Introduction to Parallel Programming for Multicore/Manycore Clusters

Part II-3: Parallel FVM using MPI

Kengo Nakajima
Information Technology Center
The University of Tokyo

Overview

- Introduction
- Local Data Structure & Communication
  - 1D
  - 2D

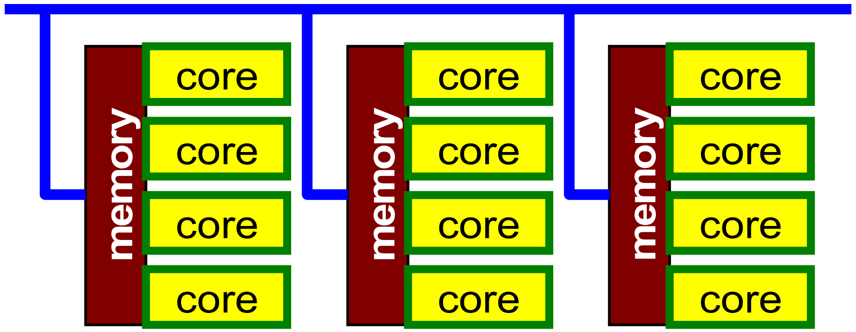
Goal of the Last Part

- FVM Code with OpenMP/MPI Hybrid Parallel Programming Model based-on the Initial Code (L1-sol on the first day)

- Diagonal/Point Jacobi Preconditioning (METHOD=3)
  - OpenMP: Straight Forward
    - NO Data Dependency
    - Just insert OpenMP Directives
  - MPI
    - Distributed Computation
    - Special Data Structure

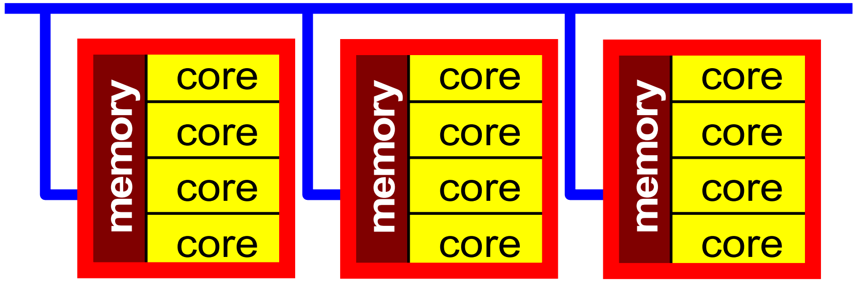
<u>Flat-MPI：Each Core -> Independent</u>

- MPI only
- Intra/Inter Node



<u>Hybrid：Hierarchal Structure</u>

- OpenMP
- MPI

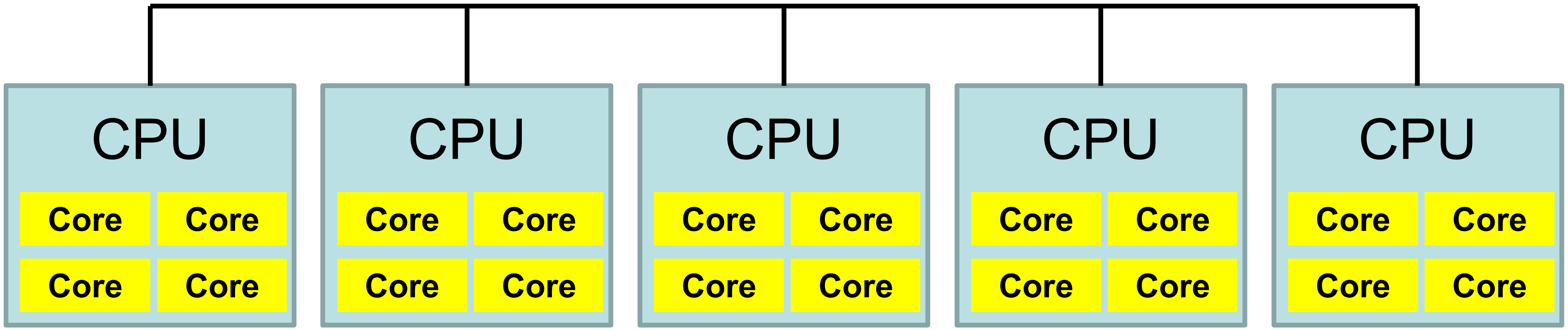


Some Technical Terms

- Processor, Core
  - Processing Unit (H/W), Processor=Core for single-core proc’s
- Process
  - Unit for MPI computation, nearly equal to “core”
  - Each core (or processor) can host multiple processes (but not efficient)
- PE (Processing Element)
  - PE originally mean “processor”, but it is sometimes used as “process” in this class. Moreover it means “domain” (next)
    - In multicore proc’s: PE generally means “core”
- Domain
  - domain=process (=PE), each of “MD” in “SPMD”, each data set
- Process ID of MPI (ID of PE, ID of domain) starts from “0”
  - if you have 8 processes (PE’s, domains), ID is 0~7

Parallel Computing on Distributed Memory Architecture

- Faster, Larger & More Complicated
- Scalability
  - Solving N^x scale problem using N^x computational resources during same computation time
    - for large-scale problems: **Weak Scaling**
    - e.g. CG solver: more iterations needed for larger problems
  - Solving a problem using N^x computational resources during 1/N computation time
    - for faster computation: **Strong Scaling**

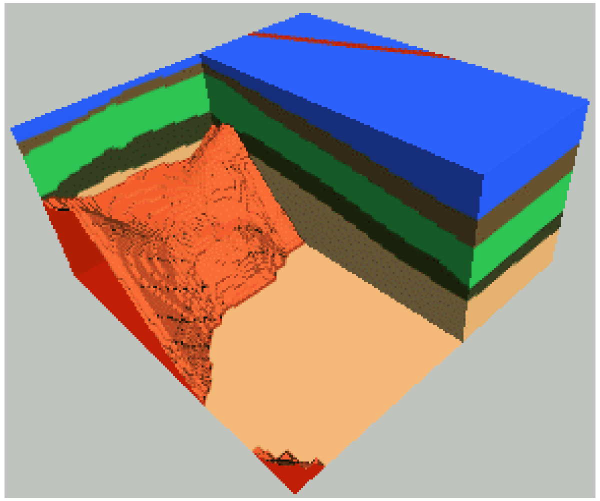


What is “Parallel” Computing ? (1/2)

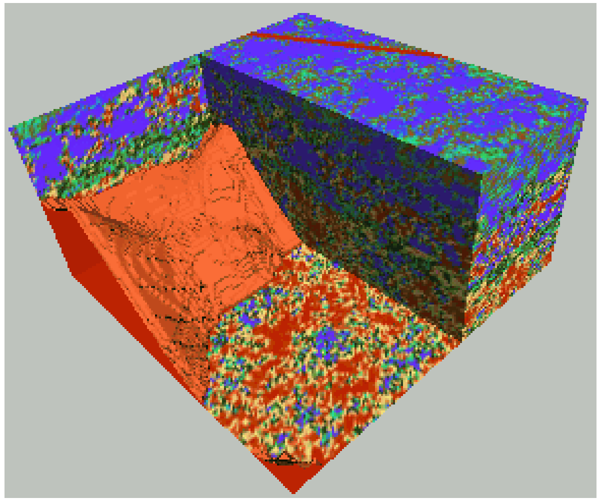
- to solve larger problems faster

Homogeneous/Heterogeneous Porous Media

Lawrence Livermore National Laboratory



Homogeneous

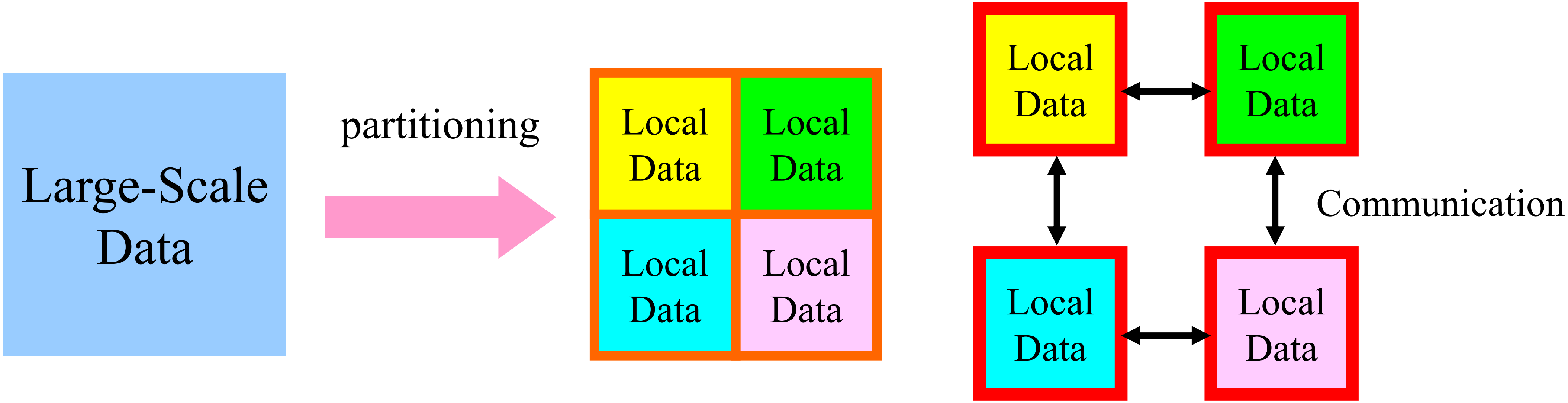


Heterogeneous

very fine meshes are required for simulations of heterogeneous field.

What is “Parallel” Computing ? (2/2)

- PC with 1GB memory : 1M meshes are the limit for FEM
  - Southwest Japan with 1,000km x 1,000km x 100km in 1km mesh -> 10^8 meshes
- Large Data -> Domain Decomposition -> Local Operation
- Inter-Domain Communication for Global Operation

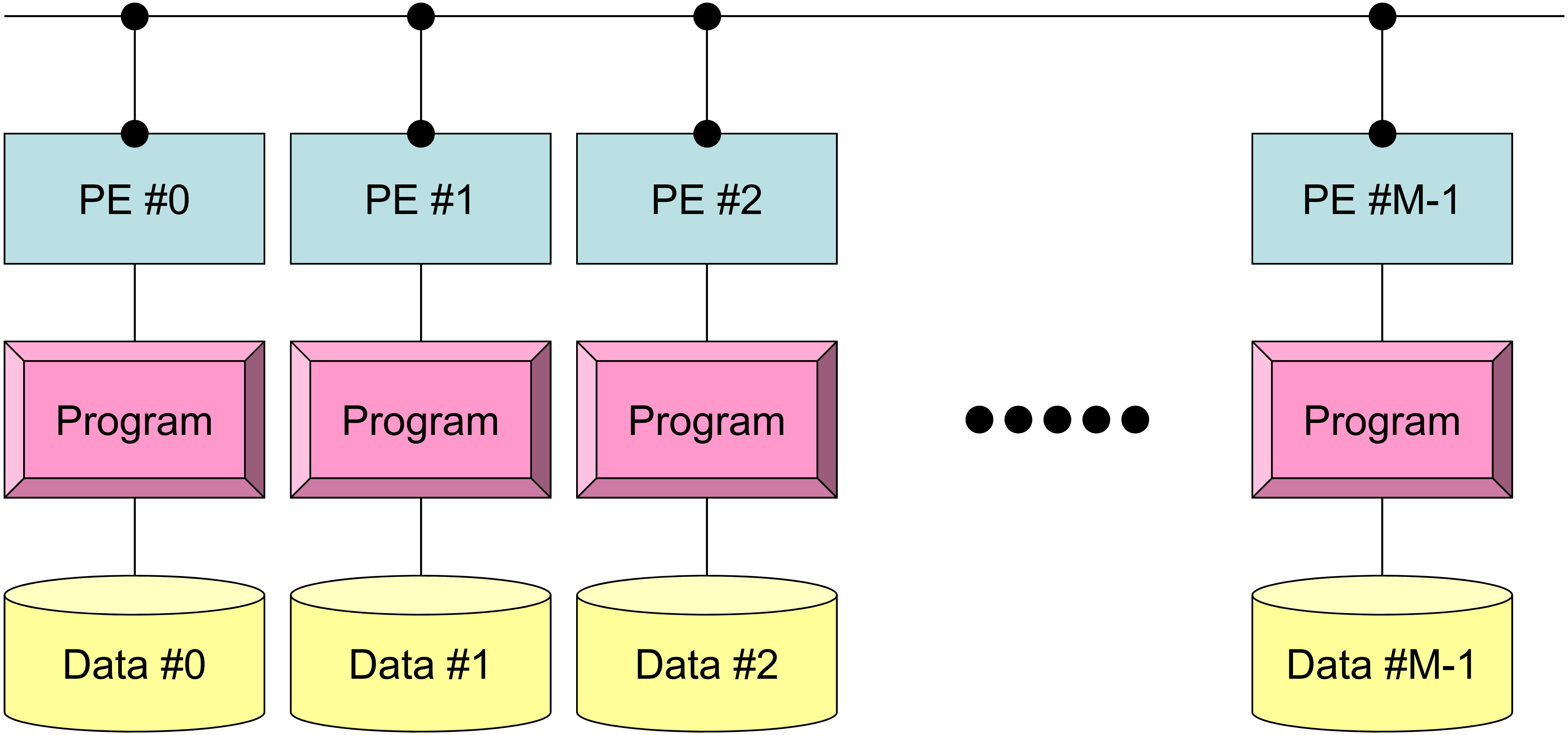


PE: Processing Element
Processor, Domain, Process

SPMD

You understand 90% MPI, if you understand this figure.

```
mpirun -np M <Program>
```



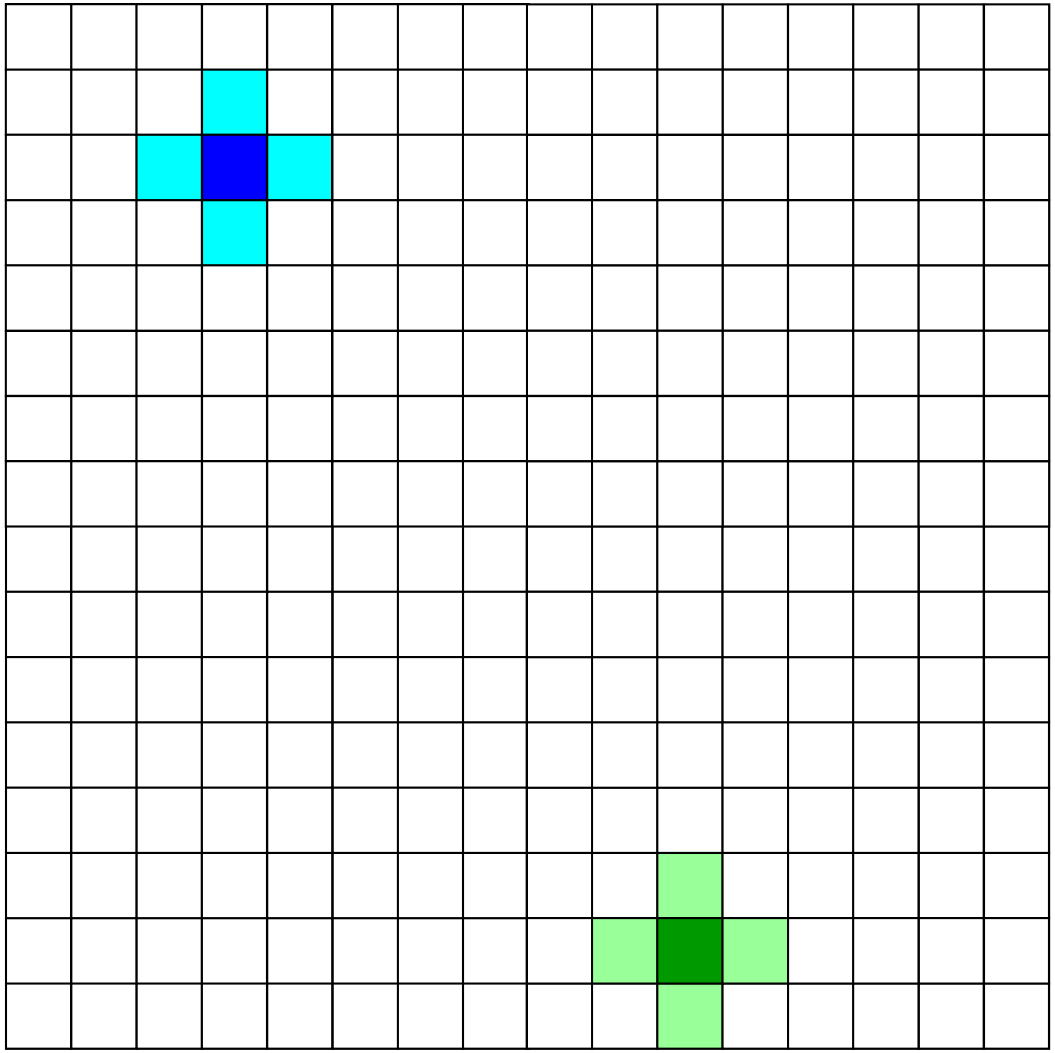
Each process does same operation for different data

Large-scale data is decomposed, and each part is computed by each process
It is ideal that parallel program is not different from serial one except communication.

What is Communication ?

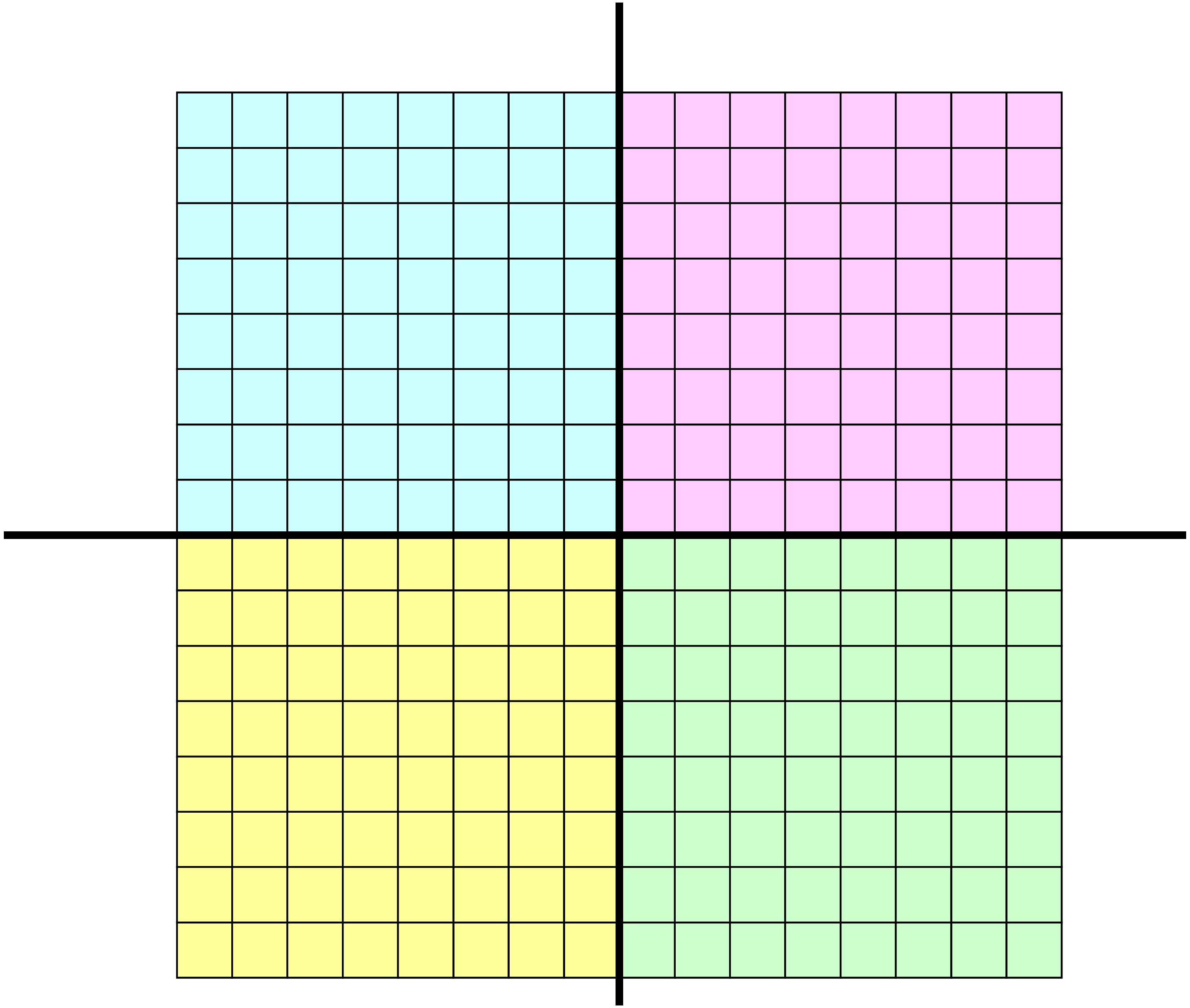
- Parallel Computing -> Local Operations
- Communications are required in Global Operations for Consistency.

Local Data Structures for Parallel FVM/FDM using Krylov Iterative Solvers Example: 2D Mesh (5-point stencil)



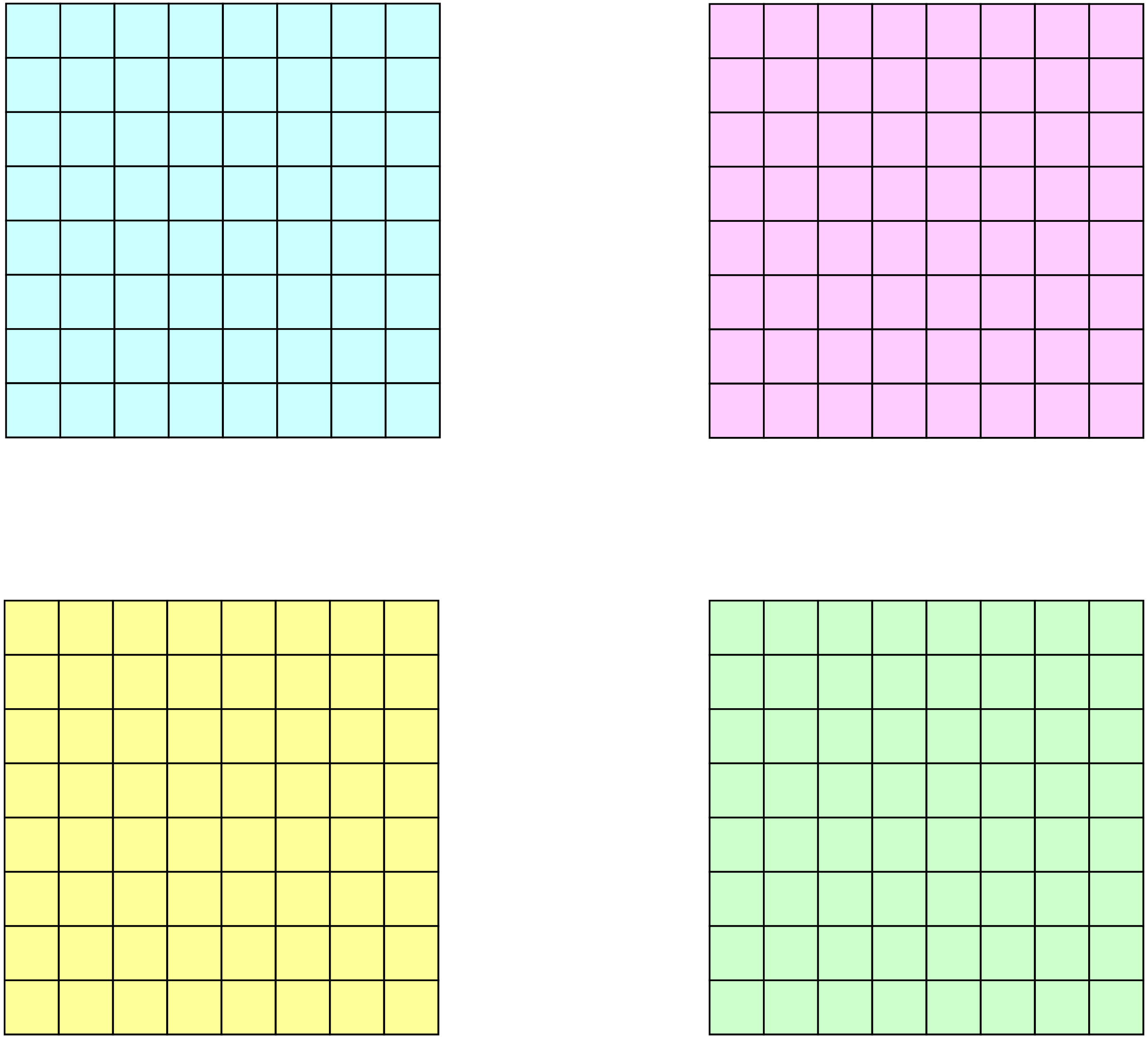
Example: 2D FDM Mesh (5-point stencil)

4-regions/domains



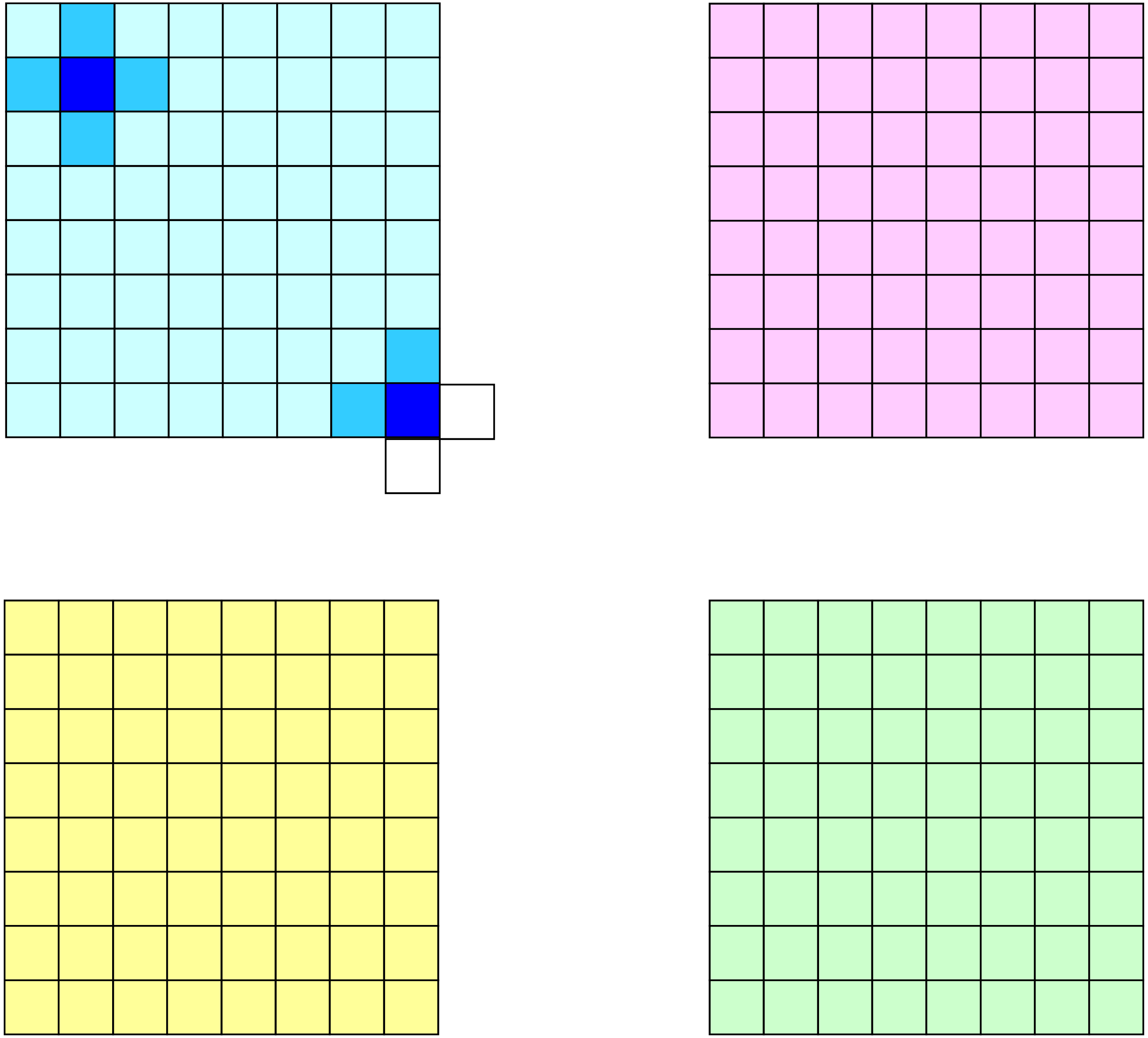
Example: 2D Mesh (5-point stencil)

4-regions/domains



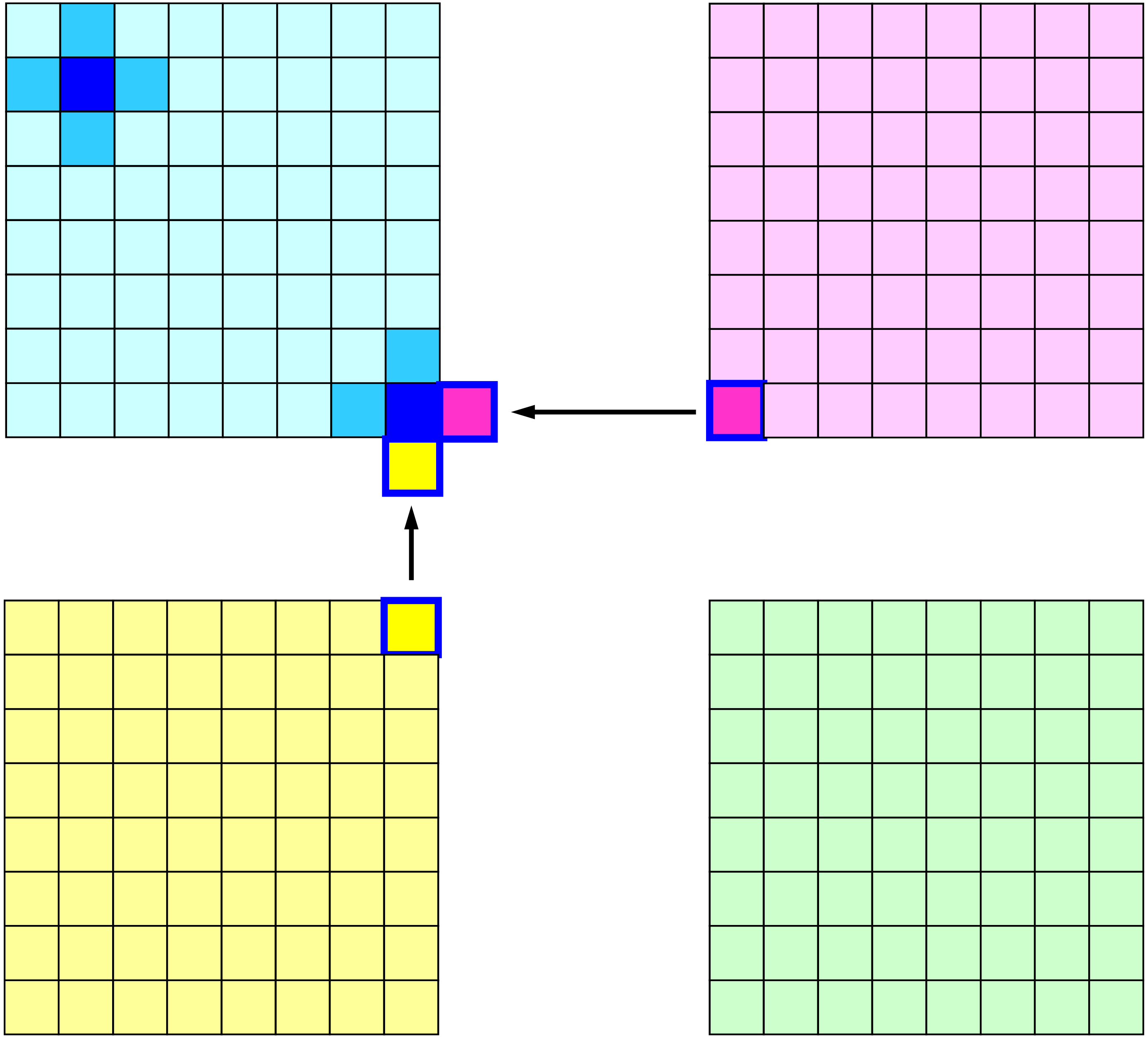
Example: 2D Mesh (5-point stencil)

meshes at domain boundary need info. neighboring domains



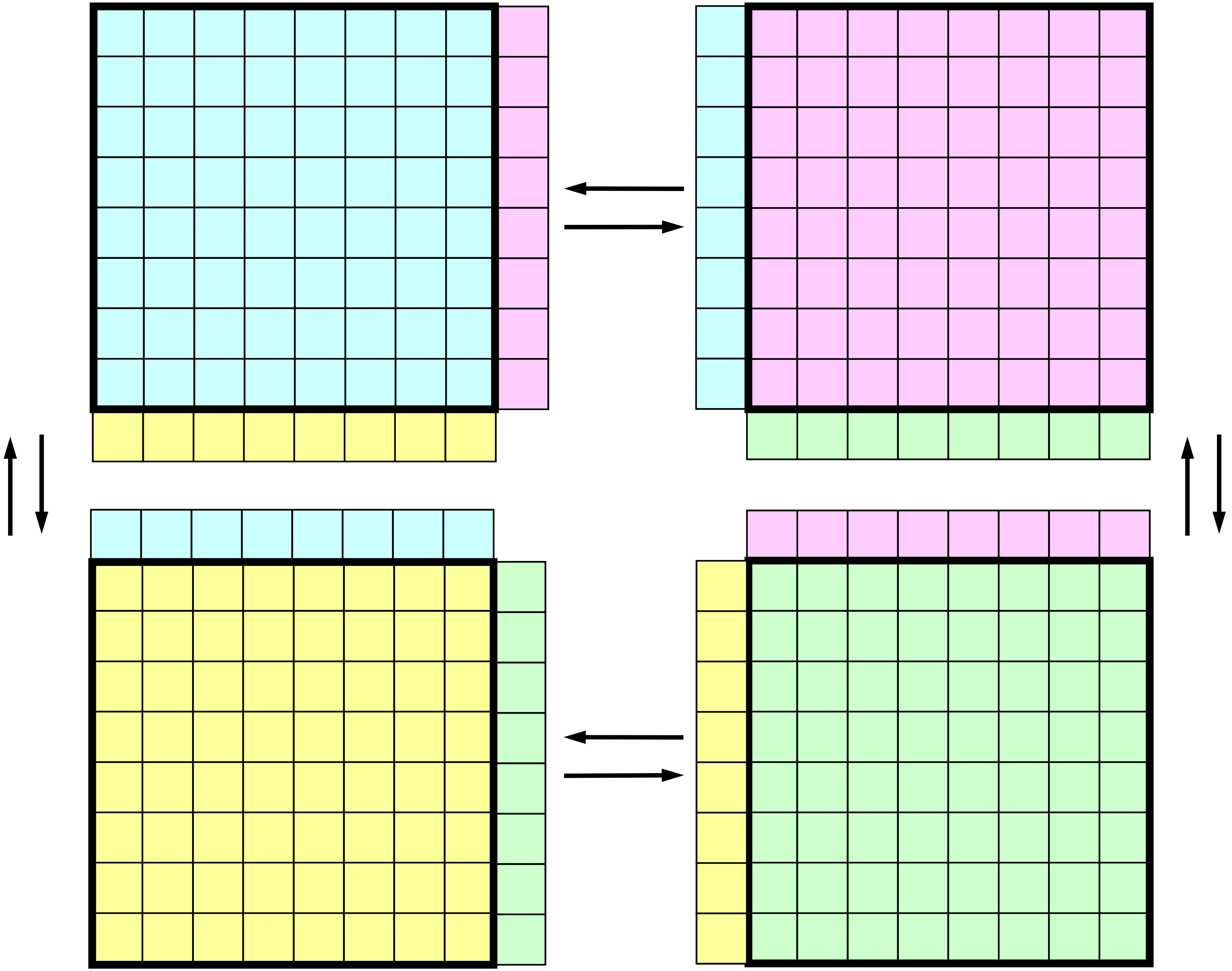
Example: 2D Mesh (5-point stencil)

meshes at domain boundary need info. neighboring domains

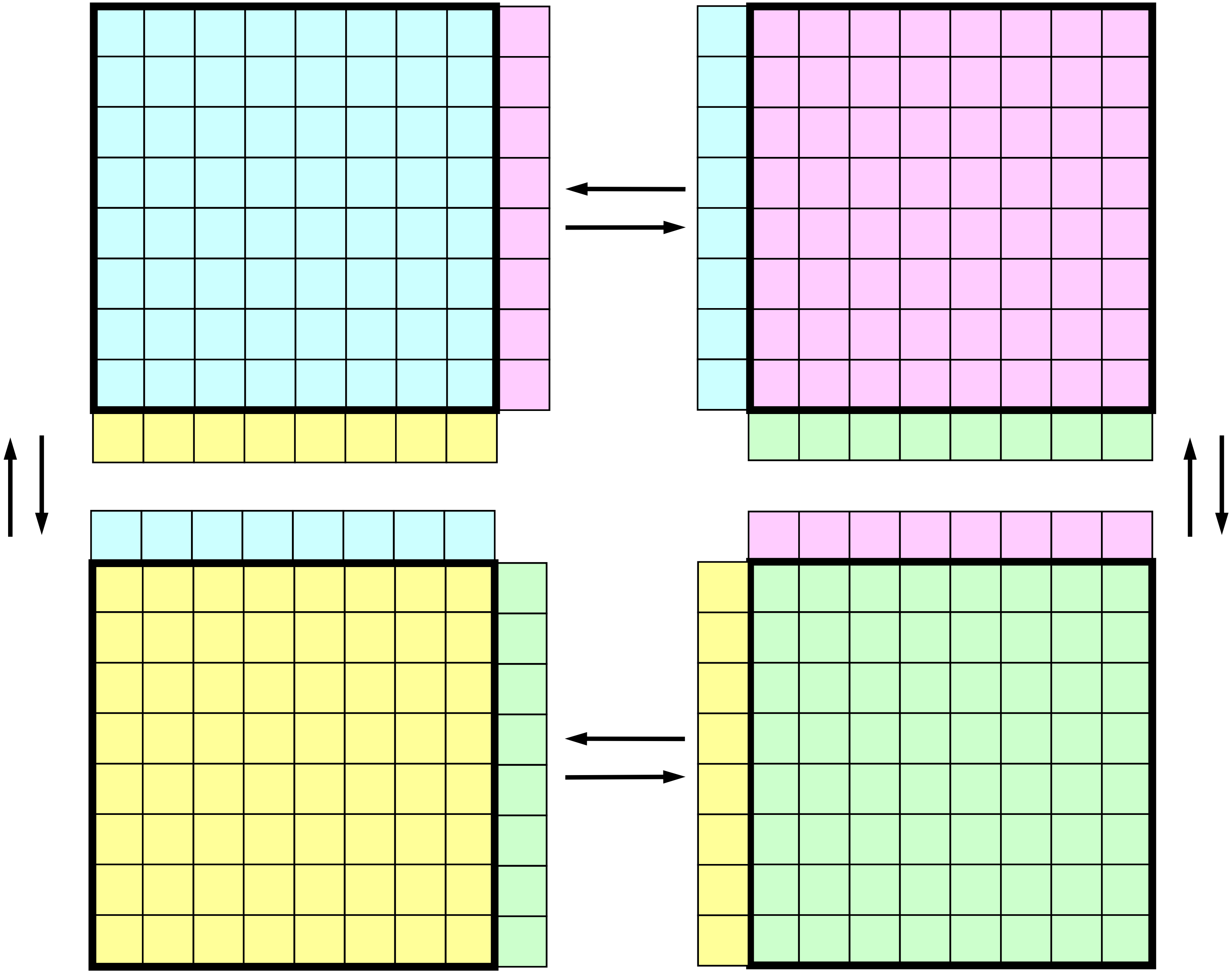


Example: 2D Mesh (5-point stencil)

communications using “HALO (overlapped meshes)”



Coefficient Matrices for □ can be locally generated on each partition by this data structure



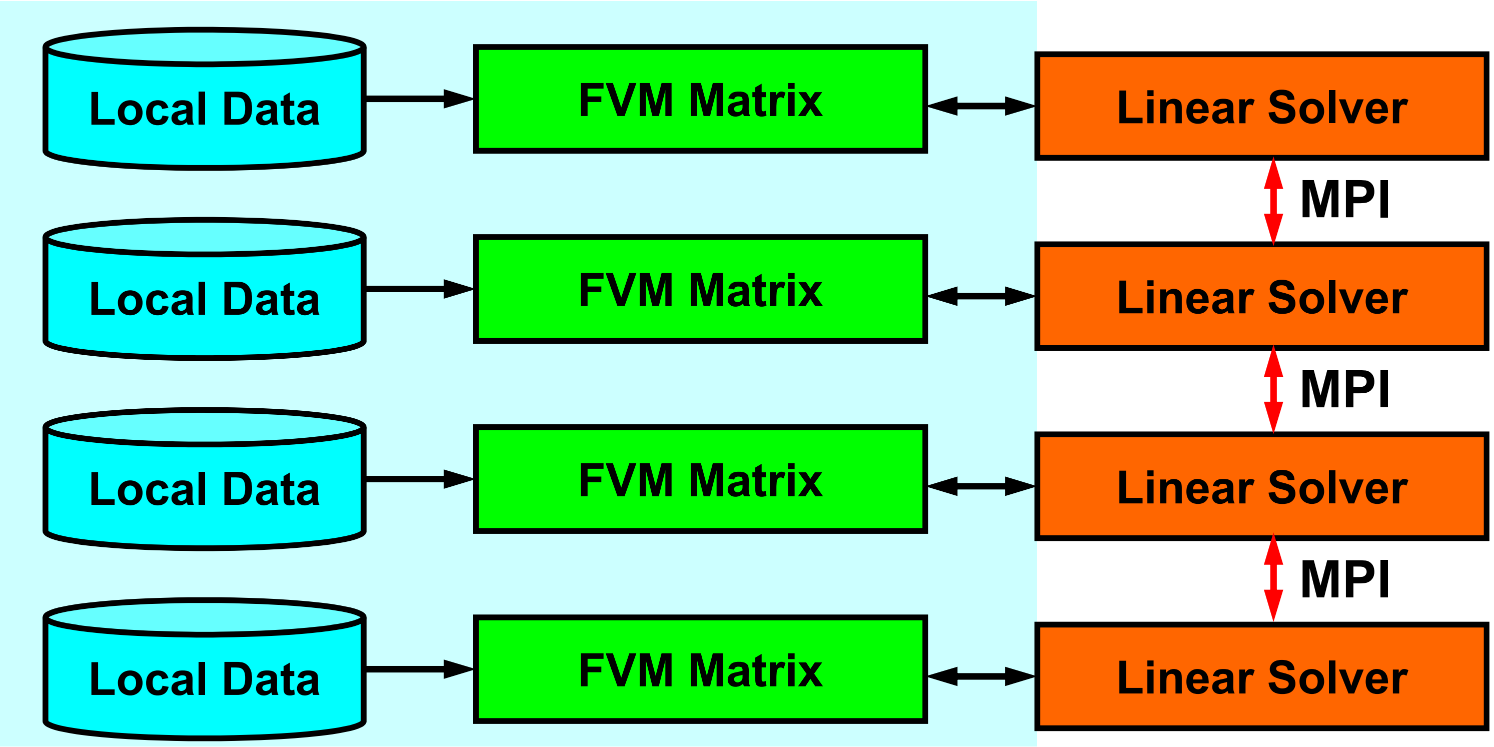
Operations in Parallel FVM

SPMD: Single-Program Multiple-Data

Large Scale Data -> partitioned into Distributed Local Data Sets.

FVM code can assembles coefficient matrix for each local data set : this part could be completely local, same as serial operations

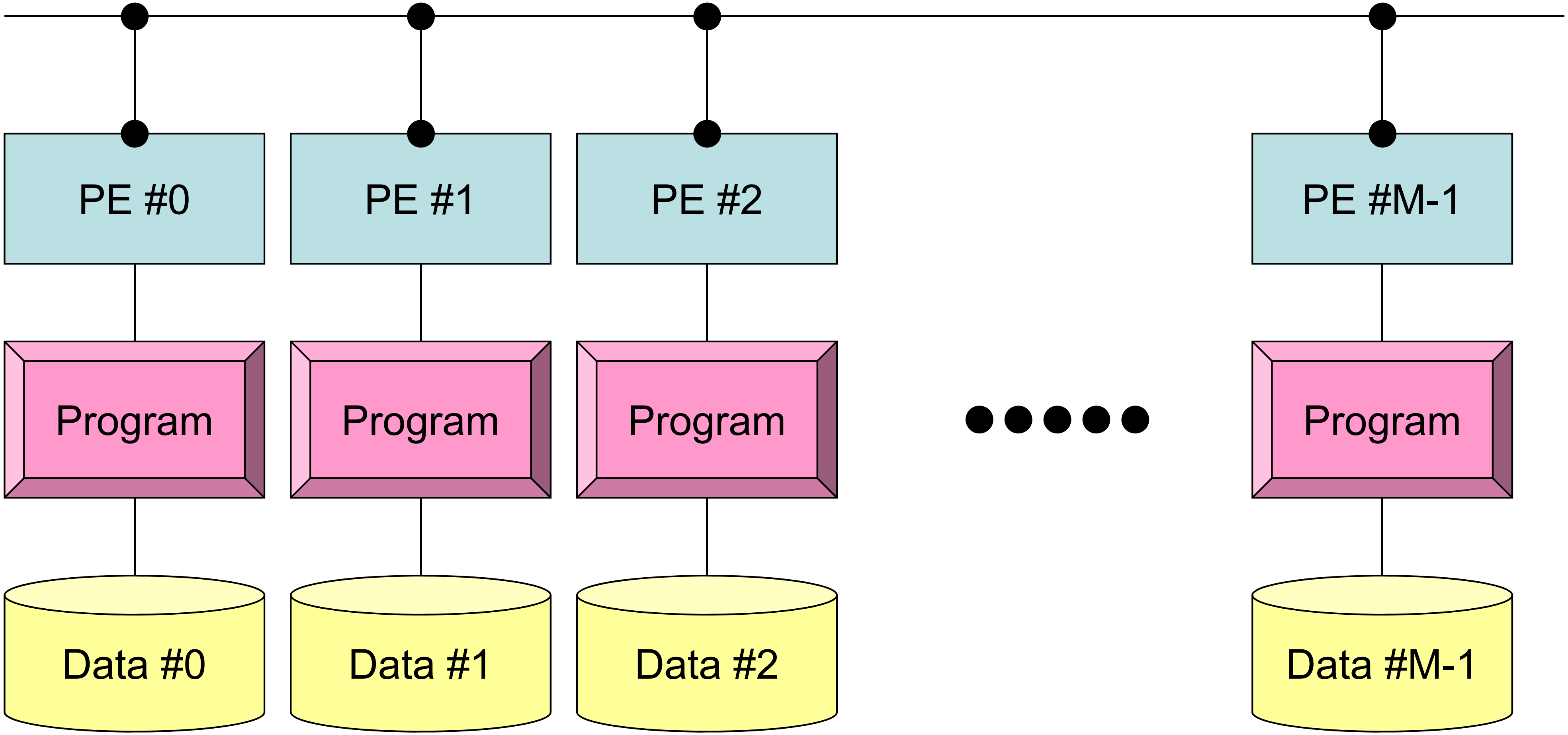
Global Operations & Communications happen only in Linear Solvers dot products, matrix-vector multiply, preconditioning



PE: Processing Element
Processor, Domain, Process

SPMD

```
mpirun -np M <Program>
```



Each process does same operation for different data

Large-scale data is decomposed, and each part is computed by each process

It is ideal that parallel program is not different from serial one except communication.

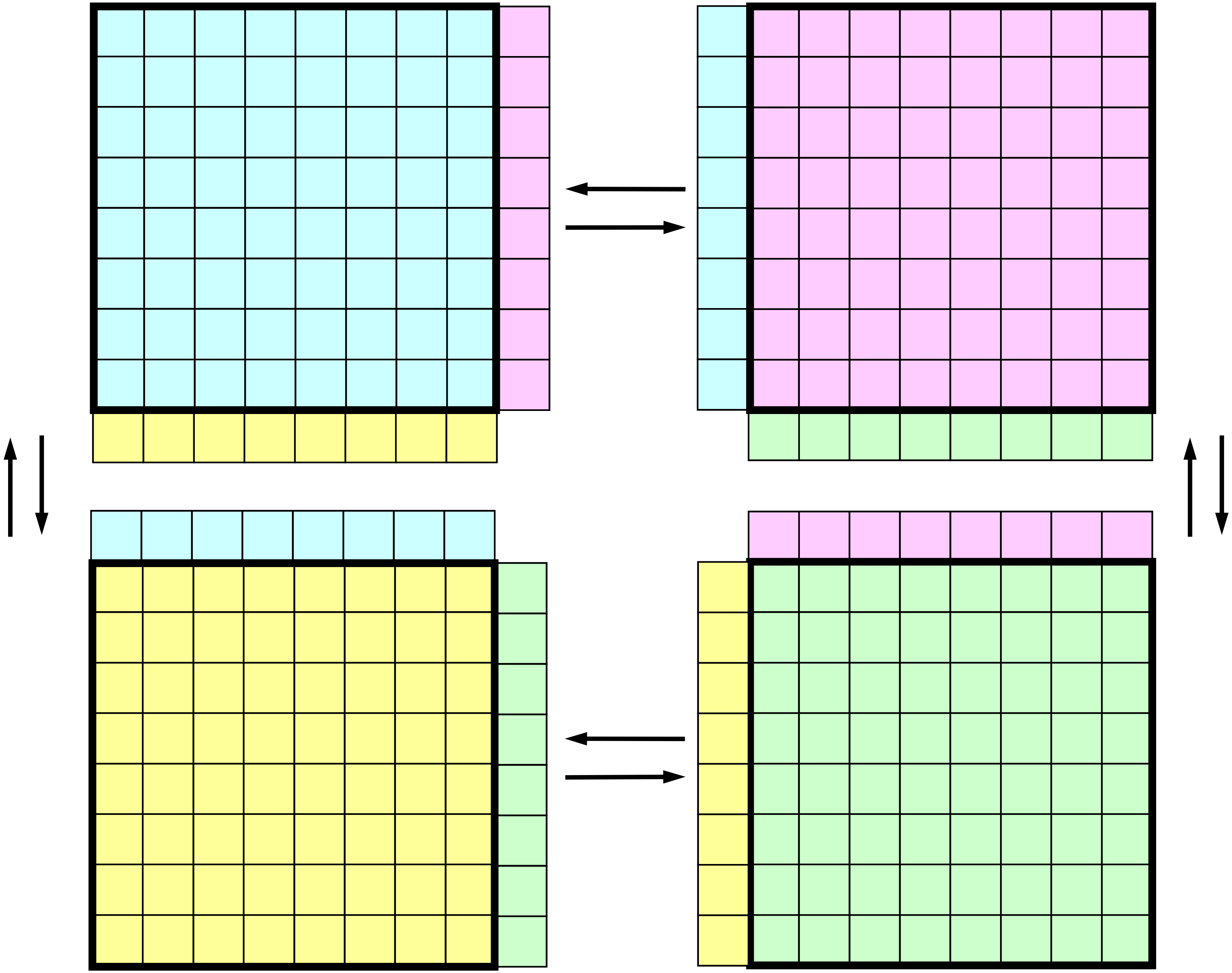
Parallel FVM Procedures

- Design on “Local Data Structure” is important
  - for SPMD-type operations in the previous page

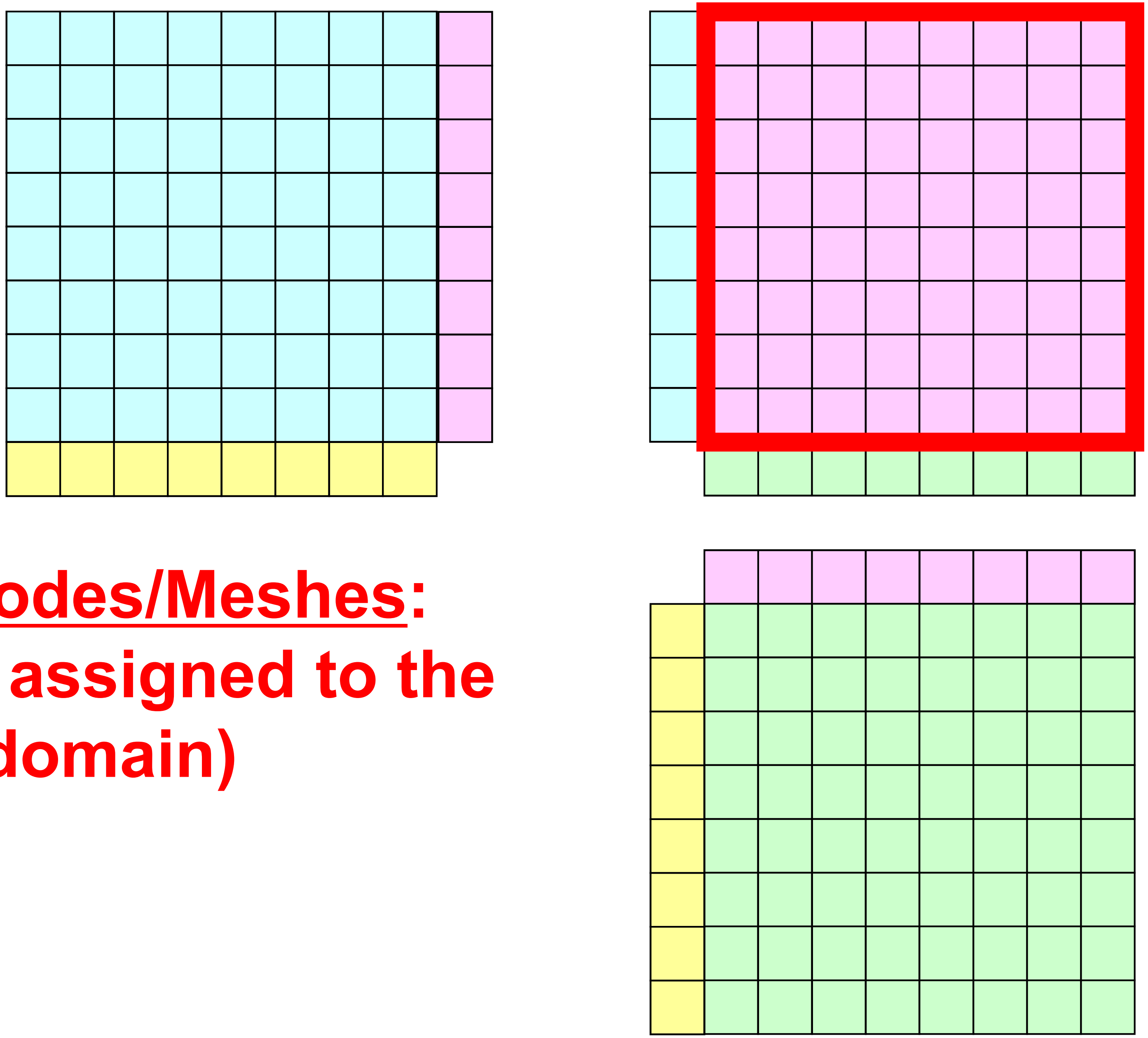
- Matrix Generation
- Preconditioned Iterative Solvers for Linear Equations

Example: 2D Mesh (5-point stencil)

communications using “HALO (overlapped meshes)”

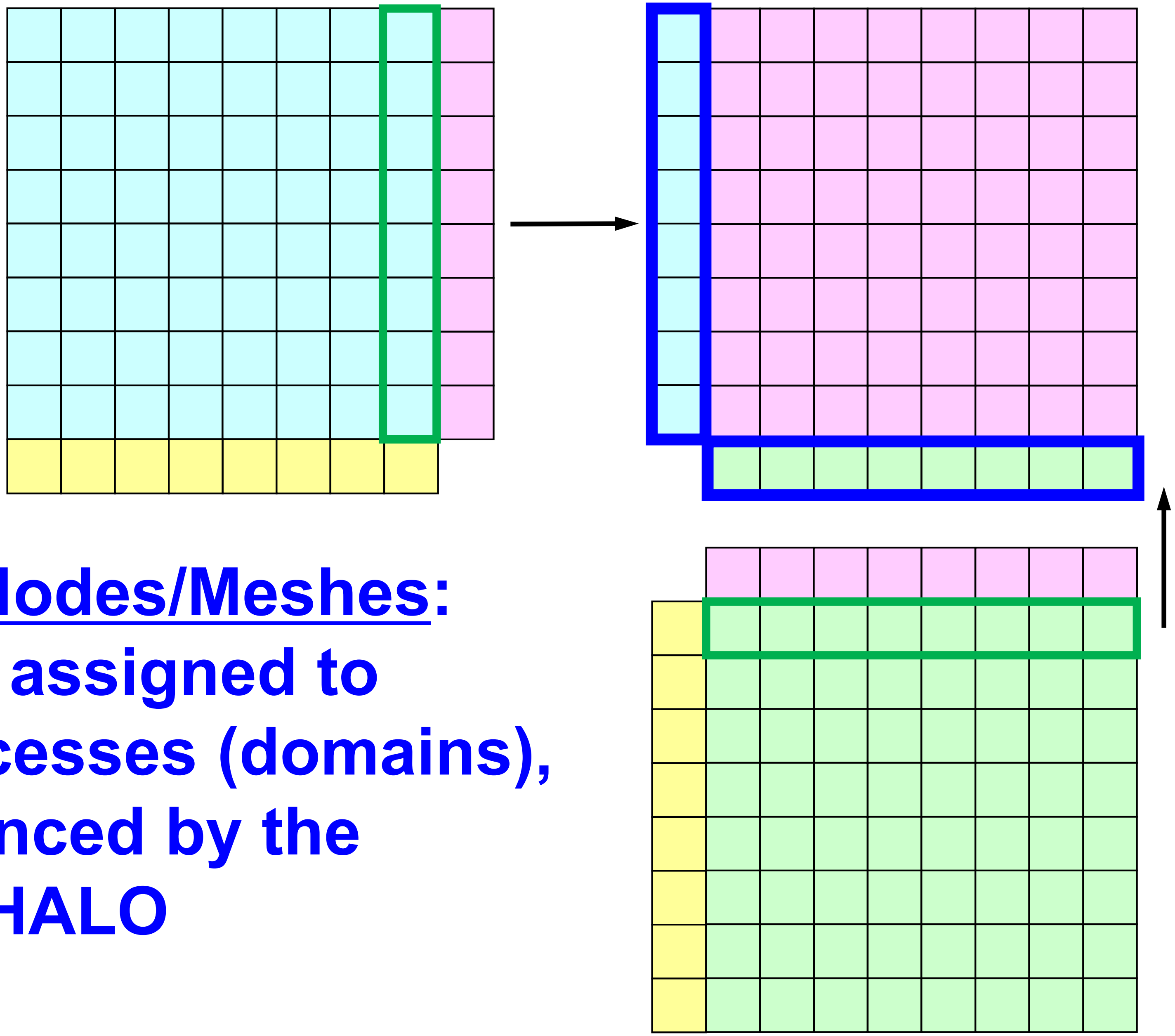


Internal / External / Boundary Nodes



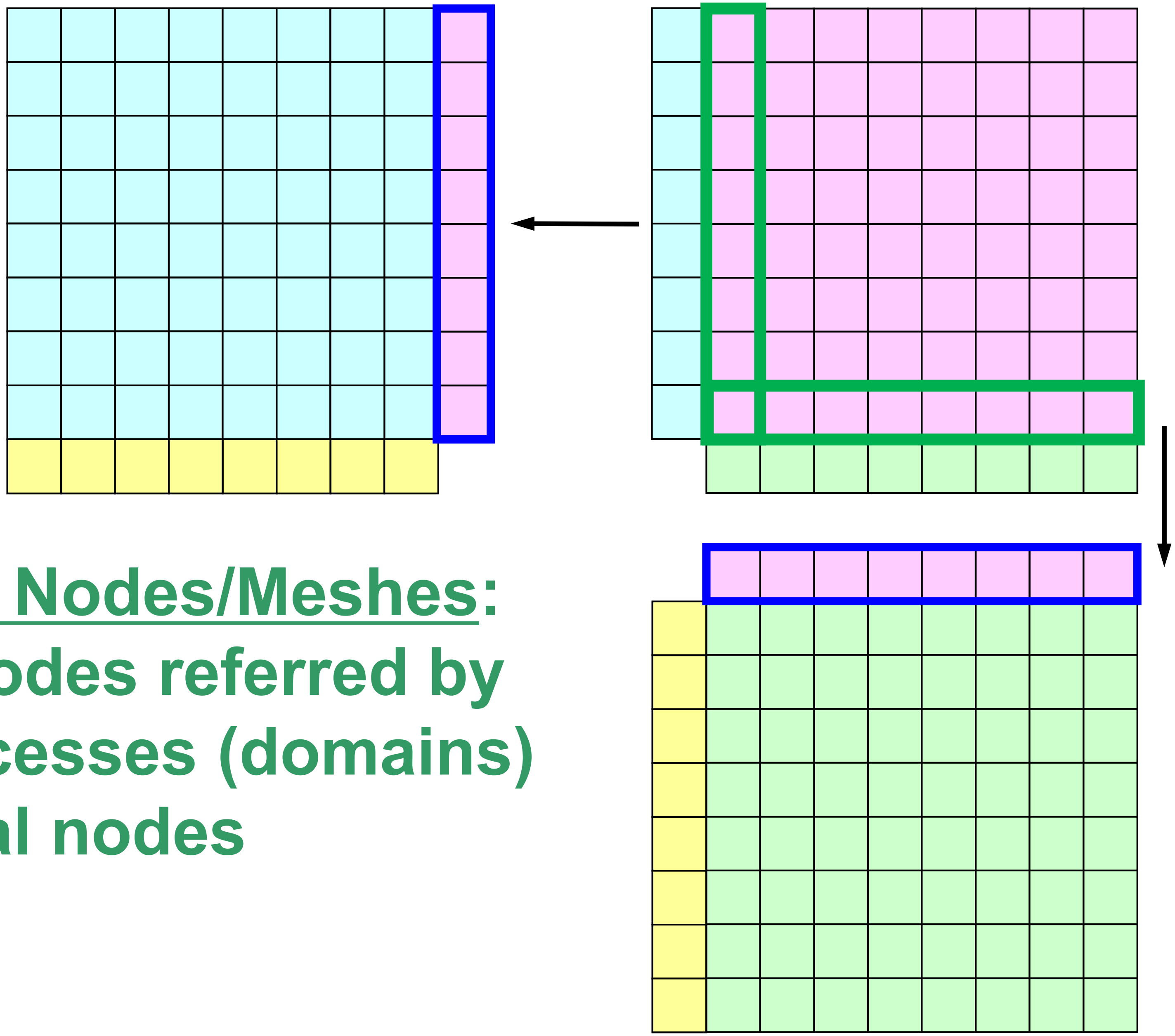
Internal Nodes/Meshes: Originally assigned to the process (domain)

Internal / External / Boundary Nodes



External Nodes/Meshes:
Originally assigned to other processes (domains), but referenced by the process: HALO

Internal / External / Boundary Nodes



Boundary Nodes/Meshes:
Internal nodes referred by other processes (domains) as external nodes

What is Communications ?

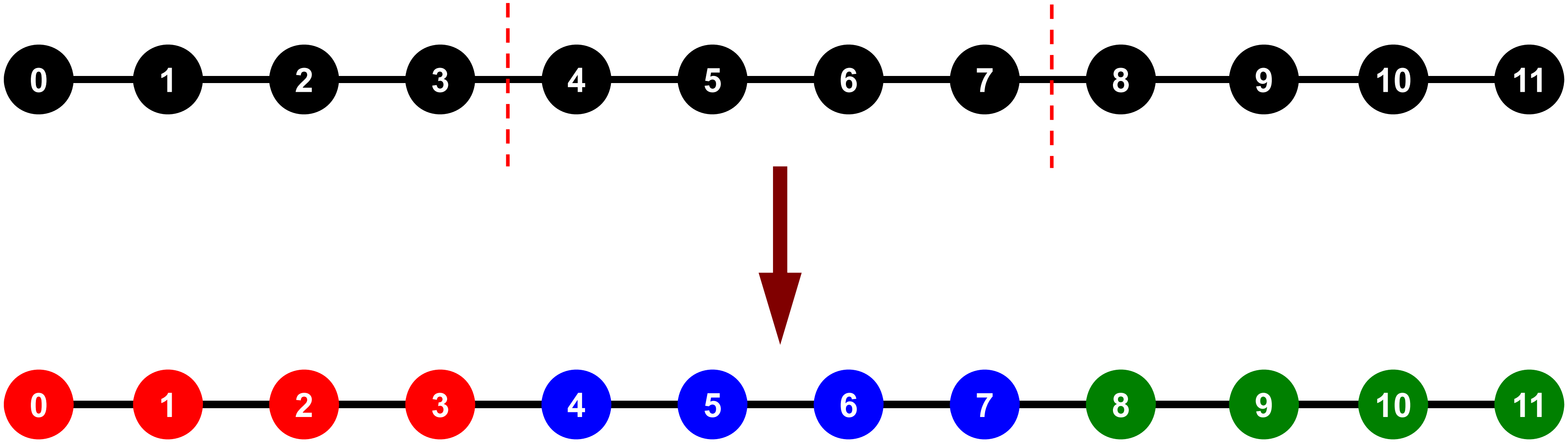
- Getting information of “external nodes” from external partitions (local data)

- In this study, “Generalized Communication Tables” contain the information

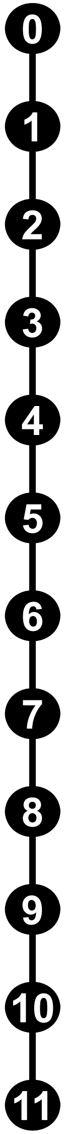
- Introduction
- Quick Overview of MPI

- **Local Data Structure & Communication**
  - **1D**
  - 2D

1D FVM: 12 meshes/3 domains



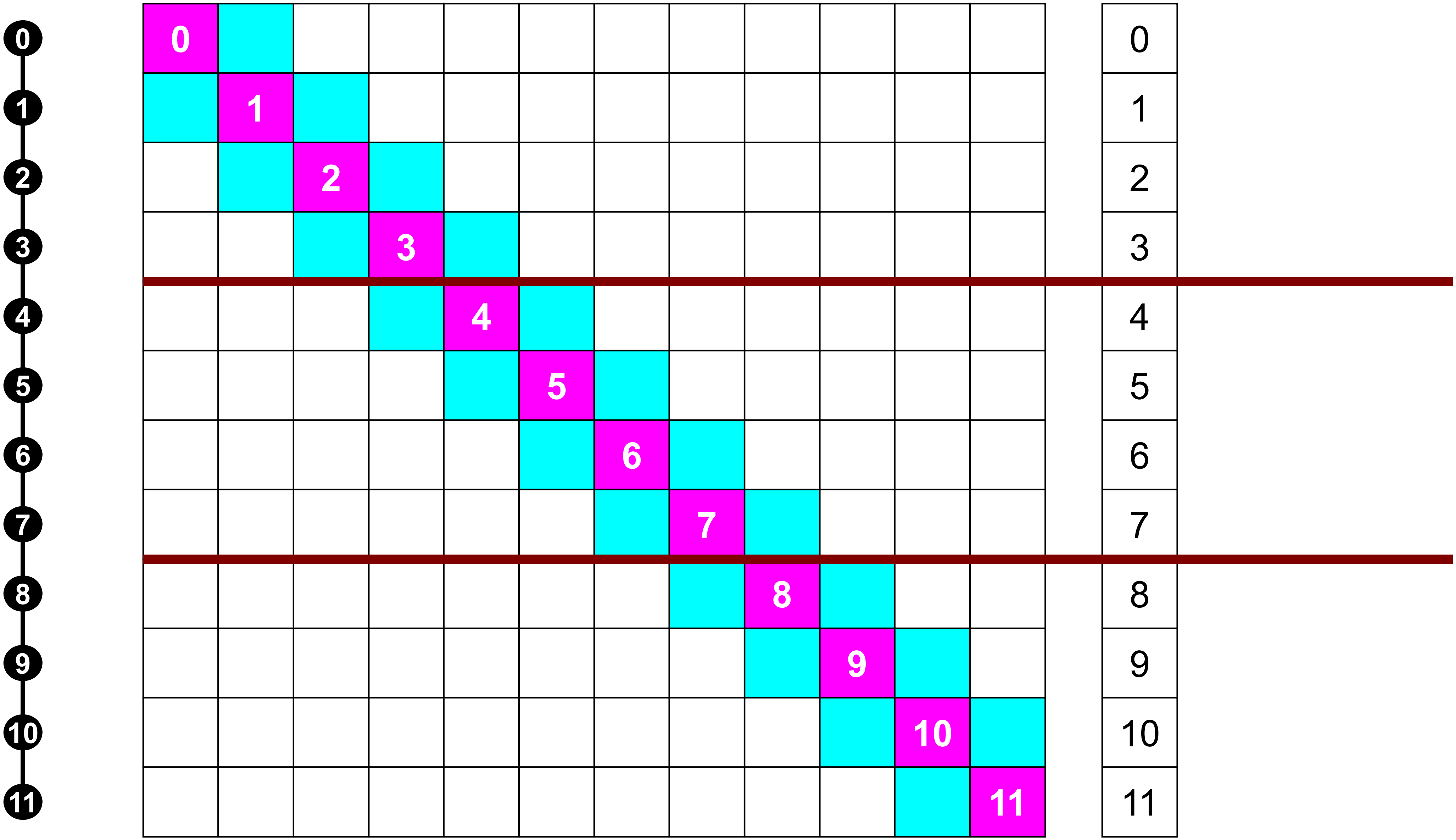
1D FVM: 12 meshes/3 domains



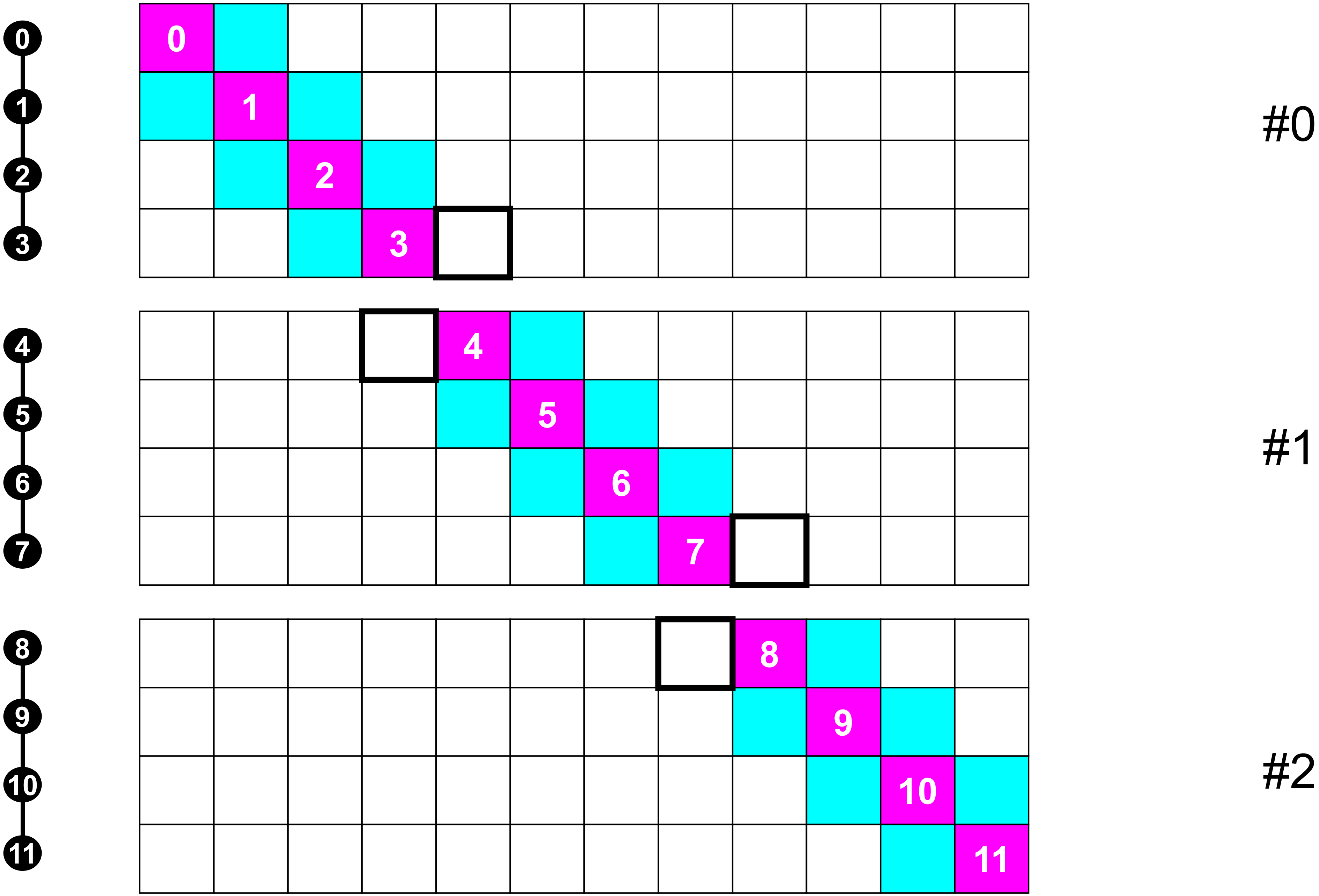
0	🟦										
🟦	1	🟦									
	🟦	2	🟦								
		🟦	3	🟦							
			🟦	4	🟦						
				🟦	5	🟦					
					🟦	6	🟦				
						🟦	7	🟦			
							🟦	8	🟦		
								🟦	9	🟦	
									🟦	10	🟦
										🟦	11

0
1
2
3
4
5
6
7
8
9
10
11

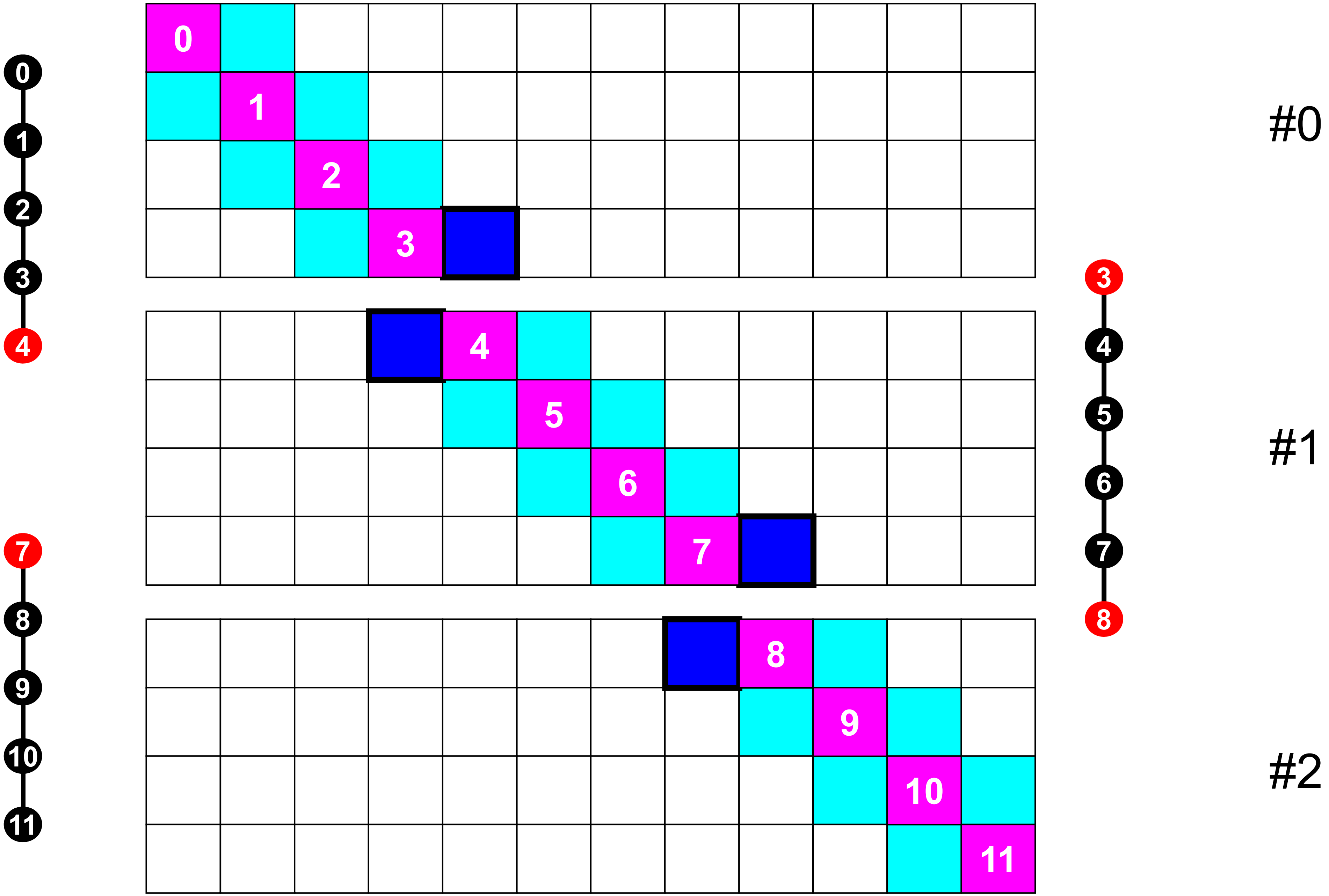
# “Internal Nodes” should be balanced



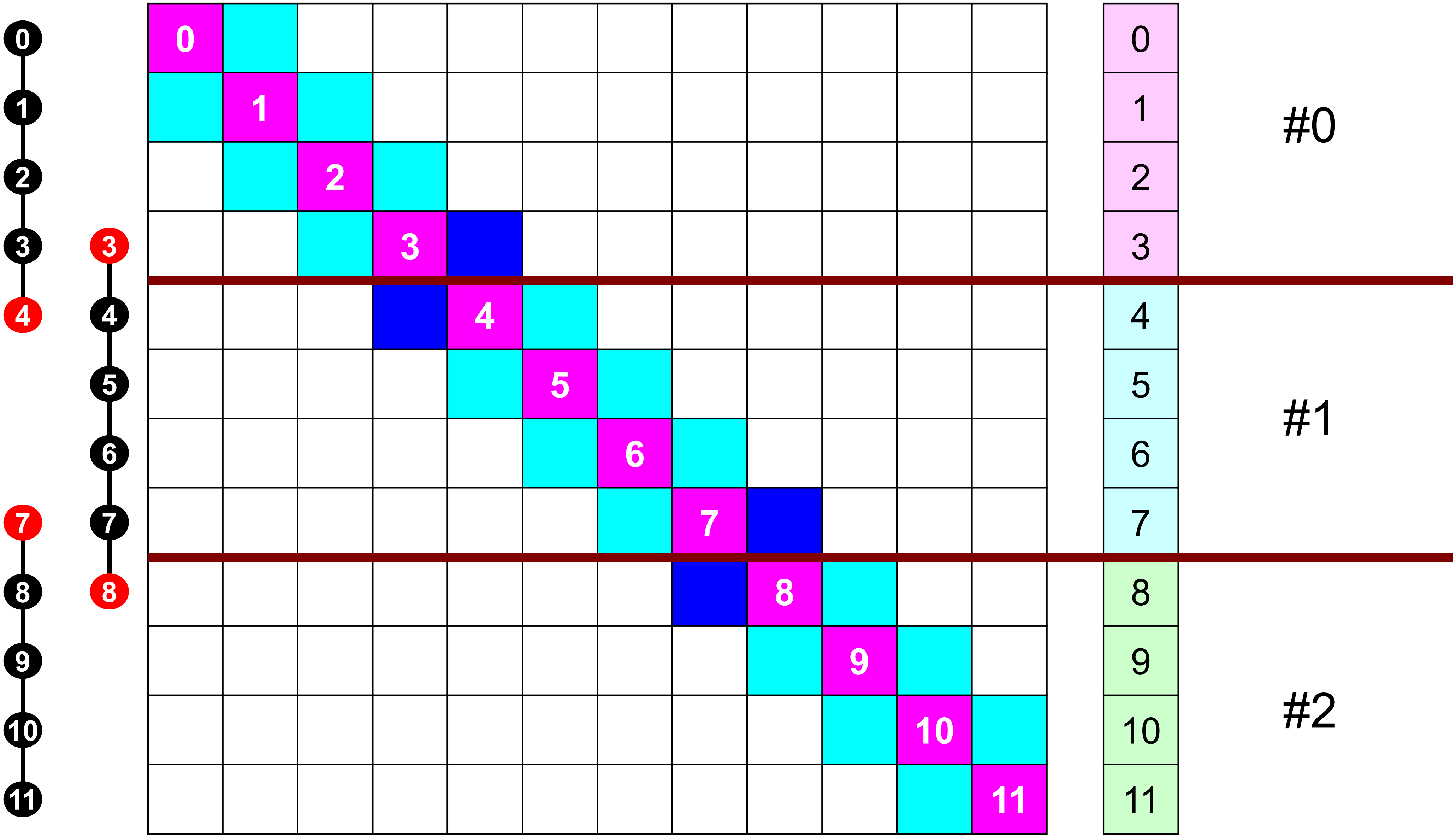
Matrices are incomplete !



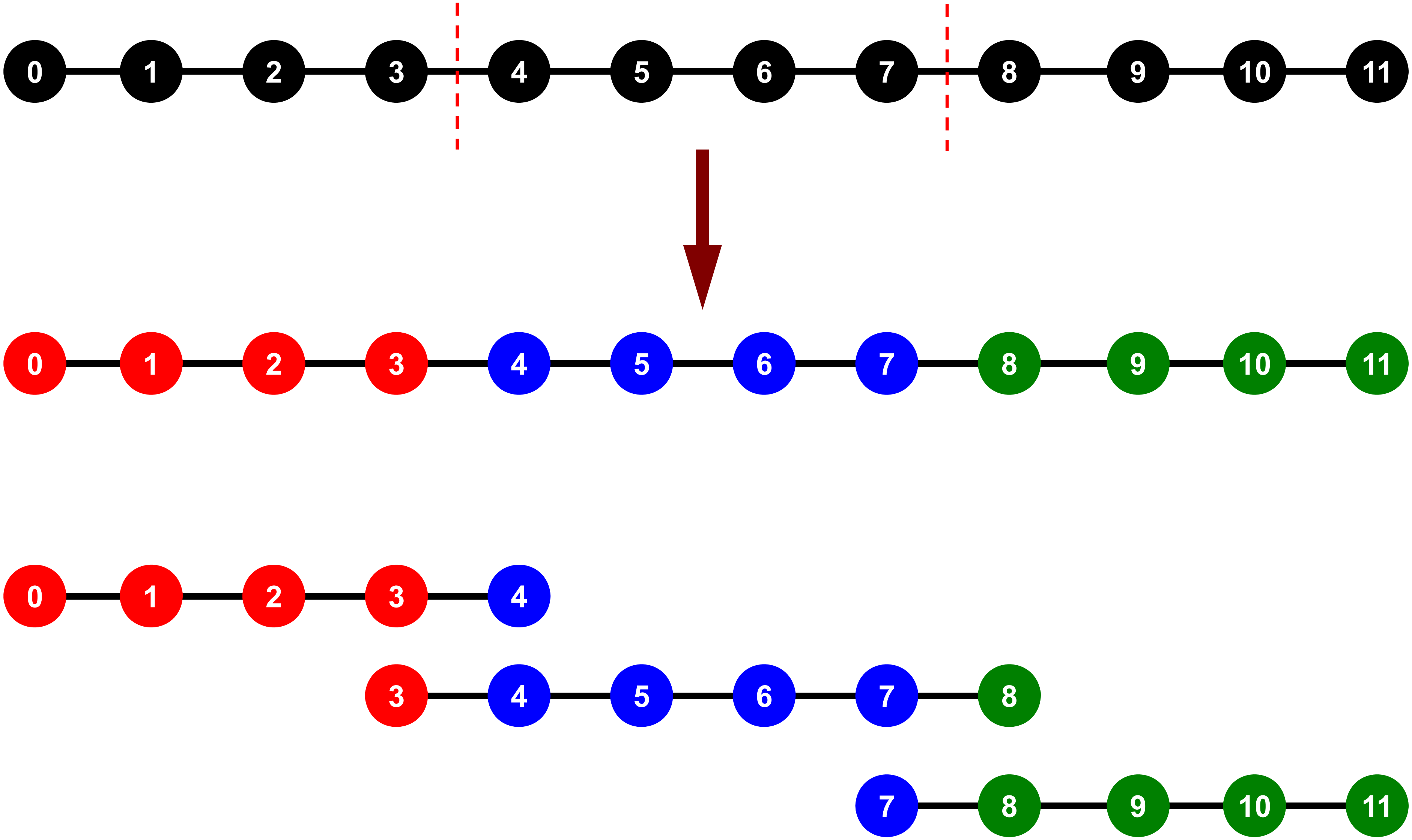
Connected Cell’s + External Nodes



1D FVM: 12 meshes/3 domains

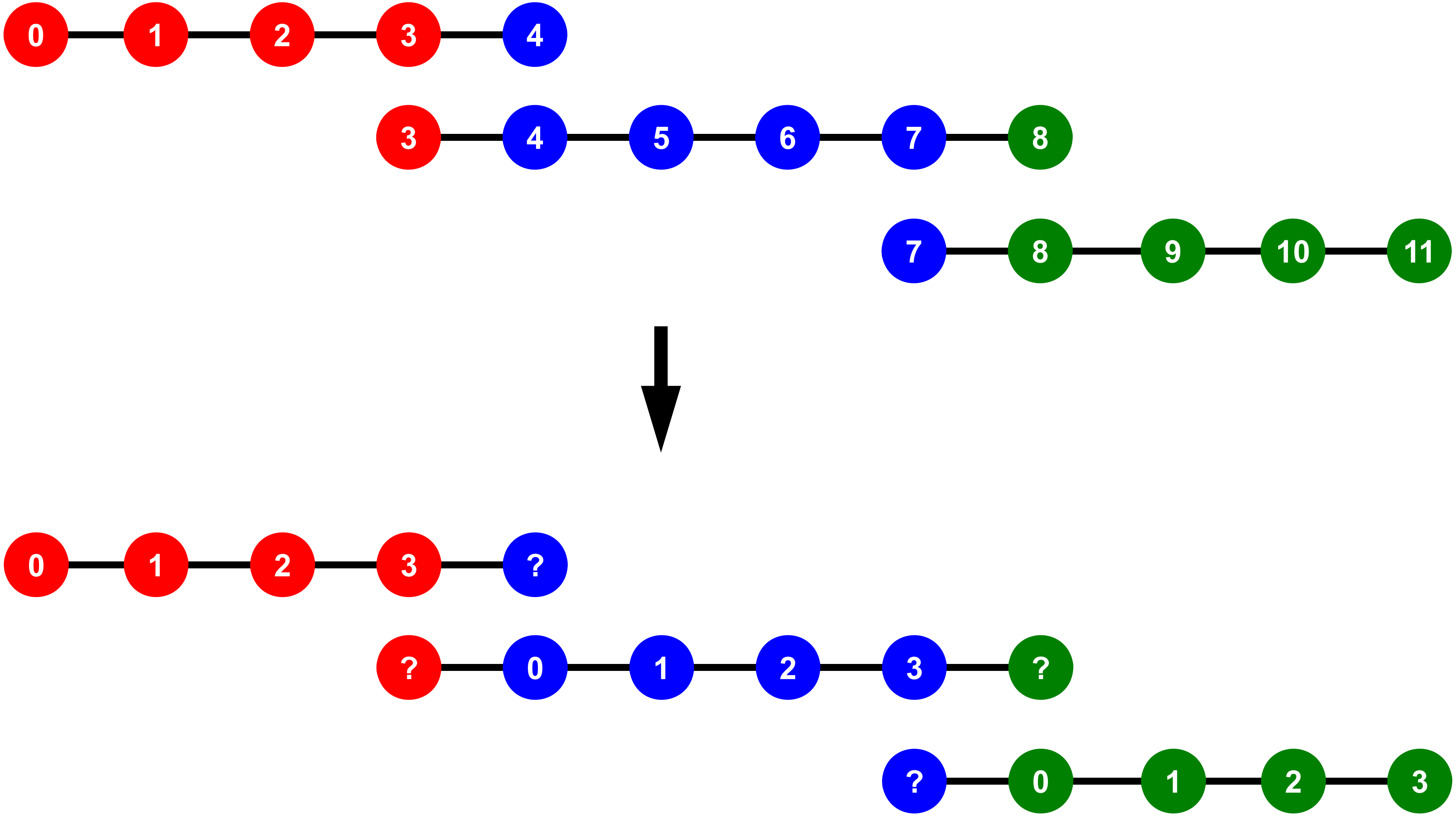


1D FVM: 12 meshes/3 domains



Local Numbering for SPMD

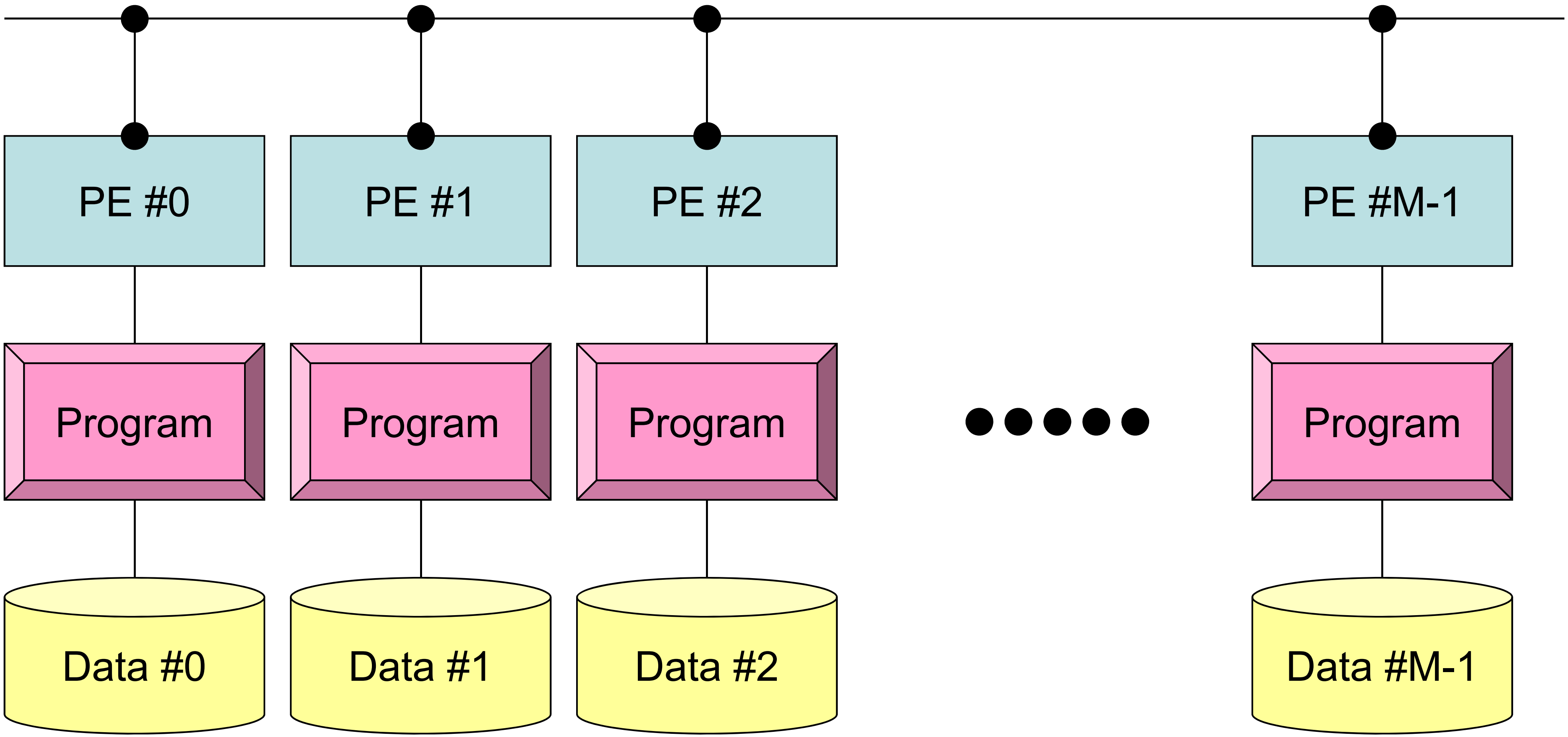
Numbering of internal nodes is 1-N (0-N-1), same operations in serial program can be applied. How about numbering of external nodes ?



PE: Processing Element
Processor, Domain, Process

SPMD

```
mpirun -np M <Program>
```



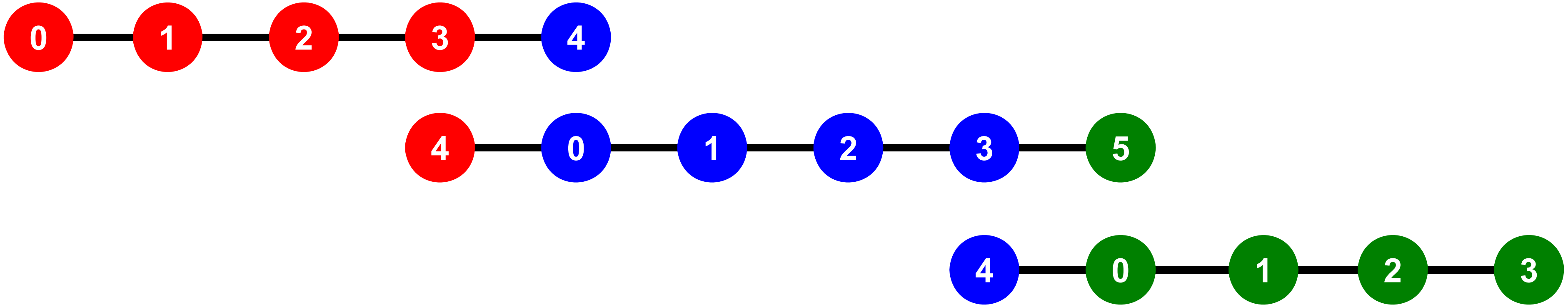
Each process does same operation for different data

Large-scale data is decomposed, and each part is computed by each process

It is ideal that parallel program is not different from serial one except communication.

Local Numbering for SPMD

Numbering of external nodes: N+1, N+2 (N,N+1)



Preconditioned CG Solver

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

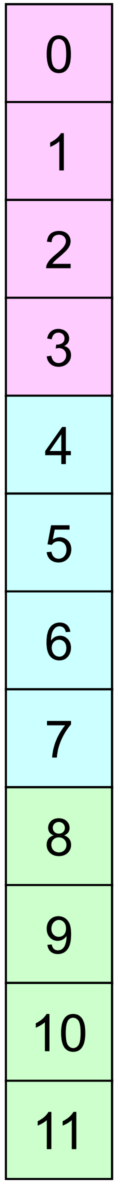
$$[M]=\begin{bmatrix} D_1 & 0 & \ldots & 0 & 0 \\ 0 & D_2 & & 0 & 0 \\ \ldots & & \ldots & & \ldots \\ 0 & 0 & & D_{N-1} & 0 \\ 0 & 0 & \ldots & 0 & D_N \end{bmatrix}$$

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];
    }
```

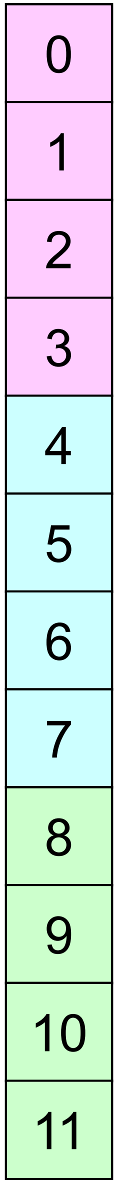
```
/*
//-- {x}= {x} + ALPHA*{p}       DAXPY: double a{x} plus {y}
//    {r}= {r} - ALPHA*{q}
*/
  for(i=0;i<N;i++){
      U[i]    += Alpha * W[P][i];
      W[R][i] -= Alpha * W[Q][i];
  }
```



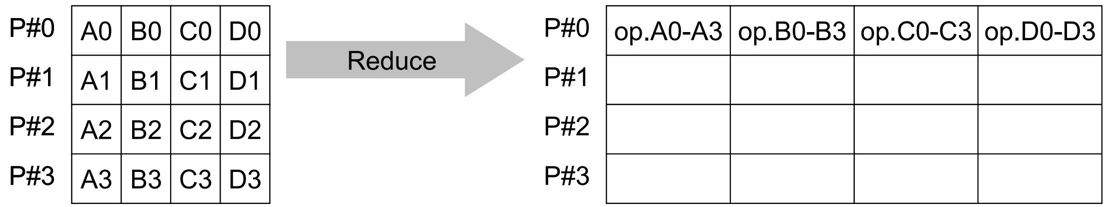
Dot Products

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;
```



MPI_Reduce



P#0	A0	B0	C0	D0
P#1	A1	B1	C1	D1
P#2	A2	B2	C2	D2
P#3	A3	B3	C3	D3

Reduce

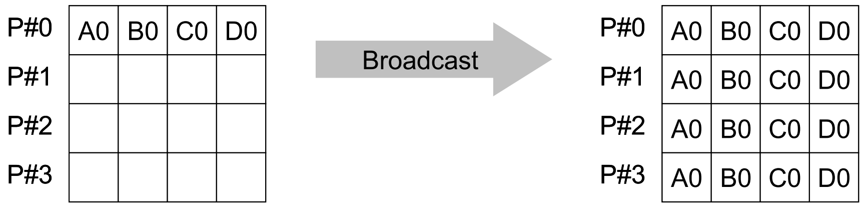
P#0	op.A0-A3	op.B0-B3	op.C0-C3	op.D0-D3
P#1				
P#2				
P#3				

- Reduces values on all processes to a single value
  - Summation, Product, Max, Min etc.

- **`MPI_Reduce (sendbuf,recvbuf,count,datatype,op,root,comm)`**
  - **sendbuf** choice I starting address of send buffer
  - **recvbuf** choice O starting address receive buffer
    type is defined by "**datatype**"
  - **count** int I number of elements in send/receive buffer
  - **datatype** MPI_Datatype I data type of elements of send/recive buffer
    FORTRAN MPI_INTEGER, MPI_REAL, MPI_DOUBLE_PRECISION, MPI_CHARACTER etc.
    C MPI_INT, MPI_FLOAT, MPI_DOUBLE, MPI_CHAR etc
  - **op** MPI_Op I reduce operation
    MPI_MAX, MPI_MIN, MPI_SUM, MPI_PROD, MPI_LAND, MPI_BAND etc
    Users can define operations by **MPI_OP_CREATE**
  - **root** int I rank of root process
  - **comm** MPI_Comm I communicator

C

MPI_Bcast



P#0	A0	B0	C0	D0
P#1				
P#2				
P#3				

Broadcast

P#0	A0	B0	C0	D0
P#1	A0	B0	C0	D0
P#2	A0	B0	C0	D0
P#3	A0	B0	C0	D0

- Broadcasts a message from the process with rank "root" to all other processes of the communicator

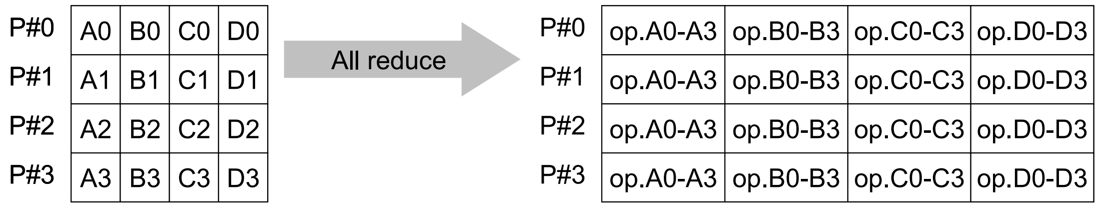
- **`MPI_Bcast (buffer,count,datatype,root,comm)`**
  - **buffer** choice `I/O` starting address of buffer
    type is defined by ”**datatype**”
  - **count** int `I` number of elements in send/recv buffer
  - **datatype** MPI_Datatype `I` data type of elements of send/recv buffer

```
FORTRAN  MPI_INTEGER, MPI_REAL, MPI_DOUBLE_PRECISION, MPI_CHARACTER etc.
C        MPI_INT, MPI_FLOAT, MPI_DOUBLE, MPI_CHAR etc.
```

  - **root** int `I` rank of root process
  - **comm** MPI_Comm `I` communicator

C

MPI_Allreduce



P#0	A0	B0	C0	D0
P#1	A1	B1	C1	D1
P#2	A2	B2	C2	D2
P#3	A3	B3	C3	D3

P#0	op.A0-A3	op.B0-B3	op.C0-C3	op.D0-D3
P#1	op.A0-A3	op.B0-B3	op.C0-C3	op.D0-D3
P#2	op.A0-A3	op.B0-B3	op.C0-C3	op.D0-D3
P#3	op.A0-A3	op.B0-B3	op.C0-C3	op.D0-D3

- MPI_Reduce + MPI_Bcast
- Summation (of dot products) and MAX/MIN values are likely to utilized in each process

- **`call MPI_Allreduce (sendbuf,recvbuf,count,datatype,op, comm)`**

– **sendbuf**	choice	I	starting address of send buffer
– **recvbuf**	choice	O	starting address receive buffer type is defined by "**datatype**"
– **count**	int	I	number of elements in send/recv buffer
– **datatype**	MPI_Datatype	I	data type of elements of send/recv buffer
– **op**	MPI_Op	I	reduce operation
– **comm**	MPI_Comm	I	communicator

C

“op” of MPI_Reduce/Allreduce

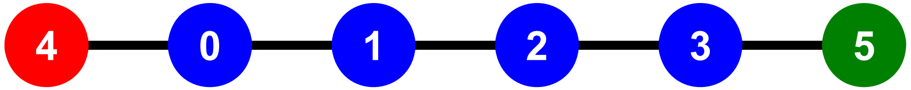
```
MPI_Reduce
(sendbuf,recvbuf,count,datatype,op,root,comm)
```

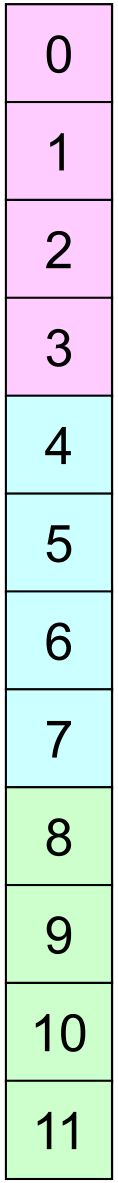
MPI_MAX, **MPI_MIN**	Max, Min
MPI_SUM, **MPI_PROD**	Summation, Product
MPI_LAND	Logical AND

Matrix-Vector Products

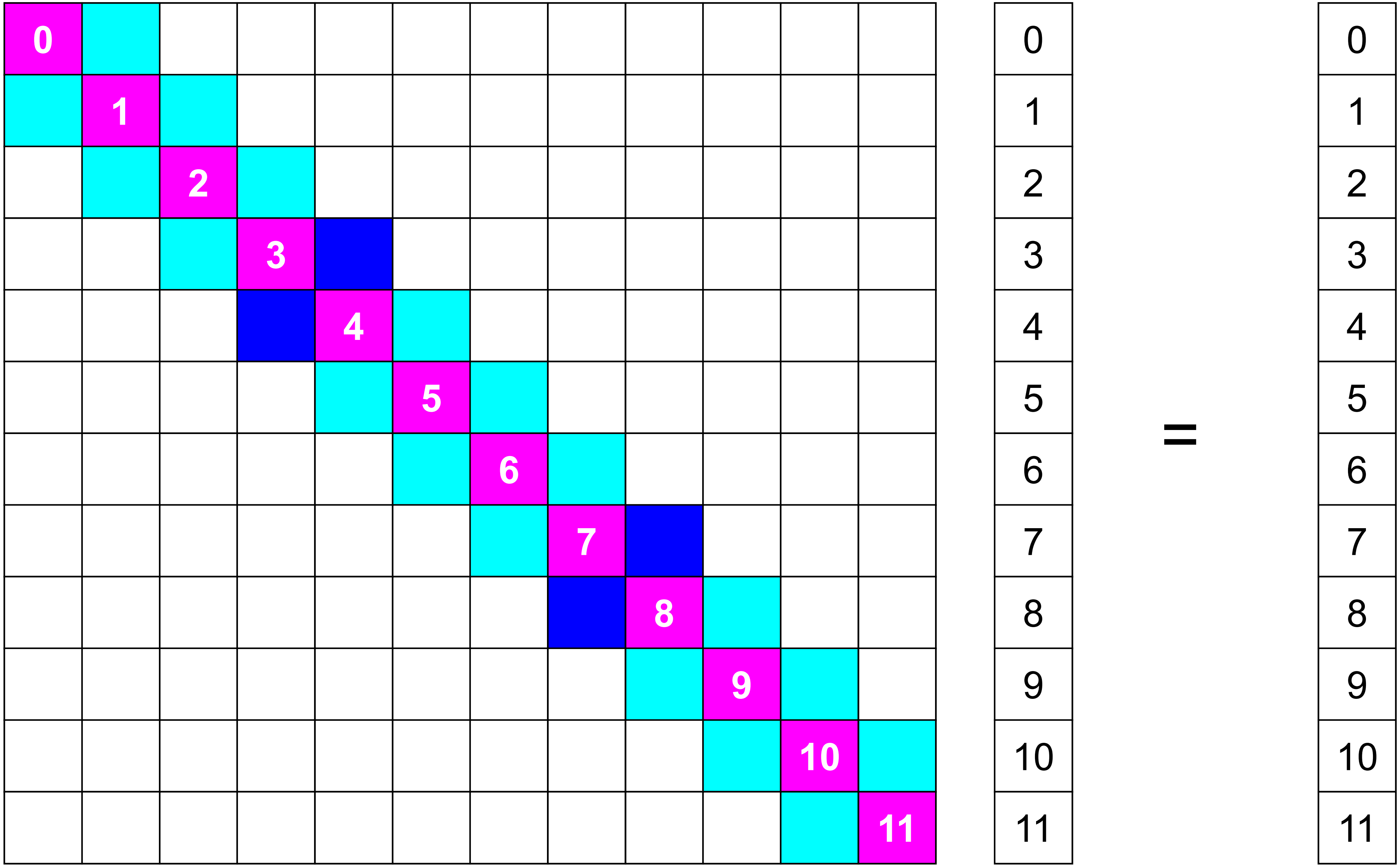
Values at External Points: P-to-P Communication

```
/*
//-- {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]-1];
          }
      }
```

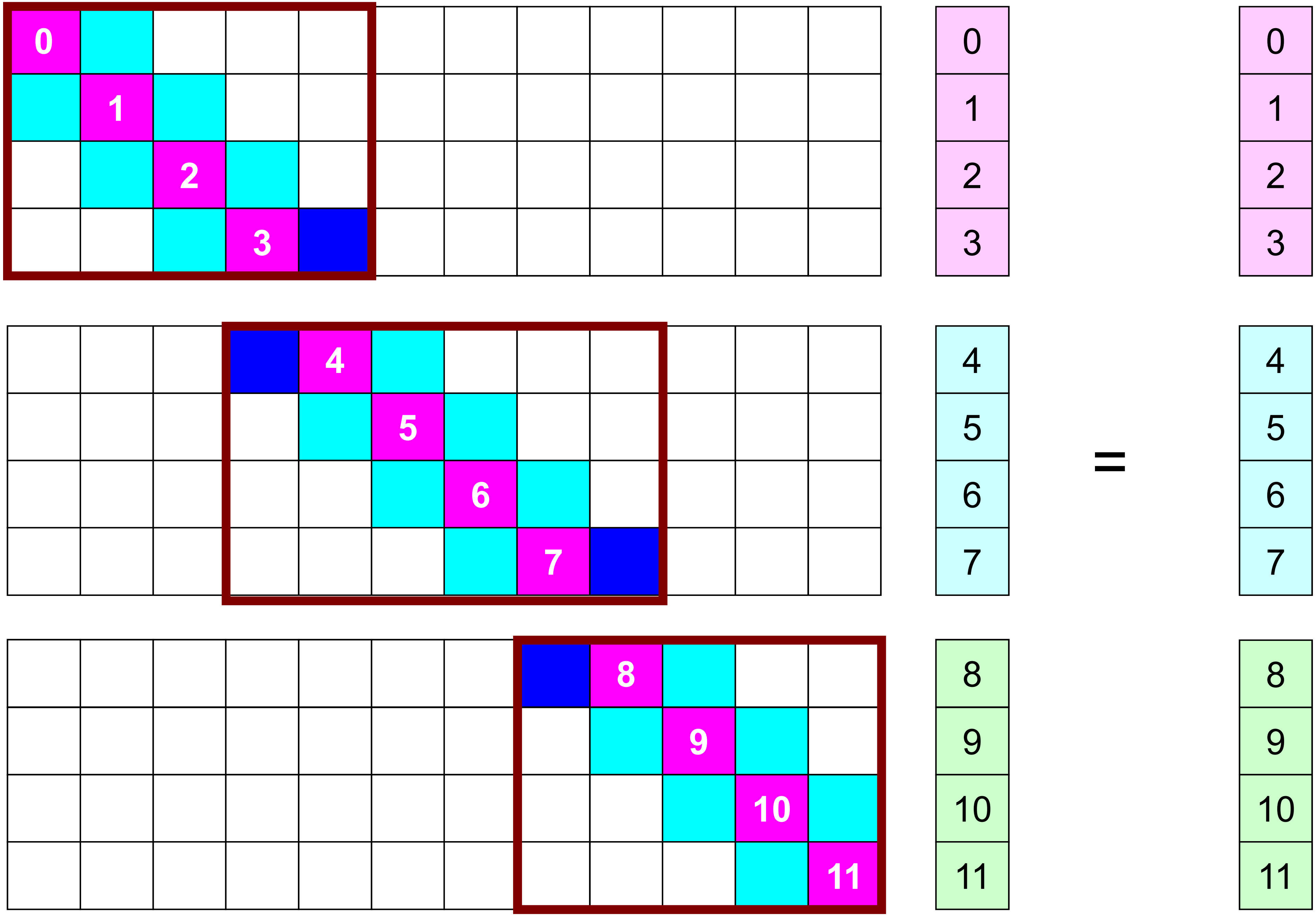




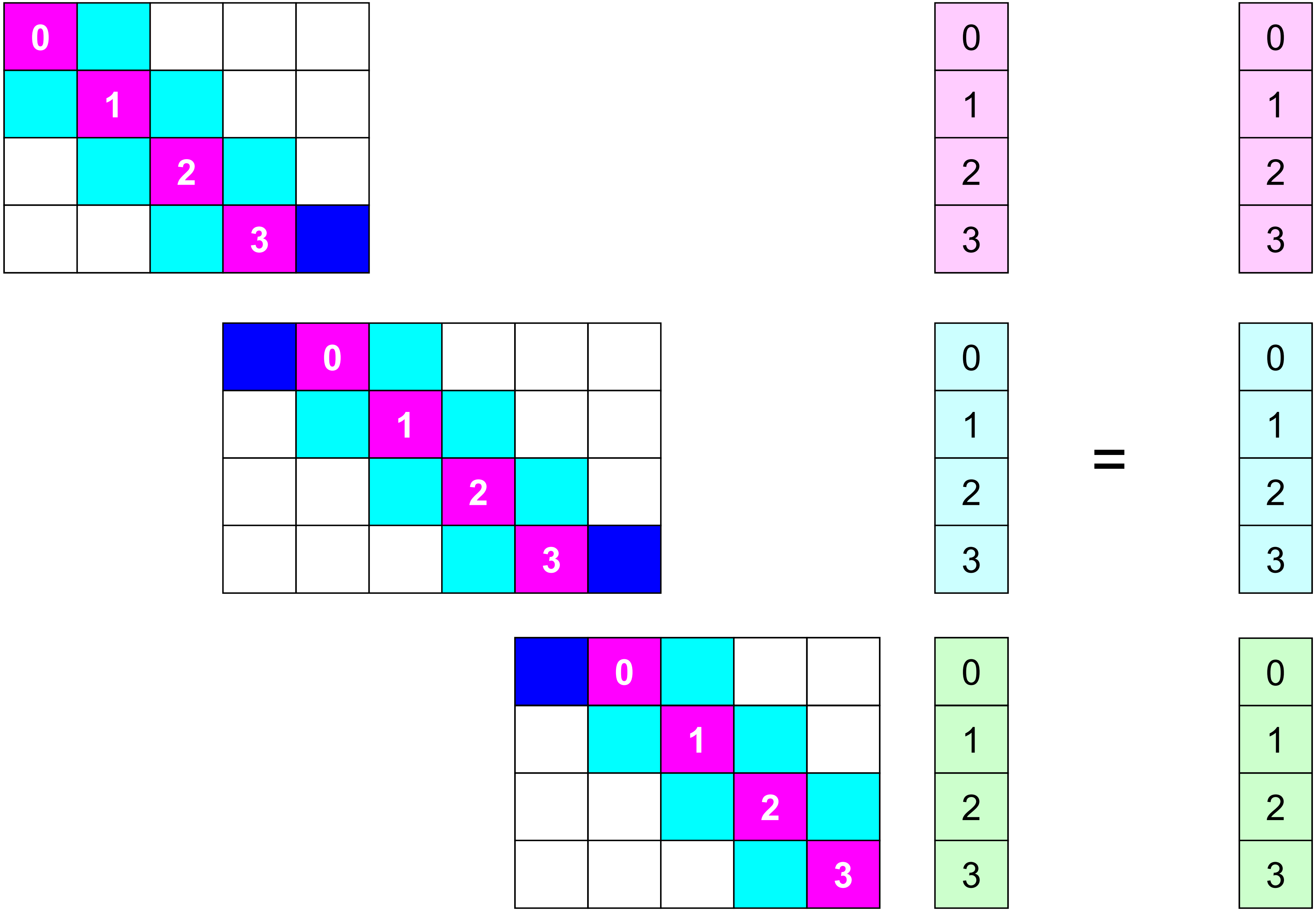
Mat-Vec Products: Local Op. Possible



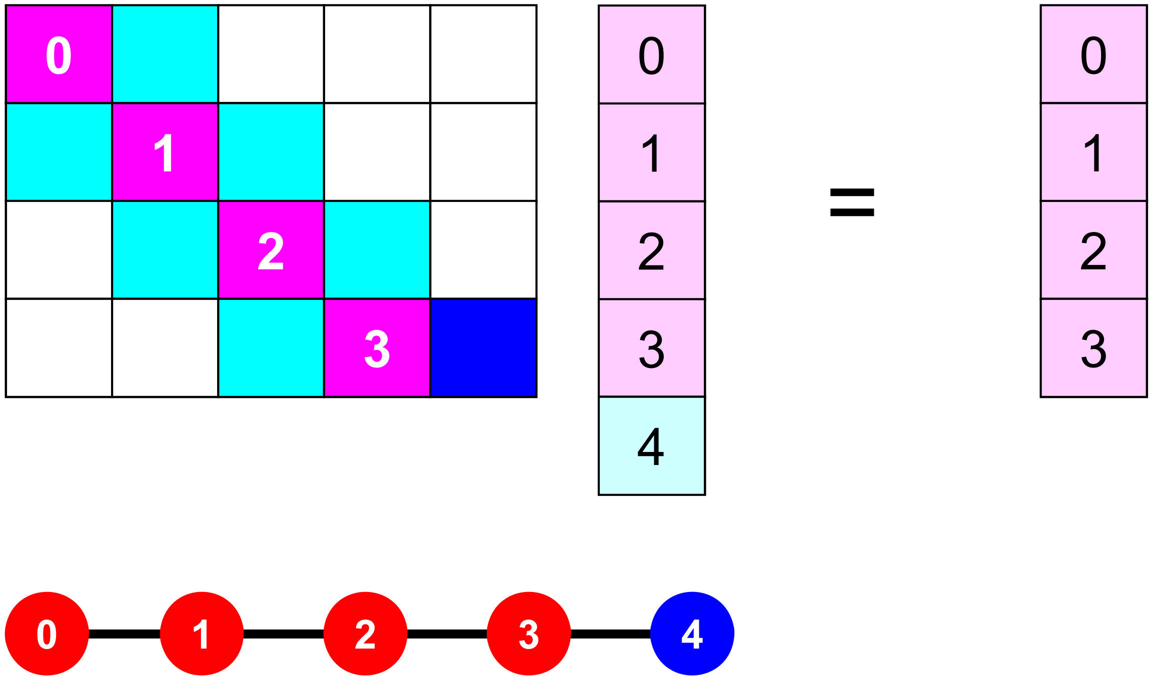
Mat-Vec Products: Local Op. Possible



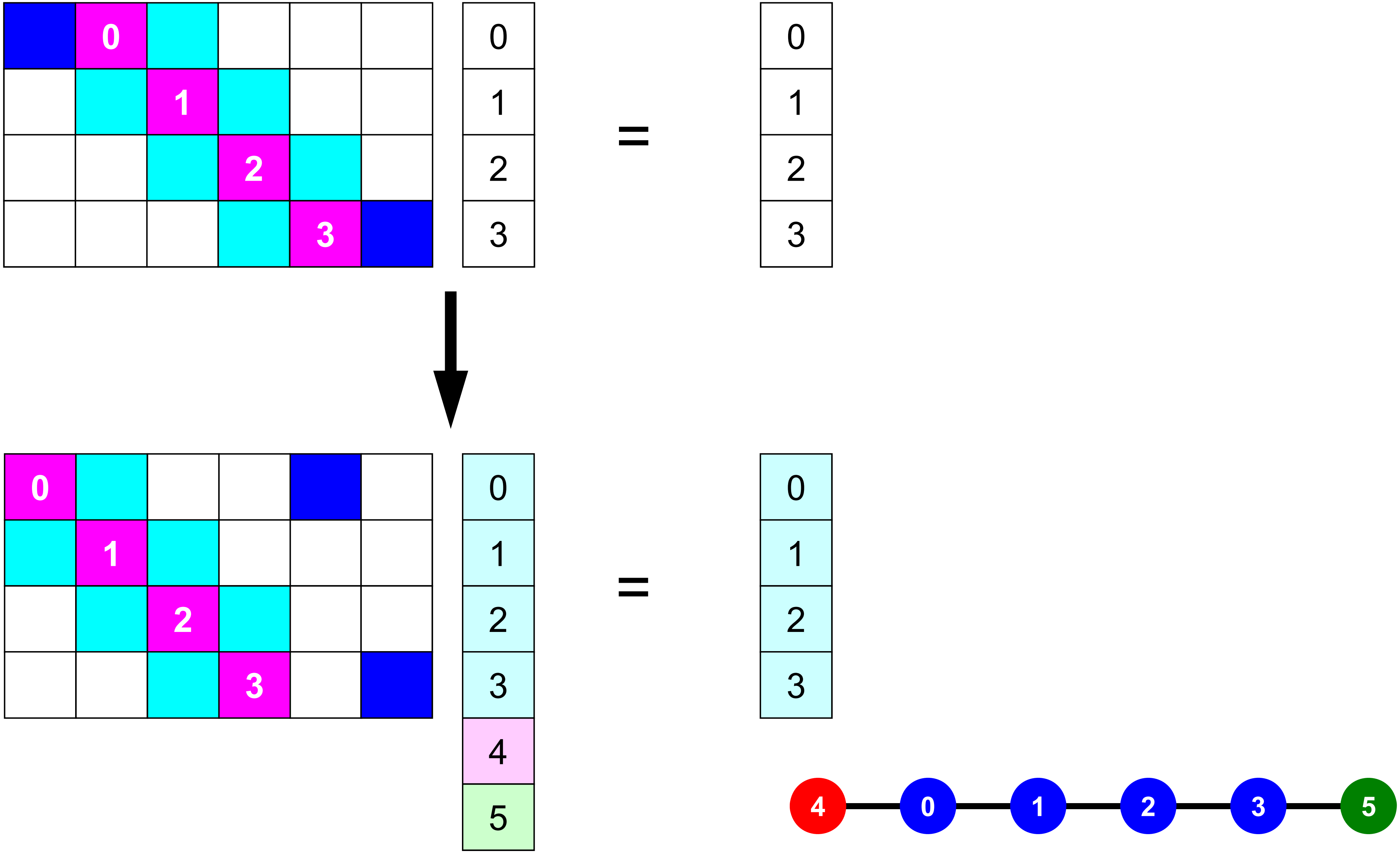
Mat-Vec Products: Local Op. Possible



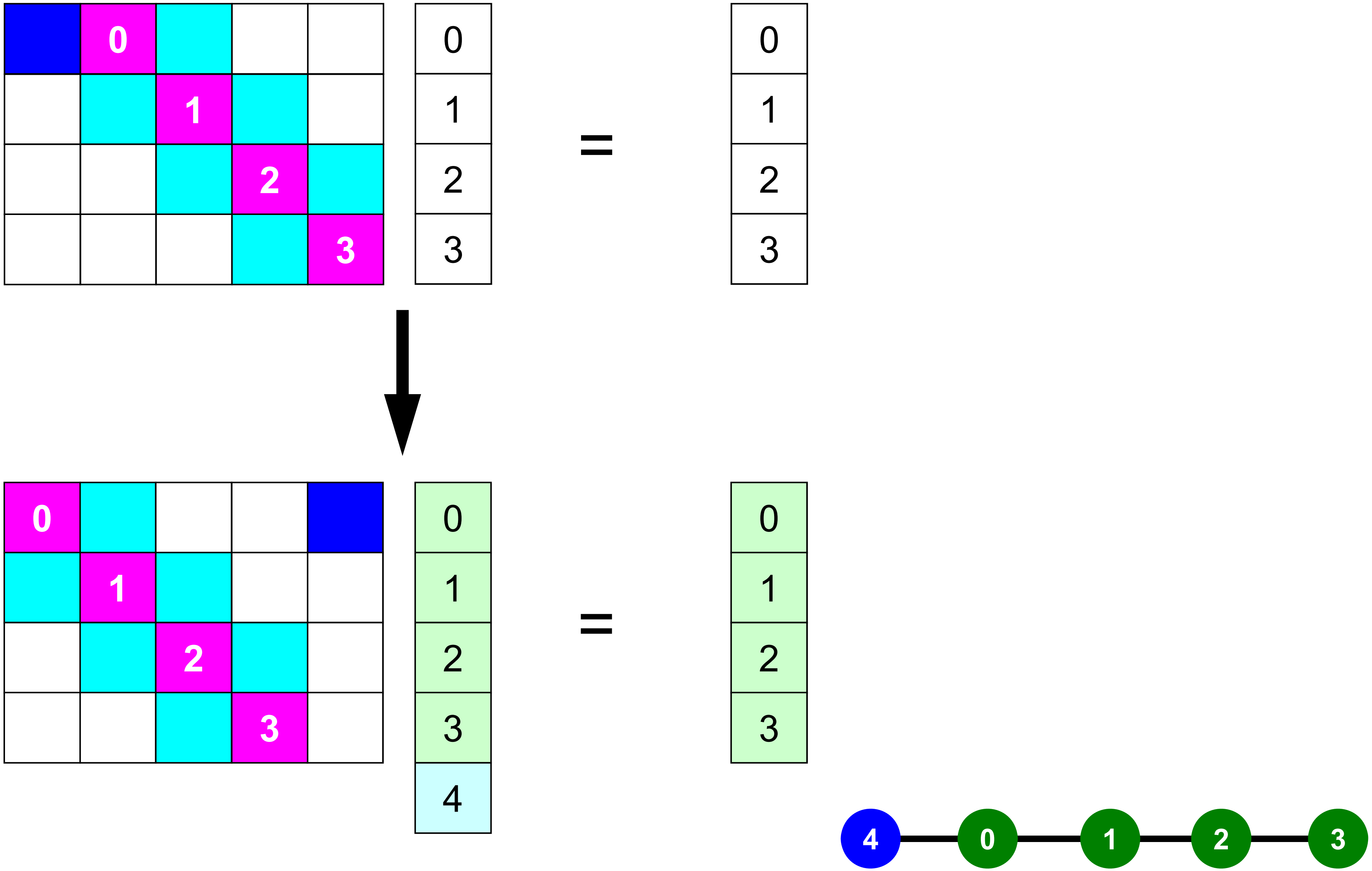
Mat-Vec Products: Local Op. #0



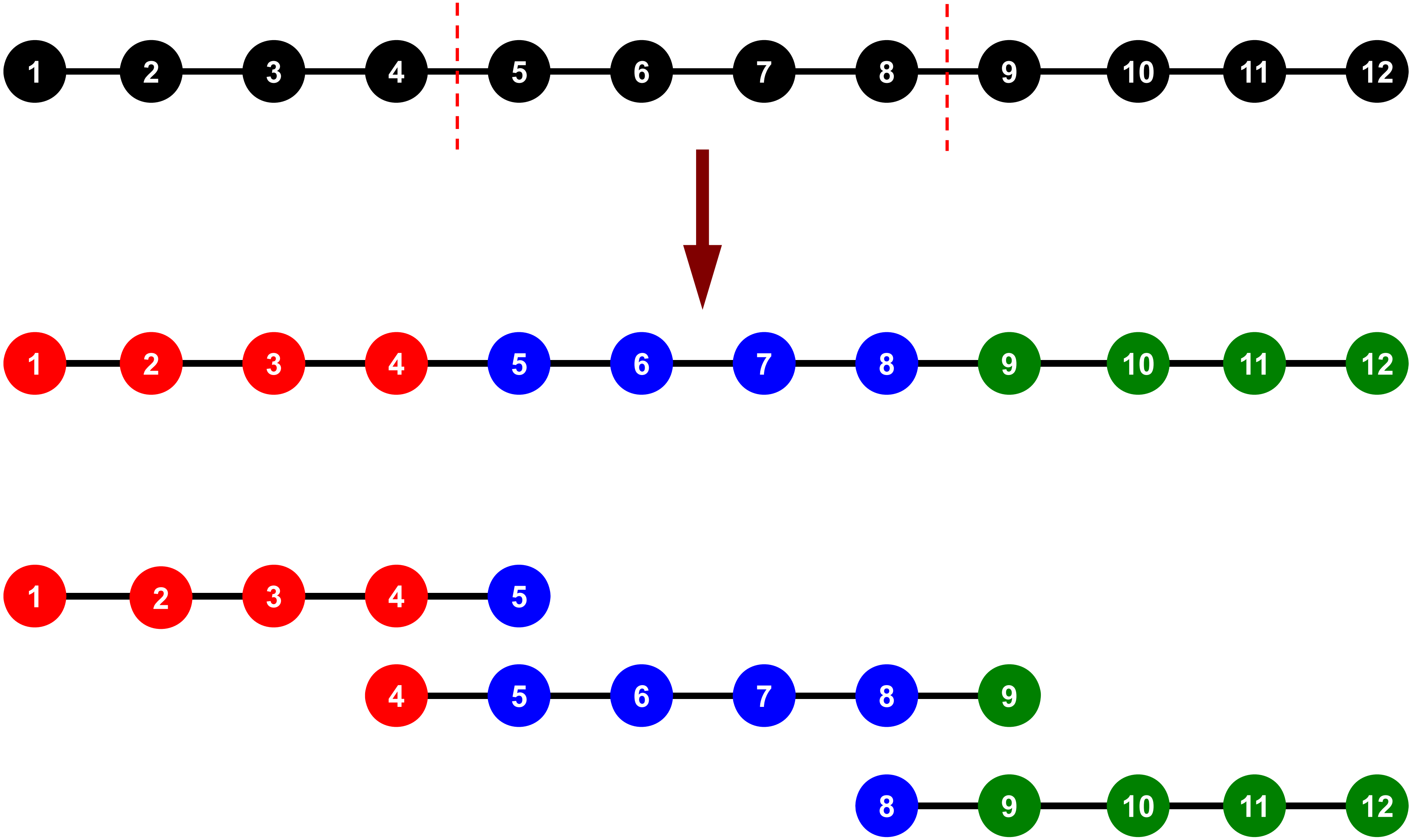
Mat-Vec Products: Local Op. #1



Mat-Vec Products: Local Op. #2

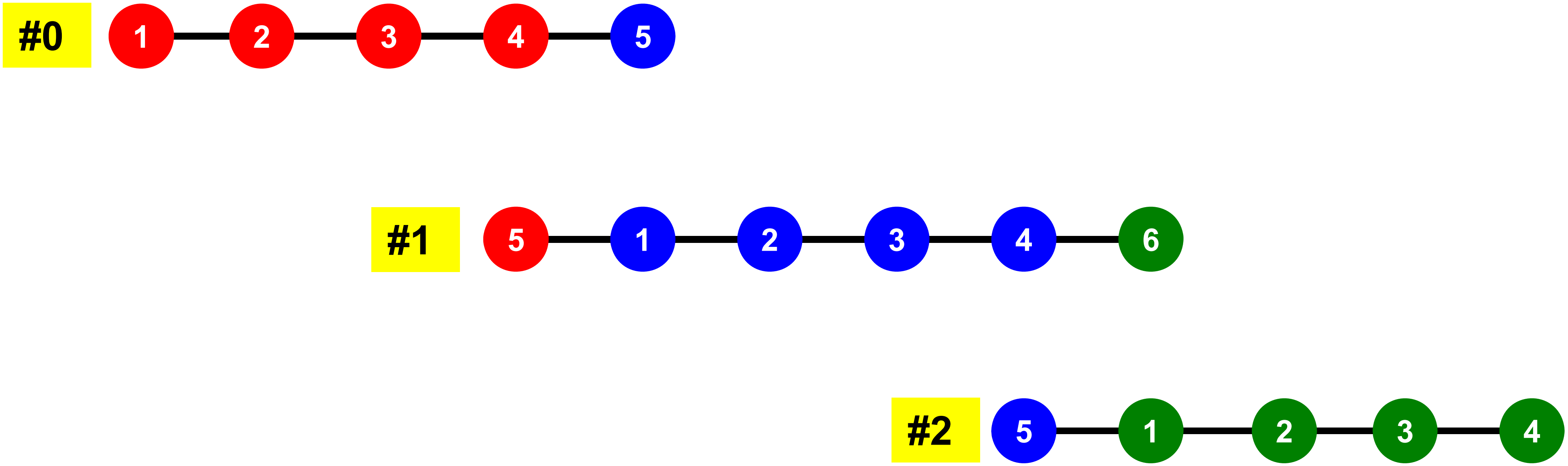


1D FVM: 12 meshes/3 domains



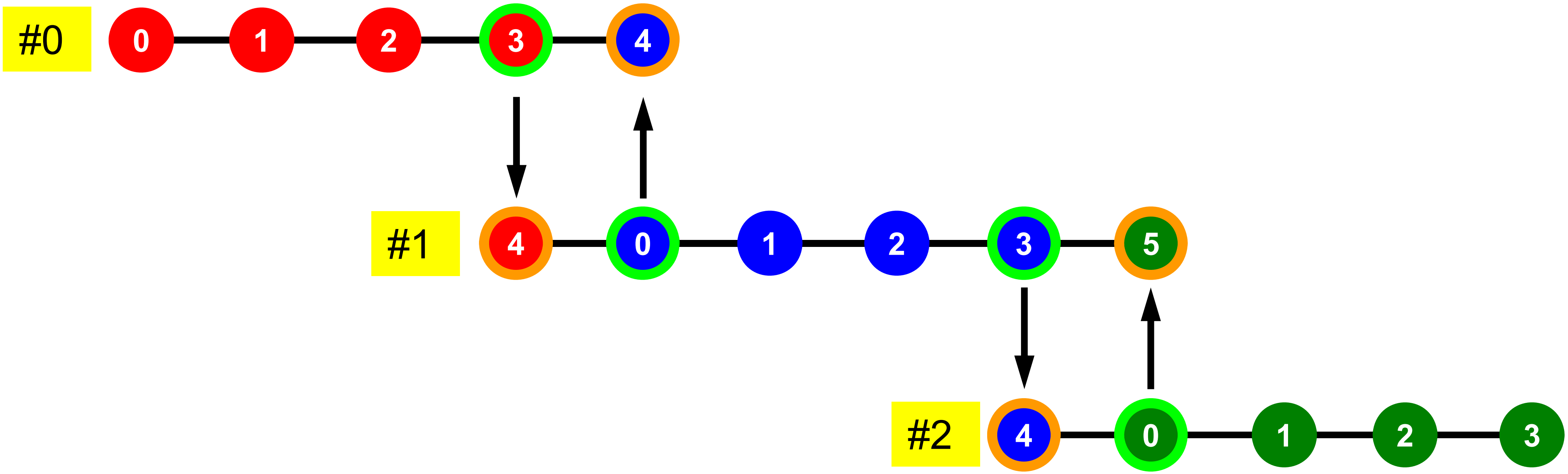
1D FVM: 12 meshes/3 domains

Local ID: Starting from 0 for mesh at each domain



1D FVM: 12 meshes/3 domains

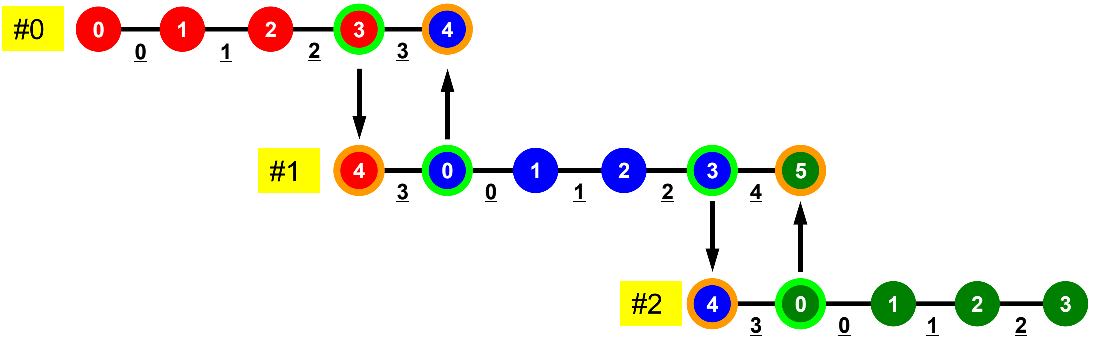
Internal/External Nodes



Collective/Point-to-Poin Communication

- Collective Communication（集団通信）
  - 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（一対一通信）
  - MPI_Send, MPI_Recv
  - Communication with limited processes
    - Neighbors
  - Application Area
    - FEM, FDM: Localized Method

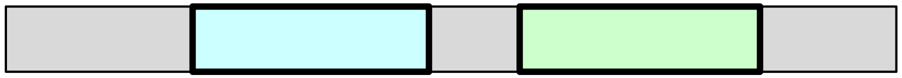


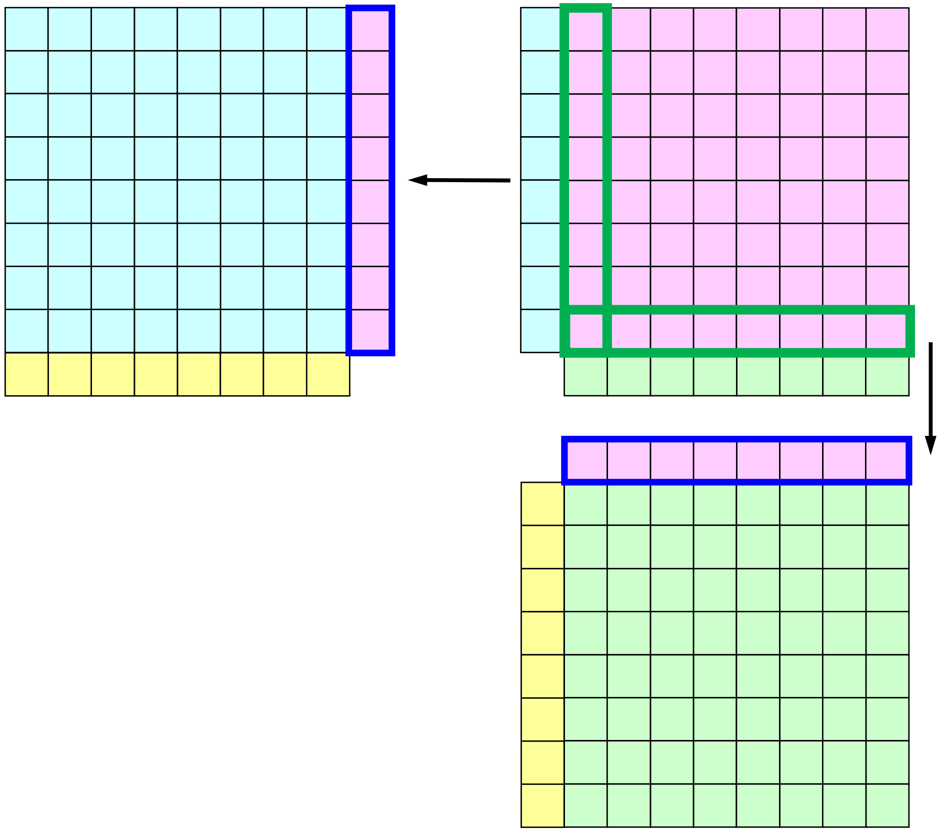
SEND: sending from boundary nodes

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





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)`**

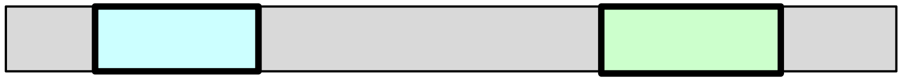
`sendbuf`	choice	I	starting address of sending buffer
`count`	int	I	number of elements in sending buffer
`datatype`	MPI_Datatype	I	datatype of each sending buffer element
`dest`	int	I	rank of destination
`tag`	int	I	message tag This integer can be used by the application to distinguish messages. Communication 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`

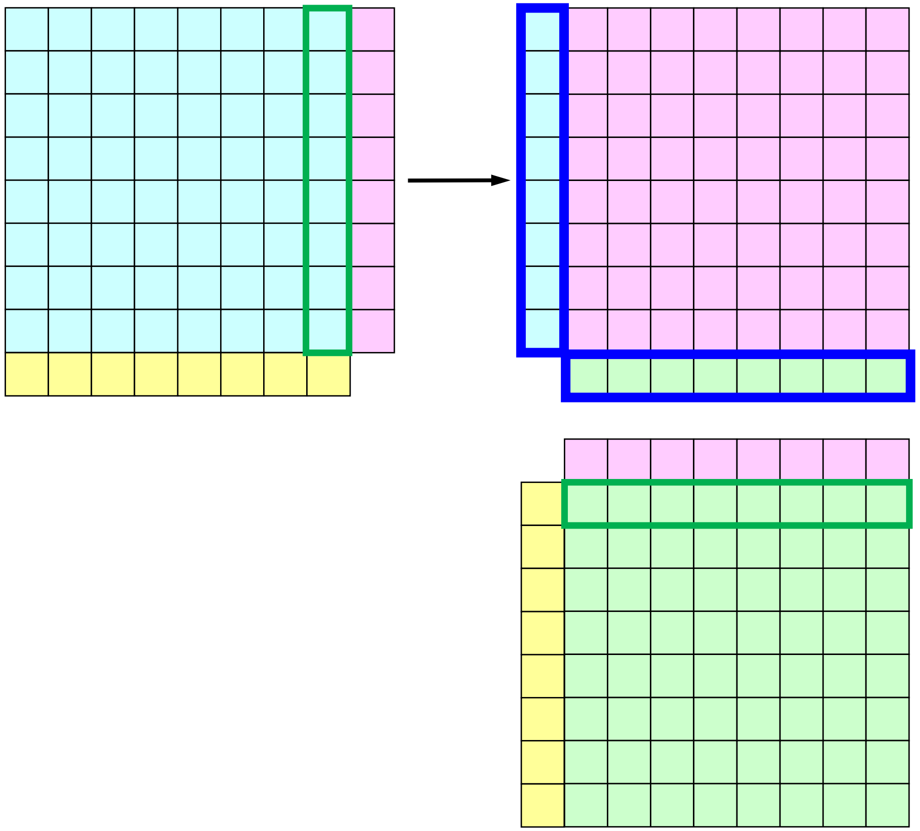
RECV: receiving to external nodes

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





MPI_Irecv

C

- 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'

Distributed Local Data Structure for Parallel Computation

- Distributed local data structure for domain-to-doain 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

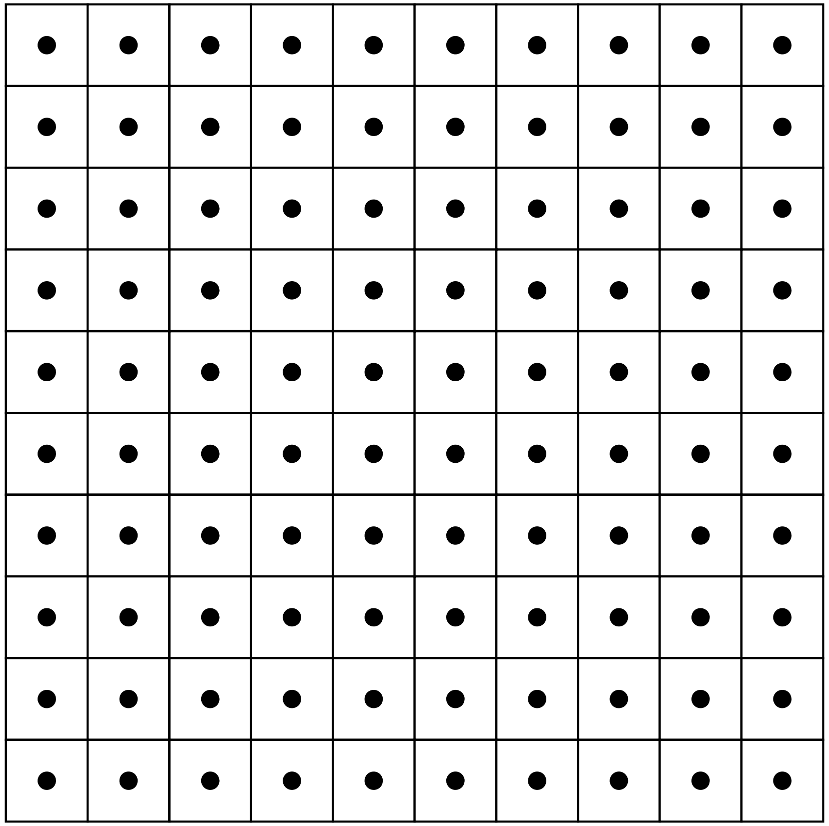
- Everything is easy, if proper data structure is defined:
  - Values at boundary pts are copied into sending buffers
  - Send/Recv
  - Values at external pts are updated through receiving buffers

- Introduction
- Quick Overview of MPI

- **Local Data Structure & Communication**
  - 1D
  - **2D**

2D FDM (1/5)

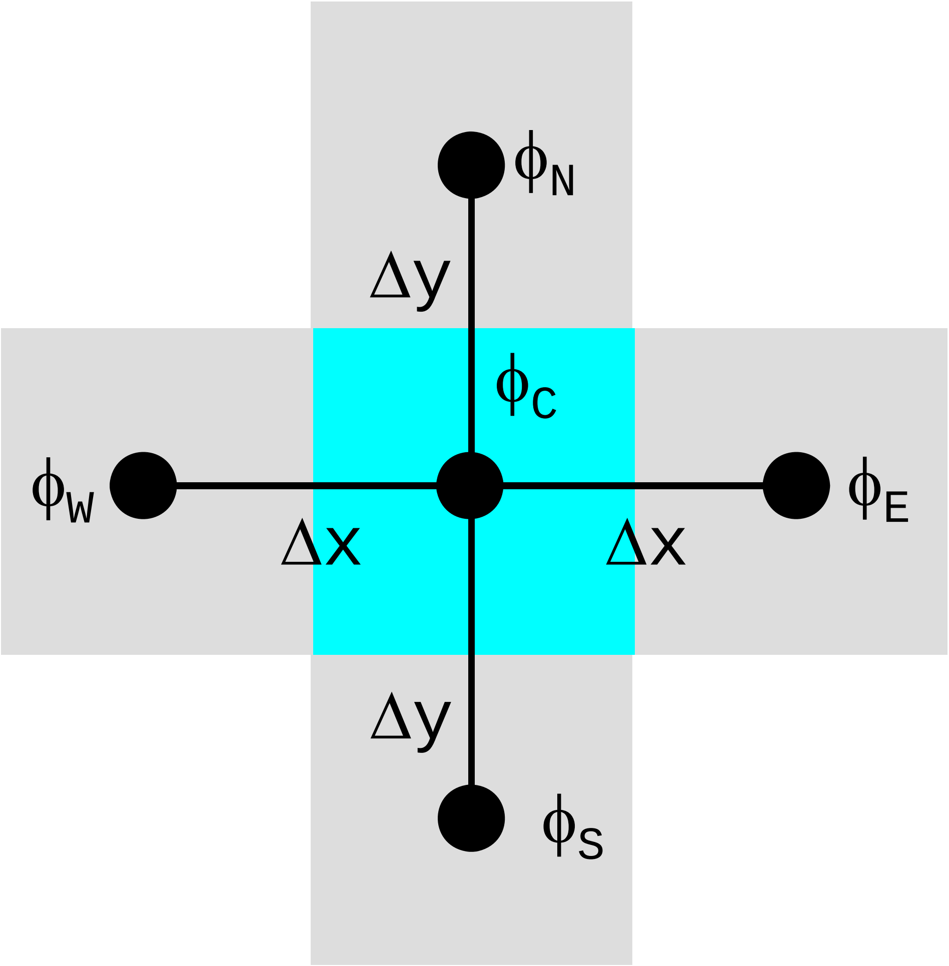
Entire Mesh



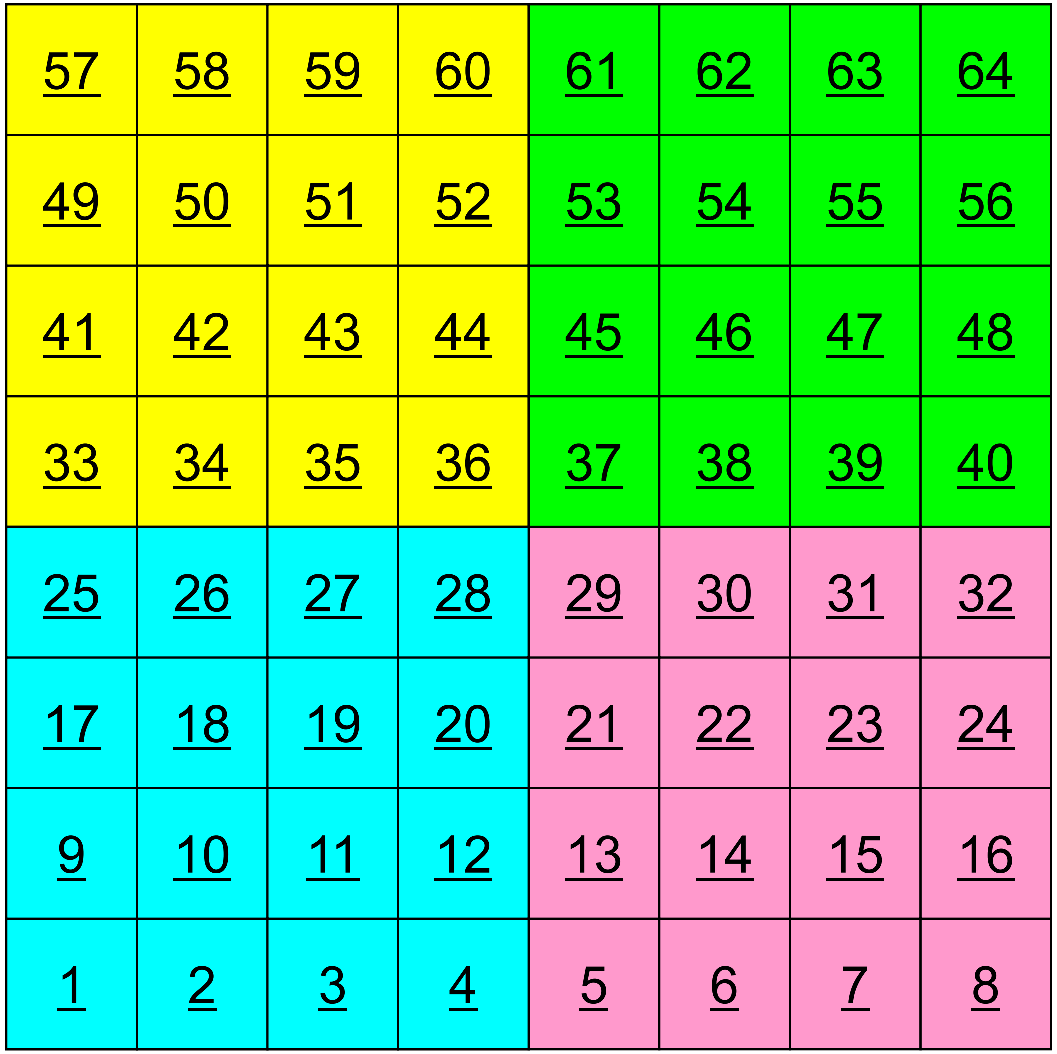
2D FDM (5-point, central difference)

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

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

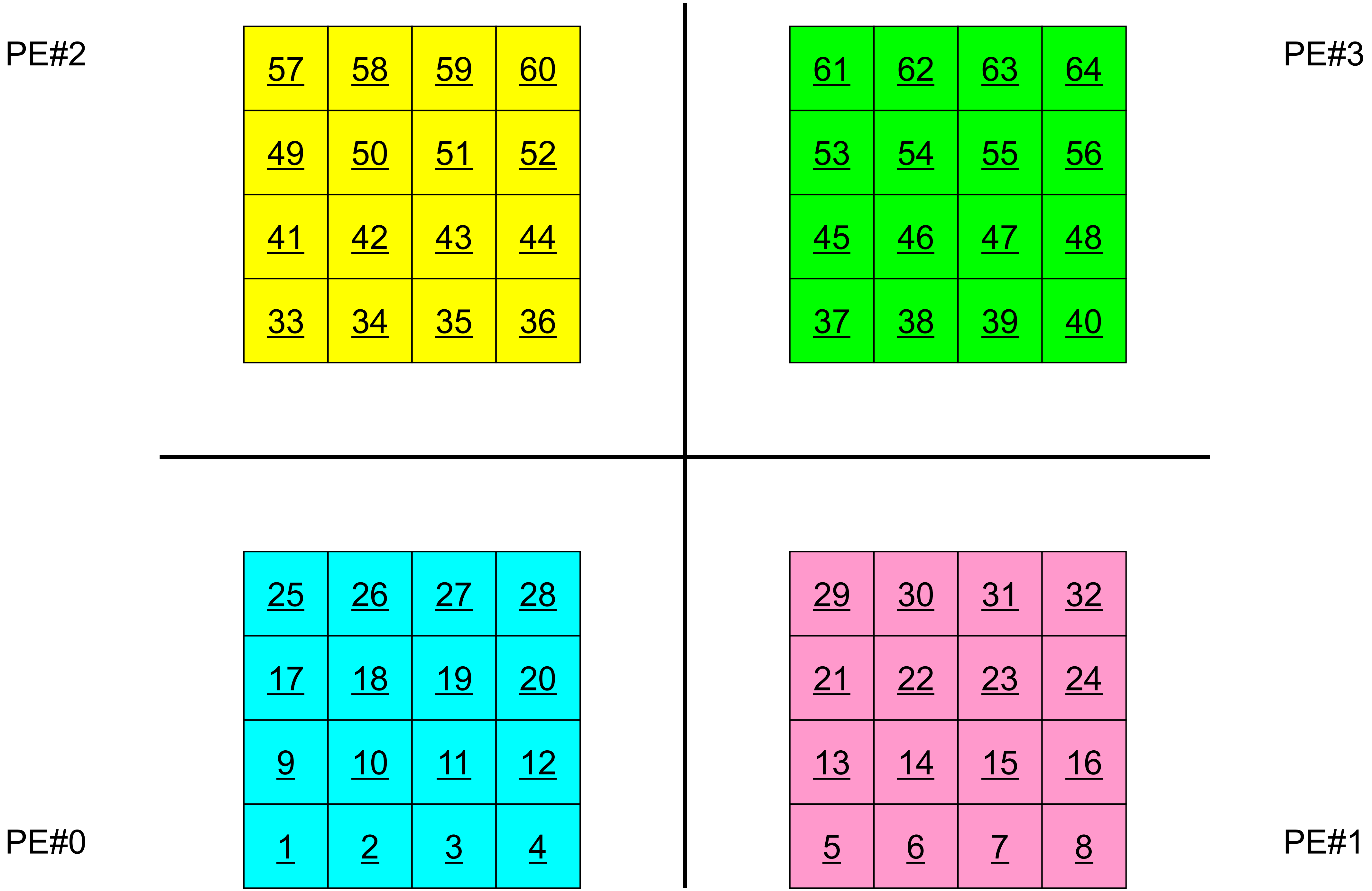


Decompose into 4 domains



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



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

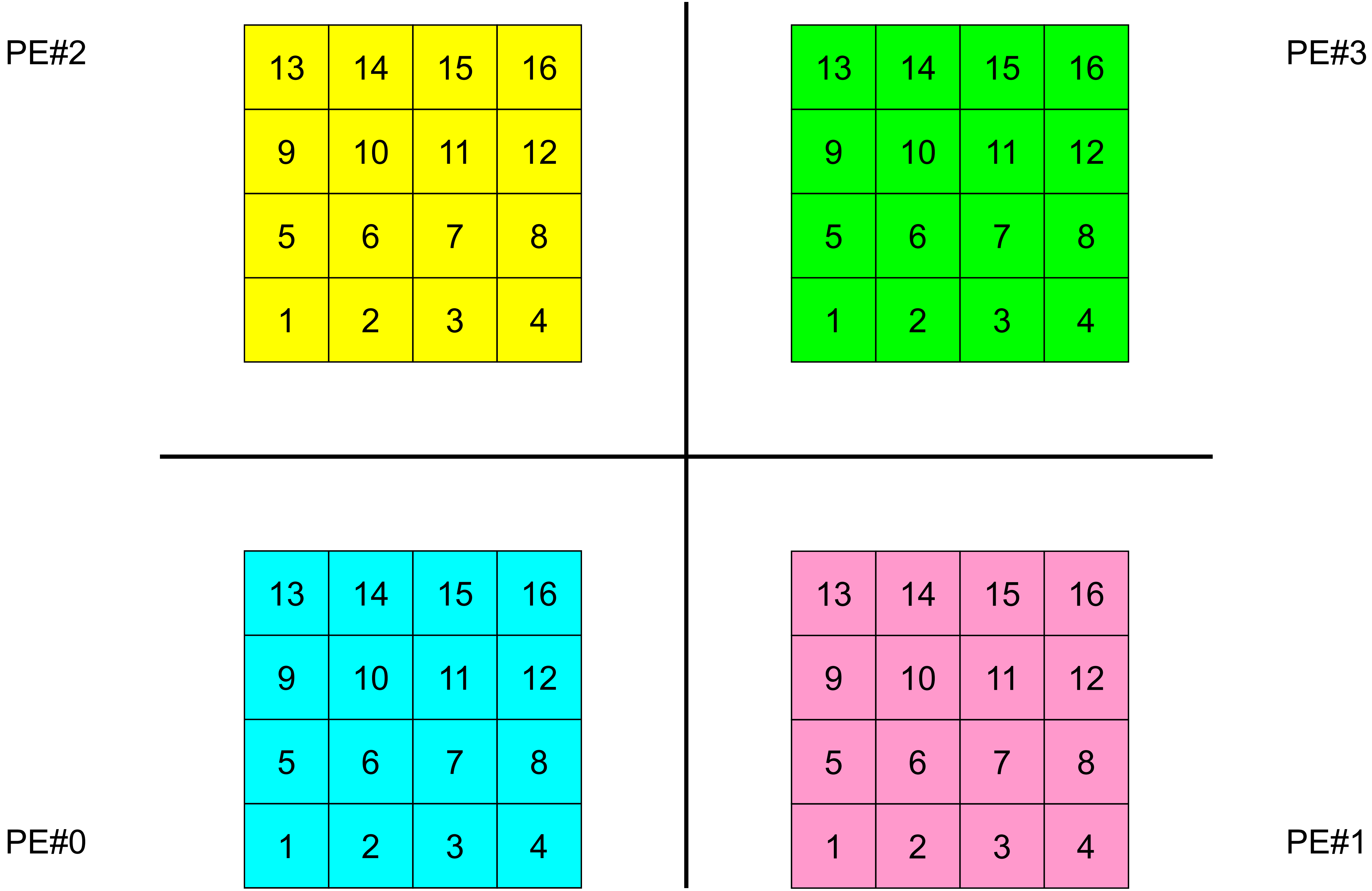
PE#0

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



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

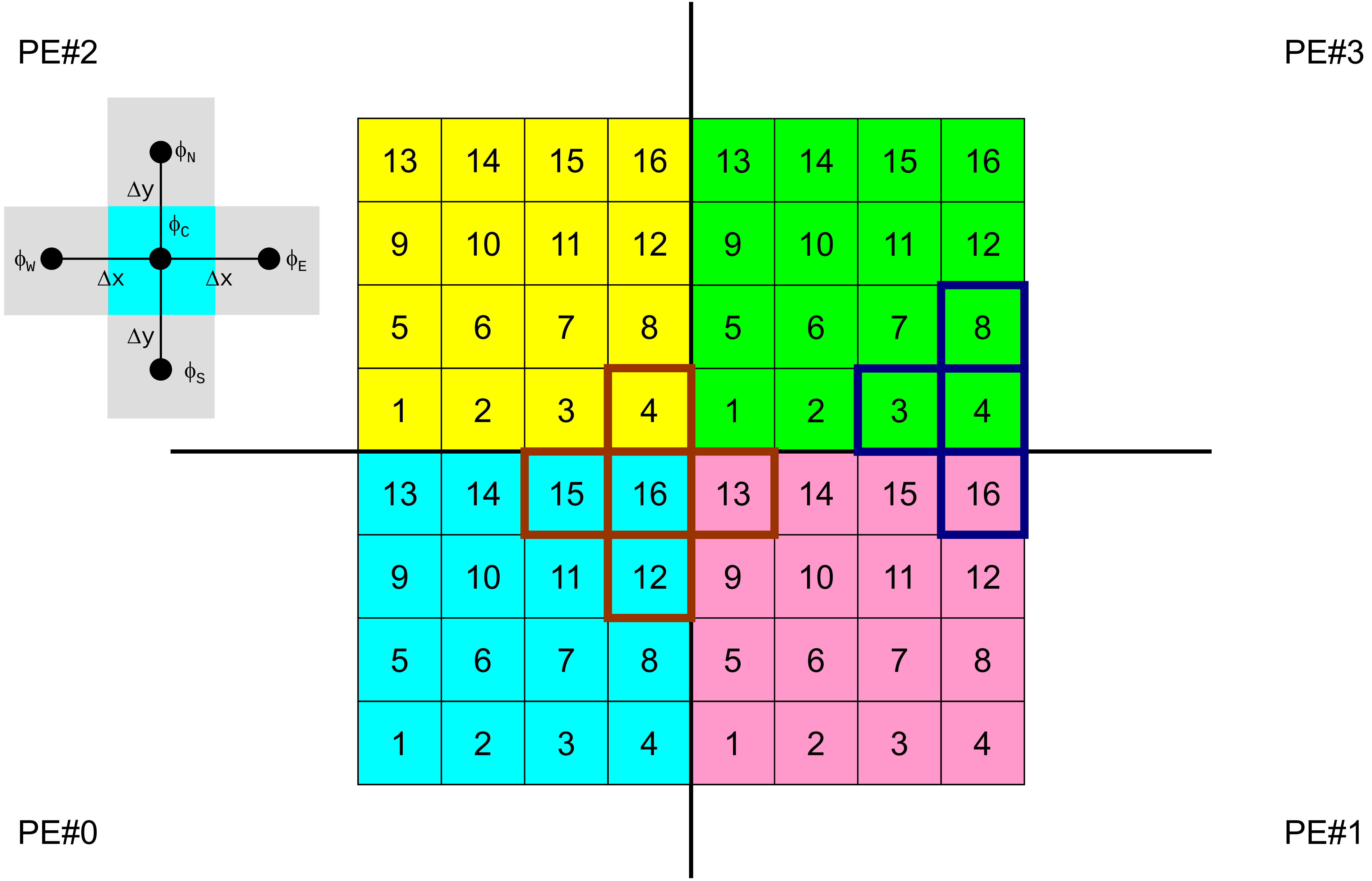
PE#0

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

PE#1

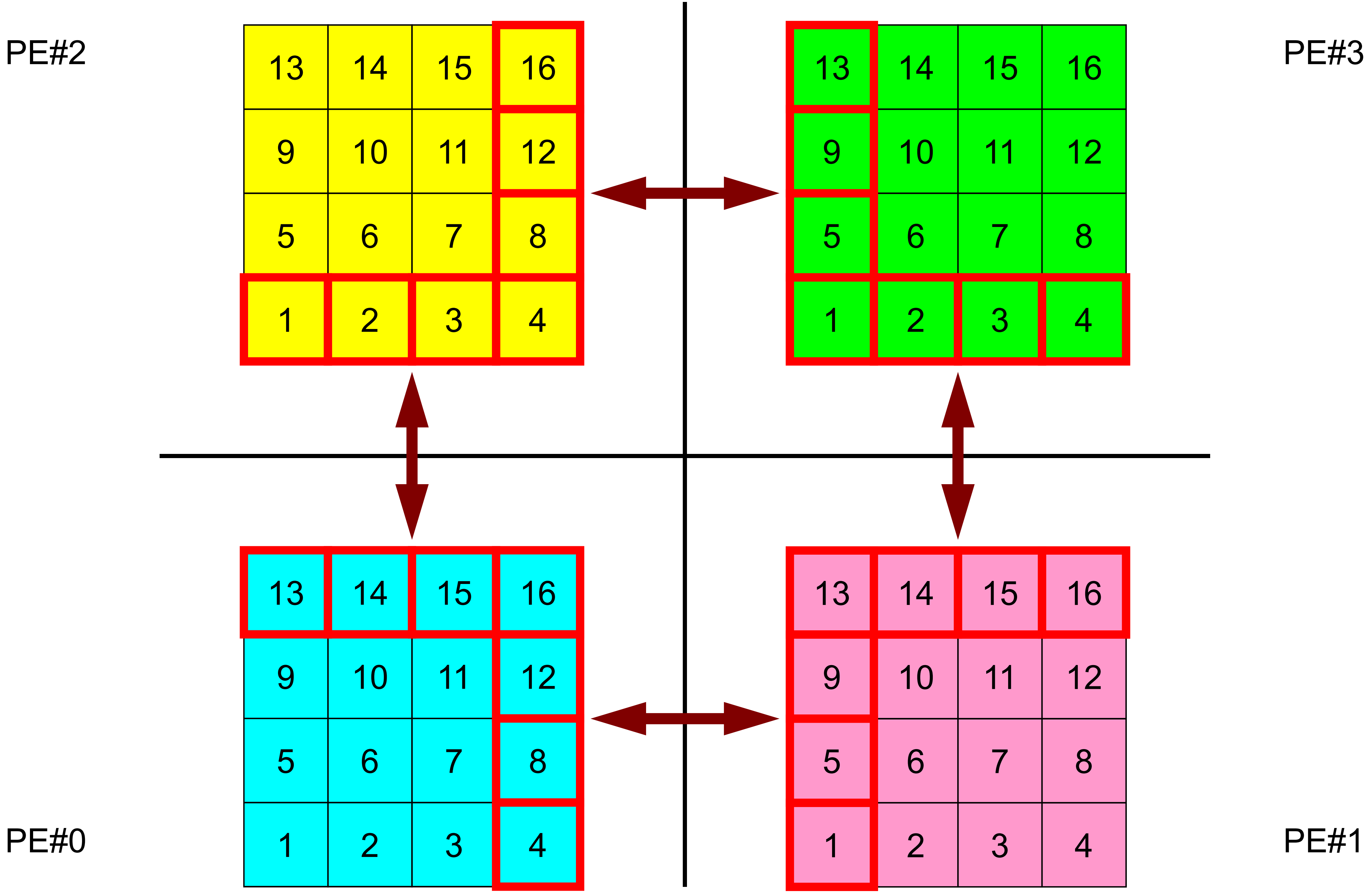
13	14	15	16
9	10	11	12
5	6	7	8
1	2	3	4

External Points: Overlapped Region



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

External Points: Overlapped Region



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

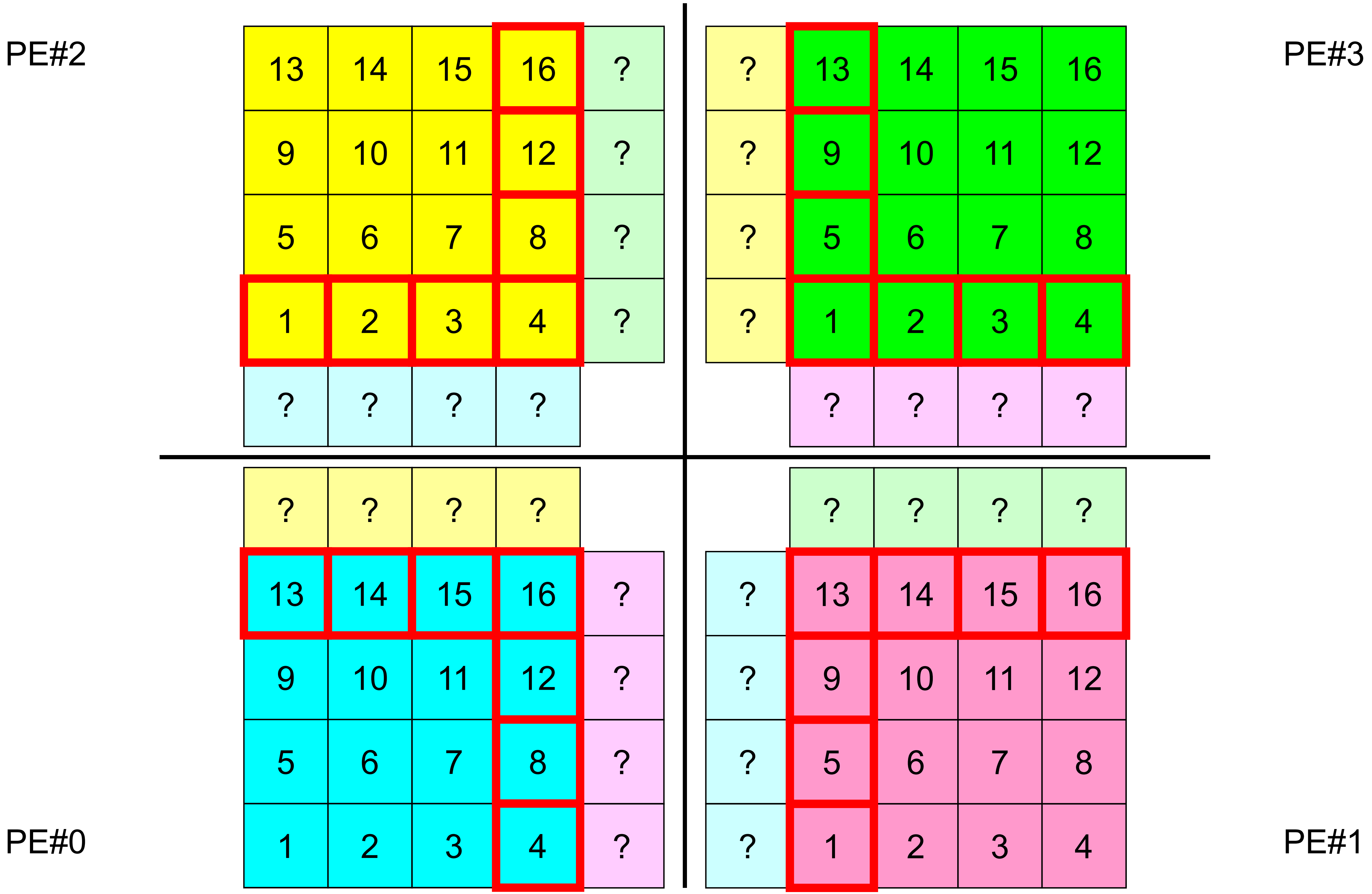
PE#0

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

Local ID of External Points ?



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
	?	?	?	?

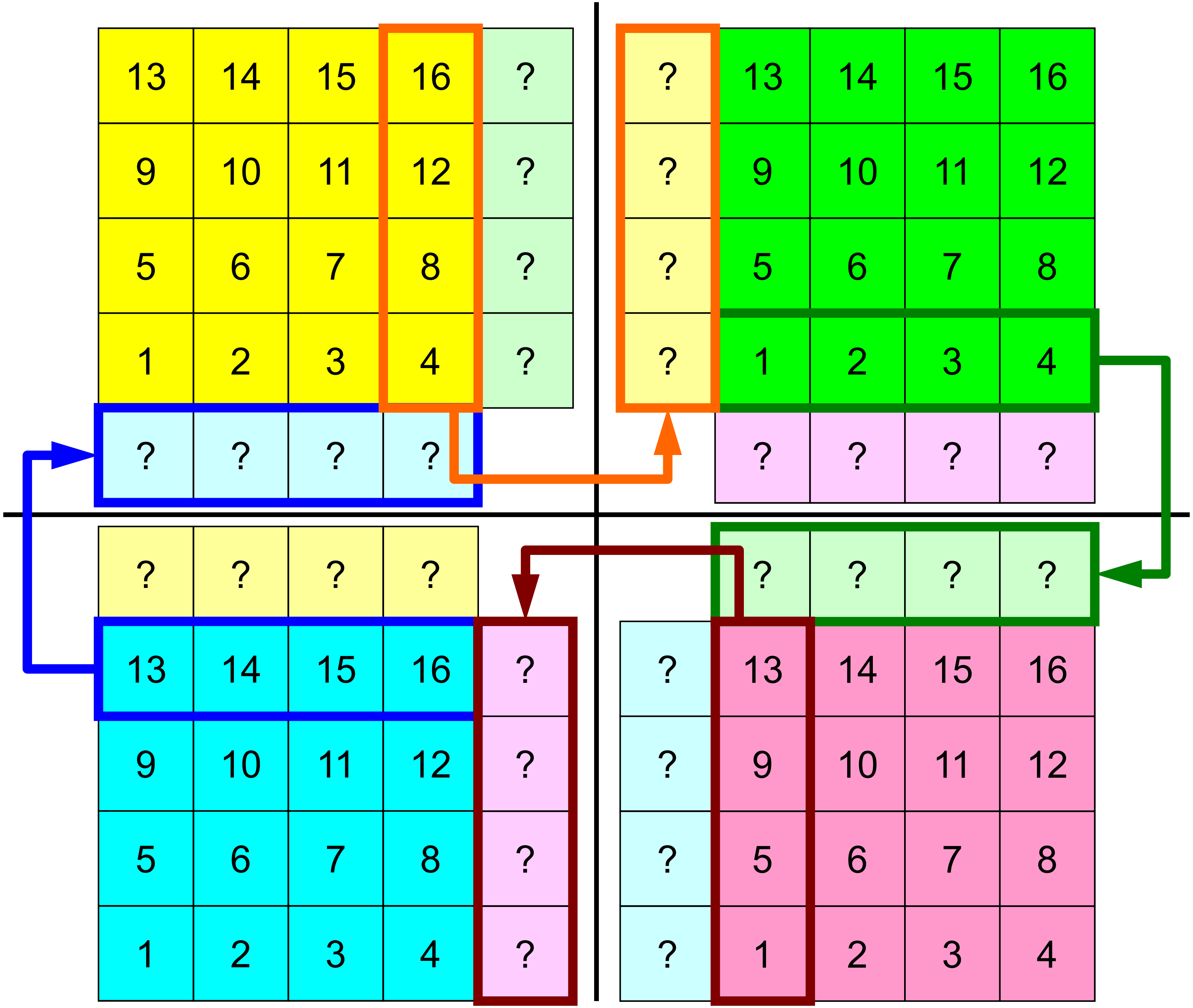
PE#0

?	?	?	?	
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

Overlapped Region



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
	?	?	?	?

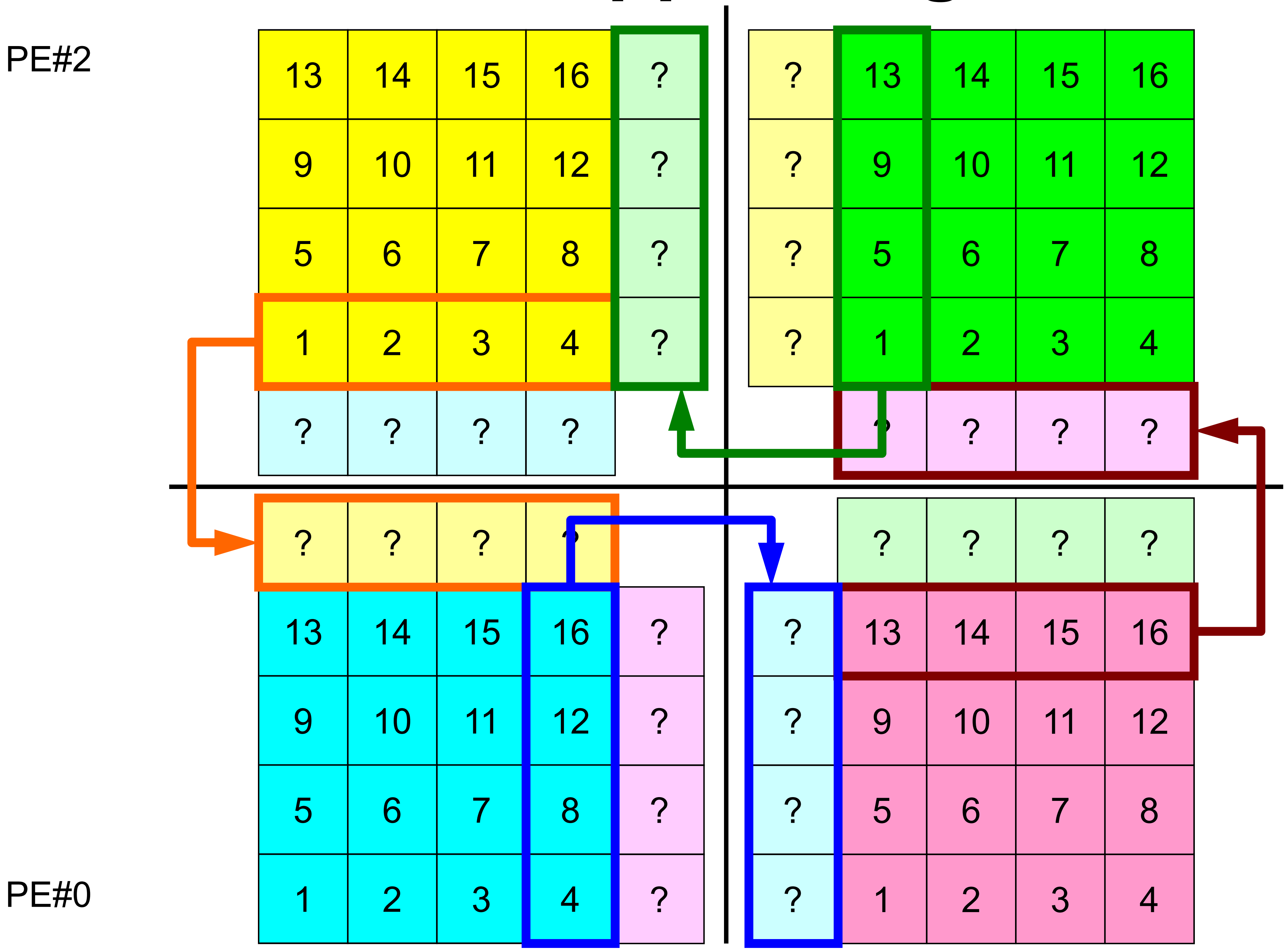
PE#0

?	?	?	?	
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

Overlapped Region

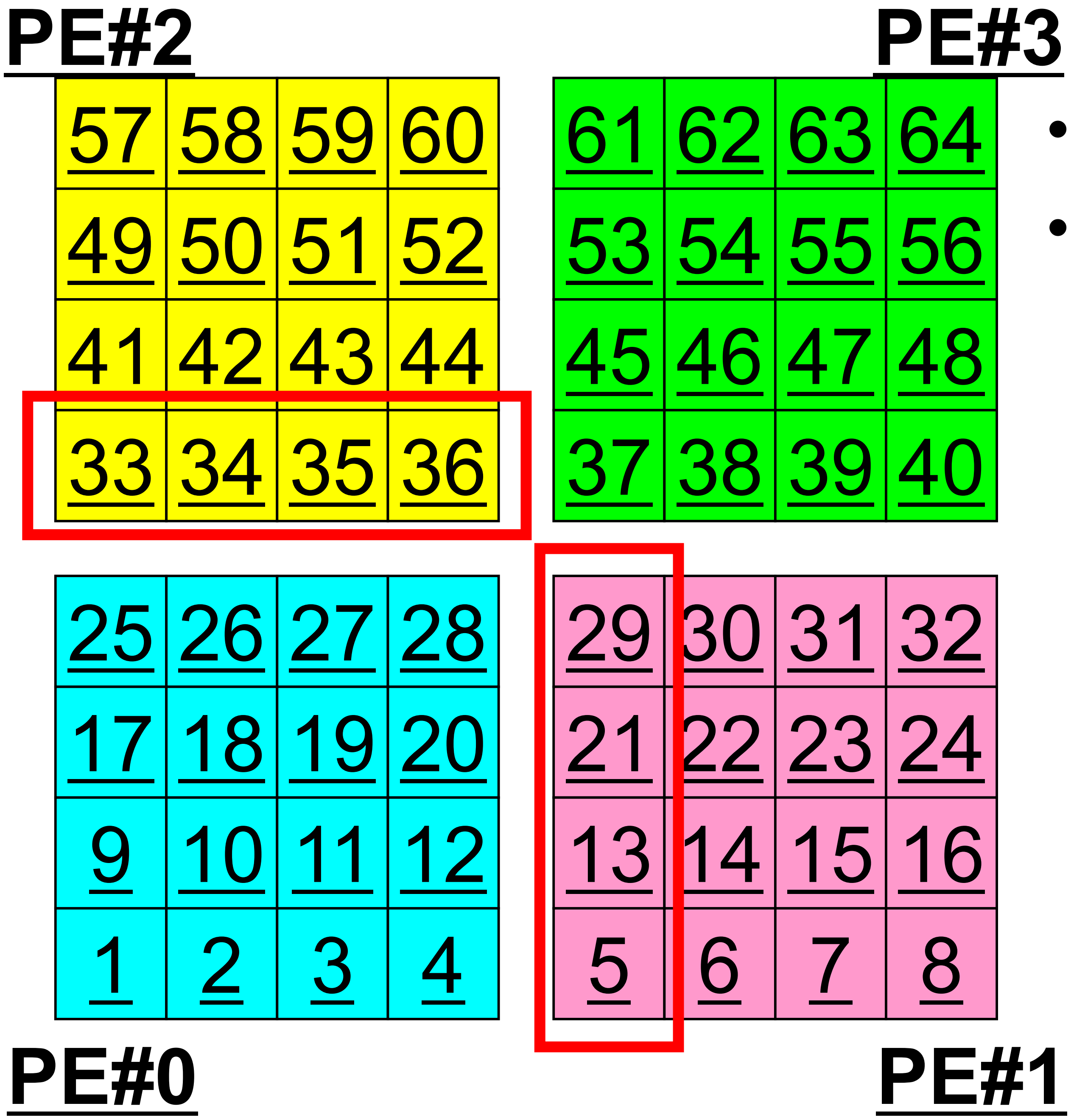


Problem Setting: 2D FDM

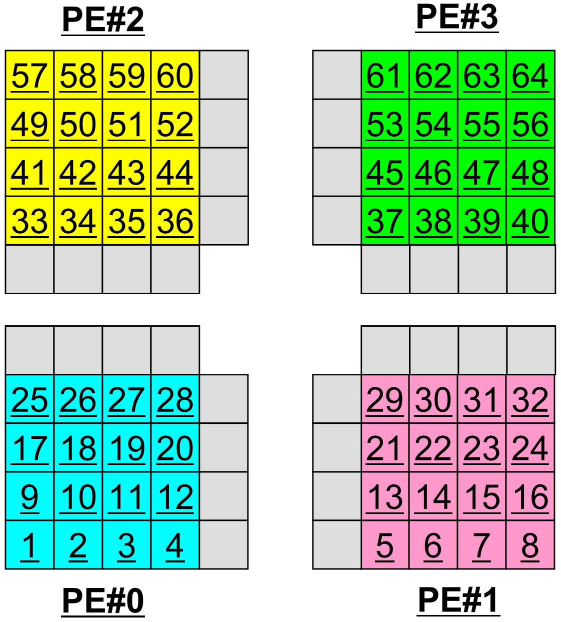
57	58	59	60	61	62	63	64
49	50	51	52	53	54	55	56
41	42	43	44	45	46	47	48
33	34	35	36	37	38	39	40
25	26	27	28	29	30	31	32
17	18	19	20	21	22	23	24
9	10	11	12	13	14	15	16
1	2	3	4	5	6	7	8

- 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



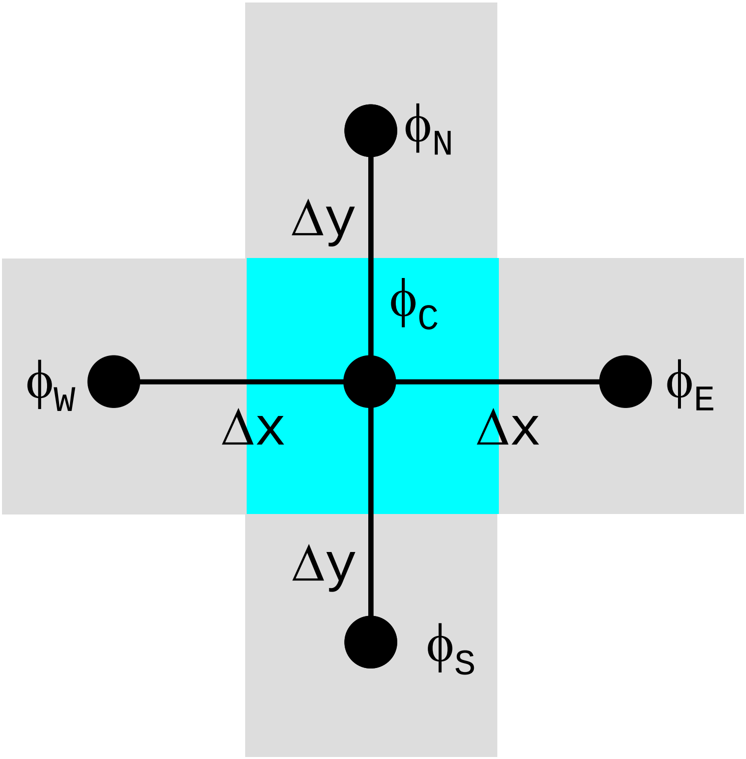
- 4 sub-domains.
- Info. of external points (global ID of mesh) is received from neighbors.
  - PE#0 receives □

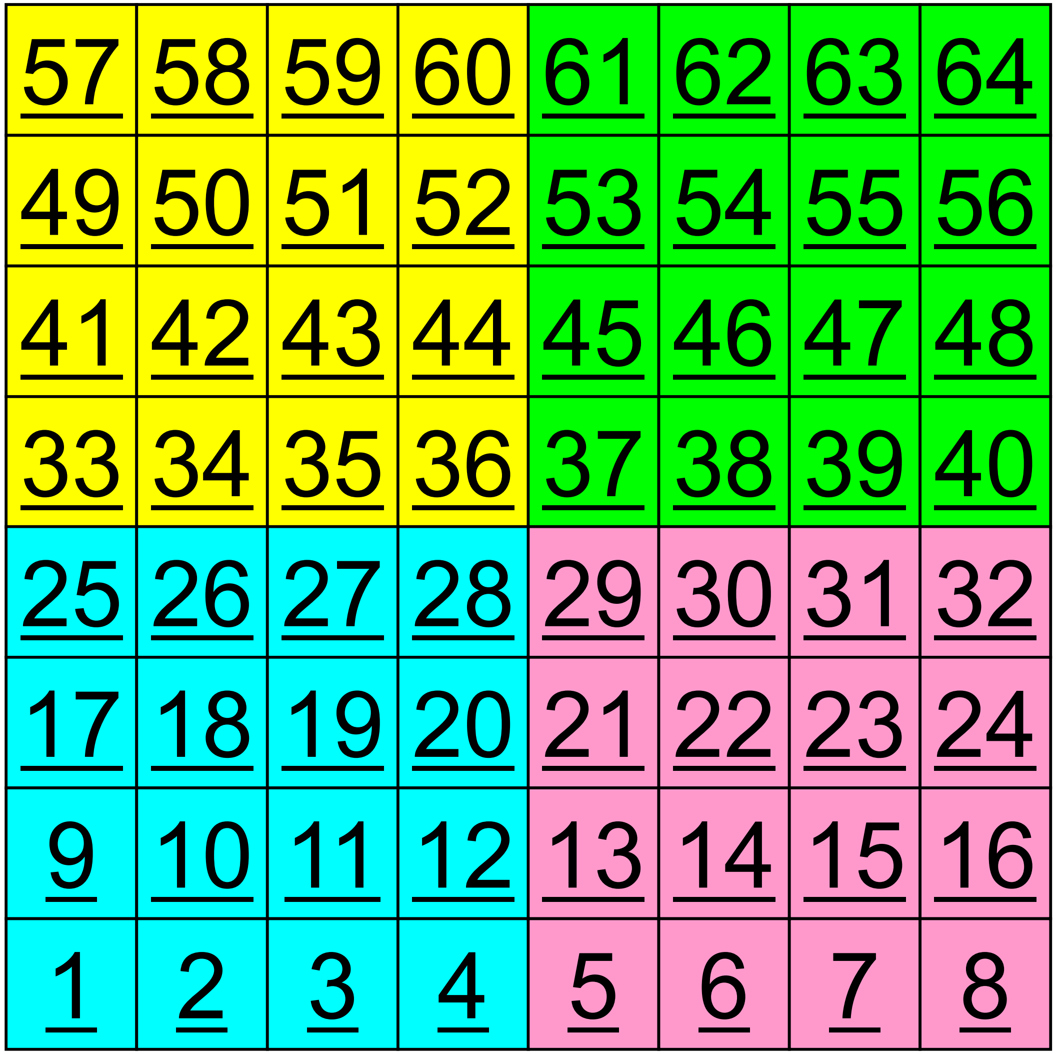


Operations of 2D FDM

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

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



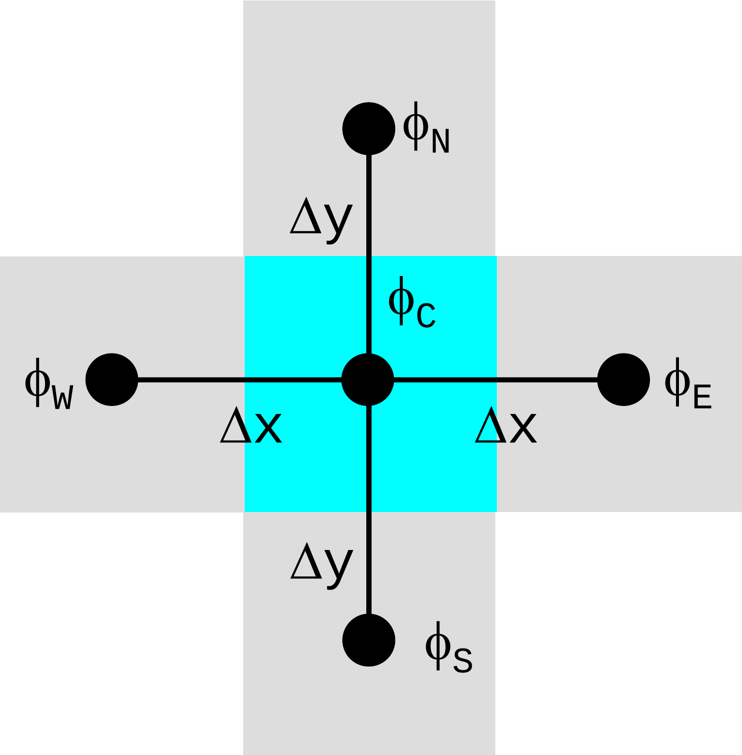


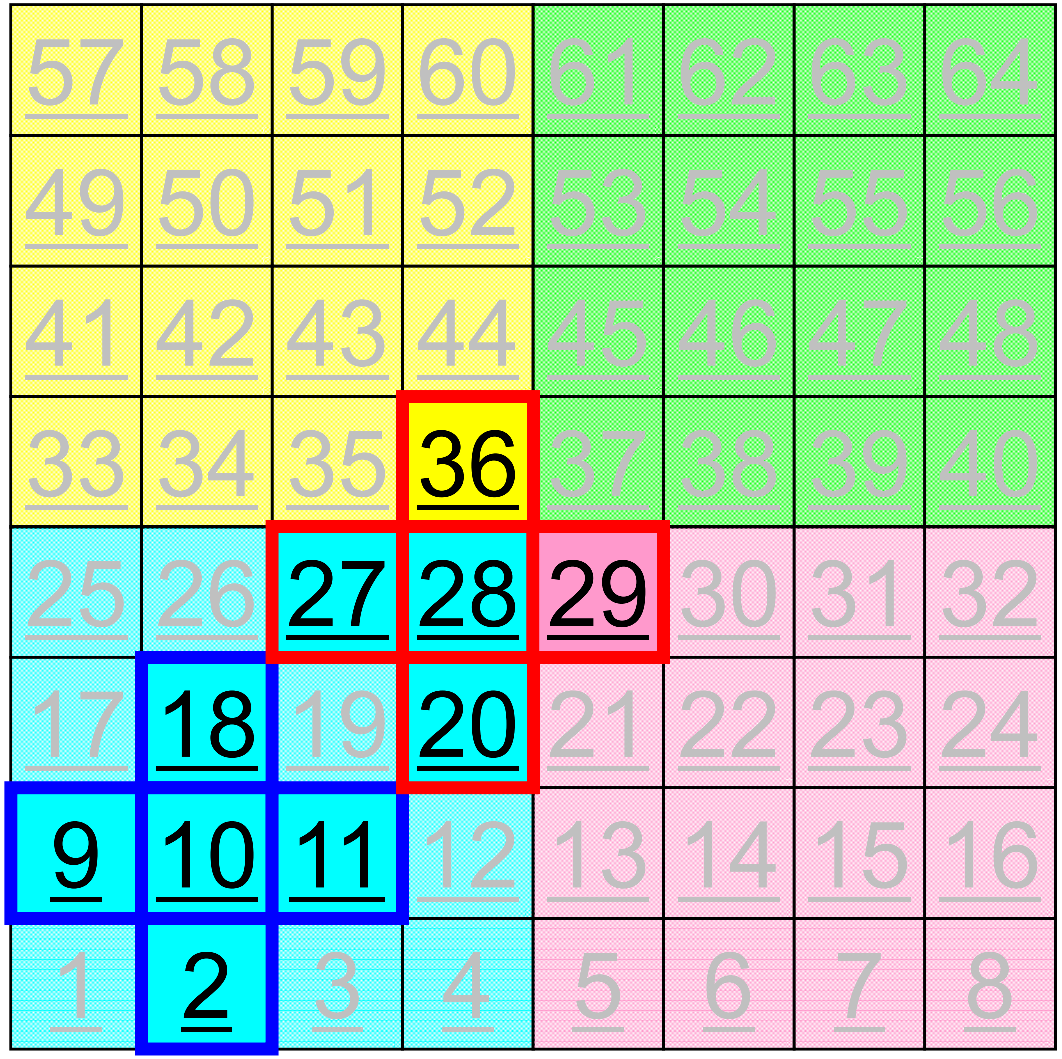
57	58	59	60	61	62	63	64
49	50	51	52	53	54	55	56
41	42	43	44	45	46	47	48
33	34	35	36	37	38	39	40
25	26	27	28	29	30	31	32
17	18	19	20	21	22	23	24
9	10	11	12	13	14	15	16
1	2	3	4	5	6	7	8

Operations of 2D FDM

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

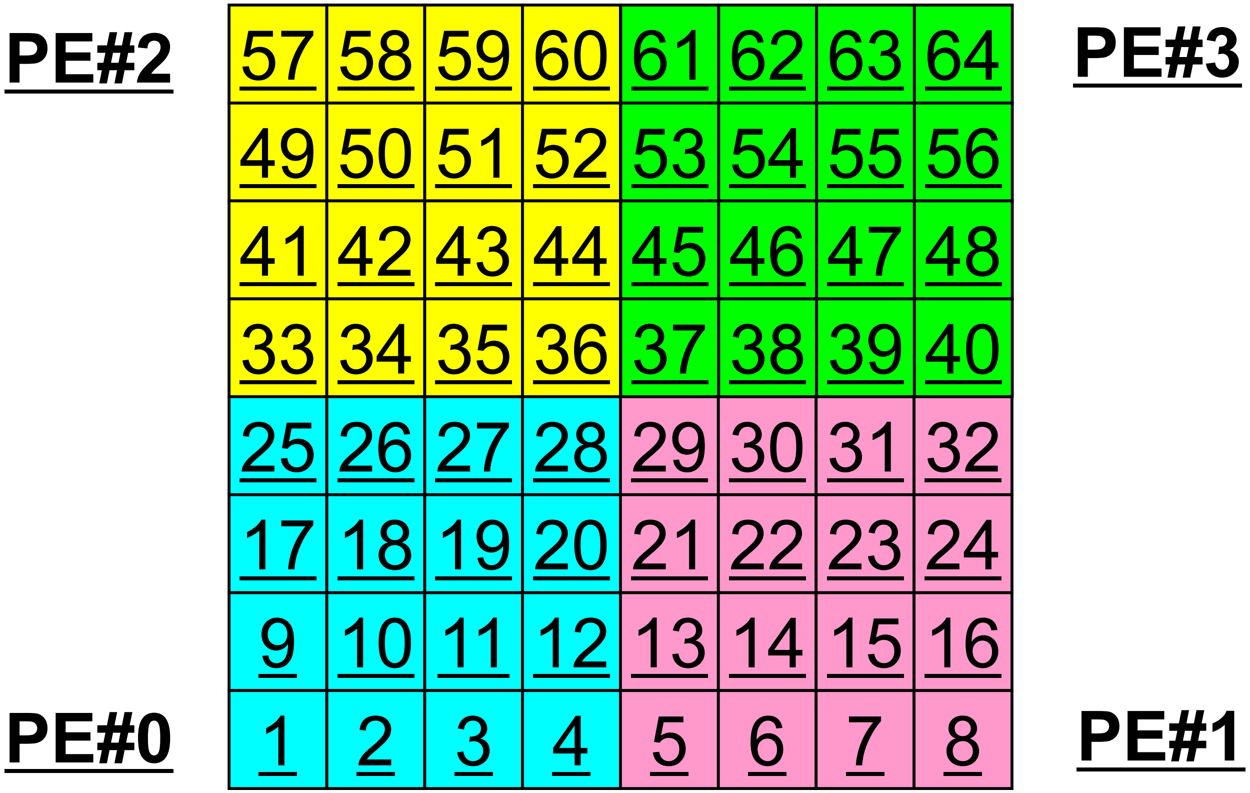
$$\left(\frac{\phi_E-2\phi_C+\phi_W}{\Delta x^2}\right)+\left(\frac{\phi_N-2\phi_C+\phi_S}{\Delta y^2}\right)=f_C$$





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

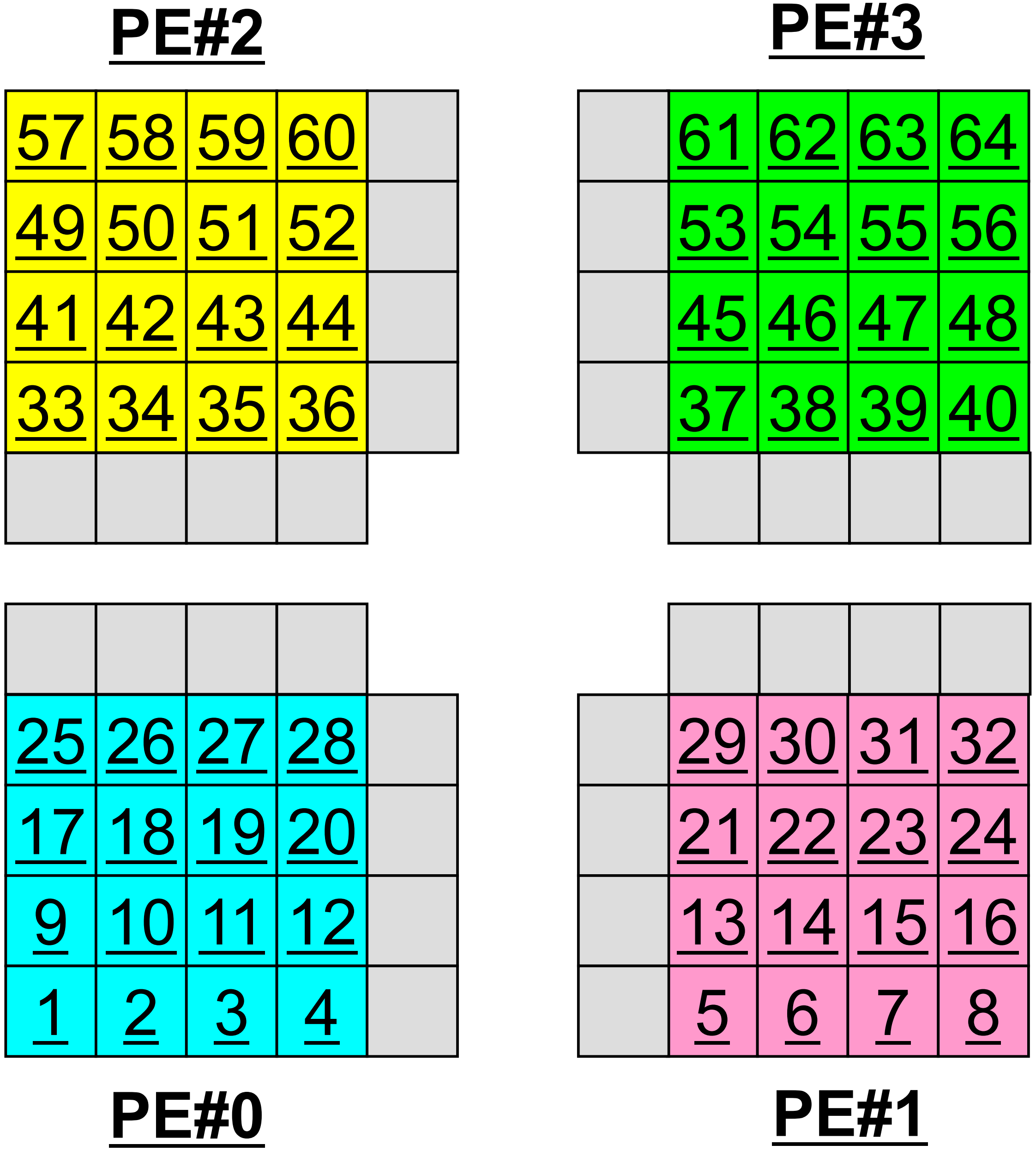
Computation (1/3)



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



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#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

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: ?

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

PE#0

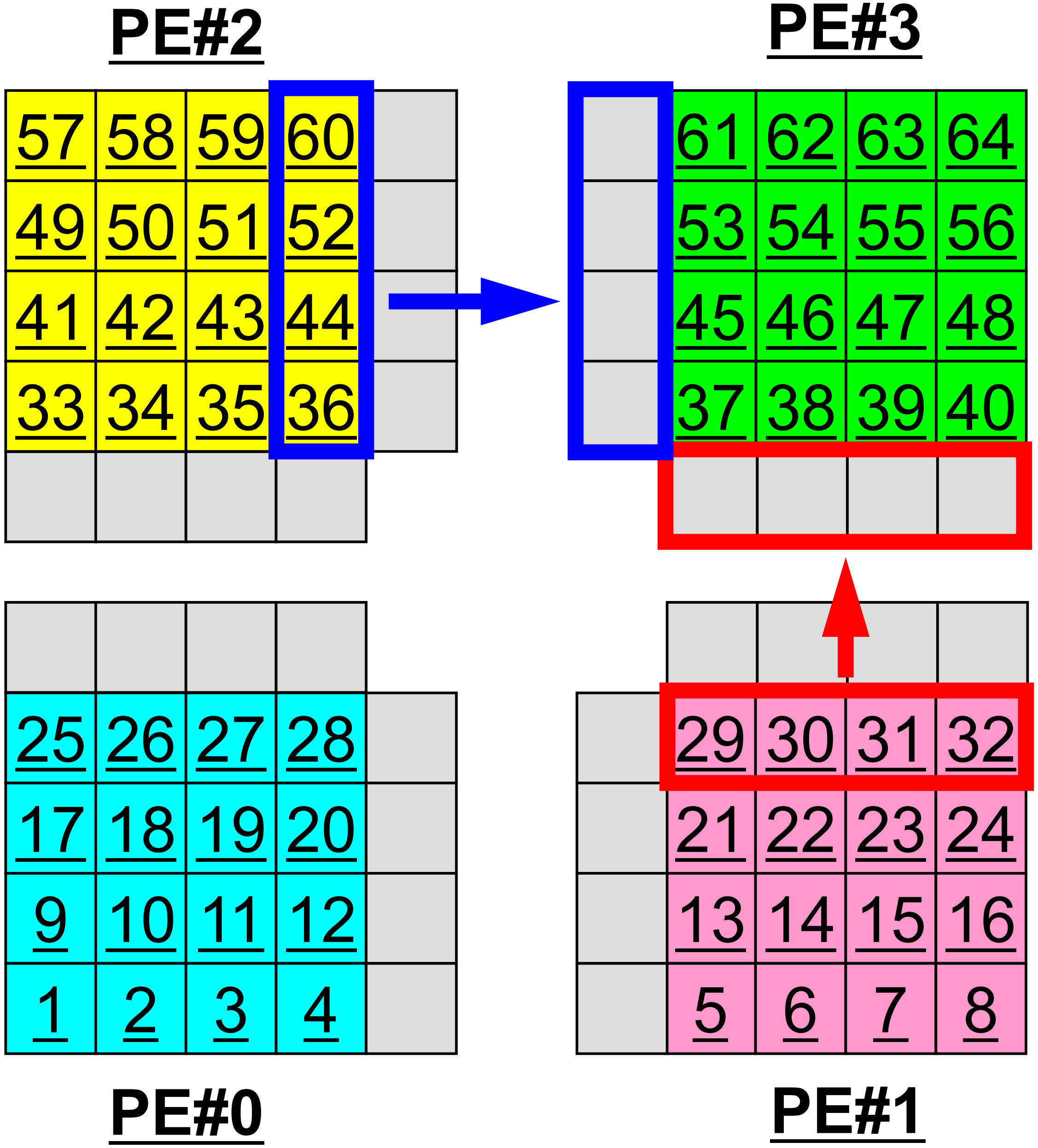
	29	30	31	32
	21	22	23	24
	13	14	15	16
	5	6	7	8

PE#1

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: ?



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

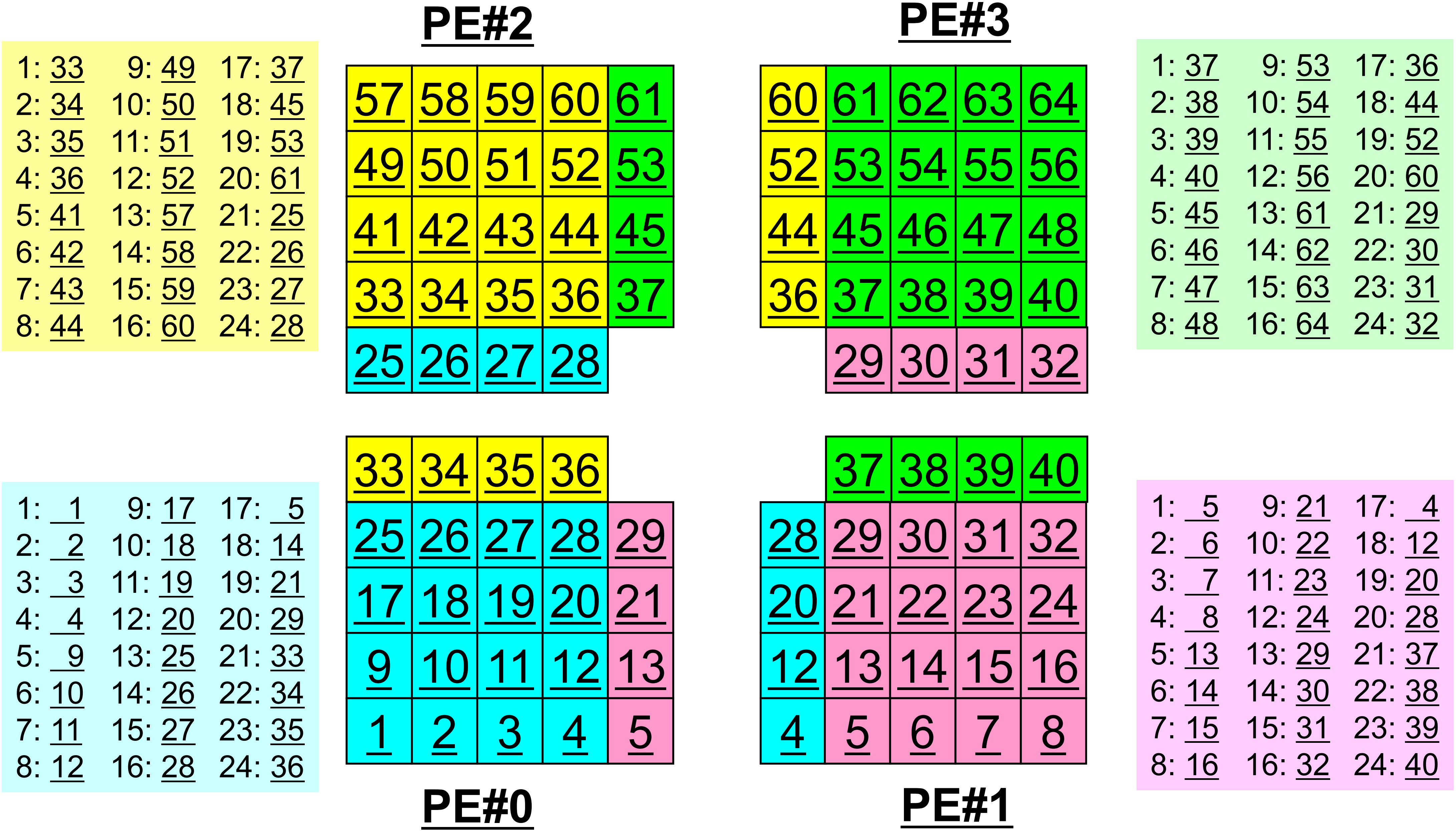
29	30	31	32
21	22	23	24
13	14	15	16
5	6	7	8

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

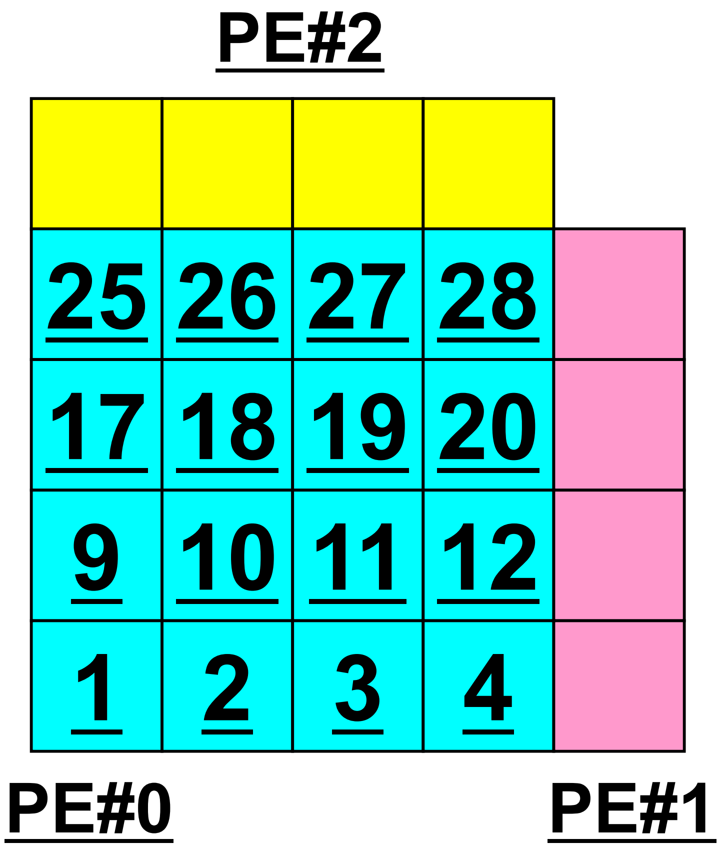
	37	38	39	40
28	29	30	31	32
20	21	22	23	24
12	13	14	15	16
4	5	6	7	8

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

Overview of Distributed Local Data

Example on PE#0

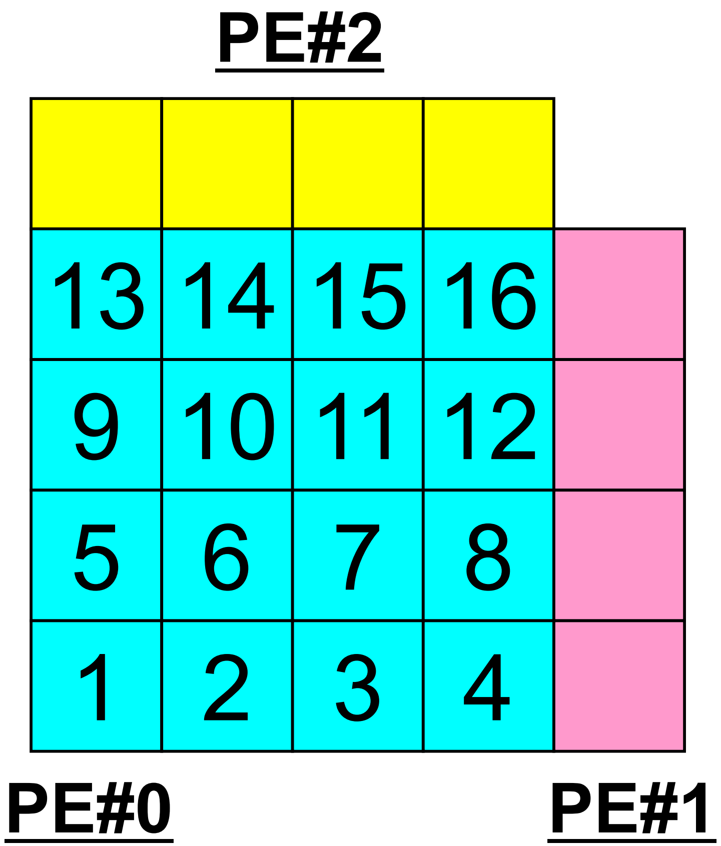


PE#2

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

PE#0 **PE#1**

Value at each mesh (= Global ID)



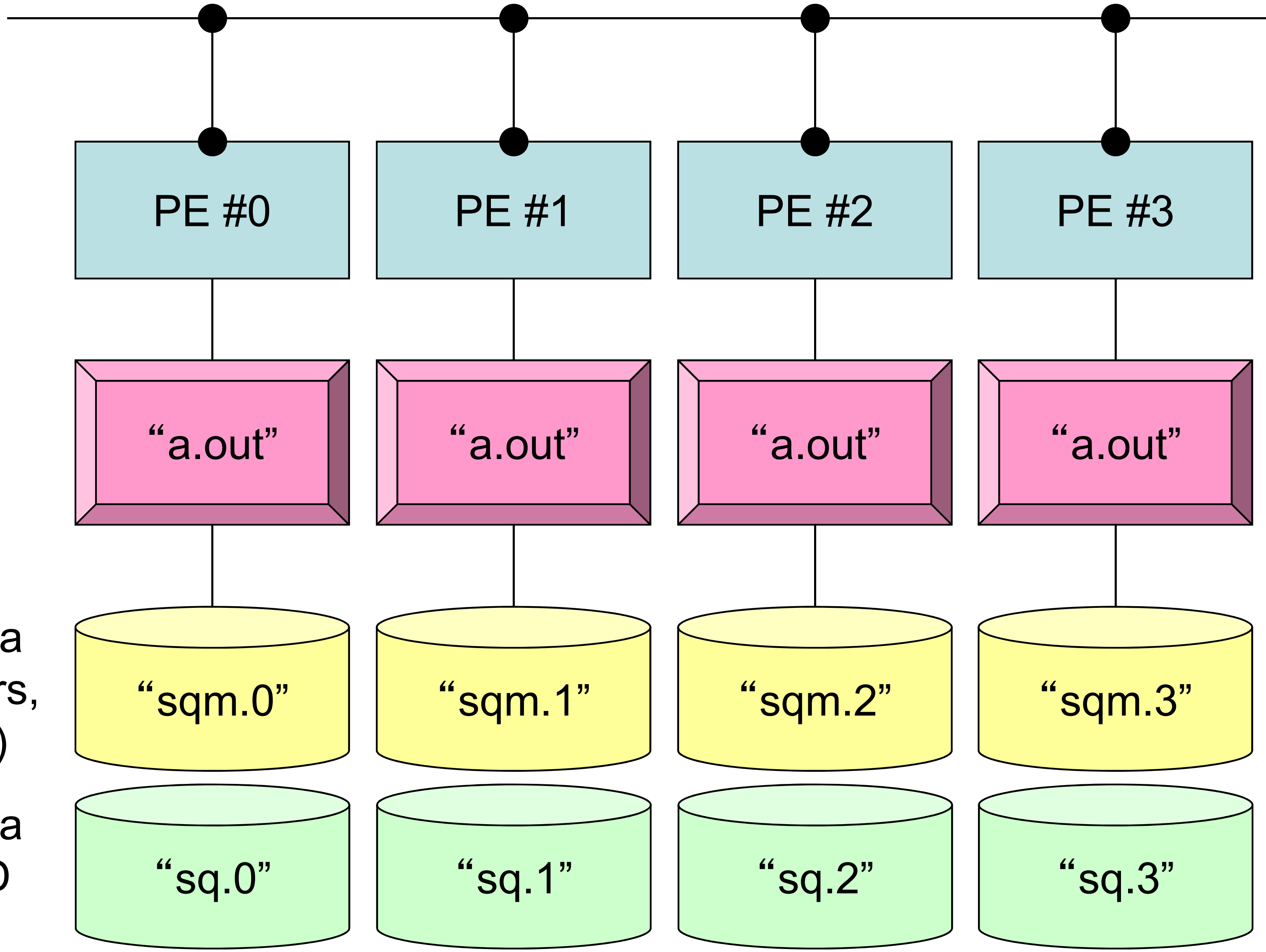
PE#2

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

PE#0 **PE#1**

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)

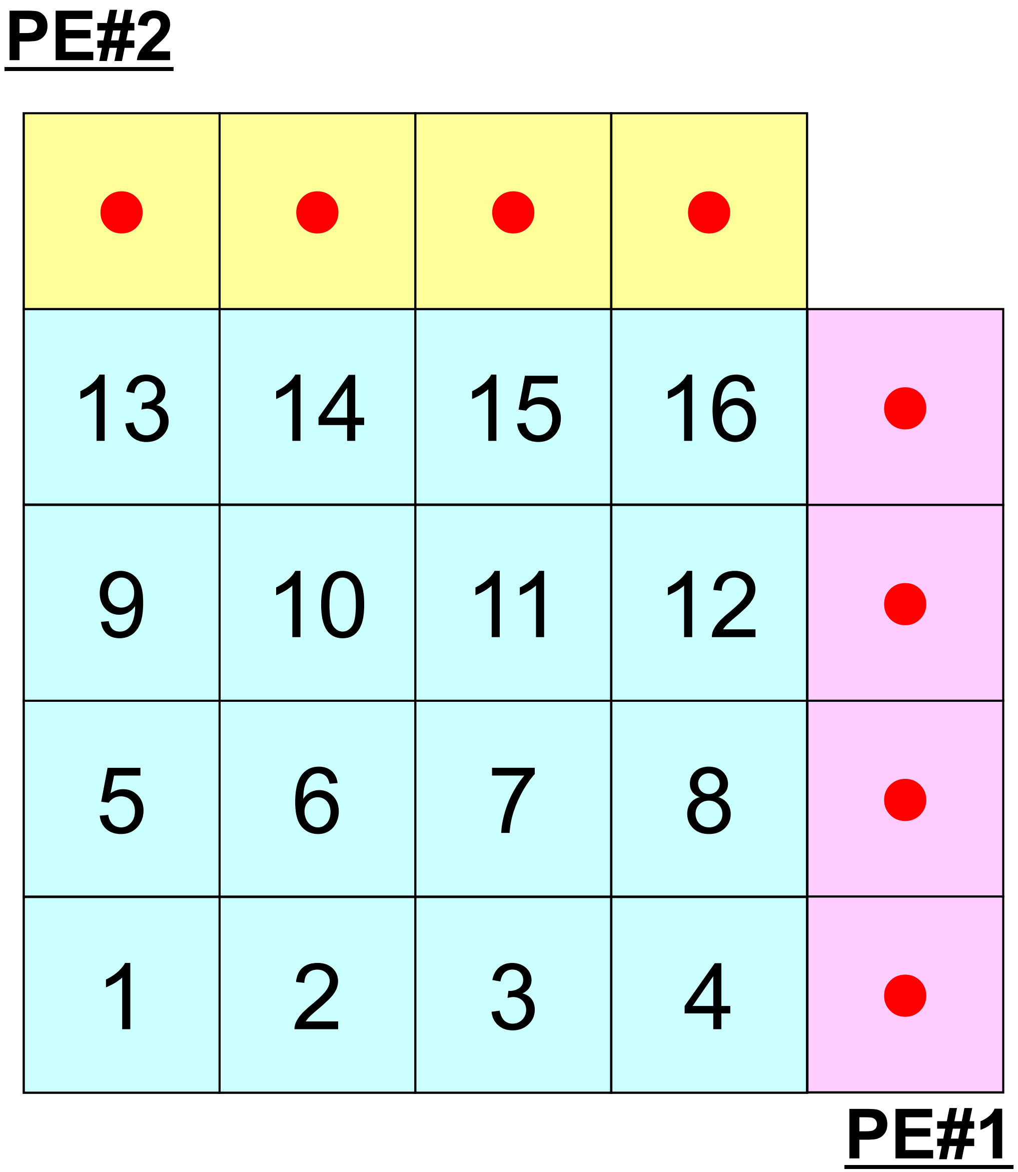
Internal Nodes/Points/Meshes

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)



Internal Nodes/Points/Meshes

Meshes originally assigned to the domain

External Nodes/Points/Meshes

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

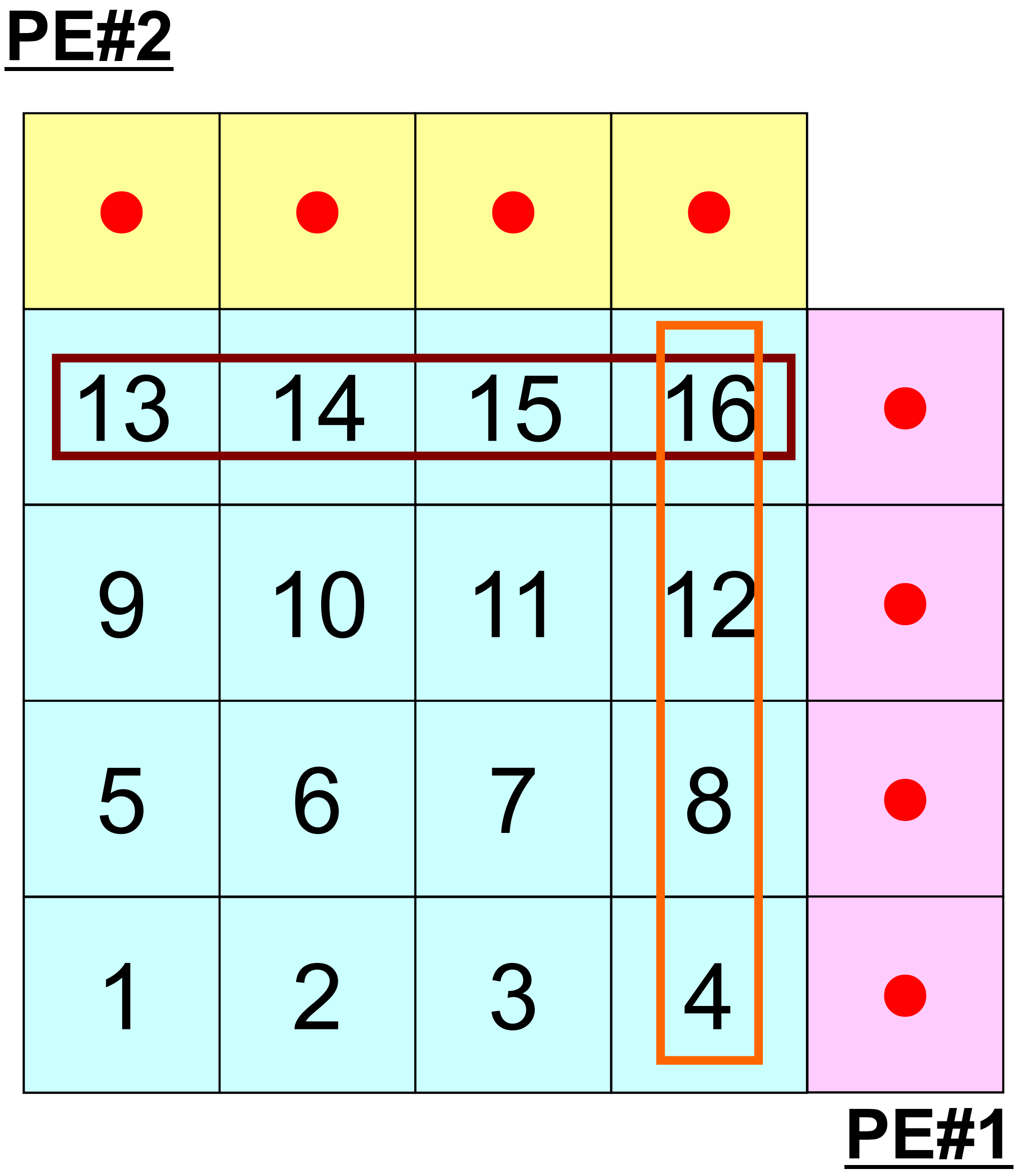
・Sleeves

・Halo

暈輪

2D FDM: PE#0

Information at each domain (3/4)



Internal Nodes/Points/Meshes

Meshes originally assigned to the domain

External Nodes/Points/Meshes

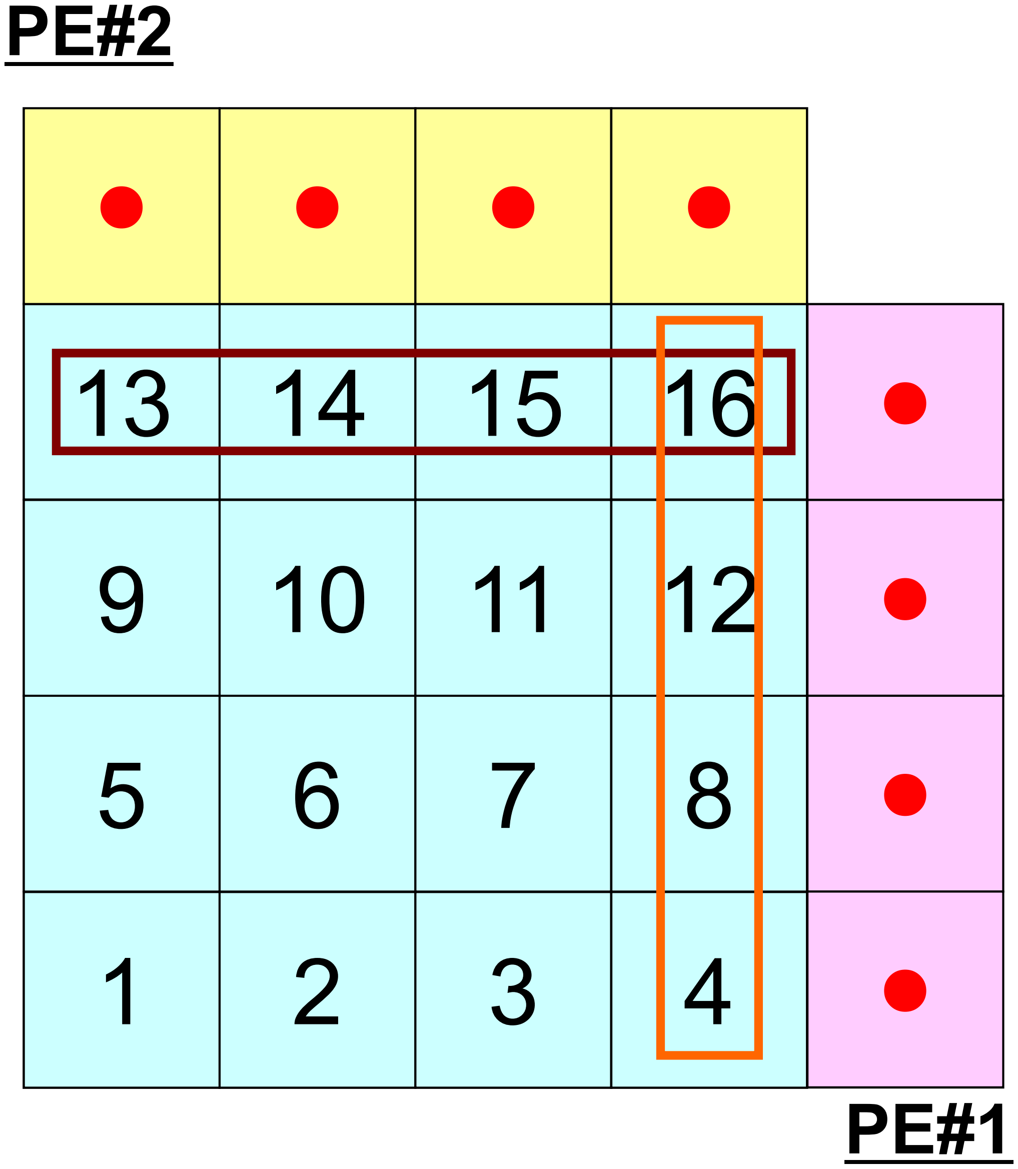
Meshes originally assigned to different domain, but required for computation of meshes in the domain (meshes in overlapped regions)

Boundary Nodes/Points/Meshes

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 Nodes/Points/Meshes

Meshes originally assigned to the domain

External Nodes/Points/Meshes

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

Boundary Nodes/Points/Meshes

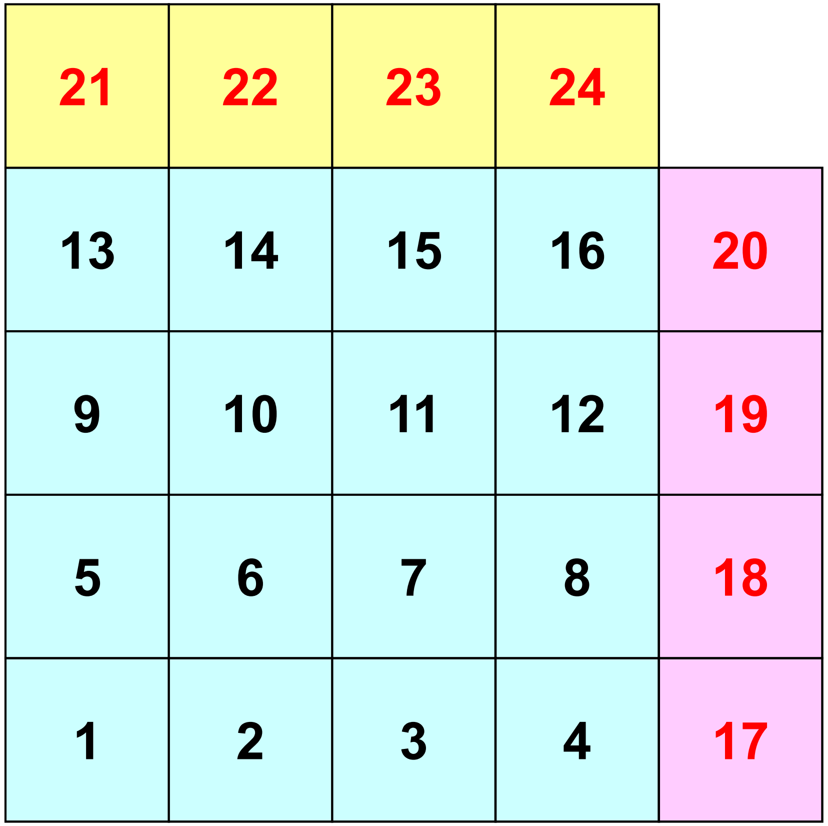
Internal points, which are also external points of other domains (used in computations of meshes in other domains)

Relationships between Domains

Communication Table: External/Boundary Points

Neighbors

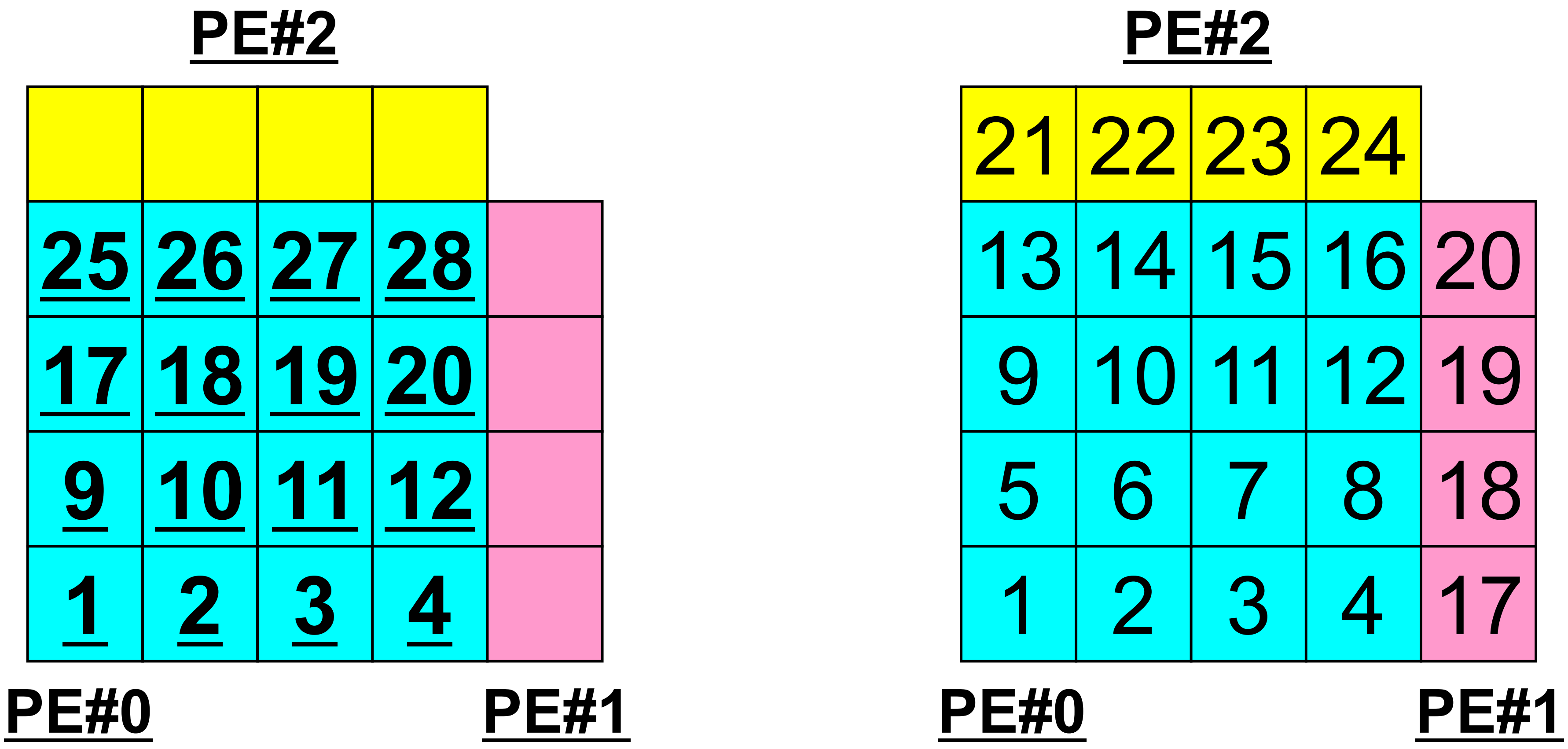
Description of Distributed Local Data



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

Overview of Distributed Local Data

Example on PE#0



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

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

Value at each mesh (= Global ID)

Local ID

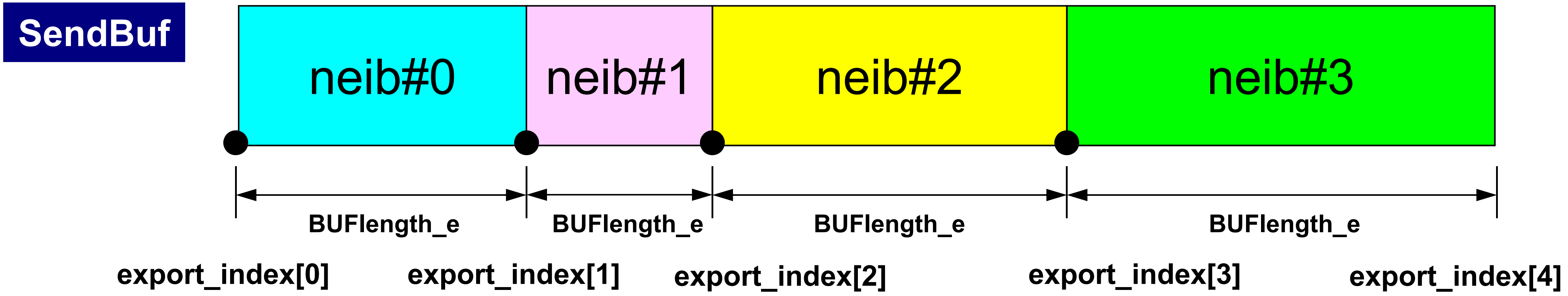
Generalized Communication Table: Send

- Neighbors
  - NeibPETot, NeibPE[neib]
- Message size for each neighbor
  - export_index[neib], neib= 0, NeibPETot-1
- ID of **boundary** nodes
  - export_item[k], k= 0, export_index[NeibPETot]-1
- Messages to each neighbor
  - SendBuf[k], k= 0, export_index[NeibPETot]-1

C

SEND: MPI_Isend/Irecv/Waitall

C



SendBuf

neib#0 neib#1 neib#2 neib#3

BUFlength_e BUFlength_e BUFlength_e BUFlength_e

export_index[0] export_index[1] export_index[2] export_index[3] export_index[4]

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];
   }
}
```

Copied to sending buffers

```
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);
```

Generalized Communication Table: Receive

- Neighbors
  - NeibPETot , NeibPE[neib]
- Message size for each neighbor
  - import_index[neib], neib= 0, NeibPETot-1
- ID of **external** nodes
  - import_item[k], k= 0, import_index[NeibPETot]-1
- Messages from each neighbor
  - RecvBuf[k], k= 0, import_index[NeibPETot]-1

C

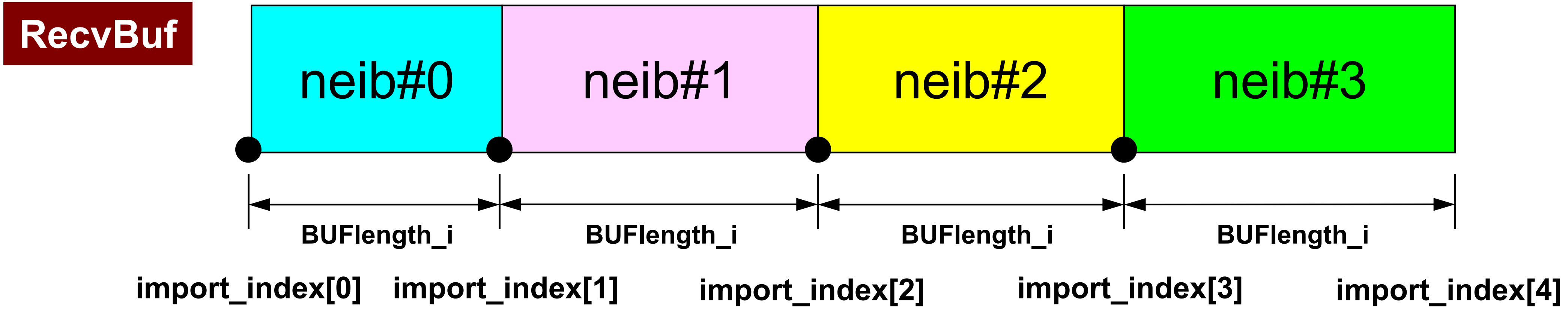
RECV: MPI_Isend/Irecv/Waitall

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



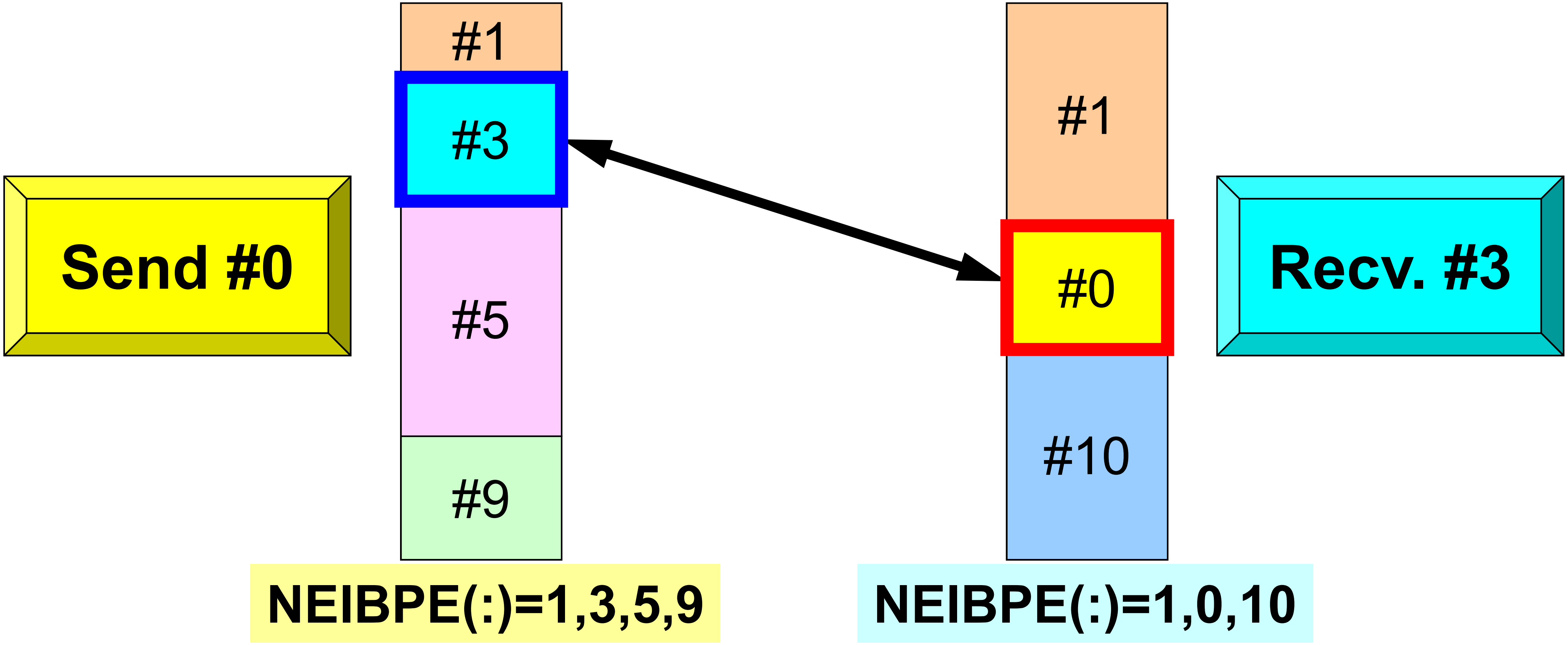
Relationship SEND/RECV

```
 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
```

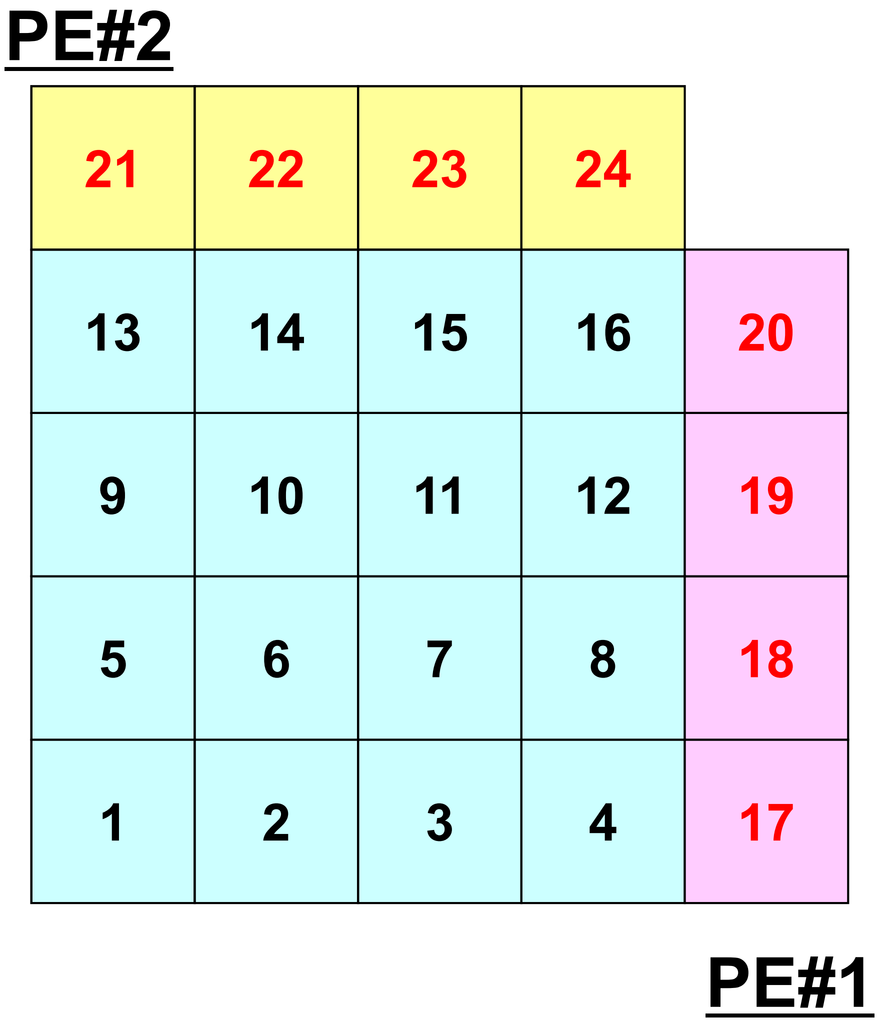
- Consistency of ID’s of sources/destinations, size and contents of messages !
- Communication occurs when NEIBPE[neib] matches

Relationship SEND/RECV (#0 to #3)



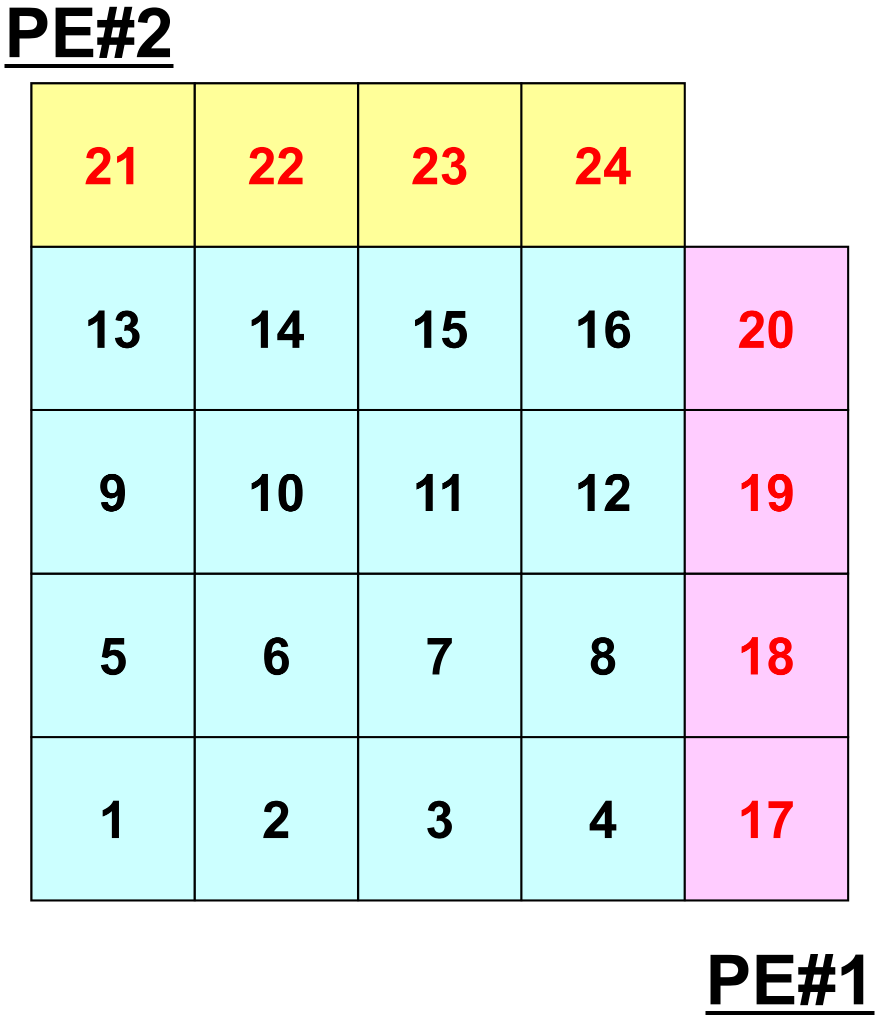
- Consistency of ID’s of sources/destinations, size and contents of messages !
- Communication occurs when NEIBPE(neib) matches

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)



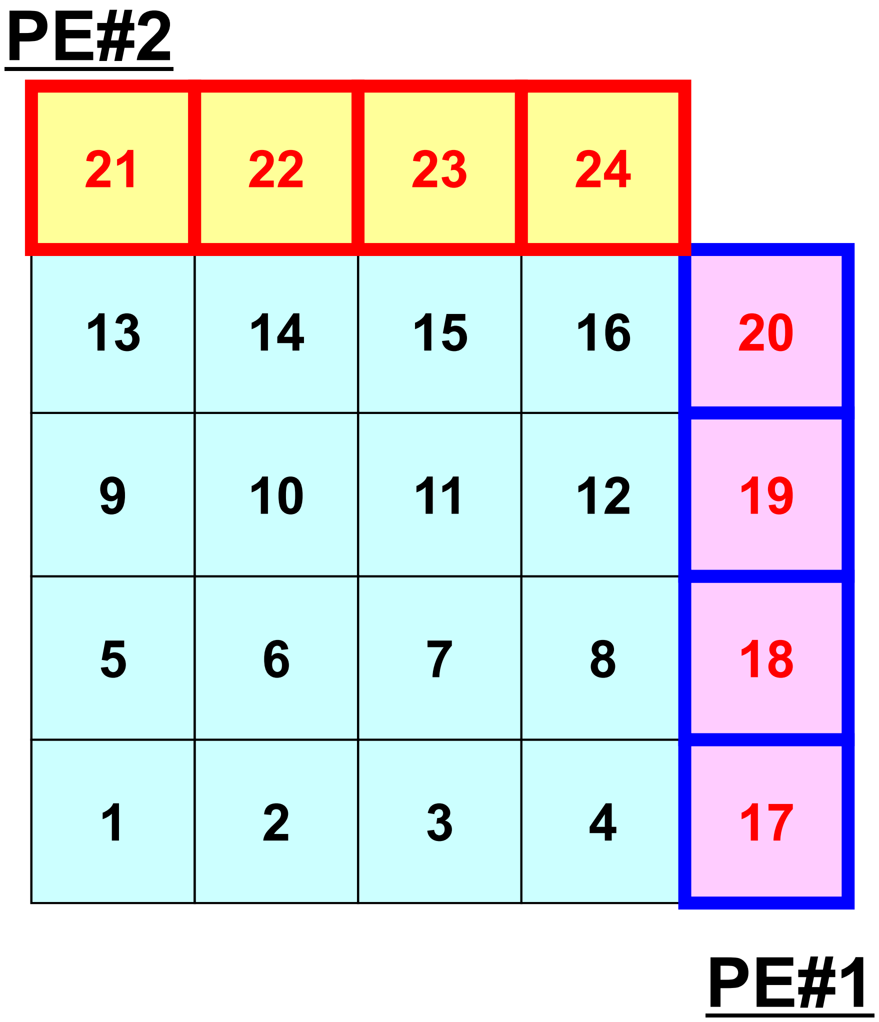
PE#2

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

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
```

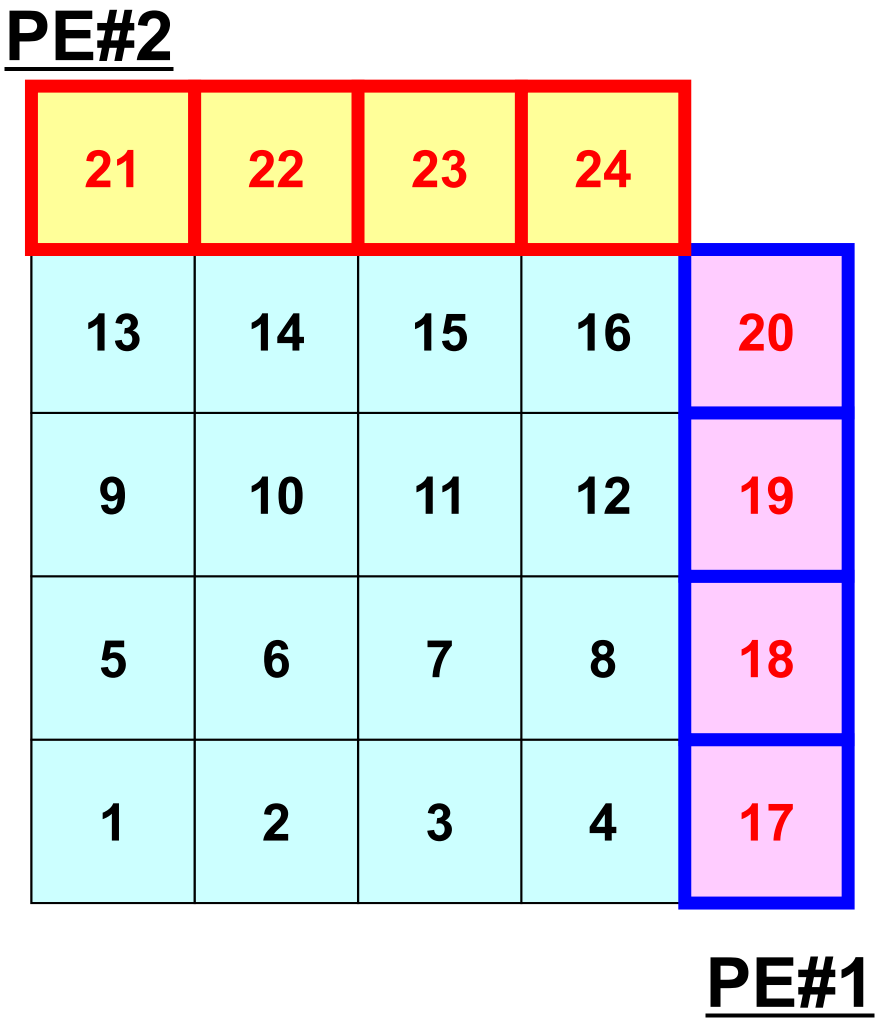
Generalized Comm. Table (3/6)



```
#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)

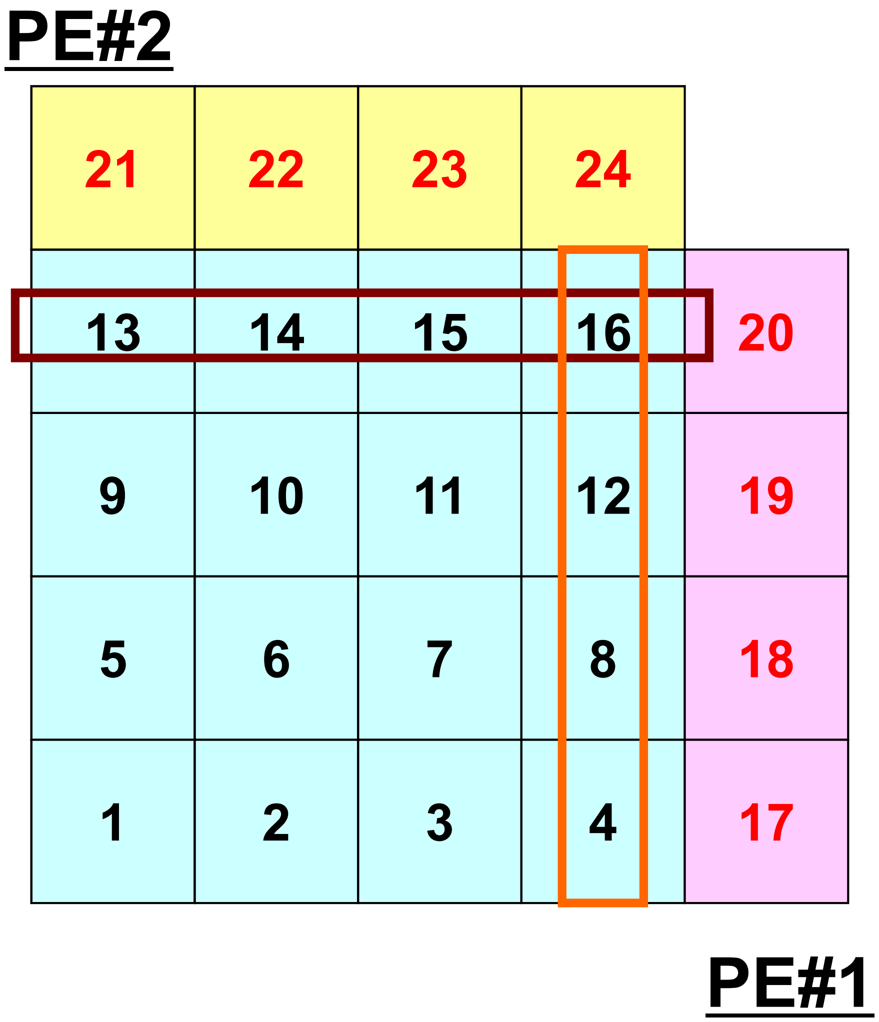


```
#NEIBPEtot
2
#NEIBPE
1  2
#NODE
24 16
#IMPORT_index
4  8
#IMPORT_items
17
18
19
20
21
22
23
24
#EXPORT_index
4  8
#EXPORT_items
4
8
12
16
13
14
15
16
```

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

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

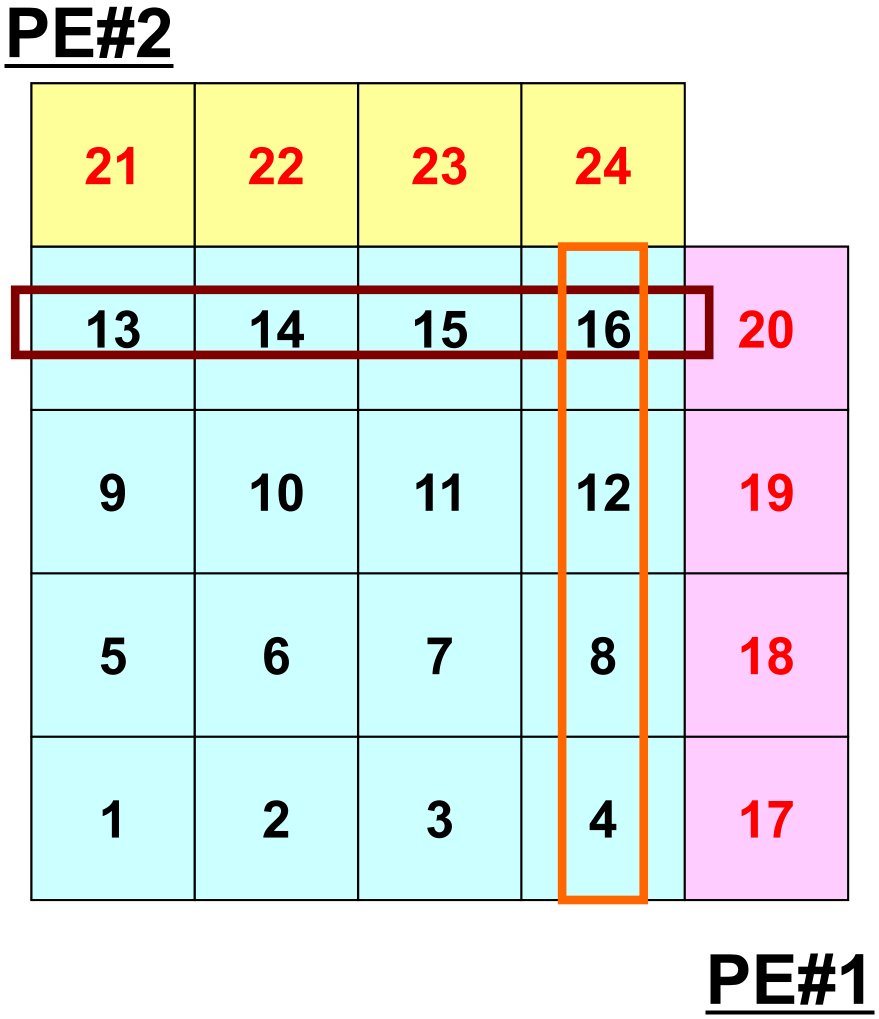
Generalized Comm. Table (5/6)



```
#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).

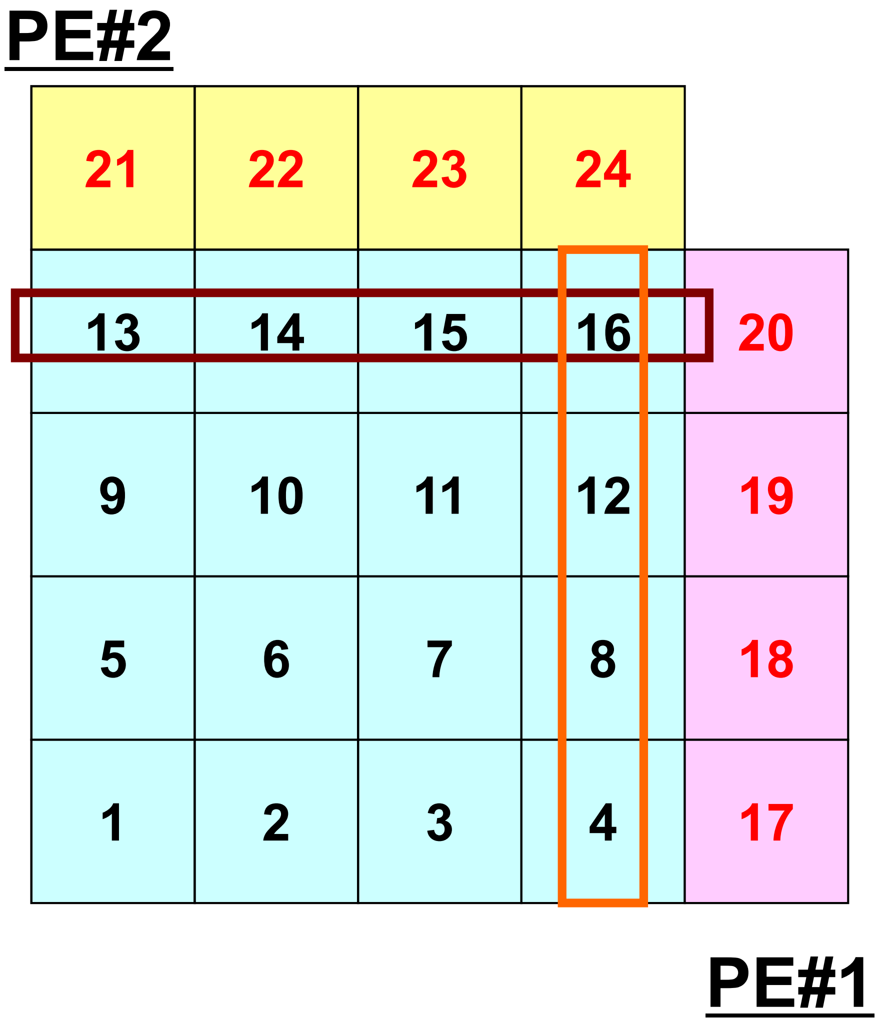
Generalized Comm. Table (6/6)



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

```
#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             exported to 1st Neighbor
12            (PE#1) (1st-4th items)
16
13
14            exported to 2nd Neighbor
15            (PE#2) (5th-8th items)
16
```

Generalized Comm. Table (6/6)

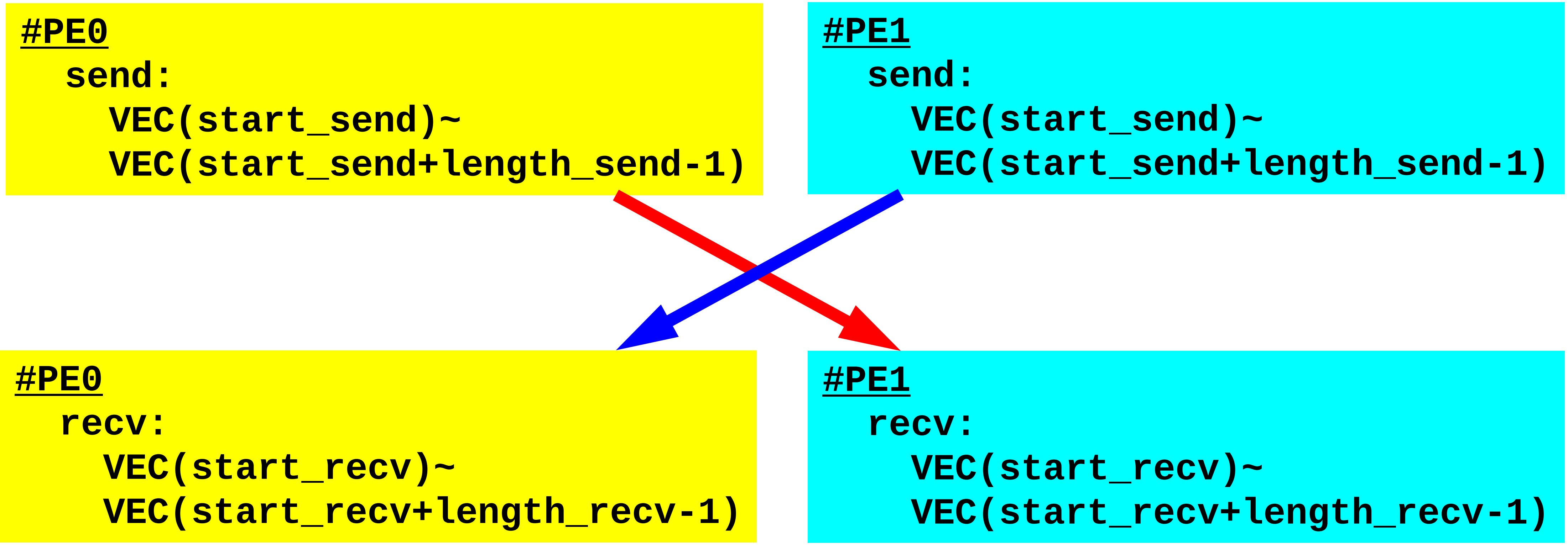


21	22	23	24	
13	14	15	16	20
9	10	11	12	19
5	6	7	8	18
1	2	3	4	17

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

Copying files/2D FDM on Oakleaf-FX

```
>$ cd
>$ cp /home/z30088/omp/hybrid-c.tar .
>$ cp /home/z30088/omp/hybrid-f.tar .
>$ tar xvf hybrid-c.tar (or hybrid-f.tar)
>$ cd hybrid
>$ ls
   S2  fvm  (<$O-S2>, <$O-fvm>)
```

```
$ cd <$O-S2>
$ mpifrtpx –Kfast sq-sr1.f
$ mpifccpx –Kfast sq-sr1.c

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

Job Script for FX10:go4.sh

- **<$O-S2>/go4.sh**
- Scheduling + Shell Script

```
#!/bin/sh
#PJM -L “node=1“                Number of Nodes
#PJM -L “elapse=00:10:00“       Computation Time
#PJM -L “rscgrp=lecture7“       Name of “QUEUE”
#PJM -g “gt17“                  Group Name (Wallet)
#PJM -j
#PJM -o “teat.lst“              Standard Output
#PJM --mpi “proc=4“             MPI Process #

mpiexec ./a.out                 Execs
```

8 proc’s	16 proc’s	32proc’s	64 proc’s	192 proc’s
“node=1“	“node=1“	“node=2“	“node=4“	“node=12“
“proc=8”	“proc=16”	“proc=32”	“proc=64”	“proc=192”

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]--;}
```

C

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

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)

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

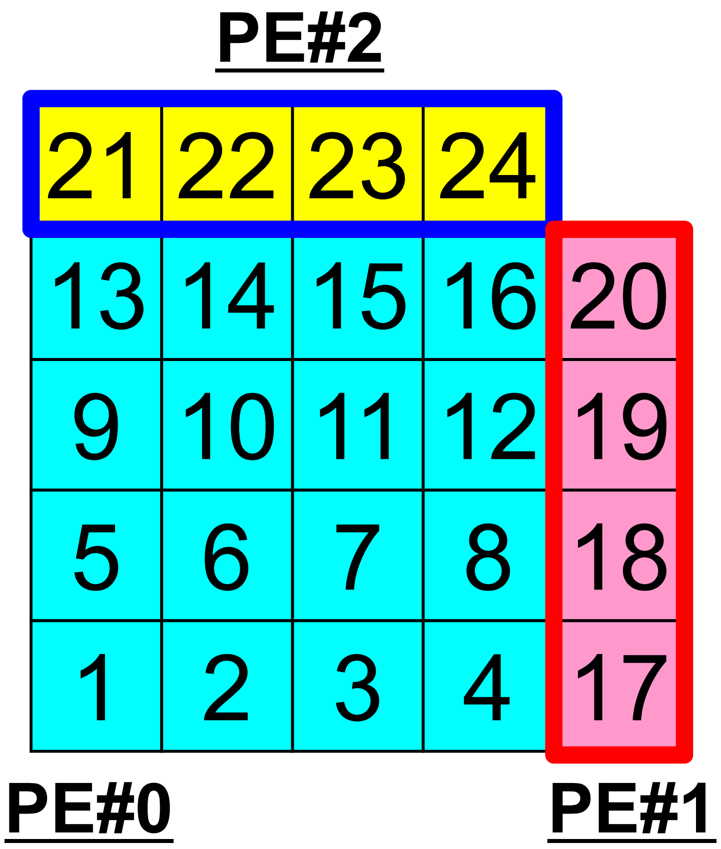
Reading distributed local data files (sqm.*)

```
/*
!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

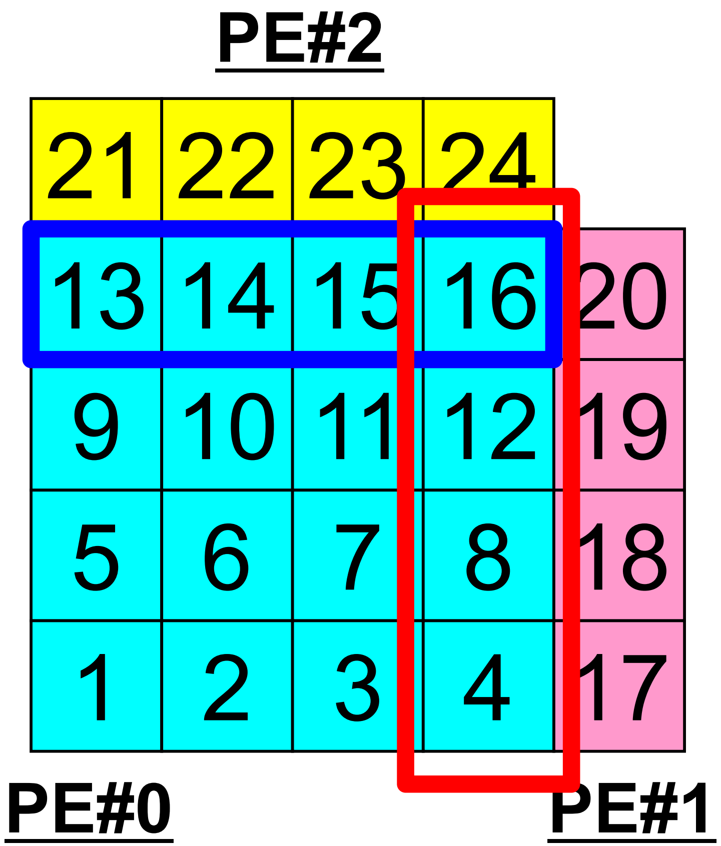
Reading distributed local data files (sqm.*)

```
/*
!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
```



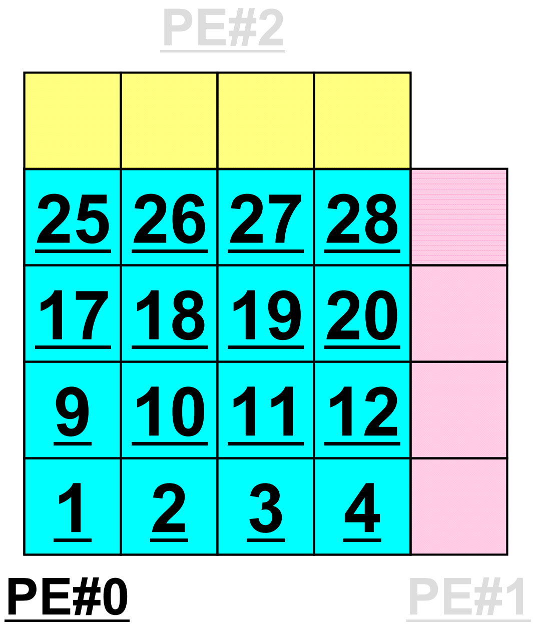
Example: sq-sr1.c (3/6)

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)

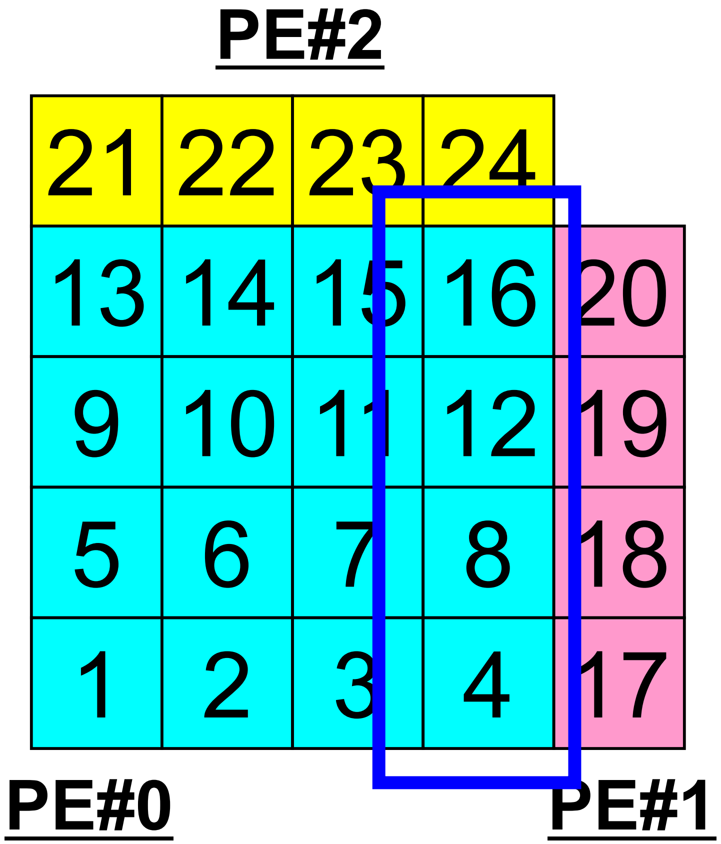
Preparation of sending/receiving buffers

```
/*
!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 ...

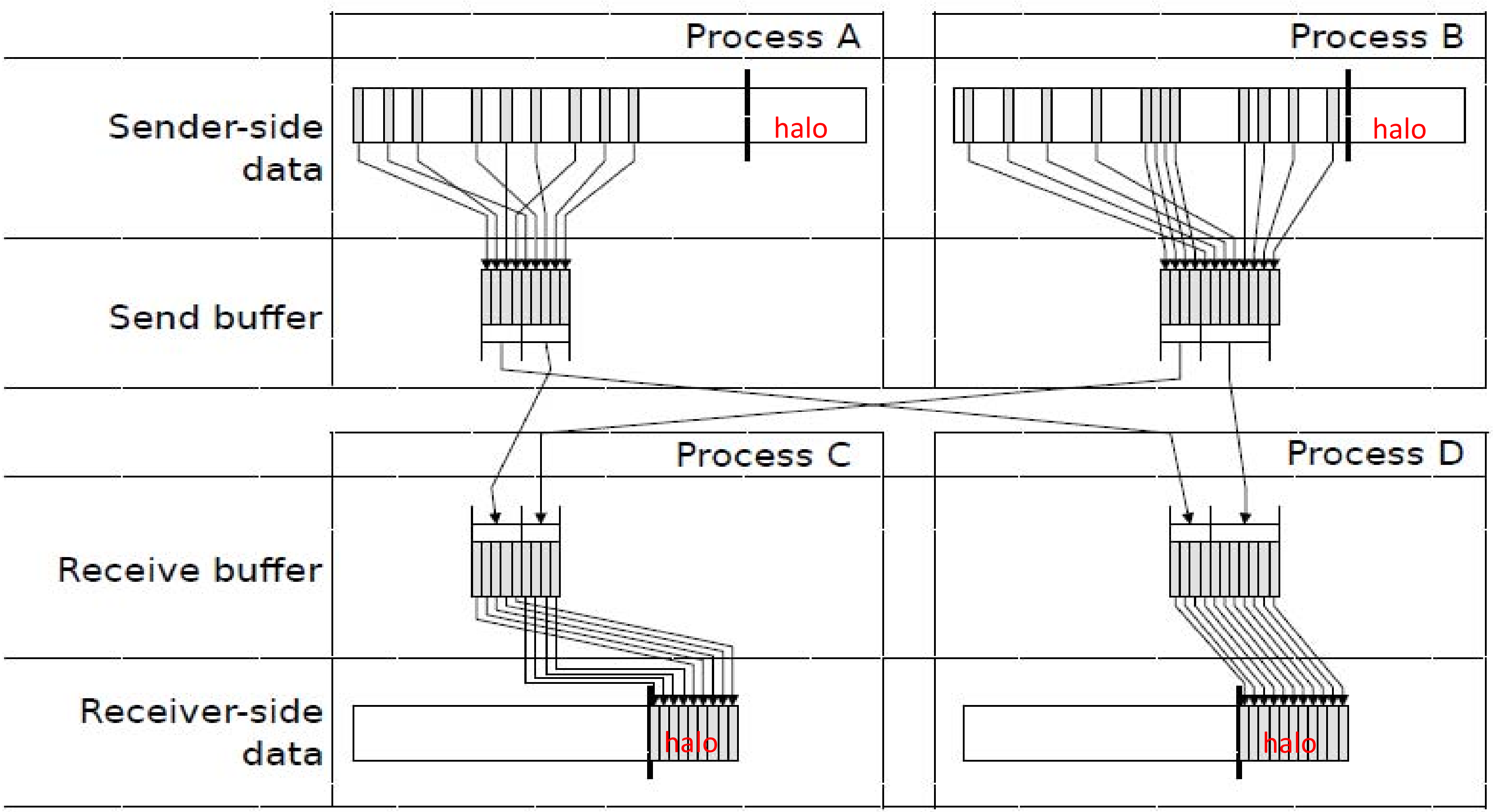
```
for (neib=0; neib<NeibPETot; neib++){
   tag= 0;
   iS_e= export_index[neib];
   iE_e= export_index[neib+1];
   BUFlength_e= iE_e - iS_e

   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

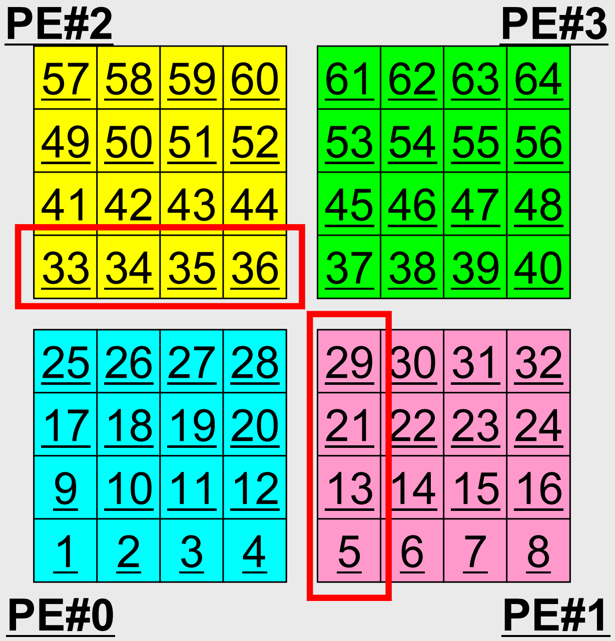


Dr. Osni Marques
（Lawrence Berkeley National Laboratory）

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

SEND/Export: MPI_Isend

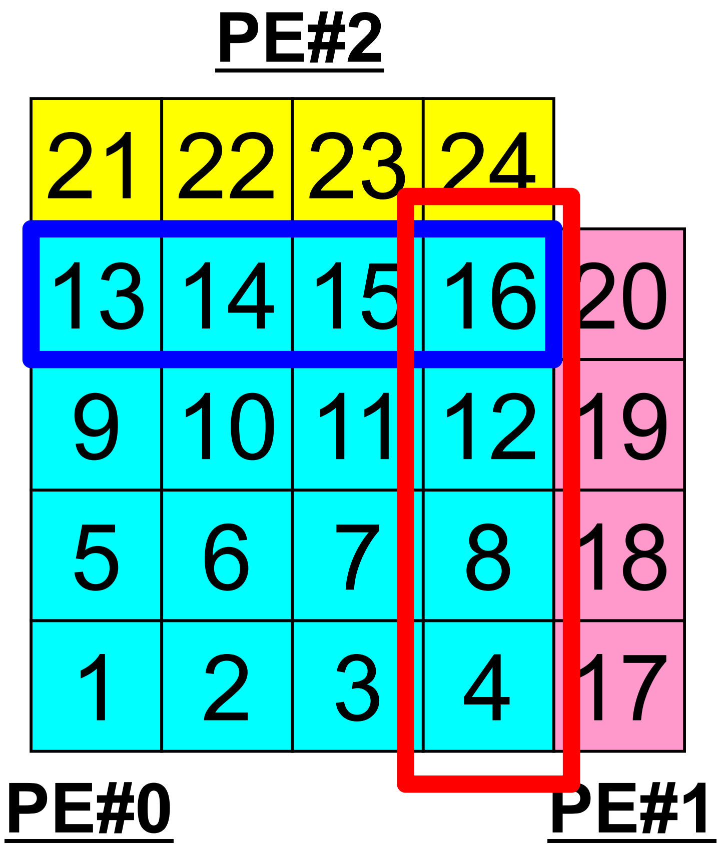
```
/*
!C
!C +-----------+
!C | SEND-RECV |
!C +-----------+
!C===*/
        StatSend = malloc(sizeof(MPI_Status) * NeibPeTot);
        StatRecv = malloc(sizeof(MPI_Status) * NeibPeTot);
        RequestSend = malloc(sizeof(MPI_Request) * NeibPeTot);
        RequestRecv = malloc(sizeof(MPI_Request) *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



PE#2							PE#3
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
PE#0							PE#1

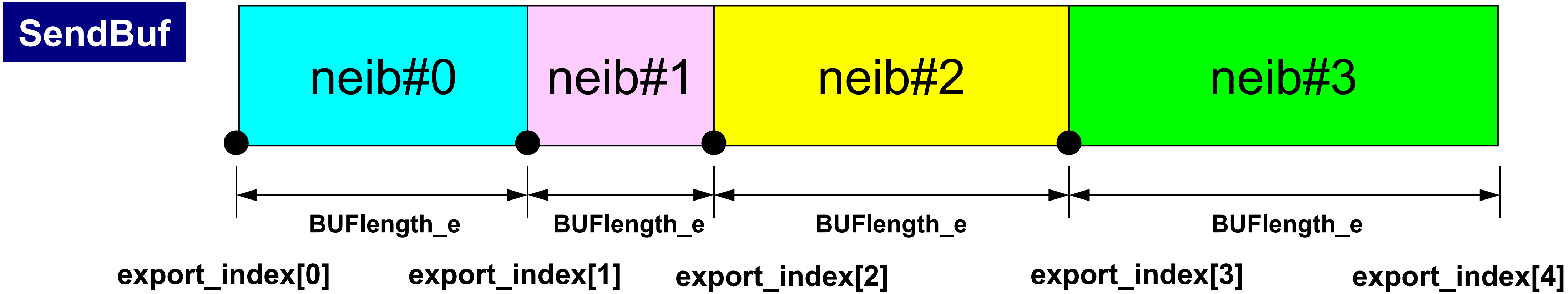
SEND/Export: PE#0

```
#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

MPI_Waitall

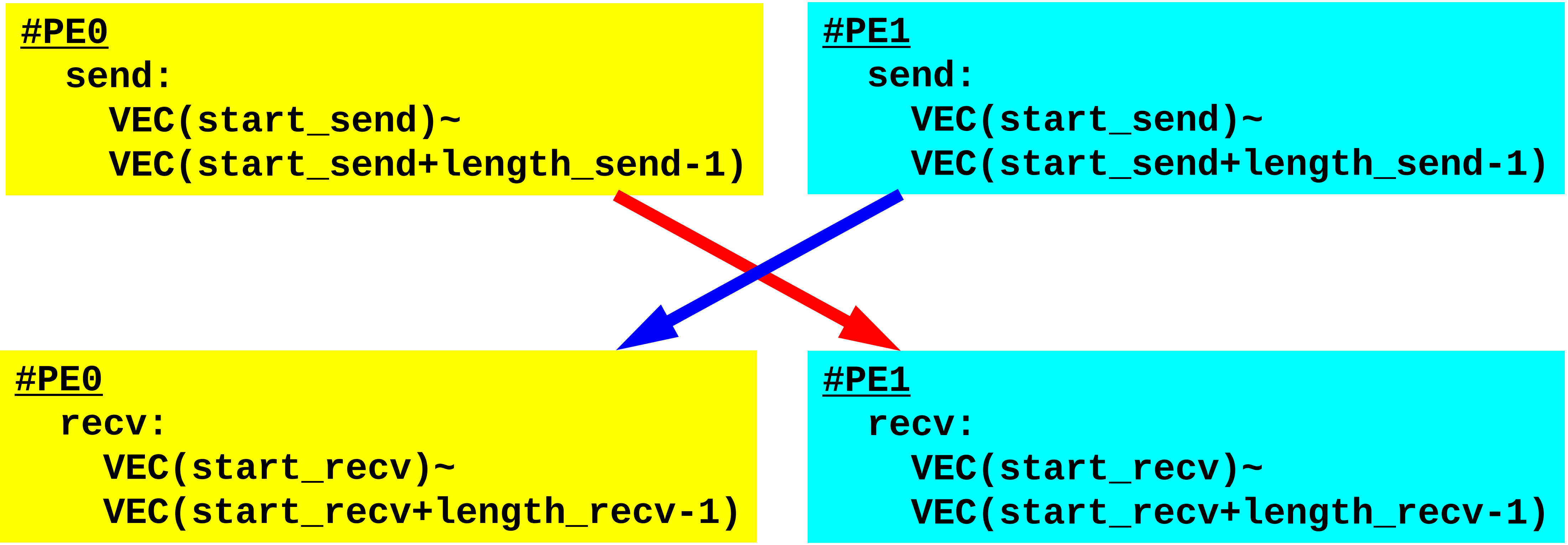
C

- **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'

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

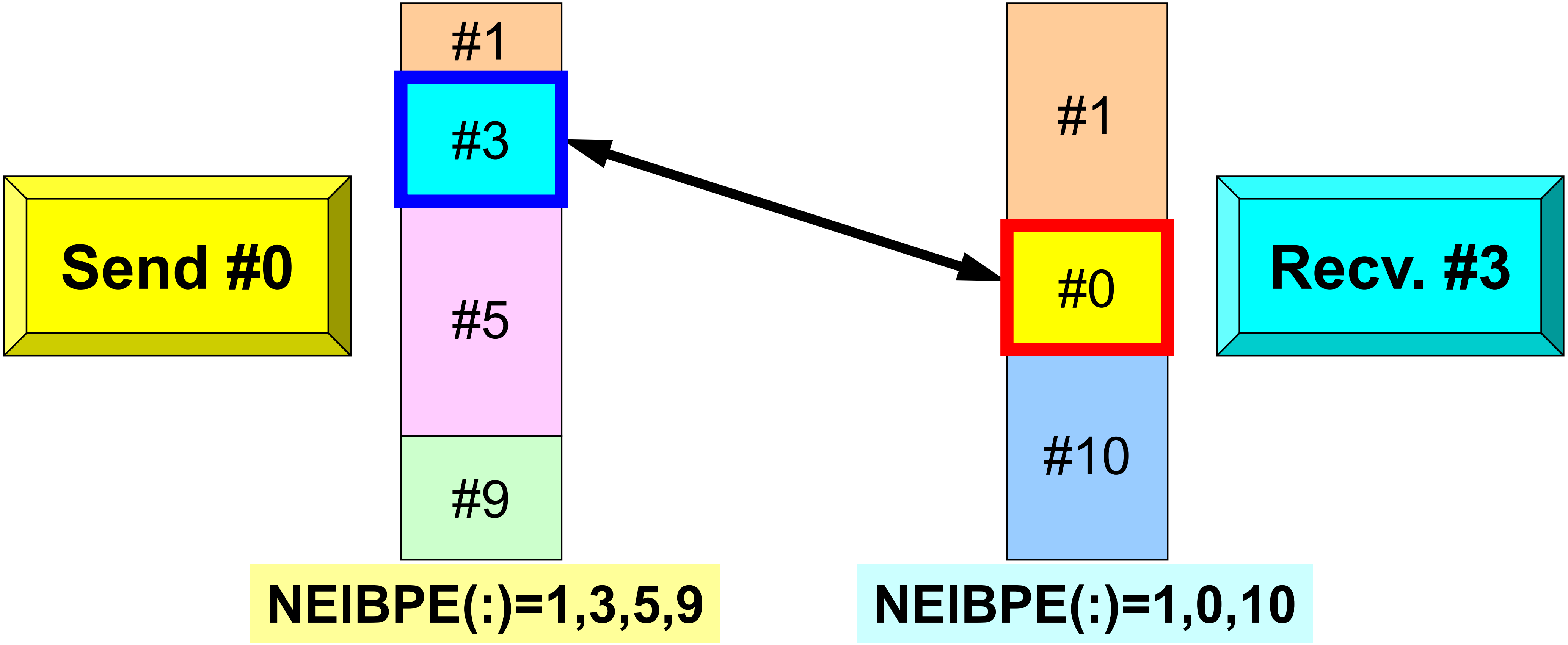
Relationship SEND/RECV

```
  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)



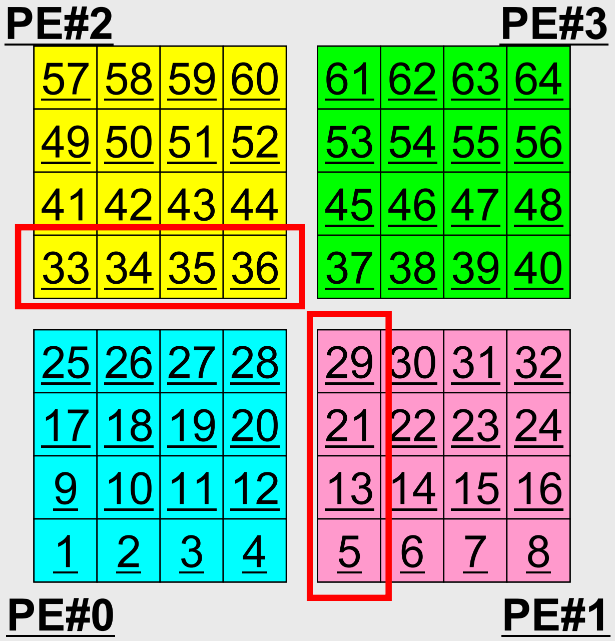
- Consistency of ID’s of sources/destinations, size and contents of messages !
- Communication occurs when NEIBPE(neib) matches

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

C

RECV/Import: MPI_Irecv

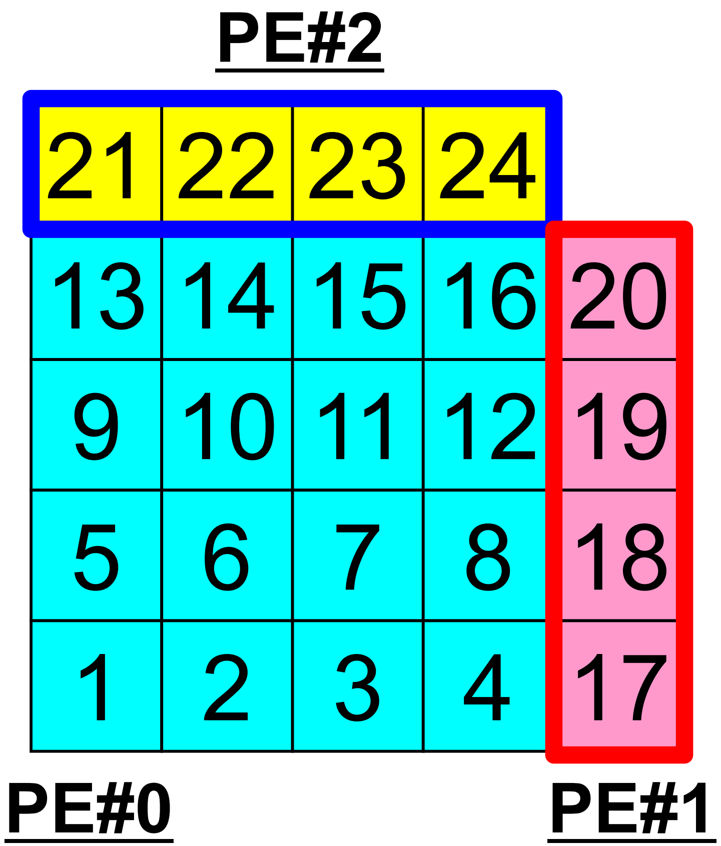
```
/*
!C
!C +-----------+
!C | SEND-RECV |
!C +-----------+
!C===*/
        StatSend = malloc(sizeof(MPI_Status) * NeibPeTot);
        StatRecv = malloc(sizeof(MPI_Status) * NeibPeTot);
        RequestSend = malloc(sizeof(MPI_Request) * NeibPeTot);
        RequestRecv = malloc(sizeof(MPI_Request) *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



PE#2							PE#3
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
PE#0							PE#1

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
```



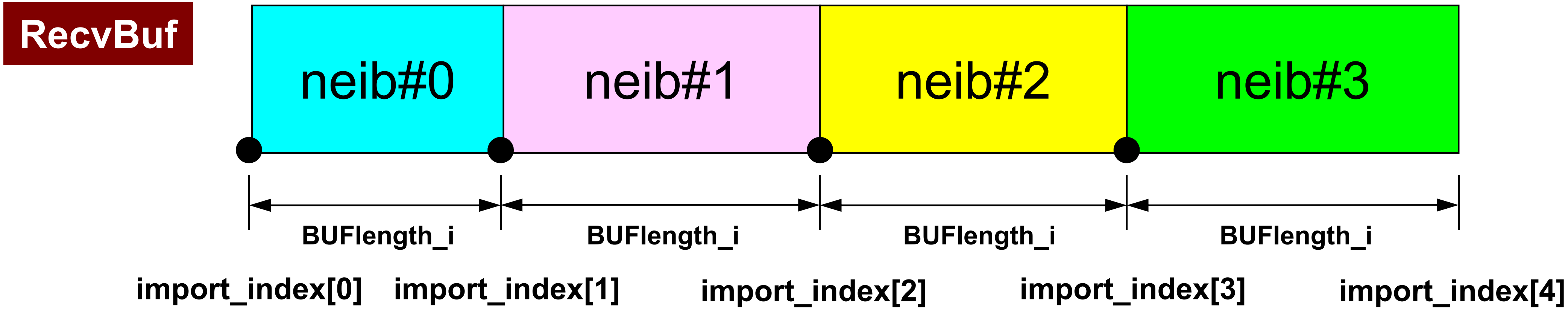
RECV: MPI_Isend/Irecv/Waitall

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



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.

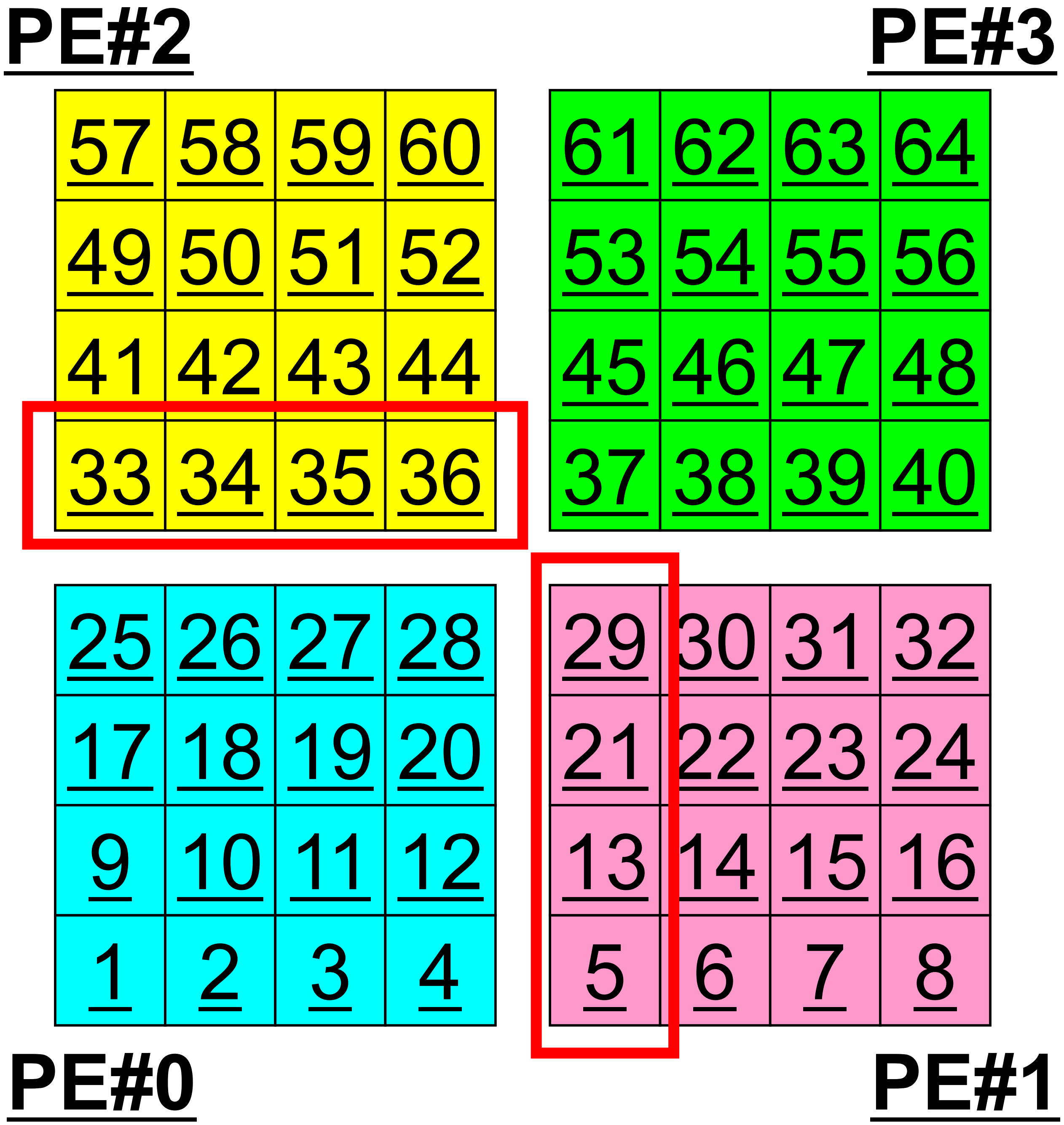
Example: sq-sr1.c (6/6)

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)



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

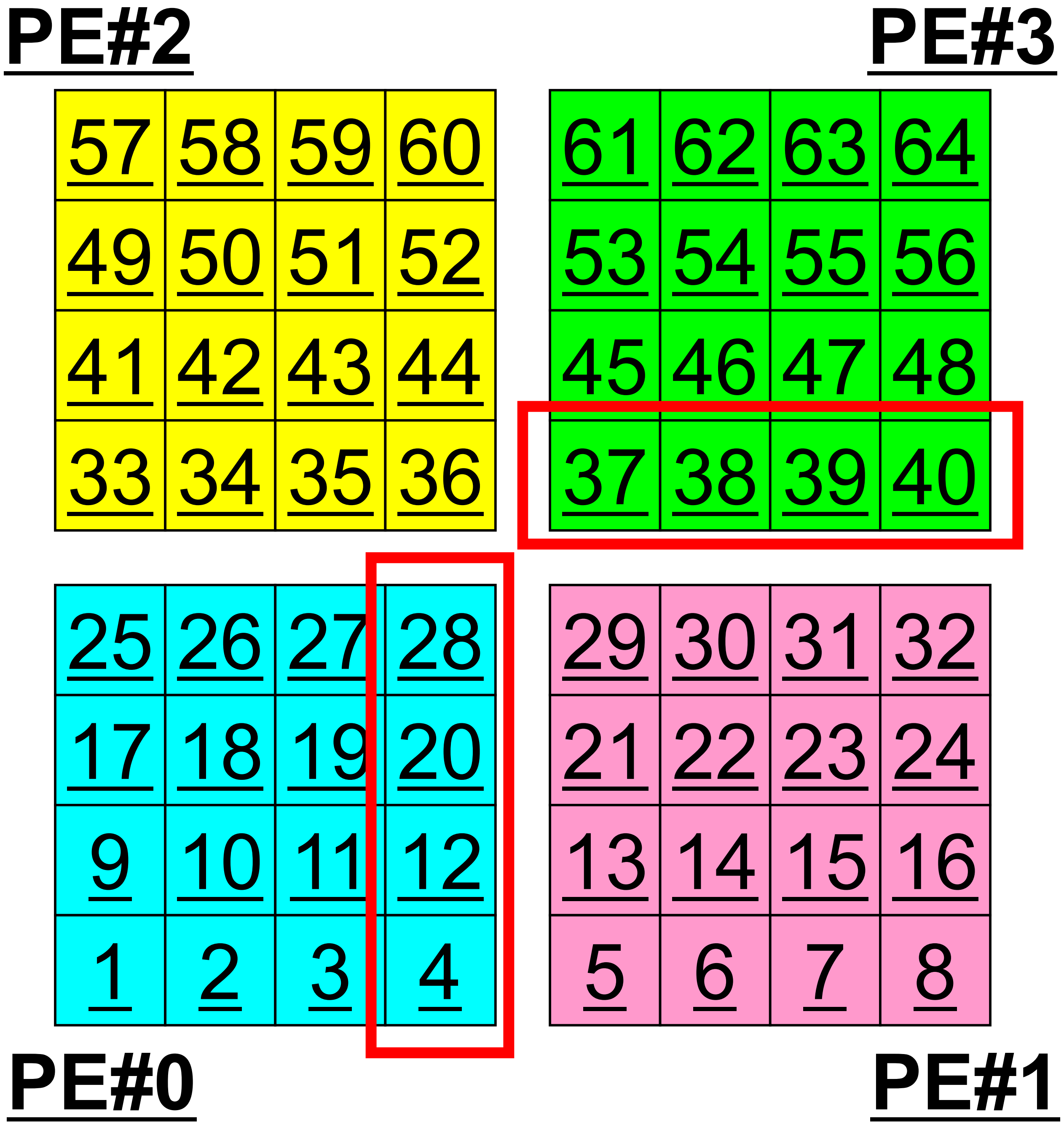
25	26	27	28
17	18	19	20
9	10	11	12
1	2	3	4

PE#1

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

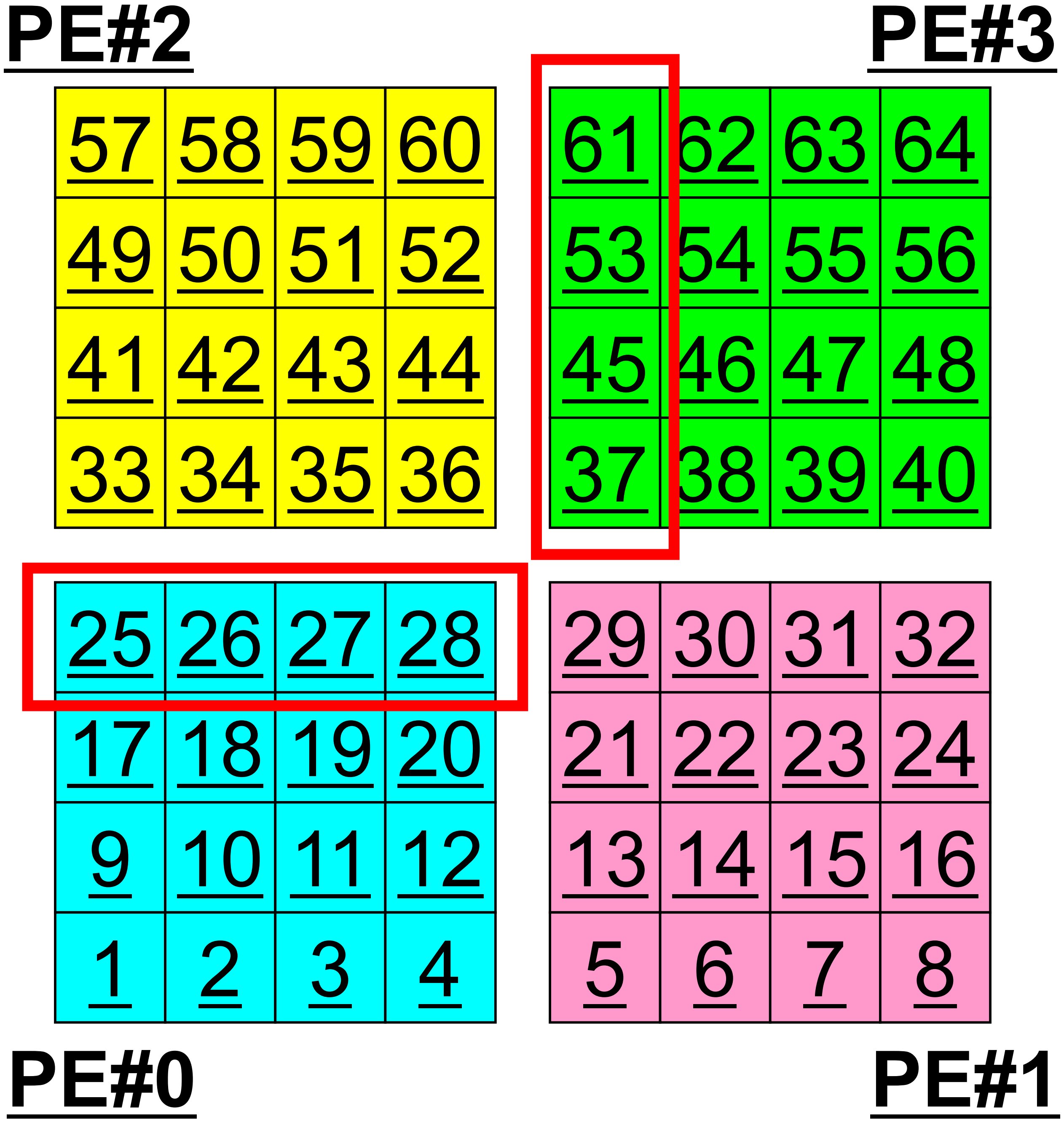
```
RECVbuf          0          1          5
RECVbuf          0          1         13
RECVbuf          0          1         21
RECVbuf          0          1         29
RECVbuf          0          2         33
RECVbuf          0          2         34
RECVbuf          0          2         35
RECVbuf  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



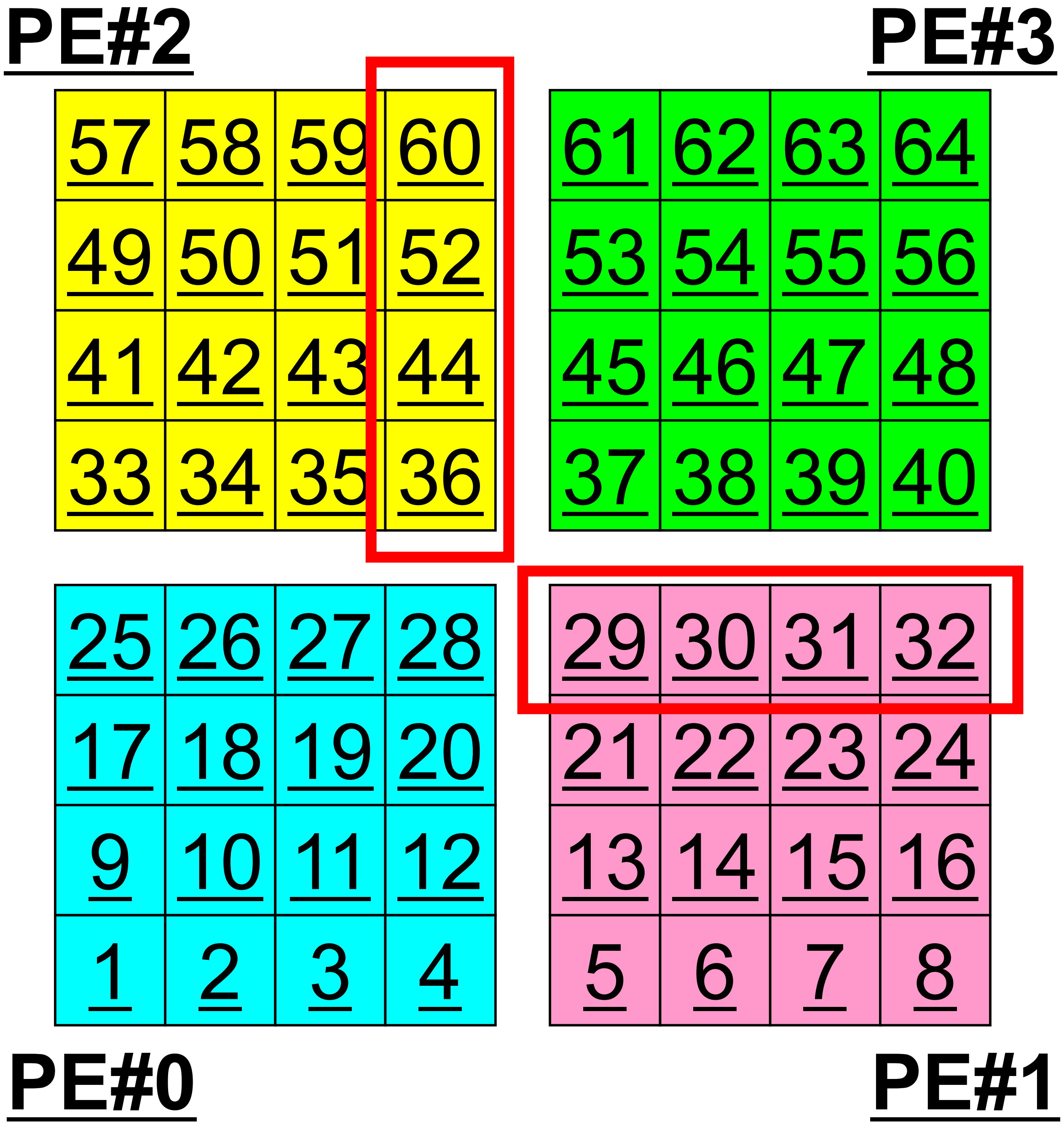
```
RECVbuf        0        1        5
RECVbuf        0        1       13
RECVbuf        0        1       21
RECVbuf        0        1       29
RECVbuf        0        2       33
RECVbuf        0        2       34
RECVbuf        0        2       35
RECVbuf        0        2       36

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)



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

```
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
```

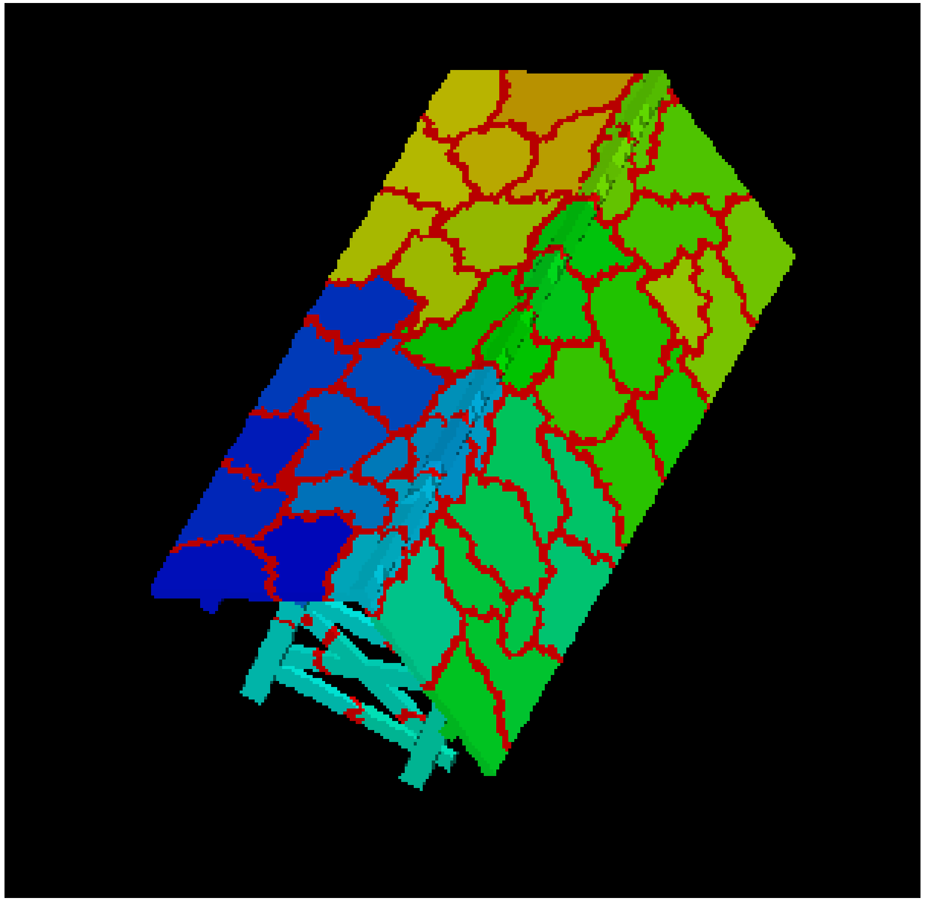
Distributed Local Data Structure for Parallel Computation

- Distributed local data structure for domain-to-doain 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

This Idea can be applied to FEM with more complicated geometries.

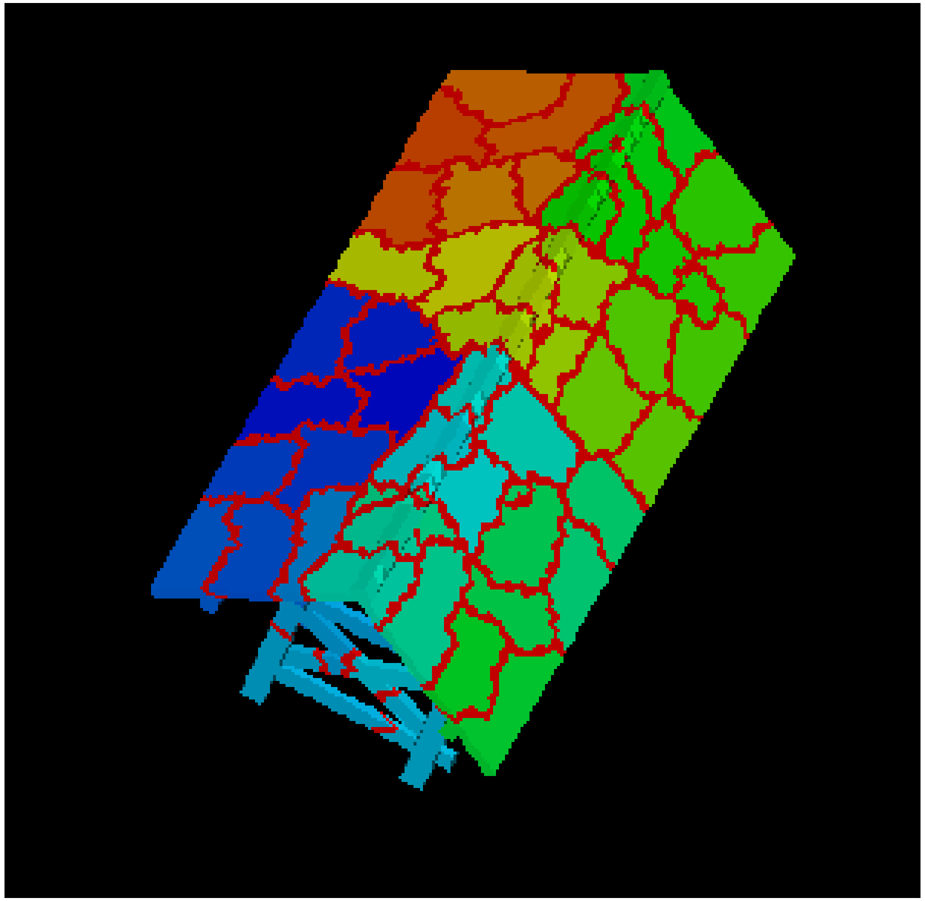
Red Lacquered Gate in 64 Pes, 40,624 elements



k-METIS

Load Balance= 1.03

edgecut = 7,563



p-METIS

Load Balance= 1.00

edgecut = 7,738

Exercise

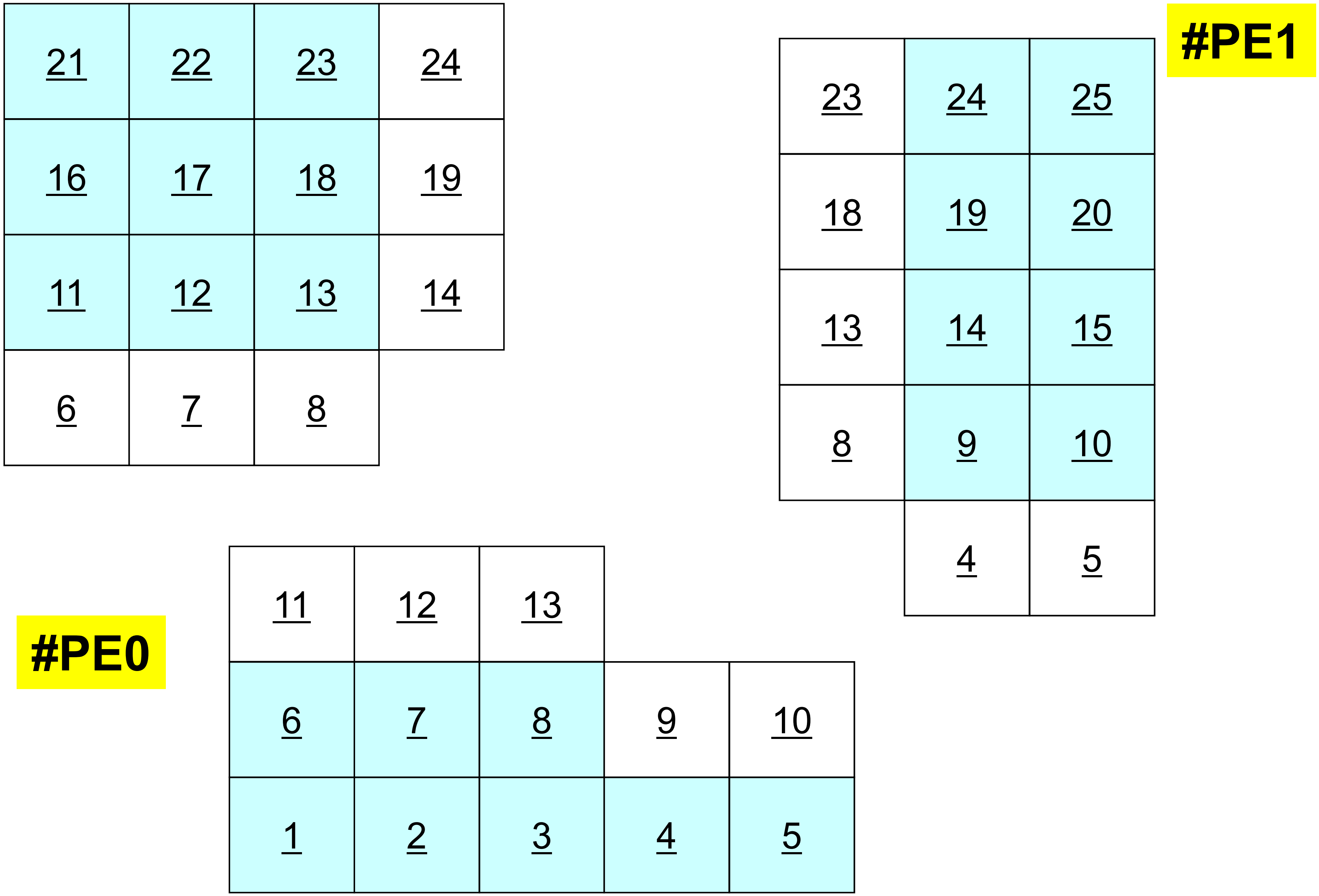
Initial Mesh

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

Exercise

#PE2



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

#PE1

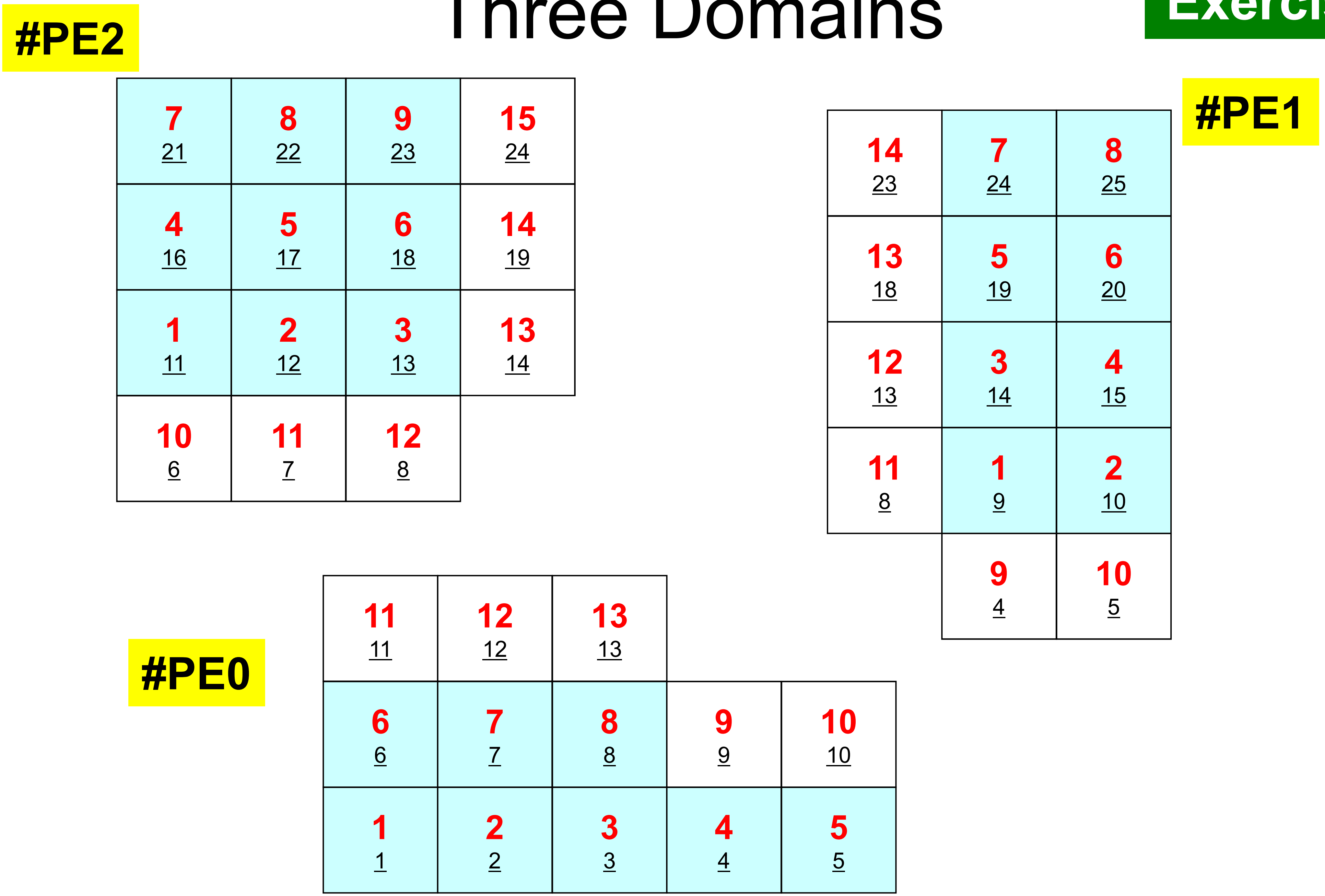
23	24	25
18	19	20
13	14	15
8	9	10
	4	5

#PE0

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

Three Domains

Exercise



#PE2

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

#PE1

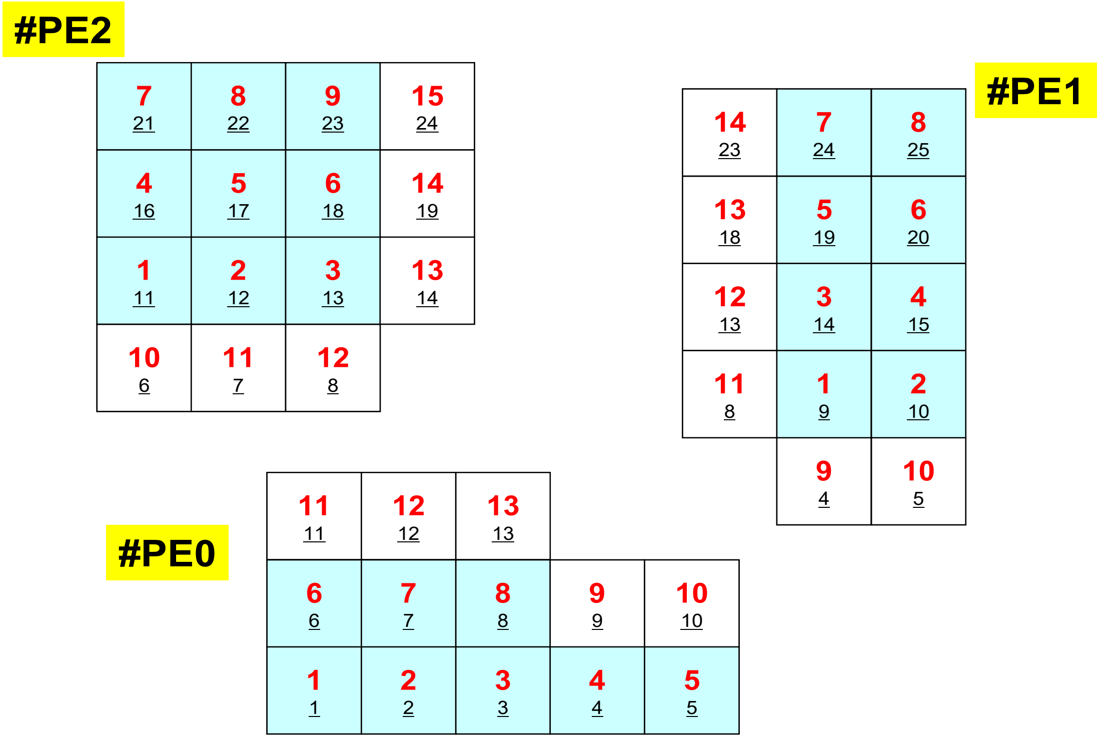
14 23	7 24	8 25
13 18	5 19	6 20
12 13	3 14	4 15
11 8	1 9	2 10
	9 4	10 5

#PE0

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

PE#0: sqm.0: fill ○'s

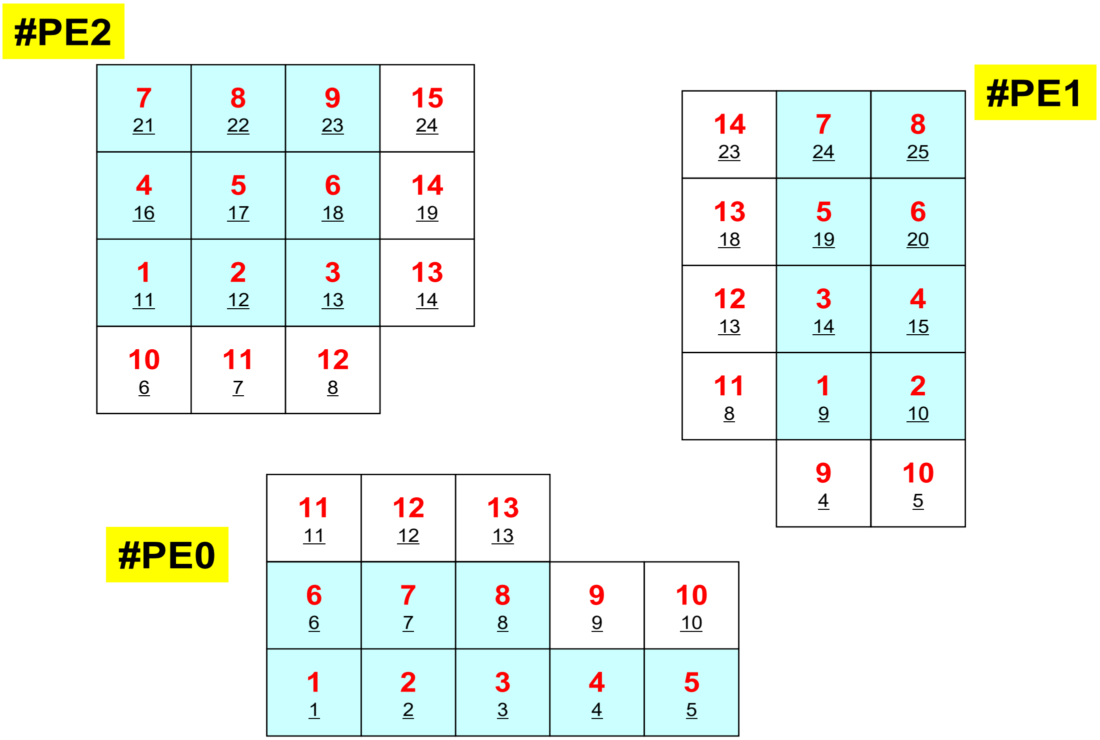
Exercise



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

Exercise

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

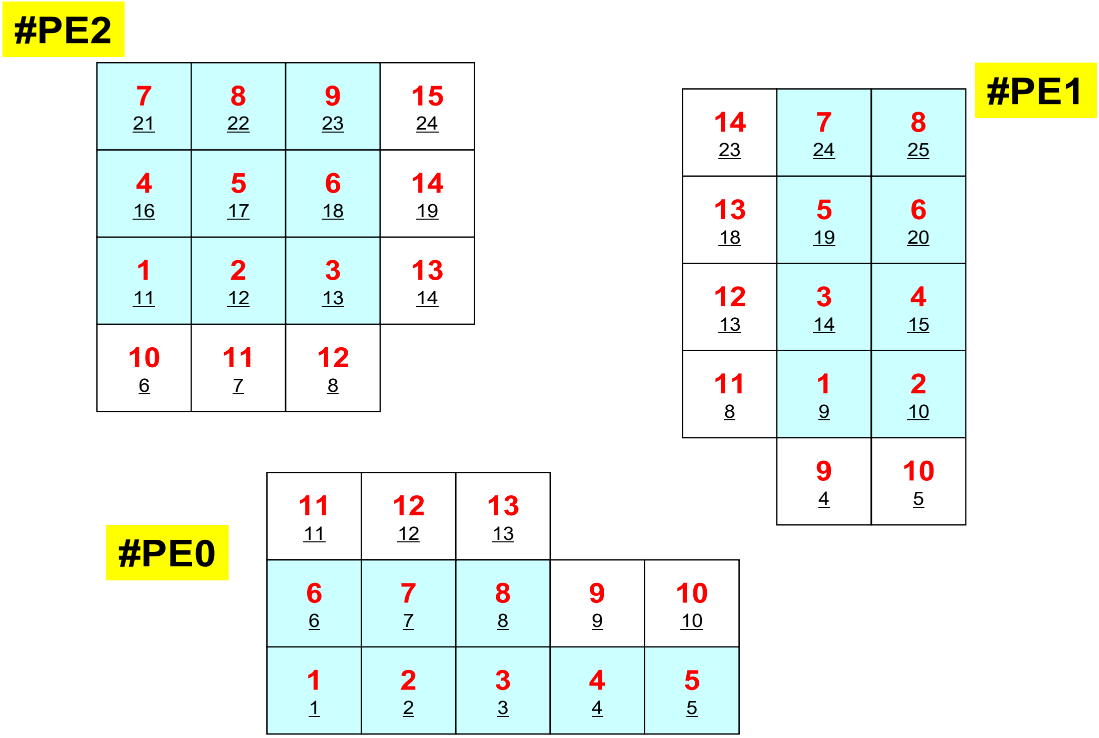
#PE0

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
    8   14 (int+ext, int pts)
#IMPORTindex
    ○    ○
#IMPORTitems
    ○…
#EXPORTindex
    ○    ○
#EXPORTitems
    ○…
```

Exercise

PE#2: sqm.2: fill ○'s



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

Exercise

#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

Exercise

- 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.c) to <$O-S2>/ex
- pjsub go3.sh