

XcalableMPワークショップ  
**COARRAYの便利な使い方**

2017年10月31日

富士通株式会社 )次世代TC開発本部

原口正寿

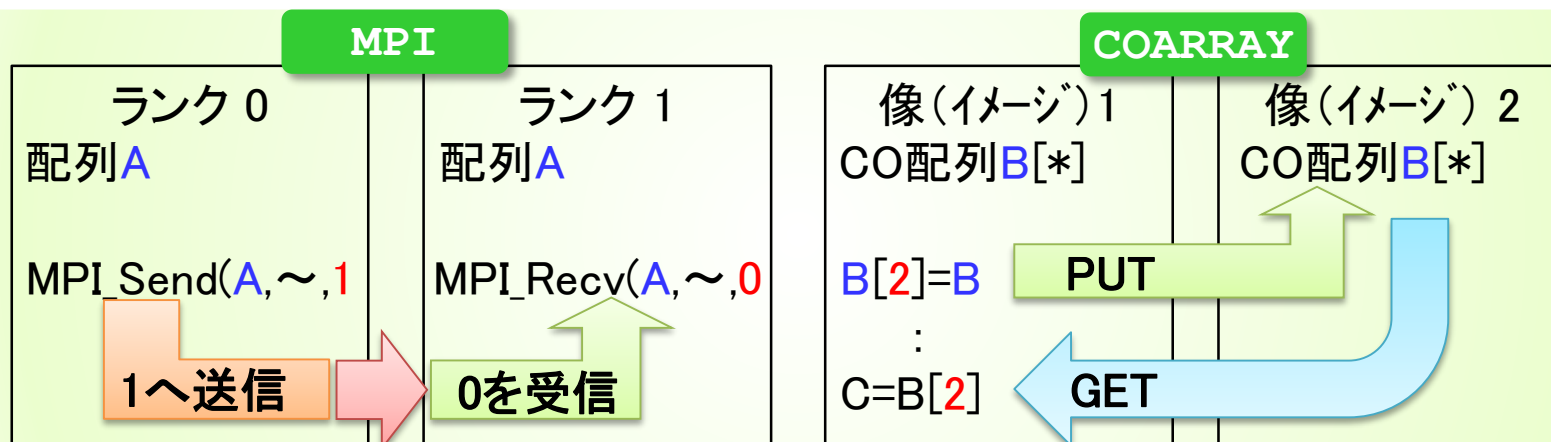
## ■ Fortran 2008に組み込まれた分散並列機能

- 指示文とサービスサブルーチンではなく、文法として組み込まれた
- “[”, “]” (角括弧)によるプロセス間通信と、同期のための文、アトミックサブルーチンなど組み込み手続

## ■ SPMDモデル (Single Program Multi Data)

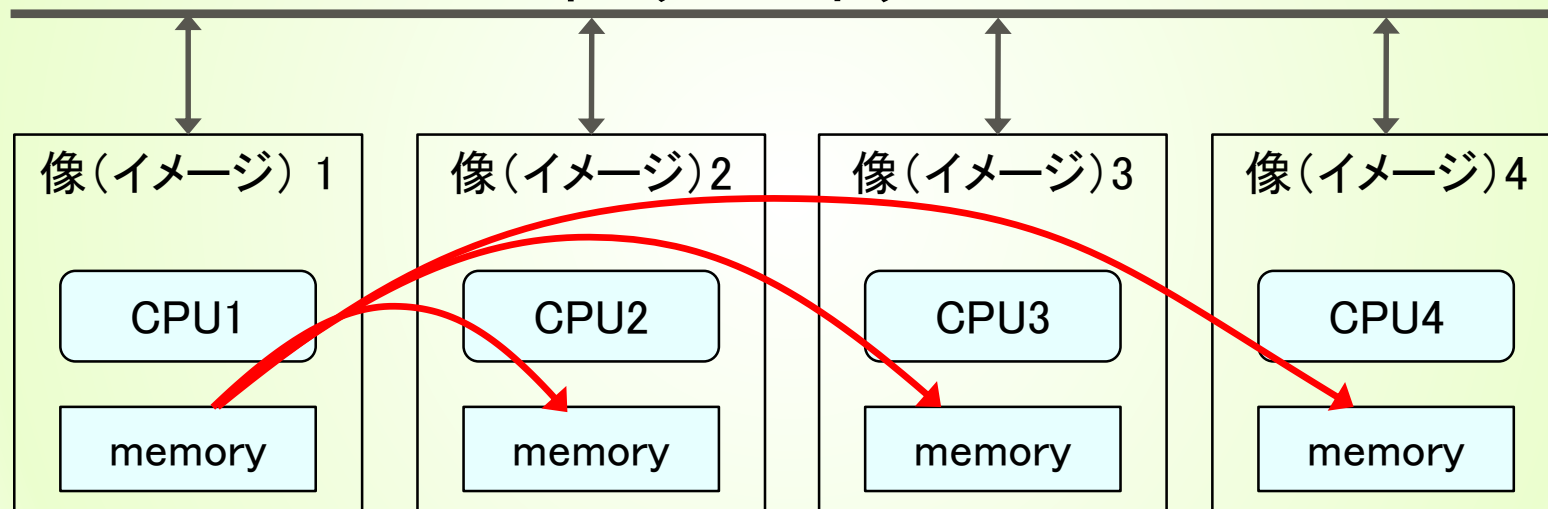
## ■ 明示的なデータ分散、計算分散、片側通信、同期

- COARRAYは通信相手の状態に関係無く他像のデータをアクセスする片側通信



【像1から像2~4にデータを渡したい】

インターコネクト



⇒ COARRAYでどう書くの? ... 任意の像(イメージ)上の動作をコーディング

```
real(kind=8), dimension(:), codimension[*], allocatable :: array
:
id = this_image()
:
if (id /= 1) then
  array(1:100) = array(1:100) [1]
endif
```

[特徴的な角括弧]

数字は像番号

COARRAYの配列

いかなる像からも直接参照または定義できる配列またはスカラー変数

## プログラムの記述性について

### ■ シンプル過ぎるCOARRAY仕様

#### ■ お客様の声

- 使い方がイメージできない
- 書きたいことが書けるのか？

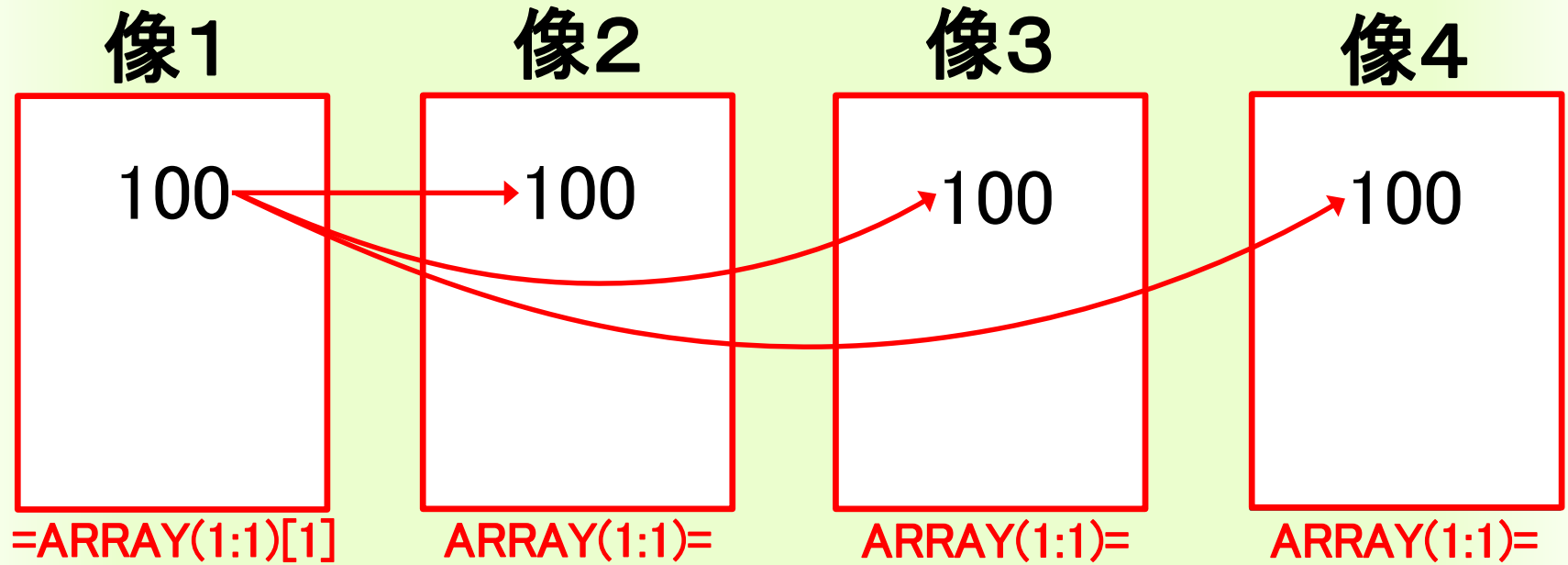
### ■ 集団通信や並列入出力の事例を紹介

#### ■ 主な集団通信

- ブロードキャスト
- ギャザー／スキッター
- 全対全通信
- リダクション演算

#### ■ 並列I/O

- ローカルファイルの読み書き
- 共有ファイルの読み書き



# 集団通信:ブロードキャスト(2)

## MPI

```
REAL (8), DIMENSION (:), ALLOCATABLE :: ARRAY
```

! 配列ARRAYの宣言

```
...
```

```
CALL MPI_COMM_SIZE (MPI_COMM_WORLD, NIMG, IERR)
```

! ランク数の取得

```
CALL MPI_COMM_RANK (MPI_COMM_WORLD, ID, IERR)
```

! ランク番号の取得

```
...
```

! ルートランクのARRAY配列をMSGSIZE要素分、全ランクにブロードキャスト

```
CALL MPI_BCAST (ARRAY, MSGSIZE, MPI_REAL8, 0, MPI_COMM_WORLD, IERR)
```

## COARRAY

```
REAL (8), DIMENSION (:), CODIMENSION [ : ], ALLOCATABLE :: ARRAY
```

! COARRAY配列ARRAYの宣言

```
...
```

```
NIMG= NUM_IMAGES ()
```

! 像数の取得

```
ID= THIS_IMAGE ()
```

! 像番号の取得

```
...
```

```
SYNC ALL
```

! 全イメージで同期

! 像1のCOARRAY配列ARRAYを自像のCOARRAY配列に代入(=ブロードキャスト)

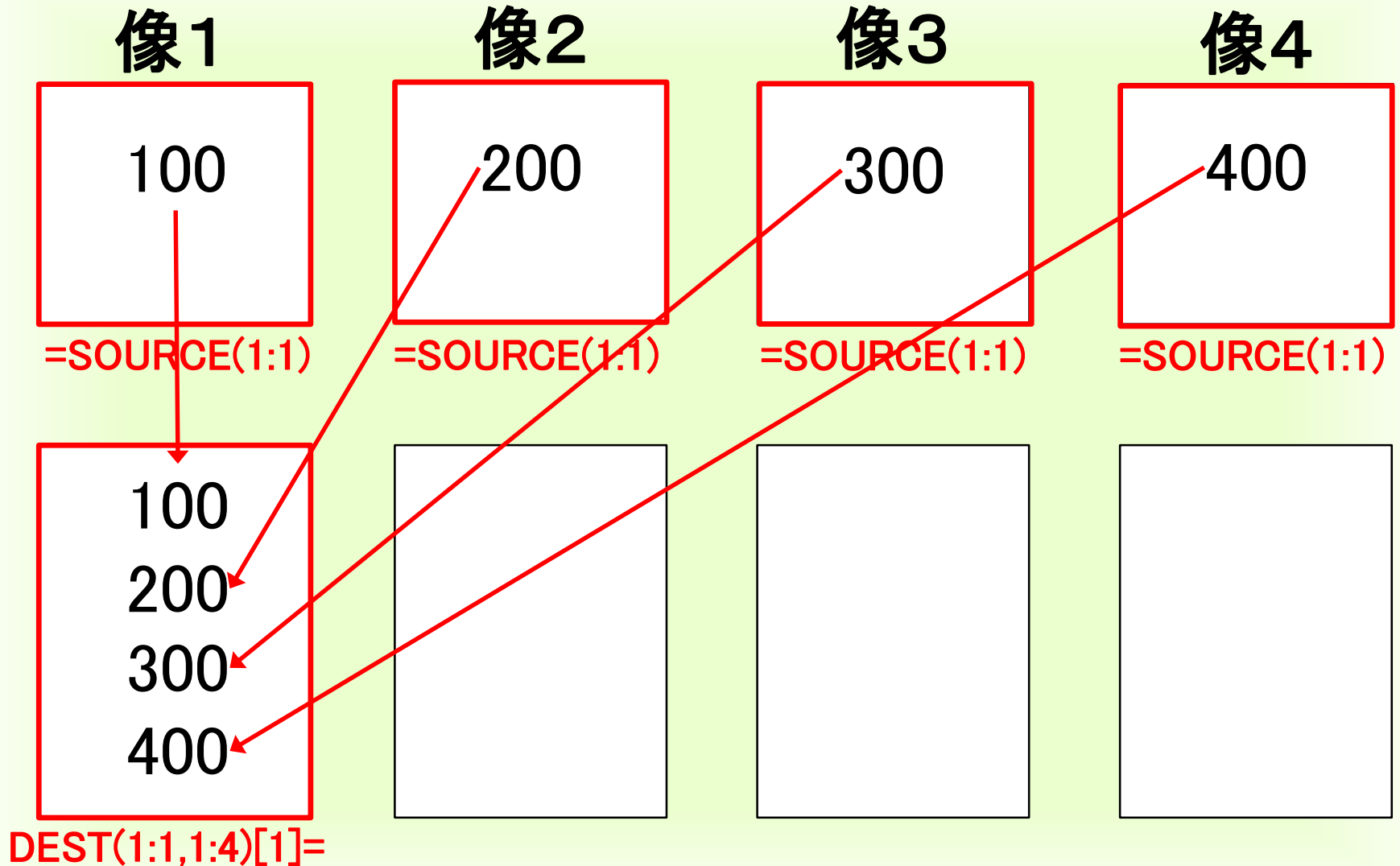
```
IF (ID /= 1) THEN
```

```
    ARRAY (1:MSGSIZE) = ARRAY (1:MSGSIZE) [1]
```

```
END IF
```

```
SYNC ALL
```

! 全イメージで同期



# 集団通信：ギャザー(2)

## MPI

```
REAL(8), DIMENSION(:, :), ALLOCATABLE :: DEST
REAL(8), DIMENSION(:), ALLOCATABLE :: SOURCE
...
CALL MPI_COMM_SIZE(MPI_COMM_WORLD, NIMG, IERR)
CALL MPI_COMM_RANK(MPI_COMM_WORLD, ID, IERR)
...
```

! 配列DESTの宣言

! 配列SOURCEの宣言

! ランク数の取得

! ランク番号の取得

! 配列SOURCEをMSGSIZE要素分送信し、受信ランク0の配列DESTで受信

```
CALL MPI_GATHER(SOURCE, MSGSIZE, MPI_REAL8, DEST, MSGSIZE,
                MPI_REAL8, 0, MPI_COMM_WORLD, IERR)
```

## COARRAY

```
REAL(8), DIMENSION(:, :), CODIMENSION[:], ALLOCATABLE :: DEST
REAL(8), DIMENSION(:), ALLOCATABLE :: SOURCE
...
NIMG= NUM_IMAGES()
ID= THIS_IMAGE()
...
SYNC ALL
```

! COARRAY配列DESTの宣言

! 配列SOURCEの宣言

! 像数の取得

! 像番号の取得

! 全像で同期

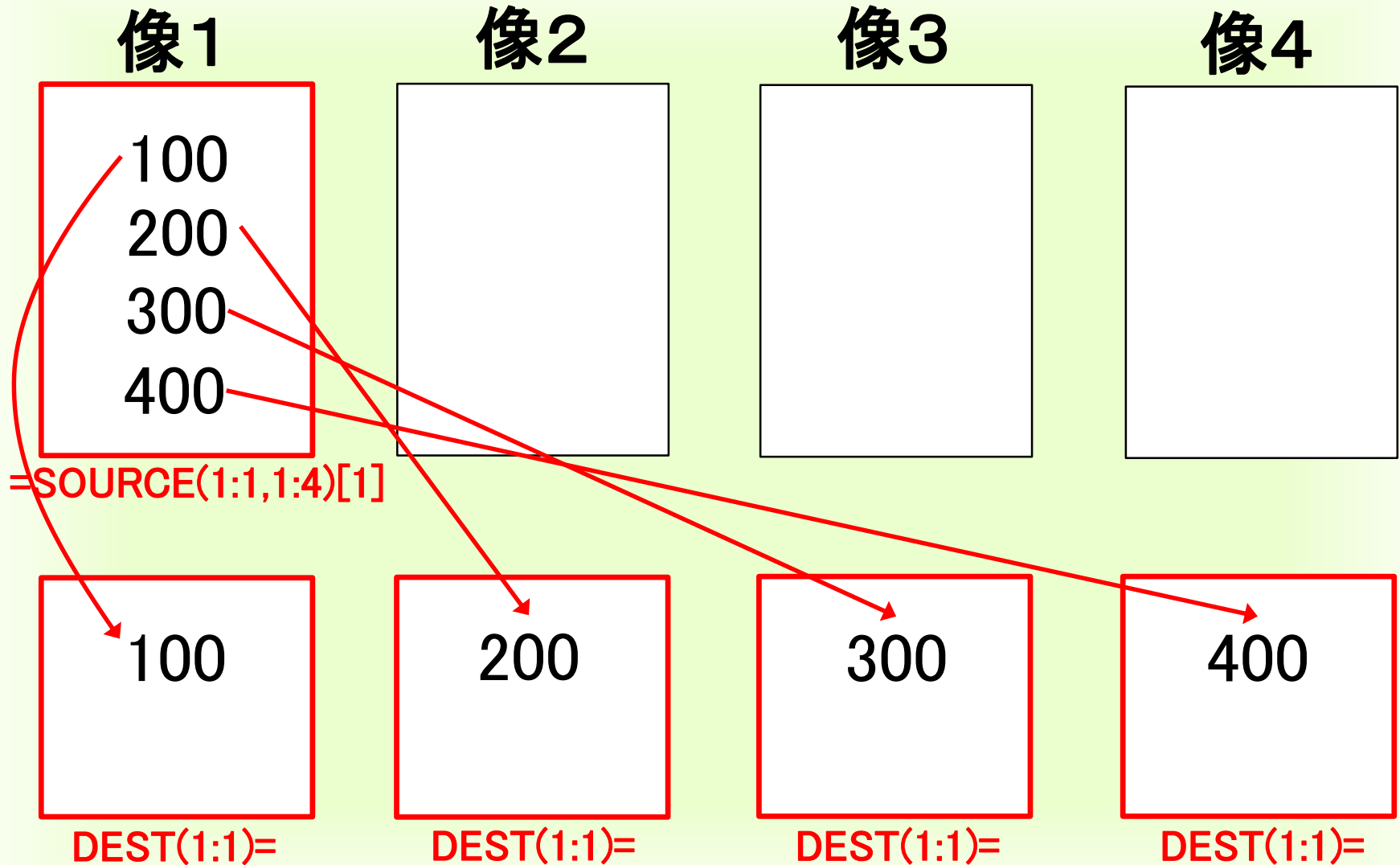
! 全像において、自像の配列SOURCEを像1の配列DESTの自像番号位置に代入(=ギャザー)

```
DEST(1:MSGSIZE, ID)[1] = SOURCE(1:MSGSIZE)
```

```
SYNC ALL
```

! 全像で同期





# 集団通信: スキャッター(2)

## MPI

```
REAL(8), DIMENSION(:, :), ALLOCATABLE :: DEST  
REAL(8), DIMENSION(:), ALLOCATABLE :: SOURCE
```

! 配列DESTの宣言

! 配列SOURCEの宣言

...

```
CALL MPI_COMM_SIZE(MPI_COMM_WORLD, NIMG, IERR)
```

! ランク数の取得

```
CALL MPI_COMM_RANK(MPI_COMM_WORLD, ID, IERR)
```

! ランク番号の取得

...

! ルートランクが送信する配列SOURCEのMSGSIZE要素分を各ランクの配列DESTで受信

```
CALL MPI_SCATTER(SOURCE, MSGSIZE, MPI_REAL8, DEST, MSGSIZE,  
                MPI_REAL_8, 0, MPI_COMM_WORLD, IERR)
```

## COARRAY

```
REAL(8), DIMENSION(:, :), CODIMENSION[:], ALLOCATABLE :: SOURCE
```

! COARRAY配列DESTの宣言

```
REAL(8), DIMENSION(:), ALLOCATABLE :: DEST
```

! 配列SOURCEの宣言

...

```
NIMG= NUM_IMAGES()
```

! 像数の取得

```
ID= THIS_IMAGE()
```

! 像番号の取得

...

```
SYNC ALL
```

! 全像で同期

! 全像において、像1のCOARRAY配列SOURCEにある自像向けの要素を

! ローカル配列DESTに代入(=スキャッター)

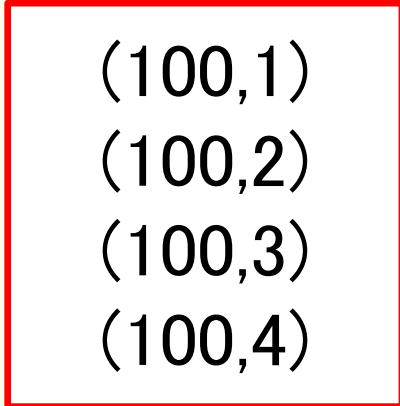
```
DEST(1:MSGSIZE) = SOURCE(1:MSGSIZE, ID) [1]
```

! 全像で同期

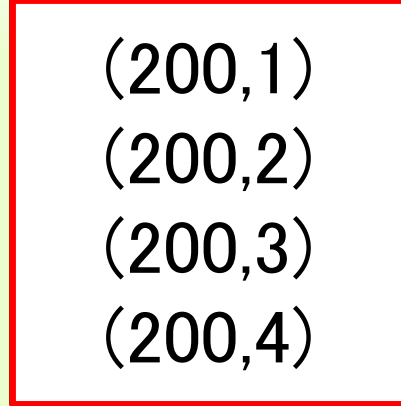
```
SYNC ALL
```

# 集团通信：全对全通信(1)

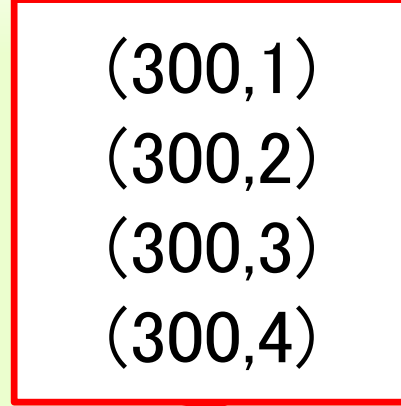
## 像1



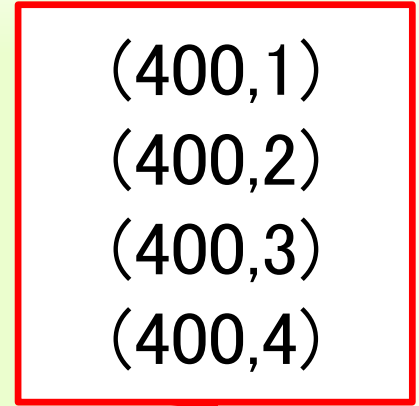
## 像2



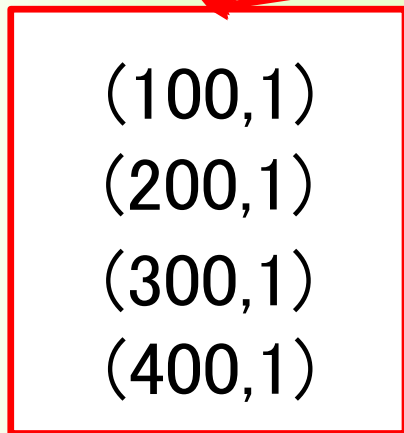
## 像3



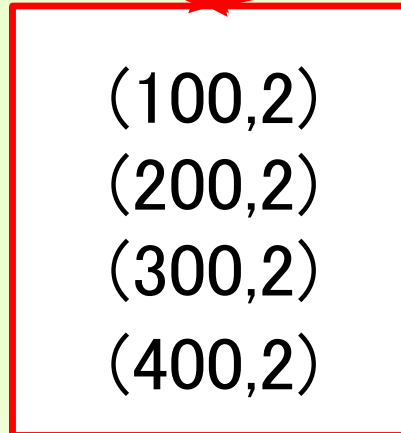
## 像4



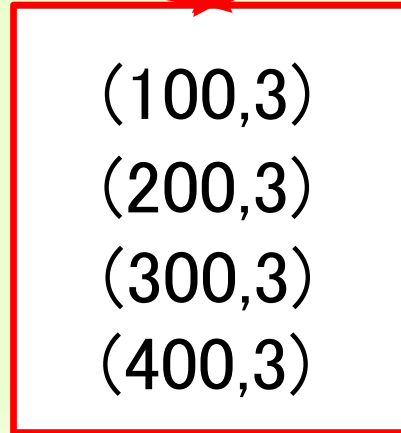
=SOURCE(1:1,1:4)   =SOURCE(1:1,1:4)   =SOURCE(1:1,1:4)   =SOURCE(1:1,1:4)



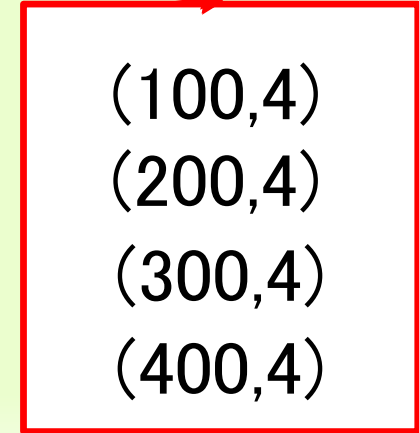
DEST(1:1,1)[1]=



DEST(1:1,2)[2]=



DEST(1:1,3)[3]=



DEST(1:1,4)[4]=

# 集団通信:全対全通信(2)

## MPI

```
REAL (8), DIMENSION (:, :), ALLOCATABLE :: DEST, SOURCE
```

...

```
CALL MPI_COMM_SIZE (MPI_COMM_WORLD, NIMG, IERR)
```

```
CALL MPI_COMM_RANK (MPI_COMM_WORLD, ID, IERR)
```

...

! 各ランクの配列SOURCEのMSGSIZE要素分をランク0から順に送信し、各ランクの配列DESTで受信  
! 全ランクが全ランクに対してスキッターしたのと同じ

```
CALL MPI_ALLTOALL (SOURCE, MSGSIZE, MPI_REAL8, DEST, MSGSIZE,  
MPI_REAL_8, MPI_COMM_WORLD, IERR)
```

! 配列DSEST,SOURCEの宣言

! ランク数の取得

! ランク番号の取得

## COARRAY

```
REAL (8), DIMENSION (:, :), CODIMENSION [:], ALLOCATABLE :: DEST
```

```
REAL (8), DIMENSION (:, :), ALLOCATABLE :: SOURCE
```

...

```
NIMG= NUM_IMAGES ()
```

```
ID= THIS_IMAGE ()
```

```
... SYNC ALL
```

! 全像において、自像の配列SOURCEの各像向けデータを、COARRAY配列DESTの  
! 自像番号位置に代入(=全対全通信)

```
DO I=1,NIMG
```

```
DEST (1:MSGSIZE, ID) [I] = SOURCE (1:MSGSIZE, I)
```

```
ENDDO
```

```
SYNC ALL
```

! COARRAY配列DESTの宣言

! 配列SOURCEの宣言

! 像数の取得

! 像番号の取得

! 全像で同期

! 全像で同期

## 像1

(100,1)
(100,2)
(100,3)
(100,4)

=SOURCE(1:1,1:4)

## 像2

(200,1)
(200,2)
(200,3)
(200,4)

=SOURCE(1:1,1:4)

## 像3

(300,1)
(300,2)
(300,3)
(300,4)

=SOURCE(1:1,1:4)

## 像4

(400,1)
(400,2)
(400,3)
(400,4)

=SOURCE(1:1,1:4)

**衝突**

(100,1)
(200,1)
(300,1)
(400,1)

DEST(1:1,1)[1]=

--

DEST(1:1,2)[2]=

--

DEST(1:1,3)[3]=

--

DEST(1:1,4)[4]=

# 集団通信: 全対全通信 (通信の衝突回避)

## COARRAY

```
INTEGER :: NIMG, ID, I, ISHIFT, MSGSIZE
REAL (8), DIMENSION (:, :), CODIMENSION [:], ALLOCATABLE :: DEST
REAL (8), DIMENSION (:, :), ALLOCATABLE :: SOURCE
...
NIMG=NUM_IMAGES ()
ID=THIS_IMAGE ()
...
ALLOCATE (DEST (MSGSIZE, NIMG) [*])
ALLOCATE (SOURCE (MSGSIZE, NIMG))
...
SYNC ALL
```

! COARRAY配列DESTの宣言

! 配列SOURCEの宣言

! 像数の取得

! 像番号の取得

! 全像で同期

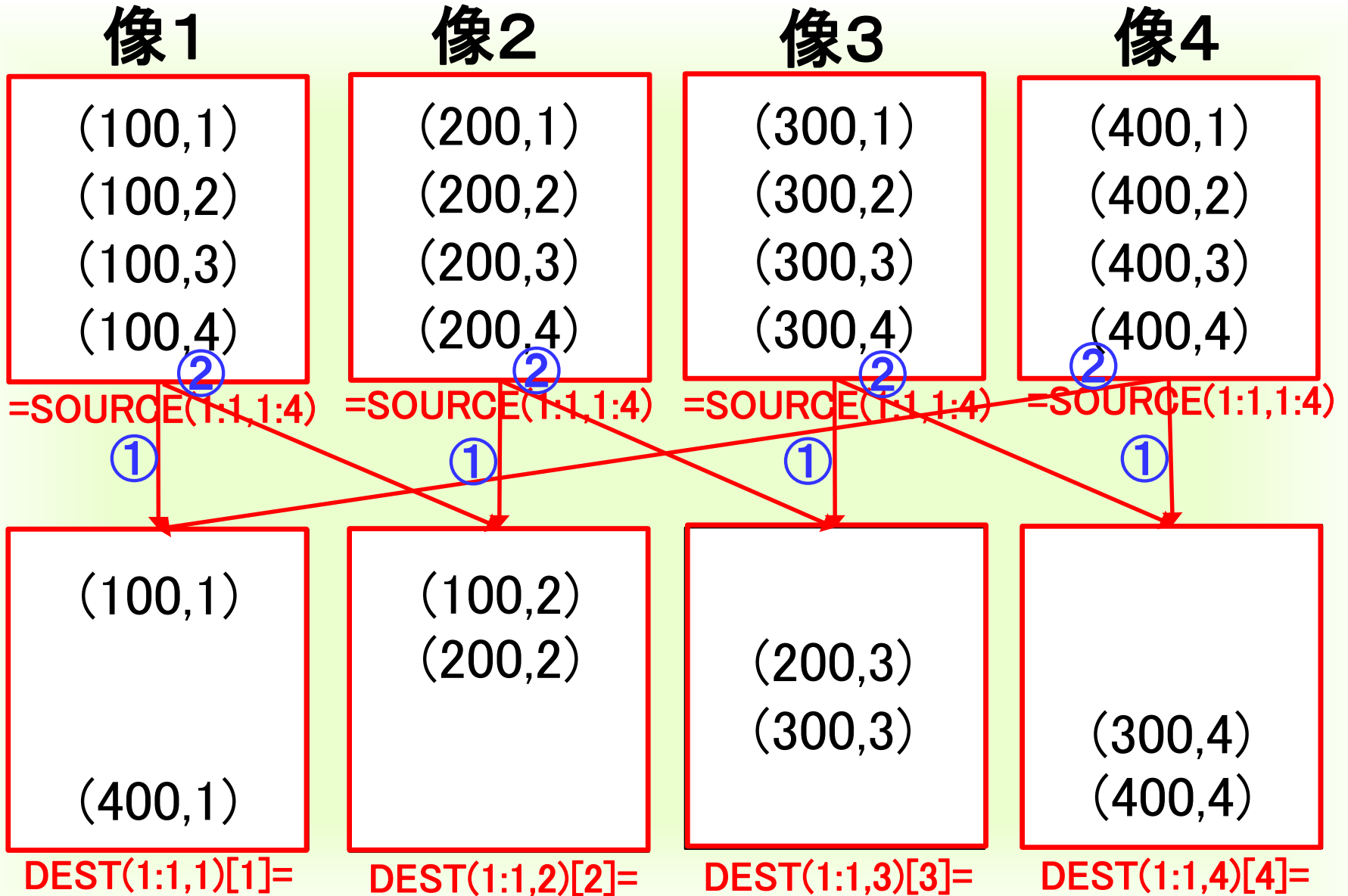
! 通信相手をずらして通信の衝突を避けると、前例より高速実行が期待できる

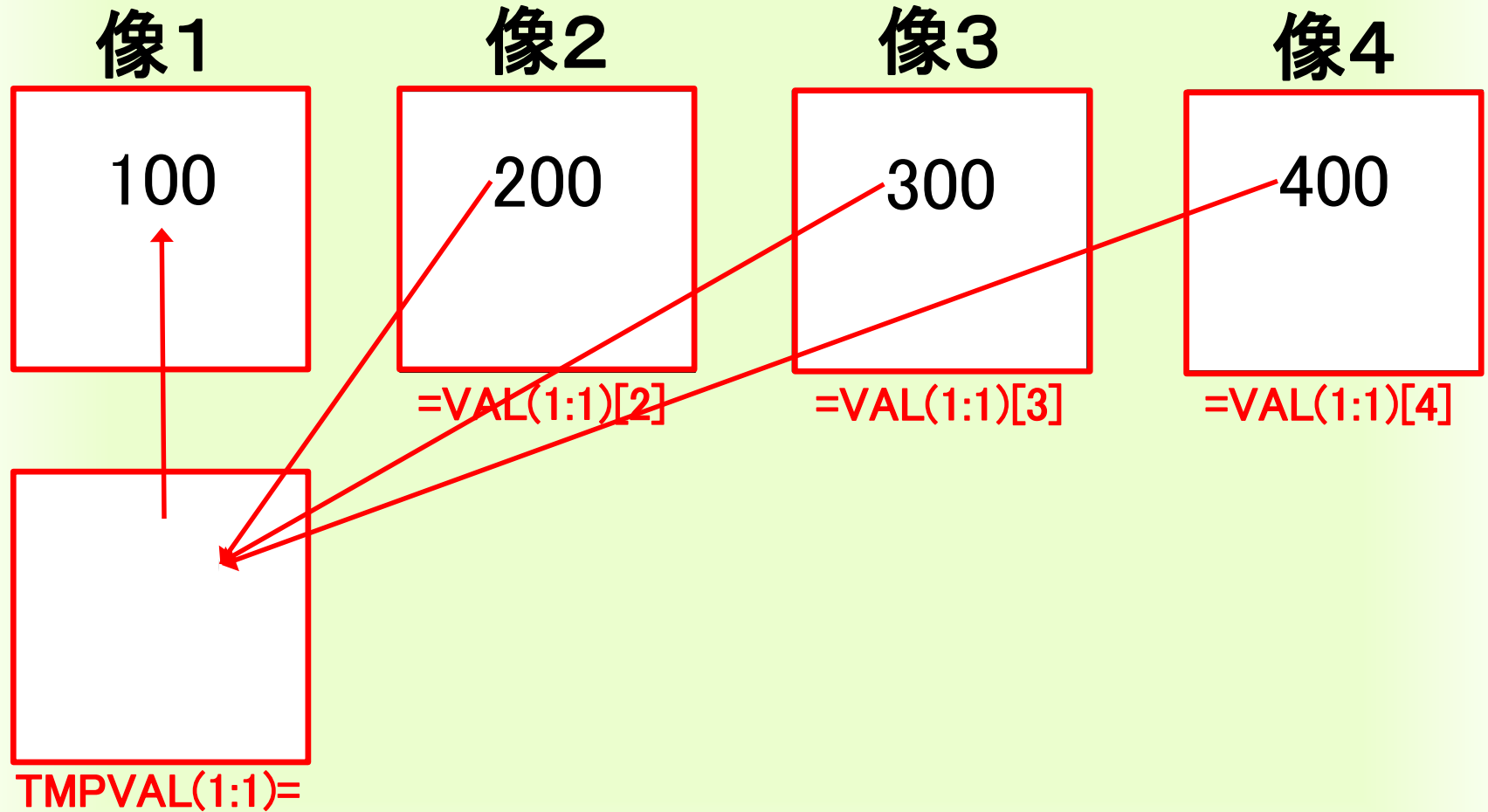
高速化

```
ISHIFT=ID
DO I=1, NIMG
  DEST (1:MSGSIZE, ID) [ISHIFT]=SOURCE (1:MSGSIZE, ISHIFT)
  ISHIFT=ISHIFT+1
  IF (ISHIFT>NIMG) THEN
    ISHIFT=1
  END IF
END DO
SYNC ALL
...
```

! 全像で同期

# 通信の衝突回避の結果







# 集団通信:リダクション演算(2)

## MPI

```
REAL(8), DIMENSION(:), ALLOCATABLE :: ORGVAL, VAL
```

! 配列ORGVAL,VALの宣言

...

```
CALL MPI_COMM_SIZE(MPI_COMM_WORLD, NIMG, IERR)
```

! ランク数の取得

```
CALL MPI_COMM_RANK(MPI_COMM_WORLD, ID, IERR)
```

! ランク番号の取得

...

! ORGVALの配列のMSGSIZE要素分だけ、総和演算して、その結果をルートランクのVAL配列に設定

```
CALL MPI_REDUCE(ORGVAL, VAL, MSGSIZE, MPI_REAL8,  
               MPI_SUM, 0, MPI_COMM_WORLD, IERR)
```

## COARRAY

! COARRAY配列VALの宣言

```
REAL(8), DIMENSION(:), CODIMENSION[:], ALLOCATABLE :: VAL
```

```
REAL(8), DIMENSION(:), ALLOCATABLE :: TMPVAL ...
```

! 配列TMPVALの宣言

```
NIMG= NUM_IMAGES()
```

! 像数の取得

```
ID= THIS_IMAGE() ...
```

! 像番号の取得

```
SYNC ALL
```

! 全像で同期

! 像1で、各像のCOARRAY配列をローカル配列に転送

! ローカル配列と自像のCOARRAY配列で要素毎に総和演算

```
IF (ID == 1) THEN
```

```
  DO I=2, NIMG
```

```
    TMPVAL(1:MSGSIZE) = VAL(1:MSGSIZE)[I]
```

```
    VAL(1:MSGSIZE) = VAL(1:MSGSIZE) + TMPVAL(1:MSGSIZE)
```

```
  ENDDO
```

```
ENDIF
```

```
SYNC ALL
```

! 全像で同期

## 複数の像が各像のローカルファイルを作成して並列実行

COARRAY

```
INTEGER, PARAMETER :: NMAX=1000
INTEGER :: ID, NIMG
CHARACTER(100) :: FNAME_W, FNAME_R
REAL(4), SAVE :: A(NMAX) [*], B(NMAX) [*]
...
NIMG=NUM_IMAGES()
ID=THIS_IMAGE()
...
```

! COARRAY配列A,Bの宣言

! 「A{像番号}.DAT」というファイルをオープンし、COARRAY配列Aの内容を書き込む

```
WRITE(FNAME_W, '("A", I4.4, ".DAT")') ID
OPEN(10, FILE=FNAME_W, FORM='UNFORMATTED')
WRITE(10) A
CLOSE(10)
...
```

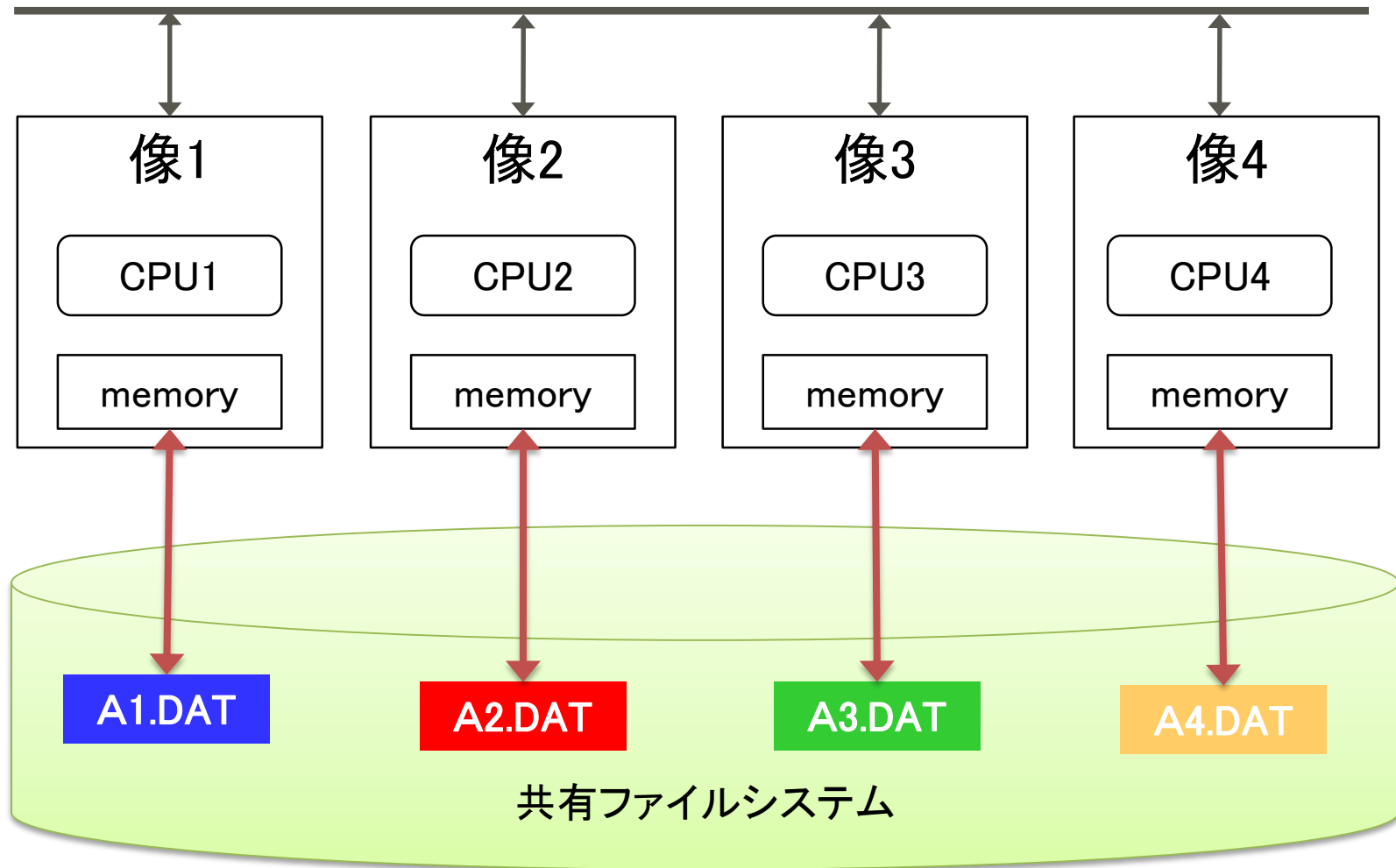
書き込み処理

! 「A{像番号}.DAT」というファイルをオープンし、COARRAY配列Bに読み込む

```
WRITE(FNAME_R, '("A", I4.4, ".DAT")') ID
OPEN(20, FILE=FNAME_R, FORM='UNFORMATTED')
READ(20) B
CLOSE(20)
```

読み込み処理

# 並列I/O: ローカルファイルの読み書き(2)



# 並列I/O: 共有ファイルの読み書き(1)

## 複数の像が1つの共有ファイルを並列にアクセス

COARRAY

```
INTEGER, PARAMETER :: NMAX=1000
INTEGER :: ID, NIMG
REAL(4), SAVE :: A(NMAX) [*]
REAL(4), DIMENSION(:, :), ALLOCATABLE :: D
NIMG=NUM_IMAGES()
ID=THIS_IMAGE()
SYNC ALL
```

! COARRAY配列Aの宣言

レコード長を各像  
の出力サイズに

! 自像のCOARRAY配列Aの大きさを1レコードとし、像番号を記録番号として出力

```
OPEN(30, FILE='A.DAT', ACCESS='DIRECT', FORM='UNFORMATTED', RECL=NMAX*4)
WRITE(30, REC=ID) A
CLOSE(30, STATUS='FSYNC') ! FSYNCでファイルの内容を同期
...
SYNC ALL
```

書き込み処理

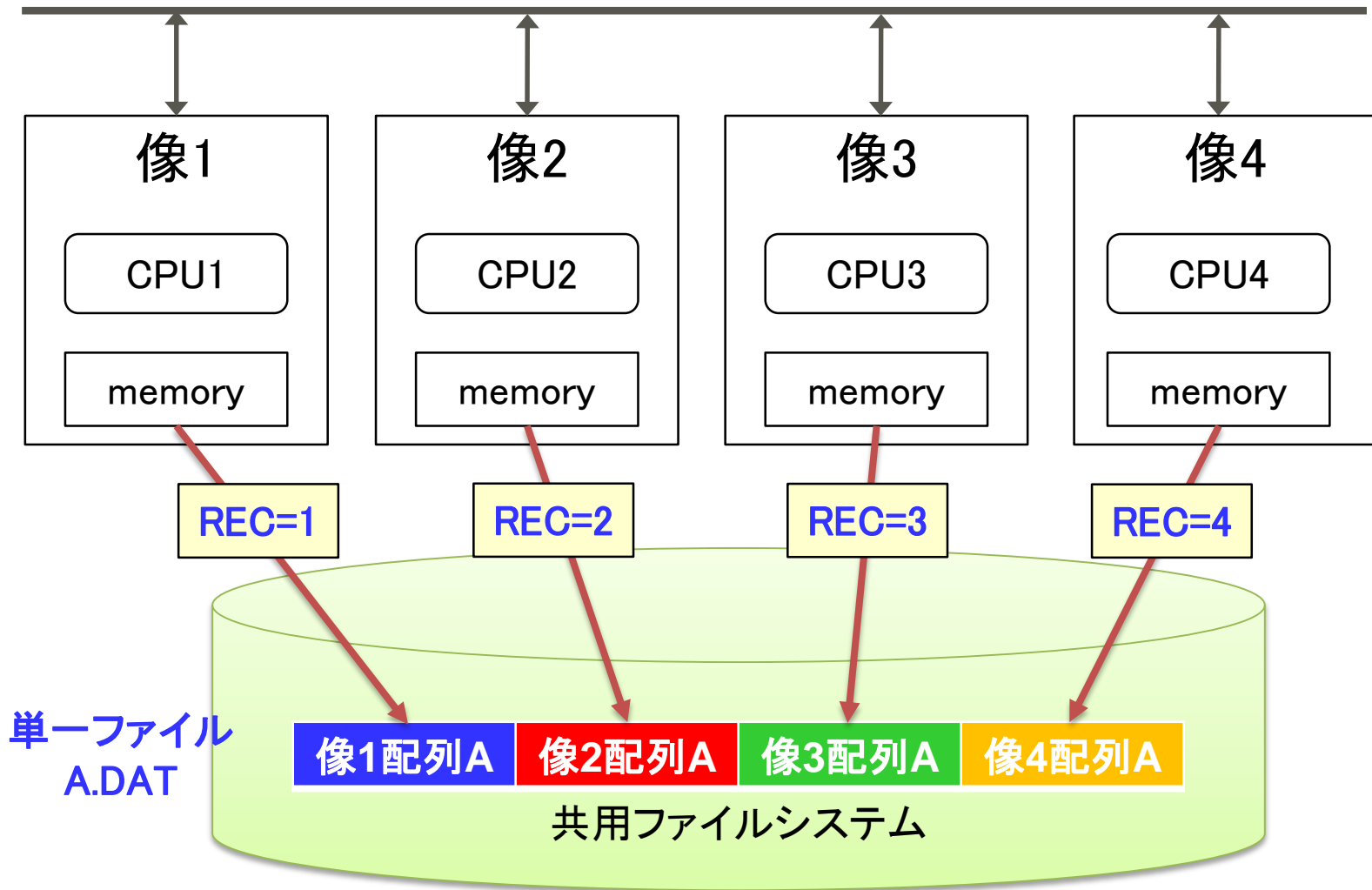
像番号でレコード位置を指定

! 像1が代表してファイルに出力した全像の配列Aを動的配列Dに読み込む

```
IF (ID==1) THEN
  ALLOCATE (D(NMAX, NIMG))
  OPEN(40, FILE='A.DAT', FORM='BINARY')
  READ(40) D
  CLOSE(40)
END IF
```

読み込み処理

# 並列I/O: 共有ファイルへの書き込み



# デバッグについて

- 対デッドロック性

## 【言語仕様のサポート】

同期文や集団サブルーチンには、STAT指定子を指定可能

⇒ 誤り状態を認識でき、MPIに比べデッドロックし難い

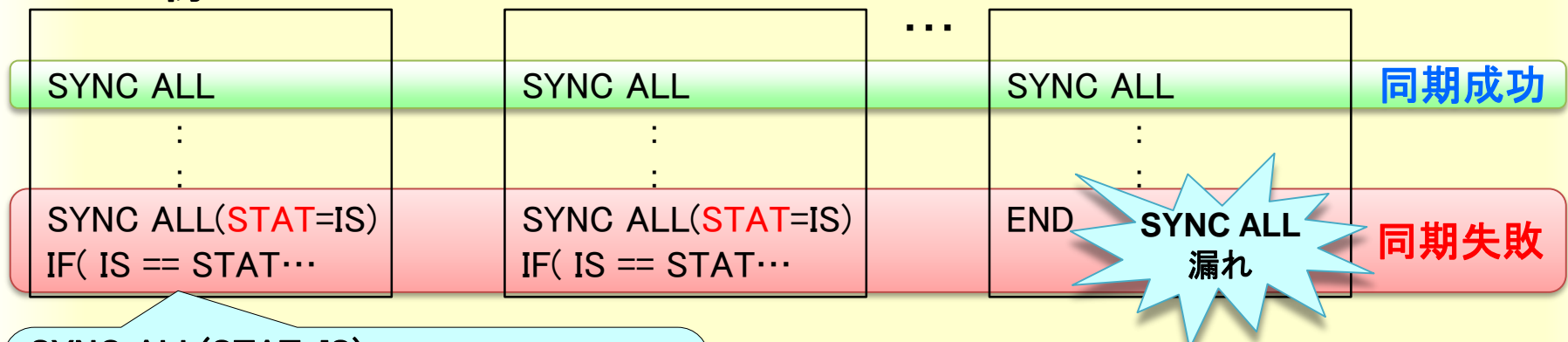
STAT指定子には、ISO\_FORTRAN\_ENV で定義された名前付き定数値が設定される。

値	意味
STAT_STOPPED_IMAGEなど	実行を終えた像との同期が必要になったことを示す

像1

像2

像N



```
SYNC ALL(STAT=IS)
IF (IS == STAT_STOPPED_IMAGE) THEN
  PRINT *, 'IMAGE=',ID,' stopped image'
ENDIF
```

ISに「STAT\_STOPPED\_IMAGE」が設定され  
SYNC ALL漏れの像の存在を認識可能

※ MPIの場合は、MPI\_BARRIER漏れするとデッドロックする。

# 富士通のCOARRAY

- サポート言語規格
- COARRAY機能とMPIとの親和性
- デバッグ支援



## ■ サポート言語規格

- Fortran 2008のCOARRAY仕様をサポート
- Fortran 2015のCOARRAY仕様も一部サポート(次頁で紹介)  
co\_sum, co\_max, co\_min

## ■ COARRAY機能とMPIとの親和性

- MPIと共通の基盤を使用しているため、同一関数内でも混在が可能(次々頁で紹介)
- COARRAYで書いて一部をMPI化することも可能
- mpiexecで実行バイナリが実行可能

## Fortran 2015の先取り(co\_sum,co\_max,co\_min)

**Fortran 2008** (前例のリダクション演算のカーネル部分)

```
...
IF (ID == 1) THEN
  DO I=2,NIMG
    TMPVAL(1:MSGSIZE) = VAL(1:MSGSIZE)[I]
    VAL(1:MSGSIZE) = VAL(1:MSGSIZE) + TMPVAL(1:MSGSIZE)
  ENDDO
ENDIF
...
```

! Fortran 2008レベルのプログラミング

**Fortran 2015**

```
REAL(8), DIMENSION(:), CODIMENSION[:], ALLOCATABLE
```

! COARRAY配列VALの宣言

```
...
NIMG= NUM_IMAGES()
ID= THIS_IMAGE()
...
SYNC ALL
```

! 像数の取得

! 像番号の取得

! 全像で同期

! COARRAY配列VALの全像で総和し、結果を「RESULT\_IMAGE=」で指定した像1のVALに設定

```
CALL CO_SUM(VAL(1:MSGSIZE), RESULT_IMAGE=1)
```

```
SYNC ALL
```

! 全像で同期

## COARRAYで書いて一部MPI化したサンプル

COARRAY

+

MPI

```
INCLUDE 'mpif.h'  
REAL (8) ::M_VAL,R_VAL  
REAL (8) ,SAVE ::VAL[*]
```

! MPIヘッダファイルのインクルード

MPI

...

```
NIMG=NUM_IMAGES()  
ID=THIS_IMAGE()  
NID=ID+1; IF(NID > NIMG) NID=1
```

! NIDに次の像番号を設定

...

```
VAL[NID]=DBLE(ID)
```

! 次像のCOARRAY配列VALに値設定(PUT)

COARRAY

```
SYNC ALL
```

```
// 全像からVAL[1]のチェック
```

```
IF(VAL[1] /= DBLE(NIMG)) THEN
```

! 像1のCOARRAY配列VALの値参照(GET)

COARRAY

```
// エラー処理
```

```
ENDIF
```

! COARRAY配列VALを全ランク積(MPI\_PROD)し結果をランク0のM\_VALに設定

MPI

```
CALL MPI_REDUCE(VAL,M_VAL,1,MPI_REAL8,MPI_PROD,0,MPI_COMM_WORLD,IERR)
```

...

```
IF(ID == 1) THEN
```

```
PRINT *,M_VAL
```

```
ENDIF
```

## 【MPIと同様のデバッグの難しさ】

プログラムの  
不具合発生

大量のエラー  
メッセージ

不具合の根本的な  
原因特定が難しい

時間軸を頼りにエラー発生の根本原因を探ることが多い

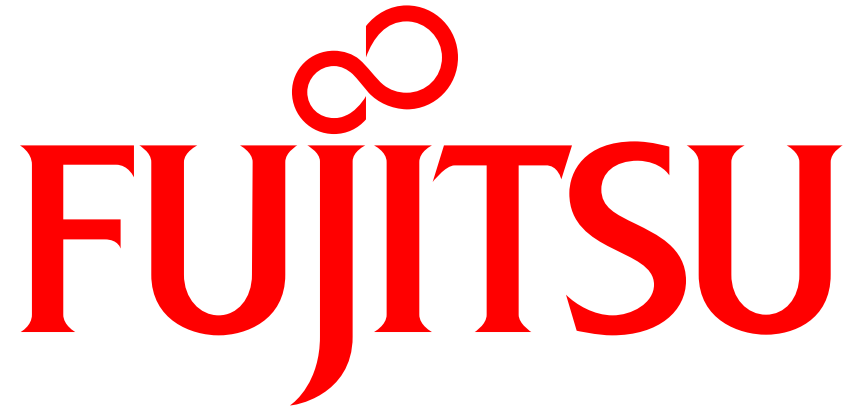
タイムスタンプおよび像番号と実行文を特定しやすい  
エラーメッセージをサポート

タイムスタンプ[**像番号**/像数:プロセスID]エラーメッセージ

```
1507791882.059731[ 2/16:6177] jwe1731i-s line 12 The SYNC ALL statement was ...
1507791882.059731[ 1/16:6176] jwe1731i-s line 12 The SYNC ALL statement was ...
1507791882.432477[ 2/16:6177] error occurs at MAIN__ line 12 loc 0000000000400b...
1507791882.432492[ 2/16:6177] MAIN__ at loc 0000000000400ad0 called from o.s.
1507791882.432504[ 2/16:6177] jwe0903i-u Error number 1731 was detected. Maxi...
1507791882.432546[ 1/16:6176] error occurs at MAIN__ line 12 loc 0000000000400b...
1507791882.432556[ 1/16:6176] MAIN__ at loc 0000000000400ad0 called from o.s.
1507791882.432567[ 1/16:6176] jwe0903i-u Error number 1731 was detected. Maxi...
```

- COARRAYは、Fortran 2008で正式文法として採用
- COARRAYの登場で、ちょっとした文法を覚えるだけで、MPIを使わなくても分散並列プログラムが記述できる
- 複雑な通信や入出力も簡単に記述できる

**皆さんもCOARRAYを試してみてください**



**shaping tomorrow with you**