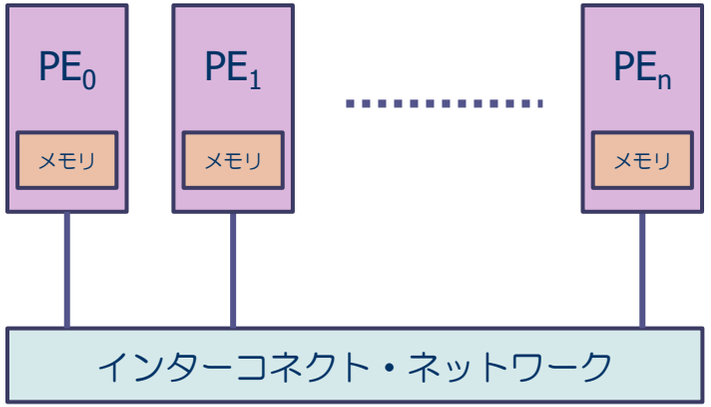
KOBE HPC サマースクール 2026（初級）

9. 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!

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

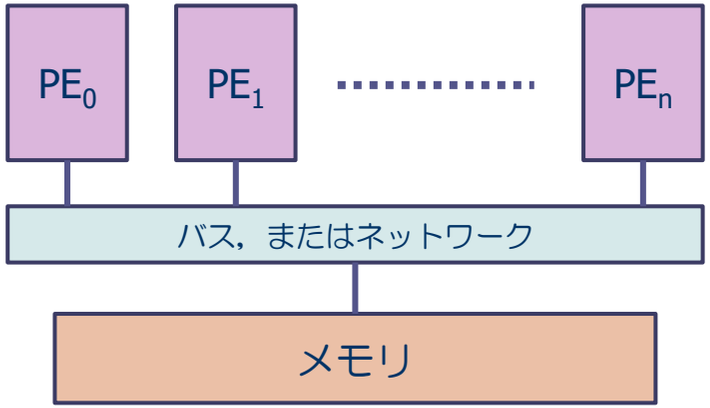


- 特徴
  - 数千から数万PE規模の並列システムが可能
  - PE の間のデータ分散を意識したプログラミングが必要．

- プログラミング方法
  - メッセージ・パシング・インターフェイス（MPI）を用いたプロセス並列プログラミング

（参考）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

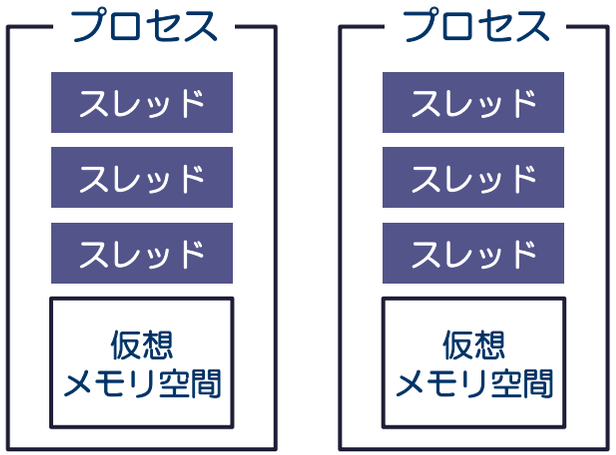


- 特徴
  - メモリ空間が単一なので，プログラミングが容易
  - PE の数が多いと，同一メモリアドレスへのアクセス競合が生じ，性能が低下

- プログラミング方法
  - OpenMPを用いたスレッド並列プログラミングまたは自動並列化
  - ただし，MPIプログラムを実行することも可能

プロセス並列とスレッド並列の違い

- スレッド並列（OpenMP）は，一つのプロセス内で行う並列処理.
  - メモリ空間は一つだった.
  - どのスレッドも，共有メモリ上の変数をアクセスできた.



- プロセス並列は，複数のプロセスによる並列処理.
  - プロセスは，独立したメモリ空間を持っている.
  - あるプロセスは，他のプロセスのメモリ上の変数にアクセスできない.
  - 他のプロセスの持つ変数の値が欲しい場合には，プロセス間でデータのやり取り（送受信）をしないといけない.
    - プロセス間でデータのやり取りを記述する方法が必要.
  - プログラミング技術：「MPI」を用いたSPMDプログラミング

メッセージ・パッシング・インターフェイス（MPI）

- 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（片方向通信など）
    - 2012　MPI-3.0
    - 2021　MPI-4.0（分割通信、MPIセッションなど）
    - 2023　MPI-4.1
    - 2025　MPI-5.0
      - https://www.mpi-forum.org/docs/

MPIの実装ライブラリ

- MPIは規格であって，それを実現しているのが実装ライブラリ
  - 共通の規格なので，他の計算機システムでコンパイル，実行することが可能.

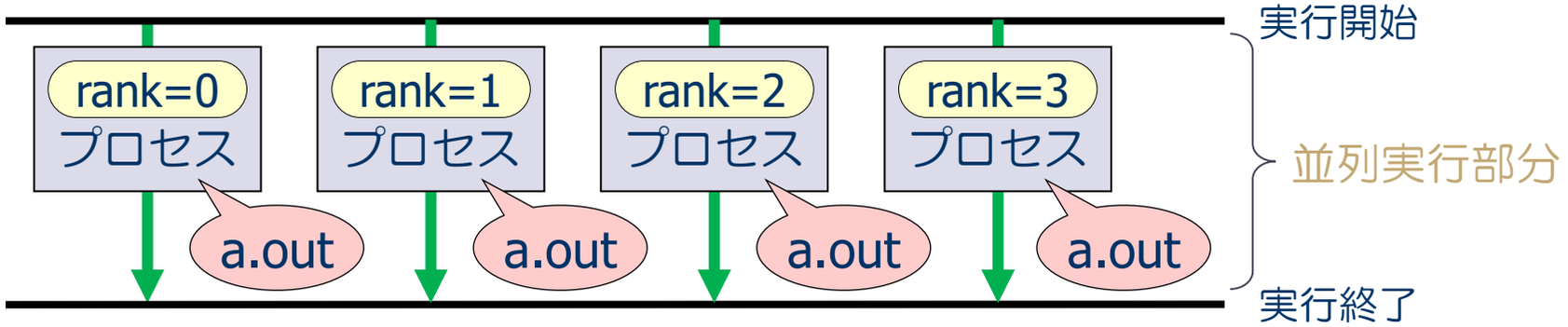
- 代表的なMPIライブラリ
  - MPICH　　：米国アルゴンヌ国立研究所が開発
  - MVAPICH　：米国オハイオ州立大学
  - Open MPI　：Open MPIコンソーシアム
    （FT-MPI，LA-MPI，LAM/MPI，PACX-MPIの統合プロジェクト）
  - xxxxx　　：各計算機ベンダーの独自MPIライブラリ
    例．MPT: SGI Message Passing Toolkit ⇒ 本授業で使用

※ 各メーカのハードウェア機構を活かす独自仕様が含まれている場合がある.

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.

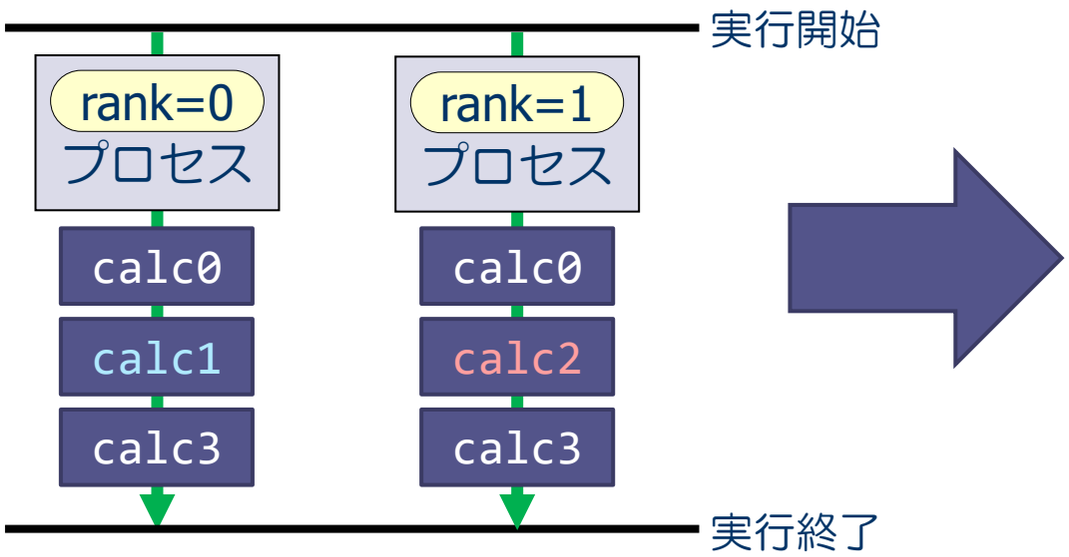


MPIの実行モデル（続き）

- 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

⇒ ある単一の MPIプログラムを実行するとき，各プロセスが処理する内容を頭の中でイメージ（頭の中で動作をシミュレート）することが重要

```
calc0();
if( rank == 0 ) {
   calc1();
} else { /*if( rank == 1)*/
   calc2();
}
calc3();
```



- プロセス0

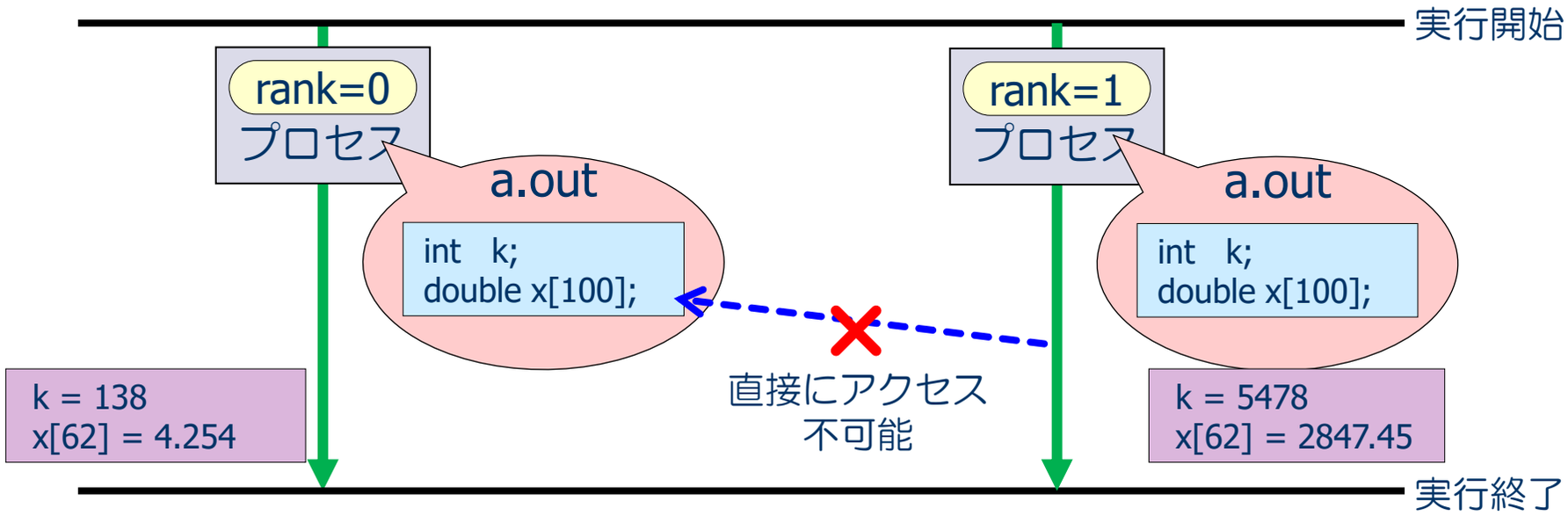
```
calc0();
calc1();
calc3();
```

- プロセス1

```
calc0();
calc2();
calc3();
```

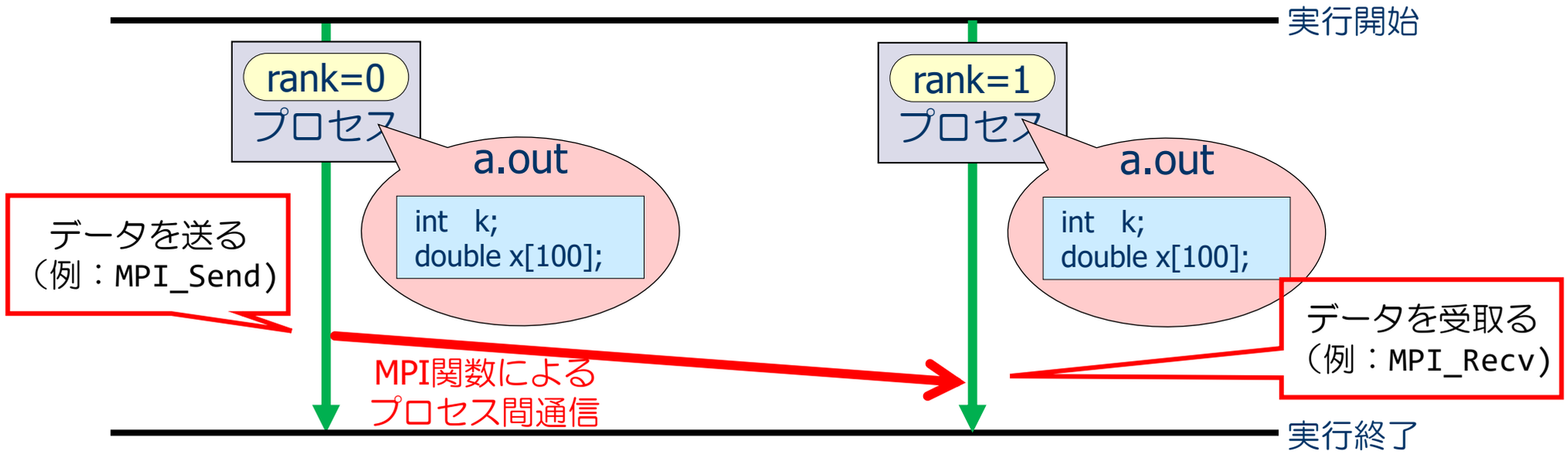
MPIの実行モデル（続き）

- 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>

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

- `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（hello_mpi.c）： Hello, world!

```
#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 );

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

  MPI_Finalize();
  return 0;
}
```

県立大 Rokko システムの SGI版MPI を使う手順

- 環境設定：インテルコンパイラ，SGI版MPI を使えるようにする．

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

- コンパイル方法

```
$  icx 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 エックス　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：ファイル名は 「***.sh」（たとえば jobm.sh など）としておくと良い

```
#!/bin/bash
#PBS -q WS
#PBS -l select=1:ncpus=2:mpiprocs=2

#PBS -N hello
#PBS -j oe

source /etc/profile.d/modules.sh
module load intel/2022.3.1
module load mpt
cd ${PBS_O_WORKDIR}
mpiexec_mpt -np XX dplace -s1 ./a.out
```

キューの設定：スクール専用キューWS（8/26～9/8の期間に使用可）

select:ノード数，ncpus:ノード内コア数

MPIだけの並列プログラム実行の場合はncpus と mpiprocs の値は同じにする．

helloは，出力ファイルヘッダ

XX: MPI並列数　-np XX を指定しない場合は，スクリプトの値が使われる．

演習9-1：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␣/home/guest60/share/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，プログラムを確認すること.
`$ icx␣hello_mpi.c␣-lmpi`	ソースプログラムをコンパイル
`$ cp␣/home/guest60/share/jobm.sh␣./`	ジョブスクリプト jobm.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␣jobm.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.onnnnn`	※ hello は，jobm.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.
```

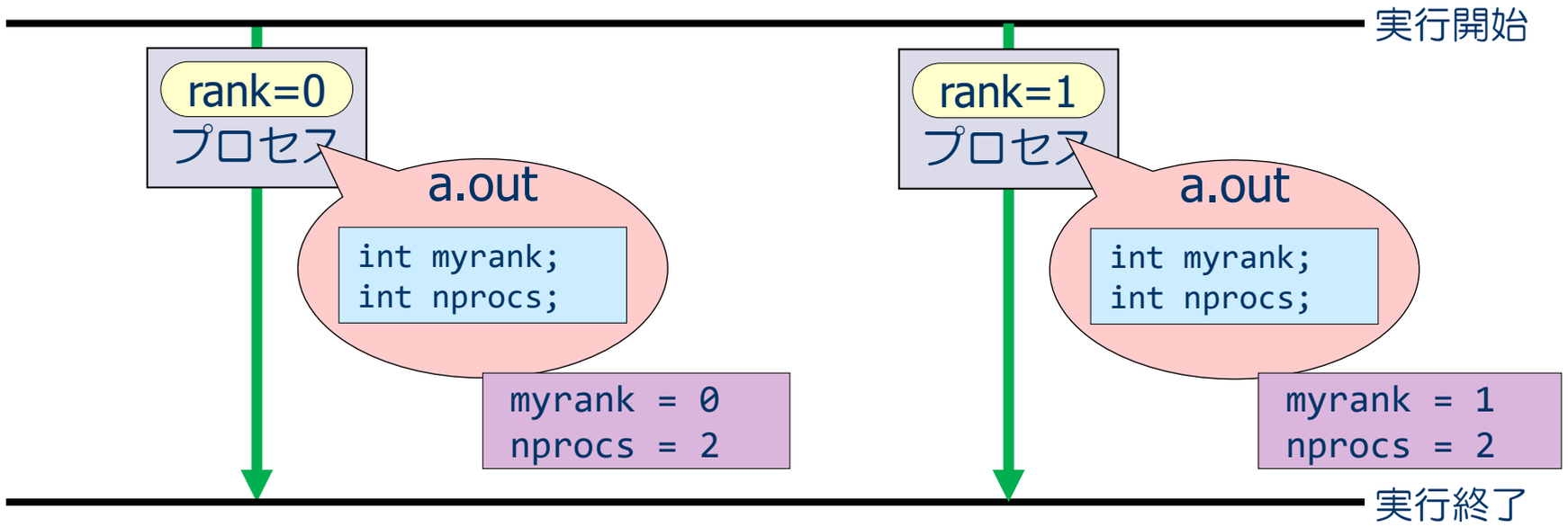
（注意）出力はランク順に並ぶとは限らず，また，実行ごとに出力の順番が異なることがある.

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．
    - 同じ変数名や配列名だが保持するプロセスが異なれば「実体は別物」．つまり**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（hello_mpi.f90） の説明

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

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

`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
`call mpi_finalize( ierr )`	（おまじない） MPIの終了処理
`end program hello_by_mpi`	

演習9-1：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                                   今日の演習用のディレクトリを作成する.
$  cd␣MPI                                      ※ ␣はスペースを表わす.

$  mkdir␣M-1                                   演習M-1用のディレクトリを作成する.
$  cd␣M-1

$  cp␣/home/guest60/share/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.

$  ifx␣hello_mpi.f90␣-lmpi                     ソースプログラムをコンパイル

$  cp␣/home/guest60/share/jobm.sh␣./           ジョブスクリプト jobm.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␣jobm.sh                                ジョブを投入し，実行結果を確認する.
nnnnnn.rokko1                                   ※ nnnnnn はジョブ番号

$  cat␣hello.onnnnn                            ※ hello は，jobm.sh内で指定した jobname のこと
```

スペース