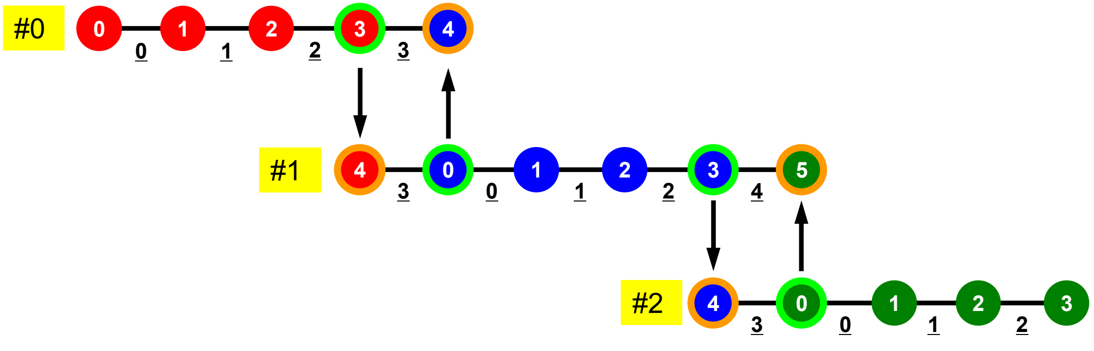
What is Point-to-Point Comm. ?

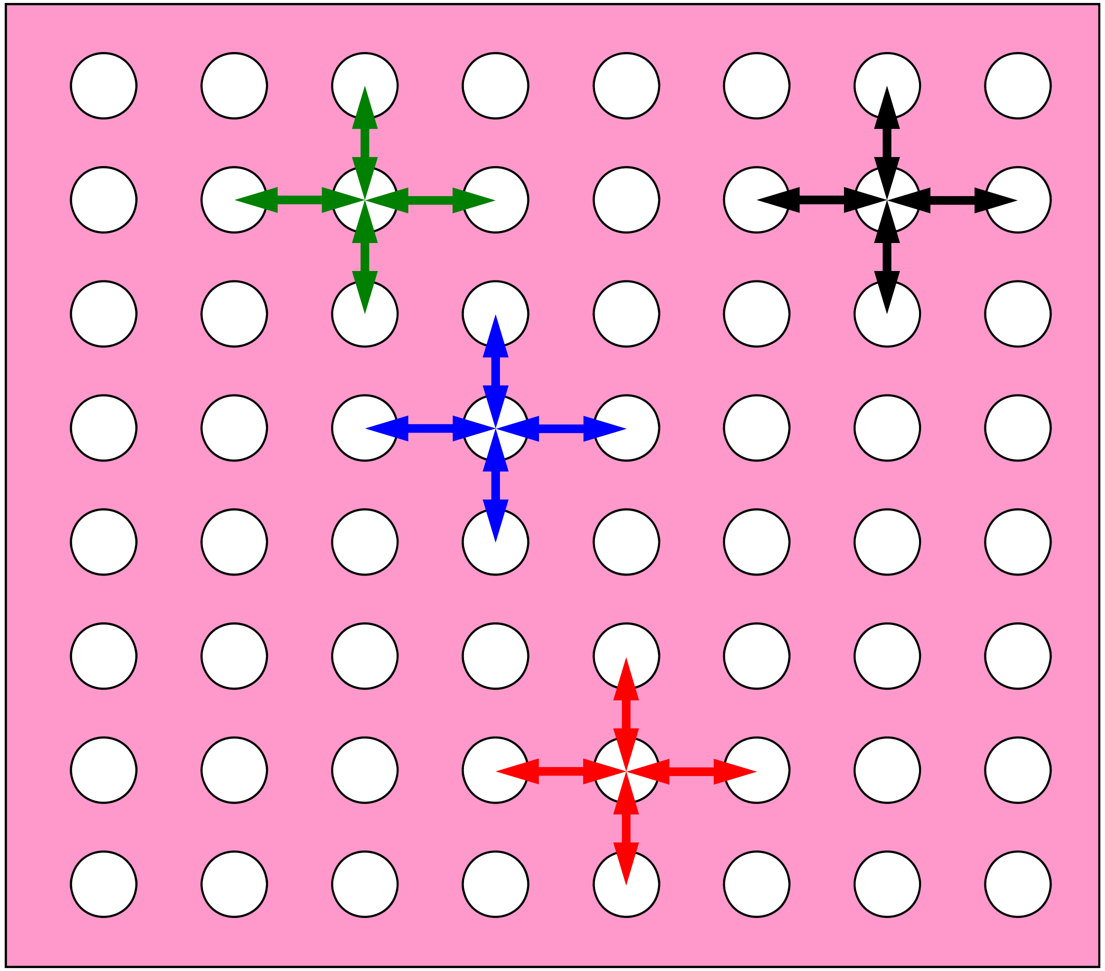
- Collective Communication
  - MPI_Reduce, MPI_Scatter/Gather etc.
  - Communications with all processes in the communicator
  - Application Area
    - BEM, Spectral Method, MD: global interactions are considered
    - Dot products, MAX/MIN: Global Summation & Comparison

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



Collective/**PtoP** Communications

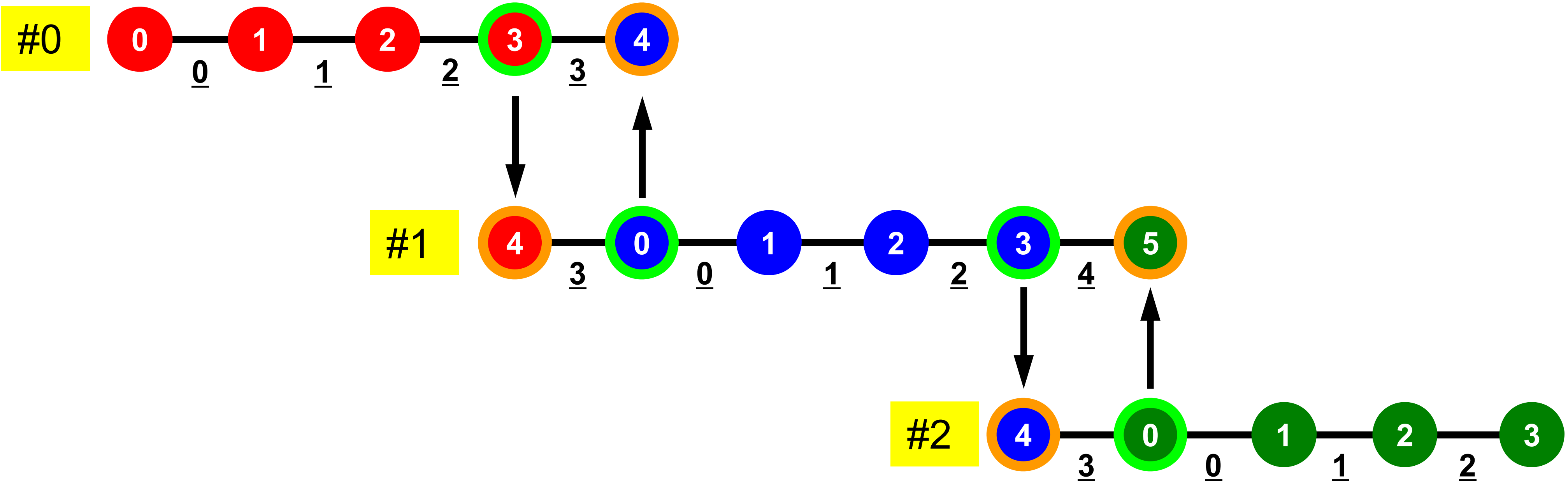
Interactions with only Neighboring Processes/Element Finite Difference Method (FDM), Finite Element Method (FEM)



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

Info in neighboring domains is required for FEM operations

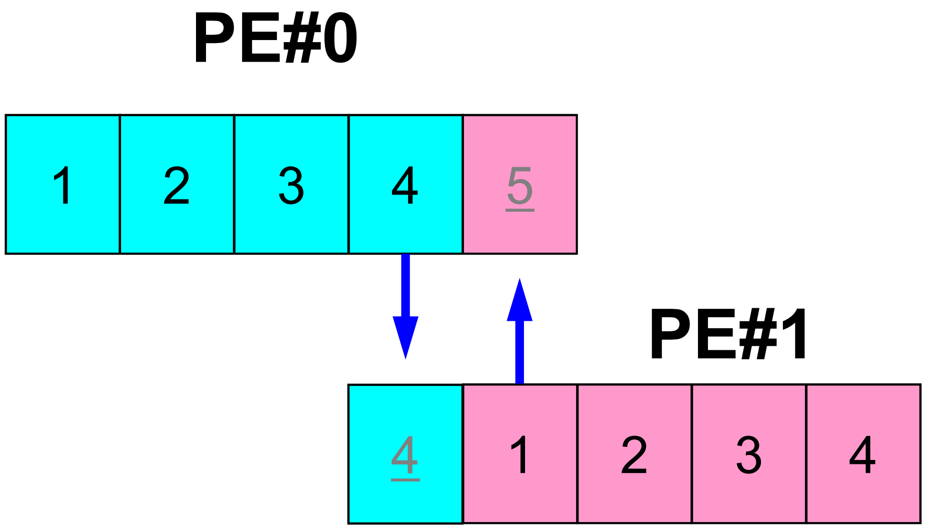
Matrix assembling, Iterative Method



Method for PtoP 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.
- Please just remember that “there are such functions”.

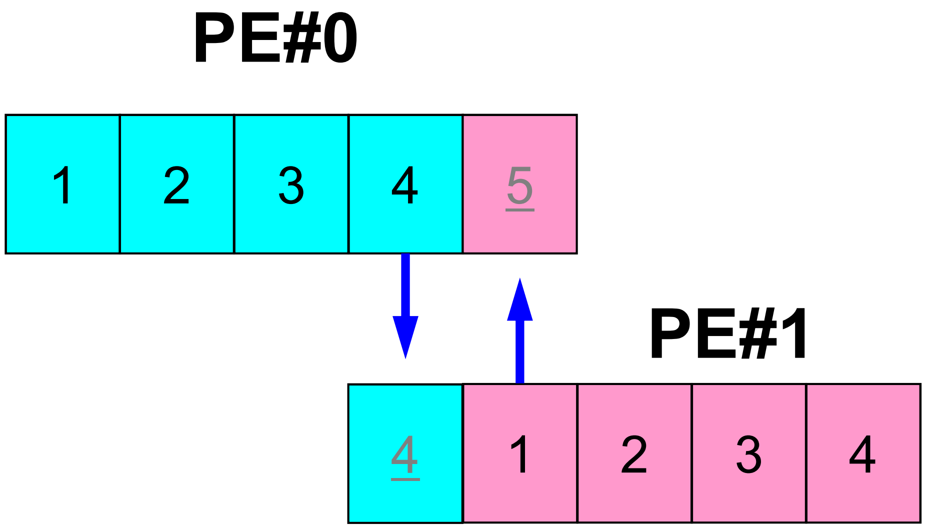
MPI_Send/MPI_Recv



```
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).

MPI_Send/MPI_Recv (cont.)

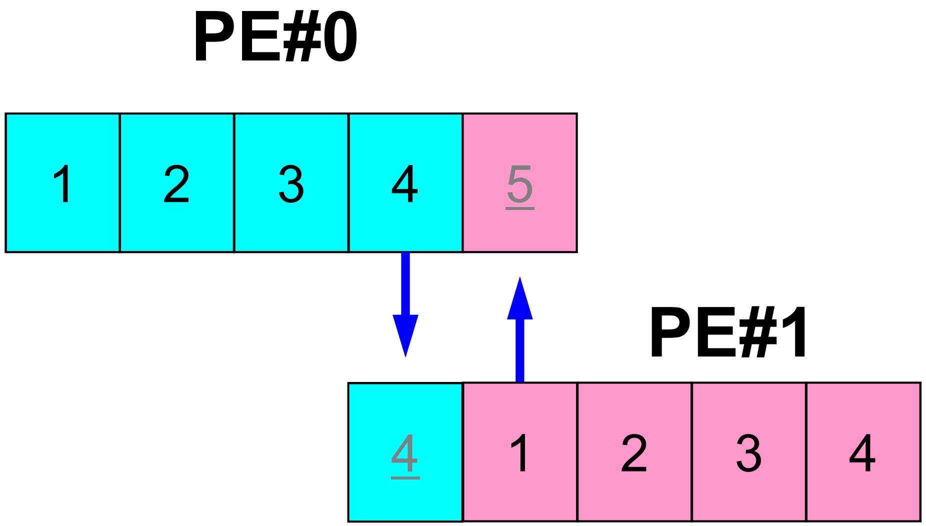


```
if (my_rank.eq.0) NEIB_ID=1
if (my_rank.eq.1) NEIB_ID=0

…
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 ... but

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 synchronizing **MPI_Isend** and **MPI_Irecv** in this class.
- 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_Waitall` (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_Waitall` (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

C

MPI_Sendrecv

- MPI_Send+MPI_Recv : <u>not recommended, many restrictions</u>

- **`MPI_Sendrecv (sendbuf,sendcount,sendtype,dest,sendtag,recvbuf, recvcount,recvtype,source,recvtag,comm,status)`**

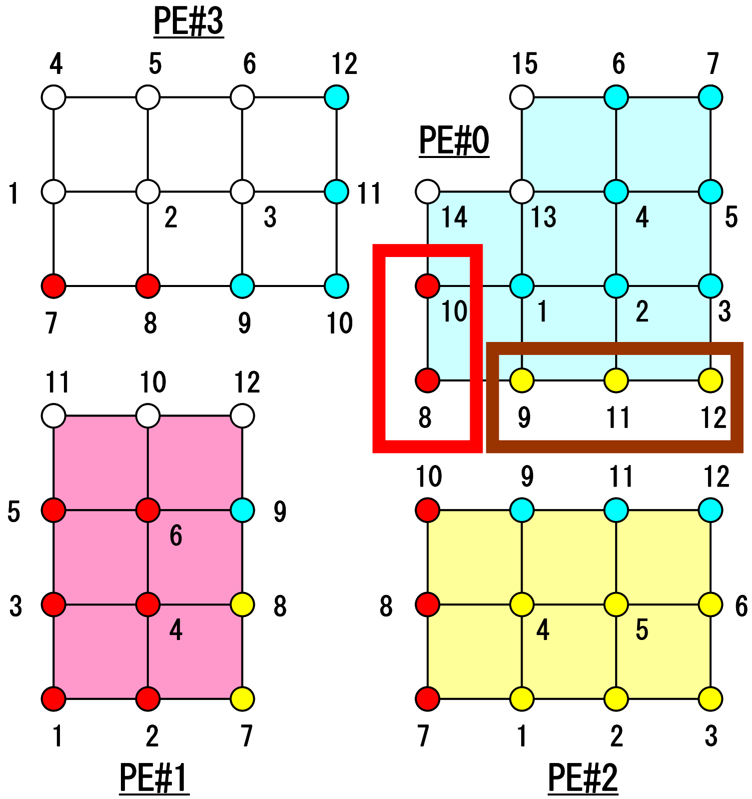
– **<u>`sendbuf`</u>**	choice	I	starting address of sending buffer
– **<u>`sendcount`</u>**	int	I	number of elements in sending buffer
– **<u>`sendtype`</u>**	MPI_Datatype	I	datatype of each sending buffer element
– **<u>`dest`</u>**	int	I	rank of destination
– **<u>`sendtag`</u>**	int	I	message tag for sending
– **<u>`comm`</u>**	MPI_Comm	I	communicator
– **<u>`recvbuf`</u>**	choice	I	starting address of receiving buffer
– **<u>`recvcount`</u>**	int	I	number of elements in receiving buffer
– **<u>`recvtype`</u>**	MPI_Datatype	I	datatype of each receiving buffer element
– **<u>`source`</u>**	int	I	rank of source
– **<u>`recvtag`</u>**	int	I	message tag for receiving
– **<u>`comm`</u>**	MPI_Comm	I	communicator
– **<u>`status`</u>**	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

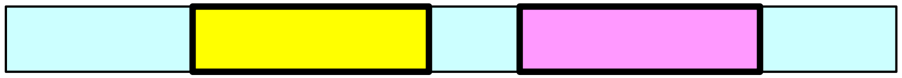


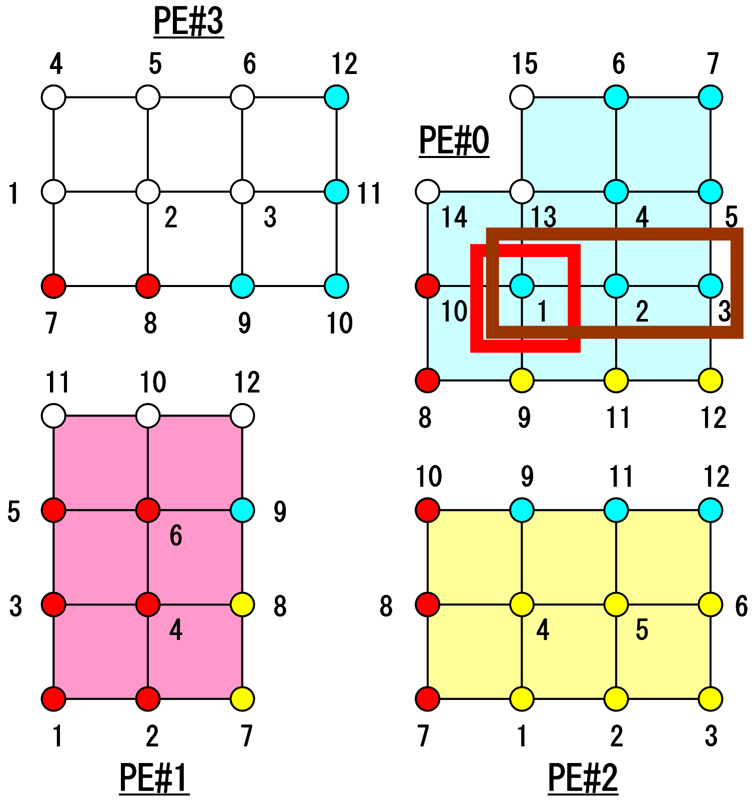
SEND: sending from boundary nodes

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));
```