

MPI（片方向通信）

2015年3月6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

講義の内容

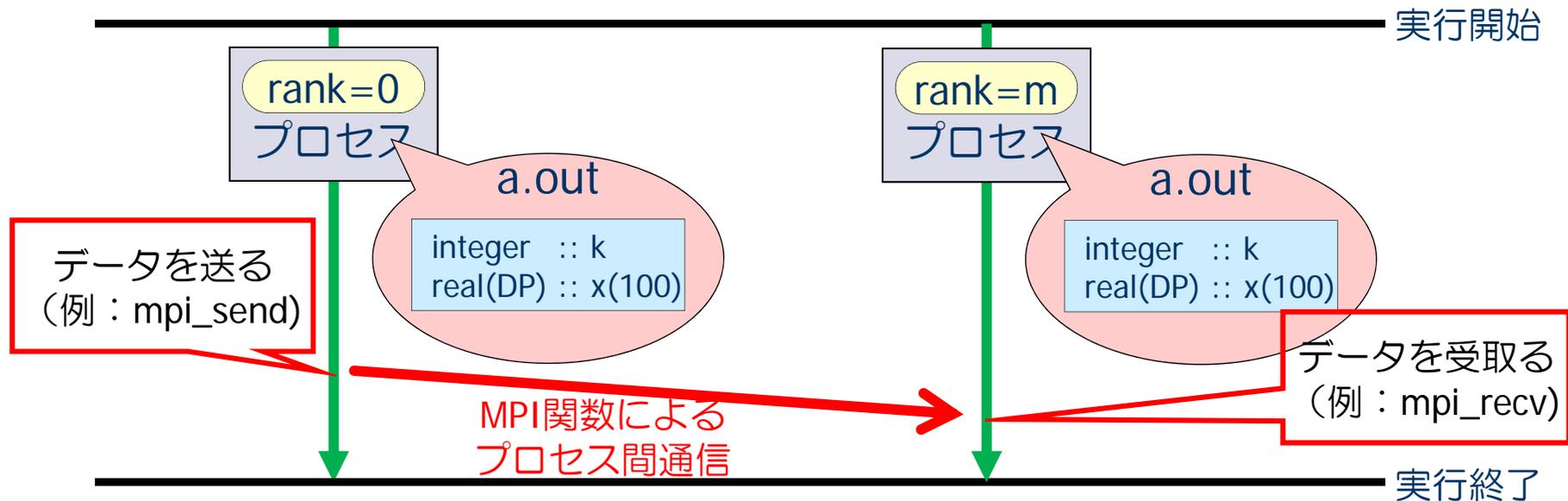
- メッセージ・パッシング・インターフェイス (MPI)
- 双方向通信
- 片方向通信
- 演習問題

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ー、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，対応する関数を呼び出す。
- MPI_send, MPI_recvなどの組合せ



【復習】 1対1通信 - 送信関数 (送り出し側)

```
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)

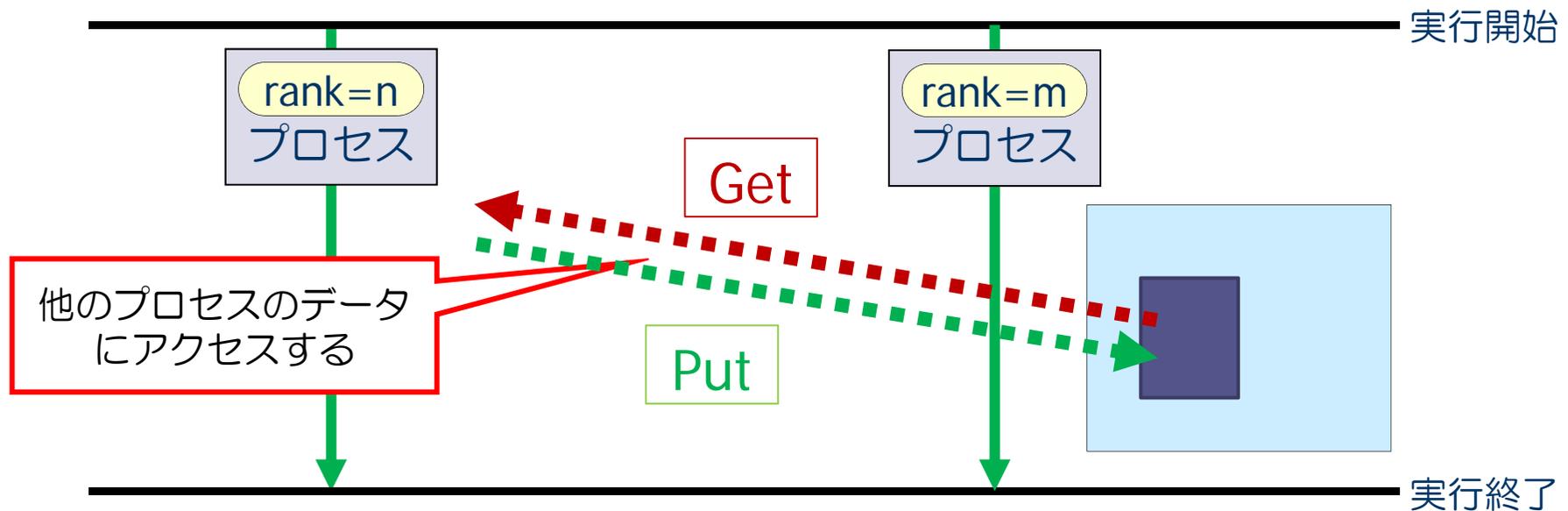
【復習】 1対1通信 - 受信関数 (受け取り側)

```
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)

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

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

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

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.
- ◆ info: 情報 (整数型)
 - ◆ Cの場合, MPI_INFO_NULL
- ◆ comm: コミュニケータ (例えば, MPI_COMM_WORLD)

[Output]

- ◆ win: Windowオブジェクト
 - MPI_getなどで利用
- ◆ ierr: 戻りコード (整数型)

Windowオブジェクトの開放

```
【F】 mpi_win_free( win, ierr )
```

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

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

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

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

[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 (整数型)
- ◆ 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, $base + tdisp \times disp_unit$
 - ◆ 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)
- ◆ 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

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

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

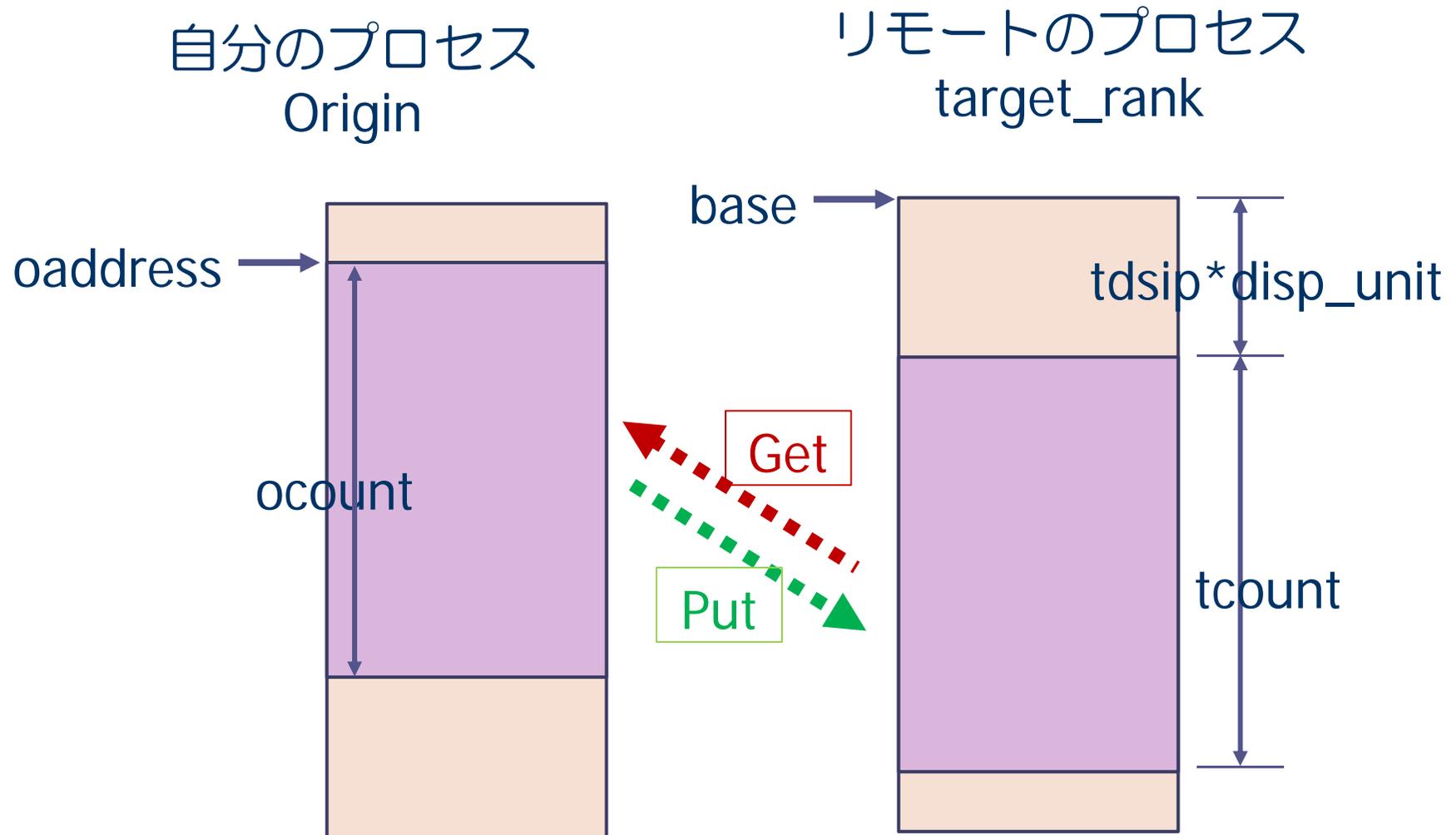
[Input/Output]

- ◆ 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 (整数型)
- ◆ 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, $base + tdisp \times disp_unit$
 - ◆ 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)
- ◆ 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

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

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

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

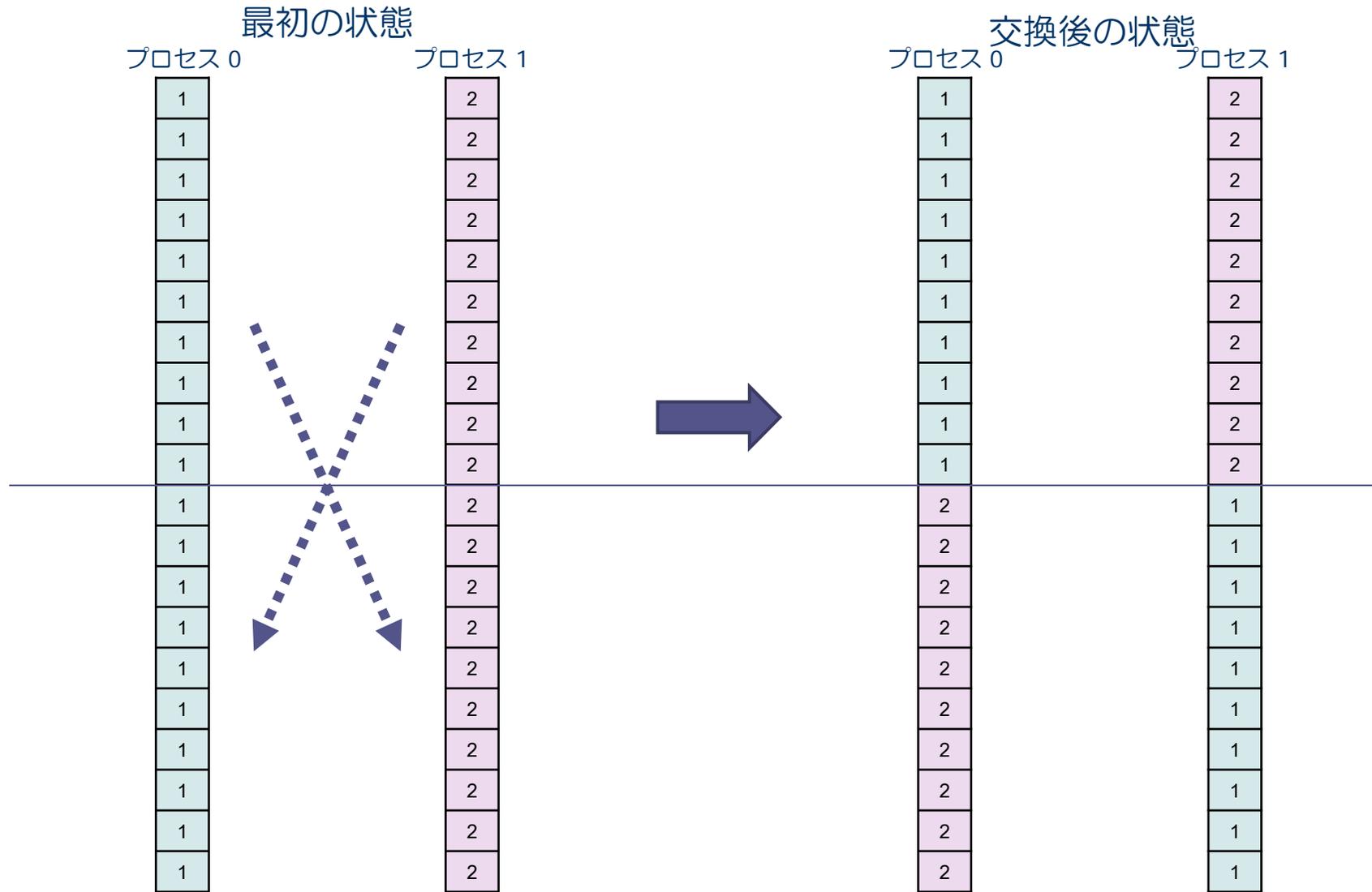
[Output]

- ◆ 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で確認できる。

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：C言語のsend, receive

【復習】 1対1通信 - 送信関数 (送り出し側)

```
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)

```
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)