Introduction to Parallel FEM in C
Parallel Data Structure

Kengo Nakajima
RIKEN R-CCS

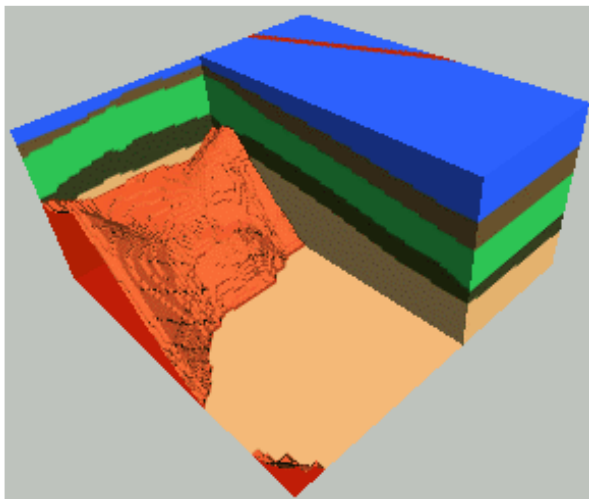# **Parallel Computing**

- 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**
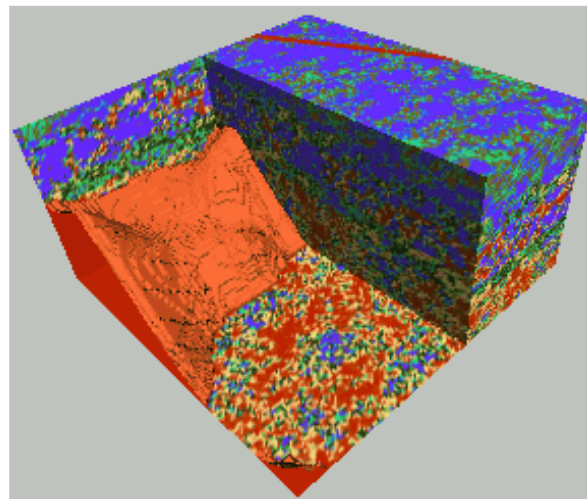
What is Parallel Computing ? (1/2)

- to solve larger problems faster

Homogeneous/Heterogeneous Porous Media
Lawrence Livermore National Laboratory
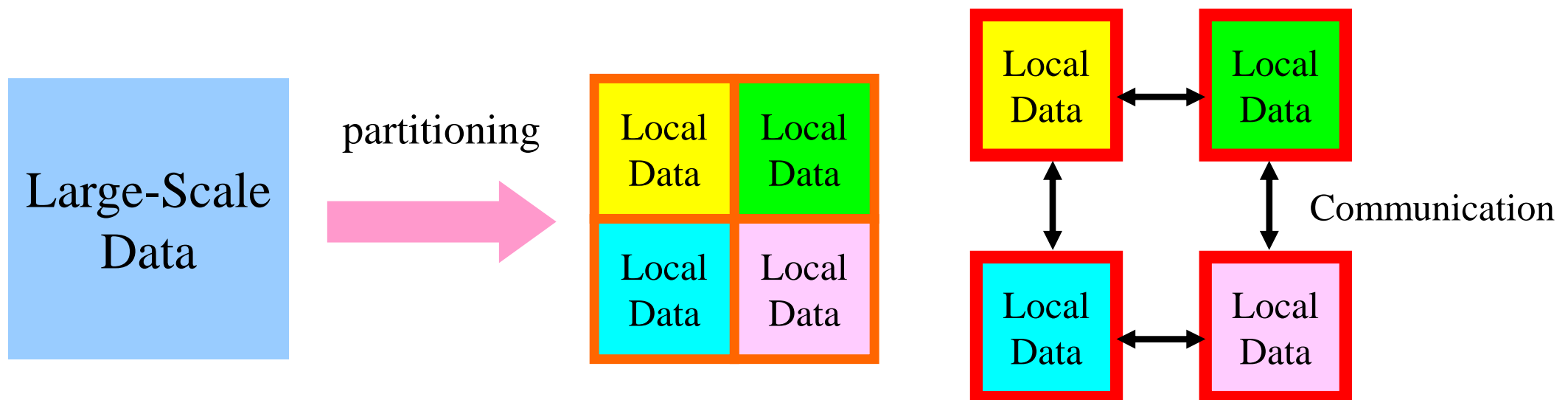

Homogeneous

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

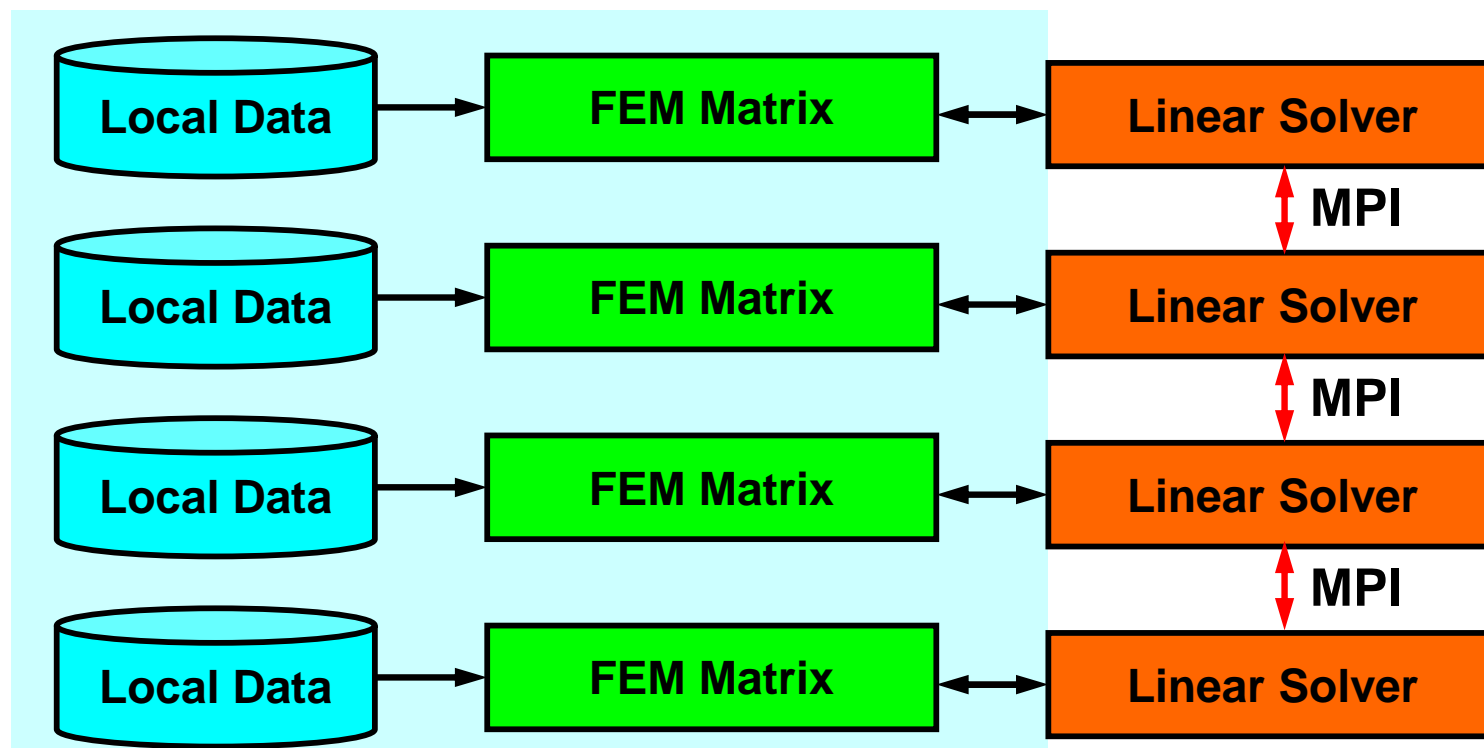- Communications are required in Global Operations for Consistency.

Operations in Parallel FEM
SPMD: <u>S</u>ingle-<u>P</u>rogram <u>M</u>ultiple-<u>D</u>ata

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
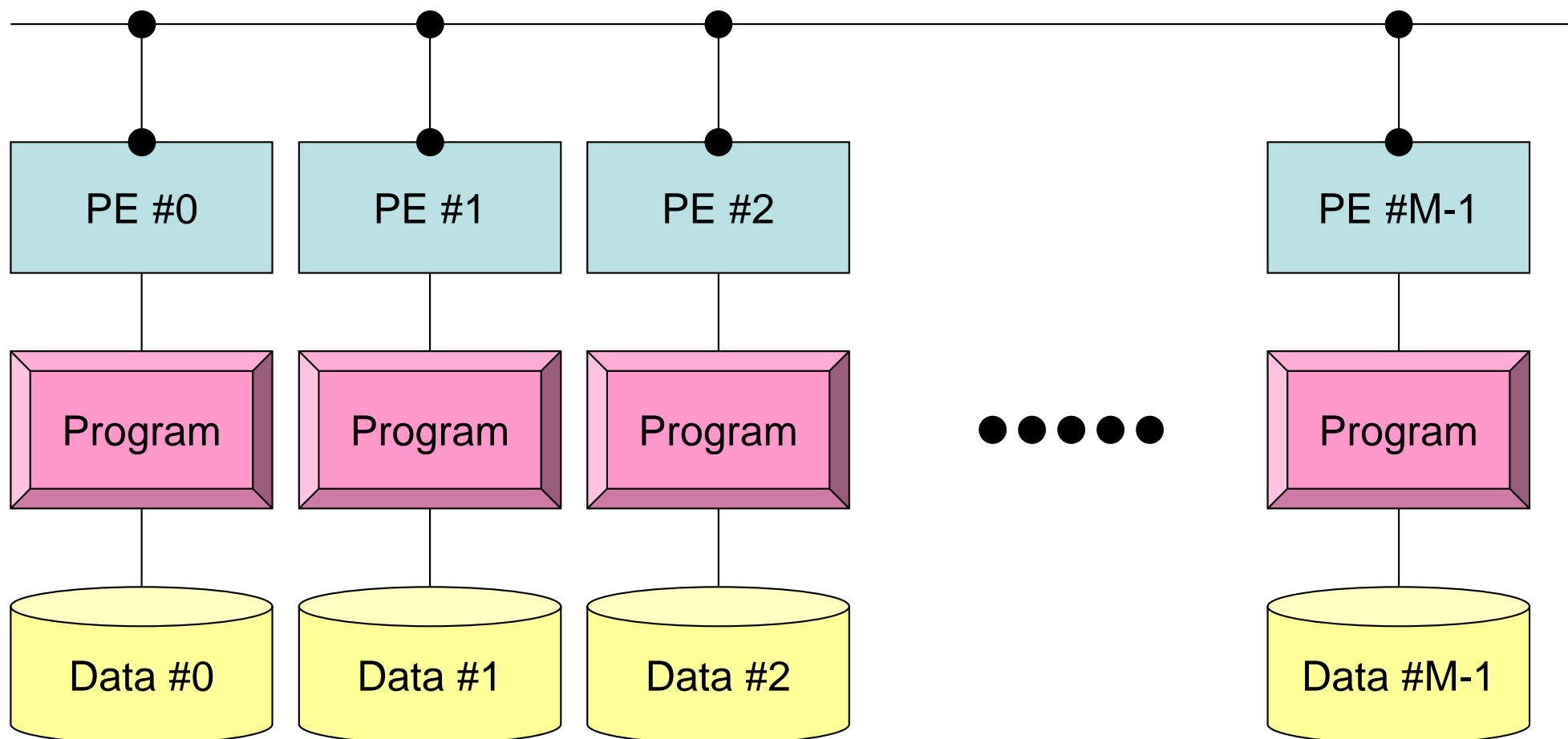
PE: Processing Element
Processor, Domain, Process

SPMD

`mpirun -np M <Program>`

PE #0   PE #1   PE #2   PE #M-1

Program   Program   Program   ● ● ● ● ●   Program

Data #0   Data #1   Data #2   Data #M-1

Each process does same operation for different data

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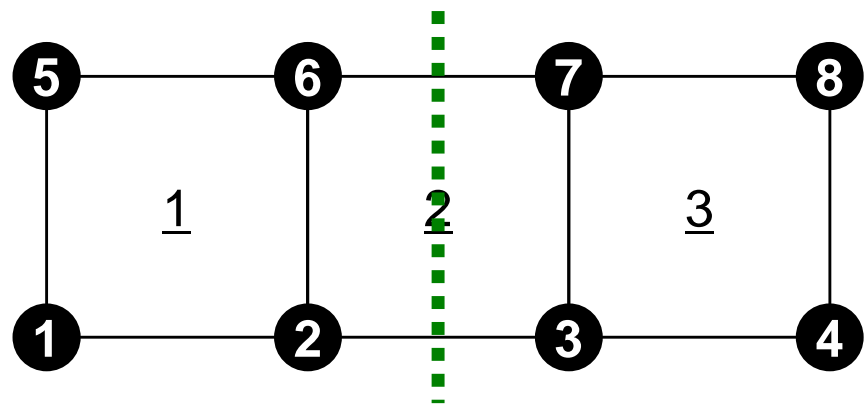It is ideal that parallel program is not different from serial one except communication.

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
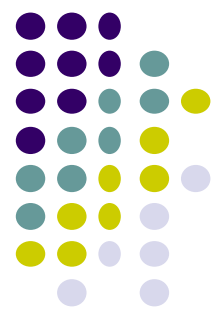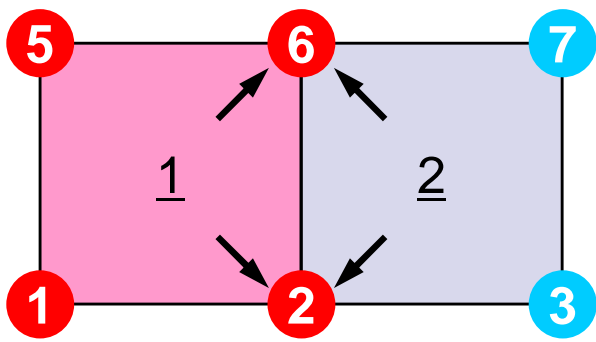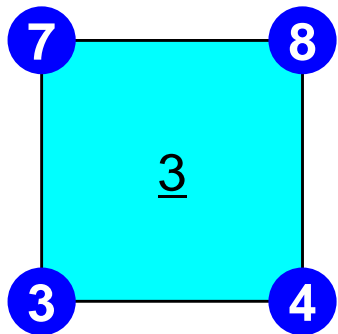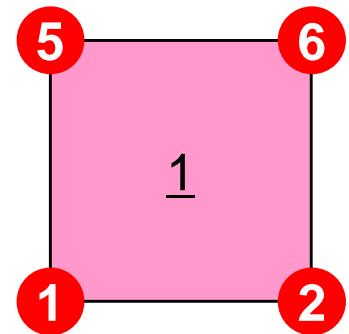
Bi-Linear Square Elements
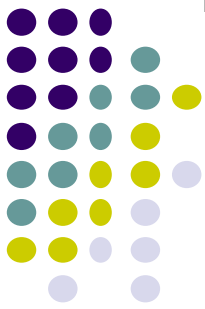## Values are defined on each node

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.
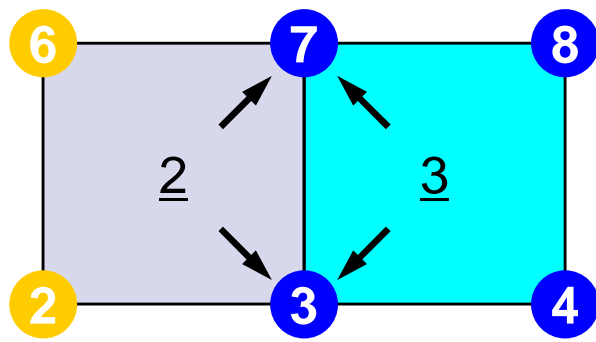
Local Data of Parallel FEM

- 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
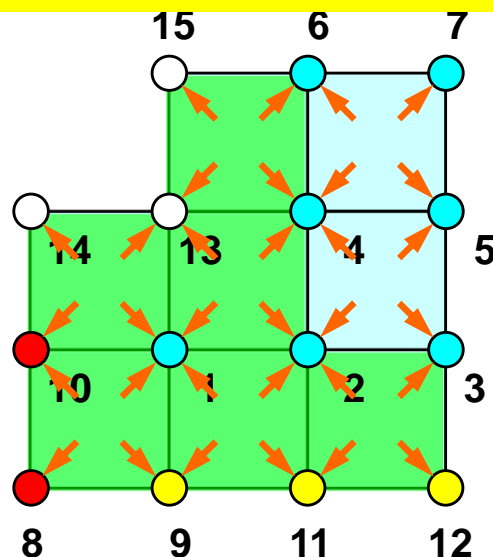internal nodes - elements - external nodes

PE#1

PE#0

PE#3

PE#2

PE#1

PE#0

PE#3

PE#2
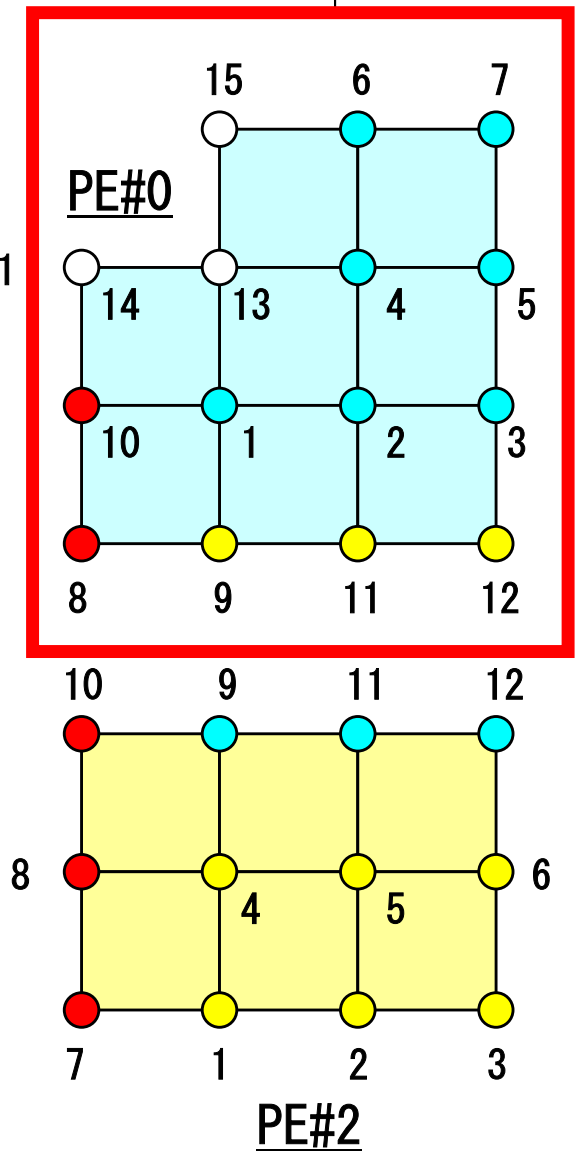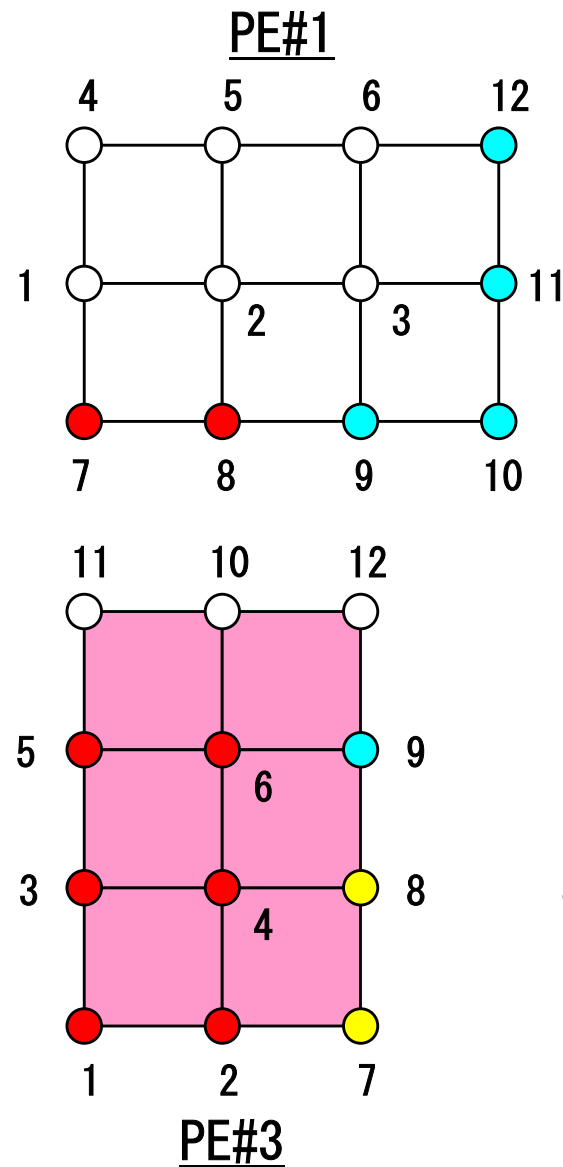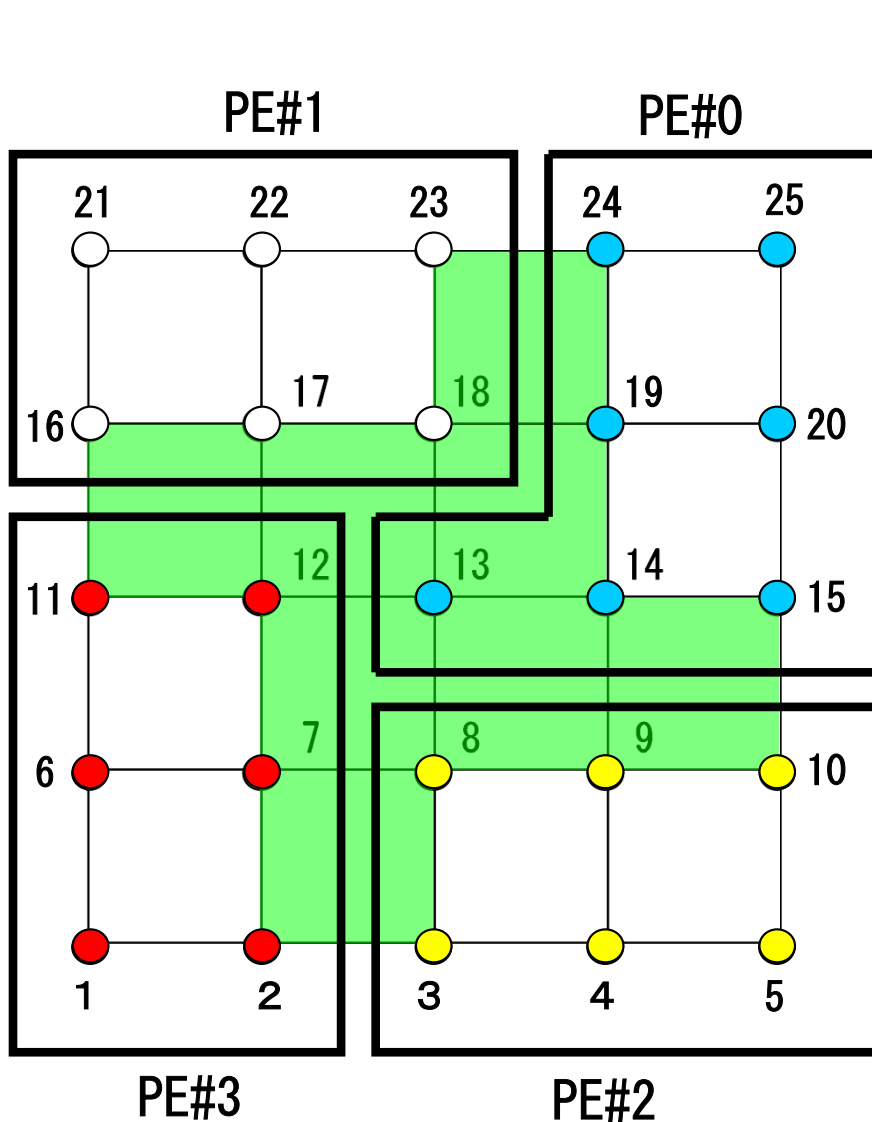
Node-based Partitioning
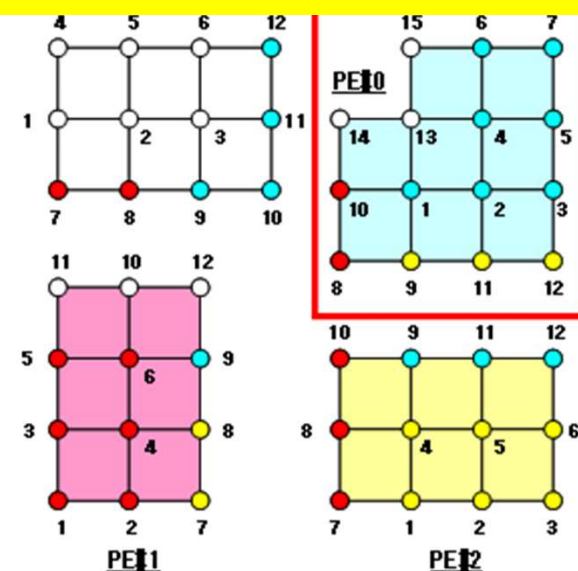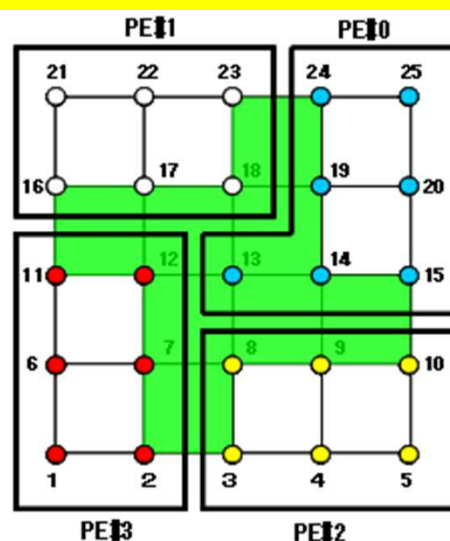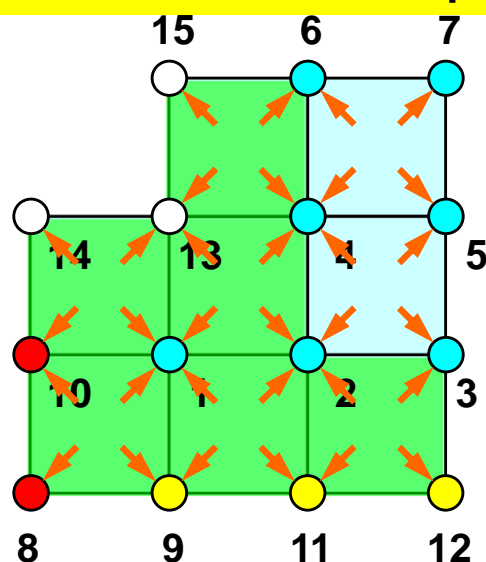## internal nodes - elements - external nodes

- 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.
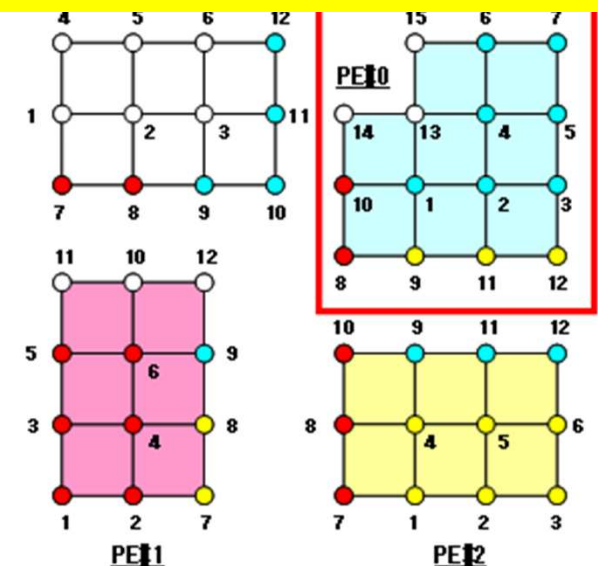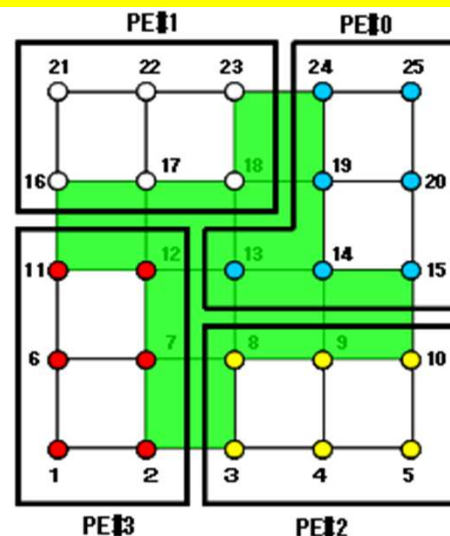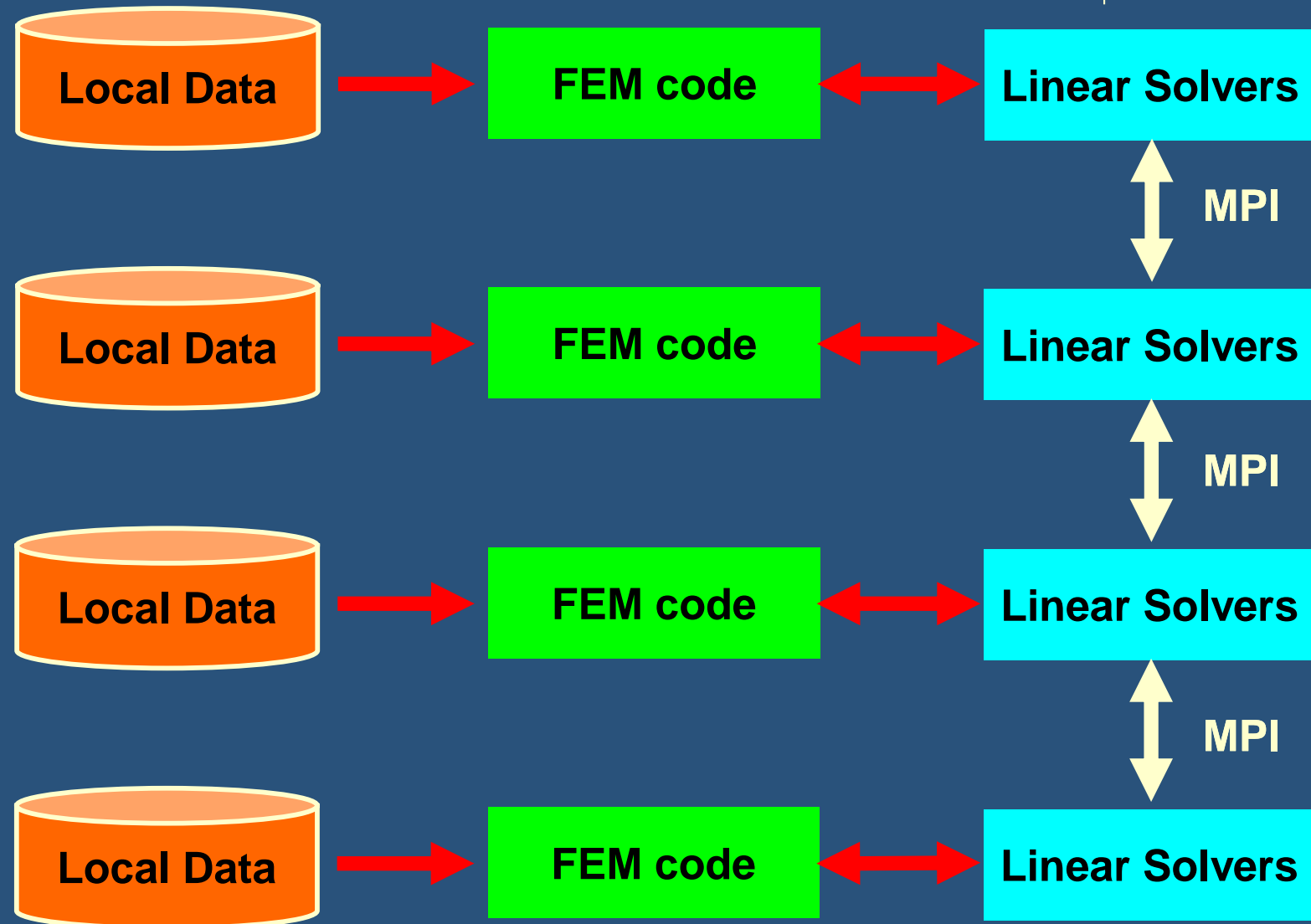
We do not need communication during matrix assemble !!

- 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.
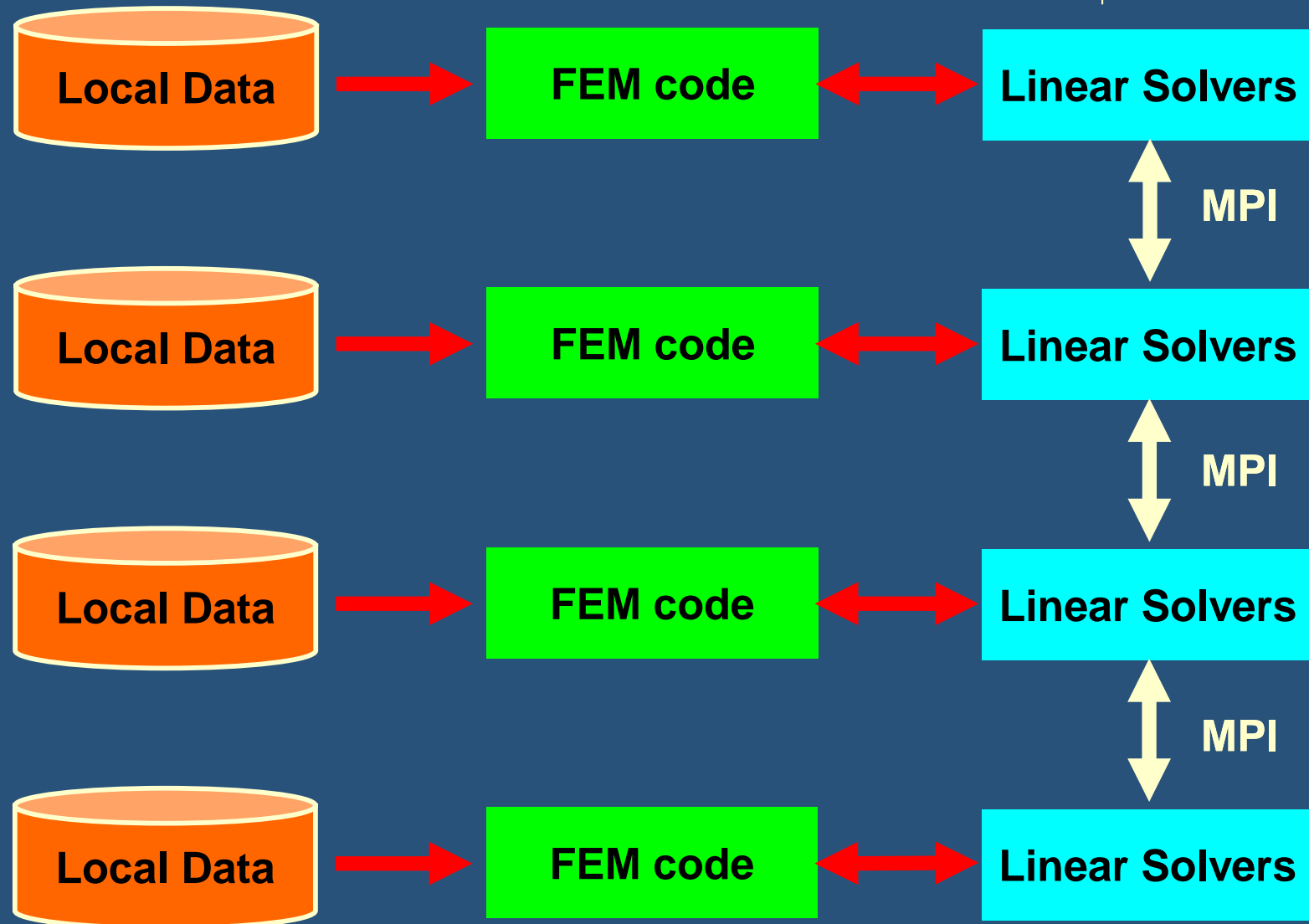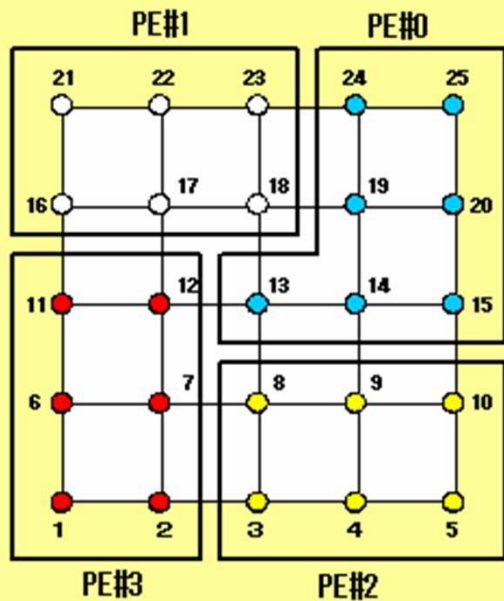
Parallel Computing in FEM
SPMD: <u>S</u>ingle-<u>P</u>rogram <u>M</u>ultiple-<u>D</u>ata

| Local Data | → | FEM code | ↔ | Linear Solvers |

MPI

| Local Data | → | FEM code | ↔ | Linear Solvers |

MPI

| Local Data | → | FEM code | ↔ | Linear Solvers |

MPI

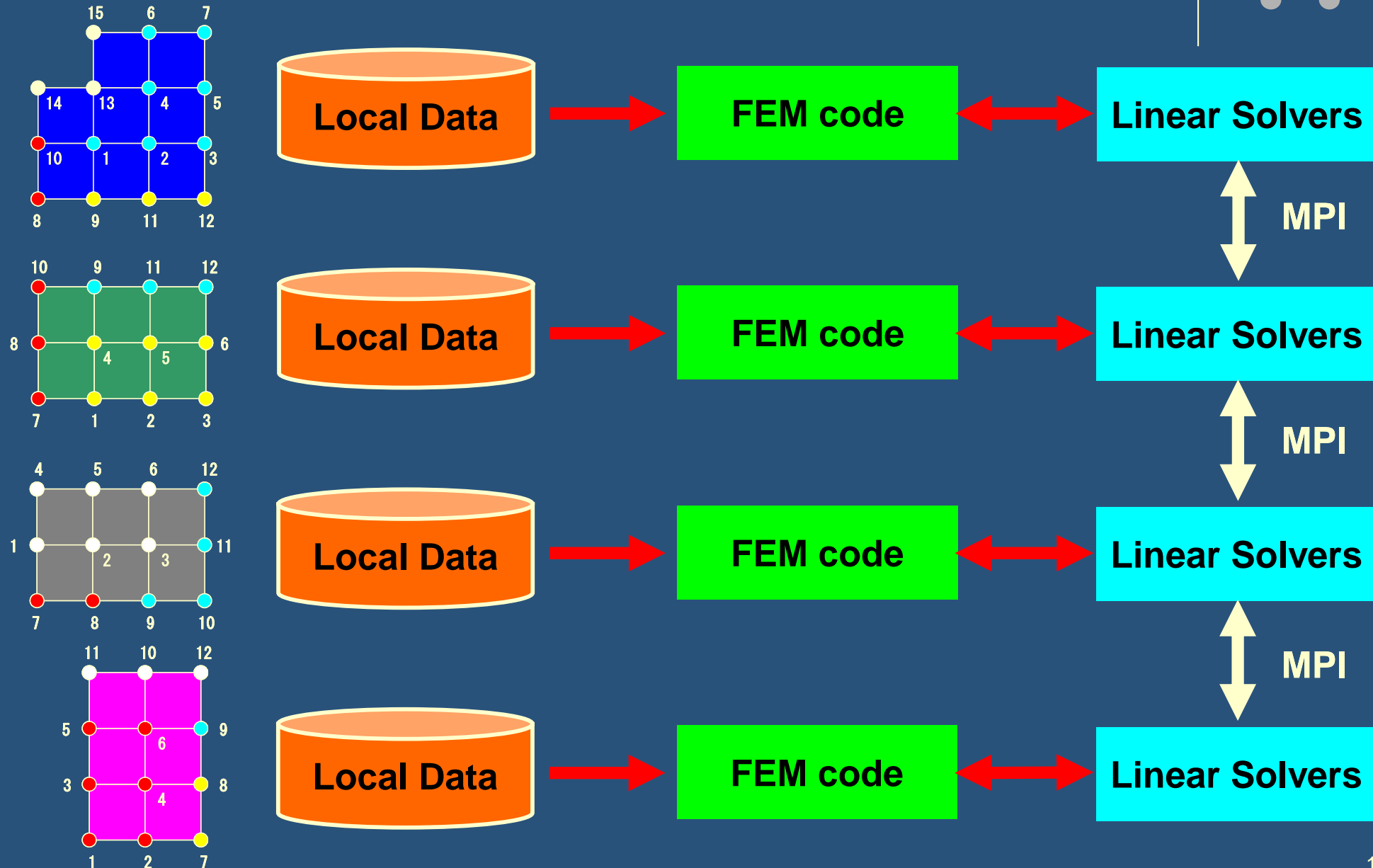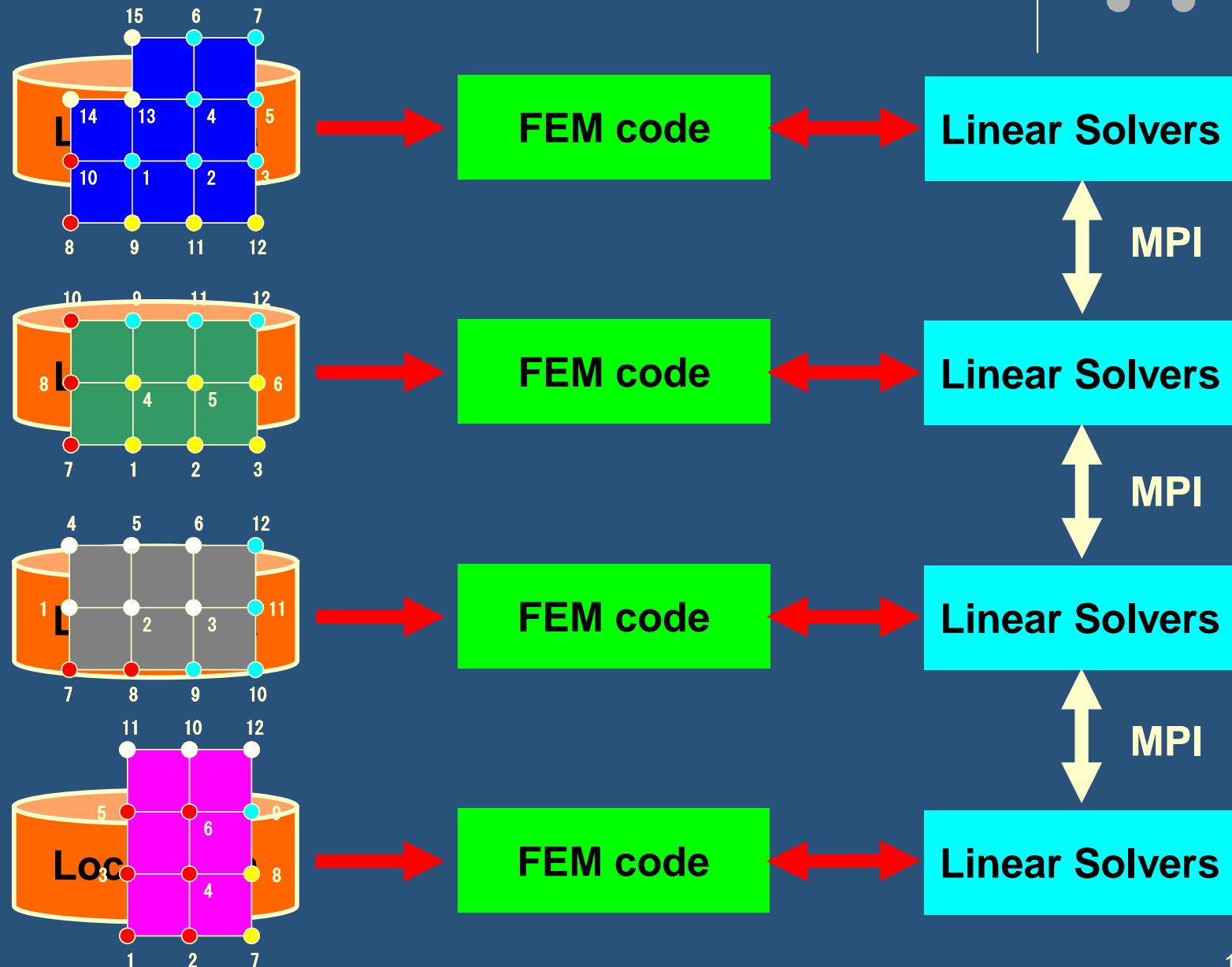| Local Data | → | FEM code | ↔ | Linear Solvers |

Parallel Computing in FEM
SPMD: Single-Program Multiple-Data

Parallel Computing in FEM
SPMD: Single-Program Multiple-Data

Parallel Computing in FEM
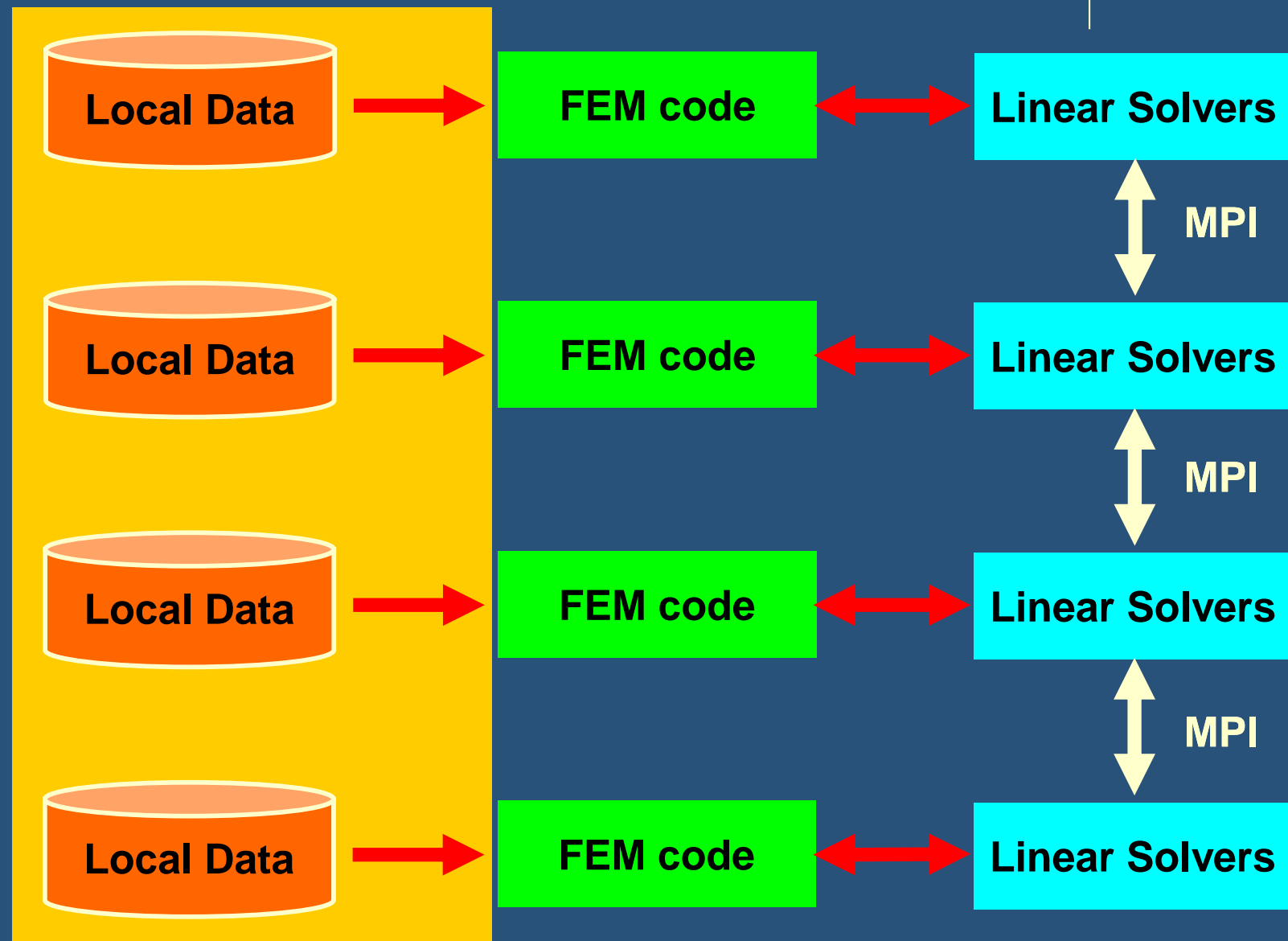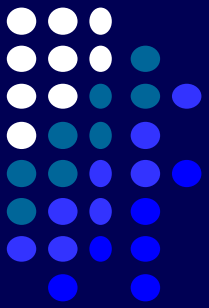## SPMD: Single-Program Multiple-Data

Parallel Computing in FEM
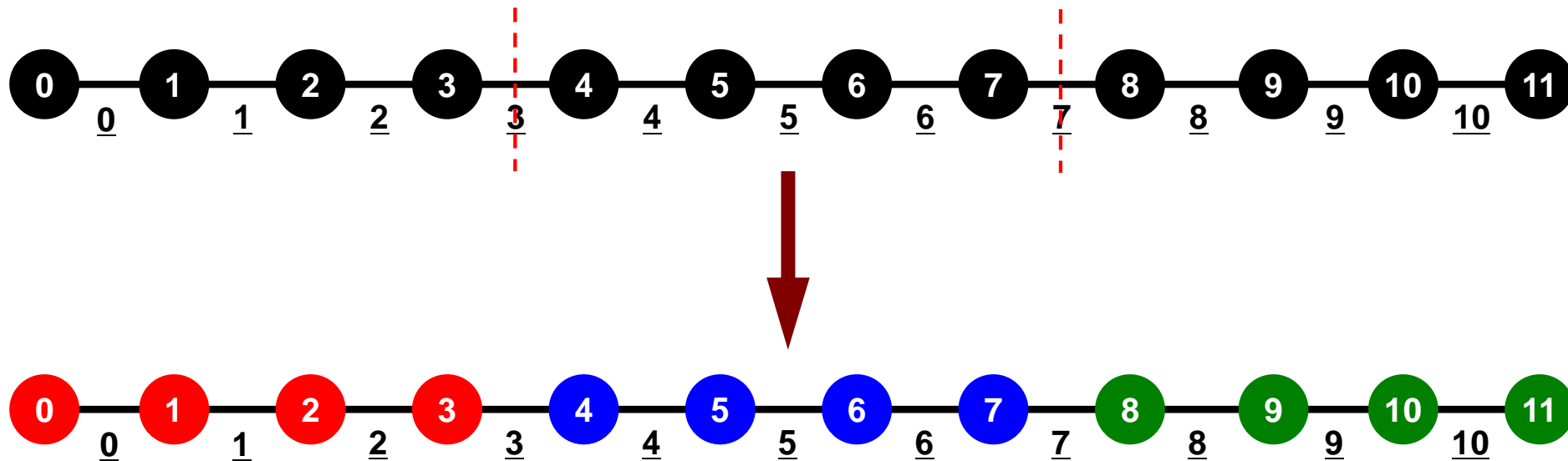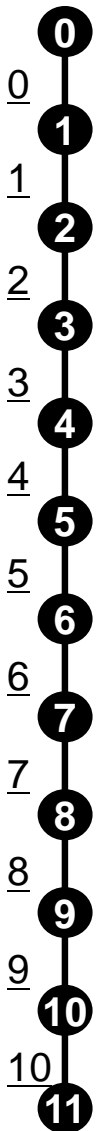## SPMD: Single-Program Multiple-Data

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
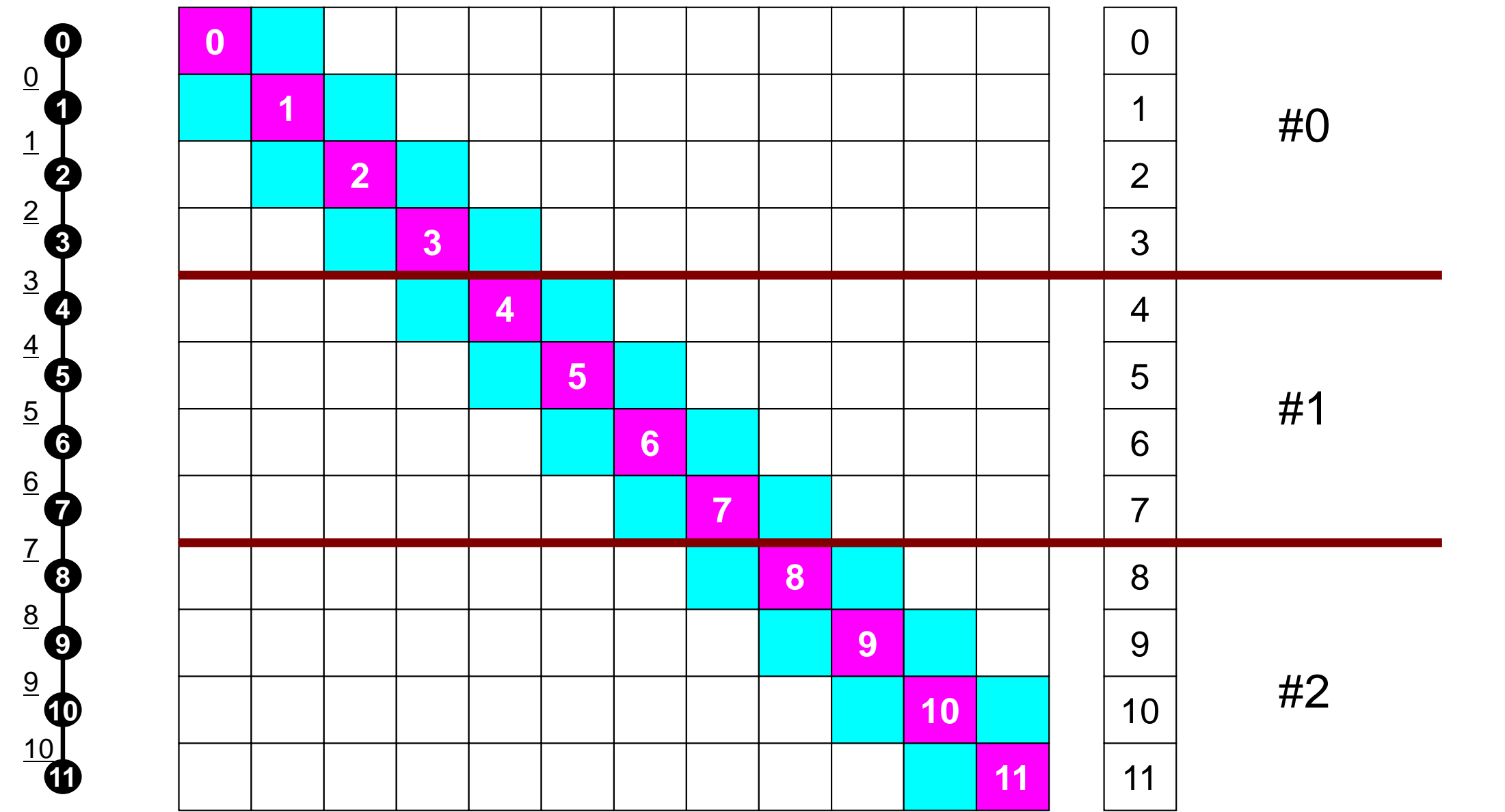
1D FEM: 12 nodes/11 elem's/3 domains
三重対角行列：Tri-Diagonal Matrix

"Internal Nodes" should be balanced

Matrices are incomplete !
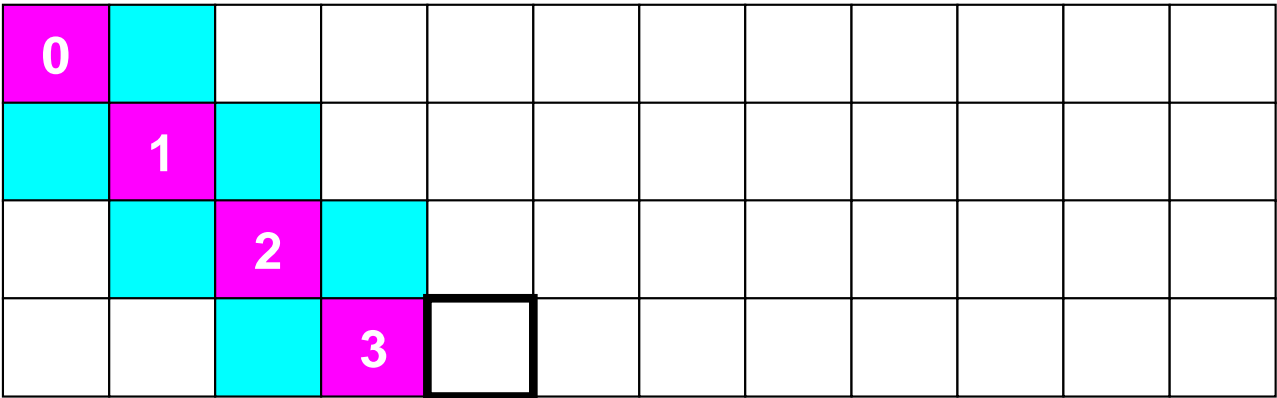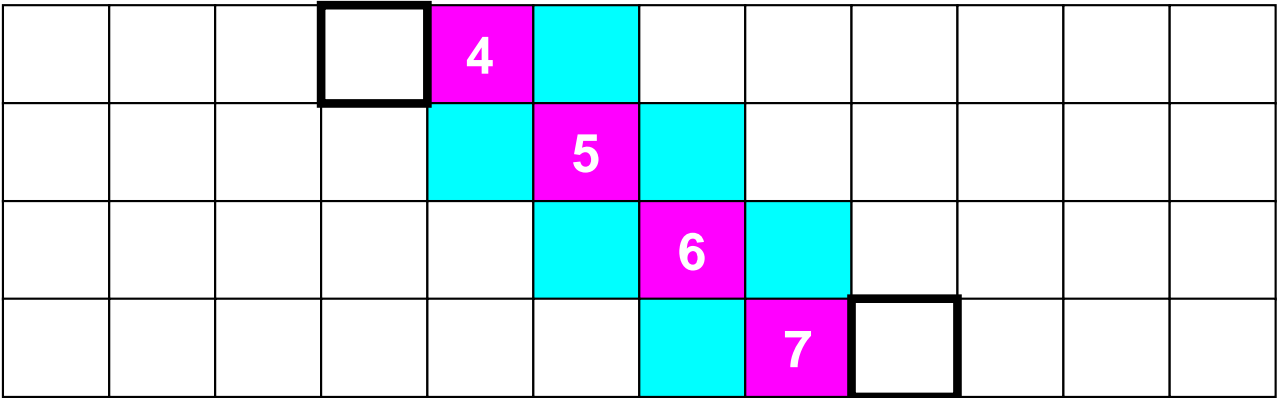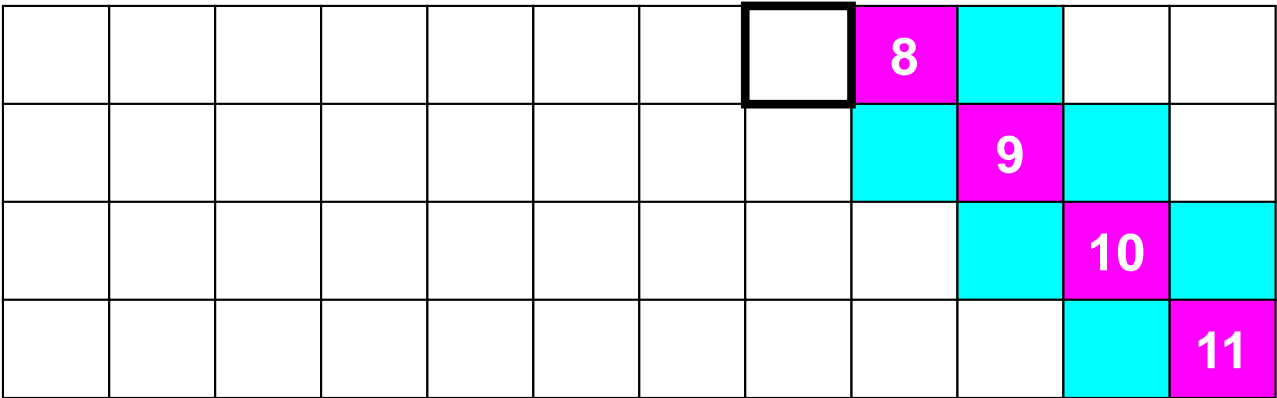
Connected Elements + External Nodes
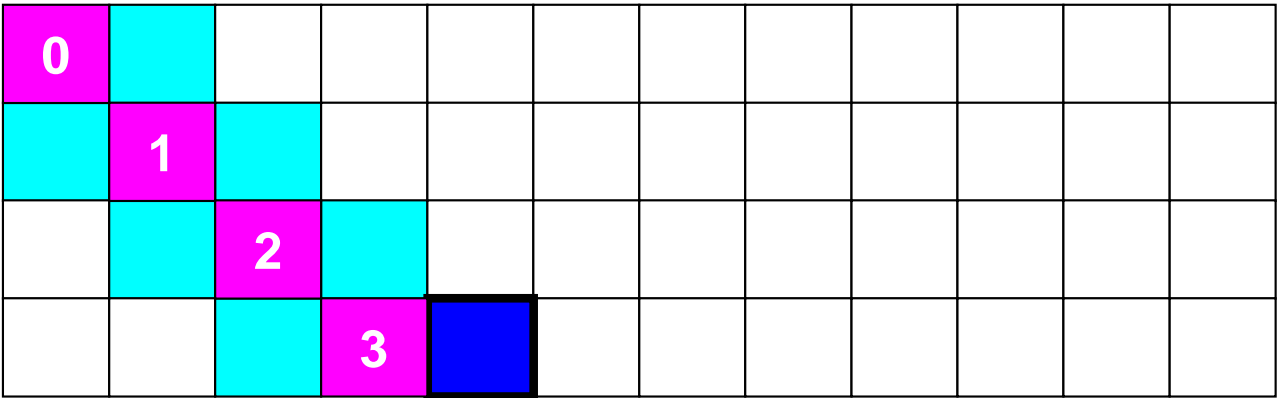
#0
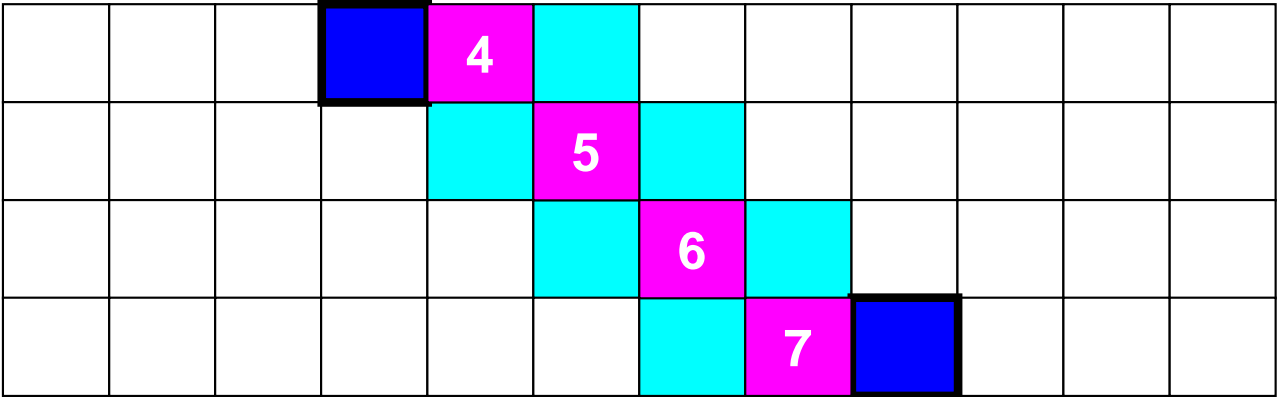
#1

#2
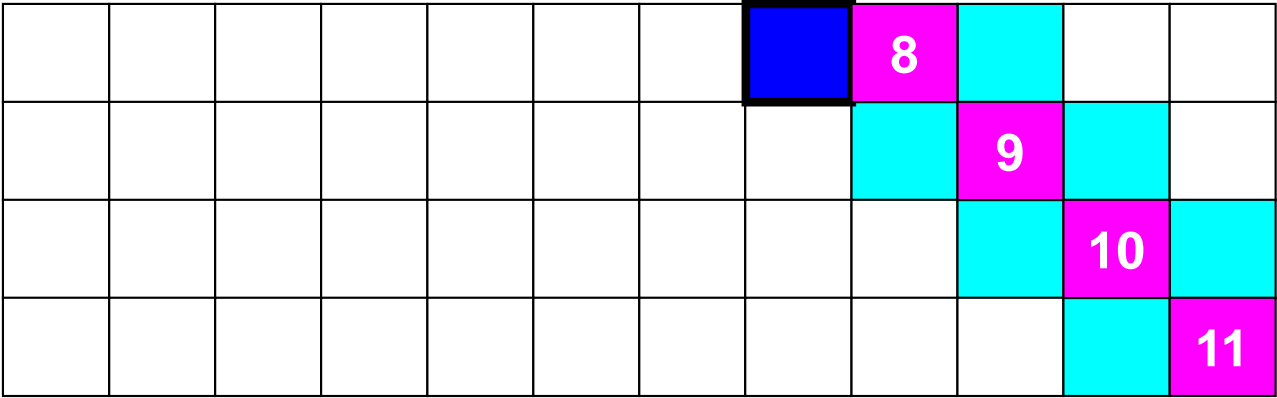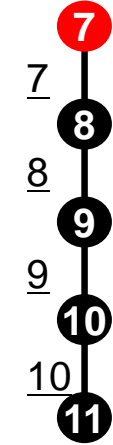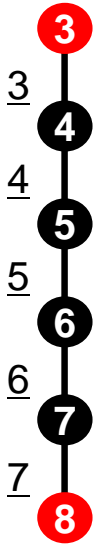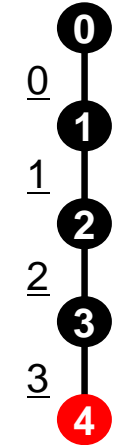
1D FEM: 12 nodes/11 elem's/3 domains
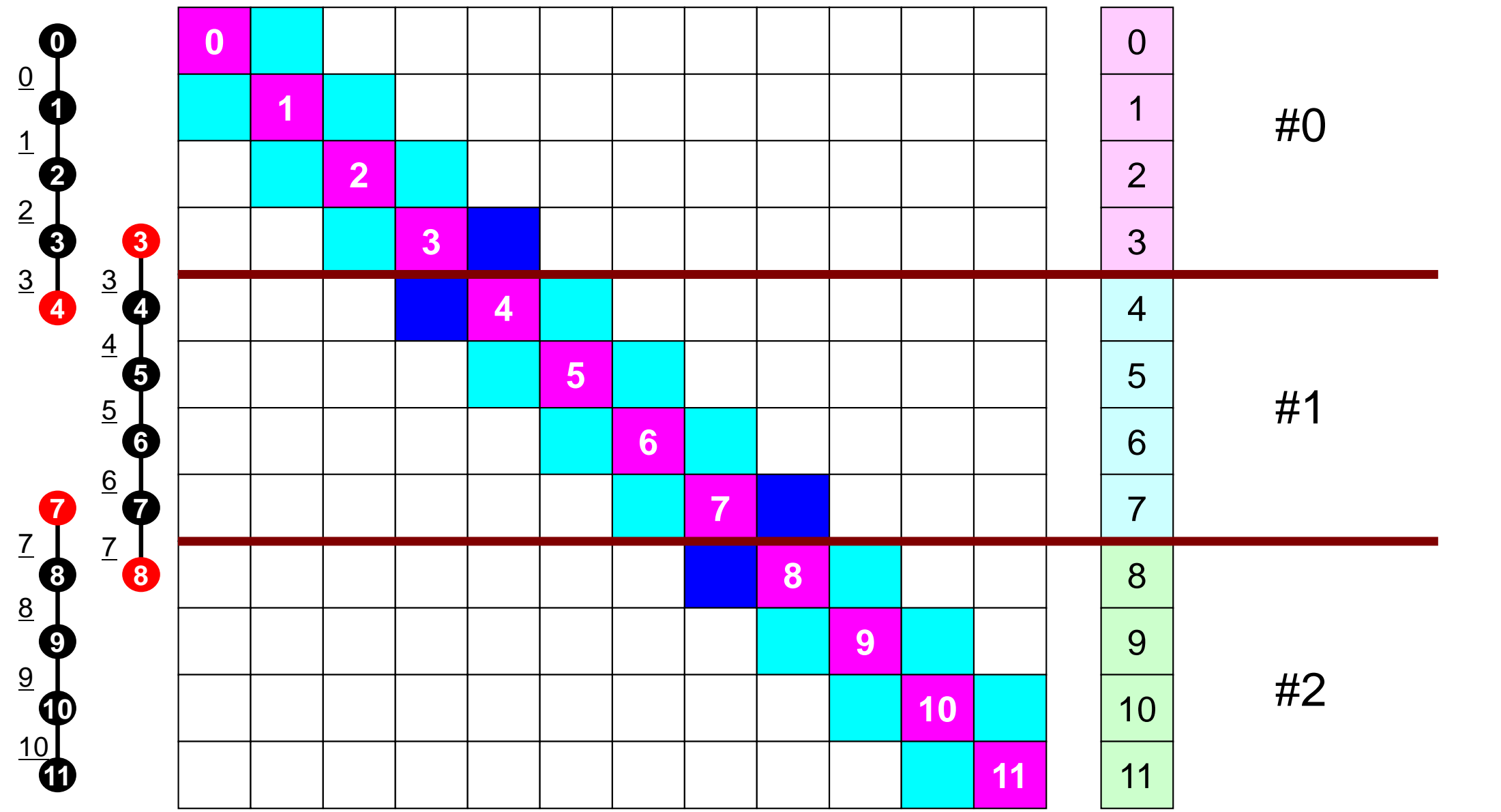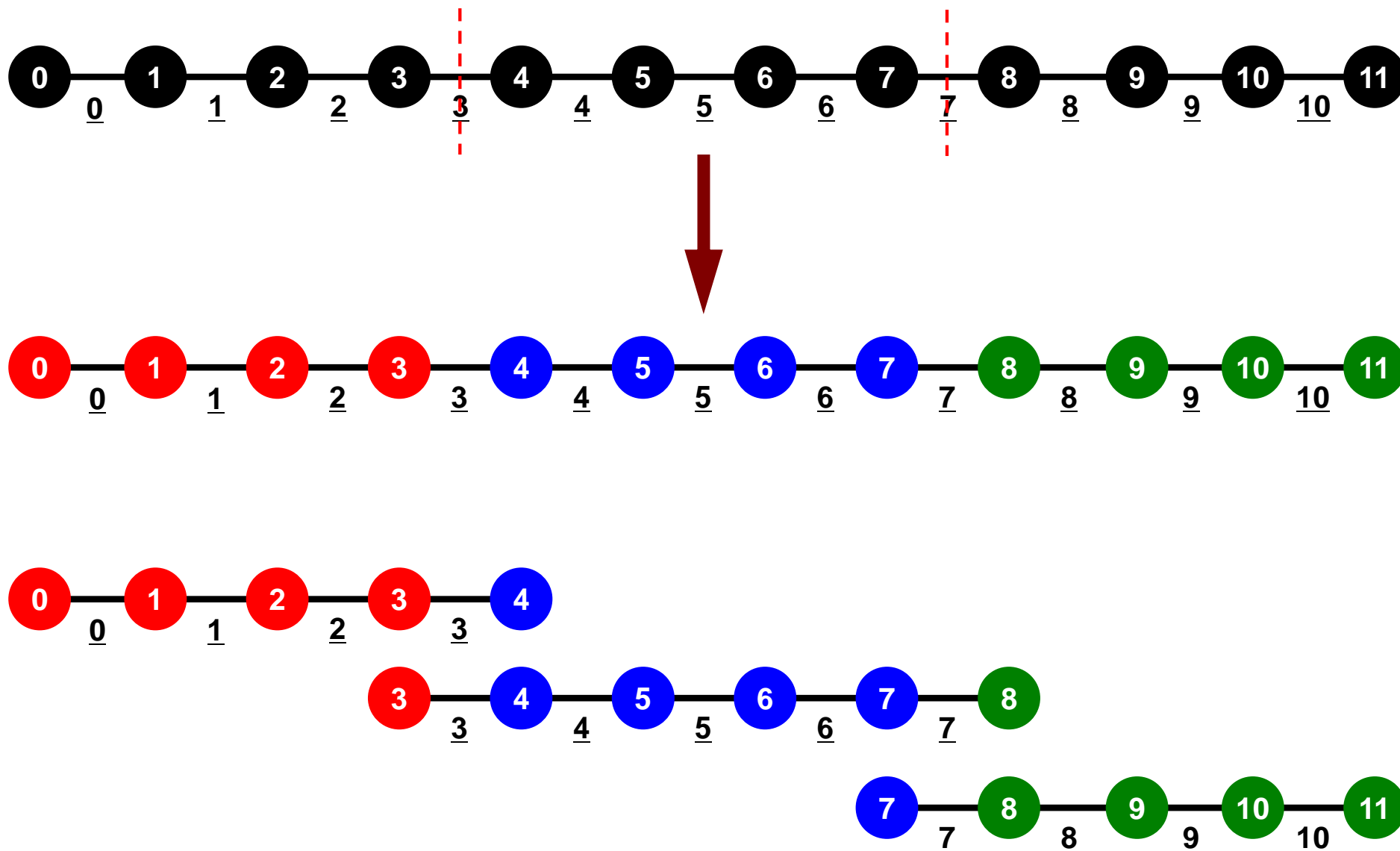
1D FEM: 12 nodes/11 elem's/3 domains

PE: Processing Element
Processor, Domain, Process

SPMD

`mpirun -np M <Program>`

PE #0	PE #1	PE #2	PE #M-1
Program	Program	Program	● ● ● ● ●	Program
Data #0	Data #1	Data #2	Data #M-1
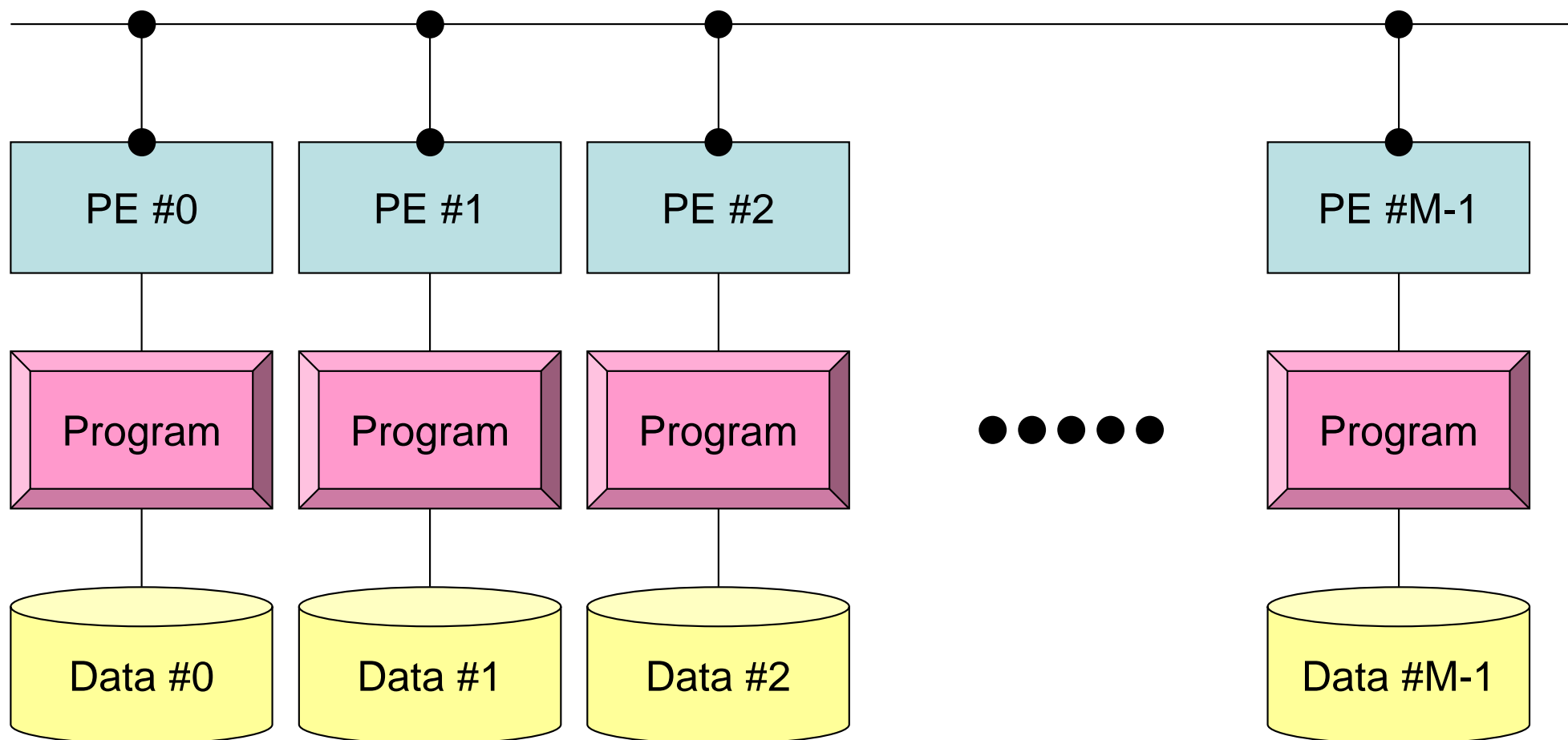
Each process does same operation for different data

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)
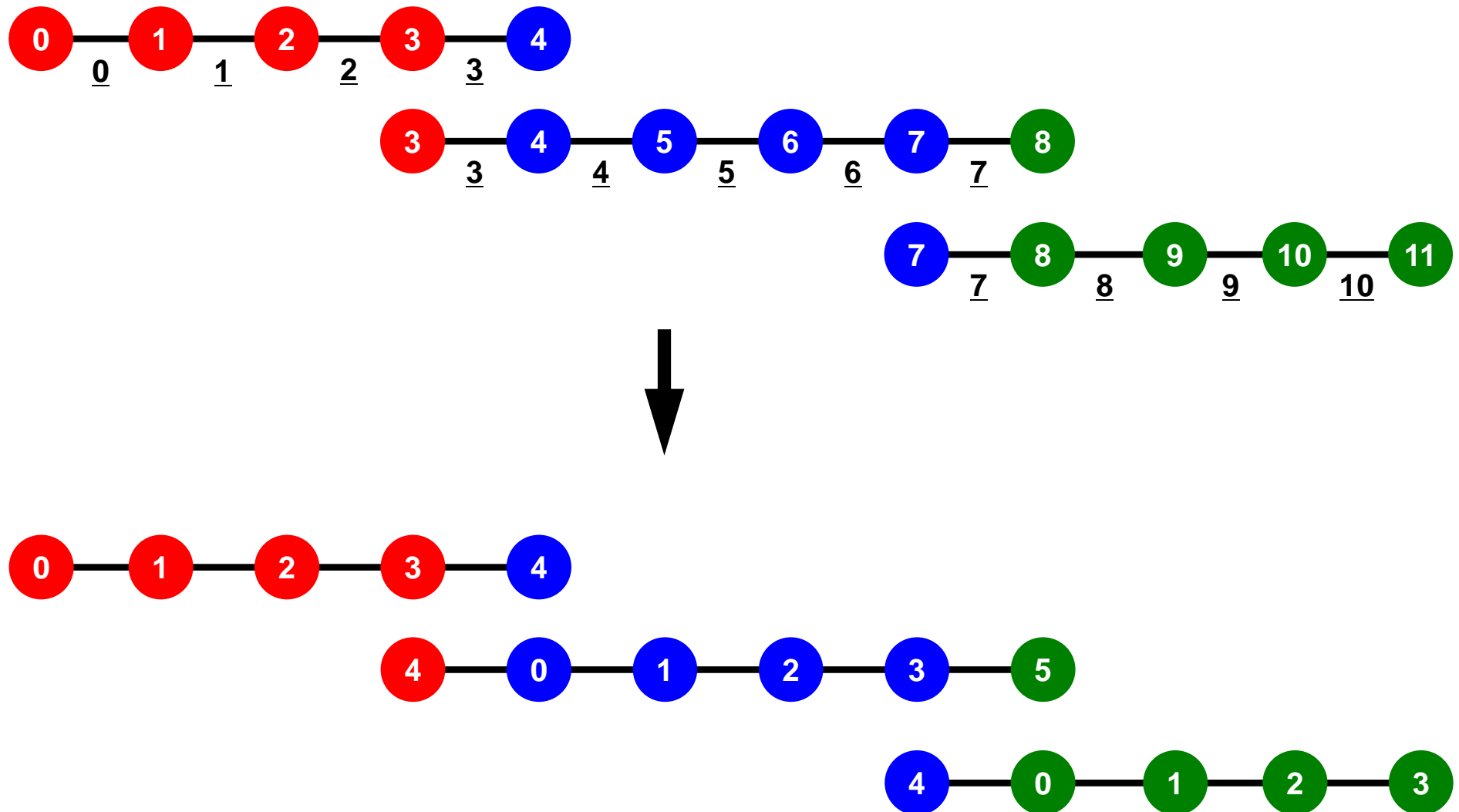
1D FEM: 12 nodes/11 elem's/3 domains
Integration on each element, element matrix -> global matrix
Operations can be done by info. of internal/external nodes
and elements which include these nodes

Because the matrix is sparse, the union of the local matrices forms the global matrix !
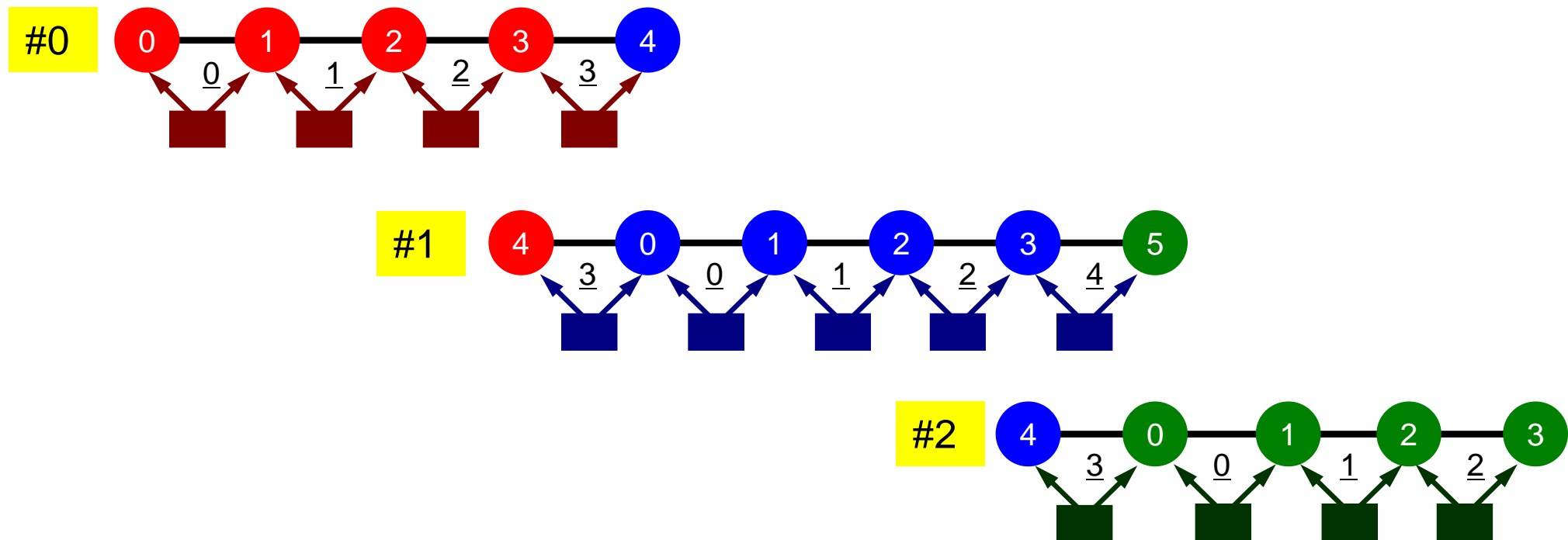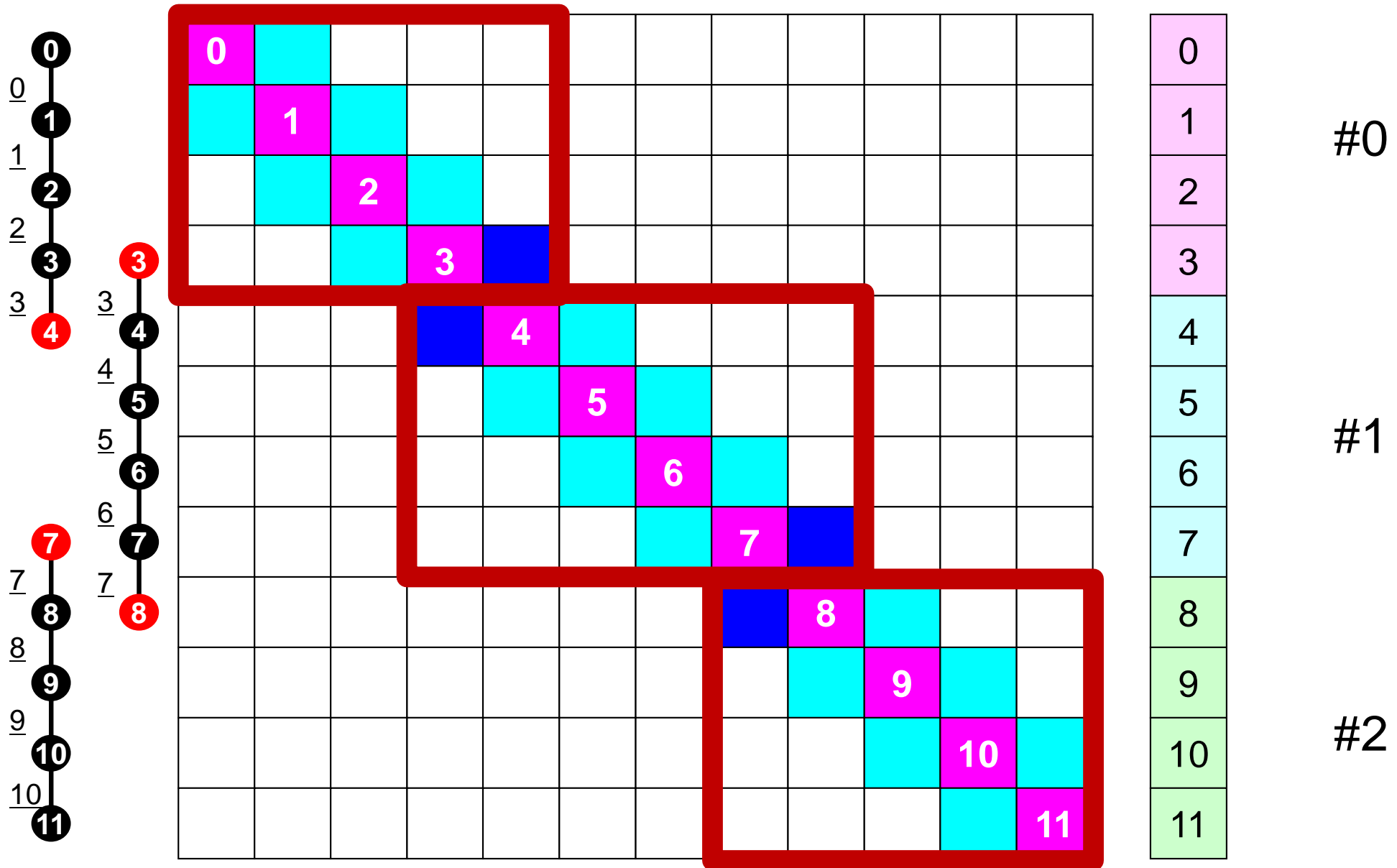
Finite Element Procedures

- ## 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 r⁽⁰⁾= b-[A]x⁽⁰⁾
for i= 1, 2, …
    solve [M]z⁽ⁱ⁻¹⁾= r⁽ⁱ⁻¹⁾
    ρᵢ₋₁= r⁽ⁱ⁻¹⁾ z⁽ⁱ⁻¹⁾
    if i=1
      p⁽¹⁾= z⁽⁰⁾
     else
       βᵢ₋₁= ρᵢ₋₁/ρᵢ₋₂
       p⁽ⁱ⁾= z⁽ⁱ⁻¹⁾ + βᵢ₋₁ p⁽ⁱ⁻¹⁾
    endif
    q⁽ⁱ⁾= [A]p⁽ⁱ⁾
    αᵢ = ρᵢ₋₁/p⁽ⁱ⁾q⁽ⁱ⁾
    x⁽ⁱ⁾= x⁽ⁱ⁻¹⁾ + αᵢp⁽ⁱ⁾
    r⁽ⁱ⁾= r⁽ⁱ⁻¹⁾ - αᵢq⁽ⁱ⁾
    check convergence |r|
end
```

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

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

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];
  }
```
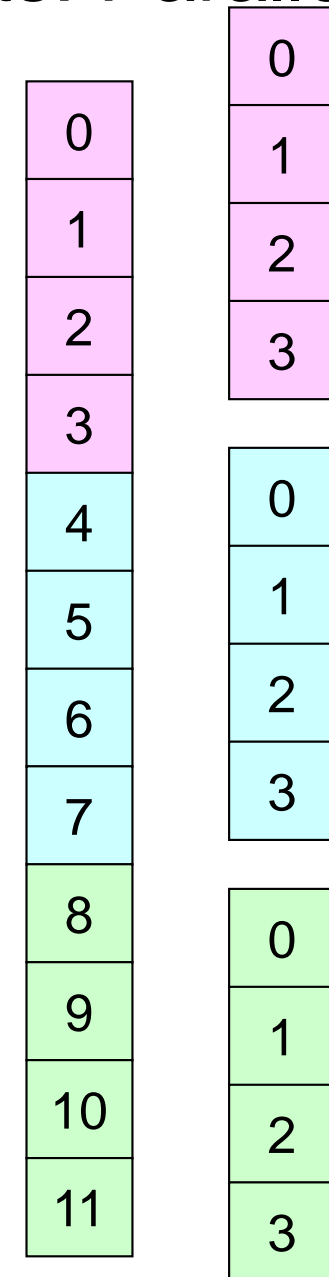
Dot Products
Global Summation needed: Communication ?

```
/*
//-- 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

Preconditioned CG Solver

```
Compute r⁽⁰⁾= b-[A]x⁽⁰⁾
for i= 1, 2, …
    solve [M]z⁽ⁱ⁻¹⁾= r⁽ⁱ⁻¹⁾
    ρᵢ₋₁= r⁽ⁱ⁻¹⁾ z⁽ⁱ⁻¹⁾
    if i=1
      p⁽¹⁾= z⁽⁰⁾
     else
       βᵢ₋₁= ρᵢ₋₁/ρᵢ₋₂
       p⁽ⁱ⁾= z⁽ⁱ⁻¹⁾ + βᵢ₋₁ p⁽ⁱ⁻¹⁾
    endif
    q⁽ⁱ⁾= [A]p⁽ⁱ⁾
    αᵢ = ρᵢ₋₁/p⁽ⁱ⁾q⁽ⁱ⁾
    x⁽ⁱ⁾= x⁽ⁱ⁻¹⁾ + αᵢp⁽ⁱ⁾
    r⁽ⁱ⁾= r⁽ⁱ⁻¹⁾ – αᵢq⁽ⁱ⁾
    check convergence |r|
end
```

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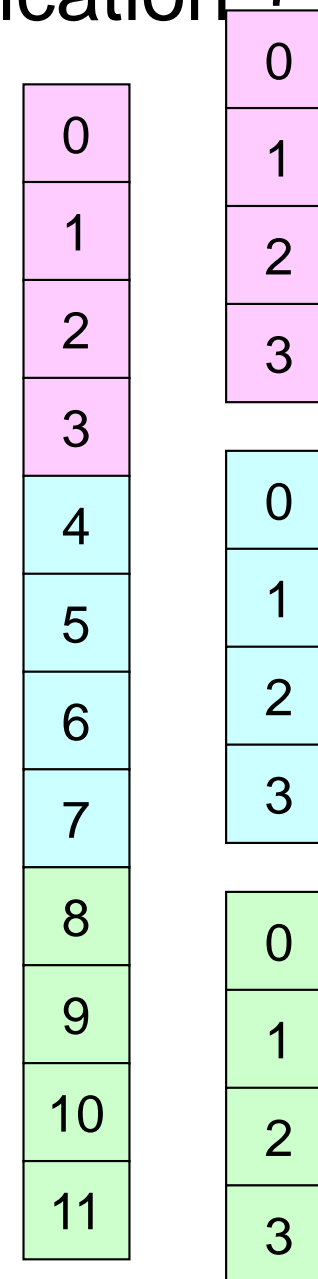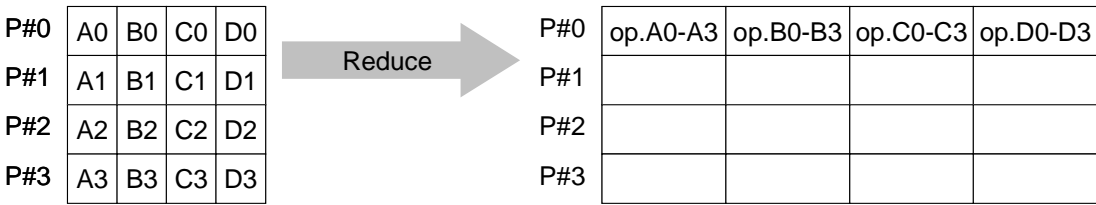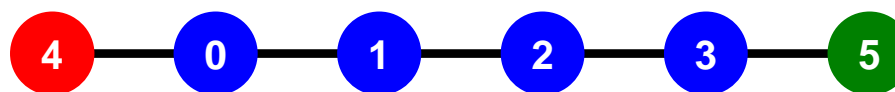- Parallel operations are required in
  - Dot Products
  - Mat-Vec. Multiplication
    - SpMV: Sparse Mat-Vec. Mult.

$$[M] = \begin{bmatrix} D_1 & 0 & ... & 0 & 0 \\ 0 & D_2 & & 0 & 0 \\ ... & & ... & & ... \\ 0 & 0 & & D_{N-1} & 0 \\ 0 & 0 & ... & 0 & D_N \end{bmatrix}$$
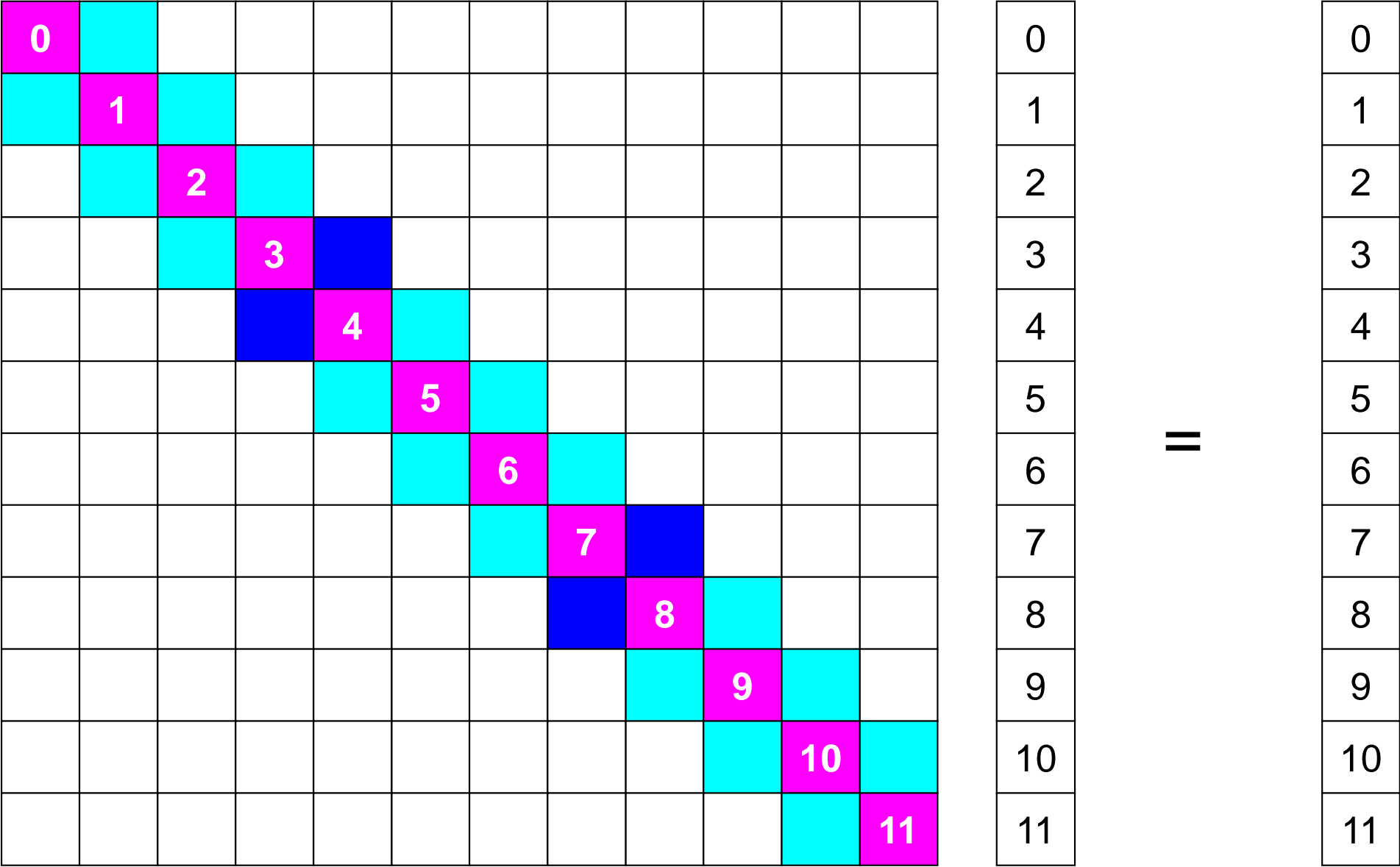
Matrix-Vector Products
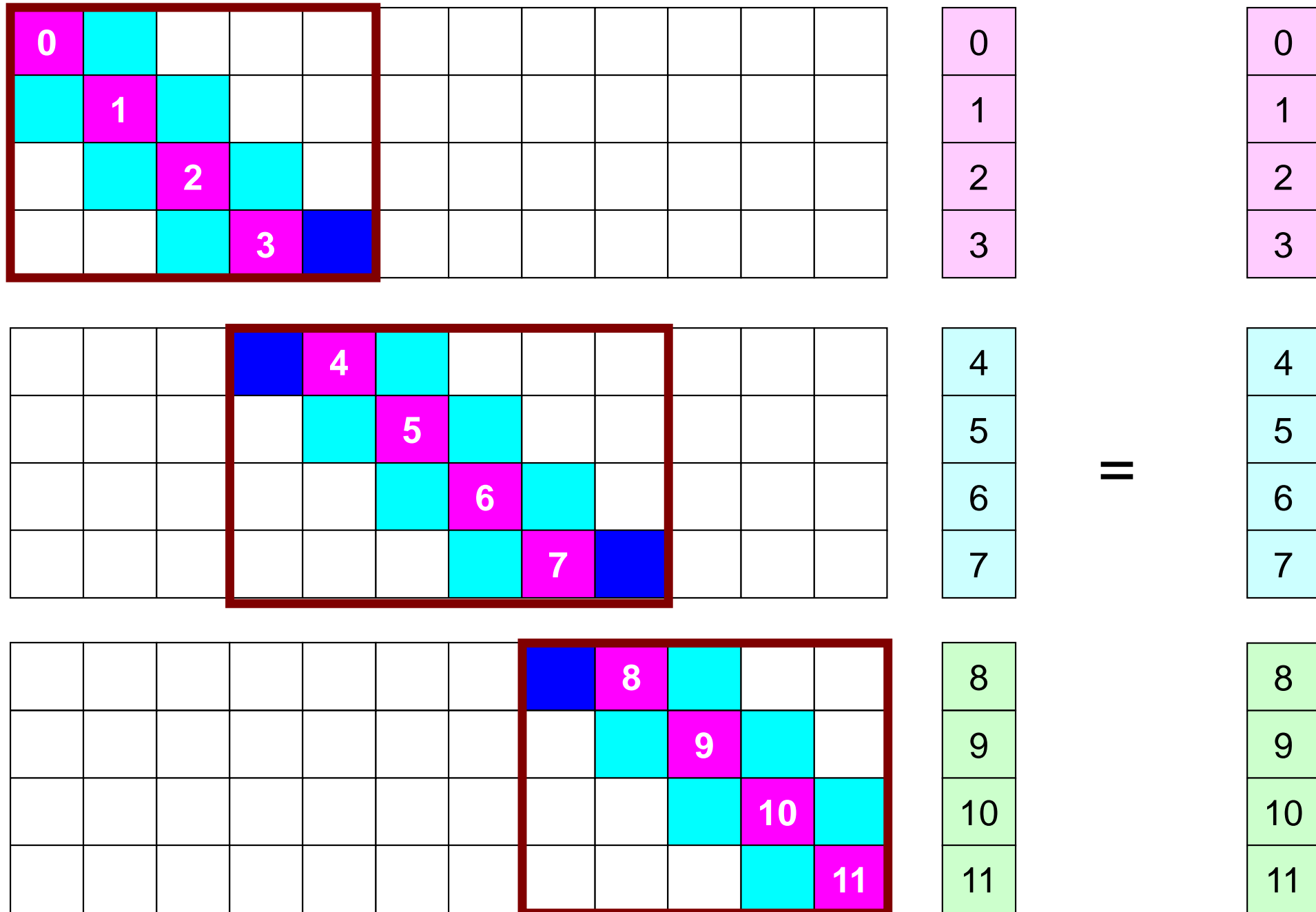## Values at External Points: P-to-P Communication

```
/*
//-- {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]];
          }
      }
```
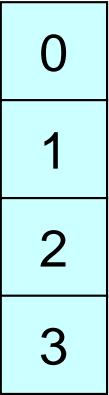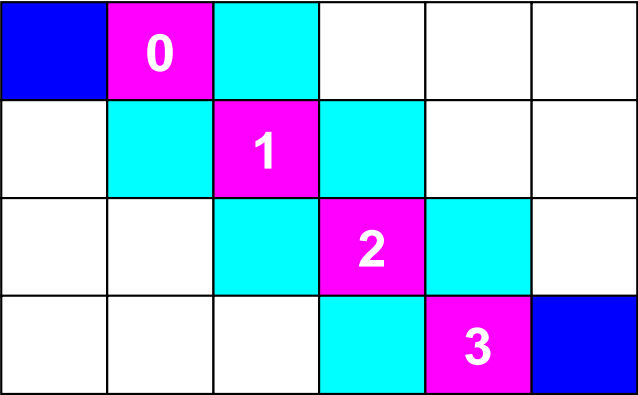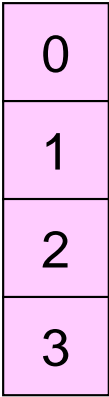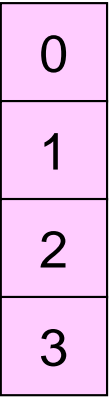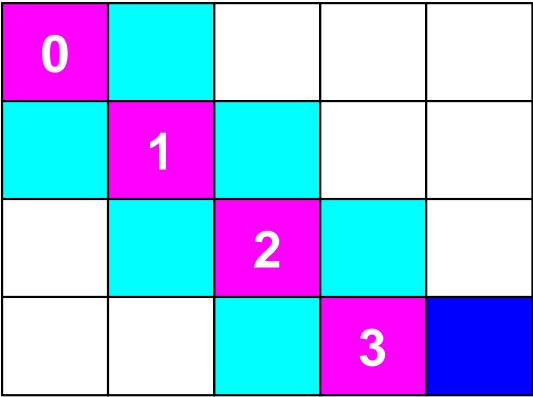
Mat-Vec Products: Local Op. Possible

Mat-Vec Products: Local Op. Possible

Mat-Vec Products: Local Op. Possible

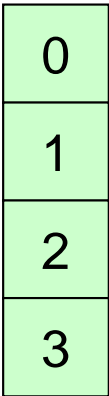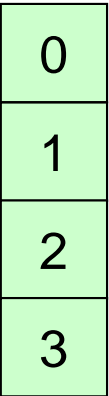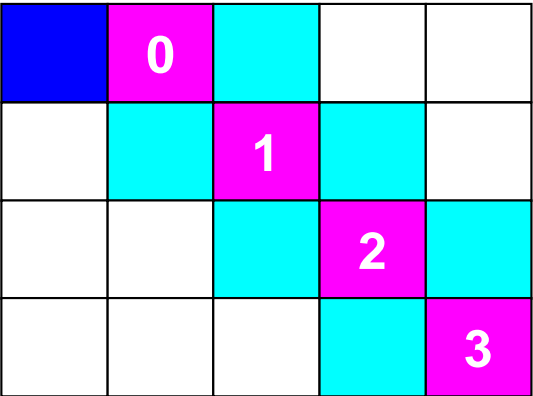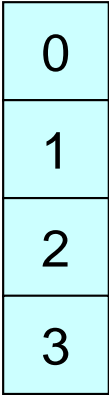# Mat-Vec Products: Local Op. #0
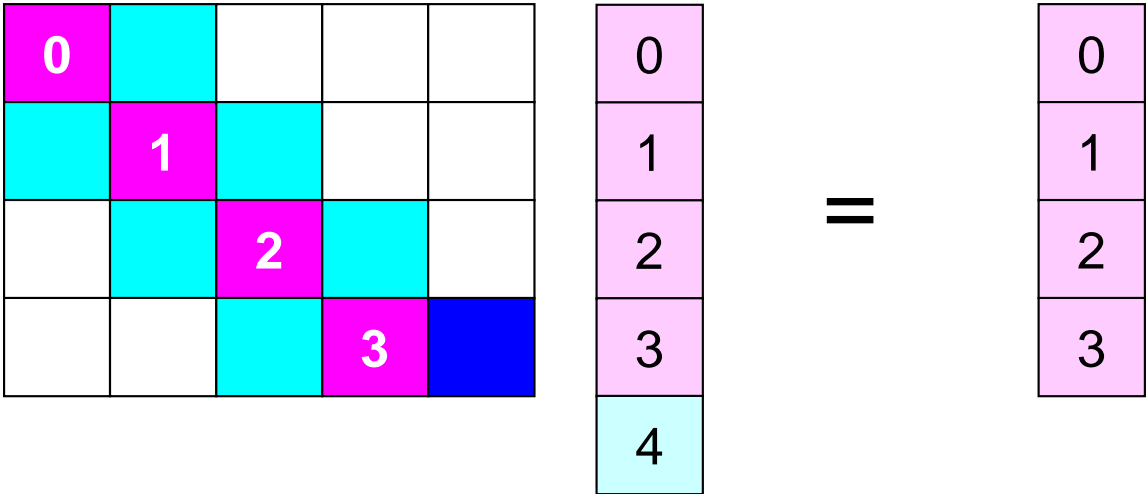
Mat-Vec Products: Local Op. #1
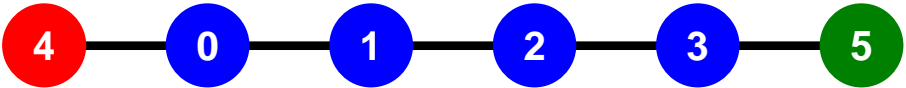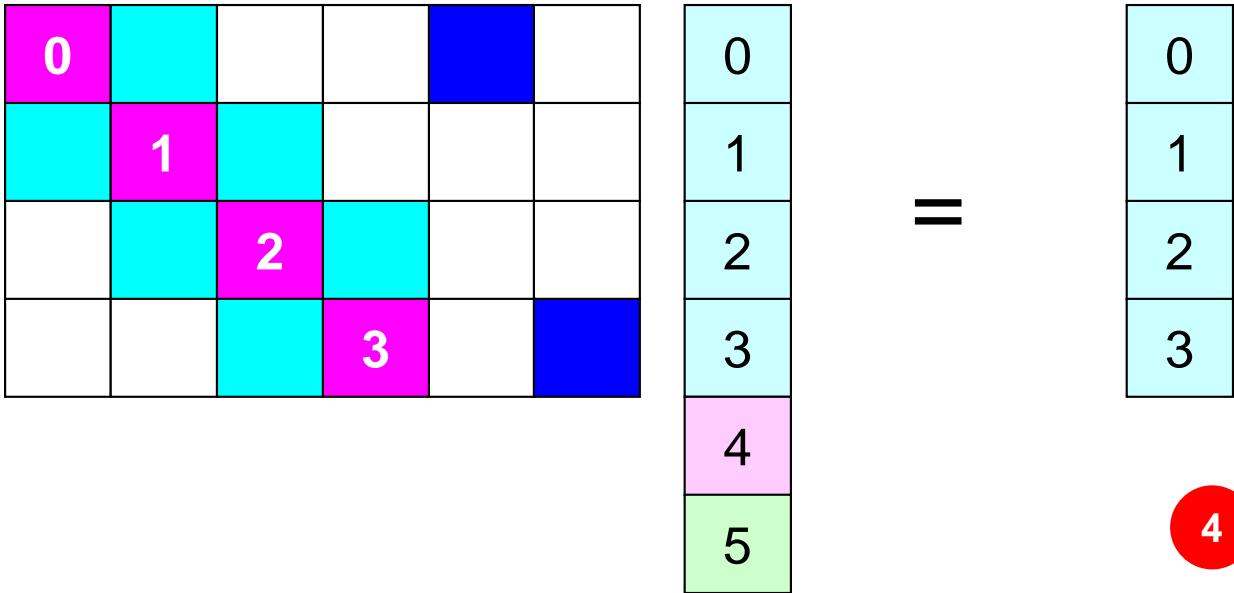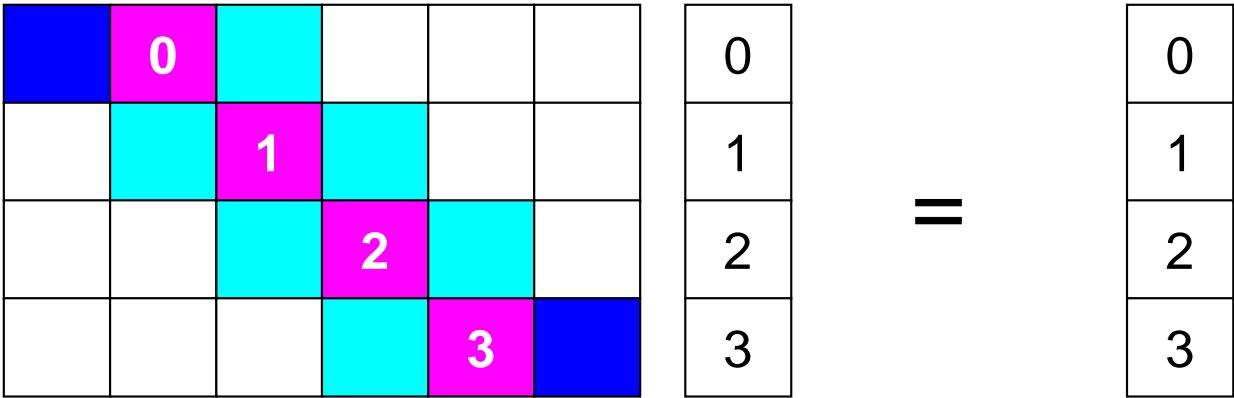
Mat-Vec Products: Local Op. #2
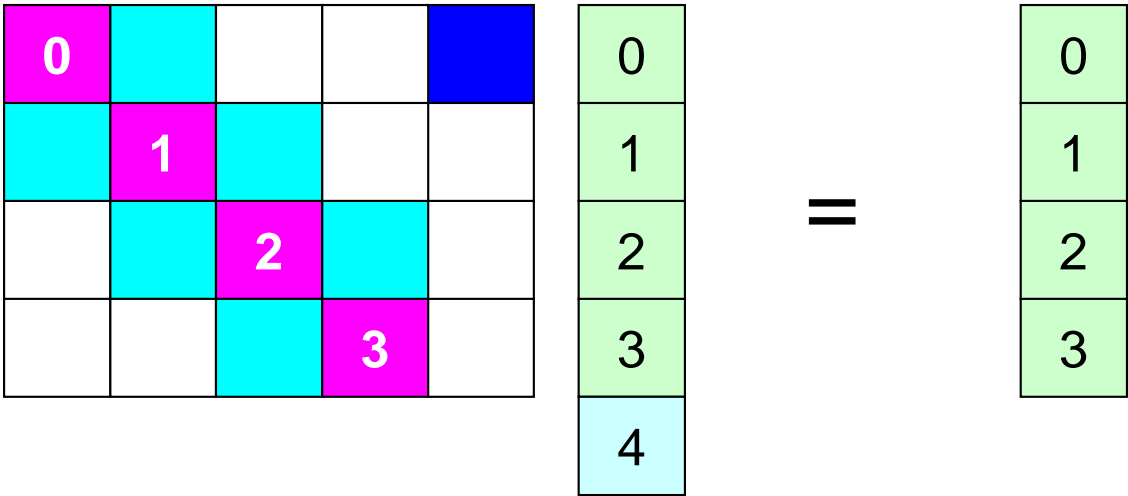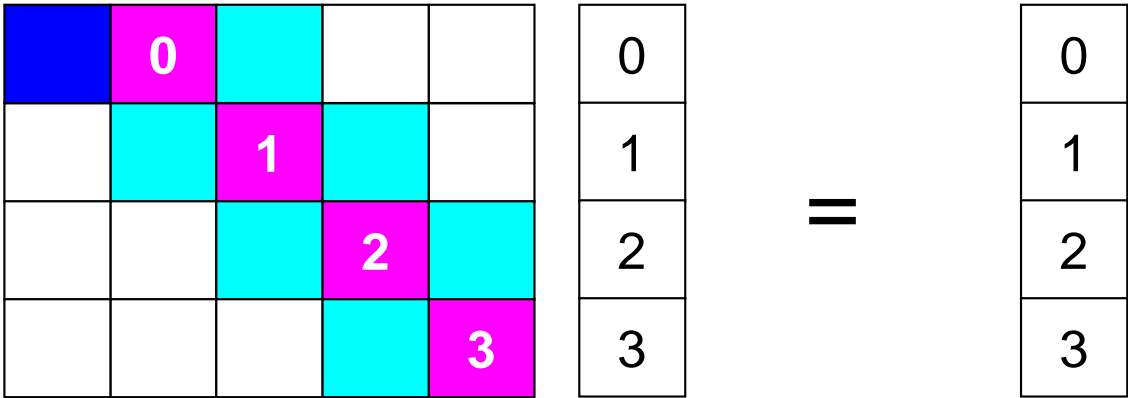
1D FEM: 12 nodes/11 elem's/3 domains
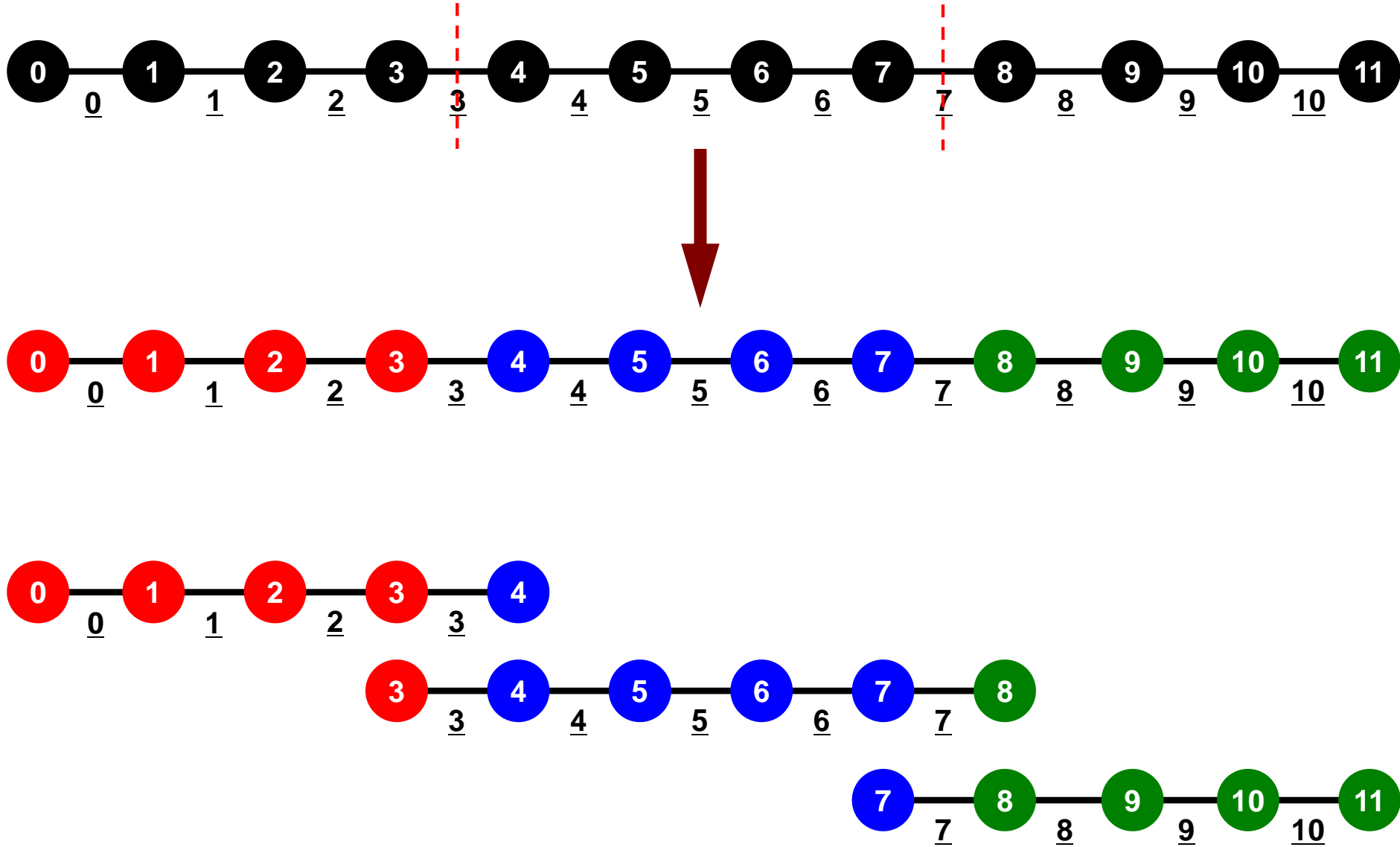
1D FEM: 12 nodes/11 elem's/3 domains
Local ID: Starting from 0 for node and elem at each domain
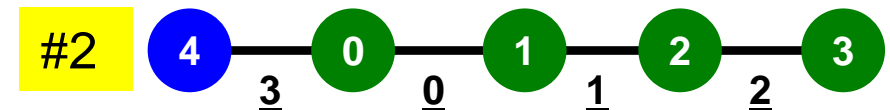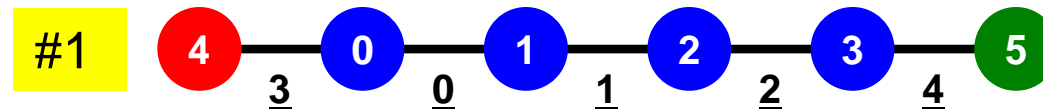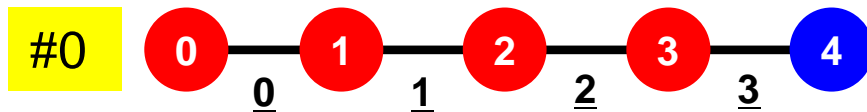
1D FEM: 12 nodes/11 elem's/3 domains
Internal/External/Boundary Nodes
Boundary Nodes: Part of Internal Nodes, and External Nodes
of Other Domains
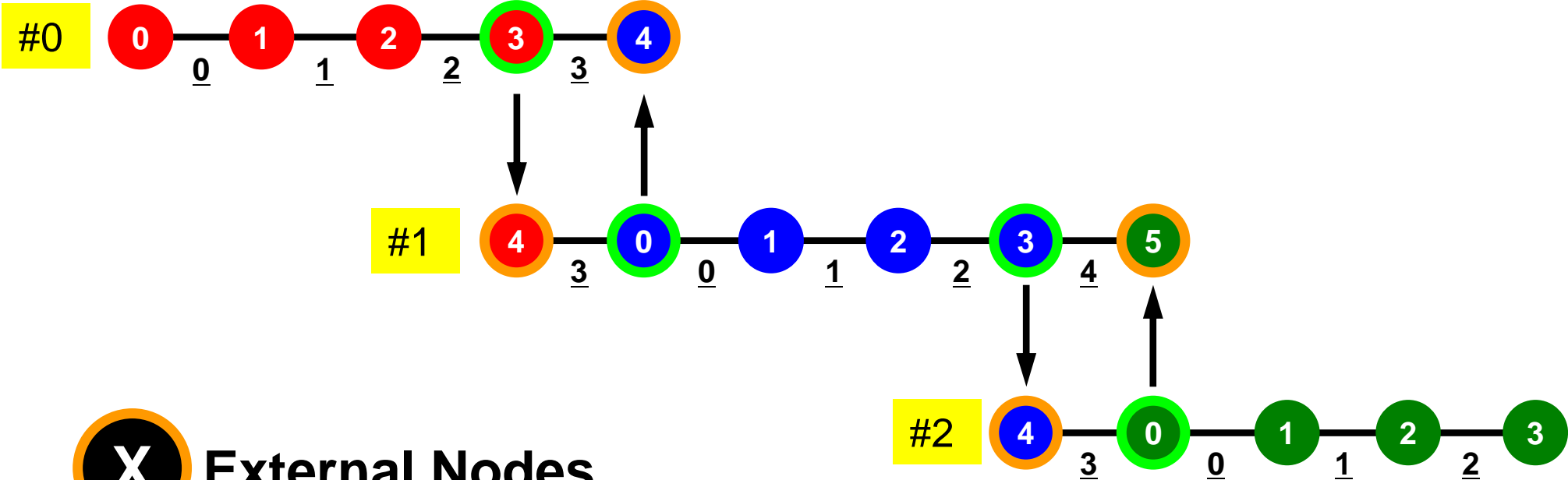
What is Point-to-Point Communication ?

- ## 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

MPI_Isend

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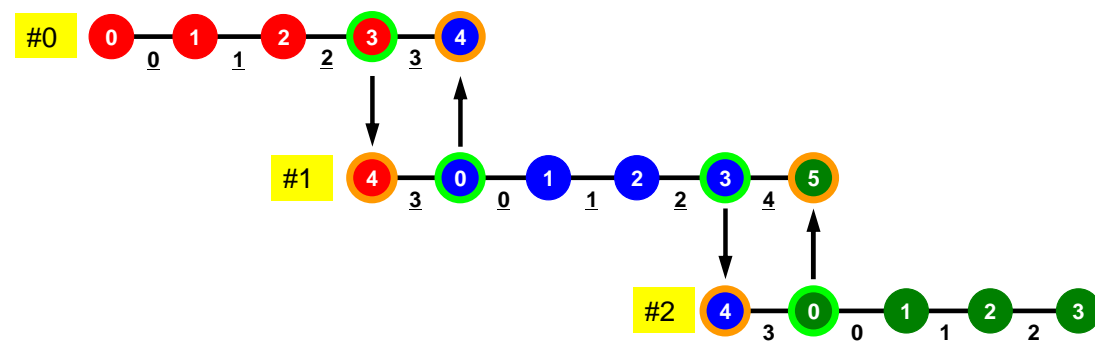
- Begins a non-blocking send
  - Send the contents of sending buffer (starting from **sendbuf**, number of messages: **count**) to **dest** with **tag** .
  - Contents of sending buffer cannot be modified before calling corresponding **MPI_Waitall**.

- **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

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'