REFINING FORTRAN FAILED IMAGES

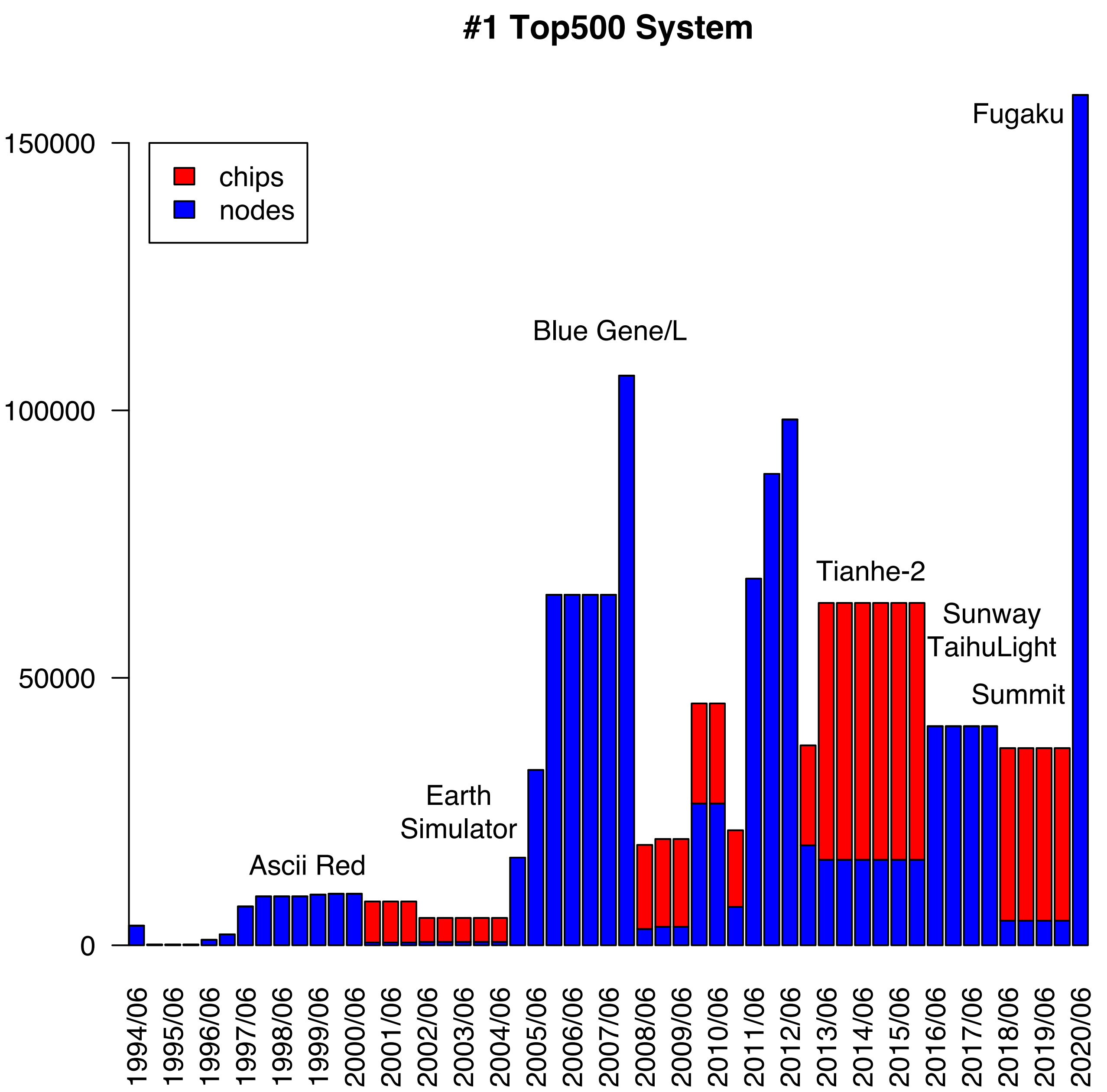
Nathan Weeks, Glenn Luecke, Gurpur Prabhu

Fifth International IEEE Workshop on Extreme Scale Programming Models and Middleware (ESPM2 2020)
November 11, 2020

PRESENTATION TOPICS

- Fortran 2018 features for fault-tolerant parallel programming
- Prototype implementation (“OpenCoarrays-ft”)
- Proposed changes to Fortran 2018 standard

MOTIVATION


#1 Top500 System
chips
nodes
150000
100000
50000
0
Fugaku
Blue Gene/L
Tianhe-2
Sunway
TaihuLight
Summit
Earth
Simulator
Ascii Red
1994/06
1995/06
1996/06
1997/06
1998/06
1999/06
2000/06
2001/06
2002/06
2003/06
2004/06
2005/06
2006/06
2007/06
2008/06
2009/06
2010/06
2011/06
2012/06
2013/06
2014/06
2015/06
2016/06
2017/06
2018/06
2019/06
2020/06

For Tianhe-2, the MTBF is about 2h on 8192 nodes.

Chen et al., Toward fault-tolerant hybrid programming over large-scale heterogeneous clusters via checkpointing/restart optimization (2019)

> ***On average [over 261 days], an application failure caused by a system-related issue [on Blue Waters] occurs every 15 min…***
>
> *Di Martino et al., Measuring the Resiliency of Extreme-Scale Computing Environments (2016)*

> ***1.53% of applications failed due to system problems, these… account for about 9% of total production node hours.***
>
> *Di Martino et al., Measuring the Resiliency of Extreme-Scale Computing Environments (2016)*

> *…an increase of 20x in the application failure probability… when scaling XE applications from 10,000 to 22,000 nodes.*
>
> *Di Martino et al., Measuring the Resiliency of Extreme-Scale Computing Environments (2016)*

> ***…failed applications that are not recovered through checkpoint/restart add potentially $421,878 to the Blue Waters energy bill…***
>
> *Di Martino et al., Measuring the Resiliency of Extreme-Scale Computing Environments (2016)*

> ***Therefore, the impact of system errors on applications and costs of ownership is substantial and destined to grow for larger machines.***
>
> *Di Martino et al., Measuring the Resiliency of Extreme-Scale Computing Environments (2016)*

STATE OF THE PRACTICE

STATE OF THE PRACTICE

COORDINATED CHECKPOINT/RESTART

- Processes periodically save state off node
- *Global rollback (backward) recovery* if a process fails:
  - All (non-failed) processes terminate
  - Newly-launched processes read state and resume execution

EXAMPLES OF OTHER RECOVERY MODELS

- Local non-shrinking backward recovery
  - Can use in-memory checkpoint from peer/buddy process
- Forward shrinking recovery

EXASCALE COMPUTING PROJECT (ECP) 2017 SURVEY

Applications	**28**
Use MPI	28
Call MPI from Fortran	13

> *MPI itself provides no mechanisms for handling processor failures.*
>
> *MPI 3.1 (2015)*

- MPI Extensions:
  - Reinit
  - **User-Level Failure Mitigation (ULFM)**

FORTRAN 2018

FORTRAN 2018

FAILED IMAGES

FORTRAN PROCESSORS SUPPORTING FAILED IMAGES

- OpenCoarrays
  - partial support, to allow an application to detect & exit
- Intel Fortran 19.1
- NAG Fortran Compiler 7.0
  - Single node

INTRODUCING OPENCOARRAYS-FT

- Prototype extensions to OpenCoarrays
- Adds support for:
    - FORM TEAM
        - New teams excludes failed images
        - NEW_INDEX=
    - STAT= and ERRMSG= in:
        - FORM TEAM
        - CHANGE TEAM / END TEAM
        - SYNC TEAM
        - CRITICAL
    - ULFM2 (Open-MPI)

OVERVIEW OF FORTRAN 2018 FEATURES FOR FAULT-TOLERANT PARALLEL PROGRAMMING

Fortran	MPI
image	process
image index	rank
team	communicator

An *image* can be in one of 3 states:

Failed

Fail-stop/crash failure

Stopped

- Reached end of program or *STOP* statement
- Coarray data can still be accessed by active images

Active

An image that has neither stopped nor failed

FAILED IMAGES SEMANTICS

```
integer :: X[*]
...
X = X[1]
```

FAILED IMAGES SEMANTICS

```
use intrinsic :: iso_fortran_env, only: STAT_FAILED_IMAGE
integer :: s, X[*]
...
X = X[1, STAT=s]
if (s == STAT_FAILED_IMAGE) then
    ... handle image failure ...
```

FAILED IMAGES SEMANTICS

IMAGE CONTROL STATEMENTS

An *image control statement* “affects the execution ordering between images” ($\implies$ synchronization)

FAILED IMAGES SEMANTICS

IMAGE CONTROL STATEMENTS

```
SYNC TEAM(STAT=s)
if (s == STAT_FAILED_IMAGE) then
    failed = FAILED_IMAGES()
    ... handle image failure ...
```

FAILED IMAGES SEMANTICS

IMAGE CONTROL STATEMENTS

```
SYNC TEAM
! error termination of image failure detected
```

IMAGE CONTROL STATEMENTS

> *If the STAT= specifier appears in… a CHANGE TEAM, END TEAM, EVENT POST, FORM TEAM, SYNC ALL, SYNC IMAGES, or SYNC TEAM statement… [and] one of the images involved has failed…* ***the intended action is performed on the active images involved*** *and stat-variable is assigned the value STAT_FAILED_IMAGE…*

If Fortran 2018 requires many image control statements to perform their intended action in the presence of failed images

IMAGE CONTROL STATEMENTS

> *If the STAT= specifier appears in… a **CHANGE TEAM, END TEAM,** ~~EVENT POST~~, **FORM TEAM, SYNC ALL,** ~~SYNC IMAGES~~, or **SYNC TEAM** statement… [and] one of the images involved has failed… the intended action is performed on the active images involved and stat-variable is assigned the value STAT_FAILED_IMAGE…*

Some of these image control statements involve synchronization among all images in a team.

TEAM SYNCHRONIZATION WITH STAT=

```
// OpenCoarray-ft implementation
MPI_Comm team_comm;
...
int rc, flag = 1;
do {
  MPIX_Comm_failure_ack(team_comm);
  rc = MPIX_Comm_agree(team_comm, &flag);
} while (rc != MPI_SUCCESS);
MPIX_Comm_failure_get_acked(team_comm, &failed_group);
MPI_Group_size(failed_group, &num_failed_in_group);
if (num_failed_in_group > 0) {
    *stat = STAT_FAILED_IMAGE;
... translate ranks to MPI_COMM_WORLD (initial team)
... and add to MPI group of known process failures
```

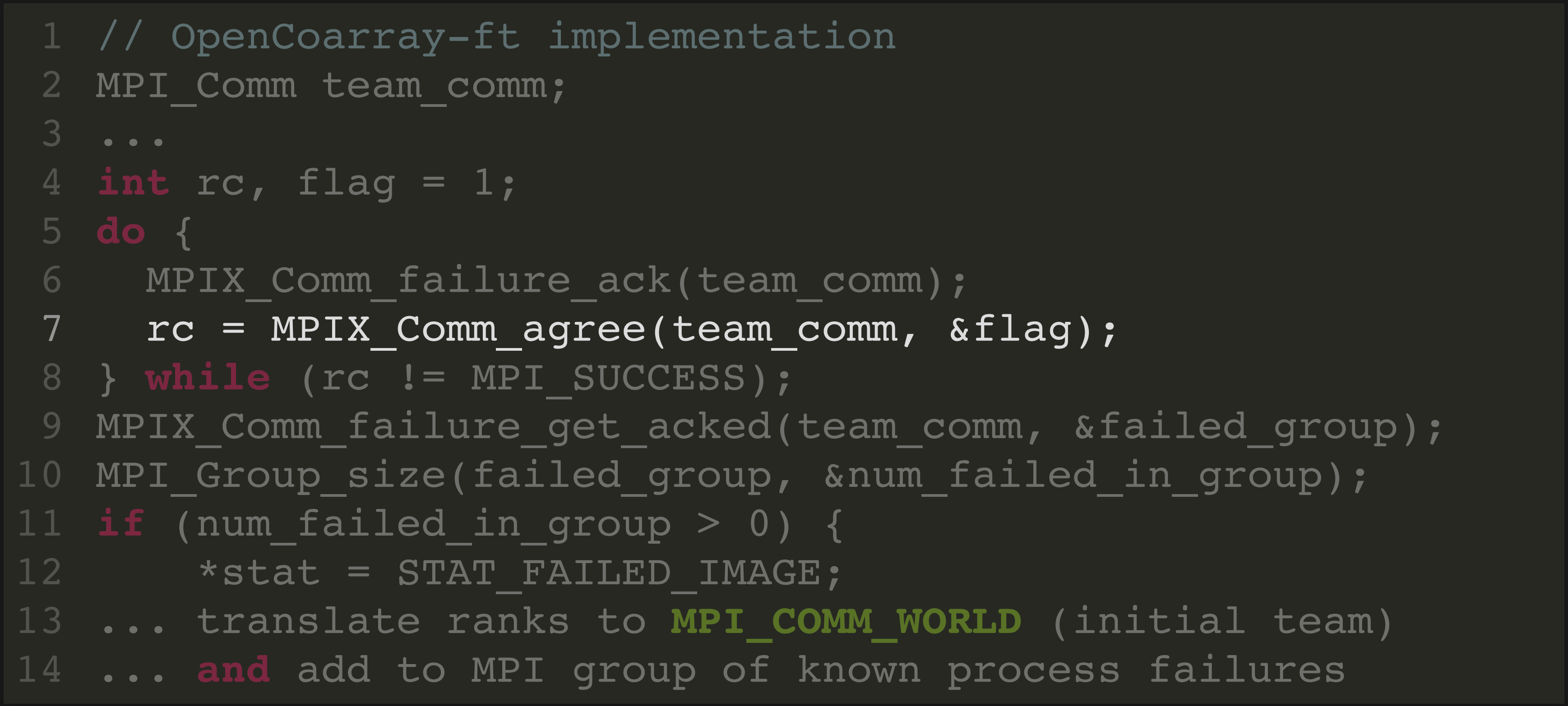
See ULFM spec: “Fault-Tolerant Consistent Group of Failures Example (Agree variant)”

TEAM SYNCHRONIZATION WITH STAT=

```
// OpenCoarray-ft implementation
MPI_Comm team_comm;
...
int rc, flag = 1;
do {
  MPIX_Comm_failure_ack(team_comm);
  rc = MPIX_Comm_agree(team_comm, &flag);
} while (rc != MPI_SUCCESS);
MPIX_Comm_failure_get_acked(team_comm, &failed_group);
MPI_Group_size(failed_group, &num_failed_in_group);
if (num_failed_in_group > 0) {
    *stat = STAT_FAILED_IMAGE;
... translate ranks to MPI_COMM_WORLD (initial team)
... and add to MPI group of known process failures
```

MPI_COMM_FAILURE_ACK *acknowledges* process failures in *team_comm* detected by the caller

TEAM SYNCHRONIZATION WITH STAT=



```
// OpenCoarray-ft implementation
MPI_Comm team_comm;
...
int rc, flag = 1;
do {
  MPIX_Comm_failure_ack(team_comm);
  rc = MPIX_Comm_agree(team_comm, &flag);
} while (rc != MPI_SUCCESS);
MPIX_Comm_failure_get_acked(team_comm, &failed_group);
MPI_Group_size(failed_group, &num_failed_in_group);
if (num_failed_in_group > 0) {
    *stat = STAT_FAILED_IMAGE;
... translate ranks to MPI_COMM_WORLD (initial team)
... and add to MPI group of known process failures
```

MPI_COMM_AGREE: fault-tolerant consensus

1. MPI_ALLREDUCE w/ MPI_BAND on *flag* (unused)
2. Synchronizes acknowledged failed processes

TEAM SYNCHRONIZATION WITH STAT=

```
// OpenCoarray-ft implementation
MPI_Comm team_comm;
...
int rc, flag = 1;
do {
  MPIX_Comm_failure_ack(team_comm);
  rc = MPIX_Comm_agree(team_comm, &flag);
} while (rc != MPI_SUCCESS);
MPIX_Comm_failure_get_acked(team_comm, &failed_group);
MPI_Group_size(failed_group, &num_failed_in_group);
if (num_failed_in_group > 0) {
    *stat = STAT_FAILED_IMAGE;
... translate ranks to MPI_COMM_WORLD (initial team)
... and add to MPI group of known process failures
```

TEAM SYNCHRONIZATION WITH STAT=

- Propagates a consistent knowledge of failed images in team
  - `FAILED_IMAGES()` returns list of images (in current team) known by caller to have failed
  - Fortran 2018 requires only at least 1 failed image

TEAM SYNCHRONIZATION WITH STAT=

Rationale:

1. OpenCoarrays-ft doesn’t reliably support detecting new image failures in coarray operations, e.g. `X = X[1, STAT=s]`
   - ULFM2 lacks explicit support for detecting process failure in MPI one-sided operations
2. Consistent knowledge of image failure can aid recovery
   - And not that costly, as we’ll see later…

COLLECTIVE SUBROUTINES

defn:

intrinsic subroutine that performs a calculation on a team of images *without requiring synchronization*

- `CO_BROADCAST`, `CO_MAX`, `CO_MIN`, `CO_REDUCE`, `CO_SUM`

COLLECTIVE SUBROUTINES

```
call co_sum(A, STAT=s)
```

- `s == STAT_FAILED_IMAGE`:
  - May be true for subset of images in current team[1]
  - Result (*A*) is undefined
  - Current team cannot be used for collectives
    - Would need to form a new team w/o failed images

[1] Fortran 2018: implies all images see the same STAT= value; will change in future standard

FORM TEAM

```
use, intrinsic: iso_fortran_env, only: team_type
type(team_type) :: team_variable
integer :: team_number
...
FORM TEAM (team_number, new_team)
```

Similar to:

```
call MPI_Comm_split(comm    = current_team, &
                    color   = team_number,  &
                    key     = 0,            &
                    newcomm = new_team)
```

CHANGE TEAM CONSTRUCT

```
use, intrinsic: iso_fortran_env, only: team_type
type(team_type) :: team_variable
integer :: team_number
...
FORM TEAM (team_number, new_team)
CHANGE TEAM(new_team)
... image indices & collectives refer to new team ...
END TEAM
```

- All operations in CHANGE TEAM construct refer to new team
- CHANGE/END TEAM cause synchronization among images in `new_team`

FORM TEAM

```
use, intrinsic: iso_fortran_env, only: team_type
type(team_type) :: team_variable
...
FORM TEAM (team_number, new_team, STAT=s)
IF (s == STAT_FAILED_IMAGE) ... handle image failure ...
CHANGE TEAM(new_team, STAT=s)
   IF (s == STAT_FAILED_IMAGE) ... handle image failure ...
... image indices & collectives refer to new team ...
END TEAM(STAT=s)
IF (s == STAT_FAILED_IMAGE) ... handle image failure ...
```

Adding FORM TEAM STAT= allows *shrinking* recovery

- Failed images removed from *new_team*
- Image indices in *new_team* processor-dependent

EXAMPLE

EXAMPLE

PARALLEL MONTE CARLO PI

PARALLEL MONTE CARLO PI

```fortran
 call random_init(repeatable=.false., image_distinct=.true.)

do sample = 1, SS
  call random_number(x); call random_number(y)
  if (hypot(x, y) <= 1) n = n + 1
end do

 call co_sum(n, result_image=1)

if (this_image() == 1) write(*,*) 4.0d0*n/SS/NUM_IMAGES()
```

PARALLEL MONTE CARLO PI

```fortran
 call random_init(repeatable=.false., image_distinct=.true.)

do sample = 1, SS
  call random_number(x); call random_number(y)
  if (hypot(x, y) <= 1) n = n + 1
end do

 call co_sum(n, result_image=1)

if (this_image() == 1) write(*,*) 4.0d0*n/SS/NUM_IMAGES()
```

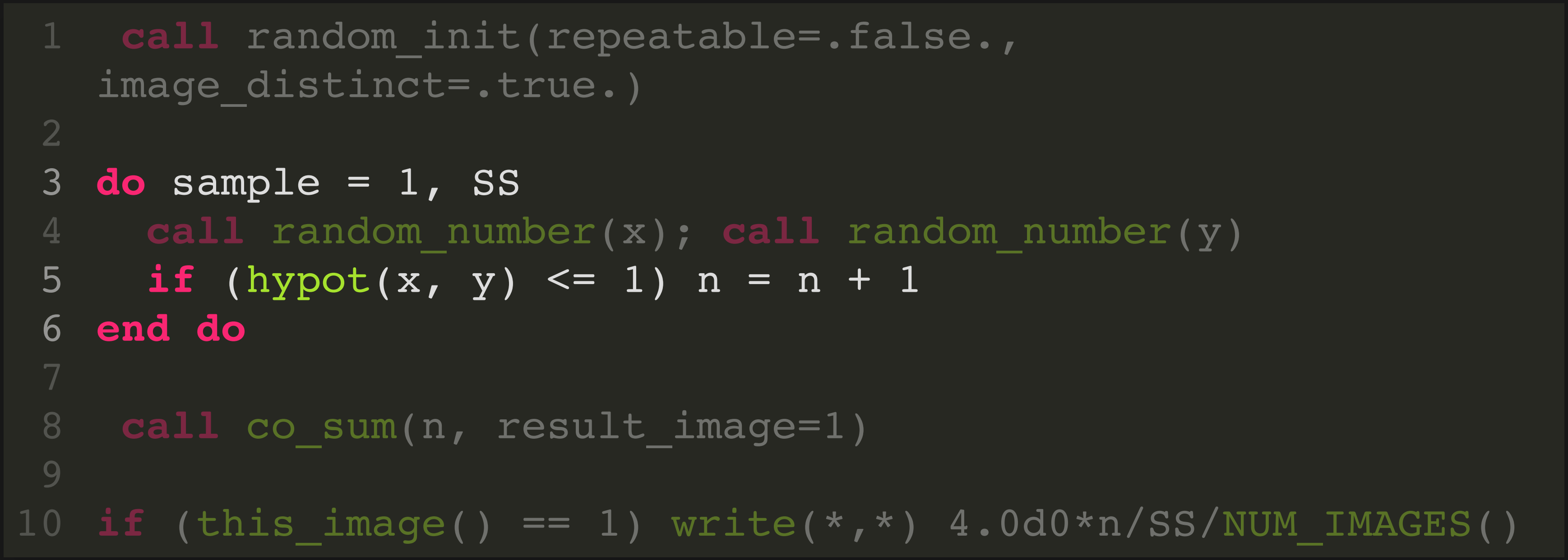
Seed the random number generator on each image.

PARALLEL MONTE CARLO PI

```fortran
 call random_init(repeatable=.false., image_distinct=.true.)

do sample = 1, SS
  call random_number(x); call random_number(y)
  if (hypot(x, y) <= 1) n = n + 1
end do

 call co_sum(n, result_image=1)

if (this_image() == 1) write(*,*) 4.0d0*n/SS/NUM_IMAGES()
```

Randomly sample SS ordered pairs $(x, y) \in [0, 1)$

PARALLEL MONTE CARLO PI



```fortran
 call random_init(repeatable=.false., image_distinct=.true.)

do sample = 1, SS
  call random_number(x); call random_number(y)
  if (hypot(x, y) <= 1) n = n + 1
end do

 call co_sum(n, result_image=1)

if (this_image() == 1) write(*,*) 4.0d0*n/SS/NUM_IMAGES()
```

Count the number that are within the unit circle.

Note $hypot(x, y) == \sqrt{x^2 + y^2}$

PARALLEL MONTE CARLO PI

```fortran
 call random_init(repeatable=.false., image_distinct=.true.)

do sample = 1, SS
  call random_number(x); call random_number(y)
  if (hypot(x, y) <= 1) n = n + 1
end do

 call co_sum(n, result_image=1)

if (this_image() == 1) write(*,*) 4.0d0*n/SS/NUM_IMAGES()
```

Sum counts from each image.

Save the results on image 1.

PARALLEL MONTE CARLO PI

```fortran
 call random_init(repeatable=.false.,image_distinct=.true.)

do sample = 1, SS
  call random_number(x); call random_number(y)
  if (hypot(x, y) <= 1) n = n + 1
end do

 call co_sum(n, result_image=1)

if (this_image() == 1) write(*,*) 4.0d0*n/SS/NUM_IMAGES()
```

$$\pi \approx \frac{4 \times \frac{n_{\text{sum}}}{SS}}{\text{NUM\_IMAGES()}}$$

PARALLEL MONTE CARLO PI (RESILIENT)

```fortran
 call random_init(repeatable=.false., image_distinct=.true.)

do sample = 1, SS
  call random_number(x); call random_number(y)
  if (hypot(x, y) <= 1) n = n + 1
end do

n_copy = n

do
  form team(1, team_active_images, stat=status)
  change team (team_active_images, stat=status)
    image_in_team = this_image()
    call co_sum(n, result_image=1, stat=status)
```

Forward, *shrinking* recovery from failure of any image[1]

[1] Except `image_in_team == 1` after END TEAM

PARALLEL MONTE CARLO PI (RESILIENT)

```fortran
 call random_init(repeatable=.false., image_distinct=.true.)

do sample = 1, SS
  call random_number(x); call random_number(y)
  if (hypot(x, y) <= 1) n = n + 1
end do

n_copy = n

do
  form team(1, team_active_images, stat=status)
  change team (team_active_images, stat=status)
    image_in_team = this_image()
    call co_sum(n, result_image=1, stat=status)
```

On image failure, restore n from a copy…

PARALLEL MONTE CARLO PI (RESILIENT)

```fortran
 call random_init(repeatable=.false., image_distinct=.true.)

do sample = 1, SS
  call random_number(x); call random_number(y)
  if (hypot(x, y) <= 1) n = n + 1
end do

n_copy = n

do
  form team(1, team_active_images, stat=status)
  change team (team_active_images, stat=status)
    image_in_team = this_image()
    call co_sum(n, result_image=1, stat=status)
```

…form a new team (excluding failed images)…

PARALLEL MONTE CARLO PI (RESILIENT)

```fortran
 call random_init(repeatable=.false., image_distinct=.true.)

do sample = 1, SS
  call random_number(x); call random_number(y)
  if (hypot(x, y) <= 1) n = n + 1
end do

n_copy = n

do
  form team(1, team_active_images, stat=status)
  change team (team_active_images, stat=status)
    image_in_team = this_image()
    call co_sum(n, result_image=1, stat=status)
```

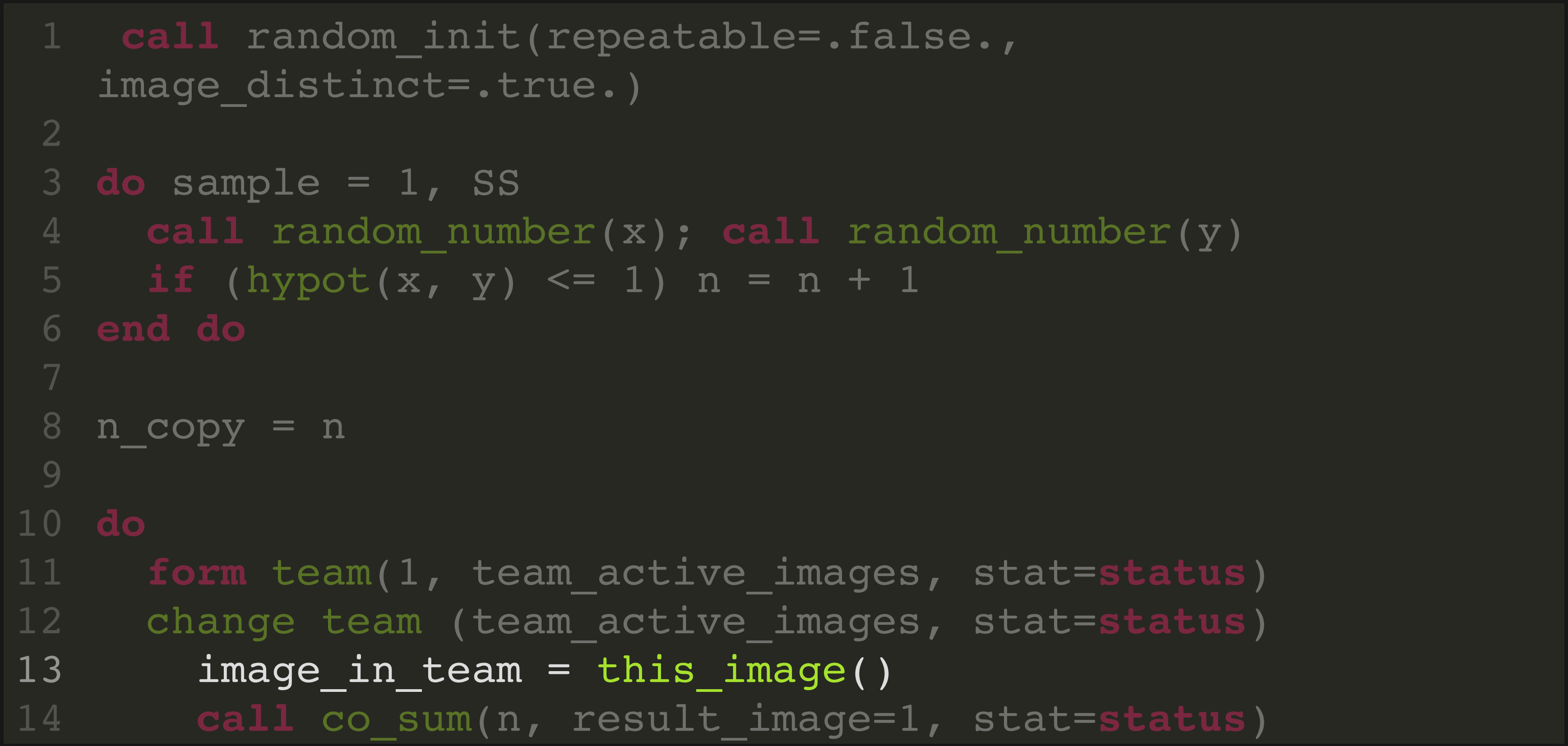
Use the new team for `CO_SUM`

PARALLEL MONTE CARLO PI (RESILIENT)

```
 call random_init(repeatable=.false., image_distinct=.true.)

do sample = 1, SS
  call random_number(x); call random_number(y)
  if (hypot(x, y) <= 1) n = n + 1
end do

n_copy = n

do
  form team(1, team_active_images, stat=status)
  change team (team_active_images, stat=status)
    image_in_team = this_image()
    call co_sum(n, result_image=1, stat=status)
```

If no (further) image failure is detected after CO_SUM and END TEAM, then EXIT the DO loop…

PARALLEL MONTE CARLO PI (RESILIENT)



```fortran
 call random_init(repeatable=.false., image_distinct=.true.)

do sample = 1, SS
  call random_number(x); call random_number(y)
  if (hypot(x, y) <= 1) n = n + 1
end do

n_copy = n

do
  form team(1, team_active_images, stat=status)
  change team (team_active_images, stat=status)
    image_in_team = this_image()
    call co_sum(n, result_image=1, stat=status)
```

Output result (adjusting by # of active images) from image that was image 1 *in team_active_images*[1].

[1] Fortran disallows THIS_IMAGE(TEAM=*child*), NUM_IMAGES(TEAM=*child*)

FORM TEAM

```
MPI_Comm team = current_team;
redo:
rc = MPI_Comm_split(team, team_number, 0, new_team);
flag = (rc == MPI_SUCCESS);
rc = MPIX_Comm_agree(team, &flag);
if (MPI_SUCCESS != rc || !flag) {
  *stat = STAT_FAILED_IMAGE;
  MPIX_Comm_shrink(current_team, new_team);
  MPIX_Comm_failure_ack(current_team);
  MPIX_Comm_failure_get_acked(current_team, &failed);
  ... union with group of known failed images ...
  team = *new_team;
  goto redo;
}
```

OpenCoarrays-ft implementation (approximate)[1]

[1] Based on original OpenCoarrays MPI error handler.

FORM TEAM

```c
MPI_Comm team = current_team;
redo:
rc = MPI_Comm_split(team, team_number, 0, new_team);
flag = (rc == MPI_SUCCESS);
rc = MPIX_Comm_agree(team, &flag);
if (MPI_SUCCESS != rc || !flag) {
  *stat = STAT_FAILED_IMAGE;
  MPIX_Comm_shrink(current_team, new_team);
  MPIX_Comm_failure_ack(current_team);
  MPIX_Comm_failure_get_acked(current_team, &failed);
  ... union with group of known failed images ...
  team = *new_team;
  goto redo;
}
```

Attempt to create an MPI communicator for the new team

FORM TEAM

```
MPI_Comm team = current_team;
redo:
rc = MPI_Comm_split(team, team_number, 0, new_team);
flag = (rc == MPI_SUCCESS);
rc = MPIX_Comm_agree(team, &flag);
if (MPI_SUCCESS != rc || !flag) {
  *stat = STAT_FAILED_IMAGE;
  MPIX_Comm_shrink(current_team, new_team);
  MPIX_Comm_failure_ack(current_team);
  MPIX_Comm_failure_get_acked(current_team, &failed);
  ... union with group of known failed images ...
  team = *new_team;
  goto redo;
}
```

Fault-tolerant consensus on success of split operation (bitwise-AND among non-failed processes)

FORM TEAM

```
MPI_Comm team = current_team;
redo:
rc = MPI_Comm_split(team, team_number, 0, new_team);
flag = (rc == MPI_SUCCESS);
rc = MPIX_Comm_agree(team, &flag);
if (MPI_SUCCESS != rc || !flag) {
  *stat = STAT_FAILED_IMAGE;
  MPIX_Comm_shrink(current_team, new_team);
  MPIX_Comm_failure_ack(current_team);
  MPIX_Comm_failure_get_acked(current_team, &failed);
  ... union with group of known failed images ...
  team = *new_team;
  goto redo;
}
```

If MPI_COMM_SPLIT failed at any process (due to locally-detected process failure), or any (locally-unacknowledged) process failure is detected during MPI_COMM_AGREE…

FORM TEAM

```
MPI_Comm team = current_team;
redo:
rc = MPI_Comm_split(team, team_number, 0, new_team);
flag = (rc == MPI_SUCCESS);
rc = MPIX_Comm_agree(team, &flag);
if (MPI_SUCCESS != rc || !flag) {
  *stat = STAT_FAILED_IMAGE;
  MPIX_Comm_shrink(current_team, new_team);
  MPIX_Comm_failure_ack(current_team);
  MPIX_Comm_failure_get_acked(current_team, &failed);
  ... union with group of known failed images ...
  team = *new_team;
  goto redo;
}
```

MPI_COMM_SHRINK (ULFM): create new comm w/o failed processes

FORM TEAM

```
MPI_Comm team = current_team;
redo:
rc = MPI_Comm_split(team, team_number, 0, new_team);
flag = (rc == MPI_SUCCESS);
rc = MPIX_Comm_agree(team, &flag);
if (MPI_SUCCESS != rc || !flag) {
  *stat = STAT_FAILED_IMAGE;
  MPIX_Comm_shrink(current_team, new_team);
  MPIX_Comm_failure_ack(current_team);
  MPIX_Comm_failure_get_acked(current_team, &failed);
  ... union with group of known failed images ...
  team = *new_team;
  goto redo;
}
```

Acknowledge failed processes in *current_team*

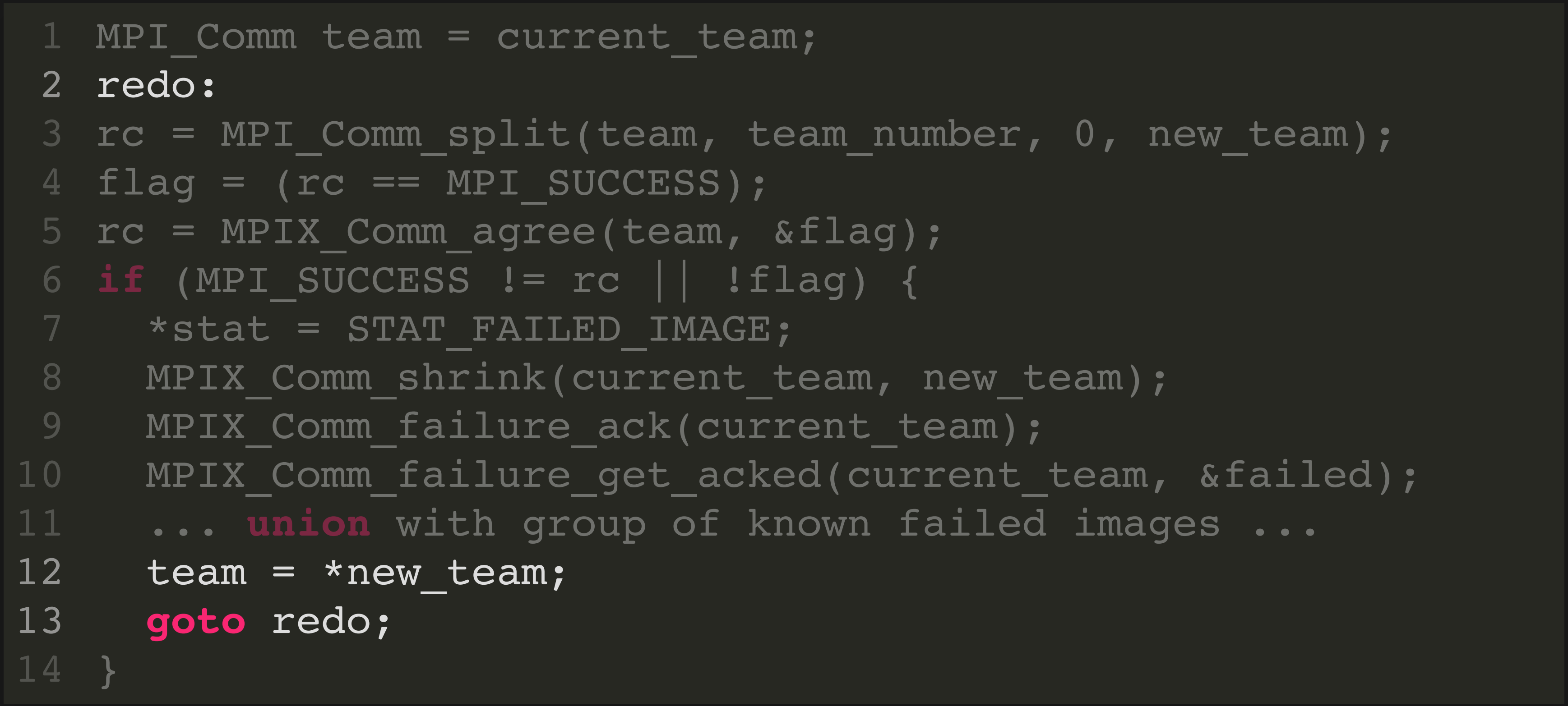
Allows subsequent use without MPIX_ERR_PROC_FAILED

FORM TEAM

```
MPI_Comm team = current_team;
redo:
rc = MPI_Comm_split(team, team_number, 0, new_team);
flag = (rc == MPI_SUCCESS);
rc = MPIX_Comm_agree(team, &flag);
if (MPI_SUCCESS != rc || !flag) {
  *stat = STAT_FAILED_IMAGE;
  MPIX_Comm_shrink(current_team, new_team);
  MPIX_Comm_failure_ack(current_team);
  MPIX_Comm_failure_get_acked(current_team, &failed);
  ... union with group of known failed images ...
  team = *new_team;
  goto redo;
}
```

Get group of failed processes in *current_team*

FORM TEAM



```
MPI_Comm team = current_team;
redo:
rc = MPI_Comm_split(team, team_number, 0, new_team);
flag = (rc == MPI_SUCCESS);
rc = MPIX_Comm_agree(team, &flag);
if (MPI_SUCCESS != rc || !flag) {
  *stat = STAT_FAILED_IMAGE;
  MPIX_Comm_shrink(current_team, new_team);
  MPIX_Comm_failure_ack(current_team);
  MPIX_Comm_failure_get_acked(current_team, &failed);
  ... union with group of known failed images ...
  team = *new_team;
  goto redo;
}
```

Retry, creating a new team from the current team – failed images

FORM TEAM

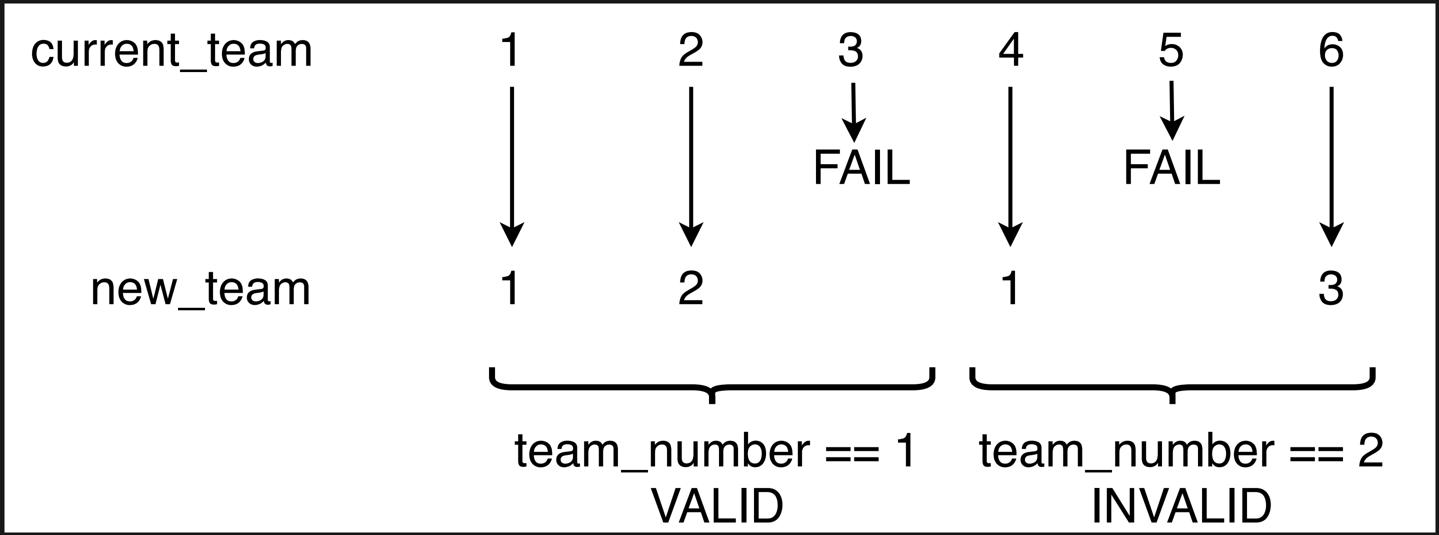
Non-shrinking recovery

```
team_num = 1
if (this_image >= 100) team_num = 2 ! spare image
...
FORM TEAM(team_num, new_team, NEW_INDEX=new_team_idx, STAT=s)
```

- Initial team contains “spare” images
- Image index ordering preserved with NEW_INDEX=
  - Added to OpenCoarrays-ft (and custom GFortran)
- Example C.6.8 (Fortran 2018 standard)
  - Issues; See paper for enhanced version

FORM TEAM NUM_IMAGES=

Fortran 2018 semantics



current_team 1 2 3 4 5 6

FAIL FAIL

new_team 1 2 1 3

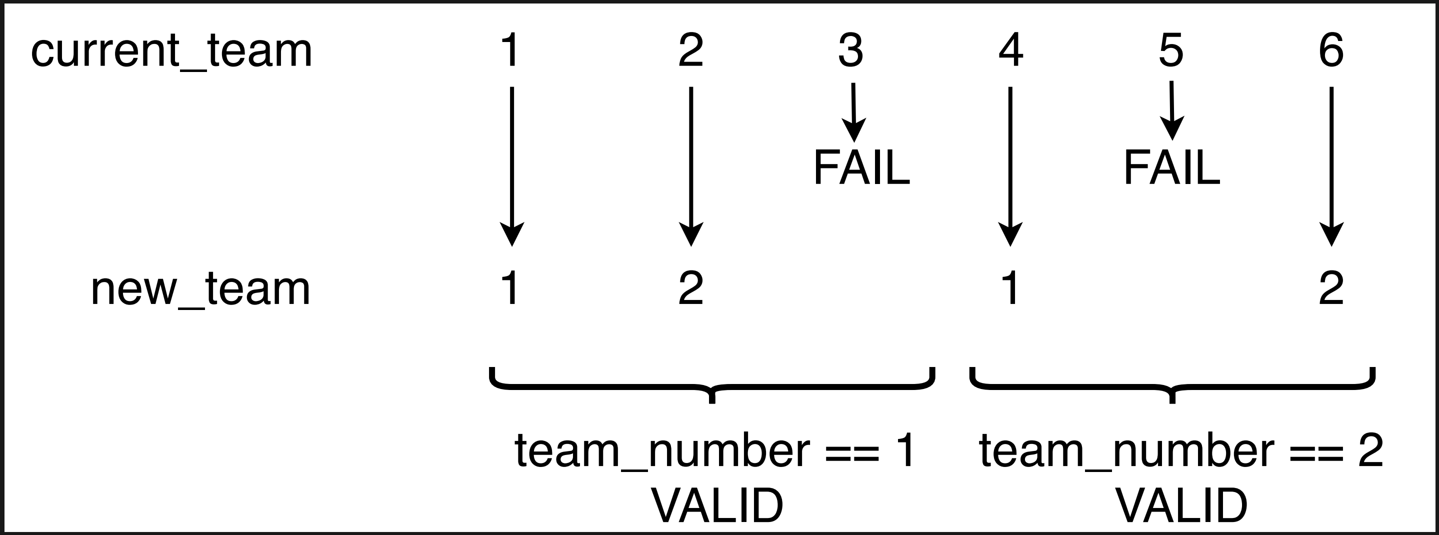
team_number == 1
VALID

team_number == 2
INVALID

```
FORM TEAM((THIS_IMAGE()-1)/(NUM_IMAGES()/2)+1), &
          new_team, &
          NEW_INDEX=MOD(THIS_IMAGE()-1,NUM_IMAGES()/2)+1, &
          STAT=stat)
```

FORM TEAM NUM_IMAGES=

Proposed semantics



```
! Similar to MPI_COMM_SPLIT, with non-consecutive key values
 call MPI_Comm_split(comm    = current_team,&
                     color   = team_number, &
                     key     = new_index,   &
                     newcomm = new_team,    &
                     ierror  = stat)
```

BENCHMARKS

SOFTWARE ENVIRONMENT

- OpenCoarrays-ft
- ULFM2
- GFortran 9.3.0 (modified)
  - STAT= and ERRMSG= in additional image control statements
  - FORM TEAM NEW_INDEX= specifier

SOFTWARE ENVIRONMENT

Available as a software container image (Alpine Linux)

Docker

```
$ alias dcaf='docker run -it --rm -v $PWD:/mnt -w $PWD:/mnt
ghcr.io/nathanweeks/espm2-2020:latest'
$ dcaf caf prog.f90
$ dcaf cafrun -np 8 ./a.out
```

Singularity

```
$ singularity pull docker://ghcr.io/nathanweeks/espm2-
2020:latest
$ singularity exec espm2-2020_latest.sif caf prog.f90
$ singularity exec espm2-2020_latest.sif cafrun -np 8 ./a.out
```

Caveat: expect bugs!

SOFTWARE ENVIRONMENT

- NERSC Cori (KNL)
- Shifter
  - TCP BTL

TEAM SYNCHRONIZATION

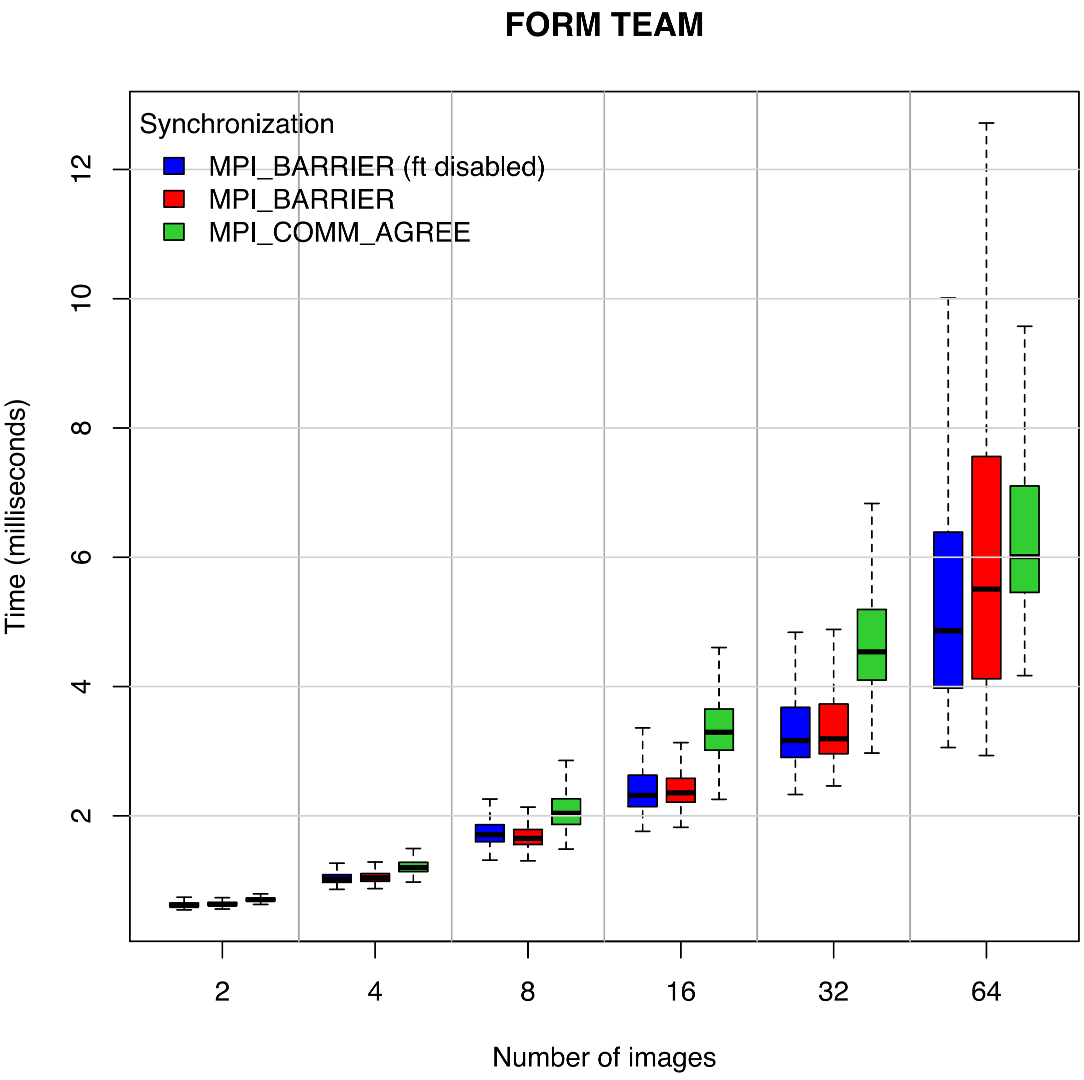
MPI_BARRIER-based synchronization

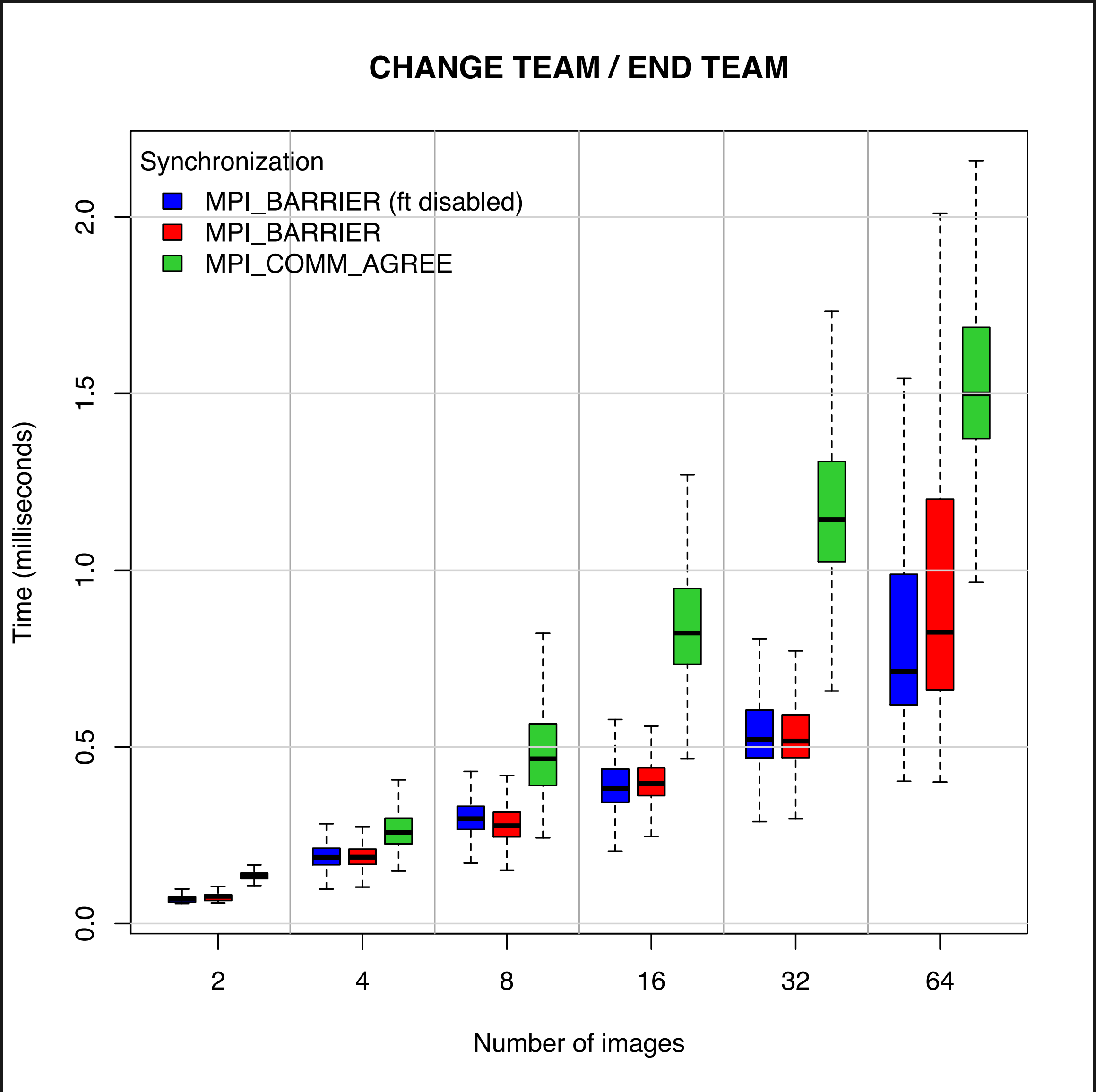
```fortran
do i = 1, 1000
  call system_clock(...)
  form team(1, active_images)
  call system_clock(...)
  change team(active_images)
    call system_clock(...)
  end team
end do
call system_clock(...)
```

TEAM SYNCHRONIZATION

Fault-tolerant synchronization (MPI_COMM_AGREE)

```fortran
do i = 1, 1000
  call system_clock(...)
  form team(1, active_images, stat=status)
  call system_clock(...)
  change team(active_images, stat=status)
    call system_clock(...)
  end team(stat=status)
end do
call system_clock(...)
```


FORM TEAM
Synchronization
🟦 MPI_BARRIER (ft disabled)
🟥 MPI_BARRIER
🟩 MPI_COMM_AGREE
Time (milliseconds)
2
4
6
8
10
12
2
4
8
16
32
64
Number of images

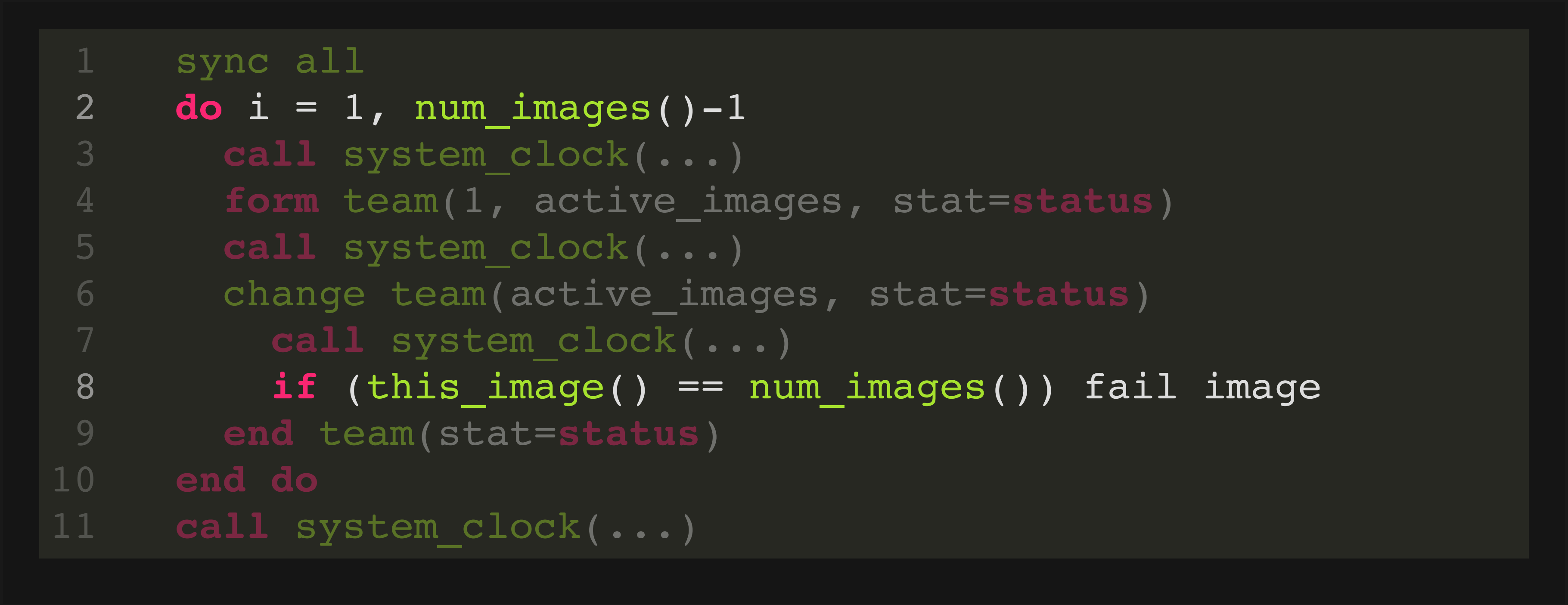

CHANGE TEAM / END TEAM
Synchronization
MPI_BARRIER (ft disabled)
MPI_BARRIER
MPI_COMM_AGREE
Time (milliseconds)
0.0
0.5
1.0
1.5
2.0
2
4
8
16
32
64
Number of images

SOLE SURVIVOR

```fortran
sync all
do i = 1, num_images()-1
  call system_clock(...)
  form team(1, active_images, stat=status)
  call system_clock(...)
  change team(active_images, stat=status)
    call system_clock(...)
    if (this_image() == num_images()) fail image
  end team(stat=status)
end do
call system_clock(...)
```

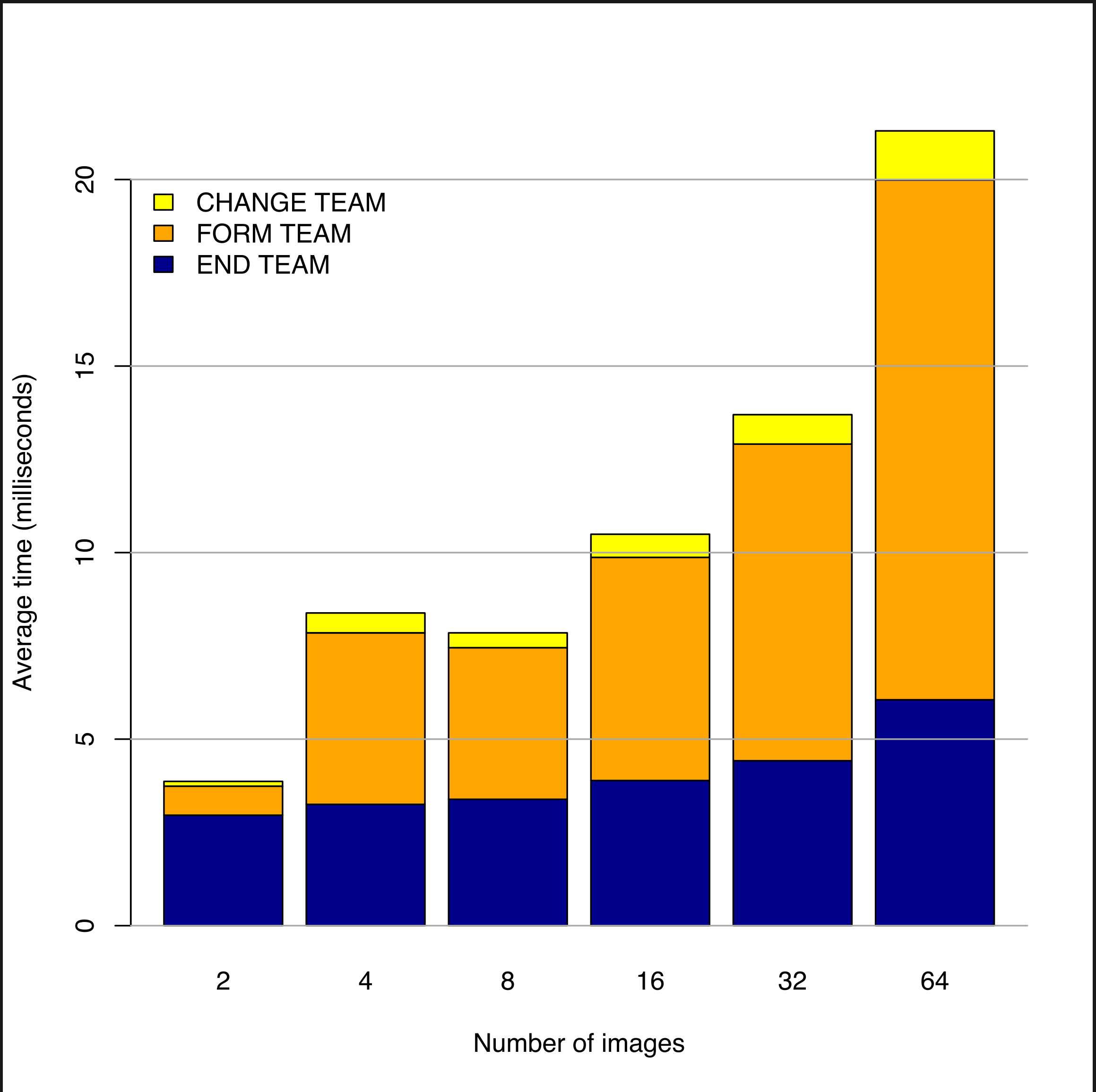
Time FORM / CHANGE / END team in the presence of image failures.

SOLE SURVIVOR



```
sync all
do i = 1, num_images()-1
  call system_clock(...)
  form team(1, active_images, stat=status)
  call system_clock(...)
  change team(active_images, stat=status)
    call system_clock(...)
    if (this_image() == num_images()) fail image
  end team(stat=status)
end do
call system_clock(...)
```

One image fails per iteration, until only one remains.


CHANGE TEAM
FORM TEAM
END TEAM
Average time (milliseconds)
0
5
10
15
20
2
4
8
16
32
64
Number of images

SUMMARY

- Fortran 2018 defines abstractions for fault tolerance
- Prototype Fortran failed images + teams
- Debug/validate Fortran 2018
  - Fortran standard changes needed to facilitate *portable* resilient applications

ACKNOWLEDGEMENT

This research used resources of the National Energy Research Scientific Computing Center (NERSC), a U.S. Department of Energy Office of Science User Facility operated under Contract No. DE-AC02-05CH11231, ROR ID https://ror.org/05v3mvq14.

QUESTIONS?