

Fortran 2008のcoarray機能

2013年11月

NEC

林 康晴

内容

Fortran 2008のcoarray機能

- プログラミングモデル
- データの整合性を保証する方法

XMPにおけるcoarray機能

- XMP/Fortran・XMP/Cとcoarray
- coarrayの拡張仕様

Fortran 2008のcoarray機能まとめ

Fortran 2008のcoarray機能

Fortran 2008

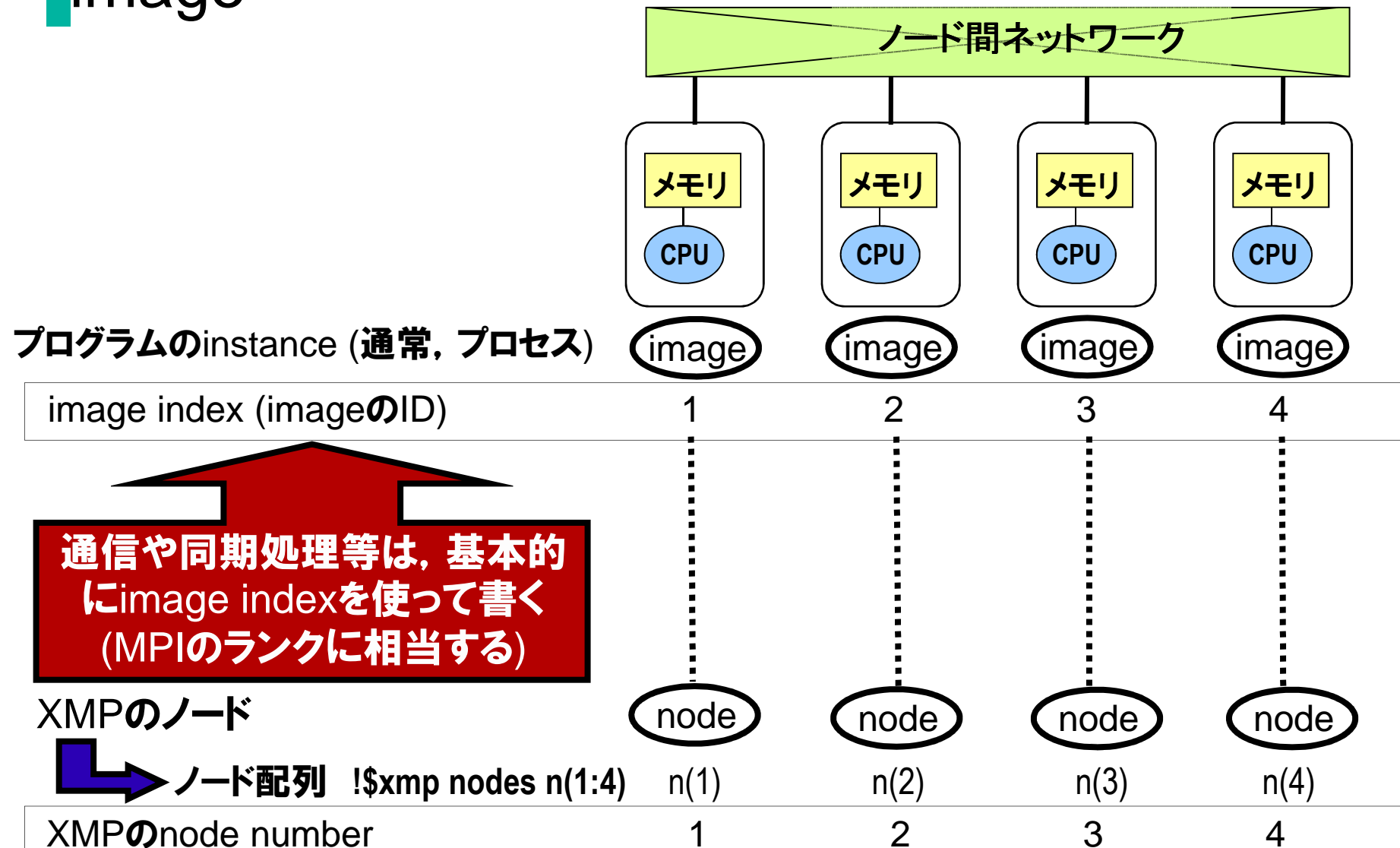
- 2010年, 最新のISO規格に

目玉の1つとして, coarray機能が導入された

- (主として)分散並列処理のための機能
- (MPIのような)ローカルモデル・全手動の並列化
 - 通信は, 代入文の形式で, 片側通信方式により記述する

Fortran 2008のcoarray機能概要(1)

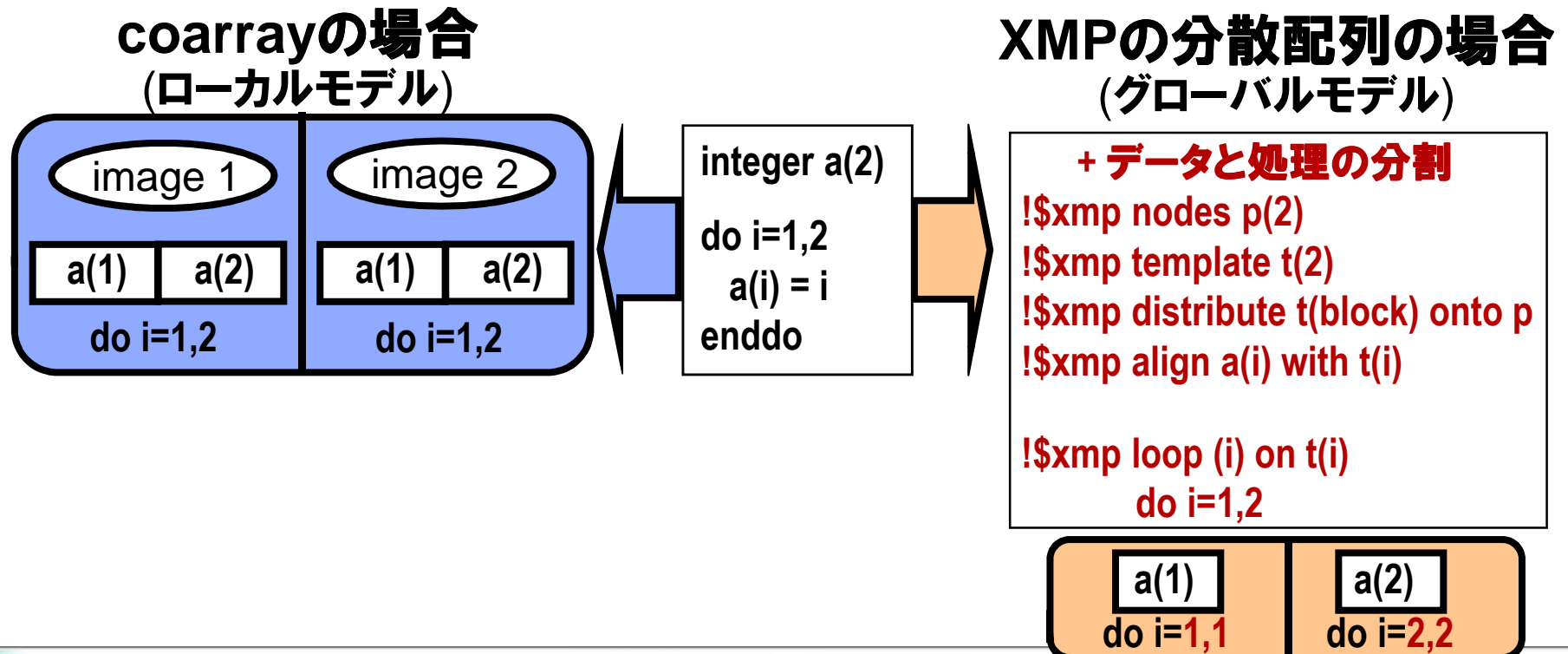
image



Fortran 2008のcoarray機能概要(2)

データマッピングと処理の分割

- データ・処理は、各image上にそれぞれ割り当てられる
 - プログラム上のデータ・処理は、各image上にそれぞれ割り当てられる（ローカルモデル）
 - データマッピング・処理の分割は、プログラムと一体（プログラミングの時点で完了していなければならない）



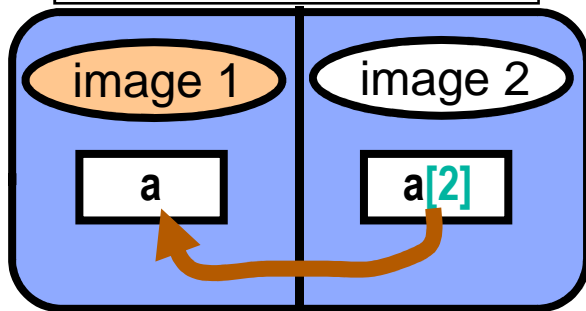
Fortran 2008のcoarray機能概要(3)

通信(片側)

- imageの構成を, 宣言時に`codimension([*])`で指定する
 - `[n1,n2,...,*]`のように, 多次元の指定も可能
- coarrayの参照時に, `image selector([i])`を指定して, 他のimage *i* 上のデータにアクセスできる(通信)

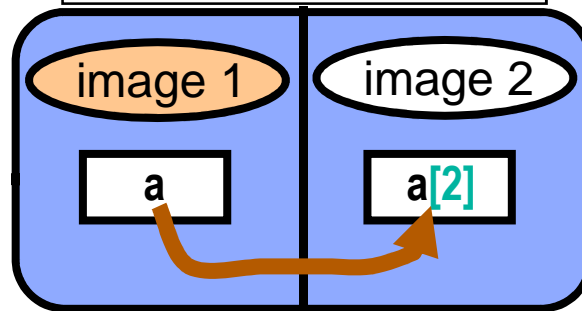
get

```
integer,save :: a[*]  
if(this_image().eq.1)then  
  a = a[2]  
endif
```



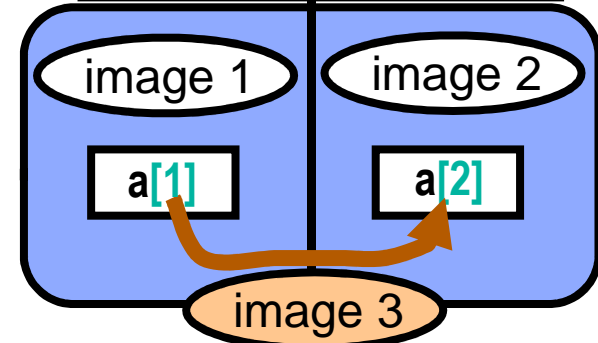
put

```
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
```

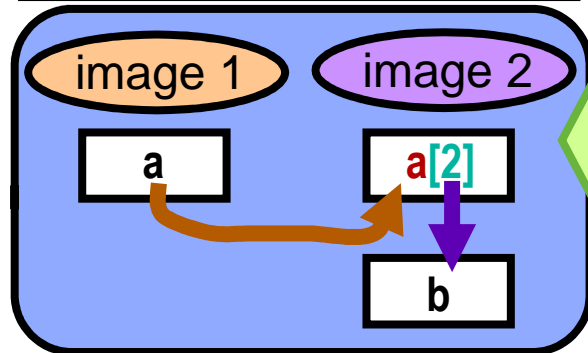


Fortran 2008のcoarray機能概要(4)

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

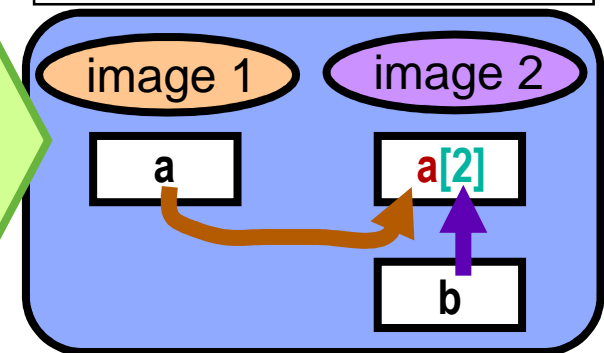
```
integer,save :: b, a[*]  
  
if(this_image().eq.1)then  
  a[2] = a  
else if(this_image().eq.2)then  
  b = a  
endif
```



ユーザーが、
正しい順序での実行を保証するような
コードを書く

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)
 - SYNC ALL文, SYNC IMAGES文
 - CRITICAL構文
 - LOCK文, UNLOCK文
- Fortran以外の同期機能 + SYNC MEMORY文
 - SYNC MEMORY文で, メモリ操作の完了を保証できる。
- atomic subroutineによる同期 + SYNC MEMORY文
 - atomic subroutine (atomic_ref・atomic_define)の引用によって同期をとることができる。

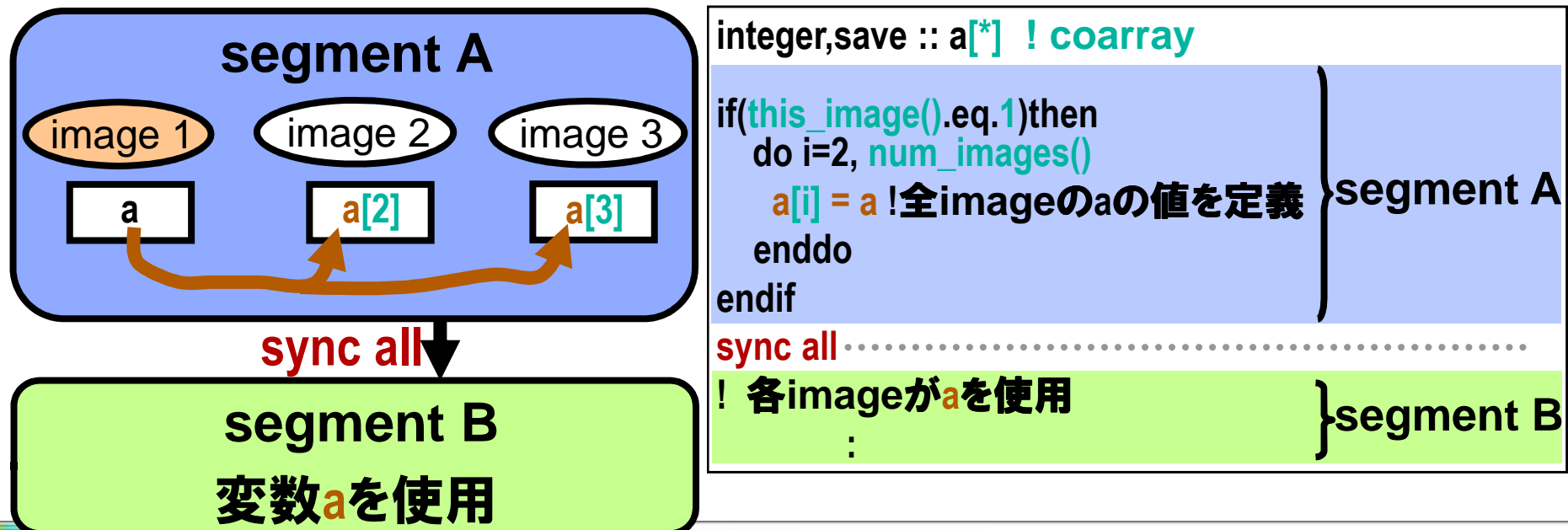
データの整合性を保証するための手段 (1)

Fortranの同期機能 1/4

● SYNC ALL文

- 全imageで同期を取る(プログラムはsegmentに分割される)
- 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。



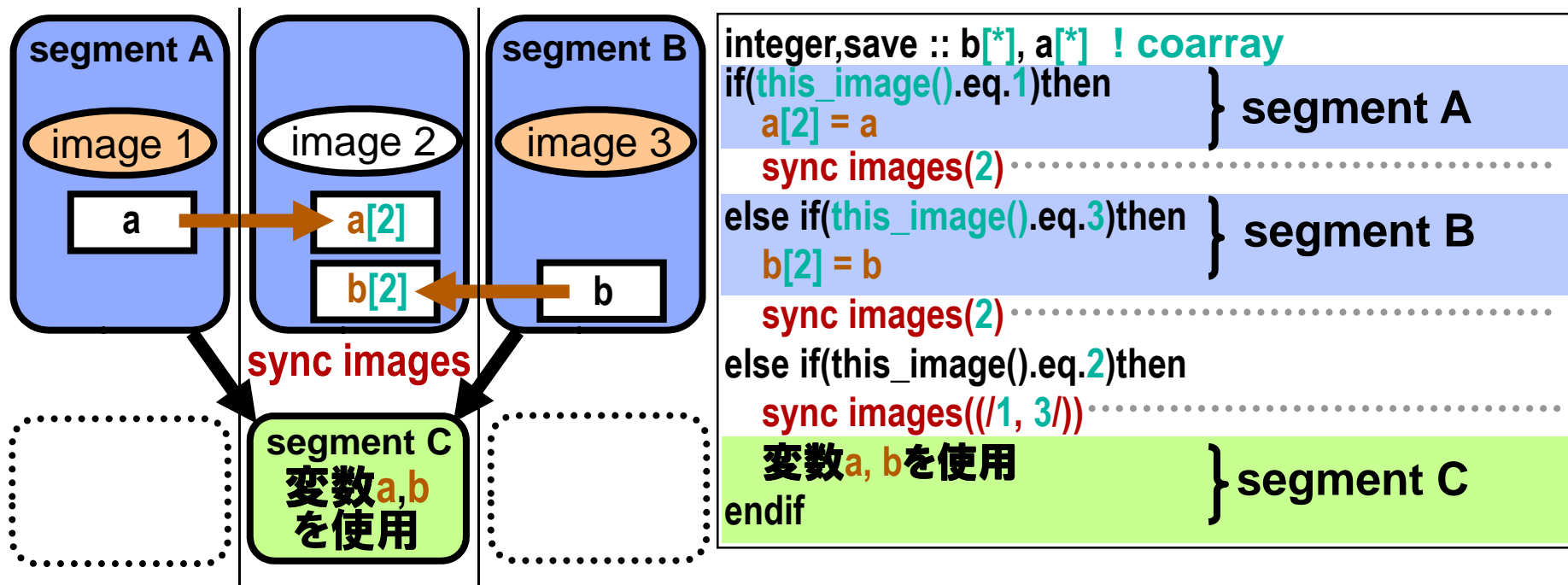
データの整合性を保証するための手段 (2)

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。



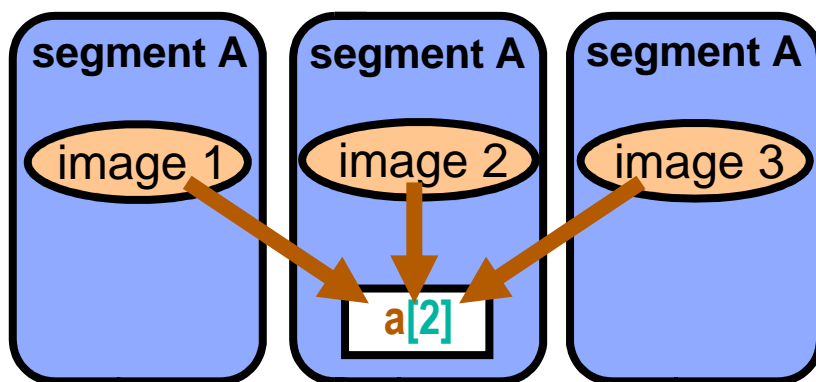
データの整合性を保証するための手段 (3)

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

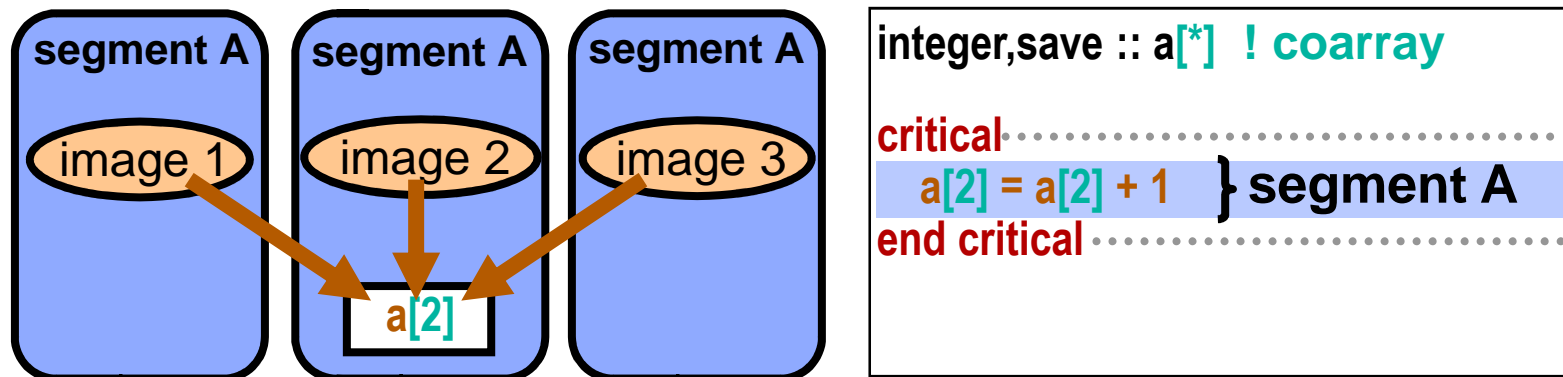
データの整合性を保証するための手段 (4)

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。



データの整合性を保証するための手段 (5)

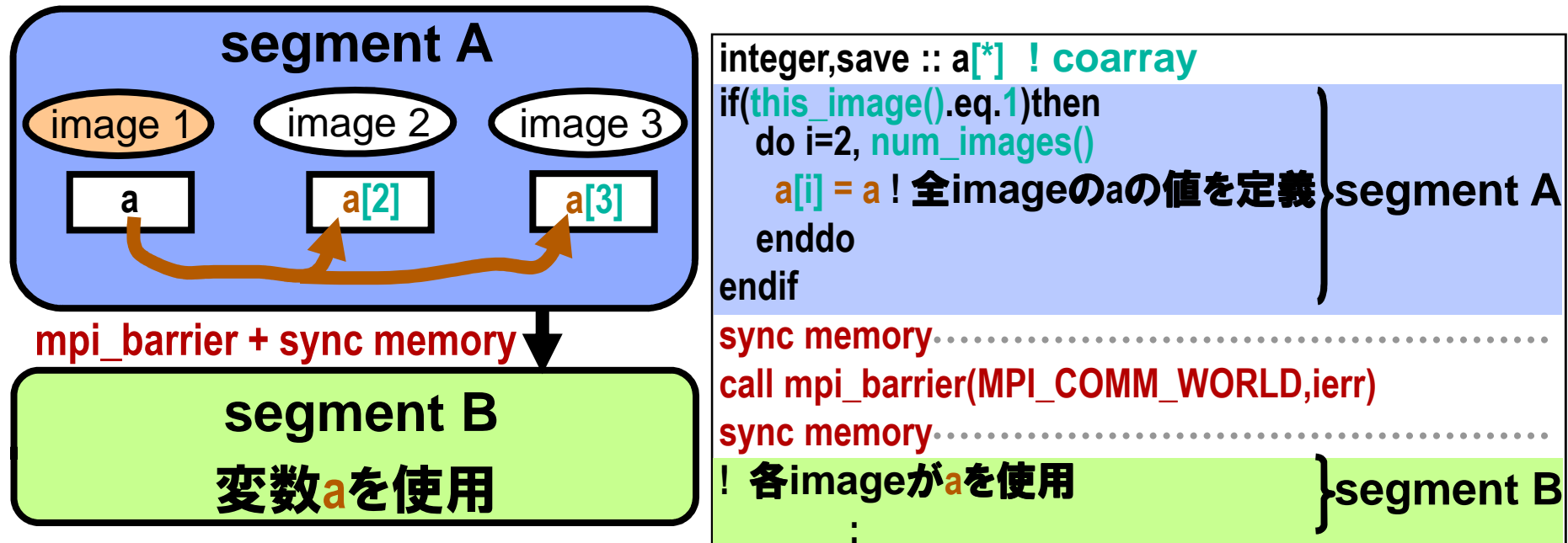
Fortran以外の同期機能 + SYNC MEMORY文

● 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。



データの整合性を保証するための手段 (6-1)

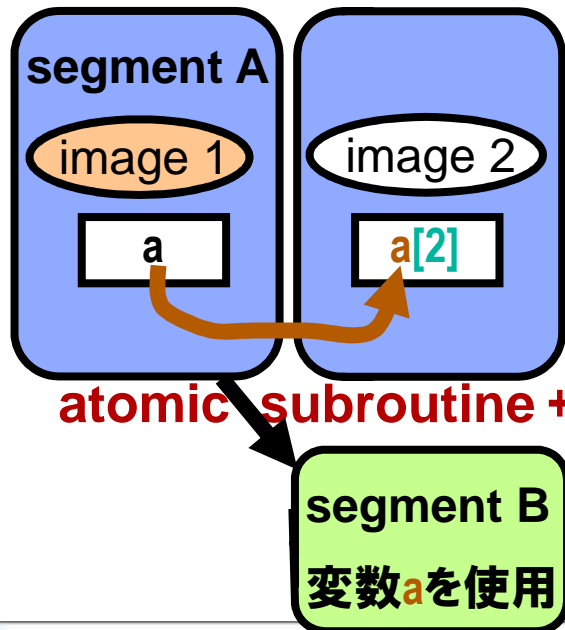
atomic subroutine (2種類)

- **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, 組み込みモジュール ISO_FORTRAN_ENVで定義されている
- **同じATOM引数に対する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 (6-2)

atomic subroutineによる同期 + SYNC MEMORY文

例: segment Bは, segment Aの実行後に実行される。



wait[2]を偽に
設定する

waitが偽にな
るのを待つ

```

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
    
```

XMPにおけるcoarray機能

XMP/Fortranは、Fortranの上位互換

- 全てのcoarray機能を利用可能

XMP/Cは、拡張によりcoarray機能をサポート

- *codimension*の宣言・*image selector*の指定が可能

```
float a[10]:[*], b[10]:[*] /* coarrayの宣言 */  
b[:]:[1:10] = a[:]      /* coarrayの参照 */
```

- XMP/Cでも、*codimension*・*image selector*部分の添字の下限値は1

- XMPのライブラリ関数により同期機能をサポート

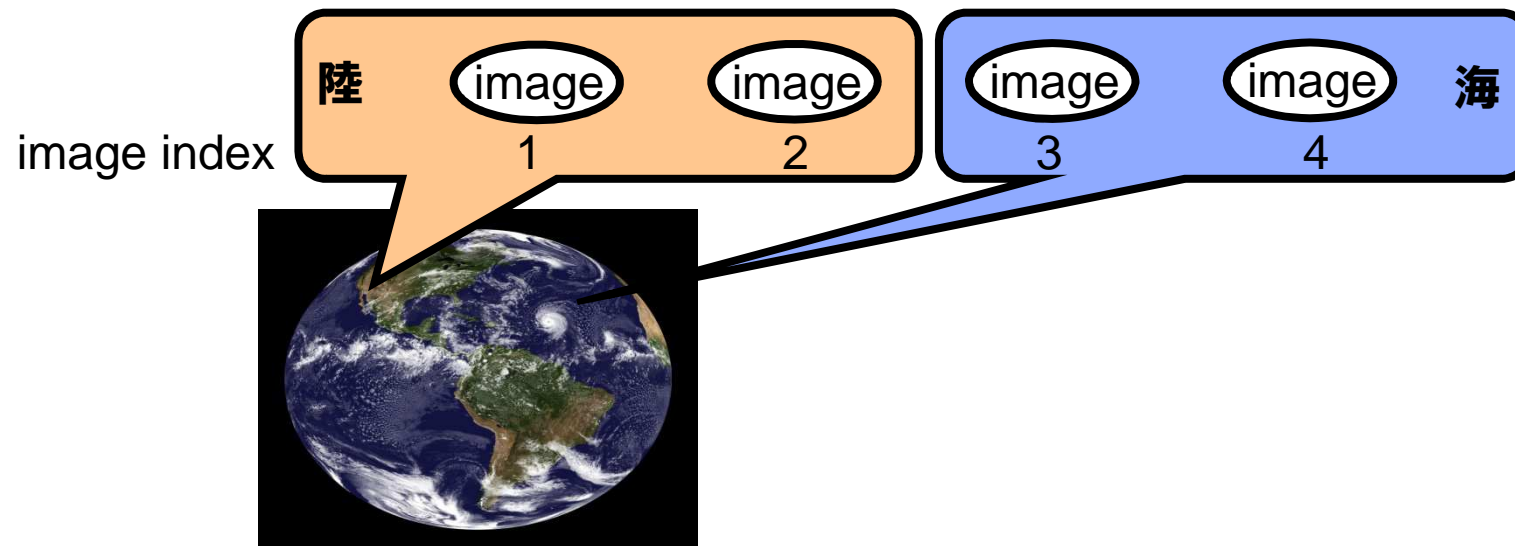
```
void xmp_sync_all(int *stat)           /* sync all文と同様 */  
void xmp_sync_memory(int *stat)       /* sync memory文と同様 */  
void xmp_sync_image(int i, int *stat) /* sync images(i)文と同様 */  
void xmp_sync_images(int num, int*, int *stat) /* sync images((/i1,i2,.../))文と同様*/  
void xmp_sync_images_all(int *stat)   /* sync images(*)文と同様 */
```

さらに、タスク並列のための拡張機能を提供

Fortran 2008のcoarray機能の問題点の1つ

タスク並列プログラムが書きにくい

- imageとimage indexの対応は固定(contextは1つ)
- 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、

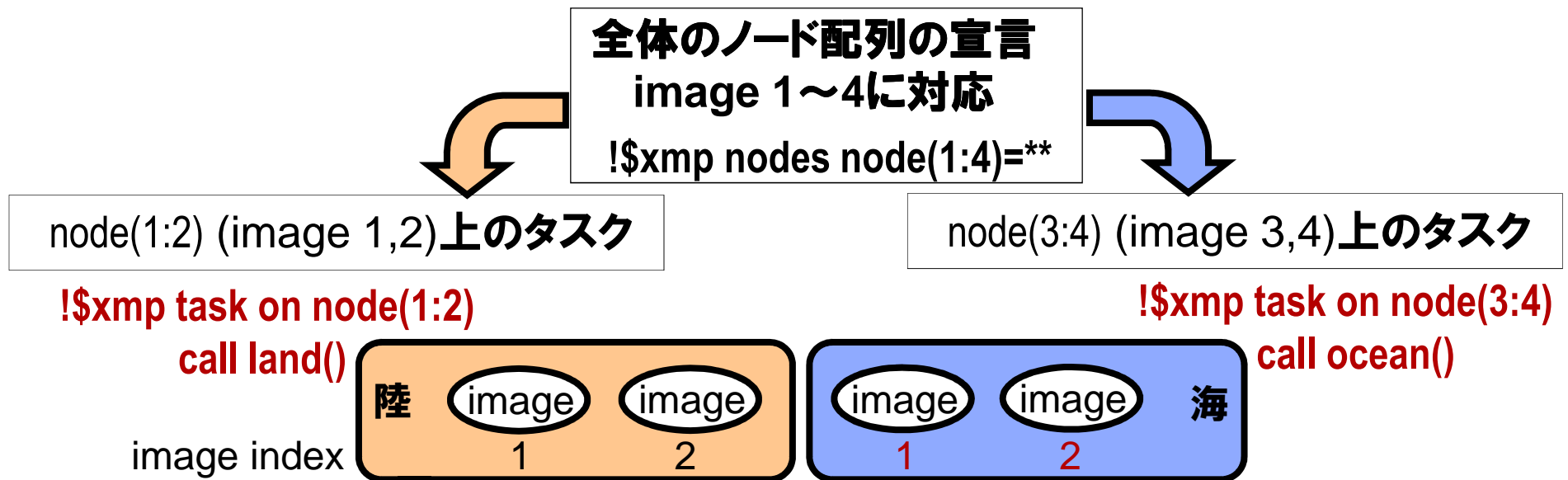
- image index 3と4を使用しなければならない
- sync all と書くと、陸・海全体での同期になってしまう

➡ 既存のプログラムをそのまま組み込めない

[XMP] タスク並列のための拡張(1) 1/3

■ タスク内処理

- XMPのTASK指示文と「**全image**」を連動させる
 - image indexにTASK指示文に対応したcontextを持たせる



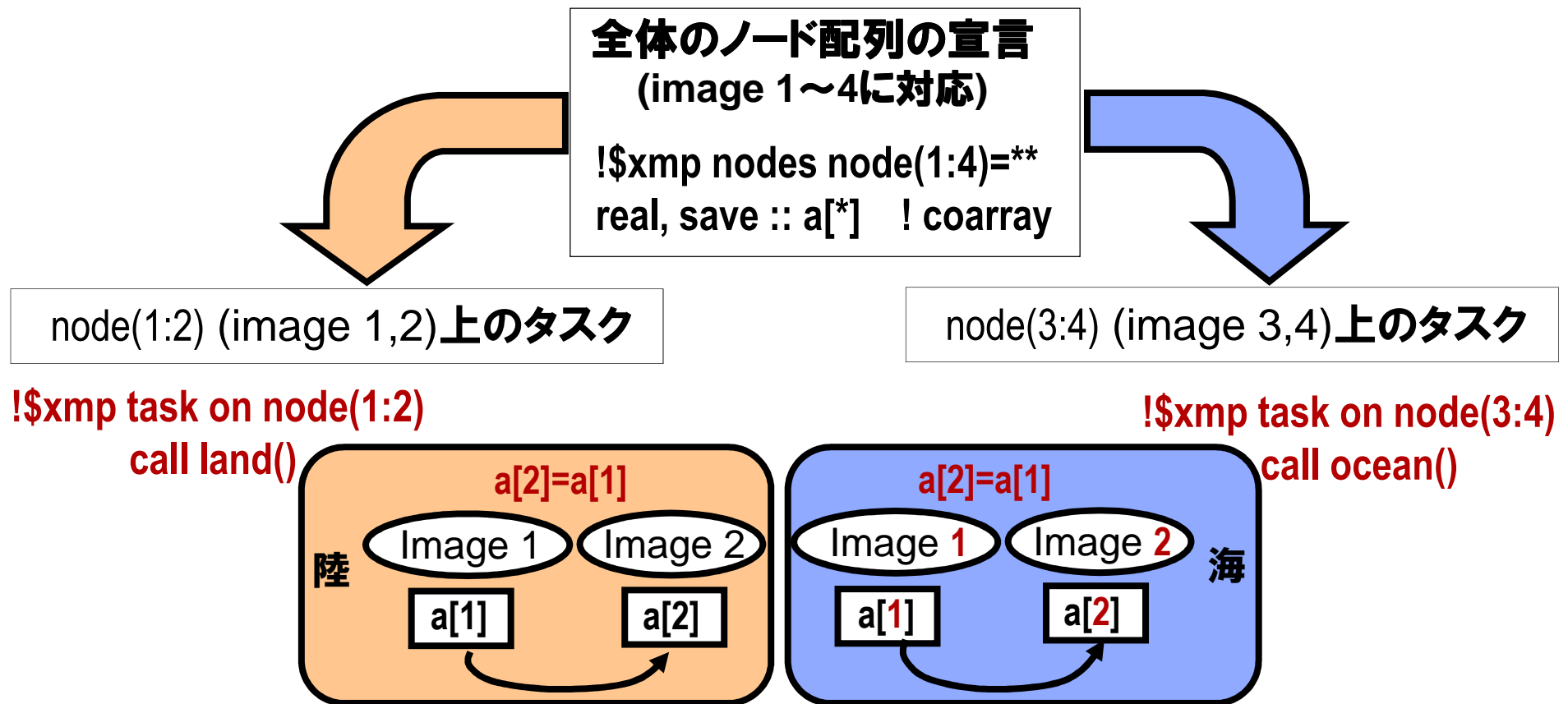
- TASK指示文で指定されたimageの集合が「**全image**」
➔ タスク内に閉じたimage indexでプログラムを記述できる

[XMP] タスク並列のための拡張(1) 2/3

■ タスク内処理: 通信

- (既定値では)タスク内のimage上のcoarrayだけが参照可能

[例]陸タスク・海タスク共に、タスク内のimage 1のaをimage 2のaに代入する

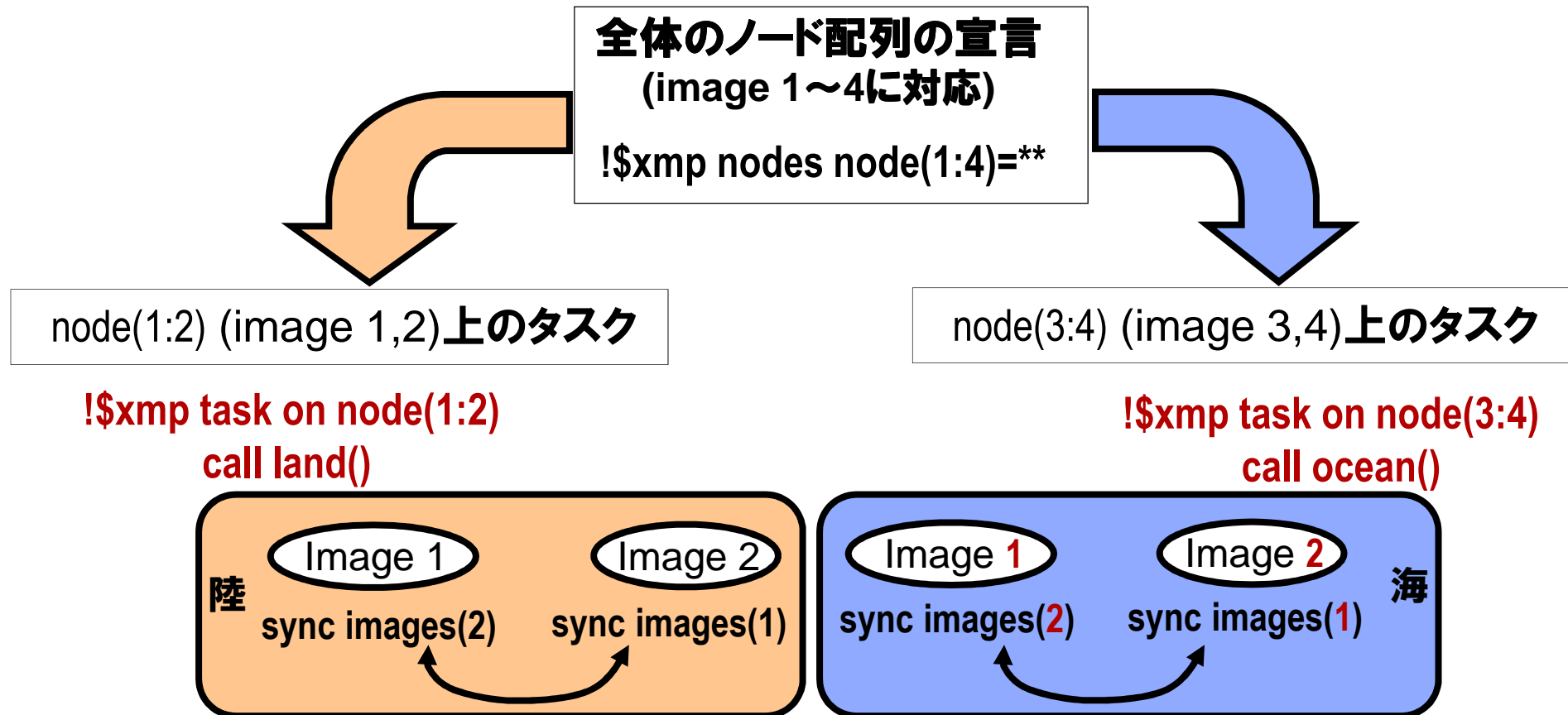


[XMP] タスク並列のための拡張(1) 3/3

■ タスク内処理:同期

- (既定値では)同期等のimage間処理もタスク内に閉じる

[例]陸タスク・海タスク共に、タスク内のimage間で同期を取る



■ タスク間処理 (contextの指定)

- image indexが指すimageを変更する指示文を提供

宣言時にcoarrayに指定する
COARRAY指示文

coarrayのimage selector内: $b[1] = b[2]$

image間処理実行文に指定する
IMAGE指示文

同期の対象: $\text{sync all, sync images}((/1,2,.../))$

■ : image index

- 異なるimage集合間のimage index変換手続を提供
 - (プログラム開始時の)全image集合におけるimage indexへの変換
 - 実行中のタスクを構成するimage集合におけるimage indexへの変換

タスク間処理 (contextの指定)

● COARRAY指示文 (宣言の指示文)

- 各coarrayが参照可能なimage集合の範囲を指定できる
- TASK指示文に関係なく、常に指定されたノード配列に対応するimage集合の範囲が参照可能になる(見える)

(例)

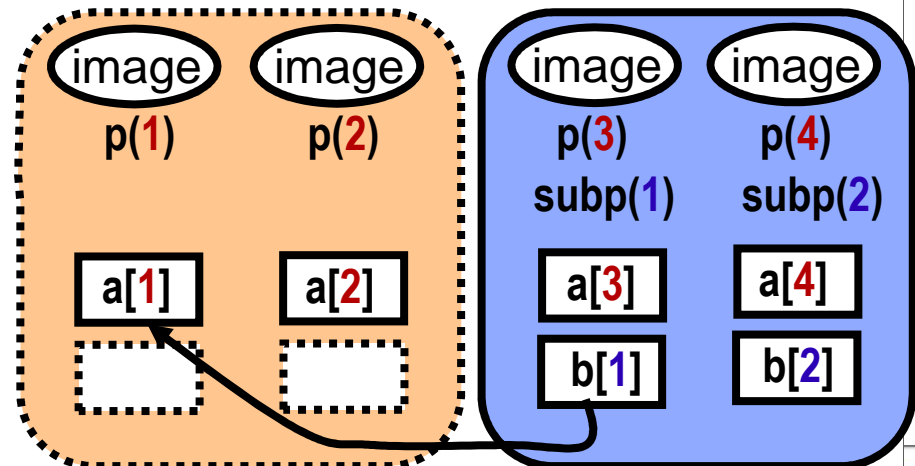
```
!$xmp nodes p(4) =**
      real,save :: a[*], b[*]
!$xmp coarray on p :: a
      :
!$xmp task on p(3:4)
      block  ||
!$xmp nodes subp(2)=*

      me = this_image()
      if(me.eq.1) a[1] = b[1]
      end block
!$xmp end task
```

! 全体のノード配列pの宣言

! coarray aとb

! coarray aは常にp(1:4)全体上の実体が見える



タスク間処理 (contextの指定)

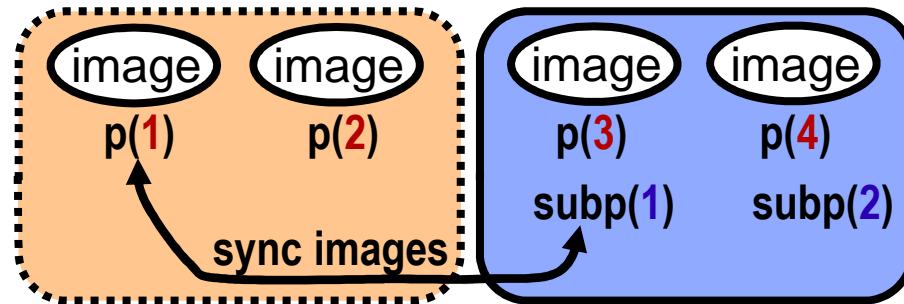
● IMAGE指示文 (実行文への指示文)

- 同期等の実行文のimage indexのcontextを指定できる
- TASK指示文に関係なく、常に指定されたノード配列に対応するimage集合上でのimage indexを意味する。

(例)

```
!$xmp nodes p(4) =**  
:  
!$xmp task on p(3:4)  
  block  ||  
    !$xmp nodes subp(2)=*  
      me = this_image()  
      if(me.eq.1)then  
        !$xmp image (p)  
          sync images(1)  
        endif  
      end block  
    !$xmp end task
```

! 全体のノード配列pの宣言



! 1は、ノード配列p上の1 (p(1))を意味する。

Fortran 2008のcoarray機能まとめ

データマッピング

coarrayは全image上に必ず割り付けられる

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

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]

■ : image index

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,.../))

メモリ操作完了の保証: sync memory

1 image毎の実行: CRITICAL構文

ロック・アンロック: LOCK文, UNLOCK文

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

XMPでは

- Cでもcarray機能を利用可能
- グローバルモデルと併用可能
- タスク並列処理が記述しやすい