Toward Fine Grain Recovery in MPI-5 (aka ULFM v2)

Aurelien Bouteiller

May. 24, 2022

“Virtual MPI Forum Meeting”

Outlines

- History (aka ULFM v1)
- New Features of ULFM v2 in this reading
  - Error Range
  - Error Uniformity (implicit consistency)
- The bigger picture
  - Additional features
  - Interaction with MPI_Reinit
  - Interaction with Sessions

User Level Failure Mitigation: User Adoption

Judicael A. Zounmevo, Dries Kimpe, Robert Ross, and Ahmad Afsahi. 2013. **Using MPI in high-performance computing services.**

Fenix Framework: user-level C/R With scoped recovery

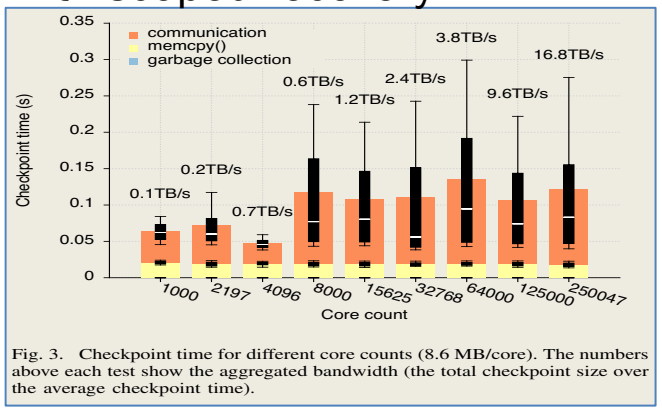


Fig. 3. Checkpoint time for different core counts (8.6 MB/core). The numbers above each test show the aggregated bandwidth (the total checkpoint size over the average checkpoint time).

SAP: Resilient Databases over MPI

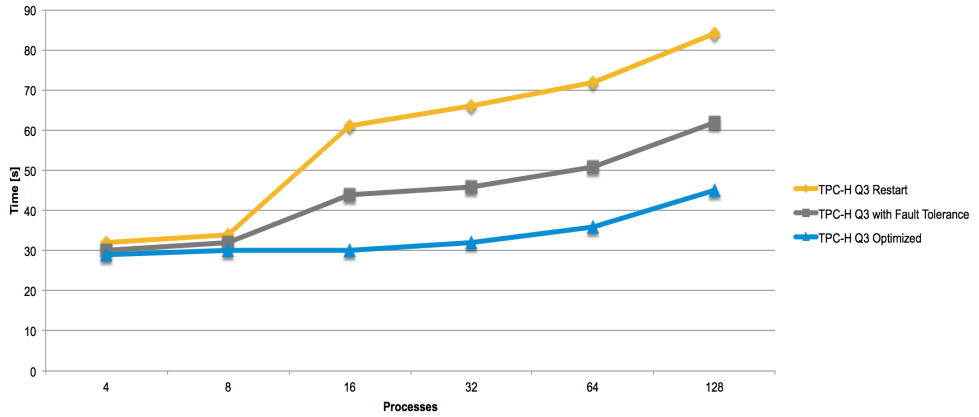


Figure 5.24: Optimization: Runtime of TPC-H Benchmark Query 3 with Failure in Phase 4 (1GB Data per Process)

Master-Thesis von Jan Stengler aus Mainz April 2017

MapReduce

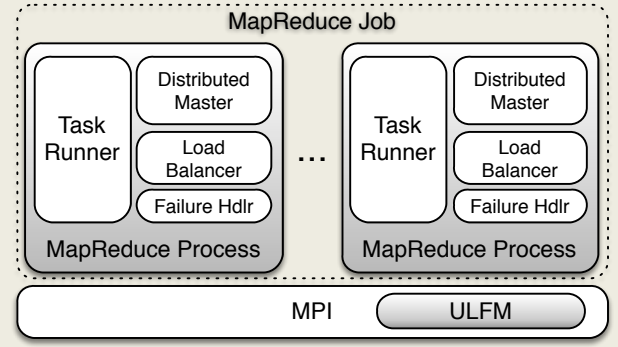


Figure 2: The architecture of FT-MRMPI.

Domain Decomposition PDE

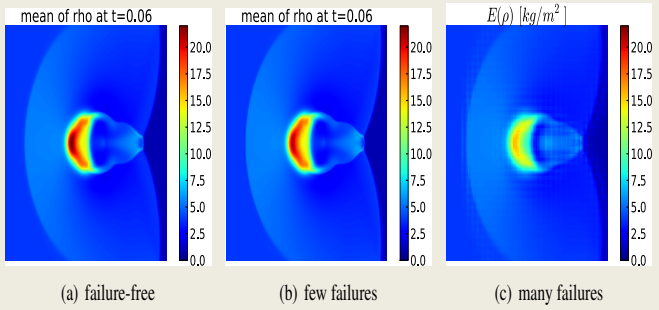
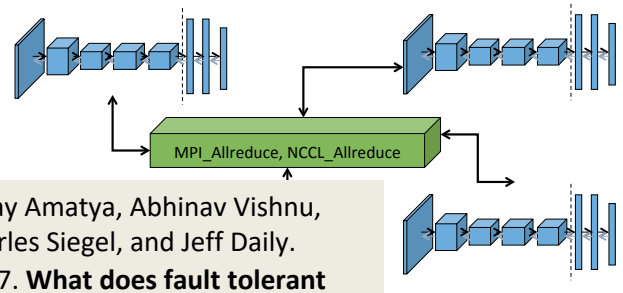


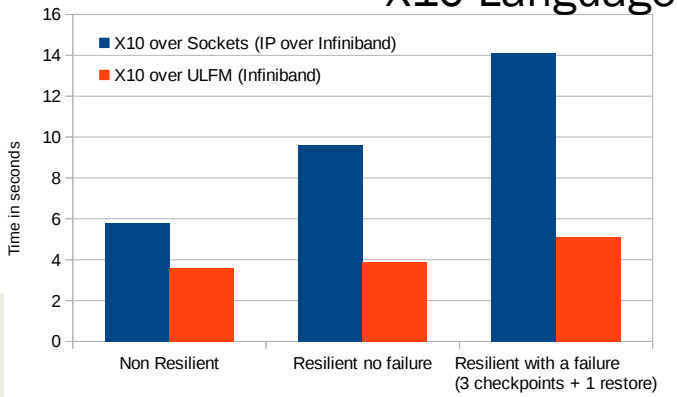
Figure 5. Results of the FT-MLMC implementation for three different failure scenarios.



Vinay Amatya, Abhinav Vishnu, Charles Siegel, and Jeff Daily. 2017. **What does fault tolerant Deep Learning need from MPI**?. In *EuroMPI, EuroMPI'17*

Source: Sara Hamouda, Benjamin Herta, Josh Milthorpe, David Grove, Olivier Tardieu. ***Resilient X10 over Fault Tolerant MPI.***

X10 Language



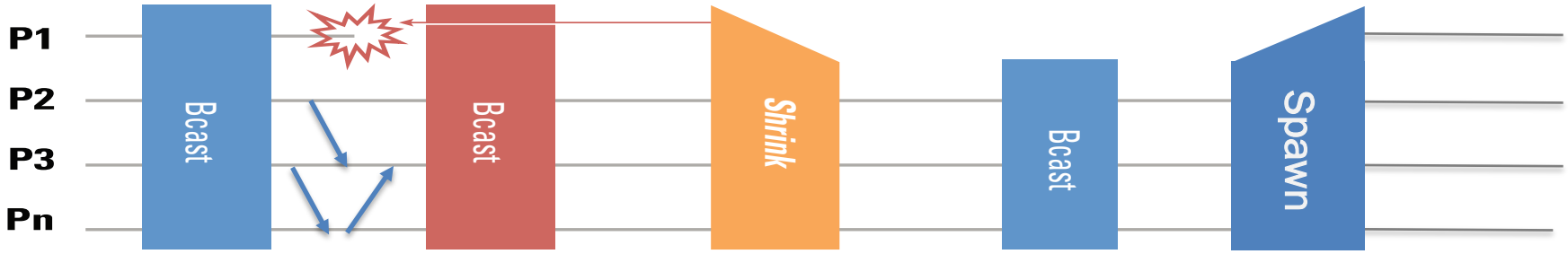
The performance improvement due to using ULFM v1.0 for running the LULESH proxy application [3] (a shock hydrodynamics stencil based simulation) running on 64 processes on 16 nodes with

And many more...

- Fortran CoArrays "failed images" uses ULFM-RMA to support Fortran TS 18508 in gcc-7.2

Stefan Pauli, Manuel Kohler, Peter Arbenz: ***A fault tolerant implementation of Multi-Level Monte Carlo methods.*** PARCO 2013: 471-480

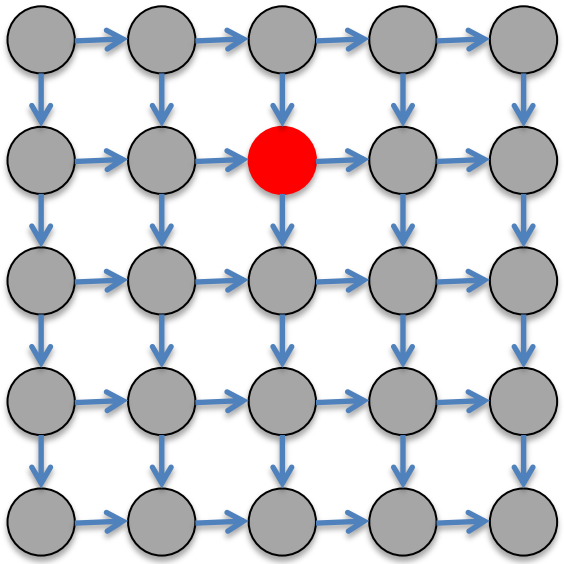
ULFM MPI Crash Recovery (Background on ULFM v1)



- 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.



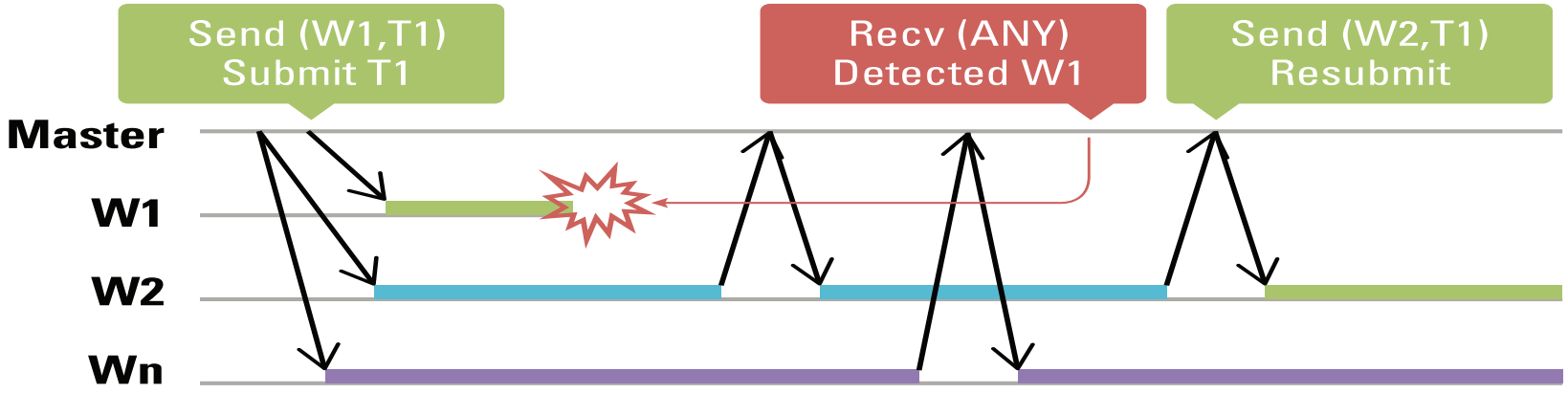
Who should be notified of a failure?
What is the scope of a failure?
What actions should be taken?

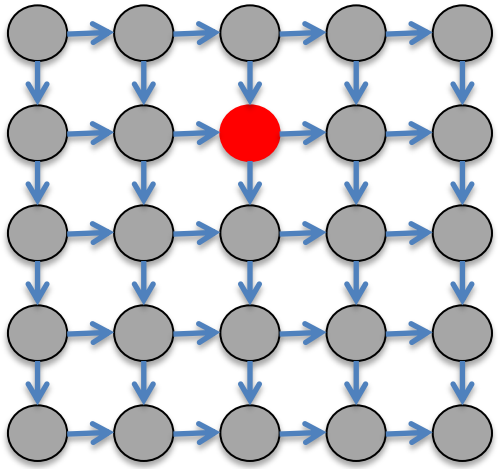
- Adds 3 error codes and 5 functions to manage process crash
  - Error codes: interrupt operations that may block due to process crash
  - MPI_COMM_FAILURE_ACK / GET_ACKED: continued operation with ANY-SOURCE RECV and observation known failures
  - MPI_COMM_REVOKE lets applications interrupt operations 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

Who should know about an error?

Keep it local model (ULFM v1)

- Error notifications do not break MPI
  - App can continue to communicate on the communicator
  - More errors may be raised if the op cannot complete (typically, most collective ops are expected to fail), but p2p between non-failed processes works
- In this Master-Worker example, we can continue w/o recovery!
  - Master sees a worker failed
  - Resubmit the lost work unit onto another worker
  - Quietly continues
- Same story with *mendable* Stencil pattern!
  - Exchange with next neighbor in the same direction instead





What if more need to know about an error?

Explicit propagation model (ULFM v1)

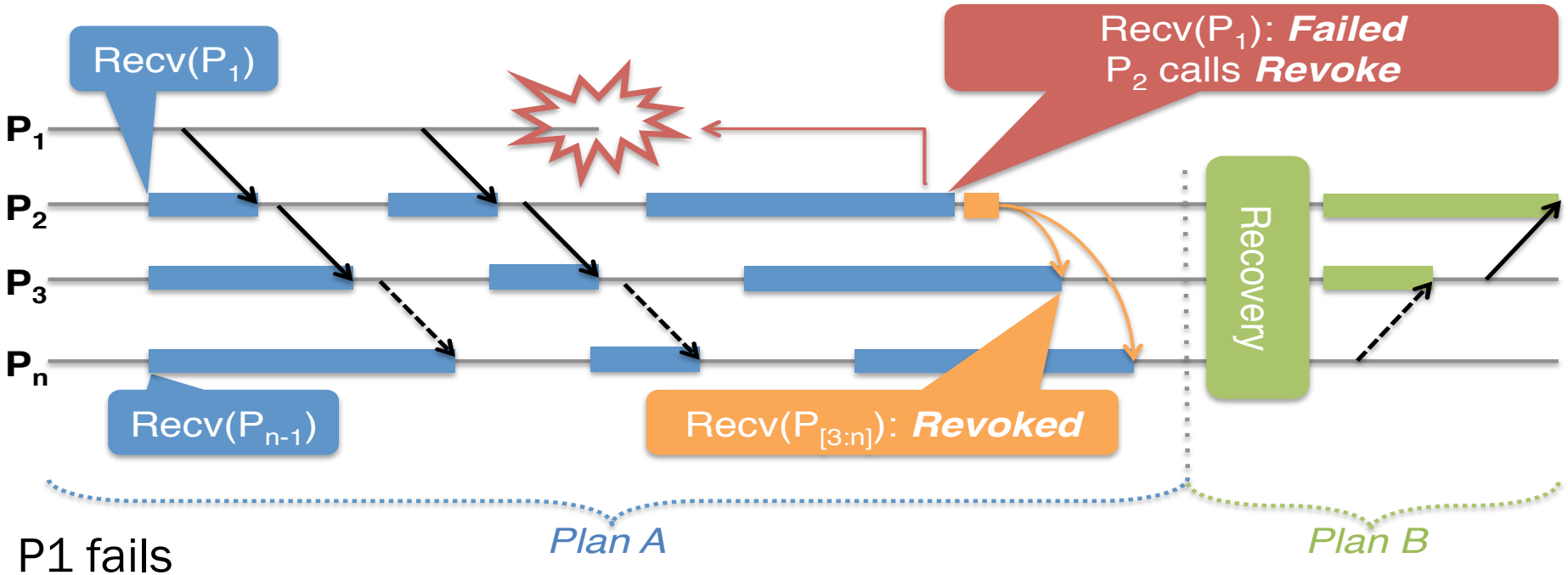
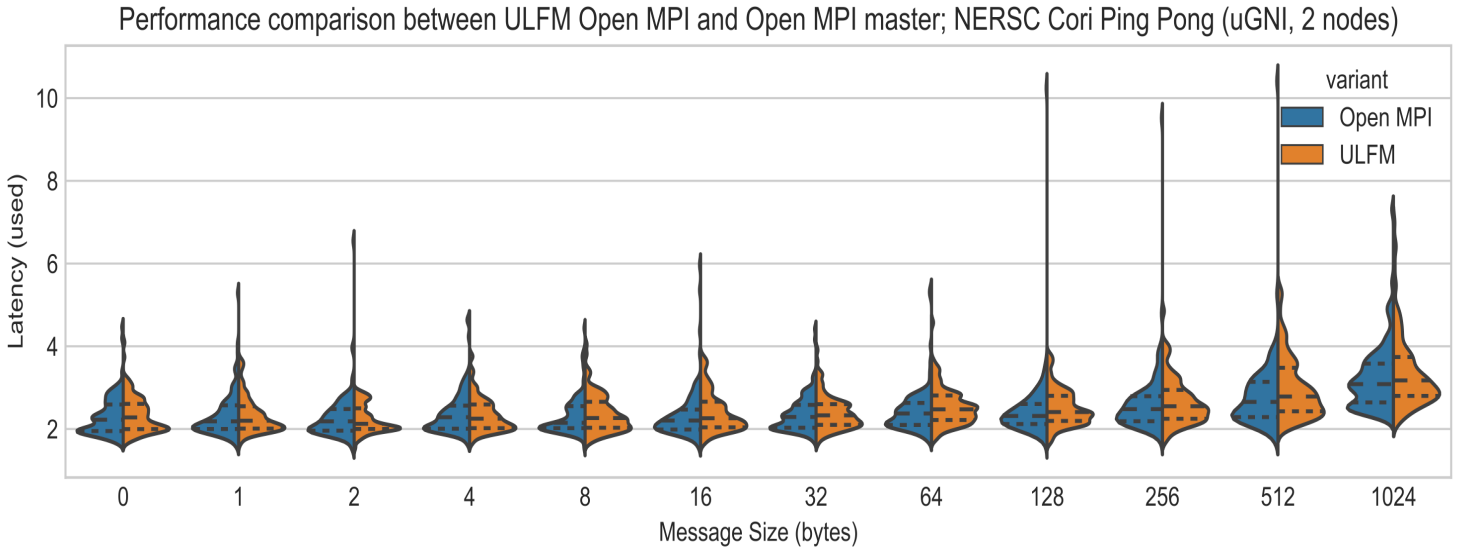


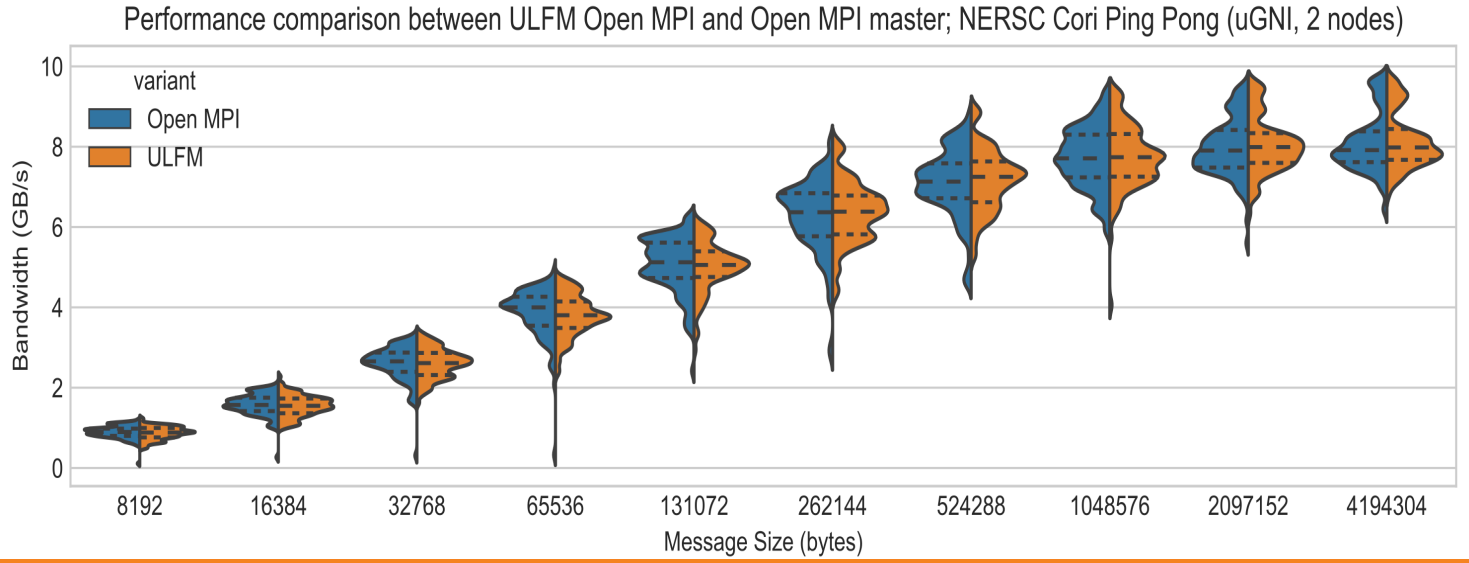
Exhibit 1: The A-Team is thinking about a Plan B

- P1 fails
- P2 raises an error and stop *Plan A* to enter application recovery *Plan B*
- but P3..Pn are stuck in their posted recv
- P2 can unlock them with Revoke ☺
- P3..Pn join P2 in the recovery

ULFM available in Open MPI 5.0, MPICH

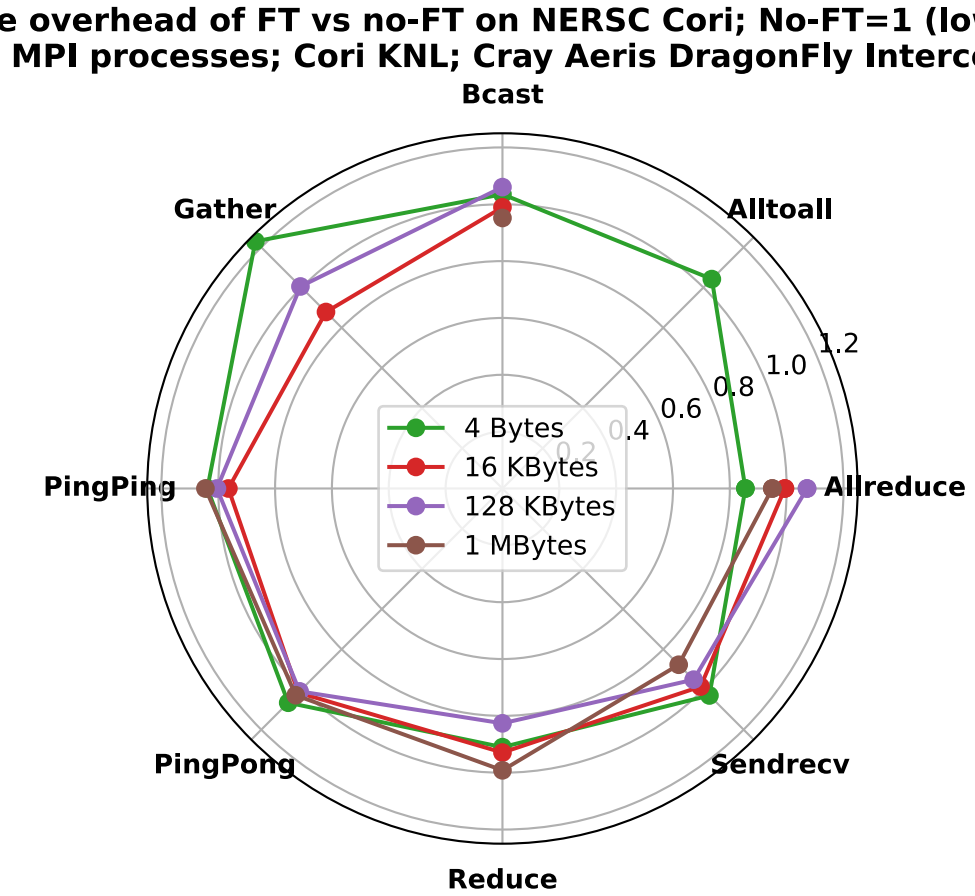
- ULFM (User Level Failure Mitigation) is a standardization proposal that enables the execution of Fault Tolerant Applications over MPI
- Applications can use error handlers to react and repair the effect of failures
- ULFM is now directly available in Open MPI 5.0
- *mpiexec –with-ft=mpi*





ULFM available in Open MPI 5.0

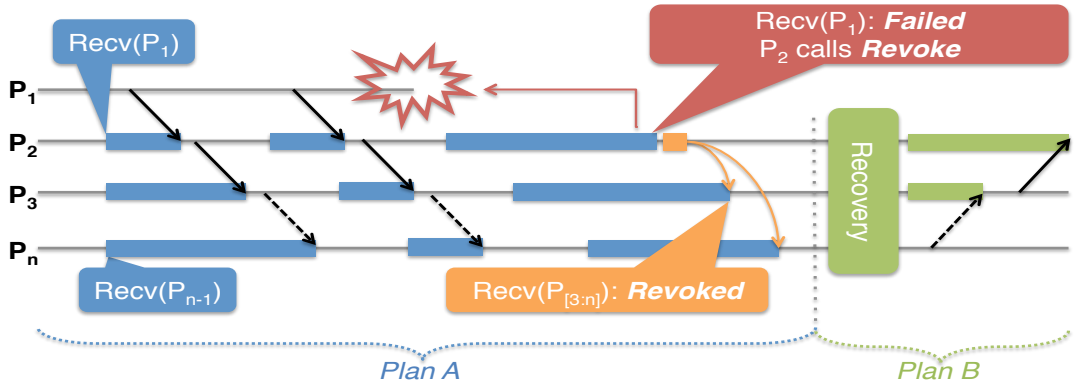
- ULFM (User Level Failure Mitigation) is a standardization proposal that enables the execution of Fault Tolerant Applications over MPI
- Applications can use error handlers to react and repair the effect of failures
- ULFM is now directly available in Open MPI 5.0
- *mpiexec –with-ft=mpi*



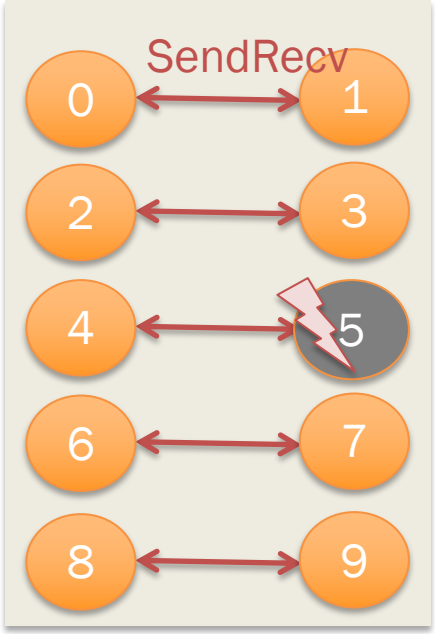
Outlines

- History (aka ULFM v1)
- New Features of ULFM v2 in this reading
  - Error Range
  - Error Uniformity (implicit consistency)
- The bigger picture
  - Additional features
  - Interaction with MPI_Reinit
  - Interaction with Sessions

ULFM Evolutions:
Control Error Detection Range (aka auto-revoke)



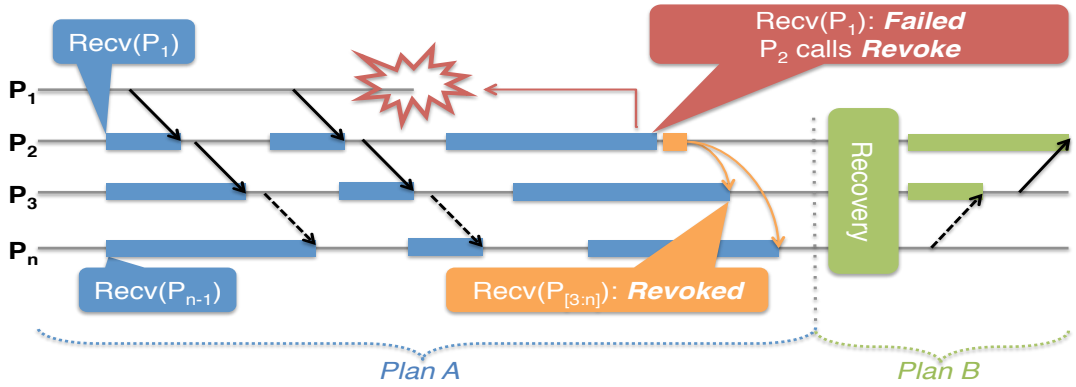
MPIX_COMM_REVOKE can be called to explicitely propagate an error to related procs; In some cases (e.g., SPMD), implicit propagation is desirable



- By default, errors only in communication dependent upon the failed process (ULFM v1)
  - Default range is "involved" processes (w.r.t. communication pattern)
  - Idea: additional modes for (unconditional) scoped and non-scoped recovery
- MPI_Info object set on the communicator/win/file
  - mpi_error_range=local: current ULFM behavior (default/unset) (e.g. only in recv from failed process)
  - mpi_error_range=group: report errors (i.e. REVOKE) for a failure at any process with a rank in the comm/win/file (e.g. in recv from an alive process in comm)
  - mpi_error_range=universe: report errors (i.e. REVOKE) for a failure anywhere in "universe"
- In *group* and *universe* modes, ERR_REVOKED is produced

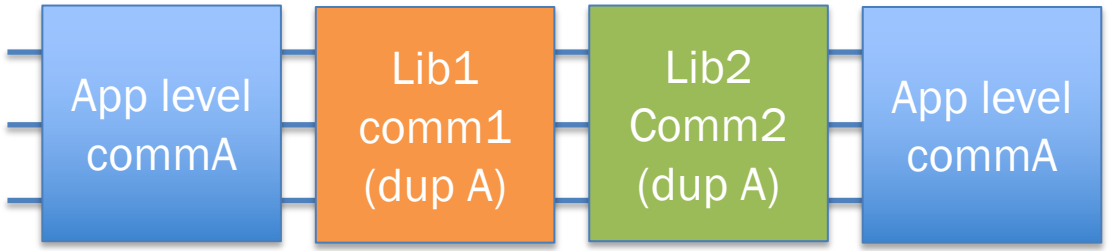
example:
- Local mode: only rank 4 should report the failure of rank 5
- Group mode: all ranks in comm will report the failure of rank 5

Error Detection Range Benefits: Enabling Modular Coordination of Recovery Events



- MPI Communicators define a communication group
- Explicit propagation scoped to the communicator
- What if a library module needs to trigger a global recovery procedure? (and the overspan communicator handle is not visible from the current code scope)?
- Libraries can select to observe failures at all ranks, and thus coordinate to trigger a recovery event.

Temporal SPMD library composition



Domain SPMD decomposition

Nesting (w/o overlap)

Example Using Ranges

Group Scope Example

```
int odd= rank%2;
MPI_Comm_split(MPI_COMM_WORLD, odd, rank, &comm);
MPI_Info_create(&info);
MPI_Info_set(info, “mpi_error_range”, “group”);
MPI_Comm_set_info(comm, info);
MPI_Comm_set_errhandler(comm, &errh);

/* tokens ring circulates left to right in rank order */
MPI_Sendrecv(..., right, ..., left, ..., comm,...);
```

Errhandler triggerred for any error at any rank in comm (i.e., if any ‘odd’ process fails, operations on the ’odd’ communicator raise error, but no on the ‘even’ communicator, and vice-versa).

Global Scope example

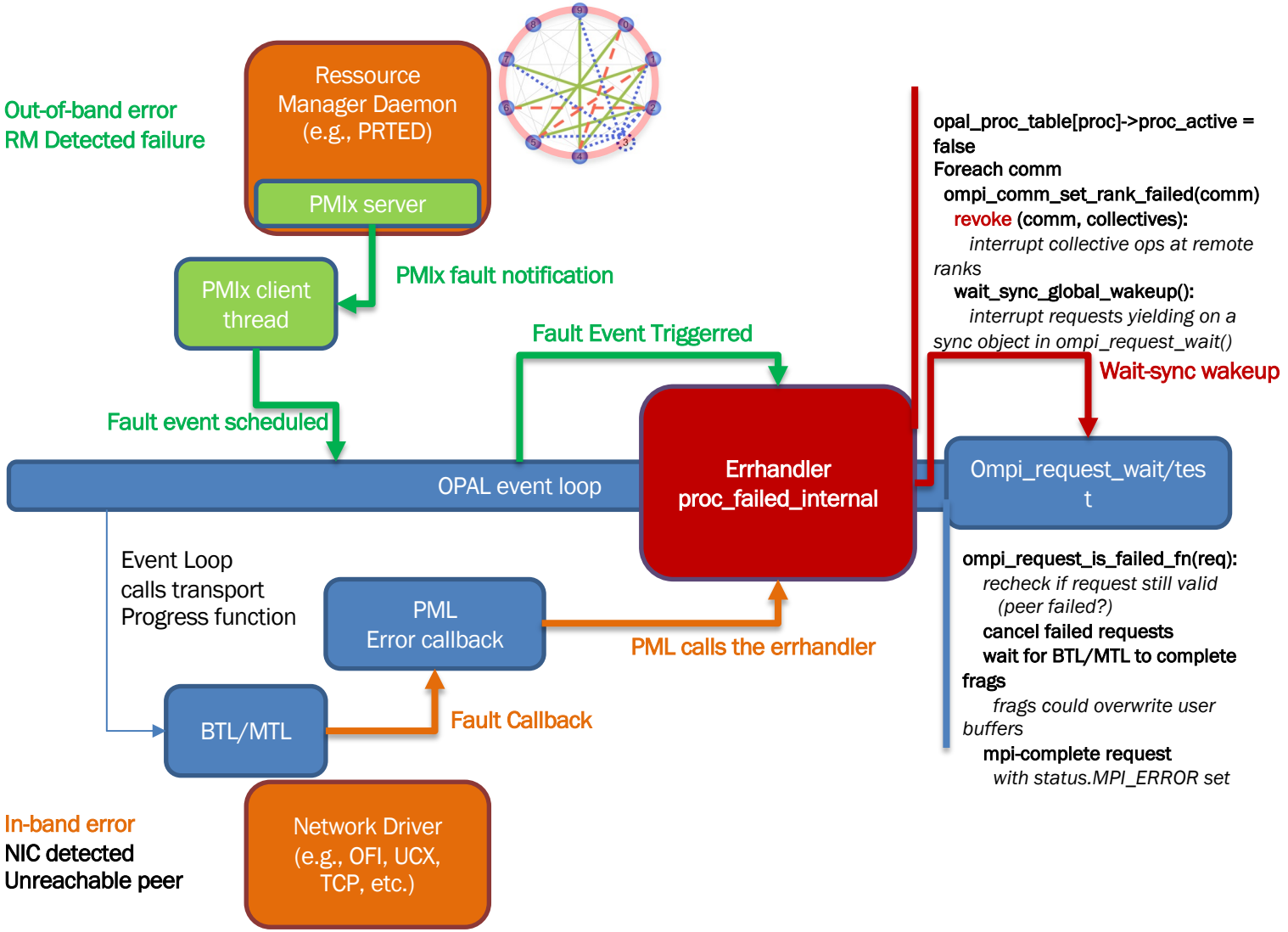
```
int odd= rank%2;
MPI_Comm_split(MPI_COMM_WORLD, odd, rank, &comm);
MPI_Info_create(&info);
if(odd) MPI_Info_set(info, “mpi_error_range”, “group”);
else MPI_Info_set(info, “mpi_error_range”, “universe”);
MPI_Comm_set_info(comm, info);
MPI_Comm_set_errhandler(comm, &errh);

/* tokens ring circulates left to right in rank order */
MPI_Sendrecv(..., right, ..., left, ..., comm,...);
```

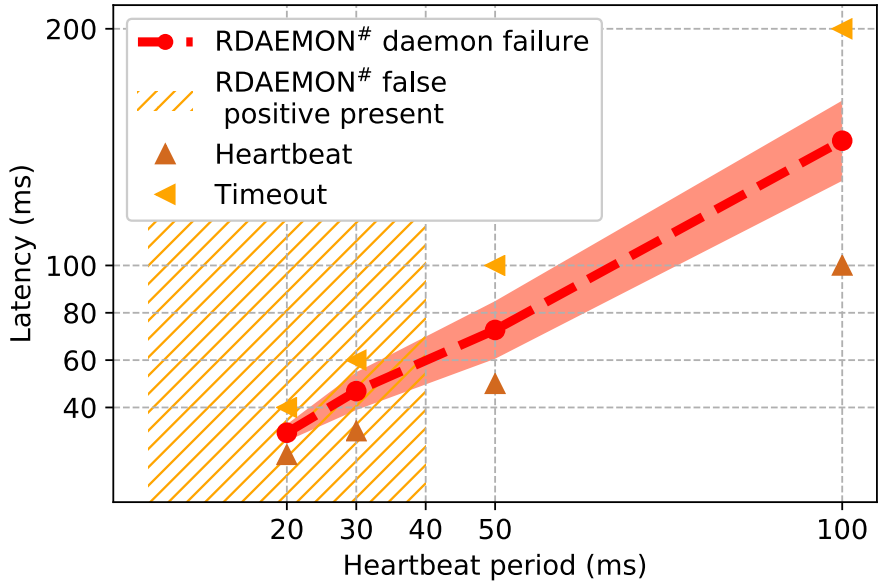
*On ‘Odd’ processes, failures at other ‘Odd’ processes raises an error when using comm. On ‘even’ processes, failure at **any** process raises an error when using comm.*

Consequences for failure detection

- ”local” scope does not mandate out-of-band failure detection
  - In-band (i.e., errors from the network driver) sufficient
- “group” and “universe” scope require at a minimum out-of-band propagation and often require active monitoring



Cost of performing detection and implicit propagation in the runtime



- Accuracy of detection is very good (in the order of 100ms can be achieved in practice at scale)
- False detection rate **independent of the application communication pattern**
  - *Prior MPI-based detector would produce false positive when application does not call MPI procedures*
- *Reusable in different programming models*

Performance variability in **GRAPH500 with an active PMIx-PRRTE Failure detector**
Left: **MPI** Right: **OpenSHMEM**

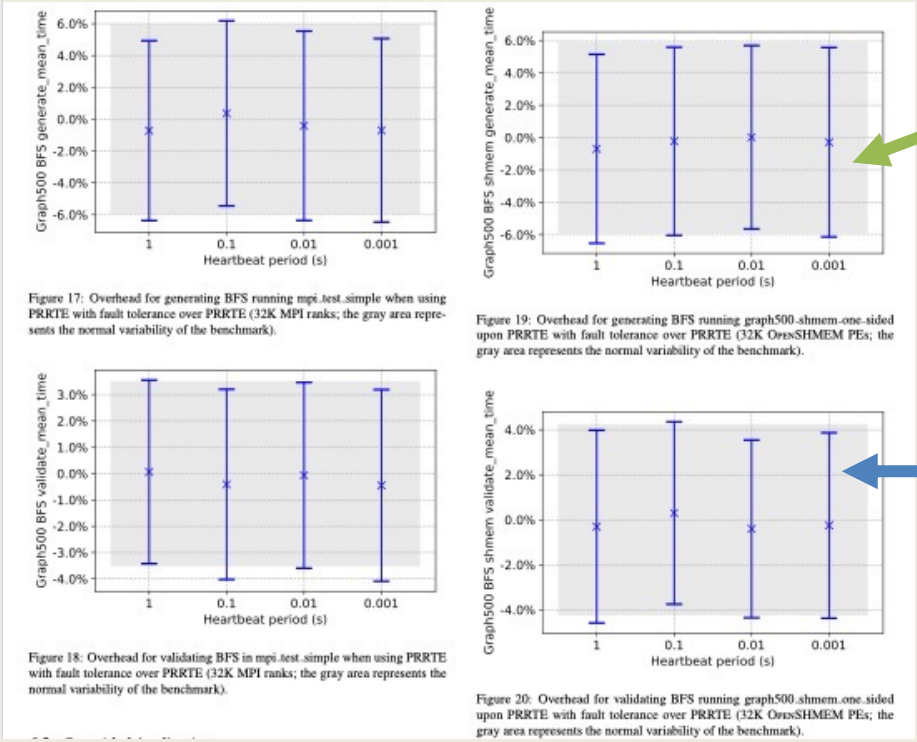


Figure 17: Overhead for generating BFS running mpi_test_simple when using PRRTE with fault tolerance over PRRTE (32K MPI ranks; the gray area represents the normal variability of the benchmark).

Figure 19: Overhead for generating BFS running graph500_shmem_one_sided upon PRRTE with fault tolerance over PRRTE (32K OpenSHMEM PEs; the gray area represents the normal variability of the benchmark).

Figure 18: Overhead for validating BFS in mpi_test_simple when using PRRTE with fault tolerance over PRRTE (32K MPI ranks; the gray area represents the normal variability of the benchmark).

Figure 20: Overhead for validating BFS running graph500_shmem_one_sided upon PRRTE with fault tolerance over PRRTE (32K OpenSHMEM PEs; the gray area represents the normal variability of the benchmark).

Gray area represents normal benchmark variability

Blue error bars show the variability as measured with detection ON

Experiments performed on NERSC's Cori: Cray XC40 supercomputer with Intel Xeon "Haswell" processors and the Cray "Aries" high speed inter-node network, 32 cores per node, 32K processes total.

Outlines

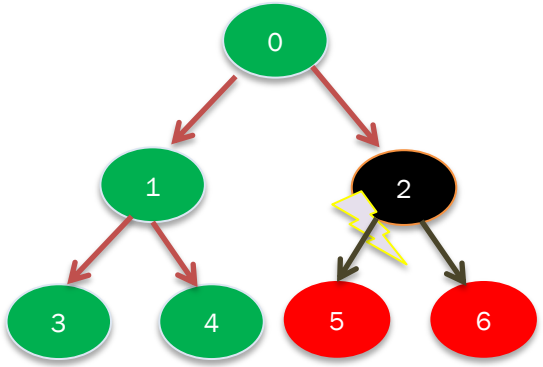
- History (aka ULFM v1)
- New Features of ULFM v2 in this reading
  - Error Range
  - Error Uniformity (implicit consistency)
- The bigger picture
  - Additional features
  - Interaction with MPI_Reinit
  - Interaction with Sessions

Error Uniformity in counting, bulk synchronous programs

```
counting_collectives(void) {
    for(iteration=0; iteration<target; iteration++) {
        compute(iteration);
        rc = MPI_Allreduce(buff, count, datatype, 0, comm);
        MPI_Comm_agree(comm, &rc);
        if(rc != MPI_SUCCESS) {
            recovery(iteration);
        }
    }
}
```

Above code snippet solves the issue, but…

- Must be inserted after every collective operation
- Lost capability of using error handlers



Uniformity example: an error is reported only at some leaf node in a broadcast topology with a failure

Lax consistency: Exceptions are raised only at ranks where the Allreduce couldn't succeed

- In a tree-based Allreduce, only the subtree under the failed process sees the failure
- Other ranks succeed and proceed to the next iteration
- Revoke solves potential deadlocks, but…
- Ranks that couldn't complete enter "*recovery*" with a different iteration counter!
- Ranks that could complete the allreduce **altered the memory** performing an extra *compute(iteration)*

Example Using Uniform

Non-uniform example

```
MPI_Comm_dup(MPI_COMM_WORLD, &comm);
MPI_Comm_set_errhandler(comm, MPI_ERRORS_RETURN);

While(i++ < niter) {
  compute_step1(i);
  rc = MPI_Scatter(..., comm);
  flag = (MPI_SUCCESS == rc);
  MPI_Comm_agree(comm, &flag);
  if (!flag) {
    errhandling(i);
  }
 compute_step2(i);
  rc = MPI_Bcast(..., comm);
  flag = (MPI_SUCCESS == rc);
  MPI_Comm_agree(comm, &flag);
  if (!flag) {
    errhandling(i);
  }
  ...
}
```

Uniform example

```
MPI_Comm_dup(MPI_COMM_WORLD, &comm);
MPI_Info_create(&info);
MPI_Info_set(info, "mpi_error_uniform", "coll");
MPI_Comm_set_info(comm, info);
MPI_Comm_set_errhandler(comm, &errh);

while(i++ < niter) {
    compute_step1(i);
    MPI_Scatter(..., comm);
    compute_step2(i);
    MPI_Bcast(..., comm);
    compute_step3(i);
    MPI_Allreduce(..., comm);
}
```

Solution: Error Uniformity Controls

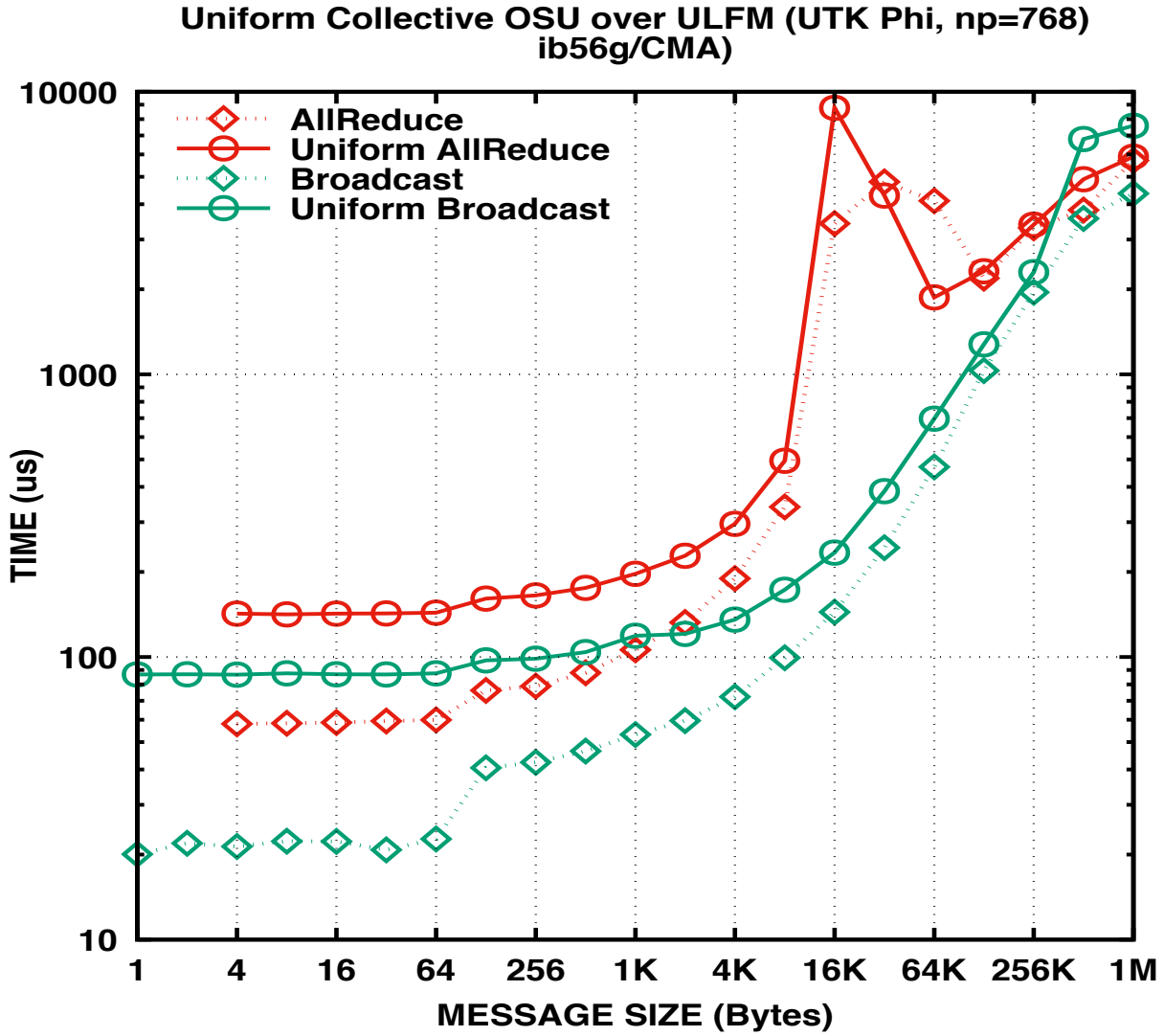
- Idea: control error uniformity (with communicator Info keys again)

- mpi_error_uniform=local: errors reported as needed to inform of invalid outputs (buffers/comms) at the reporting rank (i.e. other ranks may report success); **default, current ULFM**
- mpi_error_uniform=construct: if communicator/win/file creation operations (e.g. comm_split, file_open, win_create, comm_spawn,…) reports at a rank, it has reported the same ERR_PROC_FAILED/REVOKED at all ranks.
- mpi_error_uniform=coll: same as above, for all collectives (including creates)

Error Uniformity: performance impact

- Latency for small messages is greatly impacted
- Bcast: 20->90us
- Allreduce: 60->140us

- Cost amortized on large message bandwidth



Outlines

- History (aka ULFM v1)
- New Features of ULFM v2 in this reading
  - Error Range
  - Error Uniformity (implicit consistency)
- The bigger picture
  - AGREE/IAGREE (future reading)
  - SHRINK/ISHRINK and non-blocking recovery (future reading)
  - RMA/Files (future reading)
  - Introspection/control, e.g. MPI_FT attribute, mpiexec params (future reading)
  - Error Synchrony (exploratory)
  - Interaction with MPI_Reinit
    - Settled: coexist in text, coexist in impl., flip-flop between models (time decomposition)
    - Exploratory: both models active at the same time (nested, or rank-domain decomposition)
  - Interaction with Sessions (exploratory)

ICL THE UNIVERSITY OF TENNESSEE KNOXVILLE

Non-blocking recovery: ISHRINK (not part of reading today)

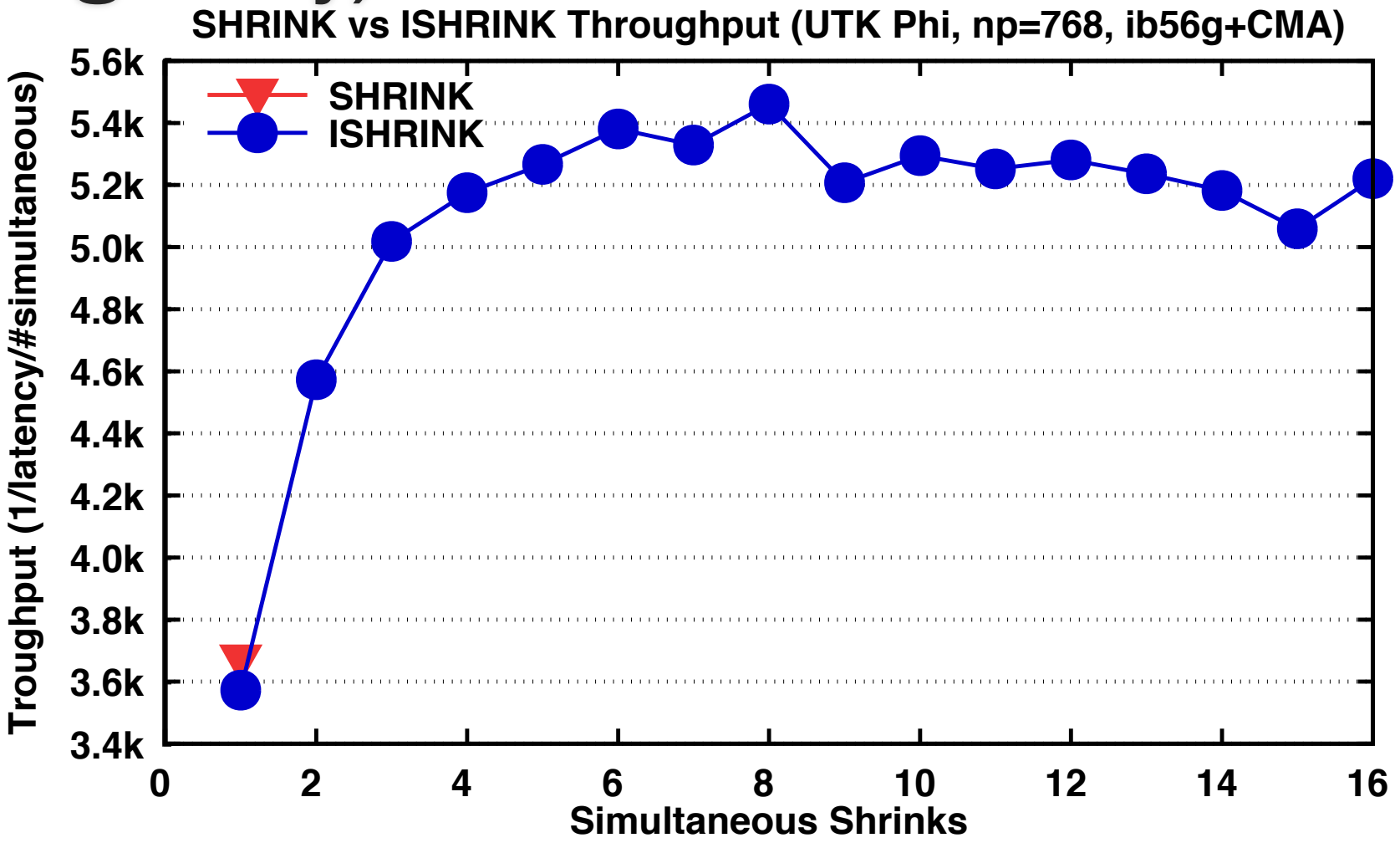
- Performance advantage: overlap shrink (e.g. with I/O to reload a checkpoint)
- In non-blocking libraries, when the error handler cannot block
- When recovering multiple overlapping comms, relaxed shrink ordering can be required

- MPIX_COMM_ISHRINK(comm, ncomm, req)
- Same as SHRINK, but non-blocking
- Resolves most ordering problems
  - Post order does not matter (by definition all SHRINK in different comms)
  - Completion order does not matter (as soon as all ishrink posted, they all have to progress regardless of wait ordering)

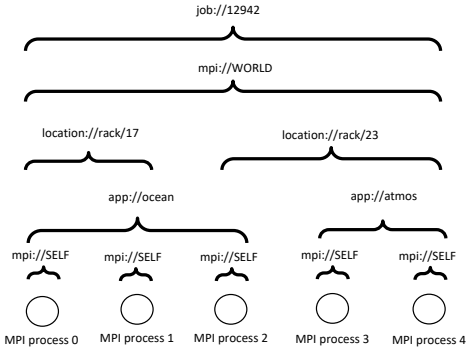
ICL THE UNIVERSITY OF TENNESSEE KNOXVILLE

Performance: ISHRINK w/overlap (not part of reading today)

- ISHRINK latency similar to SHRINK
- Simultaneous ISHRINK overlap



Sessions and FT (exploratory)



- MPIX_SESSION_REVOKE(session)
  - Same as COMM_REVOKE, but triggers on all communicators derived from the session
  - Can be used as a handle to stop a library (e.g., fault detected at the application level, but a non-blocking library has active communication)
- We still want to have MPIX_COMM_SHRINK
  - 1 stop shop: 1. eliminates dead processes, 2. concensus, 3. create new comm (cid)
  - We still want to have COMM_SHRINK (for communicator-centric recovery models)
  - Needed for MPI-3 style apps (i.e., no sessions)
- How could one use MPI_COMM_CREATE_FROM_GROUP to handle shrink/replace recovery modes?
  - We also want to investigate how COMM_CREATE_FROM_GROUP and GROUP_FROM_SESSION_PSET can be used for recovery
  - Concensus-like meaning for COMM_CREATE_FROM_GROUP?
- Variadic psets?
  - In MPI-4 process sets are static
  - Ideas around relaxing and versioning
  - mpi://world:3 would obtain the 'third' world (with group membership agreed upon by the runtime with an implicit concensus)
  - Other idea is to have explicit resource allocation calls on the session
  - Call can be local, however we should have a way to test for progress/completion without blocking
  - One can then obtain the group mpi://10th-spawn-from-rank10, do MPI_GROUP_UNION, and thus create a mended "world"

```
MPI_Info_create(&sinfo);
MPI_Info_set(sinfo, "mpi_thread_level_support", "MPI_THREAD_MULTIPLE");
rc = MPI_Session_init(sinfo, MPI_ERRORS_RETURN, &lib_shandle);
if (rc != MPI_SUCCESS) goto error;

/*create a group from the WORLD process set */
rc = MPI_Group_from_session_pset(lib_shandle, "mpi://WORLD", &wgroup);
if (rc != MPI_SUCCESS) goto error;

/* get a communicator */
rc = MPI_Comm_create_from_group(wgroup,
        "org.mpi-forum.mpi-v4_0.example-ex10_8",
        MPI_INFO_NULL, MPI_ERRORS_RETURN, &lib_comm);
```

Concluding Remarks

- ULFM v1 goals:
  - flexible approach to recovering MPI communication capability (repair what you need)
  - Communicator centric approach

- ULFM v2 Added goals:
  - Permit easier expression of recovery codes (implicit actions)
  - Automate tedious/repetitive code (implicit actions)
  - Permit modularization of recovery procedures (non-blocking actions, controllable error reporting scope)
  - Non blocking recovery: recovery of state and data can overlap
  - Compatibility between non-global and global recovery (e.g., compatible with reinit)

  - Add Session/Group centric approach

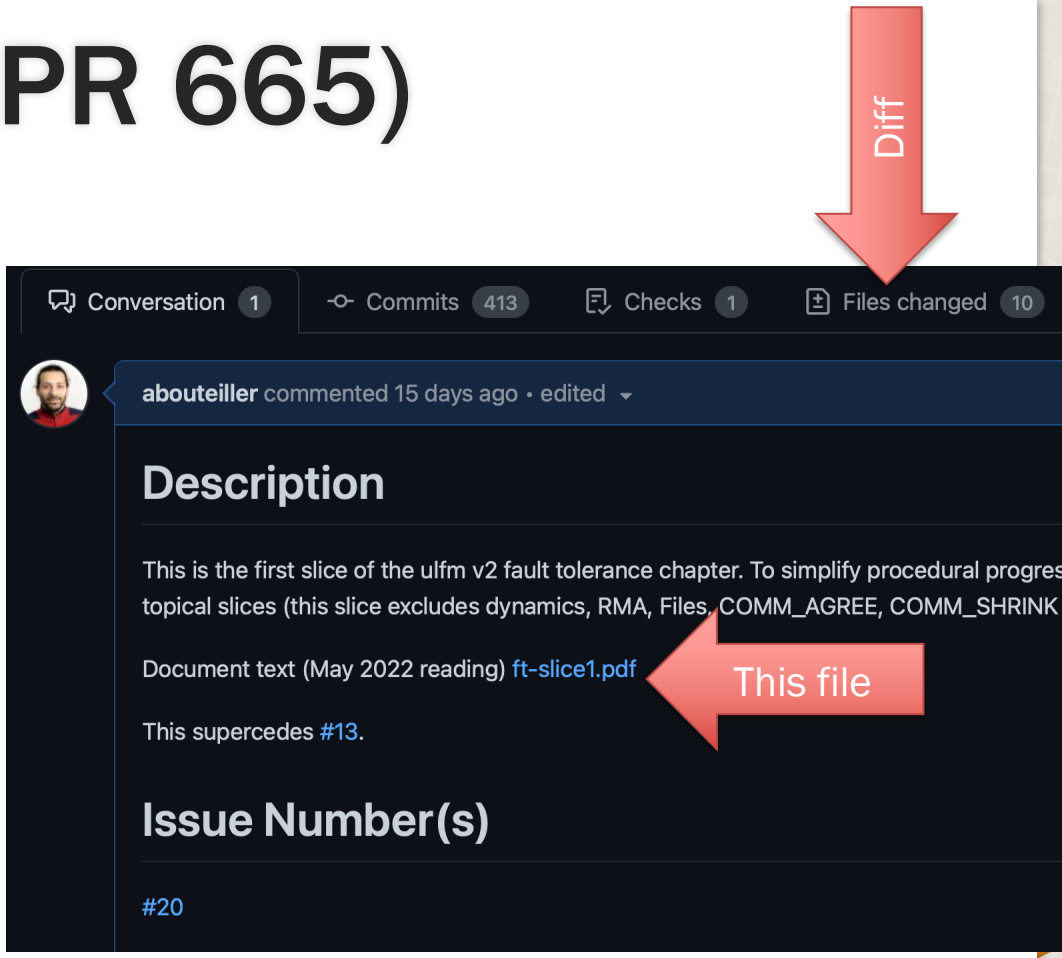
- Fully implemented, represents state of the art in the literature with large body of work using ULFM v1 in varied contexts (programming language extensions, C/R frameworks, Stencil, PDE, ABFT, etc.)

ICL THE UNIVERSITY OF TENNESSEE KNOXVILLE

What we are reading today (PR 665)

PR #665 https://github.com/mpi-forum/mpi-standard/pull/665/files

The core of ULFM v2: error reporting modes and controls, communication flow interruption



Small additions in the works (not for reading today)

- Advice about “other” fault types: https://github.com/mpiwg-ft/mpi-standard/pull/17
- Should MPI_Irecv raise FT errors?: https://github.com/mpiwg-ft/mpi-standard/pull/18/files
- Should we use an Attribute to query if FT is runtime active? https://github.com/mpiwg-ft/mpi-standard/pull/19/files#diff-1487c38b2632cb01aeb3d10f9dc182c364bf6699a4f35f431e1db59e52d4f2bc

The bigger picture: the full ULFM v2 proposal (not for reading today): RMA, Files, AGREE, SHRINK, MPI_FT attribute, etc
https://github.com/mpiwg-ft/mpi-standard/pull/19

ANY Source matching, or why PROC_FAILED_PENDING?

- Mix of NAMED and ANY_SOURCE matching in receiver queue
- If we PROC_FAILED the iANY, the matching the matching order is changed
- Thus we need to maintain the matching queue in order => (PF_PENDING)

Senders: P1: S1t1, S2
P2: S1t2, S2t2
P3: dead

P4: Receiver Queue (posted/pending receives)

Case1: iANY cause PROC_FAILED

iR1(ANY, t=2), iR2(ANY,t=1), iR3(2, t=2), iR4(ANY)

ERR_PROC_FAILED: completes in error

~~iR1(ANY, t=2), iR2(ANY,t=1),~~ iR3(2, t=2~~), iR4(ANY)~~

User repost: messages now in disorder/matching incorrect

iR3(2, t=2), *iR1(ANY, t=2), iR2(ANY,t=1), iR4(ANY)*

Case2: iANY cause PROC_FAILED_PENDING

iR1(ANY, t=2), iR2(ANY,t=1), iR3(2, t=2), iR4(ANY)

ERR_PROC_FAILED_PENDING: procedure returns, request still pending

iR1(ANY, t=2), iR2(ANY,t=1), iR3(2, t=2), iR4(ANY)

Calling completion on ANY returns from procedure with error
Matching still active and in order meanwhile
User Calls ACK_FAILED (acking P3) to stop procedures returning with PFP

iR1(ANY, t=2), iR2(ANY,t=1), iR3(2, t=2), iR4(ANY)