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

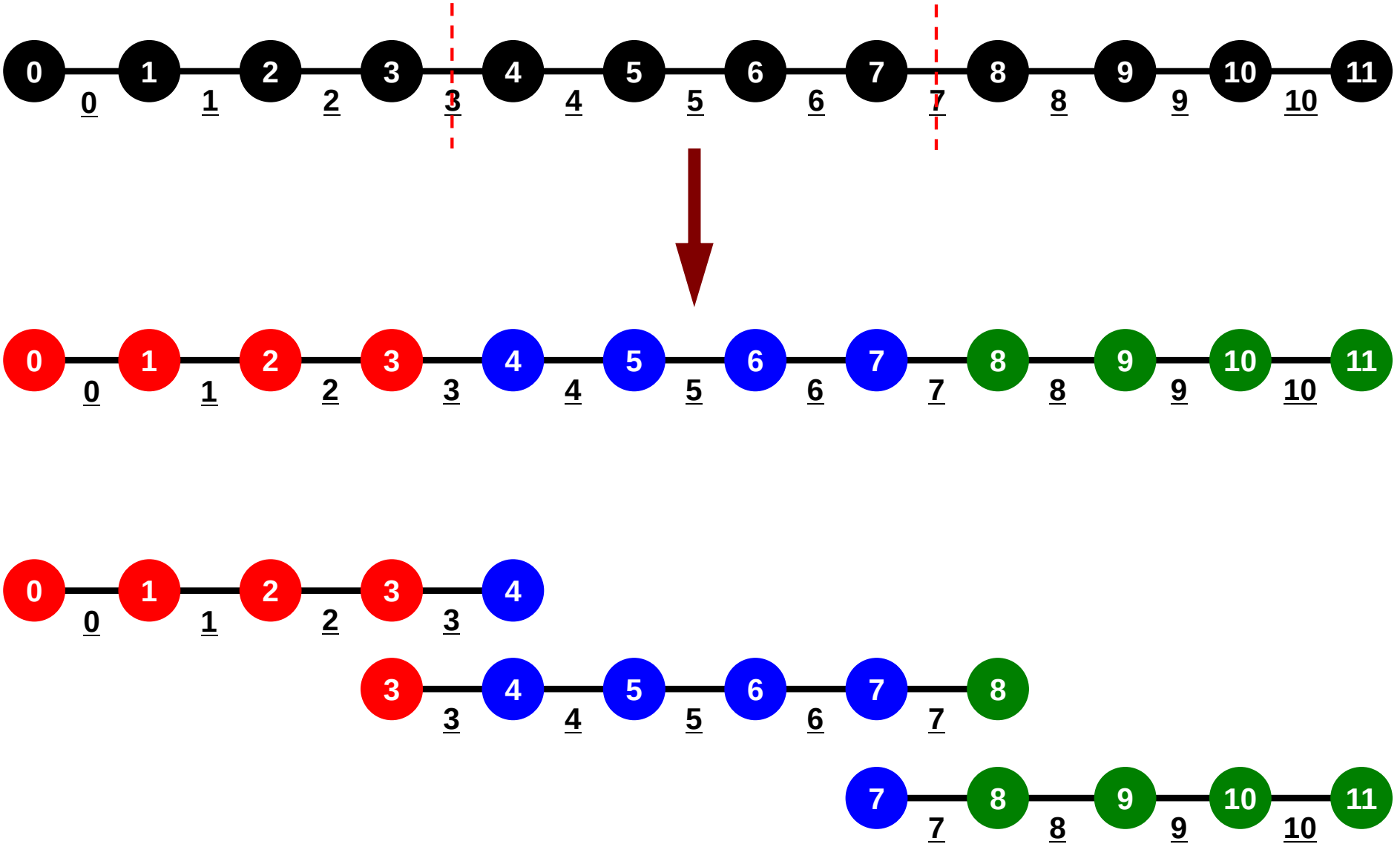
Kengo Nakajima
RIKEN R-CCS

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

Point-to-Point Communication
1対1通信

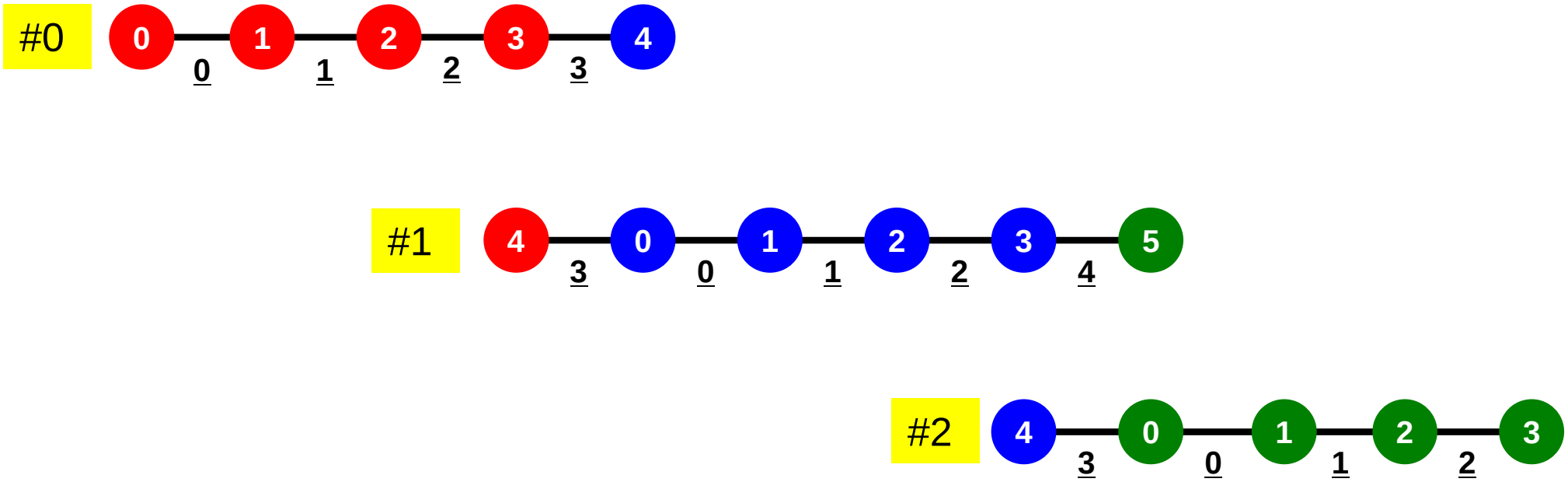
- What is PtoP Communication ?
- 2D Problem, Generalized Communication Table
- Report S2

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



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

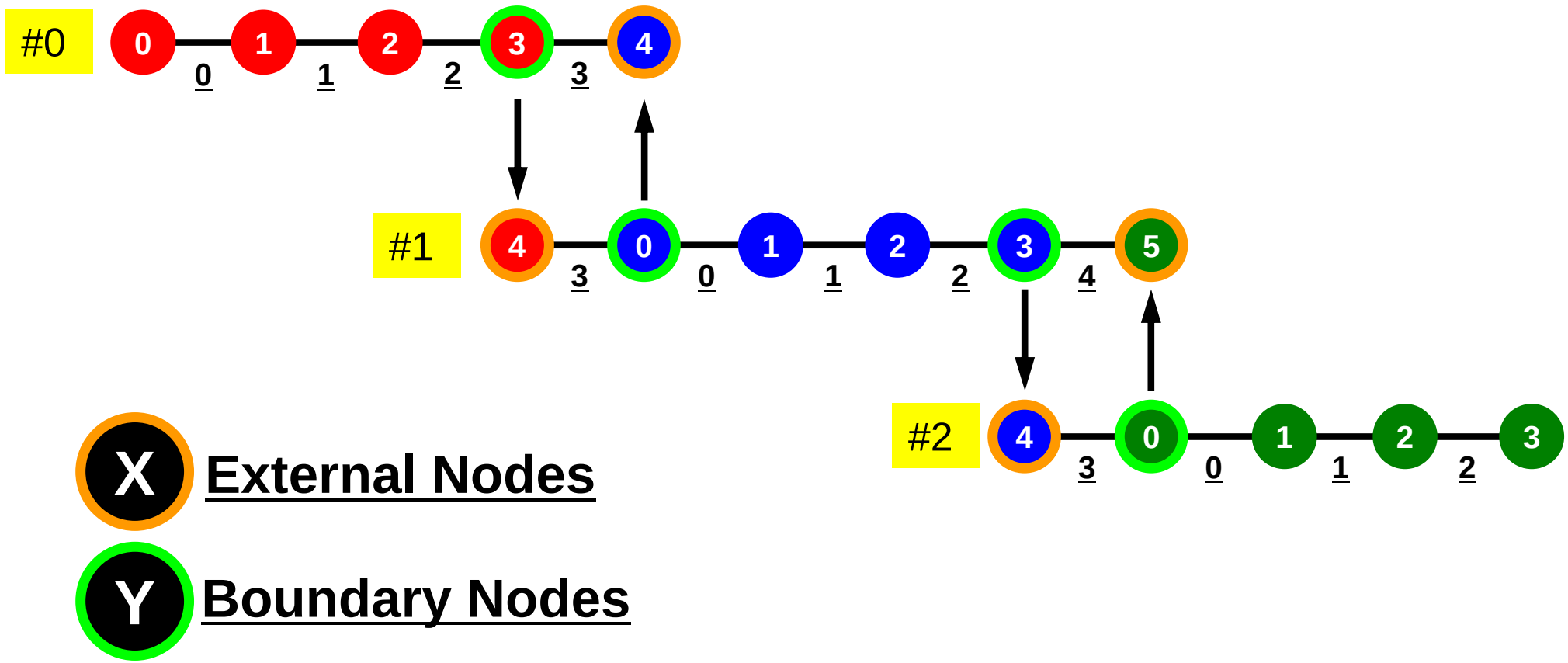
Local ID: Starting from 0 for node and elem at each domain



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

Preconditioner:

Diagonal Scaling
Point-Jacobi Preconditioning

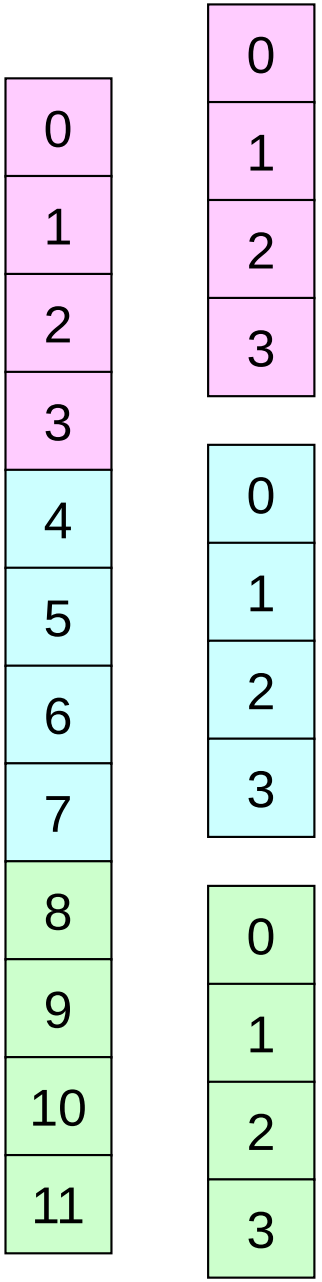
$$[M]=\begin{bmatrix} D_1 & 0 & \dots & 0 & 0 \\ 0 & D_2 & & 0 & 0 \\ \dots & & \dots & & \dots \\ 0 & 0 & & D_{N-1} & 0 \\ 0 & 0 & \dots & 0 & D_N \end{bmatrix}$$

Preconditioning, DAXPY

Local Operations by Only Internal Points: Parallel Processing is possible

```
/*
//-- {z}= [Minv]{r}
*/
        for(i=0;i<N;i++){
          W[Z][i] = W[DD][i] * W[R][i];
        }
```

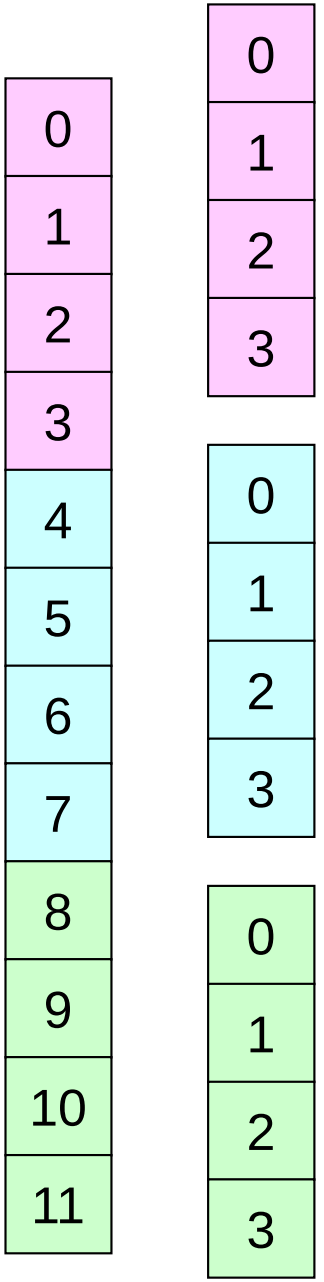
```
/*
//-- {x}= {x} + ALPHA*{p}      DAXPY: double a{x} plus {y}
//   {r}= {r} - ALPHA*{q}
*/
        for(i=0;i<N;i++){
          PHI[i]  += Alpha * W[P][i];
          W[R][i] -= Alpha * W[Q][i];
        }
```



Dot Products

Global Summation needed: Communication ?

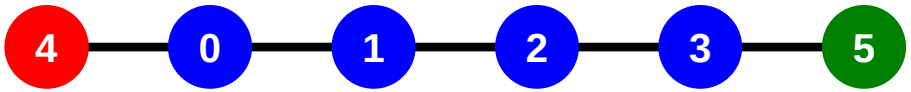
```
/*
//-- ALPHA= RHO / {p}{q}
*/
  C1 = 0.0;
  for(i=0;i<N;i++){
      C1 += W[P][i] * W[Q][i];
  }

  Alpha = Rho / C1;
```



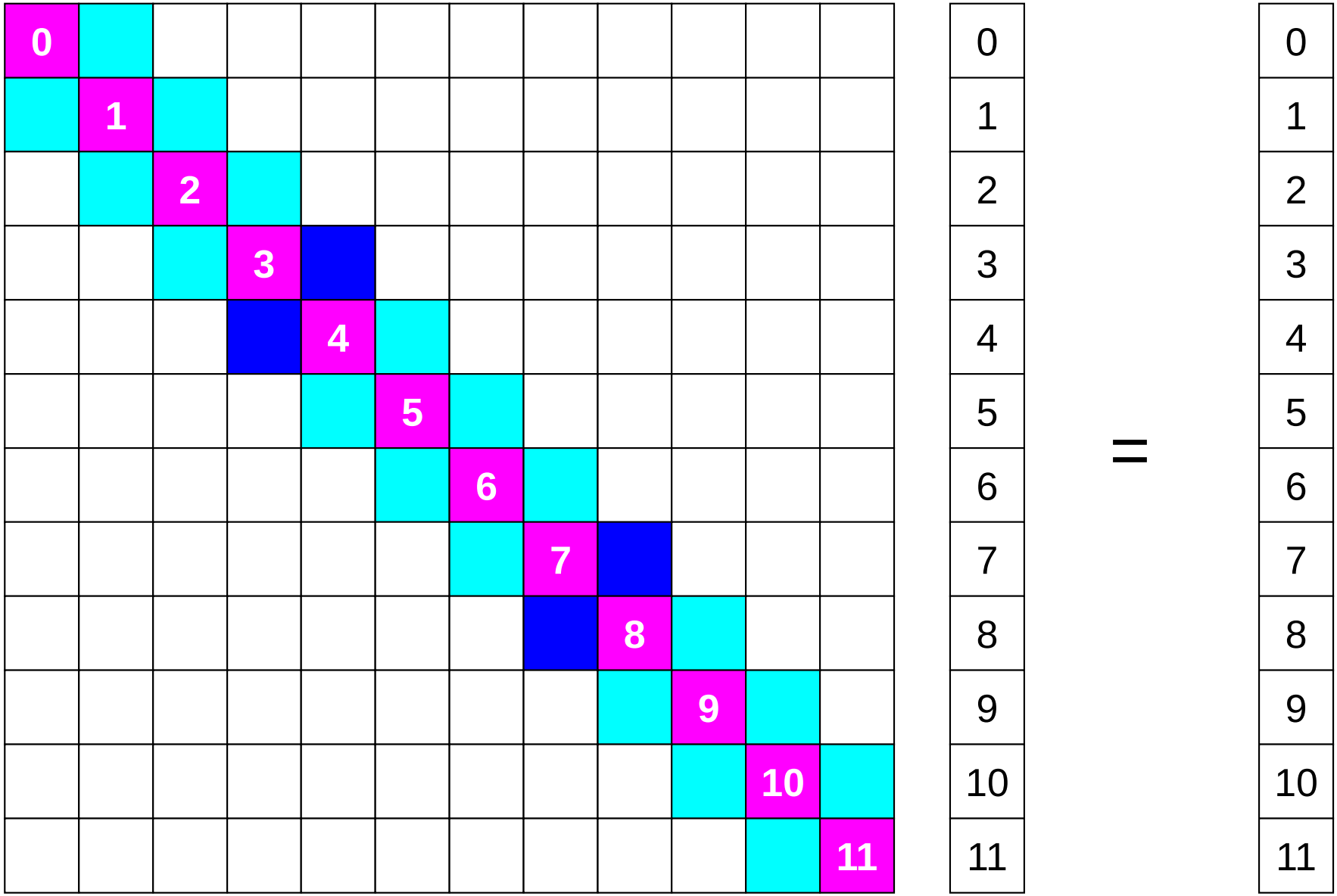
Matrix-Vector Products

Values at External Points: P-to-P Communication

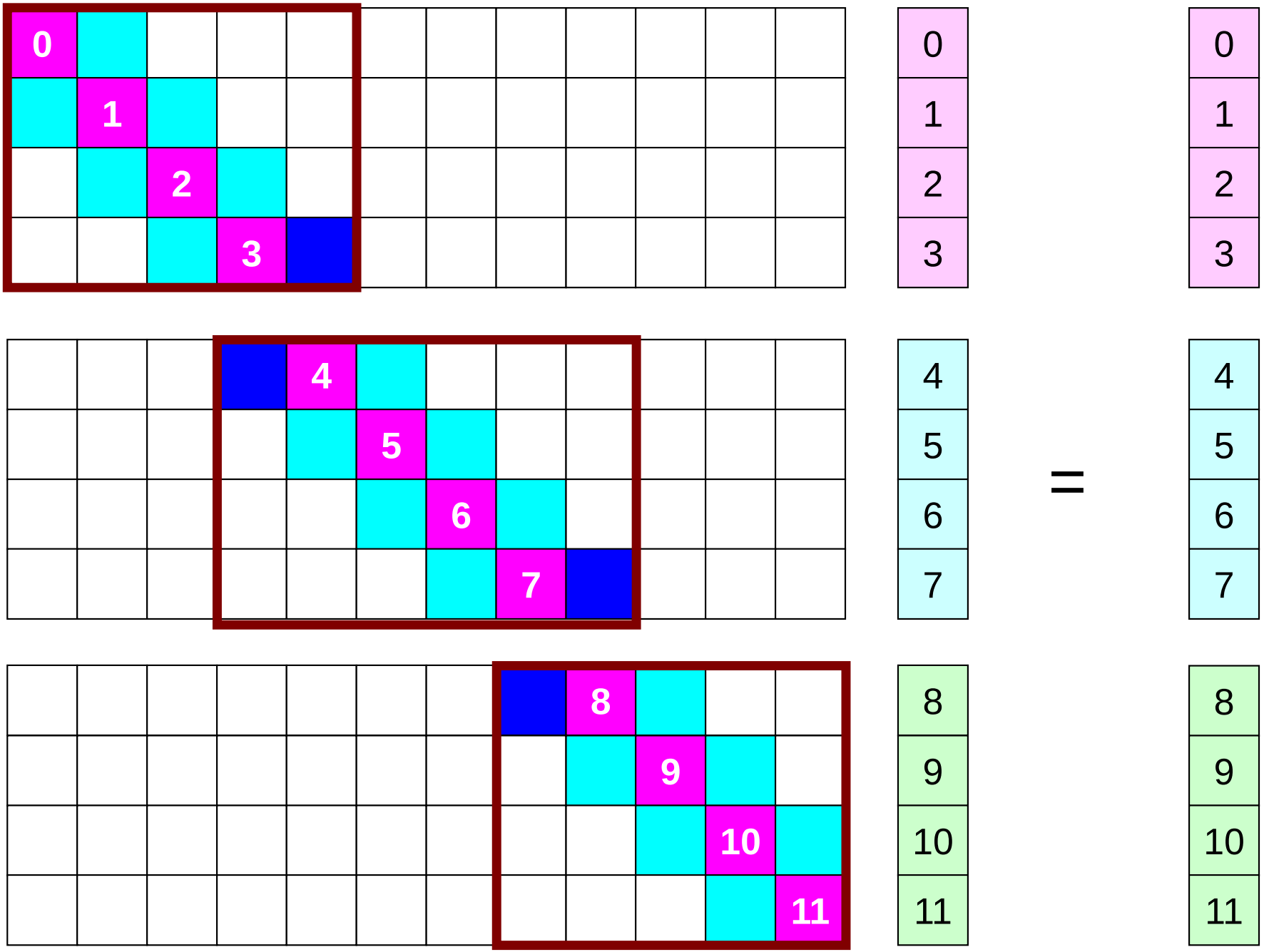
```
/*
//-- {q}= [A]{p}
*/
  for(i=0;i<N;i++){
      W[Q][i] = Diag[i] * W[P][i];
      for(j=Index[i];j<Index[i+1];j++){
          W[Q][i] += AMat[j]*W[P][Item[j]];
          }
      }
```



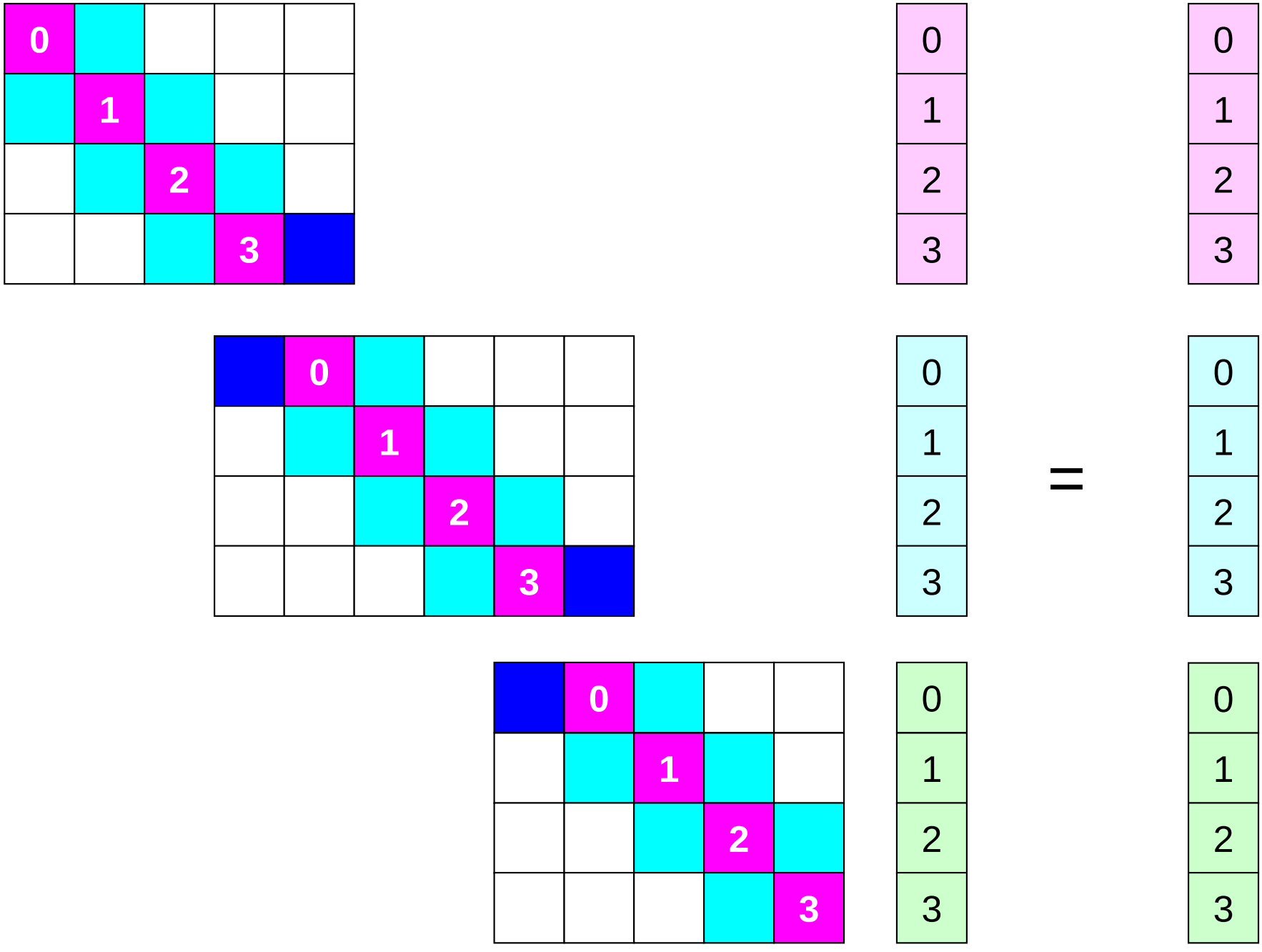
Mat-Vec Products: Local Op. Possible



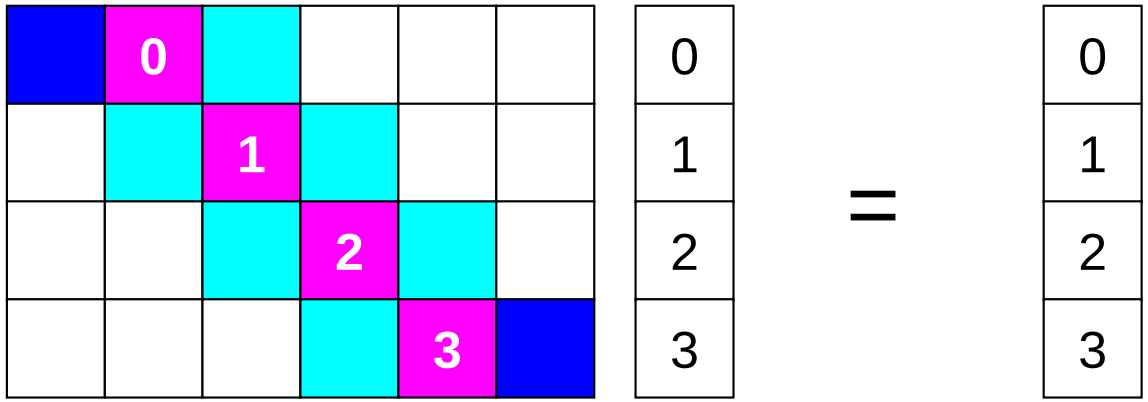
Mat-Vec Products: Local Op. Possible

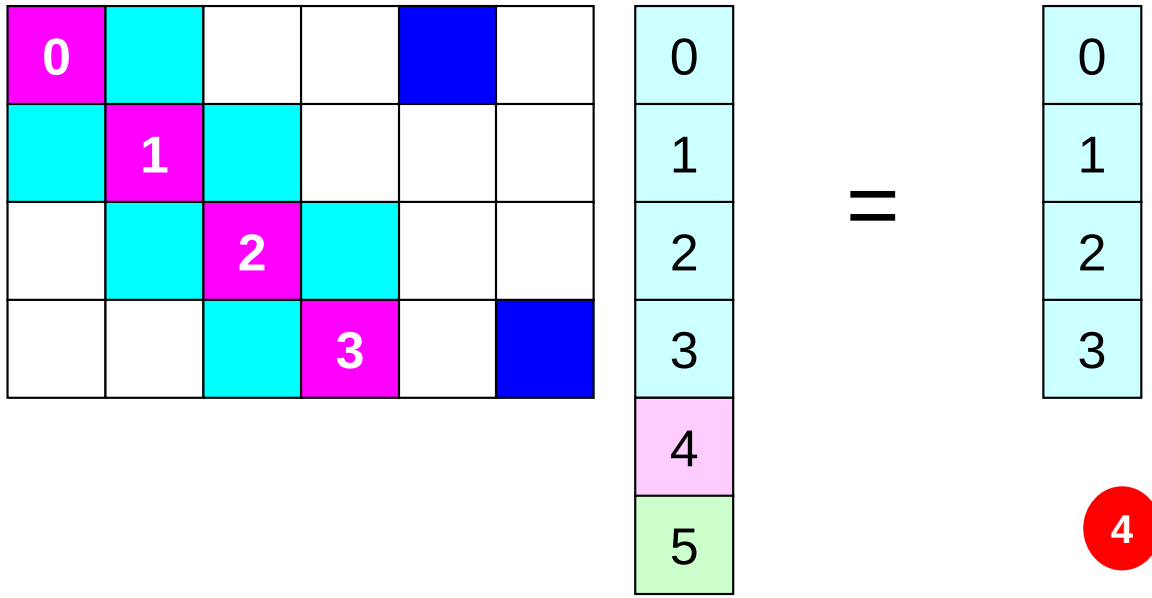


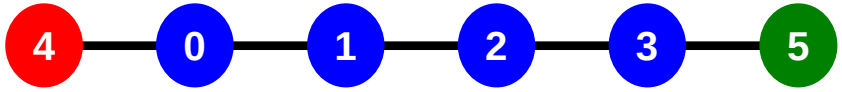
Mat-Vec Products: Local Op. Possible



Mat-Vec Products: Local Op. #1



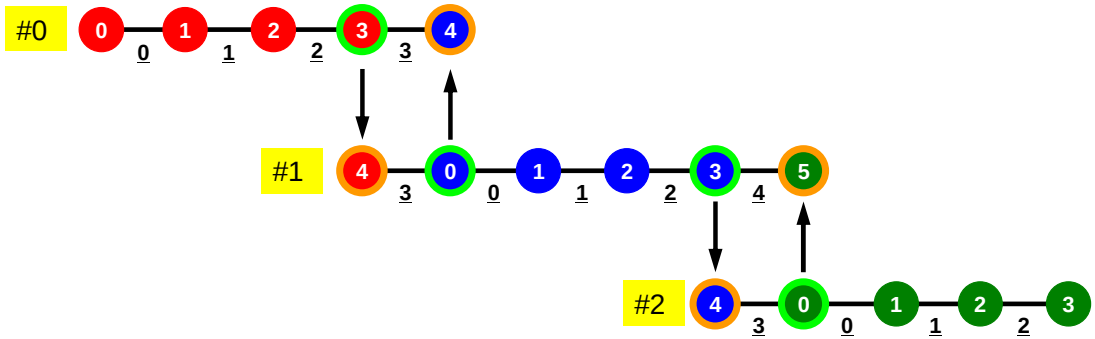




What is Point-to-Point Comm. ?

- 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

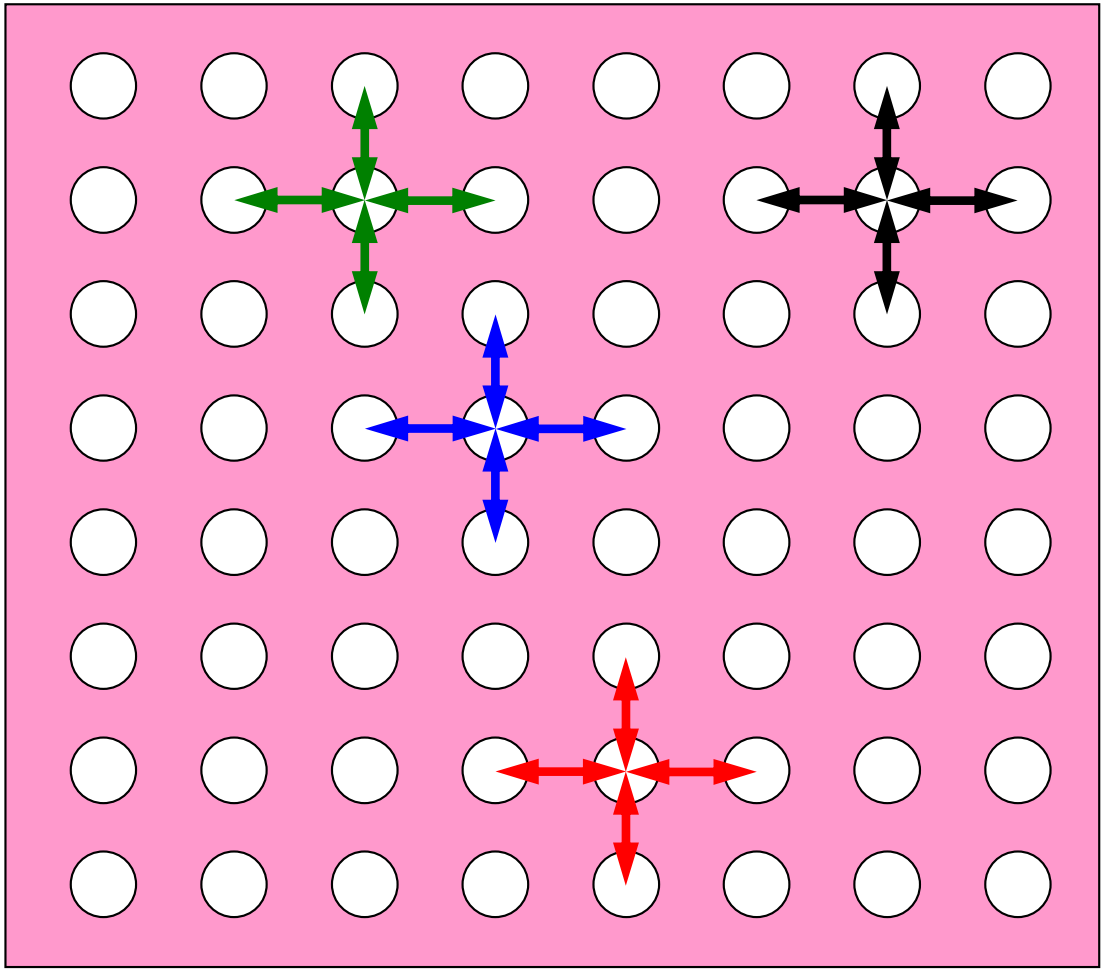
- Point-to-Point
  - MPI_Send, MPI_Recv
  - Communication with limited processes
    - Neighbors
  - Application Area
    - FEM, FDM: Localized Method



Collective/PtoP Communications

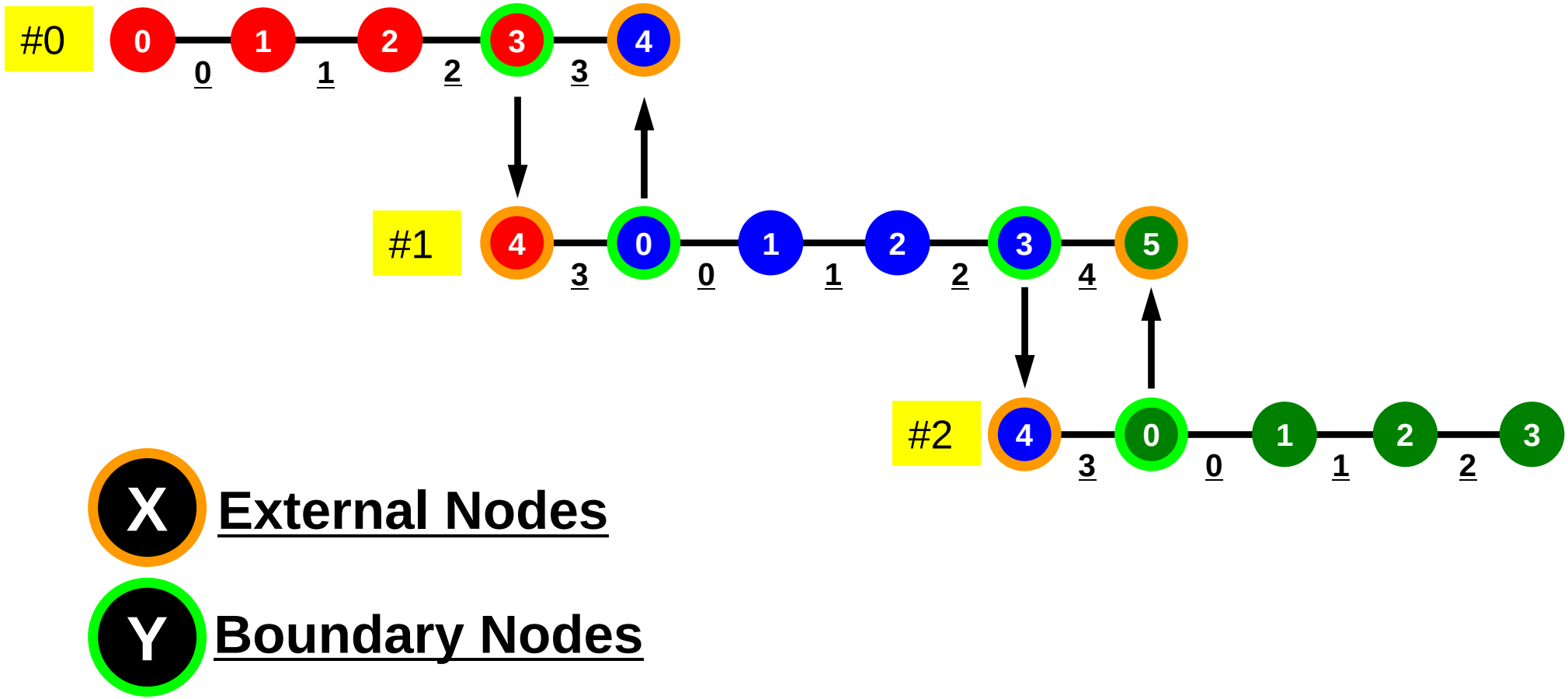
Interactions with only Neighboring Processes/Element

Finite Difference Method (FDM), Finite Element Method (FEM)



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

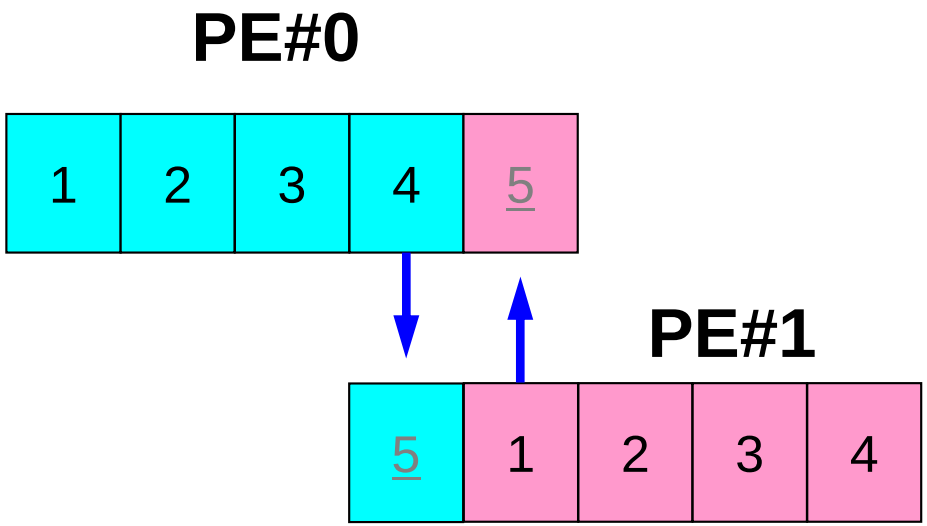
Info in neighboring domains is required for FEM operations
Matrix assembling, Iterative Method



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

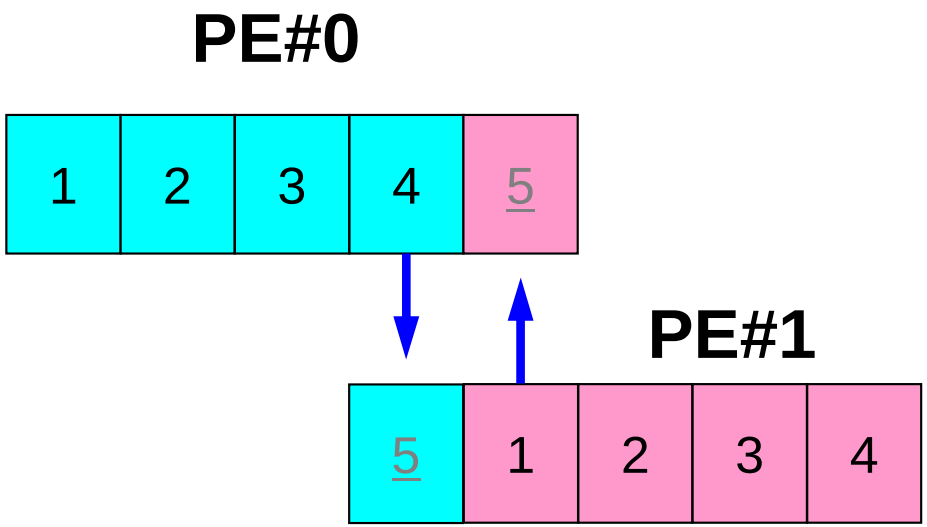
```
call MPI_SEND (NEIB_ID, arg's)
call MPI_RECV (NEIB_ID, arg's)
```

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.)

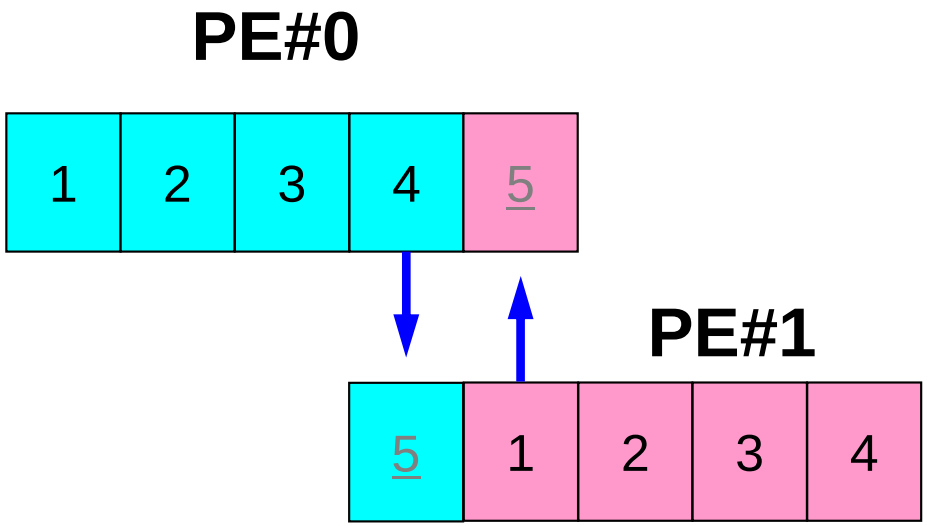



```
if (my_rank.eq.0) NEIB_ID=1
if (my_rank.eq.1) NEIB_ID=0

…
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

C

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**.

- **MPI_Isend (sendbuf,count,datatype,dest,tag,comm,request)**
  - **sendbuf** choice I starting address of sending buffer
  - **count** int I number of elements in sending buffer
  - **datatype** MPI_Datatype I datatype of each sending buffer element
  - **dest** int I rank of destination
  - **tag** int 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** MPI_Comm I communicator
  - **request** MPI_Request O communication request array used in `MPI_Waitall`

Communication Request: request 通信識別子

- **`MPI_Isend (sendbuf,count,datatype,dest,tag,comm,request)`**
  - **`sendbuf`** choice I — starting address of sending buffer
  - **`count`** int I — number of elements in sending buffer
  - **`datatype`** MPI_Datatype I — datatype of each sending buffer element
  - **`dest`** int I — rank of destination
  - **`tag`** int 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`** MPI_Comm I — communicator
  - **`request`** MPI_Request O — communication request used in `MPI_Waitall`
    Size of the array is total number of neighboring processes

- Just define the array

C

C

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`**.

- **`MPI_Irecv (recvbuf,count,datatype,source,tag,comm,request)`**
  - **`recvbuf`** choice I — starting address of receiving buffer
  - **`count`** int I — number of elements in receiving buffer
  - **`datatype`** MPI_Datatype I — datatype of each receiving buffer element
  - **`source`** int I — rank of source
  - **`tag`** int 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`** MPI_Comm I — communicator
  - **`request`** MPI_Request O — communication request array used in `MPI_Waitall`

MPI_Waitall

C

- **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.

- **MPI_Waitall (count,request,status)**
  - **count** int I number of processes to be synchronized
  - **request** MPI_Request I/O comm. request used in MPI_Isend/Irecv (array size: count)
  - **status** MPI_Status O array of status objects
    MPI_STATUS_SIZE: defined in 'mpif.h', 'mpi.h'

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

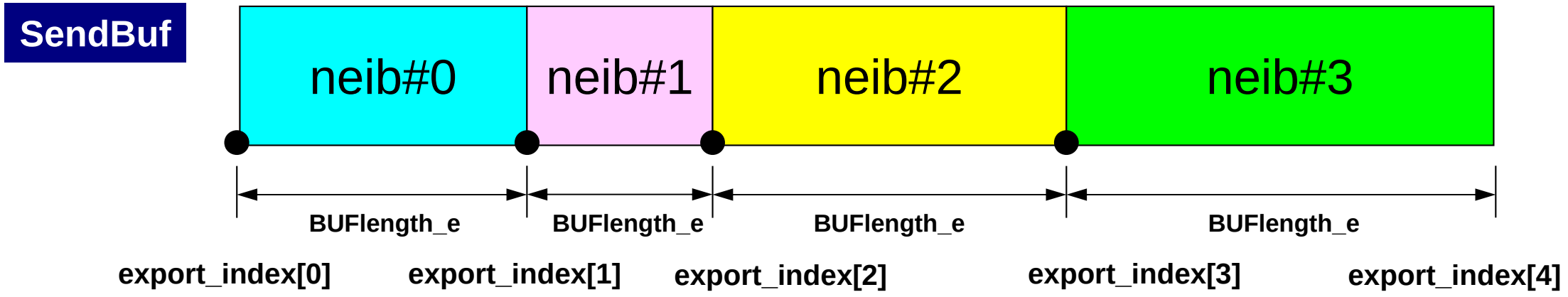
- **`MPI_Waitall (count,request,status)`**
  - **`count`** int `I` number of processes to be synchronized
  - **`request`** MPI_Request `I/O` comm. request used in `MPI_Isend/Irecv` (array size: `count`)
  - **`status`** MPI_Status `O` array of status objects
    `MPI_STATUS_SIZE: defined in ‘mpif.h’, ‘mpi.h’`

- Just define the array

C

SEND: MPI_Isend/Irecv/Waitall

C



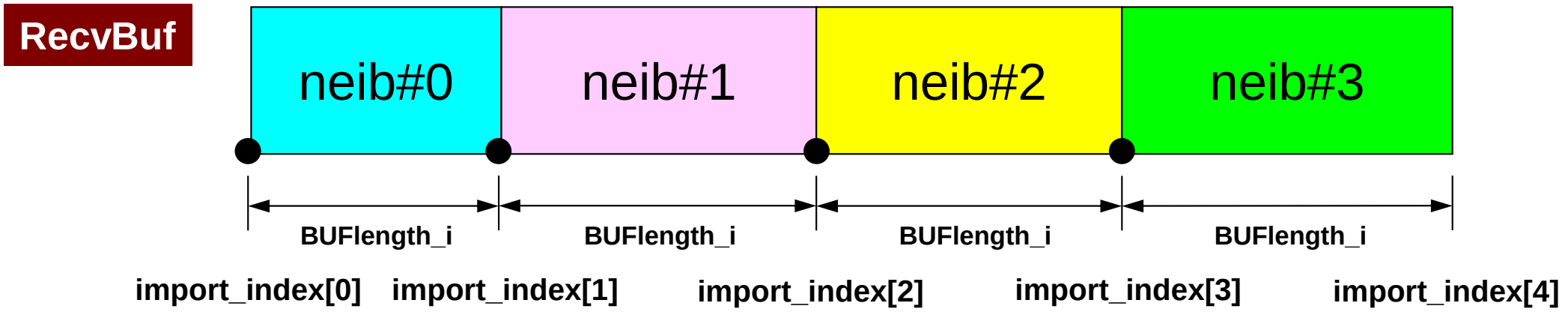

```
for (neib=0; neib<NeibPETot; neib++){
  tag= 0;
  iS_e= export_index[neib];
  iE_e= export_index[neib+1];
  BUFlength_e= iE_e - iS_e

  ierr= MPI_Isend
        (&SendBuf[iS_e], BUFlength_e, MPI_DOUBLE,
                NeibPE[neib], 0,
          MPI_COMM_WORLD, &ReqSend[neib])
}

MPI_Waitall(NeibPETot, ReqSend, StatSend);
```

RECV: MPI_Isend/Irecv/Waitall

C

```
for (neib=0; neib<NeibPETot; neib++){
   tag= 0;
   iS_i= import_index[neib];
   iE_i= import_index[neib+1];
   BUFlength_i= iE_i - iS_i

   ierr= MPI_Irecv
         (&RecvBuf[iS_i], BUFlength_i, MPI_DOUBLE,
           NeibPE[neib], 0,
           MPI_COMM_WORLD, &ReqRecv[neib])
}

MPI_Waitall(NeibPETot, ReqRecv, StatRecv);
```

import_item (import_index[neib]:import_index[neib+1]-1) are received from neib-th neighbor



C

MPI_Sendrecv

- MPI_Send+MPI_Recv: not recommended, many restrictions

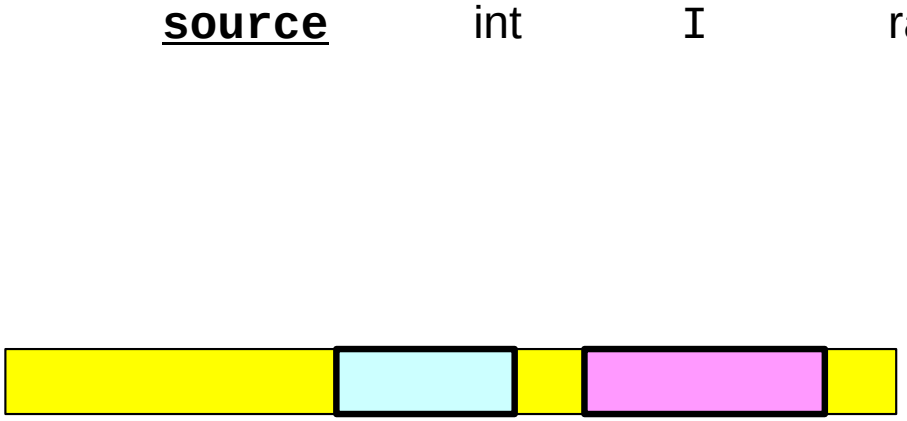
- **`MPI_Sendrecv (sendbuf,sendcount,sendtype,dest,sendtag,recvbuf, recvcount,recvtype,source,recvtag,comm,status)`**
  - **`sendbuf`** choice I starting address of sending buffer
  - **`sendcount`** int I number of elements in sending buffer
  - **`sendtype`** MPI_Datatype I datatype of each sending buffer element
  - **`dest`** int I rank of destination
  - **`sendtag`** int I message tag for sending
  - **`comm`** MPI_Comm I communicator
  - **`recvbuf`** choice I starting address of receiving buffer
  - **`recvcount`** int I number of elements in receiving buffer
  - **`recvtype`** MPI_Datatype I datatype of each receiving buffer element
  - **`source`** int I rank of source
  - **`recvtag`** int I message tag for receiving
  - **`comm`** MPI_Comm I communicator
  - **`status`** MPI_Status O array of status objects
    `MPI_STATUS_SIZE: defined in ‘mpif.h’, ‘mpi.h’`

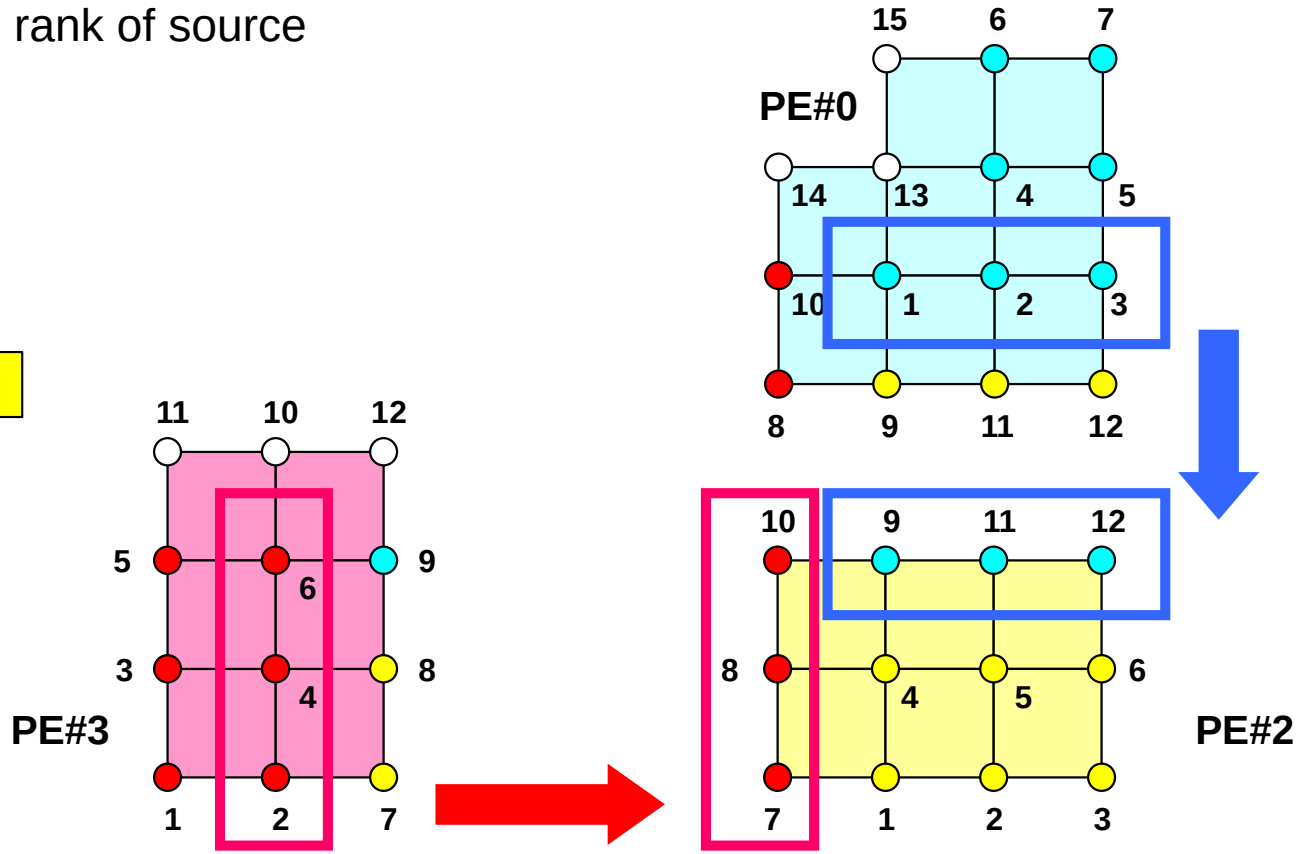
RECV: receiving to external nodes

Recv. continuous data to recv. buffer from neighbors

- **`MPI_Irecv`**
  `(recvbuf,count,datatype,source,tag,comm,request)`

`recvbuf`	choice	I	starting address of receiving buffer
`count`	int	I	number of elements in receiving buffer
`datatype`	MPI_Datatype	I	datatype of each receiving buffer element
`source`	int	I	rank of source



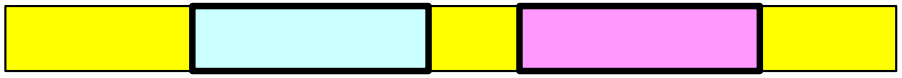


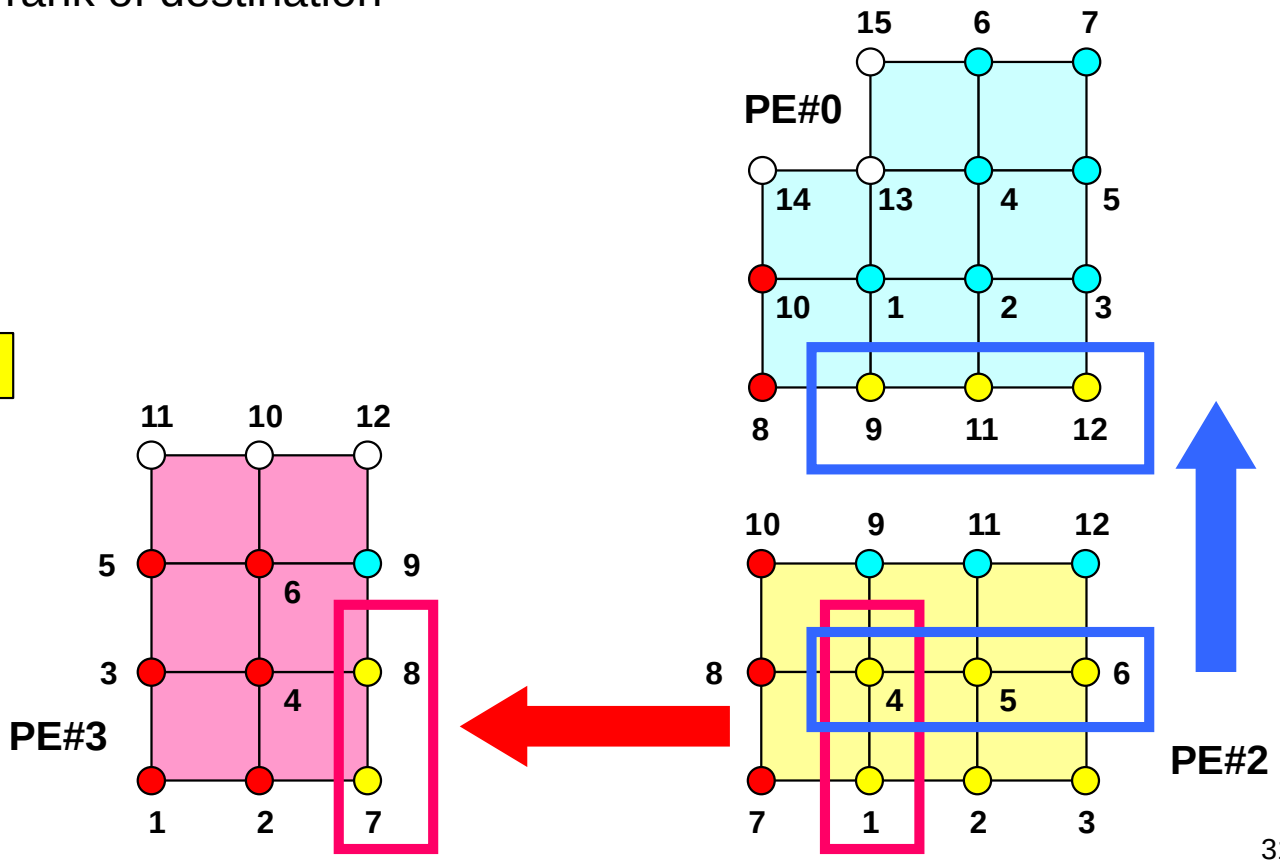
SEND: sending from boundary nodes

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`	int	I	number of elements in sending buffer
`datatype`	MPI_Datatype	I	datatype of each sending buffer element
`dest`	int	I	rank of destination





Request, Status in C Language Special TYPE of Arrays

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

```
MPI_Status  *StatSend, *StatRecv;
MPI_Request *RequestSend, *RequestRecv;
...
StatSend = malloc(sizeof(MPI_Status) * NEIBpetot);
StatRecv = malloc(sizeof(MPI_Status) * NEIBpetot);
RequestSend = malloc(sizeof(MPI_Request) * NEIBpetot);
RequestRecv = malloc(sizeof(MPI_Request) * NEIBpetot);
```

- **MPI_Sendrecv: status**

```
MPI_Status *Status;
...
Status = malloc(sizeof(MPI_Status));
```

Files on Odyssey

Fortan

```
>$ cd /work/gt36/t36XXX/pFEM/
>$ module load fj
>$ cp /work/gt00/z30088/pFEM/F/s2-f.tar .
>$ tar xvf s2-f.tar
```

C

```
>$ cd /work/gt36/t36XXX/pFEM/
>$ module load fj
>$ cp /work/gt00/home/z30088/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=0) NEIB= 1;
if (my_rank=1) NEIB= 0;

MPI_Isend (&VAL    ,1,MPI_DOUBLE,NEIB,…,req_send[0],…);
MPI_Irecv (&VALtemp,1,MPI_DOUBLE,NEIB,…,req_recv[0],…);
MPI_Waitall (…,req_recv,stat_recv,…); Recv.buf VALtemp can be used
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 /work/gt36/t36XXX/pFEM/mpi/S2
$> module load fj
$> mpifccpx –Nclang -Kfast ex1-1.c
$> pjsub go2.sh
```

```
#include <stdio.h>
#include <stdlib.h>
#include "mpi.h"
int main(int argc, char **argv){
        int neib, MyRank, PeTot;
        double VAL, VALx;
        MPI_Status *StatSend, *StatRecv;
        MPI_Request *RequestSend, *RequestRecv;

        MPI_Init(&argc, &argv);
        MPI_Comm_size(MPI_COMM_WORLD, &PeTot);
        MPI_Comm_rank(MPI_COMM_WORLD, &MyRank);
        StatSend = malloc(sizeof(MPI_Status) * 1);
        StatRecv = malloc(sizeof(MPI_Status) * 1);
        RequestSend = malloc(sizeof(MPI_Request) * 1);
        RequestRecv = malloc(sizeof(MPI_Request) * 1);

        if(MyRank == 0) {neib= 1; VAL= 10.0;}
        if(MyRank == 1) {neib= 0; VAL= 11.0;}

        MPI_Isend(&VAL , 1, MPI_DOUBLE, neib, 0, MPI_COMM_WORLD, &RequestSend[0]);
        MPI_Irecv(&VALx, 1, MPI_DOUBLE, neib, 0, MPI_COMM_WORLD, &RequestRecv[0]);
        MPI_Waitall(1, RequestRecv, StatRecv);
        MPI_Waitall(1, RequestSend, StatSend);

        VAL=VALx;
        MPI_Finalize();
        return 0; }
```

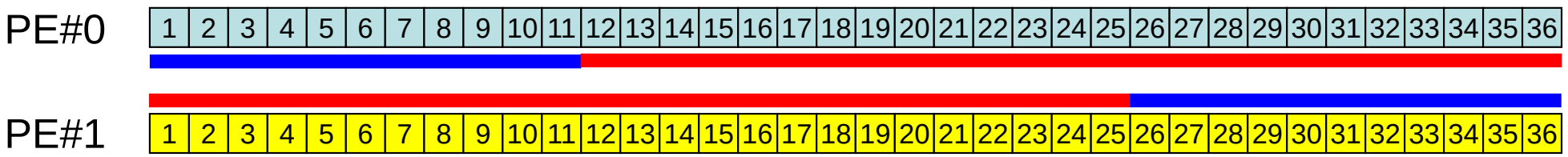
go2.sh

```
#!/bin/sh
#PJM -N "test"
#PJM -L rscgrp=lecture6-o
#PJM -L node=1
#PJM --mpi proc=2
#PJM -L elapse=00:15:00
#PJM -g gt36
#PJM -j
#PJM -e err
#PJM -o test.lst

module load fj
module load fjmpi

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[0-35]= 101,102,103,~,135,136
  - PE#1 VEC[0-35]= 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=0) then
  MPI_Isend (&VEC[ 0],11,MPI_DOUBLE,1,…,req_send[0],…)
  MPI_Irecv (&VEC[11],25,MPI_DOUBLE,1,…,req_recv[0],…)
endif

if (my_rank=1) then
  MPI_Isend (&VEC[ 0],25,0,…,req_send[0],…)
  MPI_Irecv (&VEC[25],11,0,…,req_recv[0],…)
endif

MPI_Waitall (…,req_recv,stat_recv,…)
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=0) then
  NEIB= 1
   start_send=   1
  length_send=  11
   start_recv=  length_send + 1
  length_recv=  25
endif

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

MPI_Isend
(&VEC[start_send-1],length_send,MPI_DOUBLE,NEIB,…,req_send[0],…)
MPI_Irecv
(&VEC[start_recv-1],length_recv,MPI_DOUBLE,NEIB,…,req_recv[0],…)

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

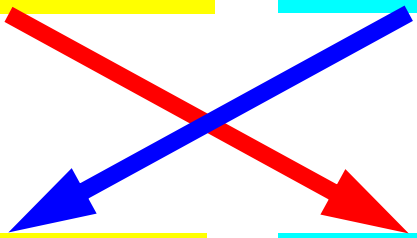
This is “SMPD” !!

t1

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)
```

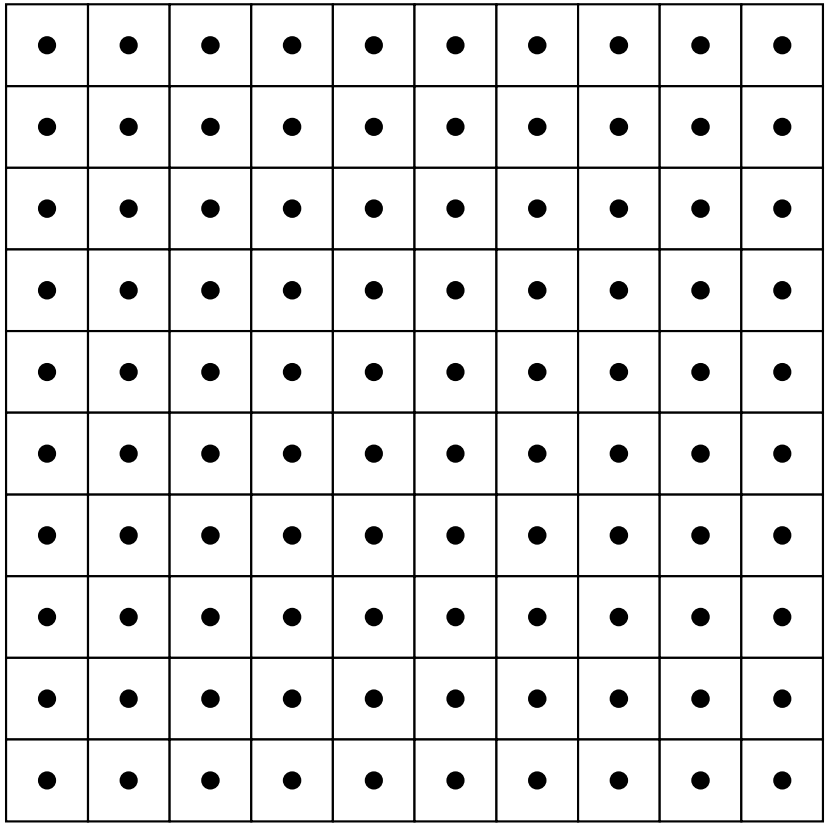
- “length_send” of sending process must be equal to “length_recv” of receiving process.
  - PE#0 to PE#1, PE#1 to PE#0
- “sendbuf” and “recvbuf”: different address

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

2D FDM (1/5)

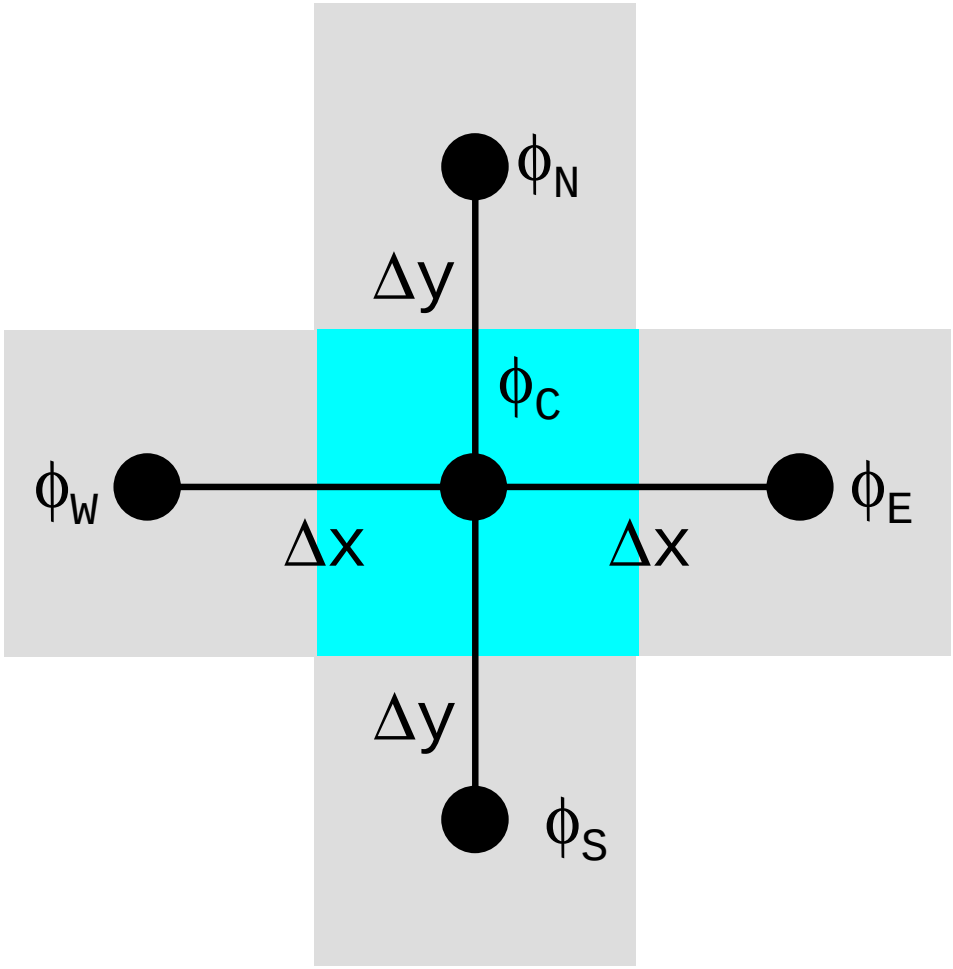
Entire Mesh



2D FDM (5-point, central difference)

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

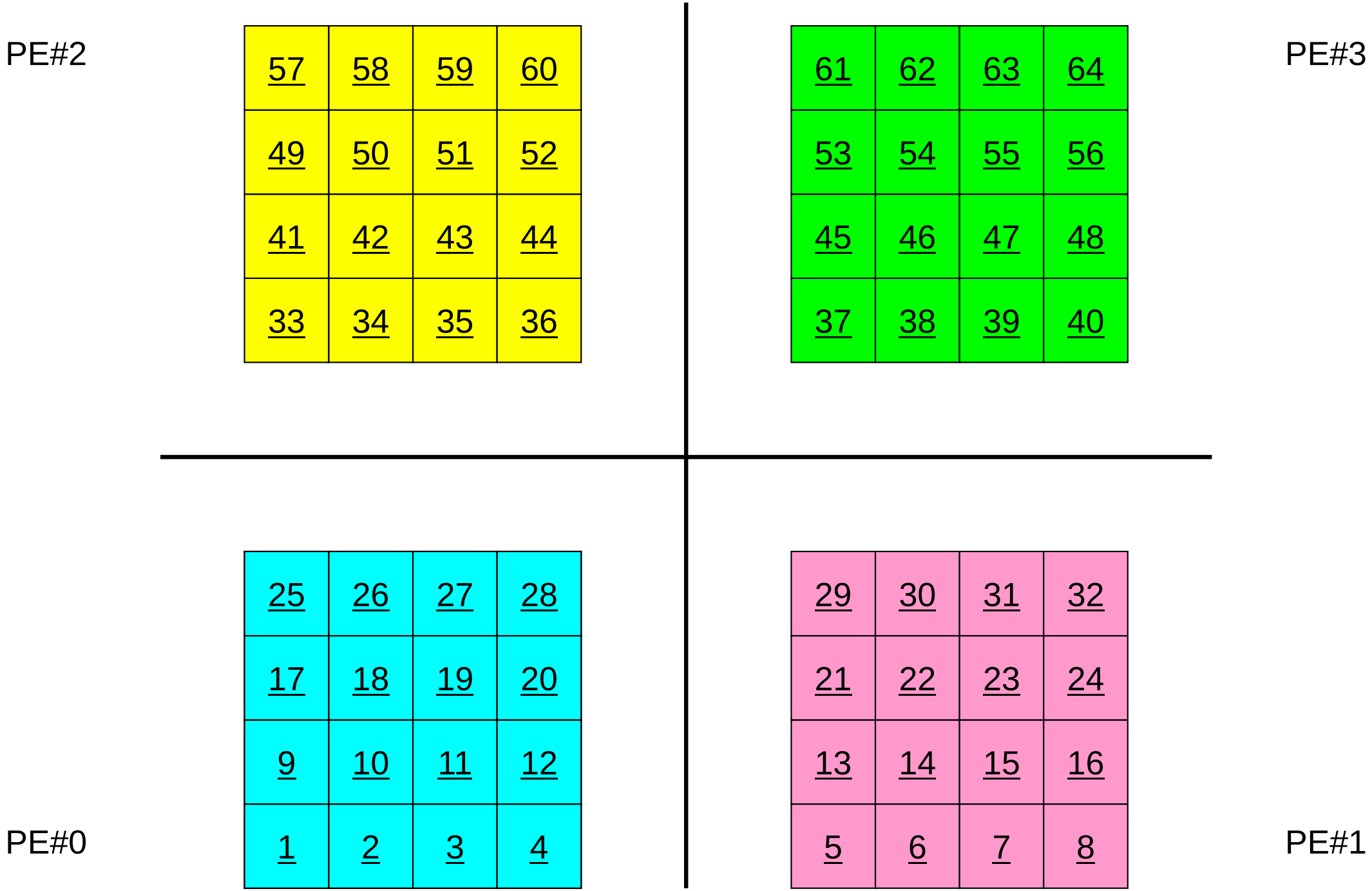
$$\left(\frac{\phi_E - 2\phi_C + \phi_W}{\Delta x^2} \right) + \left(\frac{\phi_N - 2\phi_C + \phi_S}{\Delta y^2} \right) = f_C$$



Decompose into 4 domains

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

4 domains: Global ID



4 domains: Local ID

PE#2

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

PE#3

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

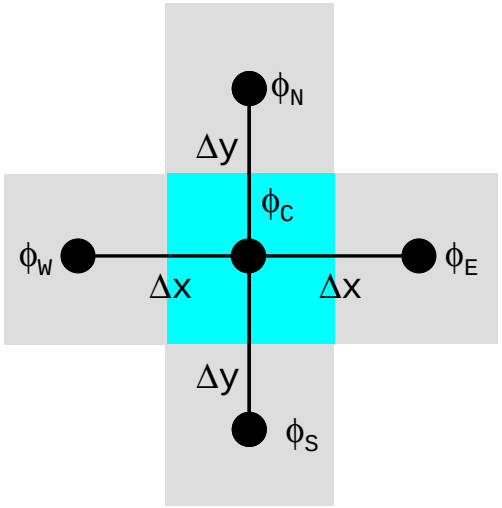
PE#0

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

PE#1

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

External Points: Overlapped Region



PE#2

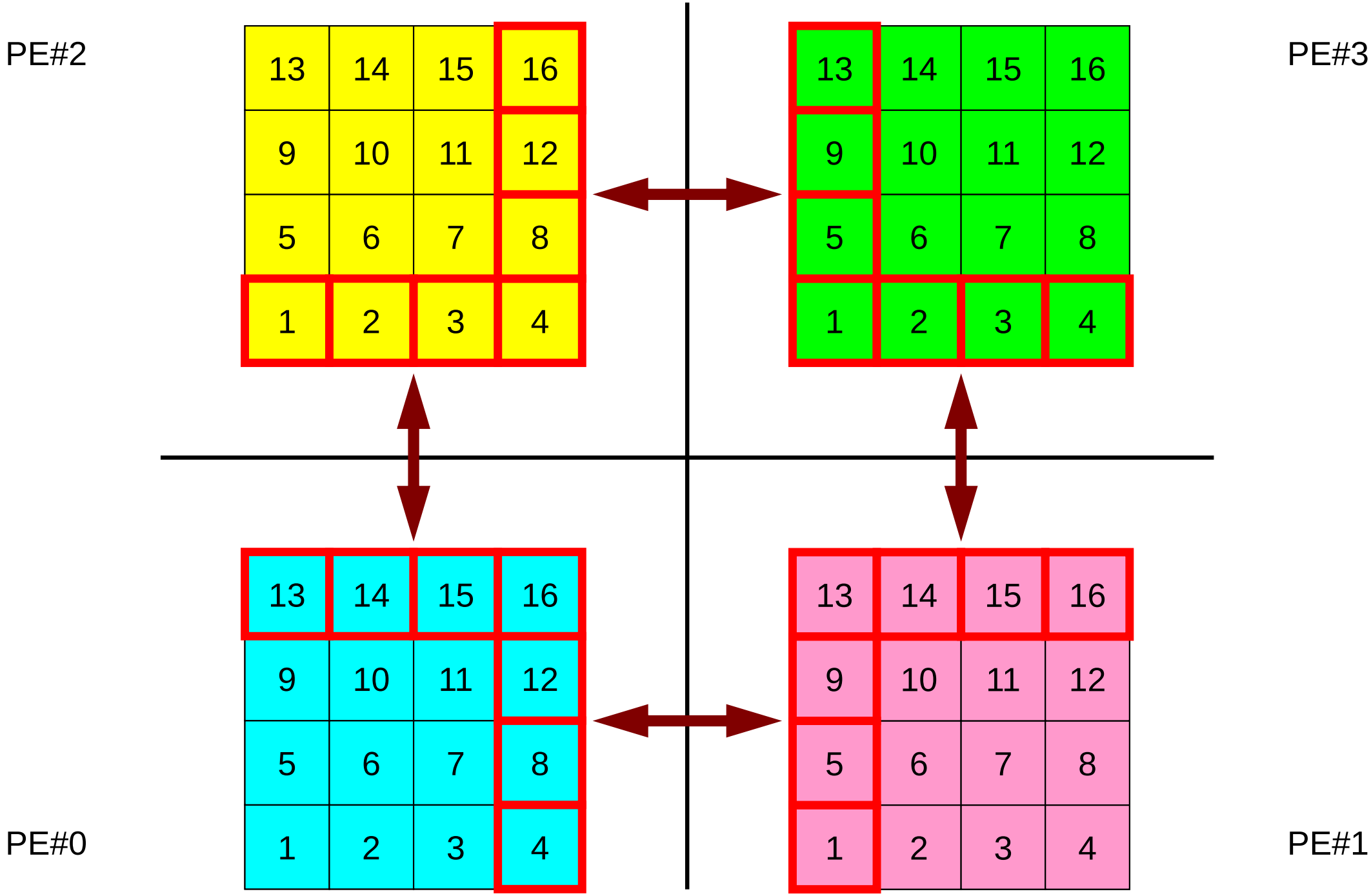
PE#3

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

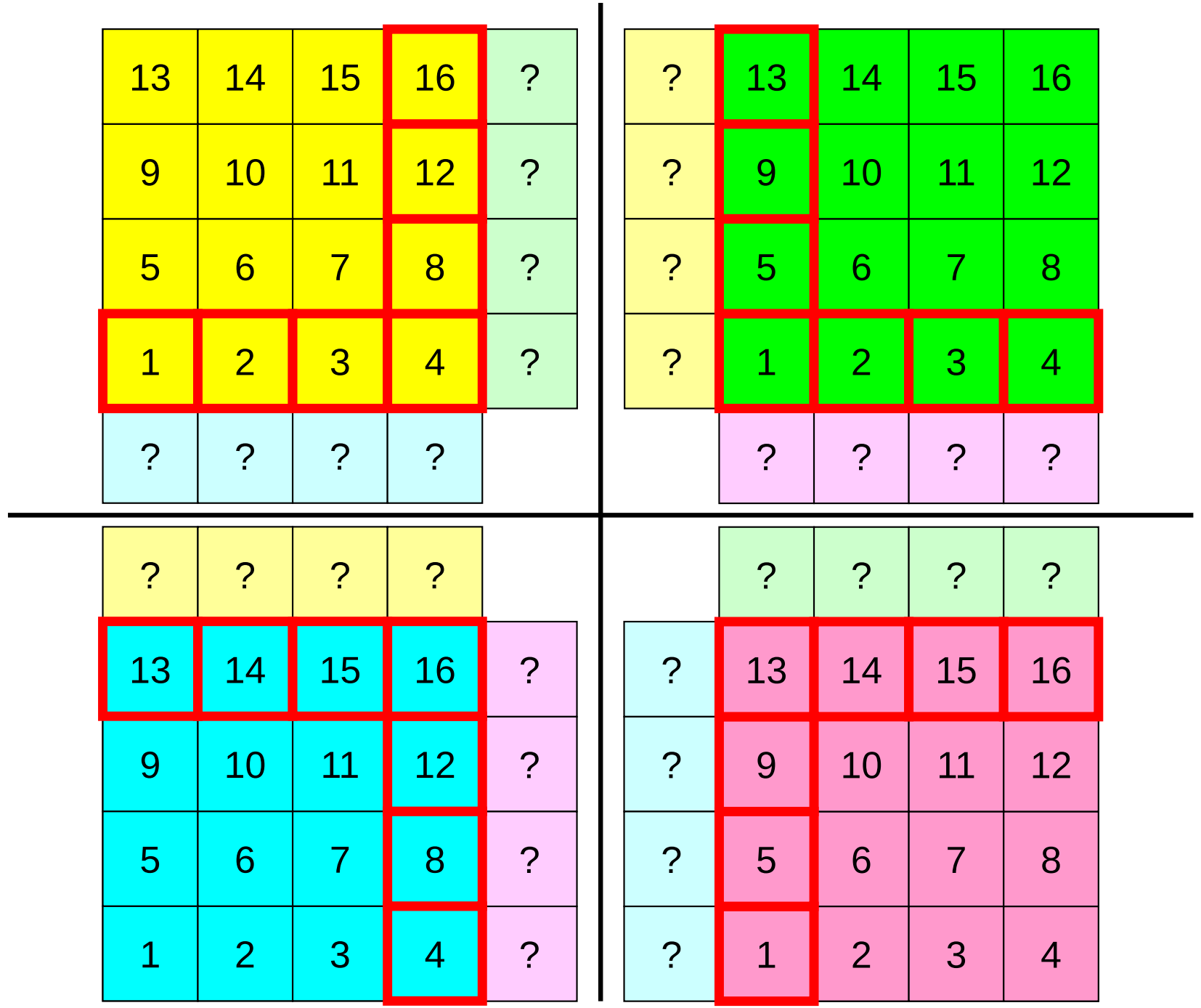
PE#0

PE#1

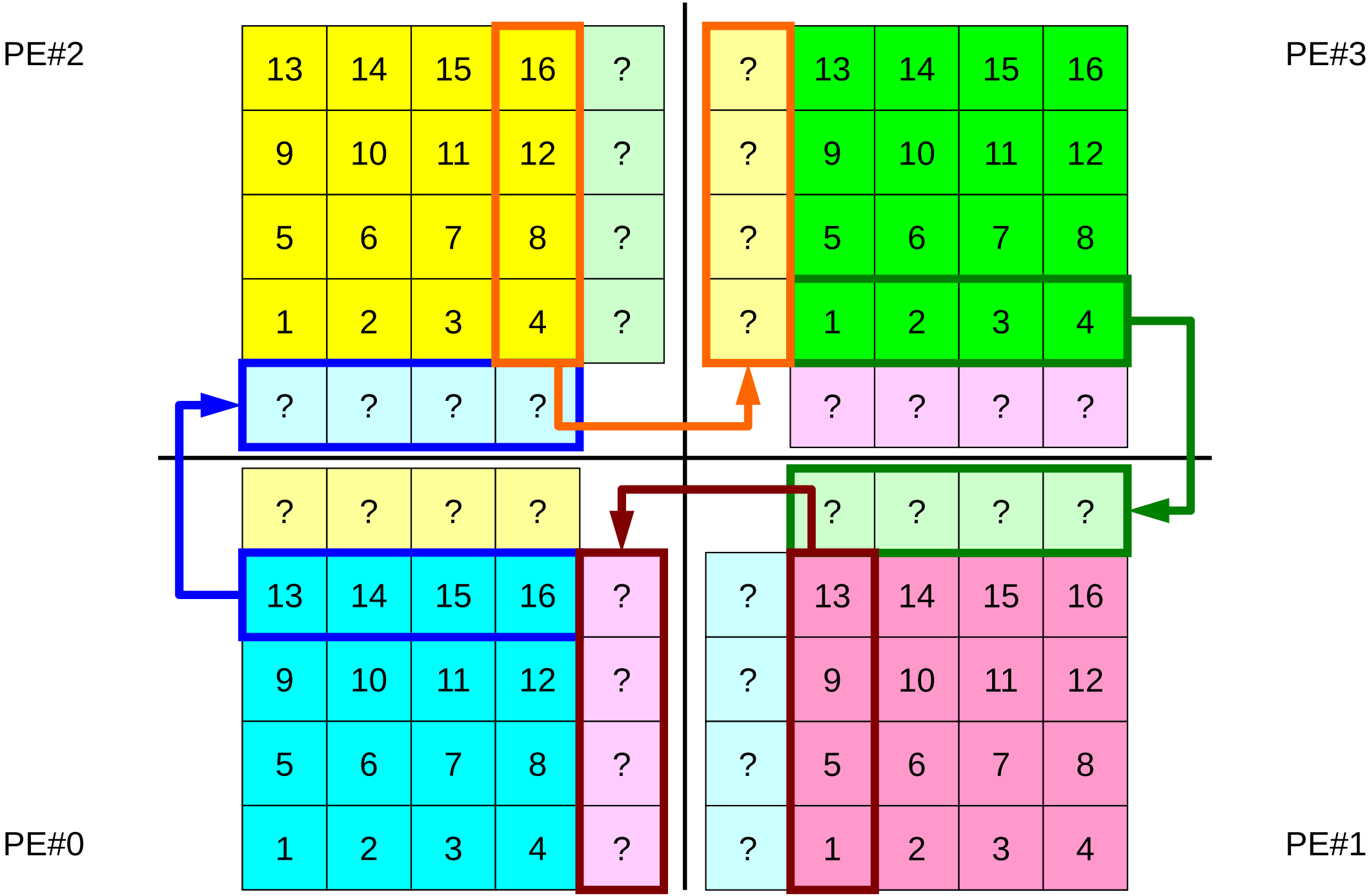
External Points: Overlapped Region



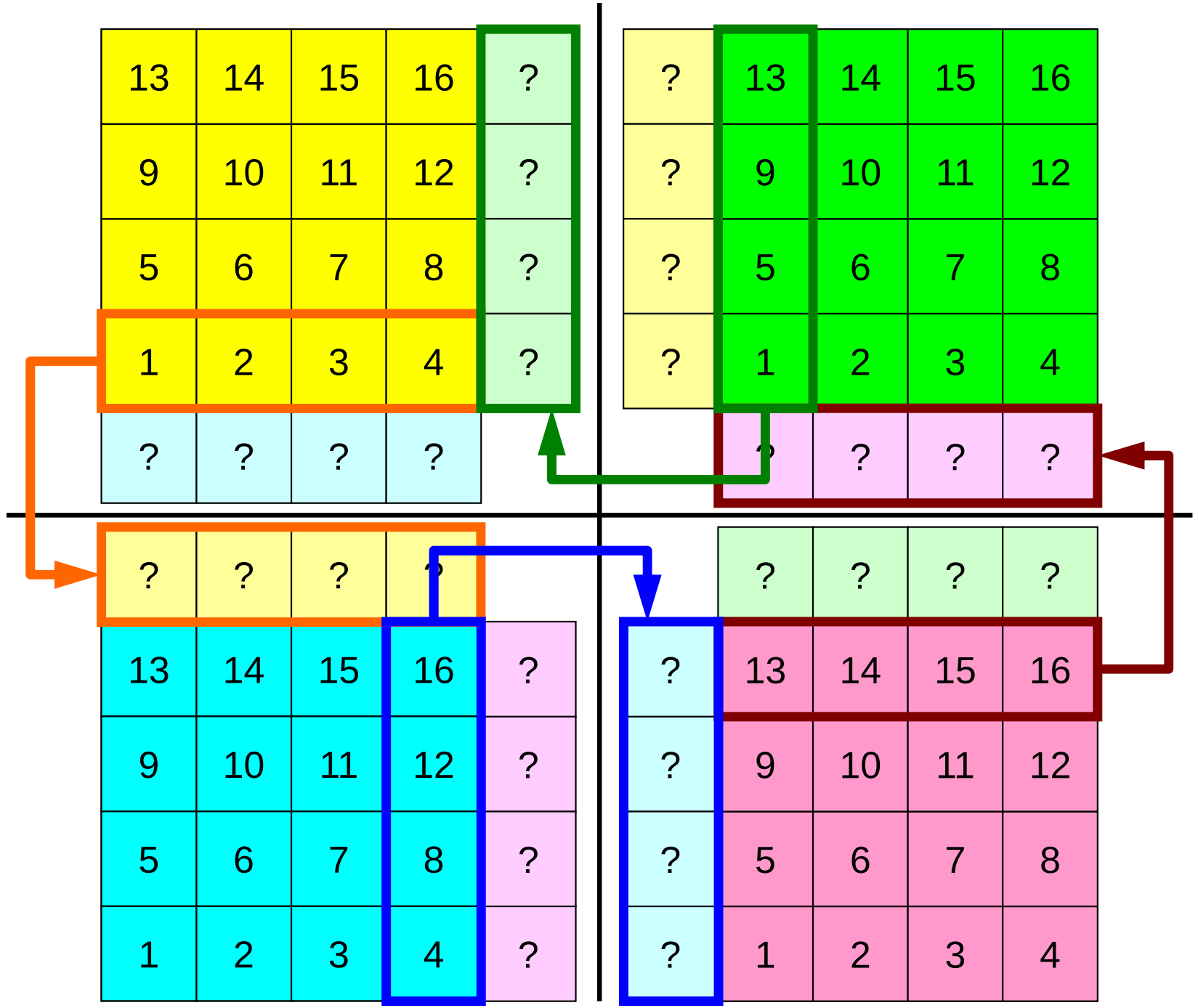
Local ID of External Points ?



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

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

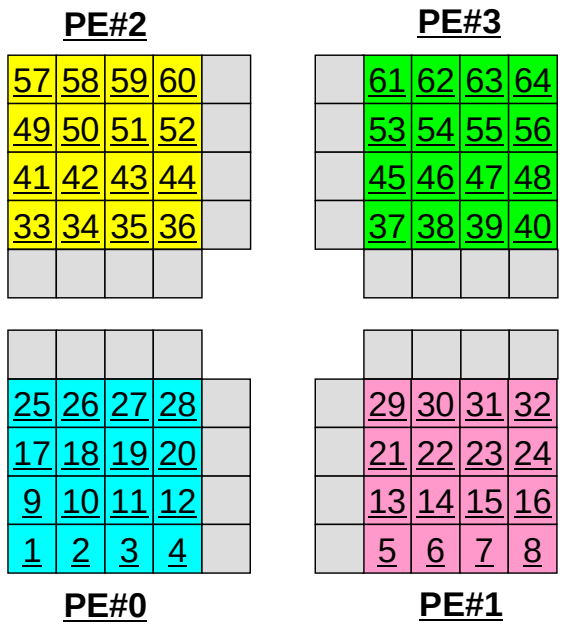
PE#0

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

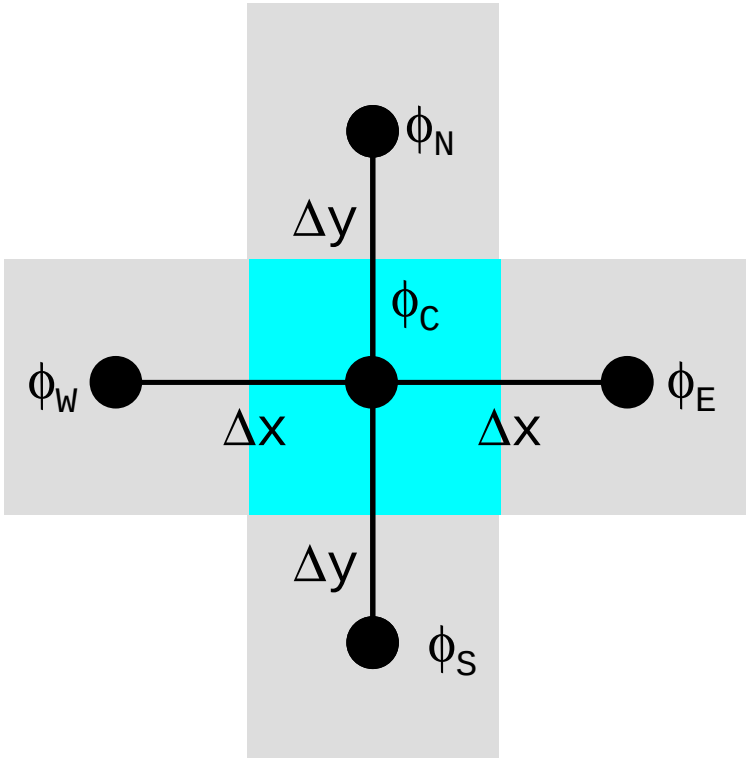
- 4 sub-domains.
- Info. of external points (global ID of mesh) is received from neighbors.
  - PE#0 receives □



Operations of 2D FDM

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

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

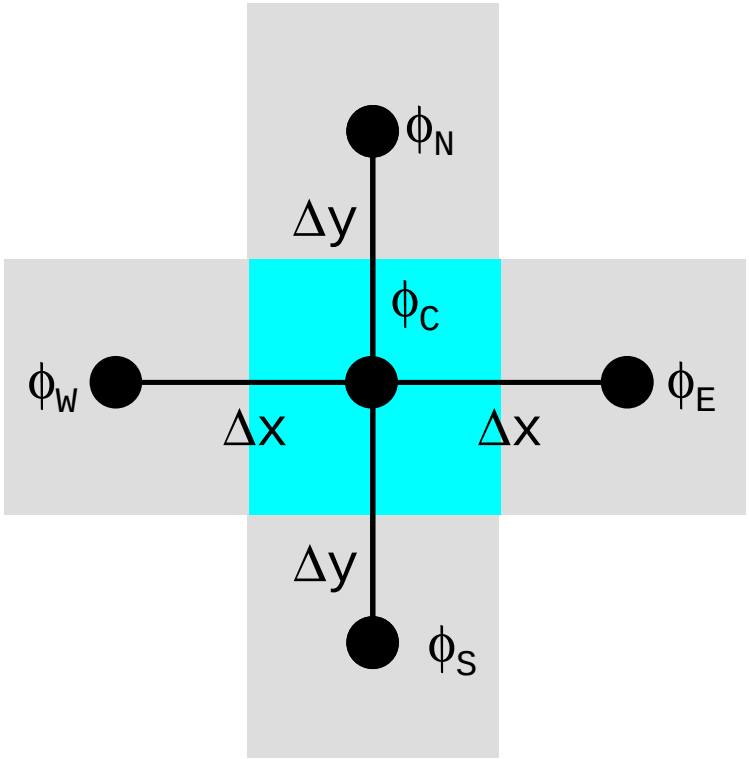


57	58	59	60	61	62	63	64
49	50	51	52	53	54	55	56
41	42	43	44	45	46	47	48
33	34	35	36	37	38	39	40
25	26	27	28	29	30	31	32
17	18	19	20	21	22	23	24
9	10	11	12	13	14	15	16
1	2	3	4	5	6	7	8

Operations of 2D FDM

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

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



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

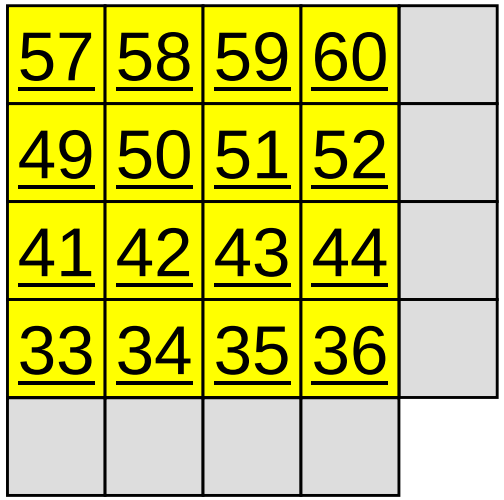
Computation (1/3)

- 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

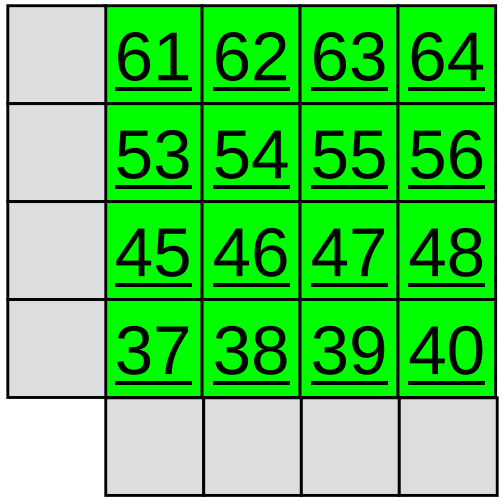
PE#2

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: ?

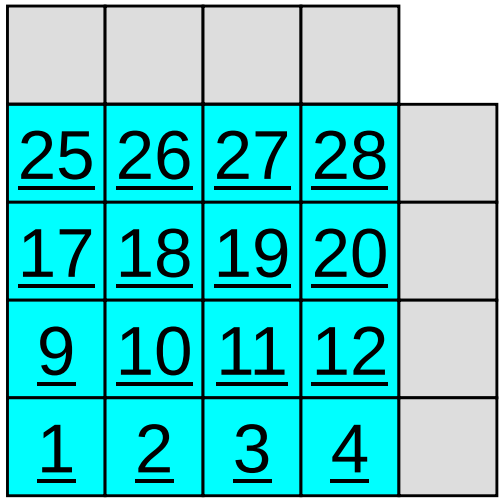


PE#3

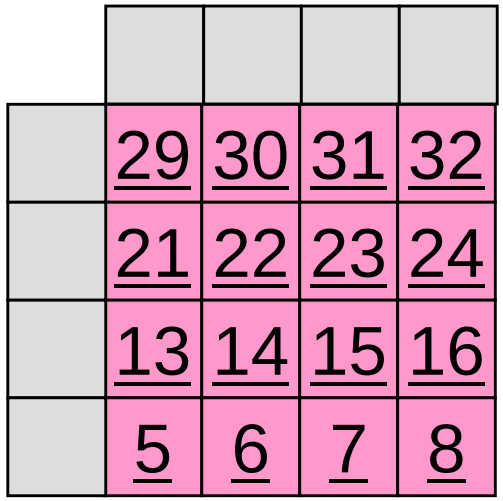
1: 37	9: 53	17: ?
2: 38	10: 54	18: ?
3: 39	11: 55	19: ?
4: 40	12: 56	20: ?
5: 45	13: 61	21: ?
6: 46	14: 62	22: ?
7: 47	15: 63	23: ?
8: 48	16: 64	24: ?



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: ?



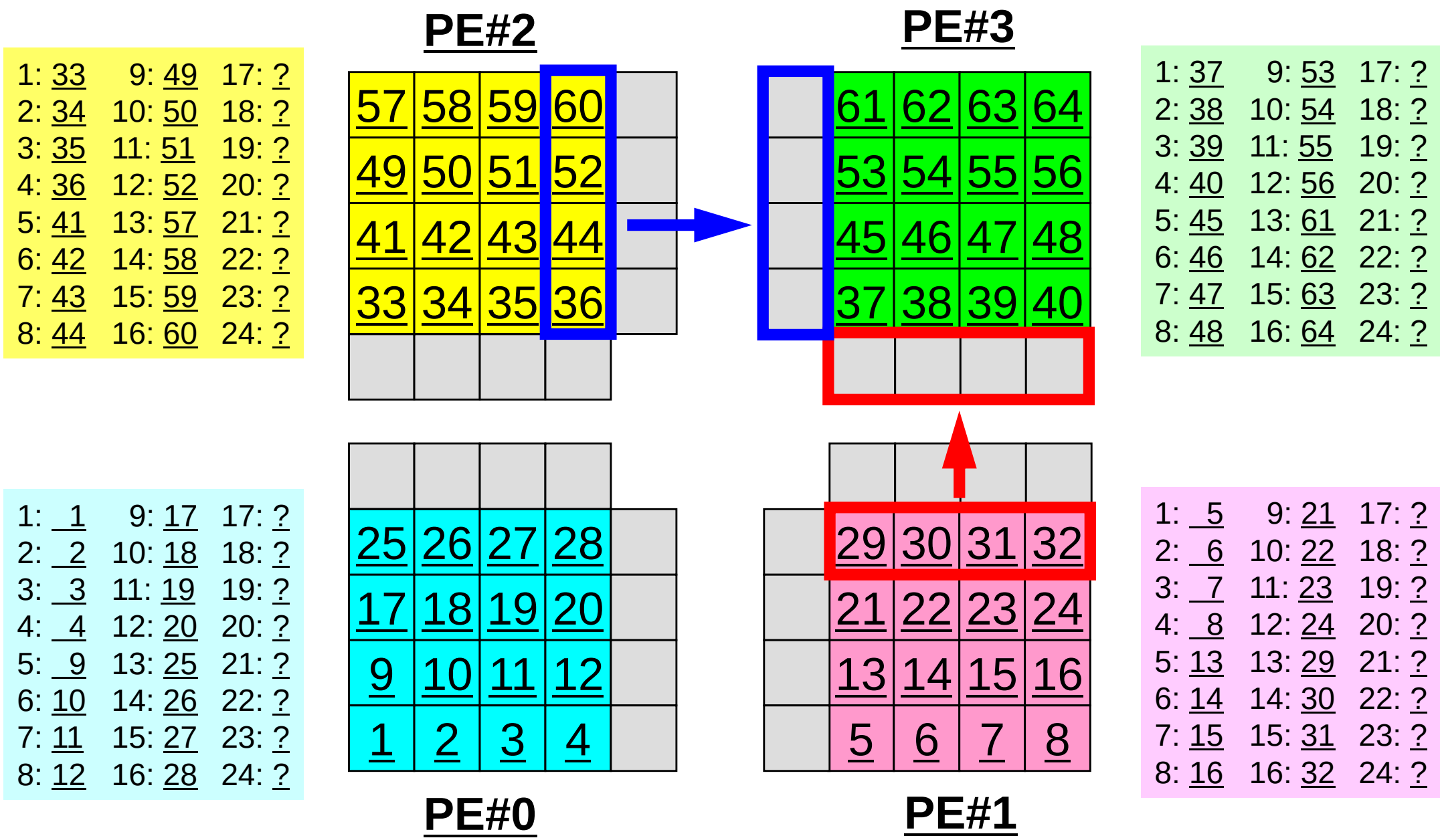
PE#0



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: ?

Computation (2/3): Before Send/Recv



Computation (3/3): After Send/Recv

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

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

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

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

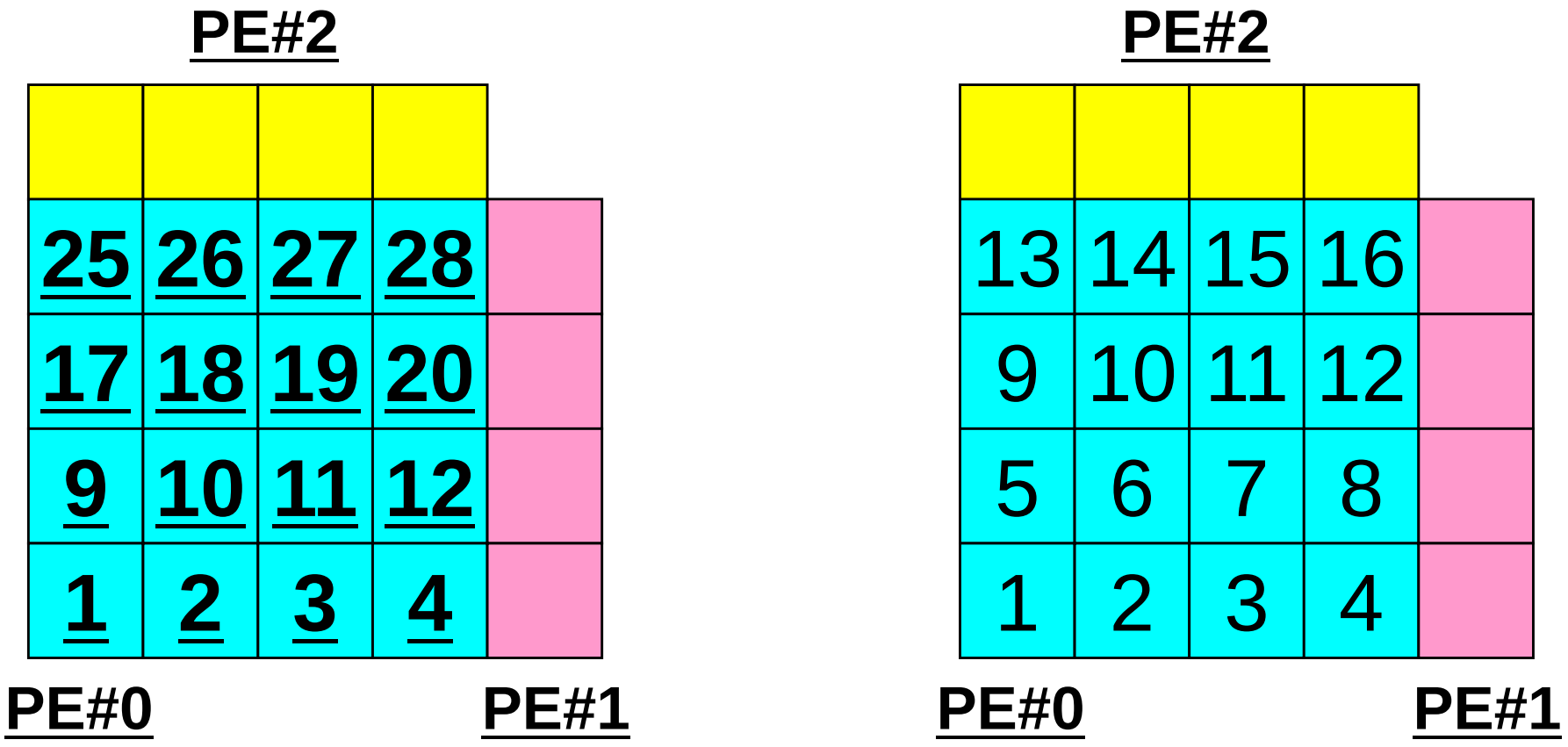
PE#1

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

Overview of Distributed Local Data

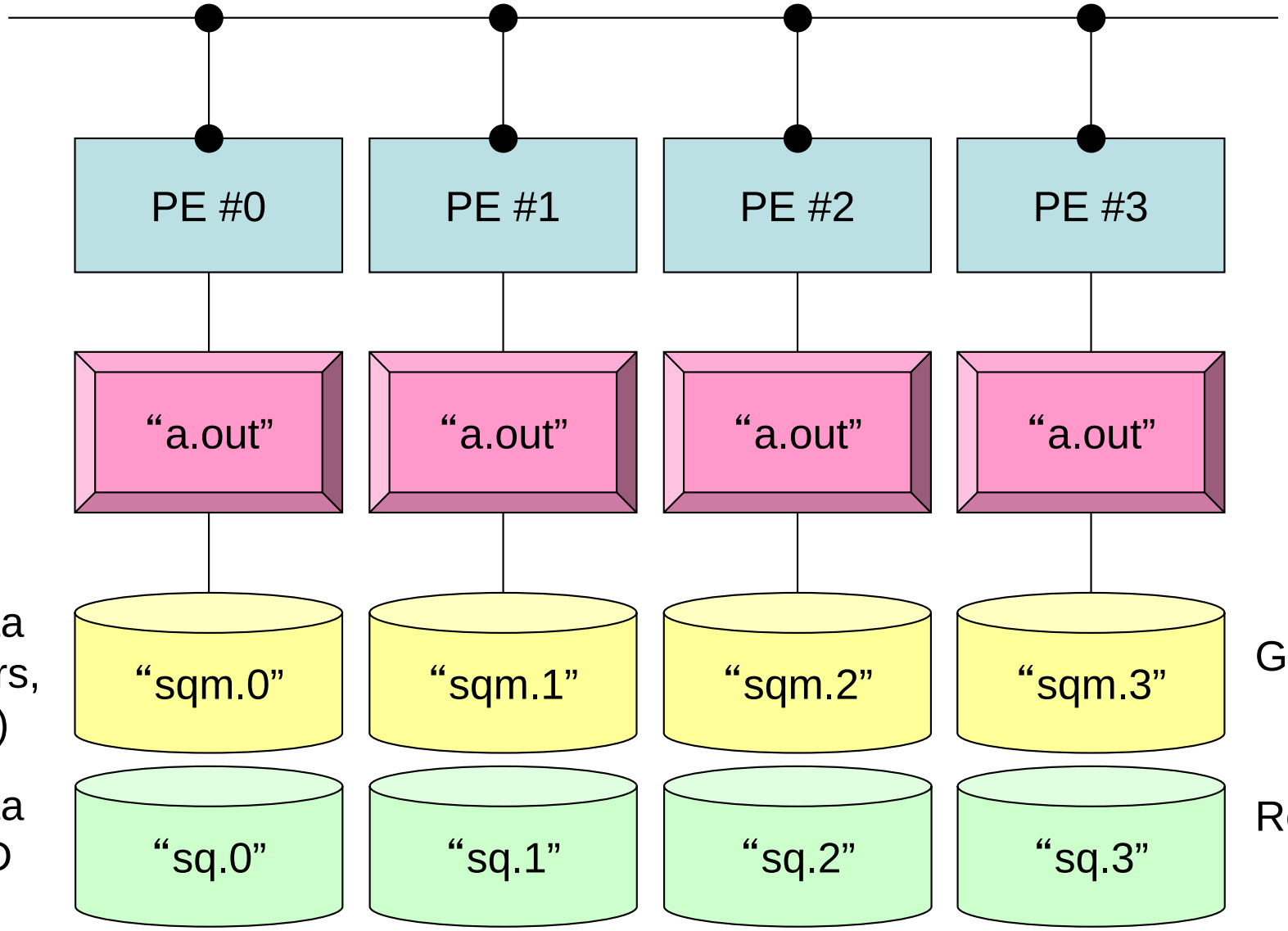
Example on PE#0



Value at each mesh (= Global ID)

Local ID

SPMD・・・



Dist. Local Data Sets (Neighbors, Comm. Tables)

Geometry

Dist. Local Data Sets (Global ID of Internal Points)

Results

2D FDM: PE#0

Information at each domain (1/4)

Internal Points

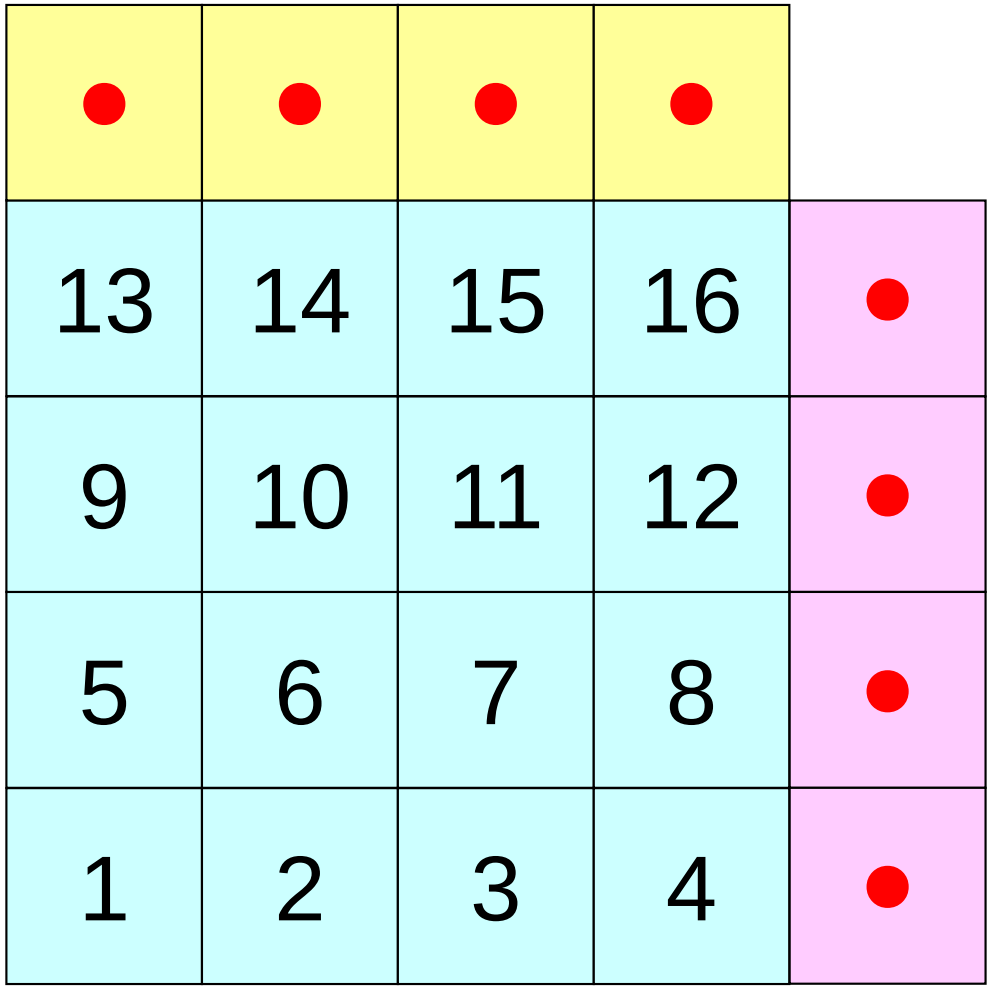
Meshes originally assigned to the domain

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

2D FDM: PE#0

Information at each domain (2/4)

PE#2



PE#1

Internal Points

Meshes originally assigned to the domain

External Points

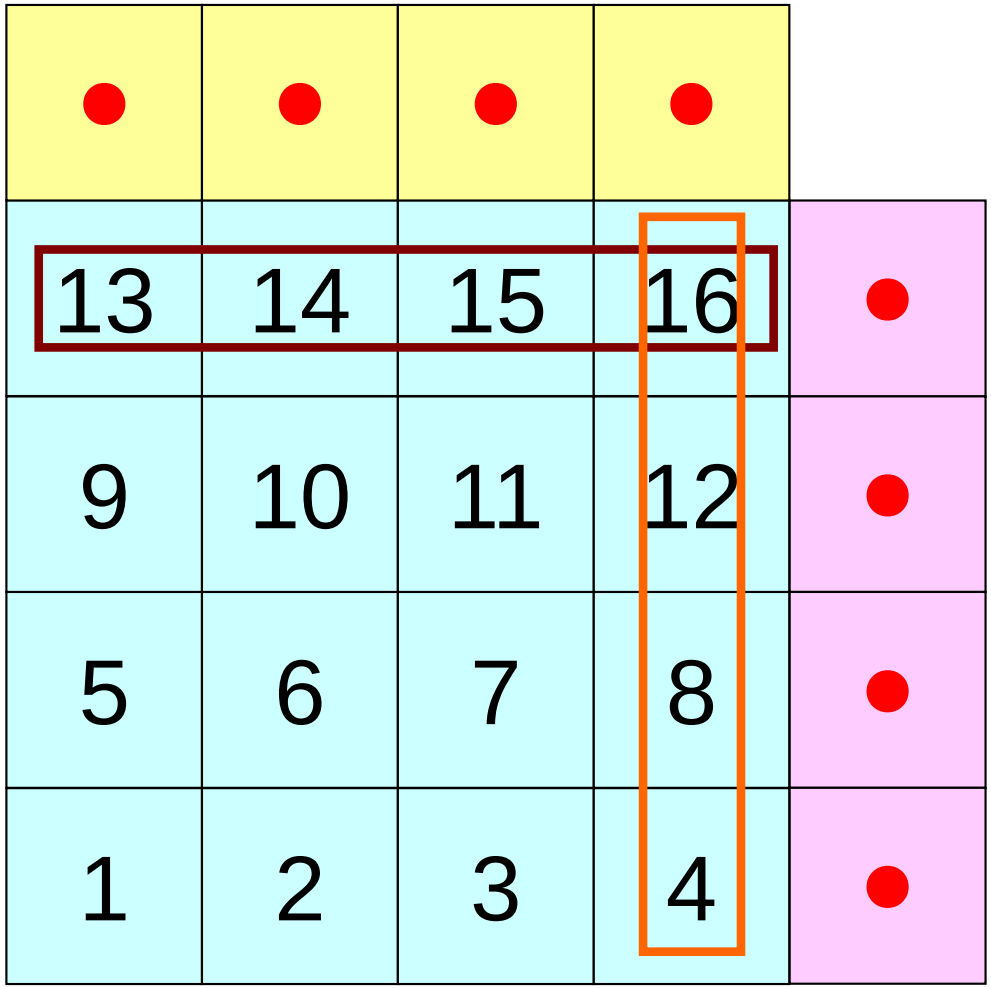
Meshes originally assigned to different domain, but required for computation of meshes in the domain (meshes in overlapped regions)

- Sleeves
- Halo

2D FDM: PE#0

Information at each domain (3/4)

PE#2



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)

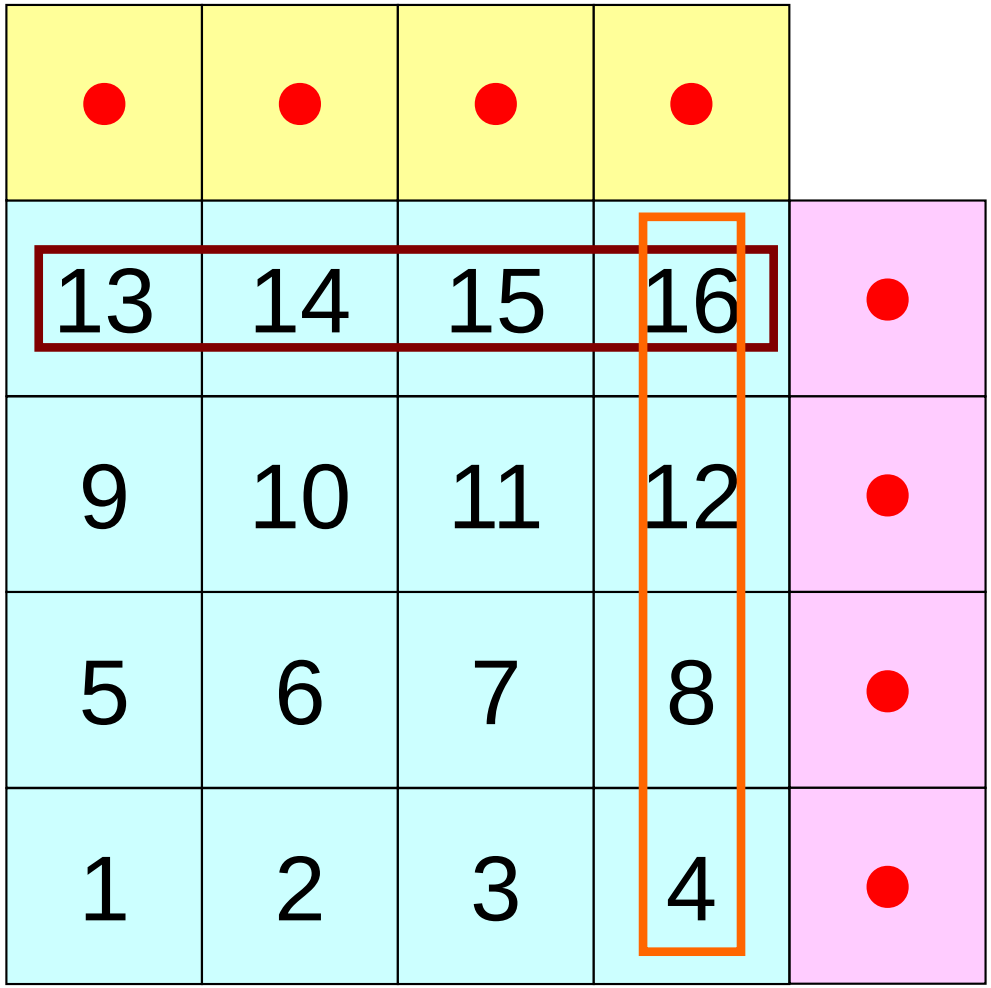
Boundary Points

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)

PE#2



PE#1

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)

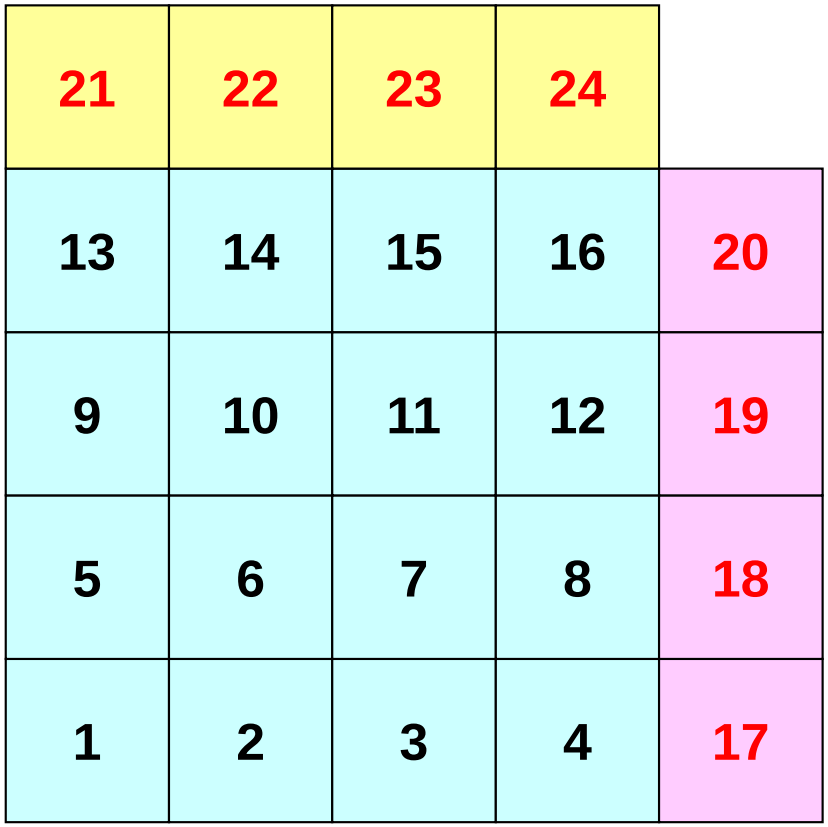
Boundary Points

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

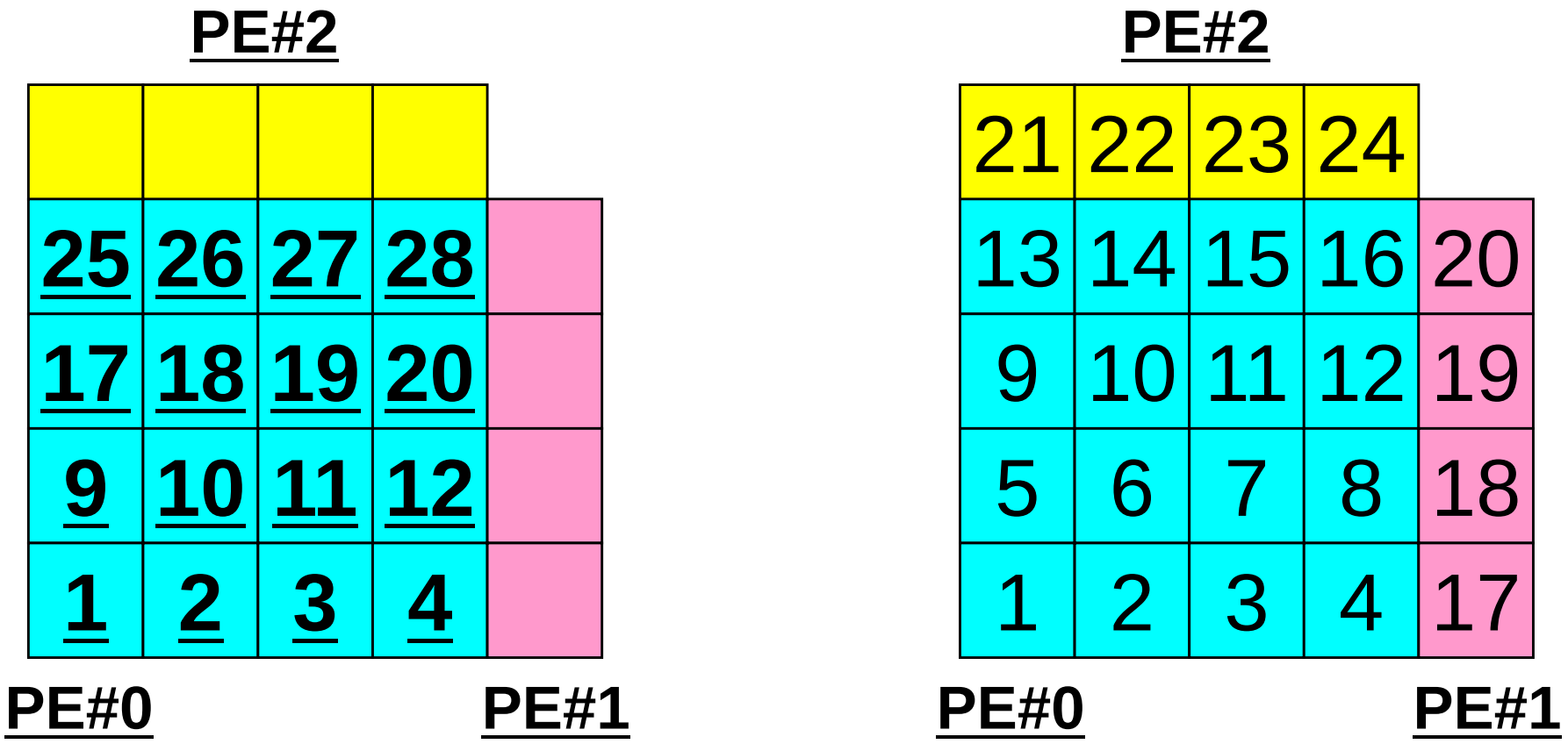
Description of Distributed Local Data



- 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

0	1.1 ◎	2.4 ①,0	3.2 ④,1		
1	3.6 ①	4.3 ◎,2	2.5 ③,3	3.7 ⑤,4	9.1 ⑦,5
2	5.7 ②	1.5 ④,6	3.1 ⑥,7		
3	9.8 ③	4.1 ①,8	2.5 ④,9	2.7 ⑤,10	
4	11.5 ④	3.1 ◎,11	9.5 ①,12	10.4 ②,13	4.3 ⑥,14
5	12.4 ⑤	6.5 ②,15	9.5 ⑥,16		
6	23.1 ⑥	6.4 ①,17	2.5 ②,18	1.4 ⑤,19	13.1 ⑦,20
7	51.3 ⑦	9.5 ①,21	1.3 ②,22	9.6 ③,23	3.1 ⑤,24

Diag [i]	Diagonal Components (REAL, i=0~N-1)
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=0, index[N]）
Amat[k]	Off-Diagonal Components (Value) （ REAL, k=0, index[N] ）

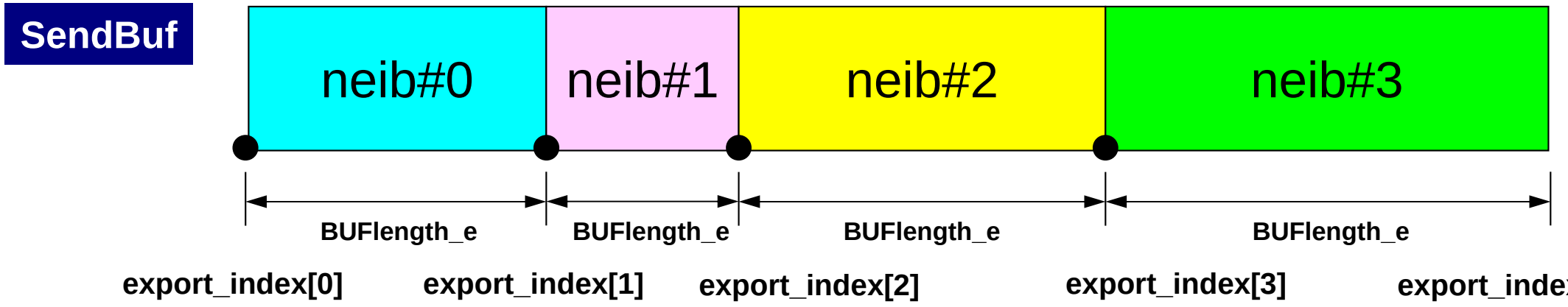
```
{Y}=[A]{X}

for(i=0;i<N;i++){
    Y[i] = Diag[i] * X[i];
    for(k=Index[i];k<Index[i+1];k++){
        Y[i] += AMat[k]*X[Item[k]];
    }
}
```

- Neighbors
  - NeibPETot, NeibPE[NeibPETot]
- Message size for each neighbor
  - export_index[NeibPETot+1]
- ID of **boundary** points
  - export_item[export_index[NeibPETot]]
- Messages to each neighbor
  - SendBuf[export_index[NeibPETot]

C

SEND: MPI_Isend/Irecv/Waitall

C



export_item (export_index[neib]:export_index[neib+1]-1) are sent to neib-th neighbor

```
for (neib=0; neib<NeibPETot;neib++){
   for (k=export_index[neib];k<export_index[neib+1];k++){
      kk= export_item[k];
      SendBuf[k]= VAL[kk];
   }
}

for (neib=0; neib<NeibPETot; neib++){
  tag= 0;
  iS_e= export_index[neib];
  iE_e= export_index[neib+1];
  BUFlength_e= iE_e - iS_e

  ierr= MPI_Isend
        (&SendBuf[iS_e], BUFlength_e, MPI_DOUBLE, NeibPE[neib], 0,
         MPI_COMM_WORLD, &ReqSend[neib])
}

MPI_Waitall(NeibPETot, ReqSend, StatSend);
```

Copied to sending buffers

Generalized Comm. Table: Receive

- Neighbors
  - NeibPETot, NeibPE[NeibPETot]
- Message size for each neighbor
  - import_index [NeibPETot+1]
- ID of **external** points
  - import_item [import_index[NeibPETot]]
- Messages from each neighbor
  - RecvBuf [import_index[NeibPETot]]

C

RECV: MPI_Isend/Irecv/Waitall

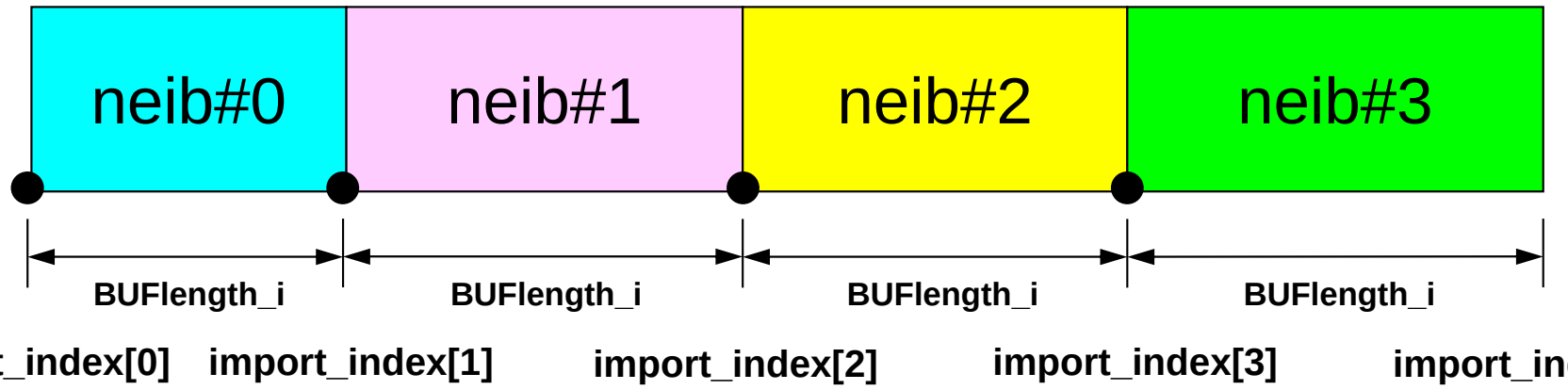
C

```
for (neib=0; neib<NeibPETot; neib++){
   tag= 0;
   iS_i= import_index[neib];
   iE_i= import_index[neib+1];
   BUFlength_i= iE_i - iS_i

   ierr= MPI_Irecv
         (&RecvBuf[iS_i], BUFlength_i, MPI_DOUBLE, NeibPE[neib], 0,
          MPI_COMM_WORLD, &ReqRecv[neib])
}

MPI_Waitall(NeibPETot, ReqRecv, StatRecv);

for (neib=0; neib<NeibPETot;neib++){
    for (k=import_index[neib];k<import_index[neib+1];k++){
       kk= import_item[k];
       VAL[kk]= RecvBuf[k];
    }
}
```

Copied from receiving buffer

import_item (import_index[neib]:import_index[neib+1]-1) are received from neib-th neighbor

RecvBuf



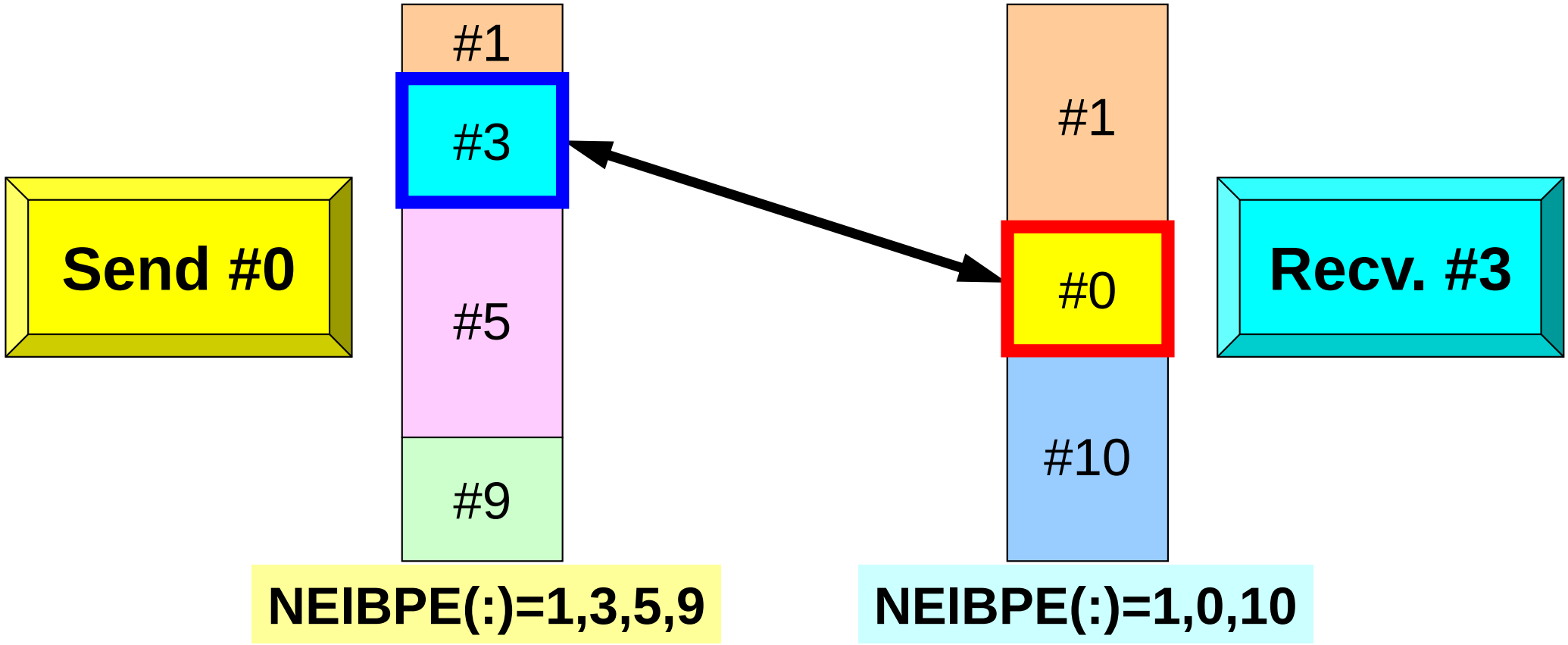
Relationship SEND/RECV

```
 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

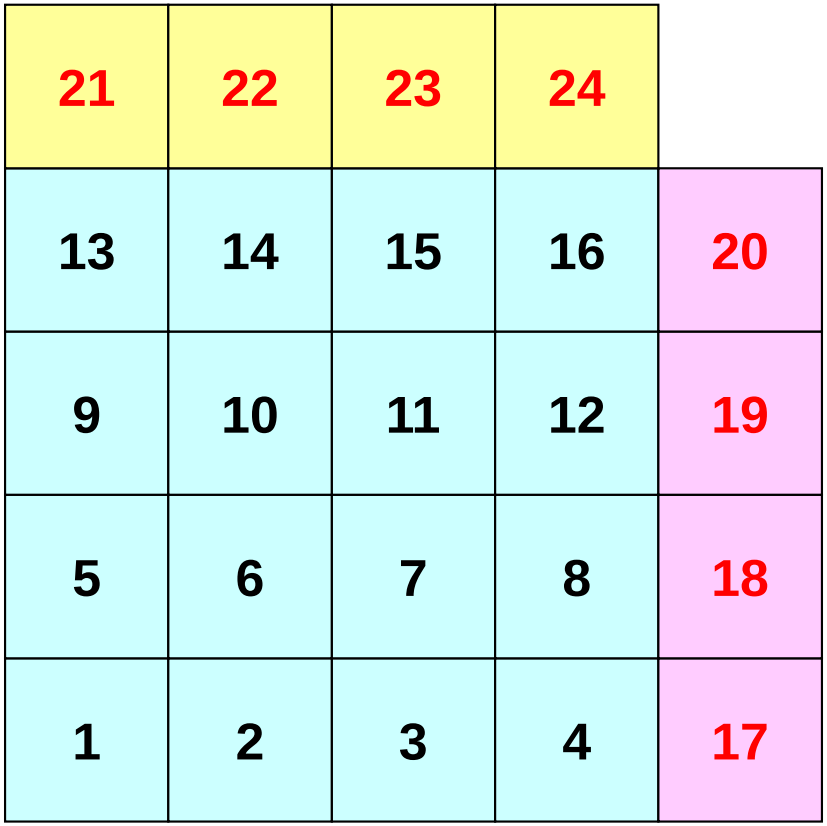
Relationship SEND/RECV (#0 to #3)



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

Generalized Comm. Table (1/6)

PE#2

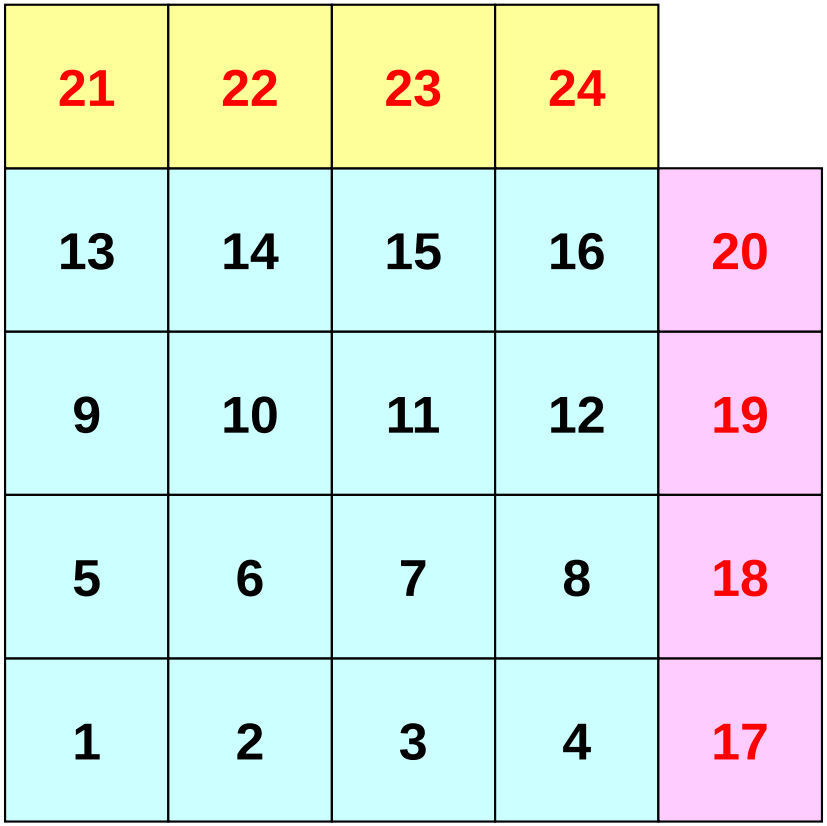


PE#1

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

Generalized Comm. Table (2/6)

PE#2

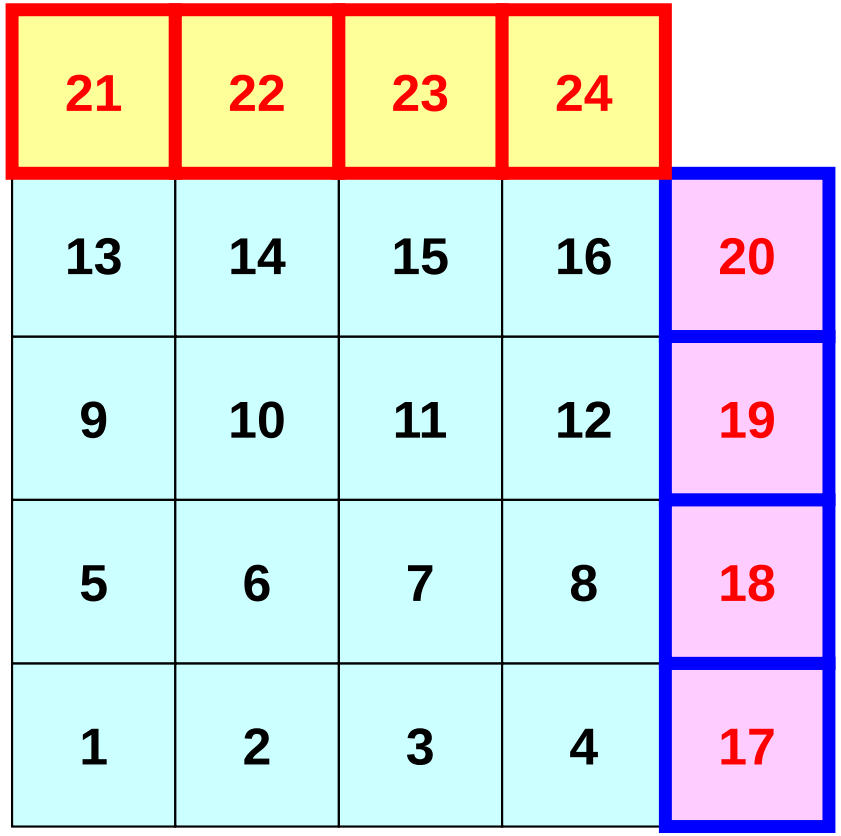


PE#1

```
#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)

PE#2



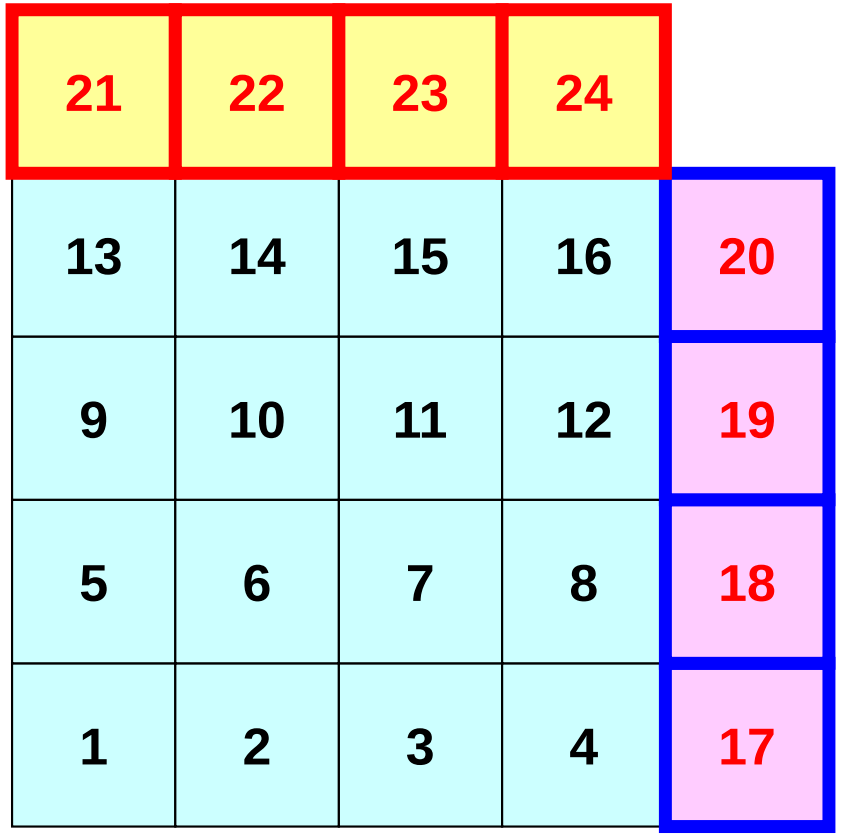
PE#1

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

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)

PE#2



PE#1

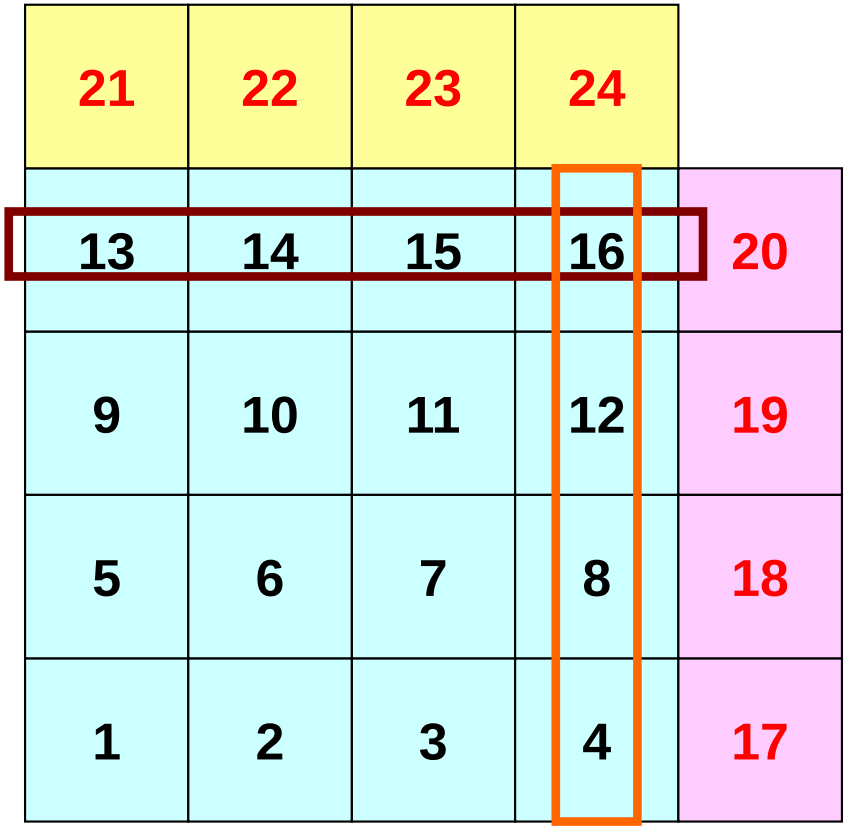
```
#NEIBPEtot
2
#NEIBPE
1  2
#NODE
24 16
#IMPORT_index
4  8
#IMPORT_items
17
18
19
20
21
22
23
24
#EXPORT_index
4  8
#EXPORT_items
4
8
12
16
13
14
15
16
```

imported from 1st Neighbor (PE#1) (1st-4th items)

imported from 2nd Neighbor (PE#2) (5th-8th items)

Generalized Comm. Table (5/6)

PE#2



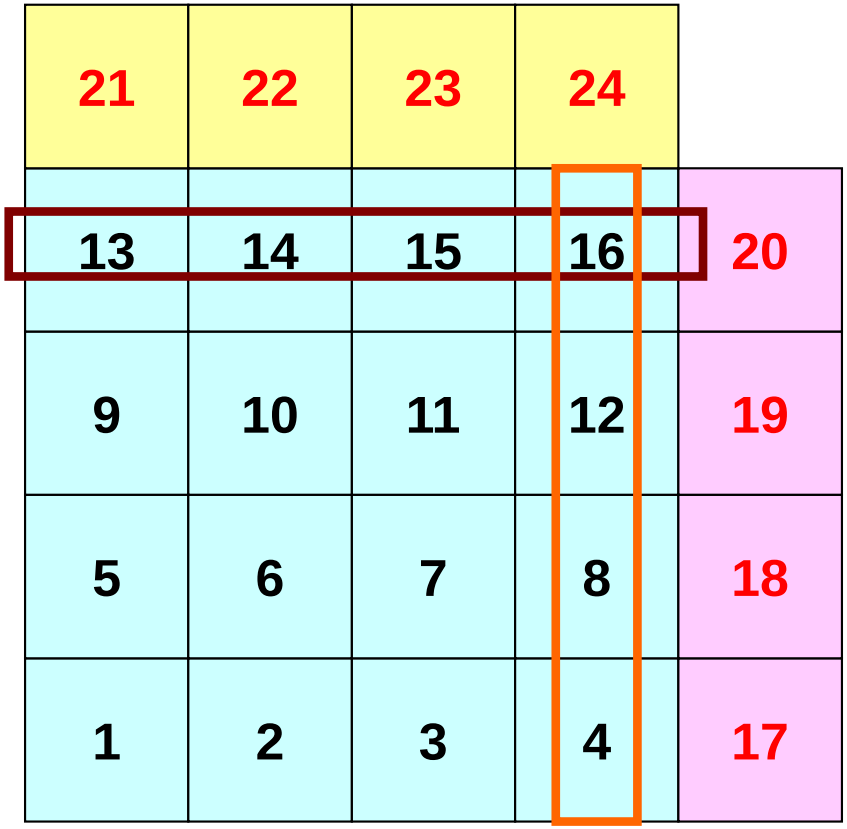
PE#1

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

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)

PE#2



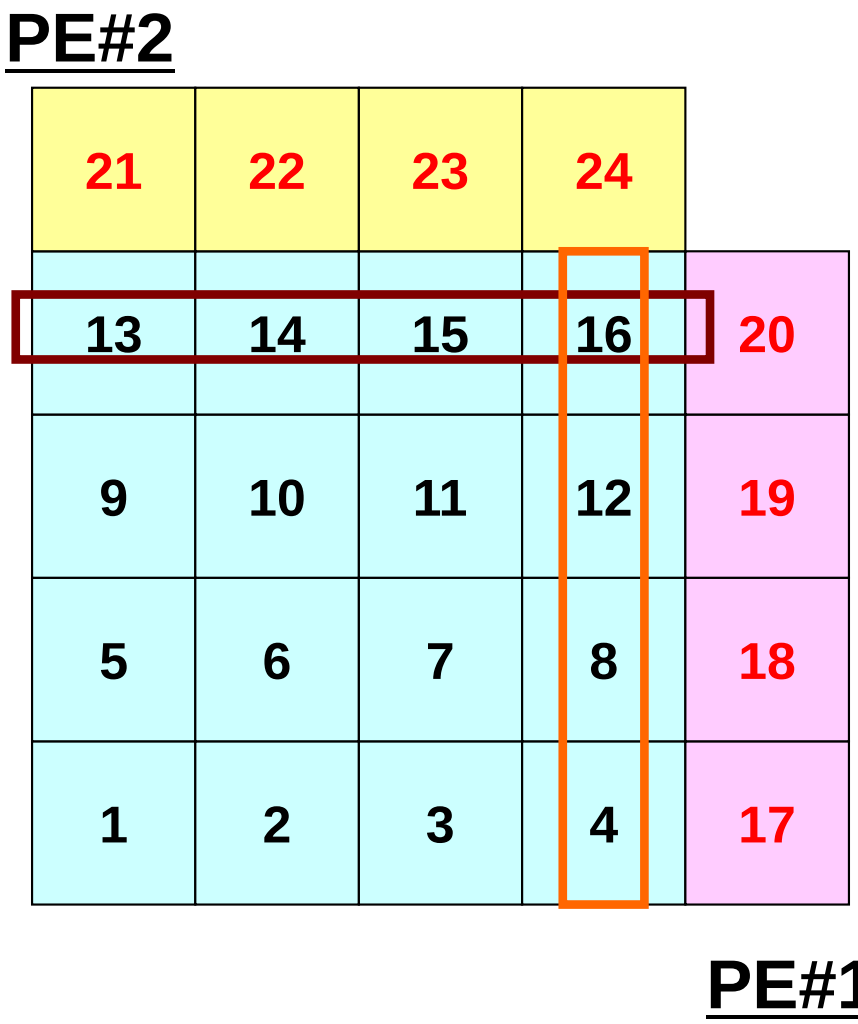
PE#1

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

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

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

Generalized Comm. Table (6/6)



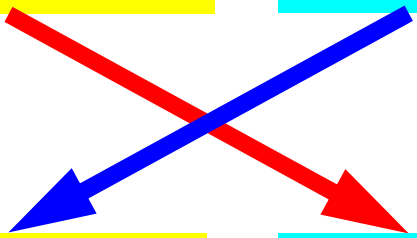
An external point is only sent from its original domain.

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 /work/gt36/t36XXX/pFEM/mpi/S2
$ module load fj

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

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

go4.sh

```
#!/bin/sh
#PJM -N "test"
#PJM -L rscgrp=lecture6-o
#PJM -L node=1
#PJM --mpi proc=4
#PJM -L elapse=00:15:00
#PJM -g gt36
#PJM -j
#PJM -e err
#PJM -o test.lst

module load fj
module load fjmpi

mpiexec ./a.out
```

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

C

Initialization

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <assert.h>
#include "mpi.h"
int main(int argc, char **argv){

        int n, np, NeibPeTot, BufLength;
        MPI_Status *StatSend, *StatRecv;
        MPI_Request *RequestSend, *RequestRecv;

        int MyRank, PeTot;
        int *val, *SendBuf, *RecvBuf, *NeibPe;
        int *ImportIndex, *ExportIndex, *ImportItem, *ExportItem;

        char FileName[80], line[80];
        int i, nn, neib;
        int iStart, iEnd;
        FILE *fp;

/*
!C +-----------+
!C | INIT. MPI |
!C +-----------+
!C===*/
        MPI_Init(&argc, &argv);
        MPI_Comm_size(MPI_COMM_WORLD, &PeTot);
        MPI_Comm_rank(MPI_COMM_WORLD, &MyRank);
```

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

C

Reading distributed local data files (sqm.*)

```
/*
!C +-----------+
!C | DATA file |
!C +-----------+
!C===*/
        sprintf(FileName, "sqm.%d", MyRank);
        fp = fopen(FileName, "r");

        fscanf(fp, "%d", &NeibPeTot);
        NeibPe = calloc(NeibPeTot, sizeof(int));
        ImportIndex = calloc(1+NeibPeTot, sizeof(int));
        ExportIndex = calloc(1+NeibPeTot, sizeof(int));

        for(neib=0;neib<NeibPeTot;neib++){
                fscanf(fp, "%d", &NeibPe[neib]);
        }
        fscanf(fp, "%d %d", &np, &n);

        for(neib=1;neib<NeibPeTot+1;neib++){
                fscanf(fp, "%d", &ImportIndex[neib]);}
        nn = ImportIndex[NeibPeTot];
        ImportItem = malloc(nn * sizeof(int));
        for(i=0;i<nn;i++){
                fscanf(fp, "%d", &ImportItem[i]); ImportItem[i]--;}

        for(neib=1;neib<NeibPeTot+1;neib++){
                fscanf(fp, "%d", &ExportIndex[neib]);}
        nn = ExportIndex[NeibPeTot];
        ExportItem = malloc(nn * sizeof(int));

        for(i=0;i<nn;i++){
                fscanf(fp, "%d", &ExportItem[i]);ExportItem[i]--;}
```

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

C

Reading distributed local data files (sqm.*)

```
/*
!C +-----------+
!C | DATA file |
!C +-----------+
!C===*/
        sprintf(FileName, "sqm.%d", MyRank);
        fp = fopen(FileName, "r");

        fscanf(fp, "%d", &NeibPeTot);
        NeibPe = calloc(NeibPeTot, sizeof(int));
        ImportIndex = calloc(1+NeibPeTot, sizeof(int));
        ExportIndex = calloc(1+NeibPeTot, sizeof(int));

        for(neib=0;neib<NeibPeTot;neib++){
                fscanf(fp, "%d", &NeibPe[neib]);
        }
        fscanf(fp, "%d %d", &np, &n);

        for(neib=1;neib<NeibPeTot+1;neib++){
                fscanf(fp, "%d", &ImportIndex[neib]);}
        nn = ImportIndex[NeibPeTot];
        ImportItem = malloc(nn * sizeof(int));
        for(i=0;i<nn;i++){
                fscanf(fp, "%d", &ImportItem[i]); ImportItem[i]--;}

        for(neib=1;neib<NeibPeTot+1;neib++){
                fscanf(fp, "%d", &ExportIndex[neib]);}
        nn = ExportIndex[NeibPeTot];
        ExportItem = malloc(nn * sizeof(int));

        for(i=0;i<nn;i++){
                fscanf(fp, "%d", &ExportItem[i]);ExportItem[i]--;}
```

```
#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.c (2/6)

C

Reading distributed local data files (sqm.*)

```
/*
!C +-----------+
!C | DATA file |
!C +-----------+
!C===*/
        sprintf(FileName, "sqm.%d", MyRank);
        fp = fopen(FileName, "r");

        fscanf(fp, "%d", &NeibPeTot);
        NeibPe = calloc(NeibPeTot, sizeof(int));
        for(neib=0;neib<NeibPeTot;neib++){
                fscanf(fp, "%d", &NeibPe[neib]);
        }
        fscanf(fp, "%d %d", &np, &n);

        for(neib=1;neib<NeibPeTot+1;neib++){
                fscanf(fp, "%d", &ImportIndex[neib]);}
        nn = ImportIndex[NeibPeTot];
        ImportItem = malloc(nn * sizeof(int));
        for(i=0;i<nn;i++){
                fscanf(fp, "%d", &ImportItem[i]); ImportItem[i]--;}

        for(neib=1;neib<NeibPeTot+1;neib++){
                fscanf(fp, "%d", &ExportIndex[neib]);}
        nn = ExportIndex[NeibPeTot];
        ExportItem = malloc(nn * sizeof(int));

        for(i=0;i<nn;i++){
                fscanf(fp, "%d", &ExportItem[i]);ExportItem[i]--;}
```

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.c (2/6)

C

Reading distributed local data files (sqm.*)

```
/*
!C +-----------+
!C | DATA file |
!C +-----------+
!C===*/
        sprintf(FileName, "sqm.%d", MyRank);
        fp = fopen(FileName, "r");

        fscanf(fp, "%d", &NeibPeTot);
        NeibPe = calloc(NeibPeTot, sizeof(int));
        ImportIndex = calloc(1+NeibPeTot, sizeof(int));
        ExportIndex = calloc(1+NeibPeTot, sizeof(int));

        for(neib=0;neib<NeibPeTot;neib++){
                fscanf(fp, "%d", &NeibPe[neib]);
        }
        fscanf(fp, "%d %d", &np, &n);

        for(neib=1;neib<NeibPeTot+1;neib++){
                fscanf(fp, "%d", &ImportIndex[neib]);}
        nn = ImportIndex[NeibPeTot];
        ImportItem = malloc(nn * sizeof(int));
        for(i=0;i<nn;i++){
                fscanf(fp, "%d", &ImportItem[i]); ImportItem[i]

        for(neib=1;neib<NeibPeTot+1;neib++){
                fscanf(fp, "%d", &ExportIndex[neib]);}
        nn = ExportIndex[NeibPeTot];
        ExportItem = malloc(nn * sizeof(int));

        for(i=0;i<nn;i++){
                fscanf(fp, "%d", &ExportItem[i]);ExportItem[i]--;}
```

```
#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.c (2/6)

C

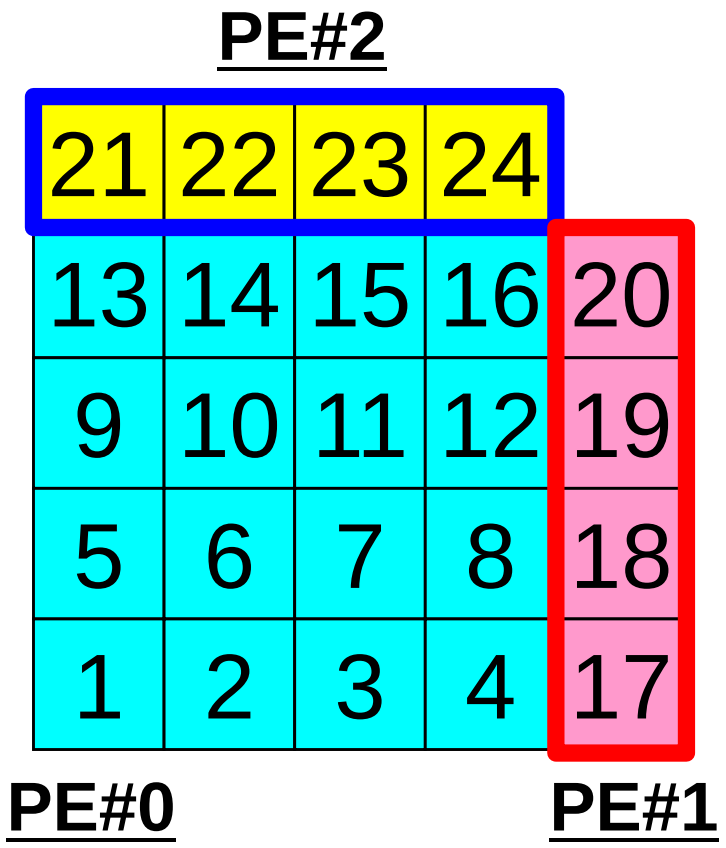
Reading distributed local data files (sqm.*)

```
/*
!C +-----------+
!C | DATA file |
!C +-----------+
!C===*/
        sprintf(FileName, "sqm.%d", MyRank);
        fp = fopen(FileName, "r");

        fscanf(fp, "%d", &NeibPeTot);
        NeibPe = calloc(NeibPeTot, sizeof(int));
        ImportIndex = calloc(1+NeibPeTot, sizeof(int));
        ExportIndex = calloc(1+NeibPeTot, sizeof(int));

        for(neib=0;neib<NeibPeTot;neib++){
                fscanf(fp, "%d", &NeibPe[neib]);
        }
        fscanf(fp, "%d %d", &np, &n);

        for(neib=1;neib<NeibPeTot+1;neib++){
                fscanf(fp, "%d", &ImportIndex[neib]);}
        nn = ImportIndex[NeibPeTot];
        ImportItem = malloc(nn * sizeof(int));
        for(i=0;i<nn;i++){
                fscanf(fp, "%d", &ImportItem[i]); ImportItem[i]-

        for(neib=1;neib<NeibPeTot+1;neib++){
                fscanf(fp, "%d", &ExportIndex[neib]);}
        nn = ExportIndex[NeibPeTot];
        ExportItem = malloc(nn * sizeof(int));

        for(i=0;i<nn;i++){
                fscanf(fp, "%d", &ExportItem[i]);ExportItem[i]--;}
```

```
#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.c (2/6)

C

Reading distributed local data files (sqm.*)

```
/*
!C +-----------+
!C | DATA file |
!C +-----------+
!C===*/
        sprintf(FileName, "sqm.%d", MyRank);
        fp = fopen(FileName, "r");

        fscanf(fp, "%d", &NeibPeTot);
        NeibPe = calloc(NeibPeTot, sizeof(int));
        ImportIndex = calloc(1+NeibPeTot, sizeof(int));
        ExportIndex = calloc(1+NeibPeTot, sizeof(int));

        for(neib=0;neib<NeibPeTot;neib++){
                fscanf(fp, "%d", &NeibPe[neib]);
        }
        fscanf(fp, "%d %d", &np, &n);

        for(neib=1;neib<NeibPeTot+1;neib++){
                fscanf(fp, "%d", &ImportIndex[neib]);}
        nn = ImportIndex[NeibPeTot];
        ImportItem = malloc(nn * sizeof(int));
        for(i=0;i<nn;i++){
                fscanf(fp, "%d", &ImportItem[i]); ImportItem[i]--;}

        for(neib=1;neib<NeibPeTot+1;neib++){
                fscanf(fp, "%d", &ExportIndex[neib]);}
        nn = ExportIndex[NeibPeTot];
        ExportItem = malloc(nn * sizeof(int));

        for(i=0;i<nn;i++){
                fscanf(fp, "%d", &ExportItem[i]);ExportItem[i]--;}
```

```
#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.c (2/6)

C

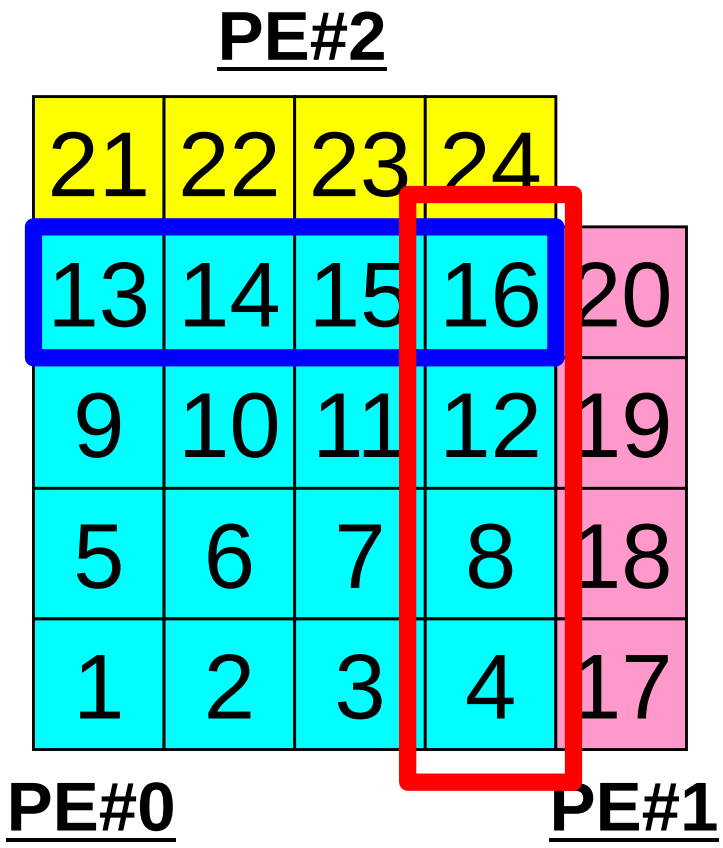
Reading distributed local data files (sqm.*)

```
/*
!C +-----------+
!C | DATA file |
!C +-----------+
!C===*/
        sprintf(FileName, "sqm.%d", MyRank);
        fp = fopen(FileName, "r");

        fscanf(fp, "%d", &NeibPeTot);
        NeibPe = calloc(NeibPeTot, sizeof(int));
        ImportIndex = calloc(1+NeibPeTot, sizeof(int));
        ExportIndex = calloc(1+NeibPeTot, sizeof(int));

        for(neib=0;neib<NeibPeTot;neib++){
                fscanf(fp, "%d", &NeibPe[neib]);
        }
        fscanf(fp, "%d %d", &np, &n);

        for(neib=1;neib<NeibPeTot+1;neib++){
                fscanf(fp, "%d", &ImportIndex[neib]);}
        nn = ImportIndex[NeibPeTot];
        ImportItem = malloc(nn * sizeof(int));
        for(i=0;i<nn;i++){
                fscanf(fp, "%d", &ImportItem[i]); ImportItem[i]--;}

        for(neib=1;neib<NeibPeTot+1;neib++){
                fscanf(fp, "%d", &ExportIndex[neib]);}
        nn = ExportIndex[NeibPeTot];
        ExportItem = malloc(nn * sizeof(int));

        for(i=0;i<nn;i++){
                fscanf(fp, "%d", &ExportItem[i]);ExportItem[i]--;}
```

```
#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.c (3/6)

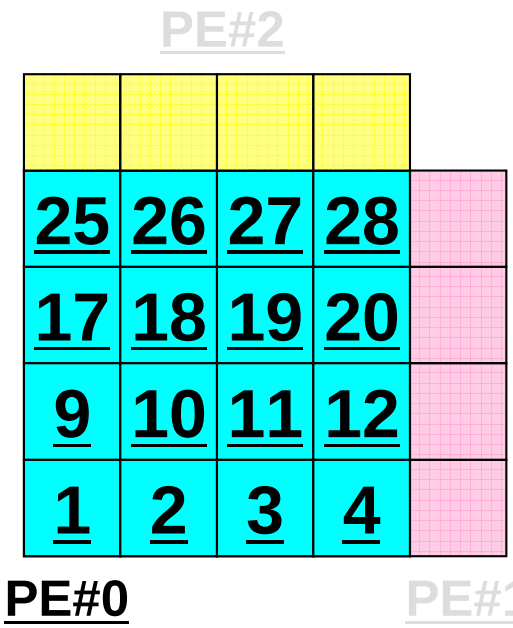
C

Reading distributed local data files (sq.*)

```
sprintf(FileName, "sq.%d", MyRank);

fp = fopen(FileName, "r");
assert(fp != NULL);

val = calloc(np, sizeof(*val));
for(i=0;i<n;i++){
        fscanf(fp, "%d", &val[i]);
}
```

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.c (4/6)

C

Preparation of sending/receiving buffers

```
/*
!C
!C +--------+
!C | BUFFER |
!C +--------+
!C===*/
        SendBuf = calloc(ExportIndex[NeibPeTot], sizeof(*SendBuf));
        RecvBuf = calloc(ImportIndex[NeibPeTot], sizeof(*RecvBuf));

        for(neib=0;neib<NeibPeTot;neib++){
                iStart = ExportIndex[neib];
                iEnd   = ExportIndex[neib+1];
                for(i=iStart;i<iEnd;i++){
                        SendBuf[i] = val[ExportItem[i]];
                }
        }
```

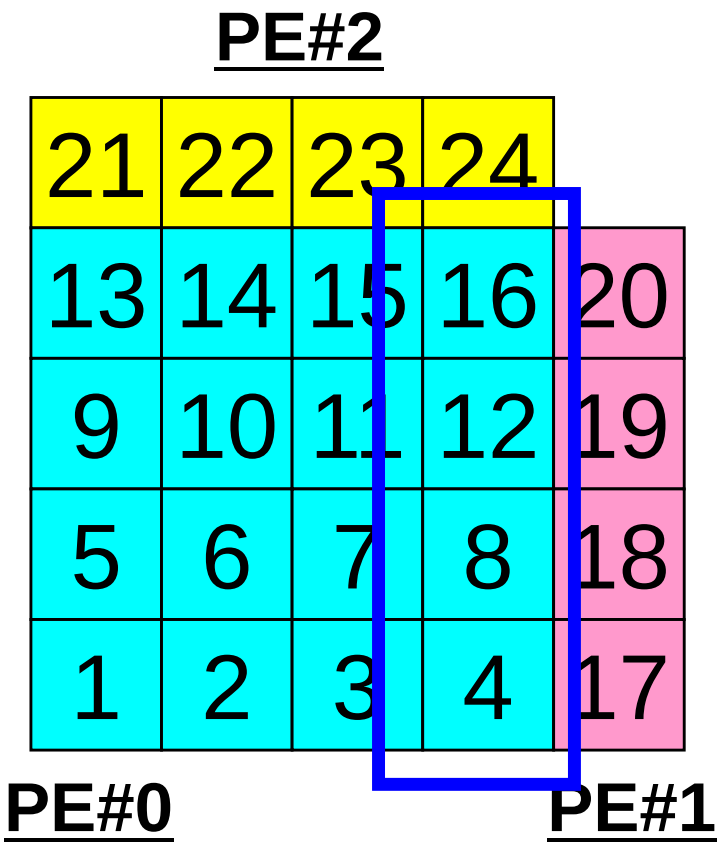
Info. of boundary points is written into sending buffer (**SendBuf**).

Info. sent to **NeibPe[neib]** is stored in **SendBuf[ExportIndex[neib]: ExportInedx[neib+1]-1]**

Sending Buffer is nice ...

C

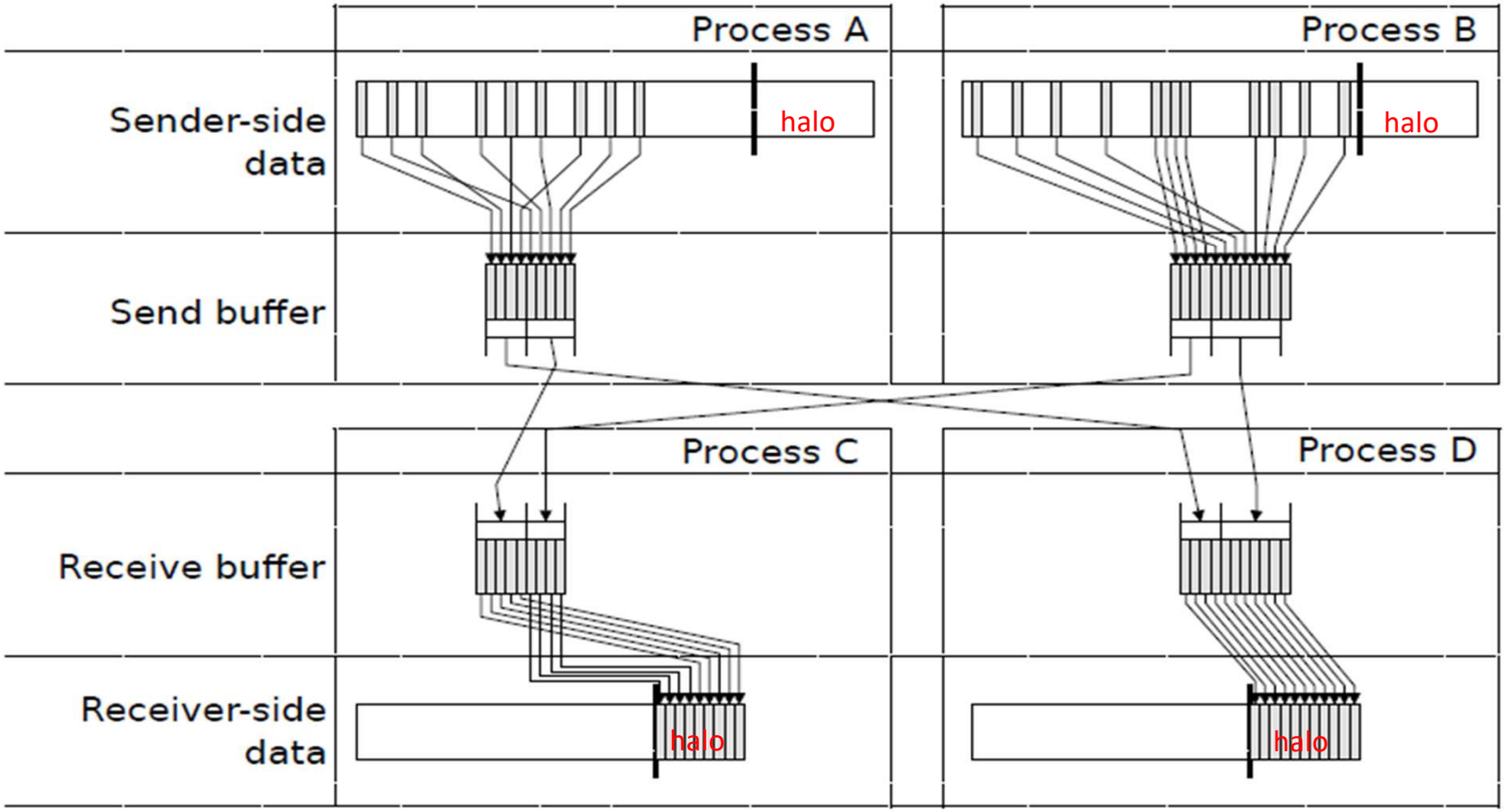
```
for (neib=0; neib<NeibPETot; neib++){
   tag= 0;
   iS_e= export_index[neib];
   iE_e= export_index[neib+1];
   BUFlength_e= iE_e - iS_e

   ierr= MPI_Isend
         (&SendBuf[iS_e], BUFlength_e, MPI_DOUBLE, NeibPE[neib], 0,
          MPI_COMM_WORLD, &ReqSend[neib])
 }
```



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



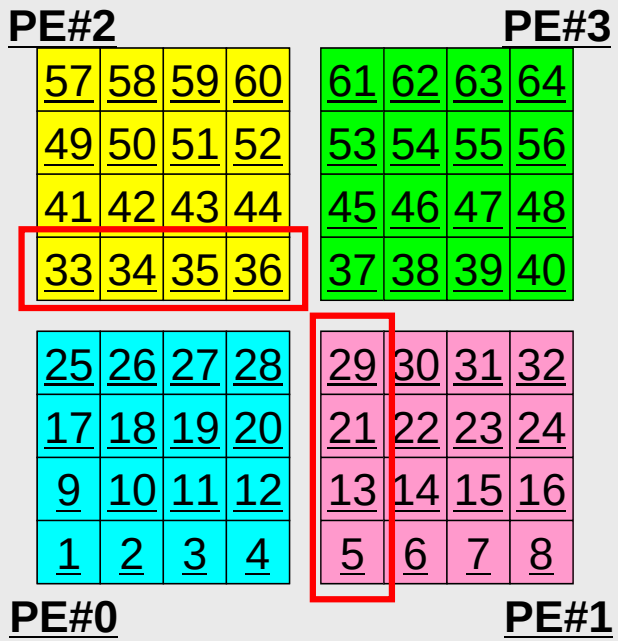
Dr. Osni Marques
（Lawrence Berkeley National Laboratory）

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

C

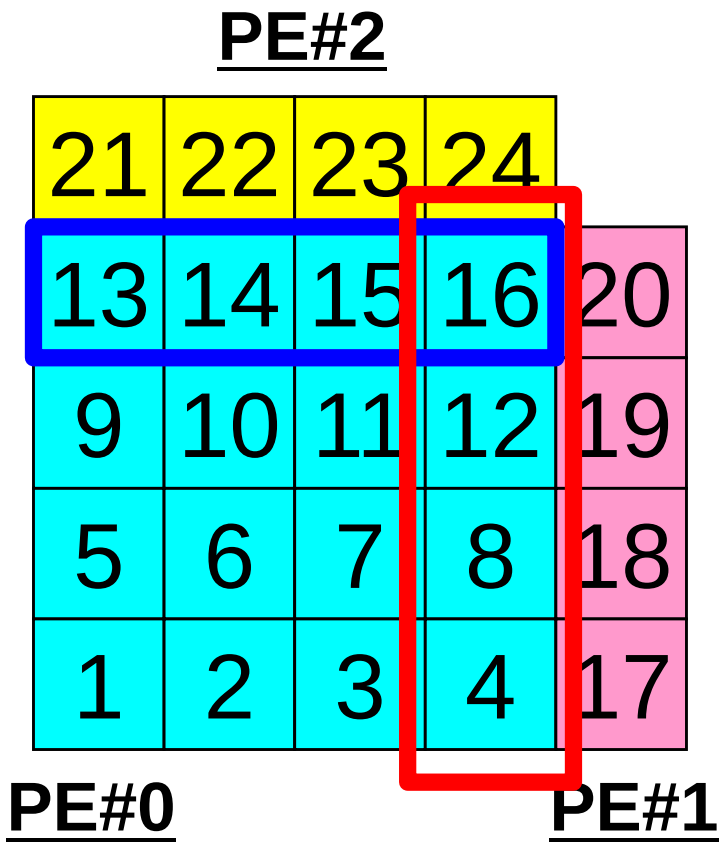
SEND/Export: MPI_Isend

```
/*
!C
!C +-----------+
!C | SEND-RECV |
!C +-----------+
!C===*/
        StatSend = malloc(sizeof(MPI_Status) * NeibPeTot);
        StatRecv = malloc(sizeof(MPI_Status) * NeibPeTot);
        RequestSend = malloc(sizeof(MPI_Request) * NeibPeTot);
        RequestRecv = malloc(sizeof(MPI_Request) * NeibPeTot);

        for(neib=0;neib<NeibPeTot;neib++){
                iStart = ExportIndex[neib];
                iEnd   = ExportIndex[neib+1];
                BufLength = iEnd - iStart;
                MPI_Isend(&SendBuf[iStart], BufLength, MPI_INT,
                                NeibPe[neib], 0, MPI_COMM_WORLD, &RequestSend[neib]);
        }

        for(neib=0;neib<NeibPeTot;neib++){
                iStart = ImportIndex[neib];
                iEnd   = ImportIndex[neib+1];
                BufLength = iEnd - iStart;

                MPI_Irecv(&RecvBuf[iStart], BufLength, MPI_INT,
                                NeibPe[neib], 0, MPI_COMM_WORLD, &RequestRecv[neib]);
        }
```



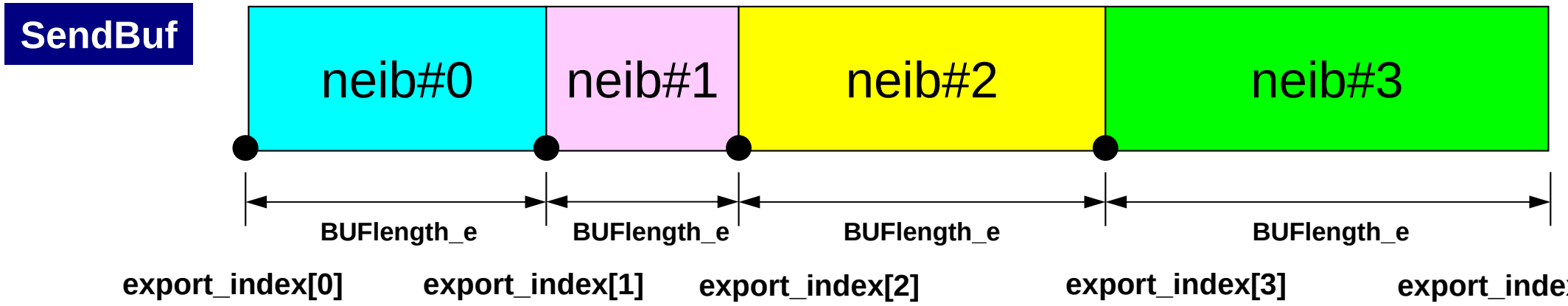
SEND/Export: PE#0

```
#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

C



export_item (export_index[neib]:export_index[neib+1]-1) are sent to neib-th neighbor

```
for (neib=0; neib<NeibPETot;neib++){
   for (k=export_index[neib];k<export_index[neib+1];k++){
      kk= export_item[k];
      SendBuf[k]= VAL[kk];
   }
}

for (neib=0; neib<NeibPETot; neib++){
  tag= 0;
  iS_e= export_index[neib];
  iE_e= export_index[neib+1];
  BUFlength_e= iE_e - iS_e

  ierr= MPI_Isend
        (&SendBuf[iS_e], BUFlength_e, MPI_DOUBLE, NeibPE[neib], 0,
         MPI_COMM_WORLD, &ReqSend[neib])
}

MPI_Waitall(NeibPETot, ReqSend, StatSend);
```

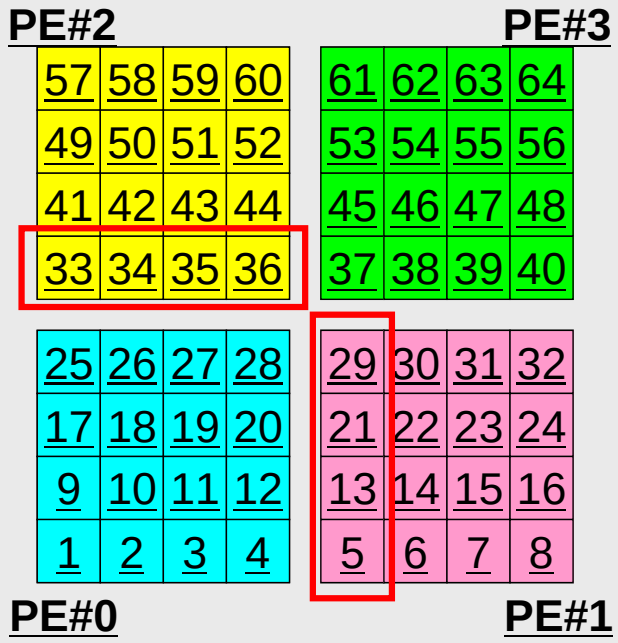
Copied to sending buffers

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

C

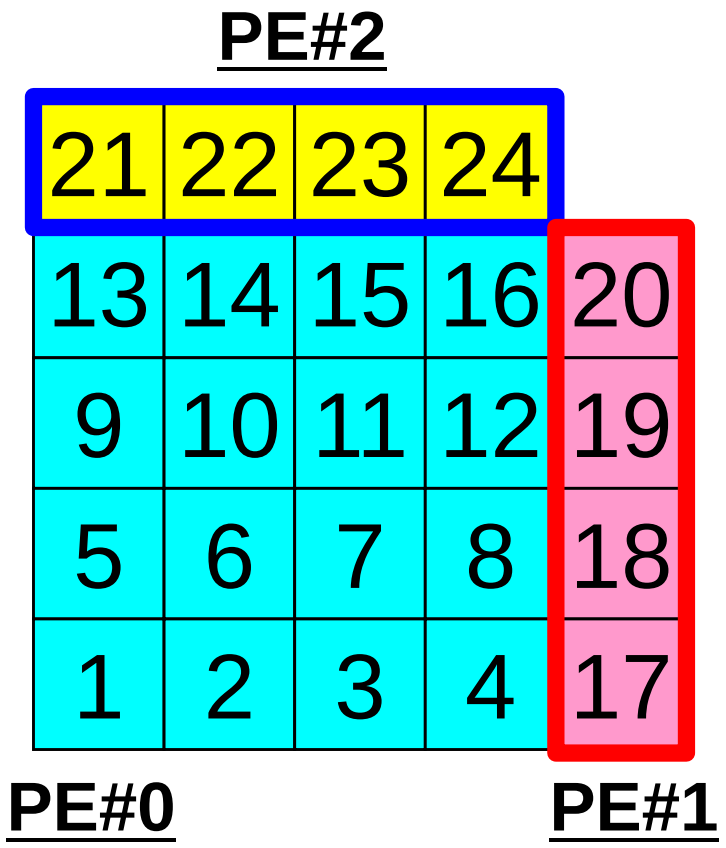
RECV/Import: MPI_Irecv

```
/*
!C
!C +-----------+
!C | SEND-RECV |
!C +-----------+
!C===*/
        StatSend = malloc(sizeof(MPI_Status) * NeibPeTot);
        StatRecv = malloc(sizeof(MPI_Status) * NeibPeTot);
        RequestSend = malloc(sizeof(MPI_Request) * NeibPeTot);
        RequestRecv = malloc(sizeof(MPI_Request) * NeibPeTot);

        for(neib=0;neib<NeibPeTot;neib++){
                iStart = ExportIndex[neib];
                iEnd   = ExportIndex[neib+1];
                BufLength = iEnd - iStart;
                MPI_Isend(&SendBuf[iStart], BufLength, MPI_INT,
                                NeibPe[neib], 0, MPI_COMM_WORLD, &RequestSend[neib]);
        }

        for(neib=0;neib<NeibPeTot;neib++){
                iStart = ImportIndex[neib];
                iEnd   = ImportIndex[neib+1];
                BufLength = iEnd - iStart;

                MPI_Irecv(&RecvBuf[iStart], BufLength, MPI_INT,
                                NeibPe[neib], 0, MPI_COMM_WORLD, &RequestRecv[neib]);
        }
```



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

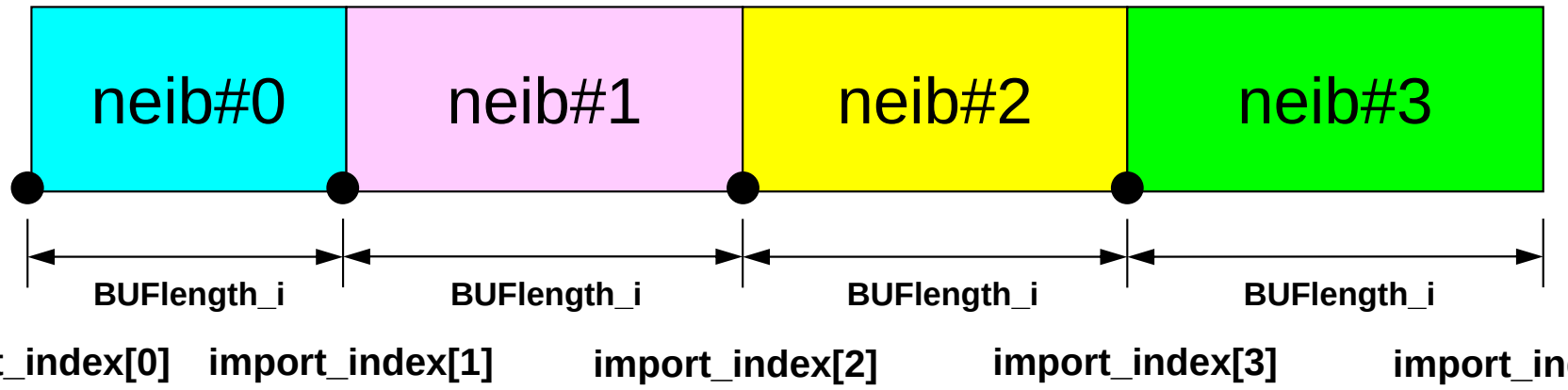
C

```
for (neib=0; neib<NeibPETot; neib++){
   tag= 0;
   iS_i= import_index[neib];
   iE_i= import_index[neib+1];
   BUFlength_i= iE_i - iS_i

   ierr= MPI_Irecv
         (&RecvBuf[iS_i], BUFlength_i, MPI_DOUBLE, NeibPE[neib], 0,
          MPI_COMM_WORLD, &ReqRecv[neib])
}

MPI_Waitall(NeibPETot, ReqRecv, StatRecv);

for (neib=0; neib<NeibPETot;neib++){
    for (k=import_index[neib];k<import_index[neib+1];k++){
       kk= import_item[k];
       VAL[kk]= RecvBuf[k];
    }
}
```

Copied from receiving buffer

import_item (import_index[neib]:import_index[neib+1]-1) are received from neib-th neighbor

RecvBuf



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

C

Reading info of ext pts from receiving buffers

```
        MPI_Waitall(NeibPeTot, RequestRecv, StatRecv);

        for(neib=0;neib<NeibPeTot;neib++){
                iStart = ImportIndex[neib];
                iEnd   = ImportIndex[neib+1];
                for(i=iStart;i<iEnd;i++){
                        val[ImportItem[i]] = RecvBuf[i];
                }
        }
        MPI_Waitall(NeibPeTot, RequestSend, StatSend); /*

!C +--------+
!C | OUTPUT |
!C +--------+
!C===*/
        for(neib=0;neib<NeibPeTot;neib++){
                iStart = ImportIndex[neib];
                iEnd   = ImportIndex[neib+1];
                for(i=iStart;i<iEnd;i++){
                        int in = ImportItem[i];
                        printf("RECVbuf%8d%8d%8d¥n", MyRank, NeibPe[neib], val[in]);
                }
        }
        MPI_Finalize();

        return 0;
}
```

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

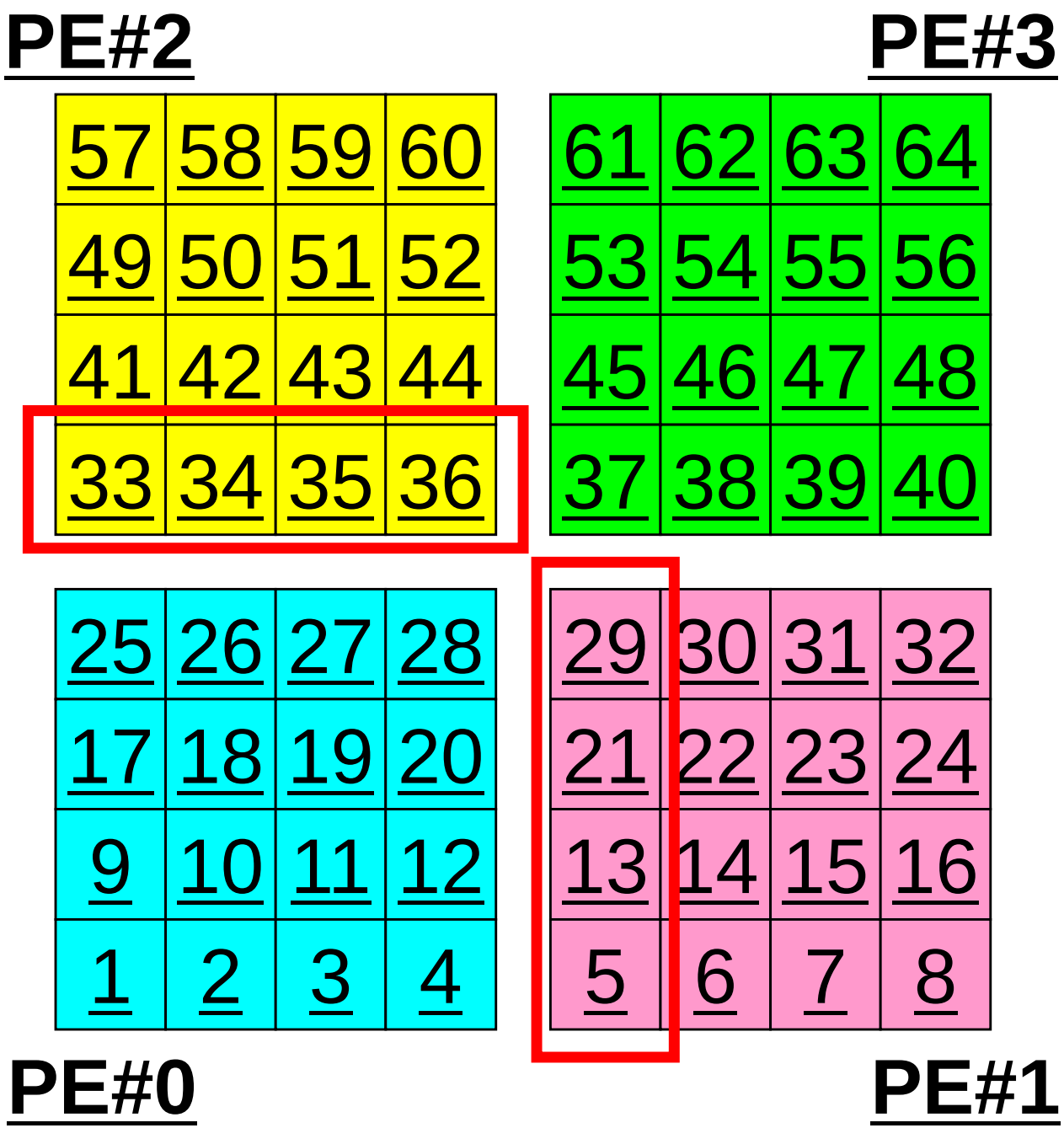
Example: sq-sr1.c (6/6)

C

Writing values at external points

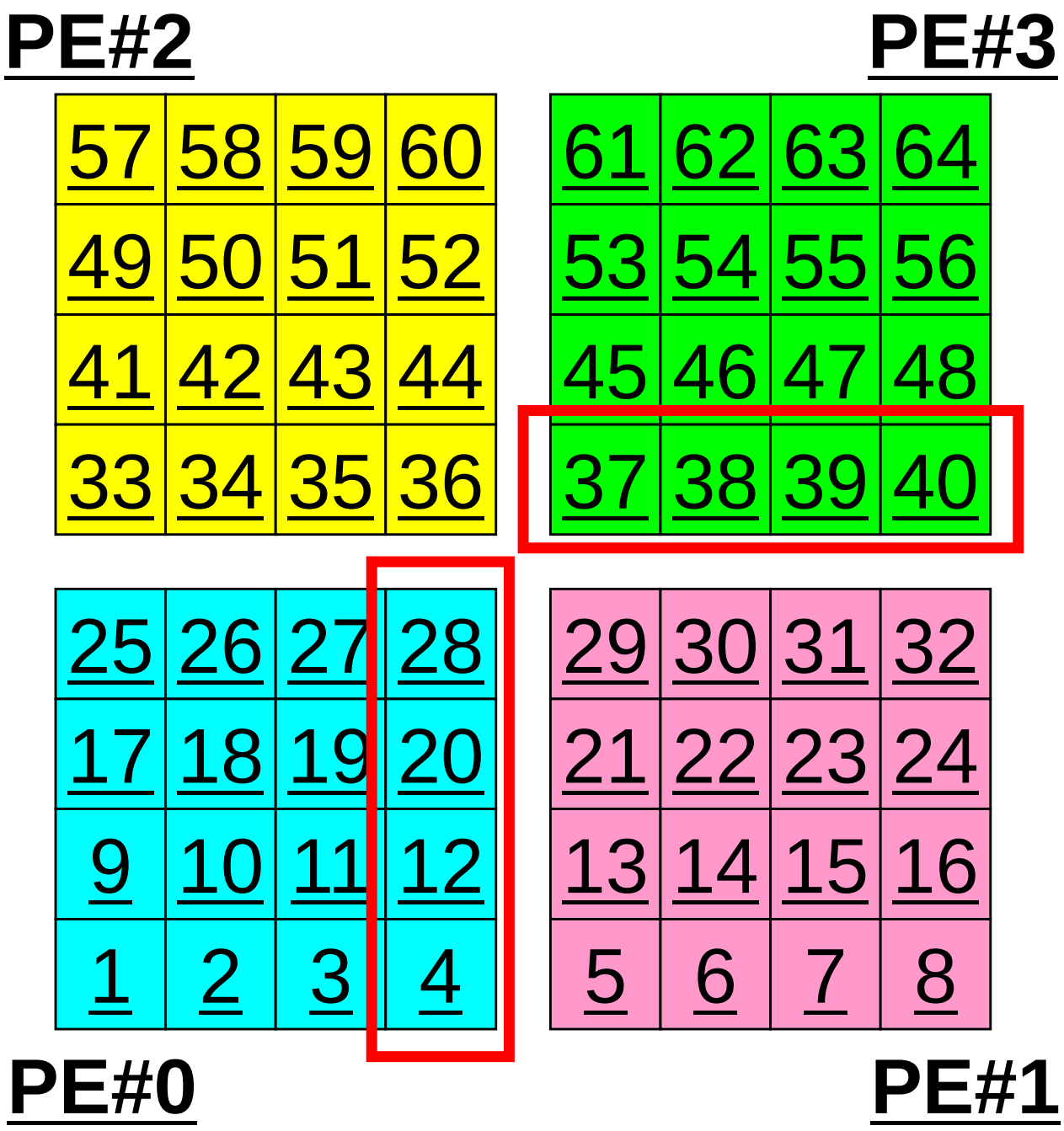
```
        MPI_Waitall(NeibPeTot, RequestRecv, StatRecv);

        for(neib=0;neib<NeibPeTot;neib++){
                iStart = ImportIndex[neib];
                iEnd   = ImportIndex[neib+1];
                for(i=iStart;i<iEnd;i++){
                        val[ImportItem[i]] = RecvBuf[i];
                }
        }
        MPI_Waitall(NeibPeTot, RequestSend, StatSend); /*

!C +--------+
!C | OUTPUT |
!C +--------+
!C===*/
        for(neib=0;neib<NeibPeTot;neib++){
                iStart = ImportIndex[neib];
                iEnd   = ImportIndex[neib+1];
                for(i=iStart;i<iEnd;i++){
                        int in = ImportItem[i];
                        printf("RECVbuf%8d%8d%8d¥n", MyRank, NeibPe[neib], val[in]);
                }
        }
        MPI_Finalize();

        return 0;
}
```

Results (PE#0)



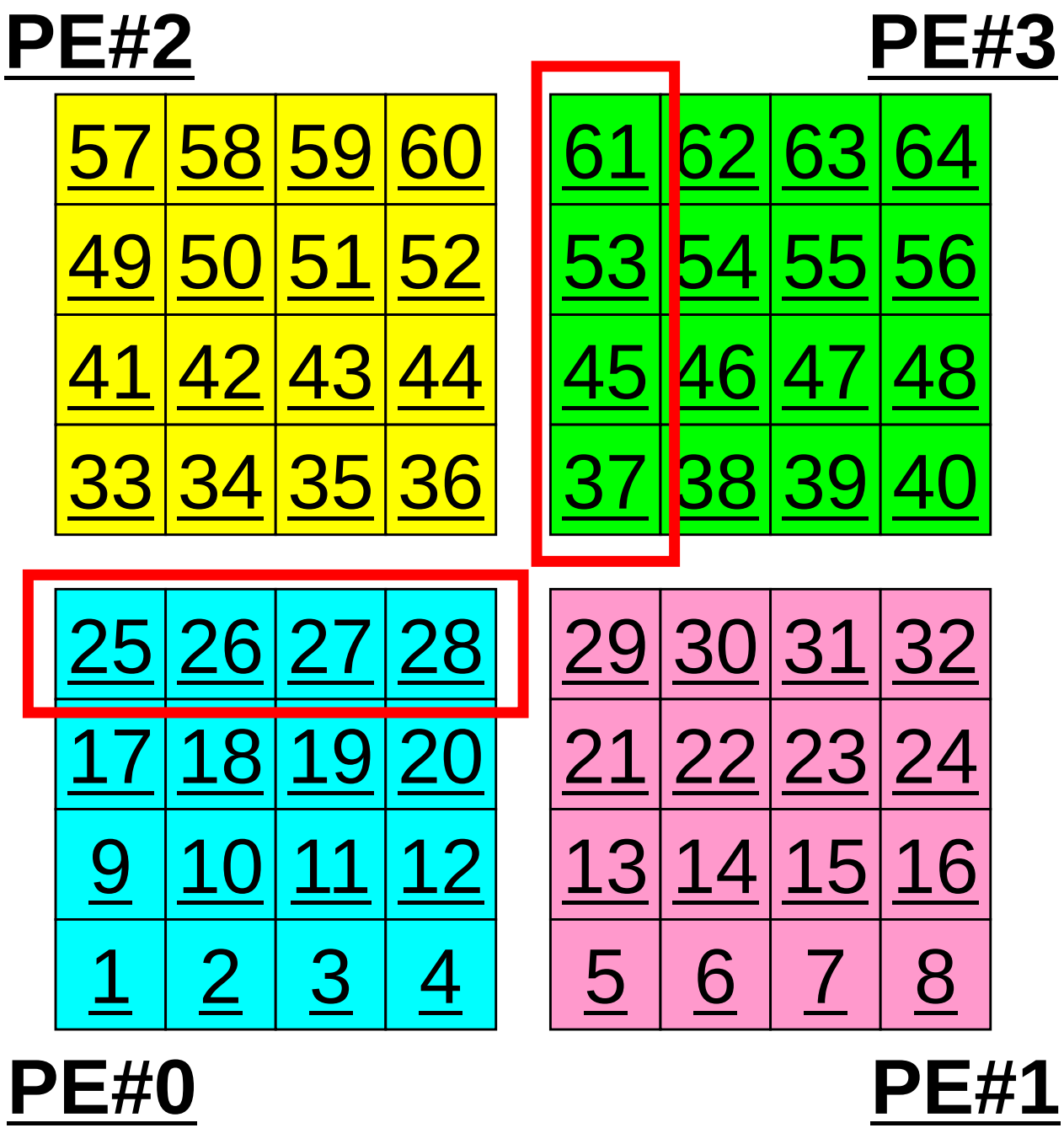

```
RECVbuf      0      1      5
RECVbuf      0      1     13
RECVbuf      0      1     21
RECVbuf      0      1     29
RECVbuf      0      2     33
RECVbuf      0      2     34
RECVbuf      0      2     35
RECVbuf      0      2     36

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)



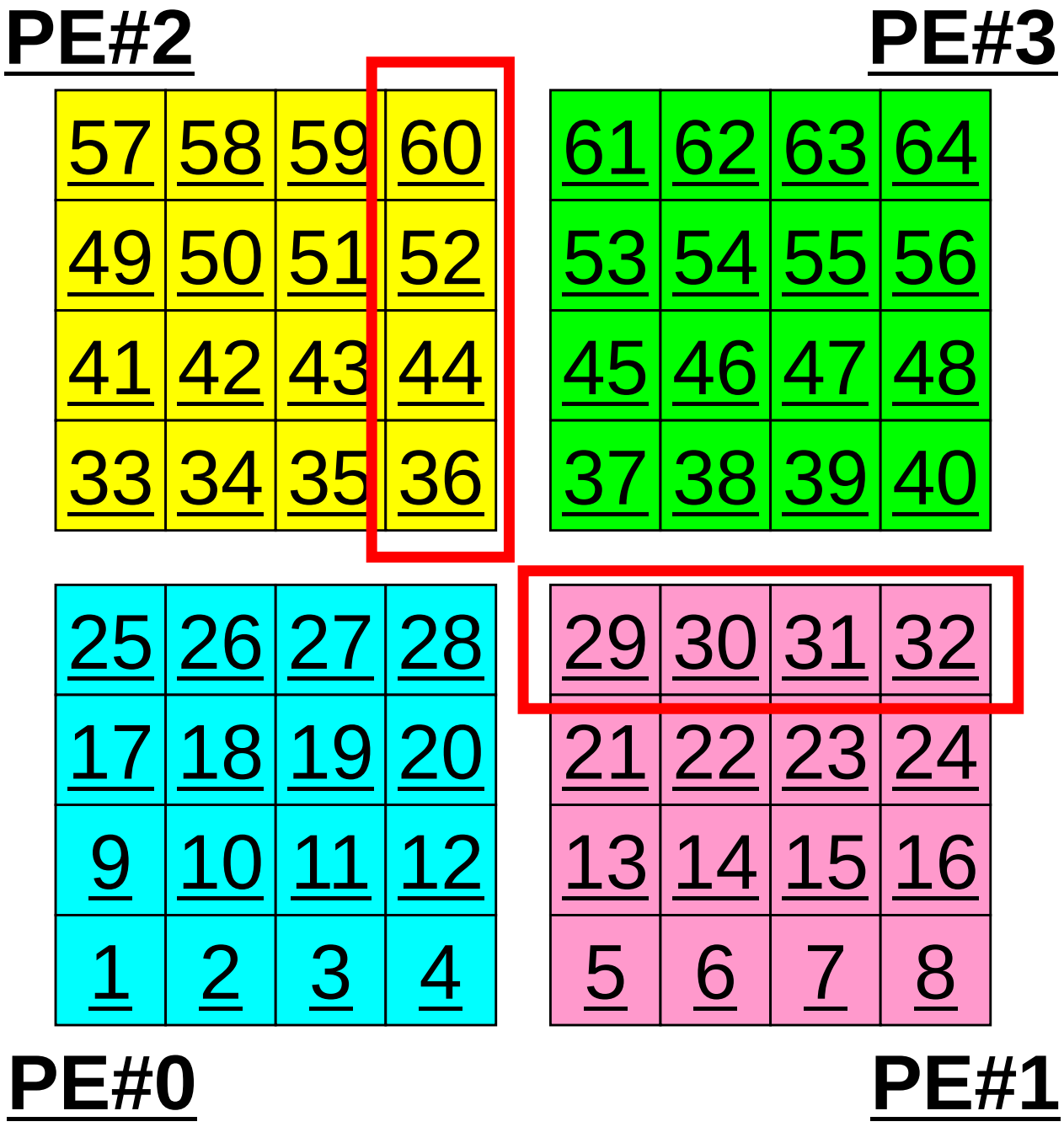

```
RECVbuf      0      1       5
RECVbuf      0      1      13
RECVbuf      0      1      21
RECVbuf      0      1      29
RECVbuf      0      2      33
RECVbuf      0      2      34
RECVbuf      0      2      35
RECVbuf      0      2      36

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

#PE2

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

#PE1

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

#PE0

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

Three Domains

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

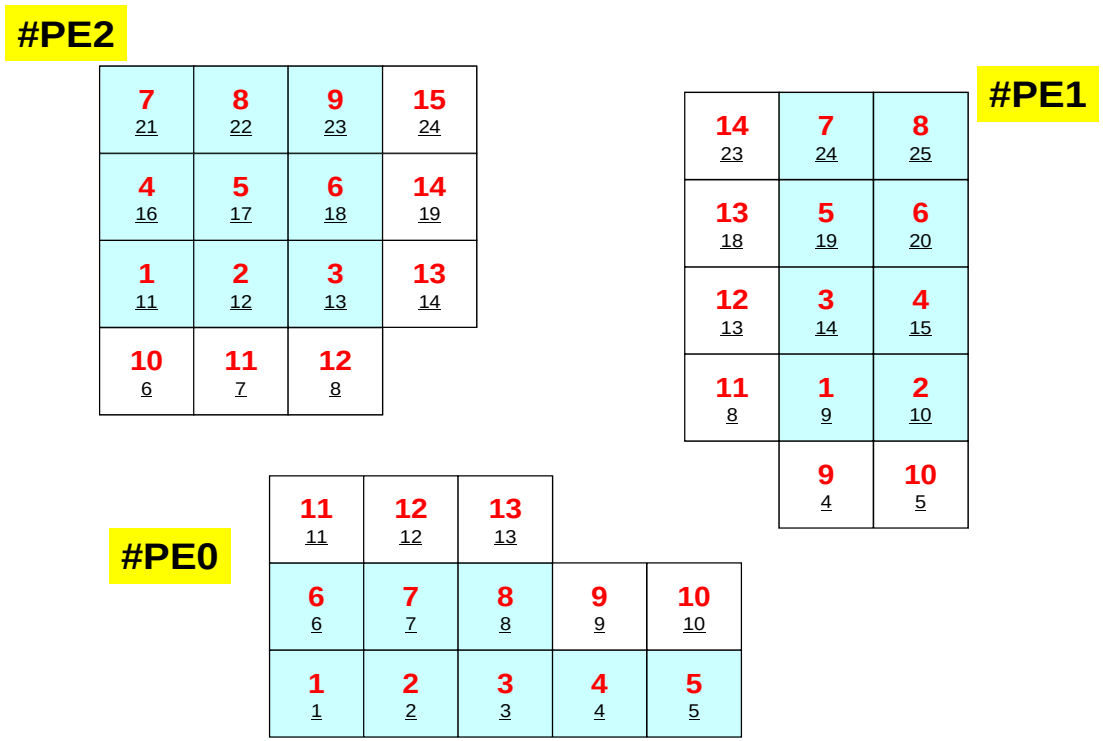
14 23	7 24	8 25
13 18	5 19	6 20
12 13	3 14	4 15
11 8	1 9	2 10
	9 4	10 5

#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

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

#PE2

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

#PE1

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

#PE0

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

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

t2

PE#2: sqm.2: fill ○'s

#PE2

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

#PE1

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

#PE0

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

```
#NEIBPEtot
    2
#NEIBPE
    1    0
#NODE
   15    9 (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

t2

Procedures

- 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

Report 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

Report S2 (2/2)

- ~~Deadline: January 24th (Wed), 2024, 17:00@ITC-LMS~~
- 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.
- ~~Report~~
  - ~~Cover Page: Name, ID, and Problem ID (S2) must be written.~~
  - ~~Less than eight pages including figures and tables (A4).~~
    - ~~Strategy, Structure of the Program, Remarks~~
  - ~~Source list of the program (if you have bugs)~~
  - ~~Output list (as small as possible)~~

a012.sh

```
#!/bin/sh
#PJM -N "test"
#PJM -L rscgrp=lecture6-o
#PJM -L node=1
#PJM --mpi proc=12
#PJM -L elapse=00:15:00
#PJM -g gt36
#PJM -j
#PJM -e err
#PJM -o test.lst

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

a048.sh

```
#!/bin/sh
#PJM -N "test"
#PJM -L rscgrp=lecture6-o
#PJM -L node=1
#PJM --mpi proc=48
#PJM -L elapse=00:15:00
#PJM -g gt36
#PJM -j
#PJM -e err
#PJM -o test.lst

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

a384.sh

```
#!/bin/sh
#PJM -N "test"
#PJM -L rscgrp=lecture6-o
#PJM -L node=8
#PJM --mpi proc=384
#PJM -L elapse=00:15:00
#PJM -g gt36
#PJM -j
#PJM -e err
#PJM -o test.lst

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

a576.sh

```
#!/bin/sh
#PJM -N "test"
#PJM -L rscgrp=lecture6-o
#PJM -L node=12
#PJM --mpi proc=576
#PJM -L elapse=00:15:00
#PJM -g gt36
#PJM -j
#PJM -e err
#PJM -o test.lst

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

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

Number of Processes

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

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