並列有限要素法への道
―並列データ構造―
C言語編

2012年夏季集中講義

中島研吾

並列計算プログラミング（616-2057）・先端計算機演習I（616-4009）

並列計算の目的

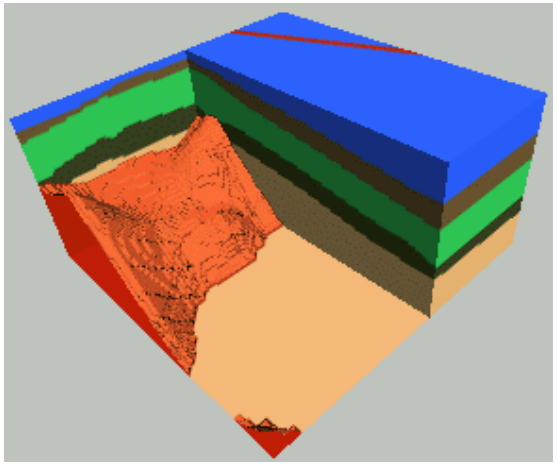
- 高速，大規模
  - 「大規模」の方が「新しい科学」という観点からのウェイトとしては高い。しかし，「高速」ももちろん重要である。
    - 細かいメッシュ
- +複雑
- 理想：Scalable
  - N倍の規模の計算をN倍のCPUを使って，「同じ時間で」解く（大規模性の追求：Weak Scaling）
    - 実際はそうは行かない
      - 例：共役勾配法⇒問題規模が大きくなると反復回数が増加
  - 同じ問題をN倍のCPUを使って「1/Nの時間で」解く・・・という場合もある（高速性の追求：Strong Scaling）
    - これも余り簡単な話では無い

並列計算とは？(1/2)

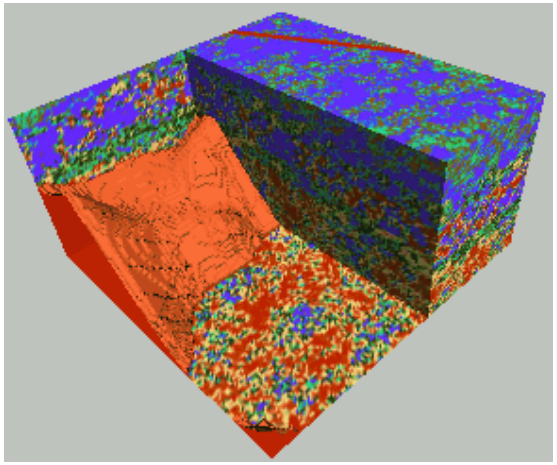
- より大規模で複雑な問題を高速に解きたい

Homogeneous/Heterogeneous Porous Media

Lawrence Livermore National Laboratory



Homogeneous

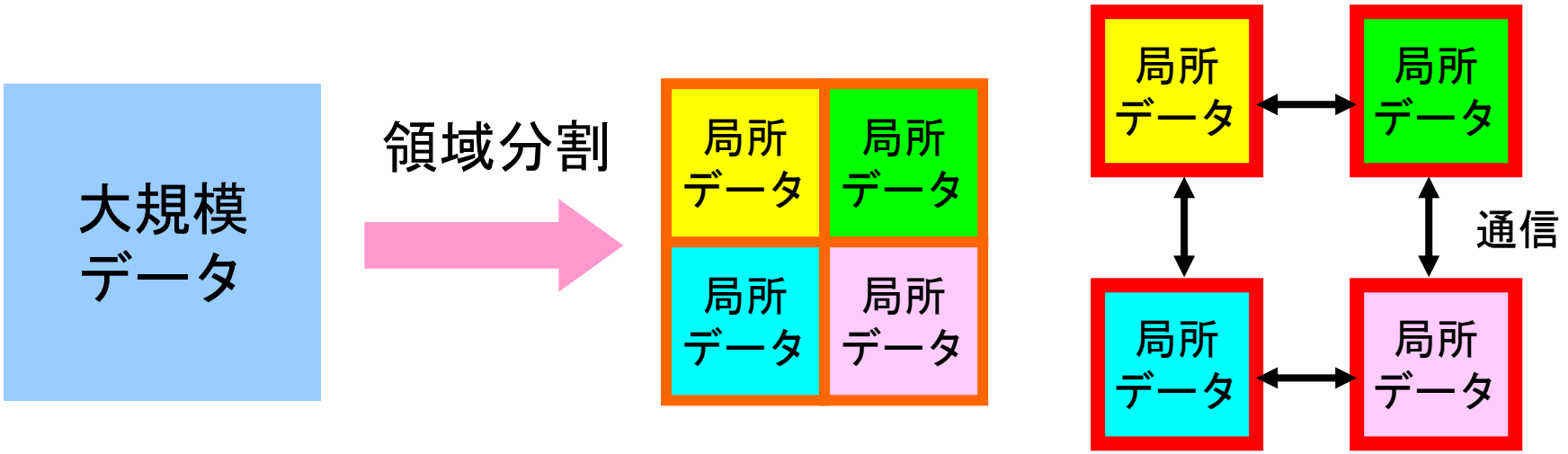


Heterogeneous

このように不均質な場を模擬するには非常に細かいメッシュが必要となる

並列計算とは？(2/2)

- 1GB程度のPC → <10^6メッシュが限界：FEM
  - 1000km×1000km×100kmの領域（西南日本）を1kmメッシュで切ると10^8メッシュになる
- 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



通信とは？

- 並列計算とはデータと処理をできるだけ「局所的（local）」に実施すること。
  - 要素マトリクスの計算
  - 有限要素法の計算は本来並列計算向けである

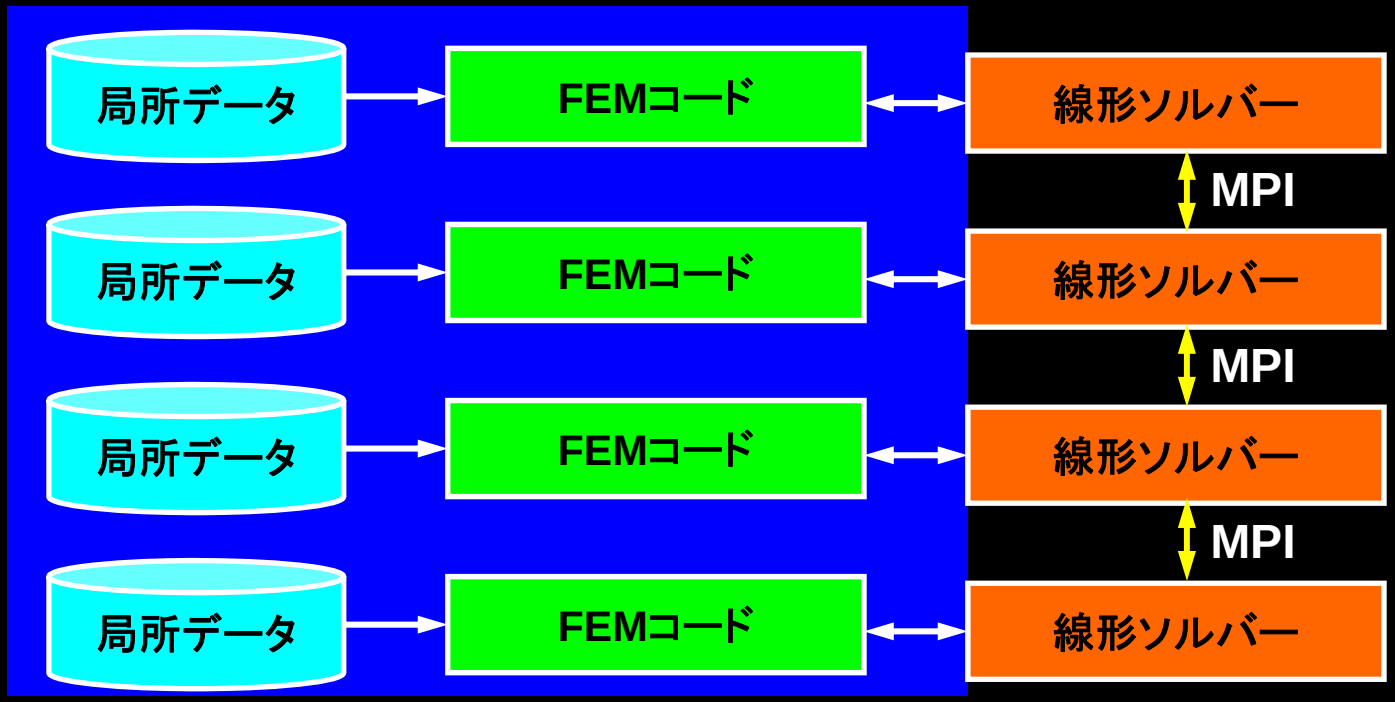
- 「大域的（global）」な情報を必要とする場合に通信が生じる（必要となる）。
  - 全体マトリクスを線形ソルバーで解く

並列有限要素法の処理：SPMD

巨大な解析対象 → 局所分散データ，領域分割（Partitioning）

有限要素コードは領域ごとに係数マトリクスを生成：要素単位の処理によって可能：シリアルコードと変わらない処理

グローバル処理，通信は線形ソルバーのみで生じる
　内積，行列ベクトル積，前処理



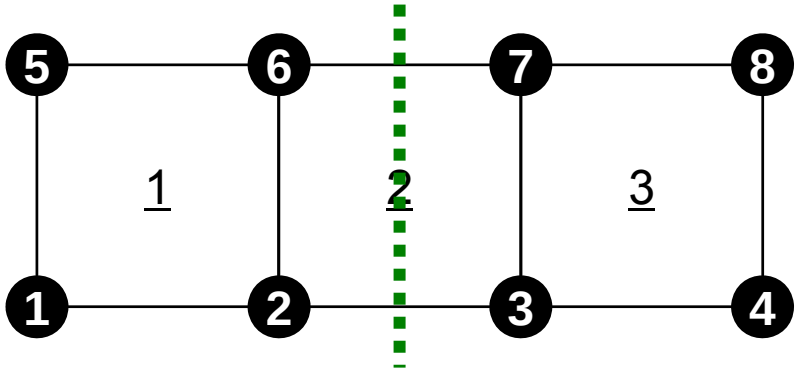
並列有限要素法プログラムの開発

- 前頁のようなオペレーションを実現するためのデータ構造が重要
  - アプリケーションの「並列化」にあたって重要なのは，適切な局所分散データ構造の設計である。

- 前処理付反復法
- マトリクス生成：ローカルに処理

- はじめに
- MPIとは
- 並列有限要素法とは？
  - 基本的な考え方
  - 局所データ構造

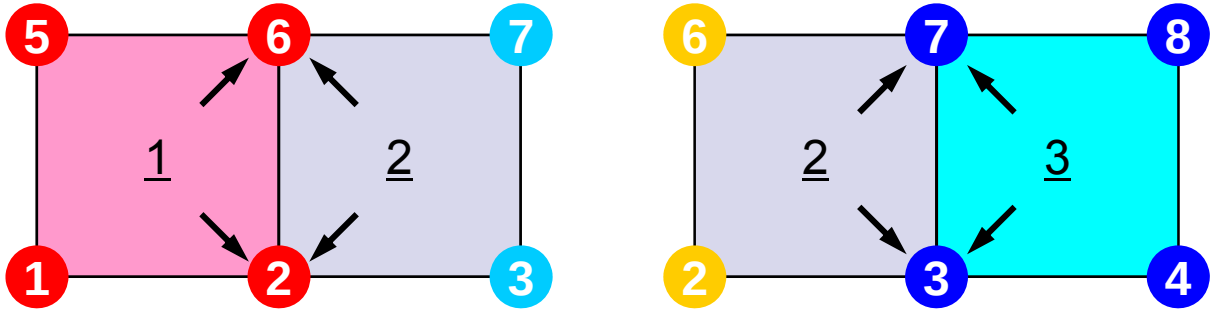
四角形要素



「節点ベース（領域ごとの節点数がバランスする）」の分割

自由度：節点上で定義

これではマトリクス生成に必要な情報は不十分



マトリクス生成のためには，オーバーラップ部分の要素と節点の情報が必要

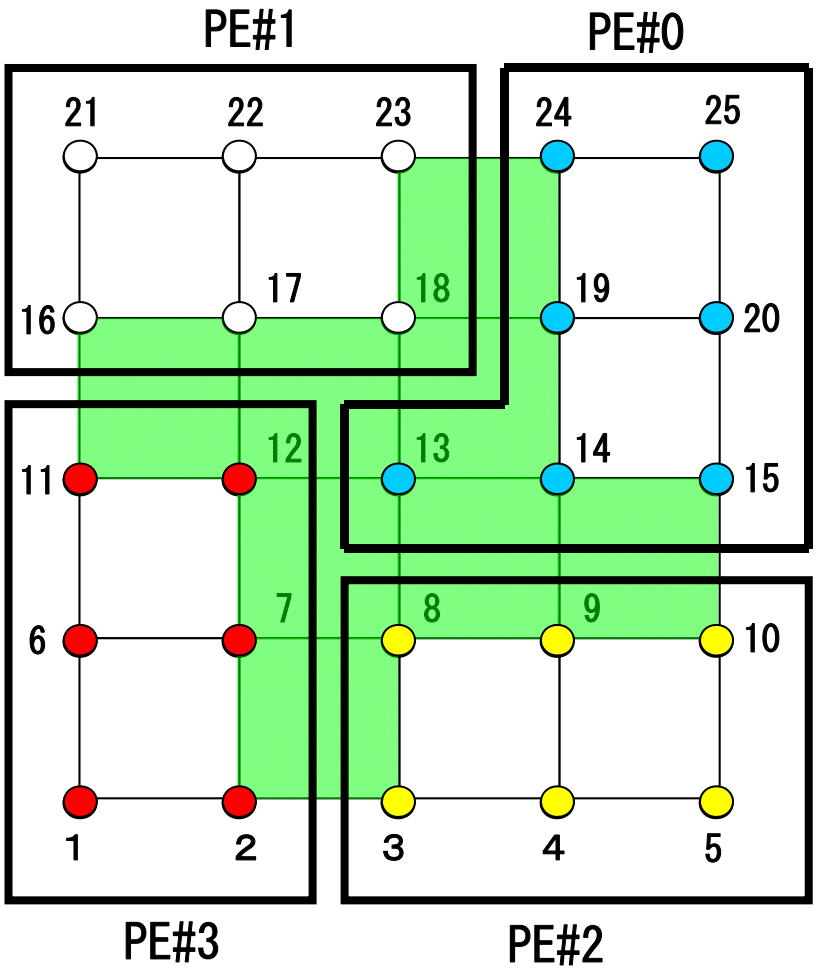
並列有限要素法の局所データ構造

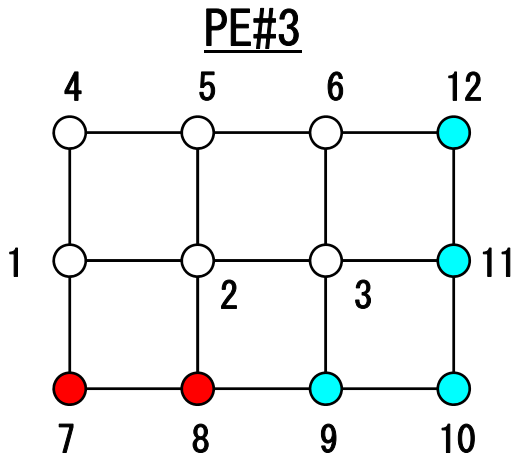
- 節点ベース：Node-based partitioning
- 局所データに含まれるもの：
  - その領域に本来含まれる節点
  - それらの節点を含む要素
  - 本来領域外であるが，それらの要素に含まれる節点
- 節点は以下の3種類に分類
  - 内点：Internal nodes　　その領域に本来含まれる節点
  - 外点：External nodes　　本来領域外であるがマトリクス生成に必要な節点
  - 境界点：Boundary nodes　　他の領域の「外点」となっている節点
- 領域間の通信テーブル

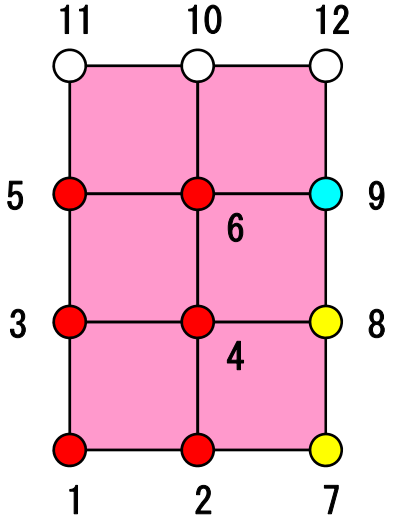
- 領域間の接続をのぞくと，大域的な情報は不要
  - 有限要素法の特性：要素で閉じた計算

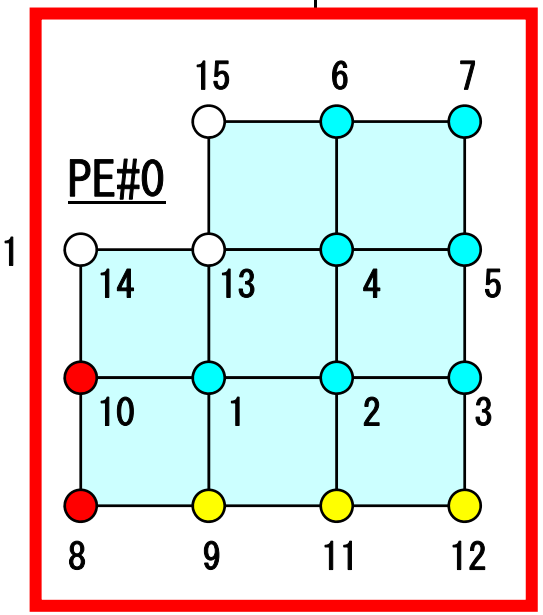
Node-based Partitioning

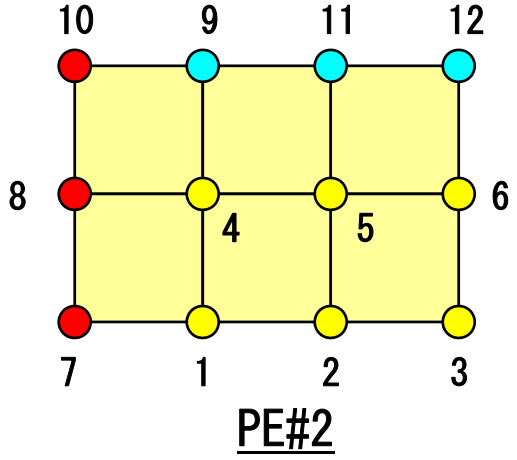
internal nodes - elements - external nodes





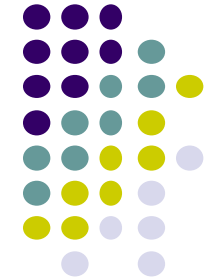




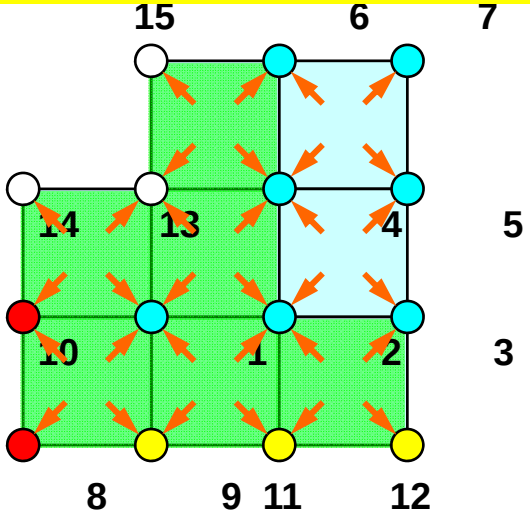


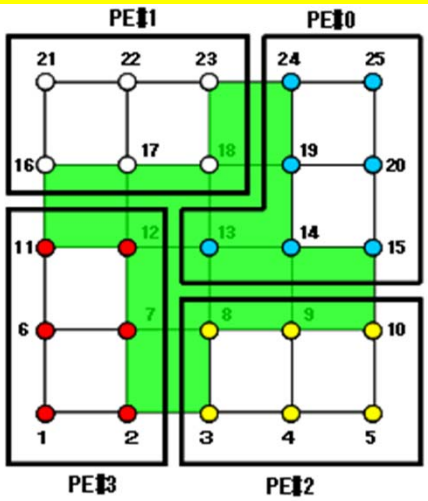
Node-based Partitioning

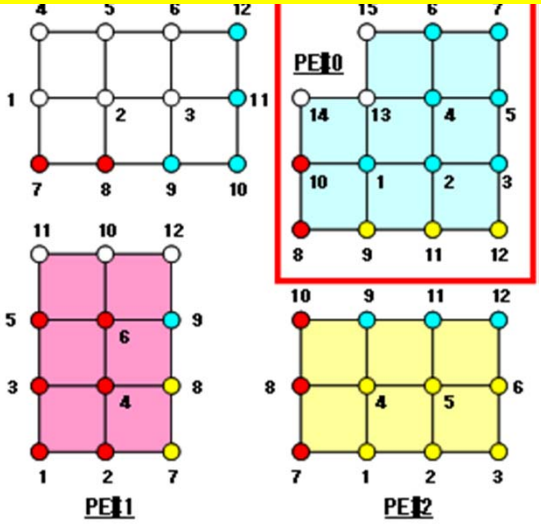
internal nodes - elements - external nodes



- 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.

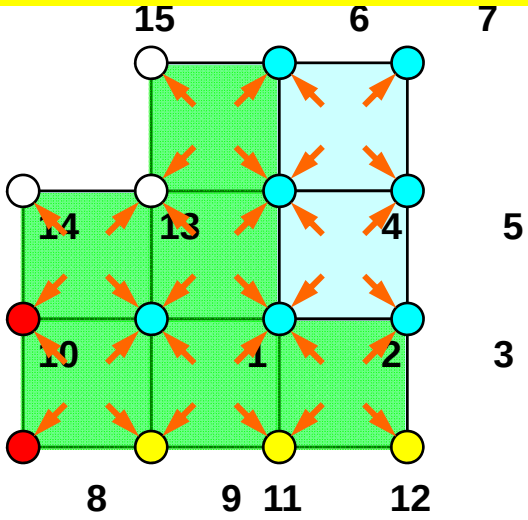


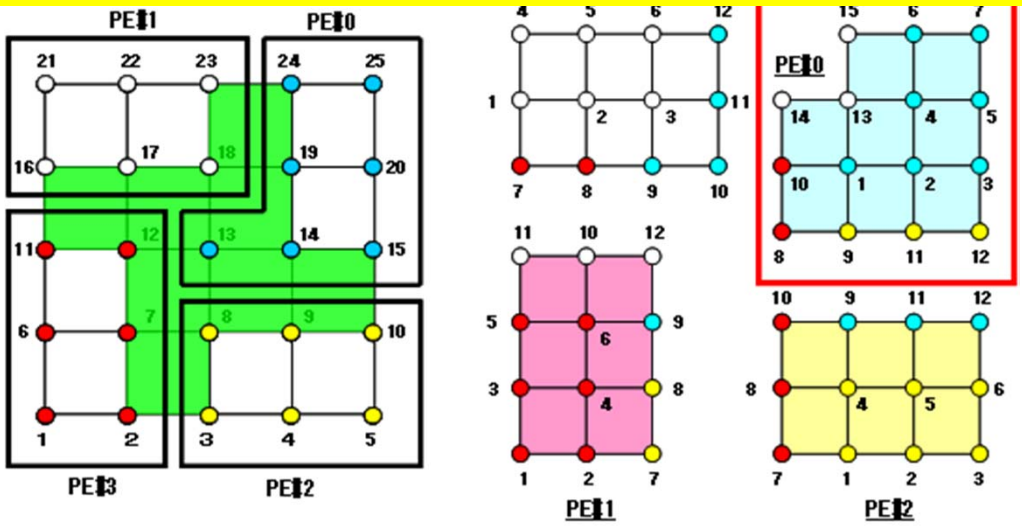




マトリクス生成時の通信は不要

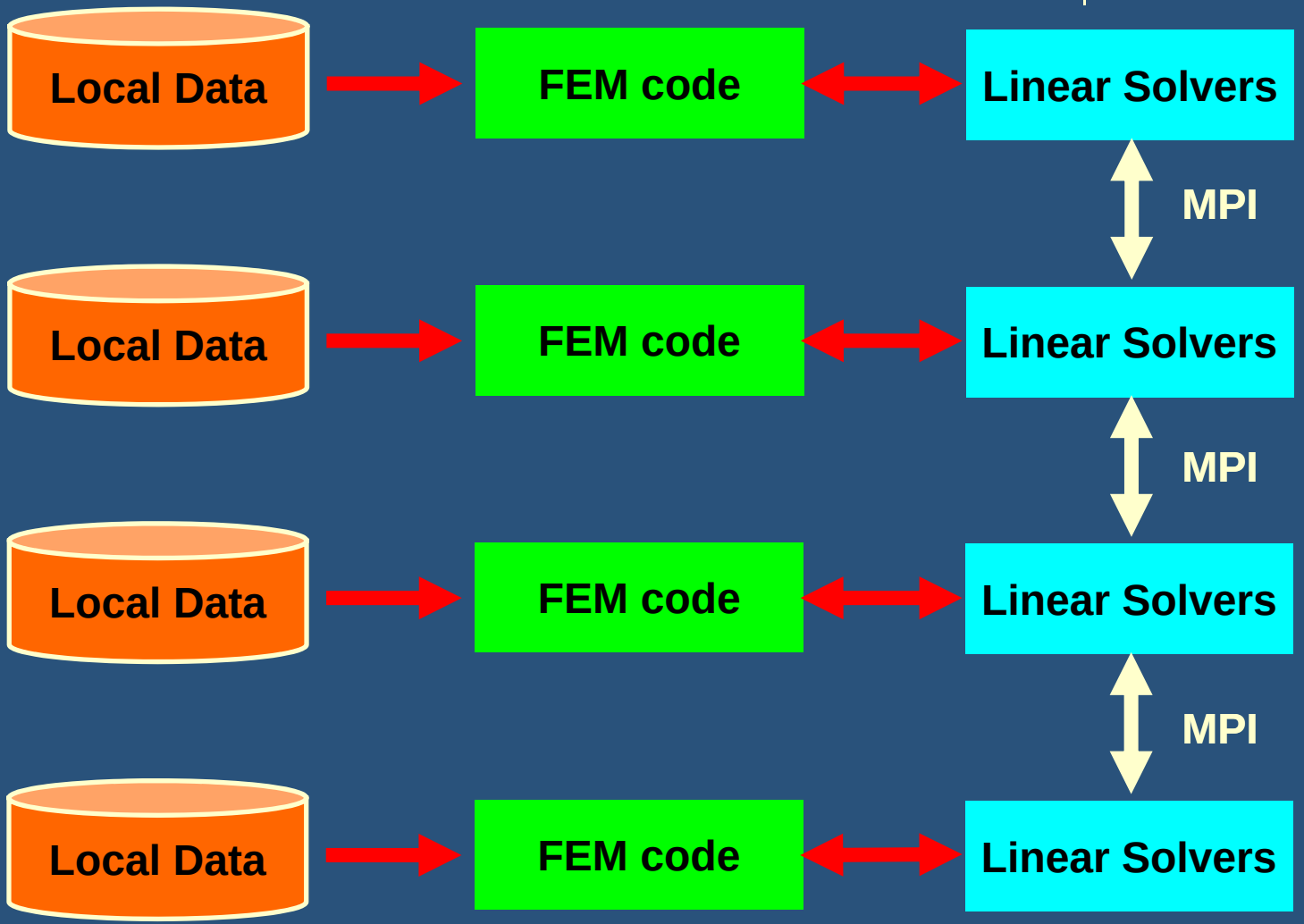
- Partitioned nodes themselves (Internal Nodes)　内点
- Elements which include Internal Nodes　内点を含む要素
- External Nodes included in the Elements　外点
  in overlapped region among partitions.
- Info of External Nodes are required for completely local element–based operations on each processor.





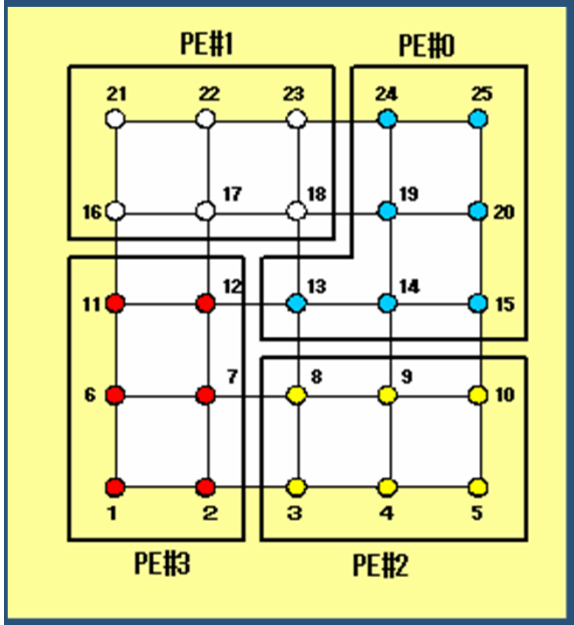
Parallel Computing in FEM

SPMD: Single-Program Multiple-Data



Parallel Computing in FEM

SPMD: Single-Program Multiple-Data



Local Data → FEM code ↔ Linear Solvers

MPI

Local Data → FEM code ↔ Linear Solvers

MPI

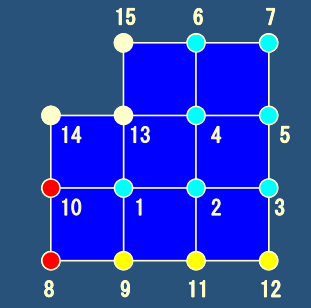
Local Data → FEM code ↔ Linear Solvers

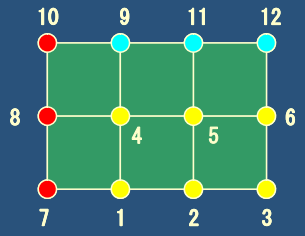
MPI

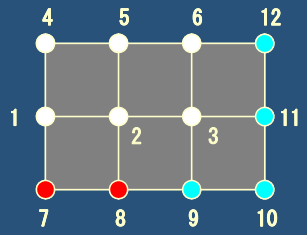
Local Data → FEM code ↔ Linear Solvers

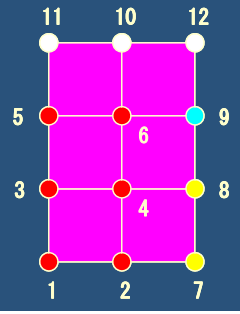
Parallel Computing in FEM

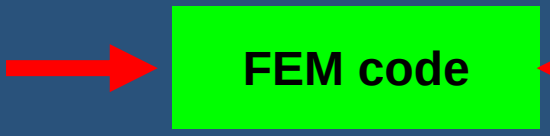
SPMD: Single-Program Multiple-Data

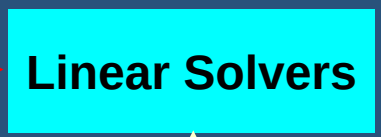


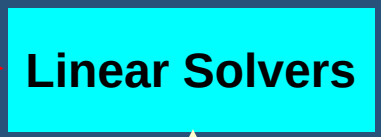


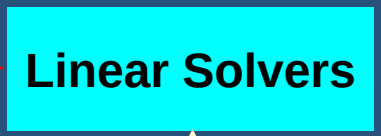


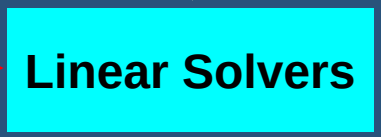






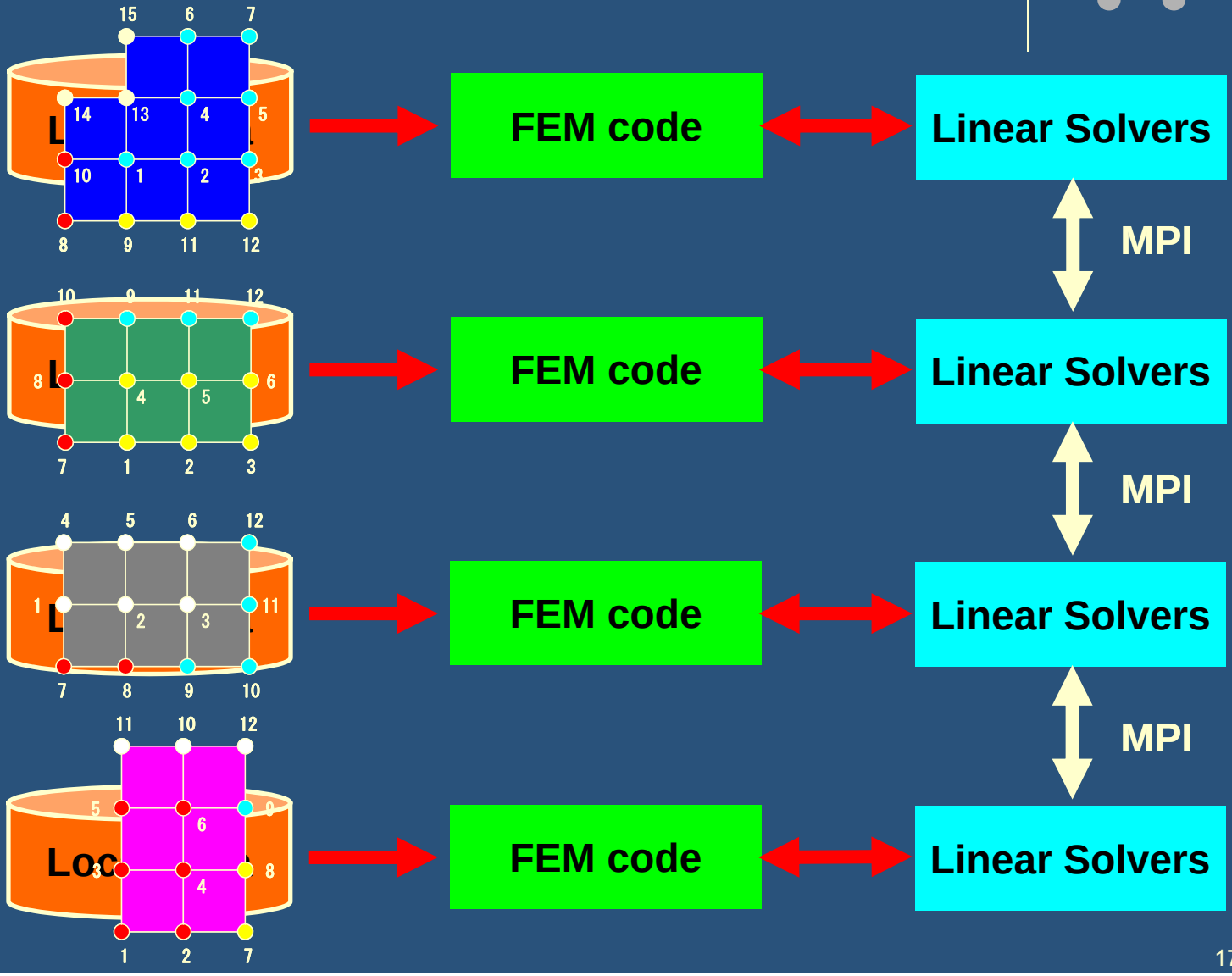






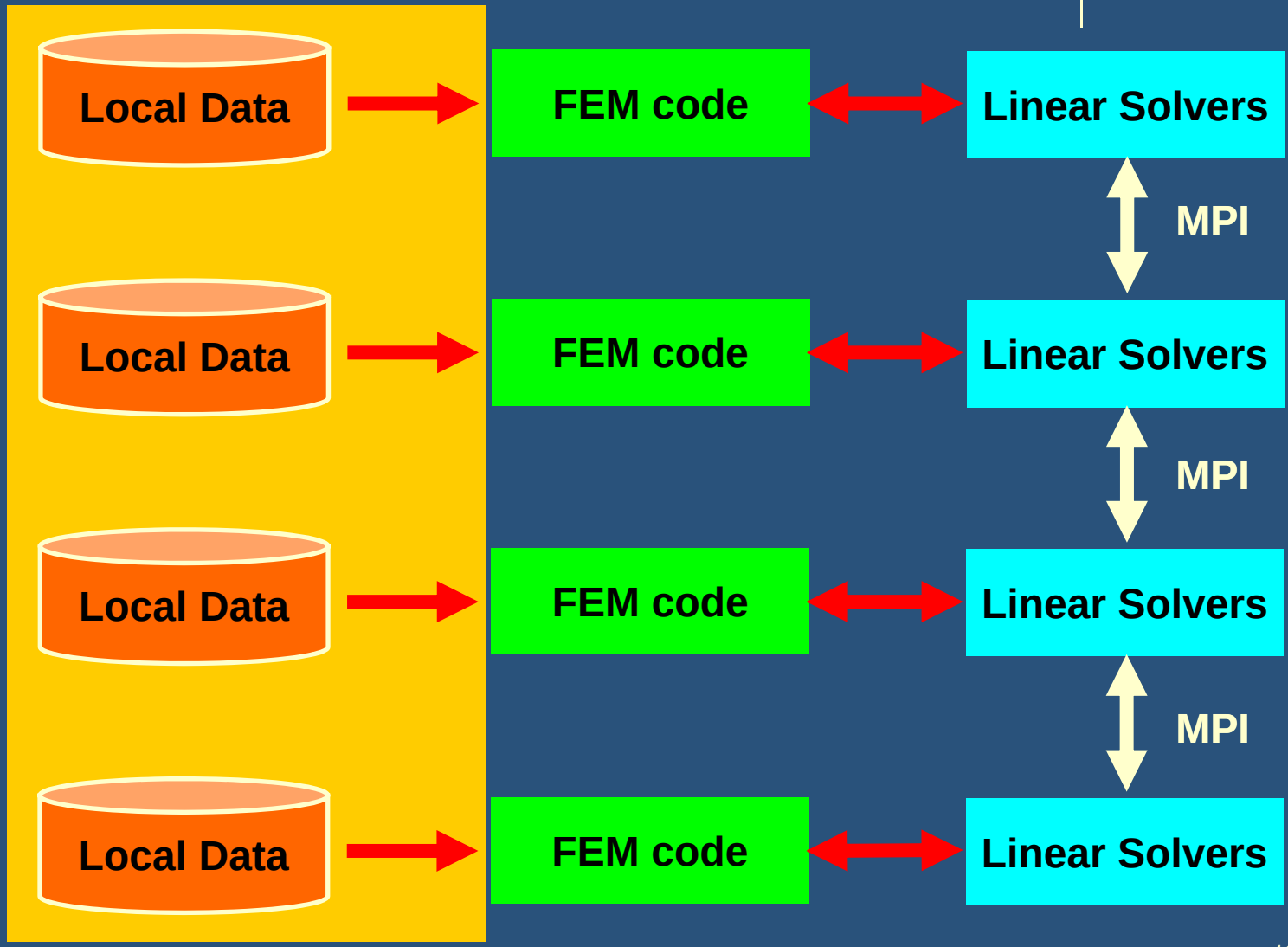
Parallel Computing in FEM

SPMD: Single-Program Multiple-Data

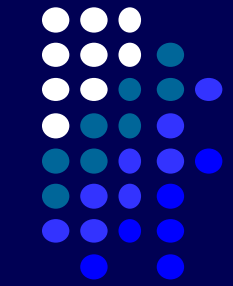


Parallel Computing in FEM

SPMD: Single-Program Multiple-Data

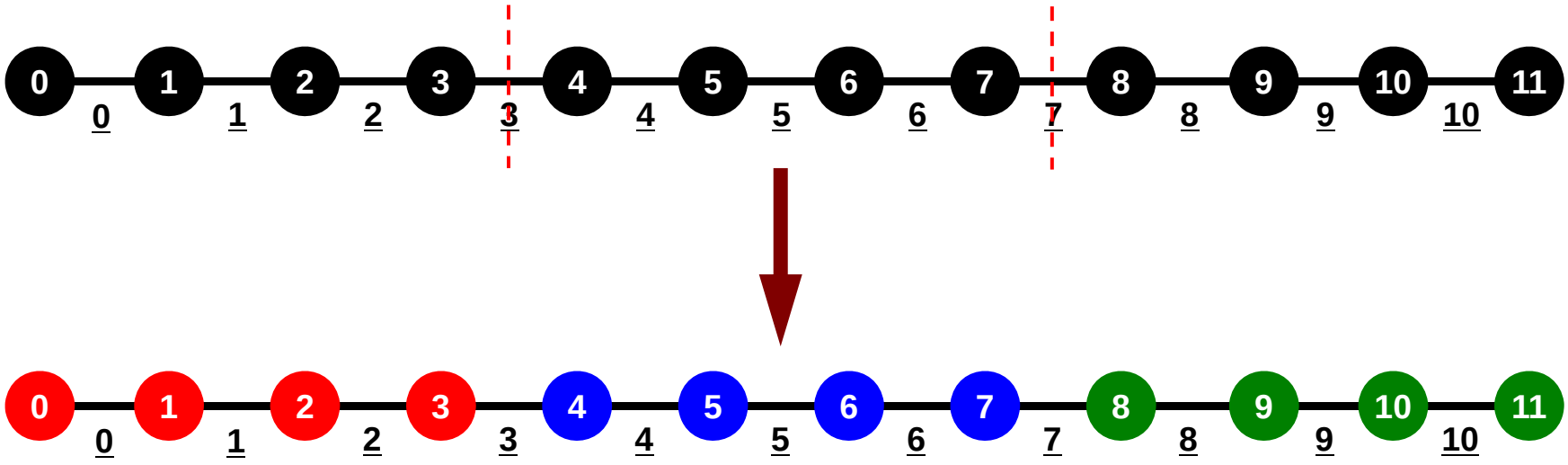


通信とは何か？

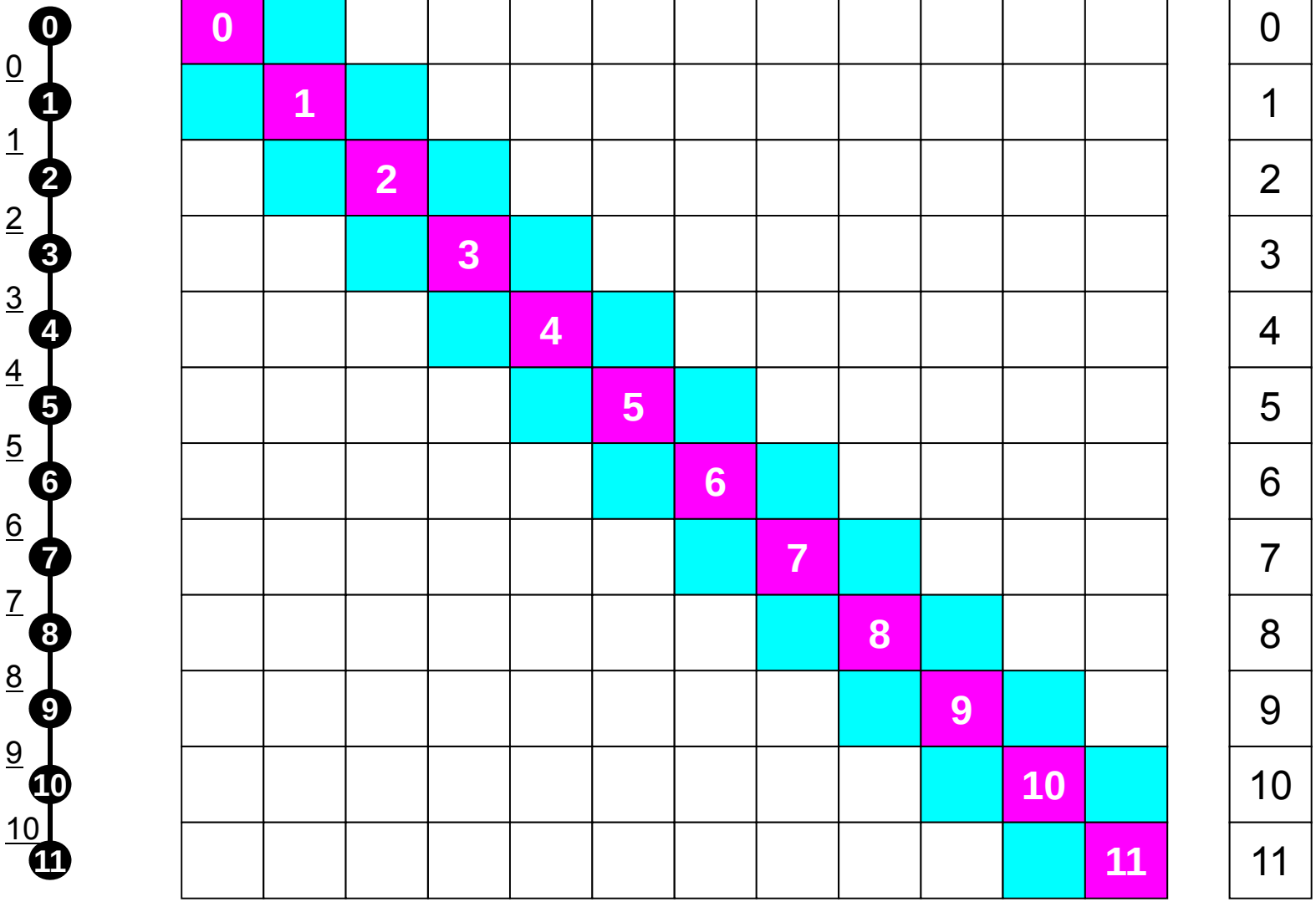


- 「外点」の情報を外部の領域からもらってくること
- 「通信テーブル」にその情報が含まれている

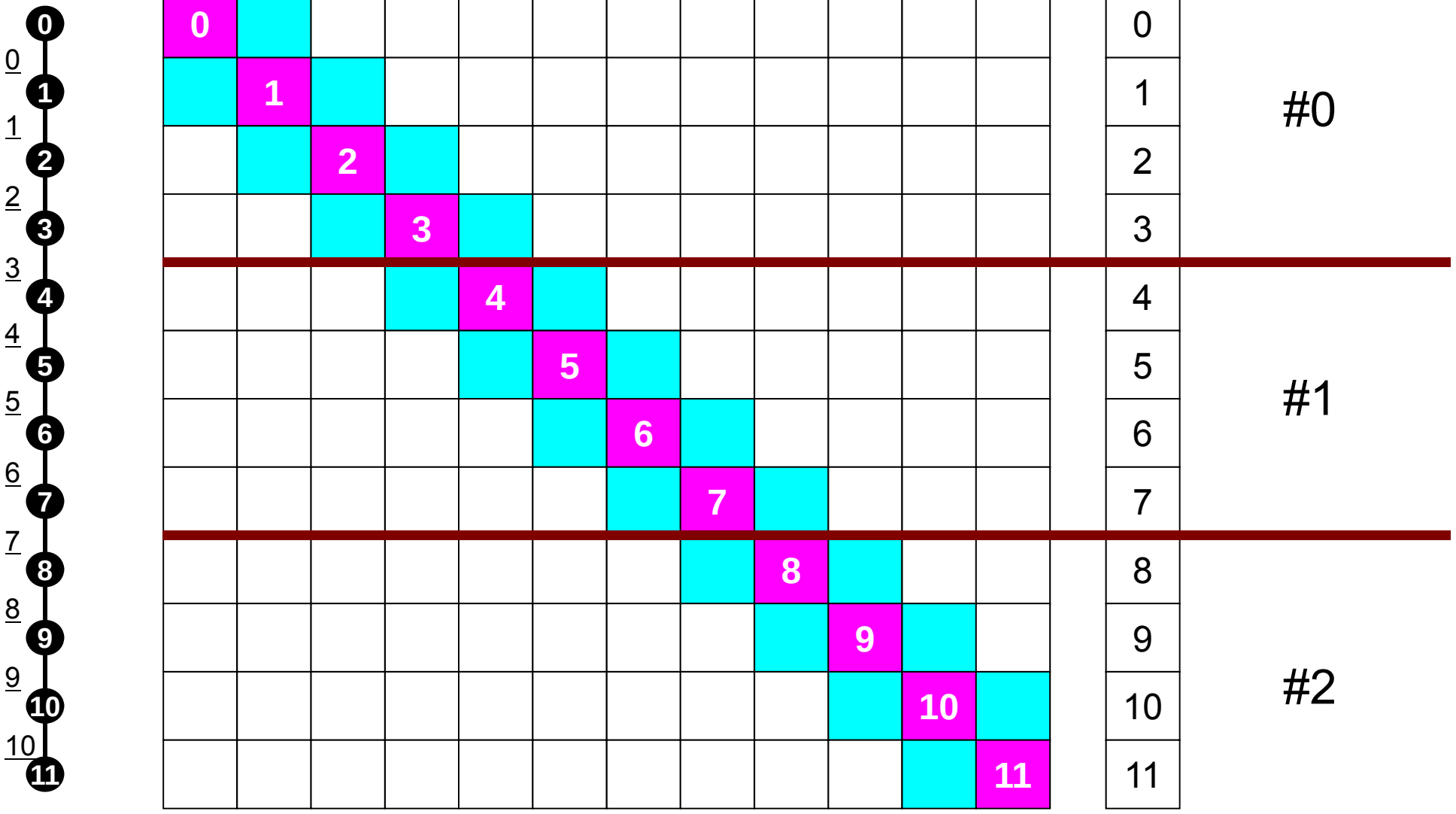
一次元問題：11要素，12節点，3領域



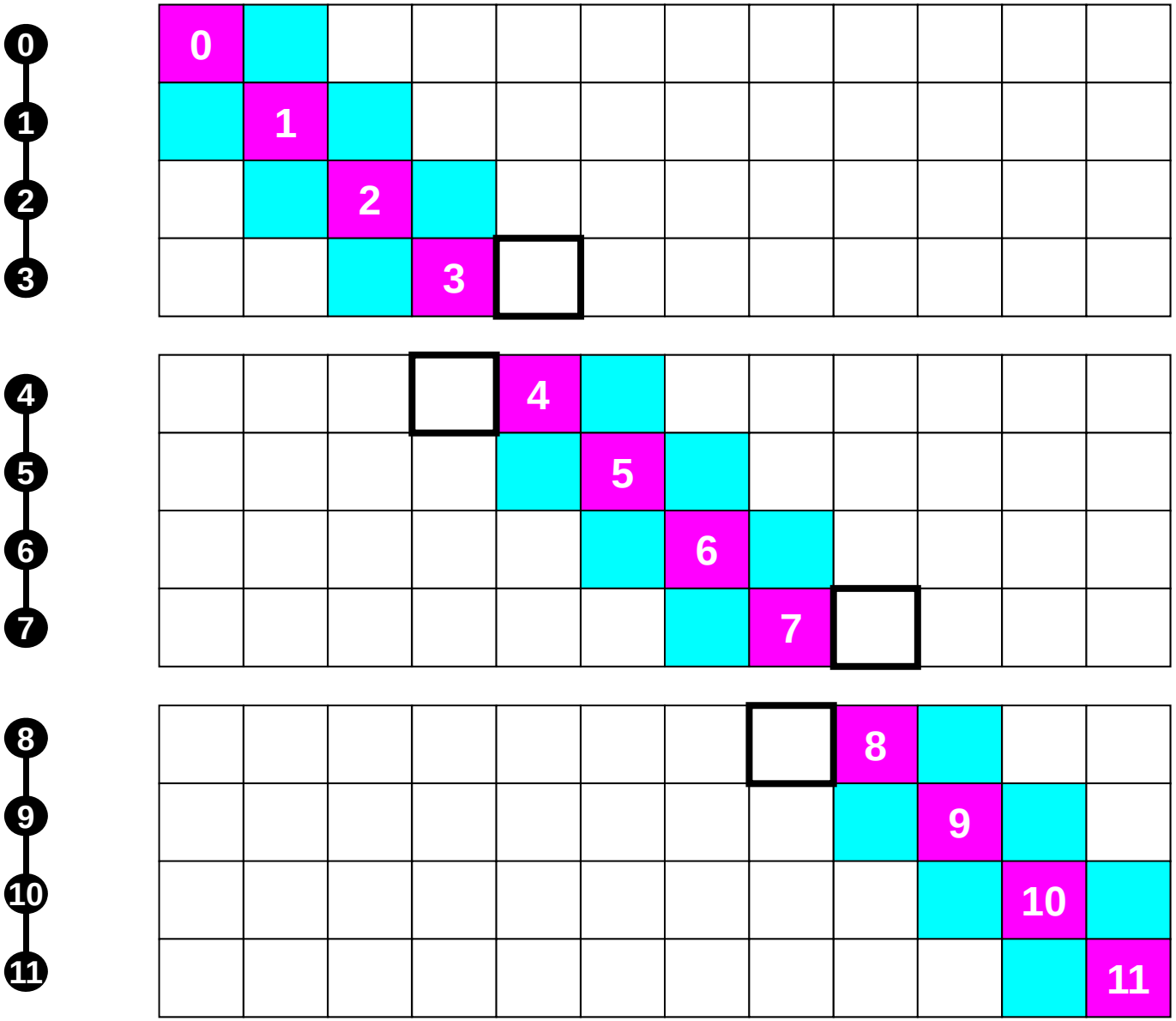
一次元問題：11要素，12節点，3領域



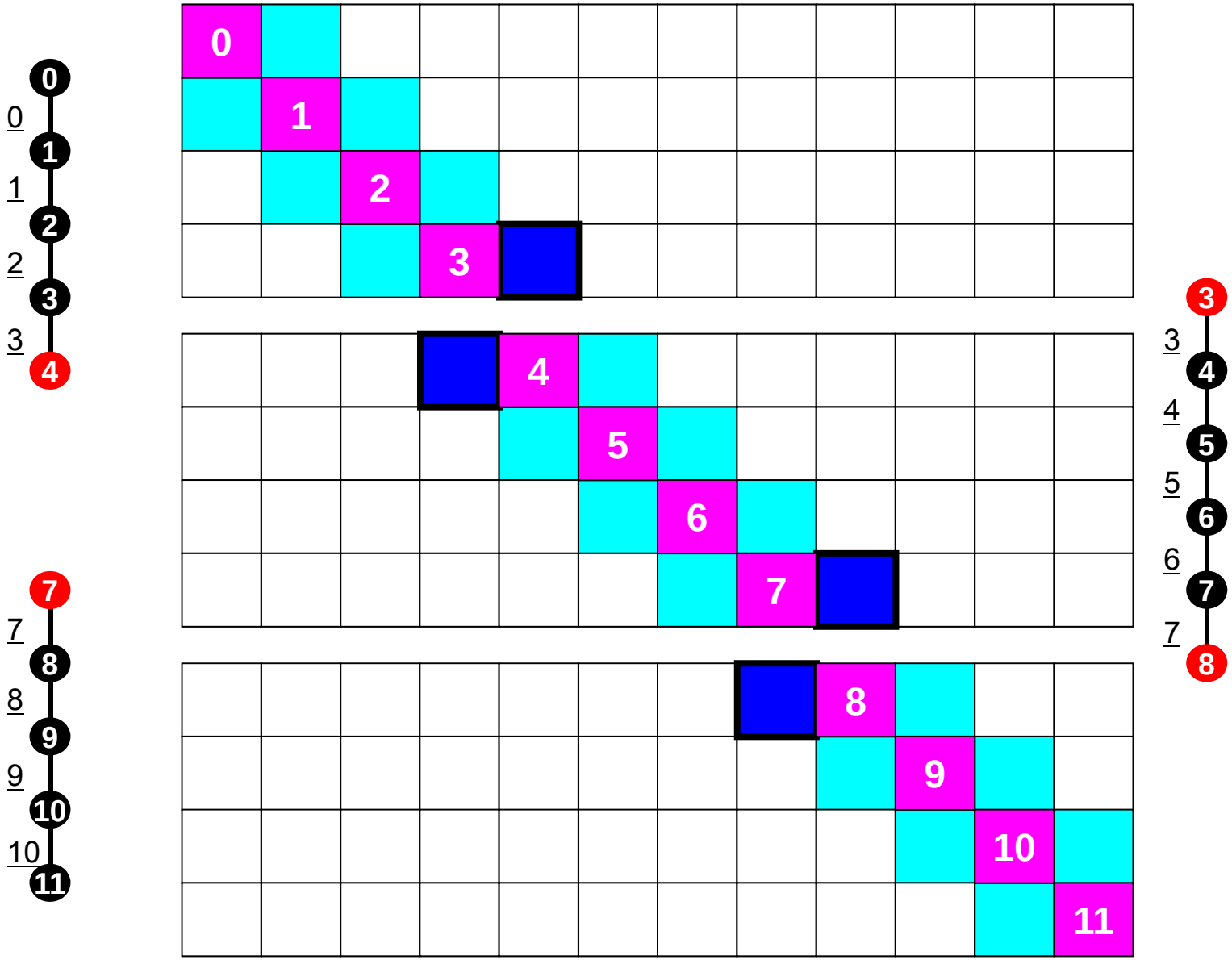
節点がバランスするよう分割：内点



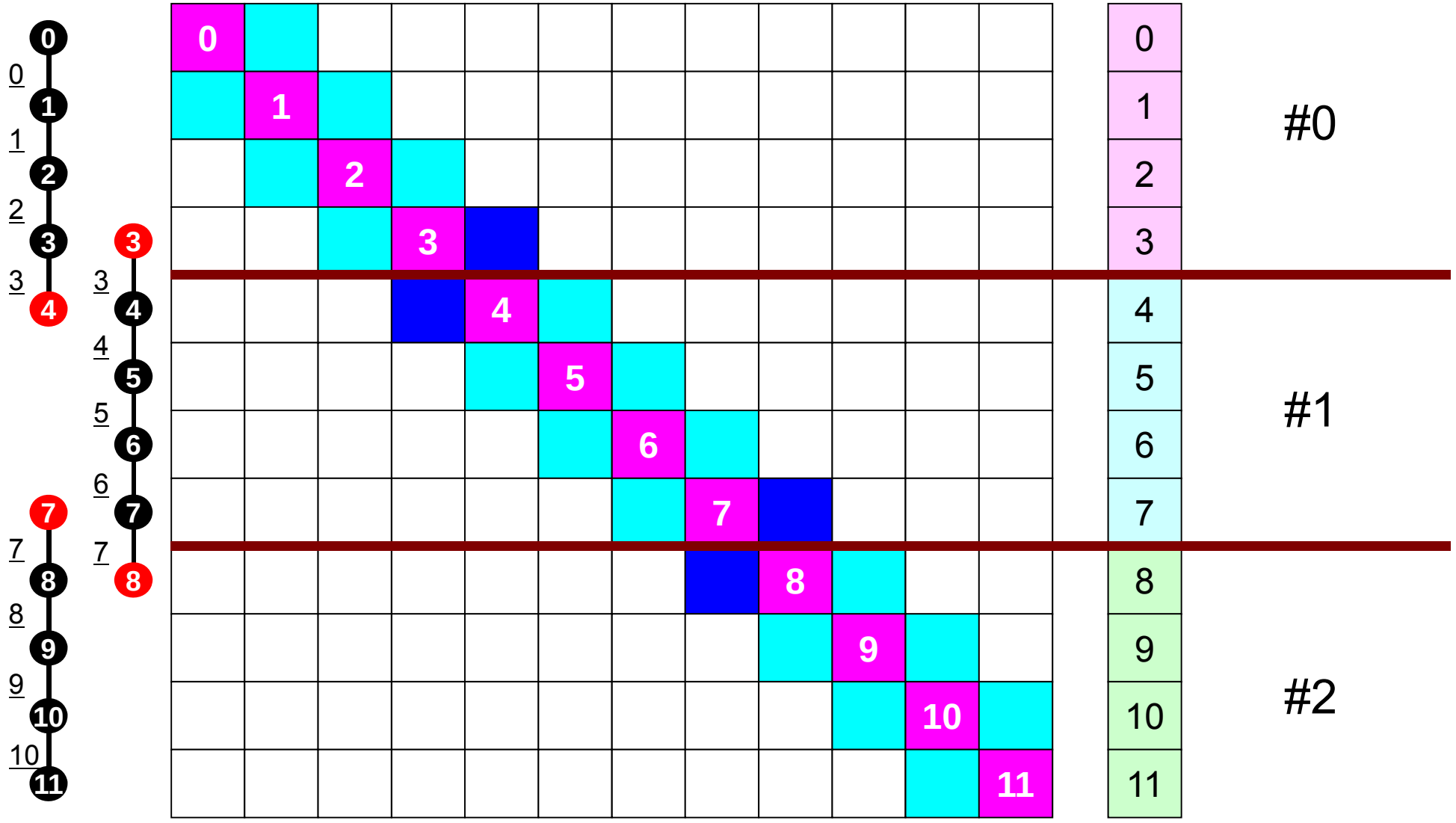
内点だけで分割するとマトリクス不完全



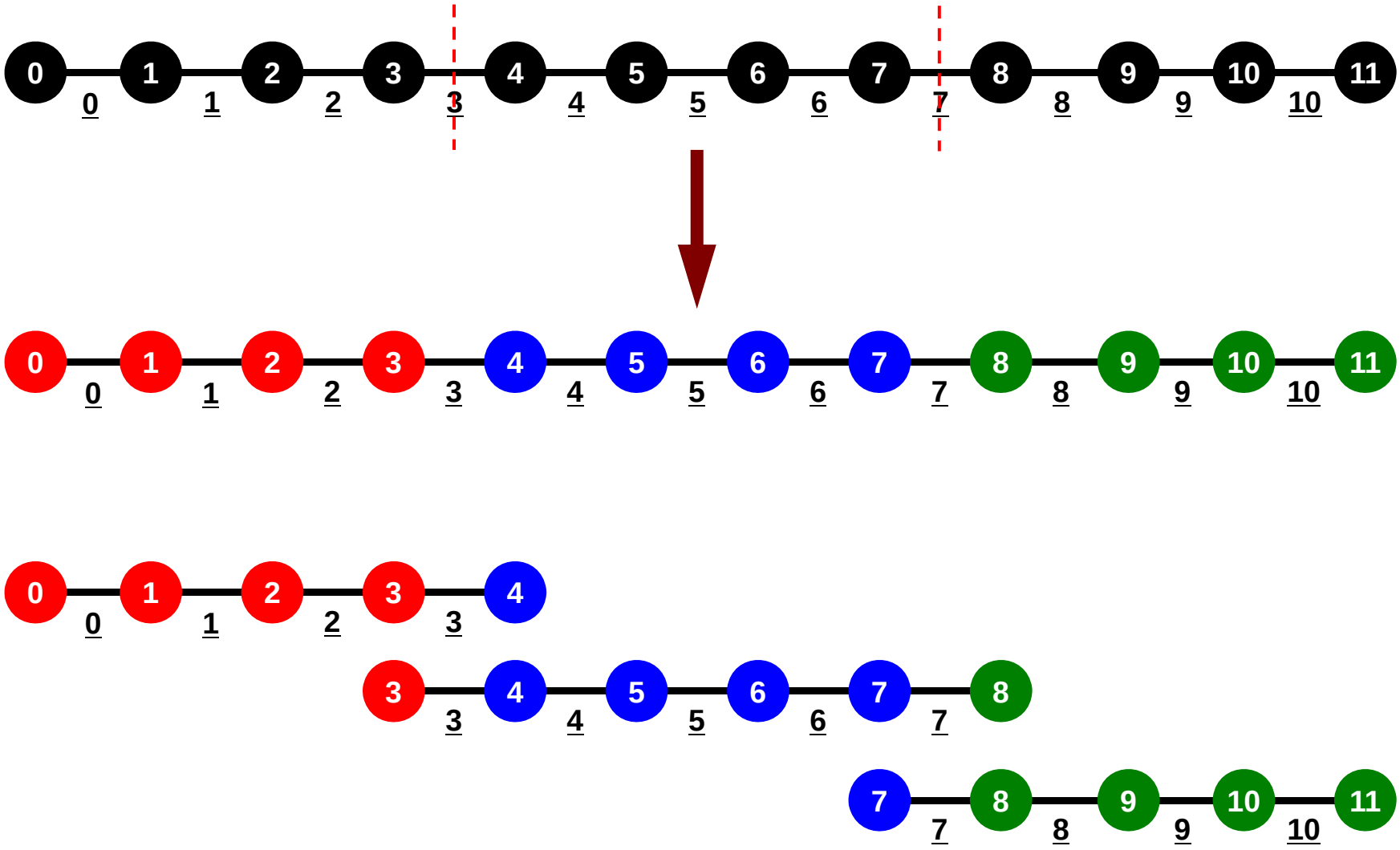
要素＋外点



一次元問題：11要素，12節点，3領域



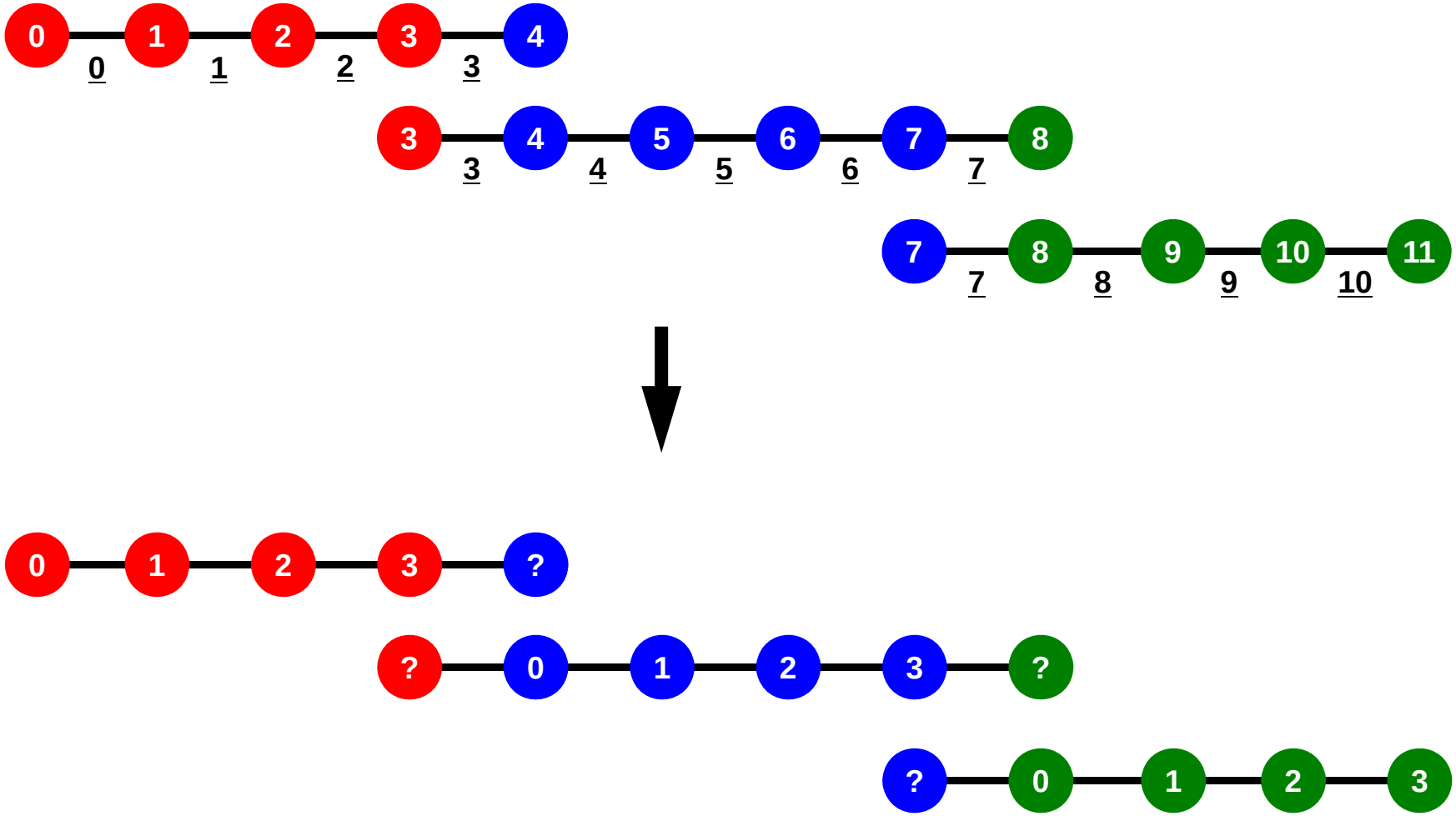
一次元問題：11要素，12節点，3領域



SPMD向け局所番号付け

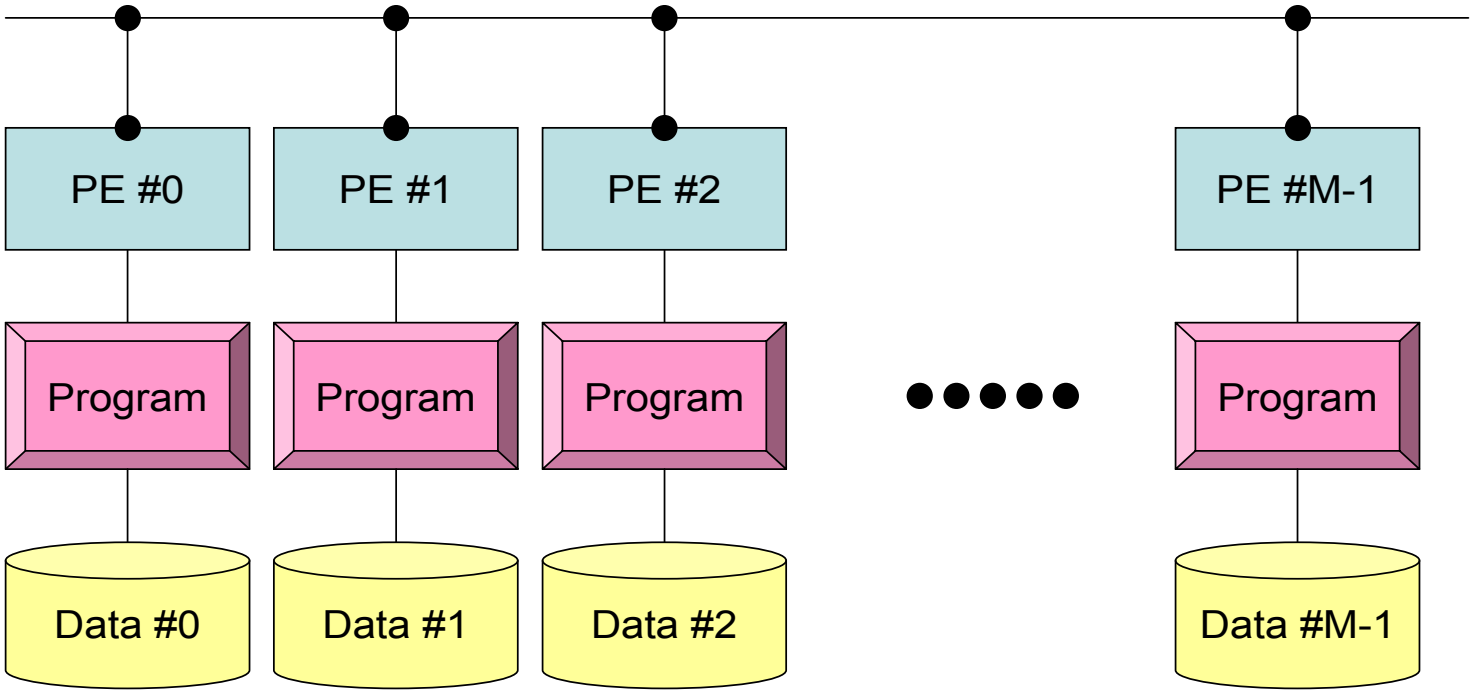
内点が1~N（0~N）となっていれば，もとのプログラムと同じ

外点の番号は?



MPIによる並列化：SPMD

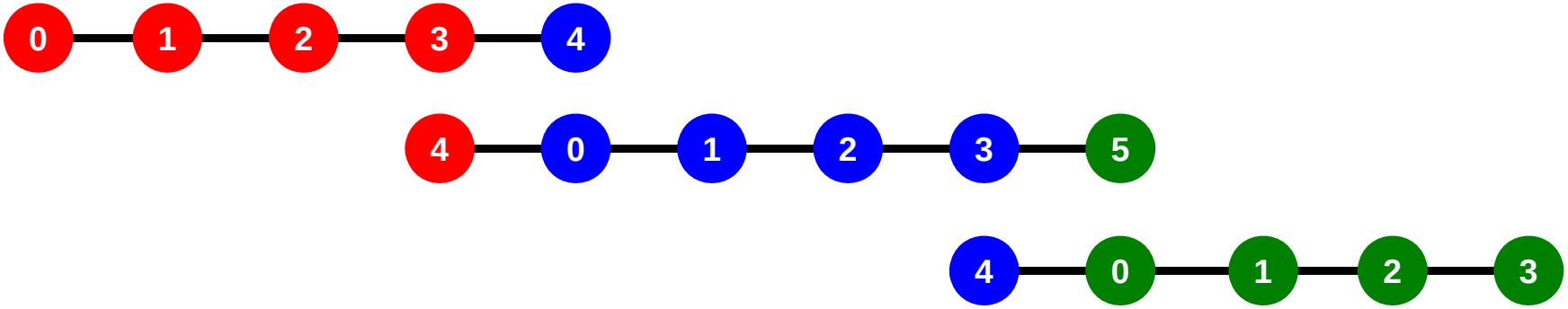
- Single Program/Instruction Multiple Data
- 基本的に各プロセスは「同じことをやる」が「データが違う」
  - 大規模なデータを分割し，各部分について各プロセス（プロセッサ）が計算する
- 全体データと局所データ，全体番号と局所番号
- 通信以外は単体CPUと同じ，というのが理想



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

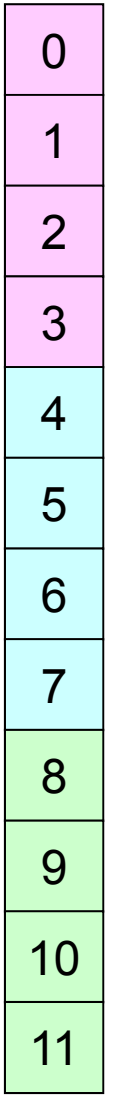
前処理：対角スケーリング

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];
  }
```

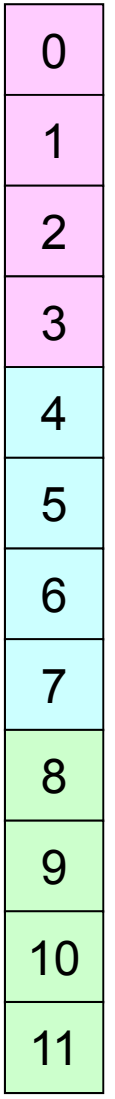


内積

全体で和をとる必要がある⇒通信?

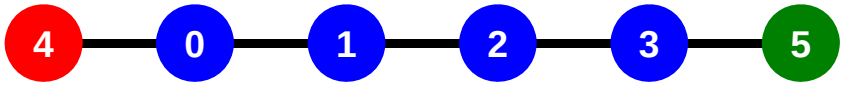
```
/*
//-- ALPHA= RHO / {p}{q}
*/
  C1 = 0.0;
  for(i=0;i<N;i++){
      C1 += W[P][i] * W[Q][i];
  }

  Alpha = Rho / C1;
```

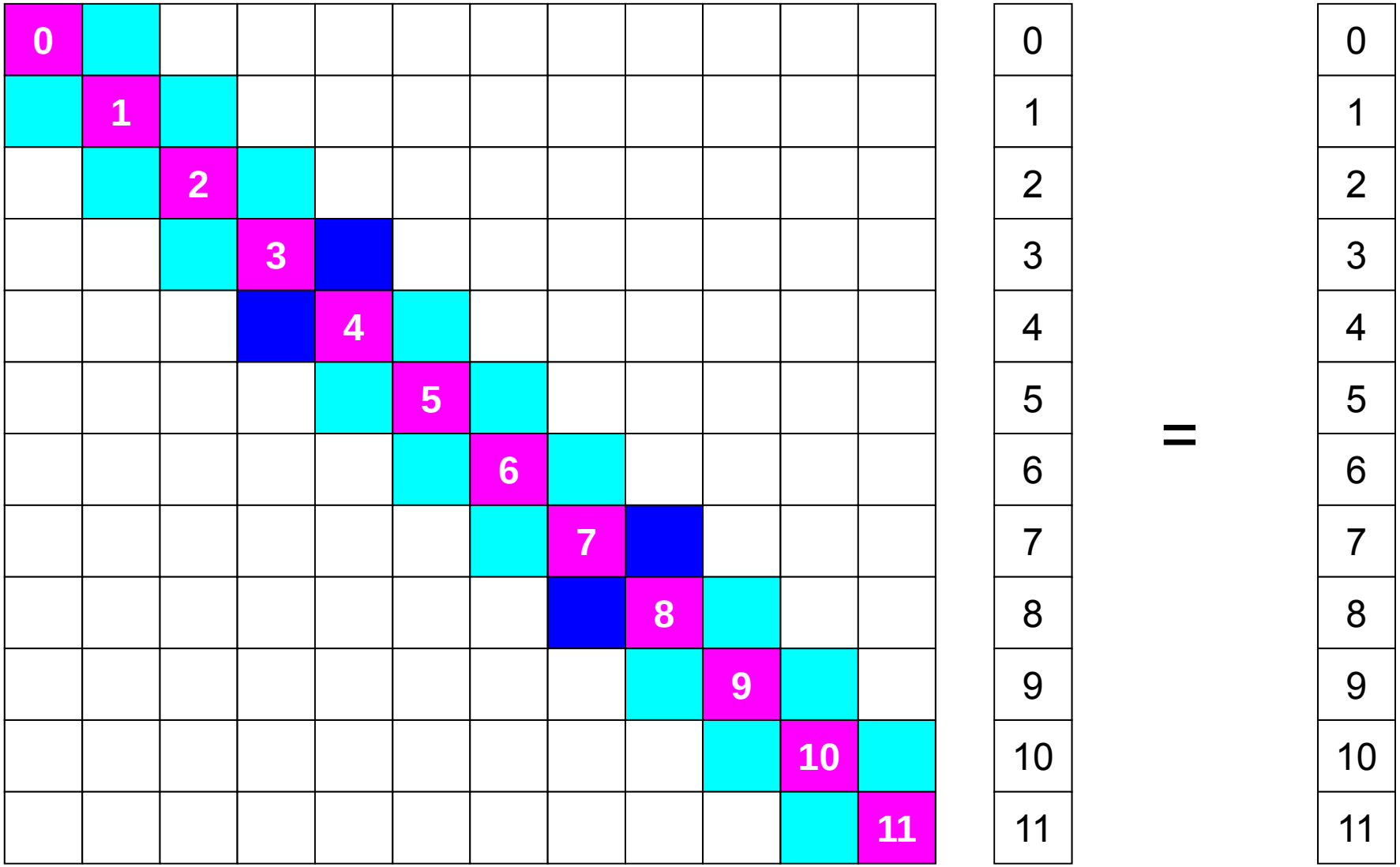


行列ベクトル積
外点の値が必要⇒通信?

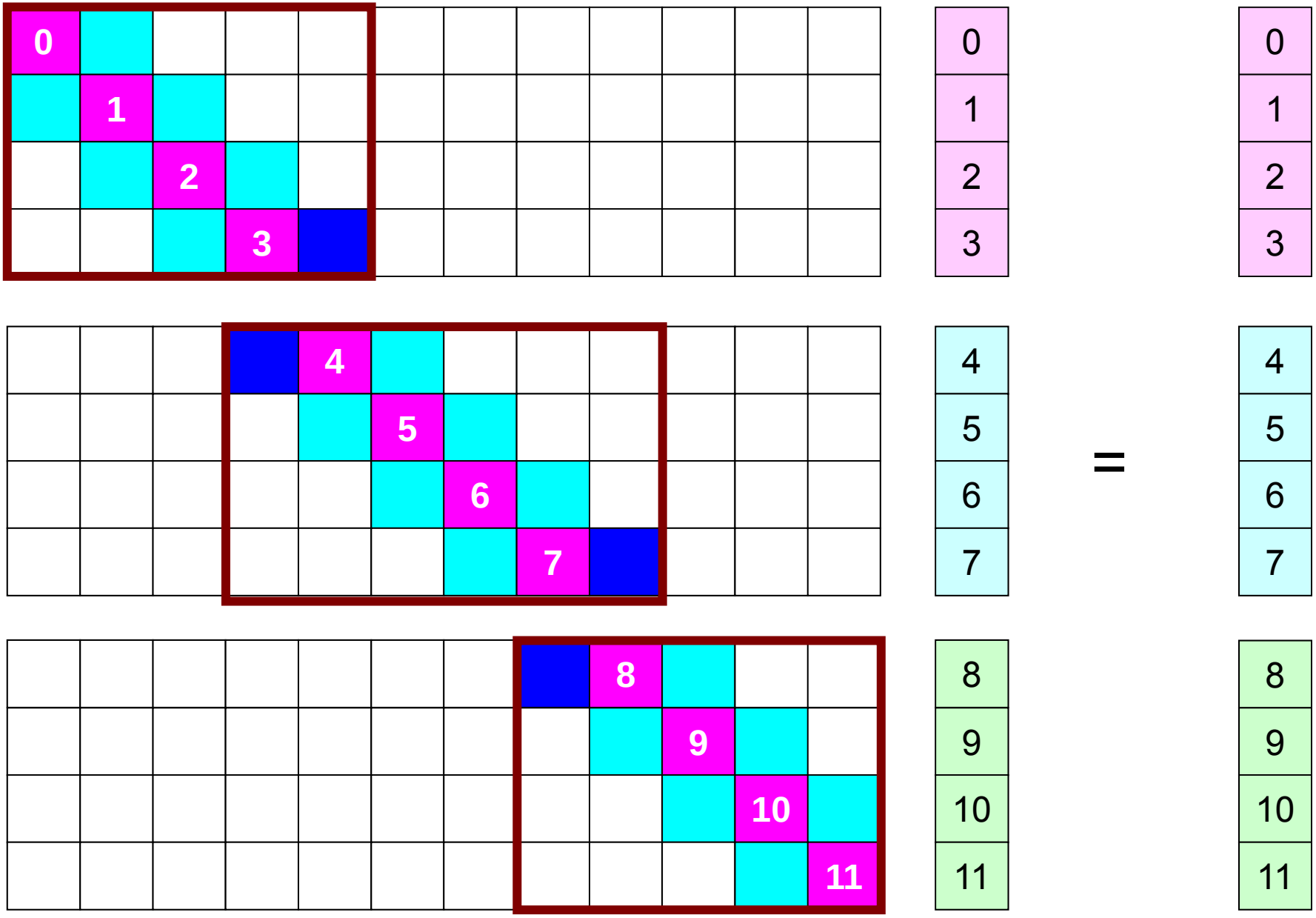
```
/*
//-- {q}= [A]{p}
*/
  for(i=0;i<N;i++){
      W[Q][i] = Diag[i] * W[P][i];
      for(j=Index[i];j<Index[i+1];j++){
          W[Q][i] += AMat[j]*W[P][Item[j]];
          }
      }
```



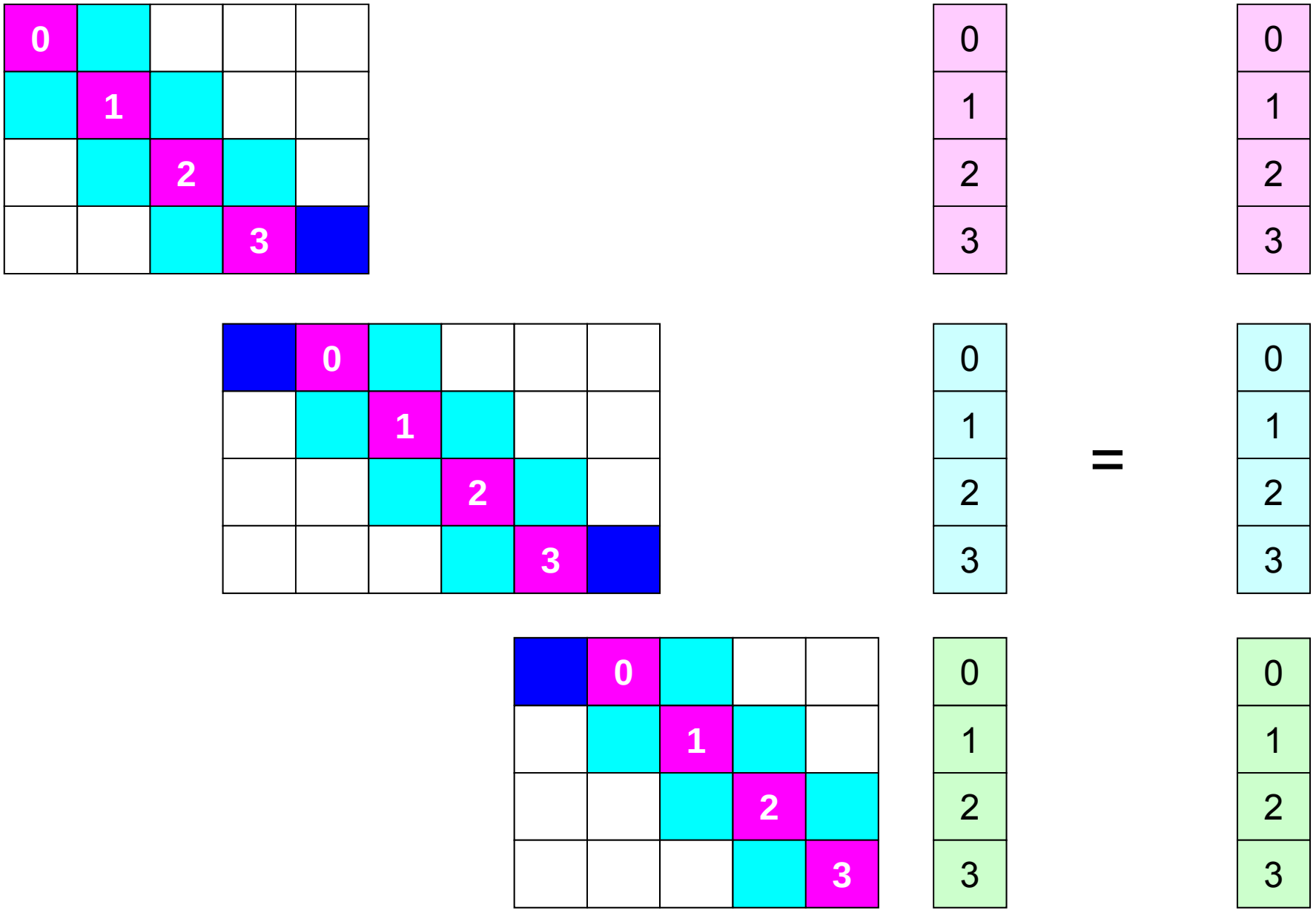
行列ベクトル積：ローカルに計算実施可能



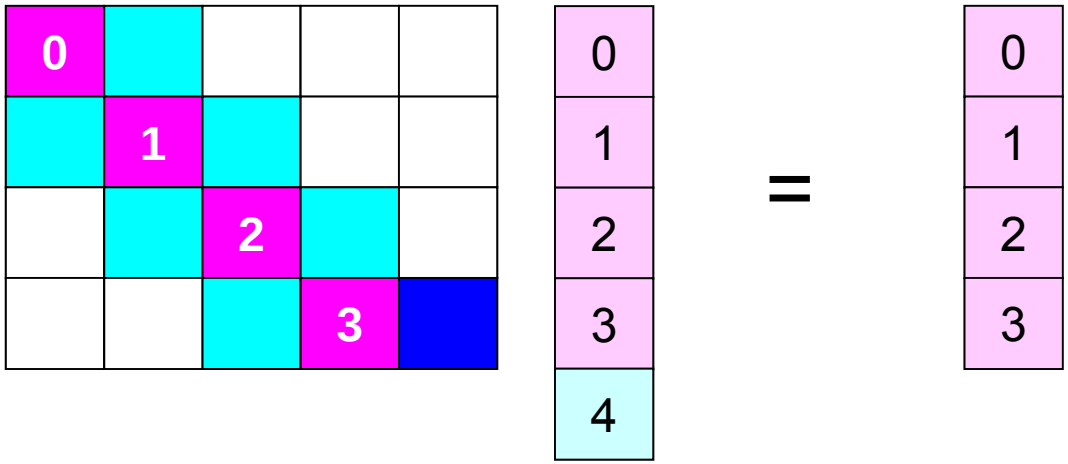
行列ベクトル積：ローカルに計算実施可能



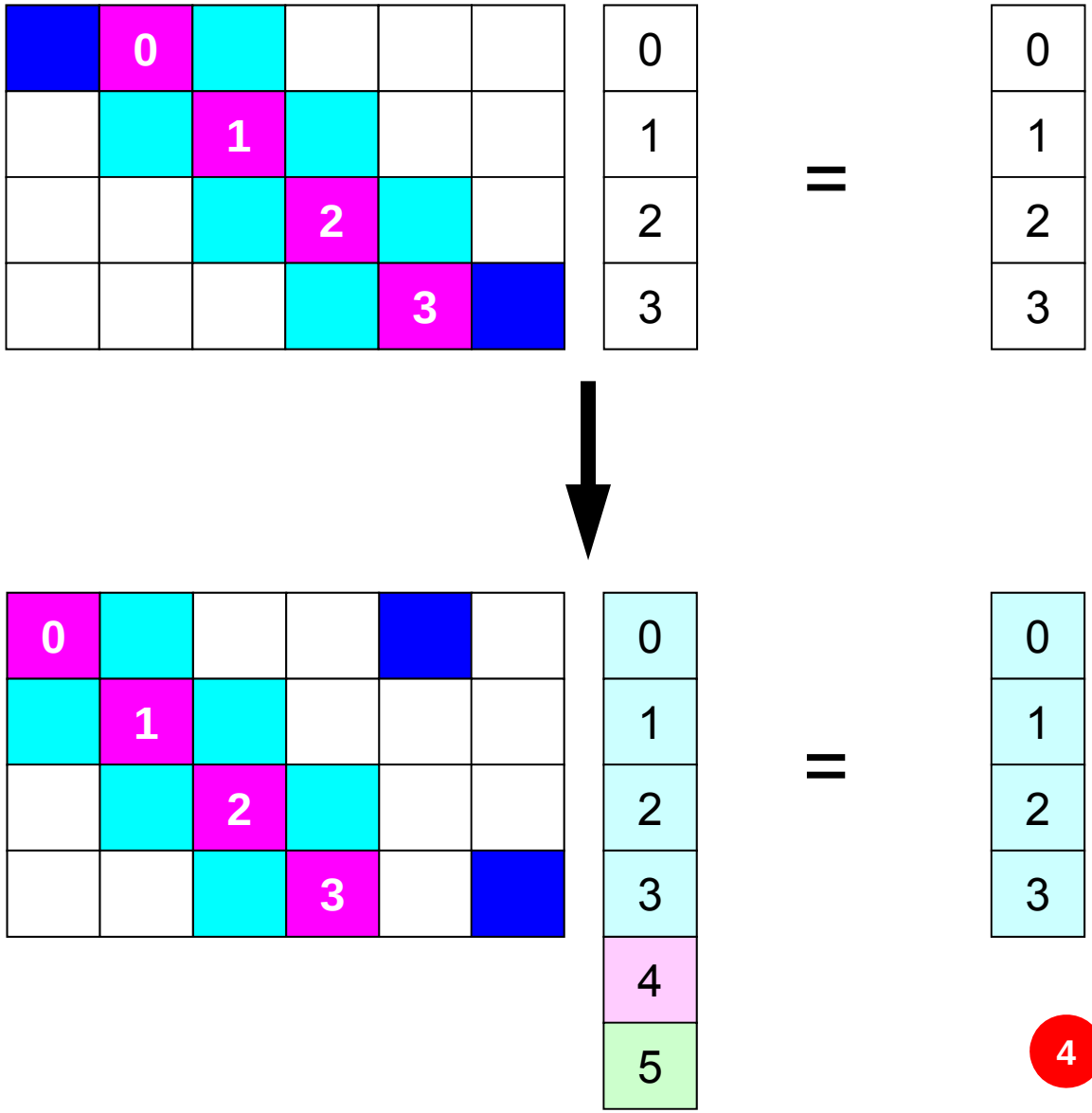
行列ベクトル積：ローカルに計算実施可能



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



行列ベクトル積:ローカル計算 #2