Introduction to Programming by MPI for Parallel FEM Report S1 & S2 in Fortran (2/2)

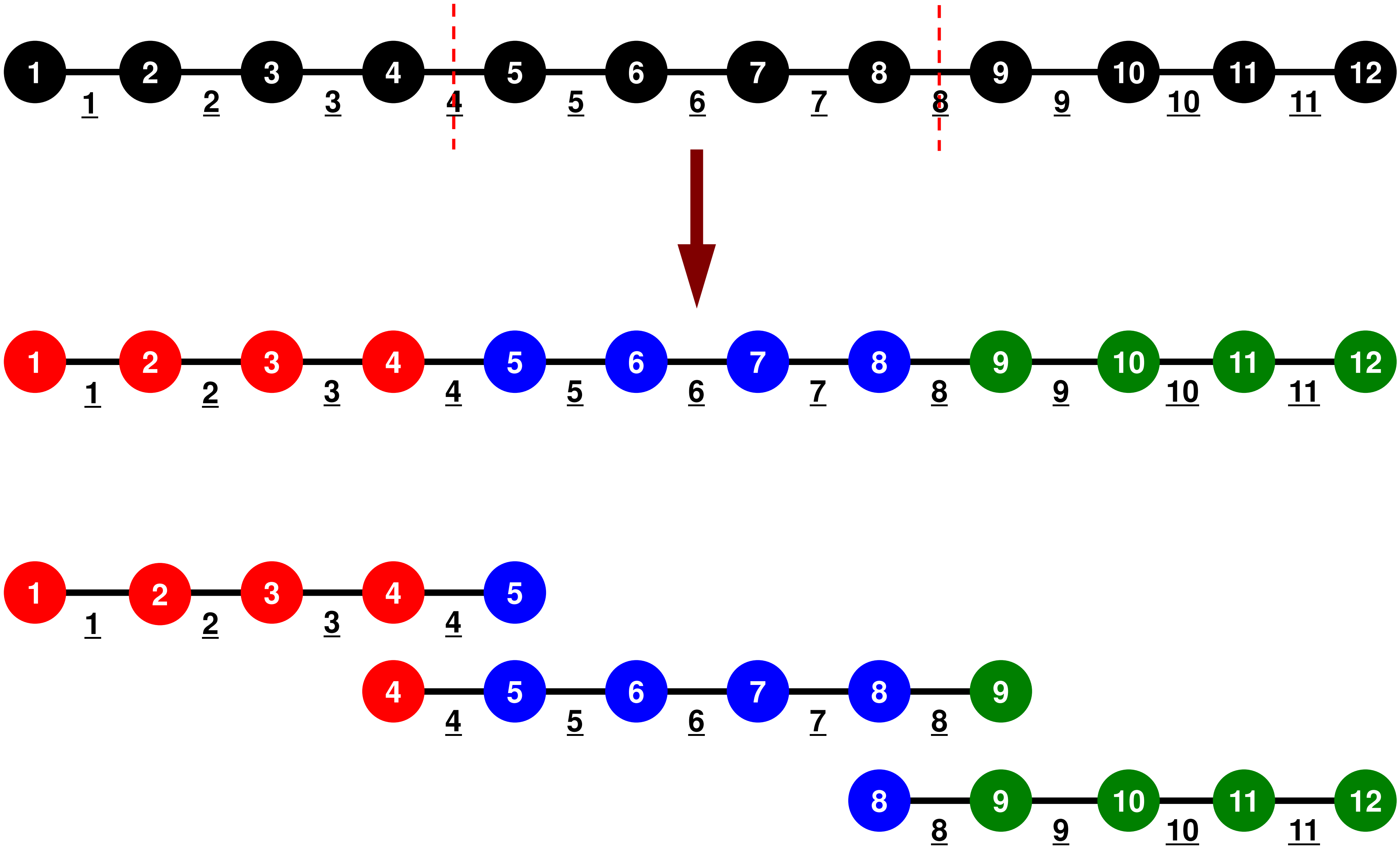
Kengo Nakajima
RIKEN R-CCS

- What is MPI ?
- Your First MPI Program: Hello World
- Collective Communication
- **Point-to-Point Communication**

Point-to-Point Communicatio
1対1通信

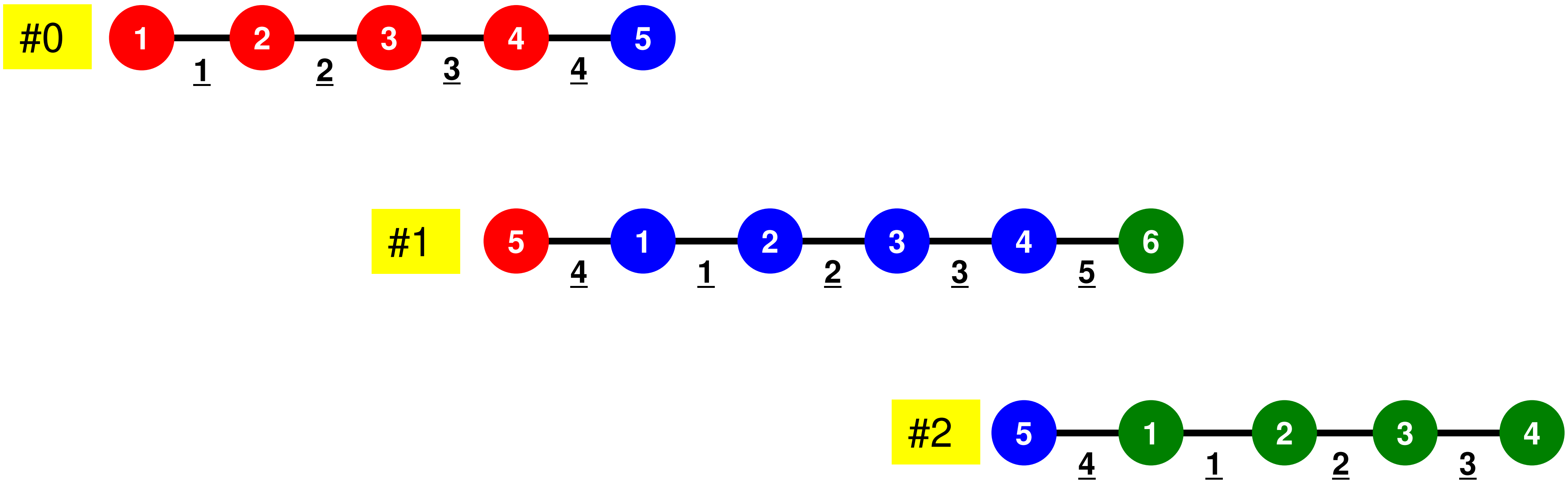
- What is PtoP Communication ?
- 2D Problem, Generalized Communication Table
- Report S2

1D FEM: 12 nodes/11 elem's/3 domains



1D FEM: 12 nodes/11 elem's/3 domains

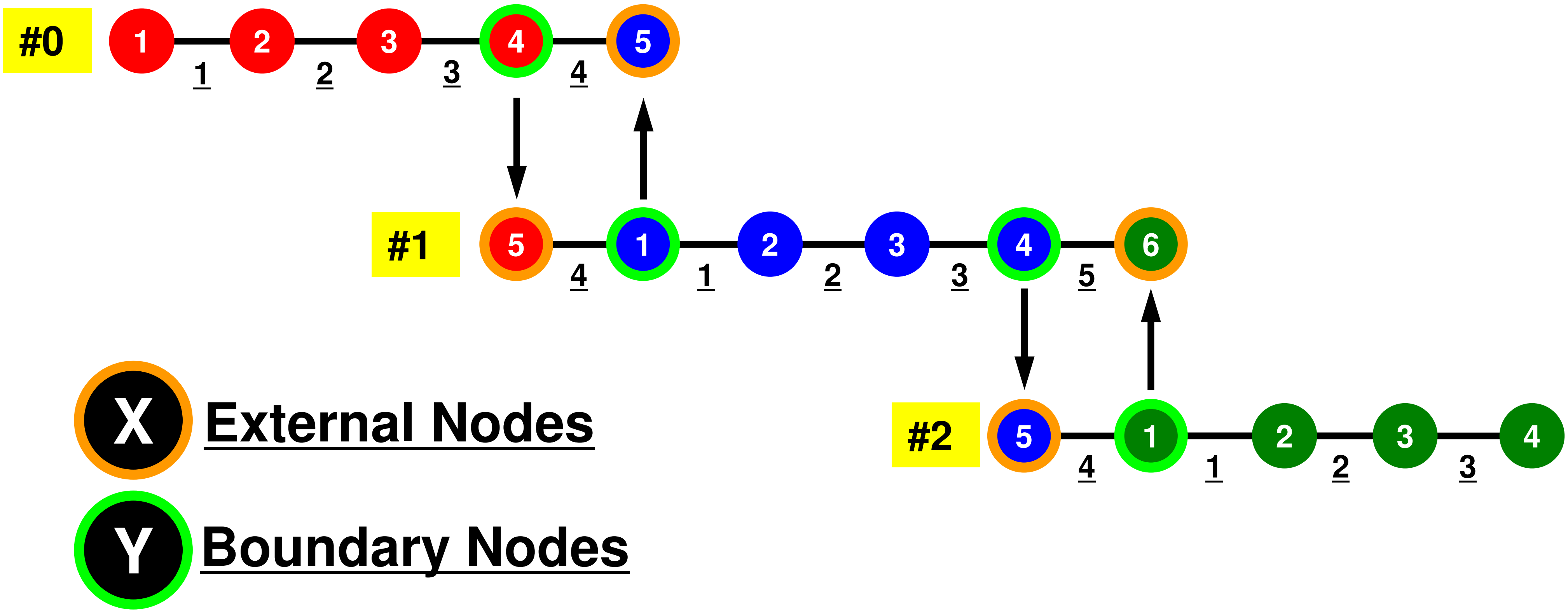
Local ID: Starting from 1 for node and elem at each domain



1D FEM: 12 nodes/11 elem's/3 domains

Internal/External/Boundary Nodes

Boundary Nodes: Part of Internal Nodes, and External Nodes of Other Domains



Preconditioned Conjugate Gradient Method (CG)

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

Preconditioner:

Diagonal Scaling
Point-Jacobi Preconditioning

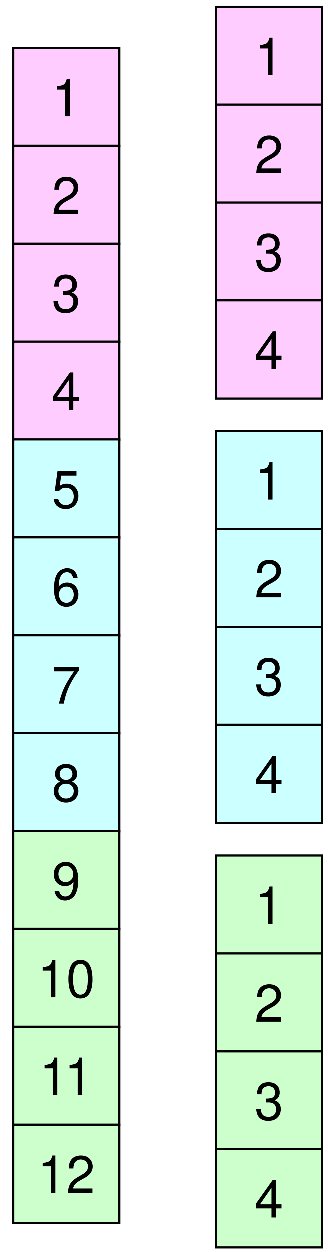
$$[M]=\begin{bmatrix} D_1 & 0 & \dots & 0 & 0 \\ 0 & D_2 & & 0 & 0 \\ \dots & & \dots & & \dots \\ 0 & 0 & & D_{N-1} & 0 \\ 0 & 0 & \dots & 0 & D_N \end{bmatrix}$$

Preconditioning, DAXPY

Local Operations by Only Internal Points: Parallel Processing is possible

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

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

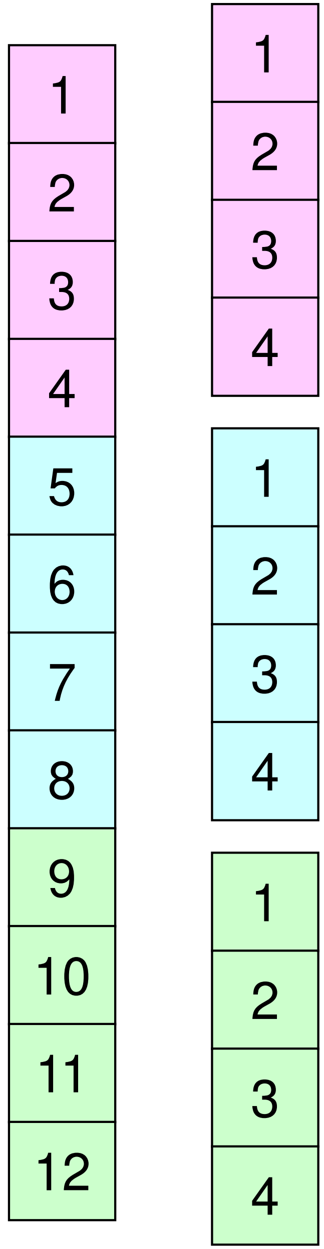
```
!C
!C-- {x}= {x} + ALPHA*{p}        DAXPY: double a{x} plus {y}
!C   {r}= {r} - ALPHA*{q}

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



Dot Products

Global Summation needed: Communication ?

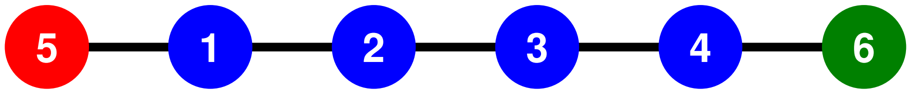
```
!C
!C-- ALPHA= RHO / {p}{q}

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



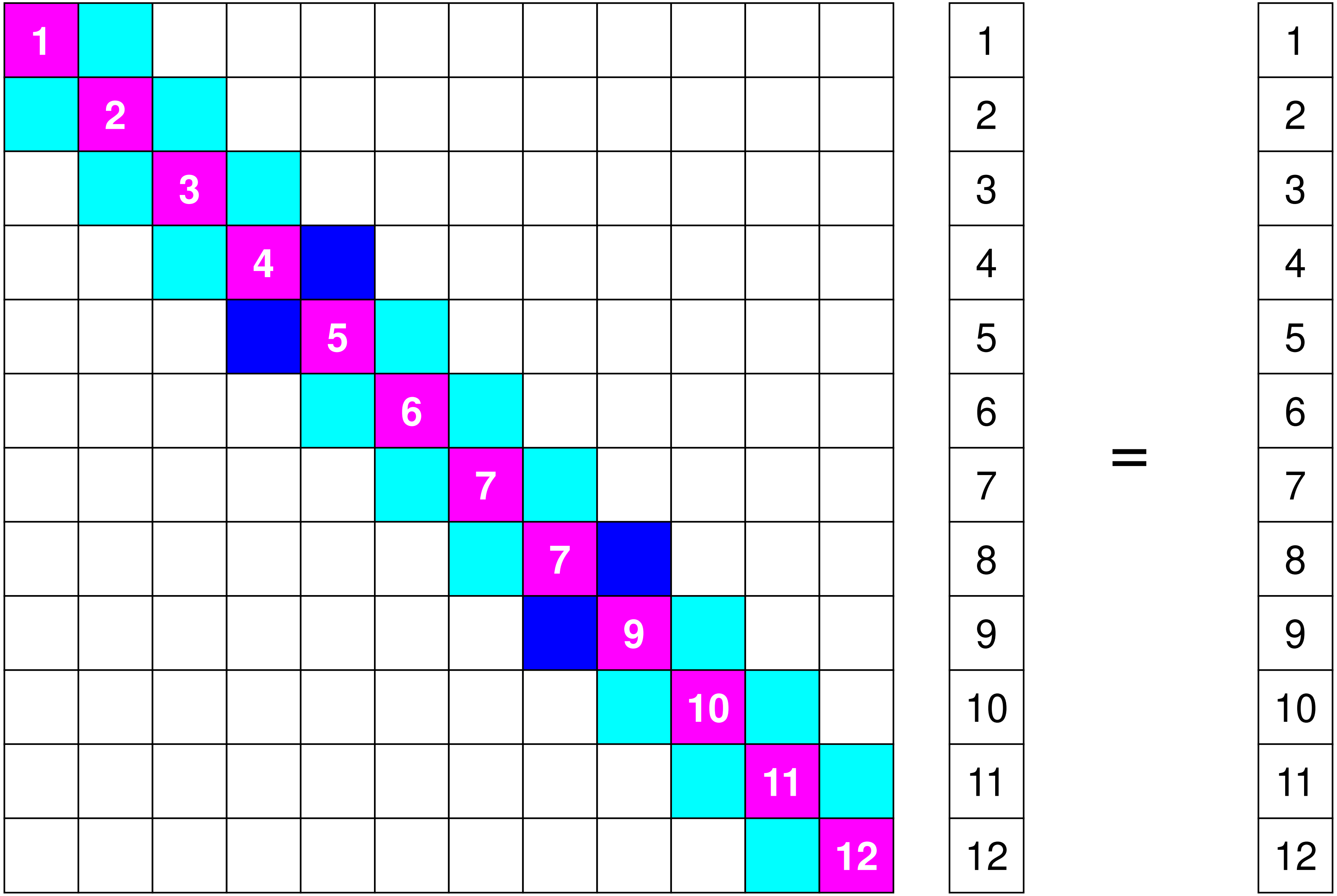
Matrix-Vector Products

Values at External Points: P-to-P Communication

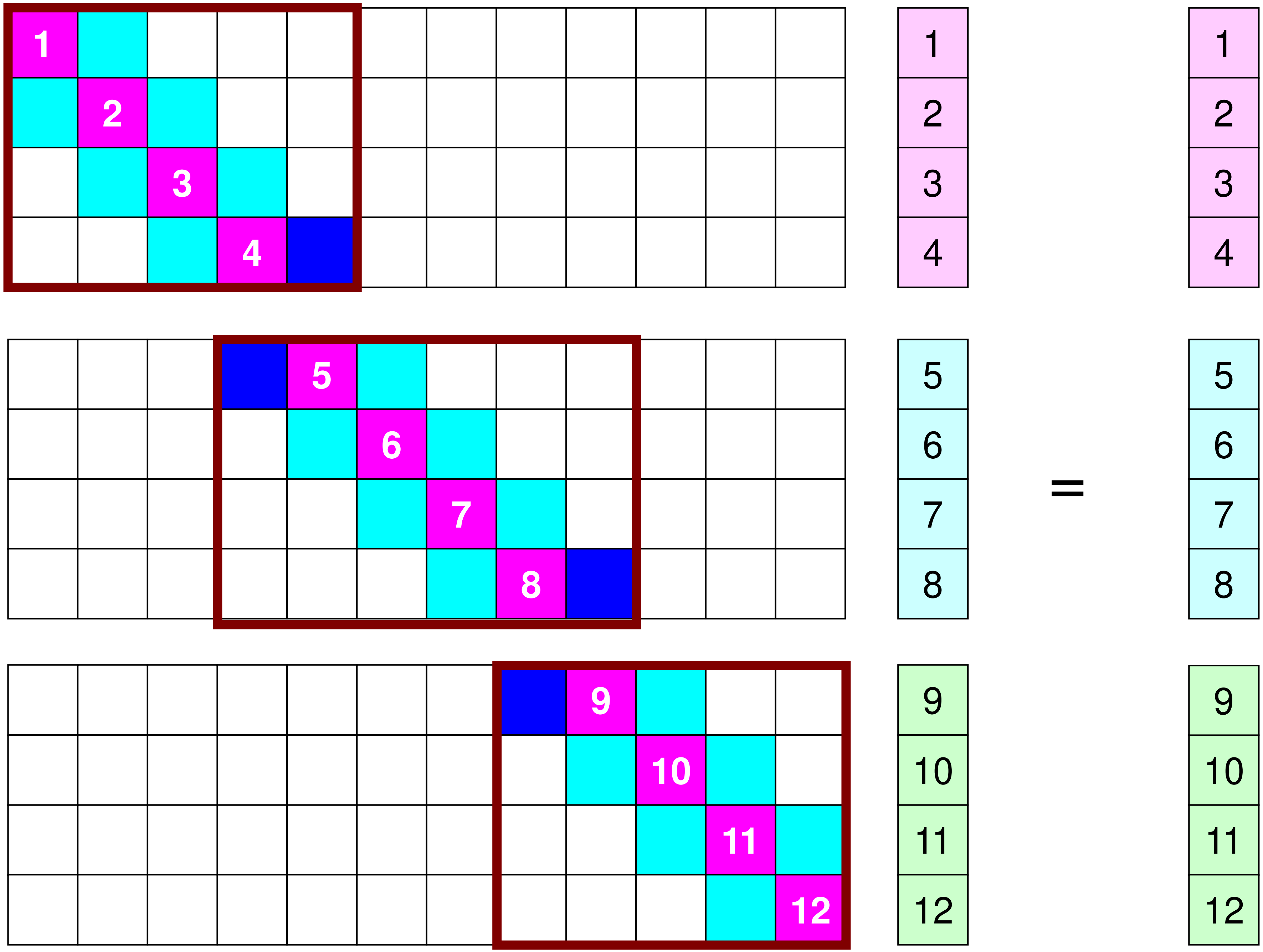
```
!C
!C-- {q}= [A]{p}

      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
```



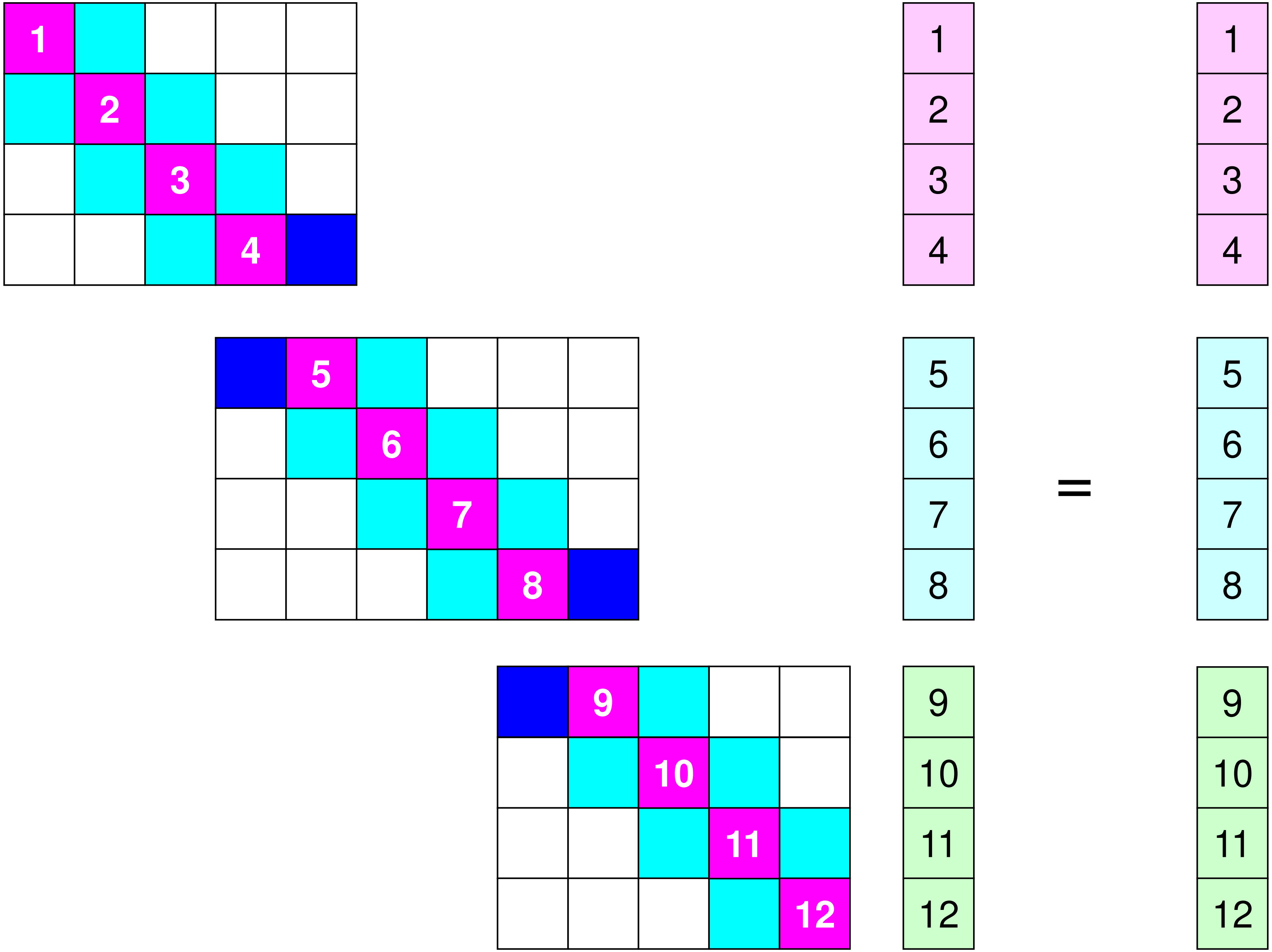
Mat-Vec Products: Local Op. Possible



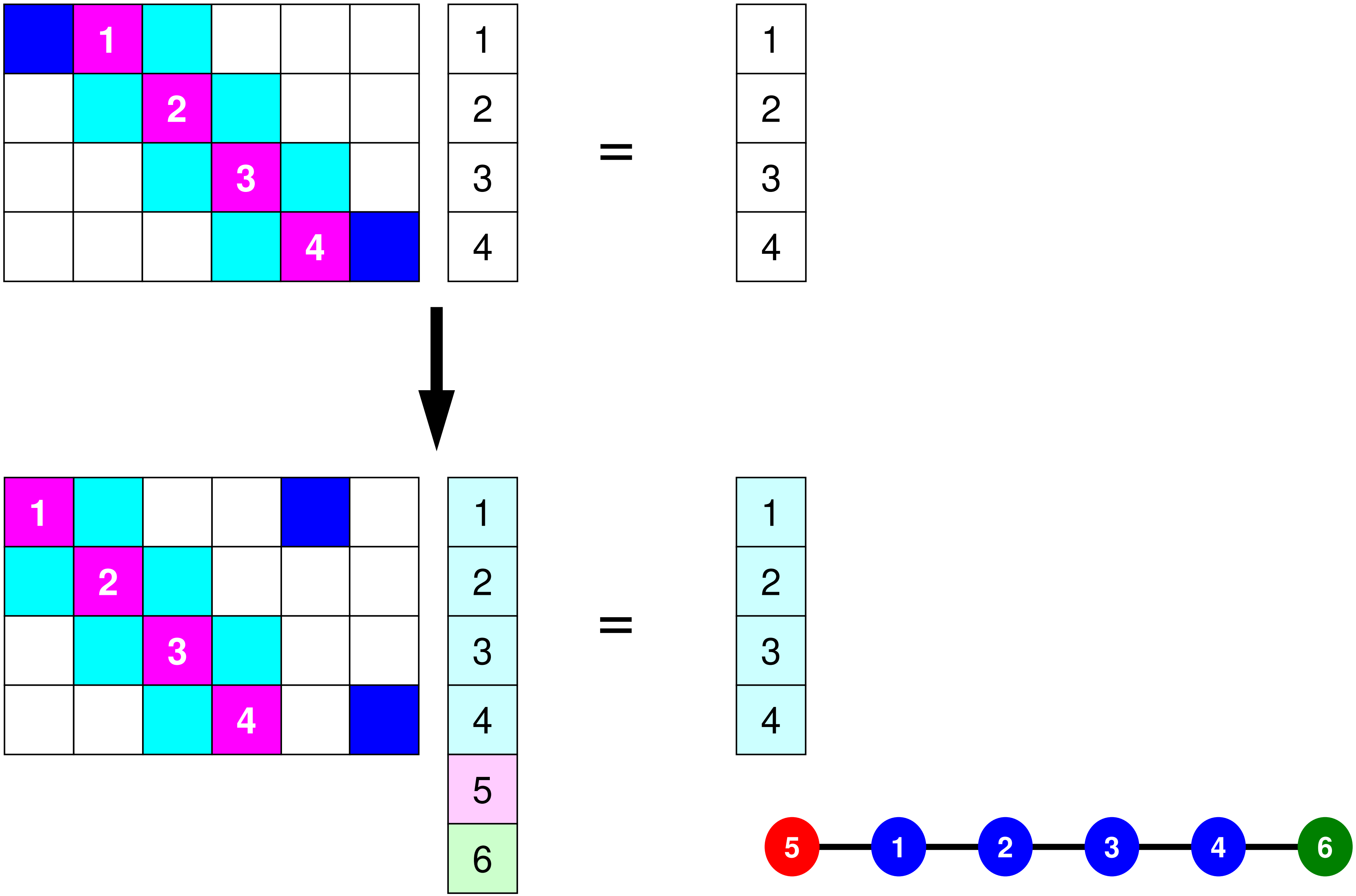
Mat-Vec Products: Local Op. Possible



Mat-Vec Products: Local Op. Possible



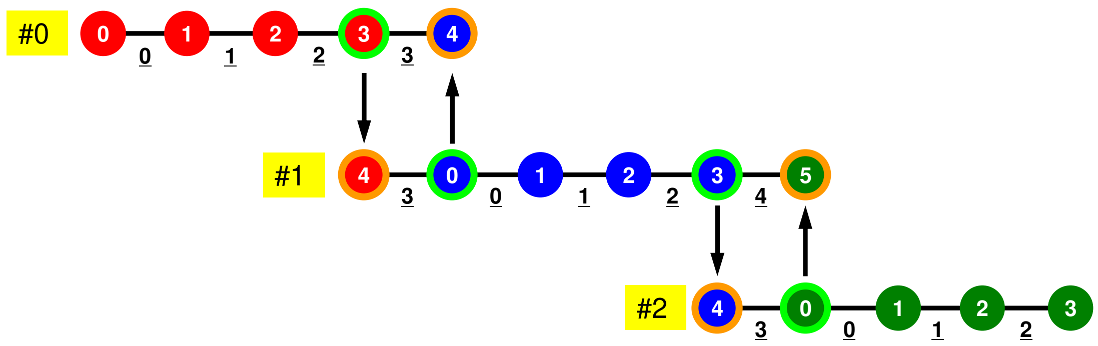
Mat-Vec Products: Local Op. #1



What is Point-to-Point Comm. ?

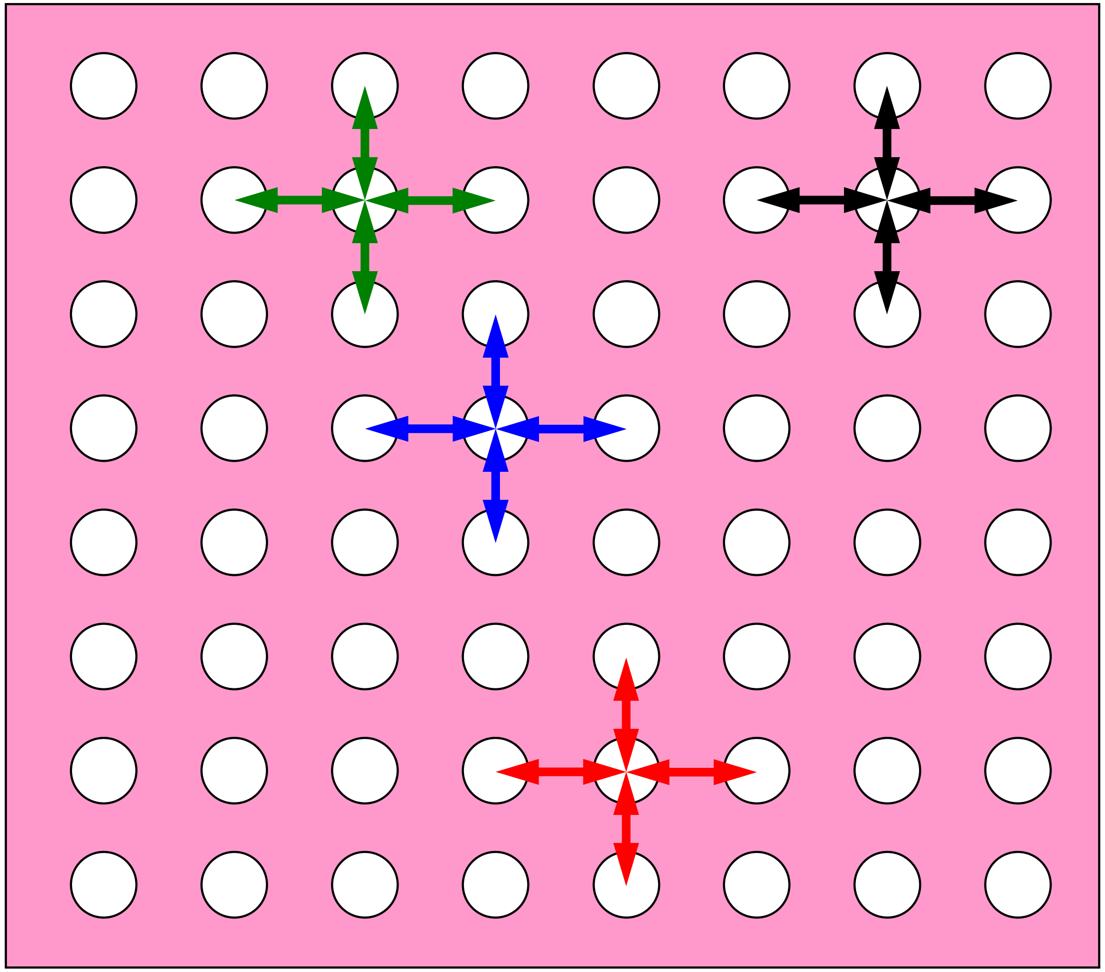
- Collective Communication
  - MPI_Reduce, MPI_Scatter/Gather etc.
  - Communications with all processes in the communicator
  - Application Area
    - BEM, Spectral Method, MD: global interactions are considered
    - Dot products, MAX/MIN: Global Summation & Comparison

- Peer-toPeer/Point-to-Point
  - MPI_Send, MPI_Recv
  - Communication with limited processes
    - Neighbors
  - Application Area
    - FEM, FDM: Localized Method



Collective/PtoP Communications

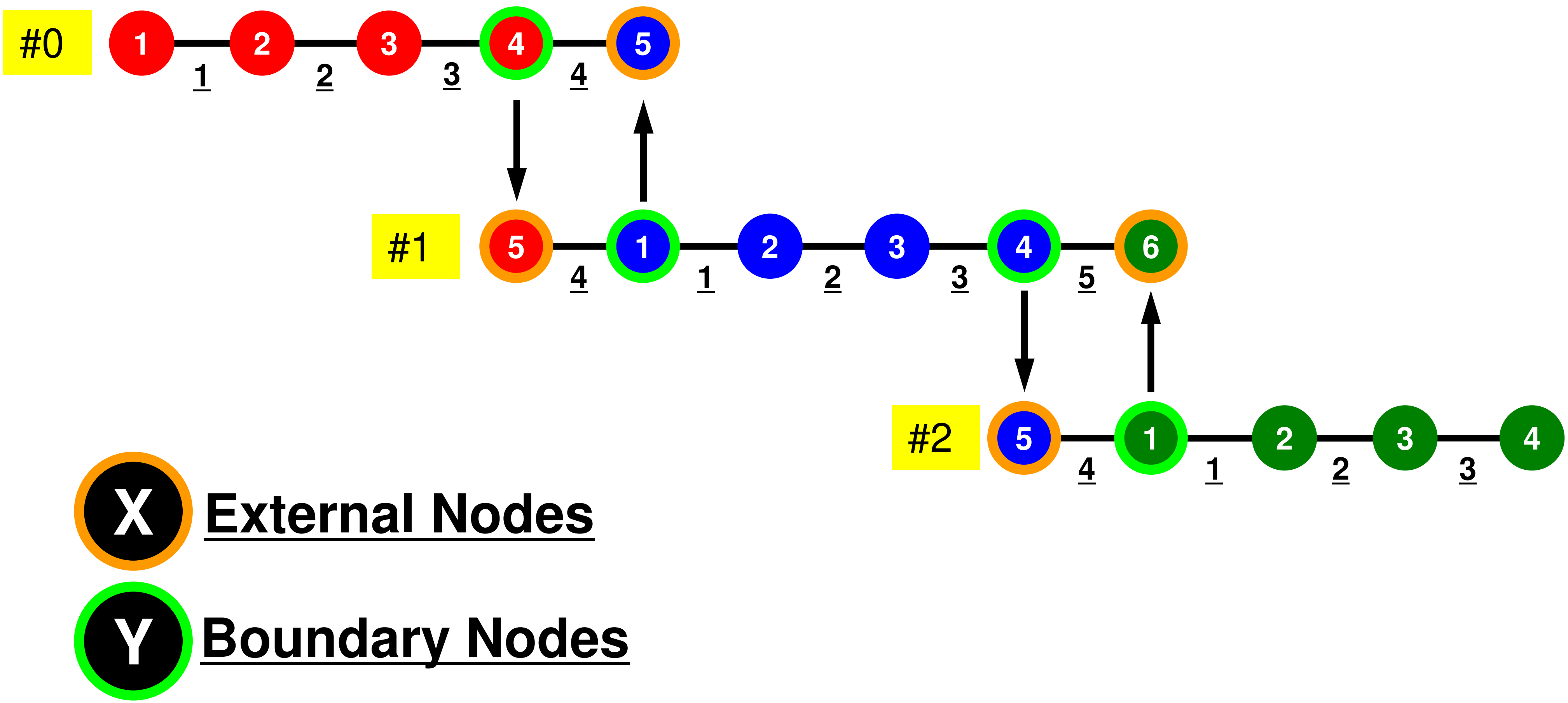
Interactions with only Neighboring Processes/Element
Finite Difference Method (FDM), Finite Element Method (FEM)



When do we need PtoP comm.: 1D-FEM

Info in neighboring domains is required for FEM operations

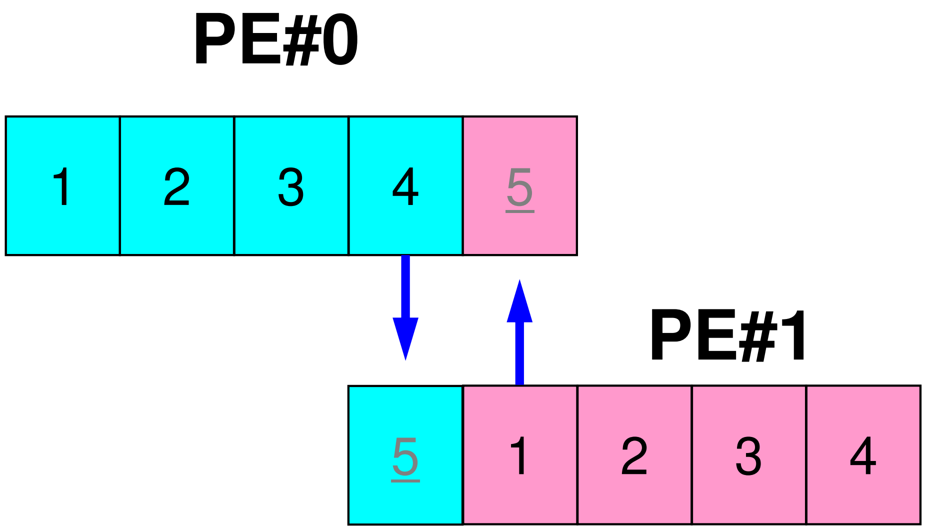
Matrix assembling, Iterative Method



Method for P-to-P Comm.

- **`MPI_Send`**, **`MPI_Recv`**
- These are "blocking" functions.
  - "Dead lock" occurs for these "blocking" functions.
- A "blocking" MPI call means that the program execution will be suspended until the message buffer is safe to use.
- The MPI standards specify that a blocking SEND or RECV does not return until the send buffer is safe to reuse (for MPI_Send), or the receive buffer is ready to use (for MPI_Recv).
  - Blocking comm. confirms "secure" communication, but it is very inconvenient and impractical.
- Please just remember that "there are such functions".

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)
…
```

- This seems reasonable, but it stops at MPI_Send/MPI_Recv.
  - Sometimes it works (according to implementation).
- "Sending 0-to-1" does not terminate, until "Receiving@1-from-0 is done.
  - But, "Sending 1-to-0" does not terminate, if "Receiving@0-from-1" is not done.

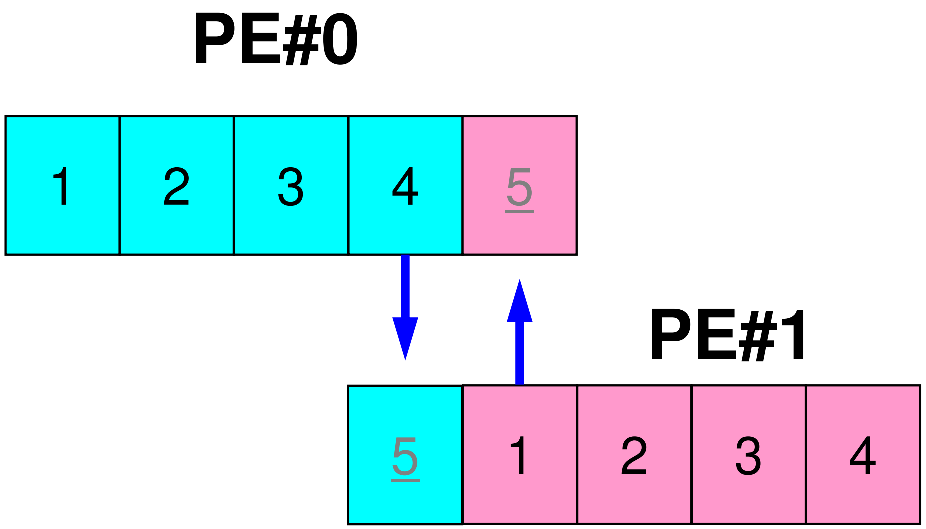
MPI_Send/MPI_Recv

```
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)
```

- Both of PE#0 and PE#1 are "sending"
- "Sending" does not terminate, if "receiving" at destination is not completed
- Both of PE#0 and PE#1 are waiting for completion of "receiving" after "sending" -> Processes stop

MPI_Send/MPI_Recv (cont.)

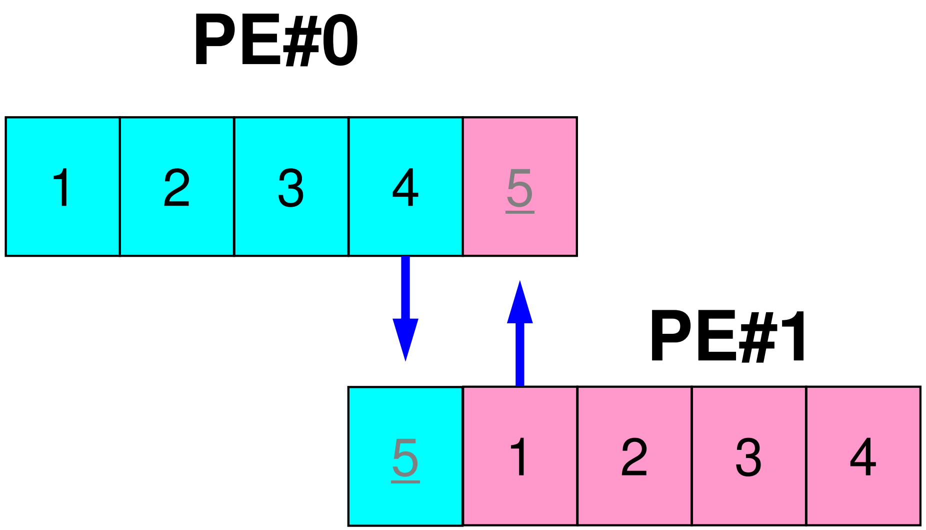



```
if (my_rank.eq.0) NEIB_ID=1
if (my_rank.eq.1) NEIB_ID=0

…
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

…
```

- It works.
- It is OK for Structured Meshes for FDM (Finite Difference Method).

How to do PtoP Comm. ?

- Using “non-blocking” functions **`MPI_Isend`** & **`MPI_Irecv`** together with **`MPI_Waitall`** for synchronization
- **`MPI_Sendrecv`** is also available.




```
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)
```

`MPI_Waitall` for both of **`MPI_Isend/MPI_Irecv`** is possible

Fortran

MPI_ISEND

- Begins a non-blocking send
  - Send the contents of sending buffer (starting from **sendbuf**, number of messages: **count**) to **dest** with **tag** .
  - Contents of sending buffer cannot be modified before calling corresponding **MPI_Waitall**.

- **call MPI_ISEND (sendbuf,count,datatype,dest,tag,comm,request, ierr)**

– **sendbuf**	choice	I	starting address of sending buffer
– **count**	I	I	number of elements sent to each process
– **datatype**	I	I	data type of elements of sending buffer
– **dest**	I	I	rank of destination
– **tag**	I	I	message tag This integer can be used by the application to distinguish messages. Communication occurs if `tag's` of `MPI_Isend` and `MPI_Irecv` are matched. Usually tag is set to be “0” (in this class),
– **comm**	I	I	communicator
– **request**	I	O	communication request array used in `MPI_Waitall`
– **ierr**	I	O	completion code

Communication Request: request 通信識別子

- **call MPI_ISEND (sendbuf,count,datatype,dest,tag,comm,request, ierr)**

– **sendbuf**	choice	I	starting address of sending buffer
– **count**	I	I	number of elements sent to each process
– **datatype**	I	I	data type of elements of sending buffer
– **dest**	I	I	rank of destination
– **tag**	I	I	message tag This integer can be used by the application to distinguish messages. Communication occurs if `tag's` of `MPI_Isend` and `MPI_Irecv` are matched. Usually tag is set to be "0" (in this class),
– **comm**	I	I	communicator
– **request**	I	O	communication request used in `MPI_Waitall` Size of the array is total number of neighboring processes
– **ierr**	I	O	completion code

- Just define the array

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

Fortran

Fortran

MPI_IRECV

- Begins a non-blocking receive
  - Receiving the contents of receiving buffer (starting from **recvbuf**, number of messages: **count**) from **source** with **tag** .
  - Contents of receiving buffer cannot be used before calling corresponding **MPI_Waitall**.

- **call MPI_IRECV (recvbuf,count,datatype,dest,tag,comm,request, ierr)**

– **recvbuf**	choice	I	starting address of receiving buffer
– **count**	I	I	number of elements in receiving buffer
– **datatype**	I	I	data type of elements of receiving buffer
– **source**	I	I	rank of source
– **tag**	I	I	message tag This integer can be used by the application to distinguish messages. Communication occurs if `tag's` of `MPI_Isend` and `MPI_Irecv` are matched. Usually tag is set to be "0" (in this class),
– **comm**	I	I	communicator
– **request**	I	O	communication request used in `MPI_Waitall`
– **ierr**	I	O	completion code

MPI_WAITALL

Fortran

- **MPI_Waitall** blocks until all comm's, associated with **request** in the array, complete. It is used for terminating **MPI_Isend** and **MPI_Irecv**.
- At sending phase, contents of sending buffer cannot be modified before calling corresponding **MPI_Waitall**. At receiving phase, contents of receiving buffer cannot be used before calling corresponding **MPI_Waitall**.
- **MPI_Isend** and **MPI_Irecv** can be synchronized simultaneously with a single **MPI_Waitall** if it is consitent.
  - Same **request** should be used in **MPI_Isend** and **MPI_Irecv**.
- Its operation is similar to that of **MPI_Barrier** but, **MPI_Waitall** can not be replaced by **MPI_Barrier**.
  - Possible troubles using **MPI_Barrier** instead of **MPI_Waitall**: Contents of **request** and **status** are not updated properly, very slow operations etc.

- **call MPI_WAITALL (count,request,status,ierr)**
  - **count** I I number of processes to be synchronized
  - **request** I I/O comm. request used in `MPI_Waitall` (array size: `count`)
  - **status** I O array of status objects
    `MPI_STATUS_SIZE: defined in 'mpif.h', 'mpi.h'`
  - **ierr** I O completion code

Array of status object：status 状況オブジェクト配列

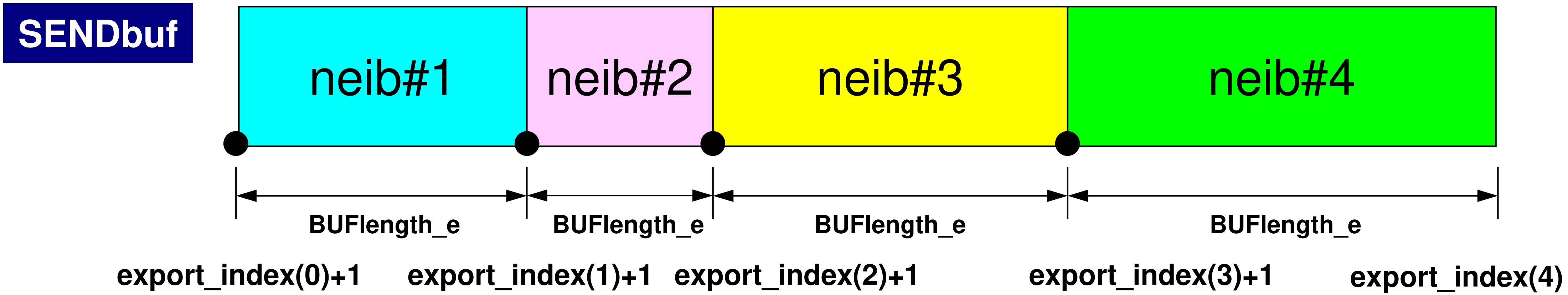
- **call MPI_WAITALL (count,request,status,ierr)**
  - **count** I I number of processes to be synchronized
  - **request** I I/O comm. request used in `MPI_Waitall` (array size: `count`)
  - **status** I O array of status objects
    `MPI_STATUS_SIZE: defined in 'mpif.h', 'mpi.h'`
  - **ierr** I O completion code

- Just define the array

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

Fortran

SEND: MPI_Isend/Irecv/Waitall

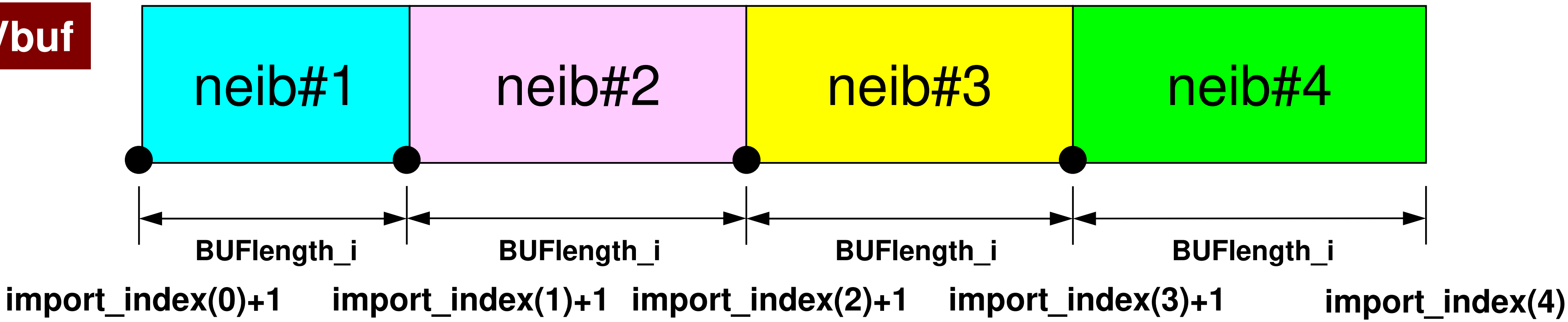



```
 do neib= 1, NEIBPETOT
   iS_e= export_index(neib-1) + 1
   iE_e= export_index(neib  )
   BUFlength_e= iE_e + 1 - iS_e

   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)
```

RECVbuf



Fortran

MPI_SENDRECV

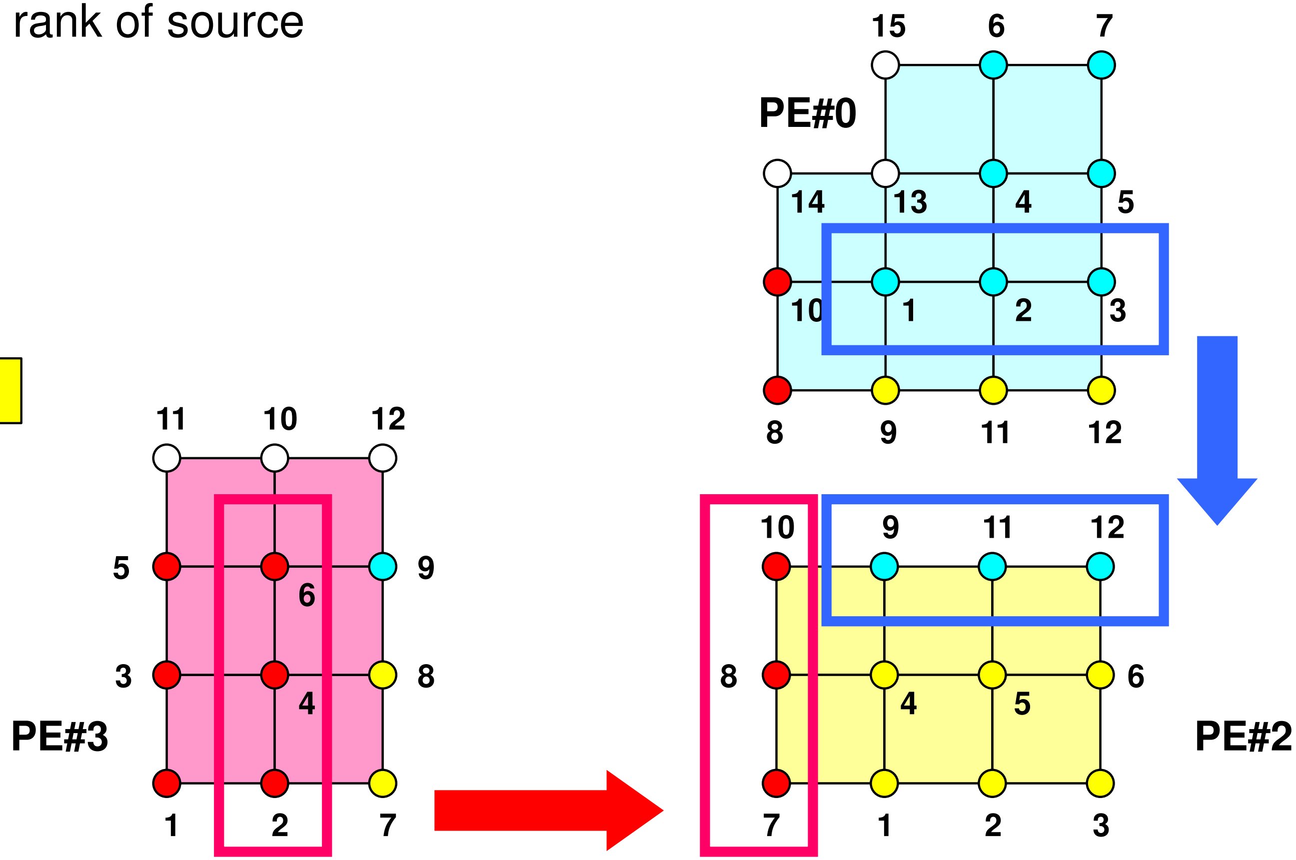
- MPI_Send+MPI_Recv : not recommended, many restrictions

- **call MPI_SENDRECV (sendbuf,sendcount,sendtype,dest,sendtag,recvbuf, recvcount,recvtype,source,recvtag,comm,status,ierr)**
  - **sendbuf** choice I starting address of sending buffer
  - **sendcount** I I number of elements in sending buffer
  - **sendtype** I I datatype of each sending buffer element
  - **dest** I I rank of destination
  - **sendtag** I I message tag for sending
  - **comm** I I communicator
  - **recvbuf** choice I starting address of receiving buffer
  - **recvcount** I I number of elements in receiving buffer
  - **recvtype** I I datatype of each receiving buffer element
  - **source** I I rank of source
  - **recvtag** I I message tag for receiving
  - **comm** I I communicator
  - **status** I O array of status objects
    MPI_STATUS_SIZE: defined in 'mpif.h', 'mpi.h'
  - **ierr** I O completion code

RECV: receiving to external nodes

Recv. continuous data to recv. buffer from neighbors

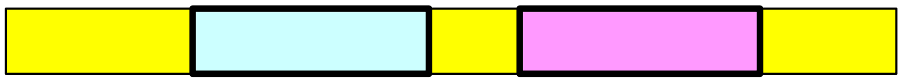
- **`MPI_Irecv`**
  `(recvbuf,count,datatype,dest,tag,comm,request)`
  - **`recvbuf`** choice `I` starting address of receiving buffer
  - **`count`** `I` `I` number of elements in receiving buffer
  - **`datatype`** `I` `I` data type of elements of receiving buffer
  - **`source`** `I` `I` rank of source

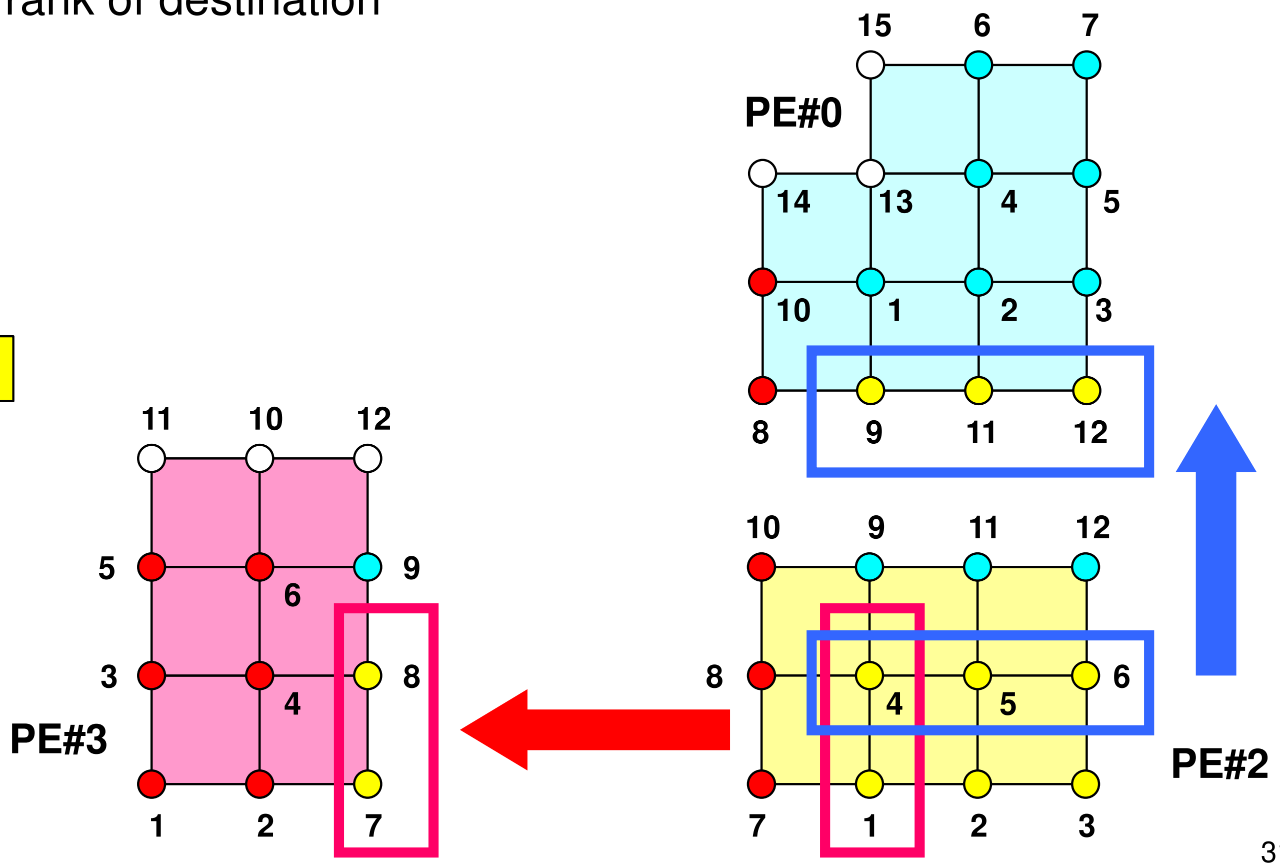


SEND: sending from boundary nodes

Send continuous data to send buffer of neighbors

- **MPI_Isend**
  (sendbuf,count,datatype,dest,tag,comm,request)
  - **sendbuf** choice I starting address of sending buffer
  - **count** I I number of elements sent to each process
  - **datatype** I I data type of elements of sending buffer
  - **dest** I I rank of destination





Request, Status in 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)
```

Files on Fugaku

Fortan

```
>$ cd /home/ra020019/<Your-UID>/pFEM
>$ cp /vol0001/ra020019/pFEM/F/s2-f.tar .
>$ tar xvf s2-f.tar
```

C

```
>$ cd /home/ra020019/<Your-UID>/pFEM
>$ cp /vol0001/ra020019/pFEM/C/s2-c.tar .
>$ tar xvf s2-c.tar
```

Confirmation

```
>$ ls
    mpi

>$ cd mpi/S2
```

This directory is called as <$O-S2>.

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

Ex.1: Send-Recv a Scalar

- Exchange VAL (real, 8-byte) between PE#0 & PE#1

```
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,…) Recv.buf VALtemp can be used
call MPI_Waitall (…,req_send,stat_send,…) Send buf VAL can be modified
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
```

Name of recv. buffer could be "VAL", but not recommended.

Ex.1: Send-Recv a Scalar

Isend/Irecv/Waitall

```
$> cd /home/ra020019/<Your-UID>/pFEM/mpi/S2
$> mpifrtpx -Kfast ex1-1.f
$> 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
```

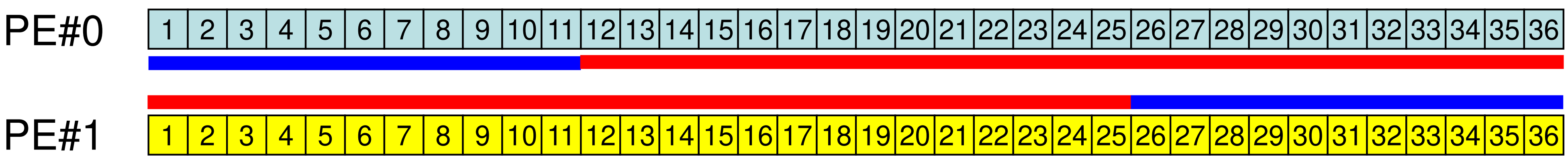
go2.sh

```
#!/bin/bash
#PJM -N "test"
#PJM -L “rscgrp=small”
#PJM -L “node=1:torus”
#PJM --mpi “max-proc-per-node=2”
#PJM -L elapse=00:15:00
#PJM -g ra020019
#PJM -j
#PJM -e err
#PJM -o test.lst


mpiexec ./a.out
```

Ex.2: Send-Recv an Array (1/3)

- Exchange VEC (real, 8-byte) between PE#0 & PE#1
- PE#0 to PE#1
  - PE#0: send VEC(1)-VEC(11) (length=11)
  - PE#1: recv. as VEC(26)-VEC(36) (length=11)
- PE#1 to PE#0
  - PE#1: send VEC(1)-VEC(25) (length=25)
  - PE#0: recv. as VEC(12)-VEC(36) (length=25)

- Practice: Develop a program for this operation.



t1

Practice: t1

- Initial status of VEC(:):
  - PE#0 VEC(1-36)= 101,102,103,~,135,136
  - PE#1 VEC(1-36)= 201,202,203,~,235,236

- Confirm the results in the next page

- MPI_Isend/Irecv/Waitall

Estimated Results

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.
```

t1

Ex.2: Send-Recv an Array (2/3)

```
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,…)
```

It works, but complicated operations.
Not looks like SPMD.
Not portable.

t1

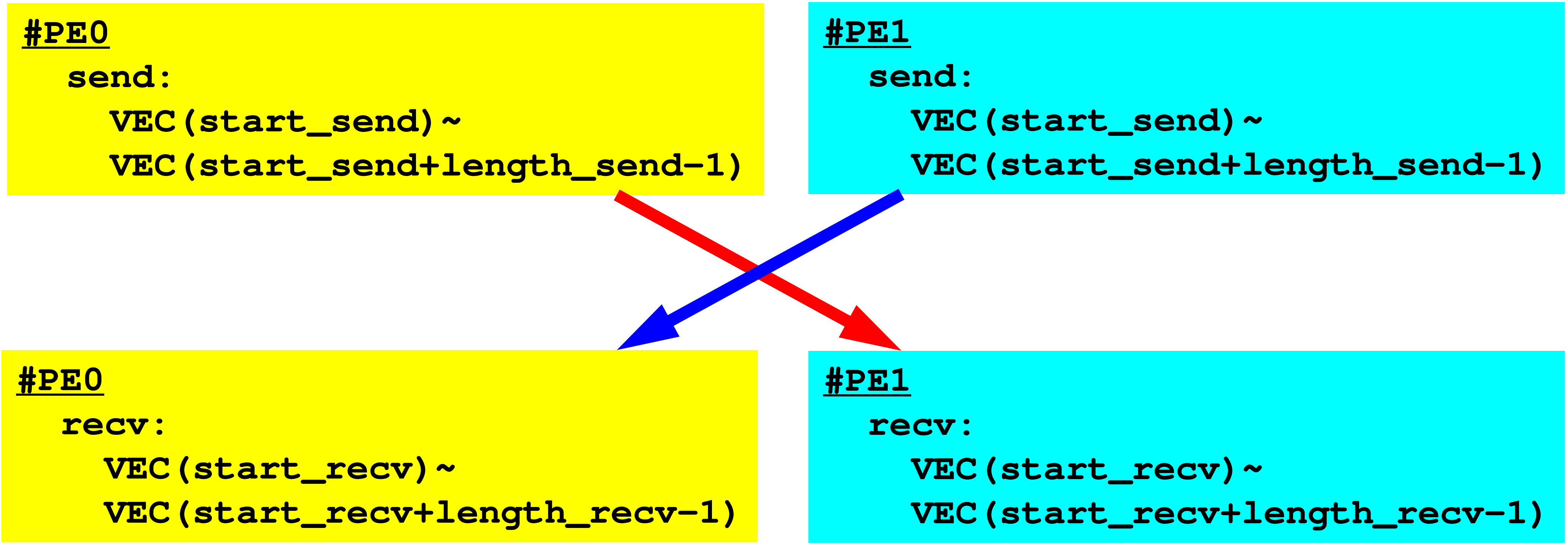
Ex.2: Send-Recv an Array (3/3)

```
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,…)
```

This is “SMPD” !!

t1

Notice: Send/Recv Arrays



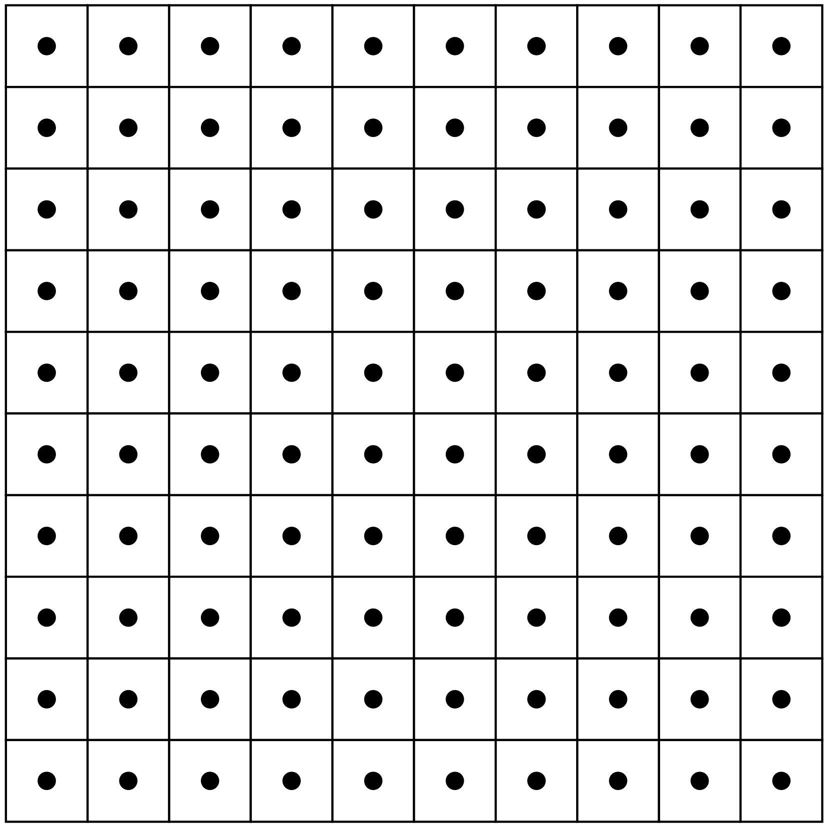
- “length_send” of sending process must be equal to “length_recv” of receiving process.
  - PE#0 to PE#1, PE#1 to PE#0
- “sendbuf” and “recvbuf”: different address

Point-to-Point Communication

- What is PtoP Communication ?
- 2D Problem, Generalized Communication Table
  - 2D FDM
  - Problem Setting
  - Distributed Local Data and Communication Table
  - Implementation
- Report S2

2D FDM (1/5)

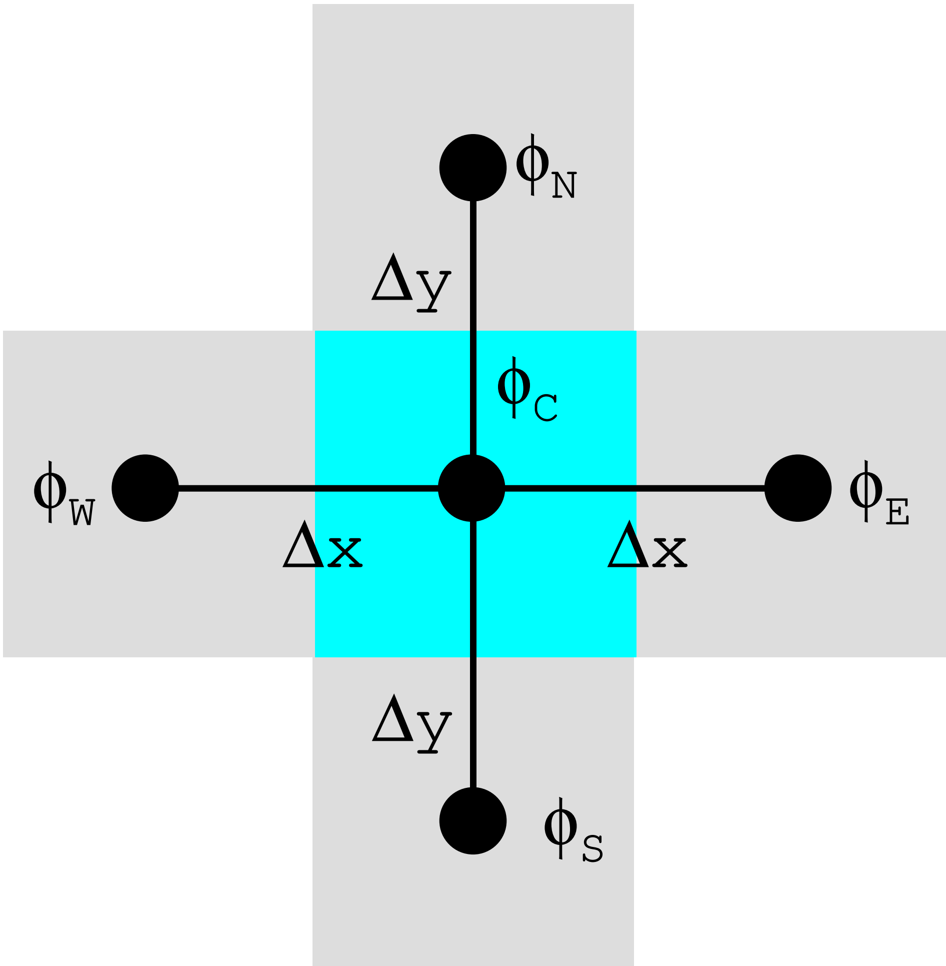
Entire Mesh



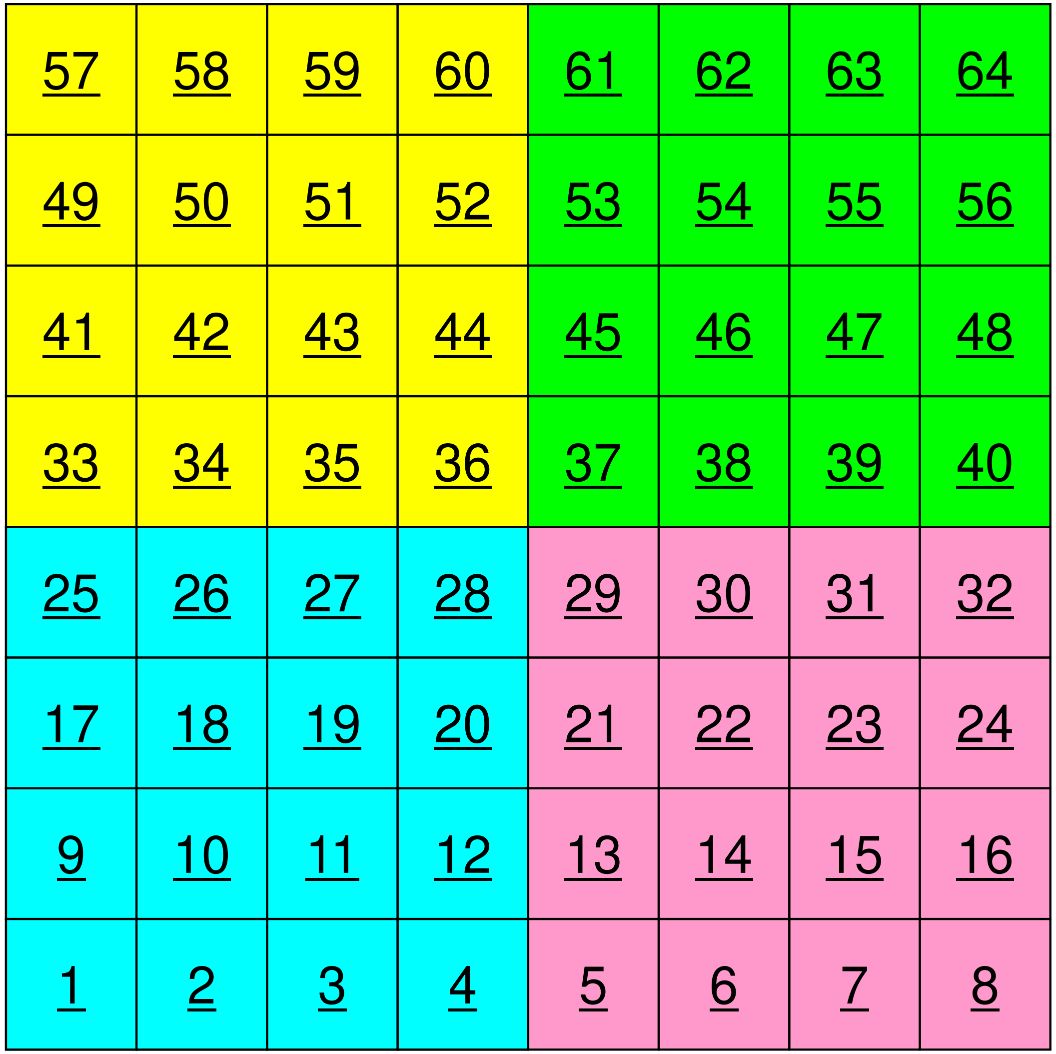
2D FDM (5-point, central difference)

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

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



Decompose into 4 domains



4 domains: Global ID

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

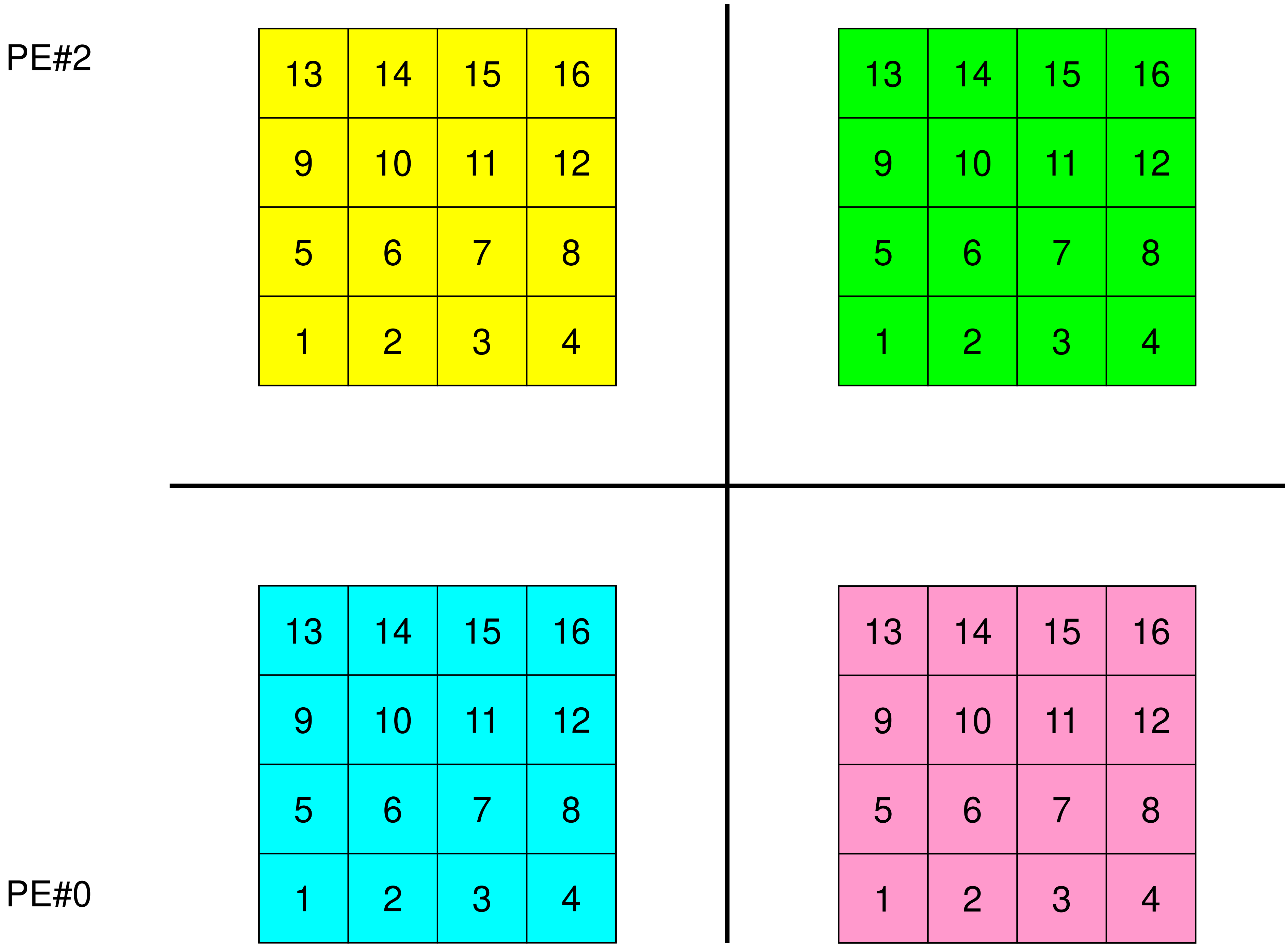
PE#0

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

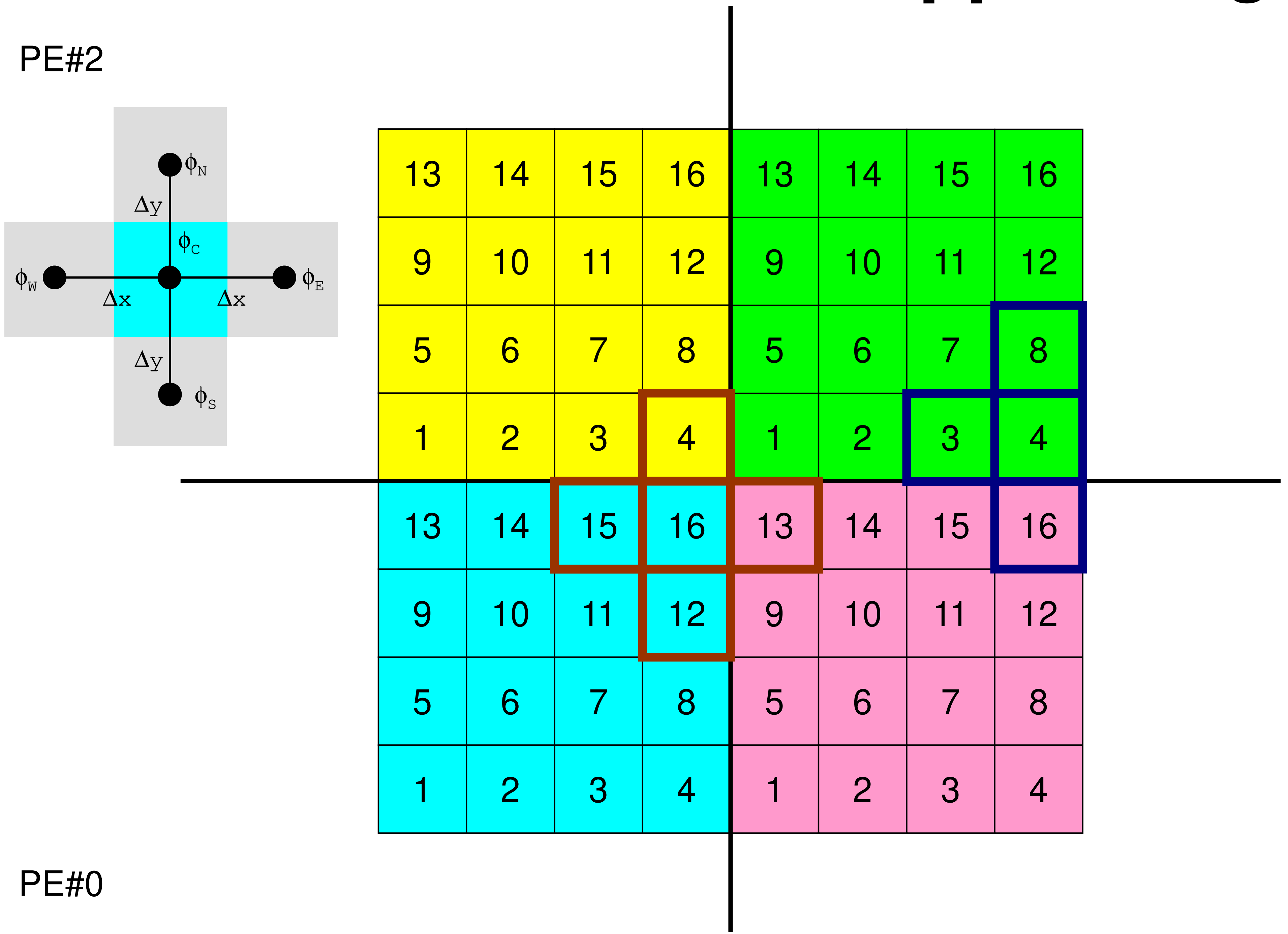
PE#1

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

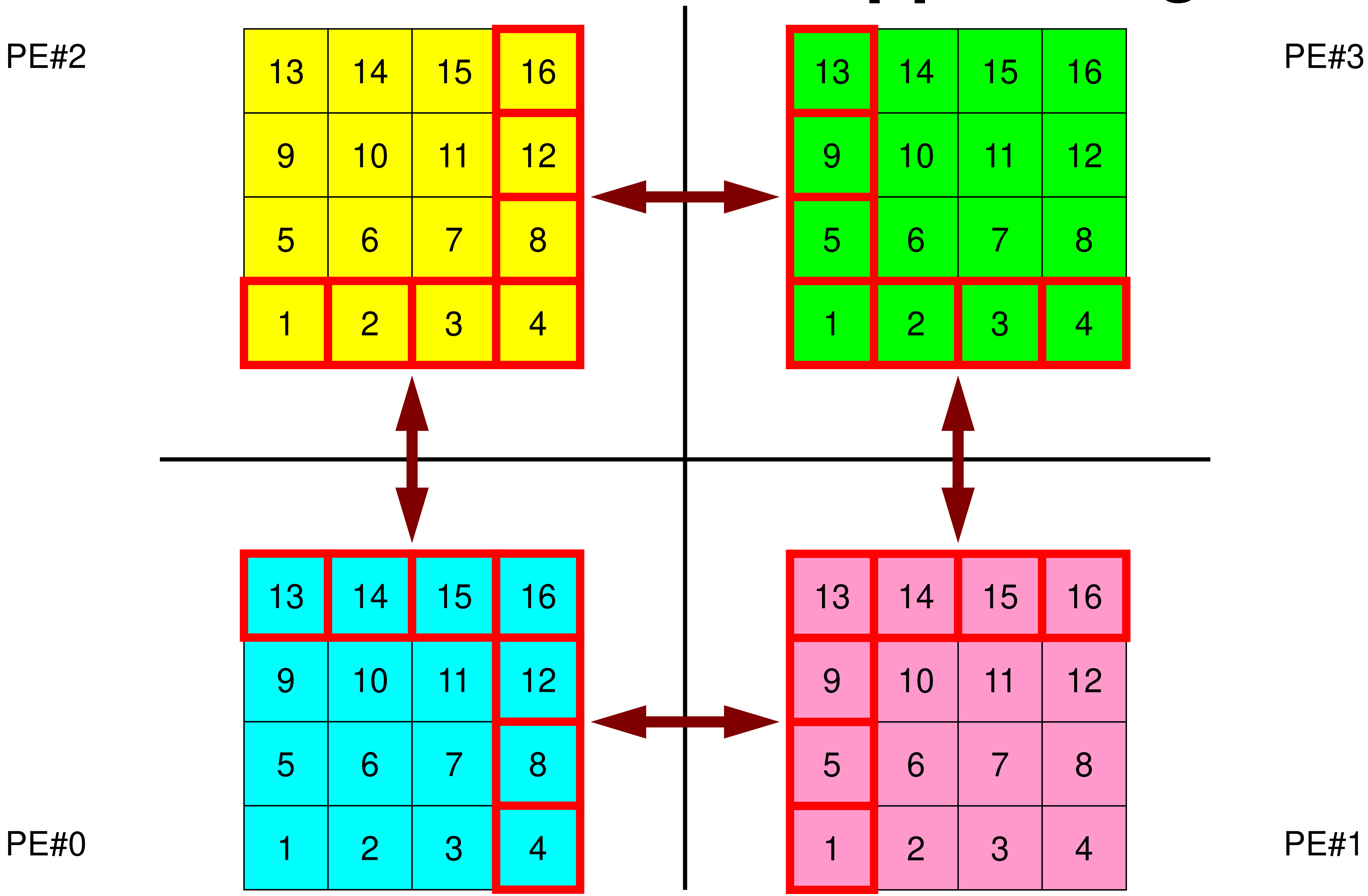
4 domains: Local ID



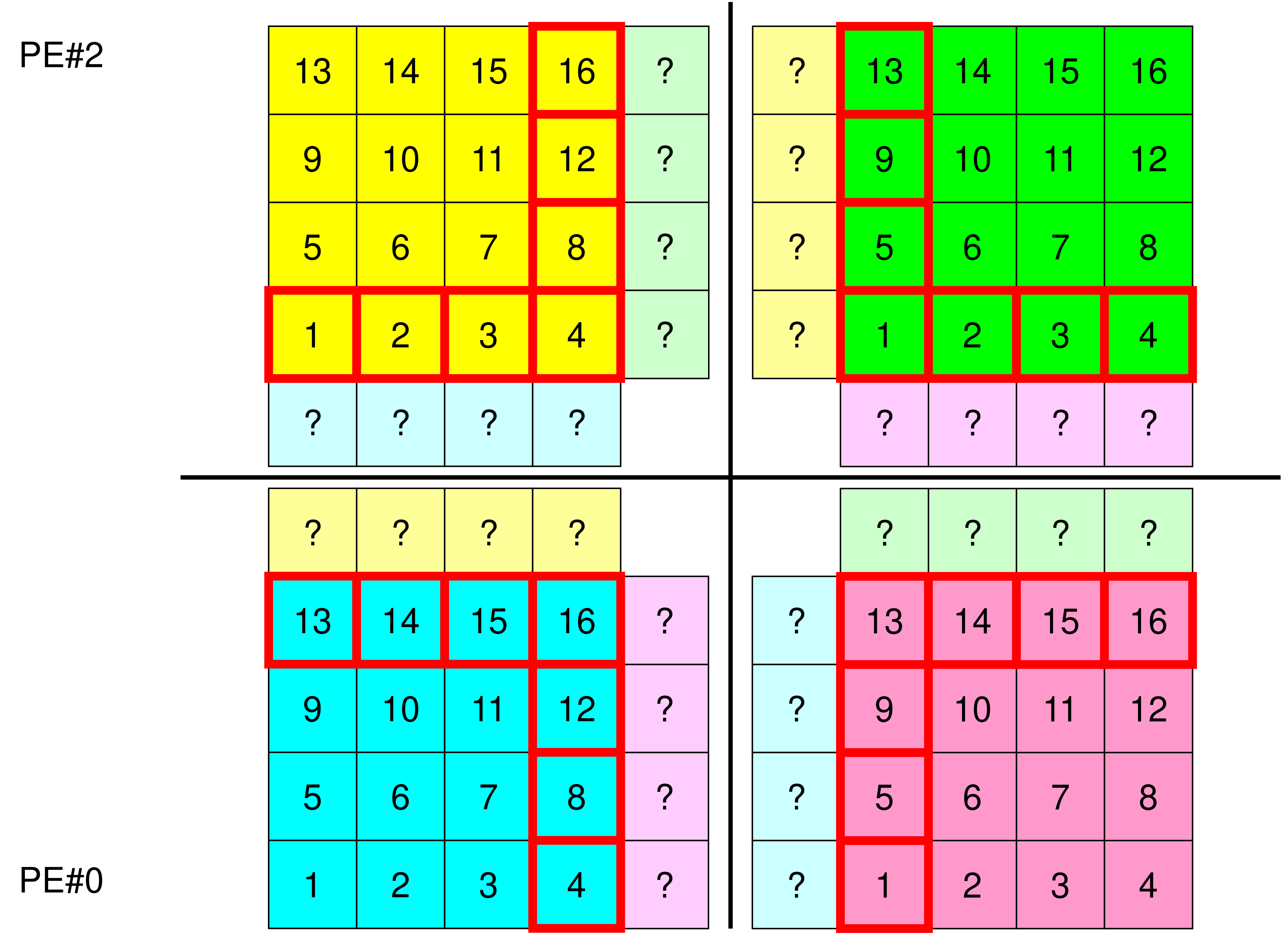
External Points: Overlapped Region



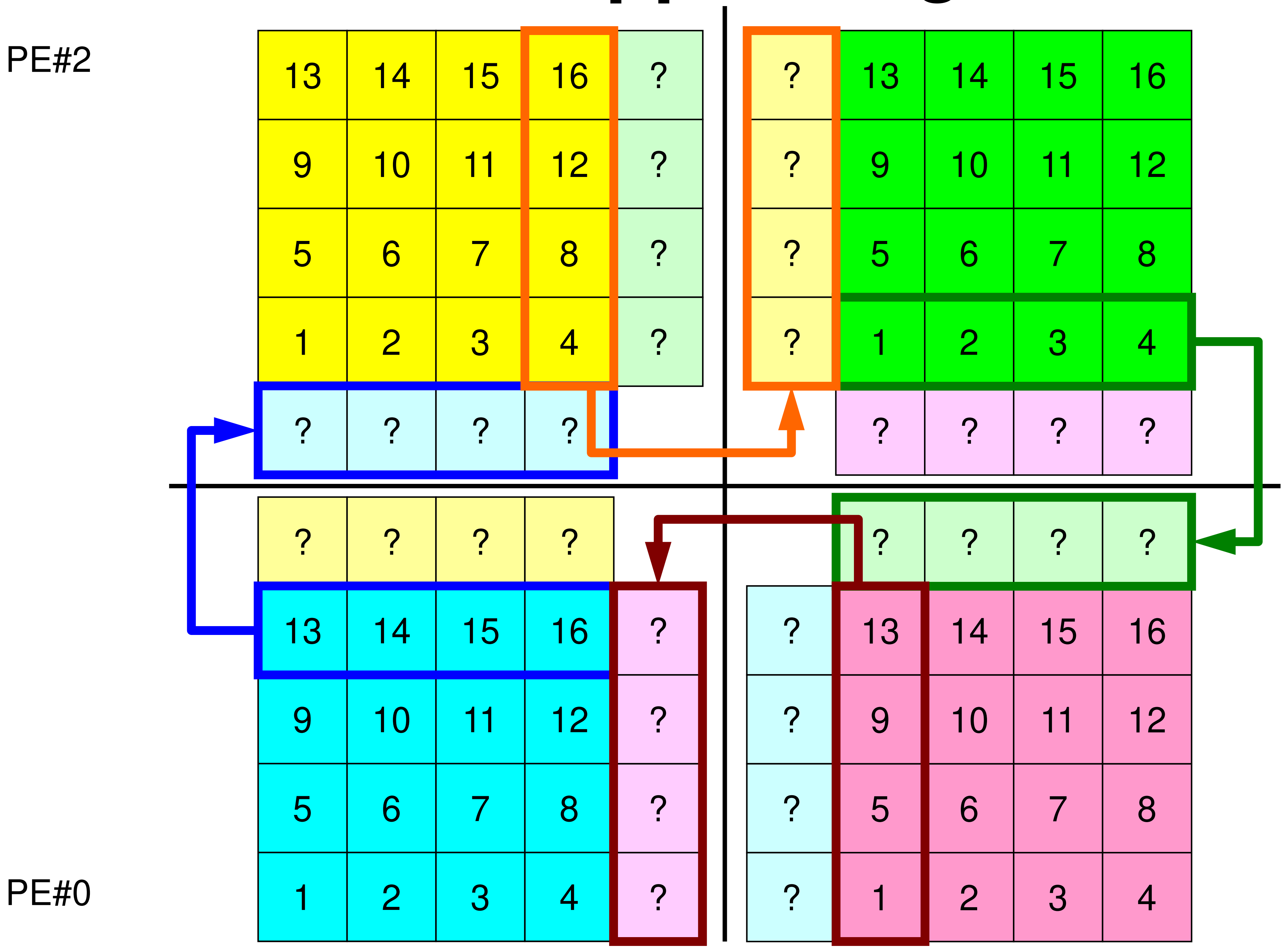
External Points: Overlapped Region



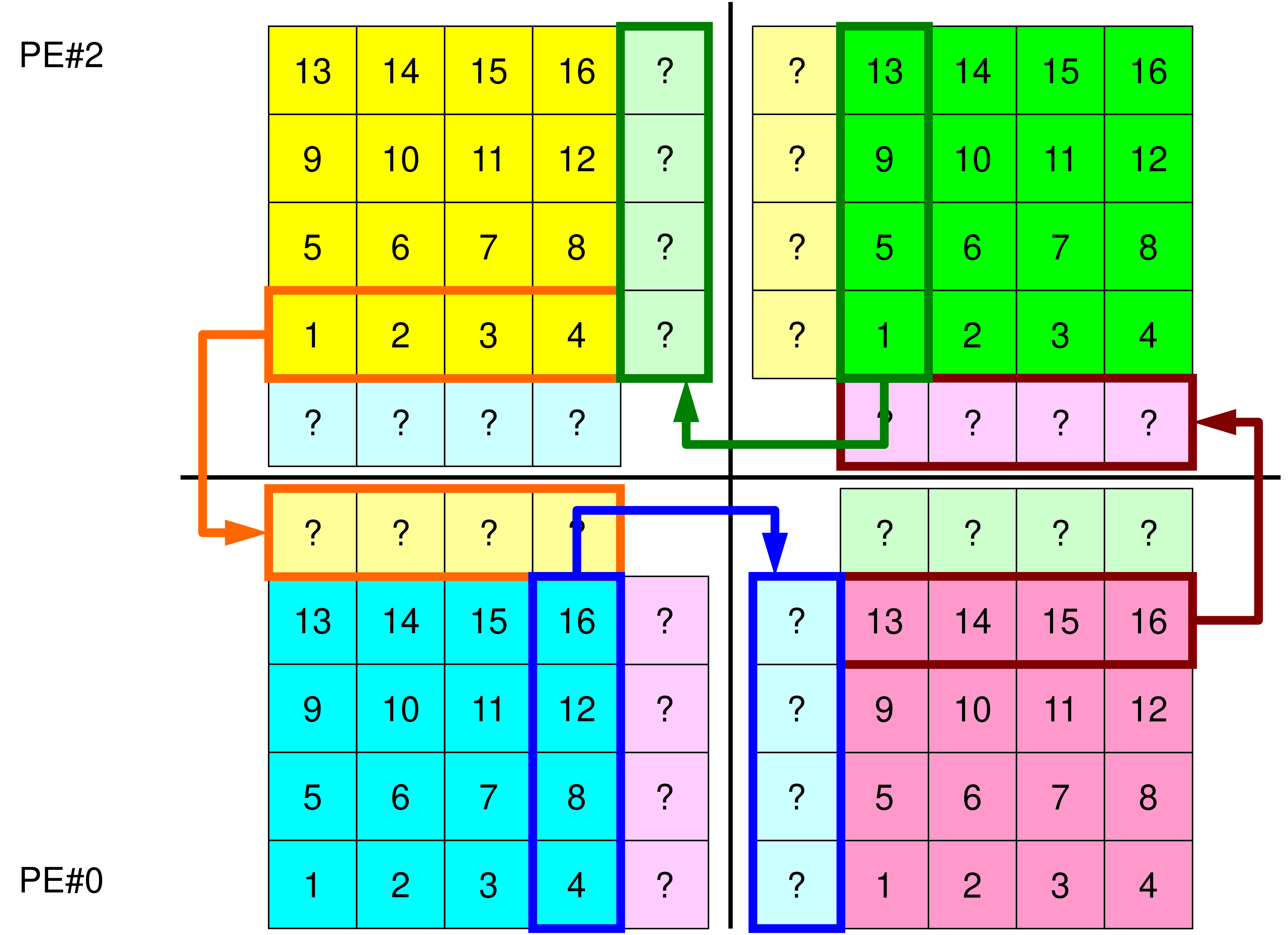
Local ID of External Points ?



Overlapped Region



Overlapped Region



Point-to-Point Communication

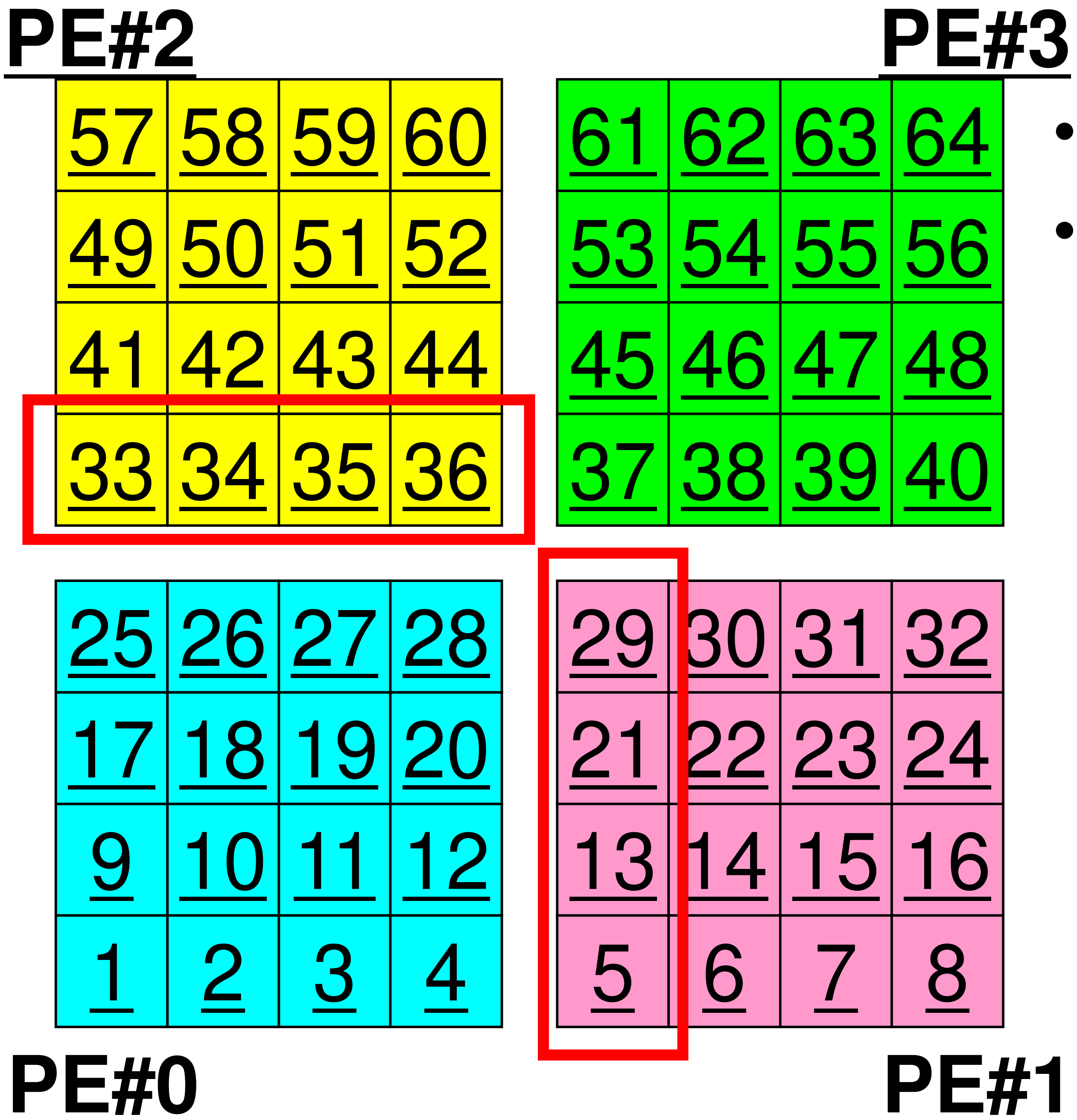
- What is PtoP Communication ?
- 2D Problem, Generalized Communication Table
  - 2D FDM
  - Problem Setting
  - Distributed Local Data and Communication Table
  - Implementation
- Report S2

Problem Setting: 2D FDM

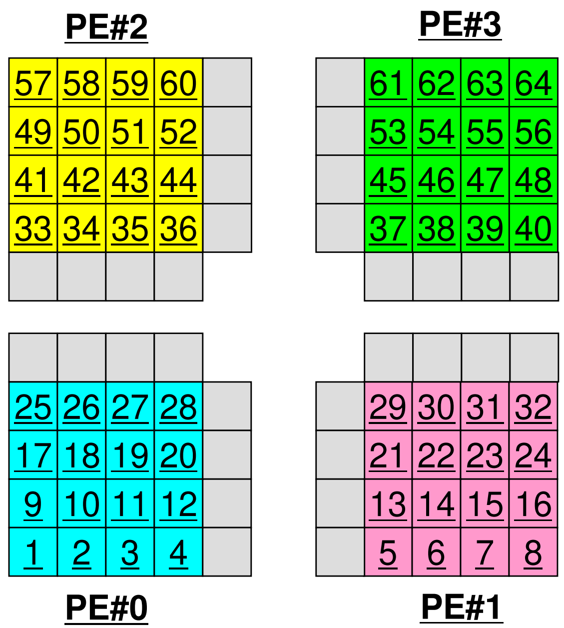
57	58	59	60	61	62	63	64
49	50	51	52	53	54	55	56
41	42	43	44	45	46	47	48
33	34	35	36	37	38	39	40
25	26	27	28	29	30	31	32
17	18	19	20	21	22	23	24
9	10	11	12	13	14	15	16
1	2	3	4	5	6	7	8

- 2D region with 64 meshes (8x8)
- Each mesh has global ID from 1 to 64
  - In this example, this global ID is considered as dependent variable, such as temperature, pressure etc.
  - Something like computed results

Problem Setting: Distributed Local Data



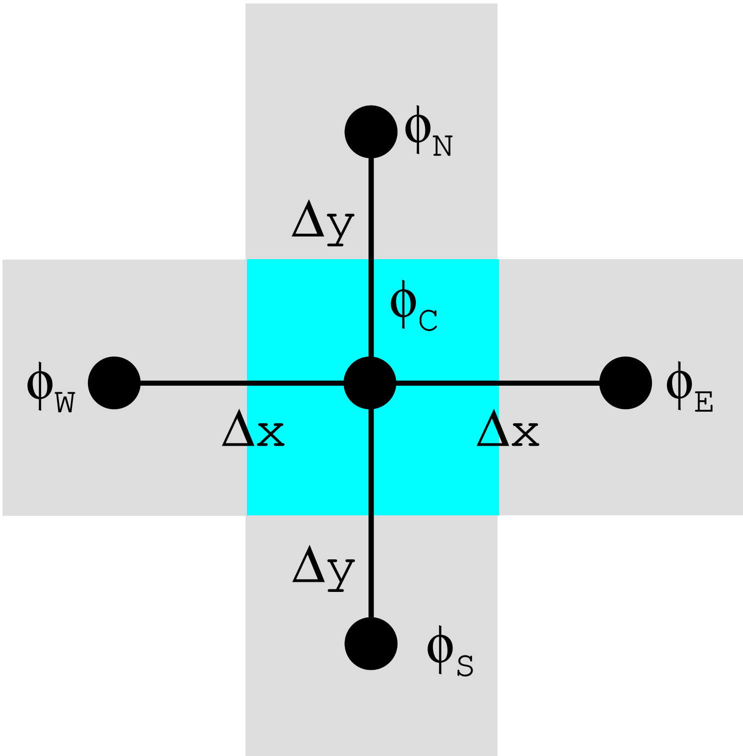
- 4 sub-domains.
- Info. of external points (global ID of mesh) is received from neighbors.
  - PE#0 receives □



Operations of 2D FDM

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

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

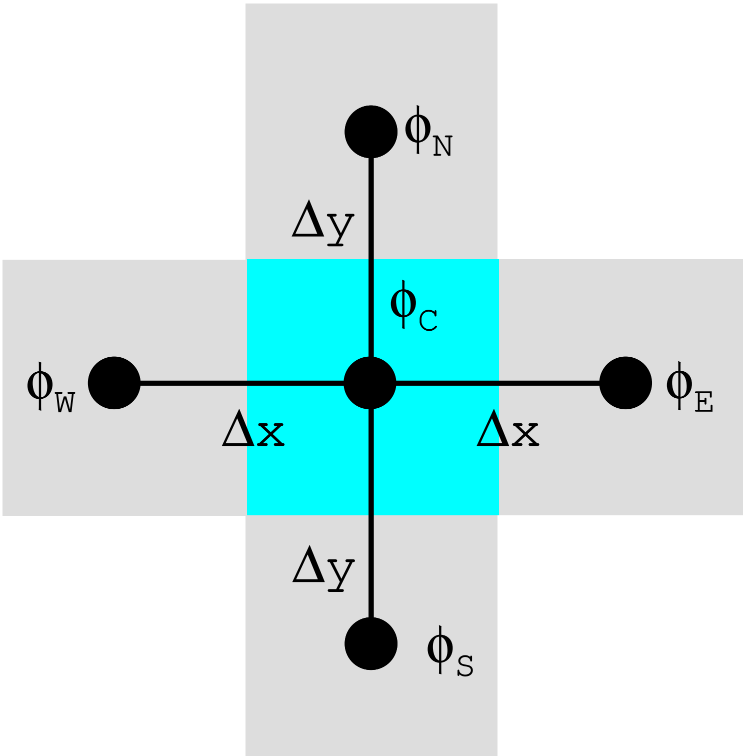


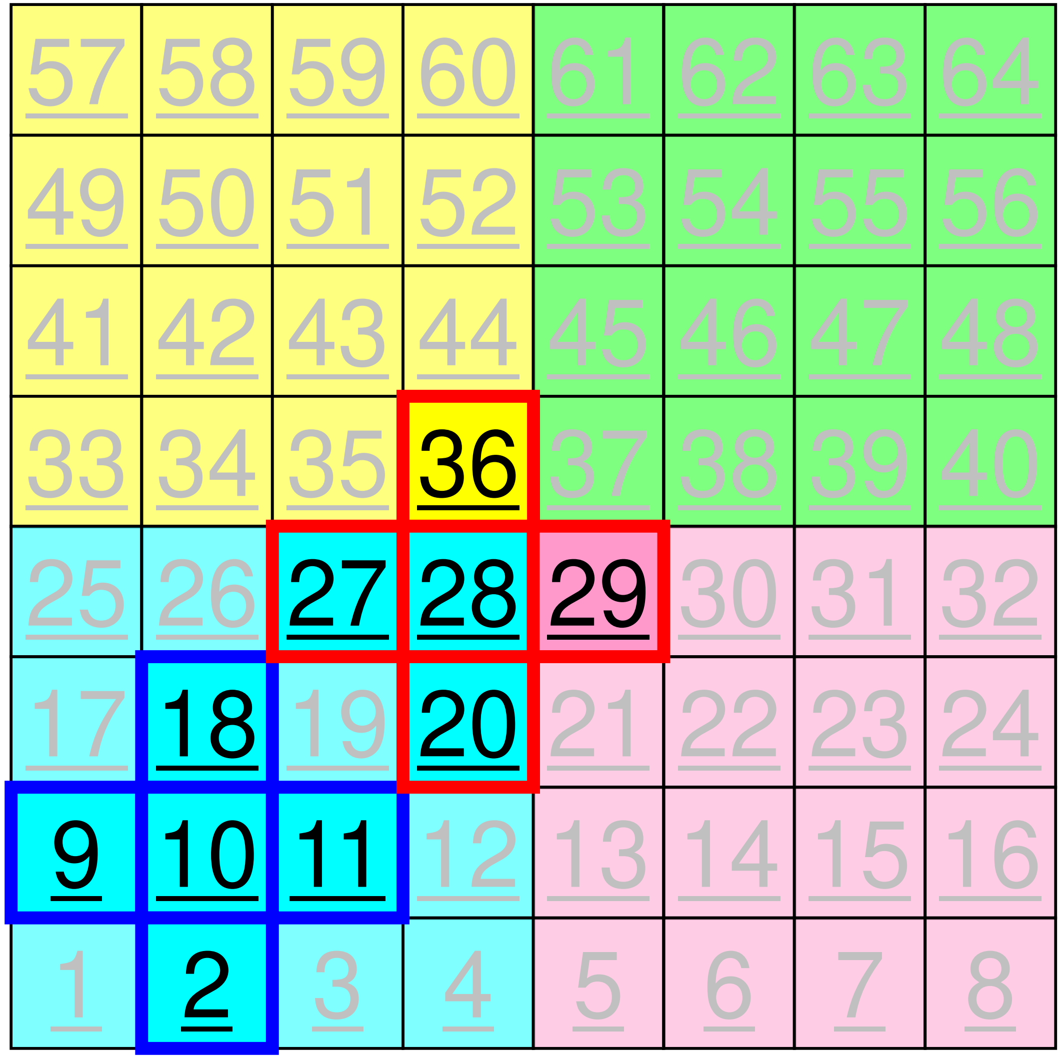
57	58	59	60	61	62	63	64
49	50	51	52	53	54	55	56
41	42	43	44	45	46	47	48
33	34	35	36	37	38	39	40
25	26	27	28	29	30	31	32
17	18	19	20	21	22	23	24
9	10	11	12	13	14	15	16
1	2	3	4	5	6	7	8

Operations of 2D FDM

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

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





Computation (1/3)

PE#2	57	58	59	60	61	62	63	64	PE#3
	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	
PE#0	1	2	3	4	5	6	7	8	PE#1

- On each PE, info. of internal pts (i=1-N(=16)) are read from distributed local data, info. of boundary pts are sent to neighbors, and they are received as info. of external pts.

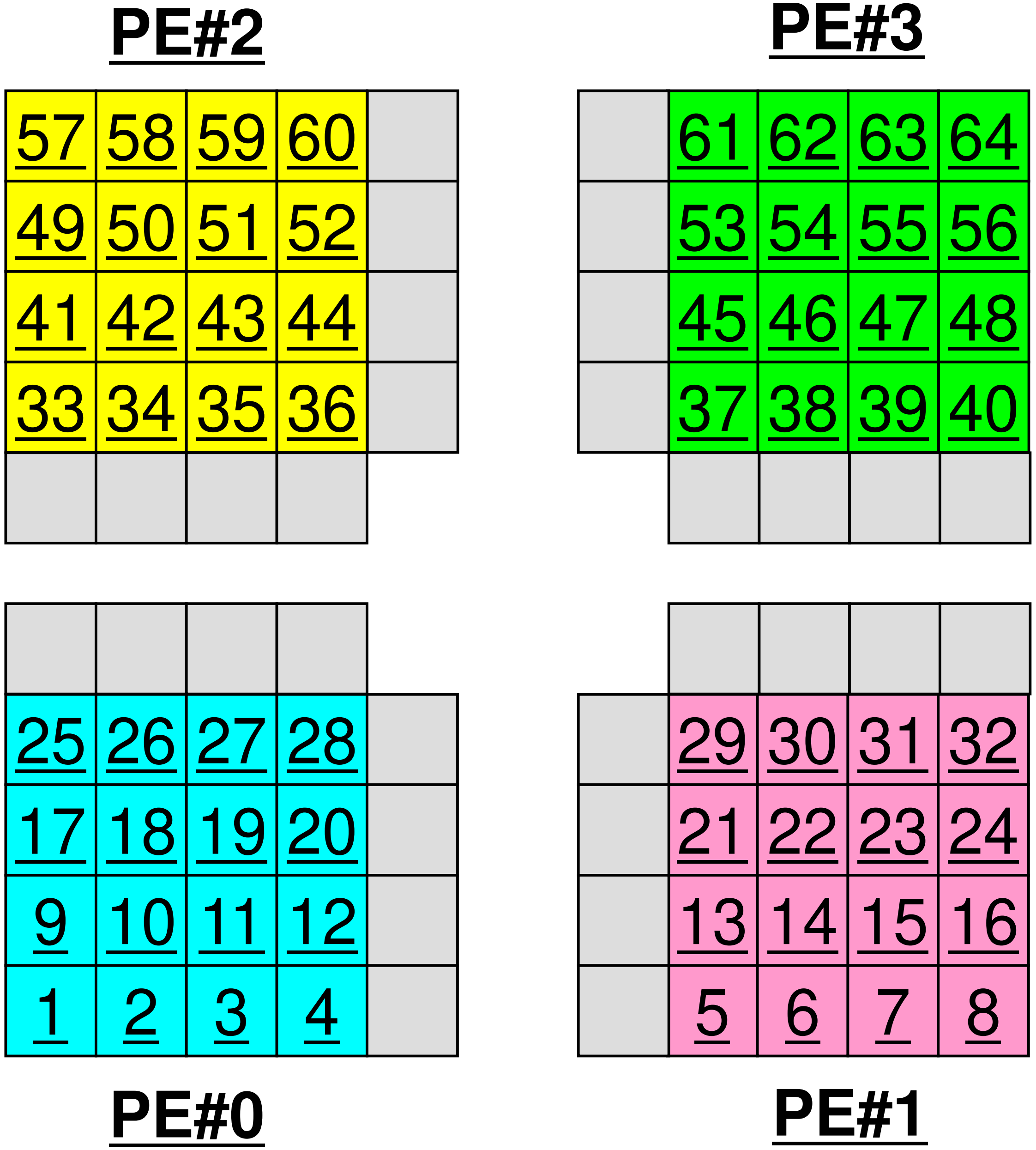
Computation (2/3): Before Send/Recv

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: ?



PE#0

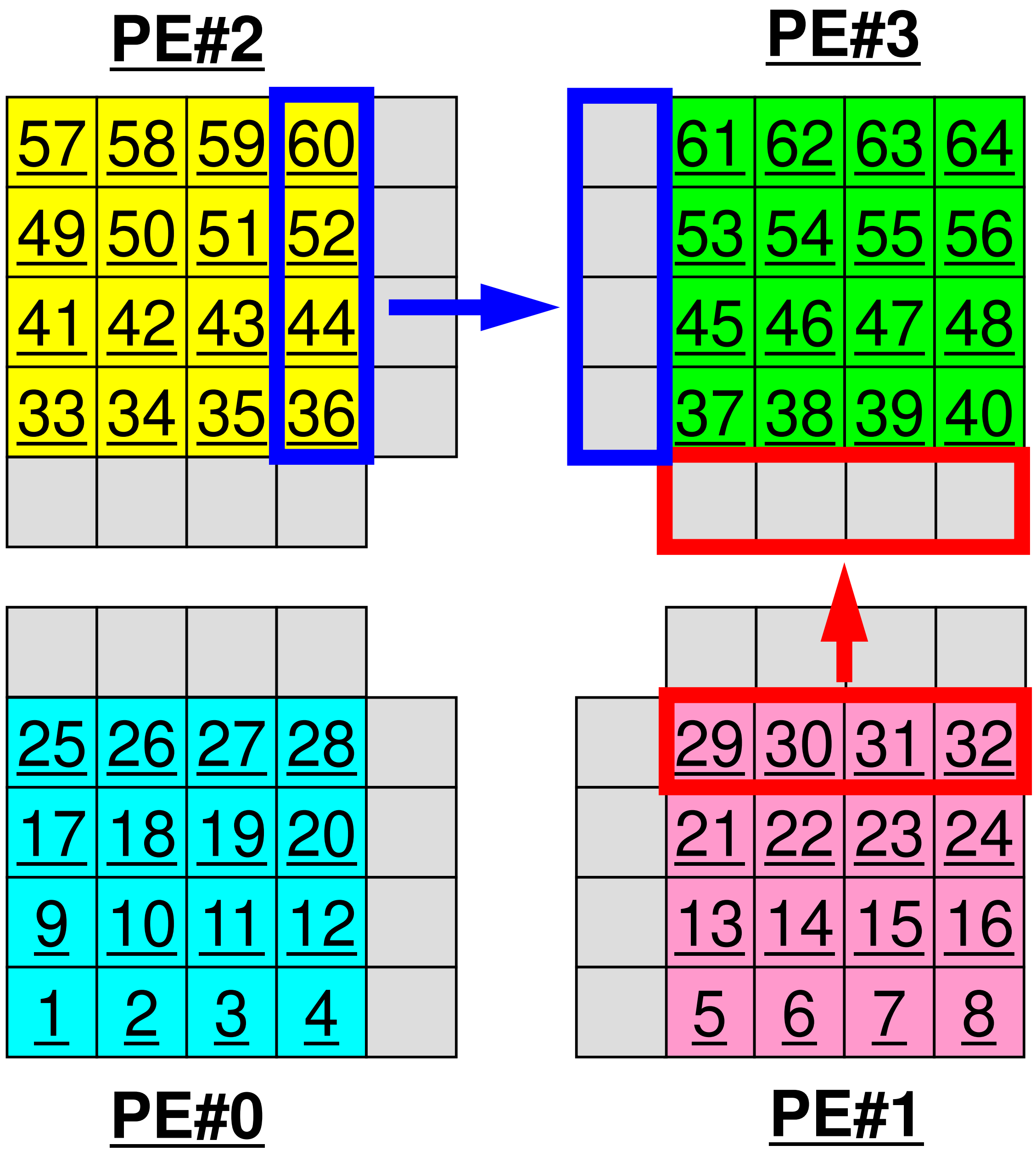
1: 1	9: 17	17: ?
2: 2	10: 18	18: ?
3: 3	11: 19	19: ?
4: 4	12: 20	20: ?
5: 9	13: 25	21: ?
6: 10	14: 26	22: ?
7: 11	15: 27	23: ?
8: 12	16: 28	24: ?

PE#1

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: ?

Computation (2/3): Before Send/Recv

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: ?



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: ?

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: ?

Computation (3/3): After Send/Recv

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

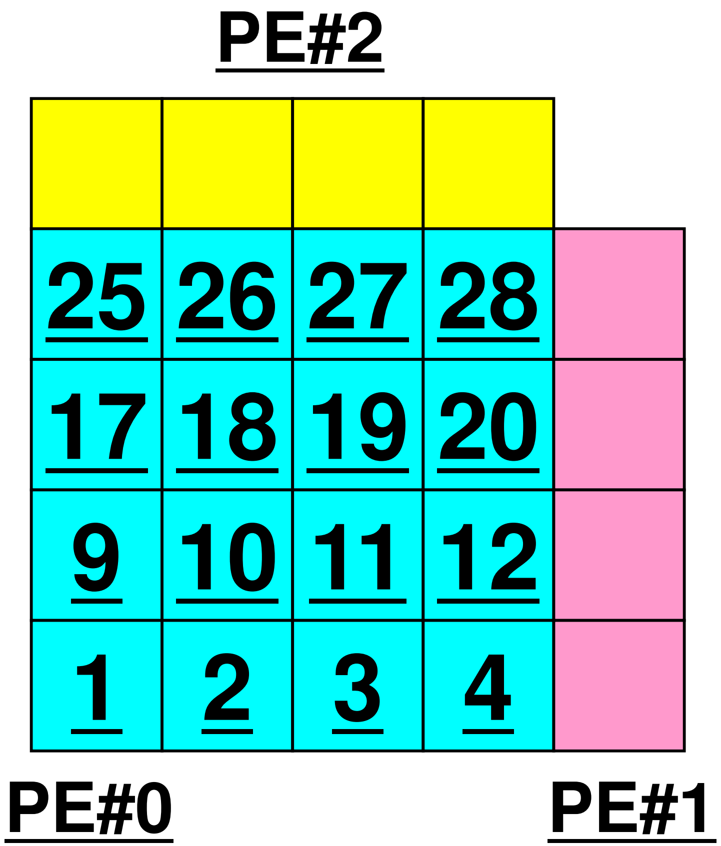
1: 5	9: 21	17: 4
2: 6	10: 22	18: 12
3: 7	11: 23	19: 20
4: 8	12: 24	20: 28
5: 13	13: 29	21: 37
6: 14	14: 30	22: 38
7: 15	15: 31	23: 39
8: 16	16: 32	24: 40

Peer-to-Peer Communication

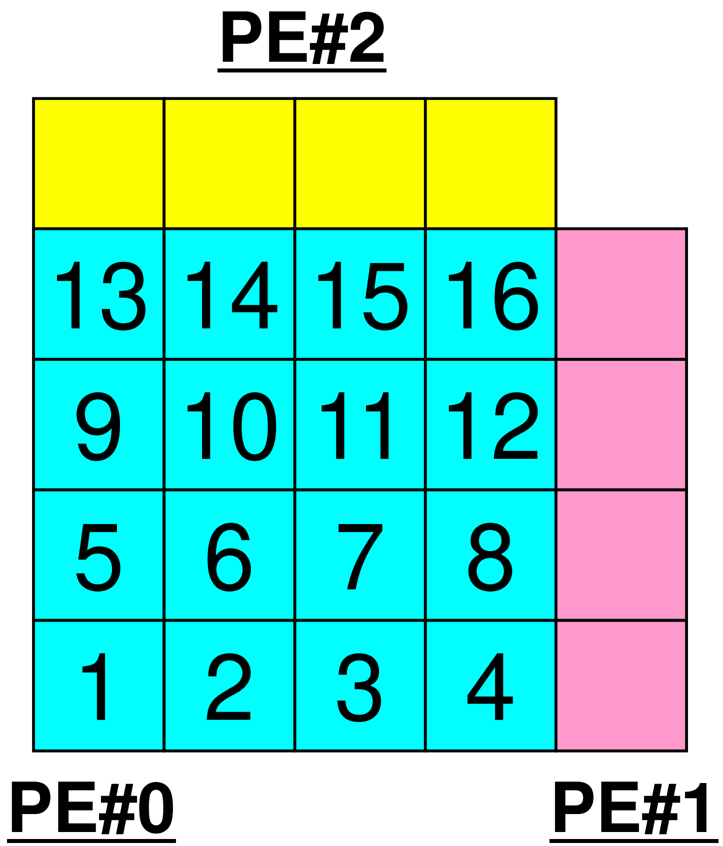
- What is P2P Communication ?
- 2D Problem, Generalized Communication Table
  - 2D FDM
  - Problem Setting
  - Distributed Local Data and Communication Table
  - Implementation
- Report S2

Overview of Distributed Local Data

Example on PE#0

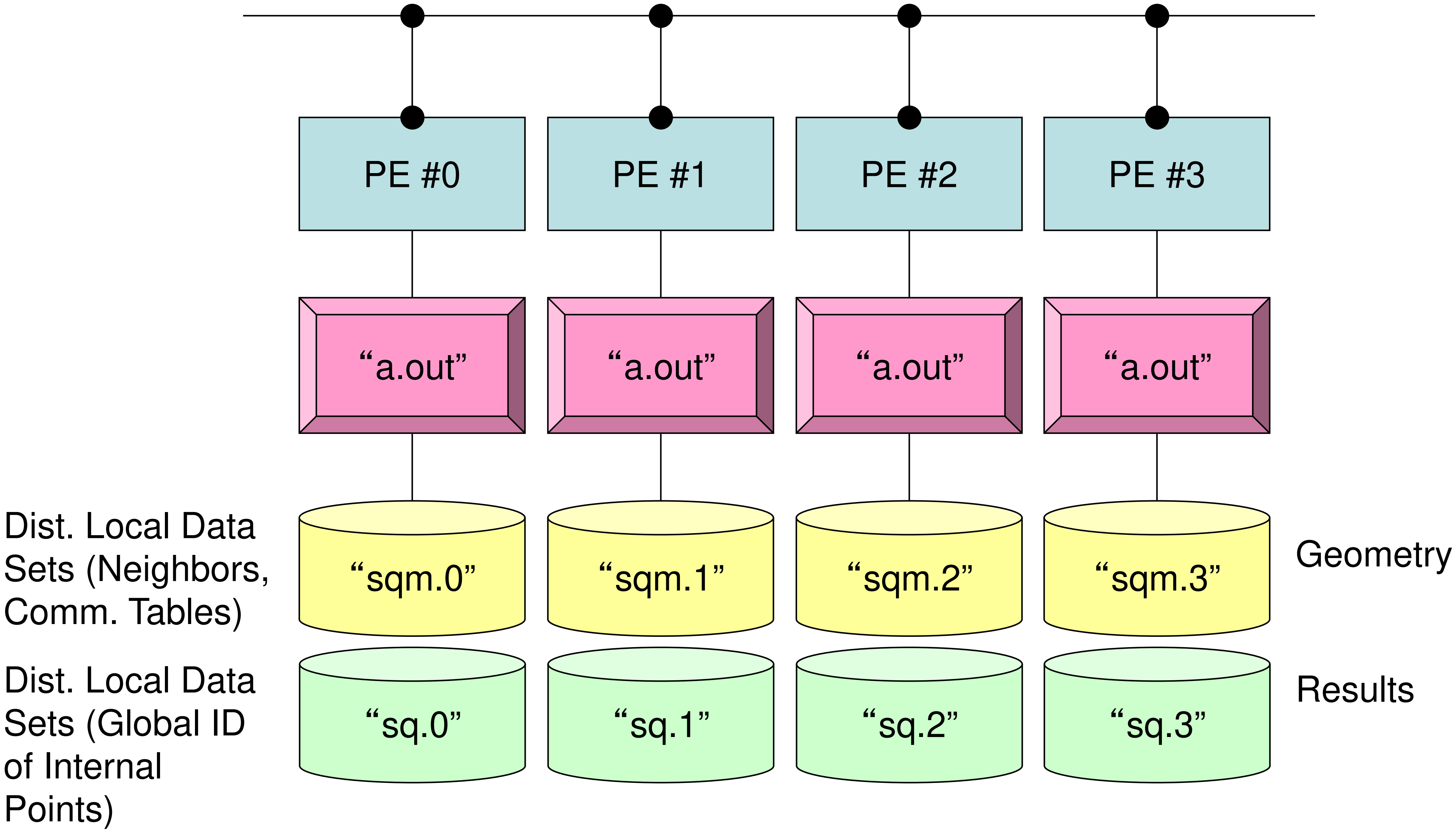


Value at each mesh (= Global ID)



Local ID

SPMD･･･

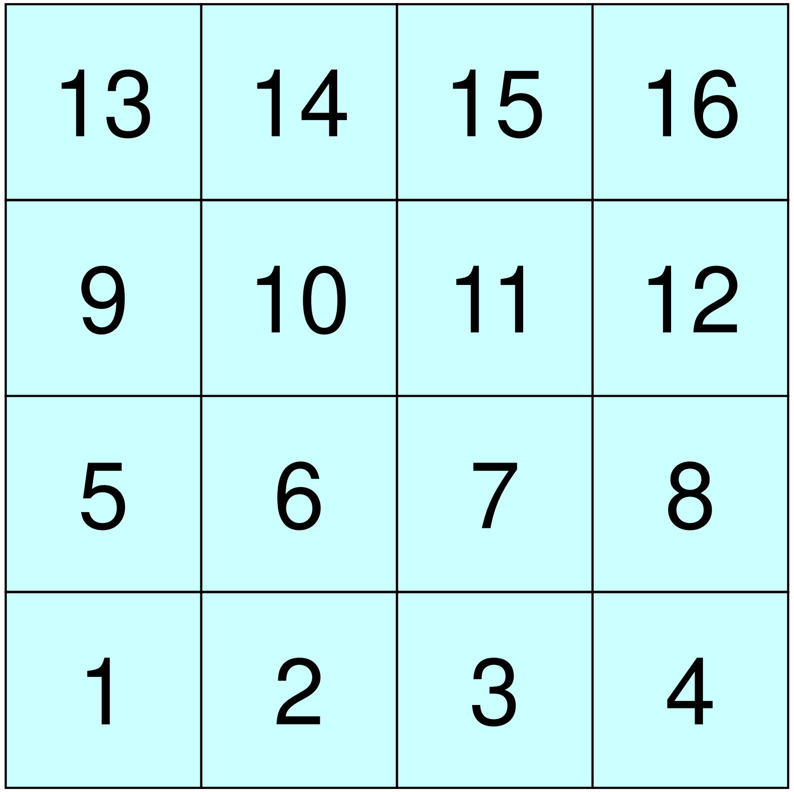

PE #0
PE #1
PE #2
PE #3
“a.out”
“a.out”
“a.out”
“a.out”
Dist. Local Data Sets (Neighbors, Comm. Tables)
“sqm.0”
“sqm.1”
“sqm.2”
“sqm.3”
Geometry
Dist. Local Data Sets (Global ID of Internal Points)
“sq.0”
“sq.1”
“sq.2”
“sq.3”
Results

2D FDM: PE#0

Information at each domain (1/4)

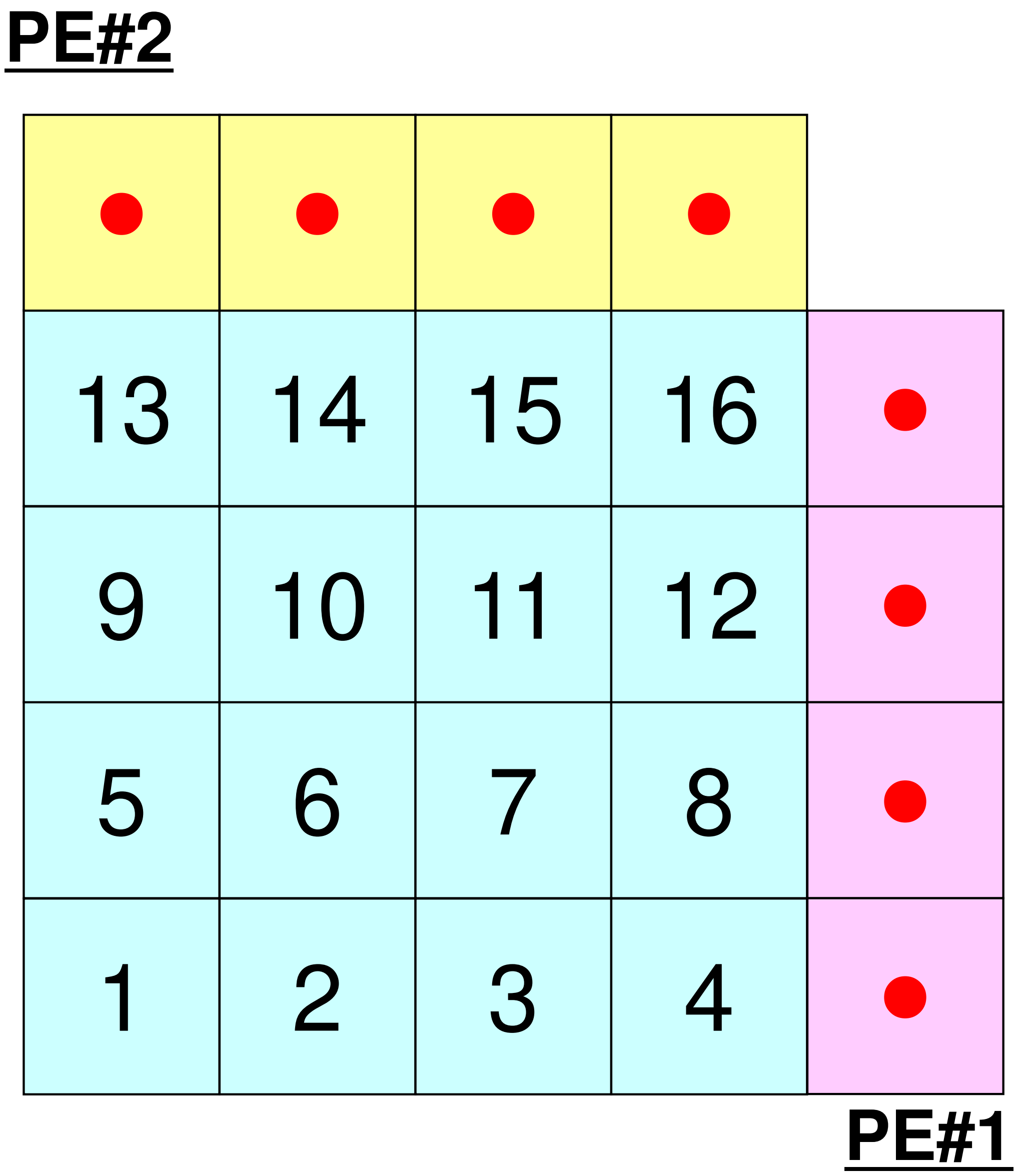
Internal Points

Meshes originally assigned to the domain



2D FDM: PE#0

Information at each domain (2/4)



Internal Points

Meshes originally assigned to the domain

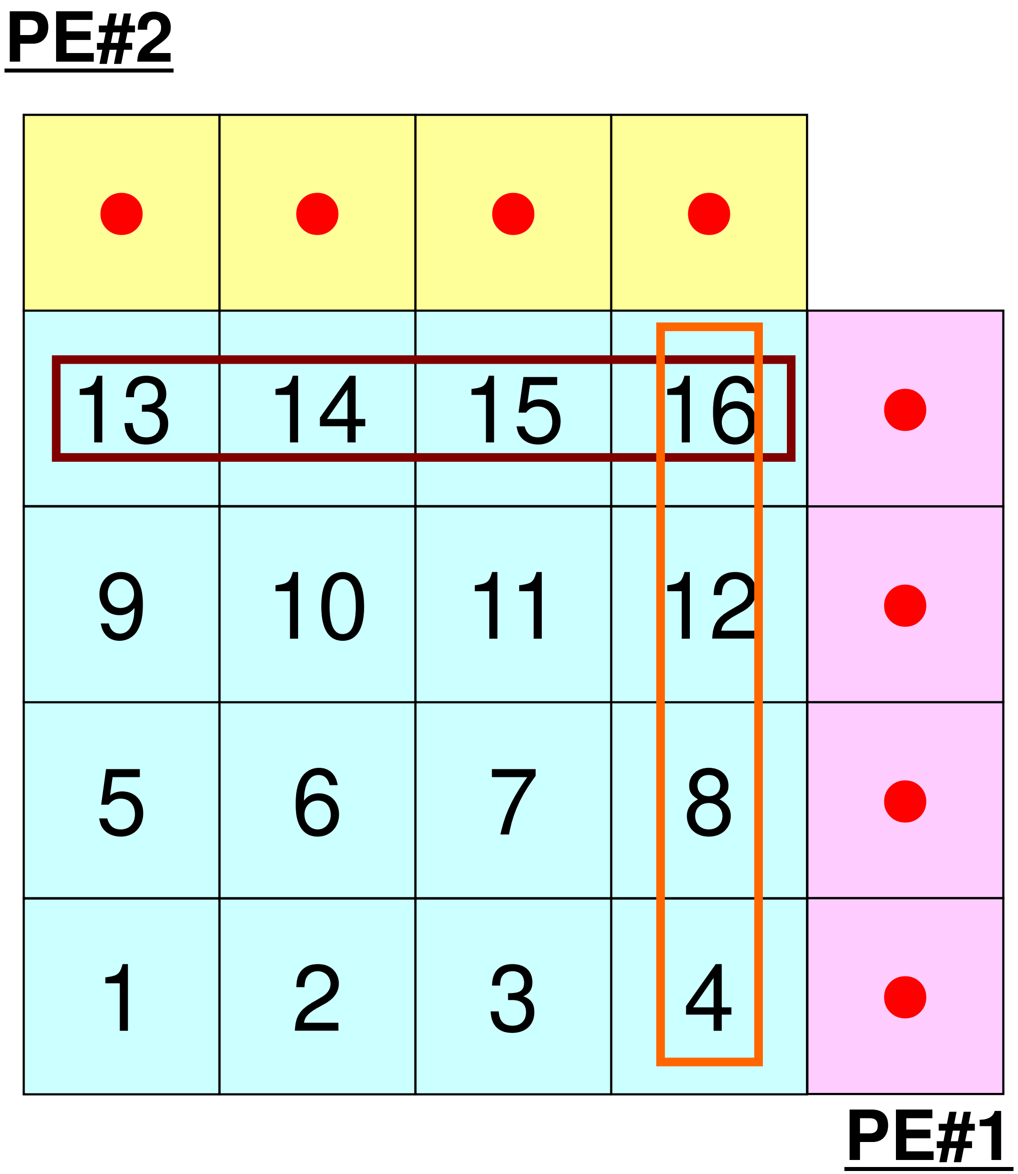
External Points

Meshes originally assigned to different domain, but required for computation of meshes in the domain (meshes in overlapped regions)

- Sleeves
- Halo

2D FDM: PE#0

Information at each domain (3/4)



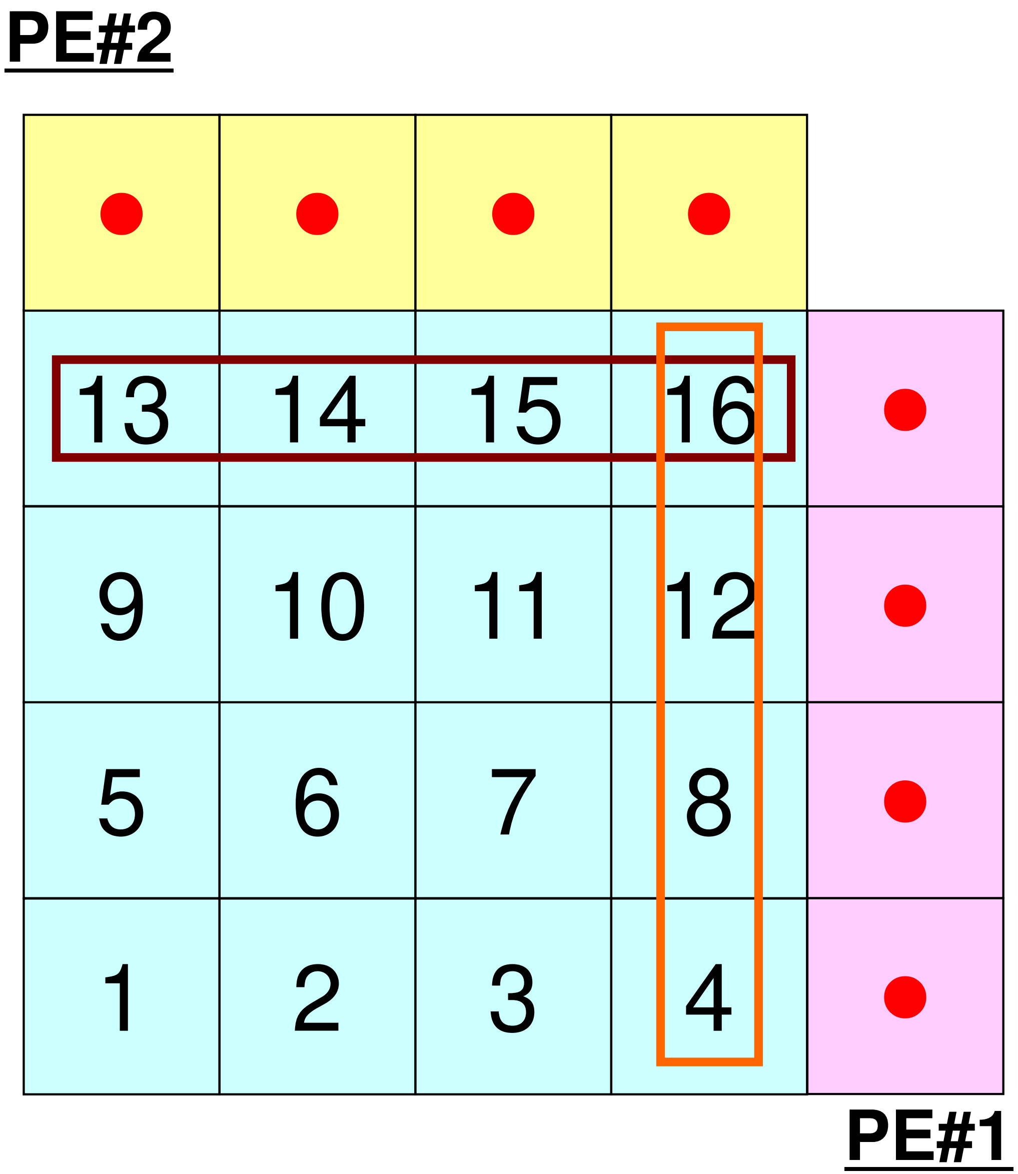
Internal Points
Meshes originally assigned to the domain

External Points
Meshes originally assigned to different domain, but required for computation of meshes in the domain (meshes in overlapped regions)

Boundary Points
Internal points, which are also external points of other domains (used in computations of meshes in other domains)

2D FDM: PE#0

Information at each domain (4/4)



Internal Points
Meshes originally assigned to the domain

External Points
Meshes originally assigned to different domain, but required for computation of meshes in the domain (meshes in overlapped regions)

Boundary Points
Internal points, which are also external points of other domains (used in computations of meshes in other domains)

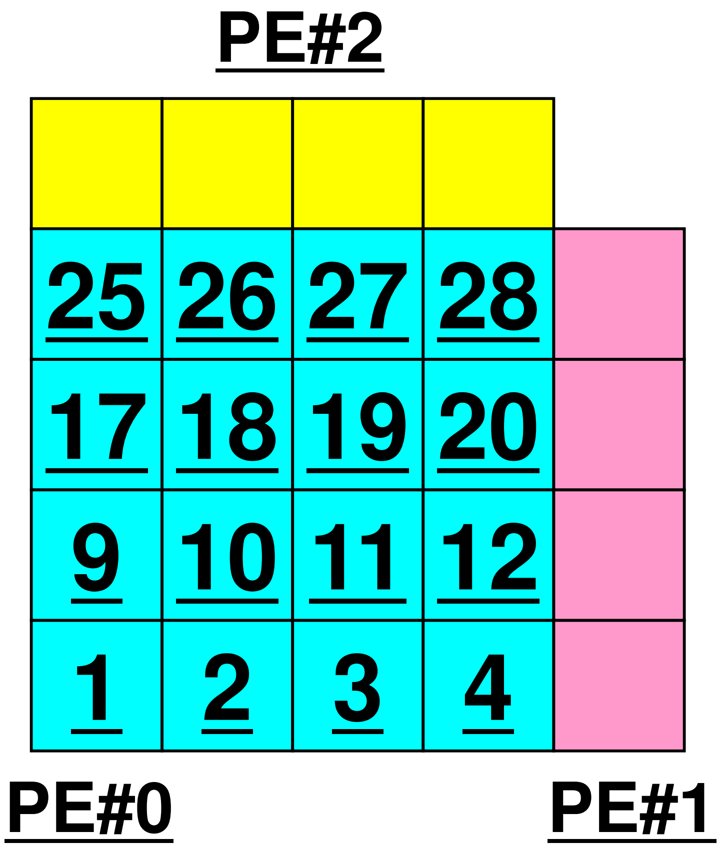
Relationships between Domains
Communication Table: External/Boundary Points
Neighbors

Description of Distributed Local Data

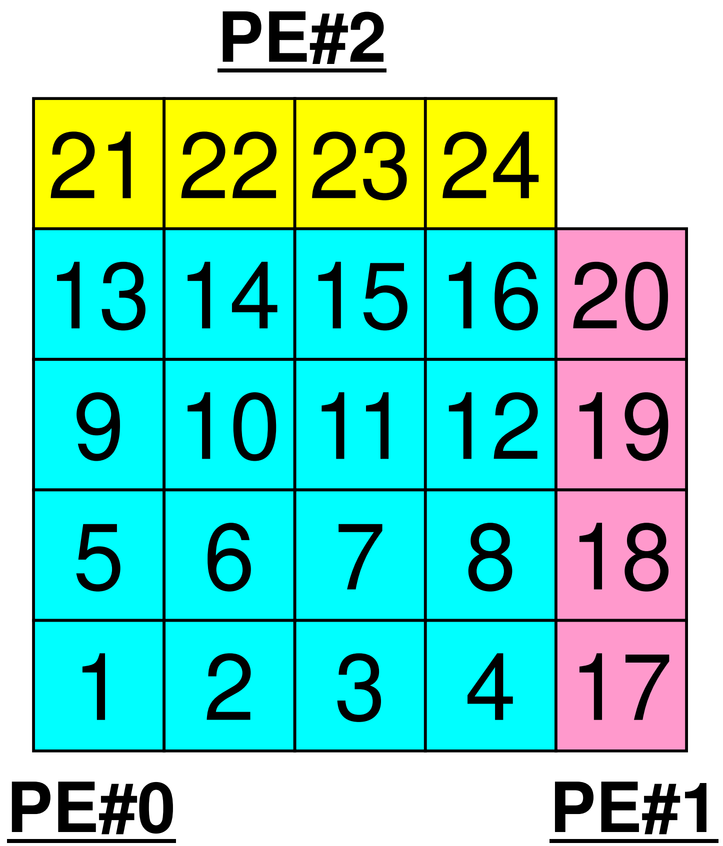
- Internal/External Points
  - Numbering: Starting from internal pts, then external pts after that
- Neighbors
  - Shares overlapped meshes
  - Number and ID of neighbors
- Import Table (Receive)
  - From where, how many, and which external points are received/imported ?
- Export Table (Send)
  - To where, how many and which boundary points are sent/exported ?

Overview of Distributed Local Data

Example on PE#0



Value at each mesh (= Global ID)



Local ID

Generalized Comm. Table: Send

1	1.1 ①	2.4 ②,1	3.2 ⑤,2		
2	3.6 ②	4.3 ①,3	2.5 ④,4	3.7 ⑥,5	9.1 ⑧,6
3	5.7 ③	1.5 ⑤,7	3.1 ⑦,8		
4	9.8 ④	4.1 ②,9	2.5 ⑤,10	2.7 ⑥,11	
5	11.5 ⑤	3.1 ①,12	9.5 ②,13	10.4 ③,14	4.3 ⑦,15
6	12.4 ⑥	6.5 ③,16	9.5 ⑦,17		
7	23.1 ⑦	6.4 ②,18	2.5 ③,19	1.4 ⑥,20	13.1 ⑧,21
8	51.3 ⑧	9.5 ②,22	1.3 ③,23	9.6 ④,24	3.1 ⑥,25

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))

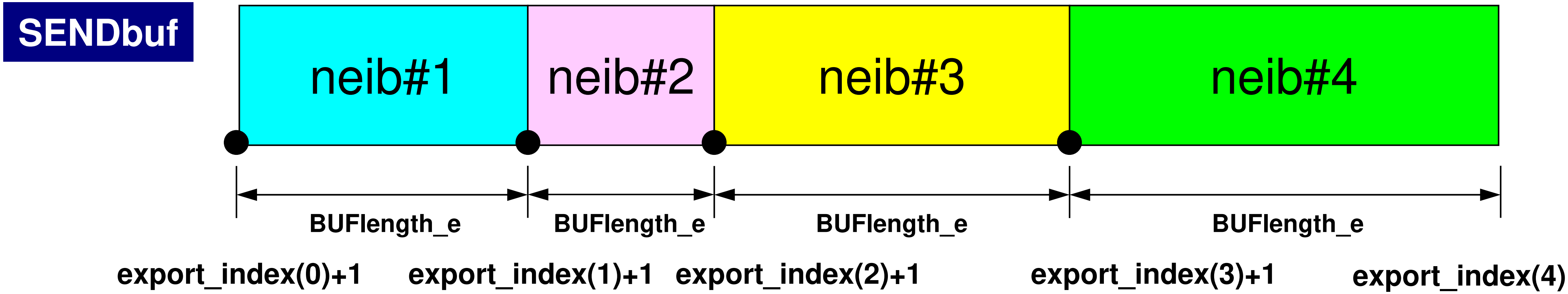
```
{Y}= [A]{X}

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
```

- Neighbors
  - NEIBPETOT, NEIBPE(NEIBPETOT)
- Message size for each neighbor
  - export_index(neib), neib= 0, NEIBPETOT
- ID of **boundary** points
  - export_item(k), k= 1, export_index(NEIBPETOT)
- Messages to each neighbor
  - SENDbuf(k), k= 1, export_index(NEIBPETOT)

Fortran

SEND: 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)
```

Copied to sending buffers

Generalized Comm. Table: Receive

- Neighbors
  - NEIBPETOT, NEIBPE(NEIBPETOT)
- Message size for each neighbor
  - import_index(neib), neib= 0, NEIBPETOT
- ID of **external** points
  - import_item(k), k= 1, import_index(NEIBPETOT)
- Messages from each neighbor
  - RECVbuf(k), k= 1, import_index(NEIBPETOT)

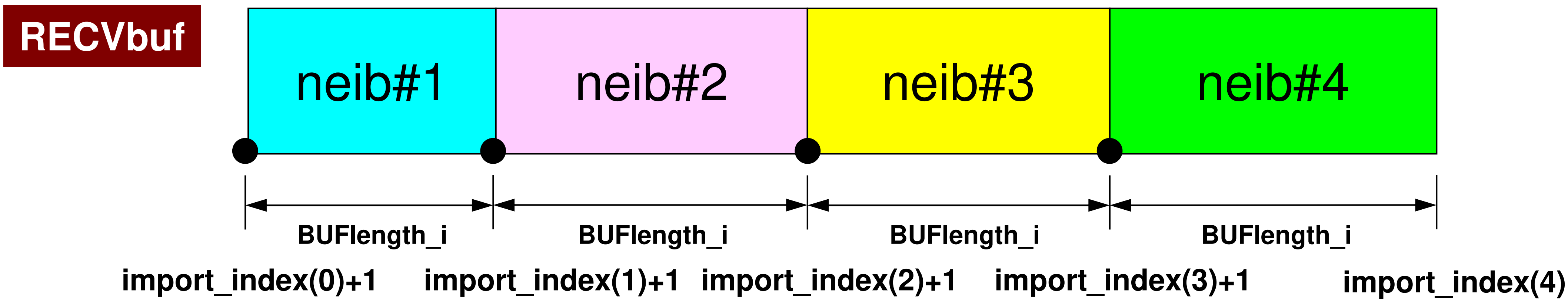
Fortran

RECV: 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)
     enddo
  enddo
```

Copied from receiving buffer



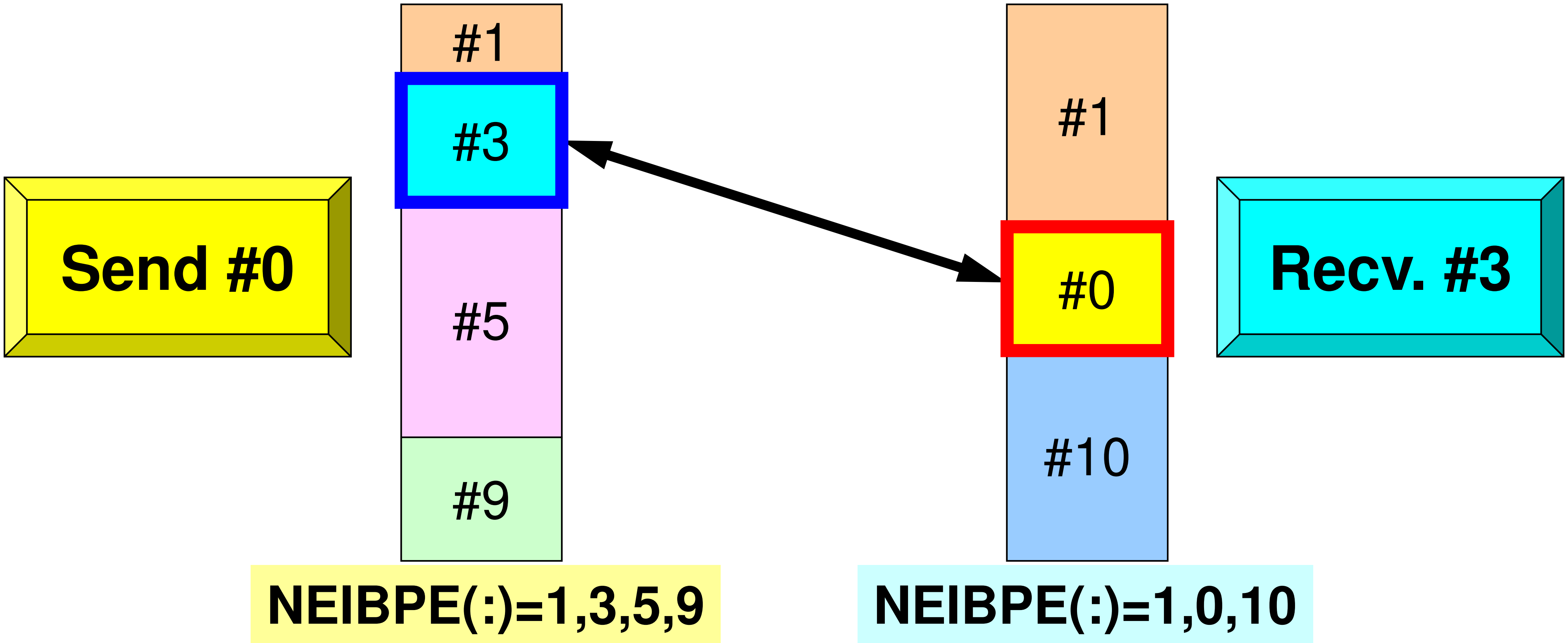
Relationship SEND/RECV

```
 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
```

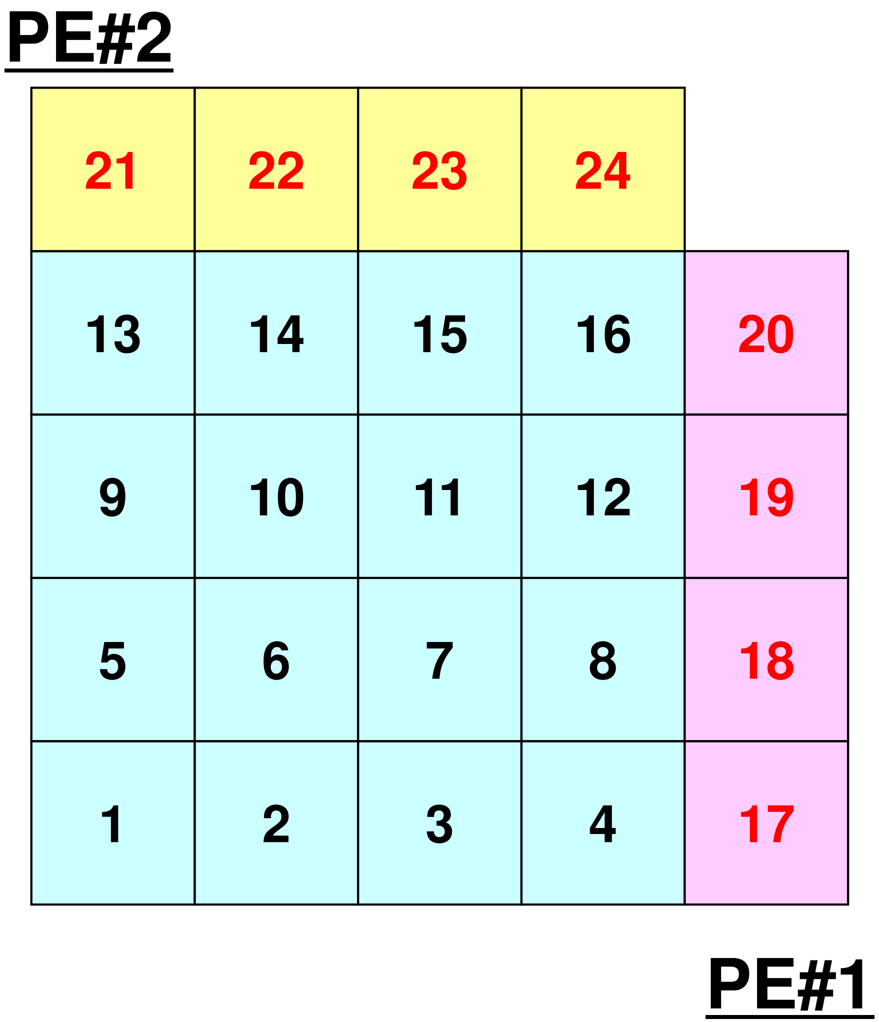
- Consistency of ID's of sources/destinations, size and contents of messages !
- Communication occurs when NEIBPE(neib) matches

Relationship SEND/RECV (#0 to #3)



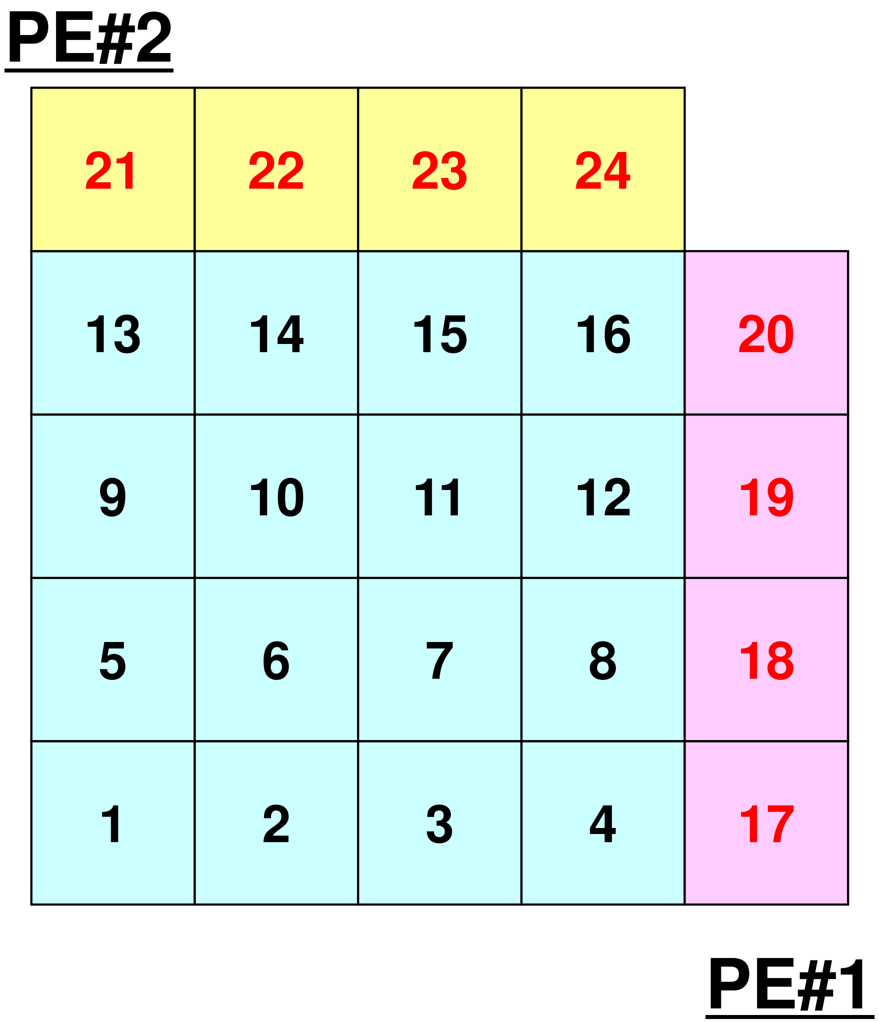
- Consistency of ID's of sources/destinations, size and contents of messages !
- Communication occurs when NEIBPE(neib) matches

Generalized Comm. Table (1/6)



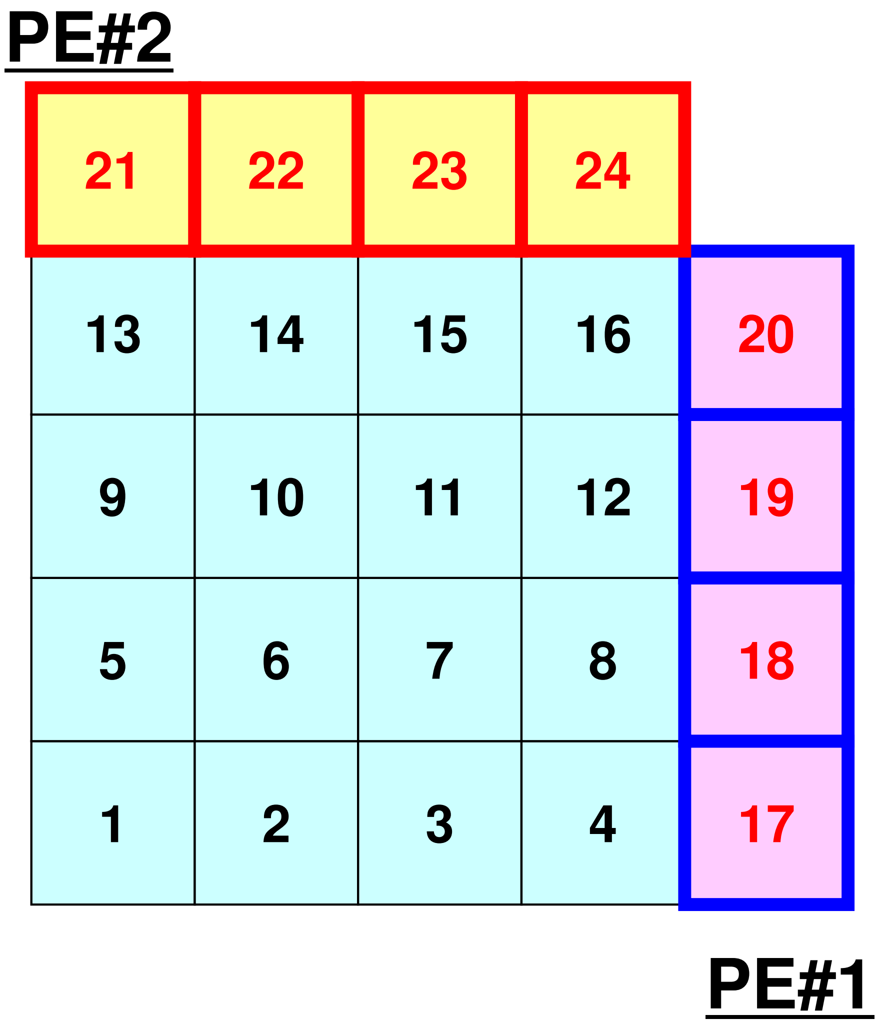

```
#NEIBPEtot
2
#NEIBPE
1  2
#NODE
24 16
#IMPORT_index
4   8
#IMPORT_items
17
18
19
20
21
22
23
24
#EXPORT_index
4   8
#EXPORT_items
4
8
12
16
13
14
15
16
```

Generalized Comm. Table (2/6)




```
#NEIBPEtot        Number of neighbors
2
#NEIBPE           ID of neighbors
1   2
#NODE
24 16             Ext/Int Pts, Int Pts
#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
```

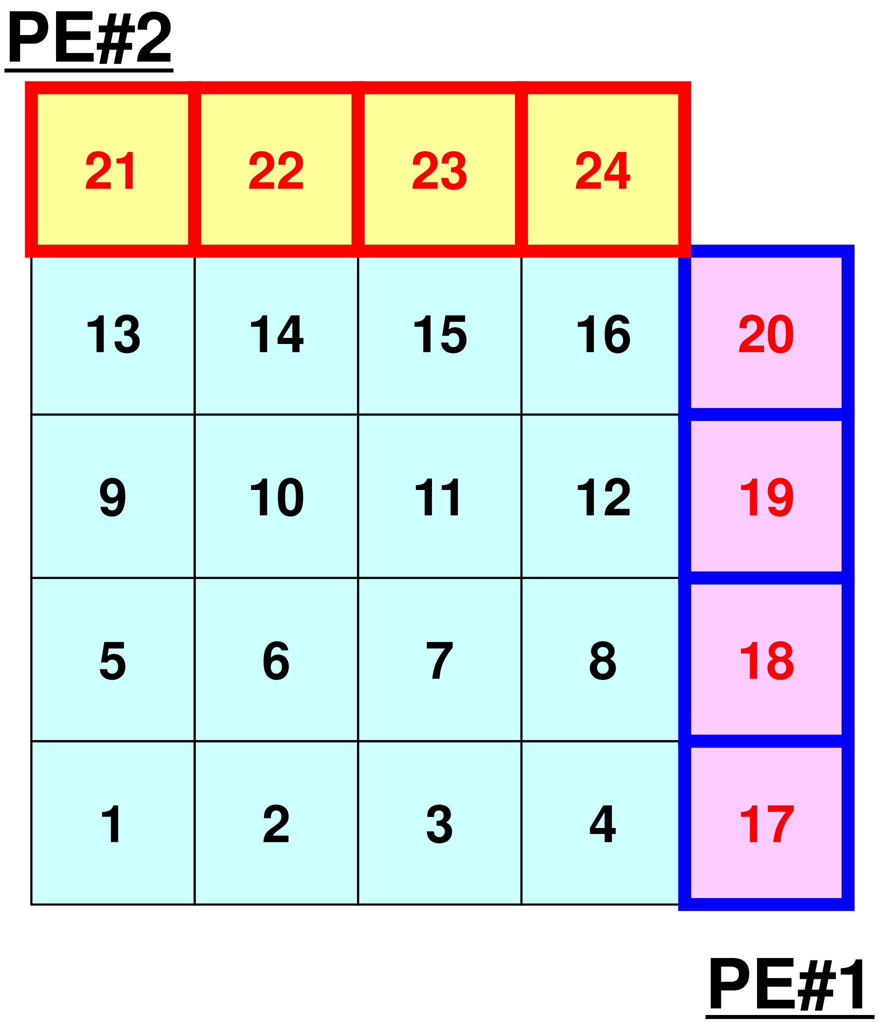
Generalized Comm. Table (3/6)




```
#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
```

Four ext pts (1st-4th items) are imported from 1st neighbor (PE#1), and four (5th-8th items) are from 2nd neighbor (PE#2).

Generalized Comm. Table (4/6)

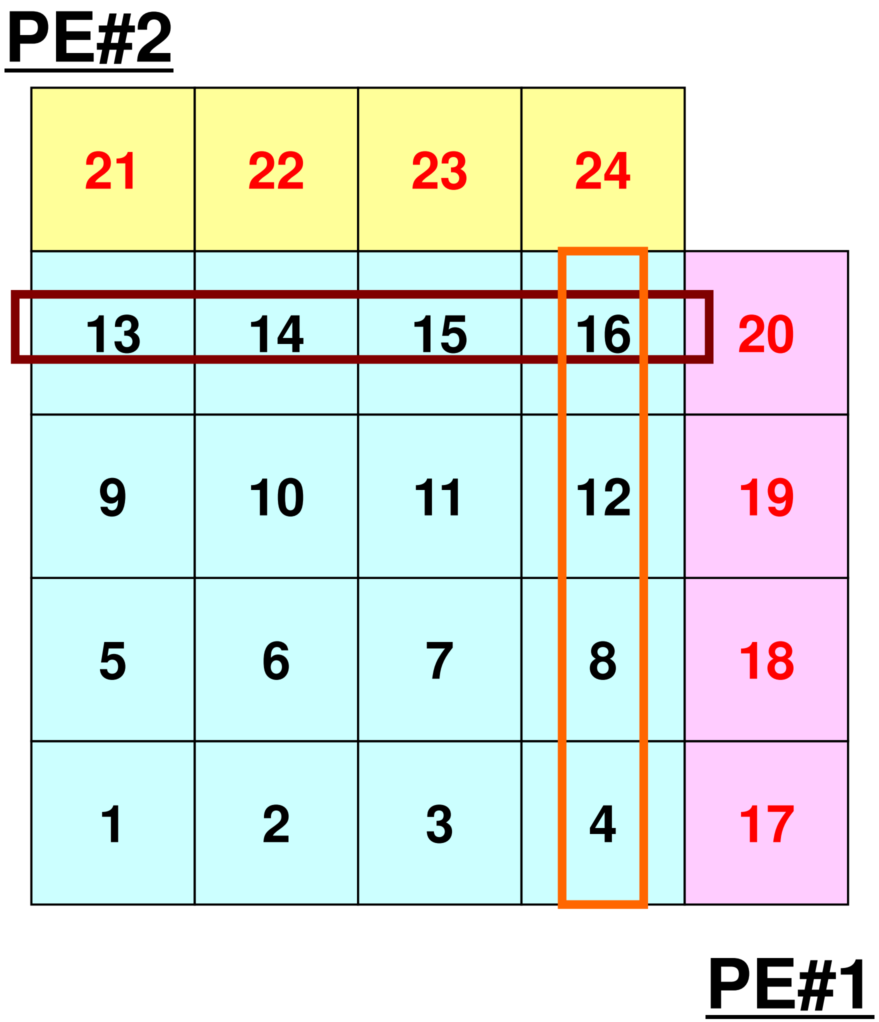



```
#NEIBPEtot
2
#NEIBPE
1  2
#NODE
24 16
#IMPORT_index
4  8
#IMPORT_items
17
18
19
20
21
22
23
24
#EXPORT_index
4  8
#EXPORT_items
4
8
12
16
13
14
15
16
```

17–20: imported from 1st Neighbor (PE#1) (1st-4th items)

21–24: imported from 2nd Neighbor (PE#2) (5th-8th items)

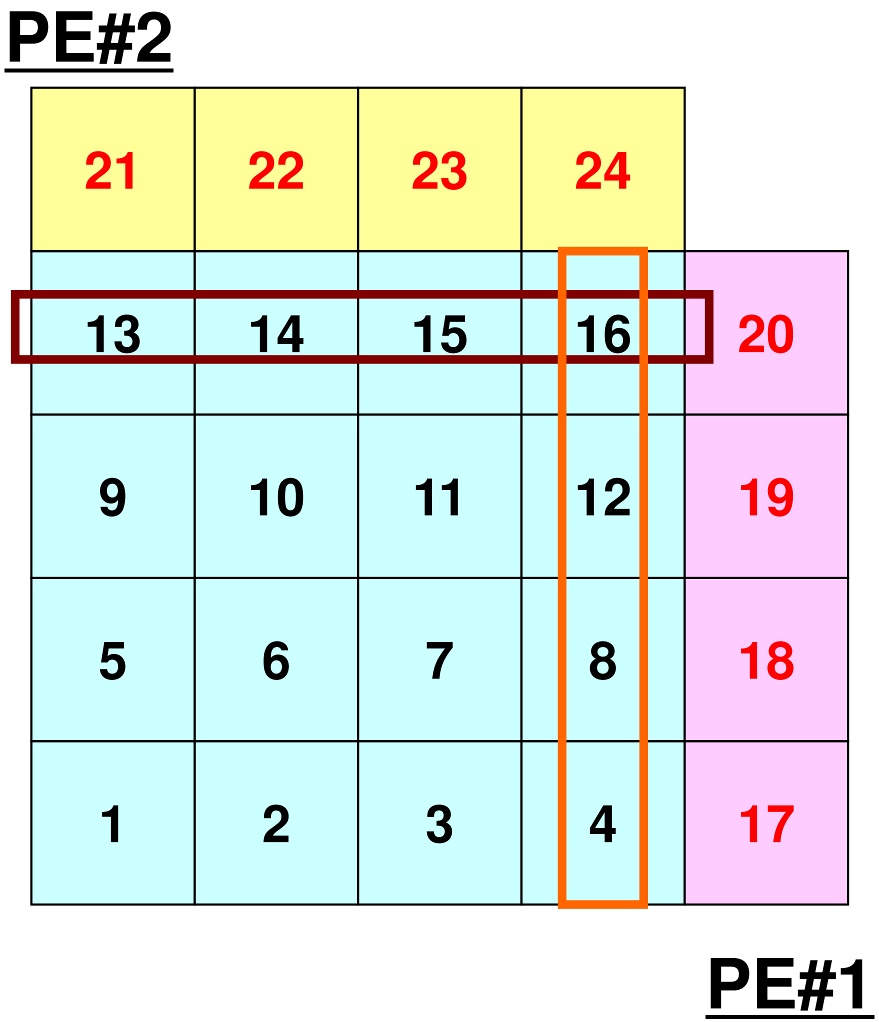
Generalized Comm. Table (5/6)




```
#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
```

Four boundary pts (1st-4th items) are exported to 1st neighbor (PE#1), and four (5th-8th items) are to 2nd neighbor (PE#2).

Generalized Comm. Table (6/6)

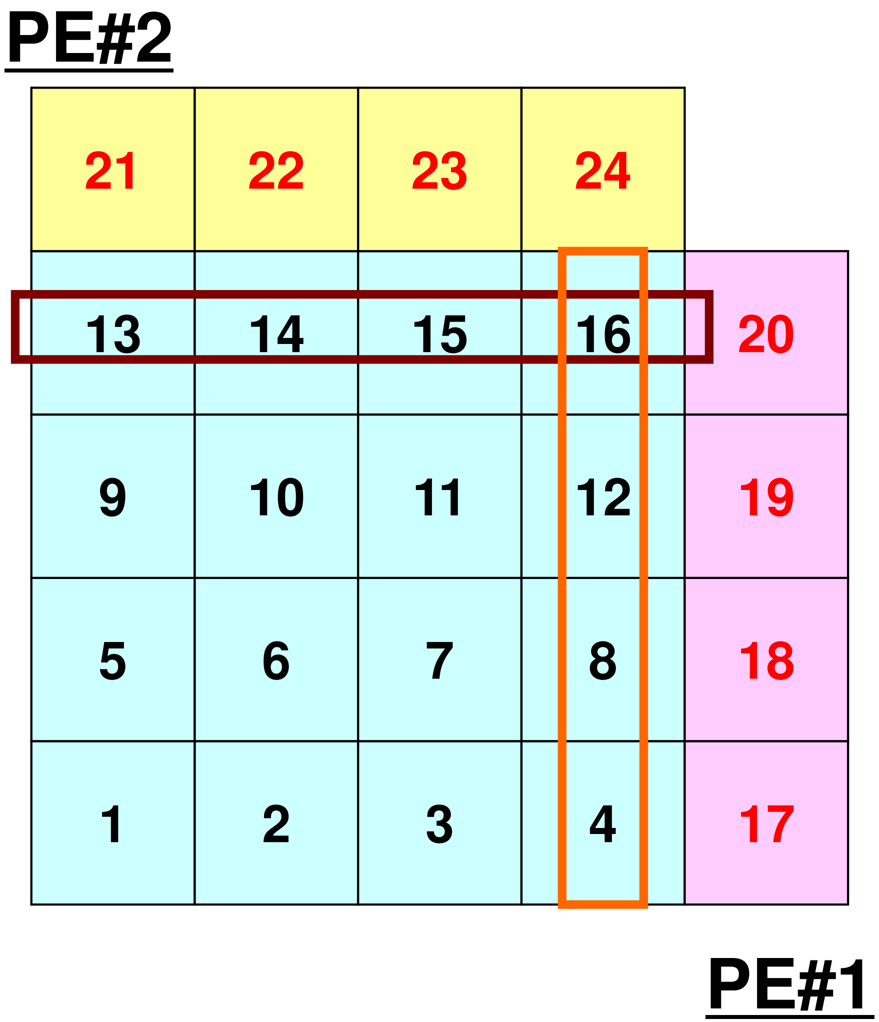



```
#NEIBPEtot
2
#NEIBPE
1  2
#NODE
24 16
#IMPORT_index
4  8
#IMPORT_items
17
18
19
20
21
22
23
24
#EXPORT_index
4  8
#EXPORT_items
4
8
12
16
13
14
15
16
```

exported to 1st Neighbor (PE#1) (1st-4th items)

exported to 2nd Neighbor (PE#2) (5th-8th items)

Generalized Comm. Table (6/6)



An external point is only sent from its original domain.

A boundary point could be referred from more than one domain, and sent to multiple domains (e.g. 16th mesh).

Notice: Send/Recv Arrays

```
#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)
```

- “length_send” of sending process must be equal to “length_recv” of receiving process.
  - PE#0 to PE#1, PE#1 to PE#0
- “sendbuf” and “recvbuf”: different address

Point-to-Point Communication

- What is PtoP Communication ?
- 2D Problem, Generalized Communication Table
  - 2D FDM
  - Problem Setting
  - Distributed Local Data and Communication Table
  - Implementation
- Report S2

Sample Program for 2D FDM

```
$ cd /home/ra020019/<Your-UID>/pFEM/mpi/S2

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

(modify go4.sh for 4 processes)
$ pjsub go4.sh
```

go4.sh

```
#!/bin/bash
#PJM -N "test"
#PJM -L "rscgrp=small"
#PJM -L "node=1"
#PJM --mpi "max-proc-per-node=4"
#PJM -L elapse=00:15:00
#PJM -g ra020019
#PJM -j
#PJM -e err
#PJM -o test.lst


mpiexec ./a.out
```

Example: sq-sr1.f (1/6)

Fortran

Initialization

```
      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 )
```

Example: sq-sr1.f (2/6)

Fortran

Reading distributed local data files (sqm.*)

```
!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)
```

Example: sq-sr1.f (2/6)

Fortran

Reading distributed local data files (sqm.*)

```
!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
```

Example: sq-sr1.f (2/6)

Fortran

Reading distributed local data files (sqm.*)

```
!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
                                                         ETOT)
        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 Number of all meshes (internal + external)

N Number of internal meshes

```
#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
```

Example: sq-sr1.f (2/6)

Fortran

Reading distributed local data files (sqm.*)

```
!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
```

Example: sq-sr1.f (2/6)

Fortran

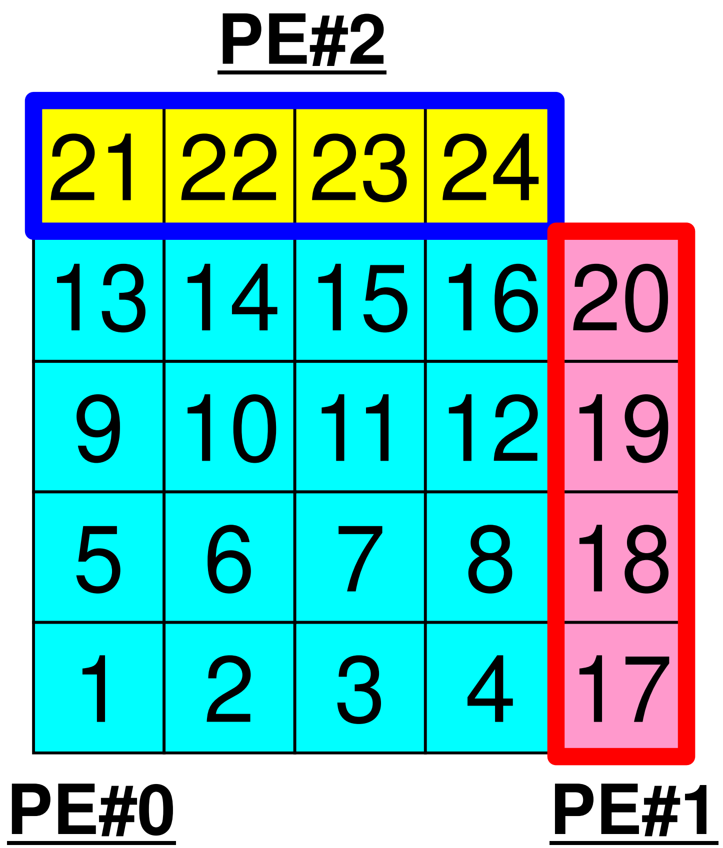
Reading distributed local data files (sqm.*)

```
!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
```

RECV/Import: PE#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
```



Example: sq-sr1.f (2/6)

Fortran

Reading distributed local data files (sqm.*)

```
!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
```

Example: sq-sr1.f (2/6)

Fortran

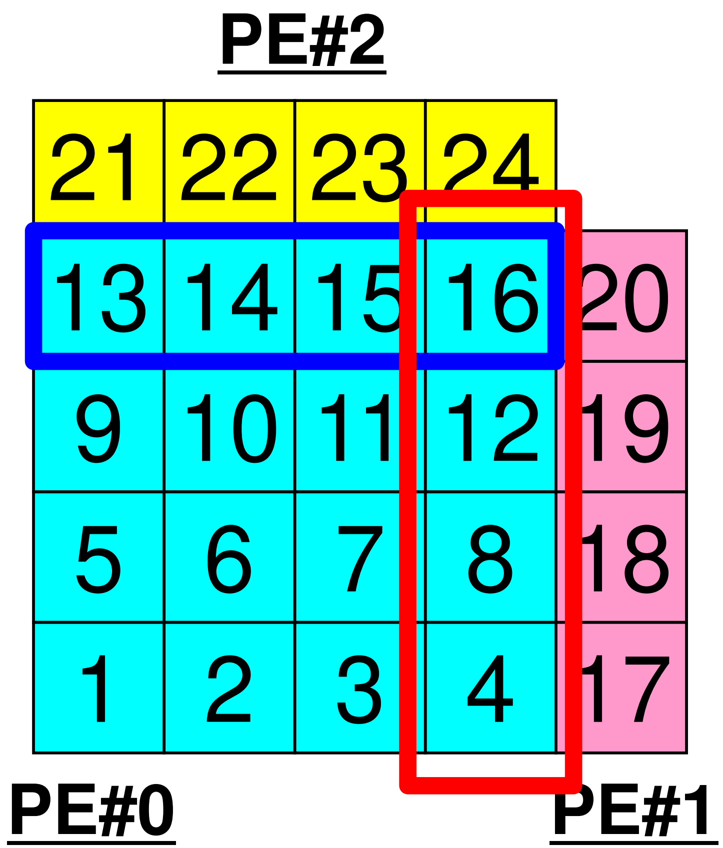
Reading distributed local data files (sqm.*)

```
!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
```

SEND/Export: PE#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
```



Example: sq-sr1.f (3/6)

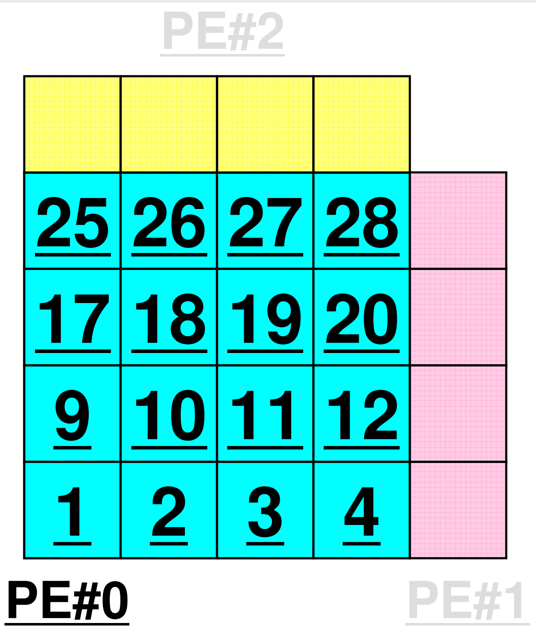
Fortran

Reading distributed local data files (sq.*)

```
!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 : Number of internal points
VAL: Global ID of meshes

VAL on external points are unknown at this stage.




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

Example: sq-sr1.f (4/6)

Fortran

Preparation of sending/receiving buffers

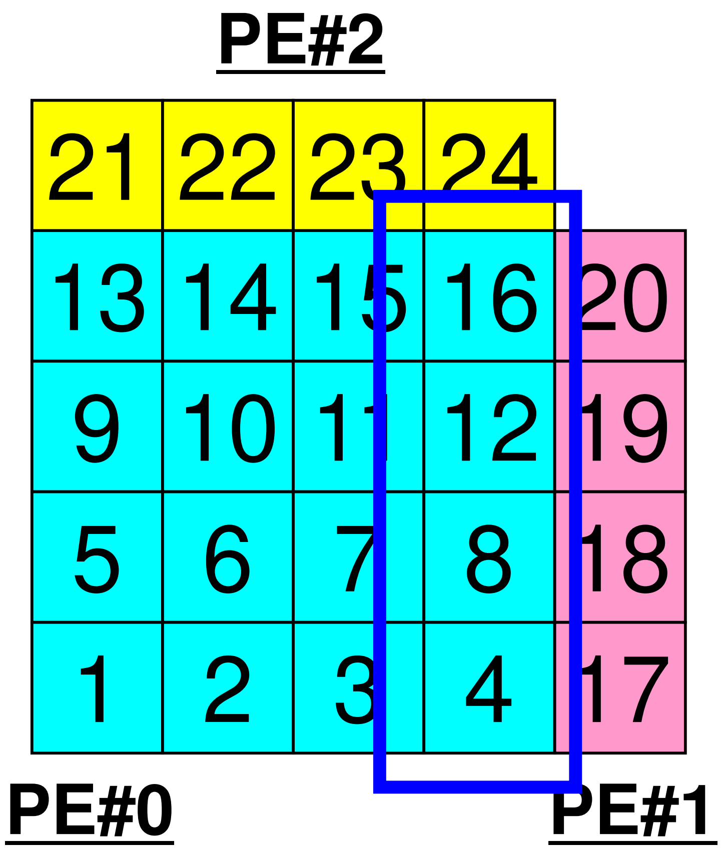
```
!C
!C +--------+
!C | BUFFER |
!C +--------+
!C===
      allocate (SENDbuf(export_index(NEIBPETOT)))
      allocate (RECVbuf(import_index(NEIBPETOT)))

      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===
```

Info. of boundary points is written into sending buffer (**SendBuf**). Info. sent to **NEIBPE(neib)** is stored in **export_index(neib-1)+1:export_inedx(neib)**

Sending Buffer is nice ...

Fortran

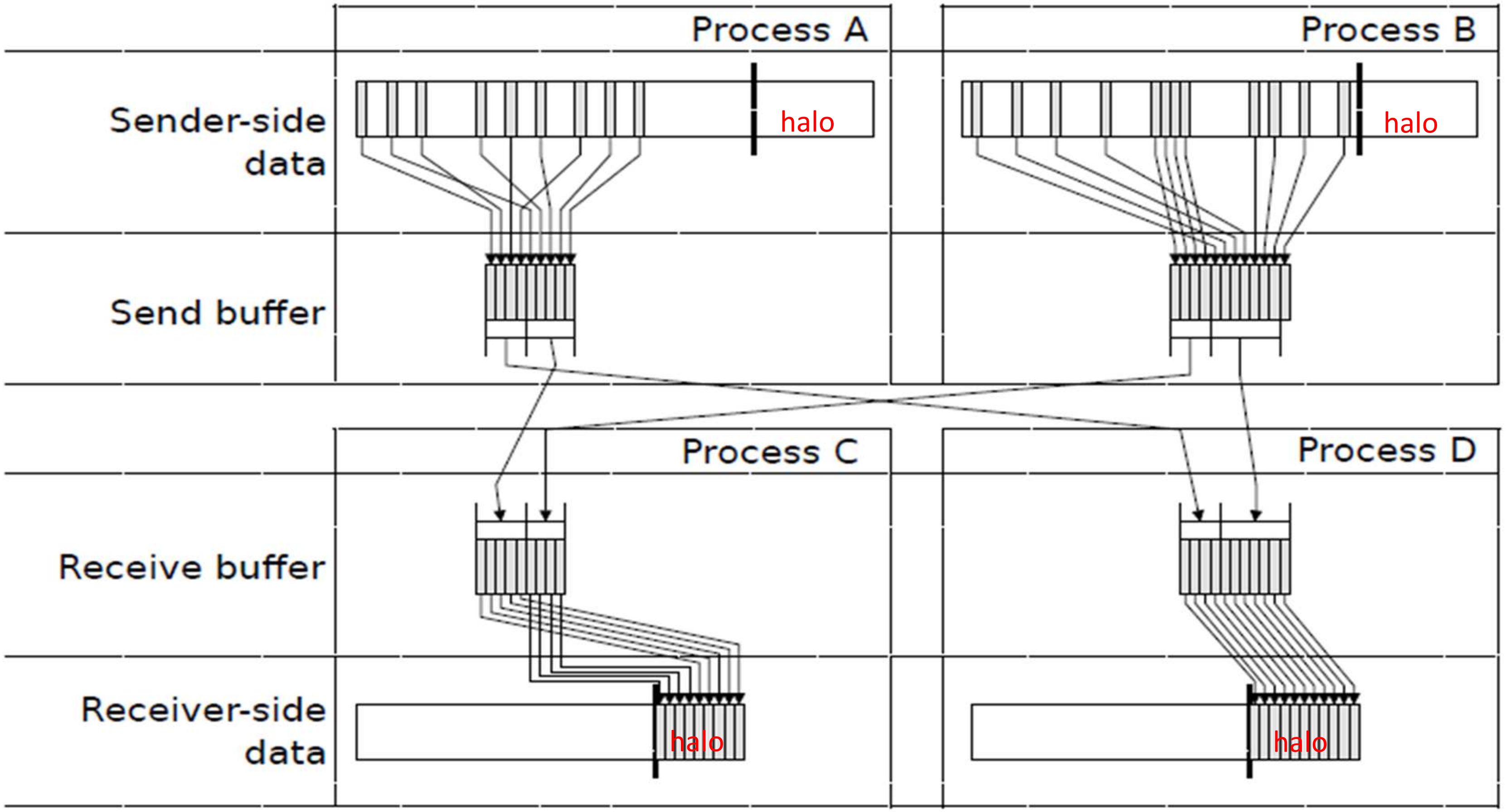
```
 do neib= 1, NEIBPETOT
   iS_e= export_index(neib-1) + 1
   iE_e= export_index(neib  )
   BUFlength_e= iE_e + 1 - iS_e

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



Numbering of these boundary nodes is not continuous, therefore the following procedure of MPI_Isend is not applied directly:

- Starting address of sending buffer
- XX-messages from that address

Communication Pattern using 1D Structure



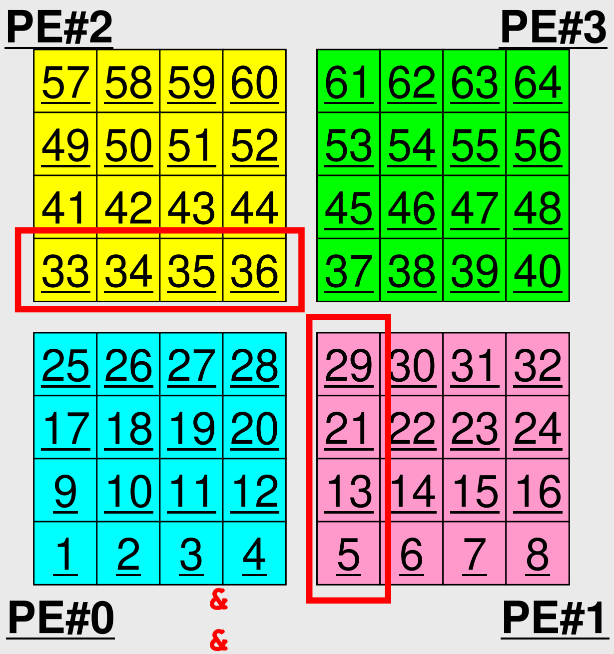
Dr. Osni Marques
（Lawrence Berkeley National Laboratory）より借用

Example: sq-sr1.f (5/6)

Fortran

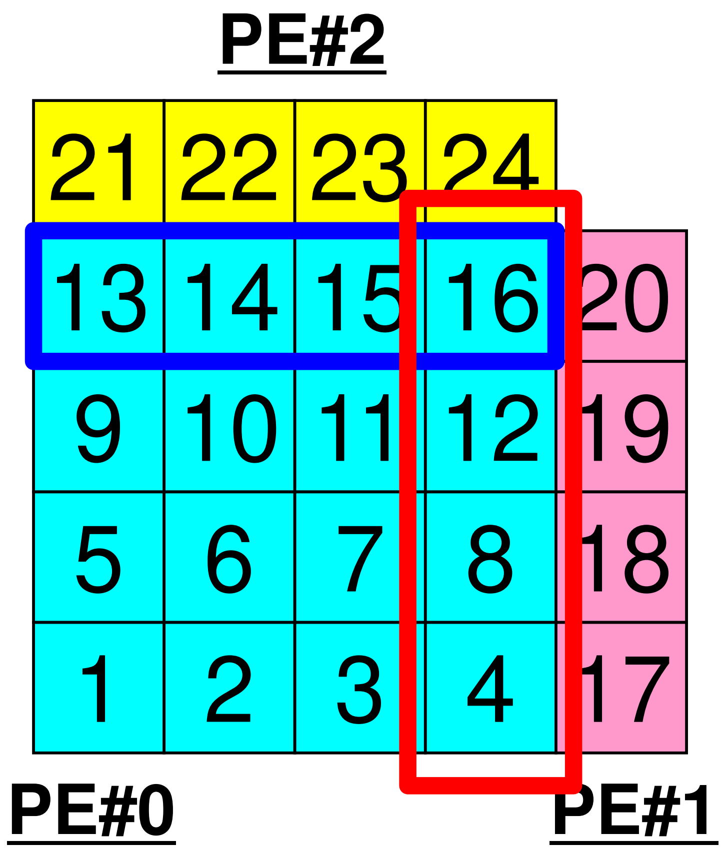
SEND/Export: MPI_Isend

```
!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
```



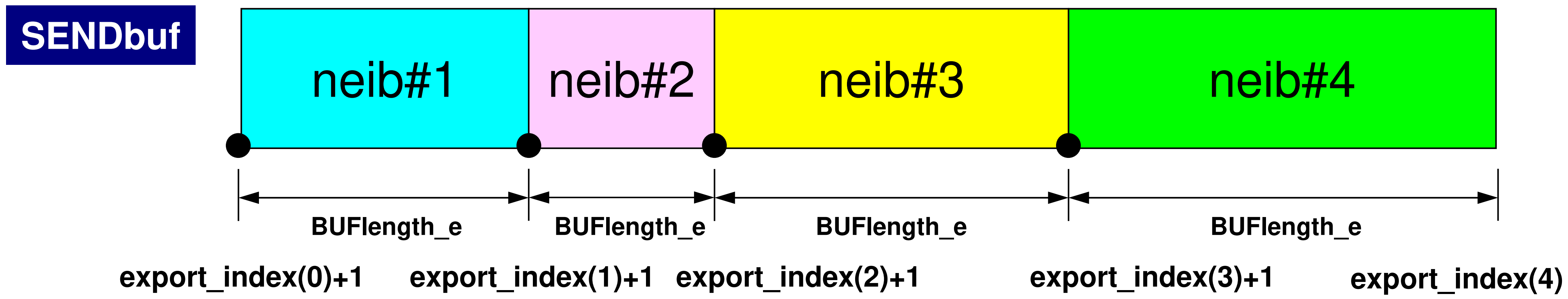
SEND/Export: PE#0

```
#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
```



SEND: 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)
```

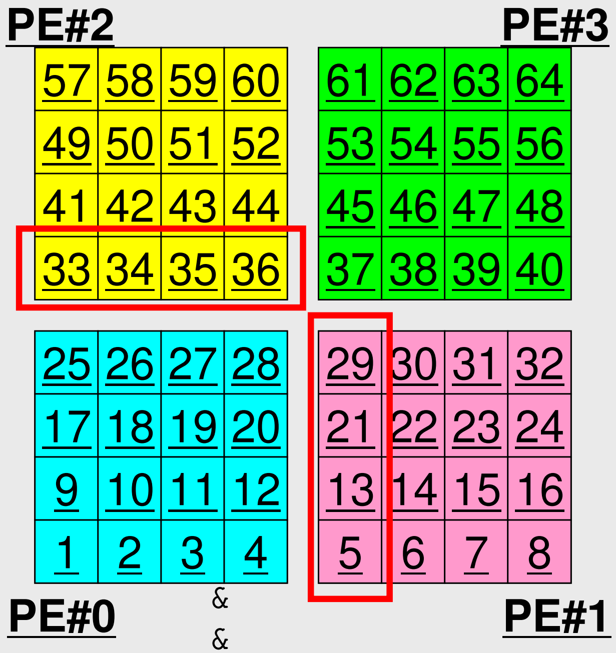
Copies to sending buffers

Example: sq-sr1.f (5/6)

Fortran

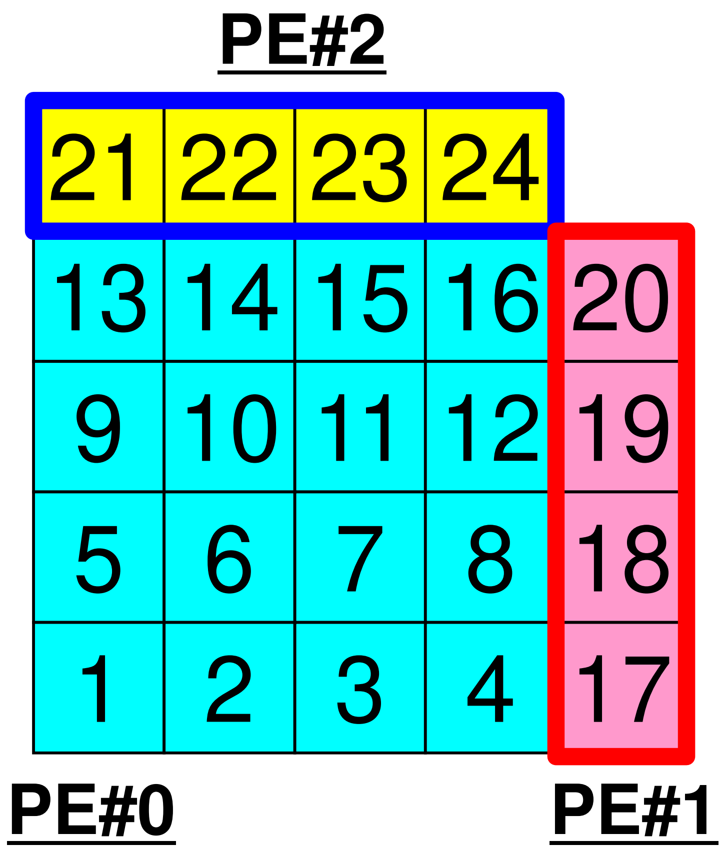
RECV/Import: MPI_Irecv

```
!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
```



RECV/Import: PE#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
```

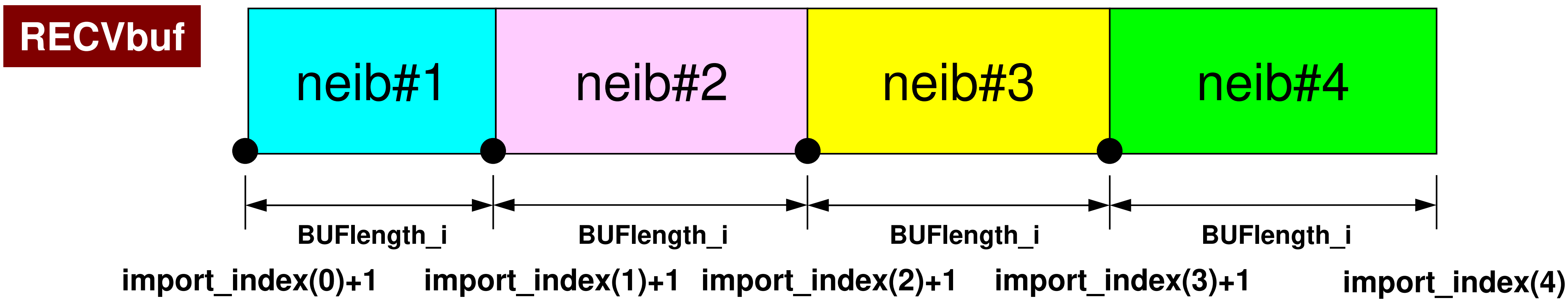


RECV: MPI_Isend/Irecv/Waitall

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)
     enddo
  enddo
```

Copies from receiving buffers



Example: sq-sr1.f (6/6)

Fortran

Reading info of ext pts from receiving buffers

```
      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
```

Contents of **RecvBuf** are copied to values at external points.

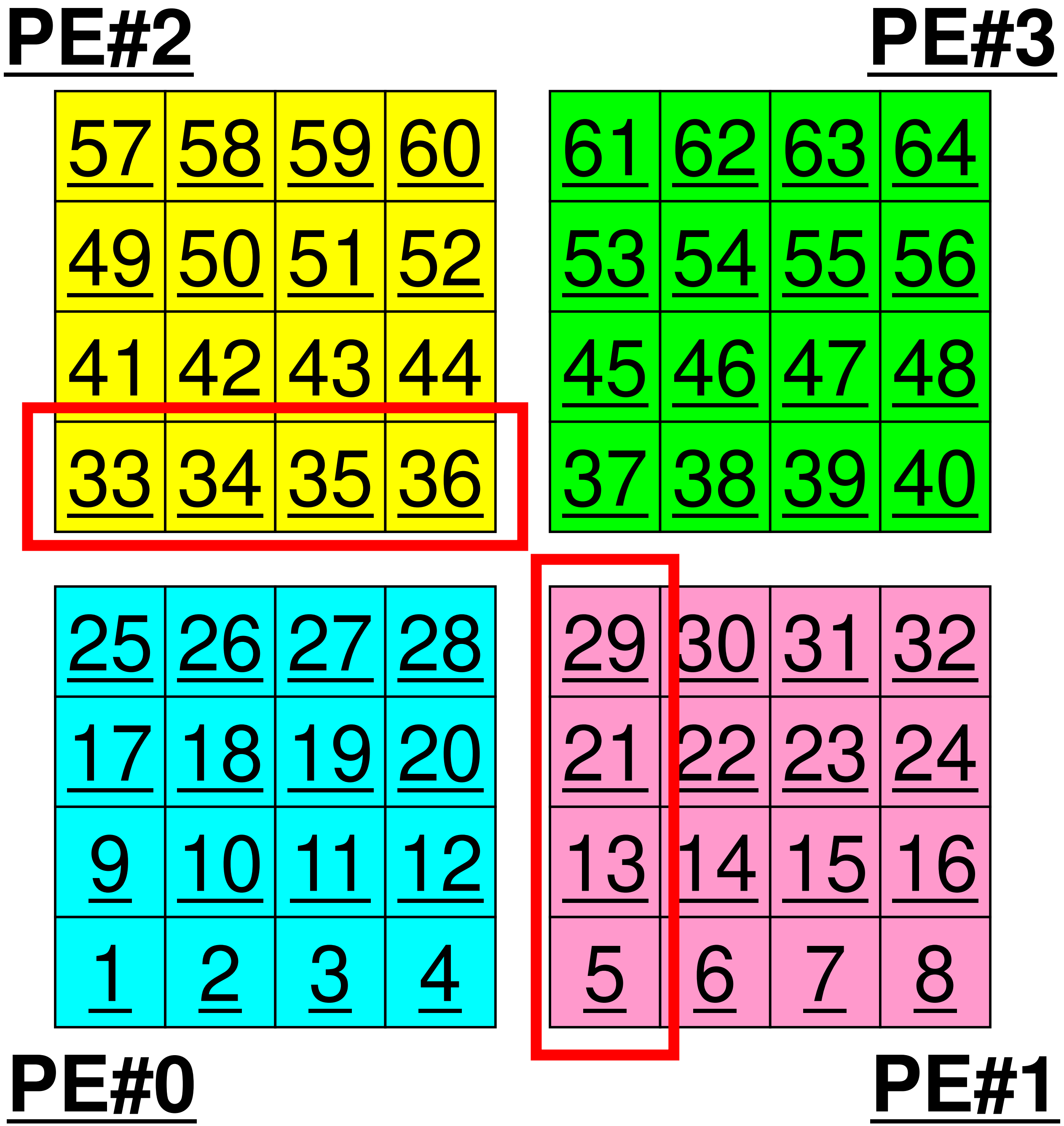
Example: sq-sr1.f (6/6)

Fortran

Writing values at external points

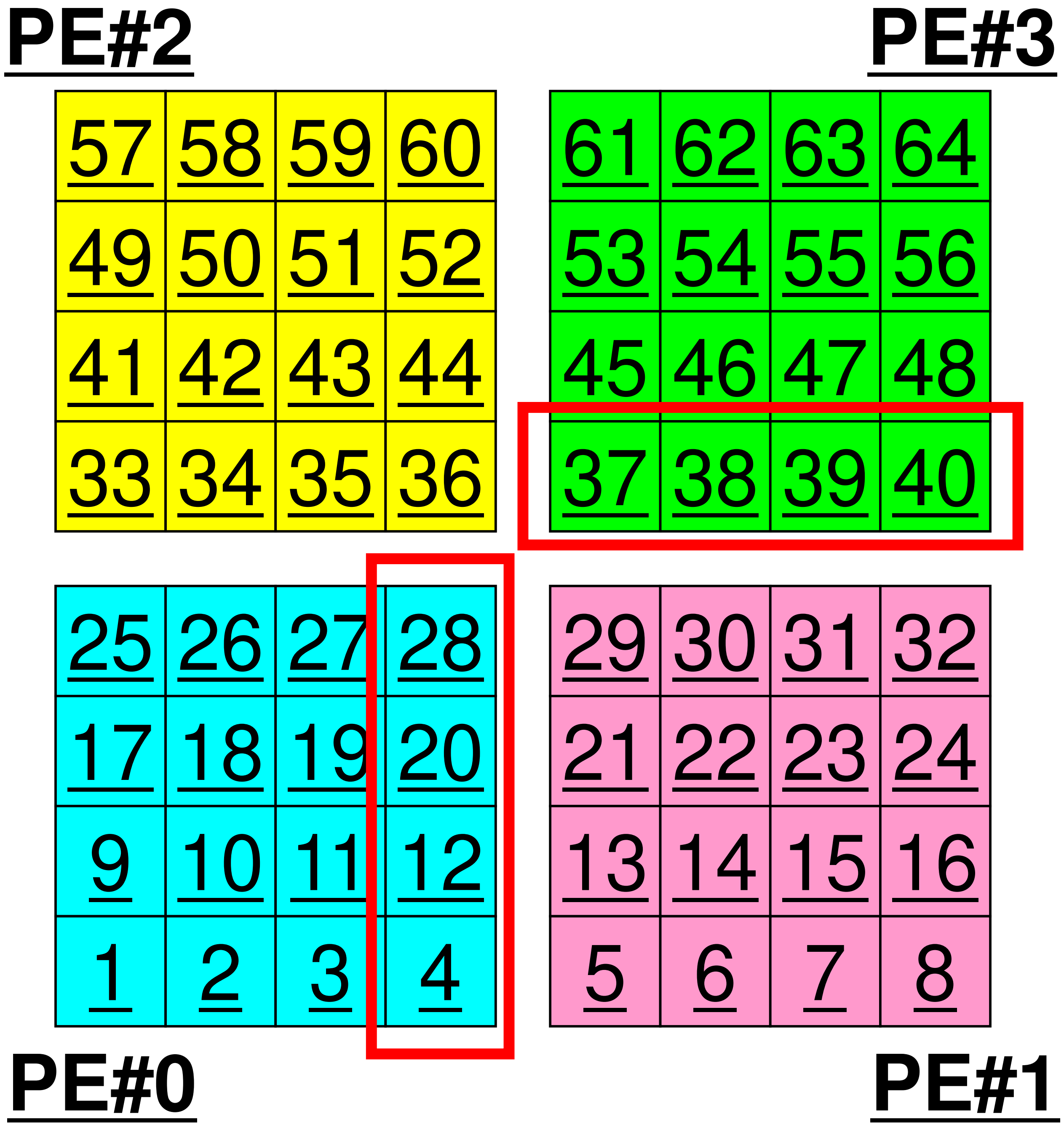
```
      call MPI_WAITALL (NEIBPETOT, request_recv, stat_recv, ierr)

      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
```

Results (PE#0)



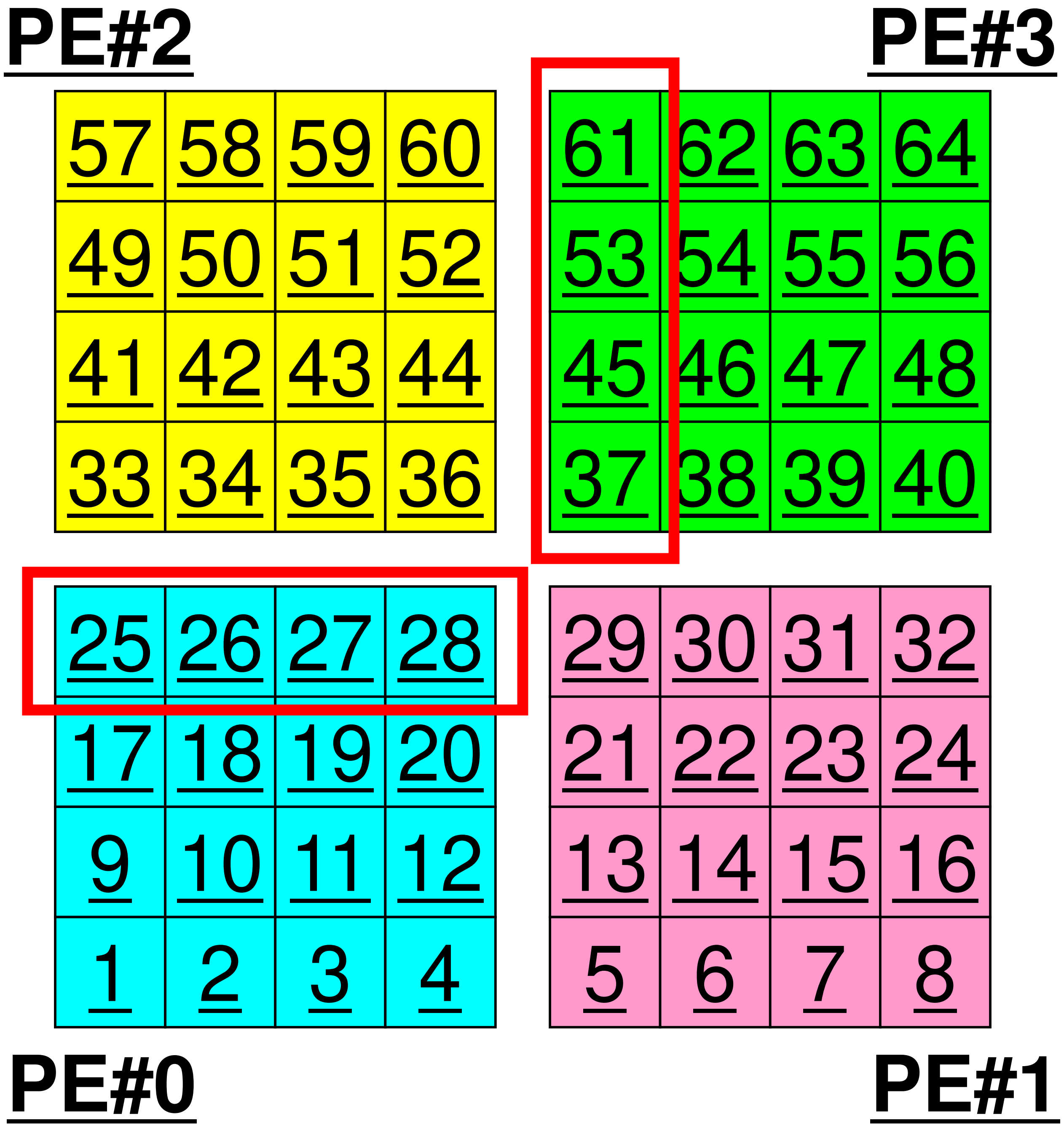
RECVbuf	**0**	**1**	**5**
RECVbuf	**0**	**1**	**13**
RECVbuf	**0**	**1**	**21**
RECVbuf	**0**	**1**	**29**
RECVbuf	**0**	**2**	**33**
RECVbuf	**0**	**2**	**34**
RECVbuf	**0**	**2**	**35**
RECVbuf	**0**	**2**	**36**
RECVbuf	1	0	4
RECVbuf	1	0	12
RECVbuf	1	0	20
RECVbuf	1	0	28
RECVbuf	1	3	37
RECVbuf	1	3	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

Results (PE#1)



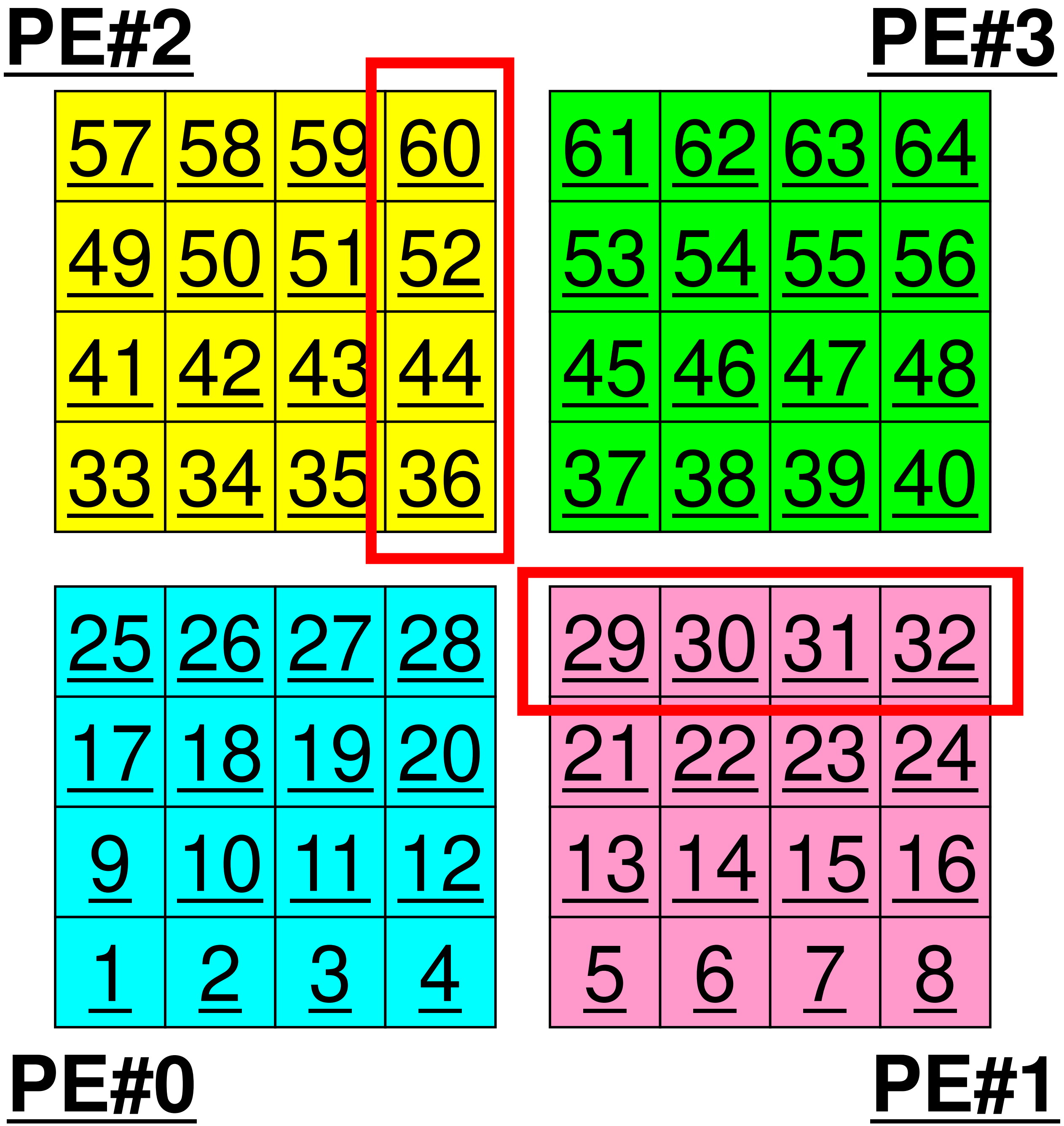
RECVbuf	0	1	5
RECVbuf	0	1	13
RECVbuf	0	1	21
RECVbuf	0	1	29
RECVbuf	0	2	33
RECVbuf	0	2	34
RECVbuf	0	2	35
RECVbuf	0	2	36
RECVbuf	**1**	**0**	**4**
RECVbuf	**1**	**0**	**12**
RECVbuf	**1**	**0**	**20**
RECVbuf	**1**	**0**	**28**
RECVbuf	**1**	**3**	**37**
RECVbuf	**1**	**3**	**38**
RECVbuf	**1**	**3**	**39**
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

Results (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

Results (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
```

Distributed Local Data Structure for Parallel Computation

- Distributed local data structure for domain-to-domain communications has been introduced, which is appropriate for such applications with sparse coefficient matrices (e.g. FDM, FEM, FVM etc.).
  - SPMD
  - Local Numbering: Internal pts to External pts
  - Generalized communication table

- Everything is easy, if proper data structure is defined:
  - Values at boundary pts are copied into sending buffers
  - Send/Recv
  - Values at external pts are updated through receiving buffers

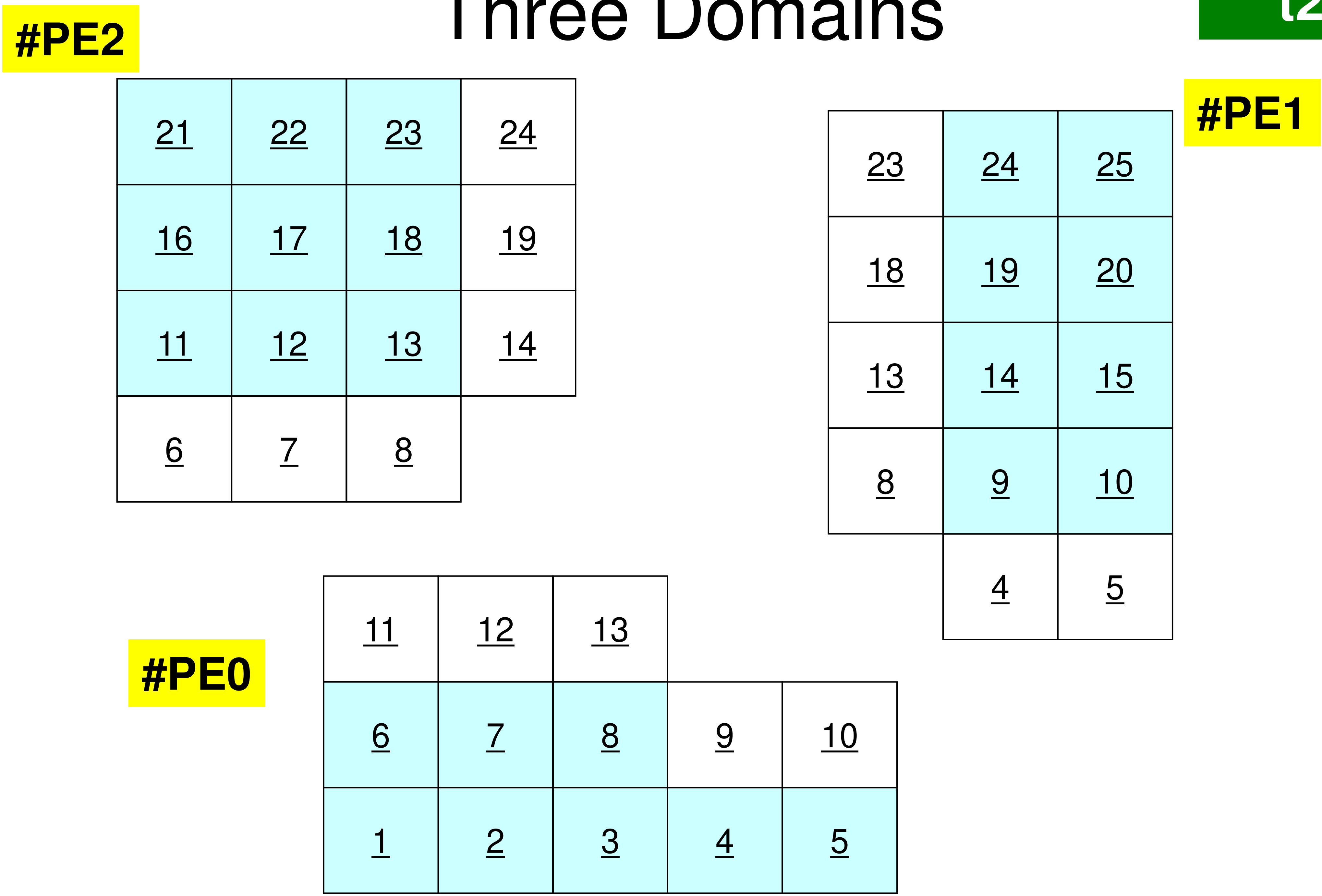
Initial Mesh

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

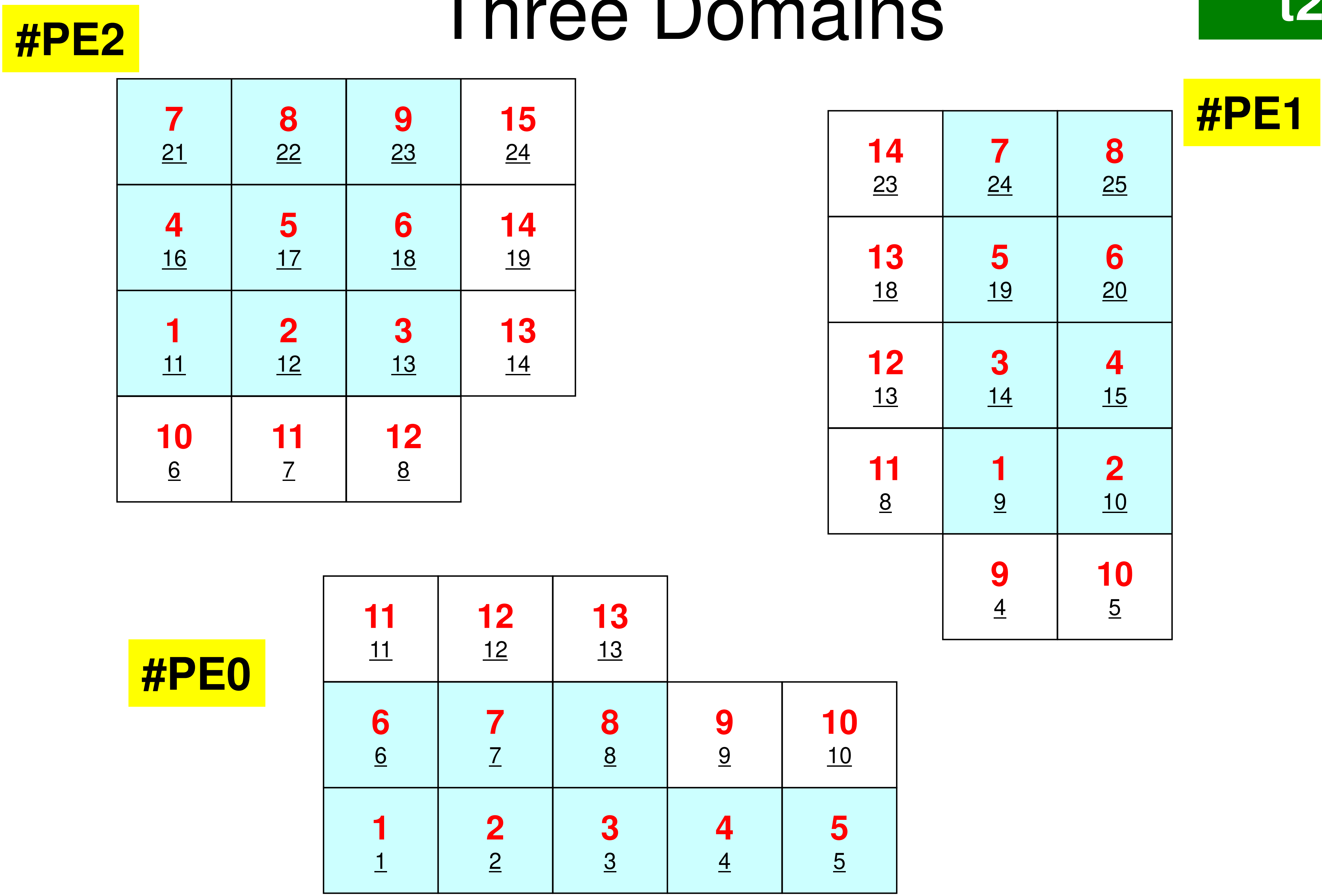
Three Domains

t2



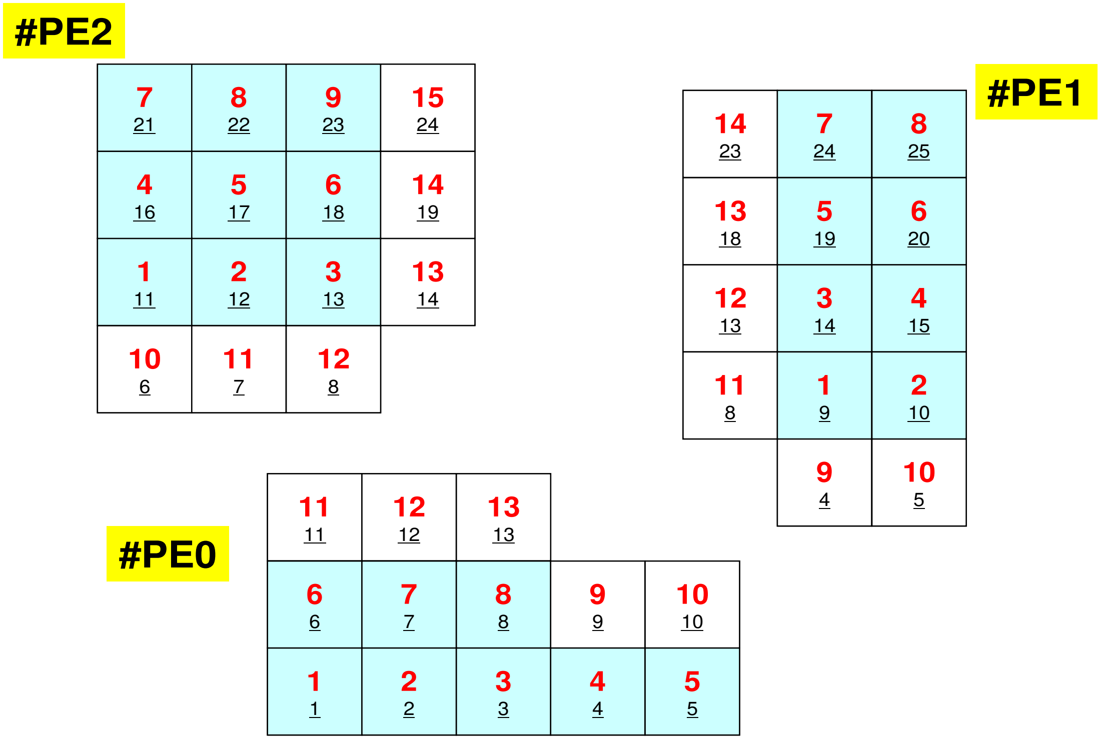
t2

Three Domains



t2

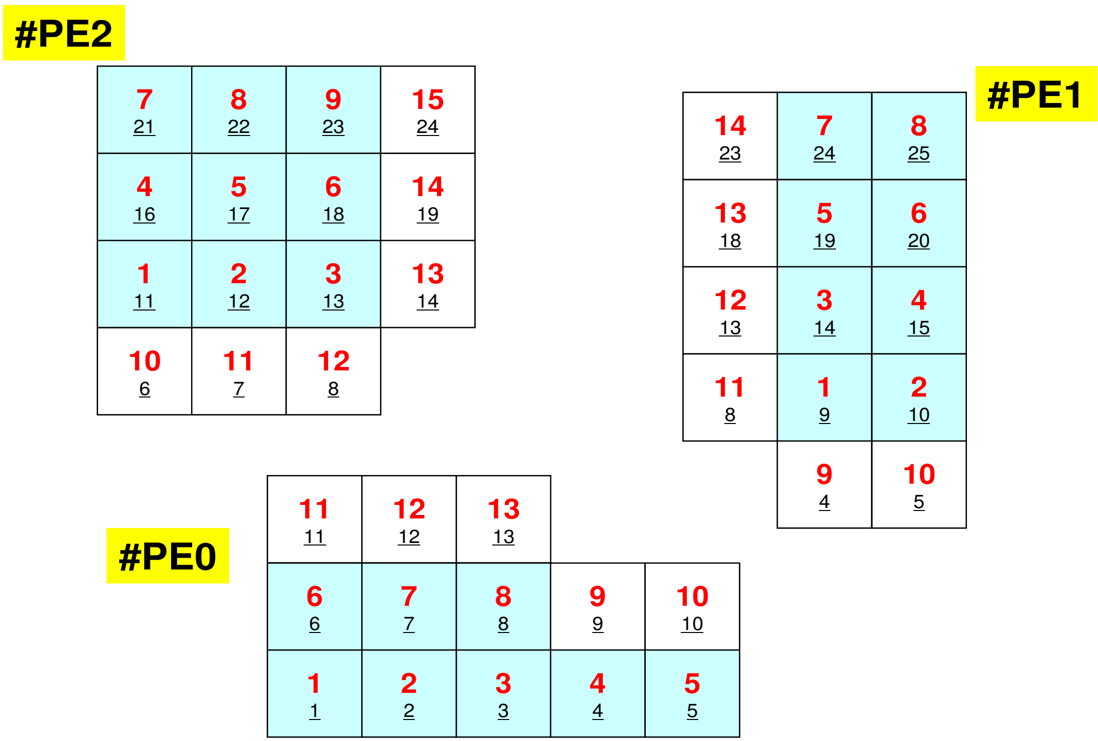
PE#0: sqm.0: fill ○'s




```
#NEIBPEtot
    2
#NEIBPE
    1    2
#NODE
   13    8   (int+ext, int pts)
#IMPORTindex
    ○    ○
#IMPORTitems
    ○…
#EXPORTindex
    ○    ○
#EXPORTitems
    ○…
```

t2

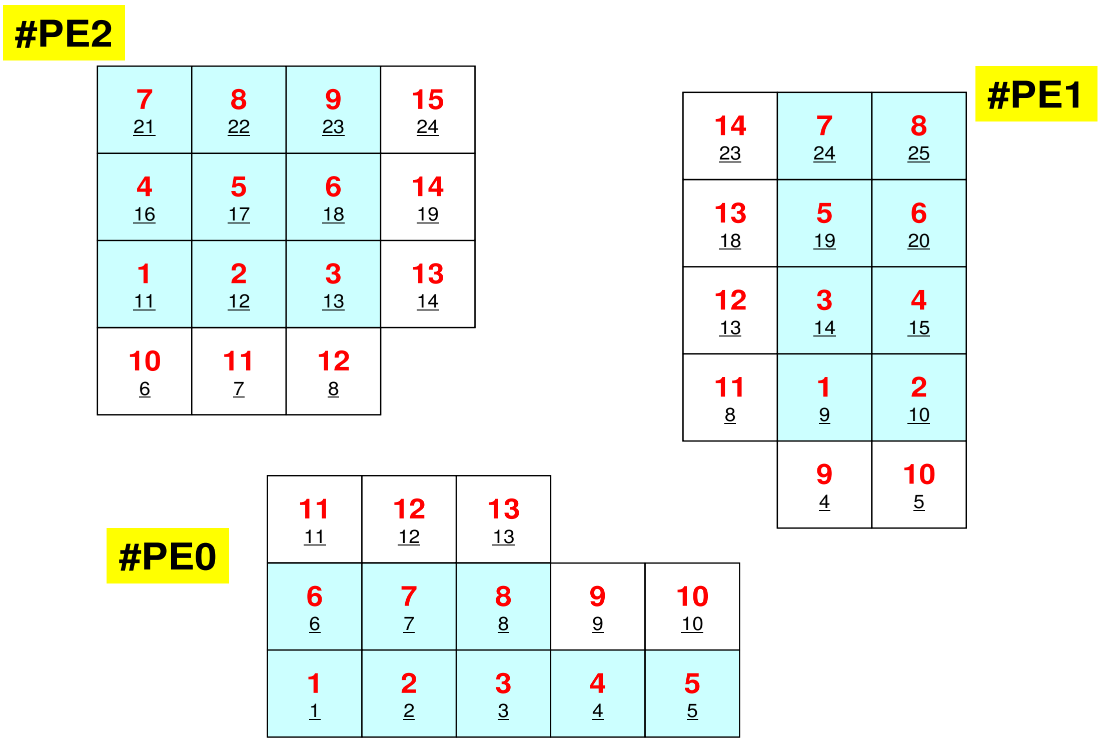
PE#1: sqm.1: fill ○'s



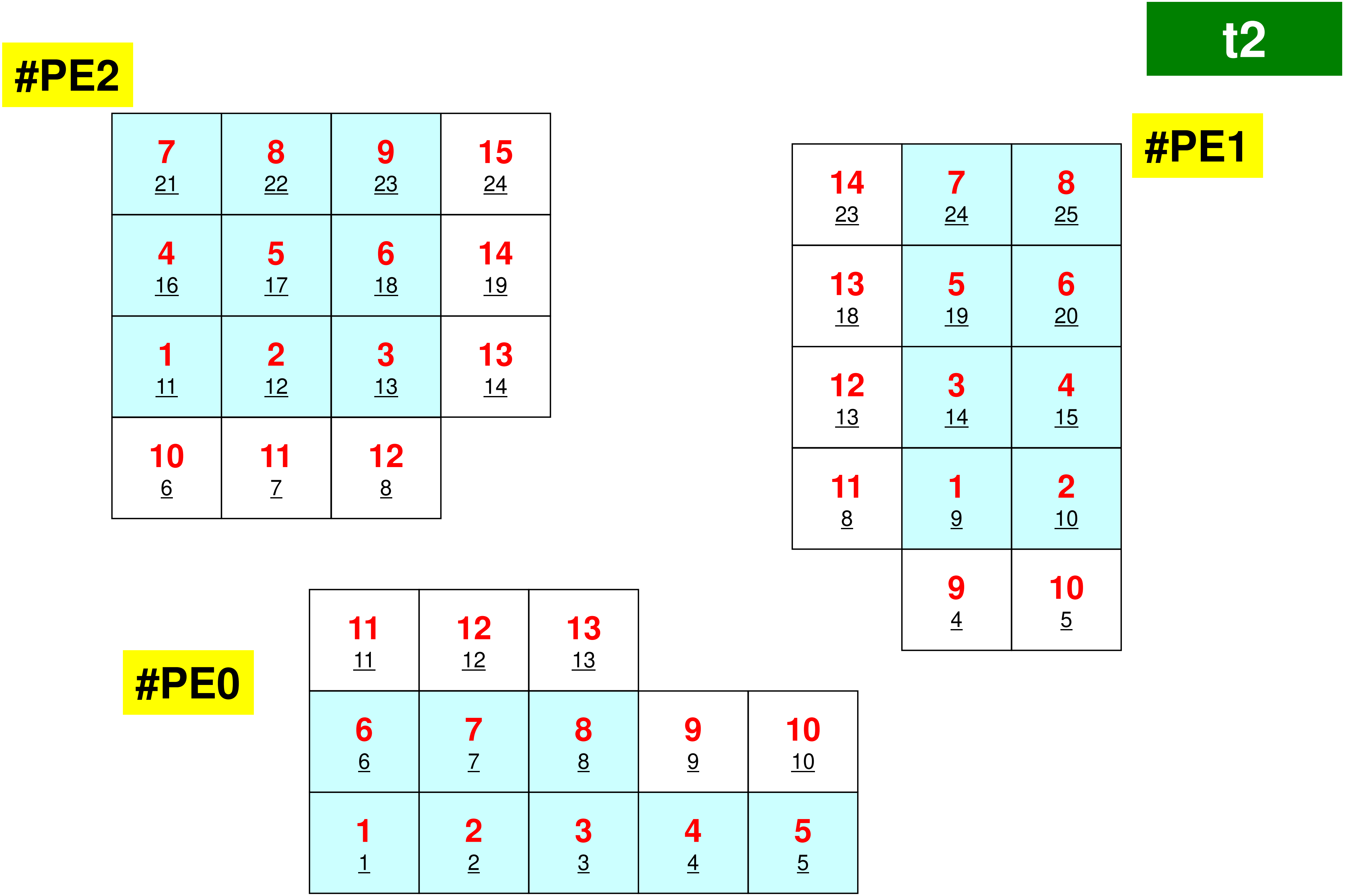

```
#NEIBPEtot
    2
#NEIBPE
    0     2
#NODE
   14      8  (int+ext, int pts)
#IMPORTindex
    ○     ○
#IMPORTitems
    ○…
#EXPORTindex
    ○     ○
#EXPORTitems
    ○…
```

t2

PE#2: sqm.2: fill ○'s




```
#NEIBPEtot
    2
#NEIBPE
    1     0
#NODE
   15     9 (int+ext, int pts)
#IMPORTindex
    ○     ○
#IMPORTitems
    ○…
#EXPORTindex
    ○     ○
#EXPORTitems
    ○…
```


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

Procedures

t2

- Number of Internal/External Points
- Where do External Pts come from ?
  - **IMPORTindex**, **IMPORTitems**
  - Sequence of **NEIBPE**
- Then check destinations of Boundary Pts.
  - **EXPORTindex**, **EXPORTitems**
  - Sequence of **NEIBPE**
- “sq.*” are in **<$O-S2>/ex**
- Create “sqm.*” by yourself
- copy `<$O-S2>/a.out` (by sq-sr1.f) to `<$O-S2>/ex`
- pjsub go3.sh

Exercise S2 (1/2)

- Parallelize 1D code (1d.c) using MPI
- Read entire element number, and decompose into sub-domains in your program

- Validate the results
  - Answer of Original Code = Answer of Parallel Code
  - Explain why number of iterations does not change, as number of MPI processes changes.
- Measure parallel performance

Exercise S2 (2/2)

- Problem
  - Apply “Generalized Communication Table”
  - Read entire elem. #, decompose into sub-domains in your program
  - Evaluate parallel performance
    - You need huge number of elements, to get excellent performance.
    - Fix number of iterations (e.g. 100), if computations cannot be completed.

a012.sh

```
#!/bin/bash
#PJM -N "test"
#PJM -L "rscgrp=small"
#PJM -L "node=1"
#PJM --mpi "max-proc-per-node=12"
#PJM -L elapse=00:15:00
#PJM -g ra020019
#PJM -j
#PJM -e err
#PJM -o test.lst

mpiexec ./a.out
mpiexec numactl -l ./a.out
```

a048.sh

```
#!/bin/bash
#PJM -N "test"
#PJM -L "rscgrp=small"
#PJM -L "node=1"
#PJM --mpi "max-proc-per-node=48"
#PJM -L elapse=00:15:00
#PJM -g ra020019
#PJM -j
#PJM -e err
#PJM -o test.lst

mpiexec ./a.out
mpiexec numactl -l ./a.out
```

a384.sh

```
#!/bin/sh
#PJM -N "test"
#PJM -L "rscgrp=small"
#PJM -L "node=8:torus"
#PJM --mpi "max-proc-per-node=48"
#PJM -L elapse=00:15:00
#PJM -g ra020019
#PJM -j
#PJM -e err
#PJM -o test.lst

mpiexec ./a.out
mpiexec numactl -l ./a.out
```

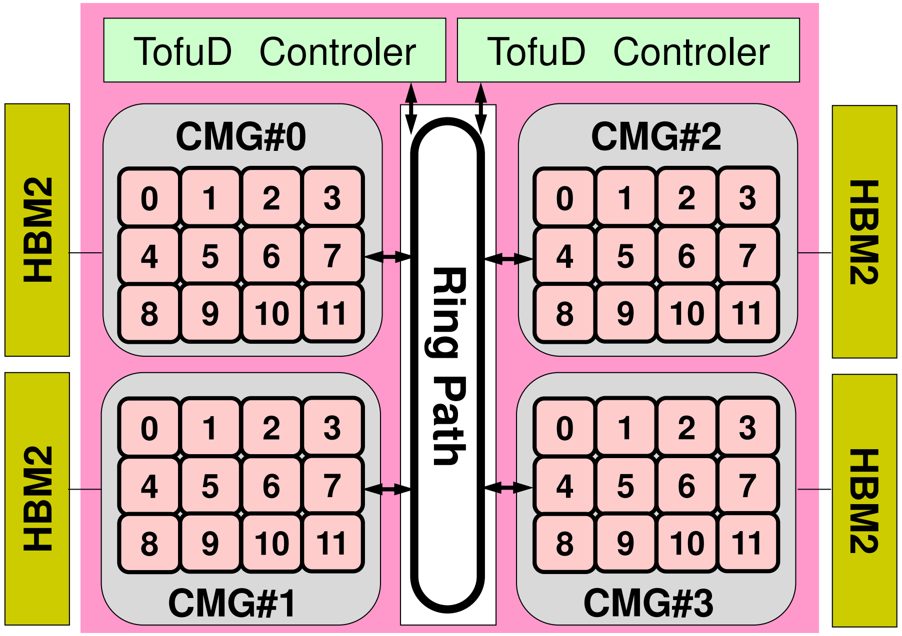
a576.sh

```
#!/bin/sh
#PJM -N "test"
#PJM -L "rscgrp=small"
#PJM -L "node=12:torus"
#PJM --mpi "max-proc-per-node=48
#PJM -L elapse=00:15:00
#PJM -g ra020019
#PJM -j
#PJM -e err
#PJM -o test.lst

mpiexec ./a.out
mpiexec numactl -l ./a.out
```

`numactl -l/--localalloc` for utilizing local memory (no effects)

Number of Processes

```
#PJM -L "node=1";#PJM --mpi "max-proc-per-node=1"  Proc.#=  1
#PJM -L "node=1";#PJM --mpi "max-proc-per-node=4"  Proc.#=  4
#PJM -L "node=1";#PJM --mpi "max-proc-per-node=12" Proc.#= 12
#PJM -L "node=1";#PJM --mpi "max-proc-per-node=24" Proc.#= 24
#PJM -L "node=1";#PJM --mpi "max-proc-per-node=48" Proc.#= 48

#PJM -L "node=4:torus"; #PJM --mpi "max-proc-per-node=48" Proc.#=192
#PJM -L "node=8:torus"; #PJM --mpi "max-proc-per-node=48" Proc.#=384
#PJM -L "node=12:torus";#PJM --mpi "max-proc-per-node=48" Proc.#=576
```



Because Fugaku is now very crowded, it is recommended to add **":torus"** after **"node=XX"** in the script for getting computational resources smoothly, **if XX is larger than 1**. Example for 512 nodes: 12x12x4 with "torus", 14x19x2 without "torus"