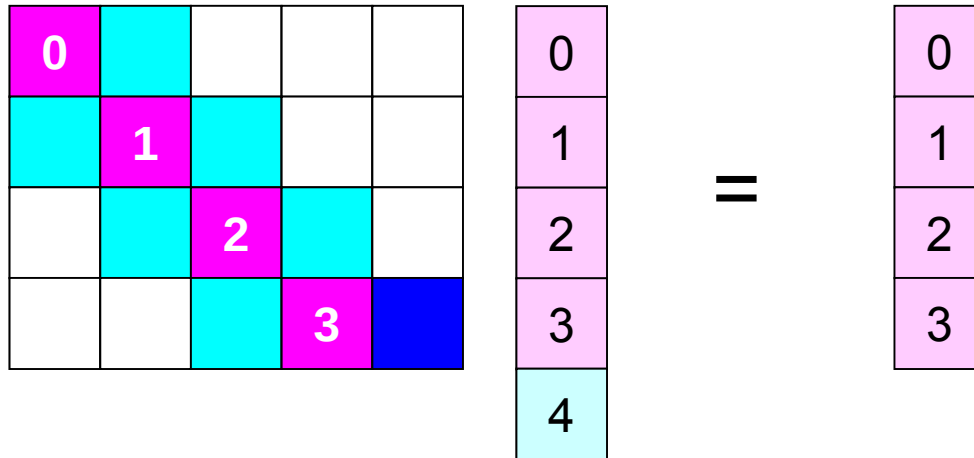
0
1
2
3

	0			
		1		
			2	
				3

0
1
2
3

0
1
2
3

行列ベクトル積: ローカル計算 #0



行列ベクトル積: ローカル計算 #1

	0					0
		1				1
			2			2
				3		3

=

0
1
2
3



0						0
	1					1
		2				2
			3			3
						4
						5

=

0
1
2
3



行列ベクトル積:ローカル計算 #2

