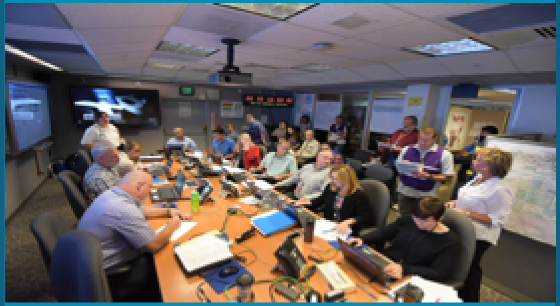
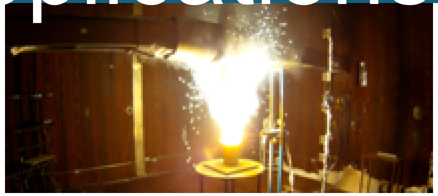
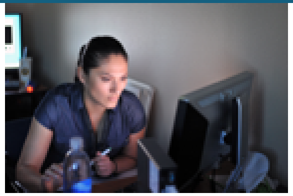
SAND2022-0717C

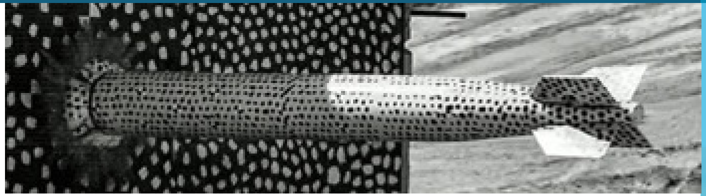
Sandia National Laboratories

Asynchrony and Failure Masking via Pseudo-Local Process Recovery in MPI Stencil Applications









Hemanth Kolla, Jackson Mayo, Matthew Whitlock, Keita Teranishi, Rob Armstrong

Sandia National Laboratories, Livermore, CA.

Sandia National Laboratories is a multimission laboratory managed and operated by National Technology & Engineering Solutions of Sandia, LLC, a wholly owned subsidiary of Honeywell International Inc., for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA0003525.

SIAM PP, Feb 23-26, 2022

Sandia National Laboratories is a multimission laboratory managed and operated by National Technology & Engineering Solutions of Sandia, LLC, a wholly owned subsidiary of Honeywell International Inc., for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA0003525.

Synopsis

In this talk we

- Advocate for HPC fault tolerance, for optimizing performance on current systems and co-design of future heterogeneous systems
- Distinguish between global recovery and local recovery, and why the latter is better
- For hard failures in MPI applications describe the challenges for localizing recovery
- Demonstrate how pseudo-local recovery can be accomplished with existing middleware
- Show results from experiments with MPI stencil 1D app that demonstrate “failure masking”

Motivation

- Fault characteristics of modern HPC systems are not well understood
  - Systems are still studied in retrospect (post-analysis of system error logs)
  - Unexpected/unanticipated failures occur that have system-wide impact (e.g. Titan)
- Extreme heterogeneity (systems and software) makes the fault landscape ever more complex
- Improving fault tolerance of applications, middleware can alleviate constraints
  - Free up operational power budget by allowing for higher fault rates
  - Co-design new systems with inherently higher fault rates, but also higher computational throughput.

Fault-tolerance could be an important knob in the design and operation of HPC

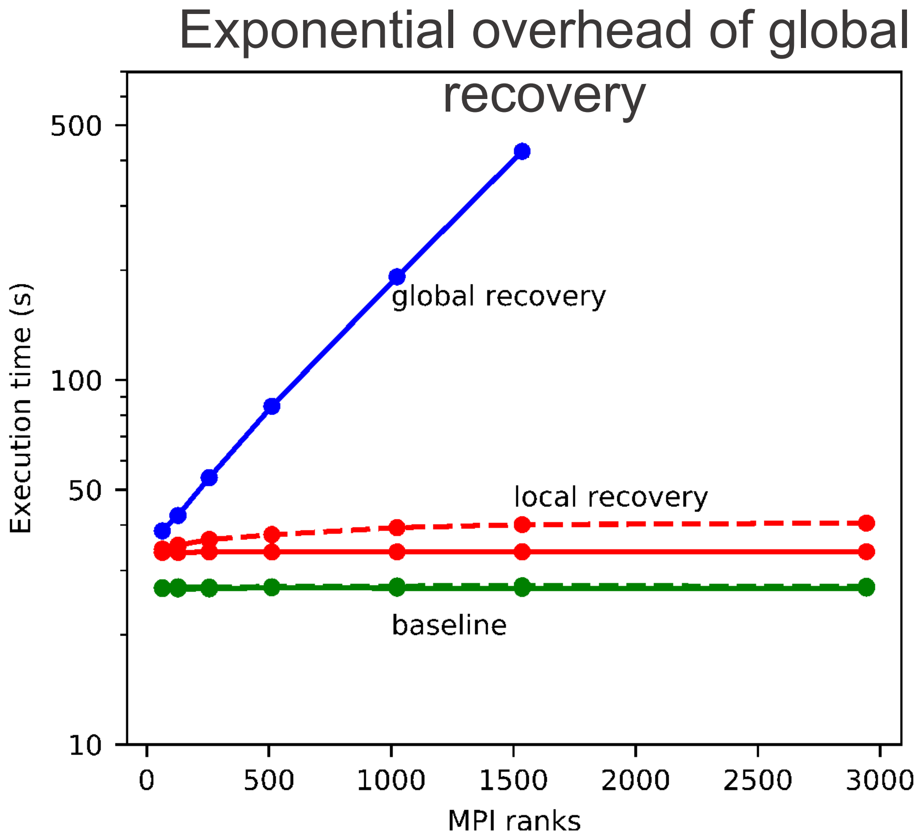
Global vs Local recovery

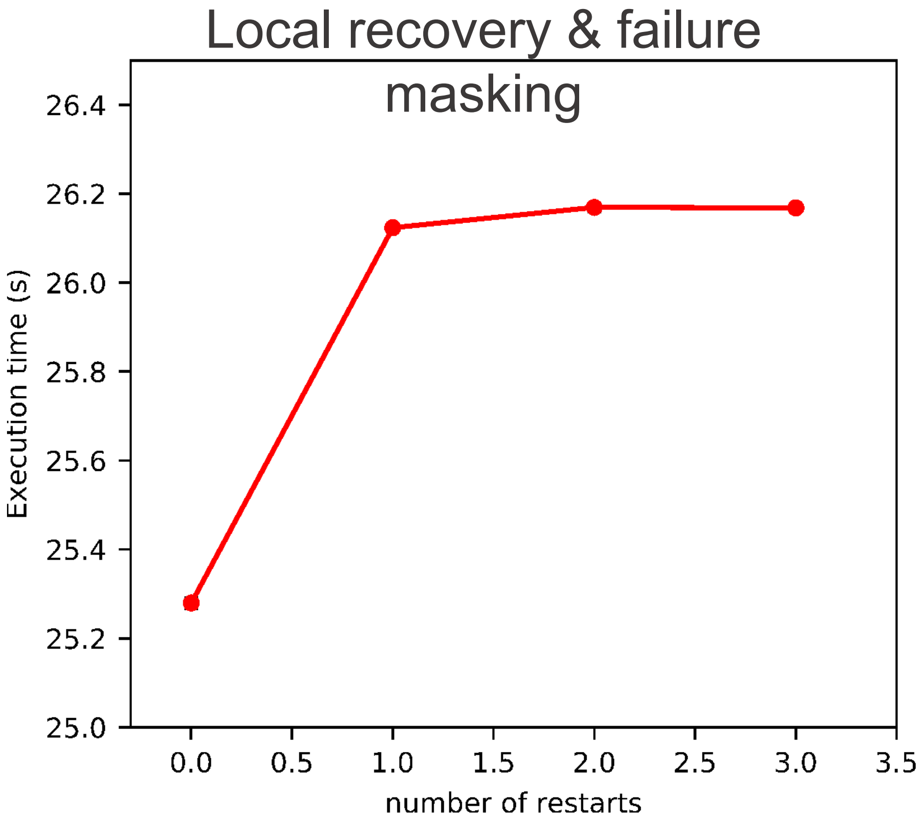
- Global recovery
  - *All* participating processes roll back to last safe execution point and resume
  - Easier to implement, ensure correctness of execution
  - Inherently limited scalability (w.r.t. problem size, fault rates); exponential scaling of recovery overhead
- Local recovery
  - Only process near failure rolls back and resumes; others continue progress
  - Harder to implement, correctness requires care
  - Application may have inherent constraints on extent of localizing recovery
- Multiple studies have demonstrated the potential benefits of local recovery
  - Algorithm-based detection and recovery for silent error (Mayo et al..)
  - Online recovery and failure masking for local recovery from hard failure (Gamell et al.)
  - Scalable recovery in MPI+AMT programming models (Paul et al.)

Global vs Local recovery & failure masking (silent errors)

Kolla *et al.*, "Improving Scalability of Silent-Error Resilience for Message-Passing Solvers via Local Recovery and Asynchrony", FTXS-2020

- Checksums for process-local detection of silent errors
- Recovery from local in-memory checkpoints, no need for process recovery



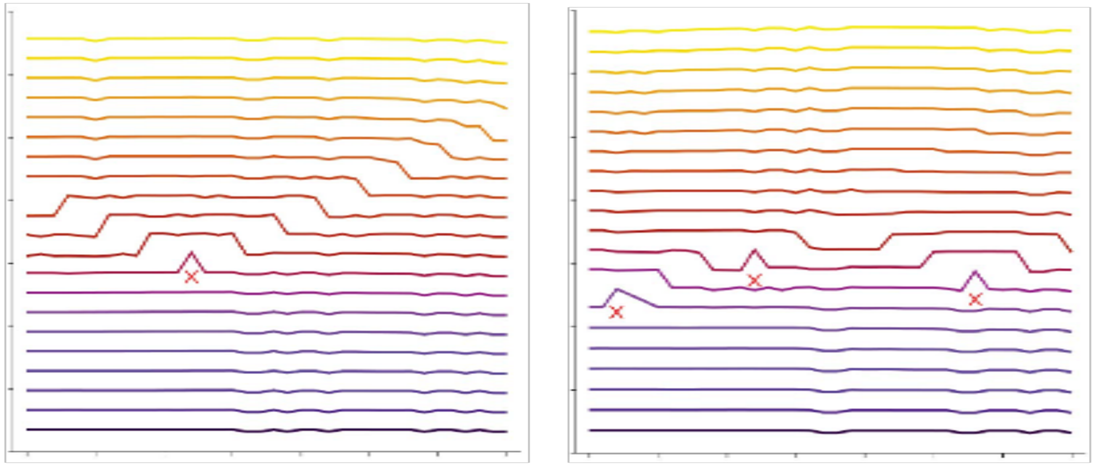


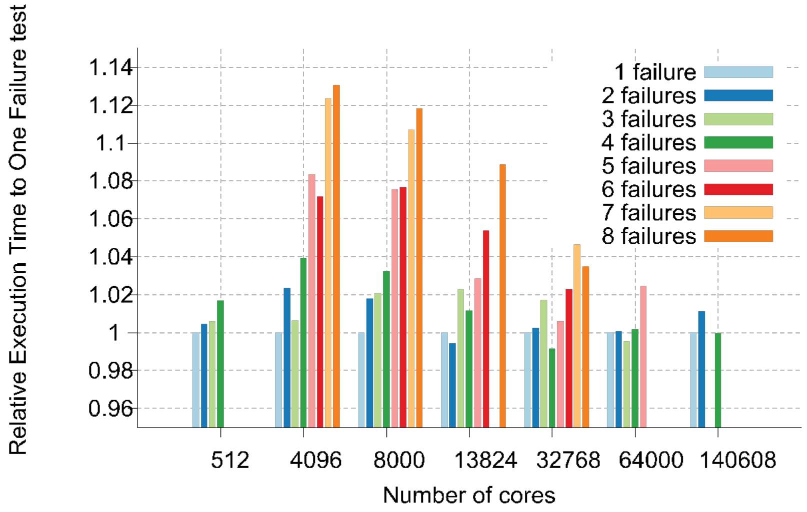
Localizing recovery for hard failures

- Recovery from process/node failures in MPI applications is challenging:
  - Need to recover MPI environment, processes, state, messages
  - Tedious to do entirely at application level, requires middleware support
- User Level Failure Mitigation (ULFM) – spec of resilience features – only recently became mainstream MPI
- Gamell *et al.,* 2015
  - Bypassed MPI (ULFM) altogether, implemented recovery directly at uGNI layer
  - *Data Resiliency* for data checkpoint, recovery message logging; *Process Resiliency* for managing spare processes, replacing failed processes
  - Not compatible with general MPI applications
- Losada *et al.,* 2019, implemented local recovery using three components:
  - ULFM, ComPiler for Portable Checkpointing (CPPC) for checkpointing, process respawn, communicator repair, VProtocol for message logging
  - Recovery is completely transparent to application

Local recovery & failure masking (hard failures)

Gamell *et al.*, "Local Recovery and Failure Masking for Stencil-based Applications at Extreme Scales", SC15



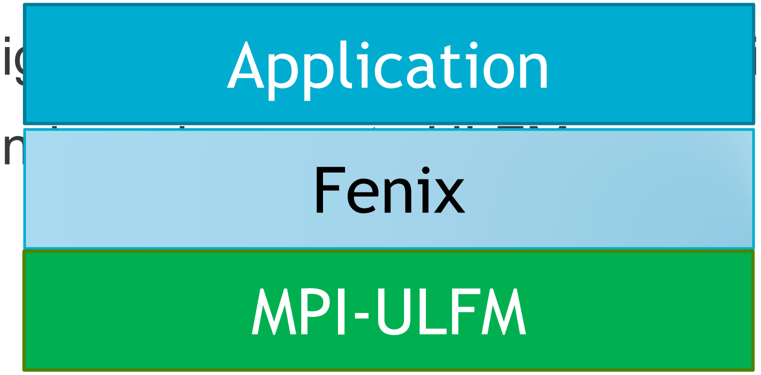


- Recovering locally from a failure introduces a delay
- Delay propagates to other ranks gradually
- Delays from successive failures *mask* each other, i.e. the delay of *N* failures is sub-linear w.r.t. delay of single failure

- Sub-linear scaling of recovery cost
- Recovery cost scales favorably to large number of failures, allowing runs on large number of ranks.

Our approach: pseudo-local recovery

- Objective: demonstrate pseudo-local recovery from hard failures with application-specific refinements
- Use existing Fenix middleware (with a few enhancements) to realize failure masking for MPI stencil code
- Pseudo-local: all processes participate in recovery, but asynchronous progress is preserved
  - In a stencil code neighbours can be ahead by one iteration; distant processes can be ahead by many
- Our desig… ication level. Only a few (straightforward) changes to Fenix, an…



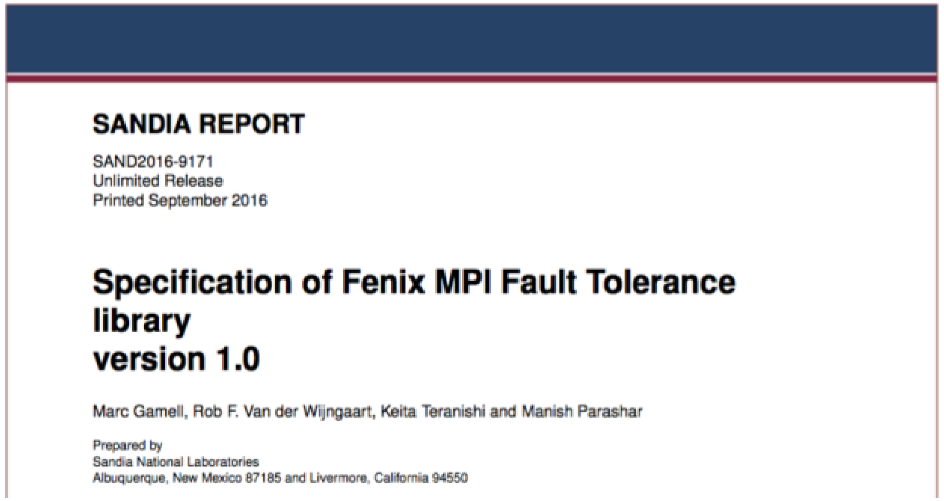
MPI stencil 1D code

A general framework for MPI-based fault tolerance

Fault tolerant MPI layer

Fenix

Fault Tolerant Programming Framework for MPI Applications


SANDIA REPORT

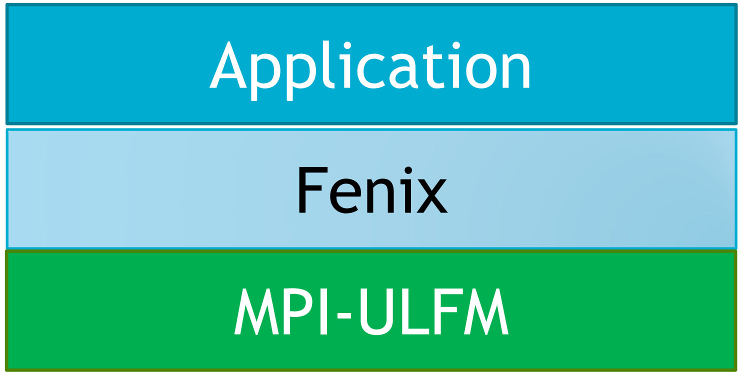
SAND2016-9171
Unlimited Release
Printed September 2016

Specification of Fenix MPI Fault Tolerance library version 1.0

Marc Gamell, Rob F. Van der Wijngaart, Keita Teranishi and Manish Parashar

Prepared by
Sandia National Laboratories
Albuquerque, New Mexico 87185 and Livermore, California 94550

- Separation between process and data recovery
  - Allows third party software for data recovery
  - Multiple Execution Models
- Process recovery
  - Extend MPI-ULFM
  - Multiple modes: **non-shrink** (hot spare process pool), shrink, spawn
  - Process failure is handled within custom MPI error handler and recovery happens automatically under the cover
- Data recovery
  - In-memory data redundancy
  - Multi-versioning (similar to GVR by U Chicago & ANL)



Fenix process recovery interface

```
void Fenix_Init (MPI_Comm comm,
                 MPI_Comm *newcomm,
                 int *role,
                 int *argc, int ***argv,
                 int num_spare_ranks,
                 int spawn,
                 MPI_Info,
                 int *error);
```

If newcomm is NULL, Fenix tacitly replaces comm everywhere with resilient communicator

App should use resilient communicator (newcomm) instead of comm.

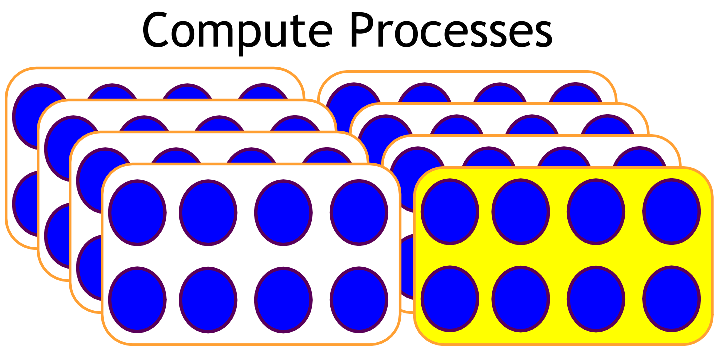
role:
- FENIX_ROLE_INITIAL_RANK
- FENIX_ROLE_RECOVERED_RANK
- FENIX_ROLE_SURVIVOR_RANK

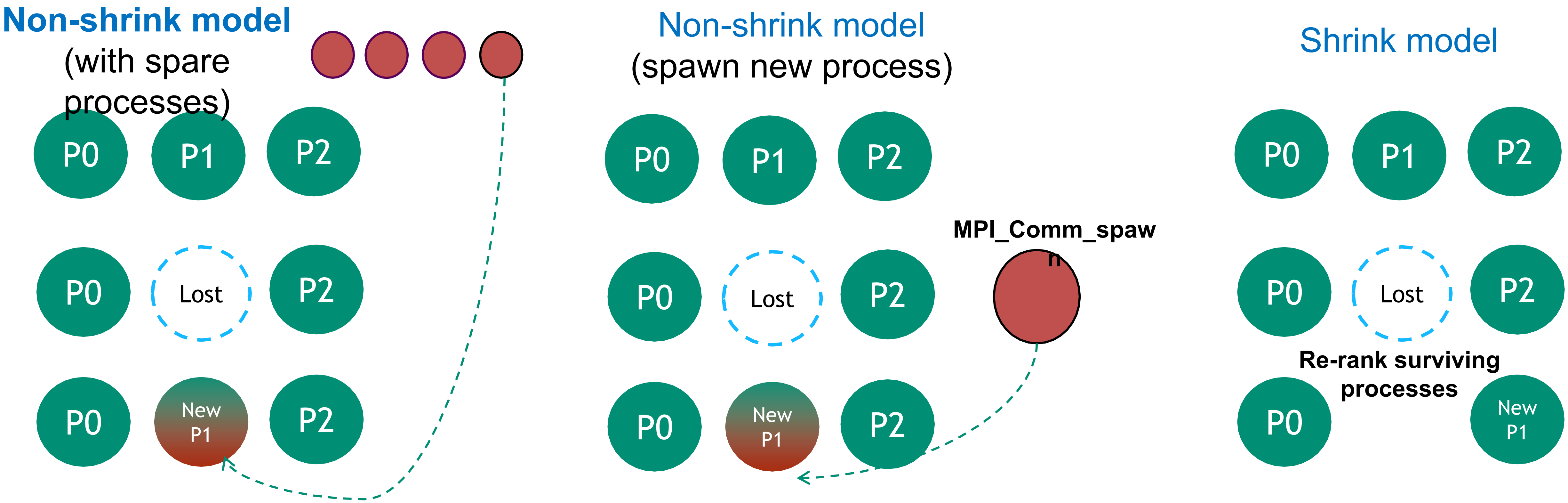
spawn:
- 0:NO_SPAWN
- 1:SPAWN

Process failure triggers process recovery and long-jump to Fenix

```
void Fenix_Finalize ( );
```

Fenix process recovery mechanics

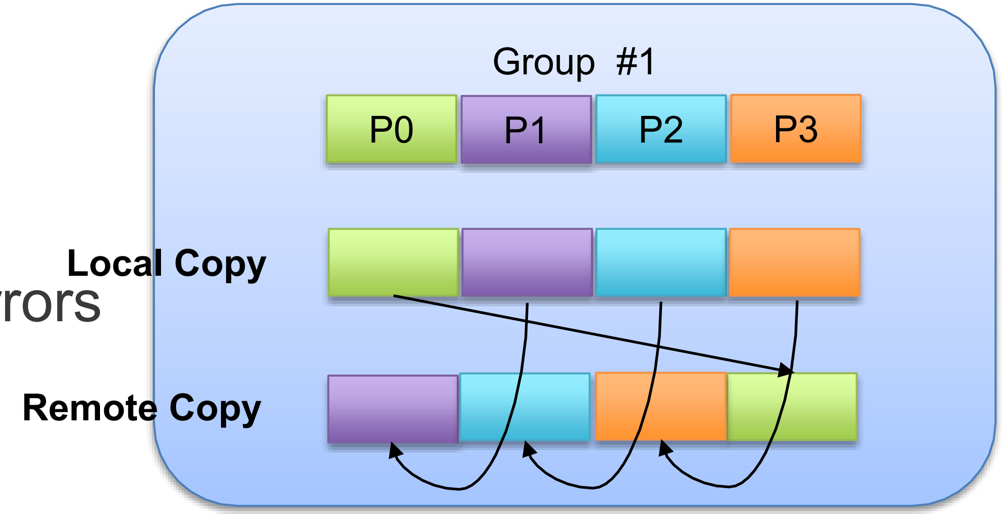




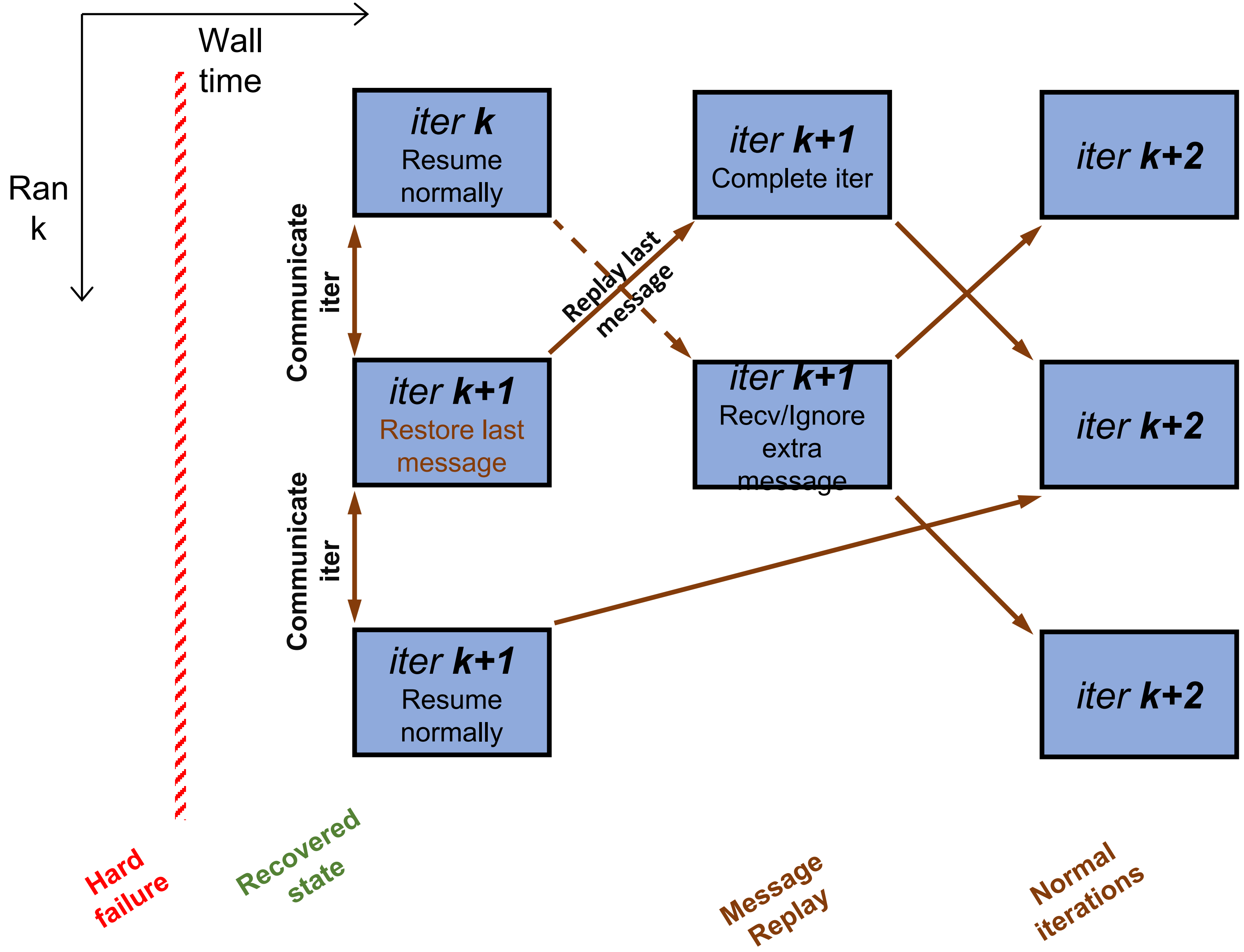
Data Recovery using Fenix

- In-memory checkpointing
  - Partner-copy redundancy
  - Fast but Memory hungry
- Versioning
  - Similar to GVR (U Chicago & ANL)
  - Failure is manifested not immediately after faults/errors
- Data store and commit
  - Data is distributed in MPI model
  - Store operation: (1) Fenix creates local copy, (2) moves the local data to remote (buddy) process location
  - Commit freezes a group of data objects in the redundant store
  - Restore operation: transfer data from the buddy copy to local buffer

`Fenix_Data_member_store()`



Stencil 1D: Preserving asynchrony



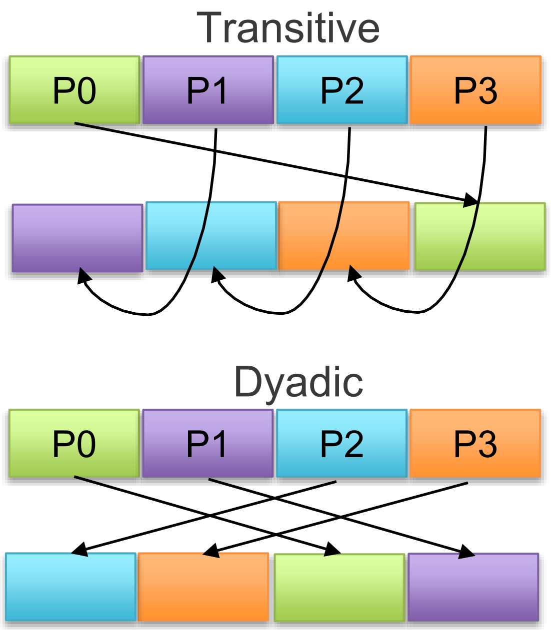
- Stencil 1D communication pattern:
  - Nearest neighbour
  - No collectives
- To preserve asynchrony at recovery:
  - Check the iteration neighbour is resuming from
  - Replay message before failure if neighbour needs it
- Message replay using Fenix:
  - Add halo data (sent messages) to checkpoint state
  - Track iterations completed and committed
  - Replay the previous message (local copy of halo), not latest, when appropriate

Stencil 1D: Pseudocode excerpt

```
Fenix_Init();

if(initial_rank)
   initialize_state();
   Fenix_Data_member_create();

if(recovered_rank)
   Fenix_Data_member_restore();

if(recovered_rank || survivor_rank)
   check_neighbour_rank_iteration();
   if(neighbour_iter < my_iter)
      if(current_iter == commit_iter)
         Fenix_Data_member_local_restore(halo_data);
      MPI_Send(halo_data);

while(current_iter < total_iters)
   communicate_halo_data();
   advance_iteration(); current_iter++;
   Fenix_Data_member_store();
   Fenix_Data_member_commit(); commit_iter++;
```

- Stencil 1D communication pattern:
  - Nearest neighbour
  - No collectives
- To preserve asynchrony at recovery:
  - Check the iteration neighbour is resuming from
  - Replay message before failure if neighbour needs it
- Message replay using Fenix:
  - Add halo data (sent messages) to checkpoint state
  - Track iterations completed and committed
  - Replay the previous message (local copy of halo), not latest, when appropriate

Fenix extensions to enable pseudo-local recovery



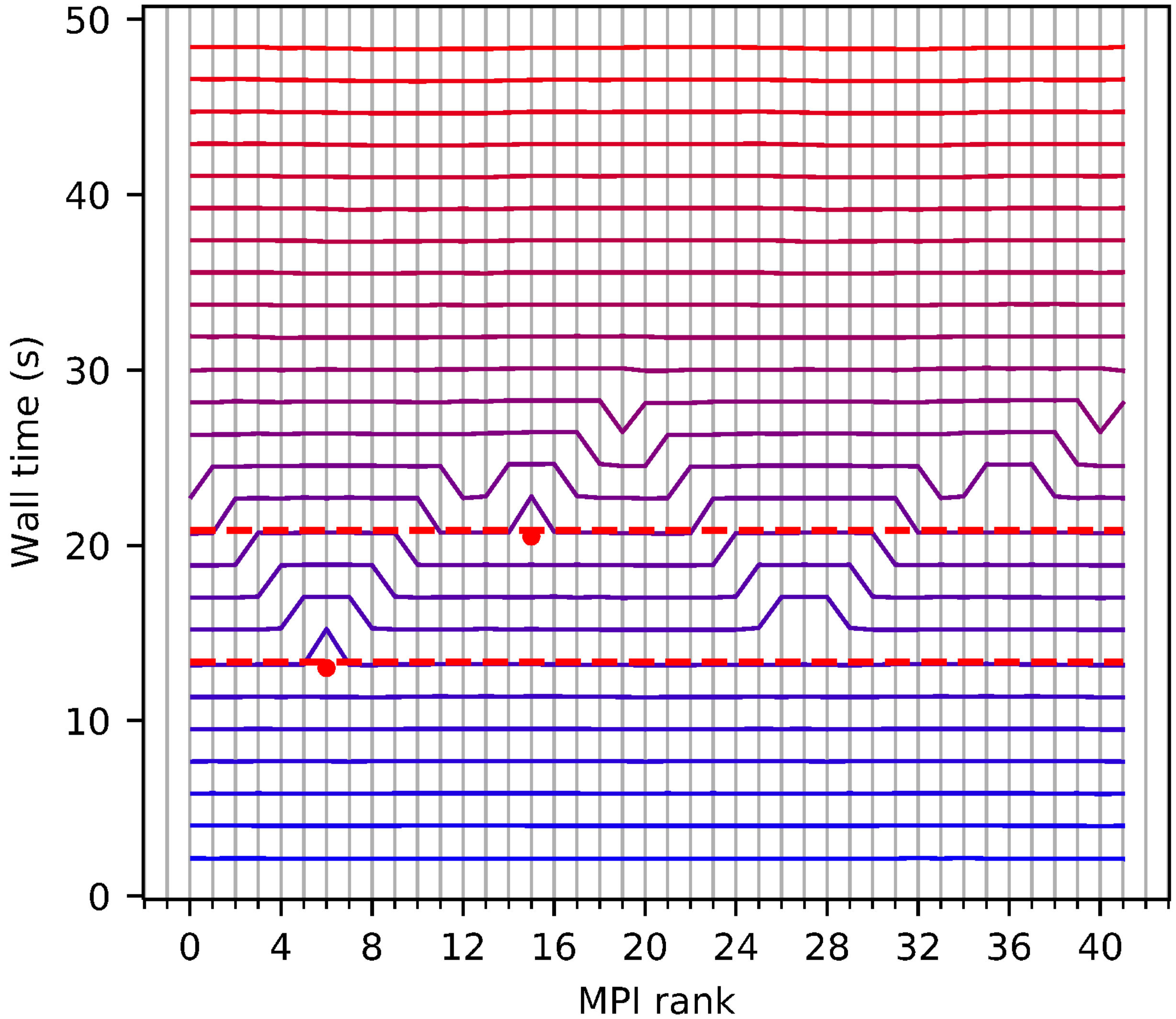
Fenix was designed originally for global rollback (owing to ULFM constraints)

Changes/extensions to Fenix to facilitate pseudo-local recovery

- Dyadic checkpoint buddies:
  - Originally, checkpoint buddies were not dyadic (I'm not my buddy's buddy)
  - Introduced transitive dependency; failure delay propagates much farther
- `Null_restore`:
  - Fenix `data_restore` is a synchronized call between checkpoint buddies; involves comm, typically all ranks call
  - For "survivor" (unaffected) ranks a checkpoint restore might undo progress; `null_restore`, a dummy operation, ensures current state is not overwritten and progress maintained
- `Local_restore`
  - Originally, `data_restore` transferred remote checkpoint copy to local checkpoint copy & application buffer
  - Some failure scenarios require replaying not the latest message, but the previous one

Results

Failure masking with pseudo-local recovery



- Experiments on a Cray XC40 platform:
  - Intel Haswell (32 cores per node)
- Stencil 1D run parameters:
  - 11 nodes, 4 MPI ranks per node (42 active ranks, 2 spare ranks)
  - 5 million grid points/rank
  - 25 iterations, 20 timesteps/iteration (500 total)
- Realistic failure injection:
  - Auxiliary error injector thread on each rank
  - Error injector thread calls `exit()`based on failure rate and time elapsed
- Checkpoint buddy is 21 ranks away
  - Recovery delay manifests at the buddy
- Solid line – iteration completion, dashed line – failure detection

Ongoing work

- Resolving issues within ULFM
  - Recently fixed issue pertaining to subsequent failure on same node
- Fenix enhancements
  - Improving efficiency (reducing cost/latency) of error detection
  - Periodic replenishment of spare ranks with repaired (previously failed) resources
  - Asynchronous error detection and communicator repair
- Applications
  - Stencil code: Run at greater scale (full machine), with a high enough failure rate that requires half ranks as spare
  - Integrating silent error mitigation with hard error recovery
  - Extending to iterative solvers (CG) that involve collectives