

並列有限要素法による  
三次元定常熱伝導解析プログラム  
(1/2) Fortran編

中島 研吾

東京大学情報基盤センター

# 扱うプログラム

- fem3dの並列版
- MPIによる並列化

- プログラムのインストール
- 実行
  - 並列有限要素法の手順
  - 領域分割とは?
  - 本当の実行
- データ構造

# Preparation (Odyssey)

## FORTRAN

```
>$ cd /work/gt00/t00XYZ/pFEM
>$ cp /work/gt00/z30088/pFEM/F/fem3d.tar .
>$ tar xvf fem3d.tar
```

## C

```
>$ cd /work/gt00/t00XYZ/pFEM
>$ cp /work/gt00/z30088/pFEM/C/fem3d.tar .
>$ tar xvf fem3d.tar
```

## Confirmation

```
>$ ls
    mpi    fem3d    pfem3d
>$ cd pfem3d
```

# Compilation (Odyssey)

## Mesh Generator

```
>$ cd /work/gt00/t00XXX/pFEM/pfem3d/mesh  
>$ module load fj  
>$ frtpx -Kfast mgcube.f -o mgcube
```

## Domain Partitioner

```
>$ cd ../part  
>$ module load metis/4.0.3  
>$ make  
>$ ls ../mesh/part  
part
```

## Parallel FEM

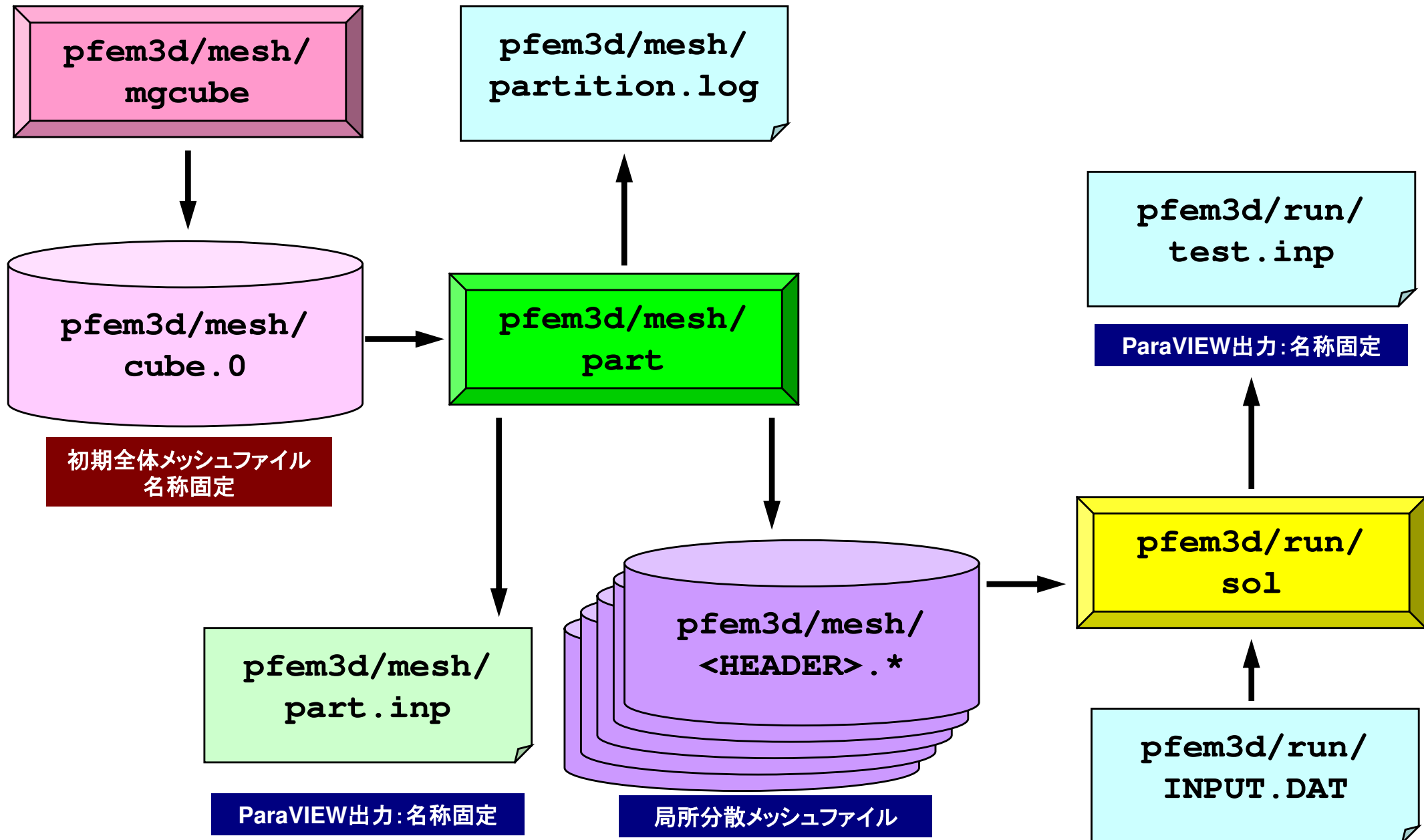
```
>$ cd ../src  
>$ make  
>$ ls ../run/sol  
sol
```

- プログラムのインストール
- 実行
  - 並列有限要素法の手順
  - 領域分割とは？
  - 本当の実行
- データ構造

# 並列有限要素法の手順

- 初期全体メッシュファイルを作成する
  - `/work/gt00/t00XYZ/pFEM/pfem3d/mesh/mg.sh`
- 領域を分割する(局所分散メッシュファイル)
  - `/work/gt00/t00XYZ/pFEM/pfem3d/mesh/part_XXX.sh`
- 計算を実施する
  - `/work/gt00/t00XYZ/pFEM/pfem3d/run/go.sh`

# 並列有限要素法の手順





- プログラムのインストール
- 実行
  - 並列有限要素法の手順
  - 領域分割とは?
  - 本当の実行
- データ構造

# 領域分割機能：Partitioner

初期全体メッシュデータを与えることによって、  
自動的に局所分散メッシュデータを生成する  
一次元⇒プログラム内で実行, 三次元⇒困難

- 内点, 外点
  - 局所分散メッシュデータ
  - 内点～外点となるように局所番号をつける
- 通信テーブル
  - 隣接領域情報
    - 隣接領域数
    - 隣接領域番号
  - 外点情報
    - どの領域から, 何個の, どの外点の情報を「import」するか
  - 境界点情報
    - 何個の, どの境界点の情報を, どの領域に「export」するか

# Partitioning とは？

- Graph/Graphic Partitioningの略
- 並列計算のための領域分割を実現するための手法
- 1PEでは計算できないような巨大な全体領域を局所データに分割する

# Graph/Graphic Partitioning とは？

Graph/Graphic Partitioningとは「グラフ」（*graphs* : 節点と辺の集合）に関する「グラフ理論」を並列計算における領域分割に応用した手法である

一筆書き，四色問題

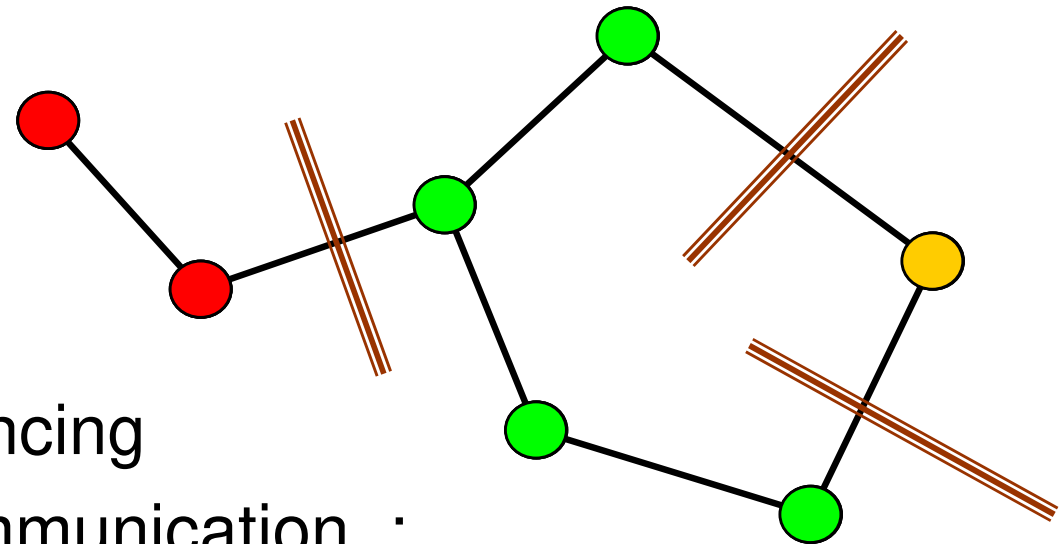
## 良い領域分割

領域間の負荷均等 : Load balancing

領域間通信量最小 : Small Communication :

前処理つき反復法の収束に影響

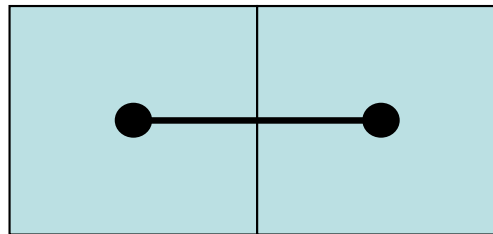
隣接領域数最小



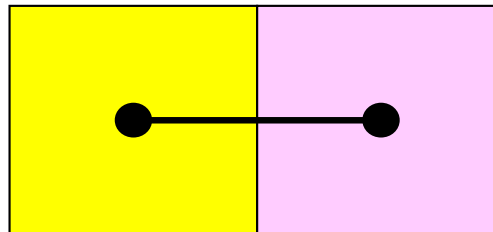
# EDGE-CUTとは？

- 辺の両端の節点（または要素）が異なった領域に属している場合、「EDGE-CUTが生じている。」という。
- EDGE-CUTが少ないほど、通信は少ない。

EDGE-CUT無し



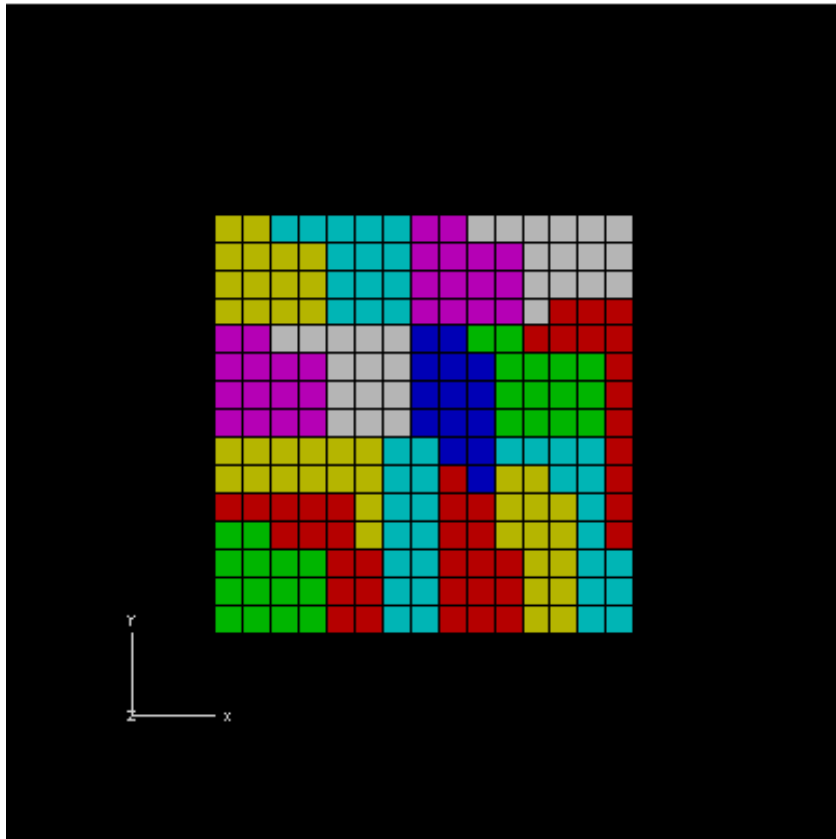
EDGE-CUT有り



# Partitioning の反復法収束への影響

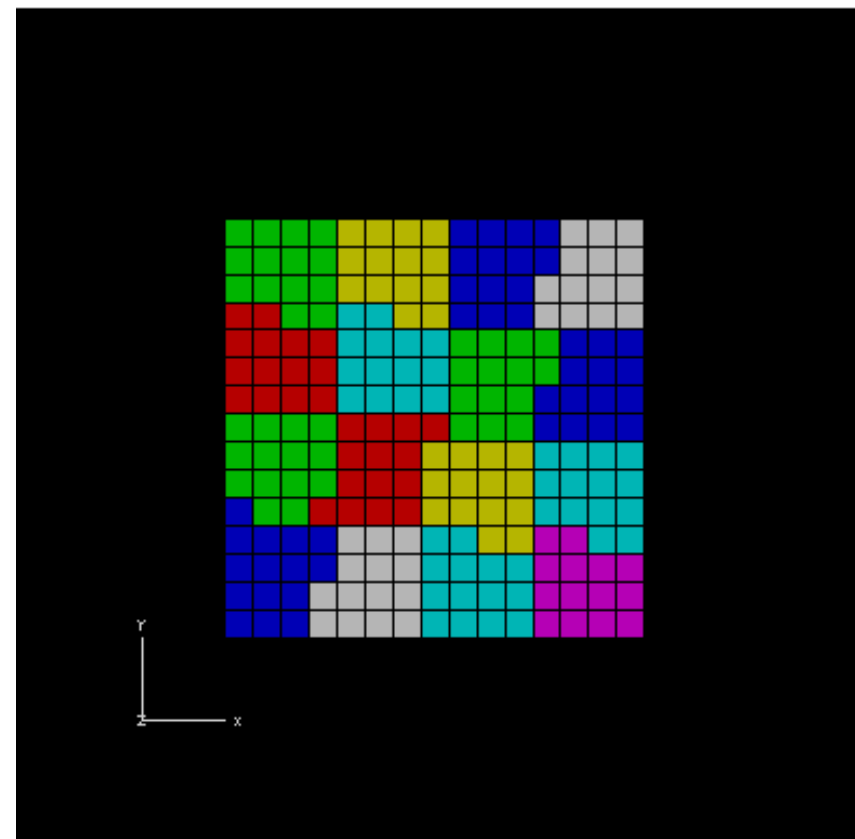
15 × 15領域を16分割：負荷バランスは取れている

Edge-Cut多い



RGB

Edge-Cut少ない



RSB

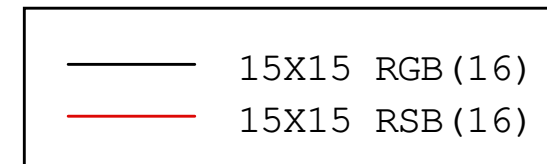
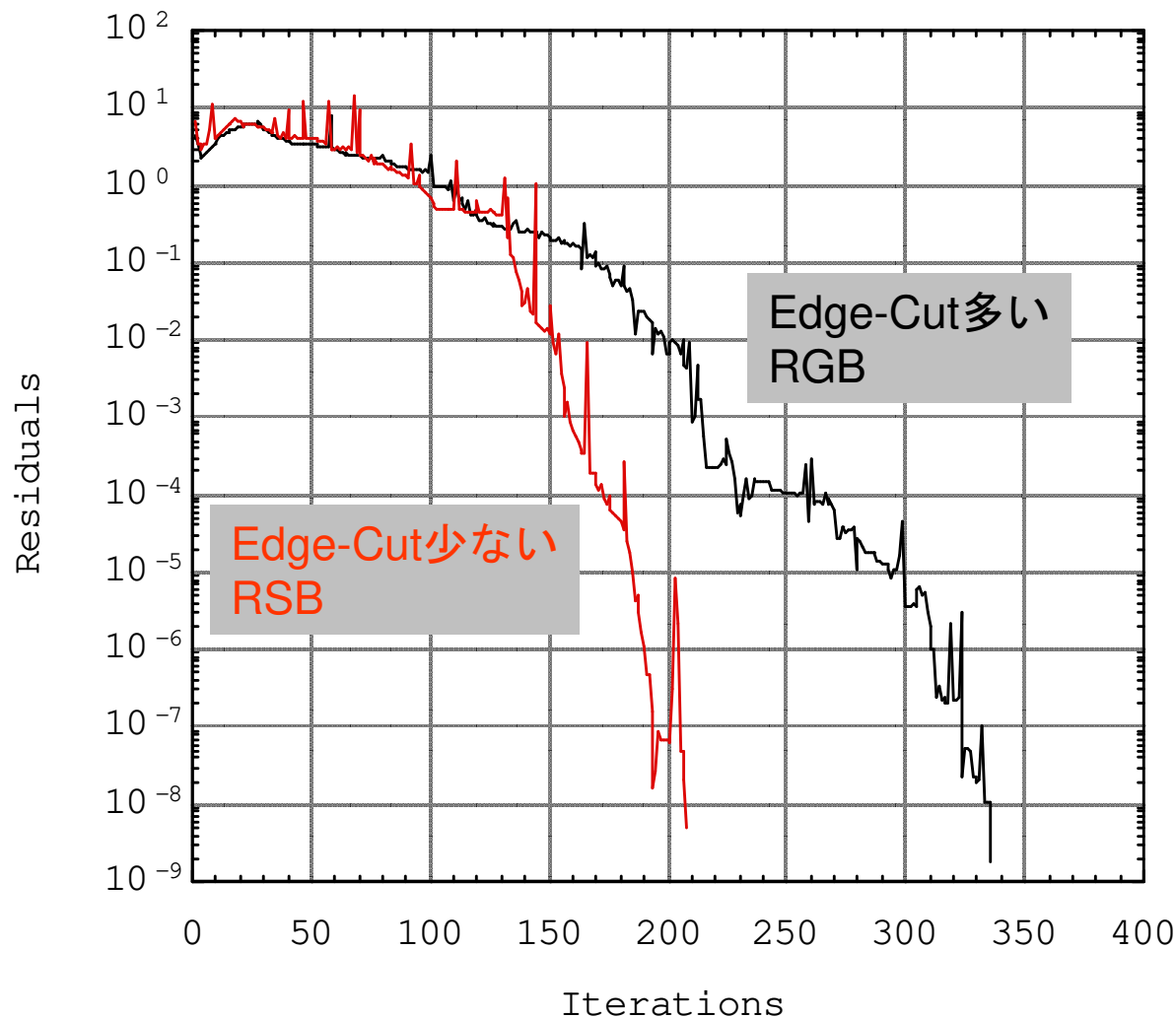
# Partitioning の反復法収束への影響

BiCGSTAB with Localized ILU(0) Preconditioning

15X15 region, RGB/RSB for 16 PE's, Poisson eqn's

Edge-Cutが少ないほど（通信が少ないほど）収束は速い

今回は前処理が対角スケーリングなので無関係だが



	<b>RGB</b>	<b>RSB</b>
Neighboring PEs (Ave., max)	3.63, 7	3.63, 6
Boundary Edges (Ave, max)	15.1, 19	12.5, 18

1996年2月頃  
やった計算

# Partitioning手法

- 嘗ては多くの研究グループがあったが今は、**MeTiS** (ミネソタ大学) と **Scotch/PT-Scotch** にほぼ集約
- **MeTiS**: Univ. Minnesota
  - <http://glaros.dtc.umn.edu/gkhome/views/metis/>
- **Scotch/PT-Scotch**: 比較的最近
  - <http://www.labri.fr/perso/pelegrin/scotch/>
- **JOSTLE**: Univ. Greenwich
  - <http://staffweb.cms.gre.ac.uk/~c.walshaw/jostle/>

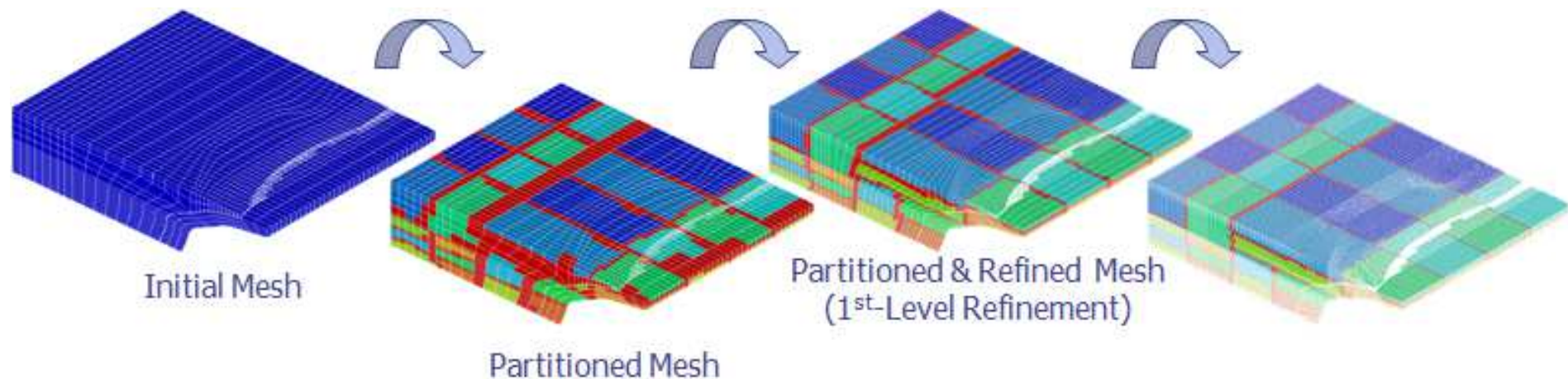
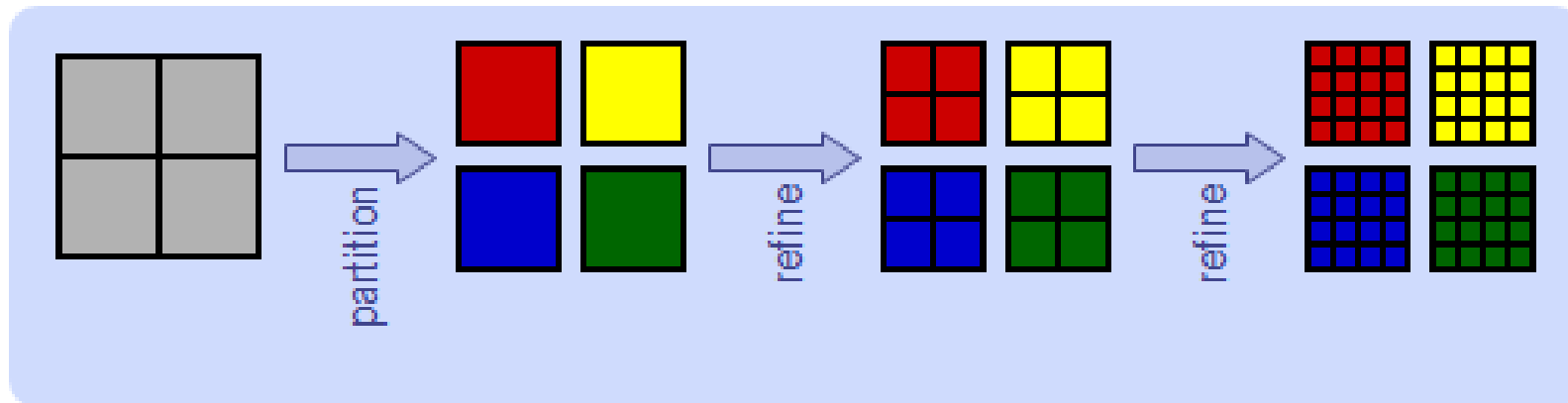


# ~/pFEM/pfem3d/mesh/part

- 初期全体メッシュデータを対象とした簡易ツール
  - シリアル処理
- 初期全体メッシュデータを入力として，局所分散メッシュデータ，通信情報を出力する。
- 分割手法
  - RCB (Recursive Coordinate Bisection) 法
  - METIS
    - kmetis                    領域間通信最小 (edge-cut最小)
    - pmetis                    領域間バランス最適化

# 実際の大規模計算

- そもそも「初期全体メッシュ」を単一ファイルとして用意できない場合もある。
- 「粗い」初期メッシュ→分割→整合性をとりながら局所的に細分化，という方式が適用されることが多い

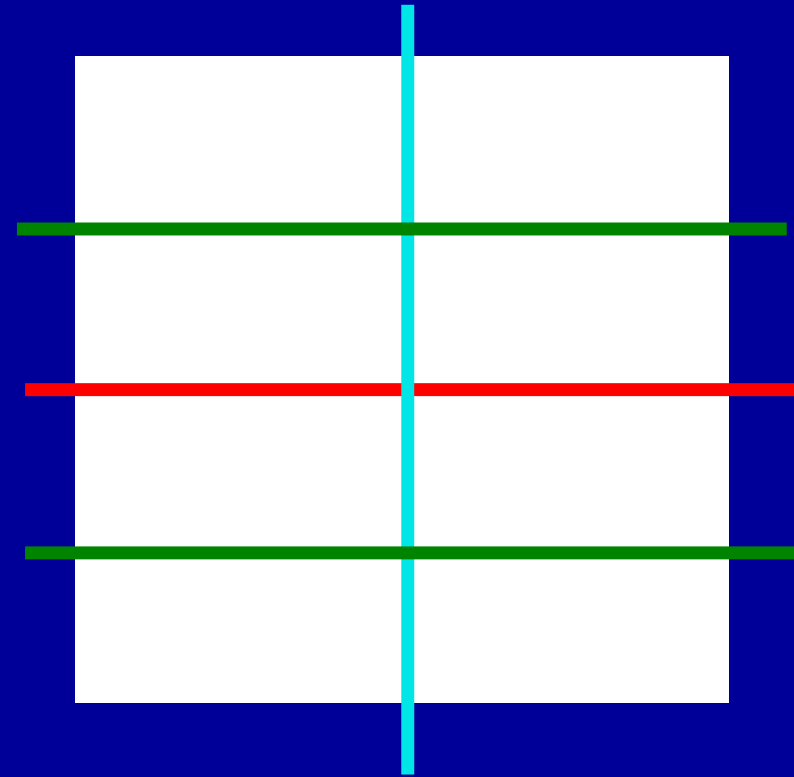


# RCB法

## Recursive Coordinate Bisection

H.D.Simon "Partitioning of unstructured problems for parallel processing", Comp. Sys. in Eng., Vol.2, 1991.

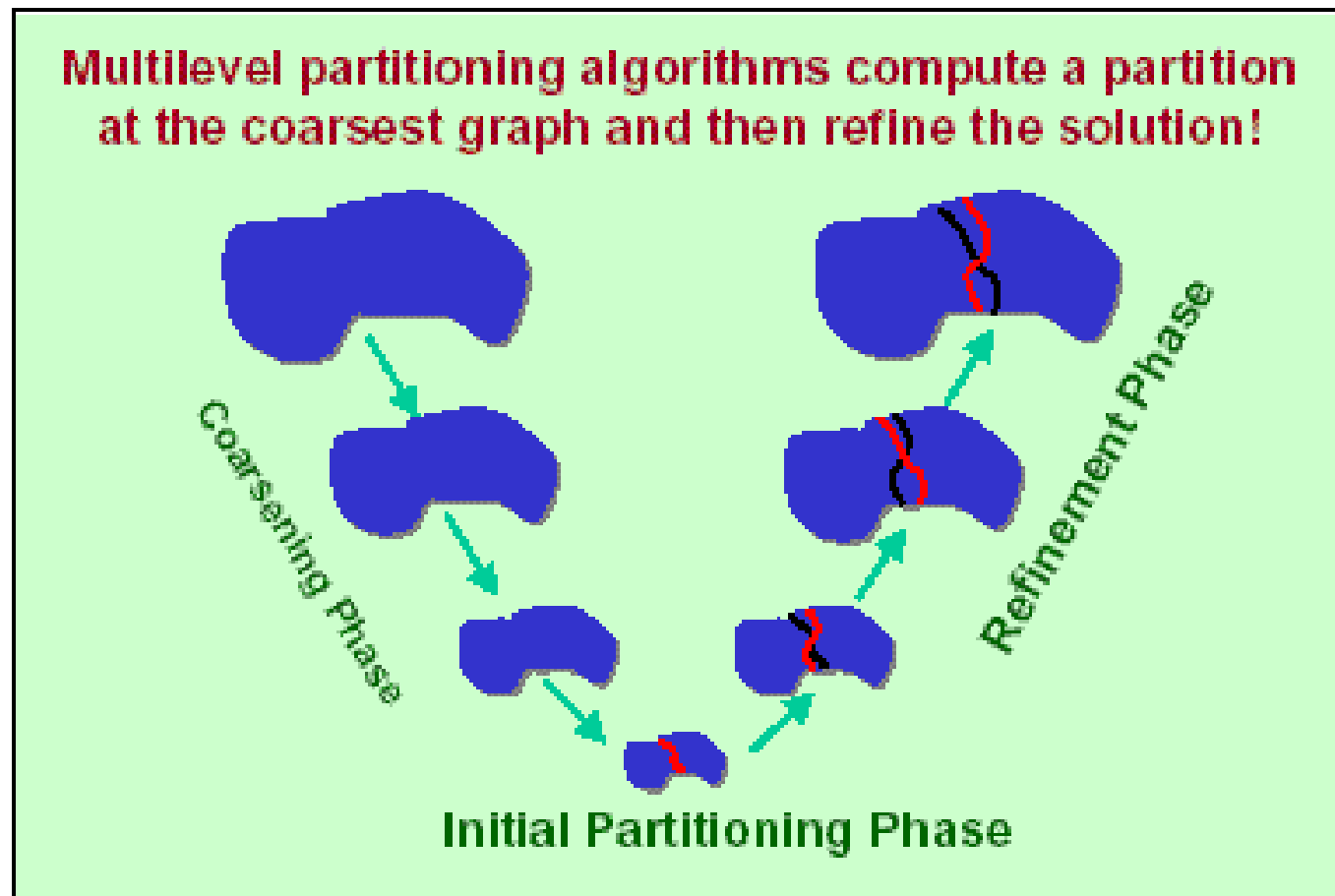
- XYZ座標成分の大小をとりながら分割
- 分割基準軸は形状に応じて任意に選択できる
- たとえば細長い形状では同じ方向への分割を続ける
- 2<sup>n</sup>領域の分割しかできない
- 高速, 簡易形状ではMETISより良い



# METIS

<http://glaros.dtc.umn.edu/gkhome/views/metis/>

- マルチレベルグラフ理論に基づいた方法



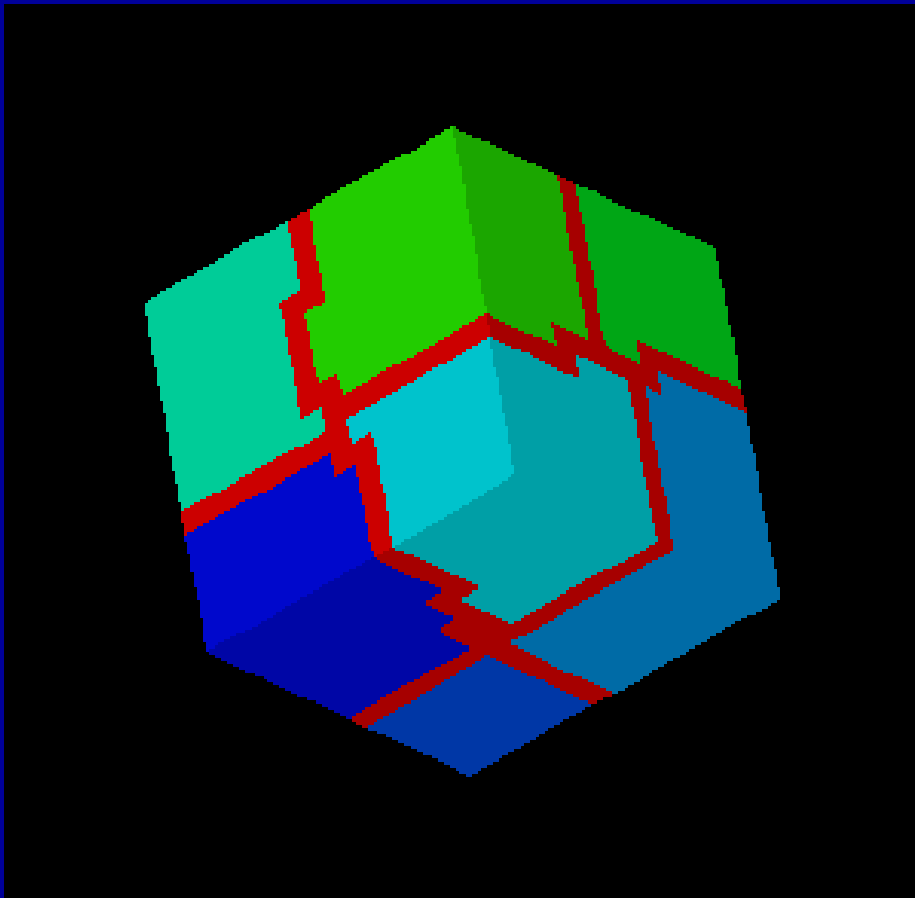
# METIS

<http://glaros.dtc.umn.edu/gkhome/views/metis/>

- マルチレベルグラフ理論に基づいた方法
  - 特に通信 (edge-cut) が少ない分割を提供する
  - 安定, 高速
  - フリーウェア, 他のプログラムに組み込むことも容易
- 色々な種類がある
  - k-METIS      通信量 (edge-cut) 最小
  - p-METIS      領域間バランス最適化
  - ParMETIS      並列版
  - 領域分割だけでなく, オーダリング, データマイニングなど色々な分野に使用されている
    - 接触, 衝突問題における並列接触面探索

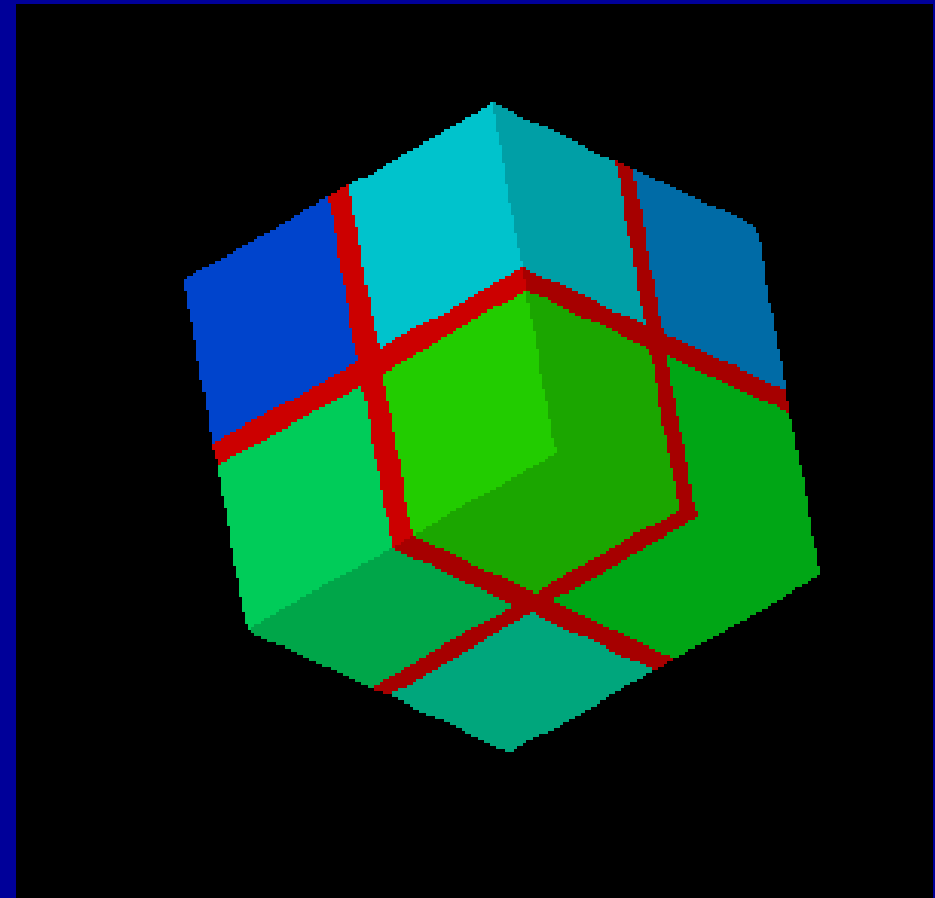
# 領域分割例：立方体領域：8分割

3,375要素 ( $=15^3$ ) , 4,096節点  
単純な形状ではむしろRCBが良い



**k-METIS**

edgecut = 882



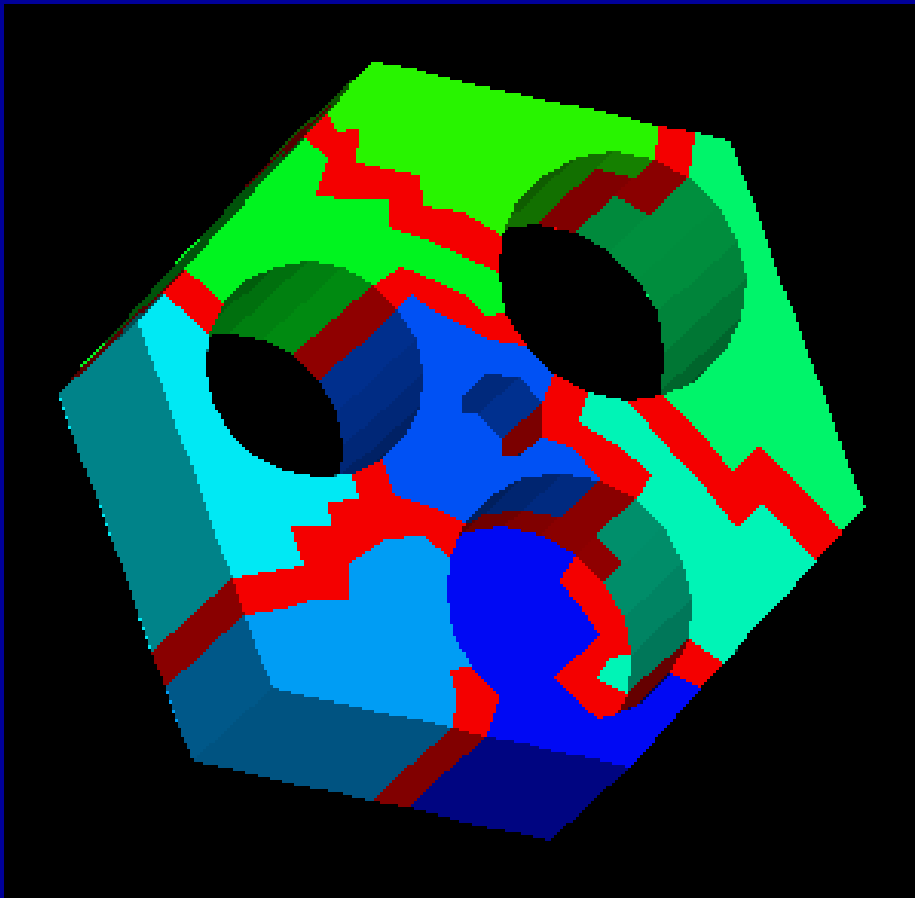
**RCB**

edgecut = 768

# 領域分割例：黒鉛ブロック：8分割

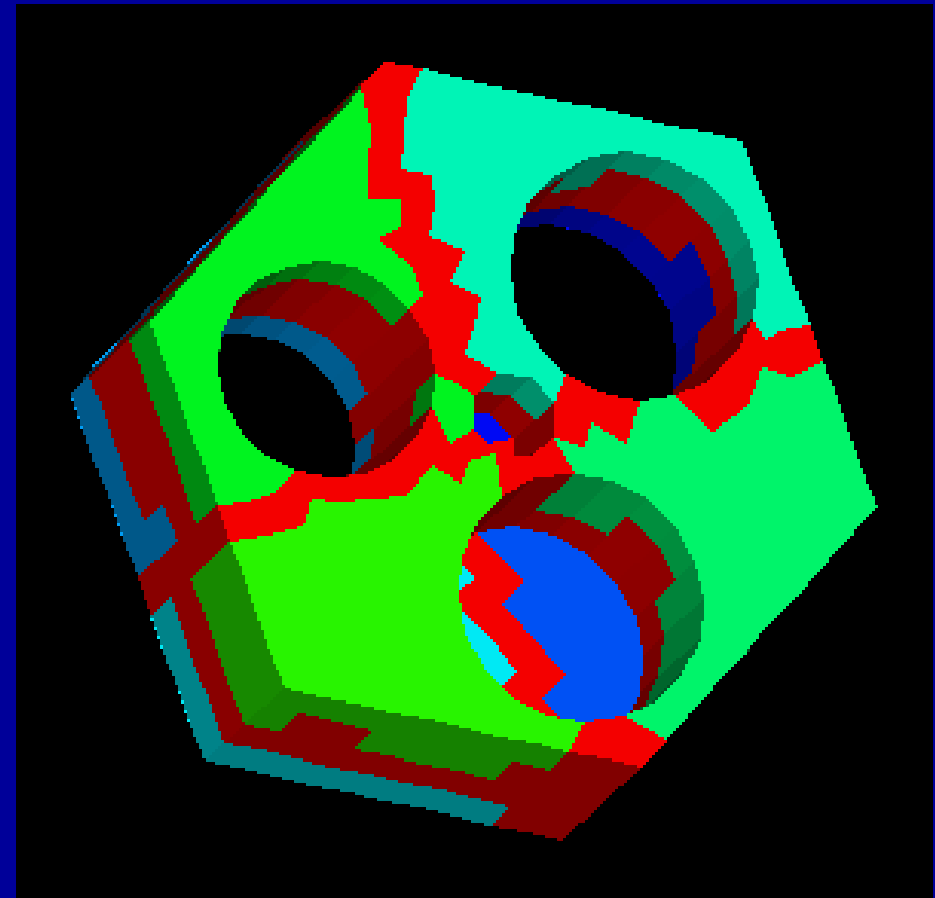
795要素, 1,308節点

複雑形状ではMETISが良い：Overlap領域細かい



**k-METIS**

edgcut = 307



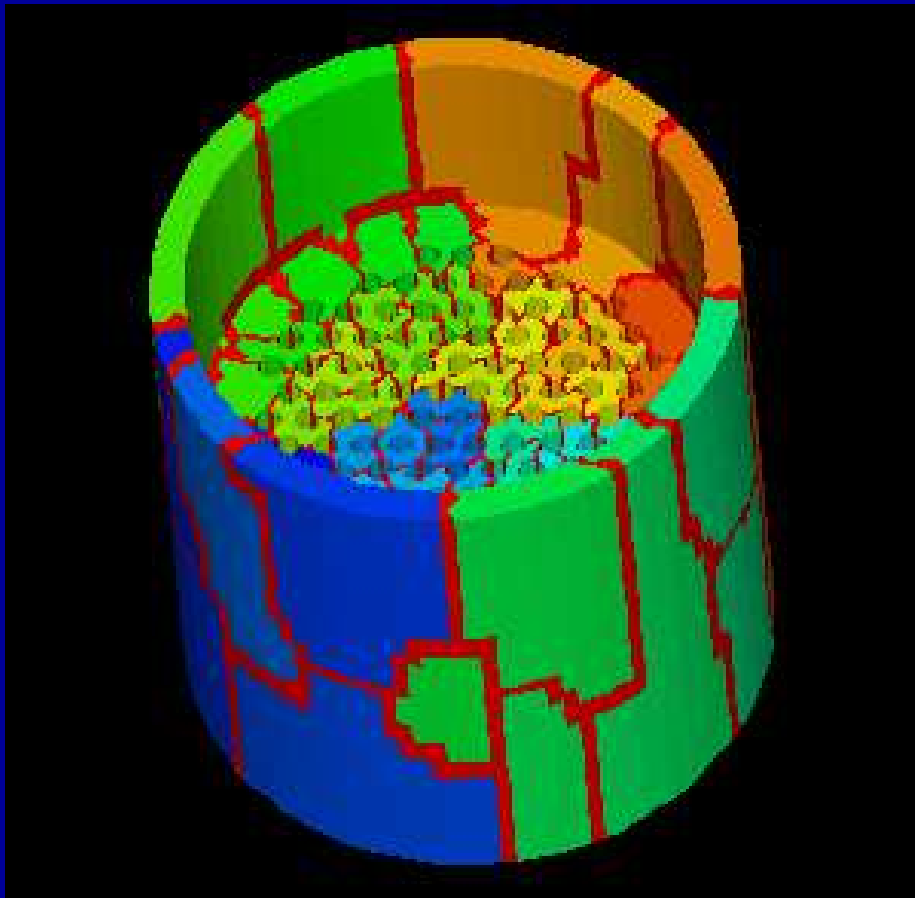
**RCB**

edgcut = 614

# 領域分割例：管板：64分割

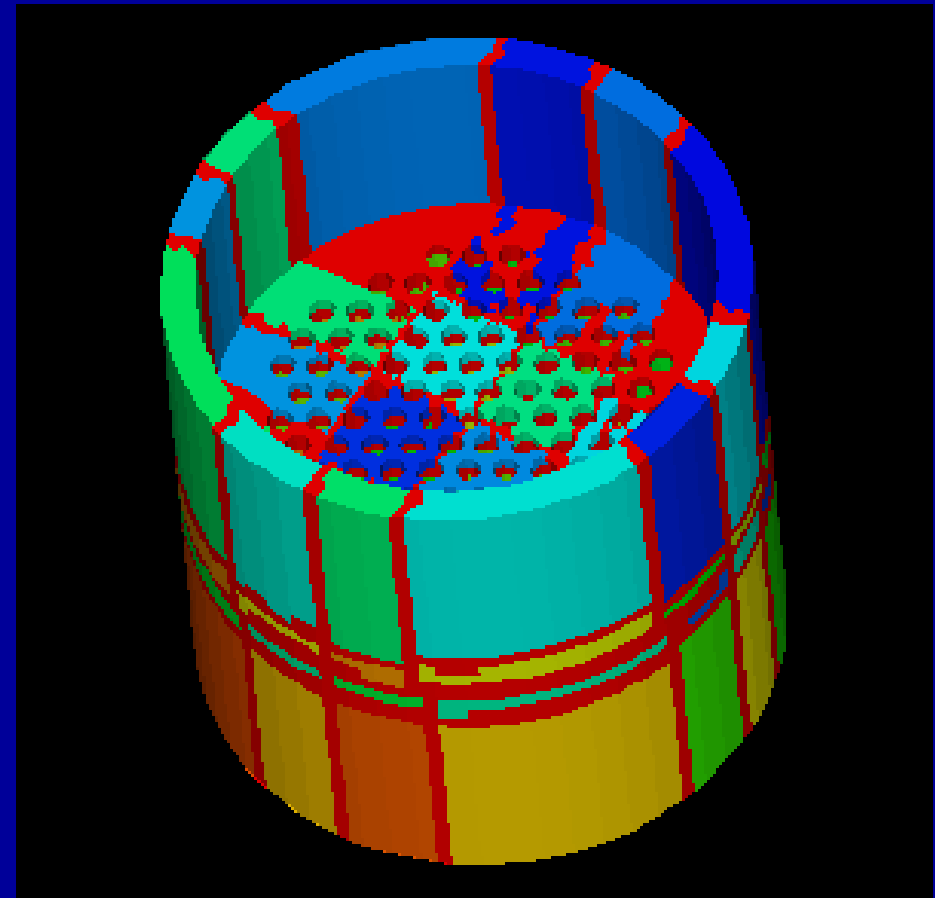
40,416要素, 54,084節点

複雑形状ではMETISが良い：EdgeCut少ない



**k-METIS**

edgecut = 9,489



**RCB**

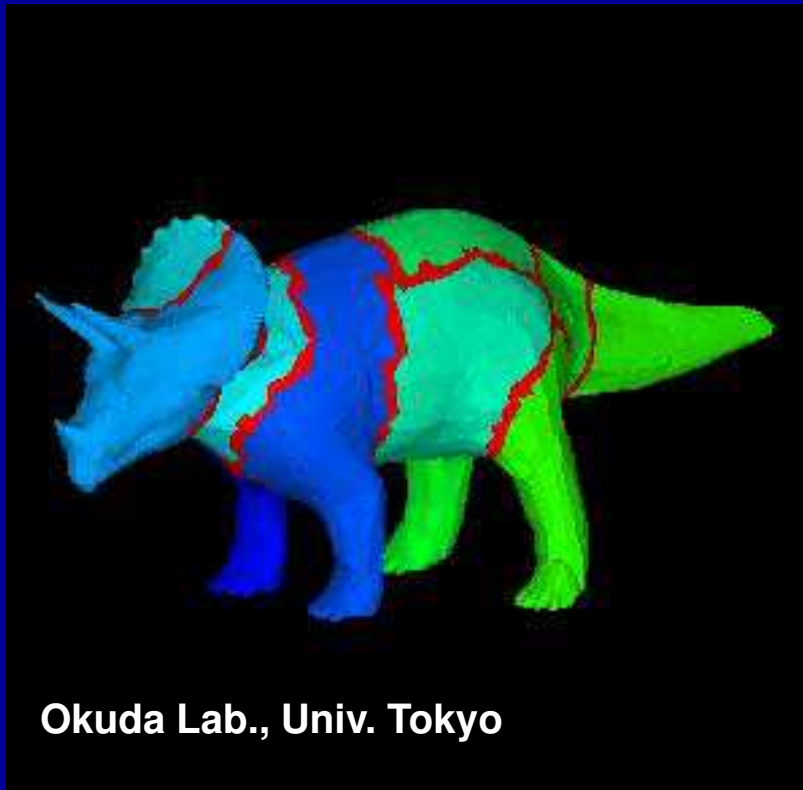
edgecut = 28,320



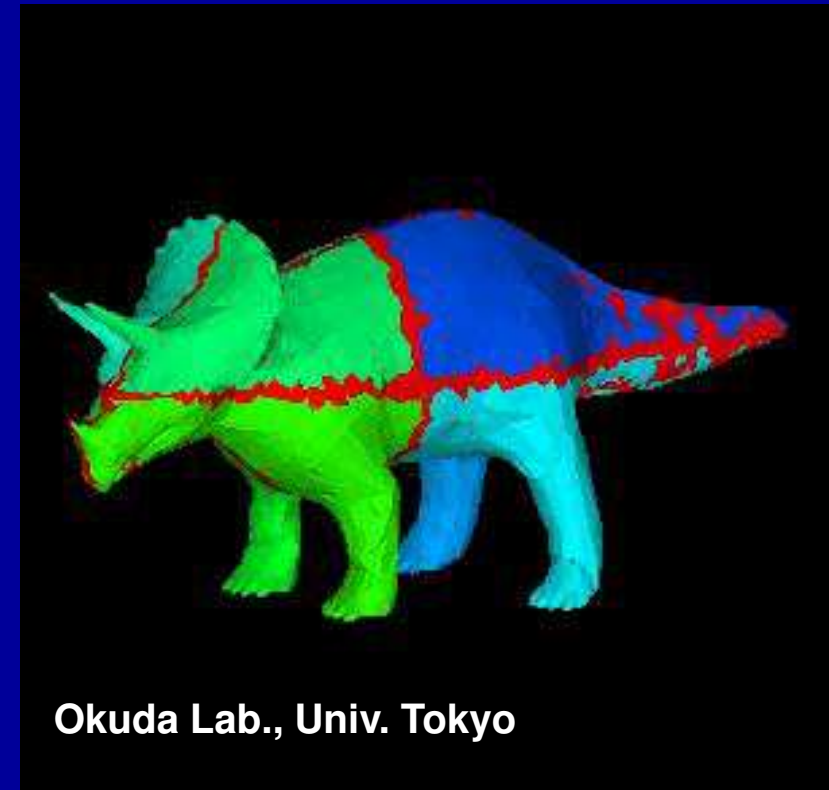
# Strange Animal in 8 PEs

53,510 elements, 11,749 nodes.

METIS is better for complicated geometries.



**k-METIS**  
edgecut = 4,573

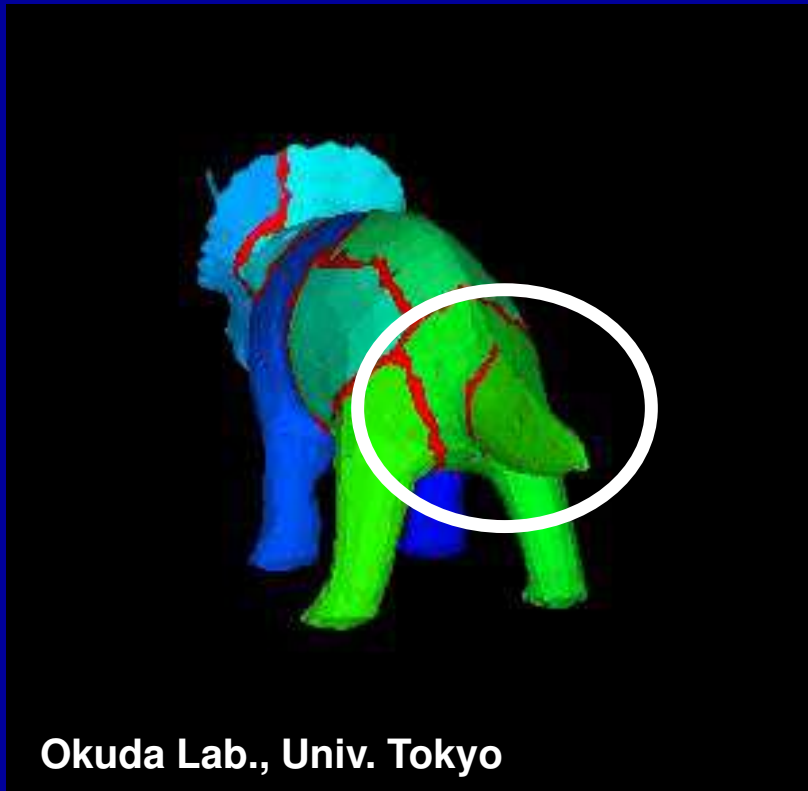


**RCB**  
edgecut = 7,898

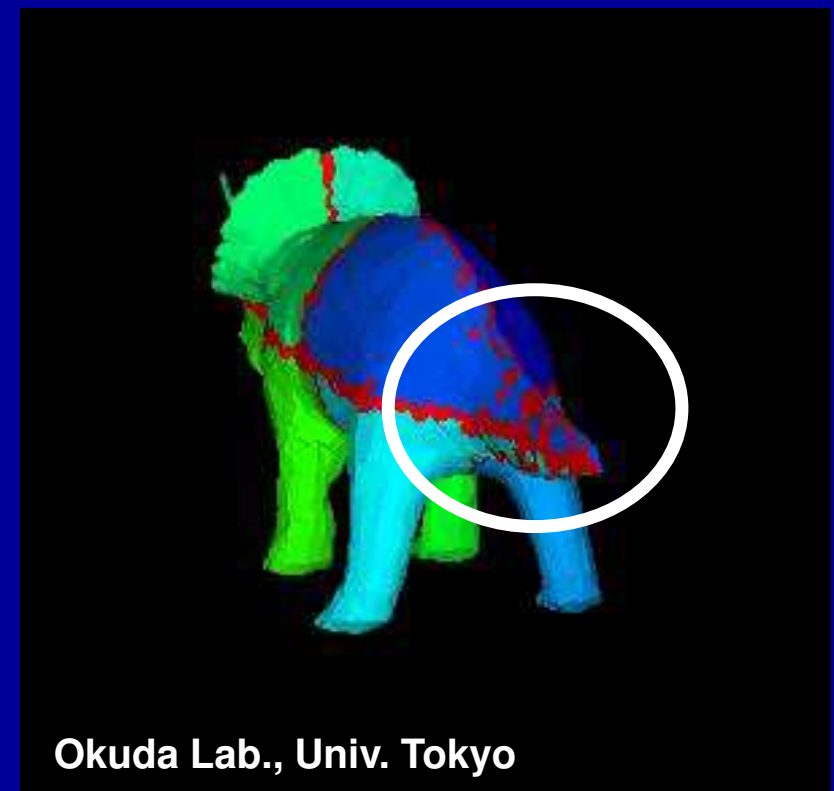
# Strange Animal in 8 PEs

53,510 elements, 11,749 nodes.

METIS is better for complicated geometries.



**k-METIS**  
edgecut = 4,573



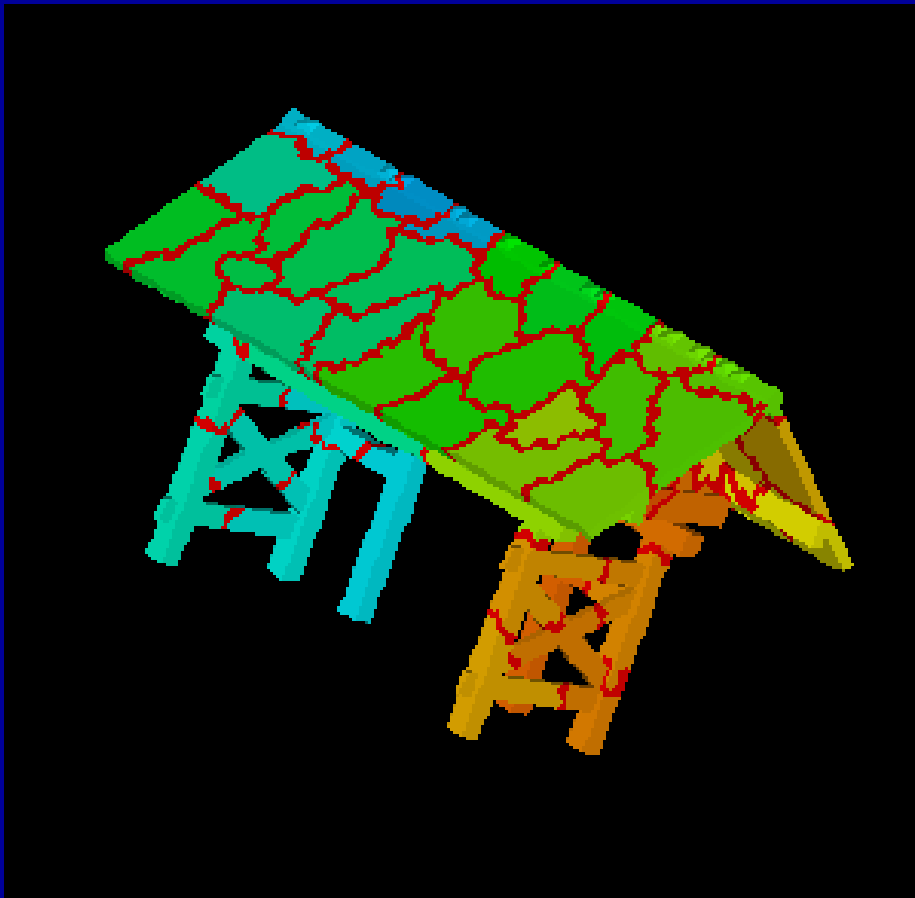
**RCB**  
edgecut = 7,898

# 領域分割例：東大赤門：64分割

movie

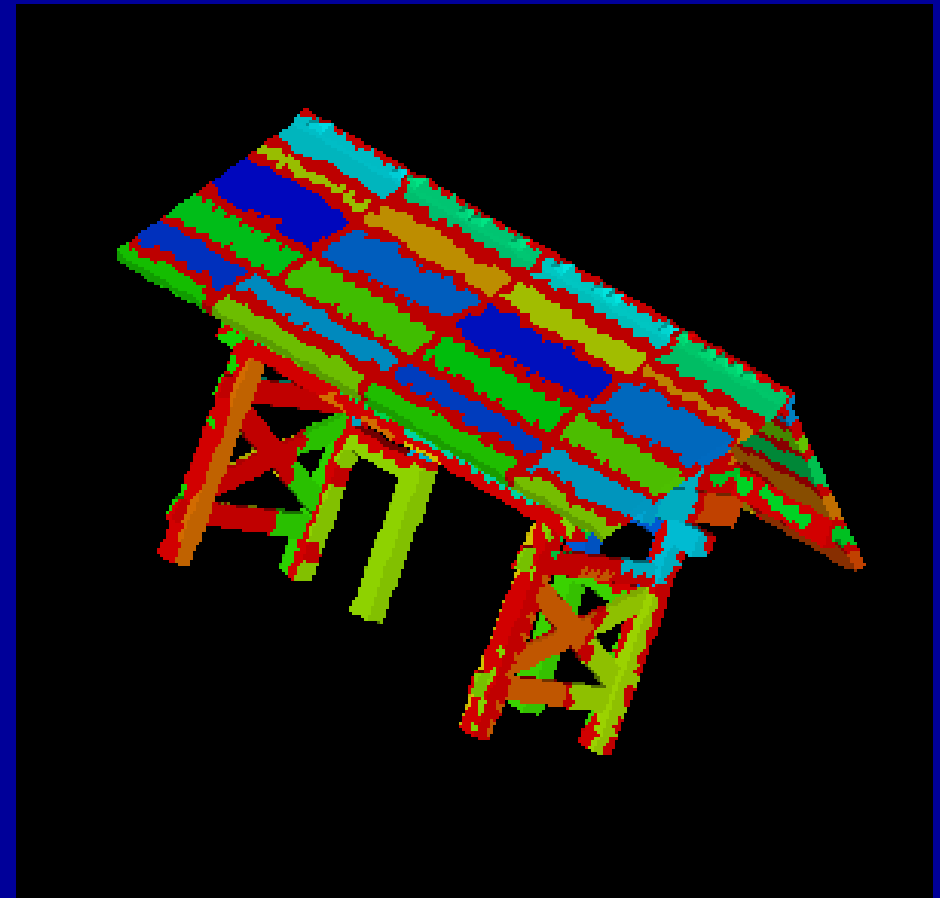
40,624要素, 54,659節点

複雑形状ではMeTiSが良い：EdgeCut少ない



**k-MeTiS**

edgecut = 7,563

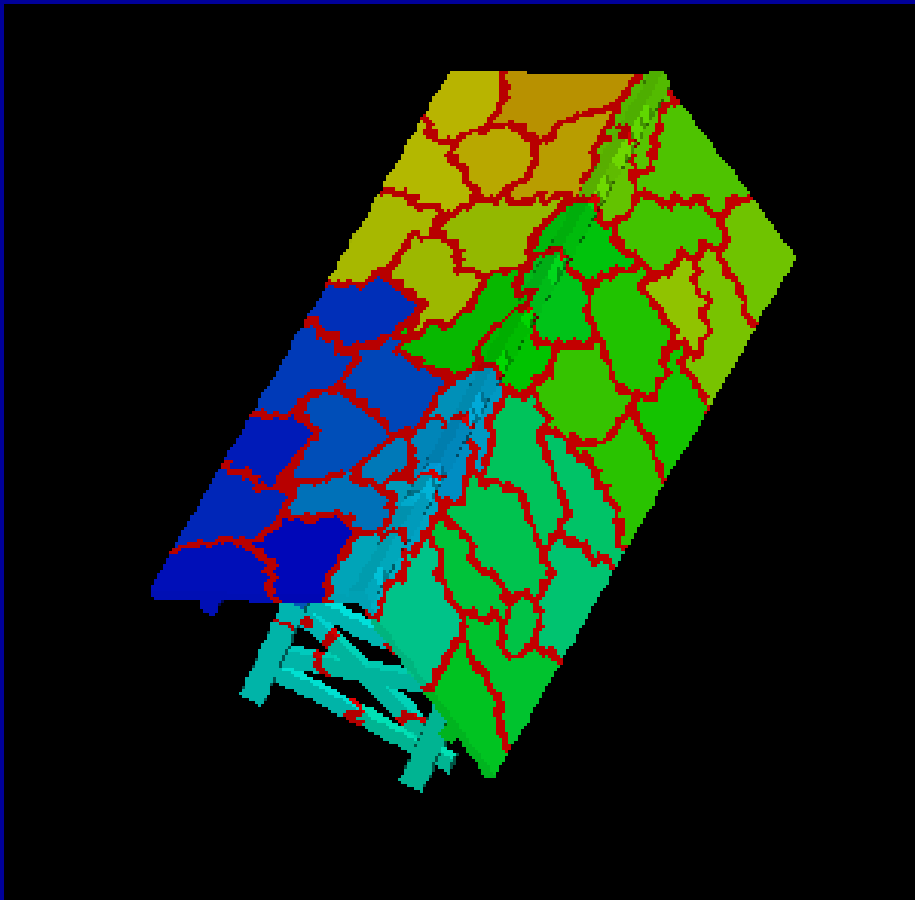


**RCB**

edgecut = 18,624

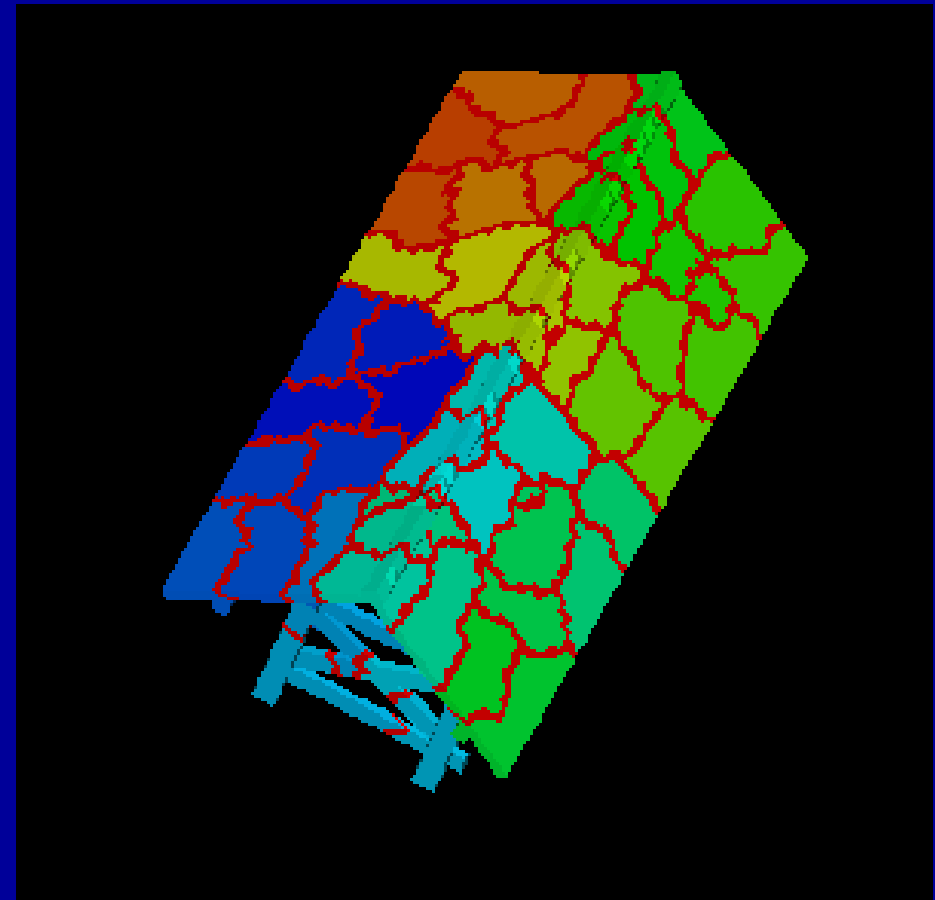
# 領域分割例：東大赤門：64分割

40,624要素, 54,659節点



**k-METIS**

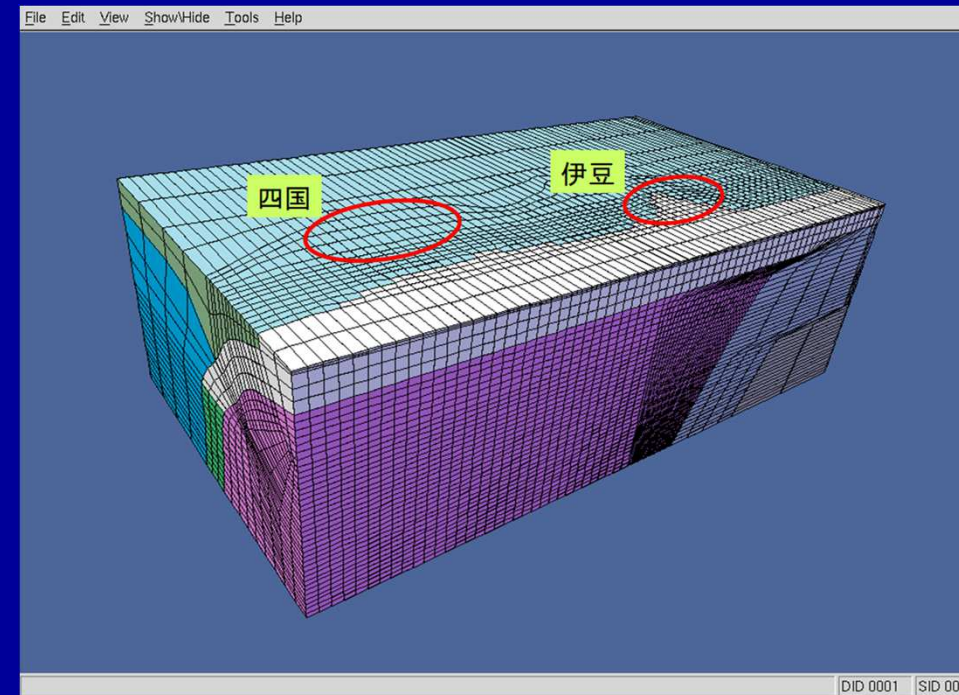
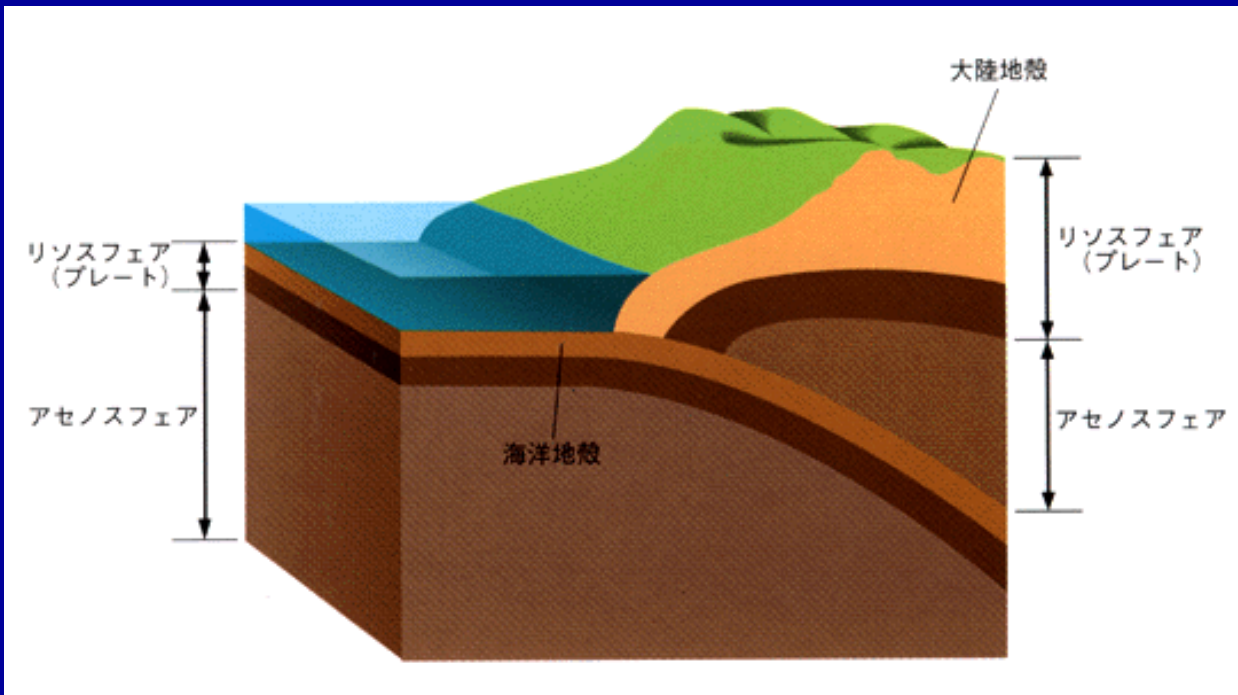
Load Balance= 1.03  
edgecut = 7,563



**p-METIS**

Load Balance= 1.00  
edgecut = 7,738

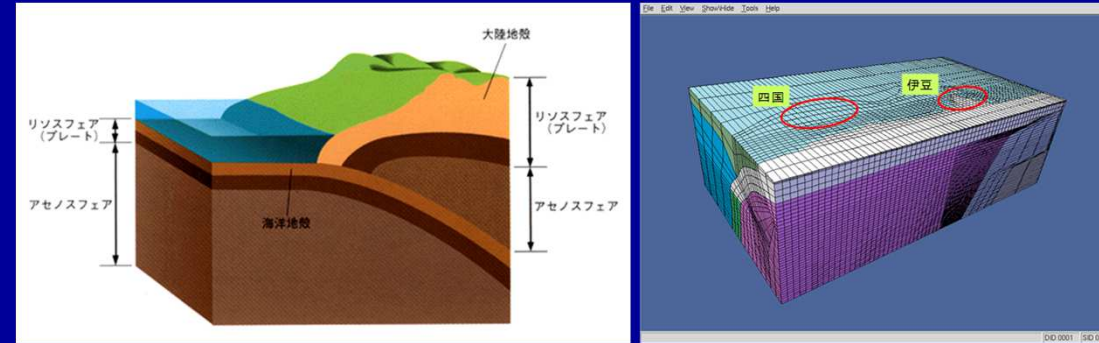
# 領域分割例： 西南日本



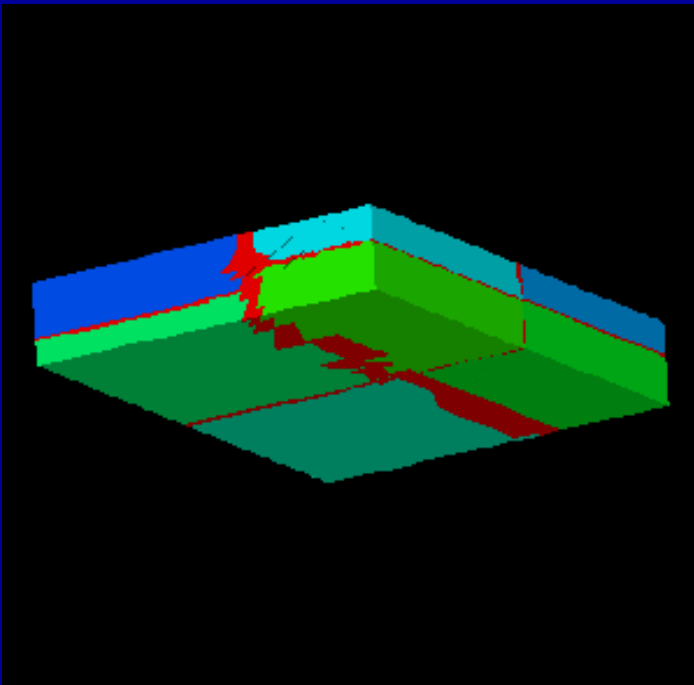
# 領域分割例：

## 西南日本：8分割

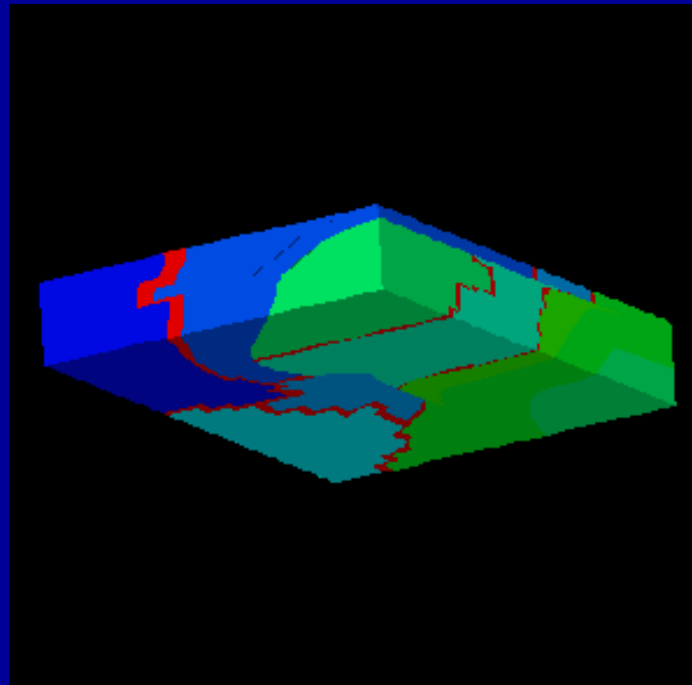
57,205要素, 58,544節点



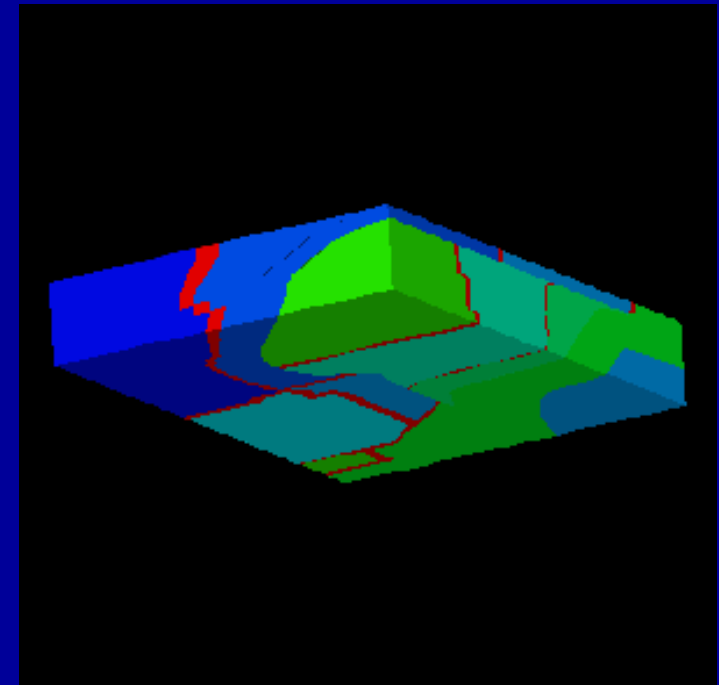
movie



RCB e.c.=7433



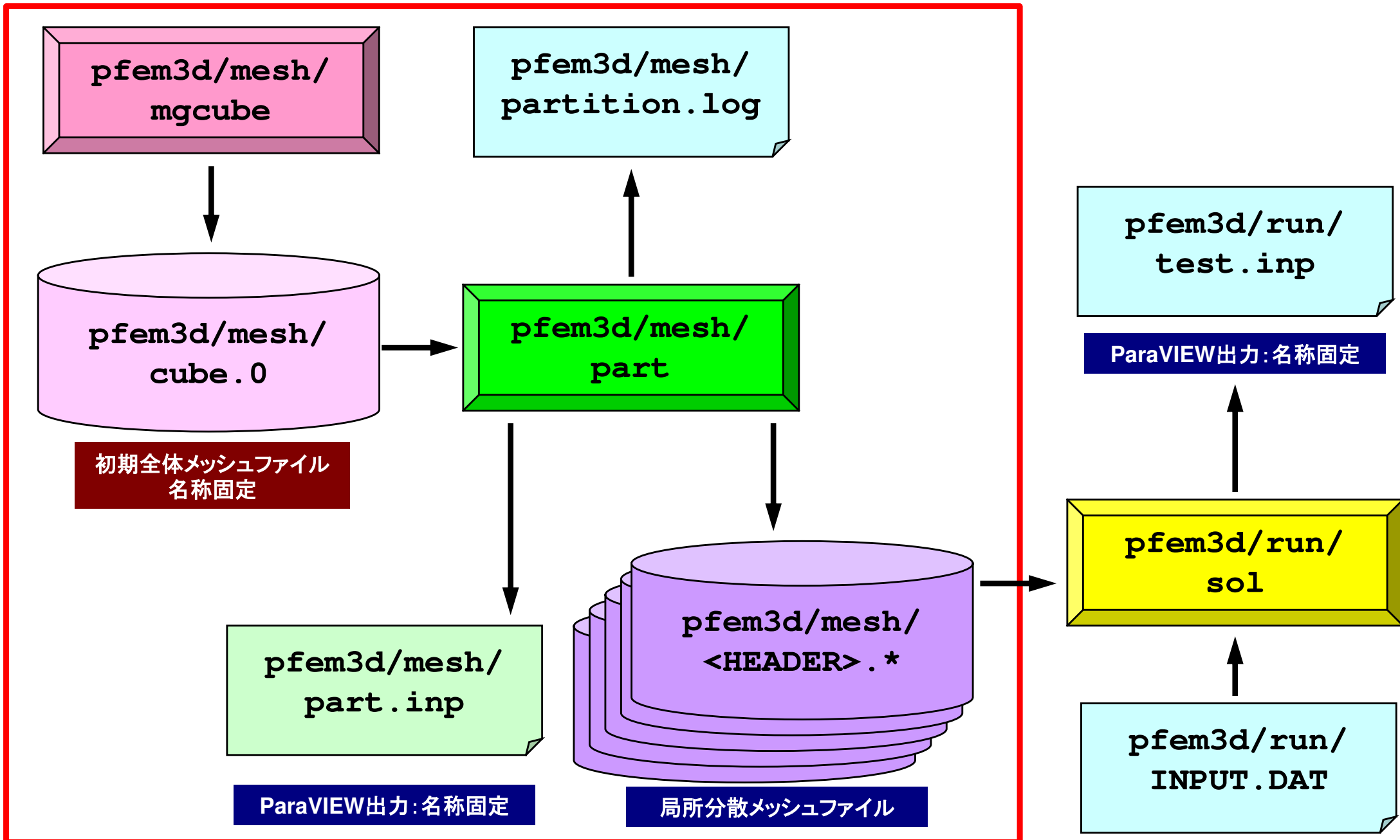
k-METIS :4,221



p-METIS :3,672

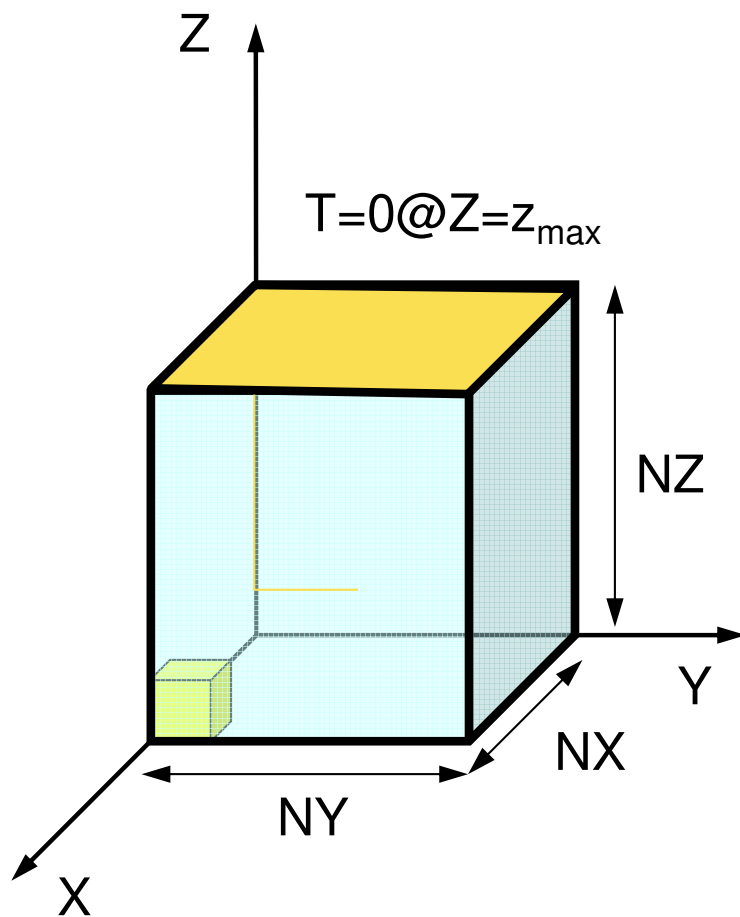
- プログラムのインストール
- 実行
  - 並列有限要素法の手順
  - 領域分割とは?
  - 本当の実行
- データ構造

# 並列有限要素法の手順





# 初期全体メッシュ生成



```
>$ cd /work/gt00/t00XYZ/pFEM/pfem3d/mesh  
>$ ./mgcube
```

```
NX, NY, NZ
```

← 各辺長さを  
訊いてくる  
← このように  
入れてみる

```
20, 20, 20
```

```
>$ ls cube.0      生成を確認
```

```
cube.0
```

とやりたいところだが、OBCX上でこれはできない。

# ということでバッチジョブでお願いします

```
>$ cd /work/gt00/t00XYZ/pFEM/pfem3d/mesh
>$ pjsub mg.sh
```

```
...
>$ ls cube.0      生成を確認
```

**cube.0**

## mg.sh

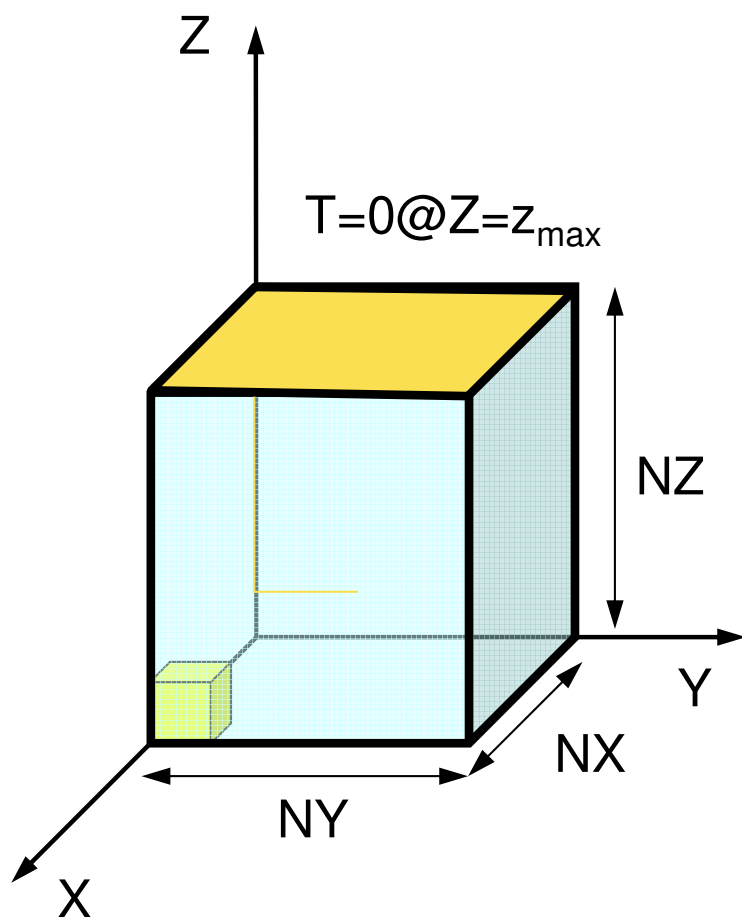
```
#!/bin/sh
#PJM -L rscgrp=lecture-o
#PJM -L node=1
#PJM --mpi proc=1
#PJM -L elapse=00:15:00
#PJM -g gt00
#PJM -j
#PJM -e err
#PJM -o mg.lst
```

```
module load fj
module load fjmpi
```

```
./mgcube < inp_mg
```

## inp\_mg

```
20 20 20
```



# 領域分割

- 初期全体メッシュファイル名 (cube.0)
  - バイナリ出力です (スパコン上)
- 分割方法 (RCB, METIS)
- 分散メッシュファイルヘッダー
  - “work” という名前を使ってはいけない
  
- RCB
  - 分割数, 分割座標軸
- METIS (kmetis, pmetis)
  - 分割数

# <\$O-TOP>/pfem3d/part/Makefile

```
F77      = frtpx
F90      = frtpx
FLINKER  = $(F77)
F90LINKER = $(F90)
FLIB_PATH =
INC_DIR  =
OPTFLAGS = -Kfast
FFLAGS  = $(OPTFLAGS)
FLIBS   = -lmetis

TARGET = ../mesh/part
default: $(TARGET)
OBJS =¥
geofem_util.o partitioner.o input_grid.o main.o ¥
calc_edgcut.o cre_local_data.o define_file_name.o ¥
interface_nodes.o metis.o¥
neib_pe.o paraset.o proc_local.o local_data.o¥
double_numbering.o output_ucd.o util.o

$(TARGET):  $(OBJS)
             $(F90LINKER) $(OPTFLAGS) -o $(TARGET) $(OBJS) $(FLIBS)
clean:
    /bin/rm -f *.o $(TARGET) *~ *.mod
.f.o:
    $(F90) $(FFLAGS) $(INC_DIR) -c  $*.f
.SUFFIXES: .f
```

```

>$ cd ~/pFEM/pfem3d/mesh
>$ ./part

Original GRID-FILE ?
cube.0
* INODTOT =      9261
* GRID
* IELMTOT =      8000
* ELM
* BOUNDARY : NODE group
Xmin
Ymin
Zmin
Zmax
* IEDGTOT =      26460      37044

# select PARTITIONING METHOD
RCB                (1)
K-METIS            (2)
P-METIS            (3)

Please TYPE 1 or 3 or 4 !!

>>>
1

*** RECURSIVE COORDINATE BISECTION (RCB)
How many partitions (2**n)?

>>>
3

***      8 REGIONS

```

```

# HEADER of the OUTPUT file ?
HEADER should not be <work>

>>>
aaa

##### 1-th BiSECTION #####

in which direction ? X:1, Y:2, Z:3

>>>
1
X-direction

##### 2-th BiSECTION #####

in which direction ? X:1, Y:2, Z:3

>>>
2
Y-direction

##### 3-th BiSECTION #####

in which direction ? X:1, Y:2, Z:3

>>>
3
Z-direction

RECURSIVE COORDINATE BISECTION

*** GRID file

      8 PEs

TOTAL EDGE      #      26460
TOTAL EDGE CUT #      1593

TOTAL NODE      #      9261
TOTAL CELL      #      8000

```

PE	NODE#	CELL#
0	1158	1223
1	1158	1188
2	1158	1222
3	1158	1176
4	1158	1188
5	1157	1179
6	1157	1188
7	1157	1175

MAX.node/PE	1158
MIN.node/PE	1157
MAX.cell/PE	1223
MIN.cell/PE	1175

OVERLAPPED ELEMENTS 1373

PE/NEIB-PE#	NEIB-PEs						
0 7	7	6	4	5	2	1	3
1 7	7	5	6	0	2	4	3
2 7	7	6	0	5	1	4	3
3 6	7	2	6	1	5	0	
4 6	6	7	5	0	2	1	
5 7	7	6	4	0	1	2	3
6 7	7	5	4	0	2	1	3
7 7	6	5	4	0	2	1	3

PE: 0	1626	1158	468	435
PE: 1	1589	1158	431	411
PE: 2	1620	1158	462	490
PE: 3	1560	1158	402	409
PE: 4	1574	1158	416	421
PE: 5	1565	1157	408	397
PE: 6	1580	1157	423	414
PE: 7	1564	1157	407	440

(内点+外点)数    内点数    外点数    境界点数

KCHF091R STOP \* normal termination

```
>$ ls -l aaa.*
```

```
-rw-r--r-- 1 t18013 t18 268829 Jan 12 14:57 aaa.0
-rw-r--r-- 1 t18013 t18 261490 Jan 12 14:57 aaa.1
-rw-r--r-- 1 t18013 t18 268086 Jan 12 14:57 aaa.2
-rw-r--r-- 1 t18013 t18 257631 Jan 12 14:57 aaa.3
-rw-r--r-- 1 t18013 t18 258719 Jan 12 14:57 aaa.4
-rw-r--r-- 1 t18013 t18 256853 Jan 12 14:57 aaa.5
-rw-r--r-- 1 t18013 t18 259093 Jan 12 14:57 aaa.6
-rw-r--r-- 1 t18013 t18 257161 Jan 12 14:57 aaa.7
```

## • 局所分散メッシュファイル

- <HEADER>.<領域番号>
- 領域番号は「0」から(MPIの都合)

とやりたいところだが、OBCX上で  
これはできない。ということで再び  
バッチジョブでお願いいたします。

## part\_rcb.sh

```
#!/bin/sh
#PJM -N "RCB"
#PJM -L rscgrp=tutorial-o
#PJM -L node=1
#PJM --mpi proc=1
#PJM -L elapse=00:15:00
#PJM -g gt00
#PJM -j
#PJM -e err
#PJM -o test.lst
```

```
module load fj
module load fjmpi
module load metis/4.0.3
```

```
./part < inp_rcb
rm work.*
```

# RCB: part\_rcb.sh inp\_rcb

## inp\_rcb

```
cube.0 Initial Global File (fixed)
1      1:RCB, 2:KMETIS, 3:PMETIS
3      m: 2m PE's
aaa    Header of Distributed Local Files
1      Reference Axis (X:1, Y:2, Z:3)
2
3
```

## inp\_rcb: 1-PE

```
cube.0 Initial Global File (fixed)
1      1:RCB, 2:KMETIS, 3:PMETIS
0      m: 2m PE's
aaa    Header of Distributed Local Files
```

# kmetis: part\_kmetis.sh inp\_kmetis

## Minimum Edge-Cut

### part\_kmetis.sh

```
#!/bin/sh
#PJM -N "K-MeTiS"
#PJM -L rscgrp=tutorial-o
#PJM -L node=1
#PJM --mpi proc=1
#PJM -L elapse=00:15:00
#PJM -g gt00
#PJM -j
#PJM -e err
#PJM -o test.lst
```

```
module load fj
module load fjmpi
module load metis/4.0.3
```

```
./part < inp_kmetis
rm work.*
```

### inp\_kmetis

cube.0	Initial Global File (fixed)
2	1:RCB, 2:KMETIS, 3:PMETIS
8	Number of PE's
aaa	Header of Distributed Local Files



# pmetis: part\_pmetis.sh inp\_pmetis

## Optimum Load-Balancing

### part\_pmetis.sh

```
#!/bin/sh
#PJM -N "P-MeTiS"
#PJM -L rscgrp=tutorial-o
#PJM -L node=1
#PJM --mpi proc=1
#PJM -L elapse=00:15:00
#PJM -g gt00
#PJM -j
#PJM -e err
#PJM -o test.lst
```

```
module load fj
module load fjmpi
module load metis/4.0.3
```

```
./part < inp_pmetis
rm work.*
```

### inp\_pmetis

```
cube.0  Initial Global File (fixed)
3       1:RCB, 2:KMETIS, 3:PMETIS
8       Number of PE's
aaa     Header of Distributed Local Files
```

# partition.log

RECURSIVE COORDINATE BISECTION

\*\*\* GRID file

8 PEs

TOTAL EDGE # 26460  
TOTAL EDGE CUT # 1593

TOTAL NODE # 9261  
TOTAL CELL # 8000

PE	NODE#	CELL#
0	1158	1223
1	1158	1188
2	1158	1222
3	1158	1176
4	1158	1188
5	1157	1179
6	1157	1188
7	1157	1175

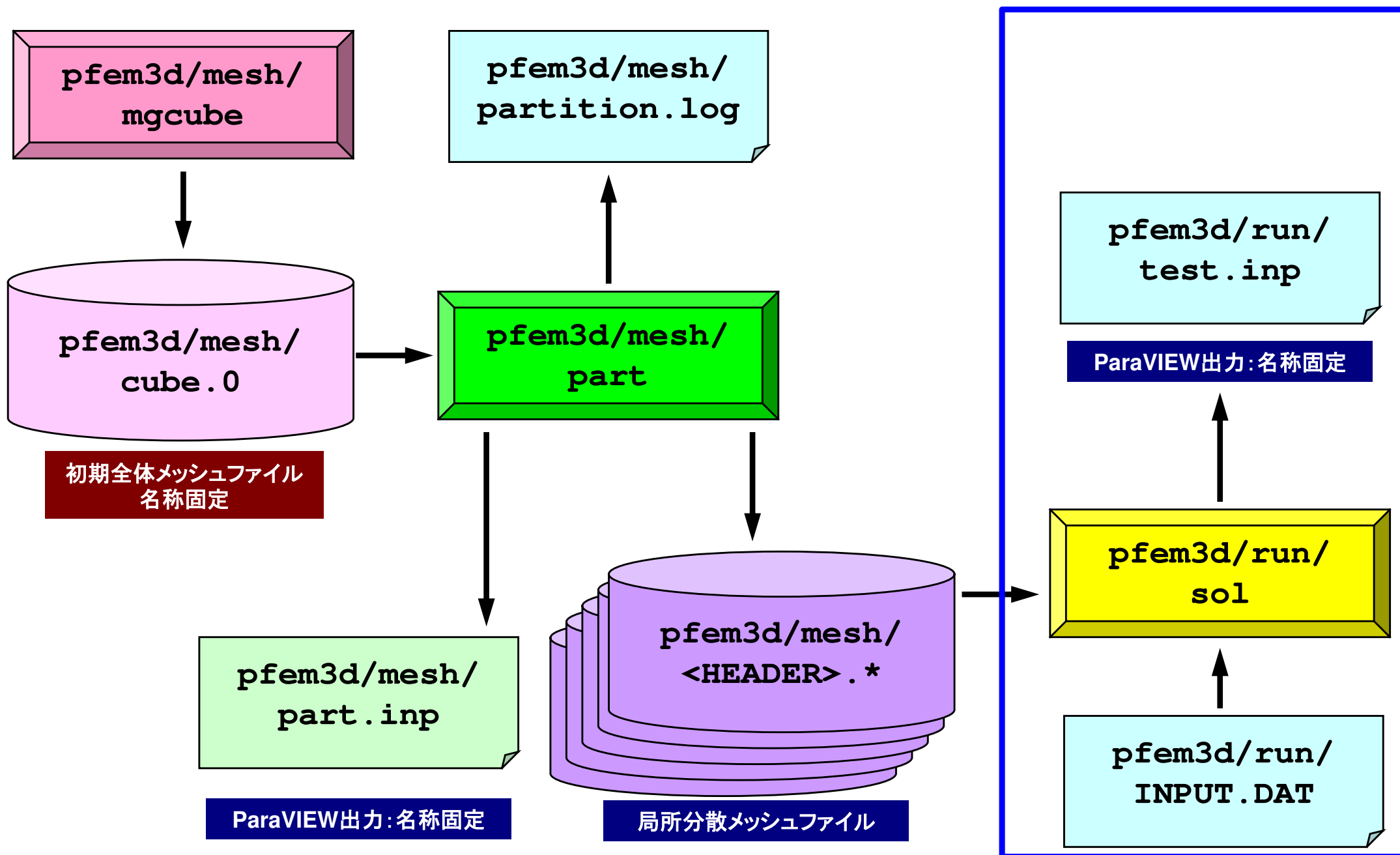
MAX.node/PE 1158  
MIN.node/PE 1157  
MAX.cell/PE 1223  
MIN.cell/PE 1175

OVERLAPPED ELEMENTS 1373

PE/NEIB-PE#	NEIB-PEs							
0	7	7	6	4	5	2	1	3
1	7	7	5	6	0	2	4	3
2	7	7	6	0	5	1	4	3
3	6	7	2	6	1	5	0	
4	6	6	7	5	0	2	1	
5	7	7	6	4	0	1	2	3
6	7	7	5	4	0	2	1	3
7	7	6	5	4	0	2	1	3

**$NX=NY=NZ=9$ , RCB :  $2^3$ 領域**

# 並列有限要素法の手順



# 制御ファイル: INPUT.DAT

## INPUT.DAT

```

../mesh/aaa    HEADER
2000           ITER
1.0 1.0       COND, QVOL
1.0e-08       RESID

```

- HEADER: 局所分散メッシュファイルのヘッダー
- ITER: 反復回数上限
- COND: 熱伝導率
- QVOL: 体積当たり発熱量係数
- RESID: 反復法の収束判定値

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

$$\dot{Q}(x, y, z) = QVOL |x_c + y_c|$$

# pFEM/pfem3d/run/a08.sh

```
#!/bin/sh
#PJM -N "flat-08"
#PJM -L rscgrp=tutorial-o
#PJM -L node=8
#PJM -mpi proc=384
#PJM -L elapse=00:15:00
#PJM -g gt00
#PJM -j
#PJM -e err
#PJM -o a08.lst

module load fj
module load fjmpi

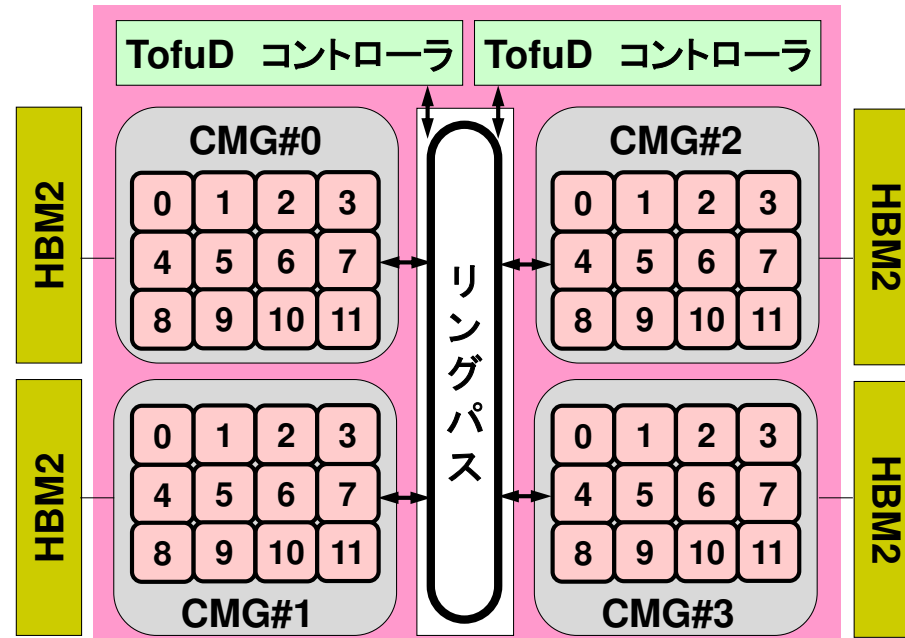
mpiexec ./sol
mpiexec numactl -l ./sol
```

Job Name
Name of "Queue/Resource Group"
Node #
Total MPI # (384/8= 48 per node)
Computation Time
Group Name (Wallet)
Standard Error
Standard Output

# プロセス数

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

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



- プログラムのインストール
- 実行
  - 並列有限要素法の手順
  - 領域分割とは？
  - 本当の実行
- **データ構造**



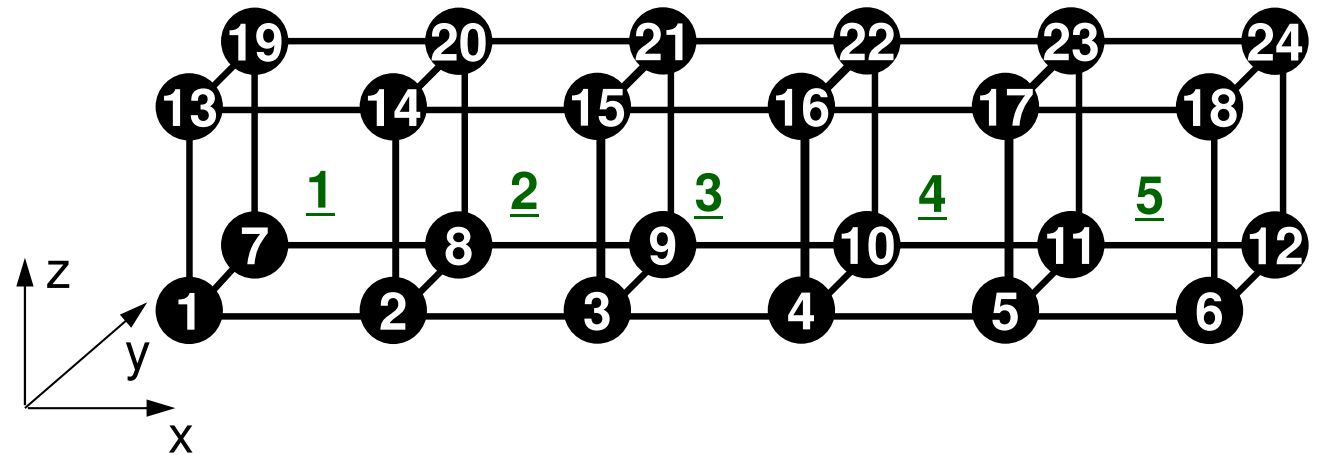
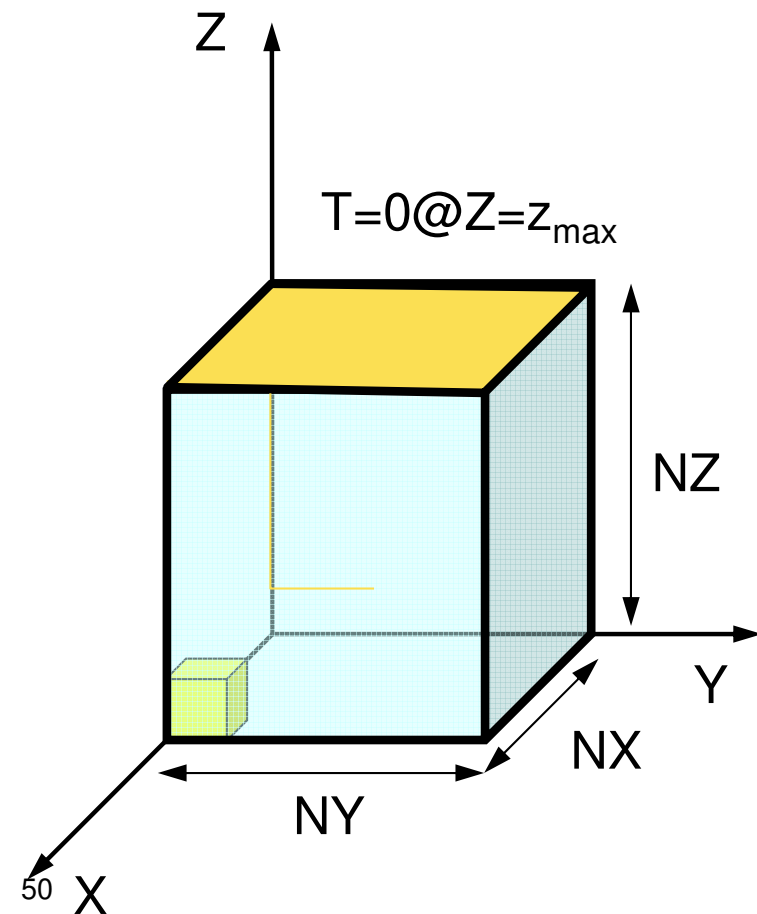
# その前に・・・

- 特に現行領域分割機能はシリアル処理：以下のような場合には時間がかかる
  - 問題規模が大きい
  - 分割数が多い
- そのかわりに並列メッシュ生成プログラムを使う

# Distributed Local Meshes

```

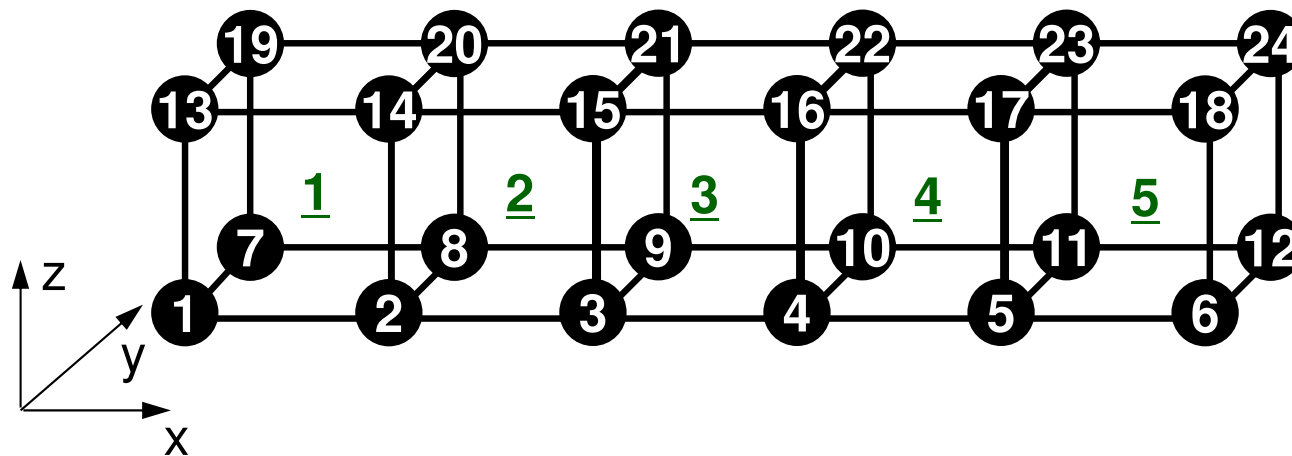
>$ cd /work/gt00/t00XXX/pFEM/pfem3d/pmesh
>$ mpifrtpx -Kfast pmesh.f -o pmesh
>$ <modify "mg.sh", "mesh.inp">
>$ pjsub mg.sh
  
```



# “mesh.inp” : 並列メッシュ生成

(値)	(変数名)	(変数内容)
6 2 2	$np_x, np_y, np_z$	X, Y, Z軸方向の総節点数 前頁の $N_x, N_y, N_z$
2 1 1	$nd_x, nd_y, nd_z$	X, Y, Z軸方向の分割数
pcube	HEADER	分散メッシュファイルのヘッダ名

- $np_x, np_y, np_z$ は $nd_x, nd_y, nd_z$ で割り切れる必要あり
- $nd_x \times nd_y \times nd_z$ が総MPIプロセス数
  - 上記の場合は $6 \times 2 \times 2$ 節点,  $5 \times 1 \times 1$ 要素, X方向2分割



# mg.sh: parallel mesh generation

"proc" must be equal to  $(ndx \times ndy \times ndz)$

Each MPI process generates each local mesh file

## mg.sh

```
#!/bin/sh
#PJM -N "pmg"
#PJM -L rscgrp=tutorial-o
#PJM -L node=1
#PJM --mpi proc=2
#PJM -L elapse=00:10:00
#PJM -g gt00
#PJM -j
#PJM -e err
#PJM -o pmg.lst

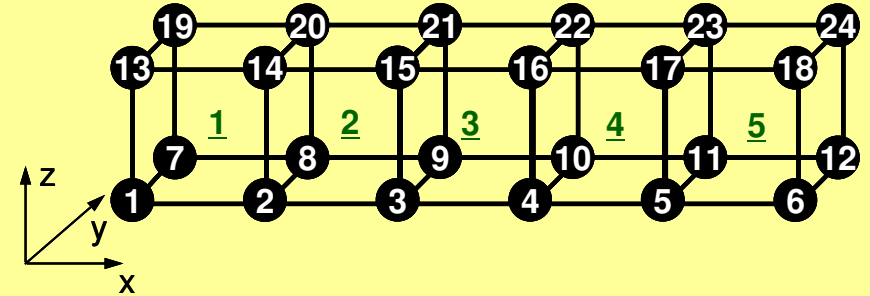
module load fj
module load fjmpi
mpiexec ./pmesh

rm wk.*
```

# 初期全体メッシュ (1CPU) (1/2)

24

1	0.000000E+00	0.000000E+00	0.000000E+00
2	1.000000E+00	0.000000E+00	0.000000E+00
3	2.000000E+00	0.000000E+00	0.000000E+00
4	3.000000E+00	0.000000E+00	0.000000E+00
5	4.000000E+00	0.000000E+00	0.000000E+00
6	5.000000E+00	0.000000E+00	0.000000E+00
7	0.000000E+00	1.000000E+00	0.000000E+00
8	1.000000E+00	1.000000E+00	0.000000E+00
9	2.000000E+00	1.000000E+00	0.000000E+00
10	3.000000E+00	1.000000E+00	0.000000E+00
11	4.000000E+00	1.000000E+00	0.000000E+00
12	5.000000E+00	1.000000E+00	0.000000E+00
13	0.000000E+00	0.000000E+00	1.000000E+00
14	1.000000E+00	0.000000E+00	1.000000E+00
15	2.000000E+00	0.000000E+00	1.000000E+00
16	3.000000E+00	0.000000E+00	1.000000E+00
17	4.000000E+00	0.000000E+00	1.000000E+00
18	5.000000E+00	0.000000E+00	1.000000E+00
19	0.000000E+00	1.000000E+00	1.000000E+00
20	1.000000E+00	1.000000E+00	1.000000E+00
21	2.000000E+00	1.000000E+00	1.000000E+00
22	3.000000E+00	1.000000E+00	1.000000E+00
23	4.000000E+00	1.000000E+00	1.000000E+00
24	5.000000E+00	1.000000E+00	1.000000E+00

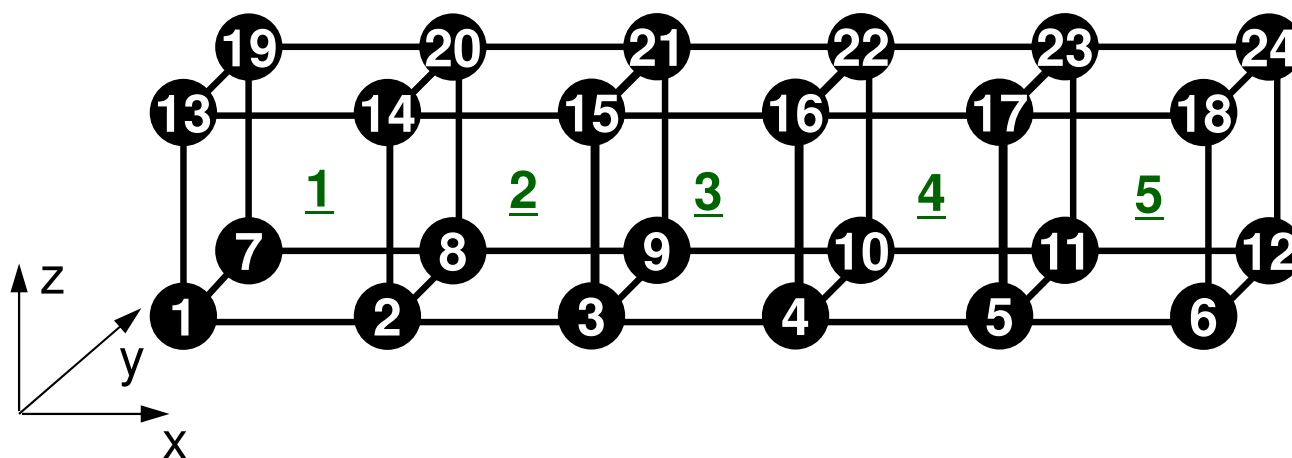


5

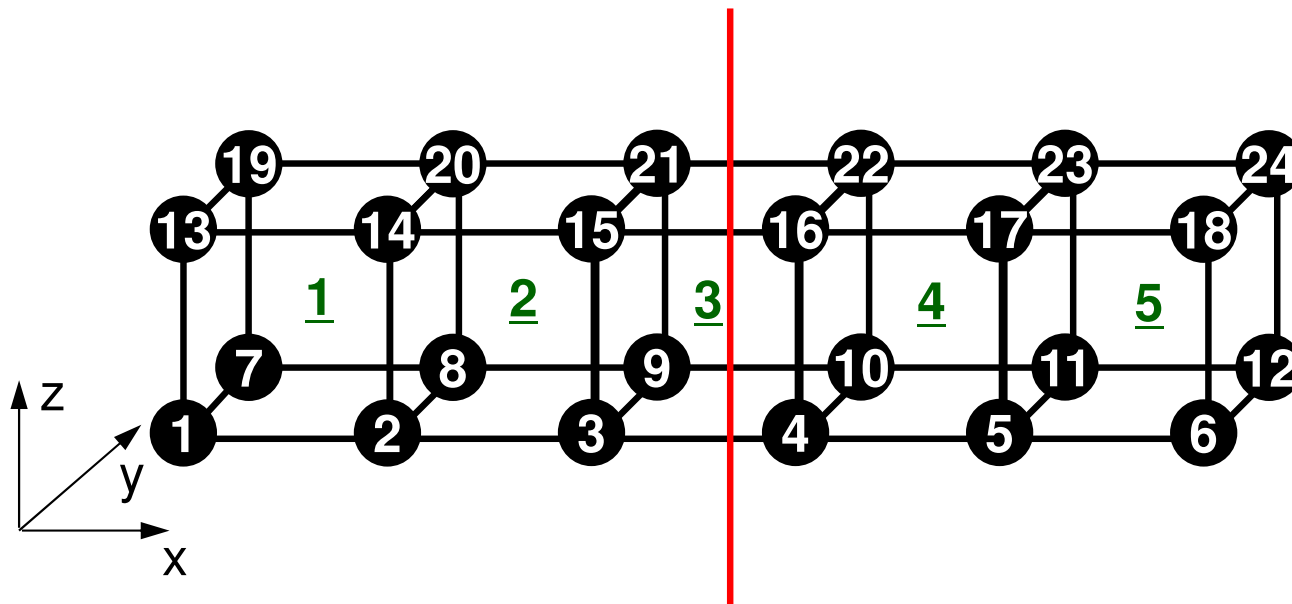
361	361	361	361	361						
1	1	1	2	8	7	13	14	20	19	
2	1	2	3	9	8	14	15	21	20	
3	1	3	4	10	9	15	16	22	21	
4	1	4	5	11	10	16	17	23	22	
5	1	5	6	12	11	17	18	24	23	

# 初期全体メッシュ (1CPU) (2/2)

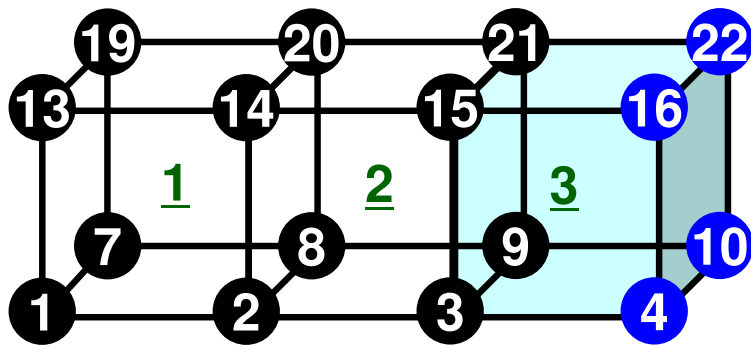
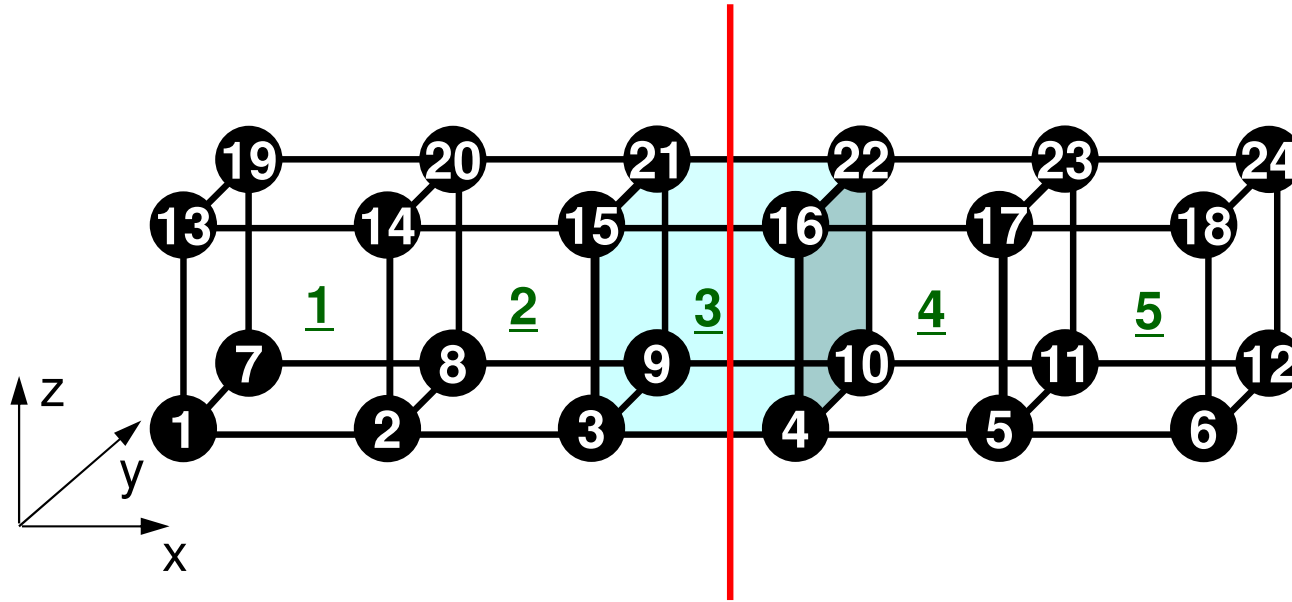
	4											
Xmin	4	16	28	40								
Ymin	1	7	13	19								
Zmin	1	2	3	4	5	6	13	14	15	16		
	17	18										
Zmax	1	2	3	4	5	6	7	8	9	10		
	11	12										
	13	14	15	16	17	18	19	20	21	22		
	23	24										



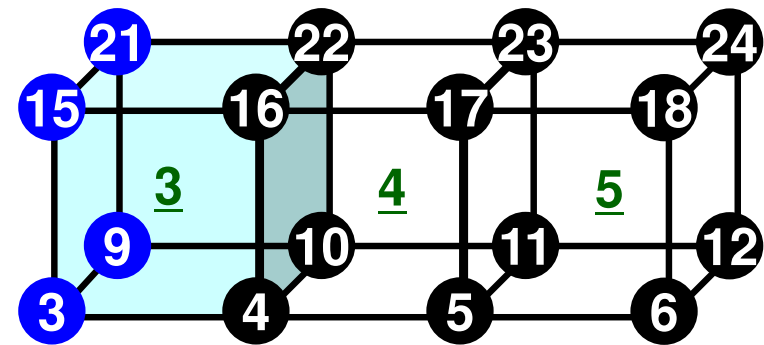
# 領域分割：X軸方向に2分割



# 領域分割：X軸方向に2分割



pcube.0



pcube.1



# 局所分散メッシュデータ

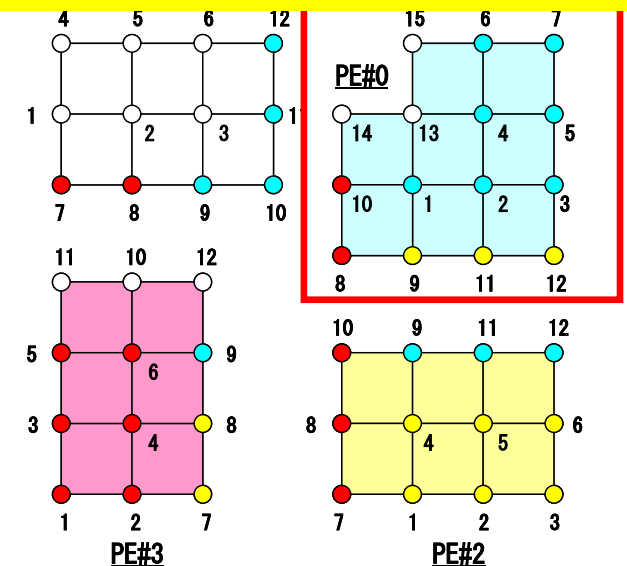
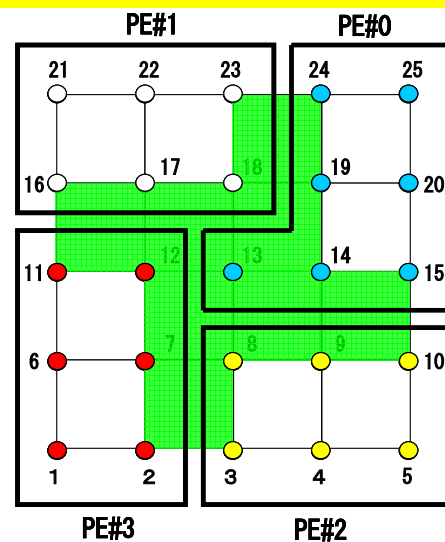
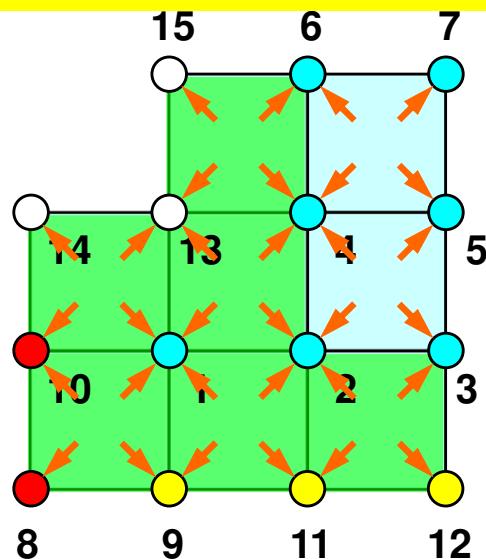
- 隣接領域
- 節点
- 要素
- 受信テーブル
- 送信テーブル
- 節点グループ

# Node-based Partitioning

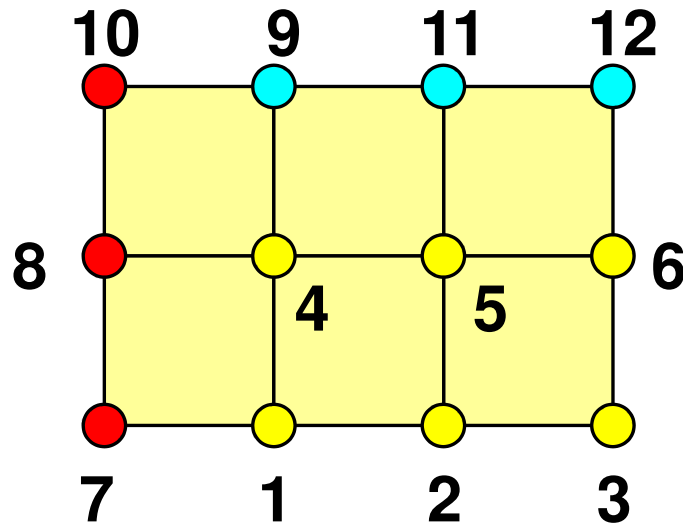
internal nodes - elements - external nodes



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



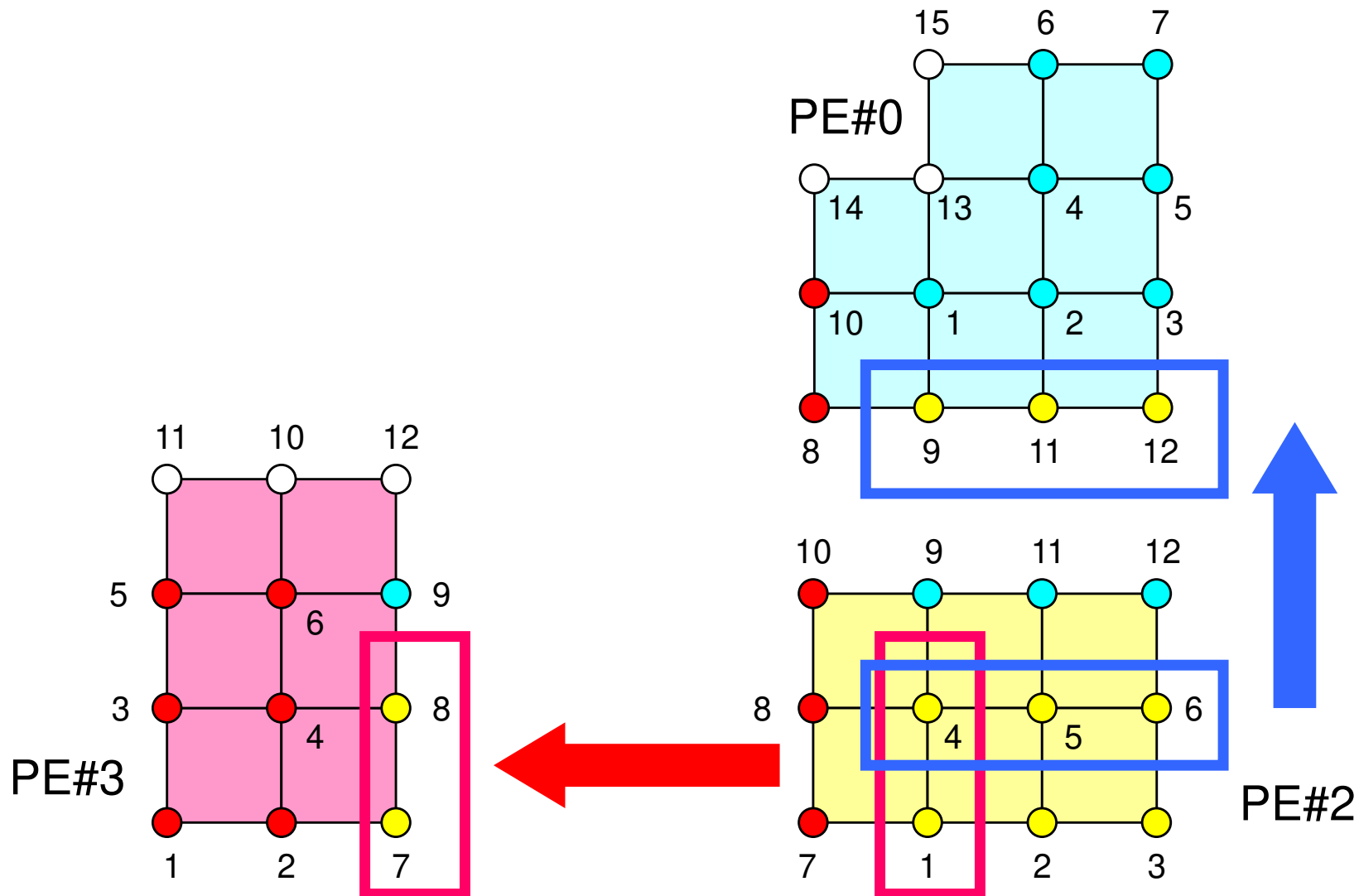
# 各領域データ(局所データ)仕様



- 内点, 外点 (internal/external nodes)
  - 内点~外点となるように局所番号をつける
- 隣接領域情報
  - オーバーラップ要素を共有する領域
  - 隣接領域数, 番号
- 外点情報
  - どの領域から, 何個の, どの外点の情報を「受信:import」するか
- 境界点情報
  - 何個の, どの境界点の情報を, どの領域に「送信:export」するか

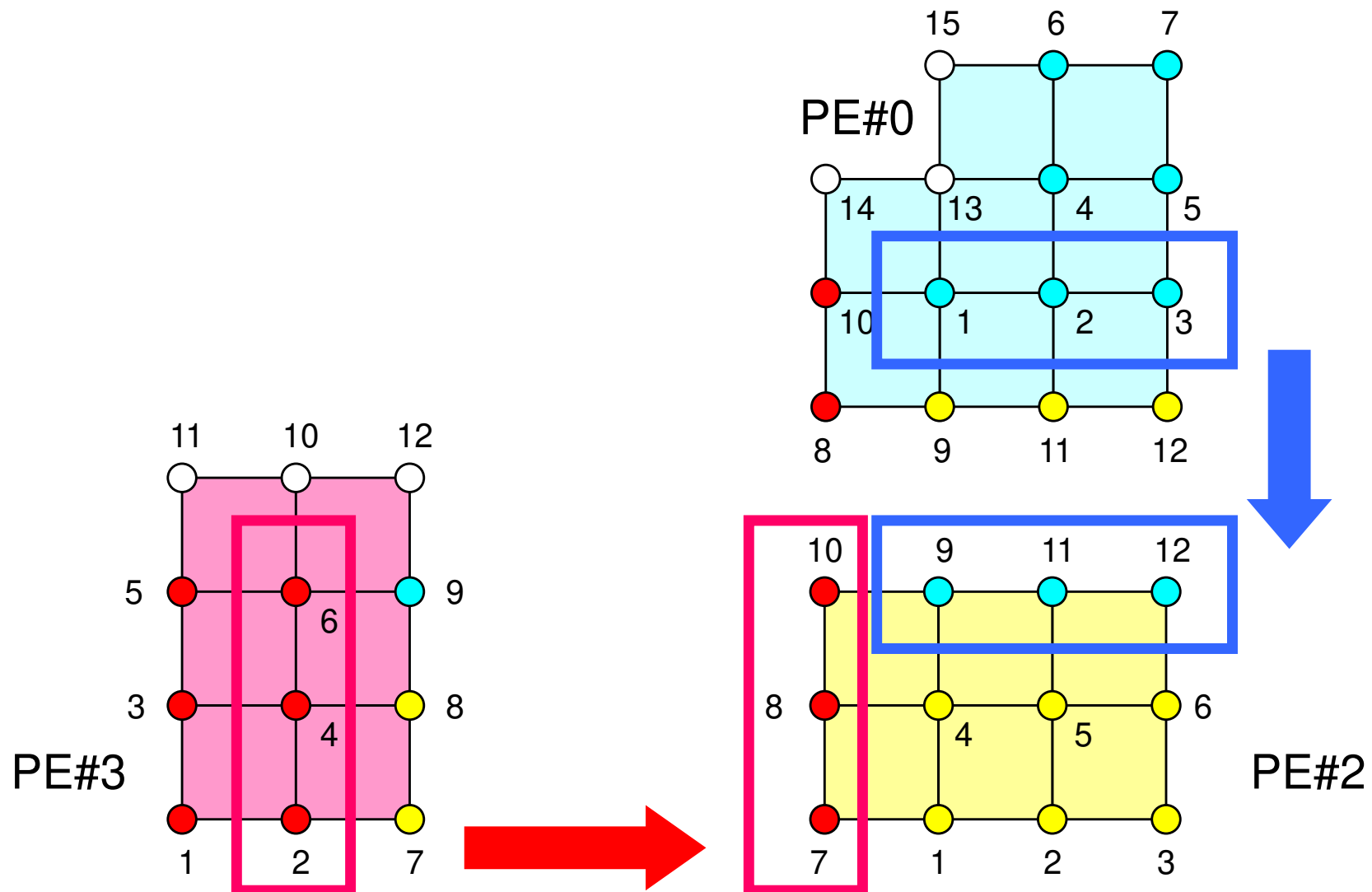
# Boundary Nodes (境界点) : SEND

PE#2 : send information on “boundary nodes”



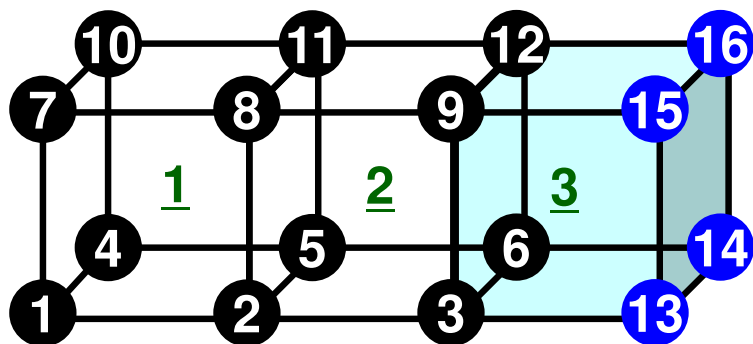
# External Nodes (外点) : RECEIVE

PE#2 : receive information for “external nodes”

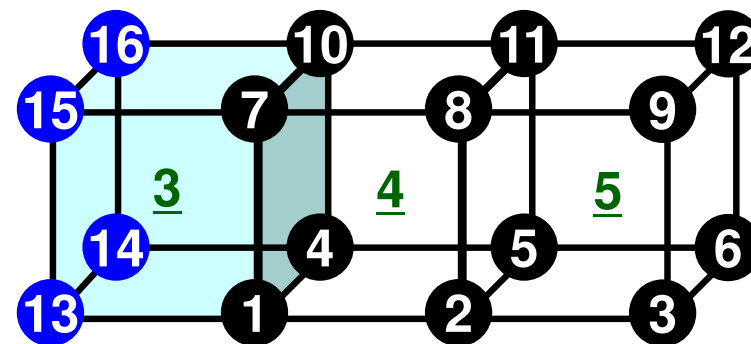


# 隣接領域

pc.0



pc.1



```

0
1
1
16      12
1      0      0.00      0.00      0.00
2      0      1.00      0.00      0.00
3      0      2.00      0.00      0.00
4      0      0.00      1.00      0.00
5      0      1.00      1.00      0.00
6      0      2.00      1.00      0.00
7      0      0.00      0.00      1.00
8      0      1.00      0.00      1.00
9      0      2.00      0.00      1.00
10     0      0.00      1.00      1.00
11     0      1.00      1.00      1.00
12     0      2.00      1.00      1.00
1      1      3.00      0.00      0.00
4      1      3.00      1.00      0.00
7      1      3.00      0.00      1.00
10     1      3.00      1.00      1.00

```

```

1
1
0
16      12
1      1      3.00      0.00      0.00
2      1      4.00      0.00      0.00
3      1      5.00      0.00      0.00
4      1      3.00      1.00      0.00
5      1      4.00      1.00      0.00
6      1      5.00      1.00      0.00
7      1      3.00      0.00      1.00
8      1      4.00      0.00      1.00
9      1      5.00      0.00      1.00
10     1      3.00      1.00      1.00
11     1      4.00      1.00      1.00
12     1      5.00      1.00      1.00
3      0      2.00      0.00      0.00
6      0      2.00      1.00      0.00
9      0      2.00      0.00      1.00
12     0      2.00      1.00      1.00

```

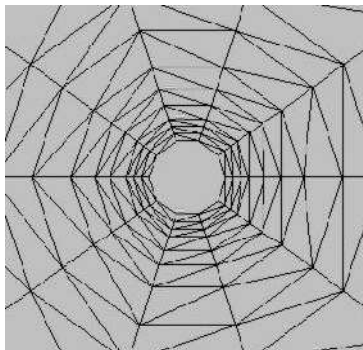
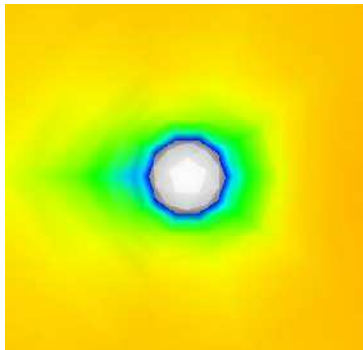
領域ID  
隣接領域数 NEIBPETOT  
隣接領域ID NEIBPE (neib)

# 局所番号付け: 節点

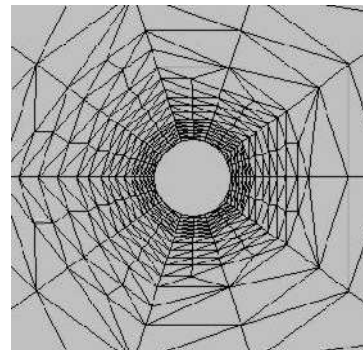
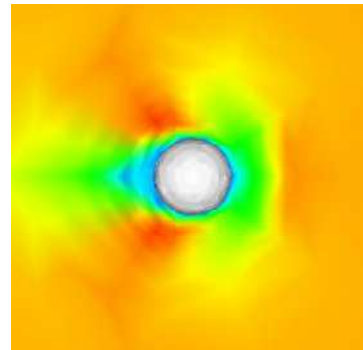
- 局所番号は各領域「1」から番号付け
  - 1CPUの場合と同じプログラムを使用可能: SPMD
  - 要素番号も同じように「1」から番号付け
- 内点⇒外点という順番で番号付け
- Double Numbering
  - 本来の所属領域での局所節点番号: **NODE\_ID (i, 1)**
  - 所属領域番号: **NODE\_ID (i, 2)**

# Supersonic Flow around a Sphere

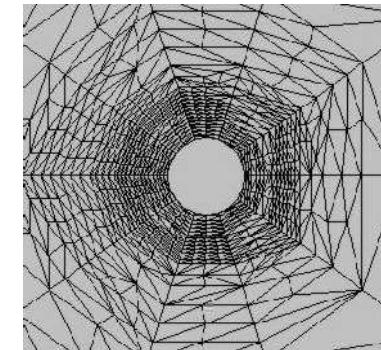
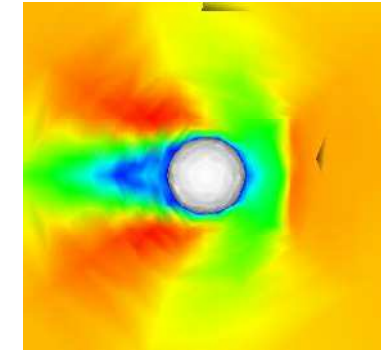
Ideal Gas,  $M=1.40$ , Uniform Flow,  $Re=10^6$   
before/after Dynamic Load Balancing



**Initial Grid**



**1-Lev. Adapted**



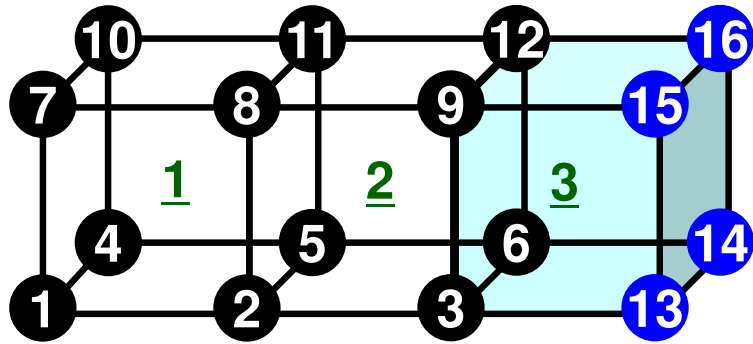
**2-Lev. Adapted**

			<u>before</u>	<u>after</u>	<u>before</u>	<u>after</u>
PE0	137	-	793	652	3834	2527
PE1	137	-	696	650	2769	2526
PE2	136	-	668	652	2703	2522
PE3	136	-	448	651	1390	2524

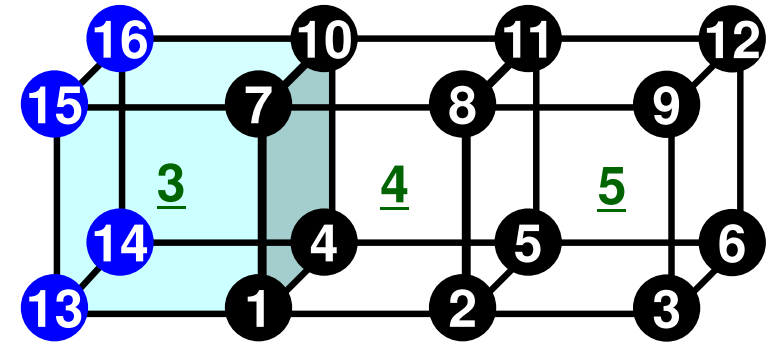


# 内点, 外点

pc.0



pc.1

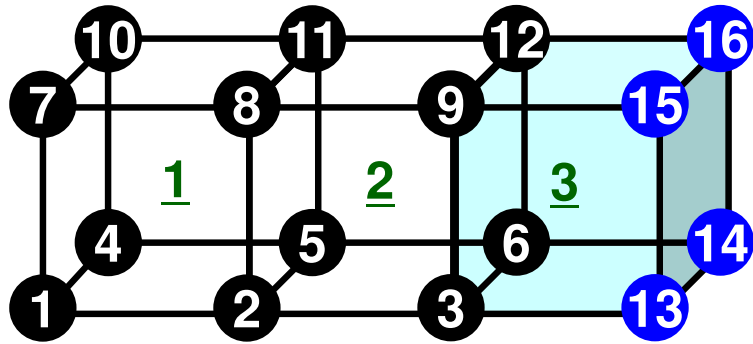


0				
1				
1				
16		12		
1	0	0.00	0.00	0.00
2	0	1.00	0.00	0.00
3	0	2.00	0.00	0.00
4	0	0.00	1.00	0.00
5	0	1.00	1.00	0.00
6	0	2.00	1.00	0.00
7	0	0.00	0.00	1.00
8	0	1.00	0.00	1.00
9	0	2.00	0.00	1.00
10	0	0.00	1.00	1.00
11	0	1.00	1.00	1.00
12	0	2.00	1.00	1.00
1	1	3.00	0.00	0.00
4	1	3.00	1.00	0.00
7	1	3.00	0.00	1.00
10	1	3.00	1.00	1.00

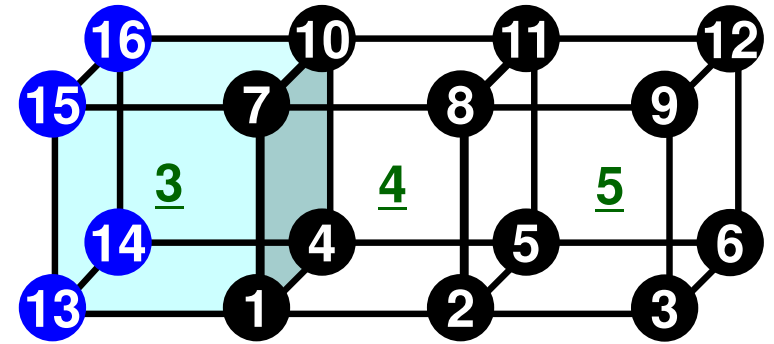
1				
1				
0				
16		12	(総節点数, 内点数)	
1	1	3.00	0.00	0.00
2	1	4.00	0.00	0.00
3	1	5.00	0.00	0.00
4	1	3.00	1.00	0.00
5	1	4.00	1.00	0.00
6	1	5.00	1.00	0.00
7	1	3.00	0.00	1.00
8	1	4.00	0.00	1.00
9	1	5.00	0.00	1.00
10	1	3.00	1.00	1.00
11	1	4.00	1.00	1.00
12	1	5.00	1.00	1.00
3	0	2.00	0.00	0.00
6	0	2.00	1.00	0.00
9	0	2.00	0.00	1.00
12	0	2.00	1.00	1.00

# 局所番号付け: 節点

pc.0



pc.1



0					
1					
1					
16	12				
1	0	0.00	0.00	0.00	①
2	0	1.00	0.00	0.00	②
3	0	2.00	0.00	0.00	③
4	0	0.00	1.00	0.00	④
5	0	1.00	1.00	0.00	⑤
6	0	2.00	1.00	0.00	⑥
7	0	0.00	0.00	1.00	⑦
8	0	1.00	0.00	1.00	⑧
9	0	2.00	0.00	1.00	⑨
10	0	0.00	1.00	1.00	⑩
11	0	1.00	1.00	1.00	⑪
12	0	2.00	1.00	1.00	⑫
1	1	3.00	0.00	0.00	⑬
4	1	3.00	1.00	0.00	⑭
7	1	3.00	0.00	1.00	⑮
10	1	3.00	1.00	1.00	⑯

所属領域とそこでの番号

座標

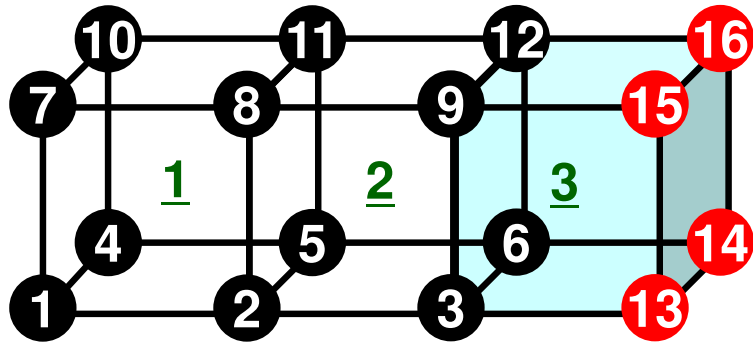
1					
1					
0					
16	12				
1	1	3.00	0.00	0.00	①
2	1	4.00	0.00	0.00	②
3	1	5.00	0.00	0.00	③
4	1	3.00	1.00	0.00	④
5	1	4.00	1.00	0.00	⑤
6	1	5.00	1.00	0.00	⑥
7	1	3.00	0.00	1.00	⑦
8	1	4.00	0.00	1.00	⑧
9	1	5.00	0.00	1.00	⑨
10	1	3.00	1.00	1.00	⑩
11	1	4.00	1.00	1.00	⑪
12	1	5.00	1.00	1.00	⑫
3	0	2.00	0.00	0.00	⑬
6	0	2.00	1.00	0.00	⑭
9	0	2.00	0.00	1.00	⑮
12	0	2.00	1.00	1.00	⑯

所属領域とそこでの番号

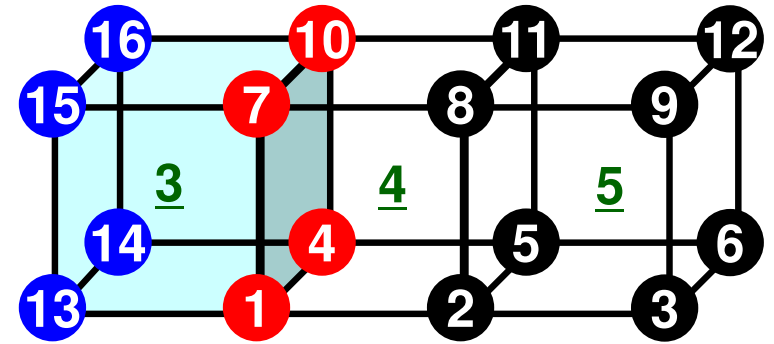
座標

# 局所番号付け: 節点

pc.0



pc.1



0					
1					
1					
1					
16	12				
1	0	0.00	0.00	0.00	①
2	0	1.00	0.00	0.00	②
3	0	2.00	0.00	0.00	③
4	0	0.00	1.00	0.00	④
5	0	1.00	1.00	0.00	⑤
6	0	2.00	1.00	0.00	⑥
7	0	0.00	0.00	1.00	⑦
8	0	1.00	0.00	1.00	⑧
9	0	2.00	0.00	1.00	⑨
10	0	0.00	1.00	1.00	⑩
11	0	1.00	1.00	1.00	⑪
12	0	2.00	1.00	1.00	⑫
1	1	3.00	0.00	0.00	⑬
4	1	3.00	1.00	0.00	⑭
7	1	3.00	0.00	1.00	⑮
10	1	3.00	1.00	1.00	⑯

所属領域とそこでの番号

座標

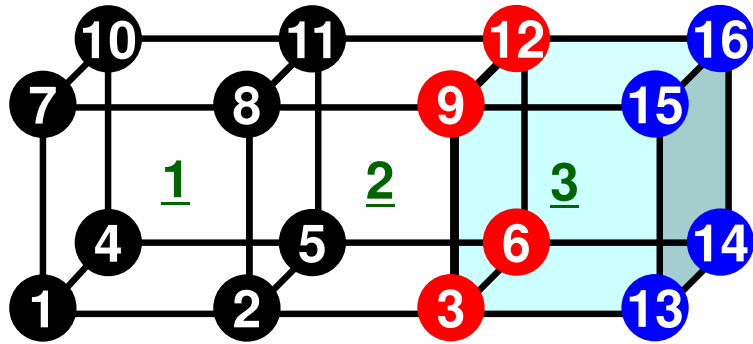
1					
1					
0					
16	12				
1	1	3.00	0.00	0.00	①
2	1	4.00	0.00	0.00	②
3	1	5.00	0.00	0.00	③
4	1	3.00	1.00	0.00	④
5	1	4.00	1.00	0.00	⑤
6	1	5.00	1.00	0.00	⑥
7	1	3.00	0.00	1.00	⑦
8	1	4.00	0.00	1.00	⑧
9	1	5.00	0.00	1.00	⑨
10	1	3.00	1.00	1.00	⑩
11	1	4.00	1.00	1.00	⑪
12	1	5.00	1.00	1.00	⑫
3	0	2.00	0.00	0.00	⑬
6	0	2.00	1.00	0.00	⑭
9	0	2.00	0.00	1.00	⑮
12	0	2.00	1.00	1.00	⑯

所属領域とそこでの番号

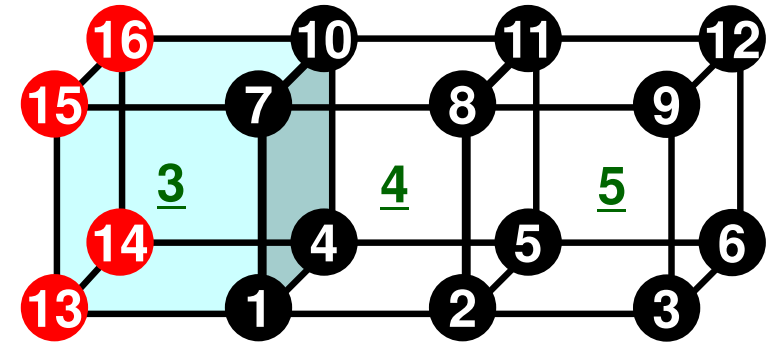
座標

# 局所番号付け: 節点

pc.0



pc.1



0					
1					
1					
16	12				
1	0	0.00	0.00	0.00	①
2	0	1.00	0.00	0.00	②
3	0	2.00	0.00	0.00	③
4	0	0.00	1.00	0.00	④
5	0	1.00	1.00	0.00	⑤
6	0	2.00	1.00	0.00	⑥
7	0	0.00	0.00	1.00	⑦
8	0	1.00	0.00	1.00	⑧
9	0	2.00	0.00	1.00	⑨
10	0	0.00	1.00	1.00	⑩
11	0	1.00	1.00	1.00	⑪
12	0	2.00	1.00	1.00	⑫
1	1	3.00	0.00	0.00	⑬
4	1	3.00	1.00	0.00	⑭
7	1	3.00	0.00	1.00	⑮
10	1	3.00	1.00	1.00	⑯

所属領域とそこでの番号

座標

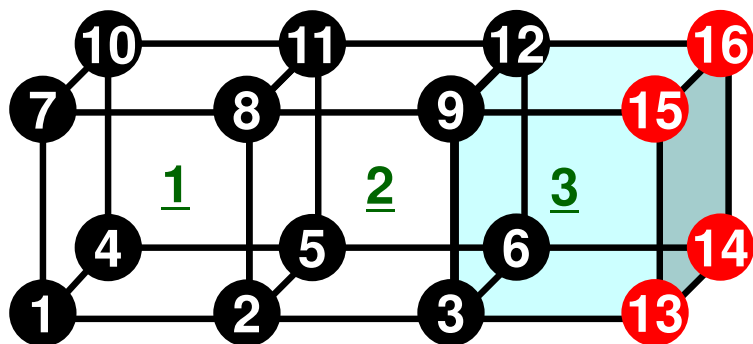
1					
1					
0					
16	12				
1	1	3.00	0.00	0.00	①
2	1	4.00	0.00	0.00	②
3	1	5.00	0.00	0.00	③
4	1	3.00	1.00	0.00	④
5	1	4.00	1.00	0.00	⑤
6	1	5.00	1.00	0.00	⑥
7	1	3.00	0.00	1.00	⑦
8	1	4.00	0.00	1.00	⑧
9	1	5.00	0.00	1.00	⑨
10	1	3.00	1.00	1.00	⑩
11	1	4.00	1.00	1.00	⑪
12	1	5.00	1.00	1.00	⑫
3	0	2.00	0.00	0.00	⑬
6	0	2.00	1.00	0.00	⑭
9	0	2.00	0.00	1.00	⑮
12	0	2.00	1.00	1.00	⑯

所属領域とそこでの番号

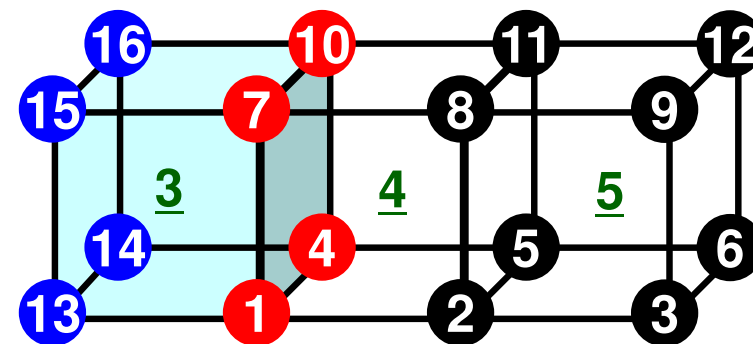
座標

# 局所番号付け: 節点

pc.0



pc.1



0					
1					
1					
16	12				
1	0	0.00	0.00	0.00	①
2	0	1.00	0.00	0.00	②
3	0	2.00	0.00	0.00	③
4	0	0.00	1.00	0.00	④
5	0	1.00	1.00	0.00	⑤
6	0	2.00	1.00	0.00	⑥
7	0	0.00	0.00	1.00	⑦
8	0	1.00	0.00	1.00	⑧
9	0	2.00	0.00	1.00	⑨
10	0	0.00	1.00	1.00	⑩
11	0	1.00	1.00	1.00	⑪
12	0	2.00	1.00	1.00	⑫
1	1	3.00	0.00	0.00	⑬
4	1	3.00	1.00	0.00	⑭
7	1	3.00	0.00	1.00	⑮
10	1	3.00	1.00	1.00	⑯

所属領域とそこでの番号

座標

1					
1					
0					
16	12				
1	1	3.00	0.00	0.00	①
2	1	4.00	0.00	0.00	②
3	1	5.00	0.00	0.00	③
4	1	3.00	1.00	0.00	④
5	1	4.00	1.00	0.00	⑤
6	1	5.00	1.00	0.00	⑥
7	1	3.00	0.00	1.00	⑦
8	1	4.00	0.00	1.00	⑧
9	1	5.00	0.00	1.00	⑨
10	1	3.00	1.00	1.00	⑩
11	1	4.00	1.00	1.00	⑪
12	1	5.00	1.00	1.00	⑫
3	0	2.00	0.00	0.00	⑬
6	0	2.00	1.00	0.00	⑭
9	0	2.00	0.00	1.00	⑮
12	0	2.00	1.00	1.00	⑯

所属領域とそこでの番号

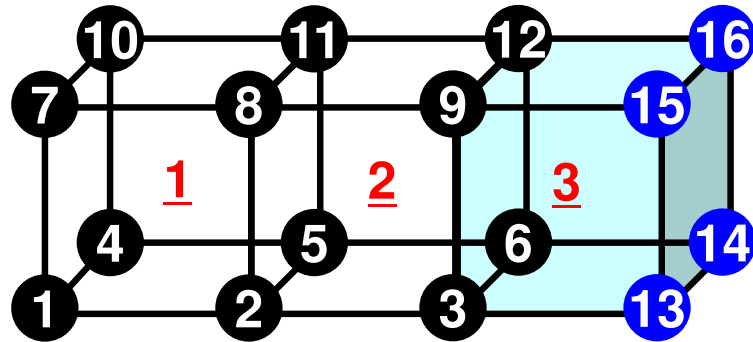
座標

以降のデータ, プログラム内部で使うのは丸付き数字(局所節点番号)

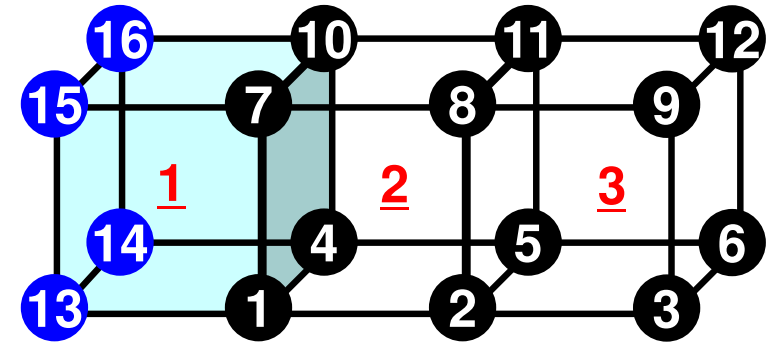


# 局所番号付け:要素

pc.0

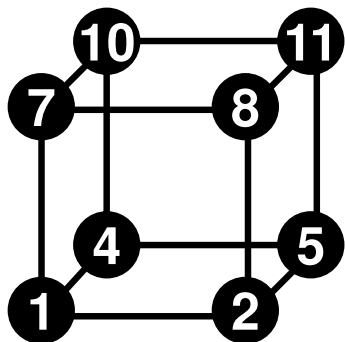


pc.1



3		3		361		361		361		
1	0	1	1	2	5	4	7	8	11	10
2	0	1	2	3	6	5	8	9	12	11
3	0	1	3	13	14	6	9	15	16	12
1	2	3								

3		2		(全要素, 領域所属要素)		361		361		361	
3	0	1	13	1	4	14	15	7	10	16	
1	1	1	1	2	5	4	7	8	11	10	
2	1	1	2	3	6	5	8	9	12	11	
2	3										



## • 要素が所属する領域

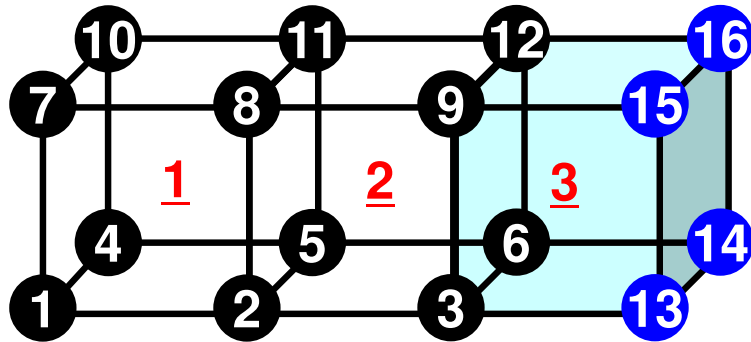
- 8個の節点の所属する領域によって決定
- 全て「内点」であれば, 節点と同じ領域
- 「外点」を含む場合は, 節点の所属領域番号の最も若い領域に属する
- 本ケースのオーバーラップ要素は「0」領域に所属



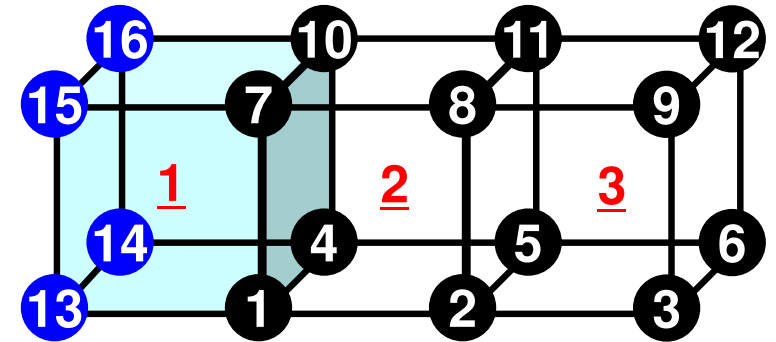


# 局所番号付け:要素

pc.0



pc.1



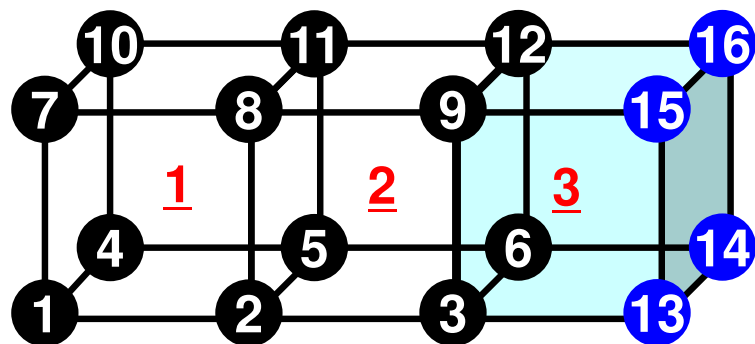
<u>3</u>	3														
361	361	361													
1	0	1	1	2	5	4	7	8	11	10	<u>1</u>				
2	0	1	2	3	6	5	8	9	12	11	<u>2</u>				
3	0	1	3	13	14	6	9	15	16	12	<u>3</u>				
1	2	3													

<u>3</u>	2														
361	361	361													
3	0	1	13	1	4	14	15	7	10	16	<u>1</u>				
1	1	1	1	2	5	4	7	8	11	10	<u>2</u>				
2	1	1	2	3	6	5	8	9	12	11	<u>3</u>				
2	3														

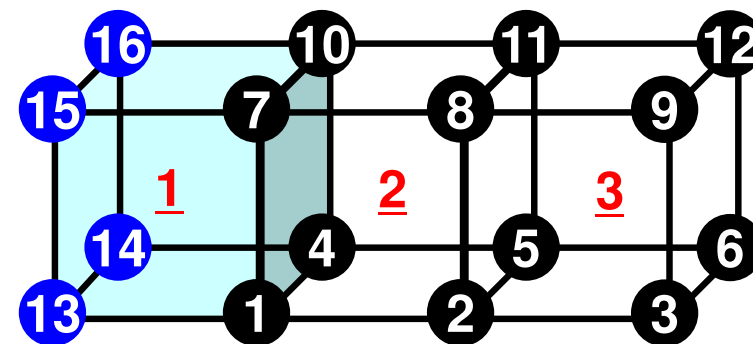
- 要素についてもDouble Numbering
  - 本来の所属領域での局所要素番号  $ELEM\_ID(i, 1)$
  - 所属領域番号  $ELEM\_ID(i, 2)$
- 材料番号
- 8個の節点
- 以降の計算では下線付の「局所要素番号」を使用

# 局所番号付け:要素

pc.0



pc.1



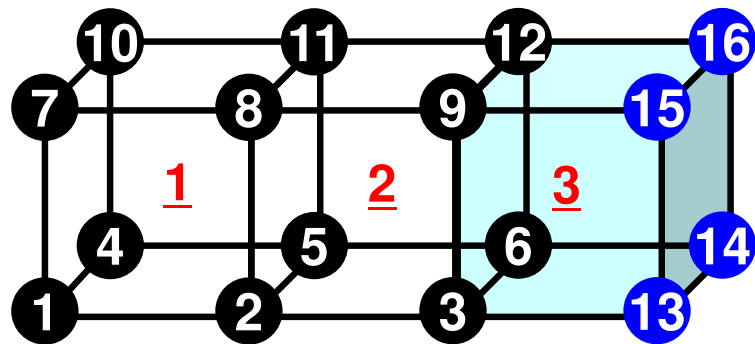
3	<u>3</u>														
361	361	361													
<u>1</u>	0	1	1	2	5	4	7	8	11	10	<u>1</u>				
2	0	1	2	3	6	5	8	9	12	11	<u>2</u>				
3	0	1	3	13	14	6	9	15	16	12	<u>3</u>				
<u>1</u>	<u>2</u>	<u>3</u>													

3	<u>2</u>														
361	361	361													
3	0	1	13	1	4	14	15	7	10	16	<u>1</u>				
1	1	1	1	2	5	4	7	8	11	10	<u>2</u>				
2	1	1	2	3	6	5	8	9	12	11	<u>3</u>				
<u>2</u>	<u>3</u>														

- pc.0
  - 1, 2, 3 の要素が「領域所属要素」
- pc.1
  - 2, 3 の要素が「領域所属要素」

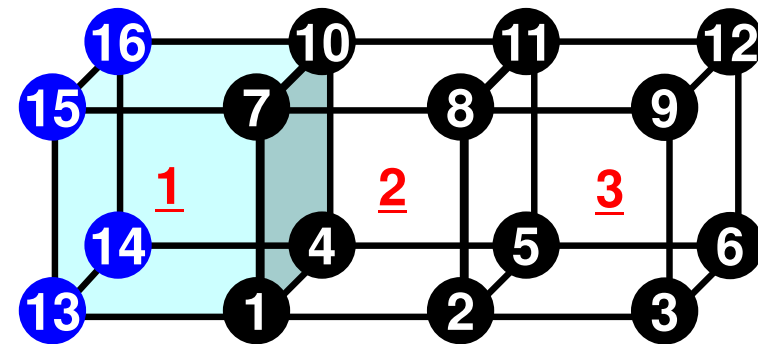
# 通信テーブル: 受信, 送信

pc.0



4  
13  
14  
15  
16  
4  
3  
6  
9  
12

pc.1



4  
13  
14  
15  
16  
4  
1  
4  
7  
10

# 領域間通信

## 一般化された通信テーブル

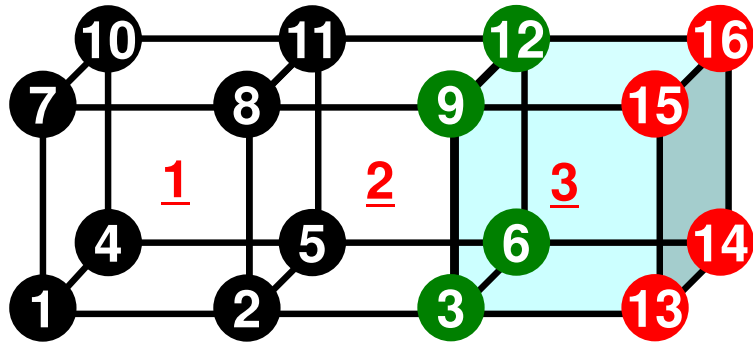
- 「通信」とは「外点」の情報を, その「外点」が本来属している領域から得ることである。
- 「通信テーブル」とは領域間の外点の関係の情報を記述したもの。
  - 「送信テーブル (export)」、 「受信テーブル (import)」がある。
- 送信側: 「境界点」として送る
- 受信側: 「外点」として受け取る

# 一般化された通信テーブル: 送信 (F)

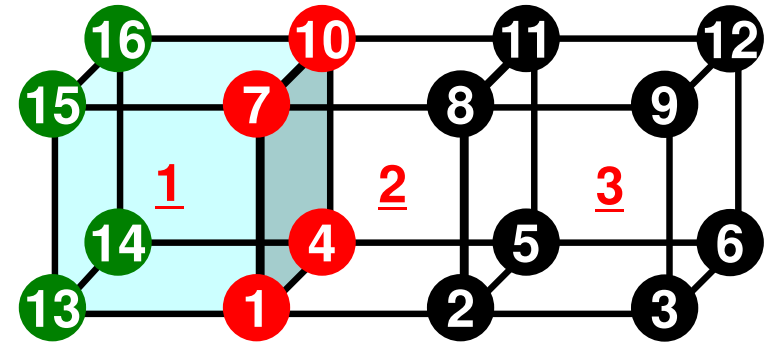
- 送信相手
  - NEIBPETOT, NEIBPE(neib)
- それぞれの送信相手に送るメッセージサイズ
  - export\_index(neib), neib= 0, NEIBPETOT
- 「境界点」番号
  - export\_item(k), k= 1, export\_index(NEIBPETOT)
- それぞれの送信相手に送るメッセージ
  - SENDbuf(k), k= 1, export\_index(NEIBPETOT)

# 通信テーブル(送信)

pc.0



pc.1



```

4
13
14
15
16
4
3
6
9
12

```

```

4
13
14
15
16
4
1
4
7
10

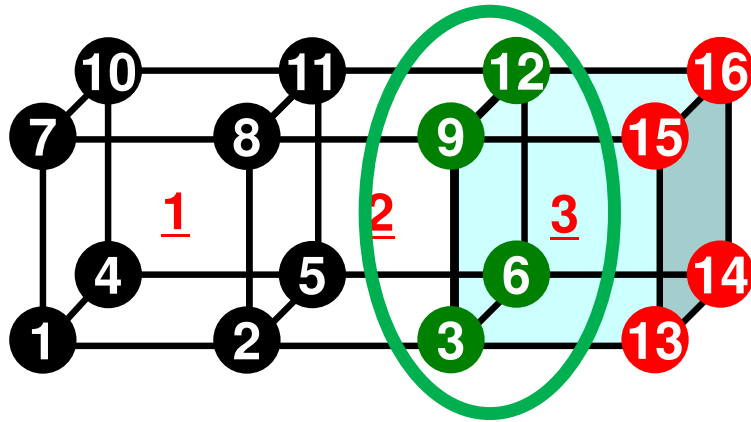
```

`export_index(neib)` : 送信節点数  
`export_item` : 節点番号

- `export_index` 各隣接領域に送信する外点の数(累積数)
  - 現在:隣接領域数は1
- `export_item` 境界点の番号

# 通信テーブル(送信)

pc.0

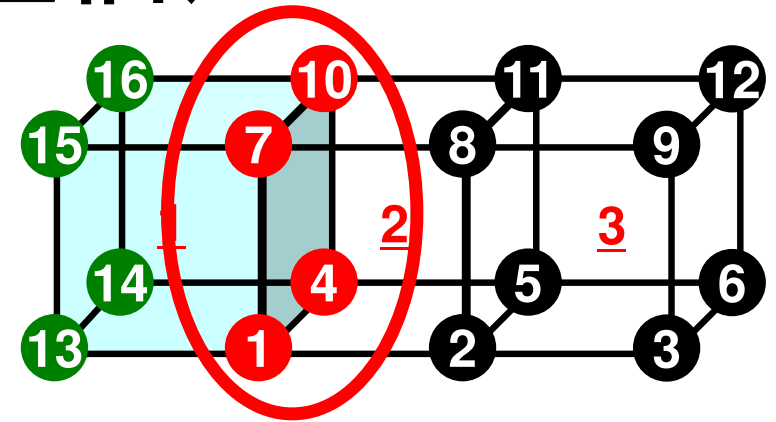


```

4
13
14
15
16
4
3
6
9
12

```

pc.1



```

4
13
14
15
16
4
1
4
7
10

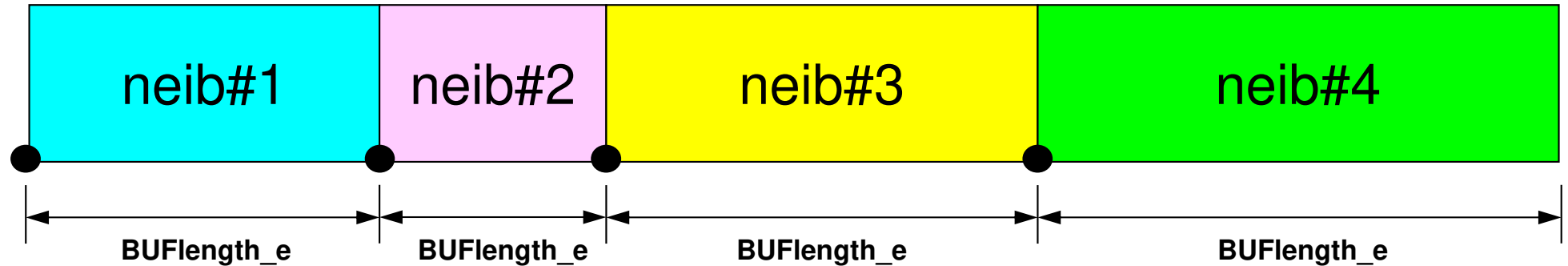
```

export\_index(neib) : 送信節点数  
export\_item : 節点番号

- `export_index` 各隣接領域に送信する外点の数(累積数)
  - 現在:隣接領域数は1
- `export_item` 境界点の番号

# 送信 (MPI\_Isend/Irecv/Waitall) (F)

## 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
```

```
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)
```

送信バッファへの代入  
温度などの変数を直接送信, 受信に使うのではなく, このようなバッファへ一回代入して計算することを勧める。

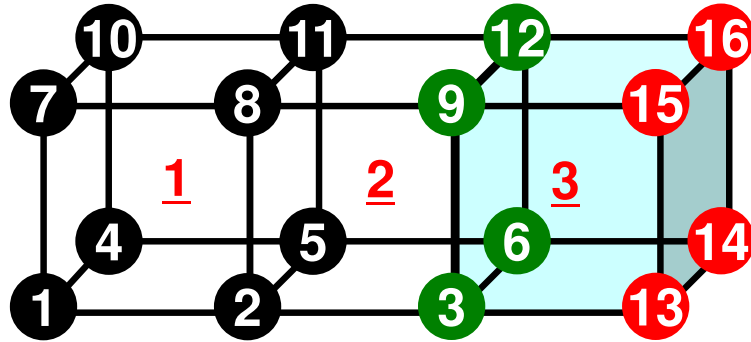


# 一般化された通信テーブル: 受信 (F)

- 受信相手
  - NEIBPETOT, NEIBPE(neib)
- それぞれの受信相手から受け取るメッセージサイズ
  - import\_index(neib), neib= 0, NEIBPETOT
- 「外点」番号
  - import\_item(k), k= 1, import\_index(NEIBPETOT)
- それぞれの受信相手から受け取るメッセージ
  - RECVbuf(k), k= 1, import\_index(NEIBPETOT)

# 通信テーブル(受信)

pc.0

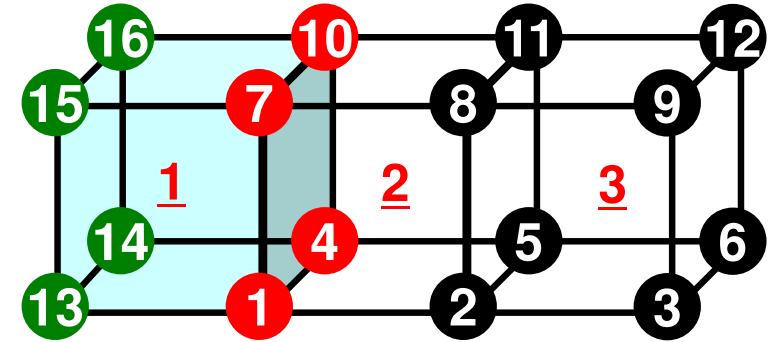


```

4
13
14
15
16
4
3
6
9
12

```

pc.1



```

4
13
14
15
16
4
1
4
7
10

```

```

import_index(neib) 受信節点数
import_item  節点番号

```

```

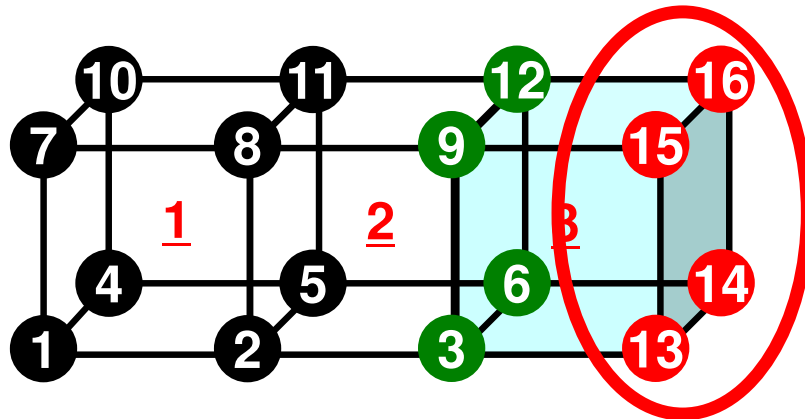
export_index(neib)
export_item

```

- import\_index 各隣接領域から受信する外点の数(累積数)
  - 現在:隣接領域数は1
- import\_item 外点の番号, 所属領域

# 通信テーブル(受信)

pc.0

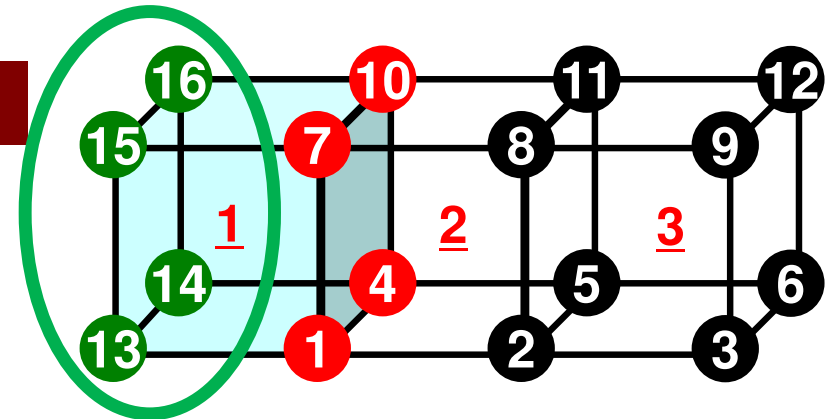


```

4
13
14
15
16
4
3
6
9
12

```

pc.1



```

4
13
14
15
16
4
1
4
7
10

```

```

import_index(neib) 受信節点数
import_item  節点番号

```

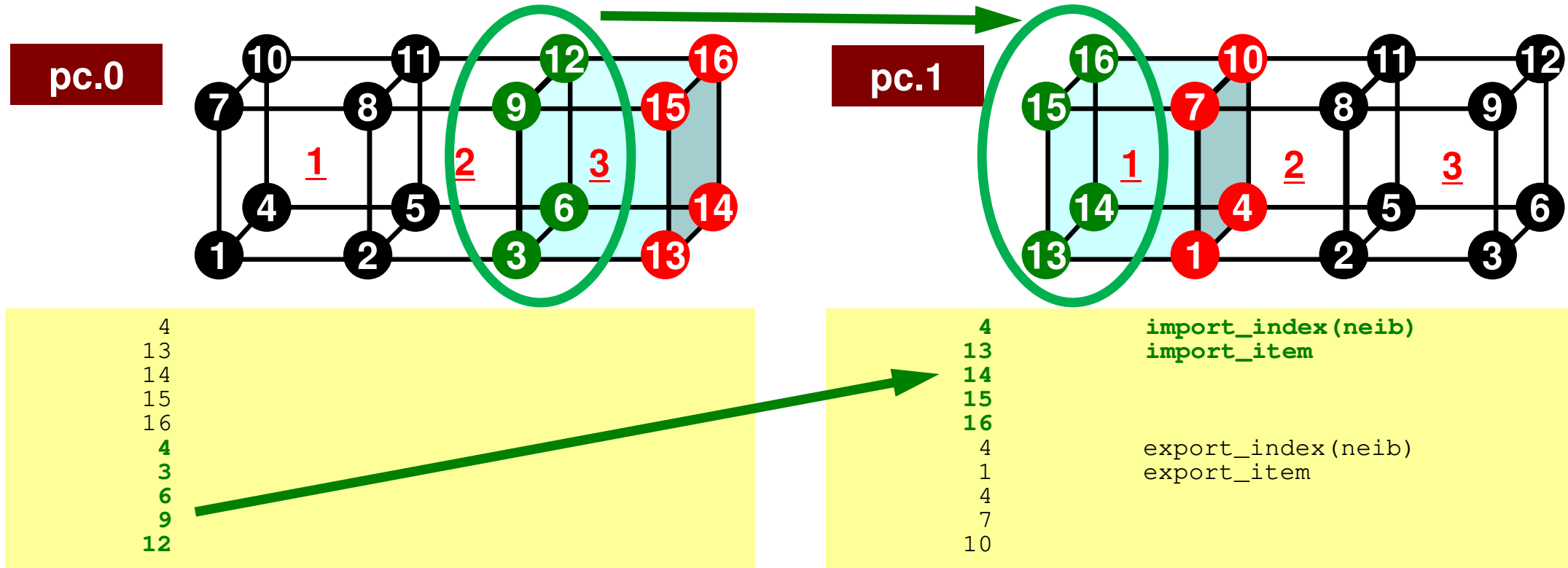
```

export_index(neib)
export_item

```

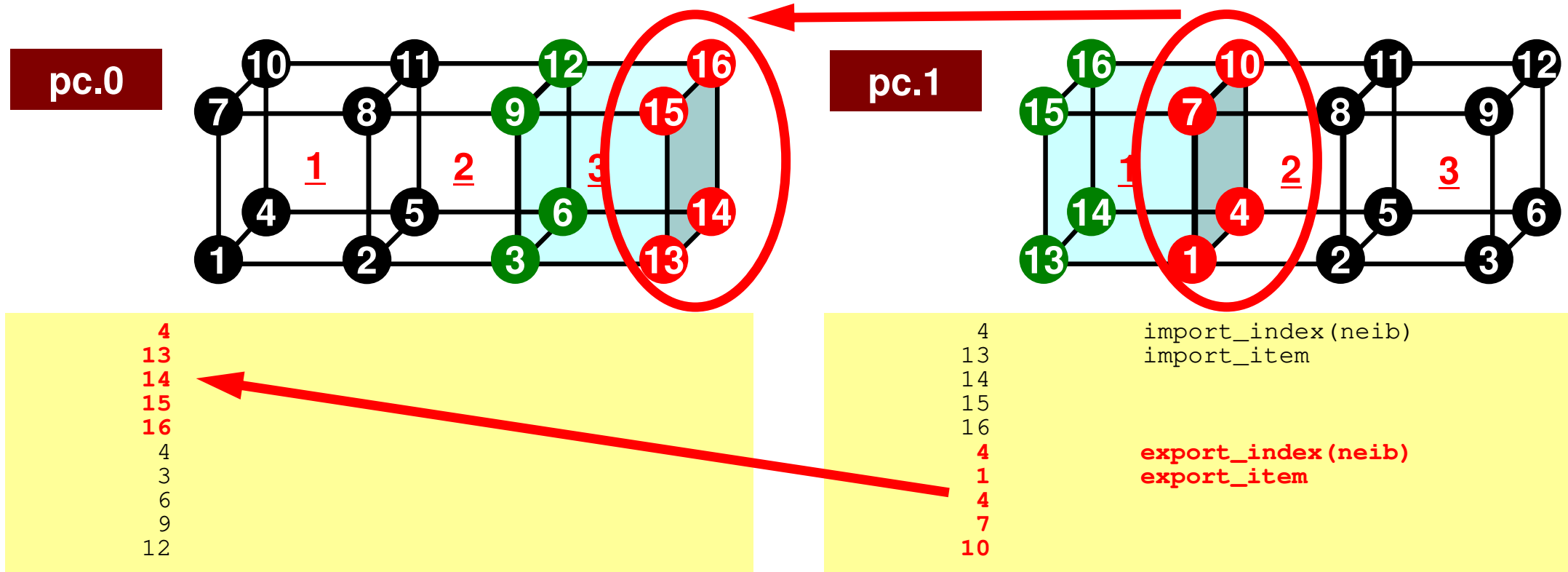
- import\_index 各隣接領域から受信する外点の数(累積数)
  - 現在:隣接領域数は1
- import\_item 外点の番号, 所属領域

# 通信テーブル(受信)



- `import_index` 各隣接領域から受信する外点の数(累積数)  
– 現在:隣接領域数は1
- `import_item` 外点の番号, 所属領域

# 通信テーブル(受信)



- import\_index 各隣接領域から受信する外点の数(累積数)
  - 現在:隣接領域数は1
- import\_item 外点の番号, 所属領域

# 受信 (MPI\_Isend/Irecv/Waitall) (F)

```

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

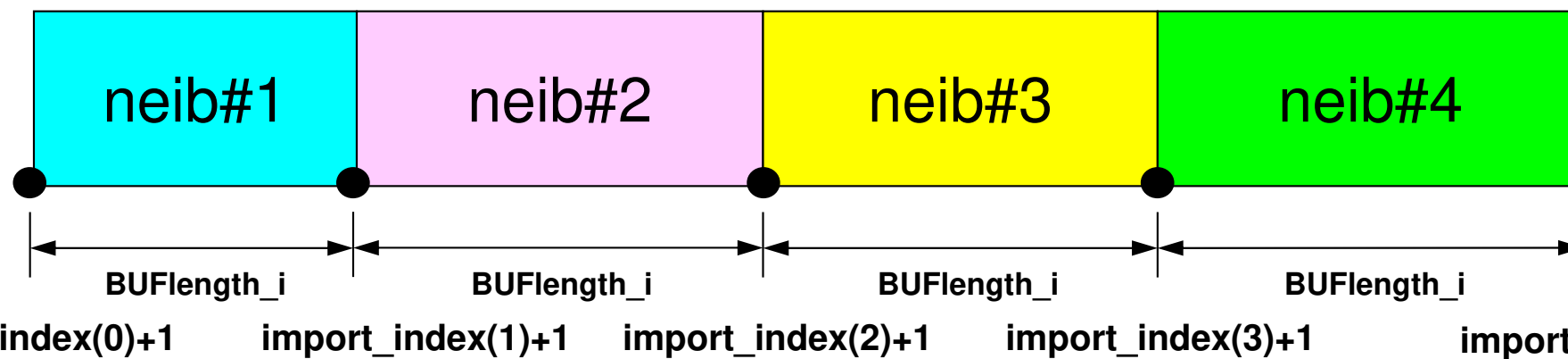
call MPI_WAITALL (NEIBPETOT, request_recv, stat_recv, ierr)

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

```

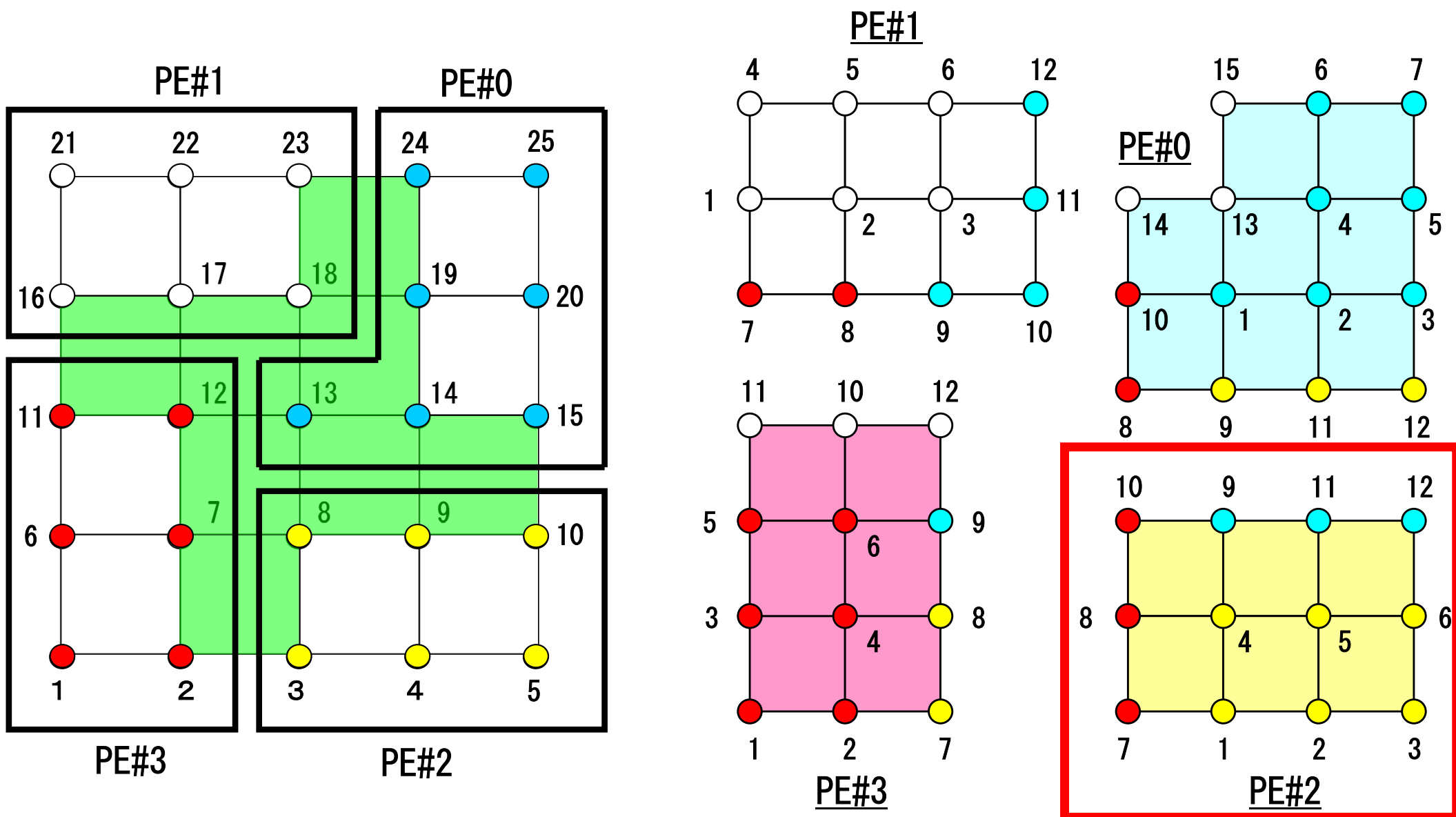
受信バッファから代入

## RECVbuf

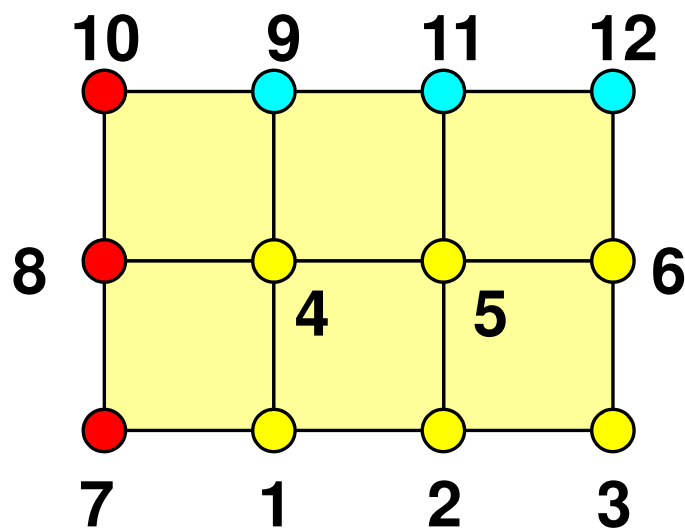


# Node-based Partitioning

internal nodes - elements - external nodes



# 各領域データ(局所データ)仕様

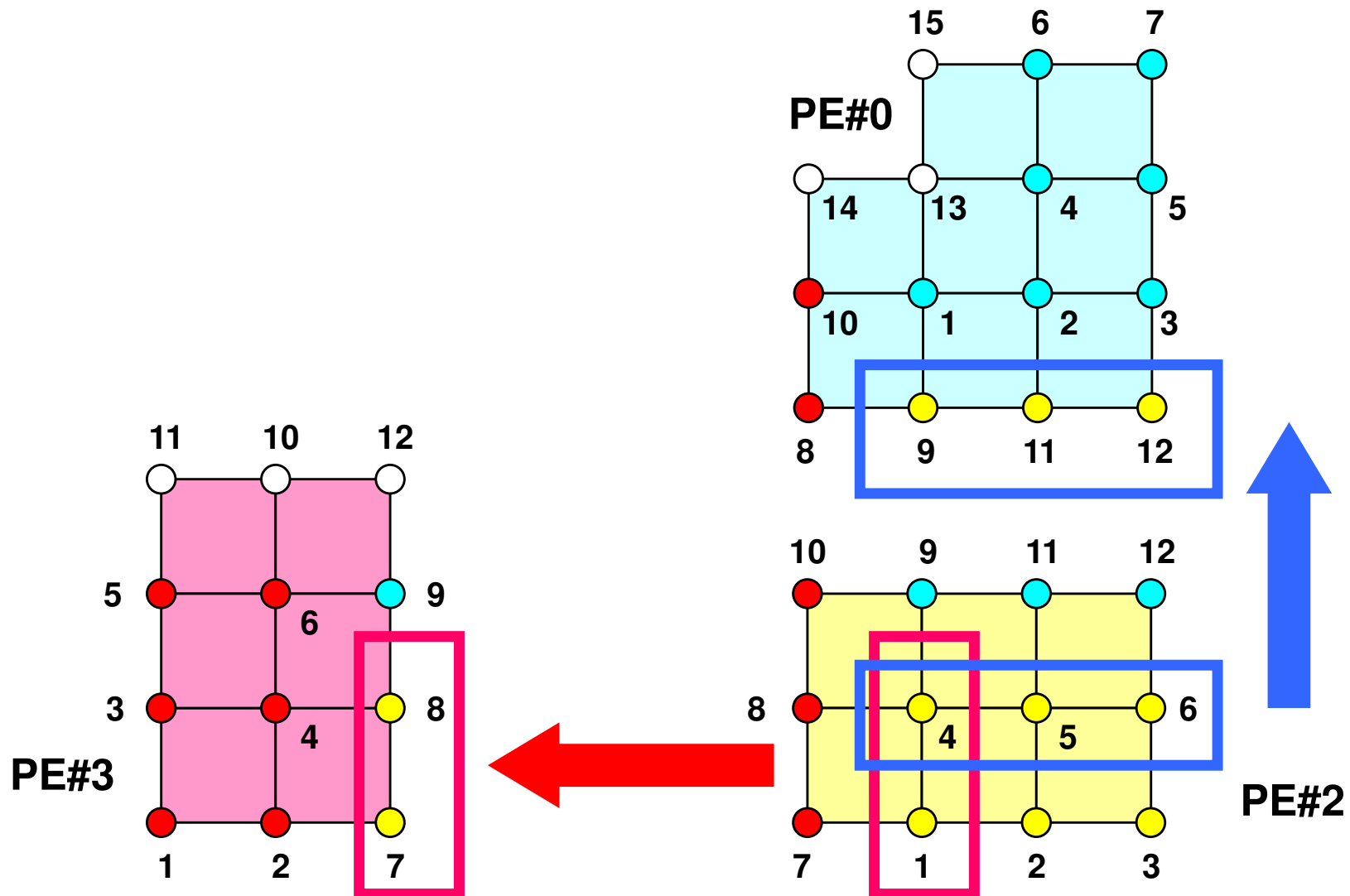


- 内点, 外点 (internal/external nodes)
  - 内点～外点となるように局所番号をつける
- 隣接領域情報
  - オーバーラップ要素を共有する領域
  - 隣接領域数, 番号
- 外点情報
  - どの領域から, 何個の, どの外点の情報を「受信:import」するか
- 境界点情報
  - 何個の, どの境界点の情報を, どの領域に「送信:export」するか



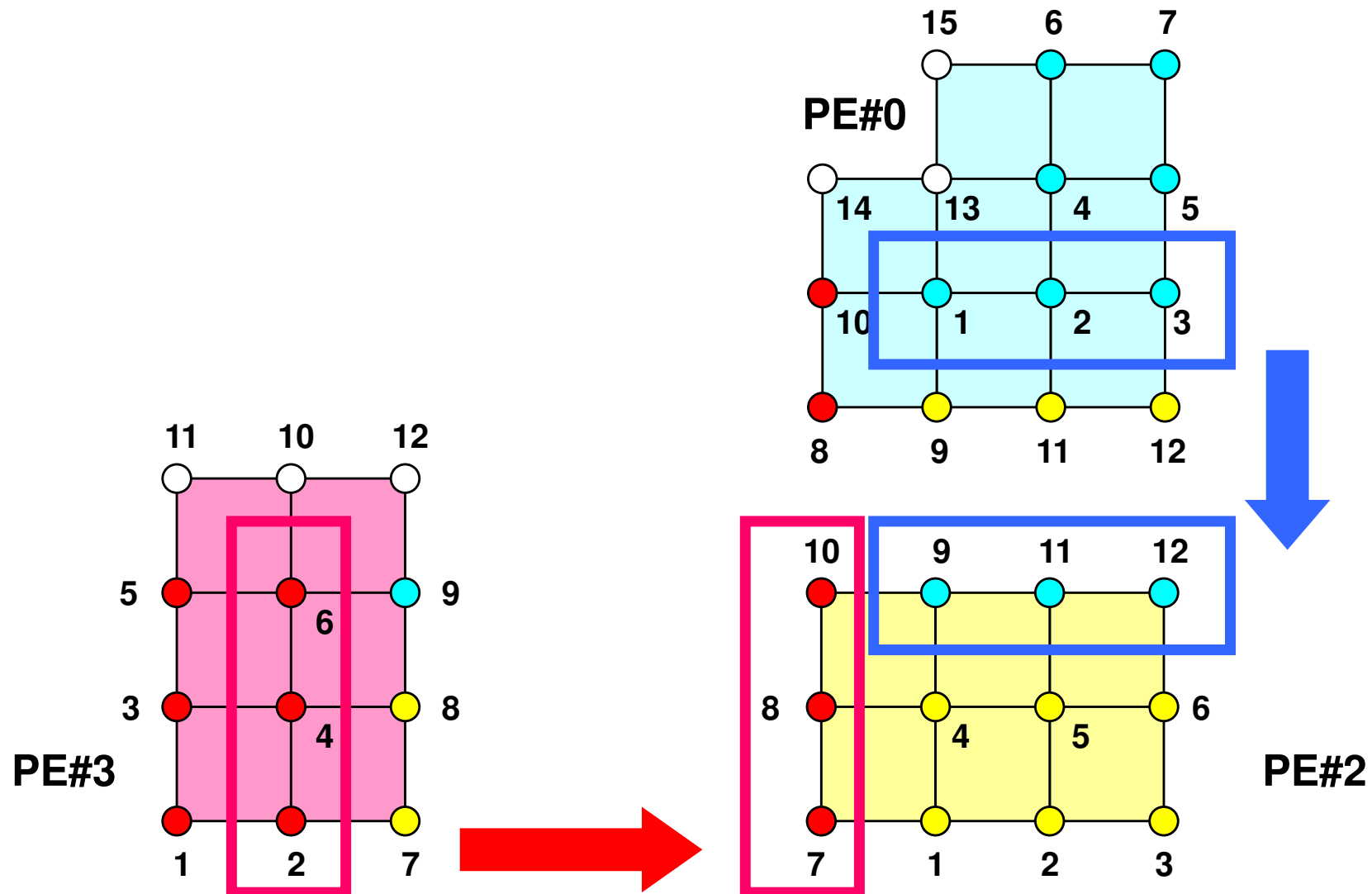
# Boundary Nodes (境界点) : SEND

PE#2 : send information on “boundary nodes”

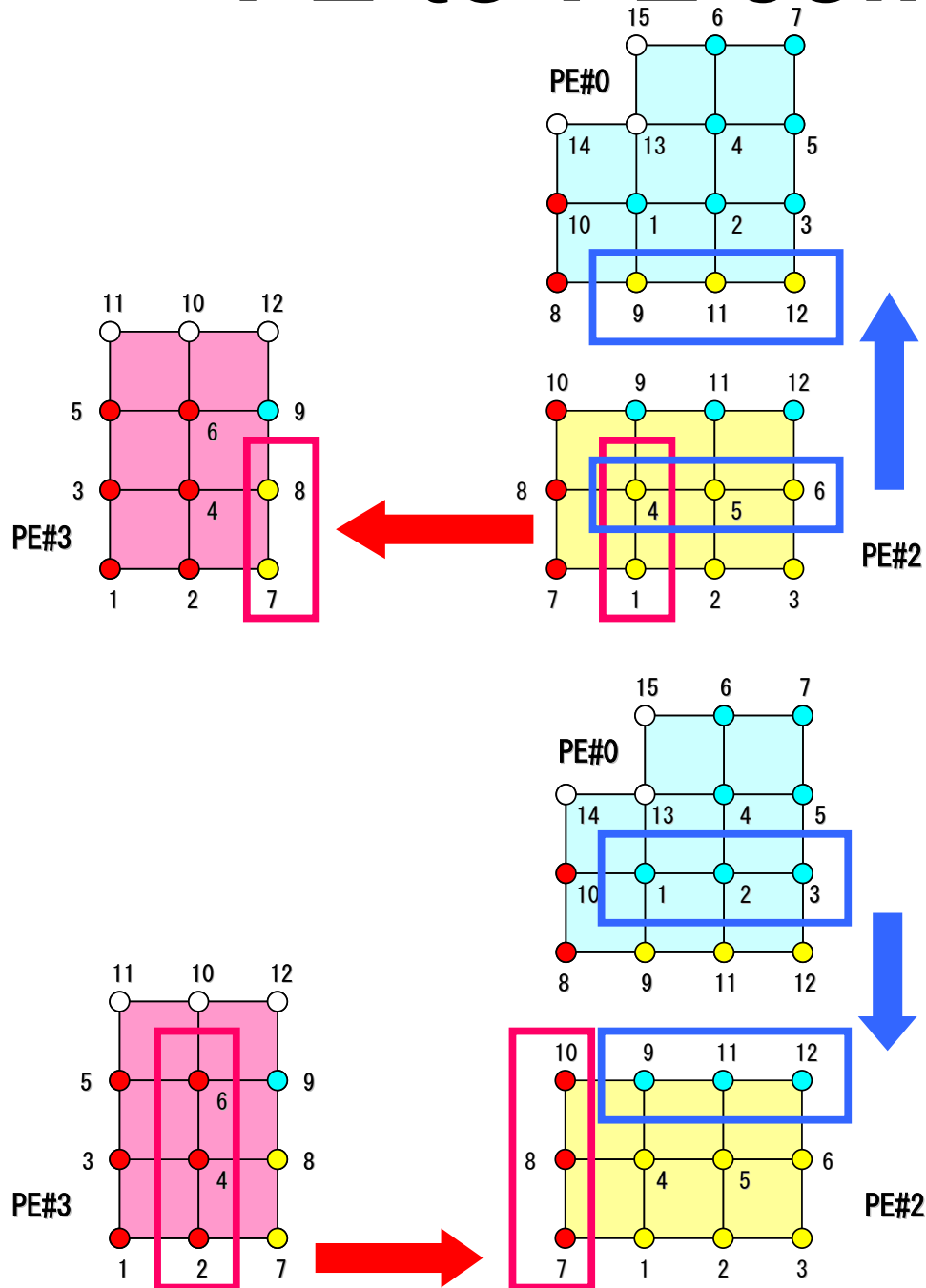


# External Nodes (外点) : RECEIVE

PE#2 : receive information for “external nodes”



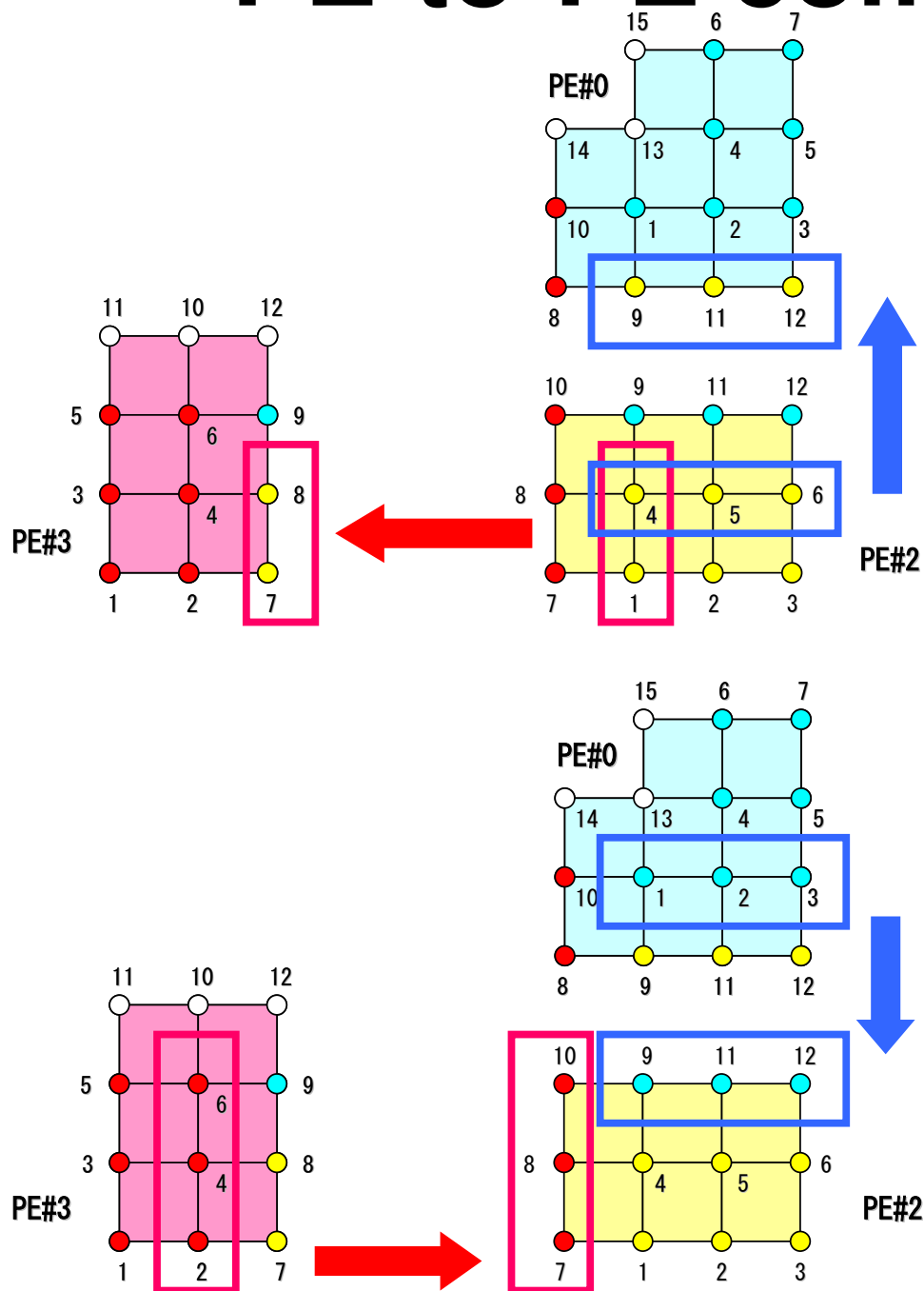
# PE-to-PE comm. : Local Data



(中略)

2	
2	
3	0
3	6
7	
8	
10	
9	
11	
12	
2	5
1	
4	
4	
5	
6	

# PE-to-PE comm. : Local Data

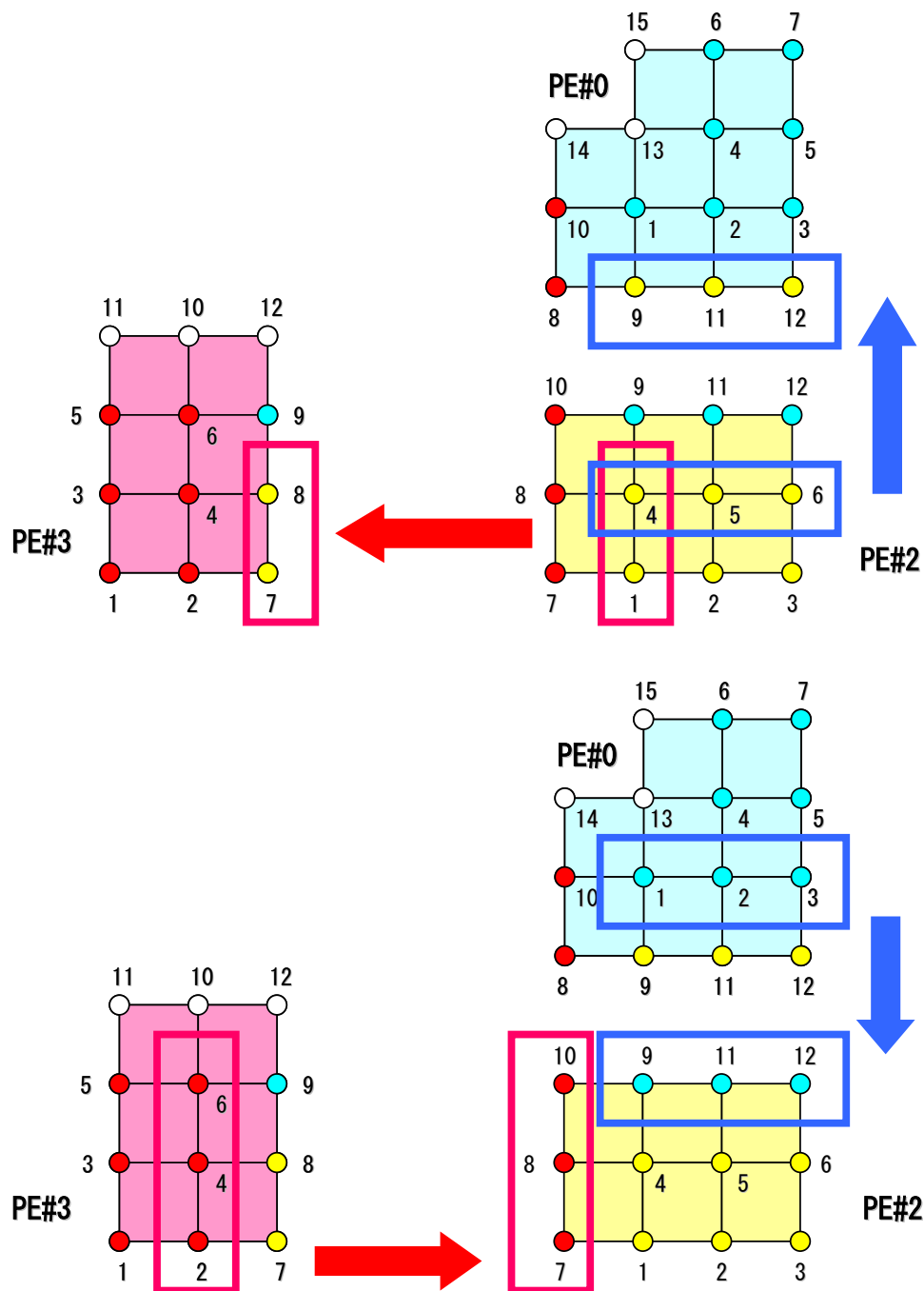


	領域ID	隣接領域数	隣接領域
	2		
	2		
	3	0	
(中略)			
	3	6	
	7		
	8		
	10		
	9		
	11		
	12		
	2	5	
	1		
	4		
	4		
	5		
	6		

NEIBPE= 2  
 NEIBPE[0]=3, NEIBPE[1]= 0

NEIBPE= 2  
 NEIBPE(1)=3, NEIBPE(2)= 0

# PE-to-PE comm. : SEND



```

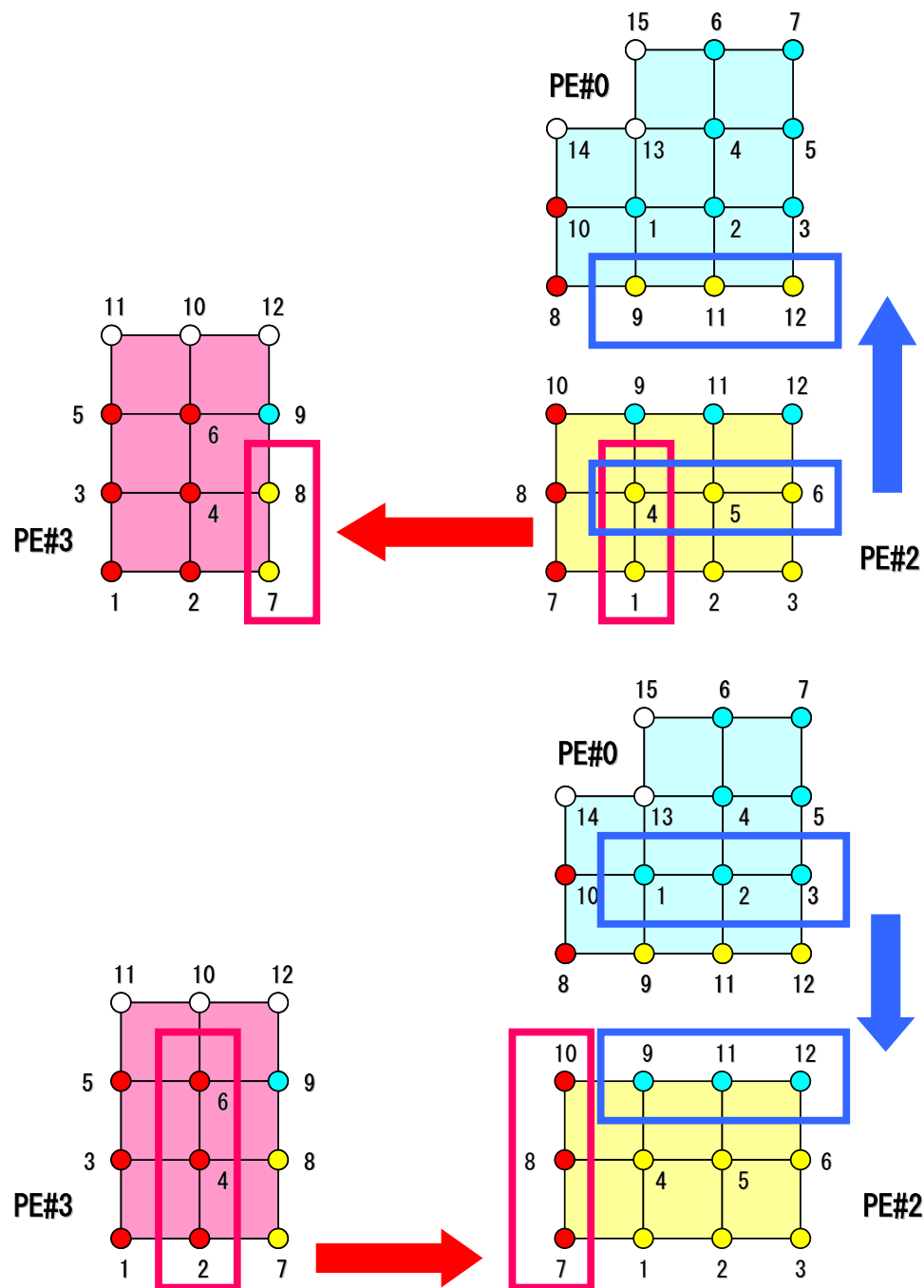
2
2
3      0
(中略)
3      6
7
8
10
9
11
12
2      5 export_index
1
4
4
5
6
    
```

```

export_index[0] = 0
export_index[1] = 2
export_index[2] = 2 + 3 = 5

export_item[0-4] = 1, 4, 4, 5, 6
export_item(1-5) = 1, 4, 4, 5, 6
4番の節点は2つの領域に送られる
    
```

# PE-to-PE comm. : RECV



```

2
2
3
(中略)
3
7
8
10
9
11
12
2
1
4
4
5
6
0
6 import_index
5
    
```

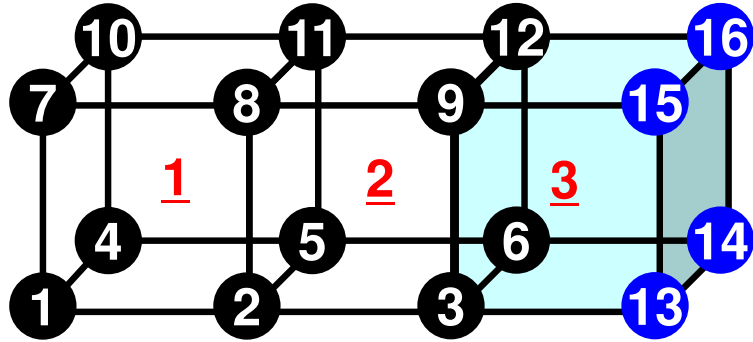
```

import_index[0] = 0
import_index[1] = 3
import_index[2] = 3 + 3 = 6

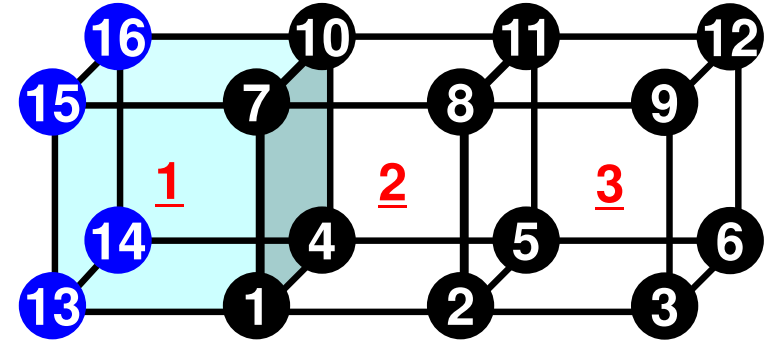
import_item[0-5] = 7, 8, 10, 9, 11, 12
import_item(1-6) = 7, 8, 10, 9, 11, 12
    
```

# 節点グループ

pc.0



pc.1



```

4
4 12 20 28
Xmin
1 4 7 10
Ymin
1 2 3 13 7 8 9 15
Zmin
1 2 3 13 4 5 6 14
Zmax
7 8 9 15 10 11 12 16

```

```

4
0 8 16 24
Xmin
Ymin
13 1 2 3 15 7 8 9
Zmin
13 1 2 3 14 4 5 6
Zmax
15 7 8 9 16 10 11 12

```

- pc.1

- Xminに属する節点が無いため、節点数が「0」となっている