

# **Introduction to Parallel FEM in C**

## **Parallel Data Structure**

Kengo Nakajima  
Information Technology Center

Programming for Parallel Computing (616-2057)  
Seminar on Advanced Computing (616-4009)

# Parallel Computing

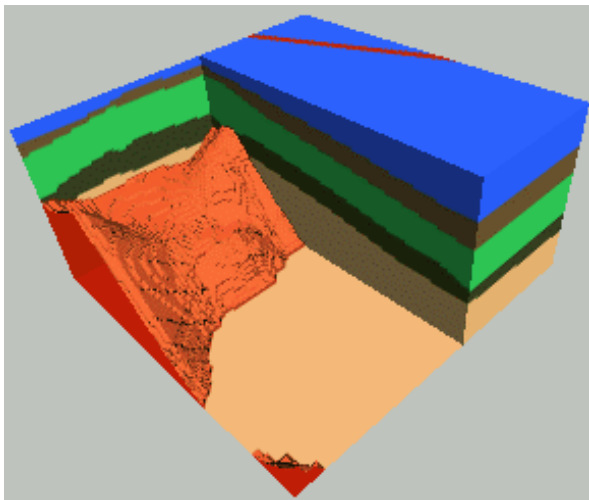
- Faster, Larger & More Complicated
- Scalability
  - Solving  $N^x$  scale problem using  $N^x$  computational resources during same computation time
    - for large-scale problems: Weak Scaling
    - e.g. CG solver: more iterations needed for larger problems
  - Solving a problem using  $N^x$  computational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

# What is Parallel Computing ? (1/2)

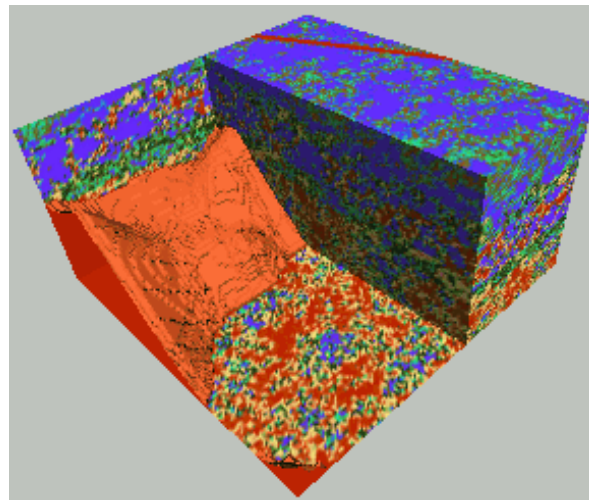
- to solve larger problems faster

## Homogeneous/Heterogeneous Porous Media

Lawrence Livermore National Laboratory



Homogeneous

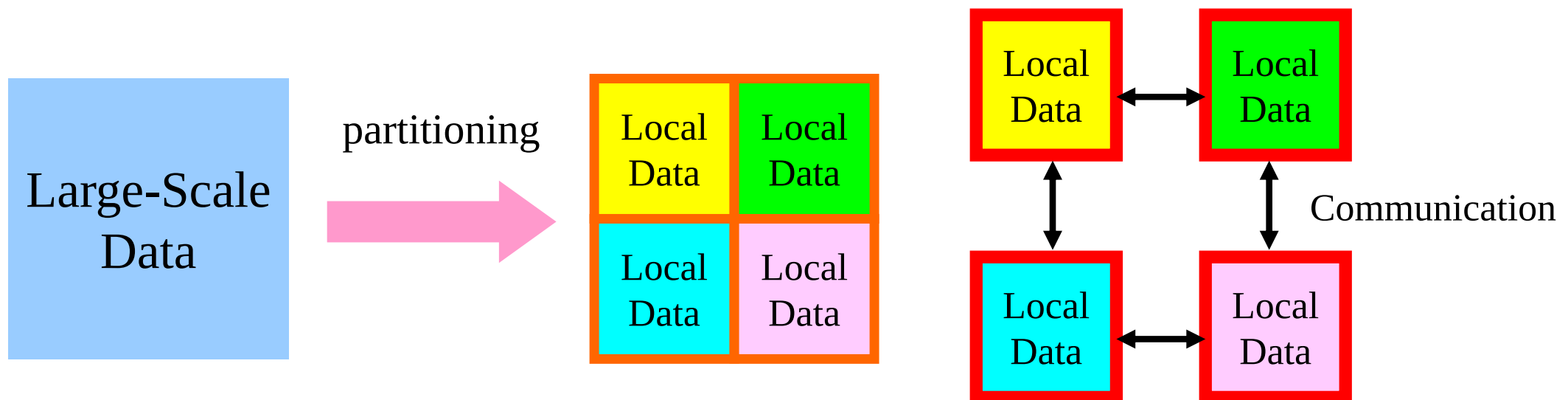


Heterogeneous

**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.

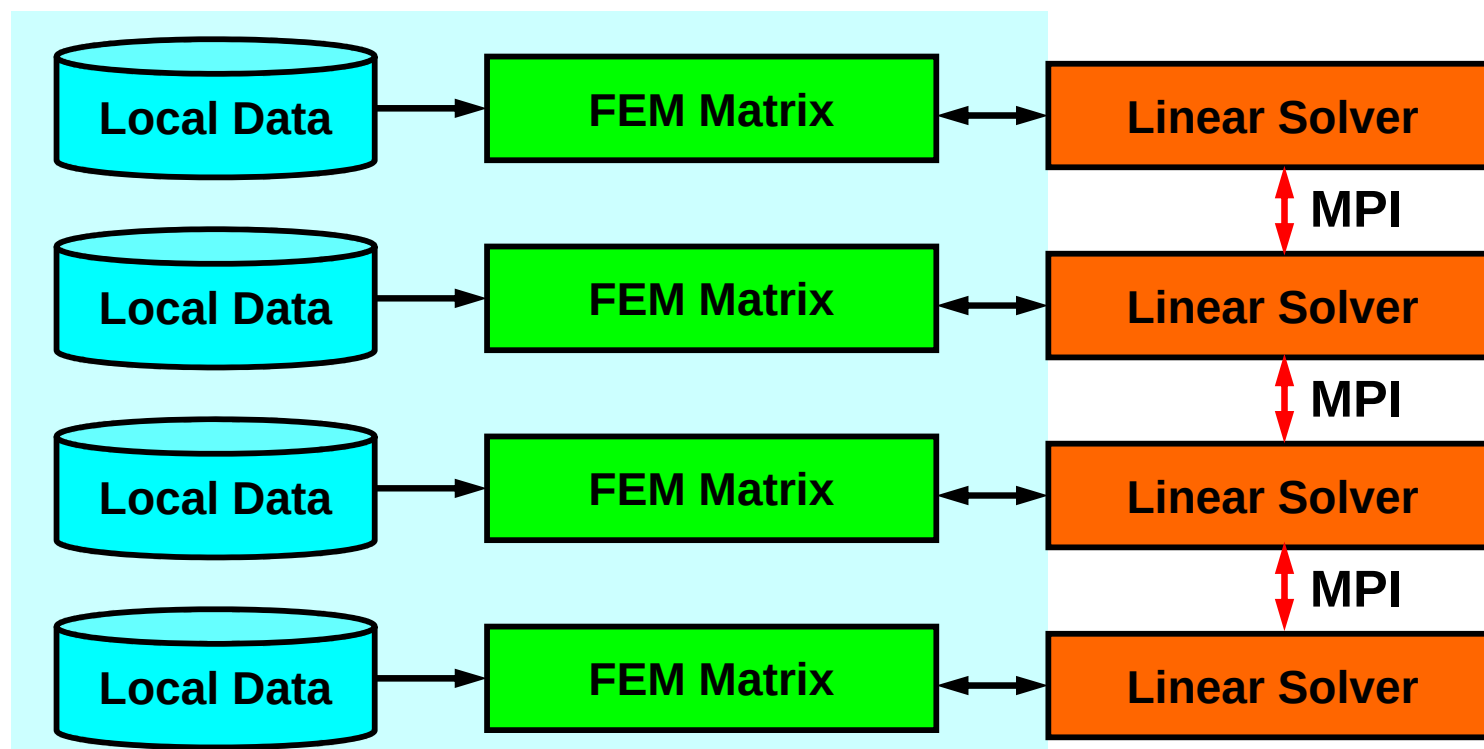
# Operations in Parallel FEM

## SPMD: Single-Program Multiple-Data

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

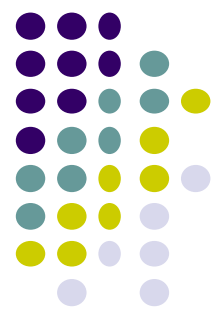
FEM code can assemble coefficient matrix for each local data set :  
this part could be completely local, same as serial operations

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



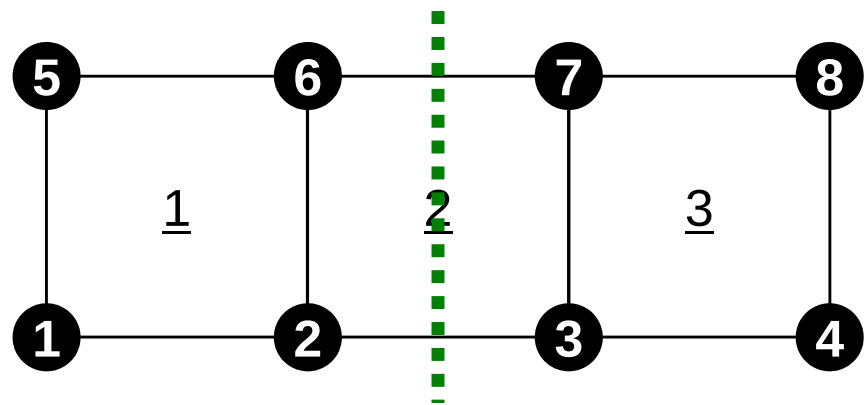
# Parallel FEM Procedures

- 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

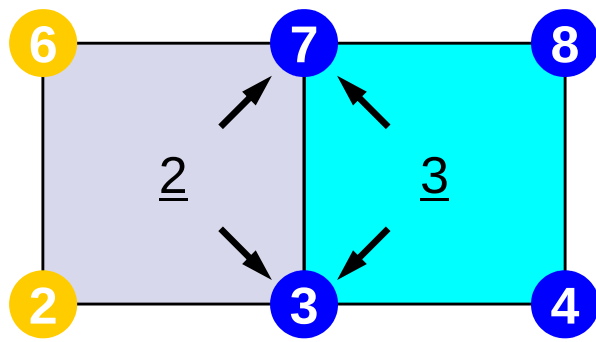
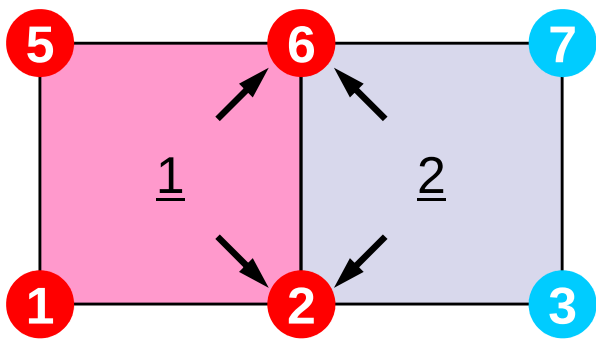
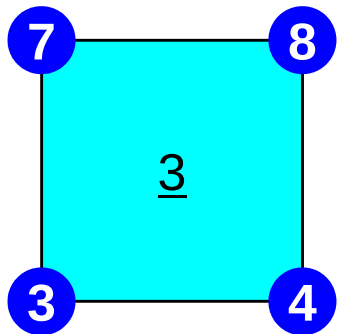
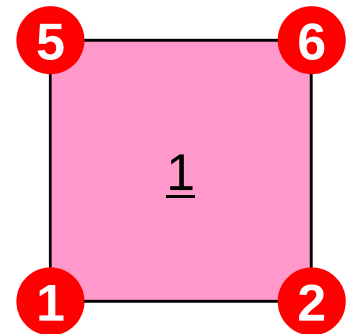
Values are defined on each node



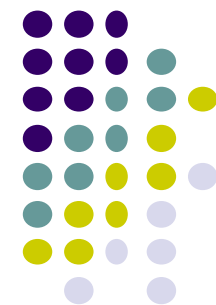
divide into two domains by “node-based” manner, where number of “nodes (vertices)” are balanced.

Local information is not enough for matrix assembling.

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





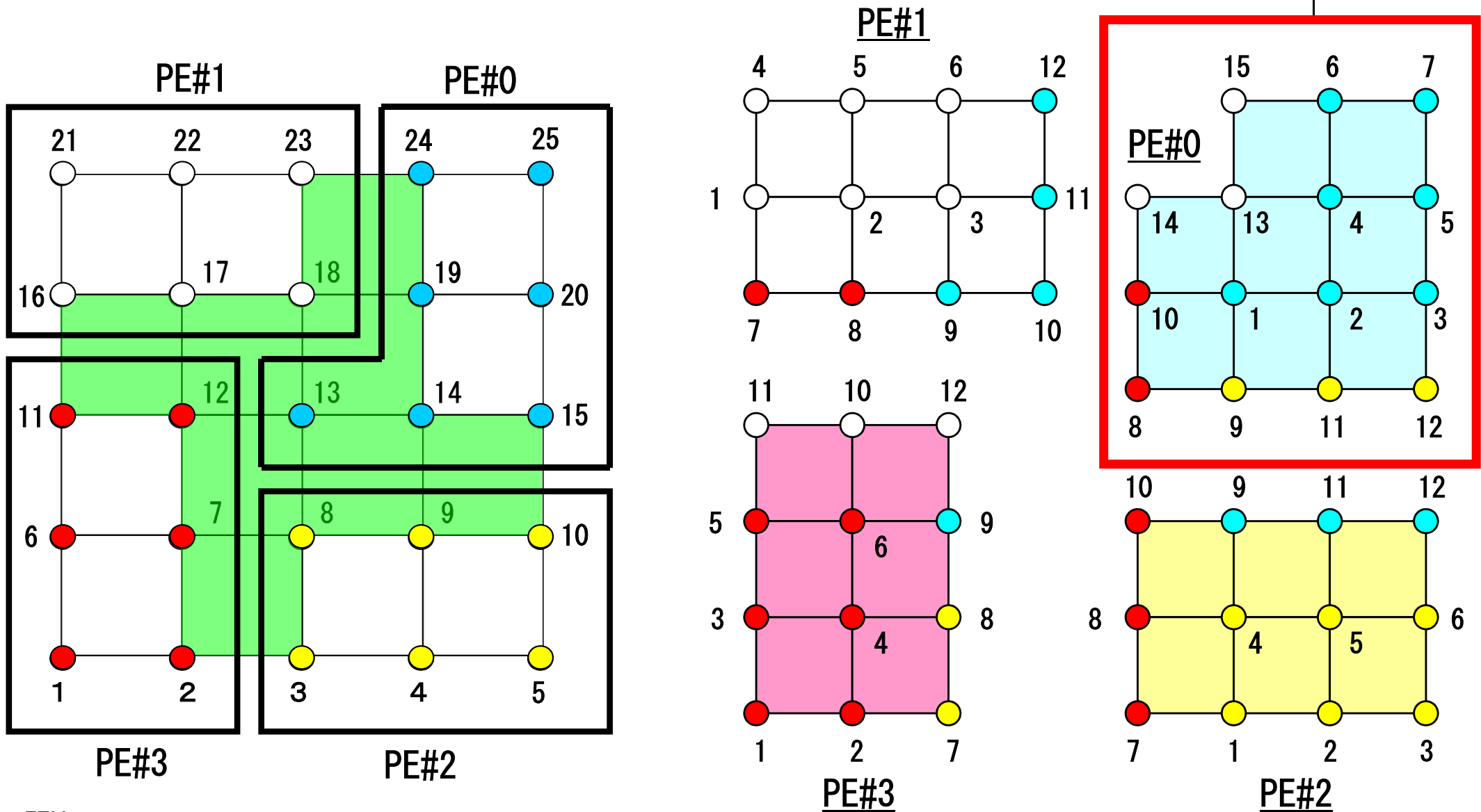


# Local Data of Parallel FEM

- **Node-based partitioning for IC/ILU type preconditioning methods**
- Local data includes information for :
  - Nodes originally assigned to the partition/PE
  - Elements which include the nodes : Element-based operations (Matrix Assemble) are allowed for fluid/structure subsystems.
  - 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
- Communication table between partitions
- NO global information required except partition-to-partition connectivity

# Node-based Partitioning

internal nodes - elements - external nodes

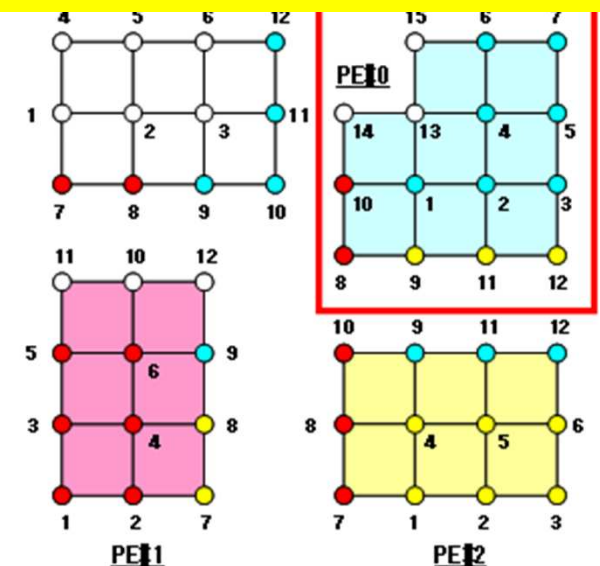
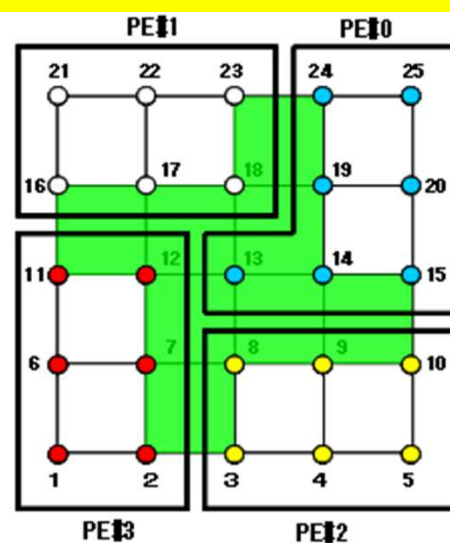
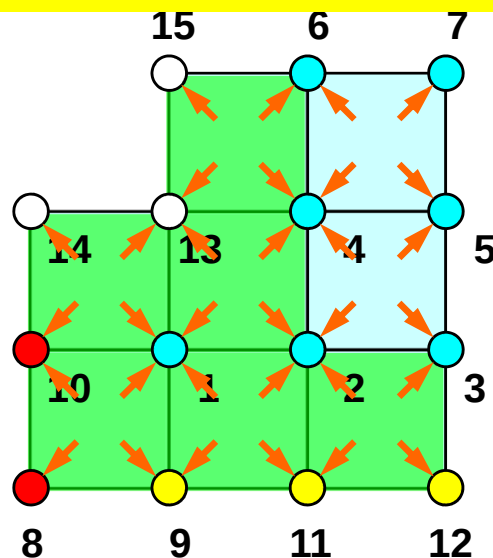


# Node-based Partitioning

internal nodes - elements - external nodes

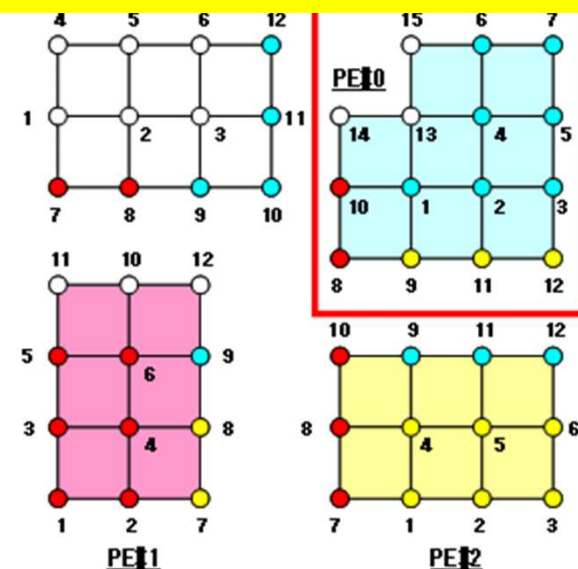
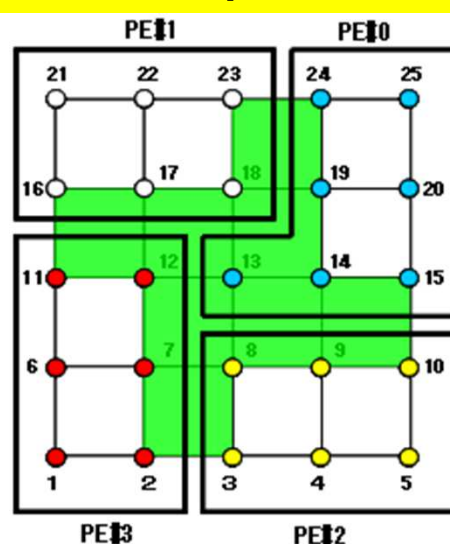
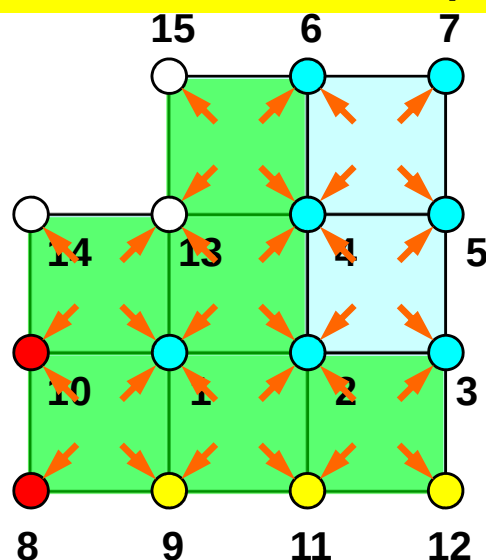


- Partitioned nodes themselves (Internal Nodes) 内点
- Elements which include Internal Nodes 内点を含む要素
- External Nodes included in the Elements 外点  
in overlapped region among partitions.
- Info of External Nodes are required for completely local element-based operations on each processor.



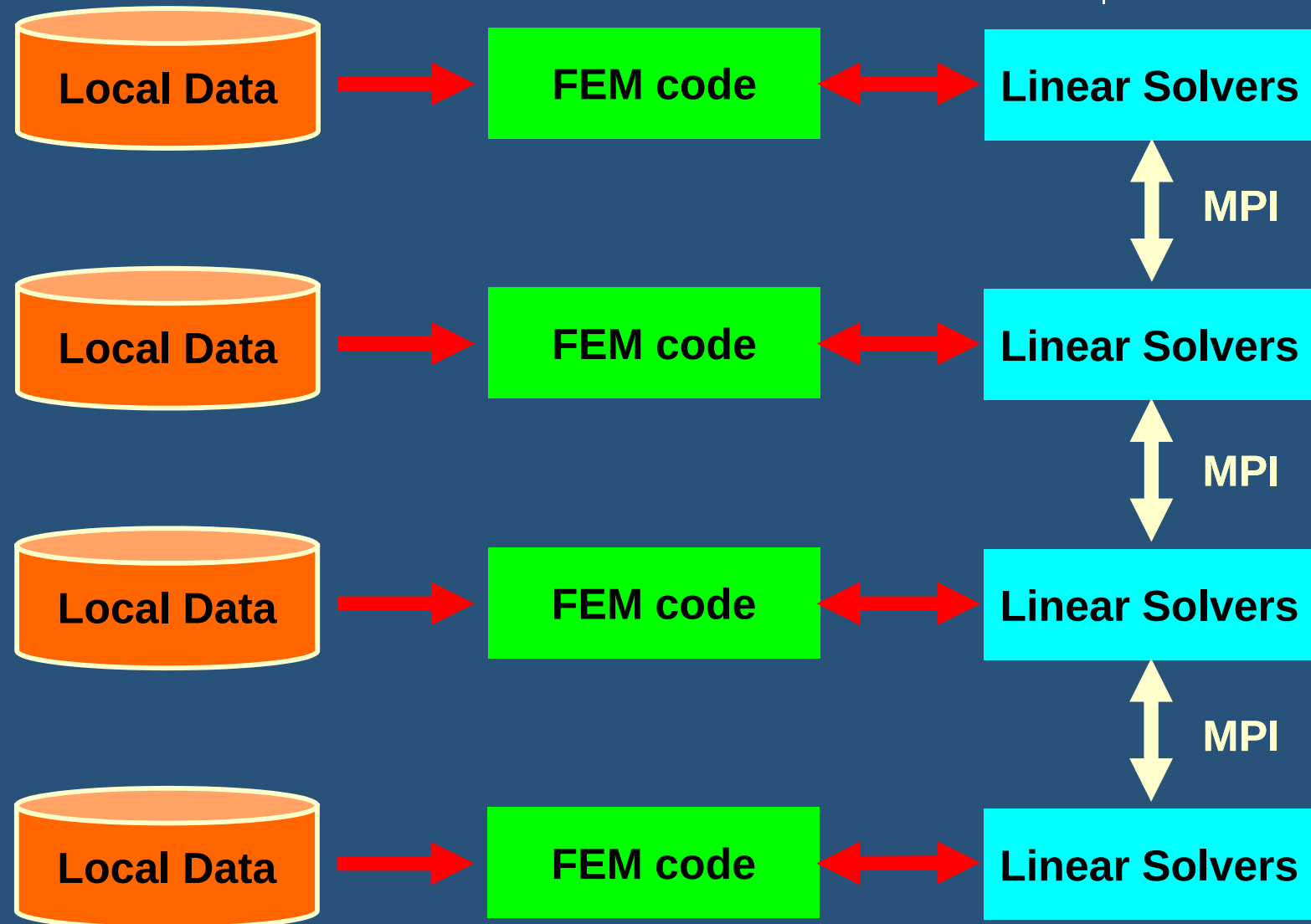
# We do not need communication during matrix assemble !!

- 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.



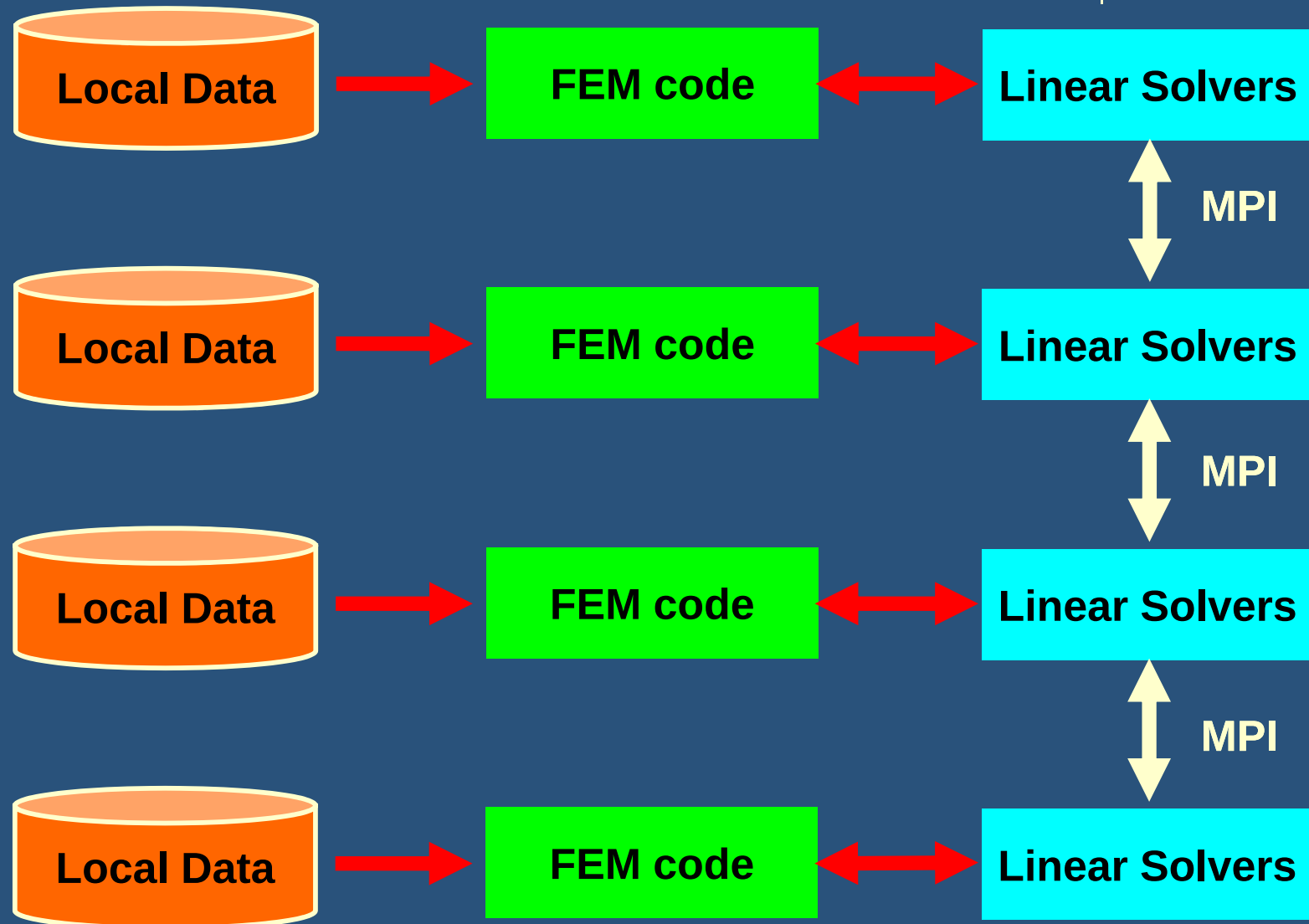
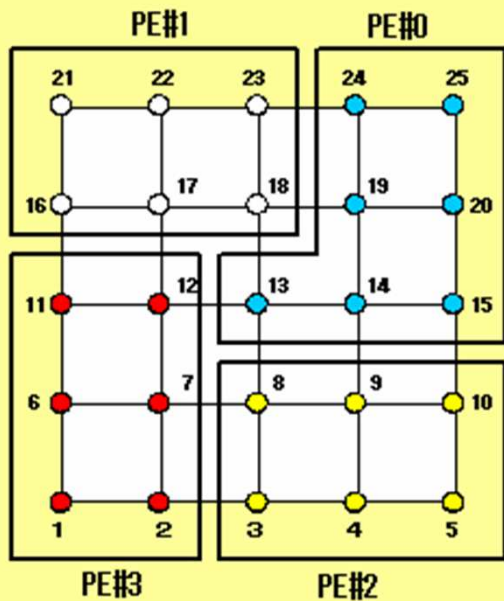
# Parallel Computing in FEM

## SPMD: Single-Program Multiple-Data



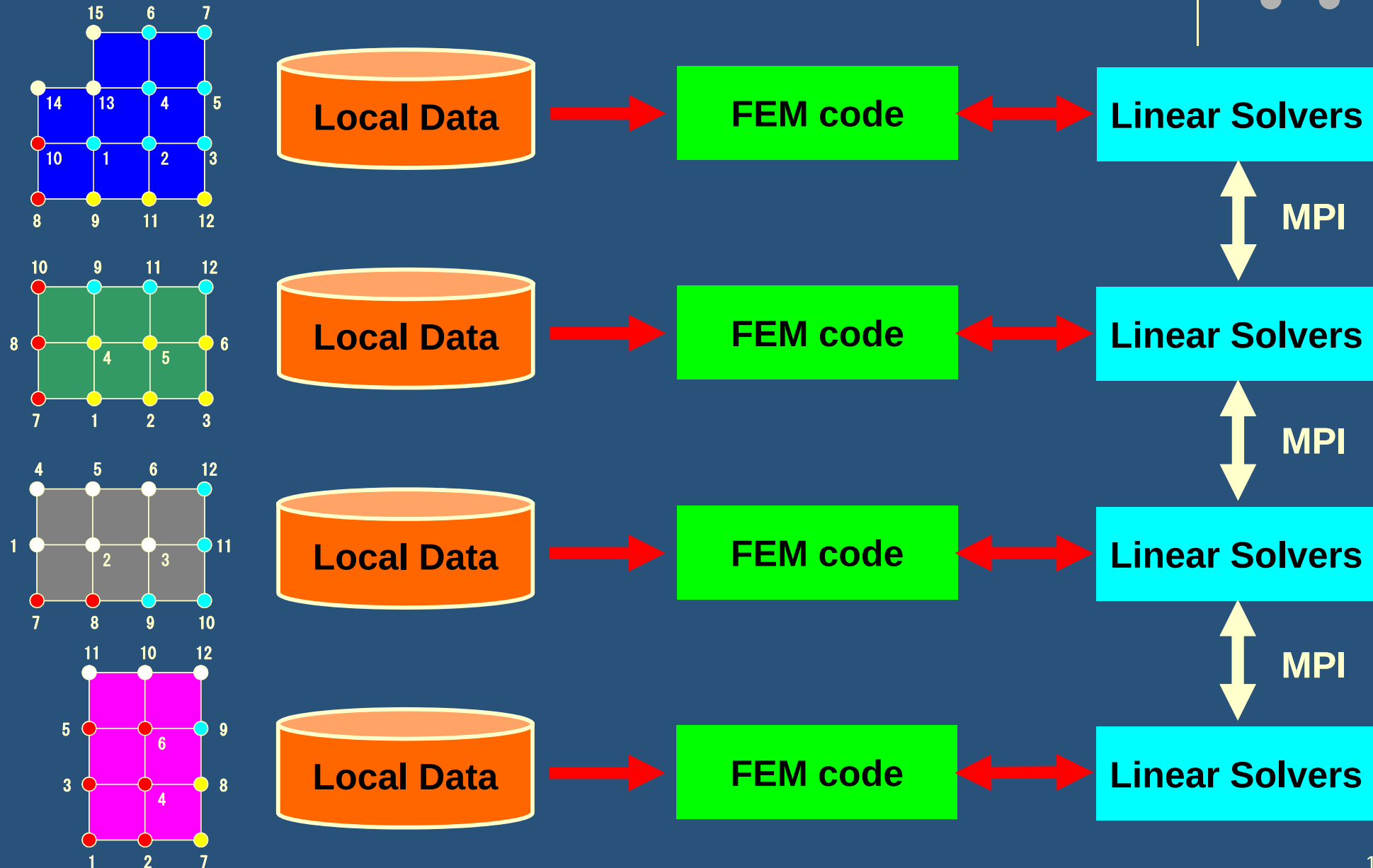
# Parallel Computing in FEM

## SPMD: Single-Program Multiple-Data



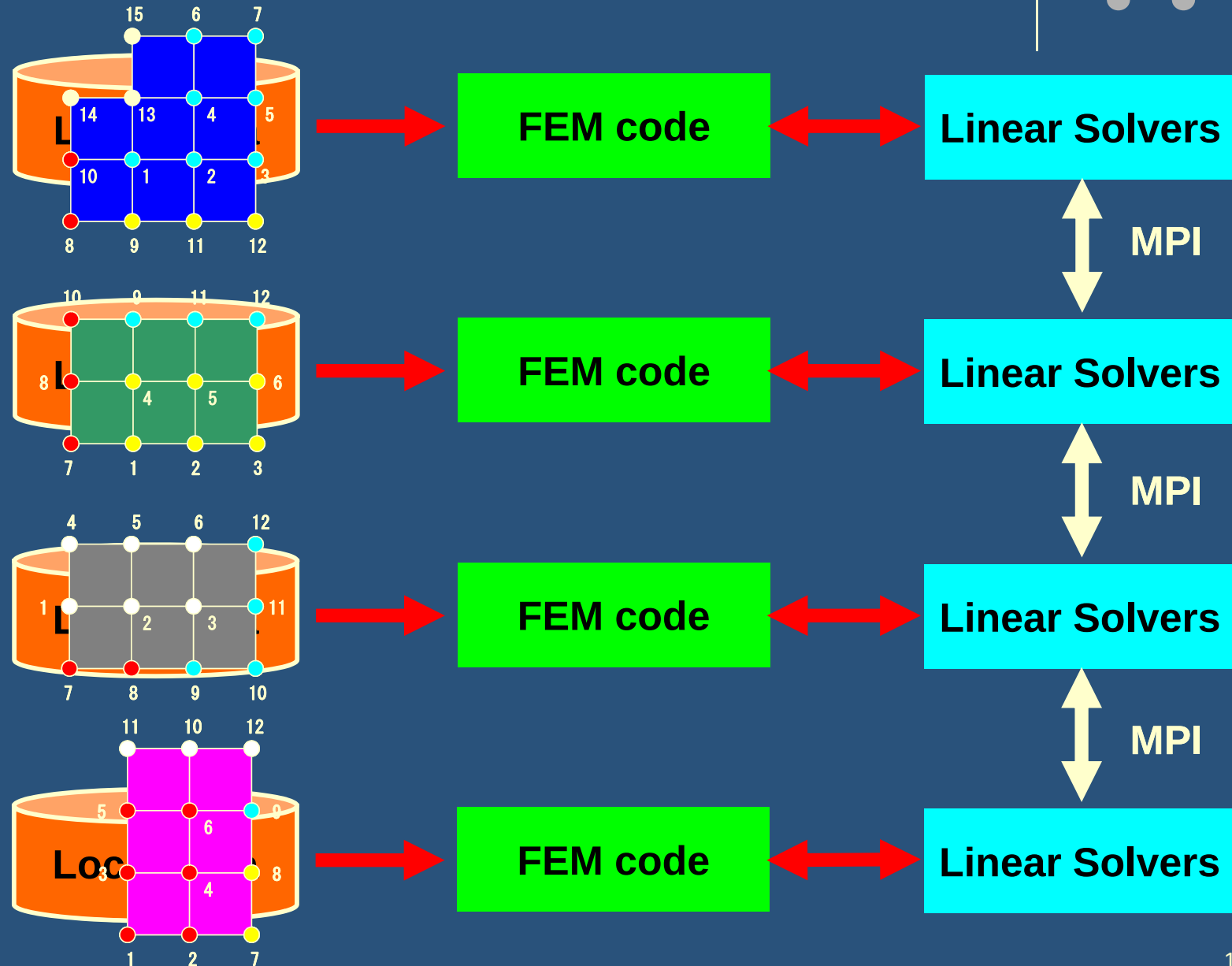
# Parallel Computing in FEM

## SPMD: Single-Program Multiple-Data



# Parallel Computing in FEM

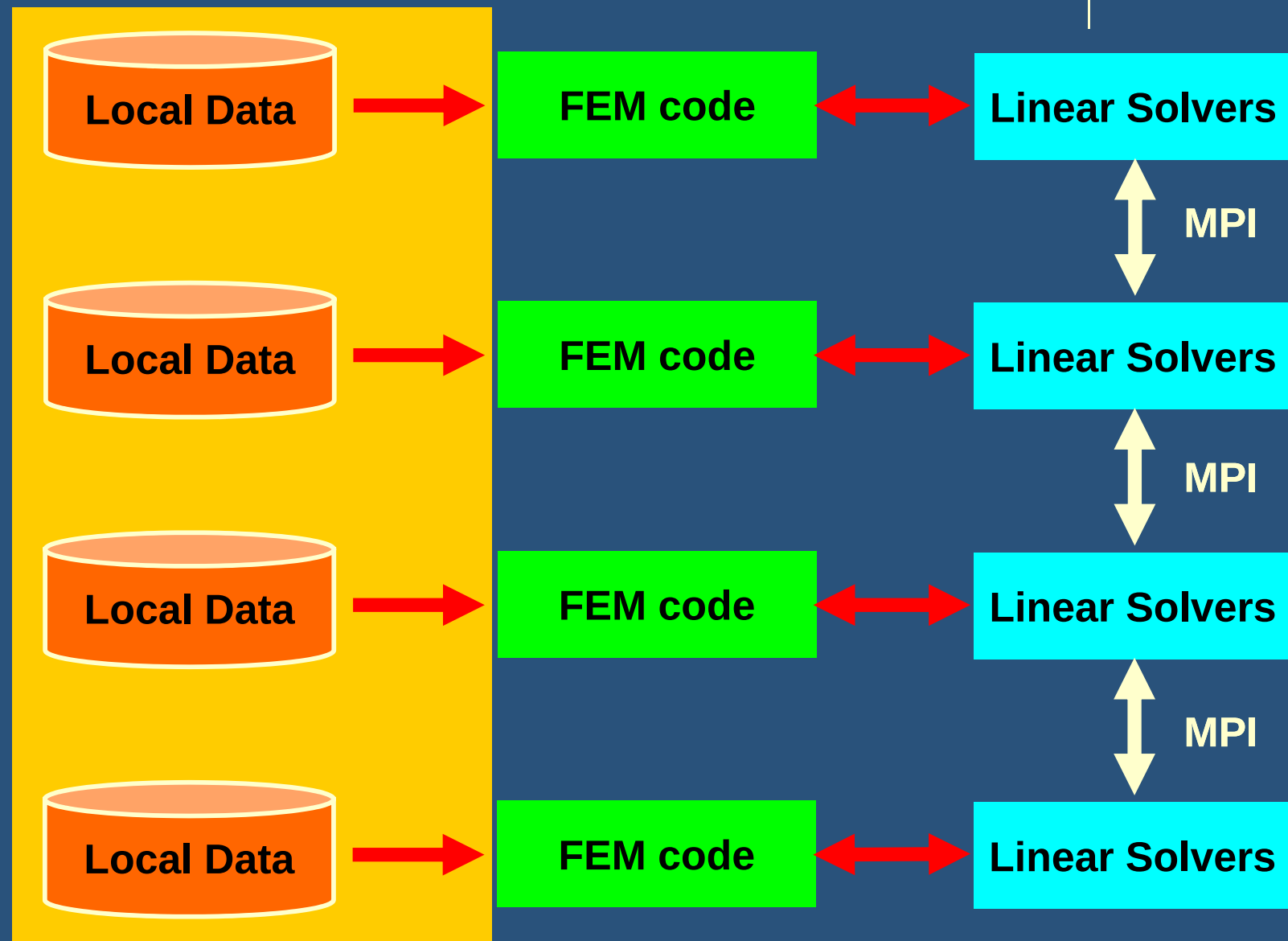
## SPMD: Single-Program Multiple-Data



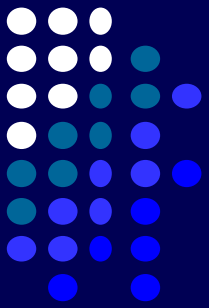


# Parallel Computing in FEM

## SPMD: Single-Program Multiple-Data

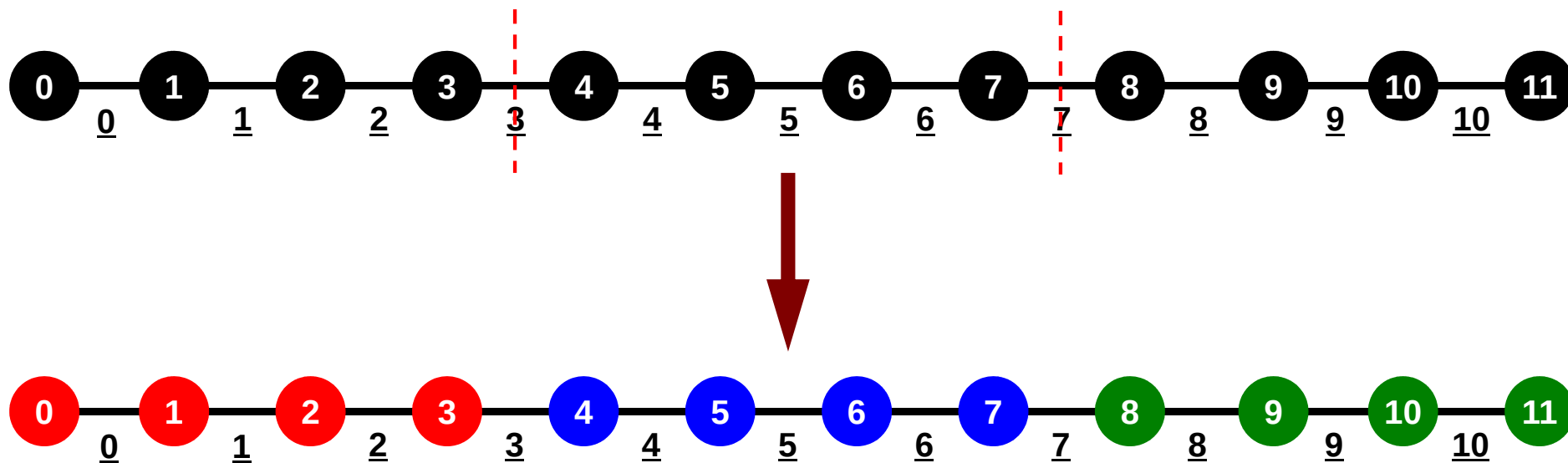


# What is Communications ?



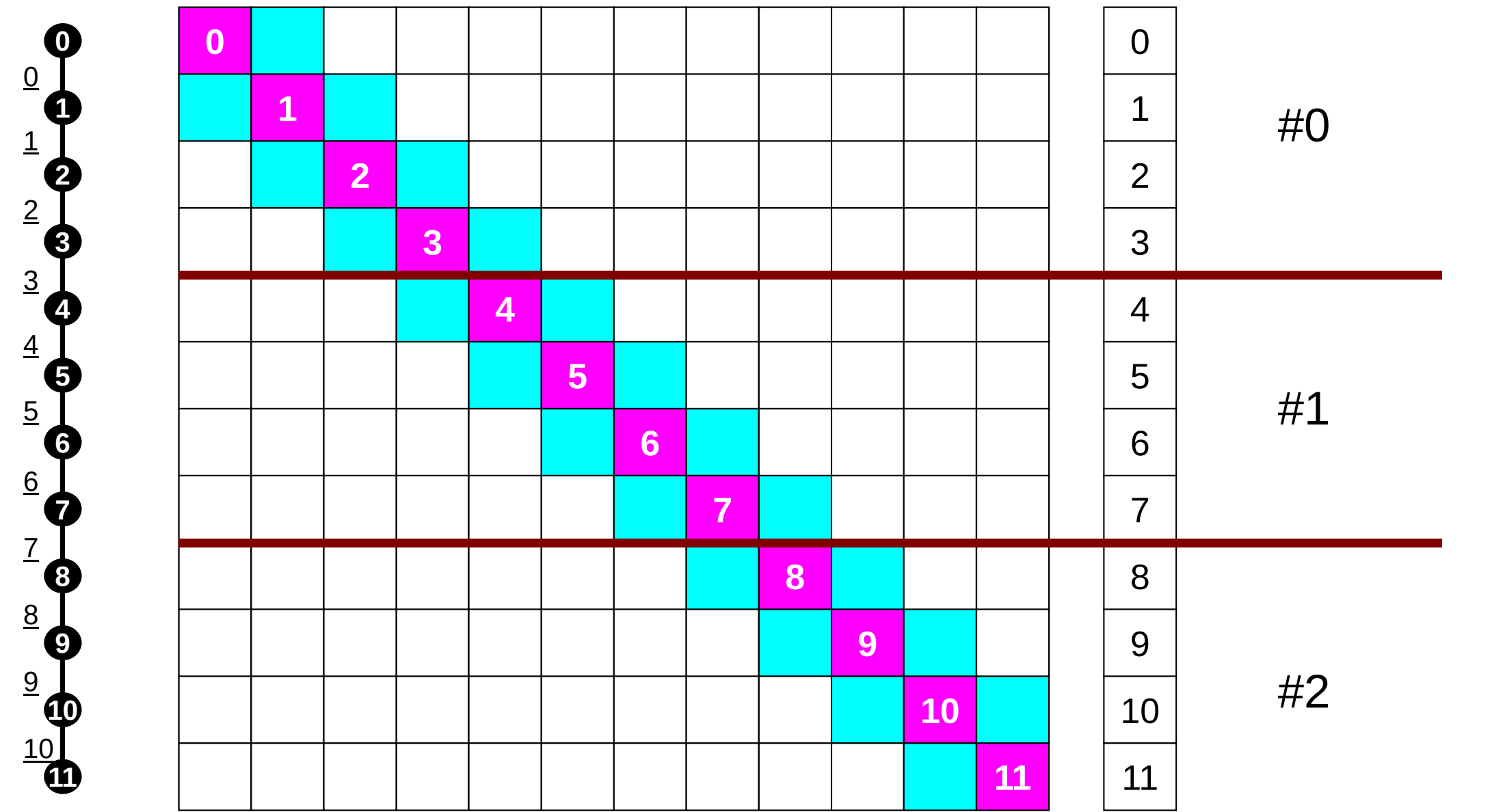
- to get information of “external nodes” from external partitions (local data)
- “Communication tables” contain the information

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





# # “Internal Nodes” should be balanced



# Matrices are incomplete !

- 0
- 1
- 2
- 3

|   |   |   |   |  |  |  |  |  |  |  |  |
|---|---|---|---|--|--|--|--|--|--|--|--|
| 0 |   |   |   |  |  |  |  |  |  |  |  |
|   | 1 |   |   |  |  |  |  |  |  |  |  |
|   |   | 2 |   |  |  |  |  |  |  |  |  |
|   |   |   | 3 |  |  |  |  |  |  |  |  |

#0

- 4
- 5
- 6
- 7

|  |  |  |  |   |   |   |   |  |  |  |  |
|--|--|--|--|---|---|---|---|--|--|--|--|
|  |  |  |  | 4 |   |   |   |  |  |  |  |
|  |  |  |  |   | 5 |   |   |  |  |  |  |
|  |  |  |  |   |   | 6 |   |  |  |  |  |
|  |  |  |  |   |   |   | 7 |  |  |  |  |

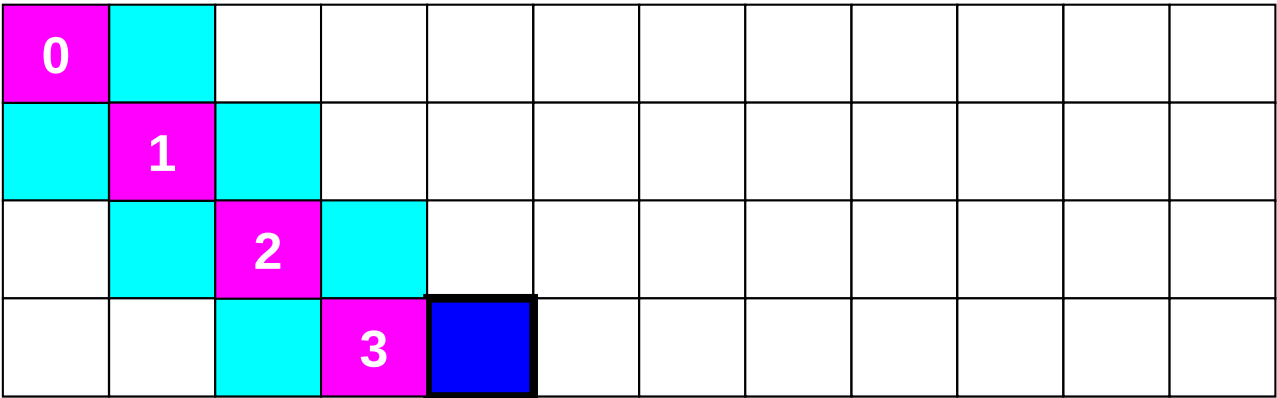
#1

- 8
- 9
- 10
- 11

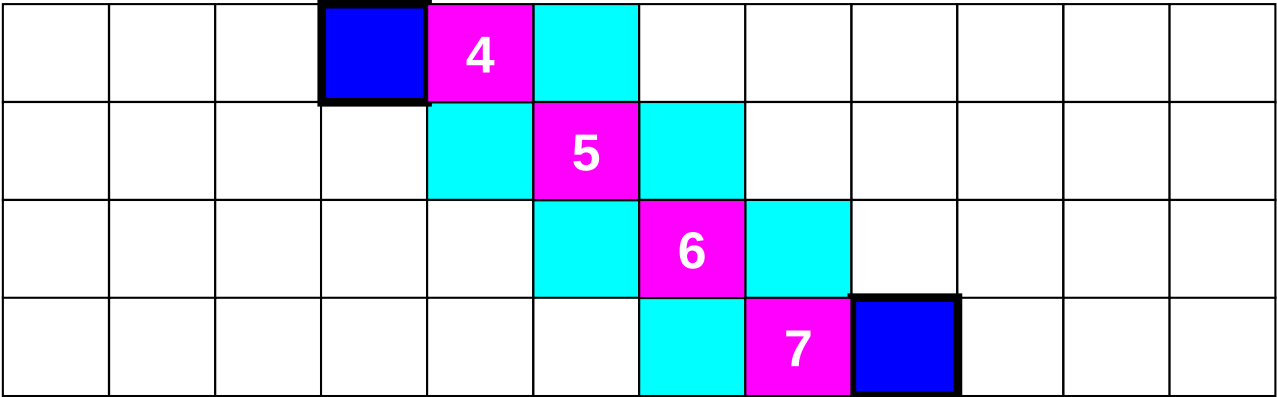
|  |  |  |  |  |  |  |  |   |   |    |    |
|--|--|--|--|--|--|--|--|---|---|----|----|
|  |  |  |  |  |  |  |  | 8 |   |    |    |
|  |  |  |  |  |  |  |  |   | 9 |    |    |
|  |  |  |  |  |  |  |  |   |   | 10 |    |
|  |  |  |  |  |  |  |  |   |   |    | 11 |

#2

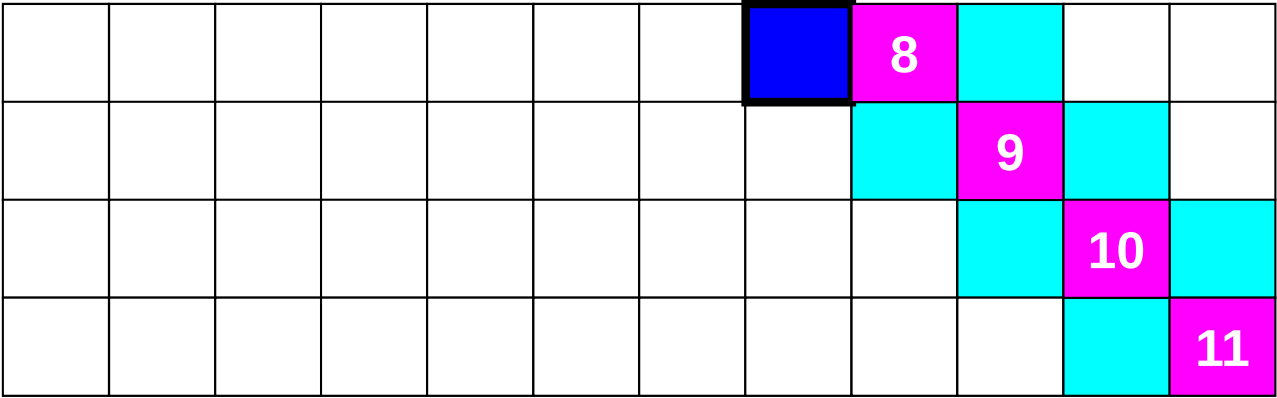
# Connected Elements + External Nodes



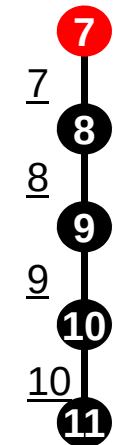
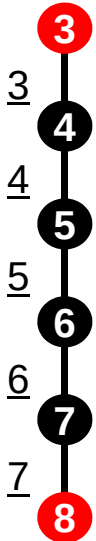
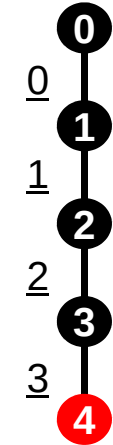
#0



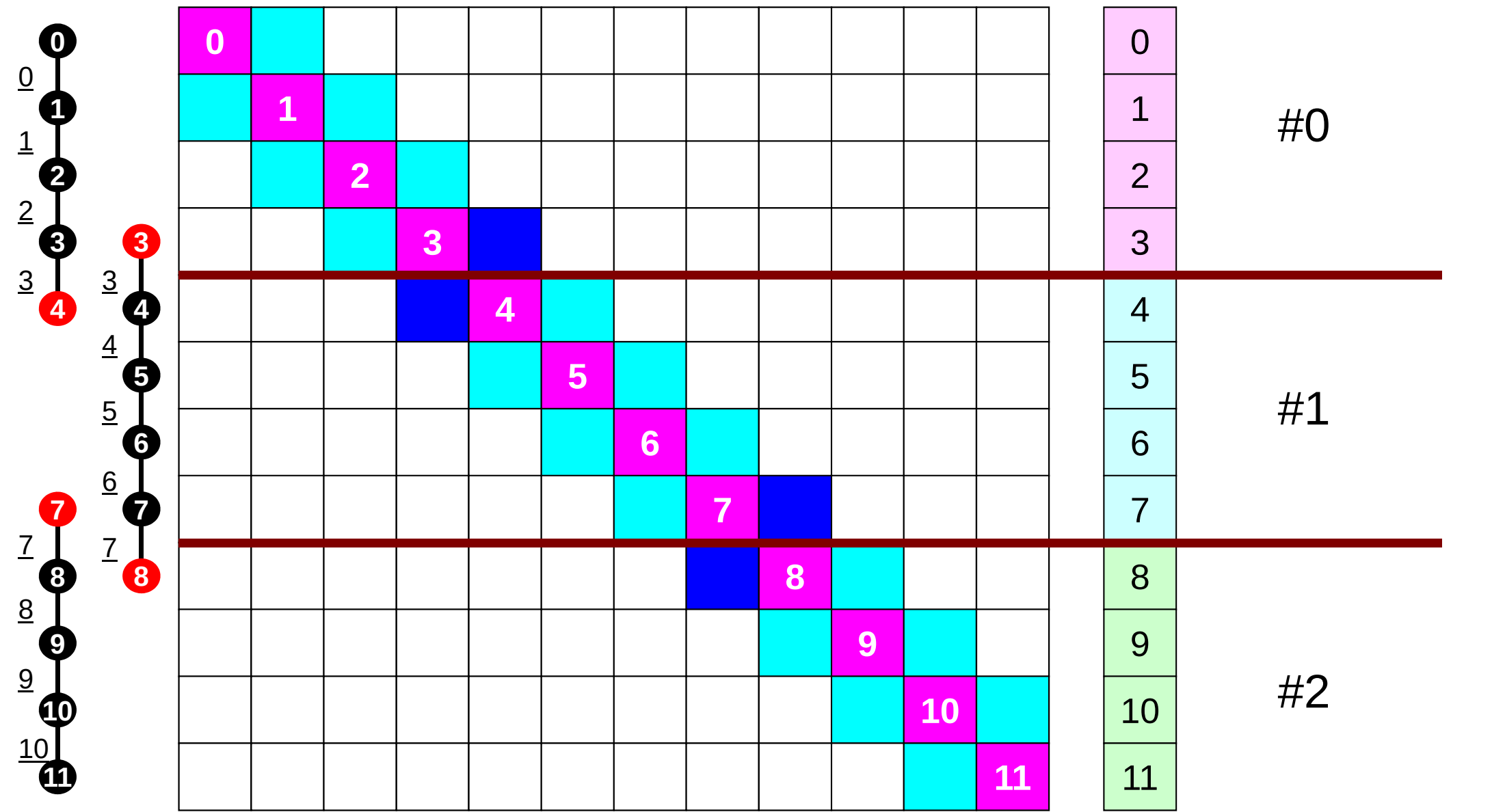
#1



#2

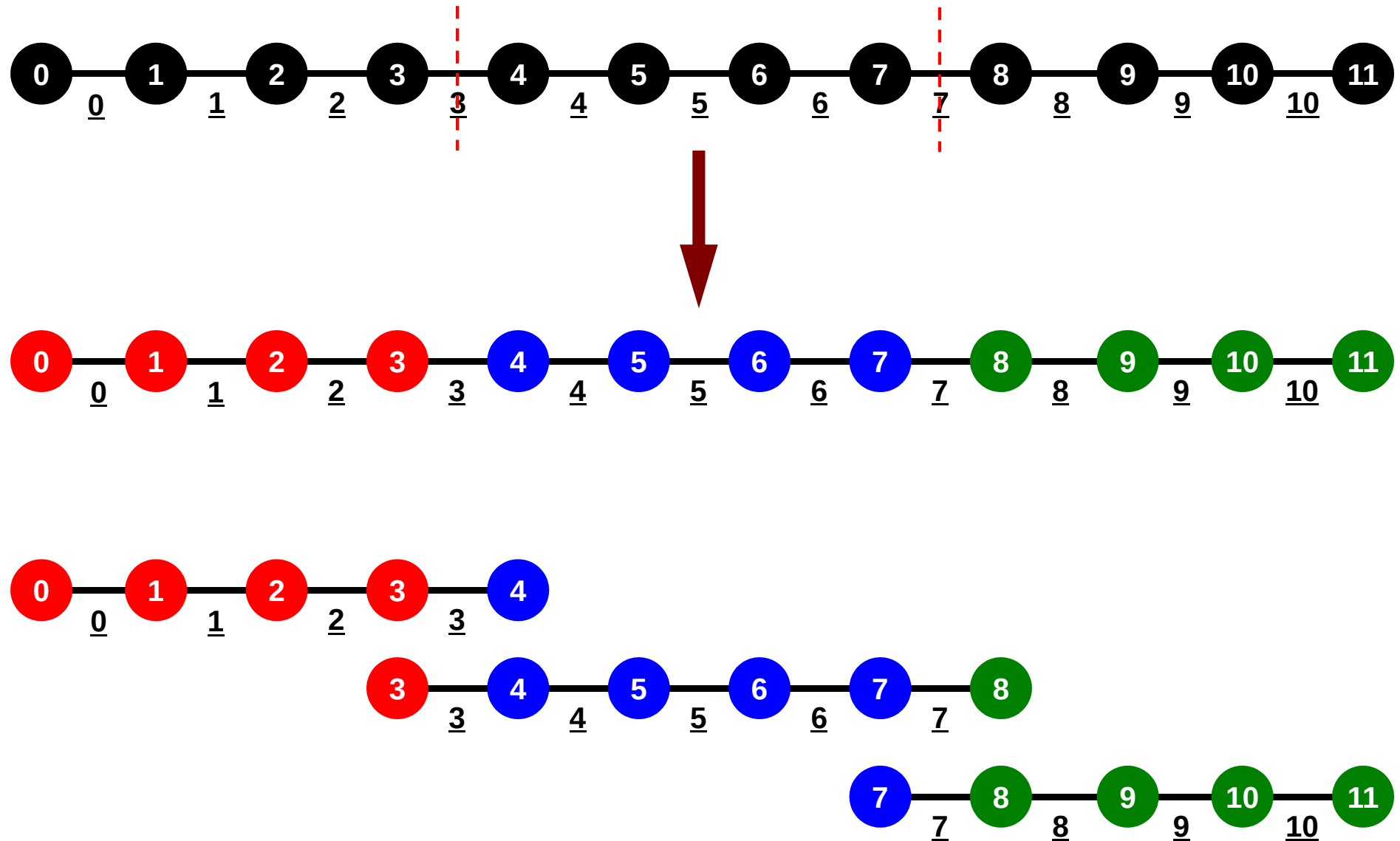


# 1D FEM: 12 nodes/11 elem's/3 domains



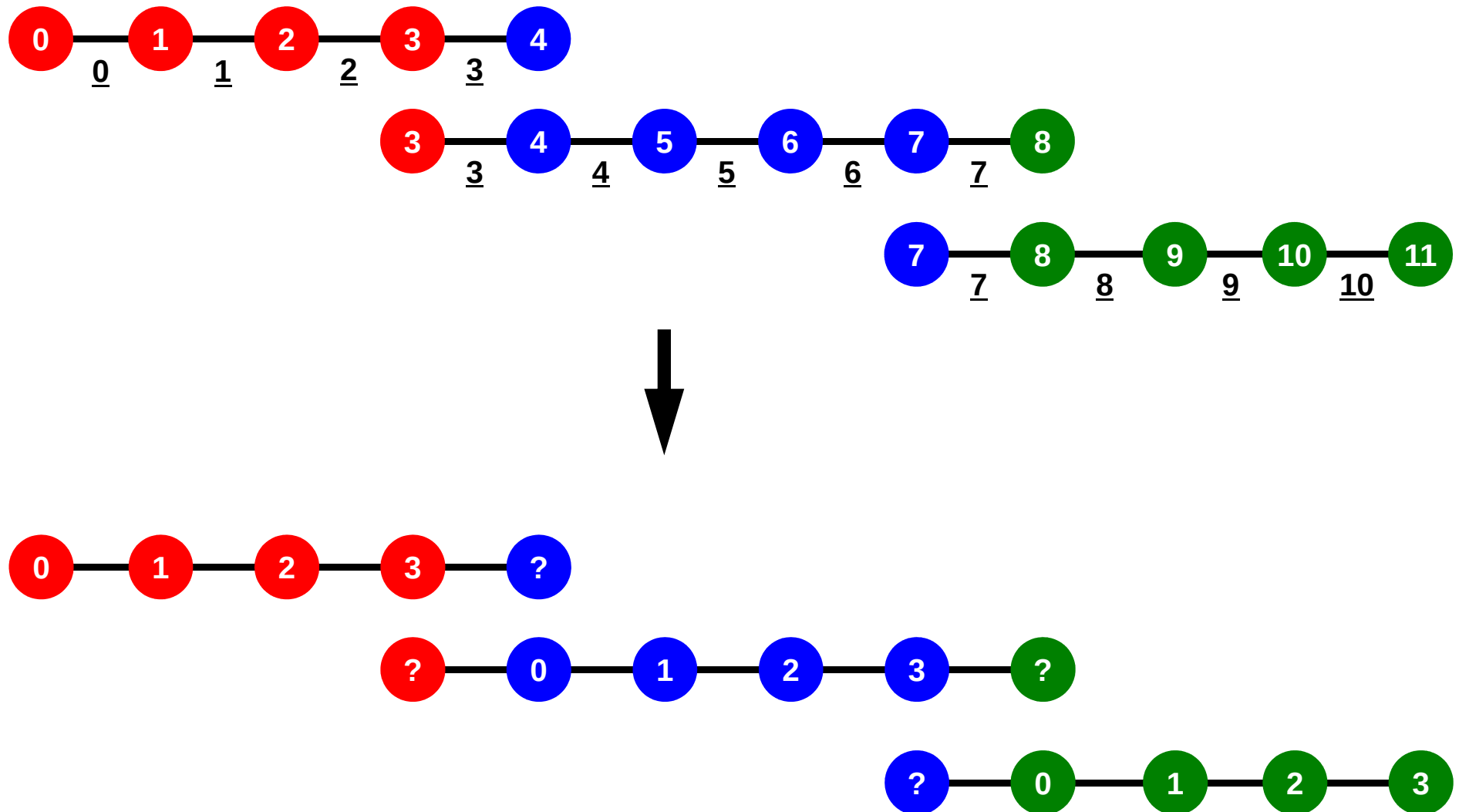


# 1D FEM: 12 nodes/11 elem's/3 domains



# 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 ?

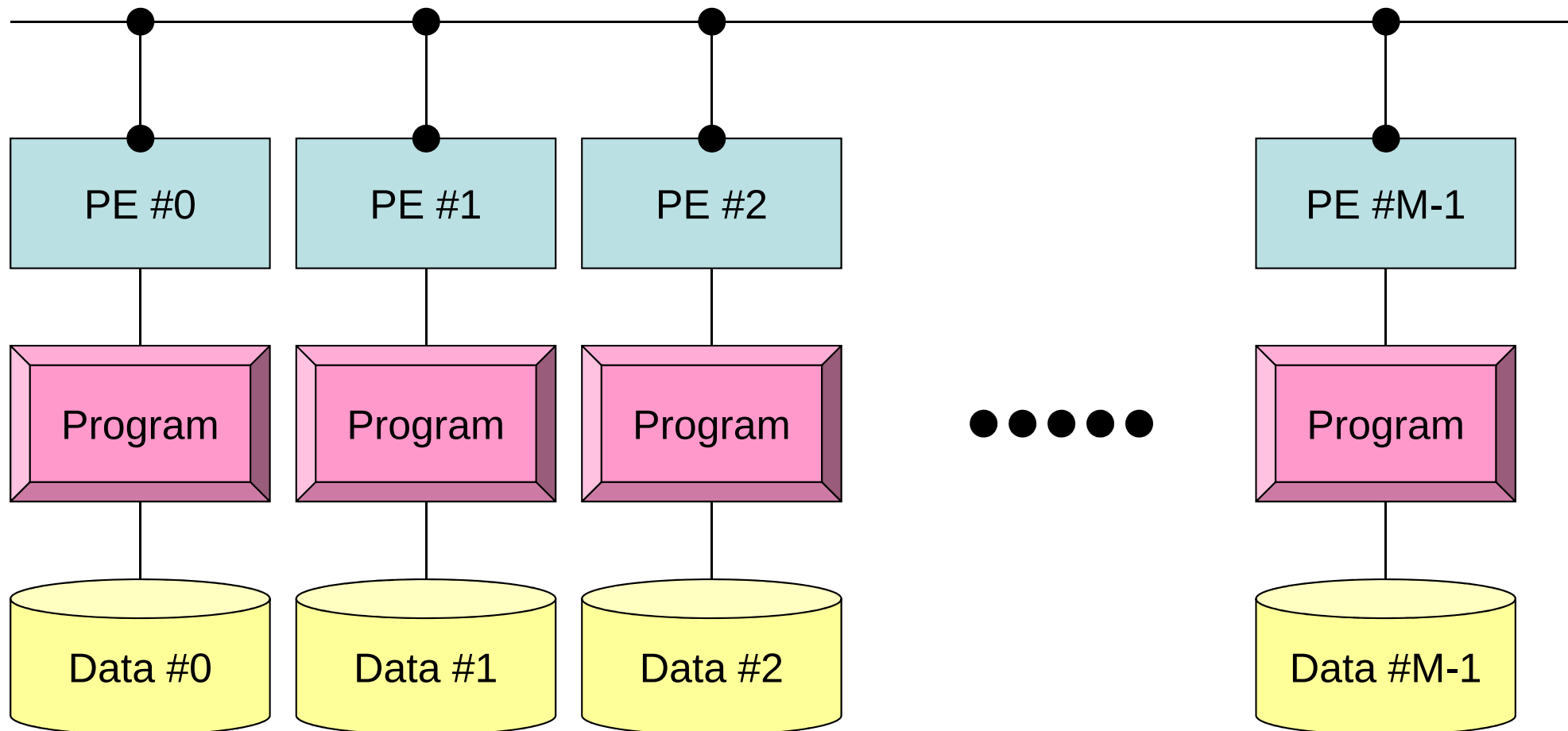


PE: Processing Element  
Processor, Domain, Process

# SPMD:

## Single Program Multiple Data

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



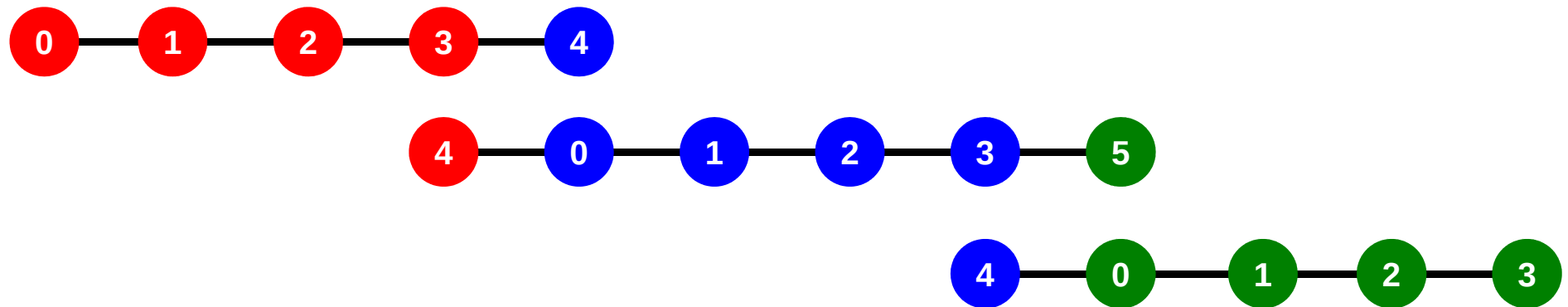
Each process does same operation for different data

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

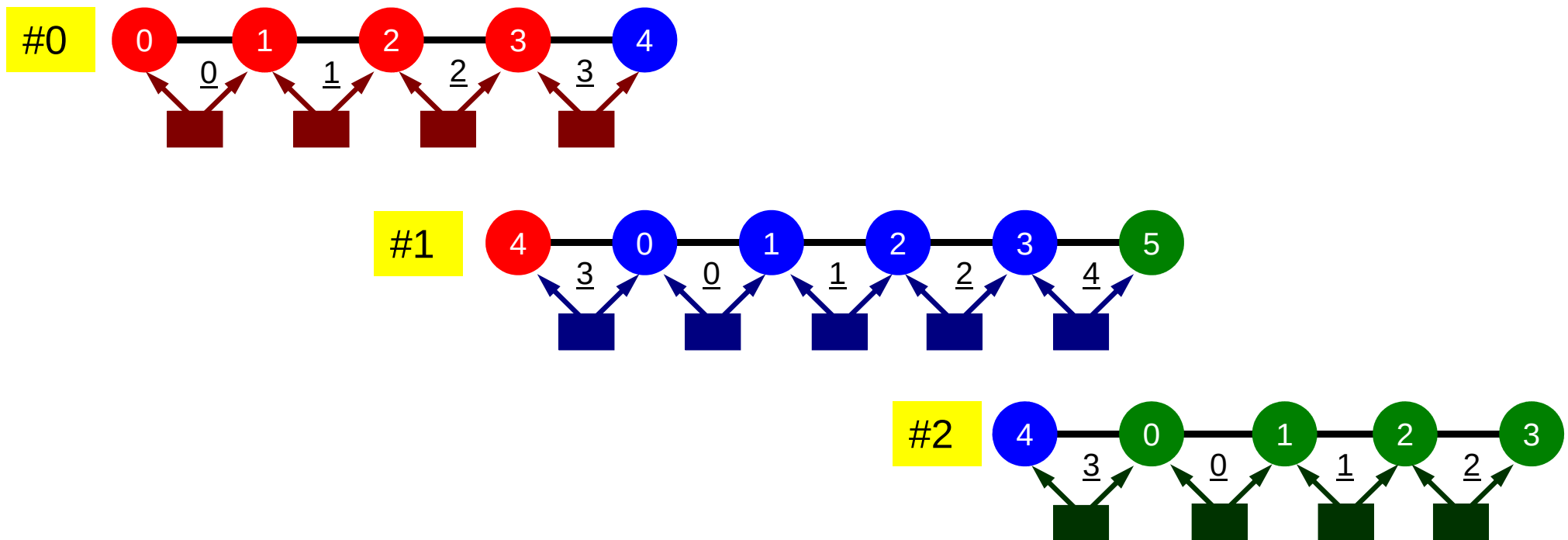
Numbering of external nodes:  $N+1$ ,  $N+2$  ( $N, N+1$ )



# 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, ...
  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.

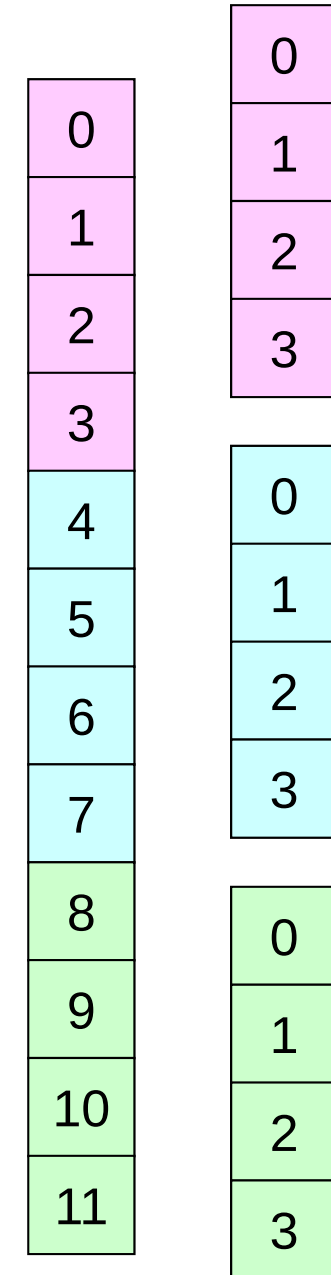
$$[\mathbf{M}] = \begin{bmatrix} D_1 & 0 & \dots & 0 & 0 \\ 0 & D_2 & & 0 & 0 \\ \dots & & \dots & & \dots \\ 0 & 0 & & D_{N-1} & 0 \\ 0 & 0 & \dots & 0 & D_N \end{bmatrix}$$

# Preconditioning, DAXPY

Local Operations by Only Internal Points: Parallel Processing is possible

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

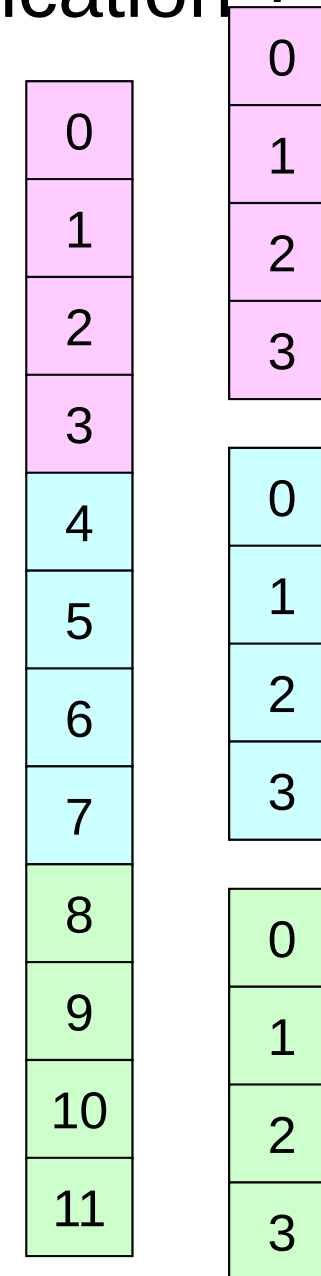




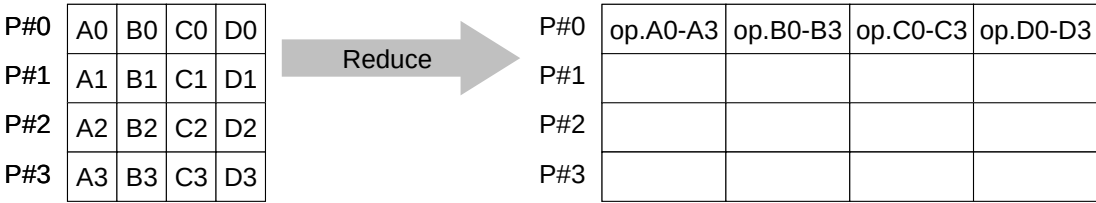
# Dot Products

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;
```



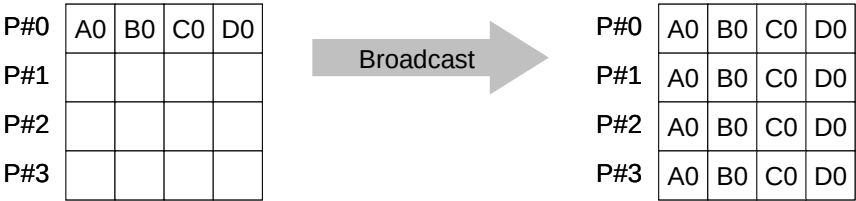
# MPI\_Reduce



- 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
  - **count** int I number of elements in send/receive buffer
  - **datatype** MPI\_Datatype I data type of elements of send/recive buffer
    - FORTTRAN 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

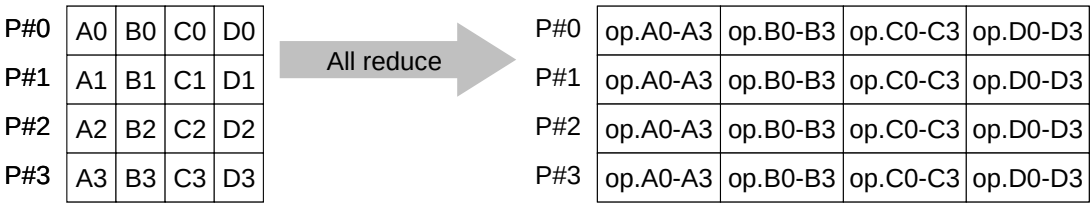


# MPI\_Bcast



- 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
    - FORTAN 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

# MPI\_Allreduce



- 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

# “op” of MPI\_Reduce/Allreduce

C

## **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

# Preconditioned CG Solver

```

Compute  $\mathbf{r}^{(0)} = \mathbf{b} - [\mathbf{A}]\mathbf{x}^{(0)}$ 
for i= 1, 2, ...
  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

```

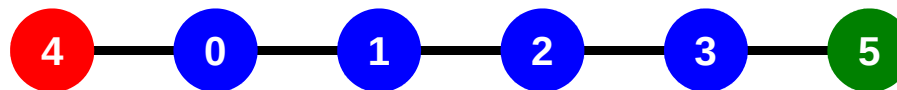
- Preconditioning
  - Diagonal Scaling/Point Jacobi
- Parallel operations are required in
  - Dot Products
  - Mat-Vec. Multiplication
    - SpMV: Sparse Mat-Vec. Mult.

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

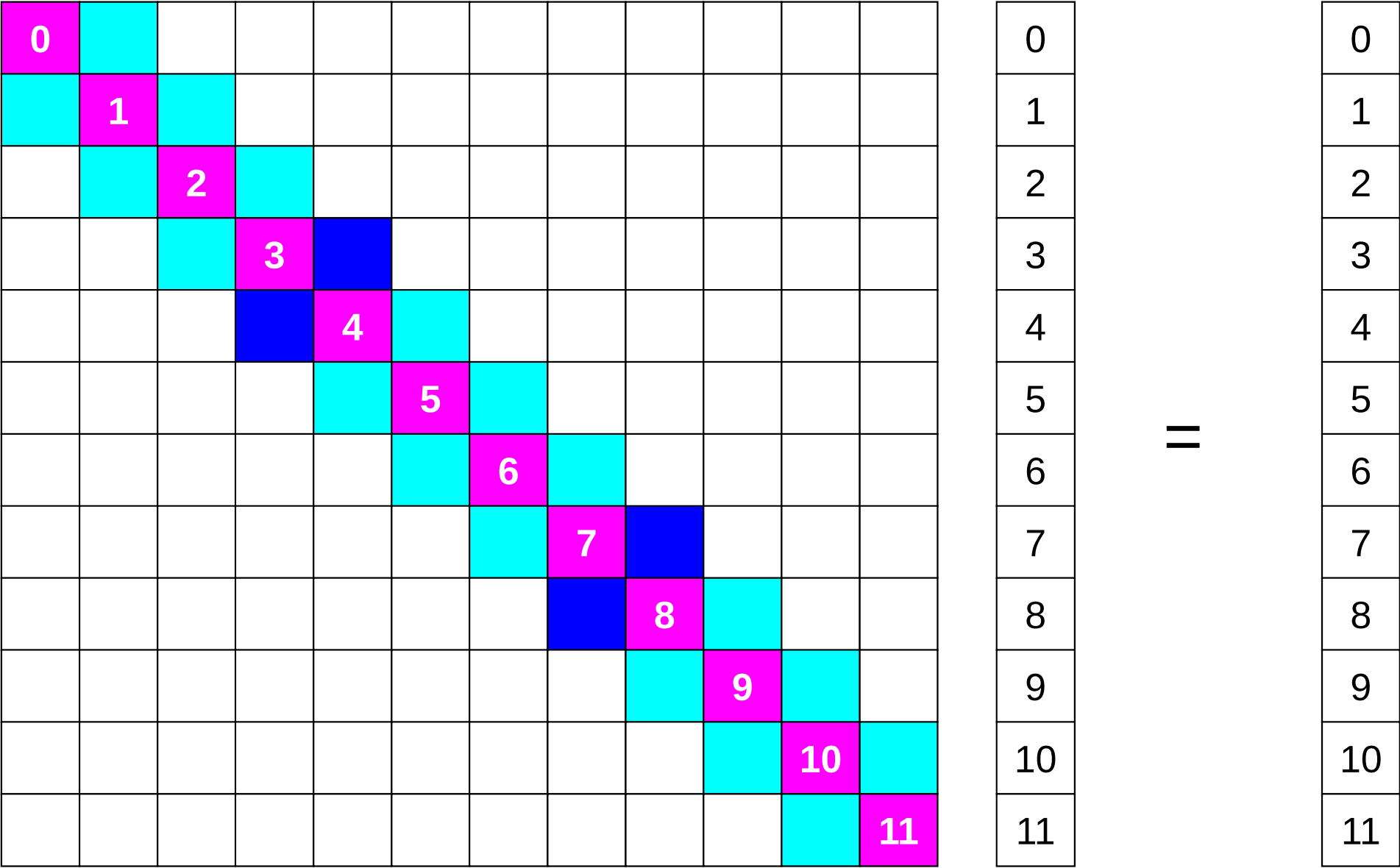
# Matrix-Vector Products

## Values at External Points: P-to-P Communication

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

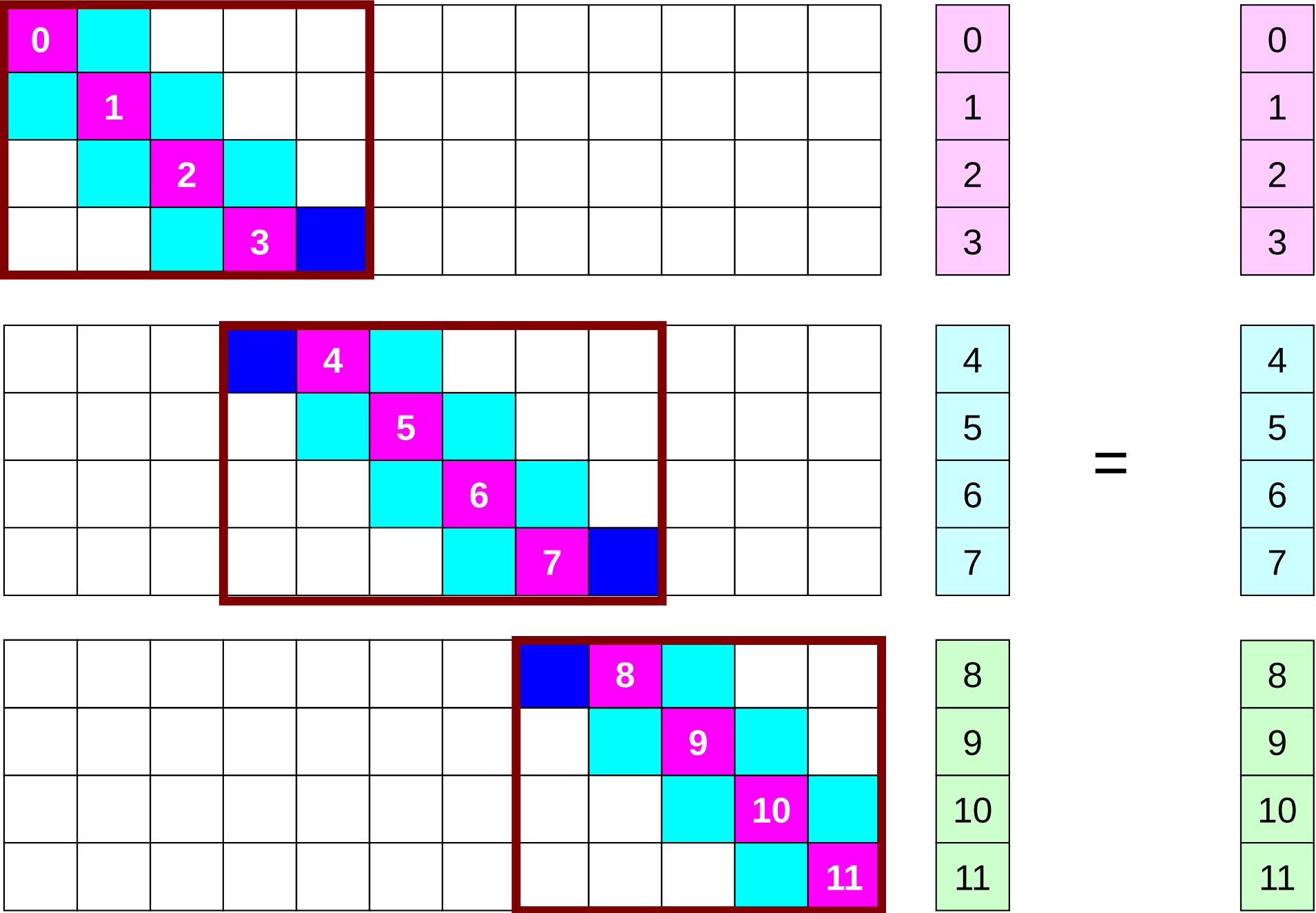


# Mat-Vec Products: Local Op. Possible

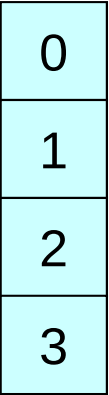
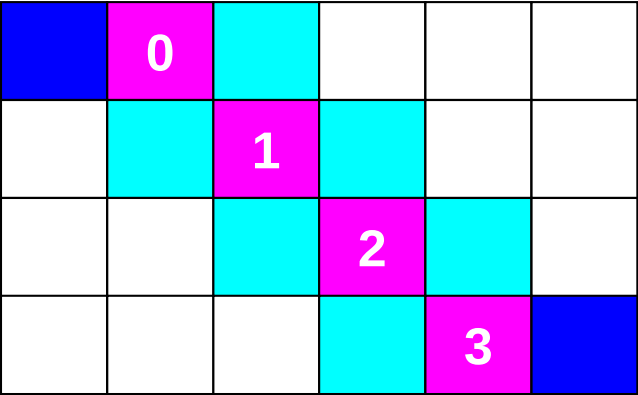
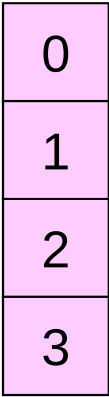
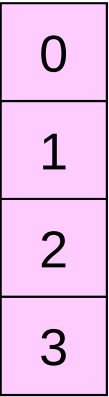
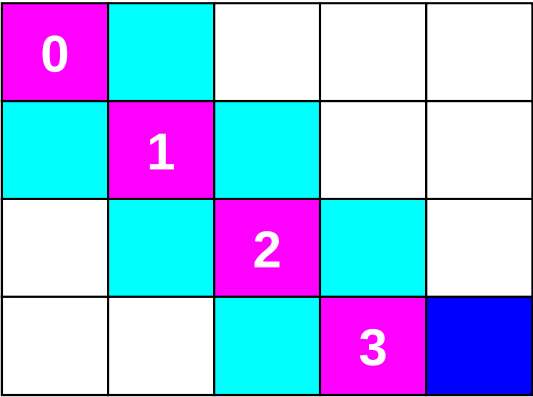




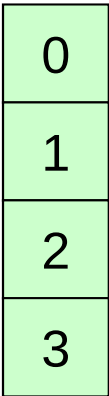
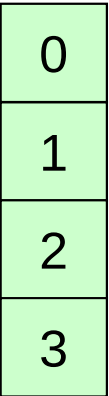
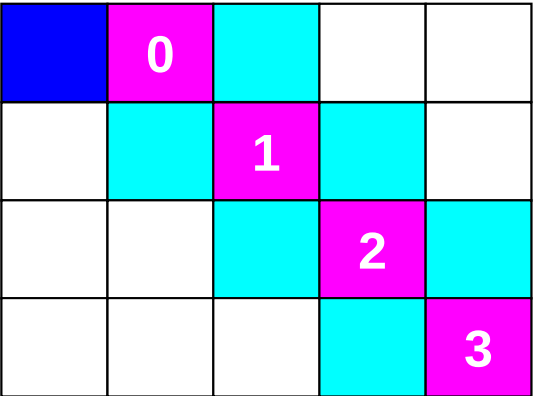
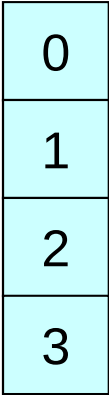
# Mat-Vec Products: Local Op. Possible



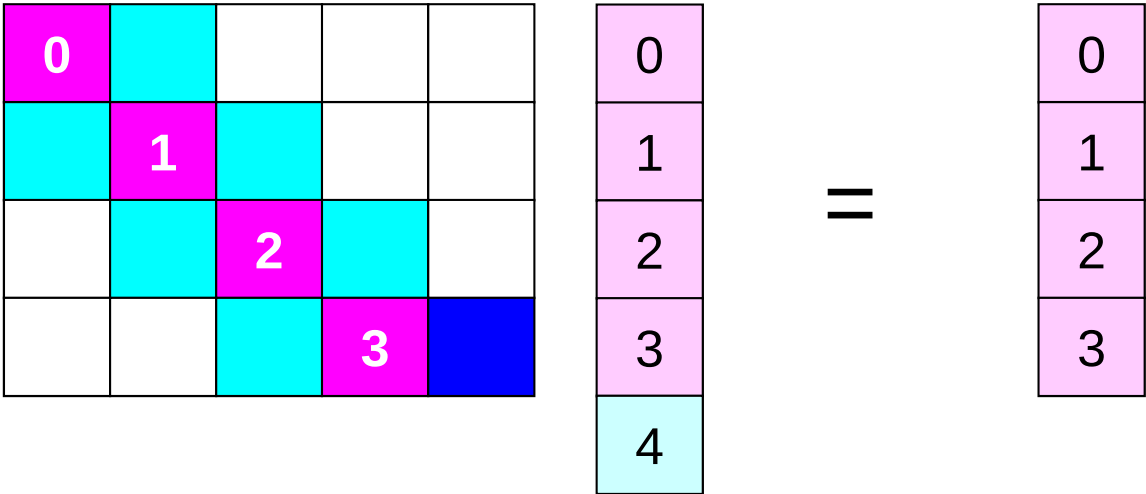
# Mat-Vec Products: Local Op. Possible



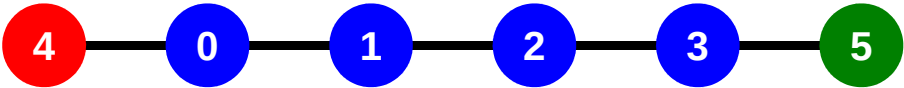
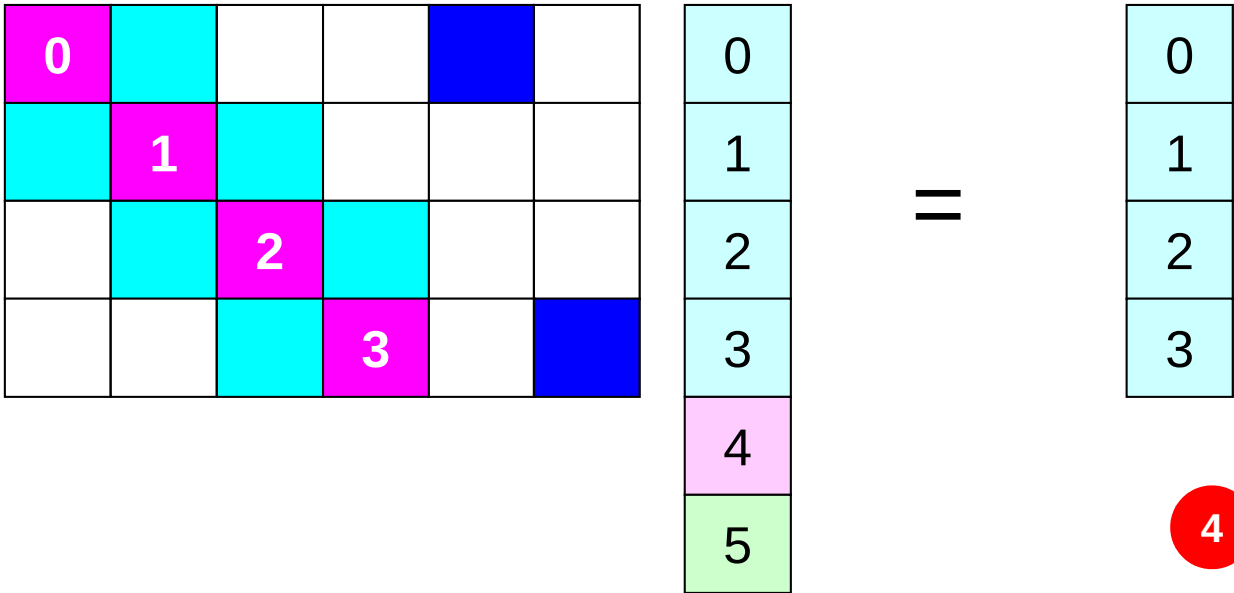
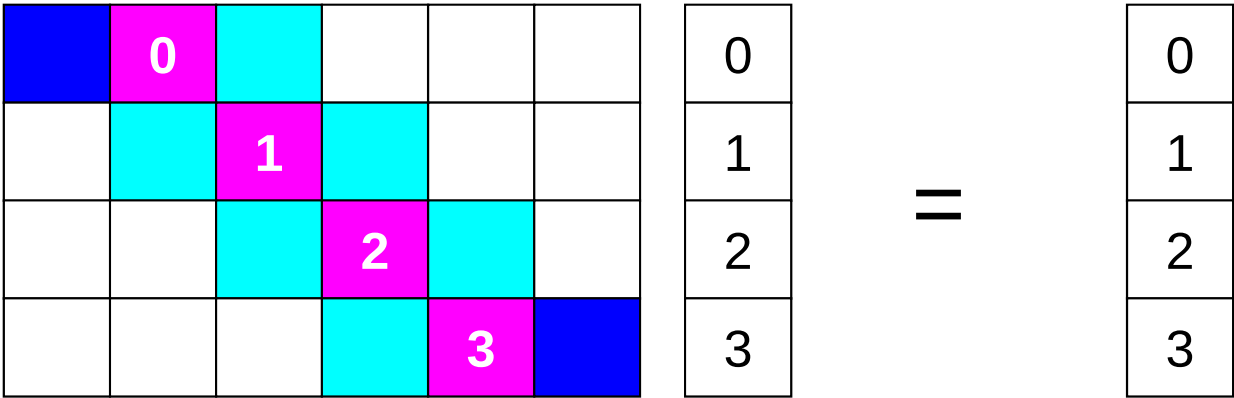
=



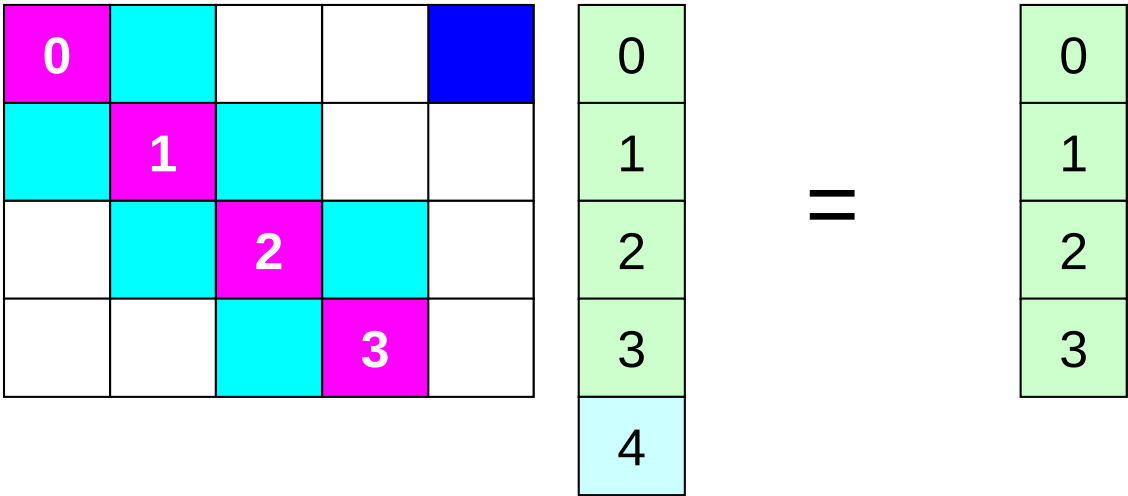
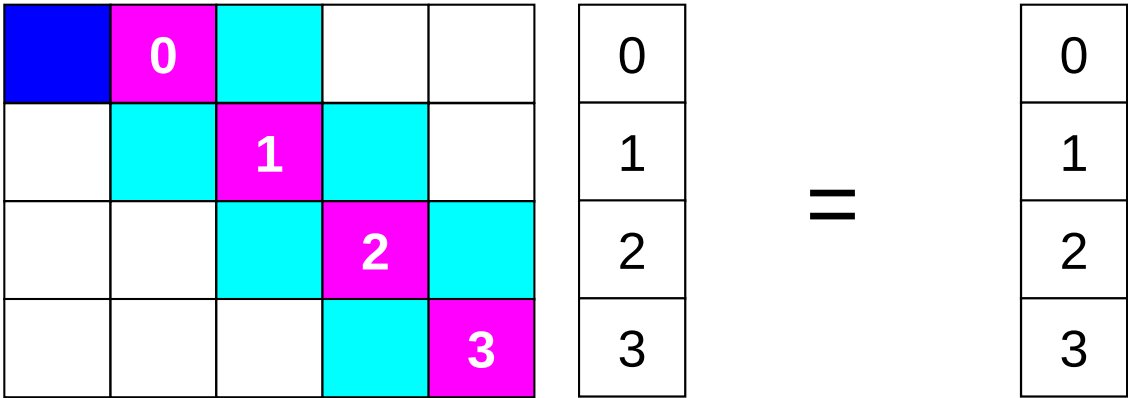
# Mat-Vec Products: Local Op. #0



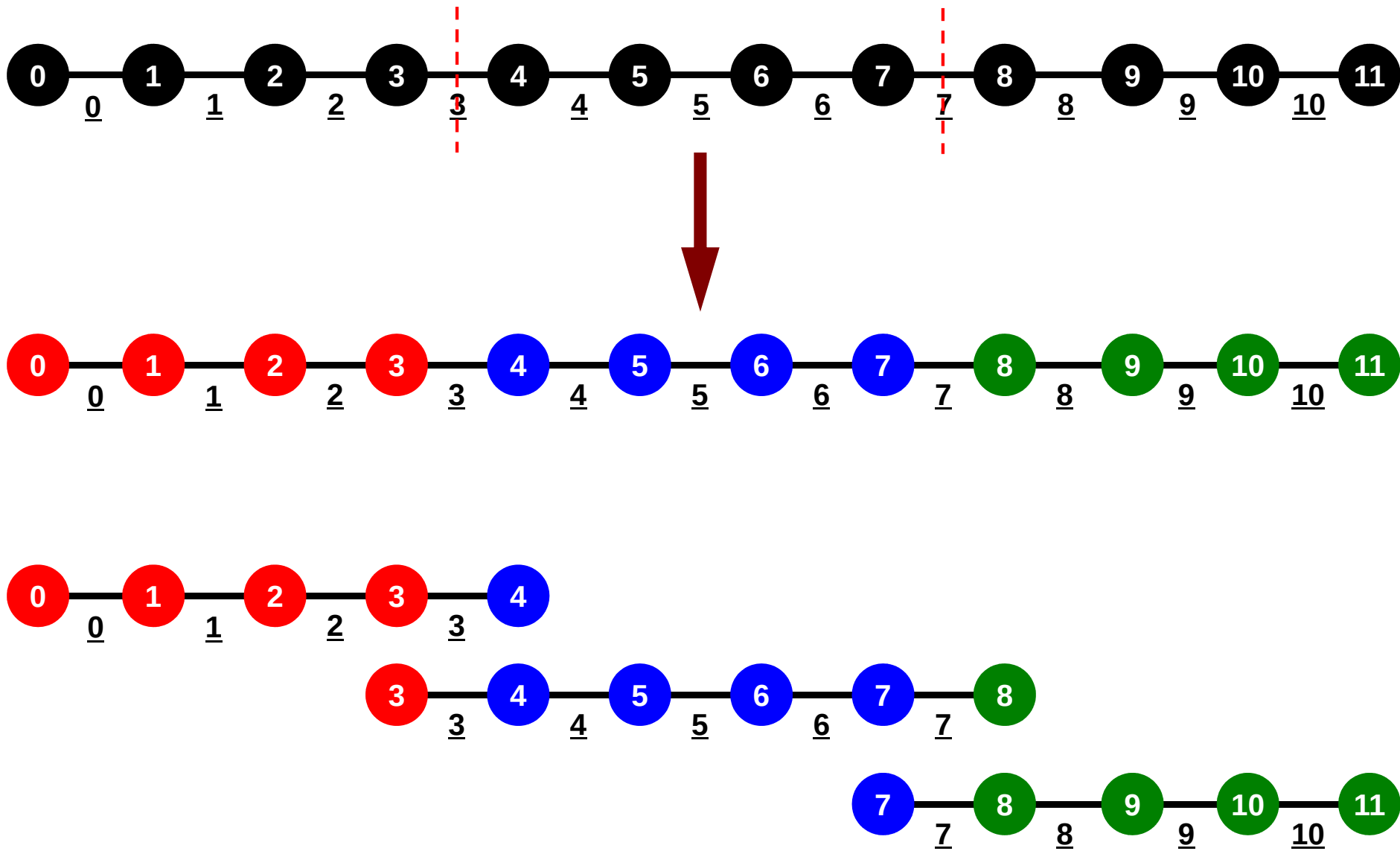
# Mat-Vec Products: Local Op. #1



# Mat-Vec Products: Local Op. #2

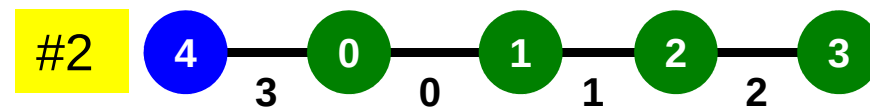
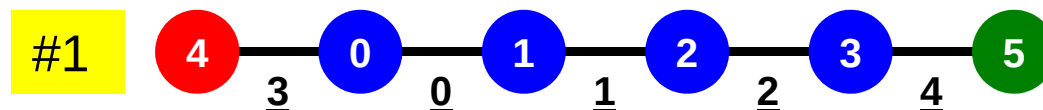
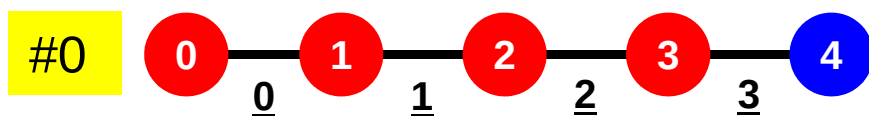


# 1D FEM: 12 nodes/11 elem's/3 domains



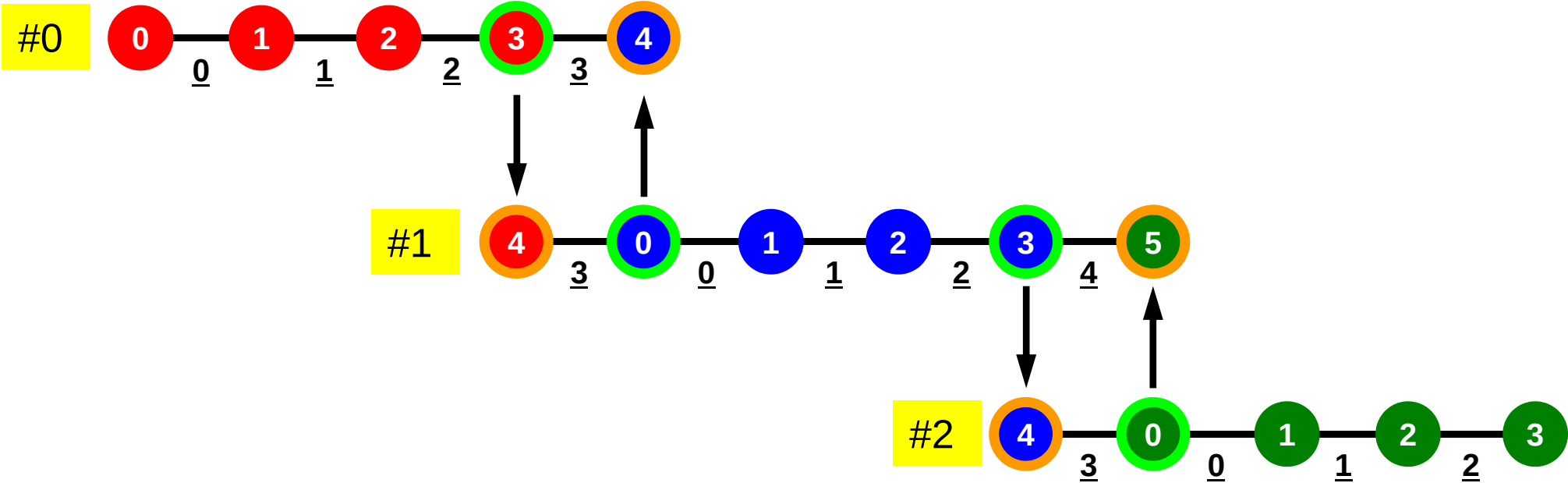
# 1D FEM: 12 nodes/11 elem's/3 domains

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



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

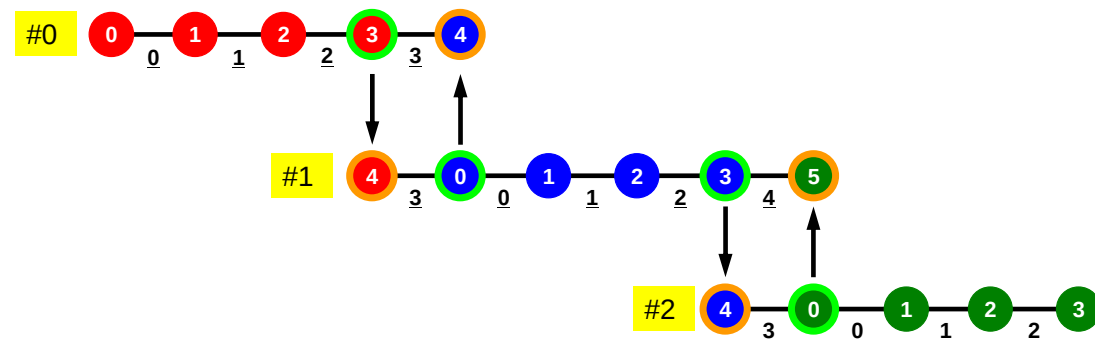
Internal/External Nodes





# What is Point-to-Point 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
- Point-to-Point
  - MPI\_Send, MPI\_Recv
  - Communication with limited processes
    - Neighbors
  - Application Area
    - FEM, FDM: Localized Method



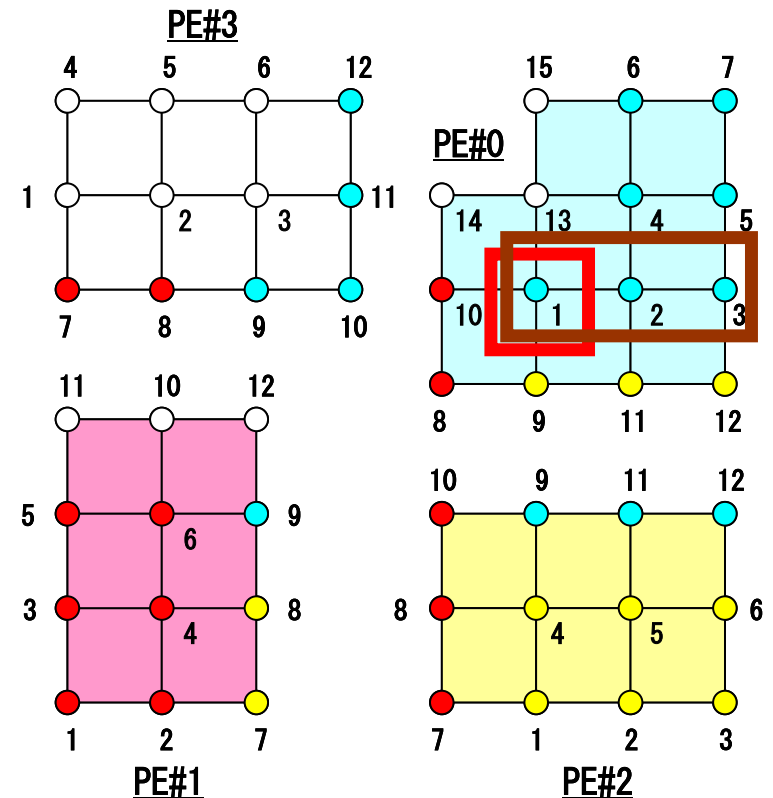
# SEND: sending from boundary nodes

## 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





# 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

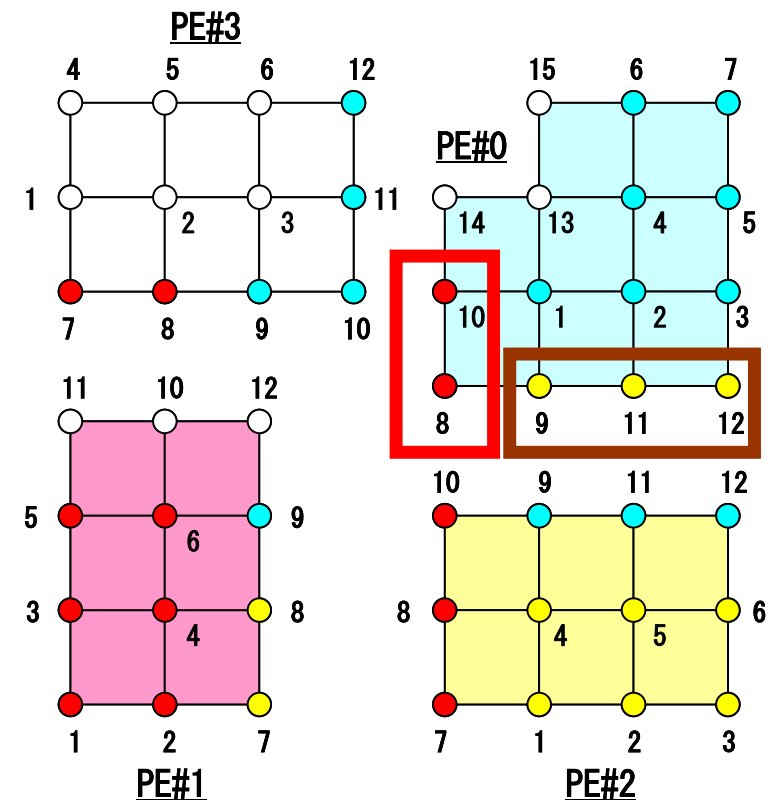
# RECV: receiving to external nodes

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

- 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



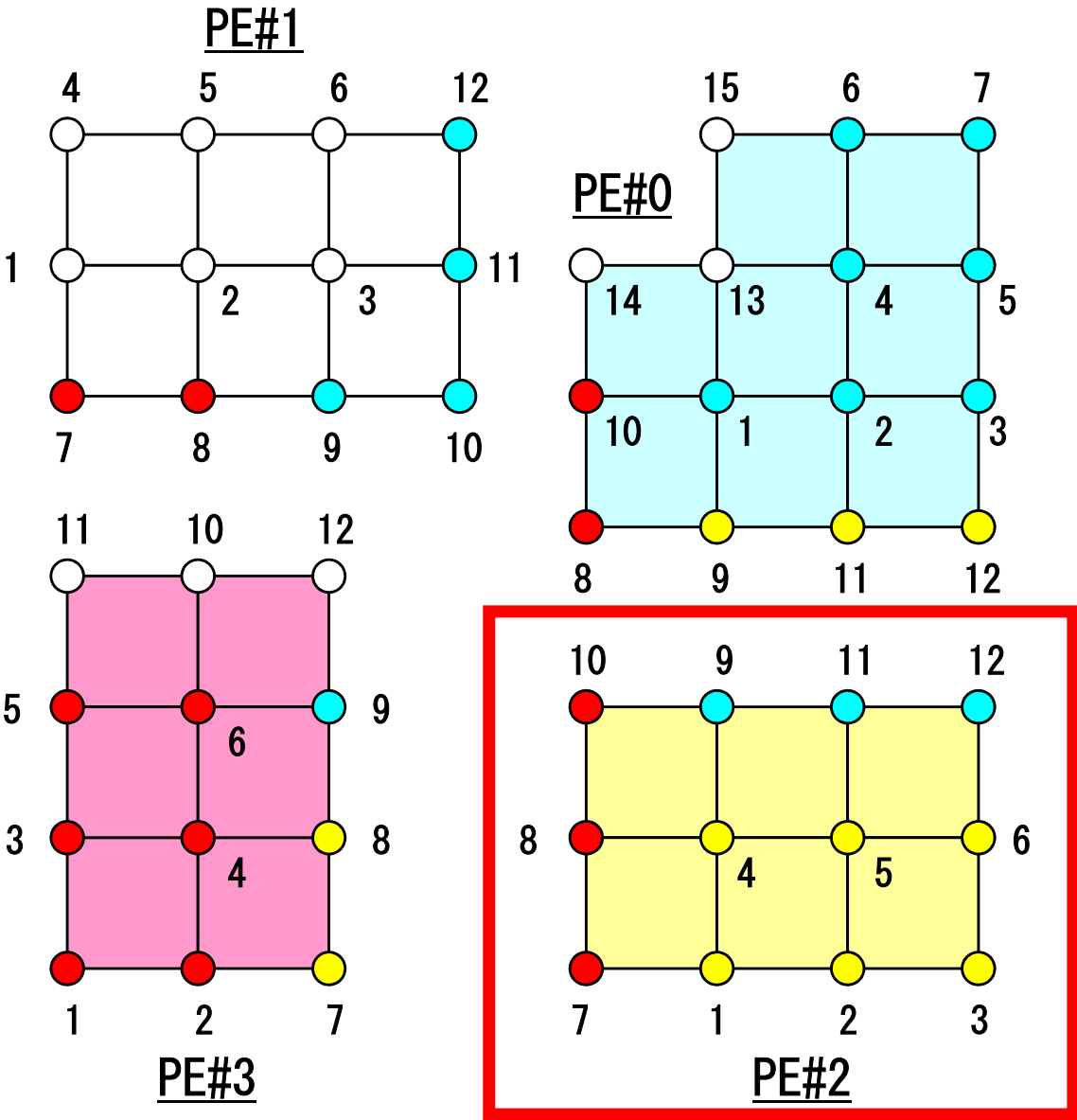
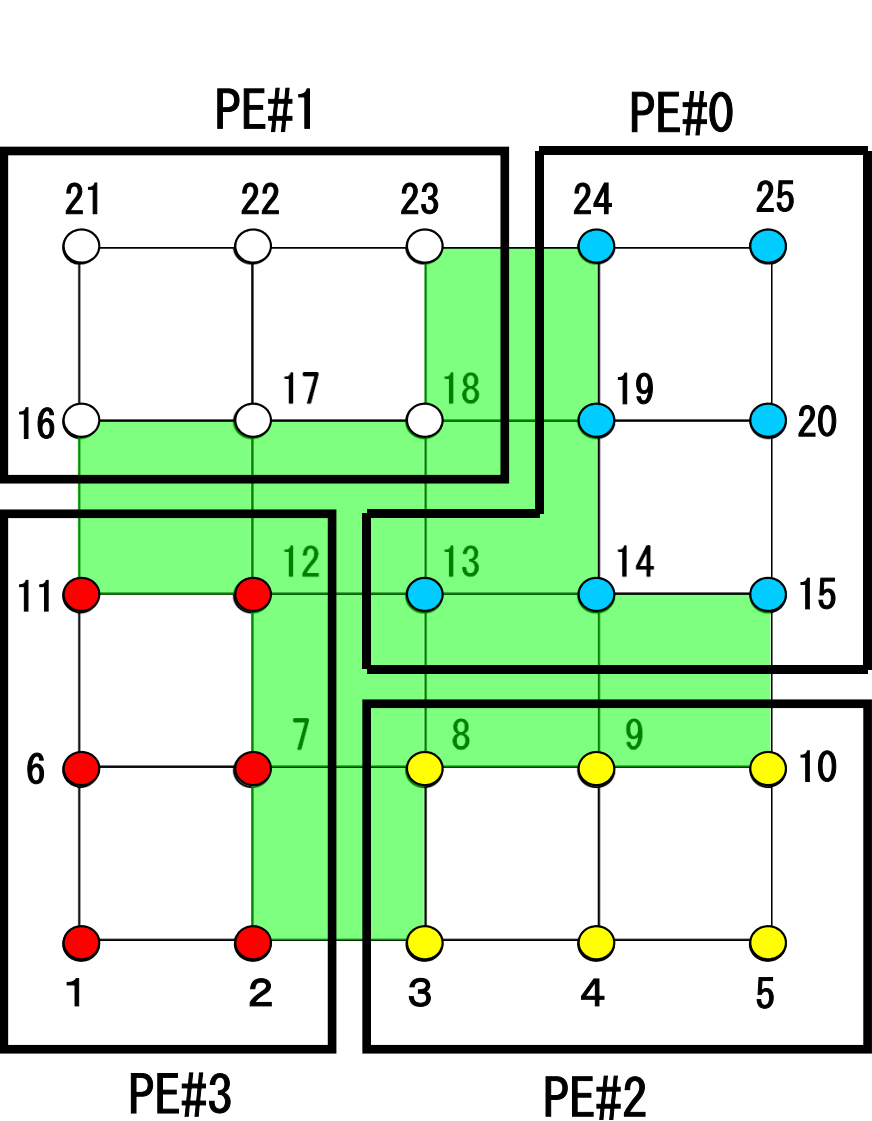
# MPI\_Waitall

- **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.
- **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

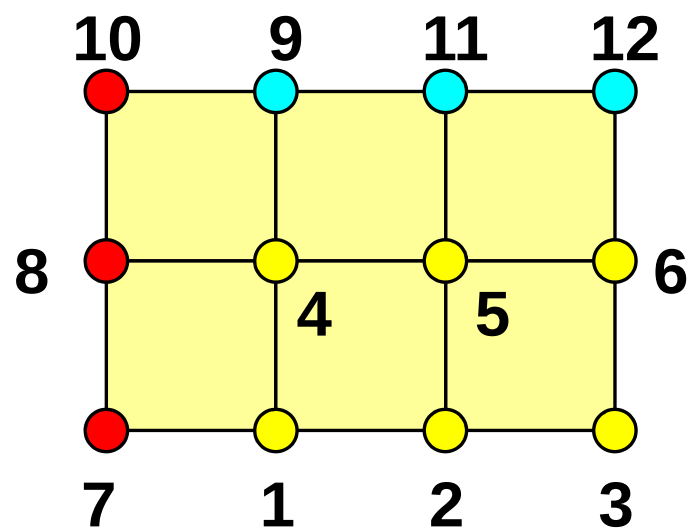
MPI\_STATUS\_SIZE: defined in 'mpif.h', 'mpi.h'

# Node-based Partitioning

internal nodes - elements - external nodes



# Description of Distributed Local Data

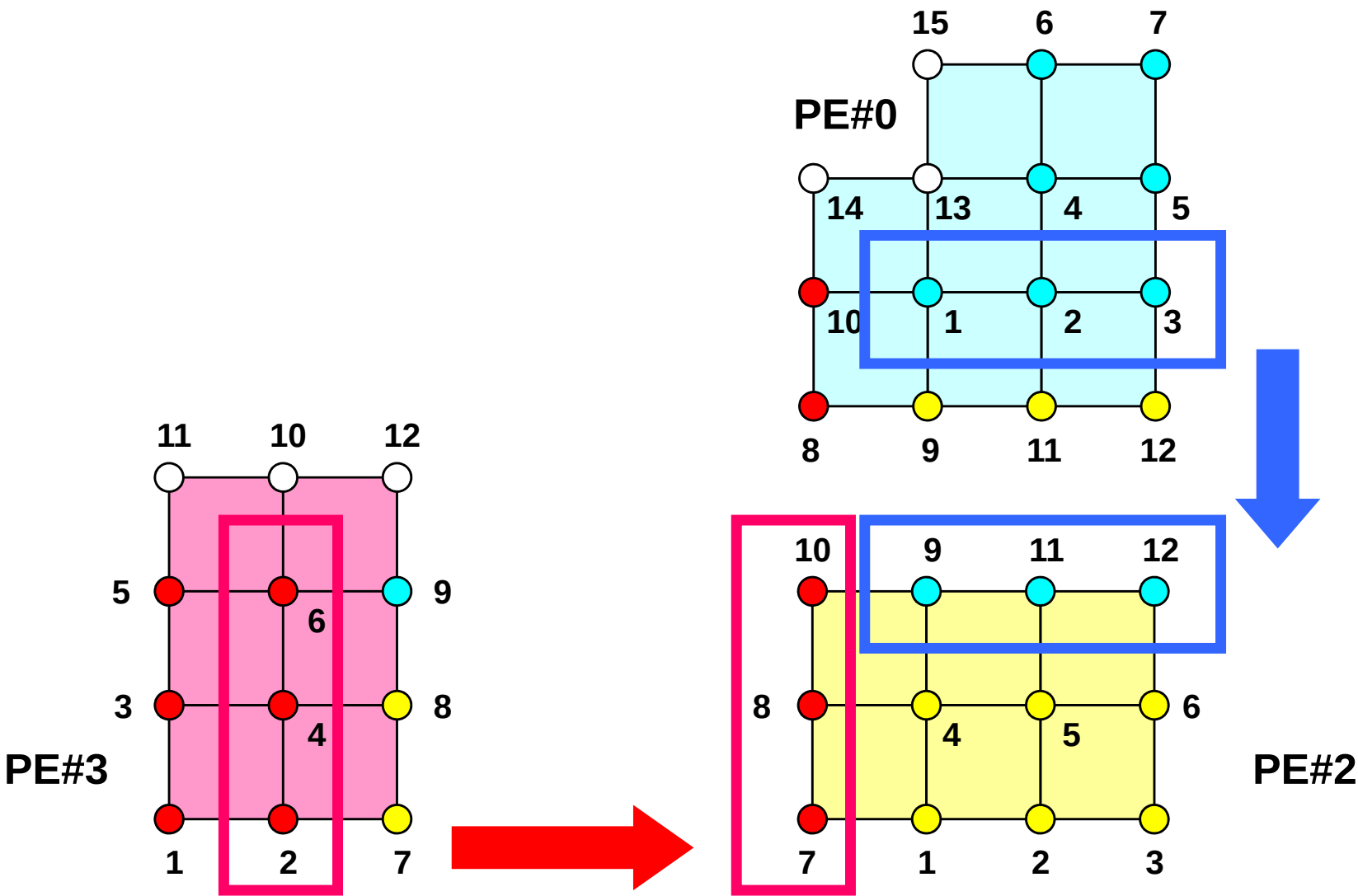


- Internal/External Points
  - Numbering: Starting from internal pts, then external pts after that
- Neighbors
  - Shares overlapped meshes
  - Number and ID of neighbors
- External Points
  - From where, how many, and which external points are received/imported ?
- Boundary Points
  - To where, how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



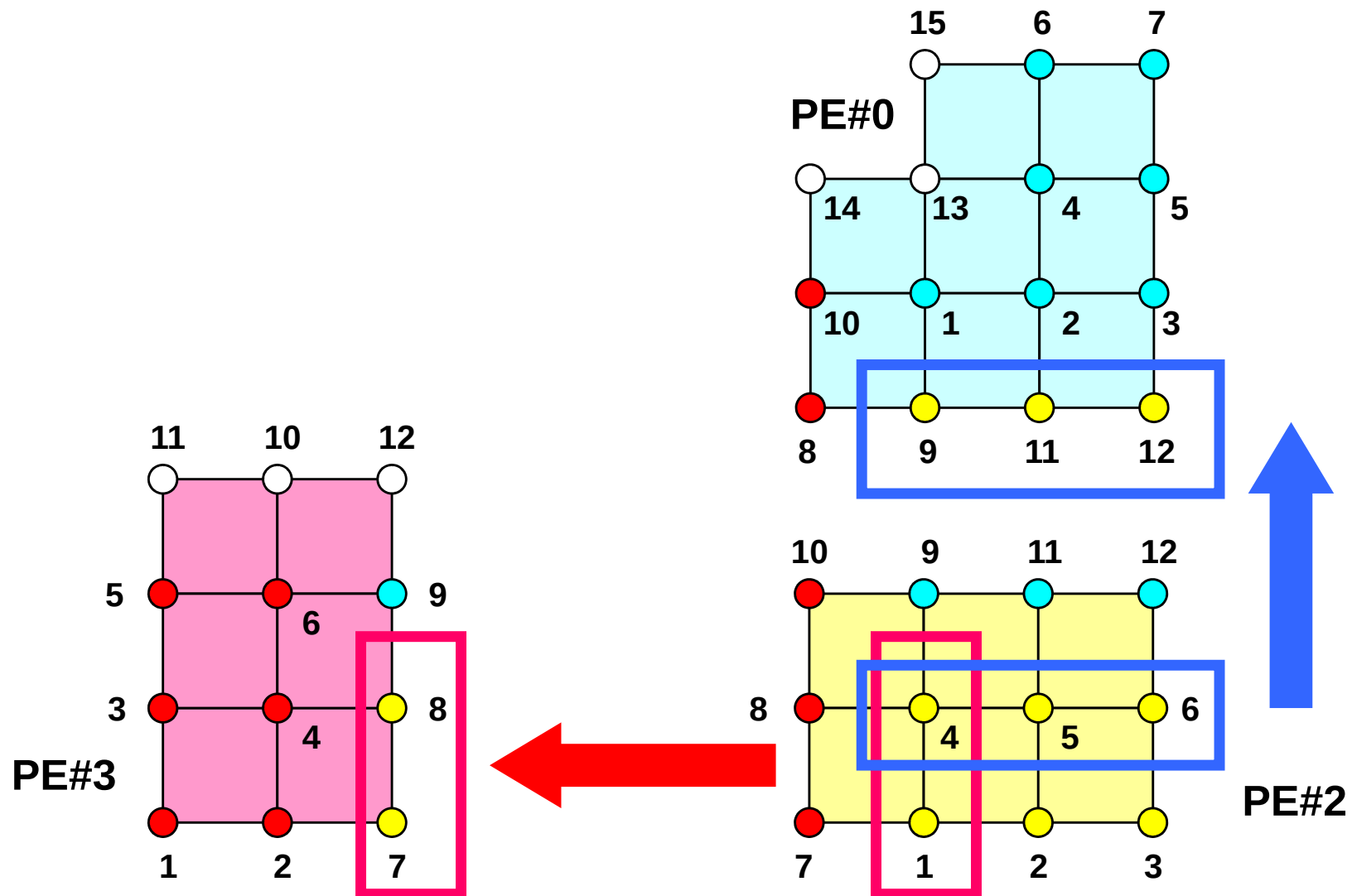
# External Nodes (外点) : RECEIVE

PE#2 : receive information for “external nodes”



# Boundary Nodes (境界点) : SEND

PE#2 : send information on “boundary nodes”



# Distributed Local Data Structure for Parallel Computation

- Distributed local data structure for domain-to-domain communications has been introduced, which is appropriate for such applications with sparse coefficient matrices (e.g. FDM, FEM, FVM etc.).
  - SPMD
  - Local Numbering: Internal pts to External pts
  - Generalized communication table
- Everything is easy, if proper data structure is defined:
  - Values at boundary pts are copied into sending buffers
  - Send/Recv
  - Values at external pts are updated through receiving buffers