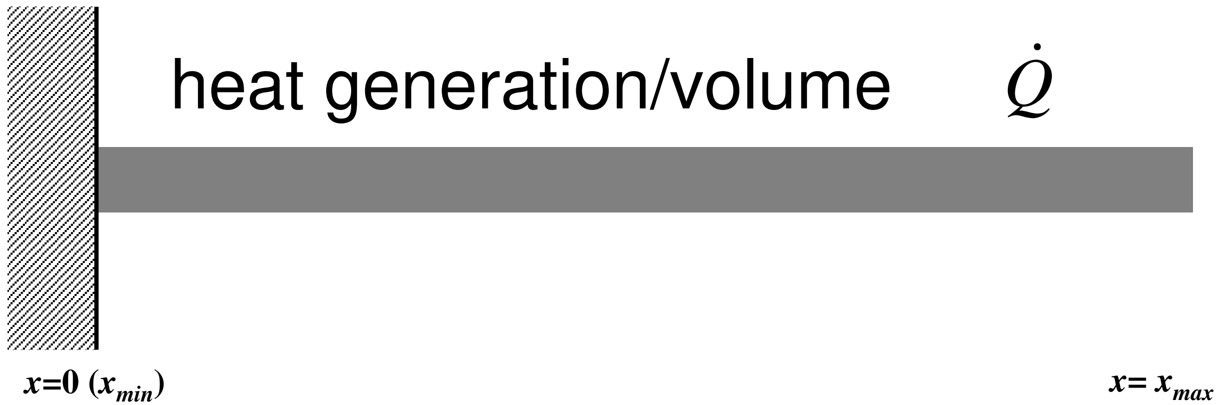
Exercise S2
Fortran

Kengo Nakajima
RIKEN R-CCS

- Overview
- Distributed Local Data
- Program
- Results

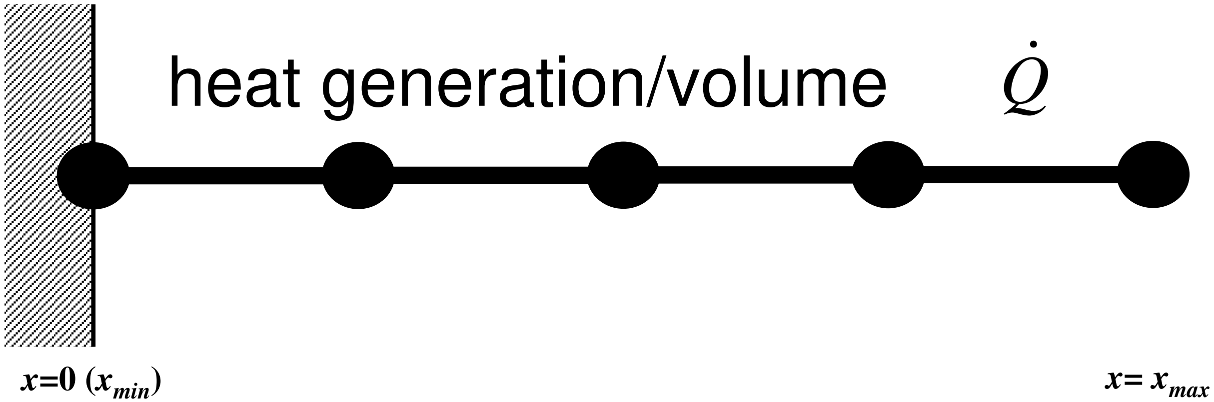
1D Steady State Heat Conduction



$$\frac{\partial}{\partial x}\left(\lambda \frac{\partial T}{\partial x}\right) + \dot{Q} = 0$$

- Uniform: Sectional Area: A, Thermal Conductivity: λ
- Heat Generation Rate/Volume/Time [$QL^{-3}T^{-1}$] $\dot{Q}$
- Boundary Conditions
  - $x=0$: $T=0$ (Fixed Temperature)
  - $x=x_{max}$: $\dfrac{\partial T}{\partial x} = 0$ (Insulated)

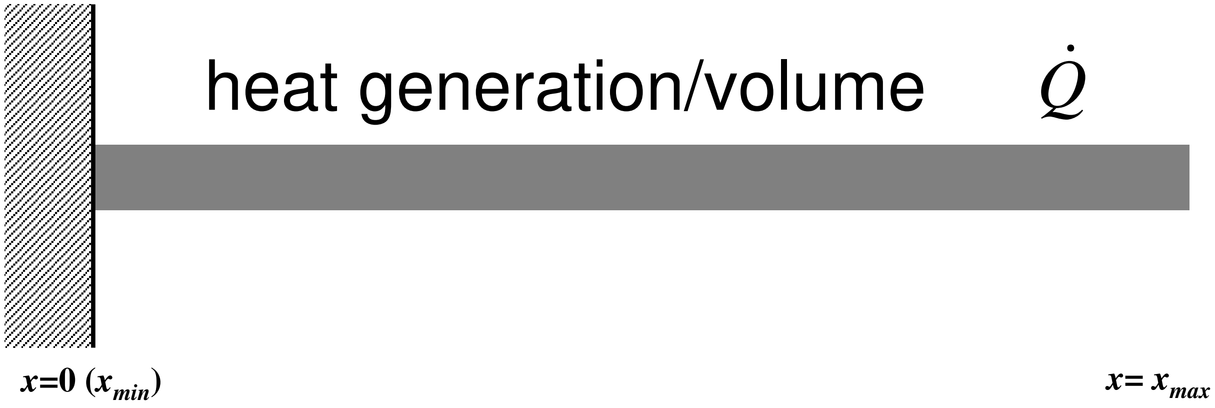
1D Steady State Heat Conduction



$$\frac{\partial}{\partial x}\left(\lambda\frac{\partial T}{\partial x}\right)+\dot{Q}=0$$

- Uniform: Sectional Area: A, Thermal Conductivity: λ
- Heat Generation Rate/Volume/Time [$QL^{-3}T^{-1}$] $\dot{Q}$
- Boundary Conditions
  - $x=0$: $T=0$ (Fixed Temperature)
  - $x=x_{max}$: $\dfrac{\partial T}{\partial x}=0$ (Insulated)

Analytical Solution



$$\frac{\partial}{\partial x}\left(\lambda \frac{\partial T}{\partial x}\right) + \dot{Q} = 0$$

$$T = 0 \; @ \; x = 0$$

$$\frac{\partial T}{\partial x} = 0 \; @ \; x = x_{\max}$$

$$\lambda T'' = -\dot{Q}$$

$$\lambda T' = -\dot{Q}x + C_1 \Rightarrow C_1 = \dot{Q}x_{\max}, \quad T' = 0 \; @ \; x = x_{\max}$$

$$\lambda T = -\frac{1}{2}\dot{Q}x^2 + C_1 x + C_2 \Rightarrow C_2 = 0, \quad T = 0 \; @ \; x = 0$$

$$\therefore \quad T = -\frac{1}{2\lambda}\dot{Q}x^2 + \frac{\dot{Q}x_{\max}}{\lambda}x$$

Report S2 (1/2)

- Parallelize 1D code (1d.f) using MPI
- Read entire element number, and decompose into sub-domains in your program

- Validate the results
  - Answer of Original Code = Answer of Parallel Code
  - Explain why number of iterations does not change, as number of MPI processes changes.
- Measure parallel performance

Report S2 (2/2)

- Deadline: January 25th (Wed), 2023, 17:00@ITC-LMS
- Problem
  - Apply "Generalized Communication Table"
  - Read entire elem. #, decompose into sub-domains in your program
  - Evaluate parallel performance
    - You need huge number of elements, to get excellent performance.
    - Fix number of iterations (e.g. 100), if computations cannot be completed.
- Report
  - Cover Page: Name, ID, and Problem ID (S2) must be written.
  - Less than eight pages including figures and tables (A4).
    - Strategy, Structure of the Program, Remarks
  - Source list of the program (if you have bugs)
  - Output list (as small as possible)

Copy and Compile

Fortran

```
>$ cd /work/gt18/t18XXX/pFEM
>$ moule load fj
>$ cp /work/gt00/z30088/pFEM/F/s2r-f.tar .
>$ tar xvf s2r-f.tar
```

C

```
>$ cd /work/gt18/t18XXX/pFEM
>$ module load fj
>$ cp /work/gt00/z30088/pFEM/C/s2r-c.tar .
>$ tar xvf s2r-c.tar
```

Confirm/Compile

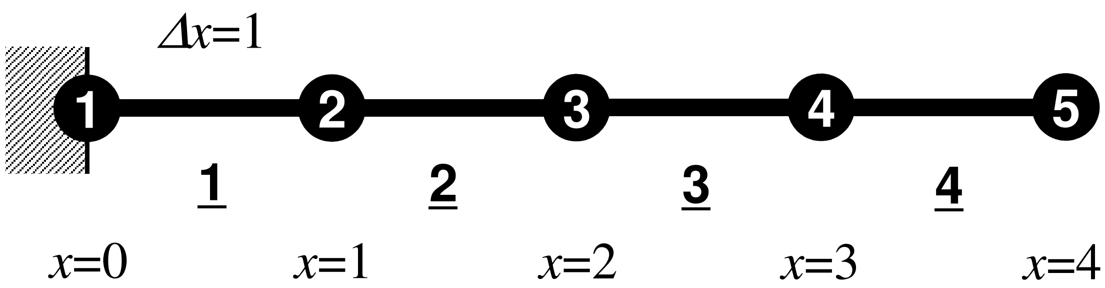
```
>$ cd mpi/S2-ref
>$ mpifrtpx -Kfast 1d.f -o 1d
>$ mpifrtpx -Kfast 1d2.f -o 1d2

>$ mpifccpx -Nclang -Kfast 1d.c -o 1d
>$ mpifccpx -Nclang -Kfast 1d2.c -o 1d2
```

<$O-S2r> = <$O-TOP>/mpi/S2-ref

Control File: input.dat

Control Data `input.dat`

`1000000`	`NE` (Number of Elements)
`1.0 1.0 1.0 1.0`	Δx (Length of Each Elem.: `L`) , `Q`, `A`, λ
`100`	Number of MAX. Iterations for CG Solver
`1.e-8`	Convergence Criteria for CG Solver



Element ID
Node ID (Global)

a384.sh: 8-nodes, 384-cores

```
#!/bin/sh
#PJM -N "m384"
#PJM -L rscgrp=lecture8-o
#PJM -L node=8
#PJM --mpi proc=384
#PJM -L elapse=00:15:00
#PJM -g gt18
#PJM -j
#PJM -e err
#PJM -o z384.lst

module load fj
module load fjmpi

mpiexec ./1d
mpiexec ./1d
mpiexec ./1d
mpiexec ./1d
mpiexec ./1d
```

a012.sh

```
#!/bin/sh
#PJM -N "test"
#PJM -L rscgrp=lecture8-o
#PJM -L node=1
#PJM --mpi proc=12
#PJM -L elapse=00:15:00
#PJM -g gt18
#PJM -j
#PJM -e err
#PJM -o test.lst

module load fj
module load fjmpi
mpiexec ./a.out
mpiexec numactl -l ./a.out
```

a048.sh

```
#!/bin/sh
#PJM -N "test"
#PJM -L rscgrp=lecture8-o
#PJM -L node=1
#PJM --mpi proc=48
#PJM -L elapse=00:15:00
#PJM -g gt18
#PJM -j
#PJM -e err
#PJM -o test.lst

module load fj
module load fjmpi
mpiexec ./a.out
mpiexec numactl -l ./a.out
```

a384.sh

```
#!/bin/sh
#PJM -N "test"
#PJM -L rscgrp=lecture8-o
#PJM -L node=8
#PJM --mpi proc=384
#PJM -L elapse=00:15:00
#PJM -g gt18
#PJM -j
#PJM -e err
#PJM -o test.lst

module load fj
module load fjmpi
mpiexec ./a.out
mpiexec numactl -l ./a.out
```

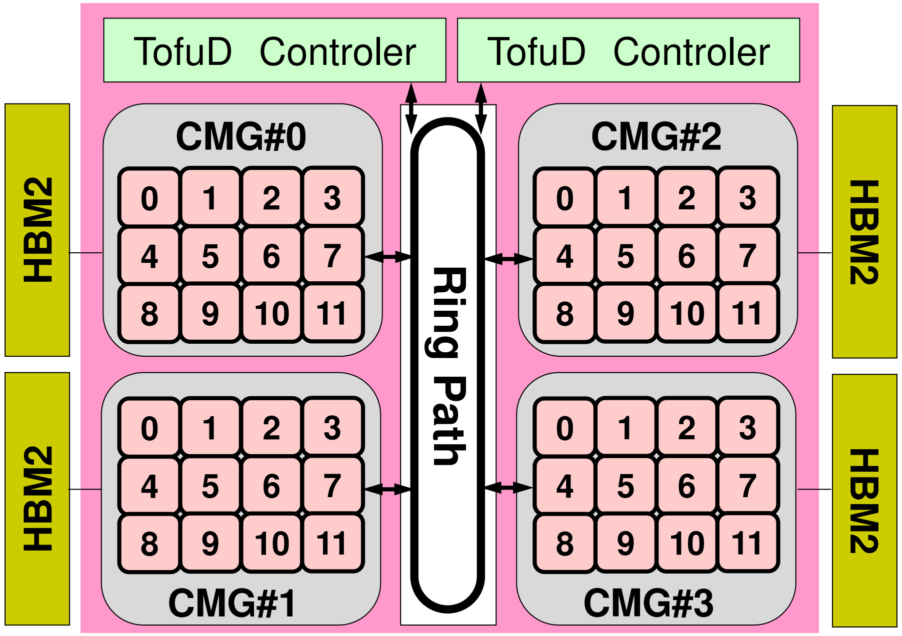
a576.sh

```
#!/bin/sh
#PJM -N "test"
#PJM -L rscgrp=lecture8-o
#PJM -L node=12
#PJM --mpi proc=576
#PJM -L elapse=00:15:00
#PJM -g gt18
#PJM -j
#PJM -e err
#PJM -o test.lst

module load fj
module load fjmpi
mpiexec ./a.out
mpiexec numactl -l ./a.out
```

`numactl -l/--localalloc` for utilizing local memory (no effects)

Number of Processes

```
#PJM -L node=1;#PJM --mpi proc= 1    1-node, 1-proc, 1-proc/n
#PJM -L node=1;#PJM --mpi proc= 4    1-node, 4-proc, 4-proc/n
#PJM -L node=1;#PJM --mpi proc=12    1-node,12-proc,12-proc/n
#PJM -L node=1;#PJM --mpi proc=24    1-node,24-proc,24-proc/n
#PJM -L node=1;#PJM --mpi proc=48    1-node,48-proc,48-proc/n

#PJM -L node= 4;#PJM --mpi proc=192   4-node,192-proc,48-proc/n
#PJM -L node= 8;#PJM --mpi proc=384   8-node,384-proc,48-proc/n
#PJM -L node=12;#PJM --mpi proc=576  12-node,576-proc,48-proc/n
```



Example (1/2)

```
>$ cd /work/gt18/t18XYZ/pFEM/mpi/S2-ref
(modify input.dat,go1.sh)

>$ pjsub go1.sh

(see go1.lst)
```

go1.sh: a single process (1 core)

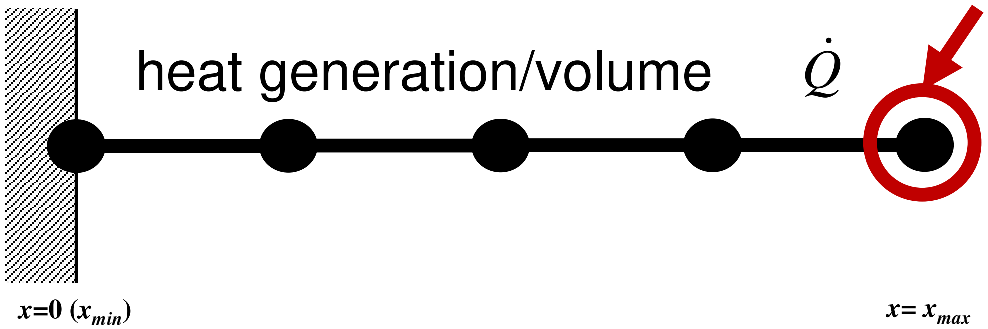
```
#!/bin/sh
#PJM -N "go1"
#PJM -L rscgrp=lecture8-o
#PJM -L node=1
#PJM --mpi proc=1
#PJM -L elapse=00:15:00
#PJM -g gt18
#PJM -j
#PJM -e err
#PJM -o go1.lst

module load fj
module load fjmpi

mpiexec ./1d
```

input.dat (10^4 elements, 1,000 iterations)

```
10000
1.0  1.0  1.0  1.0
1000
```

go1.lst

```
   10000
   1000      9.000337E+01  (=|b-Ax|)
   1001      9.000337E+01
   1.397971E-04    4.453332E-02sec.

### TEMPERATURE
      0    10001  9.500000000000E+06
```



Example (2/2)

```
>$ cd /work/gt18/t18XYZ/pFEM/mpi/S2-ref
 (modify input.dat,go2.sh)

>$ pjsub go2.sh

 (see go2.lst)
```

go2.lst

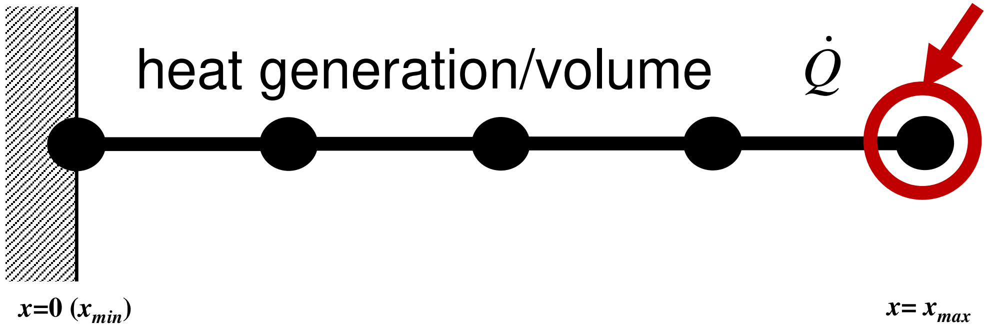
```
      10000
     1000      9.000337E+01(=|b-Ax|)
     1001      9.000337E+01
     4.786998E-07    5.098703E-02sec.

### TEMPERATURE
      383        26 9.500000000000E+06
```

go2.sh: 384 process (384 cores)

```
#!/bin/sh
#PJM -N "go2"
#PJM -L rscgrp=lecture8-o
#PJM -L node=8
#PJM --mpi proc=384
#PJM -L elapse=00:15:00
#PJM -g gt18
#PJM -j
#PJM -e err
#PJM -o go2.lst

module load fj
module load fjmpi

mpiexec ./1d
```

input.dat (10^4 elements, 1,000 iterations)

```
10000
1.0  1.0  1.0  1.0
1000
```

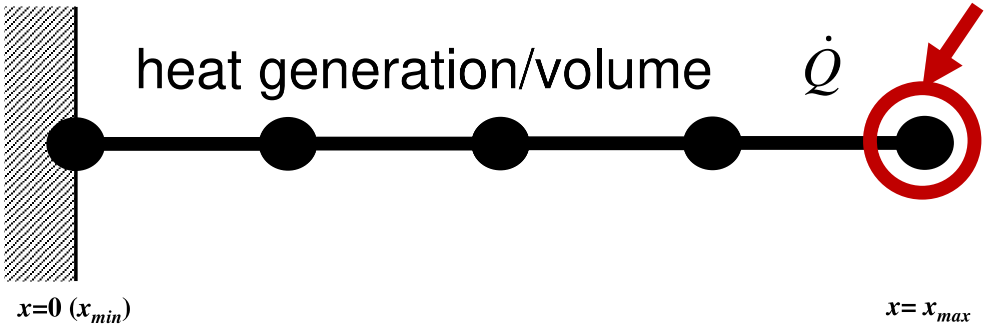


1D Code on PC

input.dat (10^4 elements, 10,000 iterations)

```
10000
1.0   1.0   1.0   1.0
10000
```

```
 10001       5.000E+07       5.000E+07
```



input.dat (10^4 elements, 1,000 iterations)

```
10000
1.0   1.0   1.0   1.0
1000
```

```
 10001       9.500E+06       5.000E+07
```

Procedures for Parallel FEM

- Reading control file, entire element number etc.
- Creating “distributed local data” in the program
- Assembling local and global matrices for linear solvers
- Solving linear equations by CG

- Not so different from those of original code

- Overview
- **Distributed Local Data**
- Program
- Results

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

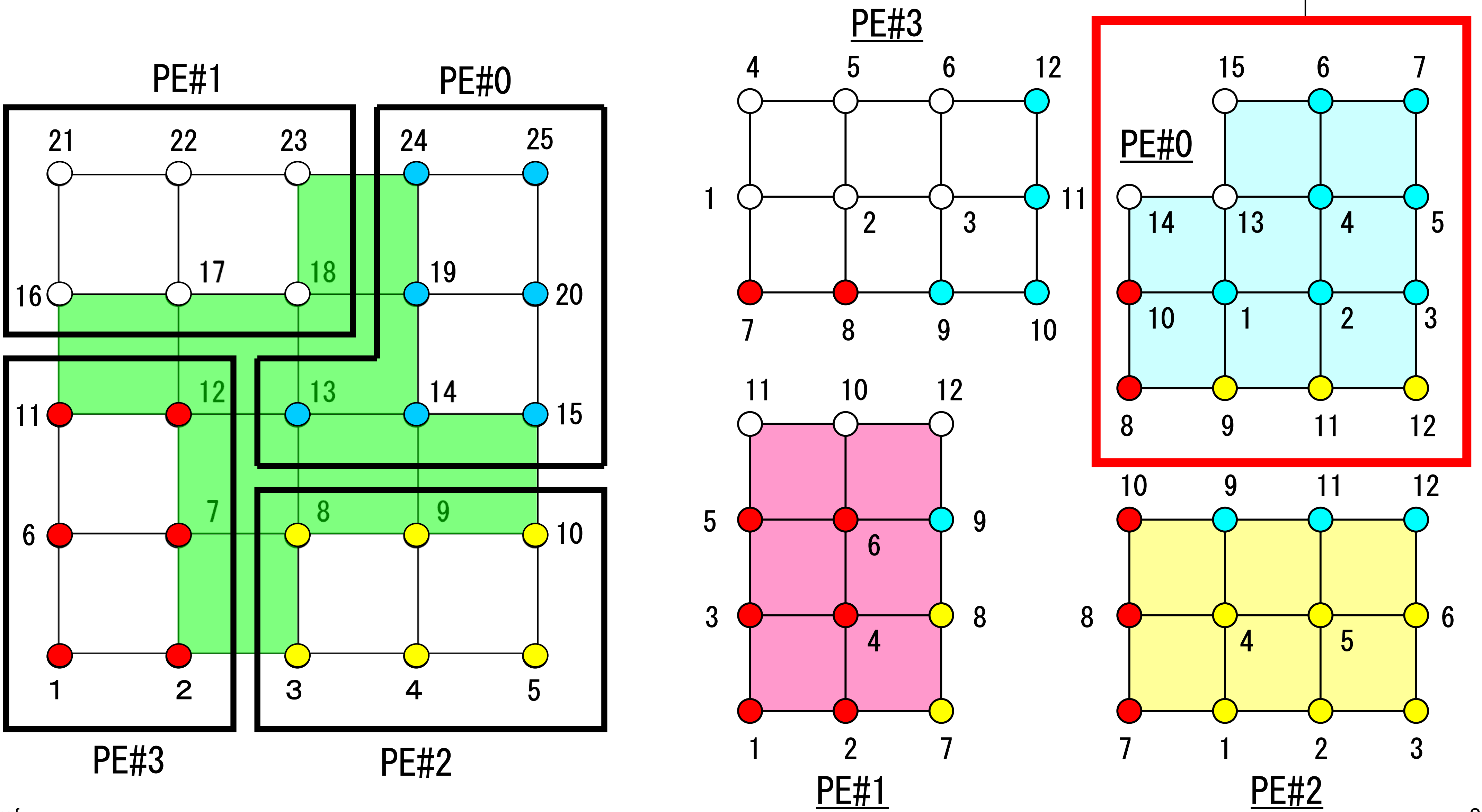
Distributed Local Data Structure for Parallel FEM

- Node-based partitioning
- Local data includes:
  - Nodes originally assigned to the domain/PE/partition
  - Elements which include above nodes
  - Nodes which are included above elements, and originally NOT-assigned to the domain/PE/partition
- 3 categories for nodes
  - Internal nodes   Nodes originally assigned to the domain/PE/partition
  - External nodes   Nodes originally NOT-assigned to the domain/PE/partition
  - Boundary nodes   External nodes of other domains/PE's/partitions
- Communication tables

- Global info. is not needed except relationship between domains
  - Property of FEM: local element-by-element operations

Node-based Partitioning

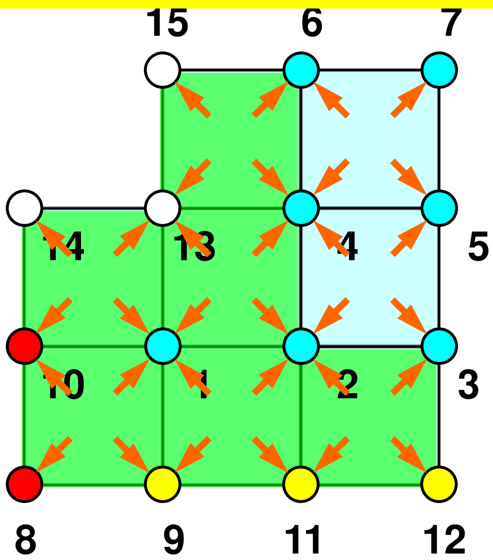
internal nodes - elements - external nodes

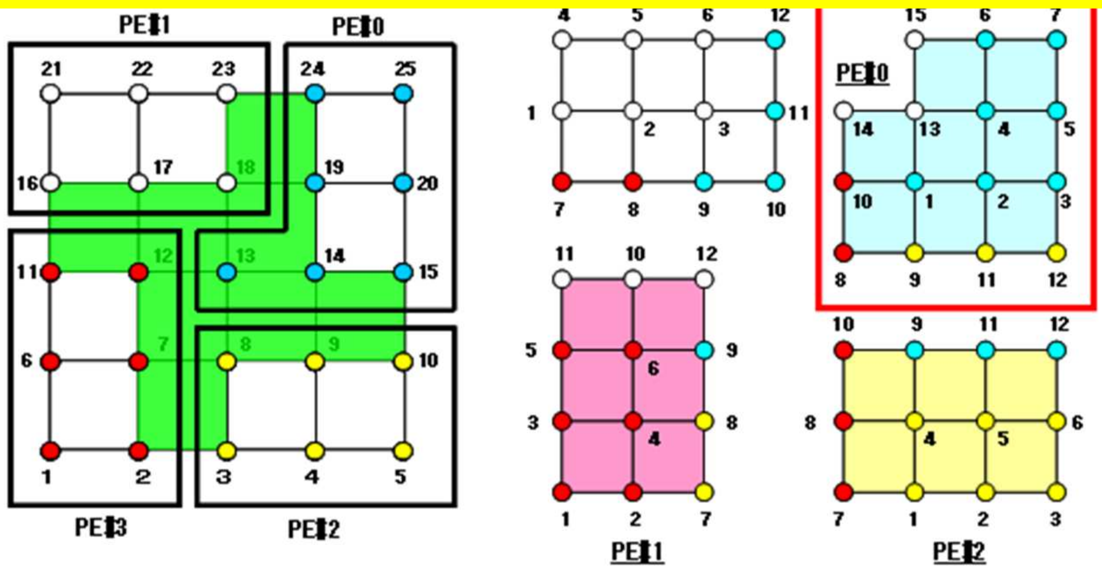


Node-based Partitioning

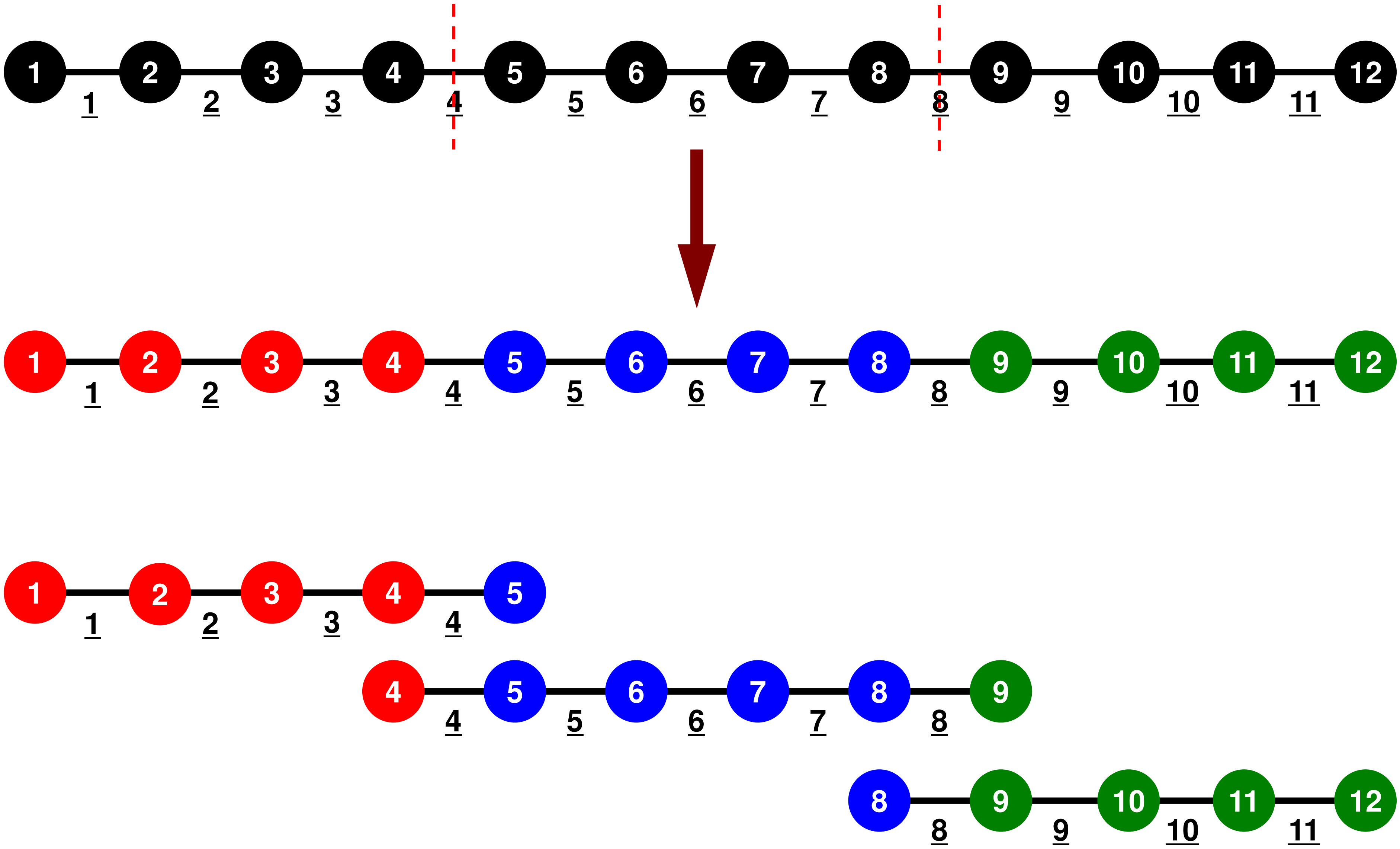
internal nodes - elements - external nodes

- Partitioned nodes themselves (<u>Internal Nodes</u>) 内点
- Elements which include Internal Nodes 内点を含む要素
- <u>External Nodes</u> 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.

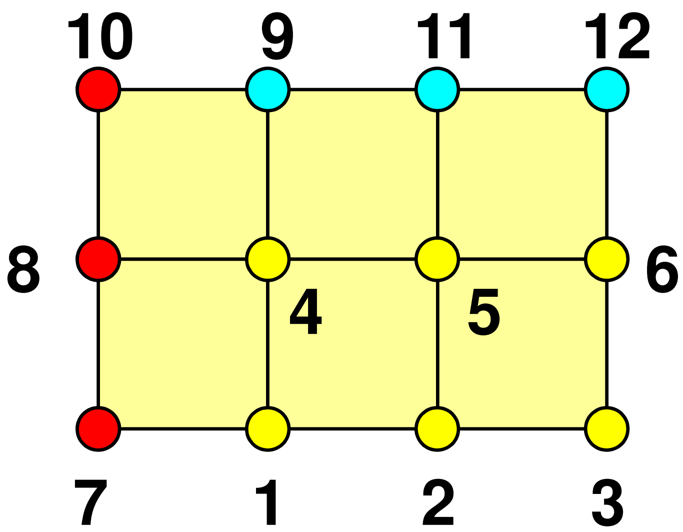




1D FEM: 12 nodes/11 elem's/3 domains



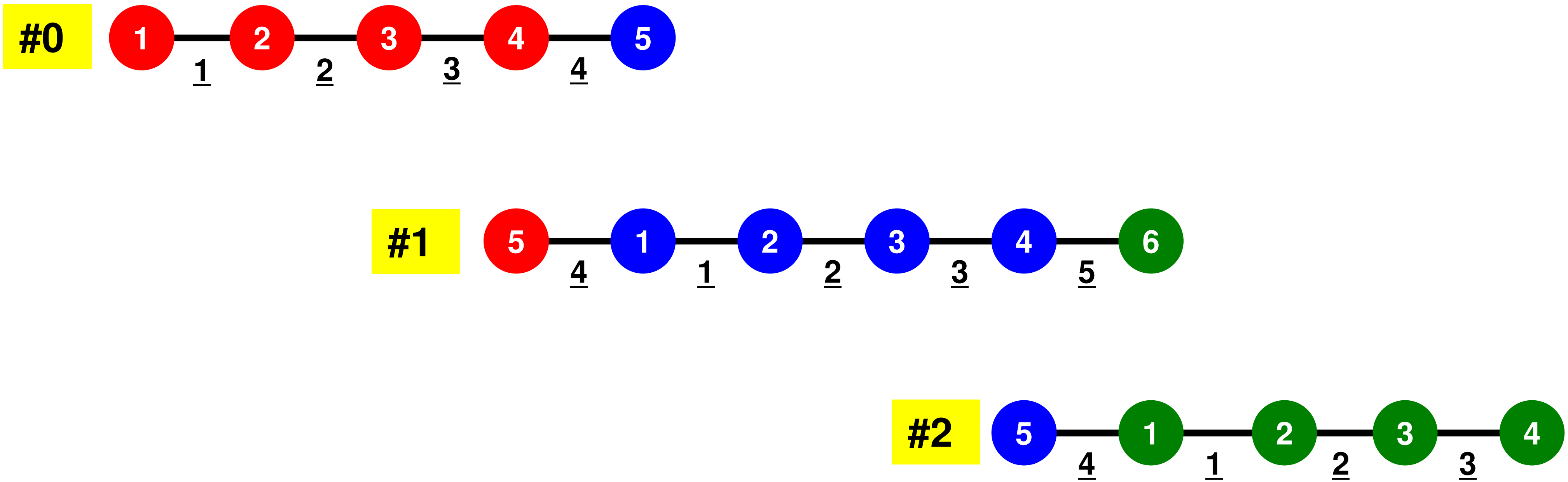
Description of Distributed Local Data



- 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 ?

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

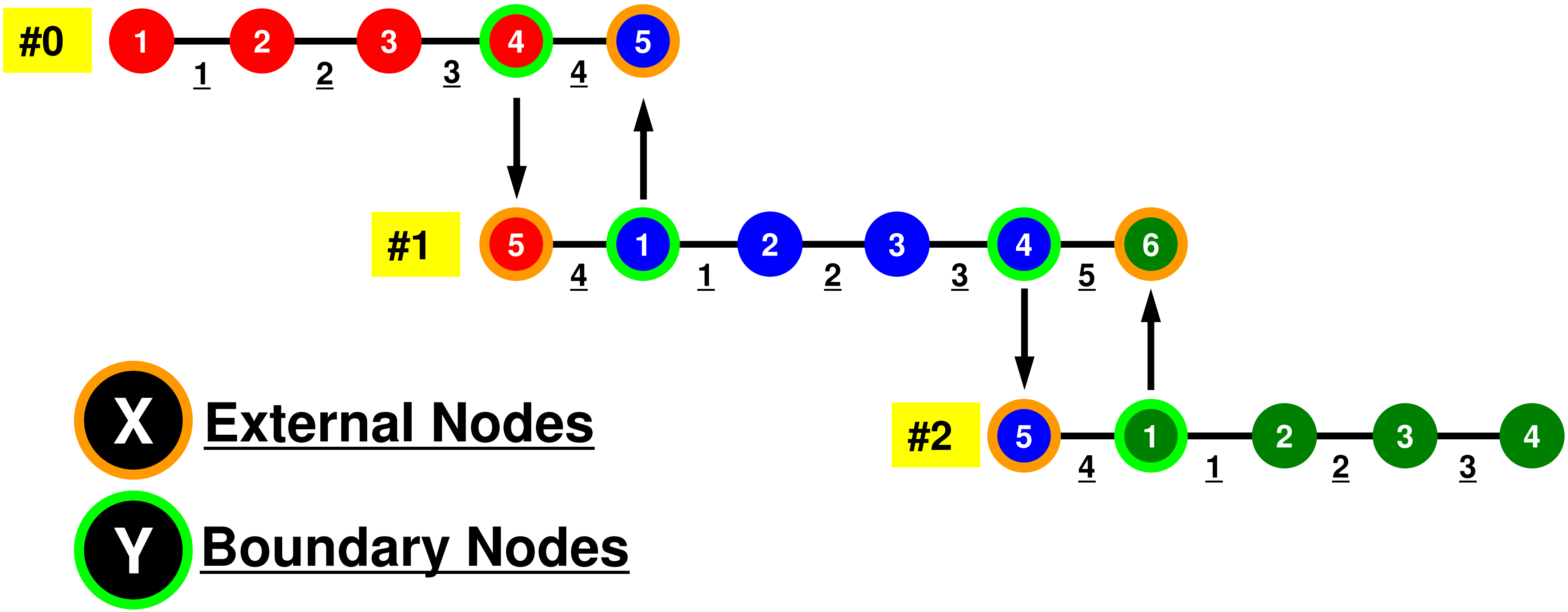
Local ID: Starting from 1 for node and elem at each domain



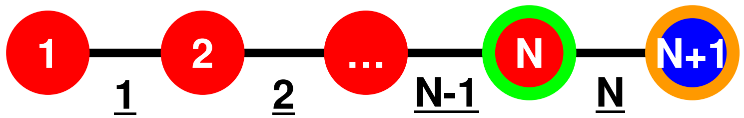
1D FEM: 12 nodes/11 elem's/3 domains

Internal/External/Boundary Nodes

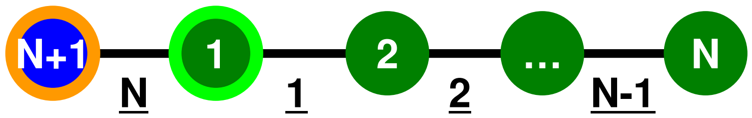
Boundary Nodes: Part of Internal Nodes, and External Nodes of Other Domains



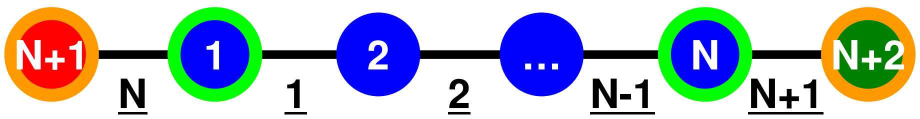
1D FEM: Numbering of Local ID



#0:
N+1 nodes
N elements



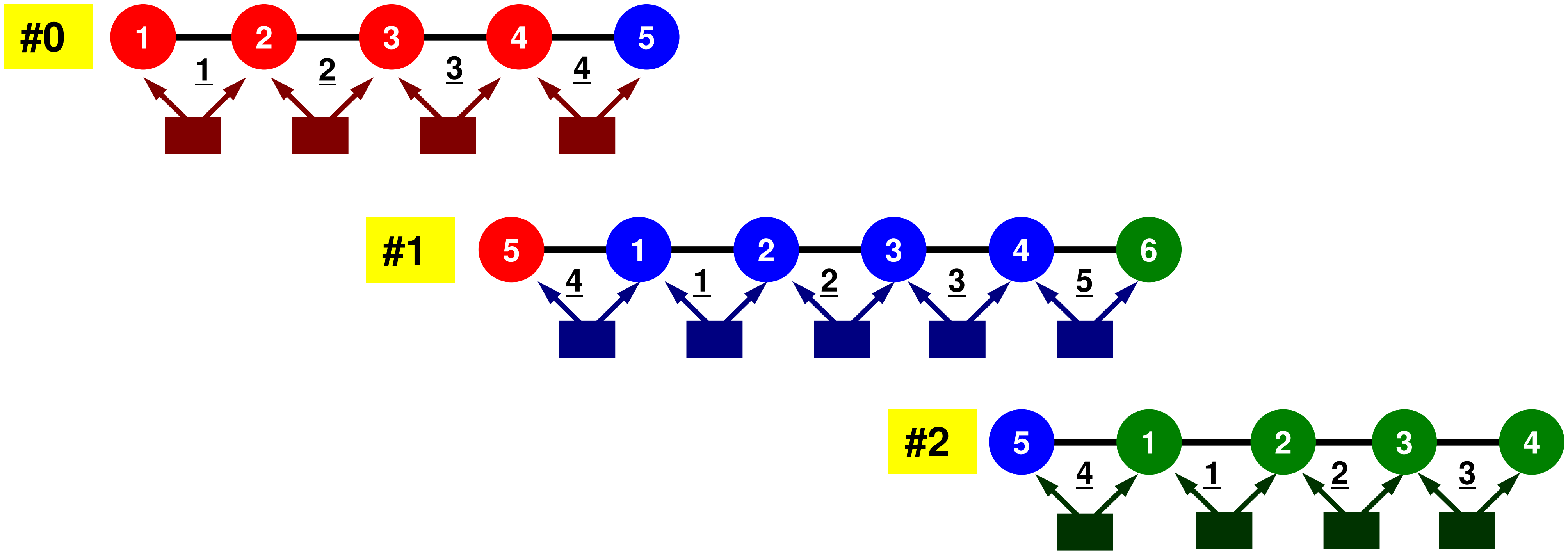
#PETot-1:
N+1 nodes
N elements



Others (General):
N+2 nodes
N+1 elements

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



Preconditioned Conjugate Gradient Method (CG)

```
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
```

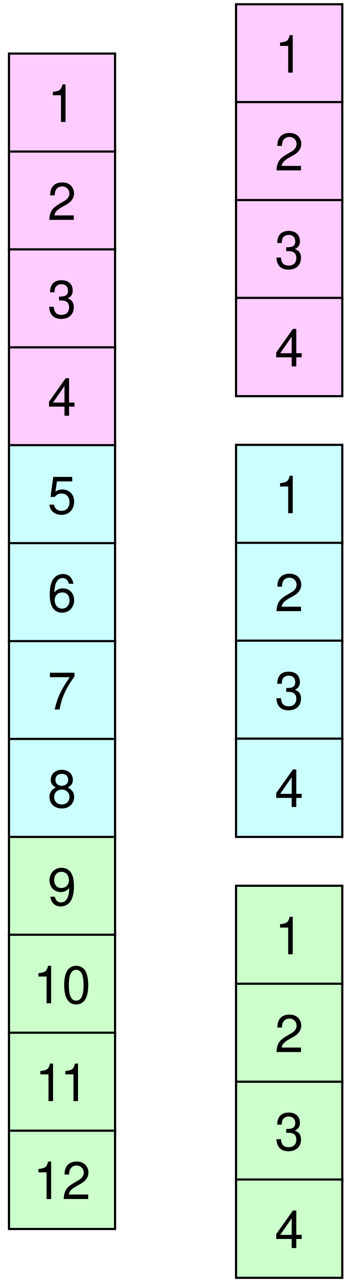
Preconditioning:
Diagonal Scaling
(or Point Jacobi)

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
```

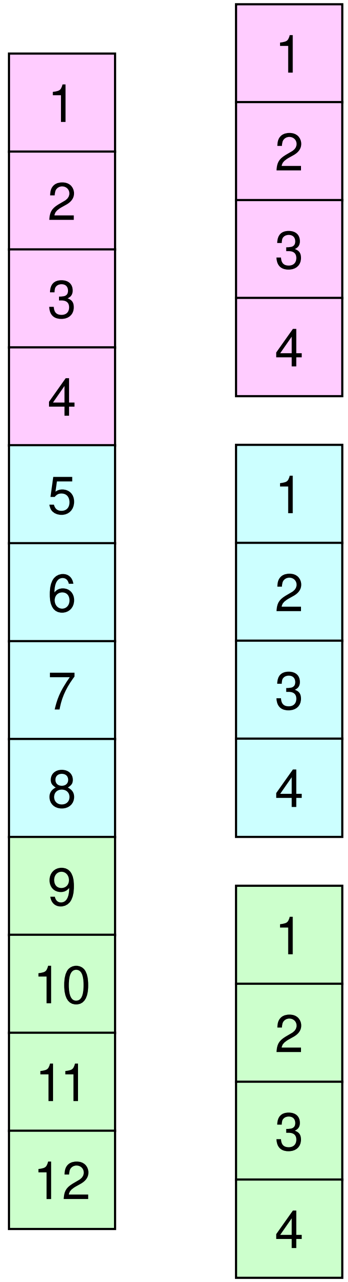
```
!C
!C-- {x}= {x} + ALPHA*{p}
!C    {r}= {r} - ALPHA*{q}

      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 ?

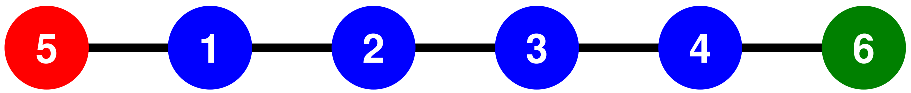
```
!C
!C-- ALPHA= RHO / {p}{q}

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



Matrix-Vector Products

Values at External Points: P-to-P Communication

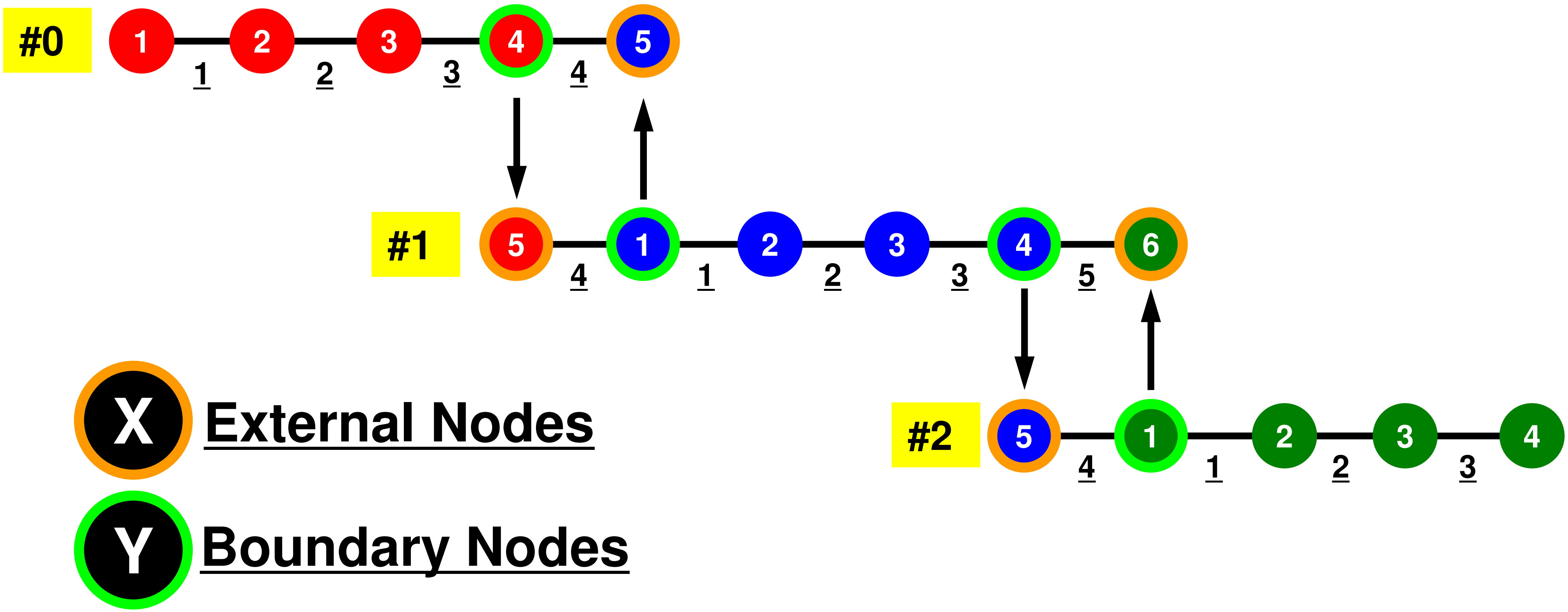
```
!C
!C-- {q}= [A]{p}

      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
```



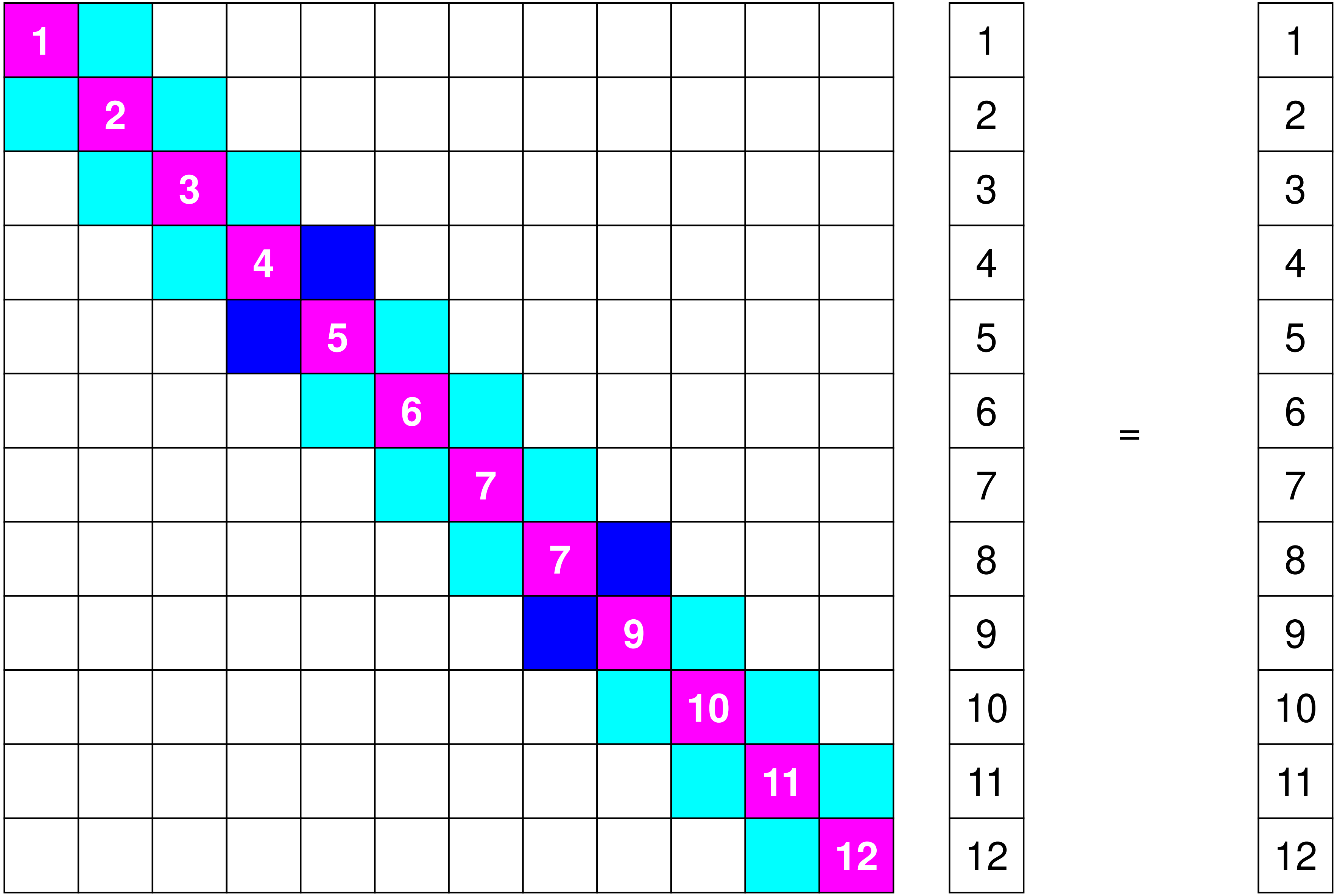
1D FEM: 12 nodes/11 elem's/3 domains

Internal/External/Boundary Nodes

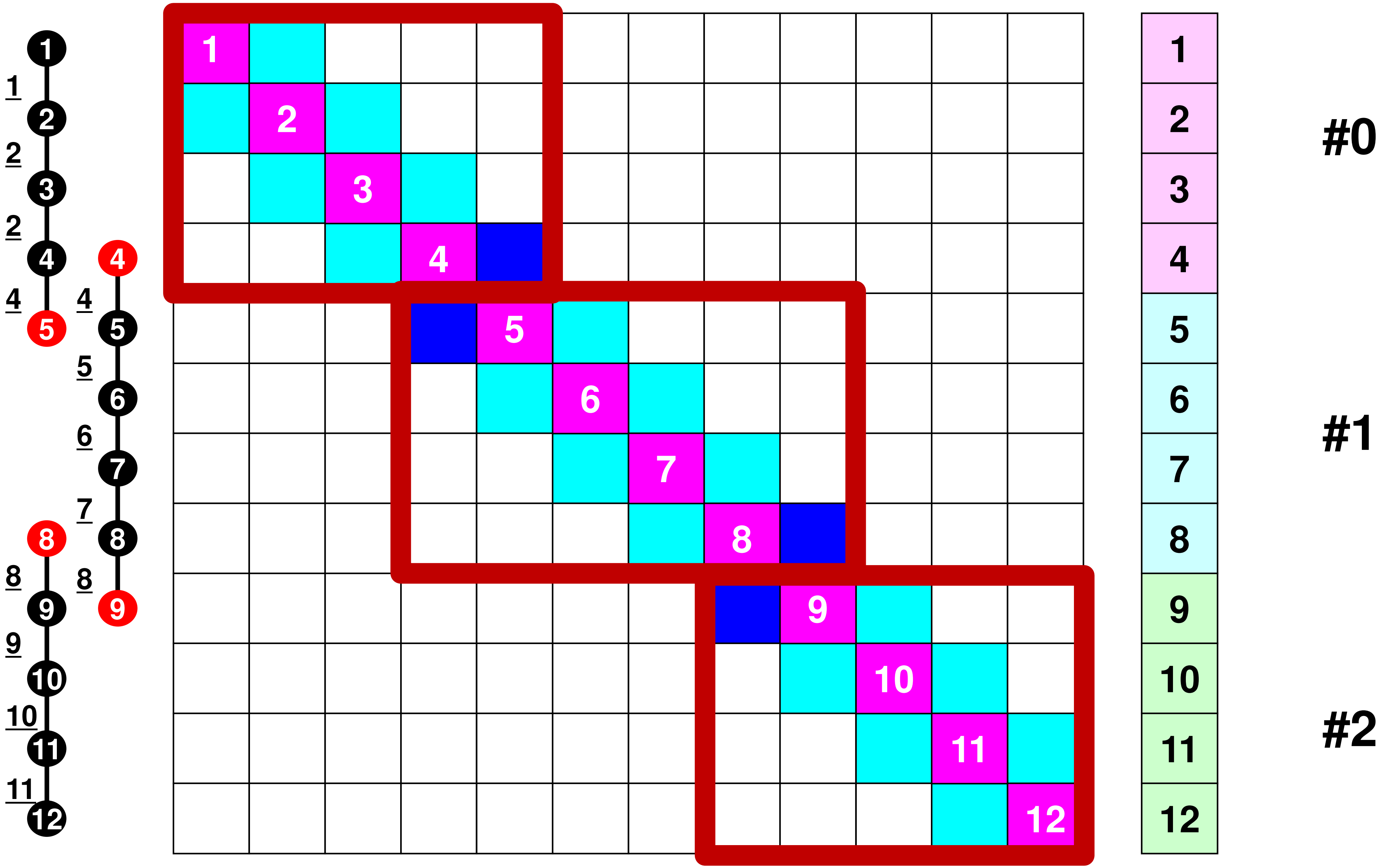
Boundary Nodes: Part of Internal Nodes, and External Nodes of Other Domains



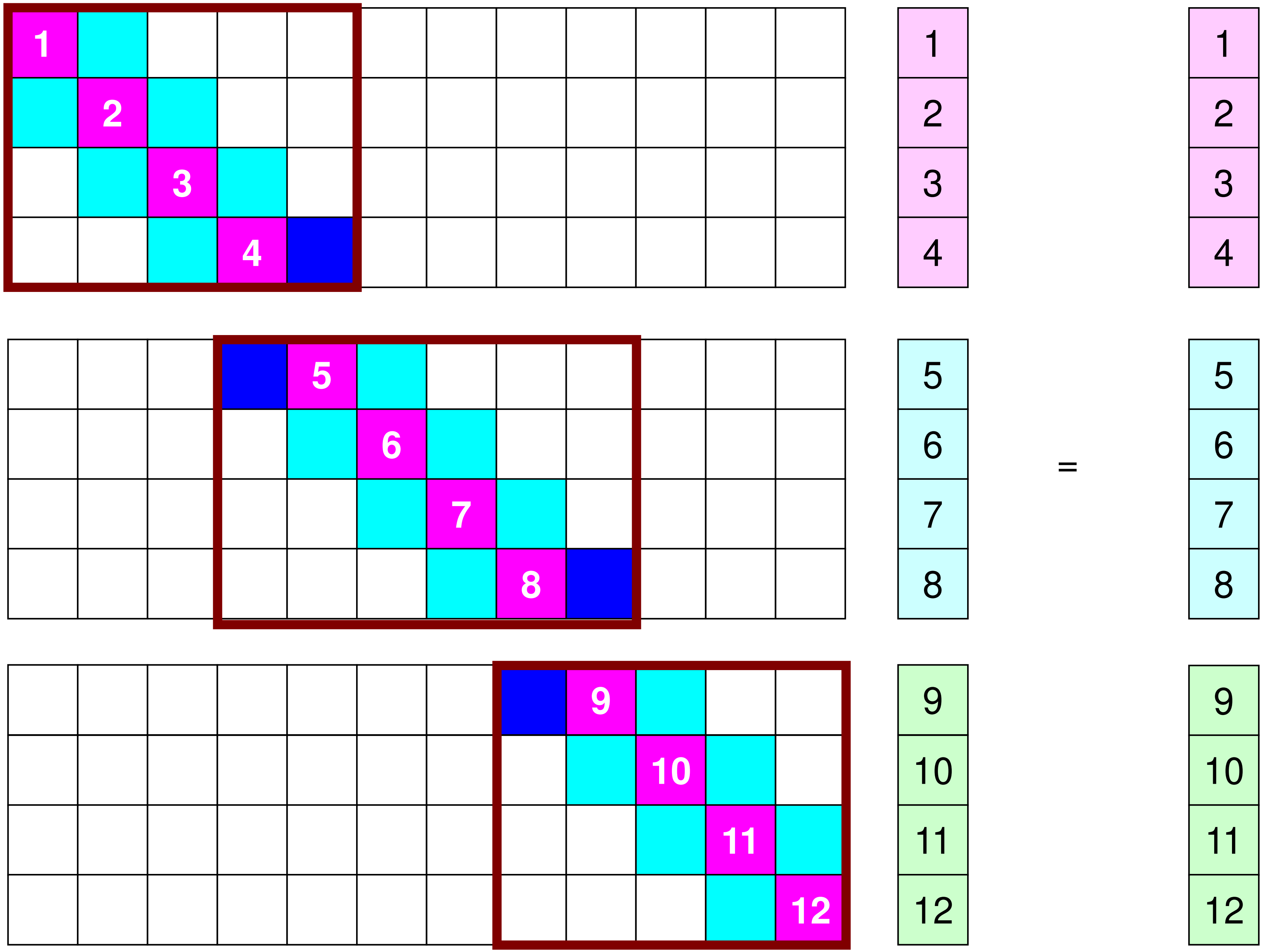
Mat-Vec Products: Local Op. Possible



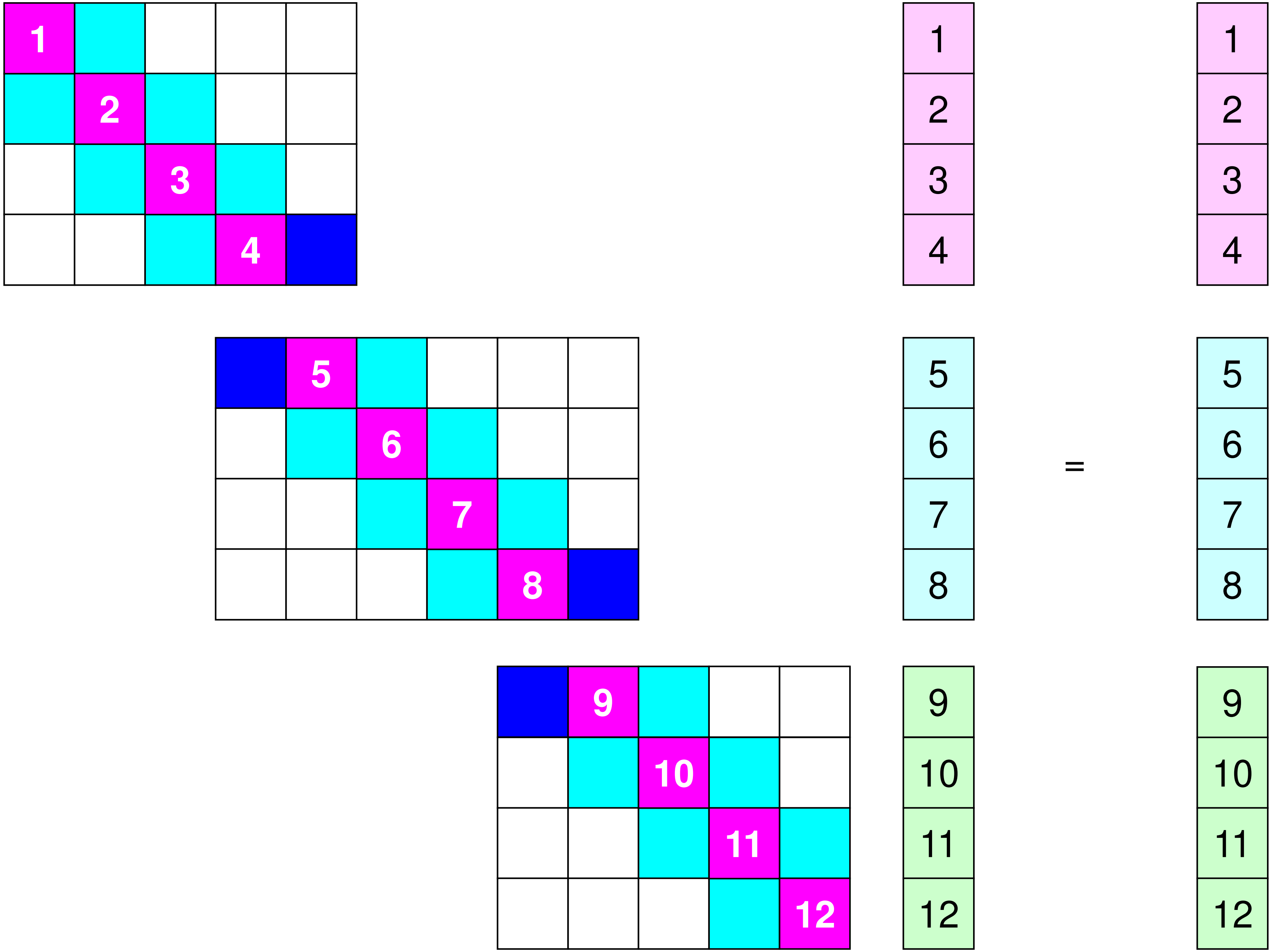
Because the matrix is sparse, the union of the local matrices forms the global matrix !



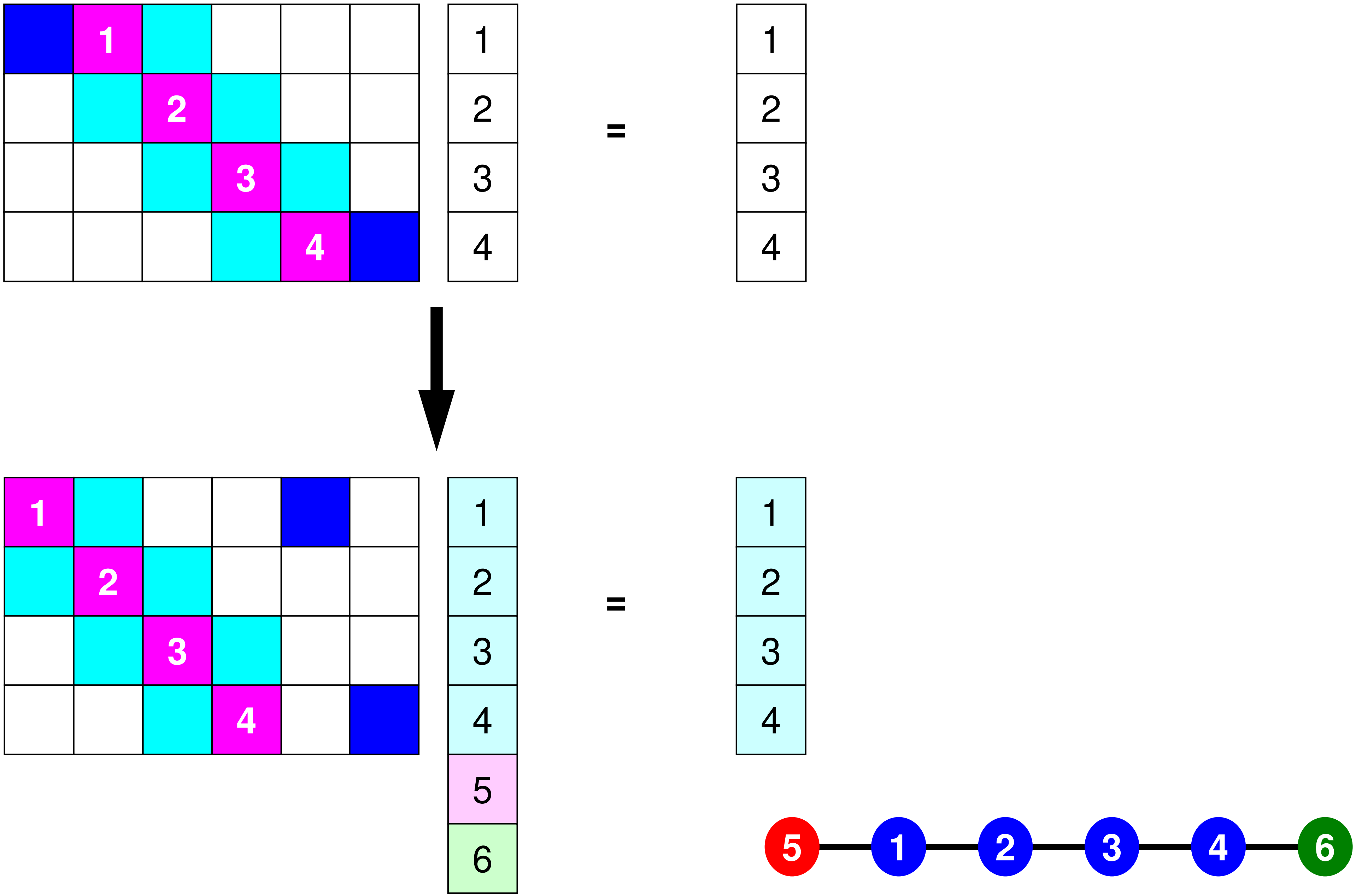
Mat-Vec Products: Local Op. Possible



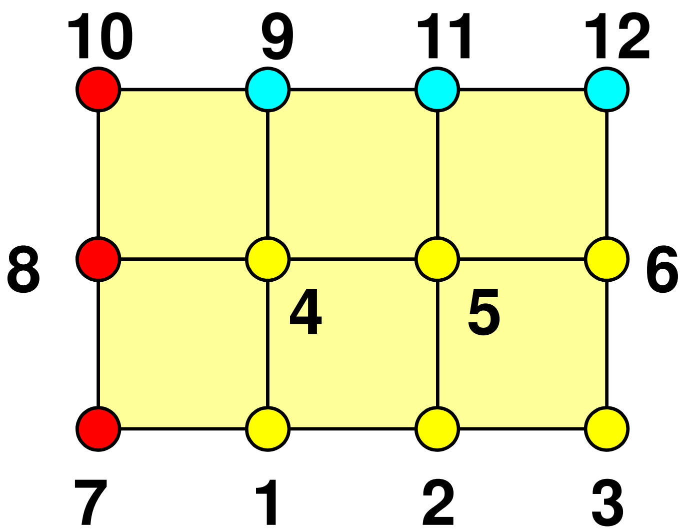
Mat-Vec Products: Local Op. Possible



Mat-Vec Products: Local Op. #1



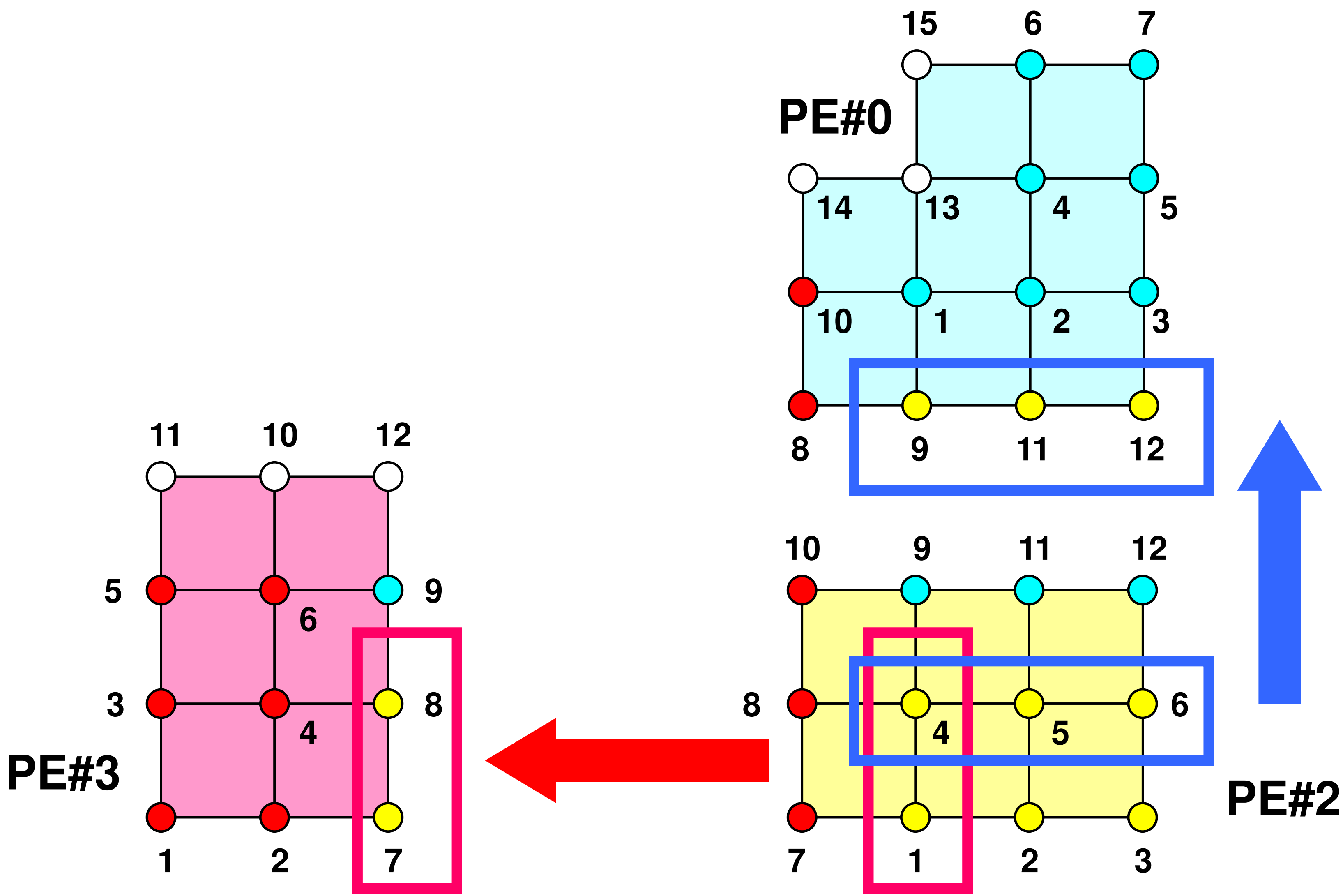
Description of Distributed Local Data



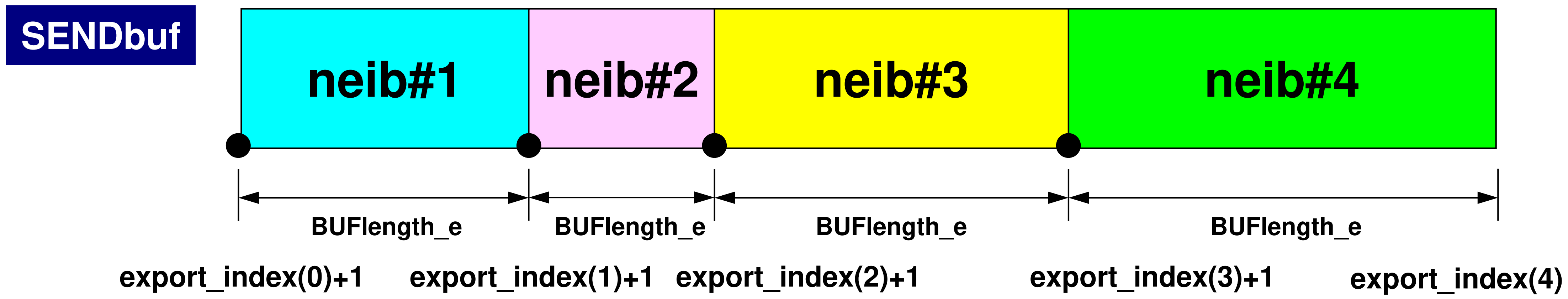
- Internal/External Points
  - Numbering: Starting from internal pts, then external pts after that
- Neighbors
  - Shares overlapped meshes
  - Number and ID of neighbors
- External Points
  - From where, how many, and which external points are received/imported ?
- Boundary Points
  - To where, how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

Boundary Nodes（境界点）：SEND

PE#2 : send information on “boundary nodes”



SEND: MPI_Isend/Irecv/Waitall

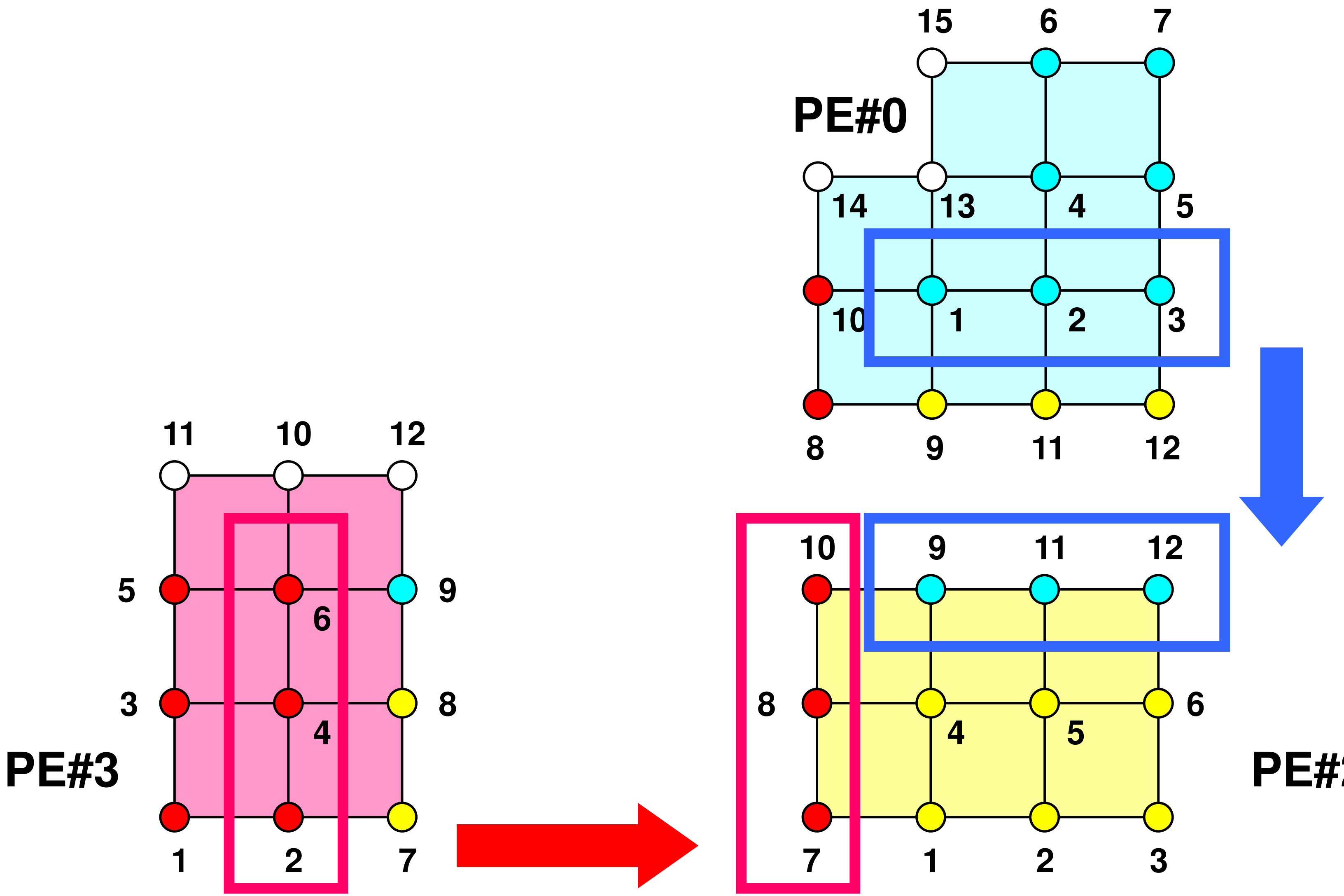



```
 do neib= 1, NEIBPETOT
   do k= export_index(neib-1)+1, export_index(neib)
     kk= export_item(k)
     SENDbuf(k)= VAL(kk)
   enddo
 enddo

 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

 call MPI_WAITALL (NEIBPETOT, request_send, stat_send, ierr)
```

Copied to sending buffers

External Nodes（外点）: RECEIVE

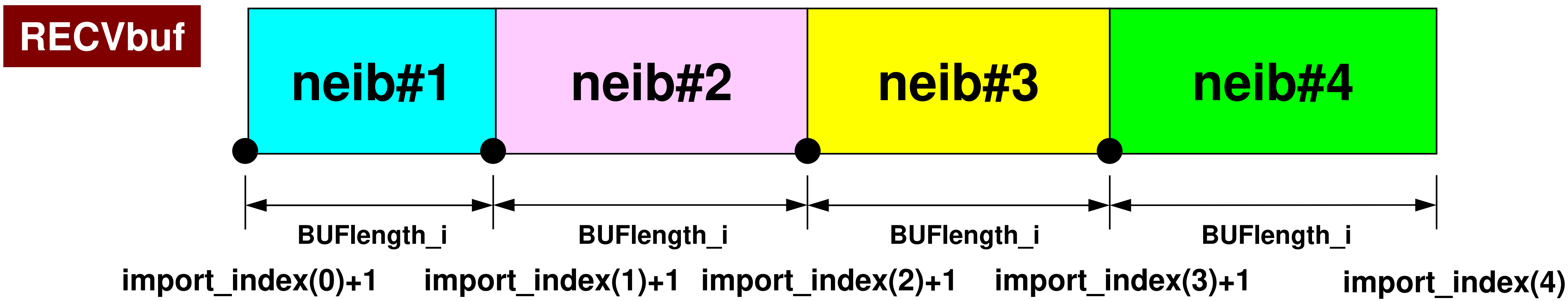
PE#2 : receive information for “external nodes”



RECV: MPI_Isend/Irecv/Waitall

```
 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

  call MPI_WAITALL (NEIBPETOT, request_recv, stat_recv, ierr)

  do neib= 1, NEIBPETOT
     do k= import_index(neib-1)+1, import_index(neib)
       kk= import_item(k)
       VAL(kk)= RECVbuf(k)
     enddo
  enddo
```

Copied from receiving buffer



- Overview
- Distributed Local Data
- **Program**
- Results

Program: 1d.f (1/11)

Variables

```
      program heat1Dp
      implicit REAL*8 (A-H,O-Z)
      include 'mpif.h'

      integer :: N, NPLU, ITERmax
      integer :: R, Z, P, Q, DD

      real(kind=8) :: dX, RESID, EPS
      real(kind=8) :: AREA, QV, COND
      real(kind=8), dimension(:), allocatable :: PHI, RHS
      real(kind=8), dimension(:  ), allocatable :: DIAG, AMAT
      real(kind=8), dimension(:,:), allocatable :: W

      real(kind=8), dimension(2,2) :: KMAT, EMAT

      integer, dimension(:), allocatable :: ICELNOD
      integer, dimension(:), allocatable :: INDEX, ITEM
      integer(kind=4) :: NEIBPETOT, BUFlength, PETOT
      integer(kind=4), dimension(2) :: NEIBPE

      integer(kind=4), dimension(0:2) :: import_index, export_index
      integer(kind=4), dimension(  2) :: import_item , export_item

      real(kind=8), dimension(2) :: SENDbuf, RECVbuf

      integer(kind=4), dimension(:,:), allocatable :: stat_send
      integer(kind=4), dimension(:,:), allocatable :: stat_recv
      integer(kind=4), dimension(:  ), allocatable :: request_send
      integer(kind=4), dimension(:  ), allocatable :: request_recv
```

Variable/Arrays (1/3)

Name	Type	Size	Definition
NE, Neg	I		# Element (Local, Global)
N, NP	I		# Node (Internal, Internal+External)
NPLU	I		# Non-Zero Off-Diag. Components
IterMax	I		MAX Iteration Number for CG
errno	I		ERROR flag
R, Z, Q, P, DD	I		Name of Vectors in CG
dX	R		Length of Each Element
Resid	R		Residual for CG
Eps	R		Convergence Criteria for CG
Area	R		Sectional Area of Element
QV	R		Heat Generation Rate/Volume/Time
COND	R		Thermal Conductivity

$\dot{Q}$

Variable/Arrays (2/3)

Name	Type	Size	Definition
X	R	NP	Location of Each Node
PHI	R	NP	Temperature of Each Node
Rhs	R	NP	RHS Vector
Diag	R	NP	Diagonal Components
W	R	(N,4)	Work Array for CG
Amat	R	NPLU	Off-Diagonal Components (Value)
Index	I	0:NP	Number of Non-Zero Off-Diagonals at Each ROW
Item	I	NPLU	Off-Diagonal Components (Corresponding Column ID)
Icelnod	I	2*NE	Node ID for Each Element
Kmat	R	(2,2)	Element Matrix [*k*]
Emat	R	(2,2)	Element Matrix

Variable/Arrays (3/3)

Name	Type	Size	Definition
PETOT	I		Total Number of MPI Processes
my_rank	I		Rank ID
NEIBPETOT	I		Total Number of Neighbors
NEIBPE	I	2	ID of Neighbors
import_index **export_index**	I	0:2	Size of Import/Export Arrays for Communication Table
import_item	I	2	Receiving Table (External Points)
export_item	I	2	Sending Table (Boundary Points)
RECVBuf	R	2	Receiving Buffer
SENDBuf	R	2	Sending Buffer

Program: 1d.f (2/11)

Control Data

```
!C
!C +-------+
!C | INIT. |
!C +-------+
!C===
!C
!C-- MPI init.

      call MPI_Init      (ierr)                                   Initialization
      call MPI_Comm_size (MPI_COMM_WORLD, PETOT, ierr )           Entire Process #: PETOT
      call MPI_Comm_rank (MPI_COMM_WORLD, my_rank, ierr )         Rank ID (0-PETot-1): my_rank

!C
!C-- CTRL data
      if (my_rank.eq.0) thenn
        open  (11, file='input.dat', status='unknown')
          read (11,*) NEg
          read (11,*) dX, QV, AREA, COND
          read (11,*) ITERmax
          read (11,*) EPS
        close (11)
      endif

      call MPI_Bcast (NEg    , 1, MPI_INTEGER, 0, MPI_COMM_WORLD, ierr)
      call MPI_Bcast (ITERmax, 1, MPI_INTEGER, 0, MPI_COMM_WORLD, ierr)
      call MPI_Bcast (dX     , 1, MPI_DOUBLE_PRECISION, 0, MPI_COMM_WORLD, ierr)
      call MPI_Bcast (QV     , 1, MPI_DOUBLE_PRECISION, 0, MPI_COMM_WORLD, ierr)
      call MPI_Bcast (AREA   , 1, MPI_DOUBLE_PRECISION, 0, MPI_COMM_WORLD, ierr)
      call MPI_Bcast (COND   , 1, MPI_DOUBLE_PRECISION, 0, MPI_COMM_WORLD, ierr)
      call MPI_Bcast (EPS    , 1, MPI_DOUBLE_PRECISION, 0, MPI_COMM_WORLD, ierr)
```

Program: 1d.f (2/11)

Control Data

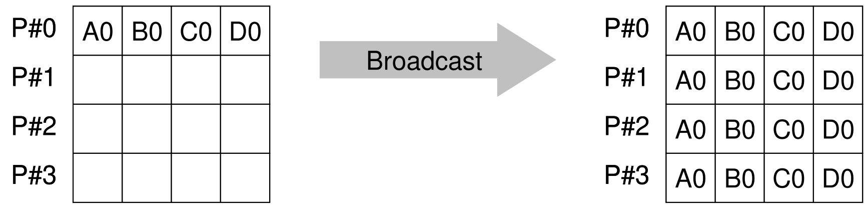
```
!C
!C +--------+
!C | INIT.  |
!C +--------+
!C===
!C
!C-- MPI init.

      call MPI_Init      (ierr)                                 Initialization
      call MPI_Comm_size (MPI_COMM_WORLD, PETOT, ierr )         Entire Process #: PETOT
      call MPI_Comm_rank (MPI_COMM_WORLD, my_rank, ierr )       Rank ID (0-PETot-1): my_rank

!C
!C-- CTRL data
      if (my_rank.eq.0) then                                    Reading control file if my_rank=0
        open  (11, file='input.dat', status='unknown')
         read (11,*) Neg                                        Neg: Global Number of Elements
         read (11,*) dX, QV, AREA, COND
         read (11,*) ITERmax
         read (11,*) EPS
        close (11)
      endif

      call MPI_Bcast (NEg    , 1, MPI_INTEGER, 0, MPI_COMM_WORLD, ierr)
      call MPI_Bcast (ITERmax, 1, MPI_INTEGER, 0, MPI_COMM_WORLD, ierr)
      call MPI_Bcast (dX     , 1, MPI_DOUBLE_PRECISION, 0, MPI_COMM_WORLD, ierr)
      call MPI_Bcast (QV     , 1, MPI_DOUBLE_PRECISION, 0, MPI_COMM_WORLD, ierr)
      call MPI_Bcast (AREA   , 1, MPI_DOUBLE_PRECISION, 0, MPI_COMM_WORLD, ierr)
      call MPI_Bcast (COND   , 1, MPI_DOUBLE_PRECISION, 0, MPI_COMM_WORLD, ierr)
      call MPI_Bcast (EPS    , 1, MPI_DOUBLE_PRECISION, 0, MPI_COMM_WORLD, ierr)
```

Program: 1d.f (2/11)

Control Data

```
!C
!C +-------+
!C | INIT. |
!C +-------+
!C===
!C
!C-- MPI init.

      call MPI_Init      (ierr)                              Initialization
      call MPI_Comm_size (MPI_COMM_WORLD, PETOT, ierr )      Entire Process #: PETOT
      call MPI_Comm_rank (MPI_COMM_WORLD, my_rank, ierr )    Rank ID (0-PETot-1): my_rank

!C
!C-- CTRL data
      if (my_rank.eq.0) then                                 Reading control file if my_rank=0
        open  (11, file='input.dat', status='unknown')
         read (11,*) Neg                                     Neg: Global Number of Elements
         read (11,*) dX, QV, AREA, COND
         read (11,*) ITERmax
         read (11,*) EPS
        close (11)
      endif

      call MPI_Bcast (NEg    , 1, MPI_INTEGER, 0, MPI_COMM_WORLD, ierr) Parameters are sent to each proces
      call MPI_Bcast (ITERmax, 1, MPI_INTEGER, 0, MPI_COMM_WORLD, ierr) from Process #0.
      call MPI_Bcast (dX     , 1, MPI_DOUBLE_PRECISION, 0, MPI_COMM_WORLD, ierr)
      call MPI_Bcast (QV     , 1, MPI_DOUBLE_PRECISION, 0, MPI_COMM_WORLD, ierr)
      call MPI_Bcast (AREA   , 1, MPI_DOUBLE_PRECISION, 0, MPI_COMM_WORLD, ierr)
      call MPI_Bcast (COND   , 1, MPI_DOUBLE_PRECISION, 0, MPI_COMM_WORLD, ierr)
      call MPI_Bcast (EPS    , 1, MPI_DOUBLE_PRECISION, 0, MPI_COMM_WORLD, ierr)
```

MPI_BCAST



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

- **`call MPI_BCAST (buffer,count,datatype,root,comm,ierr)`**
  - **buffer** `choice` `I/O` starting address of buffer
    type is defined by "**datatype**"
  - **count** `I` `I` number of elements in send/recv buffer
  - **datatype** `I` `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** `I` `I` rank of root process
  - **comm** `I` `I` communicator
  - **ierr** `I` `O` completion code

Program: 1d.f (3/11)

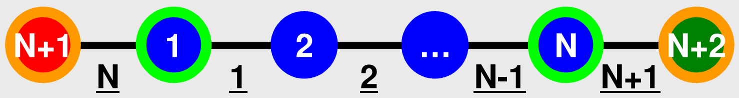
Distributed Local Mesh

```
!C
!C-- Local Mesh Size

      Ng= NEg + 1                            Global Number of Nodes
      N = Ng / PETOT                         Local Number of Nodes

      nr = Ng - N*PETOT                      mod(Ng,PETOT) .ne. 0
      if (my_rank.lt.nr) N= N+1

      NE= N - 1 + 2
      NP= N + 2

      if (my_rank.eq.0) NE= N - 1 + 1
      if (my_rank.eq.0) NP= N + 1

      if (my_rank.eq.PETOT-1) NE= N - 1 + 1
      if (my_rank.eq.PETOT-1) NP= N + 1

      if (PETOT.eq.1) NE= N-1
      if (PETOT.eq.1) NP= N

!C
!C- ARRAYs

      allocate (PHI(NP), DIAG(NP), AMAT(2*NP-2), RHS(NP))
      allocate (ICELNOD(2*NE))
      allocate (INDEX(0:NP), ITEM(2*NP-2), W(NP,4))
       PHI= 0.d0
      AMAT= 0.d0
      DIAG= 0.d0
       RHS= 0.d0
```

Program: 1d.f (3/11)

Distributed Local Mesh, Uniform Elements

```
!C
!C-- Local Mesh Size

      Ng= NEg + 1                        Global Number of Nodes
      N = Ng / PETOT                     Local Number of Nodes

      nr = Ng - N*PETOT                  mod(Ng,PETOT) .ne. 0
      if (my_rank.lt.nr) N= N+1

      NE= N - 1 + 2                      Number of Elements (Local)
      NP= N + 2                          Total Number of Nodes (Local) (Internal + External Nodes)

      if (my_rank.eq.0) NE= N - 1 + 1
      if (my_rank.eq.0) NP= N + 1

      if (my_rank.eq.PETOT-1) NE= N - 1 + 1
      if (my_rank.eq.PETOT-1) NP= N + 1

      if (PETOT.eq.1) NE= N-1
      if (PETOT.eq.1) NP= N

!C
!C- ARRAYs

      allocate (PHI(NP), DIAG(NP), AMAT(2*NP-2), RHS(NP))
      allocate (ICELNOD(2*NE))
      allocate (INDEX(0:NP), ITEM(2*NP-2), W(NP,4))
       PHI= 0.d0
      AMAT= 0.d0
      DIAG= 0.d0
       RHS= 0.d0
```

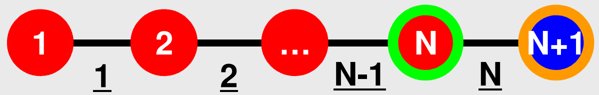


Others (General):
N+2 nodes
N+1 elements

Program: 1d.f (3/11)

Distributed Local Mesh, Uniform Elements

```
!C
!C-- Local Mesh Size

      Ng= NEg + 1                     Global Number of Nodes
      N = Ng / PETOT                  Local Number of Nodes

      nr = Ng - N*PETOT               mod(Ng,PETOT) .ne. 0
      if (my_rank.lt.nr) N= N+1

      NE= N - 1 + 2                   Number of Elements (Local)
      NP= N + 2                       Total Number of Nodes (Local) (Internal + External Nodes)

      if (my_rank.eq.0) NE= N - 1 + 1
      if (my_rank.eq.0) NP= N + 1

      if (my_rank.eq.PETOT-1) NE= N - 1 + 1
      if (my_rank.eq.PETOT-1) NP= N + 1

      if (PETOT.eq.1) NE= N-1
      if (PETOT.eq.1) NP= N

!C
!C- ARRAYs

      allocate (PHI(NP), DIAG(NP), AMAT(2*NP-2), RHS(NP))
      allocate (ICELNOD(2*NE))
      allocate (INDEX(0:NP), ITEM(2*NP-2), W(NP,4))
       PHI= 0.d0
      AMAT= 0.d0
      DIAG= 0.d0
       RHS= 0.d0
```

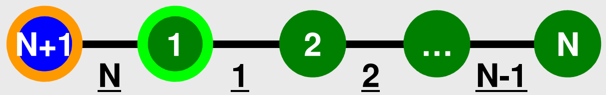


#0:
N+1 nodes
N elements

Program: 1d.f (3/11)

Distributed Local Mesh, Uniform Elem

```
!C
!C-- Local Mesh Size

      Ng= NEg + 1                     Global Number of Nodes
      N = Ng / PETOT                  Local Number of Nodes

      nr = Ng - N*PETOT               mod(Ng,PETOT) .ne. 0
      if (my_rank.lt.nr) N= N+1

      NE= N - 1 + 2                   Number of Elements (Local)
      NP= N + 2                       Total Number of Nodes (Local) (Internal + External Nodes)

      if (my_rank.eq.0) NE= N - 1 + 1
      if (my_rank.eq.0) NP= N + 1

      if (my_rank.eq.PETOT-1) NE= N - 1 + 1
      if (my_rank.eq.PETOT-1) NP= N + 1

      if (PETOT.eq.1) NE= N-1
      if (PETOT.eq.1) NP= N

!C
!C- ARRAYs

      allocate (PHI(NP), DIAG(NP), AMAT(2*NP-2), RHS(NP))
      allocate (ICELNOD(2*NE))
      allocate (INDEX(0:NP), ITEM(2*NP-2), W(NP,4))
       PHI= 0.d0
      AMAT= 0.d0
      DIAG= 0.d0
       RHS= 0.d0
```



#PETot-1:
N+1 nodes
N elements

Program: 1d.f (3/11)

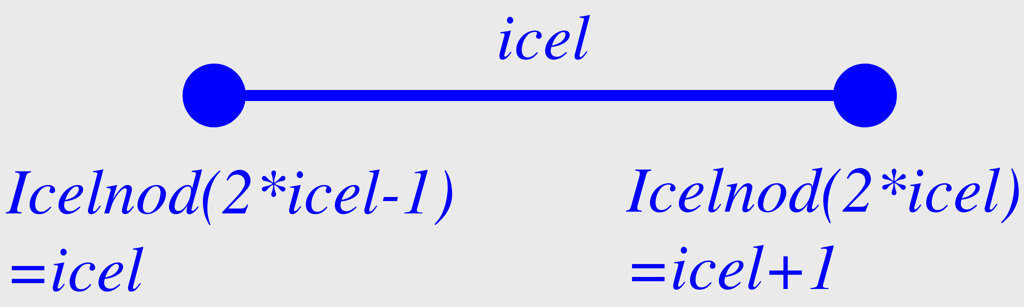
Distributed Local Mesh, Uniform Elements

```
!C
!C-- Local Mesh Size

      Ng= NEg + 1                         Global Number of Nodes
      N = Ng / PETOT                      Local Number of Nodes

      nr = Ng - N*PETOT                   mod(Ng,PETOT) .ne. 0
      if (my_rank.lt.nr) N= N+1

      NE= N - 1 + 2                       Number of Elements (Local)
      NP= N + 2                           Total Number of Nodes (Local) (Internal + External Nodes)

      if (my_rank.eq.0) NE= N - 1 + 1
      if (my_rank.eq.0) NP= N + 1

      if (my_rank.eq.PETOT-1) NE= N - 1 + 1
      if (my_rank.eq.PETOT-1) NP= N + 1

      if (PETOT.eq.1) NE= N-1
      if (PETOT.eq.1) NP= N

!C
!C- ARRAYs

      allocate (PHI(NP), DIAG(NP), AMAT(2*NP-2), RHS(NP))     Size of arrays is “NP”, not “N”
      allocate (ICELNOD(2*NE))
      allocate (INDEX(0:NP), ITEM(2*NP-2), W(NP,4))
       PHI= 0.d0
      AMAT= 0.d0
      DIAG= 0.d0
       RHS= 0.d0
```

Program: 1d.f (4/11)

Initialization of Arrays, Elements-Nodes

```
      do icel= 1, NE
        ICELNOD(2*icel-1)= icel
        ICELNOD(2*icel  )= icel + 1
      enddo

      if (PETOT.gt.1) then

      if (my_rank.eq.0) then
        icel= NE
        ICELNOD(2*icel-1)= N
        ICELNOD(2*icel  )= N + 1


       else if (my_rank.eq.PETOT-1) then
        icel= NE
        ICELNOD(2*icel-1)= N + 1
        ICELNOD(2*icel  )= 1


       else
        icel= NE - 1
        ICELNOD(2*icel-1)= N + 1
        ICELNOD(2*icel  )= 1
        icel= NE
        ICELNOD(2*icel-1)= N
        ICELNOD(2*icel  )= N + 2

      endif
      endif
```

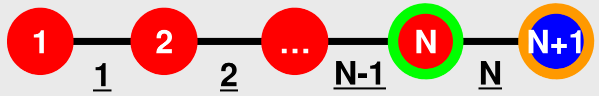


Program: 1d.f (4/11)

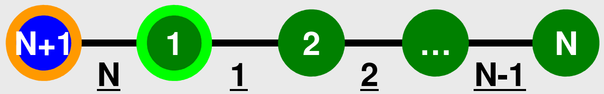
Initialization of Arrays, Elements-Nodes

```
do icel= 1, NE
  ICELNOD(2*icel-1)= icel
  ICELNOD(2*icel  )= icel + 1
enddo

if (PETOT.gt.1) then

if (my_rank.eq.0) then
  icel= NE
  ICELNOD(2*icel-1)= N
  ICELNOD(2*icel  )= N + 1

 else if (my_rank.eq.PETOT-1) then
  icel= NE
  ICELNOD(2*icel-1)= N + 1
  ICELNOD(2*icel  )= 1

 else
  icel= NE - 1
  ICELNOD(2*icel-1)= N + 1
  ICELNOD(2*icel  )= 1
  icel= NE
  ICELNOD(2*icel-1)= N
  ICELNOD(2*icel  )= N + 2

endif
endif
```

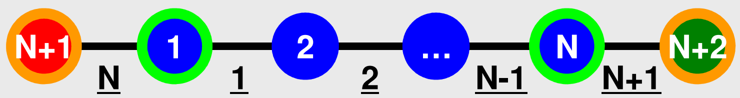
e.g. Element-1 includes node-1 and node-2



#0:
N+1 nodes
N elements



#PETot-1:
N+1 nodes
N elements

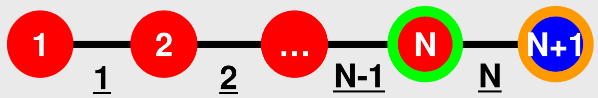


Others (General):
N+2 nodes
N+1 elements

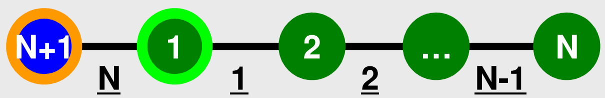
Program: 1d.f (5/11)

"Index"

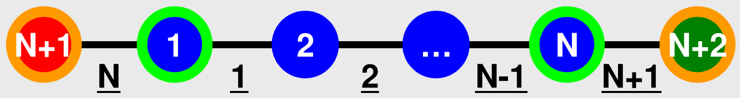
```
 KMAT(1,1)= +1.d0
        KMAT(1,2)= -1.d0
        KMAT(2,1)= -1.d0
        KMAT(2,2)= +1.d0
!C===

!C
!C +--------------+
!C | CONNECTIVITY |
!C +--------------+
!C===
        INDEX    = 2

        INDEX(0)= 0

        INDEX(N+1)= 1
        INDEX(NP )= 1

        if (my_rank.eq.0)         INDEX(1)= 1
        if (my_rank.eq.PETOT-1)  INDEX(N)= 1

        do i= 1, NP
          INDEX(i)= INDEX(i) + INDEX(i-1)
        enddo

        NPLU= INDEX(NP)
        ITEM= 0
```



#0:
N+1 nodes
N elements



#PETot-1:
N+1 nodes
N elements

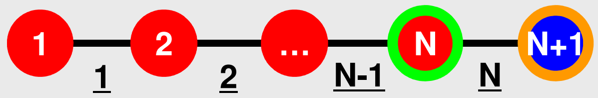


Others (General):
N+2 nodes
N+1 elements

Program: 1d.f (6/11)

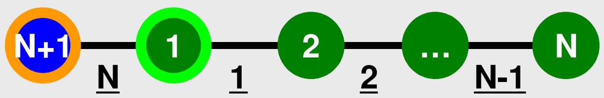
"Item"

```
do i= 1, N
  jS= INDEX(i-1)
  if (my_rank.eq.0.and.i.eq.1) then
    ITEM(jS+1)= i+1
   else if (my_rank.eq.PETOT-1.and.i.eq.N) then
    ITEM(jS+1)= i-1
   else
    ITEM(jS+1)= i-1
    ITEM(jS+2)= i+1
    if (i.eq.1) ITEM(jS+1)= N + 1
    if (i.eq.N) ITEM(jS+2)= N + 2
    if (my_rank.eq.0.and.i.eq.N) ITEM(jS+2)= N + 1
  endif
enddo
```

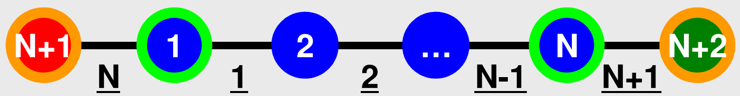


#0:
N+1 nodes
N elements

```
i = N + 1
jS= INDEX(i-1)
if (my_rank.eq.0) then
  ITEM(jS+1)= N
 else
  ITEM(jS+1)= 1
endif

i = N + 2
if (my_rank.ne.0.and.my_rank.ne.PETOT-1) then
  jS= INDEX(i-1)
  ITEM(jS+1)= N
endif
```



#PETot-1:
N+1 nodes
N elements

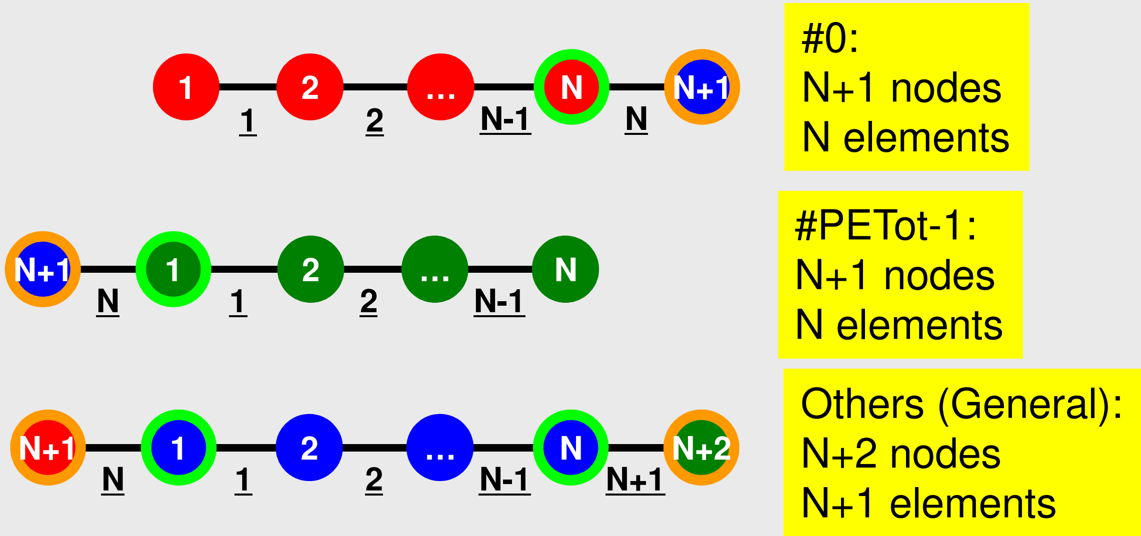


Others (General):
N+2 nodes
N+1 elements

Program: 1d.f (7/11)

Communication Tables

```
!C
!C-- COMMUNICATION
      NEIBPETOT= 2
      if (my_rank.eq.0       ) NEIBPETOT= 1
      if (my_rank.eq.PETOT-1) NEIBPETOT= 1
      if (PETOT.eq.1)          NEIBPETOT= 0

      NEIBPE(1)= my_rank - 1
      NEIBPE(2)= my_rank + 1

      if (my_rank.eq.0       ) NEIBPE(1)= my_rank + 1
      if (my_rank.eq.PETOT-1) NEIBPE(1)= my_rank - 1

      BUFlength= 1

      import_index(1)= 1
      import_index(2)= 2
      import_item (1)= N+1
      import_item (2)= N+2

      export_index(1)= 1
      export_index(2)= 2
      export_item (1)= 1
      export_item (2)= N

      if (my_rank.eq.0) then
        import_item (1)= N+1
        export_item (1)= N
      endif
!C
!C-- INIT. arrays for MPI_Waitall
      allocate (stat_send(MPI_STATUS_SIZE,NEIBPETOT), stat_recv(MPI_STATUS_SIZE,NEIBPETOT))
      allocate (request_send(NEIBPETOT), request_recv(NEIBPETOT))
```



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 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.

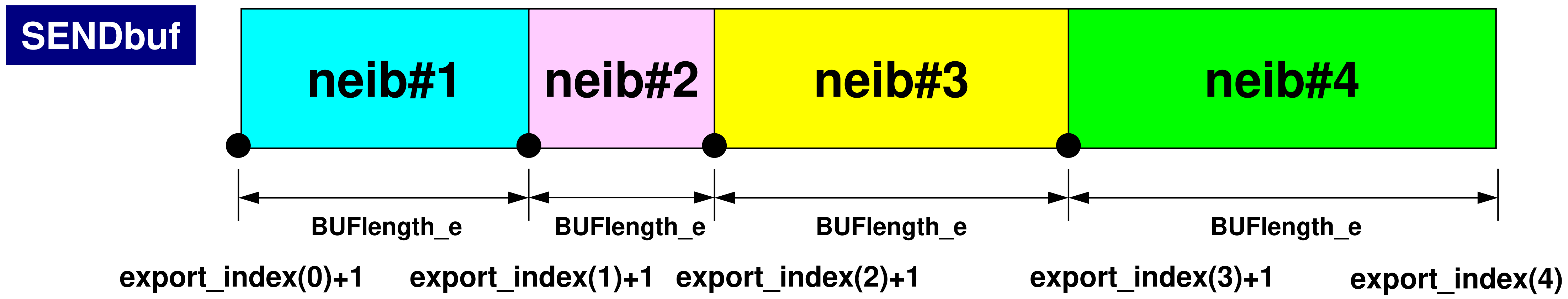
- **call MPI_WAITALL (count,request,status,ierr)**

– **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

Generalized Comm. Table: Send

- Neighbors
  - NEIBPETOT, NEIBPE(neib)
- Message size for each neighbor
  - export_index(neib), neib= 0, NEIBPETOT
- ID of **boundary** points
  - export_item(k), k= 1, export_index(NEIBPETOT)
- Messages to each neighbor
  - SENDbuf(k), k= 1, export_index(NEIBPETOT)

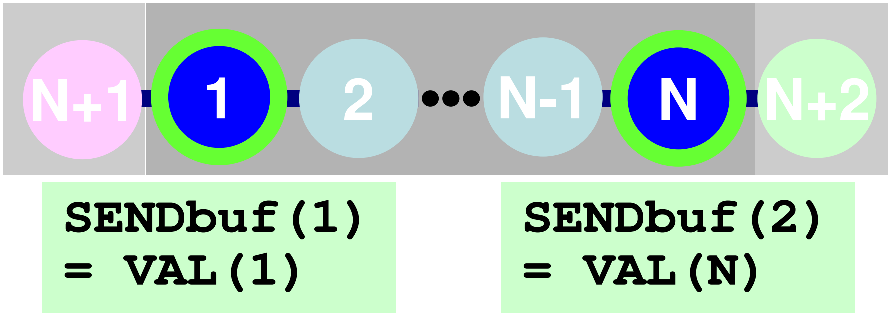
SEND: MPI_Isend/Irecv/Waitall




```
 do neib= 1, NEIBPETOT
   do k= export_index(neib-1)+1, export_index(neib)
     kk= export_item(k)
     SENDbuf(k)= VAL(kk)
   enddo
 enddo

 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

 call MPI_WAITALL (NEIBPETOT, request_send, stat_send, ierr)
```

Copied to sending buffers

SEND/Export: 1D Problem



- Neighbors
  - NEIBPETOT, NEIBPE(neib)
    - NEIBPETOT=2, NEIB(1)= my_rank-1, NEIB(2)= my_rank+1
- Message size for each neighbor
  - export_index(neib), neib= 0, NEIBPETOT
    - export_index(0)=0, export_index(1)= 1, export_index(2)= 2
- ID of **boundary** points
  - export_item(k), k= 1, export_index(NEIBPETOT)
    - export_item(1)= 1, export_item(2)= N
- Messages to each neighbor
  - SENDbuf(k), k= 1, export_index(NEIBPETOT)
    - SENDbuf(1)= BUF(1), SENDbuf(2)= BUF(N)

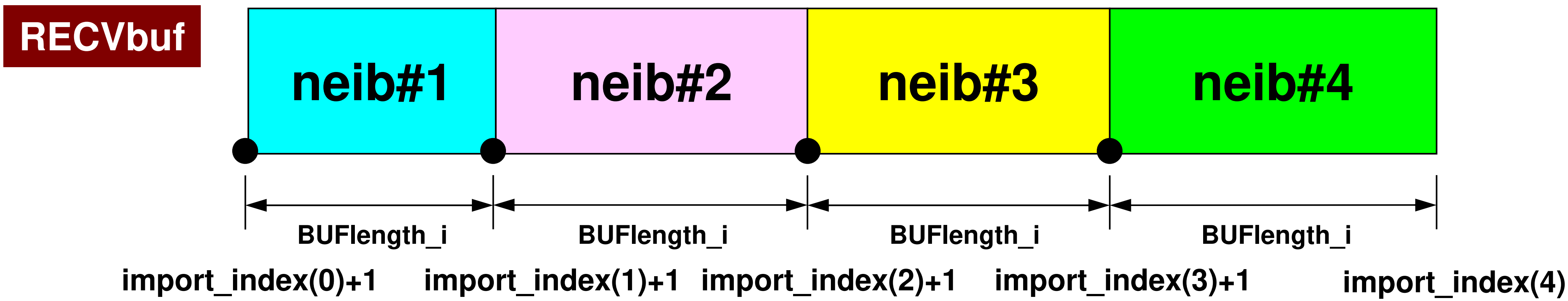
Generalized Comm. Table: Receive

- Neighbors
  - NEIBPETOT, NEIBPE(neib)
- Message size for each neighbor
  - import_index(neib), neib= 0, NEIBPETOT
- ID of **<u>external</u>** points
  - import_item(k), k= 1, import_index(NEIBPETOT)
- Messages from each neighbor
  - RECVbuf(k), k= 1, import_index(NEIBPETOT)

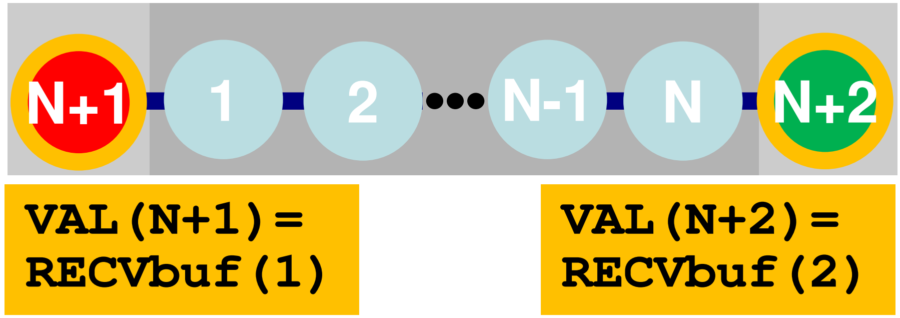
RECV: MPI_Isend/Irecv/Waitall

```
 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

  call MPI_WAITALL (NEIBPETOT, request_recv, stat_recv, ierr)

  do neib= 1, NEIBPETOT
     do k= import_index(neib-1)+1, import_index(neib)
       kk= import_item(k)
       VAL(kk)= RECVbuf(k)
     enddo
  enddo
```

Copied from receiving buffer

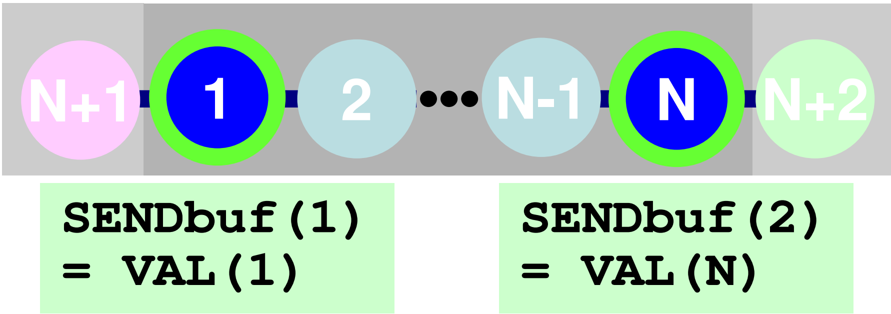


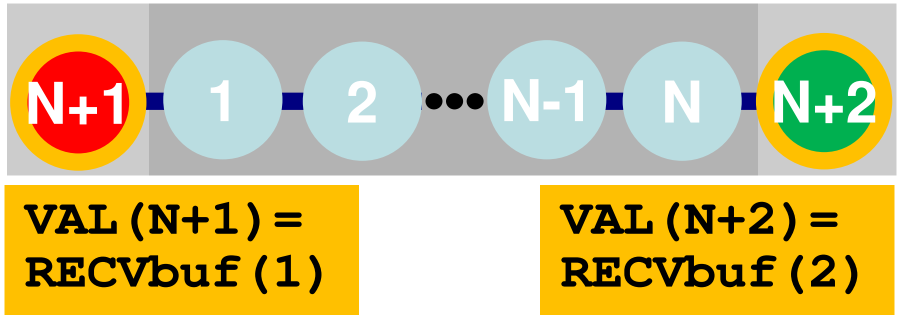
RECV/Import: 1D Problem



- Neighbors
  - NEIBPETOT, NEIBPE(neib)
    - NEIBPETOT=2, NEIB(1)= my_rank-1, NEIB(2)= my_rank+1
- Message size for each neighbor
  - import_index(neib), neib= 0, NEIBPETOT
    - import_index(0)=0, import_index(1)= 1, import_index(2)= 2
- ID of **external** points
  - import_item(k), k= 1, import_index(NEIBPETOT)
    - import_item(1)= N+1, import_item(2)= N+2
- Messages from each neighbor
  - RECVbuf(k), k= 1, import_index(NEIBPETOT)
    - BUF(N+1)=RECVbuf(1), BUF(N+2)=RECVbuf(2)

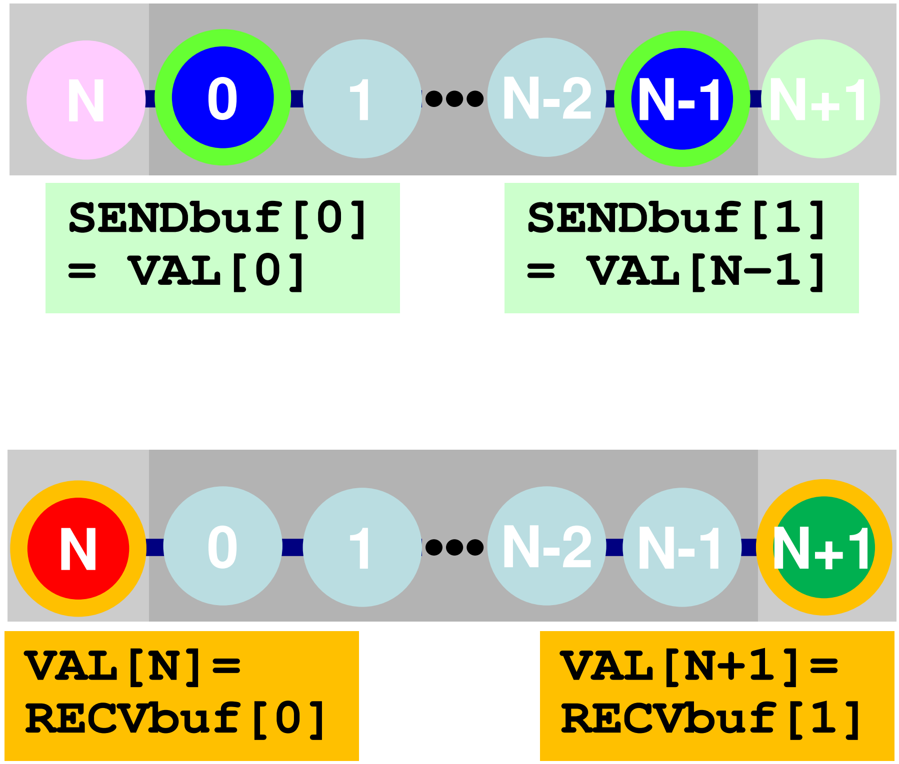
Generalized Comm. Table: Fortran






```
NEIBPETOT= 2
NEIBPE(1)= my_rank - 1
NEIBPE(2)= my_rank + 1

import_index(1)= 1
import_index(2)= 2
import_item (1)= N+1
import_item (2)= N+2

export_index(1)= 1
export_index(2)= 2
export_item (1)= 1
export_item (2)= N

if (my_rank.eq.0) then
  import_item (1)= N+1
  export_item (1)= N
  NEIBPE(1)= my_rank+1
endif
```

Generalized Comm. Table: C

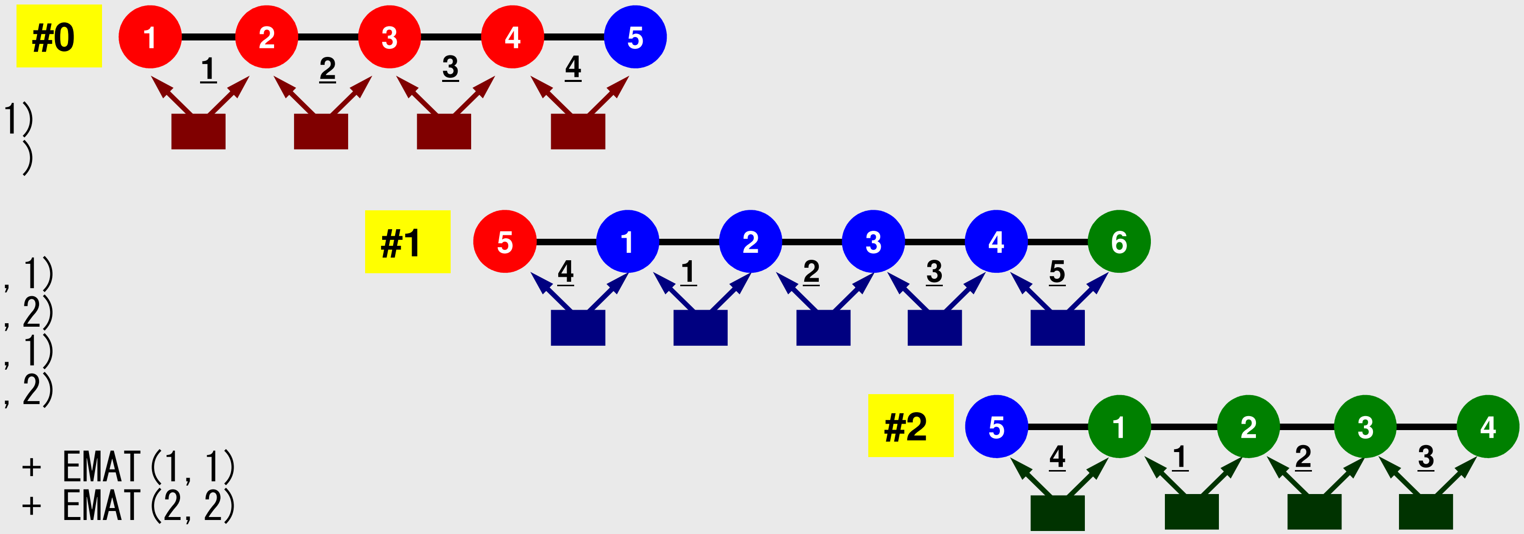



```
NEIBPETOT= 2
NEIBPE[0]= my_rank - 1
NEIBPE[1]= my_rank + 1

import_index[1]= 1
import_index[2]= 2
import_item [0]= N
import_item [1]= N+1

export_index[1]= 1
export_index[2]= 2
export_item [0]= 0
export_item [1]= N-1

if (my_rank.eq.0) then
  import_item [0]= N
  export_item [0]= N-1
  NEIBPE[0]= my_rank+1
endif
```

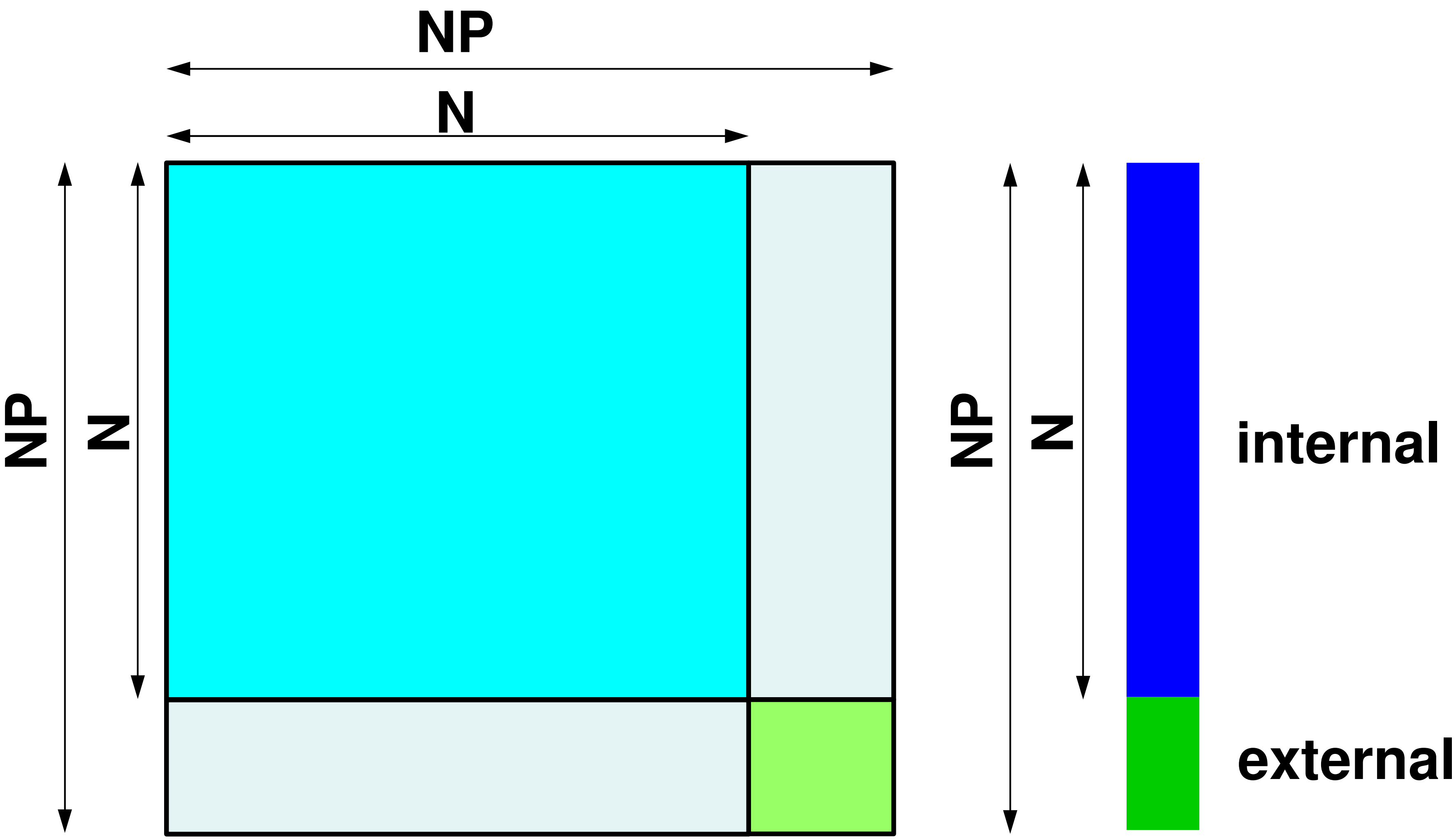
Program: 1d.f (8/11)

Matrix Assembling, NO changes from 1-CPU co

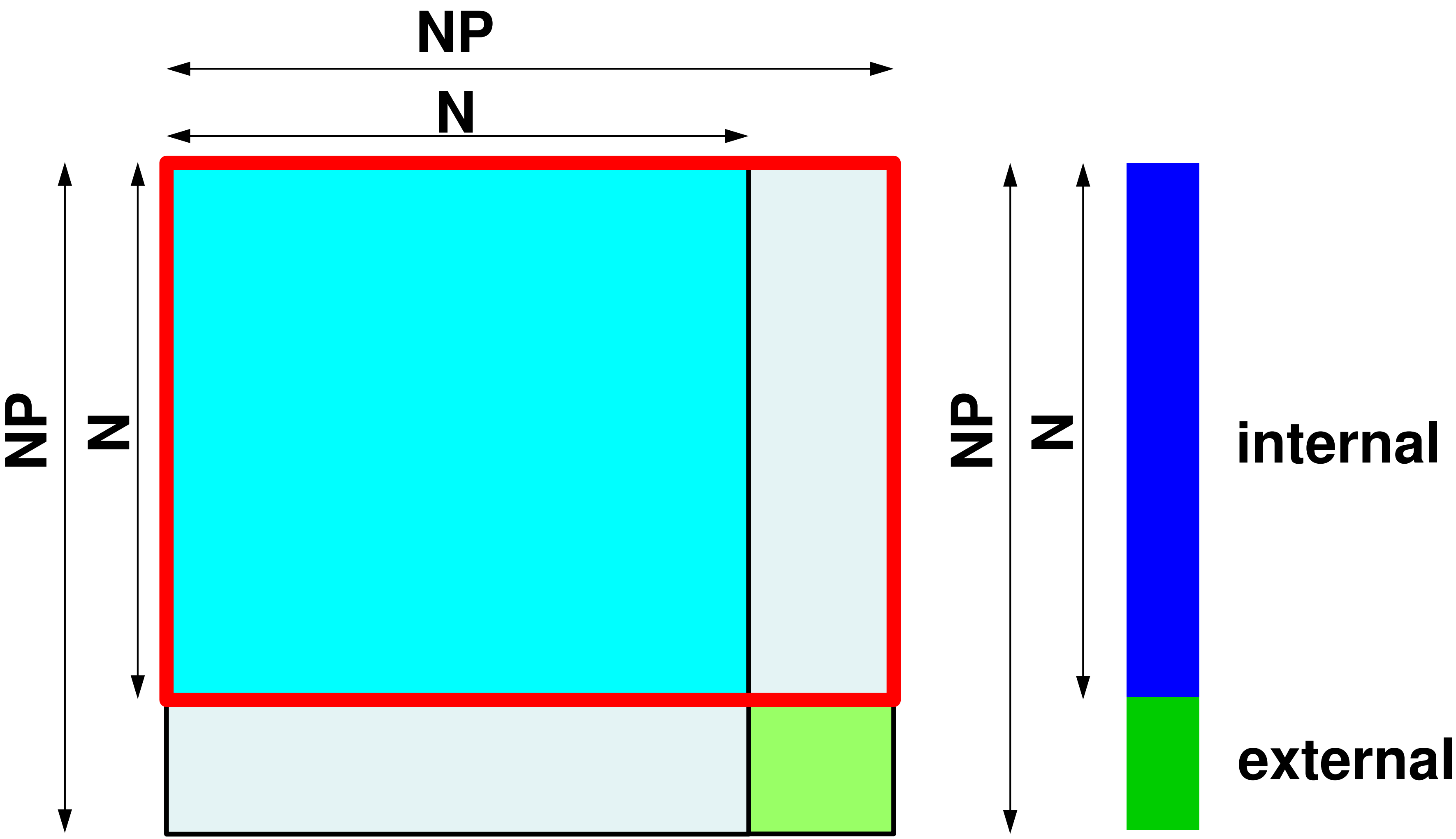
```
!C
!C +-----------------+
!C | MATRIX ASSEMBLE |
!C +-----------------+
!C===
      do icel= 1, NE
        in1= ICELNOD(2*icel-1)
        in2= ICELNOD(2*icel  )
        DL = dX
        cK= AREA*COND/DL
        EMAT(1,1)= Ck*KMAT(1,1)
        EMAT(1,2)= Ck*KMAT(1,2)
        EMAT(2,1)= Ck*KMAT(2,1)
        EMAT(2,2)= Ck*KMAT(2,2)

        DIAG(in1)= DIAG(in1) + EMAT(1,1)
        DIAG(in2)= DIAG(in2) + EMAT(2,2)

        if (my_rank.eq.0.and.icel.eq.1) then
          k1= INDEX(in1-1) + 1
         else
          k1= INDEX(in1-1) + 2
        endif
        k2= INDEX(in2-1) + 1

        AMAT(k1)= AMAT(k1) + EMAT(1,2)
        AMAT(k2)= AMAT(k2) + EMAT(2,1)

        QN= 0.50d0*QV*AREA*DL
        RHS(in1)= RHS(in1) + QN
        RHS(in2)= RHS(in2) + QN
      enddo
!C===
```



Local Matrix



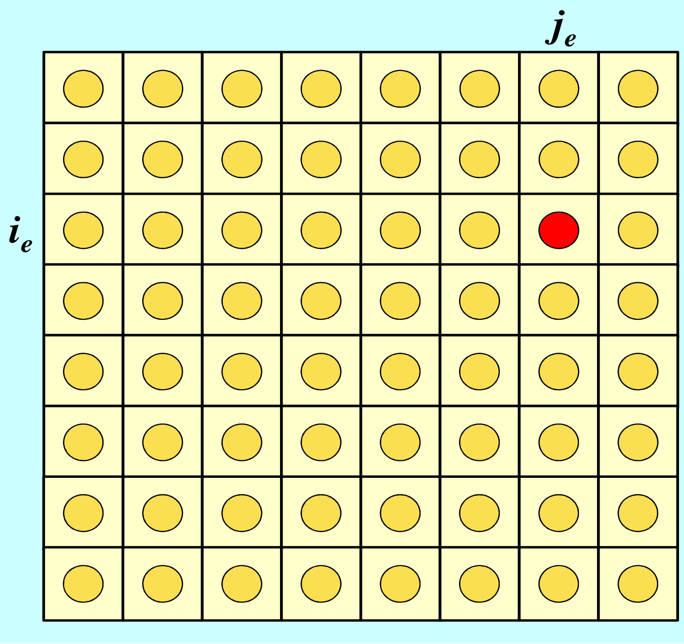
We really need these parts:



MAT_ASS_MAIN: Overview

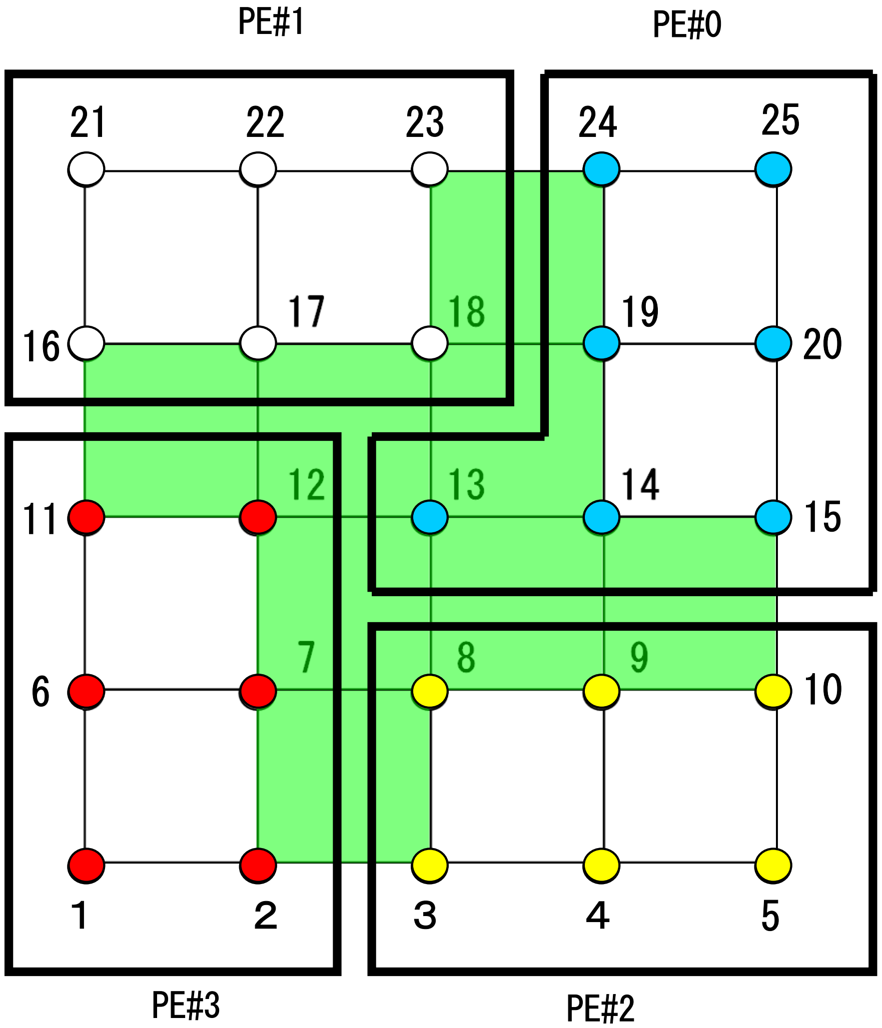
```
do kpn= 1, 2          Gaussian Quad. points in ζ-direction
  do jpn= 1, 2        Gaussian Quad. points in η-direction
    do ipn= 1, 2      Gaussian Quad. Pointe in ξ-direction
      Define Shape Function at Gaussian Quad. Points (8-points)
      Its derivative on natural/local coordinate is also defined.
    enddo
  enddo
enddo
```

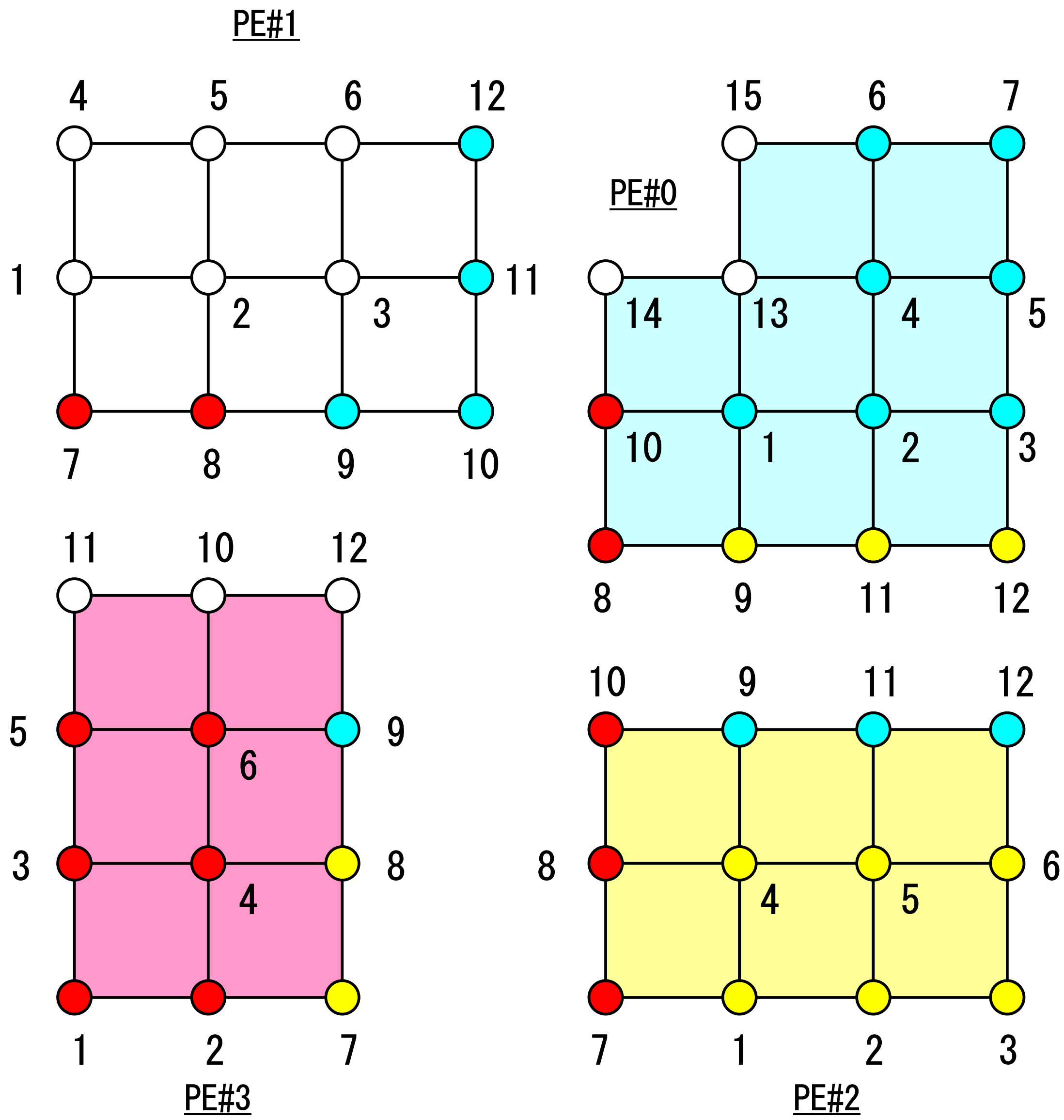
```
do icel= 1, ICELTOT   Loop for Element
  Jacobian and derivative on global coordinate of shape functions at
  Gaussian Quad. Points are defined according to coordinates of 8 nodes. (JACOBI)

  do ie= 1, 8           Local Node ID
    do je= 1, 8         Local Node ID
      Global Node ID: ip, jp
      Address of A_ip,jp in "item" : kk

      do kpn= 1, 2        Gaussian Quad. points in ζ-direction
        do jpn= 1, 2      Gaussian Quad. points in η-direction
          do ipn= 1, 2    Gaussian Quad. points in ξ-direction
            integration on each element
            coefficients of element matrices
            accumulation to global matrix
          enddo
        enddo
      enddo
    enddo
  enddo
enddo
```



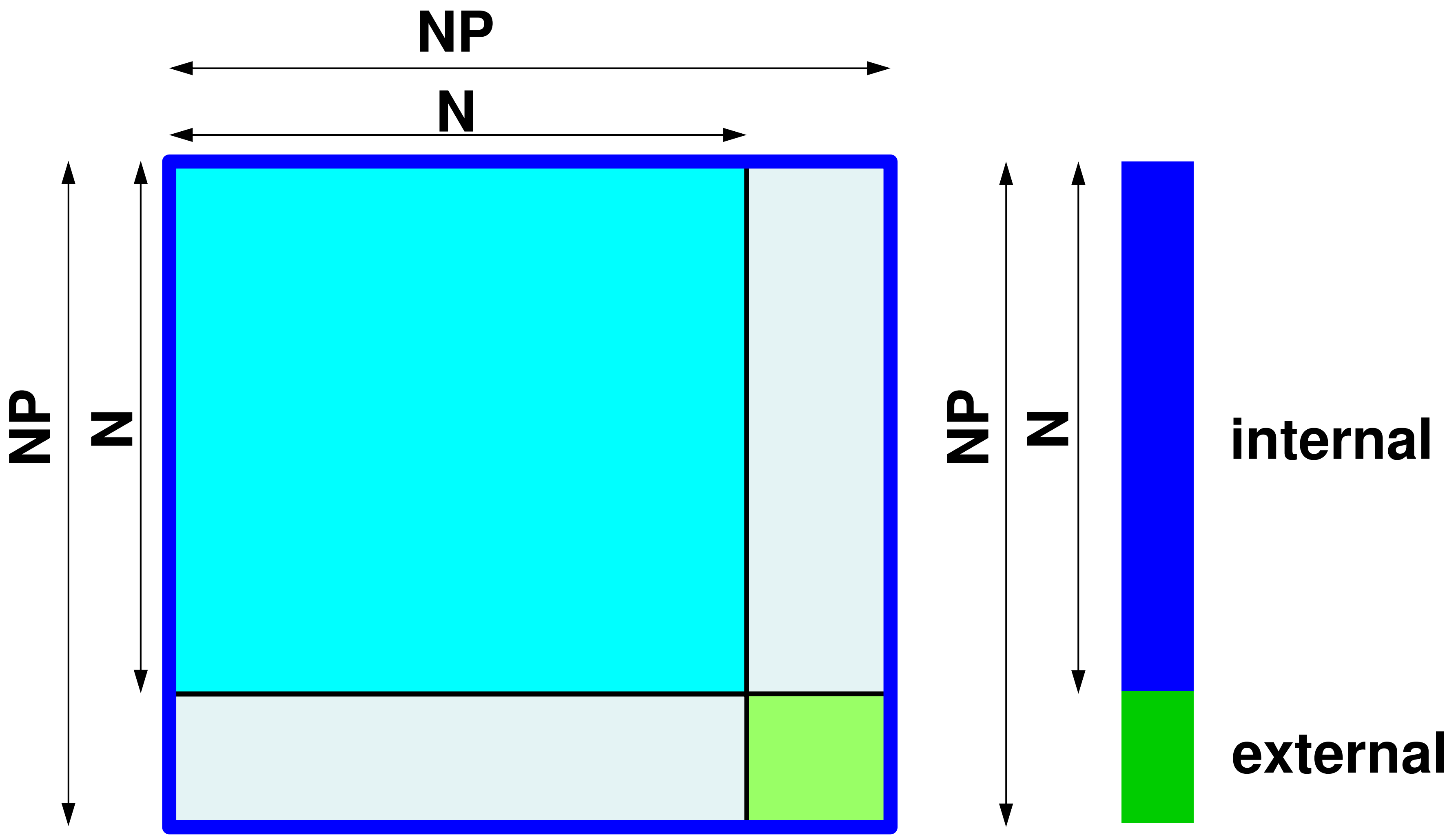
MAT_ASS_MAIN visits all elements

including overlapped elements with external nodes

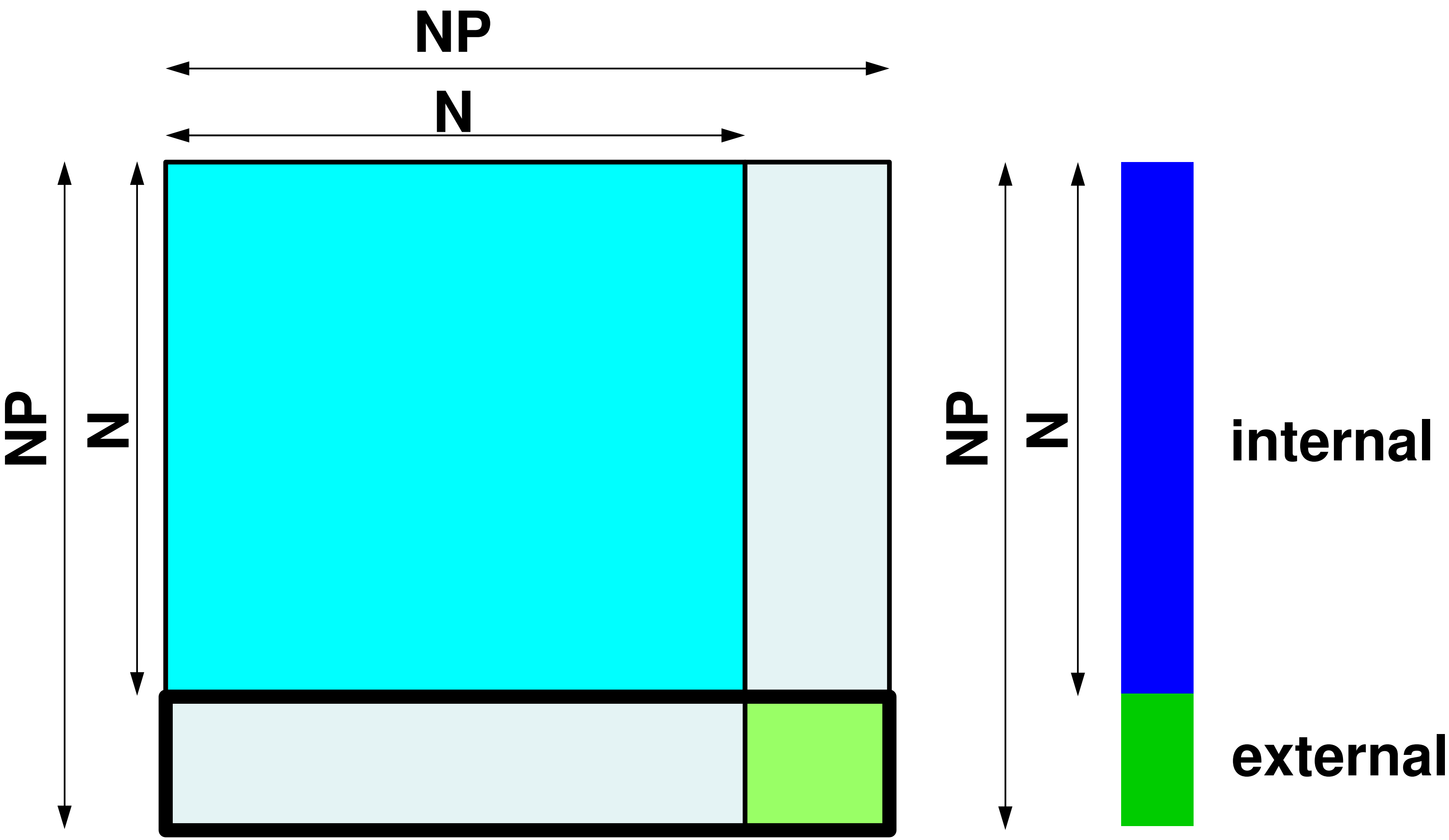




Therefore, we have this matrix



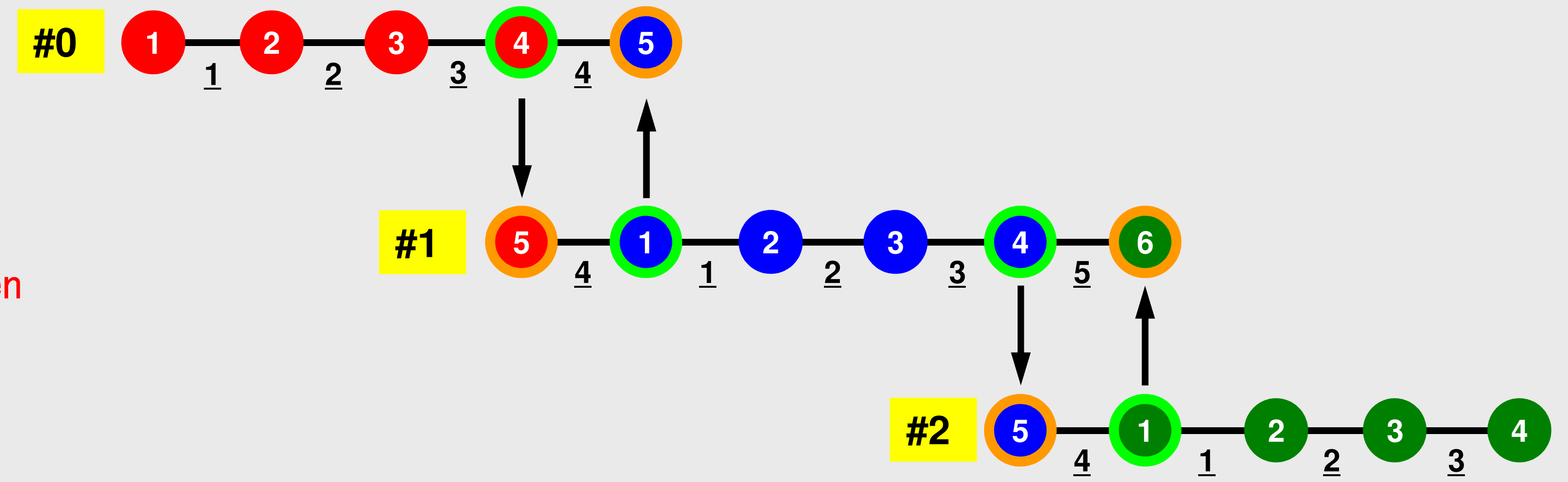
But components of this part are not complete, and not used in computation



Program: 1d.f (9/11)

Boundary Cond., ALMOST NO changes from 1-CPU code

```
!C
!C +---------------------+
!C | BOUNDARY CONDITIONS |
!C +---------------------+
!C===

!C
!C-- X=Xmin

      if (my_rank.eq.0) then
        i = 1
        jS= INDEX(i-1)

        AMAT(jS+1)= 0.d0
        DIAG(i)= 1.d0
        RHS (i)= 0.d0
        do k= 1, NPLU
          if (ITEM(k).eq.1) AMAT(k)= 0.d0
        enddo
      endif

!C===
```



Program: 1d.c(10/11)

Conjugate Gradient Method

```
!C
!C +---------------+
!C | CG iterations |
!C +---------------+
!C===
      R = 1
      Z = 2
      Q = 2
      P = 3
      DD= 4

      do i= 1, N
        W(i,DD)= 1.0D0 / DIAG(i)
      enddo

!C
!C-- {r0}= {b} - [A]{xini} |

!C-    init

        do neib= 1, NEIBPETOT
          do k= export_index(neib-1)+1, export_index(neib)
            kk= export_item(k)
            SENDbuf(k)= PHI(kk)
          enddo
        enddo
```

```
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
```

Conjugate Gradient Method (CG)

- Matrix-Vector Multiply
- Dot Product
- Preconditioning: in the same way as 1CPU code
- DAXPY: in the same way as 1CPU code

Preconditioning, DAXPY

```
!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}
!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
```

Matrix-Vector Multiply (1/2)

Using Comm. Table, {p} is updated before computation

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

      do neib= 1, NEIBPETOT
        do k= export_index(neib-1)+1, export_index(neib)
          kk= export_item(k)
          SENDbuf(k)= W(kk,P)
        enddo
      enddo

      do neib= 1, NEIBPETOT
        is   = export_index(neib-1) + 1
        len_s= export_index(neib) - export_index(neib-1)
        call MPI_Isend (SENDbuf(is), len_s, MPI_DOUBLE_PRECISION,      &
     &                  NEIBPE(neib), 0, MPI_COMM_WORLD, request_send(neib), ierr)
      enddo

      do neib= 1, NEIBPETOT
        ir   = import_index(neib-1) + 1
        len_r= import_index(neib) - import_index(neib-1)
        call MPI_Irecv (RECVbuf(ir), len_r, MPI_DOUBLE_PRECISION,      &
     &                  NEIBPE(neib), 0, MPI_COMM_WORLD, request_recv(neib), ierr)
      enddo

      call MPI_Waitall (NEIBPETOT, request_recv, stat_recv, ierr)

      do neib= 1, NEIBPETOT
        do k= import_index(neib-1)+1, import_index(neib)
          kk= import_item(k)
          W(kk,P)= RECVbuf(k)
        enddo
      enddo
```

Matrix-Vector Multiply (2/2)

{q}= [A]{p}

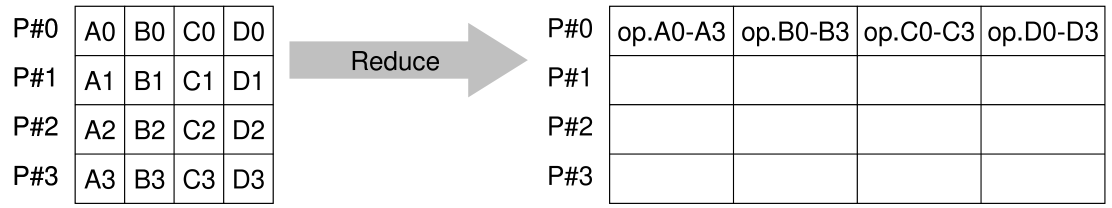
```
call MPI_Waitall (NEIBPETOT, request_send, stat_send, ierr)

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
```

Dot Product

Global Summation by MPI_Allreduce

```
!C
!C-- RHO= {r}{z}

      RHO0= 0.d0
      do i= 1, N
        RHO0= RHO0 + W(i,R)*W(i,Z)
      enddo
      call MPI_Allreduce (RHO0, RHO, 1, MPI_DOUBLE_PRECISION,          &
     &                    MPI_SUM, MPI_COMM_WORLD, ierr)
```

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

Send/Receive Buffer (Sending/Receiving)

- Arrays of “send (sending) buffer” and “receive (receiving) buffer” often appear in MPI.

- Addresses of “send (sending) buffer” and “receive (receiving) buffer” must be different.

Example of MPI_Reduce (1/2)

```
call MPI_REDUCE
(sendbuf,recvbuf,count,datatype,op,root,comm,ierr)
```

```
real(kind=8):: X0, X1

call MPI_REDUCE
(X0, X1, 1, MPI_DOUBLE_PRECISION, MPI_MAX, 0, <comm>, ierr)
```

```
real(kind=8):: X0(4), XMAX(4)

call MPI_REDUCE
(X0, XMAX, 4, MPI_DOUBLE_PRECISION, MPI_MAX, 0, <comm>, ierr)
```

Global Max values of X0(i) go to XMAX(i) on #0 process（i=1~4）

Example of MPI_Reduce (2/2)

```
call MPI_REDUCE
(sendbuf,recvbuf,count,datatype,op,root,comm,ierr)
```

```
real(kind=8):: X0, XSUM

call MPI_REDUCE
(X0, XSUM, 1, MPI_DOUBLE_PRECISION, MPI_SUM, 0, <comm>, ierr)
```

Global summation of X0 goes to XSUM on #0 process.

```
real(kind=8):: X0(4)

call MPI_REDUCE
(X0(1), X0(3), 2, MPI_DOUBLE_PRECISION, MPI_SUM, 0, <comm>, ierr)
```

- Global summation of X0(1) goes to X0(3) on #0 process.
- Global summation of X0(2) goes to X0(4) on #0 process.

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

All reduce →

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,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/recv buffer
  - **datatype** I I data type of elements in send/recv buffer
  - **op** I I reduce operation
  - **comm** I I commuinicator
  - **ierr** I O completion code

CG method (1/5)

```
!C
!C-- {r0}= {b} - [A]{xini}
    do neib= 1, NEIBPETOT
      do k= export_index(neib-1)+1, export_index(neib)
        kk= export_item(k)
        SENDbuf(k)= PHI(kk)
      enddo
    enddo

    do neib= 1, NEIBPETOT
      is   = export_index(neib-1) + 1
      len_s= export_index(neib) - export_index(neib-1)
      call MPI_Isend (SENDbuf(is), len_s,
 &                    MPI_DOUBLE_PRECISION,
 &                    NEIBPE(neib), 0, MPI_COMM_WORLD,
 &                    request_send(neib), ierr)
    enddo

    do neib= 1, NEIBPETOT
      ir   = import_index(neib-1) + 1
      len_r= import_index(neib) - import_index(neib-1)
      call MPI_Irecv (RECVbuf(ir), len_r,
 &                    MPI_DOUBLE_PRECISION,
 &                    NEIBPE(neib), 0, MPI_COMM_WORLD,
 &                    request_recv(neib), ierr)
    enddo
    call MPI_Waitall (NEIBPETOT, request_recv, stat_recv, ier

    do neib= 1, NEIBPETOT
      do k= import_index(neib-1)+1, import_index(neib)
        kk= import_item(k)
        PHI(kk)= RECVbuf(k)
      enddo
    enddo
    call MPI_Waitall (NEIBPETOT, request_send, stat_send, ierr)
```

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

CG method (2/5)

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

      BNRM20= 0.0D0
      do i= 1, N
        BNRM20 = BNRM20  + RHS(i)  **2
        W(i,R) = RHS(i) - W(i,R)
      enddo
      call MPI_Allreduce (BNRM20, BNRM2, 1,
     &                    MPI_DOUBLE_PRECISION,
     &                    MPI_SUM, MPI_COMM_WORLD, ierr)

!C*************************************************************
      do iter= 1, ITERmax

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

      do i= 1, N
        W(i,Z)= W(i,DD) * W(i,R)
      enddo
!C
!C-- RHO= {r}{z}

      RHO0= 0.d0
      do i= 1, N
        RHO0= RHO0 + W(i,R)*W(i,Z)
      enddo
      call MPI_Allreduce (RHO0, RHO, 1, MPI_DOUBLE_PRECISION,
     &                    MPI_SUM, MPI_COMM_WORLD, ierr)
```

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

CG method (3/5)

```
!C
!C-- {p} = {z} if      ITER=1
!C   BETA= RHO / RHO1  otherwise

      if ( iter.eq.1 ) then
        do i= 1, N
          W(i,P)= W(i,Z)
        enddo
       else
         BETA= RHO / RHO1
         do i= 1, N
           W(i,P)= W(i,Z) + BETA*W(i,P)
         enddo
      endif


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

      do neib= 1, NEIBPETOT
        do k= export_index(neib-1)+1, export_index(neib)
          kk= export_item(k)
          SENDbuf(k)= W(kk,P)
        enddo
      enddo

      do neib= 1, NEIBPETOT
        is   = export_index(neib-1) + 1
        len_s= export_index(neib) - export_index(neib-1)
        call MPI_Isend (SENDbuf(is), len_s, MPI_DOUBLE_PRECISION,      &
     &                  NEIBPE(neib), 0, MPI_COMM_WORLD,               &
     &                  request_send(neib), ierr)
      enddo
```

Compute $r^{(0)} = b-[A]x^{(0)}$

for $i = 1, 2, \ldots$

  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

CG method (4/5)

```
  do neib= 1, NEIBPETOT
    ir   = import_index(neib-1) + 1
    len_r= import_index(neib) - import_index(neib-1)
    call MPI_Irecv (RECVbuf(ir), len_r,
&                   MPI_DOUBLE_PRECISION,
&                   NEIBPE(neib), 0, MPI_COMM_WORLD,
&                   request_recv(neib), ierr)
  enddo
  call MPI_Waitall (NEIBPETOT, request_recv, stat_recv, ierr)

  do neib= 1, NEIBPETOT
    do k= import_index(neib-1)+1, import_index(neib)
      kk= import_item(k)
      W(kk,P)= RECVbuf(k)
    enddo
  enddo
  call MPI_Waitall (NEIBPETOT, request_send, stat_send, ierr)

  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

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

  C10= 0.d0
  do i= 1, N
    C10= C10 + W(i,P)*W(i,Q)
  enddo
  call MPI_Allreduce (C10, C1, 1, MPI_DOUBLE_PRECISION, MPI_SUM, MPI_COMM_WORLD, ierr)
  ALPHA= RHO / C1
```

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

  $\mathbf{q^{(i)}= [A]p^{(i)}}$

  $\boldsymbol{\alpha_i = \rho_{i-1}/p^{(i)}q^{(i)}}$

  $x^{(i)}= x^{(i-1)} + \alpha_i p^{(i)}$

  $r^{(i)}= r^{(i-1)} - \alpha_i q^{(i)}$

  check convergence $|r|$

end

CG method (5/5)

```
!C
!C-- {x}= {x} + ALPHA*{p}
!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

      DNRM20 = 0.0
      do i= 1, N
        DNRM20= DNRM20 + W(i,R)**2
      enddo

      call MPI_Allreduce (DNRM20, DNRM2, 1,
     &                    MPI_DOUBLE_PRECISION,
     &                    MPI_SUM, MPI_COMM_WORLD, ierr)

        RESID= dsqrt(DNRM2/BNRM2)

        if (my_rank.eq.0.and.mod(iter,1000).eq.0) then
          write (*, '(i8,1pe16.6)') iter, RESID
        endif

        if ( RESID.le.EPS) goto 900
        RHO1 = RHO

      enddo
```

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

  $q^{(i)}= [A]p^{(i)}$

  $\alpha_i = \rho_{i-1}/p^{(i)}q^{(i)}$

  $\mathbf{x}^{(i)}= \mathbf{x}^{(i-1)} + \alpha_i \mathbf{p}^{(i)}$

  $\mathbf{r}^{(i)}= \mathbf{r}^{(i-1)} - \alpha_i \mathbf{q}^{(i)}$

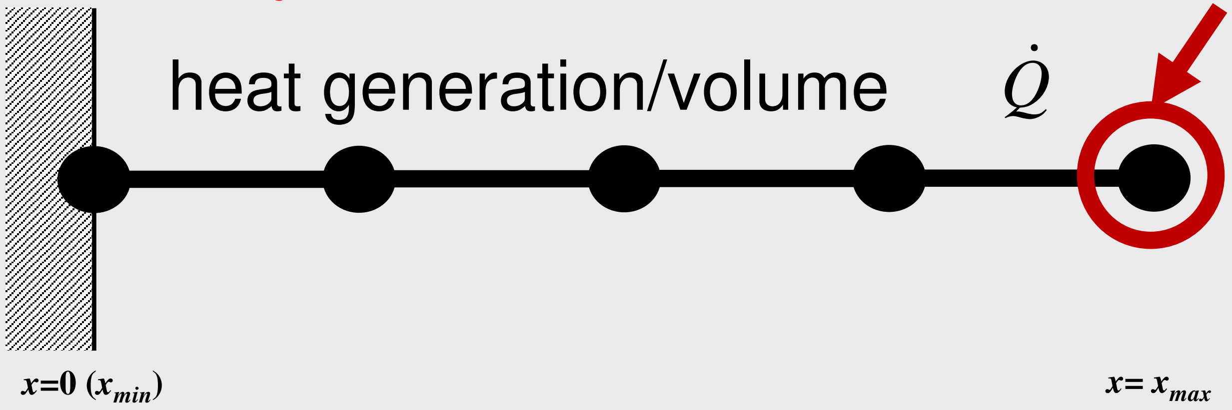
  check convergence $|r|$

end

Program: 1d.f (11/11)

Output

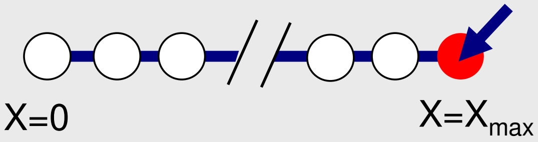
```
      S1Time= MPI_Wtime ()
<Matrix Assembling>
      E1Time= MPI_Wtime ()
<Linear Solver>
      E2Time= MPI_Wtime ()

      if (my_rank.eq.0) then
        write (*, '(i8,1pe16.6)') iter, RESID
      endif

      if (my_rank.eq.0) then
        write (*,'(2(1pe16.6),a)') E1Time-S1Time, E2Time-E1Time,
     &                                'sec.'
      endif

      if (my_rank.eq.PETOT-1) then
        write (*,'(/a)') '### TEMPERATURE'
        write (*,'(2i8, 1pe27.20,//)') my_rank, N, PHI(N)
      endif

      call MPI_FINALIZE (ierr)
      end program heat1Dp
```



- Overview
- Distributed Local Data
- Program
- **Results**

Validation (2/2)
NEg=1,000

```
!C
!C-- OUTPUT
      if (my_rank.eq.0) then
        write (*,'(2(1pe16.6))') E1Time-S1Time, E2Time-E1Time
      endif

      if (my_rank.eq.PETOT-1) then
        write (*,'(/a)') '### TEMPERATURE'
        write (*,'(2i8, 1pe27.20,//)') my_rank, N, PHI(N)
      endif

      call MPI_FINALIZE (ierr)
      end program heat1Dp
```



$$T = -\frac{1}{2\lambda}\dot{Q}x^2 + \frac{\dot{Q}x_{\max}}{\lambda}x$$

$$\lambda = 1, \dot{Q} = 1$$

$$x = x_{\max} = 1000 \Rightarrow T = -\frac{1}{2}(1000)^2 + (1000)^2 = 5.0 \times 10^5$$

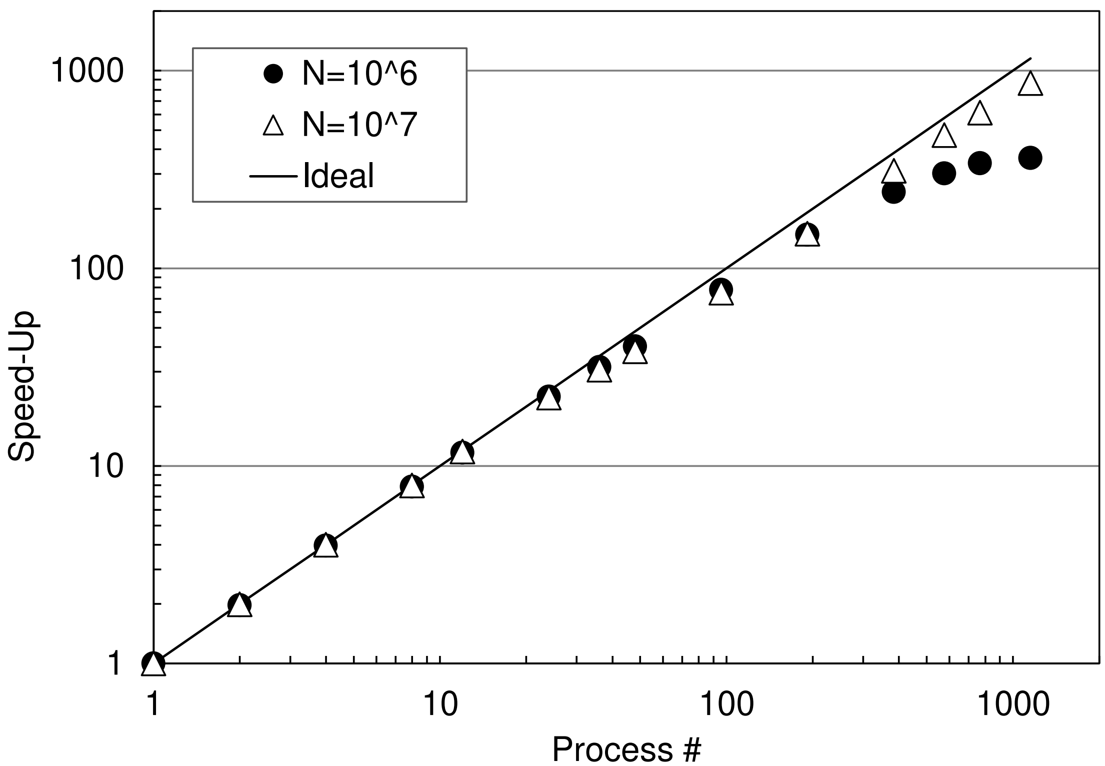
PPn	Iter's	N	PHI(N)
1	1000	1000	5.000000e5
2	1000	500	5.000000e5
4	1000	250	5.000000e5
8	1000	125	5.000000e5
16	1000	62	5.000000e5
32	1000	31	5.000000e5
48	1000	20	5.000000e5

Time for CG Computation: $N=10^6$, 10^7

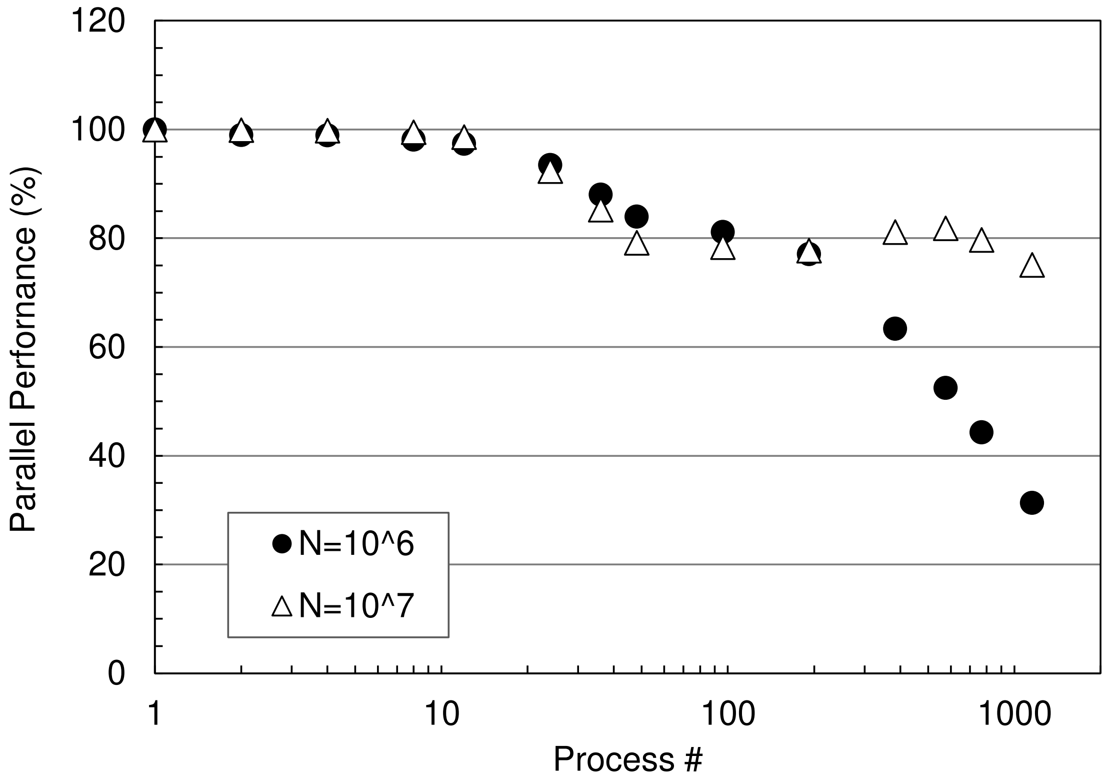
200 Iterations, Strong Scaling, C Language

Based on the performance of a single core, 48 cores/node for more than 2 nodes, Fastest for 5 measurements

Speed-Up



Parallel Performance



Performance is lower than ideal one

- Time for MPI communication
  - Time for sending data
  - Communication bandwidth between nodes
  - Time is proportional to size of sending/receiving buffers
- Time for starting MPI
  - latency
  - does not depend on size of buffers
    - depends on number of calling, increases according to process #
  - $O(10^0)$-$O(10^1)$ μsec.
- Synchronization of MPI
  - Increases according to number of processes

Summary: Parallel FEM

- Proper design of data structure of distributed local meshes.

- Open Technical Issues
  - Parallel Mesh Generation, Parallel Visualization
  - Parallel Preconditioner for Ill-Conditioned Problems
  - Large-Scale I/O

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 <u>boundary</u> pts are copied into sending buffers
  - Send/Recv
  - Values at <u>external</u> pts are updated through receiving buffers

If numbering of external nodes is continuous in each neighboring process ...

	84	81	85	82	83	86	88	87	
96	57	58	59	60	61	62	63	64	73
95	49	50	51	52	53	54	55	56	74
94	41	42	43	44	45	46	47	48	80
93	33	34	35	36	37	38	39	40	79
92	25	26	27	28	29	30	31	32	78
91	17	18	19	20	21	22	23	24	77
90	9	10	11	12	13	14	15	16	76
89	1	2	3	4	5	6	7	8	75
	65	66	67	68	69	70	71	72	

[A]{p}= {q} (Original), 1d.f

```
      allocate (stat_send(MPI_STATUS_SIZE,NEIBPETOT))
      allocate (stat_recv(MPI_STATUS_SIZE,NEIBPETOT))
      allocate (request_send(NEIBPETOT)); allocate (request_recv(NEIBPETOT))

      do neib= 1, NEIBPETOT
        do k= export_index(neib-1)+1, export_index(neib)
          kk= export_item(k)
          SENDbuf(k)= W(kk,P)
        enddo
      enddo

      do neib= 1, NEIBPETOT
        is   = export_index(neib-1) + 1
        len_s= export_index(neib) - export_index(neib-1)
        call MPI_Isend (SENDbuf(is), len_s, MPI_DOUBLE_PRECISION,       &
     &                  NEIBPE(neib), 0, MPI_COMM_WORLD,                &
     &                  request_send(neib), ierr)
      enddo

      do neib= 1, NEIBPETOT
        ir   = import_index(neib-1) + 1
        len_r= import_index(neib) - import_index(neib-1)
        call MPI_Irecv (RECVbuf(ir), len_r, MPI_DOUBLE_PRECISION,       &
     &                  NEIBPE(neib), 0, MPI_COMM_WORLD,                &
     &                  request_recv(neib), ierr)
      enddo
      call MPI_Waitall (NEIBPETOT, request_recv, stat_recv, ierr)

      do neib= 1, NEIBPETOT
        do k= import_index(neib-1)+1, import_index(neib)
          kk= import_item(k)
          W(kk,P)= RECVbuf(k)
        enddo
      enddo
      call MPI_Waitall (NEIBPETOT, request_send, stat_send, ierr)
```

[A]{p}= {q} (Mod.): No Copy for RECV
1d2.f, a little bit faster

```
      allocate (stat_send(MPI_STATUS_SIZE,2*NEIBPETOT))
      allocate (request_send(2*NEIBPETOT))

      do neib= 1, NEIBPETOT
        do k= export_index(neib-1)+1, export_index(neib)
          kk= export_item(k)
          SENDbuf(k)= W(kk,P)
        enddo
      enddo

      do neib= 1, NEIBPETOT
        is   = export_index(neib-1) + 1
        len_s= export_index(neib) - export_index(neib-1)
        call MPI_Isend (SENDbuf(is), len_s, MPI_DOUBLE_PRECISION,       &
     &                  NEIBPE(neib), 0, MPI_COMM_WORLD,                &
     &                  request_send(neib), ierr)
      enddo

      do neib= 1, NEIBPETOT
        ir   = import_index(neib-1) + 1
        len_r= import_index(neib) - import_index(neib-1)
        call MPI_Irecv (W(ir+N,P), len_r, MPI_DOUBLE_PRECISION,       &
     &                  NEIBPE(neib), 0, MPI_COMM_WORLD,                &
     &                  request_send(neib+NEIBPETOT), ierr)
      enddo

      call MPI_Waitall (2*NEIBPETOT, request_send, stat_send, ierr)
```