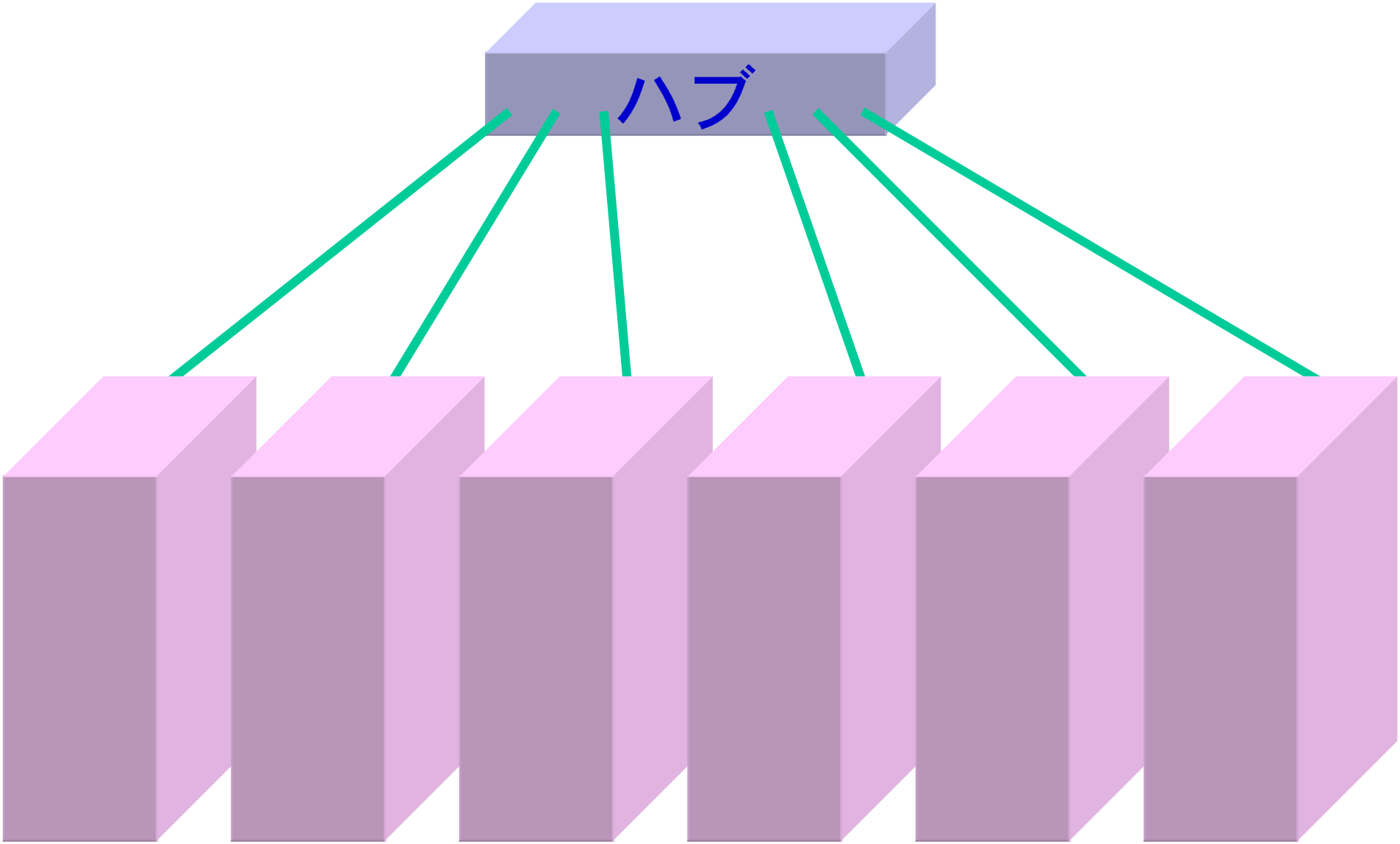
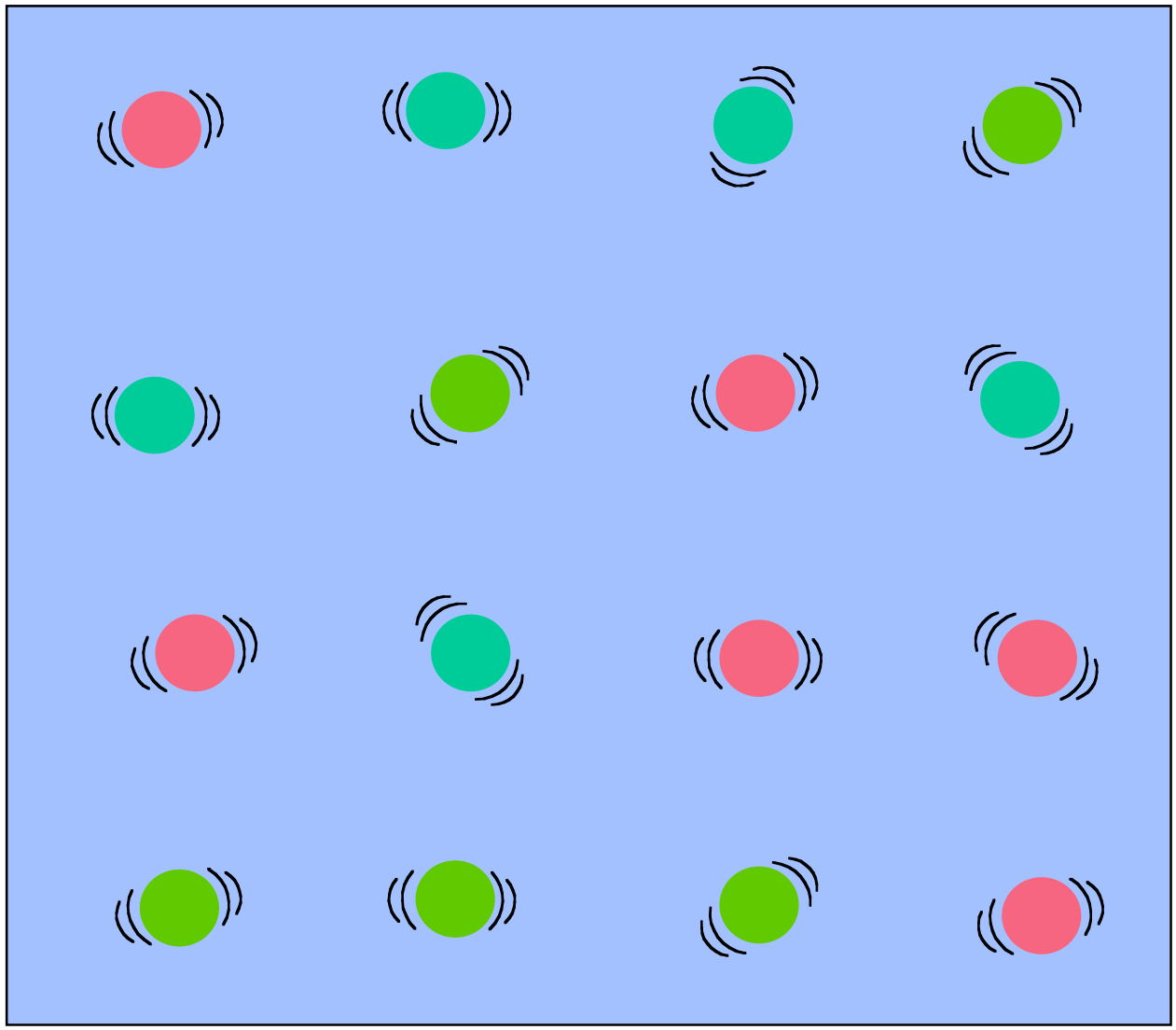
並列プログラミング入門

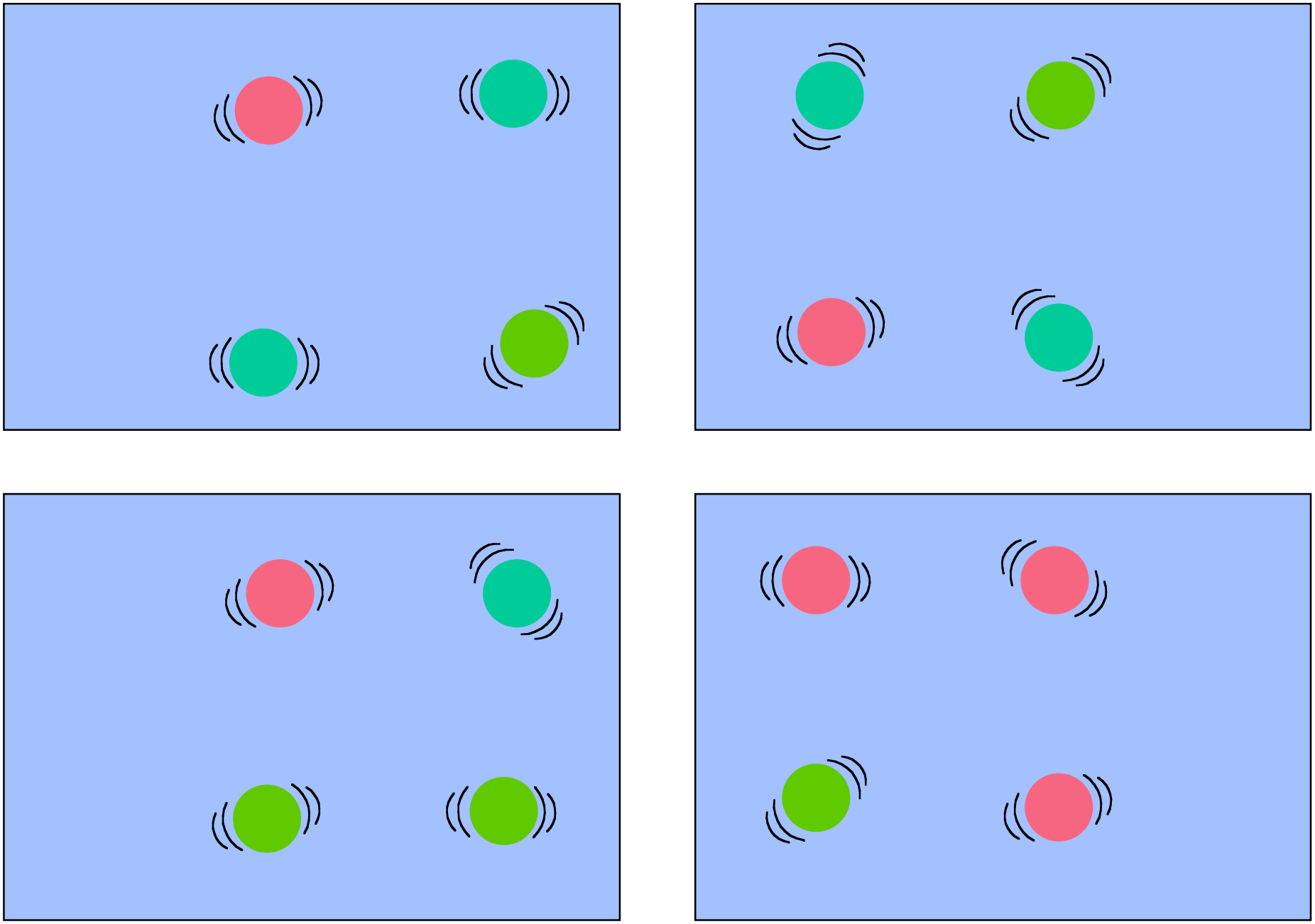
羽田野　直道

並列計算機の構造



並列計算とは





MPI による並列計算の流れ

(MPI = Message Passing Interface)

全 CPU に共通のプログラムを走らせる

MPI 初期化

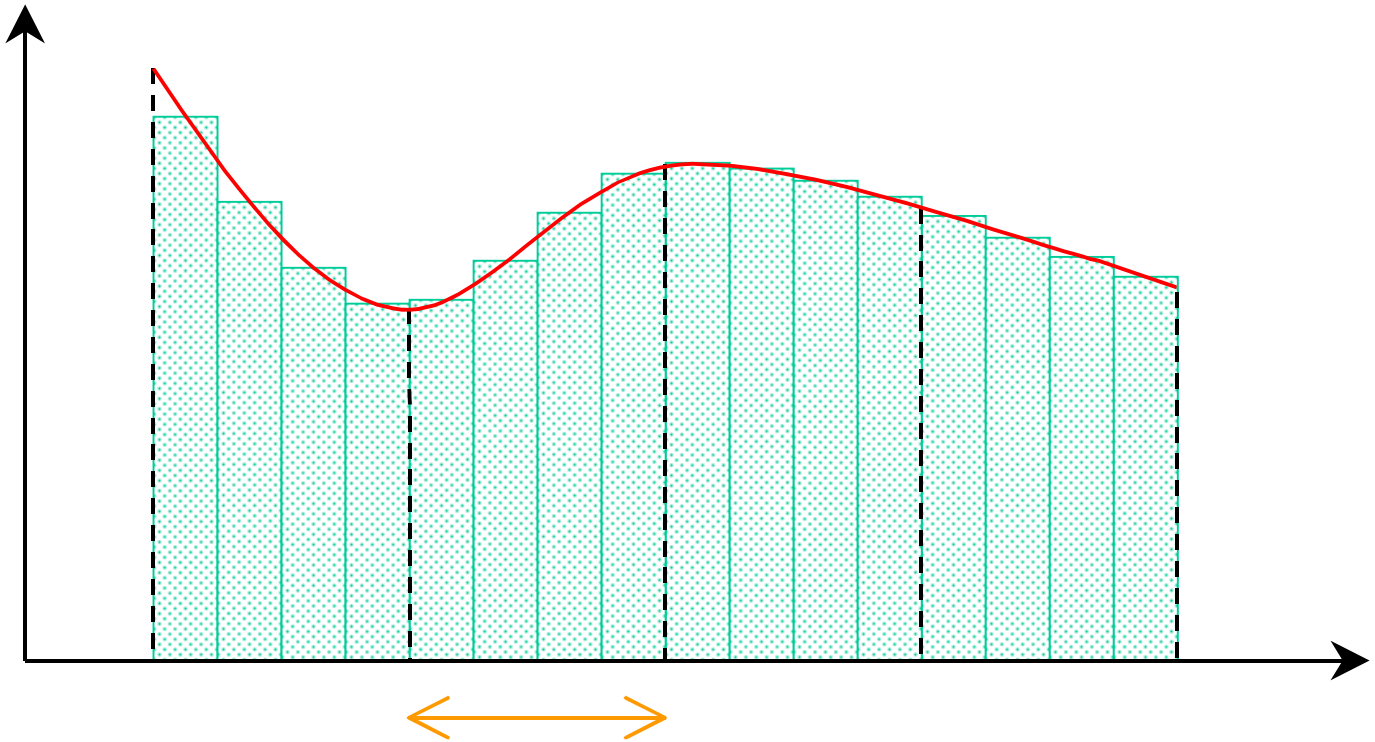
番号の割り当て
　N 個の CPU に 0 番から N-1 番まで番号をふる

自分が 0 番の CPU なら・・・・・・・・・

自分が 1 番の CPU なら・・・・・・・・・
4 番から 7 番へ・・・・・・・というデータを転送
・・・・・・・・・・

MPI 終了

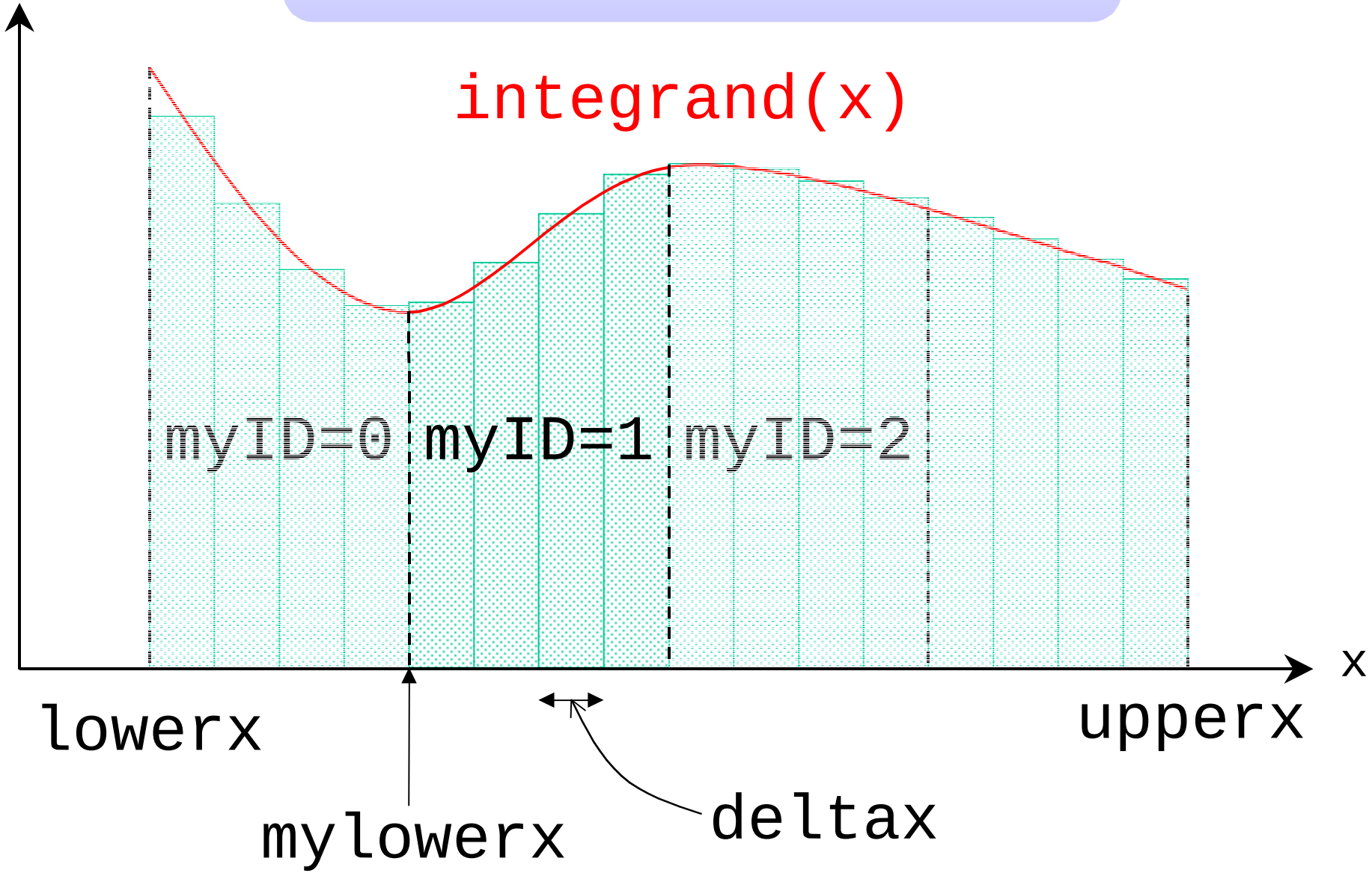
例：積分



プログラム

```
#include <math.h>
#include "mpi.h"

double integrand(double x);

/*
  この積分を台形則を使って計算する。

         / 1
         |       4
         |    --------   dx   =   3.14159265358979323846264338327950288419 7...
         |          2
         / 0   1 + x
*/

int main(int argc, char *argv[])
{
  int Nproc, myID, Ninterval, myNinterval, i;
  double integral, myintegral=0.0, x, deltax, lowerx=0.0, upperx=1.0, mylowerx;

  /*
    Nproc: 使うCPUの個数。
    myID : 自分の番号。myID=0, 1, 2, ..., Nproc-1.
    Ninterval: 積分範囲 0 -- 1 を Ninterval 個にわける。
    myNinterval: Ninterval のうち、各CPUが計算する個数。
  */

  /* MPIの準備 */
    MPI_Init(&argc,&argv);                    /* MPIの初期化 */
    MPI_Comm_size(MPI_COMM_WORLD, &Nproc);    /* CPUの個数を知る */
    MPI_Comm_rank(MPI_COMM_WORLD, &myID);     /* 自分の番号を知る */

  /* Ninterval を入力 */
    if(myID == 0) {           /* 0番の CPU から Ninterval を入力 */
      printf("The number of intervals? (Must be a multiple of %d): ",Nproc);
      scanf("%d", &Ninterval);
    }
    MPI_Bcast(&Ninterval, 1, MPI_INT, 0, MPI_COMM_WORLD);
                  /* 入力された Ninterval の値を他のCPUに教える */

  /* 積分の準備 */
    myNinterval = Ninterval / Nproc;
    deltax = (upperx - lowerx) / Ninterval;
    mylowerx = lowerx + (upperx - lowerx)/Nproc * myID;

  /* 積分の実行 */
    for(i = 1; i <= myNinterval; i++) {
      x = mylowerx + deltax * (i - 0.5);
      myintegral += deltax * integrand(x);
    }
    MPI_Reduce(&myintegral, &integral, 1, MPI_DOUBLE, MPI_SUM, 0, MPI_COMM_WORLD);
                                  /* 各CPUの計算した部分を合計する */

  /* 結果の出力 */
    if(myID == 0) printf("The integral is approximately %.16f.¥n", integral);

  /* MPIの終了 */
    MPI_Finalize();
}


double integrand(double x)
{return 4.0 / (1.0 + x * x);}
```

プログラム解説


integrand(x)
myID=0
myID=1
myID=2
x
lowerx
upperx
mylowerx
deltax

```
/* 積分の準備 */
    myNinterval = Ninterval / Nproc;
    deltax = (upperx - lowerx) / Ninterval;
    mylowerx = lowerx + (upperx - lowerx)/Nproc * myID;

  /* 積分の実行 */
    for(i = 1; i <= myNinterval; i++) {
      x = mylowerx + deltax * (i - 0.5);
      myintegral += deltax * integrand(x);
    }

……

double integrand(double x)
{return 4.0 / (1.0 + x * x);}
```

```
MPI_Init(&argc,&argv);
MPI_Comm_size(MPI_COMM_WORLD, &Nproc);
MPI_Comm_rank(MPI_COMM_WORLD, &myID);
```

MPI の使用を開始する。 ← MPI_Init

CPU の数を調べて ← MPI_Comm_size

Nproc に代入する。 ← &Nproc

自分の番号を調べて ← MPI_Comm_rank

myID に代入する。 ← &myID

全 CPU の情報は MPI_COMM_WORLD を参照せよ。

```
if(myID == 0) {
  printf("The number of intervals?: ",Nproc);
  scanf("%d", &Ninterval);
}
```

自分が第０CPU なら ← if(myID == 0)

プロンプトを出力して ← printf(.....)

Ninterval を読み込む。← scanf(.....)

注意：if(myID==0) としないと、
全 CPU がプロンプトを出力して、
全 CPU がデータを読み込もうとする。

```
MPI_Bcast(&Ninterval, 1, MPI_INT,
          0, MPI_COMM_WORLD);
```

Ninterval の値 ← &Ninterval

（整数が ← MPI_INT

１個分）を ← 1

第０ CPU から ← 0

全 CPU に転送する。 ← MPI_Bcast

全 CPU の情報は MPI_COMM_WORLD を参照せよ。

```
MPI_Reduce(&myintegral, &integral, 1, MPI_DOUBLE,
           MPI_SUM, 0, MPI_COMM_WORLD);
```

myintegral の値 ← &myintegral

（倍精度実数が ← MPI_DOUBLE

１個分のデータ）を ← 1

第０CPUに転送して ← 0

integral に ← &integral

足しあわせる。 ← MPI_SUM

転送先などの情報は MPI_COMM_WORLD を参照せよ。

```
#include "mpi.h"
```

MPI ライブラリを読み込む。

```
MPI_Finalize();
```

MPI の使用を終了する。

コンパイルと実行

```
% mpicc integral.c -o integral.e
```

mpicc によって、必要なライブラリが自動的にリンクされる。

```
% mpirun -machinefile machines.dat
                    -np 4 integral.e
```

使用するコンピューターの名前を書き込んだファイルを -machinefile で指定する。
そのうち、幾つの CPU を実際に使うかを -np で指定する。

例：行列とベクトルのかけ算

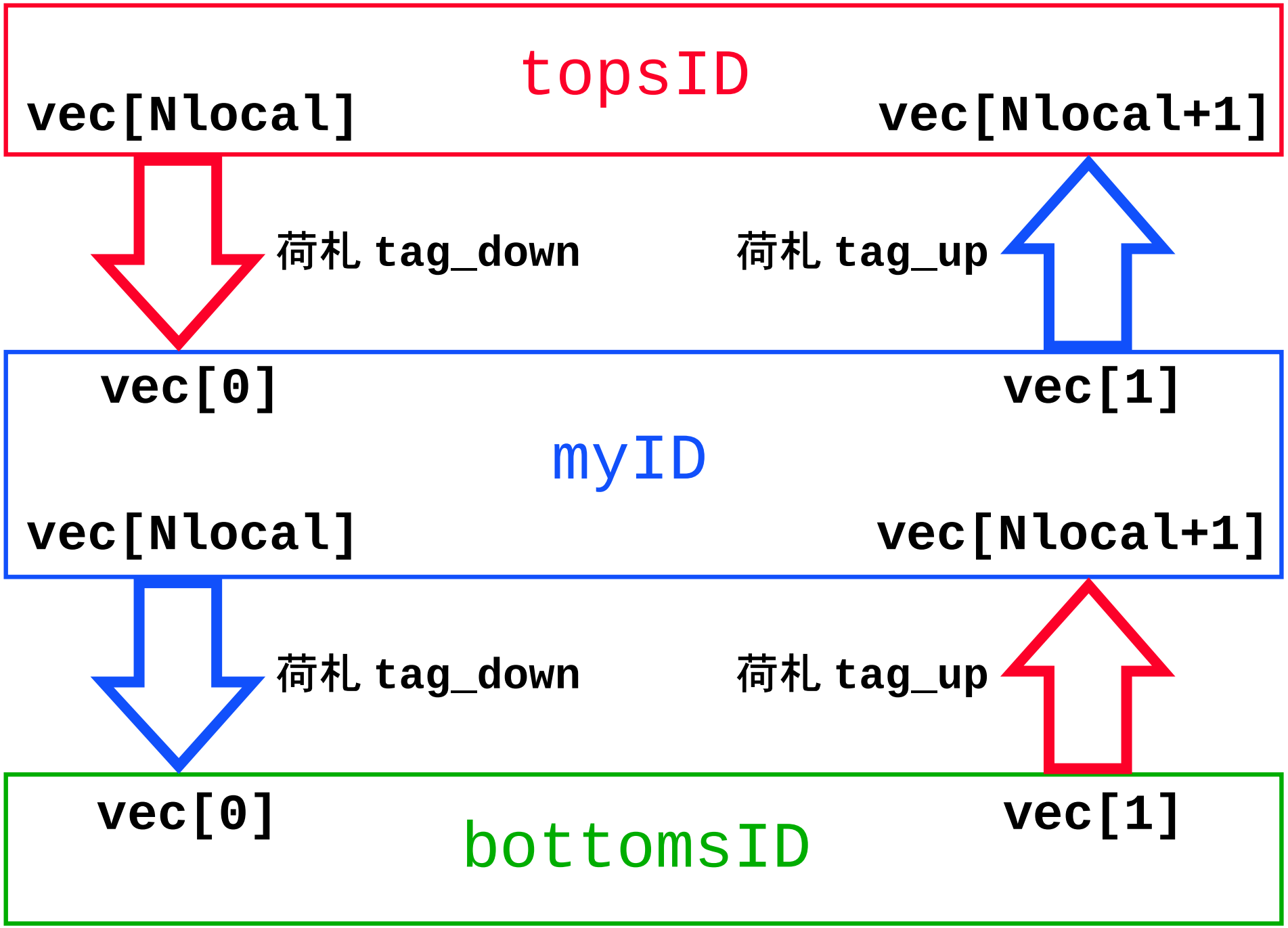
$$
\begin{pmatrix}
V^{(1)}(1) \\ V^{(1)}(2) \\ V^{(1)}(3) \\ \vdots \\ V^{(1)}(N_{\mathrm{loc}}) \\ \hline
V^{(2)}(1) \\ V^{(2)}(2) \\ V^{(2)}(3) \\ \vdots \\ V^{(2)}(N_{\mathrm{loc}}) \\ \hline
V^{(3)}(1) \\ \vdots \\ \vdots \\ \vdots \\ V^{(N_{\mathrm{p}}-1)}(N_{\mathrm{loc}}) \\ \hline
V^{(N_{\mathrm{p}})}(1) \\ V^{(N_{\mathrm{p}})}(2) \\ V^{(N_{\mathrm{p}})}(3) \\ \vdots \\ V^{(N_{\mathrm{p}})}(N_{\mathrm{loc}})
\end{pmatrix}
= \frac{1}{2}
\left(\begin{array}{ccccc|ccccc|ccccc|ccccc}
0 & 1 & & & & & & & & & & & & & 1 & & & & & \\
1 & 0 & 1 & & & & & & & & & & & & & & & & & \\
 & 1 & 0 & 1 & & & & & & & & & & & & & & & & \\
 & & \ddots & & \ddots & & & & & & & & & & & & & & & \\
 & & & 1 & 0 & 1 & & & & & & & & & & & & & & \\ \hline
 & & & & 1 & 0 & 1 & & & & & & & & & & & & & \\
 & & & & & 1 & 0 & 1 & & & & & & & & & & & & \\
 & & & & & & 1 & 0 & 1 & & & & & & & & & & & \\
 & & & & & & & \ddots & & \ddots & & & & & & & & & & \\
 & & & & & & & & 1 & 0 & 1 & & & & & & & & & \\ \hline
 & & & & & & & & & 1 & 0 & 1 & & & & & & & & \\
 & & & & & & & & & & \ddots & & \ddots & & & & & & & \\
 & & & & & & & & & & & \ddots & & \ddots & & & & & & \\
 & & & & & & & & & & & & \ddots & & \ddots & & & & & \\
 & & & & & & & & & & & & & 1 & 0 & 1 & & & & \\ \hline
 & & & & & & & & & & & & & & 1 & 0 & 1 & & & \\
 & & & & & & & & & & & & & & & 1 & 0 & 1 & & \\
 & & & & & & & & & & & & & & & & \ddots & & \ddots & \\
 & & & & & & & & & & & & & & & & & 1 & 0 & \\
1 & & & & & & & & & & & & & & & & & & & \\
\end{array}\right)
\begin{pmatrix}
v^{(1)}(1) \\ v^{(1)}(2) \\ v^{(1)}(3) \\ \vdots \\ v^{(1)}(N_{\mathrm{loc}}) \\ \hline
v^{(2)}(1) \\ v^{(2)}(2) \\ v^{(2)}(3) \\ \vdots \\ v^{(2)}(N_{\mathrm{loc}}) \\ \hline
v^{(3)}(1) \\ \vdots \\ \vdots \\ \vdots \\ v^{(N_{\mathrm{p}}-1)}(N_{\mathrm{loc}}) \\ \hline
v^{(N_{\mathrm{p}})}(1) \\ v^{(N_{\mathrm{p}})}(2) \\ v^{(N_{\mathrm{p}})}(3) \\ \vdots \\ v^{(N_{\mathrm{p}})}(N_{\mathrm{loc}})
\end{pmatrix}
$$

```
for(Idim=1; Idim <= Nlocal; Idim++) {
  vecout[Idim] = 0.5*(vec[Idim-1] + vec[Idim+1]);
}
```

行列をベクトルに掛ける。

$$V_i = \frac{1}{2}\left(v_{i-1} + v_{i+1}\right)$$

ここで vec[0] と vec[Nlocal+1] は隣の CPU から受信しなくてはならない。同時に vec[1] と vec[Nlocal] を隣の CPU へ送信しなくてはならない。


topsID
vec[Nlocal]
vec[Nlocal+1]
荷札 tag_down
荷札 tag_up
vec[0]
vec[1]
myID
vec[Nlocal]
vec[Nlocal+1]
荷札 tag_down
荷札 tag_up
vec[0]
vec[1]
bottomsID

プログラム

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

/*  ベクトルに行列をかける。行列は、対角要素の一つ上と一つ下に0.5がある。  */

int MLTPLY(double *vec, double *vecout, int Nlocal, int topsID, int bottomsID, MPI_Comm communicator) {

int ierr = 0, tag_down = 0, tag_up = 1, Idim;
MPI_Status status[4];          /*  送受信が完了したかどうかを知るための変数 */
MPI_Request request[4];        /*  送信や受信につける番号 */

/* 受信要請 */

 /* topsID という CPU から tag_down という名札のついた倍精度実数１個を受け取り vec[0] に格納したい。*/
   ierr = MPI_Irecv(&vec[0],        1, MPI_DOUBLE, topsID,   tag_down, communicator, &request[0]);

 /* bottomsID という CPU から tag_up という名札のついた倍精度実数１個を受け取り vec[Nlocal+1] に格納したい。*/
   ierr = MPI_Irecv(&vec[Nlocal+1], 1, MPI_DOUBLE, bottomsID,tag_up,   communicator, &request[1]);

/* 送信開始 */

 /* bottomsID という CPU へ tag_down という名札をつけて倍精度実数１個 vec[Nlocal] を送信する。*/
  ierr = MPI_Isend(&vec[Nlocal],   1, MPI_DOUBLE, bottomsID,tag_down, communicator, &request[2]);

 /* topsID という CPU へ tag_up という名札をつけて倍精度実数１個 vec[1] を送信する。*/
  ierr = MPI_Isend(&vec[1],        1, MPI_DOUBLE, topsID,   tag_up,   communicator, &request[3]);

/* 送受信完了まで待機 */
  ierr = MPI_Waitall(4, request, status);

/* 行列の掛け算 */
  for(Idim=1; Idim <= Nlocal; Idim++) {
    vecout[Idim] = 0.5 * (vec[Idim-1] + vec[Idim+1]);
  }

return ierr;
}
```

受信要請

```
MPI_Irecv(&vec[0],          1, MPI_DOUBLE,
          topsID,   tag_down,
             communicator, &request[0]);
MPI_Irecv(&vec[Nlocal+1], 1, MPI_DOUBLE,
          bottomsID,tag_up,
             communicator, &request[1]);
```

topsID という CPU から、tag_down という荷札のついた倍精度実数１個を受信して vec[0] に格納したい。

bottomsID という CPU から、tag_up という荷札のついた倍精度実数１個を受信して vec[Nlocal+1] に格納したい。

送信開始

```
MPI_Isend(&vec[Nlocal],   1, MPI_DOUBLE,
            bottomsID,tag_down,
               communicator, &request[2]);
MPI_Isend(&vec[1],           1, MPI_DOUBLE,
            topsID,   tag_up,
               communicator, &request[3]);
```

bottomsID という CPU へ、tag_down という荷札をつけて倍精度実数１個 vec[Nlocal] を送信する。

topsID という CPU へ、tag_up という荷札をつけて倍精度実数１個 vec[1] を送信する。

```
MPI_Waitall(4, request, status);
```

Request[0] から request[3] まで４つの送受信が全て完了するまで待機する。
（MPI_Irecv は受信要請を出すと、先へ進んでしまう。ちゃんと受信したかどうかを確認しておく必要がある。）

中級編・上級編へ向けて