

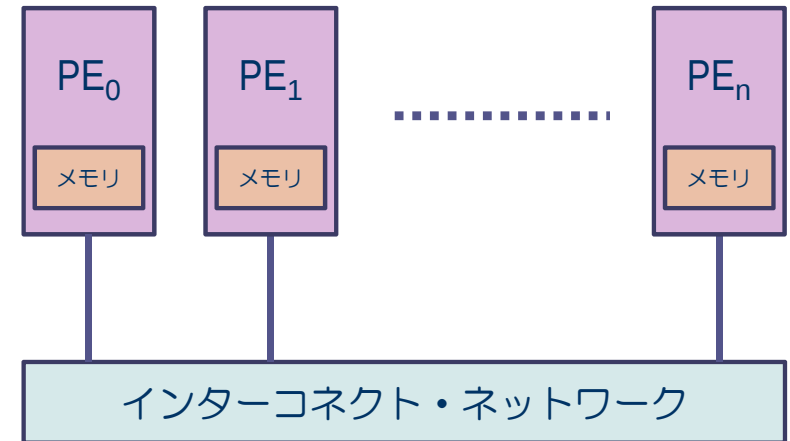
MPI（片方向通信）

2019年3月15日

神戸大学大学院システム情報学研究科計算科学専攻
横川三津夫

分散メモリ型並列計算機

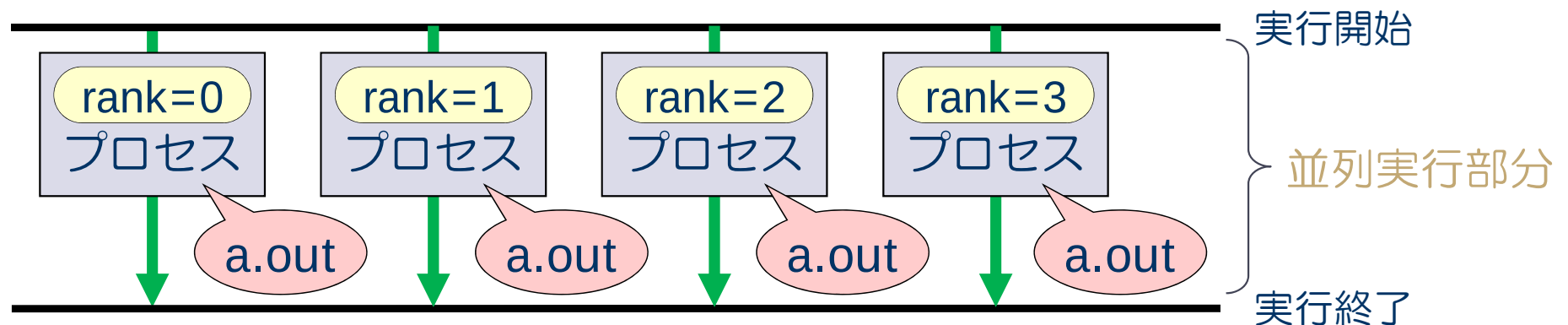
- 複数のプロセッサがネットワークで接続されており，それぞれのプロセッサ（PE）が，メモリを持っている.
 - ◆ 各PEが自分のメモリ領域のみアクセス可能



- 特徴
 - ◆ 数千から数万PE規模の並列システムが可能
 - ◆ PEの間のデータ分散を意識したプログラミングが必要.
- プログラミング技術
 - ◆ メッセージ・パッシング・インターフェイス（MPI）によるプログラミング

MPIの実行モデル：SPMD (Single Program, Multiple Data)

- 複数のプロセスにより並列実行
- 実行開始から終了まで，全プロセスが**同じプログラム**を実行
- 各MPIプロセスは**固有の番号（ランク番号）**を持つ
 - ◆ P個のプロセスで実行する場合，プロセス番号は0から(P-1)までの整数
- 各プロセスで処理を変えたいときは，ランク番号を使った分岐により，各プロセスの処理を記述する。

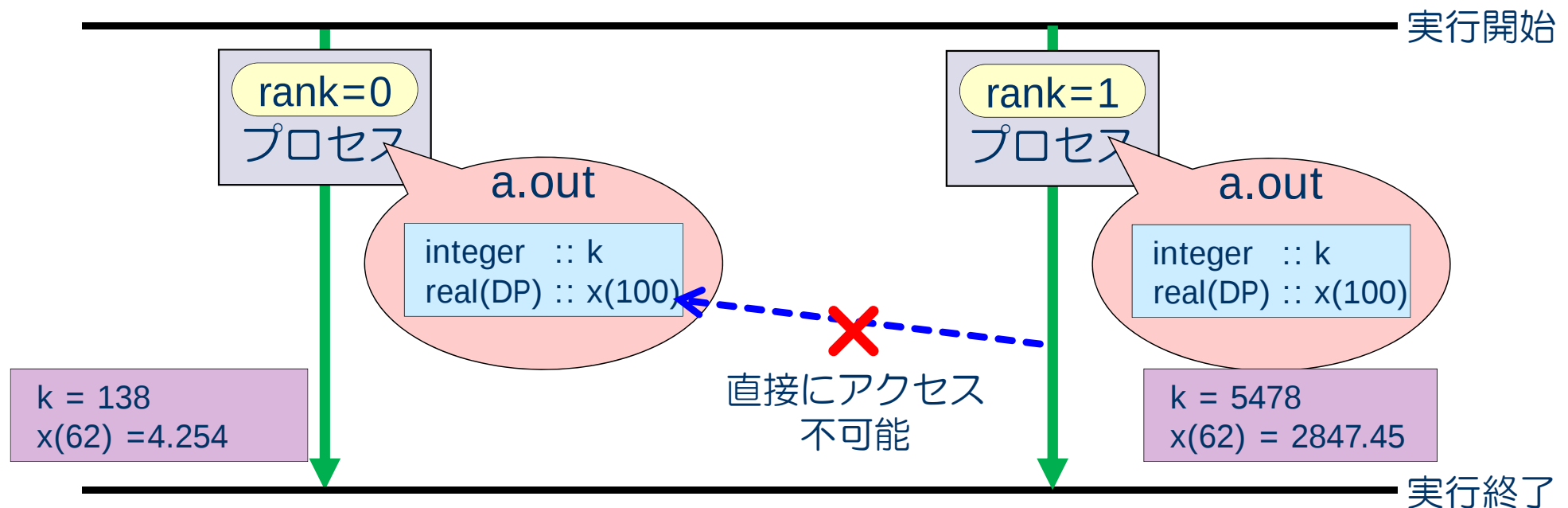


MPIの実行モデル（続き）

■ メモリ空間

◆ プロセスごとに**独立したメモリ空間**を保持

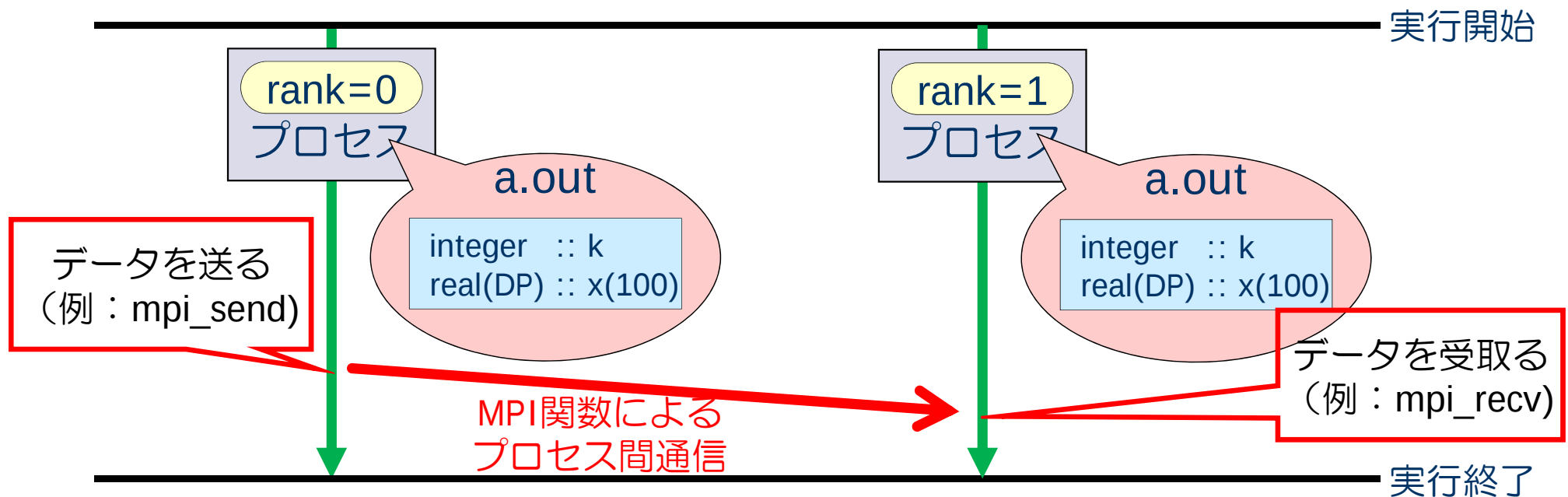
- プログラム中で定義された変数や配列は、**同じ名前**で独立に各プロセスのメモリ上に割り当てられる。
- 同じ変数や配列に対して、**プロセスごとに違う値を与えることが可能**
- **他のプロセスの持つ変数や配列には、直接にアクセスできない。**



MPIの実行モデル（続き）

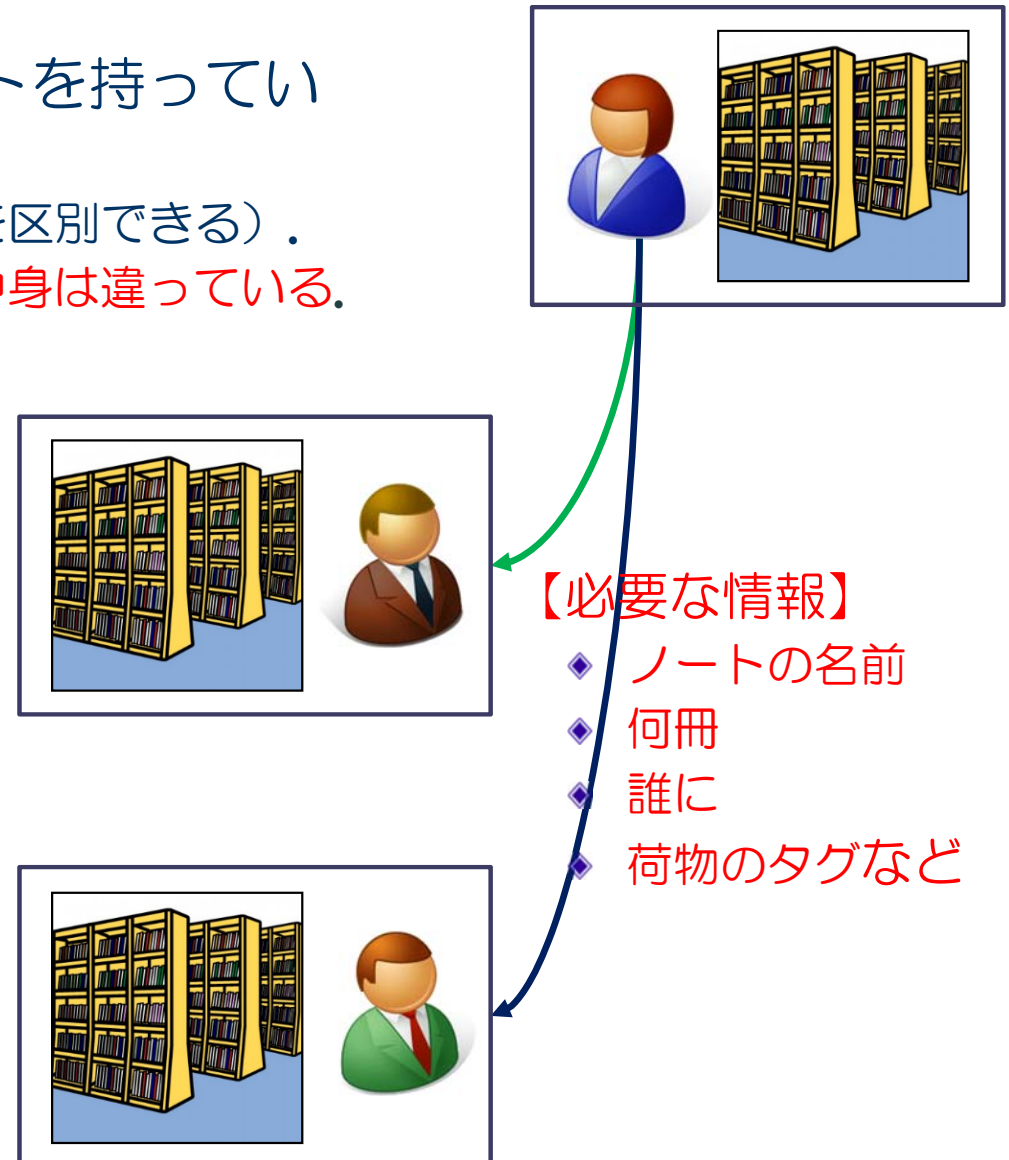
■ プロセス間通信

- ◆ 他のプロセスの持つ変数や配列のデータにアクセスできない。
⇒ プロセス間通信によりデータを送ってもらう。
- ◆ メッセージパッシング方式：メッセージ（データ）の送り手と受け手
- ◆ この方式によるプロセス間通信関数の集合 ≡ MPI



SPMDのイメージ

- それぞれの人が、本棚に一連のノートを持っている。
 - ◆ それぞれの人には、名前が付いている（人を区別できる）。
 - ◆ ノートには同じ名前が付けられているが、中身は違っている。
- 本棚のノートに対し、それぞれの人が、読んだり書いたり...
 - ◆ 大体は同じ作業をするが、最初のノートの中身が違うので、中身はそれぞれ違う。
 - ◆ ある人に、他の人とは違う作業をさせたい場合には、名前で作業と指示してあげる。
- 時々、他の人のノートを見たい。
 - ◆ 相手にノートの中身を送ってあげる。
 - ◆ 送られた人は、それを違う名前のノートに中身を書き写す。



MPIプログラムのスケルトン (C)

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

MPIモジュールの取り込み（おまじない1）

```
int main(int argc, char **argv)
{
```

```
    int  nprocs, myrank;
```

MPIで使う変数の宣言

```
    MPI_Init( &argc, &argv );
```

MPIの初期化（おまじない2）

```
    MPI_Comm_size( MPI_COMM_WORLD, &nprocs );
```

MPIで使うプロセス数を nprocs に取得
自分のプロセス番号を myrank に取得

```
    MPI_Comm_rank( MPI_COMM_WORLD, &myrank );
```

(この部分に並列実行するプログラムを書く)

```
    MPI_Finalize();
```

```
    return 0 ;
```

MPIの終了処理（おまじない3）

```
}
```

▮ それぞれのプロセスで異なった処理をする場合は、myrankの値で場合分けし、うまく仕事が振り分けられるようにする。

MPIプログラムのスケルトン (Fortran)

```
program main
use mpi
implicit none
integer :: nprocs, myrank, ierr
```

MPIモジュールの取り込み (おまじない1)

MPIで使う変数の宣言

```
call mpi_init( ierr )
call mpi_comm_size( MPI_COMM_WORLD, nprocs, ierr )
call mpi_comm_rank( MPI_COMM_WORLD, myrank, ierr )
```

MPIの初期化 (おまじない2)

MPIで使うプロセス数を `nprocs` に取得
自分のプロセス番号を `myrank` に取得

(この部分に並列実行するプログラムを書く)

```
call mpi_finalize( ierr )
```

MPIの終了処理 (おまじない3)

```
end program main
```

📌 それぞれのプロセスが何の計算をするかは、`myrank`の値で場合分けし、うまく仕事が振り分けられるようにする。

講義の内容

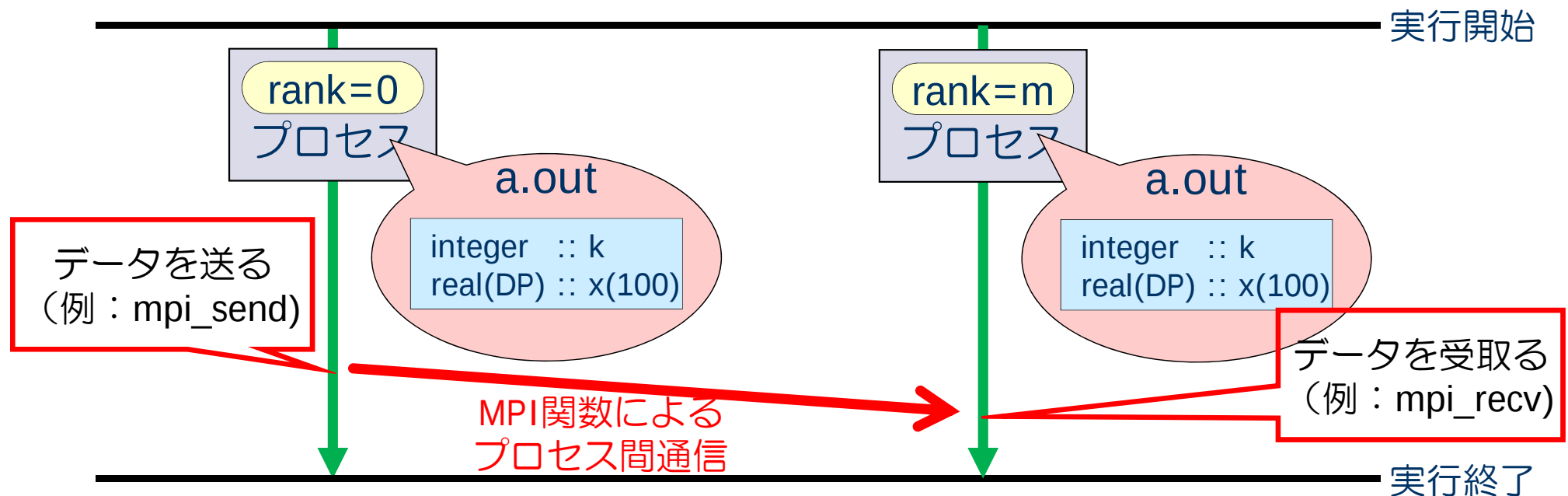
- メッセージ・パッシング・インターフェイス (MPI)
- 双方向通信
- 片方向通信
- 演習問題

メッセージ・パッシング・インターフェイス

- Message Passing Interface (MPI) とは. . .
 - ◆ 複数の独立したプロセス間で、並列処理を行うためのプロセス間メッセージ通信の標準規格
 - ◆ 1992年頃より米国の計算機メーカ、大学などを中心に標準化
 - ◆ MPI規格化の歴史
 - 1994 MPI-1
 - 1997 MPI-2（一方向通信など）
 - 2012 MPI-3
- ④ <http://www.mpi-forum.org/docs/mpi-3.0/mpi30-report.pdf>

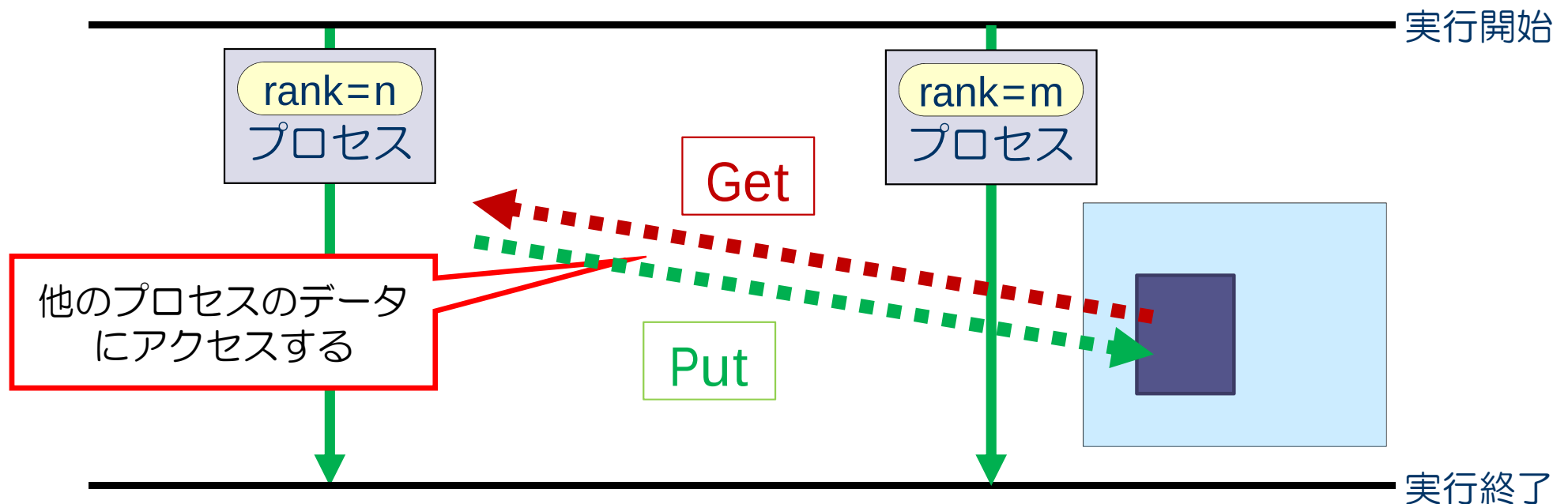
双方向通信 (one-to-one communication)

- 送信側と受信側で、対応する関数を呼ぶ。
- MPI_send, MPI_recvなどの組合せ



片方向通信（One-sided communication）

- 通信相手の状態に関係なく，他のプロセスのデータをアクセスする通信方法
 - ◆ Get 相手のプロセスのデータを獲得
 - ◆ Put 相手のプロセスにデータを挿入



片方向通信の利点

- プロセス間の同期待ちを削減
 - ◆ 性能向上の可能性
- データコピーの回数を削減
 - ◆ 双方向通信は、データの間接バッファを用いた実装
- RDMA（Remote Direct Memory Access）機構を持つシステムでは、高速実行が可能となる。
 - ◆ 計算とデータ通信のオーバーラップ
- 大きなデータ通信において高速実行される場合がある。

Windowオブジェクトの生成

```
【C】 int MPI_Win_create(void *base, MPI_Aint size, int disp_unit,  
                        MPI_Info info, MPI_Comm comm, MPI_Win *win )
```

```
【F】 mpi_win_create( base, size, disp_unit, info, comm, win, ierr )
```

RMA操作に必要なWindowオブジェクトを生成する.

[Input]

- ◆ base: windowの先頭アドレス
- ◆ size: windowのサイズ (整数型, バイト単位で指定)
 - ◆ Fortranの場合, `integer(kind=MPI_ADDRESS_KIND):: size` とする.
- ◆ disp_unit: ずれ当りのサイズ (整数型, バイト単位で指定)
 - ◆ MPI_INTEGER, MPI_REALなどではない. MPI_Get, MPI_Putなどで使用する.
- ◆ info: 情報 (Fortranでは整数型, CではMPI_Info型)
 - ◆ infoは今回は使わないので, MPI_INFO_NULL とすればよい.
- ◆ comm: コミュニケータ (MPI_COMM_WORLDなど)

[Output]

- ◆ win: Windowオブジェクト (Fortranでは整数型, CではMPI_Win型)
 - ◆ MPI_get, MPI_Putなどで利用
- ◆ ierr: 戻りコード (整数型)

Windowオブジェクトの開放

```
【C】 int MPI_Win_free( MPI_Win *win )
```

```
【F】 mpi_win_free( win, ierr )
```

生成したWindowオブジェクトを開放する.

[Input/Output]

◆ win: 生成したwindowオブジェクト

[Output]

◆ ierr: 戻りコード (整数型)

リモートプロセスのデータの獲得

```
【C】 int MPI_Get( void *oaddr, int ocount, MPI_Datatype odatatype,  
                  int target_rank, MPI_Aint tdisp, int tcount, MPI_Datatype  
                  tdatatype, MPI_Win win)
```

```
【F】 mpi_get( oaddr, ocount, odatatype, target_rank, tdisp, tcount, tdatatype,  
              win, ierr )
```

[Input/Output]

- ◆ oaddr: 自分のプロセスの, データを格納する変数の先頭アドレス
- ◆ ocount: データの個数 (整数型)
- ◆ odatatype: データの型, MPI_INTEGER, MPI_REALなど. CではMPI_INTなど.
- ◆ target_rank: リモートプロセスのMPIランク番号
- ◆ tdisp: 先頭からのずれ (整数型)
 - ◆ 獲得する変数の先頭アドレスは, $\text{base} + \text{tdisp} \times \text{disp_unit}$
 - ◆ Fortranの場合, `integer(kind=MPI_ADDRESS_KIND):: tdisp` と宣言する.
- ◆ tcount: ターゲット側のデータの個数 (整数型)
- ◆ tdatatype: ターゲット側のデータの型
 - ◆ FortranではMPI_INTEGER, MPI_REALなど. CではMPI_INTなど
- ◆ win: 通信するwindowオブジェクト

[Output]

- ◆ ierr: 戻りコード (整数型)

リモートプロセスへのデータの書き込み

```
【C】 int MPI_Put( const void *oaddr, int ocount, MPI_Datatype odatatype,  
                  int target_rank, MPI_Aint tdisp, int tcount, MPI_Datatype  
                  tdatatype, MPI_Win win)
```

```
【F】 mpi_put( oaddr, ocount, odatatype, target_rank, tdisp, tcount, tdatatype,  
              win, ierr )
```

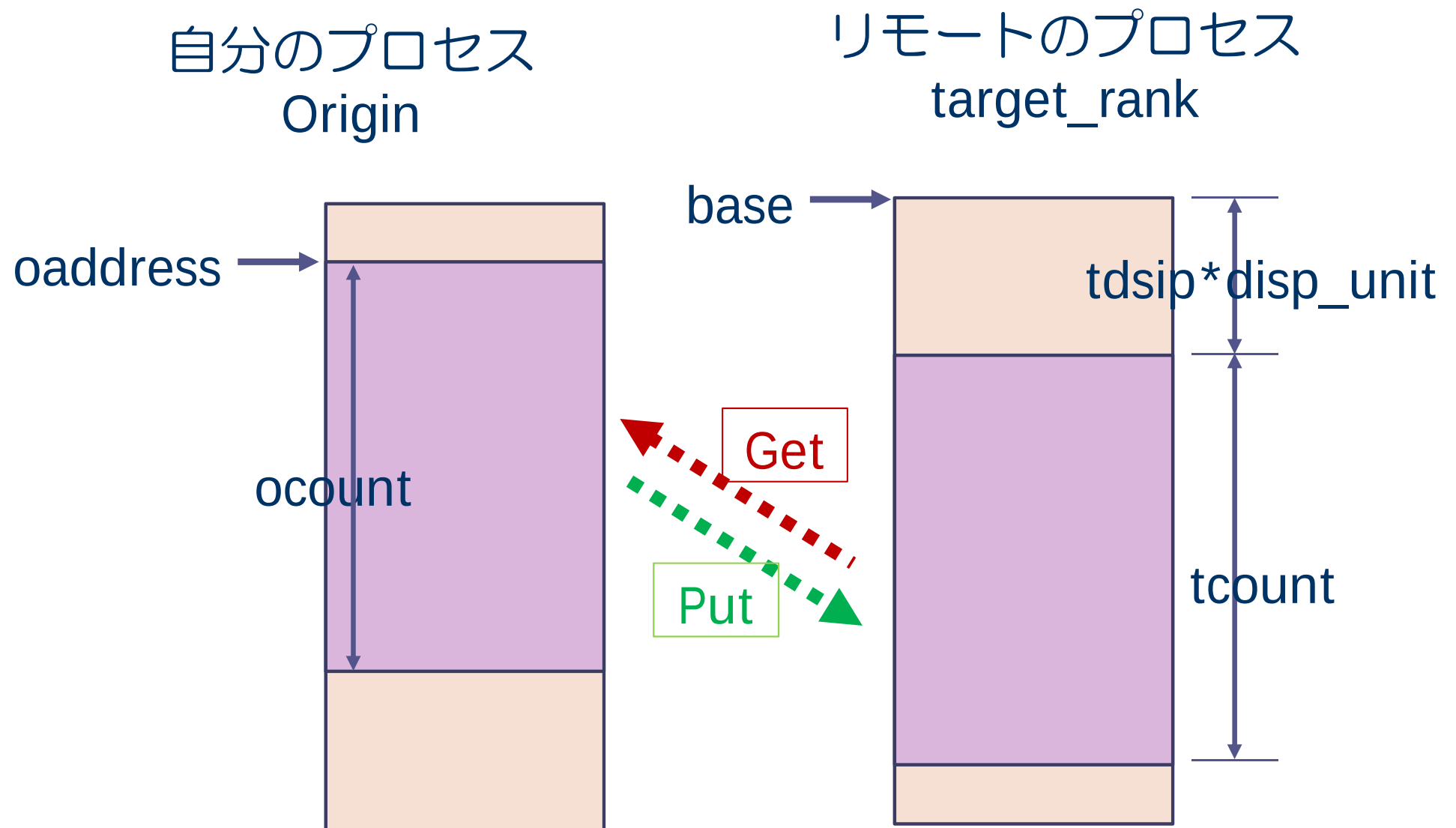
[Input/Output]

- ◆ oaddr: 自分のプロセスの、書き込むデータの先頭アドレス
- ◆ ocount: データの個数（整数型）
- ◆ odatatype: データの型, MPI_INTEGER, MPI_REALなど. CではMPI_INTなど.
- ◆ target_rank: リモートプロセスのMPIランク番号
- ◆ tdisp: 先頭からのずれ（整数型）
 - ◆ 獲得する変数の先頭アドレスは, $\text{base} + \text{tdisp} \times \text{disp_unit}$
 - ◆ Fortranの場合, `integer(kind=MPI_ADDRESS_KIND):: tdisp` と宣言する.
- ◆ tcount: ターゲット側のデータの個数（整数型）
- ◆ tdatatype: ターゲット側のデータの型（整数型）
 - ◆ MPI_INTEGER, MPI_REALなど. CではMPI_INTなど.
- ◆ win: 通信するwindowオブジェクト

[Output]

- ◆ ierr: 戻りコード（整数型）

片方向通信関数の引数の意味



片方向通信の同期

【C】 `int MPI_Win_fence( int assert, MPI_Win win )`

【F】 `mpi_win_fence( assert, win, ierr )`

winオブジェクトの片方向通信関数の同期を取る。fence関数の前に片方向通信が終了している。

[Input/Output]

◆ `assert`: Windowの状態の確認用。
通常は 0 でよい。MPI_MODE_NOPRECEDE など

◆ `win`: windowオブジェクト

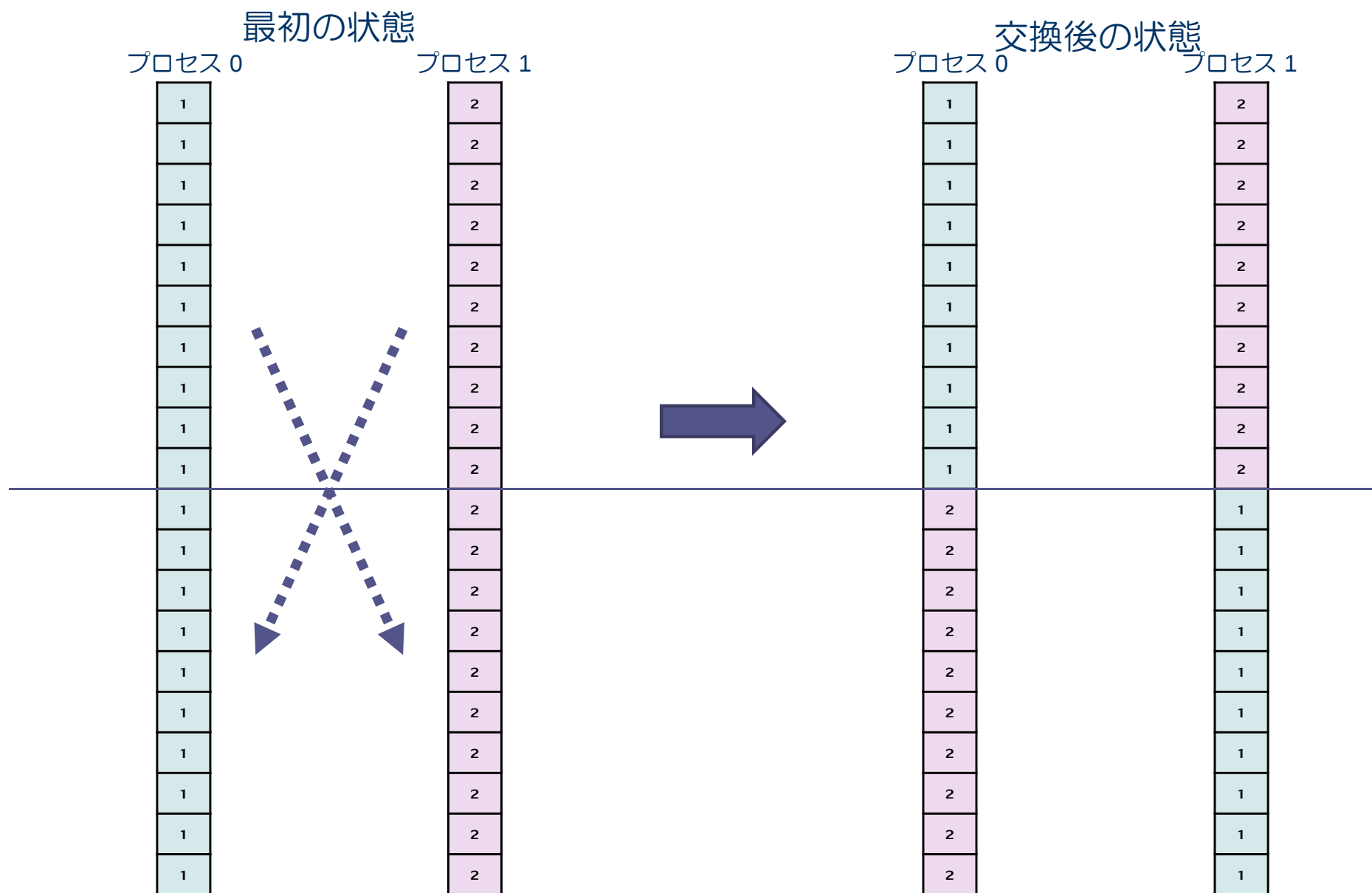
[Output]

◆ `ierr`: 戻りコード（整数型）

問題：次のプログラムを作れ

- 2つのMPIプロセスにおいて,
 - ◆ step 1: 長さ $2n$ の配列を用意する. どんな型 (整数型や実数型) でも良い.
 - ◆ step 2: 最初の状態として, プロセス0では配列に1を代入. プロセス1では配列に2を代入.
 - ◆ step 3: プロセス0の配列の前半部分 (長さ n) をプロセス1の配列の後半 (長さ n) にコピーし, プロセス1の配列の前半部分 (長さ n) をプロセス0の配列の後半 (長さ n) にコピーする.
- この作業について, send-recvの組合せと, put-getの組合せの2つのプログラムを作り, 結果を確認する.
- $n=10$ くらいでやると, 出力をすることにより動作を確認できる.

プログラムの処理イメージ



プログラム・スケルトン

send-recv

```
# ヘッダ

# mpiの初期化

# 配列を準備


if( myrank == 0 ) then

    rank = 0 の処理 (sendして, recvする)

else

    rank = 1 の処理 (recvして, sendする)

endif


# 結果の確認

# MPIの終了
```

put-get

```
# ヘッダ

# mpiの初期化

# 配列を準備

# put, get用のWindowをセット

# プロセス0側だけから操作

if( myrank == 0 ) then

    プロセス0の配列の前半部分を, プロセス1の配列の後半部分に書き込む

    プロセス1の配列の前半部分をもらい, プロセス0の配列の後半部分に書き込む

endif


# put, getの同期待ち. MPI_Win_fence

# 結果の確認

# MPIの終了
```

参考：send関数, receive関数

【復習】 1対1通信 – 送信関数（送り出し側）【C】

```
int MPI_Send(void *buff, int count, MPI_Datatype datatype, int dest,  
             int tag, MPI_Comm comm)
```

※ ランク番号destのプロセスに、変数buffの値を送信する。

- ◆ buff: 送信するデータの変数名（先頭アドレス）
- ◆ count: 送信するデータの数（整数型）
- ◆ datatype: 送信するデータの型
 - MPI_CHAR, MPI_INT, MPI_DOUBLEなど
- ◆ dest: 送信先プロセスのランク番号
- ◆ tag: メッセージ識別番号. 送るデータを区別するための番号
- ◆ comm: コミュニケータ（例えば, MPI_COMM_WORLD）
- ◆ 関数の戻りコードは, エラーコード

【復習】 1対1通信 – 送信関数（送り出し側） 【Fortran】

```
mpi_send( buff, count, datatype, dest, tag, comm, ierr )
```

- ◆ buff: 送信するデータの変数名（先頭アドレス）
- ◆ count: 送信するデータの数（整数型）
- ◆ datatype: 送信するデータの型
 - MPI_INTEGER, MPI_DOUBLE_PRECISION, MPI_CHARACTER など
- ◆ dest: 送信先のプロセス番号
- ◆ tag: メッセージ識別番号. 送るデータを区別するための番号
- ◆ comm: コミュニケータ（例えば, MPI_COMM_WORLD）
- ◆ ierr: 戻りコード（整数型）

【復習】 1対1通信 – 受信関数（受け取り側） 【C】

```
int MPI_Recv(void *buff, int count, MPI_Datatype datatype, int source,  
             int tag, MPI_Comm comm, MPI_Status *status)
```

※ ランク番号sourceのプロセスから送られたデータを、変数buffに格納する。

- ◆ buff: 受信するデータのための変数名（先頭アドレス）
- ◆ count: 受信するデータの数（整数型）
- ◆ datatype: 受信するデータの型
 - MPI_CHAR, MPI_INT, MPI_DOUBLEなど
- ◆ source: 送信してくる相手プロセスのランク番号
- ◆ tag: メッセージ識別番号。送られて来たデータを区別するための番号
- ◆ comm: コミュニケータ（例えば, MPI_COMM_WORLD）
- ◆ status: 通信の状態を格納するオブジェクト。MPI_Sendにはないので注意。
- ◆ 関数の戻りコードは、エラーコード

【復習】 1対1通信 – 受信関数（受け取り側） 【Fortran】

```
mpi_recv( buff, count, datatype, source, tag, comm, status, ierr )
```

- ◆ buff: 受信するデータのための変数名（先頭アドレス）
- ◆ count: 受信するデータの数（整数型）
- ◆ datatype: 受信するデータの型
 - MPI_INTEGER, MPI_DOUBLE_PRECISION, MPI_CHARACTER など
- ◆ source: 送信してくる相手のプロセス番号
- ◆ tag: メッセージ識別番号. 送られて来たデータを区別するための番号
- ◆ comm: コミュニケータ（例えば, MPI_COMM_WORLD）
- ◆ status: 受信の状態を格納するサイズMPI_STATUS_SIZEの配列（整数型）
- ◆ ierr: 戻りコード（整数型）