

# **Introduction to Programming by MPI for Parallel FEM Report S1 & S2 in Fortran**

**Kengo Nakajima**

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

# Motivation for Parallel Computing

## (and this class)

- Large-scale parallel computer enables fast computing in large-scale scientific simulations with detailed models. Computational science develops new frontiers of science and engineering.
- Why parallel computing ?
  - faster & larger
  - “larger” is more important from the view point of “new frontiers of science & engineering”, but “faster” is also important.
  - + more complicated
  - Ideal: Scalable
    - Solving  $N^x$  scale problem using  $N^x$  computational resources during same computation time.

# Overview

- What is MPI ?
- Your First MPI Program: Hello World
- Global/Local Data
- Collective Communication
- Peer-to-Peer Communication

# What is MPI ? (1/2)

- Message Passing Interface
- “Specification” of message passing API for distributed memory environment
  - Not a program, Not a library
    - <http://phase.hpcc.jp/phase/mpi-j/ml/mpi-j-html/contents.html>
- History
  - 1992 MPI Forum
  - 1994 MPI-1
  - 1997 MPI-2: MPI I/O
  - 2012 MPI-3: Fault Resilience, Asynchronous Collective
- Implementation
  - mpich ANL (Argonne National Laboratory), OpenMPI, MVAPICH
  - H/W vendors
  - C/C++, FORTRAN, Java ; Unix, Linux, Windows, Mac OS

# What is MPI ? (2/2)

- “mpich” (free) is widely used
  - supports MPI-2 spec. (partially)
  - MPICH2 after Nov. 2005.
  - <http://www-unix.mcs.anl.gov/mpi/>
- Why MPI is widely used as *de facto standard* ?
  - Uniform interface through MPI forum
    - Portable, can work on any types of computers
    - Can be called from Fortran, C, etc.
  - mpich
    - free, supports every architecture
- PVM (Parallel Virtual Machine) was also proposed in early 90’s but not so widely used as MPI

# References

- W.Gropp et al., Using MPI second edition, MIT Press, 1999.
- M.J.Quinn, Parallel Programming in C with MPI and OpenMP, McGrawhill, 2003.
- W.Gropp et al., MPI : The Complete Reference Vol.I, II, MIT Press, 1998.
- <http://www-unix.mcs.anl.gov/mpi/www/>
  - API (Application Interface) of MPI

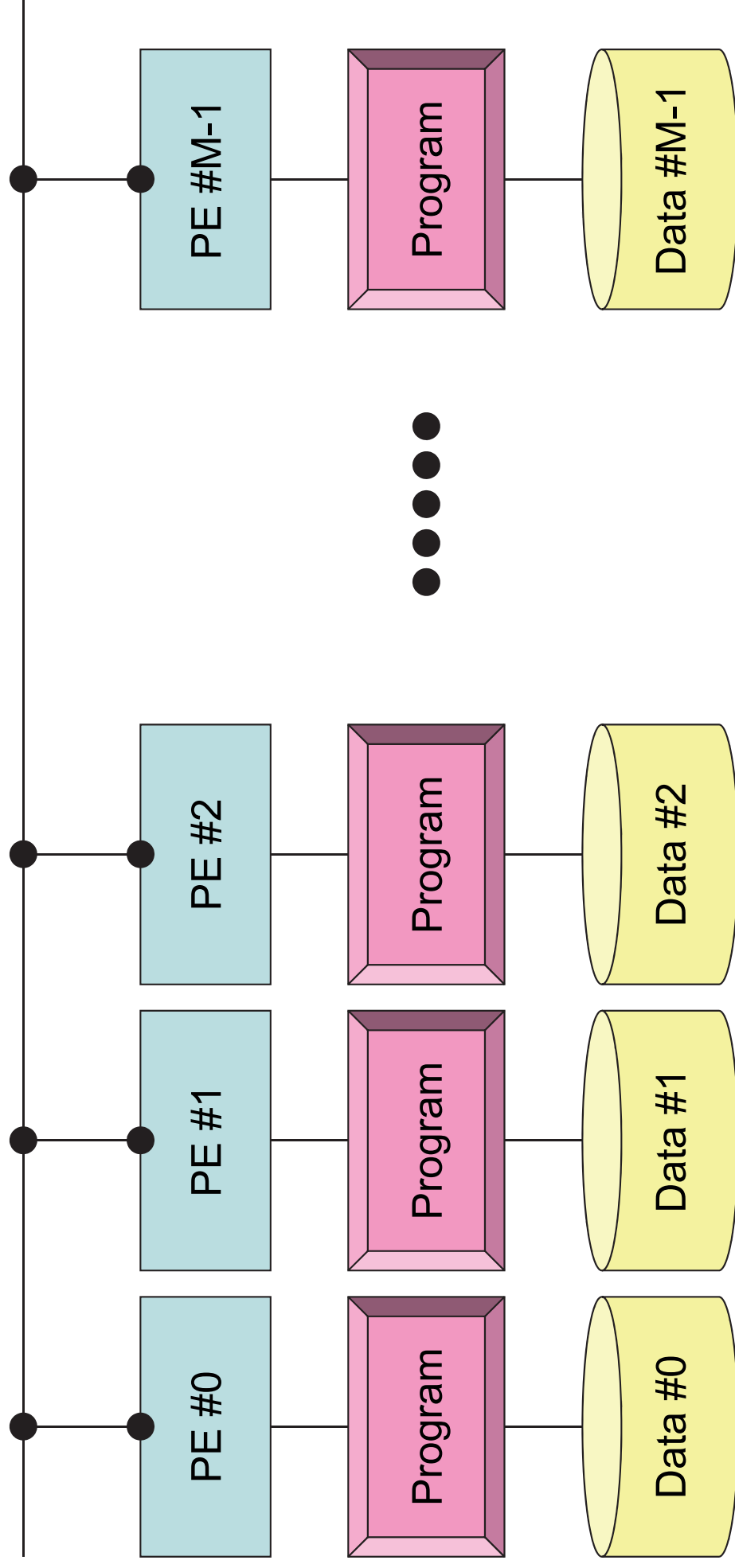
# How to learn MPI (1/2)

- Grammar
  - 10-20 functions of MPI-1 will be taught in the class
    - although there are many convenient capabilities in MPI-2
  - If you need further information, you can find information from web, books, and MPI experts.
- Practice is important
  - Programming
  - “Running the codes” is the most important
- Be familiar with or “grab” the idea of SPMD/SIMD op’s
  - Single Program/Instruction Multiple Data
  - Each process does same operation for different data
    - Large-scale data is decomposed, and each part is computed by each process
  - Global/Local Data, Global/Local Numbering

PE: Processing Element  
Processor, Domain, Process

# SPMD

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



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

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.



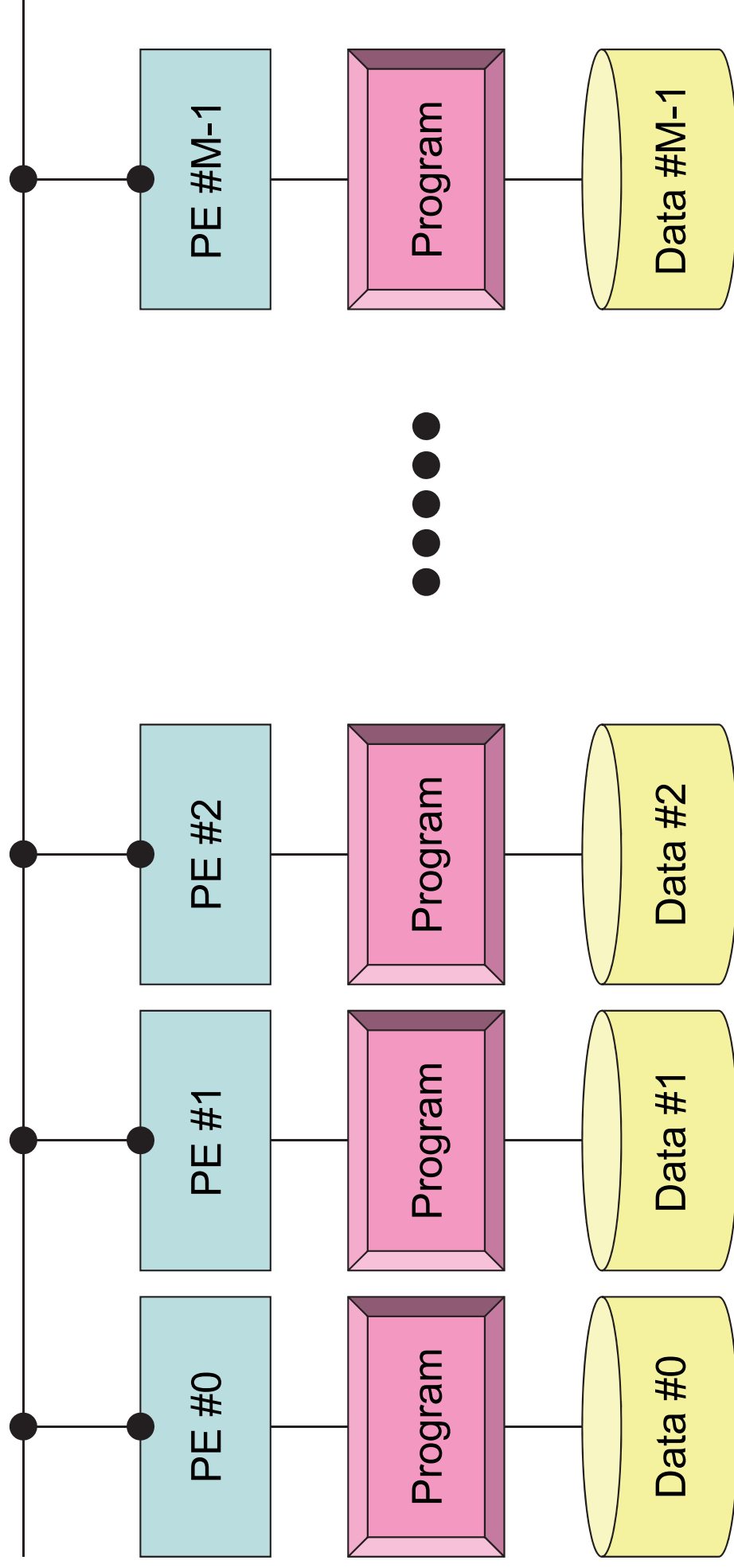
# Some Technical Terms

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

PE: Processing Element  
Processor, Domain, Process

# SPMD

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



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

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.

# How to learn MPI (2/2)

- NOT so difficult.
- Therefore, 2-3 lectures are enough for just learning grammar of MPI.
- Grab the idea of SPMD !

# Schedule

- MPI
  - Basic Functions
  - Collective Communication
  - Point-to-Point (or Peer-to-Peer) Communication
- 90 min. x 4-5 lectures
  - Collective Communication
    - Report S1
  - Point-to-Point/Peer-to-Peer Communication
    - Report S2: Parallelization of 1D code
  - At this point, you are almost an expert of MPI programming.

- What is MPI ?
- **Your First MPI Program: Hello World**
- Global/Local Data
- Collective Communication
- Peer-to-Peer Communication

# Login to Oakleaf-FX

```
ssh t85**@oakleaf-fx.cc.u-tokyo.ac.jp
```

## Create directory

```
>$ cd  
>$ mkdir fem2 (your favorite name)  
>$ cd fem2
```

In this class this top-directory is called **<\$O-fem2>**.  
Files are copied to this directory.

Under this directory, s1, s2, s1-ref are created:

```
<$O-s1> = <$O-fem2>/mpi/s1  
<$O-s2> = <$O-fem2>/mpi/s2
```

Oakleaf-FX

ECCS2012

# Copying files on Oakleaf-FX

## Copy

```
>$ cd <$O-fem2>  
>$ cp /home/z30088/fem2/F/s1.tar .  
>$ tar xvf s1.tar
```

## Confirm directory

```
>$ ls  
mpi
```

```
>$ cd mpi/S1
```

**This directory is called as <\$O-S1>.**  
**<\$O-S1> = <\$O-fem2>/mpi/S1**

# First Example

**hello.f**

```
implicit REAL*8 (A-H,O-Z)
include 'mpif.h'
integer :: PETOT, my_rank, ierr

call MPI_INIT      (ierr)
call MPI_COMM_SIZE (MPI_COMM_WORLD, PETOT, ierr )
call MPI_COMM_RANK (MPI_COMM_WORLD, my_rank, ierr )

write (*,'(a,2i8)') 'Hello World FORTRAN', my_rank, PETOT

call MPI_FINALIZE (ierr)

stop
end
```

**hello.c**

```
#include "mpi.h"
#include <stdio.h>
int main(int argc, char **argv)
{
    int n, myid, numprocs, i;

    MPI_Init(&argc,&argv);
    MPI_Comm_size(MPI_COMM_WORLD,&numprocs);
    MPI_Comm_rank(MPI_COMM_WORLD,&myid);
    printf ("Hello World %d\n", myid);
    MPI_Finalize();
}
```



# Compiling hello.f/c

```
>$ cd <$0-S1>
>$ mpifrtpx -Kfast hello.f
>$ mpifccpx -Kfast hello.c
```

## FORTRAN

```
$> mpifrtpx -Kfast hello.f
```

“mpifrtpx”:

FORTRAN90+MPIによってプログラムをコンパイルする際に  
必要な、コンパイラ、ライブラリ等がバインドされている

## C言語

```
$> mpifccpx -Kfast hello.c
```

“mpifccpx”:

C+MPIによってプログラムをコンパイルする際に  
必要な、コンパイラ、ライブラリ等がバインドされている

# Running Job

- Batch Jobs
  - Only batch jobs are allowed.
  - Interactive executions of jobs are not allowed.
- How to run
  - writing job script
  - submitting job
  - checking job status
  - checking results
- Utilization of computational resources
  - 1-node (16 cores) is occupied by each job.
  - Your node is not shared by other jobs.

# Job Script

- `<$O-S1>/hello.sh`
- Scheduling + Shell Script

```
#!/bin/sh
#PJM -L "node=1"
#PJM -L "elapsed=00:10:00"
#PJM -L "rscgrp=lecture5"
#PJM -g "gt85"
#PJM -j
#PJM -o "hello.lst"
#PJM --mpi "proc=4"

mpiexec ./a.out
```

Number of Nodes  
Computation Time  
Name of "QUEUE"  
Group Name (Wallet)  
Standard Output  
MPI Process #  
Execs

8プロセス  
"node=1"  
"proc=8"

16プロセス  
"node=1"  
"proc=16"

32プロセス  
"node=2"  
"proc=32"

64プロセス  
"node=4"  
"proc=64"

192プロセス  
"node=12"  
"proc=192"

# Submitting Jobs

```
>$ cd <$O-S1>
>$ pjsub hello.sh

>$ cat hello.lst

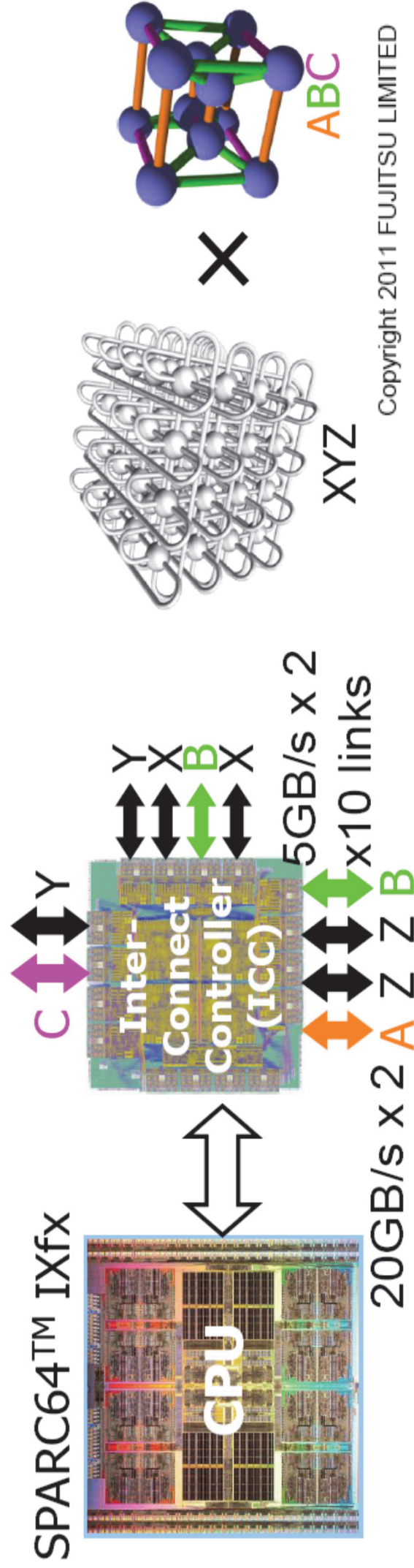
Hello World 0
Hello World 3
Hello World 2
Hello World 1
```

# Available QUEUE'S

- Following 2 queues are available.
- 1 Tofu (12 nodes) can be used
  - **lecture**
    - 12 nodes (192 cores), 15 min., valid until the end of October, 2014
    - Shared by all “educational” users
  - **lecture2**
    - 12 nodes (192 cores), 15 min., active during class time
    - More jobs (compared to **lecture**) can be processed upon availability.

# Tofu Interconnect

- Node Group
  - 12 nodes
  - A-/C- axis: 4 nodes in system board, B-axis: 3 boards
- 6D: (X,Y,Z,A,B,C)
  - ABC 3D Mesh: in each node group:  $2 \times 2 \times 3$
  - XYZ 3D Mesh: connection of node groups:  $10 \times 5 \times 8$
- Job submission according to network topology is possible:
  - Information about used “XYZ” is available after execution.



# Submitting & Checking Jobs

- Submitting Jobs
- Checking status of jobs
- Deleting/aborting
- Checking status of queues
- Detailed info. of queues
- Number of running jobs
- Limitation of submission

```

pjsub SCRIPT NAME
pjstat
pjdel JOB_ID
pjstat --rsc
pjstat --rsc -x
pjstat --rsc -b
pjstat --limit
  
```

```
[z30088@oakleaf-fx-6 S2-ref]$ pjstat
```

```
Oakleaf-FX scheduled stop time: 2012/09/28(Fri) 09:00:00 (Remain: 31days 20:01:46)
```

| JOB_ID | JOB_NAME | STATUS  | PROJECT | RSCGROUP | START_DATE     | ELAPSE   | TOKEN | NODE:COORD |
|--------|----------|---------|---------|----------|----------------|----------|-------|------------|
| 334730 | go.sh    | RUNNING | gt61    | lecture  | 08/27 12:58:08 | 00:00:05 | 0.0   | 1          |

# Basic/Essential Functions

```
implicit REAL*8 (A-H,O-Z)
include 'mpif.h'
integer :: PETOT, my_rank, ierr

call MPI_INIT      (ierr)
call MPI_COMM_SIZE (MPI_COMM_WORLD, PETOT, ierr )
call MPI_COMM_RANK (MPI_COMM_WORLD, my_rank, ierr )

write (*,'(a,2i8)') 'Hello World FORTRAN', my_rank, PETOT

call MPI_FINALIZE (ierr)

stop
end
```

**'mpif.h', "mpi.h"**

Essential Include file

"use mpi" is possible in F90

**MPI\_Init**

Initialization

**MPI\_Comm\_size**

Number of MPI Processes

`mpirun -np XX <prog>`

**MPI\_Comm\_rank**

Process ID starting from 0

**MPI\_Finalize**

Termination of MPI processes

```
#include "mpi.h"
#include <stdio.h>
int main(int argc, char **argv)
{
    int n, myid, numprocs, i;

    MPI_Init(&argc,&argv);
    MPI_Comm_size(MPI_COMM_WORLD,&numprocs);
    MPI_Comm_rank(MPI_COMM_WORLD,&myid);

    printf ("Hello World %d\n", myid);
    MPI_Finalize();
}
```



# Difference between FORTRAN/C

- (Basically) same interface
  - In C, UPPER/lower cases are considered as different
    - e.g.: **MPI\_Comm\_size**
      - MPI: UPPER case
      - First character of the function except “MPI\_” is in UPPER case.
      - Other characters are in lower case.
- In Fortran, return value `ierr` has to be added at the end of the argument list.
- C needs special types for variables:
  - MPI\_Comm, MPI\_Datatype, MPI\_Op etc.
- **MPI\_INIT** is different:
  - call `MPI_INIT (ierr)`
  - `MPI_Init (int *argc, char ***argv)`

# What's are going on ?

```
implicit REAL*8 (A-H,O-Z)
include 'mpif.h'
integer :: PETOT, my_rank, ierr

call MPI_INIT      (ierr)
call MPI_COMM_SIZE (MPI_COMM_WORLD, PETOT, ierr )
call MPI_COMM_RANK (MPI_COMM_WORLD, my_rank, ierr )

write (*,'(a,2i8)') 'Hello World FORTRAN', my_rank, PETOT

call MPI_FINALIZE (ierr)

stop
end
```

```
#!/bin/sh
#PJM -L "node=1"
#PJM -L "elapsed=00:10:00"
#PJM -L "rscgrp=lecture"
#PJM -g "gt64"
#PJM -j
#PJM -o "hello.lst"
#PJM --mpi "proc=4"

mpieexec ./a.out
```

- **mpieexec** starts up 4 MPI processes ("proc=4")
  - A single program runs on four processes.
  - each process writes a value of `myid`
- Four processes do same operations, but values of `myid` are different.
- Output of each process is different.
- That is **SPMD** !

# mpi.h, mpif.h

```
implicit REAL*8 (A-H,O-Z)
include 'mpif.h'
integer :: PETOT, my_rank, ierr

call MPI_INIT      (ierr)
call MPI_COMM_SIZE (MPI_COMM_WORLD, PETOT, ierr)
call MPI_COMM_RANK (MPI_COMM_WORLD, my_rank, ierr)

write (*,'(a,2i8)') 'Hello World FORTRAN', my_rank, PETOT

call MPI_FINALIZE (ierr)

stop
end
```

```
#include "mpi.h"
#include <stdio.h>
int main(int argc, char **argv)
{
    int n, myid, numprocs, i;

    MPI_Init(&argc, &argv);
    MPI_Comm_size(MPI_COMM_WORLD, &numprocs);
    MPI_Comm_rank(MPI_COMM_WORLD, &myid);

    printf ("Hello World %d\n", myid);
    MPI_Finalize();
}
```

- Various types of parameters and variables for MPI & their initial values.
- Name of each var. starts from “MPI\_”
- Values of these parameters and variables cannot be changed by users.
- Users do not specify variables starting from “MPI\_” in users’ programs.

# MPI\_INIT

- Initialize the MPI execution environment (required)
- It is recommended to put this BEFORE all statements in the program.

- **call MPI\_INIT (ierr)**

– ierr      I      O      Completion Code

```
implicit REAL*8 (A-H,O-Z)
include 'mpif.h'
integer :: PETOT, my_rank, ierr

call MPI_INIT (ierr)
call MPI_COMM_SIZE (MPI_COMM_WORLD, PETOT, ierr )
call MPI_COMM_RANK (MPI_COMM_WORLD, my_rank, ierr )

write (*,'(a,2i8)') 'Hello World FORTRAN', my_rank, PETOT

call MPI_FINALIZE (ierr)

stop
end
```

# MPI\_FINALIZE

- Terminates MPI execution environment (required)
- It is recommended to put this AFTER all statements in the program.
- **Please do not forget this.**

- **call MPI\_FINALIZE (ierr)**

- ierr      I      O      completion code

```
implicit REAL*8 (A-H,O-Z)
include 'mpif.h'
integer :: PETOT, my_rank, ierr

call MPI_INIT      (ierr)
call MPI_COMM_SIZE (MPI_COMM_WORLD, PETOT, ierr )
call MPI_COMM_RANK (MPI_COMM_WORLD, my_rank, ierr )

write (*,'(a,2i8)') 'Hello World FORTRAN', my_rank, PETOT

call MPI_FINALIZE (ierr)

stop
end
```

# MPI\_COMM\_SIZE

- Determines the size of the group associated with a communicator
- not required, but very convenient function
- **call MPI\_COMM\_SIZE (comm, size, ierr)**
  - comm      I      I      communicator
  - size      I      O      number of processes in the group of communicator
  - ierr      I      O      completion code

```
implicit REAL*8 (A-H,O-Z)
include 'mpif.h'
integer :: PETOT, my_rank, ierr

call MPI_INIT      (ierr)
call MPI_COMM_SIZE (MPI_COMM_WORLD, PETOT, ierr )
call MPI_COMM_RANK (MPI_COMM_WORLD, my_rank, ierr )

write (*, '(a,2i8)') 'Hello World FORTRAN', my_rank, PETOT

call MPI_FINALIZE (ierr)

stop
end
```

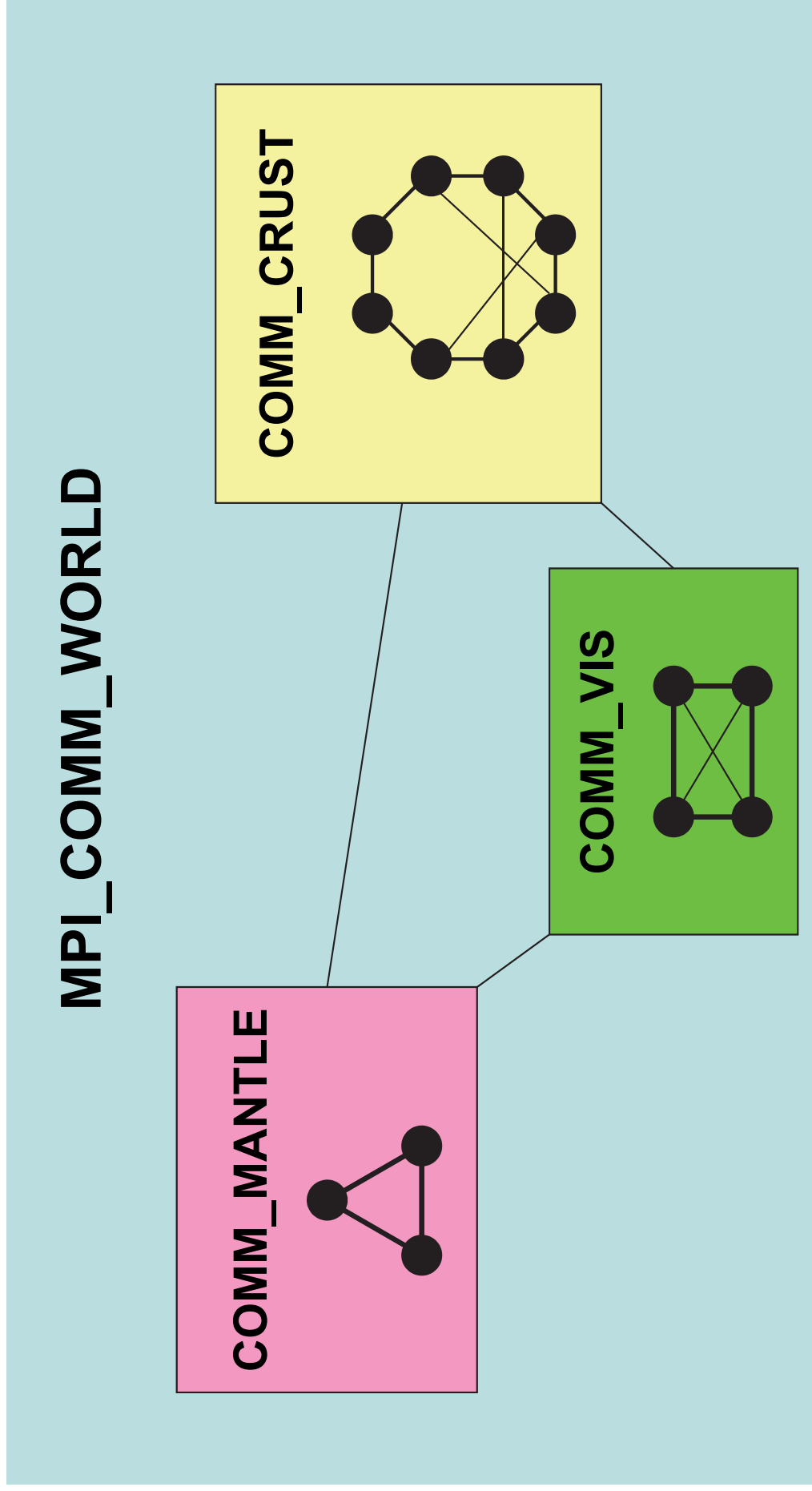
# What is Communicator ?

**MPI\_Comm\_Size (MPI\_COMM\_WORLD, PETOT)**

- Group of processes for communication
- Communicator must be specified in MPI program as a unit of communication
- All processes belong to a group, named “**MPI\_COMM\_WORLD**” (default)
- Multiple communicators can be created, and complicated operations are possible.
  - Computation, Visualization
- Only “MPI\_COMM\_WORLD” is needed in this class.

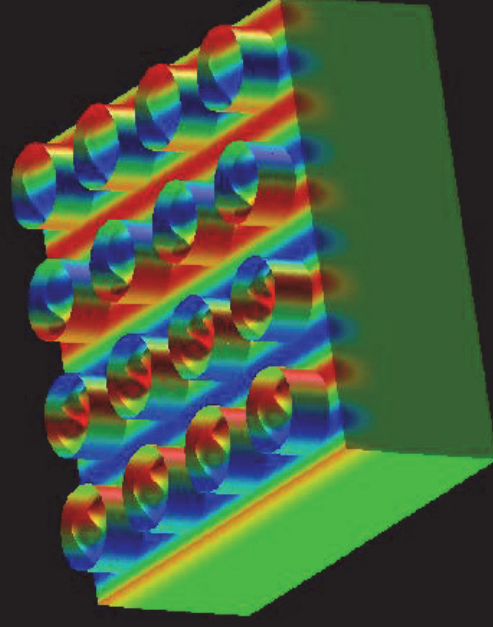
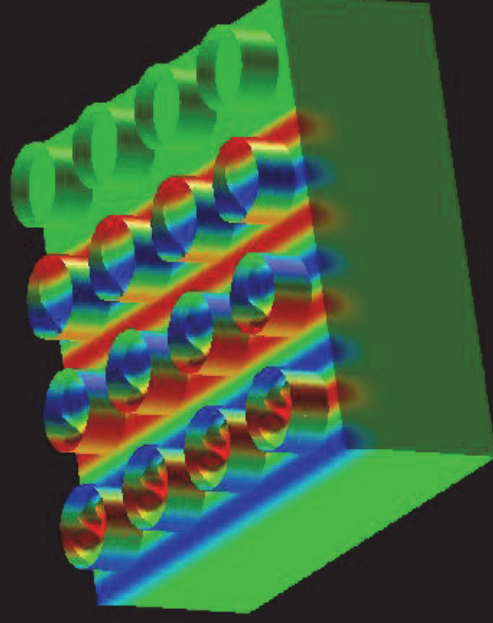
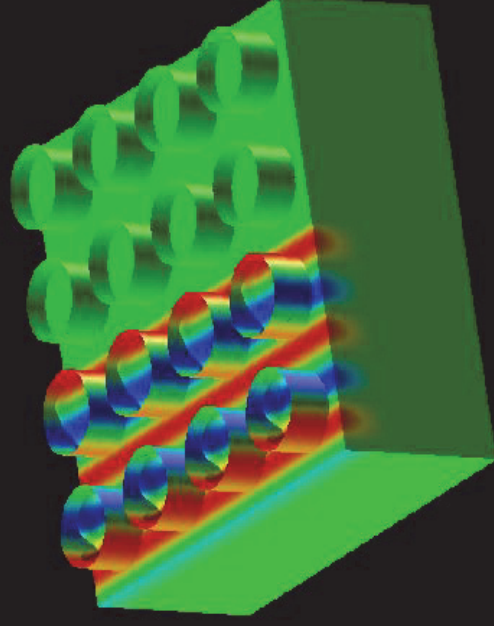
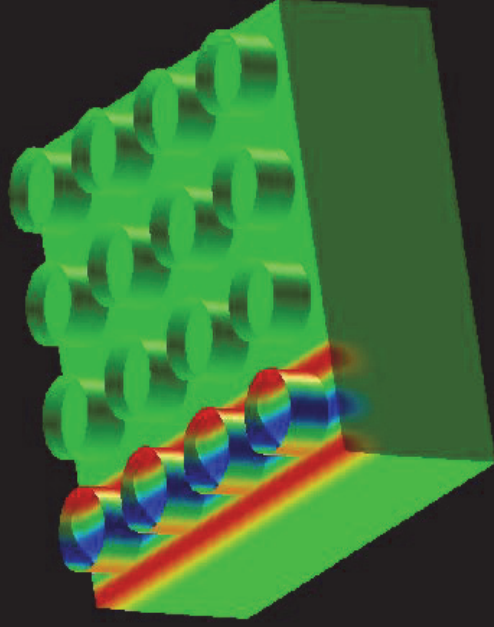
# Communicator in MPI

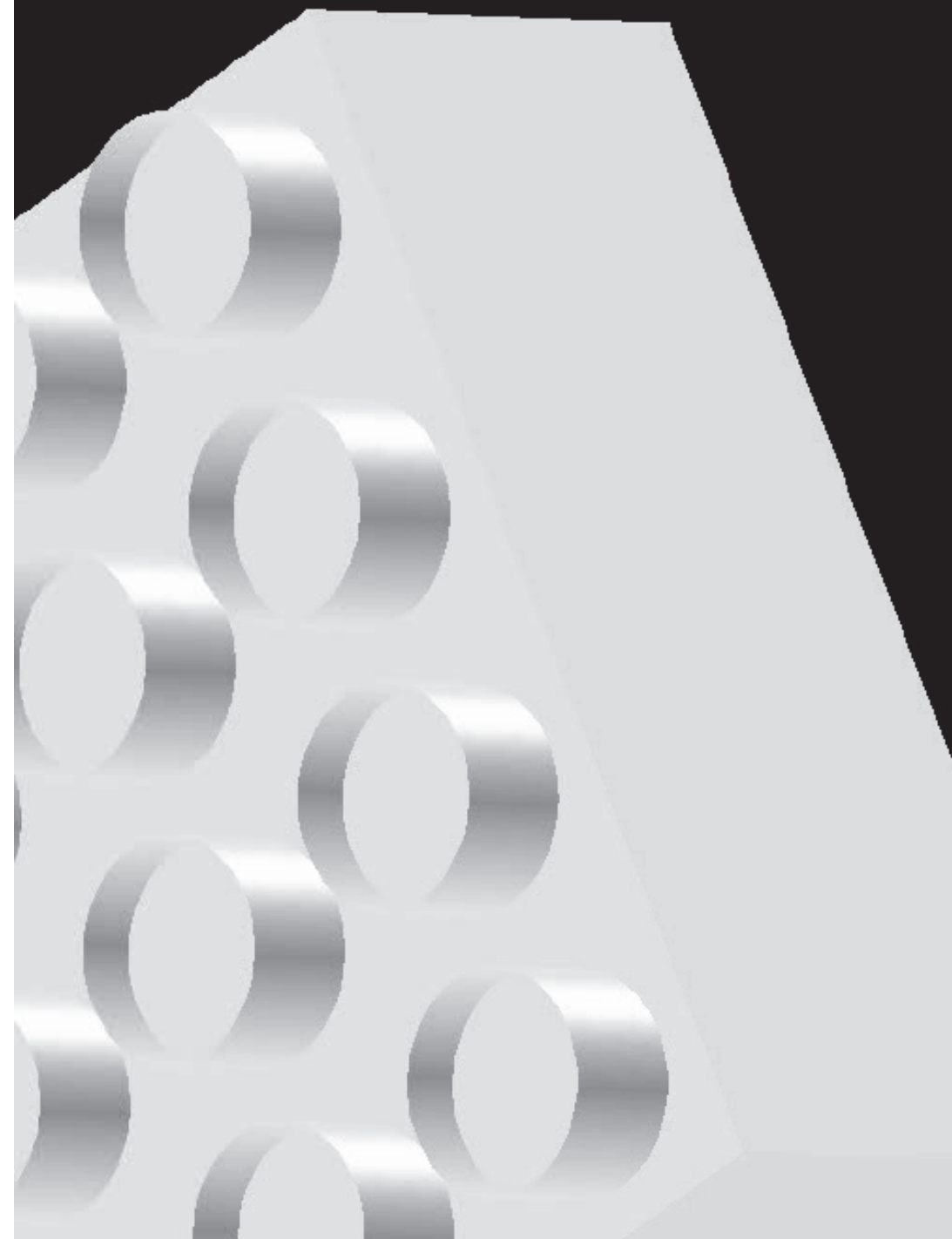
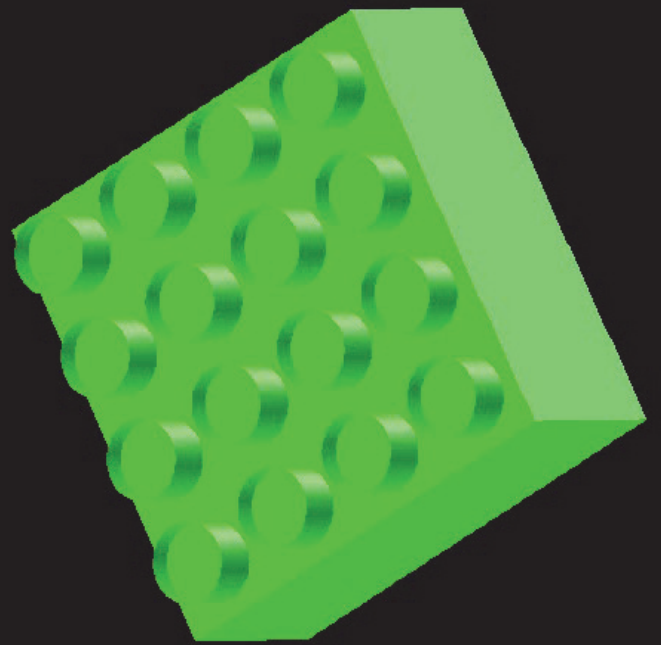
One process can belong to multiple communicators





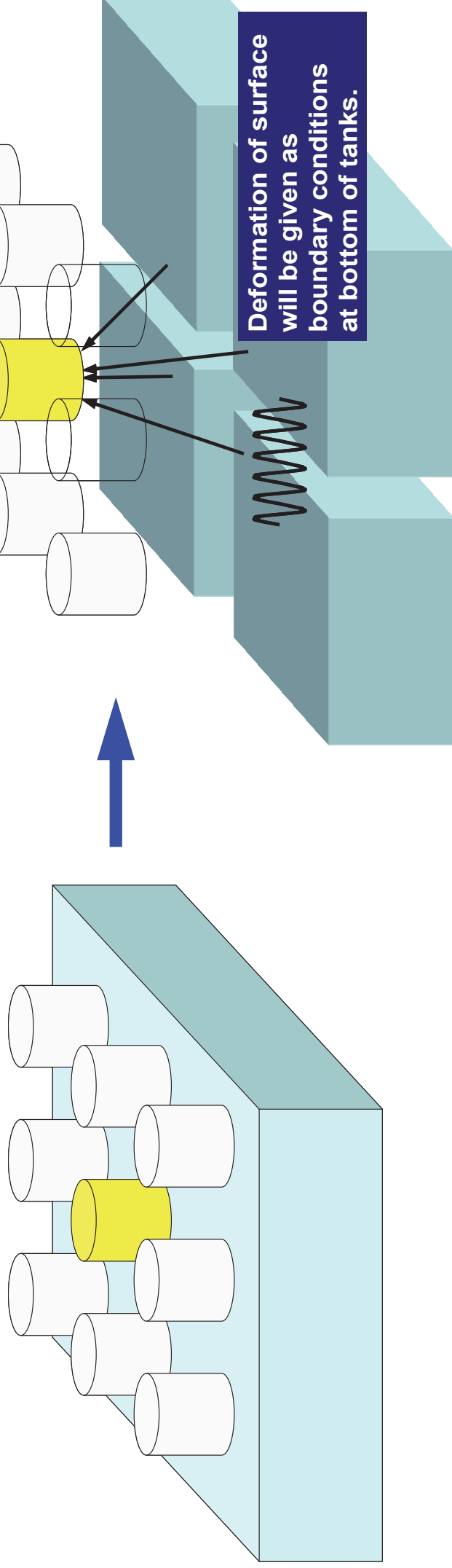
# Coupling between “Ground Motion” and “Sloshing of Tanks for Oil-Storage”





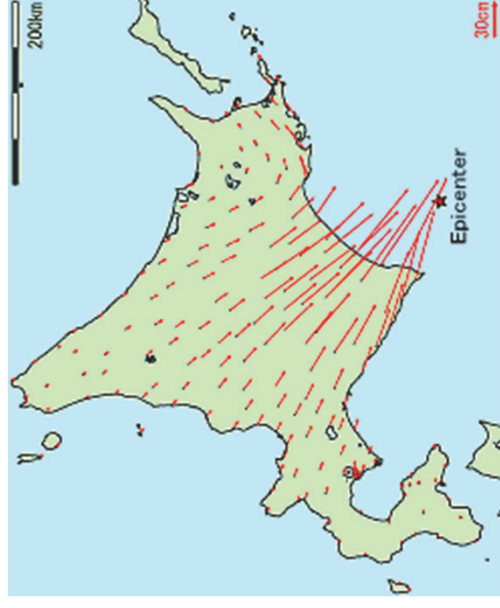
# Target Application

- Coupling between “Ground Motion” and “Sloshing of Tanks for Oil-Storage”
  - “One-way” coupling from “Ground Motion” to “Tanks” .
  - Displacement of ground surface is given as forced displacement of bottom surface of tanks.
  - 1 Tank = 1 PE (serial)



# 2003 Tokachi Earthquake (M8.0)

Fire accident of oil tanks due to long period ground motion (surface waves) developed in the basin of Tomakomai

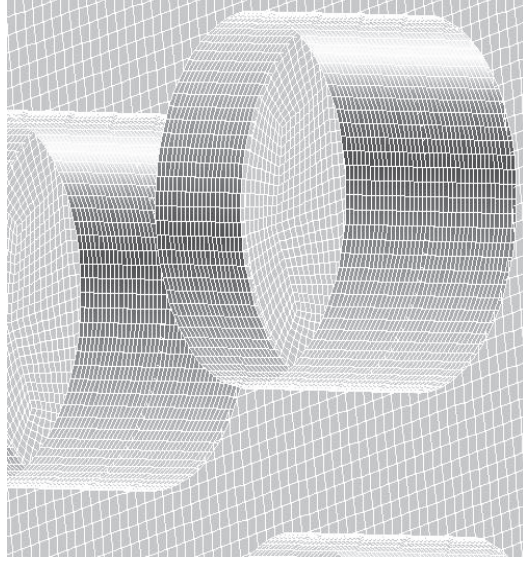
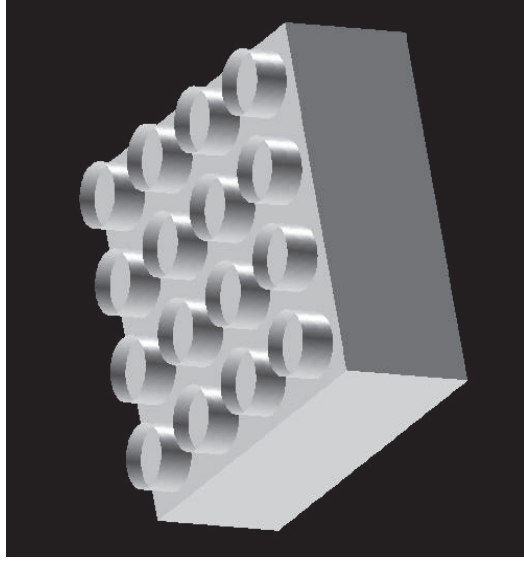




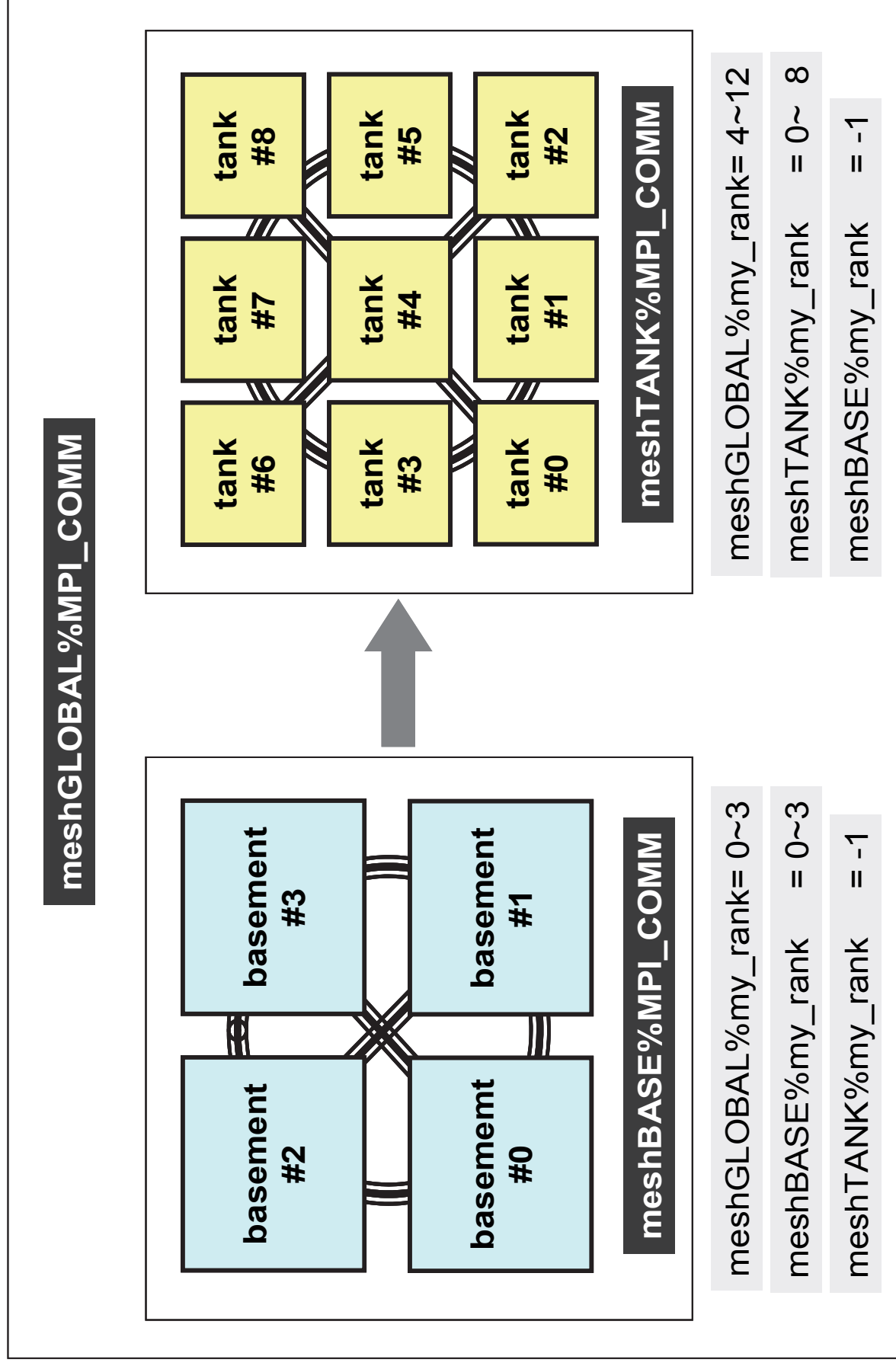
The diagram illustrates the propagation of seismic waves through various geological layers. At the top, a red box highlights the 'Sedimentary Layers' with a shear wave velocity  $V_s \sim 1,000 \sim 2,000$  m/sec. Below this, a yellow box represents the 'Bedrock' with  $V_s \sim 3,000$  m/sec. The bottom section shows the 'Earthquake' source, with arrows indicating the direction of wave propagation. A red box labeled 'Wave Propagation' encompasses the layers through which the waves travel. A blue box on the left indicates the 'Surface (slower)' layer with  $V_s \sim 400 \sim 700$  m/sec. A scale bar at the bottom left shows a distance of 2 cm/s and a time scale of 100 s. A legend at the bottom right identifies the 'P' (Pressure) and 'S' (Shear) wave components.

# Simulation Codes

- Ground Motion (Ichimura): Fortran
  - Parallel FEM, 3D Elastic/Dynamic
    - Explicit forward Euler scheme
  - Each element:  $2\text{m} \times 2\text{m} \times 2\text{m}$  cube
  - $240\text{m} \times 240\text{m} \times 100\text{m}$  region
- Sloshing of Tanks (Nagashima): C
  - Serial FEM (Embarrassingly Parallel)
    - Implicit backward Euler, Skyline method
    - Shell elements + Inviscid potential flow
  - D:  $42.7\text{m}$ , H:  $24.9\text{m}$ , T:  $20\text{mm}$ ,
  - Frequency:  $7.6\text{sec}$ .
  - 80 elements in circ.,  $0.6\text{m}$  mesh in height
  - Tank-to-Tank:  $60\text{m}$ ,  $4 \times 4$
- Total number of unknowns: 2,918,169



# Three Communicators



# MPI\_COMM\_RANK

- Determines the rank of the calling process in the communicator
  - “ID of MPI process” is sometimes called “rank”
- **MPI\_COMM\_RANK (comm, rank, ierr)**
  - comm      I      I      communicator
  - rank      I      O      rank of the calling process in the group of comm
  - ierr      I      O      **Starting from “0”**  
completion code

```
implicit REAL*8 (A-H,O-Z)
include 'mpif.h'
integer :: PETOT, my_rank, ierr

call MPI_INIT (ierr)
call MPI_COMM_SIZE (MPI_COMM_WORLD, PETOT, ierr )
call MPI_COMM_RANK (MPI_COMM_WORLD, my_rank, ierr )

write (*,'(a,2i8)') 'Hello World FORTRAN', my_rank, PETOT

call MPI_FINALIZE (ierr)
```

```
stop
end
```



# MPI\_ABORT

- Aborts MPI execution environment
- call **MPI\_ABORT (comm, errcode, ierr)**
  - comm      I      I      communication
  - errcode    I      O      error code
  - ierr        I      O      completion code

# MPI\_WTIME

- Returns an elapsed time on the calling processor

- **time= MPI\_WTIME ()**

- time      R8      0

Time in seconds since an arbitrary time in the past.

```
...
real(kind=8):: Stime, Etime

Stime= MPI_WTIME ()
do i= 1, 1000000000
    a= 1.d0
enddo
Etime= MPI_WTIME ()

write (*, '(i5,1pe16.6)') my_rank, Etime-Stime
```

# Example of MPI\_wtime

```
$> cd <$0-$1>

$> mpifccpx -O1 time.c
$> mpifrtpx -O1 time.f

(modify go4.sh, 4 processes)
$> pjsub go4.sh
```

```
0 1.113281E+00
3 1.113281E+00
2 1.117188E+00
1 1.117188E+00
```

| Process<br>ID | Time |
|---------------|------|
|---------------|------|

# MPI\_Wtick

- Returns the resolution of MPI\_Wtime
- **depends on hardware, and compiler**

- **time= MPI\_Wtick ()**

– time      R8      0      Time in seconds of resolution of MPI\_Wtime

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

```
...
TM= MPI_WTICK ()
write (*,*) TM
...
```

```
double Time;
...
Time = MPI_Wtick();
printf ("%5d%16.6E\n", MyRank, Time);
...
```

# Example of MPI\_wtick

```
$> cd <$0-$1>

$> mpifccpx -O1 wtick.c
$> mpifrtpx -O1 wtick.f

(modify go1.sh, 1 process)
$> pjsub go1.sh
```

# MPI\_BARRIER

- Blocks until all processes in the communicator have reached this routine.
- Mainly for debugging, huge overhead, not recommended for real code.
- call **MPI\_BARRIER (comm, ierr)**
  - comm      I      I      communicator
  - ierr      I      O      completion code

- What is MPI ?
- Your First MPI Program: Hello World
- **Global/Local Data**
- Collective Communication
- Peer-to-Peer Communication

# Data Structures & Algorithms

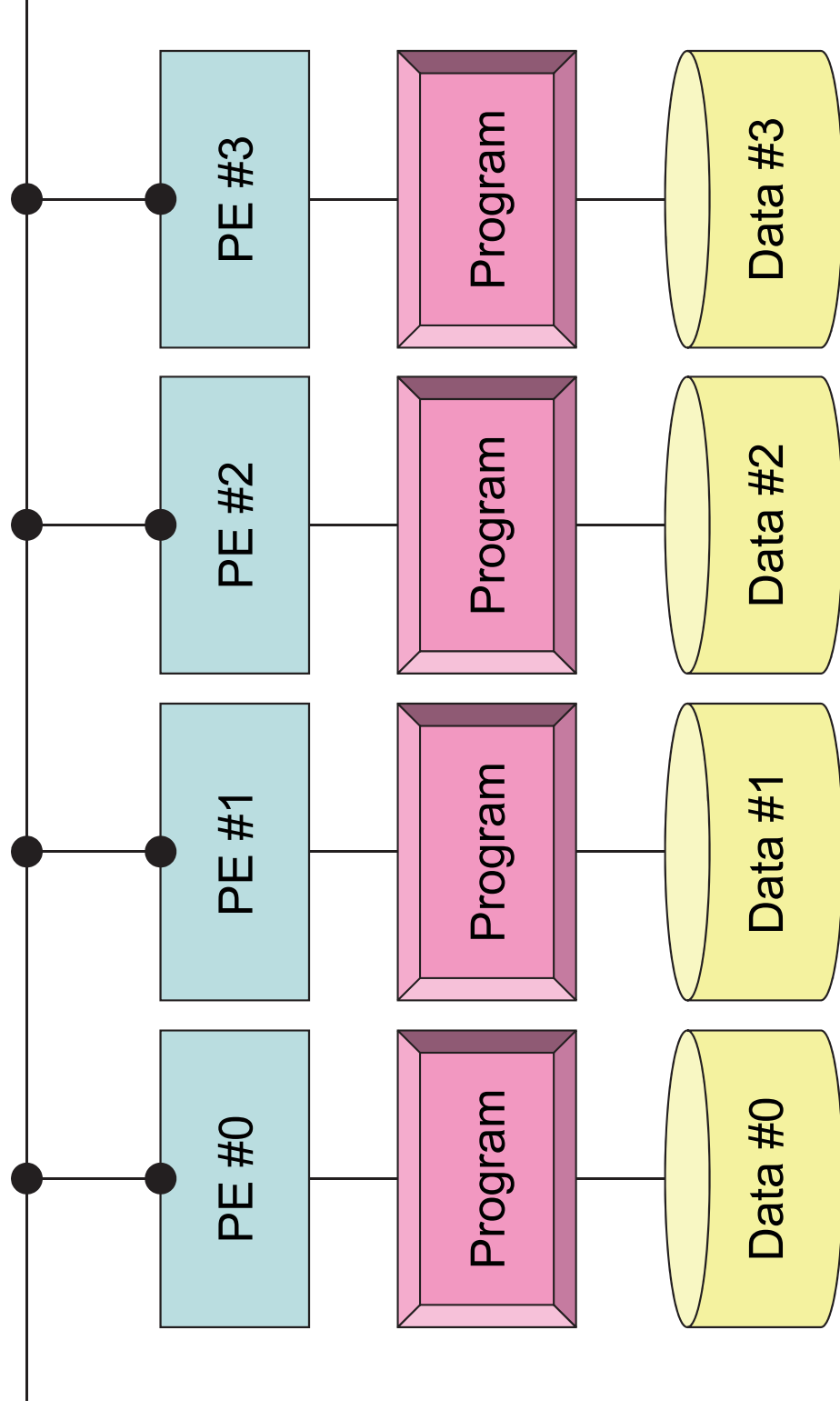
- Computer program consists of data structures and algorithms.
- They are closely related. In order to implement an algorithm, we need to specify an appropriate data structure for that.
  - We can even say that “Data Structures=Algorithms”
  - Some people may not agree with this, I (KN) think it is true for scientific computations from my experiences.
- Appropriate data structures for parallel computing must be specified before starting parallel computing.



# SPMD: Single Program Multiple Data

- There are various types of “*parallel computing*”, and there are many algorithms.
- Common issue is SPMD (Single Program Multiple Data).
- It is ideal that parallel computing is done in the same way for serial computing (except communications)
  - It is required to specify processes with communications and those without communications.

# What is a data structure which is appropriate for SPMD ?



# Data Structure for SMPD (1/2)

- SPMD: Large data is decomposed into small pieces, and each piece is processed by each processor/process
- Consider the following simple computation for vector **Vg** with length of **Ng** (=20):

```
integer, parameter :: NG= 20
real(kind=8), dimension(20) :: VG

do i= 1, NG
    VG(i) = 2.0 * VG(i)
enddo
```

- If you compute this using four processors, each processor stores and processes 5 (=20/4) components of **Vg**.

# Data Structure for SMPD (2/2)

- i.e.

```
integer, parameter :: NL= 5
real(kind=8) , dimension(5) :: VL

do i= 1, NL
    VL(i) = 2.0 * VL(i)
enddo
```

- Thus, a “single program” can execute parallel processing.
  - In each process, components of “VL” are different: Multiple Data
  - Computation using only “VL” (as long as possible) leads to efficient parallel computation.
  - Program is not different from that for serial CPU (in the previous page).

# Global & Local Data

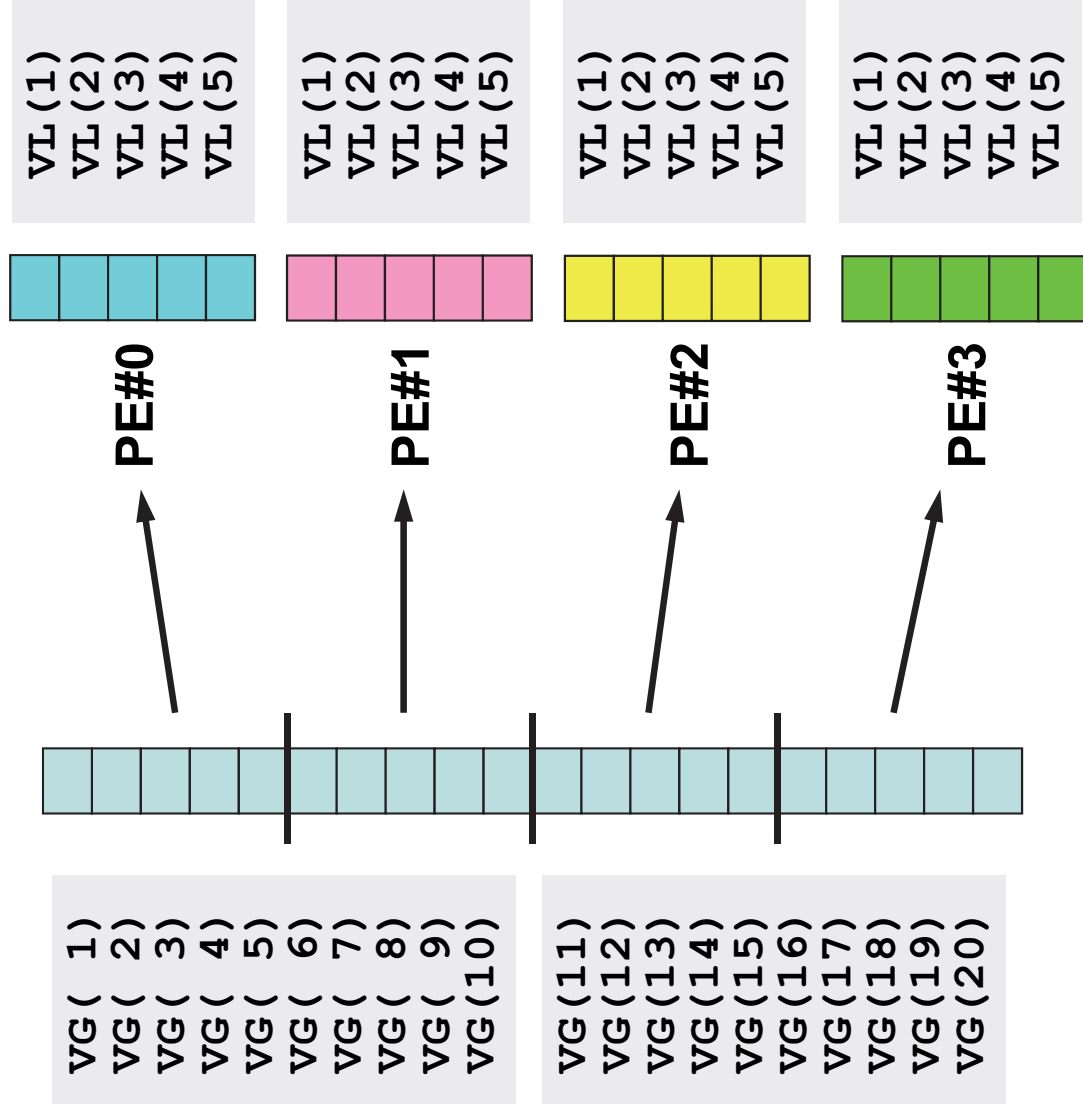
- $V_g$ 
  - Entire Domain
  - “Global Data” with “Global ID” from 1 to 20
- $V_l$ 
  - for Each Process (PE, Processor, Domain)
  - “Local Data” with “Local ID” from 1 to 5
  - Efficient utilization of local data leads to excellent parallel efficiency.

# Idea of Local Data

## Vg: Global Data

- 1<sup>st</sup>-5<sup>th</sup> comp. on PE#0
- 6<sup>th</sup>-10<sup>th</sup> comp. on PE#1
- 11<sup>th</sup>-15<sup>th</sup> comp. on PE#2
- 16<sup>th</sup>-20<sup>th</sup> comp. on PE#3

Each of these four sets corresponds to 1<sup>st</sup>-5<sup>th</sup> components of **VL** (local data) where there local ID's are 1-5.



# Global & Local Data

- **Vg**
  - Entire Domain
  - “Global Data” with “Global ID” from 1 to 20
- **VI**
  - for Each Process (PE, Processor, Domain)
  - “Local Data” with “Local ID” from 1 to 5
- **Please keep your attention to the following:**
  - How to generate VI (local data) from Vg (global data)
  - How to map components, from Vg to VI, and from VI to Vg.
  - What to do if VI cannot be calculated on each process in independent manner.
- Processing as localized as possible leads to excellent parallel efficiency:
  - Data structures & algorithms for that purpose.

- What is MPI ?
- Your First MPI Program: Hello World
- Global/Local Data
- **Collective Communication**
- Peer-to-Peer Communication



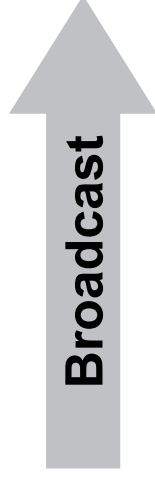
# What is Collective Communication ?

## 集団通信, グループ通信

- Collective communication is the process of exchanging information between multiple MPI processes in the communicator: one-to-all or all-to-all communications.
- Examples
  - Broadcasting control data
  - Max, Min
  - Summation
  - Dot products of vectors
  - Transformation of dense matrices

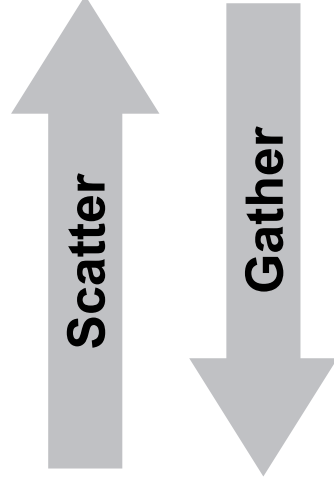
# Example of Collective Communications (1/4)

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



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

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



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

# Example of Collective Communications (2/4)

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

**All gather**

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

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

**All-to-All**

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

# Example of Collective Communications (3/4)

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



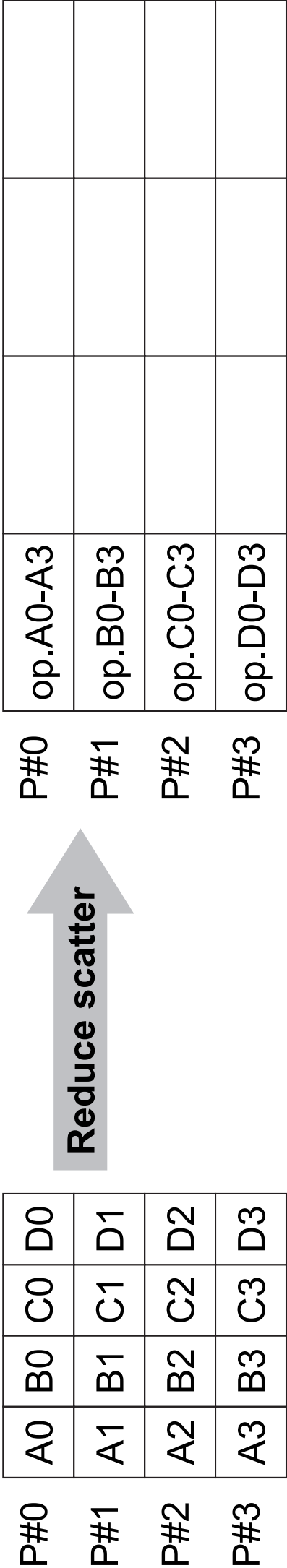
|     |          |          |          |          |
|-----|----------|----------|----------|----------|
| P#0 | op.A0-A3 | op.B0-B3 | op.C0-C3 | op.D0-D3 |
| P#1 |          |          |          |          |
| P#2 |          |          |          |          |
| P#3 |          |          |          |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

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



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

# Example of Collective Communications (4/4)



# Examples by Collective Comm.

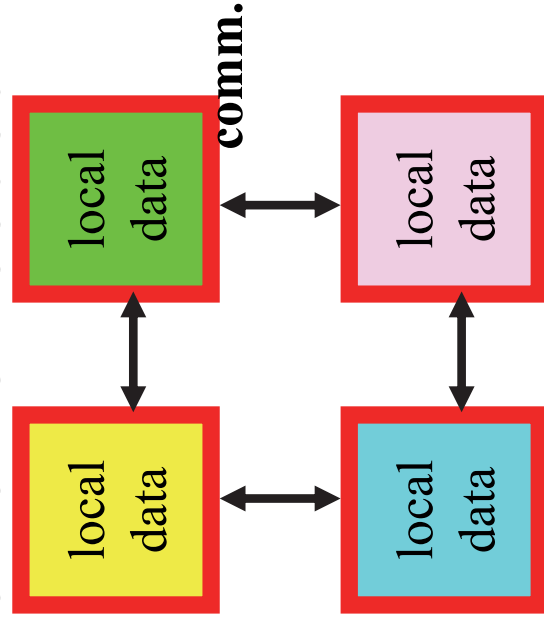
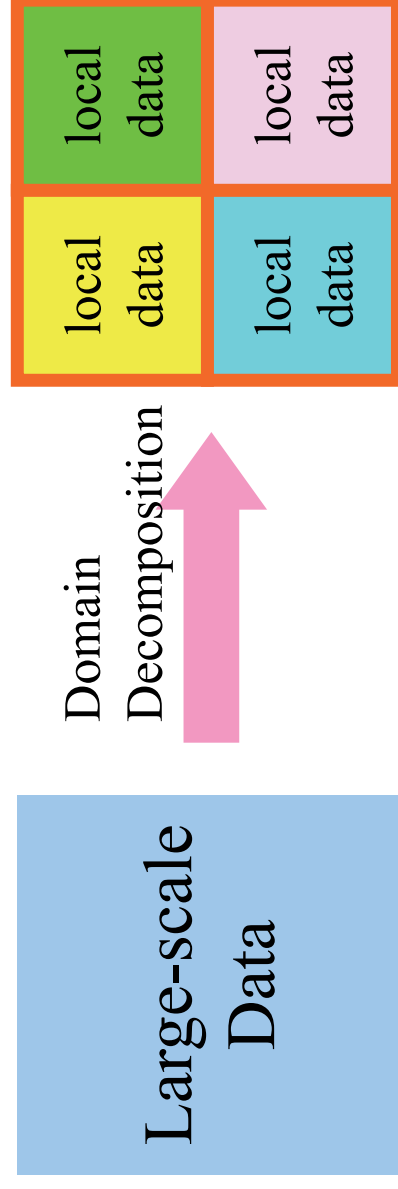
- Dot Products of Vectors
- Scatter/Gather
- Reading Distributed Files
- MPI\_Allgatherv

# Global/Local Data

- Data structure of parallel computing based on SPMD, where large scale “global data” is decomposed to small pieces of “local data” .

# Domain Decomposition/Partitioning

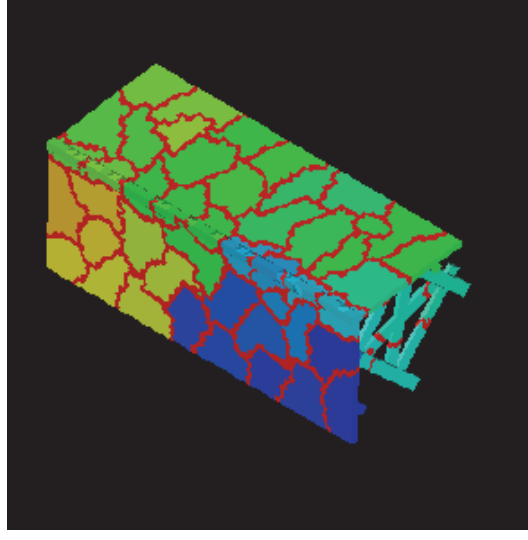
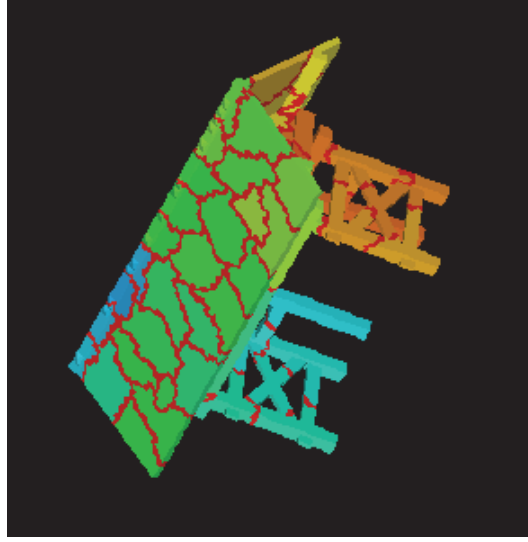
- PC with 1GB RAM: can execute FEM application with up to  $10^6$  meshes
  - $10^3\text{km} \times 10^3\text{km} \times 10^2\text{km}$  (SW Japan):  $10^8$  meshes by 1km cubes
- Large-scale Data: Domain decomposition, parallel & local operations
- Global Computation: Comm. among domains needed





# Local Data Structure

- It is important to define proper local data structure for target computation (and its algorithm)
  - Algorithms= Data Structures
- Main objective of this class !



# Global/Local Data

- Data structure of parallel computing based on SPMD, where large scale “global data” is decomposed to small pieces of “local data” .
- Consider the dot product of following VECp and VECs with length=20 by parallel computation using 4 processors

```

VECp ( 1) = 2
      ( 2) = 2
      ( 3) = 2
...
      (18) = 2
      (19) = 2
      (20) = 2

```

```

VECs ( 1) = 3
      ( 2) = 3
      ( 3) = 3
...
      (18) = 3
      (19) = 3
      (20) = 3

```

```

VECp [ 0] = 2
      [ 1] = 2
      [ 2] = 2
...
      [17] = 2
      [18] = 2
      [19] = 2

```

```

VECs [ 0] = 3
      [ 1] = 3
      [ 2] = 3
...
      [17] = 3
      [18] = 3
      [19] = 3

```

# <\$O-\$1>/dot.f, dot.c

```

implicit REAL*8 (A-H,O-Z)
real(kind=8),dimension(20):: &
    VECp, VECs

do i= 1, 20
    VECp(i)= 2.0d0
    VECs(i)= 3.0d0
enddo

sum= 0.d0
do ii= 1, 20
    sum= sum + VECp(ii)*VECs(ii)
enddo

stop
end

```

```

#include <stdio.h>
int main() {
    int i;
    double VECp[20], VECs[20]
    double sum;

    for(i=0;i<20;i++){
        VECp[i]= 2.0;
        VECs[i]= 3.0;
    }

    sum = 0.0;
    for(i=0;i<20;i++){
        sum += VECp[i] * VECs[i];
    }
    return 0;
}

```

# <\$O-\$1>/dot.f, dot.c

```
>$ cd <$O-$1>
```

```
>$ cc -O3 dot.c
```

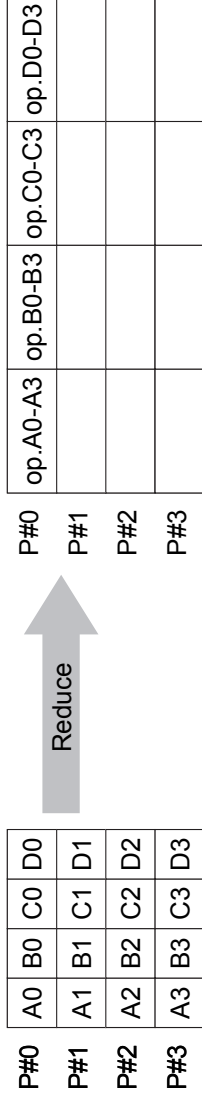
```
>$ f95 -O3 dot.f
```

```
>$ ./a.out
```

|     |    |    |
|-----|----|----|
| 1   | 2. | 3. |
| 2   | 2. | 3. |
| 3   | 2. | 3. |
| ... |    |    |
| 18  | 2. | 3. |
| 19  | 2. | 3. |
| 20  | 2. | 3. |

**dot product      120.**

# 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
  - count I I number of elements in send/receive buffer
  - datatype I I data type of elements of send/receive 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
  - root I I rank of root process
  - comm I I communicator
  - ierr I O completion code

Users can define operations by [MPI\\_OP\\_CREATE](#)

# 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)
```

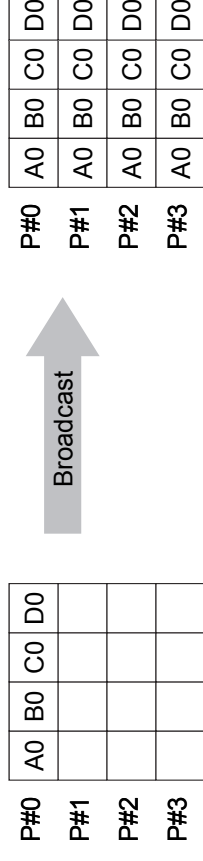
Global summation of X0 goes to XSUM on #0 process.

```
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.

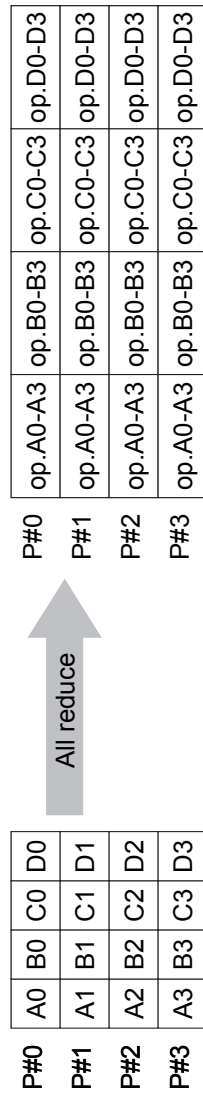


# MPI\_BCAST



- 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

# MPI\_ALLREDUCE



- **MPI\_Reduce + MPI\_Bcast**
- **Summation (of dot products) and MAX/MIN values are likely to utilized in each process**

- **call MPI\_ALLREDUCE (sendbuf, recvbuf, count, datatype, op, comm, 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        communicator
- ierr        I        O        completion code

# “op” of MPI\_Reduce/Allreduce

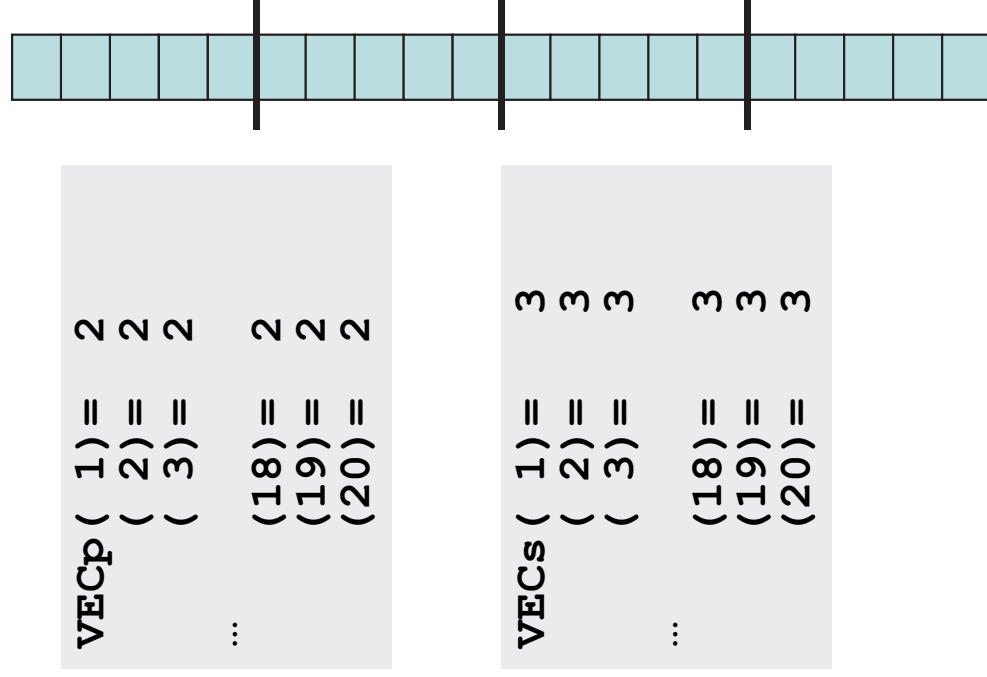
```
call MPI_REDUCE
(sendbuf,recvbuf,count,datatype,op,root,comm,ierr)
```

- MPI\_MAX, MPI\_MIN      Max, Min
- MPI\_SUM, MPI\_PROD      Summation, Product
- MPI\_LAND      Logical AND

# Local Data (1/2)

Fortran

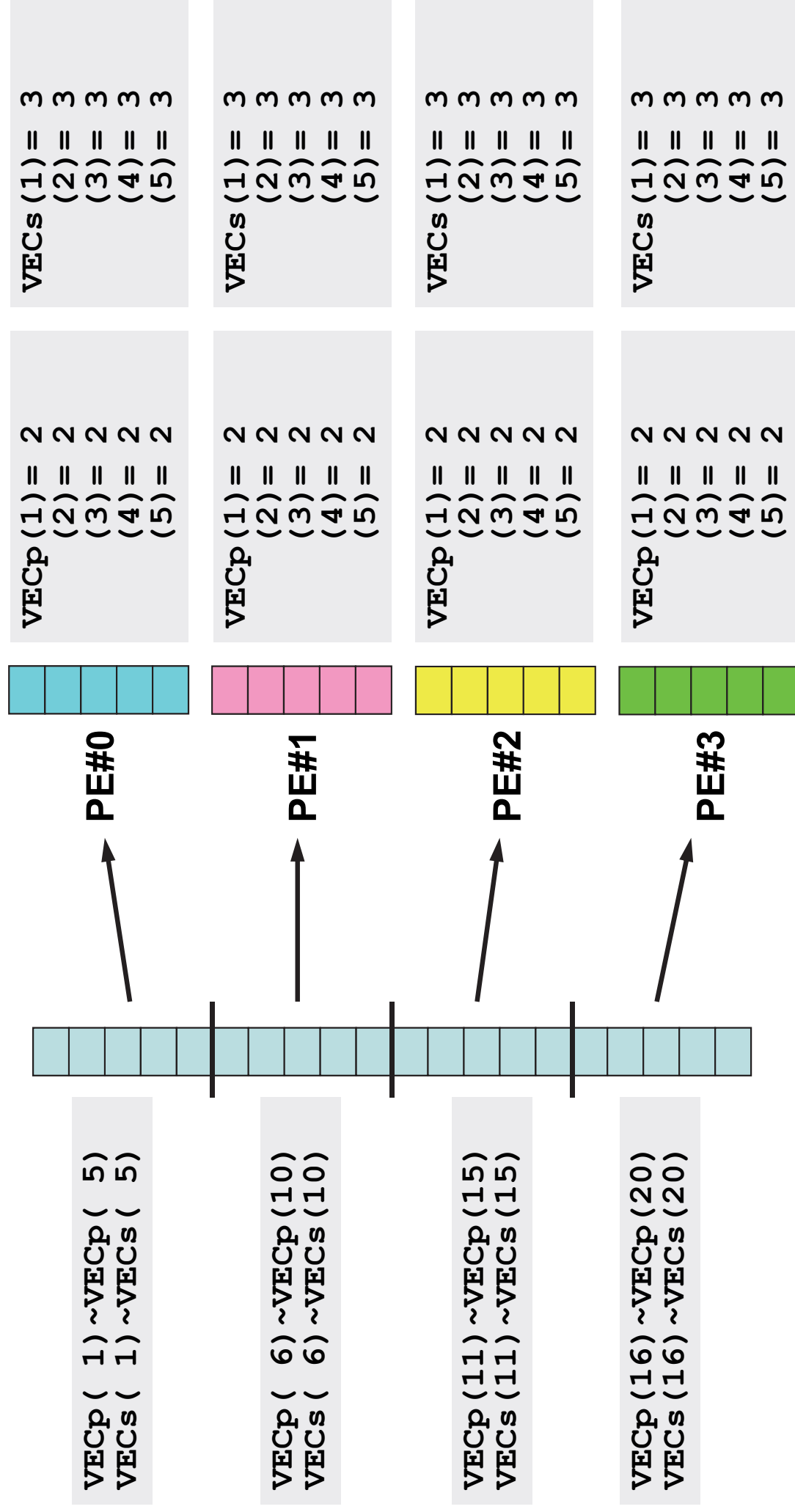
- Decompose vector with length=20 into 4 domains (processes)
- Each process handles a vector with length= 5



# Local Data (2/2)

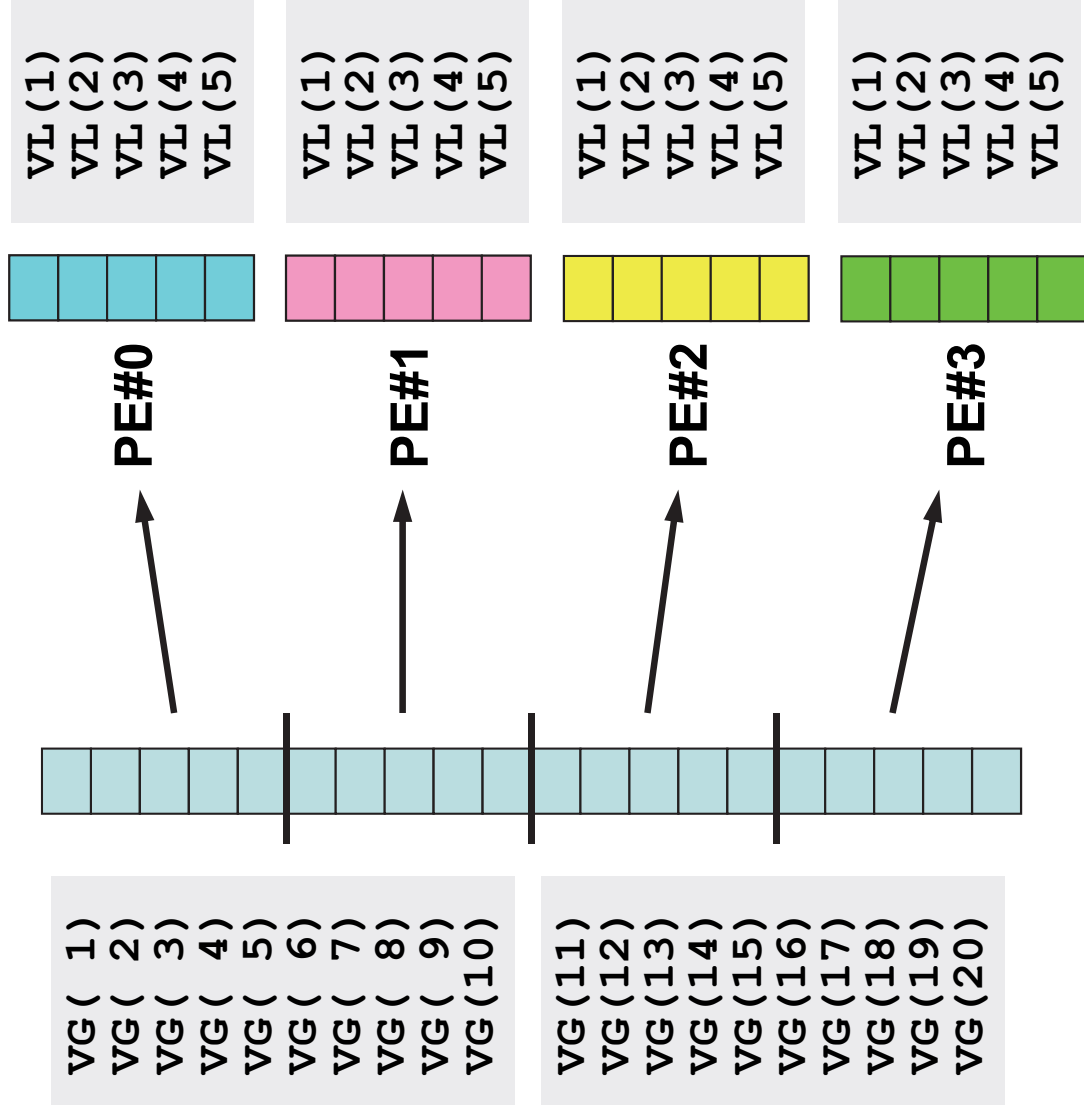
Fortran

- 1<sup>th</sup>-5<sup>th</sup> components of original global vector go to 1<sup>th</sup>-5<sup>th</sup> components of PE#0, 6<sup>th</sup>-10<sup>th</sup> -> PE#1, 11<sup>th</sup>-15<sup>th</sup> -> PE#2, 16<sup>th</sup>-20<sup>th</sup> -> PE#3.



# But ...

- It is too easy !! Just decomposing and renumbering from 1 (or 0).
- Of course, this is not enough. Further examples will be shown in the latter part.



# Example: Dot Product (1/3)

**<\$O-S1>/allreduce.f**

```
implicit REAL*8 (A-H,O-Z)
include 'mpif.h'
integer :: PETOT, my_rank, ierr
real(kind=8), dimension(5) :: VECp, VECs

call MPI_INIT      (ierr)
call MPI_COMM_SIZE (MPI_COMM_WORLD, PETOT, ierr )
call MPI_COMM_RANK (MPI_COMM_WORLD, my_rank, ierr )
```

```
sumA= 0.d0
sumR= 0.d0
do i= 1, 5
  VECp(i)= 2.d0
  VECs(i)= 3.d0
enddo
```

Local vector is generated  
at each local process.

```
sum0= 0.d0
do i= 1, 5
  sum0= sum0 + VECp(i) * VECs(i)
enddo
```

```
if (my_rank.eq.0) then
  write (*,'(a)') '(my_rank, sumALLREDUCE, sumREDUCE) '
endif
```

# Example: Dot Product (2/3)

`<$0-$1>/allreduce.f`

```
!C
!C-- REDUCE
    call MPI_REDUCE (sum0, sumR, 1, MPI_DOUBLE_PRECISION, MPI_SUM, 0, &
        MPI_COMM_WORLD, ierr)

!C
!C-- ALL-REDUCE
    call MPI_Allreduce (sum0, sumA, 1, MPI_DOUBLE_PRECISION, MPI_SUM, &
        MPI_COMM_WORLD, ierr)

write (*, '(a,i5, 2(1pe16.6))') 'before BCAST', my_rank, sumA, sumR
```

## Dot Product

Summation of results of each process (sum0)

“sumR” has value only on PE#0.

“sumA” has value on all processes by MPI\_Allreduce



# Example: Dot Product (3/3)

```
<$O-S1>/allreduce.f
```

```
!C
!C-- BCAST
  call MPI_BCAST (sumR, 1, MPI_DOUBLE_PRECISION, 0, MPI_COMM_WORLD, &
    ierr)
  write (*, '(a,i5, 2(1pe16.6))') 'after BCAST', my_rank, sumA, sumR

  call MPI_FINALIZE (ierr)

  stop
end
```

“sumR” has value on PE#1-#3 by MPI\_Bcast

# Execute `<$O-S1>/allreduce.f/c`

```
$> mpifccpx -Kfast allreduce.c
$> mpifrtpx -Kfast allreduce.f
(modify go4.sh, 4 process)
$> pjsub go4.sh
```

```
(my_rank, sumALLREDUCE, sumREDUCE)
before BCAST      0      1.200000E+02      1.200000E+02
after  BCAST      0      1.200000E+02      1.200000E+02

before BCAST      1      1.200000E+02      0.000000E+00
after  BCAST      1      1.200000E+02      1.200000E+02

before BCAST      3      1.200000E+02      0.000000E+00
after  BCAST      3      1.200000E+02      1.200000E+02

before BCAST      2      1.200000E+02      0.000000E+00
after  BCAST      2      1.200000E+02      1.200000E+02
```

# Examples by Collective Comm.

- Dot Products of Vectors
- Scatter/Gather
- Reading Distributed Files
- MPI\_Allgatherv

# Global/Local Data (1/3)

- Parallelization of an easy process where a real number  $\alpha$  is added to each component of real vector **VECg**:

```
do i= 1, NG  
    VECg(i)= VECg(i) + ALPHA  
enddo
```

```
for (i=0; i<NG; i++){  
    VECg[i]= VECg[i] + ALPHA  
}
```

# Global/Local Data (2/3)

- Configuration
  - NG= 32 (length of the vector)
  - ALPHA=1000.
  - Process # of MPI= 4
- Vector VECg has following 32 components  
(<\$T-S1>/a1x.all):

|                                                          |
|----------------------------------------------------------|
| (101.0, 103.0, 105.0, 106.0, 109.0, 111.0, 121.0, 151.0, |
| 201.0, 203.0, 205.0, 206.0, 209.0, 211.0, 221.0, 251.0,  |
| 301.0, 303.0, 305.0, 306.0, 309.0, 311.0, 321.0, 351.0,  |
| 401.0, 403.0, 405.0, 406.0, 409.0, 411.0, 421.0, 451.0)  |

# Global/Local Data (3/3)

- Procedure
  - ① Reading vector **VECG** with length=32 from one process (e.g. 0<sup>th</sup> process)
    - Global Data
  - ② Distributing vector components to 4 MPI processes equally (*i.e.* length= 8 for each processes)
    - Local Data, Local ID/Numbering
  - ③ Adding **ALPHA** to each component of the local vector (with length= 8) on each process.
  - ④ Merging the results to global vector with length= 32.
- Actually, we do not need parallel computers for such a kind of small computation.

# Operations of Scatter/Gather (1/8)

Reading VECg (length=32) from a process (e.g. #0)

- Reading global data from #0 process

```
include 'mpif.h'
integer, parameter :: NG= 32
real(kind=8), dimension(NG) :: VECg

call MPI_INIT (ierr)
call MPI_COMM_SIZE (<comm>, PETOT , ierr)
call MPI_COMM_RANK (<comm>, my_rank, ierr)

if (my_rank.eq.0) then
  open (21, file= 'a1x.all', status= 'unknown')
  do i= 1, NG
    read (21,*) VECg(i)
  enddo
  close (21)
endif
```

```
#include <mpi.h>
#include <stdio.h>
#include <math.h>
#include <assert.h>

int main(int argc, char **argv) {
  int i, NG=32;
  int PeTot, MyRank, MPI_Comm;
  double VECg[32];
  char filename[80];
  FILE *fp;

  MPI_Init(&argc, &argv);
  MPI_Comm_size(<comm>, &PeTot);
  MPI_Comm_rank(<comm>, &MyRank);

  fp = fopen("a1x.all", "r");
  if (!MyRank) for (i=0; i<NG; i++) {
    fscanf(fp, "%lf", &VECg[i]);
  }
}
```

# Operations of Scatter/Gather (2/8)

Distributing global data to 4 process equally (*i.e.* length=8 for each process)

- MPI\_Scatter



The diagram illustrates two data distribution operations: Scatter and Gather.

**Scatter Operation:** A single source (represented by a large grey arrow pointing right) distributes data to multiple destinations. The destinations are labeled P#0, P#1, P#2, and P#3, each associated with a data block (A0, B0, C0, D0).

**Gather Operation:** Multiple sources (represented by a large grey arrow pointing left) collect data into a single destination. The sources are labeled P#0, P#1, P#2, and P#3, each associated with a data block (A0, B0, C0, D0).

- FORTRAN MPI INTEGER, MPI REAL, MPI DOUBLE PRECISION, MPI CHARACTER etc.

MPI INT, MPI FLOAT, MPI DOUBLE, MPI CHAR etc.

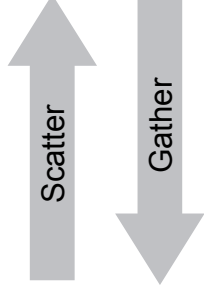
- |   |                 |        |   |                                                   |
|---|-----------------|--------|---|---------------------------------------------------|
| - | <u>recvbuf</u>  | choice | ○ | starting address of receiving buffer              |
| - | <u>rcount</u>   | I      | I | number of elements received from the root process |
| - | <u>recvtype</u> | I      | I | data type of elements of receiving buffer         |
| - | <u>root</u>     | I      | I | rank of root process                              |
| - | <u>comm</u>     | I      | I | communicator                                      |
| - | <u>ierr</u>     | I      | ○ | completion code                                   |

# MPI\_SCATTER

## (cont.)

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

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



- call `MPI_SCATTER (sendbuf, scount, sendtype, recvbuf, rcount, recvttype, root, comm, ierr)`
  - sendbuf choice I starting address of sending buffer  
**type is defined by "data type"**
  - scount I I number of elements sent to each process
  - sendtype I I data type of elements of sending buffer
  - recvbuf choice O starting address of receiving buffer
  - rcount I I number of elements received from the root process
  - recvttype I I data type of elements of receiving buffer
  - root I I **rank of root process**
  - comm I I communicator
  - ierr I O completion code

- Usually
  - **scount = rcount**
  - **sendtype= recvttype**
- This function sends scount components starting from sendbuf (sending buffer) at process #root to each process in comm. Each process receives rcount components starting from recvbuf (receiving buffer).

# Operations of Scatter/Gather (3/8)

Distributing global data to 4 processes equally

- Allocating receiving buffer **VEC** (length=8) at each process.
- 8 components sent from sending buffer **VECg** of process #0 are received at each process #0-#3 as 1<sup>st</sup>-8<sup>th</sup> components of receiving buffer **VEC**.

```
integer, parameter :: N = 8
real(kind=8), dimension(N) :: VEC
...
call MPI_Scatter
    (VECg, N, MPI_DOUBLE_PRECISION, &
    VEC, N, MPI_DOUBLE_PRECISION, &
    0, <comm>, ierr)
```

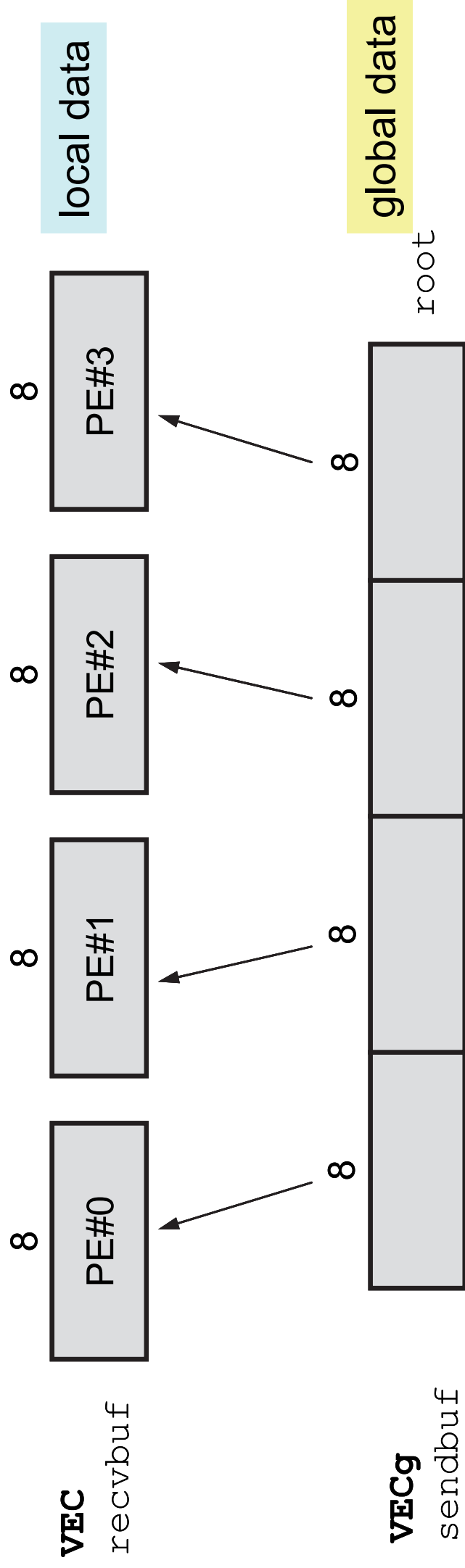
```
int N=8;
double VEC [8];
...
MPI_Scatter (VECg, N, MPI_DOUBLE, VEC, N,
MPI_DOUBLE, 0, <comm>);
```

**call MPI\_SCATTER**  
**(sendbuf, scout, sendtype, recvbuf, rcount,**  
**recvtype, root, comm, ierr)**

# Operations of Scatter/Gather (4/8)

Distributing global data to 4 processes equally

- 8 components are scattered to each process from root (#0)
- 1<sup>st</sup>-8<sup>th</sup> components of **VECg** are stored as 1<sup>st</sup>-8<sup>th</sup> ones of **VEC** at **#0**, 9<sup>th</sup>-16<sup>th</sup> components of **VECg** are stored as 1<sup>st</sup>-8<sup>th</sup> ones of **VEC** at **#1**, etc.
- **VECg**: Global Data, **VEC**: Local Data



# Operations of Scatter/Gather (5/8)

Distributing global data to 4 processes equally

- Global Data: 1<sup>st</sup>-32<sup>nd</sup> components of **VECg** at **#0**
- Local Data: 1<sup>st</sup>-8<sup>th</sup> components of **VEC** at each process
- Each component of **VEC** can be written from each process in the following way:

```
do i= 1, N
  write (*, ' (a, 2i8, f10.0)') 'before', my_rank, i, VEC(i)
enddo
```

```
for (i=0; i<N; i++) {
  printf("before %5d %5d %10.0F\n", MyRank, i+1, VEC[i]);}
```

# Operations of Scatter/Gather (5/8)

Distributing global data to 4 processes equally

- Global Data: 1<sup>st</sup>-32<sup>nd</sup> components of **VECg** at **#0**
- Local Data: 1<sup>st</sup>-8<sup>th</sup> components of **VEC** at each process
- Each component of **VEC** can be written from each process in the following way:

| <u>PE#0</u> | before | 0 | 1 | 101. |
|-------------|--------|---|---|------|
|             | before | 0 | 2 | 103. |
|             | before | 0 | 3 | 105. |
|             | before | 0 | 4 | 106. |
|             | before | 0 | 5 | 109. |
|             | before | 0 | 6 | 111. |
|             | before | 0 | 7 | 121. |
|             | before | 0 | 8 | 151. |

| <u>PE#1</u> | before | 1 | 1 | 201. |
|-------------|--------|---|---|------|
|             | before | 1 | 2 | 203. |
|             | before | 1 | 3 | 205. |
|             | before | 1 | 4 | 206. |
|             | before | 1 | 5 | 209. |
|             | before | 1 | 6 | 211. |
|             | before | 1 | 7 | 221. |
|             | before | 1 | 8 | 251. |

| <u>PE#2</u> | before | 2 | 1 | 301. |
|-------------|--------|---|---|------|
|             | before | 2 | 2 | 303. |
|             | before | 2 | 3 | 305. |
|             | before | 2 | 4 | 306. |
|             | before | 2 | 5 | 309. |
|             | before | 2 | 6 | 311. |
|             | before | 2 | 7 | 321. |
|             | before | 2 | 8 | 351. |

| <u>PE#3</u> | before | 3 | 1 | 401. |
|-------------|--------|---|---|------|
|             | before | 3 | 2 | 403. |
|             | before | 3 | 3 | 405. |
|             | before | 3 | 4 | 406. |
|             | before | 3 | 5 | 409. |
|             | before | 3 | 6 | 411. |
|             | before | 3 | 7 | 421. |
|             | before | 3 | 8 | 451. |

# Operations of Scatter/Gather (6/8)

On each process, **ALPHA** is added to each of 8 components of **VEC**

- On each process, computation is in the following way

```
real(kind=8), parameter :: ALPHA= 1000.
do i= 1, N
  VEC(i)= VEC(i) + ALPHA
enddo
```

```
double ALPHA=1000.;
...
for (i=0; i<N; i++) {
  VEC[i]= VEC[i] + ALPHA;}
```

- Results:

| <u>PE#0</u> | after 0 1 | 1101. |
|-------------|-----------|-------|
|             | after 0 2 | 1103. |
|             | after 0 3 | 1105. |
|             | after 0 4 | 1106. |
|             | after 0 5 | 1109. |
|             | after 0 6 | 1111. |
|             | after 0 7 | 1121. |
|             | after 0 8 | 1151. |

| <u>PE#1</u> | after 1 1 | 1201. |
|-------------|-----------|-------|
|             | after 1 2 | 1203. |
|             | after 1 3 | 1205. |
|             | after 1 4 | 1206. |
|             | after 1 5 | 1209. |
|             | after 1 6 | 1211. |
|             | after 1 7 | 1221. |
|             | after 1 8 | 1251. |

| <u>PE#2</u> | after 2 1 | 1301. |
|-------------|-----------|-------|
|             | after 2 2 | 1303. |
|             | after 2 3 | 1305. |
|             | after 2 4 | 1306. |
|             | after 2 5 | 1309. |
|             | after 2 6 | 1311. |
|             | after 2 7 | 1321. |
|             | after 2 8 | 1351. |

| <u>PE#3</u> | after 3 1 | 1401. |
|-------------|-----------|-------|
|             | after 3 2 | 1403. |
|             | after 3 3 | 1405. |
|             | after 3 4 | 1406. |
|             | after 3 5 | 1409. |
|             | after 3 6 | 1411. |
|             | after 3 7 | 1421. |
|             | after 3 8 | 1451. |

# Operations of Scatter/Gather (7/8)

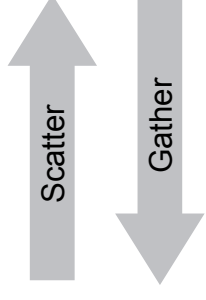
Merging the results to global vector with length= 32

- Using MPI\_Gather (inverse operation to MPI\_Scatter)



# MPI\_GATHER

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



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

- Gathers together values from a group of processes, inverse operation to MPI\_Scatter
- call `MPI_GATHER (sendbuf, scount, sendtype, recvbuf, rcount, recvttype, root, comm, ierr)`
  - sendbuf choice I starting address of sending buffer
  - scount I I number of elements sent to each process
  - sendtype I I data type of elements of sending buffer
  - recvbuf choice O starting address of receiving buffer
  - rcount I I number of elements received from the root process
  - recvttype I I data type of elements of receiving buffer
  - root I I **rank of root process**
  - comm I I communicator
  - ierr I O completion code
- recvbuf is on root process.

# Operations of Scatter/Gather (8/8)

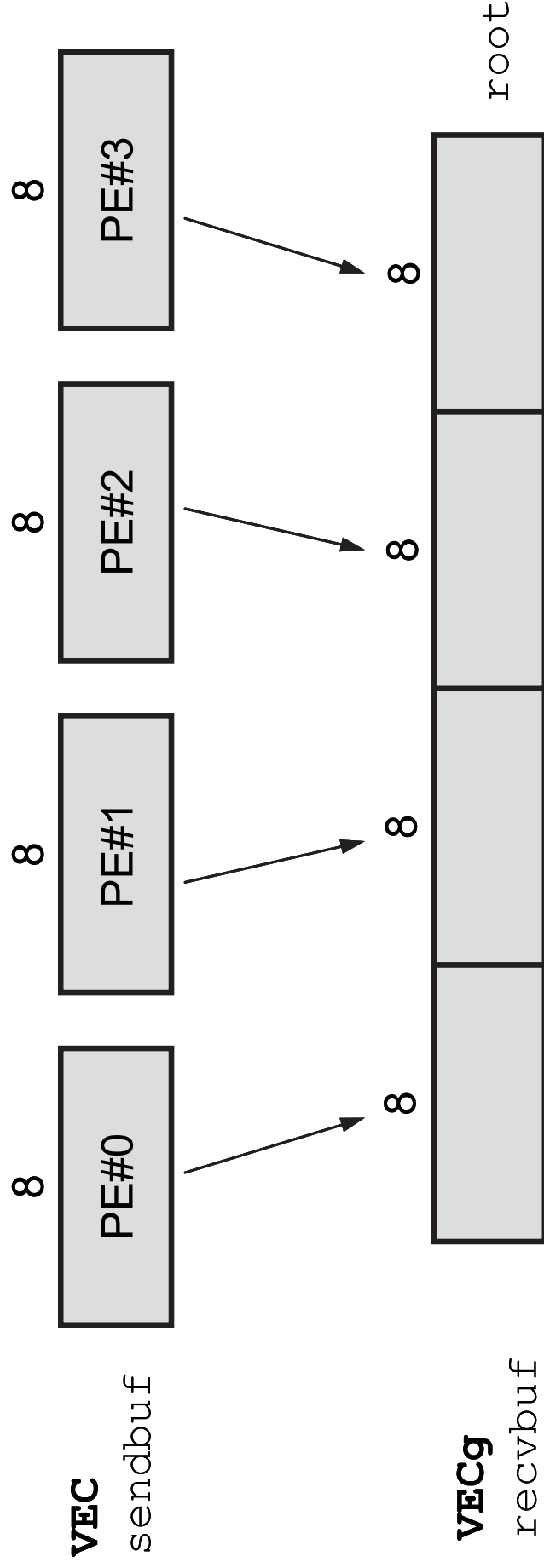
Merging the results to global vector with length= 32

- Each process components of **VEC** to **VECg** on root (#0 in this case).

```
call MPI_Gather
  (VEC , N, MPI_DOUBLE_PRECISION, &
   VECg, N, MPI_DOUBLE_PRECISION, &
   0, <comm>, ierr)
```

```
MPI_Gather (VEC, N, MPI_DOUBLE, VECg, N,
MPI_DOUBLE, 0, <comm>);
```

- 8 components are gathered from each process to the root process.



# <\$O-S1>/scatter-gather.f/c

## example

```
$> mpifccpx -Kfast scatter-gather.c
$> mpifrtpx -Kfast scatter-gather.f
$> (exec.4 proc's) go4.sh
```

PE#0  
before 0 1 101.  
before 0 2 103.  
before 0 3 105.  
before 0 4 106.  
before 0 5 109.  
before 0 6 111.  
before 0 7 121.  
before 0 8 151.

PE#1  
before 1 1 201.  
before 1 2 203.  
before 1 3 205.  
before 1 4 206.  
before 1 5 209.  
before 1 6 211.  
before 1 7 221.  
before 1 8 251.

PE#2  
before 2 1 301.  
before 2 2 303.  
before 2 3 305.  
before 2 4 306.  
before 2 5 309.  
before 2 6 311.  
before 2 7 321.  
before 2 8 351.

PE#3  
before 3 1 401.  
before 3 2 403.  
before 3 3 405.  
before 3 4 406.  
before 3 5 409.  
before 3 6 411.  
before 3 7 421.  
before 3 8 451.

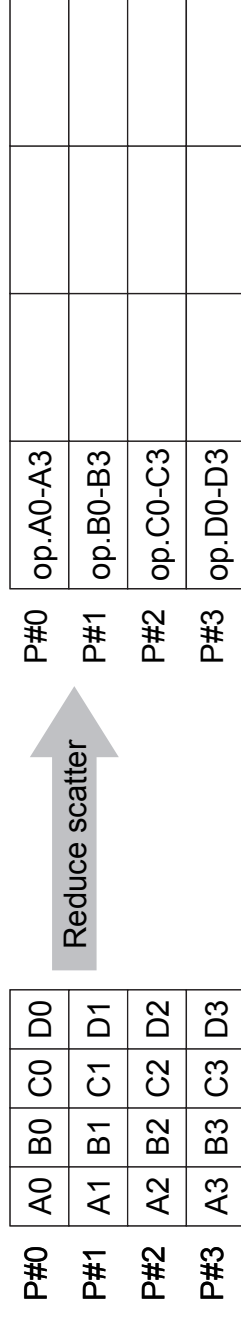
PE#0  
after 0 1 1101.  
after 0 2 1103.  
after 0 3 1105.  
after 0 4 1106.  
after 0 5 1109.  
after 0 6 1111.  
after 0 7 1121.  
after 0 8 1151.

PE#1  
after 1 1 1201.  
after 1 2 1203.  
after 1 3 1205.  
after 1 4 1206.  
after 1 5 1209.  
after 1 6 1211.  
after 1 7 1221.  
after 1 8 1251.

PE#2  
after 2 1 1301.  
after 2 2 1303.  
after 2 3 1305.  
after 2 4 1306.  
after 2 5 1309.  
after 2 6 1311.  
after 2 7 1321.  
after 2 8 1351.

PE#3  
after 3 1 1401.  
after 3 2 1403.  
after 3 3 1405.  
after 3 4 1406.  
after 3 5 1409.  
after 3 6 1411.  
after 3 7 1421.  
after 3 8 1451.

# MPI\_REDUCE\_SCATTER



- MPI\_REDUCE + MPI\_SCATTER
- call **MPI\_REDUCE\_SCATTER (sendbuf, recvbuf, rcount, datatype, op, comm, ierr)**
  - sendbuf choice I starting address of sending buffer
  - recvbuf choice O starting address of receiving buffer
  - rcount I I integer array specifying the number of elements in result distributed to each process. Array must be identical on all calling processes.
  - datatype I I data type of elements of sending/receiving buffer
  - op I I reduce operation
  - comm I I communicator
  - ierr I O completion code

# MPI\_ALLGATHER

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



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

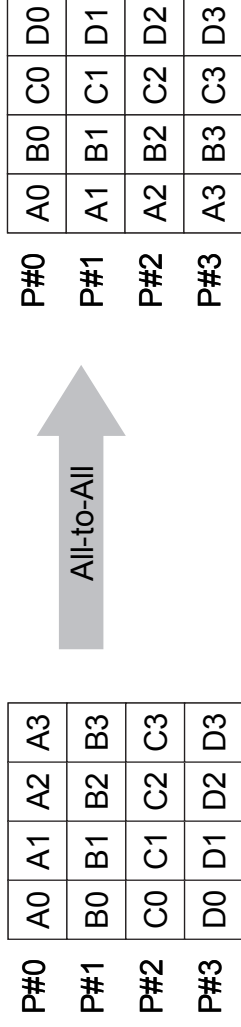
- MPI\_GATHER+MPI\_BCAST

- Gathers data from all tasks and distribute the combined data to all tasks

- call **MPI\_ALLGATHER (sendbuf, scount, sendtype, recvbuf, rcount, recvttype, comm, ierr)**

- sendbuf choice I starting address of sending buffer
- scount I I number of elements sent to each process
- sendtype I I data type of elements of sending buffer
- recvbuf choice O starting address of receiving buffer
- rcount I I number of elements received from each process
- recvttype I I data type of elements of receiving buffer
- comm I I communicator
- ierr I O completion code

# MPI\_ALLTOALL



- Sends data from all to all processes: transformation of dense matrix
- call **MPI\_ALLTOALL** (**sendbuf**, **scount**, **sendtype**, **recvbuf**, **rcount**, **recvtype**, **comm**, **ierr**)
  - sendbuf choice I starting address of sending buffer
  - scount I I number of elements sent to each process
  - sendtype I I data type of elements of sending buffer
  - recvbuf choice O starting address of receiving buffer
  - rcount I I number of elements received from each process
  - recvtype I I data type of elements of receiving buffer
  - comm I I communicator
  - ierr I O completion code

# Examples by Collective Comm.

- Dot Products of Vectors
- Scatter/Gather
- Reading Distributed Files
- MPI\_Allgatherv

# Operations of Distributed Local Files

- In Scatter/Gather example, PE#0 reads global data, that is *scattered* to each processor, then parallel operations are done.
- If the problem size is very large, a single processor may not read entire global data.
  - If the entire global data is decomposed to distributed local data sets, each process can read the local data.
  - If global operations are needed to a certain sets of vectors, MPI functions, such as MPI\_Gather etc. are available.



# Reading Distributed Local Files:

## Uniform Vec. Length (1/2)

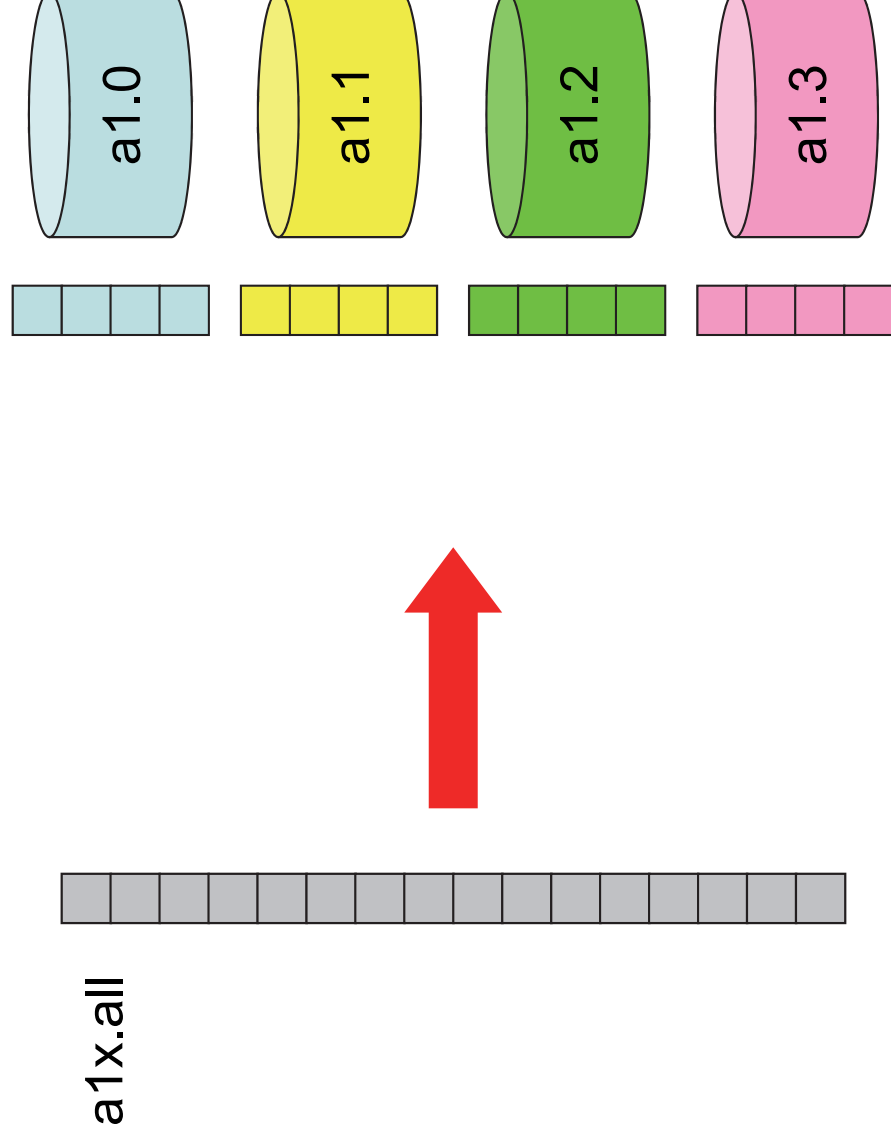
```
>$ cd <$O-S1>
>$ ls a1.*
a1.0 a1.1 a1.2 a1.3  a1x.all is decomposed to
                        4 files.

>$ mpifcpx -Kfast file.c
>$ mpifrtpx -Kfast file.f

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

# Operations of Distributed Local Files

- Local files `a1.0~a1.3` are originally from global file `a1x.all`.



# Reading Distributed Local Files:

## Uniform Vec. Length (2/2)

### <\$0-s1>/file.f

```
implicit REAL*8 (A-H, O-Z)
include 'mpif.h'
integer :: PETOT, my_rank, ierr
real(kind=8), dimension(8) :: VEC
character(len=80) :: filename

call MPI_INIT (ierr)
call MPI_COMM_SIZE (MPI_COMM_WORLD, PETOT, ierr )
call MPI_COMM_RANK (MPI_COMM_WORLD, my_rank, ierr )
```

Similar to  
"Hello"

```
if (my_rank.eq.0) filename= 'a1.0'
if (my_rank.eq.1) filename= 'a1.1'
if (my_rank.eq.2) filename= 'a1.2'
if (my_rank.eq.3) filename= 'a1.3'

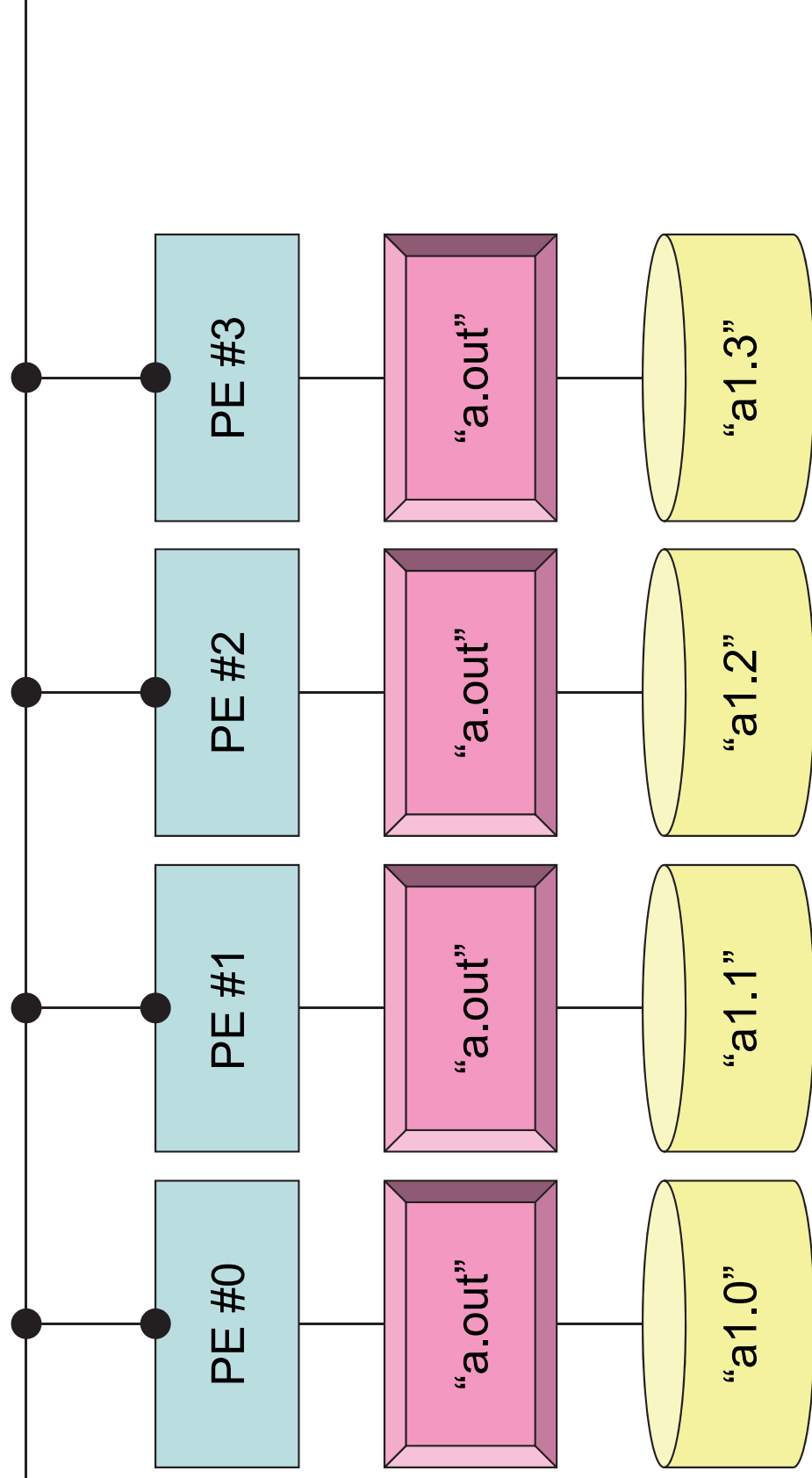
open (21, file= filename, status= 'unknown')
do i= 1, 8
  read (21,*) VEC(i)
enddo
close (21)

call MPI_FINALIZE (ierr)

stop
end
```

Local ID is 1-8

# Typical SPMD Operation



```
mpirun -np 4 a.out
```

# Non-Uniform Vector Length (1/2)

```
>$ cd <$O-S1>
>$ ls a2.*
a2.0 a2.1 a2.2 a2.3
>$ cat a2.0
5          Number of Components at each Process
201.0      Components
203.0
205.0
206.0
209.0

>$ mpifccpx -Kfast file2.c
>$ mpifrtpx -Kfast file2.f

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

# Non-Uniform Vector Length (2/2)

## <\$O-S1>/file2.f

```

implicit REAL*8 (A-H,O-Z)
include 'mpif.h'
integer :: PETOT, my_rank, ierr
real(kind=8), dimension(:), allocatable :: VEC
character(len=80) :: filename

call MPI_INIT (ierr)
call MPI_COMM_SIZE (MPI_COMM_WORLD, PETOT, ierr )
call MPI_COMM_RANK (MPI_COMM_WORLD, my_rank, ierr )

if (my_rank.eq.0) filename= 'a2.0'
if (my_rank.eq.1) filename= 'a2.1'
if (my_rank.eq.2) filename= 'a2.2'
if (my_rank.eq.3) filename= 'a2.3'

open (21, file= filename, status= 'unknown')
read (21,*) N
allocate (VEC(N))
do i= 1, N
    read (21,*) VEC(i)
enddo
close(21)

call MPI_FINALIZE (ierr)
stop
end

```

“N” is different at each process

# How to generate local data

- Reading global data ( $N=NG$ )
  - Scattering to each process
  - Parallel processing on each process
  - (If needed) reconstruction of global data by gathering local data
- Generating local data ( $N=NL$ ), or reading distributed local data
  - Generating or reading local data on each process
  - Parallel processing on each process
  - (If needed) reconstruction of global data by gathering local data
- In future, latter case is more important, but former case is also introduced in this class for understanding of operations of global/local data.

# Examples by Collective Comm.

- Dot Products of Vectors
- Scatter/Gather
- Reading Distributed Files
- **MPI\_Allgatherv**



# **MPI\_GATHERV, MPI\_SCATTERV**

- MPI\_Gather, MPI\_Scatter
  - Length of message from/to each process is uniform
- MPI\_XXXv extends functionality of MPI\_XXX by allowing a varying count of data from each process:
  - MPI\_Gatherv
  - MPI\_Scatterv
  - MPI\_Allgatherv
  - MPI\_Alltoallv

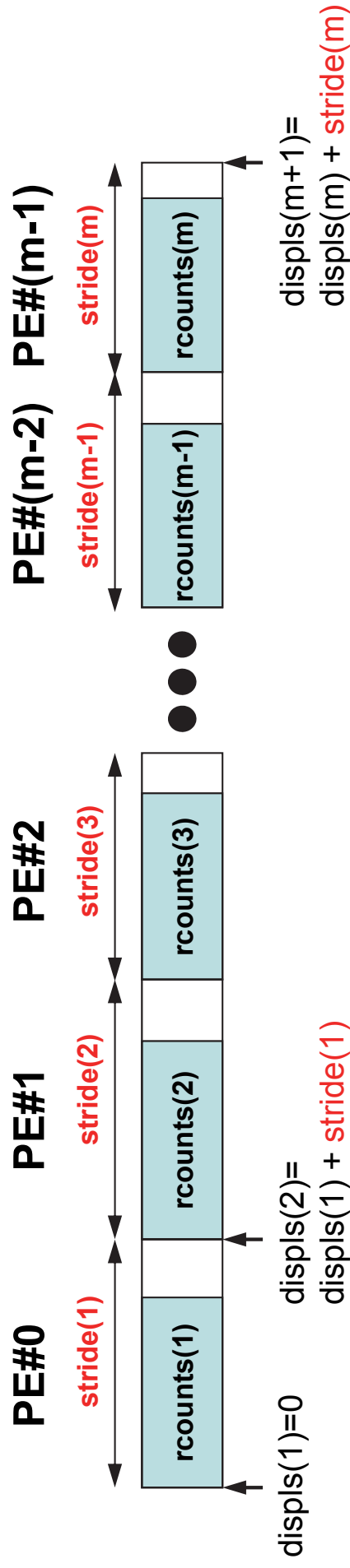
# MPI\_ALLGATHERV

- Variable count version of MPI\_Allgather
  - creates “global data” from “local data”
- call **MPI\_ALLGATHERV** (**sendbuf**, **scount**, **sendtype**, **recvbuf**, **rcounts**, **displs**, **recvtype**, **comm**, **ierr**)
  - sendbuf choice I starting address of sending buffer
  - scount I I number of elements sent to each process
  - sendtype I I data type of elements of sending buffer
  - recvbuf choice O starting address of receiving buffer
  - rcounts I I integer array (of length group size) containing the number of elements that are to be received from each process  
(array: size= PETOT)
  - displs I I integer array (of length group size). Entry *i* specifies the displacement (relative to recvbuf ) at which to place the incoming data from process *i* (array: size= PETOT+1)
  - recvtype I I data type of elements of receiving buffer
  - comm I I communicator
  - ierr I O completion code

# MPI\_ALLGATHERV (cont.)

Fortran

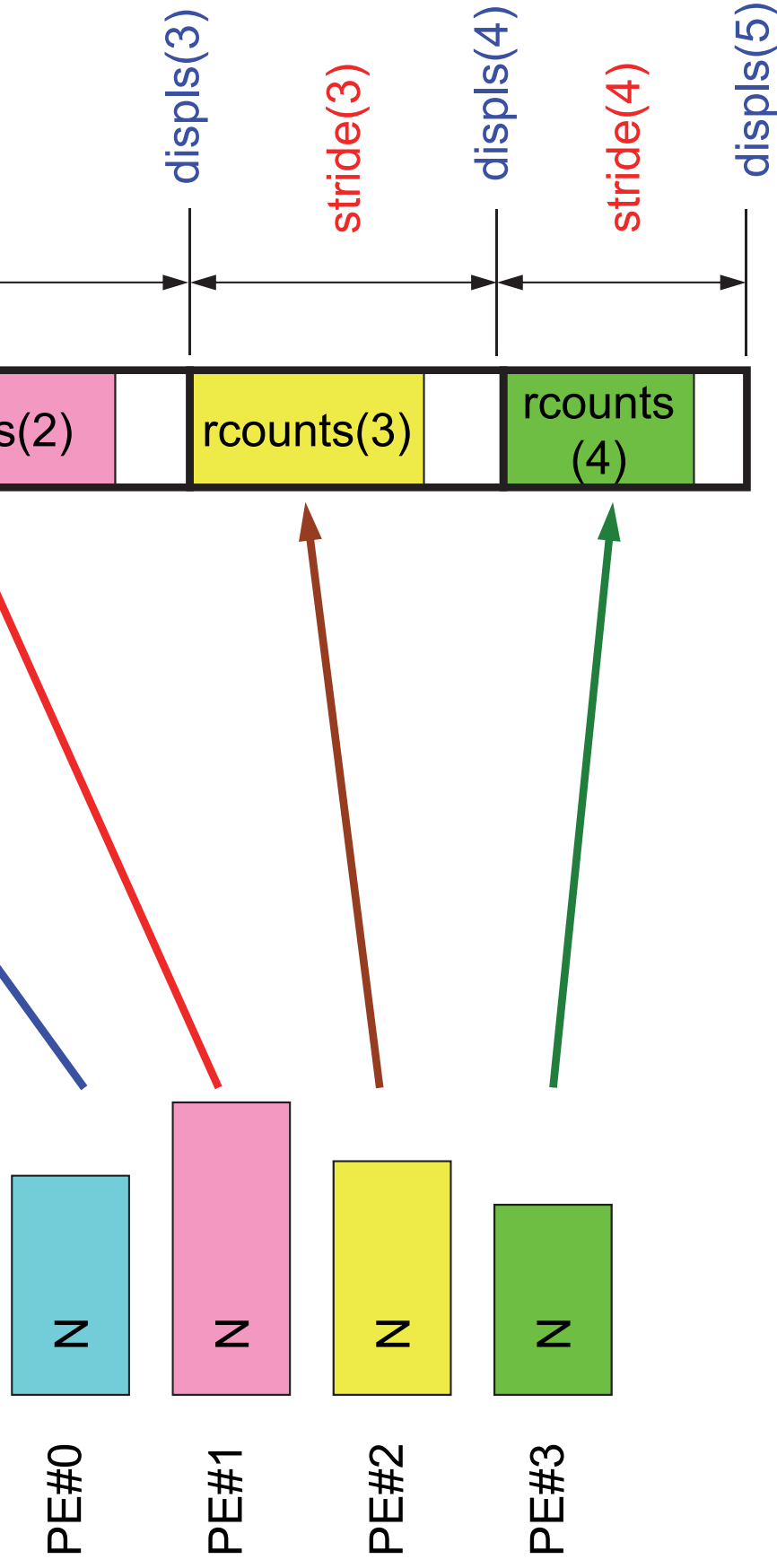
- call `MPI_ALLGATHERV (sendbuf, scount, sendtype, recvbuf, rcounts, displs, recvttype, comm, ierr)`
  - rcounts    I    I    integer array (of length group size) containing the number of elements that are to be received from each process  
(array: `size= PETOT`)
  - displs    I    I    integer array (of length group size). Entry *i* specifies the displacement (relative to `recvbuf`) at which to place the incoming data from process *i* (array: `size= PETOT+1`)
  - These two arrays are related to size of final “global data”, therefore each process requires information of these arrays (`rcounts`, `displs`)
    - Each process must have same values for all components of both vectors
  - Usually, `stride(i)=rcounts(i)`



$$\text{size(recvbuf)} = \text{displs}(\text{PETOT}+1) = \text{sum}(\text{stride})$$

# What MPI\_Allgatherv is doing

Generating global data from local data

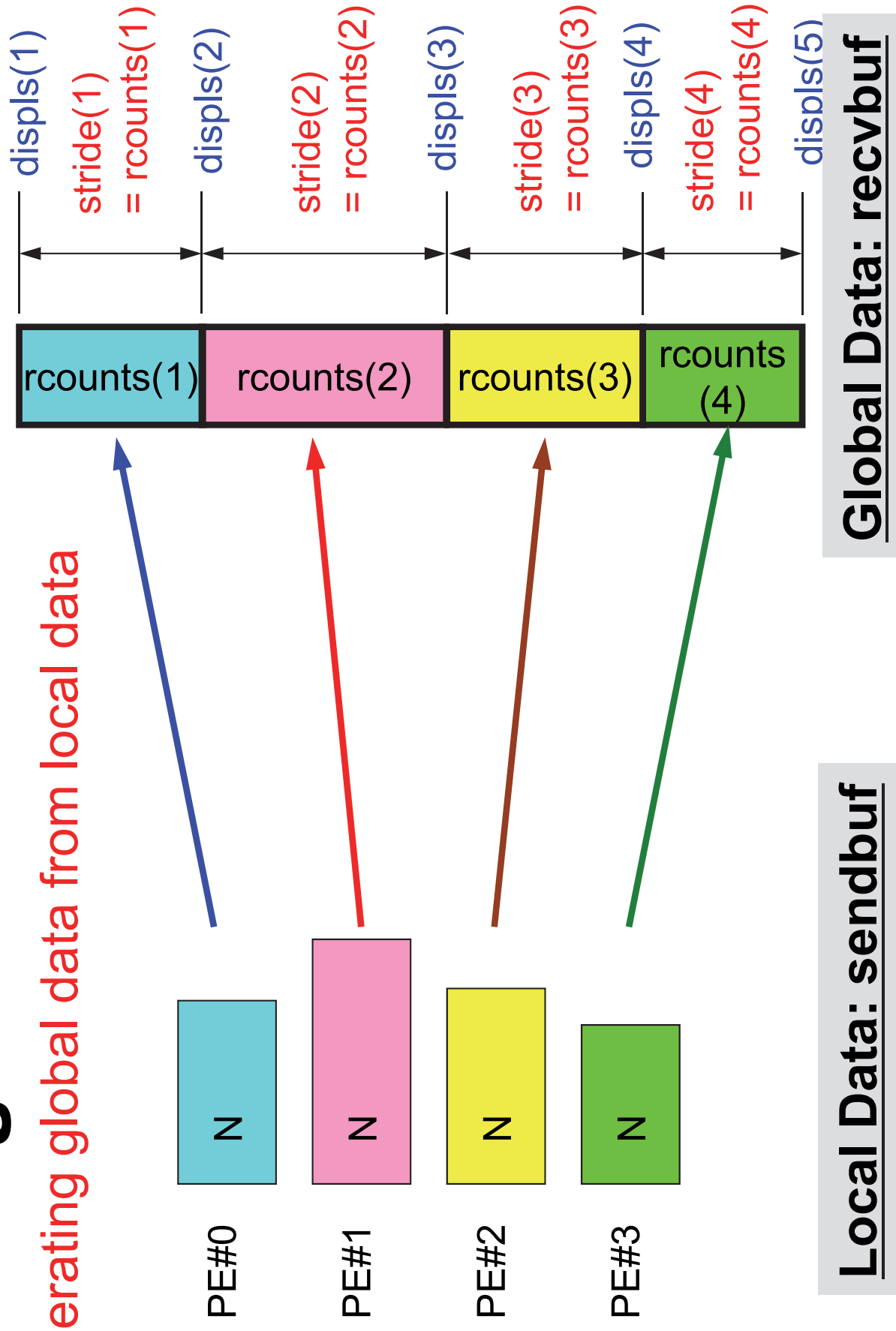


Local Data: sendbuf

Global Data: recvbuf

# What MPI\_Allgatherv is doing

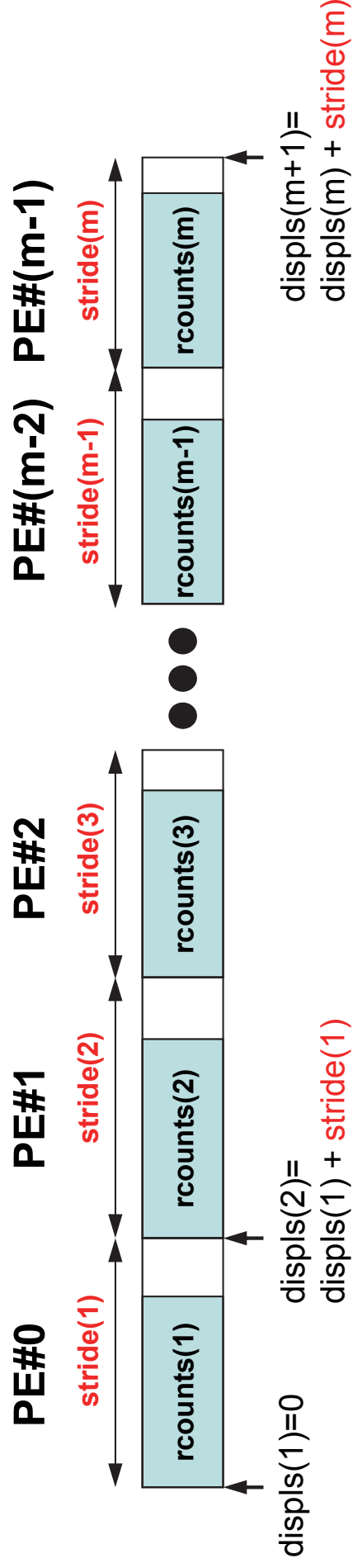
Generating global data from local data



# MPI\_Allgather in detail (1/2)

Fortran

- call `MPI_ALLGATHERV` (`sendbuf`, `scount`, `sendtype`, `recvbuf`, `rcounts`, `displs`, `recvtype`, `comm`, `ierr`)
- **rcounts**
  - Size of message from each PE: Size of Local Data (Length of Local Vector)
- **displs**
  - Address/index of each local data in the vector of global data
  - `displs(PETOT+1)` = Size of Entire Global Data (Global Vector)

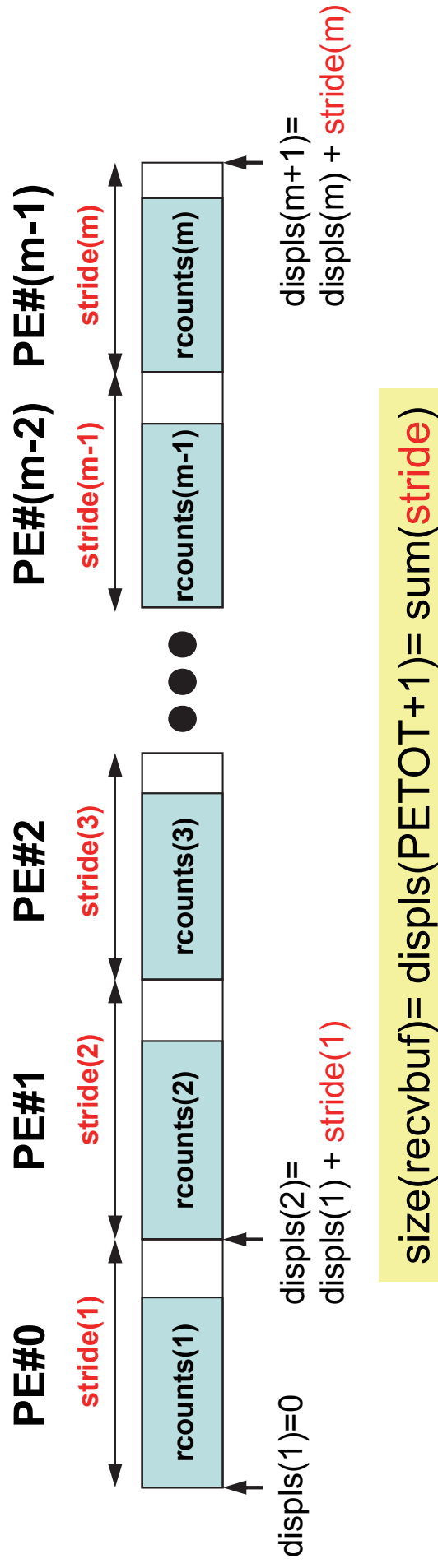


$$\text{size}(\text{recvbuf}) = \text{displs}(\text{PETOT}+1) = \text{sum}(\text{stride})$$

# MPI\_Allgather in detail (2/2)

Fortran

- Each process needs information of `rcounts` & `displs`
  - “`rcounts`” can be created by gathering local vector length “`N`” from each process.
  - On each process, “`displs`” can be generated from “`rcounts`” on each process.
    - `stride[i] = rcounts[i]`
  - Size of “`recvbuf`” is calculated by summation of “`rcounts`”.



# Preparation for MPI\_Allgather

## <\$O-S1>/agv.f

- Generating global vector from “a2.0”~”a2.3”.
- Length of the each vector is 8, 5, 7, and 3, respectively. Therefore, size of final global vector is 23  
(= 8+5+7+3).



# a2.0~a2.3

## PE#0

8  
101.0  
103.0  
105.0  
106.0  
109.0  
111.0  
121.0  
151.0

## PE#1

5  
201.0  
203.0  
205.0  
206.0  
209.0

## PE#2

7  
301.0  
303.0  
305.0  
306.0  
311.0  
321.0  
351.0

## PE#3

3  
401.0  
403.0  
405.0

# Preparation: MPI\_Allgatherv (1/4)

<\$O-S1>/agv.f

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

integer :: PETOT, my_rank, SOLVER_COMM, ierr
real(kind=8), dimension(:), allocatable :: VEC
real(kind=8), dimension(:), allocatable :: VEC2
real(kind=8), dimension(:), allocatable :: VECg
integer(kind=4), dimension(:), allocatable :: rcounts
integer(kind=4), dimension(:), allocatable :: displs
character(len=80) :: filename

call MPI_INIT (ierr)
call MPI_COMM_SIZE (MPI_COMM_WORLD, PETOT, ierr)
call MPI_COMM_RANK (MPI_COMM_WORLD, my_rank, ierr)

if (my_rank.eq.0) filename= 'a2.0'
if (my_rank.eq.1) filename= 'a2.1'
if (my_rank.eq.2) filename= 'a2.2'
if (my_rank.eq.3) filename= 'a2.3'

open (21, file= filename, status= 'unknown')
  read (21,*) N
  allocate (VEC(N))
  do i= 1, N
    read (21,*) VEC(i)
  enddo
```

N(NL) is different at  
each process

# Preparation: MPI\_Allgatherv (2/4)

<\$O-S1>/agv.f

```
allocate (rcounts(PETOT), displs(PETOT+1))
rcounts= 0
write (*, '(a,10i8)') "before", my_rank, N, rcounts

call MPI_allgather ( N      , 1, MPI_INTEGER,
&                    rcounts, 1, MPI_INTEGER,
&                    MPI_COMM_WORLD, _ierr)

write (*, '(a,10i8)') "after ", my_rank, N, rcounts
displs(1)= 0
```

Rcounts on each PE

PE#0 N=8

PE#1 N=5

PE#2 N=7

PE#3 N=3



MPI\_Allgather

rcounts(1:4)= {8, 5, 7, 3}

rcounts(1:4)= {8, 5, 7, 3}

rcounts(1:4)= {8, 5, 7, 3}

rcounts(1:4)= {8, 5, 7, 3}

# Preparation: MPI\_Allgatherv (2/4)

<\$0-\$1>/agv.f

```

allocate (rcounts(PETOT), displs(PETOT+1))
rcounts= 0
write (*,'(a,10i8)') "before", my_rank, N, rcounts

call MPI_allgather ( N      , 1, MPI_INTEGER,
&                    rcounts, 1, MPI_INTEGER,
&                    MPI_COMM_WORLD, _ierr)
                                &
                                &
                                Rcounts on each PE

write (*,'(a,10i8)') "after ", my_rank, N, rcounts
displs(1)= 0

do ip= 1, PETOT
    displs(ip+1)= displs(ip) + rcounts(ip)
enddo
                                Displs on each PE

write (*,'(a,10i8)') "displs", my_rank, displs

call MPI_FINALIZE (ierr)

stop
end

```

# Preparation: MPI\_Allgatherv (3/4)

```

> cd <$O-S1>
> mpifrtpx -Kfast agv.f, mpifccpx -Kfast agv.c

(modify go4.sh for 4 processes)
> pjsub go4.sh

before 0 8 0 0 0 0 0
after 0 8 8 7 3 0
displs 0 0 8 20 23
FORTRAN STOP

before 1 5 0 0 0 0
after 1 5 8 7 3 0
displs 1 0 8 20 23
FORTRAN STOP

before 3 3 0 0 0 0
after 3 3 8 7 3 0
displs 3 0 8 20 23
FORTRAN STOP

before 2 7 0 0 0 0
after 2 7 8 7 3 0
displs 2 0 8 20 23
FORTRAN STOP

write (*, '(a,10i8)') "before", my_rank, N, rcounts
write (*, '(a,10i8)') "after", my_rank, N, rcounts
write (*, '(a,10i8)') "displs", my_rank, displs

```

# Preparation: MPI\_Allgather (4/4)

- Only "recvbuf" is not defined yet.
- Size of "recvbuf" = "displs (PETOT+1) "

```
call MPI_allgather  
    ( VEC , N, MPI_DOUBLE_PRECISION,  
      recvbuf, rcounts, displs, MPI_DOUBLE_PRECISION,  
      MPI_COMM_WORLD, ierr)
```

# Report S1 (1/2)

- Deadline: 17:00 February 14<sup>th</sup> (Sat), 2015.
  - Send files via e-mail at **nakajima(at)cc.u-tokyo.ac.jp**
- Problem S1-1
  - Read local files `<O-S1>/a1.0~a1.3`, `<O-S1>/a2.0~a2.3`.
  - Develop codes which calculate norm  $\|x\|$  of global vector for each case.
    - `<O-S1>file.f`, `<O-S1>file2.f`
- Problem S1-2
  - Read local files `<O-S1>/a2.0~a2.3`.
  - Develop a code which constructs “global vector” using `MPI_Allgatherv`.

# Report S1 (2/2)

- Problem S1-3
  - Develop parallel program which calculates the following numerical integration using “trapezoidal rule” by MPI\_Reduce, MPI\_Bcast etc.
  - Measure computation time, and parallel performance

$$\int_0^1 \frac{4}{1+x^2} dx$$

- Report
  - Cover Page: Name, ID, and Problem ID (S1) must be written.
  - Less than two pages including figures and tables (A4) for each of three sub-problems
    - Strategy, Structure of the Program, Remarks
  - Source list of the program (if you have bugs)
  - Output list (as small as possible)



- What is MPI ?
- Your First MPI Program: Hello World
- Global/Local Data
- Collective Communication
- **Peer-to-Peer Communication**

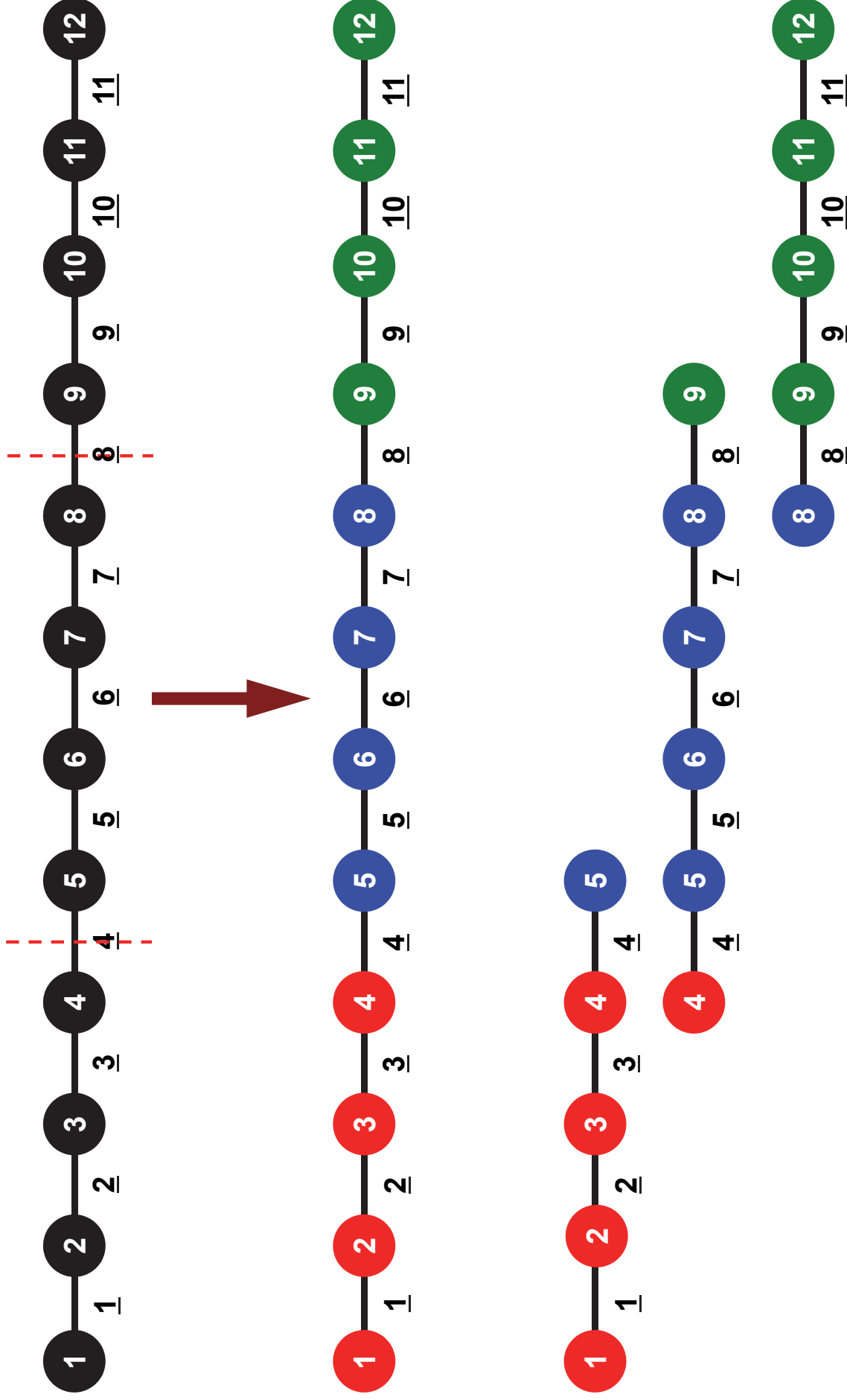
# Peer-to-Peer Communication

## Point-to-Point Communication

### 1対1通信

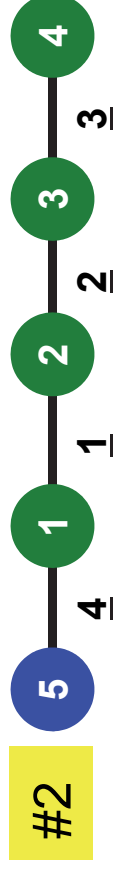
- What is P2P Communication ?
- 2D Problem, Generalized Communication Table
- Report S2

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



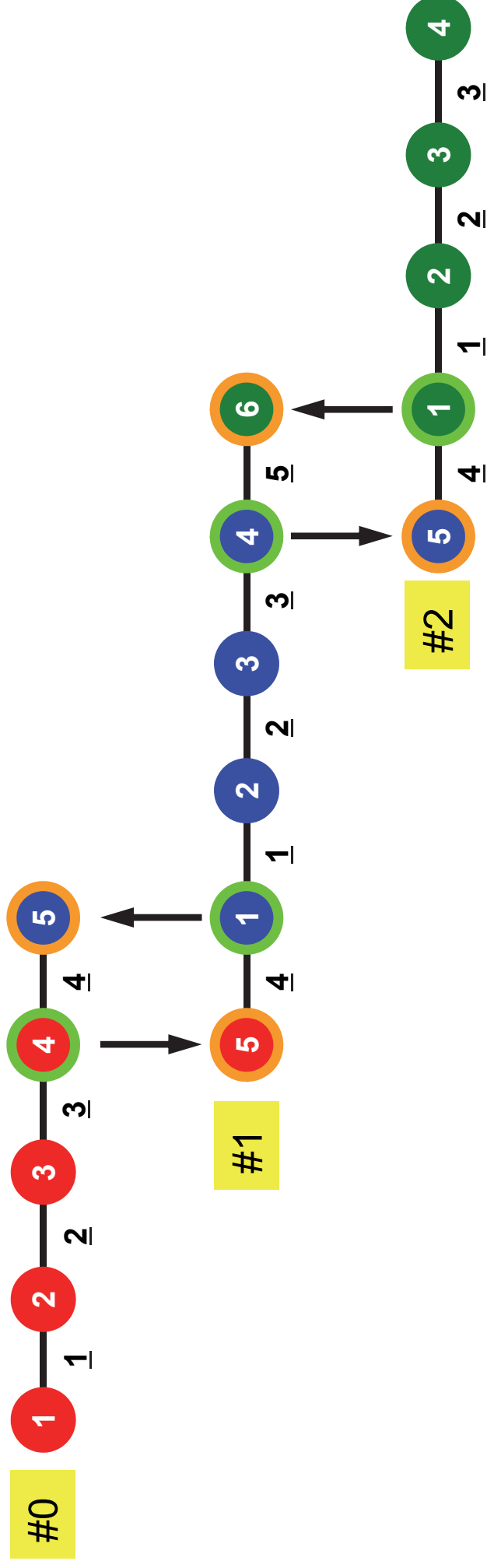
# 1D FEM: 12 nodes/11 elem's/3 domains

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



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

## Internal/External Nodes



# Preconditioned Conjugate Gradient Method (CG)

```

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

```

Preconditioner:

Diagonal Scaling

Point-Jacobi Preconditioning

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

# Preconditioning, DAXPY

Local Operations by Only Internal Points: Parallel Processing is possible

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

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

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

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

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

# Dot Products

Global Summation needed: Communication ?

```
!C
!C-- ALPHA= RH0 / {p} {q}

C1= 0.d0
do i= 1, N
  C1= C1 + W(i, P)*W(i, Q)
enddo
ALPHA= RH0 / C1
```

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



# Matrix-Vector Products

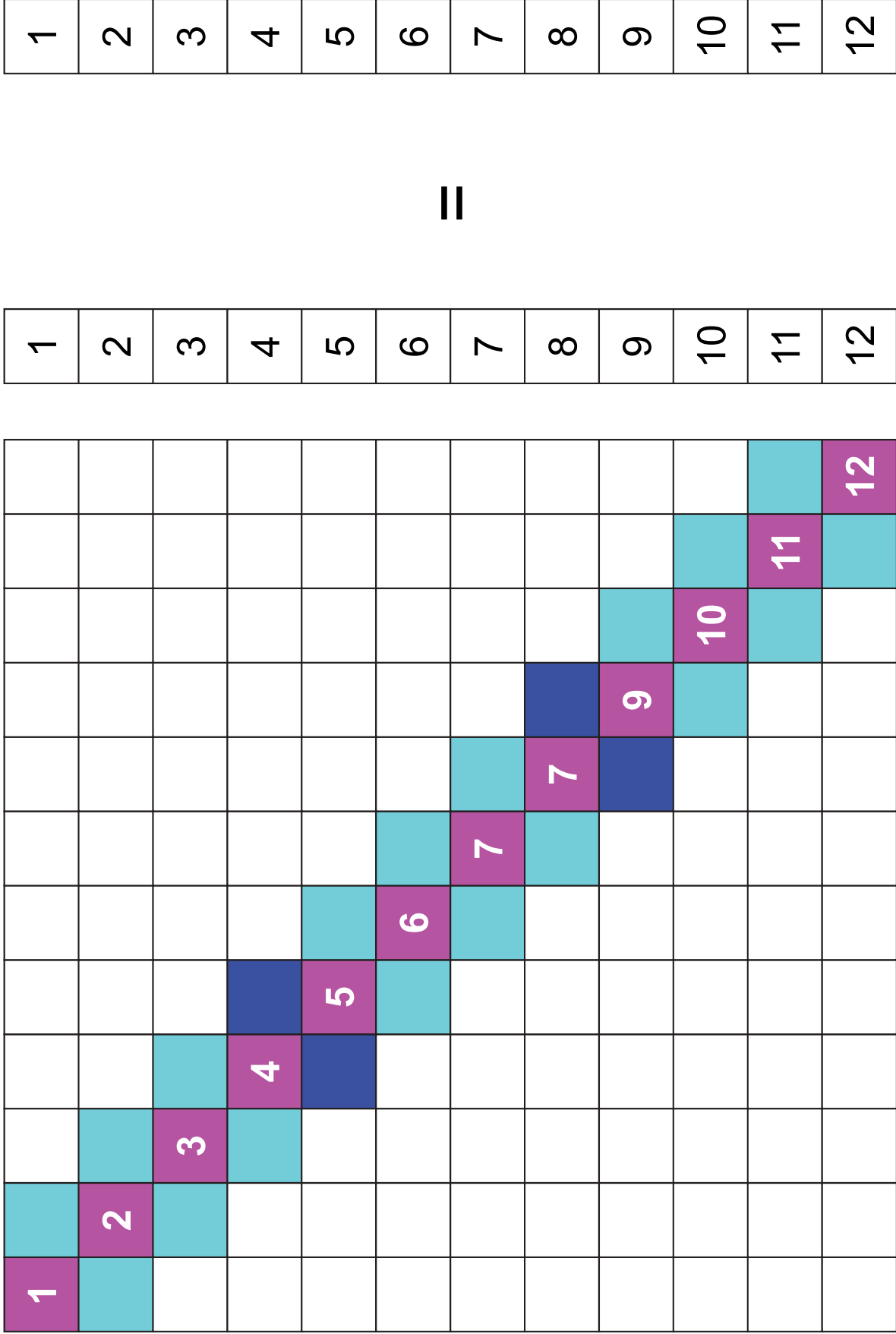
## Values at External Points: P-to-P Communication

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

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



# Mat-Vec Products: Local Op. Possible



# Mat-Vec Products: Local Op. Possible

# Mat-Vec Products: Local Op. Possible

|   |  |  |  |
|---|--|--|--|
| 1 |  |  |  |
|   |  |  |  |
|   |  |  |  |
|   |  |  |  |

|   |   |   |   |
|---|---|---|---|
| 1 | 2 | 3 | 4 |
|---|---|---|---|

|   |   |   |   |
|---|---|---|---|
| 1 | 2 | 3 | 4 |
|---|---|---|---|

|  |  |  |  |
|--|--|--|--|
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

|   |   |   |   |
|---|---|---|---|
| 5 | 6 | 7 | 8 |
|---|---|---|---|

|   |   |   |   |
|---|---|---|---|
| 5 | 6 | 7 | 8 |
|---|---|---|---|

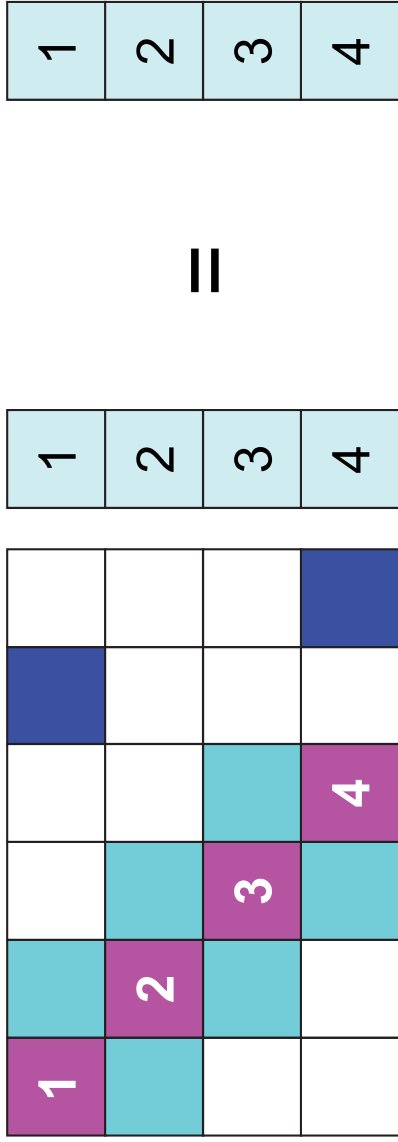
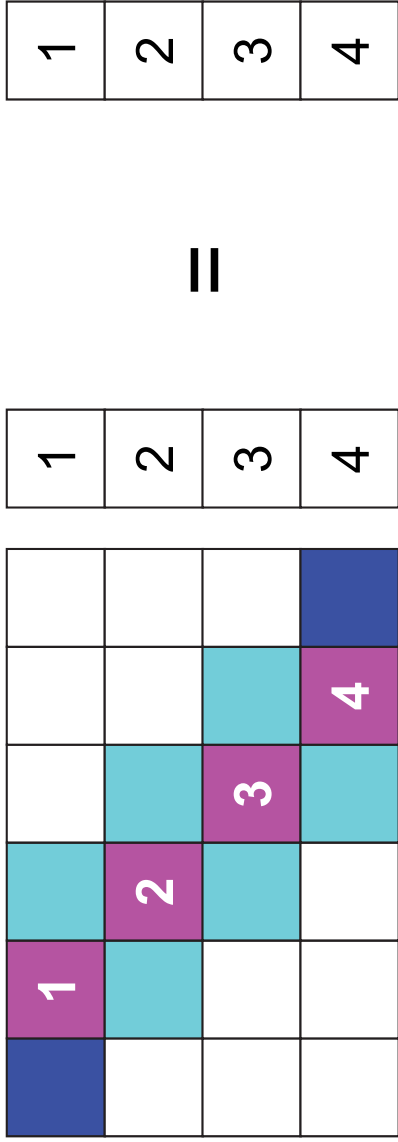
=

|  |  |  |  |
|--|--|--|--|
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

|   |    |    |    |
|---|----|----|----|
| 9 | 10 | 11 | 12 |
|---|----|----|----|

|   |    |    |    |
|---|----|----|----|
| 9 | 10 | 11 | 12 |
|---|----|----|----|

# Mat-Vec Products: Local Op. #1

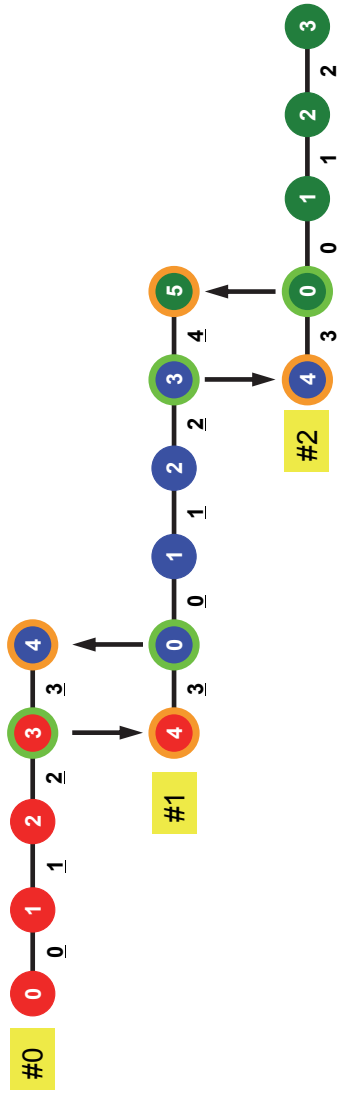


# What is Peer-to-Peer Communication ?

- Collective Communication
  - MPI\_Reduce, MPI\_Scatter/Gather etc.
  - Communications with all processes in the communicator
  - Application Area
    - BEM, Spectral Method, MD: global interactions are considered
    - Dot products, MAX/MIN: Global Summation & Comparison

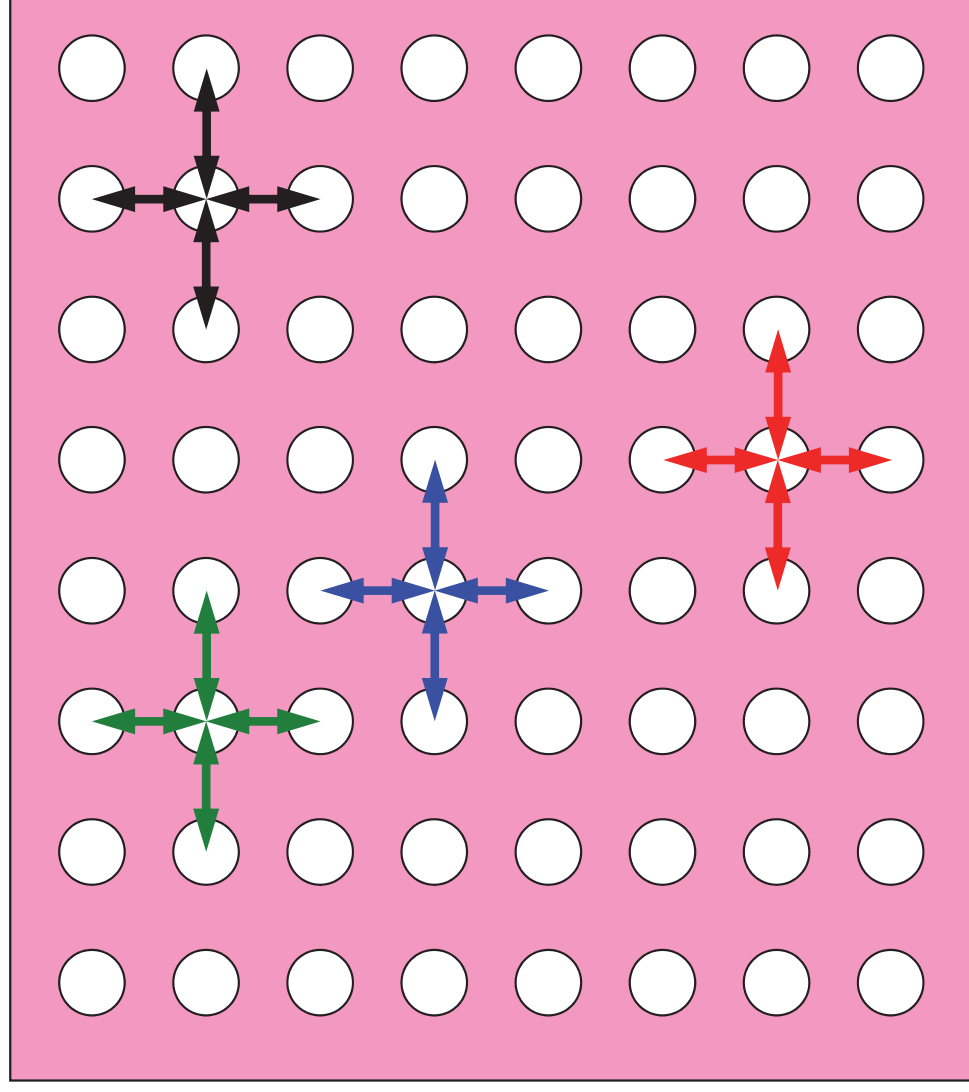
## • Peer-to-Peer/Point-to-Point

- MPI\_Send, MPI\_Receive
- Communication with limited processes
  - Neighbors
- Application Area
  - FEM, FDM: Localized Method



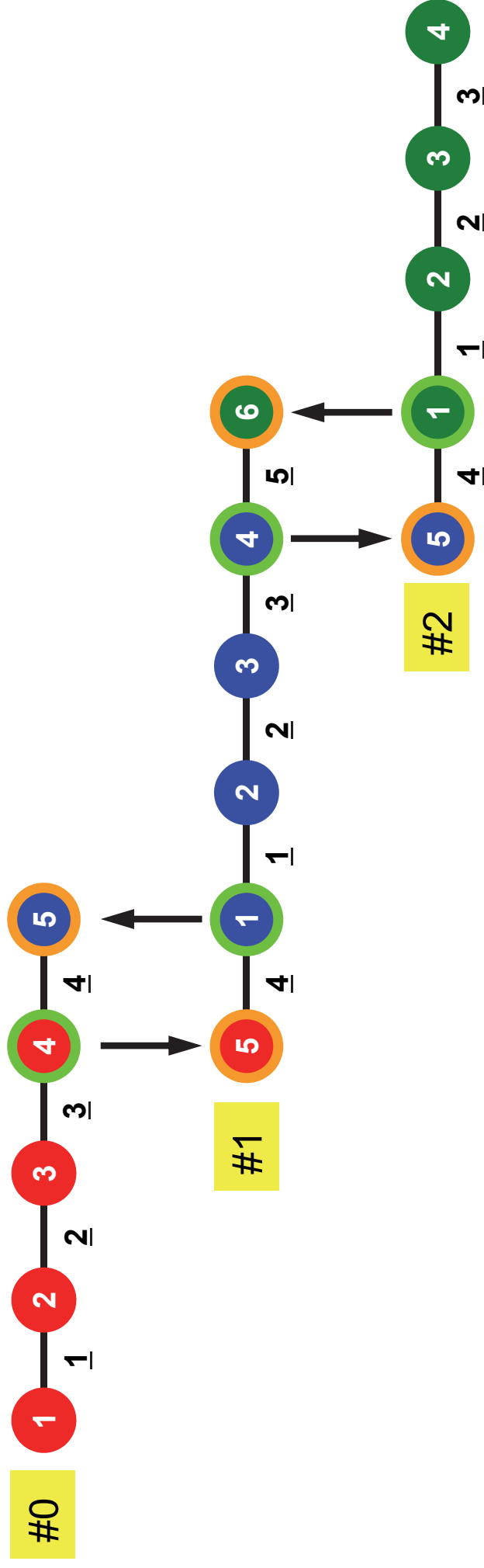
# Collective/P2P Communications

Interactions with only Neighboring Processes/Element  
Finite Difference Method (FDM), Finite Element  
Method (FEM)



# When do we need P2P comm.: 1D-FEM

Info in neighboring domains is required for FEM operations  
Matrix assembling, Iterative Method

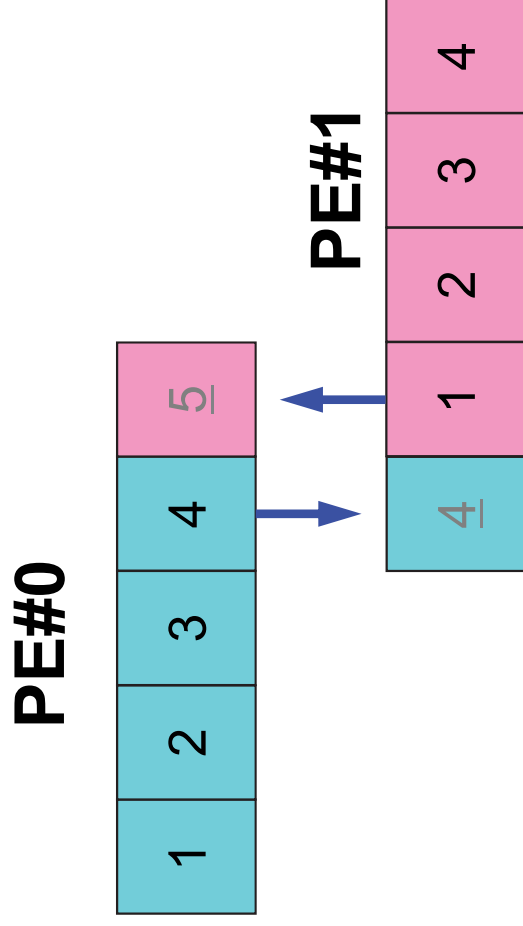




# Method for P2P Comm.

- **MPI\_Send, MPI\_Recv**
- These are “blocking” functions. “Dead lock” occurs for these “blocking” functions.
- A “blocking” MPI call means that the program execution will be suspended until the message buffer is safe to use.
- The MPI standards specify that a blocking SEND or RECV does not return until the send buffer is safe to reuse (for MPI\_Send), or the receive buffer is ready to use (for MPI\_Recv).
  - Blocking comm. confirms “secure” communication, but it is very inconvenient.
- Please just remember that “there are such functions” .

# MPI\_Send/MPI\_Recv



```

if (my_rank.eq.0) NEIB_ID=1
if (my_rank.eq.1) NEIB_ID=0

...
call MPI_SEND (NEIB_ID, arg's)
call MPI_RECV (NEIB_ID, arg's)
...

```

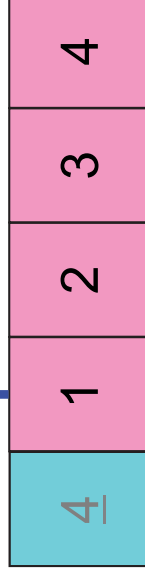
- This seems reasonable, but it stops at MPI\_Send/MPI\_Recv.
- Sometimes it works (according to implementation).

# MPI\_Send/MPI\_Recv (cont.)

PE#0



PE#1



```

if (my_rank.eq.0) NEIB_ID=1
if (my_rank.eq.1) NEIB_ID=0
...
if (my_rank.eq.0) then
  call MPI_SEND (NEIB_ID, arg's)
  call MPI_RECV (NEIB_ID, arg's)
endif

if (my_rank.eq.1) then
  call MPI_RECV (NEIB_ID, arg's)
  call MPI_SEND (NEIB_ID, arg's)
endif
...

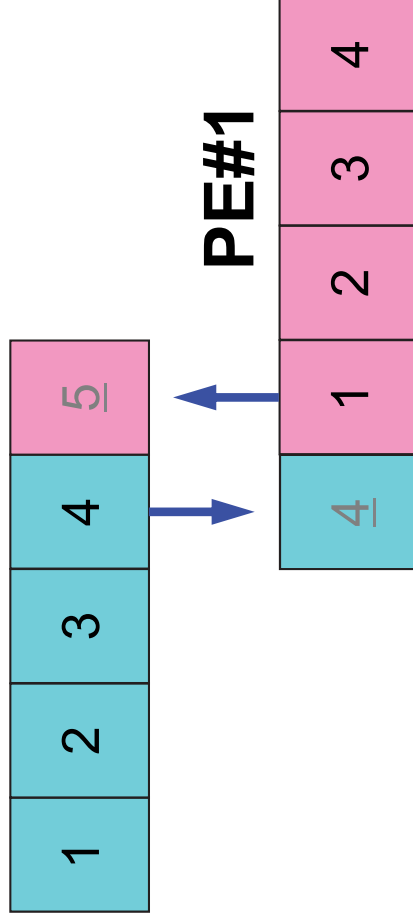
```

- It works ... but

# How to do P2P Comm. ?

- Using “non-blocking” functions `MPI_Isend` & `MPI_Irecv` together with `MPI_Waitall` for synchronization
- `MPI_Sendrecv` is also available.

**PE#0**



```

if (my_rank.eq.0) NEIB_ID=1
if (my_rank.eq.1) NEIB_ID=0

...
call MPI_Isend (NEIB_ID, arg's)
call MPI_Irecv (NEIB_ID, arg's)
...
call MPI_Waitall (for Irecv)
...
call MPI_Waitall (for Isend)
  
```

**`MPI_Waitall` for both of `MPI_Isend`/`MPI_Irecv` is possible**

# MPI\_SEND

- 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)`

- |                   |        |   |                                         |
|-------------------|--------|---|-----------------------------------------|
| – <u>sendbuf</u>  | choice | I | starting address of sending buffer      |
| – <u>count</u>    | I      | I | number of elements sent to each process |
| – <u>datatype</u> | I      | I | data type of elements of sending buffer |
| – <u>dest</u>     | I      | I | rank of destination                     |
| – <u>tag</u>      | 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),

communicator

- |                  |   |   |
|------------------|---|---|
| – <u>comm</u>    | I | I |
| – <u>request</u> | I | ○ |
| – <u>ierr</u>    | I | ○ |

**communication request array used in `MPI_Waitall`**

completion code

# Communication Request: request

## 通信識別子

- call `MPI_ISEND`  
`(sendbuf, count, datatype, dest, tag, comm, request, ierr)`
  - sendbuf choice I starting address of sending buffer
  - count I number of elements sent to each process
  - datatype I data type of elements of sending buffer
  - dest I rank of destination
  - tag 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),

communicator

communication request used in `MPI_Waitall`

**Size of the array is total number of neighboring processes**

completion code

- Just define the array

**allocate (request(NEIBPETOT) )**

# 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 `tags` of `MPI_Isend` and `MPI_Irecv` are matched.

Usually tag is set to be “0” (in this class),  
communicator

- comm            I        I    communicator
- request        I        O    **communication request used in `MPI_waitall`**
- ierr            I        O    completion code

# MPI\_WAITALL

Fortran

- `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



# Array of status object: status

## 状況オブジェクト配列

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

|                  |   |     |                                                                                        |
|------------------|---|-----|----------------------------------------------------------------------------------------|
| – <u>count</u>   | I | I   | number of processes to be synchronized                                                 |
| – <u>request</u> | I | I/O | comm. request used in <code>MPI_Waitall</code> (array size: count)                     |
| – <u>status</u>  | I | O   | array of status objects                                                                |
|                  |   |     | <code>MPI_STATUS_SIZE</code> : defined in <code>'mpif.h'</code> , <code>'mpi.h'</code> |
| – <u>ierr</u>    | I | O   | completion code                                                                        |

- Just define the array

```
allocate (stat(MPI_STATUS_SIZE, NEIBPETOT))
```

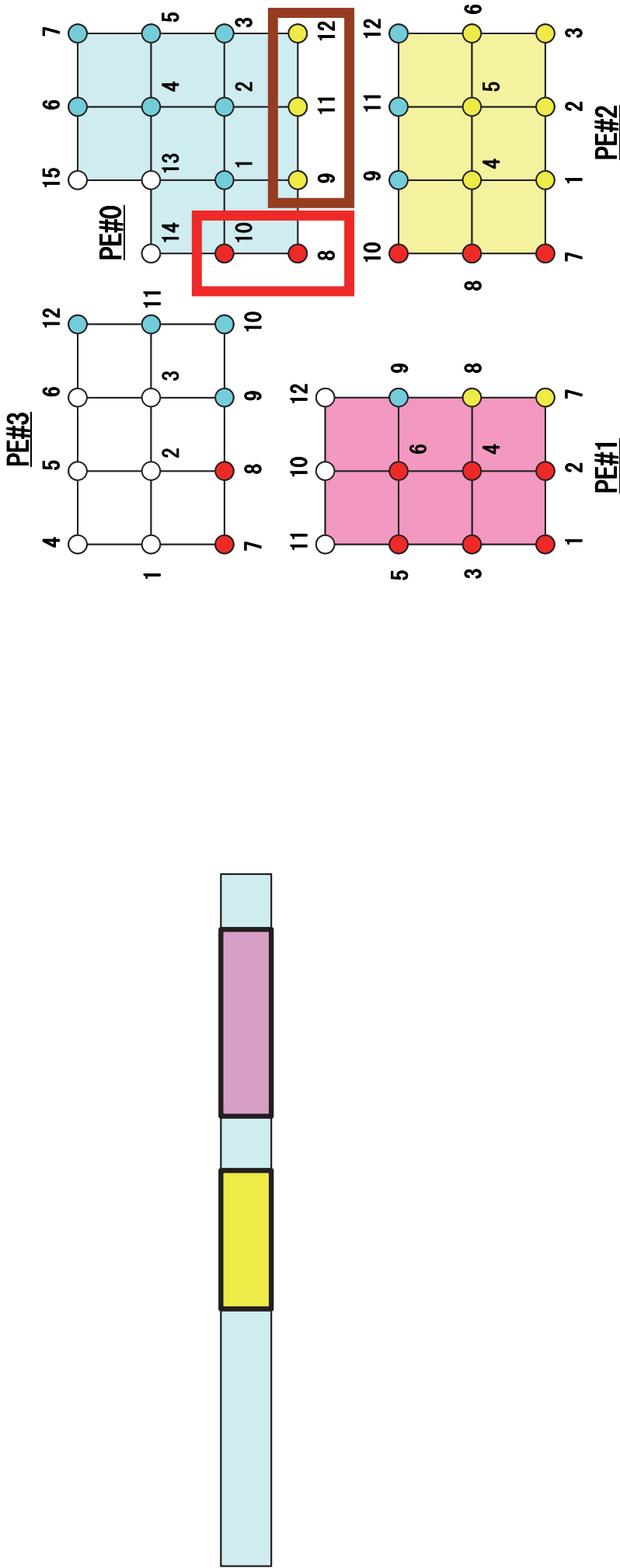
# MPI\_SENDRECV

- MPI\_Send+MPI\_Recv: not recommended, many restrictions
- call MPI\_SENDRECV  
 (sendbuf, sendcount, sendtype, dest, sendtag, recvbuf, recvcount, recvtype, source, recvtag, comm, status, ierr)
  - sendbuf choice I starting address of sending buffer
  - sendcount I I number of elements in sending buffer
  - sendtype I I datatype of each sending buffer element
  - dest I I rank of destination
  - sendtag I I message tag for sending
  - comm I I communicator
  - recvbuf choice I starting address of receiving buffer
  - recvcount I I number of elements in receiving buffer
  - recvtype I I datatype of each receiving buffer element
  - source I I rank of source
  - recvtag I I message tag for receiving
  - comm I I communicator
  - status I O **array of status objects**  
 MPI\_STATUS\_SIZE: defined in 'mpif.h', 'mpi.h'
  - ierr I O completion code

# Recv: receiving to external nodes

## Recv. continuous data to recv. buffer from neighbors

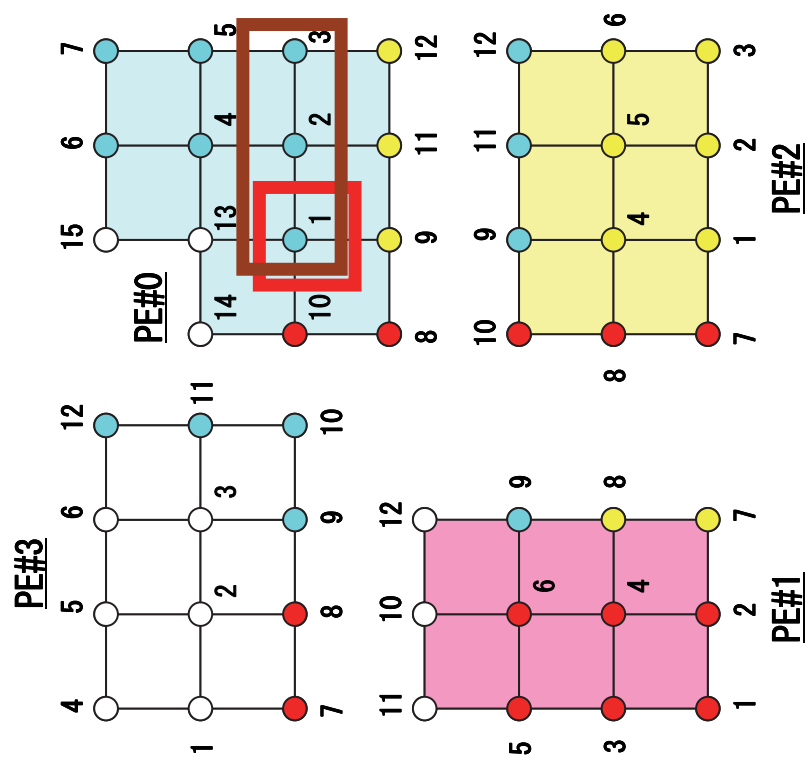
- `MPI_Irecv`  
(`recvbuf`, `count`, `datatype`, `dest`, `tag`, `comm`, `request`)
  - recvbuf   choice   I   starting address of receiving buffer
  - count       I       I   number of elements in receiving buffer
  - datatype   I       I   data type of elements of receiving buffer
  - source     I       I   rank of source



# SEND: sending from boundary nodes

## Send continuous data to send buffer of neighbors

- MPI\_Isend  
    (sendbuf, count, datatype, dest, tag, comm, request)
  - sendbuf   choice   I       starting address of sending buffer
  - count     I       I       number of elements sent to each process
  - datatype  I       I       data type of elements of sending buffer
  - dest      I       I       rank of destination



# Request, Status in Fortran

- **MPI\_Isend: request**
- **MPI\_Irecv: request**
- **MPI\_Waitall: request, status**

```
integer request (NEIBPETOT)  
integer status (MPI_STAUTS_SIZE, NEIBPETOT)
```

- **MPI\_Sendrecv: status**

```
integer status (MPI_STATUS_SIZE)
```

# Files on Oakleaf-FX

## Fortran

```
>$ cd <$O-TOP>
>$ cp /home/z30088/class_eps/F/s2-f.tar .
>$ tar xvf s2-f.tar
```

## C

```
>$ cd <$O-TOP>
>$ cp /home/z30088/class_eps/C/s2-c.tar .
>$ tar xvf s2-c.tar
```

## Confirm Directory

```
>$ ls
mpi
```

```
>$ cd mpi/s2
```

This directory is called as <\$O-S2> in this course.  
<\$O-S2> = <\$O-TOP>/mpi/s2

# Ex.1: Send-Recv a Scalar

- Exchange VAL (real, 8-byte) between PE#0 & PE#1

```

if (my_rank.eq.0) NEIB= 1
if (my_rank.eq.1) NEIB= 0

call MPI_Isend (VAL, 1, MPI_DOUBLE_PRECISION, NEIB, ..., req_send, ...)
call MPI_Irecv (VALtemp, 1, MPI_DOUBLE_PRECISION, NEIB, ..., req_recv, ...)
call MPI_Waitall (... , req_recv, stat_recv, ...) Recv.buf VALtemp can be used
call MPI_Waitall (... , req_send, stat_send, ...) Send buf VAL can be modified
VAL= VALtemp

```

```

if (my_rank.eq.0) NEIB= 1
if (my_rank.eq.1) NEIB= 0

call MPI_Sendrecv (VAL, 1, MPI_DOUBLE_PRECISION, NEIB, ... &
VALtemp, 1, MPI_DOUBLE_PRECISION, NEIB, ..., status, ...)
VAL= VALtemp

```

Name of recv. buffer could be “VAL”, but not recommended.

# Ex.1: Send-Recv a Scalar

## |send/lrecv/Waitall

```
$> cd <$0-$2>
$> mpifrtpx -Kfast ex1-1.f
$> pjsub go2.sh
```

```
implicit REAL*8 (A-H,O-Z)
include 'mpif.h'
integer(kind=4) :: my_rank, PETOT, NEIB
real (kind=8) :: VAL, VALtemp
integer(kind=4), dimension(MPI_STATUS_SIZE,1) :: stat_send, stat_recv
integer(kind=4), dimension(1)
:: request_send, request_recv

call MPI_INIT (ierr)
call MPI_COMM_SIZE (MPI_COMM_WORLD, PETOT, ierr )
call MPI_COMM_RANK (MPI_COMM_WORLD, my_rank, ierr )

if (my_rank.eq.0) then
  NEIB= 1
  VAL = 10.d0
else
  NEIB= 0
  VAL = 11.d0
endif

call MPI_ISEND (VAL, 1,MPI_DOUBLE_PRECISION,NEIB,0,MPI_COMM_WORLD,request_send(1),ierr)
call MPI_IRecv (VALx,1,MPI_DOUBLE_PRECISION,NEIB,0,MPI_COMM_WORLD,request_recv(1),ierr)
call MPI_WAITALL (1, request_recv, stat_recv, ierr)
call MPI_WAITALL (1, request_send, stat_send, ierr)
VAL= VALx

call MPI_FINALIZE (ierr)
end
```



# Ex.1: Send-Recv a Scalar

## SendRecv

```
$> cd <$0-$2>
$> mpifrtpx -Kfast ex1-2.f
$> pjsub go2.sh
```

```
implicit REAL*8 (A-H,O-Z)
include 'mpif.h'
integer(kind=4) :: my_rank, PETOT, NEIB
real    (kind=8) :: VAL, VALtemp
integer(kind=4) :: status(MPI_STATUS_SIZE)

call MPI_INIT      (ierr)
call MPI_COMM_SIZE (MPI_COMM_WORLD, PETOT, ierr )
call MPI_COMM_RANK (MPI_COMM_WORLD, my_rank, ierr )

if (my_rank.eq.0) then
  NEIB= 1
  VAL = 10.d0
endif
if (my_rank.eq.1) then
  NEIB= 0
  VAL = 11.d0
endif

call MPI_SENDRECV
& (VAL, 1, MPI_DOUBLE_PRECISION, NEIB, 0,
& VALtemp, 1, MPI_DOUBLE_PRECISION, NEIB, 0, MPI_COMM_WORLD, status, ierr)
&

VAL= VALtemp
call MPI_FINALIZE (ierr)
end
```

## Ex.2: Send-Recv an Array (1/4)

- Exchange VEC (real, 8-byte) between PE#0 & PE#1
- PE#0 to PE#1
  - PE#0: send VEC(1)-VEC(11) (length=11)
  - PE#1: recv. as VEC(26)-VEC(36) (length=11)
- **PE#1 to PE#0**
  - PE#1: send VEC(1)-VEC(25) (length=25)
  - PE#0: recv. as VEC(12)-VEC(36) (length=25)
- Practice: Develop a program for this operation.

PE#0

|   |   |   |   |   |   |   |   |   |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
|---|---|---|---|---|---|---|---|---|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 | 32 | 33 | 34 | 35 | 36 |
|---|---|---|---|---|---|---|---|---|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|

PE#1

|   |   |   |   |   |   |   |   |   |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
|---|---|---|---|---|---|---|---|---|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 | 32 | 33 | 34 | 35 | 36 |
|---|---|---|---|---|---|---|---|---|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|

# Practice: t1

- Initial status of VEC(:):
  - PE#0    VEC(1-36)= 101,102,103,~,135,136
  - PE#1    VEC(1-36)= 201,202,203,~,235,236
- Confirm the results in the next page
- Using following two functions:
  - MPI\_Isend/Irecv/Waitall
  - MPI\_Sendrecv

# Estimated Results

t1

|   |          |    |      |
|---|----------|----|------|
| 0 | #BEFORE# | 1  | 101. |
| 0 | #BEFORE# | 2  | 102. |
| 0 | #BEFORE# | 3  | 103. |
| 0 | #BEFORE# | 4  | 104. |
| 0 | #BEFORE# | 5  | 105. |
| 0 | #BEFORE# | 6  | 106. |
| 0 | #BEFORE# | 7  | 107. |
| 0 | #BEFORE# | 8  | 108. |
| 0 | #BEFORE# | 9  | 109. |
| 0 | #BEFORE# | 10 | 110. |
| 0 | #BEFORE# | 11 | 111. |
| 0 | #BEFORE# | 12 | 112. |
| 0 | #BEFORE# | 13 | 113. |
| 0 | #BEFORE# | 14 | 114. |
| 0 | #BEFORE# | 15 | 115. |
| 0 | #BEFORE# | 16 | 116. |
| 0 | #BEFORE# | 17 | 117. |
| 0 | #BEFORE# | 18 | 118. |
| 0 | #BEFORE# | 19 | 119. |
| 0 | #BEFORE# | 20 | 120. |
| 0 | #BEFORE# | 21 | 121. |
| 0 | #BEFORE# | 22 | 122. |
| 0 | #BEFORE# | 23 | 123. |
| 0 | #BEFORE# | 24 | 124. |
| 0 | #BEFORE# | 25 | 125. |
| 0 | #BEFORE# | 26 | 126. |
| 0 | #BEFORE# | 27 | 127. |
| 0 | #BEFORE# | 28 | 128. |
| 0 | #BEFORE# | 29 | 129. |
| 0 | #BEFORE# | 30 | 130. |
| 0 | #BEFORE# | 31 | 131. |
| 0 | #BEFORE# | 32 | 132. |
| 0 | #BEFORE# | 33 | 133. |
| 0 | #BEFORE# | 34 | 134. |
| 0 | #BEFORE# | 35 | 135. |
| 0 | #BEFORE# | 36 | 136. |

|   |          |    |      |
|---|----------|----|------|
| 0 | #AFTER # | 1  | 101. |
| 0 | #AFTER # | 2  | 102. |
| 0 | #AFTER # | 3  | 103. |
| 0 | #AFTER # | 4  | 104. |
| 0 | #AFTER # | 5  | 105. |
| 0 | #AFTER # | 6  | 106. |
| 0 | #AFTER # | 7  | 107. |
| 0 | #AFTER # | 8  | 108. |
| 0 | #AFTER # | 9  | 109. |
| 0 | #AFTER # | 10 | 110. |
| 0 | #AFTER # | 11 | 111. |
| 0 | #AFTER # | 12 | 201. |
| 0 | #AFTER # | 13 | 202. |
| 0 | #AFTER # | 14 | 203. |
| 0 | #AFTER # | 15 | 204. |
| 0 | #AFTER # | 16 | 205. |
| 0 | #AFTER # | 17 | 206. |
| 0 | #AFTER # | 18 | 207. |
| 0 | #AFTER # | 19 | 208. |
| 0 | #AFTER # | 20 | 209. |
| 0 | #AFTER # | 21 | 210. |
| 0 | #AFTER # | 22 | 211. |
| 0 | #AFTER # | 23 | 212. |
| 0 | #AFTER # | 24 | 213. |
| 0 | #AFTER # | 25 | 214. |
| 0 | #AFTER # | 26 | 215. |
| 0 | #AFTER # | 27 | 216. |
| 0 | #AFTER # | 28 | 217. |
| 0 | #AFTER # | 29 | 218. |
| 0 | #AFTER # | 30 | 219. |
| 0 | #AFTER # | 31 | 220. |
| 0 | #AFTER # | 32 | 221. |
| 0 | #AFTER # | 33 | 222. |
| 0 | #AFTER # | 34 | 223. |
| 0 | #AFTER # | 35 | 224. |
| 0 | #AFTER # | 36 | 225. |

|   |          |    |      |
|---|----------|----|------|
| 1 | #BEFORE# | 1  | 201. |
| 1 | #BEFORE# | 2  | 202. |
| 1 | #BEFORE# | 3  | 203. |
| 1 | #BEFORE# | 4  | 204. |
| 1 | #BEFORE# | 5  | 205. |
| 1 | #BEFORE# | 6  | 206. |
| 1 | #BEFORE# | 7  | 207. |
| 1 | #BEFORE# | 8  | 208. |
| 1 | #BEFORE# | 9  | 209. |
| 1 | #BEFORE# | 10 | 210. |
| 1 | #BEFORE# | 11 | 211. |
| 1 | #BEFORE# | 12 | 212. |
| 1 | #BEFORE# | 13 | 213. |
| 1 | #BEFORE# | 14 | 214. |
| 1 | #BEFORE# | 15 | 215. |
| 1 | #BEFORE# | 16 | 216. |
| 1 | #BEFORE# | 17 | 217. |
| 1 | #BEFORE# | 18 | 218. |
| 1 | #BEFORE# | 19 | 219. |
| 1 | #BEFORE# | 20 | 220. |
| 1 | #BEFORE# | 21 | 221. |
| 1 | #BEFORE# | 22 | 222. |
| 1 | #BEFORE# | 23 | 223. |
| 1 | #BEFORE# | 24 | 224. |
| 1 | #BEFORE# | 25 | 225. |
| 1 | #BEFORE# | 26 | 226. |
| 1 | #BEFORE# | 27 | 227. |
| 1 | #BEFORE# | 28 | 228. |
| 1 | #BEFORE# | 29 | 229. |
| 1 | #BEFORE# | 30 | 230. |
| 1 | #BEFORE# | 31 | 231. |
| 1 | #BEFORE# | 32 | 232. |
| 1 | #BEFORE# | 33 | 233. |
| 1 | #BEFORE# | 34 | 234. |
| 1 | #BEFORE# | 35 | 235. |
| 1 | #BEFORE# | 36 | 236. |

|   |          |    |      |
|---|----------|----|------|
| 1 | #AFTER # | 1  | 201. |
| 1 | #AFTER # | 2  | 202. |
| 1 | #AFTER # | 3  | 203. |
| 1 | #AFTER # | 4  | 204. |
| 1 | #AFTER # | 5  | 205. |
| 1 | #AFTER # | 6  | 206. |
| 1 | #AFTER # | 7  | 207. |
| 1 | #AFTER # | 8  | 208. |
| 1 | #AFTER # | 9  | 209. |
| 1 | #AFTER # | 10 | 210. |
| 1 | #AFTER # | 11 | 211. |
| 1 | #AFTER # | 12 | 212. |
| 1 | #AFTER # | 13 | 213. |
| 1 | #AFTER # | 14 | 214. |
| 1 | #AFTER # | 15 | 215. |
| 1 | #AFTER # | 16 | 216. |
| 1 | #AFTER # | 17 | 217. |
| 1 | #AFTER # | 18 | 218. |
| 1 | #AFTER # | 19 | 219. |
| 1 | #AFTER # | 20 | 220. |
| 1 | #AFTER # | 21 | 221. |
| 1 | #AFTER # | 22 | 222. |
| 1 | #AFTER # | 23 | 223. |
| 1 | #AFTER # | 24 | 224. |
| 1 | #AFTER # | 25 | 225. |
| 1 | #AFTER # | 26 | 101. |
| 1 | #AFTER # | 27 | 102. |
| 1 | #AFTER # | 28 | 103. |
| 1 | #AFTER # | 29 | 104. |
| 1 | #AFTER # | 30 | 105. |
| 1 | #AFTER # | 31 | 106. |
| 1 | #AFTER # | 32 | 107. |
| 1 | #AFTER # | 33 | 108. |
| 1 | #AFTER # | 34 | 109. |
| 1 | #AFTER # | 35 | 110. |
| 1 | #AFTER # | 36 | 111. |

t1

## Ex.2: Send-Recv an Array (2/4)

```
if (my_rank.eq.0) then
  call MPI_Isend (VEC( 1) , 11, MPI_DOUBLE_PRECISION, 1, ..., req_send, ...)
  call MPI_Irecv (VEC(12) , 25, MPI_DOUBLE_PRECISION, 1, ..., req_recv, ...)
endif

if (my_rank.eq.1) then
  call MPI_Isend (VEC( 1) , 25, MPI_DOUBLE_PRECISION, 0, ..., req_send, ...)
  call MPI_Irecv (VEC(26) , 11, MPI_DOUBLE_PRECISION, 0, ..., req_recv, ...)
endif

call MPI_Waitall (... , req_recv, stat_recv, ...)
call MPI_Waitall (... , req_send, stat_send, ...)
```

It works, but complicated operations.  
Not looks like SPMD.  
Not portable.

t1

## Ex.2: Send-Recv an Array (3/4)

```
if (my_rank.eq.0) then
  NEIB= 1
  start_send= 1
  length_send= 11
  start_recv= length_send + 1
  length_recv= 25
endif

if (my_rank.eq.1) then
  NEIB= 0
  start_send= 1
  length_send= 25
  start_recv= length_send + 1
  length_recv= 11
endif

call MPI_Isend
  (VEC(start_send), length_send, MPI_DOUBLE_PRECISION, NEIB, ..., req_send, ...) &
call MPI_Irecv
  (VEC(start_recv), length_recv, MPI_DOUBLE_PRECISION, NEIB, ..., req_recv, ...) &

call MPI_Waitall (... , req_recv, stat_recv, ...)
call MPI_Waitall (... , req_send, stat_send, ...)
```

This is “SMPD” !!

t1

## Ex.2: Send-Recv an Array (4/4)

```
if (my_rank.eq.0) then
  NEIB= 1
  start_send= 1
  length_send= 11
  start_recv= length_send + 1
  length_recv= 25
endif

if (my_rank.eq.1) then
  NEIB= 0
  start_send= 1
  length_send= 25
  start_recv= length_send + 1
  length_recv= 11
endif

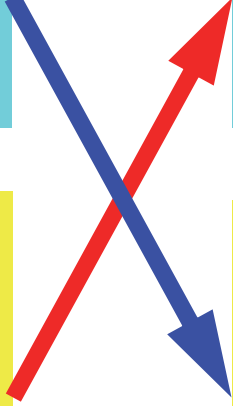
call MPI_Sendrecv
(VEC(start_send),length_send,MPI_DOUBLE_PRECISION,NEIB,...
 VEC(start_recv),length_recv,MPI_DOUBLE_PRECISION,NEIB,..., status,...)
&
&
```

# Notice: Send/Recv Arrays

**t1**

```
#PE0
send:
    VEC(start_send) ~
    VEC(start_send+length_send-1)
```

```
#PE1
send:
    VEC(start_send) ~
    VEC(start_send+length_send-1)
```



```
#PE0
recv:
    VEC(start_recv) ~
    VEC(start_recv+length_recv-1)
```

```
#PE1
recv:
    VEC(start_recv) ~
    VEC(start_recv+length_recv-1)
```

- “length\_send” of sending process must be equal to “length\_recv” of receiving process.
  - PE#0 to PE#1, PE#1 to PE#0
- “sendbuf” and “recvbuf”: different address



# Peer-to-Peer Communication

- What is P2P Communication ?
- 2D Problem, Generalized Communication Table
  - 2D FDM
  - Problem Setting
  - Distributed Local Data and Communication Table
  - Implementation
- Report S2

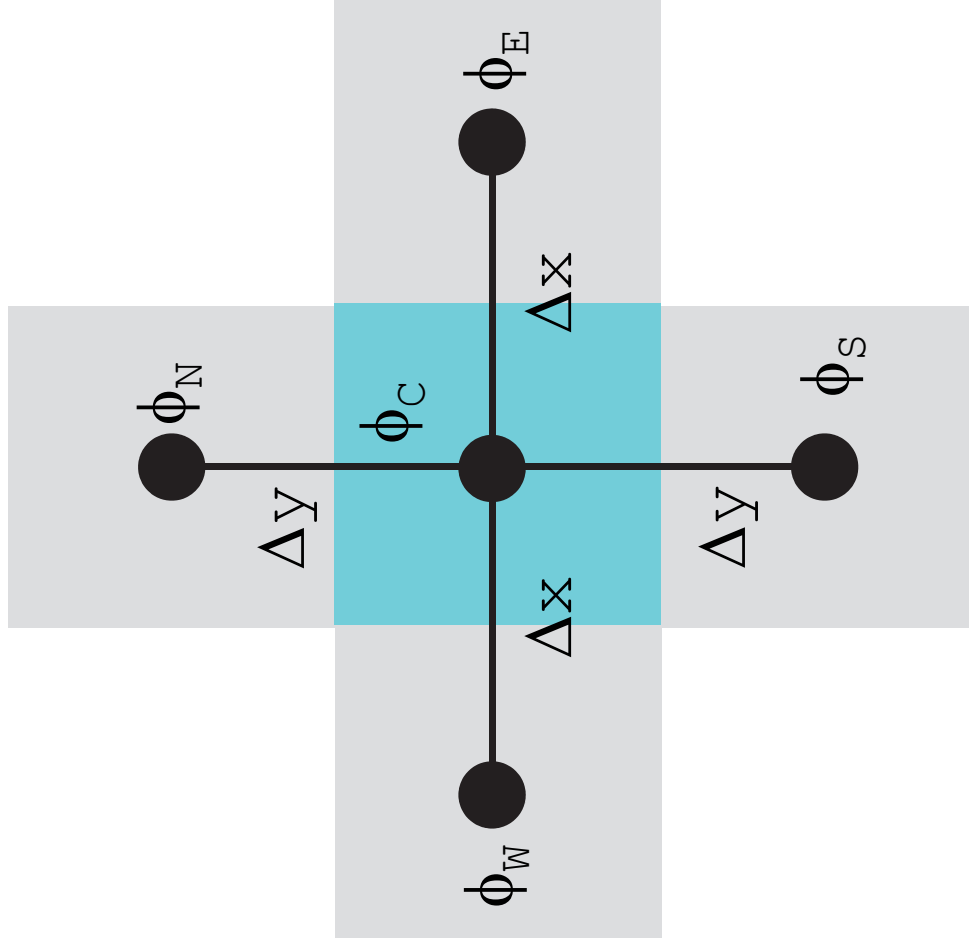
# MPI Programming

[illegible]

# 2D FDM (5-point, central difference)

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

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



# Decompose into 4 domains

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

# 4 domains: Global ID

|           |           |           |           |
|-----------|-----------|-----------|-----------|
| <u>57</u> | <u>58</u> | <u>59</u> | <u>60</u> |
| <u>49</u> | <u>50</u> | <u>51</u> | <u>52</u> |
| <u>41</u> | <u>42</u> | <u>43</u> | <u>44</u> |
| <u>33</u> | <u>34</u> | <u>35</u> | <u>36</u> |

PE#2

|           |           |           |           |
|-----------|-----------|-----------|-----------|
| <u>61</u> | <u>62</u> | <u>63</u> | <u>64</u> |
| <u>53</u> | <u>54</u> | <u>55</u> | <u>56</u> |
| <u>45</u> | <u>46</u> | <u>47</u> | <u>48</u> |
| <u>37</u> | <u>38</u> | <u>39</u> | <u>40</u> |

PE#3

|           |           |           |           |
|-----------|-----------|-----------|-----------|
| <u>25</u> | <u>26</u> | <u>27</u> | <u>28</u> |
| <u>17</u> | <u>18</u> | <u>19</u> | <u>20</u> |
| <u>9</u>  | <u>10</u> | <u>11</u> | <u>12</u> |
| <u>1</u>  | <u>2</u>  | <u>3</u>  | <u>4</u>  |

PE#0

|           |           |           |           |
|-----------|-----------|-----------|-----------|
| <u>29</u> | <u>30</u> | <u>31</u> | <u>32</u> |
| <u>21</u> | <u>22</u> | <u>23</u> | <u>24</u> |
| <u>13</u> | <u>14</u> | <u>15</u> | <u>16</u> |
| <u>5</u>  | <u>6</u>  | <u>7</u>  | <u>8</u>  |

PE#1

# 4 domains: Local ID

PE#2

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

PE#3

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

PE#0

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

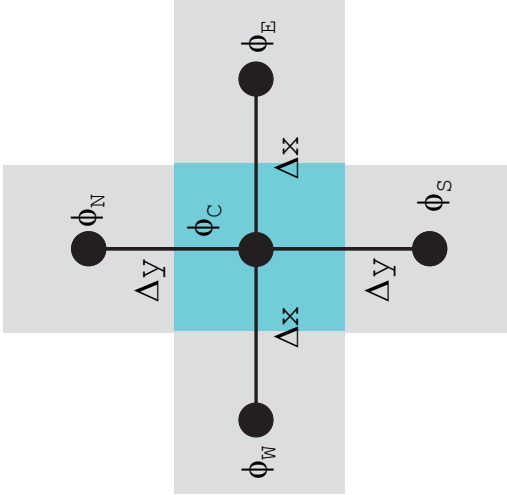
PE#1

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

# External Points: Overlapped Region

PE#2

PE#3

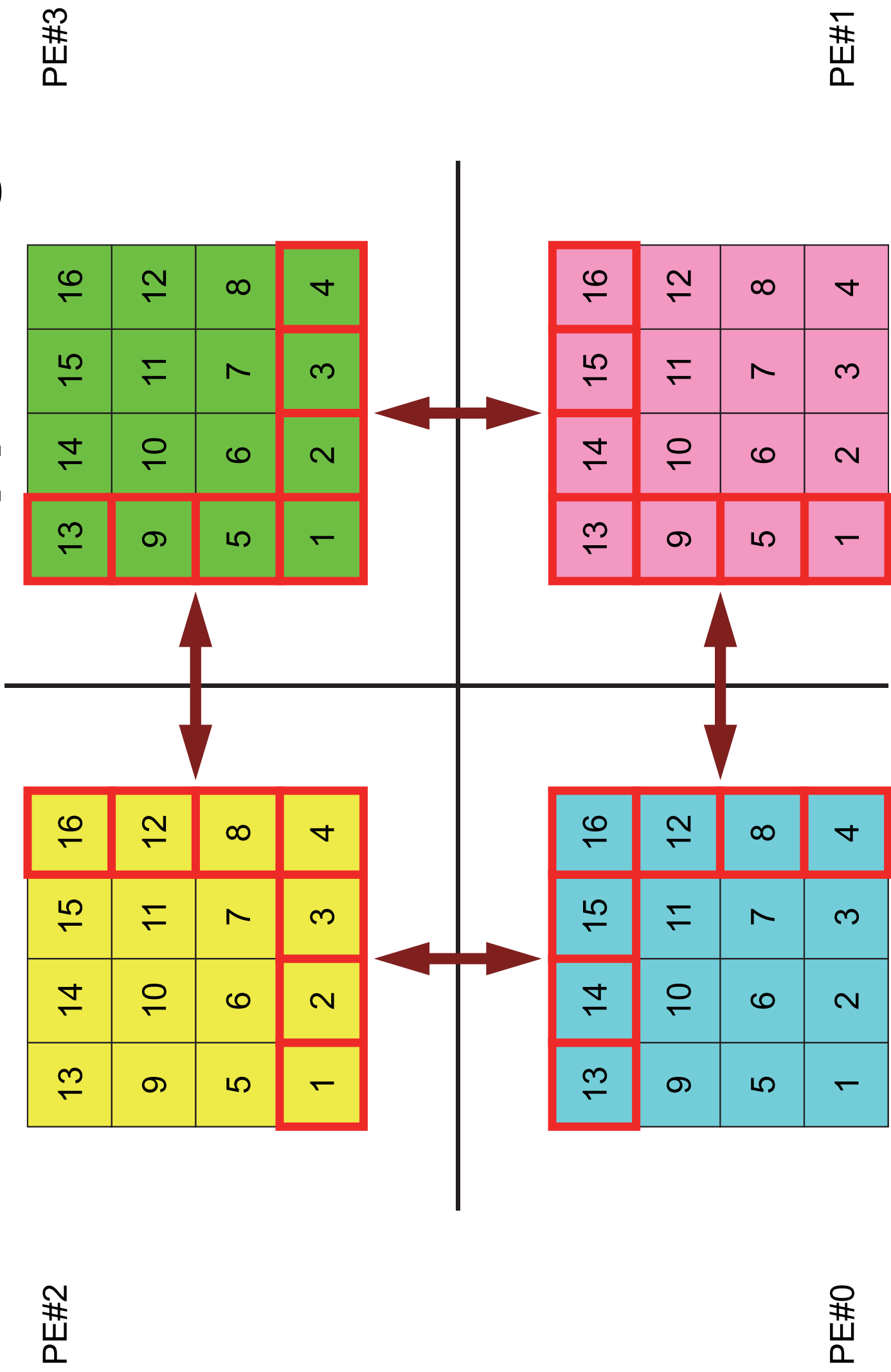


|    |    |    |    |    |    |    |    |
|----|----|----|----|----|----|----|----|
| 13 | 14 | 15 | 16 | 13 | 14 | 15 | 16 |
| 9  | 10 | 11 | 12 | 9  | 10 | 11 | 12 |
| 5  | 6  | 7  | 8  | 5  | 6  | 7  | 8  |
| 1  | 2  | 3  | 4  | 1  | 2  | 3  | 4  |
| 13 | 14 | 15 | 16 | 13 | 14 | 15 | 16 |
| 9  |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

PE#0

PE#1

# External Points: Overlapped Region

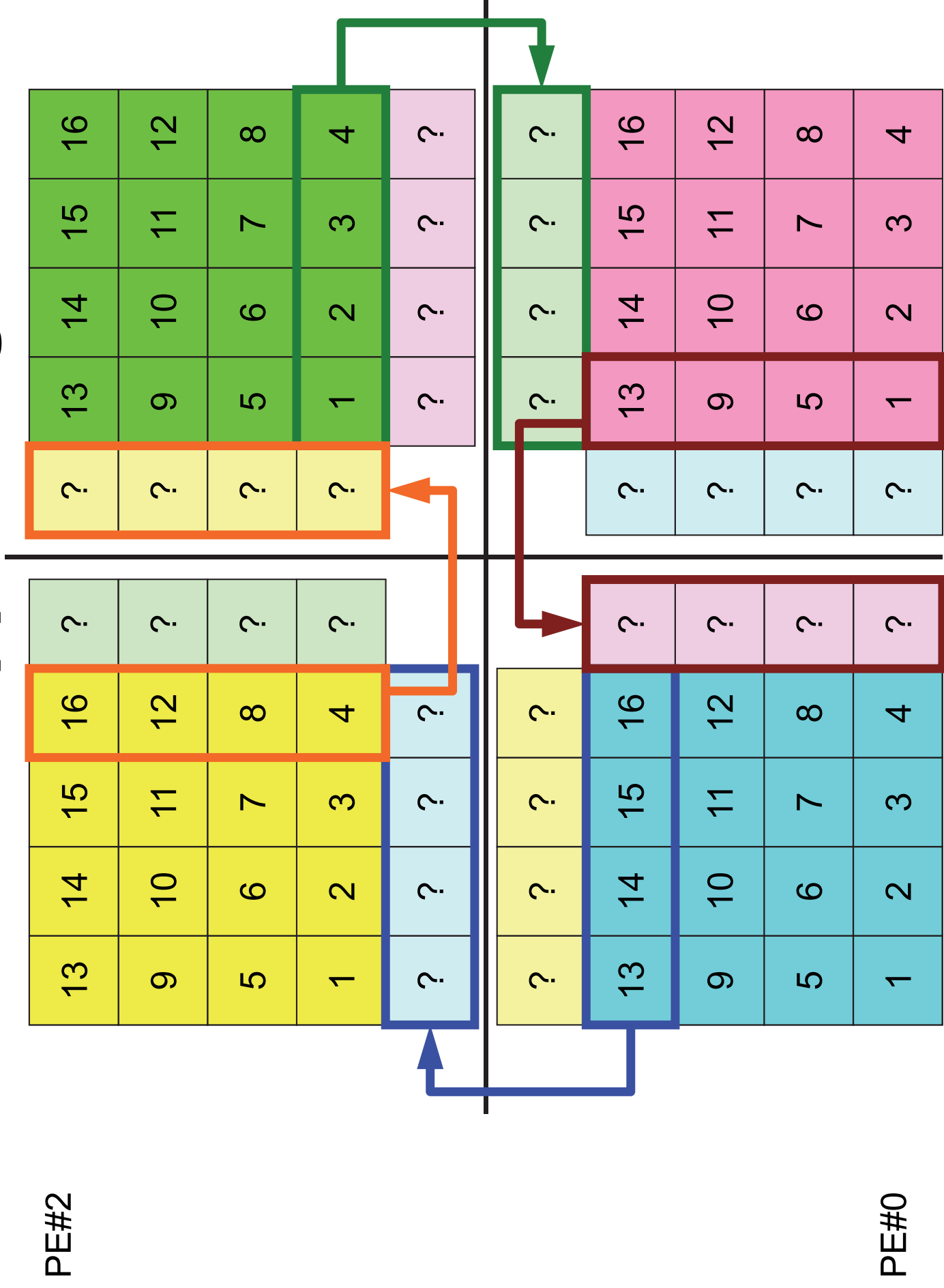




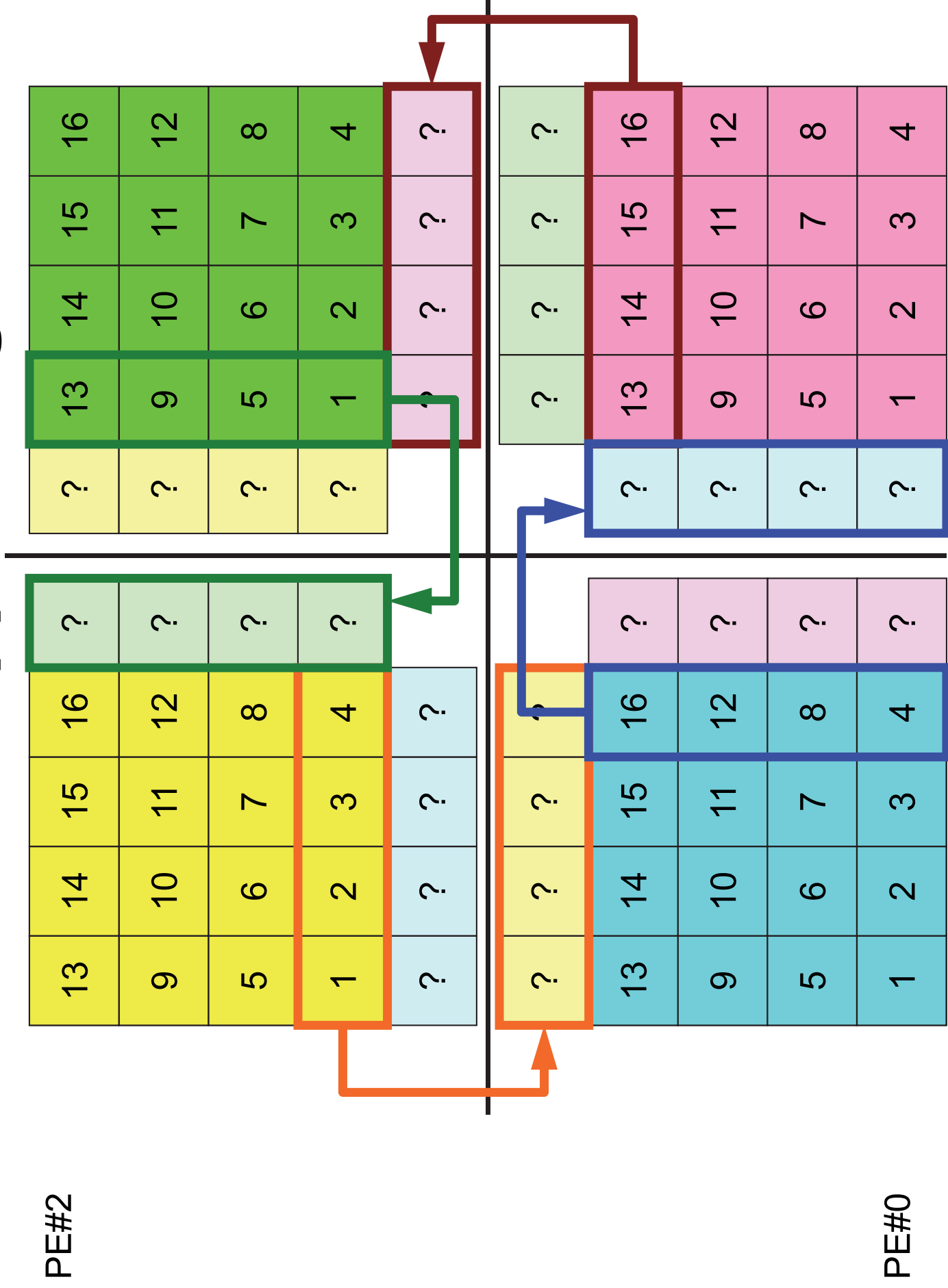
# Local ID of External Points ?



# Overlapped Region



# Overlapped Region



# Peer-to-Peer Communication

- What is P2P Communication ?
- 2D Problem, Generalized Communication Table
  - 2D FDM
  - Problem Setting
  - Distributed Local Data and Communication Table
  - Implementation
- Report S2

# Problem Setting: 2D FDM

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

- 2D region with 64 meshes (8x8)
- Each mesh has global ID from 1 to 64
  - In this example, this global ID is considered as dependent variable, such as temperature, pressure etc.
  - Something like computed results

# Problem Setting: Distributed Local Data

PE#2

|    |    |    |    |
|----|----|----|----|
| 57 | 58 | 59 | 60 |
| 49 | 50 | 51 | 52 |
| 41 | 42 | 43 | 44 |
| 33 | 34 | 35 | 36 |

PE#3

|    |    |    |    |
|----|----|----|----|
| 61 | 62 | 63 | 64 |
| 53 | 54 | 55 | 56 |
| 45 | 46 | 47 | 48 |
| 37 | 38 | 39 | 40 |

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

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

PE#0

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

PE#1

PE#2

|    |    |    |    |  |
|----|----|----|----|--|
| 57 | 58 | 59 | 60 |  |
| 49 | 50 | 51 | 52 |  |
| 41 | 42 | 43 | 44 |  |
| 33 | 34 | 35 | 36 |  |

PE#3

|  |    |    |    |    |
|--|----|----|----|----|
|  | 61 | 62 | 63 | 64 |
|  | 53 | 54 | 55 | 56 |
|  | 45 | 46 | 47 | 48 |
|  | 37 | 38 | 39 | 40 |

PE#0

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

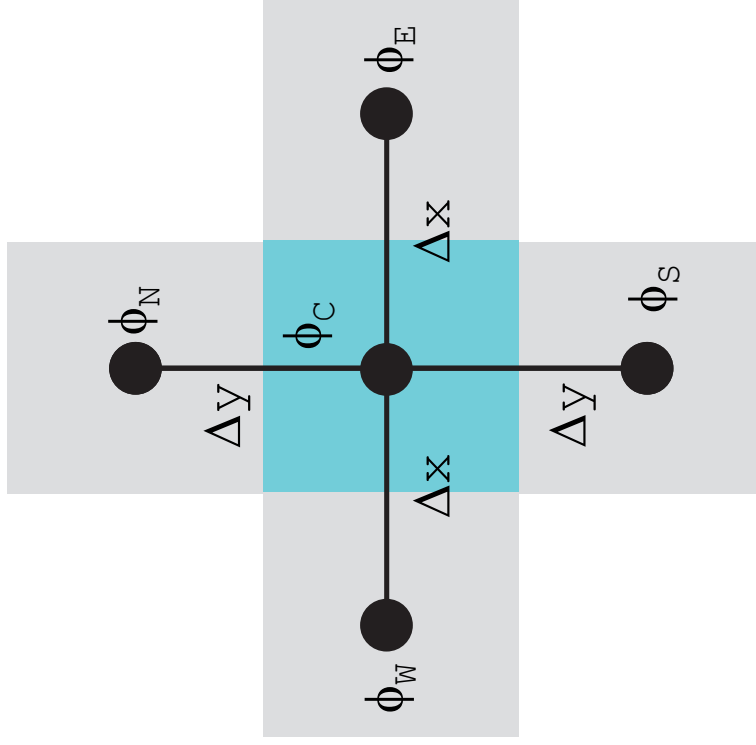
PE#1

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

# Operations of 2D FDM

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

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

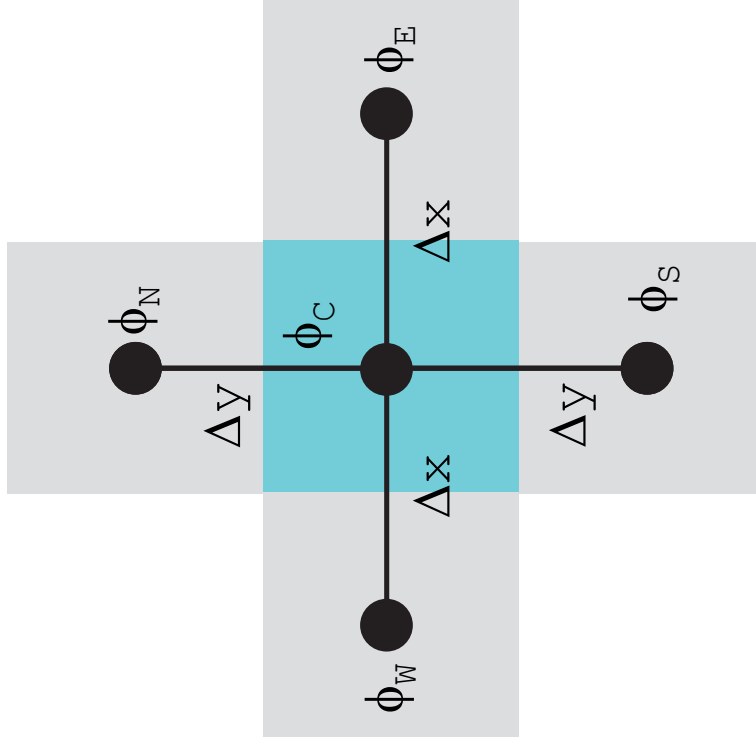


|           |           |           |           |           |           |           |           |
|-----------|-----------|-----------|-----------|-----------|-----------|-----------|-----------|
| <u>57</u> | <u>58</u> | <u>59</u> | <u>60</u> | <u>61</u> | <u>62</u> | <u>63</u> | <u>64</u> |
| <u>49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

# Operations of 2D FDM

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

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



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



# Computation (1/3)

| <u>PE#2</u> | <u>57</u> | <u>58</u> | <u>59</u> | <u>60</u> | <u>61</u> | <u>62</u> | <u>63</u> | <u>64</u> | <u>PE#3</u> |
|-------------|-----------|-----------|-----------|-----------|-----------|-----------|-----------|-----------|-------------|
|             | 49        | 50        | 51        | 52        | 53        | 54        | 55        | 56        |             |
|             | 41        | 42        | 43        | 44        | 45        | 46        | 47        | 48        |             |
|             | 33        | 34        | 35        | 36        | 37        | 38        | 39        | 40        |             |
|             | 25        | 26        | 27        | 28        | 29        | 30        | 31        | 32        |             |
|             | 17        | 18        | 19        | 20        | 21        | 22        | 23        | 24        |             |
|             | 9         | 10        | 11        | 12        | 13        | 14        | 15        | 16        |             |
| <u>PE#0</u> | 1         | 2         | 3         | 4         | 5         | 6         | 7         | 8         | <u>PE#1</u> |

- On each PE, info. of internal pts ( $i=1-N(=16)$ ) are read from distributed local data, info. of boundary pts are sent to neighbors, and they are received as info. of external pts.

# Computation (2/3): Before Send/Recv

|              |               |              |
|--------------|---------------|--------------|
| 1: <u>33</u> | 9: <u>49</u>  | 17: <u>?</u> |
| 2: <u>34</u> | 10: <u>50</u> | 18: <u>?</u> |
| 3: <u>35</u> | 11: <u>51</u> | 19: <u>?</u> |
| 4: <u>36</u> | 12: <u>52</u> | 20: <u>?</u> |
| 5: <u>41</u> | 13: <u>57</u> | 21: <u>?</u> |
| 6: <u>42</u> | 14: <u>58</u> | 22: <u>?</u> |
| 7: <u>43</u> | 15: <u>59</u> | 23: <u>?</u> |
| 8: <u>44</u> | 16: <u>60</u> | 24: <u>?</u> |

PE#2

|           |           |           |           |  |
|-----------|-----------|-----------|-----------|--|
| <u>57</u> | <u>58</u> | <u>59</u> | <u>60</u> |  |
| <u>49</u> | <u>50</u> | <u>51</u> | <u>52</u> |  |
| <u>41</u> | <u>42</u> | <u>43</u> | <u>44</u> |  |
| <u>33</u> | <u>34</u> | <u>35</u> | <u>36</u> |  |
|           |           |           |           |  |

PE#3

|  |           |           |           |           |
|--|-----------|-----------|-----------|-----------|
|  | <u>61</u> | <u>62</u> | <u>63</u> | <u>64</u> |
|  | <u>53</u> | <u>54</u> | <u>55</u> | <u>56</u> |
|  | <u>45</u> | <u>46</u> | <u>47</u> | <u>48</u> |
|  | <u>37</u> | <u>38</u> | <u>39</u> | <u>40</u> |
|  |           |           |           |           |

|              |               |              |
|--------------|---------------|--------------|
| 1: <u>37</u> | 9: <u>53</u>  | 17: <u>?</u> |
| 2: <u>38</u> | 10: <u>54</u> | 18: <u>?</u> |
| 3: <u>39</u> | 11: <u>55</u> | 19: <u>?</u> |
| 4: <u>40</u> | 12: <u>56</u> | 20: <u>?</u> |
| 5: <u>45</u> | 13: <u>61</u> | 21: <u>?</u> |
| 6: <u>46</u> | 14: <u>62</u> | 22: <u>?</u> |
| 7: <u>47</u> | 15: <u>63</u> | 23: <u>?</u> |
| 8: <u>48</u> | 16: <u>64</u> | 24: <u>?</u> |

|              |               |              |
|--------------|---------------|--------------|
| 1: <u>1</u>  | 9: <u>17</u>  | 17: <u>?</u> |
| 2: <u>2</u>  | 10: <u>18</u> | 18: <u>?</u> |
| 3: <u>3</u>  | 11: <u>19</u> | 19: <u>?</u> |
| 4: <u>4</u>  | 12: <u>20</u> | 20: <u>?</u> |
| 5: <u>9</u>  | 13: <u>25</u> | 21: <u>?</u> |
| 6: <u>10</u> | 14: <u>26</u> | 22: <u>?</u> |
| 7: <u>11</u> | 15: <u>27</u> | 23: <u>?</u> |
| 8: <u>12</u> | 16: <u>28</u> | 24: <u>?</u> |

PE#0

|           |           |           |           |  |
|-----------|-----------|-----------|-----------|--|
|           |           |           |           |  |
| <u>25</u> | <u>26</u> | <u>27</u> | <u>28</u> |  |
| <u>17</u> | <u>18</u> | <u>19</u> | <u>20</u> |  |
| <u>9</u>  | <u>10</u> | <u>11</u> | <u>12</u> |  |
| <u>1</u>  | <u>2</u>  | <u>3</u>  | <u>4</u>  |  |

PE#1

|  |           |           |           |           |
|--|-----------|-----------|-----------|-----------|
|  |           |           |           |           |
|  | <u>29</u> | <u>30</u> | <u>31</u> | <u>32</u> |
|  | <u>21</u> | <u>22</u> | <u>23</u> | <u>24</u> |
|  | <u>13</u> | <u>14</u> | <u>15</u> | <u>16</u> |
|  | <u>5</u>  | <u>6</u>  | <u>7</u>  | <u>8</u>  |

|              |               |              |
|--------------|---------------|--------------|
| 1: <u>5</u>  | 9: <u>21</u>  | 17: <u>?</u> |
| 2: <u>6</u>  | 10: <u>22</u> | 18: <u>?</u> |
| 3: <u>7</u>  | 11: <u>23</u> | 19: <u>?</u> |
| 4: <u>8</u>  | 12: <u>24</u> | 20: <u>?</u> |
| 5: <u>13</u> | 13: <u>29</u> | 21: <u>?</u> |
| 6: <u>14</u> | 14: <u>30</u> | 22: <u>?</u> |
| 7: <u>15</u> | 15: <u>31</u> | 23: <u>?</u> |
| 8: <u>16</u> | 16: <u>32</u> | 24: <u>?</u> |

# Computation (2/3): Before Send/Recv

|    |           |     |           |     |          |
|----|-----------|-----|-----------|-----|----------|
| 1: | <u>33</u> | 9:  | <u>49</u> | 17: | <u>?</u> |
| 2: | <u>34</u> | 10: | <u>50</u> | 18: | <u>?</u> |
| 3: | <u>35</u> | 11: | <u>51</u> | 19: | <u>?</u> |
| 4: | <u>36</u> | 12: | <u>52</u> | 20: | <u>?</u> |
| 5: | <u>41</u> | 13: | <u>57</u> | 21: | <u>?</u> |
| 6: | <u>42</u> | 14: | <u>58</u> | 22: | <u>?</u> |
| 7: | <u>43</u> | 15: | <u>59</u> | 23: | <u>?</u> |
| 8: | <u>44</u> | 16: | <u>60</u> | 24: | <u>?</u> |

|    |           |     |           |     |          |
|----|-----------|-----|-----------|-----|----------|
| 1: | <u>1</u>  | 9:  | <u>17</u> | 17: | <u>?</u> |
| 2: | <u>2</u>  | 10: | <u>18</u> | 18: | <u>?</u> |
| 3: | <u>3</u>  | 11: | <u>19</u> | 19: | <u>?</u> |
| 4: | <u>4</u>  | 12: | <u>20</u> | 20: | <u>?</u> |
| 5: | <u>9</u>  | 13: | <u>25</u> | 21: | <u>?</u> |
| 6: | <u>10</u> | 14: | <u>26</u> | 22: | <u>?</u> |
| 7: | <u>11</u> | 15: | <u>27</u> | 23: | <u>?</u> |
| 8: | <u>12</u> | 16: | <u>28</u> | 24: | <u>?</u> |

**PE#2**

|    |    |    |    |  |  |
|----|----|----|----|--|--|
| 57 | 58 | 59 | 60 |  |  |
| 49 | 50 | 51 | 52 |  |  |
| 41 | 42 | 43 | 44 |  |  |
| 33 | 34 | 35 | 36 |  |  |

**PE#3**

|  |  |    |    |    |    |
|--|--|----|----|----|----|
|  |  | 61 | 62 | 63 | 64 |
|  |  | 53 | 54 | 55 | 56 |
|  |  | 45 | 46 | 47 | 48 |
|  |  | 37 | 38 | 39 | 40 |

**PE#0**

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

**PE#1**

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

|    |           |     |           |     |          |
|----|-----------|-----|-----------|-----|----------|
| 1: | <u>37</u> | 9:  | <u>53</u> | 17: | <u>?</u> |
| 2: | <u>38</u> | 10: | <u>54</u> | 18: | <u>?</u> |
| 3: | <u>39</u> | 11: | <u>55</u> | 19: | <u>?</u> |
| 4: | <u>40</u> | 12: | <u>56</u> | 20: | <u>?</u> |
| 5: | <u>45</u> | 13: | <u>61</u> | 21: | <u>?</u> |
| 6: | <u>46</u> | 14: | <u>62</u> | 22: | <u>?</u> |
| 7: | <u>47</u> | 15: | <u>63</u> | 23: | <u>?</u> |
| 8: | <u>48</u> | 16: | <u>64</u> | 24: | <u>?</u> |

|    |           |     |           |     |          |
|----|-----------|-----|-----------|-----|----------|
| 1: | <u>5</u>  | 9:  | <u>21</u> | 17: | <u>?</u> |
| 2: | <u>6</u>  | 10: | <u>22</u> | 18: | <u>?</u> |
| 3: | <u>7</u>  | 11: | <u>23</u> | 19: | <u>?</u> |
| 4: | <u>8</u>  | 12: | <u>24</u> | 20: | <u>?</u> |
| 5: | <u>13</u> | 13: | <u>29</u> | 21: | <u>?</u> |
| 6: | <u>14</u> | 14: | <u>30</u> | 22: | <u>?</u> |
| 7: | <u>15</u> | 15: | <u>31</u> | 23: | <u>?</u> |
| 8: | <u>16</u> | 16: | <u>32</u> | 24: | <u>?</u> |

# Computation (3/3): After Send/Recv

|              |               |               |
|--------------|---------------|---------------|
| 1: <u>33</u> | 9: <u>49</u>  | 17: <u>37</u> |
| 2: <u>34</u> | 10: <u>50</u> | 18: <u>45</u> |
| 3: <u>35</u> | 11: <u>51</u> | 19: <u>53</u> |
| 4: <u>36</u> | 12: <u>52</u> | 20: <u>61</u> |
| 5: <u>41</u> | 13: <u>57</u> | 21: <u>25</u> |
| 6: <u>42</u> | 14: <u>58</u> | 22: <u>26</u> |
| 7: <u>43</u> | 15: <u>59</u> | 23: <u>27</u> |
| 8: <u>44</u> | 16: <u>60</u> | 24: <u>28</u> |

PE#2

|           |           |           |           |           |
|-----------|-----------|-----------|-----------|-----------|
| <u>57</u> | <u>58</u> | <u>59</u> | <u>60</u> | <u>61</u> |
| <u>49</u> | <u>50</u> | <u>51</u> | <u>52</u> | <u>53</u> |
| <u>41</u> | <u>42</u> | <u>43</u> | <u>44</u> | <u>45</u> |
| <u>33</u> | <u>34</u> | <u>35</u> | <u>36</u> | <u>37</u> |
| <u>25</u> | <u>26</u> | <u>27</u> | <u>28</u> |           |

|              |               |               |
|--------------|---------------|---------------|
| 1: <u>37</u> | 9: <u>53</u>  | 17: <u>36</u> |
| 2: <u>38</u> | 10: <u>54</u> | 18: <u>44</u> |
| 3: <u>39</u> | 11: <u>55</u> | 19: <u>52</u> |
| 4: <u>40</u> | 12: <u>56</u> | 20: <u>60</u> |
| 5: <u>45</u> | 13: <u>61</u> | 21: <u>29</u> |
| 6: <u>46</u> | 14: <u>62</u> | 22: <u>30</u> |
| 7: <u>47</u> | 15: <u>63</u> | 23: <u>31</u> |
| 8: <u>48</u> | 16: <u>64</u> | 24: <u>32</u> |

PE#3

|           |           |           |           |           |
|-----------|-----------|-----------|-----------|-----------|
| <u>60</u> | <u>61</u> | <u>62</u> | <u>63</u> | <u>64</u> |
| <u>52</u> | <u>53</u> | <u>54</u> | <u>55</u> | <u>56</u> |
| <u>44</u> | <u>45</u> | <u>46</u> | <u>47</u> | <u>48</u> |
| <u>36</u> | <u>37</u> | <u>38</u> | <u>39</u> | <u>40</u> |
|           | <u>29</u> | <u>30</u> | <u>31</u> | <u>32</u> |

|              |               |               |
|--------------|---------------|---------------|
| 1: <u>1</u>  | 9: <u>17</u>  | 17: <u>5</u>  |
| 2: <u>2</u>  | 10: <u>18</u> | 18: <u>14</u> |
| 3: <u>3</u>  | 11: <u>19</u> | 19: <u>21</u> |
| 4: <u>4</u>  | 12: <u>20</u> | 20: <u>29</u> |
| 5: <u>9</u>  | 13: <u>25</u> | 21: <u>33</u> |
| 6: <u>10</u> | 14: <u>26</u> | 22: <u>34</u> |
| 7: <u>11</u> | 15: <u>27</u> | 23: <u>35</u> |
| 8: <u>12</u> | 16: <u>28</u> | 24: <u>36</u> |

PE#0

|           |           |           |           |
|-----------|-----------|-----------|-----------|
| <u>33</u> | <u>34</u> | <u>35</u> | <u>36</u> |
| <u>25</u> | <u>26</u> | <u>27</u> | <u>28</u> |
| <u>17</u> | <u>18</u> | <u>19</u> | <u>20</u> |
| <u>9</u>  | <u>10</u> | <u>11</u> | <u>12</u> |
| <u>1</u>  | <u>2</u>  | <u>3</u>  | <u>4</u>  |

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

PE#1

|              |               |               |
|--------------|---------------|---------------|
| 1: <u>5</u>  | 9: <u>21</u>  | 17: <u>4</u>  |
| 2: <u>6</u>  | 10: <u>22</u> | 18: <u>12</u> |
| 3: <u>7</u>  | 11: <u>23</u> | 19: <u>20</u> |
| 4: <u>8</u>  | 12: <u>24</u> | 20: <u>28</u> |
| 5: <u>13</u> | 13: <u>29</u> | 21: <u>37</u> |
| 6: <u>14</u> | 14: <u>30</u> | 22: <u>38</u> |
| 7: <u>15</u> | 15: <u>31</u> | 23: <u>39</u> |
| 8: <u>16</u> | 16: <u>32</u> | 24: <u>40</u> |

# Peer-to-Peer Communication

- What is P2P Communication ?
- 2D Problem, Generalized Communication Table
  - 2D FDM
  - Problem Setting
  - Distributed Local Data and Communication Table
  - Implementation
- Report S2

# Overview of Distributed Local Data

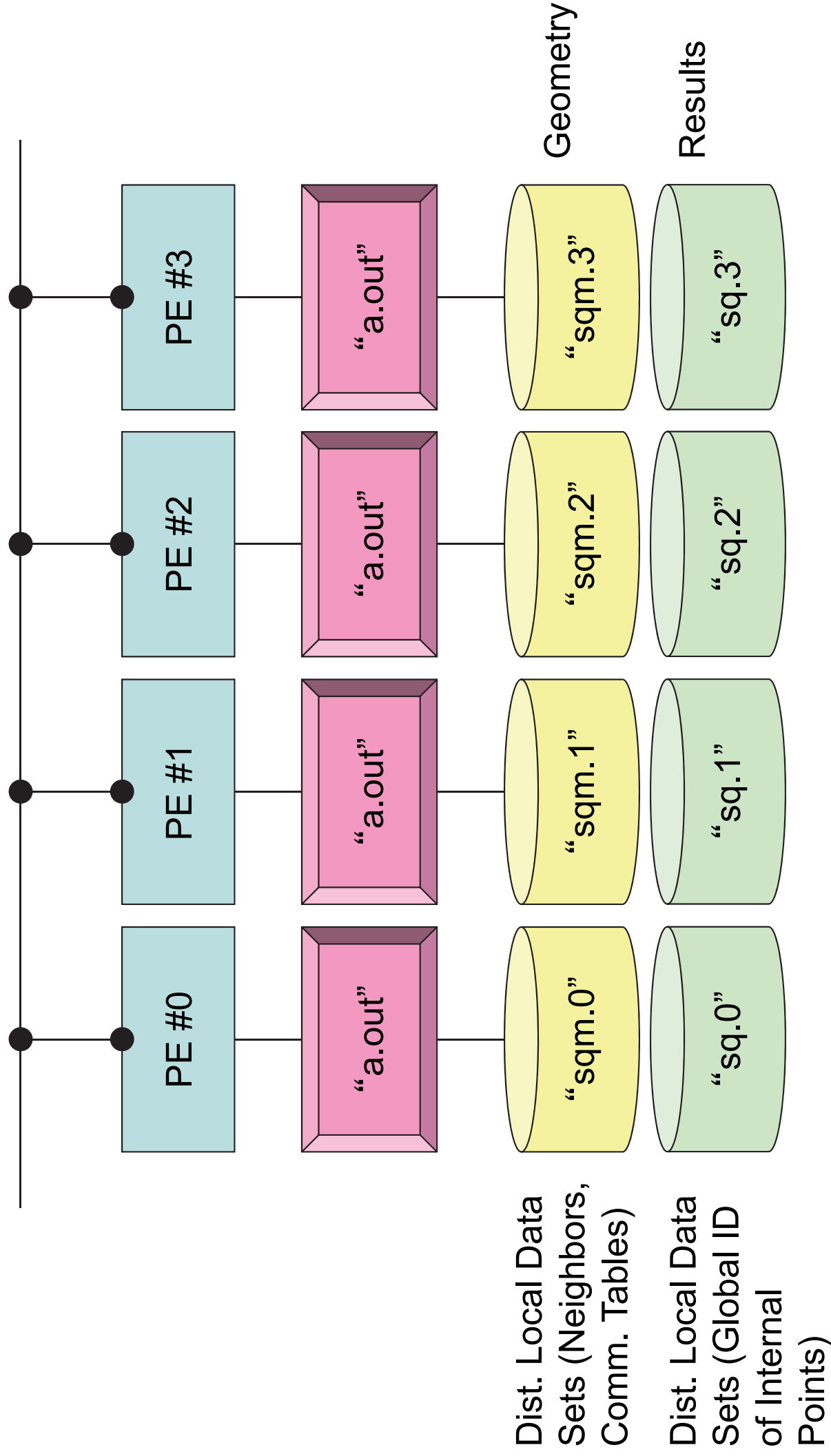
Example on PE#0

| <u>PE#2</u> |    |    |    | <u>PE#1</u> |  |  |  |
|-------------|----|----|----|-------------|--|--|--|
| <u>PE#0</u> |    |    |    |             |  |  |  |
|             | 25 | 26 | 27 | 28          |  |  |  |
|             | 17 | 18 | 19 | 20          |  |  |  |
|             | 9  | 10 | 11 | 12          |  |  |  |
|             | 1  | 2  | 3  | 4           |  |  |  |
|             | 13 | 14 | 15 | 16          |  |  |  |
|             | 9  | 10 | 11 | 12          |  |  |  |
|             | 5  | 6  | 7  | 8           |  |  |  |
|             | 1  | 2  | 3  | 4           |  |  |  |

Value at each mesh (= Global ID)

Local ID

# SPMD...



# 2D FDM: PE#0

Information at each domain (1/4)

## Internal Points

Meshes originally assigned to the domain

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



# 2D FDM: PE#0

## Information at each domain (2/4)

PE#2

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

PE#1

Internal Points

Meshes originally assigned to the domain

External Points

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

- Sleeves
- Halo



# 2D FDM: PE#0

## Information at each domain (3/4)

PE#2

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

PE#1

Internal Points

Meshes originally assigned to the domain

External Points

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

Boundary Points

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

# 2D FDM: PE#0

## Information at each domain (4/4)

PE#2

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

PE#1

Internal Points

Meshes originally assigned to the domain

External Points

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

Boundary Points

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

Relationships between Domains

Communication Table: External/Boundary Points  
Neighbors

# Description of Distributed Local Data

- Internal/External 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 ?

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

# Overview of Distributed Local Data

Example on PE#0

| <u>PE#2</u> |             |             |             | <u>PE#1</u> |             |             |             |
|-------------|-------------|-------------|-------------|-------------|-------------|-------------|-------------|
| <u>PE#0</u> | <u>PE#0</u> | <u>PE#0</u> | <u>PE#0</u> | <u>PE#0</u> | <u>PE#0</u> | <u>PE#0</u> | <u>PE#0</u> |
| 25          | 26          | 27          | 28          |             |             |             |             |
| 17          | 18          | 19          | 20          |             |             |             |             |
| 9           | 10          | 11          | 12          |             |             |             |             |
| 1           | 2           | 3           | 4           |             |             |             |             |

| <u>PE#2</u> |             |             |             | <u>PE#1</u> |             |             |             |
|-------------|-------------|-------------|-------------|-------------|-------------|-------------|-------------|
| <u>PE#0</u> | <u>PE#0</u> | <u>PE#0</u> | <u>PE#0</u> | <u>PE#0</u> | <u>PE#0</u> | <u>PE#0</u> | <u>PE#0</u> |
| 21          | 22          | 23          | 24          | 13          | 14          | 15          | 16          |
|             |             |             |             | 9           | 10          | 11          | 12          |
|             |             |             |             | 5           | 6           | 7           | 8           |
|             |             |             |             | 1           | 2           | 3           | 4           |
|             |             |             |             |             |             |             |             |
|             |             |             |             |             |             |             |             |
|             |             |             |             |             |             |             |             |
|             |             |             |             |             |             |             |             |

Value at each mesh (= Global ID)

Local ID

# 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\_Send/recv/Waitall Fortran

## SENDbuf



export\_index(0)+1    export\_index(1)+1    export\_index(2)+1    export\_index(3)+1    export\_index(4)

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

Copied to sending buffers

```
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_recv, ierr)
```

# Generalized Comm. Table: Receive

- Neighbors
  - NEIBPETOT, NEIBPE(neib)
- Message size for each neighbor
  - import\_index(neib), neib= 0, NEIBPETOT
- ID of **external** 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 Fortran

```

do neib= 1, NEIBPETOT
  iS_i= import_index(neib-1) + 1
  iE_i= import_index(neib )
  BUFlength_i=iE_i + 1 - iS_i

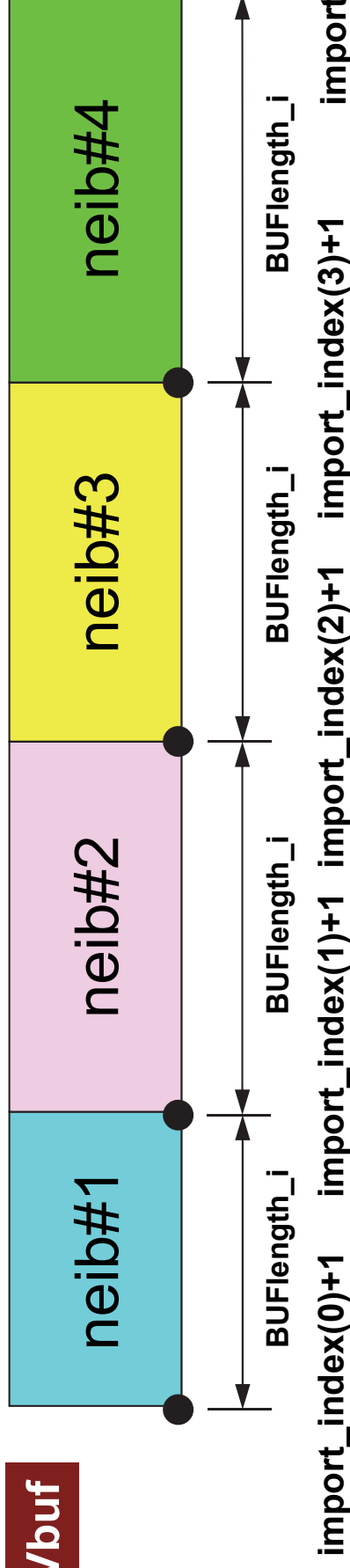
  call MPI_Irecv
    & (RECVbuf(iS_i) , BUFlength_i, MPI_INTEGER, NEIBPE(neib) , 0, &
    & MPI_COMM_WORLD, request_recv(neib) , ierr)
  enddo

  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



# Relationship SEND/RECV

```
do neib= 1, NEIBPETOT
  is_e = export_index(neib-1) + 1
  iE_e = export_index(neib )
  BUFlength_e= iE_e + 1 - is_e

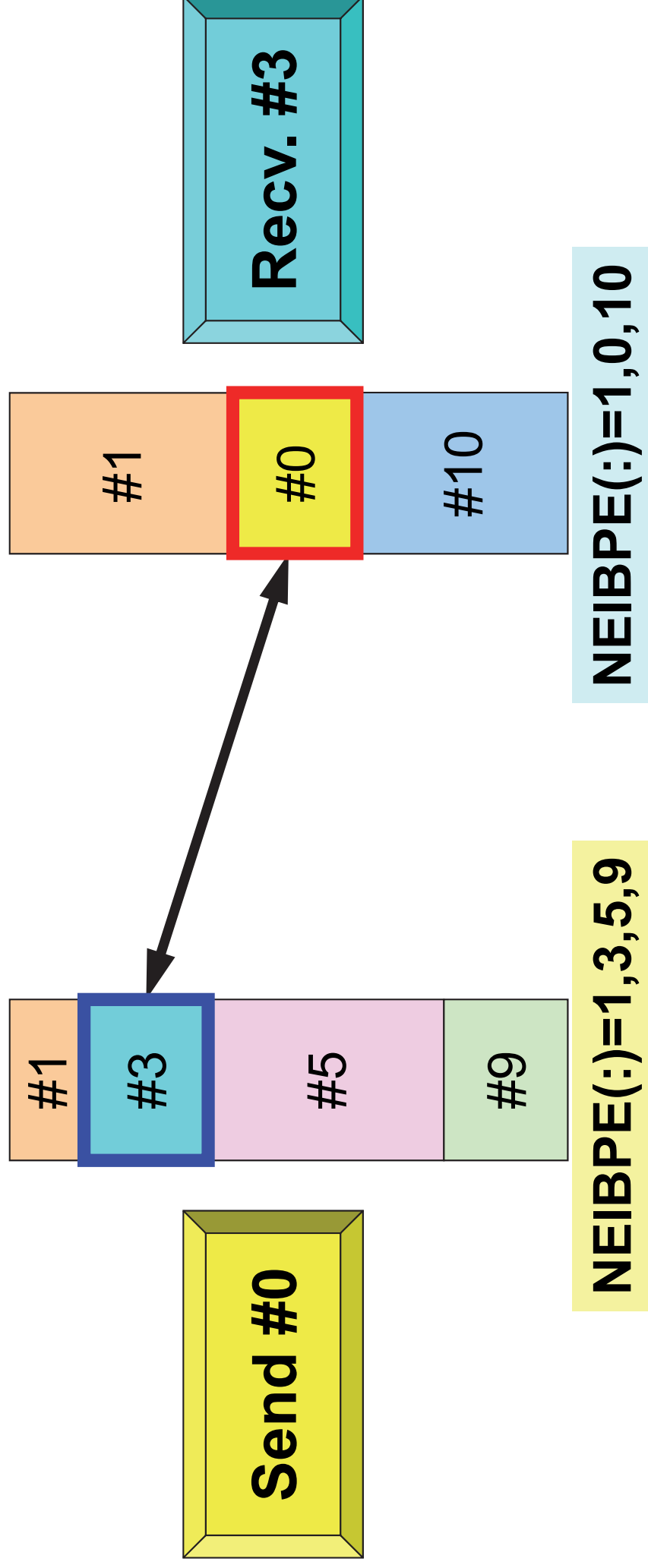
  call MPI_ISEND
    &      (SENDbuf(is_e), BUFlength_e, MPI_INTEGER, NEIBPE(neib), 0,&
    &      MPI_COMM_WORLD, request_send(neib), ierr)
enddo
```

```
do neib= 1, NEIBPETOT
  is_i = import_index(neib-1) + 1
  iE_i = import_index(neib )
  BUFlength_i= iE_i + 1 - is_i

  call MPI_IRECV
    &      (RECVbuf(is_i), BUFlength_i, MPI_INTEGER, NEIBPE(neib), 0,&
    &      MPI_COMM_WORLD, request_recv(neib), ierr)
enddo
```

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

# Relationship SEND/RECV (#0 to #3)



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

# Generalized Comm. Table (1/6)

PE#2

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

PE#1

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

# Generalized Comm. Table (2/6)

PE#2

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

PE#1

|               |                      |
|---------------|----------------------|
| #NEIBPEtot    | Number of neighbors  |
| 2             |                      |
| #NEIBPE       | ID of neighbors      |
| 1             |                      |
| 2             |                      |
| #NODE         |                      |
| 24            | Ext/Int Pts, Int Pts |
| 16            |                      |
| #IMPORT_index |                      |
| 4             |                      |
| 8             |                      |
| #IMPORT_items |                      |
| 17            |                      |
| 18            |                      |
| 19            |                      |
| 20            |                      |
| 21            |                      |
| 22            |                      |
| 23            |                      |
| 24            |                      |
| #EXPORT_index |                      |
| 4             |                      |
| 8             |                      |
| #EXPORT_items |                      |
| 4             |                      |
| 8             |                      |
| 12            |                      |
| 16            |                      |
| 13            |                      |
| 14            |                      |
| 15            |                      |
| 16            |                      |

# Generalized Comm. Table (3/6)

PE#2

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

PE#1

#NEIBPEtot  
2  
#NEIBPE  
1 2  
#NODE  
24 16  
#IMPORT\_index  
4 8  
#IMPORT\_items  
17  
18  
19  
20  
21  
22  
23  
24  
#EXPORT\_index  
4 8  
#EXPORT\_items  
4 8 12 16 13 14 15 16

Four ext pts (1<sup>st</sup>-4<sup>th</sup> items) are imported from 1<sup>st</sup> neighbor (PE#1), and four (5<sup>th</sup>-8<sup>th</sup> items) are from 2<sup>nd</sup> neighbor (PE#2).

# PE#2

# #E1

2

27

24 16

48

1711

20

24

84

4

21

0 0 0  
H L L

५ ८  
 ५ ८

9T

# Generalized Comm. Table (5/6)

PE#2

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

PE#1

#NEIBPEtot  
2  
#NEIBPE  
1 2  
#NODE  
24 16  
#IMPORT\_index  
4 8  
#IMPORT\_items  
17  
18  
19  
20  
21  
22  
23  
24  
#EXPORT\_index  
4 8  
#EXPORT\_items  
4 8 12 16 13 14 15 16

Four boundary pts (1<sup>st</sup>-4<sup>th</sup> items) are exported to 1<sup>st</sup> neighbor (PE#1), and four (5<sup>th</sup>-8<sup>th</sup> items) are to 2<sup>nd</sup> neighbor (PE#2).



# Generalized Comm. Table (6/6)

PE#2

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

PE#1

#NEIBPEtot  
2

#NEIBPE  
1 2

#NODE  
24 16

#IMPORT\_index  
4 8

#IMPORT\_items  
17  
18  
19  
20  
21  
22  
23  
24

#EXPORT\_index  
4 8

#EXPORT\_items  
4 8

exported to 1<sup>st</sup> Neighbor  
(PE#1) (1<sup>st</sup>-4<sup>th</sup> items)

exported to 2<sup>nd</sup> Neighbor  
(PE#2) (5<sup>th</sup>-8<sup>th</sup> items)

12  
16  
13  
14  
15  
16

# Generalized Comm. Table (6/6)

PE#2

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

PE#1

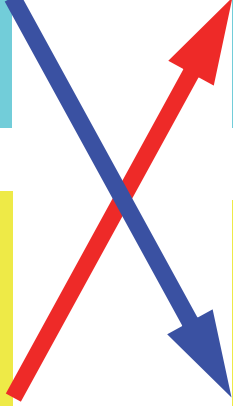
An external point is only sent from its original domain.

A boundary point could be referred from more than one domain, and sent to multiple domains (e.g. 16<sup>th</sup> mesh).

# Notice: Send/Recv Arrays

```
#PE0  
send:  
    VEC(start_send) ~  
    VEC(start_send+length_send-1)
```

```
#PE1  
send:  
    VEC(start_send) ~  
    VEC(start_send+length_send-1)
```



```
#PE0  
recv:  
    VEC(start_recv) ~  
    VEC(start_recv+length_recv-1)
```

```
#PE1  
recv:  
    VEC(start_recv) ~  
    VEC(start_recv+length_recv-1)
```

- “length\_send” of sending process must be equal to “length\_recv” of receiving process.
  - PE#0 to PE#1, PE#1 to PE#0
- “sendbuf” and “recvbuf”: different address

# Peer-to-Peer Communication

- What is P2P Communication ?
- 2D Problem, Generalized Communication Table
  - 2D FDM
  - Problem Setting
  - Distributed Local Data and Communication Table
  - Implementation
- Report S2

# Sample Program for 2D FDM

```
$ cd <$O-S2>

$ mpifrtpx -Kfast sq-sr1.f
$ mpifccpx -Kfast sq-sr1.c

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

# Example: sq-sr1.f (1/6)

Fortran

## Initialization

```

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

integer(kind=4) :: my_rank, PETOT
integer(kind=4) :: N, NP, NEIBPETOT, BUFlength

integer(kind=4), dimension(:), allocatable :: VAL
integer(kind=4), dimension(:), allocatable :: SENDbuf, RECVbuf
integer(kind=4), dimension(:), allocatable :: NEIBPE

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

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

character(len=80)      :: filename, line

!C +-----+
!C | INIT. MPI |
!C +-----+
!C===
      call MPI_INIT      (ierr)
      call MPI_COMM_SIZE (MPI_COMM_WORLD, PETOT, ierr )
      call MPI_COMM_RANK (MPI_COMM_WORLD, my_rank, ierr )

```

# Example: sq-sr1.f (2/6)

Fortran

Reading distributed local data files (sqm.\*)

```
!C
!C-- MESH
      if (my_rank.eq.0) filename= 'sqm.0'
      if (my_rank.eq.1) filename= 'sqm.1'
      if (my_rank.eq.2) filename= 'sqm.2'
      if (my_rank.eq.3) filename= 'sqm.3'
      open (21, file= filename, status= 'unknown')
      read (21,*) NEIBPETOT
      allocate (NEIBPE (NEIBPETOT))
      allocate (import_index (0:NEIBPETOT))
      allocate (export_index (0:NEIBPETOT))
      import_index= 0
      export_index= 0
      read (21,*) (NEIBPE(neib), neib= 1, NEIBPETOT)
      read (21,*) NP, N
      read (21, '(a80)') line
      read (21,*) (import_index(neib), neib= 1, NEIBPETOT)
      nn= import_index (NEIBPETOT)
      allocate (import_item(nn))

      do i= 1, nn
        read (21,*) import_item(i)
      enddo
      read (21, '(a80)') line
      read (21,*) (export_index(neib), neib= 1, NEIBPETOT)
      nn= export_index (NEIBPETOT)
      allocate (export_item(nn))

      do i= 1, nn
        read (21,*) export_item(i)
      enddo
      close (21)
```

# Example: sq-sr1.f (2/6)

Reading distributed local data files (sqm.\*)

Fortran

```
!C
!C-- MESH
      if (my_rank.eq.0) filename= 'sqm.0'
      if (my_rank.eq.1) filename= 'sqm.1'
      if (my_rank.eq.2) filename= 'sqm.2'
      if (my_rank.eq.3) filename= 'sqm.3'
      open (21, file= filename, status= 'unknown')
         read (21,*) NEIBPETOT
         allocate (NEIBPE (NEIBPETOT))
         allocate (import_index (0:NEIBPETOT))
         allocate (export_index (0:NEIBPETOT))
            import_index= 0
            export_index= 0
         read (21,*) (NEIBPE(neib), neib= 1, NEIBPETOT)
         read (21,*) NP, N

         read (21,*) (import_index(neib), neib= 1, NEIBPETOT)
         nn= import_index(NEIBPETOT)
         allocate (import_item(nn))

         do i= 1, nn
            read (21,*) import_item(i)
         enddo

         read (21,*) (export_index(neib), neib= 1, NEIBPETOT)
         nn= export_index(NEIBPETOT)
         allocate (export_item(nn))

         do i= 1, nn
            read (21,*) export_item(i)
         enddo
      close (21)
```

```
#NEIBPETOT
2
#NEIBPE
1 2
#NODE
24 16
#IMPORTindex
4 8
#IMPORTitems
17
18
19
20
21
22
23
24
#EXPORTindex
4 8
#EXPORTitems
4 8
12
16
13
14
15
16
```



# Example: sq-sr1.f (2/6)

Reading distributed local data files (sqm.\*)

Fortran

```
!C
!C-- MESH
      if (my_rank.eq.0) filename= 'sqm.0'
      if (my_rank.eq.1) filename= 'sqm.1'
      if (my_rank.eq.2) filename= 'sqm.2'
      if (my_rank.eq.3) filename= 'sqm.3'
      open (21, file= filename, status= 'unknown')
      read (21,*) NEIBPETOT
      allocate (NEIBPE (NEIBPETOT))
      allocate (import_index (0:NEIBPETOT))
      allocate (export_index (0:NEIBPETOT))
      import_index= 0
      export_index= 0
      read (21,*) (NEIBPE(neib), neib= 1, NEIBPETOT)
      read (21,*) NP, N
      read (21,*) (a80, a= 1, NP)
      NP Number of all meshes (internal + external)
      N Number of internal meshes
      do i= 1, nn
        read (21,*) import_item(i)
      enddo
      read (21, '(a80)') line
      read (21,*) (export_index(neib), neib= 1, NEIBPETOT)
      nn= export_index(NEIBPETOT)
      allocate (export_item(nn))
      do i= 1, nn
        read (21,*) export_item(i)
      enddo
      close (21)
```

```
#NEIBPETot      2
#NEIBPE         1  2
#NODE          24 16
#IMPORTIndex   4  8
#IMPORTItems  17
18
19
20
21
22
23
24 #EXPORTIndex  4  8
#EXPORTItems   4  8
4  8
12
16
13
14
15
16
```

# Example: sq-sr1.f (2/6)

Reading distributed local data files (sqm.\*)

```
!C
!C-- MESH
      if (my_rank.eq.0) filename= 'sqm.0'
      if (my_rank.eq.1) filename= 'sqm.1'
      if (my_rank.eq.2) filename= 'sqm.2'
      if (my_rank.eq.3) filename= 'sqm.3'
      open (21, file= filename, status= 'unknown')
      read (21,*) NEIBPETOT
      allocate (NEIBPE (NEIBPETOT))
      allocate (import_index(0:NEIBPETOT))
      allocate (export_index(0:NEIBPETOT))
      import_index= 0
      export_index= 0
      read (21,*) (NEIBPE(neib), neib= 1, NEIBPETOT)
      read (21,*) NP, N
      read (21,*) (import_index(neib), neib= 1, NEIBPETOT)
      nn= import_index(NEIBPETOT)
      allocate (import_item(nn))
      do i= 1, nn
        read (21,*) import_item(i)
      enddo

      read (21,*) (export_index(neib), neib= 1, NEIBPETOT)
      nn= export_index(NEIBPETOT)
      allocate (export_item(nn))
      do i= 1, nn
        read (21,*) export_item(i)
      enddo
      close (21)
```

```
#NEIBPETOT
2
#NEIBPE
1 2
#NODE
24 16
#IMPORTindex
4 8
#IMPORTitems
17
18
19
20
21
22
23
24
#EXPORTindex
4 8
#EXPORTitems
4 8
12
16
13
14
15
16
```

# Example: sq-sr1.f (2/6)

Reading distributed local data files (sqm.\*)

Fortran

```
!C
!C-- MESH
      if (my_rank.eq.0) filename= 'sqm.0'
      if (my_rank.eq.1) filename= 'sqm.1'
      if (my_rank.eq.2) filename= 'sqm.2'
      if (my_rank.eq.3) filename= 'sqm.3'
      open (21, file= filename, status= 'unknown')
      read (21,*) NEIBPETOT
      allocate (NEIBPE (NEIBPETOT))
      allocate (import_index(0:NEIBPETOT))
      allocate (export_index(0:NEIBPETOT))
      import_index= 0
      export_index= 0
      read (21,*) (NEIBPE(neib), neib= 1, NEIBPETOT)
      read (21,*) NP, N
      read (21,*) (import_index(neib), neib= 1, NEIBPETOT)
      nn= import_index(NEIBPETOT)
      allocate (import_item(nn))
      do i= 1, nn
      read (21,*) import_item(i)
      enddo

      read (21,*) (export_index(neib), neib= 1, NEIBPETOT)
      nn= export_index(NEIBPETOT)
      allocate (export_item(nn))

      do i= 1, nn
      read (21,*) export_item(i)
      enddo
      close (21)
```

```
#NEIBPETot
2
#NEIBPE
1 2
#NODE
24 16
#IMPORTindex
4 8
#IMPORTitems
17
18
19
20
21
22
23
24
#EXPORTindex
4 8
#EXPORTitems
4 8
12
16
13
14
15
16
```

# RECV/Import: PE#0

```
#NEIBPEtot
2
#NEIBPE
1 2
#NODE
24 16
#IMPORTindex
4 8
#IMPORTitems
17
18
19
20
21
22
23
24
#EXPORTindex
4 8
#EXPORTitems
4 8
12
16
13
14
15
16
```

| <u>PE#2</u> |    |    |    |    |    |    |    | <u>PE#1</u> |  |  |  |
|-------------|----|----|----|----|----|----|----|-------------|--|--|--|
| 21          | 22 | 23 | 24 |    |    |    |    |             |  |  |  |
| 13          | 14 | 15 | 16 | 20 | 19 | 18 | 17 |             |  |  |  |
| 9           | 10 | 11 | 12 |    |    |    |    |             |  |  |  |
| 5           | 6  | 7  | 8  |    |    |    |    |             |  |  |  |
| 1           | 2  | 3  | 4  |    |    |    |    |             |  |  |  |

# Example: sq-sr1.f (2/6)

Reading distributed local data files (sqm.\*)

```
!C
!C-- MESH
      if (my_rank.eq.0) filename= 'sqm.0'
      if (my_rank.eq.1) filename= 'sqm.1'
      if (my_rank.eq.2) filename= 'sqm.2'
      if (my_rank.eq.3) filename= 'sqm.3'
      open (21, file= filename, status= 'unknown')
      read (21,*) NEIBPETOT
      allocate (NEIBPE (NEIBPETOT))
      allocate (import_index(0:NEIBPETOT))
      allocate (export_index(0:NEIBPETOT))
      import_index= 0
      export_index= 0
      read (21,*) (NEIBPE(neib), neib= 1, NEIBPETOT)
      read (21,*) NP, N
      read (21,*) (import_index(neib), neib= 1, NEIBPETOT)
      nn= import_index(NEIBPETOT)
      allocate (import_item(nn))
      do i= 1, nn
        read (21,*) import_item(i)
      enddo

      read (21,*) (export_index(neib), neib= 1, NEIBPETOT)
      nn= export_index(NEIBPETOT)
      allocate (export_item(nn))
      do i= 1, nn
        read (21,*) export_item(i)
      enddo
      close (21)
```

```
#NEIBPETot
2
#NEIBPE
1 2
#NODE
24 16
#IMPORTIndex
4 8
#IMPORTItems
17
18
19
20
21
22
23
24
#EXPORTIndex
4 8
#EXPORTItems
4 8 12 16 13 14 15 16
```

# Example: sq-sr1.f (2/6)

Reading distributed local data files (sqm.\*)

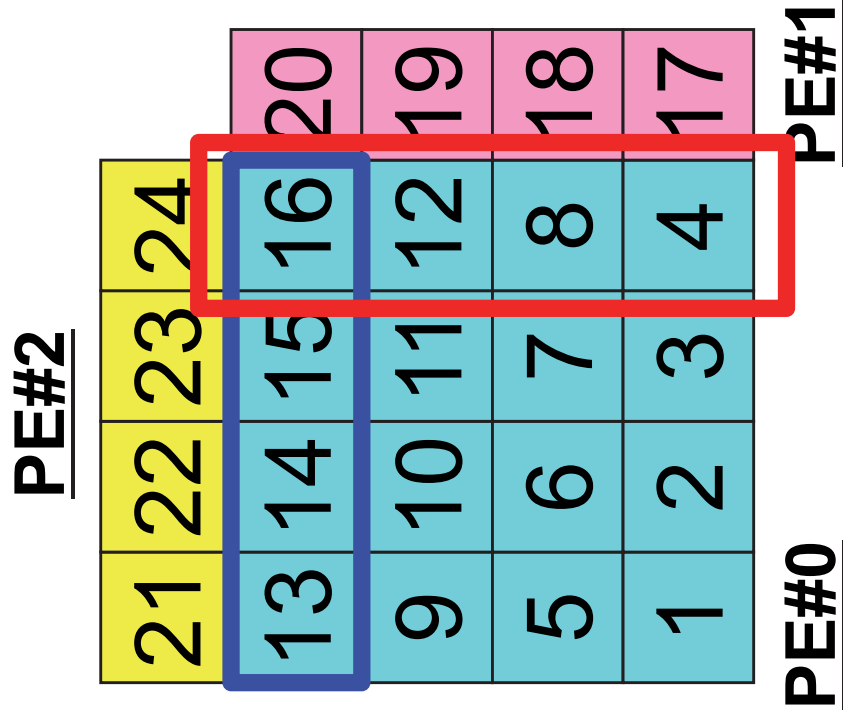
```
!C
!C-- MESH
      if (my_rank.eq.0) filename= 'sqm.0'
      if (my_rank.eq.1) filename= 'sqm.1'
      if (my_rank.eq.2) filename= 'sqm.2'
      if (my_rank.eq.3) filename= 'sqm.3'
      open (21, file= filename, status= 'unknown')
         read (21,*) NEIBPETOT
         allocate (NEIBPE (NEIBPETOT))
         allocate (import_index(0:NEIBPETOT))
         allocate (export_index(0:NEIBPETOT))
            import_index= 0
            export_index= 0
         read (21,*) (NEIBPE(neib), neib= 1, NEIBPETOT)
         read (21,*) NP, N
         read (21,*) (import_index(neib), neib= 1, NEIBPETOT)
            nn= import_index(NEIBPETOT)
            allocate (import_item(nn))
         do i= 1, nn
            read (21,*) import_item(i)
         enddo

         read (21,*) (export_index(neib), neib= 1, NEIBPETOT)
            nn= export_index(NEIBPETOT)
            allocate (export_item(nn))
         do i= 1, nn
            read (21,*) export_item(i)
         enddo
      close (21)
```

```
#NEIBPETot
2
#NEIBPE
1 2
#NODE
24 16
#IMPORTIndex
4 8
#IMPORTItems
17
18
19
20
21
22
23
24
#EXPORTIndex
4 8
#EXPORTItems
4 8 12 16
13 14 15 16
```

# SEND/Export: PE#0

```
#NEIBPEtot
2
#NEIBPE
1 2
#NODE
24 16
#IMPORTindex
4 8
#IMPORTitems
17
18
19
20
21
22
23
24
#EXPORTindex
4 8
#EXPORTitems
4 8 12 16
13 14 15 16
```



# Example: sq-sr1.f (3/6)

Fortran

Reading distributed local data files (sq.\*)

```
!C
!C-- VAL.
      if (my_rank.eq.0) filename= 'sq.0'
      if (my_rank.eq.1) filename= 'sq.1'
      if (my_rank.eq.2) filename= 'sq.2'
      if (my_rank.eq.3) filename= 'sq.3'

      allocate (VAL(NP))
      VAL= 0
      open (21, file= filename, status= 'unknown')
      do i= 1, N
        read (21,*) VAL(i)
      enddo
      close (21)

!C===
```

**N** : Number of internal points  
**VAL**: Global ID of meshes

**VAL** on external points are unknown  
 at this stage.

PE#2

|           |           |           |           |  |
|-----------|-----------|-----------|-----------|--|
|           |           |           |           |  |
| <u>25</u> | <u>26</u> | <u>27</u> | <u>28</u> |  |
| <u>17</u> | <u>18</u> | <u>19</u> | <u>20</u> |  |
| <u>9</u>  | <u>10</u> | <u>11</u> | <u>12</u> |  |
| <u>1</u>  | <u>2</u>  | <u>3</u>  | <u>4</u>  |  |

PE#0

PE#1

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



# Example: sq-sr1.f (4/6)

## Preparation of sending/receiving buffers

Fortran

```

!C
!C +-----+
!C | BUFFER |
!C +-----+
!C===
      allocate (SENDbuf (export_index (NEIBPETOT)))
      allocate (RECVbuf (import_index (NEIBPETOT)))

      SENDbuf= 0
      RECVbuf= 0

      do neib= 1, NEIBPETOT
        iS= export_index (neib-1) + 1
        iE= export_index (neib )
        do i= iS, iE
          SENDbuf(i)= VAL (export_item(i))
        enddo
      enddo
!C===

```

Info. of boundary points is written into sending buffer (SendBuf).  
 Info. sent to NEIBPE (neib) is stored in export\_index (neib-1)+1:export\_inedx (neib)

# Sending Buffer is nice ...

Fortran

```

do neib= 1, NEIBPETOT
  iS_e= export_index(neib-1) + 1
  iE_e= export_index(neib )
  BUFlenth_e= iE_e + 1 - iS_e

  call MPI_ISEND
    & (VAL(...), BUFlenth_e, MPI_INTEGER, NEIBPE(neib), 0, &
    & MPI_COMM_WORLD, request_send(neib), ierr)
enddo

```

PE#2

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

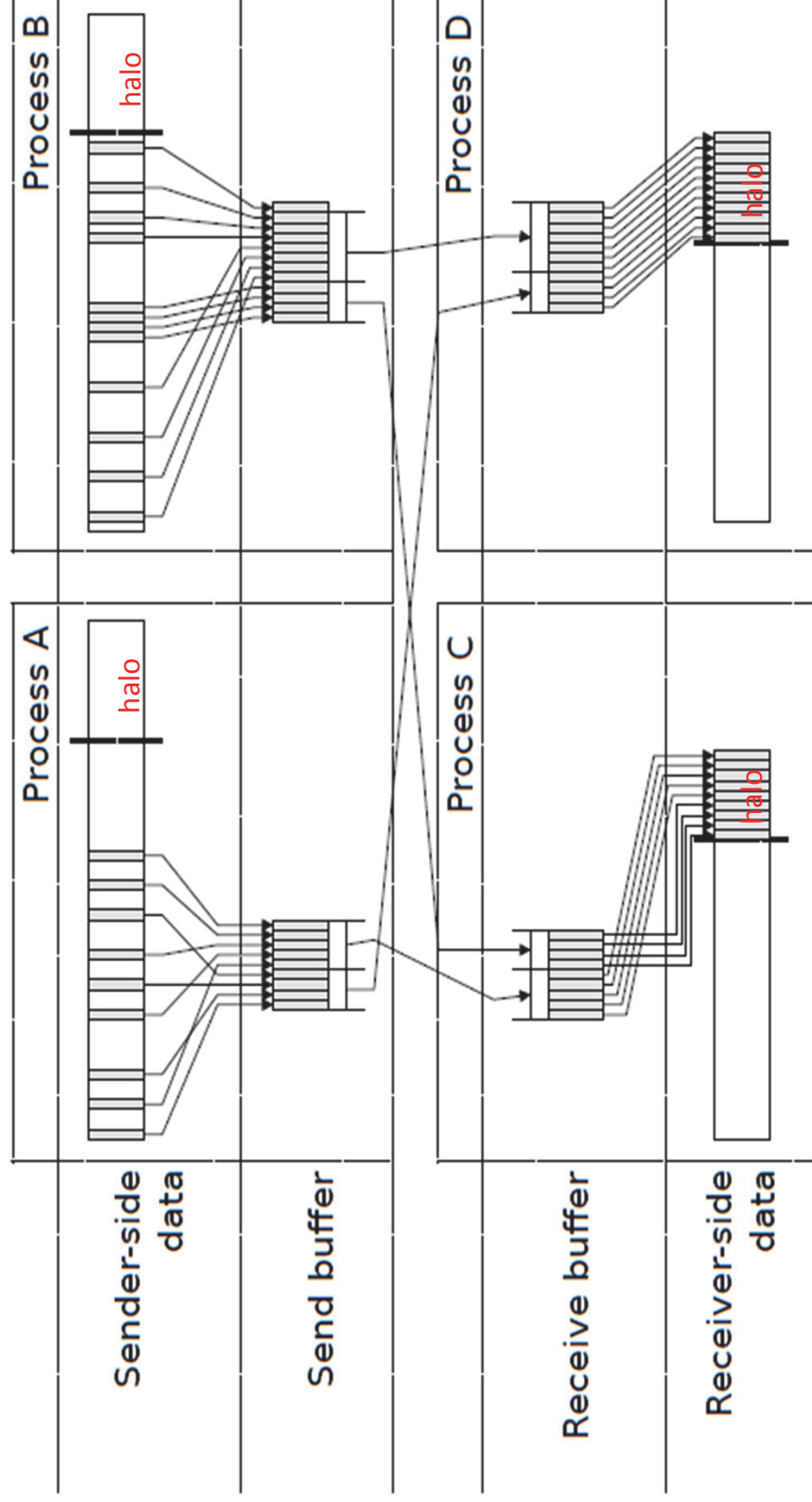
PE#0

PE#1

Numbering of these boundary nodes is not continuous, therefore the following procedure of MPI\_Isend is not applied directly:

- Starting address of sending buffer
- XX-messages from that address

# Communication Pattern using 1D Structure



Dr. Osni Marques  
(Lawrence Berkeley National  
Laboratory) より借用

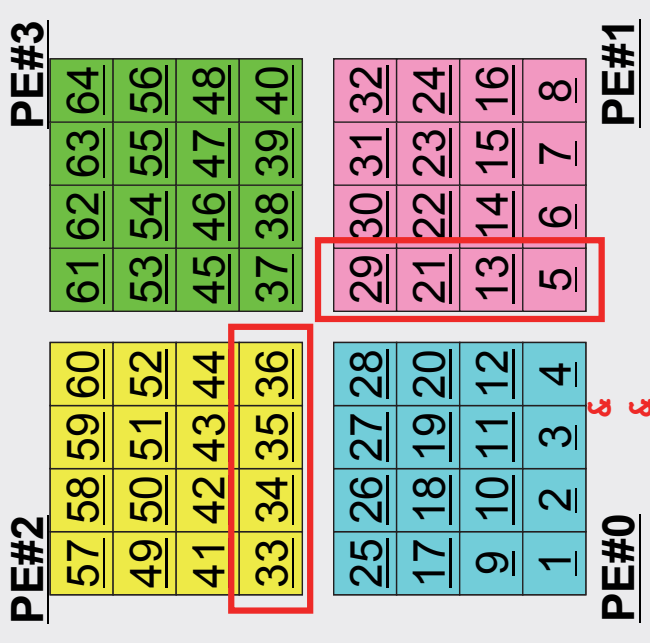
# Example: sq-sr1.f (5/6)

## SEND/Export: MPI\_Isend

```
!C
!C +-----+
!C | SEND-RECV |
!C +-----+
!C==
      allocate (stat_send(MPI_STATUS_SIZE,NEIBPETOT))
      allocate (stat_recv(MPI_STATUS_SIZE,NEIBPETOT))
      allocate (request_send(NEIBPETOT))
      allocate (request_recv(NEIBPETOT))

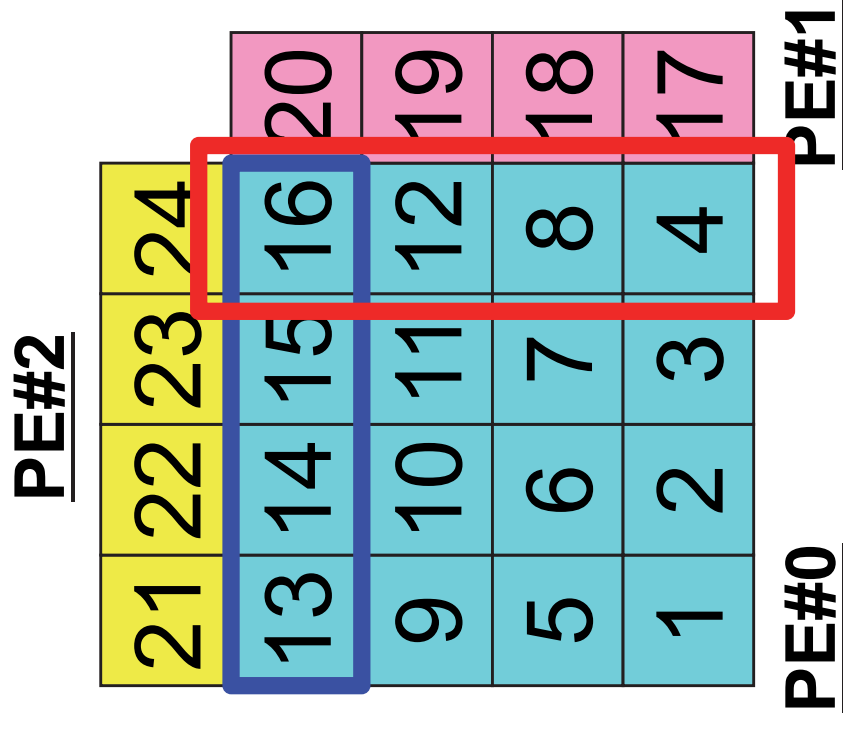
      do neib= 1, NEIBPETOT
        iS= export_index(neib-1) + 1
        iE= export_index(neib)
        BUFlength=iE + 1 - iS
        call MPI_ISEND (SENDbuf(iS), BUFlength, MPI_INTEGER,
&      NEIBPE(neib), 0, MPI_COMM_WORLD,
&      request_send(neib), ierr)
        enddo

      do neib= 1, NEIBPETOT
        iS= import_index(neib-1) + 1
        iE= import_index(neib)
        BUFlength=iE + 1 - iS
        call MPI_IRECV (RECVbuf(iS), BUFlength, MPI_INTEGER,
&      NEIBPE(neib), 0, MPI_COMM_WORLD,
&      request_recv(neib), ierr)
        enddo
```



# SEND/Export: PE#0

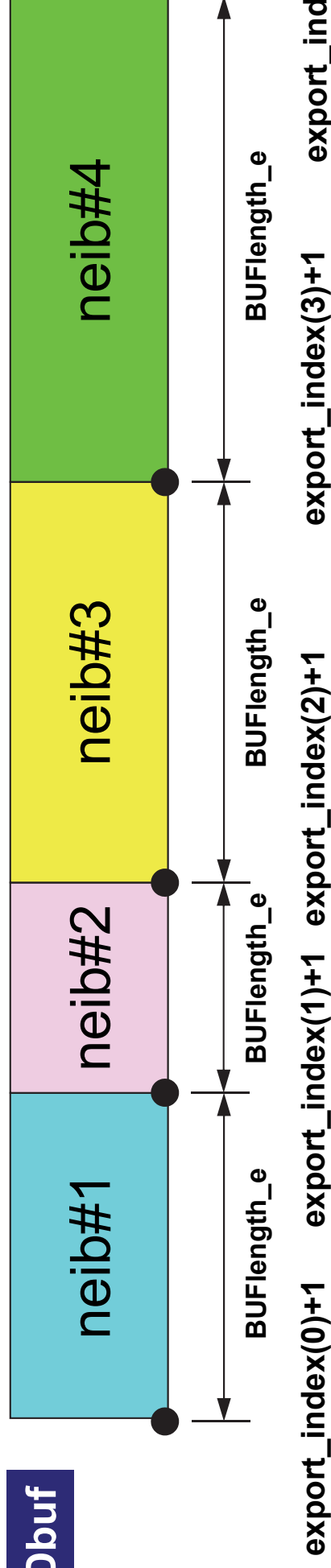
```
#NEIBPEtot
2
#NEIBPE
1 2
#NODE
24 16
#IMPORTindex
4 8
#IMPORTitems
17
18
19
20
21
22
23
24
#EXPORTindex
4 8
#EXPORTitems
4 8 12 16
13 14 15 16
```



# SEND: MPI\_Send/recv/Waitall

Fortran

SENDbuf



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

Copies to sending buffers

```
do neib= 1, NEIBPETOT
  iS_e= export_index(neib-1) + 1
  iE_e= export_index(neib )
  BUFLength_e= iE_e + 1 - iS_e

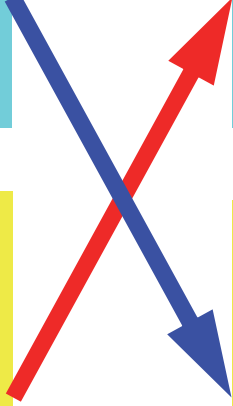
  call MPI_ISEND
    & (SENDbuf(iS_e) , BUFLength_e, MPI_INTEGER, NEIBPE(neib) , 0, &
    & MPI_COMM_WORLD, request_send(neib) , ierr)
enddo

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

# Notice: Send/Recv Arrays

```
#PE0
send:
    VEC(start_send) ~
    VEC(start_send+length_send-1)
```

```
#PE1
send:
    VEC(start_send) ~
    VEC(start_send+length_send-1)
```



```
#PE0
recv:
    VEC(start_recv) ~
    VEC(start_recv+length_recv-1)
```

```
#PE1
recv:
    VEC(start_recv) ~
    VEC(start_recv+length_recv-1)
```

- “length\_send” of sending process must be equal to “length\_recv” of receiving process.
  - PE#0 to PE#1, PE#1 to PE#0
- “sendbuf” and “recvbuf”: different address

# Relationship SEND/RECV

```

do neib= 1, NEIBPETOT
  iS_e= export_index(neib-1) + 1
  iE_e= export_index(neib )
  BUFlength_e= iE_e + 1 - iS_e

  call MPI_ISEND
    &      (SENDbuf(iS_e), BUFlength_e, MPI_INTEGER, NEIBPE(neib), 0,&
    &      MPI_COMM_WORLD, request_send(neib), ierr)
enddo

```

```

do neib= 1, NEIBPETOT
  iS_i= import_index(neib-1) + 1
  iE_i= import_index(neib )
  BUFlength_i= iE_i + 1 - iS_i

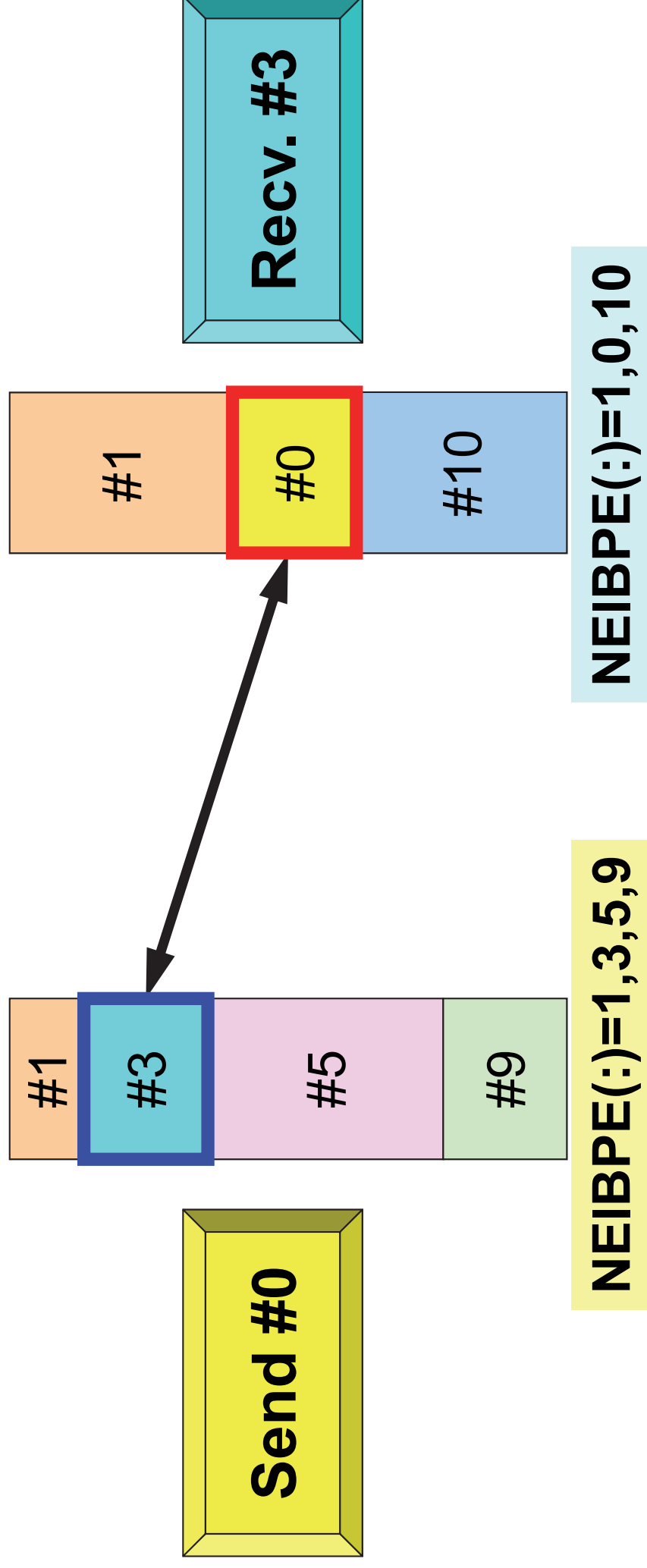
  call MPI_IRECV
    &      (RECVbuf(iS_i), BUFlength_i, MPI_INTEGER, NEIBPE(neib), 0,&
    &      MPI_COMM_WORLD, request_rcv(neib), ierr)
enddo

```

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



# Relationship SEND/RECV (#0 to #3)



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

# Example: sq-sr1.f (5/6)

Fortran

## RECV/Import: MPI\_Irecv

```

!C
!C +-----+
!C | SEND-RECV |
!C +-----+
!C==
      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
        iS= export_index(neib-1) + 1
        iE= export_index(neib)
        BUFlength=iE + 1 - iS
        call MPI_ISEND (SENDbuf(iS), BUFlength, MPI_INTEGER,
          & NEIBPE(neib), 0, MPI_COMM_WORLD,
          & request_send(neib), ierr)
      enddo

      do neib= 1, NEIBPETOT
        iS= import_index(neib-1) + 1
        iE= import_index(neib)
        BUFlength= iE + 1 - iS
        call MPI_IRECV (RECVbuf(iS), BUFlength, MPI_INTEGER,
          & NEIBPE(neib), 0, MPI_COMM_WORLD,
          & request_recv(neib), ierr)
      enddo

```



# RECV/Import: PE#0

```
#NEIBPEtot
2
#NEIBPE
1 2
#NODE
24 16
#IMPORTindex
4 8
#IMPORTitems
17
18
19
20
21
22
23
24
#EXPORTindex
4 8
#EXPORTitems
4 8
12
16
13
14
15
16
```

| <u>PE#2</u> |    |    |    |    |    |    |    | <u>PE#1</u> |    |    |    |
|-------------|----|----|----|----|----|----|----|-------------|----|----|----|
| 21          | 22 | 23 | 24 | 13 | 14 | 15 | 16 | 20          | 19 | 18 | 17 |
| 9           | 10 | 11 | 12 | 5  | 6  | 7  | 8  |             |    |    |    |
| 1           | 2  | 3  | 4  |    |    |    |    |             |    |    |    |
| <u>PE#0</u> |    |    |    |    |    |    |    |             |    |    |    |

# RECV: MPI\_Isend/Irecv/Waitall Fortran

```

do neib= 1, NEIBPETOT
  iS_i= import_index(neib-1) + 1
  iE_i= import_index(neib )
  BUFLength_i=iE_i + 1 - iS_i

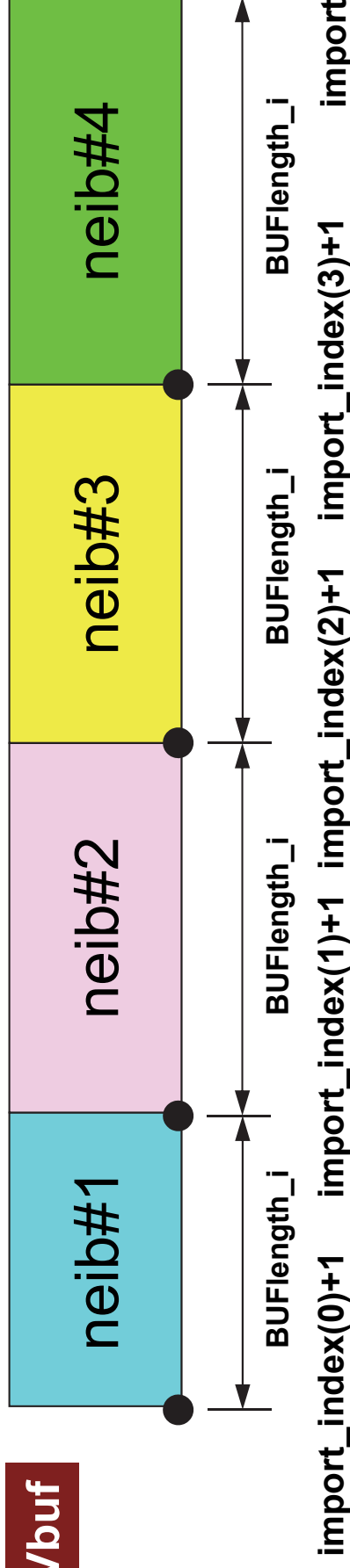
  call MPI_Irecv
    & (RECVbuf(iS_i) , BUFLength_i, MPI_INTEGER, NEIBPE(neib) , 0, &
    & MPI_COMM_WORLD, request_recv(neib) , ierr)
  enddo

  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

```

Copies from receiving buffers



# Example: sq-sr1.f (6/6)

Fortran

Reading info of ext pts from receiving buffers

```
call MPI_WAITALL (NEIBPETOT, request_recv, stat_recv, ierr)
```

```
do neib= 1, NEIBPETOT
  iS= import_index(neib-1) + 1
  iE= import_index(neib )
  do i= iS, iE
    VAL(import_item(i))= RECVbuf(i)
  enddo
enddo
```

Contents of RecvBuf are copied to values at external points.

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

```
!C===
!C +-----+
!C | OUTPUT |
!C +-----+
!C===
do neib= 1, NEIBPETOT
  iS= import_index(neib-1) + 1
  iE= import_index(neib )
  do i= iS, iE
    in= import_item(i)
    write (*,'(a, 3i8)') 'RECVbuf', my_rank, NEIBPE(neib), VAL(in)
  enddo
enddo
!C===
call MPI_FINALIZE (ierr)
stop
end
```

# Example: sq-sr1.f (6/6)

## Writing values at external points

Fortran

```

call MPI_WAITALL (NEIBPETOT, request_rcv, stat_rcv, ierr)

do neib= 1, NEIBPETOT
  iS= import_index(neib-1) + 1
  iE= import_index(neib )
  do i= iS, iE
    VAL(import_item(i))= RECVbuf(i)
  enddo
enddo

call MPI_WAITALL (NEIBPETOT, request_send, stat_send, ierr)

!C===

!C +-----+
!C | OUTPUT |
!C +-----+
!C===

do neib= 1, NEIBPETOT
  iS= import_index(neib-1) + 1
  iE= import_index(neib )
  do i= iS, iE
    in= import_item(i)
    write (*,'(a, 3i8)') 'RECVbuf', my_rank, NEIBPE(neib), VAL(in)
  enddo
enddo

!C===

call MPI_FINALIZE (ierr)
stop
end

```

# Results (PE#0)

## PE#2

|    |    |    |    |
|----|----|----|----|
| 57 | 58 | 59 | 60 |
| 49 | 50 | 51 | 52 |
| 41 | 42 | 43 | 44 |
| 33 | 34 | 35 | 36 |

## PE#3

|    |    |    |    |
|----|----|----|----|
| 61 | 62 | 63 | 64 |
| 53 | 54 | 55 | 56 |
| 45 | 46 | 47 | 48 |
| 37 | 38 | 39 | 40 |

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

## PE#0

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

## PE#1

RECVbuf  
RECVbuf  
RECVbuf  
RECVbuf  
RECVbuf  
RECVbuf  
RECVbuf  
RECVbuf

0  
0  
0  
0  
0  
0  
0  
0

1  
1  
1  
1  
2  
2  
2  
2

5  
13  
21  
29  
33  
34  
35  
36

RECVbuf  
RECVbuf  
RECVbuf  
RECVbuf  
RECVbuf  
RECVbuf  
RECVbuf  
RECVbuf

1  
1  
1  
1  
1  
1  
1  
1

0  
0  
0  
0  
3  
3  
3  
3

4  
12  
20  
28  
37  
38  
39  
40

RECVbuf  
RECVbuf  
RECVbuf  
RECVbuf  
RECVbuf  
RECVbuf  
RECVbuf  
RECVbuf

2  
2  
2  
2  
2  
2  
2  
2

3  
3  
3  
3  
0  
0  
0  
0

37  
45  
53  
61  
25  
26  
27  
28

RECVbuf  
RECVbuf  
RECVbuf  
RECVbuf  
RECVbuf  
RECVbuf  
RECVbuf  
RECVbuf

3  
3  
3  
3  
3  
3  
3  
3

2  
2  
2  
2  
1  
1  
1  
1

36  
44  
52  
60  
29  
30  
31  
32

# Results (PE#1)

## PE#2

|    |    |    |    |
|----|----|----|----|
| 57 | 58 | 59 | 60 |
| 49 | 50 | 51 | 52 |
| 41 | 42 | 43 | 44 |
| 33 | 34 | 35 | 36 |

## PE#3

|    |    |    |    |
|----|----|----|----|
| 61 | 62 | 63 | 64 |
| 53 | 54 | 55 | 56 |
| 45 | 46 | 47 | 48 |
| 37 | 38 | 39 | 40 |

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

## PE#0

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

## PE#1

|         |   |   |    |
|---------|---|---|----|
| RECVbuf | 0 | 1 | 5  |
| RECVbuf | 0 | 1 | 13 |
| RECVbuf | 0 | 1 | 21 |
| RECVbuf | 0 | 1 | 29 |
| RECVbuf | 0 | 2 | 33 |
| RECVbuf | 0 | 2 | 34 |
| RECVbuf | 0 | 2 | 35 |
| RECVbuf | 0 | 2 | 36 |
| RECVbuf | 1 | 0 | 4  |
| RECVbuf | 1 | 0 | 12 |
| RECVbuf | 1 | 0 | 20 |
| RECVbuf | 1 | 0 | 28 |
| RECVbuf | 1 | 3 | 37 |
| RECVbuf | 1 | 3 | 38 |
| RECVbuf | 1 | 3 | 39 |
| RECVbuf | 1 | 3 | 40 |
| RECVbuf | 2 | 3 | 37 |
| RECVbuf | 2 | 3 | 45 |
| RECVbuf | 2 | 3 | 53 |
| RECVbuf | 2 | 3 | 61 |
| RECVbuf | 2 | 0 | 25 |
| RECVbuf | 2 | 0 | 26 |
| RECVbuf | 2 | 0 | 27 |
| RECVbuf | 2 | 0 | 28 |
| RECVbuf | 3 | 2 | 36 |
| RECVbuf | 3 | 2 | 44 |
| RECVbuf | 3 | 2 | 52 |
| RECVbuf | 3 | 2 | 60 |
| RECVbuf | 3 | 1 | 29 |
| RECVbuf | 3 | 1 | 30 |
| RECVbuf | 3 | 1 | 31 |
| RECVbuf | 3 | 1 | 32 |



# Results (PE#2)

## PE#2

|    |    |    |    |
|----|----|----|----|
| 57 | 58 | 59 | 60 |
| 49 | 50 | 51 | 52 |
| 41 | 42 | 43 | 44 |
| 33 | 34 | 35 | 36 |

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

## PE#0

## PE#3

|    |    |    |    |
|----|----|----|----|
| 61 | 62 | 63 | 64 |
| 53 | 54 | 55 | 56 |
| 45 | 46 | 47 | 48 |
| 37 | 38 | 39 | 40 |

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

## PE#1

|         |   |   |    |
|---------|---|---|----|
| RECVbuf | 0 | 1 | 5  |
| RECVbuf | 0 | 1 | 13 |
| RECVbuf | 0 | 1 | 21 |
| RECVbuf | 0 | 1 | 29 |
| RECVbuf | 0 | 2 | 33 |
| RECVbuf | 0 | 2 | 34 |
| RECVbuf | 0 | 2 | 35 |
| RECVbuf | 0 | 2 | 36 |
| RECVbuf | 1 | 0 | 4  |
| RECVbuf | 1 | 0 | 12 |
| RECVbuf | 1 | 0 | 20 |
| RECVbuf | 1 | 0 | 28 |
| RECVbuf | 1 | 3 | 37 |
| RECVbuf | 1 | 3 | 38 |
| RECVbuf | 1 | 3 | 39 |
| RECVbuf | 1 | 3 | 40 |
| RECVbuf | 2 | 3 | 37 |
| RECVbuf | 2 | 3 | 45 |
| RECVbuf | 2 | 3 | 53 |
| RECVbuf | 2 | 3 | 61 |
| RECVbuf | 2 | 0 | 25 |
| RECVbuf | 2 | 0 | 26 |
| RECVbuf | 2 | 0 | 27 |
| RECVbuf | 2 | 0 | 28 |
| RECVbuf | 3 | 2 | 36 |
| RECVbuf | 3 | 2 | 44 |
| RECVbuf | 3 | 2 | 52 |
| RECVbuf | 3 | 2 | 60 |
| RECVbuf | 3 | 1 | 29 |
| RECVbuf | 3 | 1 | 30 |
| RECVbuf | 3 | 1 | 31 |
| RECVbuf | 3 | 1 | 32 |

# Results (PE#3)

## PE#2

|    |    |    |    |
|----|----|----|----|
| 57 | 58 | 59 | 60 |
| 49 | 50 | 51 | 52 |
| 41 | 42 | 43 | 44 |
| 33 | 34 | 35 | 36 |

## PE#3

|    |    |    |    |
|----|----|----|----|
| 61 | 62 | 63 | 64 |
| 53 | 54 | 55 | 56 |
| 45 | 46 | 47 | 48 |
| 37 | 38 | 39 | 40 |

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

## PE#0

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

## PE#1

|         |   |   |    |
|---------|---|---|----|
| RECVbuf | 0 | 1 | 5  |
| RECVbuf | 0 | 1 | 13 |
| RECVbuf | 0 | 1 | 21 |
| RECVbuf | 0 | 1 | 29 |
| RECVbuf | 0 | 2 | 33 |
| RECVbuf | 0 | 2 | 34 |
| RECVbuf | 0 | 2 | 35 |
| RECVbuf | 0 | 2 | 36 |
| RECVbuf | 1 | 0 | 4  |
| RECVbuf | 1 | 0 | 12 |
| RECVbuf | 1 | 0 | 20 |
| RECVbuf | 1 | 0 | 28 |
| RECVbuf | 1 | 3 | 37 |
| RECVbuf | 1 | 3 | 38 |
| RECVbuf | 1 | 3 | 39 |
| RECVbuf | 1 | 3 | 40 |
| RECVbuf | 2 | 3 | 37 |
| RECVbuf | 2 | 3 | 45 |
| RECVbuf | 2 | 3 | 53 |
| RECVbuf | 2 | 3 | 61 |
| RECVbuf | 2 | 0 | 25 |
| RECVbuf | 2 | 0 | 26 |
| RECVbuf | 2 | 0 | 27 |
| RECVbuf | 2 | 0 | 28 |
| RECVbuf | 3 | 2 | 36 |
| RECVbuf | 3 | 2 | 44 |
| RECVbuf | 3 | 2 | 52 |
| RECVbuf | 3 | 2 | 60 |
| RECVbuf | 3 | 1 | 29 |
| RECVbuf | 3 | 1 | 30 |
| RECVbuf | 3 | 1 | 31 |
| RECVbuf | 3 | 1 | 32 |

# Distributed Local Data Structure for Parallel Computation

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

# Initial Mesh

t2

|           |           |           |           |           |
|-----------|-----------|-----------|-----------|-----------|
| <u>21</u> | <u>22</u> | <u>23</u> | <u>24</u> | <u>25</u> |
| <u>16</u> | <u>17</u> | <u>18</u> | <u>19</u> | <u>20</u> |
| <u>11</u> | <u>12</u> | <u>13</u> | <u>14</u> | <u>15</u> |
| <u>6</u>  | <u>7</u>  | <u>8</u>  | <u>9</u>  | <u>10</u> |
| <u>1</u>  | <u>2</u>  | <u>3</u>  | <u>4</u>  | <u>5</u>  |

# Three Domains

#PE2

|           |           |           |           |
|-----------|-----------|-----------|-----------|
| <u>21</u> | <u>22</u> | <u>23</u> | <u>24</u> |
| <u>16</u> | <u>17</u> | <u>18</u> | <u>19</u> |
| <u>11</u> | <u>12</u> | <u>13</u> | <u>14</u> |
| <u>6</u>  | <u>7</u>  | <u>8</u>  |           |

#PE1

|           |           |           |
|-----------|-----------|-----------|
| <u>23</u> | <u>24</u> | <u>25</u> |
| <u>18</u> | <u>19</u> | <u>20</u> |
| <u>13</u> | <u>14</u> | <u>15</u> |
| <u>8</u>  | <u>9</u>  | <u>10</u> |
|           | <u>4</u>  | <u>5</u>  |

#PE0

|           |           |           |
|-----------|-----------|-----------|
| <u>11</u> | <u>12</u> | <u>13</u> |
| <u>6</u>  | <u>7</u>  | <u>8</u>  |
| <u>9</u>  | <u>10</u> |           |
| <u>1</u>  | <u>2</u>  | <u>3</u>  |
|           | <u>4</u>  | <u>5</u>  |

t2

# Three Domains

t2

#PE2

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

#PE1

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

#PE0

|          |          |          |
|----------|----------|----------|
| 11<br>11 | 12<br>12 | 13<br>13 |
| 6<br>6   | 7<br>7   | 8<br>8   |
| 1<br>1   | 2<br>2   | 3<br>3   |
|          | 9<br>9   | 10<br>10 |
|          | 4<br>4   | 5<br>5   |

t2

# PE#0: sqm.0: fill O's

#PE2

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

#PE1

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

#PE0

|          |          |          |
|----------|----------|----------|
| 11<br>11 | 12<br>12 | 13<br>13 |
| 6<br>6   | 7<br>7   | 8<br>8   |
| 1<br>1   | 2<br>2   | 3<br>3   |
|          | 9<br>9   | 10<br>10 |
|          | 4<br>4   | 5<br>5   |

#NEIBPEtot  
2

#NEIBPE  
12

#NODE  
13

#IMPORTindex  
00

#IMPORTitems  
0...

#EXPORTindex  
00

#EXPORTitems  
0...

2

8

(int+ext, int pts)

# PE#1: sqm.1: fill O's

#PE2

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

#PE1

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

#PE0

|          |          |          |
|----------|----------|----------|
| 11<br>11 | 12<br>12 | 13<br>13 |
| 6<br>6   | 7<br>7   | 8<br>8   |
| 1<br>1   | 2<br>2   | 3<br>3   |
|          | 9<br>9   | 10<br>10 |
|          | 4<br>4   | 5<br>5   |

```
#NEIBPEtot      2
#NEIBPE          0      2
#NODE            8      14 (int+ext, int pts)
#IMPORTindex     0      0
#IMPORTitems    0...
#EXPORTindex     0      0
#EXPORTitems    0...
```



# PE#2: sqm.2: fill O's

t2

#PE2

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

#PE1

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

#PE0

|          |          |          |
|----------|----------|----------|
| 11<br>11 | 12<br>12 | 13<br>13 |
| 6<br>6   | 7<br>7   | 8<br>8   |
| 1<br>1   | 2<br>2   | 3<br>3   |
|          | 9<br>9   | 10<br>10 |
|          | 4<br>4   | 5<br>5   |

```

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

```

t2

#PE2

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

#PE1

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

#PE0

|          |          |          |
|----------|----------|----------|
| 11<br>11 | 12<br>12 | 13<br>13 |
| 6<br>6   | 7<br>7   | 8<br>8   |
| 1<br>1   | 2<br>2   | 3<br>3   |
|          | 9<br>9   | 10<br>10 |
|          | 4<br>4   | 5<br>5   |

# Procedures

t2

- Number of Internal/External Points
- Where do External Pts come from ?
  - **IMPORTIndex**, **IMPORTItems**
  - Sequence of **NEIBPE**
- Then check destinations of Boundary Pts.
  - **EXPORTIndex**, **EXPORTItems**
  - Sequence of **NEIBPE**
- “sq.\*” are in <O-S2>/ex
- Create “sqm.\*” by yourself
- copy <O-S2>/a.out (by sq-sr1.f) to <O-S2>/ex
- pjsub go3.sh

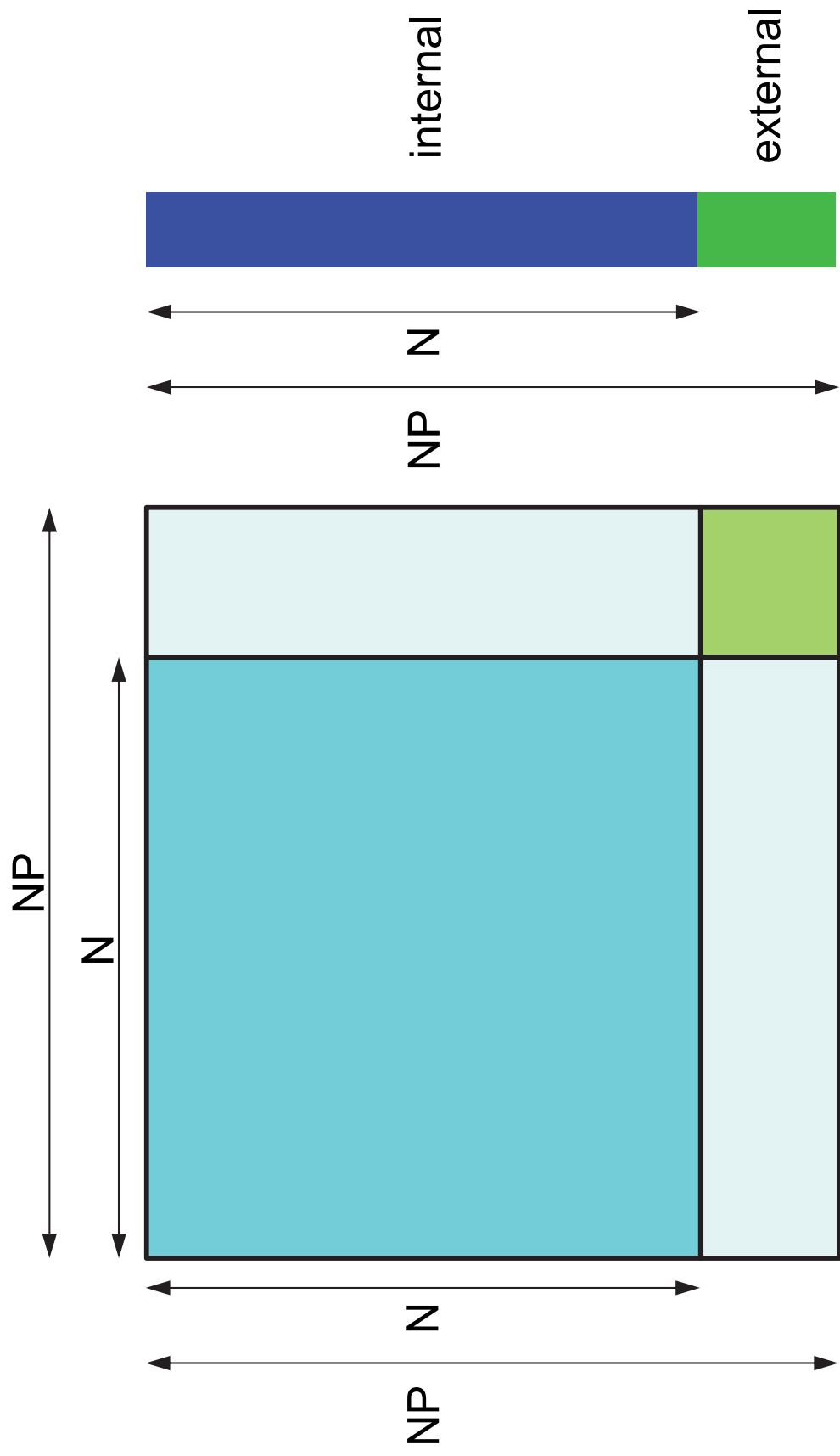
# Report S2 (1/2)

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

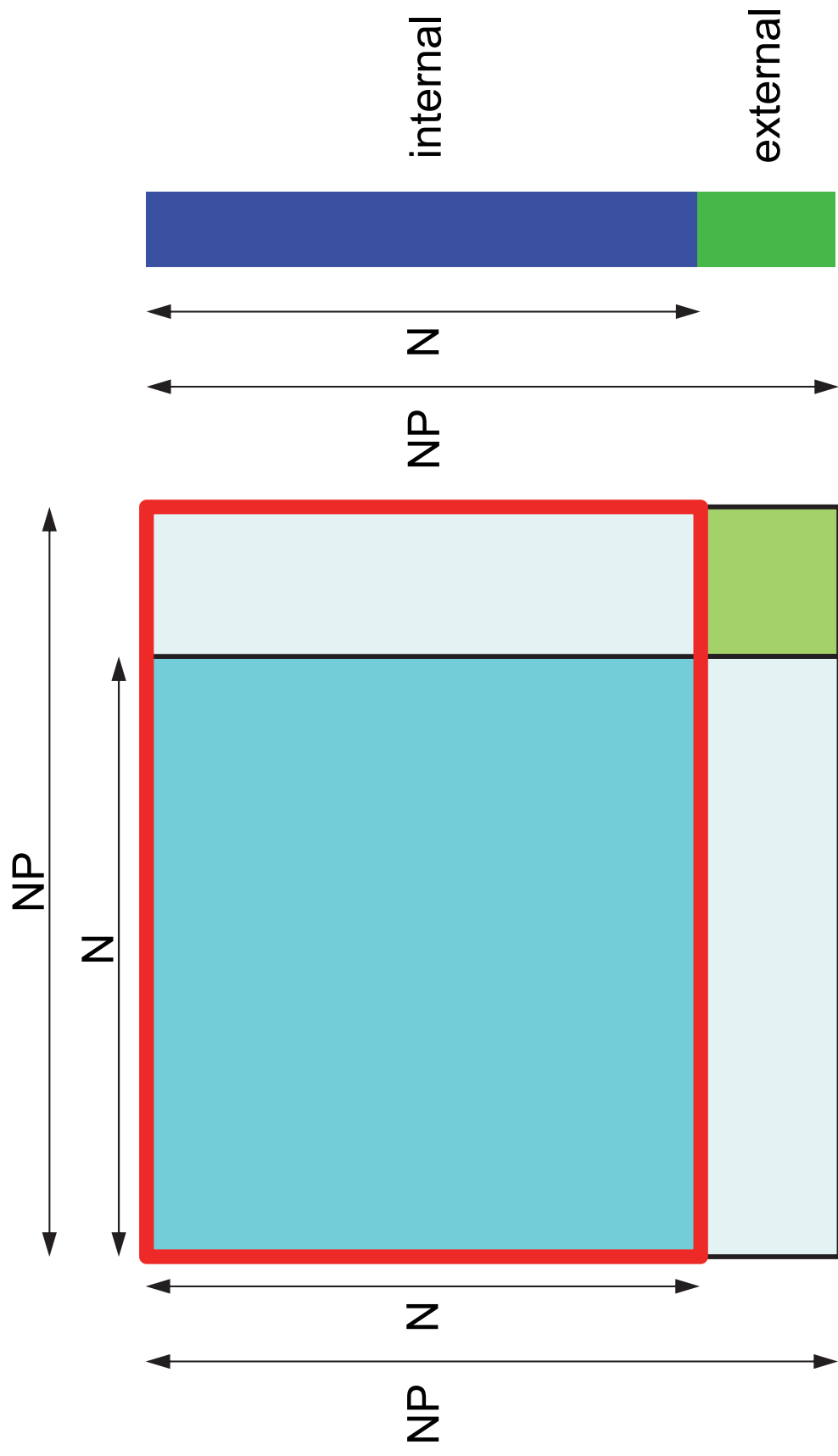
# Report S2 (2/2)

- **Deadline:** 17:00 February 14<sup>th</sup> (Sat), 2015.
  - Send files via e-mail at `nakajima(at)cc.u-tokyo.ac.jp`
- **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)

# Local Matrix



# We really need these parts:



```

!C +-----+
!C | MATRIX ASSEMBLE |
!C +-----+
!C===

do icel= 1, NE
  in1= ICELNOD(2*icel-1)
  in2= ICELNOD(2*icel )
  DL = dX

  cK= AREA*Young/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)
enddo

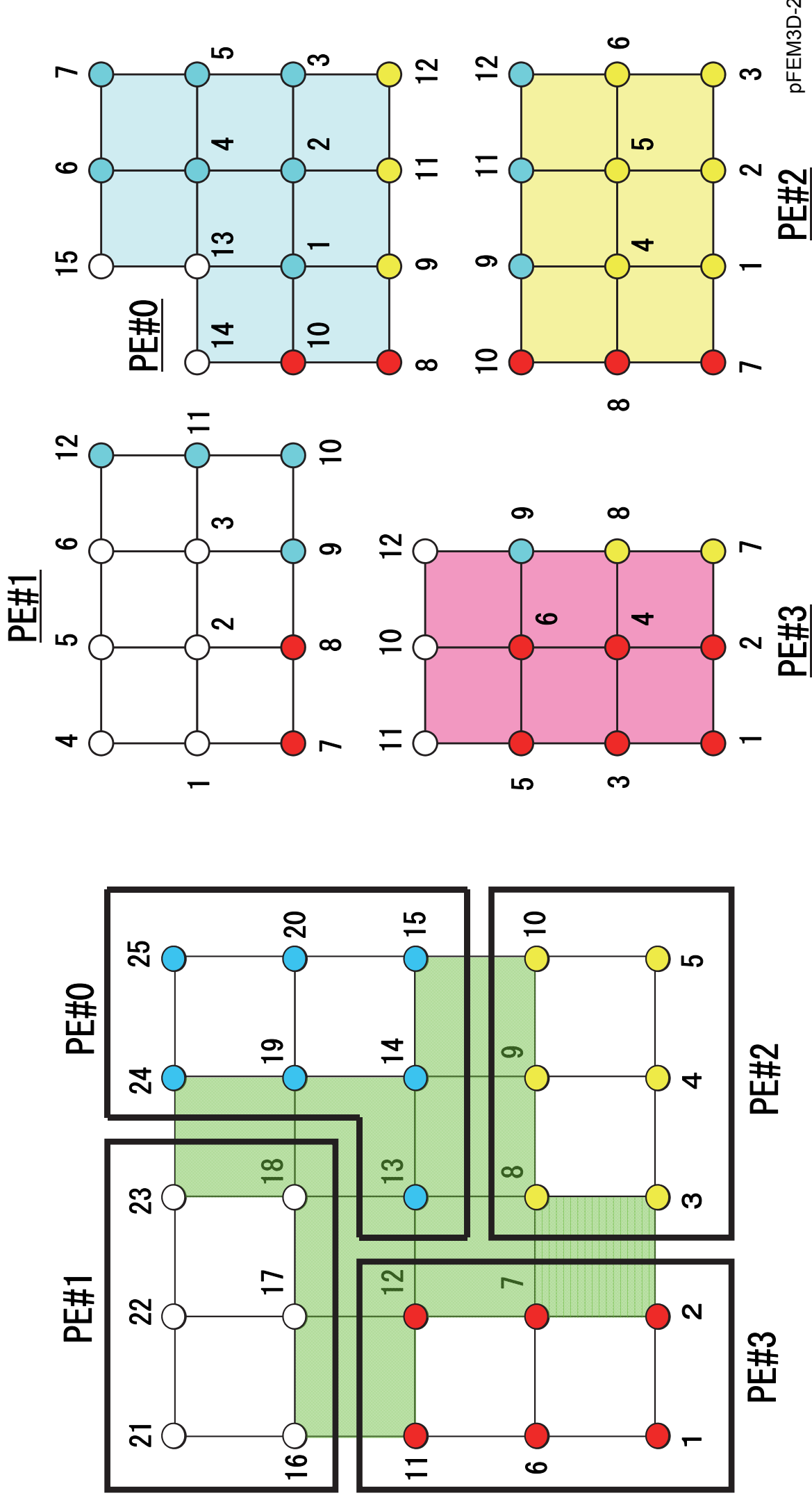
!C===

```

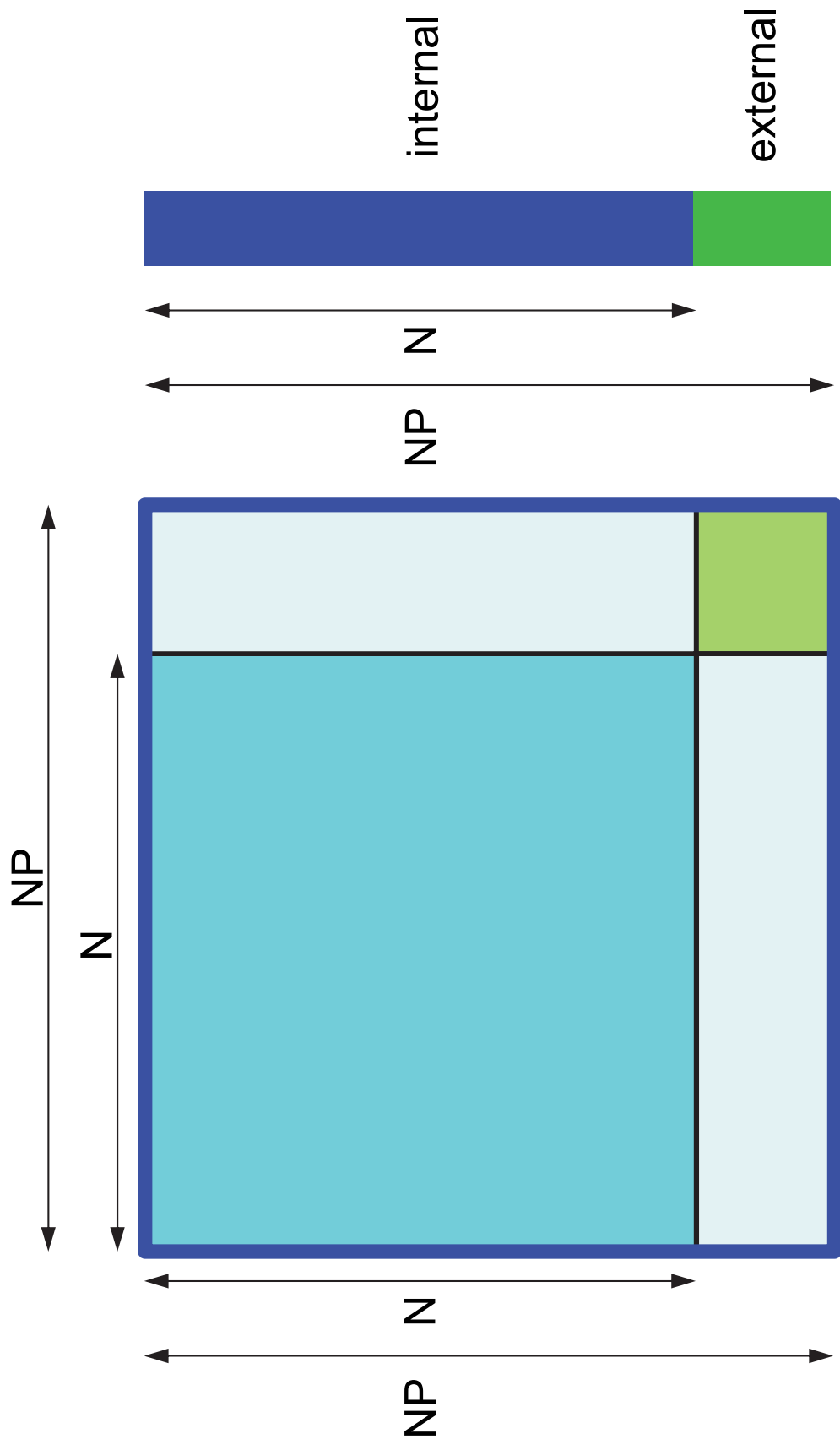


# MAT\_ASS\_MAIN visits all elements

including overlapped elements with external nodes



# Therefore, we have this matrix



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

