Empowered by Innovation NEC

次期Fortran規格の並列処理関連機能

2013年8月

NEC
林　康晴

内容

- 次期Fortran規格の方針概要
- Fortran 2008のcoarray機能復習
- 議論中の並列処理機能の拡張概要
  - 最新の仕様案(N1983: 3rd July 2013)をベースに
- まとめ

次期Fortran規格の方針

大きな機能拡張は行わない

- 機能拡張
  - Cとの相互利用可能性の拡張　・・・ TS発行済
  - 並列処理機能の拡張(coarray)　・・・　議論中
- Fortran 2008の問題点・矛盾点を修正する

2017年2月リリース予定 (Fortran 2016?)

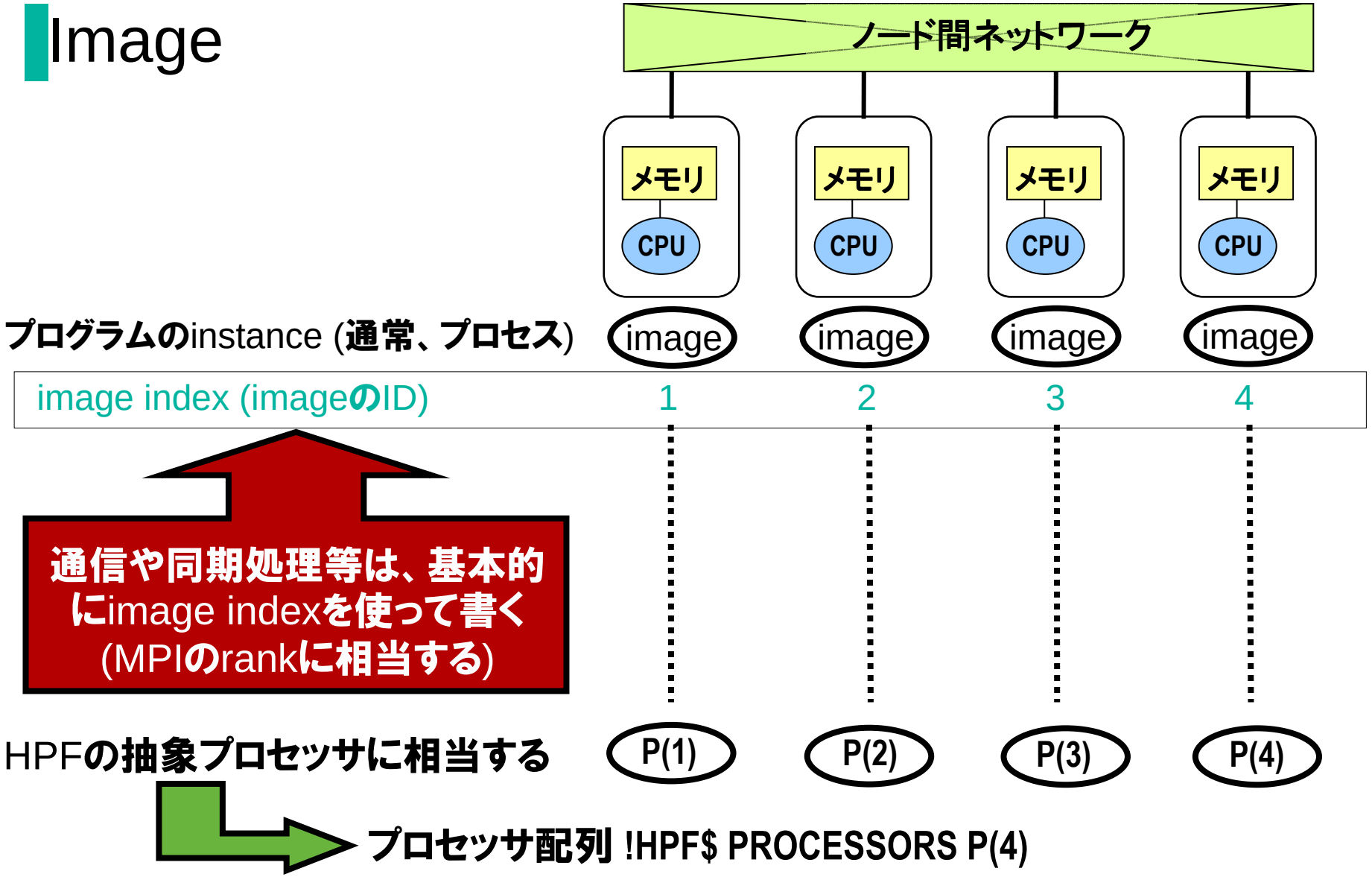
並列処理機能の拡張の仕様策定スケジュール

- 2014/06　Draft Technical Specification constructed
- 2014/11　TS published

これ以上遅延した場合，規格本体の遅延を避けるため，次期Fortran規格には入らない可能性もある

Fortran 2008のcoarray機能概要(1)

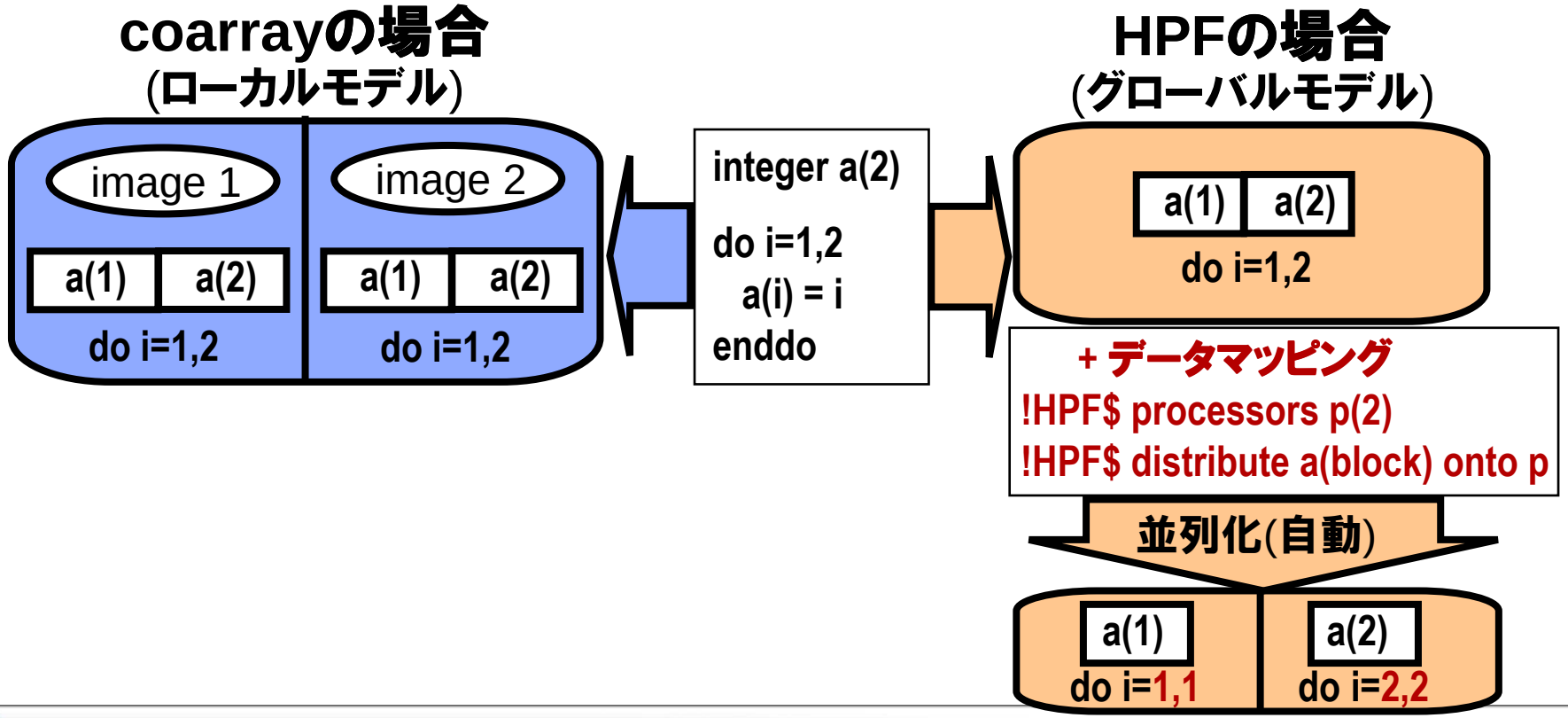
Image



Fortran 2008のcoarray機能概要(2)

データマッピングと並列化

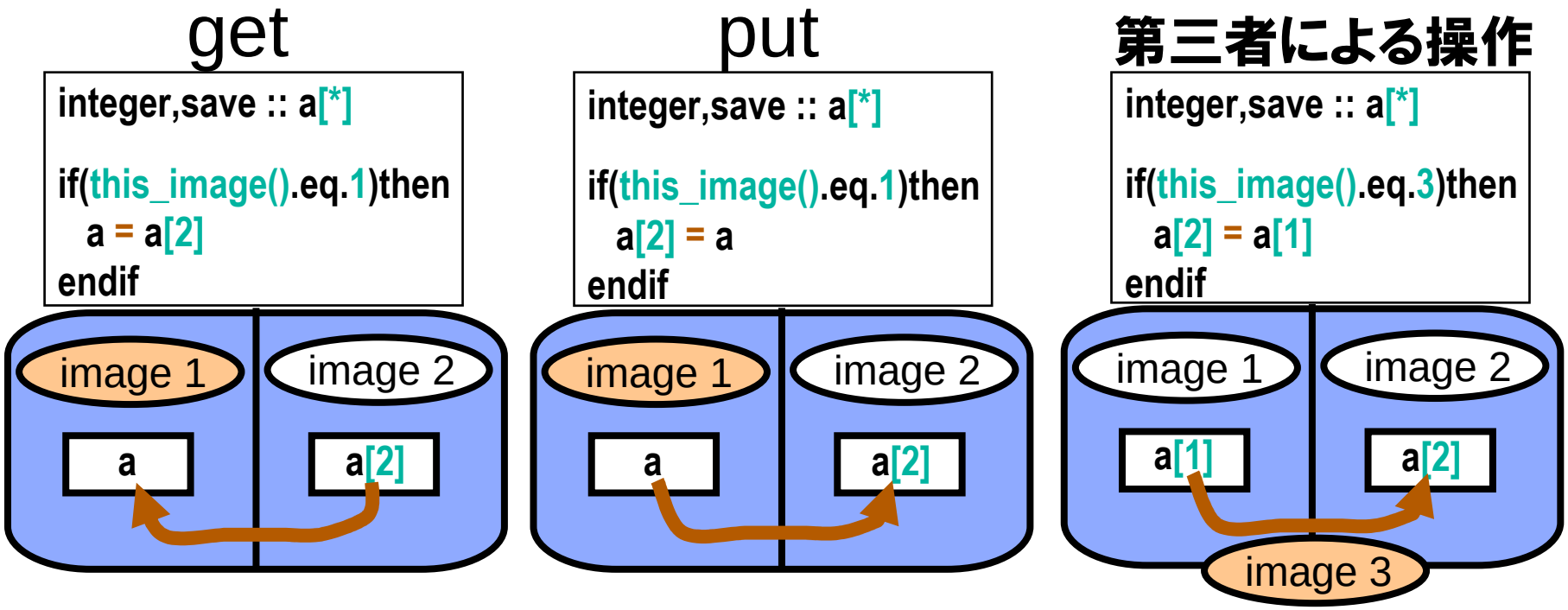
- データ・処理は、各image上にそれぞれ割り当てられる
  - 各image上のデータ・処理は、独立した別物(ローカルモデル)
  - データマッピング・処理の並列化は，プログラムと一体(プログラミングの時点で完了していなければならない)



Fortran 2008のcoarray機能概要(3)

通信(片側)

- coarrayの最後に，*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)



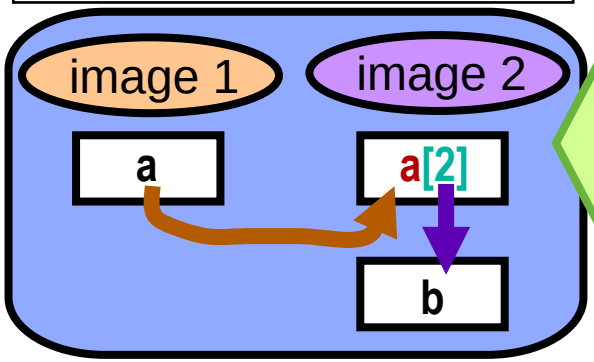
Fortran 2008のcoarray機能概要(4)

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
```

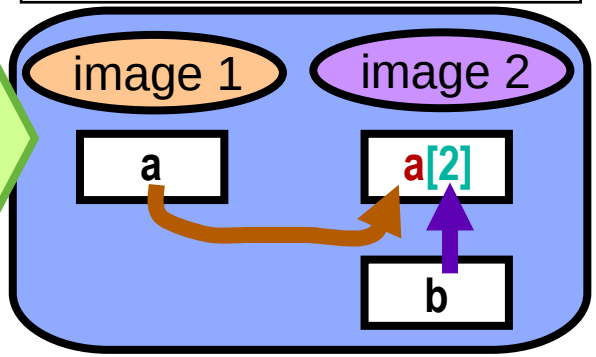


ユーザーが、正しい順序での実行を保証するようなコードを書く

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
```



Fortran 2008のcoarray機能概要(5)

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。

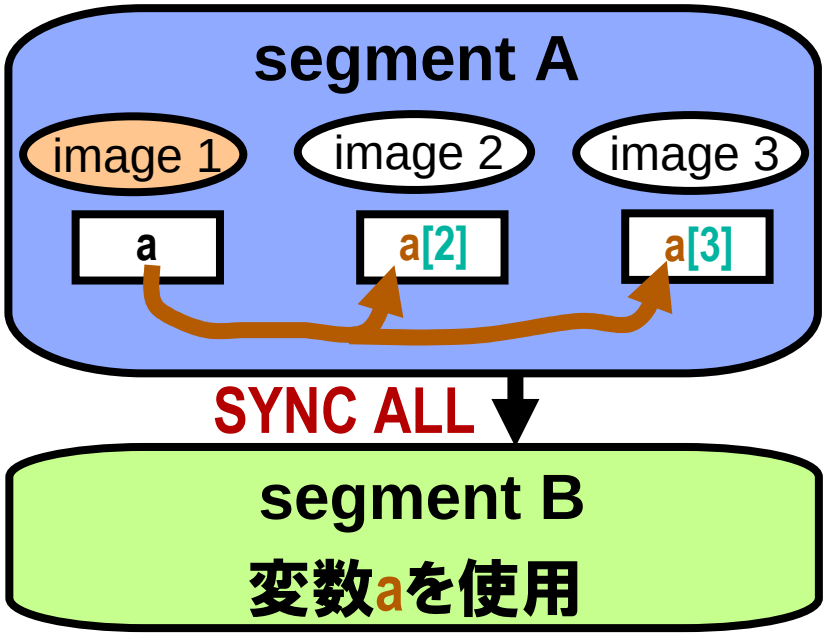
- atomicサブルーチン + SYNC MEMORY文
  - ATOMIC_REF・ATOMIC_DEFINEを引用して，atomicな引用・定義が可能

Fortran 2008のcoarray機能概要(6)

データを整合性を保証する方法の例

- SYNC ALL文
  - 全imageで同期を取る。全てのメモリ操作を完了させる。
  - 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      } segment A
   enddo
endif
SYNC ALL ..................................
! 各imageがaを使用                        } segment B
          :
```

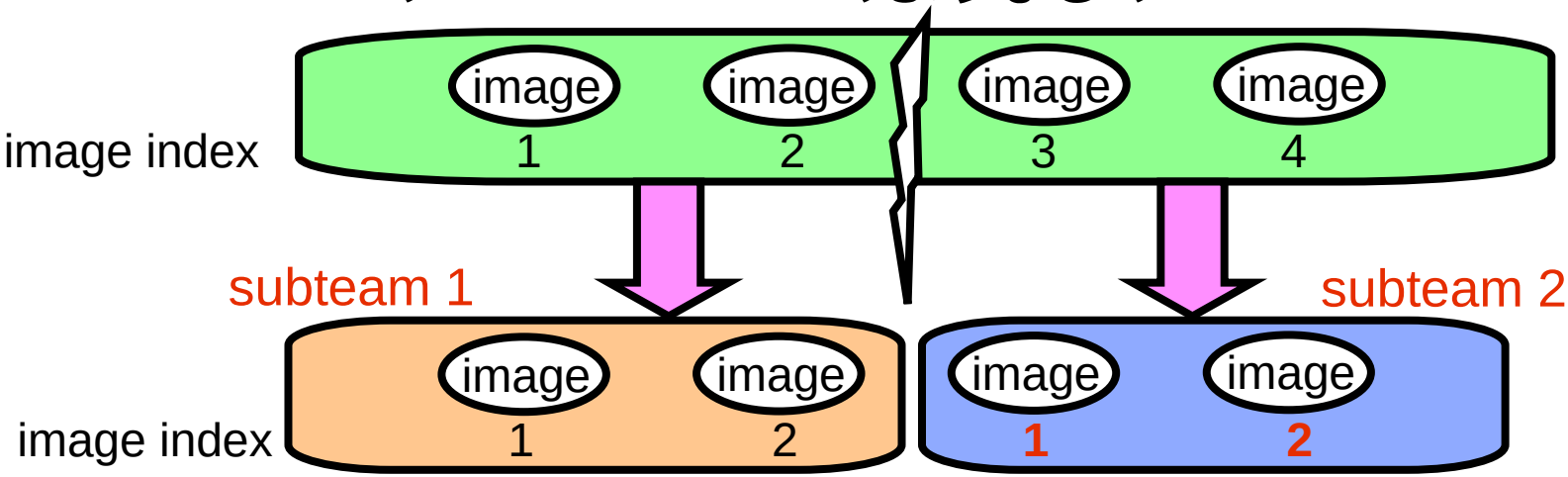
並列処理機能の拡張(現在議論中)

- タスク並列
- 片方向同期
- 集団通信組込み手続
- atomic操作組込み手続
- フォールトトレラント機能

タスク並列(1)

TEAM

- imageの集合を分割し，各部分集合(subteam)内で完結した処理を記述できる。
  - MPIのcommunicatorのようなもの



- image indexは，各subteam内で 1～image数 の値をとる
- 各subteamは，ユーザが指定した整数値で識別可能(*subteam-id*)
- subteamに閉じた処理に移行するには，team変数 (TEAM_TYPE型)を使用する（team変数の内容は処理系が管理する)。
  - TEAM_TYPE型は，組込みモジュールISO_FORTRAN_ENVで定義される

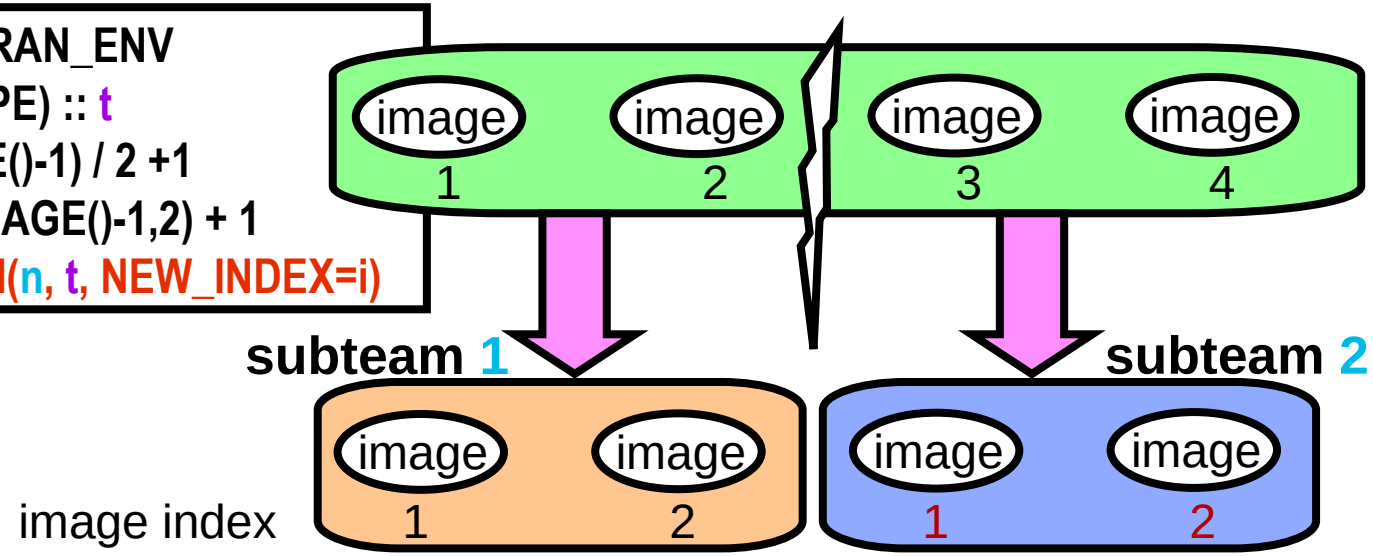
タスク並列(2)

subteamへの分割: FORM SUBTEAM文

FORM SUBTEAM (スカラ整数式, team変数 [, *form-subteam-spec-list*])

例

```
USE  ISO_FORTRAN_ENV
TYPE(TEAM_TYPE) :: t
n = (THIS_IMAGE()-1) / 2 +1
i = MOD(THIS_IMAGE()-1,2) + 1
FORM SUBTEAM(n, t, NEW_INDEX=i)
```



- **スカラ整数式(n) で，自imageが属する*subteam-id*を指定する (MPI_COMM_SPLITのCOLOR引数と同様）**
- **team変数(t) に自imageが属するsubteamの情報が設定される**
- **NEW_INDEX=i の指定によって，subteam内での自imageのimage indexを指定できる。(指定しない場合の規定はない)**

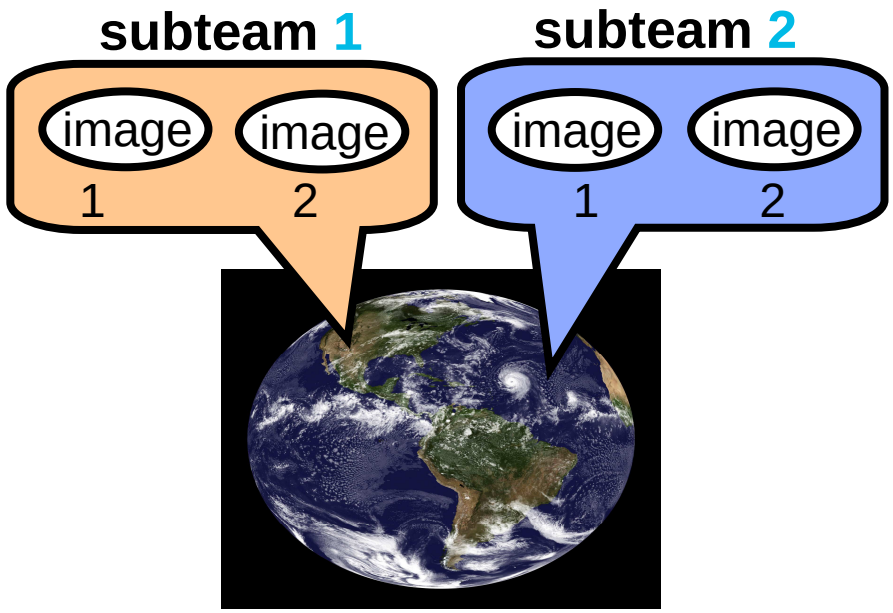
タスク並列(3)

subteam内での処理: CHANGE TEAM構文

```
CHANGE TEAM ( team変数 [, sync-stat-list ] )
   block
END TEAM
```

例

```
USE  ISO_FORTRAN_ENV
TYPE(TEAM_TYPE) :: t
     :
FORM SUBTEAM(n, t, NEW_INDEX=i)
CHANGE TEAM(t)    ! subteam t内で同期
   if(SUBTEAM_ID() .eq. 1)then
      call compute_land()
   else
      call compute_ocean()
   endif
END TEAM          ! subteam t内で同期
```



- 実行中のsubteamに属するimageの集合が，「全image」となる
- 組込み関数SUBTEAM_ID()で，現在属しているsubteam-idを取得可能

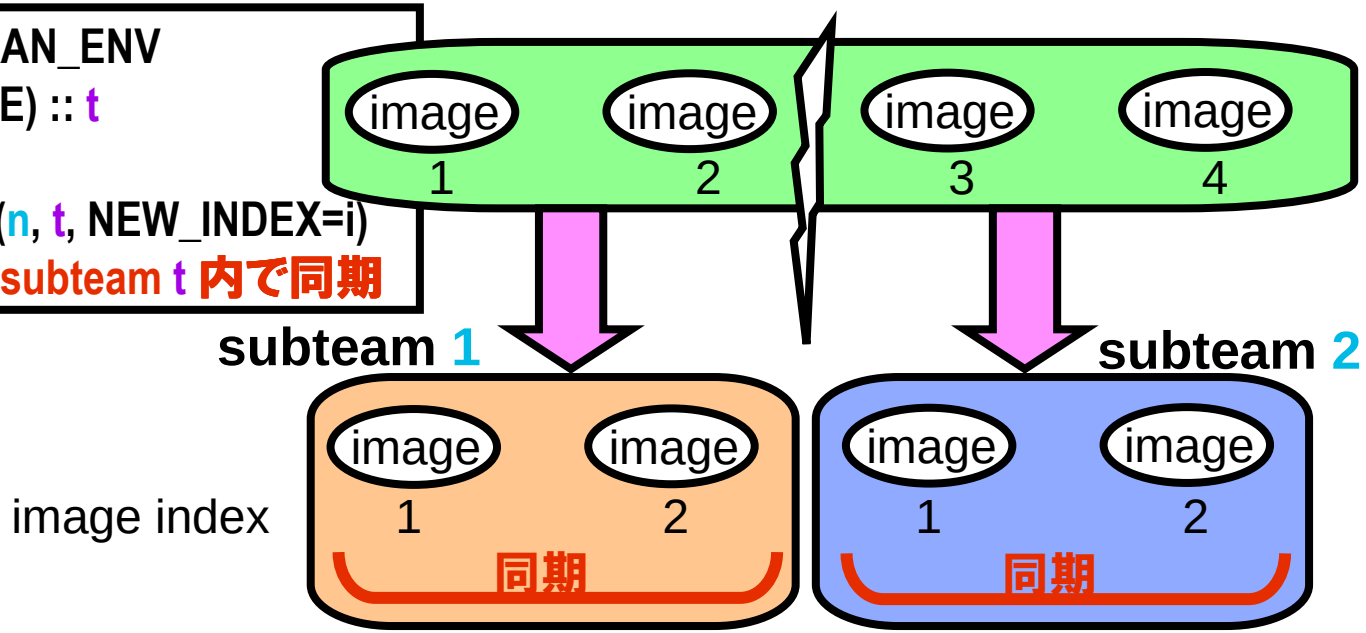
タスク並列(4)

subteam内での同期: SYNC TEAM文

SYNC TEAM (team変数 [, *sync-stat-list*])

例

```
USE  ISO_FORTRAN_ENV
TYPE(TEAM_TYPE) :: t
      :
FORM SUBTEAM(n, t, NEW_INDEX=i)
SYNC TEAM(t) ！ subteam t 内で同期
```



- **各subteam 1, 2内で，それぞれ同期をとる**
  - SYNC ALL**文の同期範囲限定版**

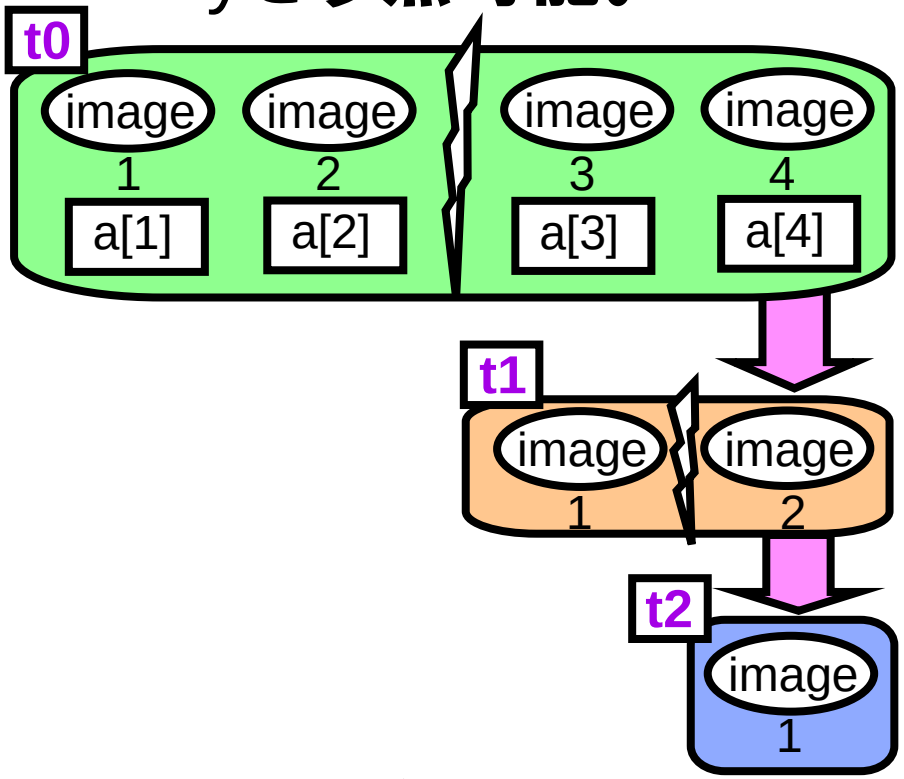
タスク並列(5)

他の(祖先の)team内のcoarrayの参照

- 祖先teamのteam変数を*image-selector*中に指定して，祖先teamのimage上のcoarrayを参照可能。

例

```
USE  ISO_FORTRAN_ENV
real a[*]
TYPE(TEAM_TYPE) :: t0, t1, t2
i = THIS_IMAGE()
FORM TEAM(0,t0,NEW_INDEX=i)
          :
CHANGE TEAM(t2)
   b = a[1]       ! team t0の a[4]の引用
   c = a[t1:: 1]  ! team t0の a[3]の引用
   d = a[t0:: 1]  ! team t0の a[1]の引用
END TEAM
```



- team変数を問い合わせる手続はないので，大域変数や引数経由で参照可能にする必要がある

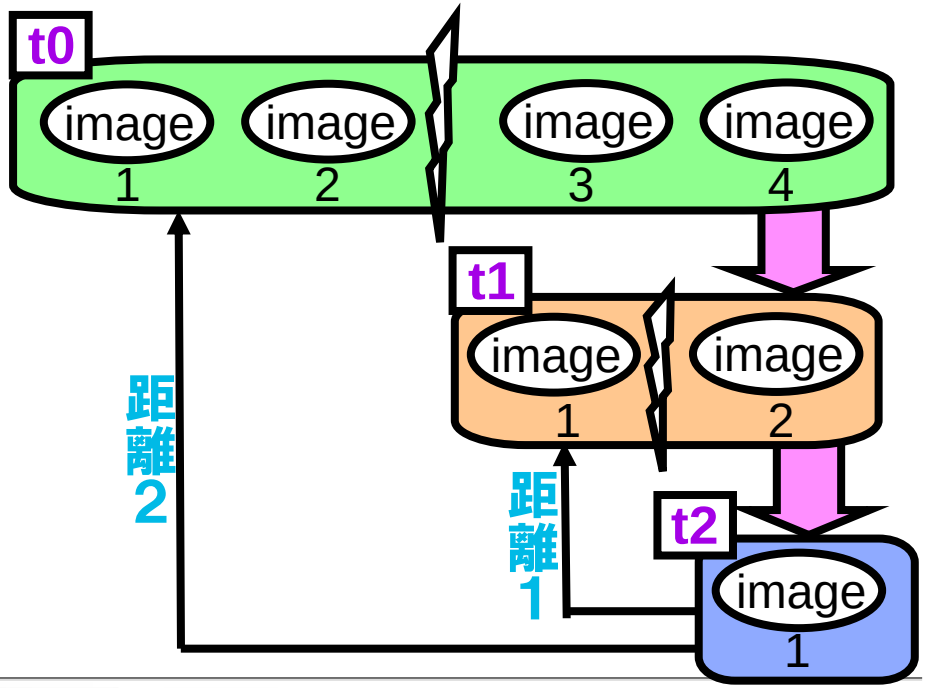
タスク並列(6)

祖先teamのimage index・image数・*subteam-id*の取得

- 組込み関数TEAM_DEPTH()により、現在のteamとプログラム開始時のteamとの距離を取得可能
- 組込み関数THIS_IMAGE(), NUM_IMAGES(), SUBTEAM_ID()の引数に現在のteamとの距離を指定すると、対応するteamにおける情報を取得可能

例

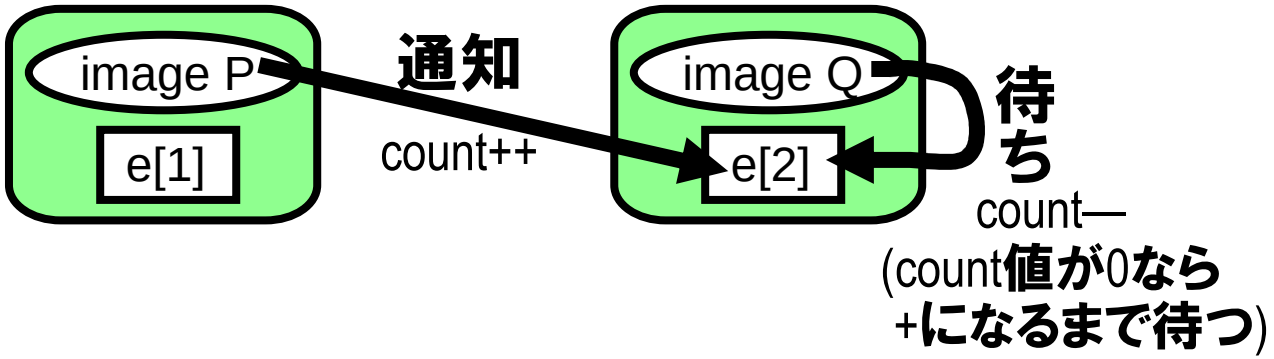
```
USE  ISO_FORTRAN_ENV
TYPE(TEAM_TYPE) :: t0, t1, t2
i = THIS_IMAGE()
FORM TEAM(0,t0,NEW_INDEX=i)
      :
CHANGE TEAM(t2)
   do k=0, TEAM_DEPTH()  !==2
      write(*)NUM_IMAGES(k)  ! 1,2,4
      write(*)THIS_IMAGE(k)    ! 1,2,4
   enddo
END TEAM
```



片方向同期(1)

EVENT

- **あるimage Pの処理の完了通知を，image Qが待つ**
- **image Pは，image Qの処理を待つ必要はない**



- **event変数 (EVENT_TYPE型)を使用して同期を行う**
  - **同期には，待つ側のimage上のcoarray event変数を使用する。**
  - **EVENT_TYPE型は，組込みモジュールISO_FORTRAN_ENV中で定義**
- **event変数は，成分にcount値(通知回数 – 待ち回数)を持つ**
  - **count値の初期値は0**
  - **組込みサブルーチン　EVENT_QUERY(event変数,COUNT [, STATUS])　で event変数のcount値を問合せ可能。**

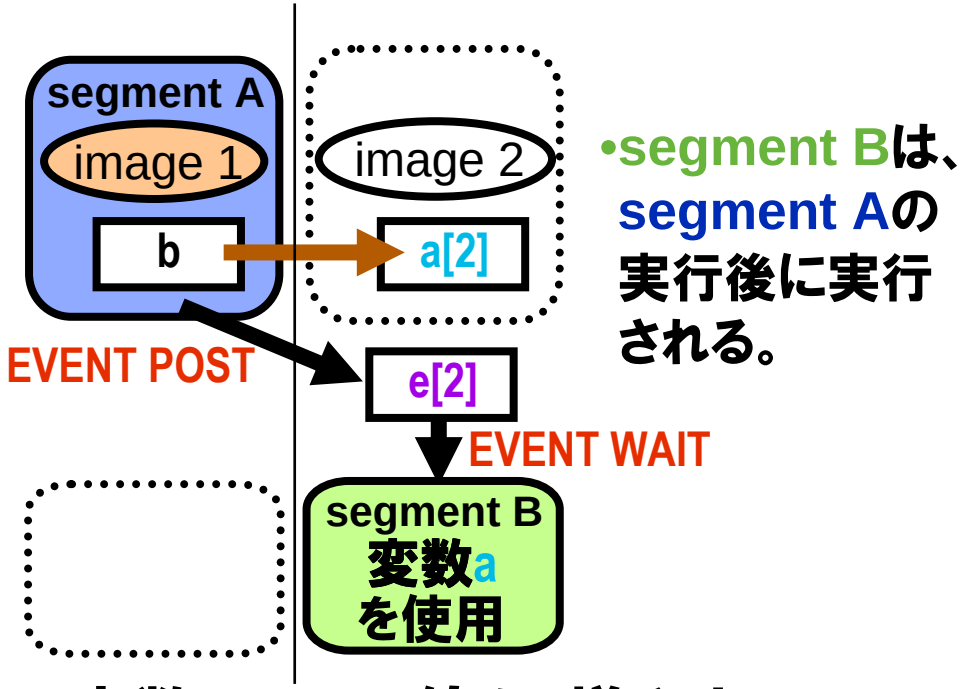
片方向同期(2)

EVENT POST文, EVENT WAIT文

EVENT POST (event変数 [, *sync-stat-list*])
EVENT WAIT (event変数 [, *sync-stat-list*])

例

```
USE  ISO_FORTRAN_ENV
TYPE(EVENT_TYPE) :: e[*]
real a[*]
        :
if(THIS_IMAGE().eq.1)then
    a[2] = b       ! aの値を設定
    EVENT POST(e[2])
else
    EVENT WAIT(e)
    call calc(a)  ! aの値を使用
endif
```



•segment Bは、segment Aの実行後に実行される。

- EVENT POST文は，event変数のcount値を1増やす
- EVENT WAIT文は，event変数のcount値が正になる迄待ち，count値を1減らす

集団通信組込みサブルーチン

ブロードキャスト

- CO_BROADCAST(SOURCE,SOURCE_IMAGE [,STAT,ERRMSG])
  - MPI_BCASTと同様(SOURCEがBUFFER, SOURCE_IMAGEがROOTに相当)

集計計算

- CO_SUM(SOURCE [,RESULT,RESULT_IMAGE,STAT,ERRMSG])
  - RESULT_IMAGEを省略した場合，MPI_ALLREDUCEと同様
    - SOURCEがSENDBUF，RESULTがRECVBUFに相当
  - RESULT_IMAGEを指定した場合，MPI_REDUCEと同様
    - RESULT_IMAGEがROOTに相当
  - RESULTを省略した場合，MPI_IN_PLACE指定に相当
    - SOURCEに結果が入る
  - CO_MAX, CO_MINもある
- CO_REDUCE(SOURCE,OPERATOR [,RESULT,RESULT_IMAGE,...
  - OPERATORには，2引数の純粋要素別処理関数を指定する。
    - 事実上，可換かつ結合則が成り立つ関数である必要がある

ATOMIC操作組込みサブルーチン

2項演算

- ATOMIC_ADD(ATOM, VALUE [, OLD])
  - ATOMの値が，ATOM+VALUEとなる。
  - 引数ATOMの型はATOMIC_INT_KIND。
    - ATOMIC_INT_KINDは組込みモジュールISO_FORTRAN_ENVで定義されている
  - OLDを指定すると，ATOMの元の値がOLDに保存される
  - bit演算(ATOMIC_AND, ATOMIC_OR, ATOMIC_XOR)もある

compare & swap

- ATOMIC_CAS(ATOM, OLD, COMPARE, NEW)
  - ATOM == COMPAREの場合
    - ATOM = NEW となる
  - 引数ATOMの型は，ATOMIC_INT_KIND 又は ATOMIC_LOGICAL_KIND
  - OLDには，ATOMの元の値が保存される

いずれも，引数ATOMに対する処理だけがatomicな処理となる。

- OLD引数の定義やVALUE引数の評価はatomicではない

フォールトトレラント機能

- 障害等により，coarrayの参照や集団処理への参加ができなくなったimage (failed image)が発生した場合にもプログラム実行が継続可能
  - failed imageは，プログラム終了までfailedのまま(復旧はしない)
- STAT_FAILED_IMAGE
  - 基本整数型の定数
    - 組込みモジュールISO_FORTRAN_ENVで定義
  - failed imageが検出された場合，STAT指定子に設定される

例
```
SYNC TEAM(t, STAT=j)
```

SYNC TEAM文の実行時に，subteam t 内でfailed imageが検出され かつ その他のエラーの発生がない場合，j の値がSTAT_FAILED_IMAGEとなる

- 組込み関数
  - NUM_IMAGES(FAILED=l)
    - FAILED引数が指定され，その値が真の場合，failed imageの数を返す。
    - それ以外の場合，failed imageを除いたimage数を返す。
  - FAILED_IMAGES([KIND])
    - failed imageのimage index値を(昇順に)持つ一次元配列を返す。

現在の状況

最新の仕様案は否決されたため，さらなる議論が必要

- 異なる(祖先の)subteam間のimage indexのマッピング

  FORM SUBTEAM(n, t, NEW_INDEX=i)

  のNEW_INDEX=iの様に，個々のimageに対して，image indexを指定する必要がある

  - NEW_INDEX=iを省略した場合，image indexの対応関係は規定されていない
- HWを意識したimageの部分集合(1つのSMPノード等)生成機能
- プログラム実行開始時のteam変数を簡単に取得可能にすべき
  - MPI_COMM_WORLDのように
- event変数は，2値に単純化すべきではないか
  - EVENT POST文とEVENT WAIT文の実行順序によりsegmentの順序が決定されるが，count値が0以外の場合，POSTとWAITが処理される順序が不明確
- atomic操作を明確に定義するためのメモリモデルが必要
- フォールトトレラント機能の有効な利用例

並列処理機能の拡張(現在議論中) まとめ

タスク並列

- FORM SUBTEAM文でsubteamを生成する
- CHANGE TEAM構文中で，subteam単位での処理を行う
- image indexはsubteamに閉じた値になる

片方向同期

- coarray event変数 (EVENT_TYPE型)を使用した同期
- EVENT POST文，EVENT WAIT文

集団通信組込み手続 (SUM, MAX, MIN, 汎用)

atomic操作組込み手続 (ADD, bit AND・OR・XOR, CAS)

フォールトトレラント機能

- 障害が発生したimageがあっても実行を継続可能
- 障害が発生したimage数等を問い合わせ可能