

# MPI Forum Fault Tolerance Working Group Roadmap

September, 2021

Fault Tolerance Working Group

# Error Handling and Resilience Available in MPI 4.0

- New MPI Error Handler - MPI\_ERRORS\_ABORT
- Localize error impact of some MPI operations.
  - MPI\_ALLOC\_MEM will now raise an error on MPI\_COMM\_SELF, not MPI\_COMM\_WORLD
- Specify that MPI should avoid fatal errors when the user doesn't use MPI\_ERRORS\_ARE\_FATAL.
- Specify that MPI\_SUCCESS indicates only the result of the operation, not the state of the MPI library.
- Allow the user to specify the default error handler at mpiexec time.
- Add MPI\_ERR\_PROC\_ABORTED.

# What Can You Do With MPI 4.0?

- Point to Point communication with sockets-like error handling
  - When you see an error of certain classes (usually `MPI_ERR_PROC_ABORTED` or `MPI_ERR_OTHER`), you can probably assume that the process is gone.
- Enables manager/worker and other non-traditional types of applications
  - Enterprise applications that want to move from sockets to MPI can do so.

# What's Next?

- Working on two main proposals:
  - Fine-grained recovery - Led by Aurelien
    - Pieces from ULFM, but composable with more models
  - Coarse-grained recovery – Led by Ignacio
    - Known in research as Reinit
- These two things are made up of smaller pieces where some are being done independently and some as a group.

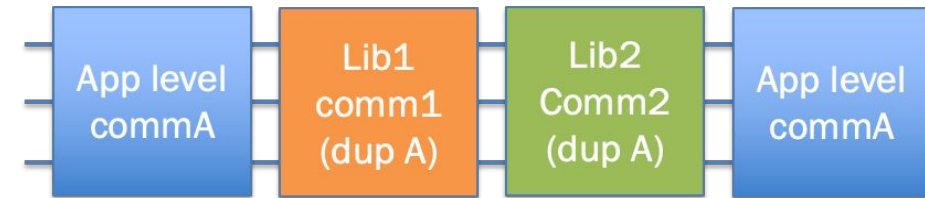
# Composability

- Show how multiple models can exist in an implementation (regardless of how many are used in an app at once).
  - The also applies to how they can exist in the MPI Standard at once.
- How can multiple models exist in the same application?
  - Where are the cases where we do not want to allow multiple models to co-exist (e.g., ULFM + Reinit might not make sense in all cases)
  - Aurelien and Tony have papers related to these topics.

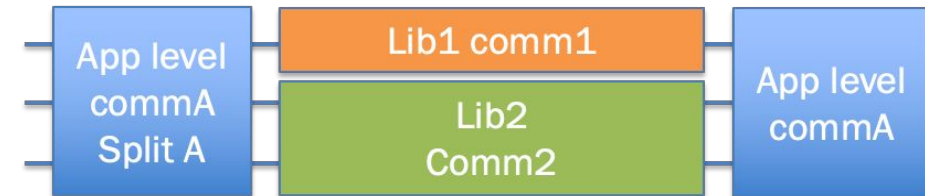
# Levels of Composability

- Level 0 – Models coexist but do not interoperate
- Level 1 – Models can be used in the same application, but not at the same time.
  - E.g., Use fine-grained recovery, then coarse-grained, then fine-grained again
- Level 2 – Models used in the same application, but not all processes are using the same models
  - E.g., One communicator uses coarse-grained recovery, another uses fine-grained
- Level 3 – Models are fully integrated and can be used interchangeably.

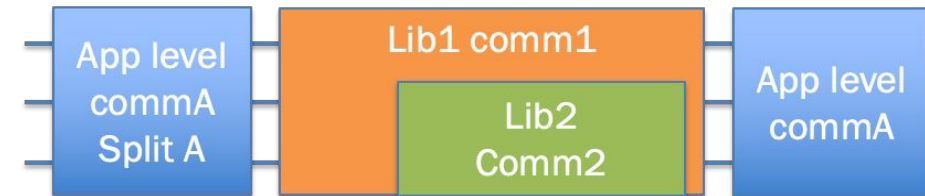
Temporal SPMD library composition



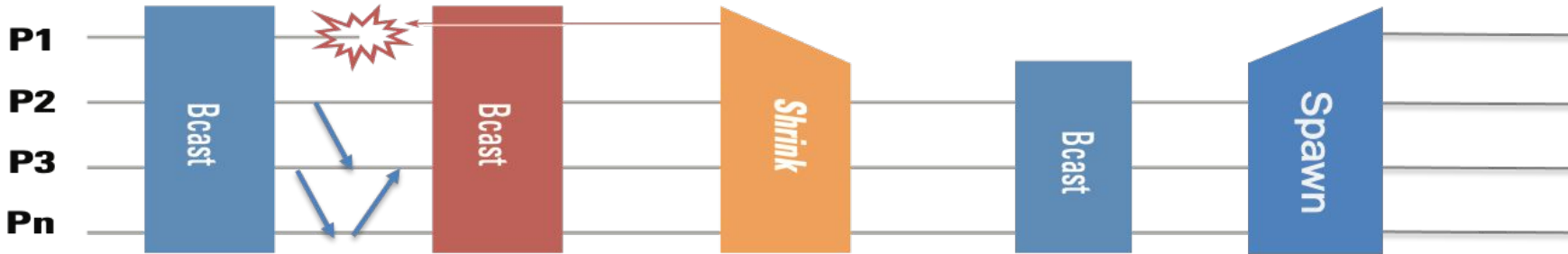
Domain SPMD decomposition



Nesting (w/o overlap)



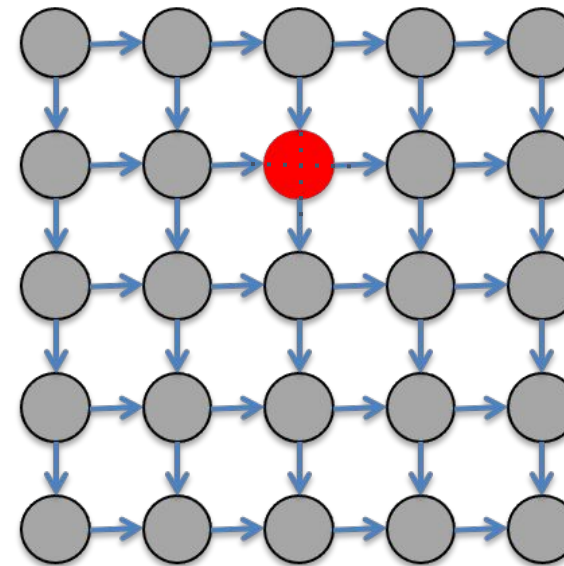
# ULFM MPI **Crash** Recovery (Background)



- Some applications can continue w/o recovery
- Some applications are malleable
  - Shrink creates a new, smaller communicator on which collectives work
- Some applications are *not* malleable
  - Spawn can recreate a “same size” communicator
  - It is easy to reorder the ranks according to the original ordering
  - Pre-made code snippets available

- **Failure Notification**
- **Error Propagation**
- **Error Recovery**
- Respawn of nodes
- Dataset restoration

*Not all recovery strategies require all of these features, that's why the interface should split notification, propagation and recovery.*



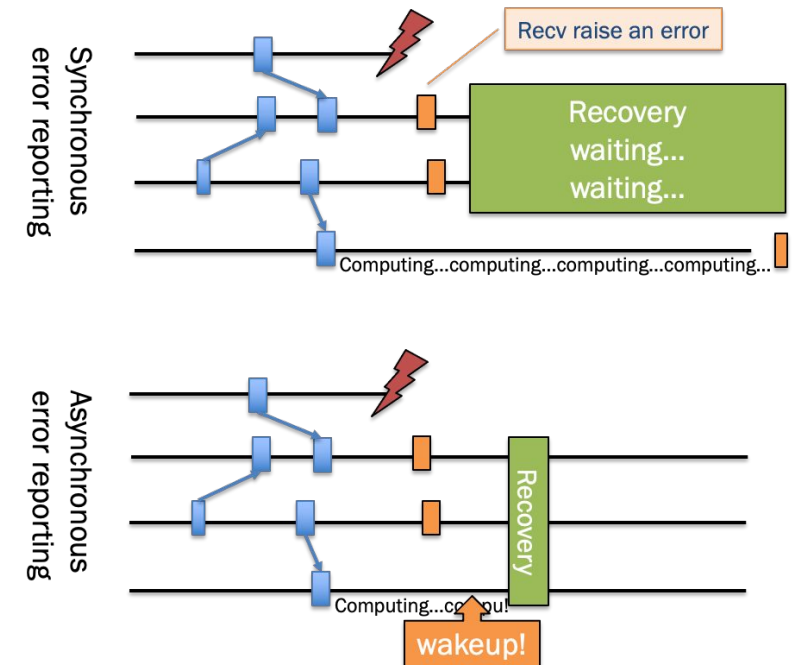
Who should be **notified** of a failure?  
What is the **scope** of a failure?  
What **actions** should be taken?

- Adds 3 error codes and 5 functions to manage process crash
  - **Error codes:** interrupt operations that may block due to process crash
  - **MPI\_COMM\_FAILURE\_ACK / GET\_ACKED:** continued operation with ANY-SOURCE RECV and observation known failures
  - **MPI\_COMM\_REVOKE** lets applications interrupt operations on a communicator
  - **MPI\_COMM\_AGREE:** synchronize failure knowledge in the application
  - **MPI\_COMM\_SHRINK:** create a communicator excluding failed processes
- More info on the MPI Forum ticket #20:  
<https://github.com/mpi-forum/mpi-issues/issues/20>

# Interrupting Error Handlers

- Allow error handlers to specify whether they should only be triggered by an MPI function or whether they should be able to interrupt any time.
- Enables both fine and coarse-grained recovery

Work in progress, not in a current proposal





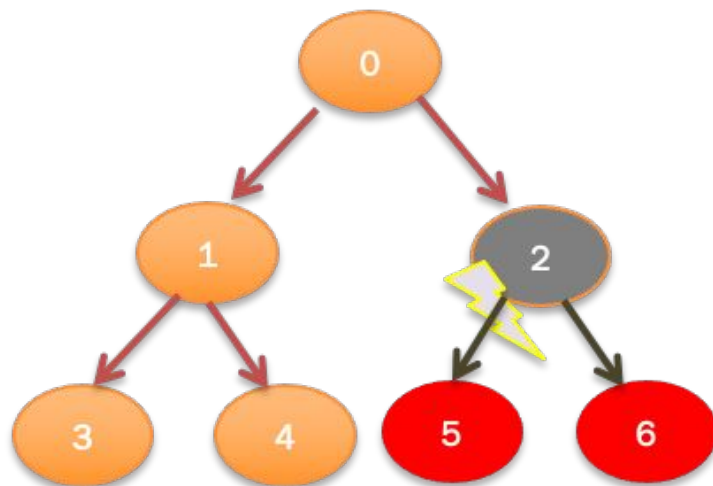
# Discover Failed Processes

- Provide a way for the user to query the list of failed processes
- Similar to the previously proposed functions `MPI_COMM_FAILURE_ACK` / `MPI_COMM_FAILURE_GET_ACKED`
  - Simplified in the common case
  - Remains flexible when necessary
- **`MPI_COMM_GET_FAILED`**(comm, out: failedgrp)
  - Gets the group of failed processes in a simple, natural way
- **`MPI_COMM_ACK_FAILED`**(comm, in: nack, out: nacked)
  - Lets user resume `ANY_SOURCE` w/o unintended side effects, can also be used to count failed procs without creating the failedgrp

New version of these functions has been drafted already

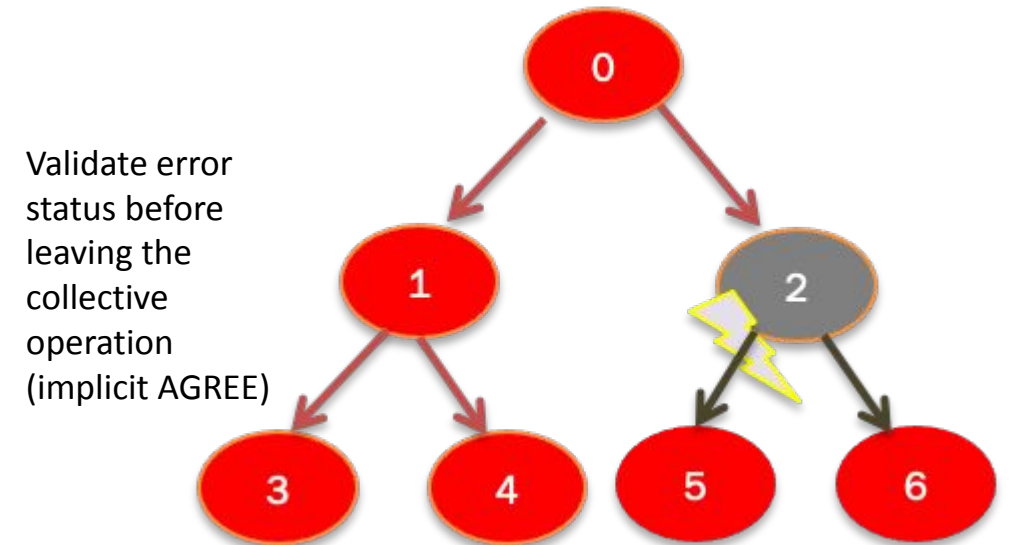
# Uniform State of MPI Collectives

- Allow the user to specify whether collectives should have uniform error codes.
- Has major performance impact, but dramatically simplifies error model.
- Can help enable a uniform error handling model (Reinit or something like it)
- Might want to use info “hints” to tell MPI to turn these on/off



Gray: dead  
Red: returned MPI\_PROC\_FAILED  
Orange: returned MPI\_SUCCESS

Left: Collective operation returns non-uniform errors

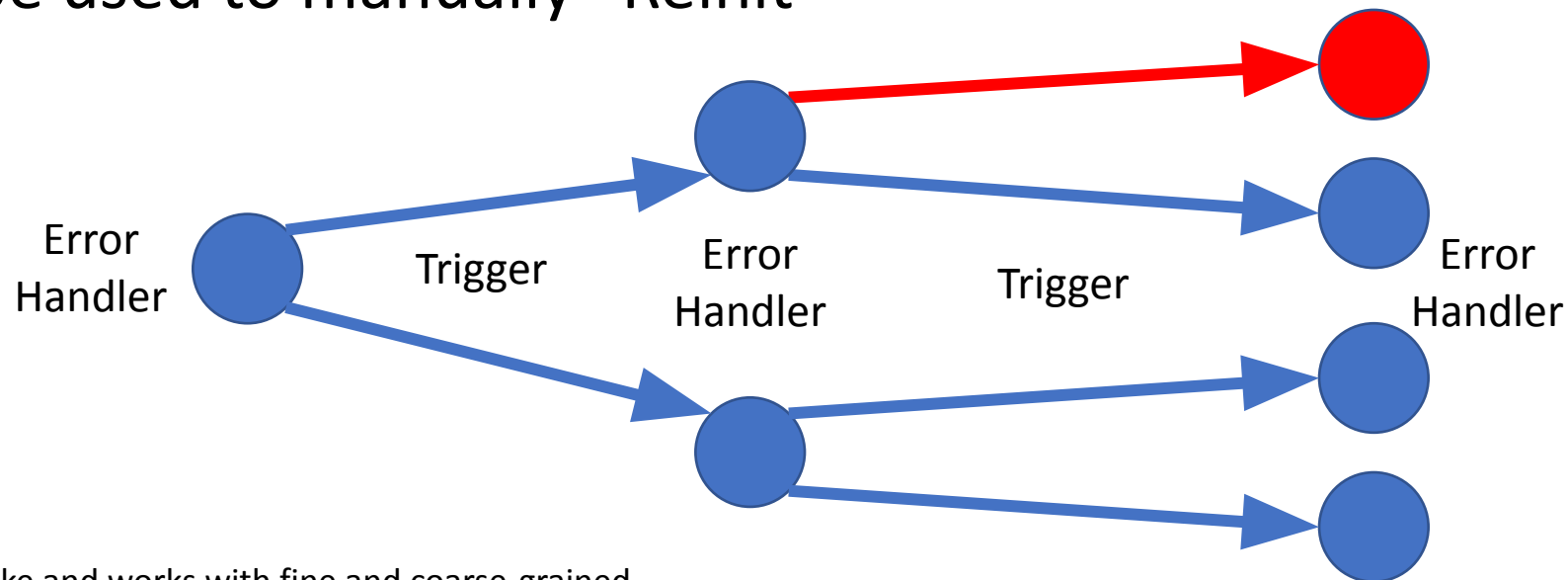


Validate error status before leaving the collective operation (implicit AGREE)

Right: Collective operation self validates and returns an uniform error

# Resilient Broadcast Which Triggers Error Handling

- Explicit error propagation when the implicit error propagation is not used (from the uniform collectives and interrupting error handlers).
- Same functionality as previously presented MPI\_COMM\_REVOKE
- Could be used to manually “Reinit”

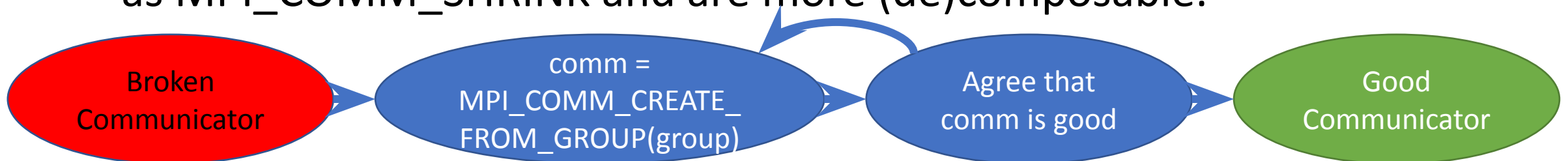


Accomplished with revoke and works with fine and coarse-grained recovery (Level 3)

# MPI\_COMM\_CREATE\_FROM\_GROUP and Agreement Protocol

Working on text for new version of shrink to be more composable

- MPI\_COMM\_CREATE\_FROM\_GROUP, SESSION\_CREATE\_FROM\_GROUP already exists, but lack required validation in faulty scenarios, which is important for constructing a new communicator after a failure.
- The agreement protocol will operate on a group, rather than a communicator, so it can be used to validate functions like communicator creation.
- These functions complement (replace?) what was previously known as MPI\_COMM\_SHRINK and are more (de)composable.



# Reinitialize MPI

- Cleans up MPI state and jumps to a specified point in the code
- Used in combination with:
  - The interrupting error handlers and uniform error reporting
- This constructs a global roll-back error recovery

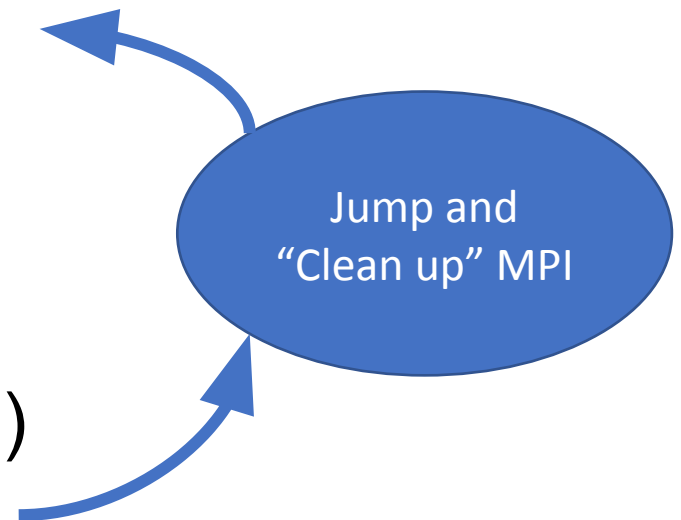
Formal text has been drafted and is getting close to a plenary:  
<https://reinit.github.io/reinit/>

```
MPI_Init()  
...initialize...
```

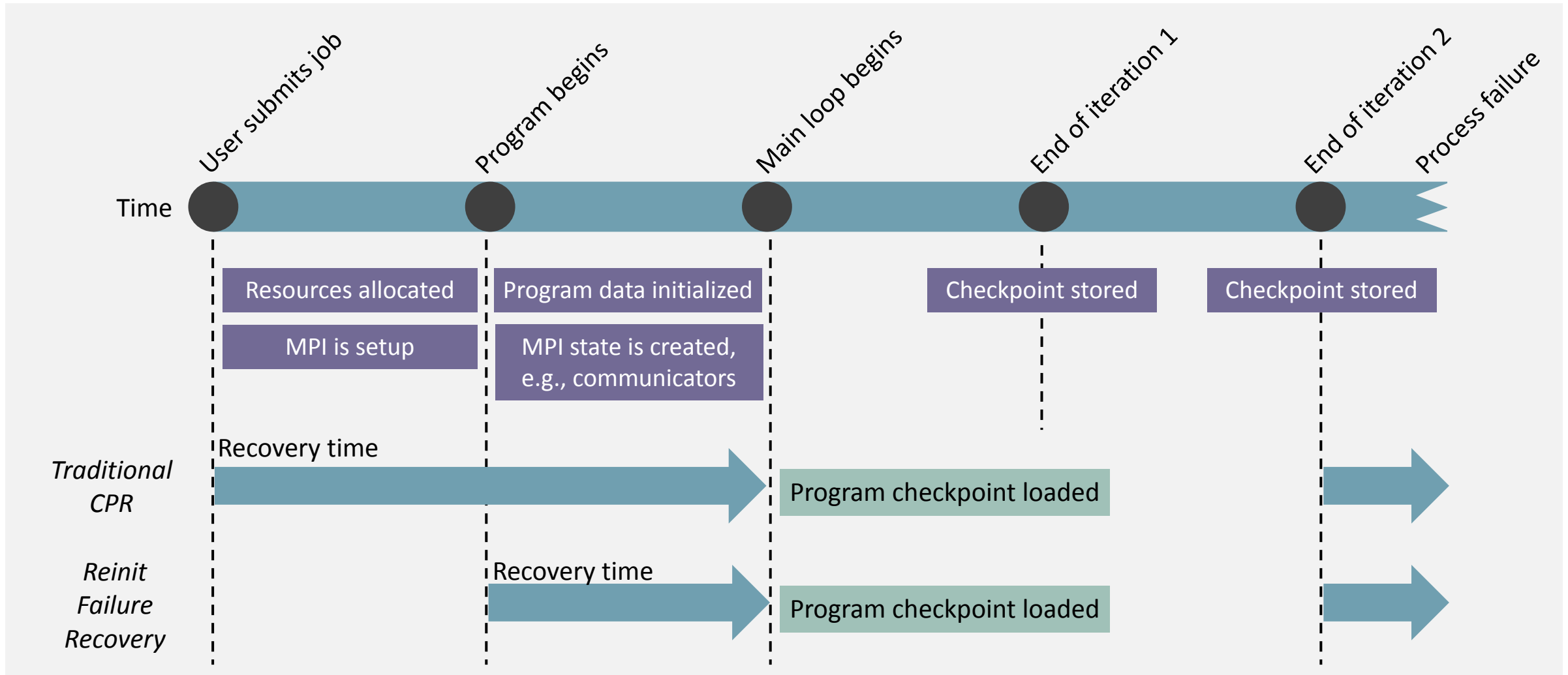
```
MPI_Reinit()
```

```
...do things...
```

```
MPI_Allreduce()  
/* ERROR */
```



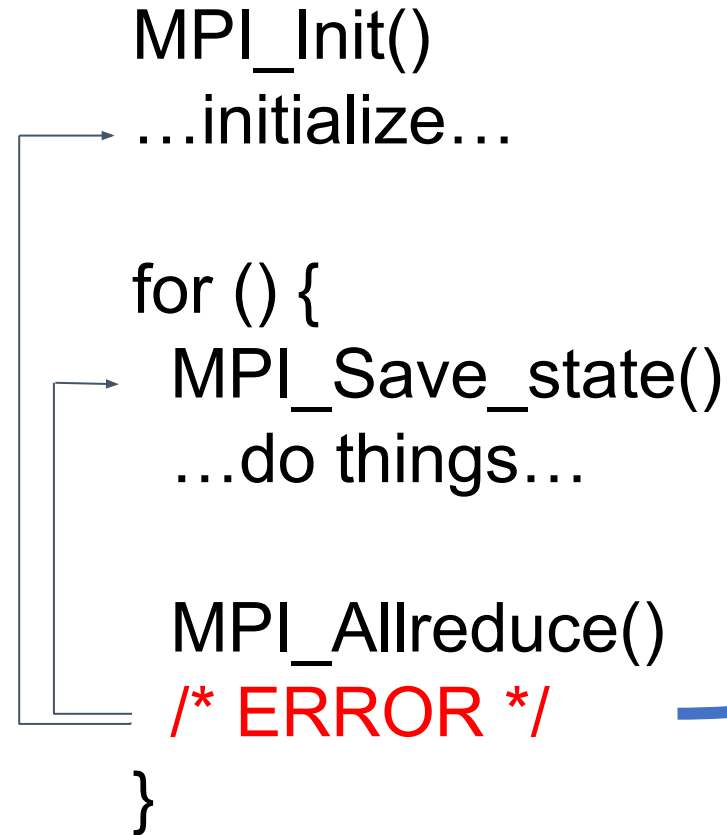
# Coarse-grained Recovery (Reinit)



# Checkpoint MPI State & Return to Previous State X

- More generic case of reinitializing MPI by allowing multiple reinitialization points
- Still in early discussions
- Likely to not be its own model, but will be a “building block” that can be used independently (level 3)

```
MPI_Init()  
...initialize...  
  
for () {  
    MPI_Save_state()  
    ...do things...  
  
    MPI_Allreduce()  
    /* ERROR */  
}
```



Jump and  
“Clean up” MPI

# RMA Story

- Fine-Grained Recovery:
  - In the event of a failure, the window will become invalid (along with all of the data) and the user will need to recreate it (probably from a checkpoint of some kind).
- Coarse-Grained Recovery:
  - In the event of a failure, the MPI state will be restored via rollback, but the user will need to recover their own data (probably also from a checkpoint).



# I/O Story

- Fine-Grained Recovery:
  - Files will return errors like communicators do. The application will be responsible for closing files and opening new ones when necessary.
- Coarse-Grained Recovery:
  - MPI File objects will be destroyed and the user will recreate them during recovery. Drafting text to describe what to do with the MPI File objects.

# Interoperability Status of Various Proposals

Parts of each proposal can be done together or independently.

Many pieces can be used to implement either recovery model

- Collective agreement is useful for lots of algorithms
- Resilient broadcast can implement “reinit”
- Interrupting error handlers can be used for both

Any restrictions will have to be function-specific.

# Timelines for Proposals

- Dec 2021 Meeting:
  - Plan to present Reinit in plenary
  - Stage 1 of ULFM (general statements and error semantic for procedures)
- Mar 2022 Meeting:
  - Stage 2 of ULFM (error trigger/aka revoke)

# Feedback

Generally positive about what was presented

We need tighter coordination with session WG, many common ideas

Discussion about fault models

We exposed how the concepts presented can be used to handle user-reported errors (e.g., soft errors, convergence errors, etc)

We discussed why proc-failed is a better fault model for end-users than exposing transient and/or link fault model (it pushes the hard to deal with part to users that have no clue); our approach in that case is that the impl. is responsible for doing something sane: promote link/transient errors to fail-stop; or do something expensive and complicated under the hood if the hardware features makes the case worth dealing with.