Empowered by Innovation NEC

F2008の並列処理関連機能

2012年4月

NEC
林　康晴

内容

- Fortran 2008の並列処理関連機能
- coarray機能の概要
- データの整合性を保証するための手段
- 次期Fortran規格に入る「かもしれない」機能
- F2008のcoarray機能の問題点のいくつか
- DO CONCURRENT構文
- coarray機能とHPF
- まとめ

Fortran 2008の並列処理関連機能

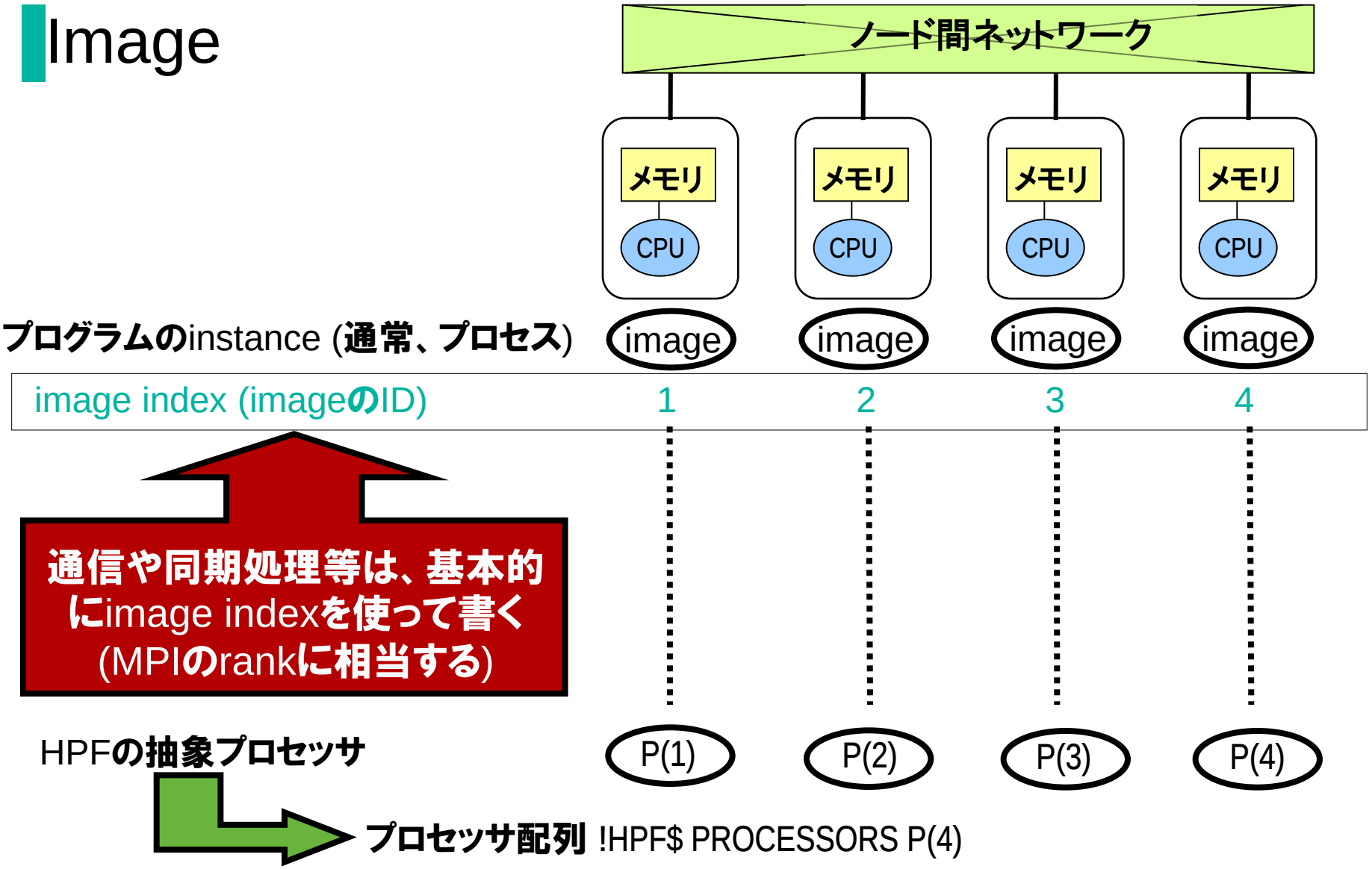
2010年、最新のISO規格に

新たな並列処理関連機能

- coarray機能
  - 主として分散並列がターゲット
  - (MPIのような)ローカルモデル・全手動の並列化
- DO CONCURRENT構文
  - INDEPENDENTループとほぼ等価な仕様
  - Fortran仕様内では、主として共有並列がターゲット

coarray機能の概要

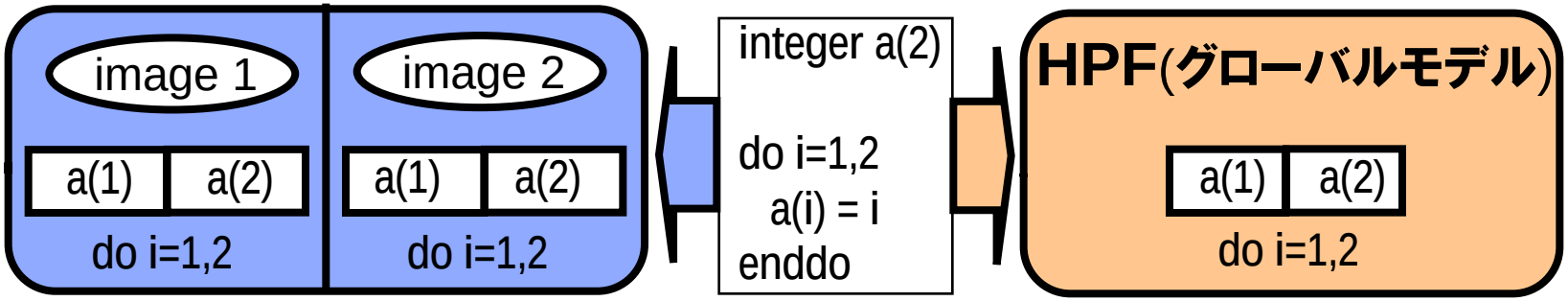
Image



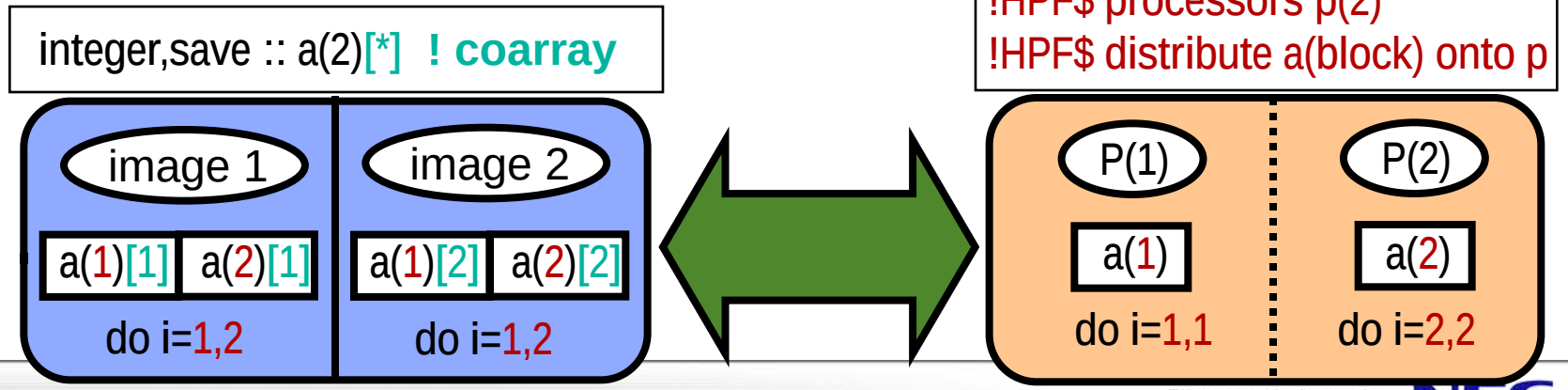
coarray機能の概要

データマッピングと並列化

- データ・処理は、各image上にそれぞれ割当てられる
  - 各image上のデータ・処理は、独立した別物(ローカルモデル)



- *image selector*([i])を使って、他のimage i上のデータにアクセスできる(通信)

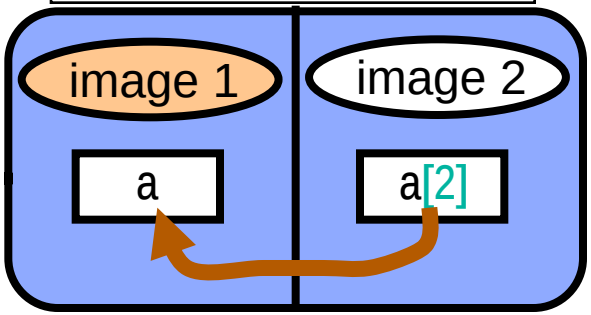


coarray機能の概要

通信(片側)

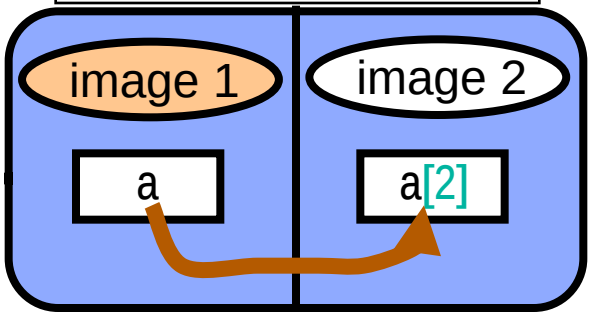
get

```
integer,save :: a[*]

if(this_image().eq.1)then
  a = a[2]
endif
```



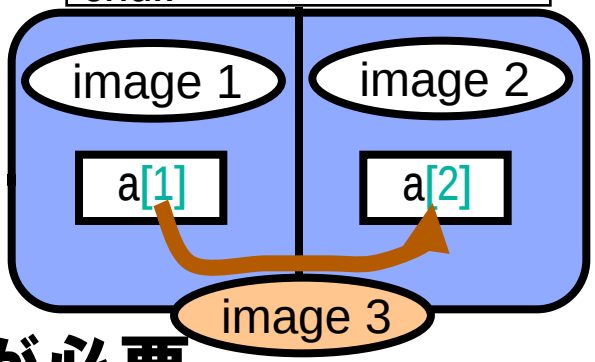
put

```
integer,save :: a[*]

if(this_image().eq.1)then
  a[2] = a
endif
```



第三者による操作

```
integer,save :: a[*]

if(this_image().eq.3)then
  a[2] = a[1]
endif
```



- **データの整合性を保つための同期が必要**

同期	sync all, sync images((/1,2,.../)), sync team(TEAM)
片方向同期	notify((/1,2,.../)), query((/1,2,.../))
実行完了の保証	sync memory
1 image**毎の実行**	CRITICAL**構文**
ロック・アンロック	LOCK**文**, UNLOCK**文**

議論中
(WG5)

coarray機能の概要

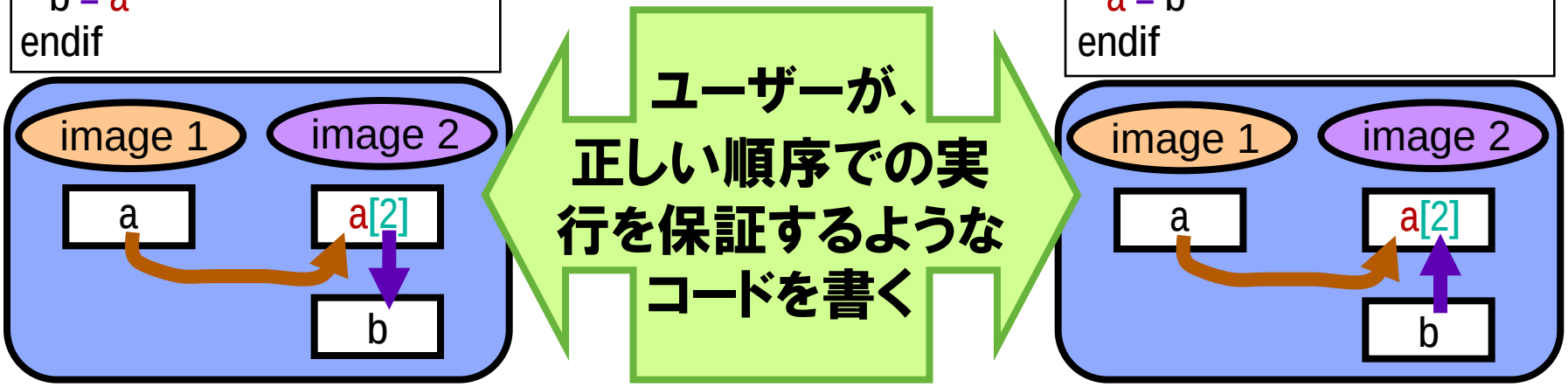
データの整合性は、ユーザが保証する必要がある

定義と使用の競合

```
integer,save :: b, a[*]

if(this_image().eq.1)then
  a[2] = a
else if(this_image().eq.2)then
  b = a
endif
```

定義と定義の競合

```
integer,save :: b, a[*]

if(this_image().eq.1)then
  a[2] = a
else if(this_image().eq.2)then
  a = b
endif
```



➢ **データの整合性を保証するための手段**

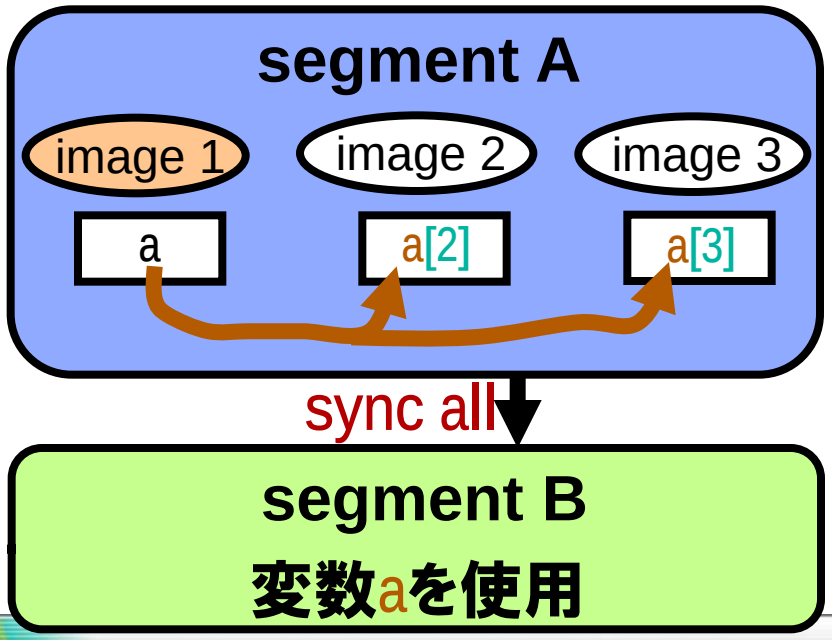
- Fortran**の同期機能**(image control statements)
- Fortran**以外の同期機能** + SYNC MEMORY**文**
- atomic subroutine + SYNC MEMORY**文**

データの整合性を保証するための手段 1/7

Fortranの同期機能 1/4

- SYNC ALL文
  - 全imageで同期を取る。
  - ALLOCATABLE属性を持つcoarrayを明示的 又は 暗黙的に割付け・解放する文(RETURN文等)も同等の効果を持つ

例: segment Bは、segment Aの実行後に実行される。




```
integer,save :: a[*]  ! coarray

if(this_image().eq.1)then
   do i=2, num_images()
     a[i] = a ! 全imageのaの値を定義
   enddo
endif
sync all
! 各imageがaを使用
        :
```

segment A

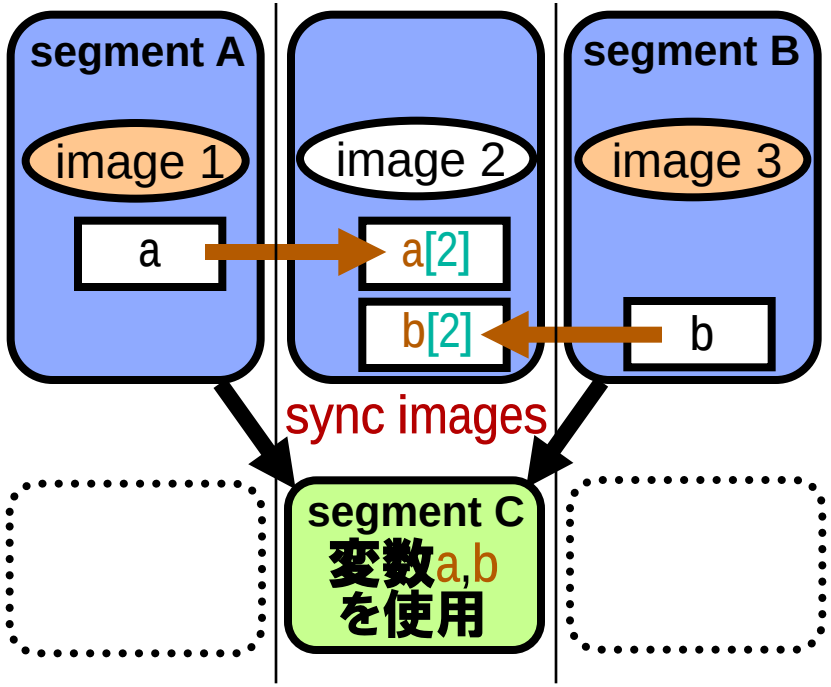
segment B

データの整合性を保証するための手段 2/7

Fortranの同期機能 2/4

- SYNC IMAGES文
  - 指定したimageと同期を取る。

例: segment Cは、segment A・Bの実行後に実行される。



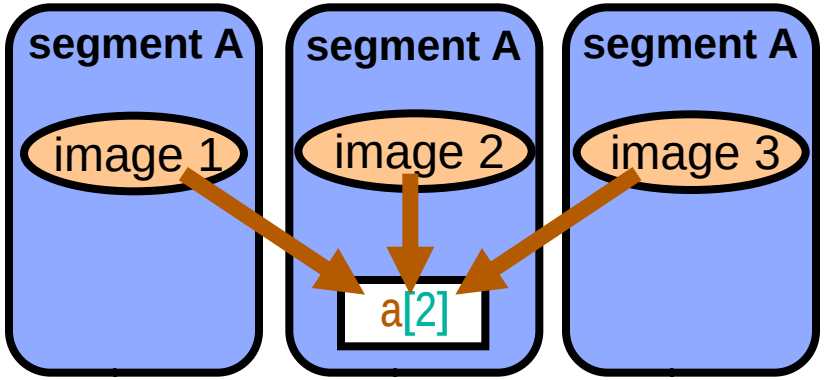

```
integer,save :: b[*], a[*]  ! coarray
if(this_image().eq.1)then                 } segment A
   a[2] = a
   sync images(2)
else if(this_image().eq.3)then            } segment B
   b[2] = b
   sync images(2)
else if(this_image().eq.2)then
   sync images((/1, 3/))
   変数a, bを使用                          } segment C
endif
```

データの整合性を保証するための手段 3/7

Fortranの同期機能 3/4

- LOCK文・UNLOCK文
  - 指定した変数(ロック変数)の、ロック・アンロック操作を行う
  - ロック変数は、LOCK_TYPE型でなければならない
    - 組込みモジュールISO_FORTRAN_ENVで定義されている

例: **segment Aは、ロックを獲得したimageだけが実行する**




```
use, intrinsic :: ISO_FORTRAN_ENV
type(LOCK_TYPE),save :: lock[*]
integer,save :: a[*]

lock(lock[2])
  a[2] = a[2] + 1   } segment A
unlock(lock[2])
```

- lock(**ロック変数**, acquired_lock=**論理型変数**)

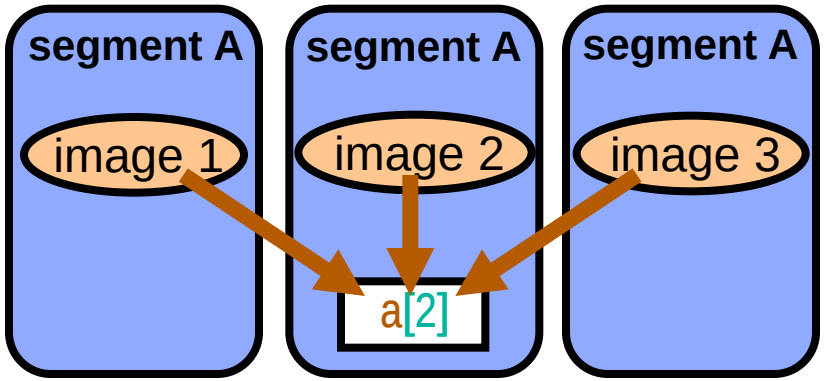
のようにACQUIRED_LOCK指定子をつけると、非ブロッキング実行となり、論理型変数が、ロックを獲得できた場合 真、出来なかった場合 偽となる

データの整合性を保証するための手段 4/7

Fortranの同期機能 4/4

- CRITICAL構文
  - 同時に実行するimageを1つに制限する。
    - コードの一部分に対するロックとも考えられる

例: segment Aは、各imageが逐次的に実行する。




```
integer,save :: a[*]  ! coarray

critical
  a[2] = a[2] + 1      } segment A
end critical
```

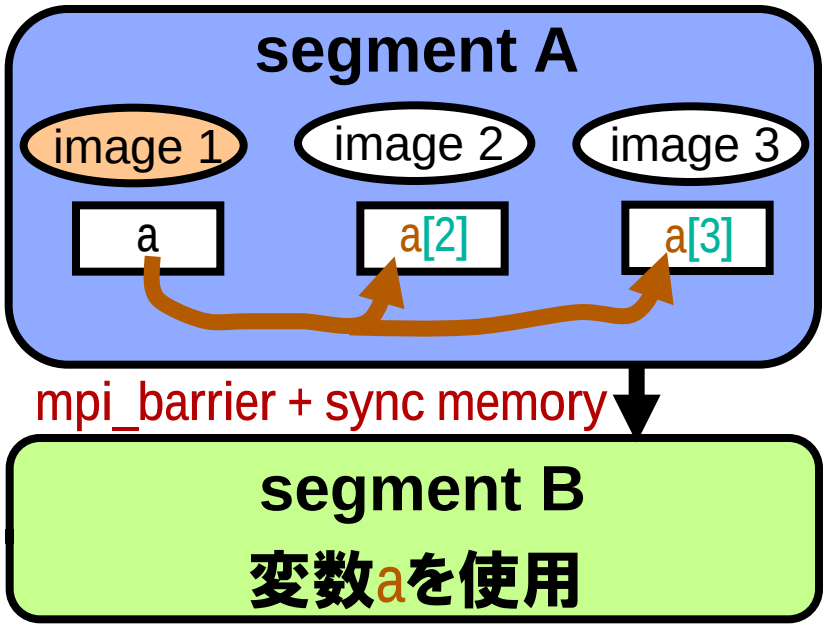
データの整合性を保証するための手段 5/7

Fortran以外の同期機能 + SYNC MEMORY文

- SYNC MEMORY文
  - segmentを分割し、コンパイラによるコード移動を制限する

例: segment Bは、segment Aの実行後に実行される。

  - sync memoryが無いと、コンパイラが、定義と使用の実行順序を変更する可能性があるため、正しい実行は保障されない。




```
integer,save :: a[*]  ! coarray
if(this_image().eq.1)then
   do i=2, num_images()
     a[i] = a ! 全imageのaの値を定義      } segment A
   enddo
endif
sync memory
call mpi_barrier(MPI_COMM_WORLD,ierr)
sync memory
! 各imageがaを使用                       } segment B
        :
```

データの整合性を保証するための手段 6/7

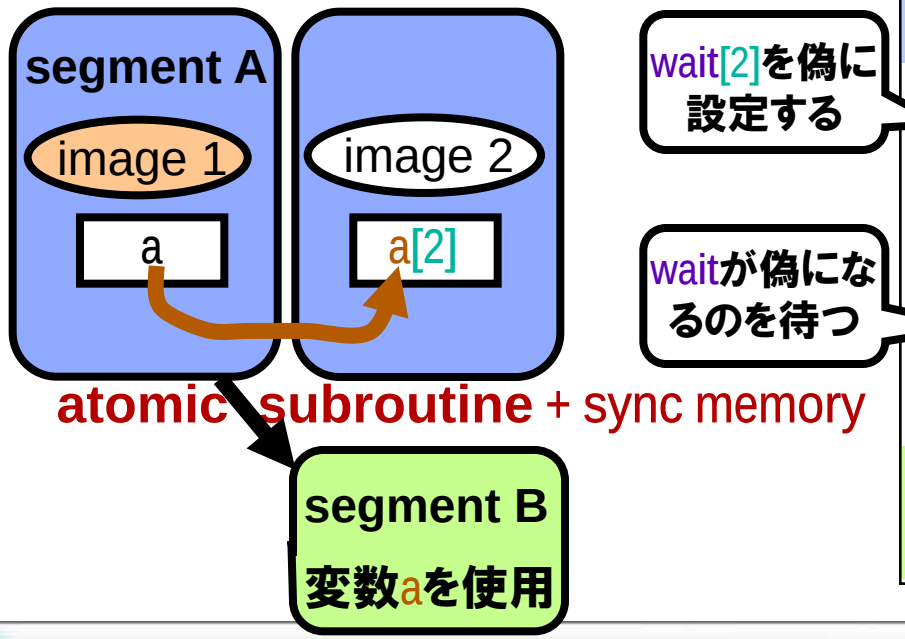
atomic subroutine (2種類)

- atomic_ref(VALUE, ATOM)
  - 引数ATOMの値を、引数VALUEにatomicに代入する
- atomic_define(ATOM, VALUE)
  - 引数VALUEの値を、引数ATOMにatomicに代入する
- ATOM引数は、以下のいずれかの型のcoarray
  - INTEGER(KIND = ATOMIC_INT_KIND)
  - LOGICAL(KIND = ATOMIC_LOGICAL_KIND)
  - ✓ ATOMIC_INT_KIND, ATOMIC_LOGICAL_KINDは、組込みモジュールISO_FORTRAN_ENVで定義されている
- 同じ引数ATOMに対するatomic subroutineの呼出しは、同時に実行される可能性があっても許される。
  - そのような場合、実行順序がどうなるかはもちろん分からない。

データの整合性を保証するための手段 7/7

atomic subroutine + SYNC MEMORY文

例: segment Bは、segment Aの実行後に実行される。




```
use, intrinsic :: ISO_FORTRAN_ENV
save
integer :: a[*]
logical :: val
logical(atomic_logical_kind) :: wait[*] = .true.
if(this_image().eq.1)then                    } segment A
  a[2] = a
  sync memory
  call atomic_define(wait[2], .false.)
else if(this_image().eq.2)then
  val = .true.
  do while(val)
    call atomic_ref(val, wait)
  enddo
  sync memory
  変数aを使用                                } segment B
endif
```

wait[2]を偽に設定する

waitが偽になるのを待つ

次期Fortran規格に入る「かもしれない」機能

現在WG5で議論中

- NOTIFY文・QUERY文
  - SYNC IMAGES文と同様、指定したimageと同期を取る。
  - 但し、QUERY文の実行側は、対応するNOTIFY文が実行されるのを待つが、NOTIFY文の実行側は待たない。
    - 非ブロッキングモードのQUERY文もある。
- 集計演算用組込み手続(総和・最大・最小など)
  - image間でcoarrayに対する集計演算を行う(co_sum, ...)
- TEAM (imageの部分集合)
  - 部分プロセス集合のコミュニケータ(MPI)に似ている
    - 但し image indexは、TEAM内ではなく、常に全imageにおけるID
  - TEAM上で、ファイルの共有や集計演算ができる。
    - I/Oは、direct access 又は WRITE専用のsequential accessのみ

F2008のcoarray機能の問題点のいくつか

データマッピング

- coarrayは、全image上で、同一の上下限値(等サイズ)でなければならない
  - image毎に異なるサイズのデータを使用したい場合、スカラの派生型coarrayを宣言し、その成分をALLOCATABLEにするしかない

```
type perimage
  real, allocatable :: a(:)
end type
type(perimage), save :: g[*]   ! 割付け配列を成分として持つcoarray
allocate(g%a(this_image()*100)) ！ 成分は、image毎にサイズが異なってよい
```

I/O

- 複数のimageでファイルを共有する方法がない。
  - 複数のimageが、同じ名前でopenしたファイルが同一かどうかは処理系依存。
  - 事前接続されているファイルは、image毎に別のファイル

F2008のcoarray機能の問題点のいくつか

- **通信性能は、プログラムの書き方に大きく依存する可能性が高い**
  - **例えば、以下のような通信パタン(broadcast?)をコンパイラが認識し、最適な通信コードに翻訳することは、一般には難しい。**
  - **各imageの実行パスは、一般に翻訳時には不明。**
  - **MPIの集団通信のような文や組込み手続はない。**

```
common /com/me
integer,save :: a(100,100)[*]
      :
if(me .eq. 1)then
   do i=2, num_images()
     a[i] = a
   enddo
endif
```

F2008のcoarray機能の問題点のいくつか

image indexとimageの対応は固定(contextが1つ)

- MPI_COMM_WORLDしかなく、tagもないMPIと同様
- 例えば、複数のSYNC IMAGES文間の対応は、SYNC IMAGES文を含む手続を呼出すだけで、変わってしまう可能性があり、汎用的なライブラリ内では使いにくい。

```
real, save :: A(100)[*]
if(this_image().eq.1)then
     coarray Aを設定
   sync images(2)    ! 1-A
endif




if(this_image().eq.2)then
    sync images(1)     ! 2-A
     coarray Aを使用
endif
```

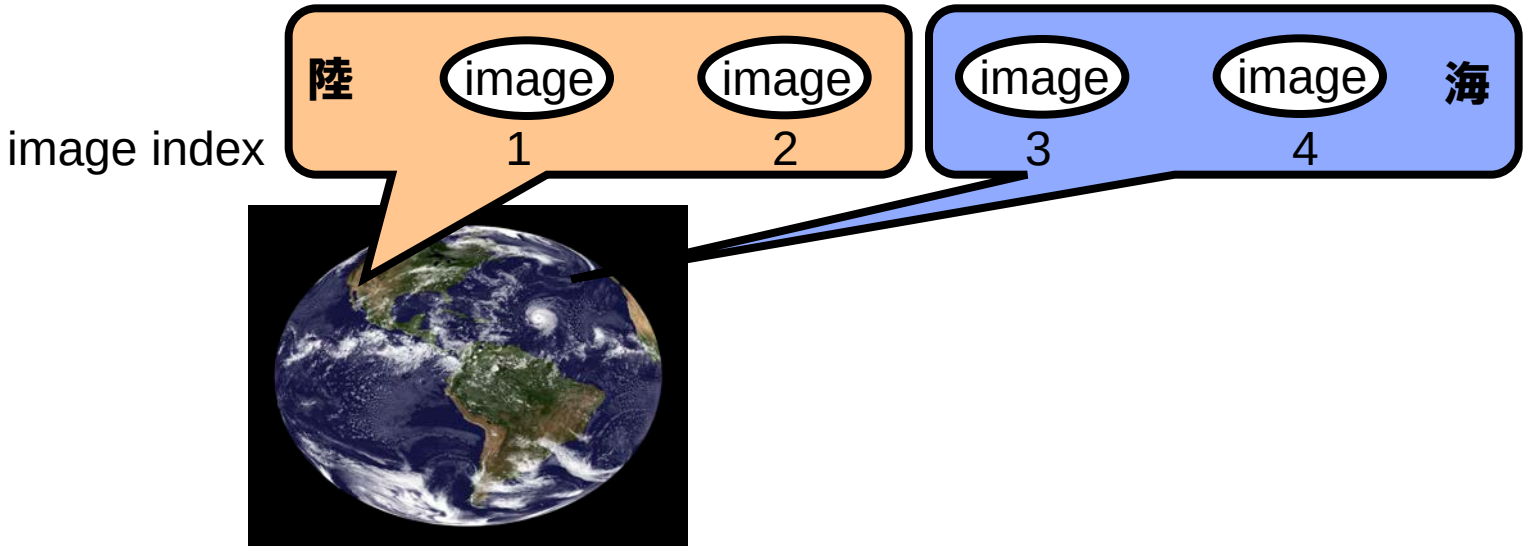
call sub()を挿入するとおかしくなる

```
subroutine sub()
real, save :: B(100,100)[*]
if(this_image().eq.1)then
     coarray Bを設定
   sync images(2)     !  1-B
else if(this_image().eq.2)then
   sync images(1)    !   2-B
     coarray Bを使用
endif
```

F2008のcoarray機能の問題点のいくつか

タスク並列プログラムが書きにくい

- 常に全image上での処理となり、タスク内処理の記述が面倒



(例)「海」側のタスクは、
- image index 3と4を使用しなければならない
- sync all と書くと、陸・海全体での同期となってしまう

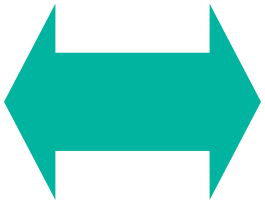
既存のプログラムをそのまま組み込めない

DO CONCURRENT構文の概要 1/2

INDEPENDENT指示文付きループとほぼ等価

- 但し、F2008での主ターゲットは共有並列
- また、呼び出せる手続は、PURE手続のみ

```
DO CONCURRENT (I=1:100)
   A(IDX(I)) = I
ENDDO
```



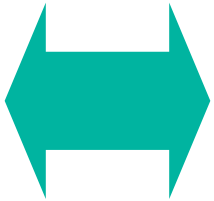
```
!HPF$ INDEPENDENT, NEW(I)
DO I=1,100
   A(IDX(I)) = I
ENDDO
```

DO CONCURRENT構文の概要 2/2

NEW変数・REDUCTION変数

- REDUCTION変数は書けない
  - 通常のDOループで書く(自動並列、OpenMP) or 組込み関数を使う
- NEW変数
  - BLOCK構文を使う (言語Cの { } と同様)

```
DO CONCURRENT (I=1:100)
   BLOCK
      REAL :: T
      T = A(I) + B(I)
      C(I) = T + SQRT(T)
   END BLOCK
ENDDO
```



```
!HPF$ INDEPENDENT, NEW(I,T)
DO I=1, 100

      T = A(I) + B(I)
      C(I) = T + SQRT(T)

ENDDO
```

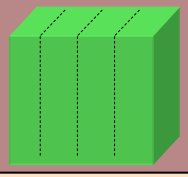
  - BLOCK構文を使わない場合、作業変数はループ終了後不定となる

coarray機能とHPF

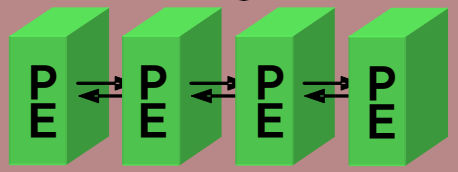
ベースとなるモデルは異なる

グローバルモデル

データをn個に分割指示しつつ，
プログラム全体の動作を記述する



処理系がn個に分割し，
同時に実行する

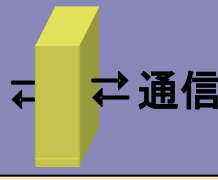


HPF

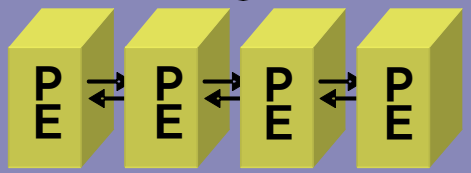
XMP
HPF_CRAFT

SPMDモデル（ローカルモデル）

各プロセッサの動作を記述する



n個同時に実行する



Coarray
MPI (MPMD)
HPF_LOCAL

HW

coarray機能とHPF

- HPF_LOCAL (ローカルモデルのHPF手続) には、自然にcoarray機能を組み込める
  - ⇒ 通信機能を持つHPF_LOCAL手続となる

- HPF (グローバルモデル) にcoarray機能を組み込むとハイブリッドなモデルとなる
  - coarrayはローカル変数、それ以外はグローバル変数
  - image毎に制御フローが異なる可能性がある点等に関して、いくつか制限が必要
  - ⇒ HPF_CRAFT や XMP に似たモデルとなる

coarrayの主要な機能まとめ (ISO Fortran + α)

データマッピング

coarrayは全image上に必ず存在する

例) 2次元配列a(10,10)は、各image上にそれぞれ存在する

```
real,save :: a(10,10)[*]
```

通信

基本的には代入文の形で書く(片側通信・手動)

例) image 2上のbを、image 1上のbに代入する

```
b[1] = b[2]
```

並列処理・同期

各imageは並列に動作し、image間処理は手動

同期： sync all, sync images((/1,2,.../)), sync team(TEAM)

片方向同期： notify((/1,2,.../)), query((/1,2,.../))

実行完了の保証： sync memory

1 image毎の実行： CRITICAL構文

ロック・アンロック： LOCK文, UNLOCK文

部分image集合生成： form_team(TEAM,(/1,2,.../))

集計計算： co_sum(...[,TEAM]), co_maxval(....等

: image index

: 現在議論中(WG5)

その他

atomicな値の引用・確定： atomic_ref(), atomic_define()

各種問合せ手続： this_image(), num_images() 等