並列アプリケーション開発法入門(II)
有限体積法:局所分散データ構造・領域分割

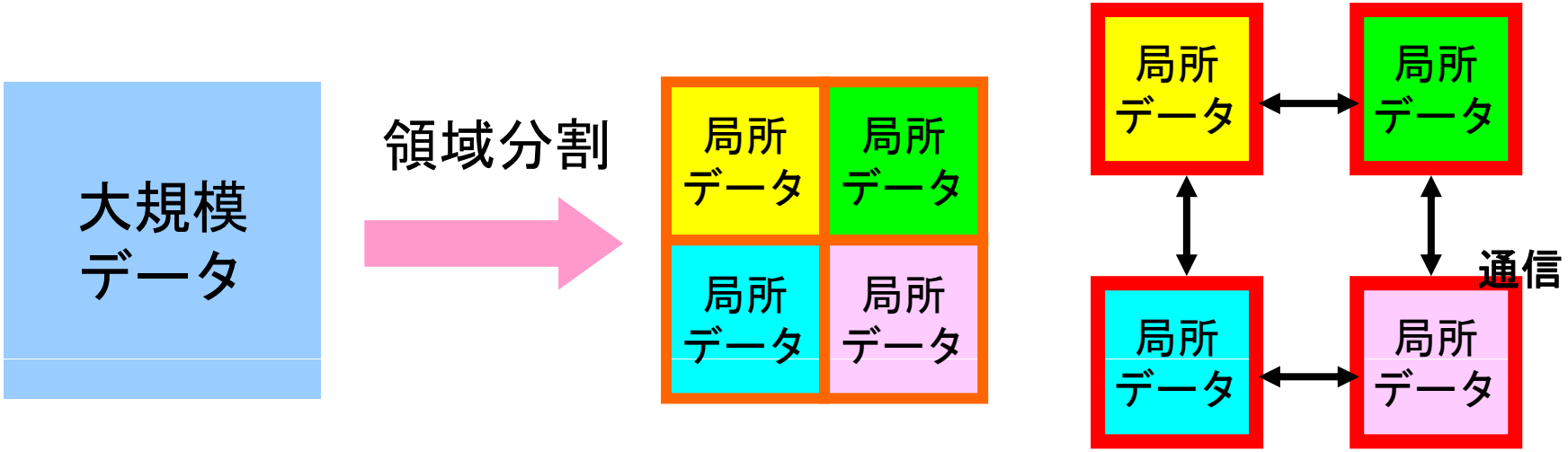
2011年夏季集中講義

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)

領域分割とは?

- 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



- データ構造とアルゴリズム：局所分散データ（復習）
- 有限体積法（FVM）における並列計算と局所分散データ構造の考え方
- 領域分割手法について
- eps_fvmにおける領域分割機能：eps_fvm_part
- eps_fvm「並列化」に向けて

データ構造とアルゴリズム

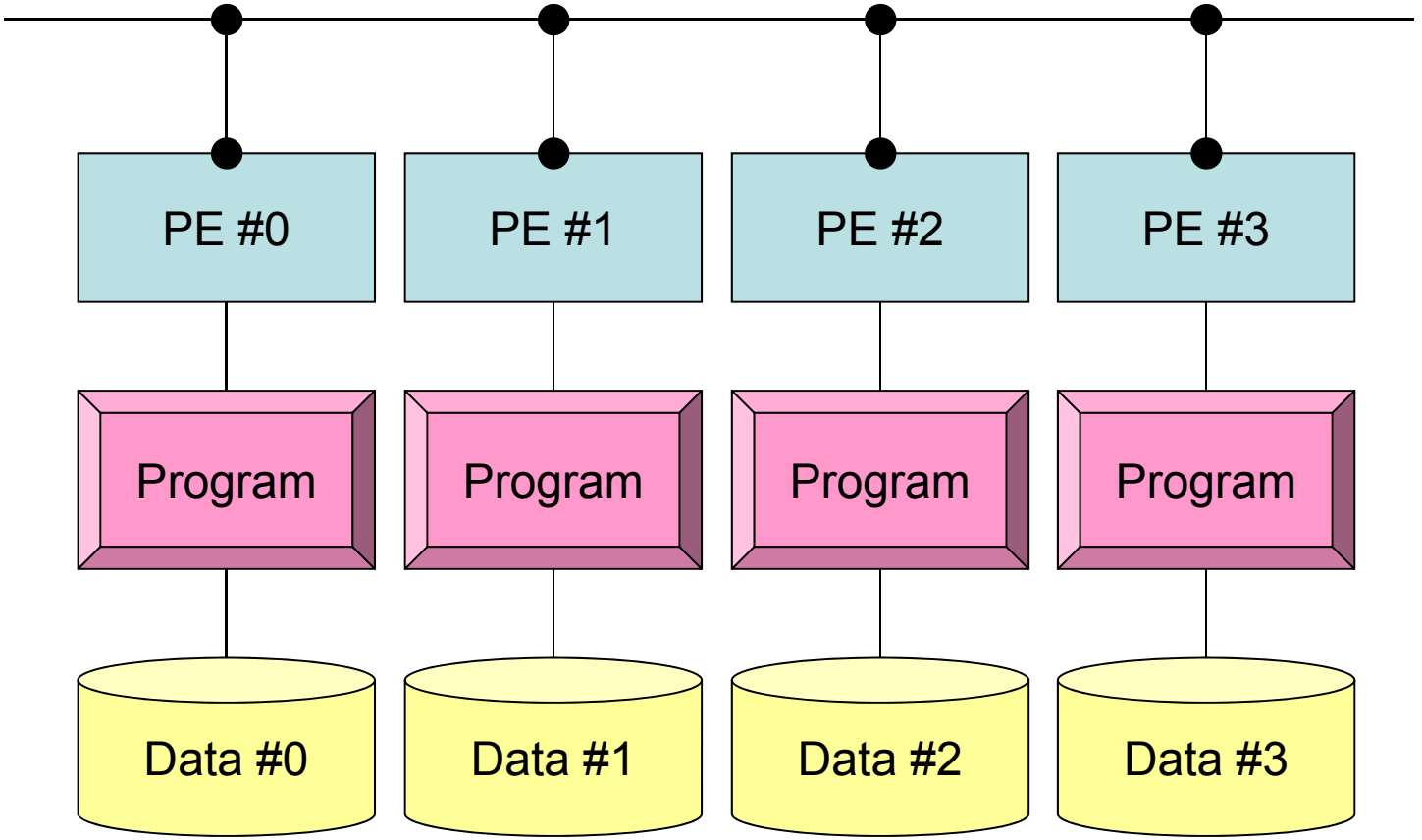
- コンピュータ上で計算を行うプログラムはデータ構造とアルゴリズムから構成される。
- 両者は非常に密接な関係にあり，あるアルゴリズムを実現するためには，それに適したデータ構造が必要である。
  - 極論を言えば「データ構造＝アルゴリズム」と言っても良い。

- 並列計算を始めるにあたって，基本的なアルゴリズムに適したデータ構造を定める必要がある。

SPMD：Single Program Multiple Data

- 一言で「並列計算」と言っても色々なものがあり，基本的なアルゴリズムも様々。
- 共通して言えることは，SPMD（Single Program Multiple Data）

SPMDに適したデータ構造とは？



SPMDに適したデータ構造（1/2）

- 大規模なデータ領域を分割して，各プロセッサ，プロセスで計算するのがSPMDの基本的な考え方
- 例えば長さNG(=20)のベクトル**VG**に対して以下のような計算を考えてみよう:

```
integer, parameter :: NG= 20
real(kind=8), dimension(20) :: VG

do i= 1, NG
  VG(i)= 2.0 * VG(i)
enddo
```

- これを4つのプロセッサで分担して計算するとすれば，20/4=5 ずつ記憶し，処理すればよい。

SPMDに適したデータ構造(2/2)

- すなわち，こんな感じ：

```
integer, parameter :: NL= 5
real(kind=8), dimension(5) :: VL

do i= 1, NL
  VL(i)= 2.0 * VL(i)
enddo
```

- このようにすれば「一種類の」プログラム（Single Program）で並列計算を実施できる。
  - 各プロセスにおいて，「VL」の中身が違う：Multiple Data
  - もとのプログラムともほぼ同じ
  - 可能な限り計算を「VL」のみで実施することが，並列性能の高い計算へつながる。

全体データと局所（分散）データ

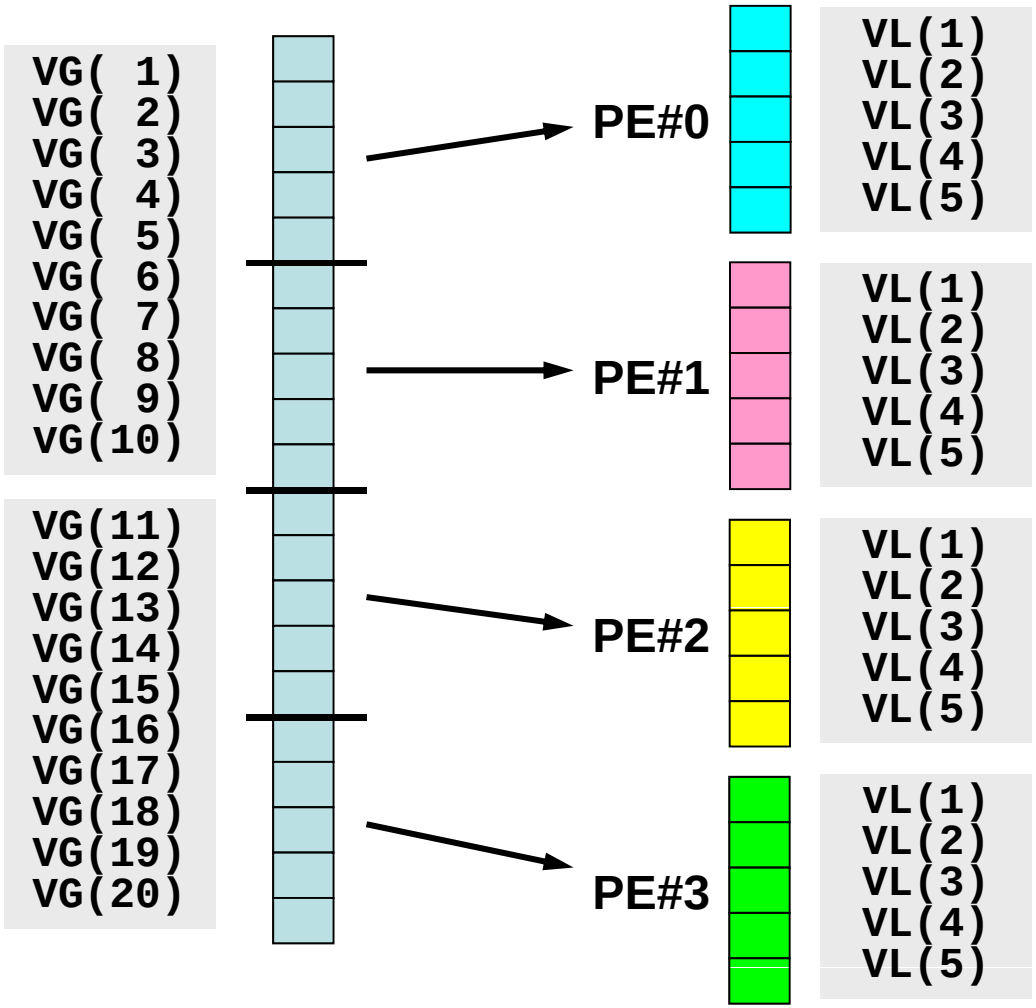
- VG
  - 領域全体
  - 1番から20番までの「全体番号」を持つ「全体データ（Global Data）」
- VL
  - 各プロセス（PE，プロセッサ，領域）
  - 1番から5番までの「局所番号」を持つ「局所（分散）データ（Local (Distributed) Data）」
  - できるだけ局所（分散）データを有効に利用することで，高い並列性能が得られる。

局所(分散)データの考え方

「全体データ」VGの:

- 1～5番成分が0番PE
- 6~10番成分が1番PE
- 11～15番が2番PE
- 16～20番が3番PE

のそれぞれ,「局所(分散)データ」VLの1番～5番成分となる(局所番号が1番～5番となる)。



全体データと局所(分散)データ

- VG
  - 領域全体
  - 1番から20番までの「全体番号」を持つ「全体データ(Global Data)」
- VL
  - 各プロセッサ
  - 1番から5番までの「局所番号」を持つ「局所(分散)データ(Local (Distributed) Data)」

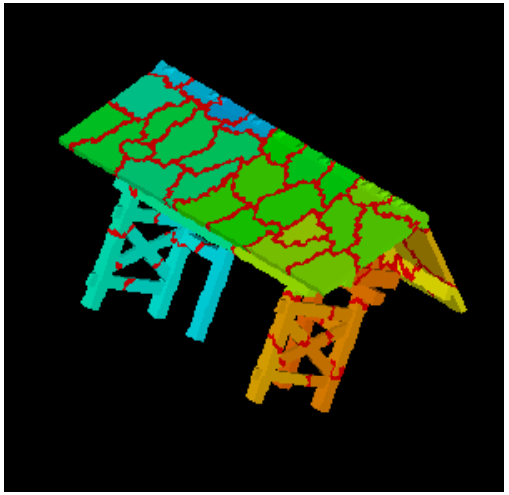
- この講義で考えること
  - VG(全体データ)からVL(局所(分散)データ)をどのように生成するか。
  - VGからVL，VLからVGへデータの中身をどのようにマッピングするか。
  - VLがプロセスごとに独立して計算できない場合はどうするか。
  - できる限り「局所性」を高めた処理を実施する⇒高い並列性能

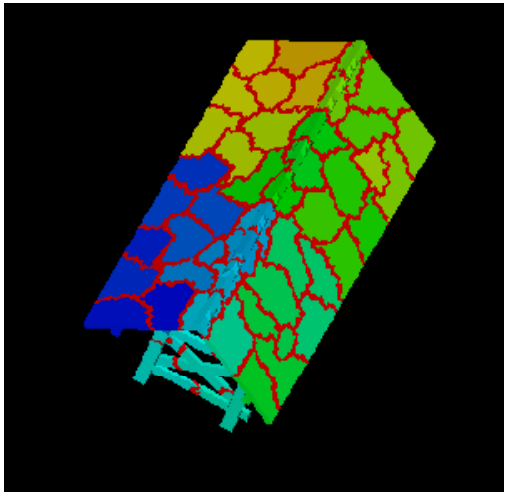
局所「分散」データ

- 大規模な全体データ（global data）を局所分散データ（local distributed data）に分割して，SPMDによる並列計算を実施する場合のデータ構造について考える。
-

局所分散データ構造

- 対象とする計算（のアルゴリズム）に適した局所分散データ構造を定めることが重要
  - アルゴリズム＝データ構造
- この講義の主たる目的の一つ





局所「分散」データ

- 大規模な全体データ（global data）を局所データ（local data）に分割して，SPMDによる並列計算を実施する場合のデータ構造について考える。

- 下記のような長さ20のベクトル，VECpとVECsの内積計算を4つのプロセッサ，プロセスで並列に実施することを考える。

```
VECp( 1)=  2
     ( 2)=  2
     ( 3)=  2
…
     (18)=  2
     (19)=  2
     (20)=  2
```

```
VECs( 1)=   3
     ( 2)=   3
     ( 3)=   3
…
     (18)=   3
     (19)=   3
     (20)=   3
```

内積の計算（長さ20のベクトル）

```
implicit REAL*8 (A-H,O-Z)
real(kind=8),dimension(20):: &
    VECp,  VECs

N=20
do i= 1, N
  VECp(i)= 2.0d0
  VECs(i)= 3.0d0
enddo

sum= 0.d0
do ii= 1, N
  sum= sum + VECp(ii)*VECs(ii)
enddo

stop
end
```

```
#include <stdio.h>
int main(){
   int i, N;
   double VECp[20], VECs[20]
   double sum;

   N=20;
   for(i=0;i<N;i++){
     VECp[i]= 2.0;
     VECs[i]= 3.0;
   }

   sum = 0.0;
   for(i=0;i<N;i++){
     sum += VECp[i] * VECs[i];
   }
   return 0;
}
```

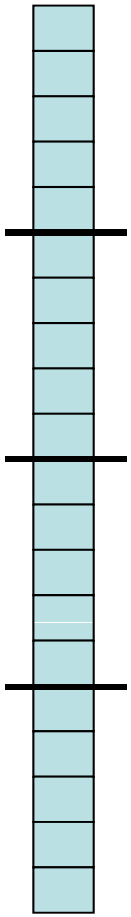
<$E-S1>/dot.f, dot.c

局所分散データの考え方(1/2)

- 長さ20のベクトルを，4つに分割する
- 各プロセスで長さ5のベクトル(1~5)

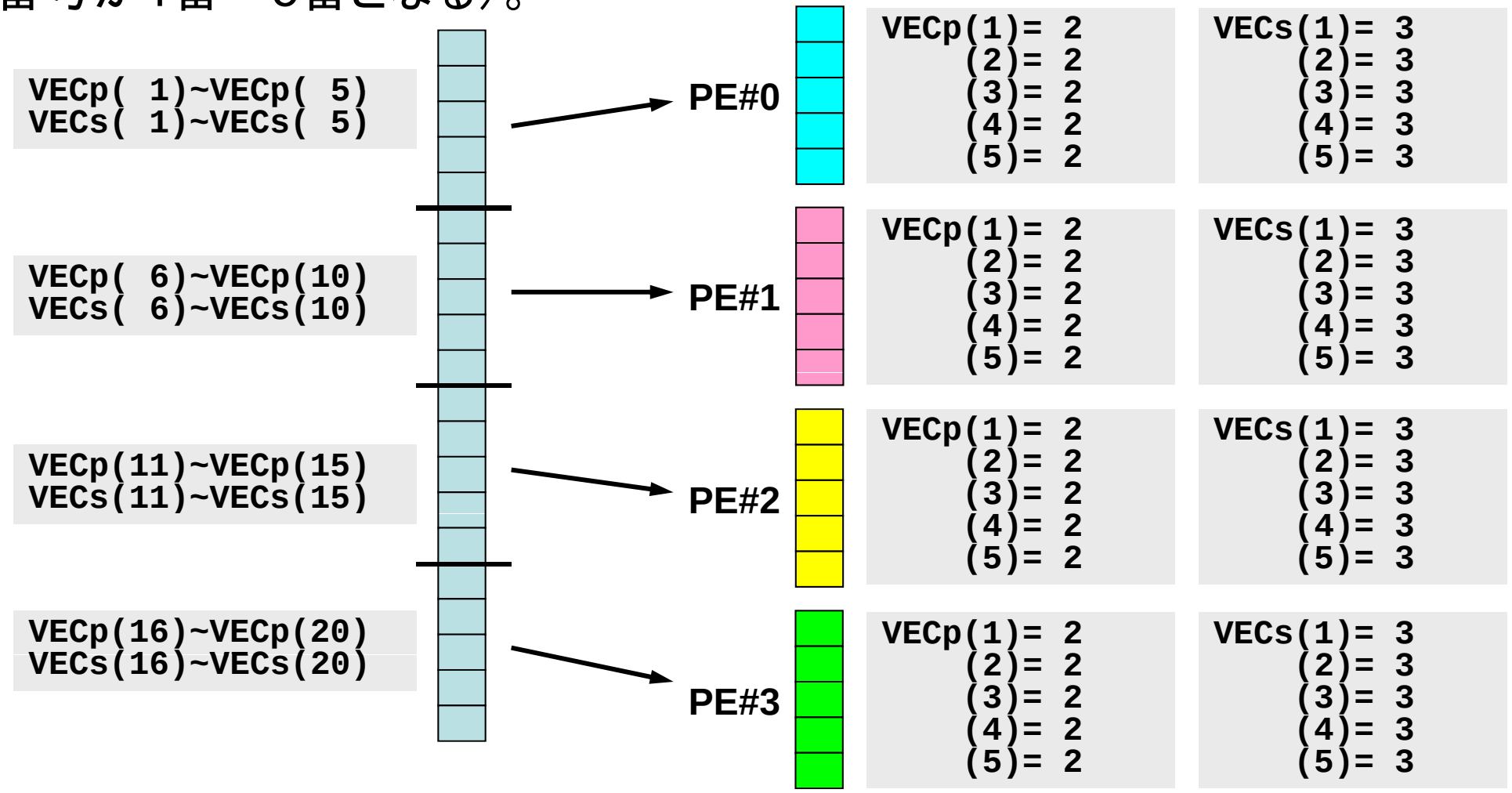
```
VECp( 1)=  2
    ( 2)=  2
    ( 3)=  2
…
    (18)=  2
    (19)=  2
    (20)=  2
```

```
VECs( 1)=   3
    ( 2)=   3
    ( 3)=   3
…
    (18)=   3
    (19)=   3
    (20)=   3
```



局所分散データの考え方(2/2)

- もとのベクトルの1～5番成分が0番PE，6~10番成分が1番PE，11～15番が2番PE，16～20番が3番PEのそれぞれ1番～5番成分となる(局所番号が1番～5番となる)。



内積の並列計算例(1/2)

```
implicit REAL*8 (A-H,O-Z)
include 'mpif.h'
integer :: PETOT, my_rank, ierr
real(kind=8), dimension(5) :: VECp,  VECs

call MPI_INIT      (ierr)
call MPI_COMM_SIZE (MPI_COMM_WORLD, PETOT, ierr )
call MPI_COMM_RANK (MPI_COMM_WORLD, my_rank, ierr )

sumA= 0.d0
sumR= 0.d0
N= 5
do i= 1, N
  VECp(i)= 2.d0
  VECs(i)= 3.d0
enddo

sum0= 0.d0
do i= 1, N
  sum0= sum0 + VECp(i) * VECs(i)
enddo
```

各ベクトルを各プロセスで独立に生成する

dot.f, dot.cとプログラムが変わっていないことに注意（Nの値が変わっただけ）

内積の並列計算例(2/2)

```
!C
!C-- ALL-REDUCE
  call MPI_allREDUCE (sum0, sumA, 1, MPI_DOUBLE_PRECISION, MPI_SUM, &
                      MPI_COMM_WORLD, ierr)

  write (*,'(a,i5, 2(1pe16.6))') 'DOT product', my_rank, sumA

  call MPI_FINALIZE (ierr)

  stop
  end
```

内積の計算

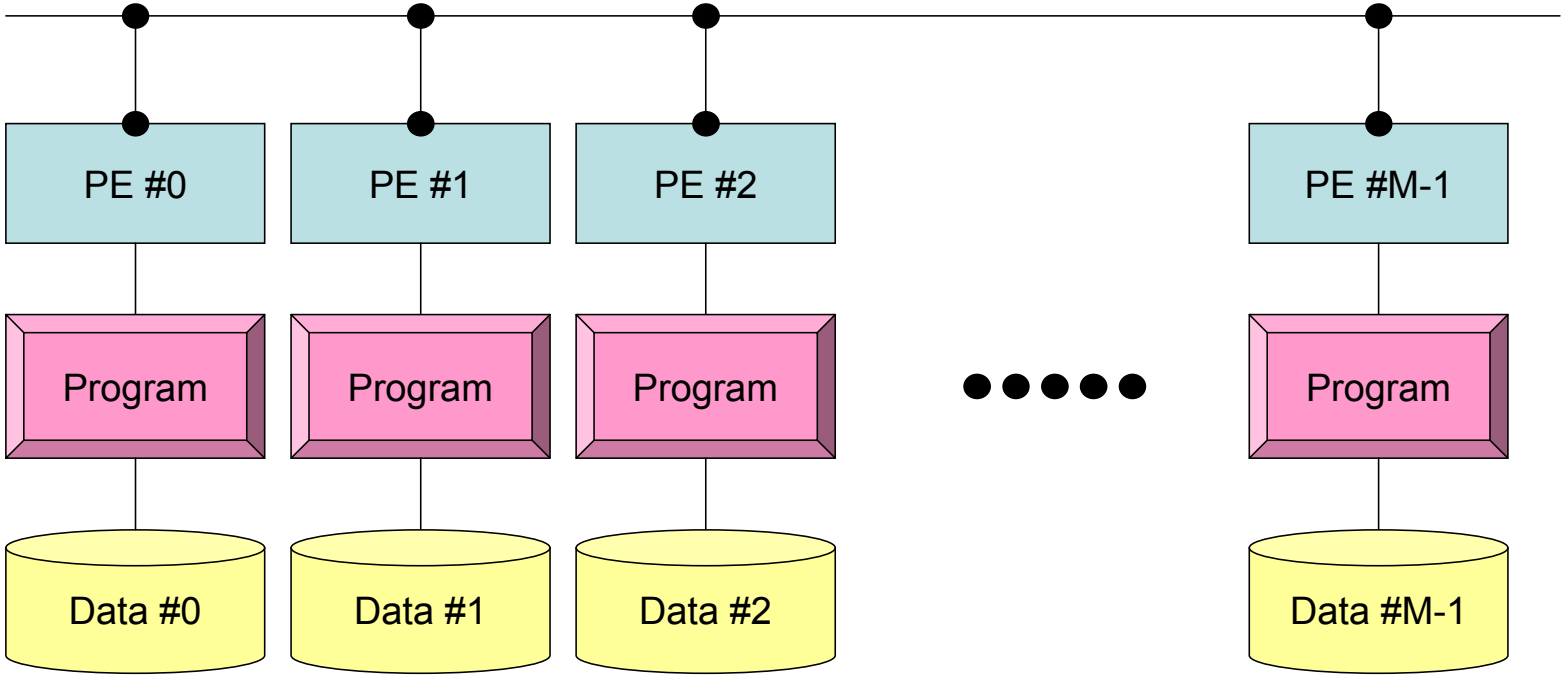
各プロセスで計算した結果「sum0」の総和をとる

sumA には，MPI_ALLREDUCEによって全プロセスに計算結果が入る。

- データ構造とアルゴリズム：局所分散データ
- **有限体積法（FVM）における並列計算と局所分散データ構造の考え方**
- 領域分割手法について
- eps_fvmにおける領域分割機能：eps_fvm_part
- eps_fvm「並列化」に向けて

（並列）局所分散データ構造

- 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。
- 「有限体積法」，共役勾配法によるアプリケーションを，MPIによるSPMDパラダイムによって並列化するためのデータ構造・・・を考える　⇒　内積ほど容易ではなかろう



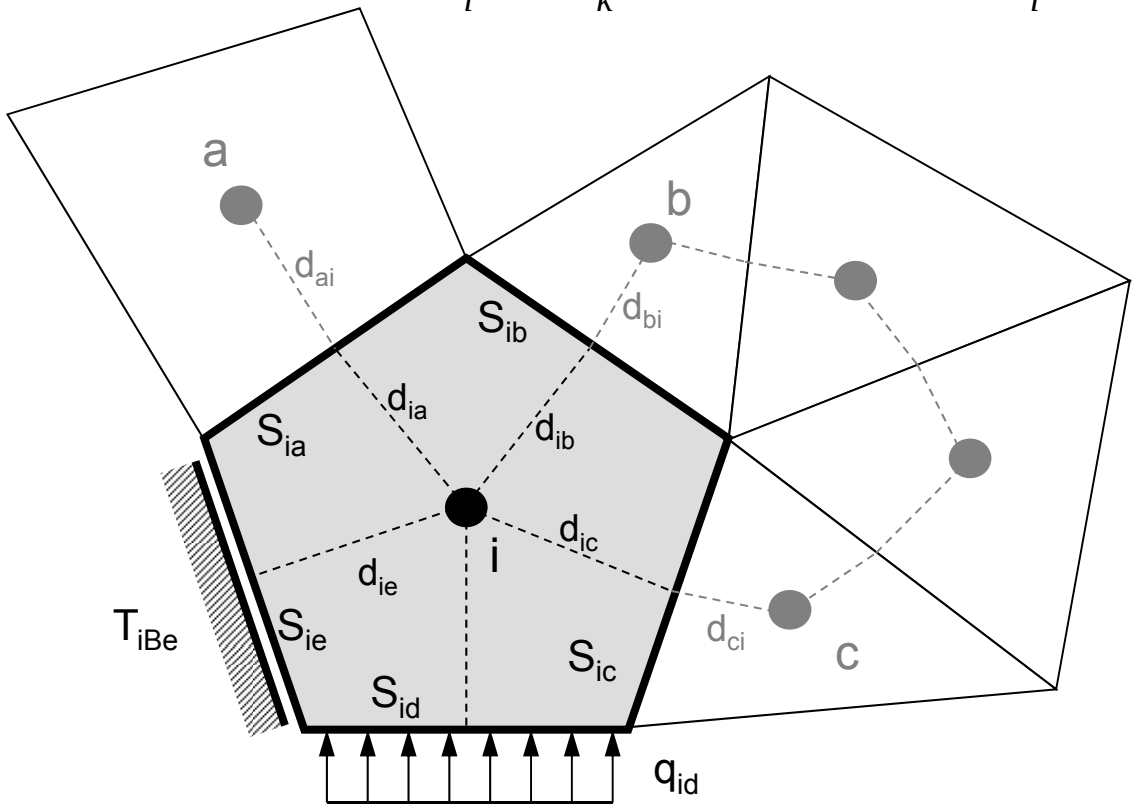
有限体積法による空間離散化

熱流束に関するつりあい式

隣接要素との熱伝導　　温度固定境界

$$\sum_k \frac{S_{ik}}{\frac{d_{ik}}{\lambda_i}+\frac{d_{ki}}{\lambda_k}}\left(T_k - T_i\right) + \sum_e \frac{S_{ie}}{\frac{d_{ie}}{\lambda_i}}\left(T_{iBe} - T_i\right) + \sum_d S_{id}\dot{q}_{id} + V_i\dot{Q}_i = 0$$

要素境界面通過熱流束　　体積発熱



- λ : 熱伝導率
- V_i : 要素体積
- S : 表面面積
- d_{ij} : 要素中心から表面までの距離
- q : 表面フラックス
- Q : 体積発熱
- T_{iB} : 境界温度

並列計算を実施する場合には?

21	22	23	24	25
16	17	18	19	20
11	12	13	14	15
6	7	8	9	10
1	2	3	4	5

並列計算を実施する場合には?

21	22	23	24	25
16	17	18	19	20
11	12	13	14	15
6	7	8	9	10
1	2	3	4	5

3領域に分割

#PE2

21	22	23
16	17	18
11	12	13

#PE1

24	25
19	20
14	15
9	10

#PE0

6	7	8		
1	2	3	4	5

用語：PE

- Processing Element　の略
- もともと，並列計算機における個々の「CPU」を表す

- 領域分割による並列計算においては，転じて以下のようなものを指す用語としても使用される
  - MPIのプロセス
  - 領域分割したときの個々の領域

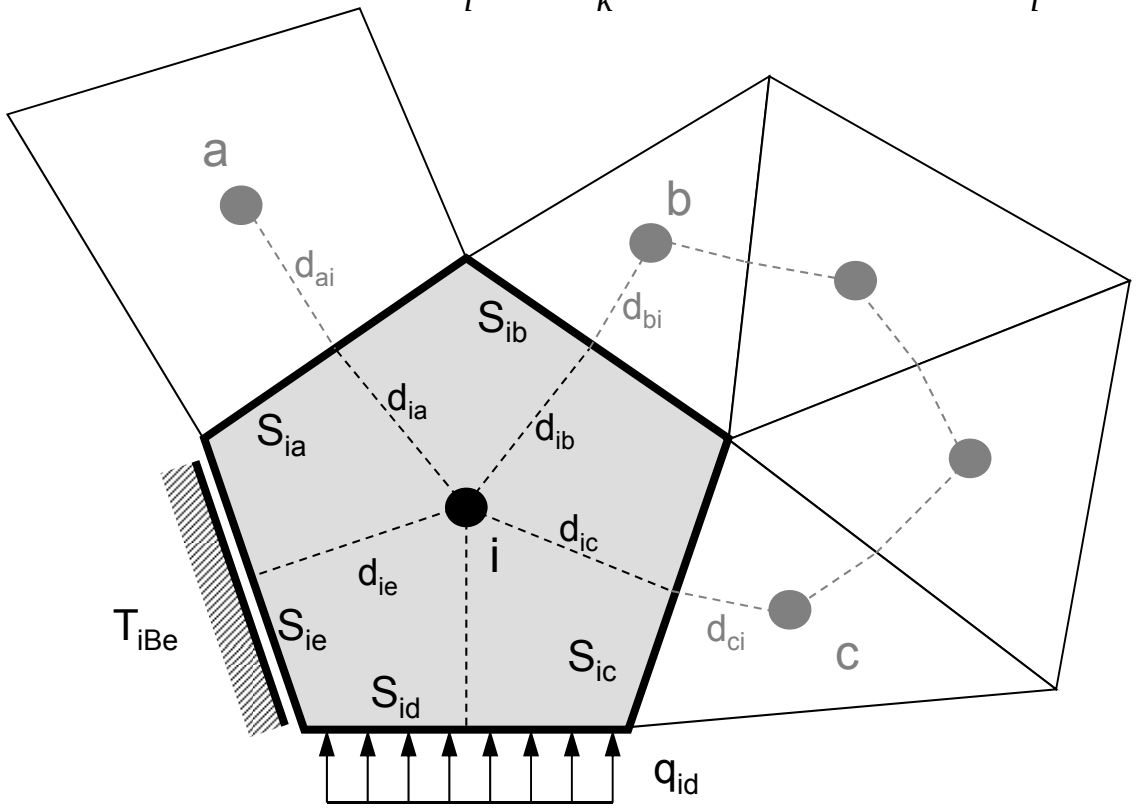
有限体積法：隣接メッシュの情報が必要

熱流束に関するつりあい式

隣接要素との熱伝導　　温度固定境界

$$\sum_k \frac{S_{ik}}{\frac{d_{ik}}{\lambda_i}+\frac{d_{ki}}{\lambda_k}}\left(T_k - T_i\right) + \sum_e \frac{S_{ie}}{\frac{d_{ie}}{\lambda_i}}\left(T_{iBe} - T_i\right) + \sum_d S_{id}\dot{q}_{id} + V_i\dot{Q}_i = 0$$

要素境界面通過熱流束　　体積発熱



λ　：熱伝導率
V_i　：要素体積
S　：表面面積
d_{ij}　：要素中心から表面までの距離
q　：表面フラックス
Q　：体積発熱
T_{iB}　：境界温度

オーバーラップ領域のデータ必要

#PE2

21	22	23	24
16	17	18	19
11	12	13	14
6	7	8	

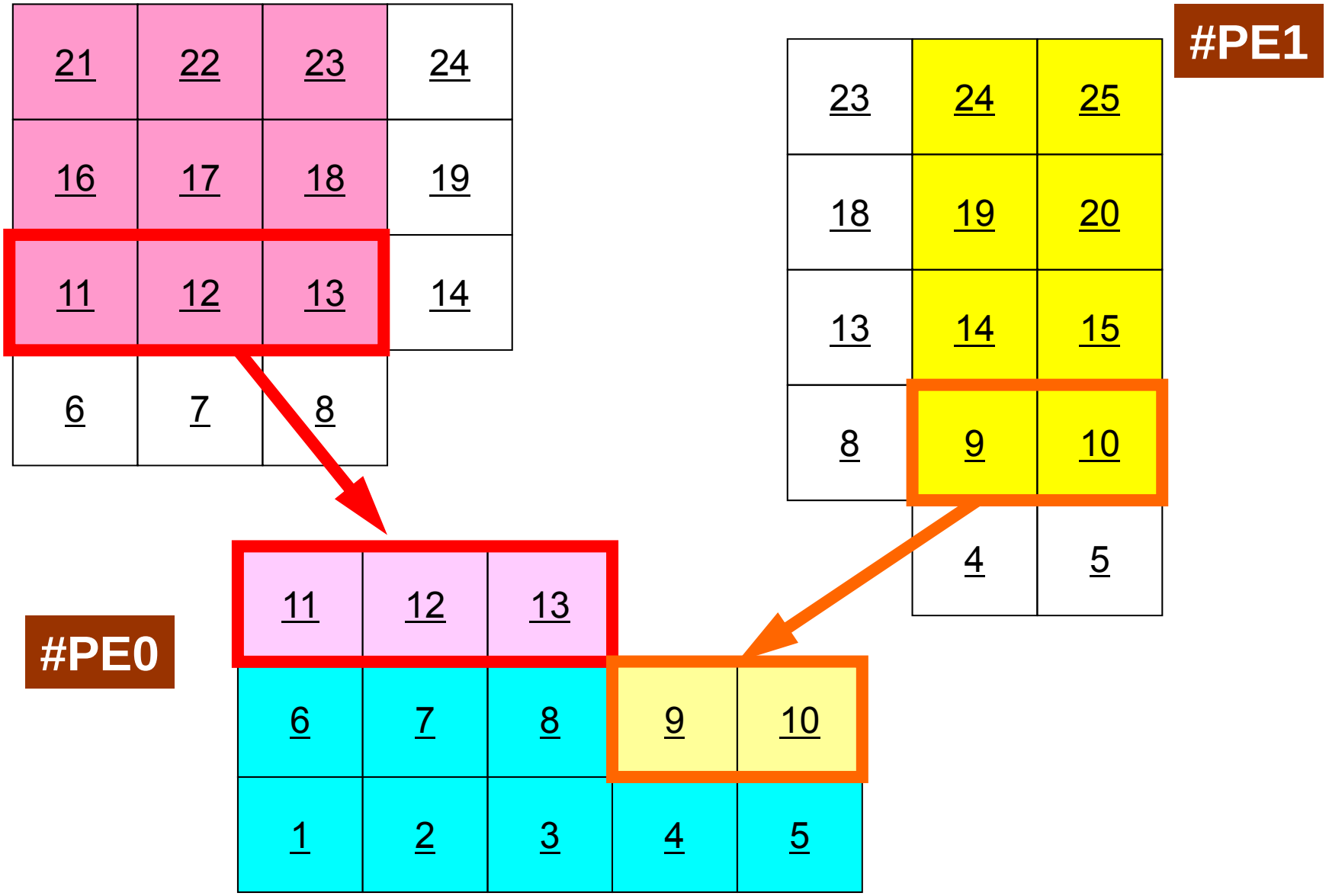
#PE1

23	24	25
18	19	20
13	14	15
8	9	10
	4	5

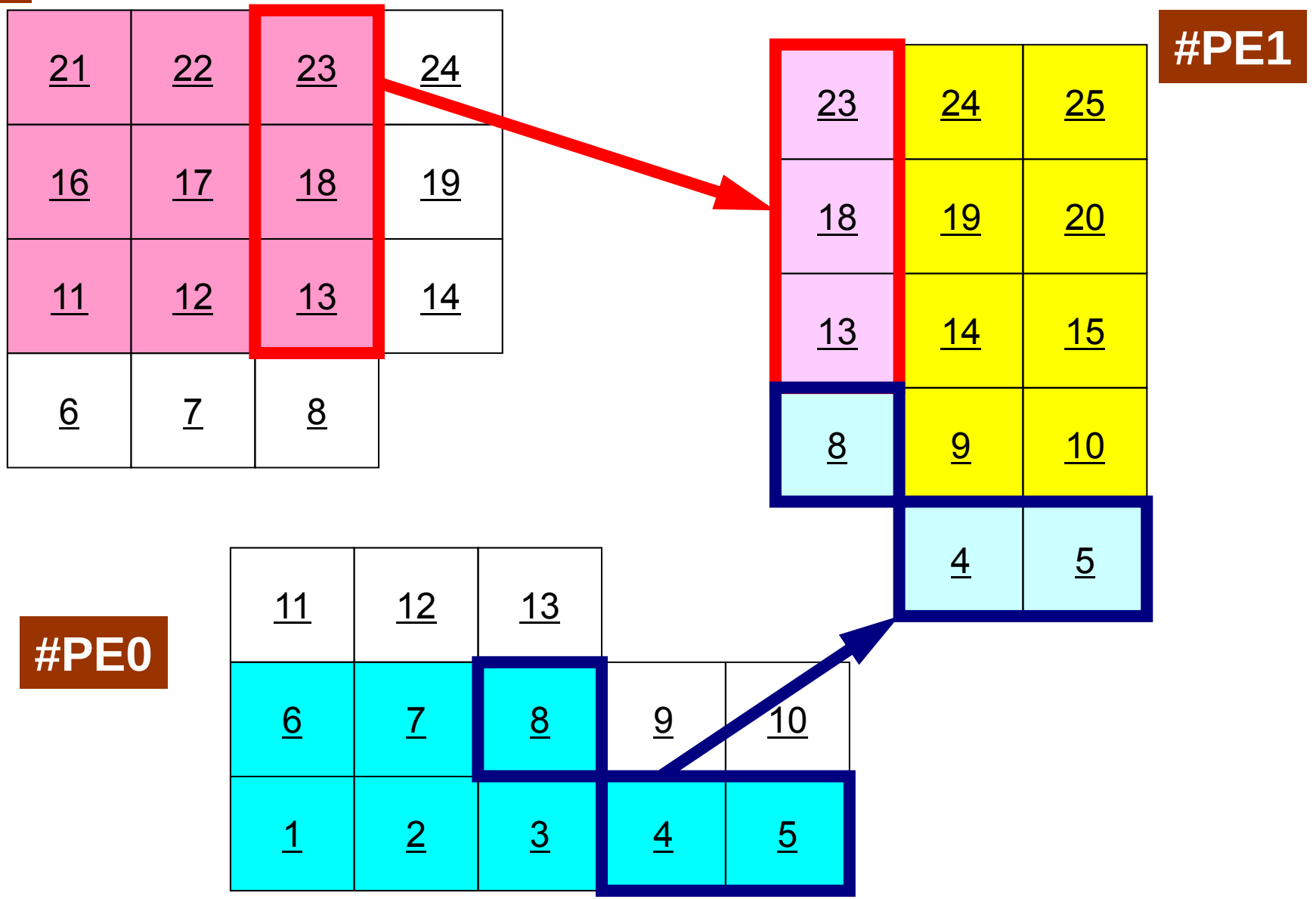
#PE0

11	12	13		
6	7	8	9	10
1	2	3	4	5

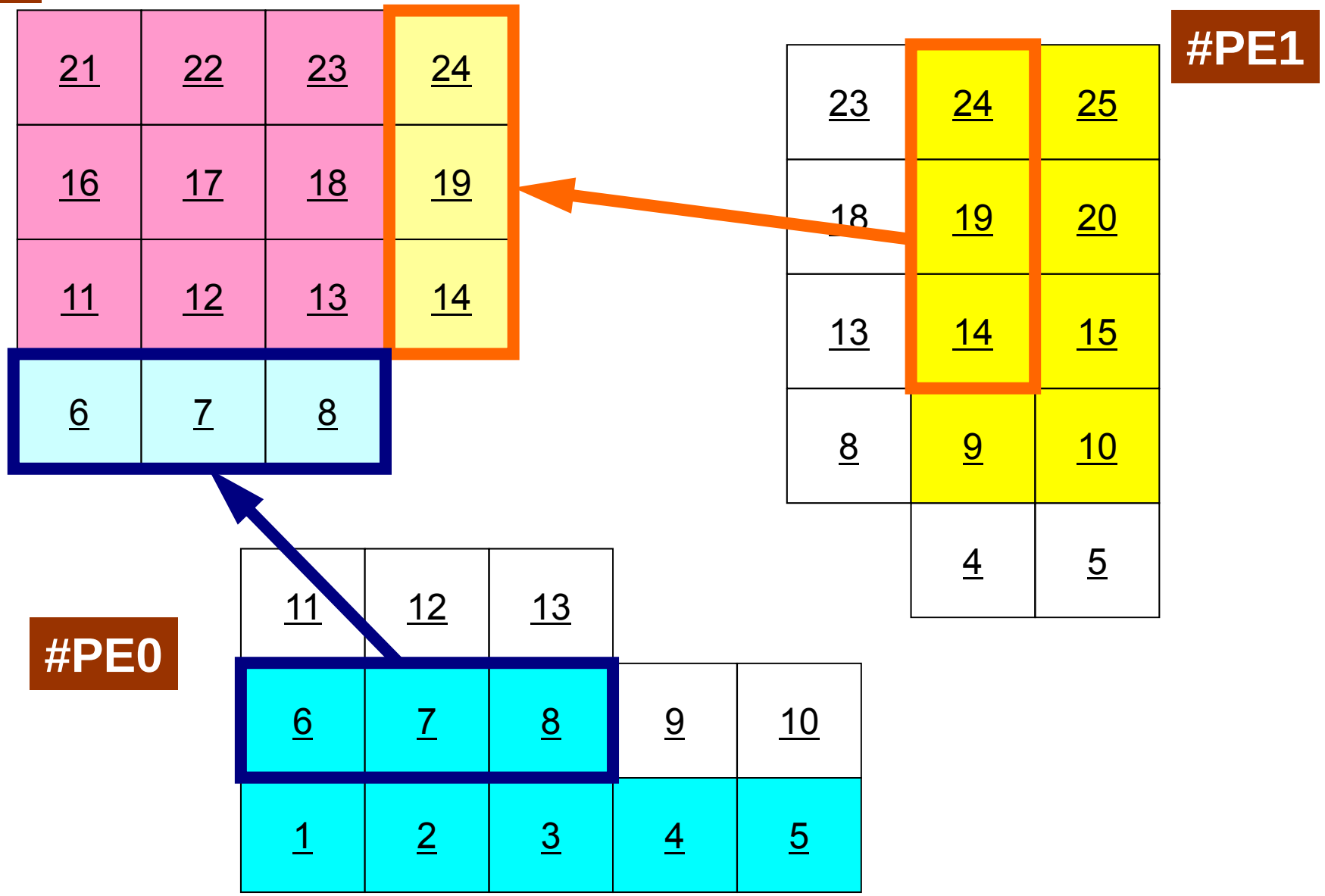
#PE2 オーバーラップ領域のデータ:通信



オーバーラップ領域のデータ:通信



#PE2 オーバーラップ領域のデータ:通信



eps_fvmの処理

```
program eps_fvm
use eps_fvm_all

implicit REAL*8 (A-H,O-Z)

call eps_fvm_input_grid        メッシュ読み込み(#S-GRID)
call poi_gen                   マトリクス生成
call eps_fvm_solver            線形ソルバー
call output_ucd                AVS用結果ファイル書き出し(S-GRID-R-UCD)

end program eps_fvm
```

オーバーラップ付きの局所分散データ構造であれば

- 「マトリクス生成」の部分は容易に並列化可能
  - 幾何学的情報のみで生成できる
- 「線形ソルバー（共役勾配法）」の部分も並列化は可能であろう（通信は伴う）
  - 隣接領域のみとの「一対一」通信

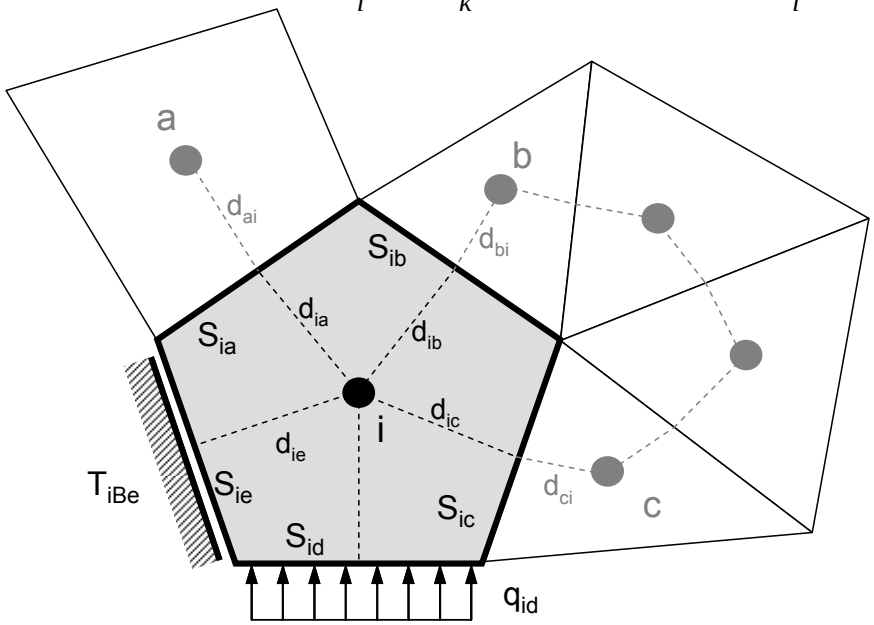
有限体積法（FVM）における処理の特徴

- 計算にあたっては，隣接した要素の情報のみが必要。
- 局所的な処理が可能。
- したがって並列計算には適した手法である。

隣接要素との熱伝導　　温度固定境界

$$\sum_k \frac{S_{ik}}{\frac{d_{ik}}{\lambda_i}+\frac{d_{ki}}{\lambda_k}}\left(T_k - T_i\right) + \sum_e \frac{S_{ie}}{\frac{d_{ie}}{\lambda_i}}\left(T_{iBe} - T_i\right) + \sum_d S_{id}\dot{q}_{id} + V_i\dot{Q}_i = 0$$

要素境界面通過熱流束　　体積発熱



λ　：熱伝導率
V_i　：要素体積
S　：表面面積
d_{ij}　：要素中心から表面までの距離
q　：表面フラックス
Q　：体積発熱
T_{iB}　：境界温度

前処理付き共役勾配法
Preconditioned Conjugate Gradient Method（CG）

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

- 行列ベクトル積
- ベクトル内積
- ベクトル定数倍の加減
- 前処理（対角スケーリング）

$x^{(i)}$ ：ベクトル
α_i ：スカラー

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 $[M]z^{(i-1)}= r^{(i-1)}$
  $\rho_{i-1}= r^{(i-1)}\ z^{(i-1)}$
  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
  $\mathbf{q^{(i)}= [A]p^{(i)}}$
  $\alpha_i\ = \rho_{i-1}/p^{(i)}q^{(i)}$
  $x^{(i)}= x^{(i-1)} + \alpha_i p^{(i)}$
  $r^{(i)}= r^{(i-1)} - \alpha_i q^{(i)}$
  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
- 前処理（対角スケーリング）

```
do i= 1, N
  Y(i)= D(i)*X(i)
  do j= INDEX(i-1)+1, INDEX(i)
    Y(i)= Y(i) + AMAT(j)*X(ITEM(j))
  enddo
enddo
```

右辺の X(ITEM(i))が「オーバーラップ」領域に所属する可能性がある：通信発生

オーバーラップ領域のデータ必要

#PE2

21	22	23	24
16	17	18	19
11	12	13	14
6	7	8	

#PE1

23	24	25
18	19	20
13	14	15
8	9	10
	4	5

#PE0

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

前処理付き共役勾配法
Preconditioned Conjugate Gradient Method（CG）

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

- 行列ベクトル積
- ベクトル内積
- ベクトル定数倍の加減
- 前処理（対角スケーリング）

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

全体領域で内積を求める必要がある：通信発生

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 $[M]z^{(i-1)}= r^{(i-1)}$
  $\rho_{i-1}= r^{(i-1)}\ z^{(i-1)}$
  if i=1
    $p^{(1)}= z^{(0)}$
  else
    $\beta_{i-1}= \rho_{i-1}/\rho_{i-2}$
    $\mathbf{p^{(i)}= z^{(i-1)} + \beta_{i-1}\ p^{(i-1)}}$
  endif
  $q^{(i)}= [A]p^{(i)}$
  $\alpha_i\ = \rho_{i-1}/p^{(i)}q^{(i)}$
  $\mathbf{x^{(i)}= x^{(i-1)} + \alpha_i p^{(i)}}$
  $\mathbf{r^{(i)}= r^{(i-1)} - \alpha_i q^{(i)}}$
  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**
- 前処理（対角スケーリング）

```
do i= 1, N
  z(i)= ALPHA*X(i) + Y(i)
enddo
```

領域内で計算が可能：通信不要

前処理付き共役勾配法
Preconditioned Conjugate Gradient Method（CG）

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

- 行列ベクトル積
- ベクトル内積
- ベクトル定数倍の加減
- 前処理（対角スケーリング）

```
do i= 1, N
  Y(i)=X(i)/D(i)
enddo
```

領域内で計算が可能：通信不要
（前処理手法に依存するが）

前処理付き共役勾配法
Preconditioned Conjugate Gradient Method（CG）

```
Compute r(0)= b-[A]x(0)
for i= 1, 2, …
    solve [M]z(i-1)= r(i-1)
    ρi-1= r(i-1) z(i-1)
    if i=1
      p(1)= z(0)
     else
      βi-1= ρi-1/ρi-2
      p(i)= z(i-1) + βi-1 p(i-1)
    endif
    q(i)= [A]p(i)
    αi  = ρi-1/p(i)q(i)
    x(i)= x(i-1) + αip(i)
    r(i)= r(i-1) - αiq(i)
    check convergence |r|
end
```

並列計算，領域間通信が必要な処理

- 行列ベクトル積
- 内積

共役勾配法の「並列化」(1/2)

- 行列ベクトル積
  - オーバーラップ領域データの交換を実施して，ベクトル値の最新値を得ておく。

```
!C
!C-- {q}= [A]{p}

      exchange W(i,P)

      do i= 1, N
        W(i,Q) = DIAG(i)*W(i,P)
        do j= INDEX(i-1)+1, INDEX(i)
          W(i,Q) = W(i,Q) + AMAT(j)*W(ITEM(j),P)
        enddo
      enddo
```

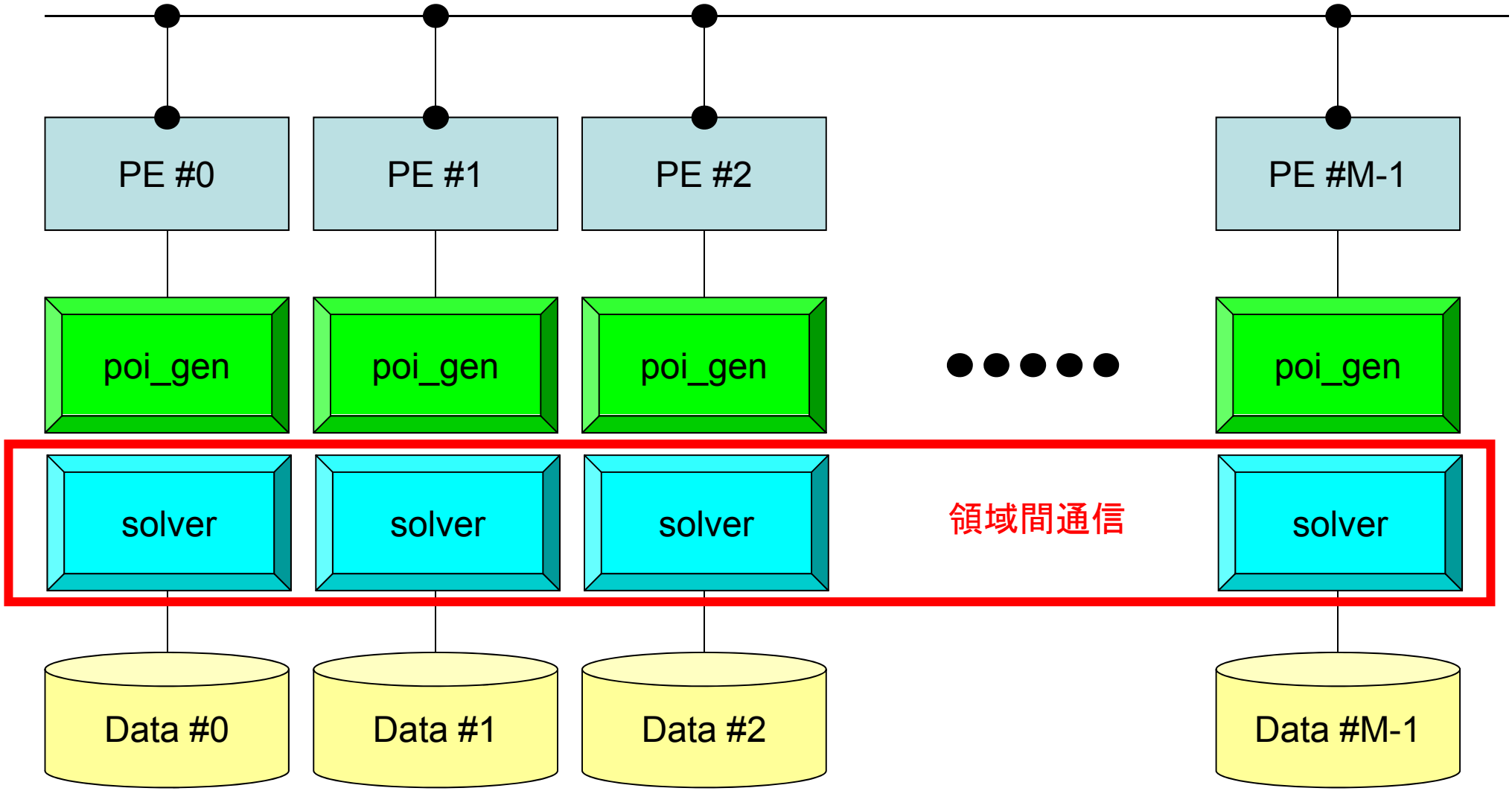
共役勾配法の「並列化」(2/2)

- 内積
  - MPI_ALLREDUCE

```
!C
!C +--------------+
!C | RHO= {r}{z} |
!C +--------------+
!C===
      RHO= 0.d0

      do i= 1, N
        RHO= RHO + W(i,R)*W(i,Z)
      enddo

      allreduce RHO

!C===
```

eps_fvm

オーバーラップ付局所分散データ構造によって SPMDが実現される



局所分散データの概要

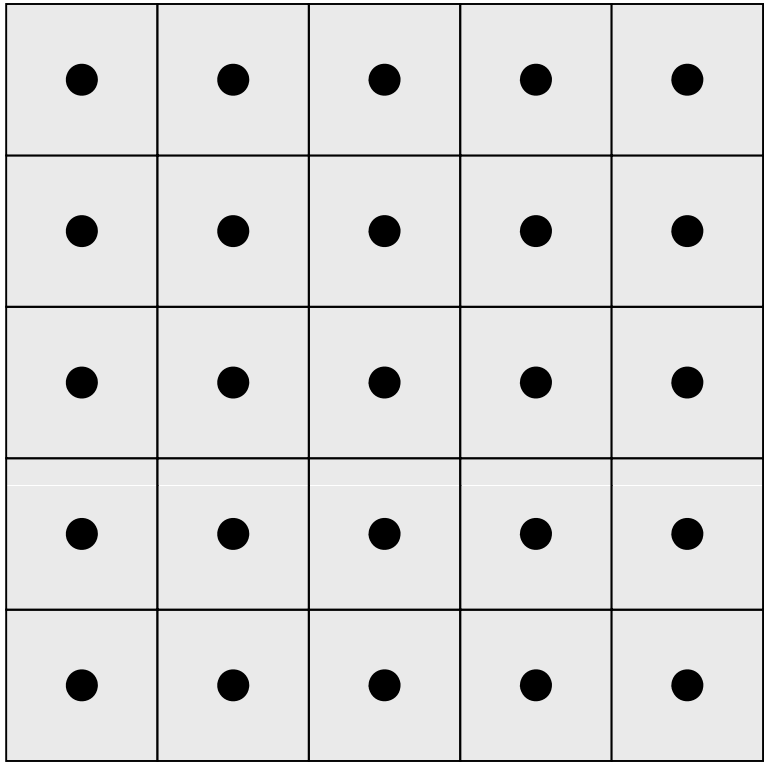
- 内点
- 外点：オーバーラップ領域
- 境界点

- 領域間通信テーブル

有限体積法

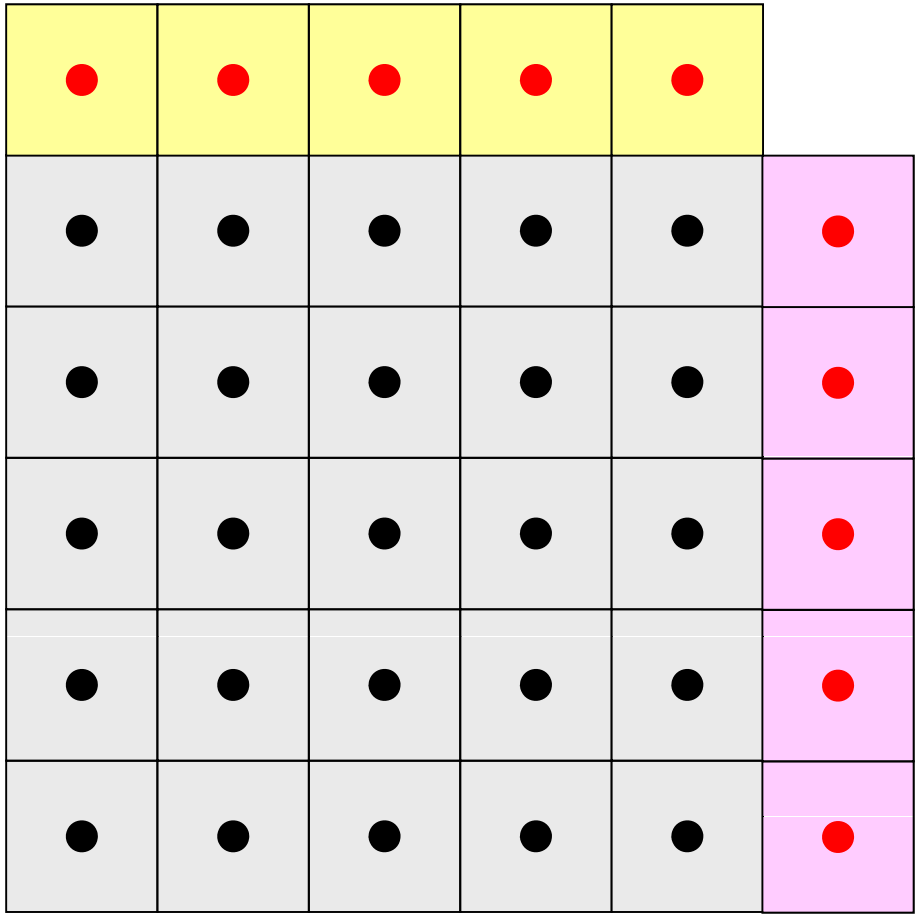
各領域に必要な情報(1/4)

内点(Internal Points)
その領域にアサインされた要素



有限体積法

各領域に必要な情報(2/4)



内点(Internal Points)

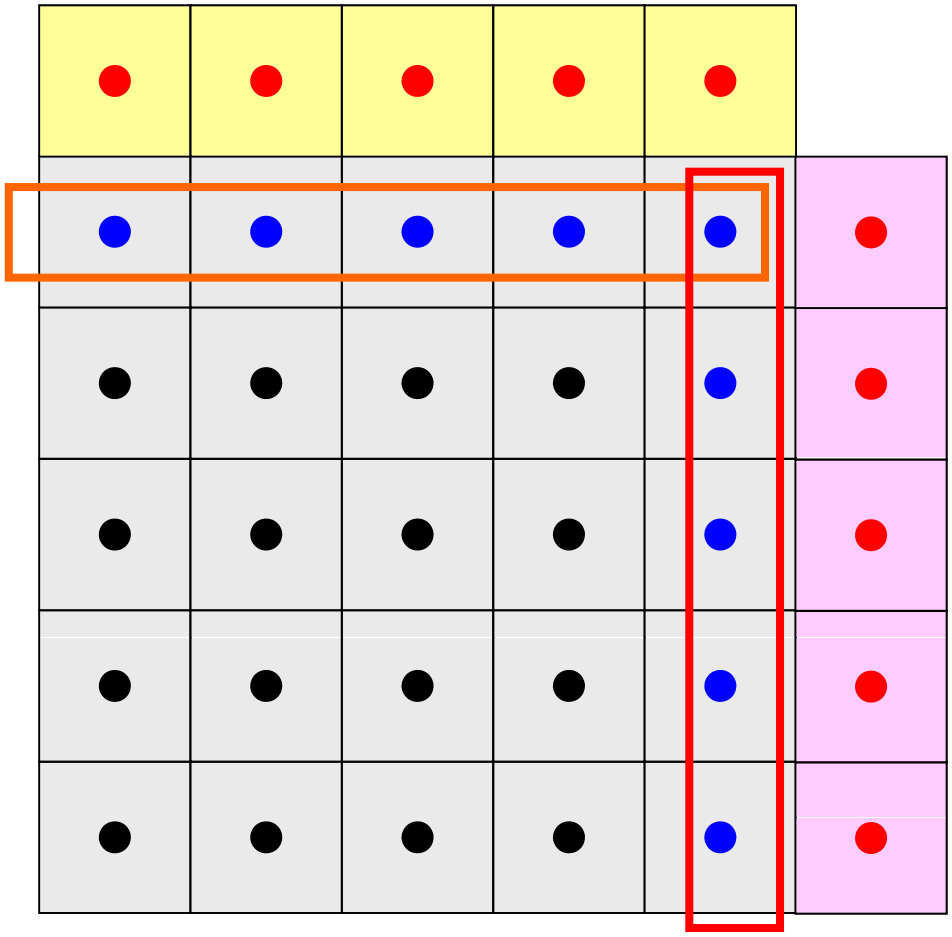
その領域にアサインされた要素

外点(External Points)

他の領域にアサインされた要素であるがその領域の計算を実施するのに必要な要素
(オーバーラップ領域の要素)

・袖領域
・Halo(後光，光輪，(太陽・月の)暈(かさ)，暈輪(うんりん))

有限体積法
各領域に必要な情報(4/4)

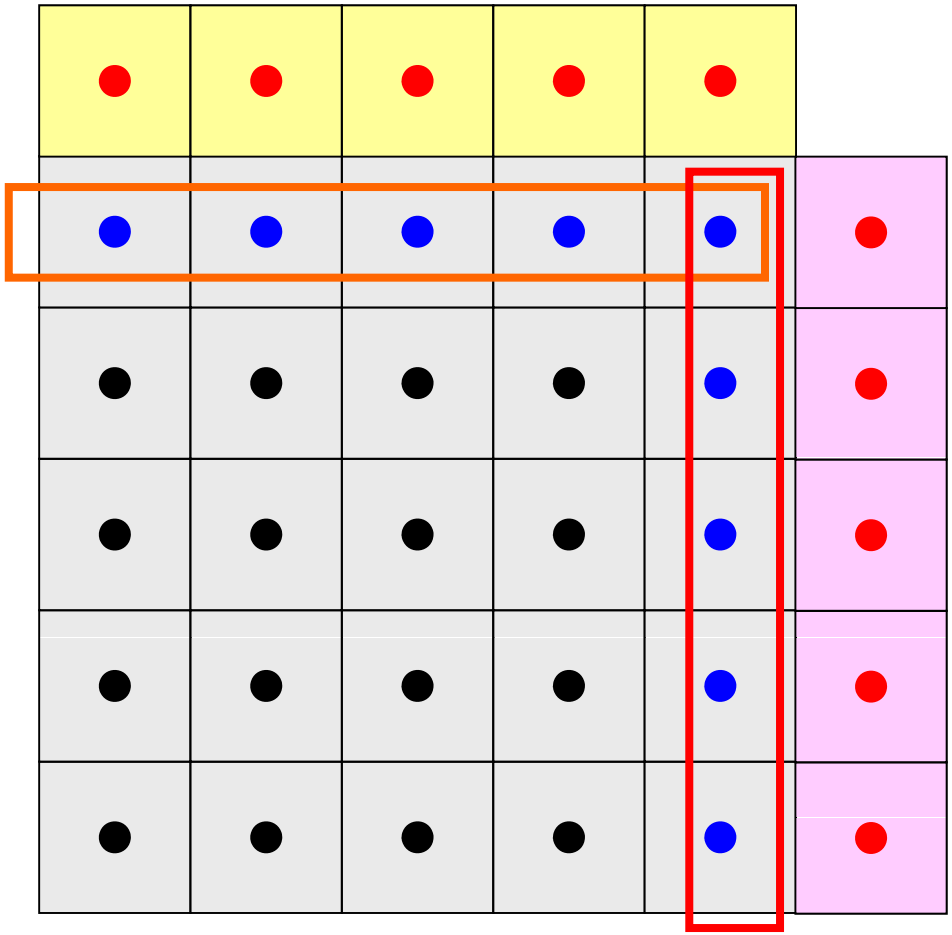


内点(Internal Points)
その領域にアサインされた要素

外点(External Points)
他の領域にアサインされた要素であるがその領域の計算を実施するのに必要な要素
(オーバーラップ領域の要素)

境界点(Boundary Points)
内点のうち，他の領域の外点となっている要素
他の領域の計算に使用される要素

有限体積法
各領域に必要な情報(4/4)



内点(Internal Points)
その領域にアサインされた要素

外点(External Points)
他の領域にアサインされた要素であるがその領域の計算を実施するのに必要な要素
(オーバーラップ領域の要素)

境界点(Boundary Points)
内点のうち，他の領域の外点となっている要素
他の領域の計算に使用される要素

領域間相互の関係
通信テーブル:外点，境界点の関係
隣接領域

各領域データ(局所分散データ)仕様

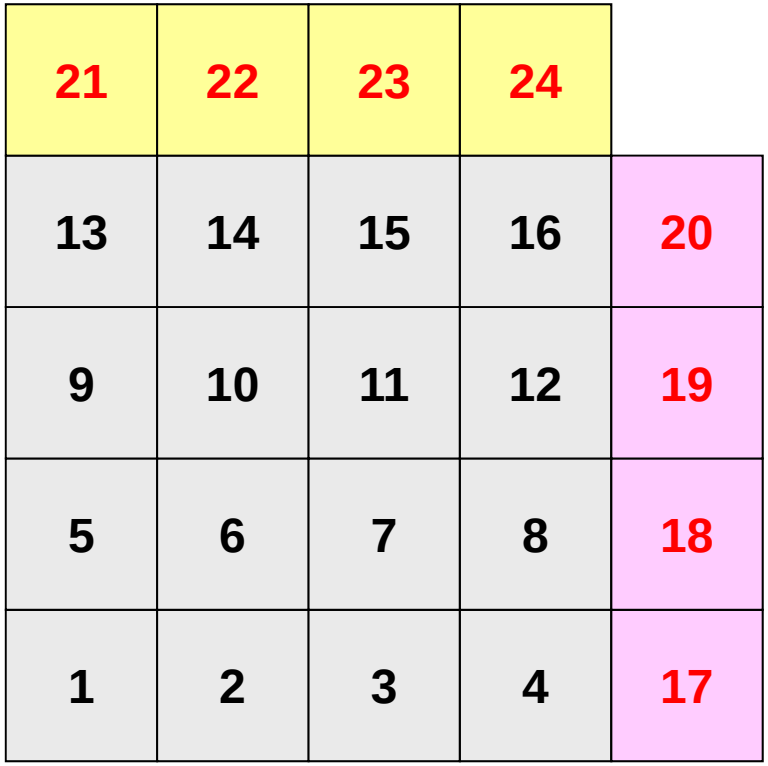
「一般化された通信テーブル」に対応

21	22	23	24	
13	14	15	16	20
9	10	11	12	19
5	6	7	8	18
1	2	3	4	17

- 内点，外点
  - 内点～外点となるように「局所番号」をつける
    - 1番から通し番号
    - SPMD
  - 並列，単体CPUの区別無いプログラムとすることができる
    - 内積のときと同様

各領域データ(局所分散データ)仕様

「一般化された通信テーブル」に対応



- 内点，外点
  - 内点～外点となるように局所番号をつける
- 隣接領域情報
  - オーバーラップ要素を共有する領域
  - 隣接領域数，番号
- 外点情報
  - どの領域から，何個の，どの外点の情報を「受信：import」するか
- 境界点情報
  - 何個の，どの境界点の情報を，どの領域に「送信：export」するか

内点

#PE2

21	22	23	24
16	17	18	19
11	12	13	14
6	7	8	

#PE1

23	24	25
18	19	20
13	14	15
8	9	10
	4	5

#PE0

11	12	13		
6	7	8	9	10
1	2	3	4	5

外点

#PE2

21	22	23	24
16	17	18	19
11	12	13	14
6	7	8	

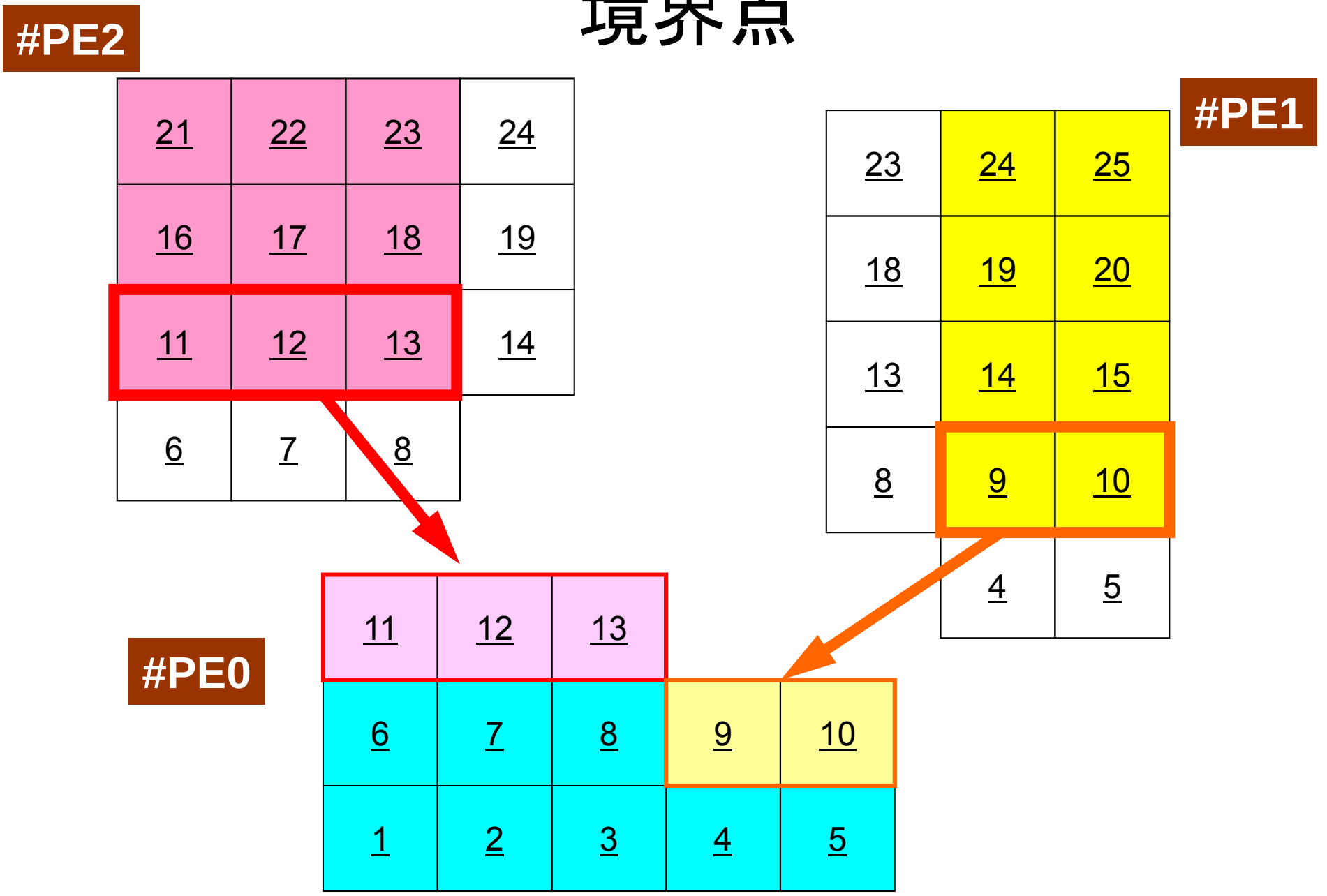
#PE1

23	24	25
18	19	20
13	14	15
8	9	10
	4	5

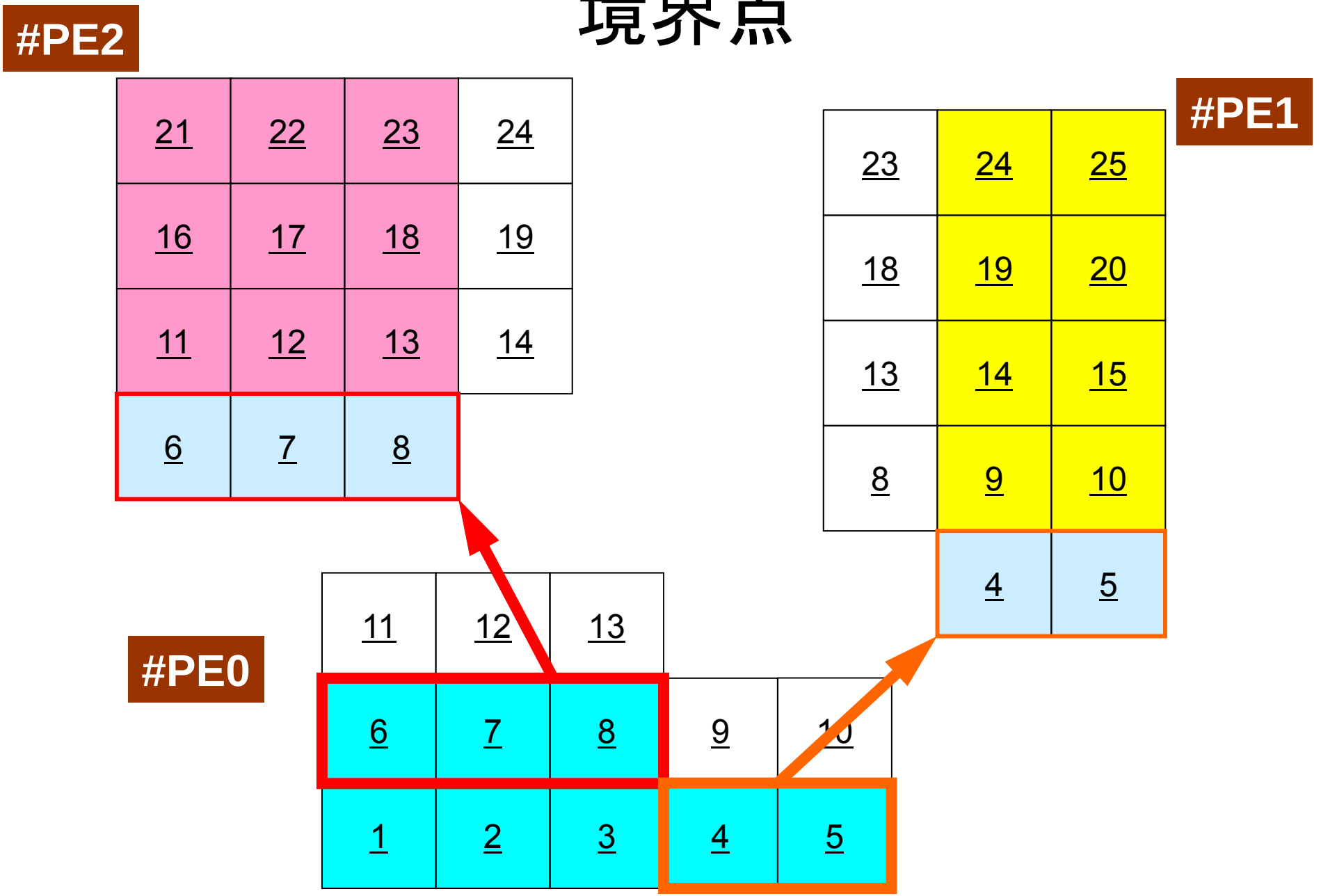
#PE0

11	12	13		
6	7	8	9	10
1	2	3	4	5

境界点


#PE2
21 22 23 24
16 17 18 19
11 12 13 14
6 7 8
#PE1
23 24 25
18 19 20
13 14 15
8 9 10
4 5
#PE0
11 12 13
6 7 8 9 10
1 2 3 4 5

境界点


#PE2
21 22 23 24
16 17 18 19
11 12 13 14
6 7 8
#PE1
23 24 25
18 19 20
13 14 15
8 9 10
4 5
#PE0
11 12 13
6 7 8 9 10
1 2 3 4 5

領域間通信
一般化された通信テーブル

- 「通信」とは「外点」の情報を，その「外点」が本来属している領域から得ることである。

- 「通信テーブル」とは領域間の外点の関係の情報を記述したもの。
  - 「送信テーブル（export）」，「受信テーブル（import）」がある。

- **隣接領域との「一対一」通信を利用できる**
  - **MPI_Isend, MPI_Irecv, MPI_Waitall**
  - **MPI_Sendrecv**

MPI_ISEND

- 送信バッファ「sendbuf」内の，連続した「count」個の送信メッセージを，タグ「tag」を付けて，コミュニケータ内の，「dest」に送信する。「MPI_WAITALL」を呼ぶまで，送信バッファの内容を更新してはならない。

- **call MPI_ISEND (sendbuf,count,datatype,dest,tag,comm,request, ierr)**
  - **sendbuf** 任意 I 送信バッファの先頭アドレス，
  - **count** 整数 I メッセージのサイズ
  - **datatype** 整数 I メッセージのデータタイプ
  - **dest** 整数 I 宛先プロセスのアドレス（ランク）
  - **tag** 整数 I メッセージタグ，送信メッセージの種類を区別するときに使用。通常は「0」でよい。同じメッセージタグ番号同士で通信。
  - **comm** 整数 I コミュニケータを指定する
  - **request** 整数 O 通信識別子。MPI_WAITALLで使用。（配列：サイズは同期する必要のある「MPI_ISEND」呼び出し数（通常は隣接プロセス数など））
  - **ierr** 整数 O 完了コード

MPI_IRECV

- 受信バッファ「recvbuf」内の，連続した「count」個の送信メッセージを，タグ「tag」を付けて，コミュニケータ内の，「dest」から受信する。「MPI_WAITALL」を呼ぶまで，受信バッファの内容を利用した処理を実施してはならない。

- **call MPI_IRECV (recvbuf,count,datatype,dest,tag,comm,request, ierr)**
  - **recvbuf** 任意 I 受信バッファの先頭アドレス，
  - **count** 整数 I メッセージのサイズ
  - **datatype** 整数 I メッセージのデータタイプ
  - **dest** 整数 I 宛先プロセスのアドレス（ランク）
  - **tag** 整数 I メッセージタグ，受信メッセージの種類を区別するときに使用。通常は「0」でよい。同じメッセージタグ番号同士で通信。
  - **comm** 整数 I コミュニケータを指定する
  - **request** 整数 O 通信識別子。MPI_WAITALLで使用。（配列：サイズは同期する必要のある「MPI_IRECV」呼び出し数（通常は隣接プロセス数など））
  - **ierr** 整数 O 完了コード

MPI_SENDRECV

- MPI_SEND+MPI_RECV

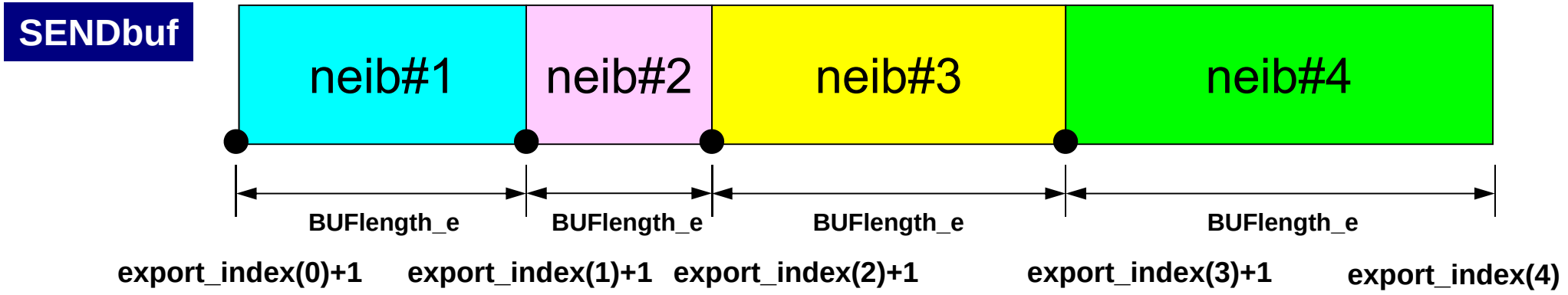
- **call MPI_SENDRECV**
  (sendbuf,sendcount,sendtype,dest,sendtag,recvbuf,
  recvcount,recvtype,source,recvtag,comm,status,ierr)
  - **sendbuf** 任意 I 送信バッファの先頭アドレス，
  - **sendcount** 整数 I 送信メッセージのサイズ
  - **sendtype** 整数 I 送信メッセージのデータタイプ
  - **dest** 整数 I 宛先プロセスのアドレス（ランク）
  - **sendtag** 整数 I 送信用メッセージタグ，送信メッセージの種類を区別するときに使用。
    通常は「0」でよい。
  - **recvbuf** 任意 I 受信バッファの先頭アドレス，
  - **recvcount** 整数 I 受信メッセージのサイズ
  - **recvtype** 整数 I 受信メッセージのデータタイプ
  - **source** 整数 I 送信元プロセスのアドレス（ランク）
  - **sendtag** 整数 I 受信用メッセージタグ，送信メッセージの種類を区別するときに使用。
    通常は「0」でよい。同じメッセージタグ番号同士で通信。
  - **comm** 整数 I コミュニケータを指定する
  - **status** 整数 O 状況オブジェクト配列（配列サイズ：（MPI_STATUS_SIZE））
    MPI_STATUS_SIZE：“mpif.h”で定められるパラメータ
  - **ierr** 整数 O 完了コード

一般化された通信テーブル：送信

- 送信相手
  - NEIBPETOT，NEIB(neib)
- それぞれの送信相手に送るメッセージサイズ（累積）
  - export_index(neib), neib= 1, NEIBPETOT
- 「境界点」番号
  - export_item(k), k= 1, export_index(NEIBPETOT)
- それぞれの送信相手に送るメッセージ
  - SENDbuf(k), k= 1, export_index(NEIBPETOT)

- CRS（行列格納法）のindex~itemの関係

送信（MPI_Isend/Irecv/Waitall）

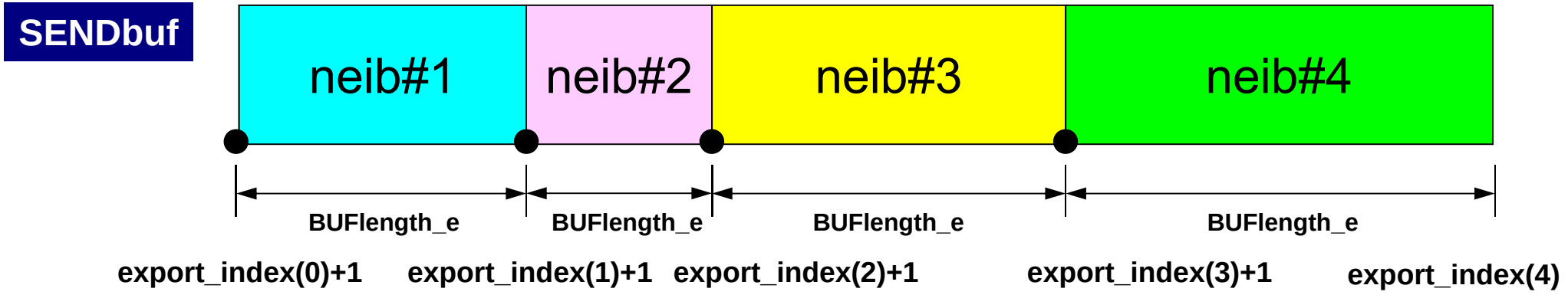


export_index(neib-1)+1～export_index(neib)番目のexport_itemがneib番目の隣接領域に送信される

```
 do neib= 1, NEIBPETOT
   do k= export_index(neib-1)+1, export_index(neib)
     kk= export_item(k)
     SENDbuf(k)= VAL(kk)
   enddo
 enddo

 do neib= 1, NEIBPETOT
   iS_e= export_index(neib-1) + 1
   iE_e= export_index(neib  )
   BUFlength_e= iE_e + 1 - iS_e
   call MPI_Isend                                                  &
&         (SENDbuf(iS_e), BUFlength_e, MPI_INTEGER, NEIBPE(neib), 0,&
&          MPI_COMM_WORLD, request_send(neib), ierr)
 enddo

 call MPI_Waitall (NEIBPETOT, request_send, stat_recv, ierr)
```

送信バッファへの代入
温度などの変数を直接送信，受信に使うのではなく，このようなバッファへ一回代入して計算する

送信（MPI_Sendrecv）



export_index(neib-1)+1～export_index(neib)番目のexport_itemがneib番目の隣接領域に送信される

```
do neib= 1, NEIBPETOT
  do k= export_index(neib-1)+1, export_index(neib)
    kk= export_item(k)
    SENDbuf(k)= VAL(kk)
  enddo
enddo

do neib= 1, NEIBPETOT
  iS_e= export_index(neib-1) + 1
  iE_e= export_index(neib  )
  BUFlength_e= iE_e + 1 - iS_e

  call MPI_Sendrecv                                                  &
&         (SENDbuf(iS_e), BUFlength_e, MPI_INTEGER, NEIBPE(neib), 0,&
&          RECVbuf(iS_i), BUFlength_i, MPI_INTEGER, NEIBPE(neib), 0,&
&          MPI_COMM_WORLD, stat_sr, ierr)
  enddo
```

送信バッファへの代入

一般化された通信テーブル：受信

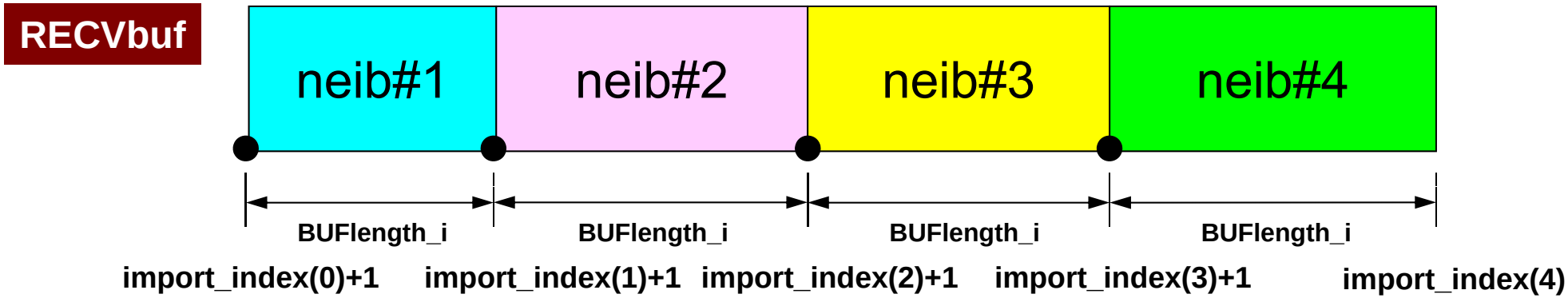
- 受信相手
  - NEIBPETOT，NEIB(neib)
- それぞれの受信相手から受け取るメッセージサイズ
  - import_index(neib), neib=1, NEIBPETOT
- 「外点」番号
  - import_item(k), k= 1, import_index(NEIBPETOT)
- それぞれの受信相手から受け取るメッセージ
  - RECVbuf(k), k= 1, import_index(NEIBPETOT)

受信（MPI_Isend/Irecv/Waitall）

```
 do neib= 1, NEIBPETOT
   iS_i= import_index(neib-1) + 1
   iE_i= import_index(neib  )
   BUFlength_i= iE_i + 1 - iS_i

   call MPI_Irecv                                                  &
&         (RECVbuf(iS_i), BUFlength_i, MPI_INTEGER, NEIBPE(neib), 0,&
&          MPI_COMM_WORLD, request_recv(neib), ierr)
  enddo
  call MPI_Waitall (NEIBPETOT, request_recv, stat_recv, ierr)

  do neib= 1, NEIBPETOT
     do k= import_index(neib-1)+1, import_index(neib)
       kk= import_item(k)
       VAL(kk)= RECVbuf(k)
     enddo
  enddo
```

受信バッファから代入

import_index(neib-1)+1～import_index(neib)番目のimport_itemがneib番目の隣接領域から受信される

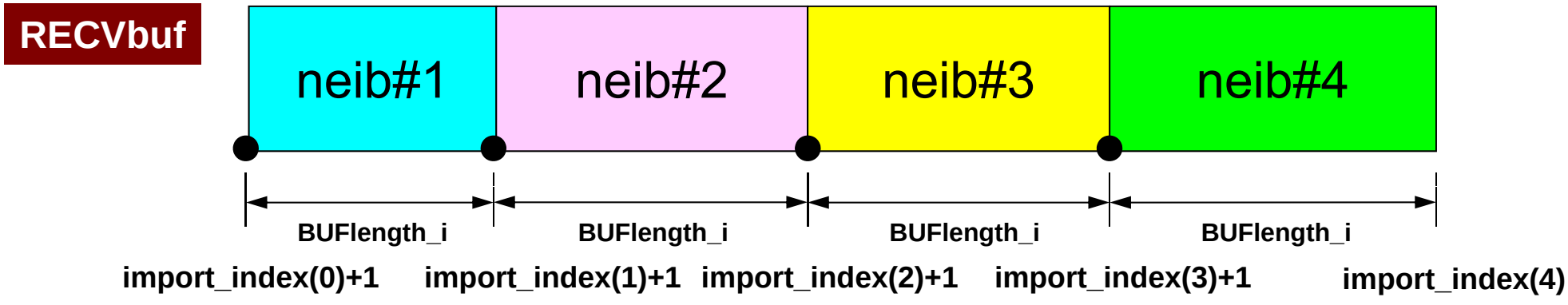


受信（MPI_Sendrecv）

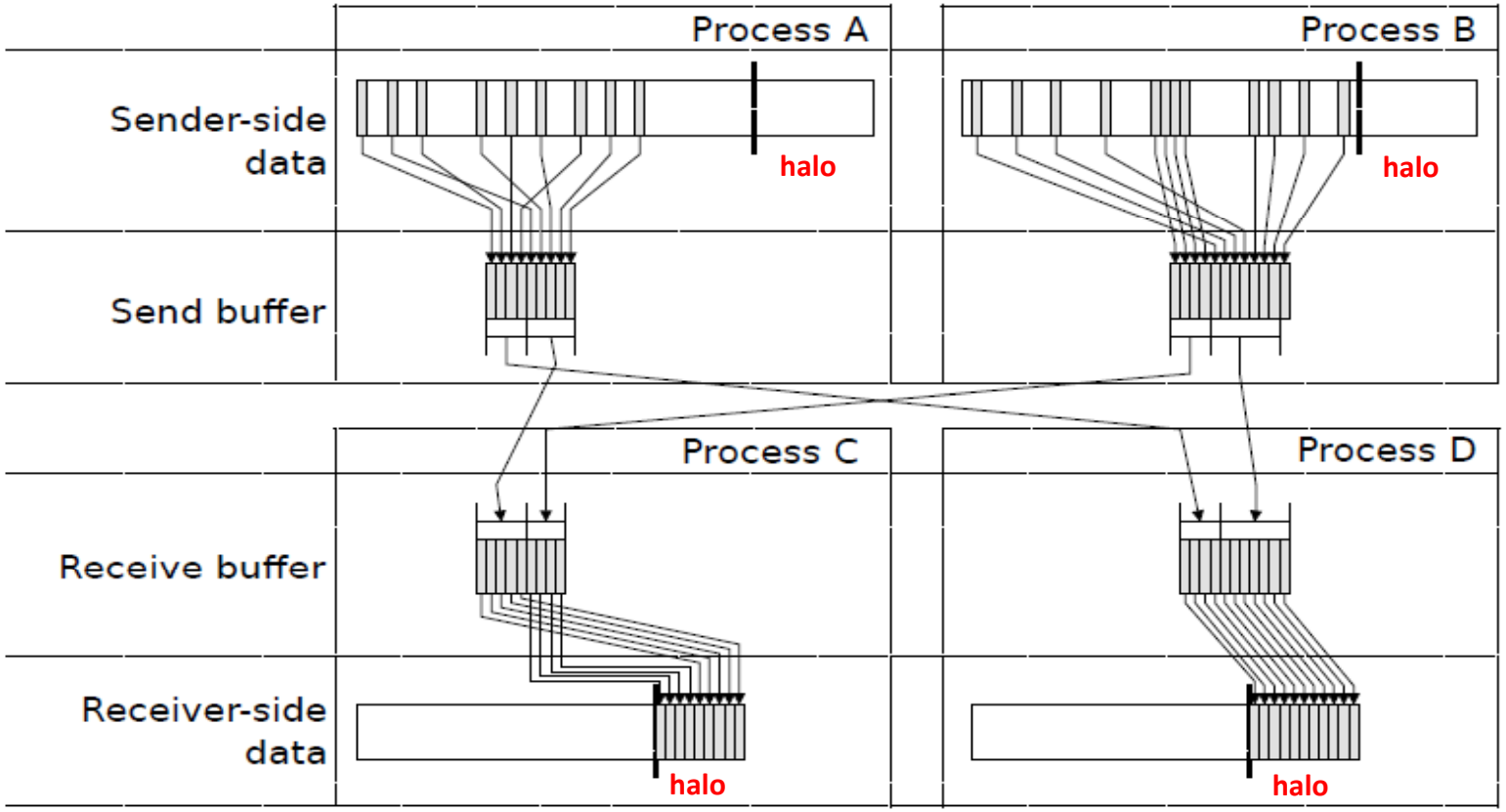
```
 do neib= 1, NEIBPETOT
   iS_i= import_index(neib-1) + 1
   iE_i= import_index(neib  )
   BUFlength_i= iE_i + 1 - iS_i

   call MPI_Sendrecv                                                  &
&         (SENDbuf(iS_e), BUFlength_e, MPI_INTEGER, NEIBPE(neib), 0,&
&          RECVbuf(iS_i), BUFlength_i, MPI_INTEGER, NEIBPE(neib), 0,&
&          MPI_COMM_WORLD, stat_sr, ierr)
  enddo

  do neib= 1, NEIBPETOT
    do k= import_index(neib-1)+1, import_index(neib)
      kk= import_item(k)
      VAL(kk)= RECVbuf(k)
    enddo
  enddo
```

受信バッファからの代入

import_index(neib-1)+1～import_index(neib)番目のimport_itemがneib番目の隣接領域から受信される



Communication Pattern using 1D Structure



Dr. Osni Marques
（Lawrence Berkeley National Laboratory）より借用

サンプルプログラム：二次元データの例 FORTRAN

```
$ cd <$T-EPS>
$ cp /home/t00000/EPSsummer/F/p1b-f.tar .

$ tar xvf p1b-f.tar
$ cd P1

$ cd local/4pe

$ mpif90 –Oss -noparallel sq-sr1.f  Isend/Irecv
$ mpif90 –Oss -noparallel sq-sr2.f  Sendrecv

(go.shを修正)
$ qsub go.sh 実行4プロセス
```

サンプルプログラム：二次元データの例 C

```
$ cd <$T-EPS>
$ cp /home/t00000/EPSsummer/c/p1b-c.tar .

$ tar xvf p1b-c.tar
$ cd P1

$ cd local/4pe

$ mpicc –Os -noparallel sq-sr1.c    Isend/Irecv
$ mpicc –Os -noparallel sq-sr2.c    Sendrecv

(go.shを修正)
$ qsub go.sh 実行4プロセス
```

問題設定：全体データ

57	58	59	60	61	62	63	64
49	50	51	52	53	54	55	56
41	42	43	44	45	46	47	48
33	34	35	36	37	38	39	40
25	26	27	28	29	30	31	32
17	18	19	20	21	22	23	24
9	10	11	12	13	14	15	16
1	2	3	4	5	6	7	8

- 8×8=64要素に分割された二次元領域を考える。
- 各要素には1～64までの全体要素番号が振られている。
  - 簡単のため，この「全体要素番号」を各要素における従属変数値（温度のようなもの）とする
  - ⇒「計算結果」のようなもの

問題設定：局所分散データ

PE#2

57	58	59	60
49	50	51	52
41	42	43	44
33	34	35	36

PE#3

61	62	63	64
53	54	55	56
45	46	47	48
37	38	39	40

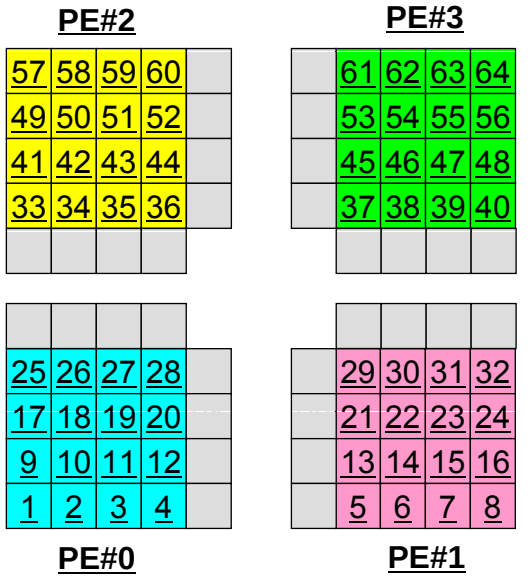
25	26	27	28
17	18	19	20
9	10	11	12
1	2	3	4

PE#0

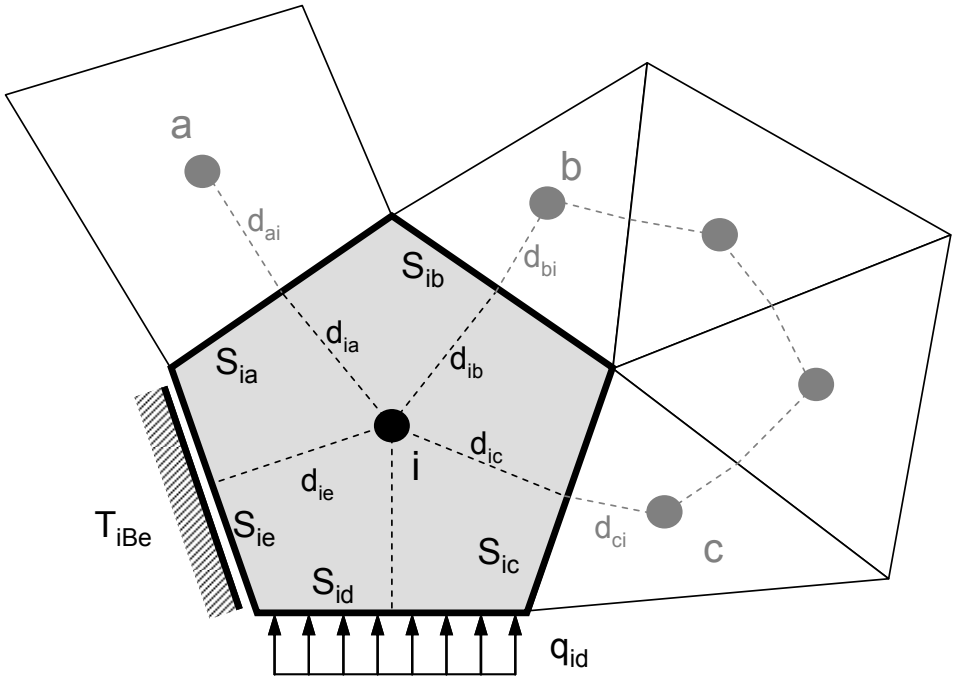
29	30	31	32
21	22	23	24
13	14	15	16
5	6	7	8

PE#1

- 左記のような4領域に分割された二次元領域において，外点の情報（全体要素番号）を隣接領域から受信する方法
  - □はPE#0が受信する情報

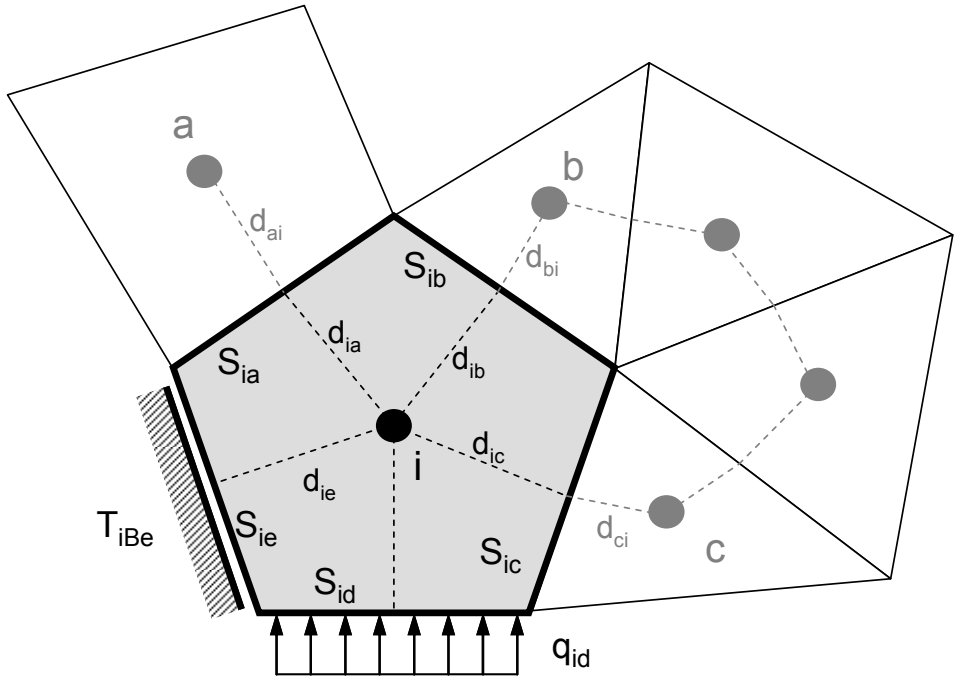


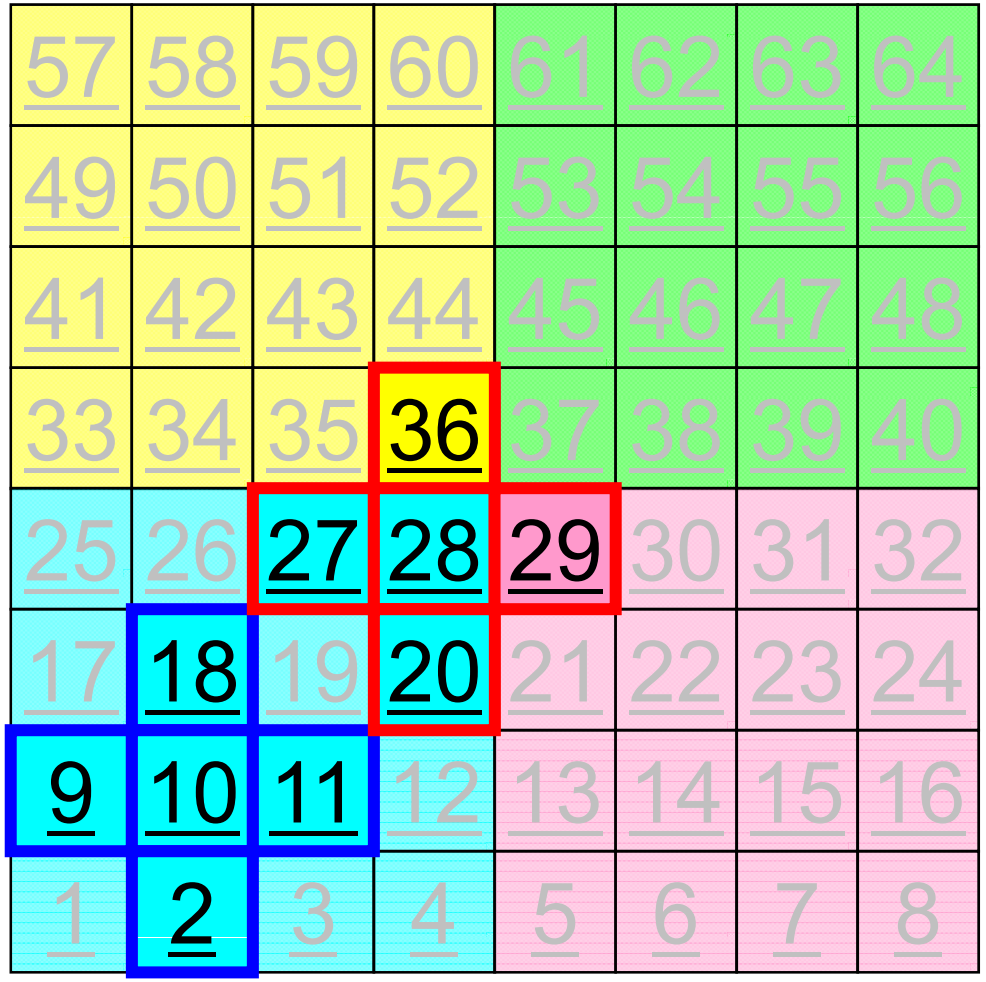
有限体積法のオペレーション



57	58	59	60	61	62	63	64
49	50	51	52	53	54	55	56
41	42	43	44	45	46	47	48
33	34	35	36	37	38	39	40
25	26	27	28	29	30	31	32
17	18	19	20	21	22	23	24
9	10	11	12	13	14	15	16
1	2	3	4	5	6	7	8

有限体積法のオペレーション





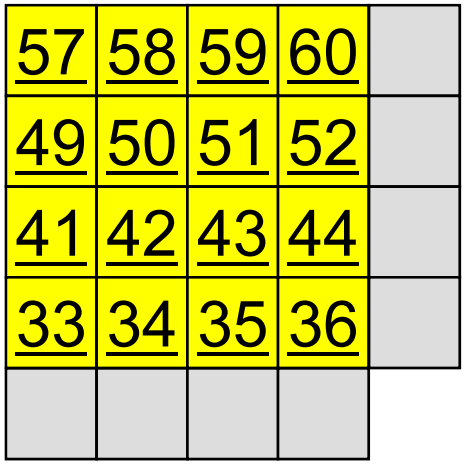
演算内容(1/3)

- 各PEの内点(i=1~N(=16))において局所データを読み込み，「境界点」のデータを各隣接領域における「外点」として配信

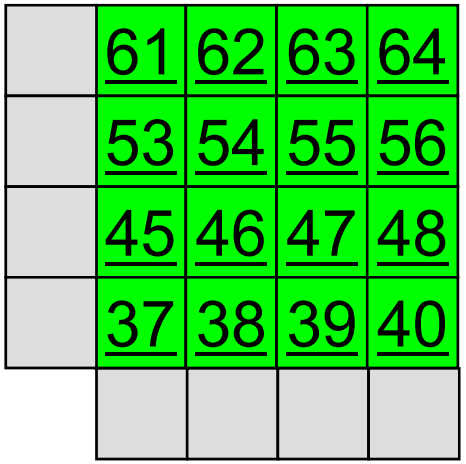
演算内容(2/3):送信，受信前

PE#2

1: 33	9: 49	17: ?
2: 34	10: 50	18: ?
3: 35	11: 51	19: ?
4: 36	12: 52	20: ?
5: 41	13: 57	21: ?
6: 42	14: 58	22: ?
7: 43	15: 59	23: ?
8: 44	16: 60	24: ?

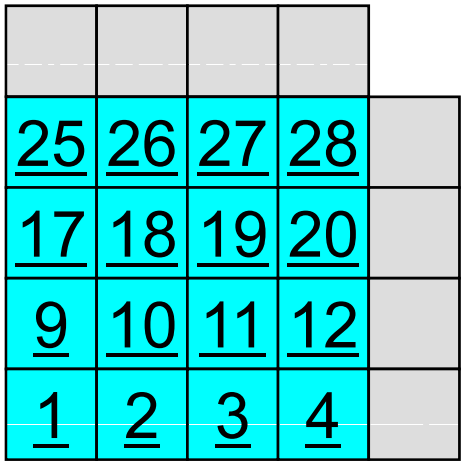


PE#3

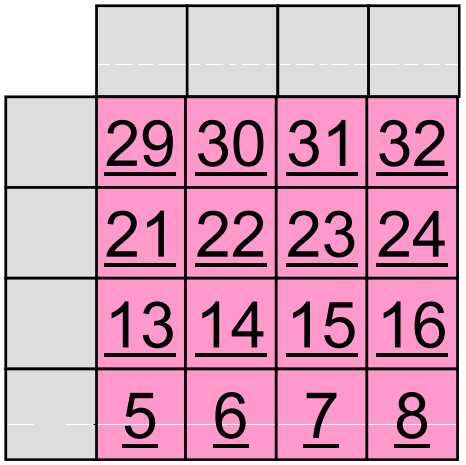


1: 37	9: 53	17: ?
2: 38	10: 54	18: ?
3: 39	11: 55	19: ?
4: 40	12: 56	20: ?
5: 45	13: 61	21: ?
6: 46	14: 62	22: ?
7: 47	15: 63	23: ?
8: 48	16: 64	24: ?

1: 1	9: 17	17: ?
2: 2	10: 18	18: ?
3: 3	11: 19	19: ?
4: 4	12: 20	20: ?
5: 9	13: 25	21: ?
6: 10	14: 26	22: ?
7: 11	15: 27	23: ?
8: 12	16: 28	24: ?



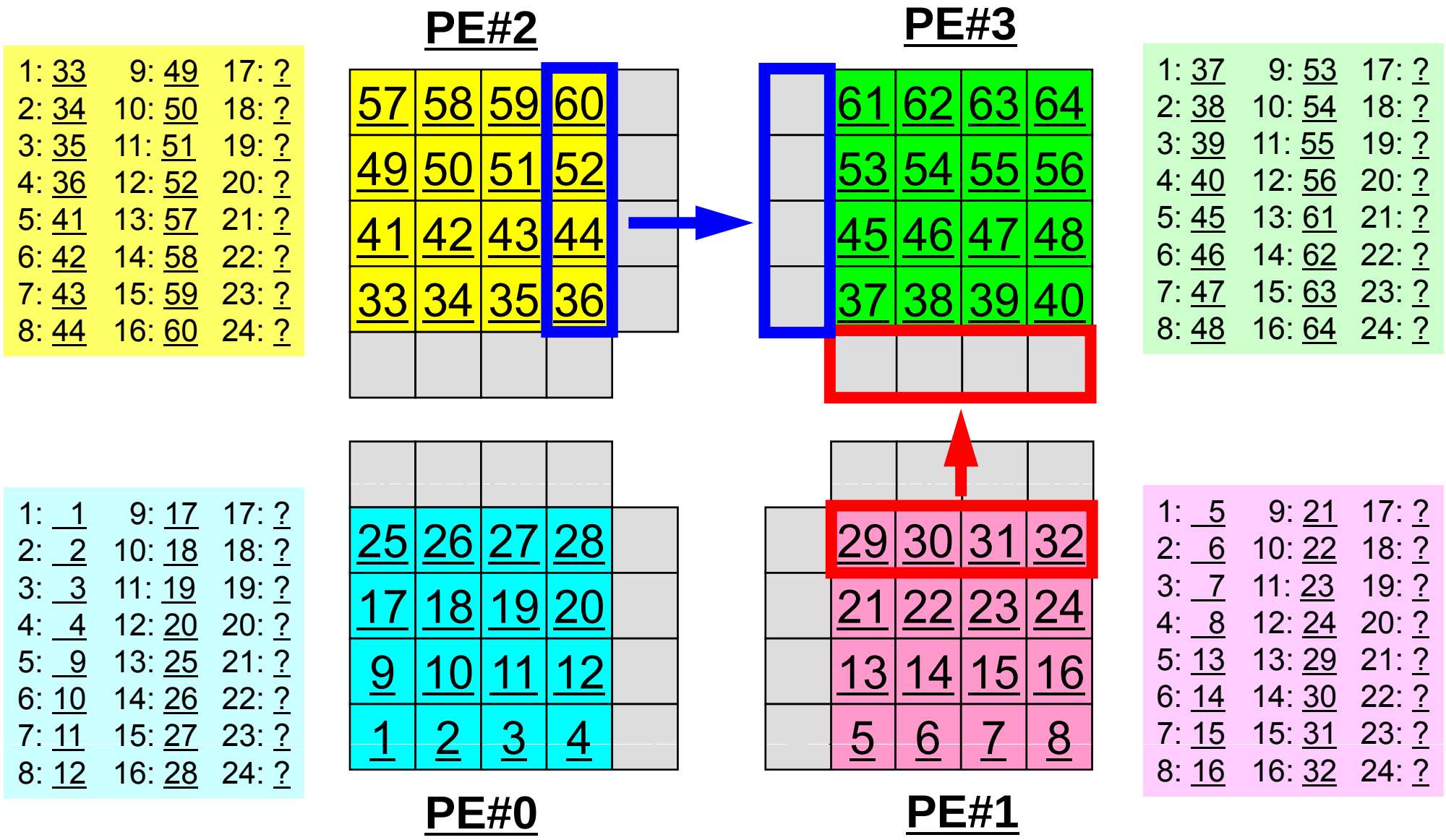
PE#0



PE#1

1: 5	9: 21	17: ?
2: 6	10: 22	18: ?
3: 7	11: 23	19: ?
4: 8	12: 24	20: ?
5: 13	13: 29	21: ?
6: 14	14: 30	22: ?
7: 15	15: 31	23: ?
8: 16	16: 32	24: ?

演算内容(2/3):送信，受信前



演算内容(3/3):送信，受信後

1: 33	9: 49	17: 37
2: 34	10: 50	18: 45
3: 35	11: 51	19: 53
4: 36	12: 52	20: 61
5: 41	13: 57	21: 25
6: 42	14: 58	22: 26
7: 43	15: 59	23: 27
8: 44	16: 60	24: 28

PE#2

57	58	59	60	61
49	50	51	52	53
41	42	43	44	45
33	34	35	36	37
25	26	27	28	

PE#3

60	61	62	63	64
52	53	54	55	56
44	45	46	47	48
36	37	38	39	40
	29	30	31	32

1: 37	9: 53	17: 36
2: 38	10: 54	18: 44
3: 39	11: 55	19: 52
4: 40	12: 56	20: 60
5: 45	13: 61	21: 29
6: 46	14: 62	22: 30
7: 47	15: 63	23: 31
8: 48	16: 64	24: 32

1: 1	9: 17	17: 5
2: 2	10: 18	18: 14
3: 3	11: 19	19: 21
4: 4	12: 20	20: 29
5: 9	13: 25	21: 33
6: 10	14: 26	22: 34
7: 11	15: 27	23: 35
8: 12	16: 28	24: 36

33	34	35	36	
25	26	27	28	29
17	18	19	20	21
9	10	11	12	13
1	2	3	4	5

PE#0

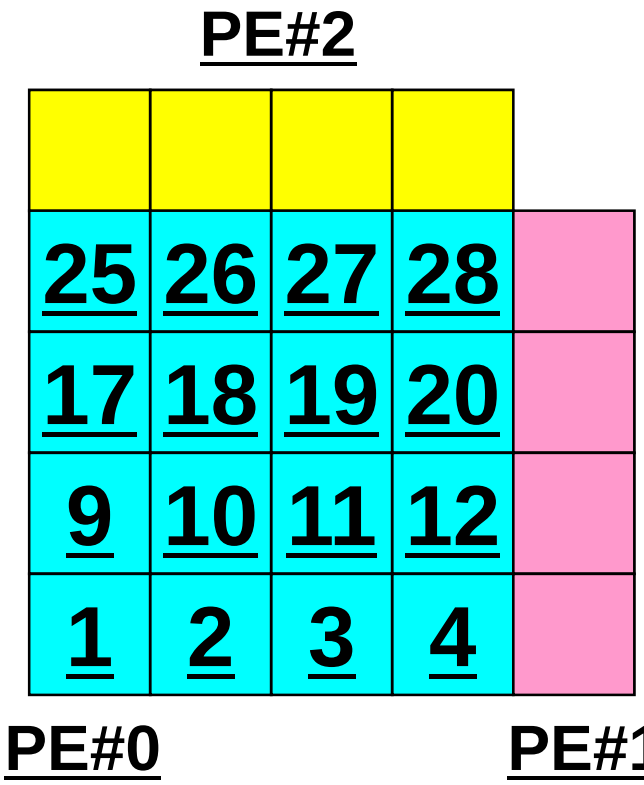
	37	38	39	40
28	29	30	31	32
20	21	22	23	24
12	13	14	15	16
4	5	6	7	8

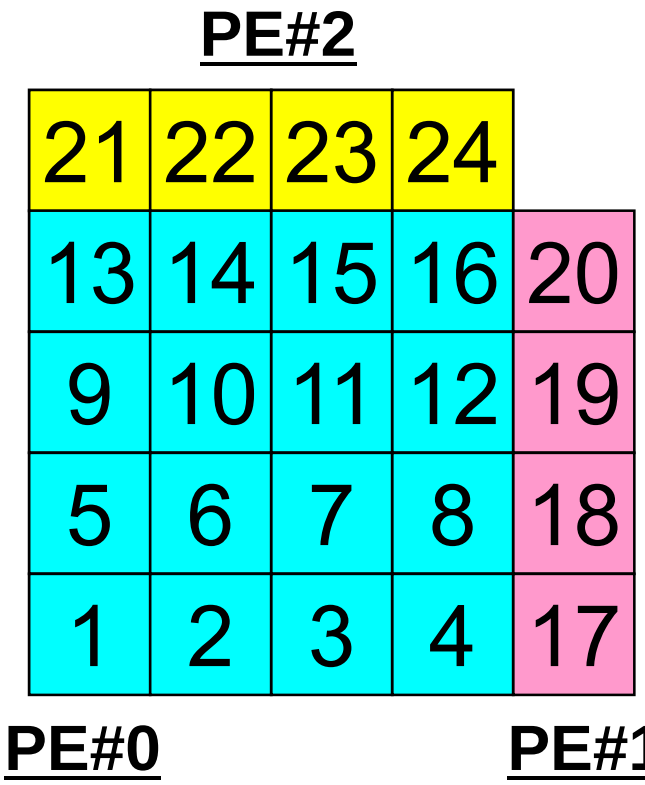
PE#1

1: 5	9: 21	17: 4
2: 6	10: 22	18: 12
3: 7	11: 23	19: 20
4: 8	12: 24	20: 28
5: 13	13: 29	21: 37
6: 14	14: 30	22: 38
7: 15	15: 31	23: 39
8: 16	16: 32	24: 40

各領域データ（局所分散データ）仕様

PE#0における局所分散データ

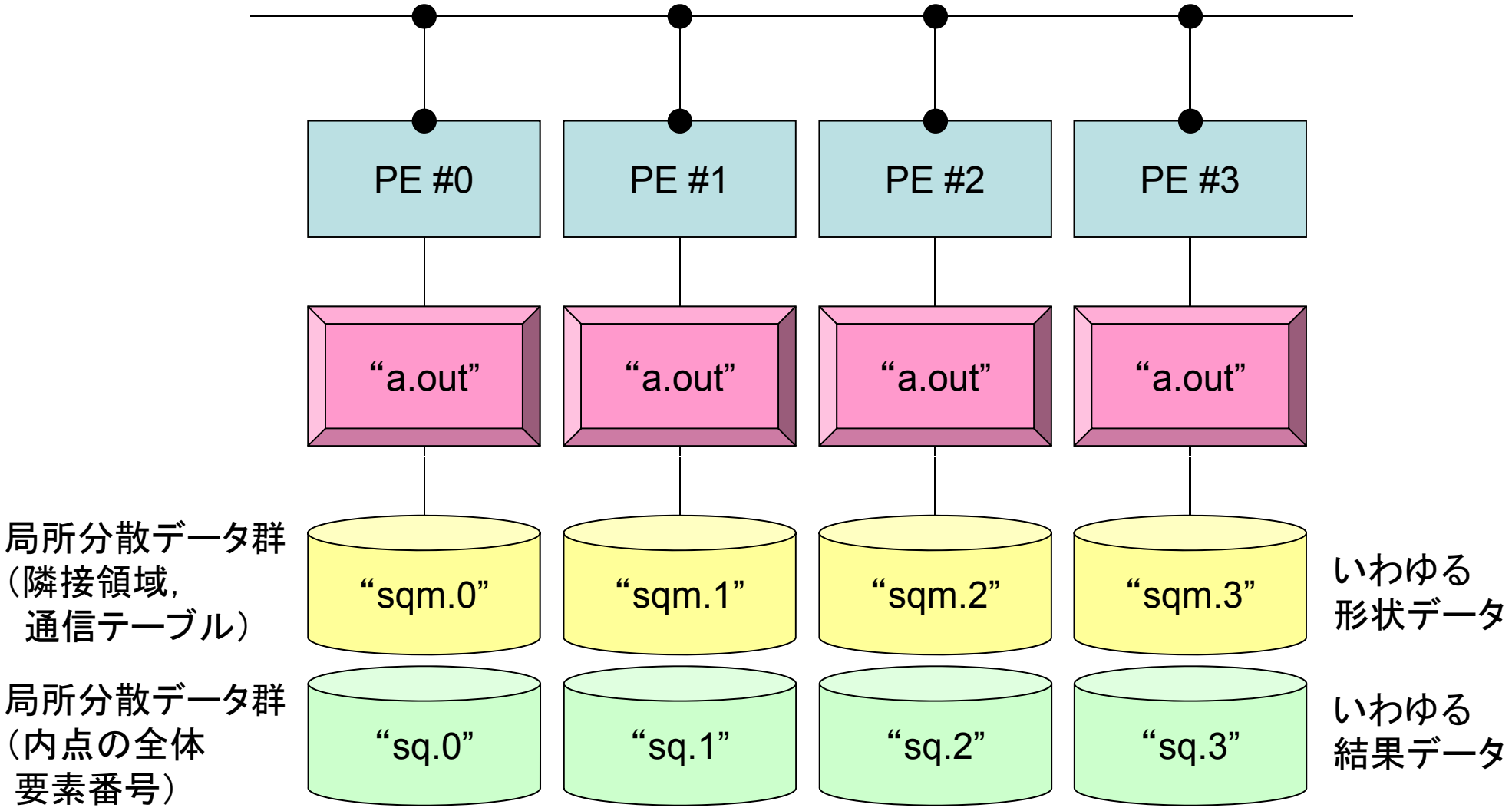




各要素における値（全体番号）

局所番号

SPMD・・・

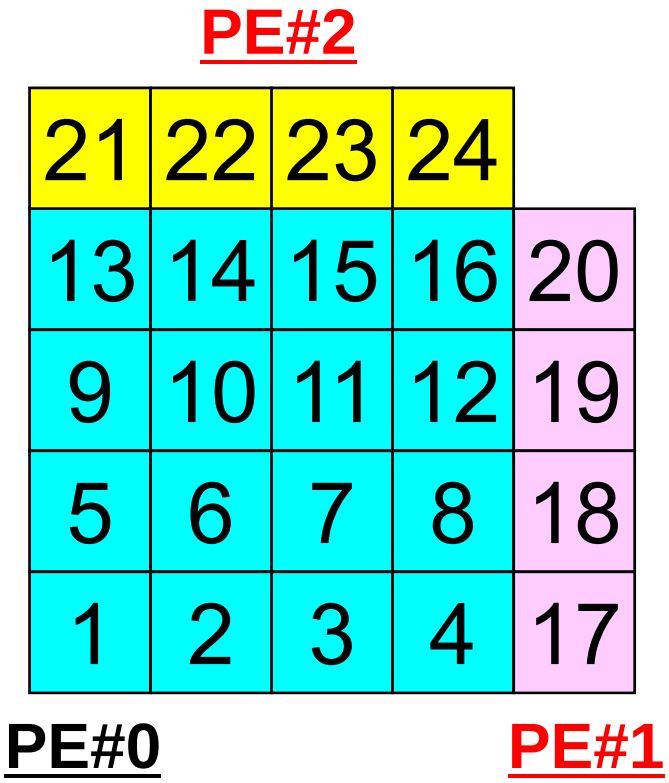

PE #0
PE #1
PE #2
PE #3
“a.out”
“a.out”
“a.out”
“a.out”
局所分散データ群
（隣接領域，
通信テーブル）
“sqm.0”
“sqm.1”
“sqm.2”
“sqm.3”
いわゆる
形状データ
局所分散データ群
（内点の全体
要素番号）
“sq.0”
“sq.1”
“sq.2”
“sq.3”
いわゆる
結果データ

PE#0における局所分散データ（1/8）: sqm.0

```
#NEIBPEtot
2
#NEIBPE
1  2
#NODE
24 16
#IMPORTindex
4  8
#IMPORTitems
17
18
19
20
21
22
23
24
#EXPORTindex
4  8
#EXPORTitems
4
8
12
16
13
14
15
16
```

PE#0における局所分散データ(2/8)

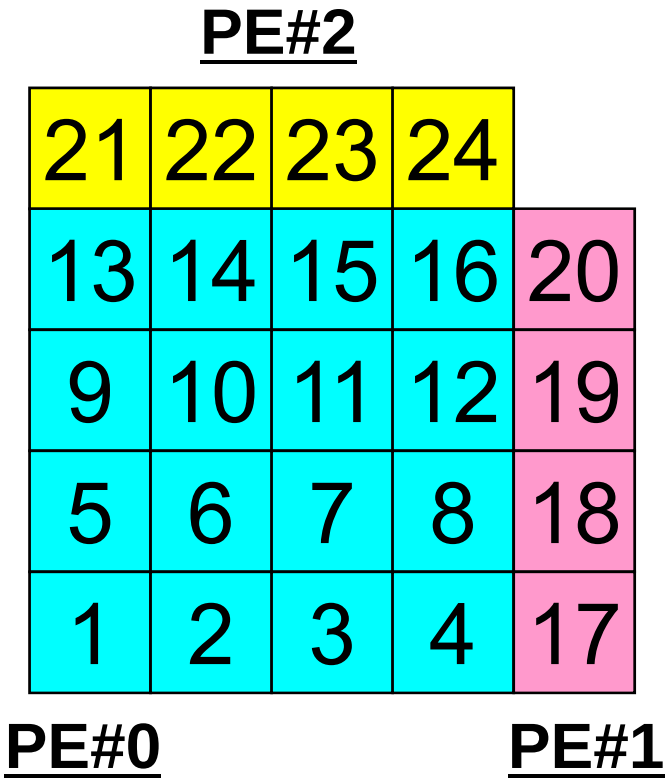
sqm.0:隣接領域数，隣接領域




```
#NEIBPEtot                隣接領域数
2
#NEIBPE
1  2                      隣接領域
#NODE
24 16
#IMPORTindex
4  8
#IMPORTitems
17
18
19
20
21
22
23
24
#EXPORTindex
4  8
#EXPORTitems
4
8
12
16
13
14
15
16
```

PE#0における局所分散データ(3/8)

sqm.0:内点数，総要素(内点＋外点)数

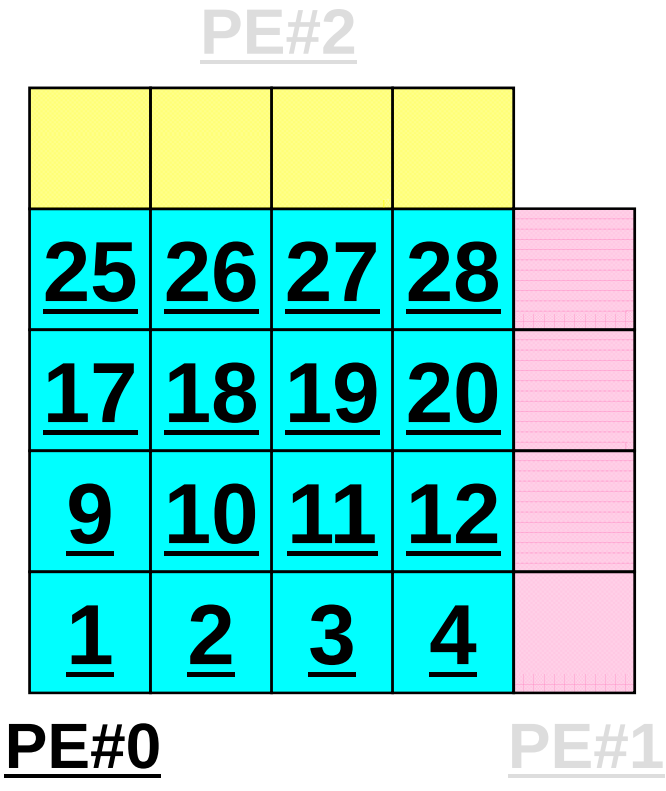



```
#NEIBPEtot
2
#NEIBPE
1  2
#NODE
24 16
#IMPORTindex
4  8
#IMPORTitems
17
18
19
20
21
22
23
24
#EXPORTindex
4  8
#EXPORTitems
4
8
12
16
13
14
15
16
```

総要素数，内点

PE#0における局所分散データ(4/8)

sq.0:内点における値:ここだけ違うファイル

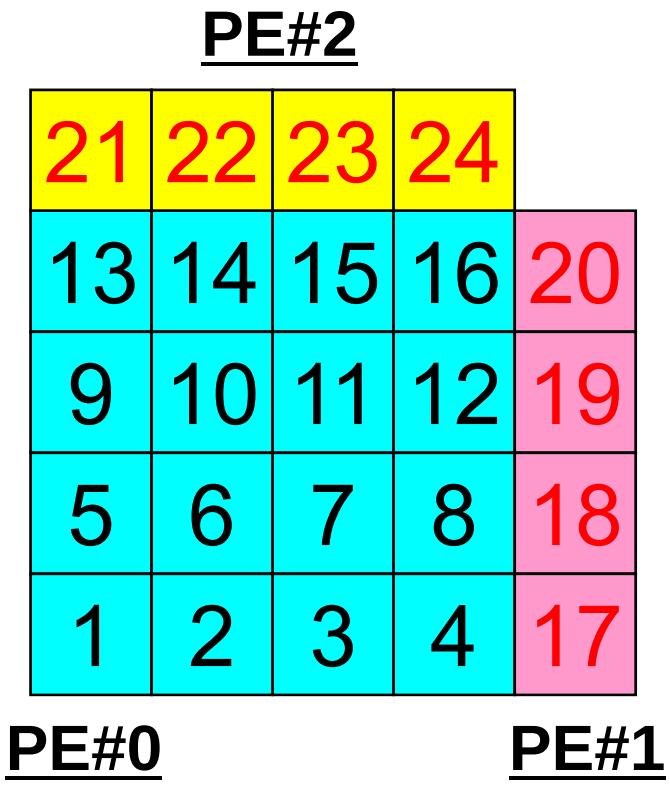



```
1
2
3
4
9
10
11
12
17
18
19
20
25
26
27
28
```

内点における値
(全体番号)

PE#0における局所分散データ(5/8)

sqm.0:「import(受信)」される「外点」の情報

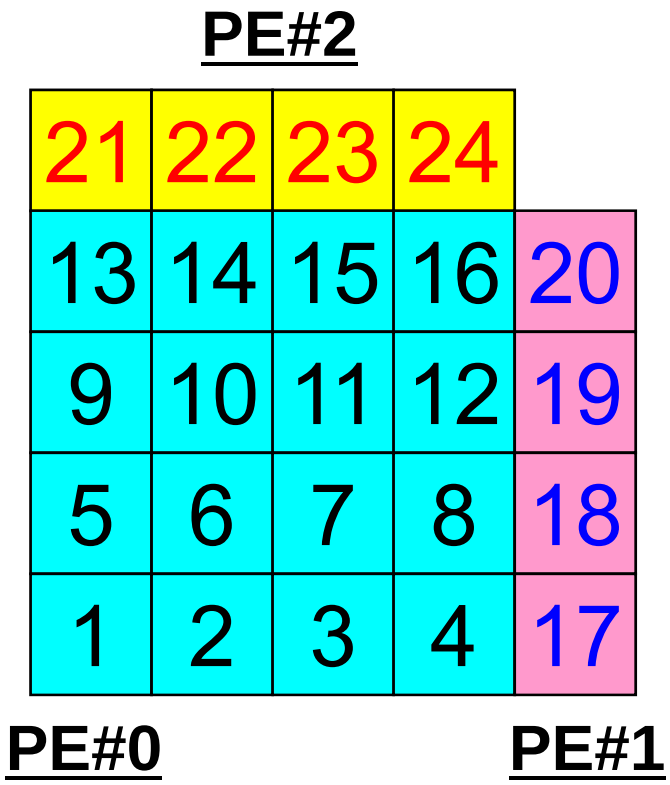



```
#NEIBPEtot
2
#NEIBPE
1  2
#NODE
24 16
#IMPORTindex
4  8
#IMPORTitems
17
18
19
20
21
22
23
24
#EXPORTindex
4  8
#EXPORTitems
4
8
12
16
13
14
15
16
```

隣接領域1から4つ(1~4),
隣接領域2から4つ(5~8)が
「import(受信)」されることを
示す。

PE#0における局所分散データ(6/8)

sqm.0:「import(受信)」される「外点」の情報



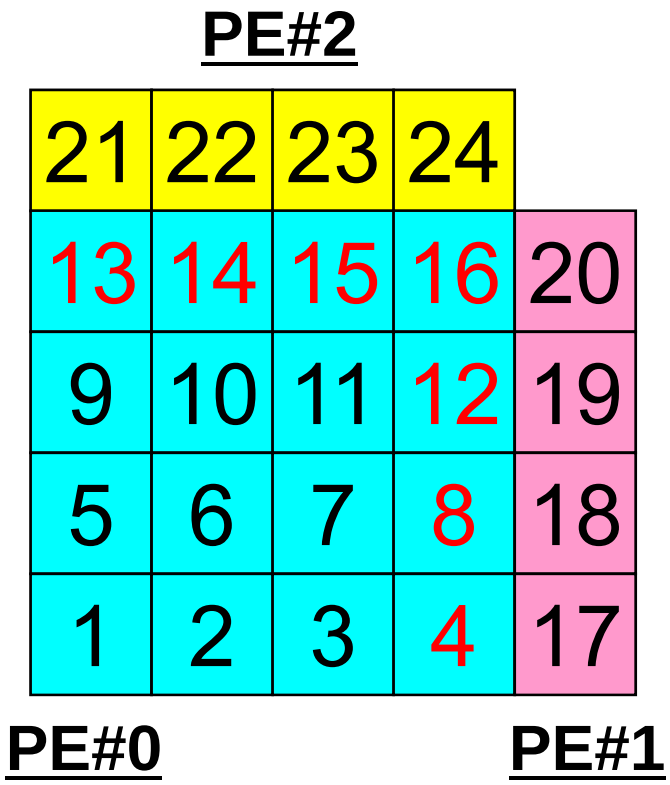

```
#NEIBPEtot
2
#NEIBPE
1  2
#NODE
24 16
#IMPORTindex
4  8
#IMPORTitems
17
18
19
20
21
22
23
24
#EXPORTindex
4  8
#EXPORTitems
4
8
12
16
13
14
15
16
```

隣接領域1から「import」する要素(1~4)

隣接領域2から「import」する要素(5~8)

PE#0における局所分散データ(7/8)

sqm.0:「export(送信)」する「境界点」の情報

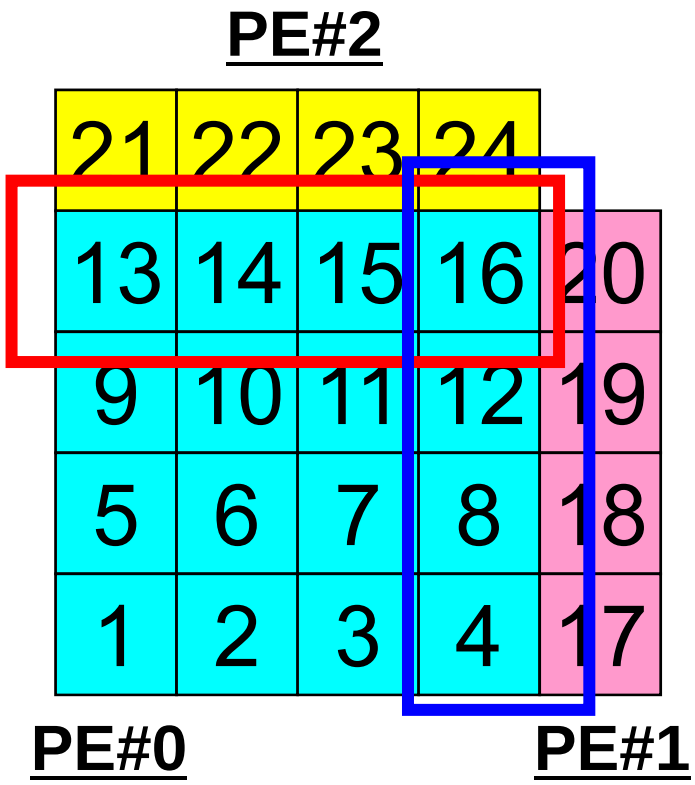



```
#NEIBPEtot
2
#NEIBPE
1  2
#NODE
24 16
#IMPORTindex
4  8
#IMPORTitems
17
18
19
20
21
22
23
24
#EXPORTindex
4  8
#EXPORTitems
4
8
12
16
13
14
15
16
```

隣接領域1へ4つ(1~4),
隣接領域2へ4つ(5~8)が
「export(送信)」されることを
示す。

PE#0における局所分散データ(8/8)

sqm.0:「export(送信)」する「境界点」の情報

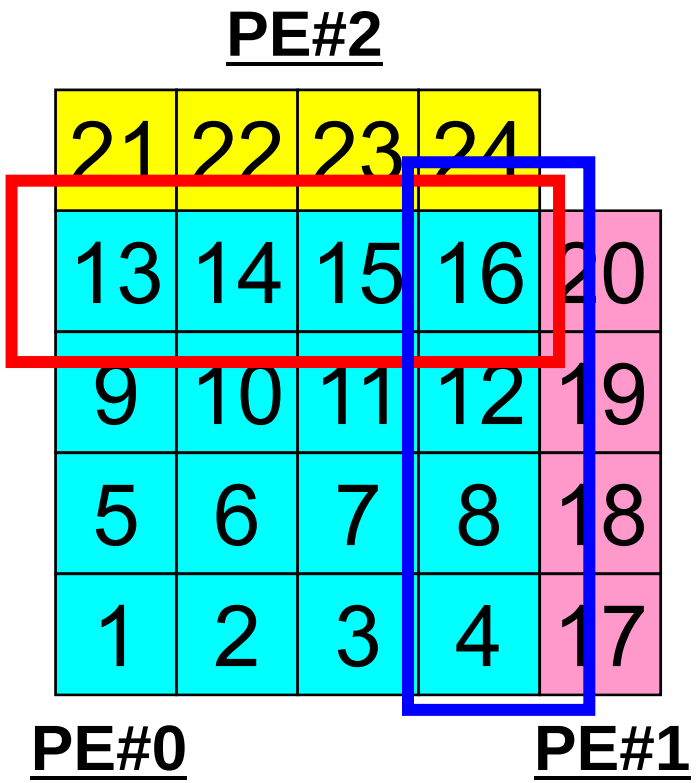


局所番号

```
#NEIBPEtot
2
#NEIBPE
1  2
#NODE
24 16
#IMPORTindex
4  8
#IMPORTitems
17
18
19
20
21
22
23
24
#EXPORTindex
4  8
#EXPORTitems
4                 隣接領域1へ
8                 「export」する要素(1~4)
12
16
13
14                隣接領域2へ
15                「export」する要素(5~8)
16
```

PE#0における局所分散データ(8/8)

sqm.0:「export(送信)」する「境界点」の情報



「外点」はその要素が本来所属している領域からのみ受信される。

「境界点」は複数の領域において「外点」となっている可能性があるので，複数の領域に送信されることもある(16番要素の例)。

プログラム例：sq-sr2.f（1/6）
初期化

```
      implicit REAL*8 (A-H,O-Z)
      include  'mpif.h'

      integer(kind=4) :: my_rank, PETOT
      integer(kind=4) :: N, NP, NEIBPETOT
      integer(kind=4) :: BUFlength_e, BUFlength_i
      integer(kind=4), dimension(:), allocatable :: VAL, SENDbuf, RECVbuf, NEIBPE

      integer(kind=4), dimension(:), allocatable :: import_index, import_item
      integer(kind=4), dimension(:), allocatable :: export_index, export_item
      integer(kind=4), dimension(:), allocatable :: stat_sr
      character(len=80)          :: filename, line

!C
!C +-----------+
!C | INIT. MPI |
!C +-----------+
!C===
      call MPI_INIT      (ierr)
      call MPI_COMM_SIZE (MPI_COMM_WORLD, PETOT, ierr )
      call MPI_COMM_RANK (MPI_COMM_WORLD, my_rank, ierr )
!C===
```

プログラム例：sq-sr2.f（2/6）

局所分散メッシュデータ（sqm.*）読み込み

```
!C
!C-- MESH
      if (my_rank.eq.0) filename= 'sqm.0'
      if (my_rank.eq.1) filename= 'sqm.1'
      if (my_rank.eq.2) filename= 'sqm.2'
      if (my_rank.eq.3) filename= 'sqm.3'
      open (21, file= filename, status= 'unknown')
        read (21,*) NEIBPETOT
                    allocate (NEIBPE(NEIBPETOT))
                    allocate (import_index(0:NEIBPETOT))
                    allocate (export_index(0:NEIBPETOT))
                              import_index= 0
                              export_index= 0
        read (21,*) (NEIBPE(neib), neib= 1, NEIBPETOT)
        read (21,*) NP, N
        read (21,'(a80)') line
        read (21,*) (import_index(neib), neib= 1, NEIBPETOT)
                     nn= import_index(NEIBPETOT)
                     allocate (import_item(nn))
        do i= 1, nn
          read (21,*) import_item(i)
        enddo
        read (21,'(a80)') line
        read (21,*) (export_index(neib), neib= 1, NEIBPETOT)
                     nn= export_index(NEIBPETOT)
                     allocate (export_item(nn))
        do i= 1, nn
          read (21,*) export_item(i)
        enddo
      close (21)
```

プログラム例：sq-sr2.f（2/6）

局所分散メッシュデータ（sqm.*）読み込み

```
!C
!C-- MESH
      if (my_rank.eq.0) filename= 'sqm.0'
      if (my_rank.eq.1) filename= 'sqm.1'
      if (my_rank.eq.2) filename= 'sqm.2'
      if (my_rank.eq.3) filename= 'sqm.3'
      open (21, file= filename, status= 'unknown')
        read (21,*) NEIBPETOT
                    allocate (NEIBPE(NEIBPETOT))
                    allocate (import_index(0:NEIBPETOT))
                    allocate (export_index(0:NEIBPETOT))
                              import_index= 0
                              export_index= 0
        read (21,*) (NEIBPE(neib), neib= 1, NEIBPETOT)
        read (21,*) NP, N

        read (21,*) (import_index(neib), neib= 1, NEIBPETOT)
                     nn= import_index(NEIBPETOT)
                     allocate (import_item(nn))
        do i= 1, nn
          read (21,*) import_item(i)
        enddo

        read (21,*) (export_index(neib), neib= 1, NEIBPETOT)
                     nn= export_index(NEIBPETOT)
                     allocate (export_item(nn))
        do i= 1, nn
          read (21,*) export_item(i)
        enddo
      close (21)
```

```
#NEIBPEtot
2
#NEIBPE
1  2
#NODE
24 16
#IMPORTindex
4  8
#IMPORTitems
17
18
19
20
21
22
23
24
#EXPORTindex
4  8
#EXPORTitems
4
8
12
16
13
14
15
16
```

プログラム例：sq-sr2.f（2/6）

局所分散メッシュデータ（sqm.*）読み込み

```
!C
!C-- MESH
      if (my_rank.eq.0) filename= 'sqm.0'
      if (my_rank.eq.1) filename= 'sqm.1'
      if (my_rank.eq.2) filename= 'sqm.2'
      if (my_rank.eq.3) filename= 'sqm.3'
      open (21, file= filename, status= 'unknown')
        read (21,*) NEIBPETOT
                    allocate (NEIBPE(NEIBPETOT))
                    allocate (import_index(0:NEIBPETOT))
                    allocate (export_index(0:NEIBPETOT))
                              import_index= 0
                              export_index= 0
        read (21,*) (NEIBPE(neib), neib= 1, NEIBPETOT)
        read (21,*) NP, N
                    ') line
                    port_index(neib), neib= 1, NEIBPETOT)
                    = import_index(NEIBPETOT)
                    locate (import_item(nn))
        do i= 1, nn
          read (21,*) import_item(i)
        enddo
        read (21,'(a80)') line
        read (21,*) (export_index(neib), neib= 1, NEIBPETOT)
                     nn= export_index(NEIBPETOT)
                     allocate (export_item(nn))
        do i= 1, nn
          read (21,*) export_item(i)
        enddo
      close (21)
```

NP　総要素数
N　内点数

```
#NEIBPEtot
2
#NEIBPE
1  2
#NODE
24 16
#IMPORTindex
4  8
#IMPORTitems
17
18
19
20
21
22
23
24
#EXPORTindex
4  8
#EXPORTitems
4
8
12
16
13
14
15
16
```

プログラム例：sq-sr2.f（2/6）

局所分散メッシュデータ（sqm.*）読み込み

```
!C
!C-- MESH
      if (my_rank.eq.0) filename= 'sqm.0'
      if (my_rank.eq.1) filename= 'sqm.1'
      if (my_rank.eq.2) filename= 'sqm.2'
      if (my_rank.eq.3) filename= 'sqm.3'
      open (21, file= filename, status= 'unknown')
        read (21,*) NEIBPETOT
                    allocate (NEIBPE(NEIBPETOT))
                    allocate (import_index(0:NEIBPETOT))
                    allocate (export_index(0:NEIBPETOT))
                              import_index= 0
                              export_index= 0
        read (21,*) (NEIBPE(neib), neib= 1, NEIBPETOT)
        read (21,*) NP, N

        read (21,*) (import_index(neib), neib= 1, NEIBPETOT)
                     nn= import_index(NEIBPETOT)
                     allocate (import_item(nn))
        do i= 1, nn
          read (21,*) import_item(i)
        enddo

        read (21,*) (export_index(neib), neib= 1, NEIBPETOT)
                     nn= export_index(NEIBPETOT)
                     allocate (export_item(nn))
        do i= 1, nn
          read (21,*) export_item(i)
        enddo
      close (21)
```

```
#NEIBPEtot
2
#NEIBPE
1  2
#NODE
24 16
#IMPORTindex
4  8
#IMPORTitems
17
18
19
20
21
22
23
24
#EXPORTindex
4  8
#EXPORTitems
4
8
12
16
13
14
15
16
```

プログラム例：sq-sr2.f（2/6）

局所分散メッシュデータ（sqm.*）読み込み

```
!C
!C-- MESH
      if (my_rank.eq.0) filename= 'sqm.0'
      if (my_rank.eq.1) filename= 'sqm.1'
      if (my_rank.eq.2) filename= 'sqm.2'
      if (my_rank.eq.3) filename= 'sqm.3'
      open (21, file= filename, status= 'unknown')
        read (21,*) NEIBPETOT
                    allocate (NEIBPE(NEIBPETOT))
                    allocate (import_index(0:NEIBPETOT))
                    allocate (export_index(0:NEIBPETOT))
                              import_index= 0
                              export_index= 0
        read (21,*) (NEIBPE(neib), neib= 1, NEIBPETOT)
        read (21,*) NP, N

        read (21,*) (import_index(neib), neib= 1, NEIBPETOT)
                     nn= import_index(NEIBPETOT)
                     allocate (import_item(nn))
        do i= 1, nn
          read (21,*) import_item(i)
        enddo

        read (21,*) (export_index(neib), neib= 1, NEIBPETOT)
                     nn= export_index(NEIBPETOT)
                     allocate (export_item(nn))
        do i= 1, nn
          read (21,*) export_item(i)
        enddo
      close (21)
```

```
#NEIBPEtot
2
#NEIBPE
1  2
#NODE
24 16
#IMPORTindex
4  8
#IMPORTitems
17
18
19
20
21
22
23
24
#EXPORTindex
4  8
#EXPORTitems
4
8
12
16
13
14
15
16
```

プログラム例：sq-sr2.f（2/6）

局所分散メッシュデータ（sqm.*）読み込み

```
!C
!C-- MESH
      if (my_rank.eq.0) filename= 'sqm.0'
      if (my_rank.eq.1) filename= 'sqm.1'
      if (my_rank.eq.2) filename= 'sqm.2'
      if (my_rank.eq.3) filename= 'sqm.3'
      open (21, file= filename, status= 'unknown')
        read (21,*) NEIBPETOT
                    allocate (NEIBPE(NEIBPETOT))
                    allocate (import_index(0:NEIBPETOT))
                    allocate (export_index(0:NEIBPETOT))
                              import_index= 0
                              export_index= 0
        read (21,*) (NEIBPE(neib), neib= 1, NEIBPETOT)
        read (21,*) NP, N

        read (21,*) (import_index(neib), neib= 1, NEIBPETOT)
                     nn= import_index(NEIBPETOT)
                     allocate (import_item(nn))
        do i= 1, nn
          read (21,*) import_item(i)
        enddo

        read (21,*) (export_index(neib), neib= 1, NEIBPETOT)
                     nn= export_index(NEIBPETOT)
                     allocate (export_item(nn))
        do i= 1, nn
          read (21,*) export_item(i)
        enddo
      close (21)
```

```
#NEIBPEtot
2
#NEIBPE
1  2
#NODE
24 16
#IMPORTindex
4  8
#IMPORTitems
17
18
19
20
21
22
23
24
#EXPORTindex
4  8
#EXPORTitems
4
8
12
16
13
14
15
16
```

プログラム例：sq-sr2.f（2/6）

局所分散メッシュデータ（sqm.*）読み込み

```
!C
!C-- MESH
      if (my_rank.eq.0) filename= 'sqm.0'
      if (my_rank.eq.1) filename= 'sqm.1'
      if (my_rank.eq.2) filename= 'sqm.2'
      if (my_rank.eq.3) filename= 'sqm.3'
      open (21, file= filename, status= 'unknown')
        read (21,*) NEIBPETOT
                    allocate (NEIBPE(NEIBPETOT))
                    allocate (import_index(0:NEIBPETOT))
                    allocate (export_index(0:NEIBPETOT))
                              import_index= 0
                              export_index= 0
        read (21,*) (NEIBPE(neib), neib= 1, NEIBPETOT)
        read (21,*) NP, N

        read (21,*) (import_index(neib), neib= 1, NEIBPETOT)
                     nn= import_index(NEIBPETOT)
                     allocate (import_item(nn))
        do i= 1, nn
          read (21,*) import_item(i)
        enddo

        read (21,*) (export_index(neib), neib= 1, NEIBPETOT)
                     nn= export_index(NEIBPETOT)
                     allocate (export_item(nn))
        do i= 1, nn
          read (21,*) export_item(i)
        enddo
      close (21)
```

```
#NEIBPEtot
2
#NEIBPE
1  2
#NODE
24 16
#IMPORTindex
4  8
#IMPORTitems
17
18
19
20
21
22
23
24
#EXPORTindex
4  8
#EXPORTitems
4
8
12
16
13
14
15
16
```

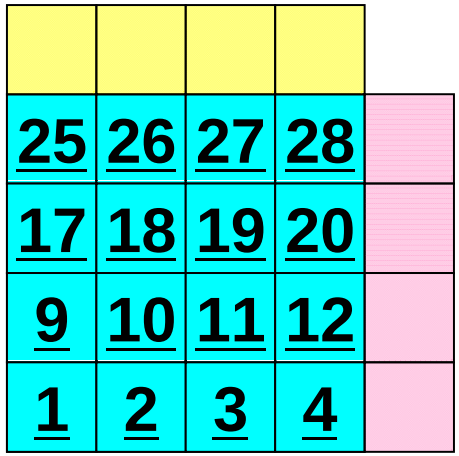
プログラム例：sq-sr2.f（3/6）

局所分散データ（全体番号の値）（sq.*）読み込み

```
!C
!C-- VAL.
      if (my_rank.eq.0) filename= 'sq.0'
      if (my_rank.eq.1) filename= 'sq.1'
      if (my_rank.eq.2) filename= 'sq.2'
      if (my_rank.eq.3) filename= 'sq.3'

      allocate (VAL(NP))
      VAL= 0
      open (21, file= filename, status= 'unknown')
        do i= 1, N
          read (21,*) VAL(i)
        enddo
      close (21)
!C===
```

N　　：内点数
VAL：全体要素番号を読み込む
この時点で外点の値はわかっていない

PE#2



PE#0　　PE#1

```
1
2
3
4
9
10
11
12
17
18
19
20
25
26
27
28
```

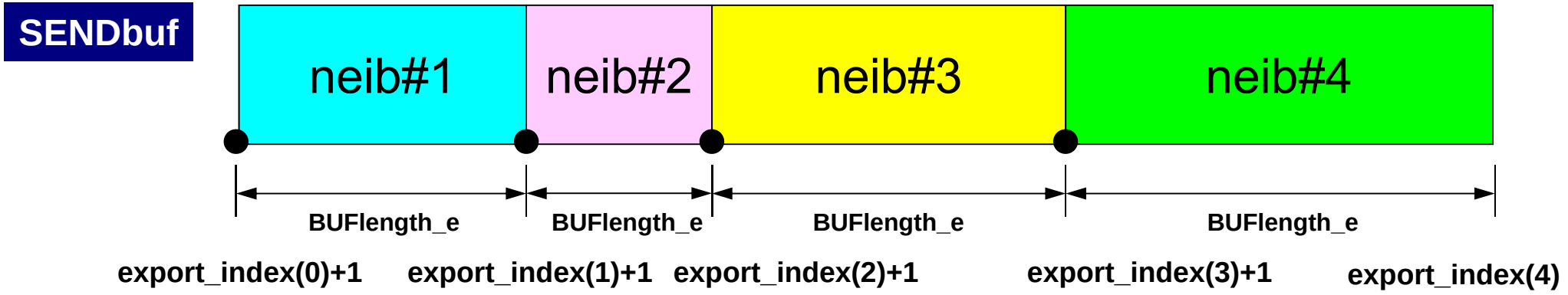
プログラム例：sq-sr2.f（4/6）

送・受信バッファ準備

```
!C
!C +--------+
!C | BUFFER |
!C +--------+
!C===
      allocate (SENDbuf(export_index(NEIBPETOT)))
      allocate (RECVbuf(import_index(NEIBPETOT)))

      SENDbuf= 0
      RECVbuf= 0

      do neib= 1, NEIBPETOT
        iS= export_index(neib-1) + 1
        iE= export_index(neib  )
        do i= iS, iE
          SENDbuf(i)= VAL(export_item(i))
        enddo
      enddo
!C===
```

送信バッファに「境界点」の情報を入れる。送信バッファの**export_index(neib-1)+1**から**export_inedx(neib)**までに**NEIBPE(neib)**に送信する情報を格納する。

送信（MPI_Isend/Irecv/Waitall）



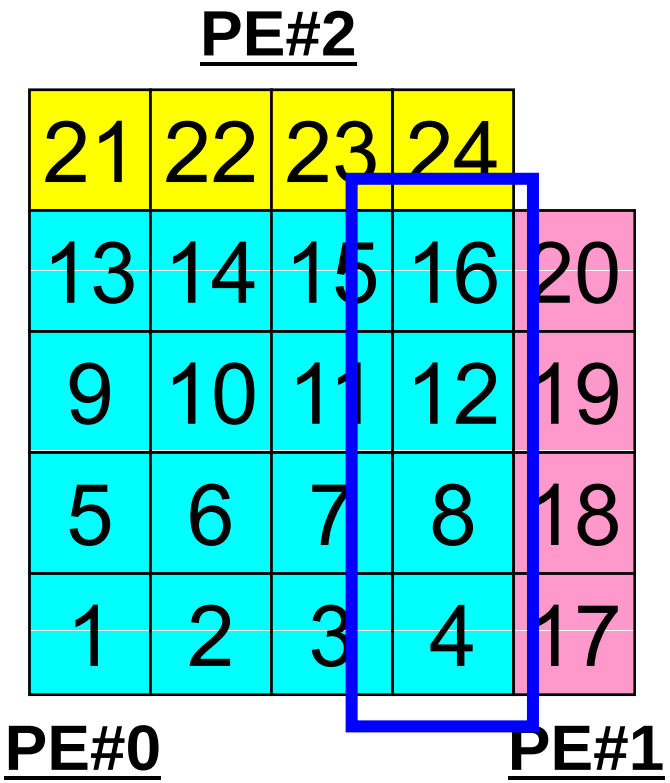
export_index(neib-1)+1～export_index(neib)番目のexport_itemがneib番目の隣接領域に送信される

```
do neib= 1, NEIBPETOT
  do k= export_index(neib-1)+1, export_index(neib)
    kk= export_item(k)
    SENDbuf(k)= VAL(kk)
  enddo
enddo

do neib= 1, NEIBPETOT
  iS_e= export_index(neib-1) + 1
  iE_e= export_index(neib  )
  BUFlength_e= iE_e + 1 - iS_e
  call MPI_ISEND                                                  &
&         (SENDbuf(iS_e), BUFlength_e, MPI_INTEGER, NEIBPE(neib), 0,&
&          MPI_COMM_WORLD, request_send(neib), ierr)
enddo

call MPI_WAITALL (NEIBPETOT, request_send, stat_recv, ierr)
```

送信バッファへの代入
温度などの変数を直接送信，受信に使うのではなく，このようなバッファへ一回代入して計算する

送信バッファの効能

```
 do neib= 1, NEIBPETOT
   iS_e= export_index(neib-1) + 1
   iE_e= export_index(neib  )
   BUFlength_e= iE_e + 1 - iS_e

   call MPI_ISEND                                                  &
&         (VAL(...), BUFlength_e, MPI_INTEGER, NEIBPE(neib), 0,&
&          MPI_COMM_WORLD, request_send(neib), ierr)
 enddo
```

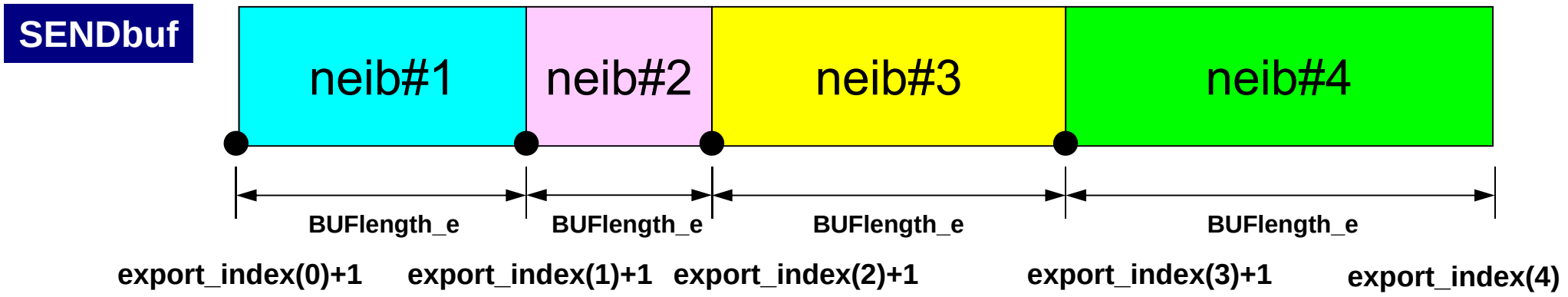


たとえば，この境界点は連続していないので，

- 送信バッファの先頭アドレス
- そこから数えて●●のサイズのメッセージ

というような方法が困難

送信（MPI_Sendrecv）



export_index(neib-1)+1～export_index(neib)番目のexport_itemがneib番目の隣接領域に送信される

```
do neib= 1, NEIBPETOT
  do k= export_index(neib-1)+1, export_index(neib)
    kk= export_item(k)
    SENDbuf(k)= VAL(kk)
  enddo
enddo

do neib= 1, NEIBPETOT
  iS_e= export_index(neib-1) + 1
  iE_e= export_index(neib  )
  BUFlength_e= iE_e + 1 - iS_e

  call MPI_SENDRECV                                                  &
&         (SENDbuf(iS_e), BUFlength_e, MPI_INTEGER, NEIBPE(neib), 0,&
&          RECVbuf(iS_i), BUFlength_i, MPI_INTEGER, NEIBPE(neib), 0,&
&          MPI_COMM_WORLD, stat_sr, ierr)
  enddo
```

送信バッファへの代入

プログラム例：sq-sr2.f（5/6）

送・受信実施（MPI_SENDRECV）

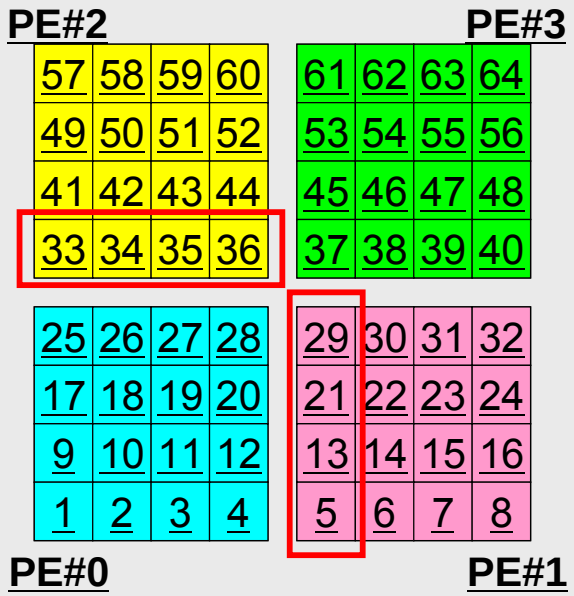
```
!C
!C +-----------+
!C | SEND-RECV |
!C +-----------+
!C===
      allocate (stat_sr(MPI_STATUS_SIZE))

      do neib= 1, NEIBPETOT
        iS_e= export_index(neib-1) + 1
        iE_e= export_index(neib  )
        BUFlength_e= iE_e + 1 - iS_e

        iS_i= import_index(neib-1) + 1
        iE_i= import_index(neib  )
        BUFlength_i= iE_i + 1 - iS_i

        call MPI_SENDRECV                                              &
     &       (SENDbuf(iS_e), BUFlength_e, MPI_INTEGER, NEIBPE(neib), 0,&
     &        RECVbuf(iS_i), BUFlength_i, MPI_INTEGER, NEIBPE(neib), 0,&
     &        MPI_COMM_WORLD, stat_sr, ierr)
      enddo

      do neib= 1, NEIBPETOT
        iS= import_index(neib-1) + 1
        iE= import_index(neib  )
        do i= iS, iE
          VAL(import_item(i))= RECVbuf(i)
        enddo
      enddo
!C===
```

送信側の「BUFlength_e」と受信側の「BUFlength_i」は一致している必要がある。

プログラム例：sq-sr2.f（5/6）

送・受信実施（MPI_SENDRECV）

```
!C
!C +-----------+
!C | SEND-RECV |
!C +-----------+
!C===
      allocate (stat_sr(MPI_STATUS_SIZE))

      do neib= 1, NEIBPETOT
        iS_e= export_index(neib-1) + 1
        iE_e= export_index(neib  )
        BUFlength_e= iE_e + 1 - iS_e

        iS_i= import_index(neib-1) + 1
        iE_i= import_index(neib  )
        BUFlength_i= iE_i + 1 - iS_i

        call MPI_SENDRECV                                              &
     &       (SENDbuf(iS_e), BUFlength_e, MPI_INTEGER, NEIBPE(neib), 0,&
     &        RECVbuf(iS_i), BUFlength_i, MPI_INTEGER, NEIBPE(neib), 0,&
     &        MPI_COMM_WORLD, stat_sr, ierr)
      enddo

      do neib= 1, NEIBPETOT
        iS= import_index(neib-1) + 1
        iE= import_index(neib  )
        do i= iS, iE
          VAL(import_item(i))= RECVbuf(i)
        enddo
      enddo
!C===
```

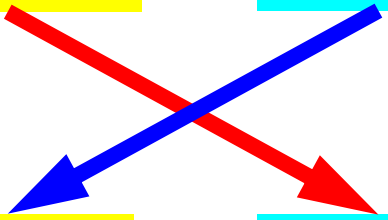


my_rank=0, NEIBPE(neib)=1のときのBUFlength_eと，my_rank=1, NEIBPE(neib)=0のときのBUFlength_iは同じでなければならない（この場合はいずれも4）。

配列の送受信:注意

```
#PE0
  send:
    SENDbuf(iS_e)~
    SENDbuf(iE_e+BUFlength_e-1)
```

```
#PE1
  send:
    SENDbuf(iS_e)~
    SENDbuf(iE_e+BUFlength_e-1)
```



```
#PE0
  recv:
    RECVbuf(iS_i)~
    RECVbuf(iE_i+Buflength_i-1)
```

```
#PE1
  recv:
    RECVbuf(iS_i)~
    RECVbuf(iE_i+Buflength_i-1)
```

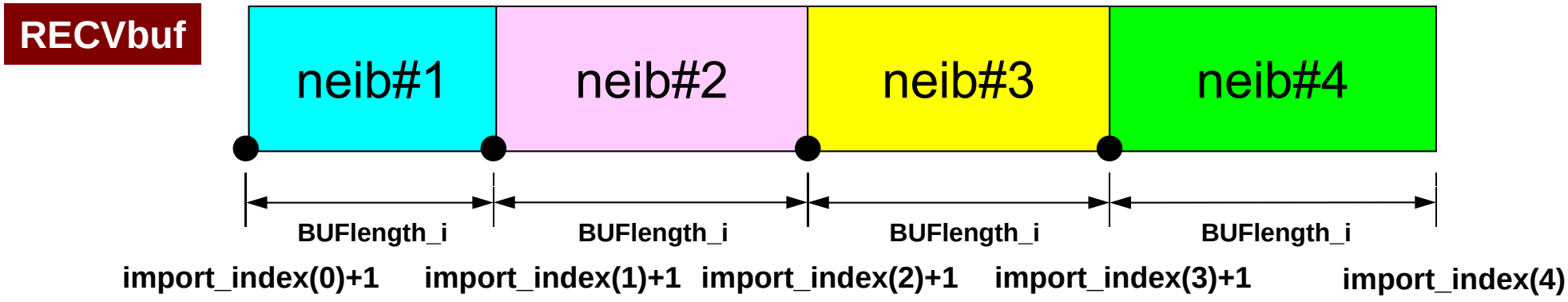
- 送信側の「BUFlength_e」と受信側の「BUFlength_i」は一致している必要がある。
  - PE#0⇒PE#1，PE#1⇒PE#0
- 「送信バッファ」と「受信バッファ」は別のアドレス

受信（MPI_Isend/Irecv/Waitall）

```
 do neib= 1, NEIBPETOT
   iS_i= import_index(neib-1) + 1
   iE_i= import_index(neib  )
   BUFlength_i= iE_i + 1 - iS_i

   call MPI_IRECV                                                 &
&         (RECVbuf(iS_i), BUFlength_i, MPI_INTEGER, NEIBPE(neib), 0,&
&          MPI_COMM_WORLD, request_recv(neib), ierr)
  enddo
  call MPI_WAITALL (NEIBPETOT, request_recv, stat_recv, ierr)

  do neib= 1, NEIBPETOT
     do k= import_index(neib-1)+1, import_index(neib)
       kk= import_item(k)
       VAL(kk)= RECVbuf(k)
     enddo
  enddo
```

受信バッファから代入

import_index(neib-1)+1～import_index(neib)番目のimport_itemがneib番目の隣接領域から受信される

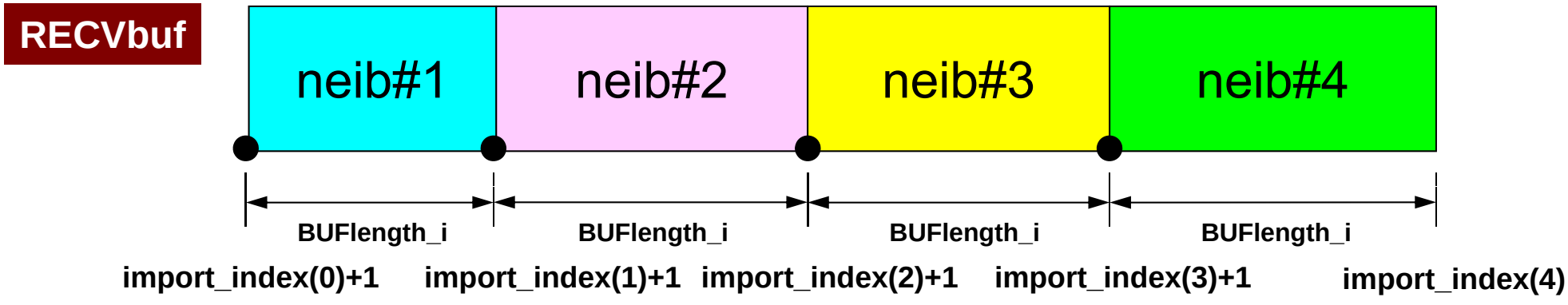


受信（MPI_Sendrecv）

```
 do neib= 1, NEIBPETOT
   iS_i= import_index(neib-1) + 1
   iE_i= import_index(neib  )
   BUFlength_i= iE_i + 1 - iS_i

   call MPI_SENDRECV                                                &
&         (SENDbuf(iS_e), BUFlength_e, MPI_INTEGER, NEIBPE(neib), 0,&
&          RECVbuf(iS_i), BUFlength_i, MPI_INTEGER, NEIBPE(neib), 0,&
&          MPI_COMM_WORLD, stat_sr, ierr)
  enddo

  do neib= 1, NEIBPETOT
    do k= import_index(neib-1)+1, import_index(neib)
      kk= import_item(k)
      VAL(kk)= RECVbuf(k)
    enddo
  enddo
```

受信バッファからの代入

import_index(neib-1)+1～import_index(neib)番目のimport_itemがneib番目の隣接領域から受信される



プログラム例：sq-sr2.f（5/6）

送・受信実施（MPI_SENDRECV）

```
!C
!C +-----------+
!C | SEND-RECV |
!C +-----------+
!C===
      allocate (stat_sr(MPI_STATUS_SIZE))

      do neib= 1, NEIBPETOT
        iS_e= export_index(neib-1) + 1
        iE_e= export_index(neib  )
        BUFlength_e= iE_e + 1 - iS_e

        iS_i= import_index(neib-1) + 1
        iE_i= import_index(neib  )
        BUFlength_i= iE_i + 1 - iS_i

        call MPI_SENDRECV                                                &
     &       (SENDbuf(iS_e), BUFlength_e, MPI_INTEGER, NEIBPE(neib), 0,&
     &        RECVbuf(iS_i), BUFlength_i, MPI_INTEGER, NEIBPE(neib), 0,&
     &        MPI_COMM_WORLD, stat_sr, ierr)
      enddo

      do neib= 1, NEIBPETOT
        iS= import_index(neib-1) + 1
        iE= import_index(neib  )
        do i= iS, iE
          VAL(import_item(i))= RECVbuf(i)
        enddo
      enddo
!C===
```

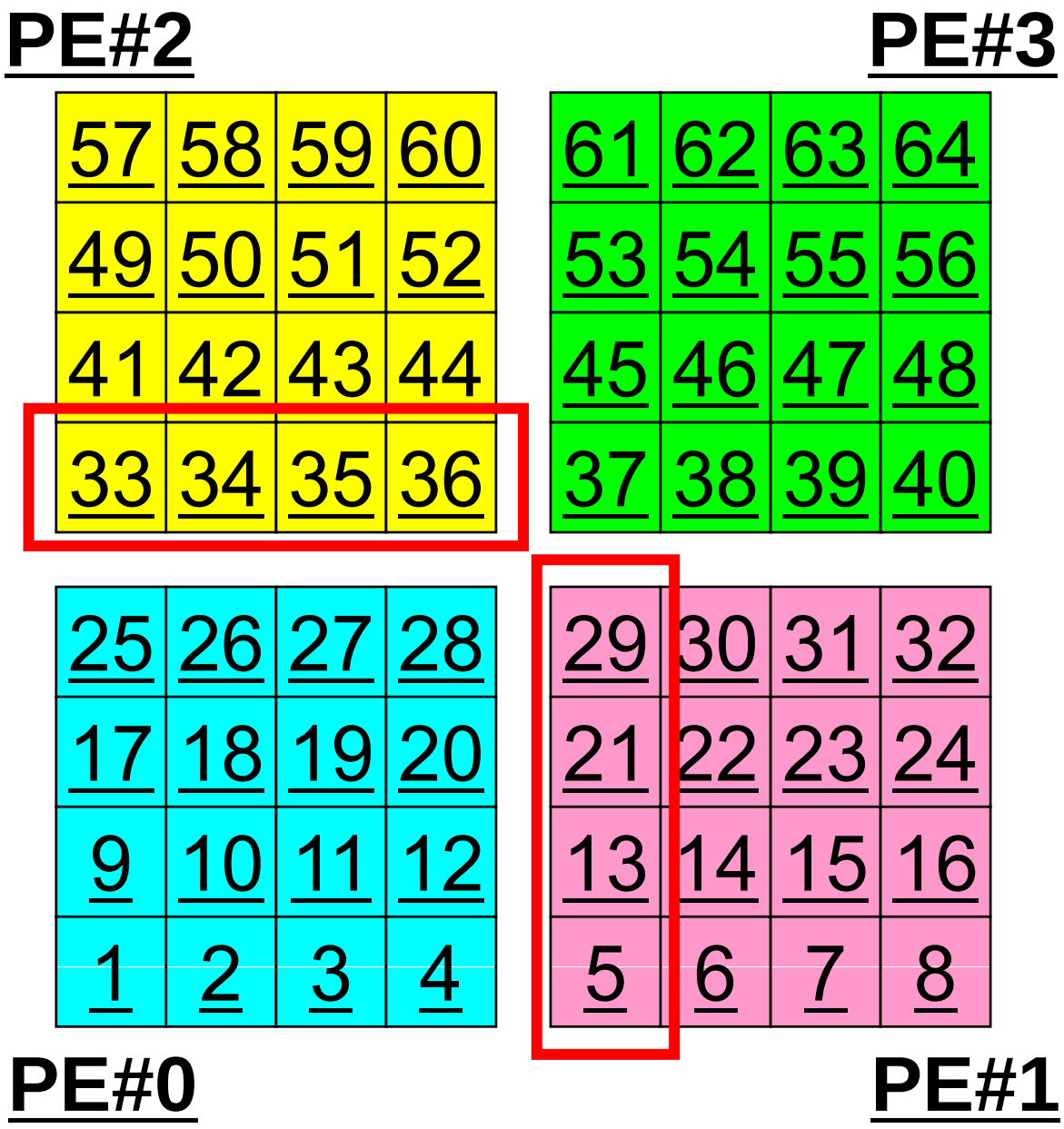
受信バッファの中身を「外点」の値として代入する。

プログラム例：sq-sr2.f（6/6）

外点の値の書き出し

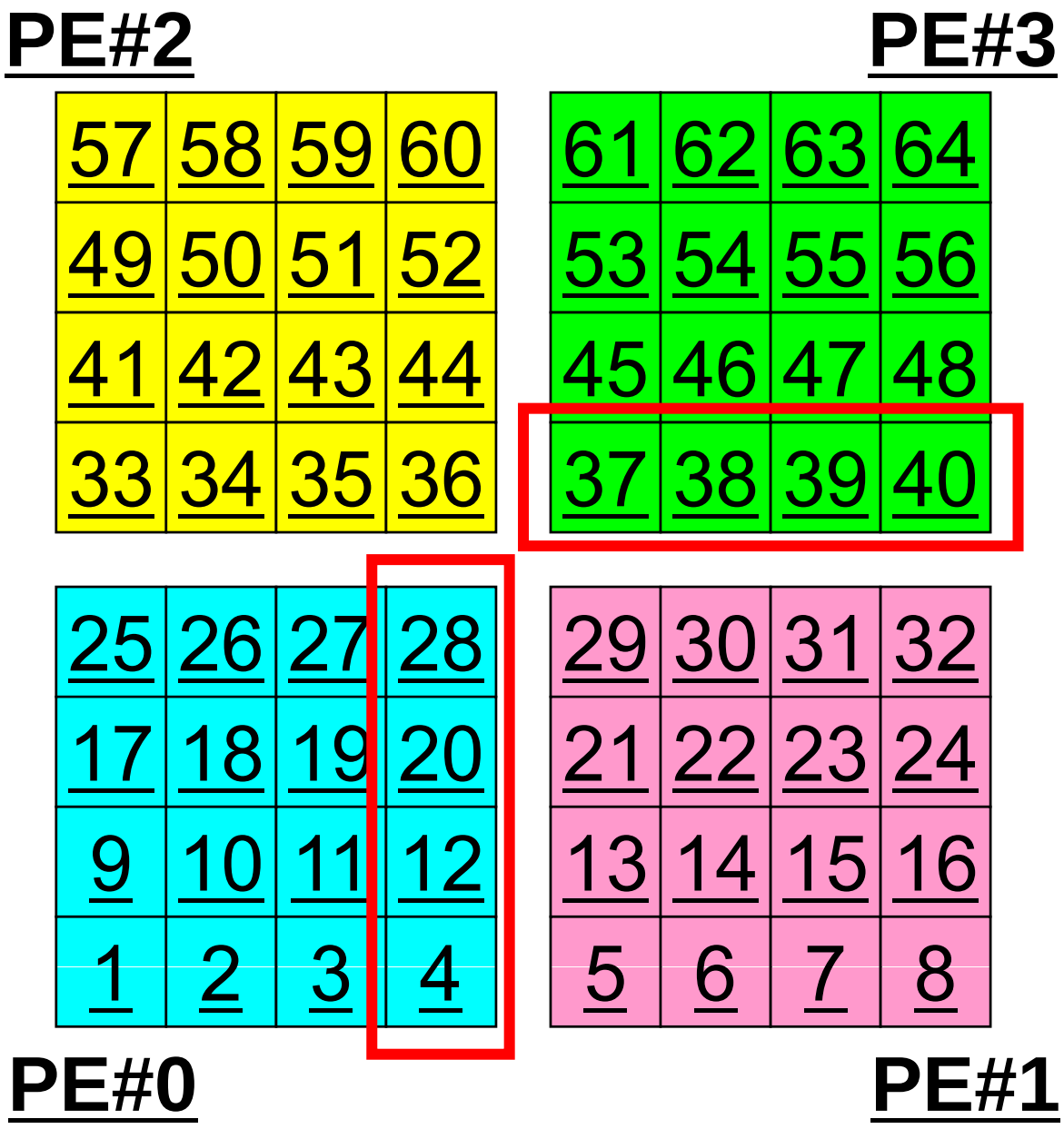
```
!C
!C +--------+
!C | OUTPUT |
!C +--------+
!C===
      do neib= 1, NEIBPETOT
        iS= import_index(neib-1) + 1
        iE= import_index(neib  )
        do i= iS, iE
          in= import_item(i)
          write (*,'(a, 3i8)') 'RECVbuf', my_rank, NEIBPE(neib), VAL(in)
        enddo
      enddo
!C===
      call MPI_FINALIZE (ierr)

      stop
      end
```

実行結果（PE#0）



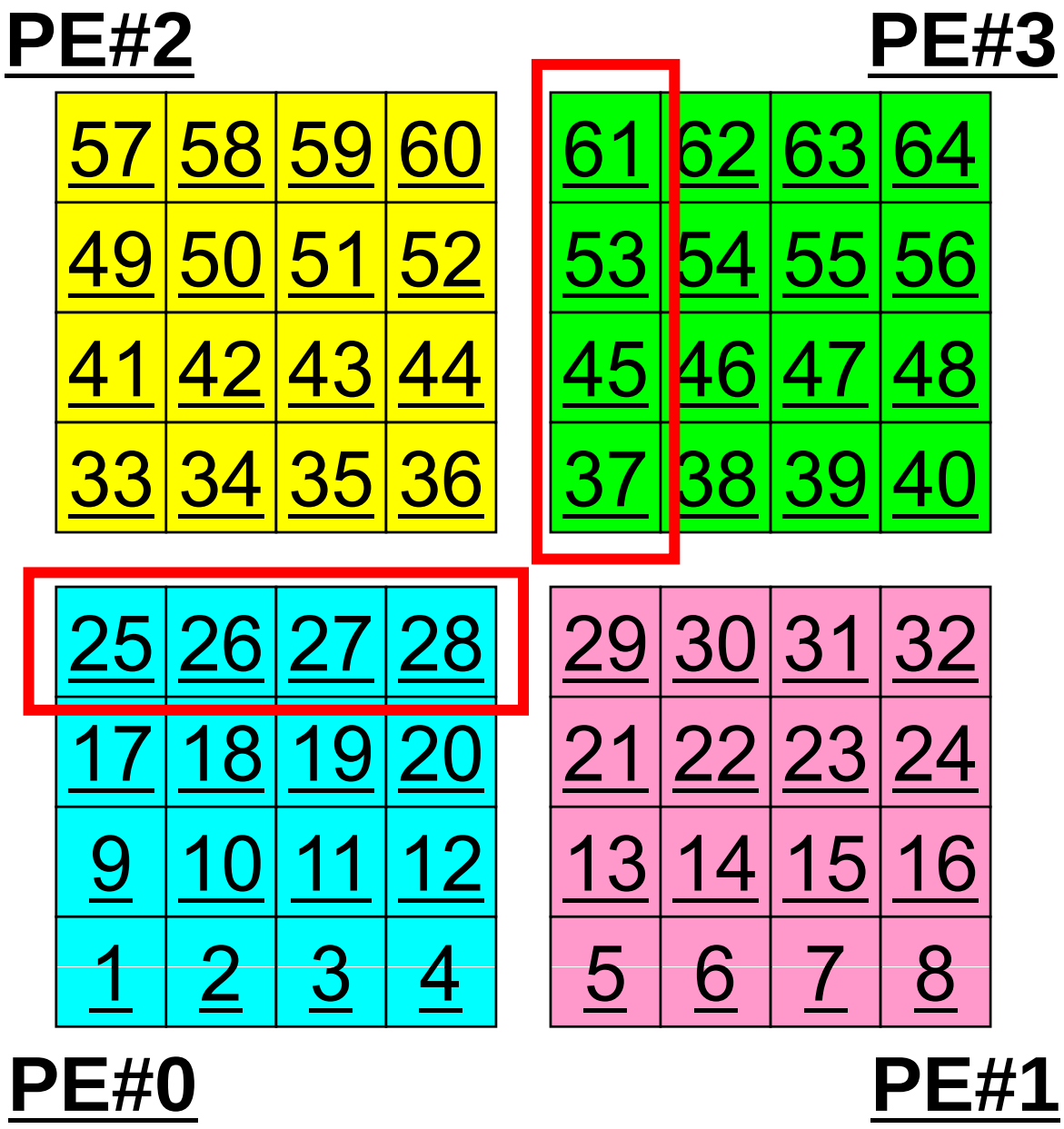

```
RECVbuf         0         1          5
RECVbuf         0         1         13
RECVbuf         0         1         21
RECVbuf         0         1         29
RECVbuf         0         2         33
RECVbuf         0         2         34
RECVbuf         0         2         35
RECVbuf         0         2         36

RECVbuf         1         0          4
RECVbuf         1         0         12
RECVbuf         1         0         20
RECVbuf         1         0         28
RECVbuf         1         3         37
RECVbuf         1         3         38
RECVbuf         1         3         39
RECVbuf         1         3         40

RECVbuf         2         3         37
RECVbuf         2         3         45
RECVbuf         2         3         53
RECVbuf         2         3         61
RECVbuf         2         0         25
RECVbuf         2         0         26
RECVbuf         2         0         27
RECVbuf         2         0         28

RECVbuf         3         2         36
RECVbuf         3         2         44
RECVbuf         3         2         52
RECVbuf         3         2         60
RECVbuf         3         1         29
RECVbuf         3         1         30
RECVbuf         3         1         31
RECVbuf         3         1         32
```

実行結果（PE#1）



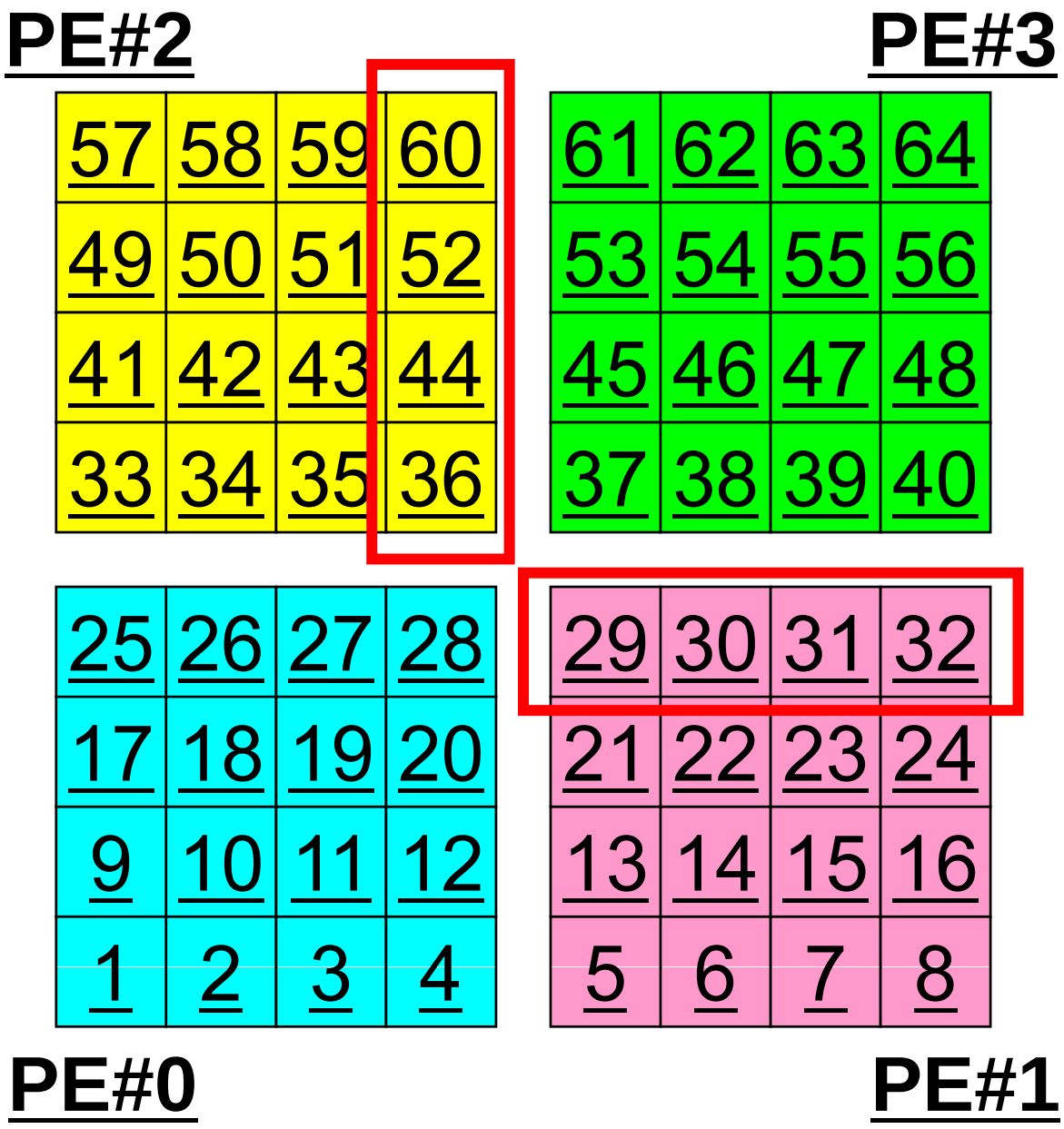

```
RECVbuf         0         1          5
RECVbuf         0         1         13
RECVbuf         0         1         21
RECVbuf         0         1         29
RECVbuf         0         2         33
RECVbuf         0         2         34
RECVbuf         0         2         35
RECVbuf         0         2         36

RECVbuf         1         0          4
RECVbuf         1         0         12
RECVbuf         1         0         20
RECVbuf         1         0         28
RECVbuf         1         3         37
RECVbuf         1         3         38
RECVbuf         1         3         39
RECVbuf         1         3         40

RECVbuf         2         3         37
RECVbuf         2         3         45
RECVbuf         2         3         53
RECVbuf         2         3         61
RECVbuf         2         0         25
RECVbuf         2         0         26
RECVbuf         2         0         27
RECVbuf         2         0         28

RECVbuf         3         2         36
RECVbuf         3         2         44
RECVbuf         3         2         52
RECVbuf         3         2         60
RECVbuf         3         1         29
RECVbuf         3         1         30
RECVbuf         3         1         31
RECVbuf         3         1         32
```

実行結果（PE#2）



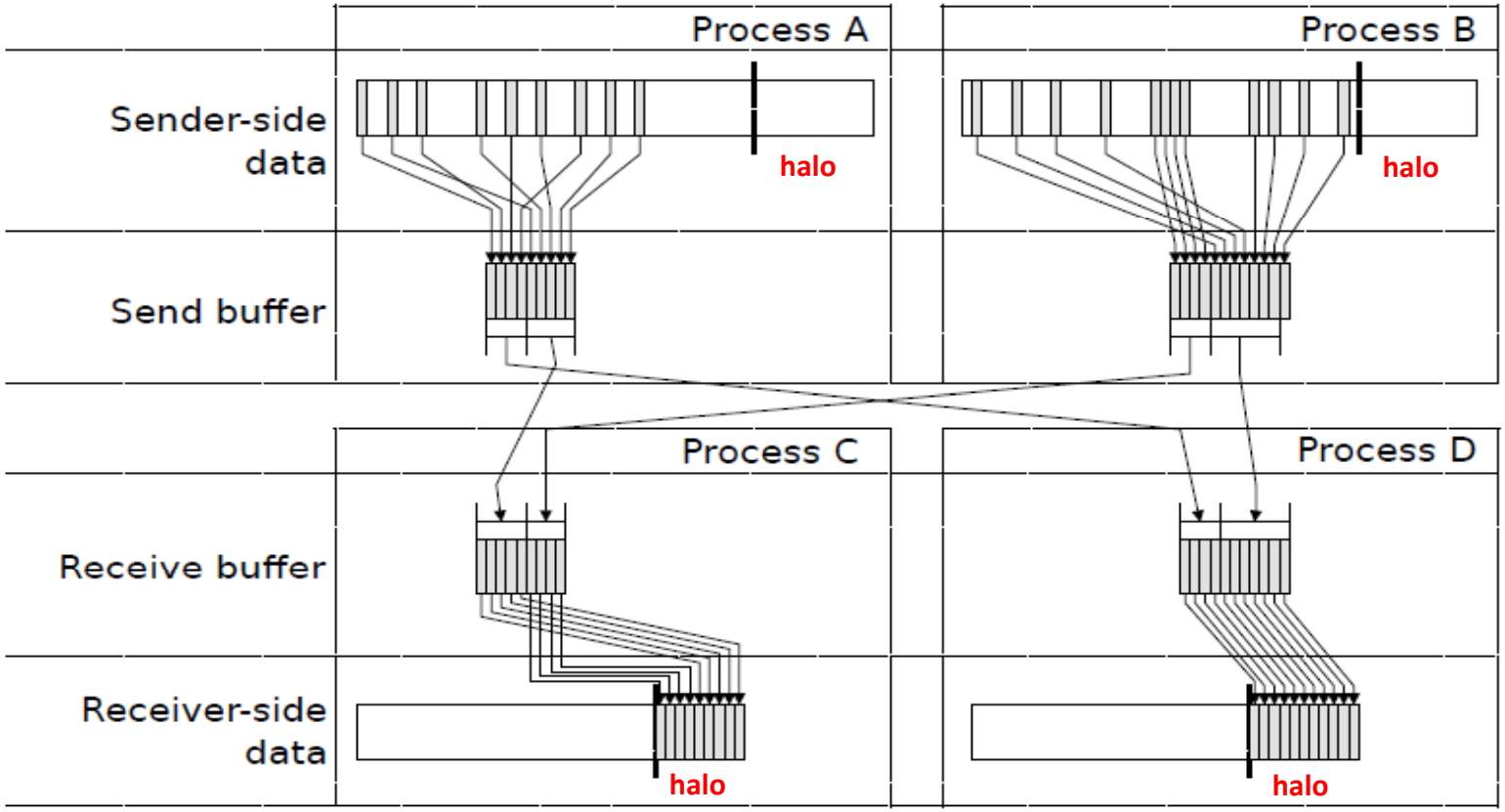

```
RECVbuf         0         1          5
RECVbuf         0         1         13
RECVbuf         0         1         21
RECVbuf         0         1         29
RECVbuf         0         2         33
RECVbuf         0         2         34
RECVbuf         0         2         35
RECVbuf         0         2         36

RECVbuf         1         0          4
RECVbuf         1         0         12
RECVbuf         1         0         20
RECVbuf         1         0         28
RECVbuf         1         3         37
RECVbuf         1         3         38
RECVbuf         1         3         39
RECVbuf         1         3         40

RECVbuf         2         3         37
RECVbuf         2         3         45
RECVbuf         2         3         53
RECVbuf         2         3         61
RECVbuf         2         0         25
RECVbuf         2         0         26
RECVbuf         2         0         27
RECVbuf         2         0         28

RECVbuf         3         2         36
RECVbuf         3         2         44
RECVbuf         3         2         52
RECVbuf         3         2         60
RECVbuf         3         1         29
RECVbuf         3         1         30
RECVbuf         3         1         31
RECVbuf         3         1         32
```

実行結果（PE#3）




```
RECVbuf         0         1          5
RECVbuf         0         1         13
RECVbuf         0         1         21
RECVbuf         0         1         29
RECVbuf         0         2         33
RECVbuf         0         2         34
RECVbuf         0         2         35
RECVbuf         0         2         36

RECVbuf         1         0          4
RECVbuf         1         0         12
RECVbuf         1         0         20
RECVbuf         1         0         28
RECVbuf         1         3         37
RECVbuf         1         3         38
RECVbuf         1         3         39
RECVbuf         1         3         40

RECVbuf         2         3         37
RECVbuf         2         3         45
RECVbuf         2         3         53
RECVbuf         2         3         61
RECVbuf         2         0         25
RECVbuf         2         0         26
RECVbuf         2         0         27
RECVbuf         2         0         28

RECVbuf         3         2         36
RECVbuf         3         2         44
RECVbuf         3         2         52
RECVbuf         3         2         60
RECVbuf         3         1         29
RECVbuf         3         1         30
RECVbuf         3         1         31
RECVbuf         3         1         32
```

Communication Pattern using 1D Structure



Dr. Osni Marques
（Lawrence Berkeley National Laboratory）より借用

初期全体メッシュ

演習

21	22	23	24	25
16	17	18	19	20
11	12	13	14	15
6	7	8	9	10
1	2	3	4	5

3領域に分割

演習

#PE2

21	22	23	24
16	17	18	19
11	12	13	14
6	7	8	

#PE1

23	24	25
18	19	20
13	14	15
8	9	10
	4	5

#PE0

11	12	13		
6	7	8	9	10
1	2	3	4	5

3領域に分割

演習

#PE2

7 21	8 22	9 23	15 24
4 16	5 17	6 18	14 19
1 11	2 12	3 13	13 14
10 6	11 7	12 8	

#PE1

14 23	7 24	8 25
13 18	5 19	6 20
12 13	3 14	4 15
11 8	1 9	2 10
	9 4	10 5

#PE0

11 11	12 12	13 13		
6 6	7 7	8 8	9 9	10 10
1 1	2 2	3 3	4 4	5 5

PE#0：局所分散データ（sqm.0）〇の部分をうめよ!

演習

#PE2

7 21	8 22	9 23	15 24
4 16	5 17	6 18	14 19
1 11	2 12	3 13	13 14
10 6	11 7	12 8	

#PE1

14 23	7 24	8 25
13 18	5 19	6 20
12 13	3 14	4 15
11 8	1 9	2 10
	9 4	10 5

#PE0

11 11	12 12	13 13		
6 6	7 7	8 8	9 9	10 10
1 1	2 2	3 3	4 4	5 5

```
#NEIBPEtot
    2
#NEIBPE
    1    2
#NODE
    〇   〇
#IMPORTindex
    〇   〇
#IMPORTitems
    〇…
#EXPORTindex
    〇   〇
#EXPORTitems
    〇…
```

PE#1：局所分散データ（sqm.1）
○の部分をうめよ!

演習

#PE2

7 21	8 22	9 23	15 24
4 16	5 17	6 18	14 19
1 11	2 12	3 13	13 14
10 6	11 7	12 8	

#PE1

14 23	7 24	8 25
13 18	5 19	6 20
12 13	3 14	4 15
11 8	1 9	2 10
	9 4	10 5

#PE0

11 11	12 12	13 13		
6 6	7 7	8 8	9 9	10 10
1 1	2 2	3 3	4 4	5 5

```
#NEIBPEtot
    2
#NEIBPE
    0    2
#NODE
    ○    ○
#IMPORTindex
    ○    ○
#IMPORTitems
    ○…
#EXPORTindex
    ○    ○
#EXPORTitems
    ○…
```

PE#2：局所分散データ（sqm.2）
○の部分をうめよ!

演習

#PE2

7 21	8 22	9 23	15 24
4 16	5 17	6 18	14 19
1 11	2 12	3 13	13 14
10 6	11 7	12 8	

#PE1

14 23	7 24	8 25
13 18	5 19	6 20
12 13	3 14	4 15
11 8	1 9	2 10
	9 4	10 5

#PE0

11 11	12 12	13 13		
6 6	7 7	8 8	9 9	10 10
1 1	2 2	3 3	4 4	5 5

```
#NEIBPEtot
    2
#NEIBPE
    1    0
#NODE
    ○    ○
#IMPORTindex
    ○    ○
#IMPORTitems
    ○…
#EXPORTindex
    ○    ○
#EXPORTitems
    ○…
```

演習

作業手順

```
>$ cd <$T-EPS>/P1/local/3pe
>$ cp ../4pe/a.out        先ほどsq-sr2.f/cをコンパイルしたもの

>$ ls
   a.out  sq.0  sq.1  sq.2

sqm.0,sqm.1,sqm.2を自分で作成する(手作業)

>$ qsub go.sh          とやって動作を確認する(3プロセス)
```

演習

作業用ディレクトリの中身

#PE2

7 21	8 22	9 23	15 24
4 16	5 17	6 18	14 19
1 11	2 12	3 13	13 14
10 6	11 7	12 8	

#PE1

14 23	7 24	8 25
13 18	5 19	6 20
12 13	3 14	4 15
11 8	1 9	2 10
	9 4	10 5

#PE0

11 11	12 12	13 13		
6 6	7 7	8 8	9 9	10 10
1 1	2 2	3 3	4 4	5 5

sq.0	sq.1	sq.2
1	9	11
2	10	12
3	14	13
4	15	16
5	19	17
6	20	18
7	24	21
8	25	22
		23

演習

#PE2

7 21	8 22	9 23	15 24
4 16	5 17	6 18	14 19
1 11	2 12	3 13	13 14
10 6	11 7	12 8	

#PE1

14 23	7 24	8 25
13 18	5 19	6 20
12 13	3 14	4 15
11 8	1 9	2 10
	9 4	10 5

#PE0

11 11	12 12	13 13		
6 6	7 7	8 8	9 9	10 10
1 1	2 2	3 3	4 4	5 5

演習

手 順

- 内点数，外点数
- 外点がどこから来ているか?
  - IMPORTindex，IMPORTitems
  - NEIBPEの順番
- それを逆にたどって，境界点の送信先を調べる
  - EXPORTindex，EXPORTitems
  - NEIBPEの順番

有限体積法(FVM)の並列計算向け局所分散データ構造

- 有限体積法(FVM)については領域間通信はこのような局所分散データによって実施可能
  - SPMD
  - 内点～外点の順に「局所」番号付け
  - 通信テーブル：一般化された通信テーブル
    - 有限要素法等疎行列を係数行列とするアプリケーションについては同様の手法で並列化可能

- 適切なデータ構造が定められれば，処理は非常に簡単。
  - 送信バッファに「境界点」の値を代入
  - 送信，受信
  - 受信バッファの値を「外点」の値として更新

- データ構造とアルゴリズム：局所分散データ
- FVMにおける並列計算と局所分散データ構造の考え方
- **領域分割手法について**
- eps_fvmにおける領域分割機能：eps_fvm_part
- eps_fvm「並列化」に向けて

領域分割機能：Partitioner

初期全体メッシュデータを与えることによって，
自動的に局所分散データを生成する

- 内点，外点
  - 局所分散メッシュデータ
  - 内点～外点となるように局所番号をつける
- 一般的な通信テーブル
  - 隣接領域情報
    - 隣接領域数
    - 隣接領域番号
  - 外点情報
    - どの領域から，何個の，どの外点の情報を「import」するか
  - 境界点情報
    - 何個の，どの境界点の情報を，どの領域に「export」するか

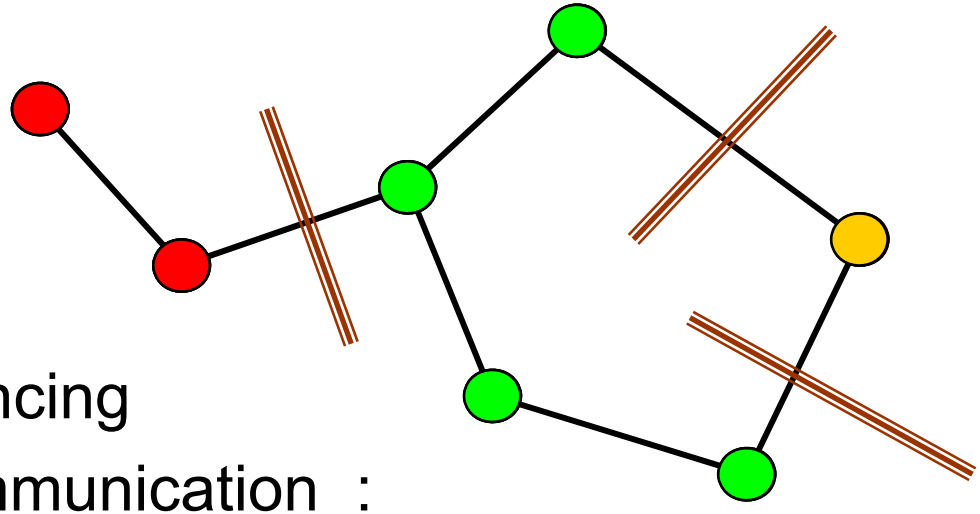
Partitioning とは？

- Graph/Graphic Partitioningの略
- 並列計算のための領域分割を実現するための手法
- 1PEでは計算できないような巨大な全体領域を局所データに分割する

Graph/Graphic Partitioning とは？

Graph/Graphic Partitioningとは「グラフ」（*graphs* ：節点と辺の集合）に関する「グラフ理論」を並列計算における領域分割に応用した手法である

- 一筆書き，四色問題

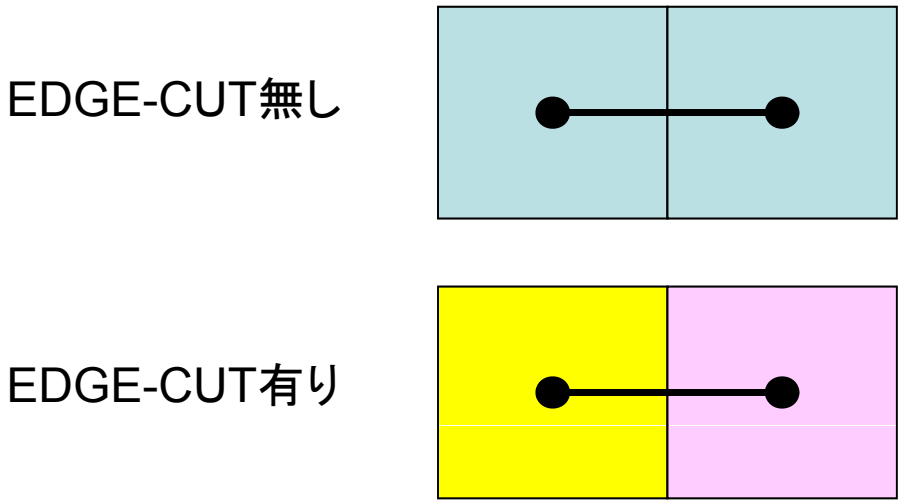


良い領域分割

- 領域間の負荷均等：Load balancing
- 領域間通信量最小：Small Communication ：
  - 前処理つき反復法の収束に影響
- 隣接領域数最小

EDGE-CUTとは？

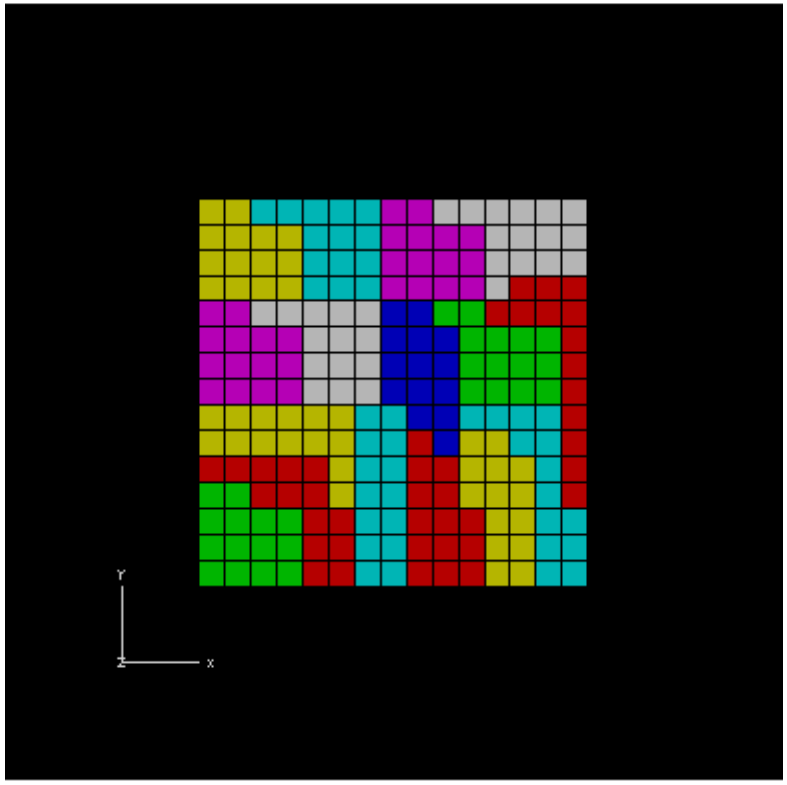
- 辺の両端の節点（または要素）が異なった領域に属している場合，「EDGE-CUTが生じている。」という。
- EDGE-CUTが少ないほど，通信は少ない。



Partitioning の反復法収束への影響

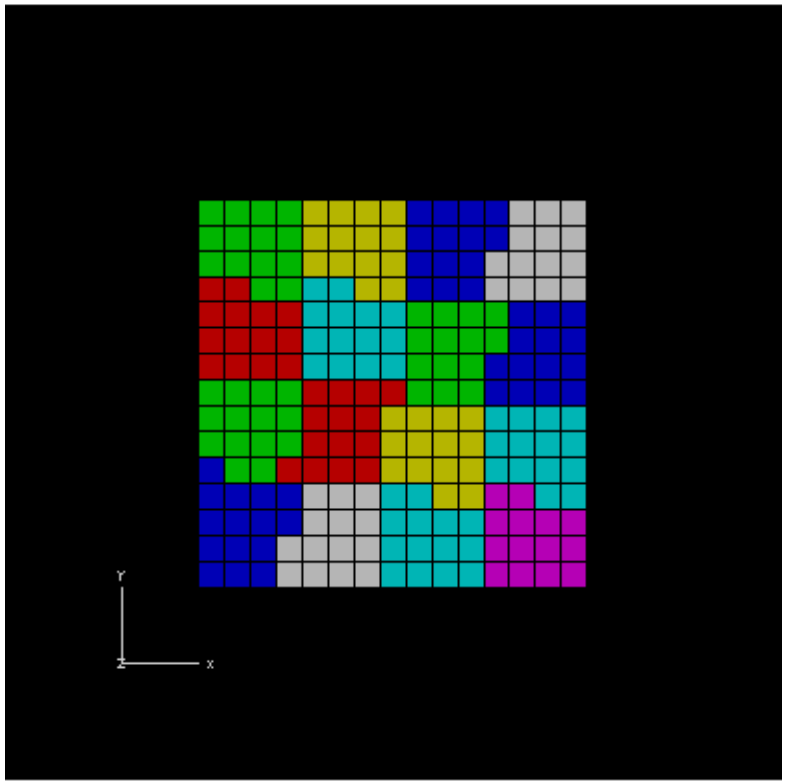
15×15領域を16分割：負荷バランスは取れている

Edge-Cut多い



RGB

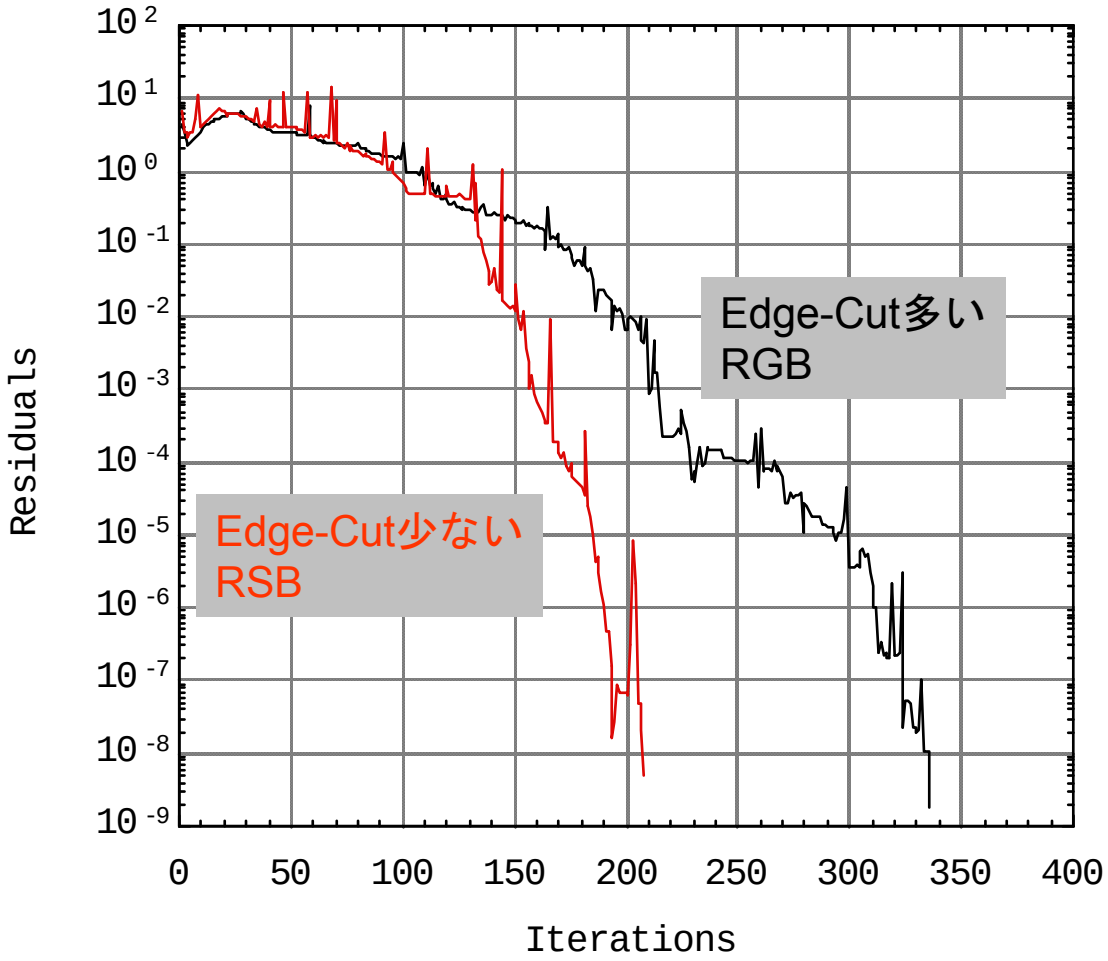
Edge-Cut少ない



RSB

Partitioning の反復法収束への影響

BiCGSTAB with Localized ILU(0) Preconditioning
15X15 region, RGB/RSB for 16 PE's , Poisson eqn's
Edge-Cutが少ないほど（通信が少ないほど）収束は速い



	RGB	RSB
Neighboring PEs (Ave., max)	3.63, 7	3.63, 6
Boundary Edges (Ave, max)	15.1, 19	12.5, 18

1996年2月頃
やった計算

Partitioning手法

- 数年前まで多くの研究グループがあったが今は，**METIS**（ミネソタ大学）と**JOSTLE**（グリニッジ大学）にほぼ集約

- **METIS：Univ.Minnesota**
  - http://glaros.dtc.umn.edu/gkhome/views/metis/
- **JOSTLE：Univ.Greenwich**
  - http://staffweb.cms.gre.ac.uk/~c.walshaw/jostle/
- **Scotch/PT-Scotch：比較的最近**
  - http://www.labri.fr/perso/pelegrin/scotch/

「eps_fvm」のPartitioningツール

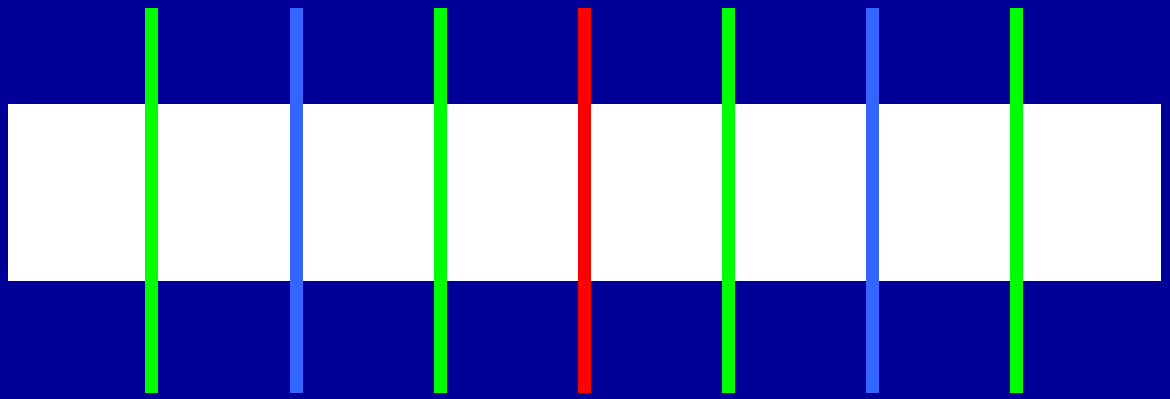
- 全体メッシュデータを対象とした簡易ツールを準備。
  - シリアル処理
- 全体メッシュデータを入力として，局所データ，通信情報を出力する。
- 分割手法
  - RCB （Recursive Coordinate Bisection）法
  - METIS
    - kmetis　　領域間通信最小（edge-cut最小）
    - pmetis　　領域間バランス最適化

RCB法

Recursive Coordinate Bisection

H.D.Simon ”Partitioning of unstructured problems for parallel processing”, Comp. Sys. in Eng., Vol.2, 1991.

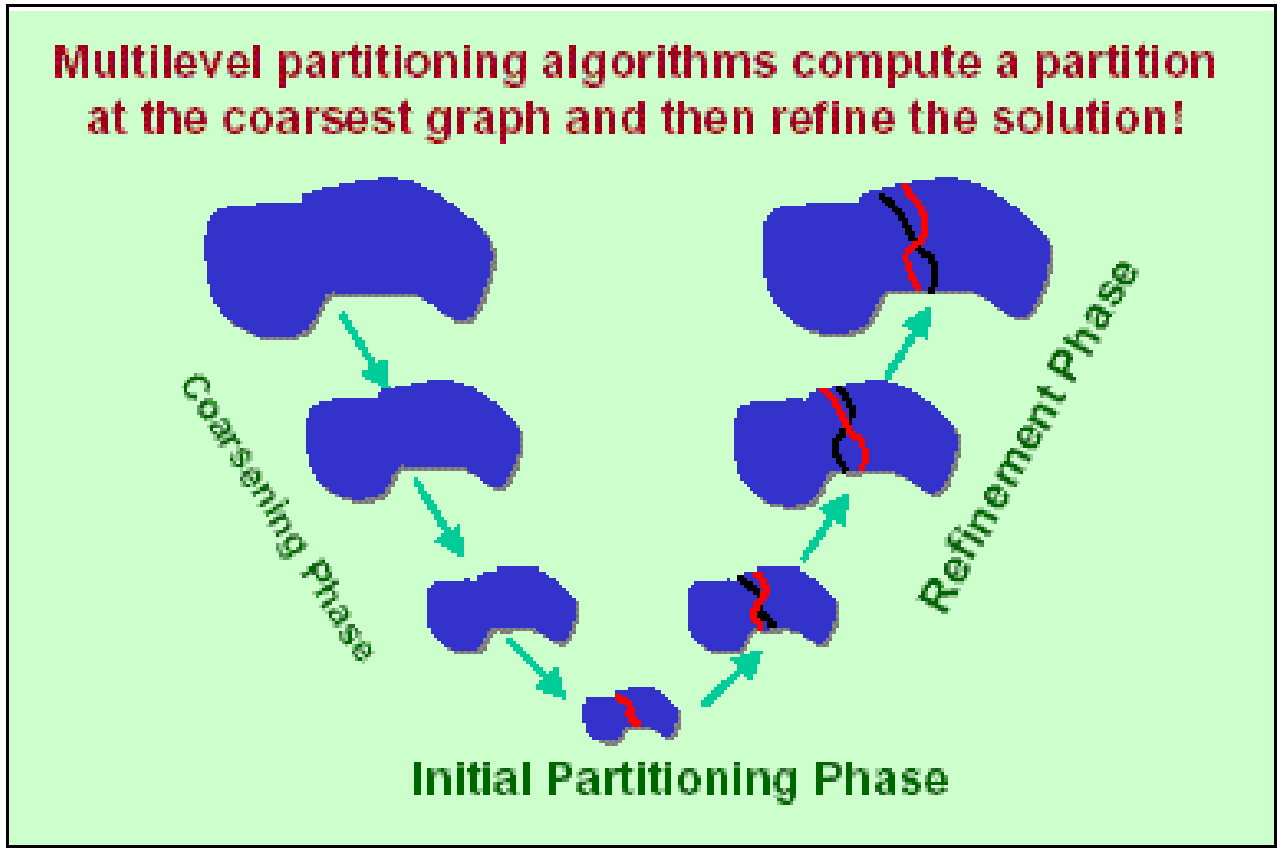
- XYZ座標成分の大小をとりながら分割
- 分割基準軸は形状に応じて任意に選択できる
- たとえば細長い形状では同じ方向への分割を続ける
- 2^n領域の分割しかできない
- 高速，簡易形状ではMETISより良い



METIS

http://glaros.dtc.umn.edu/gkhome/views/metis/

- マルチレベルグラフ理論に基づいた方法



METIS

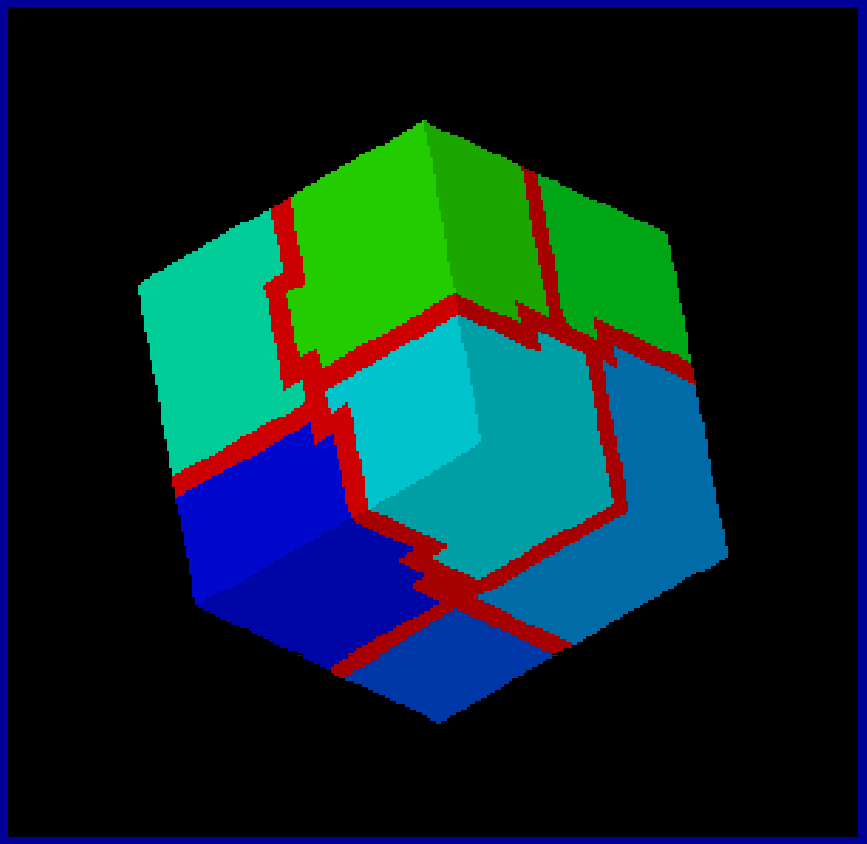
http://glaros.dtc.umn.edu/gkhome/views/metis/

- マルチレベルグラフ理論に基づいた方法
  - 特に通信（edge-cut）が少ない分割を提供する
  - 安定，高速
  - フリーウェア，他のプログラムに組み込むことも容易
- 色々な種類がある
  - k-METIS　　通信量（edge-cut）最小
  - p-METIS　　領域間バランス最適化
  - ParMETIS　　並列版
  - 領域分割だけでなく，オーダリング，データマイニングなど色々な分野に使用されている
    - 接触，衝突問題における並列接触面探索

領域分割例：立方体領域：8分割

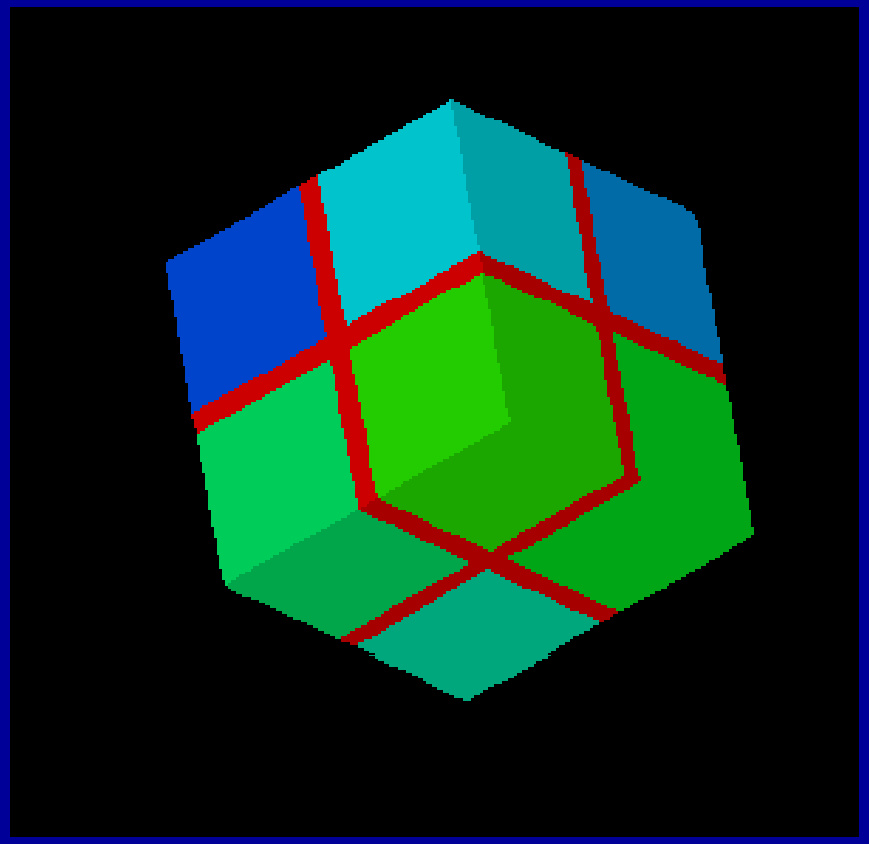
3,375要素（=15^3），4,096節点

単純な形状ではむしろRCBが良い



k-METIS

edgecut = 882



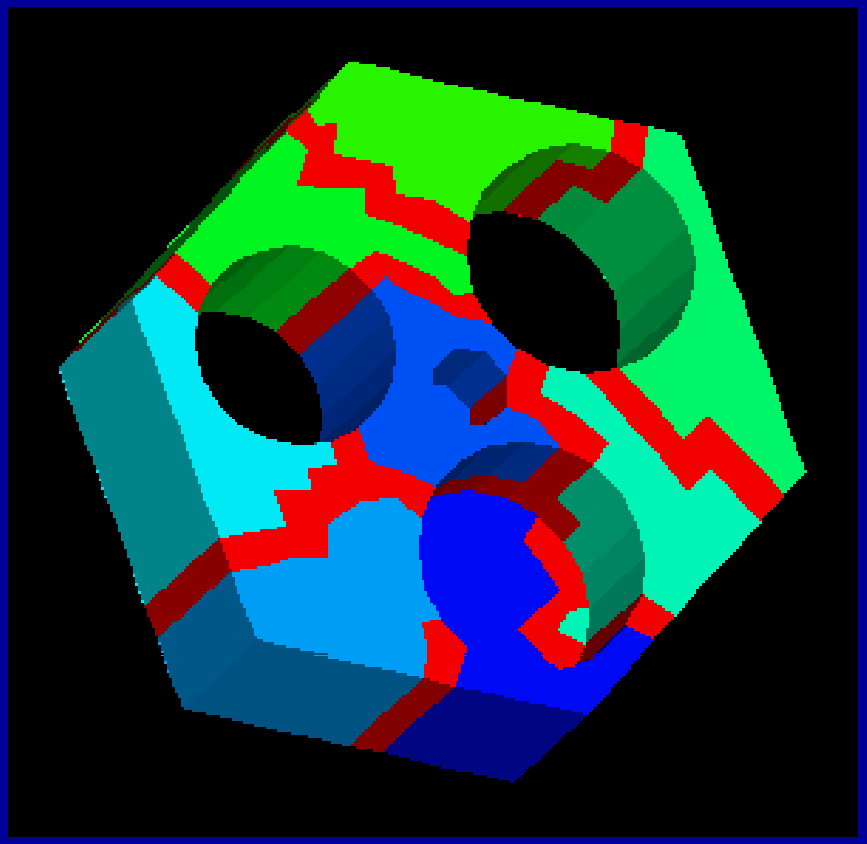
RCB

edgecut = 768

領域分割例：黒鉛ブロック：8分割

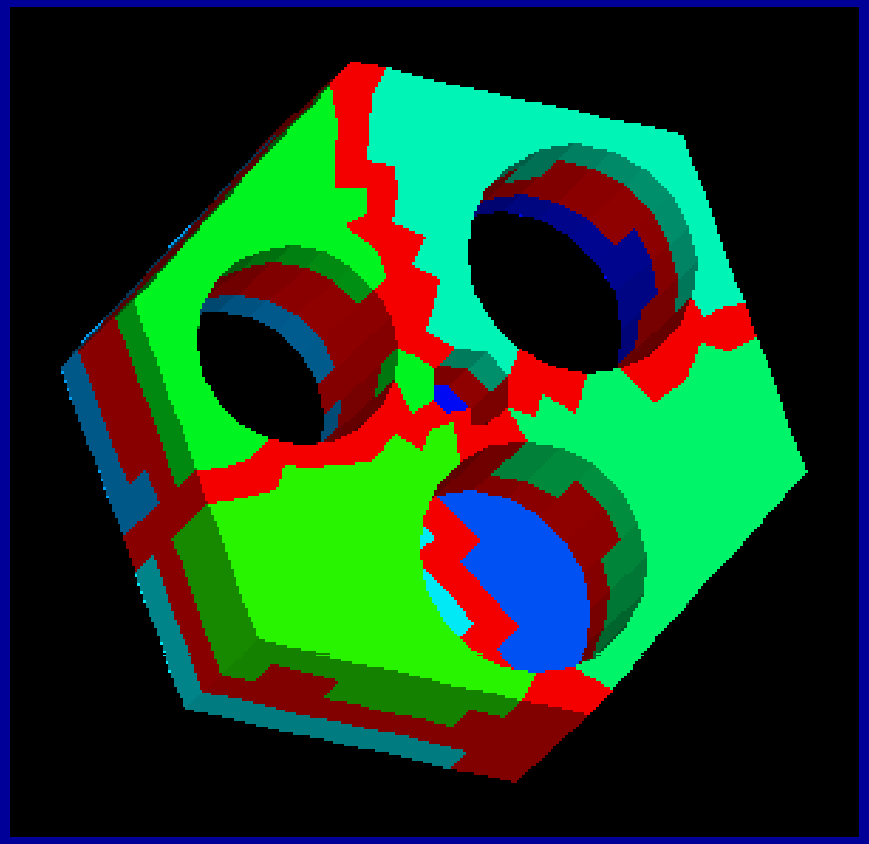
795要素，1,308節点

複雑形状ではMETISが良い：Overlap領域細い



k-METIS

edgecut = 307



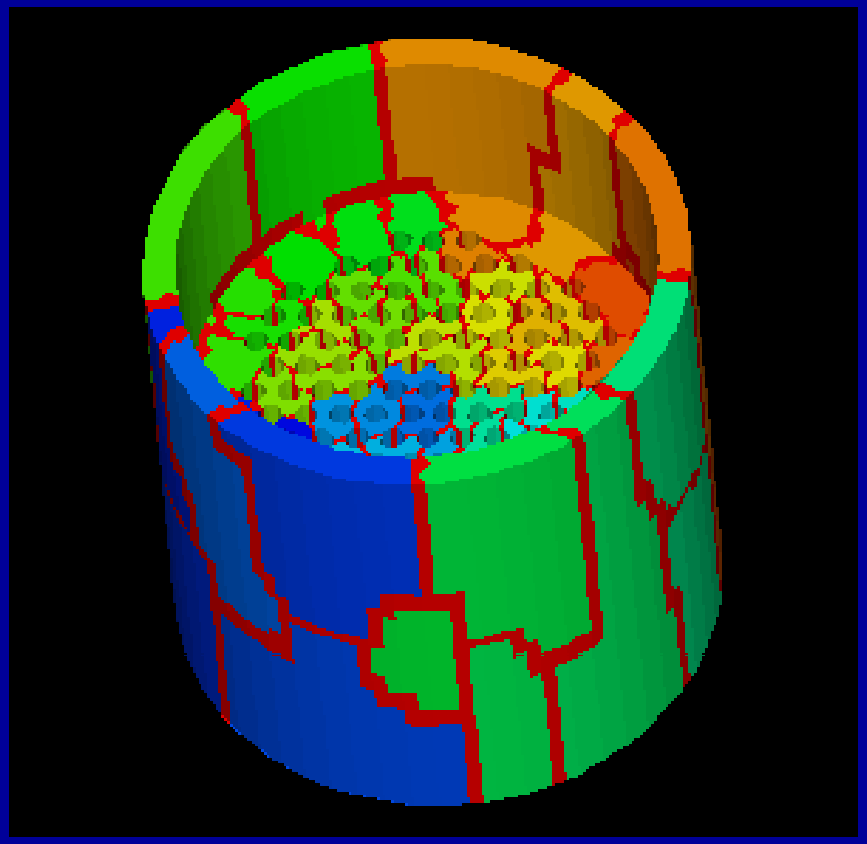
RCB

edgecut = 614

領域分割例：管板：64分割

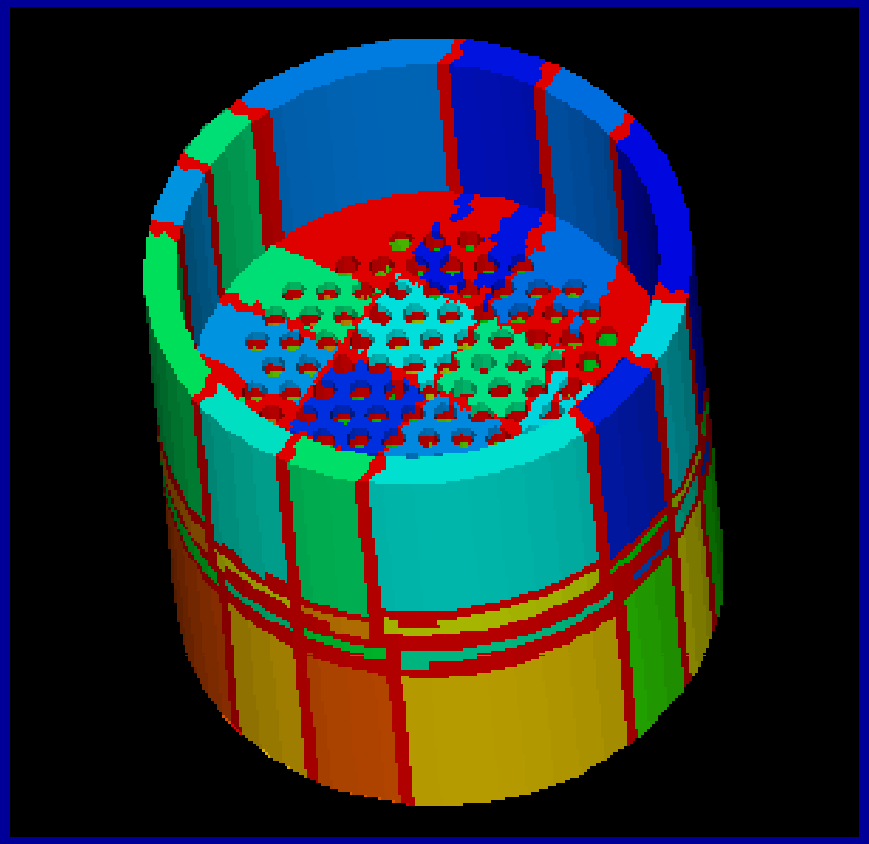
40,416要素，54,084節点

複雑形状ではMETISが良い：EdgeCut少ない



k-METIS

edgecut = 9,489



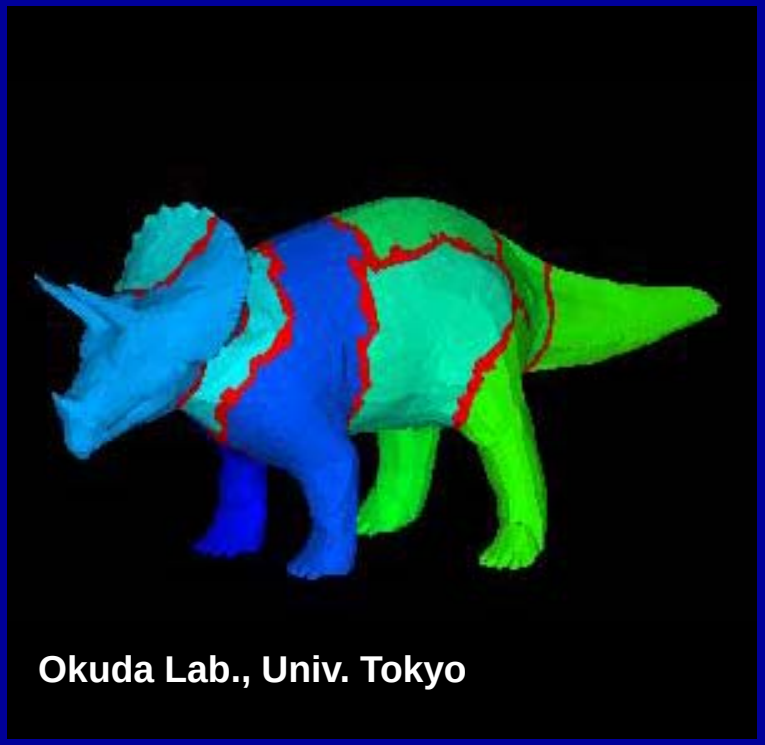
RCB

edgecut = 28,320

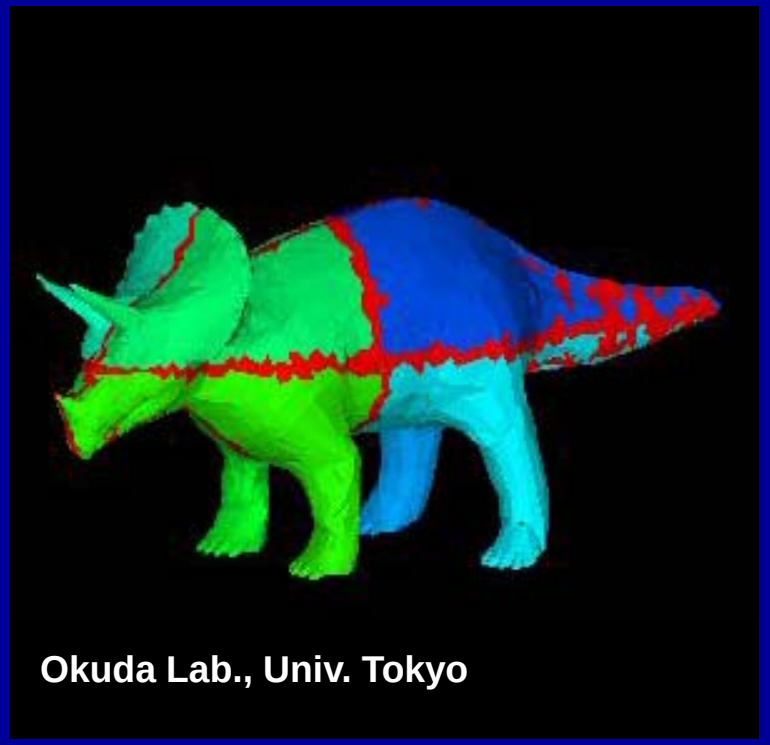
Strange Animal in 8 PEs

53,510 elements, 11,749 nodes.

METIS is better for complicated geometries.



k-METIS
edgecut = 4,573

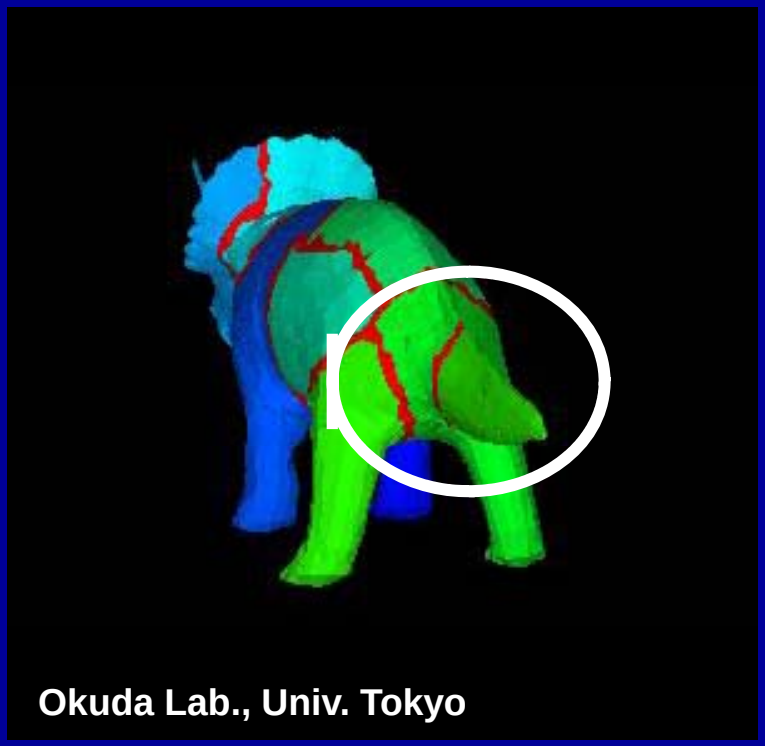


RCB
edgecut = 7,898

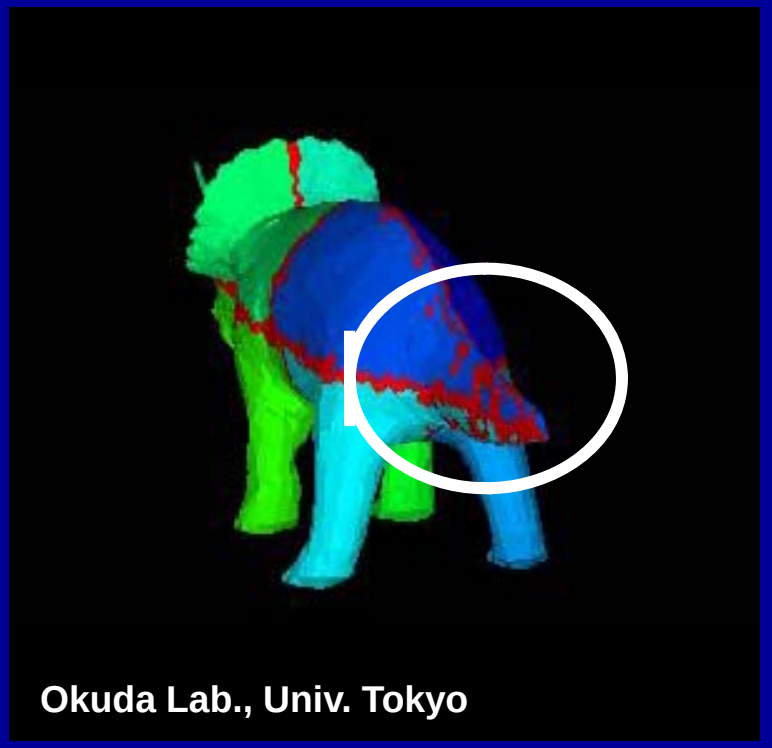
Strange Animal in 8 PEs

53,510 elements, 11,749 nodes.

METIS is better for complicated geometries.



k-METIS
edgecut = 4,573

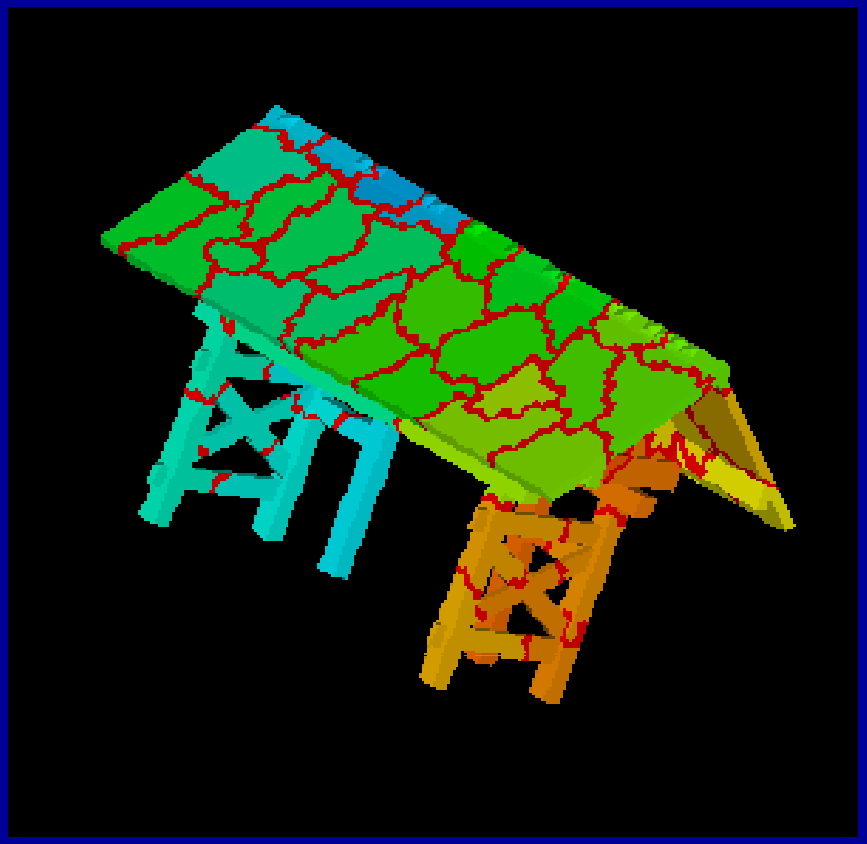


RCB
edgecut = 7,898

領域分割例：東大赤門：64分割

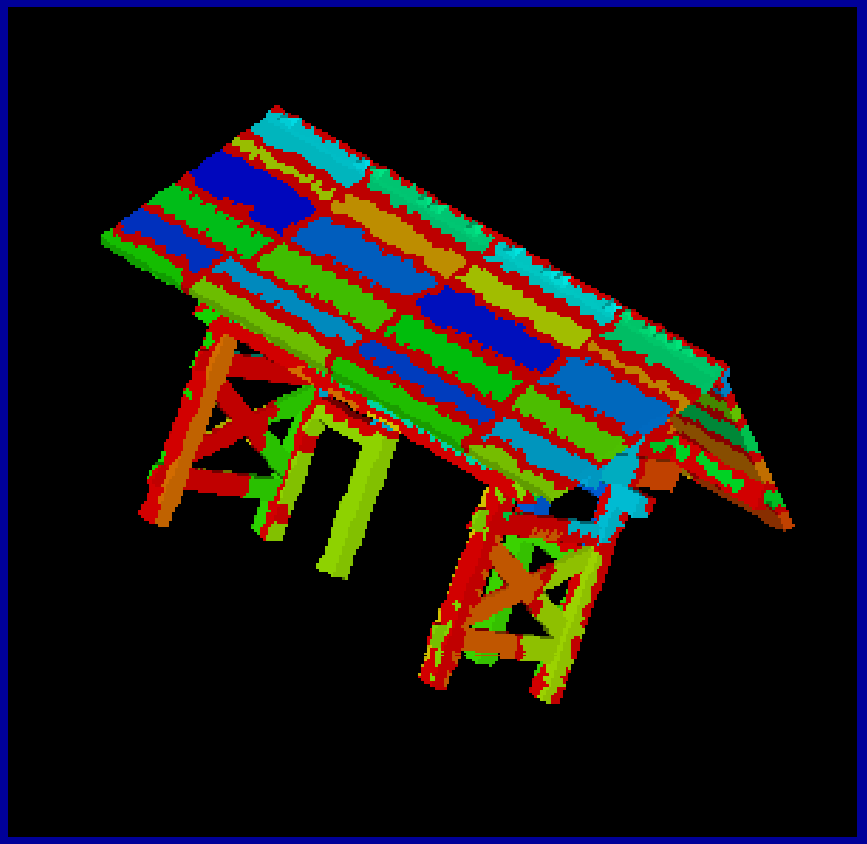
40,624要素，54,659節点

複雑形状ではMeTiSが良い：EdgeCut少ない



k-MeTiS

edgecut = 7,563



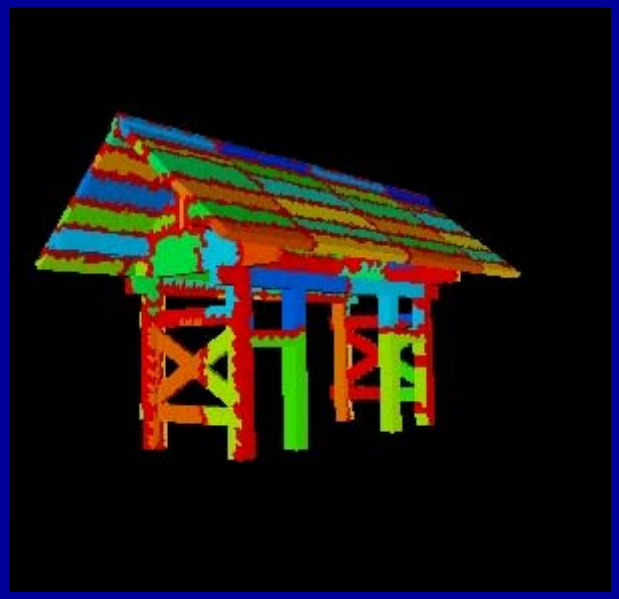
RCB

edgecut = 18,624

領域分割例：東大赤門：64分割

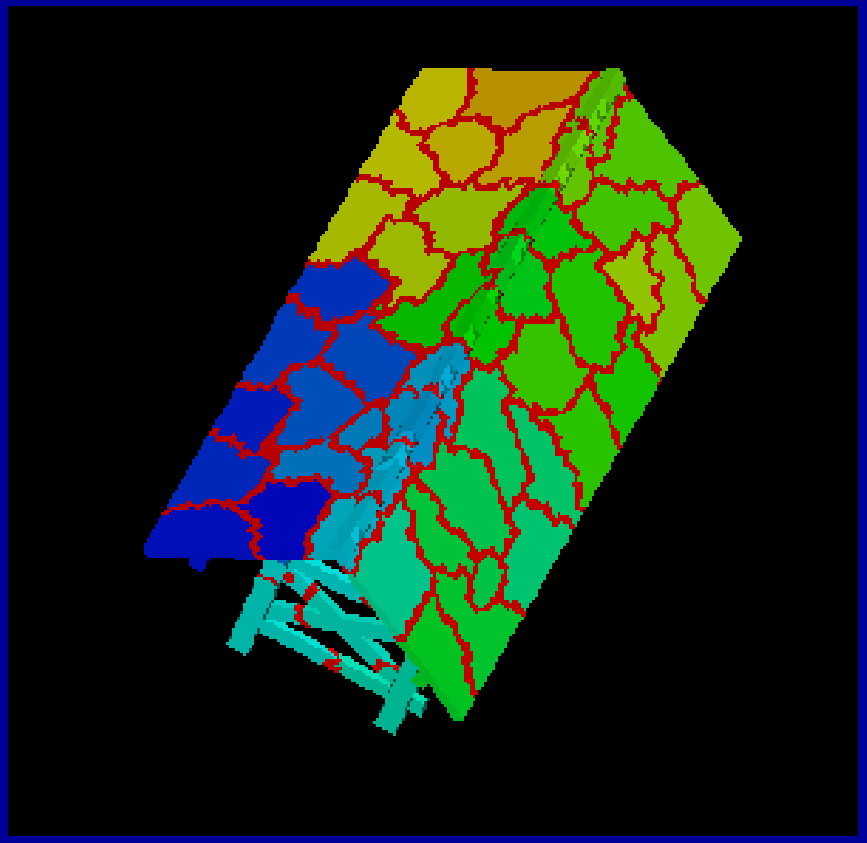
40,624要素，54,659節点

複雑形状ではMETISが良い：EdgeCut少ない



領域分割例：東大赤門：64分割

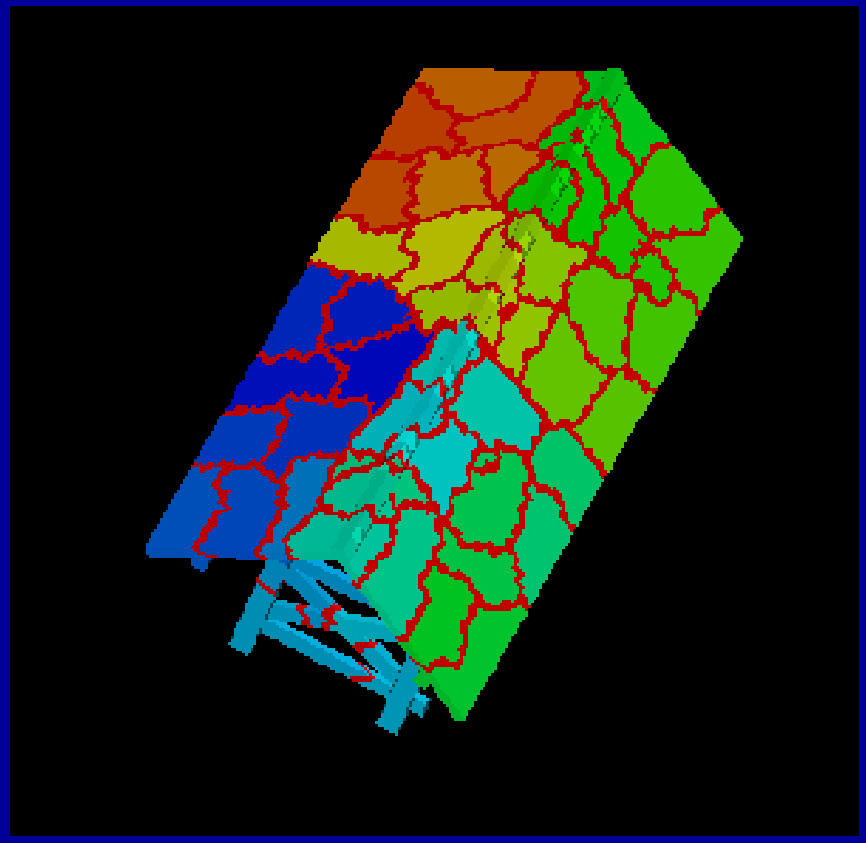
40,624要素，54,659節点



k-METIS

Load Balance= 1.03

edgecut = 7,563

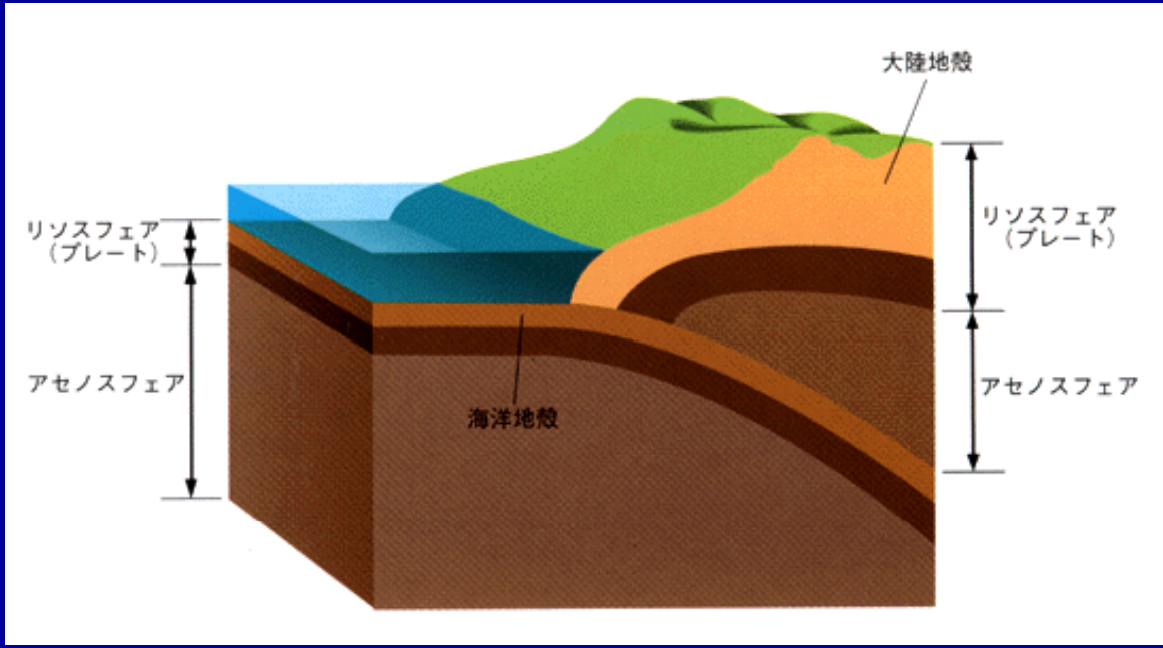


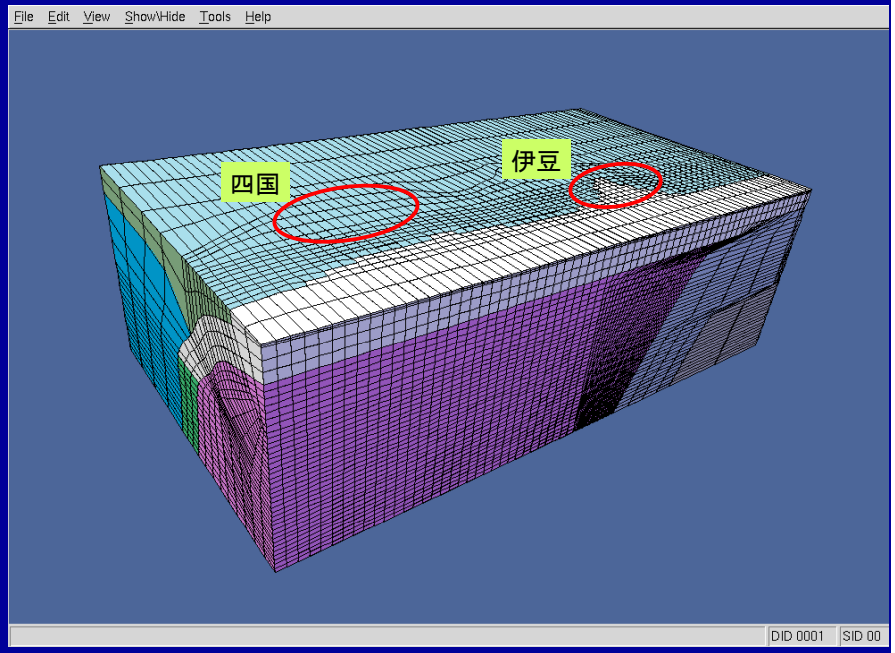
p-METIS

Load Balance= 1.00

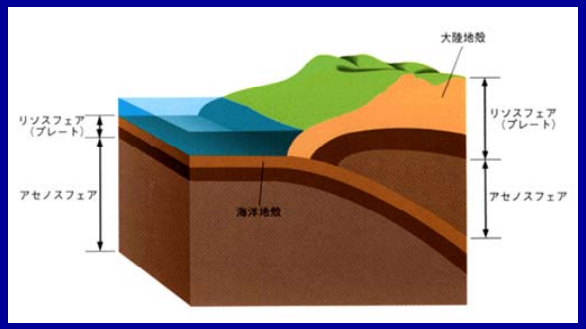
edgecut = 7,738

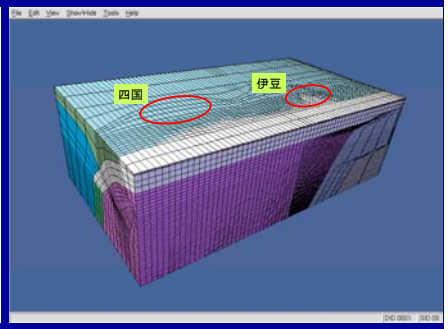
領域分割例：西南日本

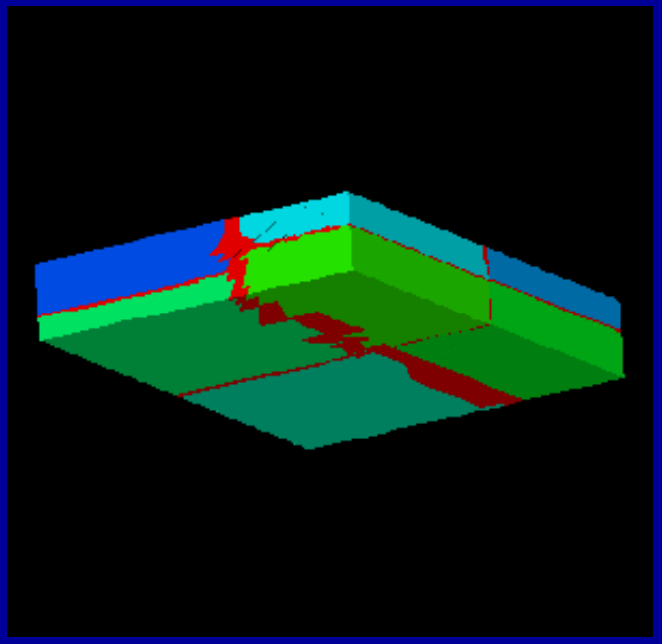

大陸地殻
リソスフェア
（プレート）
アセノスフェア
海洋地殻
リソスフェア
（プレート）
アセノスフェア


File Edit View Show/Hide Tools Help
四国
伊豆
DID 0001
SID 00

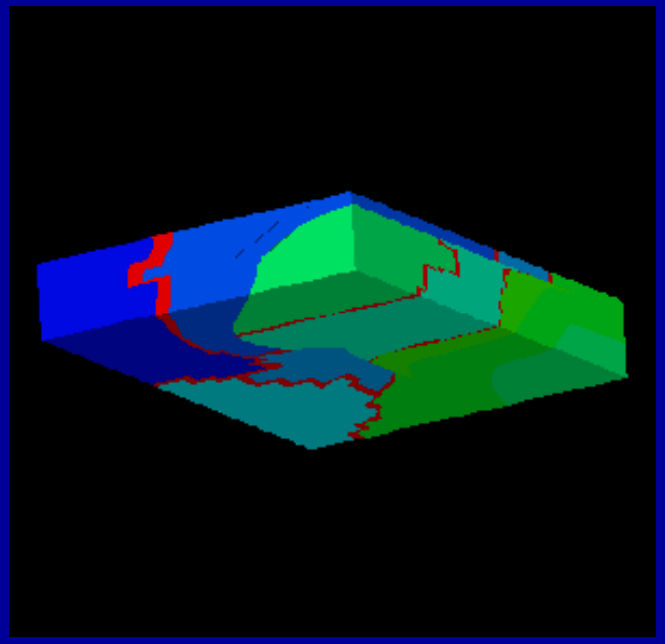
領域分割例：
西南日本：8分割
57,205要素，58,544節点



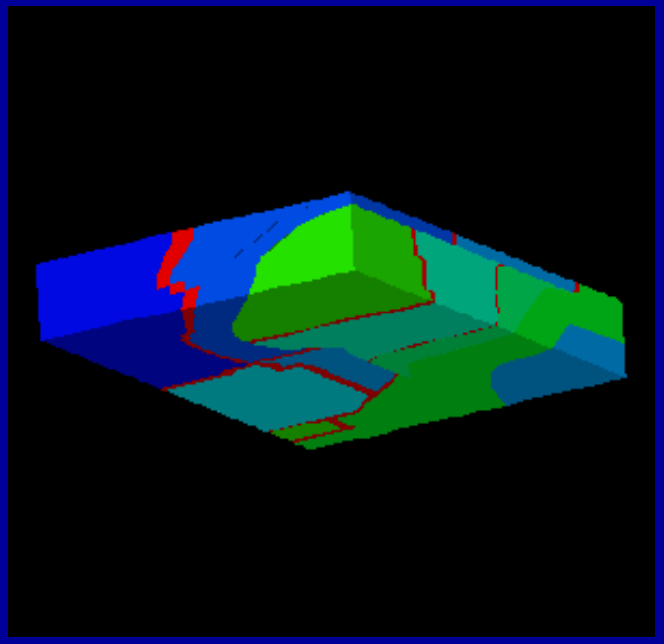




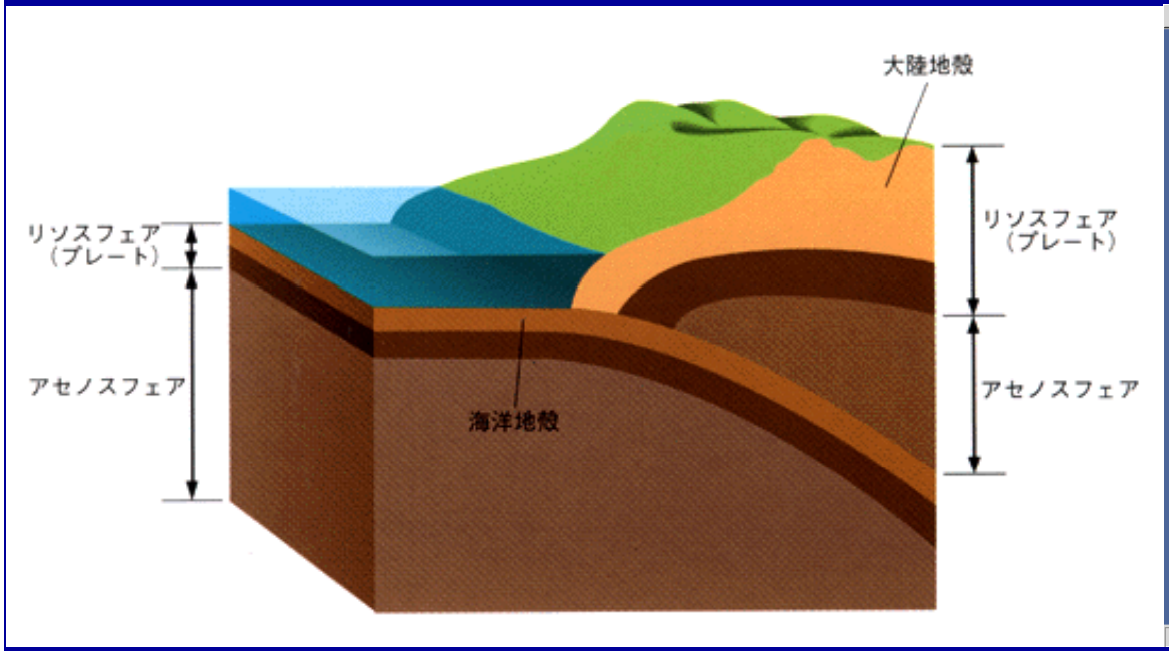
RCB e.c.=7433

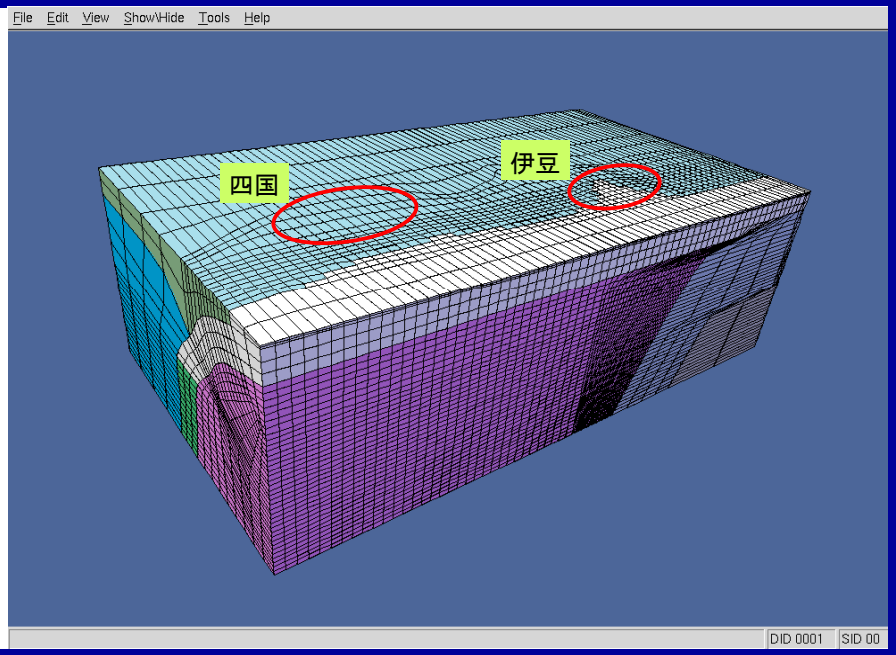


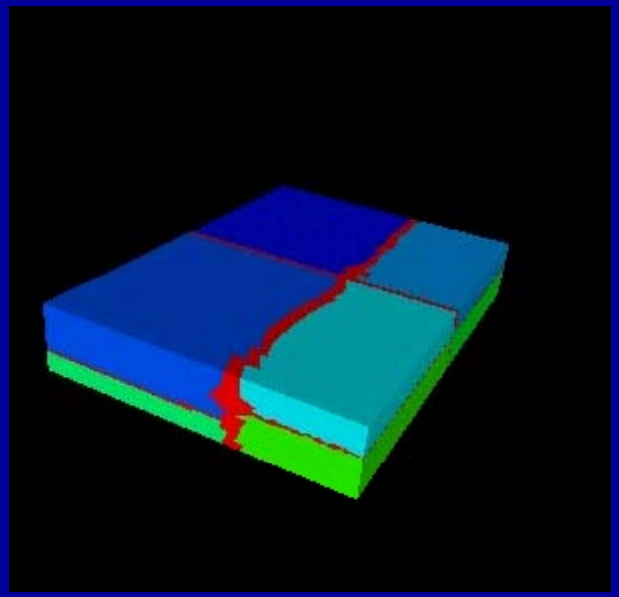
k-METIS :4,221



p-METIS :3,672


大陸地殻
リソスフェア
（プレート）
アセノスフェア
海洋地殻
リソスフェア
（プレート）
アセノスフェア


File Edit View Show\Hide Tools Help
四国
伊豆
DID 0001
SID 00



GeoFEM

- データ構造とアルゴリズム：局所分散データ
- FVMにおける並列計算と局所分散データ構造の考え方
- 領域分割手法について
- **eps_fvmにおける領域分割機能：eps_fvm_part**
- eps_fvm「並列化」に向けて

「eps_fvm」のPartitioningツール

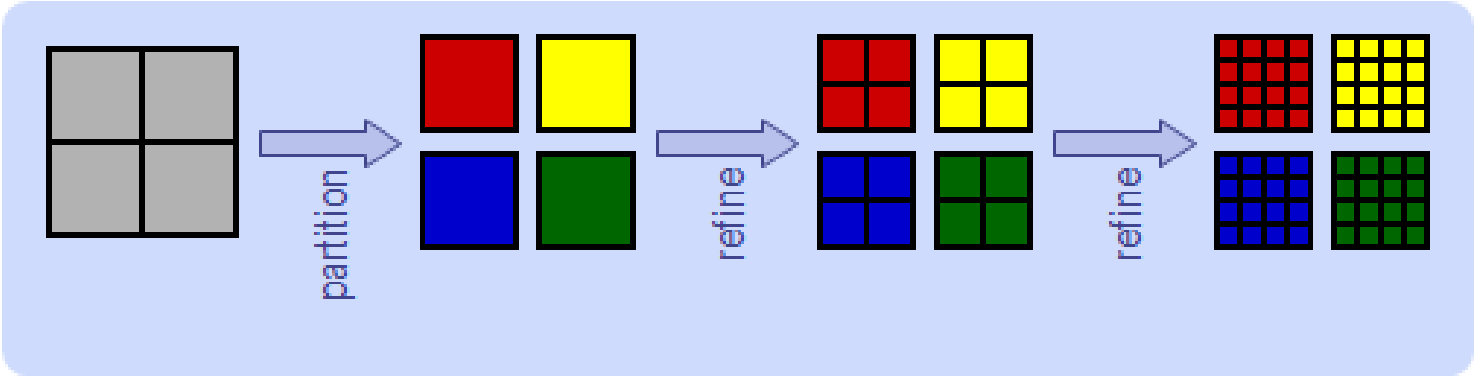
- 全体メッシュデータを対象とした簡易ツールを準備。
  - シリアル処理
- 全体メッシュデータを入力として，局所分散メッシュデータ，局所分散通信ファイルを別々のファイルとして出力。
- 分割手法
  - RCB （Recursive Coordinate Bisection）法
  - METIS
    - kmetis　　領域間通信最小（edge-cut最小）
    - pmetis　　領域間バランス最適化

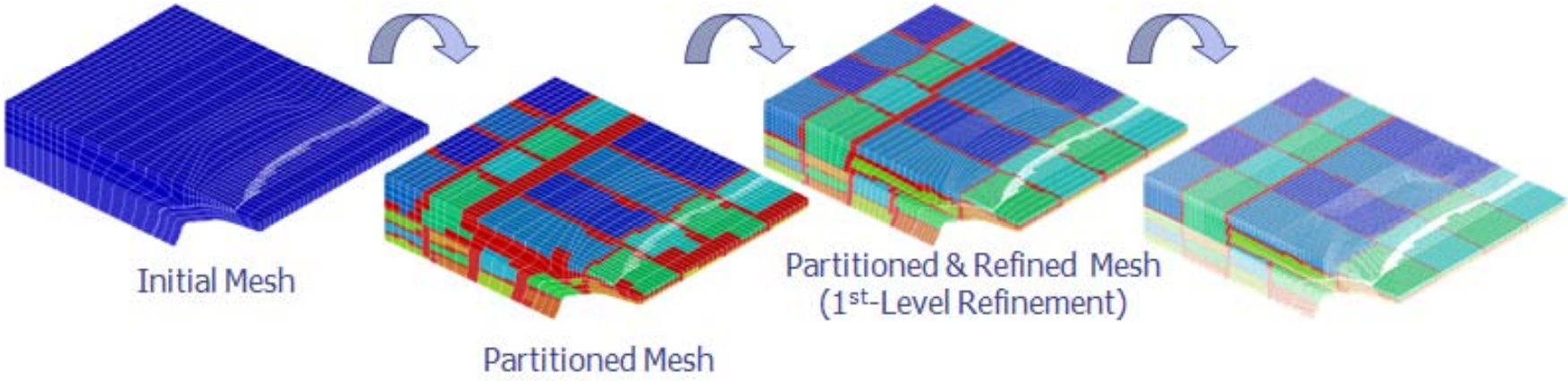
「eps_fvm」の領域分割の考え方

- 1領域= 1 PE（Processing Element）
  - ハードウェア的プロセスを意味しない
  - 領域番号は0（ゼロ）から始まる：MPIの都合
- 要素単位の領域分割
- 局所分散データ（メッシュ，通信）
  - 要素情報，要素間コネクティビティ
    - 局所番号
  - 境界条件関連
  - 通信テーブル

実際の大規模計算

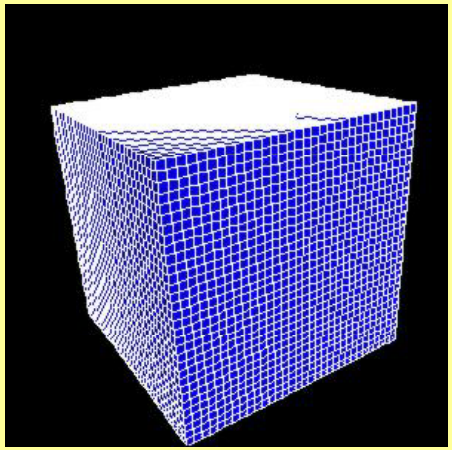
- そもそも「初期全体メッシュ」を単一ファイルとして用意できない場合もある。
- 「粗い」初期メッシュ→分割→整合性をとりながら局所的に細分化，という方式が適用されることが多い



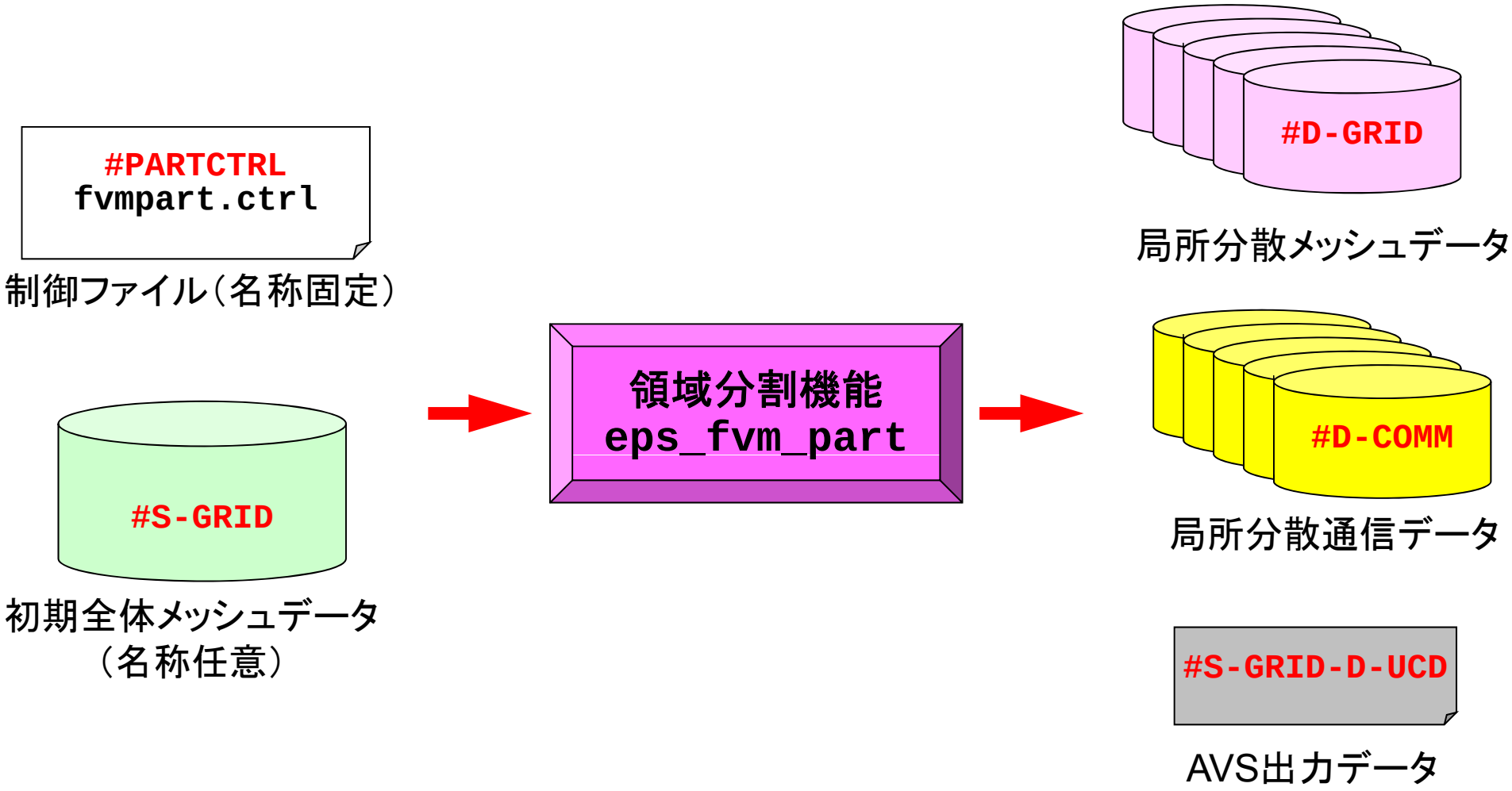


まずやってみましょう: ECCS2008

```
$> cd <$E-EPS>                                  各自作成したディレクトリ
$> cp /home03/skengon/Documents/class/EPSsummer/F/test-mesh.tar .
$> tar xvf test-mesh.tar
   <$E-P1>の下に “P1/test-mesh” というディレクトリができる

$> cd <$E-EPS>/P1/test-mesh
$> cd ex1

$> cat fvmmg.ctrl
    32 32 32
$> eps_fvm_mg                                   メッシュ生成
$> ls fvm_entire*
   fvm_entire_mesh.dat
   fvm_entire_mesh.inp
   fvm_entire_mesh.inp_geo
$> ls fvmpart.ctrl          領域分割制御用ファイル  #PARTCTRL
   fvmpart.ctrl
$> eps_fvm_part                                 領域分割 ！

32個×2のファイルが作成されます
```



領域分割機能（専用ツール）と ファイル入出力



3種類のファイルが出力される

- 局所分散メッシュデータ(#D-GRID)
  - 基本的なファイルの形式は初期全体メッシュデータと変わらない
  - 各プロセッサで扱う情報のみ細切れにされている
  - 局所番号

- 局所分散通信データ(#D-COMM)
  - 一般的な通信テーブルに基づくデータについて記述されている
  - 並列計算に特有なデータ

- AVS出力データ(#S-GRID-D-UCD)
  - 領域ごとに色分けされたメッシュデータ(全体データ)

制御ファイル “fvmpart.ctrl”について（1/4）

```
!INITIAL FILE
fvm_entire_mesh.dat

!METHOD
RCB
X,Y,Z,X,Y

!REGION NUMBER
32

!MESH FILE
mesh.rcb

!COMMUNICATION FILE
comm.rcb

!UCD
rcb-32-xyz.inp
```

- 名称は「fvmpart.ctrl」に固定。
- 「#」，または「!!」から開始される行はコメント行とみなされる。
- 以下の6つのブロック（”!”から始まる）とそれに続くパラメータから構成される。ブロック間の空行数は任意であり，記述する順番も任意である。

制御ファイル “fvmpart.ctrl”について(2/4)

```
!INITIAL FILE
fvm_entire_mesh.dat

!METHOD
RCB
X,Y,Z,X,Y

!REGION NUMBER
32

!MESH FILE
mesh.rcb

!COMMUNICATION FILE
comm.rcb

!UCD
rcb-32-xyz.inp
```

- **!INITIAL FILE**
  - 初期全体メッシュファイル名称(相対パス)。
  - パラメータ
    - 初期全体メッシュファイル #S-GRID 名称
  - このブロックは必須である。
  - #S-GRIDの名称は任意である。
- **!METHOD**
  - 分割手法指定のためのヘッダ名。
  - パラメータ
    - 手法名(RCB, KMETIS, PMETIS)
      - 必ず大文字で記入する
    - 手法として「RCB」を選択した場合は次の行に分割適用軸(X,Y,Z)を大文字+「,」で記入。
  - このブロックは必須である。

制御ファイル “fvmpart.ctrl”について(3/4)

```
!INITIAL FILE
fvm_entire_mesh.dat

!METHOD
RCB
X,Y,Z,X,Y

!REGION NUMBER
32

!MESH FILE
mesh.rcb

!COMMUNICATION FILE
comm.rcb

!UCD
rcb-32-xyz.inp
```

- **!REGION NUMBER**
  - 領域数。
  - パラメータ
    - 領域数(自然数)
  - このブロックは必須である。
  - 「!METHOD」として「RCB」を選択した場合は，2のべき乗としなければならない。
- **!MESH FILE**
  - 局所分散メッシュデータのヘッダ名(相対パス)。
  - パラメータ
    - 局所分散メッシュデータ #D-GRIDのヘッダ名
  - このブロックは必須である。
  - 分散メッシュデータは「header.領域番号」として生成される。領域番号は0から開始。

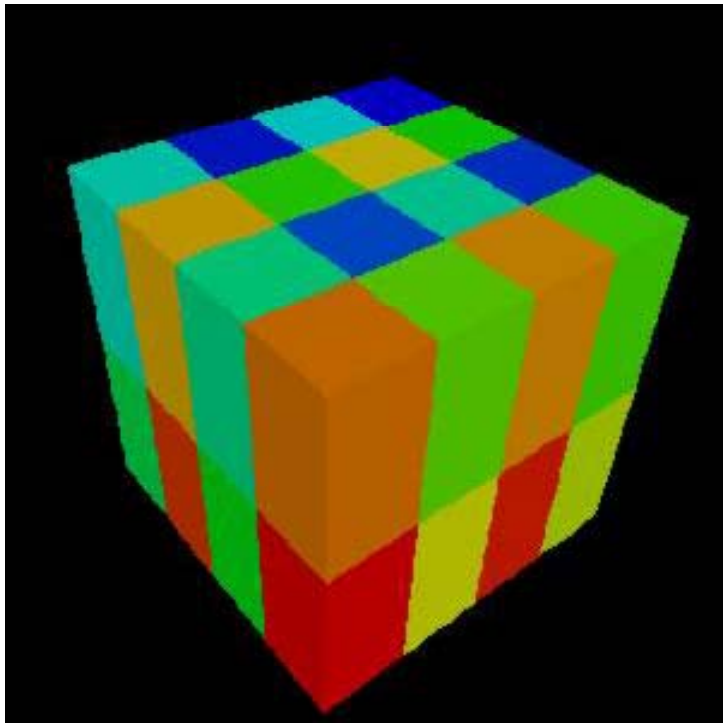
制御ファイル “fvmpart.ctrl”について(4/4)

```
!INITIAL FILE
fvm_entire_mesh.dat

!METHOD
RCB
X,Y,Z,X,Y

!REGION NUMBER
32

!MESH FILE
mesh.rcb

!COMMUNICATION FILE
comm.rcb

!UCD
rcb-32-xyz.inp
```

- **!COMMUNICATION FILE**
  - 局所分散通信データのヘッダ名(相対パス)。
  - パラメータ
    - 局所分散通信データ #D-COMMのヘッダ名
  - このブロックは必須である。
  - 分散メッシュデータは「header.領域番号」として生成される。領域番号は0から開始。
- **!UCD**
  - 領域分割の色分を表示するUCDファイル名(相対パス)。
  - パラメータ
    - UCDファイル名(拡張子として必ず「.inp」をつけること)
  - このブロックは省略可能である。

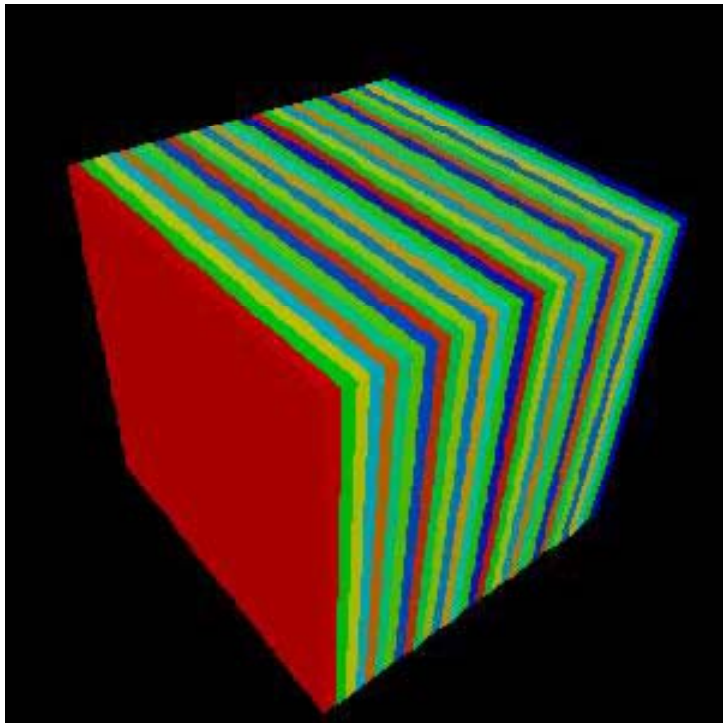
実例：RCB-1

```
!INITIAL FILE
fvm_entire_mesh.dat

!METHOD
RCB
X,Y,Z,X,Y

!REGION NUMBER
32

!MESH FILE
mesh.rcb

!COMMUNICATION FILE
comm.rcb

!UCD
rcb-32-xyz.inp
```



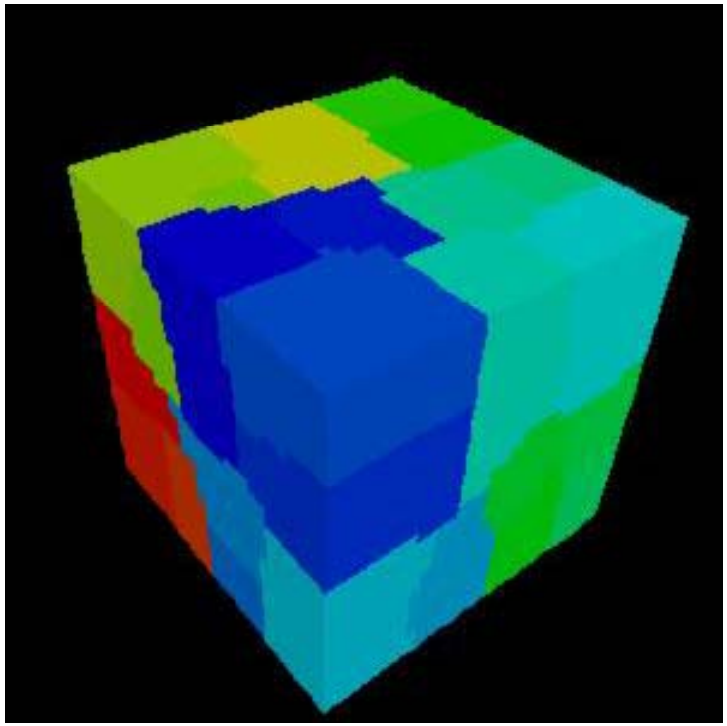
実例：RCB-2

```
!INITIAL FILE
fvm_entire_mesh.dat

!METHOD
RCB
X,X,X,X,X

!REGION NUMBER
32

!MESH FILE
mesh.rcb2

!COMMUNICATION FILE
comm.rcb2

!UCD
rcb-32-xxx.inp
```



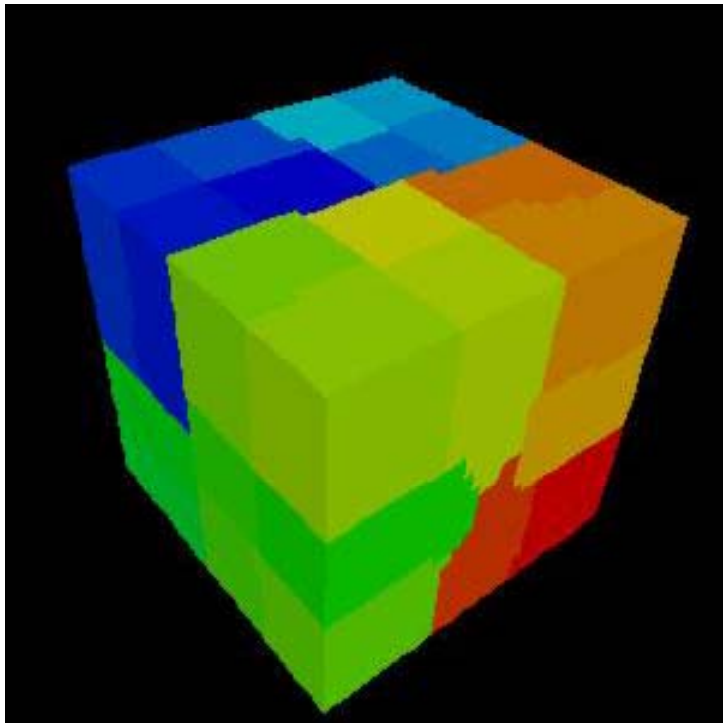
実例：kmetis

```
!INITIAL FILE
fvm_entire_mesh.dat

!METHOD
KMETIS

!REGION NUMBER
32

!MESH FILE
mesh.kmetis

!COMMUNICATION FILE
comm.kmetis

!UCD
kmetis-32.inp
```



実例：pmetis

```
!INITIAL FILE
fvm_entire_mesh.dat

!METHOD
PMETIS

!REGION NUMBER
32

!MESH FILE
mesh.pmetis

!COMMUNICATION FILE
comm.pmetis

!UCD
pmetis-32.inp
```



次はファイルの中身を見てみよう

```
$> cd <$T-EPS>/P1/test-mesh
$> cd ex2

$> cat fvmpart.ctrl                    領域分割制御用ファイル
```

```
!INITIAL FILE
2d.mesh
!METHOD
RCB
Y,X
!REGION NUMBER
4
!MESH FILE
mesh
!COMMUNICATION FILE
comm
!UCD
rcb-4.inp
```

```
$> eps_fvm_part                        領域分割 ！
$> ls mesh.*                           局所分散メッシュファイル
   mesh.0  mesh.1  mesh.2  mesh.3
$> ls comm.*                           局所分散通信ファイル
   comm.0  comm.1  comm.2  comm.3
```

局所分散データファイルのチュートリアル

http://nkl.cc.u-tokyo.ac.jp/tutorial/part_tutorial/

http://nkl.cc.u-tokyo.ac.jp/tutorial/part_tutorial.tar

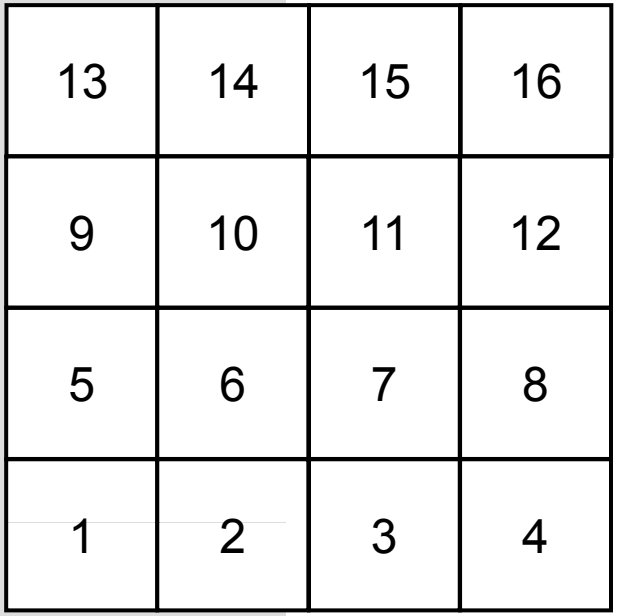
初期全体メッシュファイル（1/2）（2d.mesh）

要素

```
16
 1    1.000000E+00    1.000000E+00    5.000000E-01    5.000000E-01    5.000000E-01
 2    1.000000E+00    1.000000E+00    1.500000E+00    5.000000E-01    5.000000E-01
 3    1.000000E+00    1.000000E+00    2.500000E+00    5.000000E-01    5.000000E-01
 4    1.000000E+00    1.000000E+00    3.500000E+00    5.000000E-01    5.000000E-01
 5    1.000000E+00    1.000000E+00    5.000000E-01    1.500000E+00    5.000000E-01
 6    1.000000E+00    1.000000E+00    1.500000E+00    1.500000E+00    5.000000E-01
 7    1.000000E+00    1.000000E+00    2.500000E+00    1.500000E+00    5.000000E-01
 8    1.000000E+00    1.000000E+00    3.500000E+00    1.500000E+00    5.000000E-01
 9    1.000000E+00    1.000000E+00    5.000000E-01    2.500000E+00    5.000000E-01
10    1.000000E+00    1.000000E+00    1.500000E+00    2.500000E+00    5.000000E-01
11    1.000000E+00    1.000000E+00    2.500000E+00    2.500000E+00    5.000000E-01
12    1.000000E+00    1.000000E+00    3.500000E+00    2.500000E+00    5.000000E-01
13    1.000000E+00    1.000000E+00    5.000000E-01    3.500000E+00    5.000000E-01
14    1.000000E+00    1.000000E+00    1.500000E+00    3.500000E+00    5.000000E-01
15    1.000000E+00    1.000000E+00    2.500000E+00    3.500000E+00    5.000000E-01
16    1.000000E+00    1.000000E+00    3.500000E+00    3.500000E+00    5.000000E-01
```

コネクティビティ

```
24
 1        2    1.000000E+00    5.000000E-01    5.000000E-01
 1        5    1.000000E+00    5.000000E-01    5.000000E-01
 2        3    1.000000E+00    5.000000E-01    5.000000E-01
 2        6    1.000000E+00    5.000000E-01    5.000000E-01
 3        4    1.000000E+00    5.000000E-01    5.000000E-01
 3        7    1.000000E+00    5.000000E-01    5.000000E-01
 4        8    1.000000E+00    5.000000E-01    5.000000E-01
 5        6    1.000000E+00    5.000000E-01    5.000000E-01
 5        9    1.000000E+00    5.000000E-01    5.000000E-01
 6        7    1.000000E+00    5.000000E-01    5.000000E-01
 6       10    1.000000E+00    5.000000E-01    5.000000E-01
 7        8    1.000000E+00    5.000000E-01    5.000000E-01
 7       11    1.000000E+00    5.000000E-01    5.000000E-01
 8       12    1.000000E+00    5.000000E-01    5.000000E-01
 9       10    1.000000E+00    5.000000E-01    5.000000E-01
 9       13    1.000000E+00    5.000000E-01    5.000000E-01
10       11    1.000000E+00    5.000000E-01    5.000000E-01
10       14    1.000000E+00    5.000000E-01    5.000000E-01
11       12    1.000000E+00    5.000000E-01    5.000000E-01
11       15    1.000000E+00    5.000000E-01    5.000000E-01
12       16    1.000000E+00    5.000000E-01    5.000000E-01
13       14    1.000000E+00    5.000000E-01    5.000000E-01
14       15    1.000000E+00    5.000000E-01    5.000000E-01
15       16    1.000000E+00    5.000000E-01    5.000000E-01
```



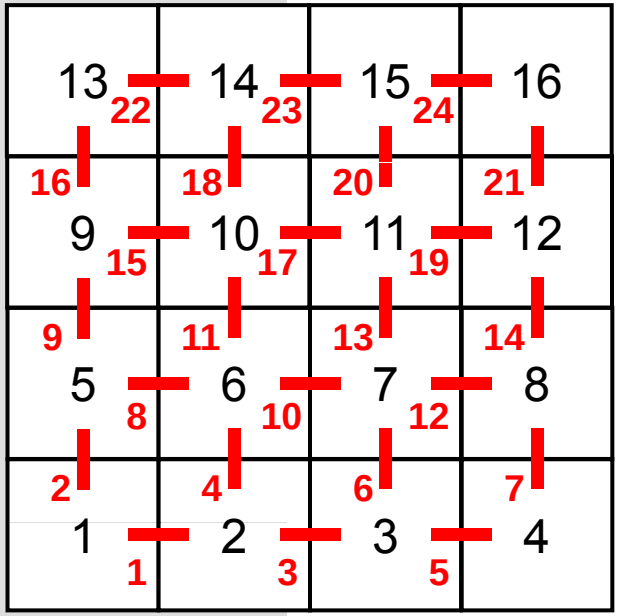
初期全体メッシュファイル（1/2）（2d.mesh）

要素

```
16
 1    1.000000E+00    1.000000E+00    5.000000E-01    5.000000E-01    5.000000E-01
 2    1.000000E+00    1.000000E+00    1.500000E+00    5.000000E-01    5.000000E-01
 3    1.000000E+00    1.000000E+00    2.500000E+00    5.000000E-01    5.000000E-01
 4    1.000000E+00    1.000000E+00    3.500000E+00    5.000000E-01    5.000000E-01
 5    1.000000E+00    1.000000E+00    5.000000E-01    1.500000E+00    5.000000E-01
 6    1.000000E+00    1.000000E+00    1.500000E+00    1.500000E+00    5.000000E-01
 7    1.000000E+00    1.000000E+00    2.500000E+00    1.500000E+00    5.000000E-01
 8    1.000000E+00    1.000000E+00    3.500000E+00    1.500000E+00    5.000000E-01
 9    1.000000E+00    1.000000E+00    5.000000E-01    2.500000E+00    5.000000E-01
10    1.000000E+00    1.000000E+00    1.500000E+00    2.500000E+00    5.000000E-01
11    1.000000E+00    1.000000E+00    2.500000E+00    2.500000E+00    5.000000E-01
12    1.000000E+00    1.000000E+00    3.500000E+00    2.500000E+00    5.000000E-01
13    1.000000E+00    1.000000E+00    5.000000E-01    3.500000E+00    5.000000E-01
14    1.000000E+00    1.000000E+00    1.500000E+00    3.500000E+00    5.000000E-01
15    1.000000E+00    1.000000E+00    2.500000E+00    3.500000E+00    5.000000E-01
16    1.000000E+00    1.000000E+00    3.500000E+00    3.500000E+00    5.000000E-01
```

コネクティビティ

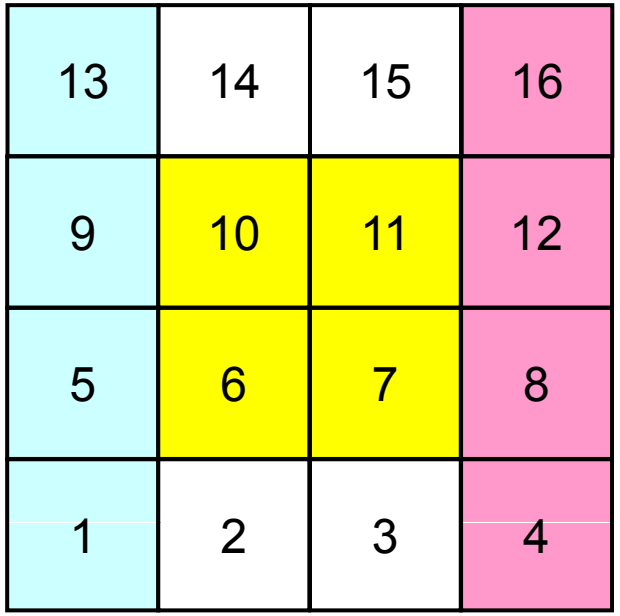
```
24
 1      2    1.000000E+00    5.000000E-01    5.000000E-01    1
 1      5    1.000000E+00    5.000000E-01    5.000000E-01    2
 2      3    1.000000E+00    5.000000E-01    5.000000E-01    3
 2      6    1.000000E+00    5.000000E-01    5.000000E-01    4
 3      4    1.000000E+00    5.000000E-01    5.000000E-01    5
 3      7    1.000000E+00    5.000000E-01    5.000000E-01    6
 4      8    1.000000E+00    5.000000E-01    5.000000E-01    7
 5      6    1.000000E+00    5.000000E-01    5.000000E-01    8
 5      9    1.000000E+00    5.000000E-01    5.000000E-01    9
 6      7    1.000000E+00    5.000000E-01    5.000000E-01   10
 6     10    1.000000E+00    5.000000E-01    5.000000E-01   11
 7      8    1.000000E+00    5.000000E-01    5.000000E-01   12
 7     11    1.000000E+00    5.000000E-01    5.000000E-01   13
 8     12    1.000000E+00    5.000000E-01    5.000000E-01   14
 9     10    1.000000E+00    5.000000E-01    5.000000E-01   15
 9     13    1.000000E+00    5.000000E-01    5.000000E-01   16
10     11    1.000000E+00    5.000000E-01    5.000000E-01   17
10     14    1.000000E+00    5.000000E-01    5.000000E-01   18
11     12    1.000000E+00    5.000000E-01    5.000000E-01   19
11     15    1.000000E+00    5.000000E-01    5.000000E-01   20
12     16    1.000000E+00    5.000000E-01    5.000000E-01   21
13     14    1.000000E+00    5.000000E-01    5.000000E-01   22
14     15    1.000000E+00    5.000000E-01    5.000000E-01   23
15     16    1.000000E+00    5.000000E-01    5.000000E-01   24
```



初期全体メッシュファイル(2/2)(2d.mesh)

ディリクレ
ノイマン
体積発熱

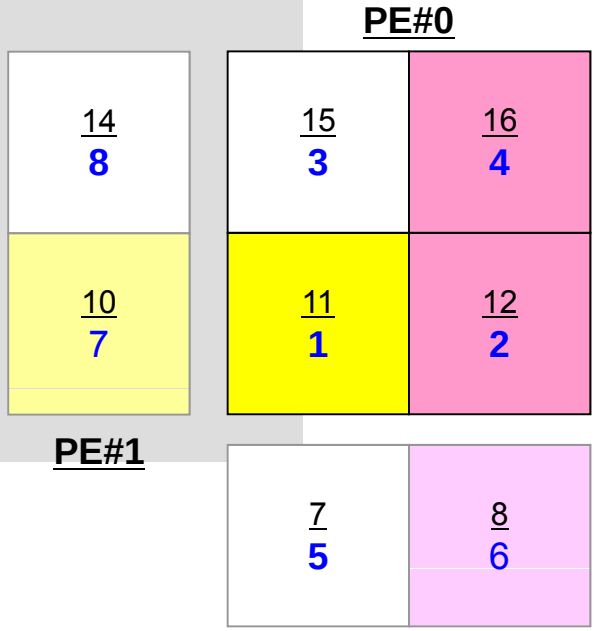
```
4
 4    1.000000E+00    5.000000E-01    0.000000E+00
 8    1.000000E+00    5.000000E-01    0.000000E+00
12    1.000000E+00    5.000000E-01    0.000000E+00
16    1.000000E+00    5.000000E-01    0.000000E+00
 4
 1    1.000000E+00    1.000000E+00
 5    1.000000E+00    1.000000E+00
 9    1.000000E+00    1.000000E+00
13    1.000000E+00    1.000000E+00
 4
 6    1.000000E+00
 7    1.000000E+00
10    1.000000E+00
11    1.000000E+00
```



局所分散メッシュファイル（mesh.0）

要素
コネクティビティ
境界条件

```
8
1    1.000000E+00    1.000000E+00    2.500000E+00    2.500000E+00    5.000000E-01
2    1.000000E+00    1.000000E+00    3.500000E+00    2.500000E+00    5.000000E-01
3    1.000000E+00    1.000000E+00    2.500000E+00    3.500000E+00    5.000000E-01
4    1.000000E+00    1.000000E+00    3.500000E+00    3.500000E+00    5.000000E-01
5    1.000000E+00    1.000000E+00    2.500000E+00    1.500000E+00    5.000000E-01
6    1.000000E+00    1.000000E+00    3.500000E+00    1.500000E+00    5.000000E-01
7    1.000000E+00    1.000000E+00    1.500000E+00    2.500000E+00    5.000000E-01
8    1.000000E+00    1.000000E+00    1.500000E+00    3.500000E+00    5.000000E-01
8
5        1    1.000000E+00    5.000000E-01    5.000000E-01
6        2    1.000000E+00    5.000000E-01    5.000000E-01
7        1    1.000000E+00    5.000000E-01    5.000000E-01
1        2    1.000000E+00    5.000000E-01    5.000000E-01
1        3    1.000000E+00    5.000000E-01    5.000000E-01
2        4    1.000000E+00    5.000000E-01    5.000000E-01
8        3    1.000000E+00    5.000000E-01    5.000000E-01
3        4    1.000000E+00    5.000000E-01    5.000000E-01
2
2    1.000000E+00    5.000000E-01    0.000000E+00
4    1.000000E+00    5.000000E-01    0.000000E+00
0
1
1    1.000000E+00
```

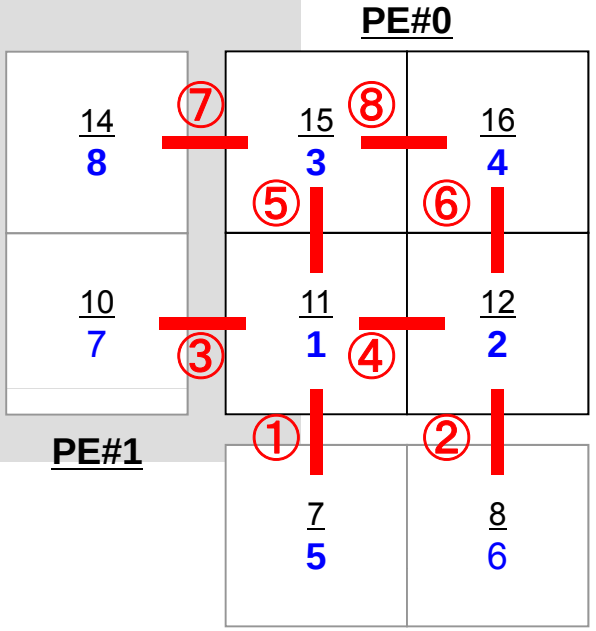


基本的に初期全体メッシュファイルと同じ
局所要素番号による記述
境界条件（ディリクレ，ノイマン，体積発熱）：「内点」のみの情報

局所分散メッシュファイル（mesh.0）

要素 / コネクティビティ / 境界条件

```
8
1    1.000000E+00    1.000000E+00    2.500000E+00    2.500000E+00    5.000000E-01
2    1.000000E+00    1.000000E+00    3.500000E+00    2.500000E+00    5.000000E-01
3    1.000000E+00    1.000000E+00    2.500000E+00    3.500000E+00    5.000000E-01
4    1.000000E+00    1.000000E+00    3.500000E+00    3.500000E+00    5.000000E-01
5    1.000000E+00    1.000000E+00    2.500000E+00    1.500000E+00    5.000000E-01
6    1.000000E+00    1.000000E+00    3.500000E+00    1.500000E+00    5.000000E-01
7    1.000000E+00    1.000000E+00    1.500000E+00    2.500000E+00    5.000000E-01
8    1.000000E+00    1.000000E+00    1.500000E+00    3.500000E+00    5.000000E-01
8
5        1    1.000000E+00    5.000000E-01    5.000000E-01  ①
6        2    1.000000E+00    5.000000E-01    5.000000E-01  ②
7        1    1.000000E+00    5.000000E-01    5.000000E-01  ③
1        2    1.000000E+00    5.000000E-01    5.000000E-01  ④
1        3    1.000000E+00    5.000000E-01    5.000000E-01  ⑤
2        4    1.000000E+00    5.000000E-01    5.000000E-01  ⑥
8        3    1.000000E+00    5.000000E-01    5.000000E-01  ⑦
3        4    1.000000E+00    5.000000E-01    5.000000E-01  ⑧
2
2    1.000000E+00    5.000000E-01    0.000000E+00
4    1.000000E+00    5.000000E-01    0.000000E+00
0
1
1    1.000000E+00
```



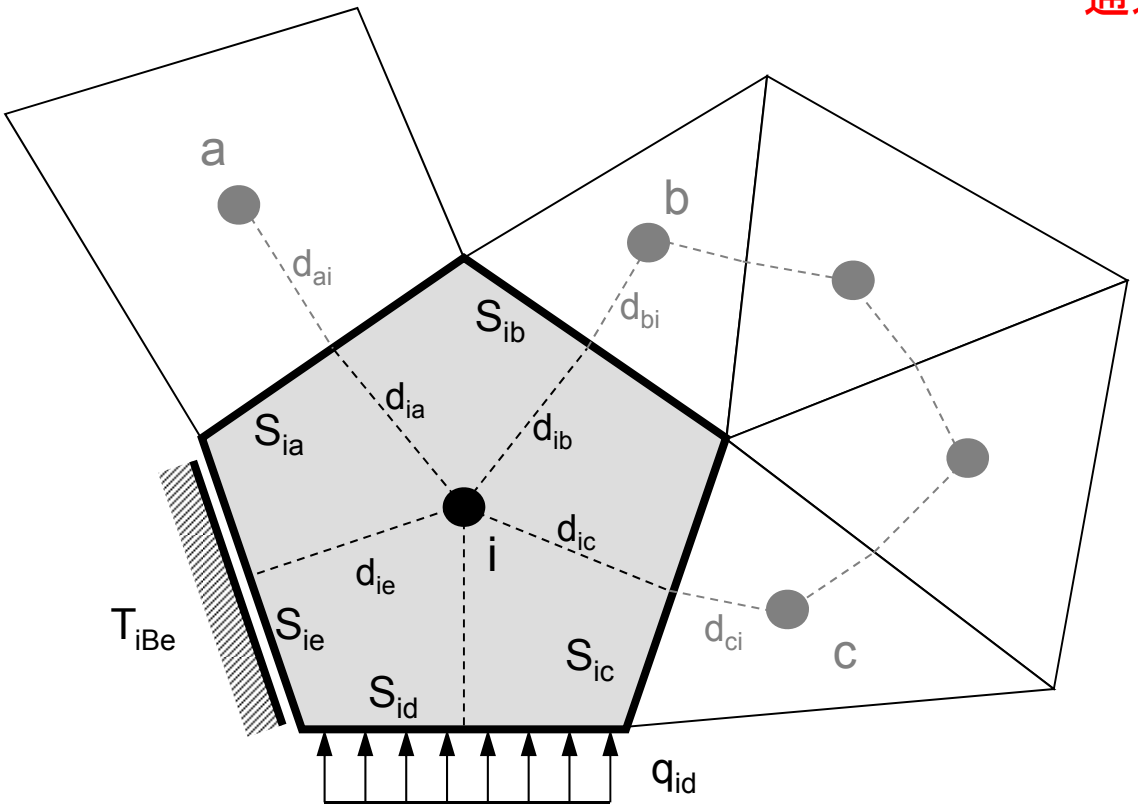
基本的に初期全体メッシュファイルと同じ
局所要素番号による記述
コネクティビティ:「内点～内点」,「内点～外点」のみの情報

全体マトリクスの生成

要素iに関する釣り合い

$$\sum_k \frac{S_{ik}}{\frac{d_{ik}}{\lambda_i}+\frac{d_{ki}}{\lambda_k}}\left(T_k - T_i\right) + \sum_e \frac{S_{ie}}{\frac{d_{ie}}{\lambda_i}}\left(T_{iBe} - T_i\right) + \sum_d S_{id}\dot{q}_{id} + V_i\dot{Q}_i = 0$$

隣接要素との熱伝導　　温度固定境界　　要素境界面通過熱流束　　体積発熱



- λ : 熱伝導率
- V_i : 要素体積
- S : 表面面積
- d_{ij} : 要素中心から表面までの距離
- q : 表面フラックス
- Q : 体積発熱
- T_{iB} : 境界温度

全体マトリクスの生成

要素iに関する釣り合い

$$\sum_k \frac{S_{ik}}{\frac{d_{ik}}{\lambda_i}+\frac{d_{ki}}{\lambda_k}}\left(T_k - T_i\right) + \sum_e \frac{S_{ie}}{\frac{d_{ie}}{\lambda_i}}\left(T_{iBe} - T_i\right) + \sum_d S_{id}\dot{q}_{id} + V_i\dot{Q}_i = 0$$

$$-\sum_k \frac{S_{ik}}{\frac{d_{ik}}{\lambda_i}+\frac{d_{ki}}{\lambda_k}} T_k + \sum_k \frac{S_{ik}}{\frac{d_{ik}}{\lambda_i}+\frac{d_{ki}}{\lambda_k}} T_i - \sum_e \frac{S_{ie}}{\frac{d_{ie}}{\lambda_i}} T_{iBe} + \sum_e \frac{S_{ie}}{\frac{d_{ie}}{\lambda_i}} T_i = \sum_d S_{id}\dot{q}_{id} + V_i\dot{Q}_i$$

定数項：右辺へ移項

全体マトリクスの生成

要素iに関する釣り合い

$$\sum_k \frac{S_{ik}}{\frac{d_{ik}}{\lambda_i}+\frac{d_{ki}}{\lambda_k}}\left(T_k - T_i\right)+\sum_e \frac{S_{ie}}{\frac{d_{ie}}{\lambda_i}}\left(T_{iBe} - T_i\right)+\sum_d S_{id}\dot{q}_{id} + V_i\dot{Q}_i = 0$$

$$-\sum_k \frac{S_{ik}}{\frac{d_{ik}}{\lambda_i}+\frac{d_{ki}}{\lambda_k}}T_k + \sum_k \frac{S_{ik}}{\frac{d_{ik}}{\lambda_i}+\frac{d_{ki}}{\lambda_k}}T_i - \sum_e \frac{S_{ie}}{\frac{d_{ie}}{\lambda_i}}T_{iBe} + \sum_e \frac{S_{ie}}{\frac{d_{ie}}{\lambda_i}}T_i = \sum_d S_{id}\dot{q}_{id} + V_i\dot{Q}_i$$

定数項：右辺へ移項

$$\left[\sum_k \frac{S_{ik}}{\frac{d_{ik}}{\lambda_i}+\frac{d_{ki}}{\lambda_k}}+\sum_e \frac{S_{ie}}{\frac{d_{ie}}{\lambda_i}}\right]T_i - \left[\sum_k \frac{S_{ik}}{\frac{d_{ik}}{\lambda_i}+\frac{d_{ki}}{\lambda_k}}T_k\right] = \sum_d S_{id}\dot{q}_{id} + V_i\dot{Q}_i + \sum_e \frac{S_{ie}}{\frac{d_{ie}}{\lambda_i}}T_{iBe}$$

D（対角成分）　**AMAT（非対角成分）**　**BFORCE（右辺）**

全体マトリクスの生成

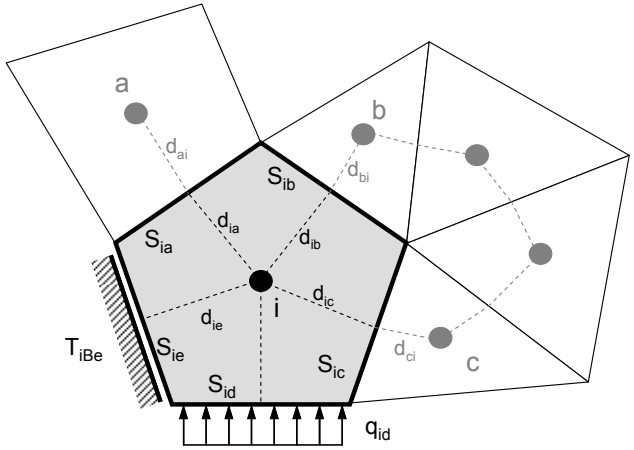
要素iに関する釣り合い

$$\left[\sum_k \frac{S_{ik}}{\frac{d_{ik}}{\lambda_i}+\frac{d_{ki}}{\lambda_k}}+\sum_e \frac{S_{ie}}{\frac{d_{ie}}{\lambda_i}}\right]T_i-\left[\sum_k \frac{S_{ik}}{\frac{d_{ik}}{\lambda_i}+\frac{d_{ki}}{\lambda_k}}T_k\right]=\sum_d S_{id}\dot{q}_{id}+V_i\dot{Q}_i+\sum_e \frac{S_{ie}}{\frac{d_{ie}}{\lambda_i}}T_{iBe}$$

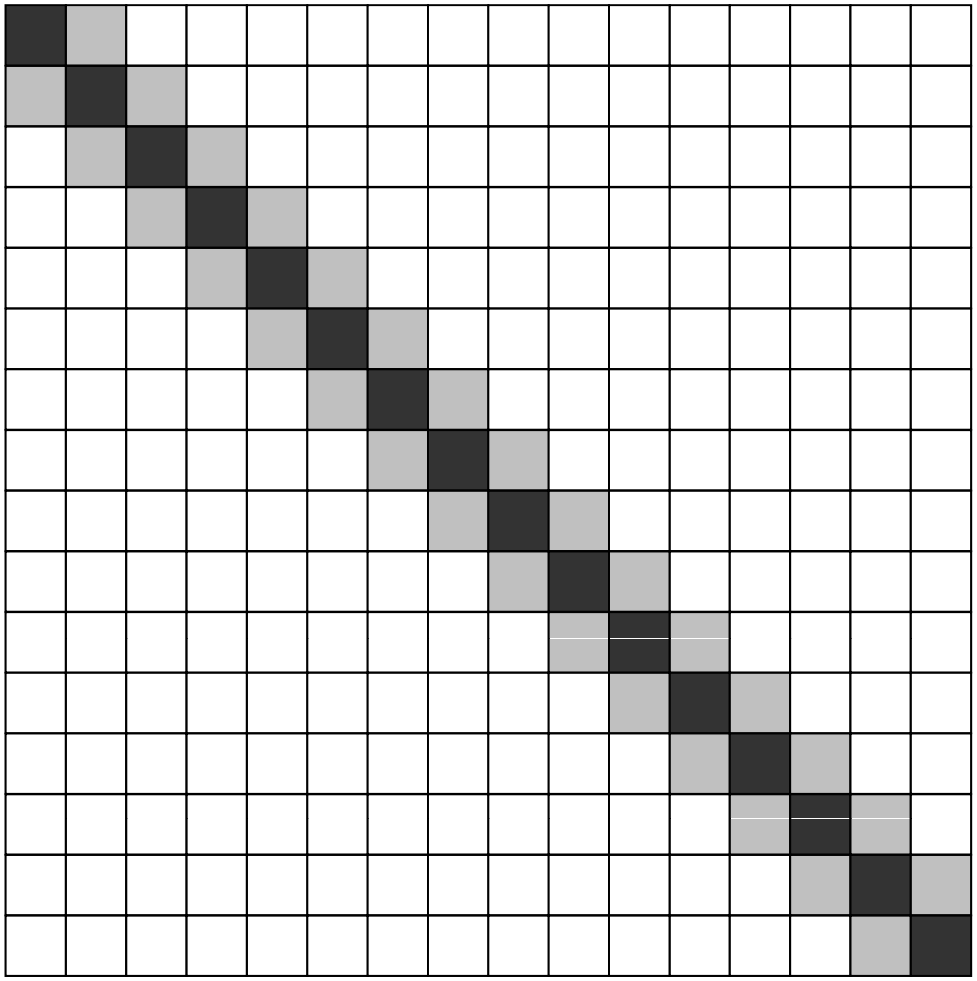
D（対角成分） → **隣接要素の情報必要**

AMAT（非対角成分） → **隣接要素の情報必要**

BFORCE（右辺） → **自分自身（要素i）の情報のみ必要**



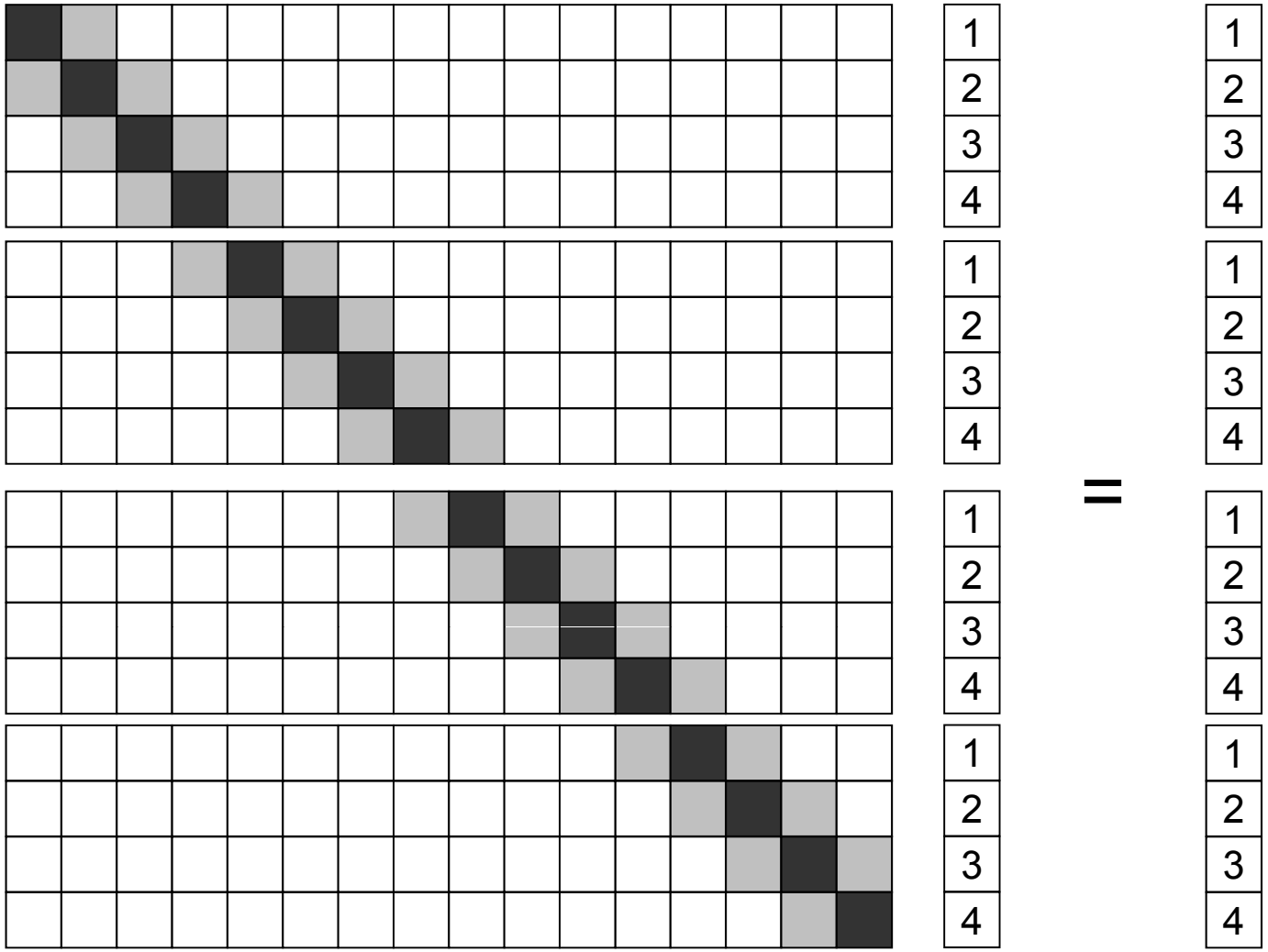
全体マトリクス（一次元熱伝導問題の例）



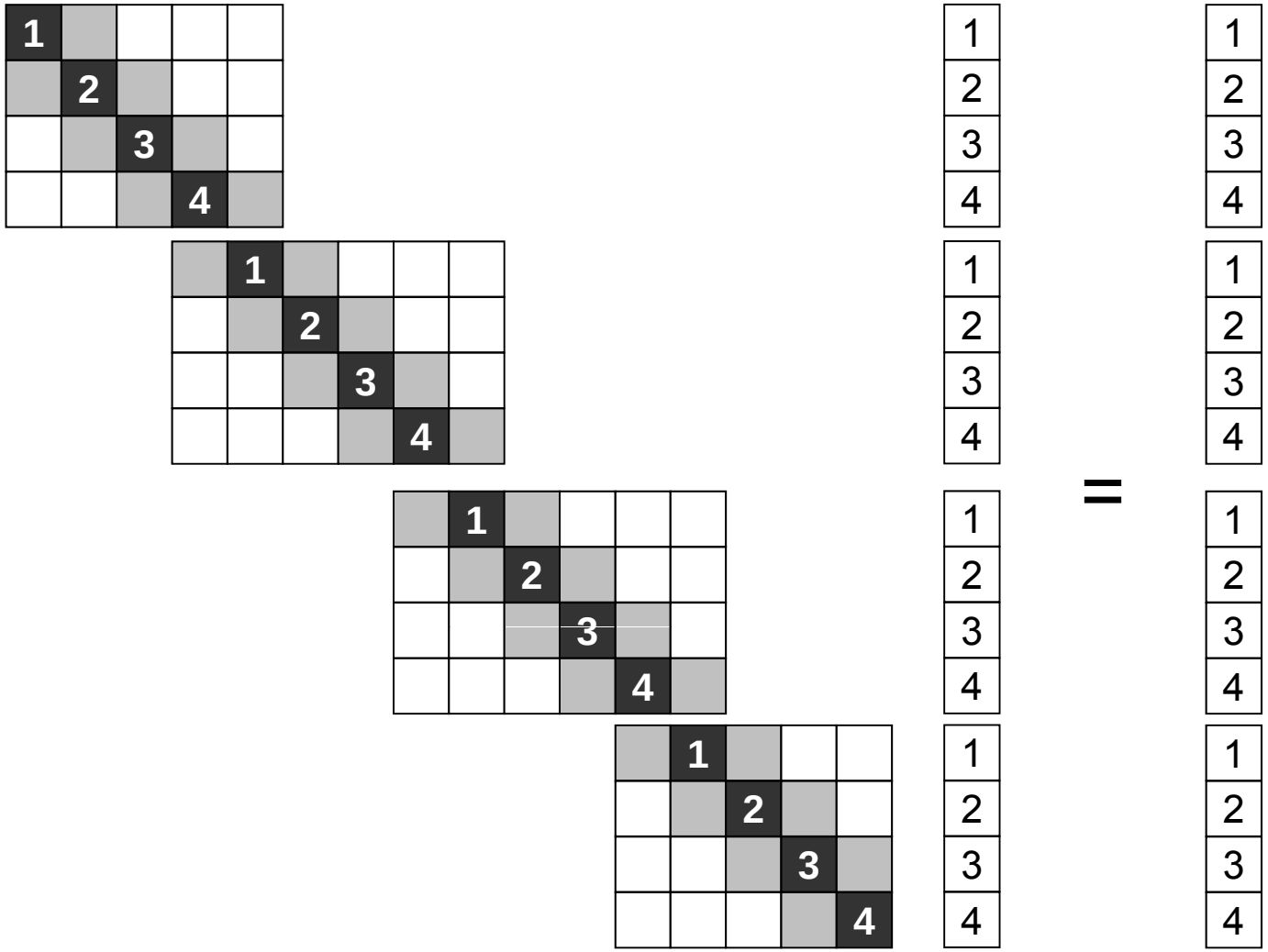
1			1
2			2
3			3
4			4
5			5
6			6
7			7
8		=	8
9			9
10			10
11			11
12			12
13			13
14			14
15			15
16			16

[AMAT+D]　　{PHI}　{RHS}

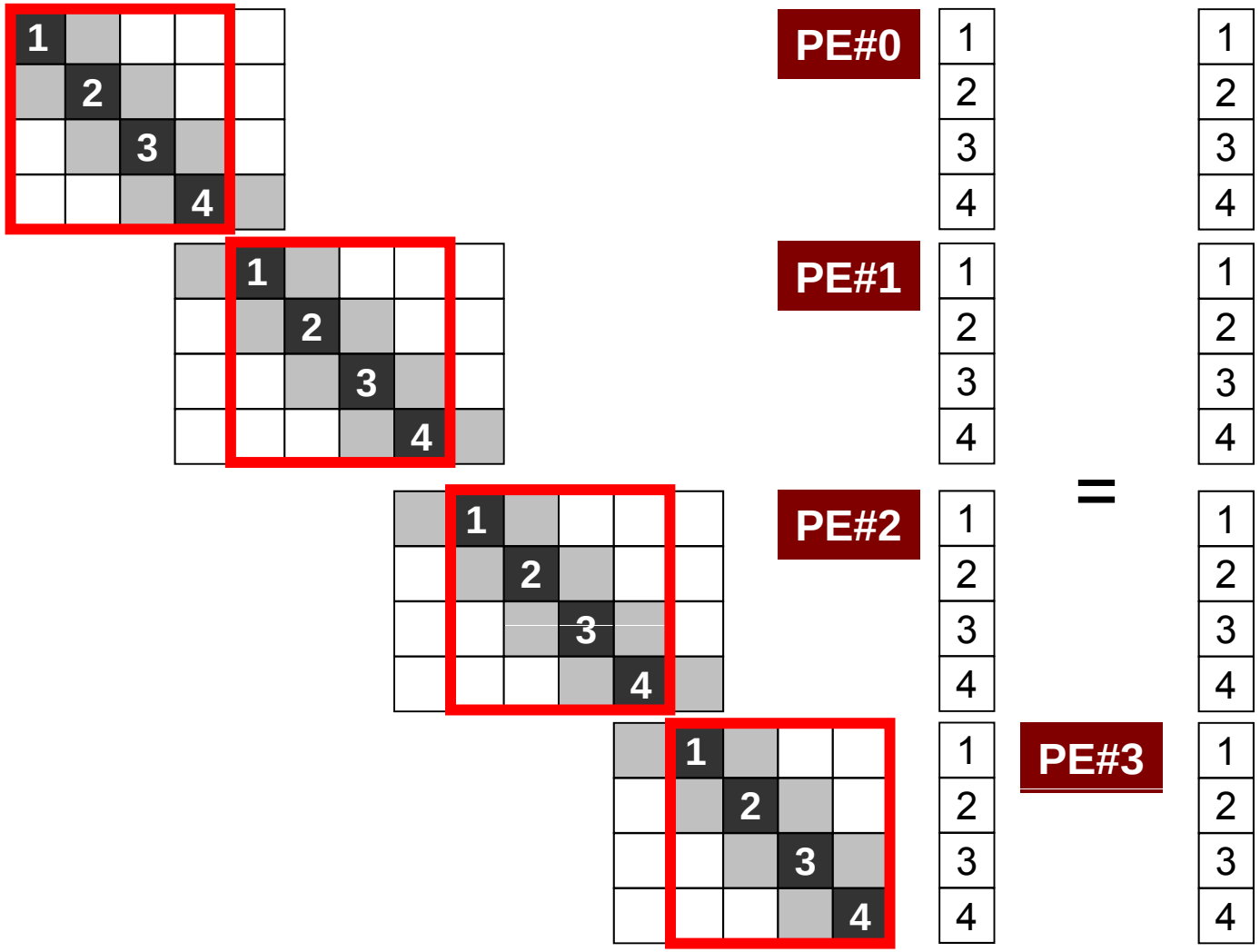
4領域に分割すると・・・



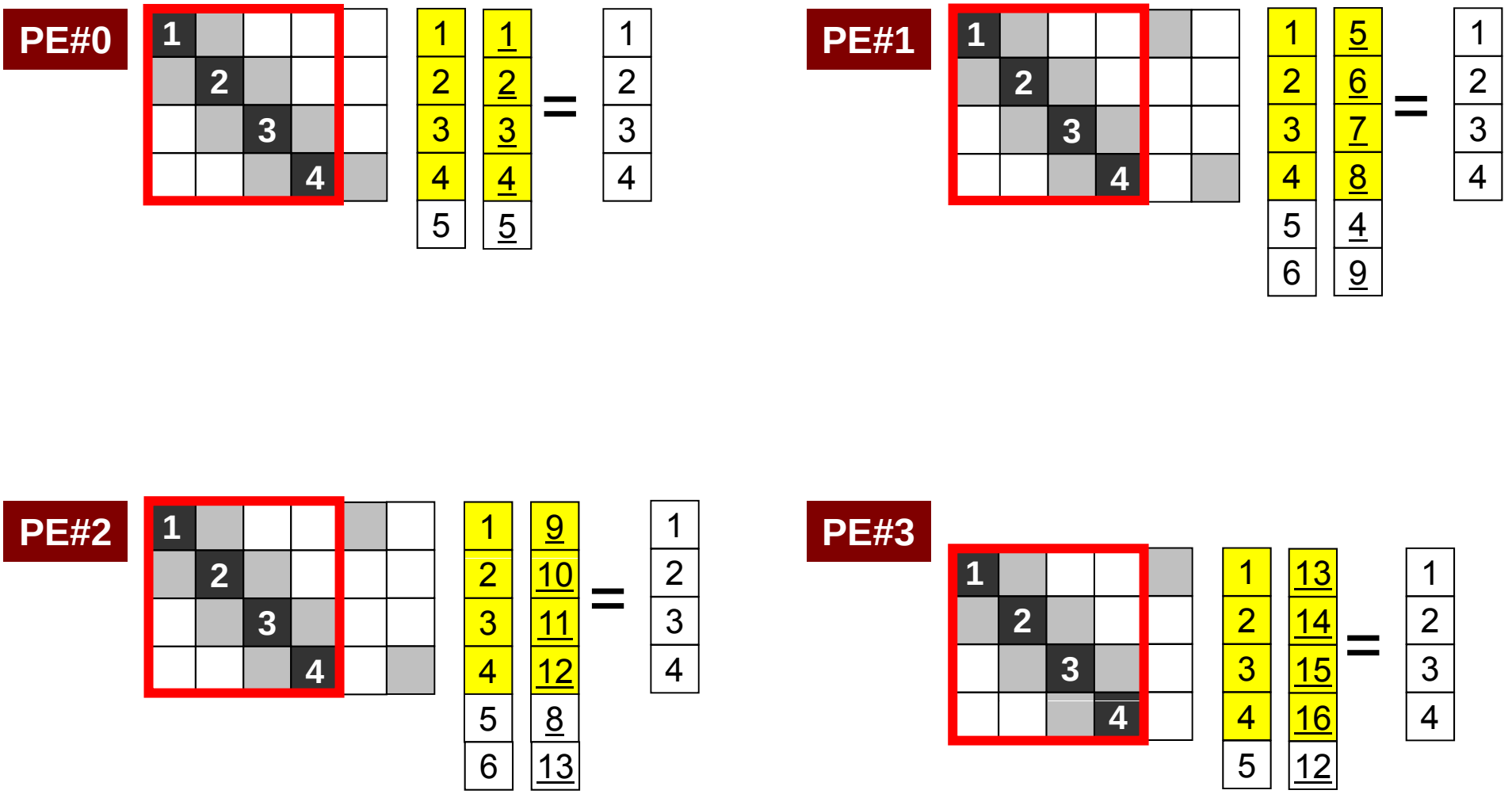
4領域に分割すると・・・



4領域に分割すると・・・



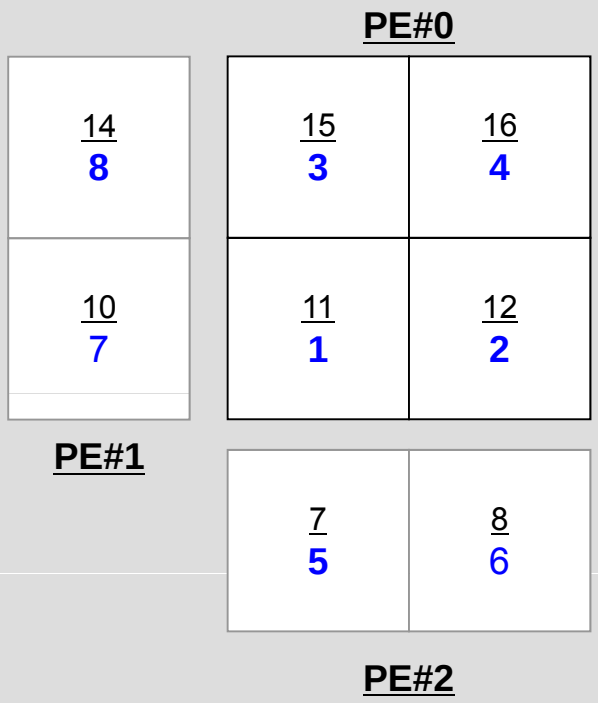
内点～外点の順番で番号付け



局所分散通信ファイル（comm.0）

隣接領域

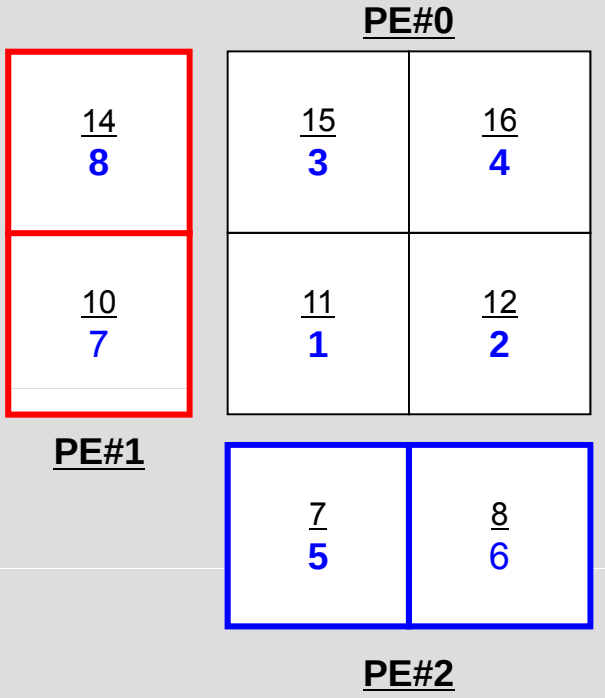
```
#NEIBPEtot
   2
#NEIBPE
   1   2
#IMPORT index
   2   4
#IMPORT items
   7   8   5   6
#EXPORT index
   2   4
#EXPORT items
   1   3   1   2
#INTERNAL NODE
   4
#TOTAL NODE
   8
#GLOBAL NODE ID
  11  12  15  16   7   8
  10  14
```



局所分散通信ファイル（comm.0）

受信テーブル，外点情報

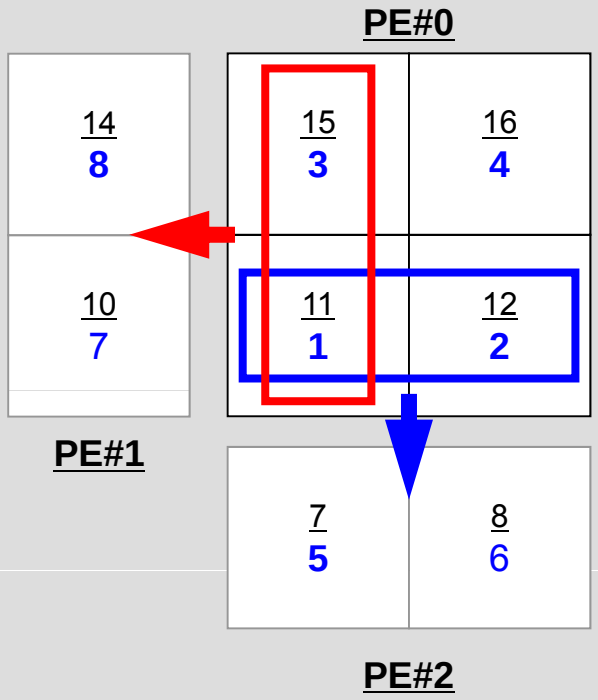
```
#NEIBPEtot
   2
#NEIBPE
   1   2
#IMPORT index
   2   4
#IMPORT items
   7   8   5   6
#EXPORT index
   2   4
#EXPORT items
   1   3   1   2
#INTERNAL NODE
   4
#TOTAL NODE
   8
#GLOBAL NODE ID
  11  12  15  16   7   8
  10  14
```



局所分散通信ファイル（comm.0）

送信テーブル，境界点情報

```
#NEIBPEtot
   2
#NEIBPE
   1   2
#IMPORT index
   2   4
#IMPORT items
   7   8   5   6
#EXPORT index
   2   4
#EXPORT items
   1   3   1   2
#INTERNAL NODE
   4
#TOTAL NODE
   8
#GLOBAL NODE ID
  11  12  15  16   7   8
  10  14
```

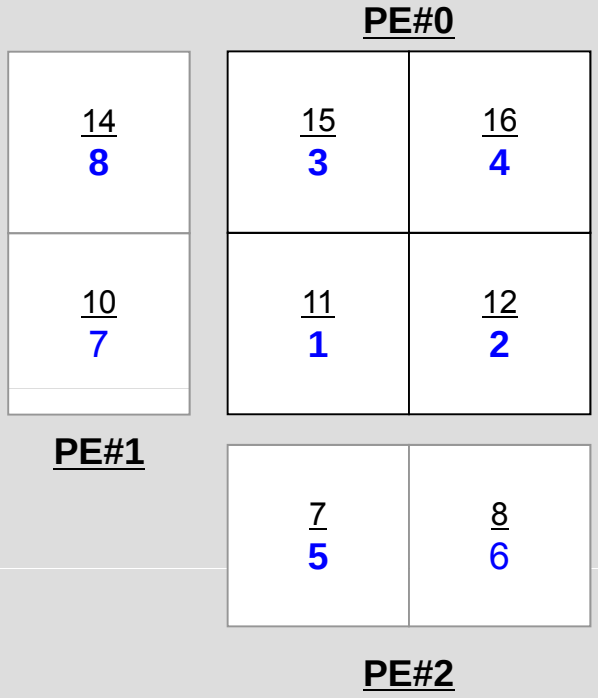


局所分散通信ファイル(comm.0)

内点数，総要素数(内点＋外点)，全体要素番号

```
#NEIBPEtot
   2
#NEIBPE
   1   2
#IMPORT index
   2   4
#IMPORT items
   7   8   5   6
#EXPORT index
   2   4
#EXPORT items
   1   3   1   2
#INTERNAL NODE
   4
#TOTAL NODE
   8
#GLOBAL NODE ID
  11  12  15  16   7   8
  10  14
```

全体要素番号(局所番号順)



- データ構造とアルゴリズム：局所分散データ
- FVMにおける並列計算と局所分散データ構造の考え方
- 領域分割手法について
- eps_fvmにおける領域分割機能：eps_fvm_part
- **eps_fvm「並列化」に向けて**

「eps_fvm」並列化に向けて

```
program eps_fvm
use eps_fvm_all

implicit REAL*8 (A-H,O-Z)

call eps_fvm_input_grid
call poi_gen
call eps_fvm_solver
call output_ucd

end program eps_fvm
```

「eps_fvm」並列化に向けて

- eps_fvm_input_grid
  - 並列分散メッシュデータ
  - 並列分散通信データ
- poi_gen
  - この部分は（多分）変更不要
- eps_fvm_solver
  - 行列ベクトル積，内積
- output_ucd
  - 並列分散処理について考慮する必要あり
- 実を言うと，局所分散データ構造を定めた時点で並列化は9割方終了している。