コミュニケータとデータタイプ
（Communicator and Datatype）

2019年3月15日

神戸大学大学院システム情報学研究科
横川三津夫

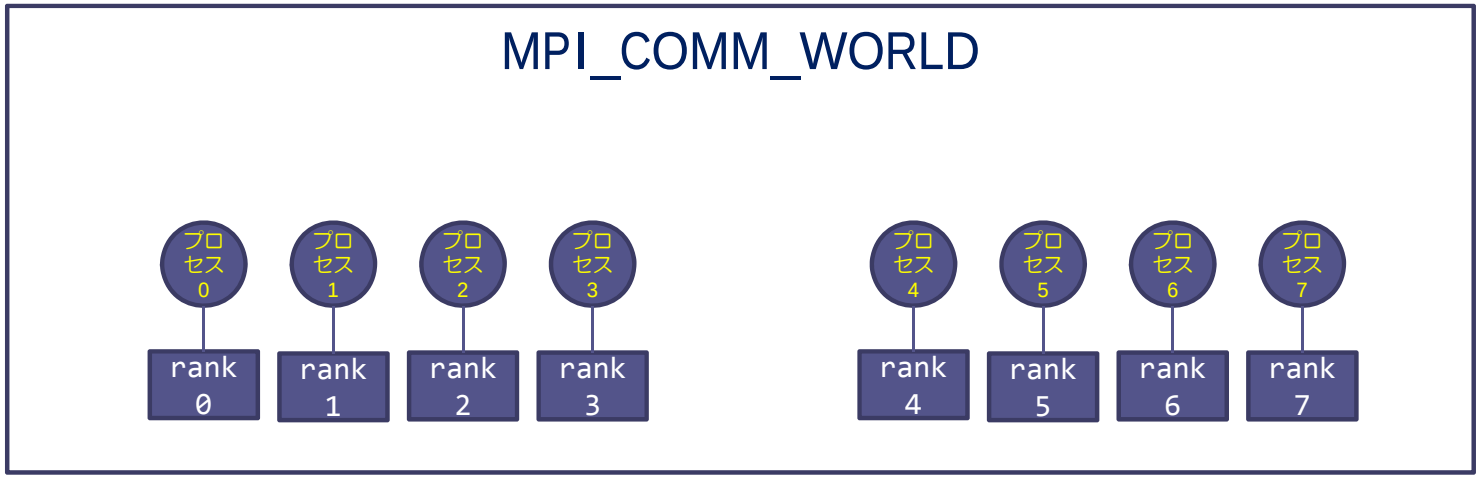
講義の内容

- コミュニケータ（Communicator）
- データタイプ（Datatype）
- 演習問題

コミュニケータ（Communicator）1/2

- MPIにおけるプロセスの集団（集合）
- 集団的な操作などにおける操作対象
  - 集団型通信（Collective communication）：通信がコミュニケータ内に閉じる.
  - 集団型I/O（Collective operation）
  - Window操作（One-sided communication）

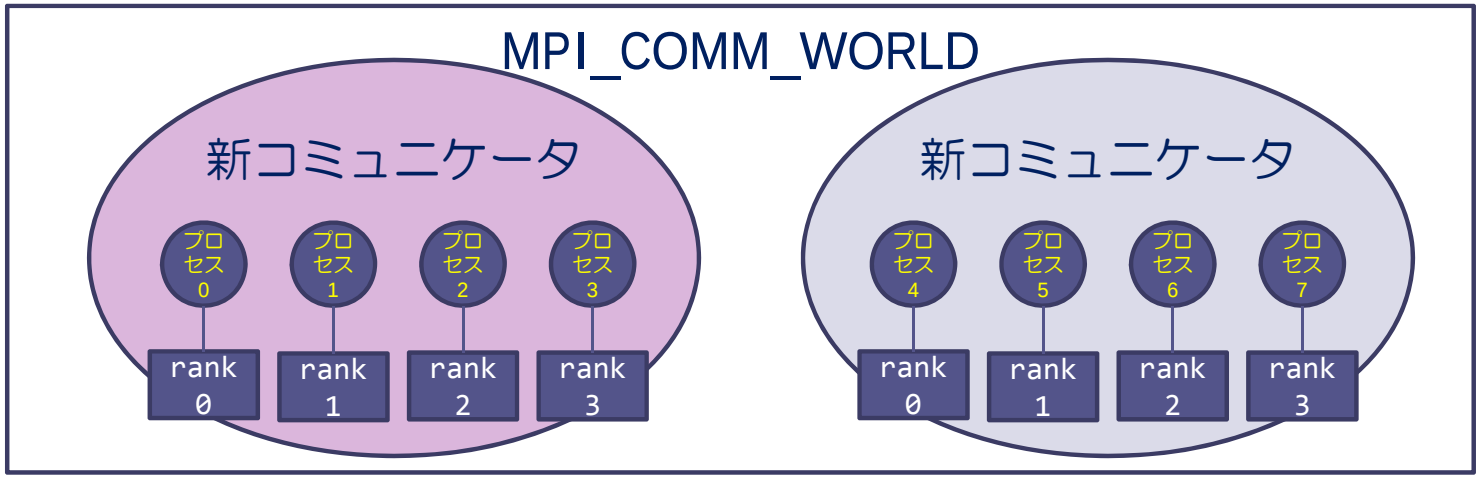
- MPI_COMM_WORLD:　MPIプログラム起動後に最初に作られるプロセス全体のコミュニケータ



コミュニケータ（Communicator）2/2

- MPIにおけるプロセスの集団（集合）
- 集団的な操作などにおける操作対象
  - 集団型通信（Collective communication）：通信がコミュニケータ内に閉じる.
  - 集団型I/O（Collective operation）
  - Window操作（One-sided communication）

- コミュニケータを分割:　新しいランク番号が割り当てられる.



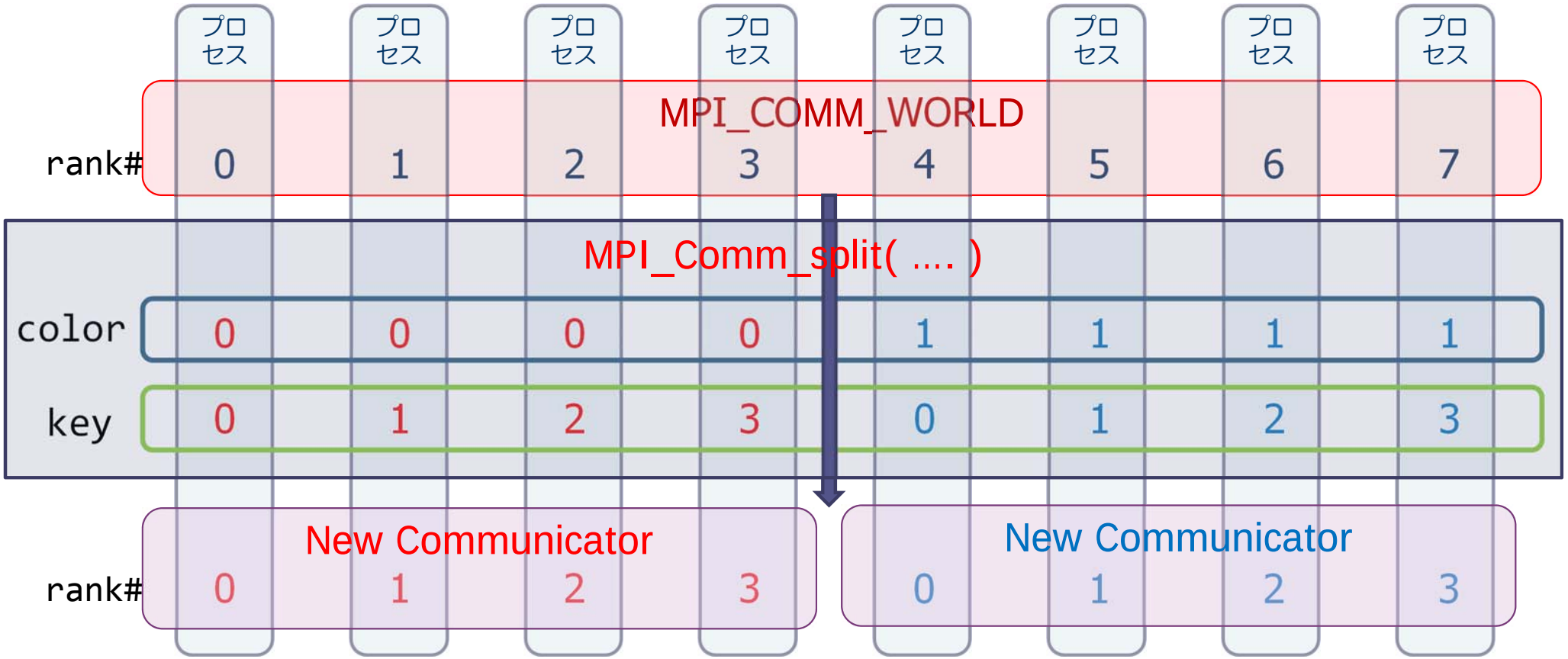
新しいコミュニケータを作る（分割）

```
C: int MPI_Comm_split( MPI_Comm comm, int color, int key, MPI_Comm *new_comm)
F: MPI_COMM_SPLIT( comm, color, key, new_comm, ierr )
```

- comm: 元々のコミュニケータ
- color: 生成する複数のコミュニケータを識別する変数（正の整数）
  同じ値を持つプロセスが一つのコミュニケータを構成.
- key: 同じコミュニケータ内のランク番号を指定（整数型）
  値が同じ場合には，親コミュニケータの順番が維持される.
- new_comm: 新しいコミュニケータ
  自分が属するサブグループで同じコミュニケータとなる.
- ierr: 戻りコード（整数型）

- コミュニケータを分割し新たな複数のコミュニケータを生成
- 同じcolorを持つプロセス群で（同じ名前の）新しいコミュニケータを生成
- keyの値の順番で，同一コミュニケータ内のランク順が決まる.
  （keyの値が同じ場合は，システムが適当に決める．）
- 各プロセスで同時に実行（SPMD）

コミュニケータを2つに分割（図解）



例えば，MPI_COMM_WORLDを2つのコミュニケータ newcomに分割するには．．．

```
color = rank# / 4;
key   = rank# % 4;
MPI_Comm_split( MPI_COMM_WORLD, color, key, &newcomm );
```

コミュニケータの複製，コミュニケータの解放

```
C: int MPI_Comm_dup( MPI_Comm comm, MPI_Comm *new_comm );
F: MPI_COMM_DUP( comm, new_comm, ierr )
```

- comm:　　コミュニケータ
- new_comm:　複製のコミュニケータ
- ierr:　　戻りコード（整数型）

- 親コミュニケータと同じプロセスグループの複製を生成
- 元のコミュニケータと，生成されたコミュニケータで通信を分離

```
C: int MPI_Comm_free( MPI_Comm *comm )
F: MPI_COMM_FREE( comm, ierr )
```

- comm:　　解放したいコミュニケータ
- ierr:　　戻りコード（整数型）

- 指定されたコミュニケータを解放
- この関数の呼び出し以降は，当該コミュニケータは使用不能

【参考】 ランクの取得，ランク数の取得

```
C: int MPI_Comm_rank( MPI_Comm comm, int *rank );
F: MPI_COMM_RANK( comm, rank, ierr )
```

- comm: コミュニケータ
- rank: 指定されたコミュニケータにおける自プロセスのランク番号を取得（整数型）
- ierr: 戻りコード（整数型）

```
C: int MPI_Comm_size( MPI_Comm comm, int *size );
F: MPI_COMM_SIZE( comm, size, ierr )
```

- comm: コミュニケータ
- size: 指定されたコミュニケータに属するランク数を取得（整数型）
- ierr: 戻りコード（整数型）

MPIで扱うデータ型

- 基本データ型
  - 整数型，文字型，実数型などの基本となるデータ型
    - C: MPI_CHAR, MPI_INT, MPI_DOUBLEなど
    - Fortran: MPI_INTEGER, MPI_REAL, MPI_DOUBLE_PRECISION

- 派生データ型（Derived Datatype）
  - 基本データ型の組合せにより，新たに生成されるデータ型
    - 不連続なデータレイアウト（メモリ内配置）をまとめたデータ型
    - 不連続なレイアウトにあるデータ群を1回の通信で処理可能

  - 派生データ型生成を行う関数により生成可能
    - 基本データ型とそのオフセットの集合で表現される.

派生データ型の生成，登録，解放をする関数群

- 生成（一部）：
  - MPI_Type_contiguous
  - MPI_Type_vector
  - MPI_Type_indexed
  - MPI_Type_create_idexed
  - MPI_Type_create_subarray
  - MPI_Type_create_darray
- 登録：生成した派生データ型は登録しなければならない．
  - MPI_Type_commit
- 解放：不要になった派生データ型を解放する．
  - MPI_Type_free

新しい派生データ型の登録，派生データ型の解放

```
C: int MPI_Type_commit( MPI_Datatype type) ;
F: MPI_TYPE_COMMIT( type, ierr )
```

- type:　　作成したデータ型
- ierr:　　戻りコード（整数型）

- 必ずこの関数を用いて登録する必要がある．
- この関数を呼び出した後は，この関数で登録したデータ型を用いた通信，I/Oなどで利用可能．

```
C: int MPI_Type_free( MPI_Datatype *type) ;
F: MPI_TYPE_FREE( type, ierr )
```

- type:　　開放したい派生データ型
- ierr:　　戻りコード（整数型）

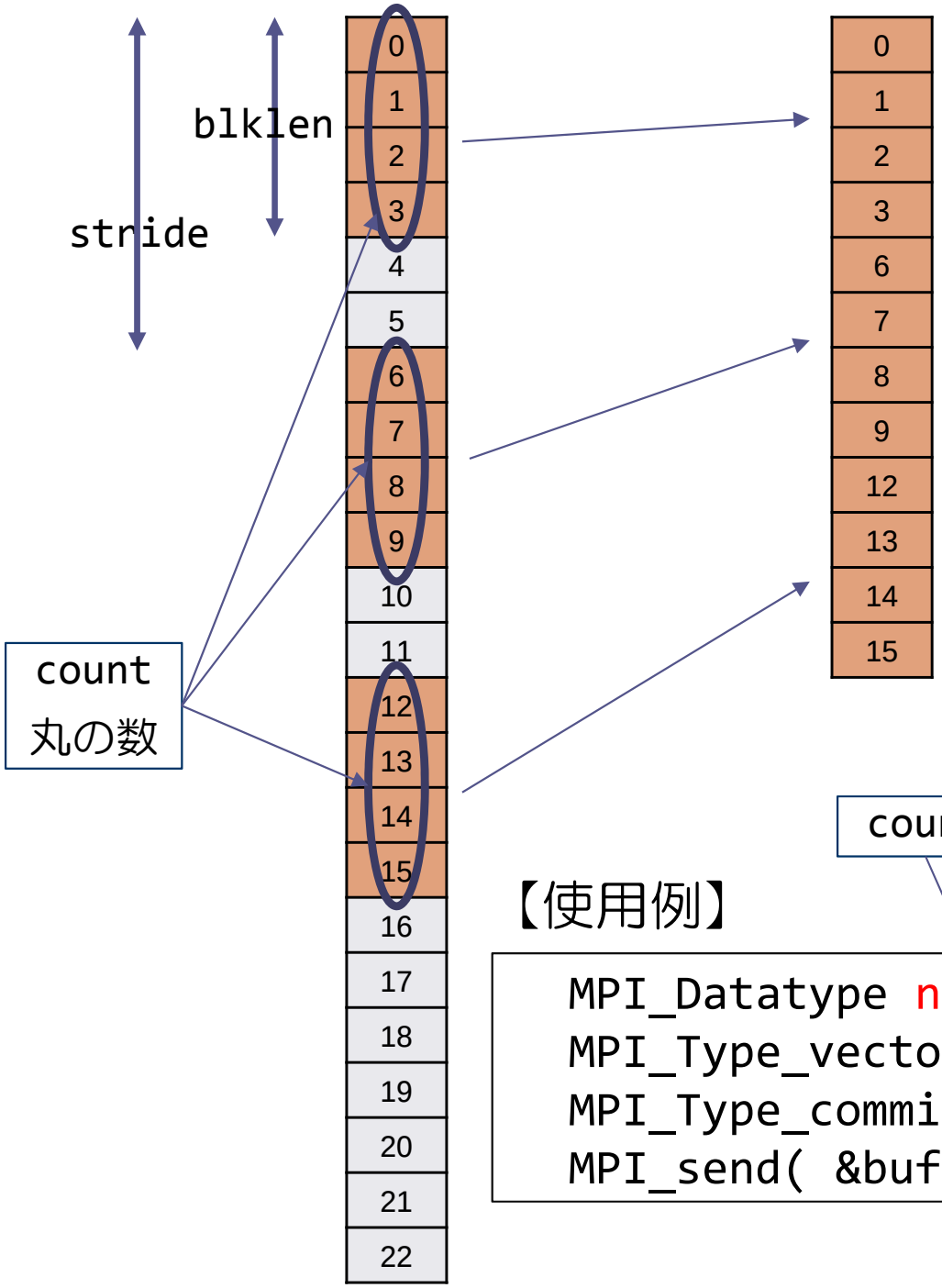
- 必要なくなった派生データ型を解放．
- この関数を呼び出した後は，解放した派生データ型は使えない．

派生データ型の関数（MPI_Type_vector）

```
C: int MPI_Type_vector( int count, int blklen, int stride,
                        MPI_Datatype otype, MPI_Datatype *ntype) ;
F: MPI_TYPE_VECTOR( count, blklen, stride, otype, ntype, ierr )
```

- count:　ブロックの数（非負整数）
- blklen:　ブロック内の要素数（非負整数）
- stride:　ブロック間の要素数（整数）
- otype:　元々のデータタイプ
- ntype:　新しいデータタイプ名
- ierr:　戻りコード（整数型）

- 親コミュニケータと同じプロセスグループで，異なる名前のコミュニケータを生成
- 親コミュニケータと生成されたコミュニケータそれぞれで通信を分離することができる.



- 飛び飛びのデータを連続なデータとして扱えるようになる.
- 派生データ型 1個として扱える.

count blklen stride

【使用例】

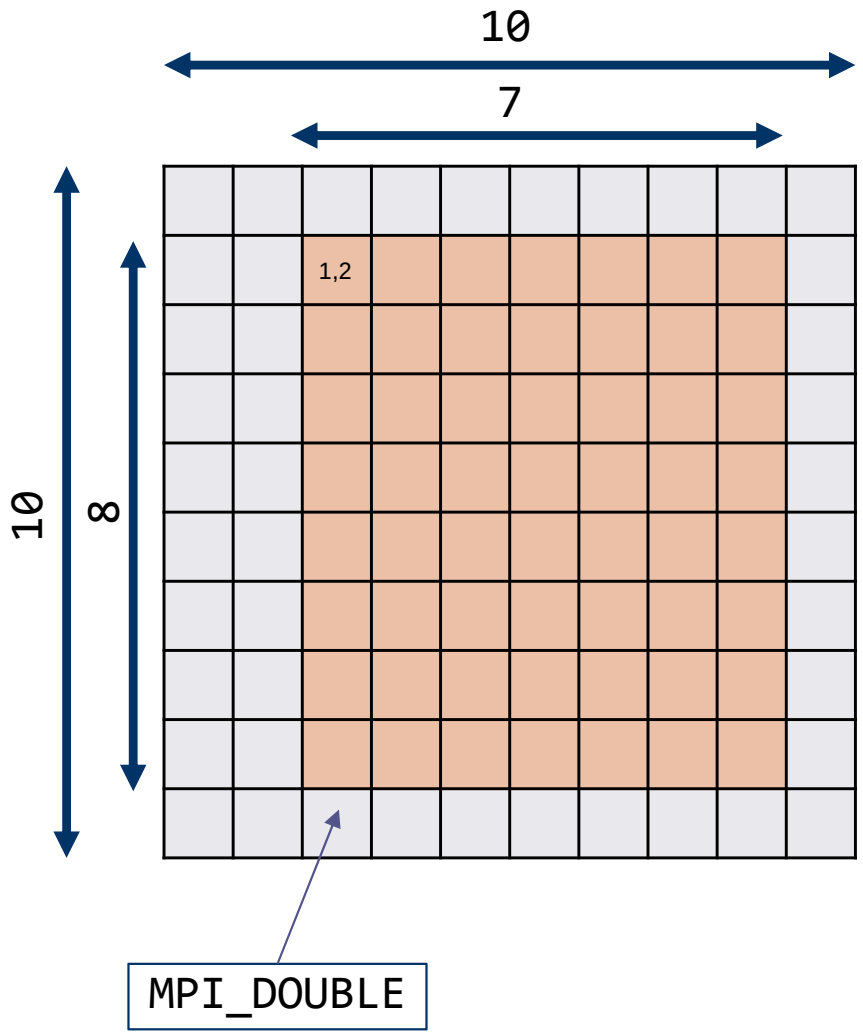
```
MPI_Datatype newtype;
MPI_Type_vector(3, 4, 6, MPI_DOUBLE, &newtype);
MPI_Type_commit( &newtype );
MPI_send( &buf, 1, newtype, dest, tag, COMM );
```

派生データ型の関数（MPI_Type_create_subarray）

```
C: int MPI_Type_create_subarray( int ndims, int array_of_sizes[], int
   array_of_subsizes[], int array_of_starts[], int order, MPI_Datatype
   otype, MPI_Datatype *ntype );

F: MPI_TYPE_CREATE_SUBARRAY( ndims, array_of_sizes, array_of_subsizes,
   array_of_starts, order, otype, ntype, ierr )
```

- ndims: ベースになる配列の次元数
- array_of_sizes: ベースになる配列の各次元の大きさ（サイズはndims）
- array_of_subsizes: 部分配列の大きさ（サイズはndims）
- array_of starts: 部分配列の開始地点（サイズはndims）
  【注意】Fortranでも 0から始まる.
- order: MPI_ORDER_C, またはMPI_ORDER_FORTRAN
- otype: 元々のデータタイプ
- ntype: 新しいデータタイプ名
- ierr: 戻りコード（整数型）

【例】 2次元データの内部（赤い部分）をアクセスする派生データ型の作成

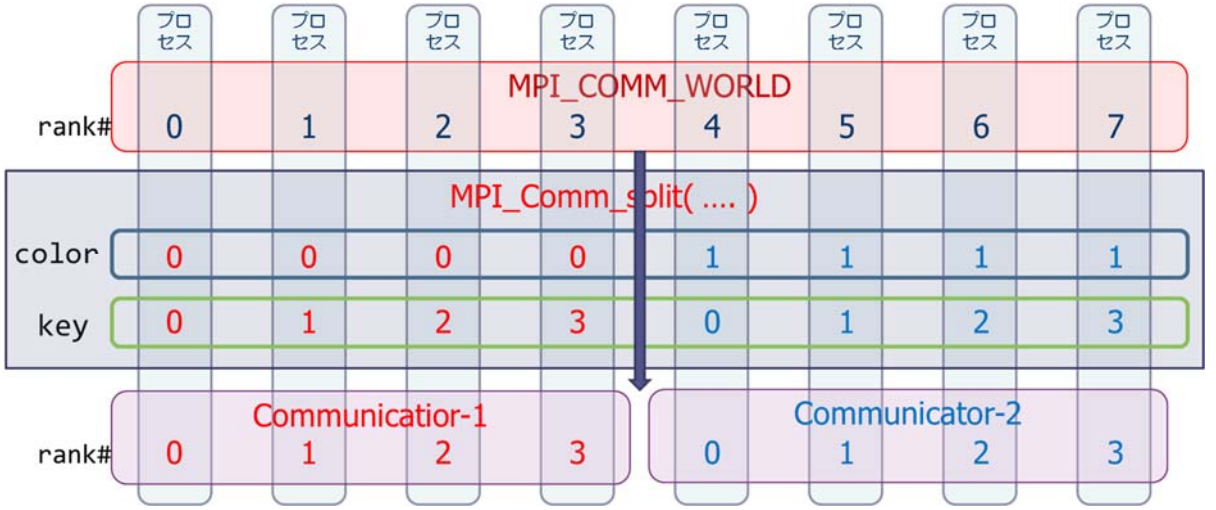



```
int ndims;
int o_sizes[2];
int n_sizes[2];
int starts[2];
MPI_Datatype ntype;

ndims = 2
o_sizes[0] = 10;
o_sizes[1] = 10;
n_sizes[0] = 7;
n_sizes[1] = 8;
starts[0]  = 2;
starts[1]  = 1;

MPI_Type_create_subarray(ndims, o_sizes,
n_sizes, starts, MPI_ORDER_C, MPI_DOUBLE,
&ntype);
```

問題：次のプログラムを作れ

- 16プロセスのMPIプログラムにおいて，起動時のコミュニケータMPI_COMM_WORLDを，2つのコミュニケータに分割せよ．また，4つのコミュニケータに分割せよ．
  - 複数のコミュニケータが出来ていることを，各プロセスの新しいランク番号を出力して確認せよ．
  - また，同時放送（MPI_Bcast）が新コミュニケータ内に閉じていることを確認せよ．

【例】8 MPIプロセスのプログラムを2つのコミュニケータに分割



Skeltonのディレクトリ，ファイル

- /tmp/Comm-Skelton
  - Cの場合
    - new_comm_skelton.c　　スケルトン
  - Fortranの場合
    - new_comm_skelton.f90　　スケルトン
    - job.sh　　ジョブスクリプト・サンプル

参考

メッセージ・パシング・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
      - http://www.mpi-forum.org/docs/mpi-3.0/mpi30-report.pdf

MPIプログラムのスケルトン（C）

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

int main(int argc, char **argv)
{
  int  nprocs, myrank;

  MPI_Init( &argc, &argv );
  MPI_Comm_size( MPI_COMM_WORLD, &nprocs );
  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

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

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（後出）．

MPIプログラムの基本構成（説明：C）

- `int MPI_Init( int *argc, char ***argv )`
  MPIの初期化を行う．MPIプログラムの最初に必ず書く．

- `int MPI_Comm_size( MPI_Comm comm, int *size )`
  - MPIの全プロセス数を取得し，2番目の引数 nprocs（整数型）に取得する．
  - MPI_COMM_WORLDはコミュニケータと呼ばれ，最初に割り当てられるすべてのプロセスの集合

- `int MPI_Comm_rank( MPI_Comm comm, int *rank )`
  自分のプロセス番号（0からnprocs-1のどれか）を，2番目の引数 myrank（整数型）に取得する．

- `int MPI_Finalize( void )`
  MPIの終了処理をする．MPIプログラムの最後に必ず書く．

MPIプログラムのスケルトン（Fortran）

```
program main
use mpi                                                  MPIモジュールの取り込み（おまじない1）
implicit none
integer :: nprocs, myrank, ierr                          MPIで使う変数の宣言

call mpi_init( ierr )                                    MPIの初期化（おまじない2）
call mpi_comm_size( MPI_COMM_WORLD, nprocs, ierr )       MPIで使うプロセス数を nprocs に取得
call mpi_comm_rank( MPI_COMM_WORLD, myrank, ierr )       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.

MPIプログラムの基本構成（説明）

- `call mpi_init( ierr )`
  - MPIの初期化を行う．MPIプログラムの最初に必ず書く．
- `call mpi_comm_size( MPI_COMM_WORLD, nprocs, ierr )`
  - MPIの全プロセス数を取得し，2番目の引数 nprocs（整数型）に取得する．
  - MPI_COMM_WORLDはコミュニケータと呼ばれ，最初に割り当てられるすべてのプロセスの集合
- `call mpi_comm_rank( MPI_COMM_WORLD, myrank, ierr )`
  - 自分のプロセス番号（0からnprocs-1のどれか）を，2番目の引数 myrank（整数型）に取得する．
- `call mpi_finalize( ierr )`
  - MPIの終了処理をする．MPIプログラムの最後に必ず書く．