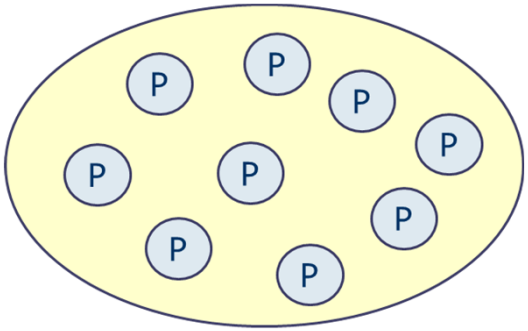
KOBE HPC サマースクール 2019（初級）

10. 分散メモリ型並列計算機とは何か？

- SPMDプログラミングによるHello World! -

並列計算機のアーキテクチャ

- 並列計算機
  - 複数のプロセッサ（ノイマン・アーキテクチャ）が，何らかの方法で接続されいて，協調して動作する計算機システム

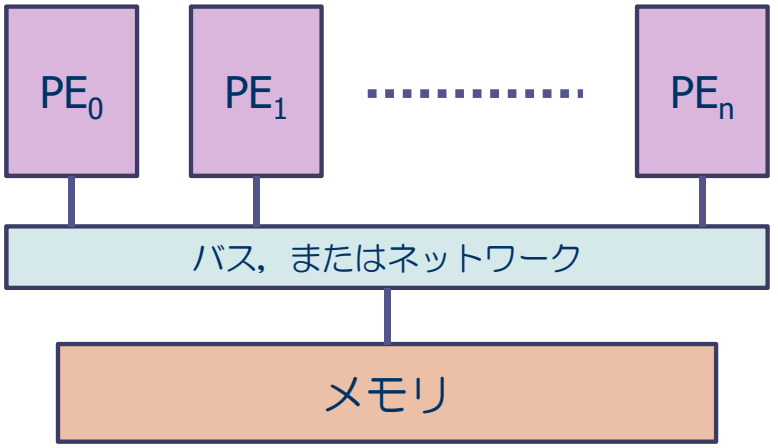


プロセッサ（P），プロセッサエレメント（PE），プロセッサユニット（PU），ノード（Node），コア（core）

- タイプ
  - SIMDとMIMD（Flynnの分類, 1966年）
    - 命令実行の流れとデータの流れに着目した分類方法
  - 共有メモリ型と分散メモリ型
    - メモリ空間に着目した分類方法
  - ホモジニアス型とヘテロジニアス型
    - ハードウェアの均質性に着目した分類方法

共有メモリ型並列計算機

- 複数のプロセッサ（PE）が，単一のメモリ空間を共有
  - どのPEも同じメモリ領域にアクセス可能

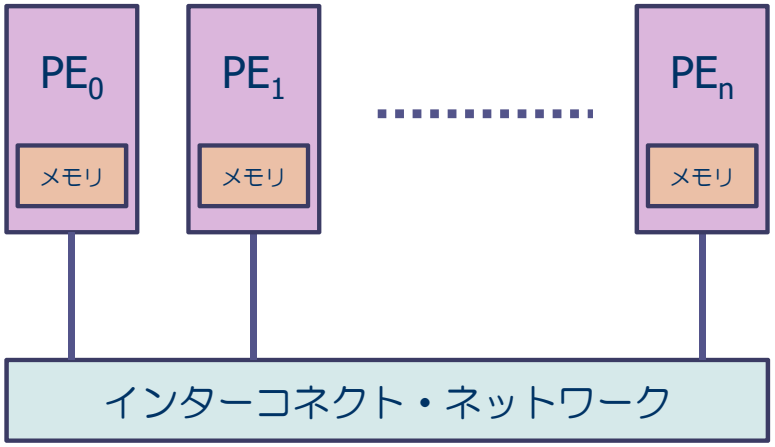


- 特徴
  - メモリ空間が単一なので，プログラミングが容易
  - PEの数が多いと，同一メモリアドレスへのアクセス競合が生じ，性能が低下

- プログラミング技術
  - OpenMP，自動並列化
  - ただし，MPIで実行することも可能

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，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.

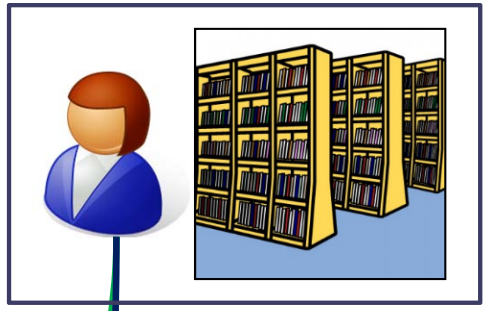
- プログラミング技術
  - メッセージ・パシング・インターフェイス（MPI）によるプログラミング

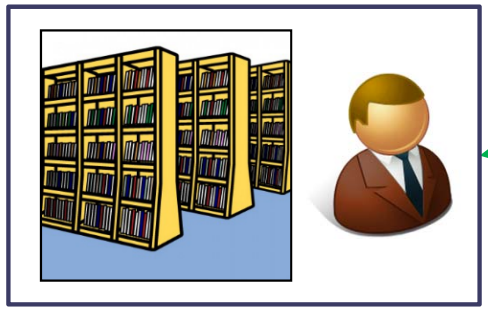
メッセージ・パシング・インターフェイス

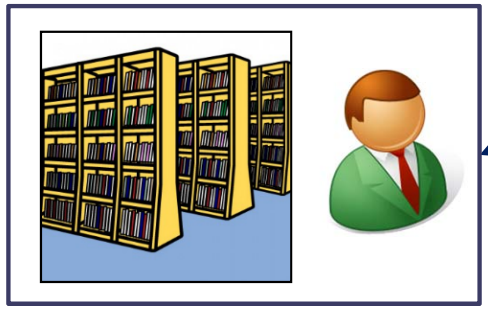
- Message Passing Interface (MPI) とは．．．
  - 複数の独立したプロセス間で，並列処理を行うためのプロセス間メッセージ通信の標準規格
  - 1992年頃より米国の計算機メーカ，大学などを中心に標準化
  - MPI規格化の歴史
    - 1994　MPI-1.0
    - 1997　MPI-2.0（一方向通信など）
    - 2012　MPI-3.0
      - http://www.mpi-forum.org/docs/mpi-3.0/mpi30-report.pdf

MPIプログラミング（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．





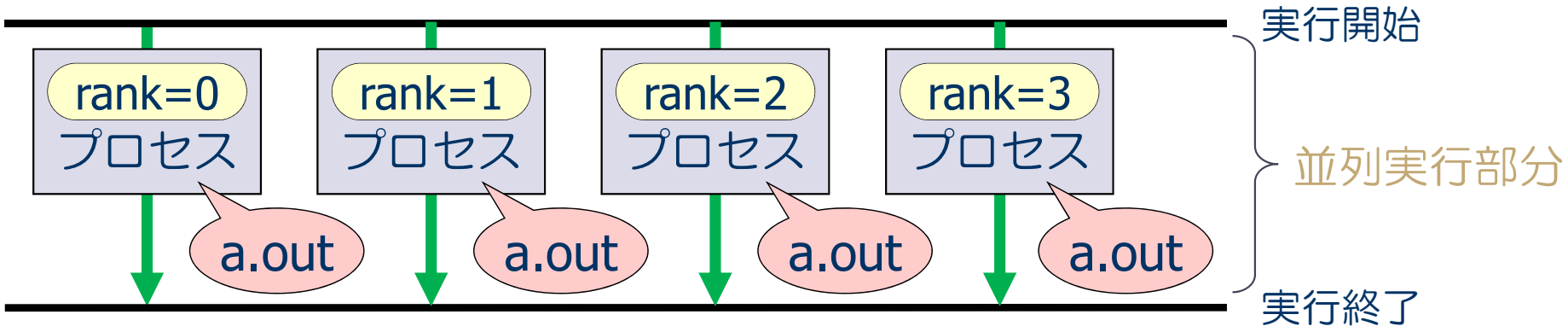


【必要な情報】
- ノートの名前
- 何冊
- 誰に
- 荷物のタグなど

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

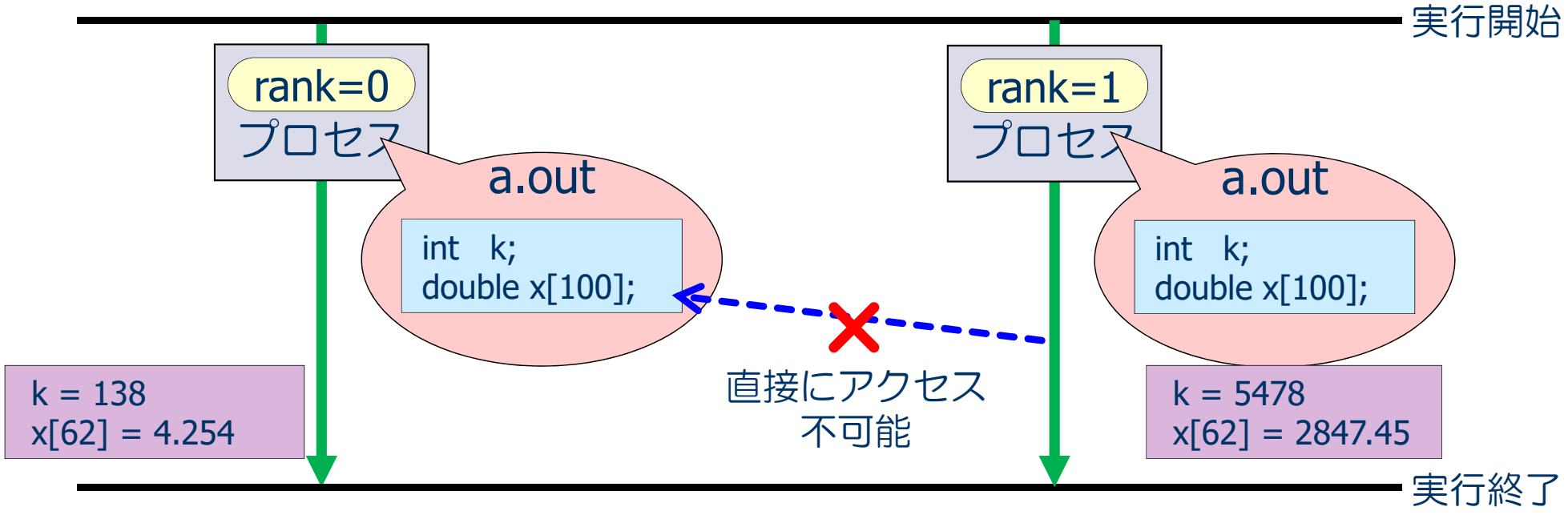
- 複数のプロセスにより並列実行
- 実行開始から終了まで，全プロセスが同じプログラムを実行
- 各MPIプロセスは固有の番号（ランク番号）を持つ
  - P個のプロセスで実行する場合，プロセス番号は0から(P-1)までの整数

- 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.



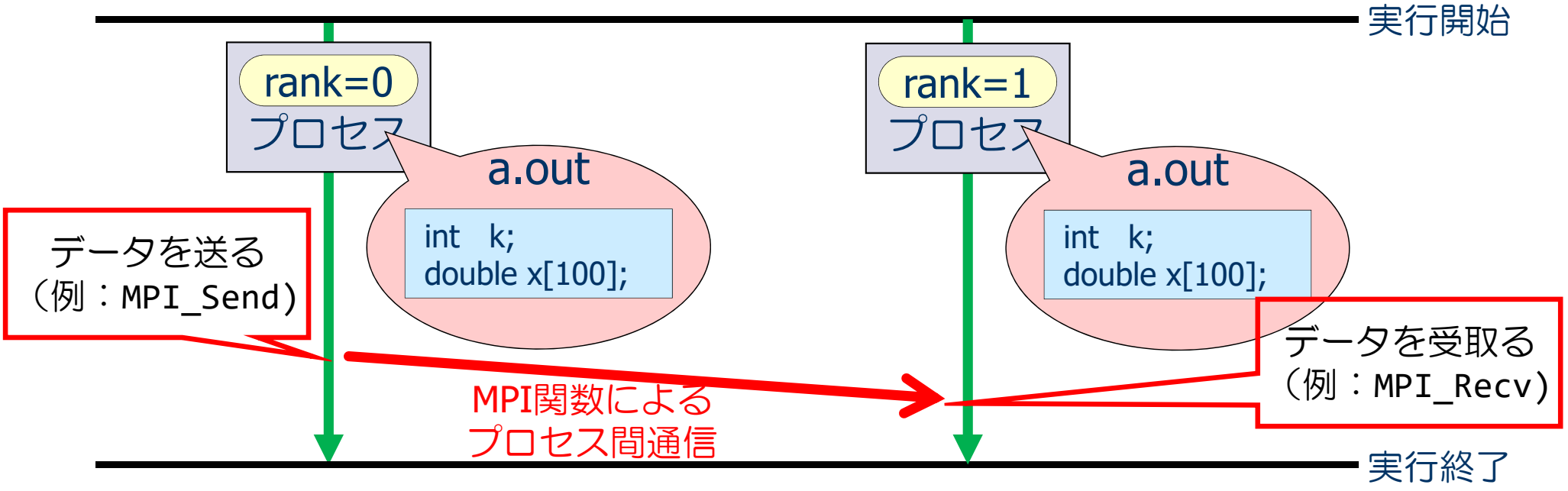
MPIの実行モデル（続き）

- 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



MPIプログラムのスケルトン

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

  MPI_Finalize() ;                                        MPIの終了処理（おまじない3）
  return 0 ;
}
```

- 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）

- `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 *nprocs )`
  - 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 *myrank )`
  - 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（M-1）： Hello, world!

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

int main( )
{
  int myrank, nprocs ;

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

  printf (“Hello, World! My rank number and nprocs are %d and %d.¥n", myrank,
nprocs );

  MPI_Finalize();

  return 0;
}
```

SGI版 MPIを使う手順

- 環境設定

```
$  module load intel
$  module load mpt
```

- コンパイル方法

```
$  icc prog.c -lmpi                                  ：prog.cはコンパイルしたいプログラムのファイル名
  （アイ シー シー  prog.c ハイフン(-) エル エム ピー アイ）
```

- バッチジョブ・スクリプト

```
#!/bin/bash
#PBS -q S
#PBS -l select=1:ncpus=4:mpiprocs=4 ← select:ノード数，ncpus:ノード内コア数
                                        MPI実行の場合はncpus と mpiprocs の値は同じにする．
#PBS -N hello  ←――――――――――――――― helloは，出力ファイルヘッダ
#PBS -j oe

source /etc/profile.d/modules.sh
module load intel
module load mpt
cd ${PBS_O_WORKDIR}
mpiexec_mpt dplace -s1 ./a.out
```

演習10-1：Hello, world! を並列に出力する.

- MPI版 “Hello, world!” を 2，4，及び8プロセスで実行し，結果を確認せよ！ 以下は実行例.

$ mkdir␣MPI $ cd␣MPI	今日の演習用のディレクトリを作成する. ※ ␣はスペースを表わす.
$ mkdir␣M-1 $ cd␣M-1	演習M-1用のディレクトリを作成する.
$ cp␣/tmp/Summer/M-1/hello_mpi.c␣./	ソースプログラム hello_mpi.c をカレントディレクトリにコピーする. ※ 中身を見て確認すること.
$ icc␣hello_mpi.c␣-lmpi	ソースプログラムをコンパイル
$ cp␣/tmp/Summer/M-1/go.sh␣./	ジョブスクリプト go.sh をコピーする. （プロセス数の指定など，必要な部分をeditする）
$ qsub␣go.sh nnnnnn.rokko1	ジョブを投入し，実行結果を確認する. ※ nnnnnnはジョブ番号
$ cat␣hello.o*nnnnn*	※ helloは，go.sh内で指定した jobname のこと

スペース

プログラム M-1の実行結果の確認

- ２プロセスでの実行結果

```
Hello, world!  My rank number and nprocs are 0, 2.
Hello, world!  My rank number and nprocs are 1, 2.
```

- ４プロセスでの実行結果

```
Hello, world!  My rank number and nprocs are 2, 4.
Hello, world!  My rank number and nprocs are 0, 4.
Hello, world!  My rank number and nprocs are 3, 4.
Hello, world!  My rank number and nprocs are 1, 4.
```

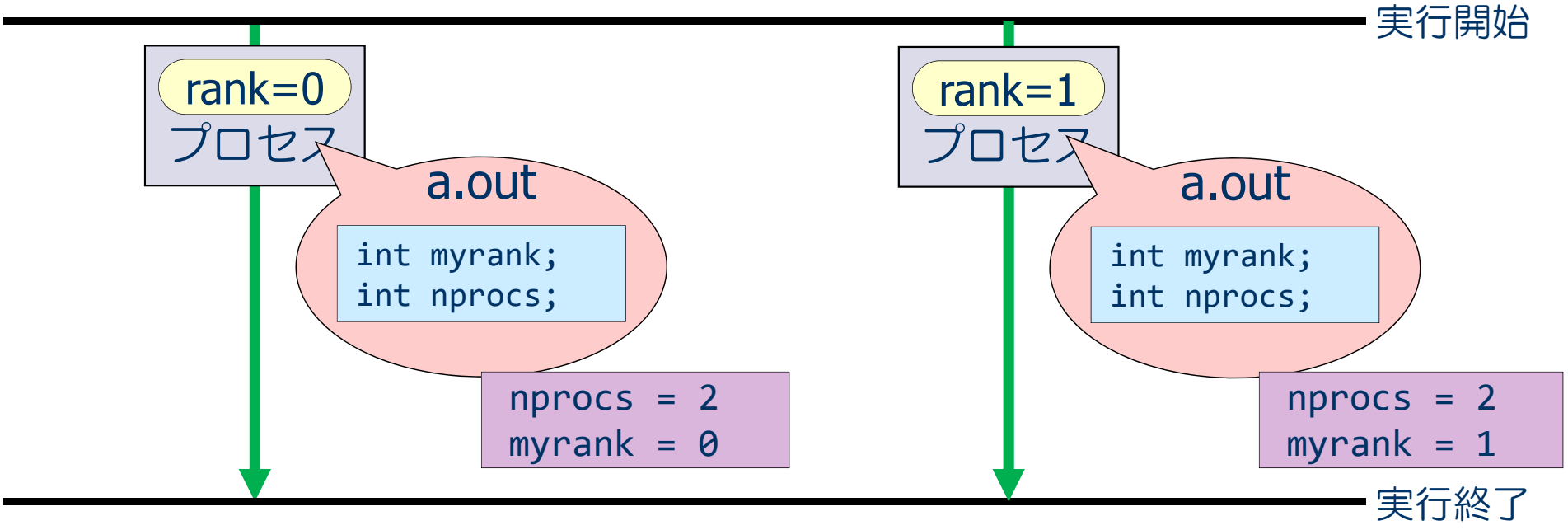
（注意）出力はランク順に並ぶとは限らず，また，実行ごとに出力の順番が異なることがある.

ポイント

- 各プロセスが同じプログラムを実行している.
- 各プロセスが持っているランク番号(myrankの値）が異なっている.

2プロセス時の変数の値（SPMD）

- 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



参考：Fortran版

MPIプログラムのスケルトン

```
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．

プログラムM-1F（hello.f90） の説明

`program hello_by_mpi`	
	（おまじない） mpi用のモジュールをインクルード
`use mpi` `implicit none`	
`integer :: nprocs, myrank, ierr`	
	（おまじない） MPIの初期化 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

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

`print *, 'Hello, world!  My rank number and nprocs are', myrank, ',',  nprocs`	
	各プロセスで myrank と nprocs を出力する. ※ myrank はプロセスごとに異なる ※ nprocs はすべてのプロセスで同じ
	（おまじない） MPIの終了処理
`call mpi_finalize( ierr )`	
`end program hello_by_mpi’`	

演習10-1F：Hello, world! を並列に出力する.

- MPI版 “Hello, world!” を 2，4，及び8プロセスで実行し，結果を確認せよ！ 以下は実行例.

$ mkdir␣MPI $ cd␣MPI	今日の演習用のディレクトリを作成する. ※ ␣はスペースを表わす.
$ mkdir␣M-1 $ cd␣M-1	演習M-1用のディレクトリを作成する.
$ cp␣/tmp/Summer/M-1/hello_mpi.f90␣./	ソースプログラム hello_mpi.c をカレントディレクトリにコピーする. ※ 中身を見て確認すること.
$ ifort␣hello_mpi.c␣-lmpi	ソースプログラムをコンパイル
$ cp␣/tmp/Summer/M-1/go.sh␣./	ジョブスクリプト go.sh をコピーする. （プロセス数の指定など，必要な部分をeditする）
$ qsub␣go.sh nnnnnn.rokko1	ジョブを投入し，実行結果を確認する. ※ nnnnnnはジョブ番号
$ cat␣hello.o*nnnnn*	※ helloは，go.sh内で指定した jobname のこと