MPIによるプログラミング概要
Fortran編（2/2）

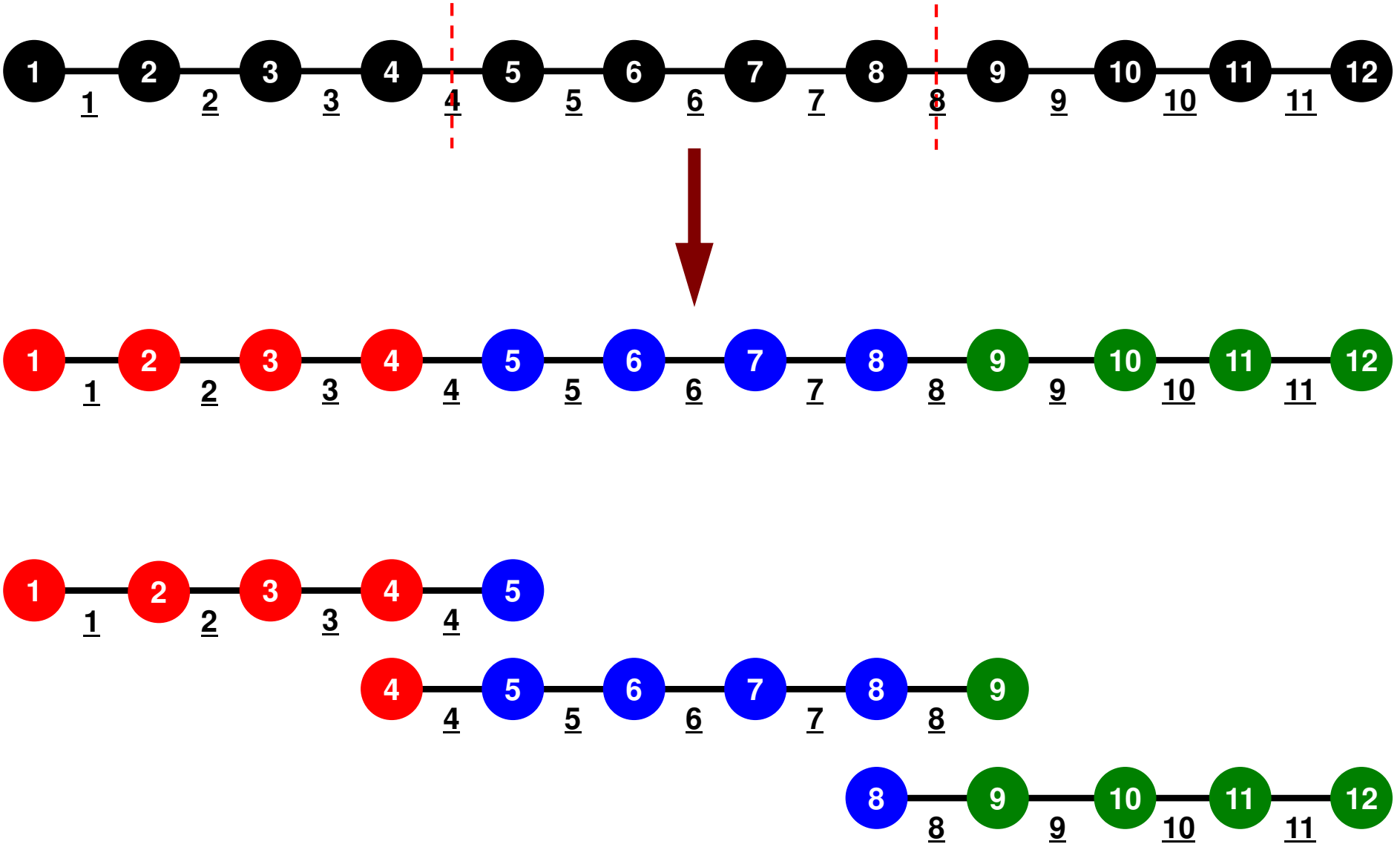
中島 研吾
東京大学情報基盤センター

- MPIとは
- MPIの基礎：Hello World
- 集団通信（Collective Communication）
- **1対1通信（Point-to-Point Communication）**

1対1通信

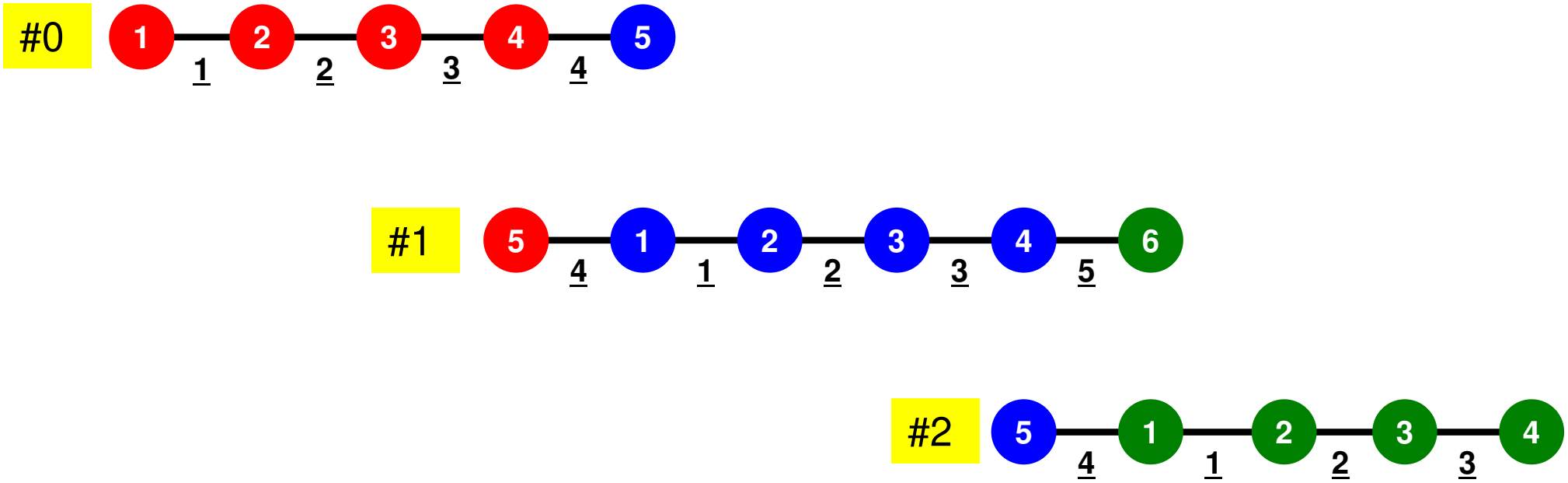
- 1対1通信とは　？
- 二次元問題，一般化された通信テーブル
- 課題S2

一次元問題:11要素，12節点，3領域



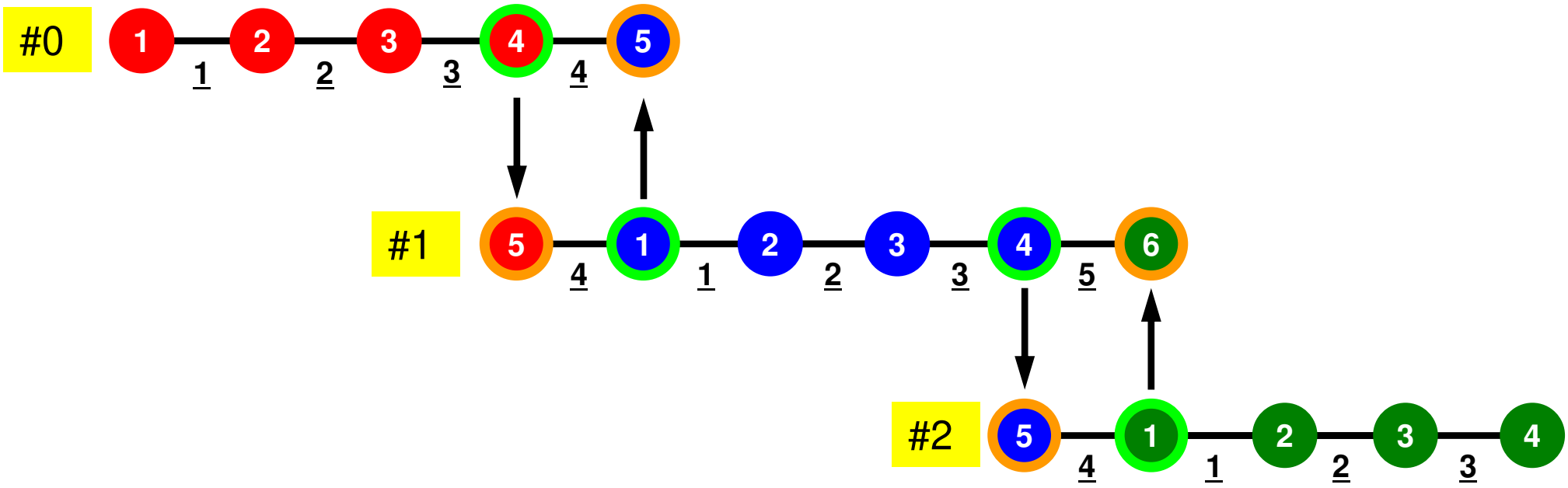
一次元問題：11要素，12節点，3領域

局所番号：節点・要素とも1からふる



一次元問題：11要素，12節点，3領域

外点・境界点



X 外点：External Nodes

Y 境界点：Boundary Nodes

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

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

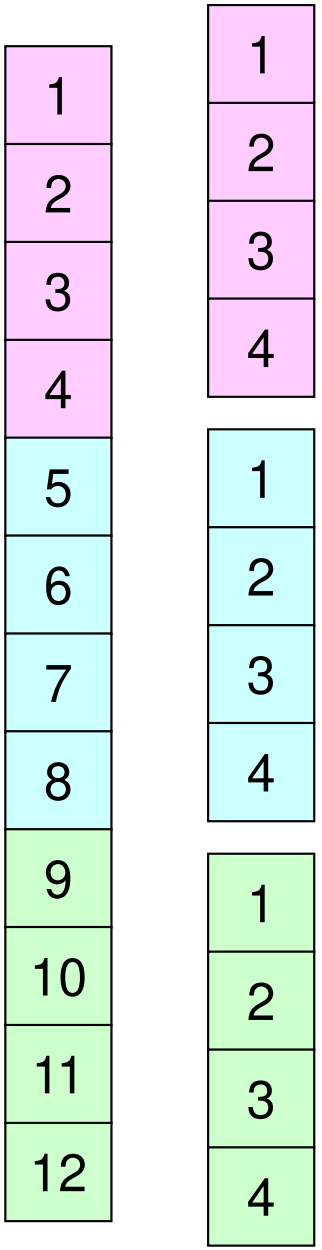
前処理：対角スケーリング

前処理，ベクトル定数倍の加減

局所的な計算（内点のみ）が可能⇒並列処理

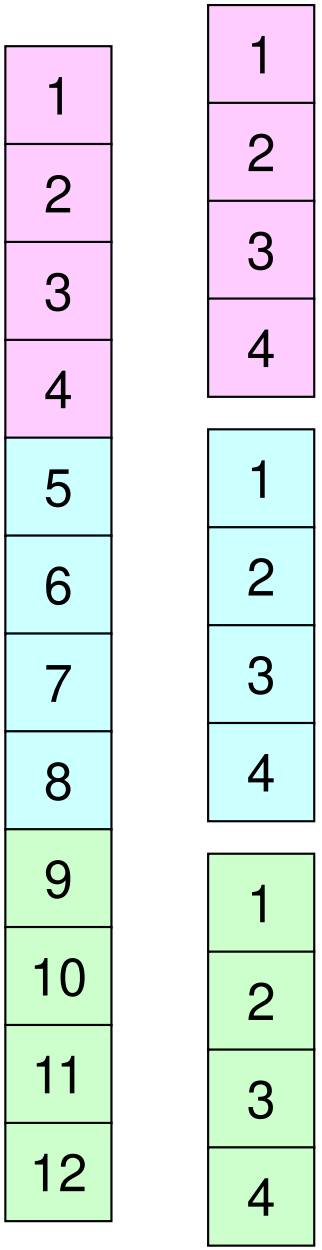
```
!C
!C-- {z}= [Minv]{r}

      do i= 1, N
        W(i,Z)= W(i,DD) * W(i,R)
      enddo
```

```
!C
!C-- {x}= {x} + ALPHA*{p}
!C   {r}= {r} - ALPHA*{q}

      do i= 1, N
        PHI(i)= PHI(i) + ALPHA * W(i,P)
        W(i,R)= W(i,R) - ALPHA * W(i,Q)
      enddo
```



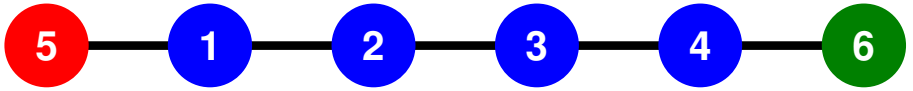
内積
全体で和をとる必要がある⇒通信?

```
!C
!C-- ALPHA= RHO / {p}{q}

      C1= 0.d0
      do i= 1, N
        C1= C1 + W(i,P)*W(i,Q)
      enddo
      ALPHA= RHO / C1
```

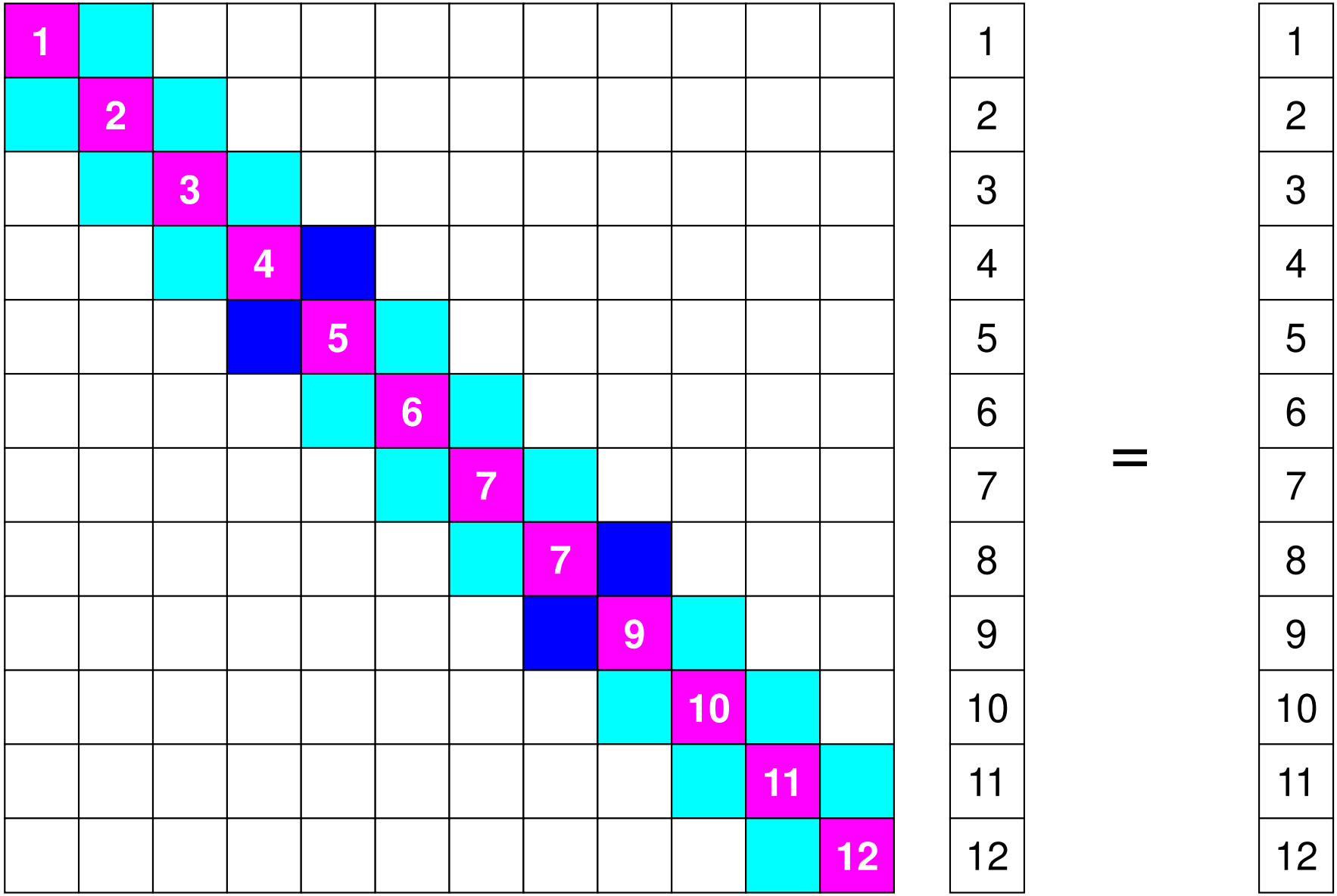


行列ベクトル積
外点の値が必要⇒1対1通信

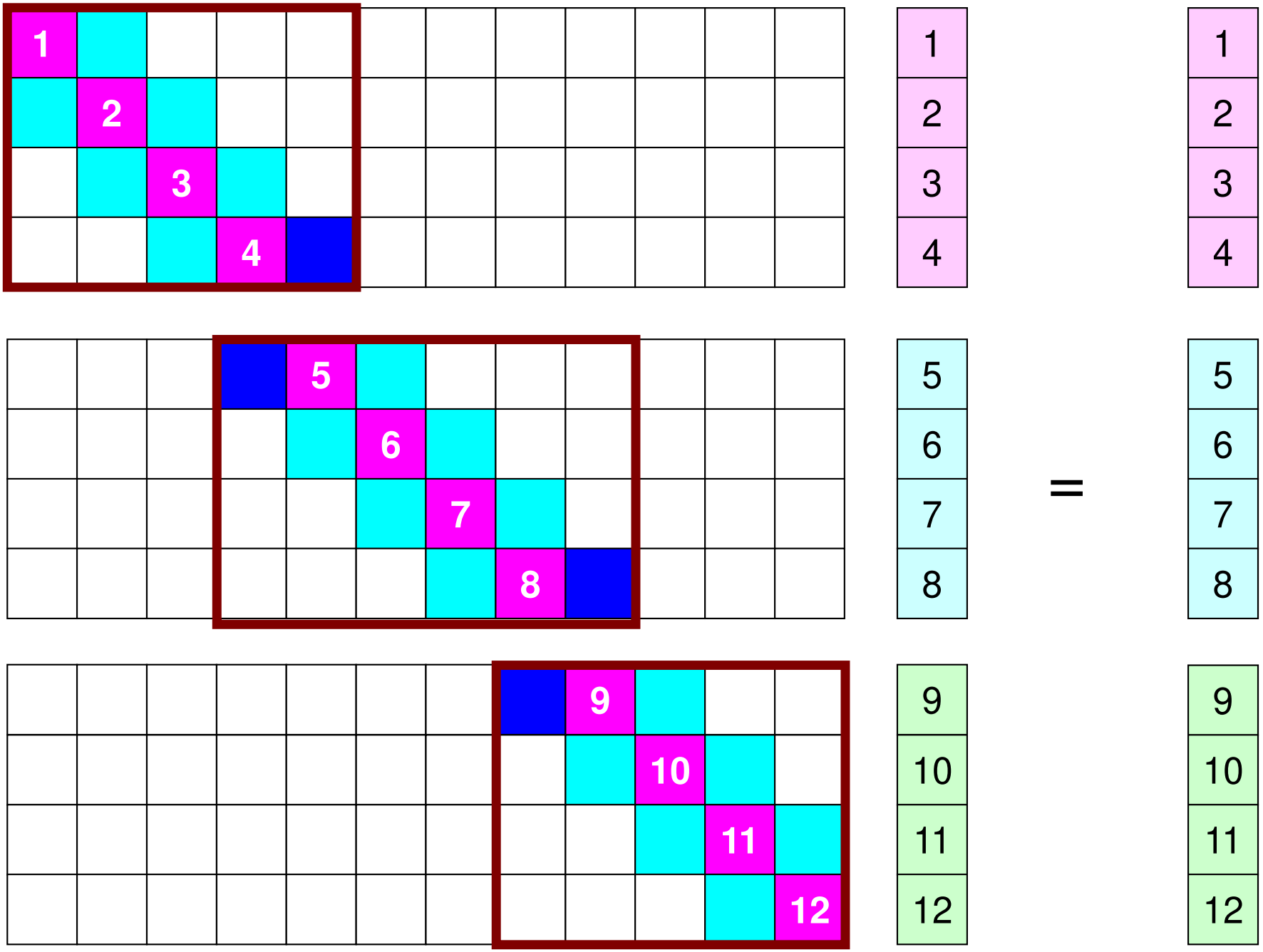
```
!C
!C-- {q}= [A]{p}

      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
```



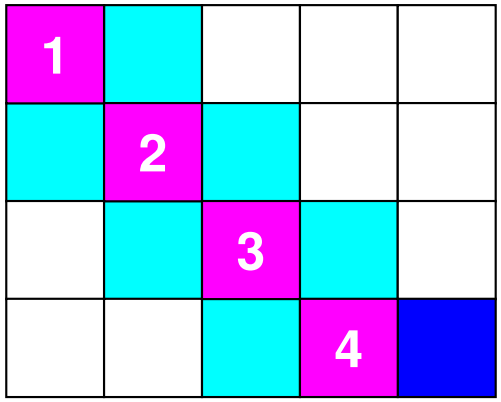
行列ベクトル積：ローカルに計算実施可能

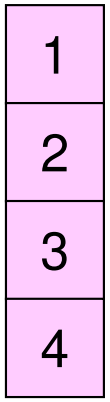


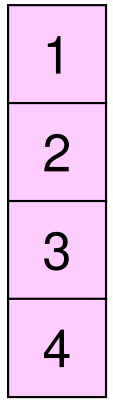
行列ベクトル積：ローカルに計算実施可能

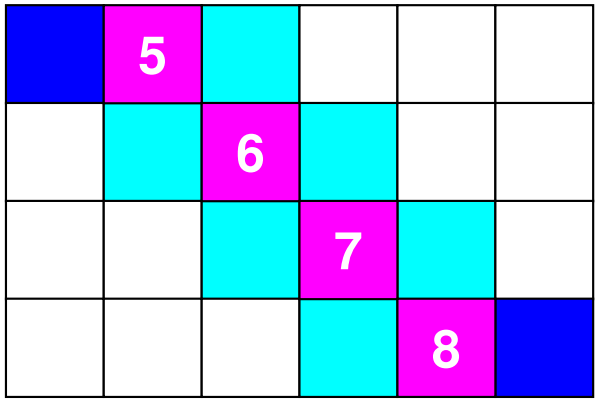


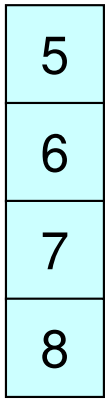
行列ベクトル積：ローカルに計算実施可能



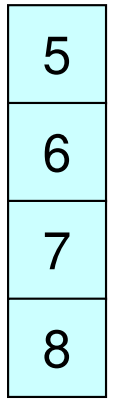


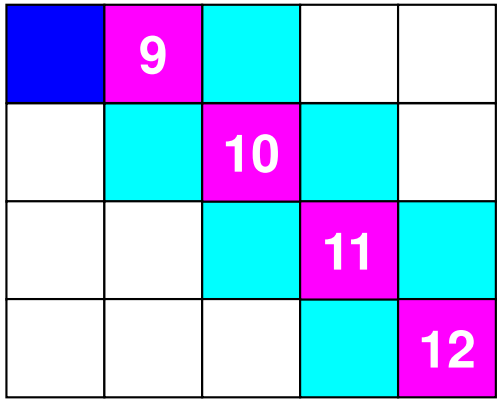


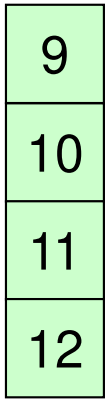


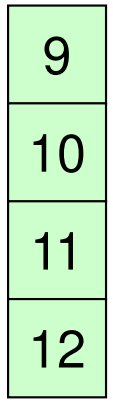


=

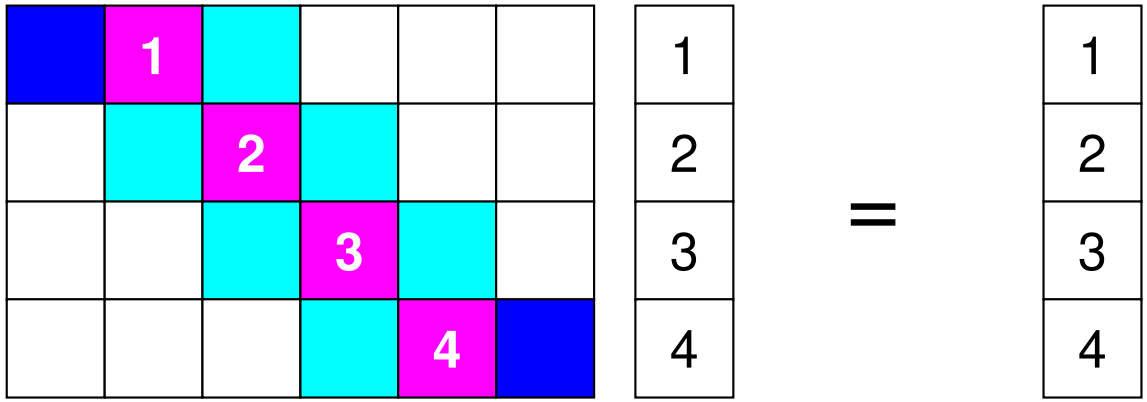


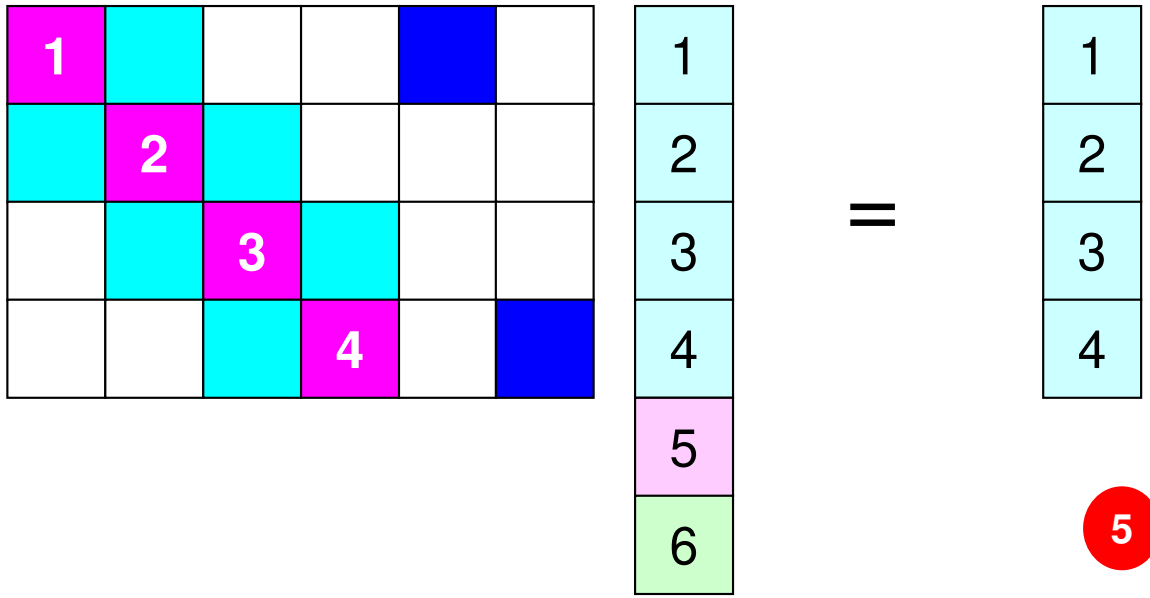


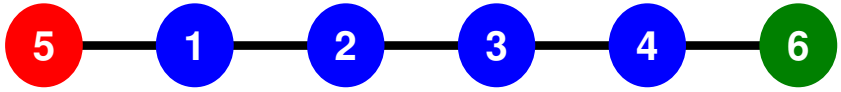




行列ベクトル積:ローカル計算 #1



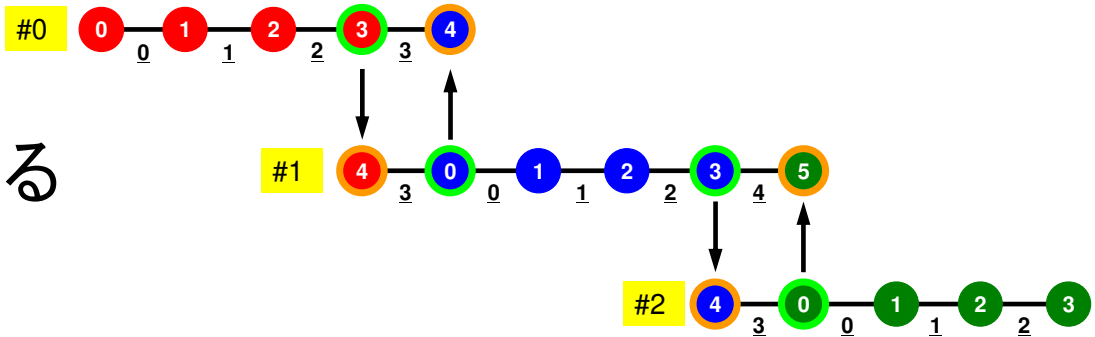




1対1通信とは？

- 集団通信：Collective Communication
  - MPI_Reduce, MPI_Scatter/Gather など
  - 同じコミュニケータ内の全プロセスと通信する
  - 適用分野
    - 境界要素法，スペクトル法，分子動力学等グローバルな相互作用のある手法
    - 内積，最大値などのオペレーション

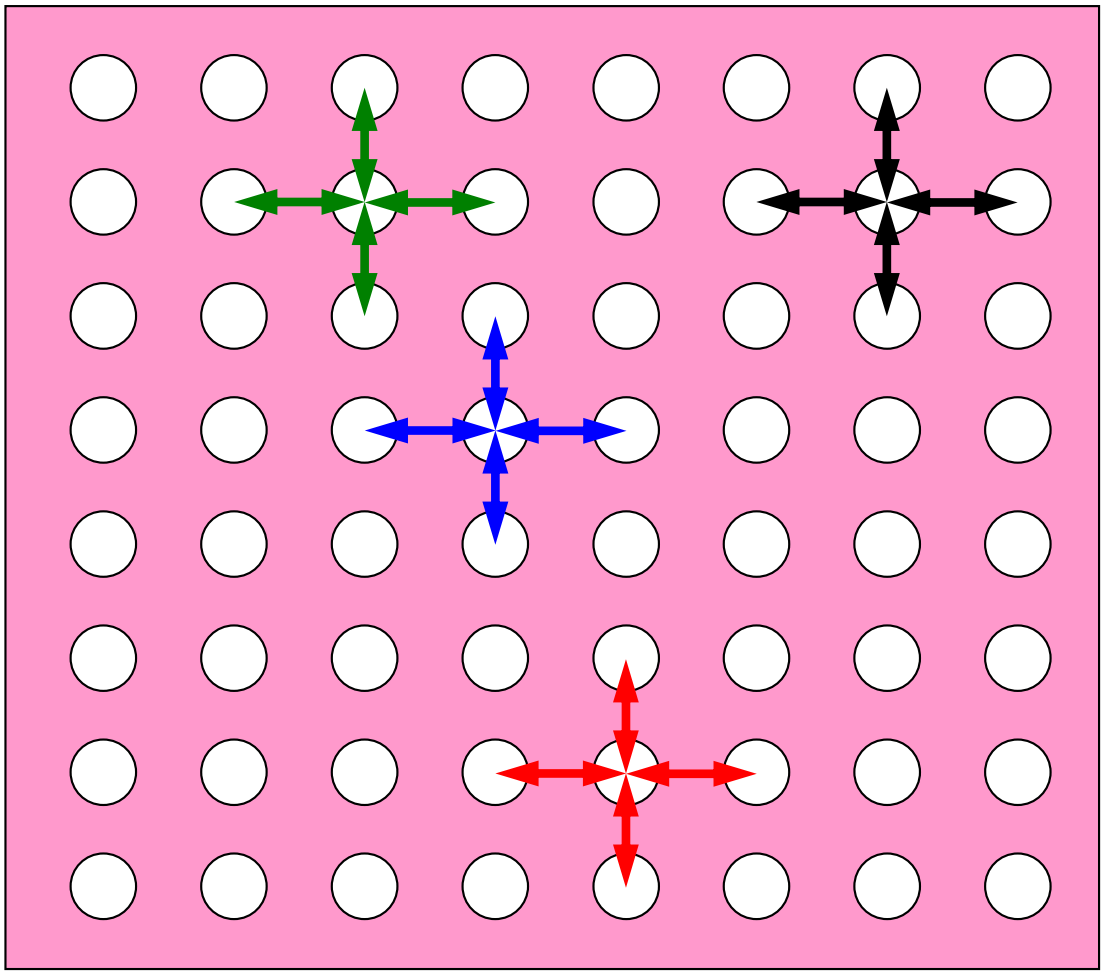
- 1対1通信：Point-to-Point
  - MPI_Send, MPI_Recv
  - 特定のプロセスとのみ通信がある
    - 隣接領域
  - 適用分野
    - 差分法，有限要素法などローカルな情報を使う手法



集団通信，1対1通信

近接PE（領域）のみとの相互作用

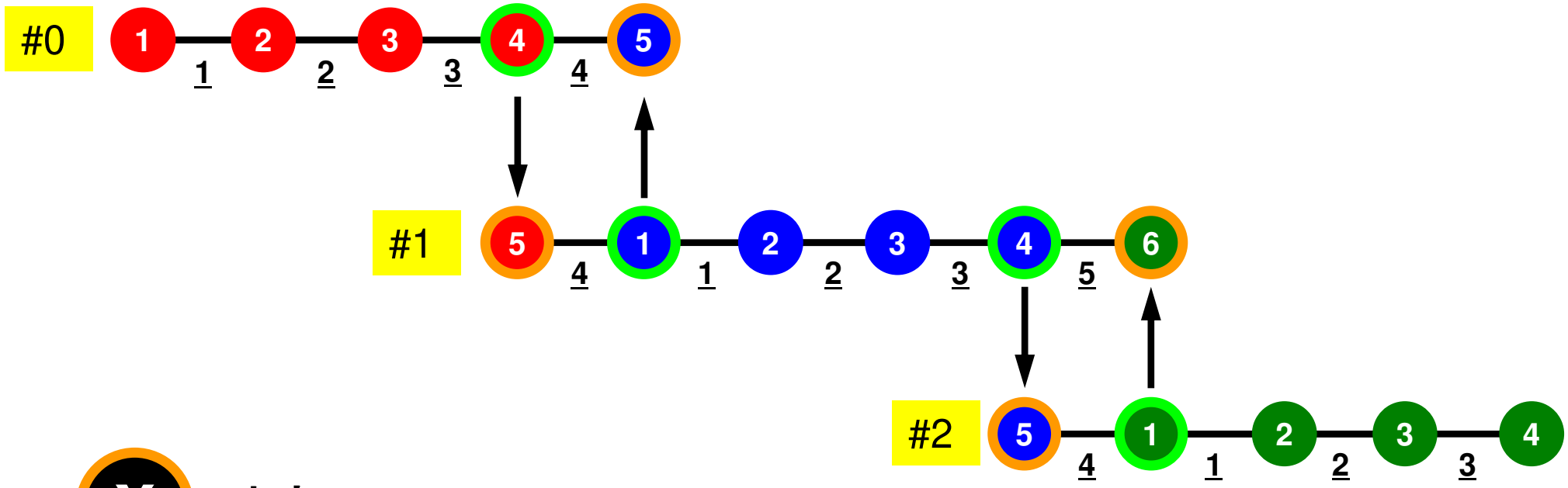
差分法，有限要素法



1対1通信が必要になる場面：1DFEM

FEMのオペレーションのためには隣接領域の情報が必要

マトリクス生成，反復法



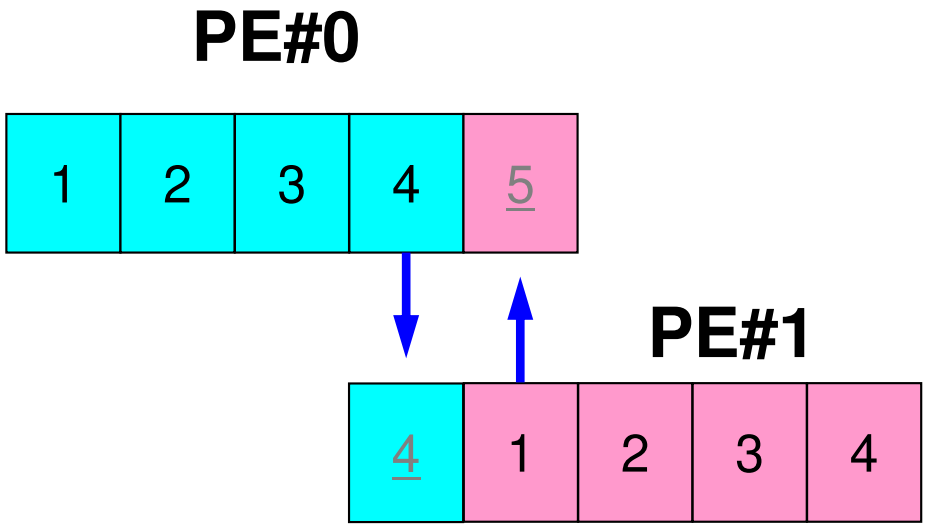
X 外点：External Nodes

Y 境界点：Boundary Nodes

1対1通信の方法

- **MPI_Send**，　**MPI_Recv**というサブルーチンがある。
- しかし，これらは「ブロッキング（blocking）」通信サブルーチンで，デッドロック（dead lock）を起こしやすい。
  - 受信（RECV）の完了が確認されないと，送信（SEND）が終了しない
- もともと非常に「secureな」通信を保障するために，MPI仕様の中に入れられたものであるが，実用上は不便この上ない。
  - したがって実際にアプリケーションレベルで使用されることはほとんど無い（と思う）。
  - 将来にわたってこの部分が改正される予定はないらしい。
- 「そういう機能がある」ということを心の片隅においておいてください。

MPI_SEND/MPI_RECV




```
if (my_rank.eq.0) NEIB_ID=1
if (my_rank.eq.1) NEIB_ID=0

…
call MPI_SEND (NEIB_ID, arg's)
call MPI_RECV (NEIB_ID, arg's)
…
```

- 例えば先ほどの例で言えば，このようにしたいところであるが，このようなプログラムを作るとMPI_Send/MPI_Recvのところで止まってしまう。
  - 動く場合もある

MPI_Send/MPI_Recv

PE#0

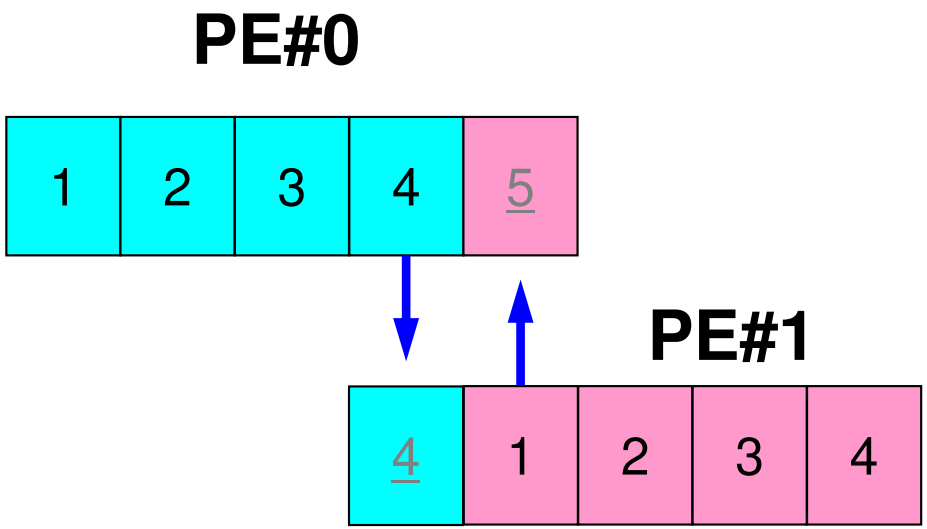
```
call MPI_SEND (NEIB_ID, arg's)
call MPI_RECV (NEIB_ID, arg's)
```

PE#1

```
call MPI_SEND (NEIB_ID, arg's)
call MPI_RECV (NEIB_ID, arg's)
```

- PE#0，PE#1共に「送信」
- 「送信」は，送信先での「受信」がされないと完了しない
- PE#0，PE#1ともに，送信後，送信先での受信が終了するのを待つ⇒プロセスは停止

MPI_SEND/MPI_RECV（続き）

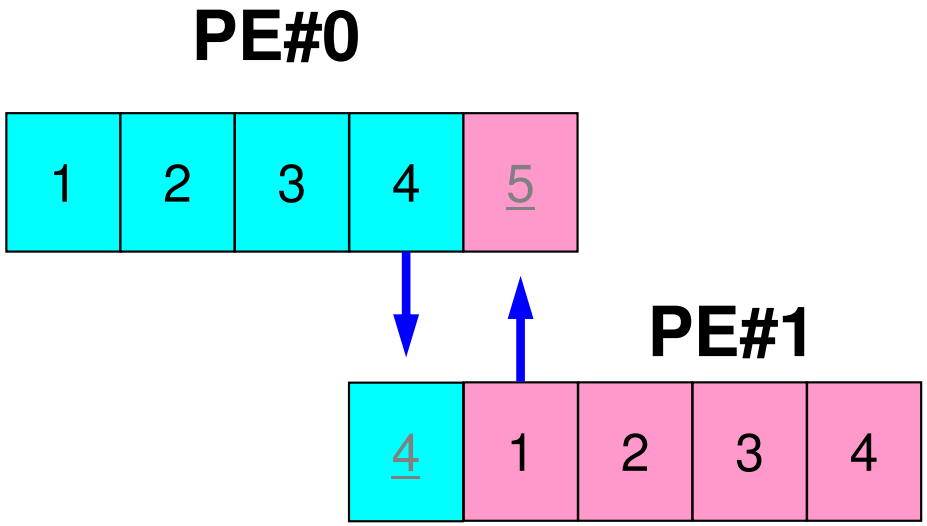



```
if (my_rank.eq.0) NEIB_ID=1
if (my_rank.eq.1) NEIB_ID=0

…
if (my_rank.eq.0) then
  call MPI_SEND (NEIB_ID, arg's)
  call MPI_RECV (NEIB_ID, arg's)
endif

if (my_rank.eq.1) then
  call MPI_RECV (NEIB_ID, arg's)
  call MPI_SEND (NEIB_ID, arg's)
endif

…
```

- このようにすれば，動く。
- 規則的な差分格子のような場合にはこれでもOKだが不規則形状では適用困難。

1対1通信の方法（実際どうするか）

- MPI_Isend，　MPI_Irecv，という「ブロッキングしない（non-blocking）」サブルーチンがある。これと，同期のための「MPI_Waitall」を組み合わせる。
- MPI_Sendrecv というサブルーチンもある（後述）。




```
if (my_rank.eq.0) NEIB_ID=1
if (my_rank.eq.1) NEIB_ID=0

…
call MPI_Isend (NEIB_ID, arg's)
call MPI_Irecv (NEIB_ID, arg's)
…
call MPI_Waitall (for Irecv)
…
call MPI_Waitall (for Isend)
```

IsendとIrecvで同じ通信識別子を使って，更に整合性が取れるのであればWaitallは一箇所でもOKです（後述）

Fortran

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））：C言語については後述
  - **ierr** 整数 O 完了コード

通信識別子（request handle）：request

- **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

- 以下のような形で宣言しておく（記憶領域を確保するだけで良い：Cについては後述）

```
allocate (request(NEIBPETOT))
```

Fortran

Fortran

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））：C言語については後述
  - **ierr** 整数 O 完了コード

Fortran

MPI_WAITALL

- 1対1非ブロッキング通信サブルーチンである「MPI_ISEND」と「MPI_IRECV」を使用した場合，プロセスの同期を取るのに使用する。
- 送信時はこの「MPI_WAITALL」を呼ぶ前に送信バッファの内容を変更してはならない。受信時は「MPI_WAITALL」を呼ぶ前に受信バッファの内容を利用してはならない。
- 整合性が取れていれば，「MPI_ISEND」と「MPI_IRECV」を同時に同期してもよい。
  - 「MPI_ISEND/IRECV」で同じ通信識別子を使用すること
- 「MPI_BARRIER」と同じような機能であるが，代用はできない。
  - 実装にもよるが，「request」，「status」の内容が正しく更新されず，何度も「MPI_ISEND/IRECV」を呼び出すと処理が遅くなる，というような経験もある。

- **call MPI_WAITALL (count,request,status,ierr)**
  - **count** 整数 I 同期する必要のある「MPI_ISEND」，「MPI_RECV」呼び出し数。
  - **request** 整数 I/O 通信識別子。「MPI_ISEND」，「MPI_IRECV」で利用した識別子名に対応。（配列サイズ：（count））
  - **status** 整数 O 状況オブジェクト配列（配列サイズ：（MPI_STATUS_SIZE,count））
    MPI_STATUS_SIZE: “mpif.h”,”mpi.h”で定められる
    パラメータ：C言語については後述
  - **ierr** 整数 O 完了コード

状況オブジェクト配列（status object）：status

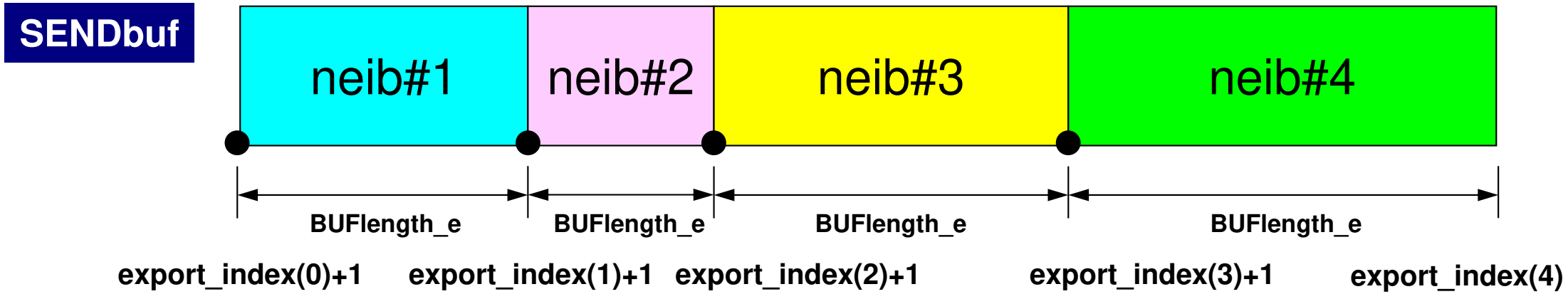
- **call MPI_WAITALL (count,request,status,ierr)**
  - **count** 整数 I 同期する必要のある「MPI_ISEND」，「MPI_RECV」呼び出し数。
  - **request** 整数 I/O 通信識別子。「MPI_ISEND」，「MPI_IRECV」で利用した識別子名に対応。（配列サイズ：（count））
  - **status** 整数 O 状況オブジェクト配列（配列サイズ：（MPI_STATUS_SIZE,count））
    MPI_STATUS_SIZE：“mpif.h”,”mpi.h”で定められるパラメータ
  - **ierr** 整数 O 完了コード

- 以下のように予め記憶領域を確保しておくだけでよい（Cについては後述）：

```
allocate (stat(MPI_STATUS_SIZE,NEIBPETOT))
```

Fortran

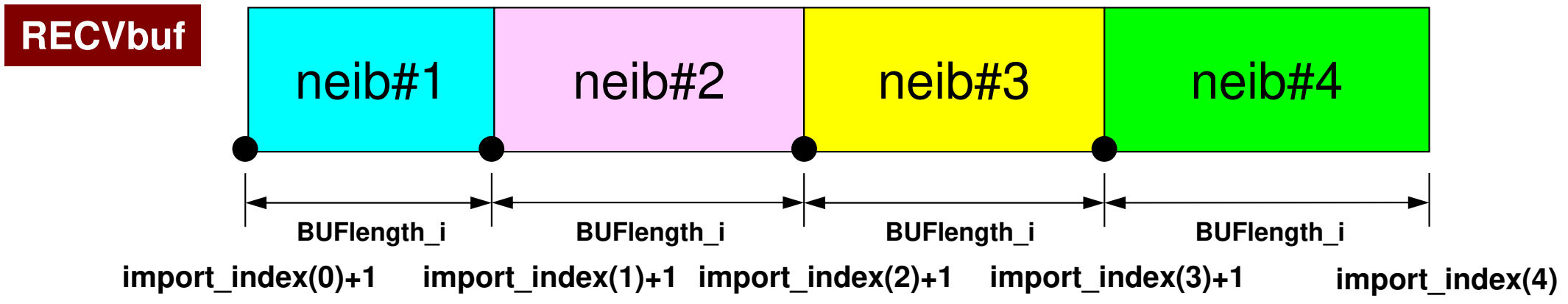
SEND: MPI_Isend/Irecv/Waitall




```
 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_send, ierr)
```

RECV: 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)
```



Fortran

MPI_SENDRECV

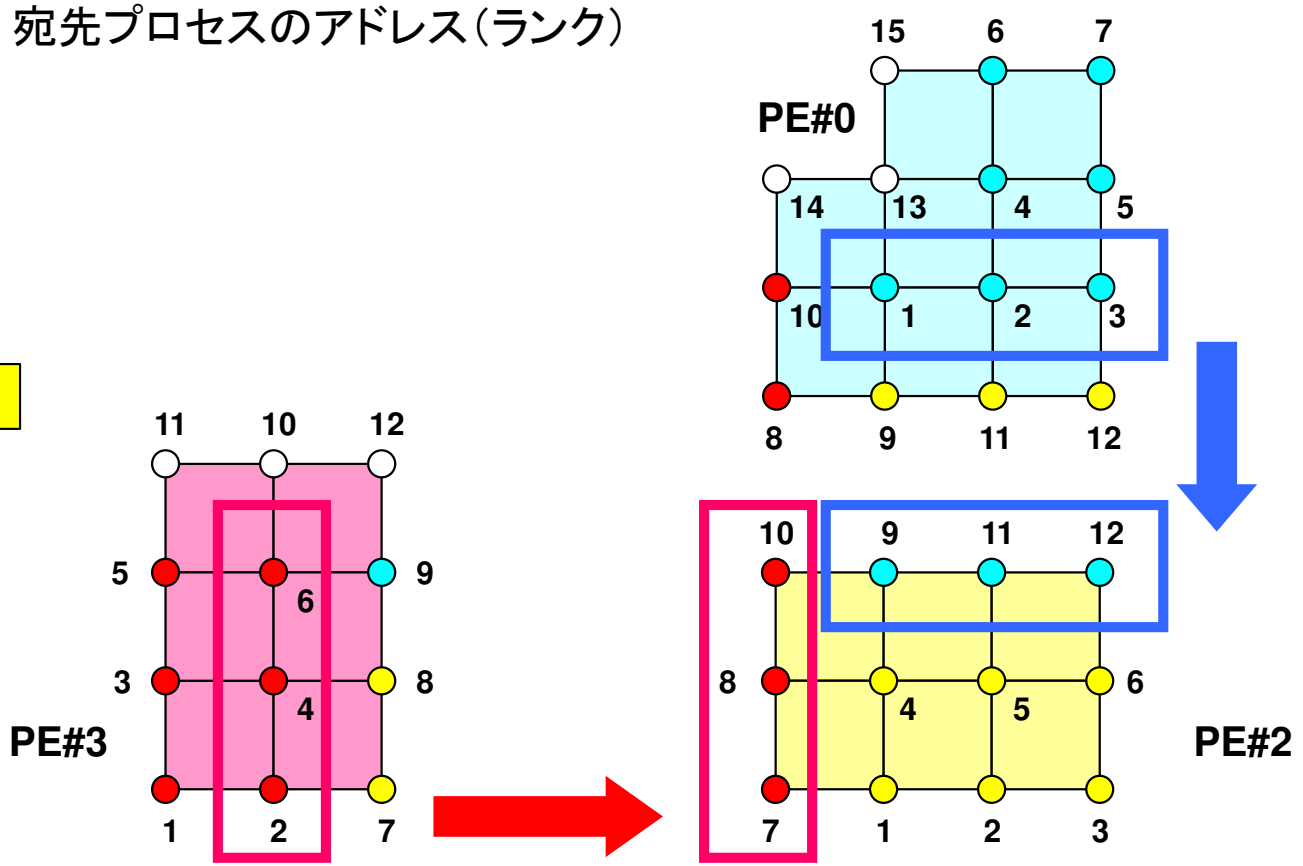
- MPI_SEND+MPI_RECV：結構制約は多いのでお勧めしない

- **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
    C言語については後述
  - **ierr** 整数 O 完了コード

RECV(受信):外点への受信

受信バッファに隣接プロセスから連続したデータを受け取る

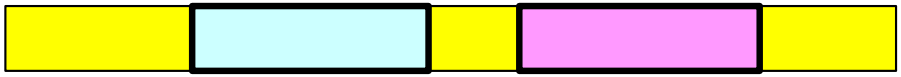
- **MPI_Irecv**
  (recvbuf,count,datatype,dest,tag,comm,request)
  - **recvbuf** 任意 I 受信バッファの先頭アドレス,
  - **count** 整数 I メッセージのサイズ
  - **datatype** 整数 I メッセージのデータタイプ
  - **dest** 整数 I 宛先プロセスのアドレス(ランク)

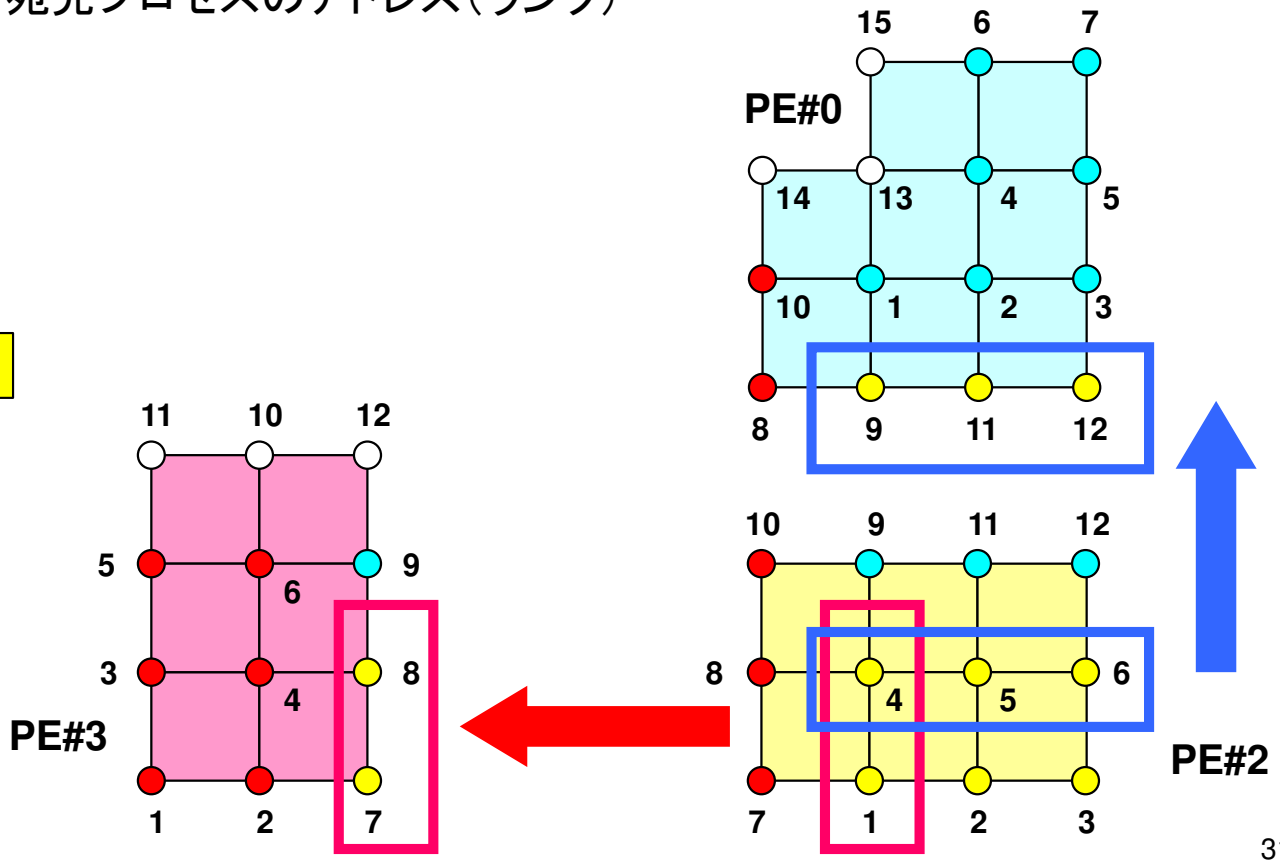


SEND（送信）：境界点の送信

送信バッファの連続したデータを隣接プロセスに送る

- **MPI_Isend**
  (sendbuf,count,datatype,dest,tag,comm,request)
  - **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）





通信識別子，状況オブジェクト配列の定義の仕方（FORTRAN）

- **MPI_Isend: request**
- **MPI_Irecv: request**
- **MPI_Waitall: request, status**

```
integer request(NEIBPETOT)
integer status (MPI_STAUTS_SIZE,NEIBPETOT)
```

- **MPI_Sendrecv: status**

```
integer status (MPI_STATUS_SIZE)
```

ファイルコピー・ディレクトリ確認

FORTRANユーザー

```
>$ cd /work/gt00/t00XXX/pFEM/
>$ module load fj
>$ cp /work/gt00/z30088/pFEM/F/s2-f.tar .
>$ tar xvf s2-f.tar
```

Cユーザー

```
>$ cd /work/gt00/t00XXX/pFEM/
>$ module load fj
>$ cp /work/gt00/home/z30088/pFEM/C/s2-c.tar .
>$ tar xvf s2-c.tar
```

ディレクトリ確認

```
>$ ls
    mpi
>$ cd mpi/S2
```

このディレクトリを本講義では <$O-S2> と呼ぶ。

<$O-S2> = <$O-TOP>/mpi/S2

利用例(1):スカラー送受信

- PE#0，PE#1間 で8バイト実数VALの値を交換する。

```
if (my_rank.eq.0) NEIB= 1
if (my_rank.eq.1) NEIB= 0

call MPI_Isend (VAL    ,1,MPI_DOUBLE_PRECISION,NEIB,…,req_send(1),…)
call MPI_Irecv (VALtemp,1,MPI_DOUBLE_PRECISION,NEIB,…,req_recv(1),…)
call MPI_Waitall (…,req_recv,stat_recv,…):受信バッファ VALtemp を利用可能
call MPI_Waitall (…,req_send,stat_send,…):送信バッファ VAL を変更可能
VAL= VALtemp
```

```
if (my_rank.eq.0) NEIB= 1
if (my_rank.eq.1) NEIB= 0

call MPI_Sendrecv (VAL    ,1,MPI_DOUBLE_PRECISION,NEIB,…              &
                   VALtemp,1,MPI_DOUBLE_PRECISION,NEIB,…, status,…)
VAL= VALtemp
```

受信バッファ名を「VAL」にしても動く場合はあるが，お勧めはしない。

利用例(1):スカラー送受信 FORTRAN

Isend/Irecv/Waitall

```
$> cd /work/gt00/t00XXX/pFEM/mpi/S2
$> module load fj
$> mpiftrpx -Kfast ex1-1.f
$> バッチジョブ実行(2プロセス) pjsub go2.sh
```

```
implicit REAL*8 (A-H,O-Z)
include  'mpif.h'
integer(kind=4) :: my_rank, PETOT, NEIB
real   (kind=8) :: VAL, VALtemp
integer(kind=4), dimension(MPI_STATUS_SIZE,1) :: stat_send, stat_recv
integer(kind=4), dimension(1)                 :: request_send, request_recv

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

if (my_rank.eq.0) then
  NEIB= 1
  VAL = 10.d0
 else
  NEIB= 0
  VAL = 11.d0
endif

call MPI_ISEND (VAL, 1,MPI_DOUBLE_PRECISION,NEIB,0,MPI_COMM_WORLD,request_send(1),ierr)
call MPI_IRECV (VALx,1,MPI_DOUBLE_PRECISION,NEIB,0,MPI_COMM_WORLD,request_recv(1),ierr)
call MPI_WAITALL (1, request_recv, stat_recv, ierr)
call MPI_WAITALL (1, request_send, stat_send, ierr)
VAL= VALx

call MPI_FINALIZE (ierr)
end
```

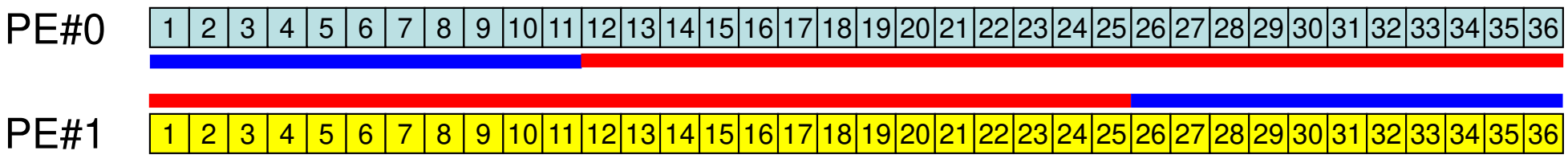
go2.sh

```
#!/bin/sh
#PJM -N "test"
#PJM -L rscgrp=tutorial-o
#PJM -L node=1
#PJM --mpi proc=2
#PJM -L elapse=00:15:00
#PJM -g gt00
#PJM -j
#PJM -e err
#PJM -o test.lst

module load fj
module load fjmpi

mpiexec ./a.out
```

利用例(2):配列の送受信(1/3)

- PE#0，PE#1間 で8バイト実数配列VECの値を交換する。
- PE#0⇒PE#1
  - PE#0:VEC(1)～VEC(11)の値を送る(長さ:11)
  - PE#1:VEV(26)~VEC(36)の値として受け取る
- PE#1⇒PE#0
  - PE#1:VEC(1)～VEC(25)の値を送る(長さ:25)
  - PE#0:VEV(12)~VEC(36)の値として受け取る

- 演習:プログラムを作成して見よう！



演習t1

演　習

- VEC(:)の初期状態を以下のようにする:
  - PE#0　VEC(1-36)= 101,102,103,~,135,136
  - PE#1　VEC(1-36)= 201,202,203,~,235,236

- 次ページのような結果になることを確認せよ

- MPI_Isend/Irecv/Waitall

予測される結果

演習t1

```
0 #BEFORE#  1      101.
0 #BEFORE#  2      102.
0 #BEFORE#  3      103.
0 #BEFORE#  4      104.
0 #BEFORE#  5      105.
0 #BEFORE#  6      106.
0 #BEFORE#  7      107.
0 #BEFORE#  8      108.
0 #BEFORE#  9      109.
0 #BEFORE# 10      110.
0 #BEFORE# 11      111.
0 #BEFORE# 12      112.
0 #BEFORE# 13      113.
0 #BEFORE# 14      114.
0 #BEFORE# 15      115.
0 #BEFORE# 16      116.
0 #BEFORE# 17      117.
0 #BEFORE# 18      118.
0 #BEFORE# 19      119.
0 #BEFORE# 20      120.
0 #BEFORE# 21      121.
0 #BEFORE# 22      122.
0 #BEFORE# 23      123.
0 #BEFORE# 24      124.
0 #BEFORE# 25      125.
0 #BEFORE# 26      126.
0 #BEFORE# 27      127.
0 #BEFORE# 28      128.
0 #BEFORE# 29      129.
0 #BEFORE# 30      130.
0 #BEFORE# 31      131.
0 #BEFORE# 32      132.
0 #BEFORE# 33      133.
0 #BEFORE# 34      134.
0 #BEFORE# 35      135.
0 #BEFORE# 36      136.
```

```
0 #AFTER #  1      101.
0 #AFTER #  2      102.
0 #AFTER #  3      103.
0 #AFTER #  4      104.
0 #AFTER #  5      105.
0 #AFTER #  6      106.
0 #AFTER #  7      107.
0 #AFTER #  8      108.
0 #AFTER #  9      109.
0 #AFTER # 10      110.
0 #AFTER # 11      111.
0 #AFTER # 12      201.
0 #AFTER # 13      202.
0 #AFTER # 14      203.
0 #AFTER # 15      204.
0 #AFTER # 16      205.
0 #AFTER # 17      206.
0 #AFTER # 18      207.
0 #AFTER # 19      208.
0 #AFTER # 20      209.
0 #AFTER # 21      210.
0 #AFTER # 22      211.
0 #AFTER # 23      212.
0 #AFTER # 24      213.
0 #AFTER # 25      214.
0 #AFTER # 26      215.
0 #AFTER # 27      216.
0 #AFTER # 28      217.
0 #AFTER # 29      218.
0 #AFTER # 30      219.
0 #AFTER # 31      220.
0 #AFTER # 32      221.
0 #AFTER # 33      222.
0 #AFTER # 34      223.
0 #AFTER # 35      224.
0 #AFTER # 36      225.
```

```
1 #BEFORE#  1      201.
1 #BEFORE#  2      202.
1 #BEFORE#  3      203.
1 #BEFORE#  4      204.
1 #BEFORE#  5      205.
1 #BEFORE#  6      206.
1 #BEFORE#  7      207.
1 #BEFORE#  8      208.
1 #BEFORE#  9      209.
1 #BEFORE# 10      210.
1 #BEFORE# 11      211.
1 #BEFORE# 12      212.
1 #BEFORE# 13      213.
1 #BEFORE# 14      214.
1 #BEFORE# 15      215.
1 #BEFORE# 16      216.
1 #BEFORE# 17      217.
1 #BEFORE# 18      218.
1 #BEFORE# 19      219.
1 #BEFORE# 20      220.
1 #BEFORE# 21      221.
1 #BEFORE# 22      222.
1 #BEFORE# 23      223.
1 #BEFORE# 24      224.
1 #BEFORE# 25      225.
1 #BEFORE# 26      226.
1 #BEFORE# 27      227.
1 #BEFORE# 28      228.
1 #BEFORE# 29      229.
1 #BEFORE# 30      230.
1 #BEFORE# 31      231.
1 #BEFORE# 32      232.
1 #BEFORE# 33      233.
1 #BEFORE# 34      234.
1 #BEFORE# 35      235.
1 #BEFORE# 36      236.
```

```
1 #AFTER #  1      201.
1 #AFTER #  2      202.
1 #AFTER #  3      203.
1 #AFTER #  4      204.
1 #AFTER #  5      205.
1 #AFTER #  6      206.
1 #AFTER #  7      207.
1 #AFTER #  8      208.
1 #AFTER #  9      209.
1 #AFTER # 10      210.
1 #AFTER # 11      211.
1 #AFTER # 12      212.
1 #AFTER # 13      213.
1 #AFTER # 14      214.
1 #AFTER # 15      215.
1 #AFTER # 16      216.
1 #AFTER # 17      217.
1 #AFTER # 18      218.
1 #AFTER # 19      219.
1 #AFTER # 20      220.
1 #AFTER # 21      221.
1 #AFTER # 22      222.
1 #AFTER # 23      223.
1 #AFTER # 24      224.
1 #AFTER # 25      225.
1 #AFTER # 26      101.
1 #AFTER # 27      102.
1 #AFTER # 28      103.
1 #AFTER # 29      104.
1 #AFTER # 30      105.
1 #AFTER # 31      106.
1 #AFTER # 32      107.
1 #AFTER # 33      108.
1 #AFTER # 34      109.
1 #AFTER # 35      110.
1 #AFTER # 36      111.
```

利用例(2):配列の送受信(2/3)

演習t1

```
if (my_rank.eq.0) then
  call MPI_Isend (VEC( 1),11,MPI_DOUBLE_PRECISION,1,…,req_send(1),…)
  call MPI_Irecv (VEC(12),25,MPI_DOUBLE_PRECISION,1,…,req_recv(1),…)
endif

if (my_rank.eq.1) then
  call MPI_Isend (VEC( 1),25,MPI_DOUBLE_PRECISION,0,…,req_send(1),…)
  call MPI_Irecv (VEC(26),11,MPI_DOUBLE_PRECISION,0,…,req_recv(1),…)
endif

call MPI_Waitall (…,req_recv,stat_recv,…)
call MPI_Waitall (…,req_send,stat_send,…)
```

これでも良いが，操作が煩雑
SPMDらしくない
汎用性が無い

利用例(2):配列の送受信(3/3)

演習t1

```
if (my_rank.eq.0) then
  NEIB= 1
   start_send=   1
  length_send=  11
   start_recv=  length_send + 1
  length_recv=  25
endif

if (my_rank.eq.1) then
  NEIB= 0
   start_send=  1
  length_send= 25
   start_recv= length_send + 1
  length_recv= 11
endif

call MPI_Isend                                                    &
(VEC(start_send),length_send,MPI_DOUBLE_PRECISION,NEIB,…,req_send(1),…)
call MPI_Irecv                                                    &
(VEC(start_recv),length_recv,MPI_DOUBLE_PRECISION,NEIB,…,req_recv(1),…)

call MPI_Waitall (…,req_recv,stat_recv,…)
call MPI_Waitall (…,req_send,stat_send,…)
```

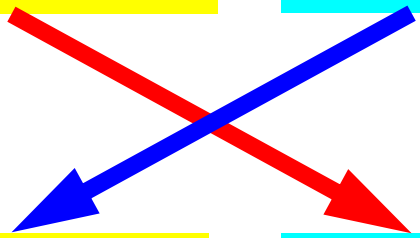
一気にSPMDらしくなる

演習t1

配列の送受信:注意

```
#PE0
  send:
    VEC(start_send)~
    VEC(start_send+length_send-1)
```

```
#PE1
  send:
    VEC(start_send)~
    VEC(start_send+length_send-1)
```



```
#PE0
  recv:
    VEC(start_recv)~
    VEC(start_recv+length_recv-1)
```

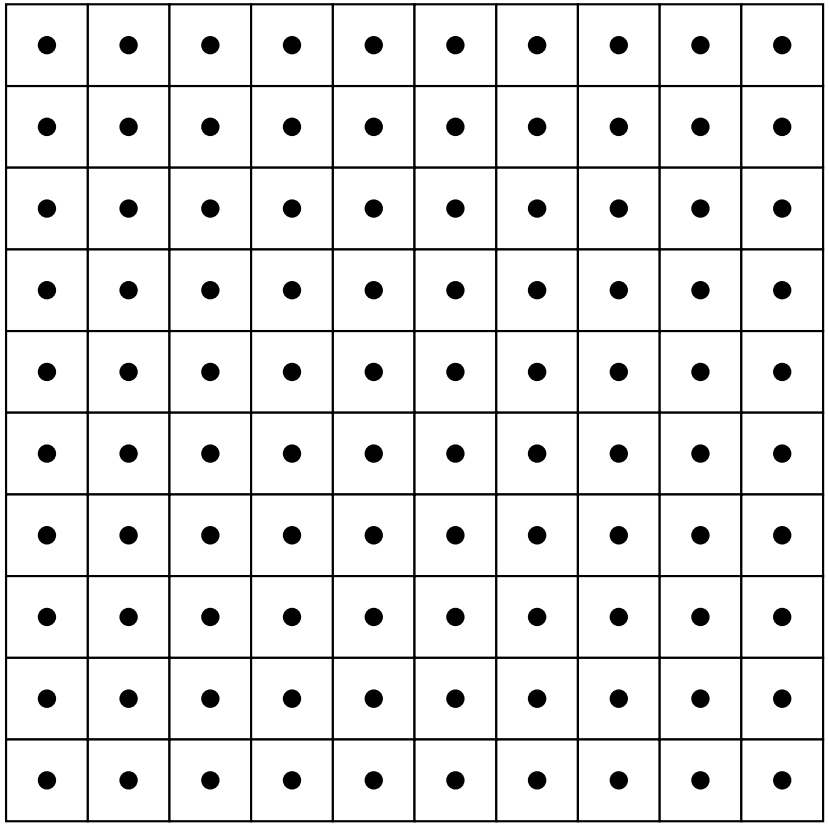
```
#PE1
  recv:
    VEC(start_recv)~
    VEC(start_recv+length_recv-1)
```

- 送信側の「length_send」と受信側の「length_recv」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

1対1通信

- 1対1通信とは　？
- 二次元問題，一般化された通信テーブル
  - 二次元差分法
  - 問題設定
  - 局所データ構造と通信テーブル
  - 実装例
- 課題S2

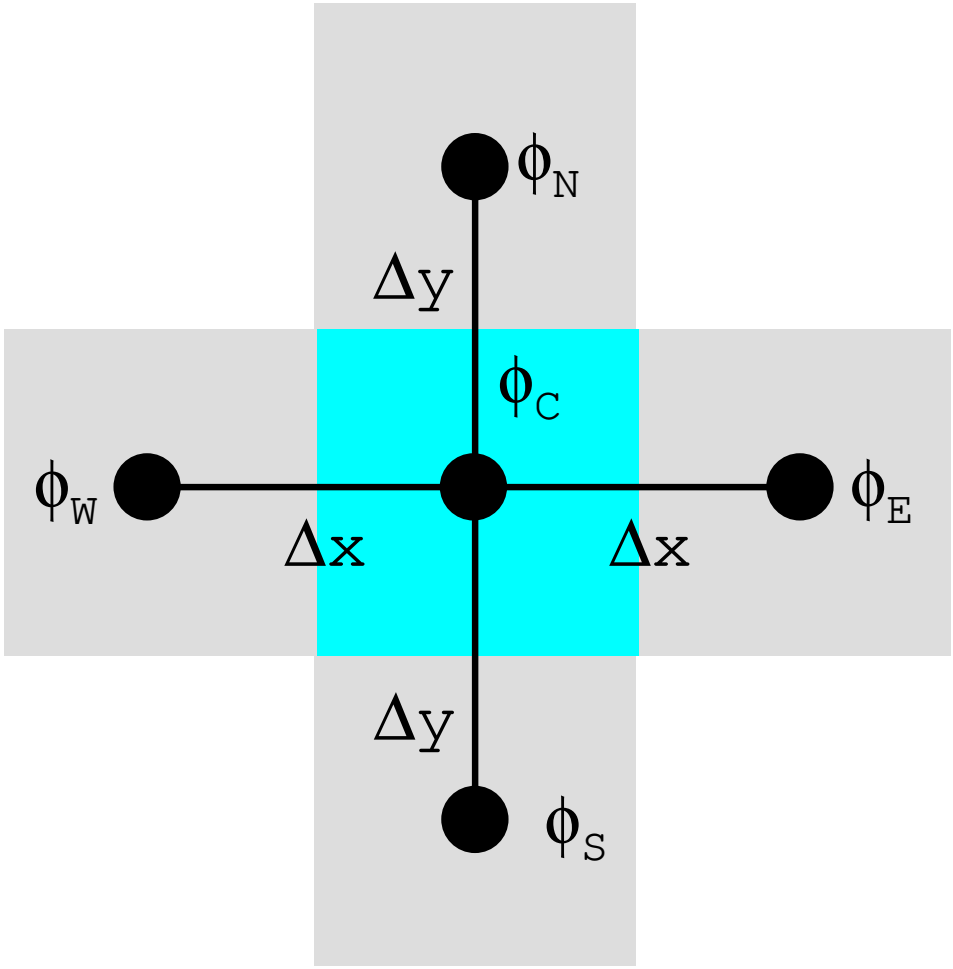
二次元差分法(1/5)
全体メッシュ



二次元中央差分法(5点差分法)の定式化

$$\frac{\partial^2 \phi}{\partial x^2} + \frac{\partial^2 \phi}{\partial y^2} = f$$

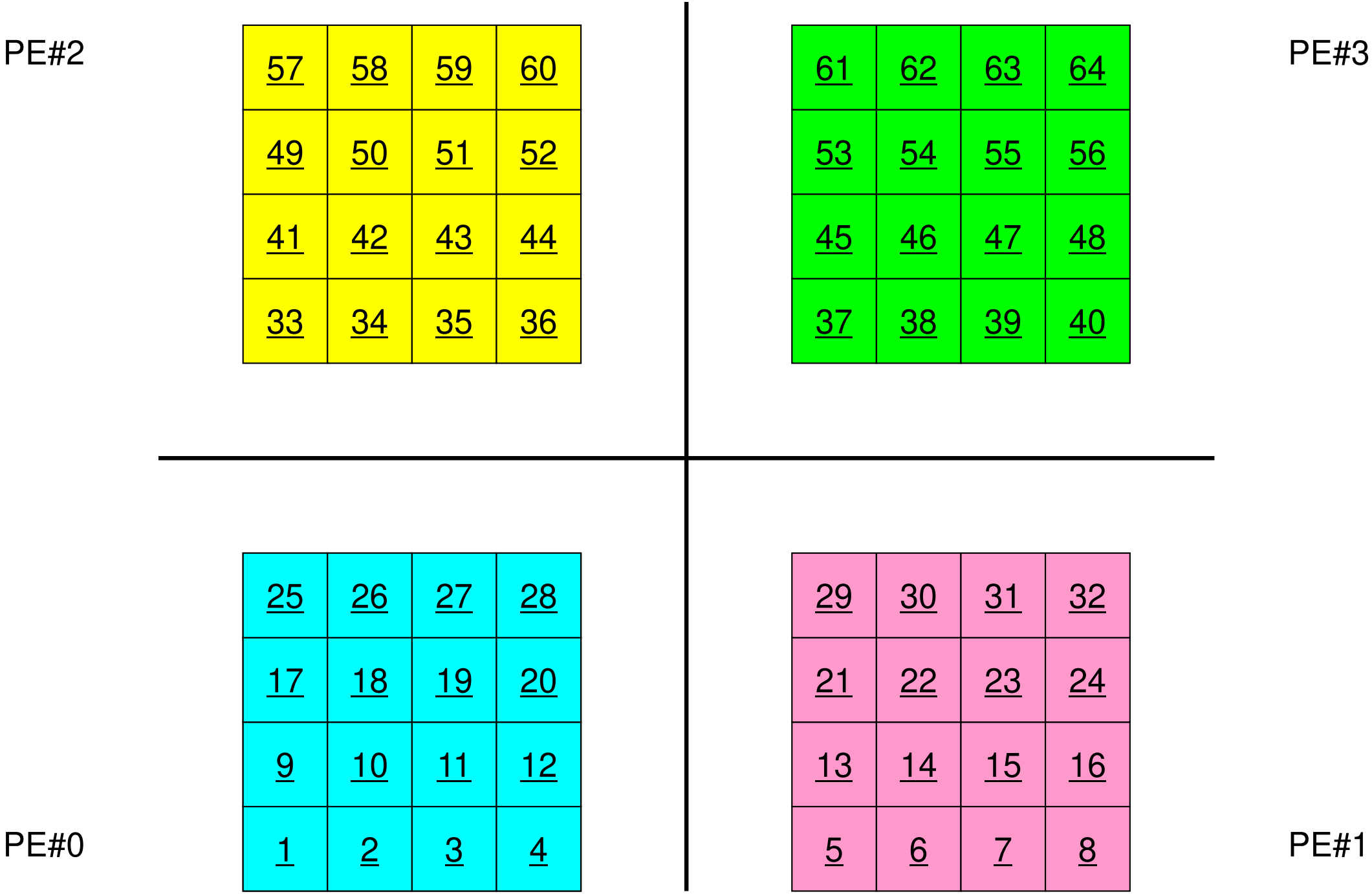
$$\left(\frac{\phi_E - 2\phi_C + \phi_W}{\Delta x^2} \right) + \left(\frac{\phi_N - 2\phi_C + \phi_S}{\Delta y^2} \right) = f_C$$



4領域に分割

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

4領域に分割：全体番号


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
PE#0
25 26 27 28
17 18 19 20
9 10 11 12
1 2 3 4
PE#1
29 30 31 32
21 22 23 24
13 14 15 16
5 6 7 8

4領域に分割：局所番号

PE#2

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

PE#3

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

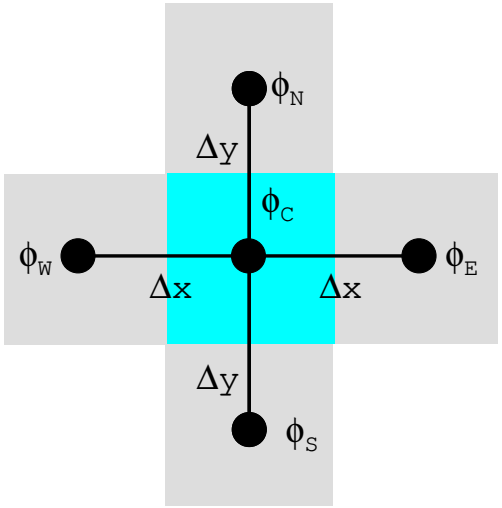
PE#0

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

PE#1

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

オーバーラップ領域の値が必要：外点



PE#2

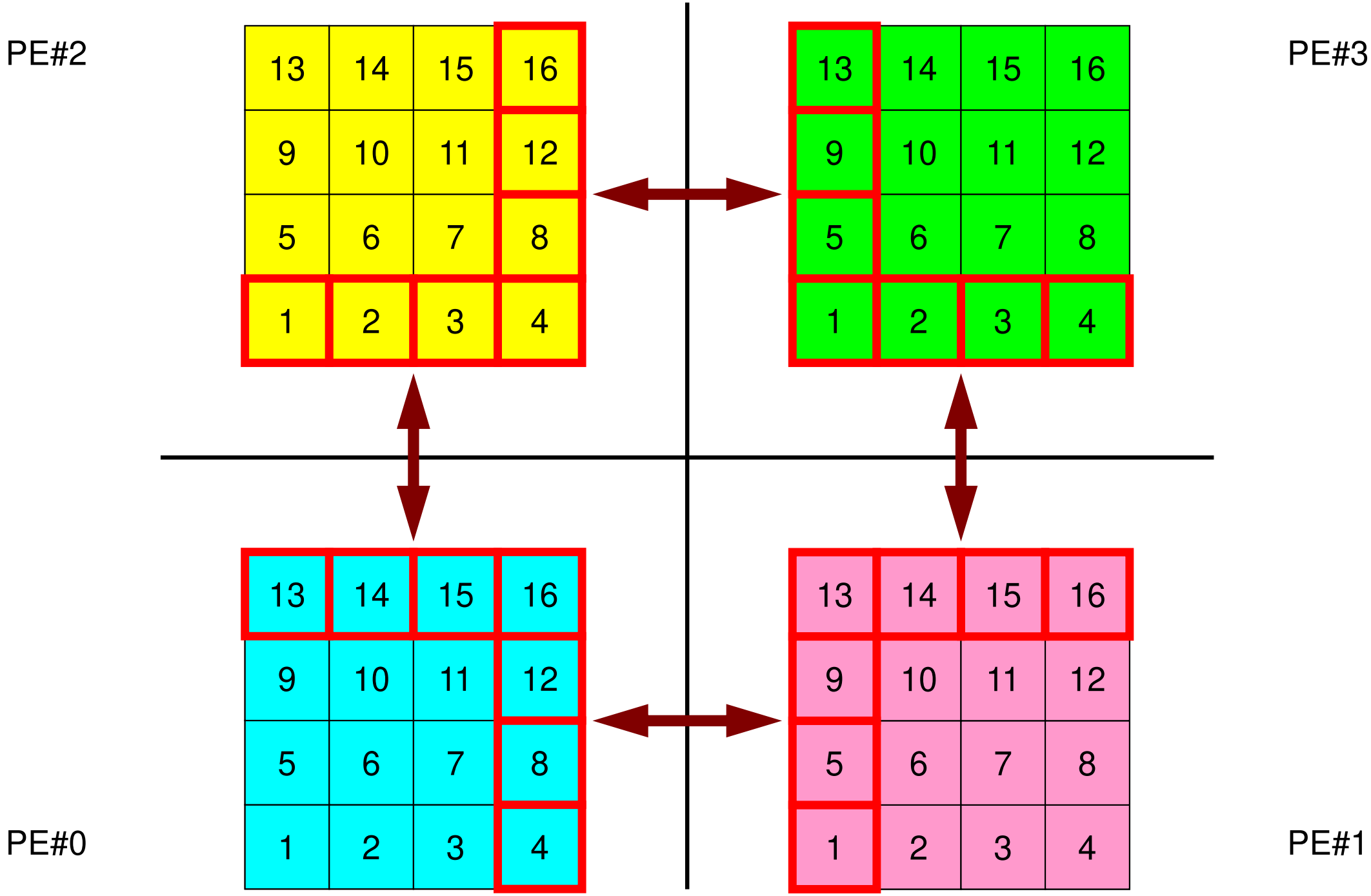
PE#3

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

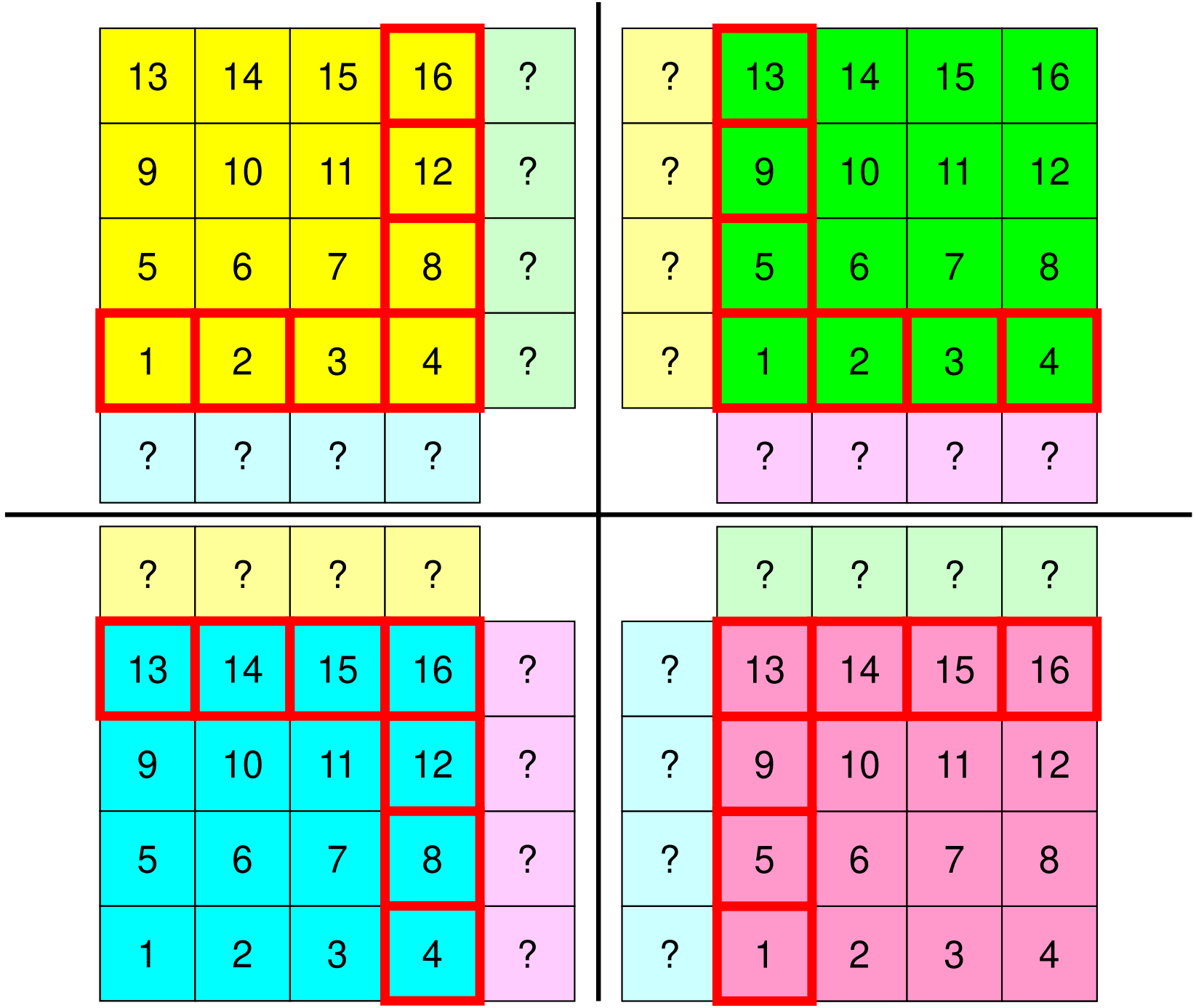
PE#0

PE#1

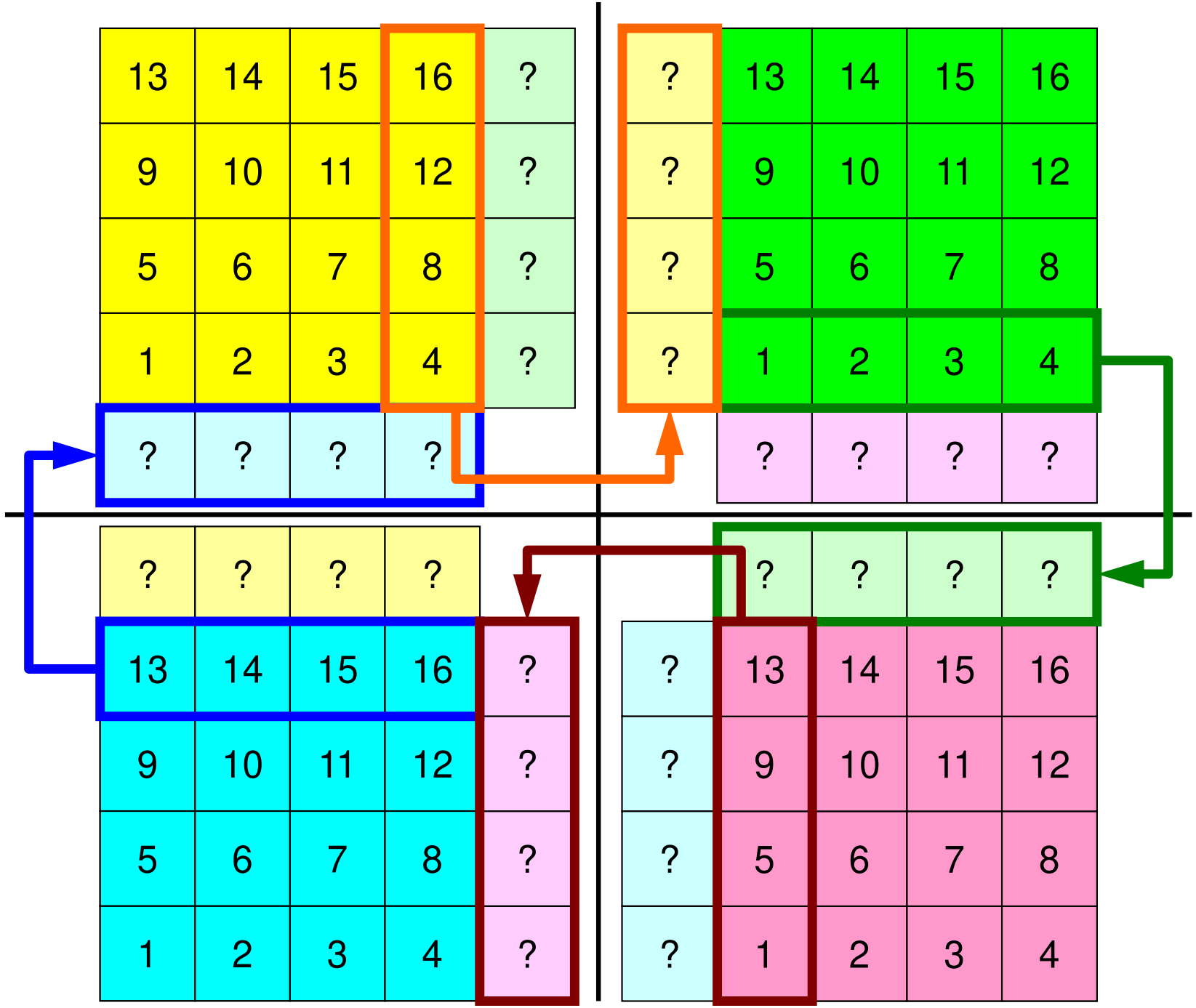
オーバーラップ領域の値が必要：外点



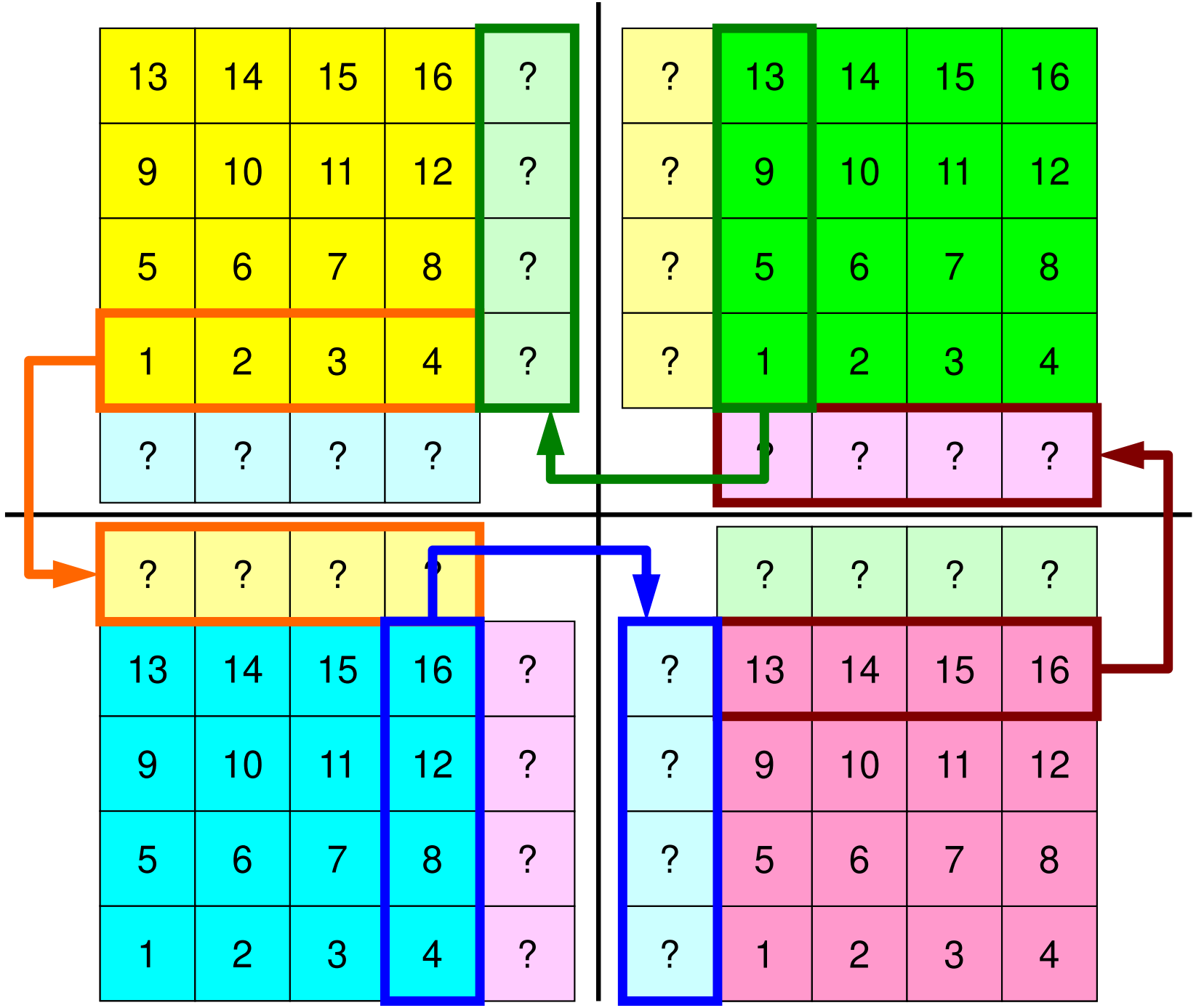
外点の局所番号はどうする?



オーバーラップ領域の値が必要



オーバーラップ領域の値が必要



1対1通信

- 1対1通信とは　？
- 二次元問題，一般化された通信テーブル
  - 二次元差分法
  - 問題設定
  - 局所データ構造と通信テーブル
  - 実装例
- 課題S2

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

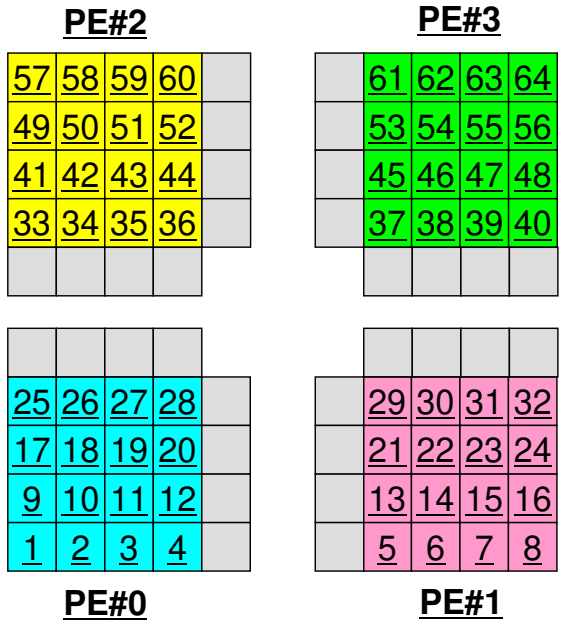
25	26	27	28
17	18	19	20
9	10	11	12
1	2	3	4

PE#0

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

PE#1

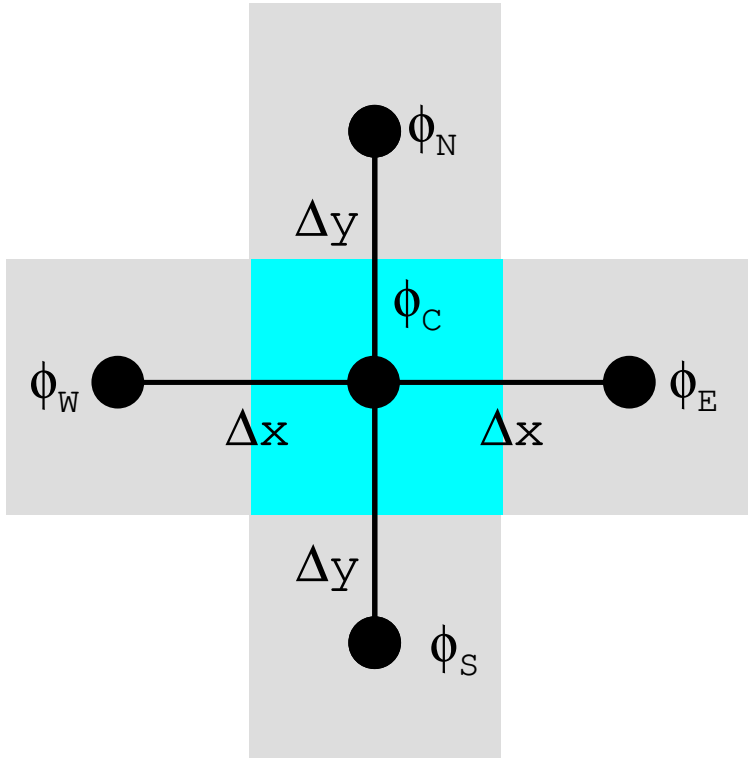
- 左記のような4領域に分割された二次元領域において，外点の情報（全体要素番号）を隣接領域から受信する方法
  - □はPE#0が受信する情報



二次元差分法のオペレーション

$$\frac{\partial^2 \phi}{\partial x^2} + \frac{\partial^2 \phi}{\partial y^2} = f$$

$$\left(\frac{\phi_E - 2\phi_C + \phi_W}{\Delta x^2} \right) + \left(\frac{\phi_N - 2\phi_C + \phi_S}{\Delta y^2} \right) = f_C$$

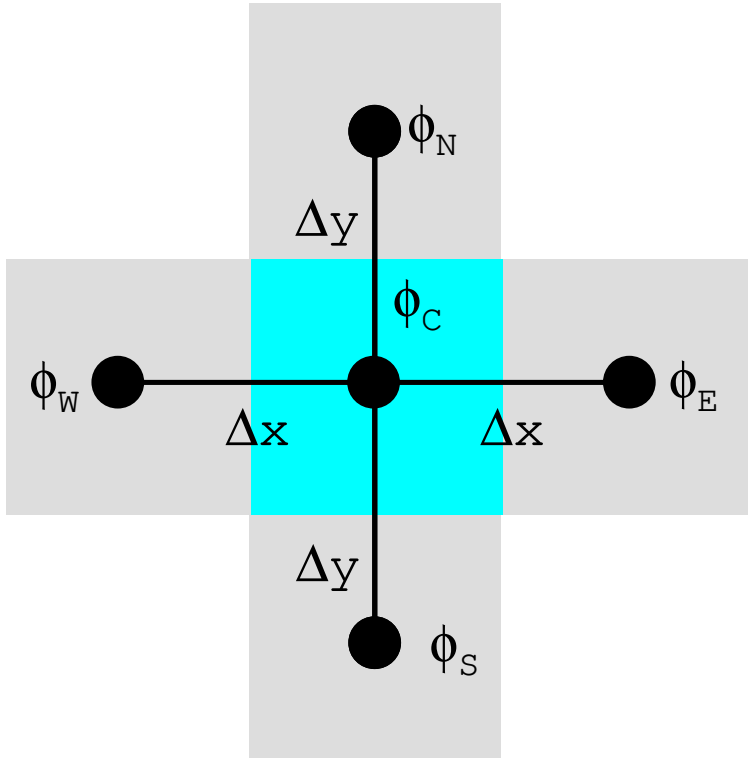


57	58	59	60	61	62	63	64
49	50	51	52	53	54	55	56
41	42	43	44	45	46	47	48
33	34	35	36	37	38	39	40
25	26	27	28	29	30	31	32
17	18	19	20	21	22	23	24
9	10	11	12	13	14	15	16
1	2	3	4	5	6	7	8

二次元差分法のオペレーション

$$\frac{\partial^2 \phi}{\partial x^2} + \frac{\partial^2 \phi}{\partial y^2} = f$$

$$\left(\frac{\phi_E - 2\phi_C + \phi_W}{\Delta x^2} \right) + \left(\frac{\phi_N - 2\phi_C + \phi_S}{\Delta y^2} \right) = f_C$$



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

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

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

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

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

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: ?

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

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: ?

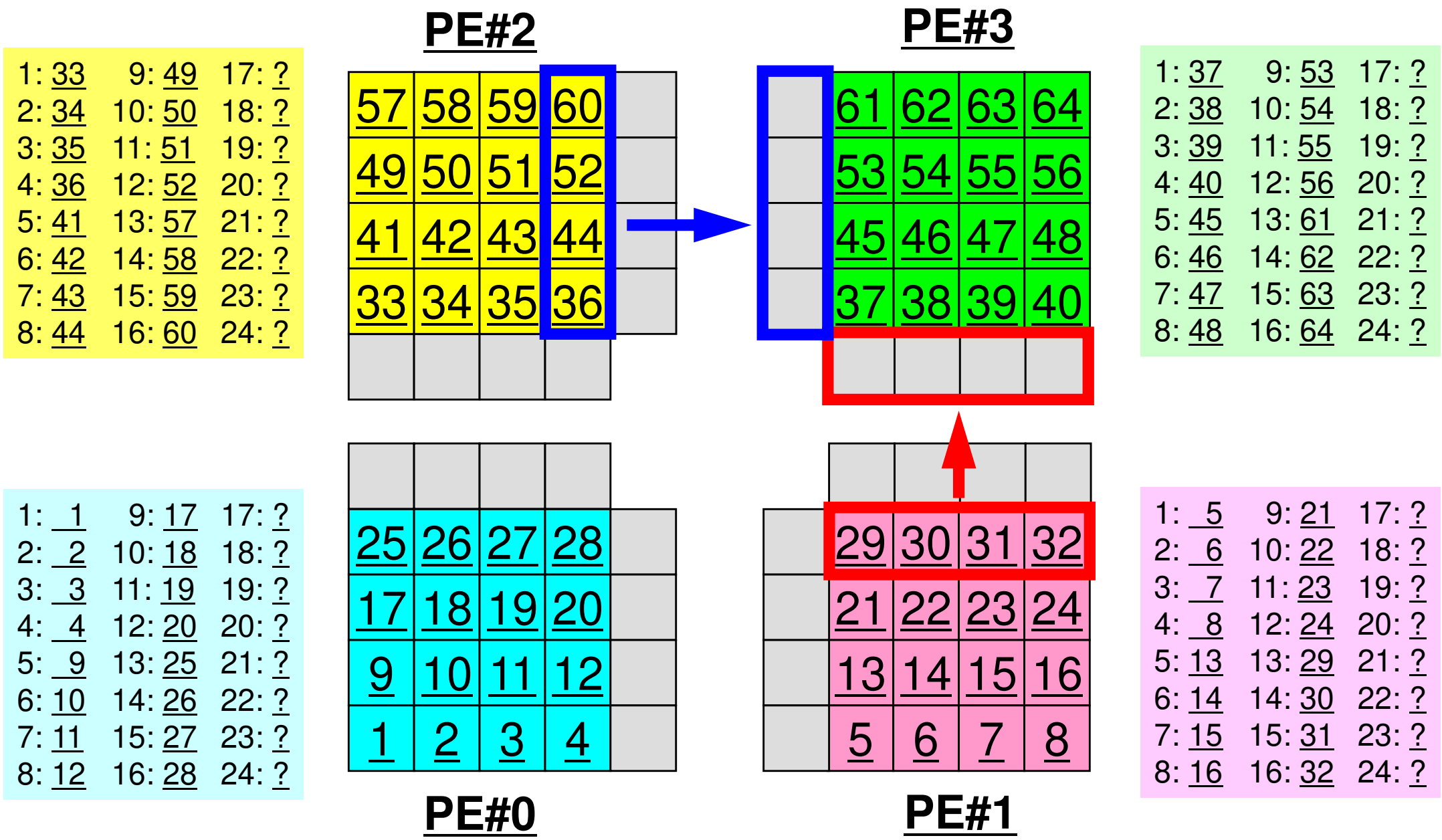
PE#0

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

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: ?

PE#1

演算内容(2/3):送信，受信前



演算内容(3/3):送信，受信後

PE#2

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

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

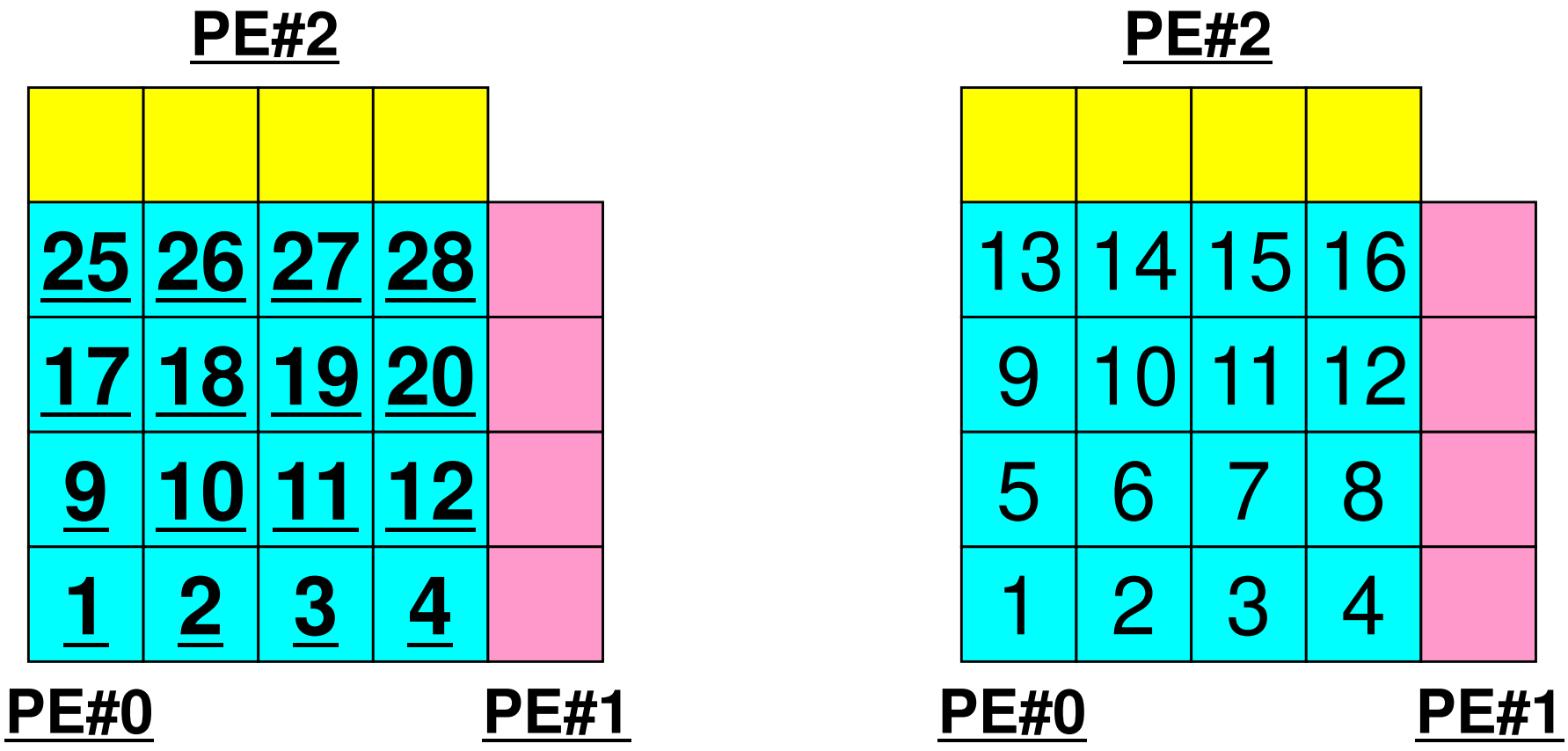
1: 5	9: 21	17: 4
2: 6	10: 22	18: 12
3: 7	11: 23	19: 20
4: 8	12: 24	20: 28
5: 13	13: 29	21: 37
6: 14	14: 30	22: 38
7: 15	15: 31	23: 39
8: 16	16: 32	24: 40

1対1通信

- 1対1通信とは　？
- 二次元問題，一般化された通信テーブル
  - 二次元差分法
  - 問題設定
  - 局所データ構造と通信テーブル
  - 実装例
- 課題S2

各領域データ(局所分散データ)仕様

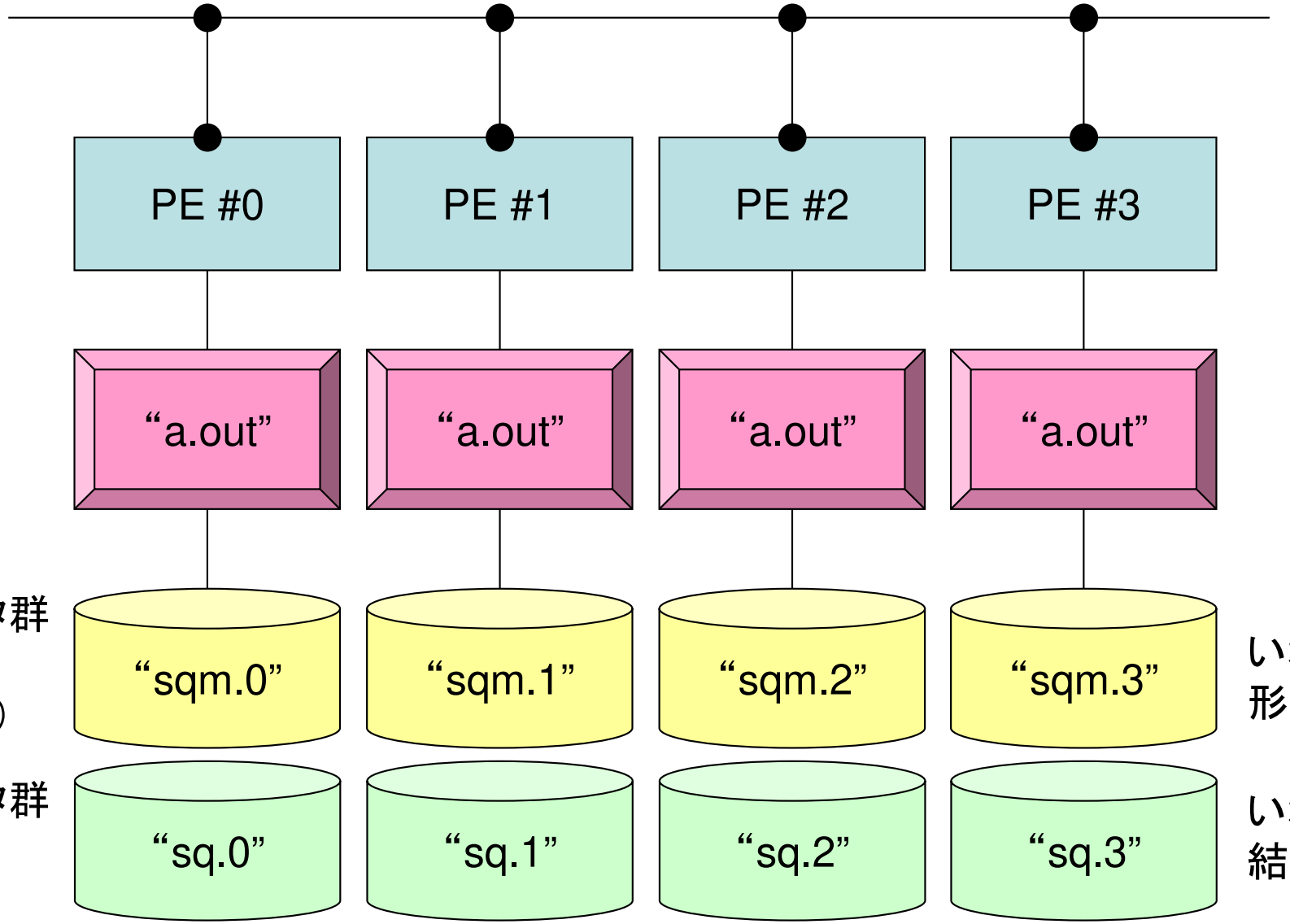
PE#0における局所分散データ



各要素における値(全体番号)

局所番号

SPMD・・・



局所分散データ群
（隣接領域，
通信テーブル）

局所分散データ群
（内点の全体
要素番号）

いわゆる
形状データ

いわゆる
結果データ

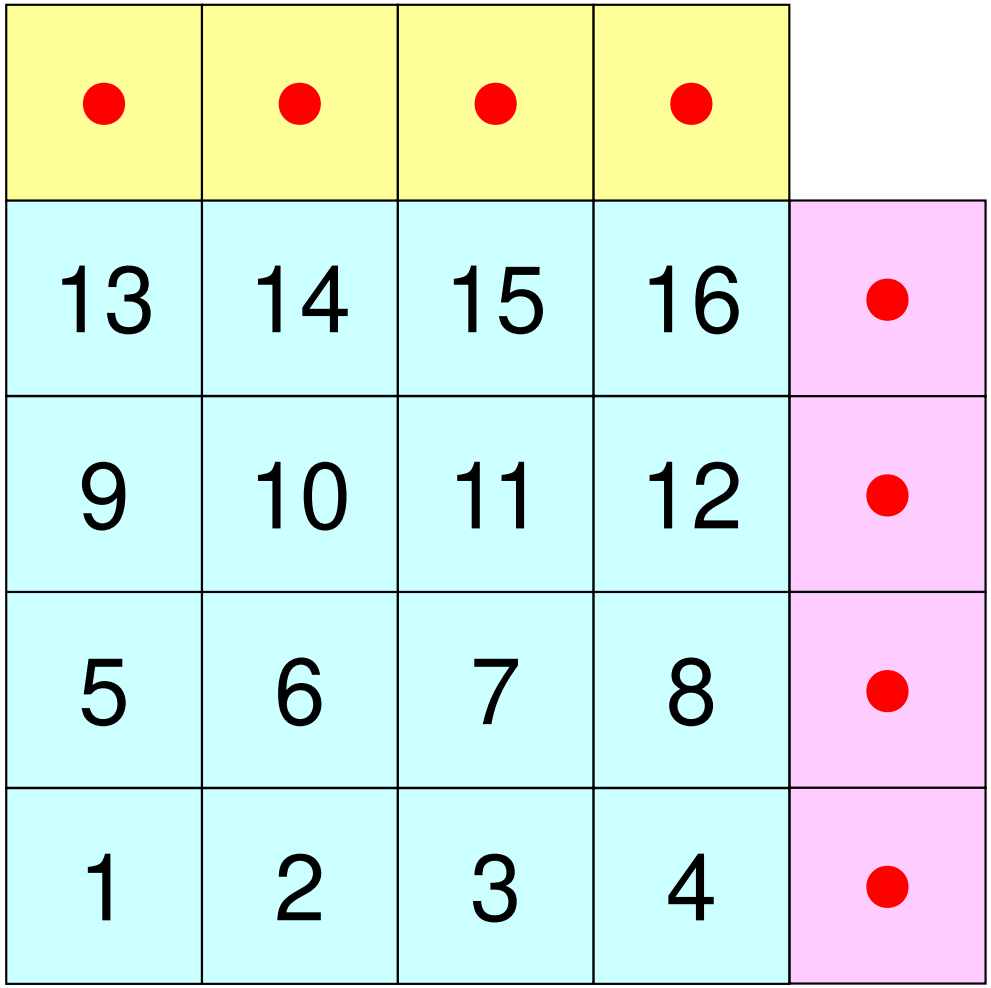
二次元差分法：PE#0
各領域に必要な情報（1/4）

内点（Internal Points）
その領域にアサインされた要素

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

二次元差分法：PE#0
各領域に必要な情報（2/4）

PE#2



PE#1

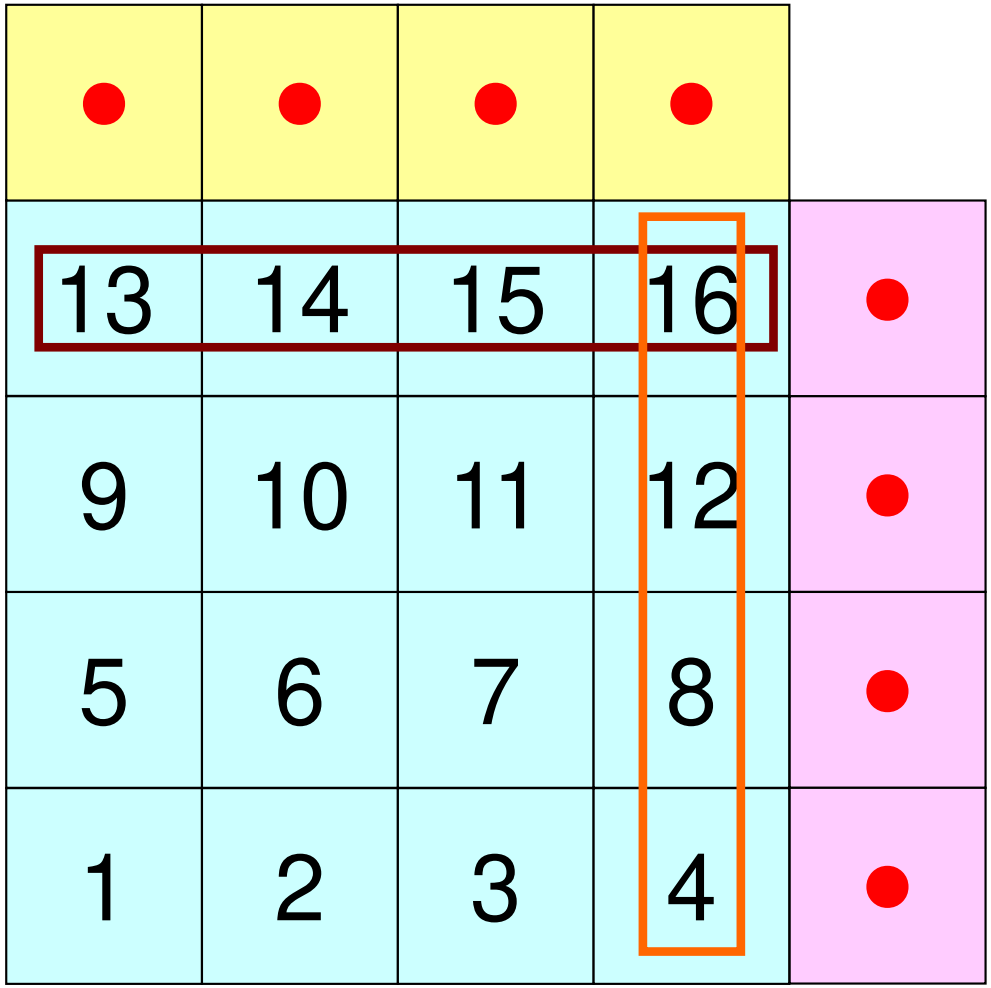
内点（Internal Points）
その領域にアサインされた要素

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））

二次元差分法：PE#0
各領域に必要な情報（4/4）

PE#2



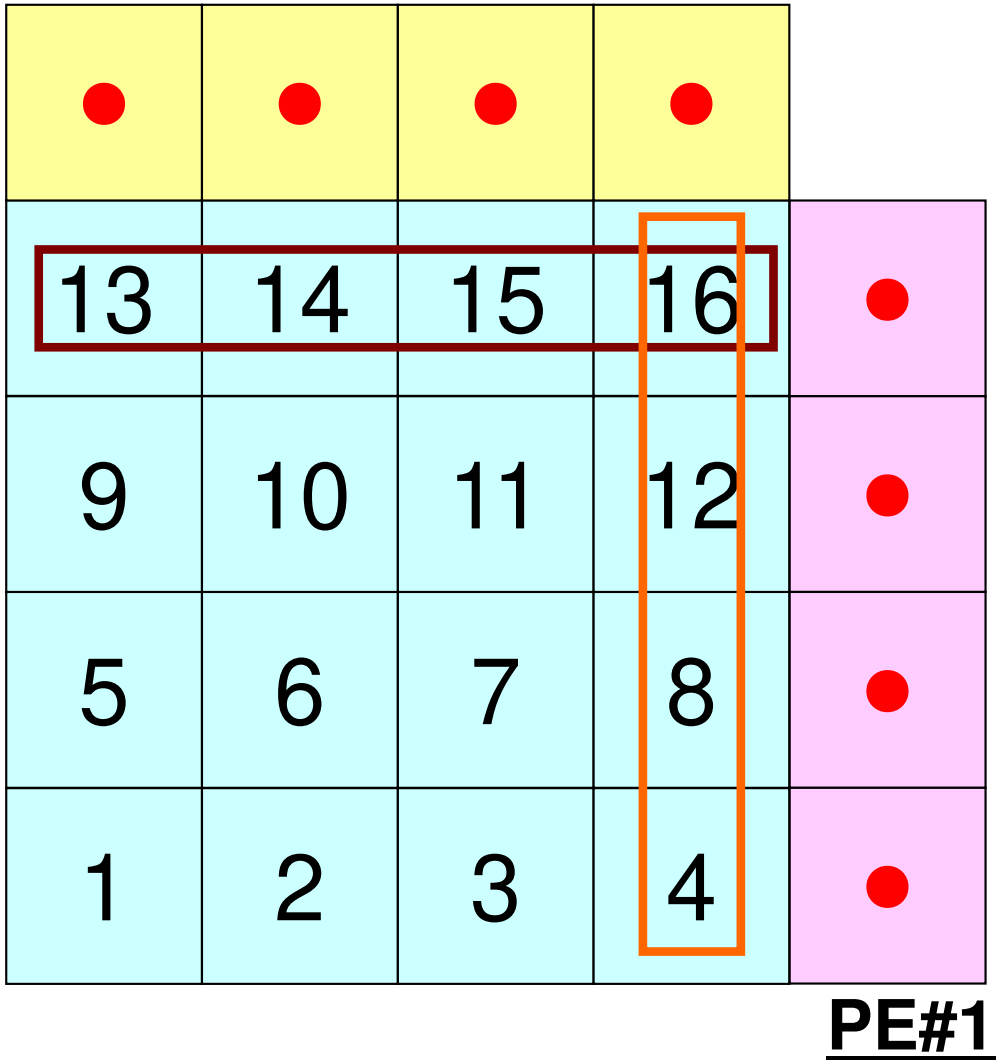
PE#1

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

二次元差分法：PE#0
各領域に必要な情報（4/4）



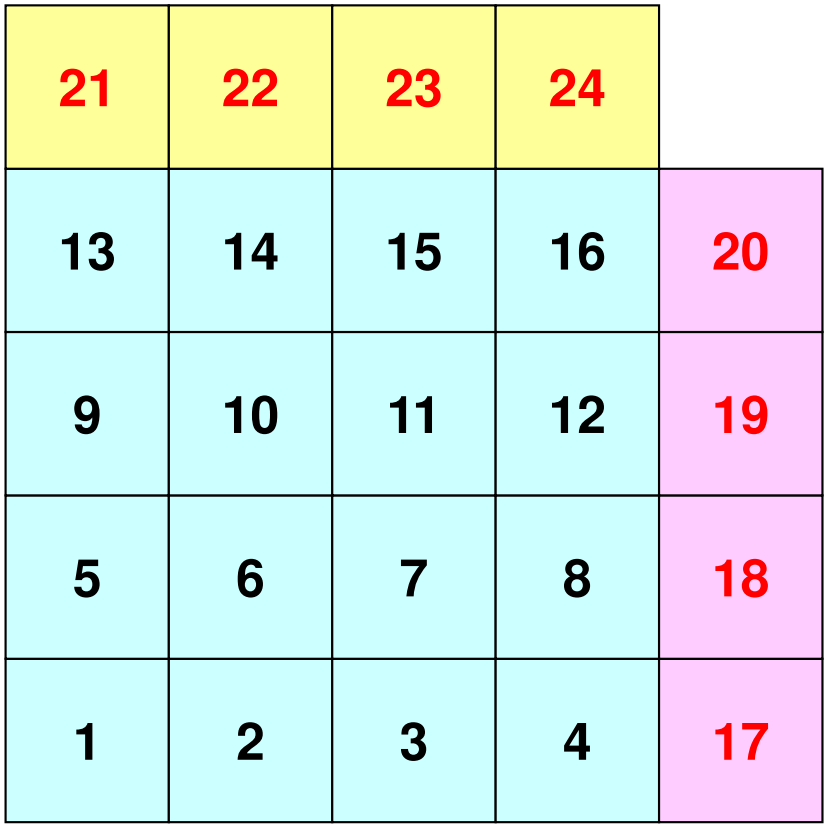
内点（Internal Points）
その領域にアサインされた要素

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

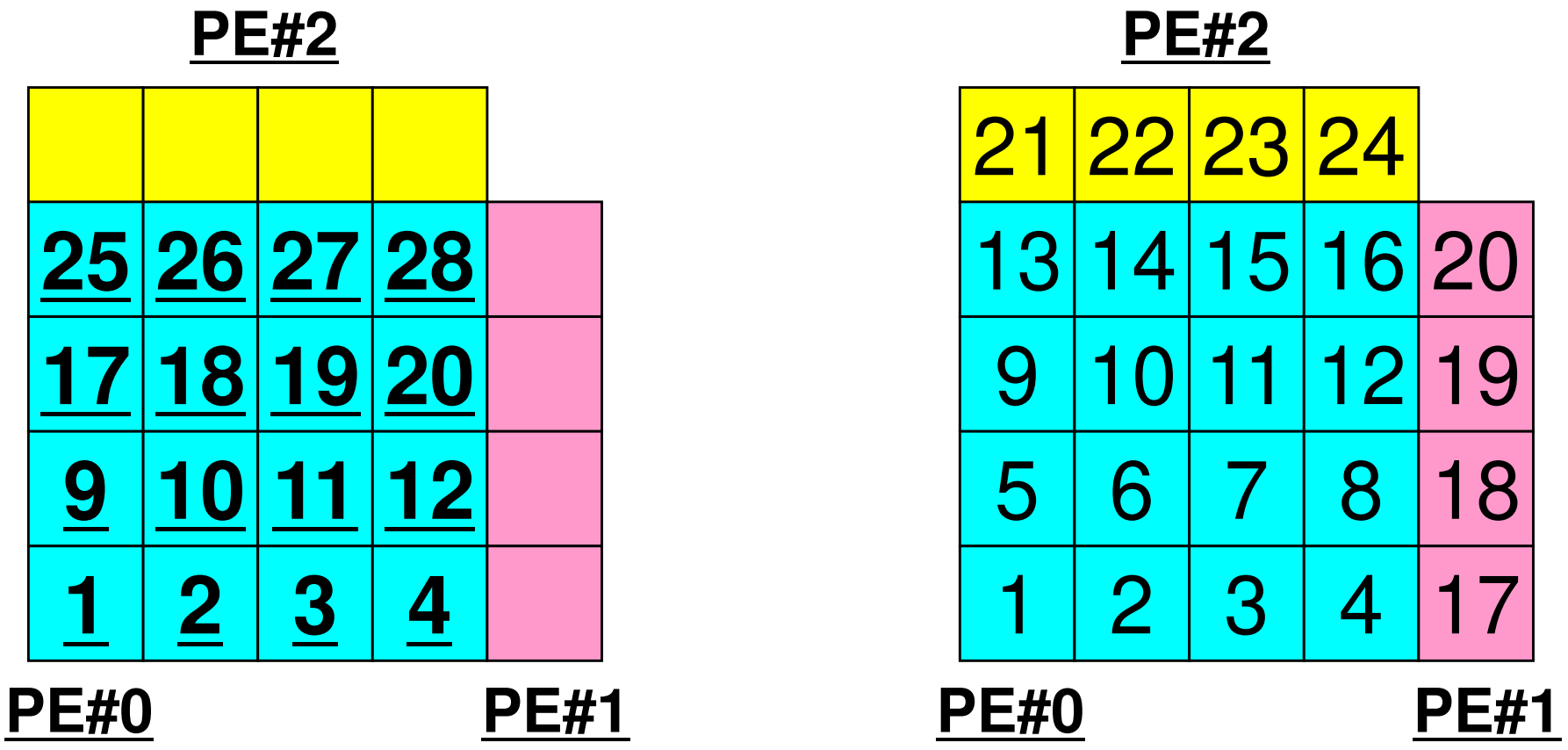
各領域データ(局所データ)仕様



- 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

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

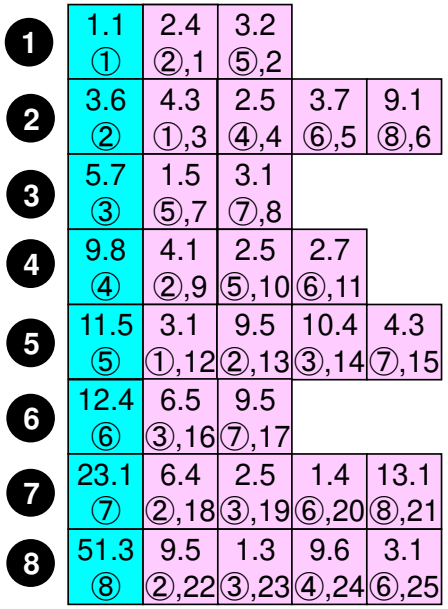


各要素における値(全体番号)

局所番号

一般化された通信テーブル：送信

Fortran

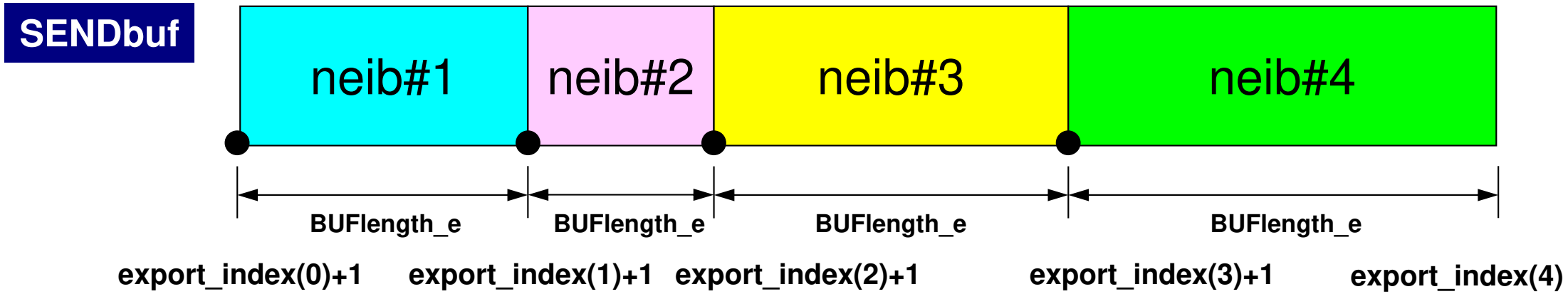


`Diag (i)`	Diagonal Components (REAL, i=1~N)
`Index(i)`	Number of Non-Zero Off-Diagonals at Each ROW (INT, i=0~N)
`Item(k)`	Off-Diagonal Components (Corresponding Column ID) (INT, k=1, index(N))
`AMat(k)`	Off-Diagonal Components (Value) (REAL, k=1, index(N))

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

do i= 1, N
  Y(i)= D(i)*X(i)
  do k= index(i-1)+1, index(i)
    Y(i)= Y(i) + AMAT(k)*X(item(k))
  enddo
enddo
```

- 送信相手
  - NEIBPETOT, NEIBPE(neib)
- それぞれの送信相手に送るメッセージサイズ
  - export_index(neib), neib= 0, 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)

送信（MPI_Isend/Irecv/Waitall） Fortran




```
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_send, 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ことを勧める。

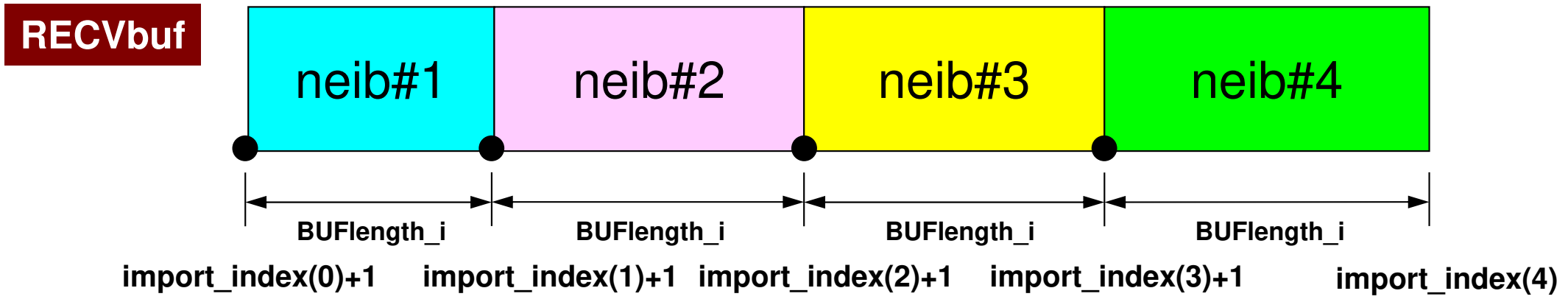
Fortran

一般化された通信テーブル：受信

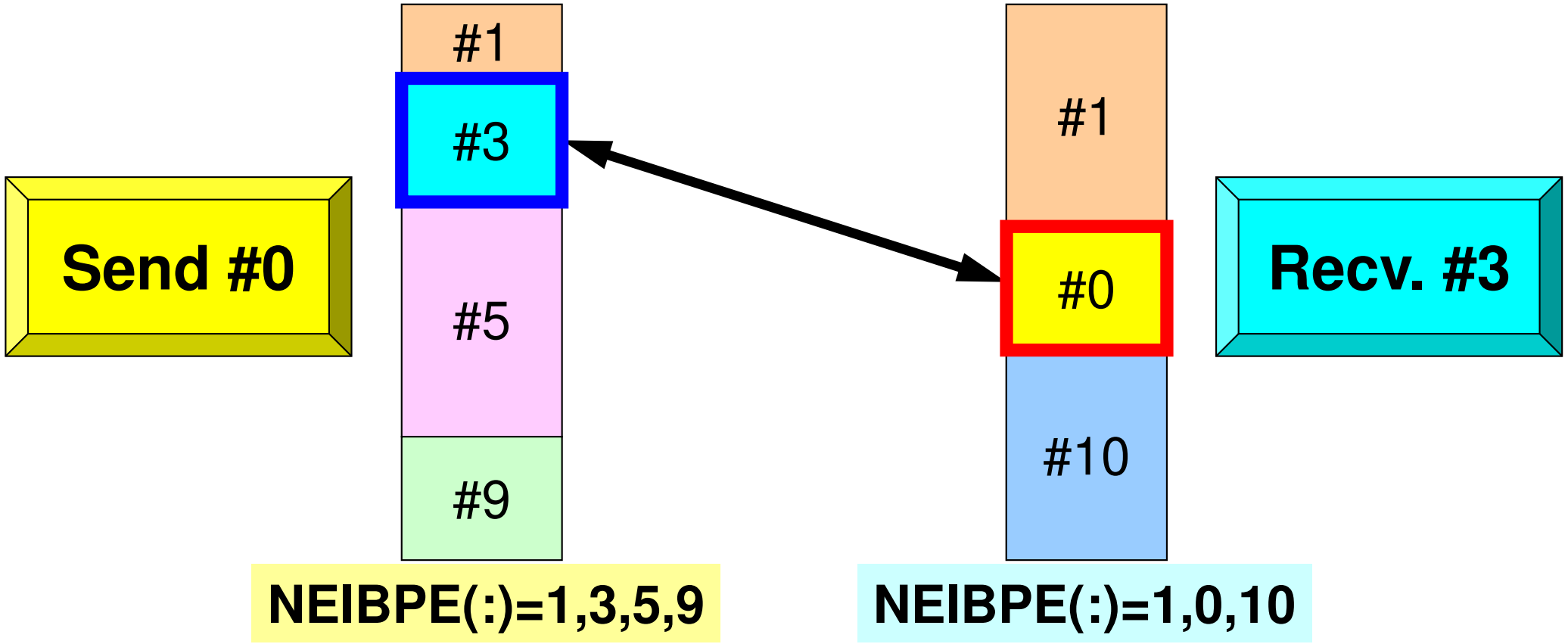
- 受信相手
  - NEIBPETOT，NEIBPE(neib)
- それぞれの受信相手から受け取るメッセージサイズ
  - import_index(neib), neib= 0, NEIBPETOT
- 「外点」番号
  - import_item(k), k= 1, import_index(NEIBPETOT)
- それぞれの受信相手から受け取るメッセージ
  - RECVbuf(k), k= 1, import_index(NEIBPETOT)

受信（MPI_Isend/Irecv/Waitall） Fortran

```
 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)                                      受信バッファから代入
     enddo
  enddo
```



SEND/RECVの関係 (#0⇒#3)



- 送信元（sources）/受信先（destinations），メッセージサイズ，メッセージの中身の整合性（consistency）
- 隣接プロセスID（NEIBPE（neib））とtag（=0）が整合すると通信発生

送信と受信の関係

```
 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
```

```
 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
```

- 送信元・受信先プロセス番号，メッセージサイズ，内容の整合性　！
- NEIBPE（neib）がマッチしたときに通信が起こる。

一般化された通信テーブル(1/6)

PE#2

PE#1

```
#NEIBPEtot
2
#NEIBPE
1  2
#NODE
24 16
#IMPORT_index
4   8
#IMPORT_items
17
18
19
20
21
22
23
24
#EXPORT_index
4   8
#EXPORT_items
4
8
12
16
13
14
15
16
```

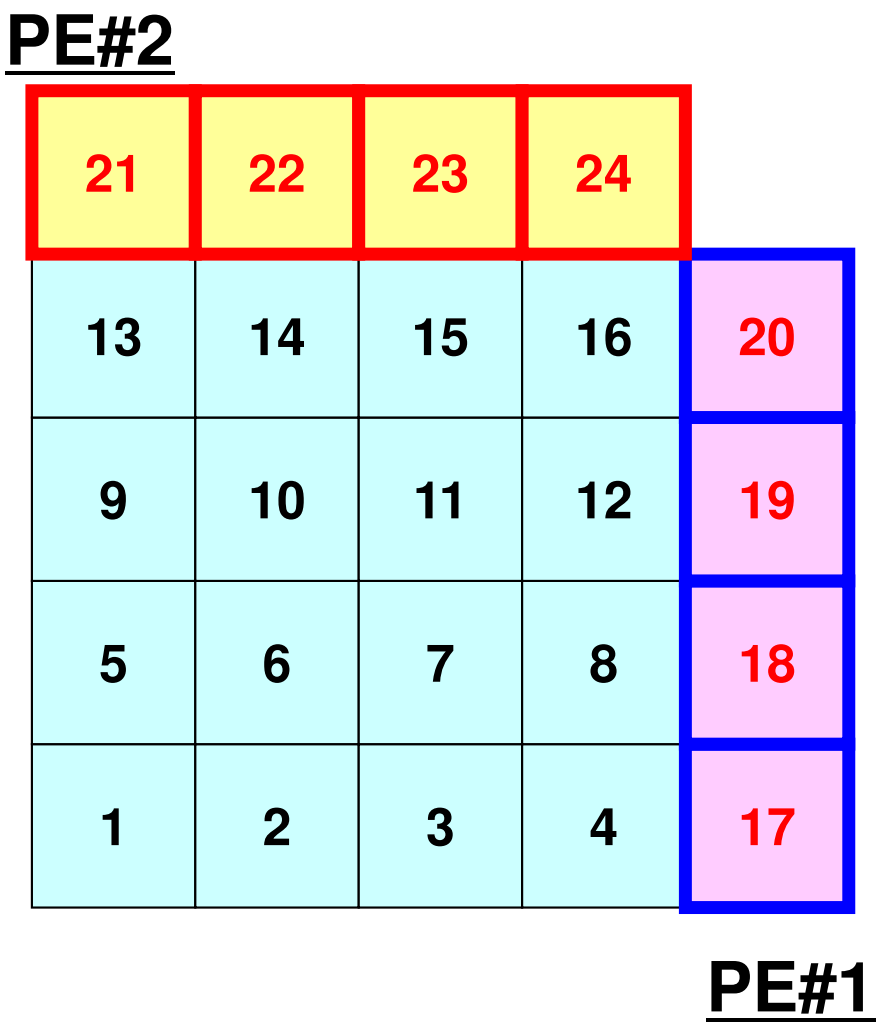
一般化された通信テーブル(2/6)

PE#2

PE#1

```
#NEIBPEtot   隣接領域数
2
#NEIBPE      隣接領域番号
1   2
#NODE
24  16       内点＋外点，内点数
#IMPORT_index
4   8
#IMPORT_items
17
18
19
20
21
22
23
24
#EXPORT_index
4   8
#EXPORT_items
4
8
12
16
13
14
15
16
```

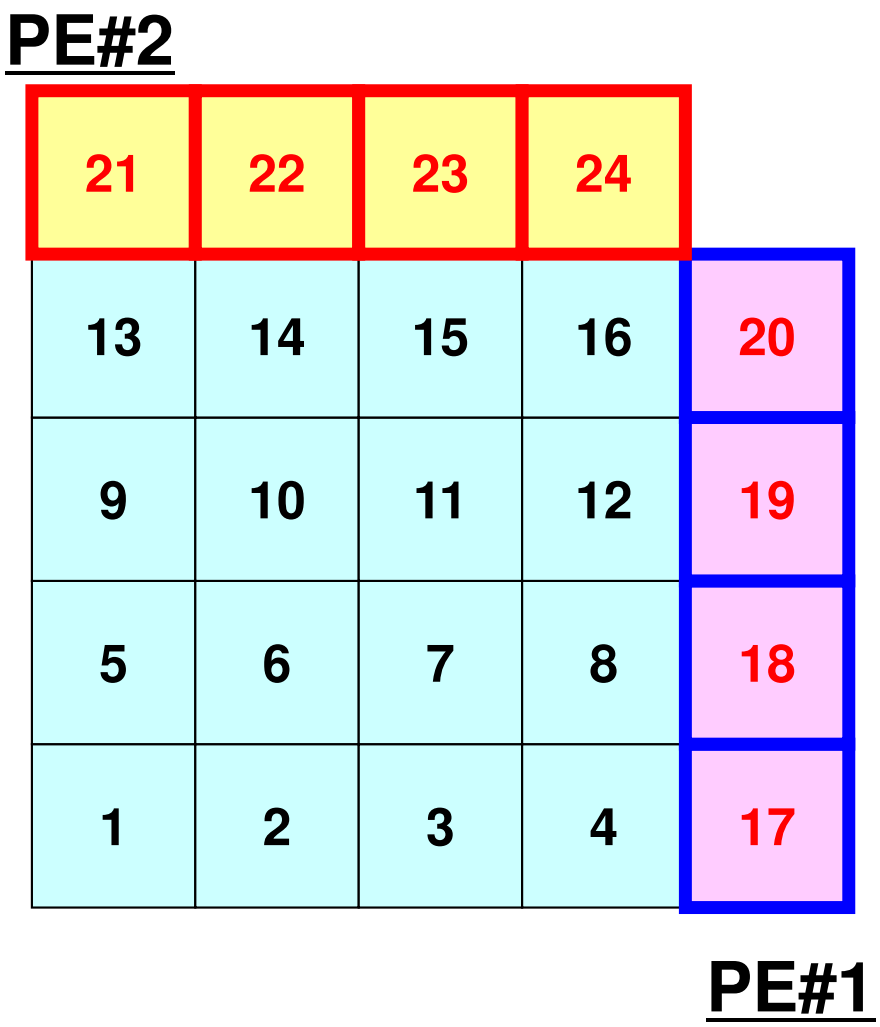
一般化された通信テーブル(3/6)




```
#NEIBPEtot
2
#NEIBPE
1  2
#NODE
24  16
#IMPORT_index
4   8
#IMPORT_items
17
18
19
20
21
22
23
24
#EXPORT_index
4   8
#EXPORT_items
4
8
12
16
13
14
15
16
```

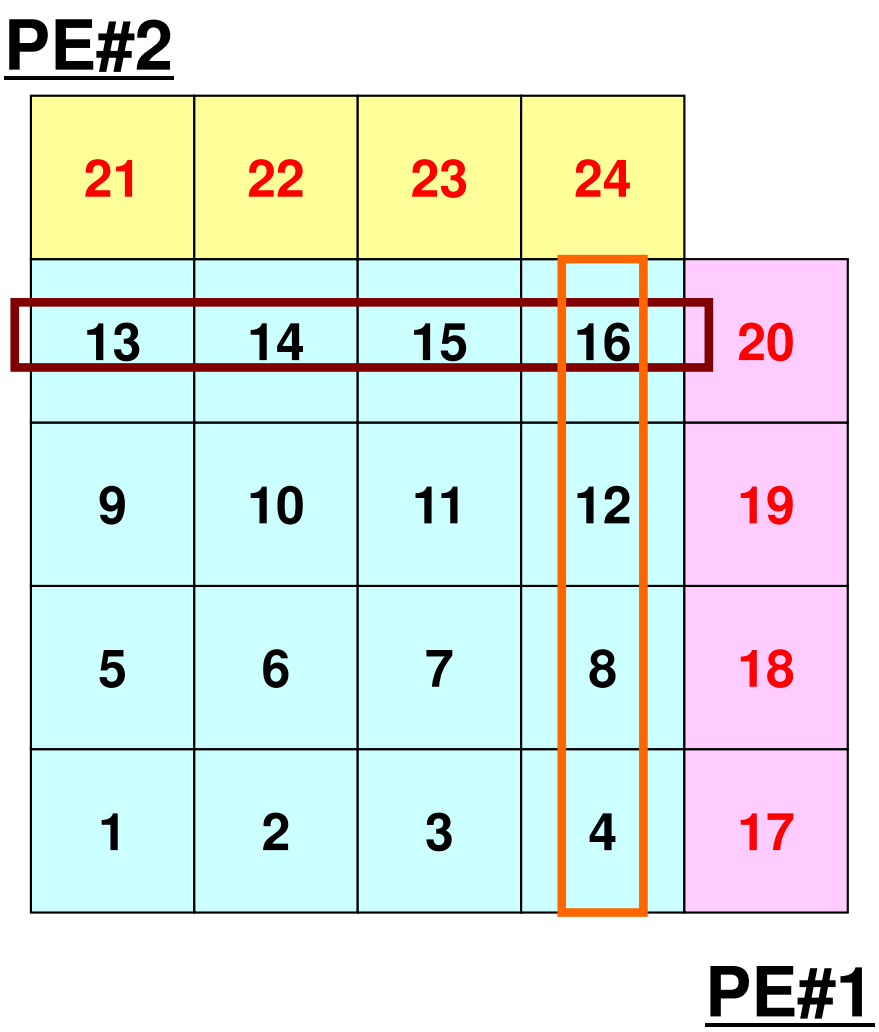
隣接領域1(#1)から4つ(1~4),
隣接領域2(#3)から4つ(5~8)が
「import(受信)」されることを示す。

一般化された通信テーブル(4/6)




```
#NEIBPEtot
2
#NEIBPE
1  2
#NODE
24  16
#IMPORT_index
4   8
#IMPORT_items
17
18                隣接領域1(#1)から
19                「import」する要素(1~4)
20
21
22                隣接領域2(#3)から
23                「import」する要素(5~8)
24
#EXPORT_index
4   8
#EXPORT_items
4
8
12
16
13
14
15
16
```

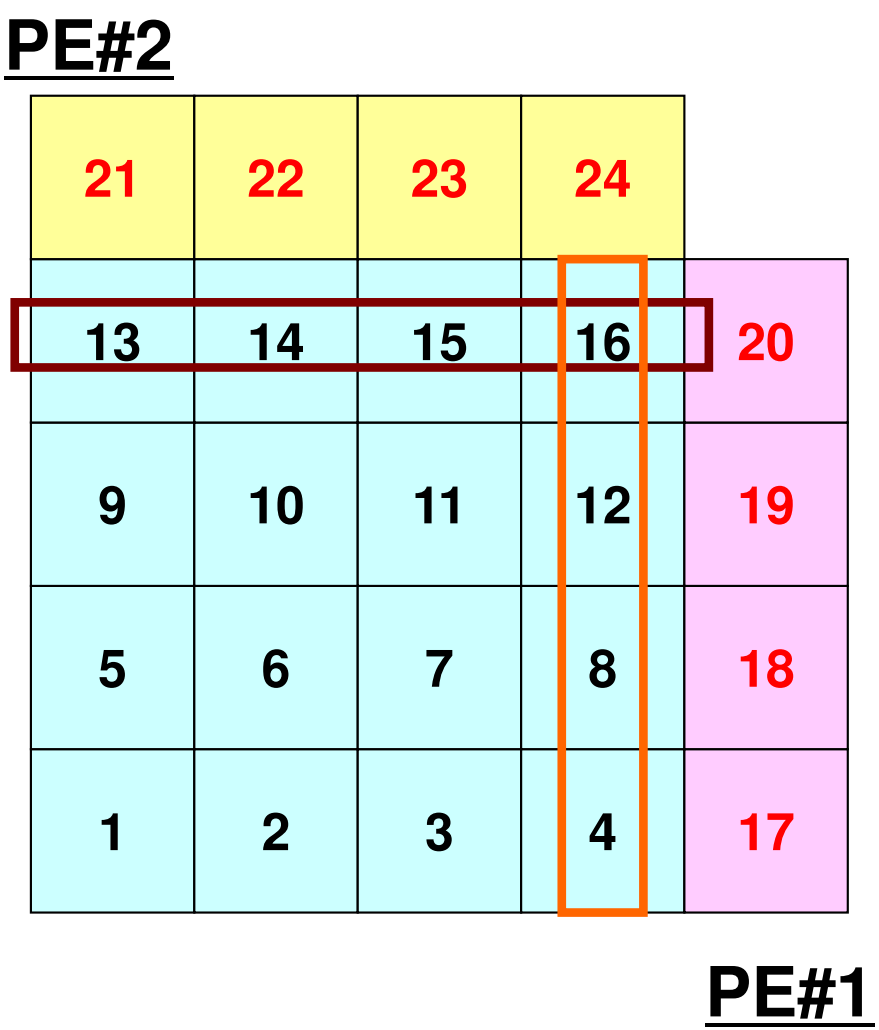
一般化された通信テーブル(5/6)




```
#NEIBPEtot
2
#NEIBPE
1   2
#NODE
24  16
#IMPORT_index
4   8
#IMPORT_items
17
18
19
20
21
22
23
24
#EXPORT_index
4   8
#EXPORT_items
4
8
12
16
13
14
15
16
```

隣接領域1(#1)へ4つ(1~4),
隣接領域2(#3)へ4つ(5~8)が
「export(送信)」されることを示す。

一般化された通信テーブル(6/6)




```
#NEIBPEtot
2
#NEIBPE
1  2
#NODE
24  16
#IMPORT_index
4   8
#IMPORT_items
17
18
19
20
21
22
23
24
#EXPORT_index
4   8
#EXPORT_items
4
8              隣接領域1(#1)へ
12             「export」する要素(1~4)
16
13
14             隣接領域2(#3)へ
15             「export」する要素(5~8)
16
```

一般化された通信テーブル(6/6)

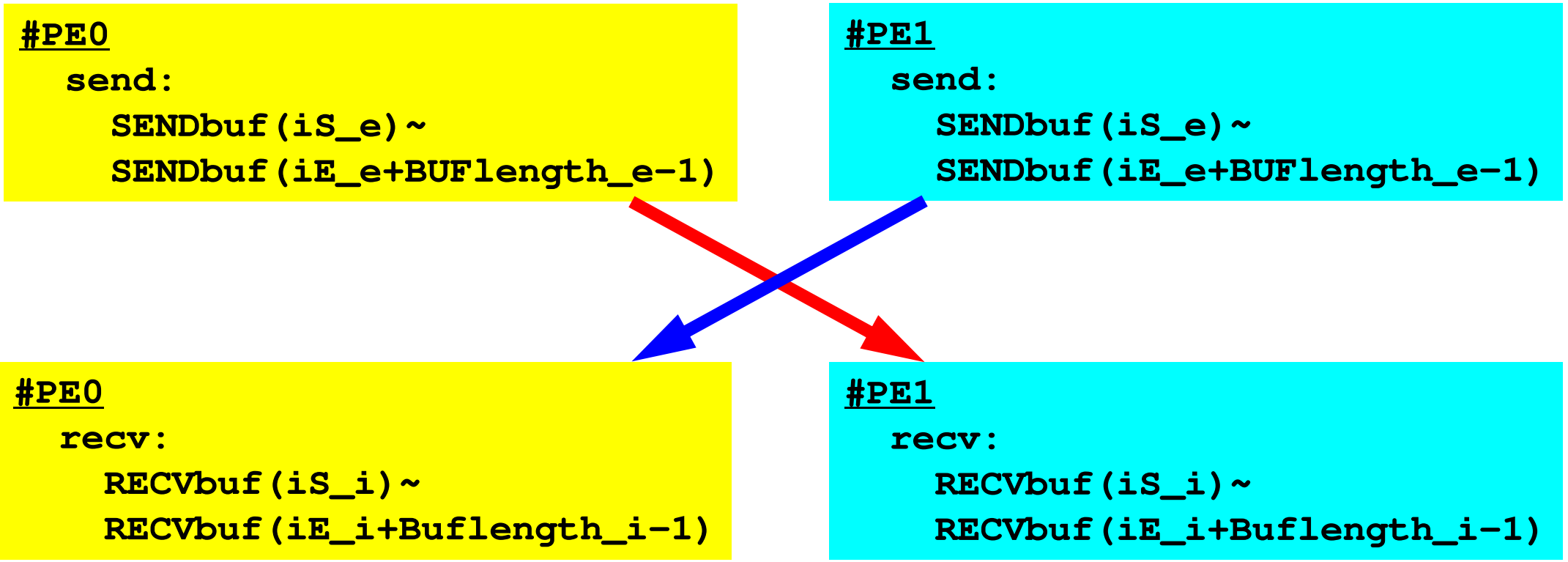
PE#2

PE#1

「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）。

配列の送受信:注意



- 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

1対1通信

- 1対1通信とは　？
- 二次元問題，一般化された通信テーブル
  - 二次元差分法
  - 問題設定
  - 局所データ構造と通信テーブル
  - 実装例
- 課題S2

サンプルプログラム：二次元データの例

```
$ cd /work/gt00/t00XXX/pFEM/mpi/S2
$ module load fj

$ mpifrtpx -Kfast sq-sr1.f
$ mpifccpx -Nclang -Kfast sq-sr1.c

$ 実行:4プロセス pjsub go4.sh
```

go4.sh

```
#!/bin/sh
#PJM -N "test"
#PJM -L rscgrp=tutorial-o
#PJM -L node=1
#PJM --mpi proc=4
#PJM -L elapse=00:15:00
#PJM -g gt00
#PJM -j
#PJM -e err
#PJM -o test.lst

module load fj
module load fjmpi

mpiexec ./a.out
```

プログラム例：sq-sr1.f（1/6）

Fortran

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, BUFlength

      integer(kind=4), dimension(:), allocatable :: VAL
      integer(kind=4), dimension(:), allocatable :: SENDbuf, RECVbuf
      integer(kind=4), dimension(:), allocatable :: 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_send, stat_recv
      integer(kind=4), dimension(:  ), allocatable :: request_send
      integer(kind=4), dimension(:  ), allocatable :: request_recv

      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 )
```

プログラム例：sq-sr1.f（2/6）

Fortran

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-sr1.f（2/6）

Fortran

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-sr1.c（2/6）

Fortran

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-sr1.c（2/6）

Fortran

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-sr1.f（2/6）

Fortran

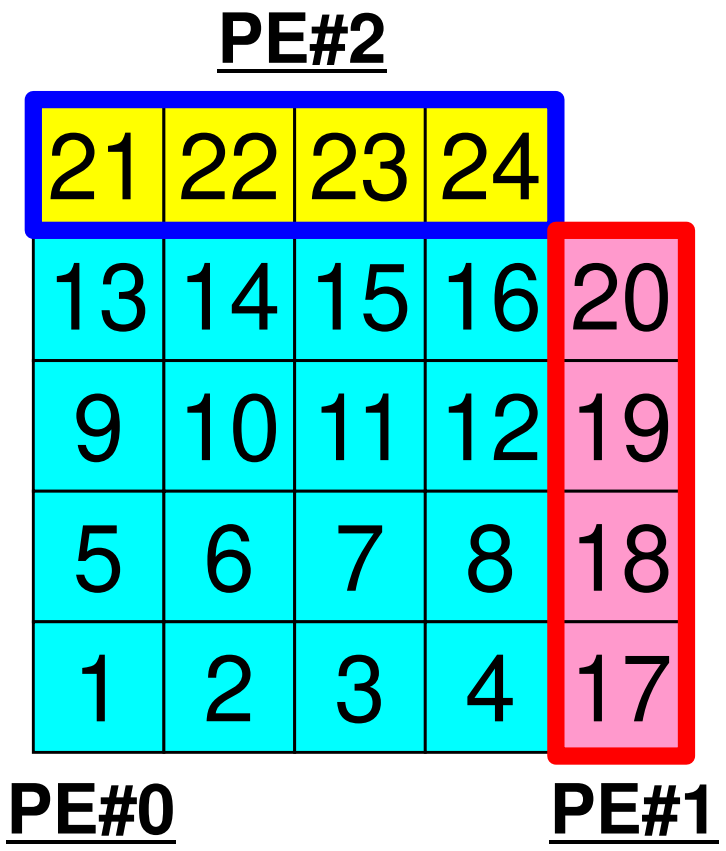
局所分散メッシュデータ（sqm.*）読み込み

```
!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
```

PE#0 受信

```
#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-sr1.f（2/6）

Fortran

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-sr1.f（2/6）

Fortran

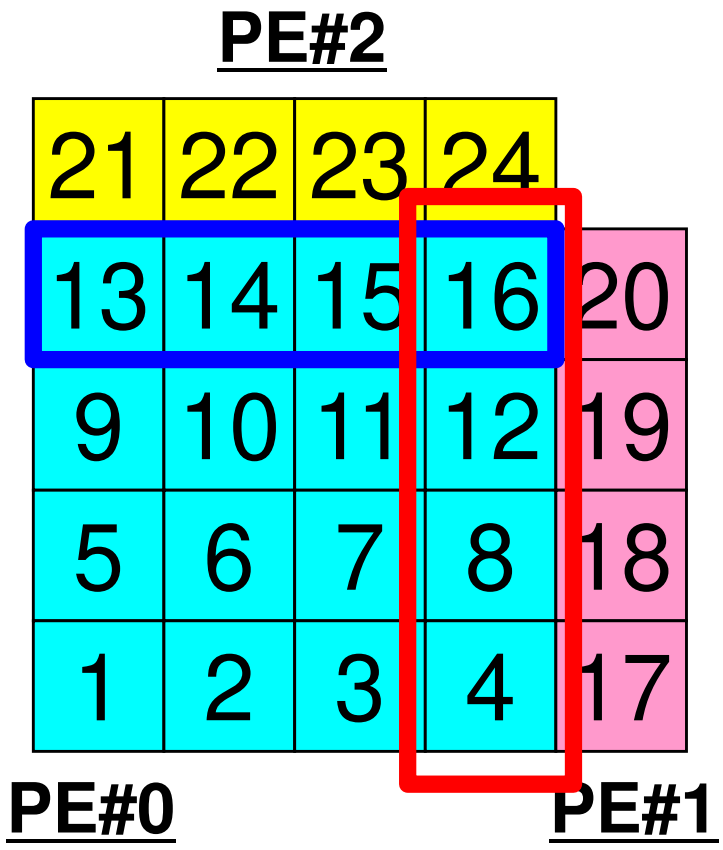
局所分散メッシュデータ（sqm.*）読み込み

```
!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
```

PE#0 送信

```
#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
```



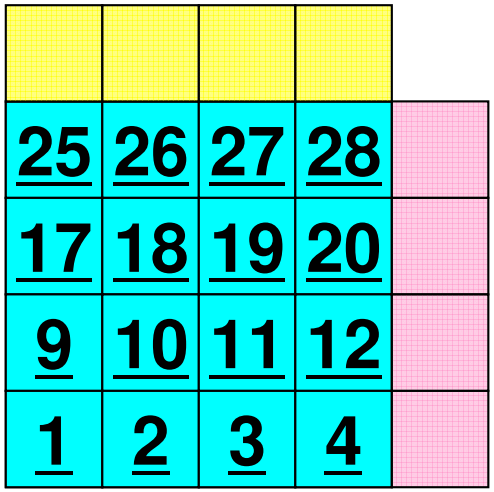
プログラム例：sq-sr1.f（3/6）

Fortran

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




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

プログラム例：sq-sr1.f（4/6）

Fortran

送・受信バッファ準備

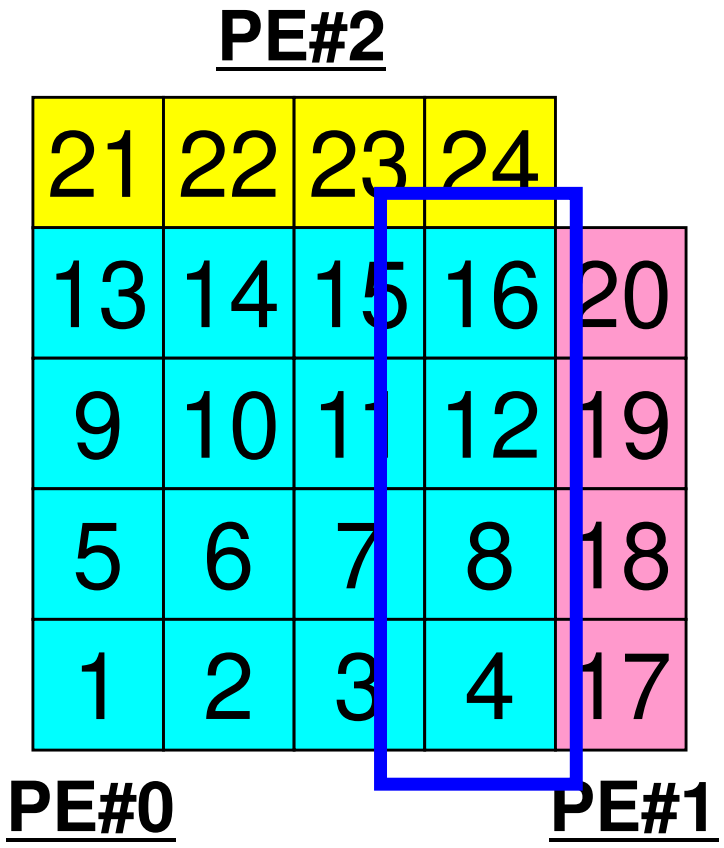
```
!C
!C +--------+
!C | BUFFER |
!C +--------+
!C===
      allocate (SENDbuf(export_index(NEIBPETOT)))
      allocate (RECVbuf(import_index(NEIBPETOT)))

      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。

送信バッファの効能

Fortran

```
 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
```

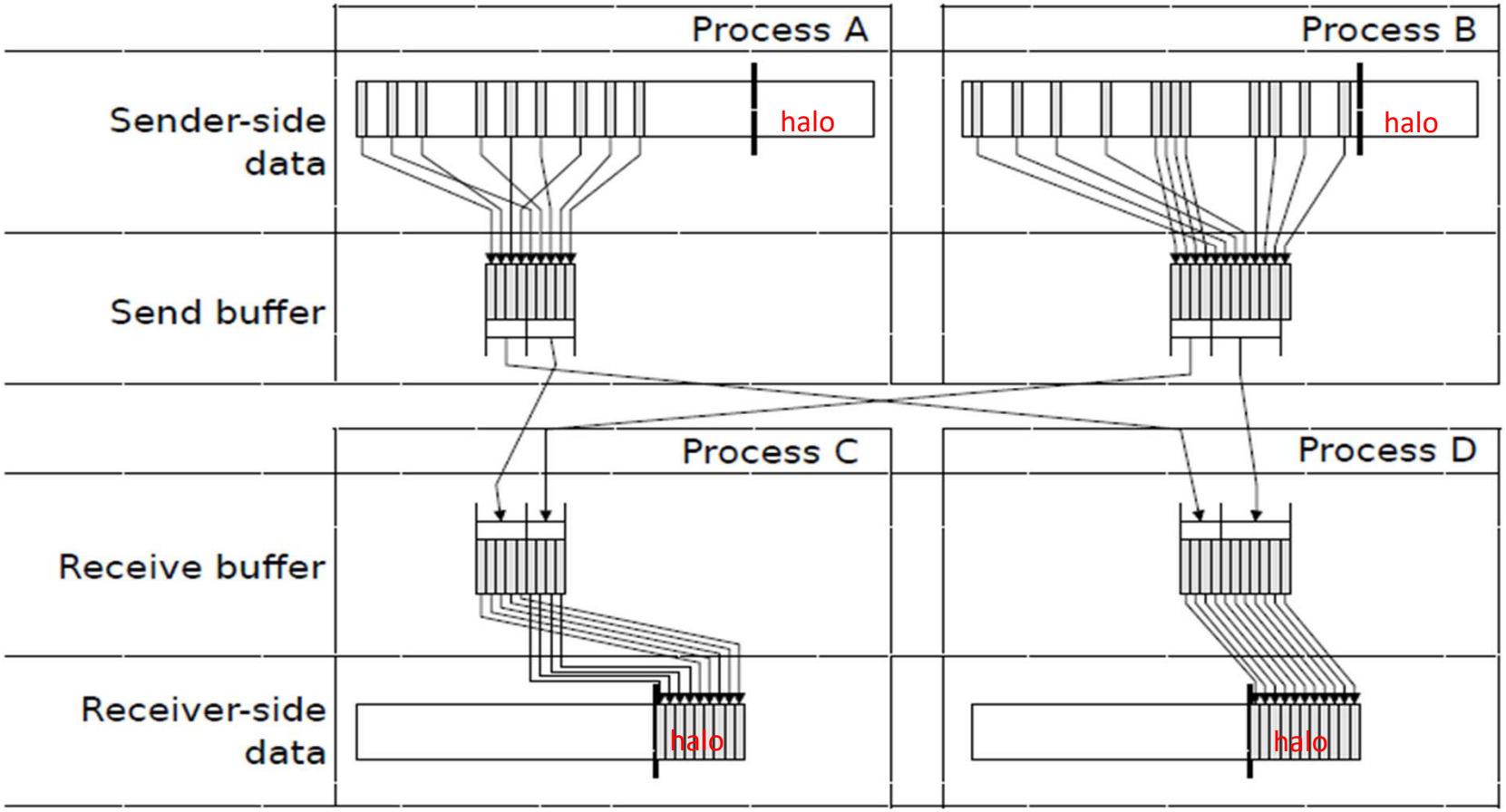


たとえば，この境界点は連続していないので，

- 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

Communication Pattern using 1D Structure



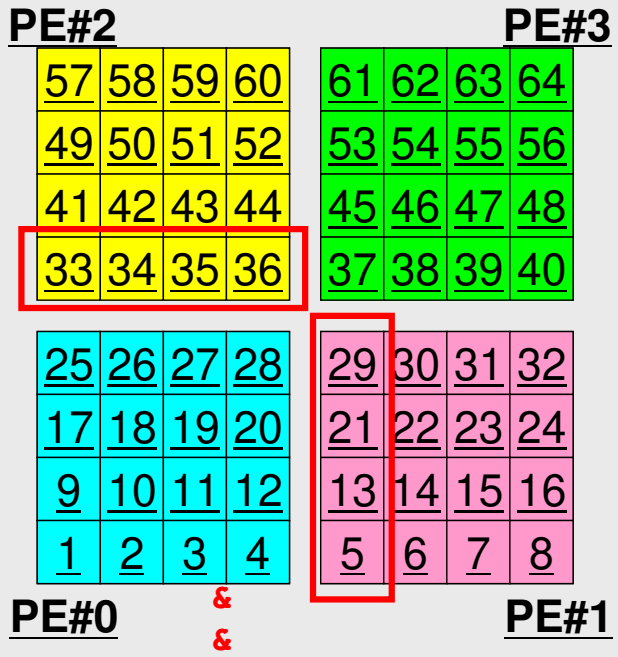
Dr. Osni Marques
（Lawrence Berkeley National Laboratory）より借用

プログラム例：sq-sr1.f（5/6）

Fortran

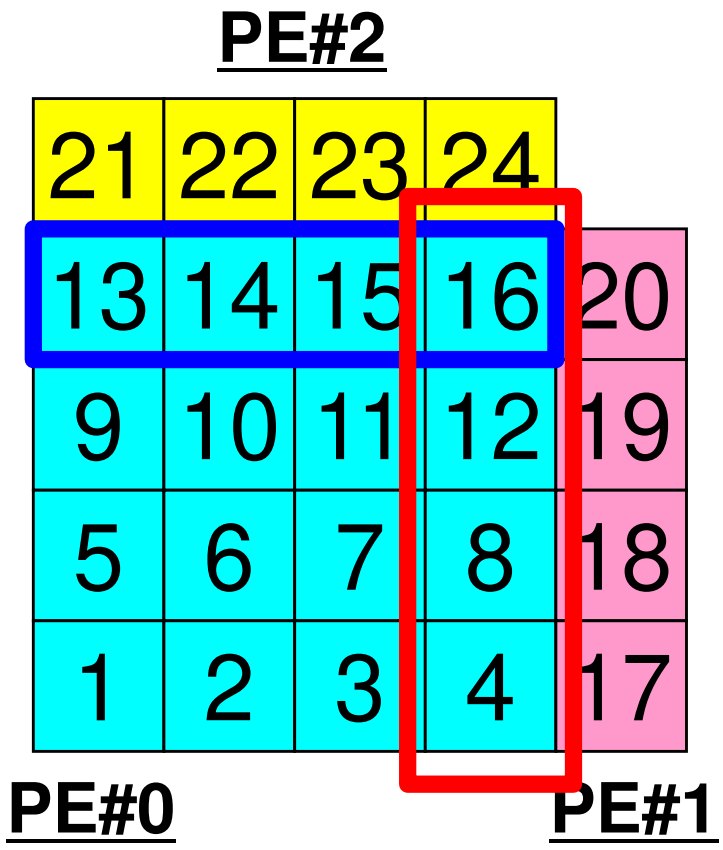
送信（MPI_Isend）

```
!C
!C +-----------+
!C | SEND-RECV |
!C +-----------+
!C===
      allocate (stat_send(MPI_STATUS_SIZE,NEIBPETOT))
      allocate (stat_recv(MPI_STATUS_SIZE,NEIBPETOT))
      allocate (request_send(NEIBPETOT))
      allocate (request_recv(NEIBPETOT))

      do neib= 1, NEIBPETOT
        iS= export_index(neib-1) + 1
        iE= export_index(neib  )
        BUFlength= iE + 1 - iS
        call MPI_ISEND (SENDbuf(iS), BUFlength, MPI_INTEGER,
     &                  NEIBPE(neib), 0, MPI_COMM_WORLD,
     &                  request_send(neib), ierr)
      enddo

      do neib= 1, NEIBPETOT
        iS= import_index(neib-1) + 1
        iE= import_index(neib  )
        BUFlength= iE + 1 - iS
        call MPI_IRECV (RECVbuf(iS), BUFlength, MPI_INTEGER,
     &                  NEIBPE(neib), 0, MPI_COMM_WORLD,
     &                  request_recv(neib), ierr)
      enddo
```

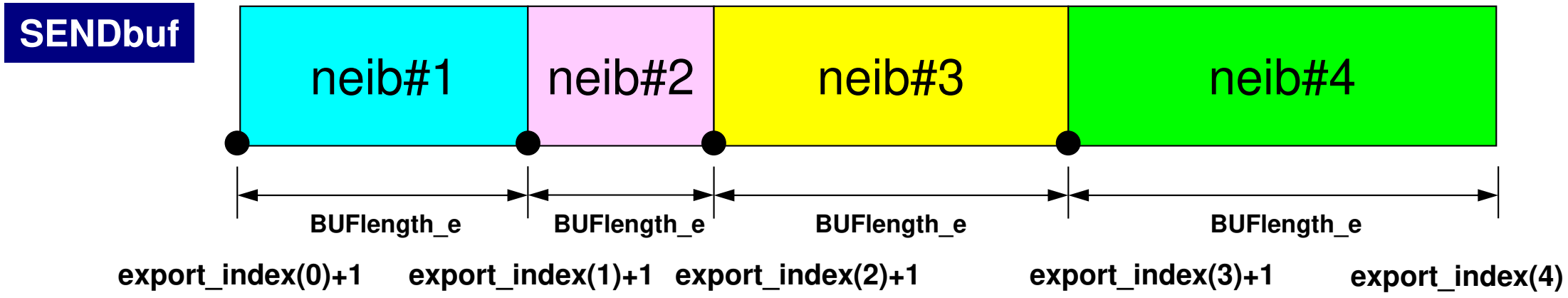


PE#0 送信

```
#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
```



送信（MPI_Isend/Irecv/Waitall） Fortran




```
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)
```

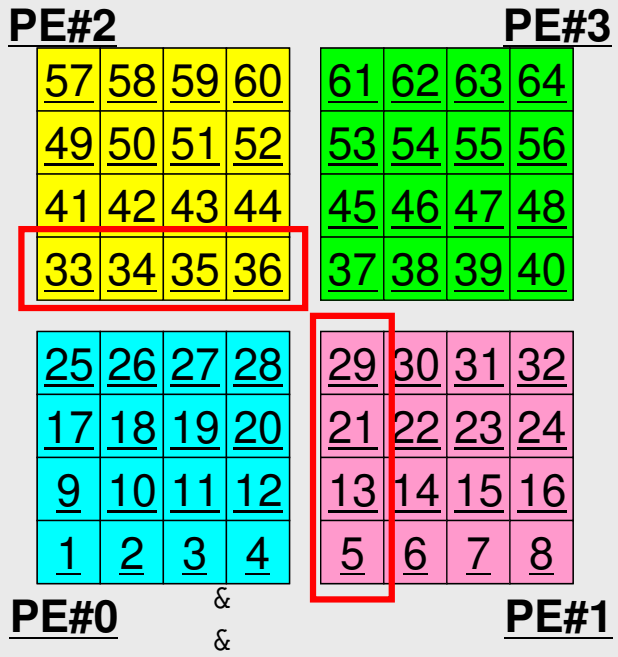
送信バッファへの代入
温度などの変数を直接送信，受信に使うのではなく，このようなバッファへ一回代入して計算することを勧める。

プログラム例：sq-sr1.f（5/6）

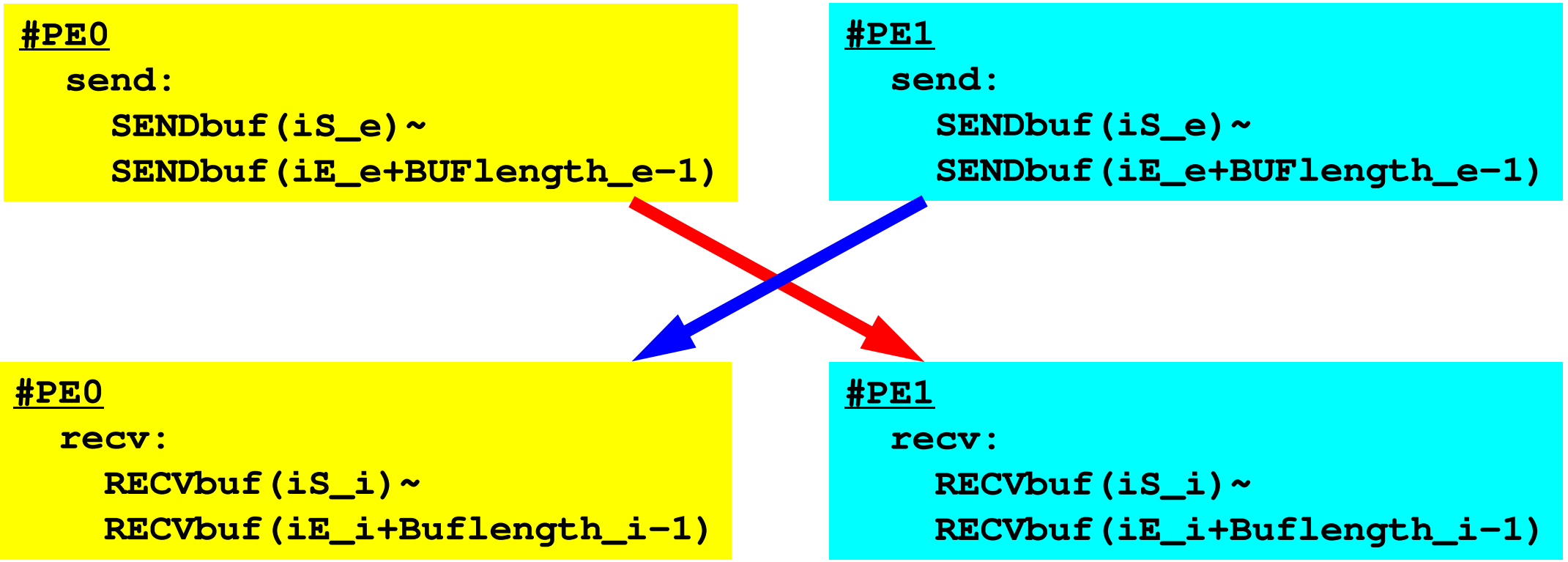
Fortran

受信（MPI_Irecv）

```
!C
!C +-----------+
!C | SEND-RECV |
!C +-----------+
!C===
      allocate (stat_send(MPI_STATUS_SIZE,NEIBPETOT))
      allocate (stat_recv(MPI_STATUS_SIZE,NEIBPETOT))
      allocate (request_send(NEIBPETOT))
      allocate (request_recv(NEIBPETOT))

      do neib= 1, NEIBPETOT
        iS= export_index(neib-1) + 1
        iE= export_index(neib  )
        BUFlength= iE + 1 - iS
        call MPI_ISEND (SENDbuf(iS), BUFlength, MPI_INTEGER,
     &                  NEIBPE(neib), 0, MPI_COMM_WORLD,
     &                  request_send(neib), ierr)
      enddo

      do neib= 1, NEIBPETOT
        iS= import_index(neib-1) + 1
        iE= import_index(neib  )
        BUFlength= iE + 1 - iS
        call MPI_IRECV (RECVbuf(iS), BUFlength, MPI_INTEGER,
     &                  NEIBPE(neib), 0, MPI_COMM_WORLD,
     &                  request_recv(neib), ierr)
      enddo
```



配列の送受信:注意



- 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

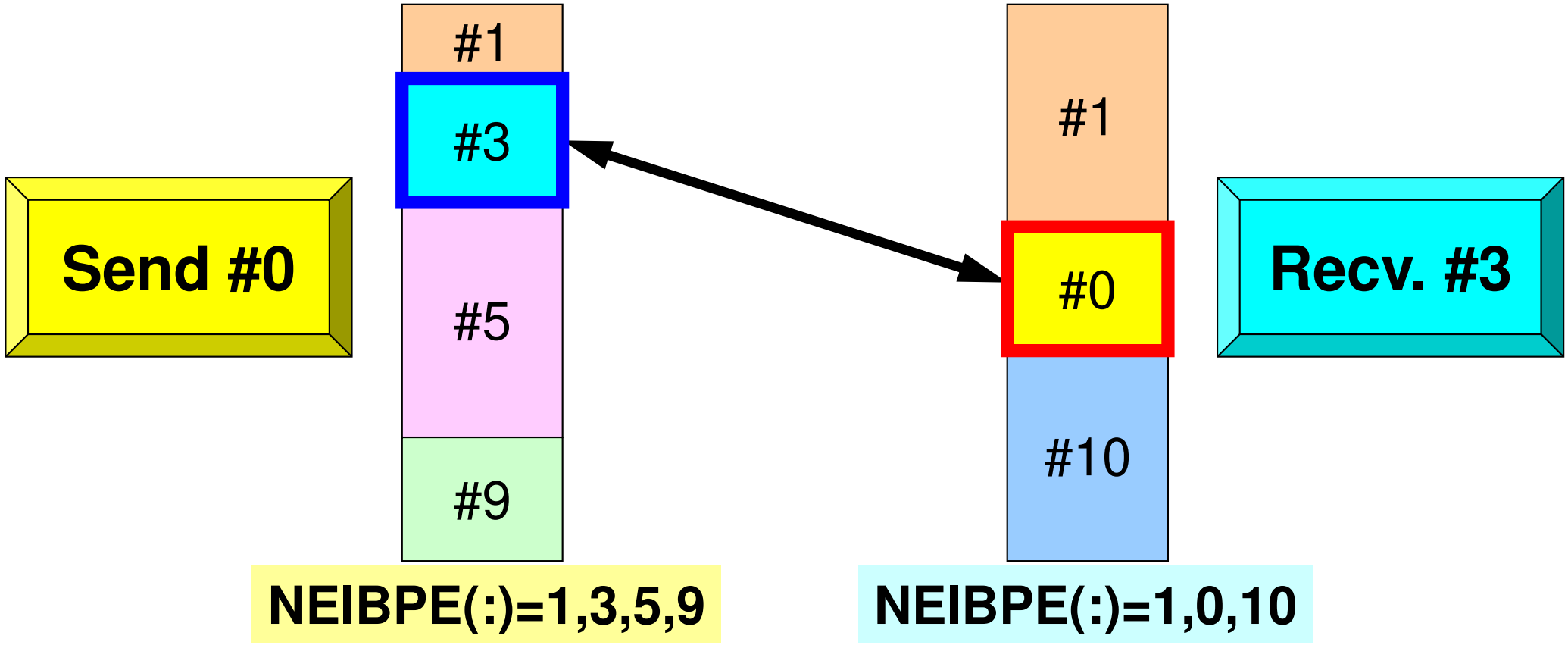
SEND/RECVの関係

```
 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
```

```
 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
```

- 送信元（sources）/受信先（destinations），メッセージサイズ，メッセージの中身の整合性（consistency）
- 隣接プロセスID（NEIBPE（neib））とtag（=0）が整合すると通信発生

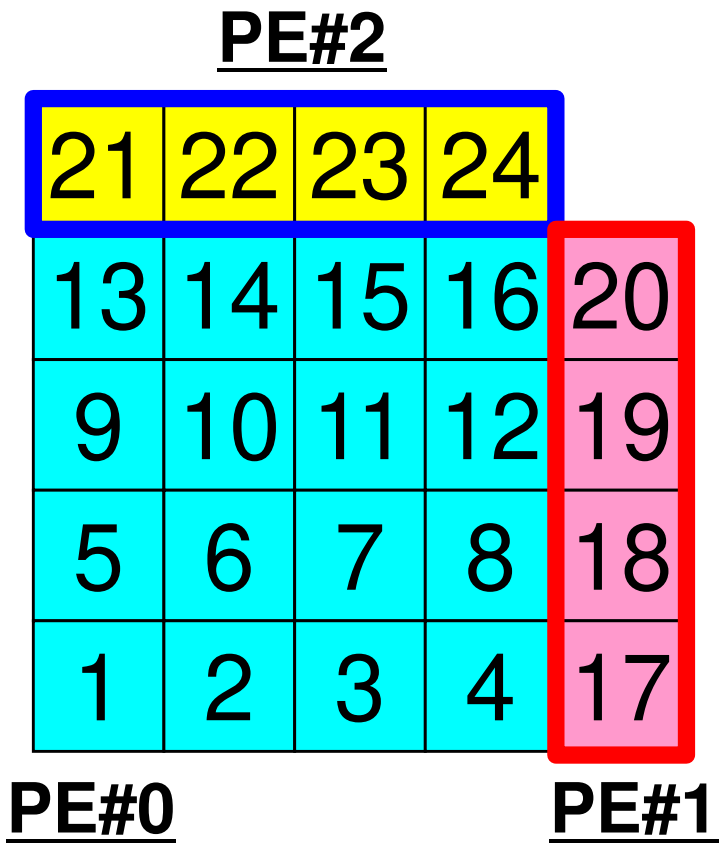
SEND/RECVの関係 (#0⇒#3)



- 送信元（sources）/受信先（destinations），メッセージサイズ，メッセージの中身の整合性（consistency）
- 隣接プロセスID（NEIBPE（neib））とtag（=0）が整合すると通信発生

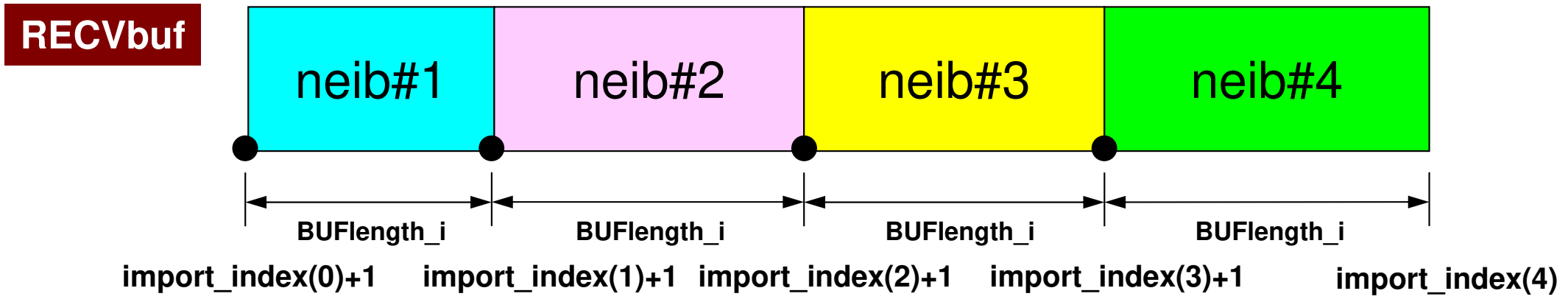
PE#0 受信

```
#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
```



受信（MPI_Isend/Irecv/Waitall） Fortran

```
 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)                                      受信バッファから代入
     enddo
  enddo
```



プログラム例：sq-sr1.f（6/6）

Fortran

受信バッファの中身の代入

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

      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

      call MPI_WAITALL (NEIBPETOT, request_send, stat_send, ierr)
!C===

!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
```

受信バッファの中身を「外点」の値として代入する。

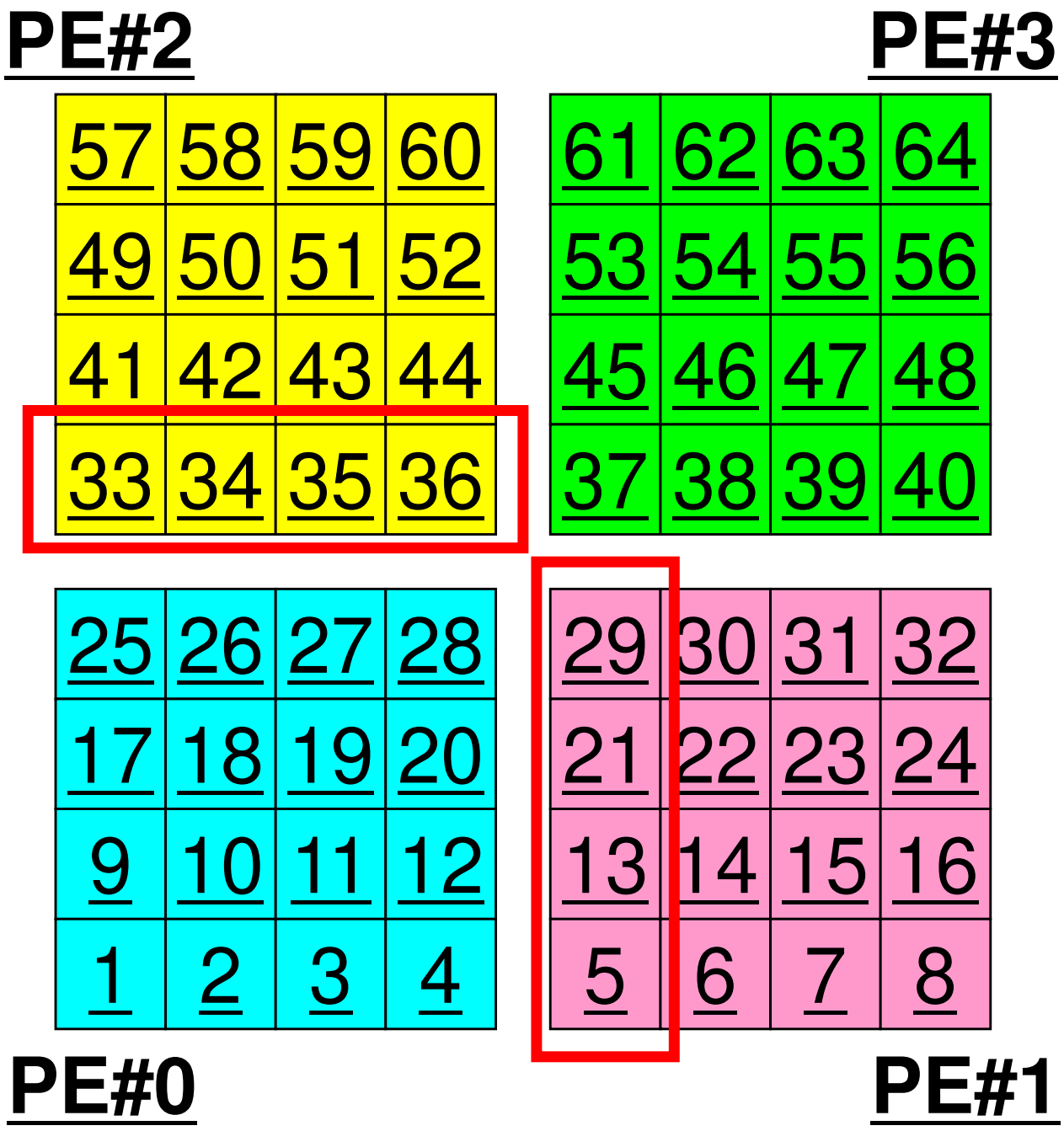
プログラム例：sq-sr1.f（6/6）

Fortran

外点の値の書き出し

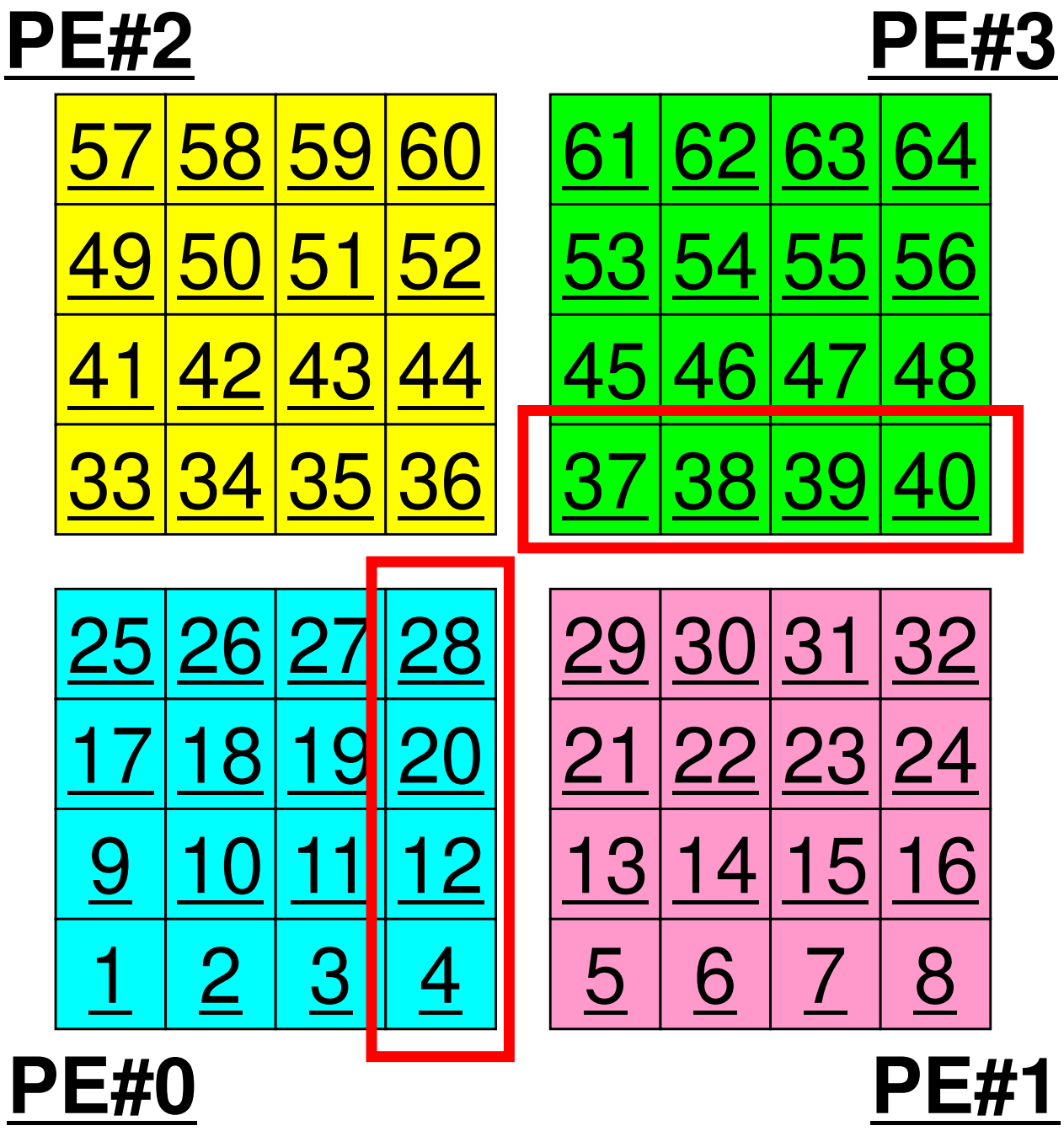
```
      call MPI_WAITALL (NEIBPETOT, request_recv, stat_recv, ierr)

      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

      call MPI_WAITALL (NEIBPETOT, request_send, stat_send, ierr)
!C===

!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）



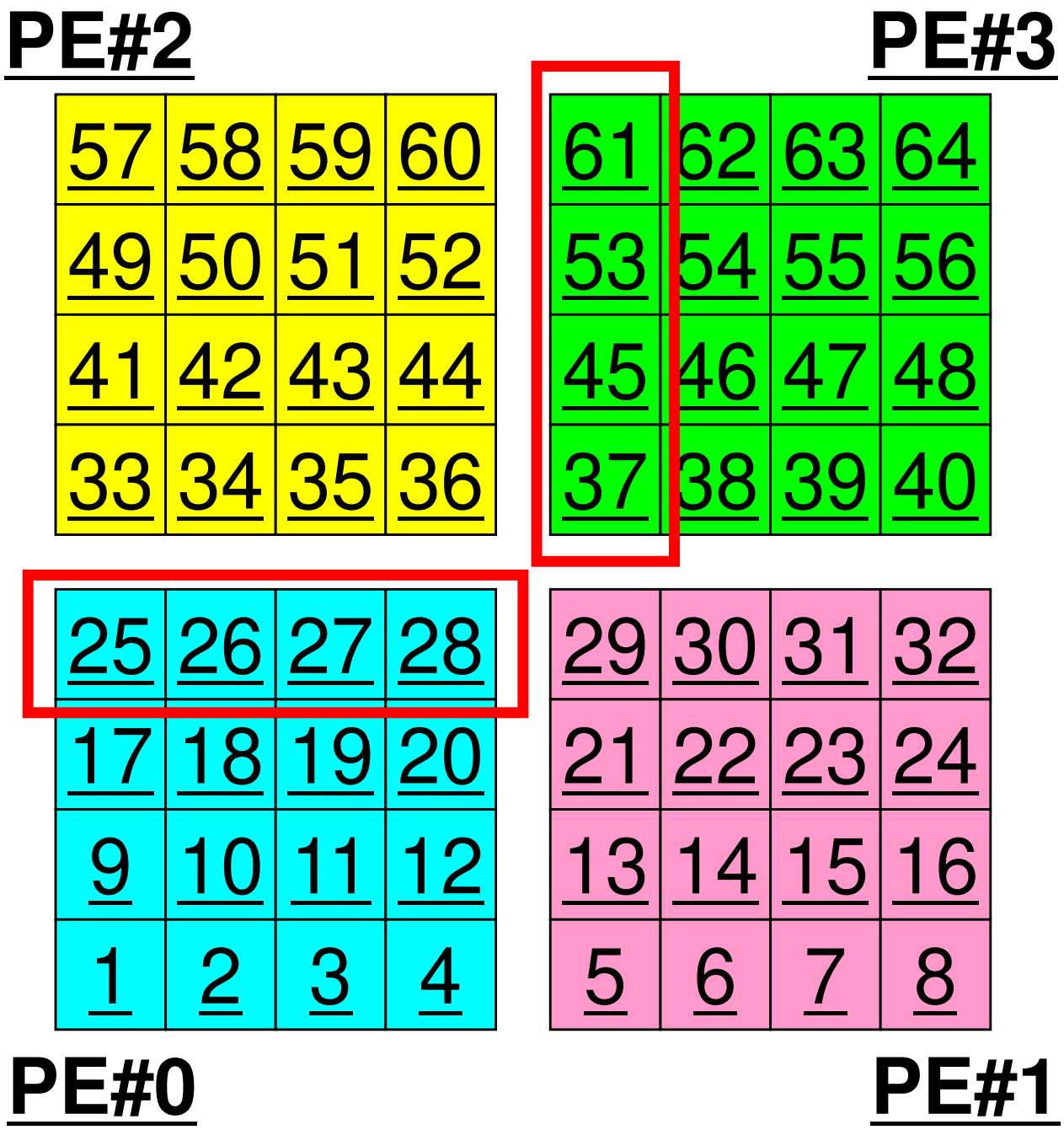

```
RECVbuf      0      1      5
RECVbuf      0      1     13
RECVbuf      0      1     21
RECVbuf      0      1     29
RECVbuf      0      2     33
RECVbuf      0      2     34
RECVbuf      0      2     35
RECVbuf      0      2     36

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）



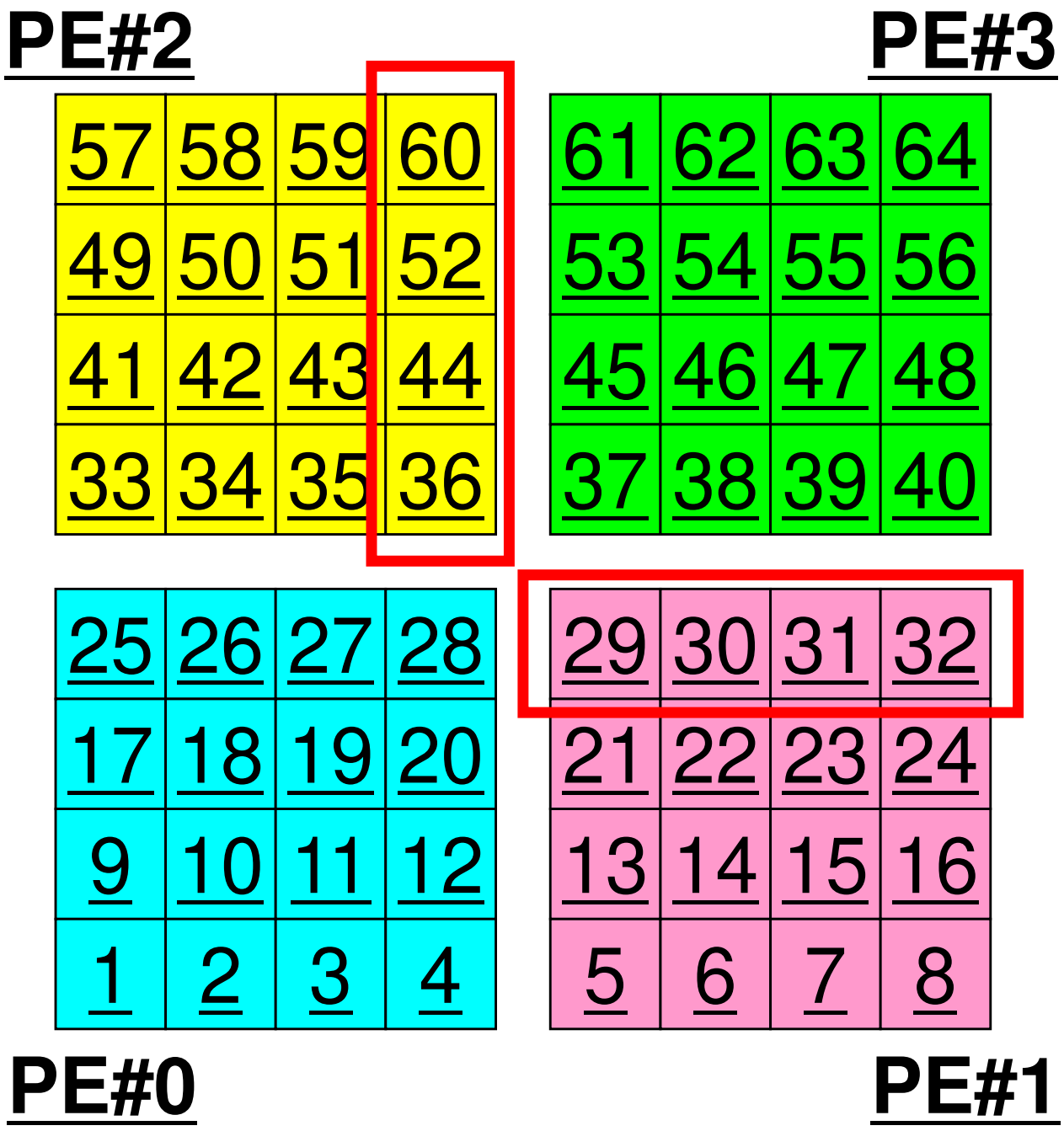

```
RECVbuf         0         1          5
RECVbuf         0         1         13
RECVbuf         0         1         21
RECVbuf         0         1         29
RECVbuf         0         2         33
RECVbuf         0         2         34
RECVbuf         0         2         35
RECVbuf         0      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

並列計算向け局所(分散)データ構造

- 差分法，有限要素法，有限体積法等係数が疎行列のアプリケーション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

- 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

初期全体メッシュ

演習t2

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t2

#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t2

#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） 演習t2
○の部分をうめよ!

#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
    13     8 （内点＋外点，内点）
#IMPORTindex
     ○     ○
#IMPORTitems
     ○…
#EXPORTindex
     ○     ○
#EXPORTitems
     ○…
```

PE#1：局所分散データ（sqm.1） 演習t2
○の部分をうめよ!

#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
    14     8 （内点，内点＋外点）
#IMPORTindex
     ○     ○
#IMPORTitems
     ○…
#EXPORTindex
     ○     ○
#EXPORTitems
     ○…
```

PE#2：局所分散データ（sqm.2） 演習t2
○の部分をうめよ!

#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
    15     9 （内点，内点＋外点）
#IMPORTindex
     ○    ○
#IMPORTitems
     ○…
#EXPORTindex
     ○    ○
#EXPORTitems
     ○…
```

演習t2

#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

手　順

演習t2

- 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

- <$O-S2>/exに「sq.*」がある
- 自分で「sqm.*」を作成する
- <$O-S2>から「sq-sr1.f/c」をコンパイルした実行形式をコピー
- pjsub go3.sh

課題S2

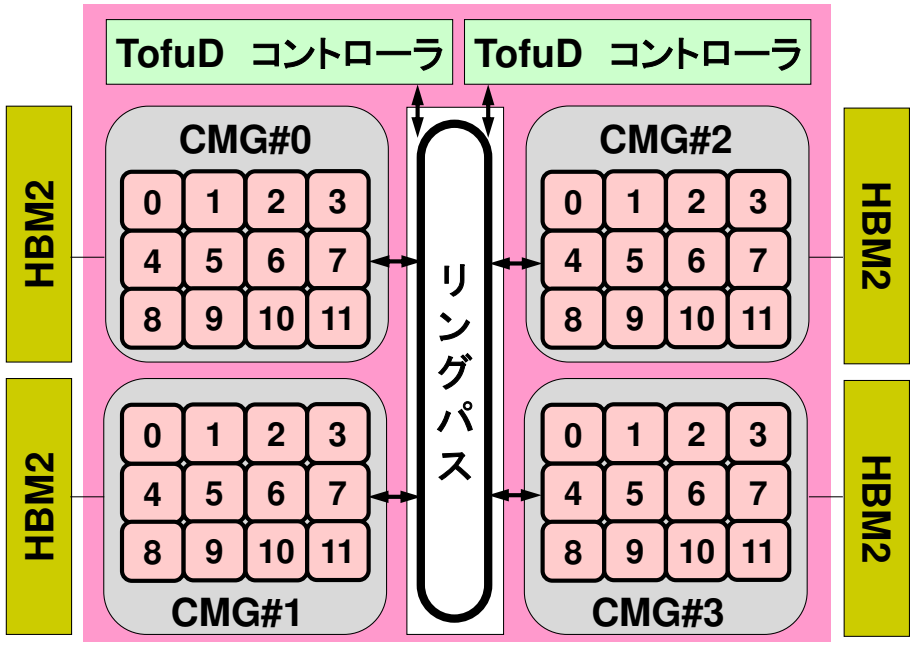
- 一次元熱伝導解析コード「1d.f，1d.c」をMPIによって並列化せよ
- 全要素数を読み込んで，プログラム内で領域分割すること
- 並列性能はあまり出ないが測定して見よ

課題S2

- 内容
  - 1d.f/1d.cを「一般化された通信テーブル」を使って並列化せよ
  - 全要素数を読み込んで，プログラム内で領域分割すること
  - 並列性能について考察すること
    - 要素数はかなり多くしないと多分性能が出ない
    - 計算が終わらないようであれば反復回数を少なくして比較
- 提出物（レポート）：最高級仕様
  - 表紙：氏名，学籍番号，課題番号を明記
  - 以下についてA4　8枚以内（図表含む）でまとめること
    - 基本方針（フロー図），プログラム構造・説明，考察・課題
  - プログラムリスト
  - 結果出力リスト（最小限にとどめること）

プロセス数

```
#PJM -L node=1;#PJM --mpi proc= 1    1-node, 1-proc, 1-proc/n
#PJM -L node=1;#PJM --mpi proc= 4    1-node, 4-proc, 4-proc/n
#PJM -L node=1;#PJM --mpi proc=12   1-node,12-proc,12-proc/n
#PJM -L node=1;#PJM --mpi proc=24   1-node,24-proc,24-proc/n
#PJM -L node=1;#PJM --mpi proc=48   1-node,48-proc,48-proc/n

#PJM -L node= 4;#PJM --mpi proc=192   4-node,192-proc,48-proc/n
#PJM -L node= 8;#PJM --mpi proc=384   8-node,384-proc,48-proc/n
#PJM -L node=12;#PJM --mpi proc=576  12-node,576-proc,48-proc/n
```



a012.sh

```
#!/bin/sh
#PJM -N "test"
#PJM -L rscgrp=tutorial-o
#PJM -L node=1
#PJM --mpi proc=12
#PJM -L elapse=00:15:00
#PJM -g gt00
#PJM -j
#PJM -e err
#PJM -o test.lst

module load fj
module load fjmpi
mpiexec ./a.out
mpiexec numactl -l ./a.out
```

a048.sh

```
#!/bin/sh
#PJM -N "test"
#PJM -L rscgrp=tutorial-o
#PJM -L node=1
#PJM --mpi proc=48
#PJM -L elapse=00:15:00
#PJM -g gt00
#PJM -j
#PJM -e err
#PJM -o test.lst

module load fj
module load fjmpi
mpiexec ./a.out
mpiexec numactl -l ./a.out
```

a384.sh

```
#!/bin/sh
#PJM -N "test"
#PJM -L rscgrp=tutorial-o
#PJM -L node=8
#PJM --mpi proc=384
#PJM -L elapse=00:15:00
#PJM -g gt00
#PJM -j
#PJM -e err
#PJM -o test.lst

module load fj
module load fjmpi
mpiexec ./a.out
mpiexec numactl -l ./a.out
```

a576.sh

```
#!/bin/sh
#PJM -N "test"
#PJM -L rscgrp=tutorial-o
#PJM -L node=12
#PJM --mpi proc=576
#PJM -L elapse=00:15:00
#PJM -g gt00
#PJM -j
#PJM -e err
#PJM -o test.lst

module load fj
module load fjmpi
mpiexec ./a.out
mpiexec numactl -l ./a.out
```

numactl -l/--localalloc ローカルなメモリを使用（効果はほとんど無い）