

Introduction to Parallel FEM in Fortran Parallel Data Structure

Kengo Nakajima
RIKEN R-CCS

Parallel Computing

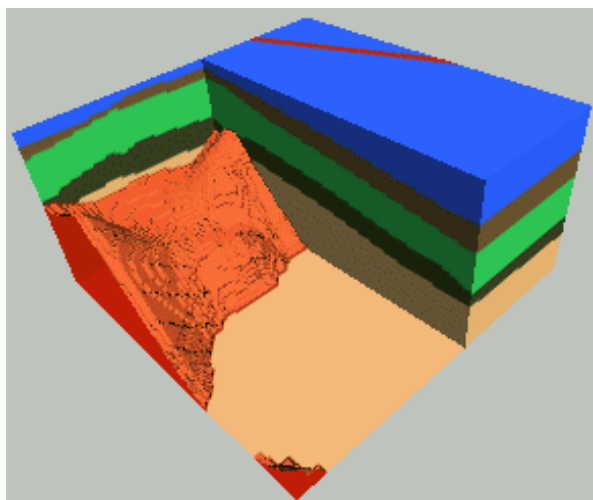
- Faster, Larger & More Complicated
- Scalability
 - Solving N^x scale problem using N^x computational resources during same computation time
 - for large-scale problems: **Weak Scaling**
 - e.g. CG solver: more iterations needed for larger problems
 - Solving a problem using N^x computational resources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

What is Parallel Computing ? (1/2)

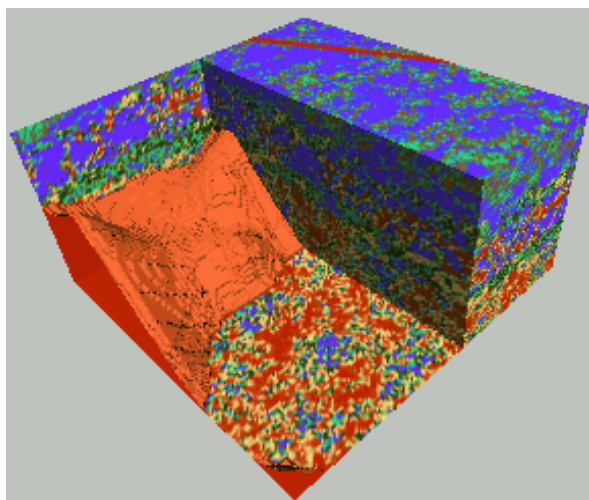
- to solve larger problems faster

Homogeneous/Heterogeneous Porous Media

Lawrence Livermore National Laboratory



Homogeneous

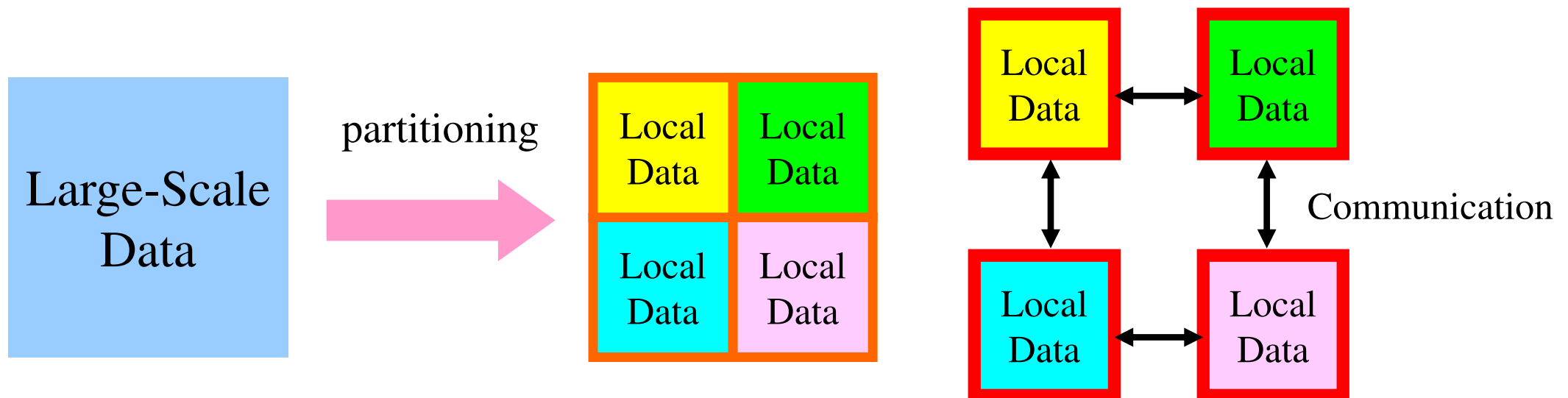


Heterogeneous

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



What is Communication ?

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

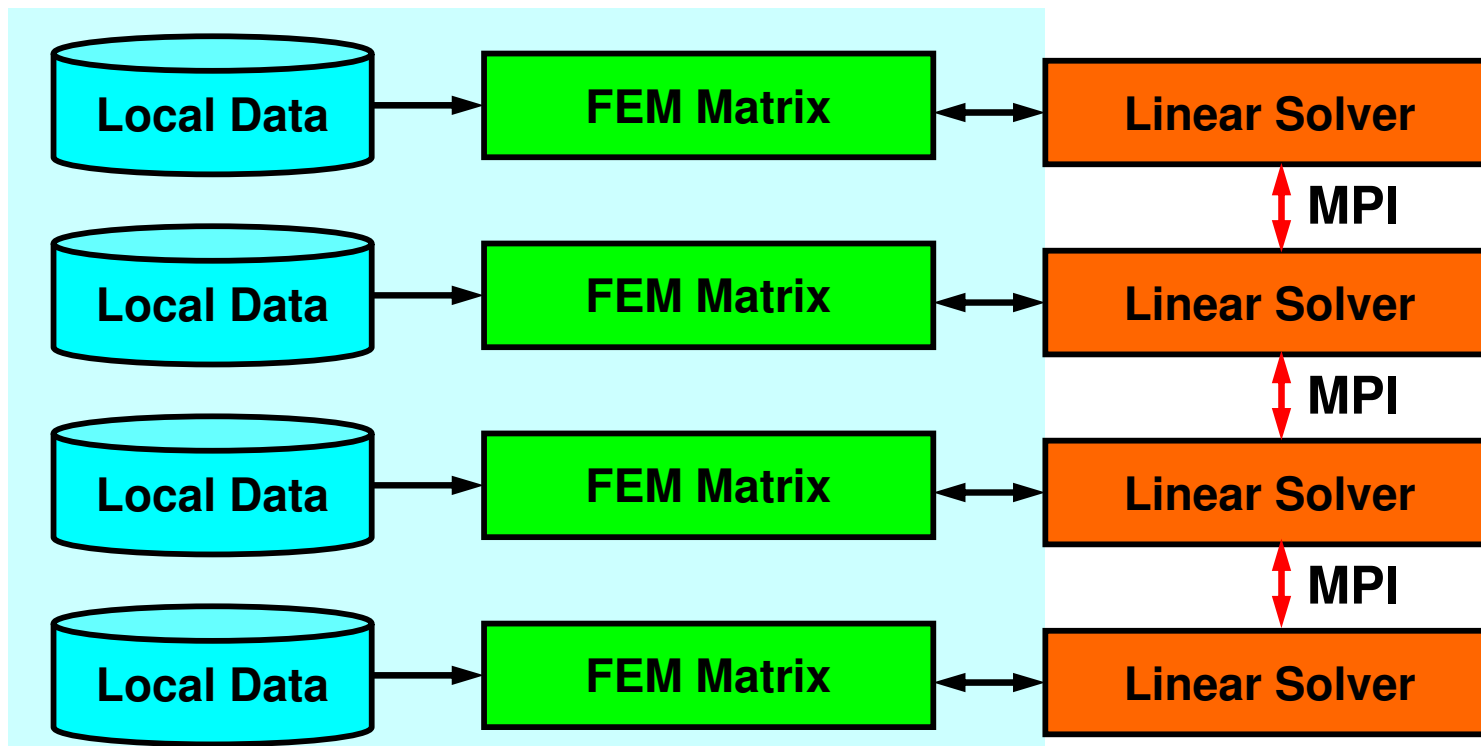
Operations in Parallel FEM

SPMD: Single-Program Multiple-Data

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

FEM code can assemble 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

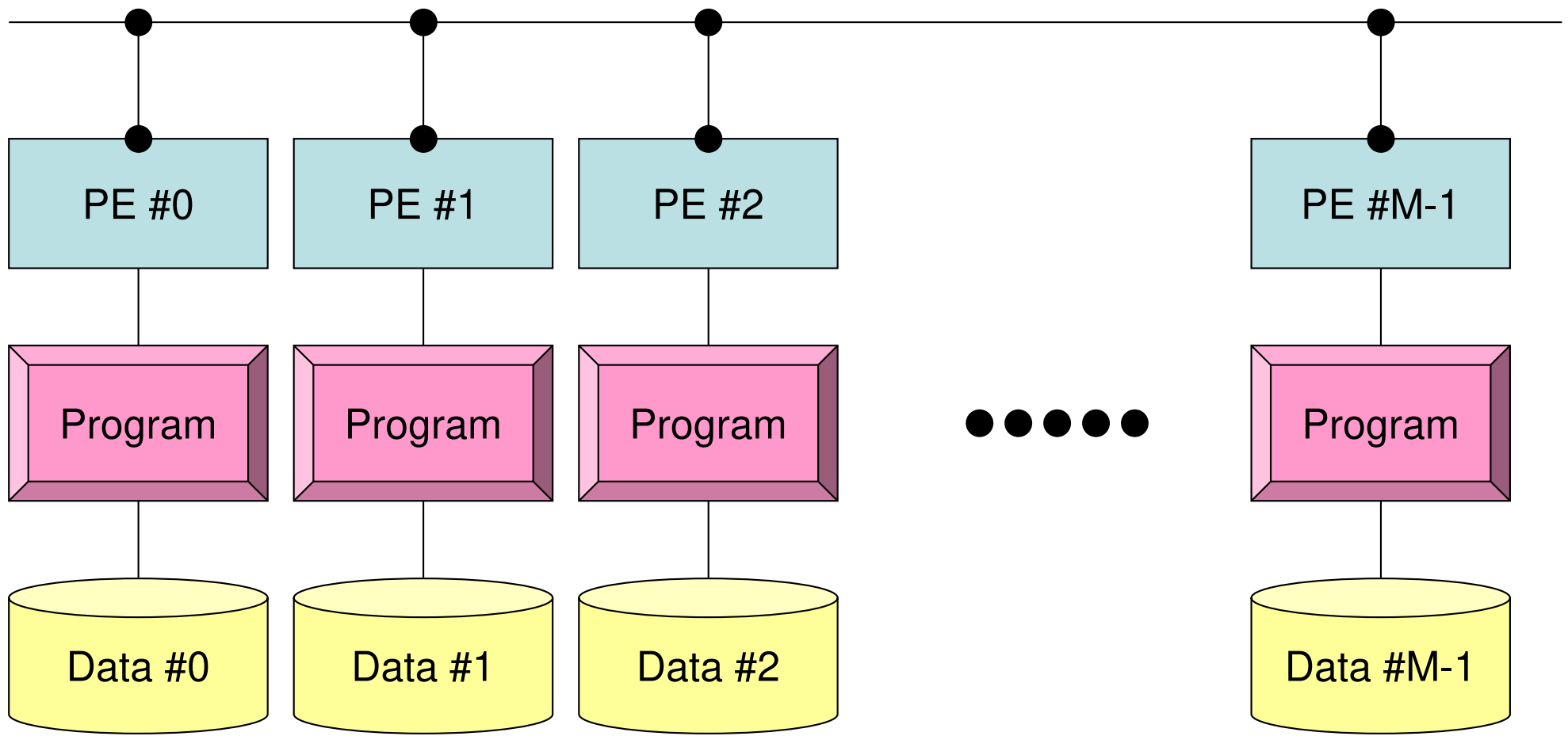


PE: Processing Element
 Processor, Domain, Process

SPMD

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

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



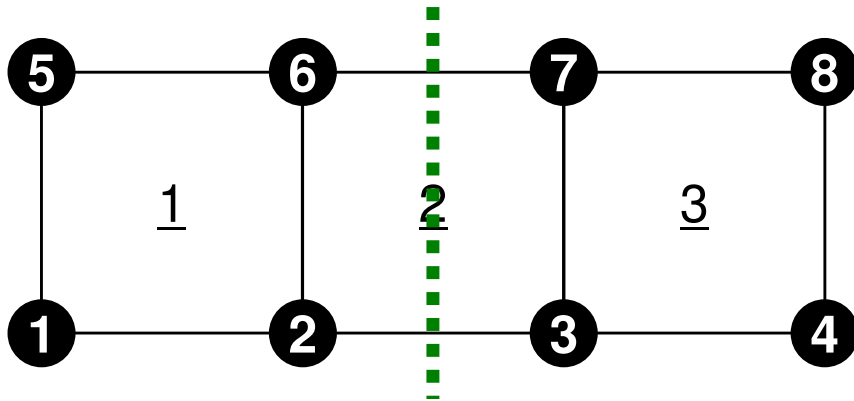
Each process does same operation for different data
 Large-scale data is decomposed, and each part is computed by each process
 It is ideal that parallel program is not different from serial one except communication.

Parallel FEM Procedures

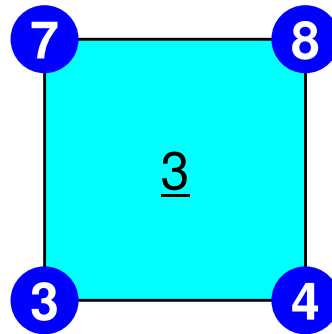
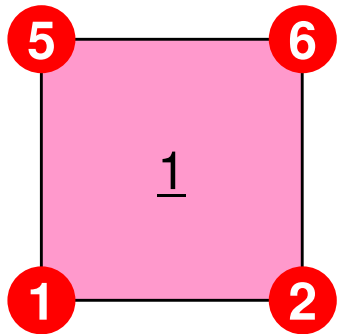
- Design on “Local Data Structure” is important
 - for SPMD-type operations in the previous page
- Matrix Generation
- Preconditioned Iterative Solvers for Linear Equations

Bi-Linear Square Elements

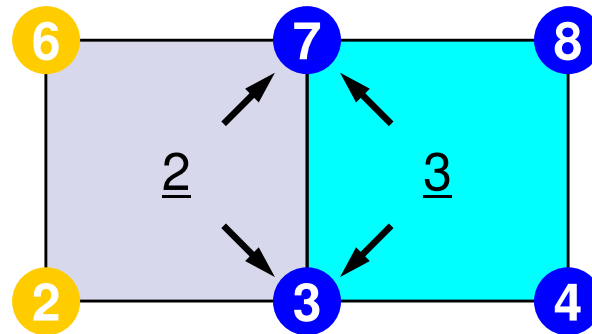
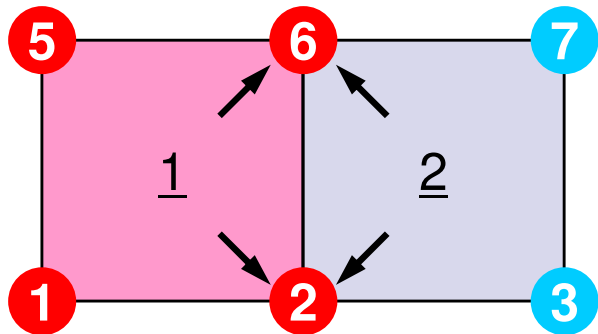
Values are defined on each node



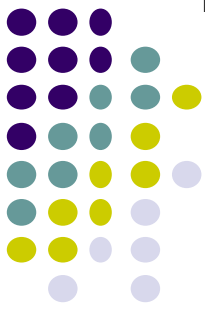
divide into two domains by “node-based” manner, where number of “nodes (vertices)” are balanced.



Local information is not enough for matrix assembling.



Information of overlapped elements and connected nodes are required for matrix assembling on boundary nodes.

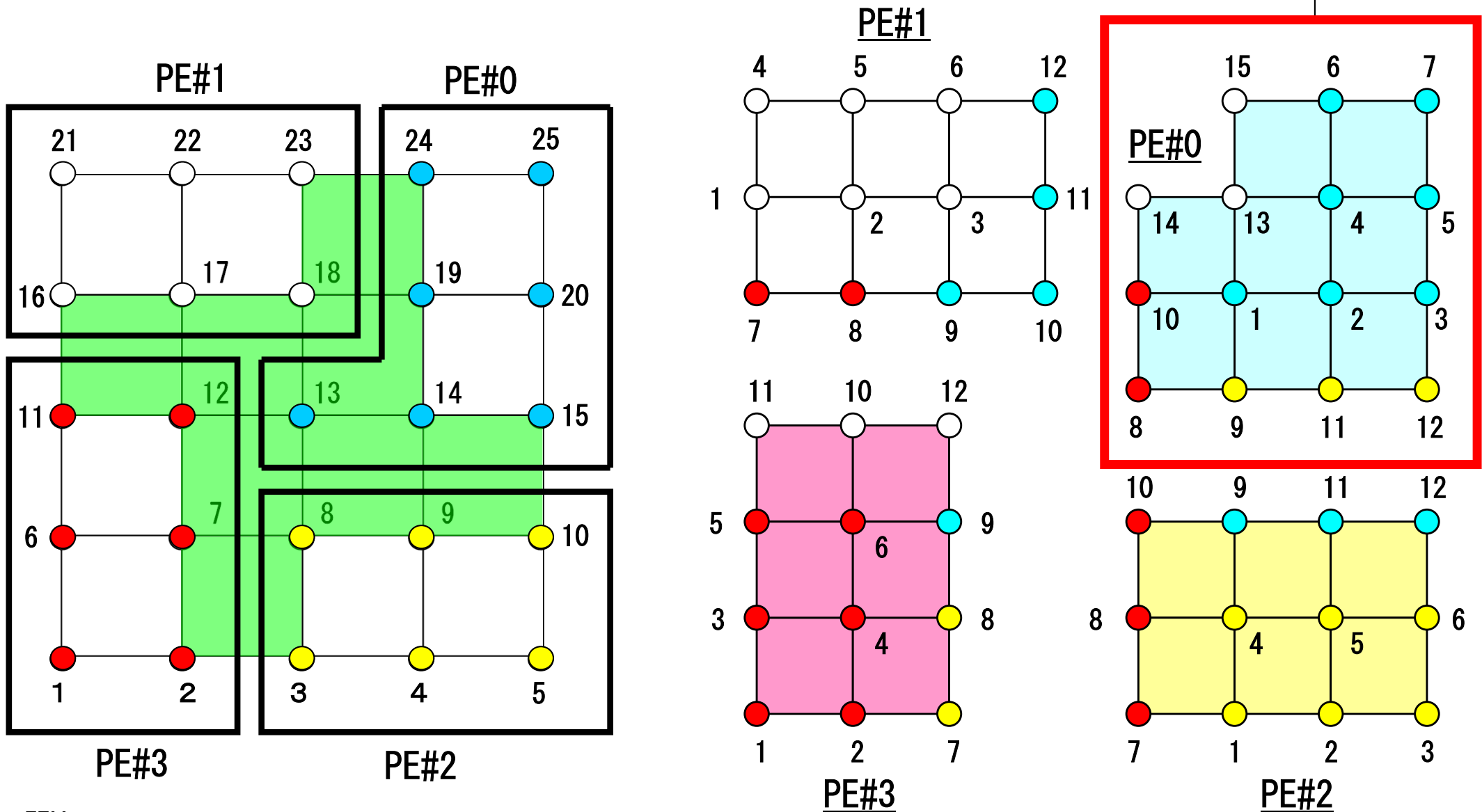


Local Data of Parallel FEM

- **Node-based partitioning for preconditioned iterative solvers**
- Local data includes information for :
 - Nodes originally assigned to the partition/PE
 - Elements which include the nodes (originally assigned to the Partition/PE)
 - All nodes which form the elements but out of the partition
- Nodes are classified into the following 3 categories from the viewpoint of the message passing
 - **Internal nodes** originally assigned nodes
 - **External nodes** in the overlapped elements but out of the partition
 - **Boundary nodes** *external nodes* of other partition (part of internal nodes)
- Communication table between partitions
- NO global information required except partition-to-partition connectivity

Node-based Partitioning

internal nodes - elements - external nodes

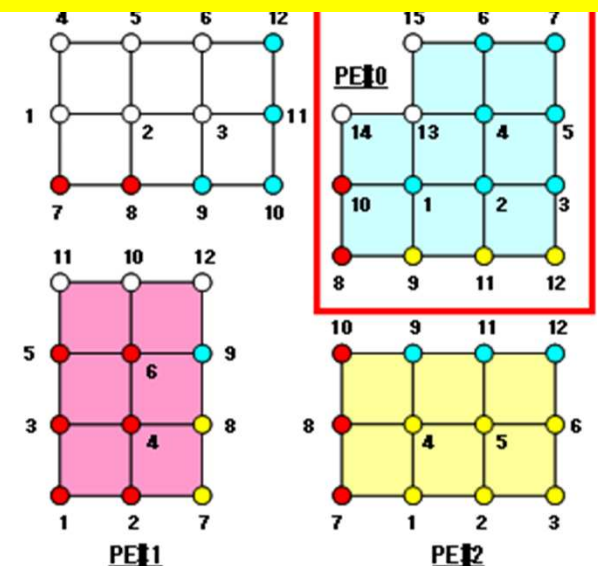
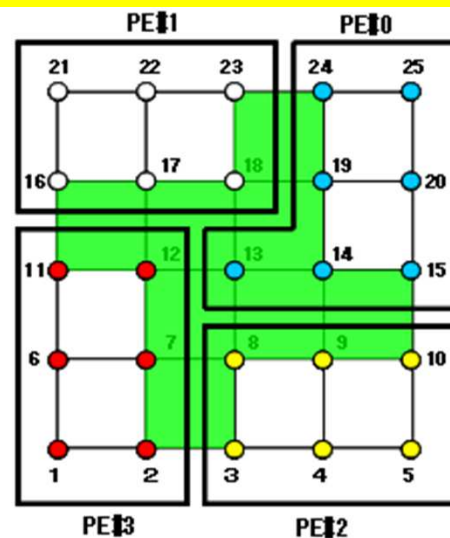
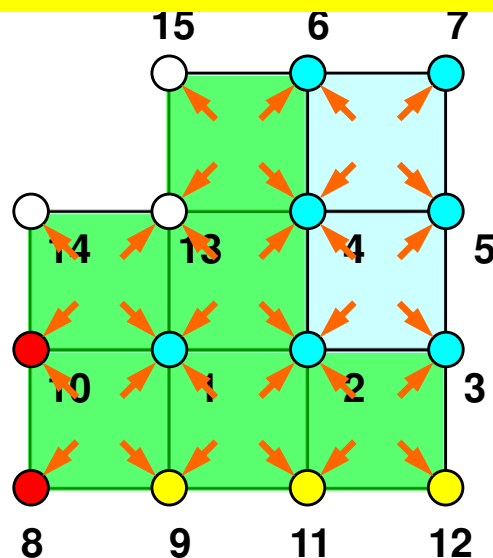




Node-based Partitioning

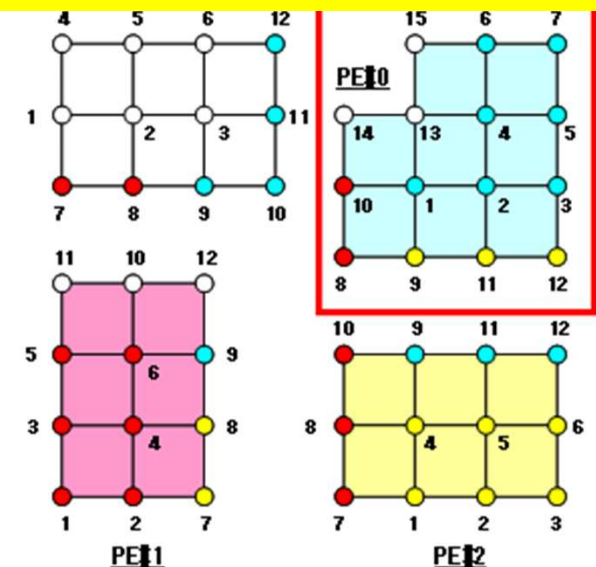
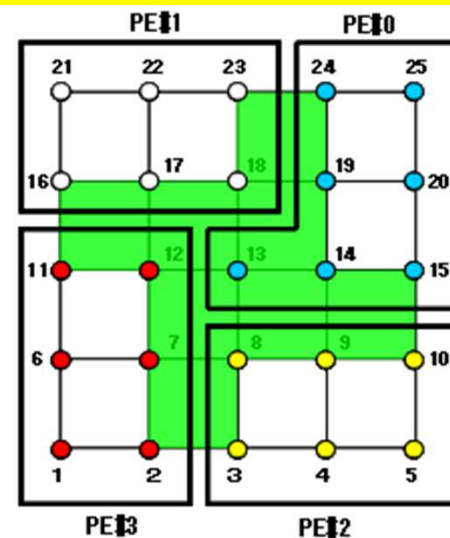
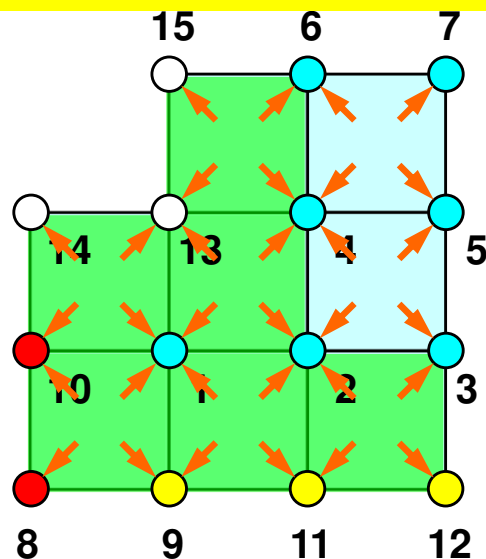
internal nodes - elements - external nodes

- Partitioned nodes themselves (Internal Nodes) 内点
- Elements which include Internal Nodes 内点を含む要素
- External Nodes included in the Elements 外点
in overlapped region among partitions.
- Info of External Nodes are required for completely local element-based operations on each processor.



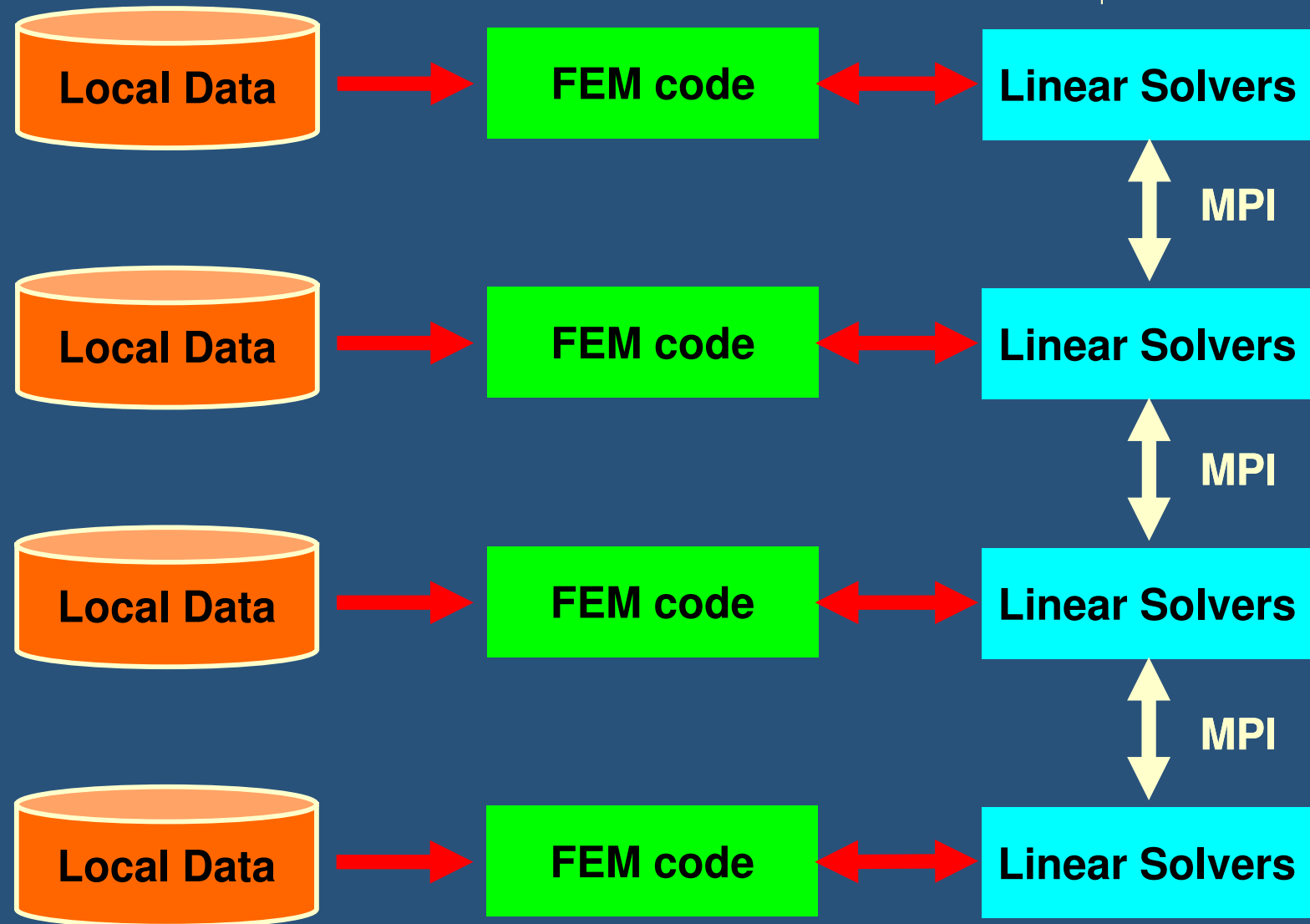
We do not need communication during matrix assemble !!

- Partitioned nodes themselves (Internal Nodes)
- Elements which include Internal Nodes
- External Nodes included in the Elements in overlapped region among partitions.
- Info of External Nodes are required for completely local element-based operations on each processor.



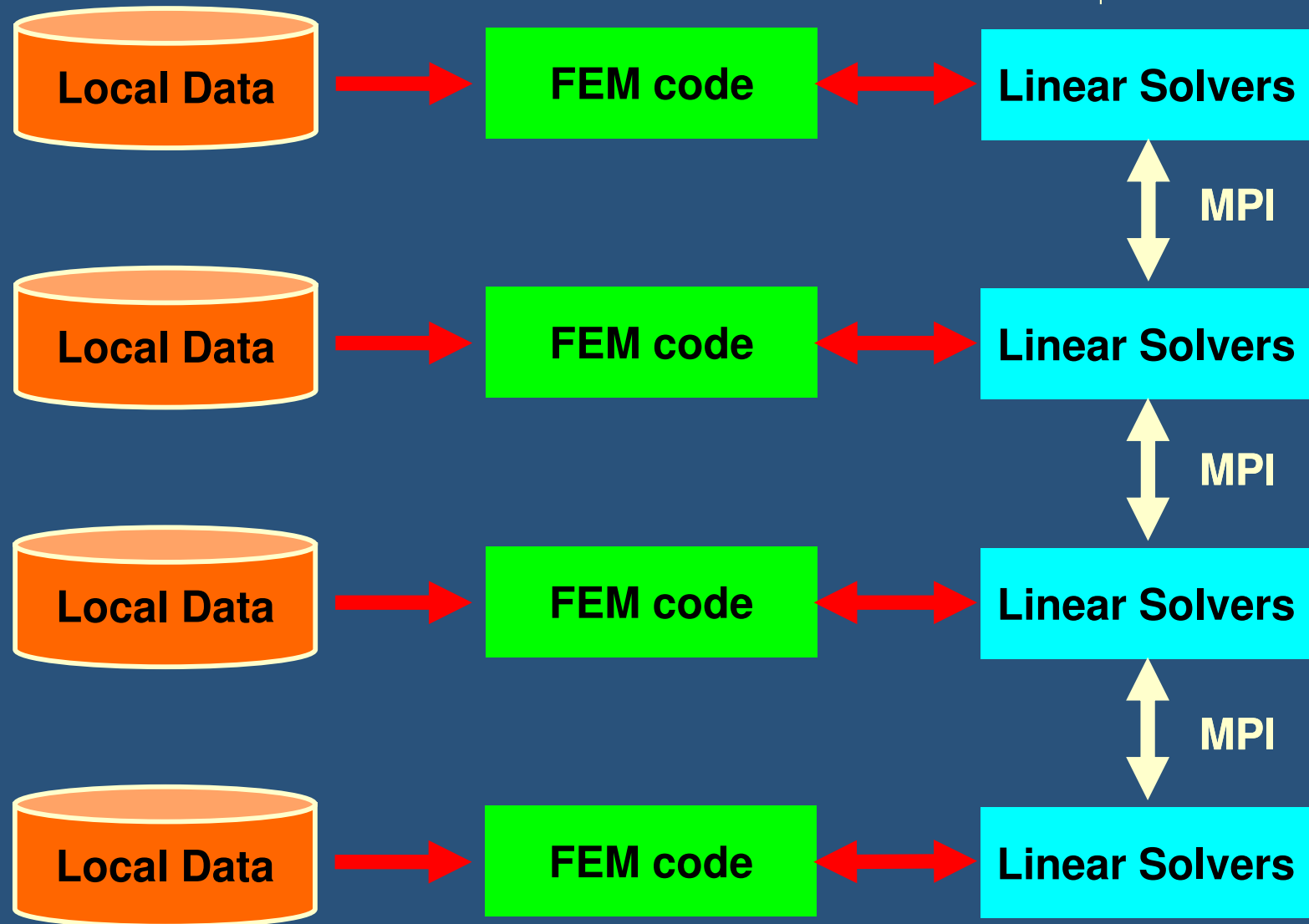
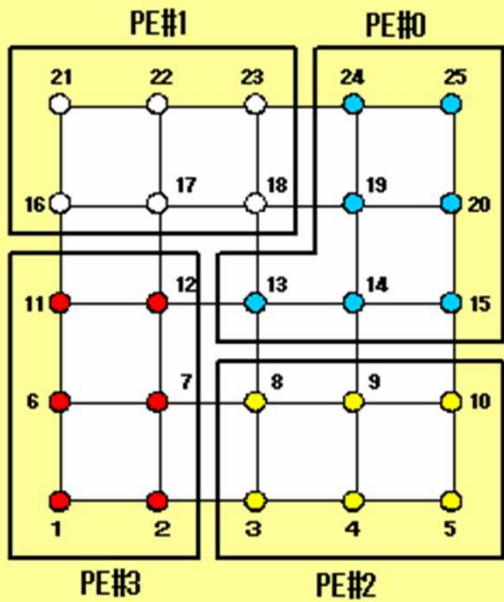
Parallel Computing in FEM

SPMD: Single-Program Multiple-Data



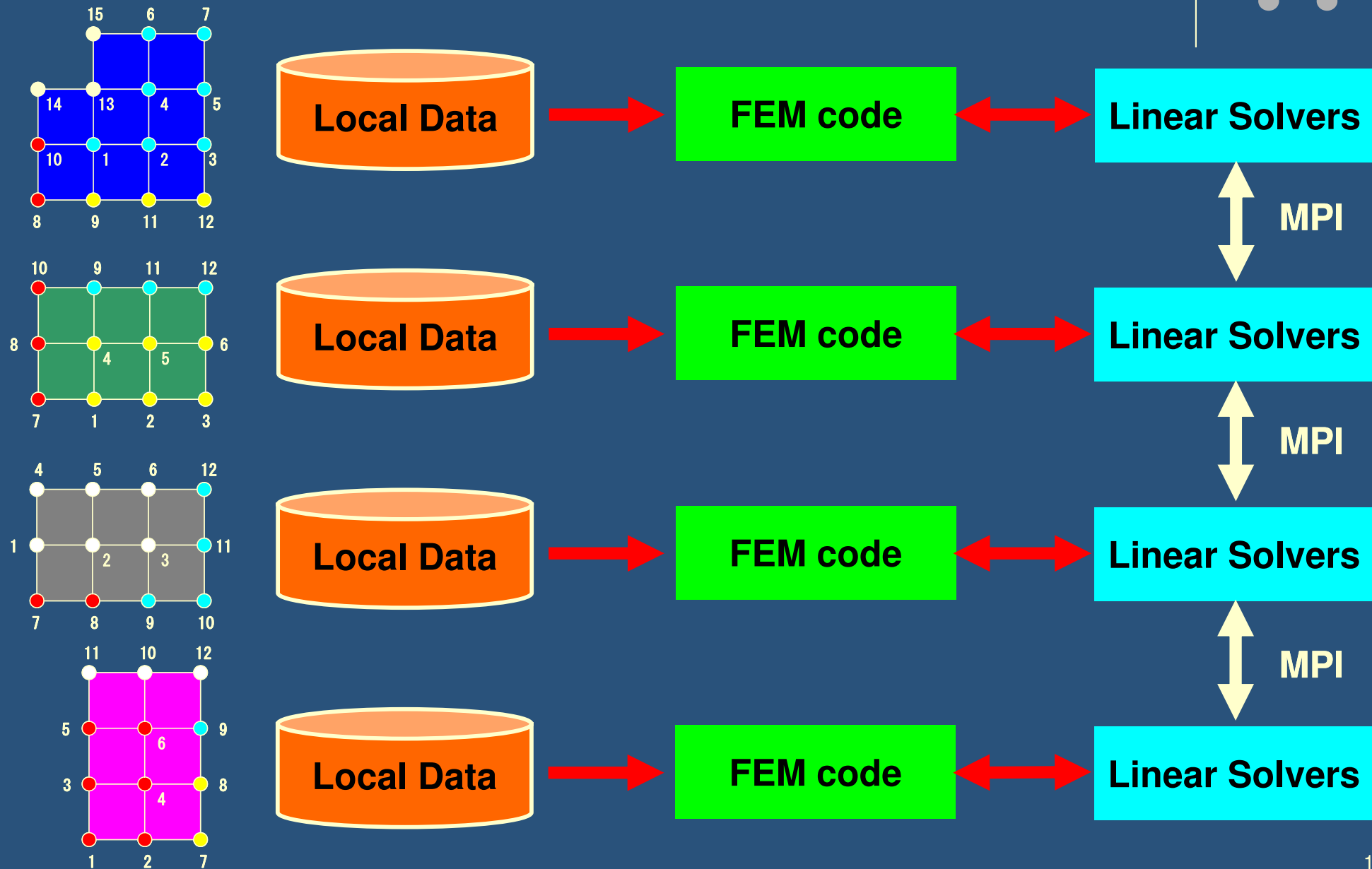
Parallel Computing in FEM

SPMD: Single-Program Multiple-Data



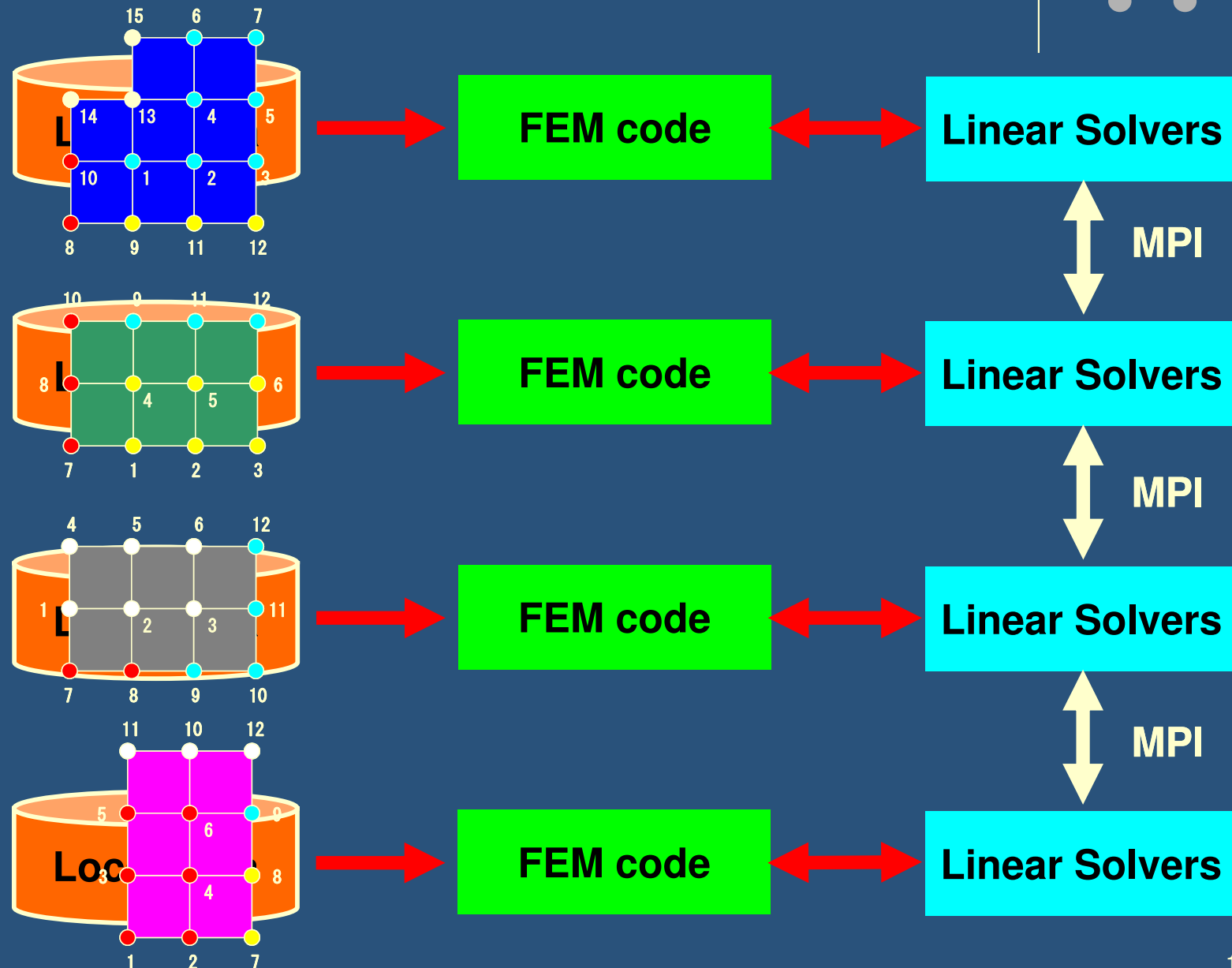
Parallel Computing in FEM

SPMD: Single-Program Multiple-Data



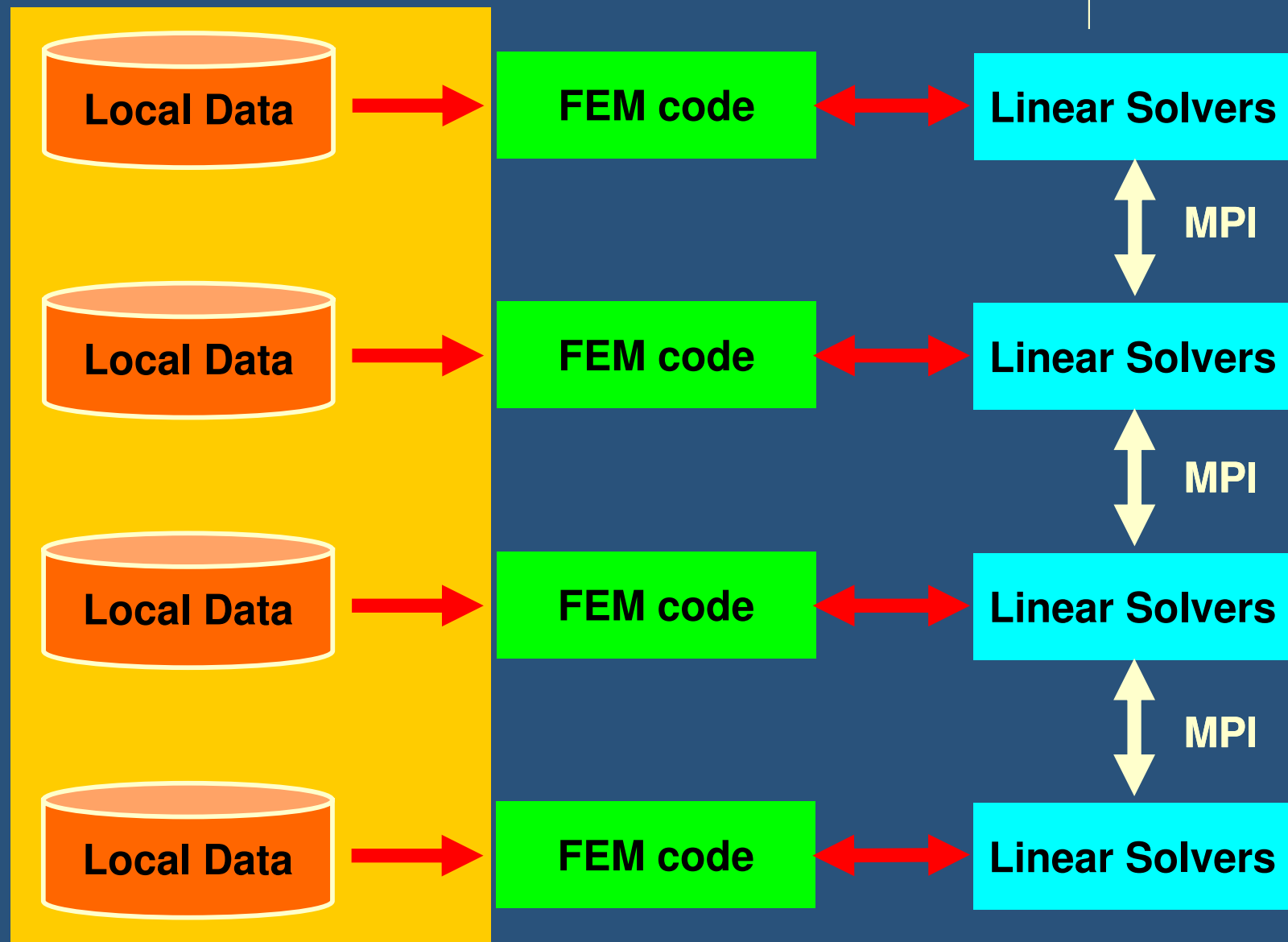
Parallel Computing in FEM

SPMD: Single-Program Multiple-Data

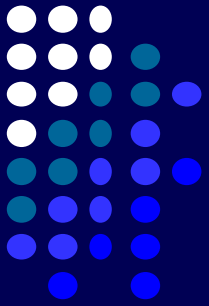


Parallel Computing in FEM

SPMD: Single-Program Multiple-Data

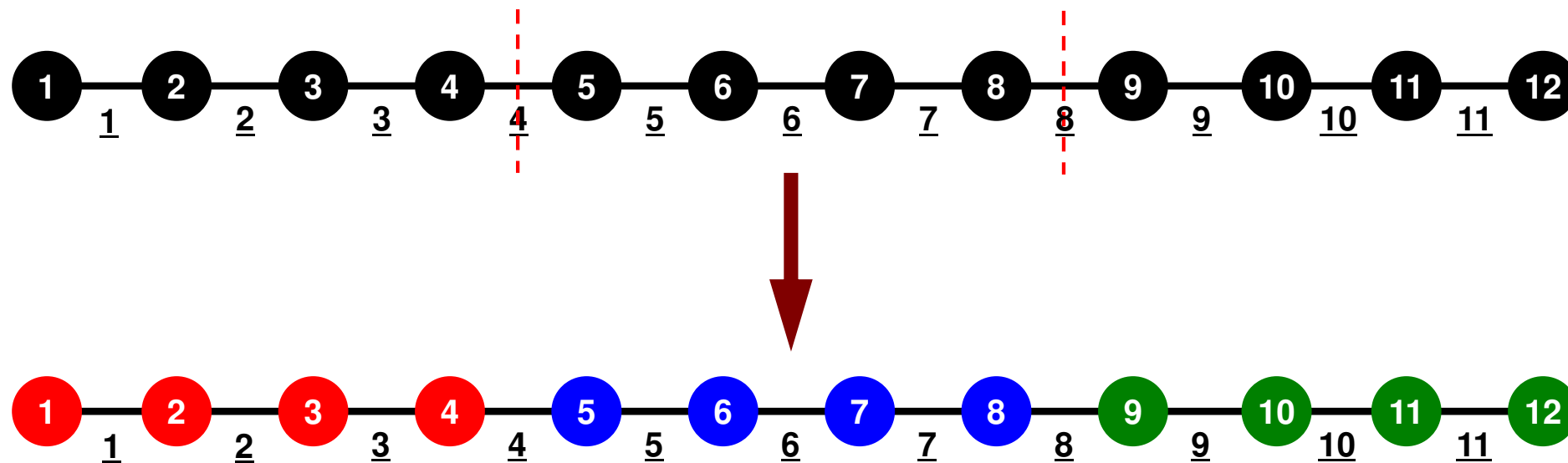


What is Communications ?



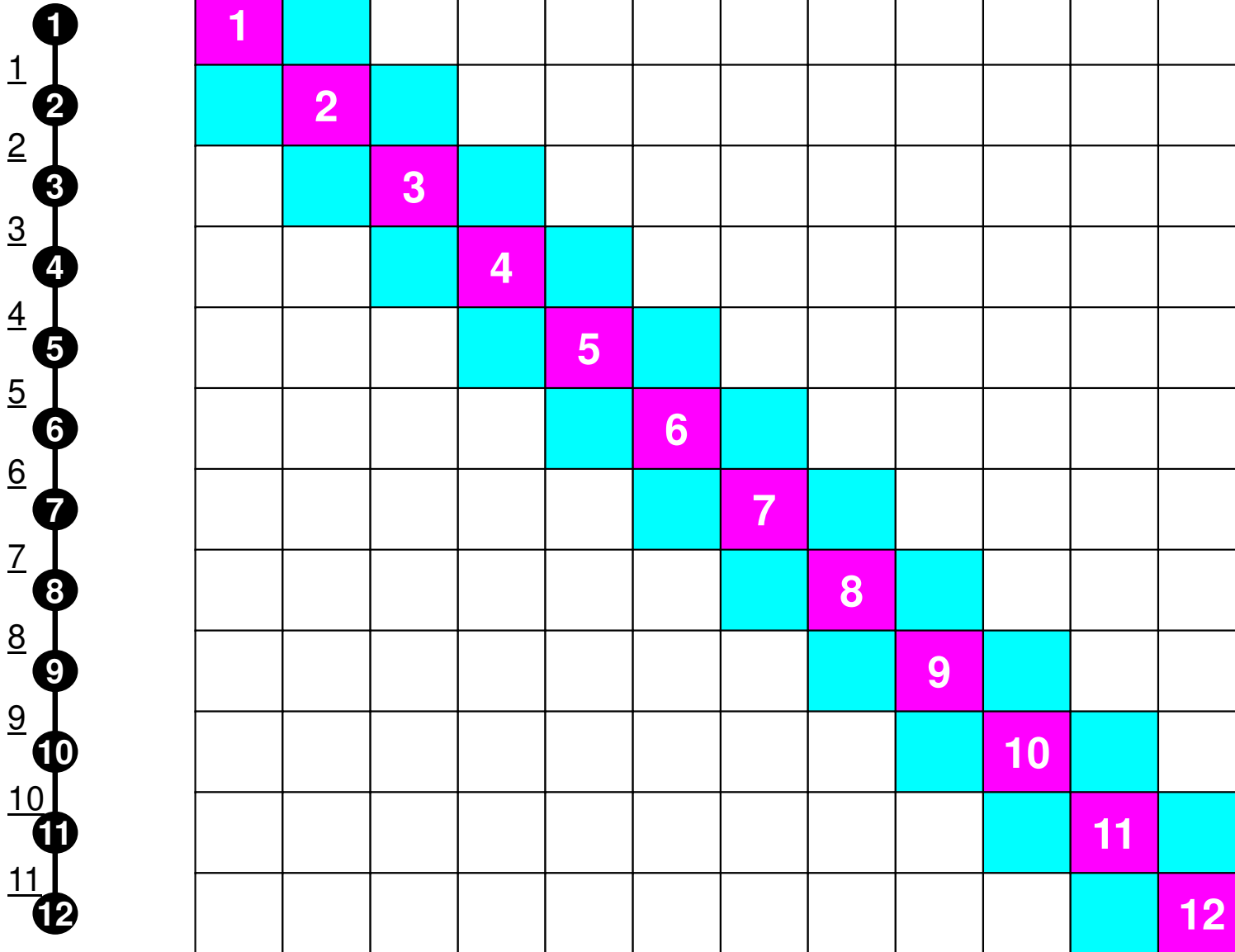
- to get information of “external nodes” from external partitions (local data)
- “Communication tables” contain the information

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



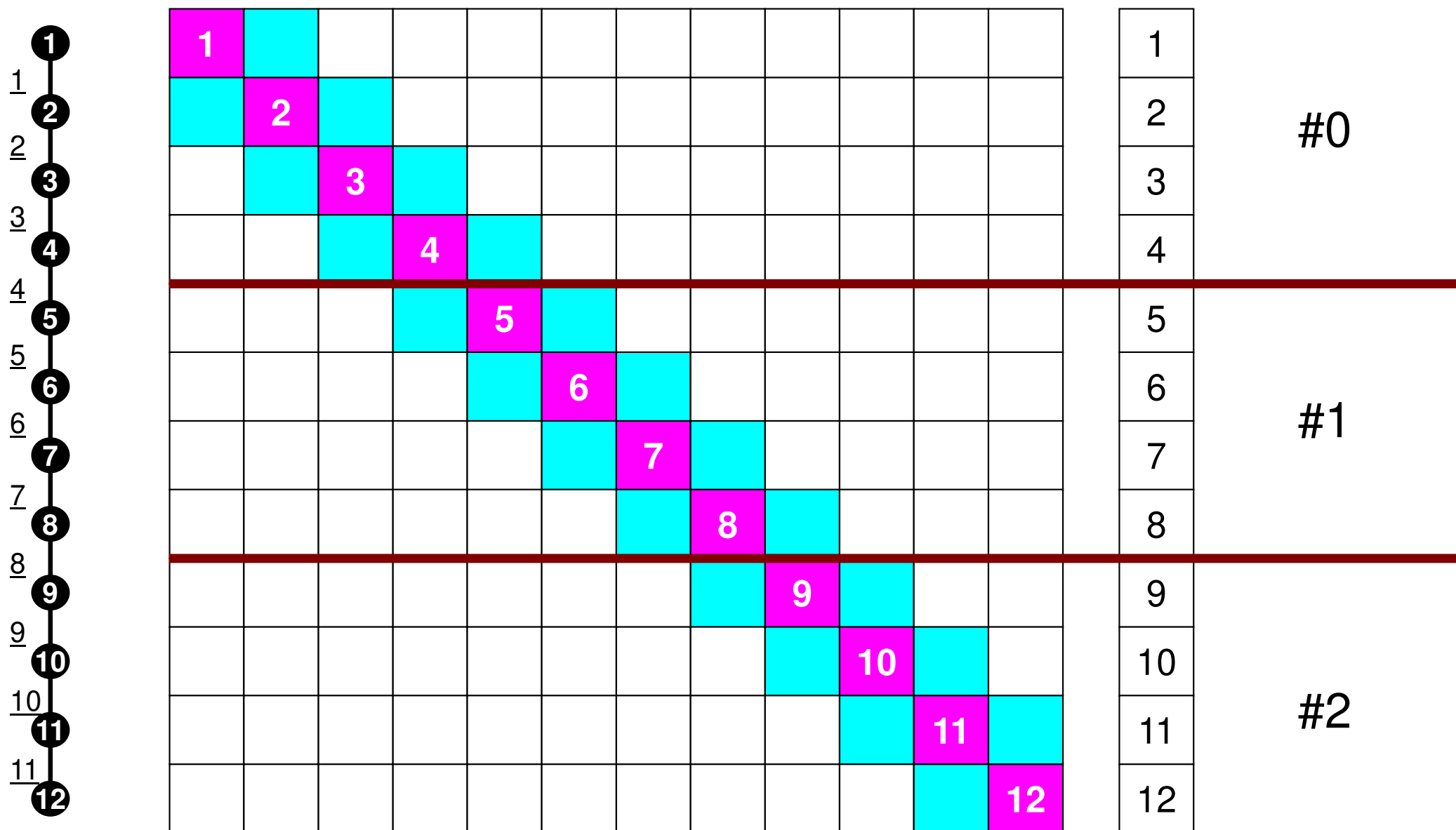
1D FEM: 12 nodes/11 elem's/3 domains

三重对角行列: Tri-Diagonal Matrix

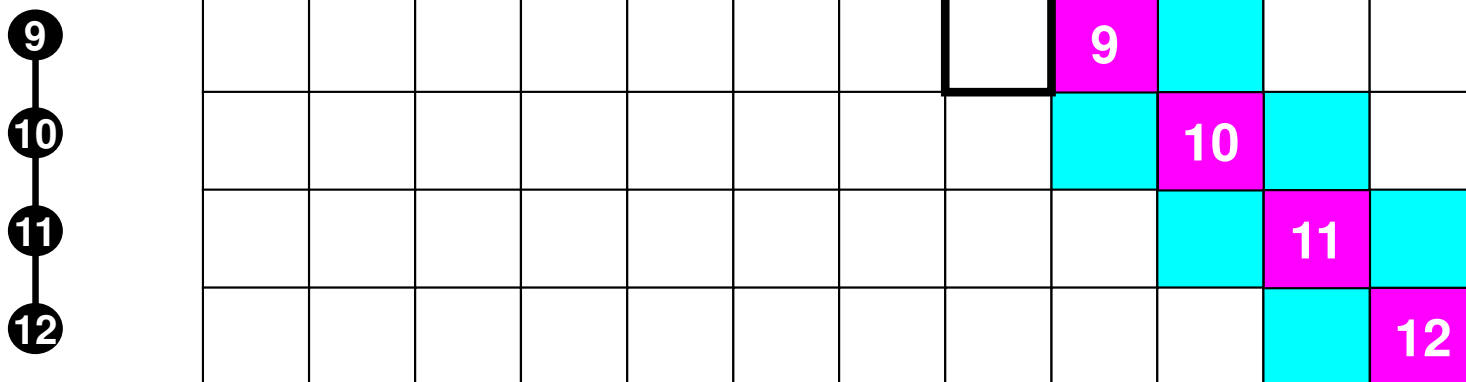
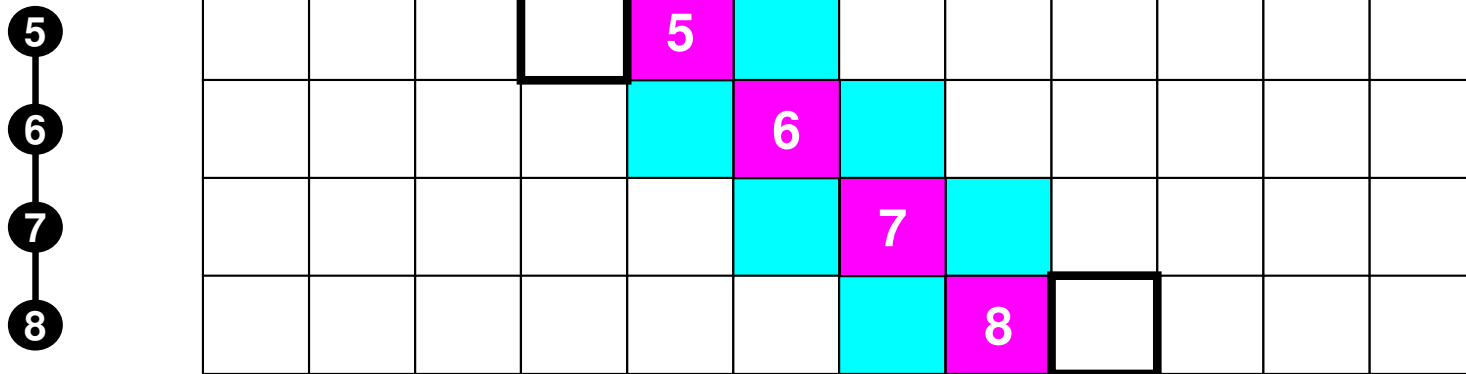
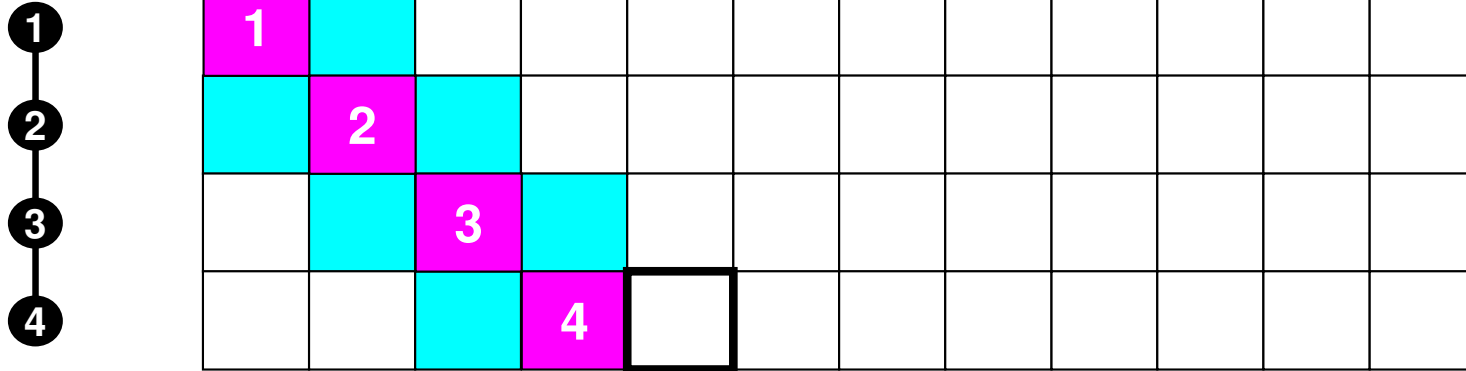


1
2
3
4
5
6
7
8
9
10
11
12

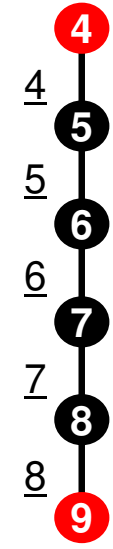
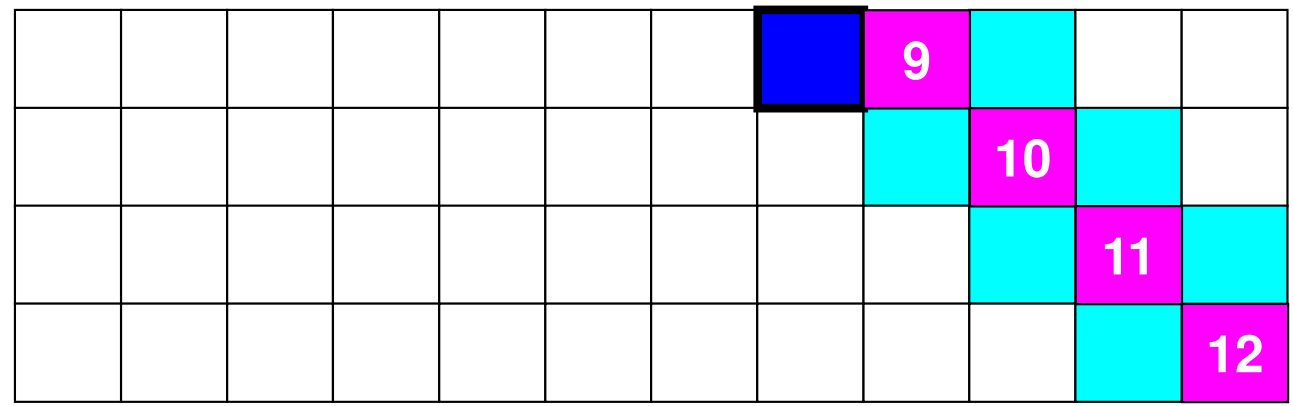
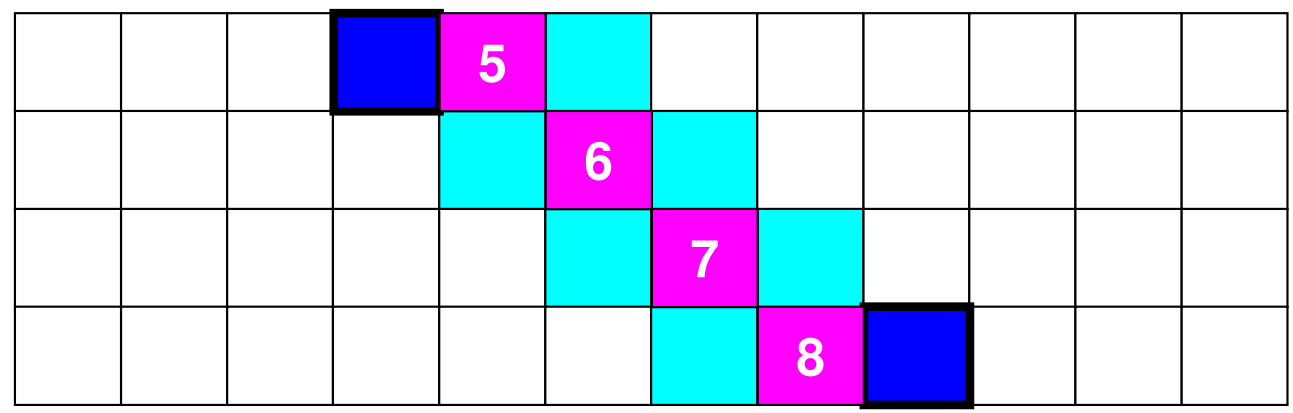
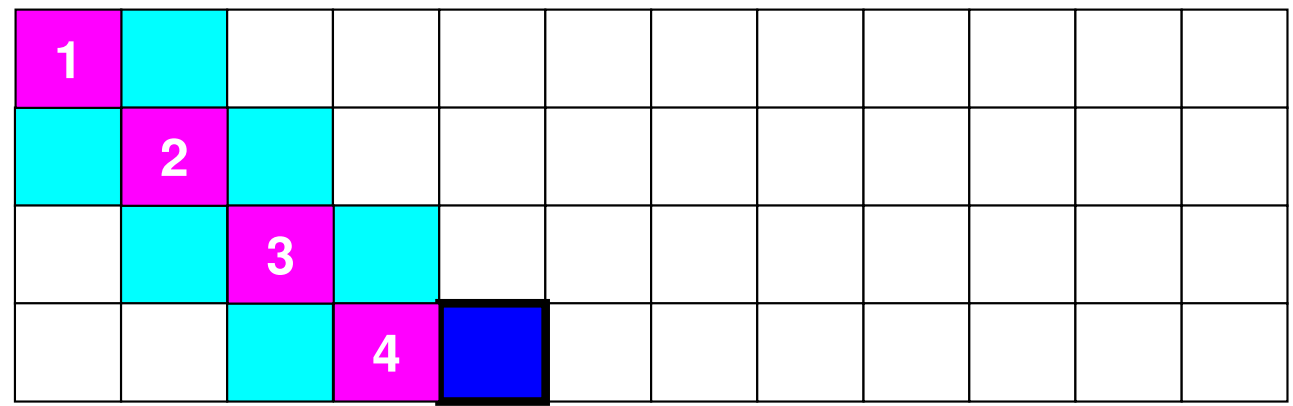
“Internal Nodes” should be balanced



Matrices are incomplete !



Connected Elements + External Nodes

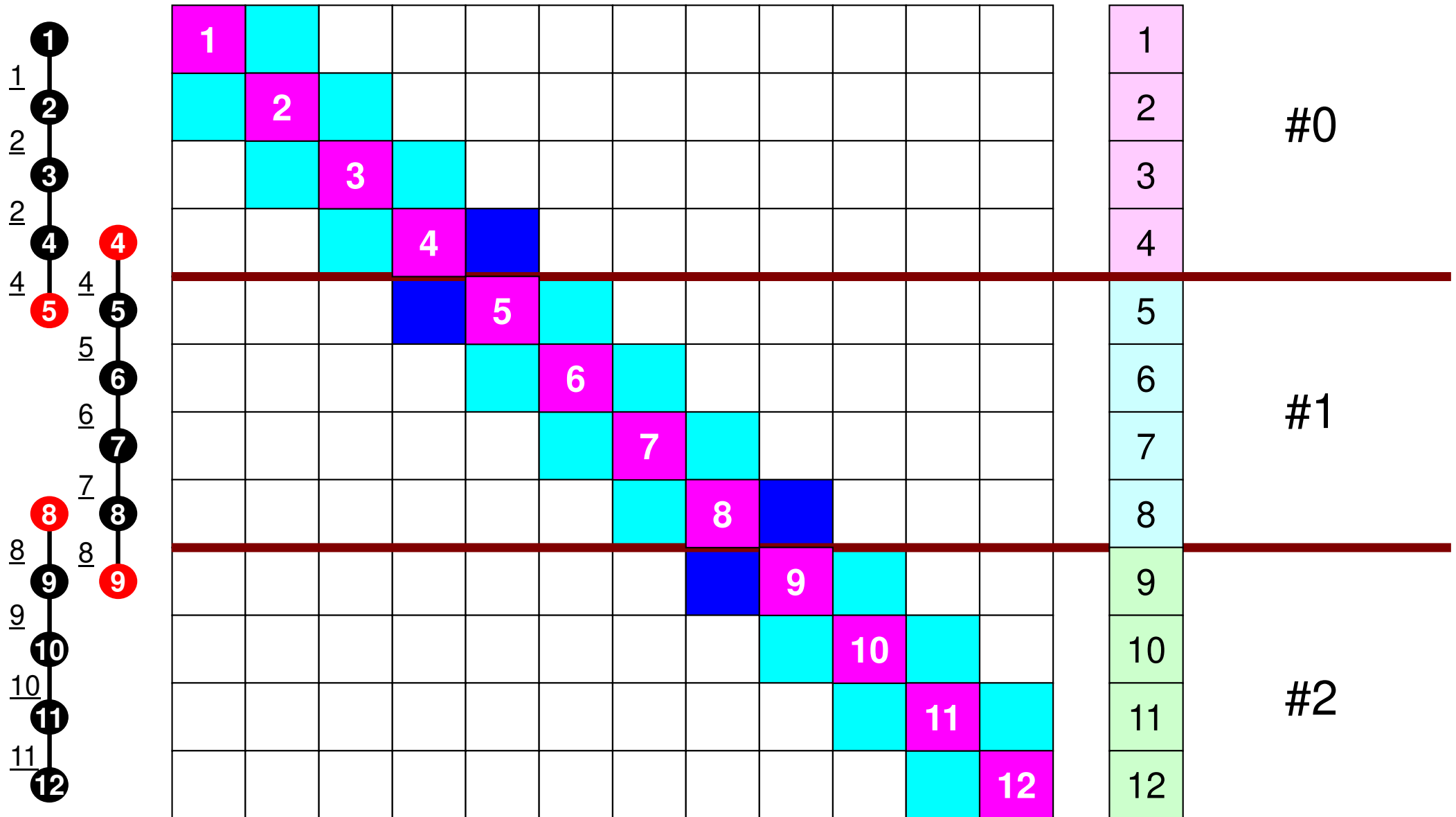


#0

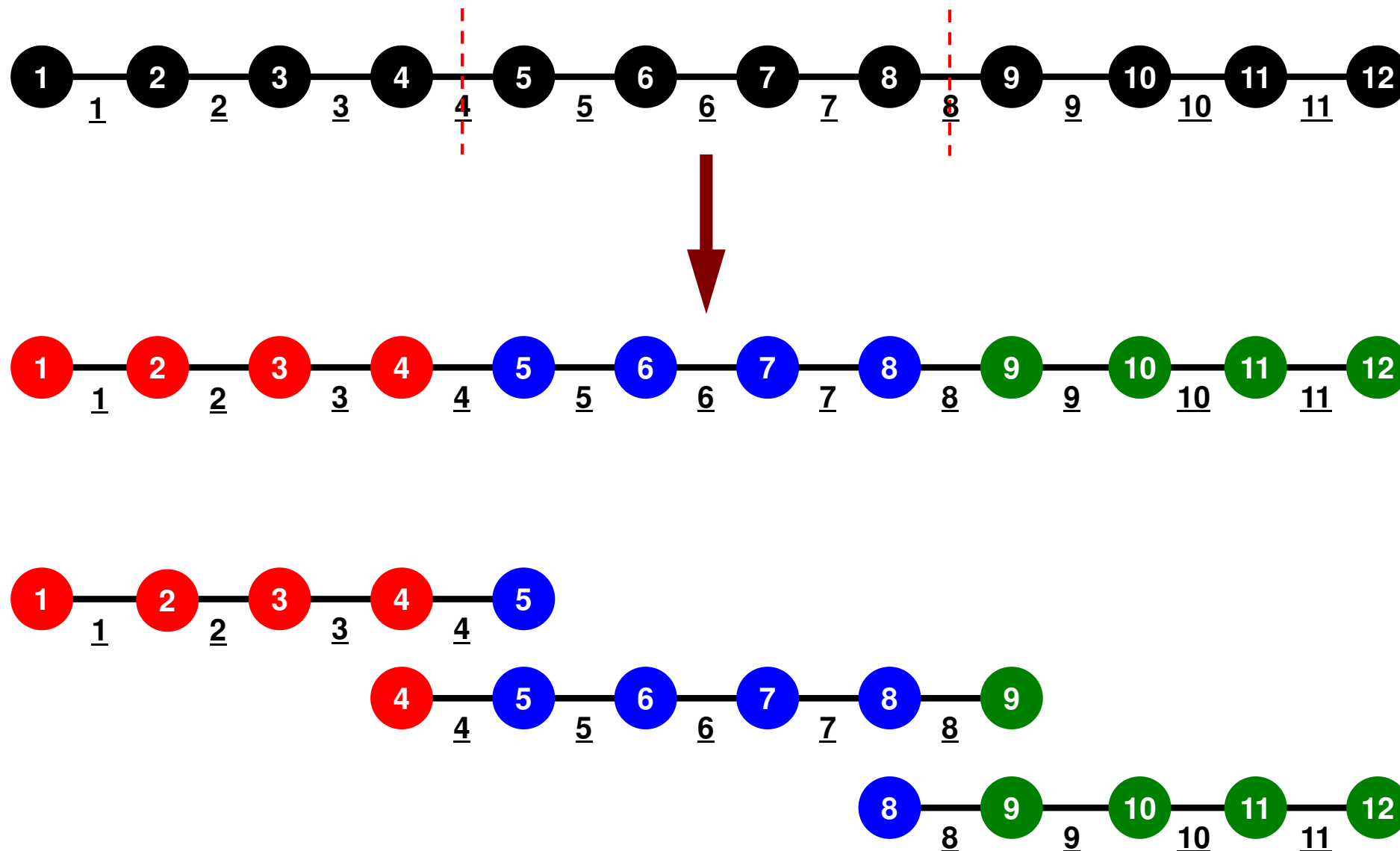
#1

#2

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



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

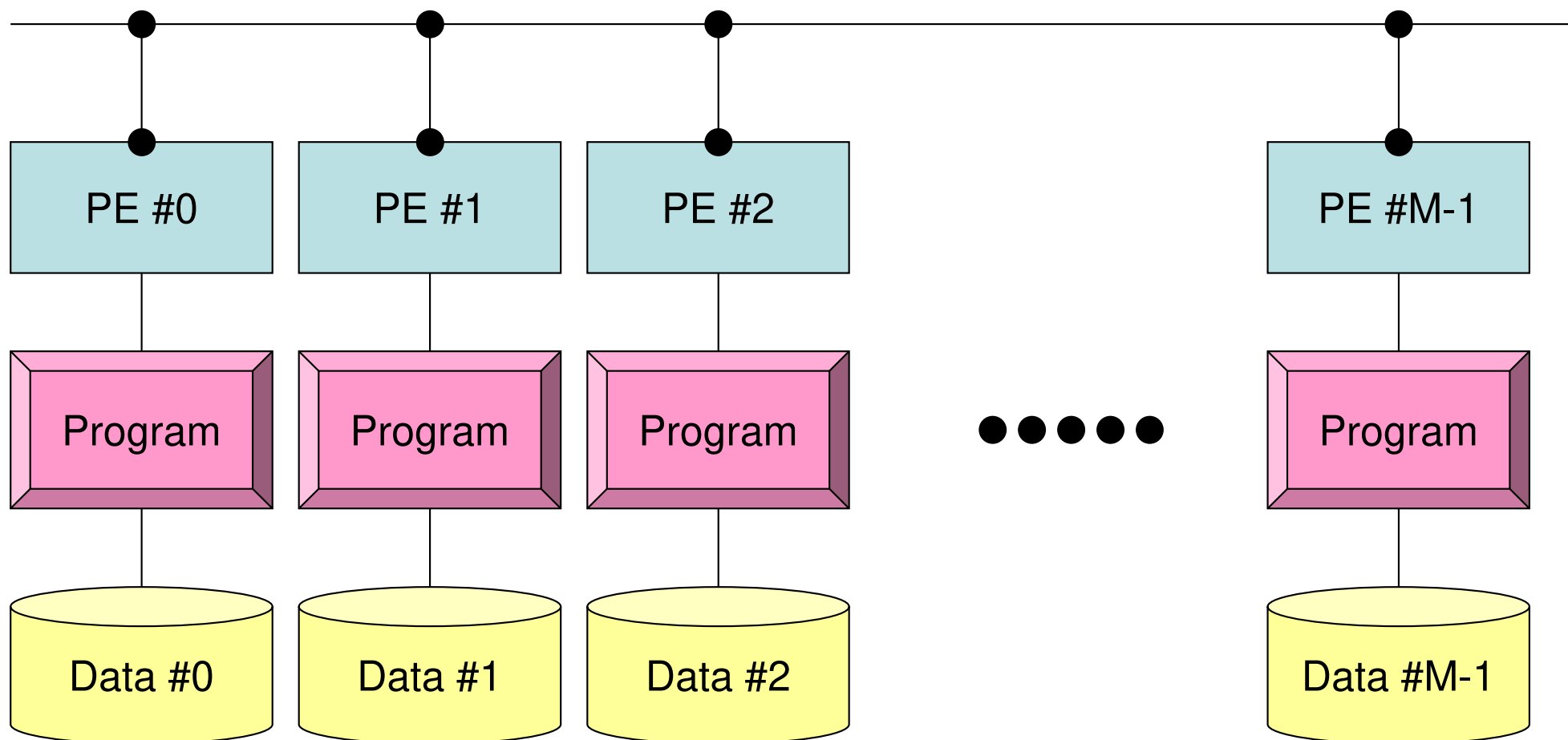


PE: Processing Element
Processor, Domain, Process

SPMD

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

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



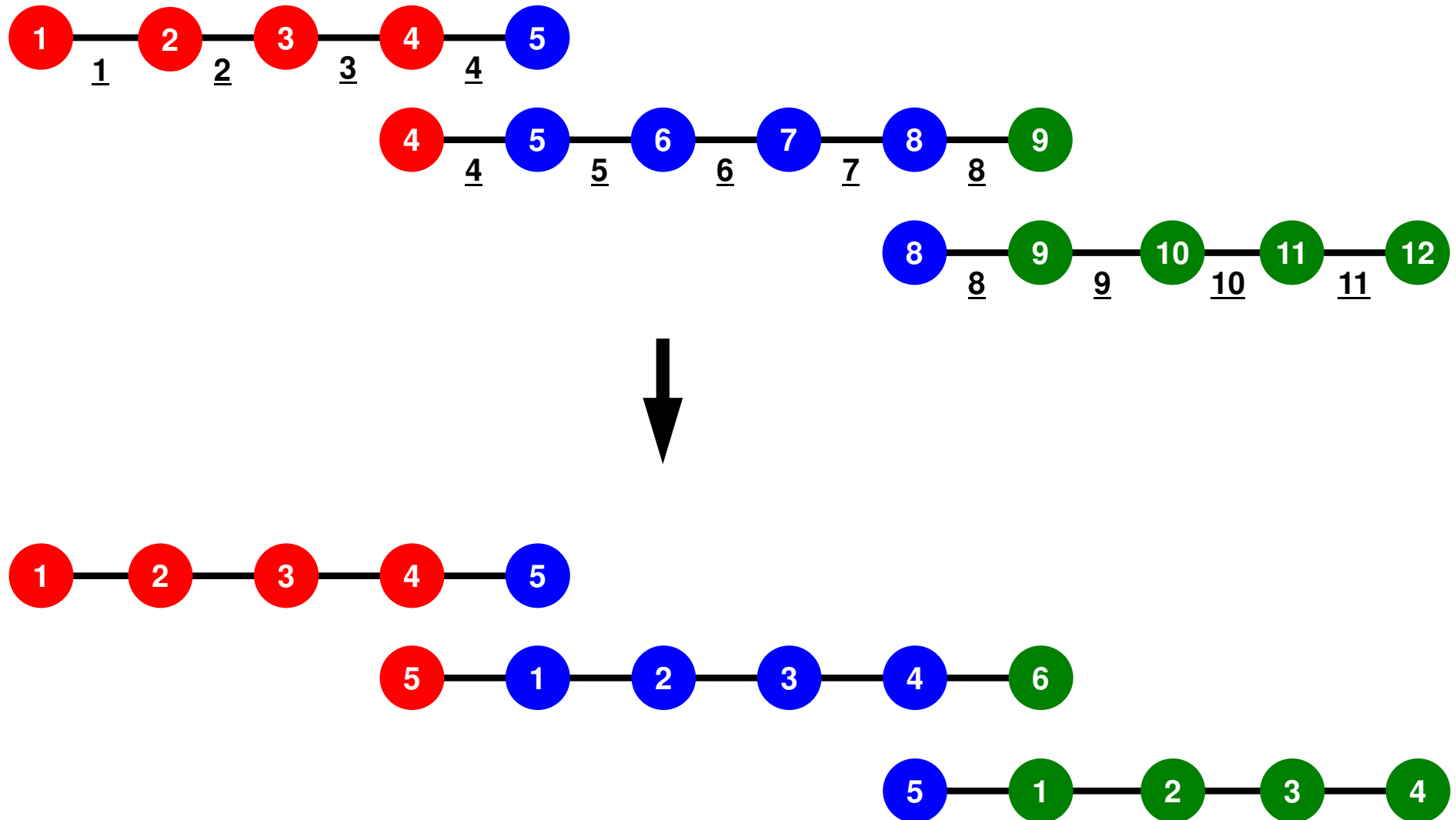
Each process does same operation for different data

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.

Local Numbering for SPMD

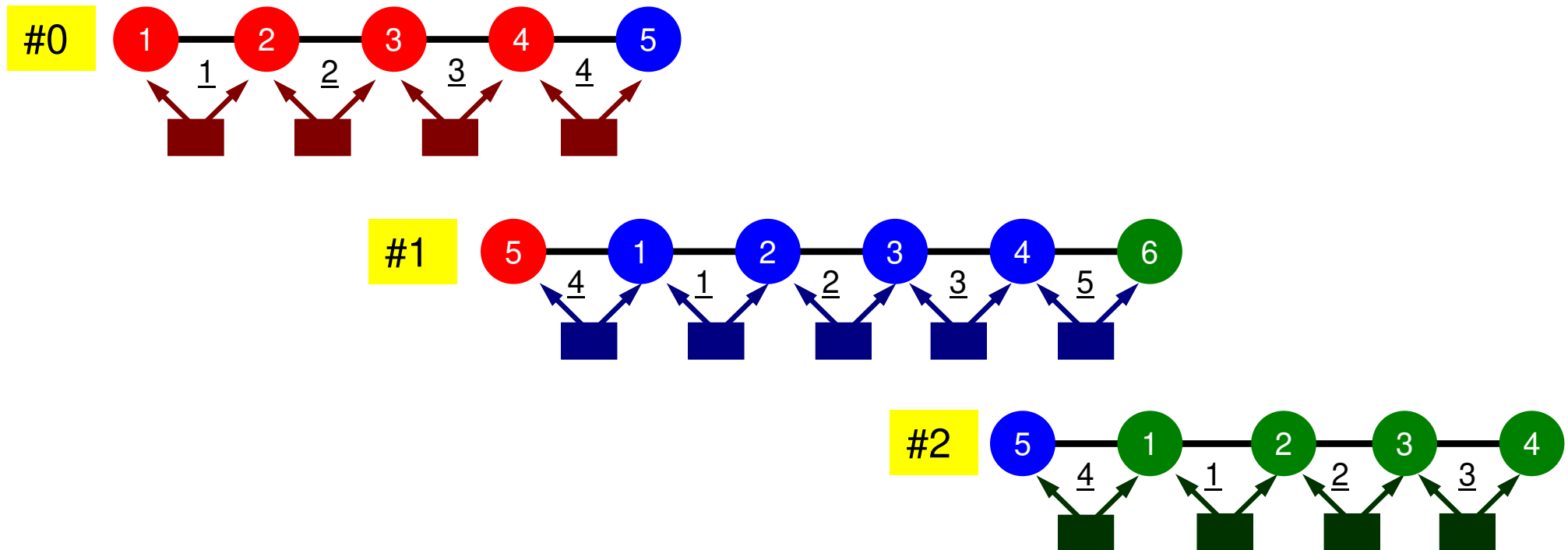
Numbering of internal nodes is 1-N (0-N-1), same operations in serial program can be applied. Numbering of external nodes: N+1, N+2 (N, N+1)



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

Integration on each element, element matrix \rightarrow global matrix

Operations can be done by info. of internal/external nodes and elements which include these nodes



Finite Element Procedures

- Initialization
 - Control Data
 - Node, Connectivity of Elements (N: Node#, NE: Elem#)
 - Initialization of Arrays (Global/Element Matrices)
 - Element-Global Matrix Mapping (Index, Item)
- Generation of Matrix
 - Element-by-Element Operations (do icel= 1, NE)
 - Element matrices
 - Accumulation to global matrix
 - Boundary Conditions
- Linear Solver
 - Conjugate Gradient Method

Preconditioned CG Solver

```

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

```

- Preconditioning
 - Diagonal Scaling/Point Jacobi
- Parallel operations are required in
 - Dot Products
 - Mat-Vec. Multiplication
 - SpMV: Sparse Mat-Vec. Mult.

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

Preconditioning, DAXPY

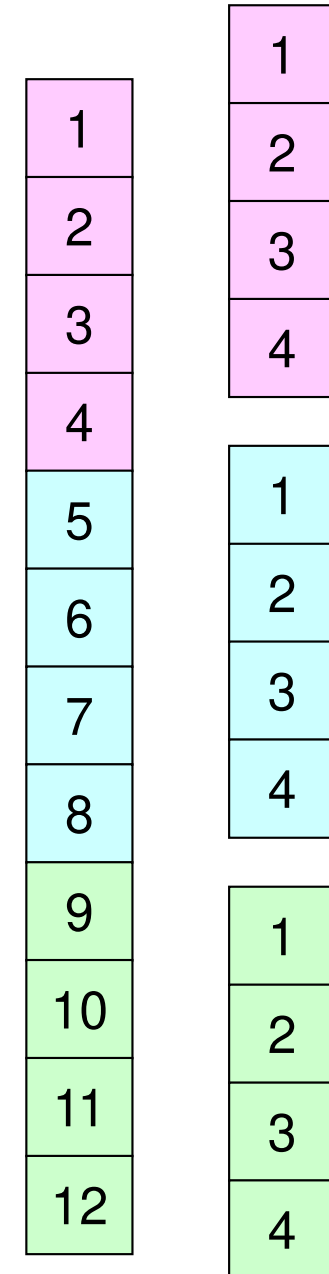
Local Operations by Only Internal Points: Parallel Processing is possible

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

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

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

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

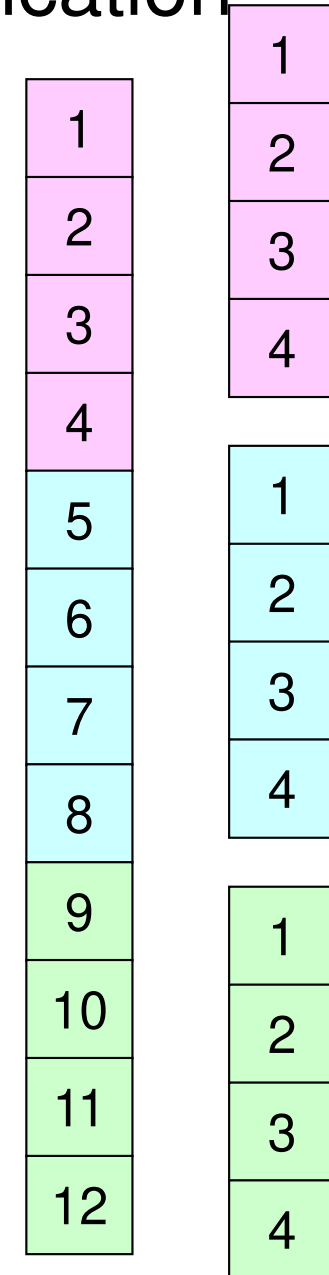


Dot Products

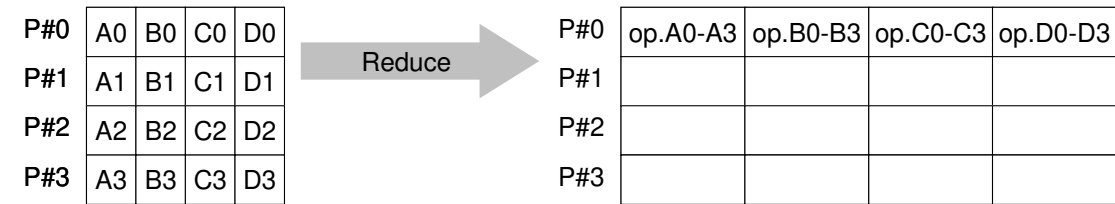
Global Summation needed: Communication ?

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

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



MPI_REDUCE



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

- **call MPI_REDUCE**

(sendbuf, recvbuf, count, datatype, op, root, comm, ierr)

- **sendbuf** choice I starting address of send buffer
- **recvbuf** choice O starting address receive buffer
type is defined by "**datatype**"
- **count** I I number of elements in send/receive buffer
- **datatype** I 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** I 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** I I rank of root process
- **comm** I I communicator
- **ierr** I O completion code

Preconditioned CG Solver

```

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

```

- Preconditioning
 - Diagonal Scaling/Point Jacobi
- Parallel operations are required in
 - Dot Products
 - Mat-Vec. Multiplication
 - SpMV: Sparse Mat-Vec. Mult.

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

Matrix-Vector Products

Values at External Points: P-to-P Communication

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

do i= 1, N
  W(i, Q) = DIAG(i)*W(i, P)
  do j= INDEX(i-1)+1, INDEX(i)
    W(i, Q) = W(i, Q) + AMAT(j)*W(ITEM(j), P)
  enddo
enddo
```



Mat-Vec Products: Local Op. Possible

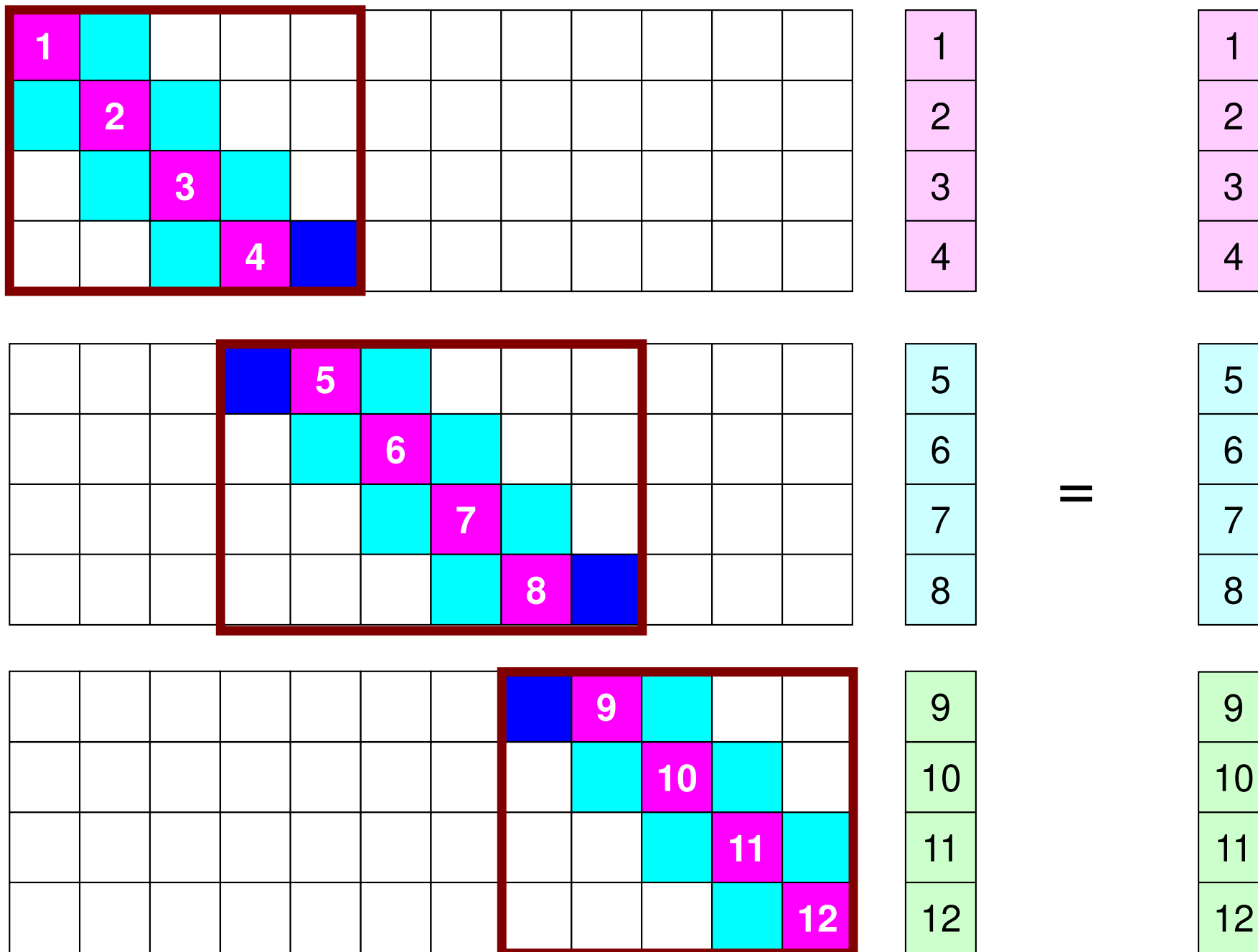
1											
	2										
		3									
			4								
				5							
					6						
						7					
							7				
								9			
									10		
										11	
											12

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

=

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

Mat-Vec Products: Local Op. Possible



Mat-Vec Products: Local Op. Possible

1				
	2			
		3		
			4	

1
2
3
4

1
2
3
4

	5			
		6		
			7	
				8

5
6
7
8

=

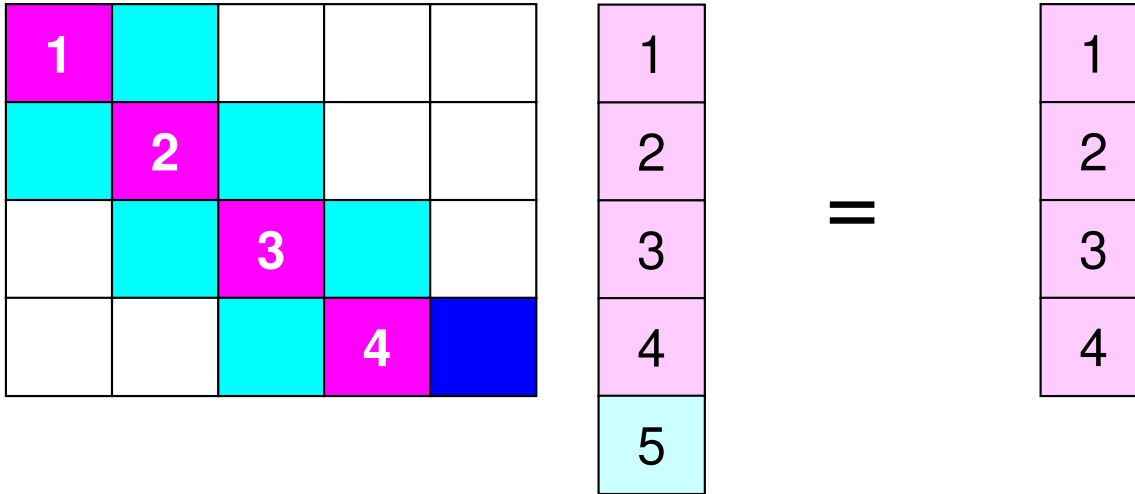
5
6
7
8

	9			
		10		
			11	
				12

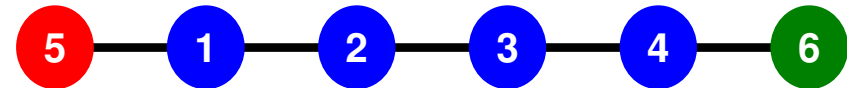
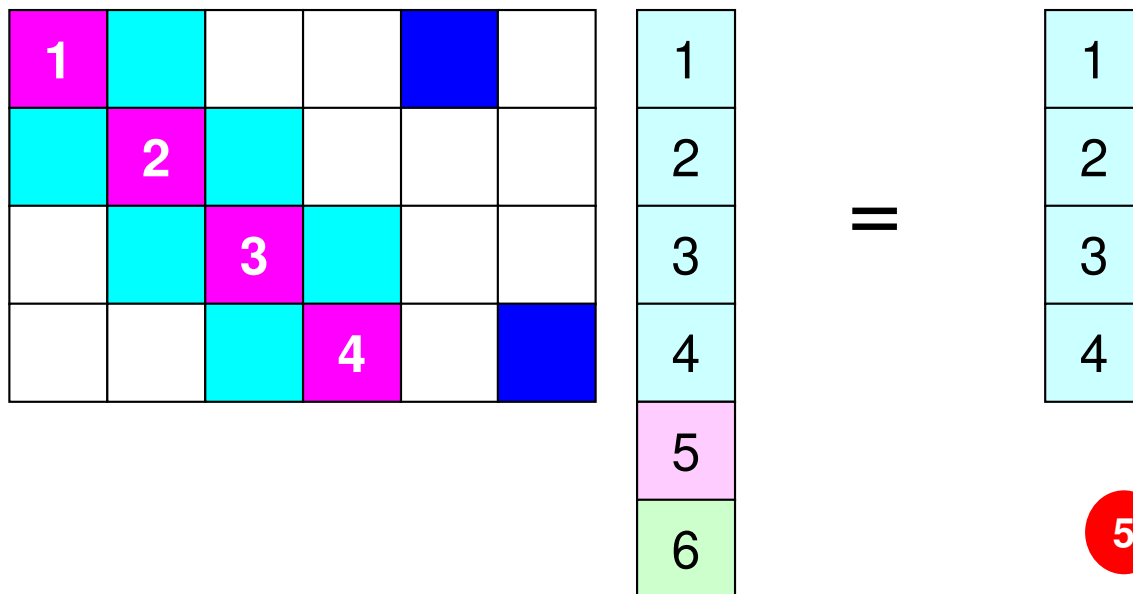
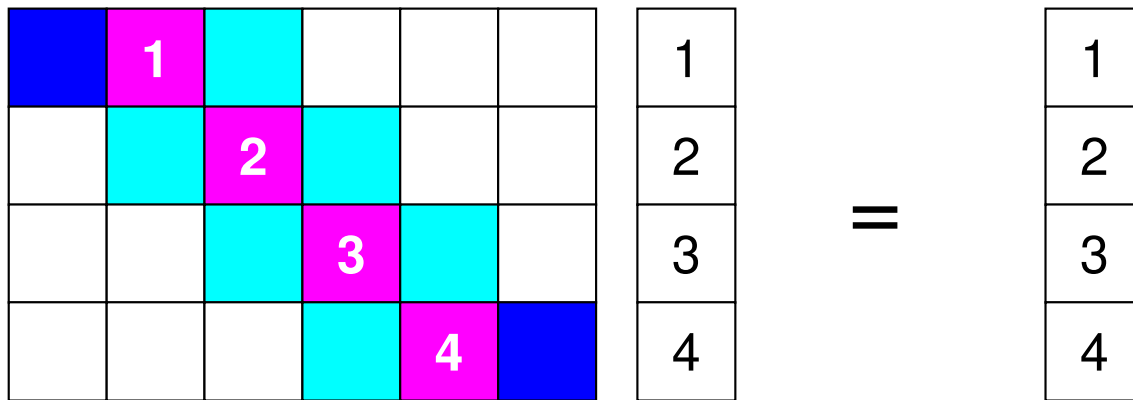
9
10
11
12

9
10
11
12

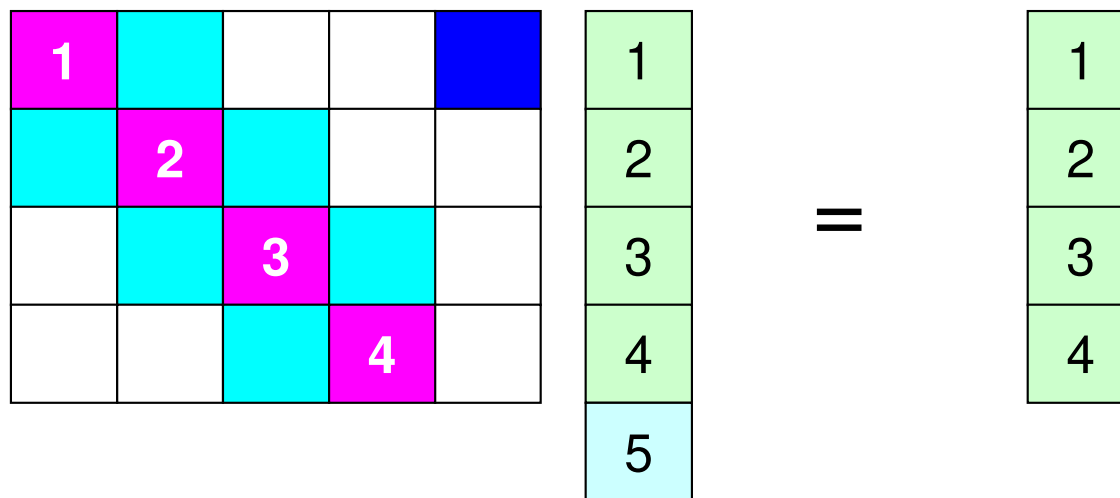
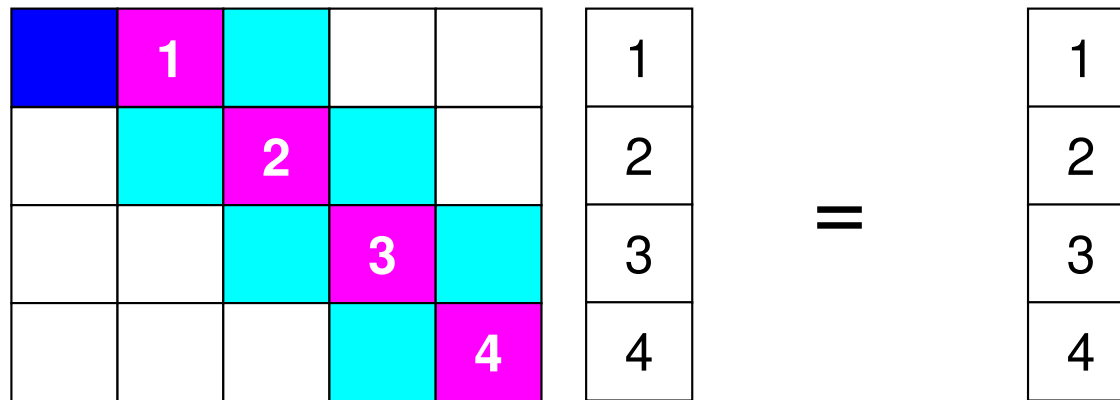
Mat-Vec Products: Local Op. #0



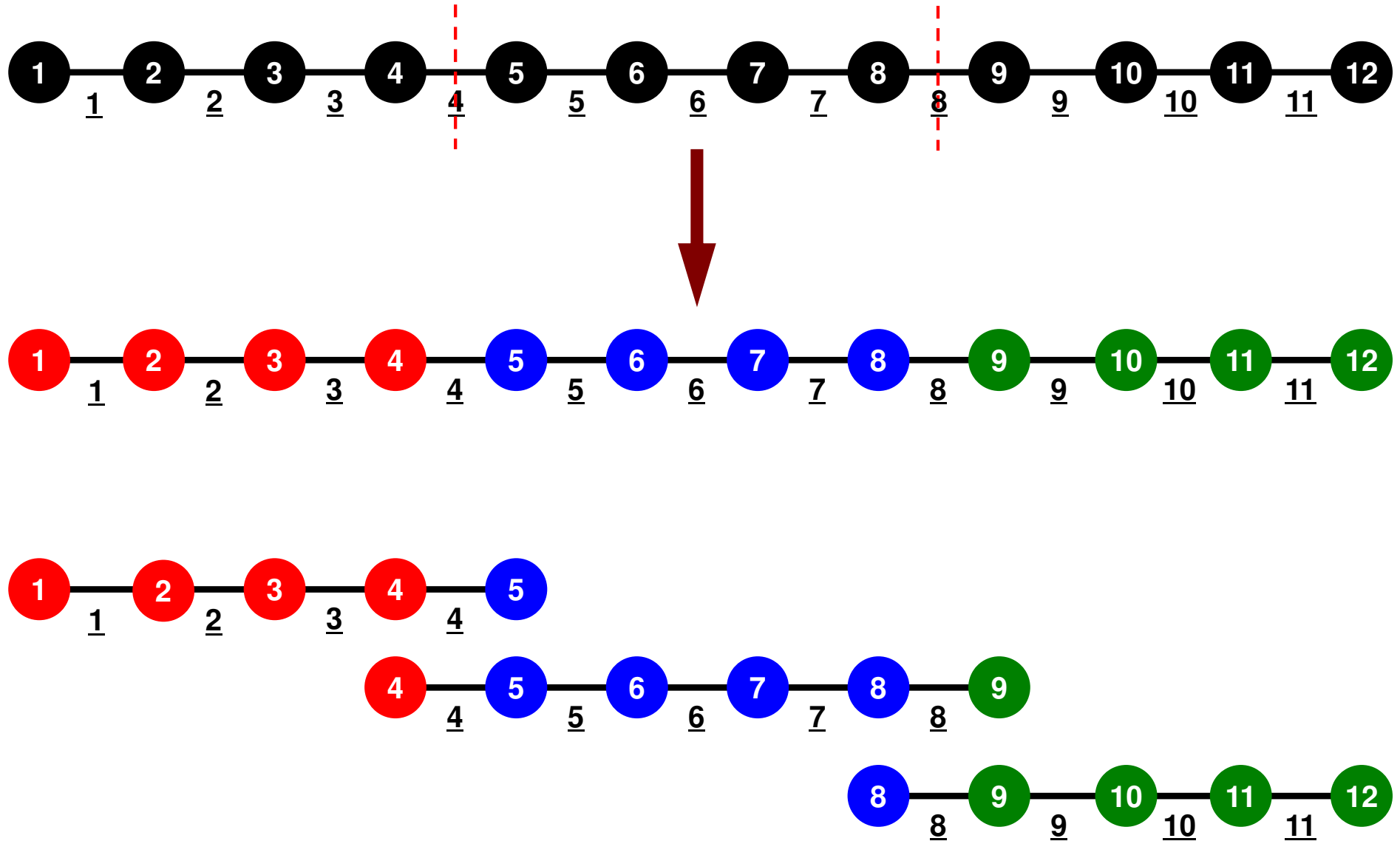
Mat-Vec Products: Local Op. #1



Mat-Vec Products: Local Op. #2

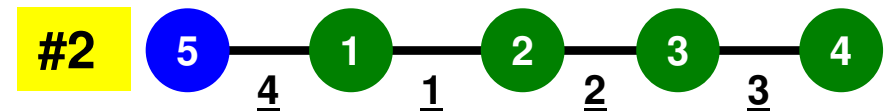
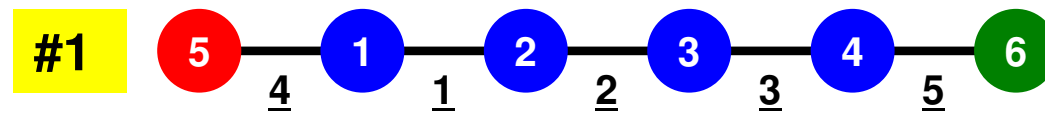
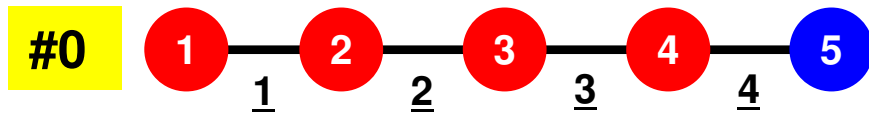


1D FEM: 12 nodes/11 elem's/3 domains



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

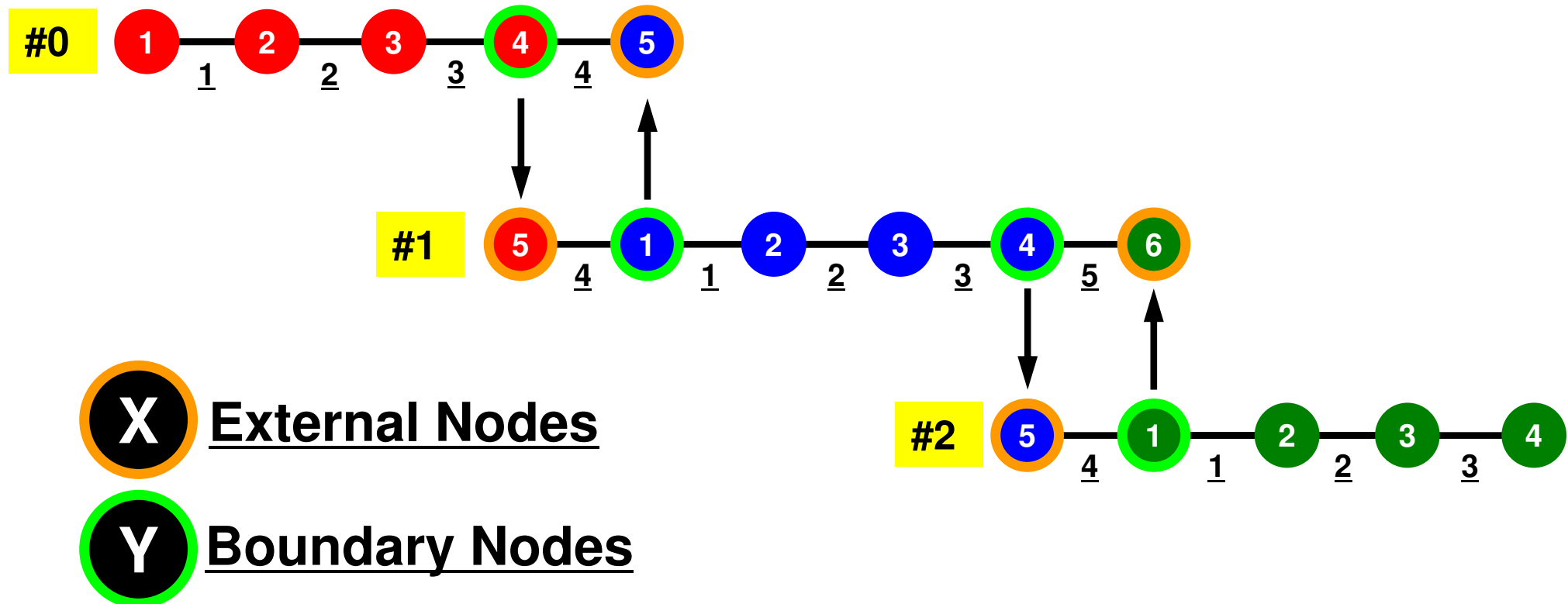
Local ID: Starting from 1 for node and elem at each domain



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

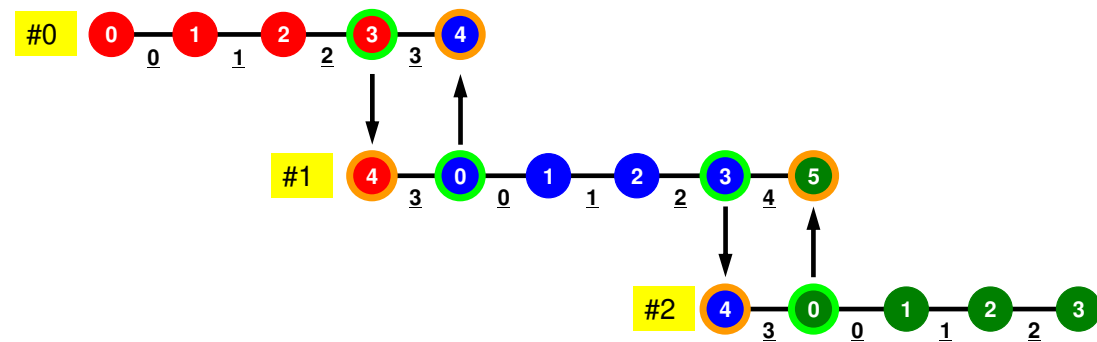
Internal/External/Boundary Nodes

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



What is Peer-to-Peer Communication ?

- Collective Communication
 - MPI_Reduce, MPI_Scatter/Gather etc.
 - Communications with all processes in the communicator
 - Application Area
 - BEM, Spectral Method, MD: global interactions are considered
 - Dot products, MAX/MIN: Global Summation & Comparison
- Peer-toPeer/Point-to-Point
 - MPI_Send, MPI_Recv
 - Communication with limited processes
 - Neighbors
 - Application Area
 - FEM, FDM: Localized Method



MPI_ISEND

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

- **call MPI_ISEND**

(sendbuf, count, datatype, dest, tag, comm, request, ierr)

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

MPI_Irecv

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

- **call MPI_Irecv**

(recvbuf, count, datatype, dest, tag, comm, request, ierr)

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

MPI_WAITALL

- **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 consistent.
 - Same request should be used in MPI_Isend and MPI_Irecv.
- Its operation is similar to that of **MPI_Barrier** but, **MPI_Waitall** can not be replaced by **MPI_Barrier**.
 - Possible troubles using **MPI_Barrier** instead of **MPI_Waitall**: Contents of **request** and **status** are not updated properly, very slow operations etc.
- **call MPI_WAITALL (count, request, status, ierr)**
 - count I I number of processes to be synchronized
 - request I I/O comm. request used in MPI_Waitall (array size: count)
 - status I O array of status objects
MPI_STATUS_SIZE: defined in 'mpif.h', 'mpi.h'
 - ierr I O completion code