MPIによるプログラミング概要 その3
課題S2出題

2007年5月9日

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）

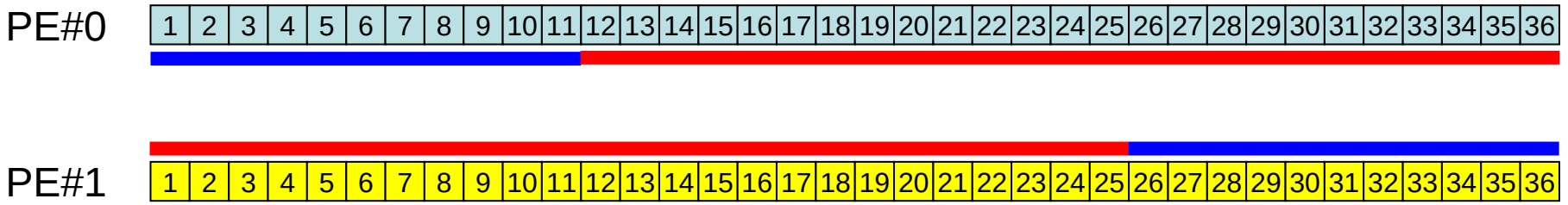
授業・課題の予定

- MPIサブルーチン機能
  - 環境管理
  - グループ通信
  - 1対1通信

- 2007年4月25日，5月2日，5月9日（＋5月16日）
  - 環境管理，グループ通信（Collective Communication）
    - 課題S1
  - 1対1通信（Point-to-Point Communication）
    - 前回「Peer-to-Peer」と言ってしまったが間違い。申し訳ない。
      - もとの資料も「point-to-point」になっていたのだが・・・
    - 課題S2：一次元熱伝導解析コードの「並列化」
  - ここまでできればあとはある程度自分で解決できます

利用例(2):配列の送受信(1/4)

- 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

- VEC(:)の初期状態を以下のようにする:
  - PE#0　VEC(1-36)= 100
  - PE#1　VEC(1-36)= 200

- 次ページのような結果になることを確認せよ

- 以下のそれぞれを使用したプログラムを作成せよ
  - MPI_Isend/Irecv/Waitall
  - MPI_Sendrecv

予測される結果

```
0 #BEFORE#  1      100.
0 #BEFORE#  2      100.
0 #BEFORE#  3      100.
0 #BEFORE#  4      100.
0 #BEFORE#  5      100.
0 #BEFORE#  6      100.
0 #BEFORE#  7      100.
0 #BEFORE#  8      100.
0 #BEFORE#  9      100.
0 #BEFORE# 10      100.
0 #BEFORE# 11      100.
0 #BEFORE# 12      100.
0 #BEFORE# 13      100.
0 #BEFORE# 14      100.
0 #BEFORE# 15      100.
0 #BEFORE# 16      100.
0 #BEFORE# 17      100.
0 #BEFORE# 18      100.
0 #BEFORE# 19      100.
0 #BEFORE# 20      100.
0 #BEFORE# 21      100.
0 #BEFORE# 22      100.
0 #BEFORE# 23      100.
0 #BEFORE# 24      100.
0 #BEFORE# 25      100.
0 #BEFORE# 26      100.
0 #BEFORE# 27      100.
0 #BEFORE# 28      100.
0 #BEFORE# 29      100.
0 #BEFORE# 30      100.
0 #BEFORE# 31      100.
0 #BEFORE# 32      100.
0 #BEFORE# 33      100.
0 #BEFORE# 34      100.
0 #BEFORE# 35      100.
0 #BEFORE# 36      100.
```

```
0 #AFTER #  1      100.
0 #AFTER #  2      100.
0 #AFTER #  3      100.
0 #AFTER #  4      100.
0 #AFTER #  5      100.
0 #AFTER #  6      100.
0 #AFTER #  7      100.
0 #AFTER #  8      100.
0 #AFTER #  9      100.
0 #AFTER # 10      100.
0 #AFTER # 11      100.
0 #AFTER # 12      200.
0 #AFTER # 13      200.
0 #AFTER # 14      200.
0 #AFTER # 15      200.
0 #AFTER # 16      200.
0 #AFTER # 17      200.
0 #AFTER # 18      200.
0 #AFTER # 19      200.
0 #AFTER # 20      200.
0 #AFTER # 21      200.
0 #AFTER # 22      200.
0 #AFTER # 23      200.
0 #AFTER # 24      200.
0 #AFTER # 25      200.
0 #AFTER # 26      200.
0 #AFTER # 27      200.
0 #AFTER # 28      200.
0 #AFTER # 29      200.
0 #AFTER # 30      200.
0 #AFTER # 31      200.
0 #AFTER # 32      200.
0 #AFTER # 33      200.
0 #AFTER # 34      200.
0 #AFTER # 35      200.
0 #AFTER # 36      200.
```

```
1 #BEFORE#  1      200.
1 #BEFORE#  2      200.
1 #BEFORE#  3      200.
1 #BEFORE#  4      200.
1 #BEFORE#  5      200.
1 #BEFORE#  6      200.
1 #BEFORE#  7      200.
1 #BEFORE#  8      200.
1 #BEFORE#  9      200.
1 #BEFORE# 10      200.
1 #BEFORE# 11      200.
1 #BEFORE# 12      200.
1 #BEFORE# 13      200.
1 #BEFORE# 14      200.
1 #BEFORE# 15      200.
1 #BEFORE# 16      200.
1 #BEFORE# 17      200.
1 #BEFORE# 18      200.
1 #BEFORE# 19      200.
1 #BEFORE# 20      200.
1 #BEFORE# 21      200.
1 #BEFORE# 22      200.
1 #BEFORE# 23      200.
1 #BEFORE# 24      200.
1 #BEFORE# 25      200.
1 #BEFORE# 26      200.
1 #BEFORE# 27      200.
1 #BEFORE# 28      200.
1 #BEFORE# 29      200.
1 #BEFORE# 30      200.
1 #BEFORE# 31      200.
1 #BEFORE# 32      200.
1 #BEFORE# 33      200.
1 #BEFORE# 34      200.
1 #BEFORE# 35      200.
1 #BEFORE# 36      200.
```

```
1 #AFTER #  1      200.
1 #AFTER #  2      200.
1 #AFTER #  3      200.
1 #AFTER #  4      200.
1 #AFTER #  5      200.
1 #AFTER #  6      200.
1 #AFTER #  7      200.
1 #AFTER #  8      200.
1 #AFTER #  9      200.
1 #AFTER # 10      200.
1 #AFTER # 11      200.
1 #AFTER # 12      200.
1 #AFTER # 13      200.
1 #AFTER # 14      200.
1 #AFTER # 15      200.
1 #AFTER # 16      200.
1 #AFTER # 17      200.
1 #AFTER # 18      200.
1 #AFTER # 19      200.
1 #AFTER # 20      200.
1 #AFTER # 21      200.
1 #AFTER # 22      200.
1 #AFTER # 23      200.
1 #AFTER # 24      200.
1 #AFTER # 25      200.
1 #AFTER # 26      100.
1 #AFTER # 27      100.
1 #AFTER # 28      100.
1 #AFTER # 29      100.
1 #AFTER # 30      100.
1 #AFTER # 31      100.
1 #AFTER # 32      100.
1 #AFTER # 33      100.
1 #AFTER # 34      100.
1 #AFTER # 35      100.
1 #AFTER # 36      100.
```

利用例(2):配列の送受信(2/4)

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

if (my_rank.eq.1) then
  call MPI_Isend (VEC( 1),25,MPI_DOUBLE_PRECISION,0,…,req_send,…)
  call MPI_Irecv (VEC(26),11,MPI_DOUBLE_PRECISION,0,…,req_recv,…)
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(3/4)

```
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,…)
call MPI_Irecv                                                    &
(VEC(start_recv),length_recv,MPI_DOUBLE_PRECISION,NEIB,…,req_recv,…)

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

一気にSPMDらしくなる

利用例(2):配列の送受信(4/4)

```
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_Sendrecv                                                    &
(VEC(start_send),length_send,MPI_DOUBLE_PRECISION,NEIB,…             &
 VEC(start_recv),length_recv,MPI_DOUBLE_PRECISION,NEIB,…, status,…)
```

配列送受信（ex2a.f）（1/3）

Isend/Irecv/Waitall

```
$> cd <$S2>
$> mpif90 -O3 ex2a.f
$> mpirun -np 2 a.out
```

```
      integer(kind=4) :: my_rank, PETOT, NEIB
      real   (kind=8) :: VEC(36)

      integer(kind=4), dimension(MPI_STATUS_SIZE),1) :: stat_send
      integer(kind=4), dimension(MPI_STATUS_SIZE),1) :: stat_recv
      integer(kind=4), dimension(1)    :: request_send
      integer(kind=4), dimension(1)    :: request_recv

      integer(kind=4) :: start_send, length_send
      integer(kind=4) :: start_recv, length_recv

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

配列送受信（ex2a.f）（2/3）

Isend/Irecv/Waitall

```
      if (my_rank.eq.0) then
        NEIB= 1
        start_send=   1
        length_send=  11
        start_recv=  length_send + 1
        length_recv=  25
        VEC= 100.
      endif

      if (my_rank.eq.1) then
        NEIB= 0
         start_send=  1
        length_send= 25
         start_recv= length_send + 1
        length_recv= 11
        VEC= 200.
      endif
```

配列送受信（ex2a.f）(3/3)

Isend/Irecv/Waitall

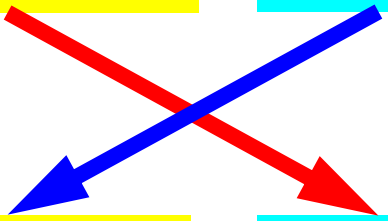
```
      do i= 1, 36
        write (*,'(i1,a,i2,f10.0)') my_rank, ' #BEFORE# ', i,VEC(i)
      enddo

      call MPI_ISEND (VEC(start_send), length_send,                     &
     &                MPI_DOUBLE_PRECISION, NEIB, 0, MPI_COMM_WORLD,    &
     &                request_send(1), ierr)
      call MPI_IRECV (VEC(start_recv), length_recv,                     &
     &                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)

      do i= 1, 36
        write (*,'(i1,a,i2,f10.0)') my_rank, ' #AFTER # ', i,VEC(i)
      enddo

      call MPI_FINALIZE (ierr)

      end
```

- 汎用性がある
- 「データが全て」という気がして来ないだろうか？

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)
```

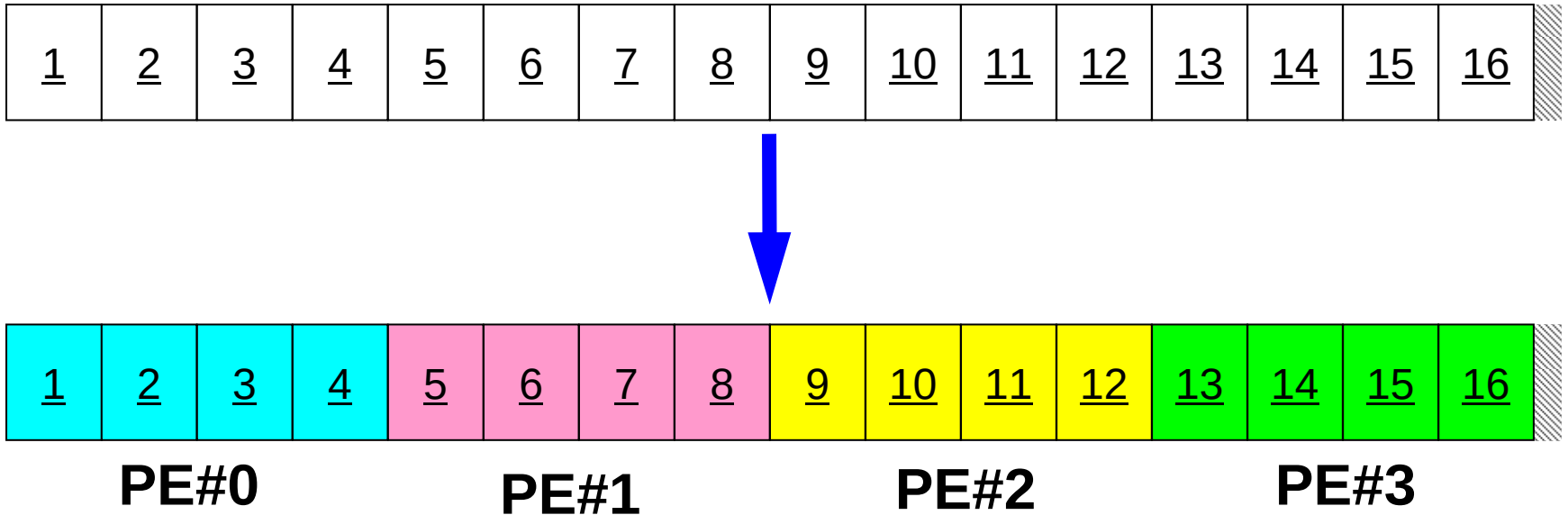
- 送信側の「length_send」と受信側の「length_recv」は一致している必要がある。
  - PE#0⇒PE#1，PE#1⇒PE#0
- 「送信バッファ」と「受信バッファ」は別のアドレス

- 差分法による一次元熱伝導方程式ソルバー
  - 概要
  - 並列化にあたって:データ構造
- 1対1通信とは
- 1対1通信の実装例
  - 一次元問題
  - 二次元問題

一次元差分法モデル局所データ構造(1/5)

NG=16の一次元領域

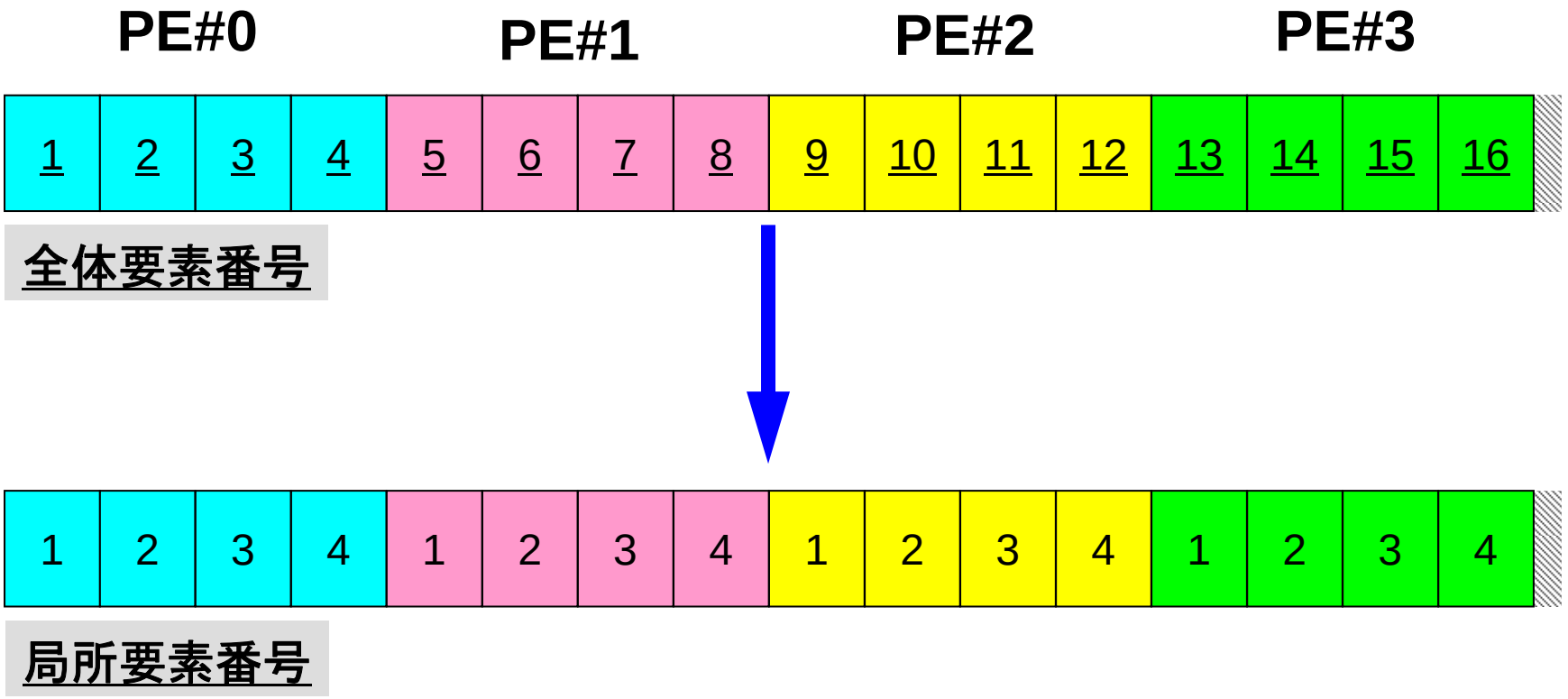
- NG=16の一次元領域を4領域(プロセッサ)に分割して並列に差分法(二次精度)の計算を実施するために必要なデータ構造を考える。



一次元差分法モデル局所データ構造(2/5)

局所要素番号

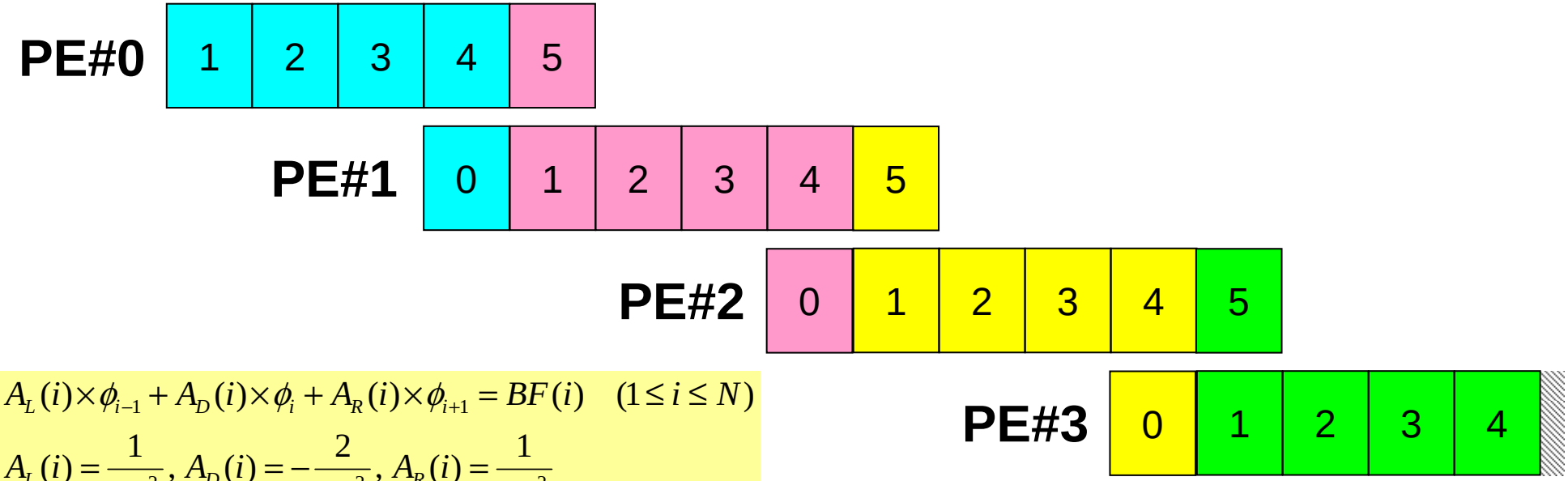
- 各領域(プロセッサ)において，N=4の局所データを扱う



一次元差分法モデル局所データ構造(3/5)

領域外の要素

- 差分法(二次精度中央差分)の計算を並列に実施するためには，本来領域外の要素の影響を考慮する必要がある。
  - 領域間オーバーラップ
- 要素番号が0，N+1(=5)の要素を加えた，局所データ構造



$$A_L(i)\times\phi_{i-1}+A_D(i)\times\phi_i+A_R(i)\times\phi_{i+1}=BF(i)\quad(1\le i\le N)$$

$$A_L(i)=\frac{1}{\Delta x^2},\ A_D(i)=-\frac{2}{\Delta x^2},\ A_R(i)=\frac{1}{\Delta x^2}$$

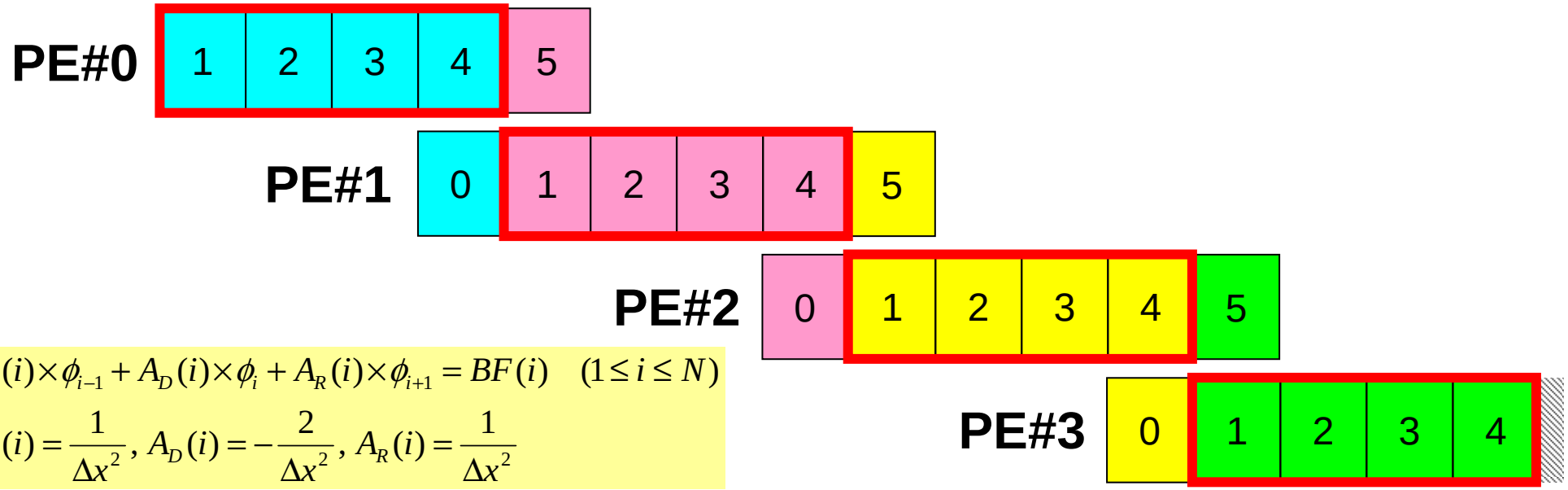
C言語ユーザーの方へ・・・

- お知らせあります。

一次元差分法モデル局所データ構造(4/5)

必要な情報

- 領域内の要素，オーバーラップ(本来領域外)要素の区別
  - 領域内要素(i=1~N)　　:内点(Interior/Internal Points)



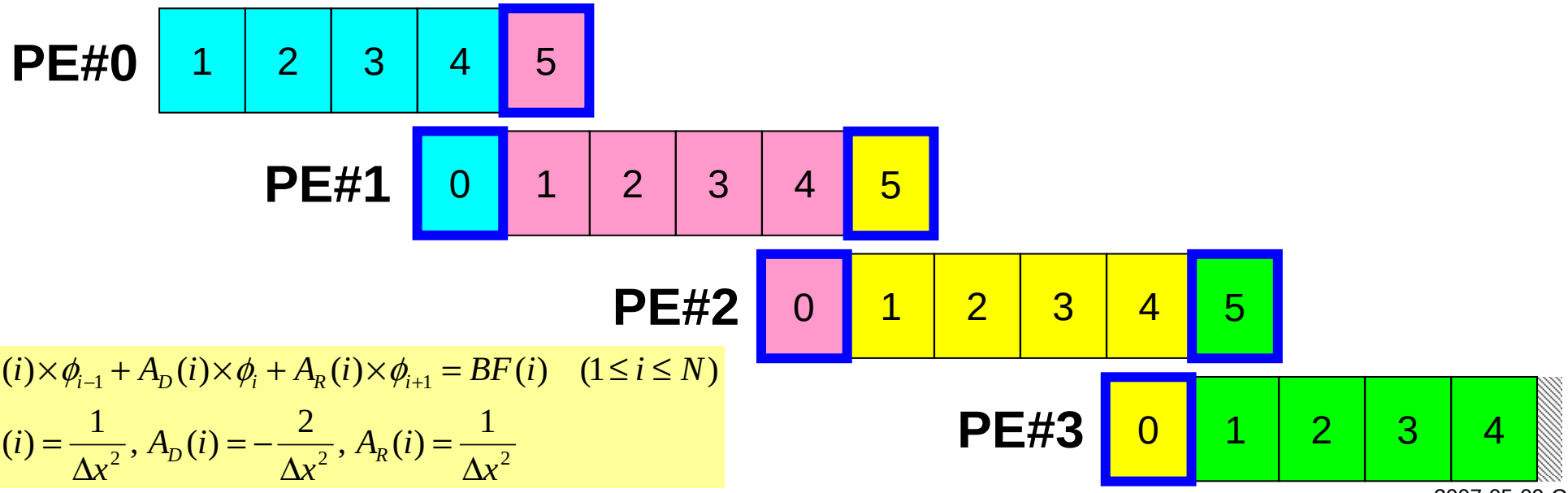
$$A_L(i)\times\phi_{i-1}+A_D(i)\times\phi_i+A_R(i)\times\phi_{i+1}=BF(i)\quad(1\le i\le N)$$

$$A_L(i)=\frac{1}{\Delta x^2},\ A_D(i)=-\frac{2}{\Delta x^2},\ A_R(i)=\frac{1}{\Delta x^2}$$

一次元差分法モデル局所データ構造(4/5)

必要な情報

- 領域内要素，オーバーラップ(本来領域外)要素の区別
  - 領域内要素(i=1~N)　　:内点(Interior/Internal Points)
  - オーバーラップ要素(i=0, N+1):外点(Exterior/External Points)
    - i=0，i=N+1の要素が存在しない場合もある



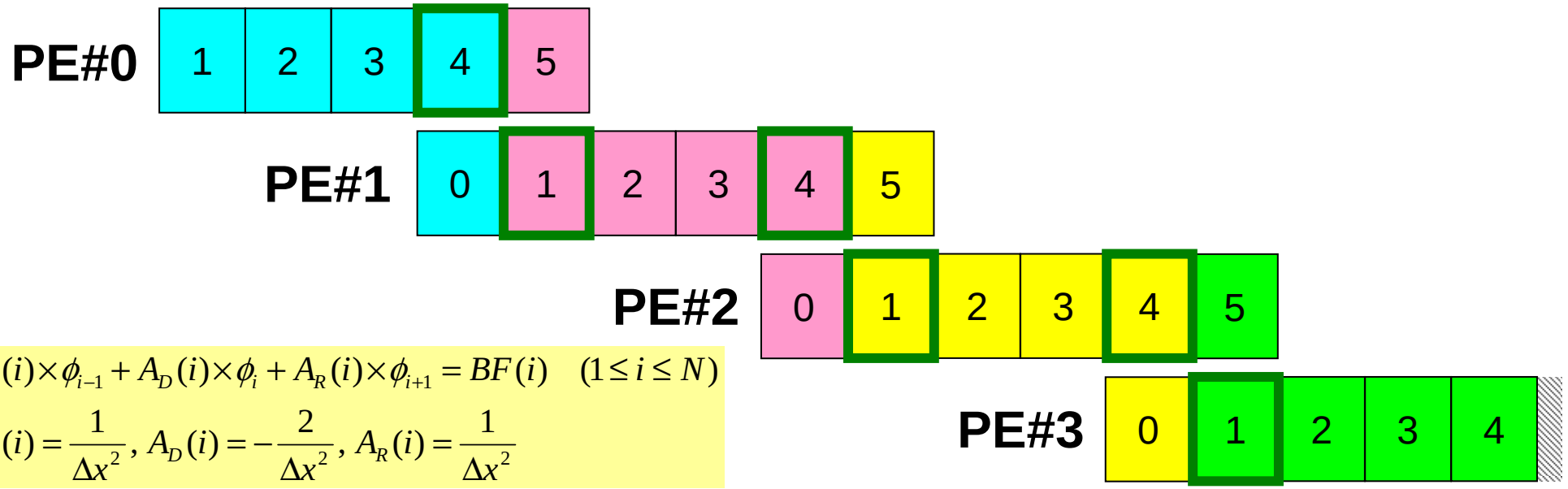
$$A_L(i)\times\phi_{i-1}+A_D(i)\times\phi_i+A_R(i)\times\phi_{i+1}=BF(i)\quad(1\le i\le N)$$

$$A_L(i)=\frac{1}{\Delta x^2},\ A_D(i)=-\frac{2}{\Delta x^2},\ A_R(i)=\frac{1}{\Delta x^2}$$

一次元差分法モデル局所データ構造(4/5)

必要な情報

- 領域内要素，オーバーラップ(本来領域外)要素の区別
  - 領域内要素(i=1~N)　　:内点(Interior/Internal Points)
  - オーバーラップ要素(i=0, N+1):外点(Exterior/External Points)
    - i=0，i=N+1の要素が存在しない場合もある
  - 「内点」のうち他領域の「外点」となっている要素:境界点(Boundary Points)



$$A_L(i)\times\phi_{i-1}+A_D(i)\times\phi_i+A_R(i)\times\phi_{i+1}=BF(i)\quad(1\le i\le N)$$

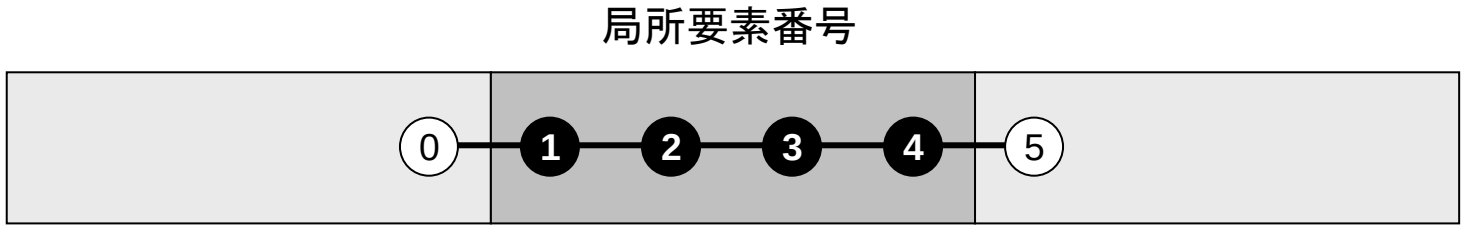
$$A_L(i)=\frac{1}{\Delta x^2},\ A_D(i)=-\frac{2}{\Delta x^2},\ A_R(i)=\frac{1}{\Delta x^2}$$

一次元差分法モデル局所データ構造(5/5)

必要な情報(続き)

- 「領域間オーバーラップ」を考慮した並列計算を実現するためには？
  - 各領域(プロセッサ)における「境界点」の情報を，各隣接領域に「外点」の情報として配信する必要がある。
  - そのための局所データ構造が必要
- 隣接している領域数，隣接している領域番号
  - 1対1通信の場合重要
  - オーバーラップ要素を共有している領域(プロセッサ)
- 隣接領域との「通信テーブル」
  - 境界点：隣接領域への「送信」(send)
  - 外点　：隣接領域からの「受信」(recv)

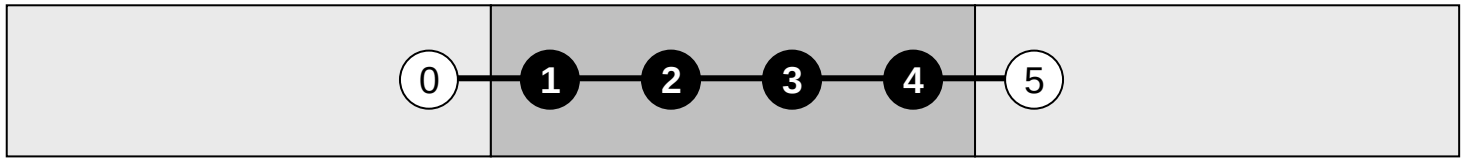
1D差分法の並列計算に必要な情報(1/3)
内点・外点・(境界点)



1D差分法の並列計算に必要な情報(2/3)
隣接プロセッサ情報

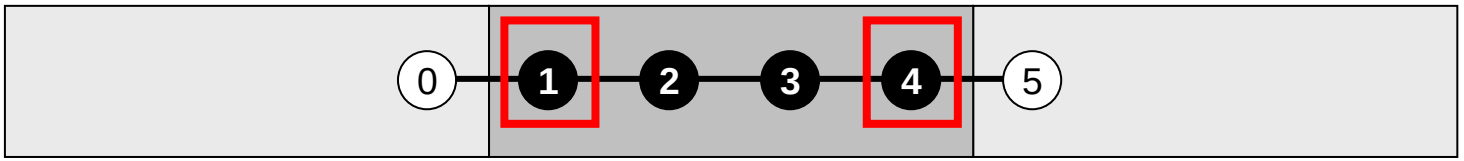
PE#(my_rank-1): NEIBPE(1)

PE#(my_rank+1): NEIBPE(2)



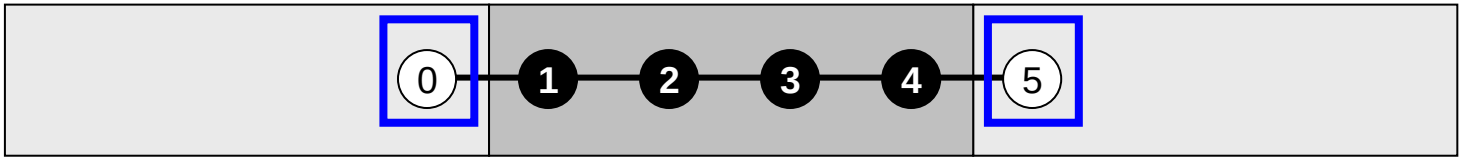
1D差分法の並列計算に必要な情報(3/3) 通信テーブル

隣接PEへの送信(境界点)

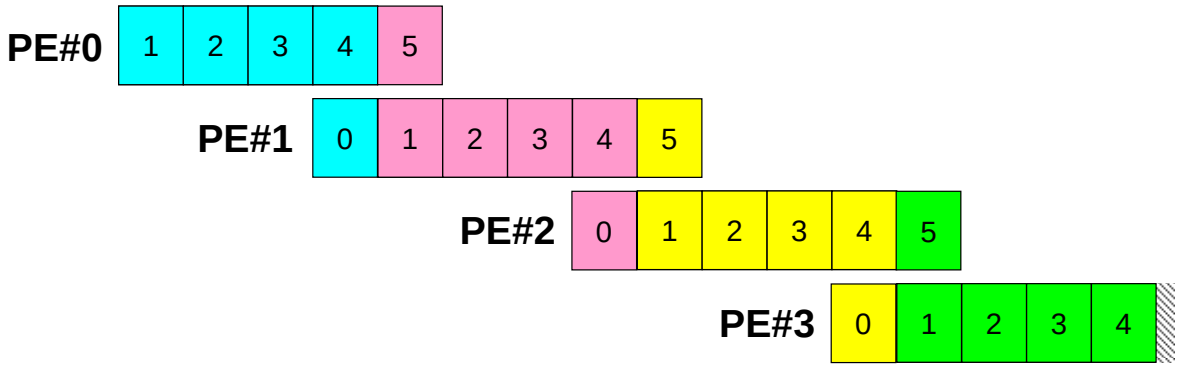



```
SENDbuf(1)=BUF(1)
SENDbuf(2)=BUF(4)
```

隣接PEからの受信(外点)

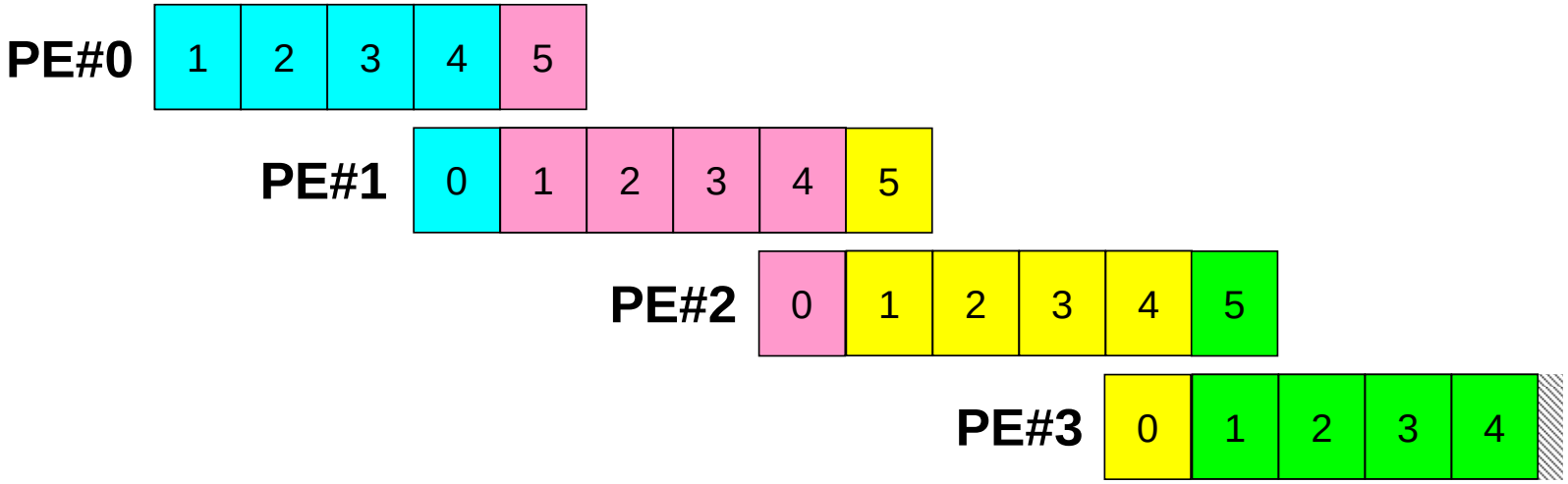



```
BUF(0)=RECVbuf(1)
BUF(5)=RECVbuf(2)
```

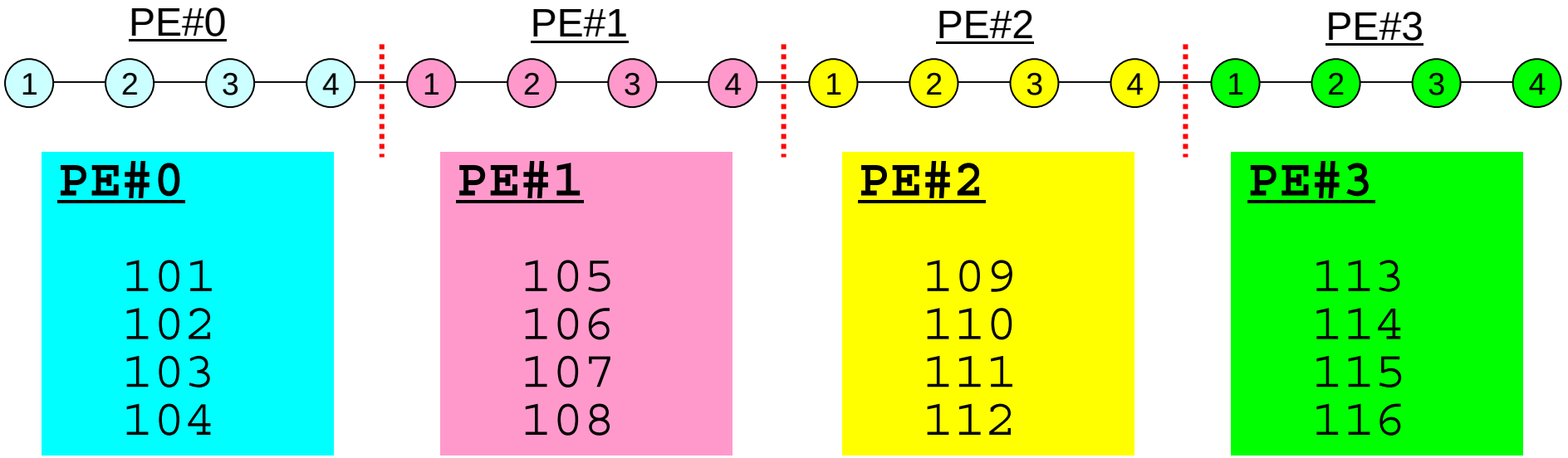


一次元モデル例題(1/10)

- オーバーラップのある一次元分散データ(NG=16，4領域)について，各プロセッサで「内点」に関する情報を入力し，「境界点」の情報を，各隣接領域に「外点」の情報として配信する。

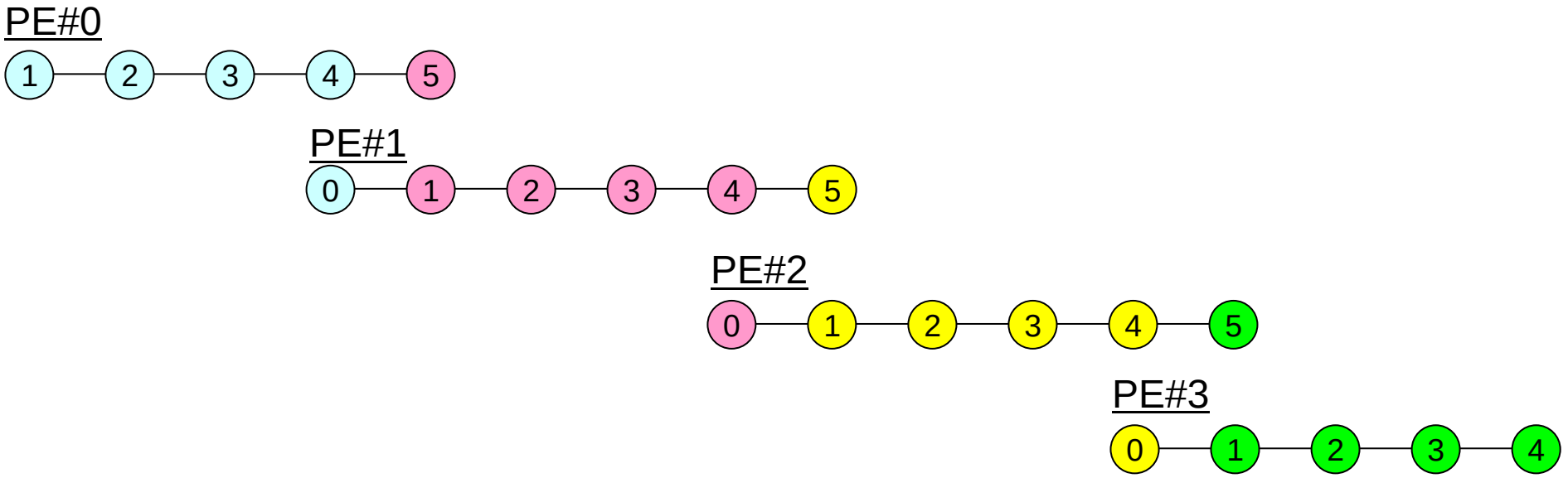


演算内容(1/3)



- 各PEの内点(i=1~N(=4))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」として配信する。

演算内容(2/3):送信，受信前



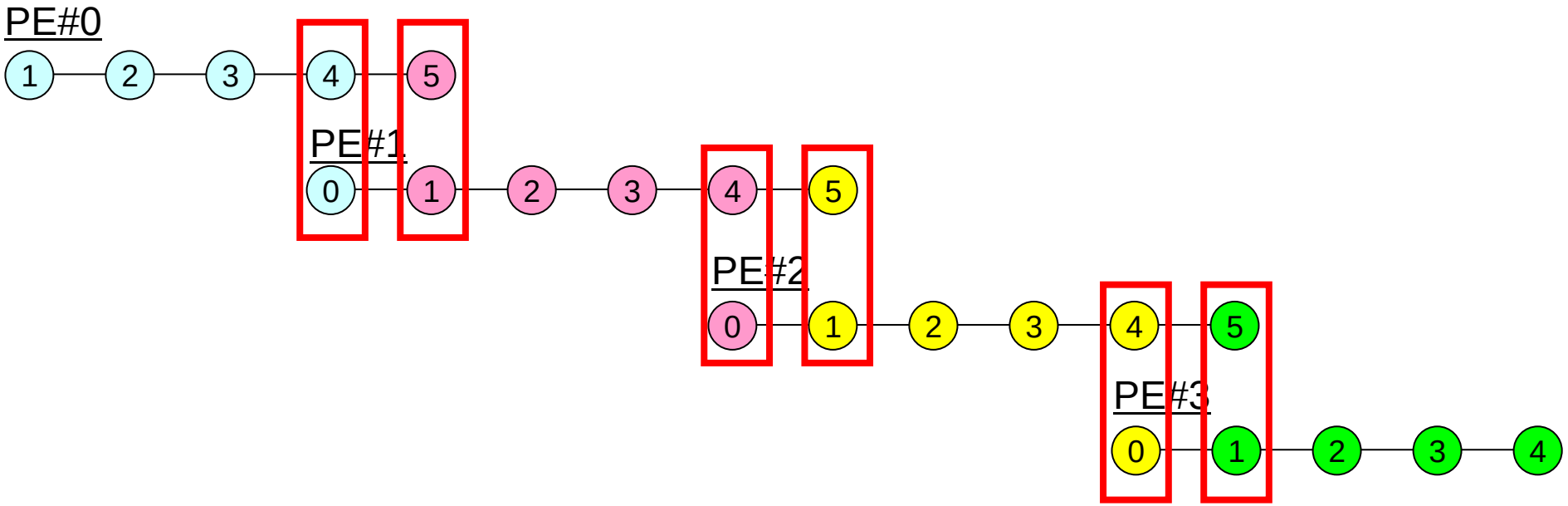
PE#0
```
i=1 101
i=2 102
i=3 103
i=4 104
i=5  ?
```

PE#1
```
i=0  ?
i=1 105
i=2 106
i=3 107
i=4 108
i=5  ?
```

PE#2
```
i=0  ?
i=1 109
i=2 110
i=3 111
i=4 112
i=5  ?
```

PE#3
```
i=0  ?
i=1 113
i=2 114
i=3 115
i=4 116
```

演算内容(3/3):送信，受信後



PE#0
i=1 101
i=2 102
i=3 103
i=4 104
i=5 105

PE#1
i=0 104
i=1 105
i=2 106
i=3 107
i=4 108
i=5 109

PE#2
i=0 108
i=1 109
i=2 110
i=3 111
i=4 112
i=5 113

PE#3
i=0 112
i=1 113
i=2 114
i=3 115
i=4 116

サンプルプログラム：一次元

FORTRAN

```
$ cd <$S2>
$ mpif90 -O3 1d-sr1.f
$ mpirun -np 4 a.out
```

ISEND/IRECV

```
$ mpif90 -O3 1d-sr2.f
$ mpirun -np 4 a.out
```

SENDRECV

C

```
$ cd <$S2C>
$ mpicc -O3 1d-sr1.c
$ mpirun -np 4 a.out
```

ISEND/IRECV

```
$ mpicc -O3 1d-sr2.c
$ mpirun -np 4 a.out
```

SENDRECV

一次元モデル例題: 1d-sr1.f（2/10）

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

      integer(kind=4) :: my_rank, PETOT
      integer(kind=4) :: N, NEIBPETOT, BUFlength
      integer(kind=4), dimension(2) :: NEIBPE
      integer(kind=4), dimension(2) :: SENDbuf, RECVbuf

      integer(kind=4), dimension(:), allocatable :: BUF

      integer(kind=4), dimension(:,:), allocatable :: stat_send
      integer(kind=4), dimension(:,:), allocatable :: stat_recv
      integer(kind=4), dimension(:  ), allocatable :: request_send
      integer(kind=4), dimension(:  ), allocatable :: request_recv

!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===
```

一次元モデル例題: 1d-sr1.f（3/10）

各PEにおいて局所データ（内点の値）読み込み

```
!C
!C +------+
!C | INIT |
!C +------+
!C===
      N= 4
      allocate (BUF(0:N+1))
      BUF= 0

      if (my_rank.eq.0) open (11, file='1d.0', status='unknown')
      if (my_rank.eq.1) open (11, file='1d.1', status='unknown')
      if (my_rank.eq.2) open (11, file='1d.2', status='unknown')
      if (my_rank.eq.3) open (11, file='1d.3', status='unknown')

        do i= 1, N
          read (11,*) BUF(i)
        enddo

      close (11)
```

N: 内点数

PE#0	PE#1	PE#2	PE#3
101	105	109	113
102	106	110	114
103	107	111	115
104	108	112	116

一次元モデル例題: 1d-sr1.f（4/10）

各領域における外点

```
      iS= 0
      iE= N+1

      if (my_rank.eq.0      ) iS= 1
      if (my_rank.eq.PETOT-1) iE= N
      do i= iS, iE
        write (*,'(a, 3i8)') '%%% before', my_rank, i, BUF(i)
      enddo
```

PE#0
- iS= 1
- iE= N+1
- 外点数:1

```
%%% before    0    1    101
%%% before    0    2    102
%%% before    0    3    103
%%% before    0    4    104
%%% before    0    5      0
```

PE#1
- iS= 0
- iE= N+1
- 外点数:2

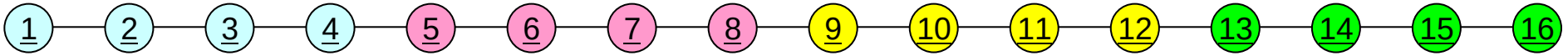
```
%%% before    1    0      0
%%% before    1    1    105
%%% before    1    2    106
%%% before    1    3    107
%%% before    1    4    108
%%% before    1    5      0
```

PE#2
- iS= 0
- iE= N+1
- 外点数:2

```
%%% before    2    0      0
%%% before    2    1    109
%%% before    2    2    110
%%% before    2    3    111
%%% before    2    4    112
%%% before    2    5      0
```

PE#3
- iS= 0
- iE= N
- 外点数:1

```
%%% before    3    0      0
%%% before    3    1    113
%%% before    3    2    114
%%% before    3    3    115
%%% before    3    4    116
```



一次元モデル例題: 1d-sr1.f（5/10）

隣接領域，領域数

```
      NEIBPETOT= 2
      if (my_rank.eq.0       ) NEIBPETOT= 1
      if (my_rank.eq.PETOT-1) NEIBPETOT= 1

      NEIBPE(1)= my_rank - 1
      NEIBPE(2)= my_rank + 1

      if (my_rank.eq.0       ) NEIBPE(1)= my_rank + 1
      if (my_rank.eq.PETOT-1) NEIBPE(1)= my_rank - 1

      BUFlength= 1
```

隣接PE数（NEIBPETOT），
隣接PE（NEIBPE）を決定

一次元モデル例題: 1d-sr1.f（6/10）

通信識別子等の宣言，記憶領域確保

```
!C
!C +-----------------------------+
!C | INIT. arrays for MPI_WAITALL |
!C +-----------------------------+
!C===
      allocate (stat_send(MPI_STATUS_SIZE,NEIBPETOT))
      allocate (stat_recv(MPI_STATUS_SIZE,NEIBPETOT))
      allocate (request_send(NEIBPETOT))
      allocate (request_recv(NEIBPETOT))
!C===
```

- 記憶領域を確保するだけで良い!!
- 通信識別子：request handle（通信リクエスト，とも言う）
  - 送信用：stat_send，受信用：stat_recv
- 状況オブジェクト配列：status object
  - 送信用：request_send，受信用：request_recv
- MPI_STATUS_SIZE
  - “mpif.h”，”mpi.h” で定義されている。
  - 勝手にこの数字を変更してはいけない。

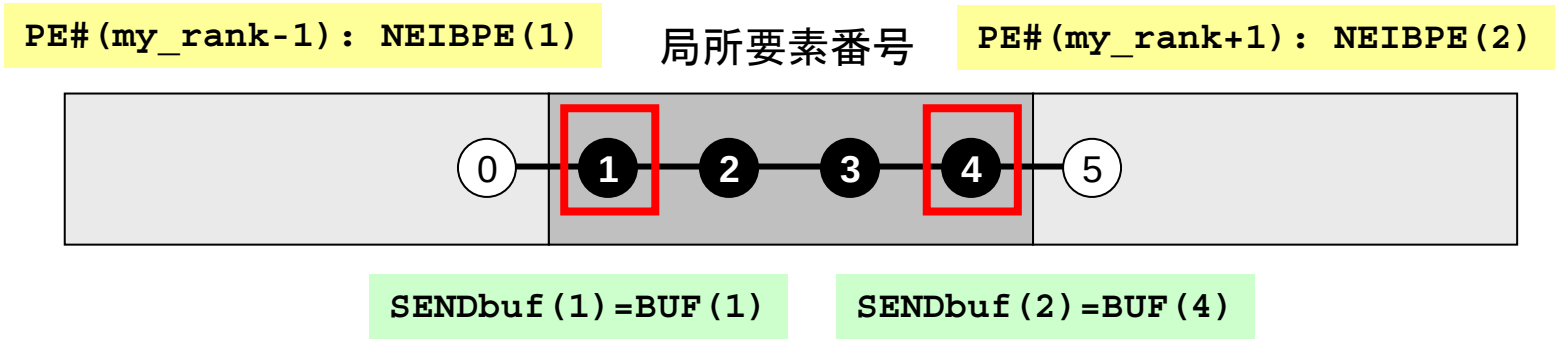
一次元モデル例題: 1d-sr1.f（7/10）

境界点情報送信準備

```
!C
!C +-----------------------+
!C | PREPARE send  buffer |
!C +-----------------------+
!C===
      SENDbuf(1)= BUF(1)
      SENDbuf(2)= BUF(N)
      if (my_rank.eq.0       ) SENDbuf(1)= BUF(N)
      if (my_rank.eq.PETOT-1) SENDbuf(1)= BUF(1)
!C===
```

隣接プロセッサに BUF(1), BUF(N) を送る準備をする

境界点情報



一次元モデル例題: 1d-sr1.f（8/10）

境界点情報の送信

```
!C
!C +------+
!C | SEND |
!C +------+
!C===
      do neib= 1, NEIBPETOT
        call MPI_ISEND (SENDbuf(neib), BUFlength, MPI_INTEGER,          &
     &                  NEIBPE(neib), 0, MPI_COMM_WORLD,                &
     &                  request_send(neib), ierr)
      enddo
!C===


!C
!C +------+
!C | RECV |
!C +------+
!C===
      do neib= 1, NEIBPETOT
        call MPI_IRECV (RECVbuf(neib), BUFlength, MPI_INTEGER,          &
     &                  NEIBPE(neib), 0, MPI_COMM_WORLD,                &
     &                  request_recv(neib), ierr)
      enddo
!C===

!C
!C +------------------+
!C | WAITall for RECV |
!C +------------------+
!C===
      call MPI_WAITALL (NEIBPETOT, request_recv, stat_recv, ierr)
!C===
```

SENDbuf(neib)から始まる，長さBUFlength(=1)のメッセージを隣接PE(NEIBPE(neib))に送信する。

一次元モデル例題: 1d-sr1.f（8/10）

外点情報の受信

```
!C
!C +------+
!C | SEND |
!C +------+
!C===
      do neib= 1, NEIBPETOT
        call MPI_ISEND (SENDbuf(neib), BUFlength, MPI_INTEGER,           &
     &                  NEIBPE(neib), 0, MPI_COMM_WORLD,                 &
     &                  request_send(neib), ierr)
      enddo
!C===


!C
!C +------+
!C | RECV |
!C +------+
!C===
      do neib= 1, NEIBPETOT
        call MPI_IRECV (RECVbuf(neib), BUFlength, MPI_INTEGER,           &
     &                  NEIBPE(neib), 0, MPI_COMM_WORLD,                 &
     &                  request_recv(neib), ierr)
      enddo
!C===

!C
!C +------------------+
!C | WAITall for RECV |
!C +------------------+
!C===
      call MPI_WAITALL (NEIBPETOT, request_recv, stat_recv, ierr)
!C===
```

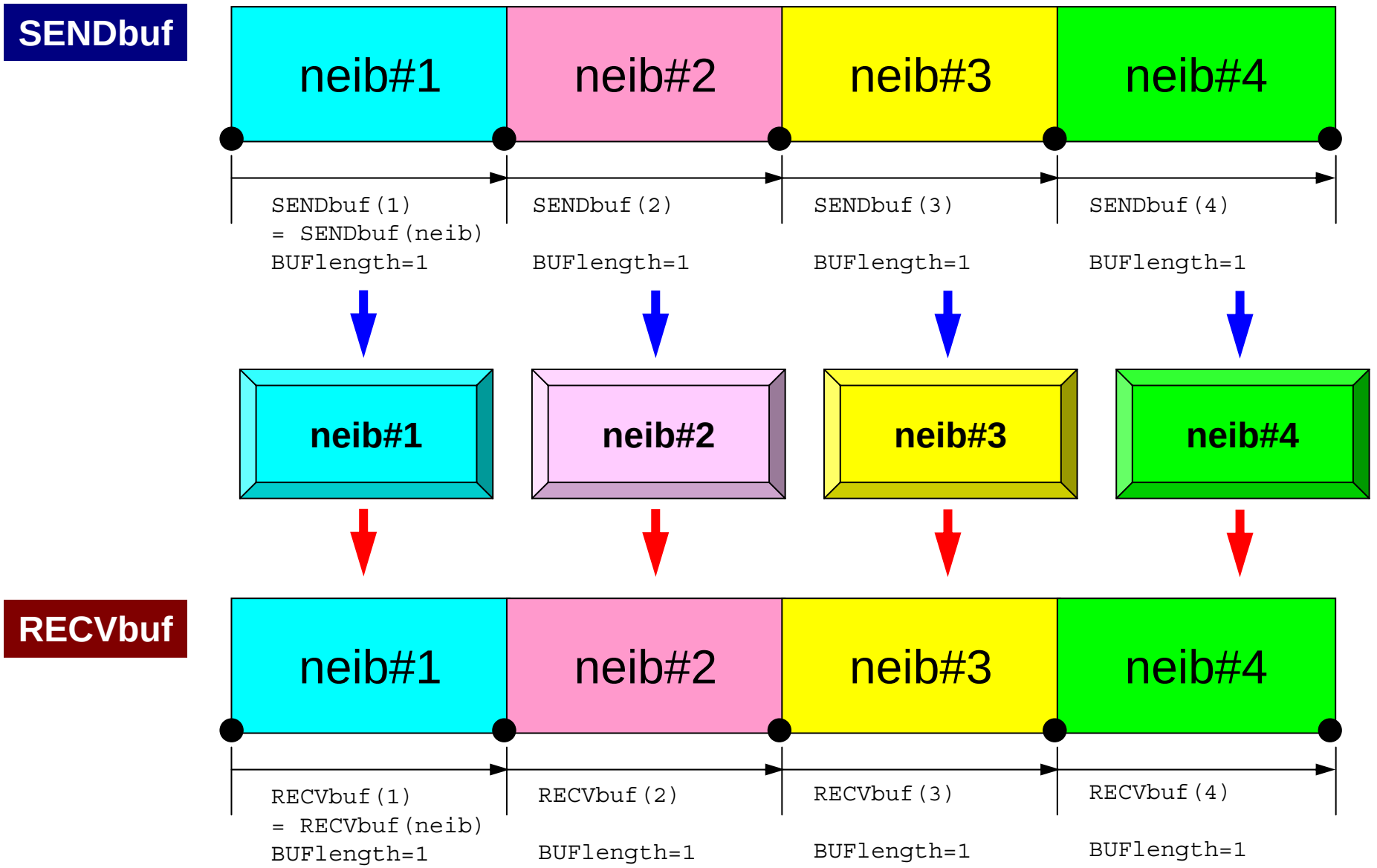
RECVbuf(neib)から始まる，長さBUFlength(=1)のメッセージを隣接PE(NEIBPE(neib))から受信する。

実際何をやっているのか

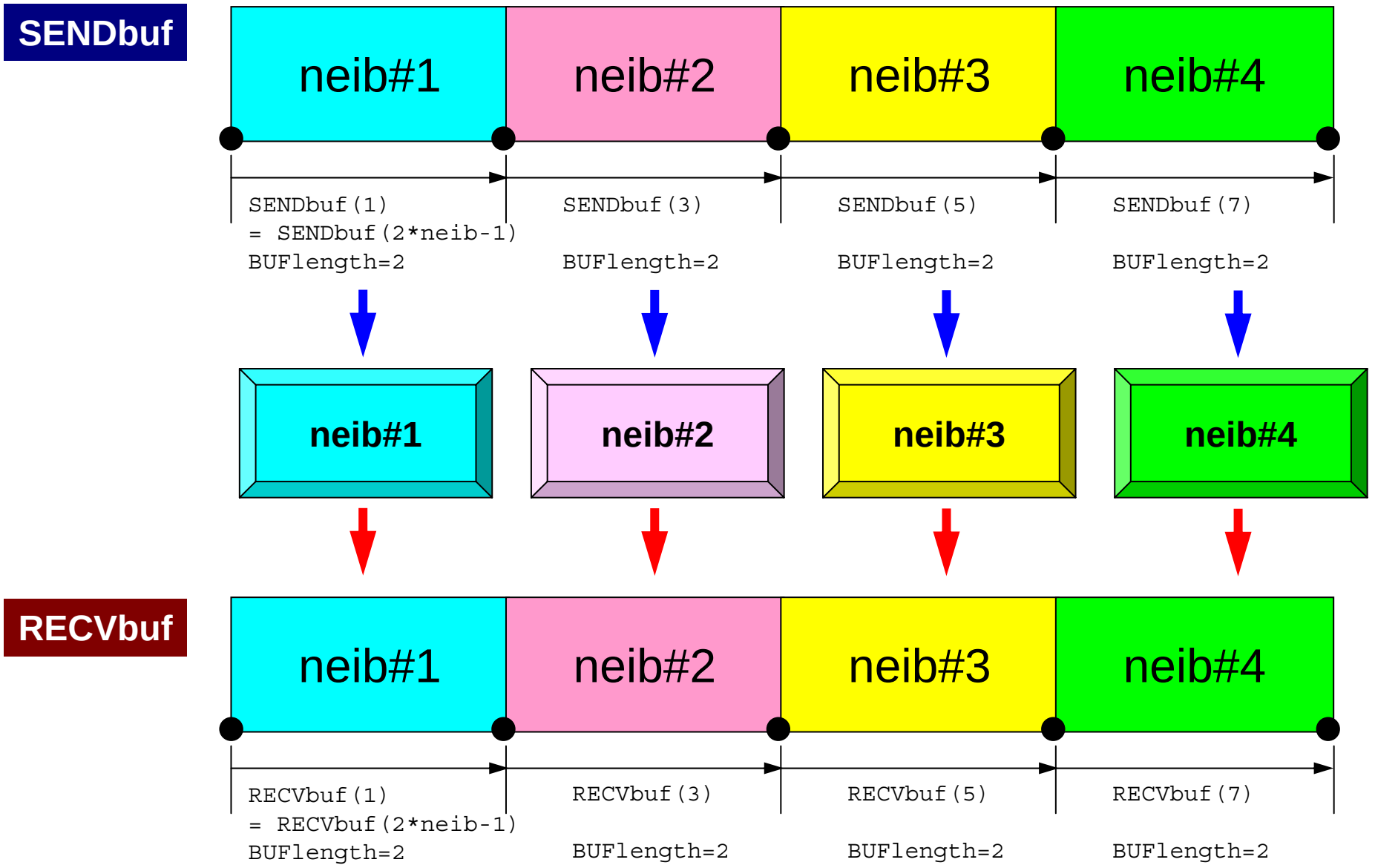
```
!C
!C +------+
!C | SEND |
!C +------+
!C===
      do neib= 1, NEIBPETOT
        call MPI_ISEND (SENDbuf(neib), BUFlength, MPI_INTEGER,           &
     &                  NEIBPE(neib), 0, MPI_COMM_WORLD,                 &
     &                  request_send(neib), ierr)
      enddo
!C===


!C
!C +------+
!C | RECV |
!C +------+
!C===
      do neib= 1, NEIBPETOT
        call MPI_IRECV (RECVbuf(neib), BUFlength, MPI_INTEGER,           &
     &                  NEIBPE(neib), 0, MPI_COMM_WORLD,                 &
     &                  request_recv(neib), ierr)
      enddo
!C===
```

- 例えば，PE#1からはPE#0，PE#2に送信して，PE#0，PE#2から受信する。
- PE#0はPE#1から，PE#2はPE#1とPE#3から受信する。
- IRECVの部分では，相手のPE（NEIBPE(neib)）がISENDのループで自分に向けて送ったメッセージを受信する。

BUFlength = 1


SENDbuf
neib#1
neib#2
neib#3
neib#4
SENDbuf(1)
= SENDbuf(neib)
BUFlength=1
SENDbuf(2)
BUFlength=1
SENDbuf(3)
BUFlength=1
SENDbuf(4)
BUFlength=1
neib#1
neib#2
neib#3
neib#4
RECVbuf
neib#1
neib#2
neib#3
neib#4
RECVbuf(1)
= RECVbuf(neib)
BUFlength=1
RECVbuf(2)
BUFlength=1
RECVbuf(3)
BUFlength=1
RECVbuf(4)
BUFlength=1

BUFlength = 2


SENDbuf
neib#1
neib#2
neib#3
neib#4
SENDbuf(1)
= SENDbuf(2*neib-1)
BUFlength=2
SENDbuf(3)
BUFlength=2
SENDbuf(5)
BUFlength=2
SENDbuf(7)
BUFlength=2
neib#1
neib#2
neib#3
neib#4
RECVbuf
neib#1
neib#2
neib#3
neib#4
RECVbuf(1)
= RECVbuf(2*neib-1)
BUFlength=2
RECVbuf(3)
BUFlength=2
RECVbuf(5)
BUFlength=2
RECVbuf(7)
BUFlength=2

一次元モデル例題: 1d-sr1.f（8/10）

```
!C
!C +------+
!C | SEND |
!C +------+
!C===
      do neib= 1, NEIBPETOT
        call MPI_ISEND (SENDbuf(neib), BUFlength, MPI_INTEGER,           &
     &                  NEIBPE(neib), 0, MPI_COMM_WORLD,                 &
     &                  request_send(neib), ierr)
      enddo
!C===


!C
!C +------+
!C | RECV |
!C +------+
!C===
      do neib= 1, NEIBPETOT
        call MPI_IRECV (RECVbuf(neib), BUFlength, MPI_INTEGER,           &
     &                  NEIBPE(neib), 0, MPI_COMM_WORLD,                 &
     &                  request_recv(neib), ierr)
      enddo
!C===

!C
!C +------------------+
!C | WAITall for RECV |
!C +------------------+
!C===
      call MPI_WAITALL (NEIBPETOT, request_recv, stat_recv, ierr)
!C===
```

このメッセージのあと，RECVbufの中身を利用することが可能となる。

一次元モデル例題: 1d-sr1.f（8/10）

```
!C
!C +------+
!C | SEND |
!C +------+
!C===
      do neib= 1, NEIBPETOT
        call MPI_ISEND (SENDbuf(neib), BUFlength, MPI_INTEGER,           &
     &                  NEIBPE(neib), 0, MPI_COMM_WORLD,                 &
     &                  request_send(neib), ierr)
      enddo
!C===


!C
!C +------+
!C | RECV |
!C +------+
!C===
      do neib= 1, NEIBPETOT
        call MPI_IRECV (RECVbuf(neib), BUFlength, MPI_INTEGER,           &
     &                  NEIBPE(neib), 0, MPI_COMM_WORLD,                 &
     &                  request_recv(neib), ierr)
      enddo
!C===

!C
!C +------------------+
!C | WAITall for RECV |
!C +------------------+
!C===
      call MPI_WAITALL (NEIBPETOT, request_recv, stat_recv, ierr)
!C===
```

通信識別子（受信）

このメッセージのあと，RECVbufの中身を利用することが可能となる。

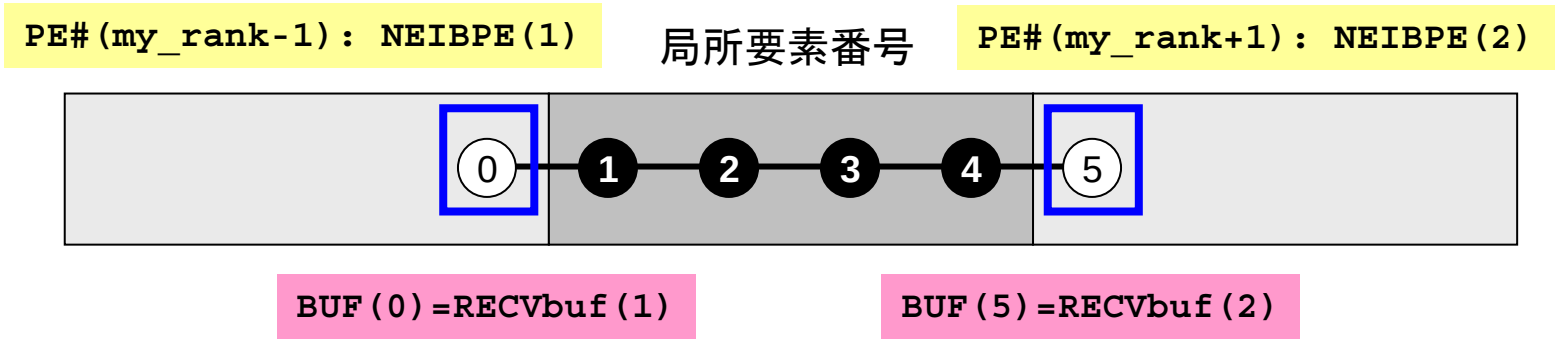
通信識別子　状況オブジェクト配列

一次元モデル例題: 1d-sr1.f（9/10）

外点情報代入

```
!C
!C +--------------+
!C | update array |
!C +--------------+
!C===
      if (my_rank.eq.0) then
        BUF(N+1)= RECVbuf(1)
       else if
     &   (my_rank.eq.PETOT-1) then
        BUF(0)  = RECVbuf(1)
       else
        BUF(0  )= RECVbuf(1)
        BUF(N+1)= RECVbuf(2)
      endif

      do i= iS, iE
        write (*,'(a, 3i8)') '### after', my_rank, i, BUF(i)
      enddo
!C===
```

隣接プロセッサから受け取った値をBUF(0), BUF(N+1)に代入する。



一次元モデル例題: 1d-sr1.f（10/10）

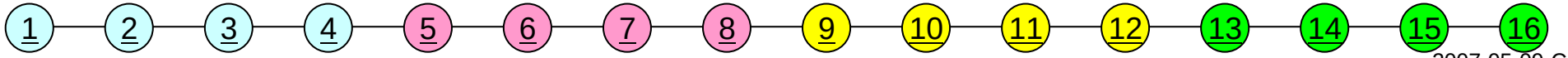
```
!C
!C +--------------+
!C | update array |
!C +--------------+
!C===
      if (my_rank.eq.0) then
        BUF(N+1)= RECVbuf(1)
       else if                                                    &
     &    (my_rank.eq.PETOT-1) then
        BUF(0)   = RECVbuf(1)
       else
        BUF(0  )= RECVbuf(1)
        BUF(N+1)= RECVbuf(2)
      endif
      do i= iS, iE
        write (*,'(a, 3i8)') '### after', my_rank, i, BUF(i)
      enddo
!C===

!C
!C +------------------+
!C | WAITall for SEND |
!C +------------------+
!C===
      call MPI_WAITALL (NEIBPETOT, request_send, stat_send, ierr)
!C===
```

```
%%% after   0   1   101
%%% after   0   2   102
%%% after   0   3   103
%%% after   0   4   104
%%% after   0   5   105
```

```
%%% after   1   0   104
%%% after   1   1   105
%%% after   1   2   106
%%% after   1   3   107
%%% after   1   4   108
%%% after   1   5   109
```

```
%%% after   2   0   108
%%% after   2   1   109
%%% after   2   2   110
%%% after   2   3   111
%%% after   2   4   112
%%% after   2   5   113
```

```
%%% after   3   0   112
%%% after   3   1   113
%%% after   3   2   114
%%% after   3   3   115
%%% after   3   4   116
```



一次元モデル例題: 1d-sr1.f（10/10）

```
!C
!C +--------------+
!C | update array |
!C +--------------+
!C===
      if (my_rank.eq.0) then
        BUF(N+1)= RECVbuf(1)
       else if                                                    &
     &    (my_rank.eq.PETOT-1) then
        BUF(0)   = RECVbuf(1)
       else
        BUF(0   )= RECVbuf(1)
        BUF(N+1)= RECVbuf(2)
      endif
      do i= iS, iE
        write (*,'(a, 3i8)') '### after', my_rank, i, BUF(i)
      enddo
!C===

!C
!C +------------------+
!C | WAITall for SEND |
!C +------------------+
!C===
      call MPI_WAITALL (NEIBPETOT, request_send, stat_send, ierr)
!C===
```

このメッセージのあと，SENDbufの中身を変更することが可能となる。

一次元モデル例題: 1d-sr1.f（10/10）

```
!C
!C +--------------+
!C | update array |
!C +--------------+
!C===
      if (my_rank.eq.0) then
        BUF(N+1)= RECVbuf(1)
       else if                                                    &
     &    (my_rank.eq.PETOT-1) then
        BUF(0)   = RECVbuf(1)
       else
        BUF(0   )= RECVbuf(1)
        BUF(N+1)= RECVbuf(2)
      endif
      do i= iS, iE
        write (*,'(a, 3i8)') '### after', my_rank, i, BUF(i)
      enddo
!C===

!C
!C +------------------+
!C | WAITall for SEND |
!C +------------------+
!C===
      call MPI_WAITALL (NEIBPETOT, request_send, stat_send, ierr)
!C===
```

通信識別子（送信用）　状況オブジェクト配列（送信用）

MPI_ISEND/IRECV/WAITALLと MPI_SENDRECV

$S2>/1d-sr1.f: Isend/Irecv/Waitall

```
      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
        call MPI_ISEND (SENDbuf(neib), BUFlength, MPI_INTEGER,           &
     &                  NEIBPE(neib), 0, MPI_COMM_WORLD,                 &
     &                  request_send(neib), ierr)
      enddo
…
      do neib= 1, NEIBPETOT
        call MPI_IRECV (RECVbuf(neib), BUFlength, MPI_INTEGER,           &
     &                  NEIBPE(neib), 0, MPI_COMM_WORLD,                 &
     &                  request_recv(neib), ierr)
      enddo
…
      call MPI_WAITALL (NEIBPETOT, request_recv, stat_recv, ierr)
      call MPI_WAITALL (NEIBPETOT, request_send, stat_recv, ierr)
```

$S2>/1d-sr2.f: Sendrecv

```
      allocate (stat_sr(MPI_STATUS_SIZE))
…
      do neib= 1, NEIBPETOT
        call MPI_SENDRECV                                                &
     &        (SENDbuf(neib), BUFlength, MPI_INTEGER, NEIBPE(neib), 0,   &
     &         RECVbuf(neib), BUFlength, MPI_INTEGER, NEIBPE(neib), 0,   &
     &         MPI_COMM_WORLD, stat_sr, ierr)
      enddo
```

MPI_ISEND/IRECV/WAITALLの使い方

$S2>/1d-sr1.f: Isend/Irecv/Waitall

```
      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
        call MPI_ISEND (SENDbuf(neib), BUFlength, MPI_INTEGER,          &
     &                  NEIBPE(neib), 0, MPI_COMM_WORLD,                &
     &                  request_send(neib), ierr)
      enddo
…
      do neib= 1, NEIBPETOT
        call MPI_IRECV (RECVbuf(neib), BUFlength, MPI_INTEGER,          &
     &                  NEIBPE(neib), 0, MPI_COMM_WORLD,                &
     &                  request_recv(neib), ierr)
      enddo
…
      call MPI_WAITALL (NEIBPETOT, request_recv, stat_recv, ierr)
      call MPI_WAITALL (NEIBPETOT, request_send, stat_recv, ierr)
```

- 通信識別子 request を定義：送信用，受信用
- 状況オブジェクト配列 status を定義：送信用，受信用
- MPI_Isend，MPI_Irecvにそれぞれ対応したMPI_Waitallを呼ぶ
  - request，statusで区別

MPI_SENDRECVの使い方

<$S2>/1d-sr2.f: Sendrecv

```
      allocate (stat_sr(MPI_STATUS_SIZE))
…
      do neib= 1, NEIBPETOT
        call MPI_SENDRECV                                              &
     &         (SENDbuf(neib), BUFlength, MPI_INTEGER, NEIBPE(neib), 0,  &
     &          RECVbuf(neib), BUFlength, MPI_INTEGER, NEIBPE(neib), 0,  &
     &          MPI_COMM_WORLD, stat_sr, ierr)
      enddo
```

- 状況オブジェクト配列 status
  - status(MPI_STATUS_SIZE) を定義するだけで良い
  - Isend/Irecv/Waitallのときとは statusのサイズが異なるので注意すること。

- 差分法による一次元熱伝導方程式ソルバー
  - 概要
  - 並列化にあたって:データ構造
- 1対1通信とは
- 1対1通信の実装例
  - 一次元問題
  - 二次元問題

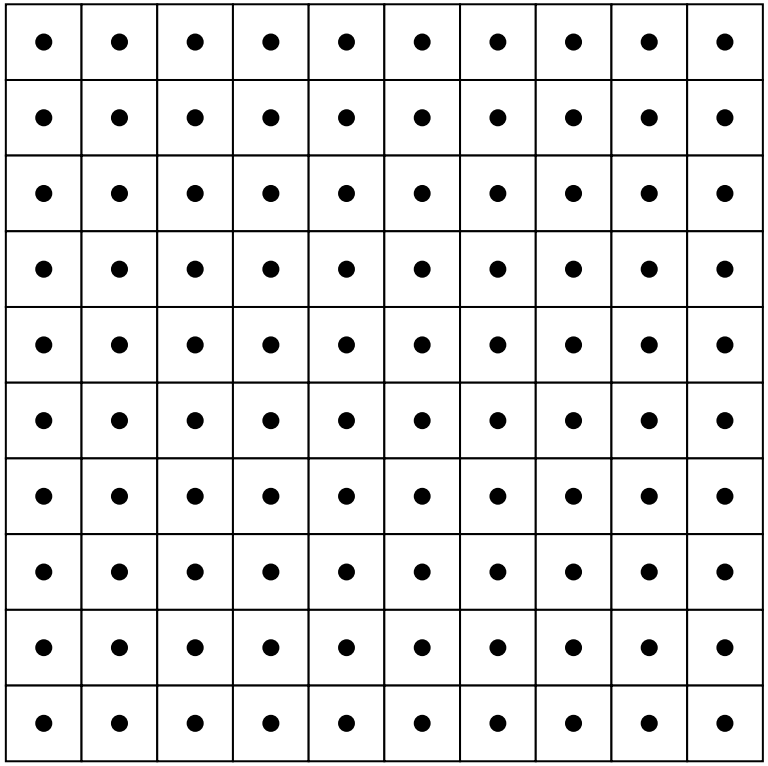
ポイント

- 一次元の場合
  - 領域において i=1~Nの要素があるとすると，基本的に「0」番目と「N+1」番目が他領域からのデータである。
- 二次元の場合
  - もっと難しい。
  - 差分格子の場合「MPI_CART_XXXX」というサブルーチン群を使って並列化を簡単に実施可能（気候・気象シミュレーション）。
  - ここでは，「二次元差分法」を例にとって，より一般的な場合にも対応できるような方法を考えてみる。

- アルゴリズムとデータ構造
  - アルゴリズム（並列二次元差分法）を実現するために適切なデータ構造を設計する。

二次元差分法(1/5)

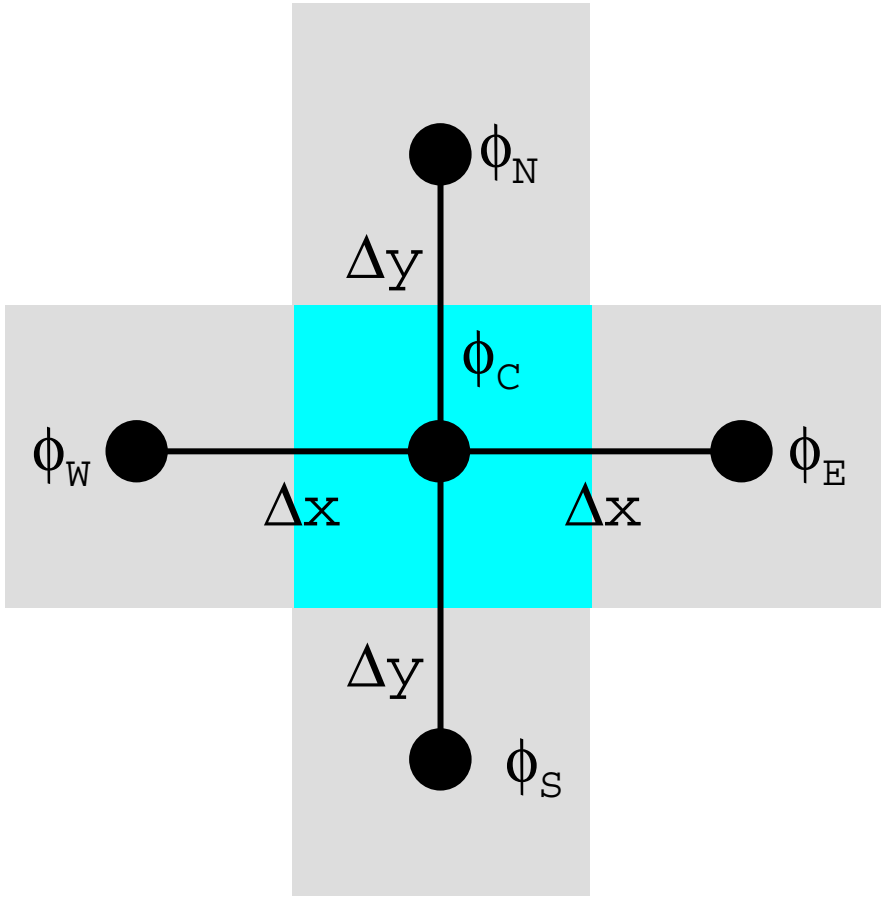
全体メッシュ



二次元中央差分法(5点差分法)の定式化

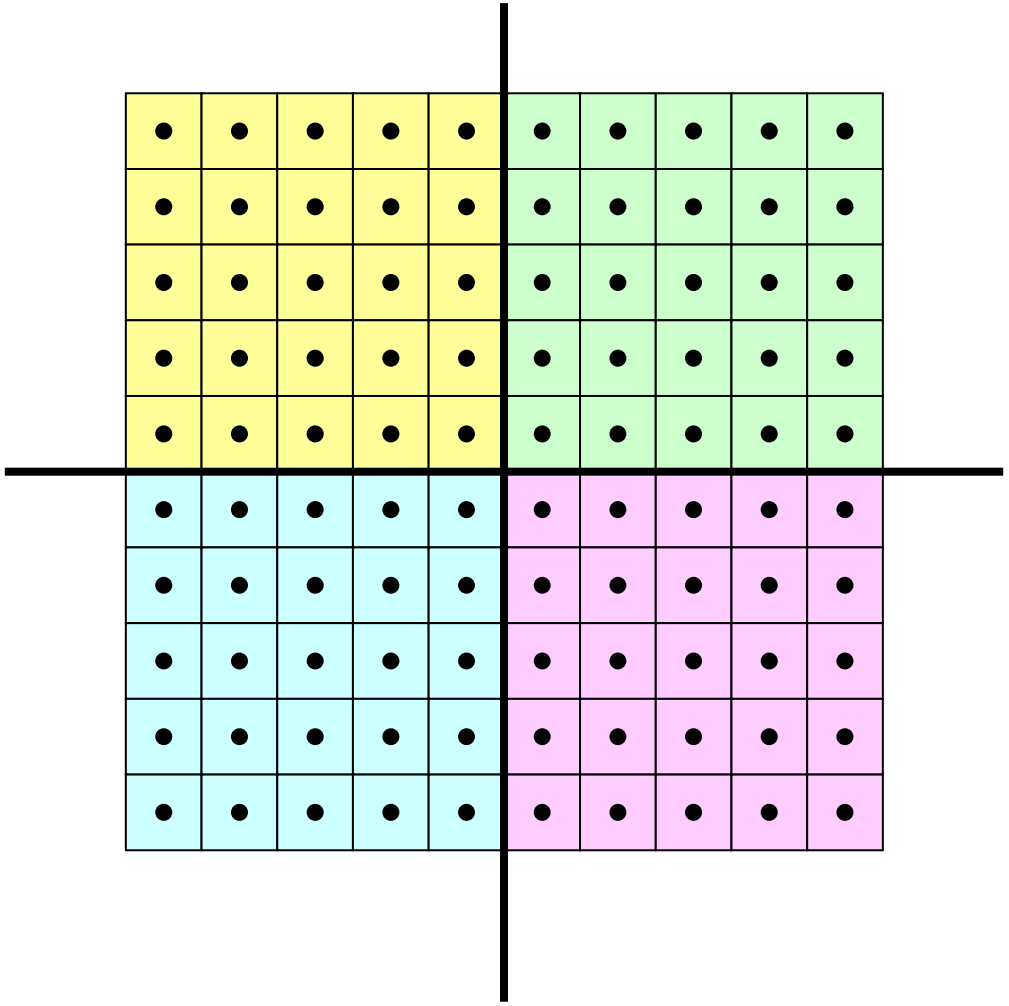
$$\frac{\partial^2 \phi}{\partial x^2} + \frac{\partial^2 \phi}{\partial y^2} = f$$

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



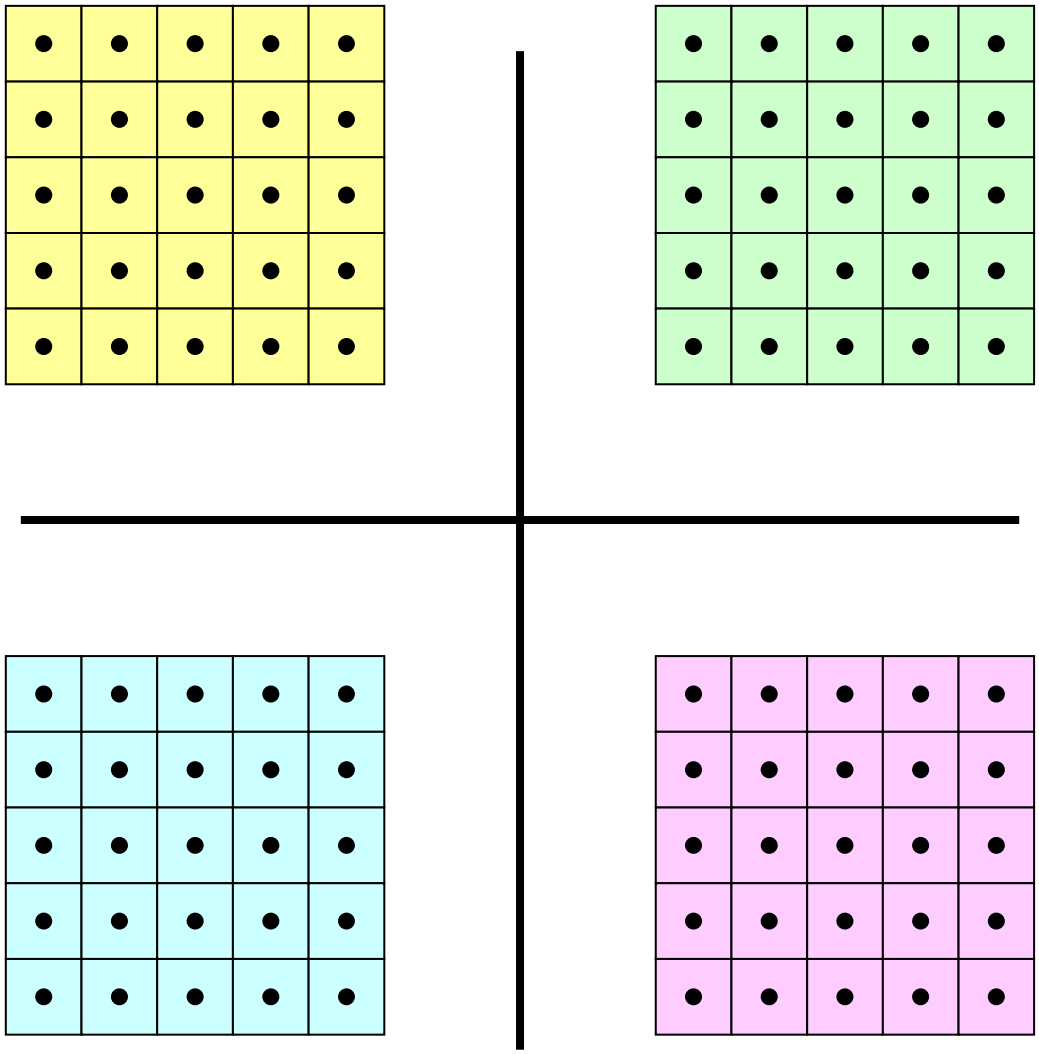
二次元差分法(2/5)

4領域に分割



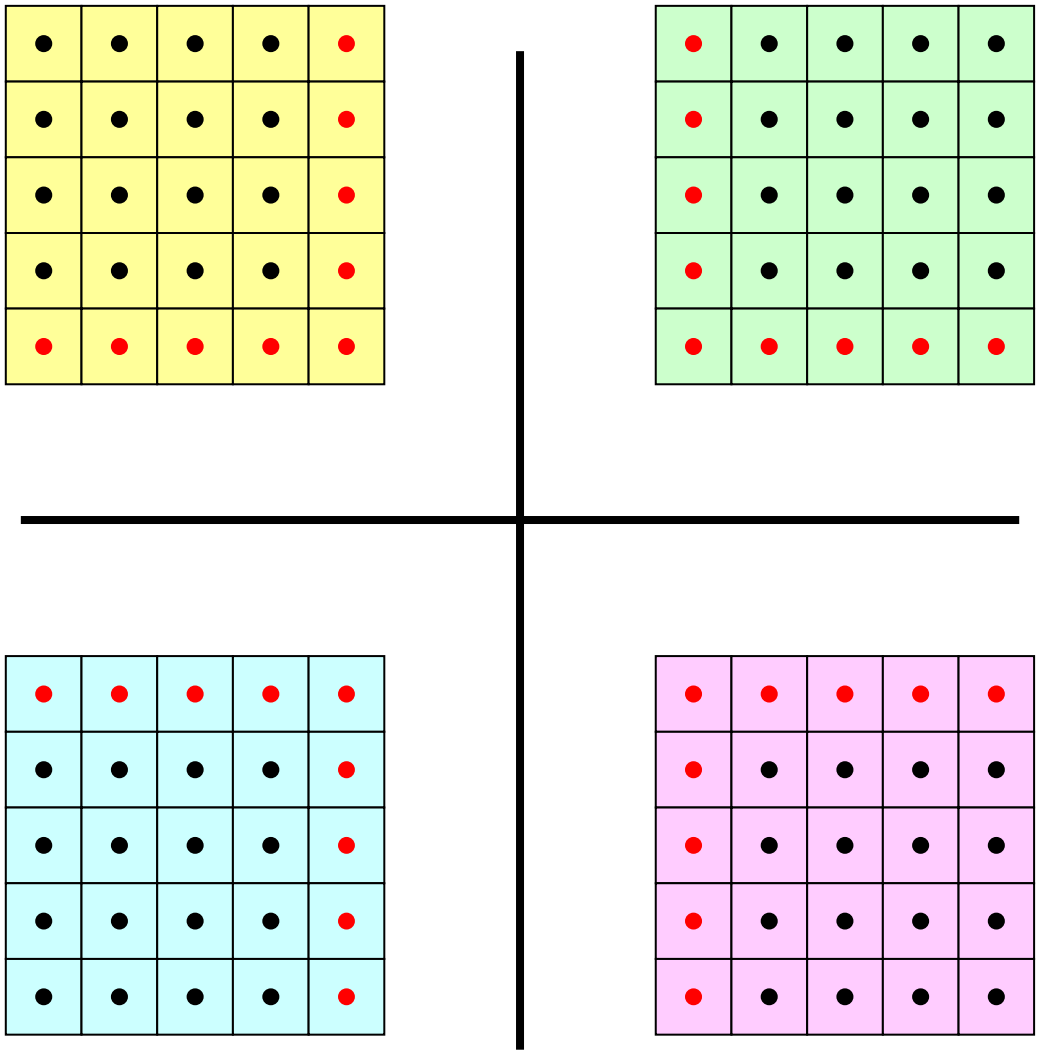
二次元差分法(3/5)

4領域に分割



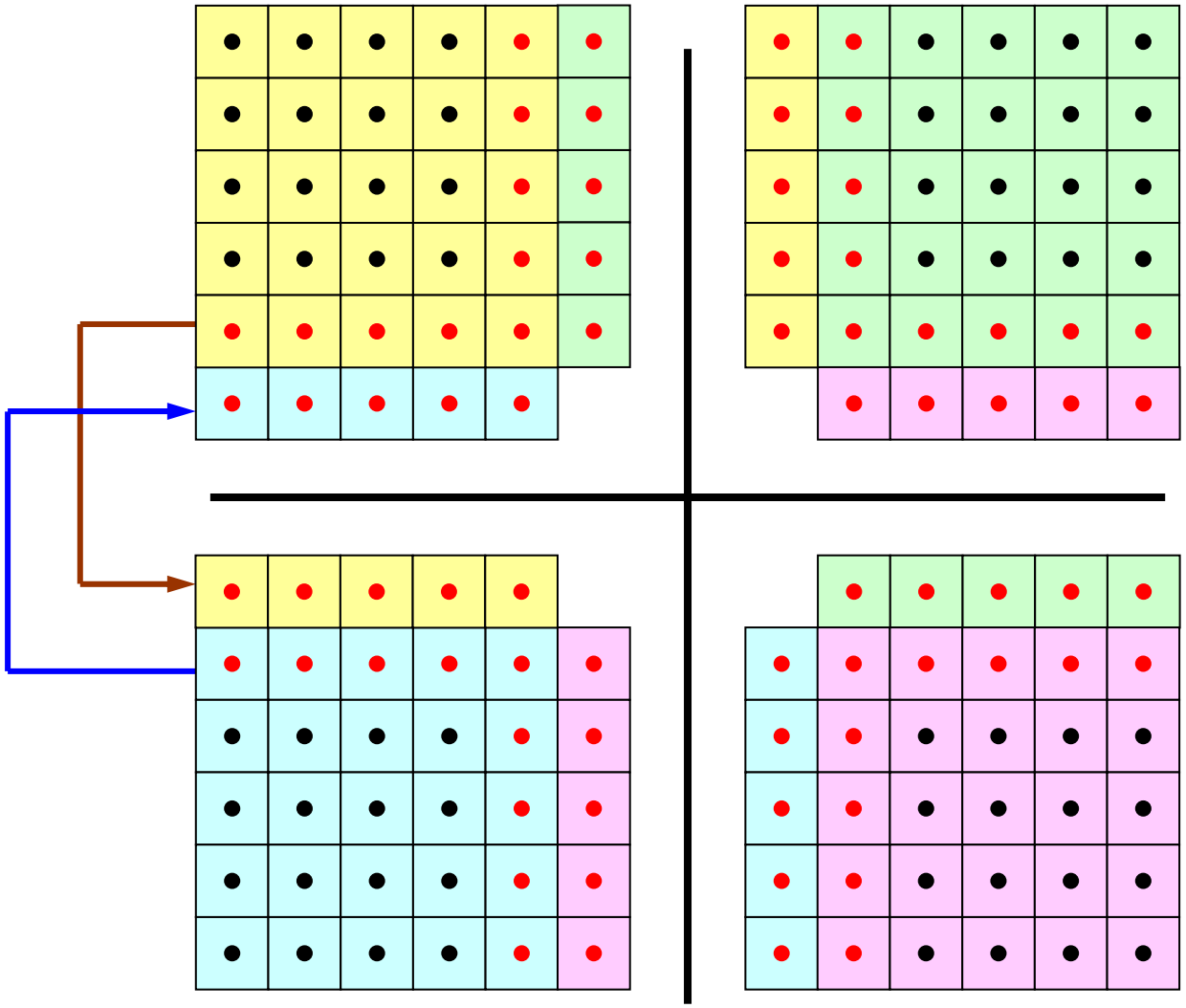
二次元差分法(4/5)

計算時には隣接領域の値が必要:オーバーラップ領域



二次元差分法(5/5)

計算時には隣接領域の値が必要:オーバーラップ領域



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（ϕのようなもの）とする

問題設定：局所データ

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

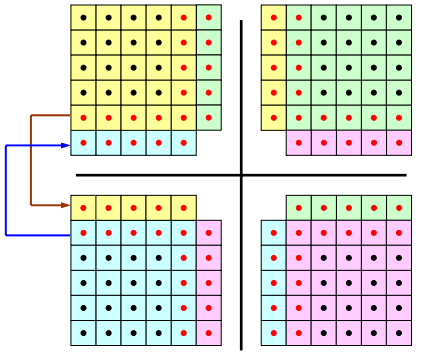
25	26	27	28
17	18	19	20
9	10	11	12
1	2	3	4

PE#0

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

PE#1

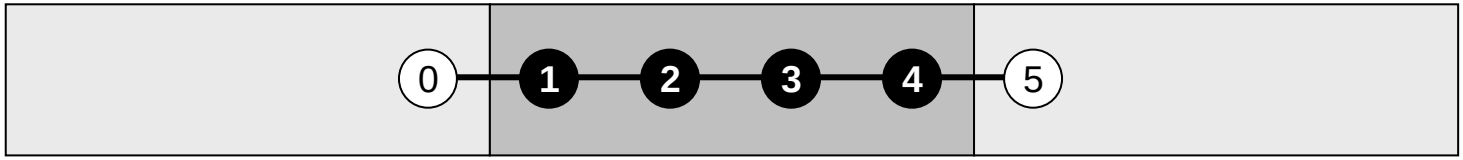
- 左記のような4領域に分割された二次元領域において，二次元中央差分（5点差分）を並列に実施するために必要な情報（全体要素番号）を隣接領域と送受信する方法。
  - □はPE#0が受信する情報。



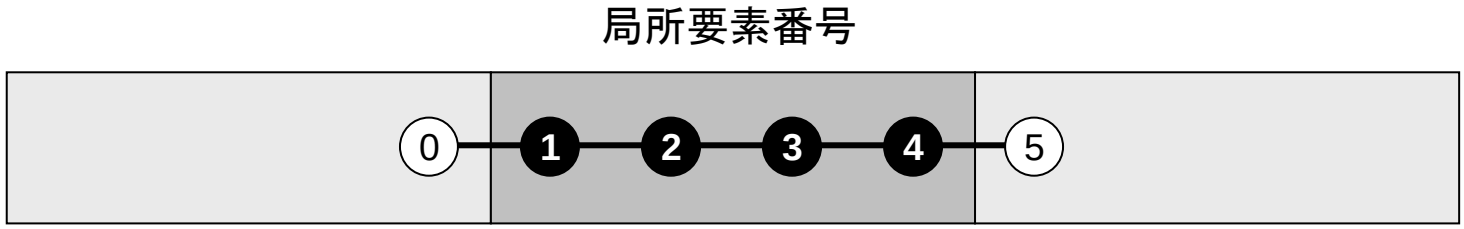
1D差分法の並列計算に必要な情報(1/3)
隣接プロセッサ情報

PE#(my_rank-1): NEIBPE(1)

PE#(my_rank+1): NEIBPE(2)

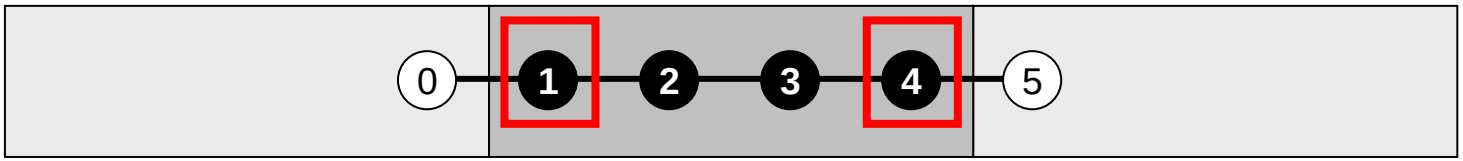


1D差分法の並列計算に必要な情報(2/3)
内点・外点



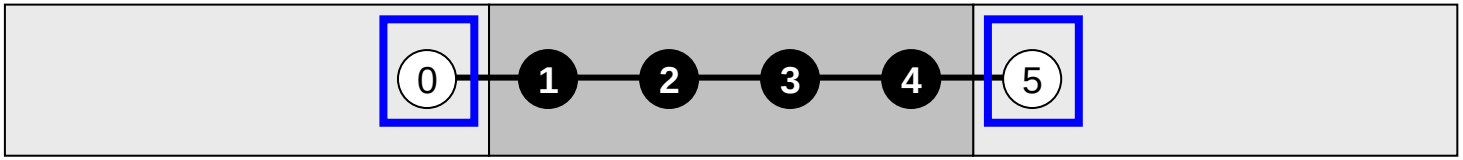
1D差分法の並列計算に必要な情報(3/3) 通信テーブル

隣接PEへの送信(境界点)

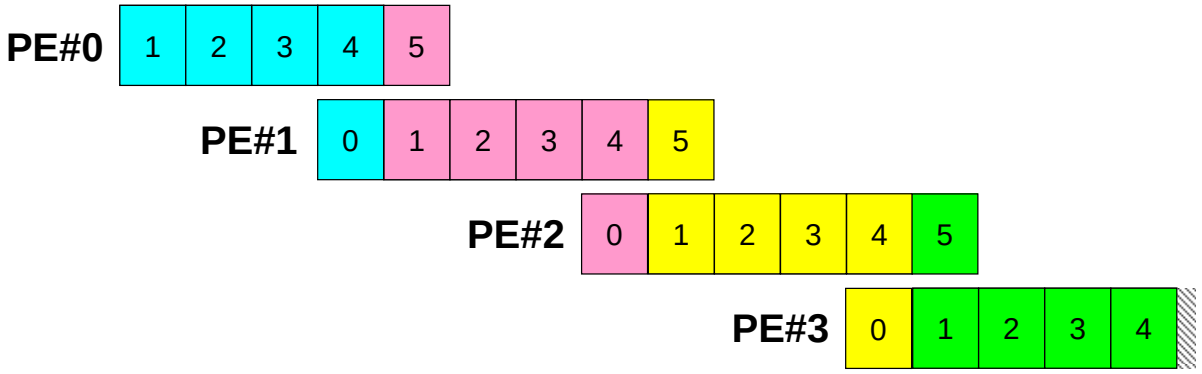



```
SENDbuf(1)=BUF(1)
SENDbuf(2)=BUF(4)
```

隣接PEからの受信(外点)

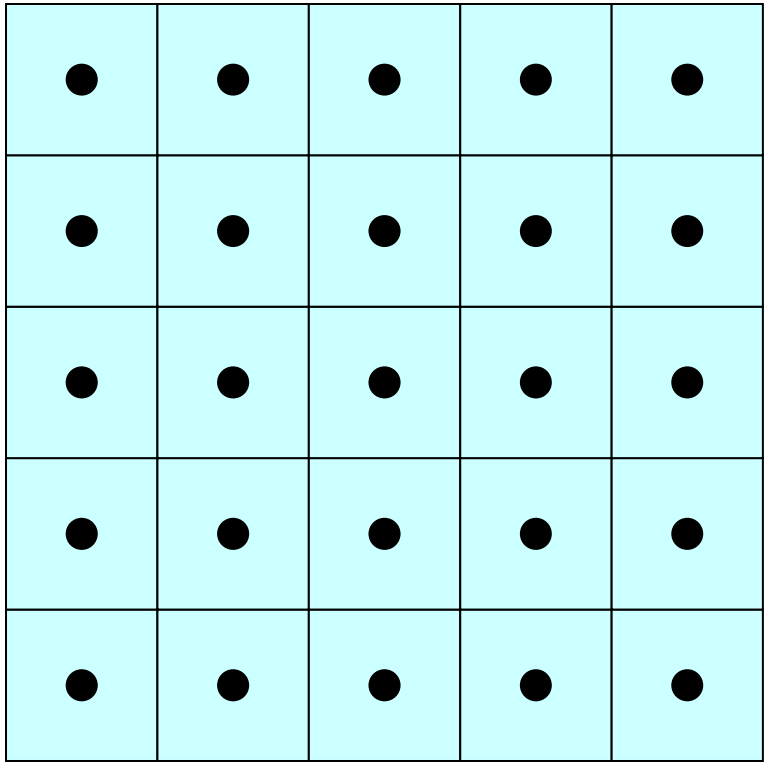



```
BUF(0)=RECVbuf(1)
BUF(5)=RECVbuf(2)
```

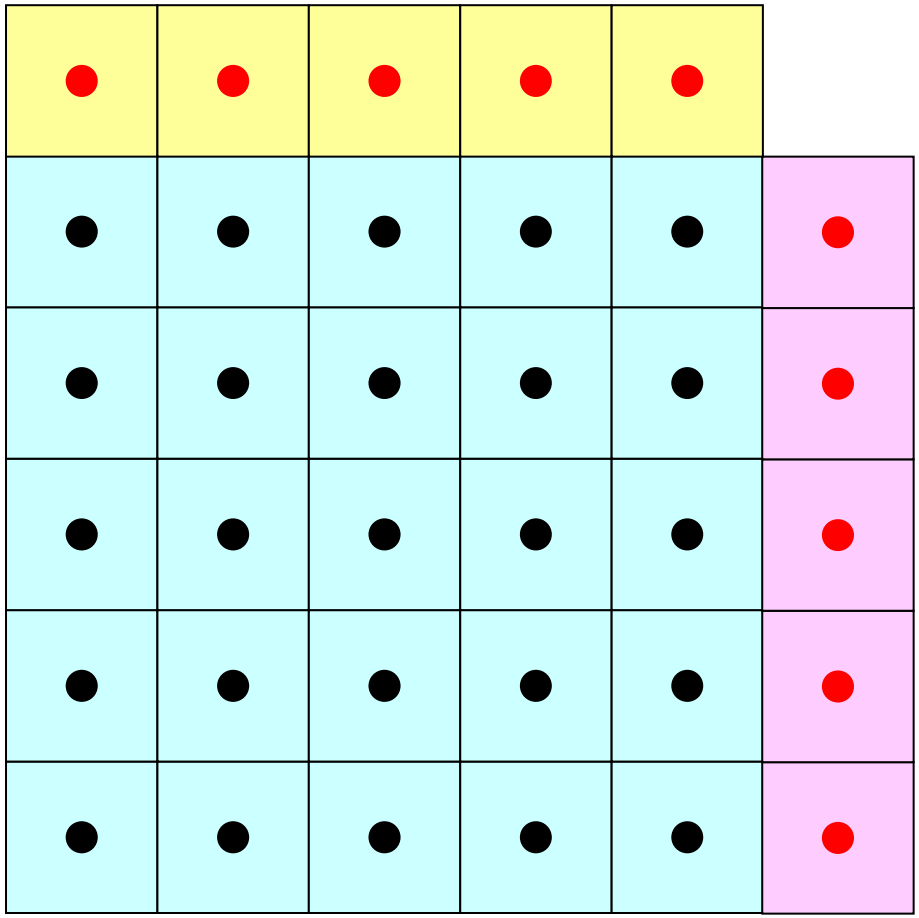


二次元差分法
各領域に必要な情報(1/4)

内点(Internal Points)
その領域にアサインされた要素



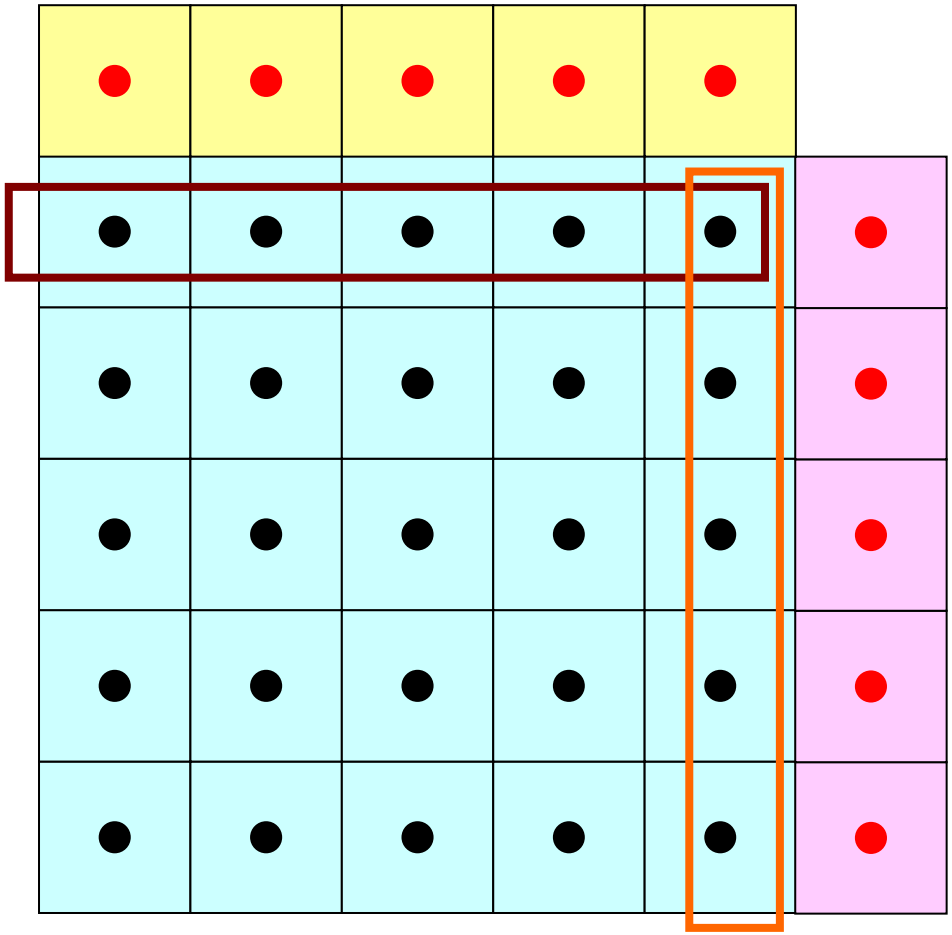
二次元差分法
各領域に必要な情報(2/4)



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)

二次元差分法
各領域に必要な情報(4/4)

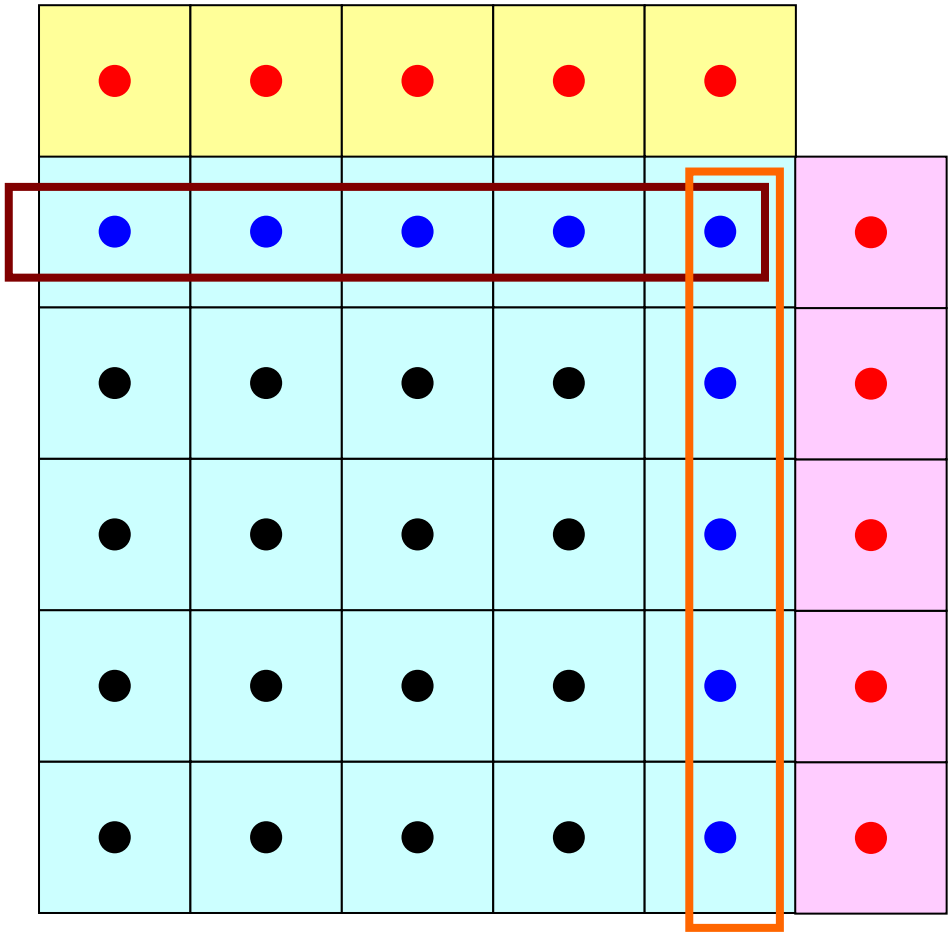


内点(Internal Points)
その領域にアサインされた要素

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

二次元差分法
各領域に必要な情報(4/4)



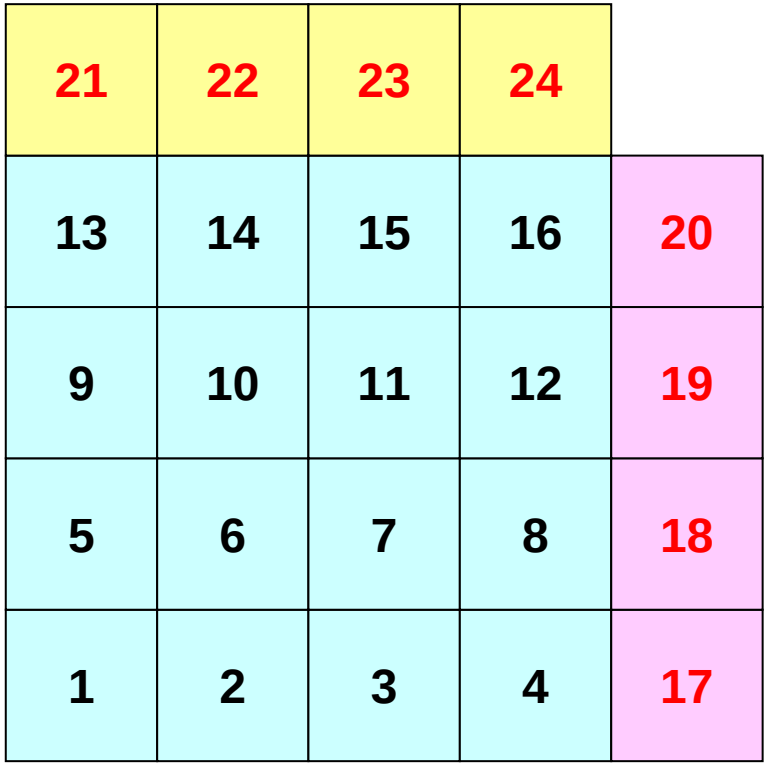
内点(Internal Points)
その領域にアサインされた要素

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

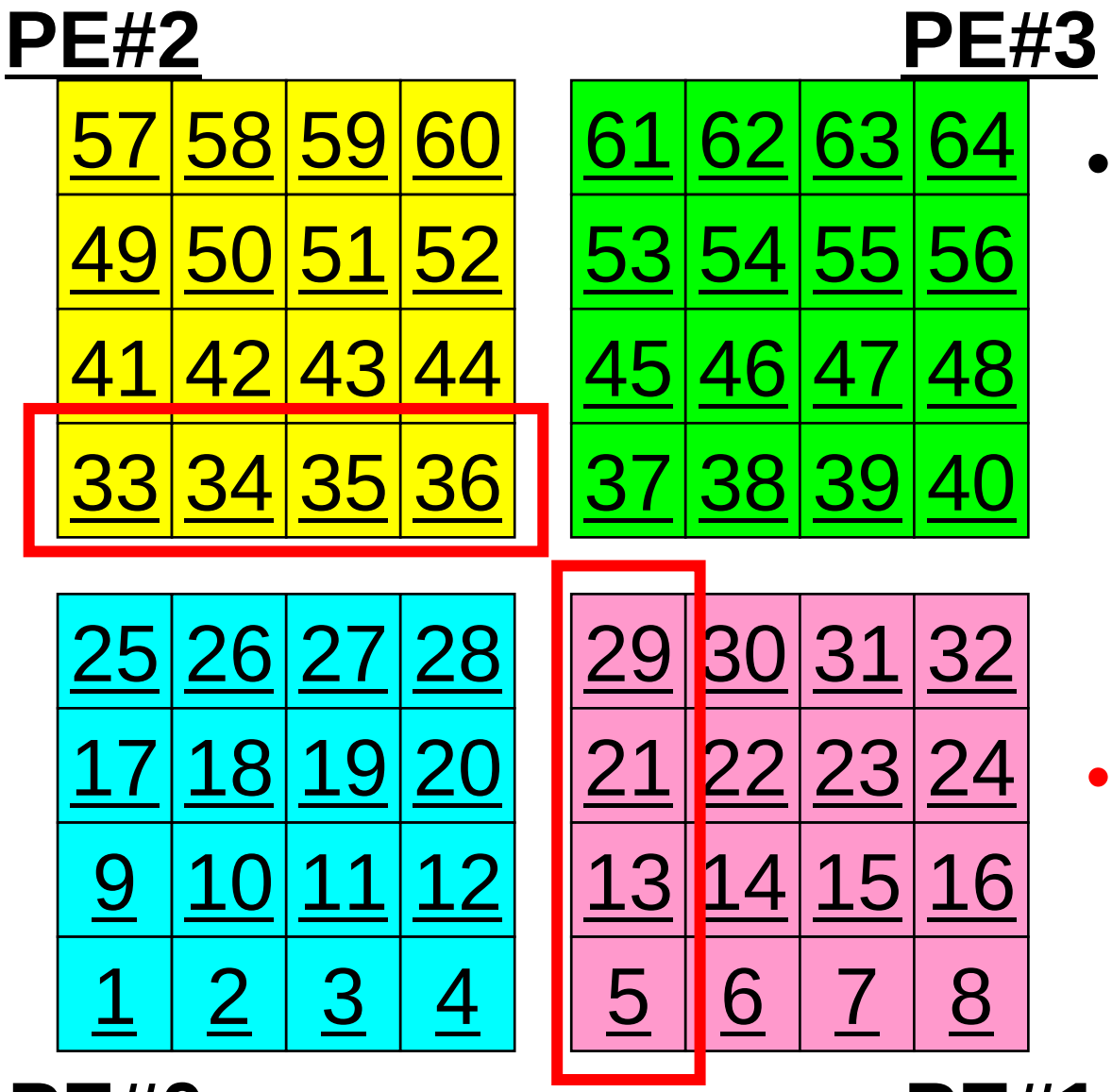
領域間相互の関係
通信テーブル:外点，境界点の関係
隣接領域

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

問題設定・・・もう少し簡単に述べると



- 左記のような4領域に分割された二次元領域において，二次元中央差分(5点差分)を並列に実施するために必要な情報(全体要素番号)を隣接領域と送受信する方法。
- 各領域で内点の全体要素番号が既知であるとき，その情報を隣接領域に送信するためのデータ構造

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: ?

PE#0

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#1

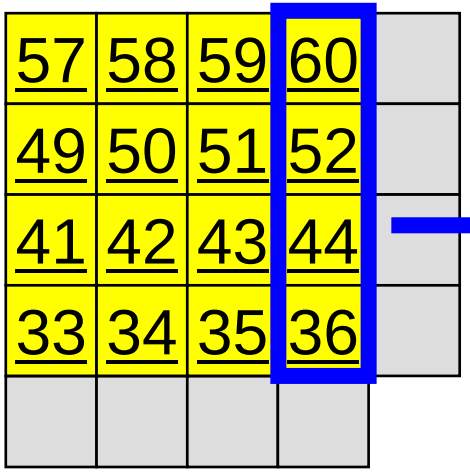
	29	30	31	32
	21	22	23	24
	13	14	15	16
	5	6	7	8

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: ?

演算内容(2/3):送信，受信前

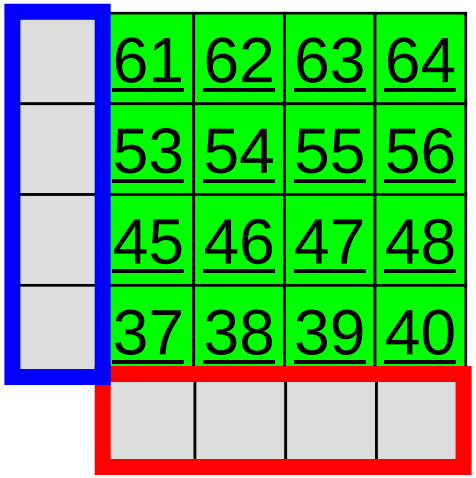
PE#2

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: ?

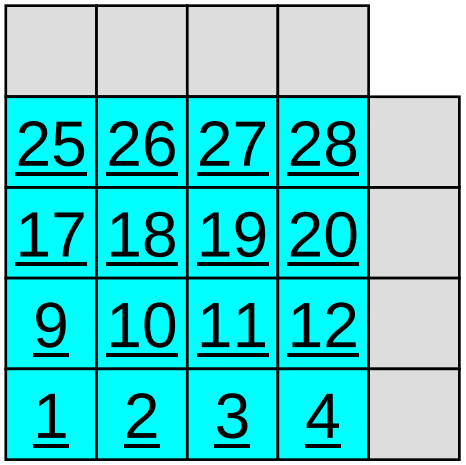


PE#3

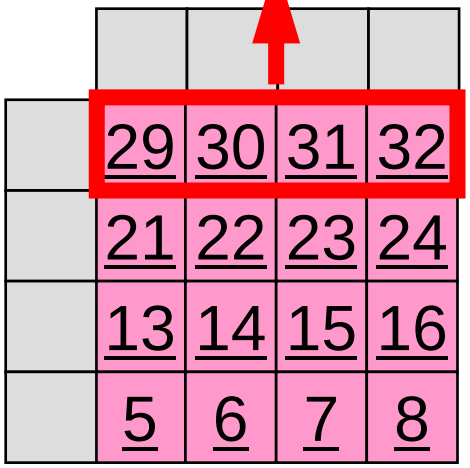
1: 37	9: 53	17: ?
2: 38	10: 54	18: ?
3: 39	11: 55	19: ?
4: 40	12: 56	20: ?
5: 45	13: 61	21: ?
6: 46	14: 62	22: ?
7: 47	15: 63	23: ?
8: 48	16: 64	24: ?



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(3/3):送信，受信後

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	

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#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

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

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

	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

サンプルプログラム：二次元

FORTRAN

```
$ cd <$S2>
$ mpif90 -O3 sq-sr1.f
$ mpirun -np 4 a.out
```

ISEND/IRECV

```
$ mpif90 -O3 sq-sr2.f
$ mpirun -np 4 a.out
```

SENDRECV

C

```
$ cd <$S2C>
$ mpicc -O3 sq-sr1.c
$ mpirun -np 4 a.out
```

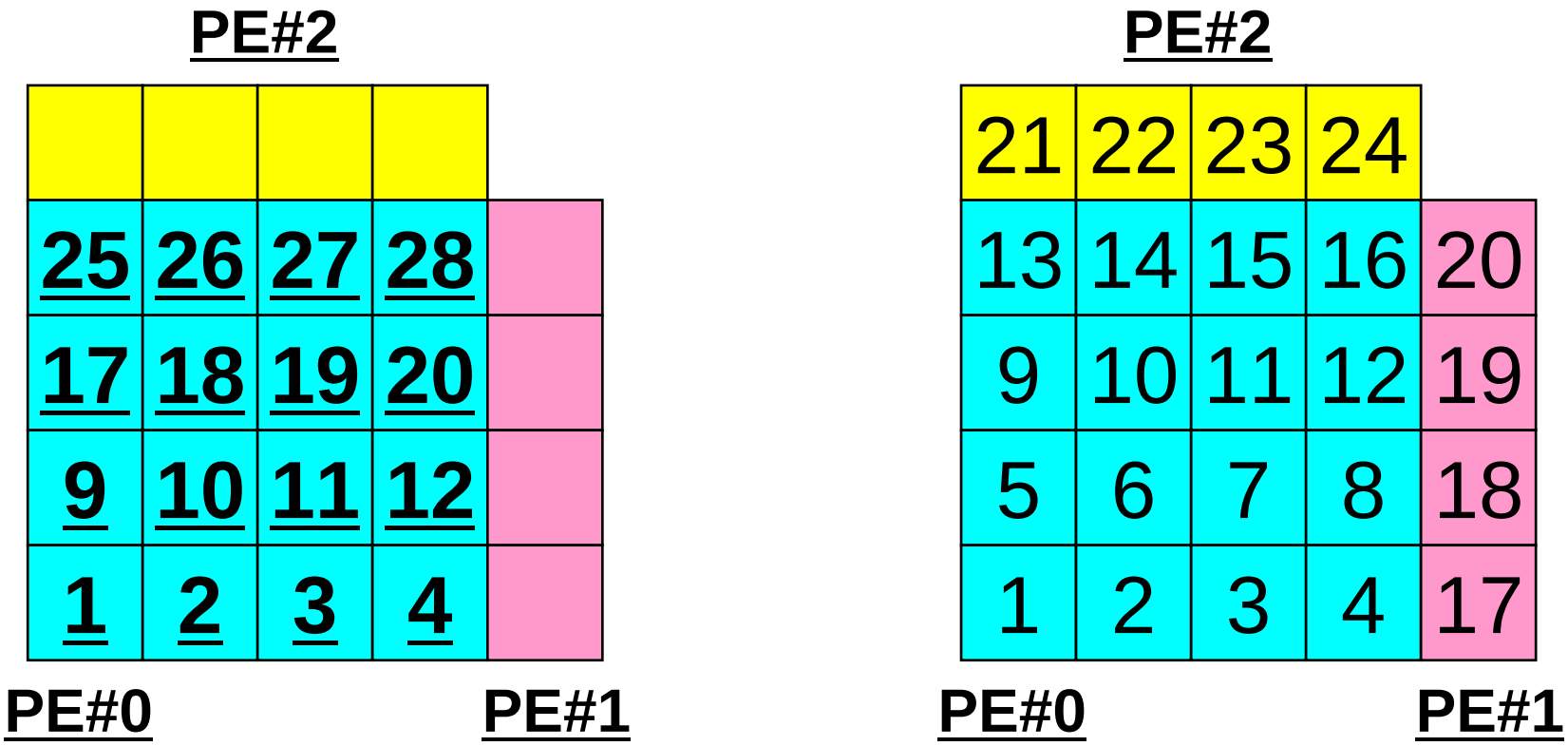
ISEND/IRECV

```
$ mpicc -O3 sq-sr2.c
$ mpirun -np 4 a.out
```

SENDRECV

各領域データ(局所データ)仕様

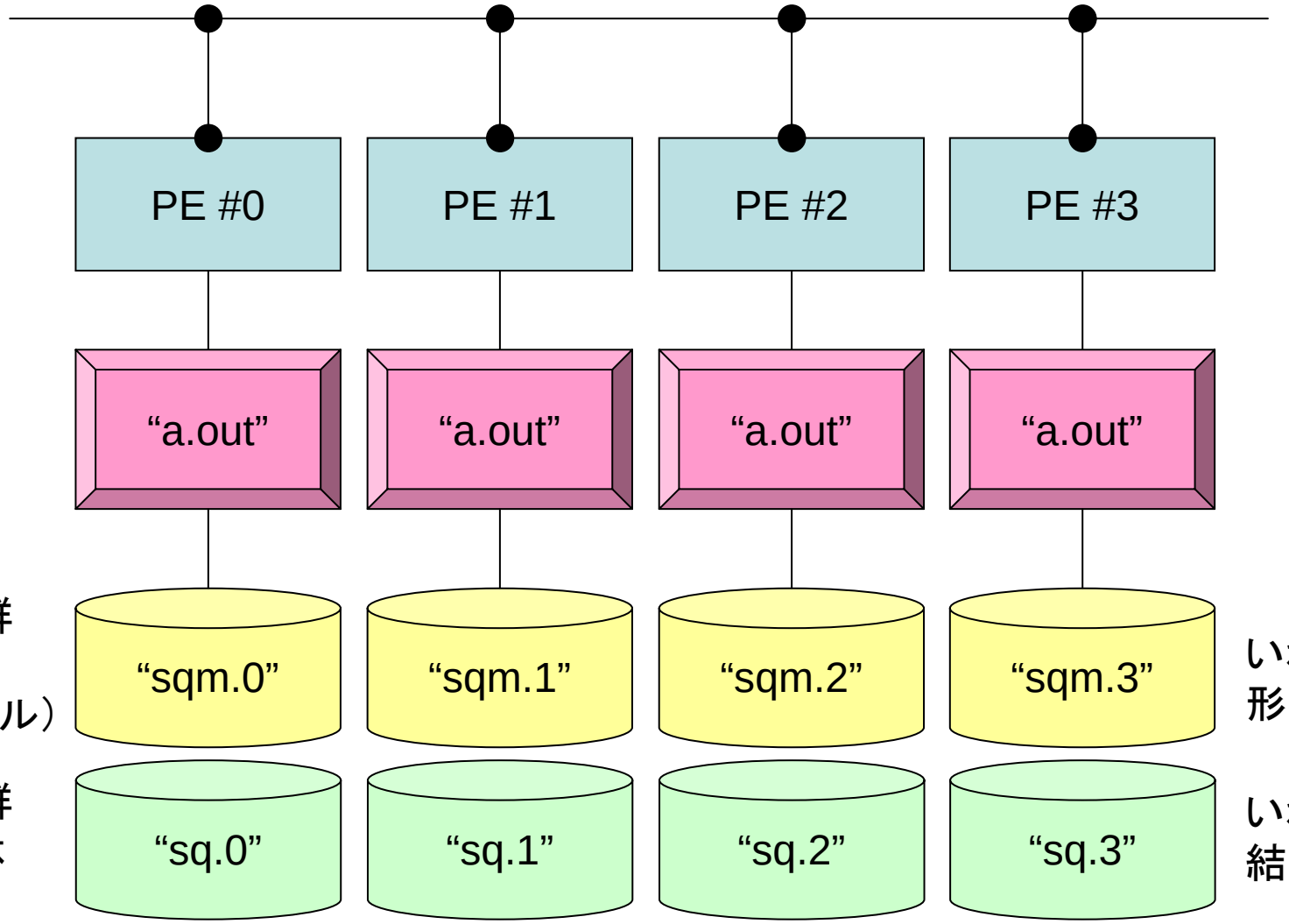
PE#0における局所データ



各要素における値(全体番号)

局所番号

SPMD・・・

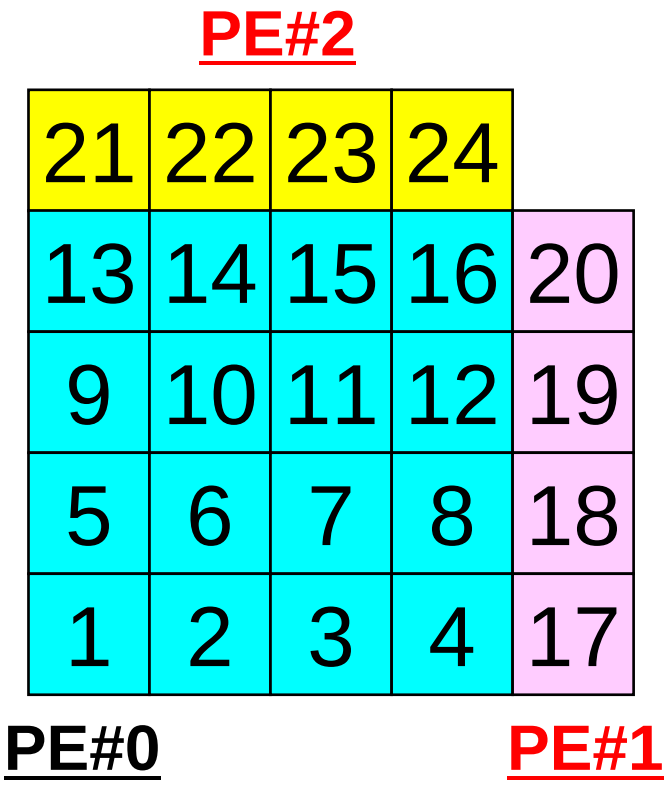

PE #0
PE #1
PE #2
PE #3
“a.out”
“a.out”
“a.out”
“a.out”
局所データ群
（隣接領域，
通信テーブル）
“sqm.0”
“sqm.1”
“sqm.2”
“sqm.3”
いわゆる
形状データ
局所データ群
（内点の全体
要素番号）
“sq.0”
“sq.1”
“sq.2”
“sq.3”
いわゆる
結果データ

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)

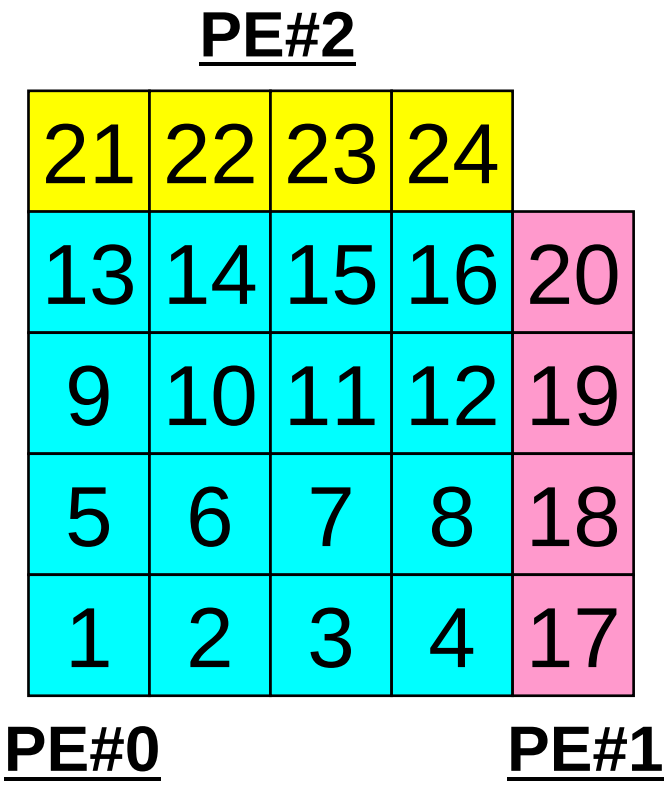
sqm.0:隣接領域数，隣接領域




```
#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



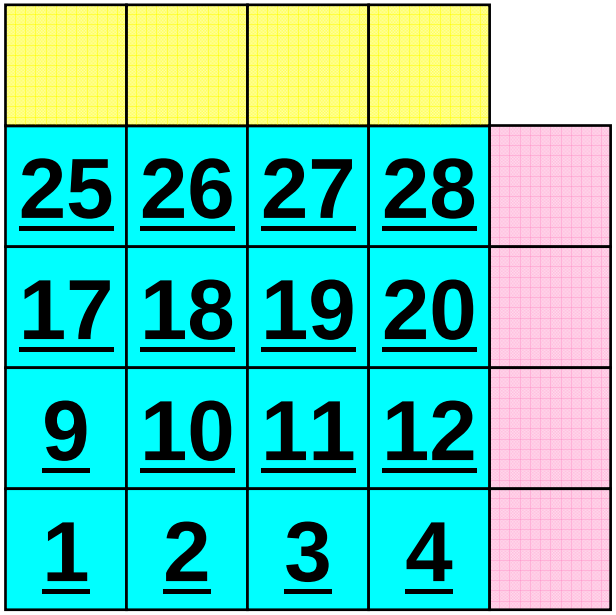

```
#NEIBPEtot
2
#NEIBPE
1   2
#NODE
24  16
#IMPORTindex
4   8
#IMPORTitems
17
18
19
20
21
22
23
24
#EXPORTindex
4   8
#EXPORTitems
4
8
12
16
13
14
15
16
```

総要素数，内点

PE#0における局所データ(4/8)

sq.0:内点における値:ここだけ違うファイル

PE#2



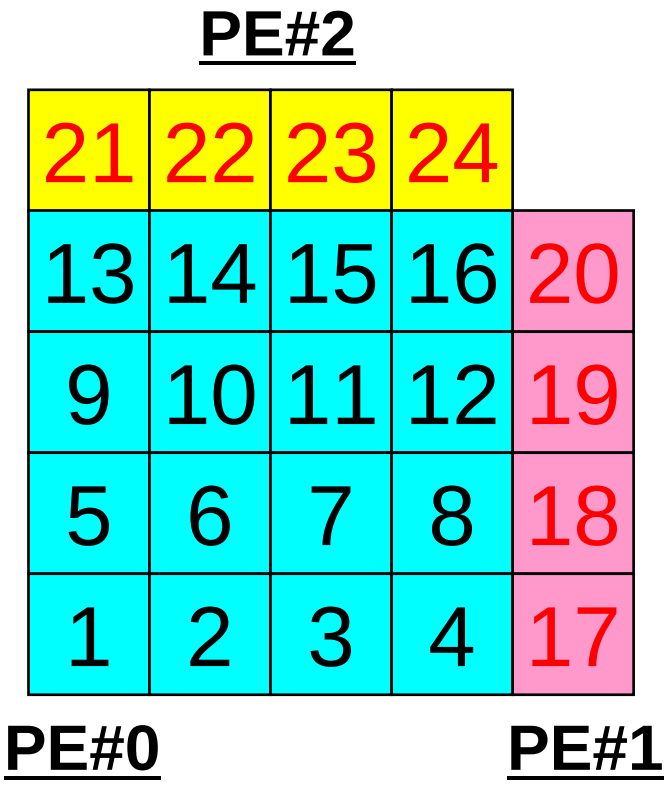
PE#0 PE#1

```
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

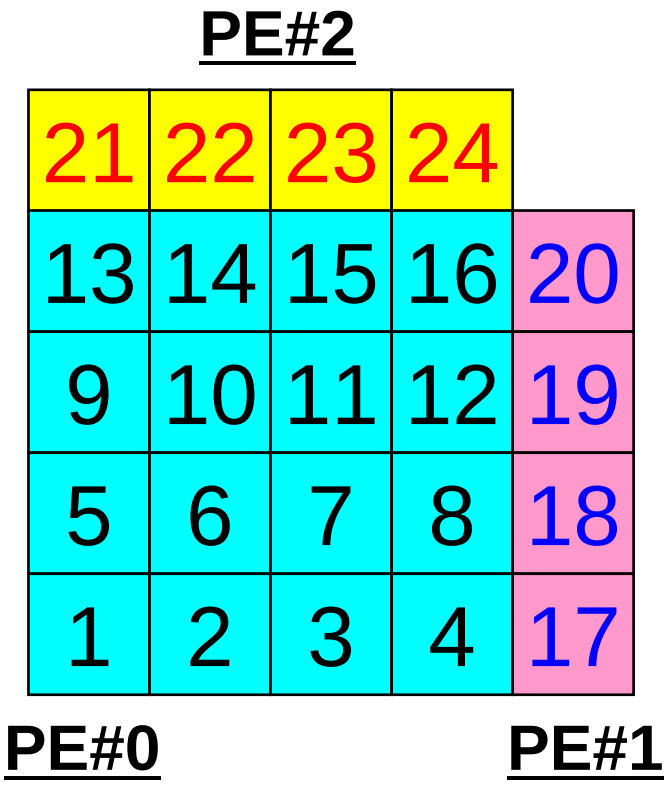



```
#NEIBPEtot
2
#NEIBPE
1   2
#NODE
24  16
#IMPORTindex
4   8
#IMPORTitems
17
18
19
20
21
22
23
24
#EXPORTindex
4   8
#EXPORTitems
4
8
12
16
13
14
15
16
```

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)

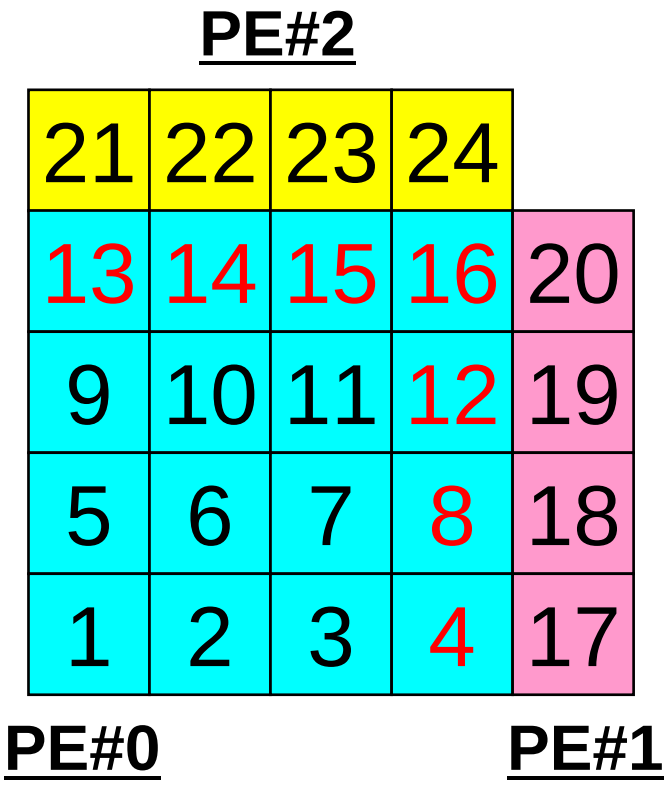
sqm.0:「import(受信)」される「外点」の情報




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

PE#0における局所データ(7/8)

sqm.0:「export(送信)」する「境界点」の情報

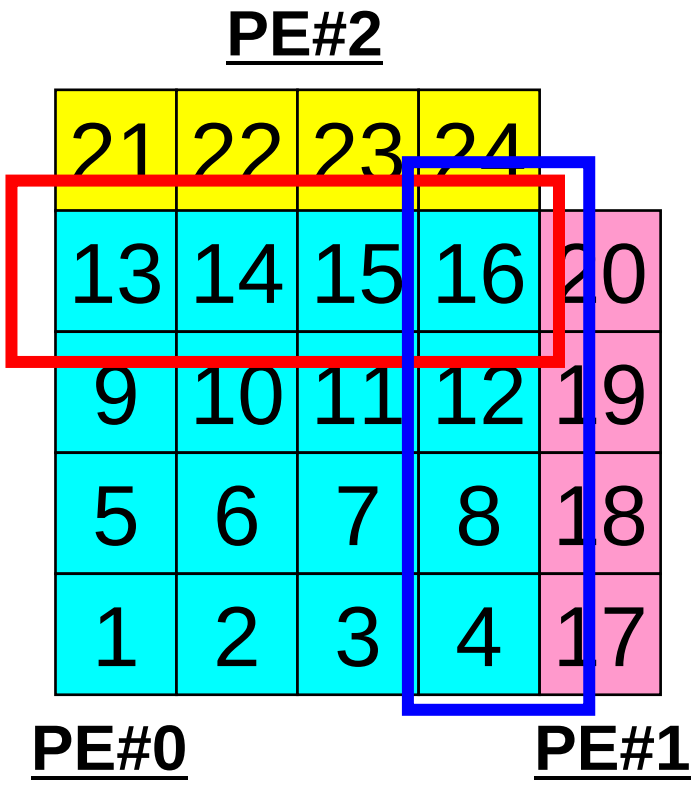



```
#NEIBPEtot
2
#NEIBPE
1   2
#NODE
24  16
#IMPORTindex
4   8
#IMPORTitems
17
18
19
20
21
22
23
24
#EXPORTindex
4   8
#EXPORTitems
4
8
12
16
13
14
15
16
```

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における局所データ(8/8)

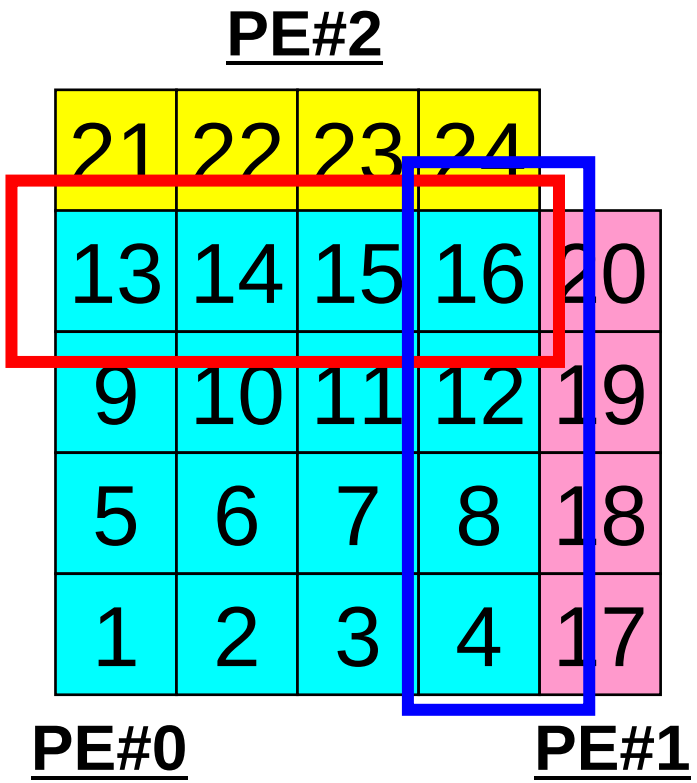
sqm.0:「export(送信)」する「境界点」の情報




```
#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                  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)
                    llocate (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)
```

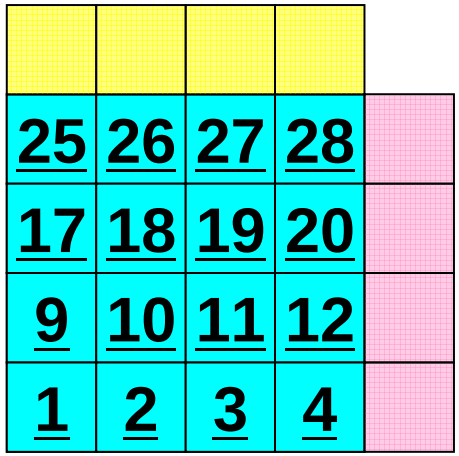
```
#NEIBPEtot
2
#NEIBPE
1  2
#NODE
24 16
#IMPORTindex
4  8
#IMPORTitems
17
18
19
20
21
22
23
24
#EXPORTindex
4  8
#EXPORTitems
4
8
12
16
13
14
15
16
```

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

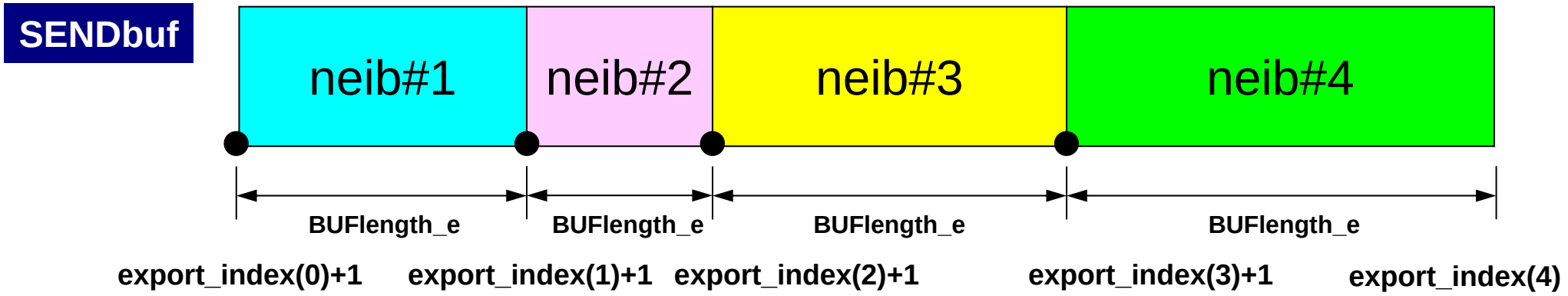
PE#2



PE#0　　PE#1

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

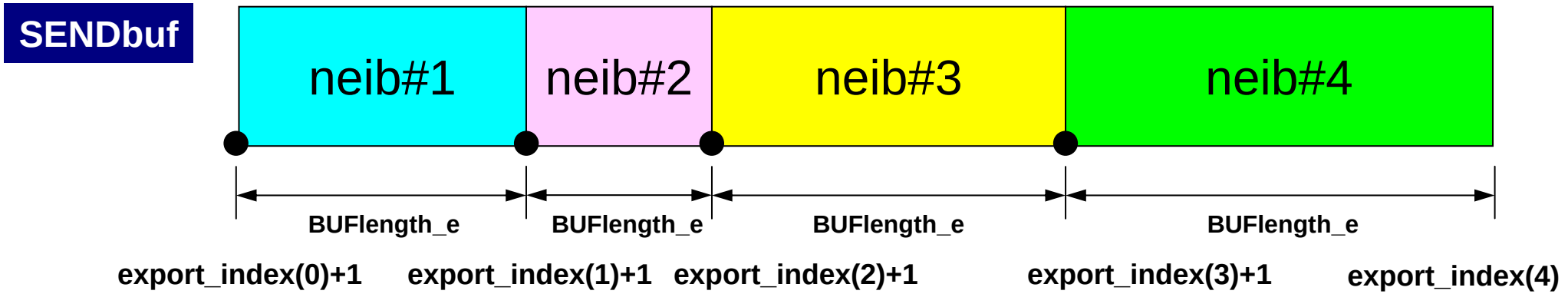
送信（MPI_Sendrecv）




```
 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

送信（MPI_Isend/Irecv/Waitall）




```
 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-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。

プログラム例：sq-sr2.f（5/6）

送・受信実施（MPI_SENDRECV）

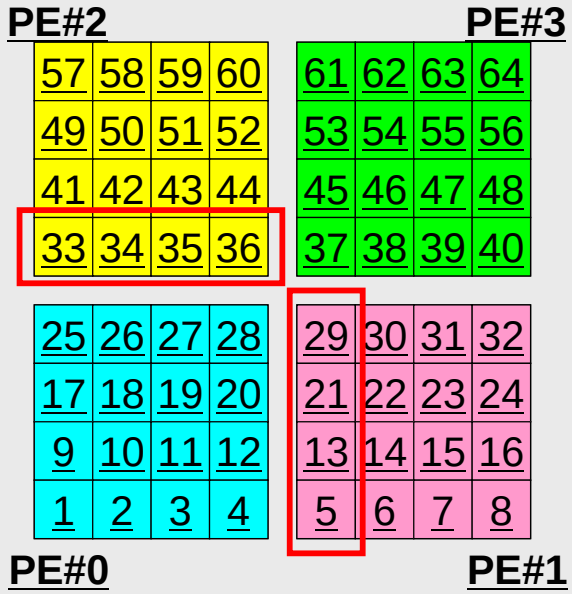
```
!C
!C +-----------+
!C | SEND-RECV |
!C +-----------+
!C===
      allocate (stat_sr(MPI_STATUS_SIZE))

      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）

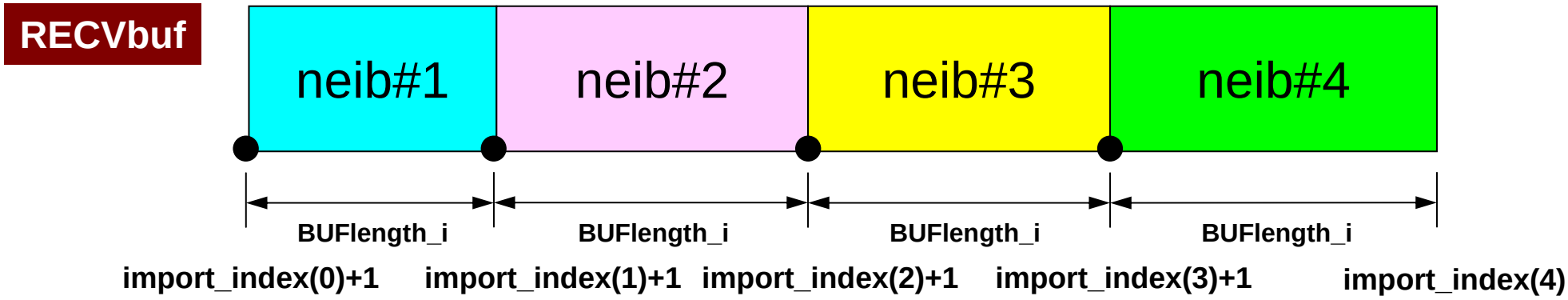
```
!C
!C +-----------+
!C | SEND-RECV |
!C +-----------+
!C===
      allocate (stat_sr(MPI_STATUS_SIZE))

      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）。

受信（MPI_Isend/Irecv/Waitall）

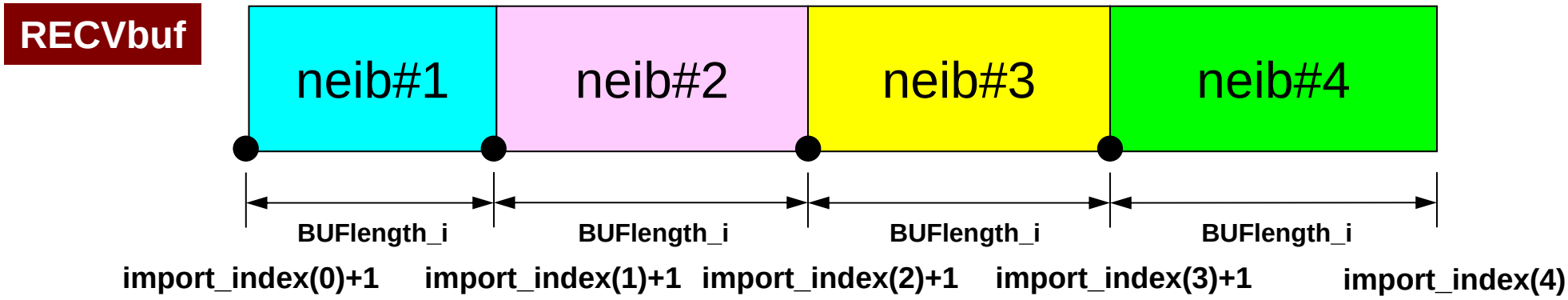
```
 do neib= 1, NEIBPETOT
   iS_i= import_index(neib-1) + 1
   iE_i= import_index(neib  )
   BUFlength_i= iE_i + 1 - iS_i

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



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



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===
```

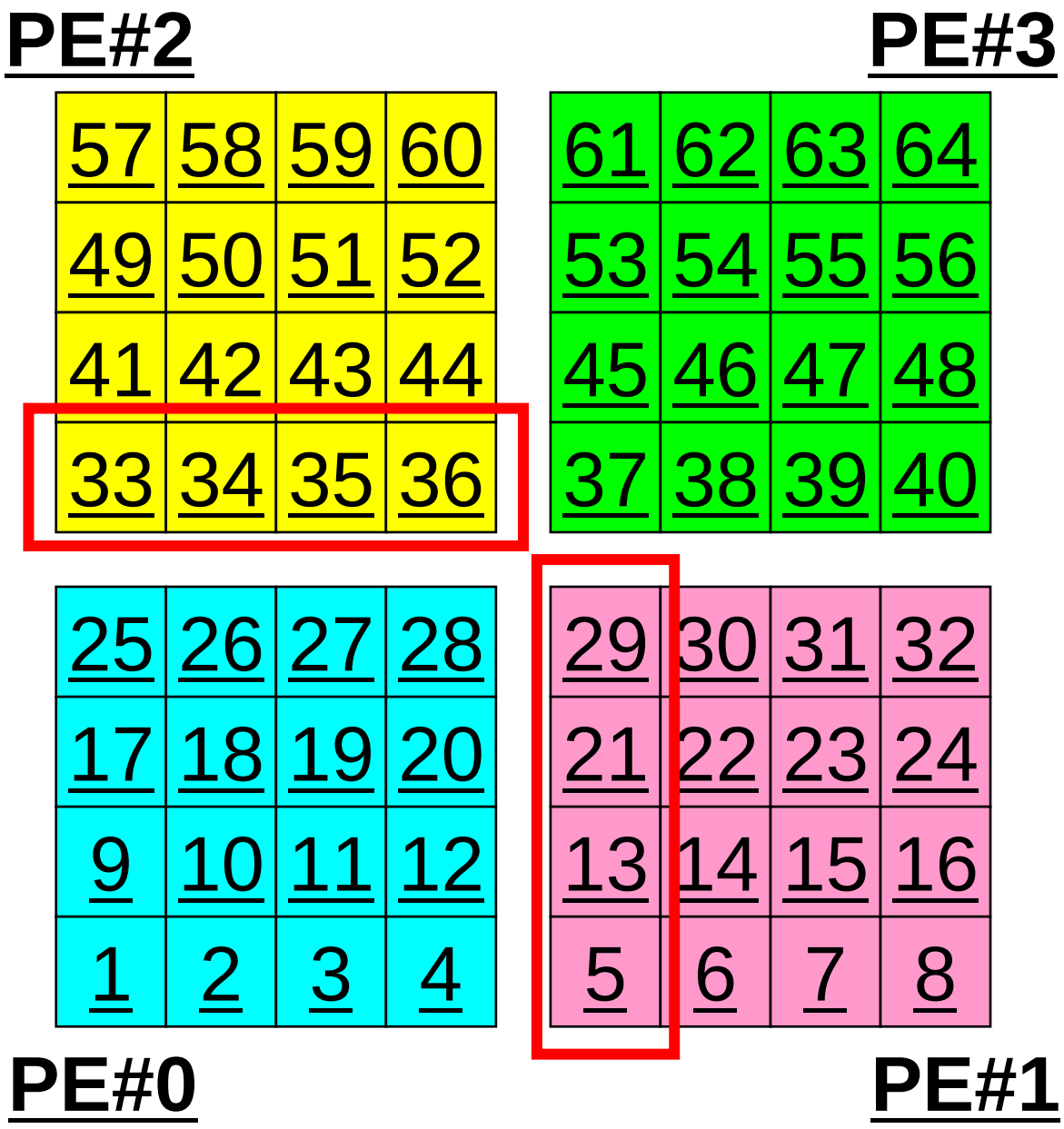
受信バッファの中身を「外点」の値として代入する。

プログラム例：sq-sr2.f（6/6）

外点の値の書き出し

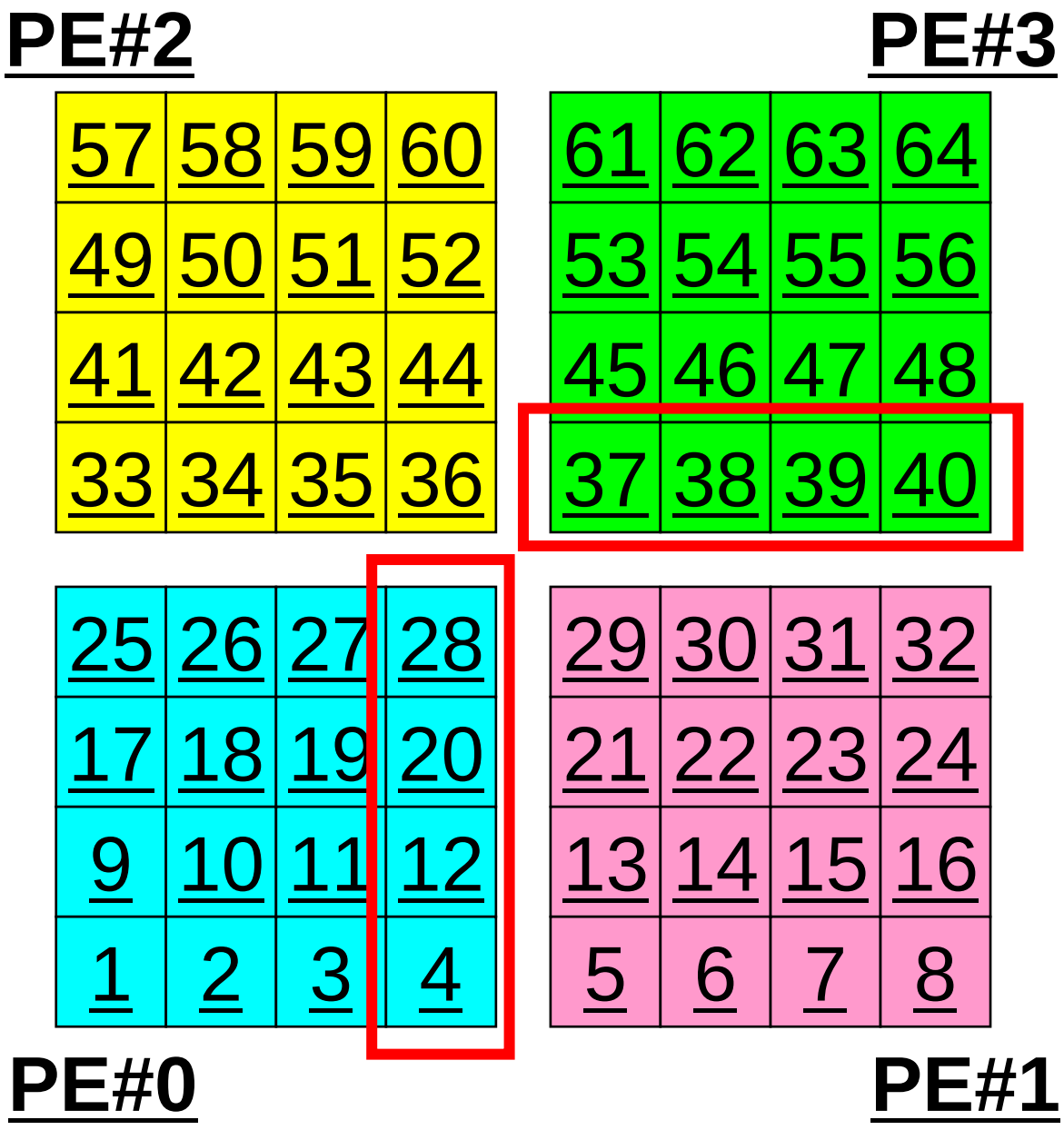
```
!C
!C +--------+
!C | OUTPUT |
!C +--------+
!C===
      do neib= 1, NEIBPETOT
        iS= import_index(neib-1) + 1
        iE= import_index(neib  )
        do i= iS, iE
          in= import_item(i)
          write (*,'(a, 3i8)') 'RECVbuf', my_rank, NEIBPE(neib), VAL(in)
        enddo
      enddo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

      stop
      end
```

実行結果（PE#0）



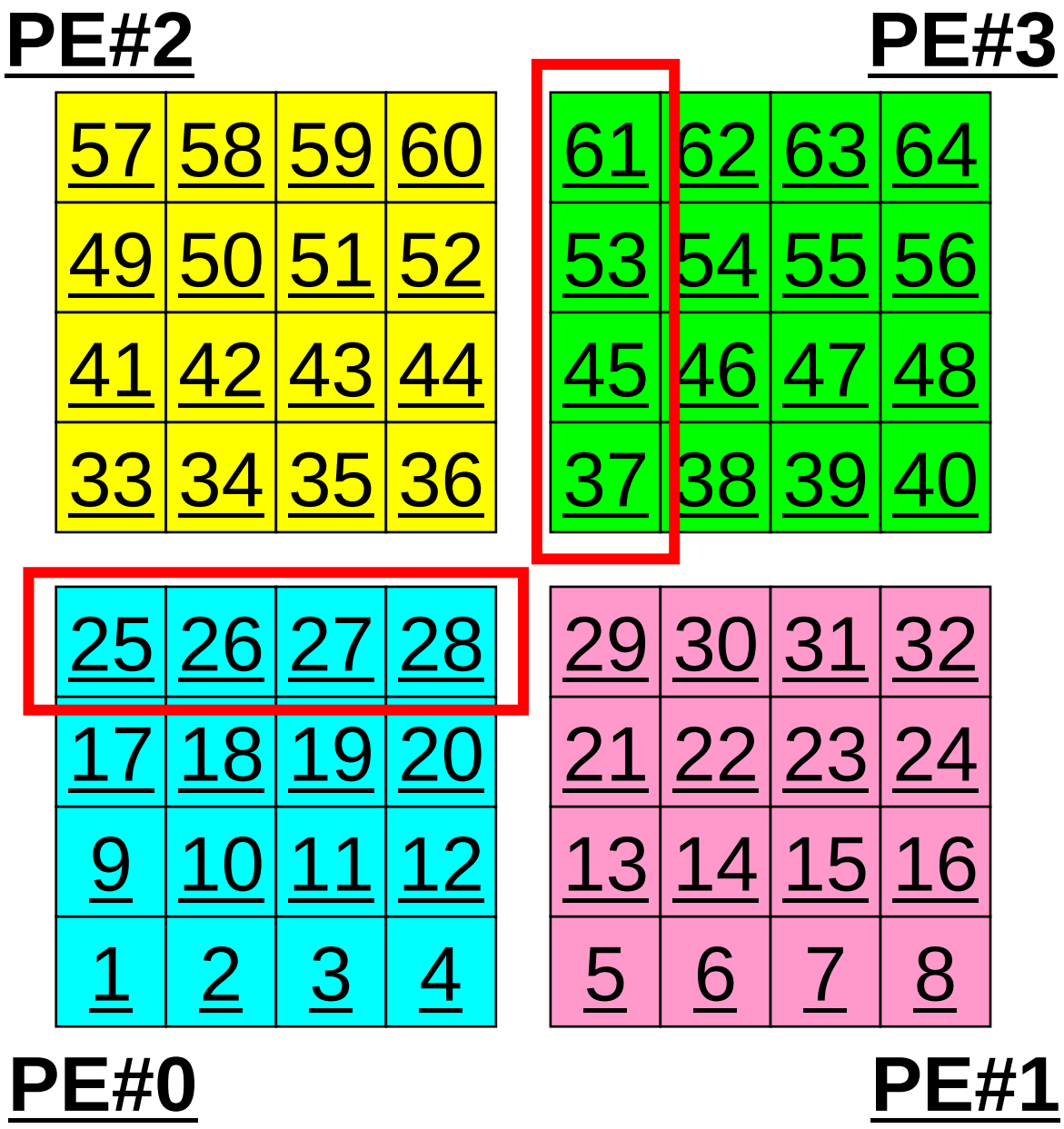

```
RECVbuf         0         1          5
RECVbuf         0         1         13
RECVbuf         0         1         21
RECVbuf         0         1         29
RECVbuf         0         2         33
RECVbuf         0         2         34
RECVbuf         0         2         35
RECVbuf         0      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



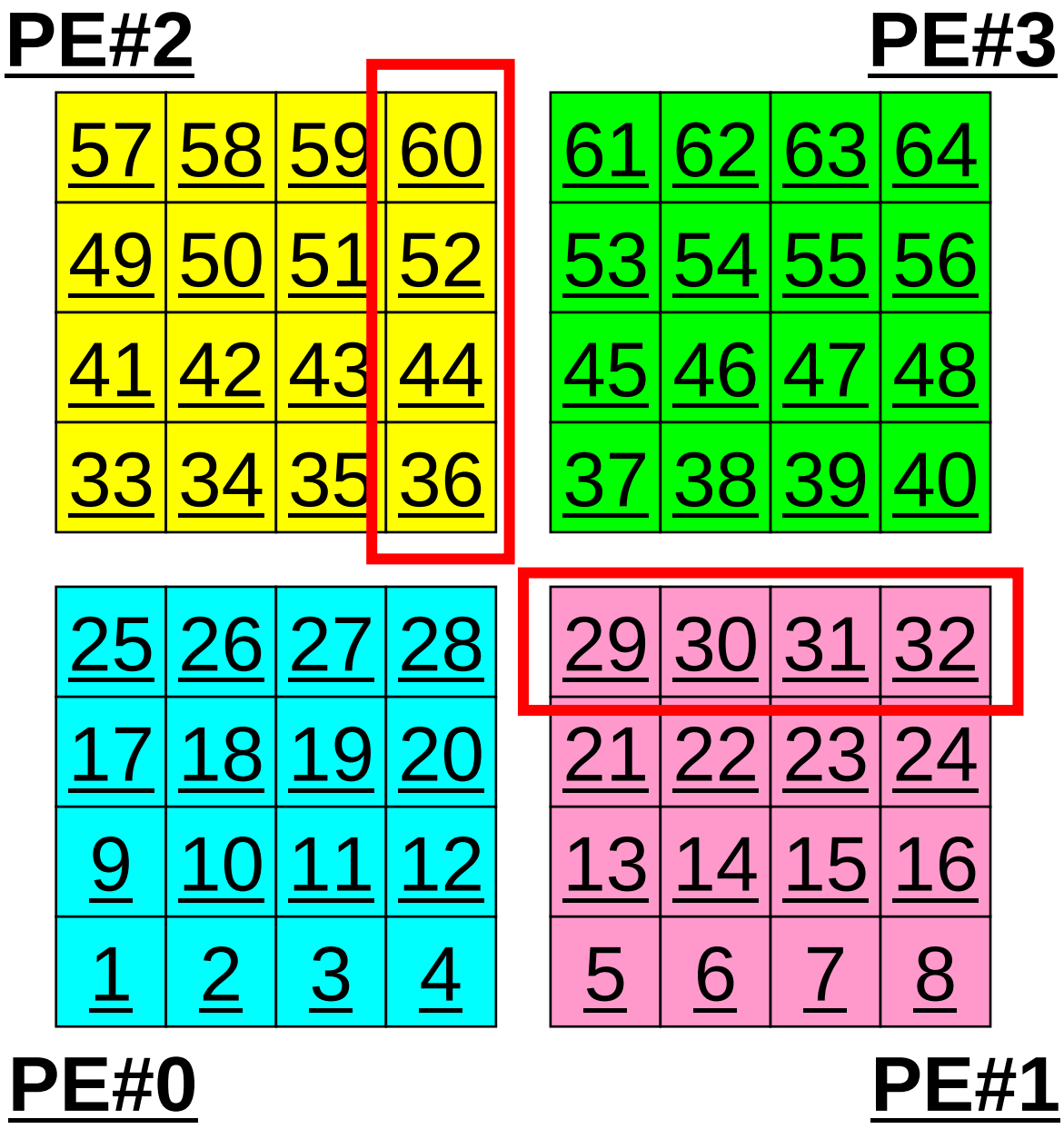

```
RECVbuf         0          1           5
RECVbuf         0          1          13
RECVbuf         0          1          21
RECVbuf         0          1          29
RECVbuf         0          2          33
RECVbuf         0          2          34
RECVbuf         0          2          35
RECVbuf  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

まとめ

- 差分法等，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方法で実施可能。
- 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
- 使っているMPIの機能は一次元の場合と全く同じ・・・である。
- 密行列の場合は，異なった手法を適用する
  - もっと簡単
  - 詳しくは7月の授業（最終回）

復習：各自でやってほしいこと

- <$S2>sq-sr1/sr2.f，<$S2>sq-sr1.c/sr2.cの内容をよく理解すること
- デバッグwriteを挿入して，各配列の中身を確認しておくように。
  - PE#2の内容のみ書き出したい場合は：

```
if (my_rank.eq.2) then
 …
endif
```

```
if(MyRank == 2){
 …
}
```

- 補足：通信テーブルの一般的な形の一次元問題への適用

サンプルプログラム: 一次元

FORTRAN

```
$ cd <$S2>
$ mpif90 -O3 1d-srb1.f
$ mpirun -np 4 a.out
```

ISEND/IRECV

```
$ mpif90 -O3 1d-srb2.f
$ mpirun -np 4 a.out
```

SENDRECV

C

```
$ cd <$S2C>
$ mpicc -O3 1d-srb1.c
$ mpirun -np 4 a.out
```

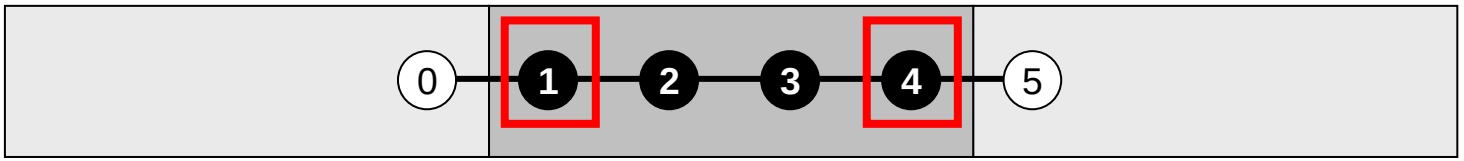
ISEND/IRECV

```
$ mpicc -O3 1d-srb2.c
$ mpirun -np 4 a.out
```

SENDRECV

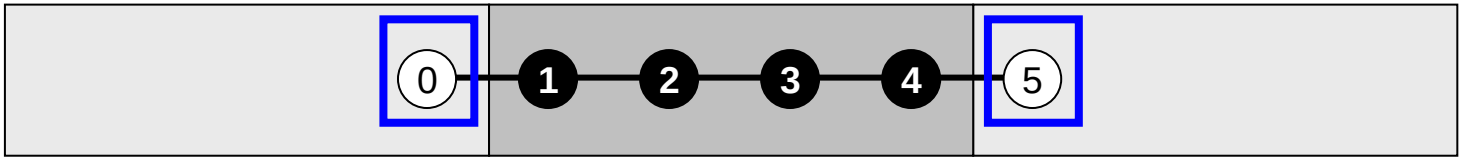
1D差分法の並列計算に必要な情報
通信テーブル

隣接PEへの
送信(境界点)

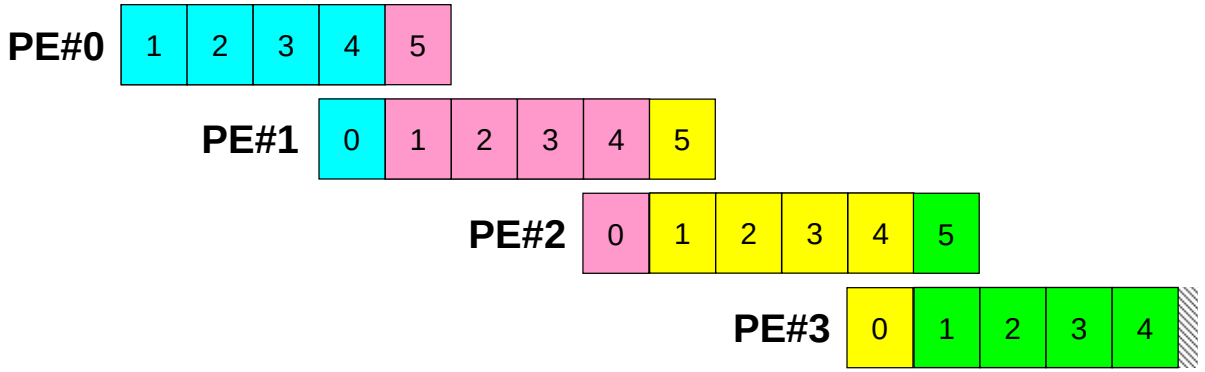



```
SENDbuf(1)=BUF(1)
SENDbuf(2)=BUF(4)
```

隣接PEからの
受信(外点)



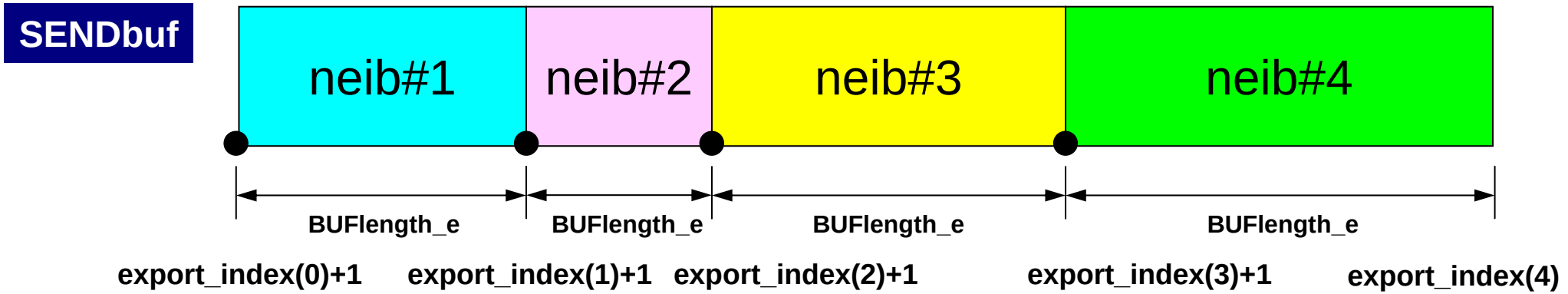

```
BUF(0)=RECVbuf(1)
BUF(5)=RECVbuf(2)
```



通信テーブルの一般的な形: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
  - 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)

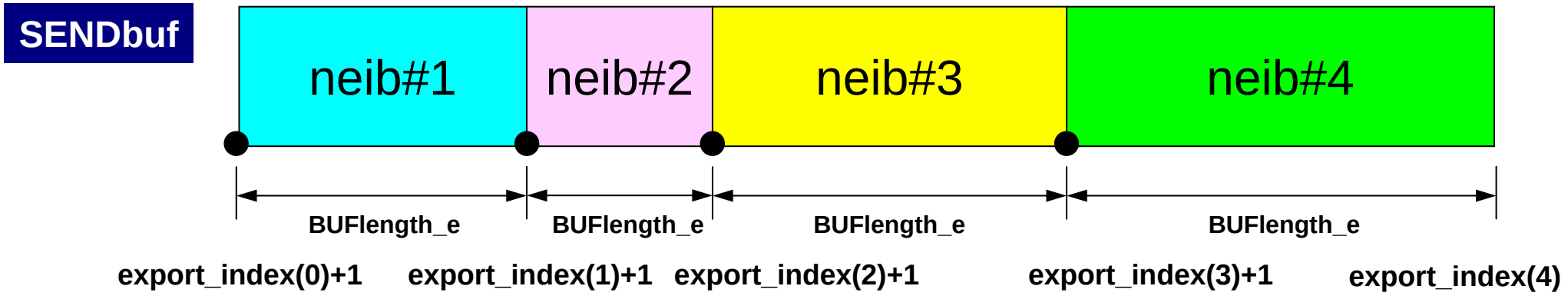
送信（MPI_Isend/Irecv/Waitall）




```
 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。

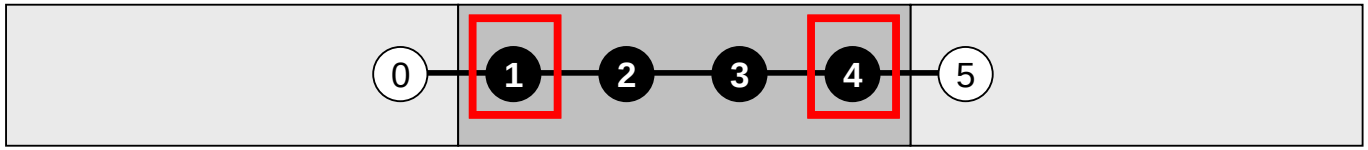
送信（MPI_Sendrecv）




```
 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

送信:一次元問題



- 送信相手
  - NEIBPETOT, NEIB(neib)
    - NEIBPETOT=2, NEIB(1)= my_rank-1, NEIB(2)= my_rank+1
- それぞれの送信相手に送るメッセージサイズ
  - export_index(neib), neib= 1, NEIBPETOT
    - export_index(0)=0, export_index(1)= 1, export_index(2)= 2
- 「境界点」番号
  - export_item(k), k= 1, export_index(NEIBPETOT)
    - export_item(1)= 1, export_item(2)= N
- それぞれの送信相手に送るメッセージ
  - SENDbuf(k), k= 1, export_index(NEIBPETOT)
    - SENDbuf(1)= BUF(1), SENDbuf(2)= BUF(N)

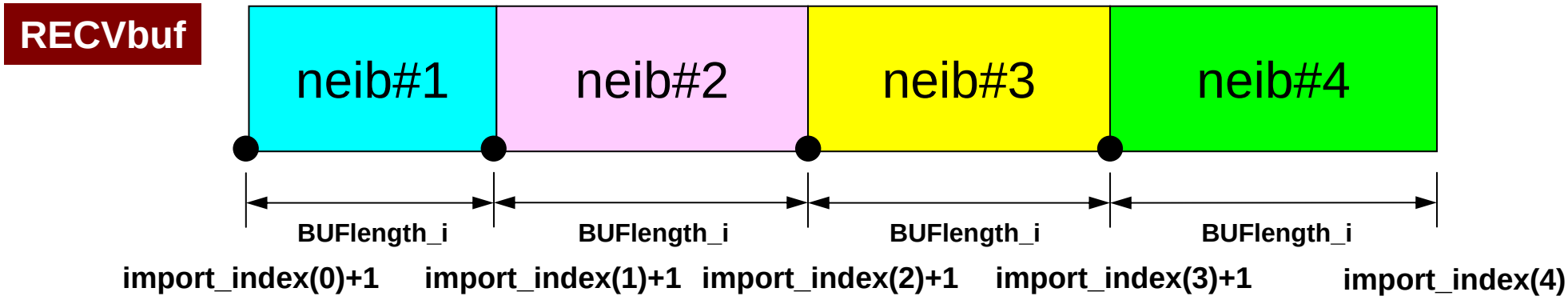
```
SENDbuf(1)=BUF(1)
SENDbuf(2)=BUF(4)
```

通信テーブルの一般的な形: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)
- 「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）

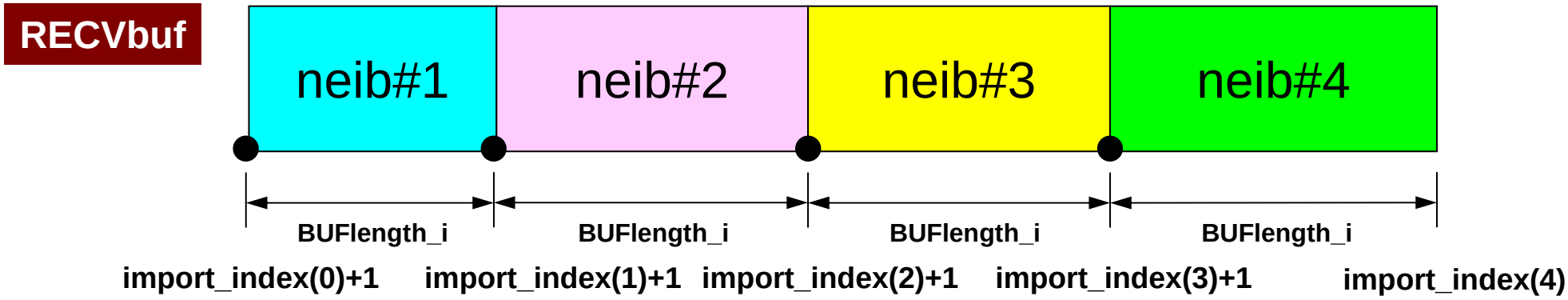
```
 do neib= 1, NEIBPETOT
   iS_i= import_index(neib-1) + 1
   iE_i= import_index(neib  )
   BUFlength_i= iE_i + 1 - iS_i

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



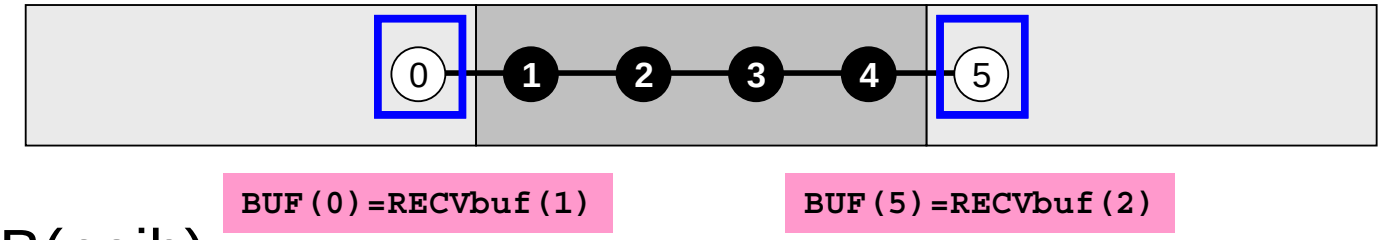
受信（MPI_Sendrecv）

```
 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



受信:一次元問題



- 受信相手
  - NEIBPETOT, NEIB(neib)
    - NEIBPETOT=2, NEIB(1)= my_rank-1, NEIB(2)= my_rank+1
- 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
    - import_index(0)=0, import_index(1)= 1, import_index(2)= 2
- 「外点」番号
  - import_item(k), k= 1, import_index(NEIBPETOT)
    - import_item(1)= 0, import_item(2)= N+1
- 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)
    - BUF(0)=RECVbuf(1), BUF(N+1)=RECVbuf(2)

一般化された通信テーブル：1d-srb1.f（1/7）

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

      integer(kind=4) :: my_rank, PETOT
      integer(kind=4) :: N, NEIBPETOT, BUFlength
      integer(kind=4), dimension(2) :: NEIBPE

      integer(kind=4), dimension(0:2) :: import_index, export_index
      integer(kind=4), dimension(  2) :: import_item , export_item

      integer(kind=4), dimension(2) :: SENDbuf, RECVbuf

      integer(kind=4), dimension(:), allocatable :: BUF

      integer(kind=4), dimension(:,:), allocatable :: stat_send
      integer(kind=4), dimension(:,:), allocatable :: stat_recv
      integer(kind=4), dimension(:  ), allocatable :: request_send
      integer(kind=4), dimension(:  ), allocatable :: request_recv

!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===
```

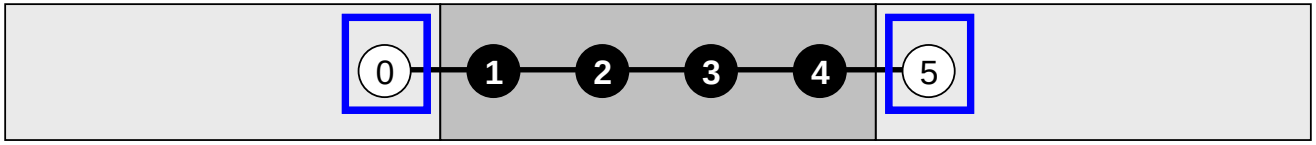
一般化された通信テーブル: 1d-srb1.f（2/7）

```
!C
!C +------+
!C | INIT |
!C +------+
!C===
      N= 4
      allocate (BUF(0:N+1))
      BUF= 0

      if (my_rank.eq.0) open (11, file='1d.0', status='unknown')
      if (my_rank.eq.1) open (11, file='1d.1', status='unknown')
      if (my_rank.eq.2) open (11, file='1d.2', status='unknown')
      if (my_rank.eq.3) open (11, file='1d.3', status='unknown')
        do i= 1, N
          read (11,*) BUF(i)
        enddo
      close (11)

      iSTART= 0
      iEND  = N+1
      if (my_rank.eq.0      ) iSTART= 1
      if (my_rank.eq.PETOT-1) iEND  = N
      do i= iSTART, iEND
        write (*,'(a, 3i8)') '%%% before', my_rank, i, BUF(i)
      enddo
```

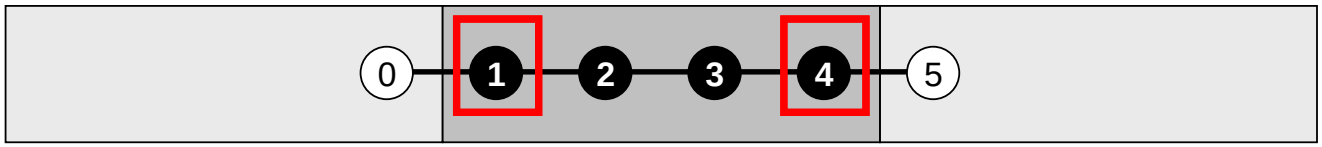
一般化された通信テーブル：1d-srb1.f（3/7）

```
!C
!C-- COMMUNICATION
      NEIBPETOT= 2
      if (my_rank.eq.0      ) NEIBPETOT= 1
      if (my_rank.eq.PETOT-1) NEIBPETOT= 1
      if (PETOT.eq.1)         NEIBPETOT= 0

      NEIBPE(1)= my_rank - 1
      NEIBPE(2)= my_rank + 1

      if (my_rank.eq.0      ) NEIBPE(1)= my_rank + 1
      if (my_rank.eq.PETOT-1) NEIBPE(1)= my_rank - 1

      import_index= 0
      export_index= 0
      import_item = 0
      export_item = 0

      import_index(1)= 1
      import_index(2)= 2
      import_item (1)= 0
      import_item (2)= N+1

      export_index(1)= 1
      export_index(2)= 2
      export_item (1)= 1
      export_item (2)= N

      if (my_rank.eq.0) then
        import_item (1)= N+1
        export_item (1)= N
      endif

      BUFlength= 1

      write (*,'(a,10i5)') '#NEIB (my_rank, NEIBPETOT, NEPBPE)',         &
     &                     my_rank, NEIBPETOT, (NEIBPE(i),i=1,NEIBPETOT)
!C===
```

一般化された通信テーブル



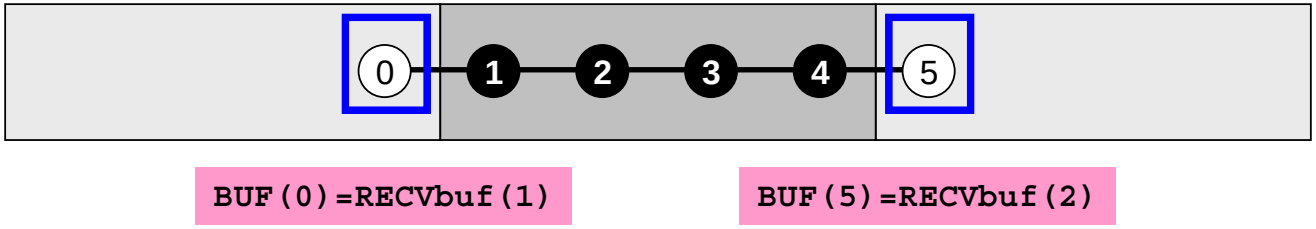
BUF(0)=RECVbuf(1)　　BUF(5)=RECVbuf(2)

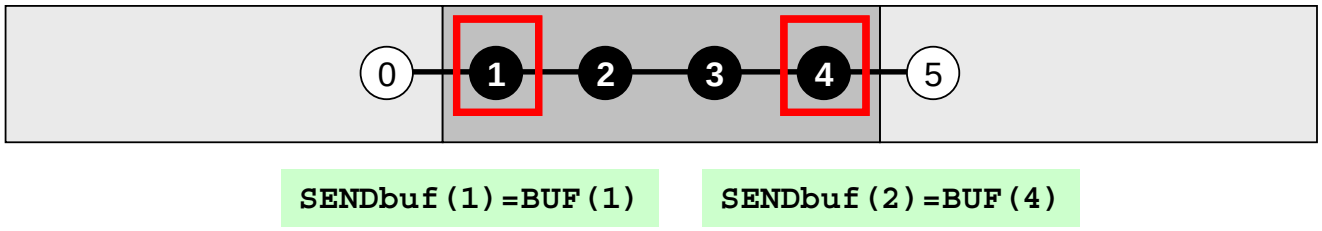


SENDbuf(1)=BUF(1)　　SENDbuf(2)=BUF(4)

```
NEIBPETOT= 2
NEIBPE(1)= my_rank - 1
NEIBPE(2)= my_rank + 1

import_index(1)= 1
import_index(2)= 2
import_item  (1)= 0
import_item  (2)= N+1

export_index(1)= 1
export_index(2)= 2
export_item  (1)= 1
export_item  (2)= N

if (my_rank.eq.0) then
  import_item  (1)= N+1
  export_item  (1)= N
  NEIBPE(1)= my_rank+1
endif
```

一般化された通信テーブル:C言語





C言語の場合BUF[-1]という配列は本来存在しないが,
コンパイラによっては通ってしまう場合がある。

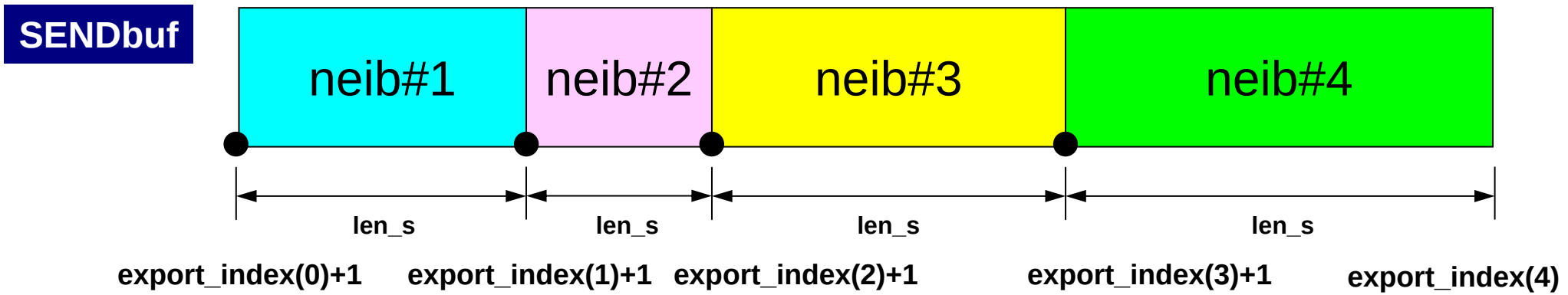
```
NEIBPETOT= 2
NEIBPE(1)= my_rank - 1
NEIBPE(2)= my_rank + 1

import_index(1)= 1
import_index(2)= 2
import_item (1)= -1
import_item (2)= N

export_index(1)= 1
export_index(2)= 2
export_item (1)= 0
export_item (2)= N-1

if (my_rank.eq.0) then
  import_item (1)= N
  export_item (1)= N-1
  NEIBPE(1)= my_rank+1
endif
```

一般化された通信テーブル: 1d-srb1.f（4/7）

```
!C
!C +-----------------------------+
!C | INIT. arrays for MPI_WAITALL |
!C +-----------------------------+
!C===
      allocate (stat_send(MPI_STATUS_SIZE,NEIBPETOT))
      allocate (stat_recv(MPI_STATUS_SIZE,NEIBPETOT))
      allocate (request_send(NEIBPETOT))
      allocate (request_recv(NEIBPETOT))
!C===
```

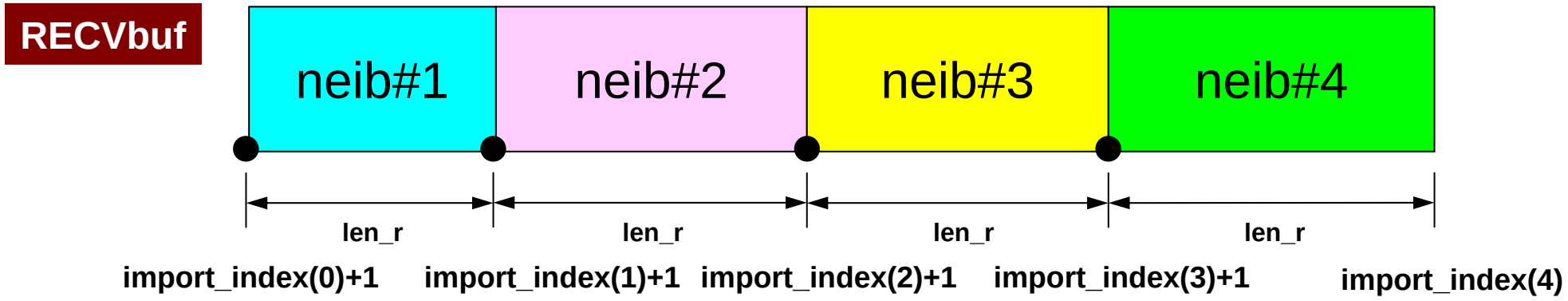
一般化された通信テーブル：1d-srb1.f（5/7）

```
!C
!C-- PREPARE send  buffer
      do neib= 1, NEIBPETOT
        do k= export_index(neib-1)+1, export_index(neib)
          kk= export_item(k)
          SENDbuf(k)= BUF(kk)
        enddo
      enddo

!C
!C-- SEND
      do neib= 1, NEIBPETOT
        is   = export_index(neib-1) + 1
        len_s= export_index(neib) - export_index(neib-1)
        call MPI_ISEND (SENDbuf(is), len_s, MPI_INTEGER,                &
     &                  NEIBPE(neib), 0, MPI_COMM_WORLD,                &
     &                  request_send(neib), ierr)
      enddo
```



一般化された通信テーブル：1d-srb1.f（6/7）

```
!C
!C-- RECV
      do neib= 1, NEIBPETOT
        ir   = import_index(neib-1) + 1
        len_r= import_index(neib) - import_index(neib-1)
        call MPI_IRECV (RECVbuf(ir), len_r, MPI_INTEGER,                &
     &                  NEIBPE(neib), 0, MPI_COMM_WORLD,                &
     &                  request_recv(neib), ierr)
      enddo

!C
!C-- WAITall for RECV
      call MPI_WAITALL (NEIBPETOT, request_recv, stat_recv, ierr)

!C
!C-- update array
      do neib= 1, NEIBPETOT
        do k= import_index(neib-1)+1, import_index(neib)
          kk= import_item(k)
          BUF(kk)= RECVbuf(k)
        enddo
      enddo
```



一般化された通信テーブル：1d-srb1.f（7/7）

```
!C
!C +-------------------+
!C | WAITall for SEND |
!C +-------------------+
!C===
      call MPI_WAITALL (NEIBPETOT, request_send, stat_send, ierr)
!C===

      call MPI_FINALIZE (ierr)

      end
```

- 課題S2

課題S2（1/2）

- 「<$S2>/heat_jacobi.f」,「<$S2>/heat_jacobi.c」を参考にして，一次元熱伝導方程式をJacobi法によって解くプログラムを並列化せよ（**S2-1**）。
- 「<$S2>/heat_gs.f」,「<$S2>/heat_gs.c」を参考にして，一次元熱伝導方程式をGauss-Seidel法によって解くプログラムを並列化せよ（**S2-2**）。

- 「<$S2>/1d-srb1.f, 1d-srb2.f」,「<$S2>/1d-srb1.c, 1d-srb2.c」を参考にして「一般化された通信テーブル」を使用せよ。
  - BUFのかわりに温度をやりとりする

課題S2（2/2）

- NG=100およびNG=103の場合について，1，2，4，8CPUを使用して計算してみよ。
  - 実は，N=100程度では並列化の効果は余り･･･全く得られない

- Gauss-Seidel法の場合，単純に並列化すると，領域分割数を増加させた場合，反復回数が増加する。その理由について考えてみよ。
  - Jacobi法の場合はこのような現象が生じない。その理由についても併せて検討すること。
- 提出期限
  - 2007年9月19日（水）17:00