集中講義（兵庫県立大）

神戸大学 システム情報学研究科
陰山　聡

MPI（の復習）

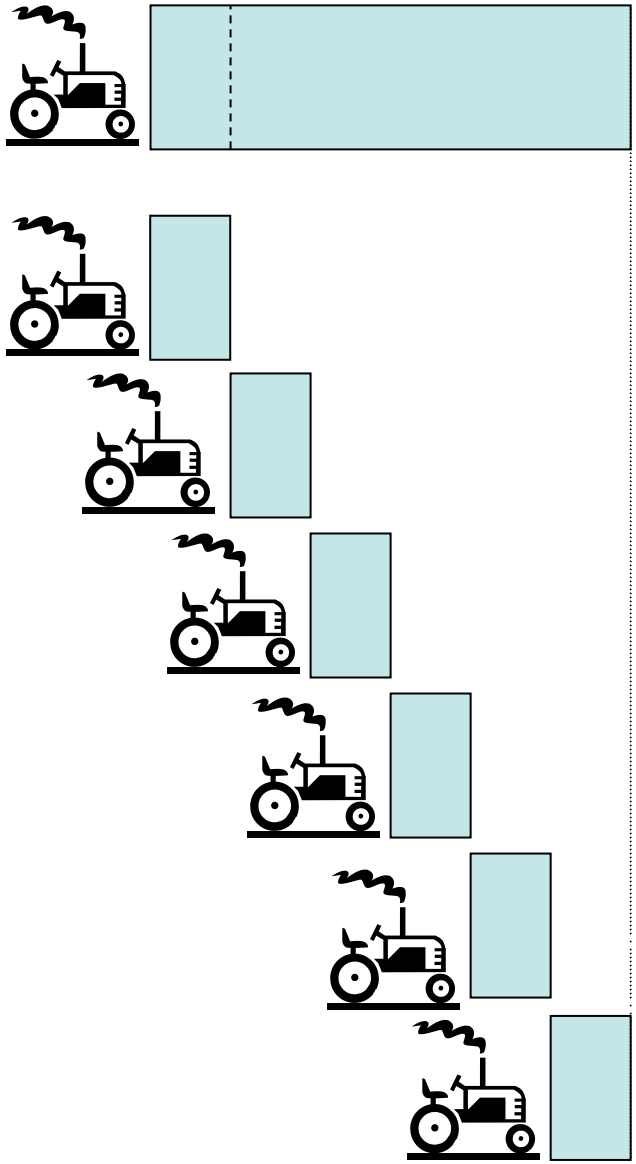
並列計算とは何か

- 作業の分担による処理の高速化
- 三千万回のかけ算・足し算の例
  - 1人で計算すると100年間
  - 100人で計算すれば1年間
  - 1千万人で計算すれ５分程度

作業の分担による処理の高速化

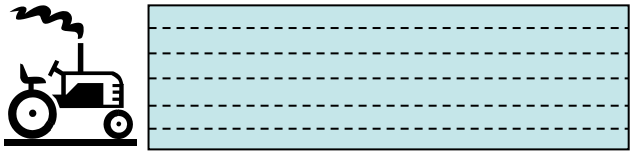
作業の分担による処理の高速化

作業の分担による処理の高速化

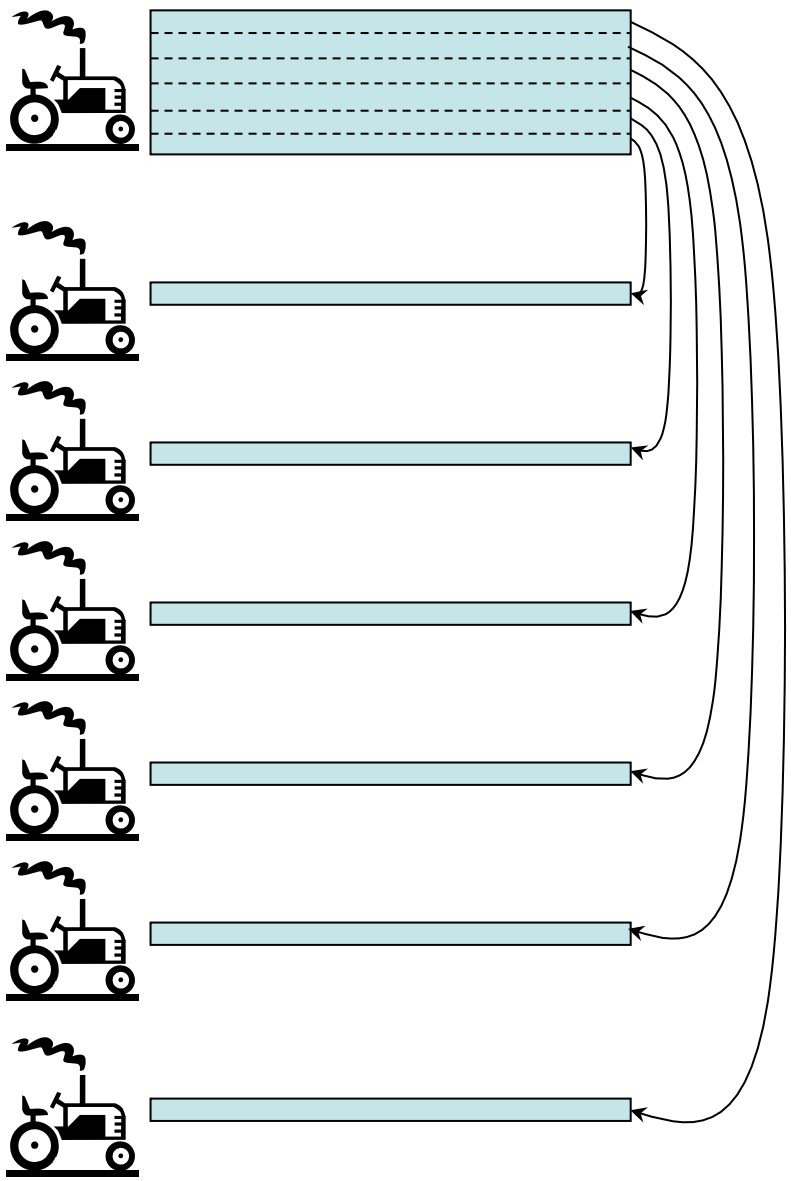


作業の分担（並列化）：別の方法

作業の分担（並列化）：別の方法



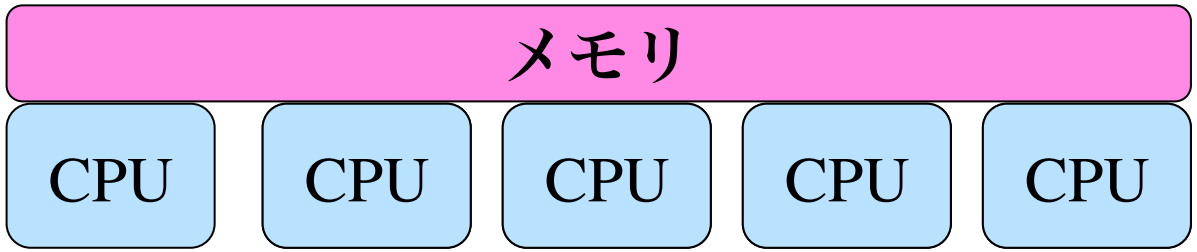
作業の分担（並列化）：別の方法



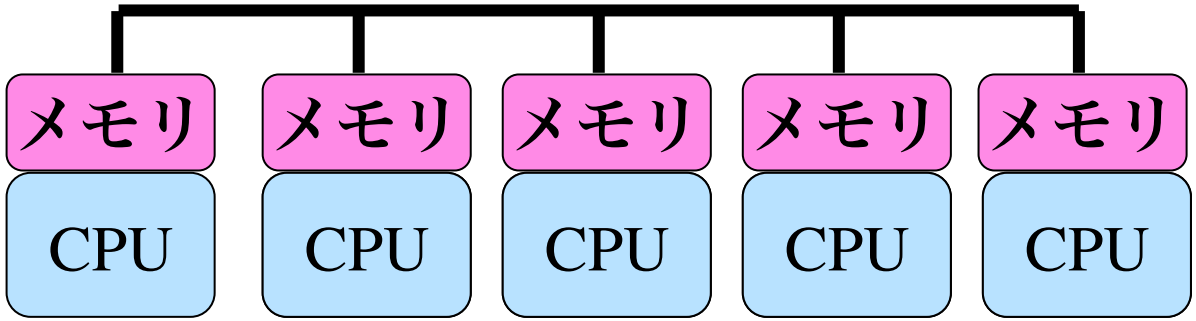
並列計算機

- 共有メモリ型

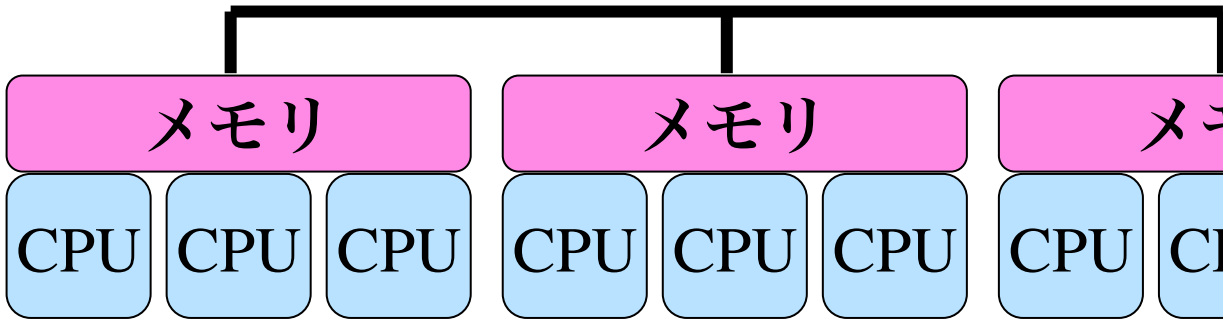
  SMP
  Symmetric MultiProcessing



- 分散メモリ型

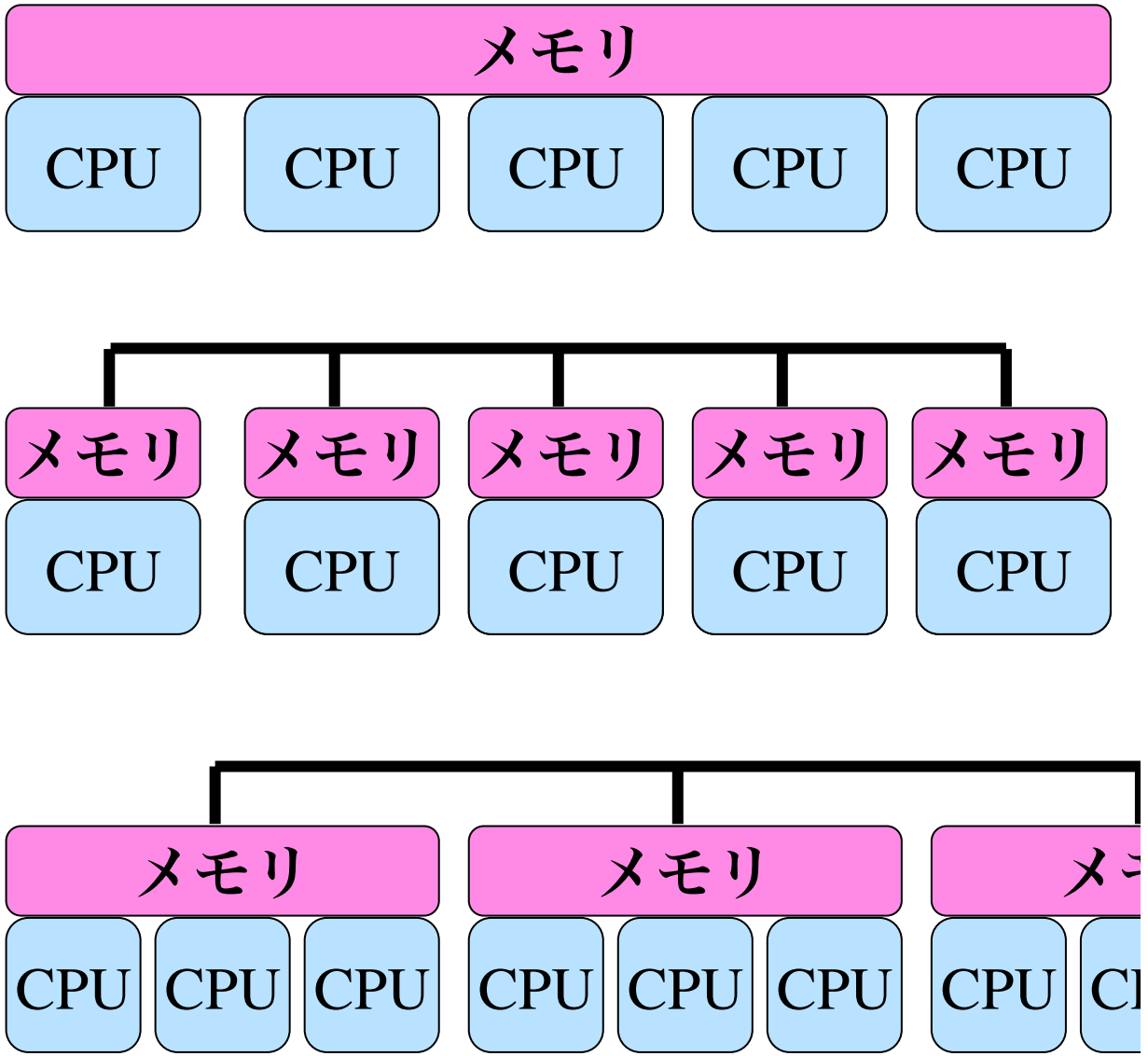


- 混在型

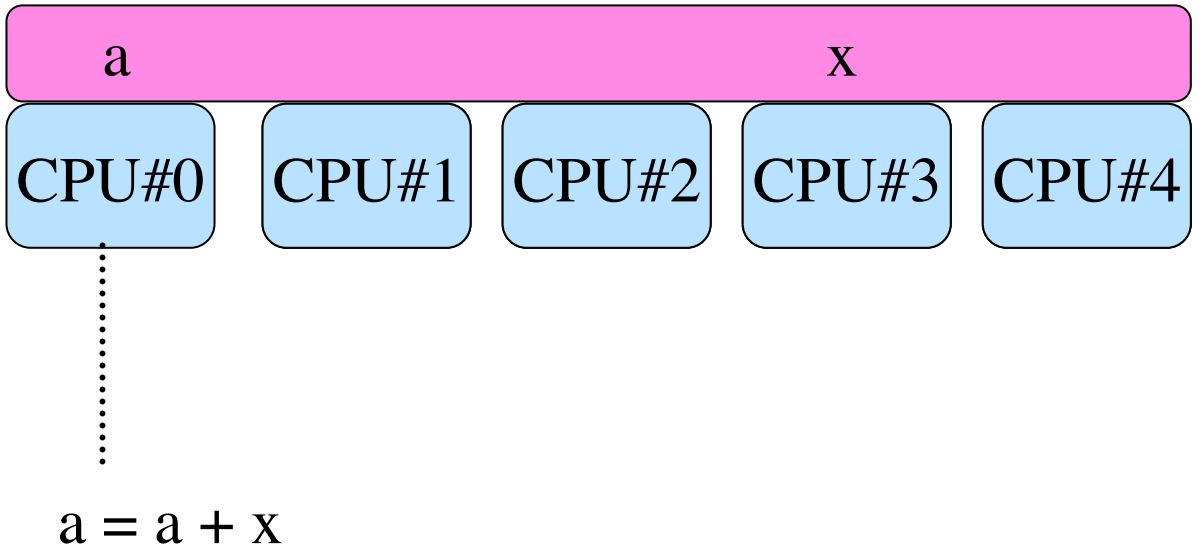


並列プログラミングのスタイル

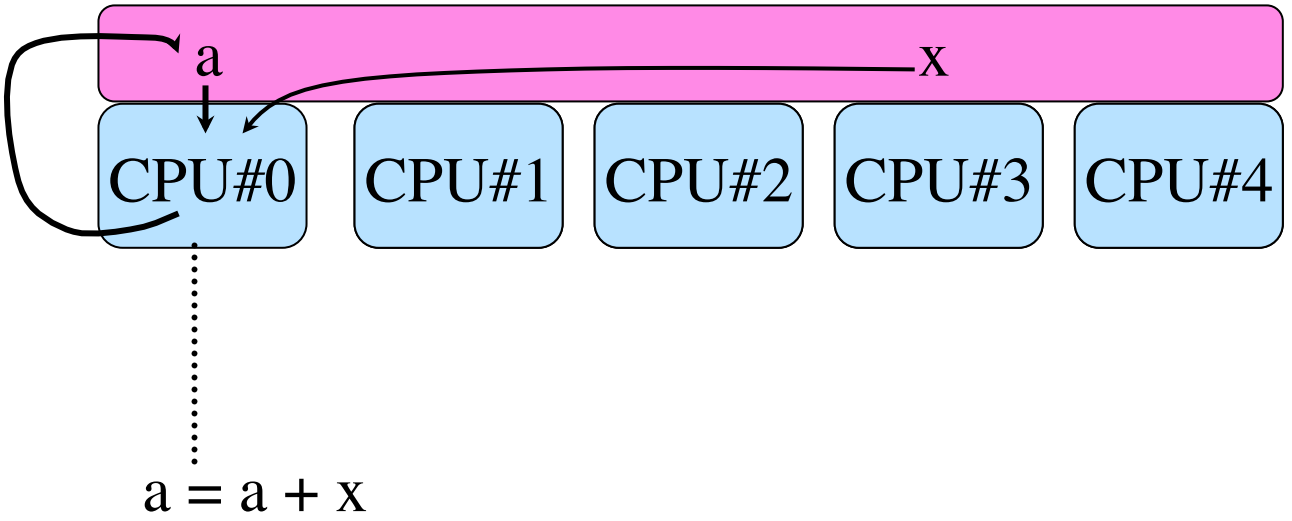
- 共有メモリ型
  - OpenMP
  - …
  - MPI
- 分散メモリ型
  - MPI
  - …
- 混在型
  - MPI
  - MPI+OpenMP
  - …



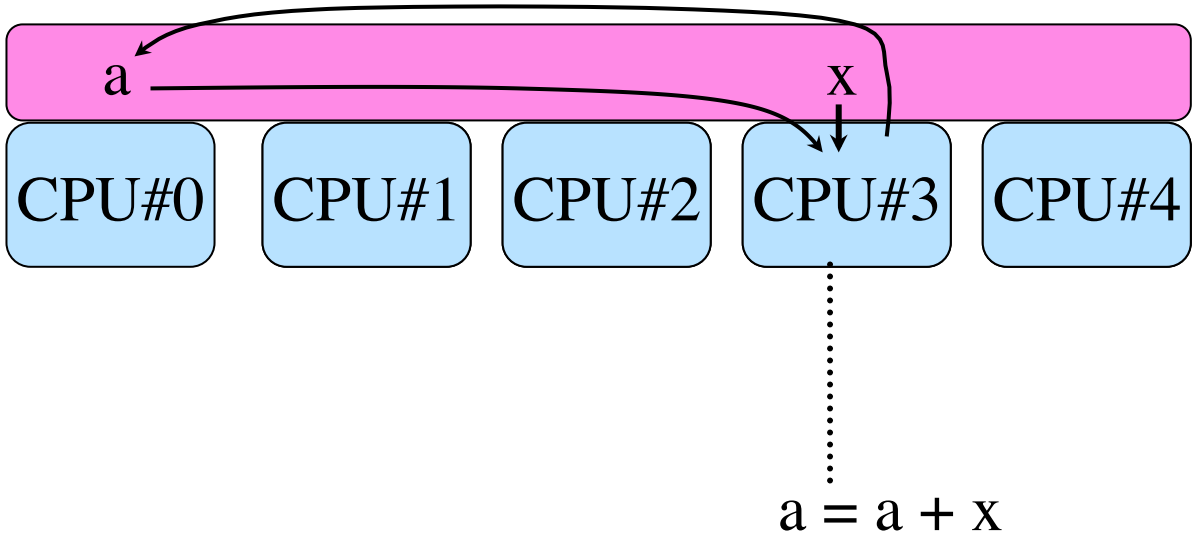
共有メモリ型並列計算機のイメージ



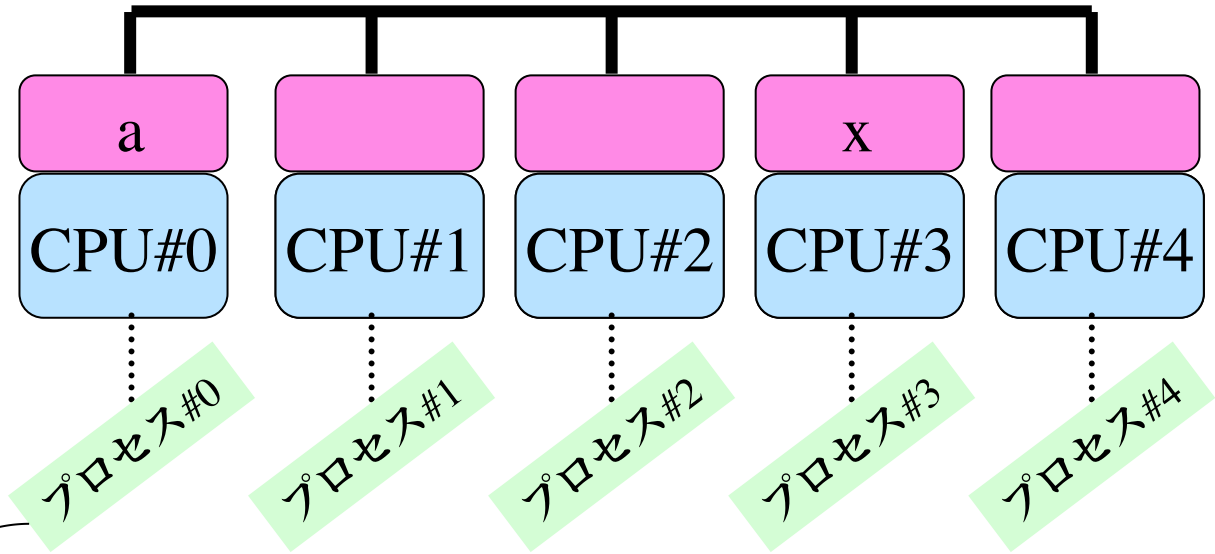
共有メモリ型並列計算機のイメージ



共有メモリ型並列計算機のイメージ



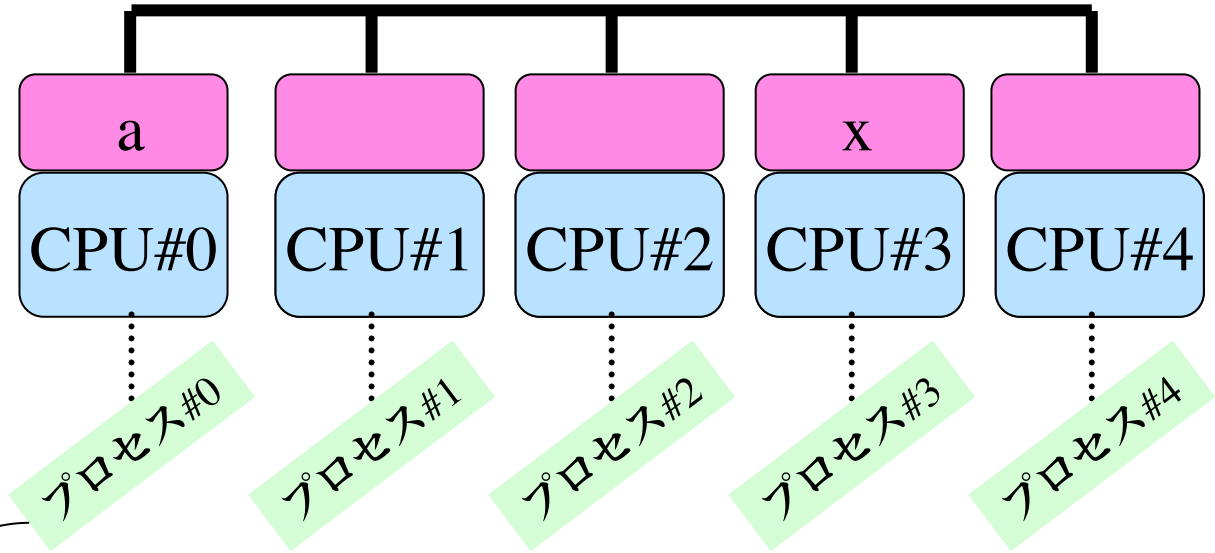
分散メモリ型並列計算機の利用方法



a = a + x

- プロセス#0は a の値は持っている。
- しかし x の値は持っていない。

分散メモリ型並列計算機の利用方法

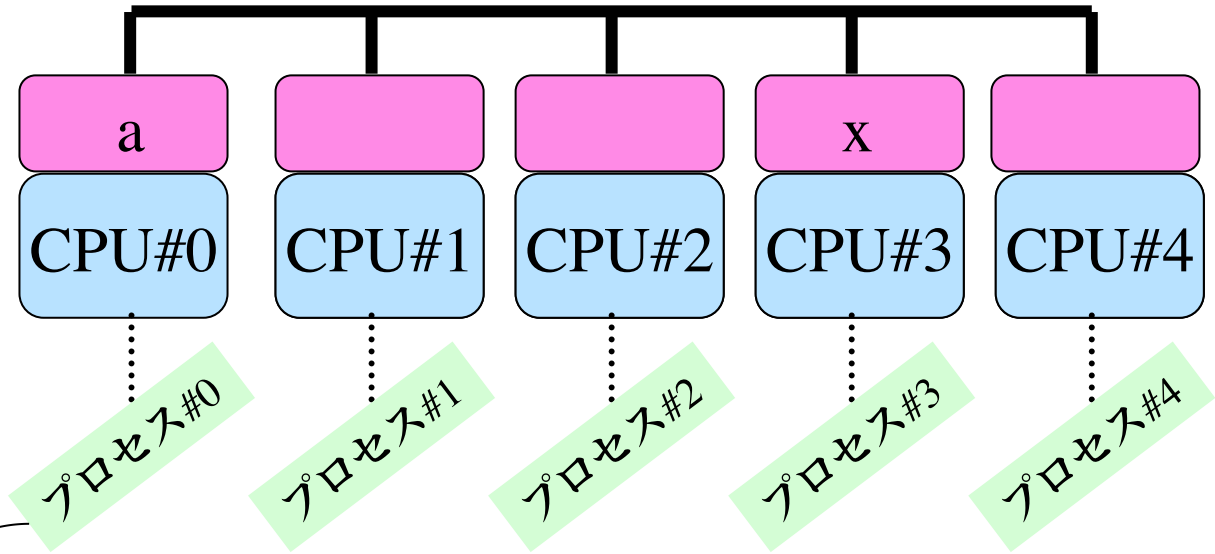


a = a + x

・プロセス#0は a の値は持っている。
・しかし x の値は持っていない。

二つの方法

分散メモリ型並列計算機の利用方法



a = a + x

・プロセス#0は a の値は持っている。
・しかし x の値は持っていない。

二つの方法

（１）プロセス#3が管理するメモリに直接アクセスする。

（２）プロセスに#3に x の値を送ってもらう。

分散メモリ型並列計算機の利用方法

（１）プロセス#3が管理するメモリに直接アクセスする。

→　Remote Memory Access

（２）プロセスに#3に x の値を送ってもらう。

→　メッセージのやりとり

= Message Passing

・MPI（=Message Passing Interface）は上の二つを可能にするライブラリ。

・リモートメモリアクセスはMPI2で実現。

MPI

- Message Passing方式の並列計算は基本。
- 簡単。
- 一般的。
- 高い性能。

MPI入門

- 具体的、実践的な入門
  - ソースコードを見ながら。
  - 実際にコンパイルと実行を行う。
  - 例題：
    - MPIを使った10行並列プログラム
    - メッセージの送受信方法
    - 送受信の同期と非同期、デッドロック
    - 「横一列赤白帽子ゲーム」のMPIプログラム版
    - 縦横並びの赤白帽子ゲーム（Game of Life）のMPIプログラム版

MPIは簡単

- 必須関数（サブルーチン）はたった７つ：
  - MPI_Init
  - MPI_Comm_Size
  - MPI_Comm_rank
  - MPI_Send
  - MPI_Recv
  - MPI_Finalize
- その他、この後の例題で出てくる関数：
  - MPI_Isend
  - MPI_Irecv
  - MPI_Sendrecv
  - MPI_Reduce
  - MPI_Cart_create
  - MPI_Shift
  - MPI_Cart_coord

MPI

- Fortran90, FORTRAN77, C, C++から利用できる。ライブラリ。
- フリーのMPI実装が容易に手に入る。
  - mpich
  - Open MPI (lam/mpi)
  - . . .
- Reference
  - “Using MPI” by Gropp et al., MIT press.
    - これ一冊読めば十分。
  - “Using MPI2”
    - MPI2の解説
  - MPI: The Complete Reference, Vol 1 and 2

MPIを使った並列 f90 プログラム

source: MPI/sample00.f90

```
program sample00
! use mpi   [or include 'mpif.h']
  implicit none
  include 'mpif.h'

  integer :: myrank, numprocs, ierr

  call MPI_INIT(ierr)
  call MPI_COMM_RANK(MPI_COMM_WORLD, myrank, ierr)
  call MPI_COMM_SIZE(MPI_COMM_WORLD, numprocs, ierr)

  print *, repeat('#',myrank+1)

  call MPI_FINALIZE(ierr)
end program sample00
```

MPIを使った並列 f90 プログラム

source: MPI/sample00.f90

MPIライブラリのf90モジュールがあればこちら

```
 program sample00
 ! use mpi   [or include 'mpif.h']
   implicit none
   include 'mpif.h'
コメント
   integer :: myrank, numprocs, ierr

   call MPI_INIT(ierr)
   call MPI_COMM_RANK(MPI_COMM_WORLD, myrank, ierr)
   call MPI_COMM_SIZE(MPI_COMM_WORLD, numprocs, ierr)

   print *, repeat('#',myrank+1)

   call MPI_FINALIZE(ierr)
 end program sample00
```

MPIを使った並列プログラム

```
program sample00
! use mpi   [or include 'mpif.h']
  implicit none
  include 'mpif.h'

  integer :: myrank, numprocs, ierr

  call MPI_INIT(ierr)
  call MPI_COMM_RANK(MPI_COMM_WORLD, myrank, ierr)
  call MPI_COMM_SIZE(MPI_COMM_WORLD, numprocs, ierr)

  print *, repeat('#',myrank+1)

  call MPI_FINALIZE(ierr)
end program sample00
```

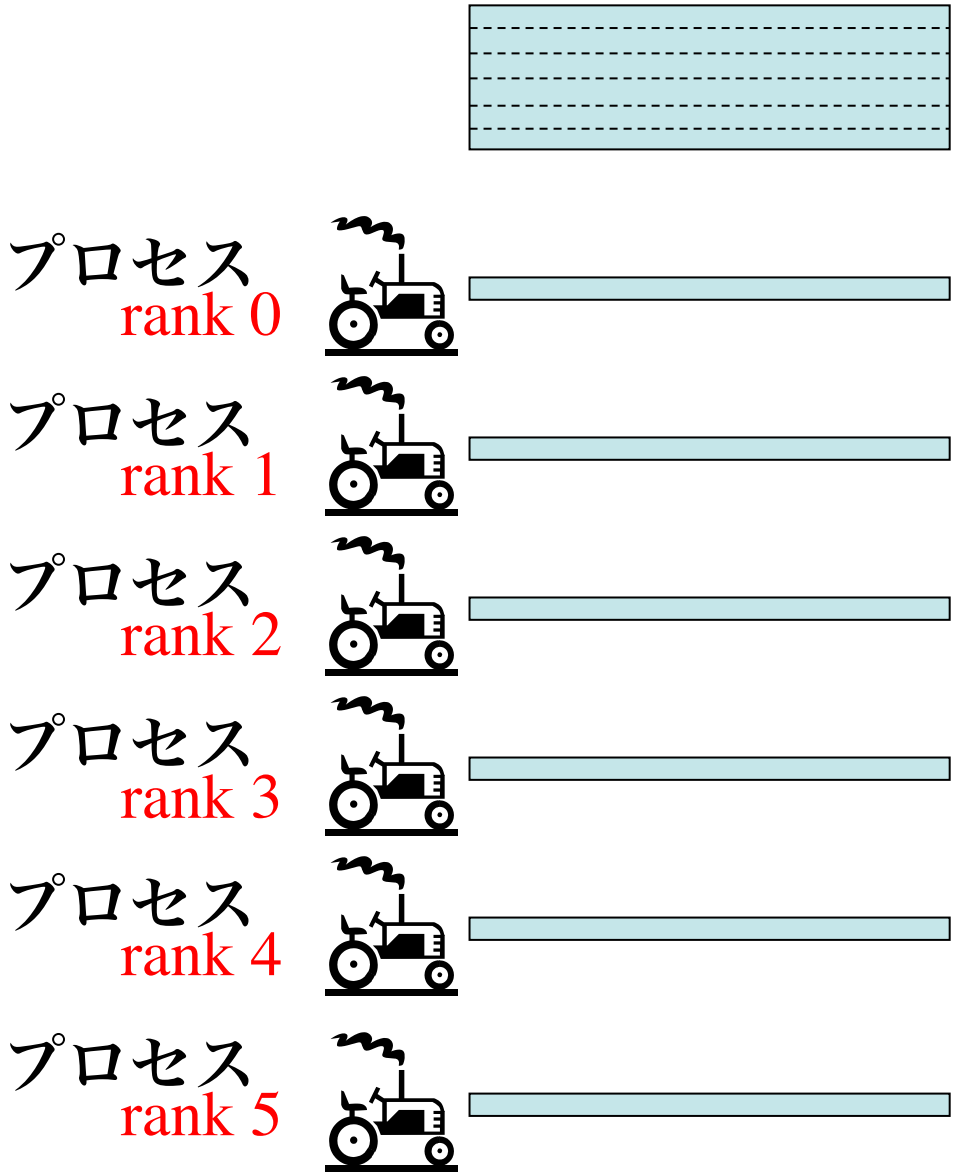
初期化。最初に呼ぶ。

ふつう使わない。

MPIを使った並列プログラム

```
program sample00
! use mpi   [or include 'mpif.h']
  implicit none
  include 'mpif.h'
      プロセスにつけられた固有の番号（ランク番号）をとる
  integer :: myrank, numprocs, ierr
                                        ランク番号
  call MPI_INIT(ierr)                       |
  call MPI_COMM_RANK(MPI_COMM_WORLD, myrank, ierr)
  call MPI_COMM_SIZE(MPI_COMM_WORLD, numprocs, ierr)

  print *, repeat('#',myrank+1)

  call MPI_FINALIZE(ierr)
end program sample00
```

６つのプロセスによる並列処理

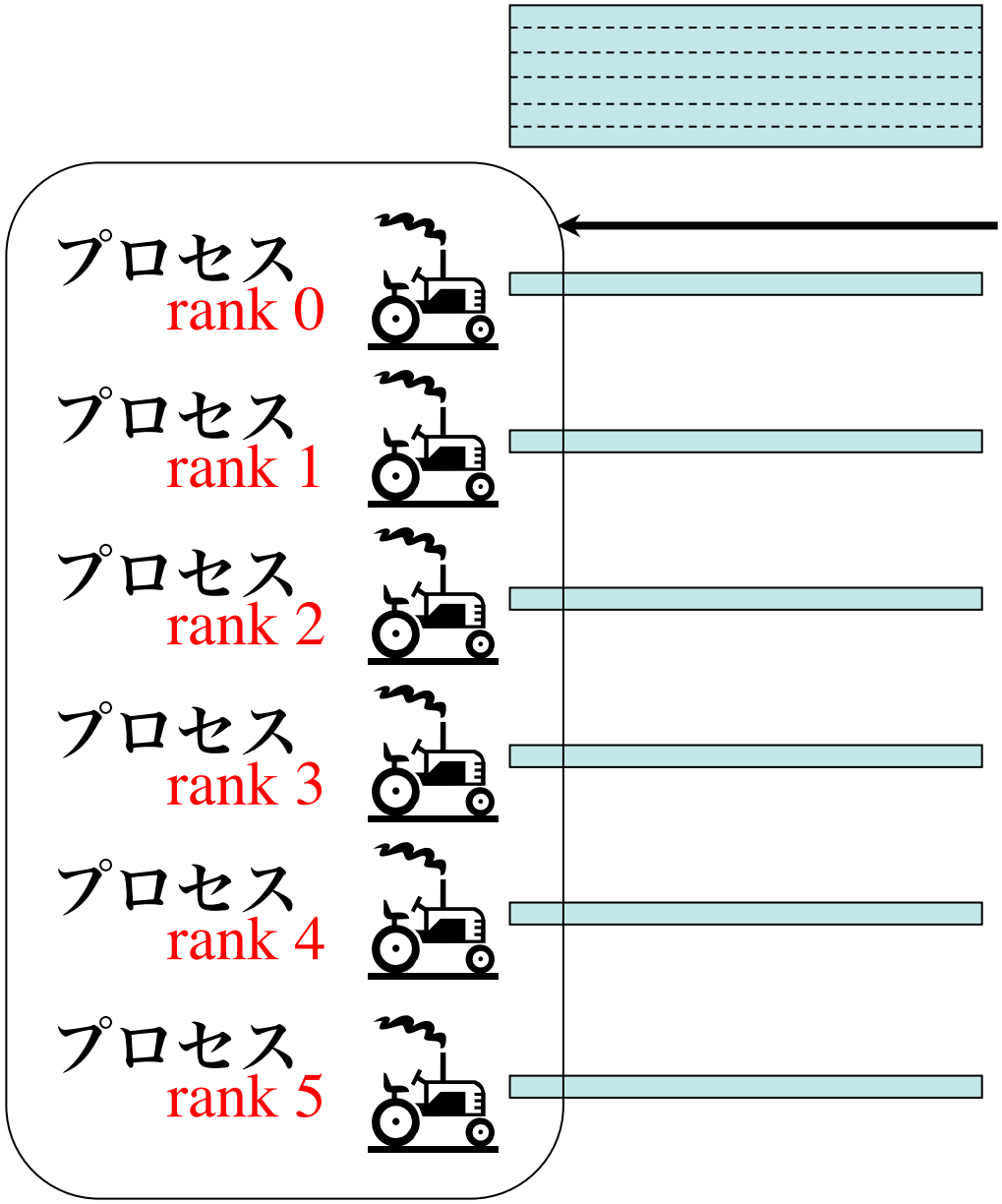


- 各プロセスには固有の番号（ランク番号）
- 他のプロセスとメッセージをやりとりするにはランク番号を使う。
- 何もしない限り同期はしない。

MPIを使った並列プログラム

```
program sample00
! use mpi   [or include 'mpif.h']
  implicit none
  include 'mpif.h'
      コミュニケータ内部の全てのプロセス数を得る
  integer :: myrank, numprocs, ierr

  call MPI_INIT(ierr)
  call MPI_COMM_RANK(MPI_COMM_WORLD, myrank, ierr)
  call MPI_COMM_SIZE(MPI_COMM_WORLD, numprocs, ierr)

  print *, repeat('#',myrank+1)

  call MPI_FINALIZE(ierr)
end program sample00
```

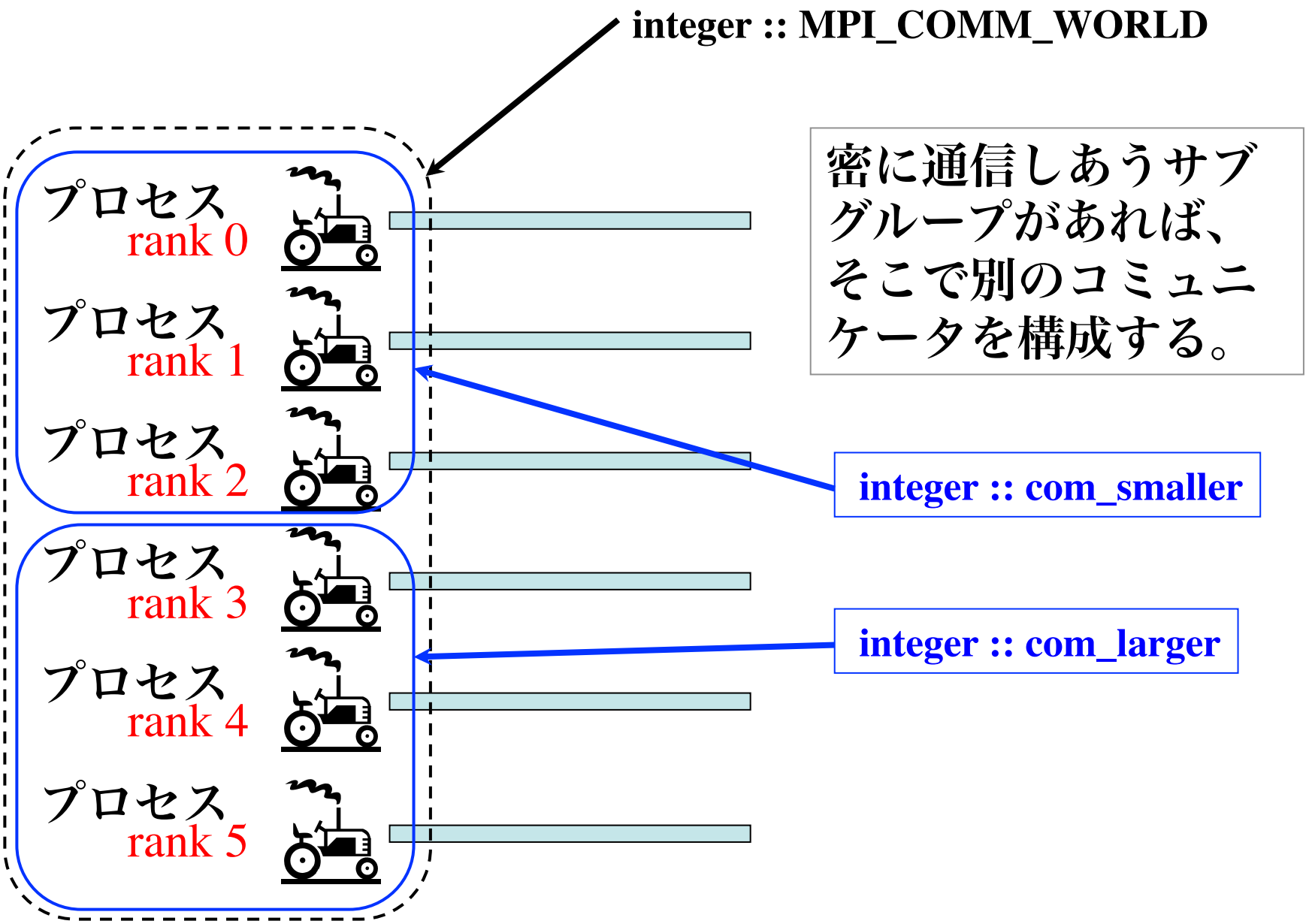
６つのプロセスによる並列処理



通信しあえる仲間どうしは
「コミュニケータ」を作る。

最初にできる全プロセスを
含むコミュニケータは
MPI_COMM_WORLD

６つのプロセスによる並列処理



MPIを使った並列プログラム

```
program sample00
! use mpi   [or include 'mpif.h']
  implicit none
  include 'mpif.h'

  integer :: myrank, numprocs, ierr

  call MPI_INIT(ierr)
  call MPI_COMM_RANK(MPI_COMM_WORLD, myrank, ierr)
  call MPI_COMM_SIZE(MPI_COMM_WORLD, numprocs, ierr)

  print *, repeat('#',myrank+1)

  call MPI_FINALIZE(ierr)
end program sample00
```

Rank番号は0から始まる

Fortran組み込み関数（文字列の繰り返し）

MPIを使った並列プログラム

```
program sample00
! use mpi   [or include 'mpif.h']
  implicit none
  include 'mpif.h'

  integer :: myrank, numprocs, ierr

  call MPI_INIT(ierr)
  call MPI_COMM_RANK(MPI_COMM_WORLD, myrank, ierr)
  call MPI_COMM_SIZE(MPI_COMM_WORLD, numprocs, ierr)

  print *, repeat('#',myrank+1)

  call MPI_FINALIZE(ierr)
end program sample00
```

MPIプログラムの最後に呼ぶ

MPIプログラムのコンパイルと実行

MPIプログラムのコンパイル

```
> mpif90 sample00.f90 -o sample00
```

（mpif90という名前ではないかもしれない。）

```
> mpirun -np 10 sample00
```

・この場合、10プロセスで並列計算。
・mpirunコマンドの引数（-npオプション）でプロセス数指定。

または（MPI Forum推奨形式）

```
> mpiexec -n 10 sample00
```

実行例

```
>  mpirun -np 10 sample00
```

実行結果：

```
#
##
###
#####
####
#########
######
##########
#######
########
```

rank 0の出力

rank 1の出力

ばらばらに実行。

MPI: 強制的に同期させる

source: MPI/sample01.f90

```
program sample01
! use mpi   [or include 'mpif.h']
  implicit none
  include 'mpif.h'
  integer :: myrank, numprocs, ierr

  call MPI_INIT(ierr)
  call MPI_COMM_RANK(MPI_COMM_WORLD, myrank, ierr)
  call MPI_COMM_SIZE(MPI_COMM_WORLD, numprocs, ierr)

  print *, repeat('#',myrank+1)

  call MPI_BARRIER(MPI_COMM_WORLD, ierr)

  if ( myrank==0 ) print *, '---------barrier---------'
```

（続く）

MPI: 同期をとる

source: MPI/sample01.f90

（続き）

```
  do i = 0 , numprocs
     if ( myrank==i ) print *, repeat('#',myrank+1)
     call MPI_BARRIER(MPI_COMM_WORLD, ierr)
  end do

  call MPI_FINALIZE(ierr)

end program sample01
```

実行例

```
>  mpirun -np 10 sample01
```

実行結果：

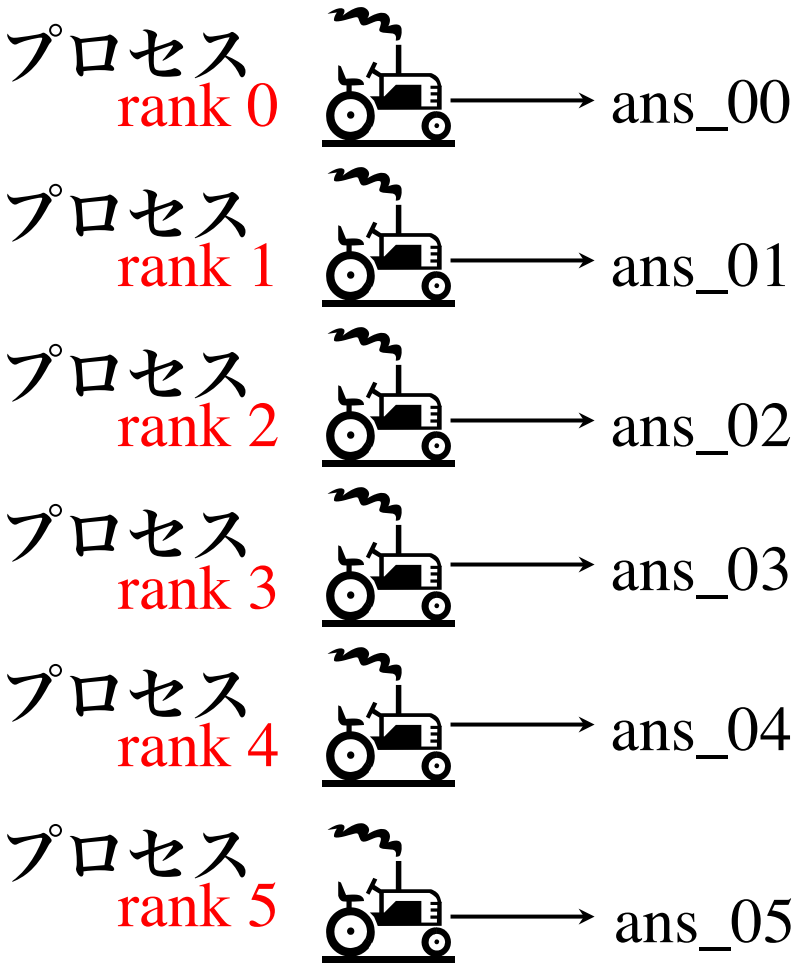
```
#
##
###
####
#####
#########
######
##########
#######
########
----------barrier----------
#
##
```

実行例

```
########
---------barrier---------
#
##
###
####
#####
######
#######
########
#########
##########
```

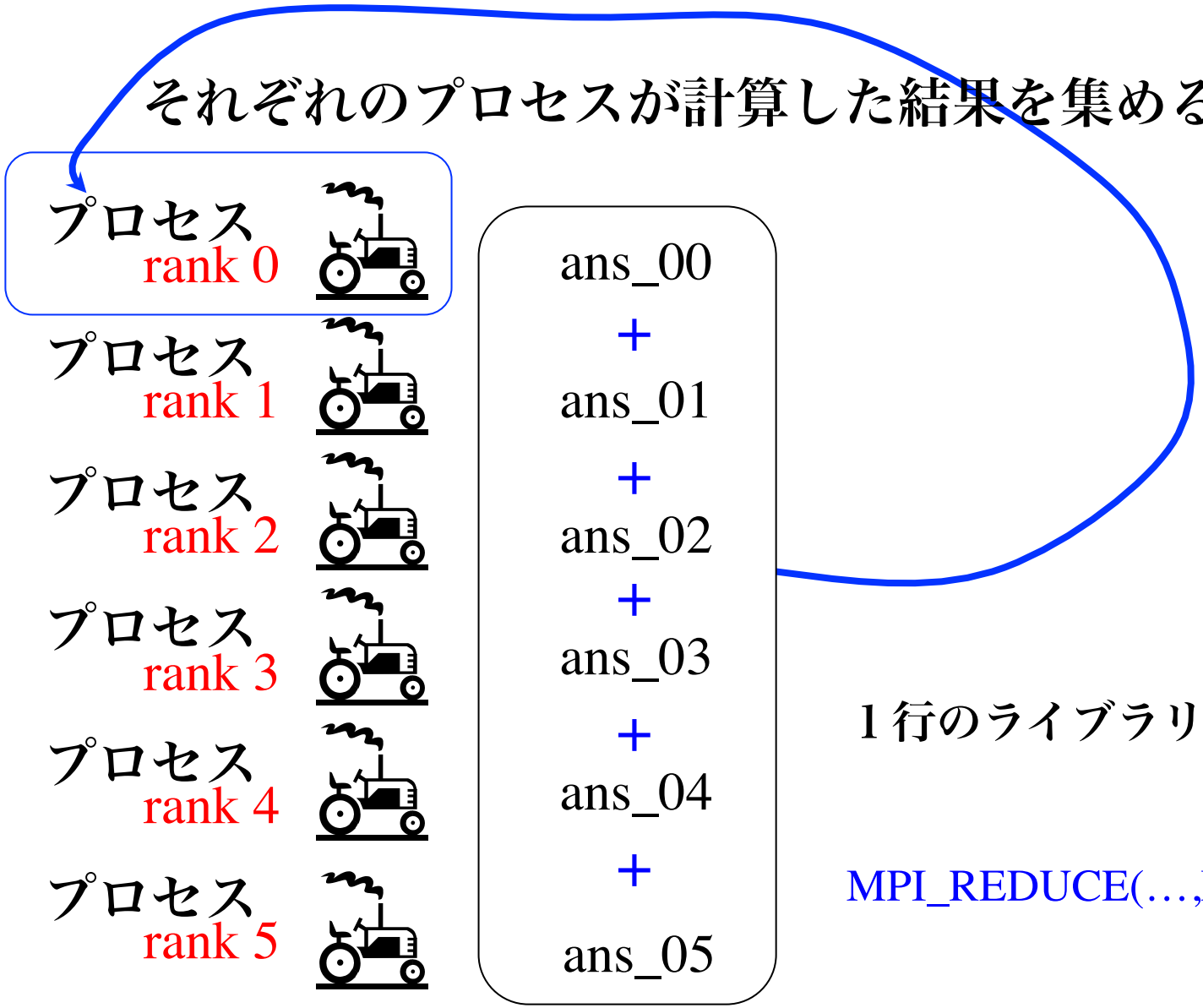
MPIを使った並列 f90 プログラム

それぞれのプロセスが計算した結果を集める



ほしいのは
ans_00 から ans_05
までの総和

MPIを使った並列 f90 プログラム



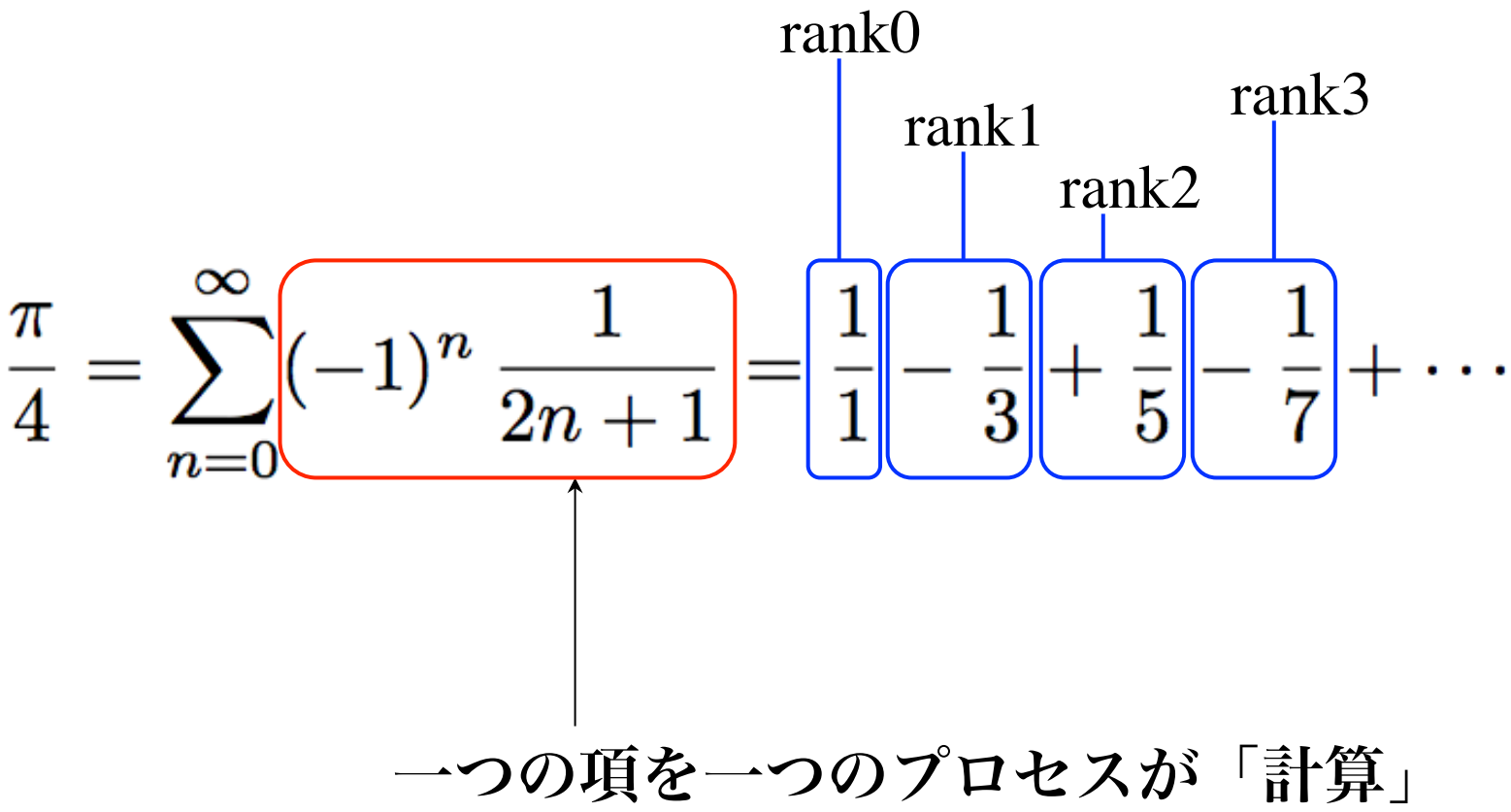
1 行のライブラリコールで可能！

MPI_REDUCE(…,MPI_SUM,…)

MPIを使った並列 f90 プログラム

$$\frac{\pi}{4} = \sum_{n=0}^{\infty} (-1)^n \frac{1}{2n+1} = \frac{1}{1} - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \cdots$$

MPIを使った並列 f90 プログラム



MPIを使った並列 f90 プログラム

source: MPI/sample02.f90

```
program sample02
  use constants
! use mpi  [or include 'mpif.h']
  implicit none
  include 'mpif.h'

  integer :: myrank, numprocs
  integer :: ierr
  integer :: wcomm = MPI_COMM_WORLD

  real(DP) :: my_value, total

  call MPI_INIT(ierr)
  call MPI_COMM_RANK(wcomm, myrank, ierr)
  call MPI_COMM_SIZE(wcomm, numprocs, ierr)
...
```

MPIを使った並列 f90 プログラム

source: MPI/sample02.f90

```
...
  my_value=(-1)**myrank/(2*(real(myrank,DP))+1.0_DP)

  call MPI_REDUCE(my_value, total,           &
                  1, MPI_DOUBLE_PRECISION,   &
                  MPI_SUM,                   &
                  0, wcomm, ierr)

  if ( myrank==0 ) print *, '4*total = ', 4*total

  call MPI_FINALIZE(ierr)

end program sample02
```

倍精度実数を一つ

rank 0に集める

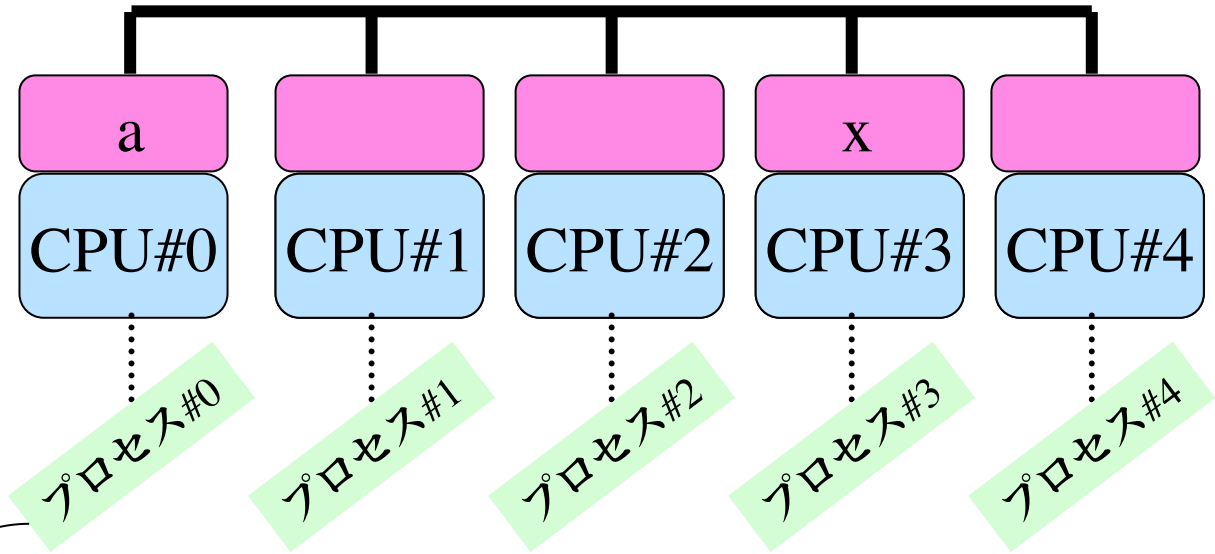
実行例

```
> mpirun -np 2 sample02
 4*total =   2.66666666666667

> mpirun -np 4 sample02
 4*total =   2.89523809523810

> mpirun -np 8 sample02
 4*total =   3.01707181707182

> mpirun -np 16 sample02
 4*total =   3.07915339419743

> mpirun -np 32 sample02
 4*total =   3.11035027369869
```

プロセス間通信: MPI_Send & MPI_Recv

Message Passing

MPI: MPI_Send & MPI_Recv



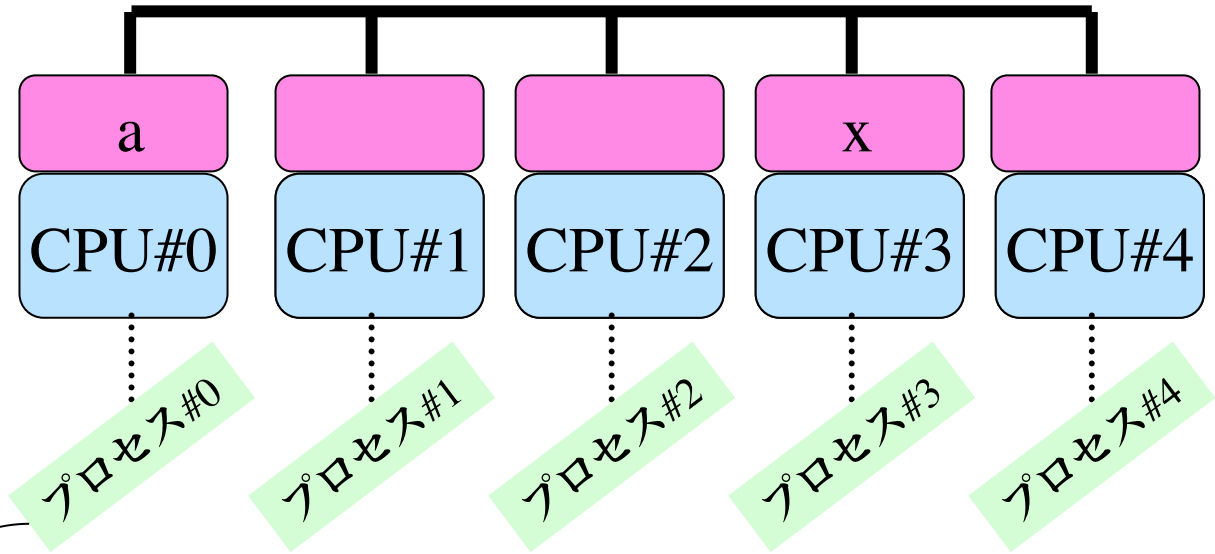
a = a + x

・プロセス#0は a の値は持っている。
・しかし x の値は持っていない。

・プロセス#3はプロセス#0に x の値を送る
　→　MPI_Send

・プロセス#0はプロセス#3から送られた x の値を受け取る
　→　MPI_Recv

MPI: MPI_Send & MPI_Recv



a = a + x

・プロセス#0は a の値は持っている。
・しかし x の値は持っていない。

- プロセス#3はプロセス#0に x の値を送る
  → MPI_Send
- プロセス#0はプロセス#3から送られた x の値を受け取る
  → MPI_Recv

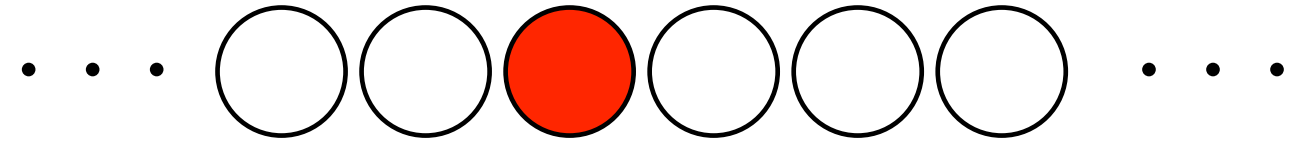
この二つは必ずペアでなければいけない。

MPI_Send&MPI_Recvのサンプル

横一列の赤白帽子ゲーム

生徒が横一列に並ぶ。表が白、裏が赤の帽子をかぶる。

先生が「パチッ」と手をたたいた瞬間に、あるルールに従って自分の帽子をひっくり返す（またはひっくり返さない）。



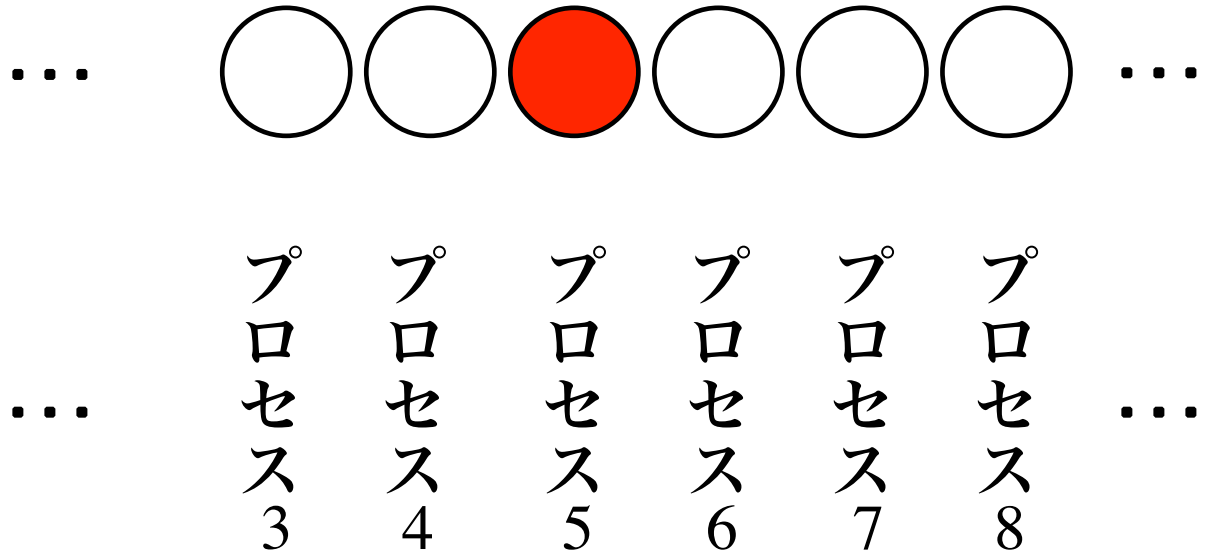
ルール：自分の左隣の人の帽子の色と自分の帽子の色が
違っている場合、自分が被っている帽子を裏返す。
それ以外の場合はなにもしない。

横一列帽子ゲーム

一つのプロセスが一人の人間を表す。
- 自分の帽子の色を記憶する。
- 帽子の色を変える／変えないべきかを判断する。

→　通信が必要

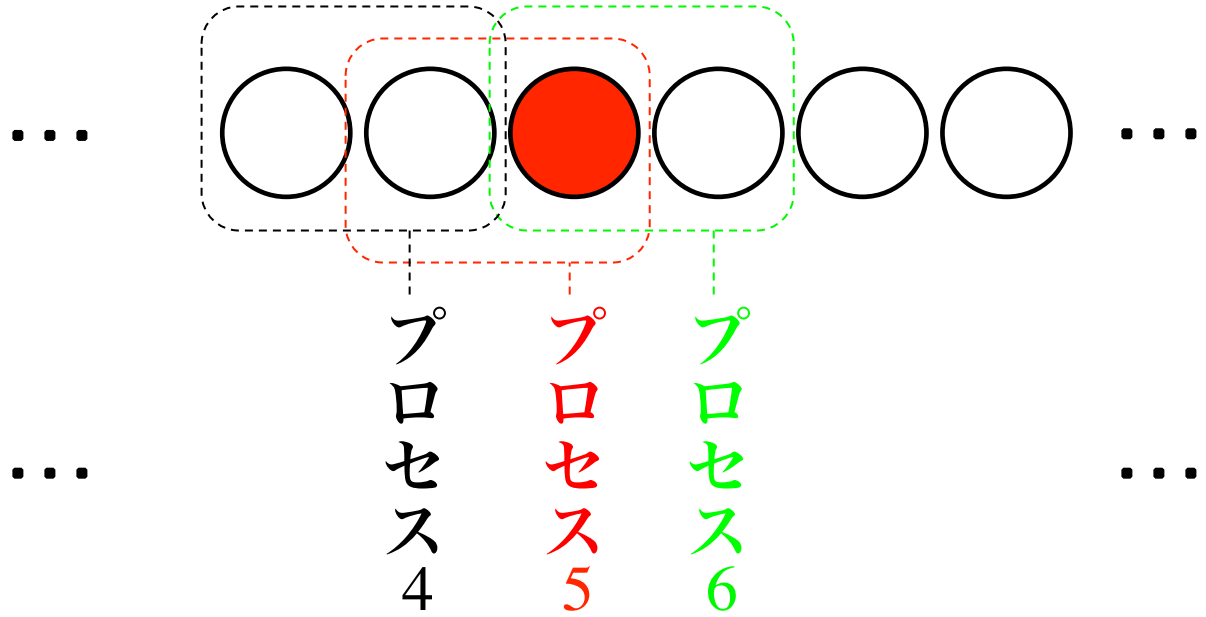
- 必要があれば自分の帽子の色を変える。



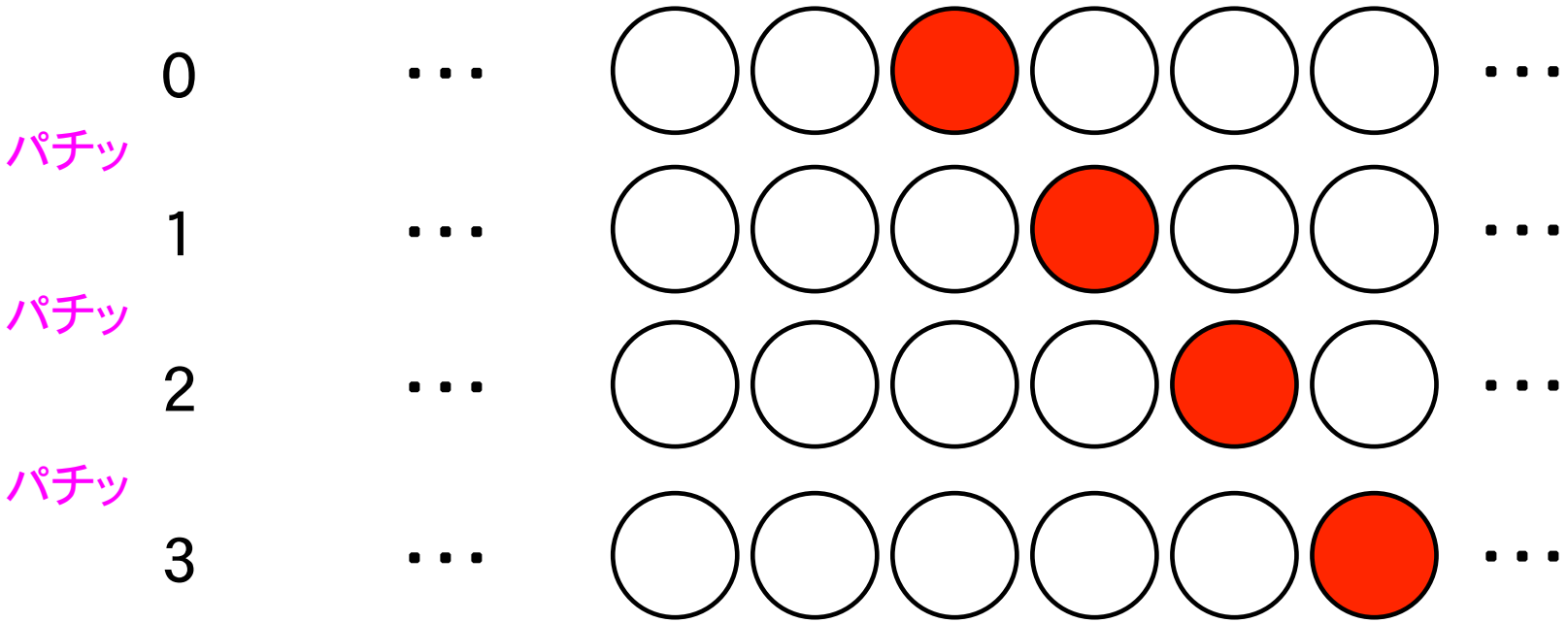
横一列帽子ゲーム

ルール：自分の左隣の人の帽子の色と自分の帽子の色が
違っている場合、自分が被っている帽子を裏返す。
それ以外の場合はなにもしない。

「左隣」の帽子の色を知らせてもらう。



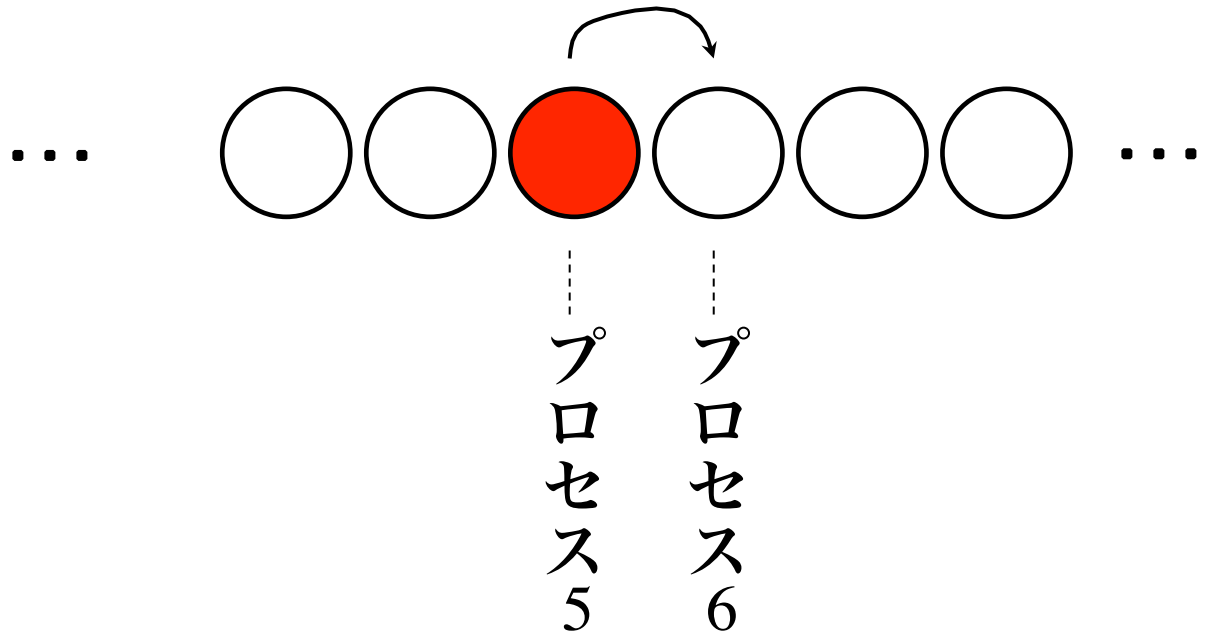
結果はわかっている



MPIプログラム

MPI_SendとMPI_Recvはペアでなければいけない。

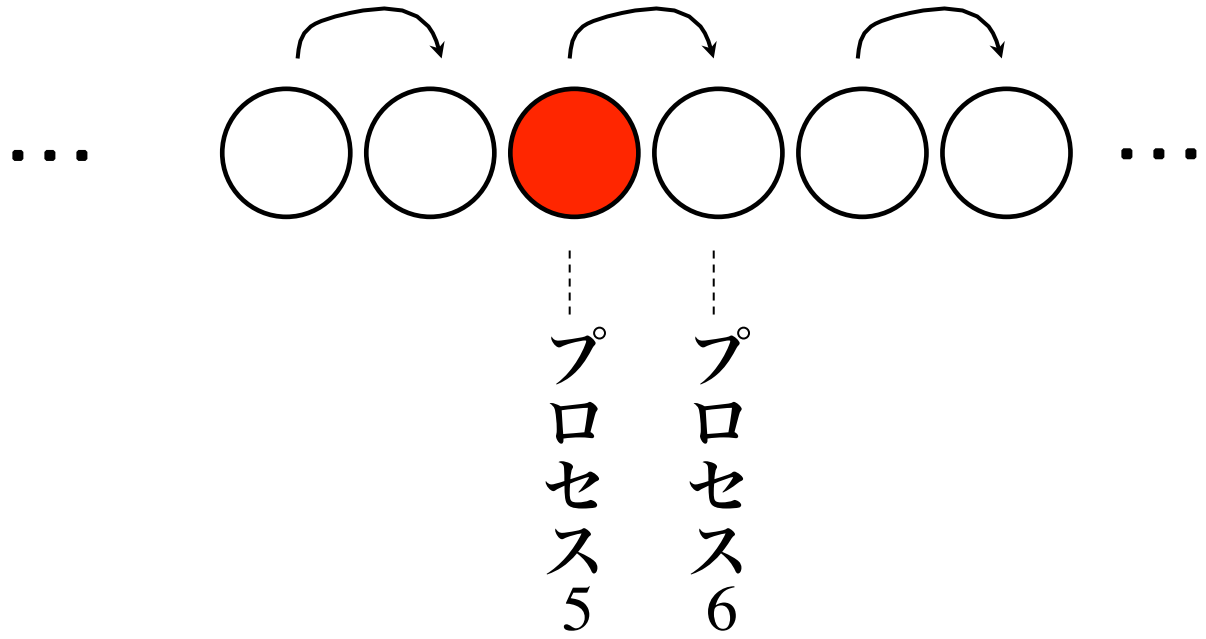
・プロセス５はプロセス６に送る（MPI_Send）
・プロセス６はプロセス５から受けとる（MPI_Recv）



MPIプログラム

MPI_SendとMPI_Recvはペアでなければいけない。

・奇数番号のプロセスは偶数番号のプロセスに送る（MPI_Send）
・偶数番号のプロセスは奇数番号のプロセスから受けとる（MPI_Recv）



sample03.f90 (1/6)

source: MPI/sample03.f90

```
 0  program sample03
 1    use constants
 2  ! use mpi  [or include 'mpif.h']
 3    implicit none
 4    include 'mpif.h'

 5    integer :: numprocs ierr
 6    integer :: wcomm = MPI_COMM_WORLD

 7    integer, dimension(MPI_STATUS_SIZE) :: status

 8    type ranks_
 9       integer :: me, right, left
10    end type ranks_

11    type(ranks_) :: ranks
```

sample03.f90 (2/6)

```
23    if ( mod(numprocs,2) == 1 ) then
24       print *, ' Numprocs must be even, to make pairs.'
25       call MPI_FINALIZE(ierr)
26    else if ( numprocs > MAX_PROCESS_NUMBER ) then
27       print *, ' Increase MAX_PROCESS_NUMBER.'
28       call MPI_FINALIZE(ierr)
29    end if

30    ranks%right = ranks%me + 1
31    if ( ranks%right == numprocs ) ranks%right = 0   ! periodic
32    ranks%left  = ranks%me - 1
33    if ( ranks%left == -1 ) ranks%left = numprocs-1  ! periodic
```

sample03.f90 (3/6)

```
34    if ( ranks%me==0 ) then
35       cap_color%me = 1 ! red
36    else
37       cap_color%me = 0 ! white
38    end if

39    call iCollectAndPrintAllCaps

40    do time_step_counter = 1 , 100
41       call iTellCapColorToNeighbors
42       call iFlipCapIfNecessary
43       call iCollectAndPrintAllCaps
44    end do

45    call MPI_FINALIZE(ierr)

46  contains
```

sample03.f90 (4/6)

```
47    !------------------------------------
48    subroutine iCollectAndPrintAllCaps
49    !------------------------------------
50      character(len=MAX_PROCESS_NUMBER) :: &
                                  cap_state_by_string
51      if ( ranks%me == 0 ) then
52         cap_state_by_string(1:1)          &
53              = iConvertToChar(cap_color%me)
54         do source = 1 , numprocs-1
55            ! Receive data sent from others.
56            call MPI_RECV(color_recv_buff,                    &
57                          1, MPI_INTEGER, source,             &
58                          MPI_ANY_TAG, wcomm,    &
                                status, ierr)
59            cap_state_by_string(source+1:source+1)            &
60                 = iConvertToChar(color_recv_buff)
61         end do
62         print *, cap_state_by_string(1:numprocs)
63      else  ! Send each color to the           &
              ! master process (rank=0).
64         call MPI_SEND(cap_color%me,           &
                        1, MPI_INTEGER, 0,      &
65                      0, wcomm, ierr)
66      end if

67    end subroutine iCollectAndPrintAllCaps
```

sample03.f90 (5/6)

```
69    !--------------------------
70    function iConvertToChar(i)
71    !--------------------------
72      integer, intent(in) :: i
73      character :: iConvertToChar

74      if (i==0) then
75         iConvertToChar = '.'
76      else
77         iConvertToChar = '#'
78      end if
79    end function iConvertToChar

80    !------------------------------
81    subroutine iFlipCapIfNecessary
82    !------------------------------
83      if ( cap_color%me /= cap_color%left ) then
84         cap_color%me = 1-cap_color%me   ! 1-->0 and 0-->1.
85      else
86         ! do nothing.
87      end if
88    end subroutine iFlipCapIfNecessary
```

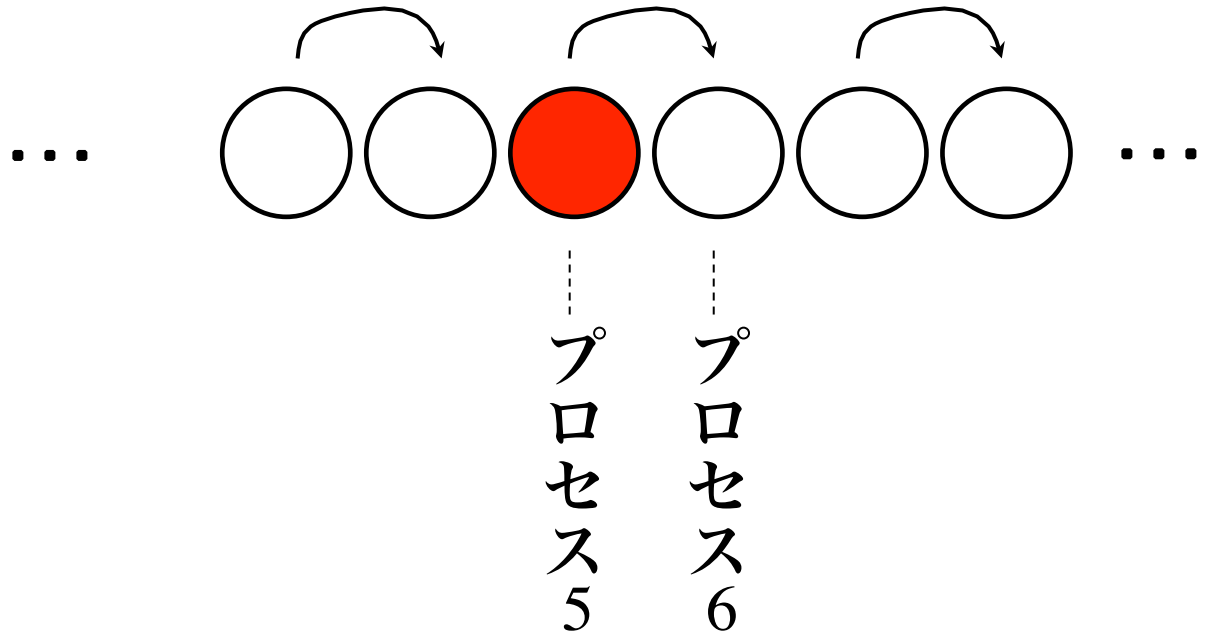
sample03.f90 (6/6)

```
 89    !------------------------------------
 90    subroutine iTellCapColorToNeighbors
 91    !------------------------------------
 92      ! Send each color data to the right neighbor.
 93      if ( mod(ranks%me,2)==0 ) then
 94         ! If you are an even process,
 95         ! send to your right (odd process),
 96         ! then receive from your left (also odd process).
 97         call MPI_SEND(cap_color%me,                          &
 98                       1, MPI_INTEGER, ranks%right,           &
 99                       0, wcomm, ierr)
100         call MPI_RECV(cap_color%left,                        &
101                       1, MPI_INTEGER, ranks%left,            &
102                       MPI_ANY_TAG, wcomm, status, ierr)
103      else
104         ! If you are on odd process,
105         ! receve from your (even) left,
106         ! then send your data to your right (even) neighbor.
107         call MPI_RECV(cap_color%left,                        &
108                       1, MPI_INTEGER, ranks%left,            &
109                       MPI_ANY_TAG, wcomm, status, ierr)
110         call MPI_SEND(cap_color%me,                          &
111                       1, MPI_INTEGER, ranks%right,           &
112                       0, wcomm, ierr)
113      end if
114    end subroutine iTellCapColorToNeighbors

115  end program sample03
```

MPIプログラム

MPI_SendとMPI_Recvはペアでなければいけない。

・奇数番号のプロセスは偶数番号のプロセスに送る（MPI_Send）
・偶数番号のプロセスは奇数番号のプロセスから受けとる（MPI_Recv）

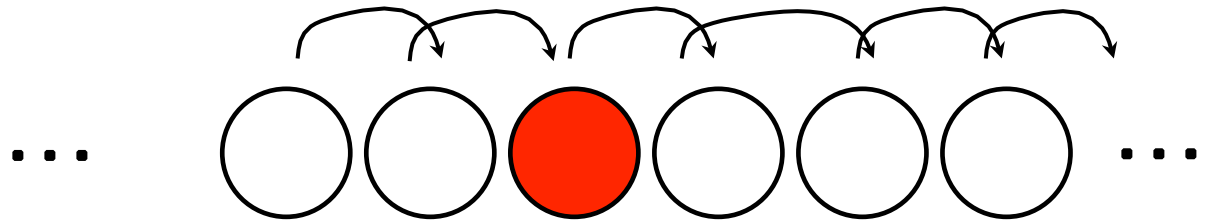


MPIプログラム

・全てのプロセスが右隣のプロセスに送る（MPI_Send）

「そして」

・全てのプロセスから左隣のプロセスから受けとる（MPI_Recv）



無理。デッドロック。

デッドロックの例

source: MPI/sample03_deadlock.f90

```
call MPI_INIT(ierr)
call MPI_COMM_RANK(wcomm, ranks%me, ierr)
call MPI_COMM_SIZE(wcomm, numprocs, ierr)

.
.
.

! deadlock !
call MPI_SEND(cap_color%me,                    &
              1, MPI_INTEGER, ranks%right,     &
              0, wcomm, ierr)
call MPI_RECV(cap_color%left,                  &
              1, MPI_INTEGER, ranks%left,      &
              MPI_ANY_TAG, wcomm, status, ierr)
```

デッドロックの回避

非同期通信 ISEND, IRECV

```
.
.
.

call MPI_ISEND(...)

call MPI_RECV(...)

call MPI_WAITALL(...)
```

メッセージの送信を発行して
次のステップに移る。

送受信の完了を確認する

デッドロックの回避

非同期通信 ISEND, IRECV

```
.
.
.
call MPI_ISEND(...)
call MPI_IRECV(...)


call MPI_WAITALL(...)
```

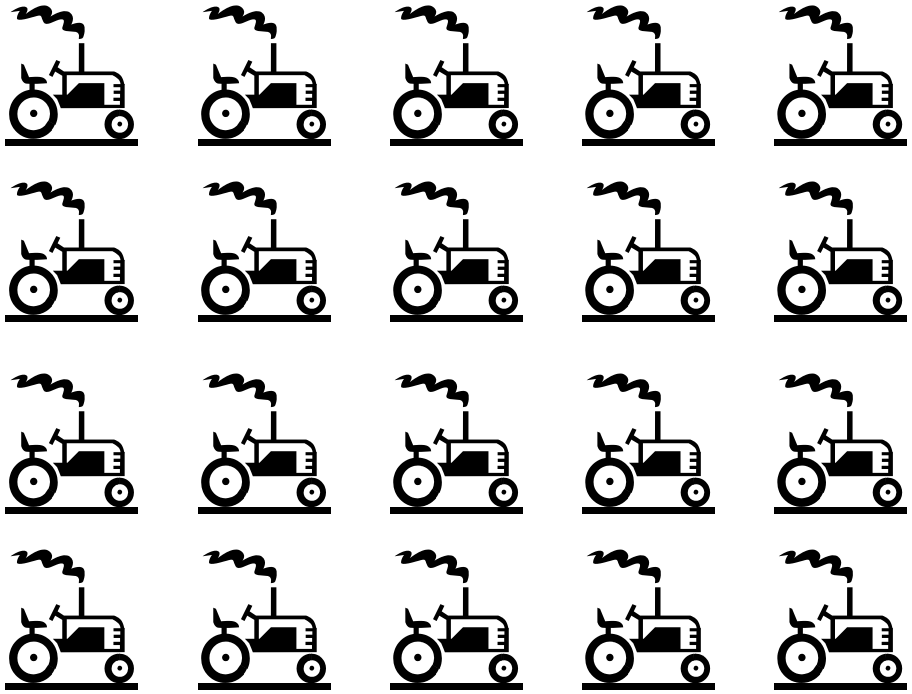
次のステップに移る。

次のステップに移る。

送受信の完了を確認する

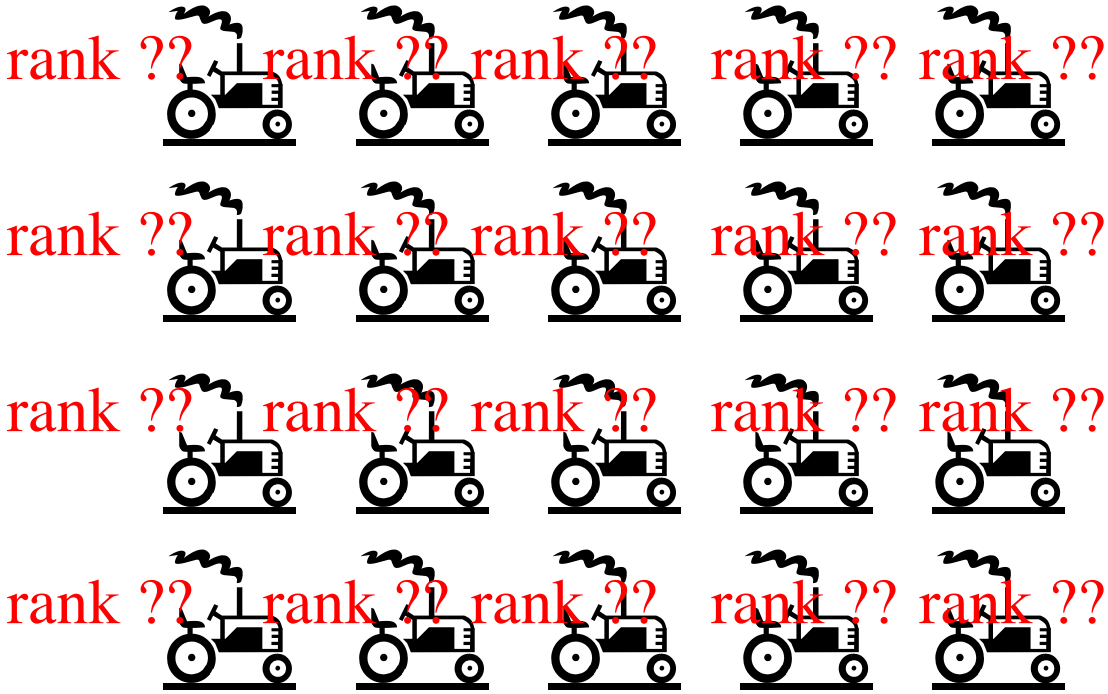
その他のMPI関数

- MPI_Sendrecv
  - 送信・受信を同時に指定
- MPI_Cart_create
  - ２次元、３次元にプロセスを配置するのに便利。



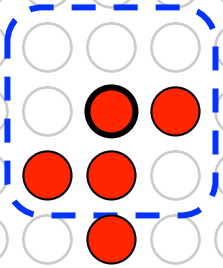
その他のMPI関数

- MPI_Sendrecv
  - 送信・受信を同時に指定
- MPI_Cart_create
  - ２次元、３次元にプロセスを配置するのに便利。



サンプル

- 縦横並びの帽子ゲーム（Game of Life）



ルール：
まわりの８人（前後、左右、斜め隣の人）が
かぶっている赤い帽子の数を合計して...

１．自分の帽子が赤のとき、それが
　　・２個か３個だったらそのまま、
　　・それ以外は裏返す（白にする）。
２．自分の帽子が白のとき、それが
　　・３個だったら裏返す（赤にする）。
　　・それ以外はそのまま。

一つの「人間（帽子状態）」を一つのプロセスが担当
source codes: MPI/Sample04/

Sample04/main.f90

```
15  program sample04
16  !=======================================================================
17  ! PROGAM SAMPLE04                                    A Simple Sample of MPI Program
18  !=======================================================================
19  !                                          by Akira Kageyama (kage@jamstec.go.jp)
20  !-----------------------------------------------------------------------
21    use constants
22  ! use mpi            ! [or include 'mpif.h']
23    use parallel
24    use cap
25    implicit none
26    include 'mpif.h'   ! [or use mpi]

27    type(cap__color_) :: cap_color

28    integer :: counter

29    call parallel__initialize
30
31    call iSetInitialCondition

32    if ( parallel__get_myrank() == 0 ) print *, '# step = ', 0
33    call parallel__barrier
34    if ( cap_color%me == CAP__RED ) print *, parallel__get_mycoord_x(),   &
35                                             parallel__get_mycoord_y()
36
```

Sample04/main.f90

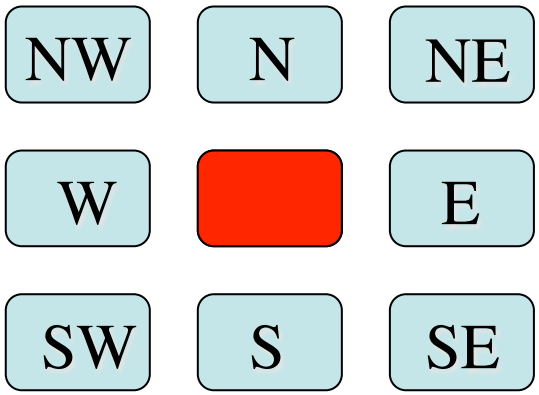
```
37    do counter = 1 , 20
38       call parallel__communicate(cap_color)
39       call cap__flip_my_color(cap_color)
40       if ( parallel__get_myrank() == 0 ) print *, '# step = ', counter
41       call parallel__barrier
42       if ( cap_color%me == CAP__RED ) print *, parallel__get_mycoord_x(),  &
43                                                parallel__get_mycoord_y()
44    end do

45    call parallel__finalize

46  contains
```

Sample04/main.f90

```
47    !-------------------------------------------------------------------
48    subroutine iSetInitialCondition
49    !-------------------------------------------------------------------
50      if (       parallel__get_mycoord_x()==0                 &
51           .and. parallel__get_mycoord_y()==1 ) then
52         cap_color%me = CAP__RED
53      else if ( parallel__get_mycoord_x()==1                  &
54           .and. parallel__get_mycoord_y()==1 ) then
55         cap_color%me = CAP__RED
56      else if ( parallel__get_mycoord_x()==2                  &
57           .and. parallel__get_mycoord_y()==1 ) then
58         cap_color%me = CAP__RED
59      else if ( parallel__get_mycoord_x()==0                  &
60           .and. parallel__get_mycoord_y()==0 ) then
61         cap_color%me = CAP__RED
62      else
63         cap_color%me = CAP__WHITE
64      end if

65    end subroutine iSetInitialCondition

66  end program sample04
```

parallel.f90

```
 7  module parallel
11    use constants
12    use cap
13  ! use mpi   [or include 'mpif.h']
14    implicit none
15    include 'mpif.h'
16    private
17    public :: parallel__barrier,           &
18              parallel__communicate,       &
19              parallel__initialize,        &
20              parallel__finalize,          &
21              parallel__get_mycoord_x,     &
22              parallel__get_mycoord_y,     &
23              parallel__get_myrank

24    type ranks_
25       integer :: me
26       integer :: north, south, west, east
27       integer :: north_east, south_east
28       integer :: north_west, south_west
29    end type ranks_

30    type(ranks_)           :: Ranks
31    integer                :: Numprocs
32    integer                :: Communicator
33    integer                :: CapsComm
34    integer, dimension(2)  :: Coords
```

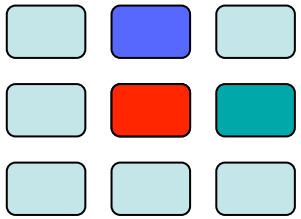


parallel.f90

```
156     !
157     !=============================================<public>=====
158     subroutine parallel__initialize
159     !============================================================
160     !
161       integer :: ierr
162       integer, dimension(2) :: dims
163       logical, dimension(2) :: is_periodic

164       logical :: reorder
165       integer :: ndim, sqrt_numprocs

166       call MPI_INIT(ierr)
167       call MPI_COMM_RANK(MPI_COMM_WORLD, Ranks%me, ierr)
168       call MPI_COMM_SIZE(MPI_COMM_WORLD, Numprocs, ierr)

169       sqrt_numprocs = nint(sqrt(real(Numprocs,DP)))

170       if ( sqrt_numprocs**2 /= Numprocs ) then
171          if ( Ranks%me == 0 ) print *, ' Numprocs must be a squared int.'
172          call MPI_FINALIZE(ierr)
173          stop
174       end if
```

parallel.f90

```
175       dims(1) = sqrt_numprocs
176       dims(2) = sqrt_numprocs
177       is_periodic(1) = .true.
178       is_periodic(2) = .true.
179       reorder = .true.
180       ndim = 2

181       call MPI_CART_CREATE(MPI_COMM_WORLD, ndim, dims, is_periodic,        &
182                            reorder, CapsComm, ierr)

183       call MPI_CART_SHIFT(CapsComm, 0,           1,                        &
184                                     Ranks%west,  Ranks%east,  ierr)
185       call MPI_CART_SHIFT(CapsComm, 1,           1,                        &
186                                     Ranks%north, Ranks%south, ierr)
187       call MPI_CART_COORDS(CapsComm, Ranks%me, 2, Coords, ierr)

188       call dataTransferToEast(Ranks%north, Ranks%north_west)
189       call dataTransferToEast(Ranks%south, Ranks%south_west)
190       call dataTransferToWest(Ranks%north, Ranks%north_east)
191       call dataTransferToWest(Ranks%south, Ranks%south_east)

192     end subroutine parallel__initialize
```



parallel.f90

```
37    !
38    !-----------------------------------------------<private>----
39    subroutine dataTransferToEast(sent_value, recv_value)
40      integer, intent(in)  :: sent_value
41      integer, intent(out) :: recv_value
42    !------------------------------------------------------------
43    !
44      integer :: ierr
45      integer, dimension(MPI_STATUS_SIZE) :: status

46      if ( mod(Coords(1),2)==0 ) then
47         call MPI_SEND(sent_value, 1, MPI_INTEGER, Ranks%east,    &
48                       0, CapsComm, ierr)
49         call MPI_RECV(recv_value, 1, MPI_INTEGER, Ranks%west,    &
50                       MPI_ANY_TAG, CapsComm, status, ierr)
51      else
52         call MPI_RECV(recv_value, 1, MPI_INTEGER, Ranks%west,    &
53                       MPI_ANY_TAG, CapsComm, status, ierr)
54         call MPI_SEND(sent_value, 1, MPI_INTEGER, Ranks%east,    &
55                       0, CapsComm, ierr)
56      end if

57    end subroutine dataTransferToEast
```

parallel.f90

```
58    !
59    !-----------------------------------------------<private>----
60    subroutine dataTransferToWest(sent_value, recv_value)
61      integer, intent(in)  :: sent_value
62      integer, intent(out) :: recv_value
63    !-------------------------------------------------------------
64    !
65      integer :: ierr
66      integer, dimension(MPI_STATUS_SIZE) :: status

67      if ( mod(Coords(1),2)==0 ) then
68         call MPI_SEND(sent_value, 1, MPI_INTEGER, Ranks%west,   &
69                       0, CapsComm, ierr)
70         call MPI_RECV(recv_value, 1, MPI_INTEGER, Ranks%east,   &
71                       MPI_ANY_TAG, CapsComm, status, ierr)
72      else
73         call MPI_RECV(recv_value, 1, MPI_INTEGER, Ranks%east,   &
74                       MPI_ANY_TAG, CapsComm, status, ierr)
75         call MPI_SEND(sent_value, 1, MPI_INTEGER, Ranks%west,   &
76                       0, CapsComm, ierr)
77      end if

78    end subroutine dataTransferToWest
```

parallel.f90

```
 87    !
 88    !=============================================<public>=====
 89    subroutine parallel__communicate(cap_color)
 90      type(cap__color_), intent(inout) :: cap_color
 91    !=============================================================
 92    !
 93      integer :: ierr
 94      integer, dimension(MPI_STATUS_SIZE) :: status

 95      type tag_
 96         integer :: north, north_east, east, south_east
 97         integer :: south, south_west, west, north_west
 98      end type tag_
 99
100      type(tag_) :: tag

101      tag%north      = 0
102      tag%north_east = 1
103      tag%east       = 2
104      tag%south_east = 3
105      tag%south      = 4
106      tag%south_west = 5
107      tag%west       = 6
108      tag%north_west = 7
```

parallel.f90

```
109       call MPI_SENDRECV(cap_color%me,   1, MPI_INTEGER,     &
110                         Ranks%north, tag%north,             &
111                         cap_color%south, 1, MPI_INTEGER,    &
112                         Ranks%south, tag%north,             &
113                         CapsComm, status, ierr)

114       call MPI_SENDRECV(cap_color%me,        1, MPI_INTEGER,         &
115                         Ranks%north_east, tag%north_east,            &
116                         cap_color%south_west, 1, MPI_INTEGER,        &
117                         Ranks%south_west, tag%north_east,            &
118                         CapsComm, status, ierr)
119
120       call MPI_SENDRECV(cap_color%me,   1, MPI_INTEGER,     &
121                         Ranks%east,  tag%east,              &
122                         cap_color%west, 1, MPI_INTEGER,     &
123                         Ranks%west,  tag%east,              &
124                         CapsComm, status, ierr)
125
126       call MPI_SENDRECV(cap_color%me,        1, MPI_INTEGER,         &
127                         Ranks%south_east, tag%south_east,            &
128                         cap_color%north_west, 1, MPI_INTEGER,        &
129                         Ranks%north_west, tag%south_east,            &
130                         CapsComm, status, ierr)
```

parallel.f90

```
132       call MPI_SENDRECV(cap_color%me,   1, MPI_INTEGER,     &
133                         Ranks%south,  tag%south,            &
134                         cap_color%north, 1, MPI_INTEGER,    &
135                         Ranks%north,  tag%south,            &
136                         CapsComm, status, ierr)
137
138       call MPI_SENDRECV(cap_color%me,        1, MPI_INTEGER,         &
139                         Ranks%south_west, tag%south_west,            &
140                         cap_color%north_east, 1, MPI_INTEGER,        &
141                         Ranks%north_east, tag%south_west,            &
142                         CapsComm, status, ierr)
143
144       call MPI_SENDRECV(cap_color%me,   1, MPI_INTEGER,     &
145                         Ranks%west,  tag%west,              &
146                         cap_color%east, 1, MPI_INTEGER,     &
147                         Ranks%east,  tag%west,              &
148                         CapsComm, status, ierr)
149
150       call MPI_SENDRECV(cap_color%me,        1, MPI_INTEGER,         &
151                         Ranks%north_west, tag%north_west,            &
152                         cap_color%south_east, 1, MPI_INTEGER,        &
153                         Ranks%south_east, tag%north_west,            &
154                         CapsComm, status, ierr)

155     end subroutine parallel__communicate
```

cap.f90

```
 7  module cap
 8  !=====================================================================
 9  ! MODULE CAP
10  !=====================================================================
11    use constants

13    implicit none
14    private
15    public :: cap__flip_my_color
16    public :: cap__color_
17    public :: CAP__RED,           &
18              CAP__WHITE

19    type cap__color_
20       integer :: me  ! 1 (red) or 0 (white)
21       integer :: north, south, west, east
22       integer :: north_east, south_east
23       integer :: north_west, south_west
24    end type cap__color_

25    integer, parameter :: CAP__WHITE = 0
26    integer, parameter :: CAP__RED   = 1
```

cap.f90

```
45    !=============================================<public>=====
46    subroutine cap__flip_my_color(cap_color)
47      type(cap__color_), intent(inout) :: cap_color
48    !=============================================================
49    !
50      integer :: red_cap_num

51      red_cap_num = count_red(cap_color)

52      select case (cap_color%me)
53      case ( CAP__RED )
54         if ( red_cap_num==2 .or. red_cap_num==3 ) then
55            ! do nothing
56         else
57            cap_color%me = 1 - cap_color%me
58         end if
59      case ( CAP__WHITE )
60         if ( red_cap_num==3 ) then
61            cap_color%me = 1 - cap_color%me
62         else
63             ! do nothing
64        end if
65      end select

66    end subroutine cap__flip_my_color
```

cap.f90

```
29    !-----------------------------------------------<private>----
30    function count_red(cap_color)
31      type(cap__color_), intent(in) :: cap_color
32      integer :: count_red
33    !-----------------------------------------------------------
34    !
35      count_red = cap_color%north         &
36                + cap_color%north_east    &
37                + cap_color%east          &
38                + cap_color%south_east    &
39                + cap_color%south         &
40                + cap_color%south_west    &
41                + cap_color%west          &
42                + cap_color%north_west

43    end function count_red
```